A TREATISE ON SYSTEMS, VOLUME 2

This second volume describes the systematic application of Promise Theory to systems, representing a significant step forward in describing functional systems with both dynamics and semantics. By combining quantitative and qualitative descriptions in a single framework, Promise Theory provides the first impartial language for multiscale system phenomena. The closer one is the intrinsic scale of a system component, the more its qualitative semantics dominate its behaviour. As one scales out to large numbers, dynamical patterns dominate the average behaviours.

This book serves as guide to graduate students and researchers in the development of a new science of systems, and contains an outstanding number of examples and lessons accumulated over ten years of working in the field, unified by a single theoretical Promise Theoretic model.

'...a landmark book in the field of network and system administration. For the first time, in one place, one can study the components of network and system administration as an evolving and emerging discipline, rather than as a set of recipes, practices, or principles.'
– Alva Couch, Tufts University 2002, from the Foreword of volume 1

'Mark Burgess' new book brings an analytical, scientific approach to bear on the general subject of systems and network administration. This original perspective opens up a wealth of ideas and possibilities which will be of interest to both the researcher and advanced practitioner'
– Paul Anderson, Edinburgh University 2002

'An unusual book ... in that it describes the theory which relates the components — computers and networks to the users and administrators. It is the only book I know that covers the 'science' underpinning systems administration.'
– Morris Sloman, Imperial College London, 2002

Also by the author:

PROMISE THEORY: PRINCIPLES AND APPLICATIONS

MONEY, OWNERSHIP, AND AGENCY—AS AN APPLICATION OF PROMISE THEORY

SMART SPACETIME: HOW INFORMATION CHALLENGES OUR IDEAS ABOUT SPACE, TIME, AND PROCESS

Other reviews:

'A landmark book in the development of our craft...'
–Adrian Cockcroft (about In Search of Certainty)

'Proud to say that I am a card-carrying member of the [Mark Burgess] fan club. And I think that it wouldn't be too much of a stretch to say that he's the closest thing to Richard Feynman within our industry (and not just because of his IQ).'
–Cameron Haight (about Smart Spacetime)

'...our whole industry is transforming based on ideas [Mark Burgess] pioneered'
–Michael Nygard (about Smart Spacetime)

'The work done by [Mark] on complexity of systems is a cornerstone in design of large scale distributed systems...'
–Jan Wiersma (about In Search of Certainty)

'Some authors tread well worn paths in comfortable realms. Mark not only blazes new trails, but does so in undiscovered countries.'
–Dan Klein (about Smart Spacetime)

A TREATISE ON SYSTEMS

THE SCALING OF INTENTIONAL SYSTEMS
WITH FAULTS, ERRORS, AND FLAWS

VOLUME 2

MARK BURGESS

χtAxis press

First published by χtAxis press 2020.
This printing with current state of corrections on September 28, 2020.

Disclaimer: Although he always tries his best, the author of this work may only be marginally pleased with its successes, while more deeply ashamed of its shortcomings. After all, he promises only best-effort in formalizing the large mass of disparate experiences and topics contained herein. The material, claiming complete innocence in the matter, reserves the right to contain naivités, errors, on the author's behalf (or bewhole), even cases where the author decides to change his capricious mind (he can be a bit like that from time to time). Comments may be welcomed, if reasonably polite, and readers should absolutely look for free updates as corrections and minor improvements are made to this edition online.

Cover design by Zhaoling Xu.

CONTENTS

PREFACE

This second volume of *Treatise on Systems* attempts to go beyond the introductory and background material in the first volume, to assemble a set of definitions, concepts, and actionable techniques for a scale dependent analysis of cooperative functional systems. This treatment is based on the language and methods of Promise Theory. It incorporates and surpasses the classical statistical analyses of component reliability to deal with issues of context dependence and non-trivial functional semantics in human-machine hybrid systems. The active agents in a system (humans, machines, etc) are generic; they are assessed only by their accuracy or *fidelity* in keeping to their promised roles. Adaptation and evolution of systems fall outside the scope of this introduction, except in the most primitive contextual forms.

It's exceedingly rare for a book to attempt such an ambitious scope, yet I felt compelled to take on the challenge, albeit on a necessarily superficial level. It feels, bluntly, as though the field of systems has been stunted between mysticism and denial for decades. This work could perhaps be considered a generalization of the early works on cybernetics, taking them beyond the continuum methods available in the post-war years, but I believe it also goes further to address modern issues of complexity and uncertainty. I'm afraid the book is not for the faint hearted, as it asks a level of rigour that doesn't usually find a willing audience in Information Technology, and a level of heuristics and approximation that is eschewed by mathematicians. My goal here is to put an umbrella over many disciplines that labour in isolation, and show that their travails are—in fact—not as unique as they often assume. We find a few simple principles at work, illuminating qualitative and quantitative issues. My hope is that eventually, some years in the future, the field of Information Technology science and engineering will reach a level of maturity where these notes will help to convene a more willing audience, ready to take them further. The unfamiliarity of the language of Promise Theory will mean that the book will not be immediately accessible to readers without some effort today. Nevertheless, those who persevere will be rewarded with an astonishingly simple set of ideas, leading to helpful insights in terms of a very simple language. There

are universal principles of causality and scale that unify all functional systems across domains, no matter whether their agents are human, machine, or plant, and these are exposed in a simple way through Promise Theory language. If I succeed in making this case alone—inspiring readers to strive for an analytical approach to the subject—then the effort will surely not have been in vain.

With such a huge amount of material to cover, the nagging issue of how to organize it for browsing and for dipping into as a reference work shadows the whole endeavour. This volume is significantly more technical than the first volume, and makes extensive use of the Promise Theory, which was developed after the completion of the original work (now volume 1). I've kept the text as concise as possible, without too much reference or discussion. I hope that this will aid contemplation rather than killing it in its tracks. Given the magnitude of covering every topic, I had to content myself with sketching out ways that others might use to follow up on these notes, with more detailed and rigorous analyses, rather than asymptotically being complete. Let it be a lexicon of methodology rather than an oracle on every issue.

Finally, I'm painfully aware of the difficulty in bridging the cultural and knowledge gap involved in understanding and applying the principles espoused in these volumes. By now, I've tried to describe the origins of the principles in my popular books *In Search of Certainty* and *Smart Spacetime*—and I refer readers to those books to get a general sense of what's going on. Trying to weave that narrative throughout the text here would make it too cumbersome, so (again) I'll leave it to others to use this terse account as the basis for shorter pieces with a special focus[1].

From time to time, sage commentators are hailed for recognizing that one cannot design a system without considering its dynamics and semantics together, e.g. it has been common to design software architectures without sufficient attention to their runtime environments. In my case, I made the opposite error of omission. The first volume in this series focused on system dynamics; this second volume repairs the holistic view to further incorporate the constraints implied by semantics. The complementary error of trying to deal only with semantics failed in Logic and is now being repeated in the guise of Category Theory[2], whose arcane machinations are compounded by their paradoxical popularity. I believe that Promise Theory's simple-minded pragmatism already surpasses their raw legacies, taking the best from them, as it focuses on expressive outcomes without the endless mind-bending abstractions and demands of exclusivity. As a basis for qualitatively and quantitatively assessing suitably idealized approximations and calibrated measurements, I'm not altogether unhappy with the result.

M.B. Oslo, 19th February 2020

CHAPTER 1

PRINCIPLES AND ASSUMPTIONS

> "As robotic system developers strive to achieve a level of autonomy, they underestimate the need for coordination with human stakeholders." [DH06]

> "The most powerful dehumanizing machine is not technology but the social machine, i.e. The formation of command structures to make humans emulate technology in order to build pyramids and skyscrapers..."
>
> –Lewis Mumford (1967)

The level of detail and intrinsic complexity needed to describe human-machine system processes can be truly breathtaking, so we seek suitably idealized approximations as models. It's important to have a framework in which to formulate a model, with clear definitions and principles to reduce unnecessary arguments about terminology. In this chapter, I sketch out what I've come to believe are those relevant principles.

Promise Theory has proven to be a convenient abstraction for describing many of the relationships concerning the arrangements of parts, as well as for capturing the information flows that drive behaviour. The raw promise formalism is described in [BB14b]. We refer readers to that reference for basics and for details.

1.1 PROMISE THEORY

Let's make a brief summary of Promise Theory, without repeating the full text of [BB14b]. Promise Theory is about what can happen in a collection of components, called *agents*, that work together[Bur05b, MK05]. It can be applied to any kind of agent, whether machine or human. Engineers should resist the temptation to think of it as a network

protocol or sequence of commands to program into a computer; rather it's a descriptive algebra from which one might design a variety of protocols for humans and machine parts to keep the promises described. In a sense, it's a level higher than an algorithm or a protocol. Computer programmers should not look for 'code' in this book—our discussion here about patterns of interaction, architecture, and dimensioning. Rather than 'just show me the code', we counter with 'just explain the ideas' to keep the analysis timeless and independent of readers' favourite technologies.

In Promise Theory, one begins with the idea of completely autonomous agents that interact through the promises they make to one another. This view is well-suited to modelling networks, as in [MB04]. Although we cannot force autonomous agents to work together, we can observe when there are sufficient promises made to conclude that they are indeed cooperating voluntarily. Our challenge in this volume, is to translate this bottom-up view into top-down, human managed requirements.

Agent is the term used for the fundamental actors or entities that can make and keep promises in Promise Theory. Agents are not necessarily human or machine, nor like the more limited concept of 'software agents': they can be any active entities like an interface that keeps promises—a human or a machine. More important than the specific make-up of an agent is the processes it enacts. Actions taken by agents are not in the scope of Promise Theory, so we'll describe them as an additional layer involved in the realization of promise-keeping. In other words, we assume that appropriate actions are taken to keep the promises. In that way, we focus on declarative intent, rather than imperative procedures.

1.2 INTENTIONAL SYSTEMS

Promise theory deals with the concept of intent, and its semantics, along side a description of the purely mechanical or dynamical aspects of behaviour. It is useful to define intent itself, and indeed its relation to promises. A unit of intent is called an intention. The body of each intention describes what is intended, i.e. what constraint is desired or expected of outcomes for an agent.

Definition 1 (Intent)**.** *A subject or type of possible behaviour. i.e. something that can be interpreted to have significance. Any agent can harbour intentions. It could be something like 'be red' for a light, or 'win the race' for a sports person.*

In philosophy, there is a long history of identifying intent with human ideas about consciousness and freewill. Thinkers have found it hard to dissociate their ideas about intent and thought from the Victorian religious doctrine of Mankind being at the top of a

ladder of species, with unique capabilities that place humans next to God, and place us above all other animals and things. Searle was prominent in criticizing this view[Sea83].

From the definition of intent above, it should be clear that an assessment of intentional behaviour lies principally in the eyes of the beholder. I accept and adopt here a broader view of intentionality than most authors—one that can be applied to humans, animals, and any kind of machinery, i.e. a definition that scales from small to large.

Definition 2 (Intentional system)**.** *A system that appears to satisfy a stable functional role within a larger composition of agents.*

This ties intent the semantics of behaviours. An intentional agent may be thought of as having an interpretation, and even a purpose, which means that it expresses behaviour whose semantics are more or less constant over an extended observational timescale. The agent expresses stability in the eyes of an observer.

The point about intent is not at all that there has to be freewill behind it, because many inert tools carry out and represent the intentions of their users, e.g. a computer program is a proxy for the intentions of its designer and its user. The origin of intent is not as relevant as the observation of a sense of purpose intent, and that is ultimately a speculative assessment made by an observer, as we observe in our natural propensity for anthropomorphism in everyday speech. Indeed, the fact that inert machinery can exhibit intentional behaviours naturally leads to the question: are we, as humans, so sure that what we imagine to be intentional, conscious, and free thinking behaviour really is what we think it is? That question is the essence of the Turing test. I'm not going to offer any further arguments on this point. Henceforth, I assume intent to be a universal property about constant behavioural promises.

1.3 IMPARTIALITY

By tradition, the manifesto of science is to seek 'impartiality'[3]. This is one reason why semantics have traditionally been eschewed from modern science—we believe that all unnecessary interpretation will lead to bias. The problem with this argument is that it is simplistic. It's analogous to saying that we can avoid bias in judging the colour of a flower by closing our eyes. If we remove all labels for discrimination, what we end up with will be necessarily trivial.

Ultimately that manifesto fails to capture the fullness of functional and contextual systems. In more modern times, the rise of statistical knowledge has seen an overstated belief in the truth of 'data'. Statements like 'data don't lie' are common. This, however, is quite misleading—and Promise Theory exposes the reason quickly and impartially

(through its simple axioms)! Briefly, data are unobservable without a process of sampling, measurement, contextualization, and interpretation. The receiver, observer, or measurement agent is an intermediary in a chain of evidence. So—whether data lie or don't (and clearly data sources can deliberately deceive in certain circumstances)—it is always the integrity of the observer which is the weak link in the chain.

As humans, we base knowledge on trust relationships. Accepting data from unverified sources (e.g. lumps of big data, large data sets, etc) without seeing the values arise one by one, over a period of time, bypasses Axelrod's game of trust, in which the payoff for deception is a lowering of trust and the payoff for confirmation is an increase. When data arrive 'all at once', we trust them no more than a single data point. So statistical answers to questions can never be trusted, regardless of how we assess their data sources—because they eliminate the very contextual and semantic basis on which trust grows. Data gathered with the aim of impartiality, and with the best of intentions, can be both misunderstood, out of context, muddled inappropriately and averaged into a formless nothingness. In other words, whether data lie or not is not the right question to ask; we must ask: have we captured the contextual semantics of the phenomena we study? This has the deepest of implications for both data science and the construction of models on all levels. Therefore, to study systems, in a meaningful way, we must reexamine the role of the observer in each case.

1.4 PROMISES AS EXPRESSIONS OF INTENT

Our starting point for systems is to clarify their intentions and their behaviours. This would be impossible without clear statements to define them, and their properties. In other words: what do they promise? What can we rely on to be true about them? What can we trust?

A *promise* is an intention that has been 'voluntarily' adopted by an agent (usually channeling a human owner, or perhaps an agreed standardization). An agent that only promises to do as it's told is *dependent* or voluntarily subordinated. It has some of the characteristics of a service: an agent makes its intended behavior known to other agents (e.g. I will serve files on demand, or forward packets when I receive them). An imposition is an attempt to induce the cooperation of another agent by imposing upon it (e.g. give me the file, take this packet).

Definition 3 (Promise)**.** *When an intention is publicly declared to some audience (called its scope) it becomes a promise. Thus a promise is a stated intention. A promise from* Promiser *to* Promisee*, with body b is written:*

$$\text{Promiser} \xrightarrow{b} \text{Promisee}.$$

Promises could be considered a collection of ad hoc *choices* or *policies* that stand as the outcomes of decisions about an agent's willing limits. Promises are entirely voluntary decisions, they cannot be imposed in general.

Definition 4 (Imposition)**.** *This is an attempt to induce cooperation in another agent, i.e. to implant an intention. It is complementary to the idea of a promise. Degrees of imposition include: hints, advice, suggestions, requests, commands, etc. Impositions are written:*

$$\text{Imposer} \xrightarrow{\ b\ } \blacksquare\ \text{Imposee}.$$

It is far more common to find the use of obligations as the model of system control (in the sense of deontic logic); for example, a system *shall* do this or that, a system *must* do the other—a system is obliged to behave properly. This stems from a long history of belief in determinism and perhaps slavery—the desire to impose one's will on the world. It ties into the command style interfaces to computers and machines, which are designed to fit our manual approach to interacting with the world by touch. The problem with this kind of thinking is that it does not scale well, because it connects a single remote source with a world of unspecified size. Wrestling manually with systems might be satisfying on the scale of what a single human can manage, but it quickly becomes a David and Goliath scenario (in either direction) when the scales are mismatched.

Promises cleanly express the limits of determinism behind a simple abstraction. They should not be assumed to originate from a human being—though many inanimate agents are proxy for the keeping promises originating from humans

Definition 5 (Obligation)**.** *An imposition that implies a cost or penalty for non-compliance. It is more 'aggressive' than a mere imposition.*

Once a promise has been made, and an attempt to keep it, any agent can attempt to assess whether it believes the promise to be kept or not. This is a fundamentally subjective matter, even in physics where one would like to believe in an objective reality. Reality may be objective, but our assessments of it are not.

Definition 6 (Assessment)**.** *A decision by a single agent A about whether a promise π has been kept or not, usually written $\alpha_A(\pi)$.*

Every agent makes its own assessment about promises it is aware of. Often assessment involves the observation of other agents' behaviours. Note that it is meaningless to compare assessments by different agents. This would be like comparing measurements in inches with measurement in centimetres. If agents can be calibrated in advance by

promising to accept the criteria of a single source—an arbiter of truth—then assessments could be compared conditionally, since calibration is a conditional promise.

Promises and impositions fall into two polarities, denoted by $\pm$. A promise to give or provide a behaviour b is denoted by a body $+b$; a promise to accept something is denoted $-b$ (or sometimes $U(b)$, meaning use-b). Similarly, an imposition on an agent to give something would have body $+b$, while an imposition to accept something has a body $-b$.

Although promises are not a network protocol, agents can exchange data. To complete any kind of exchange, we need a match an imposition (+) with a promise to use (-). To form a binding (as part of a contract), we need to match a promise to give (+) with a promise to use (-). This rule forces one to document necessary and sufficient conditions for cooperative behaviour.

A promise model thus consists of a graph of nodes (representing *agents*), and links or edges (representing either *promises* or *impositions*), which are used to describe intentions. Whatever protocol might be used to communicate promises is not defined, and is separate from whatever protocol is designed to keep the promises. Agents publish their intentions and other agents may or may not choose to pay attention. In that sense, these interactions form a binding chemistry of intent [Bur13b], with no particular manifesto, other than to decompose systems into the set of necessary and sufficient promises to model intended behavior.

The basics of Promise Theory are summarized follow essentially from the principle of a priori autonomy of parts.

Principle 1 (Autonomy)**.** *No agent can make or keep a promise on behalf of another agent.*

1. All states, actions and behaviours in a system are properties of agents.
2. Without intent, an agent's behaviour is said to be *ad hoc* or unconstrained.
3. Without coordination intent, a collection of agents cannot act in an intentional way, and its collective behaviour must be considered *autonomous*.
4. Agents are a priori atomic, i.e. independent or autonomous[4]. They may promise to give up some or all of their independence by receiving input from others.
5. Agents make promises, and assess the promises made by other agents.
6. Intentions depend on *context*, i.e. the states reported by other agents within a collaborative system. Thus agents may be able to assess context and promise to share that assessment transparently as a service to other collaborating agents.
7. A documented intention is called a promise.

- It is made *by* the agent that has the intention and can keep the promise.
- It is made *to* any stakeholders with an active interest in the information, and may be seen by others too with only passive interest.
- No agent can make a promise on behalf of another agent to keep (the latter would be called an *imposition*). Indeed, this property is elevated to the status of a basic axiom or principle.
- In a system, it is not sufficient that each agent knows its own intent: intentions may have to be shared with others, so that agents can form expectations about one another's behaviours. This is how systems work together in a collective or collaborative manner.

8. A promise does not guarantee an outcome. The promised outcome may be kept with 'best effort' by the agent concerned. An outcome that is not promised may be considered *ad hoc* or *taken for granted*. No agent in the system can form any expectation of a non-promised outcome.
9. An agent *policy* is a collection of promises that attach to assessable contexts.
10. A promise that does not promise a final state (e.g. only promises a relative transition), and which is independent of its starting state, is unstable and cannot be used to predict outcomes over time.
11. A system in which each agent does not promise a coordinated intent cannot collectively promise a desired outcome, and a user cannot expect anything about its behaviour.
12. All promises take a non-zero amount of time, as measured by a recipient, to keep, i.e. $T_O(\pi) > 0, \forall \pi$.

The importance of understanding promises can't be overstated. If an agent, behaving within a system or not, makes no promises, then it cannot be considered in error, nor can it exhibit a fault, even if it fails to meet others expectations. Thus, all components need to document their intended functions and limits.

1.5 FUNCTIONAL SYSTEMS ARE INTENTIONAL

A system begins with assembly of component parts that interact and develop. Unless trivial, systems exhibit bulk behaviours that cannot be fully understood from the behaviours of its components in isolation. Any assembly of parts, collected within any arbitrary boundary, may be viewed as a system. Such phenomena exist in nature, or they may be artificial.

Definition 7 (System)**.** *A collection of agents S that make interdependent promises within S, forming a stochastically connected graph.*

in other words, a system is a set of component resources, their interactions, and patterns of change, that can be assembled into a descriptive model with associated semantics.

If influence can be passed from an agent to another, even if it does not belong to the body, then it belongs to the total system.

Definition 8 (Stochastically connected)**.** *Two agents may be called stochastically connected, according to an observer agent O, if and only if it assesses a non-zero probability of them being adjacent, by promise, for some definition of probability.*

$$\alpha_O \left(A_1 \xrightarrow{+b} A_2, A_2 \xrightarrow{-b} A_1 \right) > 0 \tag{1.1}$$

The ability to describe a system is important: without this, we cannot speak of a system at all. Systems are always identified by human observers, which means that they are often based on arbitrary choices, and this affects the nature of what we see in them and mean by them. Subjectivity is a key aspect of systems, independently of how we might strive for impartiality in describing them.

Definition 9 (Intentional or functional body)**.** *Any system body B, which makes a $+$ promise to agents not in B may be described as expressing an intent.*

No matter whether the originator of an intention is a part of the system or not, *effective intent* is at work in systems that we assess to be functional. To this end, there is no need to become embroiled in discussions of the need for human agency or freewill, because such agency can be represented through proxies, be imitated by random processes (e.g. in evolution), and therefore we can't necessarily distinguish between apparent intent and the supposed genuine human article. That's because, like all phenomena, intent is ultimately judged in the eye of the beholder, not the originator. The key differentiator lies in the scale and semantics of the promises.

1.6 SYSTEM BOUNDARIES, OPEN AND CLOSED SYSTEMS

If a system were completely decoupled from its surroundings, and its users, we would not even be able to observe it or interact with it. So where does 'the system' start and end? Does it include users, its environment? Surely both of these interact with it, and therefore influence it. This is where system design often becomes rather wooly. One

tends to neglect the users and the environment in the interests of modularity, but, although convenient, this is paradoxical if not slightly dishonest (see figure 1.1). There has to be a way to focusing on the most relevant pieces of a problem, and approximating away the rest. This points us to the notion of a 'suitably idealized approximation' in science, in which we define subsystems as 'the system', from the viewpoint of an observer. There are well established techniques in science for making such approximations consistently.

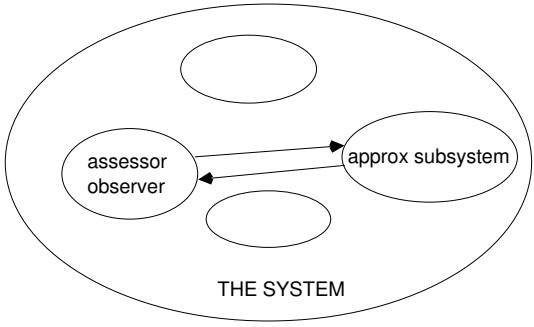


Figure 1.1: Where is the edge of a system? We should really talk about embedded subsystems, as no system has a meaningful edge if it interacts with a party defined to be outside of itself.

If a system comprises all that is connected by interaction, then we have am obvious problem trying to define a system as something that serves exterior clients. As soon as the clients interact, they become part of the system, as defined above. This makes obvious sense, but it's also not how we usually speak about systems. We are more familiar with the concept of a system of ownership, like a company or a service, which is basically a membership concept. A top level view of what defines an entity is used to define the limits of the system, without recourse to empirical truth. To capture this more familiar idea, we can refer to 'entity systems' or 'system bodies' rather than systems. This also captures the idea of a material body in physics, in the Newtonian sense.

Definition 10 (Entity system or system body)**.** *A collection of agents belonging S that promise membership of a collective body or entity B:*

$$S : A_i \xrightarrow{-(A_i \in B)} A_?, \tag{1.2}$$

$$A_j \in B \xrightarrow{+(A_i \in B)} A_i \tag{1.3}$$

By this definition, our common idea of the system is really B, but when it interacts with

exterior agents, we have to acknowledge the extension of the boundary to include client agents.

The boundary of a system is also a non-intuitive concept for a graph, as our normal idea of a boundary relates to the existence of a space in which agents are embedded. We perhaps think of a graph as herd of animals, whose outer members represent the boundary or edge of the group. But in the world of agents, there is no particular reason why this geometry would be related to which agents make promises to one another or interact as a result.

If we choose a random boundary to define a subsystem of interest the results might be surprising, because interactions across the boundary can play a major role in the outcomes within the boundary. So we must try to define boundaries sensibly as regions where the *coupling* between what is inside and outside the boundary is weak, or even absent. Interaction is key to understanding system behaviour. The body concept is useful here too.

Definition 11 (System boundary)**.** *The subset of agents belonging to a body B that promise to interact with agents not in B.*

Lemma 1 (The boundary of a system is indeterminate)**.** *The presumed boundary of a non-isolated system interacts with exterior agents, but the act of interaction extends the system and thus shifts boundary.*

The limits of a system are not an absolutely determined line in the sand. Every system has different *typed* boundaries, i.e. limits at which different types of promises no longer couple interior with exterior. A rational approach to systems therefore must take into account the types of interactions that take place. Systems are overlapping patchworks, not isolatable, modular components.

Following the traditions of physics, we may begin by defining open and closed systems:

Definition 12 (Open system)**.** *An arbitrary collection of interacting parts (called agents), each of which may interact with parts not defined to be within the system. Open systems are really subsystems with locally and globally incomplete information.*

Definition 13 (Closed system)**.** *A closed collection of parts (called agents) each of which may interact with other parts inside the collection. Closed systems are total systems with globally complete information.*

Closed systems are idealizations and approximations to reality. They are an artifice, defined to elucidate specific issues in a model, such as the formulation of simple rules. All practical systems are open systems.

Comment 1 (Top-down versus bottom-up)**.** *Many engineers (especially in computer science and engineering) are taught to design systems from the top down: that means starting with the whole and breaking it into successively smaller component parts. The problem with this methodology is: we can't build a system from the top down (except perhaps a sculptor who whittles away parts), so designing it this way leads to a mismatch between intent and capability.*

Top-down is a narrative construction: it is the way we explain through stories. It assumes to know what the top of the system is; however, we have already said that the boundary of the system is arbitrary. So what, then, are we designing? If we design from established components, that can make known promises (materials, off the shelf parts, etc), and build up, then we have an open ended process with a stable foundation. If we build from an uncertain 'top' towards an uncertain foundation, the whole edifice of the system is uncertain: nothing may be stable.

1.7 EMERGENT OR EFFECTIVE PROMISES

Emergent behaviour is a much popularized notion, and one that often defies a clear definition. Promise Theory is helpful here[BF07b]. We can define emergent behaviour as an effective promise, from the viewpoint of an observer agent O.

Definition 14 (Emergent or effective promise)**.** *Let $\{A\}$ by any collection of agents, and let $\pi^{(+)}$ be a promise of some property of behaviour that is expected by the observer O. Now suppose O promises the complementary promise $\pi^{(-)}$ to bind to $\pi^{(+)}$. If O assesses that this promise is kept, i.e. $\alpha_O(\pi^{(+)})$ is positive, when in fact no such promise has been made by A, then we claim that A exhibits an emergent promise or behaviour $\pi^{(+)}$.*

1.8 TRUST

Promises are closely related to the question of trust between agents[BB06].

Definition 15 (Trust in a promise π_X)**.** *An observer O's trust in a promise π_X is an assessment, $\alpha_O(\pi_X)$ of the probability that O can abstain from falsifying*

$$\pi_X : A \xrightarrow{+X} A_? \tag{1.4}$$

and thus avoid the cost of computing $\alpha_O(\pi)$. The combination of all such assessments about a single agent A may be called the trustworthiness of the agent A.

Trust, in turn, is related to predictability, in the eyes of an observer. One assumes that, by learning about the past or by building a relationship with system behaviours in band, we are able to predict something about the future behaviour. This, in turn, assumes a stability under the repetition of patterns.

Definition 16 (Predictability)**.** *A system that has stable and repeated observable behaviour, on a timescale much greater than the sampling rate, may be called predictable.*

1.9 PRINCIPLE OF CONSERVATION

In mechanical systems, in physics and engineering, the conservation of key quantities forms a number of principles that explain certain behaviours for closed systems. The best known include energy, current, momentum, etc, which constrain the avenues of possible change locally and globally. In economics there is a similar implicit principle of conservation of money. The generalization of these principles can be thought of in terms of information transactions. A transactional element of an information transmission is a symbol, i.e. an element of an alphabet used for the encoding of information, in the sense of Shannon's communication theory[SW49]. Information form a current, just as particles of water or electricity do. Each distinguishable 'particle' is thus a symbol and vice versa.

Example 1 (Conservation of current)**.** *In a closed region, like a pipe or network, what goes in must come out. So at a network node, one has Kirchoff's law for current J*

$$\sum_{in} J = \sum_{out} J \tag{1.5}$$

This assumes that the density of the current doesn't change. In a pipe, the current flow J and the density of a local region with density ρ: are related by a conservation equation:

$$\frac{\partial J}{\partial x} = -\frac{\partial \rho}{\partial t} \tag{1.6}$$

If the density increases in a region, it might appear to reduce the amount of flow, because it's more bunched up in size—but the accounting works as long as we count the scales consistently.

In any system, it's helpful to assume the conservation of some countable quantity whether we are certain of its immutability or not, because it allows us to measure things. By Noether's theorem this amounts to assuming the continuity of processes that carry or transport that quantity. In computing we can also treat data transactions as conserved.

Principle 2 (Conservation of transmitted information)**.** *Information, counted in bits or transactions, is never created or destroyed, but may be diverted or transformed into other forms within agents.*

Example 2 (Data circuitry)**.** *Agents may play different roles within data circuitry. Agents and promises that manipulate data tend to assume data conservation—a kind of data accounting. The table below shows a few example agents and their processes involved in this kind of accounting.*

Promised Role	*Agent*
Sources of data	*Inputs, sensors, distinguishability*
Sinks of data	*Outputs, garbage collection*
Reservoirs of data	*Databases, storage services*
Transport of data	*Network transport, data pipelines*
Amplification of data	*Services, replication*
Attenuation of data	*Idempotent semantics, indistinguishability*
	fixed point behaviours

Data density increases as interior storage is filled, within an agent. Data flow increases as data are transported to exterior agents. This example is a bit simplistic. What's missing from this story is the role of scale in how we count information.

All principles need some explanation. In some cases, they may be shown empirically true, subject to certain constraints and assumptions. For example, the flow of water through closed pipes can be measured, given that it doesn't leak. Other principles, like the conservation of energy or money are not so clear cut, because money and energy are virtual counters and they can be converted into other things. If water were turned into steam in pipes, conservation of water current would not be exactly true, unless one could account for those transformations. So it is with all conservation laws.

The conservation of data is not like the conservation of matter or energy. Creating, deleting, and moving data around costs energy at every stage, because our computing technologies are highly inefficient. The same is true of money in economics. Nevertheless, as an accounting principle, it is a useful artifact. Conservation laws are basically an assumption that's consistent with how we count phenomena.

1.10 SCALE AND CONSERVATION IN TRANSPORT PROCESSES

Data transport is one of the pernicious problems of human-computer systems. Networks are relatively scare resources compared to computers, and the demand for data intensive services is only growing. This places a tension on processes designed for the 'edge of the cloud', i.e. at the user end of the system, when they are driven by 'back end' infrastructure running in datacentres. Data centres are, almost by definition, far away from most users in a globalized world.

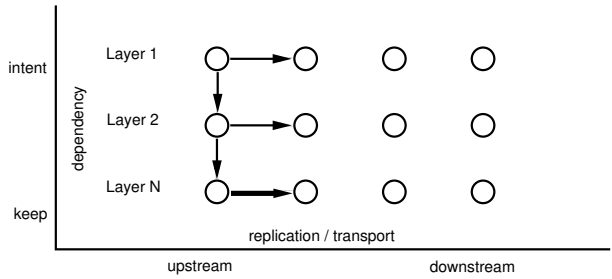


Figure 1.2: How information scales: greater data size requires greater time, so multi-part messages should be decoupled by timescale. For example, the intent to synchronize data can be separated from the actual transport of bulk. Interactions of intent are small quick messages. The keeping of bulk data promises require transactions on a much long timescale, which can be decoupled into a separate process. Many databases fold these into a single process by trying to freeze transaction time with a lock to ensure consistency of replication.

- There is therefore an imperative to avoid moving data around unnecessarily.
- The capacity for handling data depends on the relative impedances between upstream and downstream agents. If a link in the processing chain is slow it will lead to queueing and congestion.
- Local caching is the strategy of keeping data close to agents involved in frequent transactions.

In some cases, the insistence on centralizing information to avoid a human headache may lead to unnecessary communication at the wrong time. In a lot of cases, one can actually avoid embedded 'realtime' communication altogether, by ensuring that the local context is already pre-seeded with all necessary information. This was the approach

used in CFEngine: discovery of environment and pre-cached policy (advance decision-making) replaced the need to first collect names and locations and then push down a specific set of changes.

Example 3. *Measurements on a distributed platform are matrix measurements.*

How can we define a speed or a velocity? We need to define the markers from beginning to end, and thus the interval over which the measurement is made.

Distributed systems connect many processes together, and sometimes this leads to rate limiting as dependencies are queued up by strong coupling. To practise the principle of separation of scales, we try not to let slow processes delay fast processes. Unfortunately the slow processes are the ones, like storage operations, that lead to persistence of data and thus are the outcomes of greatest value. Computation is ephemeral and localized and therefore fast.

Trade off of frequency versus distance, or time to access (latency).

1.11 CONFIGURATION SPACE

The set of possible locations of states that characterize the information and condition of a system are collectively referred to as its configuration space. Location plays a role only to the extent that locations are connected by paths over which information travels.

Example 4 (Euclidean configuration space)**.** *In mathematics, Euclidean space is normally imagined as the configuration space for our physical world. The states that can be used to define distances come from the ability to identify points. Markers include matter and energy exhibiting different properties, in different coordinate locations. Coordinates themselves are an abstraction, and cannot be measured without something to mark them.*

In a computer system, the configuration space consists of the agents, computers, and their interior states and memories. How we refer to these by coordinates is a separate problem. The key principle in observing systems is observability. To observe a phenomenon we need to be granted access to the phenomenon, and promise to be observant, both of which can be expressed in promises.

The location of agents capable of keeping promises introduces another set of scales: distance, which is related to latency by the assumption that the greater the path length of information, the more likely it is to be distorted, delayed or interrupted. In IT systems, delays come mainly from switching and computation, as the speed of light in cables or air is sufficiently fast to make distance a minor concern at current levels. Obviously this may also change in the future.

1.12 DIRECTION IN CONFIGURATION SPACE

In the three dimensions of Euclidean space we are used to modelling with, dimensions are normally labelled by x, y, z. At each point, these are the possible directions in which one can travel. Not all systems have the same degree of freedom however. At some points, there may be dead ends, or only a limited number of exits to take. In a network, the outgoing routes from a location can have any number, but this number may be irregular and different at each location.

In the Promise Theory of this volume, we consider all configurations to be *graphs*, and makes no distinction between matter and spacetime, or things and empty space. Something is formally just a different state of nothing, like 1 and 0, but with more flavours. All spacetimes can be modelled as graphs, because everything that happens within them is a process, which involves a trajectory. Whether one considers spacetime to be continuous or discrete is not important—both can be represented as graphs[5], because graphs express the basic ordering relationship implicit in causality.

The notion of direction is more complicated in the case of a network, where we can't assume that every place in the network is exactly the same. Several paths from one location to another may exist through a network, either physically or virtually. So what seems to be a different direction initially just ends up leading to the same place. Dimensionality is scale dependent.

Example 5 (Computer networks)**.** *In a computer system, there might be a wireless network connecting computers along side a cable network. These offer parallel routes from one computer to another. Starting out going wireless seems to be a different choice from the perspective of the computer (a different interface), but on a larger scale the choices converge at the same place, so they are not independent at the larger scale.*

These are not parallel universes, because they connect the same points—they are shadow networks, like road and rail, or flights and shipping. There can be multiple channels with different labelled properties between the same points.

Definition 17 (Parallel)**.** *Paths or processes that do not interfere or contend for any common resource may be called parallel.*

In figure 3.3, for instance, the horizontal promises can be made in parallel if the network connections for storage and database logs are independent channels—either physically separate cables or virtually partitioned slots.

1.13 CHANNELS OF COMMUNICATION AND PROMISE-KEEPING

When referring to any kind of channel between agents, we shall assume that agents transmit singular and serial streams to propagate their influence, in the form of 'information' in the most general interpretation—one symbol at a time. Sometimes there are multiple parallel channels, which may either interfere or reinforce one another. This brings up a number of issues about processing steps, and the clocks that measure the progress of the system. We'll return to discuss these several times.

Serialization remains important at single nodes, even when parallel channels are in operation. For an agent to receive simultaneous parallel transfers, we would require the confluence of parallel channels to be sampled at within a single tick of a process clock[6].

1.14 HOW WE COUNT INTERACTIONS AND THEIR OUTCOMES

Developing formal models to describe resilient or reliable systems is a singular challenge. Systems are inherently subjective interpretations of dynamical processes, and treating subjectivity, in as impartial a manner as possible, presents several obstacles. Nevertheless, this is the challenge of any systems theory, and it is what promise theory was created for.

All systems have costs as well as benefits. It's common for engineers and designers to construct interfaces and narratives that apparently conceal costs. If we measure a system by one criterion and push the cost to a different criterion that isn't explicitly tracked, we may create the illusion of an objective optimization. It's important to count costs fully and fairly. However, such a fair viewpoint may not be available to everyone in a system, so subjective fairness can appear skewed. The exterior (or 'god's eye') view energy cost of a system may well be the only fair evaluation of a system, which is one reason why conservation laws play a key role in describing process quantitatively. However, because agents within the system have only a limited access to information, close to them, this global view might also not be available to any observer.

The greatest accomplishment of Newton and his peers was to formulate a description of the world in which we model behaviour by counting. By introducing counters (energy, money, etc), we can literally account for changes as long as we assume the counters are immutable: once created they don't just disappear without telling us. The tokens counted are deterministic in their transactions. The Newtonian approach effectively associates promises with outcomes in a transactional manner. There is no such theory in Computer Science, with the debatable exception of some aspects of database transaction theory,

but having one would be quite desirable. This is amongst the on-going goals of Promise Theory. We can then add to that the complications that come from indeterminism. We work only with partial and incomplete information. Whether underlying mechanisms are deterministic and transactional or not, we may not be able to observe deterministically.

Classical reliability theory, following a mathematical tradition, has a history of approaching the concept of manufacturing reliability statistically[Nat98, HR94]. Systems were assumed to be networks of black box components, and faults are treated as statistical occurrences within the components rather than their interactions. Moreover, the functional semantics of components were suppressed deliberately to give an impartial account of faults that was generic but limited. To modernize and complete a model of functional systems, in which semantics play a major roles (e.g. interactive systems, software, and services[WG06]), one is forced to involve systemic issues at multiple scales, as well as 'human issues' (all semantics have human origin)[HWL06, DH06, Bur04a].

The aim of this approach is to capture some of both these views, by exposing the semantics and modelling the functionality along side the fidelity of active and passive elements. Trying to describe systems without breaking them into interacting parts leads only to vague descriptions that emphasize subjectivity. By using promise theory to describe systems through agent interactions, we can find a useful middle ground that respects the traditions of physics, and places subjectivities on a clear footing.

Faults and errors involve deviations in semantics and interpretation as well as deviations in dynamical behaviour. The aim of formalization is not to present the perfect *fait accomplit*, but to bring some straightforward enhancements to the state of system formalization.

1.15 THE INTENDED 'PURPOSE' OF A SYSTEM (SEMANTICS)

Whether a system is natural or artificial in origin, we may describe it as having a *purpose* if 'observers', 'assessors', or 'users', who interact with it, infer that purpose, by projecting their own semantics onto its collective behaviours. If a system plays a role within a larger process, then it can be said to serve that process—that purpose. Each observer assesses the defined system in relation to their understanding and sense of values. This is an inherently subjective matter, which makes assessing all system outcomes controversial. The quotes at the start of the chapter illustrate a few of the pitfalls and preconceptions that await the unwary, in attempting to discuss the idea of failures and errors in systems.

It's important that we be able to explain purpose, like intent—that we don't merely reject it as a subjective or anthropomorphic irrelevance, because purpose is a dominant issue in what we mean by systemic affairs. In a system that was evolved rather than

designed, there may not seem to be an explicitly promised purpose. Evolution is merely a competition between mutation and constrained selection, and can only describe a purpose indirectly by ascribing a meaning to the selection criterion—but, then, a human decision is just a competition between alternative narratives, eventually selected by some criterion. Neither process requires advanced intelligence. In nature, the criterion is stability within a particular niche context, i.e. niche survival. In human terms, the criterion is stability of narrative. Artificial evolution may be based on design goals, or choices, e.g. genetic algorithms. In this case, the promises are implicit, and lie in the value judgement of observed behaviour.

1.16 SYSTEM CHARACTERISTICS AND OUTCOMES

The observable outcomes of a system are measurable states, defined at different scales of observation[Bur15a]. It is assumed that a system may be described by the propagation or evolution of such states over time and space[Bur14]. In fact, a timeline is defined to be a sequence of changes in the states of the system! This kind of model for 'propagation' from one state to another is well understood in physics.

Definition 18 (State of an agent (microstate) q)**.** *A description of the value taken by a variable q that characterizes the condition the agent, at a given scale of observation.*

Definition 19 (State of the system (macrostate))**.** *A bulk average description of observable microstates of the system, at the scale of observation.*

Note how the scale of observation plays a central role in describing the system, and a rescaling (coarse graining) may redefine a macrostate to be a microstate at a larger scale. This reflects the fact that we always have limited, and generally incomplete, information about a system. In other words, our knowledge of a system is always some kind of measured approximation to what is going on. States are most useful in relating promises to outcomes, thus they become a language for defining desired state outcomes.

Definition 20 (Promised outcome)**.** *A specification of the values of any variables q whose values have been promised by a promiser.*

A description of behaviour in terms of outcomes (rather than the litany of steps to achieve the outcome) is both compact and irreducible. It has numerous advantages that I'll try to outline throughout these notes. An outcome may go through a number of stages, in which case we may speak effectively of a function of 'time' $q(t)$, where the time is defined by the sequence of changes to q.

Definition 21 (Assessed outcome)**.** *A specification of the values of any variables q whose values have been chosen and assigned by any agent assessing a promise.*

Unless stated explicitly, we'll assume that promised and assessed outcomes are the same to avoid unnecessary verbiage.

When the outcome of one or more promises results in changes over time and possibly space (several agents) in a causal order, we can talk about *processes* and *trajectories*[7]. Although these words mean similar things in common parlance, it's helpful to make a distinction in promise theory. The sequence of states a single agent passes through is called its trajectory:

Definition 22 (Trajectory)**.** *A sequence of values for a state q belonging to an agent. The path in q space, at each sample time t defines a quasi function $q(t)$.*

whereas a collaborative, collective chain of promises that forms a partially ordered set of pre-requisite dependencies is what we mean by a process.

Definition 23 (Process)**.** *A directed graph of promise bindings that results in a causal chain of outcomes from a number of source agents to a number of receiver agents.*

A process represents our common understanding of workflows, manufacturing production lines, and data pipelines. A trajectory is more like the personal journey of discovery by an individual agent that may or may not be part of a process.

- Outcomes are states that are the results of interactions between agencies of the system. Interactions consist of two kinds of promise, labelled + (offered or intended) and - (accepted or received).
- Behaviours may be characterized by observers viewing at different scales.
- The strength of agent-agent interactions determines how well behaviours at different scales (de)couple.
- Strong interactions may be considered hard dependencies that form a network along the interaction paths.

Lemma 2 (Open systems have unpredictable states and trajectories)**.** *We cannot predict all behaviours within an open system, as new information can enter that is, by definition, not predictable.*

A proof of this was given in [BC11]. In a closed system we have, in principle, complete information. Of course, that assumes that the system is also closed from within, i.e. that no unexpected behaviours come from within agents in a system (such as breakages, capricious intent, or even quantum mechanical uncertainty). The notion of a closed system is essentially an idealization to help isolate certain causal trajectories in system dynamics, and enable formalization. They should not be considered realistic in the world around us.

1.17 SUPER-AGENCY AND SCALING

We may aggregate a collection of cooperating agents into an aggregate 'superagent', that behaves as a single entity at a larger scale. Promises that only affect what is inside the superagent may be called interior, and promises that affect what is outside may be called exterior (see figure 2.8). In this way, we can imagine partitioning the entire world into open subsystems that we define in any convenient manner.

1.18 PREDICTABILITY AND PROMISE

If a system is stable enough to be predictable, then we can describe its patterns of time development according to some formulae[8]. We may only be able to describe the envelope of possible outcomes by a formula, or we might be able to predict the exact state of the approximate model! Promise Theory allows both dynamic and semantic outcomes to be placed on a similar footing[Bur13a] by defining comparative scales for outcomes, which we call promises. This is a useful approach, with great flexibility.

1.19 DRIFTING INTO FAILURE

In his book Drift into Failure[Dek11], S. Dekker describes how 'failure' often creeps up on the unwitting users or participants. As systems are used, and maintained, they are improved and adapted. Gradual adaptations, changing the envelope of apparently innocent parts have unexpected consequences as the changes are amplified.

Flawed expectations, lack of knowledge or awareness of the actual behaviour of the 'unruly system' all contribute to a gradual drift of a system out of its fitness envelope and into the realm of unreliability. Problems with change (errors), lead to problems of state (faults), and expose problems of inadequate design (flaws). In other words, failures occur because systems are not simply not static. They undergo planned and unplanned change, driven by environmental forces we often don't even realize we need to know about[HWL06, Bur13a].

Could we begin to address the build up of drift? This seems plausible, in a limited sense. Managing drift is what maintenance routines are for, and tools can help. Expectation creep can be stopped by mechanical oversight, with independent auditing[9]. The key is to have a declaration of intended outcomes (promises) that can be monitored in a targeted fashion, without pressure from other concerns to relax standards. The trouble is that we can't even define drift unless we know what the system is meant to promise. Even then, the promises might be naive. This is the challenge. I will return to this in the chapter on scaling.

1.20 HOW TO USE THIS VOLUME

The material in this volume forms a unique approach to the study of systems. The discussion is extensive and yet covers only a limited set of issues, hopefully serving as an example roadmap to be extended and improved by others. Through its principles, and examples, Promise Theory offers tools to help design, express, measure, and reason about systems. It forms an impartial methodology for discussing all manner of issues. It is not a recipe for unwavering 'success'. Indeed, while the method may be impartial, its application need not be. If nothing else, it embraces uncertainty.

Designing a system, which explicitly promises and documents intent, means one can gradually test hypotheses as a system evolves, and with iterative yet focused measurement. Engineers tend to assume that designs should be either abandoned or patched in operation when they fail to behave as expected, and that generic monitoring of systems suffices to address issues that may arise. However, that assumes that you know what you're looking for. Thinking in terms of actual promised specifications, rather than *ad hoc* expectations and imposed requirements turns out to be a consistent position. If you don't know what to measure because you didn't make a clear promise about a system, then you need a leap of imagination, or a lesson in bad experience rather than just a little algebra, to help you. This is the nature of mathematics and of systems. Promise Theory offers that simple algebra of cooperation to align intent with actuality.

There is a lot here—certainly too much to learn and remember, so, while I've presented the discussions as a series of theoretical lectures, I've also tried to make the notes serve as a reference work to dip into on occasion. As engineers, we can use the tools developed here to sharpen our understandings of goals and mechanisms, to refine a critical approach to the descriptions of systems. May these volumes sit on the shelves of the best engineers, architects, and designers as inspiration and as a reminder that human-machine systems are not beyond the reach of rational analysis.

CHAPTER 2

HUMAN SYSTEMS

> 1. A robot may not injure a human being or, through inaction, allow a human being to come to harm.
> 2. A robot must obey the orders given it by human beings except where such orders would conflict with the First Law.
> 3. A robot must protect its own existence as long as such protection does not conflict with the First or Second Laws.
>
> –Isaac Asimov (1942)[Asi50]

In Isaac Asimov's famous robot stories, he presented robots and automation in human form. Instead of imagining self-driving car, a human form robot would sit in the driver's seat of a car and operate a human-machine interface instead of a human. This was a perfect way to illustrate both how we treat humans as machinery in industrial methodology, and how we also empathize with inanimate systems and have trouble in distinguishing the roles for humans and machinery in practice. Promise Theory treats all actors in a system as 'agents' of undetermined kind (or kindness). This can be a helpful impartial means of understanding systems in which human concerns are important. In particular, it can allow us to apply engineering methodology to areas where traditionally only psychologists were allowed to tread.

2.1 'SOFT TOPICS' AND PROMISE THEORY

Before getting more abstract and technical, in later chapters, it may be helpful to examine some human scenarios, where leadership and conflicting intent come under scrutiny. If we feel comfortable thinking about human issues in terms of formal agents, promises,

and assessments, it should be easier to suspend disbelief later, when matters get more complicated. Promise theory can be applied straightforwardly to deal with a number of these soft topics, or human issues. Many of these topics have resisted formalization in other frameworks, because they provoke emotional issues that are difficult to render impartially.

It would be too much to expect Promise Theory to claim unique answers to every issue, but it's simple assumptions can nevertheless be a powerful tool for clarifying issues. The purpose of the principles and their application is to draw clear lines that may simplify assessments and decisions based on them. The challenge of any theory is not to lose expressibility in so doing, but rather to curtail or even deprive the ability of individual analysts who may wish to bend words to impose their own agenda.

2.2 INFLUENCE IN A HUMAN SYSTEM

The channels of influence, in a human system, are formed from written and verbal communications, as well as less obvious things like body language. Some communications are one to one; sometimes we broadcast information in order to influence a larger number of people. Mass suggestion, propaganda, marketing, etc, are attempts to create a large scale 'field of influence' that we hope will influence. This is an uncertain business, however. Although we tend to assume that communication is either deterministic (in command and control) or sometimes probabilistic (a fraction of the population will respond as we hope), we cannot escape the basic facts of the communication channel from sender S to receiver R:

$$S \xrightarrow{+X_S} R \tag{2.1}$$

$$R \xrightarrow{-X_R} S, \tag{2.2}$$

namely that what we broadcast $+X_S$ may be interpreted differently by each agent receiving the message, by filtering is through X_R. So the transfer is only $X_S \cap X_R$. Any influence this confers onto dependent promises to third parties T are a function of this overlap:

$$R \xrightarrow{+f(X_S \cap X_R)|X_S \cap X_R} T, \tag{2.3}$$

i.e. not of the full X_S intended by the influencer. So, a field of 'force' or influence for human agents is just like any other field of influence, perhaps with more complicated semantics.

2.3 LEADERSHIP IN A HUMAN SYSTEM

The subject of leadership is an interesting one that serves as a useful lens through which to look at human roles. It overlaps with several of the foregoing topics: management, the mandate of authority responsibility, and assurance of outcome. A leader might be characterized as an agent who has assumed responsibility for an outcome, or who has received a mandate to act as an authority is making policy, or one who grants permission for subordinates to act. The basic disagreements about what leadership is supposed to be often revolve around these different interpretations—and, in practice, we speculate that problems of leadership arise because appropriate promises have not been made in an appropriate way.

When planning an outcome, as a leader or manager of a predominantly human system, it's tempting to formulate what we imagine to be a straight path to the outcome—from where we are to the outcome we desire. That path might not be the path of least resistance. Indeed, it might not even be possible to construct[10]. A project or outcome planner needs to explore what paths *can be promised*. Many management frameworks lay down processes to try to apply brute-force persuasion and resources to solve problems. From a Promise Theory perspective, this doesn't make sense. The alternative is to start by surveying what promises are currently given, then what promises might be given if invited in the right way. The human scenario is very different from a machine scenario.

Leadership may be a promise to lead, based on promises of a mandate of authority—but what tools should an authoritative agent use to maintain the fragile symbiotic cooperation of a group? An approach that works on a small scale might not work on a large scale, as we know of phase transitions, yet leaders nearly always try to maintain the same approach as companies and nations grow. Not thinking through the actual promises made may leave us with no idea about why an outcome failed. Many an outcome failed because of improper assumptions—not doing one's homework to find the actual state of affairs. Promises express semantics but also context, in the form of a network of relationships and dependencies. There is key information in those networks that tells the channels of cause and effect—how influence propagates through a system, like a human community.

Example 6 (Undifferentiated leadership as a job rather than a role)**.** *Leaders often believe that those agents structurally 'under' them, in an organizational hierarchy, have granted a mandate for them to subordinate themselves voluntarily in all matters, when in fact no such mandate has been given. Organizational structure is a blunt instrument, which doesn't normally distinguish authority based on specialization—rank in a hierarchy supercedes genuine authority. As a result, many agents may feel attacked by impositions on matters where they believe the leader agent exceeds its authority. Job title is not always a sufficient discriminator to mandate authority on a wide range of issues.*

Leadership may be viewed by some as the idealized planting of a seed that freely snowballs and accretes enthusiastic support, as it rolls down an unimpeded slope, landing in goal of common approval. Conversely, it might be viewed, by others, as a wielding of brute-force, imposing tasks onto agents, willing or unwilling, in accordance with a pre-planned schedule, designed by a central coordinator, and executed with a whip. Both forms of governance are common, and both work under the right circumstances. Circumstances will likely determine the best choices in a given context. Either way, we need to be pragmatic. Cooperation depends on the marshalling of promises in either case.

Not thinking through the actual promises made may leave us with no idea about why an outcome failed. Many an outcome failed because of improper assumptions—not doing one's homework to find the actual state of affairs. Promises express semantics but also context, in the form of a network of relationships and dependencies. There is key information in those networks that tells the channels of cause and effect—how influence propagates through a system, like a human community.

2.3.1 SEEKING COOPERATION AND SUPPORT OF OTHER AGENTS

How do agents come together and interact to form collaborations and effective processes? This is obviously a complicated issue, and yet the simplistic notions of Promise Theory can still help to frame our thinking without getting bogged down with heuristics. Let's see what happens when we apply some of the principles.

We have two tools to use: promises and impositions, in both $\pm$ varieties. If we think about cooperative processes in companies, firms, institutions, clubs, or even countries, there is a mixture of command and control, and voluntary cooperation. Command and control suggests a kind of force or coercion involved in inducing collaboration. Even in democracies, there might be free voting but before that there is an imposition of candidates who want to force their ideas onto a population. They may claim to make promises which may or may not be kept. These may even be deceptions or lies. Our notions of freedom and imposition are clouded by context and emotional considerations.

In any system with competing objectives, the challenge of winning coherent support lies in condensing the available agents around a small number of alternatives. The larger the number of alternatives, the less stable a majority 'win' can be.

Example 7. *When voters promise to support democratic choices in an election, the counting of supporting votes may lead to problems of indecision and paralysis in different ways at different scales. The insistence on a majority becomes tricky:*

$$\exists i : voters_i \gg voters_j, \forall j \neq i, \tag{2.4}$$

Here, the meaning of $\gg$ is to be decided by some arbitrary margin. This may work well with few alternative choices and large numbers of voters who promise support for each

choice, because the likelihood of a 'tie' (where no single agent satisfies the condition) is small, but the promise of a clear outcome becomes less probable as each alternative choice gets less support. If the number voters becomes too small, or the number of choices too great, one dilutes the relative strength of votes, so that it may not be possible to promise (2.4) for any choice i. Coalitions between groups, where two or more i join to form a single choice (to be determined by cooperation), may then be necessary in order for a superagent group to be able to promise a majority that can be accepted. Coalition involves a secondary level of cooperation, in which agents form superagents that now have the same problems on a smaller scale. In smaller elections one rarely sees democracy practiced, both because many voters would assess the opinions of only small numbers as arbitrary, and it could simply lead to paralysis and instability. The same issue occurs when a larger population votes for a representative, who in turn votes for them in a parliament or board of directors. Such intermediate agents vote with 'special powers' that promise to represent opinions of collections of voters—though the intermediate agent law (see [BB14a]) shows that they may not be trustworthy. When voting in small numbers, the rules may accept a majority of one ($\gg$ becomes simply $>$), and the representative elections prefer odd numbers of voters to avoid tie-breaking during votes. The process of scaling is quite unstable to numbers, and there is a sense of misrepresentation—that the forms of democracy are followed without the promise of a majority having any significance. By careful engineering of categories, one could arrange for any result to win.

The balance of power between factions is therefore fragile when agents give their support voluntarily. This leads many to assume that the use of force is a necessary part of governance.

Force can take on several forms, from the use of an army to subdue opposition, to the insistence of compliance by *obligation*—an implicit threat. We know that, statistically, impositions of these kinds may be ineffective—not because of freewill or individual unwillingness, but simply because agents may be unable to comply with an intent that didn't originate with themselves. This suggests that there needs to be a balance between imposition and promise in any kind of governance process. That seems to be true at the lowest levels of physics (even if we don't understand why) but the nature of voluntary and involuntary force has rarely been discussed impartially at the level of social systems. Instead, many authors have used Game Theory to imagine a conflict scenario.

2.3.2 INVITATION, ATTACK, AND COMMAND

Let's take a simple promise theoretic view and consider an agent S of some size who wants to influence the cooperation of another agent R. There are some distinct types of

communication from invitation to intrusion:

1. *Open Invitation*: A completely unconditional promise to either i) offer $+X$ to R, in the hope R might promise to accept:

$$S \xrightarrow{+X} R \tag{2.5}$$

or, conversely, ii) to accept $-X$ in the hope that R might promise to supply:

$$S \xrightarrow{-X} R. \tag{2.6}$$

In other words, an 'open invitation' is an opening in the form of a directed promise without coercive intent, e.g. promising to leave a door open in case someone wants to come in. The invitation does not suggest the correct or acceptable outcome, it only makes one available. We shall want to go further than this and express the meaning of a 'directed invitation' in section 2.3.3.

2. *Publish*: The generalization of an invitation is to publish to a wider audience, in the hope that agents may accept or even subscribe. This is a wider promise of availability.

$$S \xrightarrow{\pm X} * \tag{2.7}$$

For instance, the publishing of a magazine which interested parties may voluntarily choose to read.

3. *Command or targeted intrusion*: A directed imposition to accept or request:

$$S \xrightarrow{\pm X}\!\!\blacksquare\ R \tag{2.8}$$

Such an imposition might be an uninvited guest or an expectation of compliance with a directive. Steering a ship is an imposition in the sense that the ship promises to accept steering, but not a specific course. If a single course is accepted in advance, then the situation is reversed: the pilot is accepting a menu option promised freely.

4. *Broadcasting/Flooding*: The publishing of an imposition leads to a scaled intrusion, such as in broadcasting.

$$S \xrightarrow{\pm X}\!\!\blacksquare\ * \tag{2.9}$$

For instance, even an *appeal* to voluntarily give money to charity is an imposition, because the message itself is forced into the recipients by media or by street collectors. It is a targeted imposition of a request or invitation. Marketing, advertising, and propaganda are examples.

$(\pm)$	$(\mp)$	INTERACTION
INVITATION	STEPPING UP	$S \xrightarrow{\pm X} R$
PUBLISH	SUBSCRIBE	$S \xrightarrow{\pm X} *$
COMMAND	ASK A FAVOUR	$S \xrightarrow{\pm X}\!\!\blacksquare\ R$
BROADCAST	COMPLIANCE	$S \xrightarrow{\pm X}\!\!\blacksquare\ *$

Table 2.1: Comparison of interaction types in coordinating autonomous agents, in a human context.

From the complementarity between $+$ and $-$ promises, we know that there must also be an interpretation of every $+$ promise as a $-$ promise. For example:

1. *Stepping up*: A promise of $-X$ to accept an invitation:

$$S \xrightarrow{-X} R \tag{2.10}$$

   or a promise to provide $+X$ if an agent needs it

$$S \xrightarrow{+X} R. \tag{2.11}$$

   In other words, the complement of an invitation is a kind of generosity.

2. *Subscribing*: A broader promise to accept or provide to or from a variety of sources.

$$S \xrightarrow{-X} *. \tag{2.12}$$

3. *Asking for a favour*: Targeting an agent to offer a promise that was not autonomously given.

$$S \xrightarrow{\mp X}\!\!\blacksquare\ R. \tag{2.13}$$

4. *Obliging compliance*: Widespread inducement to promise certain behaviour, e.g. everyone must wear a helmet or fasten their seatbelts.

$$S \xrightarrow{\pm X}\!\!\blacksquare\ *\,. \tag{2.14}$$

2.3.3 A MORE REFINED VIEW OF INVITATION VS ATTACK

In the foregoing section it emerged that invitations cannot be quite as simple as the distinction between promises and impositions, as invitations can themselves be imposed onto unsuspecting agents (e.g. a subpoena to appear in court), or promised (e.g. a special offer displayed in a magazine). There is thus a distinction between the delivery and the intention—an extra level of diplomacy that establishes a general basis for proposing a more specific outcome (see figure 2.1). It's helpful to separate these into two kinds of invitation:

- *Open invitation*—a non-exclusive invitation, which leaves an option open for matching agents to avail themselves of.
- *Directed invitation*—a formal announcement of the promise to accept interaction, made on an exclusive basis, i.e. targeted to a specific identifiable agent.

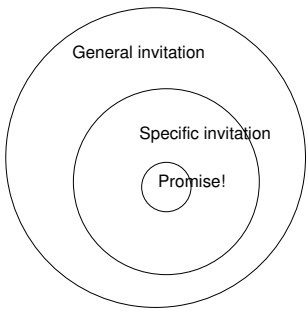


Figure 2.1: Invitation is a mechanism for homing in on a promise without a formal agreement, paving a way to agree by suggesting that a promise will be accepted.

Example 8 (Open invitation to shop)**.** *A regular shop does not usually invite customers by mailing them a ticket to enter, or by promising to sell every good in the store separately (some catalogue services do this). Nor do shoppers accept or reject every offer. This would not be a scalable matter. Rather, some superagent scaling is involved. A shop offers to be open for selling goods, and shoppers promise to accept these opening hours. This is the basic invitation framework for a promise dialogue. Customers then browse the goods, all of which follow the generic pattern: 'shop promises to sell if you pay a given price', and the shopper equally promises 'I promise to pay a given price if you sell me the goods'. This deadlock is broken by the customer imposing a choice and paying up front.*

Example 9 (My charge is $+e$)**.** *In physics, particles of different kinds advertise different kinds of charge (electric charge, colour charge, and so on). These play the roles of open invitations for other agents with charges of the same type to interact with them. The idea of an electron sending an RSVP to its neighbour is somewhat ridiculous, though messenger particles do exist for the charge interactions. The semantics of interactions are not directed by identity—electrons and quarks are indistinguishable, so their promises are non-exclusive.*

Example 10 (Invitation and deadlock in commerce)**.** *If a shop leaves its door open for all to enter and buy goods, this is an invitation. The shop also promises each good for a price. Shoppers may promise to pay the price for a good if available. The deadlock between: 'you can have the good if you give me the money' and 'you can have the money if you give me the good' is broken by the imposition of voluntary payment by customers. In service interactions, service providers usually have to be the ones to break the deadlock by performing the service first. The service is less tangible than the good so its reality may need to be established before trust is built.*

Example 11 (Service negotiation)**.** *Expanding on the simple view in the previous example, there can be several layers to a service contractor interaction, with layers of promises playing the role of invitations converging on a desired outcome.*

A contractor may advertise the availability of its service, which acts as an open invitation to hire its service. Acceptance of that leads to a problem discussion with a potential client, perhaps a Purchase Order (PO) for the solution, followed by an invoice for the purchase order (whether the solution is achieved or not) and payment of the invoice. There are layers of invitation here. Let's label the service provider or contractor by S, and the client as C:

$$S \xrightarrow{+skill,availability} C \quad \textit{(open invitation to hire)} \tag{2.15}$$

$$C \xrightarrow{-availability} S \quad \textit{(accept)} \tag{2.16}$$

$$C \xrightarrow{+problem} S \quad \textit{(explain need)} \tag{2.17}$$

$$S \xrightarrow{-problem} C \quad \textit{(listen)} \tag{2.18}$$

$$S \xrightarrow{+solution|PO} C \quad \textit{(listen)} \tag{2.19}$$

The contractor takes some risk by discussing the problem up front, but withholds the solution, instead inviting the client to make a purchase order, and promising the solution conditionally on the PO. This shifts risk over to the client since the purchase order will

be invoiced unconditionally on a solution.

$$S \xrightarrow{-PO|problem} C \quad \textit{(invitation to commit)} \tag{2.20}$$

$$C \xrightarrow{+PO} C \quad \textit{(accept and commit)} \tag{2.21}$$

$$C \xrightarrow{-solution} S \quad \textit{(accept and trust promise of solution)} \tag{2.22}$$

An invoice is promised by implication of the PO:

$$S \xrightarrow{+invoice|PO} C \quad \textit{(invite client to pay for the PO)} \tag{2.23}$$

$$C \xrightarrow{-invoice} S \quad \textit{(accept)} \tag{2.24}$$

$$C \xrightarrow{+pay|invoice} S \quad \textit{(accept and trust promise of solution)} \tag{2.25}$$

$$S \xrightarrow{-pay} C \quad \textit{(accept payment)} \tag{2.26}$$

Note that payment is not conditional on the solution outcome in this model—only on the promise of a solution.

The example above illustrates the rich dynamics that can be involved in interactions on the complex level of semantic systems. This is not a unique feature of human systems. Clearly, the role of an invitation, in the human sense, feels like much more than simply leaving the door open. It may consist of many layers, and a gradual convergence of wooing the parties to cooperate. This is also seen by animal species in the wild during mating! It's an auxiliary communication—an announcement of intent to grab attention, which may itself be imposed or promised passively. By announcing an intent to invite one may even increase the perceived value of potential acceptance.

Example 12 (Don't drop litter!)**.** *Advertising and propaganda that aim to teach people not to drop litter may be framed as invitations. These may be imposed upon them, for the public good, or targeted at individuals, depending on the scale of the targeting. It could be presented as 'you'd better do as your told or else' or as 'we invite you to be a better person'. Both approaches have been used in different countries*[11].

Invitations are sometimes handed out as promises and sometimes as impositions. The essence of an invitation is that one first tries to establish a promise acceptance at a general level to pave the way for the specific outcome. There is a progression of constraints that home in on the final targeted interaction, edging forwards 'is it okay if we do this? Good! Now what about this?' Invitation is a convergent surrogate for a process of agreement, in which no formal agreement is ever made. So how shall we decide whether the imposition of a decision for change by a manager, leader, or exterior entity of any kind is made by diplomatic invitation or by attack? What is the difference between these two? How are they defined and how can negative behaviours be avoided?

To reveal this in more detail, we may first nail down some underlying definitions pragmatically, without getting too deep into nuances. The following definition helps to show how a directed invitation works as a proxy:

Definition 24 (Directed Invitation to do X)**.** *A prior promise or imposition of $I(-X)$, conditional on X is an invitation to accept a promise of $+X$:*

$$\textit{Inviter} \xrightarrow{-X} \textit{Invitee} \tag{2.27}$$

$$\textit{Inviter} \xrightarrow{+I(\mathbf{def}(-X))|-X} \blacksquare\ \textit{Invitee}. \tag{2.28}$$

may be considered an invitation for the invitee to promise:

$$\textit{Invitee} \xrightarrow{+X|I} A_?, \tag{2.29}$$

assuming that the invitee accepts:

$$\textit{Invitee} \xrightarrow{-I(X)} A_? \tag{2.30}$$

$$\textit{Invitee} \xrightarrow{--X} A_?. \tag{2.31}$$

The latter may be considered incorporated by the promise of $+X$.

We can read these as follows: (2.27) says: there exists an X that the inviter will accept unconditionally, and (2.28) says: given that acceptance of X, The inviter issues an invitation $+I$ for invitee to accept by promising $-I(X)$, and ultimately $+X$. Notice that it is the acceptance $(-X)$ of the promise, by the Inviter, which is unconditional, and the invitation merely encapsulates knowledge $\mathbf{def}(X)$ about its proposal, without assuming the Invitee would know about it in advance. The promise of $-X$ alone may not be sufficient information to imply a proposal to match it with $+X$. The role of an explicit invitation $I(X)$ is to signal a clear promise, rather than merely laying out a path to completion. Note that, by the rules of conditional promises, the two promises (2.27) and (2.28) are identical to an unconditional promise of $+I(\mathbf{def}(-X))$, so the promising of an unconditional invitation may also be conversely reinterpreted as an explicit acceptance of a raw promise of $+X$.

For example, suppose X is 'your attendance of an event'. The inviter promises to accept the attendance unconditionally of the invitee. This offer of open acceptance may be ambiguous, so the intent to invite is emphasized with an explicit promise of $I(\mathbf{def}(-X))$ which advertises the content of the promise $-X$, so the invitee knows what it's getting itself into. This is what we call a directed invitation. It's a positive offer of the complementary promise! Note that each agent is the originator of its own promise.

The response (hopefully a full $+X$) might only be a partial overlap with the $-X$ offered. Nothing is imposed by the other party.

The opposite of an invitation is an attack. This is the imposition of an offer $+X$ before establishing grounds for general acceptance of whatever X might be.

Definition 25 (Attack on agent R)**.** *Any imposition on a recipient agent R by a source agent S:*

$$S \xrightarrow{+X} \blacksquare\, R, \tag{2.32}$$

for which there is no prior invitation or acceptance by R:

$$R \not\xrightarrow{X} S. \tag{2.33}$$

The definition of 'attack' in Promise Theory is thus an attempt to induce cooperation without acceptance or invitation. We can't claim that invitations are promises and attacks are impositions unequivocally, because often there are mixtures of the two in communication. Communication might be voluntary, but its content imposes, or vice versa.

Example 13 (Covert advertising—hidden impositions)**.** *A magazine is published and distributed for people to take freely. The reader promises to accept unknown article content, but does not promise to accept advertising. So there is a hidden imposition in the promise.*

Example 14 (Overt advertising—imposed promises)**.** *Flyers are posted through your letter-box, or messages are pushed into a social media channel. The messages promise a free sample, but the users did not promise to accept messages of this kind.*

Example 15 (Public education)**.** *We might ask: at what point does a public service become an imposition? The promise theoretic answer seems to be: when the service was not initially sought by its intended users. The distinction between 'spam' or propaganda, news, advertising, etc, is much less universal than we tend to think. The perceptions change by scale, by circumstance, and by perspective. The role of invitation thus seems to be to home in on a specific*

Interactions between agents, which iterate back-and-forth in a particular sequence, are the chief mechanism by which agents accumulate a foundation of promises on which trust is built. As with any house of cards, a stack of dependent promises is fragile to the order of proposals. We see this just from the two definitions above. The attempt to exploit the fragility of such dependency is how one disrupts, perhaps with an intent to attack. We should therefore expect to have to spend some time uncovering the consequences of strong ordering constraints in dependency relationships.

Example 16 (Attack or lead by example?)**.** *If an imposition is made* ad hoc *before an invitation has been given, it may be assessed as an attack, with destructive consequences. On the other hand, if the imposition was expected, it may be considered an initiative.*

ATTACK	INVITATION
$S \xrightarrow{+\text{initiative}} \blacksquare\, R$	$S \xrightarrow{-\text{listen/talk}} R$
	$R \xrightarrow{+\text{listen/talk}} S$
$R \xrightarrow{?} S$	$S \xrightarrow{+\text{initiative}\mid\text{listen/talk}} R$
	$R \xrightarrow{-\text{initiative}\mid\text{listen/talk}} S$

In the table above we see the contrasting approaches between a blunt attack on the autonomy of the receiver R, and the route of diplomacy on the right. Through invitation, the sender begins by promising (up front) it's willingness to listen and talk by promising its acceptance (-) of any promise about talking and listening the receiver might offer. The receiver has now been invited to cooperate, without imposition. If it quenches that invitation by promising to supply dialogue (+), the sender can then promise initiatives, building conditionally on the prior acceptance of the invitation, because by the rules of Promise Theory, an initial offer of a (-) promise (an invitation) is accepted with another (-), making a (–) which is equivalent to (+)[12]. This is now a promise rather than an imposition, since the prior invitation was sought and the proposal was expected. The receiver may then accept the promise of the initiative, or it might decline it at its option. It has only promised to listen, not to accept, so there is no attack. We see how diplomacy is the use of promises rather than impositions to open dialogue between agents. The perception of a violation of autonomy is an important assessment in cooperative mechanics, and clearly plays a role in human assessments of trustworthiness.

Example 17 (Layers of invitation)**.** *Diplomacy is a process of homing in on a sensitive area by seeking invitations back and forth.*

'Would it be a terrible imposition of me to make a suggestion?'

'I invite your suggestion, based on your thinly veiled but humble meta-invitation.'

'Then I propose that ....'

'I accept your proposition, but reject part of its content....which needs revision.'

'I accept your proposal for revision of my proposal....'

And so on...

2.3.4 THE INTENT TO DISRUPT OR SABOTAGE

We can't properly discuss systems or bodies of systems without acknowledging that they exist within a larger environment of agents, some of which may interfere (intentionally

or otherwise) with the interior promises and its intended behaviours. Since the term *interference* has a special meaning in physics, I'll choose the term *disruption* for this kind of conflicting interaction.

Definition 26 (Disruption of intent)**.** *A promise, imposition, or the withdrawal of either, made by an agent D (called the disrupter), which renders a promise from S to R to be 'not kept', according to the assessment of some observer O:*

$$\alpha_O \left(S \xrightarrow{+b} R \right) \rightarrow \textit{not kept}. \tag{2.34}$$

Looking at the assessment in the box above, the Principle of Autonomy in Promise Theory (that an agent cannot make a promise on behalf of another) implies that disruption can only be classified into three forms:

1. *Disruption of the First Kind*: the agent S intentionally fails to keep its promise to R. Then $D = S$. This is a kind of self-sabotage, i.e. a deception to R.

2. *Disruption of the Second Kind*: the agent S withdraws or fails to promise observability to the agent O assessing the outcome. $D = S$. This is a kind of subterfuge, i.e. a deception to O.

3. *Disruption of the Third Kind*: suppose that the promise body is conditional on a promise of Δ a third party T, i.e.

$$S \xrightarrow{b \equiv X|\Delta} R \tag{2.35}$$
$$T \xrightarrow{+\Delta} S \tag{2.36}$$
$$S \xrightarrow{-\Delta} T \tag{2.37}$$

   where Δ becomes a condition for S to keep its promise of X, which fails to keep its promise, or attempts to impose a counter-intent. Then $D = T$. This is an external 'man in the middle attack' by D. In this case, we have:

$$D \xrightarrow{+\Delta} S, R, O, \ldots \tag{2.38}$$
$$S \xrightarrow{+X|\Delta} R \tag{2.39}$$
$$S \xrightarrow{-\Delta} D \tag{2.40}$$

   The promise Δ may be withdrawn by D if positive, or negated. Any change to Δ applies influence or disruption to the intended outcome. Equally, S may fail to accept this promise (2.37) (through no fault of T), but this is covered by disruptions of the First Kind. Extreme cases of disruptions of the Third Kind are:

$$\Delta \rightarrow \chi, \qquad D \xrightarrow{\emptyset \ldots \neg\chi} \blacksquare\, S, \quad X|\Delta \rightarrow \emptyset \tag{2.41}$$
$$\Delta \rightarrow \neg\chi, \qquad D \xrightarrow{\chi} \blacksquare\, S, \quad X|\Delta \rightarrow \emptyset. \tag{2.42}$$

The promise model is quite helpful in clarifying the roles of the agents in these interactions, without need for speculation. Disruptions can be caused at the source, by the receiver, or by intermediate agents. The duality rule, citing the equivalence of juxtaposing $\pm$ promises in a binding, implies that the same argument applies when the roles of sender (+) and receiver (-) are exchanged. Most of us tend to ignore the essential promise to accept:

$$R \xrightarrow{-b} S, \tag{2.43}$$

yet the receiver can disrupt a promise-binding by refusal to cooperate, as well as refuse to allow an assessor or auditor of the promise to access to its behaviour. The straightforward outcome of framing disruption in that we should be on the lookout for hidden dependencies:

Lemma 3 (Third party disruption implies dependency)**.** *Only a conditional promise can be disrupted by a third party, implying a dependency in the original promise body.*

2.3.5 ACCUSATIONS, THE IMPOSITION OF JUDGEMENT, AND TAKING OFFENSE

An issue related to the foregoing examples is the matter of the taking offense at remarks made by agents to others[13]. This follows immediately from the dual status of promises to give (+) and to accept (-). It is every agent's autonomous nature to form its own judgements, as well as to express them. However, as already established above, the imposition of one's judgement may be perceived as an attack, is made uninvited. This is clear when the framing of the imposition is itself unambiguous:

$$S \xrightarrow{+\text{You are a jerk}} \blacksquare\, R. \tag{2.44}$$

This is an attack, unless permission was granted by R

$$R \xrightarrow{-\text{You are a jerk}} S. \tag{2.45}$$

S's promises or impositions are also subject to intentional misrepresentation by R in the overlap between what is promises and what is understood. Suppose, S issues a declaration:

$$S \xrightarrow{+\text{G is a nice type}} A_?. \tag{2.46}$$

This is not an attack, since it is promised freely to an unspecified recipient, without any attempt to oblige a specific recipient R. Nonetheless, R may still use this to attack S however, by interpreting 'type' as a deliberately condescending term for a 'person'. This

is a consequence of self-duals in Promise Theory that interpretation can be an attack on an agent's utterance simply by imposing its own (different) interpretation. If the sender intended the phase as a playful and complimentary expression for a ninety year old G, then the conflict becomes an imposition. Similarly, the receiver might object to the meaning of 'nice'. 'How dare you call me nice?' Too much, too little? The art of taking offence or offense is a specialized form of attack by reverse proxy.

OVERT ATTACK	REVERSE HIJACK
$S \xrightarrow{+X}\!\!\blacksquare\, R$ $R \not\xrightarrow{X} S$	$S \xrightarrow{+X} R$ $R \xrightarrow{-X}\!\!\blacksquare\, S$
Intent imposed without invitation	Interpretation imposed without intent

2.3.6 DOES THE FINAL OUTCOME JUSTIFY THE MEANS?

In the previous example, we see how sequences of dependent promises (adjustments of intent) allow agents to navigate course changes of intent throughout an ongoing dialogue. This is *adaptive behaviour*[14]. In such a sequence, a frequent issue is whether the standard of behaviour of an agent keeps, in its promise-keeping activities, is itself important in achieving the outcome it promises or not. This theme will continue into the next example too. Promise Theory clears up some of the ambiguity here—and I'll return to these issues repeatedly in later examples throughout the book, especially in connection with the concept of agent *responsibility*.

Assuming that agents get through an opening negotiation, there remains the issue of whether a promised initiative b, that has been accepted, can be judged by only implicit criteria on which is is kept:

$$S \xrightarrow{+b} R \tag{2.47}$$

$$R \xrightarrow{-b} S. \tag{2.48}$$

Does the accepting agent R have the 'right' to insist on ethical or moral standards in the way S keeps its promise? Since rights are a social convention, the initial absence of rights for agents, in their default state, is to acknowledge that they are *a priori* autonomous. Indeed, the imposition of a standard of behaviour by R on S would be considered an attack on S, if imposed without prior invitation. Thus, unless S has made a promise about how it will keep its promise, as part of the bundle of promises it is making, then R has no invitation to judge its autonomous choices. Naturally, R is free to make its own assessments of S's behaviour and decline future initiatives, which is different from

imposing standards without invitation. When we criticize parties for unethical behaviour, we are therefore admitting that the initial negotiation of promises was inadequate: no promises were violated, rather too few promises were made.

This point extends to abiding by law too. Abiding by the law of the land is a voluntary choice, which governments try to impose by force, not always successfully. Laws are more effective when then represent majority choices that citizens basically already promise to accept. So a smart agent will not assume that agents will even obey laws unless this is stated in their promises. This is one reason why contracts are based in law, as a foundation for the stack of promises that lay the groundwork of diplomacy we can avoid having to renegotiate for every new interaction[15].

Example 18 (Pet projects)**.** *When is a project a 'pet project', e.g. a vanity project by an individual. Can a process allow unbridled freedom to explore any idea or approach? Should projects be democratic or unilateral? These are choices rooted in the promises made by all the individuals in a collaboration. Sometimes projects get hijacked by individuals for ideological rather than for rational reasons. That might lead to important innovation, and it might lead to an evolutionary dead end. Ultimately, Promise Theory makes a prediction: the downstream principle says that it's up to users to take the responsibility for inappropriate projects, by ignoring them in favour of something else.*

Example 19 (Hands on or laissez faire?)**.** *Are you a 'hands on' leader or a live and let die type? Do you impose methods and goals? Does being hands on mean imposition though? Or does it mean simply getting your hands dirty to be sufficiently involved that you have active knowledge and awareness of the situation at all times? You could be hands on, by invitation, and by inviting cooperation. There is nothing exlusive about hands on meaning being bossy or imposing on others. Similarly, 'micromanagement' is about how you deal with scale, i.e. level of detail, not about whether you impose or invite cooperation. The possibilities are not mutually exclusive—in a system, you find a way to make it work.*

Example 20 (Accountability)**.** *The issues of whether decision makers are accountable or not is a complicated one. Accountability is supposed to be a feature of 'democracy', but there are many versions of power structures we call democracy. The House of Lords in the UK, for example, plays a role of checking and potentially blocking outcomes from the House of Commons, in a sense, holding them to account. However, the House of Lords is not elected by the public, so its legitimacy is often questioned. Should we choose elite expertise, or common voting by all as the mandate for wielding decision-making power? To whom should we be directly accoutable? If A is accountable to B and B is accountable to C, does this mean that A is accountable to C or not? The intermediate*

agent law, in Promise Theory, suggests that it is not. On the other hand, the more agents involved in holding a leader to account, the less effective any of their beliefs, assessments, or positions matter or can be reconciled. Accountability therefore suffers from scaling issues, like everything else. Often, when we rile against a lack of accountability, we don't take those scaling issues into account!

2.3.7 CAN WE EXPECT AGENTS TO KNOW BETTER? THE PROBLEM OF COMMON KNOWLEDGE

The final addendum to this sequence of basic examples is the argument—sometimes made—that agents (people) S should know better than to behave in a certain way, which does not follow the same standard as that understood by the receiver R.

Can we say whether S's ignorance of R's required standard is willful or unintended? It would clearly be an imposition (attack) on S for R to claim that it should know the same standards it holds itself to. In legal settings, the burden of proof is whether there is reasonable doubt of intent. However, R may claim that this standard is common knowledge. This presupposes that there has been a promise by some agent to educate S, and that S has accepted that promise. R probably cannot know if S has been made the promise of education, or whether it was not kept by its source. If, on the other hand, that promise was made and kept, but S failed to accept it, then it bears the responsibility for its ignorance.

These are complicated matters, with an element of catch-22 involved. The simplest guide to agents to avoid negative assessments is to avoid imposing on other agents at all costs—or at very least without due process. One could go further and attempt to discuss majority and minority agents, their knowledge, and their promises. I'll leave that discussion for elsewhere. What's important is that we have straightforward principles that explain what an agent offers (+) and what it accepts (-) and how these interact in positive and negative ways.

2.3.8 DAMNED IF YOU DO, DAMNED IF YOU DON'T

The simple tenet of Promise Theory, explored in these foregoing points, is that we need both (+) and (-) promises to realize cooperation. This is not something that relates only to humans or only to machines—it's a property of processes on any scale, and it helps to illustrate how a common approach to both can be of deep and lasting value. The audience for promises and behaviours therefore play a major role in determining outcomes and assessing whether intent is well-meant or malignant.

Suppose an agent S makes a promise to a split audience:

$$S \xrightarrow{+b} \begin{cases} R_1 \\ R_2 \\ R_3 \\ \vdots \end{cases} \tag{2.49}$$

Next, suppose that these agents $R_1, \ldots$ not only can assess the keeping of the promise differently, but also may offer different levels of acceptance of it:

$$\left.\begin{array}{c} R_1 \xrightarrow{-b} \\ R_2 \xrightarrow{-\{<b\}} \\ R_3 \xrightarrow{-b|C} \\ \vdots \end{array}\right\} S \tag{2.50}$$

In other words, R_1 may accept b as given, R_2 may accept a partial subset of b (including none of it, or even the complement NOT b), R_3 may accept b conditionally, and so on. Even through negotiation, S may be unable to compromise to overlap with all recipients. Those agents will then judge S differently. This is the case in politics, for instance.

In such a case, there is no 'right answer'. The more sophisticated the interior decision processes of agents, the more likely such scenarios are to occur. It would be naive to imagine that this is a problem that can be 'fixed'. Any system designer would love to work with agents that do exactly what he or she imagines is required, but that isn't how systems work. Whatever choices one makes, there will be a mixture of positive and negative outcomes. The system designer can only address this by promising to design for perturbations of all kinds.

2.3.9 CONTEXT AND OPPORTUNITY FOR AGENTS

The ability for an agent A to play a role in a larger system depends on there being a place for that role—a potential recipient for the promise it can keep. In turn, the ability to keep that promise might depend on the promises kept by other agents. The general case is that of an intermediate agent whose opportunity is to join two parts of a workflow together with its own intrinsic additions.

Suppose an agent A promises a capability c, as a result of an available promise of b by some benefactor source S. This allows A to promise:

$$A \xrightarrow{+c|b} R \tag{2.51}$$

which it would not be able to do without:

$$S \xrightarrow{+b} A, \tag{2.52}$$

and which would have no potential benefit without the open invitation:

$$R \xrightarrow{-c} A. \tag{2.53}$$

Such contextual promises from neighbouring agents, upstream and downstream in a workflow, lead to opportunities for agents to contribute to a system and perhaps reap rewards. The concept of an opportunity is this naturally associated with a network context of promise providers and receptors.

Definition 27 (Opportunity)**.** *The bundle of beneficial promises surrounding an agent A*

$$\Pi_A(b, \ldots; c, \ldots) = \begin{cases} S \xrightarrow{+b} A, & \textit{(upstream)} \\ R \xrightarrow{-c} A. & \textit{(downstream)} \end{cases} \tag{2.54}$$

may be called an opportunity for A, such that it can now make a new promise:

$$A \xrightarrow{+c|b} R, \tag{2.55}$$

that was not possible before. Opportunity is therefore to be understood as an agent's embedding in a context of (+) and (-) promises, where there is a demand for its intrinsic capability c.

Definition 28 (Benefit to an agent)**.** *The benefit to A, in this example, is whatever value it received from the ability to promise $c|b$, where:*

$$v_A(\Pi_A(b; c)) + v_A\left(A \xrightarrow{+c|b} R\right). \tag{2.56}$$

With these definitions, an opportunity is not merely an abstract assessment of value to the recipient of a promise, but becomes a practical actionable increase in its ability to keep promises.

Example 21. *The ability to pay for supper $X|B$, if B is a promise of money.*

2.4 RESPONSIBILITY

Responsibility is an emotionally charged term. It is associated with liability, blame, and reprimand. There is a tendency to want to assign 'blame' to a person when faults lead to loss. That's a punitive interpretation of responsibility, and one that does not often make sense, unless willful malice or negligence were demonstrably at work. We need a more rational understanding of responsibility, based on the science of causation, to

actually stand a chance of making a difference. If we can step back from the idea of attributing blame, there is something to be learned from asking the question: which agency or agencies are in a position to be *able* to keep a promise, and hence *could* be considered responsible?

2.4.1 SUBJECTIVITY IN ASSESSMENTS

The assessment that a promise has not been kept is a subjective one: different agents observing the system might assess it differently. Their assessments can depend on context or circumstances; so how can we easily attribute a unique source to the perceived failure? This is the challenge of a distributed system with multiple stakeholders.

Example 22. *In a restaurant, a meal is ordered from the menu. One person enjoys the meal, the other doesn't. The latter (dependent agent) assesses the meal to not be what was promised. The former (fault tolerant agent) assesses the meal as acceptable.*

- *The agent that rejects the meal might be considered discerning of quality, but goes hungry and cannot work.*
- *The agent that accepts the meal eats and continues its work.*

Can a cause be attributed to the waiter, the head chef, the sous chef, the butcher?

Whether or not a promise has been kept depends on the kind of promise binding. As we know about promise bindings, the receiver or promisee has to make its own promise to accept what is offered; thus it shares responsibility in outcomes. Indeed, refusing to accept what is offered is the ultimate control decision of autonomous agents.

Example 23. *A doctor promises a patient: if you take these pills you will be cured. If the patient does not keep a promise to take them, then it will not be cured, and thus the responsibility for being cured lies ultimately with the patient, not the doctor.*

2.4.2 THE ROLE OF CONDITIONAL PROMISES IN POINTING TO RESPONSIBILITY

In promise theory, we track provenance or causation with *conditional promises*. Each promise is the responsibility of the agent who makes the promise (the promiser). From the conditional promise law, an agent making a conditional promise has not made a promise at all unless it also promises to acquire the thing its promise is conditioned on.

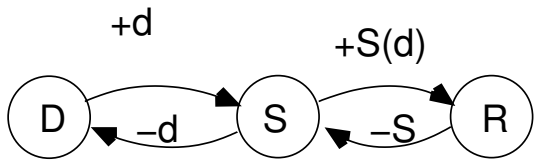


Figure 2.2: A dependency chain from upstream dependency provider D to a server S relying on the dependency, to a downstream recipient R.

Consider the scenario in figure 2.2

$$D \xrightarrow{+d} S \tag{2.57}$$
$$S \xrightarrow{-d} D \tag{2.58}$$
$$S \xrightarrow{+S(d)} R \tag{2.59}$$
$$R \xrightarrow{-d} S \tag{2.60}$$

where

$$S \xrightarrow{+S(d)} R \equiv \begin{cases} S \xrightarrow{+S|d} R \\ S \xrightarrow{-d} R \end{cases} \tag{2.61}$$

This system is fragile because the recipient has only a single choice. It has a single point of failure. The recipient could seek out redundant alternatives to provide the service S (as in figure 2.3). What happens beyond the horizon of the next agent in the chain of promise relationships is beyond the control of the recipient, and is thus beyond the limit any possible responsibility. Now consider the same scenario with redundant alternatives

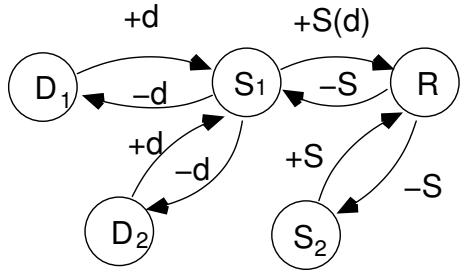


Figure 2.3: A redundant conditional promise chain, showing service delivery based on a dependency.

along the chain (see figure 2.3)

$$\{D_1, D_2\} \xrightarrow{+d} S_1 \tag{2.62}$$
$$S_1 \xrightarrow{-d} \{D_1, D_2\} \tag{2.63}$$
$$S_1 \xrightarrow{+S|d} R \tag{2.64}$$
$$S_1 \xrightarrow{-d} R \tag{2.65}$$
$$R \xrightarrow{-S(d)} S_1 \tag{2.66}$$
$$R \xrightarrow{-S} S_2 \tag{2.67}$$
$$S_2 \xrightarrow{+S} R \tag{2.68}$$
$$R \xrightarrow{-d} \{S_1, S_2\} \tag{2.69}$$

In this second scenario, both the server S_1 the recipient can choose from two providers of the promises they are trying to use. For the final recipient R, the fact that the promise from S_1 has a dependency is irrelevant, as there is nothing it can do about that except to acquire a second provider who may or may not have a dependency too. The only security the recipient R has is to have a choice of providers. No matter how hard the providers S_1 and S_2 try to keep their promises of service, unforeseen circumstances may prevent them from doing so. Indeed R may itself be negligent receiving their services.

This suggests that, while responsibility for keeping a promise lies with each source agent, only the final recipient (the location of the desired outcome) can be considered responsible for securing a successful promise outcome.

2.4.3 DOWNSTREAM PRINCIPLE

Locality offers a surprisingly simple and consistent interpretation of responsibility. The recipient of a promise carries the ultimate burden of assuring the outcome. In a chain of promises, dependencies are upstream (the source of the flow of influence) and the benefactors are downstream. Based on the consistency of responsibility described above, we can make the following straightforward observation. The assurance of the final promise outcome follows a 'Downstream Principle' that the most downstream agent has both access and opportunity to correct or absorb faults, and hence the greatest causal responsibility for an assessment of a promise not being kept. In other words, the greater the distance from the point of promise-making, the less causal responsibility an agent has in contributing to the outcome.

It's important to understand that the downstream principle is not a moral assessment, it is a purely pragmatic observation about cause and effect. However, it is interesting that it is in opposition to what is conventionally assumed about faults in hierarchies, as well as in Root Cause Analysis. The explanation for this apparent contradiction can be found

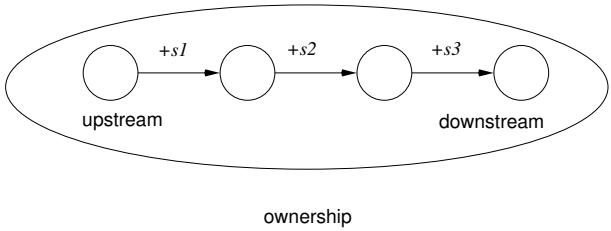


Figure 2.4: Responsibility for success in a chain flows downstream. The final responsibility is to use what is offered. An owner may formally accept responsibility for the whole chain, as a superagent, but at the microscopic level there is no other agent that can keep that promise. According to promise theory, the owner of the workflow should be the last interior agent to accept or 'sign off' on the work.

in the bi-directionality of promise bindings required for propagation of influence. The traditional assumption has been that influence is always imposed and that impositions always succeed, the latter being false.

Can we assert, then, that an agent who fails to use a promised service in order to keep its own promise is more responsible than the failure of the agent to provide the service? The user of the service could, in principle, seek a redundant alternative for such cases. But what if no alternative is available? If a promise is made conditionally, the agent (promiser) advertises a delegation of its responsibility to keep the outcome, by adding conditions. The promiser of each link in a chain of cooperation is responsible only for its own promises, i.e. the point at which it can effect change of behaviour or intent. This tells us that delocalization means divesting responsibility to others. This is the power of agent autonomy. Thus, in promise theory, responsibility is connected to locality. How does this compare to conventional component reliability theory?

- In the black box tradition of reliability theory, there is insufficient information to be able to attribute provenance for observed failures, so one can only attribute failure to a component itself. The failure of a component must be considered a random event, and the lack of information means that it makes no sense to assign moral blame.
- In Promise Theory, the assumption is that an agent that makes a promise is the only agency responsible for keeping the promise. It might be possible to argue moral or causal blame to the specifically documented promise. This is in keeping with the tradition of legal responsibility.
- By the conditional promise law, a promise that is conditional on another promise

being kept (either by the same agent or by a third party) is not a promise, unless the other promise is made by the same agent. This clarifies and documents diminished responsibility.

We can now attempt a limited but tenable definition of responsibility:

Definition 29 (Responsibility for keeping a promise)**.** *Responsibility has two main interpretations:*

- Causal responsibility*: When an agent relies on a dependency promise in order to keep its own conditional promise, causal responsibility refers to the agent's freedom to obtain a promised outcome by its own autonomous choice of interaction, especially in the presence of redundant alternatives.*
- Moral responsibility *(culpability) is a human assessment, about whether agent outcomes stem from good or for bad intent, hence it cannot be formalized except as a norm or in law. This is not a systemic issue, only a subjective assessment.*

An agent cannot be solely responsible for keeping a conditional promise. Could an agent that relies on a promise from another be culpable for a failure to find an alternative if the dependency fails to keep a promise upon which it relies? This might be considered a case of negligence, unless the agent's hands are tied by other considerations. The network of promises in which a agent finds itself ultimately determine these freedoms.

Example 24 (Supply chain)**.** *In a supply chain, agents acting as integrators, make new conditional promises based on the receipt of promised components. Each provider may promise (+) a specification. It's the responsibilty of the recipient (-) to oversee the quality and availability of what is delivered, and perhaps to seek alternative sources if the promise (+) is not kept satisfactorily. Attempting to sue (impose by attack) on is uncertain of its success in keeping the dependent promise chain.*

2.4.4 ASSUMING RESPONSIBILITY

Sometimes we hear leaders 'assuming responsibility' for their organizations. Here we are at risk of confusing responsibility with liability. An agent can 'assume responsibility' by making a symbolic promise. However this might not be a promise it can actually keep.

Causation implies that there are limits to what such a agent can really do. Promising to assume responsibility might be an unrealistic and symbolic promise, one that cannot be kept, since a single agent is potentially promising to be the single point of delivery for an entire organizational superagent. At best such a promise could only be conditional on

promises made by the other agents in the organization, and thus its reliability would only be weak. By the same token, judging or holding an agent responsible is an imposition that may not be reasonably accepted.

We can make a rough prediction, based on the foregoing discussion: the only case in which it is not redundant to impose blame on another agent S for an outcome, is when the agent R (in the role of a downstream user (-) or consumer of a service) has not promised sufficient redundant contingencies for its own role in the responsibility for the outcome. Blame is therefore used as a way of deflecting attention from missing for the lack of keeping one's own promises.

2.5 RIGHTS, PERMISSION, AND PRIVILEGES

Rights, permission, and privilege are all concepts related to the giving of promises. The concept of 'rights' is an important issue in human systems: e.g. human rights, right of way, right of settlement, right of refusal, access rights, etc). In society, rights are granted to access medical care, to enter a country with a passport, and so on. When these rights are granted discriminately, they may also be considered privileges. In human-computer systems, for instance, access rights are granted to only certain users to see certain files or use certain programs. These form a part of what we usually call 'security' or 'privilege'. There are subtle distinctions between the semantics of security and privilege: the former is about safety and protection, the latter is about rank and hierarchy—yet the two reduce, for all intents and purposes, to the same basic issues of promises. The offer of certain promises (or not) amounts to the management of boundaries and realms of access.

The terminology of rights is common in politics, often used with a deliberate intent to fire up emotions, to impose and to attack the agents—such as governments—who make promises to provide services, and who are interpreted as having greater privilege because they possess capabilities that may not be promised to all agents that seek to use them. Demanding rights is an imposition, thus we expect the seeking of rights and impositions to be related too. The widespread abuse of the terminology of 'rights' stems largely from the historical origin of moral righteousness, and the confusion between *permissions* and *capabilities*, which in turn stems from a tradition of deontic or obligatory thinking. Promise Theory makes simple and impartial sense of these matters.

2.5.1 RIGHTS DEFINED

It follows from the assumption of agent autonomy that agents can only promise capabilities they possess autonomously, and that they may not promise capabilities of other agents on their behalf[16] (see figure 2.5).

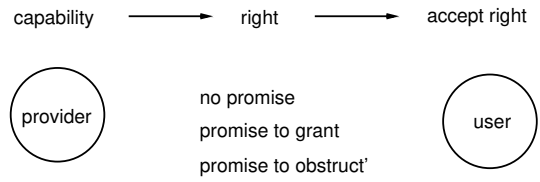


Figure 2.5: Rights are simply the status of promises about the capabilities offered by some agent. The absence of a promise X is not the same as a promise of its absence ($\neg X$).

This has the immediate implication that 'rights' are not innate and universal as sometimes suggested in political discussion[17]. They are offered as promises by an agent about matters it can keep itself. Rights, in other words, are not something that are innate or fundamental properties of a system, but rather they are promises that may be offered by capable providers, perhaps as part of a larger social contract.

Definition 30 (The right or permission to X)**.** *A promise, by a provider agent S to offer access to a resource or behaviour X:*

$$S \xrightarrow{+X} A_?. \tag{2.70}$$

This promised 'right' is not an obligation to accept, so the pre-requisite to a binding is captured precisely by the concept of a (+) promise. We may also speak of permission to behave in a certain manner. Permission is a promise of access. The semantics of permission are only subtly different from those of a 'right'. When seeking permission, one assumes that the promise $-X$ to use X exists prior to the matching offer of $+X$ which grants access to it. In all other respects permission and right are equivalent. Such rights or permissions may be granted specifically to a named individual, or may be granted to any unspecified agents. If the provider discriminates between its promisees, then an observer might assess different levels of privilege for those agents.

Definition 31 (The privilege of X)**.** *An assessment or promise of status attributed to one or more agents, based on its access to private resources. If an agent R is promised $+X$, i.e. is granted the right to avail itself of X:*

$$S \xrightarrow{+X} R, \tag{2.71}$$

In this case, we may say that R is granted X privileges. The promise of access acts as a label of privilege, which is used as a token of status. This could be made explicit as a promise of rank:

$$R \xrightarrow{+rank(X) \mid X} *. \tag{2.72}$$

i.e. given a promise of X, R may claim to be of rank X.

Again, the recipient granted or promised privileges is under no obligation to match the (+) promise with an acceptance (-). So the expression 'rank hath its privileges' is in fact the wrong way around: the promise of privileges are the foundation that define rank. Rights are therefore related to the notion of assisted promises, i.e. a promise enabled by the granting of access to another promise it depends on.

Discerning which agent starts with the capability to grant rights is not as easy as it might seem. When an agent contains a resource, it has a clear and practical capability to use it and to grant its use to others. The concept of *ownership* is a *virtual* human version of that idea. A resource doesn't have to be inside us to be under our virtual control.

Example 25 (Employee rights)**.** *According to the model of society, an agent may be said to contain and control a resource when it owns the resource. At the level of society ownership is a legal convention, different from merely holding a resource. The practical question of being able to grant access is different from the legal question of the right to control the resource a priori. The owner of a company is that legal owner. When the owner hires other into the company, they become a part of the agent, based on a relationship in which certain rights are mutually granted. This employment agreement forms the basis of what rights are granted to employees. This normally includes the promise of delegation of control over the company resources, else the workers are powerless to function. In a sense, there may be a conflict of interest between the legal right of ownership and the social right of participation implicit in employment, unless promises are carefully engineered.*

Workers' rights, in the sense of holidays and break times are not a right that can be granted by an employer. The employer can only grant its own resources: wages, free lunch, a room to have coffee etc. Only the workers can grant access to their time, whatever duress they might experience. The same applies, in principle, to machinery. If a

machine will not function, i.e. keep its promise to work, then an employer or owner can't do much about it. All such relationships involve back and forth promises. The mistakes managers and employers often make is to assume they have rights which, in fact, they don't.

2.5.2 SEEKING RIGHTS AND PERMISSIONS

Agents may not be granted the rights and privileges they need to make their own promises *a priori*. If they are unconstrained by other promises to search for providers of what they seek, they apply their downstream responsibility (see section 2.4.3) to secure their desired outcome. Agents without the freedom to search freely may want to demand additional rights from the agents they are clients of. This assumes that they have an expectation about what services can and should be promised to them by another agent (e.g. a government, an employer, etc). The basis for this belief could originate in any number of ways, and it could simply be a fortuitous search for a match, as in evolutionary processes.

Example 26 (I know my rights!)**.** *Agents may imagine or be programmed to respond to possibilities for which they have no matching promise.*

- *I have an electric charge and I am looking for a field to drive it.*
- *I have a hammer, and I am looking for a nail to complete it.*
- *I want the right to settle in Canada, and I think you should let me.*
- *I am lonely, and I demand a soul-mate.*

The rule of autonomy implies that they cannot demand compliance with these missing complements. They can only stand ready to accept promises offered to them from different sources.

Example 27 (Demanding one's rights)**.** *The attempt to demand rights is an imposition to use a resource*

$$User \xrightarrow{-X} Provider. \tag{2.73}$$

The strategy of demanding is likely to be ineffective since the provider may be unable or unwilling to comply. A strategy of invitation or search for alternatives is more likely to succeed.

A provider S only has the capability to deny another agent R access to X if it has interior private access to X, and R relies on other promises by S for its survival. We also say that the agent has the authority to grant permission (see section 2.6), meaning that it is the custodian of that privilege.

Example 28 (The right to free speech)**.** *Any agent R has an a priori* ability *to promise speech of any kind, unconditionally and autonomously. The question of a* right *to speech only arises when some agent(s) S, with which R interacts, imposes (threatens) sanctions to try to prevent an agent from exercising that ability. The implication is that R has already accepted a promise made conditionally on not exercising its capability for free speech from this other agent S, and this is being used as leverage—e.g. a promise that might be withdrawn, or a new promise which might harm R.*

$$S \xrightarrow{+food|\neg speech} R \tag{2.74}$$

The ability to speak freely thus precedes any 'right' granted concerning speech. The point is rather than the subject S has promised not to speak freely in order to receive promised benefits from S. In principle, R has the downstream responsibility to seek a replacement source for its needs if they are threatened. In common speech we commonly muddle this ability with language such as 'the fundamental right to free speech'. This is just imprecise language. The need for permission to exercise a capability one already has is an idea that can only arise in a cooperative framework in which voluntary abstention is practiced.

Rights—especially human rights—are something we often argue for, assuming that we deserve them. In Promise Theory, such a moral demand for rights is contrary to the autonomy of agents. There is also the issue of scaling, when freedoms become too invasive for a society (see example 73).

2.6 AUTHORITY, POWER, AND DELEGATION

The Oxford English Dictionary defines authority to be 'the power or right to influence others or act in a specified way'. Promise Theory tells us that there are two disjoint issues in that statement: respectively the 'power' and the 'right' to propagate influence—influence by command or by invitation. These two branches correspond loosely to the ad hoc *imposition* of influence, versus the *promise* of permission to wield influence, respectively.

Conventionally, we are more conditioned to think of the concept of authority through power. Most nations' legal stability is based on the threat of being able to overwhelm deviations from lawful behaviour by some kind of force—only later do such behaviours become norms and habits that require only simple maintenance. As a result, we perceive governments, bosses, and leaders in a historically authoritarian light (even the word has even come to take on a pejorative meaning, versus 'authoritative' which is more positive). In other usage, an authority is a source of singular expertise on a particular subject.

The unifying concept of authority is ultimately the appointment of 'trusted' agents[18]. An authoritative agent is one that promises to calibrate the definition of a kind of another promise on a particular subject. We define authority as follows:

Definition 32 (Authority for X)**.** *An agent which is the source of a promise with body X, and whose information is accepted by other agents, which all assess it to be authoritative on the matter of X. In other words, such an agent is a trusted party, according to the subordinate agents.*

The circularity of the definition reveals that authority is not an absolute property; it is only a self-consistent appointment and assessment, made on trust. The source is said to be authoritative, because it promises to be the final word on what is correct, and the subordinates accept that promise. That does not make it unique, as other authorities may have a different version of what appears to be the same promise. The recipient always assesses whether two promises are equivalent and whether the outcomes are compatible or not. An authority is thus a calibrating agent, in the language of section [BB14a].

Example 29 (Judges and supreme courts)**.** *Judges at authorities on the law, yet a panel of judges may still disagree about its interpretation. The law attempts to offer (+) clarity on certain situations, but that expression still needs to be accepted and received (-) by agents who view it from different contexts. Each judge is an authority, and the panel of judges can form its own authority as a 'supreme' superagent.*

The interpretations of authority, as power or right, may be sketched as follows:

- *Imposition*: an authority imposes influence on a subordinate:

$$\text{Authority} \xrightarrow{+\text{influence}} \blacksquare \text{ Subordinate.} \tag{2.75}$$

  The imposee of the imposition still formally needs to accept the uninvited imposition. If it fails to do so, one can still imagine that the imposer could assert its authority by force. For instance, if the imposer can conquer and subsume the subordinate, so that the agent becomes a part of it, then it can simply promise whatever it likes on its behalf, without violating any notions of autonomy. By absorbing the agent within a larger superagent boundary, its autonomy is lost as far as exterior agents are concerned.

- *Promise:* an authority can be accepted purely as a matter of voluntary cooperation, if granted a mandate M by a number of subjects to issue commands and make

decisions C:

$$\text{Subordinate} \xrightarrow{+M} \text{Authority} \tag{2.76}$$

$$\text{Authority} \xrightarrow{-M} \text{Subordinate} \tag{2.77}$$

$$\text{Authority} \xrightarrow{+C \mid M} \text{Subordinate} \tag{2.78}$$

$$\text{Subordinate} \xrightarrow{-C} \text{Authority}. \tag{2.79}$$

In this way, authority is a symbiotic relationship. Followers promise their support, the authority accepts that and uses it as a basis for making singular decisions conditionally on the mandate, which are then accepted by the followers. This is the basis structure used in democracy. The mandate M may be interpreted as the 'right' or permission to lead (see section 2.5).

In general, a coherent balanced symbiosis might have to be seeded by the imposition of force in order to stabilize a behaviour into a habit initially, which has interesting implications for statecraft and popular moral positions on leadership.

The concept of authority is also widely used, in its derivative meaning, to refer to management roles in systems, where it means the authoritative source of policy decisions—a manager or 'boss'. Authority over other agents is a derivative concept: the promise for which a manager or boss is authoritative is in calibrating policy for other agents to accept and follow—including the promise of wages, which it can withdraw as leverage. Following this agent's policy is, in turn, often assumed to be an obligation: however, based on the foregoing sections, the 'right' to impose commands as a single source of intent must be granted by the sources of its acceptance.

Definition 33 (Authority over other agents)**.** *An agent A to which a number of agents $\{C\}$ have promised to subordinate themselves, by accepting a promise of policy P, given a mandate M:*

$$A \xrightarrow{+P|M} \{C\} \tag{2.80}$$

$$\{C\} \xrightarrow{-P} A. \tag{2.81}$$

The common promise of $-P$ forms the appointment to the position of manager or boss. This appointment to take on the capability of deciding P is what makes the manager an 'authority'. The 'right to manage' is the promise to accept P.

Promise Theory tells us that such authority over other agents is not an inherent 'right' to impose upon them; the authority concept refers only to the integrity of its trusted information. However, the voluntary acceptance of imposition could be interpreted as a mandate (it plays the same formal role). The 'right' to issue commands and

directives must ultimately granted by the agent subjects, by accepting the policy—or be overwhelmed by conquering taking over the subject. This has the effect of subordinating the manager to its subjects in return for the privilege of being able to decide policy.

The implications of this are profound: unless a boss or manager's commands can be upheld by overwhelming brute force, or by threat (e.g. suspending wages), then it has to seek this cooperative mandate to play its role as manager[19]. If the manager does not control the policy about wages, then it does not have that leverage, and needs to maintain support by mutual cooperation. A manager or authority is therefore a *role by appointment*. The manager and subjects are actually coupled in a symbiotic state of mutual subordination, rather than a unidirectional hierarchy of subordination.

Partial authority may also be delegated by a manager to middle managers M, by conditionally promising to follow their decisions on a subset $p_M \in P$.

$$A \xrightarrow{-(p_M \subseteq P)} M. \tag{2.82}$$

This is a basic application of the matroid pattern from [BB14a]. The managers now have a mandate to decide matters $p \subseteq p_M$

$$M \xrightarrow{+p|p_M} C, \tag{2.83}$$

$$M \xrightarrow{+p|p_M} \blacksquare \quad A. \tag{2.84}$$

which now becomes an effective imposition on A It's interesting that, by delegating authority a single source of authority, at the top of a hierarchy, effectively places itself back into a subordinate role, having promised to accept the delegated (though limited) decisions of M, made by the delegates. It thus takes on risk—promising to honour whatever has been delegated. This assumes a further level of trust between the agents: namely that the delegate will not promise more than the authoritative would have done. Thus, with delegated authority by appointment, an organization is in a kind of equilibrium of intent, upheld by trust.

2.7 TRUSTED THIRD PARTIES AND WEBS OF TRUST

Trust is closely associated with promises and information. There are essentially only two distinct models for information distribution: centralization and *ad hoc* epidemic flooding. Alternatively one might call them, central-server versus peer-to-peer.

Example 30. *Two so-called trust models are used in contemporary technologies today, reflecting these approaches: the Trusted Third Party model (e.g. X.509 certificates, TLS, or Kerberos) and the Web of Trust (as made famous by the Pretty Good Privacy (PGP) system due to Phil Zimmerman and its subsequent clones). Let us consider how these models are represented in terms of our promise model.*

The centralized solution to "trust management" is the certificate authority model, introduced as part of the X.509 standard used in web authentication and modified for a variety of other systems (See fig. 2.6)[IT93, Rec97, HPFS02]. In this model, a central authority has the final word on identity confirmation and often acts as a broker between parties, verifying identities for both sides.

A central authority promises (often implicitly) to all agents the legitimacy of each agent's identity (hopefully implying that it verifies this somehow). Moreover, for each consultation the authority promises that it will truthfully verify an identity credential (public key) that is presented to it. The clients and users of this service promise that they will use this confirmation. Thus, in the basic interaction, the promises being made here are:

$$\text{Authority} \xrightarrow{\text{Legitimate}} \text{User} \tag{2.85}$$

$$\text{Authority} \xrightarrow{\text{Verification}} \text{User} \tag{2.86}$$

$$\text{User} \xrightarrow{U(\text{Verification})} \text{Authority} \tag{2.87}$$

To make sense of trust, we look for expectations of the promises being kept.

1. The users expect that the authority is legitimate, hence they trust its promise of legitimacy.

2. The users expect that the authority verifies identity correctly, hence they trust its promise of verification and therefore use it.

Users do not necessarily have to be registered themselves with the authority in order to use its services, so it is not strictly necessary for the authority to trust the user. However, in registering as a client a user also promises its correct identity, and the authority promises to use this.

$$\text{User} \xrightarrow{\text{Identity}} \text{Authority} \tag{2.88}$$

$$\text{Authority} \xrightarrow{U(\text{Identity})} \text{User} \tag{2.89}$$

One can always discuss the evidence by which users would trust the authority (or third party). Since information is simply brokered by the authority, the only right it has to legitimacy is by virtue of a reputation. Thus expectation 1. above is based, in general, on the rumours that an agent has heard.

Most of the trust is from users to the authority, thus there is a clear subordination of agents in this model. This is the nature or centralization.

Scepticism in centralized solutions (distrust perhaps) led to the invention of the epidemic trust model, known as the Web of Trust (see fig. 2.7)[AR97]. In this model,

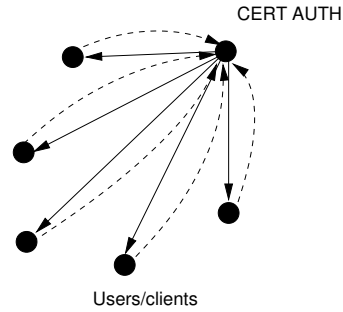


Figure 2.6: The Trusted Third Party, e.g. TLS or Kerberos. A special agent is appointed in the network as the custodian of identity. All other agents are expected to trust this. The special agent promises to verify the authenticity of an object that is shared by the agents. In return for this service, the agents pay the special agent.

each individual agent is responsible for its own decisions about trust. Agents confirm their belief in credentials by signing one another's credentials. Hence if I trust A and A has signed B's key then I am more likely to trust B. As a management approximation, users are asked to make a judgement about a key from one of four categories: i) definitely trustworthy, ii) somewhat trustworthy, iii) un-trustworthy, iv) don't know. An agent then compares these received valuations to a threshold value to decide whether or not a credential is trustworthy to it. The promises are between the owner of the credential and a random agent:

$$\text{Owner} \xrightarrow{\text{Identity}} \text{Agent} \tag{2.90}$$

$$\text{Agent} \xrightarrow{U(\text{Identity})} \text{Owner} \tag{2.91}$$

$$\text{Agent} \xrightarrow{\text{Signature}} \text{Owner} \tag{2.92}$$

$$\text{Owner} \xrightarrow{U(\text{Signature})} \text{Agent} \tag{2.93}$$

The owner must first promise its identity to an agent it meets. The agent must promise to believe and use this identity credential. The agent then promises to support the credential by signing it, which implies a promise (petition) to all subsequent agents. Finally, the owner can promise to use the signature or reject it. Trust enters here in the following ways:

1. The agent expects that the identity of the owner is correct and trusts it. This leads to a Use promise.

2. The Owner expects that the promise of support is legitimate and trusts it. This leads to a Use promise.

What is interesting about this model is that it is much more symmetrical than the centralized scheme.

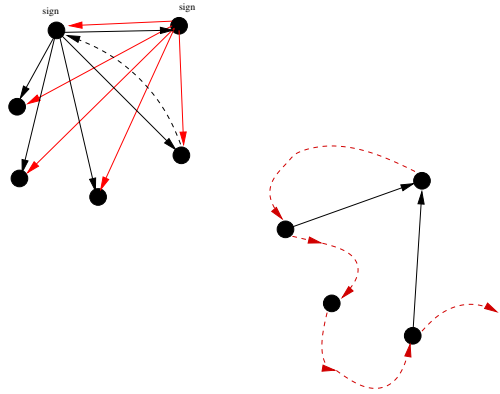


Figure 2.7: In a web of trust an agent signals a promise to all other agents that it has trusted the authenticity of the originator's identity. As a key is passed around (second figure) agents can agree by promising its authenticity, e.g. by signing it or not.

2.7.1 When is invitation preferred over command?

In a predominant part of Western culture the idea of invitation and voluntary cooperation is considered preferable to imposition and command, but this is not universally true. The preference for invitation assumes a level of mutual trust and mutual ranking. How agents perceive one another's value adds a conditional aspect to invitation.

Example 31. *In some cultures, an invitation might be perceived by a receiving agent as an act of weakness on the part of the sender. If the receiver has a low assessment of its own value, it might attach prestige to being commanded—the opposite of being attacked: I was chosen!*

Example 32. *If the receiver is an introvert, it may perceive an invitation as an attack, where an extrovert would consider it a compliment.*

Observing a cultural dependence of how agents make valuations on one another's promises suggests that we should be cautious of thinking about promises as unconditional in human systems. There may be many unspoken promises and intentions that an inviter or invitee are unaware of. It makes human interactions complex and fickle.

Example 33 (The impatient boss)**.** *Trying to exploit a commonly known promise (like the door being left open), which is not specifically directed to a particular agent, is unlikely to build trust or lead to that promise being kept with any urgency. The purpose of invitation is to build trust or perceived value without prior evidence. If we never needed to engage with an agent on a personal level, then any publicly known promise could be exploited by any agent at any time. For dumb agents (like machines or unsophisticated animals) this might be true, and might not matter much. But when agents are more internally sophisticated, as humans are, they may have other goals, and make prioritizations which are in conflict with keeping their promise to any particular. Human priorities and willingness to cooperate are based on extended relationships, as known from Axelrod's work[Axe97, Axe84]. An invitation is, in a sense, a way of 'hacking' a set of priorities, perhaps inducing a sense of obligation. In human settings, we might even consider an invitation harassment: it would be an imposition to assume someone could drop everything to keep a particular promise a leader desired. This is what we mean by being 'bossy'. The semantics of invitation are therefore subtle, and Promise Theory suggests that the key would seem to be to avoid impositions.*

2.8 HUMAN FIDELITY WITHIN A SYSTEM

The ability for humans to keep promises, depends on many factors. As agents in the system, humans may fail to work reliably relative to a situation or process, for a variety of reasons, some related to individual condition, and others to context:

- Management errors.
- Forgetfulness/carelessness.
- Misunderstanding/miscommunication.
- Confusion/stress/intoxication.
- Ignorance.
- Personal conflict.
- Slowness of response.
- Random or systematics procedural errors.
- Inability to deal with complexity.
- Inability to cooperate with others.

In system administration the problems are partly social and partly due to the cooperative nature of the many interaction software components. The unpredictability of systems is dominated by these issues.

Humans filter all communication through their own view of the world. We respond to things that make sense to us, and we tend to reject things that do not. This can lead to misunderstanding, or only partial understanding of a communicated message. It can be modelled as the projection of a signal into a digital alphabet that describes our limited domain of understanding. We match input to the closest concept we already know.

Unlike machines, humans do not generally use reliable protocols for making themselves understood (except perhaps in military operations). A system administrator or user can easily misunderstand an instruction, or misdiagnose a problem.

2.8.1 AGENT ANOMALIES: MACHINES AND HUMANS

Regardless of their role in a system, we can study what issues cause agents to act erratically, or with low process fidelity. Some characteristics are particular to the type of agent. Here are two ad hoc categories, commonly discussed:

- *Machines* tend to have relatively few parts. They are relatively simple, rigid and are therefore easy to stabilize/control with static or locking equilibria.

  Machines do not act with moral safeguards that humans take for granted, thus a machine would dispassionately kill a human being or bring down an entire system in order to keep a promise. Thus we can take less for granted when agents are working like machines.

- *Humans* have many moving parts that achieve balance through dynamic equilibria. Holding an arm up requires constant firing of muscle contractions, there is no locking mechanism.

  Humans generally need to perceive meaning in what they do, else they will not care about the outcome. Meaningless repetition is worse than meaningful repetition, with bespoke adaptation. There does not have to be a high degree of creativity involved, as long as there is a sense of fulfilling a purpose that matters to a person.

Of course, humans are also highly complex biological machines, so these are not unrelated categories: stereotypes are commonly misunderstood. In industrialization, we often ask humans to act as machines, suppressing thought and judgement for speed and efficiency. This leads to a mismatch of expectations.

Comment 2 (Discipline and constraint)**.** *Process rigidity is classical management/-control thinking, e.g. disciplining humans through forms, procedures and protocols. Machines or tools that apply strong constraints or safeguards are another example of discipline.*

In engineering we know that certain tools are fit for purpose, and others not so much. Asking humans to behave like machines may ultimately not help, in the long run, because imposed constraints are not self-motivated. Humans find ways around such mechanisms[20].

Repetitive work does not suit workers with low accuracy or fidelity, even with formulaic work, as repetition amplifies the probability estimator of error. Templates may be rendered incorrectly, with random errors, for example a simple probability of error can be amplified by application in batch $N(t)$, or by repetition over time T, giving a cumulative error of the form:

$$N_{\rm err} = \frac{1}{T}\int_0^T dt\ p_{\rm err}(t)N(t) \tag{2.94}$$

Thus, when dealing with agents whose fidelity is low, we can mitigate the errors and faults in a number of ways:

- Since the sum/integrand are purely positive, keep the working duration T short to minimize the number of errors.
- Reduce the probability of error $p_{\rm err}(t)$, and ensure that this is not an increasing function of time, due to fatigue, etc.
- Reduce the number of actions per unit time $N(t)$, to limit the exposure.

A simple way to solve all of these issues it to take the low fidelity agent out of the loop. The worst possible approach would be to connect a low fidelity agent to powerful amplifying automation.

2.8.2 FEEDBACK AND TESTING

- Remove positive feedback, and introduce negative feedback.
- Remove instabilities and non-linearities.
- Try testing at large N to detect flaws before production use (acceptance testing).

2.8.3 BLAME AND RESPONSIBILITY

The emotional reaction to loss, when a fault occurs or an error is made, often fuels a need to lash out and even seek retribution. When accidents and disasters occur, it has been common for survivors to initiate lawsuits and seek to blame scapegoats. People kick their machinery when it fails to meet expectations. Human error is often the attributed cause. Humans are an easy target. If we take any thread of cause and effect within a system, there will always be a human at the end of it, because humans are the source of all intent. Thus looking for a scapegoat often becomes a case of 'pick a card, any card' to make a pain go away like a magic trick.

This is a curious impulse, to try to alter the past instead of dealing with the future—indeed, one wonders what evolutionary purpose such behaviour might imply, if any. If we take a Promise Theory perspective on blame, there is an interesting correlation with the 'Downstream Principle' for responsibility in systems (see chapter 9). Blame is something that agents 'downstream' (or on the receiving end of a promise) tend to impose on agents 'upstream' (those keeping the promises), as the only remaining action after exhausting its own responsibilities. Thus, agents who do not seek redundancy, or place all their eggs in one basket, will lash out when the eggs break instead of going to the next basket, because they did not properly keep their own promises to accept the implicit responsibility for a reliable outcome.

One thing in humans' favour is their capacity to learn—though now we have made machinery which also has the capacity to learn. If such an agent makes a mistake, valuable experience is gained. The agent will be less likely to make the same mistake again, as long as it has proper value judgement. A simple replacement (an agent that did not learn the same lesson) could not have learned the lesson in that context—firing and re-hiring is a gamble, whereas learning from the unexpected is a 'work hardening' strategy. Thus replacing or avoiding a learning agent that commits an error is not a rational response. Replacing a faulty part that is incapable of learning might be rational if one knows that the same promise is likely to be broken again. In the search for emotional catharsis, we can make matters worse by increasing the likelihood of new faults.

2.9 THE HUMAN ACTION PERSPECTIVE

A common error that designers make about system design lies in assuming that all systems must work in the way that humans prefer to work: by exerting our will, like a level, as if a system were merely an extension of our bodies—with us at the centre of the universe, and everything else in orbit. There are sometimes advantages to this 'central command and control' idea and there are flaws. In biology, both approaches may be found

in species: from largely self-contained animals with brains to decentralized organisms like slime moulds and insect hives. This prejudice affects the way we design systems, and the way we judge and interact with them, and thus its reach into our expectations for system behaviour is long and subtle. It boils down to a choice between two extreme forms of organization and control that I like to call *brain models* (centralized thinking) versus *society models* (decentralized thinking).

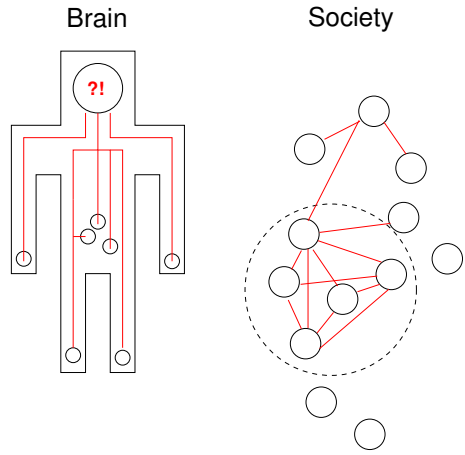


Figure 2.8: Brains versus societies. A brain model imagines a definite edge to a system. In a society model, the scaling is more arbitrary, but we can speak of super-agents with arbitrary boundaries, e.g. the dotted ring.

A *brain model* has a central controller that receives data from sensors and *imposes* onto or commands actuators from its central command (see fig. 2.8) out to the skin, sensors, and actuators of a finite organism. A *society model*, on the other hand, has no particular centre, and no particular edge. It works by cooperation based on *promises* made by agent-to-agent interactions. This is how insect colonies work. Brain models are centralized concentrations of reasoning and capability, used to drive a larger system from a localized component. Although I use the analogy of a brain, hearts and livers, and other organs also behave in this way, though they do not use the full potential of the brain model. Brains coordinate information quickly and analyze it in ways that societies can't, because of the close proximity of agencies with different specializations, and an assumption that they work faster than the systems they are controlling, with a fast communications bus to deliver messages back and forth. However, this means that brains scale by brute force to handle inputs and outputs. Both total load of information, and

the distance information has to travel play a role in limiting the scalability of control processes.

Example 34. *We sometimes say that people are too busy doing stuff to think. Or we could say that doing stuff and adapting by trial and error is a form of thinking, on a much slower timescale. We certainly learn by doing, but we can also learn more quickly and more superficially by reading cached summaries of experience in books and classes.*

The largest organism with a brain is a blue whale. It is quite slow, because of brain signalling and energy distribution of its cardiovascular network. Indeed, the larger centralized organisms get, the slower they behave, because the costs of communication and processing become an eventual burden[Wes99]. Insects colonies, on the other hand, react without centralization, based only on what policies they have cached in their DNA (effectively zero distance to travel, since there are decentralized copies throughout). Their colonies scale to a large size, before food (energy) becomes a limiting factor, but they can't easily perform the same kinds of analyses and decision making when they are sparse. These are the kinds of scales and processes we need to formalize and understand quantitatively to develop a proper understanding of systems.

We often expect that, by willing systems to behave as we would like them to, they ought to comply, because we are used to that mode of cause and effect in which we have an advantage of being able to observe and think faster than the system we are controlling. Ants or societies, on the other hand, do not work like that. Causation is subtle, and travels as if by Chinese whispers. The network has to converge towards some equilibrium at a single rate, without an external force to modulate it. What they can do with their tiny brains and simple manipulators is quite trivial, and yet the emergent behaviour of thousands or millions of ants leads to many stable and interesting outcomes.

Both kinds of system are limited by stability. Brain models are preferred because that's what we know. Societal models are hard to understand and they don't work from a position of obvious authority. In practice we need both to understand any non-trivial system.

2.10 HUMAN VALUES AND OPTIMIZATION

Human wellbeing and happiness are important aspects of systems that include people. If not happiness, then what? Duty? Obligation? Promise Theory tells us that duty and obligation might work for a while, but in general these methods are ineffective, both for machines and for humans.

Researchers claim that societies that are slower, less work intense, and less obsessed by money and progress, are happier than those that are entirely focused on

productivity[Lay06]. This bears consideration when designing societies in macro or microcosm, e.g. in forming families, tribes, companies, and in all workplaces. Mumford's quote, at the beginning of chapter 1, underlines how we can dehumanize societies—whose goal is after all to sustain human existence—and sometimes forget that the goal of what we are doing is to make the world better for everyone. Precisely what that means depends on one's point of view—it can lead to misalignments of tactical goals, which is a big problem for management.

Participation, making a difference, feeling useful and valued—these are themes that consistently pop up in human groups. Team organization is another[Bel09]. If we don't attend to agent wellbeing, human agents are unlikely to keep their promises. This is especially problematic where human qualities are paramount.

Example 35 (Working with external consultants 1)**.** *Many companies try to hire expertise in some form from external parties. This often results in failure. It might seem to be an invitation to pay a consultant, but if we invite them on false pretences, by suggesting that their work will be valued, when in fact they are sidelined, then the promises are broken, or were merely deceptions. Inviting a party and then abusing their (authority) mandate is a form of sabotage which is quite common. Managers who perceive themselves as above all others may find it hard to subordinate themselves to the advice of outsiders. An initivation to a consultant must involve a promise to listen and accept their input. All too often, consultants are simply installed into badly functioning social machinery, where they have no chance to keep their promises.*

Example 36 (Working with external consultants 2)**.** *I was once hired to work for a company with the understanding that I would work with a certain group. After starting the project, I was imposed on different group by an upper level project manager, without their consultation, who knew nothing about the project and had only passing interest in it. We had no relationship, and it was unclear who was leading the effort. Apart from showing up to meetings, none of the team ever contributed to the project and it was a complete waste of time and money. On realizing this, various attempts to pass on blame were toyed with half heartedly, but basically no one cared much because no one was really invested in the project.*

2.11 THE REMAINDER OF THE BOOK

This is not the only chapter about human systems, but it's the only chapter explicitly about human concerns. The rest of the book applies equally to systems made of the usual blend of human and mechanism, but you might have to read between the lines, or seek out the human-specific examples if you are only interested in human concerns. I think

that would be a mistake though—there are no uniquely human concerns. We are not as special as we like to believe. Particularly in industrialized processes, we work to suppress human qualities for reliability, through *discipline*. Discipline should not be thought of as a form of punishment, as sometimes implied, but rather as a constraint for seeking predictability. With those thoughts in mind, we are ready to confront the generality of promising processes.

CHAPTER 3

OBSERVATION, SEMANTICS AND ASSESSMENT

In order for any system to work, agents have to be able to interact. They effectively have to be able to sample one another's condition (as data observing interior states). This is what we mean by the ability to observe an agent. Observation is a fundamentally cooperative issue. Agents may try to be invisible—they have to promise to be visible if we want to be sure of detecting them. In IT, the granting of access rights, or establishment of controls and services may be necessary for this to occur.

3.1 OBSERVATION AND MEASUREMENT OF PROMISED BEHAVIOUR

Without the ability to observe systems, a model is purely speculative. That doesn't mean it is necessarily useless. Many models of the physical world have had useful lives in spite of being quite wrong[21]. Every interaction with another agent involves a form of observation: the receipt of some kind of information involves its observation by the recipient. Reception of influence and observation are inseparable.

We need to define the meaning of 'observer' somewhat carefully to make proper sense of phenomena in a discrete world. The characteristic scales of interacting agents play a strong role in deciding how much interior memory agents can be expected to have, and therefore how much information agents can absorb and recall about one another. Responses may be based on only a single sample (Markov processes) or they can be based on sequences or non-local phenomena (memory processes). In a computational

sense, observers need memory in order to experience phenomena in full and form a model of relative order, by knowing several identities for comparison (simply receiving one photon at a time, and immediately forgetting is not enough to observer physical phenomena). Memory also determines the extent to which observers can make use of implicit model-dependent concepts, like coordinates and relative positions, and which are not empirical phenomena, e.g. to measure velocity[22]. An agent with no memory cannot experience momentum, which requires at least two points to observe. This idea might surprise some readers. It's a hangover from calculus that we attach a non-local vector property to a point in space, by a limiting procedure. But that limit has no meaning in a discrete spacetime, so one is left with an outstanding issue: how do agents 'know' about directions?

A simple observer might be an agent that absorbs a message and changes state as a result (like an atom absorbing a photon). This represents one bit of memory. The nature of memory need not be defined for present purposes, but can be modelled as interior structure or sub-agents within bounded superagents, in a hierarchy of scaled structure. A crucial part of the scaling of agents from very small to very large is the distinction between processes interior and exterior to the agent. Every agent has a 'boundary' which defines the meaning of local for the agent[23].

Processes and promises are either inside or outside the boundary, something like Gauss' law[Bur15a]. One consequence of an information theoretic perspective on spacetime is that even elementary 'particles' must have interior processes in order to sample information from their exterior and absorb it. There is a minimum amount of structure an agent must have in order to support the kinds of promises it makes to the exterior world of other agents (figure 4.2).

Example 37 (Vector promises)**.** *In physics, an emission event or a collision where a vector like momentum is transferred has to be a process that occurs on such a scale that it is possible to represent a straight line vector. On the granular level of a discrete spacetime, paths are more likely to look like crack propagation, percolation, or even a lightning strike, than a Euclidean vector. Only on a large scale can we expect to be able to define a consistent and sustainable notion of direction.*

3.2 THE ROLE OF THE OBSERVER

How a system looks, and what it does, depends on how and where you look at it. It is a subjective sampling of the world. In any distributed system, we are forced to confront such subjectivities. Our aim is to do this in impartial way, using formal language that can be clear. This is one of the aims of Promise Theory[BB14a].

A system with a set of clear promises can use these as a calibrating 'measuring stick'. One can measure both qualitative and quantitative aspects of its behaviour against this set of promises. However, the point is not that promises are always kept. systems of people and machinery do not always work as planned. Outcomes can be thwarted in three broadly defined ways:

- Mistakes in the intended execution of a plan (errors).
- Unforeseen interruptions to behaviours that we did not plan for (faults).
- Flaws in the plan itself (ill conceived plan or changing our minds).

These three axes are not completely independent (some will even argue that it is impossible to define these at all), but I shall assume that they form a useful basis for approximating the issues and building models.

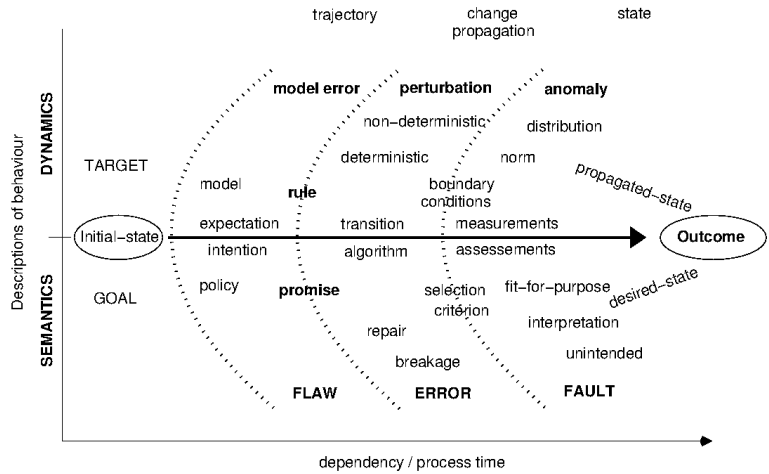


Figure 3.1: A mind map of descriptive causation, or the prediction lifecycle, comparing dynamics and semantics by analogy. There is a vast amount of related terminology in use. Predictability follows the arrow. Drift takes us into the realm of unwanted outcomes.

Context sharing leads to greater information and a more nuanced selection of promises appropriate to adapting to situations that arise. The desired-state stability of promises is what can prevent this adaptation from becoming chaotic.

Example 38 (Documented systems: situation awareness)**.** *There is a case for systems that can explain themselves to themselves as well as to others. Such systems expose their*

promises, enabling other agents to form expectations and adapt. A system that makes clear promises allows observers to be aware of the state of the system.

Very few systems are designed and documented in this way. Even worse, systems do not promise how they behave under all conditions and fail to behave predictably, because agents are unaware of either their intent or their behaviour or both. If technology is designed with clear promises, and an explanation of how those promises are kept (at all times), then it becomes self-documenting.

Systems are commonly measured ad hoc*, without a guiding scale of promises or expectations. In this case, we simply don't know what we are measuring or why. The concept of a Key Performance Indicator is like a promise assessment. Compare this to a random act of monitoring, such are systems.*

Example 39 (Taking for granted)**.** *We 'take things for granted' when no promise of a behaviour has been made, but an agent makes use of the behaviour anyway, without considering if it will persist. Promises are expensive to make and to keep, so it is statistically likely that the amount of information in our expectations exceeds the amount of information covered by our promises, hence there is a statistical likelihood for faults to arise through incomplete documentation of intent (specification).*

Such systems work by 'take it or leave it' behaviours, and the responsibility is usually left to human operators to sort of the resulting mess. This in turn leads to the generic conclusion of 'human error', as the result of a cultural heritage of treating humans as machines.

It is useful to add a definition of a system design:

Definition 34 (Design)**.** *The design of a system consists of an inventory of agencies and all the promises made, for each identifiable context, both from within and without, i.e. including at the system boundary with 'environment' and 'user'.*

A design that encompasses all agents and all promises may be called complete, else it is incomplete.

A design's success in keeping its promises may be evaluated to talk about its fitness for purpose. Proving completeness is a key issue, but it is generally impossible, as systems are open.

3.3 Mean Time to Keep a Promise

In an interaction, we need to characterize an agent's assessment of how long it took for a promise to be kept. This can vary from agent to agent, due to delays in propagation of action and observation. The notation:

Definition 35 (Time to Keep a Promise)**.** *The estimated timescale or sometime 'Mean Time To Keep A Promise' is an assessment which may be based on*

$$T_i(\pi) = \alpha_i^T(\pi) \tag{3.1}$$

Sometimes the notation $\Delta T_i(\pi)$ is used to emphasize a relative time interval rather than a timescale.

Example 40 (MTBF and MTTR)**.** *In fault diagnosis, one often speaks of the two-state model in which components promise either to be in working order or broken, i.e. in a state of failure or repair, where*

$$MTTR + MTBF = \Delta T_{total}, \tag{3.2}$$

where MTBF is the 'Mean Time Before Failure', when the system keeps its promise, and MTTR is the 'Mean Time to Repair' during which the system fails to keep its promise.

$$\begin{aligned} MTBF &= \Delta(\pi) && (3.3)\\ MTTR &= \Delta(\neg\pi) = \Delta T_{total} - MTBF. && (3.4) \end{aligned}$$

These are learned averages, as one can never promise precisely how long a repair or a failure will take to emerge.

3.4 CHARACTERISTIC SCALES: SPACE AND TIME

The study of behaviour relies on measurement, and measurements rely on our ability to observe and standardize scales. Scales nearly always depend on the ability to define a measure of time, because both durations and distances vary, but transmission rates are usually logically constant[24].

Scales are more complicated than in physics, because there is no standard one can assume for a measurement. Relative scales affect design choices like: is a process too fast or too slow for an agent involved with it? Should data transactions be as small as possible or as long as possible?

Example 41 (Designed to fail)**.** *Systems are often designed to fail by assuming they can cope with the effort to keep promises when they can't. They are effectively imposed upon to behave in way they could not promise to deliver, e.g. by pushing data at them without knowledge of their condition or capability.*

Promise negotiation between agents (contracts, if you like) has to adapt on the timescales of the processes as they occur to know what a single agent, or a collaboration

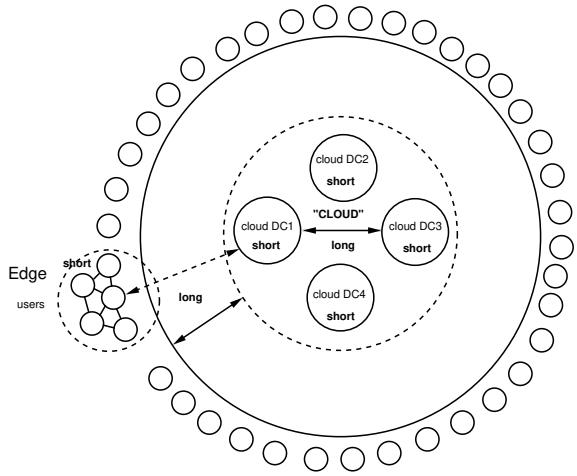


Figure 3.2: In a network, agents may be close together or far apart, in terms of the cost of communicating. This may or may not be related to geographical distance. On the interior of datacentres, effective distances may be short, while between datacentre and edge (where clients live) may be long. This plays a role in architecture, especially when large volumes of data have to be accessed and transported. In order to keep promises downstream capacity generally needs to exceed upstream capacity, else there will be non-deterministic transport—packets dropped or queues growing.

of agents can actually promise. This is a matter of scaling: identifying critical scales for a single agent, and identifying how collaborative clusters alter the capabilities of single agents or not.

Measurable dimensions in human-computer interactions fall into these categories:

Dimension	Measure	Description
Physical distance	metres	The effective catchment area of a service
Virtual distance	Hops	Routing legs.
Time durations	Ticks	Time intervals incremented by some counter.
Data size	bits	Payload during transmission or processing.

The speed of light in media is the only approximately universal constant in IT systems. Although it varies slightly as measured by exterior clocks, for computer systems, its value is effectively constant.

Wavelengths of light are the only physical distance measure of direct interest in the sense that they may determine the reliability of network communications and the rate

available. Also, wireless signals attenuate with physical distance. As geographical distances between clients and services increase, the catchment area for clients often grows as a function of distance, unless the service is private. Thus contention will grow with distance (see figure 3.2).

Most measurements boil down to local time comparisons, or latencies. Seconds are not the direct measure of time in computers. We can make an approximate conversion to seconds by using an independent hardware clock that is calibrated to count seconds.

3.5 CHANGE AND THE ROLE OF TIMESCALES

All functional systems, whether static or dynamic in nature rely on the concept of change to construct and maintain the system, on some timescale. No system is completely invariant for all time. Even the ability to observe change requires change in the agent sampling observations from a source. Change is fundamental to systems—the only question is how one defines the relevant timescales. Nearly all systems must have processes on several timescales in order to sustain their functional behaviours.

To define change , we need to think about *where* is the point of:

- Point of intent.
- Point of change.

To intend to change a remote point (one agent changes another) is a violation of autonomy, and requires either a promise of subordination or an outright attack on the remote agent by force. We also need to consider *how* change is defined:

- Relative to current state.
- As an absolute desired end-state,

Definition 36 (Self-change)**.** *A promise made by an autonomous agent A, to any other agent, concerning a change to its own internal resources:*

$$\pi \quad \rightarrow \quad \pi' \tag{3.5}$$

$$\left(A \xrightarrow{b} ?\right) \quad \rightarrow \quad \left(A \xrightarrow{b'} ?\right) \tag{3.6}$$

If $b' = b + \delta b$, the change is relative, else it is absolute.

Definition 37 (Remote change (subordinated))**.** *A cluster of promises and impositions made between a master autonomous agent M, and a slave autonomous agent S, in which S relinquishes its autonomy and promises to alter its internal resources, as directed by M:*

$$M \xrightarrow{b'} \blacksquare \; S \tag{3.7}$$

$$S \xrightarrow{-b'} M \tag{3.8}$$

$$S \xrightarrow{b \to b' | b'} M \tag{3.9}$$

In order to propose a relative change, the external master agent M needs to know the current state b of S, which must therefore be promised by S:

$$S \xrightarrow{+b} M \tag{3.10}$$

$$M \xrightarrow{-b} S \tag{3.11}$$

$$\tag{3.12}$$

To know whether the reported value of b is acceptable, there must be a notion of a policy or desired state B, such that $b \in B$ in required, then a $\Delta b = B - b$ must be computed for determine the imposition:

$$M \xrightarrow{\Delta b | b \notin B} \blacksquare \; S \tag{3.13}$$

$$S \xrightarrow{-\Delta b} M \tag{3.14}$$

$$S \xrightarrow{b \to b + \Delta b | \Delta b} M. \tag{3.15}$$

The existence of a B implies also the existence of a new timescale for change, which should satisfy the stability criterion:

$$T(B) \gg T(\Delta b). \tag{3.16}$$

3.6 THE PRINCIPLE OF SEPARATION FOR DYNAMICAL SCALES

The principle of separation of scales (especially timescales) is a design principle for interacting systems, based on the observation that dynamical influence causes timescales for change to mix. In earlier work, I've referred to this as the most significant principle for engineering—more important than the separation of concerns based on semantic (functional) separation, such as data normalization or 'class', which is the norm in Computer Science. Briefly, it says:

Principle 3 (Separation of timescales)**.** *Functional systems modularize robustly and effectively when processes with different characteristic timescales are weakly coupled.*

By 'robust', we refer to the dynamical 'stability' of the system. This principle makes a connection to the related problem of data consensus, which is a strong coupling regime that maintains data consistency over average timescales.

This principle is a dynamical principle, and therefore overrides any semantic benefits to dividing a system by function, rank, or type. The implication is that coupling processes, which operate on different scales, will lead to contention and inefficiency, while decoupling them will leave them free to optimize independently (from the first principle of autonomy). Coupling different scales tends to make responses non-linear, and introduces waiting and 'effective mass'. By separating a system based on scales, one keeps all the parts approximately linear in their responses. To understand this further, we need to define scales and coupling.

3.7 ON THE SEPARATION OF SEMANTIC SCALES

In computer programming it's common to separate concerns by writing code in a modular fashion. Different concerns represent regions of different semantics. Usually semantic concerns are separated, i.e. different conceptual or functional ideas. When these modules are combined and depend on one another, they become coupled. What criteria should we use to define modules?

Example 42 (Waiting for a slow transaction)**.** *Suppose a fast transaction needs to look up a name in a database name service. A transaction has its own characteristic scale, defined by how fast requests arrive. A name service has another scale determined by how fast the remote service can respond and be transported to the transaction. Requiring strongly that this lookup happens before the transaction can complete coupled a slow and a fast process together, making both slow.*

Example 43 (Separation of scales in parking and caching)**.** *Think of parking at airports or busy hotels. Parking is typically separated into short term and long term. Long term parking is farther from the airport, as it doesn't need to be accessed as frequently, meaning the total cost of accessing is minimized.*

In valet parking, customers simply drop their car at the gate and it is driven to a short term cache. If the car is not claimed soon, it may be moved to a longer term storage location. In this way, different caches are only weakly coupled, optimizing for timescale. In a similar way, separation of spatial scales is a related matter. One expects to use the cache that is closest to the point of access.

Example 44 (Separation of scales in cloud storage)**.** *In cloud systems engineers try to decouple computation from storage by handing off the weakly coupled intermediary services for data transfer. This idea can be repeated, in principle, in a hierarchy, analogous to the cache hierarchy in processors. For example, rigid metadata to drive fluid payload. Once committed to storage, the data integrity of the semantic content is the promise of the storage subsystem, as the database cannot promise the timeliness or integrity of data once it passes it on (no agent may make a promise on behalf of another agent).*

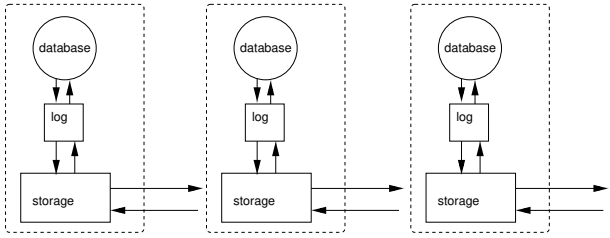


Figure 3.3: As an example of figure 1.2, a database system with a transaction engine, log store, and underlying storage has several components that are vertically coupled by dependency, and horizontally coupled by the need for data replication and disaster recovery.

Example 45 (Database dependencies)**.** *The role of the database front end is to handle concurrent connections from different causal sources, which may themselves be remote from the locale of the service point (see figure 3.3). For disaster and crash recovery, databases will set up clusters of replicas, which keep 'consistent' copies of the data. How long does it take for these replicas to reach equilibrium with the original source database, as measured an arbitrary observer's clock*[25]*? This is one of the key questions in so-called* data consistency*. Schema changes to the database format are also a common, if undesirable, occurrence. See the discussions in [VGS*$^+$*17, TDW*$^+$*12].*

Example 46 (Database replication cluster)**.** *The database cluster makes promises on a number of levels—can it keep these promises? A database's key promise is to ensure that data, once written, can be read back. A single instance of a database service can only make this promise conditionally on the promises of underlying storage being kept. In order to separate the slower timescale for writes from the possibly faster timescale for receiving transactions, it uses a log of changes that can be committed quickly and emptied as a queue. The success of this approach will depend on there being periods during which no transactions arrive (a bursty stream), else adding this extra step will only slow*

down the speed of committing data to storage. The cluster as a whole promises fault tolerance. Clusters generally try to promise results by quorum. A quorum is a critical scale above which data may be considered accepted by the superagent cluster. Before the quorum is achieved, data are still in the process of being accepted and diffusing through the network connecting the replicas.

Example 47 (Service with slow dependencies)**.** *Input/Output (I/O) is usually the bottleneck in data intensive services*[26].

Because the data are strongly dependent on the transaction log of changes, the database can be copied by copying only the deltas, in a compressed form. This can be therefore be decoupled from the vertical time for the storage writes to be handled, which may vary from location to location and are irrelevant to the leader. We cannot define the time for consistency.

We see, in this example, promises being made and attempts to keep them on all levels of clustering: amongst dependent functions (known as semantic scaling) and amongst replicas (known as dynamic scaling). An interesting example of this problem may be found in the optimized databases used by the Amazon Web Services cloud, see [VGS+17, TDW+12].

3.8 DYNAMICAL SCALE SEPARATION AND COUPLING STRENGTH

The interaction or separation of processes (discussed in section 3.6) is a question of the extent to which they are coupled. This cannot an all or nothing issue, because that would depend on the scale we consider. Average level of coupling could be high or low, either over time or by virtue of the mechanism by which coupling is handled. So for a scalable approach, we can use a more useful terminology or weak or strong coupling.

Definition 38 (Weak coupling)**.** *A weak coupling transmits influence from one agent to another without an immediate or precise response. It may be non-deterministic.*

In effective field equations, in physics, couplings are represented by energy transfer coefficients, and constants of proportionality between source and response.

Definition 39 (Strong coupling)**.** *A strong coupling is a rigid, close to deterministic causal channel.*

In a discrete graphical representation of a system, expressing a coupling as the strength of a proportionality is not always possible or appropriate. Using Promise Theory, we can

view it as a kind of probability that information passes from a source to a receiver, which is proportional to the binding between a promise to send and a promise to receive:

$$\text{Coupling strength} \propto (S \xrightarrow{+b} R \ , \ R \xrightarrow{-b} S). \tag{3.17}$$

These assertions can be justified by looking at what coupling strength means for interacting agents[27]. Phenomena that promise changes on very different timescales interact only weakly and can therefore be treated as logically separate. By contrast, agents that promise couplings on the same timescale may influence one another and therefore belong to the same class of phenomena. In terms of the foregoing definitions, we can state the meaning of separation more strongly, as a theorem:

Theorem 1 (Separation of causal influence)**.** *As the ratio of timescales becomes large $T_R \gg T_S$, the effective coupling tends to zero*

$$e \to 0. \tag{3.18}$$

tends to zero (weak coupling).

To prove this, suppose a series of partially ordered events at an agent S yields a series $E_\gamma, \gamma = 1, 2, \ldots$. Suppose a source agent S transmits the events, which are aggregated into superagents $E^{(n)}(E_\gamma)$ of dimension n, by the receiver agent R,

$$S \xrightarrow{+E_1, E_2, \ldots E_n} R \tag{3.19}$$

$$R \xrightarrow{-E_1, E_2, \ldots E_n} S \tag{3.20}$$

$$R \xrightarrow{+\alpha_E | E_1, E_2, \ldots E_n} A_? \tag{3.21}$$

so that the dimension of the information is reduced by a factor of n by R:

$$\alpha(E_1, E_2, \ldots E_n) \ \to \ \mathbb{R} \tag{3.22}$$

$$\left| E_1, E_2, \ldots E_n \right| \ = \ n \tag{3.23}$$

$$\left| \alpha_E \left(E_1, E_2, \ldots E_n \right) \right| \ = \ 1 \tag{3.24}$$

The average time between events, as assessed by R's clock, may be denoted

$$T_S \ \simeq \ 1 \tag{3.25}$$

$$T_R \ \simeq \ n. \tag{3.26}$$

So R assesses S's timescale to be 1 and its own timescale to be n:

$$T_R \geq T_S. \tag{3.27}$$

Thus the average interarrival times for the queue in (3.21) $\lambda_R \sim 1/\alpha_R$, etc, satisfy:

$$\lambda_R \leq \lambda_S \tag{3.28}$$

and the effective influence, in fraction of messages received compared to messages sent is expressed by a coupling constant:

$$e \sim \frac{\lambda_R}{\lambda_S} \leq 1. \tag{3.29}$$

In a strongly coupled system $e \to 1$, timescales converge to the shortest timescale of the interacting parts, making systems busier and more work intensive. The utility of this observation is that, if one separates causally independent parts of a system into superagents that make weaker promises to one another, any observed correlation between phenomena, that exceeds expectation, can be considered coincidental or potentially faulty. This can be detected by a change in the proper time event rate, measured by some agent within a system. This principle therefore has significance to the use of observation for detecting faults and design flaws in systems. It tends to maximize the signal to noise ratio between promised and non-promised behaviour[Bur4 b].

3.9 QUALITATIVE AND QUANTITATIVE ASSESSMENTS

Observers can make assessments quantitatively, if they have a scale against which to calibrate their observations. Quantitative measures are a special case of qualitative measures, in which we can map qualities onto a number line, e.g. the natural numbers, the real numbers, or even the complex numbers. A lack of agreement about calibrated standards makes all assessments effective qualitative.

Consider the basis for observer semantics. An observing agent $O \in \{\sigma\}$, in the scope of a promise π can an *assessment* of whether the promise has been kept or not[28]:

Definition 40 (Assessment)**.** *A 'decision' by a single agent O about whether a promise π has been kept or not, usually written $\alpha_O(\pi)$ for an assessment by O about π.*

The ability to assess and participate in observation and process implies that agents have interior capabilities. This is a 'turtle' paradox: in order to explain turtles, we need to assume 'turtles all the way down'. Mainly, this explains locality and local observation as a Shannon sampling process.

In Promise Theory an agent can try to make assessments about, say:

- Whether promises (to or from any agent) have been kept or not.
- The equivalence of agents (nodes or vertices),

- The equivalence of promises made between them (links),

provided it can observe them. In order to be observable, an observer has to have been promised access to the information (in Promise Theory parlance). This requires both an offer of information from the source and the ability and willingness to accept the information—both of which are expressed as elementary promises. The latter allows us to talk about states.

Definition 41 (The observable state of an agent)**.** *The observable state of an agent refers to an assessment of an exterior promise made by the agent.*

The interior states of an agent are unobservable. The exterior states are those that are promised to other agents, and can be sampled with fidelity. Interior states are unobservable.

Definition 42 (God's eye view)**.** *The observer is formally outside the system of agents and has complete information about the state and promises of every agent.*

Definition 43 (Local observer view)**.** *The observer is trapped within the system and can only observe what it has been promised, over channels that obey the constraints of cumulative individual agent interactions.*

The causal set relation is a global statement, so the definition of time in causal set processes is from a god's eye view—either within a locally inertial frame (coordinate patch) or for an entire spacetime. In either case, the definition does not solve the problem of scaling observer semantics—but, in quantum gravity especially, we would expect this to be a key issue. How do the limits of observation for a subatomic 'particle' differ from those for a macroscopic observer?

3.10 CHARACTERIZING PROCESS TIME

Agents and links between them have their own clocks. A single tick occurs whenever the information or state of an agent changes.

Definition 44 (Tick)**.** *A single distinguishable change in any observable variable.*

Example 48 (Ticks)**.** *Examples of time ticks are the arrival of events at a receiver, clock pulses generated by chips or devices, interrupts, etc. Any event that generates a response, with a certain probability, is a tick. If no response is generated, the change has been ignored and may as well not have happened, as nothing was advanced.*

As Einstein underlined, the only way to get a consistent picture of change is to measure time with clocks. Clocks at different locations may count time differently for a variety of reasons, and so we need to be cautious.

Definition 45 (Clock)**.** *Any observable process that accumulates and counts a series of ticks.*

Note that clocks may not count up to infinity. Counters usually wrap around after some finite number (say 12 on a wall clock). Time may therefore not be a monotonically increasing value.

Definition 46 (Time)**.** *The value of a clock, i.e. the outcome of counting distinguishable ticks into a distinguishable sum.*

If time does not increase monotonically, but a system has actually changed, then the clock may be called incomplete.

Definition 47 (Proper clocks and bad clocks)**.** *A good clock monotonically increases its count for time on observing every distinguishable configuration change promised by a system. It thus represents a proper time for the entire system. A bad clock is a clock that does not satisfy this.*

Most clocks are bad clocks over extended regions (indeed, this is Einstein's point in Special Relativity), because the clock may not be in step with the states of the system.

Example 49 (Human clocks)**.** *In the human world, we make no attempt to count changes, because changes are so regular that we assume one regular process can be used to represent all regular processes on average.*

In order to make a clock that's faithful to processes within a functional system, we have to limit its region of validity. This is what we mean by *locality*.

Lemma 4 (Proper clocks: single-valued time)**.** *If a closed system fully and completely returns to precisely the same state in a sequence, then the system has returned to the same time.*

The proof is trivially that the state of the system is indistinguishable from a state considered earlier by a 'godlike' observer outside the closed system, then it must be a finite state machine whose behaviour is cyclic.

We should always exercise an abundance of caution in thinking about distributed systems. Agent time, network time, process time, interior time, and exterior time need not be the same.

3.10.1 INTERIOR AND EXTERIOR TIME

Time is a count based on the result of racing different processes (clocks) against one another. This leads to

Definition 48 (Interior time)**.** *The count of ticks produced by the driver of activity within an agent. This inversely determines the process rate which sets the schedule for sampling of all data and computational steps.*

Definition 49 (Exterior time)**.** *The count of ticks received from exterior process sources as data, event by event. If this stream of ticks is used to drive the interior processes of the agent directly, the agent can never measure anything but a constant and regular rate of*

Definition 50 (Independent clock time)**.** *An exterior agent can maintain an interior count, and be used as a standard service that defines time.*

Variations in the rate of this clock cannot be measured without yet another clock, so its usefulness as a measure of time is limited. Nevertheless, services like NTP are used as we have many fast processes, like atomic oscillations, that can be used to count by. These minimize the problem of jitter in the relative meaning of a time interval.

Because the ability to sample exterior events depends on interior processes. Exterior time can only be measured using interior time, by Nyquist's theorem. Thus the rate of exterior time must be less than or equal to that of interior time:

$$R_{\text{exterior}} \leq R_{\text{interior}}. \tag{3.30}$$

3.10.2 EVENTS, CLOCKS, AND PROPER TIME

Every dynamical change is a process. A clock is a process that counts at a regular rate. This is, of course, a tautology. How do we know the clock counts at a regular rate? We need to compare it to another clock. In the end, we have to make an assumption about something regular. Since Einstein, our assumption has been the constancy of electromagnetic waves and the speed of light, for theoretical reasons associated with Maxwell's equations. In practice, we use oscillations of other kinds as reference clocks—but this leads to the interpretation of different frames of reference for different processes.

In an information theoretic sense, an *event* is an observation of change in data sampled from a source[SW49, CT91]. In the Einsteinian sense, this signal is a tick of a clock that an observer samples. When the tick originates from within a process (e.g. a CPU kernel tick), this defines a notion of 'proper time' for the local process, indicating

an advance in the state of the process. When there are multiple agents involved, working together, the language one often speaks of 'vector clocks' in IT[Lam78].

The concept of events plays a major role in the language used in nearly all observable processes. Let's define it here in a way that respects information theoretic transfer. Information is only 'arrives' somewhere when it is sampled.

Definition 51 (Event)**.** *A discrete unit of process in which an atomic change is observed or sampled.*

We often imagine processes being driven by a flow of events, like a stream[29]. Again, in terms of sampling, this amounts to the following:

Definition 52 (Event or message driven agent)**.** *An Event Driven Agent R makes a promise conditionally on the sampling of message events M from a S, with an average rate λ:*

$$R \xrightarrow{+E|M} \blacksquare\, O \tag{3.31}$$

i.e. R can promise an observer O that it acknowledges an event E on receipt of a message M. By Promise Theory axioms, this assumes the prior promises:

$$S \xrightarrow{+M|\lambda} \blacksquare\, R \quad \text{or} \quad S \xrightarrow{+M|\lambda} R \tag{3.32}$$

$$R \xrightarrow{-M|\mu} S \tag{3.33}$$

where μ is the queueing service rate.

Notice that by using the term sampling here, we do not take a position on whether messages were imposed by pushing from S to R, or whether R reached out to S to pull the data. These distinctions are irrelevant to the causal link that results from the message policy. Data are not received until they are sampled by the receiver. Note that there is no timescale implied by the conditional promise in (3.31)—the definition of 'immediate' or 'delayed' response is an assessment to be made by the observer O.

3.10.3 MISSED AND DROPPED SAMPLES

Observations inevitably get lost in any scientific enterprise. In empirical science this contributes to 'error bars' or uncertainties in counting of measurements—but not usually to semantics of interpretation. Interpretations are expected to be stable to such small perturbations.

Reasoning in IT has its historical origins from mathematical logic and precision: the avoidance of doubt. But doubt is a central part of tolerance in systems. If we observe

and inspect systems, we need to do so in the framework of a stable intent that overrides random fluctuations in measurement[30]. Monitoring and measurement serve no actionable purpose unless there is already a policy for behaviour in place. Ashby's model of requisite complexity or 'good regulator' in cybernetics[Ash52, Ash56] summarizes how matching information with information on the same level is required when there is no intrinsic stability in a model by which to compress such fluctuations.

3.10.4 DEFINITION OF CLOCKS

In a causal set model, time is a global construct, modelled as linked trajectories. The trajectory is the clock that measures time. This makes time non-local and ambiguous. Just how many states does the clock have? How is the count remembered? The x's are fixed global entities, without an obvious and unequivocal interpretation[Sor03, Dow08]. Spacetime is an independent construct, as in the Newtonian vision. Markopoulou criticized this construction and attempted to create a local interior view[Mar98], at the expense of introducing memory functions of different causal pasts, in the language of Category Theory. In so doing, she effectively anticipated the interior agent structure that Promise Theory predicts.

Einstein effectively pointed out in his theory of relativity that every observer may be associated with an interior clock the defines the rate of time for its processes. The significance of this is deeper than is usually implied—as one finds when one considers the implications for time in gravitational fields. Any effectively monotonic process, which changes detectably, is a clock. The rate of time cannot be defined without being measured absolutely, nor can

If follows that any agent must have its own standard interior time, which determines the rate at which it can sample its surroundings and participate in information (i.e. 'particle') exchanges.

Definition 53 (Interior time)**.** *An assessment of a change in an agent's interior state or promised behaviours is a tick of interior time. This is only observable to an exterior agent on some scale. Each detectable advancement of the agent's interior processes, including the sampling of exterior information, is a tick of an agent's interior time.*

Definition 54 (Exterior time)**.** *The assessment of a change in another agent's state or promised behaviours is a tick of exterior time.*

In Promise Theory, a tick of local time is an assessment of change—we assume that agents observe exterior ticks by the laws of universal information channels[SW49, CT91]. Proper time is counted by the collective states of distinguishable configurations on the

interior of an agent, so that exterior time is naturally filtered through an agent's interior processes[31]. The resolution of observable time is thus the overlap between exterior change and half the ticks on the interior of the agent (to account to double sampling of Nyquist's law). To an impartial observer, the rate of interior time must be at least as great as the rate of exterior time it can sample, by Nyquist's theorem. During the scaling of agents, by aggregation (figure 4.2), scaling converts exterior to interior properties, by moving the boundary between interior and exterior.

In a cause-set approach, a global time is defined as a partial ordering of event configurations. The nature of the configurations is not clearly defined. In particular, it's unclear if the configurations can be repeated (in which case this spells difficulties for local clocks), or if the expansion of spacetime states is built into the assumption of cause-sets. The difference between these approaches, which may yet be reconciled, is:

- *Causal Sets*: time is a partial ordering of configurations by a privileged observer.
- *Promise Theory*: time is what a local clock process counts.

3.10.5 DISTINGUISHABILITY (NON-LOCAL)

Agents may or may not be distinguishable to other agents, if they promise no properties by which to identify themselves uniquely. Even if agents do promise unique qualities, an observer may not be able to perceive the difference.

Definition 55 (Distinguishability)**.** *An observer O can distinguish two agents A and A', if and only if both agents promise names (scalar attributes),*

$$A \xrightarrow{+name=X} O \tag{3.34}$$

$$A' \xrightarrow{+name=Y} O \tag{3.35}$$

$$O \xrightarrow{-name=X} A \tag{3.36}$$

$$O \xrightarrow{-name=Y} A' \tag{3.37}$$

that are assessed to be unequal by the observer (i.e. may be promised conditionally to an unspecified agent $A_?$):

$$O \xrightarrow{X \neq Y \mid X,Y} A_?. \tag{3.38}$$

Distinctions can be observed within the same sample (spacelike comparison) or across subsequent sequential samples (timelike comparison) to detect change.

Lemma 5 (Observability of change depends on memory)**.** *In order to compute a timelike distinction, i.e. to measure a change from a set of states, an observer must remember at least the previous configuration. So observation of change is not a Markov process.*

The fact that promises are not always kept means that Observability cannot be treated as deterministic either. Observability requires both promises to be present and kept in order for information to be exchanged. Notice that, in physics we normally attribute properties like distinguishability of particles as bulk properties for bosons or fermions, etc. In Promise Theory, this is a property of the interaction between each pair of agents, not a global property that can be assumed for all observers.

3.11 PROPAGATION, DISTORTION, AND LOSS OF SYSTEM SEMANTICS

One of the results of promise theory is a better understanding of what we can expect from intermediate agencies within systems. Because any person, device or agency, which propagates intentions and outcomes between two parties as a relay or intermediary, is completely free to distort those intentions and outcomes, agents have a fundamental blindness when acting through proxies. They have to trust the intermediaries without question.

3.11.1 TAMPERING BY 'MEN IN THE MIDDLE' (SERIAL DISTORTION)

We can return to the details of this (known as the intermediate agent problem, or the end-to-end problem). Generally speaking, systems where outcomes are critical, should avoid proxies, or 'middle-men', as these are obvious places for distortion of intent and outcome. Each serial leg of dependency compounds the possible distortion.

Definition 56 (Tampering (serial distortion))**.** *Tampering is the imposition of a change on a system that is made contrary to a promised outcome.*

3.11.2 CROSSTALK OR CHANNEL SEPARATION

Agents that cannot discriminate between different contexts, in their assessments, can introduce errors by inaccuracy, or misinterpretation of context. Such low fidelity assessments, i.e. errors of promise utilization, and miscomprehension, can lead to catastrophic changes of intent, especially where systems amplify rather than tolerate errors.

Definition 57 (Crosstalk (parallel distortion))**.** *The proximity of two agents, making different promises to a receiver, may result in a low-fidelity recipient agent failing to discriminate between the promises, hence resulting mixing the outcomes.*

Example 50 (Wires crossed)**.** *Hitting the wrong button on a command console. Placing the intercom next to the ejector seat on an aircraft would be a poor design choice, since a small perturbation would have disastrous consequences.*

Alleviations are a general part of system fault-tolerance. They include greater clarity in the separation between contexts, according to the promised limits of perception for the agents who are the recipients. This might include visual and auditory channels for humans. Such considerations are a part of user interface design theory.

3.12 MEASURING AND ASSESSING OUTCOMES

To make reliable assessments, whether qualitative or quantitative, we need a kind of generalized coordinate basis, or a graduated scale of concepts, analogous to the coordinate axis along which one measures distances in mathematics. This need not be numerical; it could be a menu of semantics alternatives, i.e. what statisticians call 'classes'. Any agent can make an arbitrary assessment an outcome. It can also assess any other's fidelity for keeping promises. These assessments are a prior arbitrary, but we would like to do better. Assessments can be flawed: errors of measurement and interpretation are well known phenomena that draw attention to the fidelity of the assessing agents. With so many places for unintended distortion to take place, we begin to see the importance of tolerance and margins in system design.

3.12.1 PROMISES AS A SEMANTIC 'COORDINATE BASIS' FOR MEASUREMENT

No agent can assess another without a scale of reference, i.e. a measuring stick. If what was intended is not documented, then any outcome must be fair. How shall we measure purpose? The measuring stick we use is not only quantitative, but is a choice, like a vector marking alternative directions. Promises play this role: they form a spanning basis for the intended outcomes of each agent's behaviours.

Every agent that in in scope of the promises has the ability to assess whether they were kept or not, relative to their context, either inside or outside the system limits. Outcome is relative to its promised purpose. Promises thus form a simple formalized basis against which to judge the state of the system.

Comment 3 (Functional assessment of systems)**.** *Today it is common to add ad hoc monitoring of systems, with humans left as observers trying to divine meaning from the data. This can be useful as a process of scientific learning about the system, in order to gather motivation for policy, but it would be dangerous to act on such information without the prior motivation of a promise about what are considered acceptable limits on the measured values. Targeted testing is sometimes called acceptance testing or unit testing in software. This can be applied to any promise.*

Comment 4 (Ad hoc monitoring is unstable)**.** *Without a semantic calibration, we cannot know whether assessment is significant or not. Measuring or monitoring systems, without any promise of what to expect, leads to speculation about the significance of measurements. This can lead to divergence of opinion and unstable actions. Acting on an ad hoc assessment, without a promise about its expected behaviour within a larger model, could easily be disastrous, according to another agent's assessment.*

3.12.2 BASIS FOR ASSESSMENT: RELATIVITY AND SCALE

There are several parameters to consider in assessing outcomes. In any interaction, the transfer of intent depends on all the interacting parties:

- *Semantic overlap*: does the sender make a promise that is of a compatible type to that the receiver can accept?

- *Dynamic overlap*: does the receiver accept that the magnitude of the promise given by the sender satisfies its expectations? For example, an agent promising 6 kgs of apples would not be enough to satisfy an agent promising to consume $8 - 10$ kgs.

  Temporal overlap: is the promise kept within an acceptable time frame? How often should we sample the promiser to know whether the promise is kept. A promise that is too slow is the same as a promise not kept at all. e.g. promise to deliver same day, but delivery takes 2 days.

  Spatial overlap: does the right agent, in the right location, make the right promise?

In each case, we need an overlap in the basis sets of the promises to offer and accept something, i.e. is the type of the promise bodies b_+ and b_- the same? For example: if an agent accepts apples, providing oranges would not be acceptable.

3.13 OBSERVATIONS PASSED THROUGH TIERS AND STAGES

Throughout this volume, we shall have recourse to revisit a fundamental issue in distributed systems (in spacetime processes). That refers to the integrity of information propagated between agents, when agents promise to transport data with integrity through a number of stages that scale the process in different ways.

The scenarios all use a basic pattern in which some kind of source agent makes data available to a processing pipeline, or series of stages, and then some kind of recipient samples the data offered by the source. This process may be repeated in any number of stages to account for multi-tiered systems. From basic promise theory, we know that the basic transfer implies that two promises be kept:

$$S \xrightarrow{+X_S} R \tag{3.39}$$

$$R \xrightarrow{-X_R} S \tag{3.40}$$

with the effect of R observing the sample $X_S \cap X_R$. Each stage of the process depends on both ends of the link, i.e. on two causally independent (autonomous) agents, not on a single one. This issue is so widely misrepresented in the literature that it's useful to try to make a repeating issue out of it throughout the book. The implications of this are of great importance to a number of intimately related issues, e.g.

- Data consensus in databases (agreement about proposals),
- Data consistency in lookups (agreement about observations),
- Data transactions (reliable delivery),
- Data versioning (the state of change at a source),
- Centralization and decentralization of control,
- Aggregation of data across space and time,
- Measurement of time (the assessment of the current time),

and more. Figure 3.4 shows the basic configuration for the discussions. By referring to the arrangement in the figure, it will be helpful to compare issues

The figure represents a three stage process of data sampling, represented by sets S (source), R (replicas), and C (clients). Data are generated by agents S (e.g. sensors), starting on the left (also called upstream), and are sampled by agents R which store them temporarily. The R are redundant replica sets, i.e. data stores that contain copies of the data sampled from agents in S. Finally, agents in C may sample (query) the intermediate data stores in R to derive outcomes. The figure has a few features worthy of note:

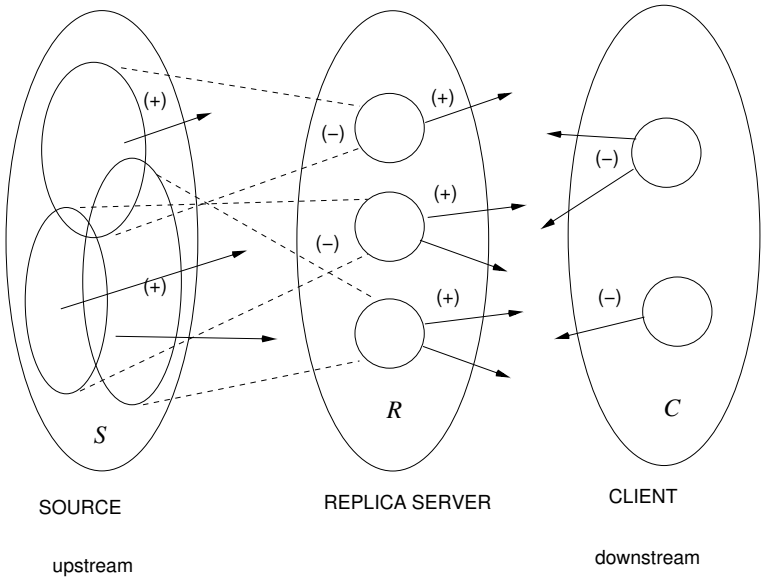


Figure 3.4: The basic configuration for sampling data through intermediaries. One the left, several redundant sources of data may provide parallel confirmation. This must be accepted by middle-men or intermediaries, which assimilate the information and pass it on to an observer. The observer downstream must in turn assess various opinions, possibly based on the acceptance of different incomplete sets of data and formulate a conclusion. This process is fraught with uncertainty.

- The set S of all sources may not be covered equally by all of the replica repositories in R. Data from some of the S agents may be sampled by more than one agent in R. Not all agents may be covered by a given agent in R. Some agents may be covered more than once.

- For some scenarios, R may represent non-overlapping 'shards' that promise to contain partitioned information that is separated by design. Partitions may be physical or virtual, i.e. by absence of signal (+) or by rejection of signal (-).

- For other scenarios, R may represent completely overlapping redundant replicas that promise to contain identical information.

- The client agents should be able to query data from R and get an answer. In different cases we may want:

- The latest result for a query.
- The fastest result for a query.
- Only results that are confirmed consistent by all replicas.
- etc.

In short, this figure represents a multitude of common cases that are applicable to information systems, whether computer, mechanical, or human. Some key use cases will include discussions of the topic of data models, sharding, command centralization, eventual consistency, and distributed transactions.

Example 51 (Quorum in replica sets)**.** *The standard approach, in computer science, for implementing decisions based on multiple redundant sources is to frame the problem as a vote, and seek a majority or quorum. This is a simplistic approach, which says: in the case of disagreements, correctness be damned, the loudest voice wins (democracy). If there is an even number of sources, there's a chance that there will be no winner, so one chooses to accept ony an odd number of sources, in order to guarantee that there will be no difficulty in selecting a majority. There is a numerical justification for this approach, if the value is known to have an invariant source value. For most cases, this is an entirely ad hoc approach for resolving a decision ambiguity, yet many system designers believe it to be 'correct'. The consequences of the decision are the responsibility of the decision-maker.*

Example 52 (Boeing MCAS tilt sensors)**.** *The fated Boeing 737 Max aircraft, which suffered two tragic crashes, implicated two redundant sensors, both of which were unreliable. A natural choice would have been to employ three sensors to simplify quorum. In practice, only a single sensor was used[BB19a].*

3.14 CONSISTENCY OF MULTIPLE SOURCES DURING OBSERVATION

As an immediate application of the picture in figure 3.4, consider the following example.

Example 53 (Counting apples)**.** *Suppose a farmer C, in an apple orchard consisting of many trees S_i asks his workers R to count the number of ripe apples. The outcome depends on a number of factors:*

- *The possible states of the system.*
- *The rate at which transitions occur from one state to another.*
- *The rate at which the samplers observe the trees.*

- *The rate at which the farmer asks the workers for their assessments.*

The workers may count different collections of trees (as in the catchment areas that comprise patches of S in figure 3.4), and report the number of ripe apples, or perhaps the fraction of ripe apples to scale the number accordingly. But how do they assess whether the states of the apple? the trees promise the information on a continuous basis, as the trees and apples grow, but if the workers sample sparsely, then they are likely to see a change from no apple to ripe apple in a single step (from 0 to 1). On the other hand, if they sample quite often, they may see several stages of apple, from nothing to proto-apple, to unripe, to ripe, and then fallen and rotten. The model of state they have for their observations depends on the overlap between the promise made by the apple to express change and their promise to receive that information:

$$\begin{aligned} X &= X_S \cap X_R \\ \text{`apple state'} &= \text{`apple growth timescale'} \cap \text{`worker schedule timescale'} \end{aligned}$$

If the workers have different schedules and sample at different rates, the promises they make to the farmer C will not only be quantitatively different but possibly qualitatively different. They may belong to different state models—all essentially a consequence of the different timescales or 'proper time' development of the agents in the system. When the farmer asks for a response from any of the workers, the answer he gets will be quite different for a number of reasons, even though the state of the apples has only one answer at the source.

- *The image of the source is only a sampling of the actual state.*
- *The sampling may be incomplete, because a complete sample is impractical or takes too long.*
- *Even if all the workers sample the same trees, they may report their answers at different times relative to when they meet the farmer's schedule (according the the farmers proper time clock).*
- *The sampling takes a finite amount of time during which the state may actually change.*

The possible options available in the system for trying to get a single stable outcome from the poll:

- *We can manage the sample sets of the trees so that all agents assess the same trees.*

- *We can manage the sample rates of the agents and sample very frequently. Then we shift an uncertainty in the change to an uncertainty in the kind of state, as we now assess there to be many more kinds of state in which the apple is ripe, e.g. a percentage.*

- *We can try to race the rate of change by making the reporting fast. The farmer then meets with the workers often too.*

- *We can try to get the workers to agree on what is the right number of apples by forming a* consensus*, or a* quorum*, before promising the result to the farmer. This makes their promise a conditional promise; it introduces dependencies and delays, during which they have to interact with one another as a cluster to arrive at an answer, All that time, the apples are growing and changing, so they need to be quick.*

- *If the farmer meets with a worker before they have reached a consensus, he may still get an uncertain result. The workers may therefore try to limit the farmer's access (limit the observability of their process outcome). They block or lock the process and prevent the farmer's proper time schedule clock from ticking. They effectively stop time for the farmer's process, relative to them.*

This list is not exhaustive, but it's already long enough. In each of the cases, the effort involved to get a better answer downstream to the farmer increases the expense of the process to obtain the result, and does not necessarily improve the certainty—it just shifts uncertainty from one issue to another. Ultimately it's up to the farmer (the client C) to deal with the uncertainty. This is called the downstream principle.

This problem characterizes a large number of data problems, where results are passing through a number of stages and the answer depends on the prior stages.

Some readers are probably thinking that the problem lies in the sampling. If the apples pushed their state to the workers immediately, and the workers pushed their state to the farmer immediately then the problem would go away. But that (very common) thinking is flawed. Assuming it were possible, it shifts the problem from a sample to queue processing. The data arrive at the workers as impositions from the apples, and may or may not be noticed by the workers, depending on their timescale. How many workers should forward their answer to the farmer? All or just one? Which one? In fact, we still can't guarantee that they will have the same answer at the same time, so at what point should they say their answer is complete and forward the result? If we increase their speed or capacity to receive data, they may respond 'synchronously' and report 'immediately' to the farmer. The results then land in the farmer's queue, but not in a predictable order. So even if the farmer gets woken up to receive the report 'immediately',

the state of the results is not known to him. Complete or incomplete? He may pick the first result, the highest result, or the lowest result. There is therefore a *policy* decision associated with the semantics of the data. He may even try to wait to see if the workers eventually come up with a consistent answer, or at least narrow the range (called an Eventual Consistency policy). All this time, the apples are growing and falling and the outcome promised by the chain of observation is only an estimate.

To define a 'correct' answer to this problem, we have to select from alternatives by recourse to a number of *policy* choices, for each of the issues presented above. The purpose of the policy is precisely to eliminate unwanted uncertainty by discarding it somehow. That doesn't mean the uncertainty doesn't exist because we are 'precise', it just means that we have designed a system to conceal the issues. This brings *stability* as a deliberate design choice, but not certainty.

We can now see how figure 3.4 applies to this picture. When we deliberately introduce stages into a system, adding multiple storage bins for data, each along paths with independent delays (latency), and independent sampling rates from source to database to observer the possibility for variance in the change process becomes large and we need to deal with that uncertainty. The culture in computer science and engineering is to assume that uncertainty is a bug to be fixed, and the try to engineer a workaround. This only shifts the uncertainty from one place to another, but it is common practice and it leads to many technological issues, such as database consensus protocols and the problem of distributed transactions.

Example 54 (CAP—Consistency, Availability, Partitioning)**.** *In computer science literature, there is often mention of what is colloquially referred to as the 'CAP Theorem'*[32]*. This is an observation that the issues of uncertainty are not independent of the ability to measure (availability) and the sample poll area (partitioning of the sample set S). The observation is trivial: when things are not independent, we can't arbitrarily get to improve everything at the same time. e.g. Distance is speed $\times$ time, so we can't change all three arbitrarily. Changing one has to change at least one of the others.*

Example 55 (Centralization or brain model)**.** *When data are aggregated from a number of sensors, it is impossible to know precisely when each of the sensory inputs occurred. The central brain's clock is what ultimately sees that signals, but these are delayed by possibly unequal amounts. In very large organisms, the time it takes for sensory signals to aggregate at a brain, and conversely for oxygen to be transported from a heart, and for nutrients to be transported from a gut is different for different parts of the body. This limits the rate at which the organism can react and move. In animals, brains compensate for time lag between different locations to present a coherent picture of the world, though this is hardly a precise. All signals essentially try to race a changing reality to the brain*

before it's too late.

Example 56 (Boeing MCAS Trim Sensor)**.** *The crash of two Boeing 737 Max aircraft was attributed to the failure of a software system that made use of a single sensor, i.e. a single point of failure. The planes were equipped with two sensors, but only one was read providing an automatic consensus even when the sensor was faulty and failed to keep its promise[BB19b, BB20].*

CHAPTER 4

PROCESSES

Although Promise Theory describes systems at the level of promises—an expression of an agent's state of intent—promises are useless without processes that can keep them. Thus, Promise Theory really concerns a theory of *processes*. Let's try to sketch out the issues that surround the description of processes—from technical issues to practical issues. In physics, one uses the idea of trajectories in a phase space; in computer science, one uses the idea of algorithms. We need to find the unification of these ideas.

Any system of change involves processes, both on the interior of agents to keep their promises, and on the exterior to enable inter-agent interaction. The basic channel of communication necessary for cooperation between agents relies on promises too. The most fundamental promises are therefore those that assure communication between agents. We begin by developing a simple formal language about promises to make contact with other mathematical disciplines.

4.1 PROCESS CAUSALITY AND ORDER OF EVENTS

The flow of activity is a system is directed by a sense of cause and effect. When the dominant interactions in a region are polarized in a particular direction, they lead to a net drift of change in the system, proceeding in a particular direction. When a change at one location is the precursor for a conditional change at a neighbouring location, the order of events is what we think of as a *causal order*. Classically, causal order is thought of as a partial order, but we know that—on a detailed microscopic level—dynamical order may contain loops and process details that differ in their directionality on different scales.

The key requirement for describing any process is the ability to distinguish key states from one another, and to be able to observe or characterize them in a serial order. It's

helpful to recall some basic definitions from mathematics in this regard. In mathematics, one is not concerned about locality—we are used to being able to impose restrictions as hoc, so such relationships have the status of impositions in Promise Theory. They are statements about all agents, without due deference to individual distinctions or locality. Nevertheless, let's start with standard definitions and then consider their local interpretation in Promise Theory.

4.1.1 THE TRANSMISSION OF INFLUENCE

The principle of causality can be stated simply by saying that earlier events are followed by later events, at a given point of action, as a result of the transmission of some information that we may call an influence.

Definition 58 (Causality)**.** *A complete graph of conditional promises representing a process. We say that S causes influences X at R with cause c iff:*

$$S \xrightarrow{+X_S \mid c} R \tag{4.1}$$

$$R \xrightarrow{-X_R} S, \tag{4.2}$$

and $X_S \subset X_R$.

If every transfer of influence in a system obeys this property, then we can define the system to be reversible:

4.1.2 GLOBAL EQUIVALENCE, OR SYMMETRY

In mathematics, an equivalence relation is a binary relation for a set X:

$$\forall x \in X, \quad x \approx x \text{ (Reflexivity).} \tag{4.3}$$

$$\forall x, y \in X, \quad x \approx y \text{ and } y \approx x \text{ implies } x \approx y \text{ (Antisymmetry).} \tag{4.4}$$

$$\forall x, y, z \in X, \quad x \approx y \text{ and } y \approx z \text{ implies } x \approx z \text{ (Transitivity).} \tag{4.5}$$

The set of equivalence classes is a partitioning of X: $\{x, \{y, z\}, \ldots\}$.

4.1.3 ORDER AND PARTIAL ORDER OF STATES

A partial order straightens out the loops by factoring out equivalences using a quotient construction. This can be considered a form of *coarse graining* of spacetime locations, and is thus a process in renormalization and scale transformation. On a large (classical) scale we might expect only partial ordering to be visible, while on a smaller scale pre-orders with acausal feedback loops play a key role in processes. This is certainly true of

other causal networks like electronic circuitry. Total ordering is a binary relation between agents, defined as a special case of partial ordering.

Definition 59 (Poset P with partial order relation $\preceq$)**.** *Let P be a set, $\preceq$ be a partial order relation on P (P is called a poset), and $=$ be an equivalence relation on P. Then:*

$$\forall x \in P, \quad x \quad \preceq \quad x \text{ (Reflexivity).} \tag{4.6}$$

$$\forall x, y, z \in P, \quad x \quad \preceq \quad y \text{ and } y \preceq z \text{ implies } x \preceq z \text{ (Transitivity).} \tag{4.7}$$

$$\forall x, y \in P, \quad x \quad \preceq \quad y \text{ and } y \preceq x \text{ implies } x = y \text{ (Antisymmetry).} \tag{4.8}$$

A total ordering adds an additional constraints, of mutual exclusion:

Definition 60 (Total order relation $\prec$)**.** *$\prec$ is a partial order relation for which*

$$\forall x, y \in P, x \prec y \text{ or } y \prec x \text{ (Connexity)} \tag{4.9}$$

and here 'or' means eXclusive OR (XOR).

These concepts are closely associated with the concept of distinguishability of elements. Note that the definition of order is absolute, not relative to a particular element of the set. Thus, partial ordering is a 'Newtonian' absolute god's eye view of a process. It's tempting to think that a partial order is the most primitive structure that supports causality, but this is too restrictive.

4.1.4 PREORDER

The most primitive kind of ordering relation is the concept of *preorder*, which is not quite a partial order as it can contain loops. In the context of the current discussion about causality, this implies the concept of acausal loops, which frequently arise in Quantum Field Theory and Promise Theory, so we should not dismiss these lightly. A preorder relation $\leq$ on a set S is a reflexive and transitive relation:

$$\forall x \in S, \quad x \quad \leq \quad x \text{ (Reflexivity).} \tag{4.10}$$

$$\forall x, y, z \in S, \quad x \quad \leq \quad y \text{ and } y \leq z \text{ implies } x \leq z \text{ (Transitivity).} \tag{4.11}$$

Note that it is not assumed here that $x \leq y$ and $y \leq x$ together simultaneously imply that $x = y$. The latter may be considered a coincidence limit of the order relation, as observed on a large scale[33]. This coincidence forms an equivalence relation on S such that $x \sim y$ if and only if $x \leq y$ and $y \leq x$ is satisfied. The quotient set $S/\sim$ is thus partially ordered (see figure 4.1). We return to this topic in section 4.4.1.

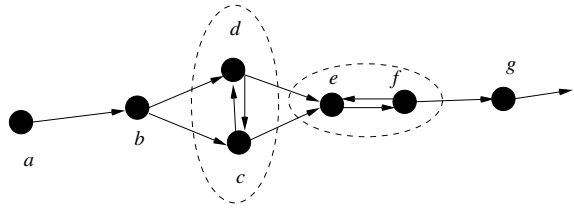


Figure 4.1: A pre-ordering turns into a partial ordering be rescaling (coarse graining) of the points. Representing the order relation by an arrow in the diagram, we have: $(a \leq b)$, $(b \leq c)$, $(b \leq d)$, $(c \leq d)$, $(d \leq c)$, $(d \leq e)$, $(c \leq e)$, $(e \leq f)$, $(f \leq e)$, $(f \leq g)$. The direction of normal time is from left to right (from a to g), but the process order has elements that defy that monotonicity, with minor loops. By forming equivalence classes and quotient sets to factor out these loops (dotted lines), giving a unique partial order.

4.1.5 CAUSAL SETS (NON-LOCAL)

A causal set is defined as a poset with the added constraint of local finiteness.

Definition 61 (Cause-set or causal set)**.** *A partially ordered set C (poset) with a constraint of local finiteness:*

$$\forall x, z \in C, \quad \text{card}\,(y \in C \mid x \preceq y \preceq z) < \infty. \tag{4.12}$$

Causal sets define an interpretation of a graph $\Gamma = (L, N)$, with links or edges L and vertices or nodes N, by identifying the partial order relation with existence of edges E and the member points $x, y, z \in C$ with nodes N.

Definition 62 (Link or edge (primitive))**.** *A pair of elements separated by a primitive countable instance or order: $x, y \in C$ such that $x \preceq y$, and with no intermediary $I | x \preceq I \preceq y$.*

As we'll see, a promise has more structure than this, which is both advantageous and assumes more, analogous to a total ordering. The distinction between partial order and total order will turn out to be related to scale and its influence on distinguishability.

Definition 63 (Chains, paths, and geodesics)**.** *A collection of points*

$$x_0, x_1, \ldots x_n \mid x_0 \preceq x_1 \preceq x_2 \preceq \ldots \preceq x_n. \tag{4.13}$$

The length of the chain is n, and the chain is called a 'path' if every chain relation is a primitive link. A geodesic between x_0 and x_n is a path between the same.

An important point to raise here is that there is normally only one *kind* or *type* of relation in a causal set model—represented by a single flavour of symbol $\prec$. This is a symptom of the preservation of separatism between spacetime and its interior processes. Yet we know of three distinct forces (perhaps four, though gravity seems better explained as spacetime structure) that distinguish events. For this reason, propagation cannot be defined within a causal set. At best we can identify a general indistinguishable potential for propagation with paths, as distinctions are unobservable. Using Promise Theory, this naturally gets extended to add multiple semantics, that can be associated with forces, matter, and energy typology.

4.2 INTERIOR AND EXTERIOR PROCESSES

Whether we consider agents to be atomic, i.e. indivisible entities, or composite entities (called superagents), formed from multiple agents with a boundary for encapsulation, an 'agent' on any scale has an effective interior and effective exterior. The interior or agents is always, *a priori* opaque to exterior observers. Any observable information must be explicitly promises by the superagent[Bur15a].

We know from the Shannon-Nyquist sampling theorem that each agent needs to have a sampling process of its own in order to accept signals from neighbours. An agent needs to be able to sample at least twice as fast as the source of a signal changes in order to capture its signal accurately. Since it is impossible for all agents in a system to be faster than all others, we have to accept a level of uncertainty in transmission of data. This uncertainty corresponds to the uncertainty in sampling, and is the origin of the Heisenberg Uncertainty Principle in quantum mechanics.

On the exterior of agents, i.e. between agents, a propagation of changes to the state of agents may be traced as a change in the exterior promises made by each agent to an observer on a larger scale. Notice that one can never escape the role of the observer in understanding behaviour. This question of the separation of interior and exterior processes, with implied boundary follows processes across all scales. It leads implicitly to the principle of separation of scales (see section 3.6).

4.3 LOCALITY AND PROCESSES

Promise Theory is a theory of cooperation between autonomous (i.e. causally independent) locations called agents. Agents are scalable by interaction and aggregation, but are also the atomic units of process. This doesn't so much explain spacetime processes as provides a clean way of modelling them, so there is some turtle stacking going on here too. Its formal definitions use the language of sets and graphs, but it can be used

quite intuitively at a purely symbolic level. Promise Theory is not about space or time, per se, but it describes configurations of entities that are spacelike or timelike, as well as capabilities or basic behaviours, which can be mapped to physical properties, like charges and boundary conditions. The result is something superficially like an Ising model, but without the Hamiltonian constraints of[KMS06].

4.3.1 PROMISES (LOCAL)

A Promise Theory treatment of process (i.e. of spacetime) will seem initially strange to physicists and mathematicians, used to the unequivocal pronouncements of mathematical relations, but the construction is quite natural from the perspective of scaling locality and relativity[34]. For example, the fact that A is next to A' (or specifically $A < A'$) need not imply that A' is next to A (or $A' > A$) in promise theory. Each agent has its own perspective as an observer, and as a source of expressible properties. In one sense, a promise agent begins in a state of completely broken symmetry, rather than of complete symmetry in the case of a manifold.

The primitive elements in a Promise Theory are *agents*, belonging to a finite collection $\{A\}$. Agents are assumed to be autonomous or a priori self-determined, which provides a clear definition of *local* that does not depend on assumptions about a light cone or a finite speed of signalling. Agents make promises, which are autonomously determined. A promise π is a primitive relation. The 'body' b of a promise defines the nature of the relation, written:

$$\pi : A \xrightarrow{b} A'. \tag{4.14}$$

An agent may make a promise to any agent, including itself–thus, promising is related to a partial ordering, but the autonomy of the agents introduces a notion of local relativity, unlike the god's eye view of the partial order.

Principle 4 (Autonomy)**.** *No agent can make or keep a promise on behalf of any other agent than itself.*

Since the relationship is directed (partially ordered), two roles are defined from $\mathrm{Promiser}$ to $\mathrm{Promisee}$, with body b as:

$$\mathrm{Promiser} \xrightarrow{b} \mathrm{Promisee}. \tag{4.15}$$

A promiser can make a promise to itself (for example an interior promise – see figure 4.2), but most promises are between one or more agents. A promise creates half a potential channel for information transfer.

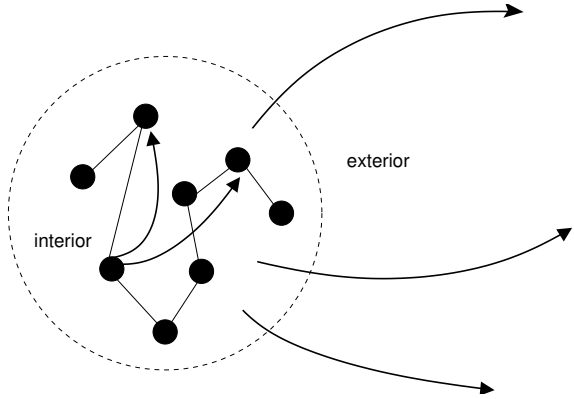


Figure 4.2: In order to act as a sampler of information (a message channel), an agent has to have an interior and and exterior at all levels. This is suppressed normally, to imagine an automatic transition, but this has implications about the nature of time. We sometimes refer to agents whose interior promises can be dissected as superagents. A superagent is just a scaled agent. Only exterior promises (from the outer boundary) are normally observable by exterior agents.

The question of distinguishability of agents is not better defined than for partial ordering at this stage. Promise bodies come in two polarities, denoted with a $\pm$ signs, as below. The $+$ sign gives origin, assertion, or offer semantics:

$$A_1 \quad \xrightarrow{+b_1} \quad A_2 \;\; (\mathrm{A}_1 \text{ offers } b_1) \tag{4.16}$$

while the $-$ sign gives projection (acceptance) semantics:

$$A_1 \quad \xrightarrow{-b_2} \quad A_2 \;\; (\mathrm{A}_1 \text{ will accept } b_2) \tag{4.17}$$

where A_i denote autonomous agents. The influence transmitted from A_1 to A_2 is oriented by the polarity, and has magnitude $b_\cap = b_1 \cap b_2$, and $b \cap \emptyset = \emptyset$. Thus, without mutual participation, information (an influence) can not be propagated from one agent to another. This is a local observer view in which each agent is fully independent—quite different from the global and absolute characterization in semantics for partial ordering. For example, in Promise Theory, the fact that a source is known to emit a message does not imply that the receiver will receive it.

4.3.2 PROPAGATION OF INFLUENCE

No influence has the potential to propagate from agent to agent without the mutual promises:

$$S \xrightarrow{+b} R \tag{4.18}$$

$$R \xrightarrow{-b} S \tag{4.19}$$

If a propagation occurs, we say that the promises have been kept, which amounts to an event. The $+b$ and $-b$ promises have roles analogous to emission and absorption of information across a causal channel. The structure is found in Shannon's channel model[SW49] and also in quantum theory of absorption[WF45]. We can interpret this basic unit of connectivity for potential propagation as a two-point function (two agent graph):

$$\Delta^{\text{ret}}_{SR} \equiv \left\{ \begin{array}{l} S \xrightarrow{+b} R \\ R \xrightarrow{-b} S \end{array} \right. \tag{4.20}$$

The symmetry between $\pm$ always permits an alternative *complementary* interpretation of a promise binding[BB14a], such that we can relabel $b \leftrightarrow \bar{b}$, and (4.19) may be reinterpreted as:

$$S \xrightarrow{-\bar{b}} R \tag{4.21}$$

$$R \xrightarrow{+\bar{b}} S. \tag{4.22}$$

This is analogous to the CPT symmetry in quantum theory, and advanced versus retarded boundary conditions.

$$\Delta^{\text{ret}}_{SR} = \overline{\Delta^{\text{ret}}_{SR}}. \tag{4.23}$$

The latter promise is acausal in one sense. The complementarity allows us to reinterpret this causally by lock and key fit. In order to assess propagation of influence,

- O assesses that the promise has been kept by S.
- O assesses that the promise has been accepted by R.

This further assumes that O is able to resolve the promises by S and R. This is sometimes represented by distinguishing the information about the promise using the notation:

$$O \xrightarrow{-\mathbf{def}(\pi)} S, \tag{4.24}$$

$$O \xrightarrow{-\alpha_R(\pi)} R. \tag{4.25}$$

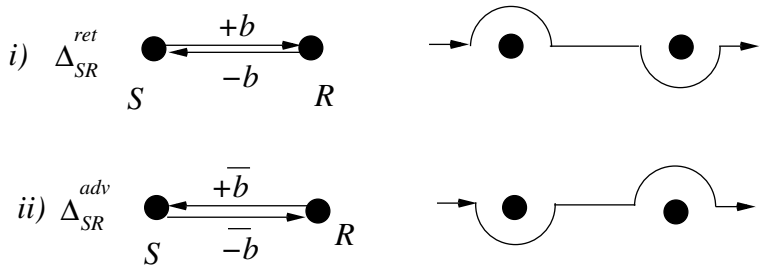


Figure 4.3: Complementary promise pairings require both + and - bindings to propagate information from a sender to a receiver. When the + and - promises match exactly, this reflects the equilibrium boundary conditions of the Feynman propagator (with contours drawn as mnemonics) in classical and quantum field theory, and the unitary symmetry $(\psi\dagger)\dagger = \psi$ in quantum mechanics.

which distinguishes knowledge of π it from the assertion that π will be kept. In general, this notation is too specific. If the we label the versions of the promise bodies by their originator:

$$\pi_S : S \xrightarrow{+b_S} O, \tag{4.26}$$

$$\pi_R : R \xrightarrow{+b_R} O, \tag{4.27}$$

$$\pi_{OS} : O \xrightarrow{-b_{OS}} S, \tag{4.28}$$

$$\pi_{OR} : O \xrightarrow{-b_{OR}} R, \tag{4.29}$$

then we need

$$b_{OS} \geq b_S \tag{4.30}$$

$$b_{OR} \geq b_R, \tag{4.31}$$

in order to observer with fidelity. A second kind of relation, known as imposition, may be written:

$$S \xrightarrow{b} R' \tag{4.32}$$

$$\text{Imposer} \xrightarrow{b} \text{Imposee}, \tag{4.33}$$

and invokes an exterior perturbation, like a boundary condition or other symmetry breaking action. While a promise is a locally autonomous but mutual binding of potential for action, an imposition may be considered an attempt to induce a response from sender $S \in \{A\}$ to receiver $R \in \{A\}$, without prior invitation (promise to accept).

In promise notation, the non-local ordering $x_1 \preceq x_2$, would be represented between agents as:

$$x_1 \xrightarrow{+\preceq} \blacksquare\, x_2 \tag{4.34}$$

$$x_2 \xrightarrow{-\preceq} x_1. \tag{4.35}$$

A promise model thus consists of a graph of vertices (*agents*), and edges (either *promises* or *impositions*) used to communicate boundary conditions or system 'intent'. A promise is a potential in the sense that promises are not guarantees. Agents are local and their interior circumstances and capabilities are opaque to exterior agents, unless promised. Thus an exterior agent may impose b onto and agent that cannot promise b. Even if the agent can promise b, it might not be able or ready to respond in any deterministic fashion. Thus, we must be concerned only with what each individual agents experiences locally, by sampling the channels of influence promised between agents.

So far, we've not specified a scale for A_i—imagining them to be elementary objects. Yet nothing is this description is particular to a scale, only to what autonomous capabilities the agents can promise. This is the route by which we can scale processes and shift from thinking about an absolute network to layers of virtual processes moving around on a statistical percolation network.

I want to point out that there is nothing in these notes that specifies nor attempts to specify the nature of the 'agents'. They could be quantum objects, atoms in a crystal, bees in a hive, or human populations in a social model. The relationships are based entirely on the notions of causal order, information, and scale. This is the virtue of Promise Theory's skeletal approach.

4.4 PROCESSES AS TRAJECTORIES IN SPACETIME

Our normal understanding of space is that processes (e.g. orbits) can revisit the same location at different times. In a Causal Set model, this cannot easily happen, so the purely causal model of events does not completely describe spacetime as we understand it. Einstein's view on phenomena was to be pragmatic: that which is not observed doesn't happen for the observer. A similar idea arises in Quantum Mechanics, as observation (wavefunction collapse) is disjoint from the causal part of the theory. In the past, philosophers have tried to mystify this by invoking human consciousness, but this is unnecessary. Even subatomic particles have to observe one another when they interact. The emission and absorption of a photon, for instance, involves the observation of a transition by both ends of the transmission channel.

Promise Theory takes a pragmatic approach here. Every change involves a fundamental act of observation (an assessment) of promises kept (events aligned according to the

semantics of the promises—which play a role something like 'potentials' in physics)—as in Shannon's theory of communication[SW49]. In other words, nothing happens unless it is observed by something.

This now has implications for the meaning of time. How is time counted in a process? In a Causal Set model, it is the path itself that counts time, but this doesn't obviously account for the normal semantics of a clock, in the Einsteinian sense. Assuming distinguishability of events x_i, the proper time of a process is the path travelled from a starting point. In order for an observer to know the time (even the order of time) it has to remember and be able to distinguish the elements of the path. This suggests (as in Einstein's relativity) that the clock lies equally with the observer and the source—not with the source alone.

An observer must have a memory sufficient to accumulate an image of the samples along a path in order to characterize it, and detect causation. The proper time of a trajectory may be effectively counted by stepping along the trajectory monotonically, but what exactly is this transition? How is the emission and absorption of a transition observed? Causal Sets have no obvious answers to these questions. In Promise Theory, we model the semantics by assigning each observer agent interior memory and counting capability that must follow the local mechanisms of information sampling. The overlap between what is offered and what is accepted: $\pi^{(+)} \cap \pi^{(-)}$ is a sampling operation, which itself is equivalent to ticks of a clock. This fits the notion of Alexandrov topology used in [DHS10][35]. Since we cannot separate this from the characterization of observables, this redraws the 'turtles problem' along pragmatic lines: as a problem in the emission and absorption of information.

4.4.1 CAUSAL AND ACAUSAL PROPAGATION IN EXTENDED COOPERATION

The existence of an initial source and a final receiver agent defines a boundary condition on the direction of causal order, which may contribute to some agents' counting of time. Classically, one assumes that time always flows in a single direction, but this is inconsistent with the local definition of clocks. Time may flow forwards or backwards relative to an observer's clock, but across different scales time (which records the advancement of a clock towards an outcome) may have as many directions as a clock has degrees of freedom, and influence can flow against the large scale direction of progress within any interior cell through feedback. Feedback loops that transmit influence backwards against the general flow of causation (see figure 4.4) are an essential dynamical characteristic that stabilizes dynamical behaviour. Such loops appear strange classically, but are common in field theory for virtual processes; and they are no more unusual than

a feedback loop in an electronic amplifier circuit.

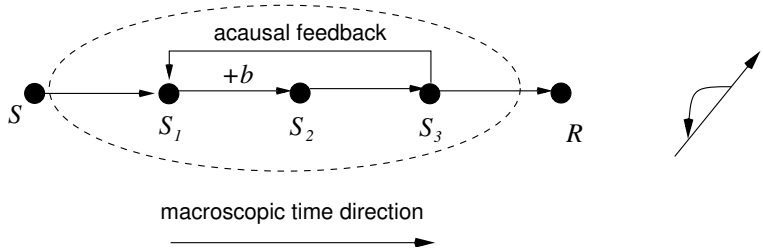


Figure 4.4: Causal and acausal are scale dependent concepts because the clock that counts progress to define the positive direction of time has a scale of its own.

From the basics of Promise Theory[BB14a], we have a complementarity law that for every promise $\pm$ there is an alternative formulation with opposite polarity $\mp$. Thus, acausal propagation has the appearance of a complementary promise in a causal direction. This is analogous to observation that particles and anti-particles in physics appear similar but time-reversed. These occur in field theory as particle processes oriented backwards in time, which may be re-interpreted as anti-particles travelling forwards in time, by the unitary conservation[Fey49]. Energy conservation is replaced in Promise Theory by the requirement of mutuality for complete binding. For every $+b$, there must be a $-b$ else nothing propagates. This leads to continuity of influence (which is the equivalent of Noether's theorem), so the association is consistent with conservation in physics, and suggests that the real meaning of Noether's theorem is about locality rather than conservation. Conservation is merely a side effect of locality.

4.4.2 THE MEANING OF ADJACENCY AND LOCAL ORDER

In order to establish which agents are next to one another, in a promise graph, we need to build a picture of mutual interaction. In Promise Theory, it is not automatically true that, if A_1 is next to A_2 then A_2 is also necessarily next to A_1. Spacetime can have one-way streets and mirrors. This is the price one pays for properly defining the semantics of locality.

By now, it should be clear that there is a difference between what may be perceived by a number of agents promising ordered relationships amongst themselves, and what a 'third party' observer can discriminate about the situation from a remote location. We need to explore this more formally. Let's explore the meaning of $A_1 \preceq A_2$ in an autonomous sense. In semantic spacetime, agents undergo connections to neighbours, which are the only locations they have direct contact with or 'knowledge of'.

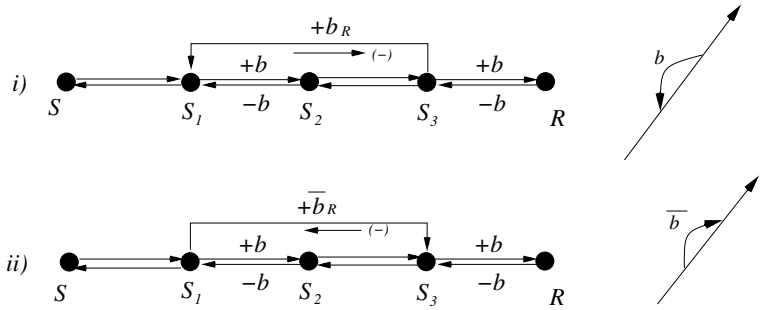


Figure 4.5: The detailed promise diagram for an interior feedback process, whose apparent acausal feedback may be interpreted as forward flowing anti-promises (complements), somewhat analogous to the field theory interpretation of anti-particle, and for essentially the same reason.

In principle the exchange of any kind of message is could be seen as evidence of proximity. We can take this a definition of adjacency: the ability to exchange a promise with a neighbour[36]. In an autonomous world, this is not a mutual promise.

Definition 64 (Adjacency promise)**.** *An adjacency promise is a promise made by an agent A_i to another specific unique agent A_j ($i \neq j$), about its claim to a local interpretation to a relative orientation i.e. direction between the two. We write:*

$$A \xrightarrow{+\text{adj}} A', \tag{4.36}$$

i.e. I promise that I am next to you, from my perspective.

The agent may, in addition, distinguish different kinds of adjacency $(x, y, z, \ldots)$, but we can drop this for now. Agents making an adjacency promises to more than one agent cannot simply be exchanged for one another without changing the linkage. Thus adjacency is the beginning of a form of local *order*, which may lead to a *Long Range Order*, by mutual cooperation[37]. Let us now examine how many primitive promises are needed to bind adjacent points in a spacetime.

Definition 65 (Adjacency promise binding)**.** *A bundle of bilateral promises, analogous to a contract, binds an agent A_n with another agent A_{n+1}, promising a channel between them.*

- *A_n promises that A_{n+1} may transmit (+) directed influence to it.*
- *A_{n+1} promises to use (-) A_n's offer.*
- *A_{n+1} promises that A_n may transmit (+) directed influence to it.*
- *A_n promises to use A_{n+1}'s offer (-).*

This can be interpreted as adjacency. In a forward (retarded) direction,

$$A_n \xrightarrow{+\text{adj}} A_{n+1} \tag{4.37}$$
$$A_n \xrightarrow{-\text{adj}} A_{n+1} \tag{4.38}$$
$$\tag{4.39}$$

And in the reverse (advanced) direction:

$$A_{n+1} \xrightarrow{+\text{adj}} A_n \tag{4.40}$$
$$A_{n+1} \xrightarrow{-\text{adj}} A_n \tag{4.41}$$

It feels strange according to the traditions of physics to think of being next to something as a promise, locally decidable by every location individually, but this is not as strange as it feels. An event horizon is a 'one way glass', for instance. In computer science, the adjacency of locations is a virtual decision made by processes to interact.

Notice that the mutuality of Newton's third law is not automatically guaranteed in Promise Theory: that which is given is not necessarily received (it's more like emission-absorption, which eliminates the privileged status of spacetime as being outside of the normal behaviours of physics); hence conservation of promised properties is not guaranteed, it must be documented with explicit promises just like charge. In this respect, familiar dynamical concepts of the continuum are puzzling from a discrete information perspective. Neither mass nor velocity are easy to incorporate.

4.4.3 OBSERVING CAUSAL ORDER

No agent is in a position to know its role in an extended region. This is quite like a cellular automation[t'H14]. Indeed, suppose an agent tried to promise an ordered relationship with its neighbour 'I am greater than you'. On what basis could it make this assessment.

Each agent is at liberty to make the same claim, so that

$$A \xrightarrow{+\text{greater than}} A' \tag{4.42}$$

could be countered with

$$A' \xrightarrow{+\text{greater than}} A. \tag{4.43}$$

This 'anti-ferromagnetic' or cellular stalemate could only be broken by a third party observer O with access to all of the agents. But this is not so straightforward either: now local names are no longer sufficient for O to keep track of the agents. O has no information about A_i unless the nodes are *distinguishable*:

$$A_i \xrightarrow{+\text{name}=A_i} O \quad (A_i \neq A_j) \tag{4.44}$$

$$O \xrightarrow{-\text{name}=A_i} A_i \tag{4.45}$$

$$A_i \xrightarrow{+A_i > A_k} O \tag{4.46}$$

$$O \xrightarrow{+A_i > A_k} A_i \quad (A_i \neq A_k) \tag{4.47}$$

$$\tag{4.48}$$

From this information, O is not able to promise an opinion, conditionally on the basis of what it has been promised first hand:

$$O \xrightarrow{+(A_i > A_j < A_k \ldots) \mid (A_i > A_m),(A_j > A_p),(A_k > A_q)} A_?. \tag{4.49}$$

Whether the agents A_i also make these order promises or impositions to one another is irrelevant to O, because (we assume, in the first approximation that) it only receives what each of the agents individually promises from them. If each agent acts unilaterally:

$$A_i \xrightarrow{+\text{greater than you}}\blacksquare \quad A_j \quad (A_i \neq A_j) \tag{4.50}$$

or if agents act by invitation (with prior affinity)

$$A_i \xrightarrow{+\text{greater than you}} A_j \quad (A_i \neq A_j) \tag{4.51}$$

$$A_j \xrightarrow{-\text{greater than you}} A_i \quad (A_i \neq A_j) \tag{4.52}$$

$$A_j \xrightarrow{+\text{less than you}} A_i \quad (A_i \neq A_j) \tag{4.53}$$

$$A_i \xrightarrow{-\text{less than you}} A_j \quad (A_i \neq A_j) \tag{4.54}$$

In order for this to work, there would need to be a common language understood through the interactions that matches "I precede you" with "I follow you" for each agent: a *gradient* promise.

If each agent also passed on overlapping information about its neighbours, then suddenly topology becomes a second order process—a statistical assessment made by every observer on a timescale greater than the timescale of adjacency interactions between the local agents (points). This implies that any observer capable of measuring relative positions is an agent equipped with sufficient interior memory as to form such a consistent map of assessments.

By promising its name (some distinguishable property), we ensure that the observer O can tell the difference between the promises it receives, without imagining a fictitious non-local coordinate system.

We see that our tacit assumption of a global manifold structure is far from easy to arrange by local properties alone. An agent capable to making observations compatible with a coordinate map must be of sufficient interior dimension to possess local memory accumulated from agents it is in contact with my messaging channels. Thus, any view of spacetime as a manifold is fundamentally non-local, from an information theoretic perspective.

4.4.4 CONDITIONAL PROMISES AND SCALING OF ORDER

In Promise Theory, the language for extended cooperation is the language of conditional promises. This begins with simple Markov processes, and may be extended to chains of 'end to end' promises with coordination throughout, to strong notions of entanglement familiar from Quantum Mechanics.

Definition 66 (Conditional promise)**.** *Let b be the promise body, and C be a pre-requisite condition that must be promised and kept before b can be kept, then we denote the promise of b given C also by*

$$A_1 \xrightarrow{b|C} A_2. \tag{4.55}$$

The quantities b and C are set valued. A conditional promise is not a true promise. It advertises no information unless the condition itself is promised. The conditional promise law states that:

$$\left.\begin{array}{l} A_1 \xrightarrow{b|C} A_2 \\ A_1 \xrightarrow{+C} A_2 \end{array}\right\} = A_1 \xrightarrow{b} A_2, \tag{4.56}$$

i.e. an agent that also promises the condition is the same as making the unconditional promise. However, a promise made conditionally on a promise being kept by a third party has a different resolution. Suppose the condition C is promised by A_3, then because A_1

cannot make a promise on behalf of A_3 it cannot promise A_2 that it will receive C, it can only signal its intent to rely on C by using

$$\left.\begin{array}{l} A_1 \xrightarrow{b|C} A_2 \\ A_1 \xrightarrow{-C} A_2 \\ A_3 \xrightarrow{+C} A_1 \end{array}\right\} \approx A_1 \xrightarrow{b} A_2, \tag{4.57}$$

The use of the equivalence relation $\approx$ now signals that these two scenarios are not completely the same. The effect is the same semantically, but the probability of the outcome being delivered may be quite different.

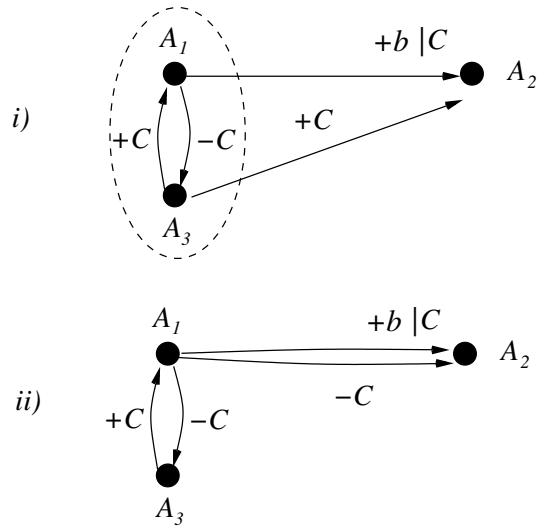


Figure 4.6: Two ways of resolving conditional promises. In i) the resolution of the conditional is equivalent to the scaling of a single agent; the observer perceives $\{A_1, A_3\}$ as a single entity in ii) the conditional action is relayed by an intermediate agent which has no direct adjacency with the recipient, in the manner of a Markov process, leading to a different assessment of the certainty. This tells us that Markov chains do not scale invariantly with respect to external observers—greater cooperation is required to preserve promise-keeping semantics. This indicates that the route by which information arrives at the observer A_2, as one would expect for scaling of locality.

This is relevant to the scaling of an agent. If we consider the collaboration between

A_1 and A_3 to be a single superagent 'molecule' then $A_{13} = \{A_1, A_3\}$, and

$$\left.\begin{array}{l} A_1 \xrightarrow{b|C} A_2 \\ A_1 \xrightarrow{-C} A_2 \\ A_3 \xrightarrow{+C} A_2 \\ A_3 \xrightarrow{+C} A_1 \end{array}\right\} \approx A_1 \xrightarrow{b} A_2, \tag{4.58}$$

which reduces to

$$\left.\begin{array}{l} \{A_1, A_3\} \xrightarrow{b|C} A_2 \\ \{A_1, A_3\} \xrightarrow{+C} A_1 \end{array}\right\} \approx \{A_1, A_3\} \xrightarrow{b} A_2, \tag{4.59}$$

and on the interior

$$\left.\begin{array}{l} A_1 \xrightarrow{-C} A_3 \\ A_3 \xrightarrow{+C} A_1 \end{array}\right\} \approx \{A_1, A_3\}, \tag{4.60}$$

as shown in figure 4.6.

Referring to figure 4.6, we see that causal fabric may express synchronicity of promises either in parallel (i) or in series (ii). In the first parallel case, the effective coarse grained agent $\{A_1, A_3\}$ promises collective action, and its exterior promises are equivalent to a scaled version of a single agent. In the second, serial case, there is no promise of action (+) by A_3, only a (-) promise about the memory of its effect by A_1. In i) there are parallel (+) promises to collaborate in the offer of a single property to A_2; in ii) there is a promise to offer b combined with a promise to recall the reason for b. The two promises made by A_1 don't match lock-and-key as in the first case (but see section 4.4.9).

Note that the processes represented by the graphs are not reversible. The process has a definite direction and nothing about the symmetry of $\pm$ implies reversibility. The reversibility implicit in the action principle and Noether's theorem arise from the implicit mutual reversibility of spacetime itself, and the assumption that there is mutual connectivity in both directions between neighbouring points. Locality is ensured by the $\pm$ symmetry. If $\pm$ became indistinguishable (as they implicitly are in a vector space) then this follows.

Note that, while the graphs may seem to imply determinism, they only express affinities, propensities, or 'intent'.

4.4.5 SCALING OF CAUSATION, ITS RELATION TO CONSERVATION AND DISTINGUISHABILITY

If we assume the existence some property, represented by the promise body $b = (\tau, \chi)$, is conserved then we assume interaction semantics in which an entity represented by b

moves from one location to another. It cannot be in two places at the same time. So, if we imagine a chain of agents that form a process involving the transfer of b:

$$A_1 \underset{\xleftarrow{-b}}{\xrightarrow{+b}} A_2 \underset{\xleftarrow{-b}}{\xrightarrow{+b}} A_3 \underset{\xleftarrow{-b}}{\xrightarrow{+b}} A_4 \ldots \underset{\xleftarrow{-b}}{\xrightarrow{+b}} A_n \tag{4.61}$$

or

$$\Delta^{\text{ret}}_{12}(b)\ \Delta^{\text{ret}}_{23}(b)\ \Delta^{\text{ret}}_{34}(b) \ldots \Delta^{\text{ret}}_{(n-1)n}(b). \tag{4.62}$$

From these promises, we cannot say *a priori* when or if the property b is passed along the chain. The fact that arrows point in both directions along the chain doesn't change the directed nature of the graph. All we can say is that there is a 'field' influence that tends to propagate b from left to right. This is not a reversible process. Regardless of when or how the promises are kept, the promises are uni-directional by virtue of the $+$ and $-$.

In order to ensure conservation, we need to introduce some form of co-dependence between the agents. This can be added in a fully reversible way (called entanglement) [BBKK18] or for the directed case above. Let's start with the directed chain.

Conservation is tricky because, in order to ensure it, for an arbitrary type of promise, with full distinguishability, we need to represent the entity as something with its own independent material existence. In fact, a unique information string suffices—something like a unique code or password that maintains its integrity. It is indivisible and uniquely identifiable. If we can relax this uniqueness, there is the possibility of a distribution.

- Sticking with uniqueness for now: let's call the property M_α, for some unique α, then we need the agents in spacetime to be able to promise the presence of the unique information M, and other locations to promise its absence $\neg M_\alpha$:

$$A_1 \quad \xrightarrow{M_\alpha} \quad * \tag{4.63}$$

$$A_i \quad \xrightarrow{\neg M_\alpha} \quad O \quad i \neq 1. \tag{4.64}$$

  where O is some observer agent, which may or may not be a member of A_i. In other words, one agent must promise 'M_α is here' and everywhere else must promise 'M_α is not here'. This is how money transactions work, for instance, for each unique transaction α[38] . Any number of such collections of promises could be made for different α. Crucially, this requires co-dependent cooperation (a form of entanglement) between every location in space, which is expensive.

- If we are unable to distinguish different messages $M_\alpha \to \hat{M}$, then at best we can promise the amount of $\hat{M}$ at each agent location:

$$A_i \xrightarrow{+\hat{M}(A_i)} O. \tag{4.65}$$

> The amount of $\hat{M}$ here is $|M|$. This is how money works when we don't keep ledgers of transactions, and treat monetary exchange as a Markov process.

Note that trying to single out one instance of a generic $\hat{M}$ is not the same as making it a properly distinguishable M_α, because one could imagine some sleight of hand by which different amounts were rerouted to redistribute $\hat{M}$ along the way. So these are the two cases.

The more elementary the scale of the agents A_i, the less likely it is that they would have the interior memory capacity to be able to promise uniqueness, and thus the more likely that one would observe elementary agents in classes of distributions.

We can now imagine forming a superagent $\mathcal{A} = \{A_i, \forall i\}$, which can promise the specific location i of $\hat{M}$ for the entire collection of agents to an observer (which must now be external to X), in any given configuration. This suggests that the promise of a specific location, for a generic quantity (discrete or continuous but without unique labels, like cash money) can have no definite position. Counters will distribute themselves and appear only as the promise of a frequency distribution

$$A \xrightarrow{+\psi} O_{\text{ext}}. \tag{4.66}$$

In QM we might write this something like:

$$\langle O|A\rangle \tag{4.67}$$

and over all agents,

$$\sum_A \left(A \xrightarrow{+\psi} O_{\text{ext}}\right) = \mathcal{A} \xrightarrow{+\psi} O_{\text{ext}} \simeq \psi(A), \tag{4.68}$$

and

$$\sum_A \left(A \xrightarrow{-\psi} O_{\text{ext}}\right) = \mathcal{A} \xrightarrow{-\psi} O_{\text{ext}} \simeq \psi^\dagger(A), \tag{4.69}$$

where the final functional inference is allowed to take on the most general representation of the $\pm$ symmetry, which—when conserved—is the unitary group.

In this model, the observer is exterior to the system and automatically has a god's eye view of it, by implicit assumption. Moreover, the distribution of M has the form of a gas, since the A_i have no promised interior structure. The blob of agents is amorphous on a large scale.

4.4.6 ORDERED AGENTS

Suppose now that a collection of agents effectively (and emergently) promises to crystallize into a coordinate lattice (figure 4.7), by each promising its role of being next to a

neighbour. Now, we need basic distinguishability of agents by their neighbours, in order to specify their precedence:

$$A_1 \xrightarrow{X_1} A_2 \xrightarrow{X_2|X_1} A_3 \ldots \tag{4.70}$$

This is a Markov chain. Filling out the promises in full[Bur14],

$$A_1 \underset{\xleftarrow{-X_1}}{\overset{+X_1}{\longrightarrow}} A_2 \underset{\xleftarrow{-X_2}}{\overset{+X_2|X_1}{\longrightarrow}} A_3 \underset{\xleftarrow{-X_3}}{\overset{+X_3|X_2}{\longrightarrow}} A_4 \ldots A_n \tag{4.71}$$

This is the promise structure represented in figure 4.6 ii). it leads to serial order, at the expense of non-local knowledge of neighbouring agents. Notice the apparently acausal feed-forward promises at each that complete the logical semantics 'I have made a conditional promise, and I promise that I have the condition in hand'.

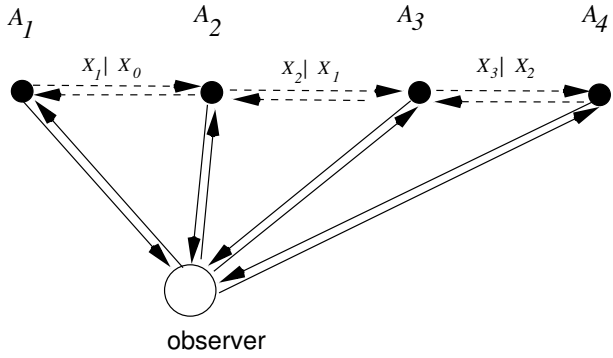


Figure 4.7: A number of agents A_i promise that they are greater than their neighbours and less than the preceding one. Their gradient promises all have only local significance, as if vanishing a Newtonian limit were taken—all agents could promise the same, from their own perspective, in opposition to one another. For an external observer, there is nothing in the information supplied by the agents that suggests their relative order unless the observer accepts such information from the remote agents on trust. Otherwise it has to trust its own ability to discriminate messages by sensory criteria (e.g. the angle of incidence of the signal on its boundary), which assumes macroscopic interior structure on the observer—suggesting that the size of the observer, and its relation to promise channels, play a role in the ability to assess order. The labels on the A_i are for convenience. Nothing should be implied by their order. Indeed, this exemplifies a common problem we have in assuming our Newtonian abilities to label and order by virtue of unlimited powers of access and observability.

The notation in (4.71) suggests that every agent need to know a unique name of its nearest neighbours. That may not be strictly true, since in a Markov process it suffices to

know that one has a predecessor in order to build a ladder. However, if we are interested in maintaining a causal order relation in which each agent knows in which direction a certain vector continues, that information is not clear without labels that go beyond a simple predecessor.

For example, suppose you are a spinning top (which has no intrinsic direction sense) trying to walk in a straight line along across a chess board by hopping from square to square. After each hop, how does the spinning top know which direction it came from in order to continue in the same direction? This clearly has implications for quantities like velocity and momentum along trajectories.

Again, there is no reversibility yet. For that, we would need an independent chain of promises in the opposite direction:

$$A_1 \xleftarrow{\overline{X}_2|\overline{X}_3} A_2 \xleftarrow{X_3|X_4} A_3 \ldots \tag{4.72}$$

The property of reversibility is not automatic, if one takes locality as an intrinsic starting point—it has to be promised independently by some or all agents. Why we should observe symmetry between left and right, forwards and backwards, up and down, is unclear. We've come to assume that symmetry is an absence of information, but if we treat all agents as a coordinate lattice, that is not the case. Indeed, the opposite is true. There is long range order.

From equation (4.71), we see that each agent has to be 'aware' of its neighbours in order to form a lattice. This happens because agents are assumed to be autonomous or independent. Normally in mathematics, we assume that points form any structure we want because we impose it thus. In a Promise Theory perspective there is no reason to assume this—and we see no such automatic structure at large scales, so there is no obvious reason to expect it a small scales either.

In crystals, multi-pole distortions of the lattice on the scale of the electron clouds exhibit these non-local effects. The indistinguishability of electrons allows the entire structure to remember some average configuration without precise and expensive promises to track every electron in the role of a unique messenger M_α. At this scale, the role of agent locations is played by atomic configurations for the different elements, which promise to fall into a finite number of classes from the periodic table.

4.4.7 SCALING OF PROCESSES

The most elementary processes in the universe are unlikely to have enough structure in order to maintain stable clocks and memory to distinguish themselves. Far more likely is that there is a virtual process layer on top of this, with superagent identities Q_A that consist of collections of agents A_i at any scale (see figure 4.8). This needn't be a

regular or homogeneous arrangement. As long as there is sufficient entropy of mixing, an observer on a sufficiently large scale will perceive the granular structure as metallic or something like a High Entropy Alloy. Each superagent Q_A is assumed to be a process

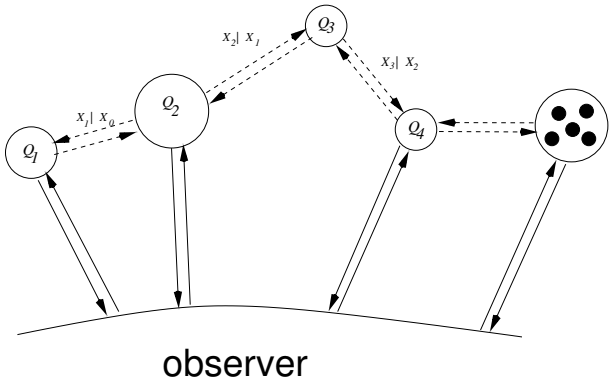


Figure 4.8: Scaling up processes based on agents that cooperate on a smaller scale takes us to virtual particles (superagents in Promise Theory), which are more likely candidates for expressing different physical properties and distinguishable information than elementary agents. These virtual entities can move with respect to one another, like amoeboid motion, recall their direction like particles with momentum, etc. What are the minimal sets of promises that would enable such behaviours?

formed by cooperation of A_i. The promises we require of Q_A are sufficient memory to form clock counters and distinguishing traits.

$$\nabla Q = \frac{(Q_A - Q_B)}{\Delta Q} \tag{4.73}$$

$$\dot{Q} = \frac{(Q_A - Q_B)}{\Delta T}, \tag{4.74}$$

where ΔQ is a 'natural scale' for the granularity of the agents, based on the number of interior states, and ΔT is a timescale associated with the binding to an observer. Each observer counts time by itself, but has to rely on the promised properties of Q_A (that is can distinguish) to report on the structure of Q_A bindings. The reason we can't determine ΔT is that we need to decide whose counter measures the transition from Q_A to Q_B. As exterior time between the neighbouring agents, there is only one transition and one tick of exterior time, so only a fixed value for $\dot{Q} = 1$ for either of the participants—but that need not be the case for onlookers who observer remotely.

If we assume that this layer of virtualization is observable, while the underlying layer A_i is not readily observable, then all traces of absolute spacetime in A_i will be rendered

moot, and observers will only experience relativistic measurements. This picture suggests that Minkowski spacetime would have something like a metallic granular phase structure. The spacetime propagator associated with motion

$$D_{AB} = \begin{cases} Q_A \xrightarrow{+\mathbf{x}} Q_B \\ Q_B \xrightarrow{-\mathbf{x}} Q_A \end{cases} \tag{4.75}$$

The effective dimensionality of Q_A may be determined by statistical percolation, as proposed in [Myr78].

The ratio of scales Q/A (or average distribution of ratios) becomes a key dimensionless parameter in the scaling of the system, which might plausibly be related to Planck scales.

4.4.8 THE MEANING OF A LINK

The counting of a regular spatial distance is analogous to the counting problem for time. In a metric space, the order and regularity of space is defined by a functional dependence on coordinates. This is often called local, but it is non-local in the sense that the properties over an entire tangent space are presumed regular and coordinatizable, which amounts to a tautology.

Every promise made between agents implies a total ordering of the agents, by virtue of their being unequal. So two agents A_1 and A_n being near or far apart depends on the channel criteria between them: a sequence of autonomously promised orderings:

$$A_1 \preceq A_2 \preceq \ldots \preceq A_n. \tag{4.76}$$

Suppose each promise is a basic spacetime fabric promise of adjacency with body adj, as in [Bur14], them an observer can assess each of these promises:

$$\begin{aligned} \alpha_O(A_1 &\xrightarrow{+\text{adj}} A_2) \\ &\ldots \\ \alpha_O(A_{n-1} &\xrightarrow{+\text{adj}} A_n) \end{aligned} \tag{4.77}$$

The assessment depends on the observer O's self-calibrated understanding of scale. Thus a metric space has no meaning without a local observer view. The observer needs a memory of coordinate associations with known sources to make this picture. Even the pairwise promises in (4.77) above are not sufficient to ensure the total order in (4.76), while it suffices for the individual agents to treat adjacency as a Markov process, for the observer the promises need to be made conditionally because it cannot a priori trust the channels over which information about relative order has arrived.

To say that an agent is close to another requires assessments of $\preceq$ over extended sequences, and potentially different types of promise. To promise distance, we have to accept all the promises of intermediacy. This is a version of the well-known the end-to-end problem in service delivery[BB14a]. The observer may be able to promise the distance between A_1 and A_n conditionally, based on the promises from A_n, but this has a high cost:

$$O \xrightarrow{d(A_1,A_n) \mid (A_1 \preceq A_2),(A_2 \preceq A_3) \ldots (A_{n-1} \preceq A_n)} A_? \tag{4.78}$$

In the absence of a god's eye channel, outside of the ordinary adjacency promise linkage, this promise cannot be made without conditional evidence[39].

Does this show that a metric space is only a consistent concept as a local tautology, or as an embedding observable from a god's eye view? The memory component of coordinates is expressed in $g_{\mu\nu}(\overline{x})$ by $\overline{x}$.

We can't absolve the problem of basically *trusting* the integrity and authenticity of self-consistent observer experiences when describing spacetime as an independent entity. The implication that information about spacetime locations has to be promised by sender and receiver, and that every receiver forms its own assessment of the information it receives, implies that a coordinate system is a *trusted label system* accumulated over many interactions with different pairs of agents. It is possibly even second hand knowledge passed through intermediaries—a model kept in the memory of an observer to classify subsequent measurements believed to originate from a labelled source. The observer need to believe it can distinguish the source reliably over multiple observations, i.e. processes need sufficient stability to form a reliable reference system to be called coordinates. This is analogous to Mach's principle about the fixed stars.

4.4.9 THE END TO END PROBLEM, LINK SEMANTICS, AND DISTINGUISHABILITY

In Causal Sets, one assumes the existence of the order relation (links) as an abstraction, but it's interesting to see how such a relation can be constructed within the simple rules of Promise Theory. This reveals some of the symmetries between the complementary interacting agents, and makes it clear how there is no automatic reversibility in spite of these symmetries.

Figure 4.9 shows a situation in which an agent in the role of 'sender' or earlier agent S makes a promise to a receiver or later agent R, with the help of a third party L, an intermediary can be viewed as a 'link' agent. The promised role of L in this example will show what it means for there to be a causal link between S and R. This takes on a special importance when scaling. We can think of the sending of a message by a runner,

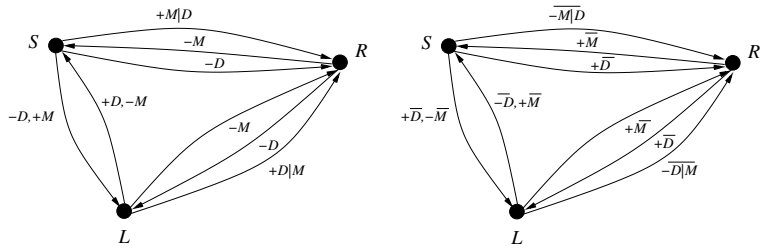


Figure 4.9: The end to end promises for a unidirectional process, with complementary transform. The right hand version is a relabelling of the left hand version.

or the postal order delivery from S to R with the help of a postal service L. The scale is not important, nor the nature of the agents, to the structure of causal message delivery.

For example, the interpretations can be read:

$$+M \quad \leftrightarrow \quad \text{Offer message} \tag{4.79}$$

$$+D \quad \leftrightarrow \quad \text{Offer delivery} \tag{4.80}$$

$$+\overline{M} = -M \quad \leftrightarrow \quad \text{Accept message} \tag{4.81}$$

$$+\overline{D} = -D \quad \leftrightarrow \quad \text{Accept delivery} \tag{4.82}$$

In the figure, we can imagine a distinguishable message M is promised by the source S to a receiver R conditionally, making use of an intermediary L which promises delivery labelled by D. In other words, L will do the actual work, but the intent is signalled by the endpoints. S promises $+M|D$, which reads 'I promise you M if I get D from elsewhere'. The source further promises that it has acquired D by promising to use D, i.e. $-D$ to R. R does not need to know the name of the delivery agent, only that it exists, so the body D does not contain that non-local information. It will be up to L itself to introduce itself to R. The receiver accepts the promise of a message, signalling its willingness to accept such a message $-M$. It doesn't have to accept the condition, since this it has no influence over that decision.

Now there is a link agent L which promises delivery service $+D$ to S, which S accepts $-D$. The link agent further promises delivery to R, conditionally on getting a message from S, i.e. $+D|M$. Its conditional role is the reverse of that for S, and the diagram shows the symmetry between S and L as they exchange roles. S also promises to deliver a package to L, i.e. $+M$, and L accepts with $-M$, unconditionally.

There is now a deadlock of precedence relationships promised, whose order can only be broken by an explicit act of symmetry breaking (i.e. a boundary condition at S).

$$S \xrightarrow{+M} L, \tag{4.83}$$

i.e. an unequivocal change of state between the sender S and the delivery agent L. A symmetrical imposition by the receiver does not return the message to sender:

$$R \xrightarrow{+\overline{D}} \blacksquare\, L, \tag{4.84}$$

because the extended and non-local process has direction built into it. It is unidirectional. It amounts to a signal or tick of its clock 'I am still waiting for a delivery', which has no causal impact on S, the origin of messages (unless the process is of sufficient scale to incorporate processes for customer satisfaction!). This underlines the fact that (in spite of the complementarity of give and take $\pm$) it takes more to create reversible system, because every point location or agent is a priori independent in its choices. This is the result of putting locality first as an organizing principle.

$$S \xrightarrow{+M|D} R \tag{4.85}$$

This promise, by itself is an empty one.

$$\overline{-M|D} \quad \rightarrow \quad -\overline{M}|\emptyset \tag{4.86}$$

$$+\overline{M} \quad \rightarrow \quad +\overline{M}|\overline{D} \tag{4.87}$$

$$\overline{-D|M} \quad \rightarrow \quad -\overline{D}|\emptyset \tag{4.88}$$

$$+\overline{D} \quad \rightarrow \quad +\overline{D}|\overline{M}. \tag{4.89}$$

This examination of the role of intermediate agents in propagation tell tells us two things. The implication is that the intermediate or 'link' agent L is irrelevant semantically and may be absorbed by superagent scaling. It also tells us that promising a completely ordered potential in a lattice becomes N^2 expensive as the number of intermediate agents N grows[BB14a]. Even at a macroscopic level, this is expensive and taxes the resources of agents. On an elementary process, it seems highly unlikely that one would find such a structure. Evidence in the chemistry of long chain molecules and metals shows at indistinguishability plays a saving role to prevent conditionality from diverging.

4.4.10 SCALE DEPENDENT ARROWS OF SPACETIME: HOW IS $\neq$ DIFFERENT FROM $>$ AND $<$?

Let's start with semantics of the partial order relations assumed in mathematics, from a local observer perspective. If $A \neq B$ and $B \neq C$, it does not follow that $A \neq C$, however if $A \overset{<}{>} B$ and $B \overset{<}{>} C$, it does follow that $A \overset{<}{>} C$, so ordered inequality is a stronger relation than non-ordered inequality. The latter implies a form of 'long range

order'. The question is: how can an observing agent make this non-local distinction based on only local information? It would seem to need memory.

- $\neq$ is determined by an external observer, and if information about the state is available this can be used to order the agents as the observer sees fit (figure 4.10). Seeing that $A \neq B \neq C \neq A$, an observer O may form its own assessment:

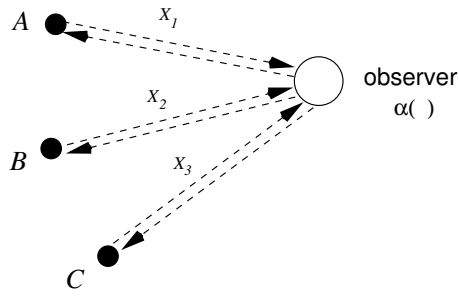


Figure 4.10: Agents promise only nearest neighbour relationships as a Markov process, and the observer forms a map, from its own assessments, as a model in its own interior memory.

$$\alpha_O(A) \overset{<}{>} \alpha_O(B) \overset{<}{>} \alpha_O(C) \tag{4.90}$$

and can now promise this conditionally (figure 4.11).

- $\overset{<}{>}$ is a borrows assessment from information promised by the superagent about the interior order, trusting knowledge of the promise relationships revealed by A, B, C:

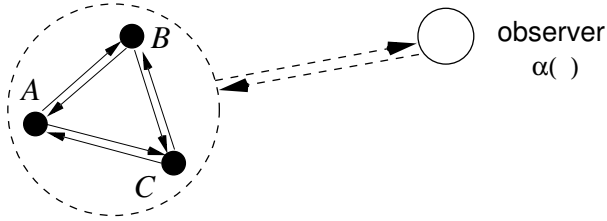


Figure 4.11: Agent's promise their relative lattice structure as a totally ordered superagent, with collective behaviour. The observer trusts the promise made to it without further validation or observation.

Direction is a scale dependent property of spacetime. This is easily understood by thinking of the following: imagine arriving at a crossroads, you seem to have three independent choices for going ahead, but it turns out that all three roads eventually meet back at the same point again. Locally, the forward dimension seems to be three (if you believe in networks), or two (if you believe in Euclidean embeddings). In fact, on a larger scale there is only one dimension, as all roads lead to Rome. This is an instance of the *percolation problem* well known in solid state physics[New03][40].

4.4.11 DISTINGUISHABILITY OF CAUSAL PATHWAYS

The previous remark points out that there is another way in which distinguishing agents scale dependent. The direction in which promised information arrives at a location may be significant, when there is incomplete information. Several routes may lead to the same location. This is the norm in computer networks, and is usually relegated to the realm of topological spacetimes in physics (cohomology/holonomy). However, we know this is not reliable when many non-straight paths can be taken by information, as testified by refraction, lensing, etc. Path curvature has implications for distinguishability (see figure 4.12).

The difference between a discrete spacetime process and a continuum one is that topological defects are the norm: we can't take anything about observation and signalling for granted. The continuum spacetime theatre model is a great artifice for separating concerns in arguments, but at the most elementary level, some concerns cannot be separated. When promises are made directly between source and receiver, there is less difficulty in making distinctions. When intermediate agents are involved, the certainty of observations falls off rapidly.

This plays a significant role in the measurement of non-local quantities like velocities and momenta that require samples over multiple points in order to infer a result—which goes a bit beyond the scope of these notes.

Semantic spacetime results in a directed graph theoretic picture, consisting of agents (nodes) and promises (that may lead to links or edges). At every 'point' in space, we assume there is a node in a graph. The dimension of spacetime at every point is a much more subtle issue in a graph than in a vector space manifold.

4.5 SCALING LOCALITY WITH CO-DEPENDENT ENTANGLEMENTS

When agents make promises that render them co-dependent, by virtue of mutually binding conditional promises, made equally in advanced and retarded directions, it no longer

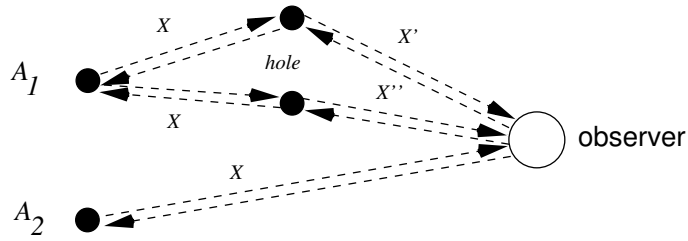


Figure 4.12: If promises are not made by direct adjacency, but intermediate agents are involved, new information can be injected into messages. An observer may be able to make a distinction between the multiple paths between source and itself. Now we need new promises in order for the observer to be able to distinguish whether it is seeing two different agents or two images of the same agent. In Promise Theory, the law of Intermediate Agents addresses this question.

makes sense to distinguish the roles of sender and receiver, as both agents are locked into both roles. In this case, we cannot speak of partial order, or even causal order on a microscopic scale. Entanglement is an explicit case of a *preorder*. The behaviour of co-dependent agents depends on the relative timescales of the processes that keep promises. Here I'll largely follow the more extensive discussion in [BBKK18].

When agents collaborate, or act cooperatively, it is natural to define a 'superagent' to label them as a collective entity[Bur15a]. When such a superagent, composed like $S = A_1 \oplus A_2$, makes promises that cannot be attributed to or kept by either of its components A_1 or A_2 alone, then we say the sub-agents are entangled, and we say that the superagent is irreducible[Bur15a]. This happens when promises are mutually conditional. Consider a two agent system, with agents forming a diatomic molecule, which we can label left and right.

Lemma 6 (Entangled with respect to b)**.** *Two agents A_L and A_R are said to be entangled or irreducible if the superagent $A_L \oplus A_R$ enveloping both of them makes a promise that neither of the two agents can make alone. This can only happen if each agent makes promises conditionally on promises made by the other.* □

This definition is compatible with the definition of entanglement in information theory[DCP17]. For any promise bodies b_L, b_R, the necessary and sufficient solution to this condition is

given by

$$A_L \xrightarrow{+b_L|b_R} A_R \tag{4.91}$$
$$A_R \xrightarrow{-b_L} A_L \tag{4.92}$$
$$A_R \xrightarrow{+b_R|b_L} A_L \tag{4.93}$$
$$A_L \xrightarrow{-b_R} A_R. \tag{4.94}$$

The proof is trivial: both sides promise b_i ($i = L, R$) with a dependence on the promise $b_{\bar{i}}$ from the other, else they would promise independently which would contradict the definition. If the agents do not promise the explicit dependence on the other in (4.92) and (4.94), then (4.91) and (4.93) are not complete promises, by the conditional promise law 6.2 of [BB14a], that no dependent promise can be given without accepting the dependent promise of the other, thus A_L must accept b_R and vice versa.

In spite of a simple proof, the dynamical behaviours of this co-dependent configuration is not completely defined. It depends on the relative timescales for communication and sampling at each end. In physics one would normally assume agent symmetry by default, but that violates the principle of autonomy, and amounts to assuming reversibility[41]. When b_L or b_R changes, these promises may be thought of as cyclicly generating an evolving sequence of preconditions, which unfolds as a chain of transaction events, until an equilibrium is possibly reached.

4.5.1 INTERIOR AND EXTERIOR TIME AND OBSERVABILITY

The keeping of promises at each agent has a timescale, as measured by each agent's interior clock. This is the clock that determines the rate at which the agent can sample information. Clocks may be entirely on the interior of a process, in which case they measure interior time. They may also include ticks 'driven' by external sources, in which case they measure exterior time.

Interior and exterior relate generally to a definition of a boundary, but agents that are strongly coupled cannot easily be separated and therefore act as a single agent. This means that entangled agents have a single shared clock, not several independent clocks. Entanglement is a co-dependent causal evolution of state; i.e. it works in both directions 'at the same time', so we must be careful what we entangle, how 'the same time' is defined, and how directionality is arranged. It affects superagent clusters, of scaling dimension $s > 1$. Promise theoretically, we can observe that there are implicit timescales as a result of irreducible co-dependence being composed from atomic elements:

If we define a timescale by $\Delta t^{(s)}$ at scale s, measured according to the clock of an exterior godlike observer (figure 4.14), with access to all information, then each tick corresponds to a single promise-keeping event. The cells cannot observe these events,

which happen in between the ticks of their 'proper time' clocks, so we might call this ability to observe the most detailed equilibrating events *subtime*[42].

A complete cycle of entangled co-dependent causation leads to a natural coarse-graining of time that corresponds to the aggregation of interspatial events (two agents L, R keeping $+, -$ promises to close the cycle). The hierarchy of interior clocks $\Delta t^{(s)}$ is bracketed below:

$$\left.\begin{array}{l}\left.\begin{array}{l}\left.\begin{array}{l}A_L \xrightarrow{+b_L|b_R} A_R \\ A_R \xrightarrow{-b_L} A_L\end{array}\right\} \Delta t^{(1)} \\ \left.\begin{array}{l}A_R \xrightarrow{+b_R|b_L} A_L \\ A_L \xrightarrow{-b_R} A_R\end{array}\right\} \Delta t^{(1)}\end{array}\right\} \Delta t^{(2)}\end{array}\right\} \Delta t^{(4)}. \tag{4.95}$$

To maintain the entanglement, a message b oscillates back and forth between the two ends on different timescales. If we assume that this happens synchronously, so that the puck is a conserved quantity, then each agent is independent for $s = 1$, but the complete process of emission and absorption of influence to keep the promise requires a tick at $s = 2$ of the pair. The same is true for passing the puck backwards, so the complete synchronous interaction takes a tick of scale $s = 4$ to achieve reversible symmetry in a strongly coupled synchronous manner.

If we drop the requirement of synchronicity, each agent may emit and absorb at its leisure, then the interior clocks no longer tick in step, and there is only a finite probability, determined by Nyquist's law, of whether or not a coordination message will be passed between them. This is a weak coupling. A third part observer with an equal view of each[43] could measure the clock tick rates of left and right agents and compare them to determine whether one is faster than the other, but without such an observer that assessment has no meaning to the agents' assessments of one another.

A moment's thought shows that the dynamics of this configuration depend on the sampling rates: whether the process distinguishes sending and receiving as independent events or not. If sending and receiving are blurred into the same coarse grain of time, then the co-dependence relation is a *deadlock* equilibrium condition that implies no net motion in either direction. If, on the other hand, the sampling is fine grained enough to be able to detect precedence, then it is a 'pump' that oscillates back and forth forever as long as the symmetry is broken on one side by an initial condition such as:

$$A_R \xrightarrow{+b_R|b_L} \blacksquare\, A_L. \tag{4.96}$$

Entanglement thus implies quantization of both space and time, because nothing independent can happen in an entangled network, but we can only observe entanglement on a coarse-grained timescale[44]. If we refer to $\Delta t^{(4)}$ as *exterior time* or co-time, and $\Delta t^{(s)}$ for $s < 4$ as *interior time*, then we can call $\Delta t^{(1)}$ specifically *subtime*. It is purely local, and not observable by any other agent. The promise of entanglement (co-dependence) is only observable at a timescale $s \geq 4$. These basic points will inform the discussion of a protocol by which can use entanglement to built a quasi-deterministic communication channel. See reference [BBKK18] for more details.

4.5.2 IRREDUCIBLE SUPERAGENT PICTURE OF SPACETIME GRAINS

Co-dependent promises, made (and kept) by the endpoints, must be maintained regardless of what other independent promises cells might make to any other agent. This happens when both agents are driven by what happens between them rather than coordinating their independent activities (see figure 4.13). Our goal in this paper is to explore the use of this property in order to keep strong promises about message delivery. Notice that these co-dependent promises are invariant under $L \leftrightarrow R$, and are thus timeless and without preferred orientation.

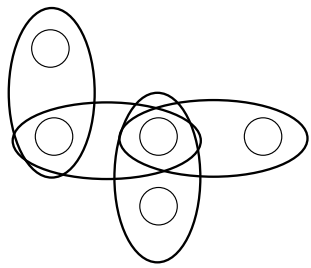


Figure 4.13: Entanglement results in a new effective picture, with overlapping irreducible superagents. Entanglement (irreducibility) is not a transitive property, as the diagram shows: the overlapping of superagents does not imply a single large superagent keeping the same cooperative promises.

Lemma 7 (Composition of entangled links)**.** *The composition of irreducible or entangled links, as in figure 4.13 cannot itself be irreducible or entangled.* □

The proof of this follows from the linear combination omits off-diagonal promises (see 11.1-13.1 in [BB14a]).

4.5.3 SINGLE-VALUED CO-TIME FOR PAIRED AGENTS

In the geometry of the link, there are two distinct possibilities for temporal evolution of the irreducible link superagent. We identify these as *local* and *non-local* in spacetime. They correspond to how we define the clock by which events move forward on the two ends of the link. When agents are independent, they can each maintain independent state,

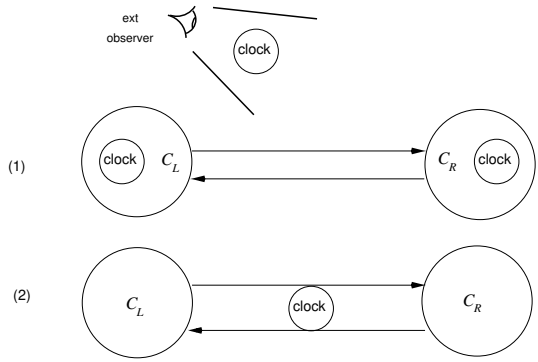


Figure 4.14: Two entanglement time models mix interior and exterior time of the agent: (1) asynchronous local and (2) synchronous non-local models are about where each agent's clock signal is sourced.

and hence have independent clocks; but when agents are entangled, or co-dependent, they share all the state that pertains to their co-dependent promises, including a common clock.

In effect, an entangled link moderates the flow of information on both sides by (b) locking observability of state. The challenge in using this as a technology is to encapsulate the promise to transfer data such that each packet can only be observed on one side or the other.

The conundrum with this arrangement (see figure 4.15) is that the passage of time will never be single-valued throughout an application unless we give up locality. Agents need to have a split brain approach to time in order to i) be able to maintain strong entanglement promises, and ii) to be able to observe when all activity has ceased on a link, in order to restart it.

Once primed, the entanglement of end points can form the basis of a simple pendulum, pump, or motor, which in turn acts as a clock or generator for transfers. Once the LR symmetry has been broken by insertion of a message (see 7.4.1 in [BB14a]), it will be superposed onto the control channel and passed from one side to the other, *if and only* if the payload can be accepted by the other side. This can be used as a basis on which to

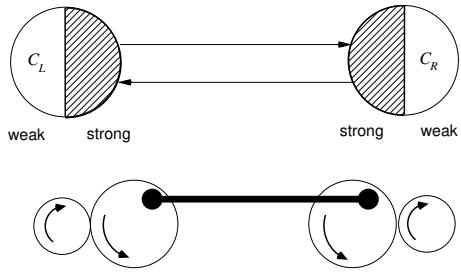


Figure 4.15: Agents need to maintain a locally split brain model to retain control over the link, and avoid harmful deadlocks. If the link drives all aspects of the agents, they become too fragile, leading to possible failures of the link. This can be imagined as wheels joined rigidly by a crank (in the entangled region), and decouplable gears that can be introduced to drive or be driven by the link. This is reminiscent of Maxwell's original vortex model of electromagnetism.

ensure a conservation of information, as we'll see in chapter 5.

4.5.4 FUNDAMENTAL ASSUMPTION OF HOMOGENEITY

The ability to 'trust' or rely on these behaviours effectively assumes a standard calibration of both ends of a link against an impartial third party (see figure 4.16). This trusted party might be common software, or a third party service, but it must exist, else no agent can be sure of what its neighbour will do with data it attempts to send (it is analogous to having the same laws of physics at both ends of the link)[Bur14, Bur15a].

Assumption 1 (Spacetime homogeneity)**.** *In order to rely on any promise propagation, all agents involved are assumed to keep the same homogeneous basic set of promises, according to their agreed left/right roles, because the effectiveness of entanglement promises depends entirely on a uniform conditional basis.* □

This is essentially an assumption of non-local trust in a computing sense, and is the basic behaviours of cells in biology. Without standard semantics, there is no 'linguistic' (i.e. symbolic information) overlap to interpret interactions by. If we don't assume this, we have to show how it can emerge—see the discussion in [Bur15a].

4.6 PROCESS MASS

The collective relational encumbrance of a process, with regard to promises or 'obligations' may be used to define its effective mass. The mass of an agent is usually only

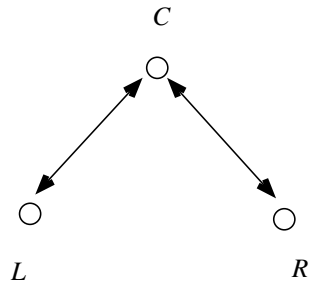


Figure 4.16: Collaboration requires a trusted calibration. The calibrator could be any implicit 'godlike' observer, or permanent non-local synchronization, e.g. use of common software. This is analogous to having the laws of physics the same on both sides of the link.

relevant when exterior changes to the agent are promised or take place. The mass itself is a dynamical property of an agent's exterior interactions, so its value is only an instantaneous sampling of its interactions at each moment. In other words, the mass of an agent can change over time, according to any observer's clock.

Definition 67 (Instantaneous process mass)**.** *A scalar measure attributed to any agent or superagent A, proportional to the number of promise bindings it maintains with agents A_i that are adjacent to it, with respect to the process concerned. For some body b, with promises that are effectively invariant over the timescale of changes pertaining to the mass:*

$$A \xrightarrow{+b_i} A_i \tag{4.97}$$

$$A_i \xrightarrow{-b_i} A, \qquad i = 1 \ldots N. \tag{4.98}$$

then we can define the mass as a scalar measure $m(A)$, with some dimensional constant, and proportional to

$$m(A) \propto N. \tag{4.99}$$

Note that an agent may have interactions with other agents on a longer timescale, but these do not impact changes that happen on a shorter timescale.

e.g. An atom may occasionally absorb and emit photons, but this does not affect its mass when moving from A to B. On the other hand, a continuous binding to surroundings in a crystal lattice does affect the ability of the atom to move from A to B.

4.7 PROCESS VELOCITY

In order to define a velocity for a process, we can't rely on Euclidean ideas, as agents know nothing about such effective embedding spaces—instead we need to define what we mean by distance and time collectively. Distance is a property of coordinate systems, which are subjective models belonging to agents. Adjacency is normally used to define distance, but adjacency is also an arbitrary kind of assessment in a multi-typed graph. As with mass, we can imagine any number of promise types b_i, where $i = 1 \ldots N$, representing different promises:

$$\pi_i : \; A \xrightarrow{b_i} A_i \tag{4.100}$$

The homogeneity of agents is simply a convention or assessment choice of the agent that defines the coordinate system. In a crystalline configuration, one might expect agents to be equidistant, but in a gaseous configuration, distance can be a fluid concept. In other words, agents are essentially free to make up their own ideas about distance. No two agents have to agree on an impartial model of distance, but if they are involved in a collaboration, then it makes sense for them to establish a consistent collaborative definition.

Definition 68 (Agent distance $D_\pi(A, A')$)**.** *Any scalar measure $D_\pi(A, A')$ attributed to a promise binding $\pi_\pm$ between two distinguishable agents A and A' by any agent in the role of observer.*

Note that each agent can choose to assess distance differently. This is what we mean by choice of a coordinate system in physics and mathematics.

Next, we define a response time for influences to propagate from one agent to another:

Definition 69 (Response time $T_\pi(E, E')$)**.** *The number of clock ticks assessed to have accrued, on an observer's reference clock, that are counted by the observer between an event E at agent A and an event $R(E)$ at agent A', where:*

$$A \xrightarrow{+E} A' \tag{4.101}$$

$$A' \xrightarrow{-E} A \tag{4.102}$$

$$A' \xrightarrow{R(E)|E} A. \tag{4.103}$$

The events are assessed to have occurred when the observer accepts the promise of the events as kept, so the observer is assumed to be in the scope of these promises. Note that the observer making the assessment may be any agent, including A and A'.

The rate at which conditional or unconditional promises propagate may now be defined, as a velocity, using these definitions.

Definition 70 (Process velocity v_π)**.** *Where both D_π and T_π can be defined, the instantaneous velocity of a process between points A and A' may be written as the ratio of*

$$v_\pi = \frac{D_\pi}{T_\pi}, \tag{4.104}$$

where D_π and T_π are defined in the foregoing definitions, for some bundle of promises π representing the process.

Moreover, given that motion is a haphazard affair on a graph with instantaneous interactions, the notion of a random walk is useful:

Definition 71 (Random Walk)**.** *The trajectory formed by a process, described as a sequence of dependent events that propagate causally.*

The assessment of randomness is a property of the observer, not of the process, since the information implicit in the promises between the agents leads to a deterministic trajectory. It is the embedding of the path in the user's map that leads to the assessment of randomness.

4.8 SUBORDINATED WORKFLOWS: FORCED PROCESSES

When staging is performed in a more linear fashion, networks become flow charts in which there is a preferred direction. One imagines pushing work through the pipeline from one end to the other, rather than imagining a more democratic signalling of equipartitioned nodes in a hierarchical network. Workflow networks are driven 'top down', where the input is the top and the output is the bottom. Scaling of staged promises from a few cases to large numbers effectively turns a sequences of transactions into a kind of average 'workflow'. These networks are very important because the model the story telling manner of thinking about processes used in industrial production and algorithmic design. Clos networks are a form of staged hierarchical decomposition (see section 11.5.6).

4.9 THE GENERAL PRACTITIONER PROBLEM

In the medical profession, patients will normally be asked to pass a gatekeeper when seeking medical help—an ordinary G.P in order to avoid wasting specialists' time.

Is this efficient or well-conceived? By putting together a few elementary Promise Theory considerations, we quickly see that the assumptions behind this approach a scale-dependent and therefore not universally true. In the worst cases, this approach may become the least successful. Medical systems in most cities around the world are, by now, antiquated with respect to modern technology, and there is enormous scope for improving them with respect both to speed and patient care.

We can begin by recalling the essential promise binding between agents. It's useful to describe these timelines for offers, withdrawals, and revisions to promises as an example. For medical emergencies, patients P impose themselves onto doctors in Accident and Emergency departments $D_{A\&E}$:

$$P \xrightarrow{+\text{emergency}} \blacksquare \; D_{A\&E} \tag{4.105}$$

$$D_{A\&E} \xrightarrow{-\text{emergency}} P. \tag{4.106}$$

In more sedate circumstances, they have to negotiate an appointment by invitation. Doctors promise time slots indiscriminately, and patients may accept them. This is the calendar appointment phase. A conditional promise of service is then made, depending on the appointment, and the doctor promises to withdraw direct the promise of the open slot exclusively to P[45].

$$\pi_1 : D \xrightarrow{+\text{open appointment}} A_?, \tag{4.107}$$

$$\pi_2 : P \xrightarrow{-\text{open appointment}} D \tag{4.108}$$

$$\pi_1 : D \xrightarrow{+\text{appointment}} P, \tag{4.109}$$

$$\pi_2 : P \xrightarrow{-\text{appointment}} D \tag{4.110}$$

$$\pi_3 : D \xrightarrow{+\text{consultation}|\text{appointment}} P. \tag{4.111}$$

As long as the doctor has π_2 from P, the conditional promise is valid and exclusive.

Now that we can be sure that the patient and doctor will meet at the appointed time, we can turn to the substance of the meeting: the consultation service. A G.P can promise a certain level of general knowledge, i.e. if we break it down we might say the G.P. can keep more kinds of promise than a specialist. G.P.s may also play a secondary role as sales agents, selling private specialist services in the case of private practices. For this reason, the structure of promises is often as shown in figure 4.17. A G.P. may refer a patient to a specialist if deemed necessary. On the other hand, if patients know roughly what's wrong with them, by specialist category then this merely introduces a screening delay that could cost the patient and the medical system unnecessary time-wasting. Of course, it is possible that a patient may not know what is wrong with them, in which case they need to consult with a general practitioner, whose breadth of expertise is

then specifically called upon without abuse. There is a finite number of specializations: Ear-Nose-Throat, eyes, heart, cancer, surgery, and so on.

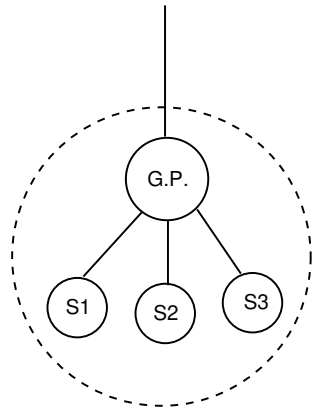


Figure 4.17: General Practitioners are used as dispatchers or 'load balancers' to protect specialists, on the assumption that specialists' time is more limited.

Once a patient has a dialogue with a doctor, there is an unspoken contract of service so we can drop the promise binding details for now and return to them if we need to. There may be repeated visits to the G.P., referrals to specialists including X-rays and testing, also including repeated visits, and follow-up visits—all of which incur and accumulation of time and cannot be performed in parallel. Doctors are a shared resource, so the appointment schedule is a *queue*, in the sense of chapter 12 of volume 1. Once we deal with queues, we pass from a transactional view to a flow probabilistic view over long times (see figure 4.18).

This is an important point: to optimize this system, we need to decide which promises we want to optimize for. That means which scale or specifically which timescale we choose for optimizing. Long term patient numbers may trade individual benefits for impressive statistics. Individual patient care optimization may lead to poor long term throughput. Some doctors may not question what they consider to be proper and ethical– but this goes to the discussion in section 2.3.6. A specific promise is needed to resolve this issue. If no promise is made, then nothing can be assumed.

We know approximately how queues behave, so we can determine to some extent where bottlenecks in this flow of patients will occur. Standards of patient care, and union rules, place a maximum number of visits per day, or a maximum service rate μ, per doctor. Patient arrival rates λ are not regulated, however, so the possibility of flash flooding is a real risk. Sudden epidemics or major incidents can quickly overwhelm the

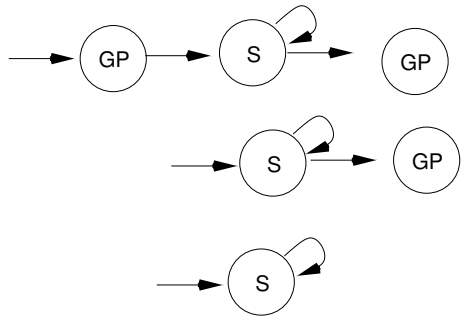


Figure 4.18: Patient consultation flows, with and without G.P. gatekeepers, or sales agents?.

process. In such cases, the normal procedure makes way for a simplified one of 'triage' that promises rather less than a regular consultation.

System scaling depends on the dimensionless ratios of the different scales, e.g. the patient arrival rates λ_{GP}/λ_S the service rates of G.P. versus specialists μ_{GP}/μ_S and the number if G.P.s to the number of specialists in each discipline: N_{GP}/N_S. So the system's scaling behaviour is a function:

$$f\left(\frac{\lambda_{GP}}{\lambda_S}, \frac{\mu_{GP}}{\mu_S}, \frac{N_{GP}}{N_S}\right) \tag{4.112}$$

and perhaps more parameters too. Here the rates are assumed adjusted for repeat visits. This function tells us that, in order to keep the same promises at the same levels, these ratios need to be preserved under ideal circumstances. Perhaps more interesting are the criteria one might promise for success:

- The total service time for each patient.
- The total service cost for each patient.
- The learning experience for doctors be exposure to cases.
- The level of fatigue amongst doctors.

It's hard to optimize for all criteria, and the complexity of the interaction flow shows that there is no obvious conclusion to draw about whether time is saved by omitting one visit to a G.P. However, but the main bottleneck of the initial consultation promises rather little, so one could easily imagine it being handled by an online questionnaire or even an 'artificial intelligence' based on some initial pre-emptive tests. This would bring patients closer to expertise more quickly, which might avoid some misdiagnoses and lost opportunities.

Clearly there is something more going on here that cannot be handled by the difference between imposition and invitation. Patients are basically under uninvited attack, by disease and accident. However, one might try to consider pre-emptive avoidance of disease. In some parts of the world, doctors are only paid if their patients are healthy. Would this theat or counter-incentive work to optimize? It has already taken some space to set up this complicated problem. We can now begin to ask the questions about whether agents, promises, and assessments are vulnerable to failures, faults, and flaws, and what can be done to minimize these issues, with respect to which promises. As we saw in game theoretic analyses of volume 1, there will be conflicts of interest in these choices, meaning that whatever happens we may be damned if we do and damned if we don't.

CHAPTER 5

SPACETIME CONSIDERATIONS

Everything that happens in a system takes place in a kind of spacetime—i.e. a basic characterization of differences between distinguishable states (space) and how they change (time), as well as the characteristic scales exhibited by different processes relative to one another. How we describe this arena for system behaviour plays a major role in how we visualize a system, and it's detailed properties constrain the possible behaviours and outcomes at a fundamental level. In this chapter, I'll sketch out some of the ways in which spacetime concepts lie at the heart of systems.

This chapter may be principally of interest to engineers and researchers who are looking to formalize a 'physics' of interactions in systems of all kinds, and to those looking to formulate a programming representation of a dynamical process. Space and time are the *configuration space* in which all phenomena occur. The nature of spacetime varies from system to system. In some cases, it refers to the familiar three dimensional human world, in other cases it could be a network or a mechanical pipeline.

Assumption 2 (Spacetime in Promise Theory)**.** *The term 'space' has a strict meaning in mathematics, but we bend that definition here. In a Promise Theory sense, all of what one normally calls 'space' is formed from a network of agents, and all of 'time' is registered as changes to observable states.*

Most of us imagine processes from the perspective of the past flowing towards the future. This is the model of causality we have been taught since the time of Newton, and Leibniz. Yet the mathematics of symmetrical processes allows for causality to be considered both forwards and backwards. In radio wave science, so-called advanced or retarded wave formulations may be used to model causation, for example. These offer different perspectives on what drives a system between measurable events in space and

time. Various principles may be used to make use of the characteristics of space and time as dependencies to offer a perspective on system behaviour.

Definition 72 (Trajectory (informal))**.** *The trajectory of a system is an ordered path through spacetime agents from an initial state to a final state.*

Scale invariance, or scale-free behaviour, allows us to characterize dynamical properties of a system as an approximately universal and continuous function of some scale parameter. Type 2 scaling is an invariant characteristic, i.e. it is not tied to a specific scale (Amdahl's and Gunther's laws compare values at N threads to a single thread rather than to an arbitrary increase in size. We saw that these laws are not scale invariant in section 8.7.3. In type 2 scaling, we return to the idea of a continuum flow approximation, and consider its generalization from one dimensional flows to the include the involvement of more spacetime dimensions.

5.1 THE ROLE OF SPACE AND TIME IN PROCESSES

Time and space (size) are intertwined in sometimes unexpected ways. In an intentional system, time does not only affect dynamics, but also semantics, because observers perceive and interpret behaviour according to their own notions of time.

The speed of propagation of influence, i.e. communication, is what connects distance and time together. Processes measure time, and interactions, which mix scales through space, also lead to the emergence of multiple scales in time: as systems become larger in space, they often become slower in bulk. Since time is an important part of whether a promise is considered kept, scaling can lead to a new subtlety. Large systems might simply fail to keep their short term promises. Conversely, the accumulation of interactions from a larger system may place burdens on resources, so that what works for the short time, might not work in the long run.

Beyond these fairly obvious examples, there may be complicated spacetime involvement in which the geometry of a system funnels work (through dynamical constraints) in a particular way, and a system needs new policy tradeoffs (i.e. new semantics) to handle the new behaviours. It is a reasonable hypothesis that such changes in scale, whether intentional or unintentional, are responsible for many unexpected 'failures' in systems. Observer time interferes with system time, when they interact.

All non-trivial systems develop in time, but some are isolated from their surrounding space. When a system is embedded in space and time, the geometry of the surrounding space constrains the flows that feed input and deliver output.

We should not be too blinkered by the idea of workflows, in particular 'queues': the narrative we have created around workflow is linear mainly because human thinking in

linear, but there are plenty of examples of systems who input or output are multidimensional:

- Water flowing into a drain from a 3 dimensional reservoir.
- Animals feeding around the carcass of prey.
- A loudspeaker radiating sound into three dimensions.
- Traffic flowing through a street network in two dimensions.
- Aircraft lift transducing airflow in one dimension to perpendicular dimension.
- The storage of data streams in two dimensional disks or three dimensional crystals.
- The representation of one dimensional data on two dimensional screens.

5.2 HIERARCHICAL NETWORKS OF PROCESSES

Processes bindings form networks, and networks have large scale structure. It's a conspicuous fact that hierarchical systems dominate the world on all levels, from the very small to the very large. In inanimate terms, this is blessing, else systems would surely be impossible to comprehend. In the human realm, we are less certain about hierarchies, as they often represent power structures that disenfranchise individuals. These are issues to be understood rather than despised—only then will we go beyond idle opinion. Great progress has already been made thanks to the language of Promise Theory.

Hierarchies play several roles. The most obvious characterization is perhaps to view hierarchies as the natural generalization of a star network (see figure 5.1). A hierarchy often has has a natural tree structure, which is a kind of staged centralization—or delegation. Instead of having only a single hub, a hierarchy has several hubs, which handle central functions locally in 'branches'.

In spite of the brittle nature of the networks, broken by a single node removal, the scaling of hierarchies is quite efficient. The cost of maintaining relationships is minimized, as the hubs have $O(n)$ connections, where n is the number of subordinates, at each stage, whereas a mesh may have up to $O(N^2)$, where $N > n$ is the entire network node number. A lot of rhetoric is used to defend centralized and decentralized designs. They have quite different properties—and a lot of confusion stems from the way they scale. A decentralized system may scale as a single centralized entity at a larger, slower scale. At each scale, a centralized system can reach an equilibrium more quickly than a mesh network because it is only $O(n) \leq O(N^2)$ in promises. However, as the scale increases, the interior functioning of a central hub must slow down in exactly the

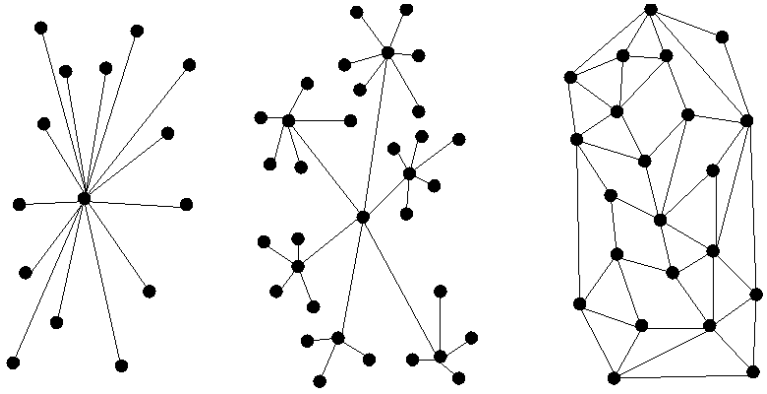


Figure 5.1: Network architectures, from a) a centralized star network, to b) a Cayley tree or delegated star network or hierarchical hub model, to c) an amorphous mesh. The centralized models are spanning trees with single points of failure, while a mesh has alternate routes between any two nodes.

same manner as a distributed system—the $O(n)$ scaling then applies at a slower rate to its exterior satellites (see section 5.12.2).

Example 57 (Data consensus)**.** *Data consensus clusters, like Raft and Paxos, are sometimes referred to as decentralized, but they are strongly centralized and monolithic on the scale of their duplicate equilibrium configuration. Since no node 'moves' independently of the others, they are in lock-step as a single, slower entity, and their sluggishness is a result of the scaling of the equilibrium time for the cluster.*

5.3 THE TIME-SERIES MODEL OF PROCESSES

Our received view of time—as a river of events that moves everything from past to future at the same rate—is a side effect of living in a rather slow world, which is close to us, and which we see with no perceptible delay. In IT, we cannot rely on this privileged view, and we need to rethink time by going back to the basics of how we measure it.

5.3.1 EVENTS, CLOCKS, AND PROPER TIME

In an information theoretic sense, an *event* is an observation of change in data sampled from a source[SW49, CT91]. In the Einsteinian sense, this signal is a tick of a clock that an observer samples. When the tick originates from within a process (e.g. a CPU kernel tick), this defines a notion of 'proper time' for the local process, indicating an advance in the state of the process. When there are multiple agents involved, working together, the language one often speaks of 'vector clocks' in IT, referring to Lamport[Lam78].

Other agents, external to a ticking process, may observe changes in it differently, either because they lack access to observe the changes or because the sampling of the changes require intermediary processes like message passing to propagate the changes from source to receiver. Thus the proper time experienced by an agent may not correspond to the exterior time generated by the sampling of remote events.

Example 58 (Thunder and lightning)**.** *When lightning strikes, observers in different contexts see and hear it at different times. Observers very close, that cannot sample faster than a certain rate, may not be able to discriminate a difference between the flash and the thundercrack. Light travels so fast the few agents can detect a delay in the signal, so they conclude that the flash occurs as 'the same time' (during the same sample). But sounds travels more slowly, so agents at different distances can discriminate the time at which the sound reaches them. If they synchronize their watches using light, they will measure different times for the sound—but the event happened due to a process that took place in a single location, over a tiny fraction of a second. What observers sample is not always a high fidelity representation of what happened at the source.*

Using the language of Promise Theory[BB14a], we can define time from two perspectives. For convenience, we'll make an identification between the concept of an event E_γ and a line transaction L_γ in a system log, or a data point recorded in a timeseries database D_γ:

Lemma 8 (Events count time)**.** *The emission of an event or 'log line' $E_\gamma = \{L_\gamma, D_\gamma\}$ is a tick of interior time clock.*

This should be obvious, as events are changes that get noticed. We can now define interior (proper) time and exterior (relative) time:

Definition 73 (Interior time of process S_i)**.** *An independent count of ticks originating from within a process S, cannot be observed by any exterior agent $A_?$, unless promised and reported:*

$$S_i \xrightarrow{+tick_i} A_?. \tag{5.1}$$

Interior time is the image of processes that originate within the boundary of agent S_i. At scale, we can consider superagents of any scale, so interior time scales and changes in meaning according to our definition of local.

Definition 74 (Exterior time of process S_i)**.** *An independent count, by a remote receiver R, of promised ticks (observed and aggregated from any number of sources on a watchlist) that increases for each sampled event arriving from a exterior process source S_i.*

$$S_i \xrightarrow{+tick_i} R \tag{5.2}$$

$$R \xrightarrow{-tick_i} S_i \tag{5.3}$$

$$S_i \xrightarrow{+tick|tick_i} A_?. \tag{5.4}$$

Exterior time is attached to remote processes that may originate on any scale. The recipient R that samples events may itself be of any scale, with associated loss of certainty about the definition of its interior clock counters, but ideally R would use a single source from an elementary agent, for precision.

On the timescales of computers, in our daily lives, this sounds straightforward, but the processes that calibrate our normal idea of time (the system clocks) are not faster than the sampling processes we are trying to discriminate by. This leads to a breakdown in the normal assumptions of universal time for all, and forces precision agents to go back to basic definitions of time in order to trace processes in band.

5.3.2 CLOCKS AT DIFFERENT SCALES

We cannot avoid the effects that scaling has on clocks. Even atomic clocks may not be considered atomic, in the transactional meaning, on the scale of subatomic processes. The lesson that Einstein taught us is that processes need to embody their own clocks, as single reference sources of truth.

Example 59 (System clock, e.g. Unix)**.** *The system clock, provided by most operating systems derives from a shallow hierarchy of exterior time services, based on processes that promise* approximate *alignment. The clock timer C is an independent agent, which promises a counter (UTC) to processes P_i,*

$$C \xrightarrow{+UTC} P_i \tag{5.5}$$

$$P_i \xrightarrow{-UTC} C \tag{5.6}$$

Using this as a conditional dependency, processes can then promise timestamps based on the interior counter

$$P_i \xrightarrow{+timestamp|UTC} R. \tag{5.7}$$

Note that the coordination between duplicate redundant clocks is weak. A time service like NTP, provided by agent N, may be used to periodically align independent clocks at a layer of the hierarchy above each system clock:

$$N \xrightarrow{+UTC_{NTP}} C \tag{5.8}$$

$$C \xrightarrow{-UTC_{NTP}} N \tag{5.9}$$

$$C \xrightarrow{+UTC|UTC_{NTP}} P_i \tag{5.10}$$

$$P_i \xrightarrow{-UTC} C. \tag{5.11}$$

Each clock is independent, so it is only meaningful to compare two timestamps from the same clock. Moreover, the relationship between timestamps and process ticks is indeterminate, since process ticks are halted relative to the system clock during timesharing. The use of timestamps in network protocols should be considered unreliable, and only for round trip comparisons.

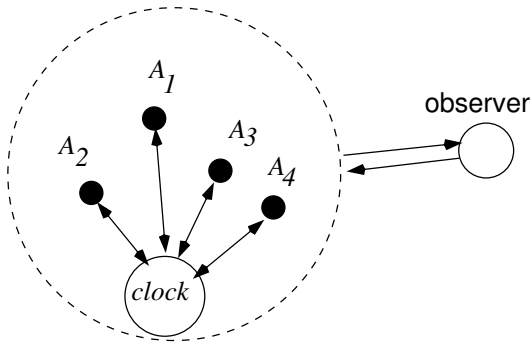


Figure 5.2: A single agent, with a reference clock can be scaled into a superagent provided the agents within promise to coordinate their behaviours. Thus a collective formed from independent sources can act as a single reliably ordered source, but at a cost growing like N^2 in the number of agents.

Example 60 (Monotonic counters)**.** *Interior process time can be obtained by incrementing a counter by an atomic operation. A process P passes a value v to a counter C, which is a persistent variable, and promises to increment it as certain milestones are*

passed:

$$P \xrightarrow{+v} C \tag{5.12}$$

$$C \xrightarrow{-v} P \tag{5.13}$$

$$C \xrightarrow{+t=(v+1)|v} P \tag{5.14}$$

$$P \xrightarrow{-t} C. \tag{5.15}$$

If independent agents need to coordinate their clocks, they can build on a single source of truth, by appointment to the role (see figure 5.2), essentially transforming interior time into exterior time.

Lemma 9 (Interior consensus of clocks)**.** *The promise to share interior time T_i from S_i, to an agent R_j, with interior clocks T_i is equivalent to the problem of data consensus between clock ticks.*

This suggests that clock synchronization by voting in band ('realtime') will lead to a significant delay in the rate of time that can be promised as agreed ticks by an entire superagent. This increased 'mass' of agent clocks will slow the rate observable by an outside sampler.

Lemma 10 (Aggregation of clocks)**.** *The aggregation of multiple sources of interior time T_i from S_i, by an agent R_j, with or without consensus, is not one to one with the interior time of the receiver.*

The proof of this may be seen from the Law of Intermediate Agents[BB14a], which tells us that, if there are agents in between the source and receiver (which is nearly always the case), then no promises are transferred automatically. We need a chain of delivery promises to form an expectation of what we are seeing. The outcome of this is that every agent may see a different arrival order, assuming that it can distinguish between data transmissions. This is a well-known result. Conversely:

Lemma 11 (Promised Order Propagation)**.** *The order of a sequence of data, from a single agent, can be promised by virtue of a single clock or counter.*

Note that this does not imply that order will be preserved, only that there is a set of promises between sender and receiver that can be made to transfer the order information (e.g. by numbering packets). This is well known in 'reliable' data communications, like TCP. Without proof, let's acknowledge that the relative order of data can indeed be transferred reliably between a sender and a receiver, if there is a promised order at the source (figure 5.2). This requires the introduction of a co-dependence between sender and receiver, and a detailed explanation has been given in [BBKK18]. Examples include the well-known TCP protocol, and other more exotic variants.

This does not imply that data are necessarily observed in the same order by source and receiver. Once data leave the agents that are entangled in this way, the promise of order is not preserved, because all agents are causally independent.

Lemma 12 (Promised Order Propagation)**.** *Data exchanged without conditional sequence promises may not be sampled in the same order as they were promised.*

The implication of this lemma is that predictable coordination of sequences as invariant features between agents is expensive and unreliable: it does not happen unintentionally, without chains of interdependent promises. This is a rather damning result for monitoring that relies on timestamps for its depiction.

Order promises can be kept by labelled (sequence numbering) or by waiting in lock-step for changes one at a time. The independence of agents, and our inability to make a promise on their behalf, means that data passing through multiple intermediaries are *independent deliveries*. Even a single agent cannot be forced to deliver data in order, unless it has promised (fully intends) to do so in advance, with full observability of the payload (and a receiver that can sample at the Nyquist rate[CT91, Bur16c]). Expecting clusters of agents to preserve order, over possibly parallel routes is even more unlikely, without prior intent. This can be expressed by saying that unless there is a single clock that determines when packets will be sampled by a receiver, the order will not be preserved. The default is that an incoming queue of samples serializes them in a random order, without a surviving chain of dependency.

We can arrange for such a single clock to be authoritative (like a shared memory counter), but this requires agents to make promises to abide by the order, which in turn requires cooperation from end to end throughout a channel, to preserve identity serially and atomically (one agent sampled exclusively at a time). Another approach to agreeing about time is to bind clocks in lock-step to form a co-dependent relationship between agents, known as *entanglement*[BBKK18]. This is used, for example, in TCP's SYN-ACK protocol. It promises synchronization at a possibly unbounded cost in terms of interior time ticks.

Lemma 13 (Promised Order Propagation)**.** *The intended order for events originating from more than one S_i may only be promised by interior cooperation at the source, and assessed uniquely by an agent R with observational capacity according to the Nyquist law. Each rescaling of aggregated time ordering introduces new uncertainty according to an observer's clock.*

There is no unique intent, for a collection of autonomous agents, unless the multiple sources subordinate themselves by cooperation to a single agreed order, but any attempt to coordinate between the agents (and thus act as single superagent making a common promise) would result in a change in the ticks observed by R, unless the sampling

resolution of R is much less than the exterior time needed to assess interior latency of interactions for agreement.

The conclusion of these extended remarks is that there is no single clock by which to define the order of events between different hosts. This is essentially because unrelated processes have no common time. The whole idea is meaningless. What observers often seek is a picture according to their own sense of time (observer time) that integrates different processes into a picture of the moment as they perceive it. Alas, that impression cannot easily be reconstructed later, even perhaps with a detailed 'post mortem', as it relies on anchoring to out of band processes that were not measured. If we introduce a single source of time for a collection of hosts, by forming a superagent (with all necessary interior cooperation), each agent within, we can define a single reference time, but it is not the proper time of any process.

5.3.3 INFORMAL IDEAS ABOUT TIME

Example 61 (Common assumptions of system time)**.** *A commonly held belief is that, in interactions like network protocols, we might define time in a number of way.*

- The 'actual' time*: there is a single source of truth, by international convention, which is the official value of UTC. This time standard exists, but is only obtainable with latencies that render it approximate. Through a hierarchy of services, like NTP, local system clocks promise to approximate this time and to count independently on their own at approximately the same rate. These rates cannot be verified, so in practice the closest we can obtain is the current value of the local system clock, which belongs to localhost.*

- The observed time*: This is a timestamp rendered by sampling the system clock, so it is relative to localhost's assumed time standard and has no significance beyond the agent that sampled it. The observed time may not be monotonic, for example if clock drift corrections occur in between samples of the clock. System time may therefore go backwards or forwards at random, over extended processes.*

- The publication time*: Timestamps may be shared between processes, or recorded, incurring additional processing delays. The resolution of a timestamp may be quite low, allowing processes to absorb processing delays, but publication times are always later than the timestamps they promise, e.g. the timestamp when a log entry is written is always later than the timestamp of the log entry.*

- The receipt or sampling time*: If timestamps are shared between agents, e.g. in recording data in a log, or transmitting data across a network, the published*

> *timestamp belongs to the sender S's clock, and the receipt time belongs to the receiver R's clock. These two times are causally independent and their comparison is strictly meaningless. If all clocks promise approximate alignment, the difference between published and received timestamps may promise an accuracy whose uncertainty is approximated by the Pythagorean average of the uncertainties of the two timestamps at S and R.*

Clearly, no protocol (except NTP) passes information about its clock time uncertainties, so network time falls foul of the Intermediate Agent law.

5.4 THE ROLE OF TIMESCALES IN PREDICTABILITY

The purpose of monitoring is to be able to explain behaviour and even predict problems in advance. Without predictability, monitoring is little more than somewhat arcane entertainment[HL93, Hog95, Hel96, PS06, SBS99, DF98]. One assumes that, by learning about the past or by building a relationship with system behaviours in band, we are able to predict something about the future behaviour. This, in turn, assumes a stability under the repetition of patterns.

Definition 75 (Predictability)**.** *A system that has stable and repeated observable behaviour, on a timescale much greater than the sampling rate, may be called predictable.*

It's, of course, paradoxical that the time when most users want to monitor systems is when they are *least predictable* and providing observations of no value.

5.4.1 SEPARATION OF TIMESCALES

We can make another observation about what happens in interactions. The principle of separation of timescales is a design principle for interacting systems, based on the observation that dynamical influence causes timescales for change to mix. In earlier work, I've referred to this as the most significant principle for engineering—more important than the separation of concerns based on semantic (functional) separation, such as data normalization or 'class', which is the norm in Computer Science. Briefly, it says:

Principle 5 (Separation of timescales)**.** *Functional systems modularize robustly and effectively when processes with different characteristic timescales are weakly coupled.*

By 'robust', we refer to 'stability'[Bur4 b]. This principle makes a connection to the related problem of data consensus, which is a strong coupling regime that maintains data consistency over average timescales.

5.4.2 DYNAMICAL COUPLING DEFINED

The foregoing assertions can be justified by looking at what coupling strength means for interacting agents[46]. Phenomena that promise changes on very different timescales interact only weakly and can therefore be treated as logically separate. By contrast, agents that promise couplings on the same timescale may influence one another and therefore belong to the same class of phenomena. In terms of the foregoing definitions, we can state the meaning of separation more strongly, as a theorem:

Theorem 2 (Separation of causal influence)**.** *As the ratio of timescales becomes large $T_R \gg T_S$, the effective coupling tends to zero*

$$e \to 0. \tag{5.16}$$

tends to zero (weak coupling).

To prove this, suppose a series of partially ordered events at an agent S yields a series $E_\gamma, \gamma = 1, 2, \ldots$. Suppose a source agent S transmits the events, which are aggregated into superagents $E^{(n)}(E_\gamma)$ of dimension n, by the receiver agent R,

$$S \xrightarrow{+E_1, E_2, \ldots E_n} R \tag{5.17}$$

$$R \xrightarrow{-E_1, E_2, \ldots E_n} S \tag{5.18}$$

$$R \xrightarrow{+\alpha_E | E_1, E_2, \ldots E_n} A_? \tag{5.19}$$

so that the dimension of the information is reduced by a factor of n by R:

$$\alpha(E_1, E_2, \ldots E_n) \to \mathbb{R} \tag{5.20}$$

$$\left| E_1, E_2, \ldots E_n \right| = n \tag{5.21}$$

$$\left| \alpha_E \left(E_1, E_2, \ldots E_n \right) \right| = 1 \tag{5.22}$$

The average time between events, as assessed by R's clock, may be denoted

$$T_S \simeq 1 \tag{5.23}$$

$$T_R \simeq n. \tag{5.24}$$

So R assesses S's timescale to be 1 and its own timescale to be n:

$$T_R \geq T_S. \tag{5.25}$$

Thus the average interarrival times for the queue in (5.19) $\lambda_R \sim 1/T_R$, etc, satisfy:

$$\lambda_R \leq \lambda_S \tag{5.26}$$

and the effective influence, in fraction of messages received compared to messages sent is expressed by a coupling constant:

$$e \sim \frac{\lambda_R}{\lambda_S} \leq 1. \tag{5.27}$$

In a strongly coupled system $e \to 1$, timescales converge to the shortest timescale of the interacting parts, making systems busier and more work intensive. The utility of this observation is that, if one separates causally independent parts of a system into superagents that make weaker promises to one another, any observed correlation between phenomena, that exceeds expectation, can be considered coincidental or potentially faulty. This can be detected by a change in the proper time event rate, measured by some agent within a system. This principle therefore has significance to the use of observation for detecting faults and design flaws in systems. It tends to maximize the signal to noise ratio between promised and non-promised behaviour[Bur4 b].

5.4.3 COUPLING STRENGTH, MEMORY, AND CONSENSUS

The concept of knowledge is already more uncertain in a distributed system than in a local system with random processes. Lamport's papers about seeking the homogeneity of data sources is effectively a monitoring problem in reverse. A collection of agents monitors one or more sources and tries to equilibrate the knowledge they promise. Data consensus is a conditional promise of policy-determined values (called quora), based on inputs reported from sources S_i:

$$S_i \xrightarrow{+E_\gamma} R \tag{5.28}$$

$$R \xrightarrow{-E_\gamma} S_i \tag{5.29}$$

$$R \xrightarrow{+\text{Quorum}(E_\gamma) \mid E_\gamma} A_?. \tag{5.30}$$

This strong coupling, represented by strong dependency on data from a complete network of dependencies, demonstrated that time and order of events are fundamental obstacles in a system of distributed computers in which observation has a finite latency (usually agents that are spatially separated)[Lam78, Lam01, OO14]. The topic touches on the relativity of simultaneity, and how to make sense of differing views about what causes what.

The relationship with time is revealed by the 'FLP result'[FLP85], which exposed the essential impossibility of consistent distributed knowledge in an uncertain 'asynchronous' environment. In an asynchronous message-passing system, source or delivery agents may delay messages indefinitely, duplicate them, or deliver them out of order. In other words, there is no fixed upper bound on how long a message will take to be received. A consensus policy promises:

- All trusted nodes promise the same result (a non-local agreement).
- All trusted nodes will eventually promise a result.

Some approaches to working around the limitations of asynchronicity play with strong synchrony promises in order to eliminate these uncertainties[BBKK18].

In an asynchronous interaction, each agent's proper time may be used to define 'time-outs' to receiving data to keep a process from waiting for ever for strong dependencies. Timeouts are a workaround that weakens the effective coupling strength of an interaction, by effectively measuring latency in interior time. There is no unambiguous meaning to a timeout, except the presence of a potential fault. Latency (round trip time) is the only covariant measure in a relativistic system, because it's one of the few measures that has a purely local meaning.

In Promise Theory, the intermediate agent theorem is the analogue of that result: it says that whenever you rely on agents that are not yourself, to acquire or deliver information, it no longer promises what its originator intended. And if a remote agent promises something, but doesn't promise it to you too, all bets are off and there is a quadratically growing cost of verifying. In monitoring, we are not usually interested in a majority view, rather we are interested in what happened specifically at what we believe was the certain place and time of origin (though this is also subject to uncertainty). We are sometimes interested in a statistical view (which is not the same as a consensus view, because it admits and even measures the statistical uncertainty of variations. If certain nodes lie (sometimes called Byzantine behaviour) we want to know about it, not merely cover it up.

There is clearly overlap in the concepts of distributed information, but monitoring seeks a picture of actuality, rather than a cover-up operation to brush uncertainty under the rug of consensus. Software Engineering therefore has a conflict of intent with monitoring: it wants to assure complete dependability (promises always kept) by invoking protocols 'in band', whereas monitoring is trying to expose when promises are not kept, to 'out of band' human observers. These are the issues we need to deal with in describing observability.

Readers may feel that the problem of distributed ordering has been solved by distributed consensus systems like Paxos and followers[Lam01, OO14], but this is not the case. Consensus systems do not promise the source order of observations, but rather an average order by which observations are reported, which is a policy decision.

5.4.4 MEMORY PROCESSES

It should be clear that memory is required to stabilize values from multiple sources. To integrate results from several sources, and to replace then with an agreed result requires

temporary memory, over the necessary clock ticks of proper time—at least as much memory as there are source dependencies for each outcome. The role of memory and its reliability also play a role, but I don't want to discuss that here. In most IT systems, memory unreliability is negligible.

5.5 THE OBSERVATIONAL SAMPLING MODEL

Given a stream of trusted values reported by agent interactions, the usual response is to try to build a timeline for a system as a movie recording of past history, using a panoramic lens. This throws us a number of questions about how often one should sample data.

5.5.1 SAMPLING RATE

The naive view in the industry is that one should collect as often as possible. Basic information theory constrains our ability to extract information from data. Many engineers feel that the virtues of fast sampling are indisputable, just as the citing of many decimal places leads to increased accuracy, but neither are true (for the same reason). Excessive use of high resolution sampling is a senseless arms race (a watched pot that never boils). Continuous high density sampling of a non-existent signal is not helpful. Nyquist's theorem tells us that we can only know fully of changes that occur half as fast as the rate at which we sample. Shannon's theorem tells us that our information about the system only increases when changes are observed.

Regarding the order of events, it's common to rely on an independent clock service, located within a network to try to synchronize clocks to some calibrated count, and then rely on the homogeneity of manufacturing in chip-sets that count time at a more or less similar rate. Traditional clocks services count time in seconds, but this sampling rate is much too infrequent to distinguish processes in modern processes, where nanosecond timing discriminations are becoming.

Physics tells us that relying on the counting of an exterior agent is futile when clocks are located in regions of very different gravitation, or when they are moving with respect to one another. This already has to be corrected for in satellite systems. The same effect applies if agents are in virtual motion with respect to one another. Only the interior proper time of a process can be relied upon for comparisons.

Local measures of observables have to be aggregated into coarse grains in order to measure them against one another. Histograms of observational distributions are usually the best we can offer in terms of observability. But distributions only tells the past probability of behaviours in fairly static cases. When we most want to know about a system, that's when it's hardest to understand. [47]

Time-like changes are normally assumed to be instances of what may potentially be significant events. The result is that human operators get excited by graphical traces that suddenly rise or fall—which has an undoubtedly hypnotic appeal, but means nothing without a larger context. Sliding windows are often used to detect gradient changes in time-series. Ensemble averages are used and even forced in data distribution and consensus processes (see figure 5.3). In other words, aggregation over time (not space) is a necessary part of the learning that provides context for prediction.

Principle 6 (The sampling rate)**.** *The sampling rate for a variable should typically be about half the auto-correlation time for a variables in order to detect meaningful stable variation.*

This is the timescale suitable for learning. For the purpose of anomaly detection, one might see a sudden change in timescale as a result of an unexpected coupling. Faster sampling could then be introduced on suspicion of a transition in behaviour—just as biological heart rates and attention spans quicken under stress. Recording and storing reams of data that are zero or constant cannot be in anyone's interest. Such data is compressible. It contains no new information. The potential problem with that approach is that the cost of sampling is not free; the impact of sampling on the system may become significant. Some authors have advocated such adaptive sampling[pap19]. One then has the decision about which part of the sampling process to scale back: the act of measuring on each local process has one cost, the act of aggregating the samples in some central repository has another cost. Neither of these is easily controllable, since multitasking operating systems make the sharing decisions to allocate cores, interrupts, network transmissions, and tasks quite opaquely. It may be difficult to assess which is the greater evil: uncertainty due to adaptive sampling or uncertainty due to oversampling or undersampling.

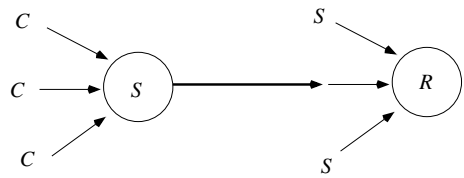


Figure 5.3: Aggregation of observations from multiple sources can happen at any node in a distributed process. When causal influences come together, in this way, the confluence point becomes an effective observer of the sources that feed into it. Observers are not only human!

5.5.2 SHARED RESOURCE COUNTERS (KERNEL METRICS)

The consequence of lemma 13 is that scaling of observations causes not only a reduction of information transmission rate, but possibly a loss of information about the origins of shared assessments. Resources counters, computed from the aggregation of data (like most of the kernel resource metrics typically recorded in popular monitoring tools), erase details that belong to their higher resolution origins, in an unrecoverable way (see section 12.6). There are many such recorded values in timesharing computer systems, because they are useful mainly to the sharing agent. The fact that they are shared with separate processes is a mixed and slightly misleading blessing. It's sad to see so much effort expended in sharing noise to observers desperate for insight. I think we can do better.

Shared measures erase the information about data origin, and thus such collective phenomena cannot be traced backwards to an appointed cause. For example, measuring the load average for a computer cannot determine which programs caused a spike in the load[Coc06]. This is not, on the other hand, a reason to retain every individual data characteristic for ever, because the opposite is also true: some data cannot be exposed without computing those high level functions.

5.5.3 INSTRUMENTATION OF PROCESSES

Our symbolic representations of processes are algorithms expressed as program 'code'. In a distributed system, programmers have begun to instrument workloads for interprocess communication using service meshes and centralized logging services. These do not reveal behaviours interior to the processes, but may provide traffic patterns by which to hypothesize about process behaviours and intentions.

For process tracing, one needs to be in the inside of the process, where the interior time ticks. From the foregoing discussion, it's clear that the system clock service is not the correct measure of time for process diagnostics. The proper time of a process is measured by the number of program counter transitions or program steps incurred by its execution. This count is significant because each increment is the result of an explicit causal jump instruction. The program counter's value can be promised at each instruction in code for debuggers to trace, forming a causal set of correspondences between program locations and increments. However, attempting to share these proper times between processes is meaningless.

The system UTC clock, even an approximate local copy of it, comes from an independent agent, and although shared between many processes its increments do not correspond to single channels of causation. Rather they represent only the implicit increase of entropy in collecting from all processes. When processes are timeshared, they may be halted and interleaved in complex ways.

5.5.4 SIGNIFICANT EVENTS AS SPATIO-TEMPORAL SIGNPOSTS

As discussed above, the major drawback in time-series thinking, for a distributed system, is that there is no unique meaning to the order of transactions originating from different sources when they are aggregated from different locations. There may be a lack of an obvious or agreed coordinate system. Observer's might have to rely on ad hoc documentation of events, without a clock or measuring stick to calibrate them.

Each observer in the universe sees events from their own perspective. The lightning bolt and the thunder arrive earlier for some than for others, because different processes propagate information at different rates, over different routes, and with different latent delays. Consensus is expensive, heavy handed, and its goals are different to the goals of observation.

Moreover, we have no uniform metric for time other than the exterior clock, which has issues of its own—it doesn't represent local causation except for the process that generates it. The question we want to answer is: what was the reason for an event E? i.e. can I infer the condition, state, or quality of the system from this event, based on what I have observed beforehand? Times are not causes.

A better approach, based in interior instrumentation is to create a semantic counter that traces a distributed process that we can trace backwards to causal origins. Samples can be taken when something is found to be significant within the context of the process itself. This is the proper passage of *significant* times. Regular sampling of processes is not an efficient way to record them because processes may be busy or idle, etc. This is what system logging enables—but the opportunity is usually squandered from a lack of a proper model.

Metric coordinates (clock times and numbered locations) are not helpful when we have no invariant measuring devices to define them by. The alternative is to use descriptive labels, or semantic coordinates.

Definition 76 (Significant event)**.** *An event marker, provided by a source process, that signals either an intentional change or an unintended deviation from expected state.*

Anomalies and faults fall into this. Processes that are not able to keep their promises may also be significant events.

When seeking the 'root cause' of an event, we really want to go back to prior events that were significant. The rationale behind this is that, in a stable system (one that we expect to be predictable), when all is as expected, unexpected events are most likely to be caused by previous significant changes and anomalies. Clearly, for every event there is a prior one, until we reach the very beginning of time (the system 'Big Bang'). However, we also have episodic boundary conditions that act as 'Little Bangs' for more constrained universes. Boundary conditions are semantically special events that we

attach special significance to—they are the origins of causation. We aren't interested in every intermediate change, only in prior events that make a splash.

We want to create a causal chain, a journal, something like a linked list. Instead of selecting data by voting, we can select values based on their perceived importance to outcomes of interest. This shifts the focus of policy from the intermediate aggregator of data to the observer: the observer is now expected to have a specific question it wants answered, rather than voyeuristically consuming data for entertainment.

5.5.5 FROM METRIC TO SEMANTIC COORDINATES: NAMING

The use of 'signposts' for labelling locations or process paths is a form of semantic coordinate assignment[Bur19c]. It traces back to before the time when maps and calendars were invented by human civilizations. Instead of imagining a regular, idealized coordinate grid, numbered impartially in an ordered sequence, signposts could be set up relating to events that were basically anomalous. Anomalous 'events' are less regular but highly recognizable things that we can observe (the big tree at the river, Mount Fuji, the Matterhorn, the year of the flood, the eclipse of the moon, etc). This provides anchors for accessing memory and plausible anomalies that might have exerted causal influence. We use these events as boundary conditions on episodic sub-processes. Sometimes, the role an agent plays in a promised arrangement is sufficient to distinguish and label it in interactions.

Example 62 (Unumbered BGP interfaces)**.** *The network routing protocol BGP sets up policy based relationships that point coherent classes of IP address to other coherent classes, allowing a scaled concept of regions to point to one another by a system of signposts. A routing table within each autonomous routing region (superagent), known as an 'AS', points the direction to send data, forming a self-organizing map. Conventionally, each interface that points to an exterior location is assigned a name in the form of an IP address. However, this is redundant and only leads to configuration complexity. When two agents are connected by a single wire, naming the ends of the wire is redundant, because they know how to find one another, and the data they exchange are sufficient to identify their roles in a process. Thus, omitting explicit names actually brings a significant simplification to BGP.*

Descriptive naming (semantic coordinate assignment) is thus more useful than ordinal naming (numerical coordinates). The checkpoints and paths that participate in processes may not be invariants, and the numerical value of coordinates is irrelevant[48]. We may need to identify a repeated pattern to some degree of approximation in order to exemplify a general concept from which lasting knowledge can be derived. Anomalies that do not recur become effective invariants, in memory, because they are rare and worth

remembering. Featureless invariants (like empty space) are indeed the most invariant of all, but have such high entropy as to contain no information of significance.

The *significance* or meaning of a signal is a kind of 'heuristic inverse' of the (incompressible) information within it—a key for the value. The more information we need to characterize a room, the less stands out about it. If there is one part that dominates, the rest is negligible—hence the principle of signalling significance.

Lemma 14 (Significance vs information)**.** *Maximum entropy distributions contain no significant events: they are causally random, and all events are observationally equivalent. Minimum entropy distributions have the maximum significance, as they imply strong correlation.*

Entropy plays a subtle role in statistical distributions, and therefore in ability to infer meaning from data.

5.5.6 REVERSIBILITY VERSUS TRACEABILITY

Why juxtapose these two concepts? The ability to throw a system into reverse, undoing itself, depends on our ability to follow it unambiguously as it unfolds. We want to be able to trace our knowledge of a system back to know the cause of an effect. The intended outcome is programmed into it, but there are also unintended outcomes caused by environment leaking into causal pathways. Because of the culture of 'rollback' thinking in IT, which originates from database transaction semantics, IT often muddles the concept of traceability with reversibility[49].

We must distinguish between the ability for an observer to trace backwards from a sequence of observations to reconstruct the cause of a significant event, and the ability to roll the state of a system back to what it once was. For example, it's possible for an observer to trace the source(s) of a river, but it is not possible to reverse the river and roll it back to an earlier state.

In the former case, the enabling condition is for no origin information to get lost in the chain of unfolding events (see figure 5.4). In the latter case, the necessary condition for being able to undo a causal sequence is that agents of the system itself have to promise the inverse of every promise in the forward direction conditionally on an undo condition—this is an additional set of promises pointing backwards along the path, which is much more than an observer being able to trace knowledge of promises backwards. The necessary condition is insufficient even to promise the result: agent processes must also be isolated from external interference, else the precise inverse operations may be deflected off course by noise[BC11].

5.5.7 PARTIAL ORDER OF AGENTS AND EVENTS

As a process propagates by passing messages, the messages separate earlier times from later times on the process's own clock or counter. Suppose this is reflected in a sequence of messages, or lines in a log. In a chain of lines L_γ belonging to a single source S_i, we can define a countable metric distance between lines by the total ordering of the sequence, also in the language of promises:

$$L_1 \xrightarrow{+\text{precedes}} L_2 \xrightarrow{+\text{precedes}} \ldots L_n \xrightarrow{+\text{precedes}} L_{n+1}. \tag{5.31}$$

The observational binding is equivalent a more classical ordering relation $<$:

$$L_1 < L_2 < \ldots L_n < L_{n+1}. \tag{5.32}$$

where

$$S < R \quad \leftrightarrow \quad \begin{cases} S \xrightarrow{+\text{precedes}} R \\ R \xrightarrow{-\text{precedes}} S \end{cases} \tag{5.33}$$

We need to be cautious about the interpretation of these promises. Each agent is making a separate promise, but (by the law of agent autonomy) these agents cannot make the assessment or promise it to an observer who happens to be watching all of them. Each agent can make its own promise available to the observer, but it's up to the observer to order them in the final instance. This ordering may be come mixed with other orderings as data are aggregated. The promises indicate that the relationships are considered persistent

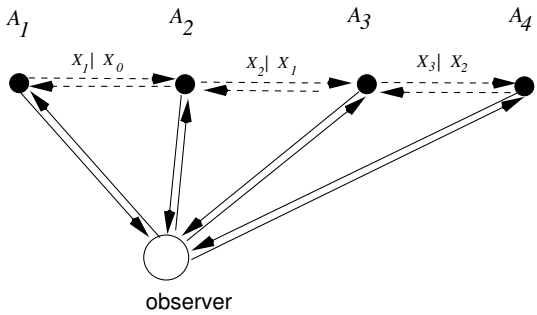


Figure 5.4: Causally ordered change in a process, and information observed about the process are two distinct things. As long as the observation of the process retains the order of the process, inferences about causality can be made, regardless of whether the system itself could be reversed. You can trace the source of the Nile, but you can't make the river flow backwards.

or even invariant by the promisers—not merely local assessments made on the basis of

spurious data. However, if at any time the events being ordered become indistinguishable, they can no longer be ordered. This can happen when data are aggregated without complete labelling.

Ordering reduces to the existence of (local) conditional promises, whose scope may extend to other observers in a scope σ[BB14a]. The n-th agent in a sequence; by the axioms of Promise Theory, we must have a chain of the form (figure 5.4):

$$A_n \xrightarrow{+X_n|X_{n-1}} A_{n+1} \tag{5.34}$$

$$A_n \xrightarrow{-X_{n-1}} A_{n+1} \tag{5.35}$$

$$A_{n+1} \xrightarrow{-X_n} A_n \tag{5.36}$$

$$\tag{5.37}$$

And, in general, we may consider a general scope for the above promises:

$$A_n \xrightarrow[+\sigma]{+X_n|X_{n-1}} A_{n+1} \tag{5.38}$$

Given such a set of promises, we can define a measure of observable distance between agents A_n and A_m by assessment. Again, we note that interior relativity makes distance an assessment by one agent about the relationship between itself and one or two others.

5.5.8 TRANSLATION OPERATOR AND NOETHER'S THEOREM

Let's make a slightly technical digression. If the agents were sufficiently homogeneous, we could consider an operator interpretation for Δ, as the generator of a translation on a set of states realized by the positions (like ladder operations):

$$\Delta|A_i\rangle \rightarrow |A_{i+1}\rangle. \tag{5.39}$$

In fact, for every kind of promise there would be a separate propagator, like a complete basis:

$$\Delta = \sum_\tau c_\tau \, \Delta_\tau. \tag{5.40}$$

The problem with this kind of interpretation is that is suggests the existence of a god's eye view once again. It takes the existence of a privileged observer to be able to order and rank the states in this way.

In classical physics, the continuity of the energy function with respect to spacetime is what generates conserved quantities like energy and momentum, thus allowing these quantities to be used consistently as counters for behavioural descriptions. We can see, from the Promise Theory, that this conclusion also follows from a privileged god's eye view of spacetime locations.

This tells us that it is the *assumption of continuity* by the observer that rationalizes the use of counting metrics, including jumps and changes in metric behaviour. If source observability does not reveal discontinuities in the assumptions amongst independent sources, the observer will not be able to discern that information merely by monitoring.

If we assume the conservation of b as an axiom, the ordering of b-influence must follow paths automatically, even when the agents make unsynchronized (asynchronous) promises, like a first order Markov process. In order to *explain* conservation and causal order over non-local regions, we need to extend the promises to be conditional on non-local neighbouring patches. Ordering information itself needs to propagate.

5.5.9 TRACEABILITY (INFERENCE)

Lemma 15 (Traceability)**.** *If an observer has complete information about promise causality, a process graph may be called reversible, i.e. for every pair*

$$S \xrightarrow{+X_S \mid c} R \tag{5.41}$$

$$R \xrightarrow{-X_R} S, \tag{5.42}$$

provided $X_S \subseteq X_R$. We can infer origin by using complementarity to interpret a reversal of causal tracing: $\overline{X}_R = c$ and $\overline{c} = X_S$, such that

$$R \xrightarrow{+\overline{X}_S \mid \overline{c}} S \tag{5.43}$$

$$S \xrightarrow{-\overline{X}_R} R. \tag{5.44}$$

If a set of agents A_n precedes another set A_{n+1} by a promise

$$A_n \xrightarrow{+X_n \mid X_{n-1}} A_{n+1} \tag{5.45}$$

$$A_{n+1} \xrightarrow{-X_n} A_n. \tag{5.46}$$

Traceability requires that O be in the scope of this chain, and that it assumes reversible semantics for X, as 'is followed by' (which is automatically interpretable as 'follows').

5.5.10 REVERSIBILITY (CAUSATION)

In the rare cases when systems can be made approximately deterministic and reversible, changes to a system can be potentially traced forwards and backwards in process time. This may not enable one to go as far as pointing out a unique 'root' cause, because aggregation points can introduce points of potential equivalence, but it will point out the causal sets that act as source (spacelike hypersurfaces) of the process.

Lemma 16 (Reversibility)**.** *Reversibility requires the much stronger criterion for information transmission, that there be a unique inverse for each step in a chain of agents:*

$$A_n \xrightarrow{+Inv(X_n)|Inv(X_{n-1})} A_{n-1} \tag{5.47}$$

$$A_{n-1} \xrightarrow{-Inv(X_n)} A_n. \tag{5.48}$$

This holds for any agent (or superagent) A_n.

The condition for forensic back-tracing of a system state (detection of cause) is that

- A complete chain of prior origin data be available across the graph.
- There should be no acausal loops in the process, else there may be branch alternatives (eigenstates) or divergent unstable behaviour.

Example 63 (Service lookup thunder and lightning)**.** *Consider the order of a process (figure 5.5) described in the following promises:*

$$A \xrightarrow{-dns} S \tag{5.49}$$

$$S \xrightarrow{+dns} A \tag{5.50}$$

$$A \xrightarrow{+relay(dns)|dns} R \tag{5.51}$$

$$R \xrightarrow{-relay(dns)} A \tag{5.52}$$

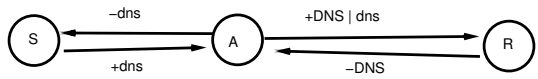


Figure 5.5: Causal order may be different from clock time. It is generated by prerequisite dependencies: either by underlying topology or by constraint. Agents can only trust directly agents that they are in scope of (in practice, their direct neighbours), as they have no calibrated information about the promises of agents.

Agent A promises to listen for a DNS lookup (a query, i.e. an invitation to reply). As long as this promise exists, it can be considered to be polling S for a response. S promises to provide DNS data, but it hasn't specified what or when. If there is a fortuitous match between the two, data will be passed from S to A. A, in turn, promises R to pass on the data it receives from A. It does a better job of promising conditionally, so it will only pass on fresh data from S when a reply is received, because the promise in (5.51) is conditional. R, in turn, promises to accept data, which can only happen after they arrive.

The effect in each interaction is to order data, but we don't know, from these promises, how many times data get passed between A and S, nor do we know how much latency is experienced by any of the agents.

The conditional promises (dependency) represent causal ordering. We can't say anything about the relative order of promise keeping unless it is constrained in some fashion. Often we rely on incidental or ad hoc serialization at a single observation point (a queue) to define the ticks of our process clock. The problem is that this serialization does not represent an invariant of the process, so it's unreliable.

5.5.11 METRIC DISTANCES

Order is important, when it can be distinguished because it allows us to measure intervals. We sometimes use intervals as significant measurements, though Einstein pointed out that intervals are not invariants, they are only 'covariant', changing with the system of measures we establish.

Principle 7 (Distance semantics)**.** *Distance is an assessment made by an observer with two complementary interpretations: distance suggests what might lie in between the bounds of the interval, or it suggests a measure of how similar two agents are, with respect to location in some criterion 'space'.*

The distance between two events L_γ and $L_{\gamma+\beta}$ is related to the ability of an observer to trace and count the number of similar events in between. The distance between events in a journal may contain implicit information about what happens in between, but it is not a substitute for the information itself. Metric distance is therefore a counter that pays just enough attention to agent properties to discriminate between them on the basis of label, and be able to count, but not necessarily enough to classify agents meaningfully. If events follow on as nearest neighbours this tells us something; if the same pattern is suddenly interleaved by more lines this could be an indication of an anomaly.

A histogram is a classification of multiple events that get counted and form a distribution. The order of the classes may (or may not) express a metric policy about how near or far events are when they fall into one of the classes. There is no a priori order to these classes, but there might be a distance. It's therefore an assessment policy of an observer to ensure proper classification according to a model presumed by the observer.

The proliferation of logs in IT systems means that they get receive disproportionate focus, in the hope of extracting far more than they are usually capable of representing. The variety and standard of logging is very poor indeed, in my view. What happens to order and distance relationships in logs after aggregation? There are many tools that imply log aggregation is a good way to bring together all logs into one location, but there is little discussion around the significance or usefulness of the result[DS05, BD07, WDSC07].

Aggregation of agents L_γ into superagents $\{L_\gamma\}$ may preserve or discard the order and interval distances between lines. Sometimes, data are not intentionally numbered by the sender and order is assumed by the order to transmission (e.g. in UDP transmissions). In that case, message packets may become reordered by network redirection, or loss. Some messages could be also be lost. Let's refer to the cases by the common terminology

- Reliable: promises all packets delivered in order.
- Unreliable: ad hoc, no promises about order or loss.

In either case the latency between transmission and final arrival is uncertain. Consensus of data is easy, because the data are point to point and there is only a single source and a single receiver for each message.

One way of trying to work around the law of intermediate agents is to build up the notion of *entanglement* between processes [BBKK18]. This takes several cycles of mutual interaction between a pair of agents, on some scale, as well as a small cache of local memory. Entanglement can transform partially reliable transmission of influence into fully reliable transmission at the expense of some added sub-cycles in the interaction.

The aggregation of messages without reference to the agent and the interior timeline that generated them implies that causal origins can never be traced backwards. Timestamps have no value, because they are unrelated to the process causation. We can thus show that a log may preserve the reverse tracing of causal history, but does not imply reversibility of state. We can trace a story back to where events played significant roles in the timelines of processes, but we can't necessarily reconstruct the states of those processes.

5.6 THE ROLE OF DIMENSIONALITY

The three strategies for output scaling are:

1. Serialization of workflow: make it faster (vertical scaling).
2. Parallelism: multiple channels to increase throughput (horizontal scaling).
3. Multiplexing processes to interleave sparse arrival processes (quasi-parallel sharing).

What may be surprising to information technologists, who work exclusively with serial workload scaling, is that the behaviour of a system can involve more than one spatial dimension, and result behaviour that is less like a queue and more like a funnel. The involvement of spacetime geometry in processes is not something that is currently

modelled in information technology. In a building, or a city, where people can move in two or three dimensions, the dimensionality is critical to avoiding contention and enabling throughput. Consider the evacuation of a building through a single emergency exit on the ground floor, and compare this to a queue to enter the building. Dimensionality is an interesting issue—far less rigid than we are taught to expect from the simple classical models we learn in school. At high speed, an agent's world is dominated by time, and a single vector of motion. In a world dominated by time, there is only serial processing, even with rudimentary parallelism, and the world is largely one dimensional. In a world dominated by space there is multidimensionality, and volumes can quickly become relevant in a continuum limit.

The promise of semantic space, or infrastructure, is to deliver one dimensional time saving services to its clients. That means it needs to deliver either high speed or multiplexing of resources, in the space of the clients. Our narrative about semantics, and promises, involves graphs, not spatial variables. From a graph viewpoint, multiple dimensionality promotes an increased average connectivity for the graph because more neighbours can crowd into a three dimensional region and a two dimensional region or a one dimensional region.

Mathematically, a graph's spacetime dimensionality at a point is the number of possible degrees of freedom radiating from the node, i.e. the number of independent vectors emanating from it, or the node degree k[Bur14]. However, in a system realized in the context of the physical world, graphs must be embedded in a spatial volume of at least two dimensions, or more generally D dimensions. Physical agents have physical dimensions, i.e. they take up space, and links between them may not be allowed to cross because they occupy space (see figure 5.6). This places restrictions on behaviour that do not apply in the virtual world.

When we reach the topic of universal scaling relations, in section 5.15.4, we deliberately ignore the differences between labelled objects, including boundaries, i.e. by reclassifying them or counting them by lumping them into fewer categories, labelled by a scale parameter. The symmetry can be broken by the presence of a boundary.

5.7 SEPARATION OF SCALES AGAIN

When we try to scale systems, intentionally, to encourage greater output (such as building a larger factory, or a bigger datacentre, adding more servers, etc), we risk the fact that system mechanisms will not actually scale in accordance with our intentions. Designing systems for a large scale, or to span several scales, warrants different considerations compared to those for a small ad hoc system.

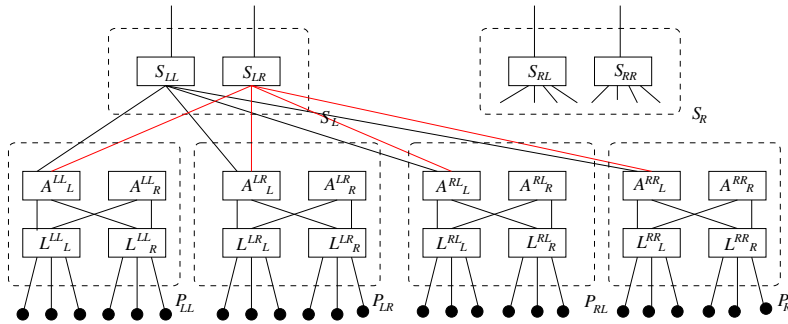


Figure 5.6: A redundant Clos fabric, takes up a three dimensional space. There is no approximation in which its behaviour can be understood in less than two dimensions.

Definition 77 (Coarse graining (informal))**.** *The elimination of detail from a system by aggregating low level details into fewer variables that represent the same situation over a new aggregate scale.*

Statistical averaging is one coarse graining procedure. After coarse graining over space, the *interior* details of a region are no longer observable, because we probe the system only as the average effects over longer distances. A consequence of this is that we are free to focus on the effective *exterior* interactions at the new scale, between the grains.

If we take an ensemble of systems, of different sizes, in similar circumstances, which keep a specific set of *exterior* promises, they might fail in different ways because of unobservable differences in the networks of interior promises. In other words, grains might appear to be superficially similar, whereas in fact they are different. The microscopic specifics of an system may play a role, if they remain significant as we scale.

Principle 8 (Separation of scales)**.** *In a weakly coupled (quasi linear) system, the details at one scale do not lead to strong effects at a different scale. This principle breaks down in non-linear systems with strong coupling.*

Non-linearity in systems is a source of great unpredictability because it prevents decoupling of scales. Small details can be amplified into large scale effects through interactions.

5.8 THE SCALING OF REGIONS OF AGENCY

To describe the spatial structure of agents, in which the composition of elements is consistent, we need to describe how agency scales collectively, through aggregation and reduction of elementary agents. The concepts of scaling are familiar to physicists for dynamics, but here we also want to extend them to incorporate semantics. This might be motivated by questions of the following kind:

- How does a team promise something as a unit?
- How does an organization appear as a coherent entity?
- How does a collection of components promise to be a car?

Since we can aggregate promises into a single promise, and aggregate agents into a single agent, the ability to detect or resolve parts within a whole depends on the observer's capabilities. Similarly, the ability for an agent to perceive a collection of individual agents with a collective identity (i.e. a superagent) depends on the capabilities of the observer agent. Elementarity and composability of agents thus go together with a hierarchy of observable agents, which needs to be elucidated.

5.8.1 SUBSPACES

A partial region of a semantic space, at any scale, may be called *subspace*. We may distinguish the boundaries of such a region in any way convenient. A subspace is assumed to be connected, but not necessarily homogeneous of isotopic.

Definition 78 (Subspace)**.** *A subspace is a collection of agents, in which every agent is adjacent to at least one other. The agents may be:*

1. *Identified and associated by an external observer by their (-) promises, or*
2. *Intentionally labelled and coordinated by its members with a (+) promise.*

Subspaces can be defined by partitioning a space, or by constructing a space agent by agent. There is good reason to consider both of these points of view, so let's describe them below.

5.8.2 INDEPENDENCE OF AGENTS UNDER AGGREGATION

We begin by considering how to identify discrete, elementary components within a system of autonomous agents, making promises. If we assume that sufficient knowledge

of agents in available, then an observer can assess the independence of agents by the absence of mutual information, i.e. zero overlap.

Definition 79 (Independent agent)**.** *Two agents A_1 and A_2 are independent iff the following overlap relation holds:*

$$A_1 \cap A_2 = \emptyset. \tag{5.53}$$

An agent that it independent of another agent may be said to be *outside* or *exterior* to the agent. An agent that overlaps with another agent may be said to be *inside* or *interior* to it.

5.8.3 COMPOSITION OF AGENTS

The treatment of a collection of agents as a single entity is a choice made by any observer. It can be made with or without promises from the composite agents themselves (see figure 7.3). Agency or 'agentness' can be defined recursively to build up hierarchies of component parts. In [Bur14], I showed how spatial boundaries can be defined by membership to a group or role. We still have to explicate the relationship between the internal members and the structure of the whole, as perceived by an observer.

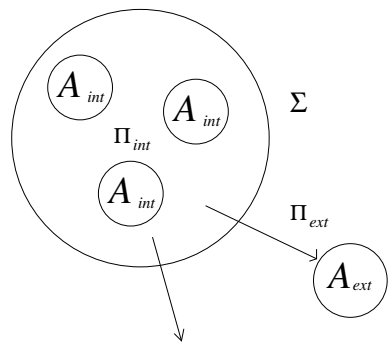


Figure 5.7: Agent structure consists of an element that makes a number of exterior promises, some of which are scalar, some vector, etc. Interior promises are invisible from the outside.

We define a collective superagent as a spacetime structure that has collective agency, i.e. its intended semantics relate to a collection of agents surrounded by a logical boundary, with collective semantics (see figure 5.7).

Definition 80 (Bare superagent)**.** *A superagent of size S is any bounded agent composed of individually separable agents, partially or completely linked by internal vector promises. The bare superagent is defined by the closed graph, without any external adjacencies. It is a doublet:*

$$A_{\rm super} = \langle \{A_i\}, \Pi_{ij}^{\rm int} \rangle, \quad i = 1, 2 \ldots S. \tag{5.54}$$

where A_i is an internal agent of $A_{\rm super}$, and Π_{ij} is the promise adjacency matrix between the S internal constituents.

Definition 81 (Dressed superagent)**.** *A dressed superagent is the bare superagent together with its set of exterior promises. It is a triplet:*

$$A_{\rm super} = \langle \{A_i\}, \Pi_{ij}^{\rm int}, \Pi_{s\epsilon_1}^{\rm ext} \rangle, \quad i = 1, 2 \ldots S. \tag{5.55}$$

Superagency allows promises to exist within and without a superagent boundary. We call these interior and exterior promises, respectively.

Definition 82 (Exterior promises)**.** *Exterior promises are made by agencies within the superagent boundary, to agents outside. They represent inputs and outputs of the superagent, i.e. how it interacts with an external space.*

Definition 83 (Interior promises)**.** *Internal promises are made by agents inside the superagent boundary to other agents inside the boundary. They represent the bindings that make the superagent behave as a single cohesive entity.*

In principle an observer could draw a line around any collection of agents and call it a cell or composite superagent. This is an assessment any agent can make, as part of its definition of an agency scale. However, it might still be of interest to distinguish special criteria by which such an arbitration might occur. In component design, for instance, the choice of boundary has often to do with the a choice interface an agent wants to interact with.

Superagent interior configurations are composed from three basic categories (see figure 5.8):

(a) A membership in a group or associative role, where the central membership authority may be either inside or outside the boundary. In this case, we are identifying a group of symmetrical agents.

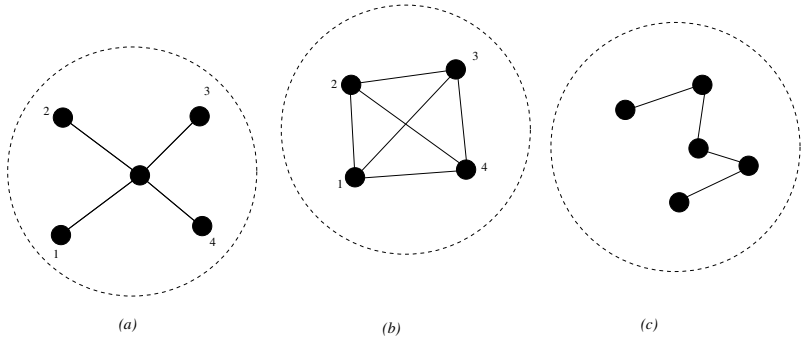


Figure 5.8: Three interior configuration patterns for binding agents into a collective agency. Another way is to simply make an arbitrary collection, unified by the promises they make (called a role).

$$A_{\rm host} \xrightarrow{+\text{membership}} \{A_{\rm tenant}\} \tag{5.56}$$

$$\{A_{\rm tenant}\} \xrightarrow{-\text{membership}} A_{\rm host} \tag{5.57}$$

(b) A total graph or collaborative role. In this case, we are identifying agents with coordinated behaviours.

$$A_i \xrightarrow{\pm\text{membership}} A_j \quad \forall A_i, A_j \in A_{\rm super}. \tag{5.58}$$

(c) A dependency graph, path or story. In this case we are identifying dependency bindings.

Let us now define the converse properties:

Definition 84 (Subagent)**.** *A sub-agent is an agent assessed to be a resident of the internal structure forming a composite (super) agent.*

Definition 85 (Residency)**.** *A sub-agent A is resident at a location L iff it is defined to be within the boundary of the agent:*

$$A \cap L \neq \emptyset. \tag{5.59}$$

Since an observer can form their own judgement about superagent boundaries, we cannot say that residence is the same as a promise of adjacency.

There are two types of adjacency, somewhat spacelike, which may or may not be interchangeable (see figure 5.9). Normal 'physical' adjacency promises, and resident adjacency, which might link agents virtually even though they are not physically adjacent. I'll return to this topic when discussing tenancy below.

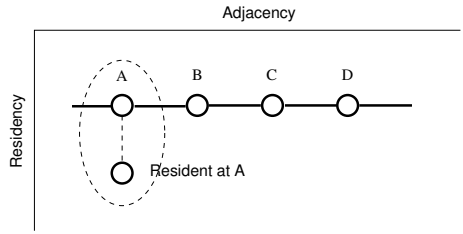


Figure 5.9: A resident adjacency forms a superagent by accreting to a seed agency that represents the location anchor.

Lesser agents can become satellites of other agents. This leads to a hierarchy or 'planetary' structure, by accretion into superagents.

5.8.4 SUPERAGENT SURFACE BOUNDARY

Superagent boundaries may be formed with different structural biases.

- Simple aggregations of agents, related through membership to a single leader (leader may be inside or outside the superagent), and the connections are made through the leader as a proxy-hub.

- A cluster of agents linked by cooperative vector promises.

- Strongly cooperative agents which are inseparable without breaking an external promise. E.g. an organism made of components that are all different and non-redundant to the functioning of the whole.

Interior promises are those entirely within the surface boundary of a superagent. We may define interior and exterior promise matrices for any agency, using a matrix

analogous to the adjacency matrix:

$$\Pi^{\rm int}_{ij} = \left.\begin{array}{ll} 1 & \text{iff } A_i \xrightarrow{*} A_j \\ 0 & \end{array}\right\} \quad A_i, A_j \in A_{\rm super} \tag{5.60}$$

$$\Pi^{\rm ext}_{i\epsilon} = \left.\begin{array}{ll} 1 & \text{iff } A_i \xrightarrow{*} A_\epsilon \\ 0 & \end{array}\right\} \quad A_i \in A_{\rm super}, A_\epsilon \notin A_{\rm super} \tag{5.61}$$

Definition 86 (Surface of a superagent)**.** *The exposed* surface Σ *of the agent is the subset of interior/internal agents that have adjacencies to agencies outside the superagent.*

$$\Sigma \equiv \{A_i\} \subset A_{\rm super} \;\Big|\; \Pi^{\rm ext}_{i\epsilon} \neq 0 \tag{5.62}$$

A superagent surface may also make new explicit promises that are not identifiable with a single component agency (see section 5.8.13).

Example 64 (Molecules)**.** *In the molecular example above, the superagent makes interior promises*[50]

$$A_1 \xrightarrow{+H} * \tag{5.63}$$

$$A_2 \xrightarrow{+H} * \tag{5.64}$$

$$A_3 \xrightarrow{+O} * \tag{5.65}$$

from internal agents, and collectively $M = \{A_1, A_2, A_3\}$

$$M \xrightarrow{+H_2O} * \tag{5.66}$$

This promise implies some interior structure, and it does not emanate from any smaller agent. Thus if an external observer is able to resolve the component agents within M*, the promise of* H_2O *is not longer a promise.*

As noted in section 6.3.1, an observing agent may or may not be able to discern an internal elementary agent within a superagent, i.e. whether the agent has internal structure, or whether it is atomic. This depends on whether the agent promises transparency across its surface.

Definition 87 (Superagent transparency)**.** *All promises whose scope extends beyond the boundary pass transparently through the surface of the superagent. Scope includes the list of promisees.*

5.8.5 SUBAGENTS AS THE SUBJECT OF A PROMISE: EMISSION AND ABSORPTION

An agent may itself be the subject of a promise body. Agents can conceivably promise to spawn new agencies: cells multiply, particles transmute into clusters, humans give birth, organizations spin off departments into new organizations, etc.

Assumption 3 (Emission and absorption, parent-child relationships)**.** *It is within the allowed behaviours of an agent to emit (send) or absorb and incorporate (receive) a child subagent, which has a formally distinguishable identity to the parent.*

The exchange of a subagent as an independent promise implies the exchange of the promises made by it too. However, there is no assumption about the promises having been kept before or after the keeping of the promise to emit or absorb the subagent. This might be reified later. The promising of an agent is thus somewhat like the passing of a point reference, with possibly late binding.

Lemma 17 (Emission and residency)**.** *In order to not violate the autonomy of agents, an agent $A_{\rm parent}$ could only make a promise to produce an agent $A_{\rm child}$ if, at the outset $A_{\rm child} \subset A_{\rm parent}$ i.e. resident.*

The understanding that agents embody strings of information (with an arbitrary physical realization) makes all of the arguments simple. An agent might spawn another, such as a device $A_{\rm device}$ emitting a network packet $A_{\rm packet}$:

$$A_{\rm device} \xrightarrow{+A_{\rm packet}} A_{\rm network}. \tag{5.67}$$

The agent referred to in the body can, in turn, promise another agent in its body, e.g. a verifier of the checksum authenticator:

$$A_{\rm packet} \xrightarrow{+A_{\rm checksum}} A_{\rm verifier}. \tag{5.68}$$

In order to be an intermediary, the exchanged package should promise to both sender and receiver:

$$A_{\rm checksum} \xrightarrow{+{\rm structure}} \{A_{\rm packet},\ A_{\rm verifier}\}. \tag{5.69}$$

This does not necessarily imply the existence of a promise about another promise. A promise of the existence of an agent has to result in an autonomous agent, by the rules of promise theory.

Definition 88 (Emission of a body part)**.**

$$A_{\rm super} \xrightarrow{+A_{\rm sub}} A_{\rm recipient} \tag{5.70}$$

$$A_{\rm super} \xrightarrow{A_{\rm super} \to \{A_{\rm super} - A_{\rm sub}\}} A_{\rm recipient} \tag{5.71}$$

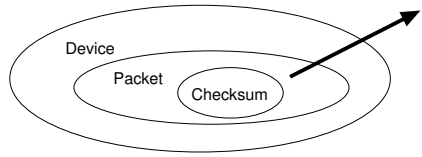


Figure 5.10: Agents may be emitted if they start as residents.

Definition 89 (Absorption of a body part)**.**

$$A_{\text{recipient}} \xrightarrow{-A_{\text{sub}}} A_{\text{sender}} \tag{5.72}$$

$$A_{\text{recipient}} \xrightarrow{A_{\text{super}} \to \{A_{\text{super}} + A_{\text{sub}}\}} A_{\text{sender}} \tag{5.73}$$

The signs inside the set braces imply union and complement removal, i.e. set difference.

5.8.6 THE EXISTENCE OF A GROUND STATE

Is there an empty space? Is there a discrete lower bound on agency? Can we infer the existence of a state of empty space in any system?

$$A \xrightarrow{\emptyset} A_? \tag{5.74}$$

Suppose there is a lowest level to the hierarchy of agency at which point no new intention can be inserted. The promises made by the agent are purely names, since a name is a promise that identifies the presence of the agent. A name or label cannot be subdivided without simply resulting in more than one name, i.e. without increasing the number of promises.

Hypothesis 1 (Lowest level of hierarchy)**.** *There is a level at which names cannot be subdivided without losing the ability to function and be understood (or connected to). The ground state is a gaseous state of maximal symmetry. At this level, the only promise that an object can make is its name (this is a tautology).*

Does an agent that makes no promises even exist? As mentioned in [Bur14], it is completely disconnected from the rest of a space, so we may consider it either to be non-existent relative to an existing spacetime, or be an entirely separate spacetime[51]. If we do not allow a complete absence of promises, then this suggests that empty spacetime must, at least, promise adjacency. That is, empty space requires at least a promise of adjacency. The promise might not be kept all the time, however, as is the case in a gaseous phase. This issue need further study.

5.8.7 AGENT SCALES

Superagency is a scaling transformation, in the sense of a dynamical system. It is therefore an important bridge between dynamics and semantics, with consequences for both. Because promises incorporate semantics, which may be arbitrarily applied to a collection of agents, boundaries for superagency may be defined around any collection of agents.

Definition 90 (Agency scale)**.** *A named collection of agencies considered to be the irreducible entities of space, i.e. it defines a set of possibly aggregate atomic agencies that are to be considered the set of addressable agents at the scale concerned.*

Unlike dynamical scales, there is no *a priori* identifiable measuring stick for semantic scales: they are non-ordered symbolic quantities, and must be promised independently by an observer, by promise types and body constraints. An agency scale is a thing in between a semantic scale and a dynamical scale. The identification of agency is a semantic issue, but the scale is a well-defined dynamical unit.

Example 65. *Suppose we have atomic agents A_1, A_2, A_3, which make exterior promises:*

$$A_1 \xrightarrow{+H} * \tag{5.75}$$

$$A_2 \xrightarrow{+H} * \tag{5.76}$$

$$A_3 \xrightarrow{+O} * \tag{5.77}$$

and interior promises

$$A_1 \xrightarrow{+e} * \tag{5.78}$$

$$A_2 \xrightarrow{+e} * \tag{5.79}$$

$$A_3 \xrightarrow{-e,-e} * \text{ (valency 2)} \tag{5.80}$$

We may combine these agents into a superagent by defining a scale:

$$\text{Molecular} \equiv \{M\}, \tag{5.81}$$

where $M \equiv \{A_1, A_2, A_3\}$. At the molecular scale, we now have a single superagent A, instead of three resolvable atomic agents.

Example 66. *Consider figure 5.11.*

At the level of atomic agents, we have exterior promises:

$$\pi_1 : A_1 \xrightarrow{+b_1} A_5 \tag{5.82}$$

$$\pi_2 : A_2 \xrightarrow{+b_1} A_5 \tag{5.83}$$

$$\pi_3 : A_4 \xrightarrow{+b_2} A_5 \tag{5.84}$$

$$\pi_4 : A_4 \xrightarrow{-b_3} A_6 \tag{5.85}$$

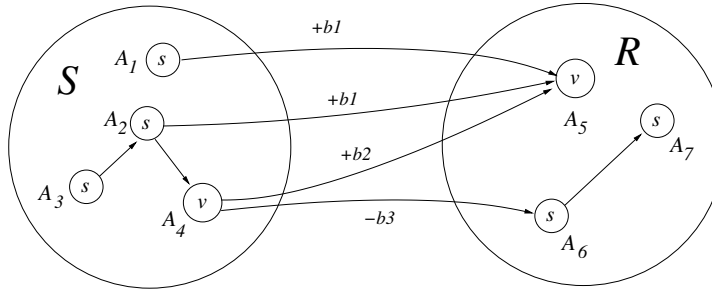


Figure 5.11: The transformation of promises under a coarse-graining transformation. How promises appear to emanate from the superagent surface.

and interior promises

$$\pi_5 : A_3 \xrightarrow{+b_4} A_2 \tag{5.86}$$

$$\pi_6 : A_2 \xrightarrow{+b_5} A_4 \tag{5.87}$$

$$\pi_7 : A_6 \xrightarrow{+b_6} A_7 \tag{5.88}$$

The superagents are defined by:

$$S = \langle \{A_1, A_2, A_3, A_4\}, \{\pi_5, \pi_6\} \rangle \tag{5.89}$$

$$R = \langle \{A_5, A_6, A_7\}, \{\pi_7\} \rangle \tag{5.90}$$

Scales have to be defined semantically in a promise model, since the interpretation of an aggregate boundary is arbitrary; thus, if we define the following scales:

$$\text{Atomic} = \{A_1, A_2, A_3, A_4, A_5, A_6, A_7\} \tag{5.91}$$

$$\text{Hybrid} = \{S, A_5, A_6, A_7\} \tag{5.92}$$

$$\text{Super} = \{S, R\} \tag{5.93}$$

Then, at hybrid scale, the promises (5.82-5.88) above collapse to:

$$S \xrightarrow{+(b_1 \cup b_2)} A_5 \tag{5.94}$$

$$S \xrightarrow{-b3} A_6 \tag{5.95}$$

and at super scale:

$$S \xrightarrow{+(b_1 \cup b_2), -b_3} R \tag{5.96}$$

Example 67. *The human eye is an example of a coarse grained receptor, i.e. a use-promise to accept radiation in certain frequency range. Light emitted at all frequencies in the range may be promised by emitters; however, only three coarse receptors (for red, green, and blue) cover this range. Thus colour vision in humans is limited to interpretation through these three channels.*

5.8.8 GAUSS' LAW FOR PROMISES

The divergence theorem (Gauss' theorem) is a simple universal identity that applies to vector fields enclosed by a boundary. We may apply it to vector promises too. In its well-known form, if may be written:

$$\int_\sigma \vec{V} \cdot d\sigma = \int_v (\vec{\nabla} \cdot \vec{V}) dv \tag{5.97}$$

i.e. the integral of vector flux emanating from a surface σ is the result of the divergence of the vector field generated from the volume enclosed by it.

Using definitions for unadorned graph theory, we can show that a coarse graining is a simple application of this result. In other words, the promises that come out of any volume or grain of space are only a result of what agencies are inside it. Consider a grain consisting of a number of agents with a surface boundary, and let

$$\pi_{ij} = \pi_{ij}^{\text{int}} + \pi_{ij}^{\text{ext}}. \tag{5.98}$$

We observe that π_{ij} is the j-th component of a local basis vector $\hat{e}_j$, surrounding any agent A_i. We can define a vector field in this basis by

$$\vec{\pi}(A_i) = \sum_{j\in} b_{ij}\hat{e}_j = \sum_{j\in grain} b_{ij}\pi_{ij}, \tag{5.99}$$

where b_{ij} is the body (semantics) of agent A_i's promise to agent A_j, where i, j run over all agents (or just the nearest neighbours of A_i) that are picked out by π_{ij}. The components of the $\vec{\pi}(A_i)$ as a row vector are thus easily constructed:

$$\vec{\pi}(A_i) = (b_{i1}, b_{i2}, \ldots). \tag{5.100}$$

The values are not really important, as long as they can be defined, since they will cancel out of the sums. The derivative of this vector is anti-symmetric in i, j:

$$\vec{d}_j \vec{\pi}(A_i) = (b_{(i+j)1} - b_{i1}, \ldots) \tag{5.101}$$

and thus the divergence is the sum of these where $i = j$. It is thus easy to see that the sum over the volume enclosed by any grain surface cancels everywhere except for those

vector components that protrude through the surface:

$$\int d(\mathrm{vol}) \sum_i (\vec{d}_i \cdot \vec{\pi}(A_i)) = 0 + \text{exterior contributions} \quad (5.102)$$

$$= \sum_{i \in \text{surface}} (b_{i\epsilon_i}, \ldots) \cdot \vec{\sigma} \quad (5.103)$$

$$= \sum_{i \in \text{surface}} \vec{\pi}^{\text{ext}}(A_i) \cdot \vec{\sigma} \quad (5.104)$$

where $\vec{\sigma}$ is the unit surface vector. This is exactly (5.97) for $\vec{V} = \vec{\pi}$, summed over the grain. i.e.

$$\sum_{i \in \text{surface}} \vec{\pi}(A_i) = \sum_{\epsilon \in \text{neighbours}} \vec{\pi}^{\text{ext}}(A_\epsilon) \quad (5.105)$$

5.8.9 COARSE-GRAINING AND AGGREGATION

Coarse-graining is what happens when one invokes a change of scale from a high level of detail to a low level of detail, i.e. from a large number of smaller agents to a smaller number of larger agents, by aggregation. This is usually done for systems assumed to have infinite detail, in the so-called continuum limit.

In a discrete system, it is straightforward to define a coarse graining by aggregation of autonomous agents into collections. In physics (dynamically), one does this by defining characteristic lengths (see the discussion for pseudo-continuous information in [Bur03]). In a semantically labelled theory, there are no such easily defined lengths, and we are forced to define granular scales explicitly as sets, see definition 90 (section 5.8.7).

Definition 91 (Coarse-graining)**.** *A transformation in which the collection of agencies representing spacetime are changed for a smaller set, with a corresponding reduction in detail. Coarse graining involves:*

- *The summation of bulk properties, and relabelling of composite superagents by a single collective label.*
- *The elimination of all interior promises, which are not visible between the agents at the coarser scale.*
- *Loss of information to exterior from the above.*

In a spacetime without boundaries, we identify self-similarity by the idea that a system is functionally and dynamically similar in its promises and behaviours, before and after coarse-graining by the coarse graining function:

Definition 92 (Coarse-graining function)**.** *A many-to-one function whose domain is the semantic spacetime* $\langle \{A_i\}, \pi_{ij}^M \rangle$*, composed of a collection of agents* $\{A_i\}$ *at scale* M*, along with its promise matrix* π_{ij}^M*, and whose co-domain is the image doublet* $\langle \{S_k\}, \pi_{kl}^{M'} \rangle$*, at scale* M'*:*

$$\langle \{A_i\}, \pi_{ij}^M \rangle \to \mathbf{G}(\langle \{A_i\}, \pi_{ij}^M \rangle) = \langle \{S_k\}, \pi^{M'} a_{kl} \rangle \tag{5.106}$$

where $i, j = 1 \ldots \dim M$*, and* $k, l \ldots \dim M'$*, where* $\dim M' \leq \dim M$

The function $\mathbf{G}()$ is a dimensionless renormalization transformation. Scale-free behaviour can not usually apply to a bounded space, as boundaries pin the system at a fixed scale. It might still be possible to compute effective equations for dynamical systems however[Bar96, Bar03]. Two spacetimes may be considered equivalent iff:

Definition 93 (Spacetime equivalence function $\simeq$)**.** *The equivalence of two semantic spacetimes designated by* $\langle \{A_i\}, \pi_{ij}^M \rangle$*, and* $\langle \{S_k\}, \pi_{kl}^{M'} \rangle$ *may be written*

$$\langle \{A_i\}, \pi_{ij}^M \rangle \simeq \langle \{S_k\}, \pi_{kl}^{M'} \rangle. \tag{5.107}$$

where there exists a permutation group relabelling of $i, j \to k, l$ $\pi_{ij}^M \equiv \pi_{kl}^{M'}$ *and* dim M = dim M'.

The transmission of influence by promise exchange is a dynamical property that must also be affected by coarse graining. This is the essence of the Renormalization Group in scaling systems[Bar96]. When a coarser grain size influences a smaller grain from the top down, it takes the form of effective boundary conditions on the behaviour of the smaller scale. This tends to introduce non-linearity, as it modifies the propagation law for behavioural trajectories. If a smaller scale interacts with a larger scale, from the bottom up, it takes the form of a perturbation, which probes the stability of the larger grains.

There is no *a priori* connection between scale and influence; this depends, as usual, on what exterior promises are made by the agencies regardless of grain-size. Multi-grain-size models are perfectly possible. Indeed, they are the natural state of materials like steel and plastics.

Let's examine the coarse graining of interior and exterior promises. Which promises are lost through coarse graining? Effective external promises, made by a superagent, have their origin in the internal constituent agents, but they do not preserve all of the internal details, and they might add new ones (see section 5.8.13). In order to scale agency, we therefore need to understand the scaling of both interior and exterior promises, and how they preserve the semantics of a system during a scaling transformation (see figure 5.12).

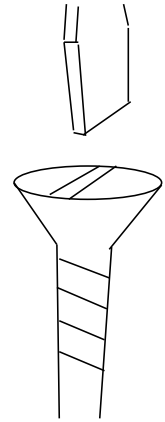


Figure 5.12: Agent coarse graining. If there is a mismatch in scale between promiser and receptor (if the screwdriver is too big to fit the screw), the promise might not make sense. In some cases, it might be possible to build a large scale solution from smaller grains, but not vice versa.

Example 68. *The chemistry promised by an atom does not depend strongly on the nature of the isotope of a chemical element, only on the electron structure. Thus electron shells made exterior promises from the surface of the agent.*

5.8.10 COARSE-GRAINING DIRECTORIES

How could we restore the transparency lost under coarse graining? Promise Theory predicts the obvious answer: a lookup table, or directory. Promises that remain exterior to a given grain, under a coarse graining process, may be combined under the idempotency rule for promises, and the combinatorics between common end-points. Hence promises of the same type will merge into a single maximal promise, defined below.

It remains possible for a single agent (coarse or fine grained) to make different promises to multiple recipients. These cannot be combined. However, promising from different subagents to the same recipient becomes an instance for the idempotence rule once the difference between the promisers is coarse-grained away. Promises with uniform promisees are idempotent, or cumulative by virtue of the set union. The general scaling rule for promisers is thus:

Definition 94 (Scaling rule for promiser)**.** *Let b_i be a set of promise bodies of the same type, i.e. $\tau(b_i) = \tau,\ \forall i$, originating from a set of distinct agents A_i. Then we may define the coarse graining of the set of agents, over constant τ, ε, by:*

$$\begin{aligned} \mathbf{G}_n \left(A_1 \xrightarrow{b_1} A_\varepsilon, \ldots, A_n \xrightarrow{b_n} A_\varepsilon \right) &= \mathbf{G}_n \left(A_1, \ldots, A_n \right) \xrightarrow{\max_i(\{b_i\})} A_\varepsilon \\ &= A_{\rm super} \xrightarrow{\max_i(\{b_i\})} A_\varepsilon \end{aligned} \tag{5.108}$$

where $\max(x_j)$ represents the union of the sets pertaining to set x_j intended for agent A_j:

$$\max(x_i) \equiv \bigcup_i x_i. \tag{5.109}$$

Not this rule does not depend on the sign of the promise body b_i.

In other words, the coarse-graining of a set of promises is equivalent to a possibly smaller set of aggregate agents, which promise the composite effect of the individual promises. The total number of promises is reduced from n to 1 in this coarse graining.

A coarse-graining procedure is not a bijection, i.e. it is not an invertible map or function. When a coarse graining of promisers is undertaken, information about the specific fine-grain promises is lost. In order to reverse a coarse-graining process, we would have to preserve the information lost.

Definition 95 (Coarse-graining directory)**.** *The information lost during coarse-graining in equation (5.108), takes the form*

$$\pi_{\rm directory}(\tau) = \left\{ A_1 \xrightarrow{b_1 \in \beta_1} A_\varepsilon, \ldots, A_n \xrightarrow{b_n \in \beta_n} A_\varepsilon \right\} \tag{5.110}$$

for promises of type τ, expressed in language β. Hence, whenever we group agents under a common umbrella, by coarse-graining, there will be associated map, of the form,

$$\langle \tau, \pi_{\rm directory}(\tau), \beta \rangle, \quad \forall \tau, \tag{5.111}$$

that preserves the lost information, and could restore the detailed view of which subagent is able to keep a promise perceived at the level of the whole. This is the information that becomes unavailable at the coarser scale[52]*. We may call such a map the superagent* directory *or* index *map, and write its promise:*

$$A \xrightarrow{+{\rm directory}(A)} A' \tag{5.112}$$

Coarse-graining replaces $\pi_{\rm directory}(\tau)$ with a single promise of type τ, and an effective promise body. So, for each external promise of type $\tau(b_i) = \tau,\ \forall i$, coarse-

graining leads to a surjective map, between the collection of promises made by subagents and the single effective promise made by the superagent. This map cannot be inverted, however it can be resolved to direct the external agent to an appropriate microscopic part of the superagent.

Lemma 18 (Resolvability of superagent detail)**.** *Let $A_{\rm super}$ be a superagent at scale M_1, formed by coarse graining promises at scale M_0. From in equation (5.108), we see that the information lost during coarse-graining $M_0 \to M_1$, takes the form of a collection of promises called a directory or index $\pi_{\rm directory}(\tau)$. By preserving, this directory we may expose and resolve the scale M_0 under M_1:*

$$\pi_{M_1} + \pi_{\rm directory}(\tau) \geq \pi_{M_0}. \tag{5.113}$$

How an external agent resolves a microscopic inspection of a superagent will be discussed in section 7.8.2.

5.8.11 PROMISEE COARSE GRAINING

For scaling of the promisees, the scaling rule is simpler than for the promiser. It basically amounts only to a redefinition of scope. Information about the scope within the promisee is lost to coarse-graining, but can be restored by using the coarse-graining directory (section 5.8.10), through scale transduction (section 7.8.5).

Definition 96 (Scaling rule for promisee)**.**

$$\begin{aligned}
\mathbf{G}_n\left(A_1 \xrightarrow[\sigma]{b_1} A_{\varepsilon_1}, \ldots, A_n \xrightarrow[\sigma]{b_n} A_{\varepsilon_n}\right) &= A_1 \xrightarrow[\mathbf{G}_n(\sigma)]{b_1} \mathbf{G}_n\left(A_{\varepsilon_1}, \ldots, A_{\varepsilon_n}\right), \ldots \\
&\quad \ldots, A_n \xrightarrow[\mathbf{G}_n(\sigma)]{b_1} \mathbf{G}_n\left(A_{\varepsilon_1}, \ldots, A_{\varepsilon_n}\right), \\
&= A_1 \xrightarrow[\mathbf{G}_n(\sigma)]{b_1} A_{\rm super}, \ldots, A_n \xrightarrow[\mathbf{G}_n\sigma)]{b_1} A_{\rm super},
\end{aligned} \tag{5.114}$$

and $\mathbf{G}_n(\sigma)$ is obtained by replacing any $A_{\varepsilon_1} \ldots A_{\varepsilon_n}$ with $A_{\rm super}$.

In other words, the coarse graining of a set of promises, in which only the promisees are aggregated, leads to the same set of promises (because promises originate from the promisers), made to the smaller set of aggregate agents, with only loss of detail and possibly redefined scope. This is now a question of definition, as the information about specific promisees and scope membership is lost as agents are aggregated.

5.8.12 INTRA-AGENT LANGUAGE UNIFORMITY

The assumption of the previous sections has been that every agent shares a single uniform language. As superagency forms by aggregation of parts, there is nothing to constrain the language inside the superagent boundary. Hence, we must expect linguistic diversity inside a superagent, limiting the interactions on the interior.

Definition 97 (Language of a superagent)**.** *For the purpose of all exterior promises, the effective language of a superagent S must be considered the union of all sublanguages:*

$$\beta_{\rm super} = \bigcup_i \beta_i, \tag{5.115}$$

where $i = 1, \ldots, \dim S$. The promised language will only be understood with a reduced probability however, since the coverage by subagents is non-uniform.

This information about linguistic diversity is also lost under coarse-graining, hence it must become part of the coarse-graining directory.

5.8.13 IRREDUCIBLE PROMISES AT SCALE M, AND COLLECTIVE BEHAVIOUR

In addition to the exterior promises, which emanate from composite superagencies, as the remnants of microscopic promises belonging to component subagencies, we may also observe that completely new promises are possible at each scale, which do not belong to any specific agent inside the surface.

Definition 98 (Irreducible superagent promises)**.** *Let M be an agency scale, and A_s be a superagent formed by an aggregation of agents. A promise with body b_s made by A_s*

$$\pi_M : A_s \xrightarrow{b_s} A_? \tag{5.116}$$

$$A_s = \{A_i, \ldots\} \tag{5.117}$$

may be called irreducible iff there is no set of subagents $A_i \in A_s$, for which

$$b_s \subset \bigcup_a b_a, \quad a = 1, \ldots \text{all promises to } A_? \tag{5.118}$$

where b_a are existing bodies promised to $A_?$ by A_i, and $A_i \neq A_s$.

In other words, if there exists a combination of promises made by one or more subagents (and we assume that the subagent is not the same as the superagent), then that is semantically equivalent to the full promise made by the composite agent, when one could say

that the superagent promise could be reduced to the promise of one of its components. As long as no single agent, working alone, can make such a promise, it makes sense to talk about the collective superagency making a new promise that is not explicit in the capabilities of its subagencies. Thus, irreducible promises at scale M take into account emergent effects, and collective effects of agent interactions.

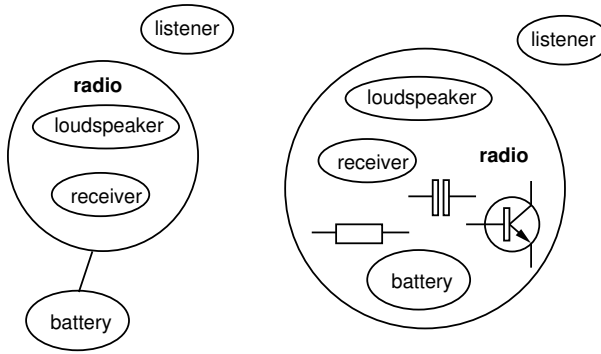


Figure 5.13: A radio is an example of a composite 'super'-agent, with interior agencies and promises of circuit connectivity, and exterior promises to play music. The radio and listener define one agency scale, but the components and a repair engineer could form another. Scales of interaction are not necessarily subject to uniform coarse-graining in a world of semantics.

Example 69. *A radio is a composite agent that makes the exterior promise of playing radio signals on a loudspeaker (see fig 5.13). Interior promises are made by many component agents like resistors, capacitors, transistors, etc. The radio has a semantic surface which interacts with people as promisees*[53]*. Does this mean that the components inside the radio have to interact with the components inside a person (cells, organs etc)? Not really. Implicitly, this might be partially true by the distributive rule, but clearly that is not a requirement. Any agency can interact, semantically, with any other agency at any agency scale, provided there is a physical channel for the communication to work.*

Irreducibility is thus a result of collective phenomena in the underlying semantics and dynamics. The latter answers classic objections against the naive reductionism of systems. Yes, we can decompose systems into a sum of parts, but one must not throw away the promises when doing do, else it will not be possible to reconstruct both semantics and dynamics. This is a consequence of boundary information or systemic topology.

The form or collective identity of a superagent can be enough to signal a promise, by association. This is an emergent promise in the sense of [BB14a], i.e. one inferred by a

potential promisee rather than given explicitly by an agency. Nevertheless, some scaled objects do make promises: tables, chairs

Law 1 (Reduction law)**.** *When reconstructing a system from components from finer-grained scale M' to a scale M, all superagencies at scales below M, and their component promises must be retained in order to reconstruct the system as the sum of its parts. This is achieved if each superagent promises its coarse-graining directory, providing dynamic transparency.*

Thus increase of detail downwards may not be at the expense of loss of upward irreducible information. If we try to view a compound agent as a collection of parts, with component promises (on the interior of the superagent), the component promises revealed do not replace the high level exterior promise: they are simply prerequisites for it. The only reason to disregard irreducible promises belonging to a coarse grain is because one is focusing on the internals of an agent in isolation (what one calls a closed system in physics).

If a superagent has no agency of its own, how can it make a promise? Clearly, a collection of agents has agency through its members, and these may or may not have the ability to keep promises related to scale M. So where do promises come from in a superagent?

- $+$ promises: the collective appearance of a superagent, at a certain scale, must provide the information to signal a promise. The declaration of the promise may or may not come from a single agent. The keeping of an irreducible promise does not come from a single agent, by definition. New irreducible promises are dependent on the individual agents, only through their cooperative behaviours.

  Example 70. *A troop unit promises to surround a house (+ promise). The troop leader can make the promise on behalf of the group, but no single agent can keep this promise, but collectively they can. In this case, the promise would often be given by a team commander, with a centralized source of intent, and subordinate agents. However, a team can also arrive at this promise by cooperative consensus.*

- $-$ promises: a promise to accept another promise, made by an external agent, might only apply to certain subagents with the appropriate capabilities. It must be provided as an exterior promise based on the by cooperative agreement amongst the agents.

  Example 71. *The Very Large Array of radio telescopes in Mexico has 27 receivers. Each receiver (- promise) is coordinated with the others so that they act as a single superagent. No single agent can see what the full array can see working*

together, up to diffraction. Hence the combined array can make promises that individual agents can't.

A superagent's *agency* is thus conditional on the existence of its subagencies, even though its promises are not locatable in any single agent. In both cases above, the promise made by a superagent could not be made by a single agency (this is what we mean by a 'host'), or by distributed consensus. Irreducible promises are thus conditional, not only on the uniform cooperation of the subagents about a single promise, but on their making all the promises that indirectly lead to the irreducible property.

Lemma 19 (Irreducible promises are conditional promises, for all scales greater than M_0)**.** *Irreducibility is not expressible directly as a sum of component promises of the same type as the irreducible promise, but it is second-order expressible in terms of subagent promises, by building on the existence of these contributing promises.*

With no promises to build on, a collective agent cannot make any kind of promise, since nothing can be communicated, and it has no independent agency. Crudely, a superagent is indeed the sum of its parts, as far as agency and promise-keeping are concerned; but, it is not merely a direct sum as new promises are possible through cooperation.

If the scope of exterior promises extends to the interior of a superagent boundary, that scope becomes ambiguous under coarse-graining. External observers can no longer see which agents are make or receive the interior promises, nor is there any way to refer to them independently after coarse-graining. Observers can only assume that the scope of a promise includes all subagents, but this might not be the case, and might result in erroneous expectations. Indeed, there is no reason why the subagents would even all use the same body language. We explore this more in section 7.8.1.

5.8.14 SCALING OF PROMISE IMPACT, AND GENERALIZED FORCE

As we scale agency to deal with larger entities, it seems unreasonable to expect the impact of a single promise from a subagent to have the same impact on the superagent as on the microscopic parts before coarse-graining. When might we be able to disregard a promise? If system is stable, i.e. small influence leads to small effect. Without going into too many details, I'll sketch how this can be dealt with.

Example 72 (Density scaling effect)**.** *In dynamics, one has the notion of a change of momentum (a force), as well as pressure (force per unit area). The way force is distributed over a region is important to how agents share the impact. It's important to construct such a notion of force density or pressure for promises too. However, we would note that even though momentum transfer alters in magnitude, the semantics of*

momentum transfer do not change, i.e. Newton's law of momentum conservation does not change.

Example 73 (Scaling of rights)**.** *Some rights and freedoms, in a society, that seem innocent enough can become a public nuisance if the population density grows too large, e.g. urinating in public, allowing stray dogs, crime, even certain forms of speech such as hate-speech. All these are unproblematic as long as they are sparse events, but like all queueing processes their significance grows non-linearly with frequency. Extremely populous regions are thus likely to regulate behaviour more than sparse regions.*

An effective force (see section 6.3.1) is one possible measure of impact. It's attraction is that it works for semantics and dynamics, though it is clearly modelled after physical dynamics. If we tried to scale this relation, we would scale the promises first, but then we also need to scale the assessment function. I'll leave that exercise for another time, or as an exercise to the reader.

The coupling of one scale to another has some coupling strength which depends on or describes the transmission effect of information between agencies. In physics we have the law of conservation of momentum as the currency of influence, and energy as a 'stored wealth'. For semantics, we need an impact if intent where intent is preserved, but not necessarily outcome. This is more analogous to the conversation of charge in physics.

There is thus a reasonable algorithmic procedure for progressively disregarding the impacts of certain promises relative to the scale of certain agencies, as we coarse-grain a system.

The characterization of weaknesses, cracks, and defects, in spatial promise structures, is an obvious follow-up question to this notion of semantic impact. If two superagents or subspaces come into contact, could there be catastrophic outcomes by which one region might not be able to withstand the influence of the other? This introduces concepts like *dynamical stability* and *material failure*. Here, instability transduces the smallest dynamical effect into the largest semantic importance.

5.9 PROPAGATION OF INFLUENCE BY STATE MESSAGES

We see that state does not lead to influence unless it is observed, and there are conditional promises that use it to promise conditional behaviour. This is the behaviour of a 'switch'.

We can now think about this in terms of dependencies. In order for remote state to affect a local process, a source agent has to share it, then the receiver has to observe, accept it, and subsequently alter its behaviour according to it (see section 5.10.2).

From section 7.7, what we call the beginning and end of a process is a scale-dependent characterization. We make a choice about which agents we want to include at a given scale. The common understanding of processes has some basic elements however. Each process has a lifecycle of major states that characterize it, which we can call epochs in the lifecycle of the process.

- Definition (of promises).
- Initialization of resources (Initial state).
- Execution (keeping promises).
- Termination (Final State).

This is basically the same model one has of any dynamical system in mathematics or physics. It maps on to the equivalent, e.g. think of solving differential equations:

- Definition the equation.
- Initial boundary condition.
- Find the propagator that computes the derivative states.
- Final boundary condition.

These are the major elements we use to describe the causality of a system, and fix its trajectory.

In a cloud setting, these correspond to

- Building software
- Configuring software settings
- Executing the software (runtime)
- The desired end state (outcome)

There is state in each of these epoch timescales. We are free to redefine the placement of changes. e.g. keeping code or configuration invariant during execution, or to write code or configuration that rewrites itself (as in learning systems).

5.9.1 SCALING OF LOCAL STATE

A proper discussion about the localization of state can only made with reference to a theory of scaling. What is local at one scale is composed of many locations on a smaller scale[54]? We therefore need to decide on the agents, or units of localization: what do we mean by entity, agent, location in a given context? As mentioned above, a certain locale could refer to anything from single chip register or a distributed database, depending on the author's state of mind!

Computer processes are made up agents, which are discrete processing units. They sometimes work together in clusters—represented here as superagents. By making promises, they form many patterns such as client-server interactions, data pipelines, object models, microservices, container pods, backup servers, redundant failover, etc. Promise Theory provides a simple view of scaling, based on boundary semantics, that easily accounts for the cases found in IT[Bur15a]. We can thus ask, to what extent are promises (e.g. about state) within or without of a boundary? Is state implicated in decision-making at the level of a conditional promise on the interior or exterior of an agent boundary? How is state implicated in propagation of assisted promise-keeping?

Locality refers then to the ability to draw a semantically defined boundary around an agent (i.e. one based on what it promises rather than based on where it happens to reside) and decide what is on its interior (local) and what is exterior to it (non-local). Every system of agents that interacts with other agents breaches its boundary or grows it to accommodate new members, so the definition of a system 'module' is always an ad hoc matter. Modules are often chosen based on functional separation in IT[55].

Part of the confusion in the colloquial use of 'stateless' is that 'state' itself refers an implicit and specific scale for many authors, namely whatever favoured object they happen to be working on (i.e. a location like a programming class, a process container, a cluster, or a host computer). Software engineering does not teach practitioners to think across multiple scales. State may therefore refer to all scales, from interior microstates to aggregate macrostates, and refer to real or virtual space. In order to observe and measure state, it needs to persist relative to the process that samples it (i.e. for some finite number of samples or duration of proper process time). Different processes tick at different rates, and interactions often lead to waiting. The issues of observation were discussed in [Bur19a].

A notion of 'total state' may be accumulated over many interactions, either laterally across many redundant concurrent processes, or longitudinally over multiple similar interactions, such as in data collection and machine learning applications. Already it seems clear that we need to distinguish different kinds of state and that the intended use of the data play a part in what is objectionable about statefulness to some authors. If we think of sampling as an information channel, in the Shannon sense, then the separation

of timescales amounts to partitioning process samples into different channels according to the timescales over which we *assume* that certain state we rely on will be invariant, i.e. constant with respect to multiple samples.

5.9.2 STATE LOCALIZATION AT DIFFERENT SCALES

If we redefine the agent boundaries or partitions of a system, we can shift state formally from one location to another, but we can't do so without altering the promises kept by the outer boundary. This assumes, naturally, that state is depended on for a purpose. Free state is irrelevant baggage.

We can try to summarize statelessness without referring to a particular case, like client-server or data pipeline, or even to a particular scale, while—at the same time—unifying semantics and dynamics for the process:

Definition 99 (Locally stateful)**.** *A locally stateful process is one in which memory is kept on the interior of a process agent or superagent cluster. This memory is promised for as long as the process agent's dependent exterior promises persist, and access to process memory occurs over interior channels.*

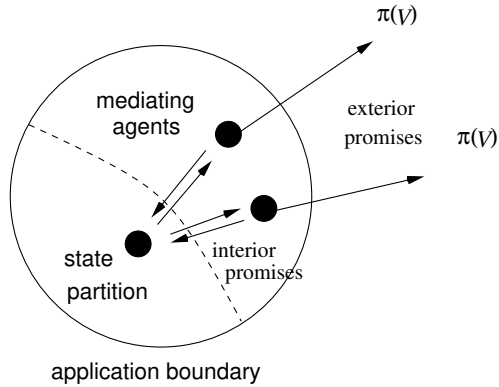


Figure 5.14: A partitioning of a process 'entity' or superagent. Within its semantic boundary there are stateful parts and stateless parts. Should we argue these as separate or integrated? The exterior promises are conditional on the interior state dependency, but the mediating agents are stateless (memoryless). This approach was used in Kubernetes, for example, where container services were initially assumed weakly stateless, with possible database services partitioned into separate containers and storage services. Later, this was rationalized to weaken the claim of statelessness.

Definition 100 (Non-locally stateful)**.** *A non-locally stateful agent is a composite agent, in which any persistent process memory accumulated over the history of interactions is partitioned and kept independently of the agent mediating an exterior conditional promise (see figure 5.14). The mediating agent is then merely a conduit for state that persists in an agent belonging to a 'backing' process partition. The loss of the mediating agent does not incur a loss of partitioned process memory for the collaboration.*

This deliberate indirection—pushing state out of one agent and into a dependency—seems to implicitly reference shared resources and risk mitigation, not whether state is promised or used[56]. So reference to state is a red herring for intended purpose, and it conceals assumptions about the timescales and number of times over which the state will be used before it changes. Such matters are critical and therefore the assumptions are unacceptable.

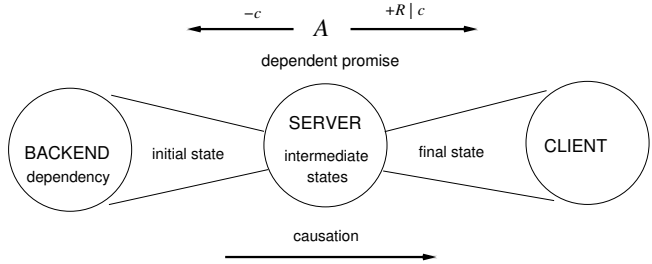


Figure 5.15: Conditionality is causality. The states that are causally implicated in a conditional promise include initial state, including code and prior configuration; then there is runtime state and shortlived intermediate computational state, which may be reconstructible from code and inputs. All these combine to an output that may be strongly or weakly dependent on the full set: how many intermediate states are involved in computing a virtual transaction? Is the scope of that state private or shared between distinct exchanges? These are all questions of process scale.

The key question about state, then, is not whether it is retained, but rather whether or not it is used as a dependency in the keeping of a larger promise. If the loss or latency of such state gets in the way of a larger dependent promise being kept, then one would be better served by a collaborative architecture in which that risk may be mitigated. The term 'shared nothing architecture'[Sto86] is more accurate than 'stateless' to address this. It implies a form of sharding or partitioning of agency in a system: possibly at either the client side or the server side. Both ends can end up having to deal with inconsistent promises[57].

Whether we keep state in primary RAM memory rather than in secondary disk

storage or even tertiary services like databases is not the issue. The issue is how do we depend on it, i.e. what happens if it's lost. What sources do we *trust* to keep stable promises?

Unfortunately, a 'shared nothing' partitioning of dependencies to localize causal interference has its own problems. A set of services (e.g. for web, database, and storage) is already made up of separate processes, even if they run on the same host, or with the same common storage. Just how much separation is 'shared nothing'? If they all share a common purpose, then they must be connected by something. Should we wrap them in layers of virtualization (containers, virtual machines, etc), or run them on different hosts, in different racks, in different datacentres? By handing off state to another agent that serves it up as a backing service, we only introduce a new shared dependency.

Example 74 (Containers and firewalls to prevent harm)**.** *We often try to protect ourselves from the ravages of 'complexity', i.e. the unwanted propagation of causal influence, by trying to build boundaries to prevent propagation. This may or may not succeed. When systems have complex networks of causation, predicting a final stable state may be practically impossible. Boundaries may help to assure local predictability, but they may also cripple the intended function of a system as consequence (this was a common complaint of security standards known as STIGs for US government).*

5.10 STATEFUL (MEMORY) PROCESSES

We can now put these key elements together to understand causal dependence, or chains and transition matrices across networks of agents. What's key about a process is what invariant information we have to constrain its trajectory. Initial and final conditions are the available external fixed points. The process rules (promises) may also implicitly contain fixed point behaviour such that the process converges to a 'desired state'.

When processes depend on one another, they observe one another's states. The amount of memory they have, internally, defines the extent of their dependence on their own causal past or future, both as memory for storing their program and for representing decisions as a 'log' or 'journal' of prior states.

5.10.1 TIME DEPENDENCE AND MEMORY PROCESSES

Processes that do not depend on any recent history do not need to carry their own clocks. In other words, they don't need to carry the memory of previous states with them as part of their processing. Such processes can be called memoryless. We can supply an invariant definition of memorylessness using stochastic processes. The term 'statelessness' is sometimes used to refer to memorylessness in the sense of a Markov

process: the dependence on current state is inevitable as long as there is input to a process, but dependence of behaviour on an accumulation of state over many iterations is what people usually mean by stateful behaviour.

Variables are embedded agents that keep simple promises to remember a value. A stateful process accumulates parametric data over a number of interior times $t_0, t_1, t_2, \ldots$, so that we could write each promise made by the process as a function of all those times:

$$\begin{aligned} \pi \quad &: \quad A \xrightarrow{+V[d(t_0),d(t_1),\ldots] \mid d(t_0),d(t_1),\ldots} A' \\ &\simeq \quad A \xrightarrow{+V[d(\bar{t})] \mid d(\bar{t})} A' \end{aligned} \tag{5.119}$$

which schematically means that

$$\frac{\partial \pi}{\partial d} \neq 0, \tag{5.120}$$

$$\frac{\partial \pi}{\partial \bar{t}} \neq 0. \tag{5.121}$$

i.e. the promise made by the agent is not constant over the interactions it promises. The memory $d(\bar{t})$ is accumulated from some initial time, making the promise evolve. This is called a *memory process*. The converse of a memory process is a memoryless process. A memoryless process may be constant in time, or it can depend on only that last known state, like a ballistic trajectory (imagine billiard balls whose change in behaviour is entirely determined by the last ball to strike them)[58].

Memoryless processes are also called Markov processes (see figure 5.15). Their behaviour is 'ballistic' in the sense that the arrival of a prerequisite state effectively triggers the release of what is promised by each agent. A Markov process is usually described as a chain of agents that satisfy a condition about random processes, and based in probabilities[GS01].

Definition 101 (Markov Chain)**.** *Let X_n be a discrete random variable, for non-negative integer $n = 0, 1, 2, \ldots$, taking values in $\{x\}$.*

$$\begin{aligned} P(X_n = s \mid X_0 = x_0, \ldots, X_{n-1} = x_{n-1}) = \\ P(X_n = s \mid X_{n-1} = x_{n-1}) \end{aligned} \tag{5.122}$$

for all $n \geq 1$ and $s \in \{x\}$. A Markov process has a transition matrix *or* scattering matrix *for the discrete set of states X_n:*

$$T_{ij}^{(n)} = P(X_{n+1} = j | X_n = i) \tag{5.123}$$

If this transition matrix is independent of n, i.e. $T_{ij}^{(n)} = p_{ij}$, for all n, then the chain is said to be homogeneous or translationally invariant.

Probabilities, in the usual sense are globally defined, but we can replace them with *assessments* in Promise Theory, which are the local equivalent. Each observer agent O in a system may form its own assessment $\alpha_O(\pi)$ of the probability that a promise π will be kept, for any definition of probability. Then the above definitions apply for any local observer by the association:

$$O_{ij}^{(n)} = \alpha_O(X_{n+1} = j | X_n = i). \tag{5.124}$$

In a Markov scattering process, each input leads to a unique output by a fixed rule. The scattering doesn't depend on the order of the inputs nor their relative frequencies. The scattering matrix does not remember past inputs; everything depends on the last one. This makes the scattering agent autonomous or causally independent[59].

Definition 102 (Causally independent)**.** *An agent is causally invariant under a promised influence x if it does not depend on a parameter x, so that.*

$$x \to x' \stackrel{\text{implies}}{\longrightarrow} V(x) = V(x'), \quad \forall x, x'. \tag{5.125}$$

In a quasi-differential shorthand, we might also be tempted to write:

$$\frac{\partial V}{\partial x} = 0. \tag{5.126}$$

Finally, we should note that this should not be taken to mean that V is differentiable, as no such mathematical property exists in the real world, but we can construct state space extended generalizations that include averaging, and so on, so I'll ask the forbearance of readers and follow common practice and use this as a shorthand for the expression of independence of V on some parameter x. For a popular discussion of the meaning of this, see [Bur13a].

5.10.2 SHORT MEMORY PROCESSES: LINEARITY

A perspective, which addresses increasingly popular ideas about complexity and chaotic behaviours, is process *linearity*. Linearity is related to weak coupling, and addresses the relative scales of process interactions (see [Bur19a]). Non-linearity is associated with memory behaviour—behaviours in which past interactions change the system so that new interactions experience a modified system. This is *learning* behaviour. A non-linear process cannot simply be replaced or restarted without access to a complete history of interactions, synchronized for all times, because it's outcomes depend on that unique memory of interactions. Non-linear agents cannot be redundant, as their unique histories distinguish them.

A system may be called linear if it comprises conditional promises that are Markov processes of order no greater than 1 (Markov processes are described in the appendix). In

other words, if the process is independent of past inputs to a scale that goes back $n > 1$ samples into the past (where the 'past' is defined to mean a chain of prior samples). This matters when the delivery of data could be carried out in a transactional way, but the promised methods that receive and process the data are changing concurrently as part of an independent process. Dependency graphs may span multiple processes implicitly. They might be quite invisible in program code.

Consider a simple interaction, of the 'client-server' variety, in which an agent C (in the role of client) promises or imposes a request c onto an agent S (in the role of server), which is accepted by S i.e.

$$C \xrightarrow{+c} \blacksquare \quad S \tag{5.127}$$

$$S \xrightarrow{-c} C. \tag{5.128}$$

The absorption of c by S implies that a state has changed in S, for some timescale that persists for a sufficiently long time to enable a response r to be returned. Let's say that the response is a simple storage lookup, like a database record or a web page. This acts as a key-value pair, where the key is c and the value is $r(c)$, which depends on c

$$S \xrightarrow{+r(c) \mid c} C. \tag{5.129}$$

In order for S to make this conditional promise, it has to contain the state variable $V = r(c)$ on its interior. The state variable is persistent, so S is clearly part of a system that promises state. Now, it might 'outsource' this capability to another agent (a backing service, in the vocabulary of [Wig17]). Then, we have an assisted promise[BB14a]. Suppose the assisting or backing agent is D, then S hands off responsibility for state to a subordinate agent, and must therefore make an assisted promise that depends both on the client request c and the promise of state storage d:

$$S \xrightarrow{+r(c) \mid c,d} C, \tag{5.130}$$

$$S \xrightarrow{-d} C, \tag{5.131}$$

$$S \xrightarrow{-c} C, \tag{5.132}$$

$$\tag{5.133}$$

where S hands off the request to its subordinate:

$$S \xrightarrow{c'(c) \mid c} D \tag{5.134}$$

$$D \xrightarrow{-c'} S \tag{5.135}$$

$$D \xrightarrow{d(c') \mid c'} S \tag{5.136}$$

$$S \xrightarrow{-d} D \tag{5.137}$$

As long as each dependence is a Markov process, forming a Markov chain, the dependency on c is linear.

Definition 103 (Linear conditional promise)**.** *A conditional promise π is linear with respect to a dependency d iff,*

$$\pi : \; S \xrightarrow{+V(d) \mid d} R \tag{5.138}$$

implies that $\partial V/\partial d = const$ over the life of π (see appendix).

Linearity literally implies that a functional dependence on d is linear (of polynomial order 1), and does not alter the functional form of the promise $V(d)$. The dependency d does not alter the promise, except to act as a lookup key. If we were to repeat the keeping of the promise over some timescale, i.e. over some chain of promise keeping assessments, an observer would not assess there to be any difference in the result of $V(d)$, over a number of samples T. The promise is therefore invariant over a timescale T.

The qualification of a bounded interval T is important, because no system is truly invariant for all future history (see figure 7.9). Changes do occur to systems and their promises: new versions of software promises are made, for example. The real issue is whether one can redefine a process to ensure that invariants are fixed somehow before runtime execution starts and all the way up to when it ends[60].

Lemma 20 (Linear promises and weak coupling)**.** *The need to wait for state history increases the service time for a queue, increasing the ratio of λ_R/λ_S.*

We need to define clear *timescales* for the assertions (promises) we make. Slowly varying changes decouple from changes that occur on the timescale of the promise because each sampling of a linear system is an independent variable, and a sequence that depends on multiple samples is independent of the sample if the sample has already been integrated (e.g. refactored) into the definition of π[61].

5.10.3 LONG MEMORY PROCESSES

Long memory processes depend on the sequences of states that led to their current state: e.g. does it matter which route you used to enter the city? This is the typical domain of machine learning.

The memory required to keep this promise determines a minimum scale for the process. Long memory processes cannot be stateless, in any definition, but it may be possible to separate part of a long memory process and isolate certain subagents whose behaviour is memoryless.

5.10.4 INVARIANT DEFINITIONS OF STATELESSNESS

Given the popular usage of the term 'stateless', it seems appropriate to accommodate the commonplace ideas with a clearer definition, so that we do least violence to present day intuitions. This leads to what I'll call weak statelessness:

Definition 104 (Weakly stateless process)**.** *A memoryless process (Markov process of order 1) promises that its interior memory of past interactions is the empty set:*

$$A \xrightarrow{+V(t)=\emptyset} *. \tag{5.139}$$

The definition is only weak, because it doesn't say much about what other behaviours the process may have. Implicitly, it suggests that that the next outcome of the process can only depend on the inputs at each step. Inputs could easily include data from long term exterior memory. The key point is that the promises that are purely local to the weakly stateless process are decoupled from, i.e. invariant, for all possible input-output transitions, as in (5.124).

Memory processes, or stateful processes, are those that are not weakly stateless.

Definition 105 (Stateful (memory) process)**.** *A process that promises:*

$$A \xrightarrow{+V(t)\neq\emptyset} *. \tag{5.140}$$

When these two kinds of process are composed, to form a superagent on a larger scale, the result is naturally stateful.

Lemma 21 (Stateful + stateless = stateful)**.** *An agent that promises to be both stateful and weakly stateless is stateful by composition.*

The proof is trivial:

$$\left.\begin{array}{l} A \xrightarrow{+\emptyset} * \\ A \xrightarrow{+V(t)} * \end{array}\right\} \equiv A \xrightarrow{+V(t)} * \tag{5.141}$$

If we want to be strict in the definition of statelessness (what we might call a purely ballistic process) then the agent responsible has to refuse all input.

Definition 106 (Strongly stateless process)**.** *A process that has no exterior (-) promises to accept input from any source during its lifetime. The agent's promise is thus completely constant: it does not rely on the order or substance of any other information.*

What we surmise is that basically all non-trivial processes must be stateful on some scale, because a promise of stateful behaviour overrides a promise of stateless behaviour on any scale.

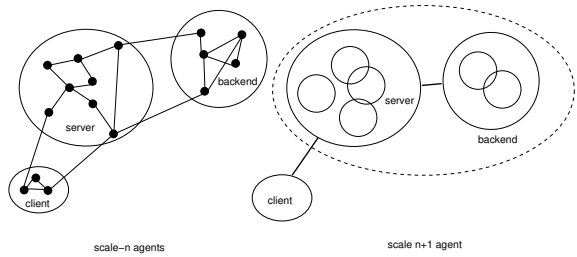


Figure 5.16: A client-server system with a backend can treat the backend as part of a service, or as a separate service. If the exterior promises remain the same, then these configurations are indistinguishable. We are always free to compose or decompose agents at scale n into agents at scale $n-1$ or $n+1$ by redrawing the boundaries around modules. This shifts a discussion about interior to exterior or vice versa, but cannot affect the outcome observed by an agent on a scale greater than the total system.

5.10.5 TRANSACTIONS ON SCALE T

It's usual to define transactions in terms of atomicity and consistency. Here we can define the concepts more simply using invariance of promises:

Definition 107 (Transaction at scale T)**.** *A transaction is the promise of an invariant sequence of messages $M_1, M_2, \ldots, M_T$, of length/number T, accepted by a process agent A, whose memory of the messages is also invariant over the sequence, and contains all the data needed to keep the conditional promise*

$$A \xrightarrow{+X|M_1,M_2,\ldots,M_T} \tag{5.142}$$

In other words, the agent A doesn't let go of the information from its cache until it is acknowledged by the receiver. Failures on a large enough scale can still wipe out all the information of the transaction, but this adds some assurance of invariance if the data survive the transaction.

With this definition, we do not presuppose any model or scale for the meaning of a transaction. As long as the transacting agent is invariant over the completion of its promised task, and the data require no dependencies. The virtue of this definition is to make such transactions repeatable, as all the conditions of the transaction are self-contained, and thus invariant. Put another way, transactions turn messages into scalable autonomous (super)agents, without exterior dependencies beyond their promised scale T.

Lemma 22 (Transactions are repeatable)**.** *Any valid transaction at scale T leads to a repeatable process, given the same message and conditional promise.*

Notice that the process is only memoryless if $T = 1$, i.e. we choose a particular scale, but the all important invariance is scale independent.

5.10.6 SCALE DEPENDENCE OF STATE AND CAUSALITY

Under scaling transformations that aggregate processes by causal dependence, the foregoing discussion should make it clear that we can state quite strongly:

Theorem 3 (Statelessness is scale dependent)**.** *A process that is weakly stateless at scale n may be stateful when causal promises are composed or decomposed at scales $n-1$ and $n+1$.*

The proof of this is elementary. Consider agents A_1 and A_2 that make promises that are stateless and stateful respectively, such that A_1 depends on the promise of A_2

$$A_1 \xrightarrow{\text{stateless}} * \tag{5.143}$$

$$A_2 \xrightarrow{\text{stateful}} * \tag{5.144}$$

$$\{A_1, A_2\} \xrightarrow{\text{stateful}} * \tag{5.145}$$

This theorem renders statements like 'transactions cannot span entities'[Hel07] meaningless, as there is no plausible definition of an entity without a clear specification of scale.

Causality itself is about the transmission of prior state, along the trajectory of each autonomous process, causality must itself be scale dependent. Indeed, as we'll see, influences may appear to be determined by states that are only reached in a process's future. Time does not follow a simple imperative ballistic view of prior state. In the frame of the process itself (the proper time) acausal changes frequently take place, by advanced boundary information.

5.11 CAUSALITY AND EVENT DRIVEN PROPAGATION

Several authors have commented on the importance to time relativity for understanding process execution[Hic09, Bon16, Car18]—already bringing insights from spacetime relativity, and 'many worlds' interpretations of Kripke and Everett[Kri63, Eve56]. Time has been the domain of physics for centuries, and it would be a mistake to not pay attention to the full range of patterns developed there. To fully understand causality in distributed systems we need to expand the simplistic understanding of universal past, now, future into a local view in which causal behaviour depends on *all three* in a *scale dependent* way.

5.11.1 PAST, PRESENT, AND NOW

Past, now, and future are representations of order relative to a process of observation. What an observer calls 'now' is a snapshot of its interior state. The interior state is, in turn, a representation of its clock (see interior time[Bur19a]). Obviously, this is not a scale invariant assertion. Agents may be aggregated into superagents, which are the smallest grains on a larger scale.

The common view of causation is the retarded view:

Definition 108 (Retarded process)**.** *In a chain of dependent promises, a process depends on an invariant initial state or boundary condition. The final state of the agent does not play a role in determining the outcome of the process.*

Example 75. *In a process to build a tower, the balance of the project bank account starts with the invariant boundary condition of zero money. Its final state is a sum of transactions related to that initial state. The final outcome of the tower plays no role in determining the final amount in the bank account.*

The contrary view, often used in radio engineering is:

Definition 109 (Advanced process)**.** *(includes desired end-state, recursion, etc). In a chain of dependent promises, a process depends on an invariant final state or boundary condition. The initial state of the agent does not play a role in determining the outcome of the process.*

Example 76. *In the space race to the moon, the final invariant outcome of the process was to land a person on the moon. The chain of transactions leading to that point was not dependent on the initial conditions of the project.*

In the latter case, the final state of an agent is implicitly or self-determined, and the promises work backwards to search for a path to reach it from the undefined initial state. This approach is used in transactional 'rollback', for instance. It's also how a GPS navigation system works, for example. A process to solve a Rubik's cube is also anchored in the invariant future state (the desired end state of ordered colour[Car18]).

Example 77 (Proper time clocks)**.** *For example, the increments of time for a cloud process could be measured by ticks that represent the starting and stopping of container processes. Or we go count each function call as a tick of a clock, or each statement. This is not nit-picking: it matters to the issue of causality how we define the evolution of progress.*

In a flowchart view of programming, which represents the most common imperative view of time, the future is thought of as a function of the past.

$$T_{\text{next}} = f(T_{\text{this}}). \tag{5.146}$$

Each prior statement leads inevitably to the next by an implicit jump instruction in the process counter. No statement changes the past, because everything advances at the same rate: the result of each statement is the essentially deterministic keeping of an exterior promise of its agent.

At a function call level, this is somewhat ambiguous, because a function call involves recursion, which poses the promise of the outcome before the execution the keeps the promise, i.e. in the assignment $x := f(y)$, the right hand side is assumes that $f(y)$ exists, which involves stack frames to create a sideways dimension of 'subtime'[62], whose incorporation into the process is acausal from the perspective of a programmer. Past and future get muddled by an assignment that behaves like an advanced boundary condition, while a subroutine advances with a locally retarded boundary condition of the function argument. The discrete scaling of a process into lumps, or subroutines, implies that time does not run in a simple fashion for any observer outside the system (see figure 5.17).

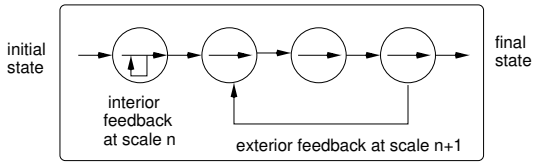


Figure 5.17: Feedback on the functional scale, including iteration and recursion, leads to apparent causation in the reverse direction relative to large scale exterior time (against the observed flow as seen by an exterior observer), but because this is unobservable, the promised outcome, measured by exterior clock time always appears to flow in a constant direction from start to finish. This is what we interpret as from past to future. The direction of time on a large scale is from left to right, but inside the subroutines it may be counter to this monotonic progress.

Functions that are non-deterministic may also employ data that are not accounted for by the promises of the agent[BC11]. Such systems are known to be irreversible, but can be made to behave consistently by using advanced (exterior future) boundary values.

Definition 110 (Interior feedback)**.** *Interior feedback at scale n is a causal sequence of messages whose channel runs counter to the direction of the system's proper time at scale $n+1$. It is unobservable from the exterior of the agent containing it. In other words, the process clock only ticks after iterations and interior subtime machinations have reached an outcome that can keep the agent's exterior promise.*

Feedback may appear causal or acausal depending on the scale of the agent making that assessment. This only illustrates how the meaning of time is naturally complicated in distributed multiscale processes. It's neither deep nor trivial.

Definition 111 (Exterior feedback)**.** *Exterior feedback is the same from the perspective on the inside. A dependency from downstream of the process (the causal future) which is merged with a dependency from upstream (the causal past).*

5.11.2 FACTS, MESSAGES, AND EVENT HORIZONS

Messages are the transport mechanism for program transitions between agents. Events are the observation of a state transition by any agent O. The preservation of an event, as an immutable fact, is not *a priori* guaranteed by any agent. It is design choice (a promise) made by the receiver, whose default is non-immutability. An invariant promise interior memory to remember it, and—since all resources are finite—there will be a cut off lifetime for such facts to be remembered (an event horizon). As the scale of a dependent process, it encompasses an increasing amount of memory, which implies a growing power cost and increased interior time latency for data retrieval. Eventually, the ability to recall prior facts must become much greater than the lifetime of the agent's promise lifetime.

Example 78. *This cost has been made clear in the early blockchains, where coherence or consistency of the chain (the transaction journal) is the causal promise.*

The idea that the past informs the future is too simplistic for distributed processes. A model of computation is a model of causally ordered events, but the order of causality is actually undefined because we place certain information in the rules of propagation and other information in the boundary conditions.

Einstein taught us that causality is what an observer sees. The arrival of messages, leading to events, defines a perceived direction for time for each observer independently. It is always measured at the scale of whatever observer assesses it. What happens on the interior, including acausal feedback loops, is usually discounted (see figure 5.17). Interior process sequence numbers may be used to pay for determinism of causal ordering by coarse graining time, i.e. by paying for order preservation with a delay in *interior time*; this may not appear to delay the process on the exterior, but adds a cost in terms of unique distinguishability of messages to sender and receiver. Other information, like desired end states, fixed points and other 'attractors' may bring about convergence around states that only exist in the relative future, and a process only ends (in interior time or subtime) when that future state has been reached[63].

5.11.3 REPEATABILITY AND SINGULARITIES (FIXED POINTS)

The true goal of information systems as tools is to strive for repeatability or predictability. It should now be clear that this is about the larger goal of arranging invariance over

process conditions, i.e. dependencies. The surrogates that often stand in place of this, such a statelessness, and causal ordering, are themselves non-invariant characteristics and should therefore be avoided.

A common mistake is to try to assure invariance by acting 'only once' (the FCFS random walk approach to state, rather than the determined fixed point). For example, in the delivery of a transaction. We might number transactions, like TCP sequence numbers, and tick them off a checklist as they are completed. This leads to a growing process memory (a stateful process). It can be replaced by a memoryless local process using advanced causation.

Advanced causation (treating the end state as a fixed point) has many uses. Systems whose interior states are changing may not have homogeneous transitions, but their choice of attractor can either lead them to trouble or redemption. This goes beyond the possible interferences noted for invisible changes to global state, leaky channels etc.

Relying on thing that happen only once is a non-invariant Messages may be echoed and isolation is not a promise that can be kept easily (process isolation is often the first thing violated by intrusions and security exploits). If we seek a deeper level of safety, it makes sense to rely not on the keeping of promises that are fed as data, but on the characteristics that are more likely to be preserved, such as convergence to fixed points[64]. The surest means to achieve repeatability is the maintain the promises on a timescale shorter than that at which they are sampled. This is the Nyquist sampling theorem in action.

Advanced propagation determines based on a desired state x_D

$$x_{\text{end}} = f(x_{\text{any}}) \tag{5.147}$$

$$x_{\text{end}} = f(x_{\text{end}}) \tag{5.148}$$

We see that the final value is insensitive to the initial value, which is in strict opposition to the functional idea of past forming immutable facts. The immutable fact lies in the definition of the function itself, which refers to an 'inevitable' future state.

The outcome is idempotent when it reaches its final state, not after a certain number of transactions 'once only' has been reached[Bur4 a, Bur04c]. The approach is what the immune system does, and was used famously in CFEngine[Bur95, Bur04c] and later configuration tools[65]. It's also the approach used in pull requests, and GPS locators. The processes are designed to favour a predetermined outcome. The outcome will only become an event in the agent's future, and will only be observable as a future event by other agents that depend on it.

On the interior of a process, a fixed point of a chain satisfies conditional promises:

$$\begin{aligned} A &\xrightarrow{+X_p|X_i} A' \\ A &\xrightarrow{+X_p|X_p} A'. \end{aligned} \tag{5.149}$$

The more familiar retarded process is a Markov chain, to some order, and has no deterministic end state unless the agents keep their promises perfectly, which is essentially impossible to promise.

Example 79 (Closure operator)**.** *A closure operator $\hat{C}$ over a set X has three properties. It is:*

1. *Monotonic: i.e. if $x \subseteq X$ then $\hat{C}(x) \subseteq \hat{C}(X)$.*
2. *Extensive: i.e. $x \subseteq \hat{C}(X)$, i.e. there is a fixed point, in the equivalence generated by the action of some quotient generate G, so that the quotient set $\hat{C}(X)/G = x$.*
3. *Idempotence of the operator: i.e. $\hat{C}^2(X) = C(X)$, so that x doesn't leak.*

The fixed point x belonging to each closure operator C is an advanced boundary condition. The extension of the x to its closure set is directly analogous to the extension of a partial order to a pre-order by a scale transformation.

It is also an equivalence in Category Theory that a so-called 'monad' over a poset is precisely the closure operator:

Example 80 (Monad over partial order $\preceq$)**.** *A monad over a poset is precisely a closure, for the following reason. A monad on a category X is a tuple $\langle C, \mu, \eta \rangle$, consisting of*

1. *A functor $C : X \to X$, that maps between self-similar structures on the same category, like a symmetry quotient generator. So $x \preceq y$ implies that $C(x) \preceq C(y)$.*
2. *A natural transformation $\eta : 1_X \to C$, so there is a map transforming every x into $C(x)$, i.e. $x \preceq C(x)$, i.e. C is extensive.*
3. *A natural transformation $\mu : C \cdot C \to C$, so that the map is idempotent on its set: $C(C(X)) \preceq C(X)$.*

All the category formulation does is rename a few things, and extend the well-known properties of the mappings between elements into formal classes and subclasses.

Example 81. *CFEngine convergent operators The CFEngine operators discussed in volume 1 were precisely closures on an orthogonal set of policy promise states [Bur95, Bur04c]. In a set of computers, user and environment behave like a collection of sources S_i that lead to the divergence of state in a number of computers R_j that one would prefer to be replicas of a desired configuration state. The CFEngine policy language was a set of operators $C(q)$ that would accept any system state $q = R_j$ as an input and output a desired state as a convergent advanced boundary condition (a closure), making $C(q)$ the downstream desired selection.*

5.12 SINGULAR REGIONS AND CENTRALIZATION

The uniqueness of singularities makes them points of special dynamical significance. They have automatic semantics. This is a property which is invaluable for calibrating intent, from an information perspective. A single fixed point has no entropy—no distribution of possible bodies. In a functional sense it is 'single valued'. Fixed points and singular instances have a functional role in systems for this reason.

5.12.1 CALIBRATED CONSISTENCY OF PROMISES

The consistency of an interaction between pairs of agents is a form of equivalence relation—a symmetry. When a collection of promisers S_i makes an ostensibly single promise to collection of promisees R_j, are we able to say whether the promises were consistent, i.e. equivalent so that is doesn't matter which S_i interacts with a given R_j, and vice versa?

To define this carefully, we need to use the (-) promise of a recipient or promisee as a *calibrator* of the (+) promises it accepts.

Definition 112 (Calibrated consistency of promises)**.** *A non-empty collection of agents A_i may be said to make consistent promises of $+b_i$ relative to a single hypothetical or real agent O, if it would assess the promises of each agent A_i to be equivalent under the promise to accept a calibrated standard $-b$: i.e.*

$$A_i \xrightarrow{+b_i} O \tag{5.150}$$

$$O \xrightarrow{-b} A_i, \alpha_O\,(b_i \cap b) = const,\ \ \forall i. \tag{5.151}$$

Notice this this depends on the resolution of the observer, and its ability to discriminate between the agents. Promises may be judged consistent relative to their recipient because they promise the same (+) or because they are accepted (-) as indistinguishable.

There are several ways in which an interaction can be observed to be consistent:

- The promisers S_i promise their behaviour is mutually consistent.
- The promisee R_j promises the behaviour observed is coincidentally (or emergently) consistent.
- The promisee R_j promises to render the behaviour consistent by normalizing or filtering.

Now, suppose we assume that a single agents R_1 makes a consistent promise, to a client C_1

$$R_1 \xrightarrow{+b} C_1 \tag{5.152}$$

$$C_1 \xrightarrow{-b} R_1 \tag{5.153}$$

and compare it to the case that a collection of replicas R_i promising the same to a collection of clients C_j. What does it take to ensure that the same consistency is true of a collection of the single collective promise?

$$\{S_1, S_2, S_3, \ldots S_n\} \xrightarrow{+b} C_1 \tag{5.154}$$

$$C_1 \xrightarrow{-b} \{S_1, S_2, S_3, \ldots S_n\} \tag{5.155}$$

In other words, how do we scale from a single consistent promise between two agents, to a single consistent promise between two superagents, rather than merely having many similar promises (see figure 5.18)?

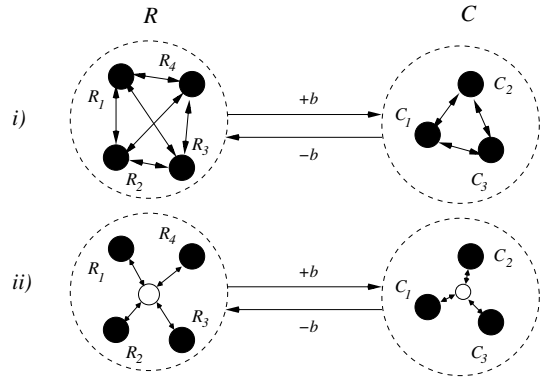


Figure 5.18: The parallel or horizontal scaling of a single promise by consistent cluster of agents. The agents must promise to 'move as one' by coordinating on the interior of the superagent boundary. Note that the interior coordination may be i) by peer, or ii) by central calibrator (white circles).

The coordination between the interior agents may be in the manner of a peer to peer agreement, in which every pair independently equilibrates with every other in a decentralized manner, or it can employ a central hub which acts as a standard for all the others (matroid construction). The latter centralization is cheaper in terms of interactions, being of $O(N)$ rather than $O(N^2)$. It also illustrates the consistency scaling a single source to a larger equivalent source, from a seed of consistency on a small scale, which is

then amplified by promising to replicate outwards. This is the strategy of centralization (see section 5.13).

Consistency seems to be the opposite of singularity. It's dissemination, but we can look at it as the scaling of uniform state.

Example 82 ('Moving as one' (teams)). *Centralized, calibrated consistency is a way to make a system 'move as a unit'. Brain-like controller behaviour allows animals, composed of billions of cellular agents, to act as a single larger superagent—an organism. This kind of coordination may lead to a faster response of the total organism than if its control is decentralized, e.g. like a slime mould. But it might not. That kind of centralization assumes that the communication from command central to extremities is faster than the situations is needs to respond to, and that the lines of communication are protected and reliable. A calibrated brain might be too simplistic in giving the same commands to different parts of the system though. In 'team' collaborations, this ability to move as a singular unit leads to agility for the whole organism, whereas the abilty to move as independent decentralized agents brings local contextual adaptation and resilience.*

5.12.2 TIME TICKS SLOWER FOR SUPERAGENT PROMISES

A very simple observation about scaling is that exterior promises get slower as the interior size of a superagent is scaled up. This is the cost of interior collaboration. This is an intuitive result, which assumes a large fraction of the interior agents make use of serial stimulus-response interactions (i.e. they are not simply a group of agents working redundantly in parallel), but nonetheless an important one. It has implications for the centralization of system functions and for equilibration of consistent data (see section 5.13). The must follow from the assumption that all promises take a non-zero amount of time, as measured by a recipient, to keep, i.e. $T_O(\pi) > 0, \forall \pi$ (chapter 1).

Suppose a single agent A_1 promises to tick at regular intervals. An observer, watching the agent, assesses that the agent ticks every Δt time units.

$$\pi_0 : A_1 \xrightarrow{+\Delta t} O \tag{5.156}$$

The timescale for keeping this promise may be assumed (according to the observer)

$$T_O\left(A_1 \xrightarrow{+\Delta t} O\right) \simeq \Delta t. \tag{5.157}$$

If we now introduce another agent with interior coordination promises between the two:

$$\pi_1 : A_1 \xrightarrow{+(\Delta t_1 = \Delta t_2)|\Delta_2} A_2 \tag{5.158}$$

$$\pi_2 : A_2 \xrightarrow{+(\Delta t_2 = \Delta t_1)|\Delta_1} A_1 \tag{5.159}$$

with supporting

$$A_1 \xrightarrow{-\Delta t_2} A_2 \tag{5.160}$$
$$A_2 \xrightarrow{-\Delta t_1} A_1. \tag{5.161}$$

and exterior promises,

$$A_1 \xrightarrow{+(\Delta t_1 = \Delta t_2)|\Delta_2} O \tag{5.162}$$
$$A_2 \xrightarrow{+(\Delta t_2 = \Delta t_1)|\Delta_1} O, \tag{5.163}$$

The interior promises are conditional, and therefore kept serially, so the time is the sum of all promises in the process, or simply:

$$T_O(\pi_1) \geq T_O(\pi_0) \tag{5.164}$$
$$T_O(\pi_2) \geq T_O(\pi_0) \tag{5.165}$$

The two exterior promises are equivalent to a collective promise:

$$\pi_{\text{ext}} : \{A_1, A_2\} \xrightarrow{+\Delta t|\Delta t=(\Delta t_1=\Delta t_2)} O \tag{5.166}$$

so

$$T_O(\pi_{\text{ext}}) \geq T_O(\pi_0). \tag{5.167}$$

The promise to the outside is now conditional on the equilibrium state, so it must be delayed by the equilibration time, since the agents have promised to keep that promise first. To verify and maintain these promises, even after equilibrium is reached, must still take non-zero time (even if it is less than the worst case relaxation time for achieving equilibrium), so we can state with certainty:

Theorem 4 (Coarse grained time is slower)**.** *The timescale for an exterior promise must be greater than the timescale for any of its dependent interior promises.*

We can go on adding agents to this cluster and increasing the serial time needed to complete the exterior promise. We can't necessarily say how long a process will take, but we can say that it is at least as long as the keeping of a single agent autonomous promise. If the receiver ticks quickly compared to the equilibration process in the replicas, then much time could pass according to its clock (like waiting for water to boil). On the other hand, if it sleeps in between, it may scarcely notice any time pass. This inertial 'mass' of being encumbered by dependencies leads to interior latent processing time.

Note, that even though an observer can observe the time difference, it is free to ignore it, and absorb it into its own promise processes[66].

5.12.3 EQUILIBRATION OF REPLICAS TO SINGULARITY

Achieving consistency is about reaching an equilibrium between the promises made amongst a set of replicas R in a service (see figure 5.19). See also example 53 for some helpful context.

Calibration of the state can be resolved by

- Promotion of a desired state by cooperation (+).
- Selection of a desired state by receiver (-).

We therefore have alternative approaches to seeking consistency. Note, however, that the downstream principle always favours a (-) promise over a (+) promise. Agents may do their best to offer a consistent state (+), and that's nice to have, but ultimately it's the responsibility of the receiver to check.

Let's apply the results above to services. Redundant replication seems to be the opposite of a singularity, but it too can be handled as a desired state, if we rescale our thinking. In services, where we need to scale access to its process 'horizontally', e.g. in scaled data services, clients want to know that the promises made by each service replica agent are the same, so that is doesn't matter which one they interact with. If there were only one agent, the promises would usually trivially self-consistent—though it need not be the case. Agents can easily interleave inconsistent promises to different clients, treating some better than others, but we'll assume that doesn't happen here. At least by minimizing the number of agents assigned a single role, one maximizes the chance of experiencing stable semantics and dynamics from the promiser. The question of consistency is how do we scale that property? What is the meaning of 'one' instance, when agents can be scaled to form superagents that make singular promises of their own?

To address this point, we return to the set of issues associated with the diagram first shown in figure 5.19, and focus on the second stage of the pipeline—the replica sets.

Example 83 (Database replica consistency)**.** *The problem of distributing data or service from a set of sources S_i across a set of identical replicas R_j is a central problem in database design (see figure 5.19). In this problem, each replica plays the role of an observer watching for changes to the source. The replicas may have different views of the source at different stages of their on-going process to collect data. If a client C should ask any of then about what happened at S, they may not report the same answer.*

In computer science, so-called consensus *technologies try to solve this problem by asking the R to agree about their position on S, even if their position on S is incomplete. In other words, no matter whether the answer is right or wrong, all replica agents in R should agree and changes should happen in lock-step.*

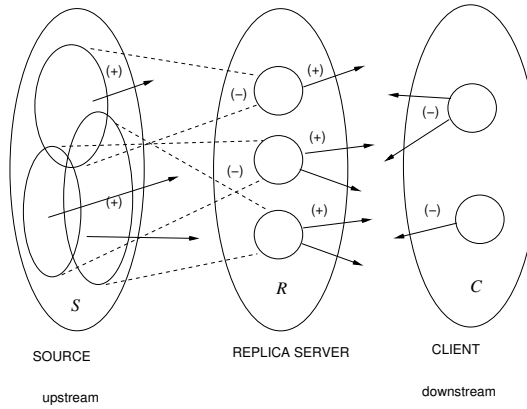


Figure 5.19: Data are stored in a database that consists of a redundant cluster of replica servers. Clients may query any of the replicas at any time. Will they experience a consistent view?

There are therefore two equilibrium processes that need to stabilize in order to receive both accurate and consistent information about S:

- *The process of transmission from S to R.*
- *The process of equalization from R to R.*

Because there is a lag in time to propagate the change from the source to the replicas, or from replica to replica, unless the right of C to observe R is restricted, the observed value may not be synchronized according to the replica processes 'proper clock'. So, without additional constraints on what R promises to C, each named record becomes a random variable.

Principle 9 (Equilibration before observation)**.** *Before measuring a variable, it should be in a quiescent state. For statistical variables the process of equilibration with whatever sources it depends on should have ended before sampling.*

The equilibration before observation sets up a race between competing processes that every system engages in either intentionally or unintentionally.

- Equilibrium between S and R:

  Since S is time dependent, by assumption, an equilibrium between S and R can only be temporary. If S changes before R can catch up, R will never be able to promise a faithful representation of S[67].

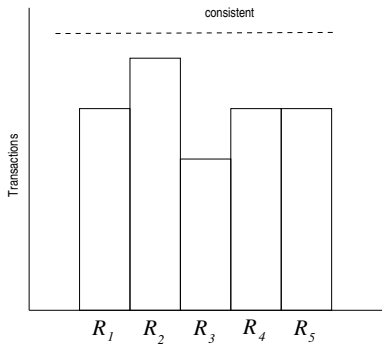


Figure 5.20: When racing S to fill up a number of replicas of S's history, the process needs to converge to a persistent fixed point. The convergent fixed point for policy is a future bar in which the cumulative transactions of the replicas converges to a desired state.

The race for R to keep up with changes from S is equivalent to a process of filling identical buckets $R_j = \{R_1, R_2, \ldots, R_n\}$ from a set of sources $S_i = \{S_1, S_2, \ldots, S_m\}$, for some n, m. There are two ways in which this process can be conducted:

- By 'push': S imposes data onto a single recipient in R, which must then impose the changes onto its replicas.
- By 'pull': Each R samples each S independently. If the sample sets are complete, there will be equilibrium of R as soon as each R has finished (see figure 5.20). If the sets are different, the R need to undergo some algorithmic process to reach agreement about the complete state of S.

In both cases, R needs to outpace S in order to capture all the data.

If we sample at regular intervals, or map the imposed signals into data records the simplest way to achieve equilibrium for a sampling interval is to make new records for each time step, i.e. not to overwrite a value, but to use sequence numbers for samples. Then, even if the buckets in R are slow to fill up, they will fill *eventually*. This is known as the *eventual consistency* of replicas. Obviously all consistency happens 'eventually', but we want to know how long 'eventually' is. The problem is equivalent to that of a *queue*, somewhere between S and R (see volume 1). The race is only won if the rate of acceptance at R (often called μ_R) is greater than the rate of change at S (called λ_S).

We can express the condition to win this race in different equivalent ways. The simplest case is to start from single agents S to R. From a rate perspective, the average rate of change of S needs to be much less than the average rate of change for R so that R can outpace S and easily match what S does. Imagining smooth differentiable processes, we would need:

$$\frac{\partial S}{\partial t} \ll \frac{\partial R}{\partial t}. \tag{5.168}$$

In fact, the processes are not smooth or differentiable at all, so we can instead appeal to Nyquist-Shannon sampling law for discrete samples[SW49, CT91]. For a discrete process, the sampling law tells us that R needs to sample each agent in S at least twice as fast as S changes in order to capture all of its changes. If f_X is the sampling rate of X, then:

$$f_R \geq 2f_S, \quad \forall R, S, \tag{5.169}$$

or in terms of queueing

$$\mu_R \geq 2\lambda_S, \quad \forall R, S. \tag{5.170}$$

As far as the rates are concerned, if the source data are constant $\partial S/\partial t = 0$, then it is straightforward to catch up.

When data are pushed to a single agent in R, rather than all sampling independently, then we still need to address how the R replicas reach internal equilibrium too, for each change in S. The fixed equilibrium needs to persist for long enough for C to see it. For a single agent, this

- Equilibrium within R :

  The time for a cluster of agents R to reach equilibrium cannot easily be predicted, since it depends on many factors. We can write it as a polynomial in the number of agents n within the cluster:

  $$\Delta T_R \sim \left(a + bn + cn^2\right) \Delta T, \tag{5.171}$$

  for some timescale ΔT, and some constants a, b, c.

  We know that, if the source data are kept constant, i.e. $\partial S/\partial t = 0$, then it is straightforward for R to catch up eventually. One way to assure that is to keep every unique sample as a proper sequence of versions (like a version control system, or timeseries). Then, even late arriving data will be mapped to the correct sequence bucket, and new data can be separated cleanly from old. Each source

data sample is then effectively constant and treated as immutable. Accidental repeat measurements are of no importance, since they are idempotent by virtue of being constant. A policy of immutability effectively engineers a fixed point for each data sample, to which the system can converge, and a fixed bar of eventual convergence for the cluster R (figure 5.19).

The concept of consistency amongst a collection of replicas is a promise made by the replicas. It might depend on the promises originally made by the sources, but that is not relevant. A promise of consistency is a promise to act as a single valued function. For every question, there is one and only one answer.

So far, we assume only that we need to get R into a consistent state. But what about the client C, who is getting second hand information about S from R. It will try to contact an agent in R for service. It might be lucky and get an up to date picture, or it might be unlucky and get a slow agent that lags behind. Once again, the introduction of a new stage introduces a sampling process and turns apparently deterministic data into a random variable. If version sequence numbers are used, the client may not see an up to date version, but it will be able to discriminate the freshness of the result.

One way to prevent observation of inconsistent results is to limit the observability of the cluster R until is has reached equilibrium. Let's assume that the cluster R has a process by which it can assess equilibrium, then we define the promise of equilibrium for some service query q_S as

$$R \xrightarrow{+\text{Equil}(q_S)} C. \tag{5.172}$$

The replicas can now limit observation of any result by making it conditional on the equilibration of q_S:

$$R \xrightarrow{+q_S \mid \text{Equil}(q_S)} C. \tag{5.173}$$

Moreover, we are tacitly assuming that no other changes occur in R, i.e. it is isolated from tampering. In this case, we can claim that C will only observe promises that are kept consistently over R.

Example 84 (Transactions isolate and limit observability)**.** *In a transactional system, all replicas are always changed in lockstep, in an isolated process protected from tampering. Transactional consistency is simply eventual consistency over a single forced time step, enforced by a lock. This is applied to data [GL06] or operations [BS97]. If a consistently disseminated state is the* desired state *in a convergent process, then we must treat the scaling of the entire region as a single agent. It is then singularity on a larger scale. Such a singularity is a closure.*

Example 85 (Eventual consistency in blockchain)**.** *In a blockchain there is no strong coordination between agents over the state, rather there is a uniform selection policy. As new blockchain data arrive on a FCFS ordered basis, different agents may append data to their parallel replica of the chain, creating more than one version. The singularity of the chain's structure (in spite of its replication) is now compromised. However, it can be restored by the selection of a desired state (figure 5.20 applies). The desired selection is the the longest chain, which is a quasi-invariant advanced boundary condition. With agents promising to make this policy selection, the replicas can converge to a single candidate everywhere.*

Example 86 (Push requests in transaction logs)**.** *Most centralized databases receive data by imposition from outside sources that 'push data in'. The database isolation kernel attempts to serialize these transactions into a transaction journal or log, which defines a convergent order (figure 5.20 applies). The desired state of a push log has no convergence principle—the database is essentially a random variable and just works as hard as it can to keep up with impositions. Impositions that cannot be accepted are dropped, like network switches. Replicas of the database R_j pre-select a single leader which is the one accepting impositions from sources S_i. The replicas promise to try to keep up with converging towards a desired state defined by the transaction log. This is a retarded boundary condition, together with an extensive journal of delta changes to reach convergence. This approach has been called a* congruent *(instead of convergent) approach. It is strongly dependent on isolation, as it can be shown that the replicas will not converge if they can be tampered with from sources other than the transaction log.*

Example 87 (Pull requests in transaction logs)**.** *In version control systems, like git, developers act as sources S_i and fork replicas R_j of the repository code. They modify them creating divergent 'parallel worlds'. The authors of these branches act as sources S_i and may submit changes to be integrated into the desired state of the main version branch, which acts as a receiver. The convergence policy is different from the blockchain example. Now, the main branch policy agent is effectively the client C, who selects can integrate parts of the state from the other branches, in no particular order. The altered main branch is then automatically selected as the desired state (by default), bringing convergence of the many worlds back into one.*

Note how the downstream principle applies again: it's the client C's responsibility to select the new desired state and repair the 'fault' created by the divergence.

Example 88 (Eventual consistency in synchronous transmission)**.** *By forming a co-dependent superagent between the sender and receiver of a promised message, one can*

lock the assessment of a time step in the manner of a closure operator so that neither side can advance without the other (see example 79). This idea has been used to implement a form of reliable distributed network transaction on a very low level[BBKK18].

Example 89 (The CALM conjecture for eventual consistency)**.** *The CALM (Consistency as Logical Monotonicity) conjecture proposes that a computer program has a consistent, coordination-free distributed implementation, if and only if it is monotonic[HA19]. As stated, it is missing the need for the full closure properties, and symmetry between independent data sampling processes across the cluster of agents we wish to assume consistent, but is essentially correct, as a trivial application of advanced causation, as proven in [Bur04c]. Of course, this says nothing about how long such consistency may take. Convergence without coordination must therefore assume much more than monotonicity of process—each agent has the independent responsibility to source the correct prerequisite dependencies in the correct version, unless the process approaches a fixed point that is independent of source data altogether, such as a constant policy determined state (an advanced boundary condition).*

5.12.4 TIMESCALES IMPLICIT BEHIND CONSISTENCY

Consistency in a changing environment is a race against 'time'. Whose time? If we refer, once again, to figure 3.4. There is a process leading to a source of change S, and a set of replicas R that attempt to capture the changes, so that they might report them to various clients C. Will all the C a consistent answer to a consistent question?

There are hidden assumptions behind promises of consistency. The first crucial assumption is that the proposal or desired outcome is invariant over the lifetime of the consensus process, else one could not stabilize a transmission in a particular direction. The proposal exposed in the first step of the agreement process, described in the previous section, may not change as the agents go about their interior promises to observe, copy, and agree to it. In the language of clocks, the object of agreement, in any common knowledge problem, needs to be persistent on a timescale longer than the promises to abide by the intermediary steps. So, whereas one typically talks about 'correctness' in computer science (a semantic assessment), it is more a question of stability (a dynamical assessment), or invariance of assumed targets, during the key change processes[Bur13a, BB14a].

So, to converge on a stable target, it is assumed that the timescales (as measured on the clock of some godlike observer) for the lifetime of the exterior promises to transmit with integrity, must be significantly longer than the timescale over which the interior promises are defined and kept, by a good margin, else one is racing against a moving

target:

$$\Delta t_{\text{exterior}} \gg \Delta t_{\text{interior}}. \tag{5.174}$$

It is not coincidence that this is also the condition of equilibrium, with equilibration or 'relaxation' time t_{relax} for transactions:

$$\Delta t_{\text{exterior}} \gg \Delta t_{\text{relax}} \gg \Delta t_{\text{interior}}, \tag{5.175}$$

or equivalently:

$$\Delta t_{\text{common knowledge}} \gg \Delta t_{\text{transaction}} \gg \Delta t_{\text{TICK}}\ , \tag{5.176}$$

In our technical implementation, this translates into the rates of interior TICK /TOCK processes relative to the rate of new messages M.

$$\Delta t_M \gg \Delta t_P \gg \Delta t_{\text{TICK}}\ , \tag{5.177}$$

We expect to be able to achieve consensus about M is there are many more TICK events than there are new messages. This (hopefully) seems like an obvious point, but most consensus discussions brush over such limitations. The upshot is that an ideal technology should seek to maximize the rate of interior equilibration. Faster is always better.

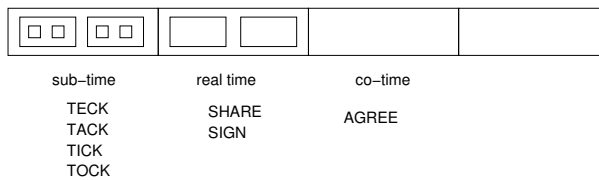


Figure 5.21: Locally, the composition of timescales for quantization of certainty may be viewed at three scales: individual subtime transactions, round trip times, and co-time assured transfers. Using the analogy of version control, only the co-time units are 'committed' as accepted new versions of the current transaction. Equilibration time is kept comparable to subtime, according the the observer's clock at each end.

The core assumption (often unstated) is, therefore, that consensus equilibrium systems is that the data cannot be allowed to change faster than the superagent can reach equilibrium or consensus. Moreover, things that depend on the value need to be frozen for the duration of the message transfer, so M is a slowly varying quantity.

Example 90 (ACID and BASE in databases)**.** *The promises made by databases are the subject of much discussion in computer science. Two broad philosophies have been*

summarized by the whimsical acronyms ACID for 'Atomic, Consistent, Isolated, Durable', and BASE for 'Basically Available, Soft state, with Eventual Consistency'. These are often treated as fundamentally different but are, in fact, rescalings of one another.

ACID makes promises about the relative timescale of processes, with a sharp separation between interior and exterior time (see section 5.14), while BASE does not. The key term 'atomic' implies that a process's exterior time should only tick once for each completed transaction, and a transaction implies that—within an isolated kernel that accepts no input from any source but the process transaction data—all replicas of that kernel of agents will promise the same information when the transaction finally ticks. This may involve 'waiting' or 'blocking' of the process on the scale of interior time (subtime). This is a deliberate manipulation of observability, widely used to provide mutual exclusion locks (mutex) for processes in computers. The isolation boundary is essential to the partitioning of time, and to the ability to keep the promise of identical replicas.

In BASE, there is no waiting for a transaction to equilibrate across all replicas, so results can be observed more quickly, on the clocks of exterior agents. There is no block on observability. One speaks of eventual consistency, as there is no promise made about when the data in replicas will be identical or complete.

In both cases, designating a dataset by either of these requires a notion of when an atomic or eventual process is complete. This builds on the notion that each atoms represents a desired, convergent (or monotonic, idempotent) state.

Example 91 (Adiabatically smooth systems)**.** *In smooth systems, as functions of time, where significant mesoscopic change can occur, it's natural to change variable for propagation functions that depend on two times $\Delta(t, t')$ to the interior and exterior coordinates:*

$$\tilde{t} = \frac{1}{2}(t + t') \tag{5.178}$$

$$\bar{t} = t - t', \tag{5.179}$$

where the average exterior timescale for change characterizes $\bar{t}$, and the interior time, which approximate relative time translation invariance is represented by $\tilde{t}$. The same transformation can be applied to the position variables for propagation in spacetime.

5.13 CENTRALIZED AND MONOLITHIC SYSTEMS

An immediate further application of the use of scaled singularities for system coherence is to be found in the meaning of *centralization* and decentralization. This is a widely misunderstood issue, as a few examples reveal. Our intuition about centralization is

based on position rather than function. This is a direct application of the equilibration consensus problem in section 5.12.3.

Definition 113 (Centralized system)**.** *A centralized system has a star configuration of promises at its core, i.e. all promisees feed or depend on a singular source. Let R be any superagent, composed of any number of subagents R_j, then if each agent promises*

$$R_j \xrightarrow{+q_j} C. \tag{5.180}$$

$$C \xrightarrow{-q_j} R_j. \tag{5.181}$$

A centralized system might maintain different contextual promises to each of its satellite clients. The client C is centralized with respect to all the R_j, but it C is not necessarily monolithic with respect to R_j—only if $q_i = q_j$ for all i, j. Centralization may be physical or virtual—there is no implication about size or relative distribution of the interior parts of the central system as the definition show: both C and R_j can be centralized.

Lemma 23 (A single atomic agent is centralized)**.** *The promise of equilibration is trivially true for a singular, elementary agent, and thus the centralized system property is trivially true.*

Definition 114 (Monolithic system)**.** *A monolithic system promises to act as 'one' entity, by exposing an opaque interface to agents on its exterior. Let R be any superagent, composed of any number of subagents R_j, then if each agent promises*

$$R_j \xrightarrow{+q_S \mid Equil(q_S)} C. \tag{5.182}$$

$$R_j \xrightarrow{+Equil(q_S)} C. \tag{5.183}$$

to an exterior observer or client C, it behaves as a coherent or singular singular source for all intents and purposes to C, because the singularity of the equilibrium dependence renders all parts equivalent. A system whose promises appear to emanate from a single boundary set of agents, e.g. a single point of contact or a number of points that act as an promise interface to outside promisees, is therefore monolithic. The boundary set is opaque to interior promises.

In a monolithic process, when a promises can't be kept a promise-keeping process will be blocked until the equilibrium of any redundant members can be promised internally. A centralized agent may or may not promise to ensure its own interior consistency before keeping its exterior promises—but, if it does, then it is effectively monolithic.

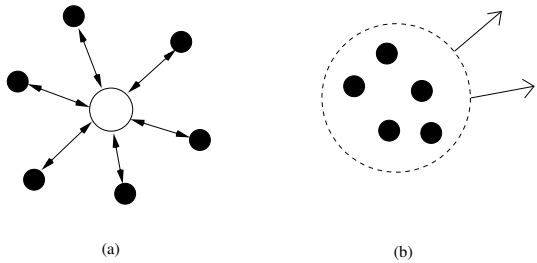


Figure 5.22: The difference between a monolithic and a centralized system is subtle: (a) is centralized as all parts rely on a single dependence; (b) is monolithic because the system has an effective boundary that acts as an entity. The core of a central system is often monolithic.

We note the difference between a singular or centralized system and a monolithic one. The perception that a system is monolithic depends on the assessment of its apparent interface. Are we talking to a single entity or to a collaboration of different representative processes? On the other hand, the perception that a system is centralized is decided by whether any perceivable interior components act tightly and inseparably within a singular boundary.

Example 92 (Monolithic computer software)**.** *A monolithic application is computing is a program that exposes all its independent promises through a single interface, so it appears to be an indivisible entity. This is sometimes used to accuse systems of a lack of modularity, but that assessment can't be made as the interior details are unknown. From the interior of a monolithic system, the agents could have a highly modular structure, whether as parallel or serial collaborations. The level of modularity is a scale independent characterization, relative to each exterior promise.*

Example 93 (Centralized computer software)**.** *A centralized application is one in which the entire system acts as a singular entity. No matter how much parallelization is concealed behind a boundary set, an exterior agent is unable to tell the difference between different interface agents on the boundary: they make identical promises to the exterior.*

Lemma 24 (Monolith versus centralization)**.** *A centralized system is monolithic if and only if it does not discriminate between the agents it promises.*

The distinction between monolithic and centralized is a subtle one. A monolithic system may be perceived as decentralized on the scale a particular observer, even though it is unable to complete its function without all the parts. The parts may lack coherence in space and time, but the final outcome promised by the complete system is singular.

Example 94 (Central and distributed systems in computing)**.** *In the past, centralized systems were single monolithic computers. Computer scientists and engineers sometimes assume that adding more computers connected by a network creates a decentralized system. Such a system is obviously physically decentralized on the scale of a single computer, but it may be centralized on the scale of a datacentre (rather than scattered around in people's homes). Moreover, a network of scattered computers may still be* virtually centralized.

To understand the uses of centralization or monolithicity, we need to look at all the different kinds of interactions the component parts of the whole make. A system may be centralized with respect to one function and decentralized with respect to another. Nearly all systems have some functions that rely on centralization or monolithic response. All specialized modules are mini-centralizations. For example, the organs in an organism, or the organisms in an ecosystem are locally centralized. In information systems, the lookup of names and addresses in directory services is centralized to a special service. All service points on the Internet have a single address entry location on the web.

Example 95 (Coordination by consensus)**.** *Over the past decade or more the practice of using small key-value pair databases to coordinate distributed systems has become increasingly common. Consensus technologies like Paxos, Zookeeper, Raft derivatives (consul, etcd, and so on) are used to centralize the command and control, index data for services to scale them horizontally. These key-value clusters act as a single entity for all intents and purposes. The cost of scaling the equilibration of key-value data limits the size of the clusters to a handful (typically 3-7 replicas).*

Example 96 (Global symmetries and cloning)**.** *One of the mysteries of natural science is why there are apparent global symmetries—for example, why all the points in the universe seem to behave like all the other points. Also why tissues of cells are all the same. John Archibald Wheeler once exclaimed that he understood why all the electrons are the same—because there is only one electron, and everything we see is an image of that. This is a matroid construction[Bur14] to explain a scalar property. The universe itself is not a single point, but the theory of the big bang tells us that all points have been effectively clones from the original singularity point far back in the history of the universe—just as cells in a biological organism all cloned by cell division from a single image. All points have common causal origins. Singularity calibrates properties and propagates them to generate global symmetries.*

Some properties of monolithic system:

1. Clean separation of interior and exterior promises, where the interior promises are kept at least as fast as the exterior promises.

2. Coherence of exterior promises.

3. Coherence is useful for *calibration*, i.e. making singular promises that can be used as a 'standard candle' comparison scale, as explained in section 5.12.

4. Dynamically, a centralized controller avoids the problem of inconsistency of imposition, i.e. contention when intentions are imposed, also called the split brain problem.

5. Monolithic systems are a single point of failure, on the exterior scale of the agent.

Some properties of centralized system:

1. Central systems are a single point of failure, on the exterior scale of the agent. Internally, their strong coherent coupling transmits faults efficiently so that the working state of the centralized system is to behave as a single agent—somewhat binary in its promise-keeping (all or nothing).

2. Some centralization may be based on locality, and for economic reasons, e.g. exterior redundancy is too expensive or too complicated to achieve[68]. Thus, in spite of the fragilities of centralization as failure points.

3. During scaling, a central point acts as an anchor for surrounding parts, like a seed, or a skeleton frame on which to hang other parts. This is the function of basic utilities and *infrastructure*.

The potential for monolithic and centralized control is thus to make a kind of 'brain' or controller model for systems (section 2.9) if it can respond quickly enough to be faster than the exterior promises it needs to keep, i.e. a place in which information can be compared and correlated, mixed and matched, very quickly. Thus monolithic 'brain' models need to scale *vertically* (see section 8.6) in a coherent manner, and this implies the following properties:

STRONG COUPLING	CO-DEPENDENCE	LOCKSTEP
ANCHOR COUPLING	UNILATERAL DEPENDENCE	FOUNDATION
WEAK COUPLING	SPORADIC DEPENDENCE	SPECIALIZATION

Command structures are often centralized for reasons of simple calibration or contention avoidance, for example. Services may be centralized for consistency and for coherence of integration.

Direct couplings can't always be easily made redundant, without significant changes in design, but if we rethink the scaling argument, those singular or 'centralized' agents can instead contain interior redundancy (like biological organs do[69])—so centralization

is not necessarily the lack of redundancy, but rather the relocation of redundancy to the interior of a coherent boundary. Any decentralized systems can bind together strongly by entanglement thus forming locally centralized singular agents. Of course, this makes them more fragile to shocks internally as the strong coupling transmits influence without loss, whereas weakly coupled systems maintain a degree of isolation and resilience.

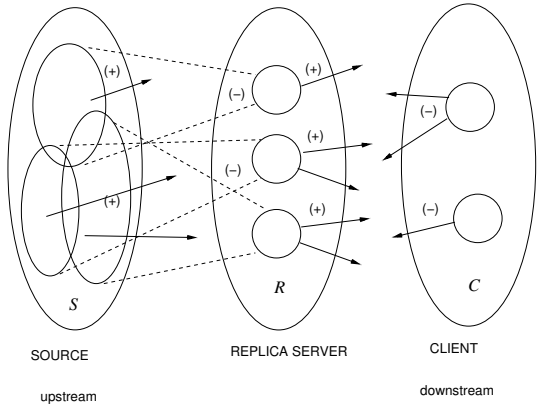


Figure 5.23: The aggregation of promised outcomes from multiple sources is another example of the staged promise scenario.

Definition 115 (Decentralized system)**.** *Any system of more than one agent, which is not centralized.*

A decentralized system is not necessarily uncoordinated. It can still make mutual promises. But what we see is that, if we zoom out to a larger scale, nearly all interacting systems eventually appear centralized.

It'd equally tempting to define a distributed system as a system which is not monolithic, but this would contravene common usage.

Example 97 (Symbiosis)**.** *When agents depend on one another, even partially for mutual benefit, we speak of symbiosis.*

$$A \xrightarrow{X|Y} A' \tag{5.184}$$

$$A' \xrightarrow{Y|X} A. \tag{5.185}$$

This is also called entanglement. See also the discussion in section 2.6.

Example 98 (Scaling of biological organisms)**.** *West also showed that energy transport is the limiting factor in the scalability of biological organisms, and space-filling networks[Wes99]. Bettencourt's work on cities suggested an analogous role for the transport and communications infrastructure in cities, as non-biological organisms[Bet13]. These data points provide a level of confidence that the principles of a promise network are correct*[70].

Example 99 (Centralization may not be easily recognized)**.** *Consider a centralized biological organism, with organs connected by a network, and wrapped in a skin boundary (figure 5.24). The heart and brain seem centralized, and organs like kidneys and lungs have independent redundancy, and are therefore decentralized subsystems. Looking at the definition of centralization, we see the conditional promise of blood is to deliver services R to client organs C. In the interior scale, the most pervasive singular dependency on which all the parts depend is the blood. This seems to be the least centralized part of the body, physically, but its functional role is the most centralized—the blood is the binding hub on which the body holds together (not brain or nervous system, which may be a close second).*

Example 100 (Centralization of power—authoritarianism?)**.** *The centralization of power is usually characterized as implicitly negative in social discourse. That prejudice doesn't stand up to scrutiny. Authoritarianism is likewise usually assumed to be monolithic. This is also false. Centralization does imply authority as single source, with or without a mandate from clients that rely on it. However, a central system does not have to be monolithic. The central agent can discriminate between its clients to maintain a 'split brain' differentiation to whatever end. A centralized source is faster at coordinating a collective superagent than independent equilibration, and with greater calibration of purpose. A slime mould, or a flock of birds can't mount as coordinated response to any challenge as a single animal—even though the animal is a also superagent of cells. This is almost certainly why organisms with brains have been so successful in evolutionary terms.*

Example 101 (Teams or individuals?)**.** *The economics, speed, and reliability of centralization suggest that training a single brain may be more efficient than training a team, as long as the single agent can offer sufficient capacity. The single agent seems more vulnerable to failure; however, if the entire team is needed to keep an exterior promise then the failure of a single member is the failure of the team. This is the nature of strong coupling. Extroverts often argue that 'two heads are better than one', but that is not necessarily supported the considerations above. The case in which two heads would be better than one would be if the heads R_1 and R_2 had not reached equilibrium, and a*

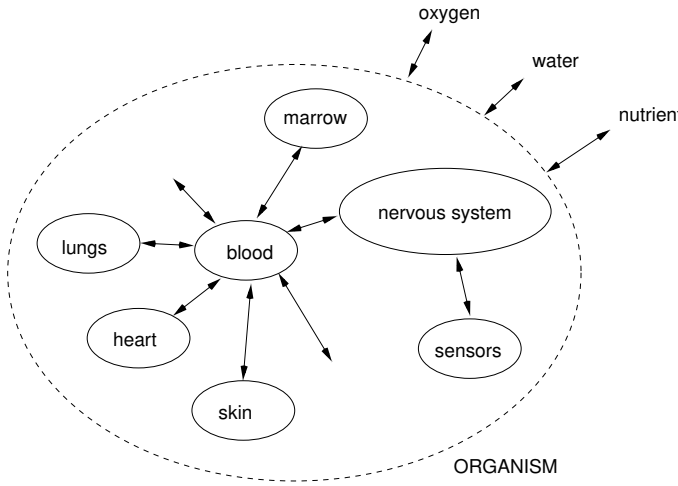


Figure 5.24: The functional agents on the interior of an organism. There is a hierarchy of cooperation, unlike a company or command structure. There is a network of mutually dependent (entangled) services. Each bounded organ may perform a specialized function in the context of an organism superagent. The system that all subagents depends on would seem to be the blood. Thus, blood is the unifying hub for the organism superagent (not the brain). This key role of infrastructure is the main weakness in all shared resource systems. Exterior dependencies on oxygen and water have a similar role. The dependence on nutrients becomes more interesting as this starts a new superagent web of food, which includes other organisms.

possible answer in C could still be found by seeking to equilibrate them, at the cost of the equilibration time.

Although this seems intuitive, Promise Theory shows us that the conventional view is of centralized and decentralized is an illusion of scale. Scaling will play an important role in the integrity of system centralization, as we'll see in chapter 7. Regardless of scale, a central (super)agent must contain internal resources and structures in promise its capabilities. The fact that these are localized (or that we choose not to open its black box), should be neither here nor there. The positioning of the agents in a system, whether real or virtual, is less relevant than their ability to keep promises, regardless of location.

- Does the relocation of agents, in the absence of other changes, affect the ability of the system to keep its promises? This might depend on the nature of the agents. If we put a fish out of water, for instance, it might not perform very well; but, this would be 'other changes'. If we assume that *all* the promises (including

environment) are equivalent and equally reliable, then there is nothing expressed in the promises that refers to (or relies on) location at all; thus, location cannot make any difference. If we outsource a department to India, or

Example 102 (Outsourcing)**.** *Human outsourcing does not respect any of these invariances mentioned here. Taking a local department, interacting regularly face to face with others in an organization (across functional boundaries) and moving them to a place of complete isolation, where only phone calls and emails are possible totally changes the dynamics of the system, so we would not expect it to behave in the same way.*

- If we try to scale the system in such a way that it can promise twice as much as some baseline, what does this say about how the component agents have to change their promises?

  In order to promise twice as much from a single representative service point, that agent needs to have twice the capacity, and the sum of its dependencies needs to generate twice the capacity, to feed it, etc. That is not the same as each agent promising twice as much, because aggregation may then lead to excess capacity, which may lead to erroneous local flooding or downstream queueing, which triggers system failures. Moreover, scaling the promises of each agent component in a system might not be possible, as agents often have hard interior limits. Humans cannot work faster than humans can (without becoming something else), and the same applies to almost any technology. To increase the speed, we might have to replace a human with a machine, or alter the architecture altogether.

The answers to these questions suggest that shrink-wrapping a collection of components into an enclosed space, building a fence around it, which is what we mean by a *centralized* or *monolithic* architecture, does not change a system fundamentally. The differences must lie in the hidden details that are only implicit in calling a system centralized. Conversely, picking a system apart and scattering it around, without a fence, need not fundamentally change a system. That said, it is rare that 'equivalent' centralized and distributed designs are in fact equivalent in all the promises they make. The distances between agents might affect the speed at which promised results can be transmitted from agent to agent. Conversely, the proximity of agents might result in a bottleneck or resource limit, or an inability to provide proper dissipation of output (heat, waste, or produce).

As always, there are two dimensions to the equivalence of systems: dynamics and semantics. Except in the case of mission critical engineering, like aerospace and shipping, the science of understanding dynamical equivalence is not taken very seriously for human-information systems. Inequivalences could arise because technologies are not able to

scale without replacing an agent for an inequivalent agent. Even without replacement, agents may not scale linearly in their behaviour: capabilities, costs, performance, and even semantics typically change as one scales them. All of these issues may introduce semantic and dynamic differences into a system that need our attention when trying to rescale an effort to larger or smaller size. Designing a system to be tolerant of scaling goes beyond mere fault tolerance.

Comment 5. *When intent is distributed at multiple locations, there might be differences in the ability to access the agents, for information to propagate freely, and to gather knowledge about them and their promises. This can make understanding the system harder. A central system has a natural point of calibration, by assumption, and thus a single timeline. A distributed system, being only loosely coupled,is more likely to have multiple clocks and rates of change in its processes. Knowledge management, and reasoning about distributed systems is thus one of the major issues in systems we call distributed. Thus centralization is about the ability to keep collaborative promises between components, rather than their locations.*

Example 103. *Technologies that bind agents strongly into rigid clusters, which can be called logically centralized, are sometimes called consensus systems. Paxos and Raft are such systems[Lam01, OO14] that maintain correlation across agents in a local region. Other systems, like single threaded processes with private memory just rely on the assumption of isolation and change control to claim 'centralized' calibration.*

5.14 BLOCKING AND NON-BLOCKING PROMISES

Another application of fixed point convergence, or centralized and monotonic behaviour, is for limiting the observability of states in a system until certain set of desired conditions are met. This is often called 'blocking' behaviour or 'waiting' in system parlance. It involves the use of conditional promises.

Conditional promises promise no outcome until their dependent precondition can be promised, and therefore they form a locally partial order. By contrast, unconditional promises are 'non-blocking', non-ordered, or potentially concurrent. The process steps that accomplish the outcome to keep the promise may be called *interior time*, and the proper *exterior time* of the process only advances by a tick once the block has been lifted and the dependent outcome is delivered.

In a sense, a promise of an advanced convergent state is a 'blocking algorithm'. The process exits when the final state has been reached. In this case it is not waiting for input, as in blocking I/O, but rather for the keeping of an interior promise. It doesn't refer to any particular message, because it acts as a quasi-invariant condition. As long as the

condition is not met, nothing with proceed. If the process drifts in and out of compliance, due to other subtime processes, then blocking may add exterior latency.

5.14.1 SHARED TIME PROCESSES

It only makes sense to speak of blocking and non-blocking in a shared time environment, i.e. an agent that has interactions with more than one dependency. Since each agent has its own process clock, the agent that shares communications with these has to share its own clock with all its dependencies—violating the 'shared nothing' notion. Analogous to reaching consensus equilibrium, any agent with multiple dependencies does have to wait for all of them to keep their promises, else it cannot keep its own promise. To summarize: an agent with multiple dependencies need not wait for dependencies in any order (as they are symmetrical with respect to the current promise, at scale n), but it must wait for all of them in total, at scale $n + 1$ (see figure 5.25).

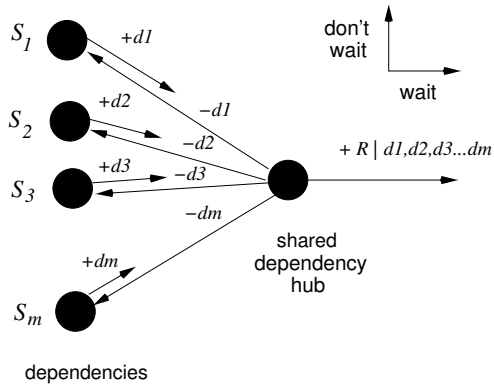


Figure 5.25: Any active $N : M$ relationship will result in partitioning of interior time at a hub. The granularity of that partitioning is an interior policy decision for the agent. Coroutines, threads, and serial blocking are three common variants of scheduling strategy for sharing the agent's state machinery between the neighbouring agents.

Example 104 (Reactive Manifesto)**.** *The Reactive Manifesto[BFKT] for software design principles calls asynchronous message passing and non-blocking processes. This is an unclear statement of intent, because it refers to non-invariant concepts. Suppose we decompose an agent into subagents, i.e. partition them as non-ordered dependencies of larger scale time-ordered dependencies (see figure 5.25). In a single process, there must be a hub that receives the results of such concurrent partitions ('threads'), which*

must block on a larger scale to aggregate the results. A process (super)agent cannot be non-blocking on a scale of exterior actions—at its exterior scale—without breaking such a conditional promise. That would alter its causal behaviour. However, the promise to depend on m mutually independent dependencies should not imply an arbitrarily imposed order; independent concurrent processes need not be starved of a shared time resource because of the need to wait for a subset of them.

A program need not suspend parallel threads or co-routines while one of them is waiting for a result. Waiting is a serial property, not a parallel one. The use of synchronous or asynchronous about communication implicitly uses the clock of one process to measure the progress of another, which has no invariant meaning.

The reason for this nitpicking is that it may mislead readers into thinking that i) waiting is never necessary, and ii) all processes can be made more responsive by parallelization. Not suspending parallel threads does not make a serial dependent thread asynchronous as measured according to its proper time. It may or may not be perceived that way according to some exterior clock time.

These points are certainly pedantic, but we need clarity as an antidote to 'best practices' that are merely advocated on the basis of unclear language without explanation.

Convergence has no dependence on past state, as long as the final outcome (desired end state) is a system invariant over each extended epoch of the system. It is an effective counter-strategy to the risk of non-linear divergence. For state that gets reused on multiple occasions, it is a strategy for maintenance that builds *intrinsic stability* into systems[Bur4 a].

State convergence is also the way we render alternatives *indistinguishable* (by forgetting incidental past), and therefore engineer greater stability through the fault tolerance. The instinct to throw away the may run counter to what many software developers are trained to do—i.e. to keep every distinct case separate in its own context, but it actually leads to greater certainty in the future. I'll go out on a limb and predict that we need a greater focus on advanced causation for sustainable and scalable computing in the future.

5.14.2 ENTANGLEMENT: SYNCHRONOUS AND ASYNCHRONOUS SIGNALS

Synchronous means literally simultaneous—at the same time, i.e. measured within the same clock tick interval. In a distributed world of many clocks this is a meaningless aspiration[Lam78]. It can only mean 'observed in the same interval'. Time intervals are not invariant (they are covariant, i.e. the change with scale and observer reference frame). Asynchronous implies that an agent may wait for an unspecified interval after receiving a signal before completing its dependent promise.

Both of these pertain to conditional promises:

$$A \xrightarrow{Y} S \tag{5.186}$$
$$S \xrightarrow{-Y} A \tag{5.187}$$
$$S \xrightarrow{X|Y} R \tag{5.188}$$
$$R \xrightarrow{-X} S \tag{5.189}$$
$$\tag{5.190}$$

Since we can only measure time differences locally at a single agent's clock, a synchronous response would imply that the difference in S's clock time between keeping promises (5.187) and (5.188) was minimized.

An asynchronously-kept conditional promise would imply an arbitrary delay between keeping promises (5.187) and (5.188). Thus synchronicity is a policy decision to set a scale for a 'timeout'. The semantics of faults also need to be considered in these promises: a 'fault' may also interpreted as a promise outcome in this loose description.

The definition is more complicated when there is a shared channel (figure 5.25).

$$S_1 \xrightarrow{Y_1} R \tag{5.191}$$
$$S_2 \xrightarrow{Y_2} R \tag{5.192}$$
$$\ldots \tag{5.193}$$
$$S_m \xrightarrow{Y_m} R \tag{5.194}$$
$$R \xrightarrow{-Y} S_1, S_2, \ldots, S_m \tag{5.195}$$
$$R \xrightarrow{X|Y_1,Y_2,\ldots,Y_m} A \tag{5.196}$$
$$A \xrightarrow{-X} R \tag{5.197}$$
$$\tag{5.198}$$

Event driven systems may be synchronous or asynchronous. This is a timescale issue.

Lemma 25 (Synchronous or asynchronous events)**.** *A conditional promise only requires the arrival of an event, but does not specify a labelled proper time interval for a response to be given. The latter is limited by other dependencies, e.g. CPU rate, memory speed, scheduling commitments.*

Usually developers think of this from their own current perspective: their viewpoint is that of an exterior observer (like a monitoring system, not drawn in the figure), which (by some magic) has instantaneous knowledge of the states of the agents. The promise (5.186) occurred when the final result was 'in the bank' R according to its own clock. This clock is not usually distinguished from the clocks of the other agents, thus effectively assuming a single global Newtonian view of time. Alas, in order to observe those agents

directly, the same promise relationships in figure (5.196) are needed. Whether these are given synchronously or asynchronously is scale dependent—a matter of definition, not an observable fact, because it relies entirely on the definitions of the final observer after it has sampled arriving signals. This is indeterminate, because to know this promise would be to ignore or violate the autonomy of the agents.

The implicit goal of the manifestos seems to be to render total systems as close to causally deterministic as possible—recreating the Newtonian view of global past. This is possible, but only at the expense of a rate of total interior time the gets slower by at least N^2 for N interior agents—as we know from consensus systems.

Philosophically, the idea of entanglement opens for a discussion of many deep ideas about relativity and mutual knowledge, and how the concept of 'consensus' can even make sense across a spatial region, limited by the propagation of packets. These questions are familiar from modern physics, but their information theoretical counterparts are only just being appreciated.

5.14.3 ENCAPSULATION OF EXTERIOR MESSAGES

How we 'quantize', or define the atomicity of outcomes, by limiting their observability, frames the way in which we interpret the units of transfer in message delivery. Entanglement (or irreducibility) of communication allows us to convert non-deterministic asynchronous message channels into effectively deterministic synchronous message channels, by restricting or 'quantizing' observability. This is a sleight of hand, based on the voluntary cooperation of the agents involved, but it can be effective.

The property of entanglement has the consequence that, once a message enters a link, it must either leave it as an indivisible unit, or have no effect whatsoever, and thus each transferrable unit must be wholly containable in the send and receive registers. No observer could see a partial state. Everything entering becomes a state of the collective superagent. In the language of distributed consensus, we can turn these promises into commitments, kept deterministically for single hops, and then build on the increased certainty to work towards larger scale consensus[Lam01, GL06, OO14, BB14a].

We packetize messages, as is normal to keep transactions predictable, and thus at least two layers are needed for packetized message delivery model (see figure 5.26): a network interface and link layer operates on the level of packets, and an 'intended message' layer, for aggregating packets into larger messages, operates between the cells. The cells are the intentional agents, originating and consuming messages as part of their larger plan. Network interface agents have intent only by proxy.

Separate entanglements may be established at each level of information promises: it is the entanglement of the network interfaces that allows us to make strong statements

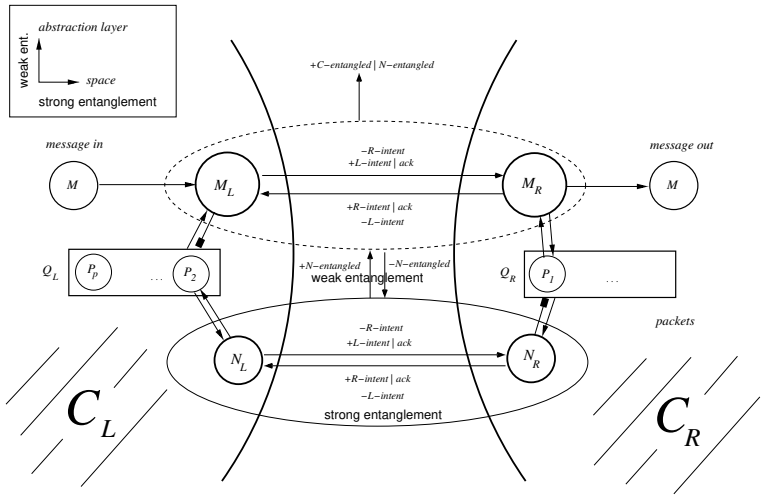


Figure 5.26: Entanglement is a bottom-up property, where higher level promises depend on the entanglement of the lower layers. The entanglement makes pairs of agents effectively into a single irreducible superagent, in the sense of [Bur15a]. By building on such dependencies, we can trade ad hoc homogeneity for large scale quasi-deterministic fabrics.

about transmission of packet chunks, and the entanglement of the cell queues or application buffers, containing the entire messages, which may synchronize complete ordered messages.

5.14.4 IRREDUCIBLE SUPERAGENT PICTURE

Co-dependent promises, made (and kept) by the endpoints, must be maintained regardless of what other independent promises cells might make to any other agent. This happens when both agents are driven by what happens between them rather than coordinating their independent activities (see figure 4.13). One can explore the use of this property in order to keep strong promises about message delivery[BBKK18]. Notice that these co-dependent promises are invariant under $L \leftrightarrow R$, and are thus timeless and without preferred orientation.

Agents that move in lock-step by co-dependence do not have independent clocks as far as the joint process is concerned. Each agent has its own clock, but the co-dependence means that they are both constrained by the process of message passing occurring between them. Thus, the two agents can only move as fast as the slowest agent in the entangled state, as its sampling rate limits the joint process. The single valued co-time of section

4.5.3 applies to this joint clock. This is how time scales for interior processes.

Example 105 (Once only delivery)**.** *The promise of once-only delivery cannot be trivially extended to multi-part multi-hop messages, in more complicated topologies, without some work. We must defer the full discussion for a sequel, and make only a few remarks here. It is* possible *for multiple copies of a packet to be observed, duplicated, and transmitted around a network, if agents fail to keep the necessary promises, no matter whether out of ignorance or malice. This is not specifically a weakness of our scheme: it is not easy to promise a negative result.*

Nor is it possible to prevent unexpected behaviours: since no agent can make a promise on behalf of another. There are two pragmatic ways to localize the responsibility for intended outcomes to the end points, away from intermediate interference:

- *One is to used shared secrets or encrypted messaging to make corruption by 'man in the middle' interference detectable. This need not be promised at all layers in a communication stack: high level encryption would suffice for detection by the intentional agents.*
- *Another way is for each packet to make a separate and uniquely labelled promise (see section 3.12 in [BB14a]) by promising a unique desired state. If each unique and intentionally different promise has its own label, and then duplication may be detected, assumed redundant, and ignored idempotently.*

Idempotence of promises means that a promise repeated n times is the same as the promise given once[71]*:*

$$\left(S \xrightarrow{+M} R\right)^n = S \xrightarrow{+M} R. \tag{5.199}$$

Idempotence must play a role in promising uniqueness, where we don't have complete control over causation.

5.15 SPACETIME INVOLVEMENT IN QUANTITATIVE SCALING

Networks are discrete, countable structures, but we often use real fractional measures to estimate their properties at scale, when counting transaction by transaction is impractical or undefined. Dimensionality then becomes essentially an average property of a network that may be used to approximate its quantitative characteristics, such as the flux of some quantity flowing through certain interfaces. To see this, we can look at some examples of how networks and dimensional spaces interact.

In a mean field model of systems, network links are counted using a continuum approximation in terms of volumes, and fractions of volumes. The details of agents and their promises are all averaged away. This approach has been used by [Bet13] to study the scaling in cities. It is an unfamiliar approach in engineering, but it plays an important role in deriving the scaling laws.

5.15.1 LINEAR SCALING OF OUTPUT FROM AGENTS

When a promised output scales in proportion to the number of agents, it's a sign that there is a direct causal linkage between the agents and the measure concerned. On a small scale, the chemistry of their interactions may be based on simple counting. Surely all outputs stem from agents, so how is this news? The point here is that the measure does not depend on assistance from other agents. Linear scaling refers to the effectively autonomous capabilities of the agent. The key must be that, if the agents have any dependencies, they do not play a role in limiting its output in any way. This may apply to promises that are requirements for survival.

Every agent more or less deterministically depends on some source infrastructure agent S, the number of promises is one to one:

$$\text{Number of consumed} = \sum_{i=1}^{N_I-1} \Pi_{iS}^{(\tau)} \simeq N_I - 1. \tag{5.200}$$

If the source is distributed over several agents, then this is still true:

$$\text{Number of consumed} = \sum_{s=1}^{S} \sum_{i=1}^{N_I-1} \Pi_{is}^{(\tau)} \simeq N_I - 1. \tag{5.201}$$

Thus, the number of promises needed to supply this demand is also proportional to N_I:

$$N_{\text{infra}} \equiv N_\tau \simeq N_I. \tag{5.202}$$

5.15.2 COMPARING CENTRALIZED AND DECENTRALIZED EFFICIENCY

Consider, as an example, the role of scaling in the time and energy usage of a central system (in a star configuration) versus a decentralized system (in a general mesh). We can consider two times: i) the response time for a client change to observe a change and consult a central server for a response, and ii) the equilibration time for all agents in the system to arrive at an agreed state.

Figure 5.27 shows the key component time scales, as measured by a single exterior observer:

- Δt_{sample} is the sampling interval for the client, where a change is observed.

- Δt_C is the process time internal to the client, at the 'edge' of the system, whether centralized or not.
- Δt_S is the process time internal to the server, at the heart of a centralized system. In a decentralized (peer to peer) architecture, the server role is performed by each client, so there is no distinction between Δt_C and Δt_S.
- Δt_{CS} is the transport time for data from client C to server S, or we can speak of $\Delta t_{CC'}$ between two clients.

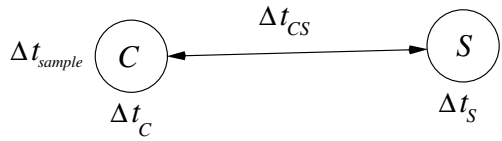


Figure 5.27: Timescales associated with system interaction.

.

The response times for different configurations are the total time for a change to be observed and for a centralized acknowledgment to be returned. Let's assume that a server can parallelize the handling of responses, so that there is no serial waiting:

- For a centralized system with a single client C and a single server, which can reply in parallel:

$$R_{\text{client-server}} = \Delta t_{\text{sample}} + \Delta t_C + 2\Delta t_{CS} + \Delta t_S. \tag{5.203}$$

- For a decentralized system with a single client C, no server communication is involved as the client does all the work:

$$R_{\text{autonomous}} = \Delta t_{\text{sample}} + \Delta t_C. \tag{5.204}$$

- For a centralized system with N clients C_i and a single server, which can reply in parallel:

$$R_{\text{central}} = \max_i \left(\Delta t_{\text{sample}} + \Delta t_{C_i} + 2\Delta t_{C_i S} + \Delta t_S(N)\right). \tag{5.205}$$

  Note that the server response time might depend on the number of clients in practice, even though there is some parallelism most systems will experience queueing.

- For a decentralized system with N clients C_i, no server communication is involved as the client does all the work:

$$R_{\text{decentral}} = \max_i \left(\Delta t_{\text{sample}} + \Delta t_{C_i}\right). \tag{5.206}$$

- Finally, for a decentralized system with N clients C_i, and peer communication,

$$R_{\text{p2p}} = \max_i \left(\Delta t_{\text{sample}} + \Delta t_{C_i} + 2\Delta t_{C_i C_j} + \Delta t_{C'}(N)\right), \text{ where } i \neq j \tag{5.207}$$

The response times depend on the dimensionless scaling ratios:

$$\frac{\Delta t_C}{\Delta t_S}, \frac{\Delta t_C}{\Delta t_{CS}}, \frac{\Delta t_S}{\Delta_{CS}}, \frac{\Delta t_{\text{sample}}}{\Delta t_C}, \frac{\Delta t_{\text{sample}}}{\Delta t_S}, \frac{\Delta t_{\text{sample}}}{\Delta t_{CS}}. \tag{5.208}$$

These are the critical ratios that play a role in changing an existing system. Clearly, there is a large number of independent timescales that govern the scaling of a distributed system, whether centralized or not. When the communication is peer to peer, in a mesh, there is an even greater number of potential timescales to consider if the agents are not homogeneous.

The equilibration or relaxation

time Δt_{eq} for agreeing about some information (a consistency time) is somewhat simpler. It's trivially zero for autonomous agents.

- For a central system:

$$\Delta t_{\text{eq}} = \max_i \left(\Delta t_{C_i S} + \Delta_S(N)\right) \tag{5.209}$$

- For a decentralized peer system:

$$\Delta t_{\text{eq}} = \max_i \left(\Delta t_{C_i C_j} + \Delta_C(N)\right) \tag{5.210}$$

The alteration of a single system involves fewer dimensionless ratios. If we want to compare different systems 1 and 2, the ratios can be simplified for the processes:

$$\frac{R_1}{R_2} \quad \text{and} \quad \frac{\Delta t_{\text{eq-1}}}{\Delta t_{\text{eq-2}}} \tag{5.211}$$

Note that, if the server cannot actually cope with parallel connections, then, in the worst case of a serial queue, the service time Δt_S will depend on the queue length, i.e. it will be the response time for a queue of length N.

The conventional argument for the cost of centralized and decentralized systems is that this can span from N for a central configuration to N^2 for a peer to peer mesh. In this case, for time, the processes are not cumulative since they run in parallel, each according to their own private clocks. So the time is not cumulative, but rather the maximum of racing parallel processes. The energy cost, on the other hand, is cumulative since power is a serial dependence, so the N to N^2 scaling will apply as the worst case bounds.

5.15.3 DERIVING METCALFE'S LAW FROM PROMISE NETWORKS

Metcalfe's law predicts on general dimensional arguments that the economics output of a network will be proportional to the square of the number of nodes in the network. We can examine this assertion in the context of Promise Theory. Promise theory predicts that links represent value in the following way. Consider then the sum of all impartial promise valuations by third party C. If we assume that all agents assess the value of interactions as strictly positive, then:

$$\text{Mean value} = \sum_{\tau} \sum_{i,j=1}^{N_I} v_C(\pi_{ij}^{\tau}) \leq c\langle \alpha_i \alpha_j \rangle N_I(N_I - 1) \tag{5.212}$$

where $N_I = \max_{\tau}(\dim(\tau))$ (see appendix B.2 about promise valuations).

Note that, in spite of the quadratic appearance from the result, this is a linear sum, so it acts automatically as a linear averaging measure. Also, for any given specialization τ, the filling fraction of the promise network is likely low; thus, a key assumption is that, when properly documented, agent's specialized promises in fact depend on many others *conditionally*, forming a wide reaching network of progressively weak coupling. Conditional promises propagate the range of value interaction[Bur16a, BB14a]. This is the ecosystem effect.

The weakness of coupling is not a problem provided the city is reasonably homogeneous in density. If we define an effective density for the network, which describes some probable average level $\rho \in [0, 1]$ of 'intercourse' between agents (any kind of sustained relationship), then it is fair to write the value of a network of promises:

$$\text{Mean value} = \sum_{\tau} \sum_{i,j=1}^{N_I} v_C(\pi_{ij}^{\tau}) = c\,\rho\, N_I(N_I - 1). \tag{5.213}$$

provided the total density of promises forms an SCC of order N_I members. This value can be distorted from the quadratic form by significant inhomogeneity. Now, for most cities, $N_I \gg 1$ and $\rho, \alpha_i \ll 1$, so for strictly positive value interactions:

$$v \simeq c\,\rho N_I^2. \tag{5.214}$$

This is Metcalfe's law. It depends on the assumption of strictly positive value (i.e. no non-profitable interactions), and sufficient density of promises to involve everyone in the city who belongs to the infrastructure. Why is this plausible, when most specialization leads to modularity? One reason is that modularity is only a separation of scales, not an elimination of dependency: dependencies form an ecosystem. Nearest neighbours might hold the greatest semantic importance to a given function, but this reductionist viewpoint is not independently sustained without the eigenstability of the entire web[BBCEM10].

5.15.4 ECONOMIES OF SCALE

Outputs and characteristics of systems, which depend somehow on the scale of a system, can be related to its size, or some other dimensionless scale. The dimensionful parameters are often linked together by a functional equation, like an equation of state in thermodynamics that describes the macrostate as as relationship between the involved scales. The result for a single variable, that exhibits scale invariance is a relation of the form:

$$V(N) = V_0 \; N^{\beta}. \tag{5.215}$$

This explains how the variable V may be expected to scale as a function of the number of agents, etc. This is scale invariant because

$$\frac{V(\alpha N)}{V(N)} = \frac{V_0 \; (\alpha N)^{\beta}.}{V_0 \; N^{\beta}} = \alpha^{\beta}. \tag{5.216}$$

The relative scaling is independent of the value of N. This is not the case in the Amdahl and Gunther laws (see sections 8.7.1 and 8.7.2). In other words, this kind of scaling is of a different nature: intrinsic (or scale free) rather than extrinsic.

The dimension of space cannot appear explicitly in this relation, due to the scale invariant form. However, it can appear in the vale for β. In fact, it can determine whether a system becomes squeezed or diluted as it scales: whether resources become burdened or plentiful. There are three cases for the scaling, as before (see figure 5.28):

- Sublinear scaling, or economies of scale $\propto N^{\beta<1}$. It costs proportionally less in terms of infrastructure investment to keep a system running. For example, biological organisms use only $N^{0.75}$ more energy as their size increases, so larger organisms are more efficient. On the other hand, they also get slower because it takes longer to transport energy and signals throughout their bodies.

- Linear consumption of resources $\propto N$ (food and energy consumption). Energy is conserved so there is no way around the this.

- Superlinear amplification of output $\propto N^{\beta<1}$ (economic output of cities, etc.). When there are 'smart' components that can specialize and collaborate, to exploit the recovery of generally poor efficiencies, size can actually result in a net gain. This happens in cities, for example.

Example 106. *Interesting studies have shown that cities (which are semantic/dynamic systems somewhat similar to IT systems) exhibit intriguing universal scaling, that is reminiscent of biology.*

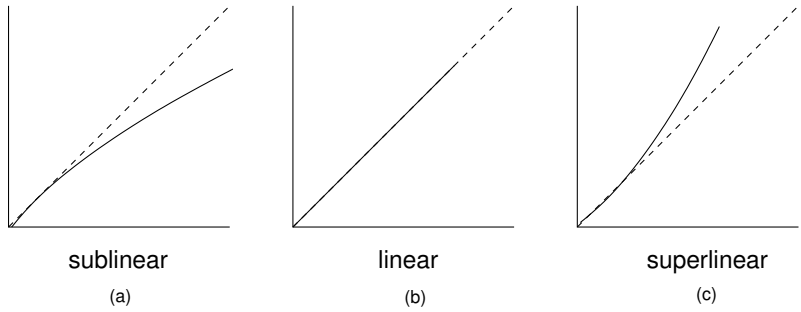


Figure 5.28: Sublinear, linear, and superlinear scaling of a value with respect to a control variable

.

5.15.5 EMBEDDING SPACE VOLUME AS ESTIMATOR

Suppose a system starts out one dimensional, with people queueing at a checkout. As the crowds get larger, the line breaks up into a two dimensional crowd and people have to interleave from all directions in two dimensions to pass through the checkout. This is an example of dimensional multiplexing. Although the agents are discrete, the simplest way to represent their structure is to imagine them as floating in a multi-dimensional space. Similarly, when water flows out of your bathtub, water from three dimensions surrounding the drain get multiplexed into an essentially one dimensional flow.

Normally, one might expect to count input and output by agent or by link. However, if the number of links converging at a single agent become so great that counting is impractical, there is no way to liken a process to a simple Poisson arrival queue, and we resort to flow counting based on density arguments. Let's now show that this is equivalent to volume of the infrastructure V_I in [Bet13]:

Consider a number of agents N_{infra} who provide infrastructure (gas stations, supermarket, etc) for a number of clients N_{client}.

$$\pi_{\text{infra}}: \quad A_{\text{infra}} \xrightarrow{+\text{infra}\#\mathcal{V}} \{A_{\text{client}}\} \tag{5.217}$$

$$\{A_{\text{client}}\} \xrightarrow{-\text{infra}} A_{\text{infra}}. \tag{5.218}$$

Suppose that each infrastructure agent A_{infra} can promise to service $\mathcal{V}$ clients simultaneously; then, using a simple valency argument, we have a detailed balance equation for the interactions at steady state:

$$\alpha_+ N_{\text{infra}} \mathcal{V} \geq \alpha_- N_{\text{client}}. \tag{5.219}$$

Thus for simple counting of distinguishable agents, we may estimate the number of

infrastructure agents needed to support a number of clients:

$$N_{\text{infra}} \geq \underbrace{\left(\frac{\alpha_-}{\alpha_+}\right)\frac{1}{\mathcal{V}}}_{\text{intrinsic}} \times N_{\text{client}}. \tag{5.220}$$

where α_-/α_+ may be interpreted as the affinity for the service, or the reciprocal compressibility. This scales linearly with the number of clients in the catchment area of the infrastructure. Moreover, there is no way, in this detailed formulation that we can count otherwise. The only economy of scale in this arrangement is the standard linear multiplexing result for the marshalling of $\mathcal{V}$ queues into a single queue with $\mathcal{V}$ servers, noted in section 8.7.2.

However, if we now ask how to count the number of clients that can be fed into a single infrastructure agent as a funnel, from a spatial volume, with traffic multiplexed equally from all dimensions, then the best estimate is to serialize the counting, as before:

$$N_{\text{client}} \quad = \quad \left(\frac{V_{\text{catchment}}}{N_{\text{users}}}\right)^{\frac{1}{D}} C(D) \times N_{\text{users}}, \tag{5.221}$$

where we imagine a catchment volume $V_{\text{catchment}}$, containing any number of agents N_{users} who are interested in the infrastructure service, and we serialize them along a tube of constant cross section $C(D)$ (see figure 5.29). Although these numbers only apply to a

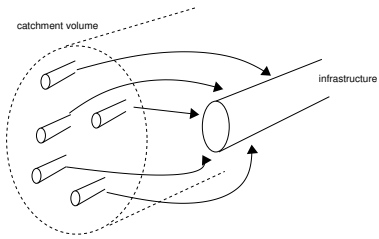


Figure 5.29: How spacetime involvement compresses serialized agent links into an effective flow of fixed cross section.

small mesoscopic volume of space, in a homogenous city, this will apply to the entire city, so we are justified in taking

$$N_{\text{use}} \simeq N_I, \tag{5.222}$$

which, combined with an equation of state for the volume, reproduces the earlier result

$$N_{\text{infra}} \simeq V^{1/D} N^{1-1/D}. \tag{5.223}$$

In this argument, it is clear that only the active agents N_I play a role in the counting, and process flow, hence this also justifies why we can assume $N_I \to N$ in [Bet13].

5.15.6 EUCLIDEAN APPROXIMATIONS TO A NETWORK

The minimum size of the infrastructure network can be estimated by squeezing the total sparse volume into a narrow, approximately one dimensional pipeline, with a small cross section. This is only plausible of the network utilization is really sparse, since then the total interaction can be compressed into the lower dimensional network, by multiplexing. The average distance between agents inside the system (in D dimensions) is

$$d = \left(\frac{V}{N}\right)^{\frac{1}{D}} . \tag{5.224}$$

An embedded infrastructure network has structure that pervades space, because it is embedded in a real world volume, and needs to reach the homogeneously distributed agents within. This 'space filling', or fractal quality, was important in deriving the biological scaling laws[Wes99].

Example 107 (Scaling of city economics)**.** *For cities, this space filling is more like an embedding than a fractal thickness to paths; however, following [Bet13], we may assume that the network fills space to some level, so that the interactions around it fill out an effective (Hausdorff) dimension $H < D$, and we may write the order of magnitude estimate for the infrastructure volume:*

$$V_I \geq g_I \left(\frac{V}{N}\right)^{\frac{H}{D}} L^{D-H} N, \tag{5.225}$$

where $g_I < 1$ is a dimensionless constant that indicates the fraction of nodes in N spanned by the particular infrastructure being considered. L is some fixed scale with the dimensions of length $[L]$, so that

$$[V] = [L]^D. \tag{5.226}$$

and $L^D \ll V_I \ll V$. In other words, the volume is the effective average linear volume swept out by a fixed cross section $L^{\frac{D_H}{D}}$, as it feeds into the N nodes connected by the infrastructure[72]. This has the schematic form of N queues that are serialized paths of dimension $V^{1/D}$. For $H = 0$ the nodes are unconnected, for $H = 1$ roads are serial or linear, and for $D > H > 1$, the roads or channels have an effective fractal 'thickness', from a coarse-grain perspective (see fig. 5.30). It turns out that we only need to look at $H = 1$, as it is serialization rather than physical dimension that is important.

For cities, the picture is been visualized in terms of physical channels, roads, cables, and transportation cost etc; however, any serial stream of work could constitute a virtual path for the infrastructure, coming from a number of agents within the volume V. The $V^{1/D}$ scaling represents a serialization of the work from across the homogeneous region.

So, we'll show later that this also applies to services provided from spot locations, without considering the communication channel at all. Transport need not be the cost of the work, but the argument still applies to the serialization of tasks in as a queue. This means

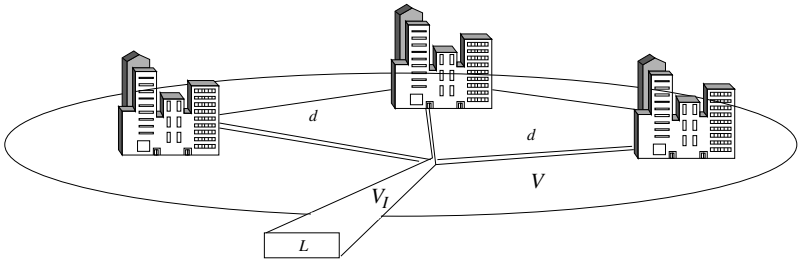


Figure 5.30: The volume of the infrastructure sparse network is negligible compared to the ball of the city.

we can write the infrastructure volume as approximately (rewriting (5.225)):

$$V_I \geq g_I V^{\frac{H}{D}} L^{D-H}\ N^{1-\frac{H}{D}}, \tag{5.227}$$

representing N_I serial queues of supporting services, being fed from a D dimensional region.

Example 108 (Relevance to Gunther's Universal Scaling)**.** *A rational queueing expression, in Gunther's Universal Scaling Law (8.11), can never explain fractional scaling exponents seen in cities, but it can demonstrate some projected superlinearity with $\alpha < 0$. To feed superlinearity, we need something more than parallel serial processes where the work is done by N point sources. Only if the work is done by N^2 interactions can a partial efficiency make the exponent greater than unity. To get this, we need to feed higher dimensional volumes into lower dimensional volumes. This is what is going on in the mean field theory of [Bet13].*

$$\textit{Output} = \textit{Maximum output}(N^2) \times \textit{Fraction infrastructure used}(N) \tag{5.228}$$

From a graph theoretical perspective, this is a change in average connectivity of the infrastructure network (i.e. the average degree of nodes k[Bur04a]). If the fraction is a fraction of a volume rather than a line or a number, there are dimensional exponents involved, which represent the contact efficiency by close packing the city population. With more dimensions, a larger surface area can be used for interaction. If we assume that the infrastructure network is sufficiently dense that it reaches almost everyone, then

this continuum approximation is reasonable.

$$\textit{Density of infrastructure users} = \frac{N_I}{V_I} = n_I \; N^{\delta}(D). \tag{5.229}$$

where $\delta(D) = 1/D(D+1)$*, for* $H = 1$*. Recalling that this volume is really a continuum approximation of a network of nodes, this translates into an average node degree utilization (or locally used connectivity) within the infrastructure channel*[73]

$$k(N) = \frac{N_I}{V_I} = n_I \; N^{\delta}(D). \tag{5.230}$$

Assuming the infrastructure is pervasive so $N \simeq N_I$*, the equivalent serialized infrastructure volume, for a single process, is something like:*

$$V_I = \left(\frac{V}{N}\right)^{\delta} \times N_I \simeq N^{1-\delta} \times \textit{cross section}, \tag{5.231}$$

$$= \textit{capture volume per agent} \times \textit{span of agents} \times \textit{cross section}, \tag{5.232}$$

Using this volume, instead of the total volume of the system (city, community, etc), recognizes two things. First that cost of the infrastructure is much less than that of the entire system; and, second, that it is the serialization of the sparse resource over a standard cross section that we want to use for comparable work output. This is like fitting the sparse output volume of the city into an idealized serial stream of fixed width to see how its length scales with the number of inhabitants[74]*. The efficiency comes from being able to use more of an unexpected fixed cost, sparsely utilized resource along with other economies of scale. The net result is an amplification of the output by* δ*:*

$$\textit{Interaction related output } Y = \frac{\textit{const}}{V_I} N_I^2 \simeq Y_0 N^{1+\delta(D)}. \tag{5.233}$$

The numerator is unexpectedly constant, but the infrastructure volume scales sublinearly, the net output appears superlinear, with these assumptions. The question is how do we know if these are the same assumptions as the used for the measurements?

Modern datacentres and networks at scale have multiple redundant paths that make their interconnection networks space filling (e.g. Clos structures[Bur13a], see figure 5.6). This brings higher dimensional scaling issues into the picture. The general picture is one of close packing of utilization. When dependencies scale more favorably than the contended processes that rely on them (relatively speaking), each process gets a larger share of the shared resource, and is accelerated for a while, provided the total utilization remains low.

5.15.7 CONDITIONAL DEPENDENCY AND OUTPUT SCALING

We can note briefly why certain occupations in the city scale differently. In a specialization society, singular individuals or agencies rarely have all the prerequisites to complete

their work. They need to collaborate and depend on others. Thus other agents are effective infrastructure relative to them. It is the accessibility of this dependency that throttles output. To drive the long range cohesion of the whole community network, specialists come to depend on specialized services (e.g. patents depend on lawyers). This leads to a number of promise configurations.

Consider four cases:

1. **Interaction scaling**: as proposed in [Bet13], for interactive value creation.

$$A_{\text{Lab}} \xrightarrow{+\text{patent}} A_{\text{observer}} \tag{5.234}$$

$$A_{\text{Lab}} \xrightarrow{\pm\text{interact}} A_{\text{services}} \tag{5.235}$$

   Patent agencies are interacting at arbitrary range with a significant fraction the total promise graph, as a part of the ecosystem.

$$Y \simeq Y_0 \left(\frac{v}{V_I}\right) N^2 \to N^{1+\delta} \simeq N^{\frac{7}{6}} = N^{1.16}. \tag{5.236}$$

   The amplified value relies on the interplay between long range mixing, and short range isolation.

2. **Scarce agent scaling**: skilled specialist experts' output is proportional to the number of skilled agents, since their queue is sparse, and not filled by a wide volume of demand. However, the same economy of scale applies to their services when they are depended on, as 'infrastructure', by others.

$$Y \simeq Y_0 \left(\frac{v}{V_I}\right) N \to N^{\delta} \simeq N^{\frac{1}{6}} = N^{0.16}. \tag{5.237}$$

3. **Interaction promises with a scarce dependency**: such as in the case of a service that depends on a source of agents to fulfill a dependency. e.g. patents can only be produced by labs that depend on the outputs of specialized R&D employees and lawyers, working in private relationships, or in secrecy. The expression in (5.248) assumes a promise configuration like that of the assisted promise law[BB14a], with a main output based on a number of agents that provide input. The dependencies produce raw output, and the 'lab' agency collates and represents the collaborative mixing, e.g.

$$A_{\text{Lab}} \xrightarrow{+\text{patent}|\text{research,legal}} A_{\text{observer}} \tag{5.238}$$

$$A_{\text{Lab}} \xrightarrow{\pm\text{interact}} A_{\text{services}} \tag{5.239}$$

$$A_{\text{Lab}} \xrightarrow{-\text{research}} A_{\text{staff}} \tag{5.240}$$

$$A_{\text{Lab}} \xrightarrow{-\text{legal}} A_{\text{lawyer}} \tag{5.241}$$

$$A_{\text{staff}} \xrightarrow{+\text{research}} A_{\text{Lab}} \tag{5.242}$$

$$A_{\text{lawyer}} \xrightarrow{+\text{legal}} A_{\text{Lab}} \tag{5.243}$$

More generically, with two stages in the process of promise keeping, each experiencing scaling (see figure 5.31),

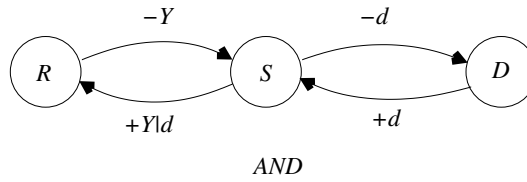


Figure 5.31: A two stage (long range) dependency has two economies of scale, when fed by a spacetime workflow. The probability of promises kept is multiplicative, like the logical 'AND' of the promises.

$$S \xrightarrow{+Y|d} R \tag{5.244}$$
$$R \xrightarrow{-Y} S \tag{5.245}$$
$$S \xrightarrow{-d} D \tag{5.246}$$
$$D \xrightarrow{+d} S \tag{5.247}$$

the total process picks up two 'economies of scale': the delivery of Y conditionally AND the delivery of the conditional dependence.

$$Y \simeq Y_0 \left(\frac{v}{V_I} \right) N^2 \times \left(\frac{v}{V_{\text{depend}}} \right) N \rightarrow N^{1+2\delta} \simeq N^{\frac{4}{3}} = N^{1.33}. \tag{5.248}$$

where $D = 2$ is used for the numerical values. These values accord better with the cited data in [BLH$^+$07], and tie in with the story about queueing.

What characterizes this interaction is the high level of specialization required to fulfill the dependencies. If the network is sparse, this is more difficult than if it is dense and diverse. This is the specialization gamble. With specialization comes individual efficiency, but also risk of instability by disconnection from key dependencies[Tai88].

They are a throttle on the process, because their absence could stop it altogether. Hence, we are justified in using the product 'AND' for combining the probably values in (5.248). This is not a hierarchical system interaction, because the services are not necessarily hidden from the long range dynamics by internal components of the superagent 'lab' (see figure 5.32 (b)). But in organizational theory, one normally assumes that all organizations are hierarchically organized (see figure 5.32 (a)).

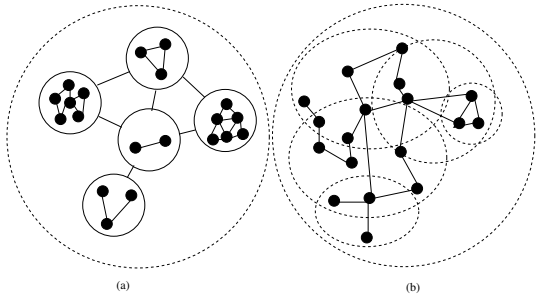


Figure 5.32: Structural recursion in an ecosystem is not like a branching process of containers (a), but rather the agents overlap with other regions of the same network to access their virtual functions. Thus their outputs are not concealed as interior substructure, but exposed as part of the flat internetwork (b). The result is that a second order recursion picks up a second economy of scale, in turn increasing the superlinearity of the derived output.

4. **Recursive promise dependency**. Let's consider what happens when the ecosystem network is based on a hierarchy of interaction ranges, i.e. promises are made recursively in fully protected shells. The agency produces a service using full community infrastructure, but also has some specialist dependency contained entirely within (see figure 5.32 (a)).

$$A_{\text{company}} \xrightarrow{+A_{\text{specialist}} \xrightarrow{+\text{solution}} A_{\text{client}}} A_{\text{client}} \tag{5.249}$$

would be viewed as a recursive operation on the infrastructure, and the economics of scale would apply to both times the (different) infrastructures were used.

$$Y_{\text{patent}} = \frac{N_I^2}{V_{\text{patent}}} \tag{5.250}$$

$$V_{\text{patent}} = g_{\text{R\&D}} \left(\frac{V_{\text{R\&D}}}{N_{\text{R\&D}}} \right)^{\frac{1}{D}} N_{\text{R\&D}} \tag{5.251}$$

$$V_{\text{R\&D}} = \left(\frac{V}{N_I} \right)^{\frac{1}{D}} \tag{5.252}$$

Substituting for $V_{\text{R\&D}}/N_{\text{R\&D}}$ from the last expression into the former,

$$V_{\text{patent}} = g_{\text{R\&D}} \left(\frac{V}{N_I} \right)^{\frac{1}{D^2}} N_{\text{R\&D}} \tag{5.253}$$

Inserting this into the output expression

$$Y_{\text{patent}} \simeq N^{1+\frac{1}{D^2}-\frac{1}{D(D+1)}} \simeq N^{\frac{13}{12}} \simeq N^{1.08}. \tag{5.254}$$

This value is almost linear, which is what we might expect on a self-contained specialization, since the outside world would not be able to tell the difference between a single agent and a single superagent.

The value is also smaller than case (1) above, not larger, so short range hierarchical scaling cannot explain the anomalously large exponents measured in cities. On the other hand, there are some smaller exponents in this range. It is interesting to examine these measures from the perspective of the promises represented, and their range in the embedding space.

There is a simple prediction here: long range dependency seems to increase output superlinearity, i.e. dependency brings strong long range coupling and activates a larger amount of the N^2 mesh. A similar effect can be obtained in [Bet13], by slightly increasing the Hausdorff dimension of the infrastructure $H > 1$. This does would correspond to a more pervasive generic infrastructure network, which is an opaque explanation at best. It seems unclear how to justify it.

Short range dependency is basically invisible at larger scales. This observation might help to explain superlinear seen in technological contexts too, through coordination[GPT15], but we have to be careful not to mix together effects that come about due to higher dimensionality, with other mechanisms for increasing the utilization of dependent resources.

5.16 GEOMETRY AND TOPOLOGY OF SYSTEMS

The ability to locate, discover, and match with other agents, to form promise bindings, depends on either physical or virtual mobility of the agents along network routes. Kinetically, agents may follow a random walk, as in ballistic discovery. A second possibility is that intermediaries perform the discovery by passing on information in a chain[75]. We can say that two agents are either

- Physically close.
- Virtually close.

Promise theory also predicts that two agents may be dynamically close or semantically close, such as when related meanings are grouped. The former depends on the length scales of the system (e.g. in a city, or in a datacentre) and its structure. The latter can be assumed approximately independent of these scales, because the carriers are very light, cheap, or fast (or, in the case of semantic distance, purely cognitive). If the cost of discovery can be neglected, the cost equation is different: collaboration can be cheaper, and the value of being in close proximity for a particular specialization is reduced[76].

Architectural assistance to optimize access to agents clearly plays a role. Directories, maps, and indices[Bur15a] are ways for agents to virtualize discovery of dependencies, and locate one another at low cost. Telephone directories map coordinate addresses to names. Yellow pages map coordinates to specializations. Similar specializations are grouped. Shopping malls and industrial estates are physical directories, where clients can expect to find services in a small volume. Directories may be discovered themselves, or formed by voluntary registration. The value of new bindings overcomes the tendency for similar specializations to repel one another: similar agents may be attracted implicitly (covalently) by the intermediate attraction to clients.

The scaling estimates of the city are based on infrastructure where physical motion of the population is based on the cost of traversing some fraction of the length of the city. We can repeat the output calculation to neglect this cost, as is the case in services that do not require physical transport.

- **Physical interaction** (transport/mobility): people move around using transport infrastructure to experience their environment. There is a promise for people to observe their surroundings, for something related to subject τ, and this promise is kept fractionally $\alpha_\tau \in [0, 1]$ during their walk.

  Let the linear range of the agent A_i be some dimensionless fraction per unit time rT_{explore} of the size of the city $V^{1/D}$, where r is the speed in units of city size[77]. If the density of impulses per unit length of city region $\mathcal{I}$ is assumed constant relative to the transport rate (because this is the basis of commerce, i.e. what the city is trying to optimize for people's finite time), then the number of impulses I_τ of type τ, experienced on such a walk, may be written:

  $$I_\tau \propto r_i \, T_{\text{explore}} \, V^{1/D} \, \mathcal{I} \, \alpha_\tau \tag{5.255}$$

  where α_τ is the probability that the person or agent will be receptive to impulses in its environment that are relevant to promises of type τ.

  Although there is room for inhomogeneous variations in the city regions, in the transport rate r, and the density of offerings $\mathcal{I}$, this will not change the average scaling argument much, as long as N is large. I make the assumption here that the density of experiences $\mathcal{I}$ is constant, even though the density of people is related to the city size. This is because the size of a city is constrained by the time rather than the distance (and we are suppressing explicit time).

  The range will be some fraction of the size of the city, available by transport infrastructure $V^{1/D}$. The cost of physically fishing for ideas thus takes the form

  $$C \simeq c_Y N_I V^{\frac{1}{D}}, \tag{5.256}$$

in agreement with the work model of [Bet13]. This applies for physical city interactions, and leads to the same output scaling expression in [Bet13].

$$Y_Y^+ \simeq N^{\frac{2D+1}{D(1+D)}} N_I^{\frac{D^2-D}{D(D+1)}}, \tag{5.257}$$

$$\simeq N_I^{\frac{7}{6}} \left(1 + \frac{N_0}{N_I}\right)^{\frac{5}{6}}. \tag{5.258}$$

- **Virtual interaction** (teleport/messaging): people are immobile and send messages to one another, watch entertainment, browse, read, talk, etc. These activities occupy an increasing amount of the time spent by people, not least because it can easily be interleaved with work time. The rate is no longer related to the size of the city, nor is there any obvious boundary to what can be discovered online (since the range of the Internet is even more diverse than a city)[78]. In this case, the impulses are more likely to be related to availability of the fountain itself (e.g. 'bandwidth' B) multiplied the time spent.

$$I_\tau \propto B\, T_{\text{explore}}\, \mathcal{I}\, \alpha_\tau \tag{5.259}$$

Discovery of information is the main issue. Before search engines, there were only directories such as white pages (by person) and yellow pages (by promise type). However delivery of what is discovered might still involve spatial constraints, e.g. locating a new car online does not allow it to be teleported to the buyer's location. However, 3d printing technology might change this, for a class of problems, soon.

Here it is not the locations that matter, but the rate at which impulses are absorbed. Once again, this is constant. When friends, books, or movies are communicating ideas to us, this happens at a rate that depends only on how quickly we can get hold of a stream. How users discover locations online, or by telephone is a separate question. Directories[Bur15a], advertisements, and chance all play a role here. The cost of fishing for ideas is thus now independent of the city size. For a community of multiple superagents, the analogous expression is:

$$C \simeq cN_I BT_{\text{explore}}. \tag{5.260}$$

If we imagine a community with no other infrastructure except its telecommunications network, and substitute (5.260) into the detailed balance equation:

$$g_Y \left(\frac{v_Y}{V}\right) N_I^2 \geq cN_I BT_{\text{explore}}. \tag{5.261}$$

Following through the calculation for the yield estimate identically, we find the scaling is no longer superlinear ($D = 2, H = 1$)

$$Y \simeq N^{\frac{H}{D}} N_I^{\left(1-\frac{H}{D}\right)} \simeq N_I^{\frac{1}{2}} \left(1 + \frac{N_0}{N_I}\right)^{-\frac{1}{2}}. \tag{5.262}$$

This simple result reflects the intuition that, if we neglect the 'universal cost' of telecommunications from the community accounting, then the value generated as a result of collaborative processes is proportional only to the fraction of participants who span the diameter of the city or community. This reproduces the well-known result for mobile ad hoc networking (MANET)[BC04, BC03].

The rate of output based on trawling of ideas and gestation in closed workgroups will be

$$I_\tau^{\rm work} = N_D \left(c_{\rm phys} I_\tau^{\rm phys} + c_{\rm virt} I_\tau^{\rm virt} \right) \tag{5.263}$$

and for the entire city of N_W workplaces:

$$I^{\rm city} = \sum_\tau I_\tau^{\rm work} \simeq N_W \bar{I}^{\rm work}. \tag{5.264}$$

5.17 SPECIALIZATION AND MODULARITY UNDER THE SPOTLIGHT

Semantics are usually scale dependent, but quantitative dynamics without strong semantics can be scale-free, e.g. output amplification. When we divide a network based on function, rather than based on timescales, the timescales associated with the different functions may lead to much weaker bindings between the modules, because unequal waiting is involved to compose the modules into a system. Redundancy can be used to mitigate the waiting and the associated fragility—but adding in redundancy tends to couple the components more tightly again, so some of the perceived benefits of modularity may be removed by having redundancy.

Some correspondences in the table below show how agents at one scale map into the role of agents at another scale, with a similar functional relationship, illustrating scale-free behaviours, provided we ignore detailed semantics. With physical or virtual scale increasing left to right:

SOFTWARE	CLOUD	CITIES
MODULES	CLUSTERS	DISTRICTS
PROCESSES	DEPLOYMENTS	WORK
APIS	APIS	OFFERS
FUNCTIONS	SERVICES	ORGANIZATIONS
USERS	TASKS	CITIZENS
PEOPLE	PEOPLE	PEOPLE
MECHANISMS	MECHANISMS	MECHANISMS

In software of all kinds, plugin modules answer one of the questions about 'software bloat': modules allow size to be limited by placing a natural boundary around a separable or independent process. Separation cannot deal with the complexity issue however, since the modules still need to be connected in the same pattern to achieve the same process functionality.

Dependency of one module on another is a source of fragility in any network. Conversely, increased efficiency in dependencies can lead economies of scale, and superlinear output increases (see section 8.7). Dependency amplifies the effects of modular scaling, for better and for worse. Redundancy goes hand in hand with dependency to follow the downstream principle (see 2.4.3).

Modularity does not necessarily ease maintenance, contrary to conventional software rhetoric, but it may break up the maintenance issues along semantic partitions, which helps to scale the expertise and investment of time needed for learning. Conversely, it can lead to narrowing specializations, with attendant lack of systemic understanding and experience. Modularity of repair is only useful if the different modules are supported by independent maintainers. Pulling a system apart to be maintained by a single person is meaningless as there is no scaling efficiency introduced. Modularity may be more expensive than monoculture unless the utilization of the modules is high. Multiplexing of resources is the mechanism that leads to efficiencies of scale (see section 8.7).

Example 109 (Modular design)**.** *Modular redesign is sometimes used as an argument for pre-planning a scaled up workforce, assuming that the workload will grow in the future. It is a gamble, because it relies on the idea that efficiencies of scale come from specialization. If the workload doesn't increase, then the cost of modularization redesign, combined with the extra overhead associated with the management of modular dependences just leads to higher costs, while the necessary extra workforce mainly sits idle.*

Example 110 (Modular IT services)**.** *The same argument applied to the use of separate machines and containers for encapsulating IT services. the 'one service per machine' idea argued that this made management easier. It also increased the workload and the capital cost of the machinery. If the individual services were not all running with high utilization, then the independent machines were mostly idle.*

We return to look at modularity in section 7.5.

CHAPTER 6

INTERACTIONS AND INFLUENCE

A collaboration between agents is what we mean by a *system*. This cannot happen without basic interation. We can't derive or prove the existence of interactions—interactions are a semantic primitive. We merely assume it, because without it there would be nothing. Interaction is an exchange of information. The representation of information is viewed quite differently in different fields: we might call the information a particle, like a photon or a gluon, or we might view it as a data packet, a molecule, a signal, and so on. What we consider to be material—as opposed to merely a signal—is simply a matter of convention and has no practical distinction. In this chapter, I want to lay out the basic approach to describing interactions.

Interactions between individual agents are what lead to collective behaviours. In this chapter, we ask: can we characterize systems in sufficient detail to understand their basic intentional operations and define the meaning of system failure at the level of keeping promises? This is plausible using Promise Theory, because:

- It leads to a directed graph of intent, which allows many standard techniques for the analysis of outcomes.
- Its outcome-oriented approach encodes more information than a classical reliability component model can because it makes intent more explicit.
- It can take into account *context* on many scales, which permits a fine graining of causal pathways, based on semantics, as well as the discussion of the larger environment in which a system operates.

Agents are, as always, the atomic building blocks, from which a greater chemistry of cooperation and functional design emerges. We may use agents to model all kinds of

components, whether 'dumb' or 'smart', human or machine. They are distinguished only by the promises they make.

6.1 DISTINGUISHABILITY, AGENT TYPES, AND LABELS

We may benefit from differentiating *types* of agents, when applying promise theory to functional spaces. A priori, agents have no type: they are homogeneous, structureless, universal elements (analogous to biological 'stem cells') that may only be differentiated via the promises they make. A *type* may thus be defined either by identification of a role[BB14a], or by explicitly making a scalar promise.

Consider a universal (typeless) agent $A_\emptyset$, that makes no initial promises, or empty promises to everyone:

$$A_\emptyset \xrightarrow{\emptyset} *. \tag{6.1}$$

We may add a scalar promise, with additional scope σ:

$$A_\emptyset \xrightarrow[\sigma]{+\mathrm{bla}} * \tag{6.2}$$

Any agent in the scope $\{*, \sigma\}$ may now identify the former agent as being equivalent to an agent of type b, making no promise. In other words, within the scope, the agent effectively has a new name:

$$A_{\mathrm{bla}} \xrightarrow[\sigma]{+\emptyset} *, \tag{6.3}$$

thence

$$\mathrm{bla} \xrightarrow[\sigma]{+\emptyset} *. \tag{6.4}$$

i.e., we can drop the promiser's agent symbol 'A' and label agents simply by their names. I'll use this notion from here on to write agent types implicitly, e.g.

$$T_1 \xrightarrow{\emptyset} A \quad \equiv \quad A_1 \xrightarrow{+T} A \tag{6.5}$$

$$R_2 \xrightarrow{\emptyset} A \quad \equiv \quad A_2 \xrightarrow{+R} A \tag{6.6}$$

$$H_3 \xrightarrow{\emptyset} A \quad \equiv \quad A_3 \xrightarrow{+H} A \tag{6.7}$$

and so on. In this way, we can move scalar labels from the promise body to the agent's identifier at will. This is in keeping with the idea that the agent's name is the basic promise that identifies it.

6.2 PROMISE VALENCY AND SATURATION

To develop Promise Theory as a formal chemistry of intent, we need to clarify how many agents a promise can support or service. In other words, how many 'slots', 'binding sites' or occupyable resources does an agent have, to share between promisees?

Declaring a finite number of such slots, explicitly allows for a simple discussion of resource exclusivity around promises. The concept is basically analogous to the valences (oxidation numbers) of electrons in physical chemistry. Think also of the binding sites for receptors, viruses and major histocompatability proteins in biology.

Definition 116 (Valence of an agent promise, and overcommitting)**.** *A promise which provides $+b$ to a number of other agents may specify how many agents n for which the promise will be kept exclusively. The valency number of an exclusive promise is a positive integer n, written*

$$A \quad \xrightarrow{+b\#n} \quad \{A_1, \ldots A_p\}. \tag{6.8}$$

A promise body may be called over-promised (or over-committed) if $p > n$.

Example 111. *A reserved parking area promises 10 spaces, to 20 employees. The parking promise is over-committed, since it cannot keep all of its promises simultaneously.*

Over-promising is not a problem unless all of the promisees accept the promise, and promise to use it. Thus a separate concept of saturation arises by using up all of the valence slots:

Definition 117 (Use-promise saturation)**.** *Suppose we have*

$$A \quad \xrightarrow{+b\#n} \quad \{A_1, \ldots A_p\} \tag{6.9}$$

$$\{A_1, \ldots A_m\} \quad \xrightarrow{-b\#m} \quad A \tag{6.10}$$

is saturated if $m \geq n$.

It is useful to define a function whose value is the net valence of a particular type of promise body.

Definition 118 (Net valence of a promise graph and utilization)**.** *$\pm b$, for a collection of agents $\{A_i\}$:*

$$\begin{aligned} \mathbf{Valence}(b;\{A_i\}) &= \sum_i \mathbf{Valence}(b;A_i) && (6.11)\\ &= n-m && (6.12)\end{aligned}$$

Hence we may assign an integer value to the level of usage, or a rational fraction m/n for utilization of the resource. If this fraction exceeds unity, or the net valency is negative, the keeping of the promise effectively becomes a queue of length $|m-n|$, requiring the agent to multiplex its resources in time to keep its promise.

Example 112. *Consider the two agents A_1, A_2:*

$$A_1 \xrightarrow{+b\#2} A_2 \quad (6.13)$$

$$A_2 \xrightarrow{-b\#3} A_1 \quad (6.14)$$

A_1 offers two possible slots for its promise of $+b$, while A_2 requests three units of it, leaving a net deficit:

$$\mathbf{Valence}(b;A_1,A_2) = -1 \quad (6.15)$$

This notation allows us to simplify the discussion of occupancy and tenancy in later sections.

Example 113. *Consider the following promises made by a network switching device:*

$$\text{switch} \xrightarrow{+(10Gb)\#48} \text{client} \quad (6.16)$$

$$\text{client} \xrightarrow{+(1Gb)\#1} \text{switch.} \quad (6.17)$$

The switch makes 48 promises offering 10Gb 'bandwidth' to the clients. The client accepts one valency slot (leaving 47 more), and promises to consume only 1Gb of the maximum possible 10Gb.

6.3 THE LANGUAGES OF PROMISED INTERACTIONS

In order to even comprehend one another's promises, agents need a common language, with which to express body content. The problem of how agents come to develop a mutually acceptable language for information exchange between independent agents has been studied in connection with linguistics both of the traditional variety and in the biology of the genetic code[DEKM98]. It is not a simple problem, and I shall not try to address it here in full; however, there are some simple things we can say about it.

In all cases, what we derive from these studies is that, regardless of whether language is executed continuously or discretely[79], the possible intended meanings form a discrete alphabet of symbols, representing capabilities, intentions, and so on. Thus, semantics constrain promise bodies to a set of linguistic atoms (morphemes) which are discrete. In nature, we see this in everything from gene codons, to cells and organisms, to Chinese ideograms[80].

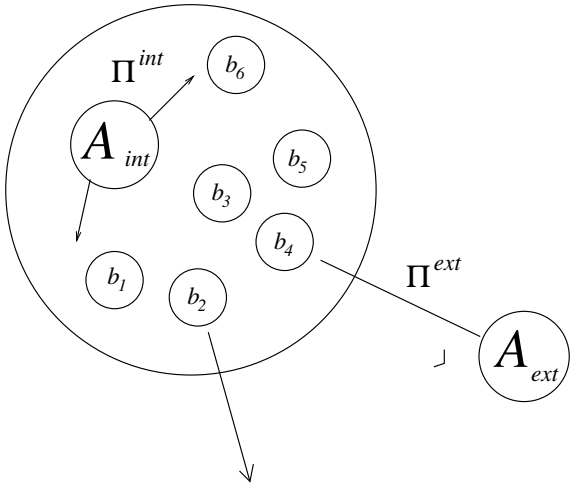


Figure 6.1: Body parts or linguistic atoms of intent may be used as a spanning set for the body of any promise in a region. This is like a coordinate system of intent. For b_4 to be promised to $A_{\rm ext}$, b_4 must be understood by both promiser and promisee agents.

The representation of promises in arbitrary natural language is unlikely to be simple, since the metaphoric basis of natural languages do not lend themselves to direct mapping. However, we can imagine formulating a set of restricted *Domain Specific Languages*, $\ell^{(\alpha)} = \{\beta_a^{(\alpha)}\}$, consisting of symbols $\beta_a^{(\alpha)}$, whose purpose is the represent specific intentional behaviours, and which map to a set of concepts that can be represented in a natural language $N^{(\alpha)}$ of all agencies in a coordinate patch:

$$\ell^{(\alpha)} \to N^{(\alpha)}. \tag{6.18}$$

Assumption 4 (Discreteness of promise body encodings)**.** *The language of promise bodies is assumed to be a discrete language pattern, of fixed, but unspecified, alphabet β,*

comprising basis symbols β_a.

$$b(A_i) = \sum_{a=1}^{C} c_a \beta_a, \tag{6.19}$$

for some coefficients c_a, *and* $C = \dim(\beta(A_i))$ *for agent* A_i.

Based on this decomposition of intent, we may now consider a local observer view on the measurement of agent structure, assuming finiteness of information. Consider the syntax space of alphabetic strings in a set of languages $\ell^{(\alpha)} \in \mathcal{L}$, where α labels different languages, each of which might have its own alphabet.

In order for agencies with different languages to be able to communicate intent, these languages must be mutually comprehensible, i.e. we must be able to map parts of them to one another. The existence of a discrete alphabet of symbolic words allows us to think of the alphabet of intentions as a matroid or spanning set β_a over a vector space. We may assume that the language of agent A_i has $\dim(\beta(A_i))$ dimensions or classes of intention, so that $a = 1, \ldots, \dim(\beta)$ (see figure 6.1). This alphabet may or may not necessarily be shared between all agents, but needs to be partially translatable for agents to make promises to one another.

Suppose we have two alphabets with symbols β and β'. In order for a promise body to be encodable in either language, we must have:

$$b = \sum_{a=1}^{\dim(\beta)} c_a \beta_a = \sum_{a'=1}^{\dim(\beta')} c'_a \beta'_a \tag{6.20}$$

i.e. both languages have to be able to span the promise body, or represent it in their own spanning sets or words. We need not require a single common spanning set of body parts to span every message, the language of agents does not have to be a global symmetry across spacetime, but we do require local continuity in the couplings, in order for information to be passed on.

6.3.1 TRANSMISSION OF INTENT

A related issue concerns the ability for an observational arbiter to distinguish between different agents. This depends on the ability of the agent to comprehend the language being promised. Suppose an observer suspects that an agent is non-atomic, i.e. it contains internal structure.

- Multiple agencies within a single agent might be identified if:
  - If an agent seems to make independent partial-promises to different promisees, it could be natural to formally resolve it into separate sub-agents for the independent promises (‘disaggregation’).

- A compound promise could be resolved into several simpler promises if:
  - The details of the promise body can be expressed as a number of independent items that can be made (+) or consumed (-) independently.
  - If, by emergent agreement, a set of primitives (like a table of elements) can be seen to form a spanning set for the promises made by an ensemble of one or more indistinguishable agents[81].

To know this information for sure, it would have to be promised. For this, we can define the concept of an agent *directory* (see section 5.8.10).

Definition 119 (Homogeneity of agent languages, and transmission of intent)**.** *Agents A_1 and A_2 may be said to have distinguishable promises that can be resolved by a receiver A_r iff the measures of the promises, which overlap with the receiver, are unequal. Suppose A_1, A_2 each make a promise to an observing receiver A_r:*

$$A_1 \xrightarrow{+b_1} A_r \tag{6.21}$$

$$A_2 \xrightarrow{+b_2} A_r \tag{6.22}$$

A_r might judge these two agents identical if $b_1 \cap b_2 = b_r \neq \emptyset$, i.e. if both promises contain a common part (the intersection $b_1 \cap b_2$), which the promisee promises to see:

$$A_r \xrightarrow{-(b_1 \cap b_2)} A_1, A_2. \tag{6.23}$$

In other words, if the receiver filters its perceptions according to what is common to all agents, then it is unable to distinguish them.

We can break this into two cases: if sources A_1 and A_2 share common components (e.g. share common genes), i.e. $b_1 \cap b_2 \neq \emptyset$, then the receiver can observe a similarity between the agents. If, further, the receiver only perceives the influence of what is common between them, i.e. it promises to accept $-b_r$, then the agents will perceive an elementary unit of promise equal to:

$$b_1 \cap b_2 \cap b_r \neq \emptyset. \tag{6.24}$$

Example 114. *In genetics, the body elements correspond to genes. A gene can be passed on $(+)$ from a parent to a child, but whether or not it is 'expressed' or activated depends on the proteins use the gene $(-)$ during morphogenesis. Thus, simply passing genes from generation to generation need not result in transmission of attributes (promises kept). Similarly, environmental conditions can play a role in activating or de-activating particular gene promises in different circumstances.*

Example 115. *For instance, imagine one agent believes it is promising to deliver a letter to a recipient. The agent receiving what the promiser considers a letter might, in fact, be promising to evidence in an investigation as a DNA sample on the letter. The rest of the letter vehicle has no semantic value. A second agent then promises to deliver a blood sample to the same recipient. This also qualifies as an evidential DNA sample to the recipient. Since the agent of DNA is encapsulated by both the letter and the blood sample:* $DNA \subset Letter$ *and* $DNA \subset Blood$*, an agent that can only measure DNA would see the letter and the blood sample as being equivalent sources of DNA.*

DNA, itself, is a vehicle (agent) for genes that are embedded within it. Exactly the same argument now applies at the level of DNA as a container. The presence or absence of an allele (gene flavour) within a strand of DNA indicates a similarity of intent.

The impact of a promise is defined through its binding strength, or effective coupling constant. In earlier work, I defined the notion of a trajectory for an agent, and the corresponding notion of a generalized force, obeying Newtonian semantics[BF07a, BF08]. Intuitively, one expects a force to be something that impacts an agent's trajectory

$$F : b \to b + \delta b \tag{6.25}$$

Though, readers should note that promise trajectories are rarely Newtonian. From this, one may construct a generalized force, which with the help of assessment function $\alpha(\pi)$ takes on a familiar form of a field-charge like interaction:

$$F \simeq \alpha \left(\underbrace{S \xrightarrow{+b} R}_{\text{Field}}, \underbrace{R \xrightarrow{-b} S}_{\text{Charge}} \right). \tag{6.26}$$

See [BF07a, BF08] for the details. This gives us a simple notion of a coupling strength by which to define such a measure of impact. The analogies to physics are attractive, but we should beware that the trajectories are 'rough walks' not smooth curves, in spite of the analogy to differential notation.

6.3.2 CONTINUITY OR SPATIAL HOMOGENEITY OF SEMANTICS

Promises comprise information transmitted between agents. The effective transmission of information requires the existence of a common language[SW49, Sha40]. If each agent is an autonomous entity, how may agents learn a common language, or equilibrate different languages, in order to understand one anothers' promises?

Consider the existence of a language operation that transforms a body string b_1 by agent A_1 into a body string b_2 for agent A_2.

$$b^{(a)} = L_{ab}(b^{(b)}) \tag{6.27}$$

$$b^{(b)} = L_{ba}(b^{(a)}). \tag{6.28}$$

In order for $L(\cdot)$ to be faithful and express transitive properties such as long-range order, we need piecewise reversibility. Substituting (6.28) into (6.27)

$$b^{(b)} = L_{ba}(L_{ab}(b^{(b)})) \tag{6.29}$$

which implies that

$$L_{ab}(L_{ba}) = 1 \tag{6.30}$$

or the inverse relationship is the transpose:

$$L_{ab} = L_{ba}^{-1}. \tag{6.31}$$

In a general matrix representation, this implies that universal representation of the matrices representing L belong to the *unitary group* over language space a, b:

$$L^{\dagger}L = I. \tag{6.32}$$

The full unitary symmetry (if we take the general solution to this seriously, in the absence of other constraints) allows for general rotations of symbols. Thus so-called entangled states are, in principle, allowed for in this observation.

The set of transformations represented by L does not have to be assumed a global symmetry. The index labels gloss over the piecewise locality of the assumption that $L_{ij}(A_i)$ is a transformation that takes place at the location A_i, on its way from A_j. Similarly $L_{ji}(A_j)$ takes place at A_j om its way from A_i[82]. The limit of locality is thus the adjacency length between A_i and A_j

$$L_{ab}(A_i^{(b)}) \cdot L_{ba}(A_j^{(a)}) = 1. \tag{6.33}$$

If A_i and A_j are nearest neighbours, this is straightforward. However, if we regard the transmission of a promise through intermediate proxy agents, then comprehension and message integrity depend on the existence of non-local correlations, somewhat analogous to entanglement in quantum mechanics.

This symmetry is closely related to the observation that, even with a common language, in any promise relationship between agents:

$$U(U(+b)) \neq +b \tag{6.34}$$

$$--b \neq +b \tag{6.35}$$

(see section 3.10.2 in [BB14a]). Both relations imply a kind of long-range cooperation between the agents. These are analogous to the global symmetries of particle physics.

On seeing this familiar symmetry of the physical world, it is tempting to look for a conserved quantity, or a conservation law for the alphabets β, but it cannot be the

case that the alphabets are preserved. A conservation law would make the transfer of alphabetic messages into a zero sum game: what was passed on to a neighbour would be lost by the sender. This is not how evolution works. Instead, the process of equilibration is more like an epidemic duplication[83]. The transformations of language a location are more likely to be non-conserved, in general, and depend on the proper time (evolutionary change). One expects transmission of symbols in both time and space, but without conservation. Thus one could imagine dividing the inter-lingual transformations into two parts:

$$L = \underbrace{L(A)}_{\text{Conserved}} + \underbrace{L(A,\tau)}_{\text{Non-conserved}} \tag{6.36}$$

The first part could lead to a zero-sum conserved current of symbols from one agent to another, allowing migration without preservation, while the latter part allows symbols to be duplicated and spread. It might be fruitful, in the future, to consider how this process takes place, and compute Kubo relations for the transmission of promises[For75].

In order to cooperate, agents interacting at a distance need to have a sufficient level of similarity to local agents in order to be able to make sense of what they are promising to one another (see figure 6.2). This is true regardless of whether they directly adjacent or not. This must be a semantic equilibrium, mediated by a dynamic exchange process, in order to this cooperative behaviour to emerge. Unlike elementary physics, where locality is more obvious, cooperation between agencies could be long range, as long as there is adjacency over a long range. Semantics tend to follow humans, companies, organizations, races, countries, etc, and humans form multiple outposts with geographic separation.

6.3.3 INTER-AGENT LANGUAGE TRANSLATIONS

It is possible, in principle, to construct a linear transformation of one language into another:

$$\beta'_a = L_{a'a}(\beta_a). \tag{6.37}$$

Then, from the linearity, we may use the distributive law to say that a body b

$$b = \sum_{a=1}^{\dim(\beta)} c_a \beta_a = \sum_{a=1}^{\dim(\beta)} c_a L_{aa'}(\beta'_a) \tag{6.38}$$

Thus, as long as the matric L exists, the languages will be translatable. If the dimensions of the languages are not the same, only a subset of meanings will be translatable from one to the other. This might be asymmetric, allowing one agent to understand another, but not

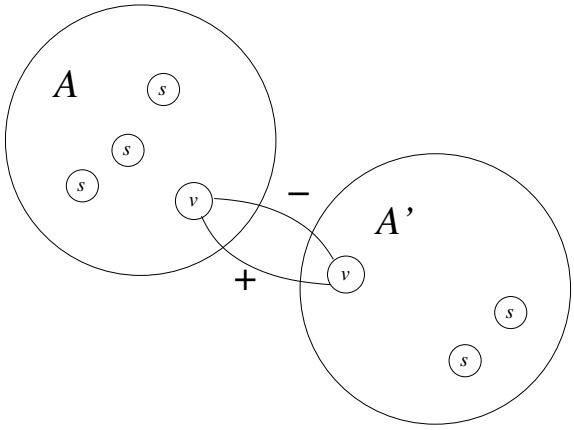


Figure 6.2: Agents need to have similar structure to make promises across agent boundaries, referring to internals of the other. The promise bodies do not have to be identical as long as each agent recognizes its own version of the other's promise.

vice versa. At the level of atomic intentions, we can introduce coding transformations a transition matrix for mapping

$$\ell^{(\alpha)}(A_i) \in L_{\alpha\beta}(\ell^{(\beta)}(A_j)) \tag{6.39}$$

for some invertible matrix-set of maps $L_{\alpha\beta}$.

Example 116 (Protocol language)**.** *Consider a body language with alphabet*

$$\beta = \{SEND, RECEIVE, SEEK, FORWARD, BACK\},$$

and $\beta' = \{PUT, GET, APPEND\}$*, then we can translate these:*

$$c'_1\beta'_1 = PUT \quad = \quad c_1\beta_1 = SEND \tag{6.40}$$

$$c'_2\beta'_2 = GET \quad = \quad c_2\beta_2 = RECEIVE \tag{6.41}$$

$$c'_3\beta'_3 = APPEND \quad = \quad c'_2\beta'_2 + c'_4\beta'_4 + c'_1\beta'_1 = SEEK + FORWARD + SEND \tag{6.42}$$

Hence there is a translation matrix:

$$L_{a'a} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \end{pmatrix} \tag{6.43}$$

which, in this case, is not invertible. Hence the language is translatable in one direction only.

In principle, however, it should be possible to restrict $\ell^{(\alpha)}$, such that translation may be performed faithfully as a bijection, by postulating a 'table of elements' for the chemistry of all promises in a semantic spacetime:

$$\ell^{(a)} \leftrightarrow \ell^{(b)} \quad \forall\, a, b \in \mathcal{L}. \tag{6.44}$$

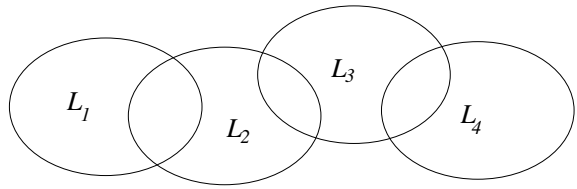


Figure 6.3: Overlapping patches of language require all agents in a patch to use a compatible language, and for (at least some) agents in each patch to comprehend a faithful translation of a neighbouring language, in order to bring long range order.

From here, the requirement for transmission of intent is that there be a piecewise continuity by partial overlap between neighbouring languages (see figure 6.3) becomes:

$$\ell^{(a)} \cap L_{a,a\pm1}(\ell^{(a+1)}) \neq \emptyset \quad \forall\, a \tag{6.45}$$

so that for $a \pm 1$ a neighbouring language of $\ell^{(a)}(A_i)$, i.e. $\Pi^{\rm ext}_{ij}(A_i) = 1$, a promise binding, of the following kind, may be mutually and equivalently comprehended:

$$\left\{ \begin{array}{ll} A_i^{(a)} & \xrightarrow{+b} A_j^{(a)} \\ A_j^{(a)} & \xrightarrow{(-b)} A_i^{(a)} \end{array} \right. \overset{a\to b}{=} \left\{ \begin{array}{ll} A_j^{(a)} & \xrightarrow{L_{ab}(+b)} A_i^{(b)} \\ A_i^{(b)} & \xrightarrow{L_{ba}(-b^{(b)})} A_j^{(a)} \end{array} \right. \tag{6.46}$$

This assumes that both L_{ab} and L_{ba} exist over the relevant bodies. The solution of the continuity relation (6.45) would then take the form:

$$\ell^{(a)} = \{\beta^{(a)}\} \cup \{L_{a,a\pm1}(\beta^{(a\pm1)})\} \quad \forall\, \alpha \tag{6.47}$$

i.e. the language of an agent in a coordinate patch α should consist of all interior body symbols, together with native translations of neighbouring languages.

6.4 PROPAGATION OF INFLUENCE (CAUSATION)

When promises are conditionally chained together in space or time, they can lead to cooperative processes that span multiple agents. Processes may therefore also be

described and documented using the language of promises. Transmission of intent leads to the propagation of influence, both good and bad. This can lead to sudden cascade failure modes, whereby systems breach catastrophically and beyond repair—an indication that the stability influence propagation needs careful attention, and prevention of failure by fault tolerance is an important strategy. This propagation of influence is what we call 'causality' in the physical sciences[84].

6.4.1 DYNAMICAL AND SEMANTIC INFLUENCES

Influence can be measured both dynamically (as changes to the quantitative behaviour of a system), and semantically (as conditional dependence on remote promises or 'services'). The semantics of such propagation are subtle, and semantics and dynamics interact inescapably. This is often a surprise to engineers who tend to separate semantics of outcomes from dynamical behaviours in their minds. It is important to recognize that semantics and dynamics a intrinsically linked in a functional system (see figure 6.4).

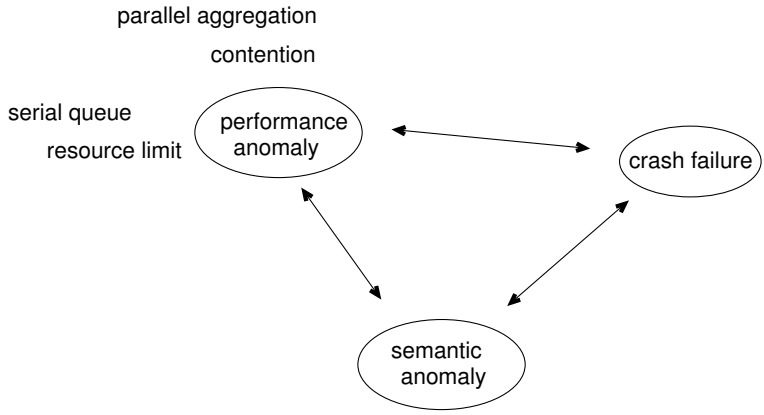


Figure 6.4: Although dynamical behaviours are the prerequisite basis for semantic relevance, the arrows of causation can go in both directions because unexpected or inappropriate semantics can effectively halt or constrain the ability for a system to continue its dynamical behaviour. An actual system crash is the extreme outcome, but incorrect dynamics can lead to unkept semantic promises, and semantic errors can lead to incorrect dynamical coupling.

In figure 6.4, the diagram shows three states of an agent that may lead to non-promise-keeping influence propagating. The assessment of these states is assumed to have undesirable influence, in the sense that some promise is not being kept. There are channels of influence in all the directions of three assessments. Let's consider these:

- A performance anomaly is what engineers know best. The best tools are available for monitoring performance, because these have the longest history. A process that is not fast enough for another agent to accept could mean a missed deadline, or an inability to accept the result. Information might come too quickly (because aggregate input rate floods the promise to receive), or it might come too slowly (because the receiver gives up listening before the answer arrives). Thus performance anomalies can lead to semantic side effects.

- A semantic anomaly can lead to a performance anomaly too. An incorrectly labelled output may lead to all outputs going to the same handler, instead of being routed separately, leading to a flood. Or output might go missing and be discarded because it is not recognized or falls below some critical threshold for acceptable quality.

- A performance anomaly can lead to an actual failure or crash because it leads to a positive feedback, such as a queue growing out of control, with subsequent thrashing and a cascade breakdown of promises within a fragile system.

- A semantic anomaly can lead to an actual failure or crash because it leads to a state that was not considered or planned for. A system might choose to fail in a controlled way, or might try to pursue a non-promised course of behaviour that once again leads to a cascade of changes from which normal behaviour cannot be recovered.

- A crash has obvious implications for semantics and dynamics. Both cease altogether at the location of the crash, which must have some influence on the remainder of agents in a collaborative state.

It is important to realize that, even if a system has been designed with redundancy and resilience aforethought, anomalous or even fatal behaviours in other agents will lead to an influence being propagated: this could necessitate actions by a remote agent, such as switching to a backup service, or delaying some followup action, etc.

Influence through dependency is different for processes based on promises and processes based on impositions (corresponding roughly to pull and push methods).

- **Promise** dependence (pull): if S promises R, and R accepts the promise, then R depends on A and thus a change in S may propagate to R, depending on whether R has promised precautions against being influenced. In this case influence by S is a 'choice' of R. To the extent that R has autonomous intent, it can (in principle) choose to block any change by seeking alternative source of the promise it relies on.

- **Imposition** dependence (push): if S imposes on R, then for R everything is the same as for the promise case, since R retains its ability to accept or reject. However, S may now be affected by the 'back reaction' of its imposition not being accepted. If S relies on the outcome of its imposition onto R, it is now vulnerable to the 'failure' of R to deliver the imposed outcome. If R does to voluntarily, S can expect to rely on a result. If R ignores the request, then S experiences a failure of expectations, and thus R influences S.

We see that imposition actually reverses the direction of influence of so-called 'fault propagation', in the view of an imposer. This is a particularly relevant issue in service relationships.

Example 117. *A client of a bank imposes a payment of an amount of money on its bank account (server). This bank account agent may refuse to accept and promise to carry out this transaction, for whatever reason. In this case, the affected party is the client rather than the bank (server).*

6.4.2 THE IMPORTANCE OF SCALE ON CAUSATION

When a change in one part of a system precedes a change in another, with a high level of probability, we use the term causation[85].

From a promise theoretical perspective, causation is a promise by a recipient R to use and promise to act A, based on information I from another agent S. We represent this by a conditional promise:

$$S \xrightarrow{+I} R \tag{6.48}$$

$$R \xrightarrow{-I} S \tag{6.49}$$

$$R \xrightarrow{A|I} T \tag{6.50}$$

Our understanding of causation is intrinsically linked to approximation into symbol categories, as information. If we aggregate all changes into a binary signal 'something happened', then causation becomes increasingly vague. If we trace very precise channels of information from atomic and isolated parts, we can pinpoint channels of causation with much greater plausibility. The dependence of causation on approximation and aggregation means that it is scale dependent. If we look at a coarse scale, it might be impossible to distinguish the order of prior events from final evidence. This is a fault of the methodology, not a proof that causation is non-existent.

If we can reduce a system to a network of low level atomic parts, then each point to point interaction may lead to transmission of influence, and thus propagation of causality. However, even this might not be quite what we expect. Isolation of neat linear stories

is not possible in general, especially in non-linear, strongly coupled systems. Thus causation can become circular.

6.4.3 SYSTEM STATE AND CAUSATION AS A CLOCK

Implicit in the notion of causality is the measurement of time. Time is also what clocks measure, and thereby define. A clock is nothing more than a Finite State Machine with its own limited resolution. When a clock changes state, we regard that as a clock tick. Thus system measured time advances when a system changes. If the state of a system does not change (i.e. it is in a steady state) then time does not pass in the system. Causation can only move through a system as fast as its slowest clock.

Our ability to detect change clearly plays a role here. If we are unable to observe interior changes, we may be unaware of the passage of time within an agent. Only exterior changes can be detected. The ability for an observer to detect change at all, requires changes in the observer too. Indeed, systems are *coupled* when they are observed.

Example 118. *For the duration of a measurement, two initially separate systems are coupled, and both may influence each other:*

$$S \xrightarrow{+measure} R \tag{6.51}$$

$$R \xrightarrow{-measure} S. \tag{6.52}$$

In this example, the fact that S promises the observability of a measure may influence its interior resources (a cost). When R accepts the offer, this might cause work to be done by S and R. What is the chain of causation? The promise of observability by S is prerequisite, but the initiation of acceptance by R is what causes a measurement to be made. If S fails to deliver the promised measure, then the failure originates at S. Notice how the semantics affect causal roles, in line with promise theory's (+) and (-) promises.

According to Nyquist's theorem, the observer has to experience changes twice as fast as the changes it is trying to observe in order to capture them with an acceptable level of approximation. Thus, an observer needs to poll changes twice as fast as what it is hoping to measure. If we are trying to detect faults, of trace causation, we must expect two cases:

- If the signals happen faster than the observer's clock, the observer can miss clues that happen too quickly relative to some fast reference time source.
- Alternatively, if the signals are persistent, but arrive quickly, relative to a fast reference clock, a slow observer might perceive time-ordered measurements as occurring within the same tick of its own clock.

Observing causality is a problem in time relativity. Any system, for which we can measure change, may also be regarded as a clock, whose changes are ticks, regardless of what other semantics we might attribute to its states.

Example 119. *Biological organisms have many cyclic changes that measure equivalence classes in time: daily rhythms or sleep and wakefulness, monthly menstruation, yearly mating, migration, and seasonal patterns. They are clearly influenced by environmental factors. Thus on an evolutionary timescale, one could speak of a causal influence. That influence is cached in local genetic mechanisms that operate on much shorter timescales. Thus causation is also operating on a biochemical level.*

Definition 120 (Simultaneity)**.** *Events are simultaneous as long as they occur with the same clock tick of a system's state clock.*

Example 120. *If we are trying to develop a model for fault diagnosis in computer systems, one might imagine looking for changes in the behaviours of the software, then reasoning that these lead to performance changes. Thus one might be able to infer the source of the software fault from the observation of performance anomalies (see figure 6.5). There are several problems with this.*

- *The server, by aggregating multiple processes, all of which share the same resources, generates entropy by stripping away the source of its load from the measurements it promises. The observer has no way of knowing which process, whether client driven or interior, led to the*
- *Process isolation (containerization, also called kernel namespaces, zone, and cgroups) are one possible answer to retaining some channel specificity, but this incomplete, since there is not full separation of resources. Even singular unikernel systems share memory subsystems, and therefore interact covalently through third party resource managers.*
- *Increased stress from distributed client impositions might provoke change in process, such as contention for kernel resources. Aggregation of workloads, by timesharing, or finite resource partitioning, generates entropy: the information about the circumstances is lost, because all the clients experience is a change in behaviour, but no cause except the 'demand' of faceless clients can be determined. Was there one particular client that broke the camel's back? Was it random or something particular about that client? Aggregation of processes couples the client promises together invisibly, and wipes out causal information.*
- *We are assuming that the stress on the server was caused by the clients somehow, but what if stress began due to a slow disk. This would have the same effect and*

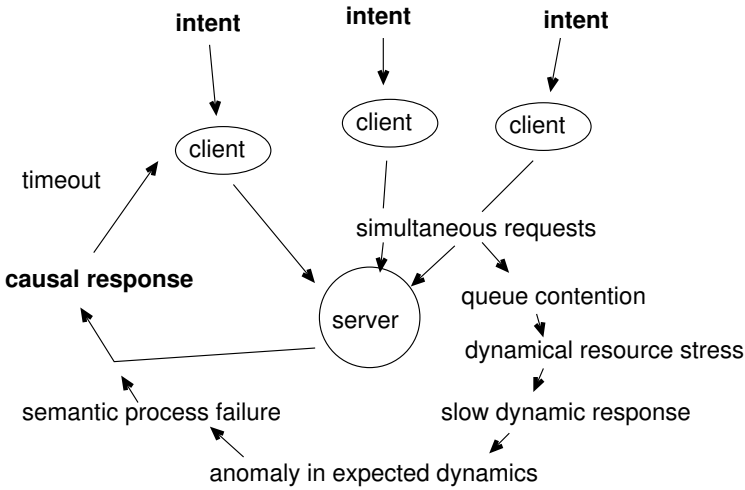


Figure 6.5: Causal feedback. The left arrow cycle shows the intended causality loop. The right arrow cycle shows how dynamics and semantics get mixed. We could regard the original intent to query a server as the root cause of a response. The aggregate distributed intent is the cause of a performance failure leading to a slow response, a timing promise not kept, and the timeout where the client gives up waiting is the cause of the process failure. Thus both sender and receiver are actively complicit in intentional behaviours that bring about faults.

> *be indistinguishable from anomalous processing due to heavy load, because over time the reduced service rate would lead to a queue build up. If we can only measure queue length (load average) we cannot distinguish heavy load from slow service.*

Finally, we should ask: on what timescales are we observing these events? Fast enough to observe the detailed interactions, or only slowly able to cumulative failures and successes?

6.4.4 THE NOTION OF A ROOT CAUSE

Because of the scale aggregation problem, and the loss of information to entropy of indistinguishability, tracing the cause from evidence of its outcomes is very difficult, and can only be achieved within the limits of the coarse grain approximations used to gather

evidence. In the chain of promises below,

$$S \xrightarrow{+I} R \tag{6.53}$$

$$R \xrightarrow{-I} S \tag{6.54}$$

$$R \xrightarrow{A|I} T \tag{6.55}$$

we can say that there is a clear sense in which S causes I, and I causes R to do A, when all promises are kept. Even if S fails to keep its promise of I, the intention is intact, and we can say that S causes the failure of A. Thus causation is not only about action. Inaction also propagates when there is intentional coupling.

Even if we have an intentional model for causation, there might be no single root cause. Agents act as causal 'switches', channing information by combination. In this example, there are two sources to a propagated effect:

$$S_1 \xrightarrow{+I_1} R \tag{6.56}$$

$$S_2 \xrightarrow{+I_2} R \tag{6.57}$$

$$R \xrightarrow{-I_1} S_1 \tag{6.58}$$

$$R \xrightarrow{-I_2} S_2 \tag{6.59}$$

$$R \xrightarrow{A|I_1,I_2} T, \tag{6.60}$$

thus there is no root cause for A. However, in the case of failure, one of the influences, say I_1 might change leading to a change in A, and (at least over this network) we can infer that S_1 was the source or cause of the transmitted fault.

When statistical methods are used, aggregation over samples is inevitable, and this leads some authors to issue blunt denials of the existence of causation, within the confines of their limited methodology. Clearly, there is a meaningful interpretation of causation, however it might not be possible to infer it backwards, because of choices that eliminate the information, e.g. averaging, clustering, grouping. This is a failure of information rather than an absence of causation.

6.4.5 DISTRIBUTED INTENT AND CENTRAL CALIBRATION

When intent is distributed it makes the challenge of inferring collective intent harder. We defer a full discussion of this until chapter 7. When agents collaborate, promising consistently requires a centralized calibration of intent. Calibration makes promises seem more deterministic, or at least more coordinated.

Once again, it is worth reminding readers that centralized promises do not necessarily imply a centralized network model, with a single computer server, or a single service location through which all traffic must pass.

Calibration of promises might be quite informal, based on trust. However, sometimes promises that are coordinated flow across agent. e.g. institutional, boundaries where there could be a loss of trust. Loss of trust is mitigated by constraining the freedoms one has for variations, e.g. restaurants may offer a single set menu instead of an à la carte selection, or the possibility to ask for a custom meal.

Example 121. *Internet of Things architectures. Many IT cloud companies have proposed an actual literal centralization of computing resources, uploading all data to a single central location. This is not necessary[BW15], and will scale infeasibly. Rather, standards of behaviour and formatting can be determined by promising to align with a central calibration standard, and data can then be accessed on demand.*

In a similar way, other standards, like electrical outlets do not require us to run cables all the way to a central organization. By standardizing the plugs and connectors and voltage standards, intent can be distributed and coordinated at the same time.

The difference between a centralized system and a decentralized one is less relevant than most system engineers think. We tend to rely on agents that are inside a trusted boundary, local, or otherwise seemingly close at hand, but there is no real difference between an agent failing to keep its promise when far away or failing to keep its promise when local. In section 5.13 we show that there is no principle difference between a system than is centralized or decentralized, other than trust in the reliability of the promises made. The same standards of promise keeping may be upheld regardless of the physical location of agent, provided the promises they make are equivalent. Thus Promise Theory expresses the invariances of a system, e.g. with respect to symmetries of relocation or repetition, and so on.

Example 122. *If one server goes down in a local datacentre, is it better or worse than if a server goes down in a remote cloud service provider?*

Example 123. *Suppose we keep a private electrical generator for emergencies, because we don't fully trust the power company to keep their promise to deliver power in all cases. The generator seems safer than trusting in the power company, because it is close by, but actually it makes its own promises to function correctly on demand. Its ability to work in case of a failure of the power company depends on the reliability of its own promises. There is nothing to say that the generator is more reliable than the power company just because it is within the trust boundary of our own home.*

Example 124. *Discussions of privacy and sovereignty of law often lead businesses and governments to require data to be stored on servers within their own jurisdictions. However, this is no guarantee that data are secure. Whether privacy is lost does not depend on whether data are stored close by or far away, it depends on the promises the data are able to keep, e.g. encryption, physical accessibility, etc.*

6.4.6 RATE OF FAULT PROPAGATION

To understand the rate of propagation of faults, we need to understand the promise bindings that lead to propagation. A promise binding has two parts:

$$S \xrightarrow{+b} R \tag{6.61}$$

$$R \xrightarrow{-b} S \tag{6.62}$$

There are several relevant timescales to consider:

- The rate at which a + promise is kept, i.e. changes in promise status take place. (MTTR)
- The rate at which changes happen to affect promise keeping (MTBF)
- The rate at which a - promise samples the results of the + promise.

In a steady state behaviour promise bindings are equilibria, but during transient activity, the initial behaviour is sensitive to these timescales. The result is called a race condition, and the outcome can depend quite sensitively and unpredictably on the particular circumstances.

- A + promise might be kept too late to be useful to the agent that relies upon it with its - promise samples the outcome before it has had time to keep its promise.
- A broken promise or fault might lie in wait for an extended time before it is relied upon and triggers a consequence that causes an observable effect. Thus the propagation rate might be significantly slower than the MTBF at the 'root cause site'. Similarly, once a critical collapse has been triggered, the catastrophe failure mode will travel at its own rate.

Example 125 (Ferry distaster)**.** *In the Estonia ferry disaster of 1994, the door to the car deck was not closed properly. Water did not start to enter the ferry destabilizing it until hours later when the sea conditions passed some critical threshold. Had the promise to keep the door closed been kept the cumulative failure threshold would have been immunized.*

6.4.7 SEPARATION OF DYNAMICAL AND SEMANTIC OUTCOMES

There are two kinds of networks:

- Adjacency of agents (spacetime), resource transport networks (Markov)
- Cooperative processes (with valued outcomes) (Prerequisite)

A semantic outcome may happen on a different timescale to the underlying processes that lead to it.

Misinterpretation of the semantics of a failure. If we don't have an extensive causal understanding of the pathways of influence, we could miss possible predictions.

6.4.8 PROMISE TRAJECTORIES

When a promise $\pi = \langle S, R, b(\tau, \chi_\tau)\rangle$ is kept, the constraint χ_τ, belonging to the promise body, expresses a restriction on the state q_τ. To elaborate on this matter, more rigorously, we need a more detailed model of agent state, by imagining sets of microstates q_τ that refer to the variables inside agents that pertain to promises of type τ. There are various possible representations of this using functions, and using matrices that we need not go into here (see [BF07a, BF08], for instance). The key features of such a representation can be described as follows.

Given a promise $\pi = \langle S, R, b(\tau, \chi_\tau)\rangle$, and a set of states that may be associated with the promise q_τ, we associate, with the enforcement of the promise, a set of operations $\hat{O}_\tau(\chi_\tau)$ such that the action of the operator on a general state propagates the agent from the old to new state $q_\tau \to q_0$ that complies with the restriction χ_τ, whose orbit is the set of promised states:

$$\chi_\tau \quad : \quad q_\tau \in \{q_\chi\} \tag{6.63}$$

$$\hat{O}_\tau(\chi_\tau) \quad : \quad \hat{O}_\tau(\chi_\tau)\{q_\chi\} = \{q_\chi\},\ \forall q_\chi. \tag{6.64}$$

$$\hat{O}_\tau(\chi_\tau) \quad : \quad \hat{O}_\tau(\chi_\tau) q_\tau = q_0 \in q_\chi. \tag{6.65}$$

Thus, another way to represent this, is to say that the operation $\hat{O}(\chi_\tau)$ effectively generates a function f_π that is absorbing for all its arguments to a fixed domain: $f_\pi(q) \to \{q_\chi\}$. This is called a convergent operation.

The introduction of a promise, or a change to an existing promise, can alter the trajectory of the state $q_\chi(t)$, i.e. the sequence of states the agent moves through, which, in general, is an orbit or function of time. The relationship between promises and agent trajectories was described in [BF07a, BF08]. For differential changes the resulting properties of motion closely resemble Newton's laws for particles, as they must, where an effective force $\hat{F}$ can be related to a complete promise binding $\delta\hat{O}$. The time-development of the trajectory leads to one kind of propagation of influence. The change of the promise that guides the trajectory leads to another, known as the response function (see section 6.4.13). For the purpose of these notes, all we need to know is that:

- Promises are kept by associating convergent operations, that are convergent and idempotent, when acting on local agent states[86].

- The repeated verification and implementation of a promise's constraint may be carried out by repeated application of the operator or function on the subset of states, within the agent, that refer to the promise.

- The resulting state, or sequence in time, is the promised τ-trajectory of the agent. This represents its observable behaviour.

- A change in the promise that selects this trajectory can only be made by the agent itself, or by the agent promising to accept changes from an external source (another agent). Such a change may be regarded as the propagation of influence over the agent's trajectory by the source, in the form of a conditional promise that depends on the source.

6.4.9 BASICS OF PROPAGATION OF INFLUENCE

If we assume that two agents are able to communicate for the purpose, the flow or propagation of intent from a point of origin to other agencies in a system involves cooperation between its component agencies.

Let's look at examples of transmission from Sources S through intermediary agents M to Receivers R, with graphical nomenclature:

$$S \to M \to R \tag{6.66}$$

We assume that the services promised by these nodes are refreshed on a continuous basis, or with some fixed time scale, so that we can speak of time to repair a promise not kept. We cannot take it for granted that influence propagates. For example, suppose we consider the promises:

$$S \xrightarrow{+\text{I am sad}} M \xrightarrow{+\text{I am happy}} R \tag{6.67}$$

There are two promises between three agents. For propagation to occur we would have to be able to say that the promise from S influenced the promise from M. But, as written, there is nothing to indicate that the cause of M's asserted happiness has anything to do with S's asserted sadness. In fact, M has not even promised to care whether S is sad or not.

To encode that, we would have to introduce conditionals, or a functional dependence, and acceptance promises:

$$S \underset{+\text{I am sad}}{\overset{-\text{You are sad}}{\longleftrightarrow}} M \underset{+\text{I am happy}|\text{You are sad}}{\overset{-\text{You are happy}}{\longleftrightarrow}} R \tag{6.68}$$

The shorthand for this is the notation

$$S \xrightarrow{+\text{I am sad}} M \xrightarrow[+\text{I am happy(you are sad)}]{\xleftarrow{-\text{You are happy}}} R. \tag{6.69}$$

Now we can say that the reason why M is happy is in response to S's sadness. The state of S has influenced the state of M. R is influenced by M's happiness in promising to accept it, but the influence goes no further.

Example 126. *A less frivolous example helps to underline the point:*

$$Outlet \xrightarrow{+power} Light \xrightarrow[+light(power)]{\xleftarrow{-light}} Person. \tag{6.70}$$

Here we have an agent promising power (a power outlet), and a light bulb accepting this (by plugging into it), and the result is a promise of light, which is accepted (perhaps by switching it on). Unlike models of electricity, a promise model does not assume an inevitable flow of current, or an inevitable flow of influence.

The model of pre-conditions encodes what we mean by causation, or propagation of influence. Causation is a special case of propagation, because it is purely dynamical. It requires a chain of dependencies:

$$S \xrightarrow{+a} M_1 \xrightarrow{+b|a} M_2 \xrightarrow{+c|b} \ldots. \tag{6.71}$$

To complete this, we have to use the conditional promise law to quench the requirements and complete the cooperation[BB14a].

Comment 6 (Conditional promise quenching)**.** *Recapping, from basic promise theory: a local conditional promise, is equivalent to a non-conditional promise, if the condition $+b$ is also promised to be 'true', i.e. satisfied:*

$$A_1 \xrightarrow{T(+b),S|b} A_2 \equiv A_1 \xrightarrow{S} A_2 \tag{6.72}$$

If the condition is only promised independently (possibly true or satisfied), then

$$A_1 \xrightarrow{+b,S|b} A_2 \simeq A_1 \xrightarrow{S} A_2. \tag{6.73}$$

A non-local conditional promise, is equivalent to a non-conditional promise, if a promise to acquire the dependency $-b$ is given:

$$A_1 \xrightarrow{-b,S|b} A_2 \equiv A_1 \xrightarrow{S(b)} A_2 \simeq A_1 \xrightarrow{S} A_2 \tag{6.74}$$

Since the latter is the usual case, we give it the special quasi-functional notation in the promise body to reduce two promises to one.

Using the result in the box above, it is straightforward to show that, in order for a chain of conditional promises to propagate some property τ, the chain must stack up dependencies, accumulating points of failure along the way. Relaying through 'middlemen' makes a system more fragile. Starting from a non-conditional edge or boundary node A_0:

$$A_0 \xrightarrow{b_0} A_1 \xrightarrow{b_1(b_0)} A_2 \xrightarrow{b_2(b_1(b_0))} A_3 \xrightarrow{b_3(b_2(b_1(b_0)))}, \ldots \tag{6.75}$$

And, the bindings are completed by

$$A_0 \xleftarrow{-b_0} A_1 \xleftarrow{-b_1} A_2 \xleftarrow{-b_2} A_3 \xleftarrow{-b_3}, \ldots \tag{6.76}$$

For causal linkage, only the final link in the chain has to be keep a promise of τ; but, if we want to speak of propagation of τ, e.g. propagation of consistent information, the it has to be the same information. The pre-conditional chain is sufficient to claim a causal pathway no matter what the previous promise types. Note that, because there is no identical equivalence between non-deterministic outcomes, we cannot say that a promise to propagate influence necessarily reduces to a produce of the propagation over the links.

6.4.10 TRAJECTORIES AND PROCESS PROPAGATION

Recall the definitions of trajectories and processes in section 1.16. When an agent makes promises, the promises become analogous to what we might think of as the 'laws of motion' for the process in physics. These 'laws' embody patterns, which unfold into trajectories that depend on the initial states of agent variables[87]. As always, in Promise Theory, we have to deal with what was promised, what was accepted, and what was assessed. Promises that are kept most of the time lead to behaviours that are regular and characterizable as trajectories through some set of states.

A propagator is a structure that describes the change in a single agent's trajectory. It takes the form of a matrix, operator, or distribution.

Definition 121 (Scalar promise trajectory propagator)**.** *For a single agent A, we may define the propagation function, transition function, or matrix T, to be the matrix that carries one set-valued state variable q to a subsequent value q', referring to interior states of A. The operation implies an acceptance transfer of state between any interior subagents. Notice that the transition implies a tick of the implicit process clock, so we may attach arbitrary interior time labels to these events.*

$$q'(t') = T_{\pi,q}(t,t')q(t), \tag{6.77}$$

transforming a local promise

$$\pi_q(A) \to \pi_{q'}(A), \;\; and \;\; t \to t'. \tag{6.78}$$

and

$$\pi_q(A) : A \xrightarrow{+q} A_?. \tag{6.79}$$

Similarly, we can consider the succession of transformations of a vector promise binding

Definition 122 (Exterior trajectory propagator)**.** *Consider a promise binding between two agents S and R:*

$$\pi_+ : S \xrightarrow{+b_S} R \tag{6.80}$$

$$\pi_- : R \xrightarrow{-b_R} S \tag{6.81}$$

The binding between these is:

$$b_{S\to R} = b_S \cap b_R, \tag{6.82}$$

where the arrow on the left denotes to polarity of the promise direction. We define a binary propagation function $T^{(2)}$ to evolve the trajectory of this binding's overlap, for set-valued variables b_S and b_R, indicating an acceptance transfer of state:

$$\left(\pi'_{b'_S}(S) \oplus \pi'_{b'_R}(R)\right) = T^{(2)}(t',t)\left(\pi_{b_S}(S) \oplus \pi_{b_R}(R)\right). \tag{6.83}$$

and t, t' now refer to the common or entangled time for the combined superagent formed from the interior shared clock for $S \oplus R$. Reversible changes are represented by invertible operators. Desired state changes are non-invertible operators[BC11].

A propagator for cooperative influence across non-local aggregate paths and chains (from agent to agent) is:

Definition 123 (Process chain propagator)**.** *Consider again a promise binding between three agents S, M, and R:*

$$\pi_+ : S \xrightarrow{+b_S} M \tag{6.84}$$

$$M \xrightarrow{-b_M} S \tag{6.85}$$

$$\pi_M : M \xrightarrow{+b_M|b_S} R \tag{6.86}$$

$$\pi_- : R \xrightarrow{-b_R} M. \tag{6.87}$$

There are now two bindings to propagate:

$$(S \oplus M) \text{ and } (M \oplus R), \tag{6.88}$$

leading to an effective propagator for

$$(S \oplus M \oplus R). \tag{6.89}$$

Propagation functions or matrices have implicit polar boundary conditions, that indicate the orientable causal direction in which they operate. At each scale, there are irreducible propagators that are independently assessable, e.g. the propagators with retarded (causal) boundary conditions:

$$\begin{aligned}
\Delta_>(S,R) &= \Delta(R|S) \\
&= S \xrightarrow{+b_0} R && (6.90)\\
&= R \xrightarrow{-b_0} S && (6.91)\\
&= \overline{\Delta}_<(R,S) && (6.92)\\
\Delta_>(S,M_1,R) &= \Delta(R|M_1S) \\
&= \Delta_>(\Delta_>(S,M_1),R) && (6.93)\\
&= \overline{\Delta}_<(R,\overline{\Delta}_<(M_1,S)) && (6.94)\\
&= S \xleftarrow{\pm b_0} M_1 \xleftarrow{\pm b_1(b_0)} R && (6.95)\\
\Delta_>(S,M_1,M_2,R) &= \Delta(R|M_2M_1S) \\
&= S \xleftarrow{\pm b_1} M_1 \xleftarrow{\pm b_1(b_0)} M_2 \xleftarrow{\pm b_2(b_1(b_0))} R && (6.96)
\end{aligned}$$

6.4.11 PROMISE TYPES THAT PROPAGATE INTENT TRANSITIVELY

There are more examples of propagation, relating to semantics, or transmission of meaning, by what we call the transitive property. The rules of semantic transitivity were discovered by A. Couch[CB09], in relation to semantic inference.

- If C is affected by B and B is affected by A, then C *might be* affected by A.

The latter expresses both potential 'tolerance' and potential uncertainty.

- If C carries B and B carries A, then C carries A.
- If C encapsulates B and B encapsulates A, then C encapsulates A.

Note that, in this case, there is no sense in which and agent carrying another agent depends on the being carried by another, so we cannot require it as a precondition. This is a different kind of propagation. Note also the subtlety of meaning. It might not be true that if C contains B and B contains A then C contains. That depends on what you mean by contains. From whose perspective?

This underlines how making influence travel from one agent to the next is straightforward by mutual agreement; but, making it travel through multiple agents, without changing character altogether is far less certain. So there are two independent issues that we can separate:

1. How to define propagation of independent channels of influence between directly differential (adjacent) agents.
2. How to sum up the semantics of influence over longer paths (the integral problem).

In both cases, promises help to define the problem.

Note that, when we talk about promise agents, it is *assumed* that they try (continuously, by repetition) to maintain the state of promise compliance, not merely in a single event. In other words promises are timeless, unless bounded in duration relative to other promises. They do not assume that a promise that was kept in the past is still kept in the present.

6.4.12 CONDITIONS FOR EXTENDED PROPAGATION (CHAINS AND PROCESSES)

The transmission of intent (and loss of intended behaviour or faults), through a system, is a network problem. The presence of a fault in a localized agent can be transmitted onwards through the system, to dependencies, with widespread semantic and dynamic repercussions.

1. On each link, the $\pm$ promise types need to match
2. On each link, the bodies $+b$ and $-b$ need to have non-zero overlap.

The propagated effect is proportional to the overlap, if and only if the forward and reverse types are the same. These properties can be be encapsulated into a generalized response function for neighbouring agents, that takes into account parallel scaling.

6.4.13 THE INSTANTANEOUS RESPONSE FUNCTION

Consider now a function that is at the heart of propagation of influence between agents. It explains how promised behaviours can spread, how agents influence one another, and change one another's promises and trajectories. This is the generalization of the well-known linear response function from hydrodynamics or electrodynamics, in which we add the labelled semantics of the promises. It measures broadly the impact of a promise made at source, in a set of agents that rely on it 'downstream'.

Let $\pi_{ij}^{(\pm)}$ be the $\pm$ promise matrices, and $\Pi_{ij}^{(\pm)}$ be the promise adjacency matrices [Bur15a] for the full promise interaction (see figures 6.9, 6.6, 6.7). It helps to define a few derivate parts of a promise, since a promise is a tuple, and we often only need parts of it. The set valued components of a promise bundle[BB14a]:

$$\pi = \{S\} \xrightarrow{b} \{R\} \tag{6.97}$$
$$b = \langle \tau, \chi \rangle \tag{6.98}$$

may be written:

$$b(\pi) = b \tag{6.99}$$
$$S(\pi) = \{S\} \tag{6.100}$$
$$R(\pi) = \{R\} \tag{6.101}$$
$$\tau(\pi) = \tau(b(\pi)) = \tau \tag{6.102}$$
$$\mathrm{sgn}(\pi) = \mathrm{sgn}(b) = \mathrm{sgn}(\tau) \in \{+, -\} \tag{6.103}$$
$$\chi(\pi) = \chi(b) = \chi. \tag{6.104}$$

With these definitions, we may now measure the number of agents affected by neutral promise bindings of any particular type of promise τ, according to what proportion of what is promised to them they choose to accept.

Definition 124 (Response function). *Consider two sets of agents: S_i (the sender set) where $i = 1, \ldots, s$, and R_j (the receiver set) where $j = 1, \ldots, r$, and let there be non-square promise matrices π^+_{ij} for promises from S to R, and π^-_{ji} for promises from R to S. Then we may defined the response function for transfer of intent from $\{S\}$ to $\{R\}$ by*

$$R\left(S, R, \pi^+, \pi^-\right) = \frac{1}{S} \sum_{i=1}^{S+R} \sum_{j=1}^{R+S} \Pi^+_{ij}(S,R)\Pi^-_{ji}(R,S) \times \left| \frac{\tau(\pi^+_{ij}) \cap -\tau(\pi^-_{ji})}{\mathrm{sgn}(\tau(\pi^+))\tau(\pi^+)} \right| \times \left(\chi(\pi^+_{ij}) \cap \tau(\chi^-_{ji})\right) \tag{6.105}$$

where $|x|$ is the cardinality of the set x. The dimensions of $R()$ are the dimensions of χ.

The notation of the definition looks intimidating, but it is straightforward. The promise matrices $\Pi^{\pm}$ describe the graph of $\pm$ promises, in a neutral binding, from the sender set to the receiver set (see figure 6.9). These need not be balanced. The product of the forward matrix, with the matrix of promises in the backwards direction, selects only those combinations of agents that have promises in both directions. Roughly speaking, this has the form:

$$\text{Response} \propto \text{channel width} \times \text{type-binding} \times \text{agreed transfer} \tag{6.106}$$

The term involving the promise type τ is always positive and dimensionless, and non-zero only if the reverse promise type is $-\tau$ when the forward promise type is τ, required to form an acceptance binding. Finally, the term involving the body constraint χ has the magnitude of the overlap of constraints between what was sent (+) and what was received (-). In other words, this is magnitude of the accepted transmission, analogous to the mutual information[CT91].

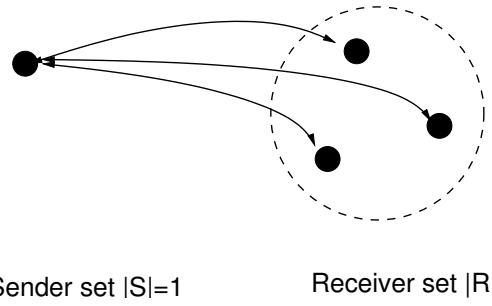


Figure 6.6: (Amplifying) gain in transferring promise bindings from one superagent set $S = 1$ to another receiver set $R = 3$ is proportional to the number of complete bindings and their overlap strength. In this case, if we assume the bindings are equal, the gain would be 3 times the strength of each binding.

Comment 7 (Complicated response function?)**.** *The apparent intricacy of the definition of response exposes the fragility of response in a system where both dynamics and semantics play a role. There are three major failure modes:*

- *Missing or incomplete binding* $\pm$*.*
- *Mismatched promise type (input validation).*
- *Insufficient overlap between provider and user (input validation).*
- *Insufficient redundancy or channel capacity.*
- *Accidental positive gain, or misconfiguration leading to unintended amplification.*

The response function as designed applies for a static snapshot of a system configuration, and does not allow for the fast that the agents and promises might be in a state of flux. Thus we call it the instantaneous response. If we define a system real time t, then it could be added to the promise quantities $R\left(S(t), R(t), \pi^+(t), \pi^-(t)\right)$ to account for time-varying dynamics. Although a single snapshot has a linear form, the effect might be non-linear due to the non-deterministic nature of promise keeping, and effectively changing topology.

6.4.14 PROPAGATION OF UNCERTAINTY

Uncertainty in these bindings can occur from both ends. It is also useful to refer to the error function, for characterizing the degree of variability in the behaviour of the agents.

The error function combines the resulting 'error' or variation from a number of sources $a, b, c, \ldots$:

$$\mathrm{err}(a, b, c \ldots) \tag{6.107}$$

In the case of independent random errors, this reduces to the normal Gaussian Pythagorean expression:

$$\mathrm{err}_G(a, b, c \ldots) = \sqrt{a^2 + b^2 + c^2 + \ldots} \tag{6.108}$$

By using continuous variability in the keeping of promises, we may choose the threshold at which the failure to meet tolerance expectations is exceeded, and a variation becomes a fault.

If an agent is fault tolerant, then it can, in principle, absorb faults and propagate a non-faulty response forward. If not, then a fault will propagate by omission of an accepted promise kept.

Lemma 26 (Quantitative response)**.** *The propagated effect of a promise π from S to R may be written $\Delta_\pi(R|S)$, written in a pre-conditional notation:*

$$\pi_+ : S \quad \xrightarrow{+b_S \pm \delta_S} \quad R \tag{6.109}$$

$$\pi_- : R \quad \xrightarrow{-b_R \pm \delta_R} \quad S \tag{6.110}$$

$$\Delta_{\pi_\pm}(R|S) = \quad S \quad \xleftarrow{(b_S \cap b_R) \pm \left(\sqrt{\delta_S^2 + \delta_R^2} \cap b_R\right)} \quad R \tag{6.111}$$

The recipient is the ultimate arbiter of what is propagated, hence the final expression is limited by b_R

Thus, instead of the component probability:

$$P(S \ \mathbf{AND} \ R) = P(S)P(R|S), \tag{6.112}$$

the reliability of this promised propagation is the probability that there is propagation

$$P(\Delta(R|S)) = P(\pi_+)P(\pi_-|\pi_+) \tag{6.113}$$

where $P(\pi_+)$ is the reliability of keeping the promise π_+. This is a more complicated matter, now involving the join probability distributions for the agents' promises. The basic result is:

$$P(y_- \leq Y \leq y_+|X) = \int_{y_-}^{y_+} \frac{f(x, \alpha)}{f_X(x)} d\alpha \tag{6.114}$$

6.4.15 SPEED OF RESPONSE PROPAGATION

The rate at which influence travels is a distributed quantity, and thus it is subjective . From a god's eye view perspective we note that each promise $\pi^{(\pm)}(A_i)$, made at agent A_i takes time $\Delta T(\pi^{(\pm)}(A_i))$, so that total path time of $A_1, \ldots, A_P$ is:

$$\Delta T_{\rm path} = \sum_{i=1}^{P} \Big(\Delta T(\pi^{(+)}(A_{i\neq P})) + \Delta T(\pi^{(-)}(A_{i\neq 1})) \Big) \tag{6.115}$$

which in a broadly homogeneous system is approximately proportional to the path length, or the total number of promises to be kept. In the broader scheme of time, the local time taken might vary too, so one can allow some random variation. An initial implementation of state might take longer than the time needed to maintain it.

Some agents experiencing faults may prevent propagation of influence altogether, in which case the total path time tends to infinity, and the local speed of propagation becomes zero. The time perceived to have passed by any agent in the system depends on the agent's own clock. Each agent will, in principle, measure time differently.

6.4.16 THE INSTANTANEOUS INTENTIONAL GAIN

Definition 125 (Intentional gain per agent)**.** *The intentional gain, per agent, in going from a sender set of agents $\{S\}$ to a receiver set $\{R\}$, with a compatible promise binding, may be defined as the number of agents making promises from a sender set. The gain $G_\tau(S,R)$ may be written as the dimensionless normalized value of the response function applied to promise bundle matrices of a type set $\{\tau\}$:*

$$G(S,R,\pi^{\pm}) = \frac{1}{S} \sum_{i=1}^{S+R} \sum_{j=1}^{R+S} \Pi^{+}_{ij} \Pi^{-}_{ji} \left| \frac{\tau(\pi^{+}_{ij}) \cap -\tau(\pi^{-}_{ji})}{{\rm sgn}(\tau(\pi^{+}))\tau(\pi^{+})} \right| \left(\frac{\chi(\pi^{+}_{ij}) \cap \tau(\chi^{-}_{ji})}{\chi(\pi^{+}_{ij})} \right) \tag{6.116}$$

Lemma 27 (The maximum gain of any superagent interaction)**.** *The instantaneous gain of any intentional response is limited to*

$$G(S,R,\pi^{\pm}) \leq |R|, \tag{6.117}$$

i.e. the number of receiver agents.

The proof is follows from the fact that:

$$\left(\frac{\chi(\pi^{+}_{ij}) \cap \tau(\chi^{-}_{ji})}{\chi(\pi^{+}_{ij})} \right) \leq 1, \qquad \frac{1}{S} \sum_{i=1}^{S+R} \sum_{j=1}^{R+S} \Pi^{+}_{ij} \Pi^{-}_{ji} \leq \frac{SR}{S} \to R, \tag{6.118}$$

and

$$\left| \frac{\tau(\pi_{ij}^{+}) \cap -\tau(\pi_{ji}^{-})}{\mathrm{sgn}(\tau(\pi^{+}))\tau(\pi^{+})} \right| \in \{1, 0\}. \tag{6.119}$$

Consider the implications of the lemma. Gain is blind to success or failure. It assumes only that the promise made by the sender is made and kept to some degree. If the promise could not be honoured, and became a bottleneck, e.g. if too many receivers tried to make use of its promise, then the gain would simply amplify the failure to keep the promise by th source. Thus, a failure is not only about whether or not this binding configuration is reliable: if the binding is completely reliable, there is still the possibility of an error, fault or flaw at source being transmitted perfectly to a larger number of agents, with the same gain.

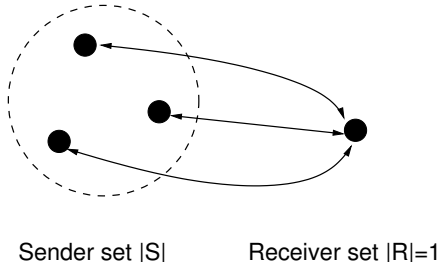


Figure 6.7: A load sharing system is one with unit gain, i.e. the response is exactly the strength of the promise binding.

Definition 126 (Intrinsically amplifying system)**.** *If every agent in a causal dependency process has intentional gain $G > 1$.*

6.4.17 IMPEDIMENTS TO PROPAGATION

The category of issues involving reliance on intermediaries or proxies is a large topic. We rely on intermediaries all the time, often without thinking. Indeed, we are so used to ignoring intermediaries for our intent that it becomes a basic source of design flaws: to ignore the promises made by intermediaries. This goes back to our predilection for thinking of the world as an extension of ourselves.

Example 127 (Climbing rope)**.** *If you are promising to hold on to a mountain, using a rope, you are trusting the rope.*

A chain:

$$A \xrightarrow{+c} B \xrightarrow{+b|c} C \tag{6.120}$$

Satellites of agent C:

$$A \xrightarrow{-b} C \xleftarrow{-b} B. \tag{6.121}$$

Same, just no conditional. Both are influenced by C. E.g. C could be a share resource, such as tenancy host. Two tenants are competing for resources. If keeping the promise to one affects C's ability to keep its promise to the other, then there is an undocumented influence between A and B, through this third party. This is called a second order effect.

Example 128 (Containment as a modular process strategy)**.** *A container is a form of isolation kernel for processes or partial processes. Containment is a popular strategy linked to modularity, especially in computing. It is also used in human organizations in the form of departments, teams, etc. Popular technologies for containers in computing include Docker and Kubernetes to name just a couple for the Linux operating system. Containers in Linux use a feature of the Linux kernel, in which partial kernel isolation is provided by process groups or 'cgroups' . Containers are processes that have private, but not guaranteed resources. For example, a process running in a container cannot write to shared workspace, it must attach its own private space. However, that shifts responsibility for isolation to the process itself, and opens for the possibility of covert shared channels, e.g. a shared database. Indeed, without resource sharing, containment would be of little use. Linux containers may still interact by resource interference.*

Docker containers offer a useful packaging mechanism for software modules, when expressed as services, through which one may bundle related services according to documented promise dependencies. This enforces some kind of access control rigour between agents that potentially make similar promises. Thus is helps to modularize development and increase predictability of version management.

Since containers can be restarted, based on a fixed interior image, they are sometimes used to reset the runtime state of a service which has run into trouble. This is the reset approach to repair. When restarted, such modules lose their runtime state, so a 'stateless' interior strategy is advocated—i.e. runtime state is pushed to the exterior of a process. This leads to a proliferation of dependencies, which may in turn have adverse consequences, including increased software complexity. By pushing dependencies outside of modules, one introduces a new layer of management and potential inefficiency.

The allusion between containers and boundaries means that containers are sometimes thought to form fault domains, and even security boundaries, limiting the propagation of faults and promises. That is not strictly accurate, as there is no complete isolation

as long as a module is connected by input and output to others. Containers do nothing to assure fault tolerance. Moreover, the absence of a promise is not a promise of absence, so the fact that promises don't explicitly reach through a boundary is no assurance that an assessment will not lead to effective or implicit propagation—and negative promises are hard to keep.

6.4.18 PROPAGATION OF INFORMATION (AWARENESS)

Awareness of the state of a system is the key to being able to respond quickly to faults, and potentially repairing them before the Nyquist sampling interval expires and the fault has assessed consequences.

The reliability of information, concerning the keeping of promises, moves through a system may be considered a measure of system self-awareness. The information may be destined for a the assessment of a human operator, or for direct assessment and reliance by other component agents. The conundrum for agent networks is that information travelling from one location to another can be distorted by passing through intermediate agents along an information path. In some cases, it might be possible to detect the loss of integrity, but it cannot necessarily be repaired.

6.4.19 IMPLICIT AND EXPLICIT AWARENESS BY REPETITIVE PROMISE KEEPING

Regular periodic sampling (Nyquist Fourier method) allows us effectively continuous insight into system state of up to half the frequency of the sampling.

State awareness can be basically redundant close to stability, with low frequency random faults, since repeatedly maintaining a convergent state is a stable equilibrium. If we repeat this fast enough, like a heartbeat, we simulate continuity of operational state.

Then we are in some kind of Zen state of knowing without needing to know(!)

6.4.20 DISTORTED PROPAGATION - 'CHINESE WHISPERS'

The children's game of Chinese Whispers illustrates how intermediate agents can distort influence, when propagated in a chain. The same scenario was studied with mutual promises rather than impositions, in [BB14a], as the Consistent Knowledge Theorem. Let S and R be any two agent nodes, and let $\pi : k$ be a promise body that implies

communicating knowledge k. Nodes S and R have consistent knowledge k iff

$$S \xrightarrow{+k_s} R \tag{6.122}$$
$$R \xrightarrow{-k_s} S \tag{6.123}$$
$$R \xrightarrow{+k_r} S \tag{6.124}$$
$$S \xrightarrow{-k_r} R \tag{6.125}$$
$$S \xrightarrow{(k_s = f(k_r, k_s))} R \tag{6.126}$$
$$R \xrightarrow{(k_r = f(k_r, k_s))} S \tag{6.127}$$

where $f(a, b)$ is some function of the original values that must be agreed upon. The proof is given in [BB14a].

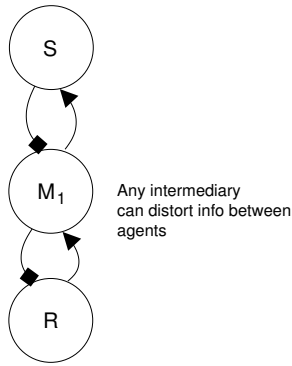


Figure 6.8: Man in the middle and loss of integrity. Shown with impositions.

Knowledge $\mathbf{def}$ can be passed on from agent to agent with integrity, but we must distinguish between agents that end up with a different picture of the information as a result of their local policies for receiving and relaying their knowledge.

1. Accepted from a source, ignored and passed on to a third party intact.

$$\begin{aligned} S &\xrightarrow{+\mathbf{def}_1} M \\ M &\xrightarrow{-\mathbf{def}_1} S \\ M &\xrightarrow{\mathbf{def}_1} R \end{aligned} \tag{6.128}$$

   Note that the agent M does not assimilate the knowledge here by making its own version $\mathbf{def}_2$ equals to $\mathbf{def}_1$, it merely passes on the value as hearsay.

2. Accepted from a source, ignored and local knowledge is then passed on to a third party instead.

$$\begin{array}{lll} S & \xrightarrow{\mathbf{def}_1} & M \\ M & \xrightarrow{U(\mathbf{def}_1)} & S \\ M & \xrightarrow{\mathbf{def}_2} & R \end{array} \tag{6.129}$$

Here the agent M accepts the information but instead of passing it on, passes on its own version. The source does not know that M has not relayed its data with integrity.

3. Accepted and assimilated by an agent before being passed on to a third party with assurances of integrity.

$$\begin{array}{lll} S & \xrightarrow{+\mathbf{def}_1} & M \\ M & \xrightarrow{-\mathbf{def}_1} & S \\ M & \xrightarrow{\mathbf{def}_2=\mathbf{def}_1} & S \\ M & \xrightarrow{\mathbf{def}_2=\mathbf{def}_1} & R \\ M & \xrightarrow{\mathbf{def}_2|\mathbf{def}_1} & R \end{array} \tag{6.130}$$

M uses the data from S, assimilates it ($\mathbf{def}_2 = \mathbf{def}_1$) and promises to pass it on (conditionally $\mathbf{def}_2|\mathbf{def}_1$) if it receives $\mathbf{def}_1$. It also promises to both involved parties to assimilate the knowledge. Only in this case does the knowledge $\mathbf{def}$ become common knowledge if one continues this chain.

The first two situations are indistinguishable by the receiving agents. In the final case the promises to make $\mathbf{def}_1 = \mathbf{def}_2$ provide the information that guarantees consistency of knowledge throughout the scope of this pattern. We can now define scope.

Definition 127 (Scope of common knowledge)**.** *Let S be a set of source agents with consistent knowledge $\mathbf{def}$, and let the set $\mathcal{A}(X, \mathbf{def})$ mean the set of nodes that have assimilated knowledge from a set of agents X. The scope $\mathcal{S}(\mathbf{def})$ of $\mathbf{def}$ is the union of S with all agents that have assimilated knowledge originating from S, i.e.*

$$\mathcal{S}(\mathbf{def}) = S \ \cup \ \mathcal{A}(S \cup \mathcal{S}(\mathbf{def}), \mathbf{def}) \tag{6.131}$$

Comment 8 (Consistency)**.** *The simplest way to achieve common knowledge is to have all agents assimilate the knowledge directly from a single (centralized) source agent. This minimizes the potential uncertainties, and the source itself can be the judge of whether the appropriate promises have been given by all agents mediating in the interaction. This is the common understanding of how a network* directory service *works. Although simplest, the centralized source model is not better than one in which data are passed on epidemically from peer to peer. The problem then is simply in knowing the boundaries of scope. Agents may thus have consistent knowledge from an authoritative source, either with or without centralization.*

6.5 PROPAGATION WITH BRANCHING

Processes often need to specialize into segregated narratives. They do so by branching. Each branch point becomes a gateway of scale transducer for the subordinate process.

6.5.1 BRANCHING WITH INSTANTANEOUS SERIAL AMPLIFICATION

The ability to amplify effect is both a useful function and an essential fragility. Amplification of input can also mean amplification of error or fault[88]. A fault at source could at best mean a loss of gain, depending on the promise structure..

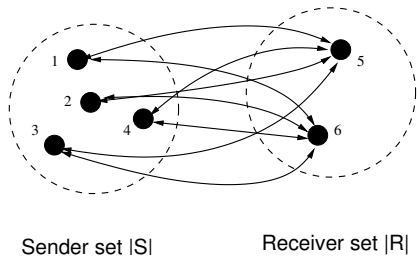


Figure 6.9: Gain in transferring promise bindings from one superagent set $S = 3$ to another receiver set $R = 3$ is proportional to the number of complete bindings and their overlap strength. In this case, if we assume the bindings are equal, the gain would still be 3 times the strength of each binding, since the gain is per agent of the sender set.

Looking at figure 6.9, we see that the the number of source agents $|S| = 4$, and the receiver set has $|R| = 2$ agents.

Definition 128 (Promise valency of an agent)**.** *The number of promises of given type made by an agent, measured in bindings per agent.*

The valency of the agents in the source and receiver sets may be written $s = 2$ and $r = 4$. Because promises of the same type are idempotent[BB14a], the binding valency of source agents can never exceed the number of agents in the receiver set, and vice versa, thus:

$$\begin{aligned} r &\leq |S| \\ s &\leq |R| \end{aligned} \tag{6.132}$$

The valencies, representing unbound promises, can at best combine in the promise graph to give the product, if each agent binds maximally:

$$\begin{aligned} \text{Tr}\left(\pi^+(S)\pi^-(R)\right) &= \text{Tr}\left(\begin{array}{cccc|c} & & & & 0 \\ & & & & \\ & & 0 & & \\ & & & & \\ \hline + & + & + & + & 0 \\ + & + & + & + & \end{array}\right)\left(\begin{array}{cc|cc} & & - & - \\ & & - & - \\ & 0 & - & - \\ & & - & - \\ \hline & 0 & 0 & \\ & & & \end{array}\right) \\ &= \text{Tr}\left(\begin{array}{cccc|cc} 0 & & & & & \\ & 0 & & & & \\ & & 0 & & & \\ & & & 0 & & \\ \hline & & & & r & \\ & & & & & r \end{array}\right) \\ &= rs \end{aligned} \tag{6.133}$$

So, we can say that

$$\text{Tr}\left(\pi^+(S)\pi^-(R)\right) \leq rs \leq |R||S|, \tag{6.134}$$

So

$$\frac{1}{|S|}\text{Tr}\left(\pi^+(S)\pi^-(R)\right) \leq |R| \text{ bindings per source agent.} \tag{6.135}$$

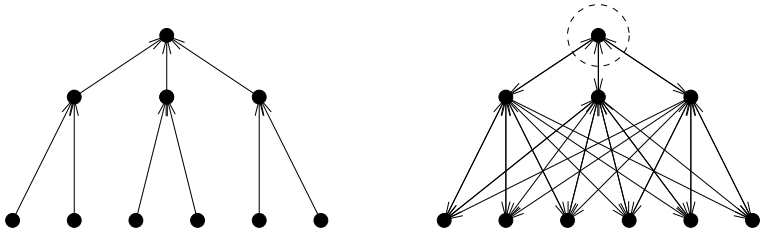
This is the upper bound for propagation of response. The interesting conclusion here is that amplification doesn't depend on $|S|$ but only on $|R|$. This is slightly counterintuitive, but arises from the spreading of S into R, which is maximized if there are plenty of possible destinations for the source, and we have already measured this relative to the size of the source pool. It is, of course, possible that some agents do not make or keep the full battery of promises in this kind of redundant arrangement, but it's useful to assume some homogeneity and list a few examples of the amplification.

Example 129 (Gain for some binding configurations)**.** *Assume a cooperative binding between two super-agent clusters (source and receiver) of different sizes, and assume that each agent in these promises to fixed numbers of agents s, r in the other.*

Config	*(+)*	*(-)*	*Response*	*Gain*	*Comment*
$S \leftrightarrow R$	1	1	$\leq \chi_+$	≤ 1	*Pass through*
$S \leftrightarrow RRR$	1	r	$\leq r\chi$	$\leq r$	*r-linear ampl. (multicast)*
$SSSS \leftrightarrow RRR$	s	r	$\leq \frac{rs}{\|S\|}\chi$	$\frac{sR}{\|S\|} \leq R$	*Redundant proxy*
$SSSS \leftrightarrow RRR$	s	$r = 1$	$\leq \chi$	≤ 1	*Redundant load sharing*
$SSSS \leftrightarrow R$	s	1	$\leq \chi$	≤ 1	*Aggregation*

In each case, we can interpret the gain factors as both an amplification of intent and a potential dependency leading to amplification of faults, errors, and flaws in design. Any promise not kept near the source will be amplified by the same functional arrangement.

In multi-stage or multi-tiered cooperative structures, like directed acyclic graphs (DAG) forming branching processes, we can look specifically at the branching of tree-like structures as amplifiers. Staging adds a further dependence on the previous stage, thus some amplification is compounded, which each tier introduces new possibility for failure. The branch points in a tree are called 'single points of failure' (see definition 129), meaning that a fault at one of these has consequences for a fault in the total system. If we add fault-tolerance, the branching simply leads to amplification of the inaccuracy or staleness of the promises, but the system can continue.



(a) (b)

Figure 6.10: Trees are fragile from the root down, as errors are amplified by branching with dependence.

Comment 9 (Human reasoning)**.** *Human reasoning is a branching process that leads to multi-stage instability. Each train of reasoning is non-unique, and multiple viewpoints abound. To assure stability (e.g. through semantic averaging, or selection) one must evaluate all possible trains of thought. This is a time consuming process, hence placing humans in the position of critical decision-making is recipe for instability. The aim must be to decouple reasoning from rapid action, by pre-consideration (modelling). Taxonomy is a model that attempts to exploit branching to avoid inconsistency, unfortunately many properties along which a taxonomy is categorized are shared and non-unique. This often causes anomalies and leads to places where agents should be associated, in a way that contradicts the assumptions of the model.*

6.5.2 CUMULATIVE AMPLIFICATION OF RESPONSE

Since the instantaneous response function determines the gain only at a time t, the maximum impact is

$$\text{Impact} = \int_{t_1}^{t_2} G(t)dt \tag{6.136}$$

This applies to fault amplification as well as intentional effect. Thus during faults, minimizing the impact can mean minimizing the gain, or minimizing the time over which the fault exists. As usual, there are two ways for the continuity of a response to be maintained over a trajectory:

- Redundancy of components enabling failover to a redundant component in case of interruption or delay.
- Rapid repair, with MTTR faster than the sampling time of the downstream dependent process.

6.5.3 BRANCHING PROCESSES WITH UNCERTAINTY

A branching process is, by its very nature, unstable. To prevent an uncontrolled 'explosion' of effect, we have to contain it to a finite number of agents. Errors may be transmitted or become compounded in series (see fig 6.11).

Definition 129 (Single point of failure)**.** *An agent that one or more other agents rely on to fulfill a conditional promise.*

This definition goes beyond a simple physical structure definition of a classical approach to reliability, to include logical promise relationships. A single point of failure is a node that can amplify a fault from a single point to a cascade of derivative promises.

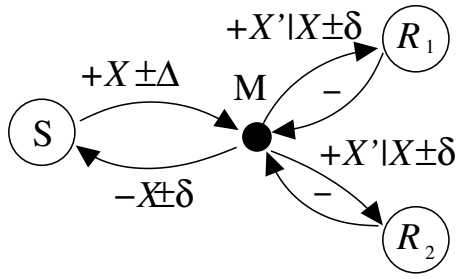


Figure 6.11: An error at the head of a chain of dependences cannot later be mitigated without consequences.

1. S promises service X, within its accuracy tolerance $X \pm \Delta$.

2. Intermediary/relay M, promises to accept the value in the range $X \pm \delta$, where $\max(\delta) < \max(\Delta)$.

3. If Δ happens to be small enough to be within the range $\pm\delta$, it can be accepted and the intermediate agent is able to promise X' to R, within the limits of its own accuracy $\mathrm{err}(\Delta, \delta)$.

4. If $\Delta > \delta$, I is not able to accept the service from X, and cannot promise X' to R, hence there is a complete failure.

If intermediate agents are sufficiently tolerant of inaccurate promise keeping, they can transmit dependent promises within the limits of their own fidelity. However, if the source feeds in a service that is out of the range of tolerance of the intermediary, it must fail to keep its promise, as the condition $S \pm \delta$ is not satisfied

Because of the branching in the latter stage, the error will be amplified by the number of branches, i.e. it will be passed on along each of the causal branches.

6.5.4 Dependency chains, workflow pipelines, agent reducibility, and promise propagation

When a system forms a causal chain, or a workflow pipeline, its output may depend on a number of stages, each of which plays a role in the detailed causal structure. The structure of a pipeline system is an end-to-end delivery problem, with intermediate agents (see section 11.3-11.4 in [BB14a]). Each link in the delivery chain may be a scaled version of the transducer pattern (described in section 4.4.7 of [BB14a]).

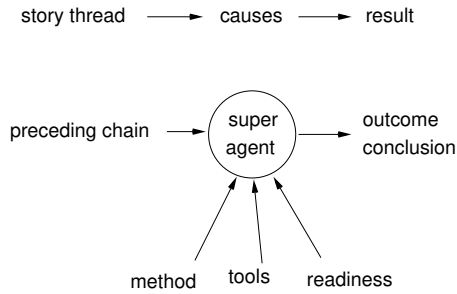


Figure 6.12: The result promised by one or more preceding chains is transmuted into one or more outcomes. The transmuting agent has its own dependencies that partially determine when it needs to redo work in order to keep its latest promise. The output depends not only on previous stages, but on the methods, tools, and state of readiness of each stage of the pipeline. So a change in method would require the agent to rework its promises and cache the result making it available at all times to the next stage.

The pipeline is a collaboration in which each previous agent promises an output conditional on its input. An important question is: what happens when the inputs are themselves aggregate quantities, like big data or assorted components, with possibly different processes involved in computing the promised output, how does the next link in the chain know when the previous agent has finished its transformation? We need to limit the promise of observability of the intermediate states, to make changes atomic. In computer science, such a transformation is sometimes called a critical section, a monitored change, or a mutex.

Example 130 (Change observability semantics)**.** *In Unix file semantics, if an agent has a file open, it should see that same file until it is closed. It will also see changes to the same file. However, if a new file is created and renamed to the first file, any open files will see the old version, and only new agents will see the changed version, which is why it is normal to make changes by making changes to a copy:*

```
copy file.ready to file.new
transform file.new
rename file.new to file.ready
```

securing the 'immutability' of the data, like a privately scoped variable.

Indeterminate, intermediate states should not normally be visible to the next agent in a chain, else this might result in a fault. The agent is therefore responsible for making a clear promise to inform when it is safe to exchange data.

Example 131 (Unsafe observability)**.** *Suppose that the agent being passed through the work chain were a letter. One agent provides the paper, the next agent types the letter, and the final agent puts it into an envelope and sends it. Without a proper promise about what constitutes a finished letter, then as soon as the second agent typed a single character of the letter, the final agent would see a change and grab the letter and send it, possibly leading to an error or fault down the line. Agents therefore need to have control over what changes they permit downstream agents to see. This is the essence of permitted observability or state locking.*

Consider what happens when we chain together a number of agents to transform inputs into outputs, along an end to end process (figure 6.12). The input agents (data, raw materials, etc) are transformed into various output agents (produce, processed data, etc) by each stage in the chain. The simplest case is that there is a linear chain, but a more realistic case might include branching of outputs and aggregation of several inputs from different sources. In any scaled system this would be expected at a detailed level, because each stage in a chain would already be a redundant superagent set of agents, as we shall discuss in the next chapter. In the simplest case, at least, each pathway from a chain which may look something like the example in figure 10.1, provided we disregard the semantics of components as in classical reliability theory; in the semantic world of modern systems each stage takes on a form more like figure 6.12.

The signalling of a causal change in the status of a dependency binding, in figure 6.12, will depend on various criteria, which must be explicitly promised, because each promise is assessed by the receiving (super)agent as the trigger on a conditional promise of its own. To makes assessments, protected from observability of intermediate states, we might need to reduce the superagent to its component parts in order to fully understand how this causation is restricted (as in figure 6.13).

The idea of outcome propagation is that a change in the state of what is promised upstream triggers activation of the next conditional promise downstream. So there is a question about how a change is detected or assessed by the downstream agent. For instance, agents might look at:

- A change in the upstream promise definition (like a new version of its software).
- A change in the output of the same promise (the data output changes).
- A change in the downstream agent's criteria for assessment (software update downstream).
- A change in the method or assumptions or other dependencies by which any chained promise is kept.

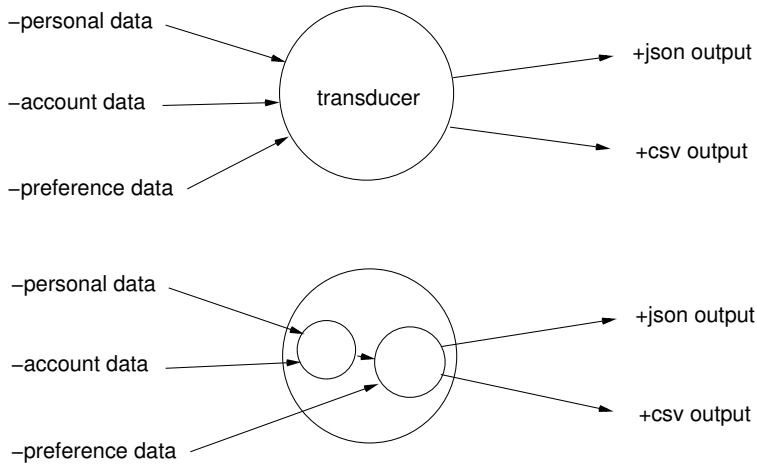


Figure 6.13: When agents in a pipeline need to aggregate outcomes into several subsets of inputs, in order to keep their outgoing promises, they may be reducible, i.e. reveal internal logical structure. Understanding reducibility and irreducibility of outcomes may be critical to promising accurate signalling during cooperation.

Clearly, there are all kinds of changes we might choose to respond to, or simply ignore. The distinguishability of the states of the system are a matter for design, which is why these promises need to be known precisely.

Example 132 (Software building pipelines: from 'make' to 'CFEngine')**.** *The well known Unix software utility* `make`*, used to build desired end state targets used a text file to represent a dependency graph for a build process:*

```
target: dependency list
    actions if dependencies newer than target
```

Promise theoretically, `make` *is a hybrid tool, which promises actions rather than outcomes in response to certain prerequisite states. Its criteria for promise keeping only looks at the timestamps of the data objects involved. So if the content changed, or a transducer tool changed this would not necessarily result in updated versions being built.*

CFEngine later extended this idea to introduce 'smart sensors' based on object types, and models for all the possible promises different kinds of object could keep, including time stamps, sizes, content patterns, permissions, etc. Moreover, the actions to be taken were designed to be convergent so that no change would ever break a promise made by an object. By being more specific, CFEngine could trigger repairs to broken promises,

moving only in the direction of compliance with a promised end state, somewhat like an immune system.

Programming pipelines can be arranged, promise theoretically, to keep any number of promises, in chains of dependency, provided downstream agents observe what is promised to them from upstream, with detailed attention to semantics.

Example 133 (Programming type safety)**.** *Intermediate states of processing may arise from incomplete promise-keeping, flouting expectations. Data type 'safety' in programming is an expression that has become widely used. There is a (slightly naive) implication that as long as data are transmitted in accordance with type range rules the outcome of a program must be correct, or 'unsafe' things cannot happen. Promise theoretically, one is asking: does the data keep a set of promises, according to the intended recipient's expectations? This presumes that the necessary cooperation to agree on these expectations has been carried out. Ultimately, however, the concept of type safety carries the unspoken assumption that programming is a series of impositions, not a series of agreed promises. Promise theoretically, it is clear that downstream agents have to be ready to deal with the situation in which their use-promises are not quenched by what is offered from upstream, and so it is the recipient's responsibility to promise its behaviour under all conditions, i.e. to ensure type safety, regardless of what the provider promises.*

Agent reducibility becomes an issue in connection with the observability of trigger states in the following way. A superagent boundary, defined according to one criterion, might not be an accurate description of indivisibility in all for all criteria. If a particular agent handles certain promises differently, then it is reducible to sub-agents that effectively handle those differences.

Definition 130. *Agent reducibility A super agent, which makes independent and distinguishable promises, may be reduced to its independent parts, if exterior agents rely on these distinctions.*

For instance, looking at figure 6.13: if two inputs are aggregated, separately from the third, before being observed to trigger a downstream change, then components (within the superagent) need to make different promises to react to changes for the aggregated channels and the independent channel. In a promise chain, transducers responsible for processing work may appear as atomic stages, but may in fact contain reducible elements. Unless downstream agents are aware of these constraints, their dependent promises may not be kept as intended.

6.5.5 'PULL' VERSUS 'PUSH' IN A CHAIN

In the pipeline example of section 6.5.4, we did not mention how a chain of promises arranged its communication. Communication implicitly decides the level of determinism in the outcome: the two main ways are by imposition ('catch!') and by persistent promise ('ready when you are'). (see figure 9.5). If we want to make pipelines predictable under all circumstances this needs to be determined.

Although push is essentially an illusory mechanism, built from the illusion of rapid polling, it is a useful one. When updates are rare push can provide a quick notification of a change, where constant polling might be deemed a waste of time: a little non-local state management can provide implicit propagation of readiness to send information. On the other hand, when arrivals are frequent, push leads to queueing instabilities, and polling offers safe buffering (readiness to receive), and localized management of state.

A push based methodology, thinking entirely in terms of events is stateless in the sense that the intermediate stages of the pipeline are ephemeral, not cached. If they are not caught immediately by the next stage immediately, they may be dropped, like packets on a network. Impositional queues work in this way. However, this does not allow for self-healing characteristics, and may lead to unpredictable behaviour, especially close to the queueing instability. The alternative, like the 'make' pipeline is that all intermediate stages are retained, in their current best state of repair, ready to be used by the next stage if needed.

As pointed out in figure 6.12, a pipeline does not only depend on the promises of previous stages, but on the properties of the transmuting agent. If a change occurs in its local promises (with no change in its inputs) the it may want to redo its own work according to a new standard.

Example 134 (Dependencies and Makefiles)**.** *Make bases its rebuilding of cached state, along its pipeline, only on the timestamps of dependencies. Usually one should always make the Makefile itself a dependency of every object, because a change in the tooling or optional characteristics, e.g. compiler flags, optimization levels, new software versions, etc, are intended to affect the next stage of the pipeline, but those changes would not be seen until the pipeline received a new longitudinal change, from its source, unless promises were also dependent on lateral changes like agent characteristics.*

A pull strategy works well to maintain a state of readiness, because each autonomous agent in the chain is responsible for updating its product outputs in accordance with all its dependent promises. The next stage can pull this at any time (for example, even if there is a loss of data and the pipeline needs to be reconstructed). It is not desirable to have to work through an entire pipeline from the beginning just to repair one detail.

Example 135 (Automotive pipeline)**.** *Suppose a car, emerging from a production line is discovered to have fault brake settings. The car can be returned to a particular stage of the build process for repairs. There is no need to dump the entire car and rebuild a new one from the beginning.*

We expect pipelines that promise definite outcomes to converge towards their promised states.

6.5.6 PROPAGATION WITH CONVERGENCE

Whether a process involves forward or backward branching of causal pathways or not, the state space of outcomes may branch out or converge. Aggregation, composition, and convergence of the state space of a system may or may not map onto its physical representation. If states are aggregated internally to an agent, we may not be able to see formally independent agents. The agents are then said to be *reducible*.

The opposite of a divergent amplifying process is a funnel, or convergence from a larger number of agents or states towards a smaller number of agents or states. Aggregation over multiple alternatives may mean redundancy, but not if the aggregated promises are all distinct. In the worst case, aggregation is merely composition, affording no improvement in stability. We have to pay attention to the promise bodies, i.e. the types and constraints, to determine the nature of the aggregation.

6.5.7 INTRINSICALLY CONVERGING SYSTEMS

Systems that strive to operate within a stable region, e.g. to achieve a fixed point or drift towards an attractor basin are converging. See also related notes about closures and monads.

Definition 131 (Intrinsically converging system)**.** *If every agent in a causal dependency process has intentional gain $G \leq 1$, it may be convergent i.e. it is not amplifying.*

Lemma 28 (Intrinsically converging system with high fidelity)**.** *A strictly converging system with has $G = 1$ and $|R| \leq |S|$*

A system is amplifying if $G > 1$, and may therefore converge If $G < 1$, then either the receiver is throttling the or filtering for partial acceptance of the source, or there are faults in propagation of intent. In either case, there is a reduction in effect, whether intentional or unintentional.

6.6 CAN WE DEFINE RESPONSIBILITY FOR KEEPING A PROMISE?

Responsibility is an emotionally charged term in system forensics. It can be associated with liability, blame, and reprimand. There is a tendency to want to assign 'blame' a person when faults lead to loss. This is a punitive interpretation of responsibility, and one that does not often make sense, unless willful malice or negligence were demonstrably at work. One also sees attempts to find simplistic root causes (provenance), and quick fixes for future prevention. These tendencies are understandable emotional reactions to loss, but we need a more rational understanding of responsibility, based on the science of causation, to actually stand a chance of making a difference.

Responsibility, in an 'atomic' promise viewpoint, has an interesting simplicity about it. If we can step back from the idea of attributing blame, there is something to be learned from asking the question: which agency or agencies are in a position to be *able* to keep a promise, and hence *could* be considered responsible?

6.6.1 SUBJECTIVITY IN ASSESSMENT OF FAULTS AND ERRORS

The idea of responsibility is diffuse, but it relates to the ability of an agent to respond to a situation in order to keep a promise. A responsible agent (literally a responder) might be one that has decision-making ability, or even full control over whether a promise can be kept (for a stakeholder). In order to be able to make a decision, there have to be alternatives. An agent telling a lie, or exaggerating its claims might be called irresponsible. There is thus an ethical/moral dimension to responsibility too.

The assessment that a promise has not been kept is a subjective one: different players in the system might assess it differently. Their assessments can depend on context or circumstances; so how can we easily attribute a unique source to the perceived failure? This is the challenge of a distributed system with multiple stakeholders.

Example 136. *In a restaurant, a meal is ordered from the menu. One person enjoys the meal, the other doesn't. The latter (dependent agent) assesses the meal to not be what was promised. The former (fault tolerant agent) assesses the meal as acceptable.*

- *The agent that rejects the meal might be considered discerning of quality, but goes hungry and cannot work.*
- *The agent that accepts the meal eats and continues its work.*

Can a cause be attributed to the waiter, the head chef, the sous chef, the butcher?

Whether or not a promise has been kept depends on the kind of promise binding. As we know about promise bindings, the receiver or promisee has to make its own promise to accept what is offered; thus it shares responsibility in outcomes. Indeed, refusing to accept what is offered is the ultimate control decision of autonomous agents.

Example 137. *A doctor promises a patient: if you take these pills you will be cured. If the patient does not keep a promise to take them, then it will not be cured, and thus the responsibility for being cured lies ultimately with the patient, not the doctor.*

6.6.2 SMART AND DUMB AGENT RESPONSES

A complicating factor in systems is the existence of smart and dumb agents.

Definition 132 (Dumb agent)**.** *A dumb agent keeps its promises but does not adapt to external circumstances.*

Definition 133 (Smart agent)**.** *A smart agent may try to adapt to external circumstances in order to try to keep its perception of responsibility, rather than sticking to the scripted promises.*

When dumb agents fail, they may simply stop or keep trying, but a smart agent might try to adapt. Humans often act as smart agents, but are also asked to forego smart behaviours to remain in a predictable regime. Smart behaviours effectively change the set of promises in real time, in response to external events. This means that the predictability of the promises may be compromised, leading to new behaviours outcomes that were not promised, or expected by others. In this sense, smart behaviours are often frowned upon, because they violate initial expectations. Without criteria for limiting smart behaviours, one must consider intelligent adaptation to be 'out of control'.

6.7 'PUSH VERSUS PULL' CAUSAL INFLUENCE

'Push versus pull' is one of the choices systems designers have in designing cooperation[89]. In a push method, the source is favoured as the deciding agency (like a brain model). In pull methods, the receivers are favoured as the deciding agencies (like a cooperative society model). The law of agent autonomy would thus seem to favour pull methods over push: push must be imposed, with the intent to violate a receiver's autonomy. We shall explore whether this is indeed the case in detail. This is not a moral issue, but one that affects our ability to make a consistent interpretation of the transmission.

6.7.1 DEFINITIONS OF PUSH AND PULL

The terminology pulling and pushing data are often used about attitudes to causal *intent* in communications. These lead to some confusion, so it's worth mentioning them. I define these in accordance with the same information channel principles.

Definition 134 (Push)**.** *A method of communication in which a source agent S imposes its messages onto a recipient R without invitation.*

$$S \xrightarrow{+M} \blacksquare\, R. \tag{6.137}$$

In data signalling, packets may be carried over a wire, enter a network interface and be queued up for sampling by a receiver, before the receiving process is ready to accept them. This is imposition. The receiver may then promise to sample (-) the messages from the shared queue. The channel flow is thus controlled by the sender.

Definition 135 (Pull)**.** *A method of communication (sometimes called publish-subscribe) in which a receiver invites a source to provide a certain quota of messages for sampling via an agreed channel.*

$$R \xrightarrow{-M} S \tag{6.138}$$

which is then promised by the source, voluntarily subscribing to it,

$$S \xrightarrow{+M} R \tag{6.139}$$

The flow along the channel is thus controlled by the receiver.

Pull is always the fundamental 'last mile' stage of a sampling operation; it may involve active polling of the queue to match timescales that satisfy the rigours of Nyquist's theorem. It optimizes message transfer according to the downstream capabilities. Push driven systems are sometimes associated with *reactive* or *event driven* systems—though this can be misleading. Push and pull are effective on different timescales. Push (notification) is useful in connection with small signals when source data are sparse and a receiver needs a short wake up message to collect a package from the source, enabling it to conserved resources. Push therefore provides non-redundant information when arrivals are sparse or infrequent, on the timescale of the receiver's sampling[BB08]. Pull systems make more efficient use of queue processing, by utilizing the information about autonomous capacity to balance load.

The definitions of push and pull do not refer to complete methods for communication or influence; rather, they characterize aspects of complete descriptions. Both push and

pull methods for cooperation can be analyzed using promises and impositions (see fig. 6.14). Neither construction is a complete description of a system, so the properties of these approaches depend on what other promises are made around them.

6.7.2 PROPERTIES OF PUSH AND PULL

If we draw a super-agent boundary around agents that are transmitting influence either by push or pull, what can be say about the components' reliabilities, i.e. the fidelity of that particular constellation of agents?

• *Speed of response (agility)* The speed of response depends only on the path length and processing time of the agents, which is independent of push and pull.

In a push model, a message from source can be imposed immediately without delay, but the recipient might not be awake, or have time to pay attention to it straightaway.

$$R \xrightarrow{-b} S \tag{6.140}$$

$$S \xrightarrow{-b} \blacksquare\, R \tag{6.141}$$

$$R \xrightarrow{+rsvp|b} S \tag{6.142}$$

$$*S \xrightarrow{-rsvp} R \tag{6.143}$$

The star shows the point at which the sender may claim awareness of the outcome. This has one prerequisite. Because the outcome happens at a location R that is inaccessible to the source of the intent S, for confirmation of success, the recipient R must promise a confirmation of receipt service 'RSVP'. Only when this notification of a response is accepted does the sender know the outcome of the imposed push.

In a pull model, the rate at which an recipient polls the source for updates might be fast or slow, but it happens at its own behest, so it is ready to the update by definition.

$$S \xrightarrow{+b} R \tag{6.144}$$

$$*R \xrightarrow{-b} S \tag{6.145}$$

The star shows the point at which the sender may claim awareness of the outcome. This time, there are no prerequisites for this to happen, and the outcome is immediate on initiating the pull. Not the provider does not know about the state of the recipient (it may not need to, since it has no stake in the intent to acquire the service), so one could make the same kind of monitoring promise from recipient to server, but this is no longer a necessity.

Given the uncertainties in both cases, it is impossible to say whether push or pull might be faster than the other. This depends on the circumstances. What we can say is that the push case is not guaranteed to affect the recipient (if it is not awake or willing to

accept the push). In the pull case, the acceptance of a message is implicit in the attempt to subscribe to it, but the source might not be available when the recipient wants to pull.

Lemma 29 (The speed of response for pull vs push)**.** *The speed of responding to an intended outcome is strictly indeterminate for both push and pull. Either could be faster under particular circumstances.*

The proof follows from the autonomy of the agents, and the fact that a promise binding depends on both (+) and (-) promises, each of which may fail to keep its promises. There is thus insufficient information to determine the weak link in a chain.

• *Is the initiator (source of intent) able to know the outcome (success/failure) of the cooperation?* In the case of the push, the initiator is the source, and the result is the recipient, thus intent and result are at opposite ends of the operation to obtain the result. In the absence of further promises to feedback the result, the initiator cannot know the outcome of the operation. This is sometimes called a 'shot in the dark' or 'throwing it over the wall'.

The initiator may promise to monitor the result, if the recipient promises to allow it as a service. Thus we may add two additional promises (with corresponding uncertainties) to determine whether the push succeeded. The recipient can, of course, lie. The source's distance from the point of compliance is a fundamental problem for verification. Indeed, if the receiver intentionally did not accept the push in the first place, it is unlikely to report back truthfully either. Thus, a push scenario is inherently unknowable.

In the case of pull, the initiator is the recipient, and it is the affected party. Thus there is immediate confirmation of success or failure. This represents maximum possible certainty. It is impossible to lie about the success of the promise.

Lemma 30 (The uncertainty of pull vs push)**.** *The uncertainty that a promise will be kept, leading to an intended outcome is strictly less for pull than for push.*

The uncertainty experienced by an agent when imposing or pushing (+) is never less than the uncertainty of a single push. In case of a failure to keep a promise, the agent can retry serially, which adds to the time to outcome. In case of disseminating to multiple recipients, the time is never less than the time for each single promise to be kept. The uncertainty experience by an agent when accepting of pulling (-) from a service is the same as for a push imposition when aggregating results from serial or parallel agents without redundancy. If we add redundancy, the time for failover or pre-fetching of results may be less than a serial wait. Thus pull-based systems offer more avenues for scaling the reduction of response times than push-based systems.

• *Can conflicts be resolved if multiple sources have different intended outcomes for a single recipient target?* If several agents try to push to a single recipient, the result

is inconclusive, and potentially order dependent. This is the multithreading contention problem. Serialization may be provided by locking, or the requests could be separated into independent non-shared contexts, if the recipient promises to support that. The context for each push is computed by the source agents, and is unknown to the recipient, and hence cannot be reproduced later for adaptation or recall by the recipient.

If an agent tries to pull from multiple sources, the result can be intentionally serialized by the recipient, and either integrated into a single context, or separated into reproducible, independent contexts. The contexts are computed by the instigating, pulling recipient and thus may be reproduced later for adaptation.

This problem is considered further in section 6.7.5.

Lemma 31 (Conflict resolution in pull vs push)**.** *Pull based systems can resolve conflicts, but push systems cannot resolve conflicts.*

All information to resolve conflicts in a pull scenario is local to a single receiver agent, hence the agent in control knows both intent and outcome immediately. Information in the push scenario is delocalized: the initiator is not the affected agent, and the outcome is not automatically known to the initiator. The receiver may be unable or unwilling to accept the imposition.

• *Can failures be repaired?* A pull agent has full information about what resources it is missing to keep a promise, so it can attempt to acquire this from any agent that can keep such a promise. A push agent does not know what is missing from the recipient agent, so it can only shoot resources blindly[90].

6.7.3 SITUATION AWARENESS IN PULL AND PUSH

The question we seek to ask here is: to what extent are we are able to claim reliability of a push or pull method, within a system? We see that pull-based approaches have several advantage over push based methods. In section 6.7.5 we will also see how pull resolves consistency conflicts, or so-called split brain problems.

There is a fundamental motivational conundrum for push-based (imposition) controls: how does one motivate an external intervention in a system without prior reliable information? A random imposition could be a shot in the dark. Remote control thus has a bootstrap problem for the awareness of state.

In a local scenario, where information about system state is both available and trusted, the motivation for an intervention or repair is always present; but, in a remote control scenario, information is not close at hand. This means:

- We have to trust the source of information about remote state.
- Information has to travel quickly enough to be current.

- Information has to be separable from other remote effects in order for the remote agent to understand the possible consequences of an imposition (shot in the dark).

Thus the problem of motivating remote interventions is risky, the father away from the point of action one initiates intent.

Lemma 32 (Locality and the completeness of information)**.** *All information in a pull scenario is local to the recipient agent. Information in the push scenario is delocalized.*

This is self-evident as pull increases an agent's knowledge, whereas push reduces it by introducing new information that is unknowable without delayed cooperation.

6.7.4 THE RELATIVE STABILITY OF PUSH AND PULL

Consider the representation of a single atomic agent making promises to push and pull influence (e.g. data, material, etc) in figure 6.14. The characteristics of these

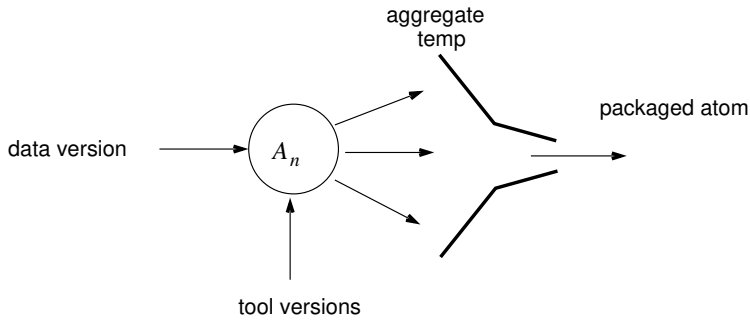


Figure 6.14: Push and pull methods in relation to a single agent. The geometry of a push, from sender S, suggests dissemination, amplification and delocalization of influence. The geometry of pull, to receiver R, suggests localization and resolution by selection from alternatives. Note the direction of intent is marked by the promise arrows, while the direction of service flow is indicated by the downward arrow.

arrangements seem to have properties that are a natural fit for their topologies. The topology of a push, from sender S, suggests dissemination, amplification from one to many and delocalization of influence. The topology of pull, to receiver R, suggests localization and the ability to resolve multiple influences by personal selection from many alternatives to one result. The opposite scenarios also warrant attention (see section 6.7.5). Pushing from many to one would result in conflict at a point, but pulling from one to many is a common form of dissemination sometimes called publish-subscribe[91].

Comparing the left and right hand characteristics, a number of things is immediately apparent. The left hand side shows a push method, in which the service source S pushes out by imposing b onto some number of recipients. The dotted promise lines indicate the alternative promise approach. The right hand side shows upward promises from the recipient to pull $-b$ from one or more sources. In both cases the diagrams are shown from a single autonomous agent, in other words, what we are comparing in the diagram with what a single agent can accomplish without cooperation.

Some more characteristics are summarized in table 6.1.

POLARITY	NOTES / CHARACTERISTICS / ASSOCIATIONS
PUSH	BRAIN MODEL, FLOODING, BROADCASTING SINGLE SOURCE (VERTICAL SCALING) SPONTANEOUS IMPOSITION (SURPRISE), FLOW-DRIVING, AMPLIFYING, ASSUME SUCCESS, NO FEEDBACK, DIVERGENT NON-LOCAL OUTCOME—UNOBSERVED BY INITIATOR (DECALIBRATION)
PULL	SOCIETY OR CONSUMER MODEL, SUBSCRIBE (FROM PUBLISHED) MULTIPLE REDUNDANT SOURCES (HORIZONTAL SCALING) PLANNED PROMISES (PREDICTED), COUNTER-FLOW, AGGREGATES, SELF-CONFIRMING OUTCOME, IMMEDIATE FEEDBACK, CONVERGENT, LOCAL OUTCOME—OBSERVED BY INITIATOR (CALIBRATION)

TABLE 6.1: COMPARING GENERIC ATTRIBUTES OF PUSH AND PULL.

Comment 10 (Confirmation of outcome)**.** *When agents provide a confirmation of outcome as a service, e.g. return codes from software functions, they do so as an additional promise, that may or may not be kept. Confirmation does not necessarily improve our knowledge, because we have to trust the information. If an agent is unreliable in keeping a promise, why would we expect it to be any more reliable in reporting the outcome?*

6.7.5 RESOLVING CONFLICTING DEPENDENCIES (SPLIT BRAIN PROBLEM)

Let's return to the issue of conflict resolution, or how to assimilate redundant promises consistently. In imposition systems and push systems, a major source of problems is inconsistency, as there is contention over shared resources (see figure 6.15). One way to avoid the problem of contention with impositions is to ensure that impositions only cause

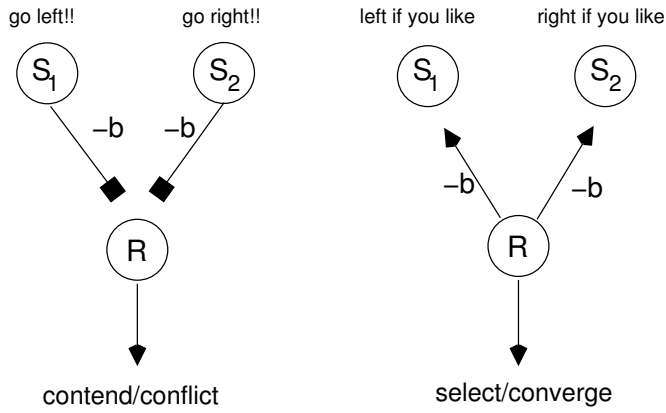


Figure 6.15: In the imposition case, one of the agents has to be at odds with its intent. In other words, expectations cannot be met, and the outcome must fail unintentionally. In the promise case, all expectations can be met and all agents have complete information about the expected outcome. Any deviation is now due to non-intended issues.

divergent outcomes. If each imposition lives entirely independently, within a private context, there is no problem (see figure 6.16).

Example 138 (Private world consistency)**.** *In information technology, client-server systems, where resources are pushed to a single queue, or which potentially in conflict with shared resources, often 'fork' their processes into privately segmented workspaces. These erect separate agencies where there is no contention. If the segmented threads still need to work with a shared resource, locks requests are used to force the imposition processes into a requesting 'pull' mode of operation.*

In the example above, a principle is used to escape from contention, which is well known as a resource lock in IT systems. Think also of the lock on toilet stalls which serializes access to the shared resource by multiple users. We make sense of contention in shared resources by having a single agent select a result from a collection of alternatives. This is the transition from the lefthand side to the righthand side in figure 6.14. Agents can avoid many of the problems associated with push or impositions by reversing their thinking to a service oriented subscription access approach.

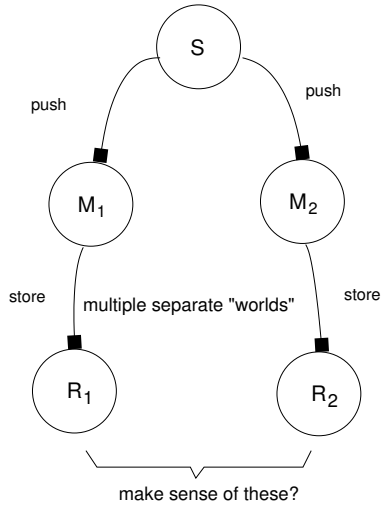


Figure 6.16: If influence is pushed redundantly into completely separate spaces, there can be no conflict or doubt within each separate instance of these 'many worlds'.

Comment 11 (From push-commit to publish-subscribe)**.** *For essentially cultural reasons, the push method is ubiquitous in all forms of technology, as well as in society. We like to treat the world as if it were an extension of our bodies. However, publish-subscribe methods (otherwise called 'pull') are becoming increasingly popular, as they resolve many issues (especially for web scale), such as allowing tolerance of inconsistency, or so-called 'eventual consistency'. It is often argued that we must have consistency, but often this is overstated, and the regions over which consistency is needed are much smaller than systems generally try to apply them over. Pull methods allow consistency on a need-to-know basis.*

6.7.6 SCALING DIFFERENCES BETWEEN PUSH AND PULL

As we have seen, many of the differences commonly used to characterize system architectures are only figments of scale. This should tell us that a single design will not scale a priori with all promises and expectations intact. A system design might need to undergo a phase transition in its mode of operation to cope with scaling effects. Consider these two possibilities.

- Clients push all work to a serial load balancing service, which dispatches the work to a horizontally scaled battery of discrete servers. This approach is convenient

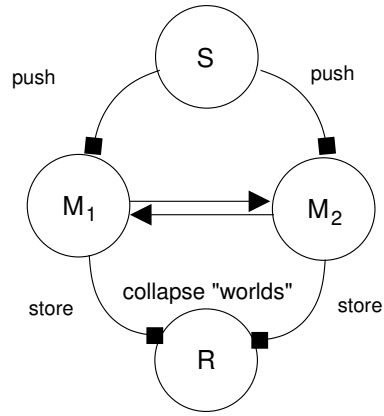


Figure 6.17: Conflict and inconsistency can result when we collapse independent worlds into a single picture. We do this when we average data in statistics. The arithmetic mean is an algorithm for resolving the inconsistencies of independent measurements. Voting or quora are another approach for odd-numbered collections of outcomes. In each case, resolution is obtained when the observer chooses (pulls or accepts with a (-) promise) a representative value from the values offered.

when traffic levels are sparse and unpredictably distributed.

- A server polls all known clients for possible work, and pulls the workload at its own behest, thus managing its internal resources optimally. This approach is measurably superior when traffic levels are high and are predictably distributed[BB08].
- Hybrid approach: clients publish their promise of work to a centralized dispatching directory service (acting in the role of indirection pointer), that promises clients availability as a list of appropriate discrete server queues with available capacity. None of the workload load is sent through the dispatcher bottleneck, only the notification of a promise. The actual server agents now pull work directly from the clients at their own behest.

Why do these different approaches exist? The first approach is a classic event driven 'give me what I want now' deterministic approach (push the problem onto the server, abdicate responsibility, and hope for the best). The second approach is a nicely cooperative and tastefully respectful approach when all the clients have promised their predictable work schedules up front, but it is inefficient when there are many clients with only sparse stochastic workloads, because the poller wastes a lot of time finding nothing to do instead

of going directly to the work. Clearly the ability to handle workloads is different for sparse events and predictably scheduled workloads. Why should this be? The answer is that locality of information leads to predictability, but that localization of workload leads to traffic intensity instabilities. So we need to ask where the relevant information and work are located and make sure the flows respect the capabilities and capacities of the agents in the system.

A promise theory perspective is helpful, as it reminds us that a client-server relationship has two sides, a client and a server, both of whom make promises essential to the completion of a cooperative transaction. Nothing happens without a promise from a server agent:

$$\text{SERVER} \xrightarrow{\text{ready to accept work}} \text{CLIENT} \tag{6.146}$$

This promise invokes the opportunity to provide clients with information about what kind of capacity it has, and how much spare capacity it has currently. Along side the server promise are the clients' choices: to either promise a predictable workload or to impose without prior warning:

$$\text{CLIENT} \xrightarrow{\text{take this work!}}\blacksquare \quad \text{SERVER} \tag{6.147}$$

$$\text{CLIENT} \xrightarrow{\text{please consider my work}} \text{SERVER} \tag{6.148}$$

The server clearly has more relevant information when the clients promise what is coming up. A smart server can use this information to optimize its resources, cost effectively, in space and over time. In simplistic terms with boils down to the appropriate usage of 'push' and 'pull' techniques, or promise and imposition.

Polling a set of incoming queues for work is a reliable way of controlling input. Random sparse events can be noticed more quickly when the initial receiver can notify a handling agent of their location. This is the approach behind 'push notifications'. We can ask, when are promise and imposition approaches best suited to an expedient handling of the work? The answer to scaling issues, of course, depends on the dimensionless ratios of the problem: the traffic intensity, filling fraction, or utilization of the queues involved (see figure 6.18). When traffic is sparse, the cost of searching, for activity to pull, exceeds the inconvenience of dealing with impositions from exterior agents. When the volume of requests is high, no searching is needed, provided all work is equivalent, and the server can expect to find work by simple polling. This also allows it to promise its own fair weighting policy to the clients, independently of what the clients attempt to impose. Thus imposition is helpful when traffic is low, and becomes problematic for the recipient at high traffic levels. Anyone who has received too much email at some time will have experienced this for themselves.

The classical approach of impositional queues, backed up by a load balancer (see figure 8.5) combines multiple message queues into a single server queue, and leads

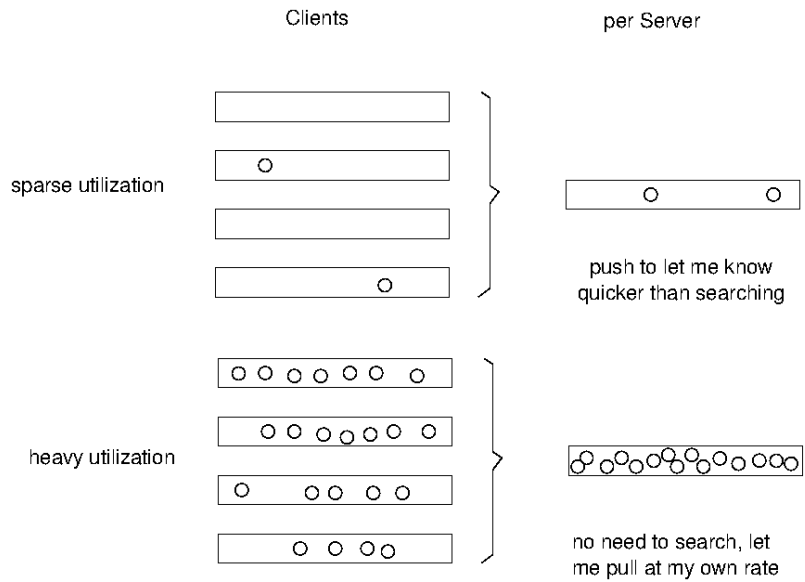


Figure 6.18: When traffic is sparse, the cost of searching for activity to pull exceeds the inconvenience of dealing with impositions from exterior agents. When the volume of requests is high, no searching is needed, and the server knows it can expect to find work by simple polling. Thus imposition works best when traffic is low, and becomes problematic for the recipient at high traffic levels.

to an imposed bottleneck that is beyond the control of the server. It compromises the server's autonomy, and becomes problematic when the server's queue approaches its instability threshold. On the other hand, keeping a lot of spare capacity around, on the off chance that it might be needed for peak traffic, is wasteful and motivates the search for an approach that exploits economies of scale[92]. The key question for the scaling of a system, which preserves stable semantics (the principal usability criterion), is thus the avoidance of a sudden first order phase transition in the performance of the agents comprising it.

The hybrid method mentioned above leads to an adaptive compromise to this dilemma. Without a change from push to pull for the bulk of workload, a serious queueing instability lies latent and immune to horizontal scaling at the dispatcher. On the other hand, without the ability to give notification hints of workload at times of sparse traffic, much time and effort can be wasted in searching for clients. This is also the thinking behind self-service models (see section 6.8.2).

Example 139 (DNS based load balancing)**.** *DNS based load balancing approximates*

the hybrid approach described above. It uses its knowledge of servers to perform a simple round robin or distance based allocation of addresses when a client attempts to look up a server. Clients don't push their entire workload to DNS, as they do to a serial load balancer, so bottlenecks can be avoided by smart point to point routing. This is a common approach in horizontal scaling of content delivery networks (CDN).

Example 140 (File system change notification)**.** *File system change notifications were introduced into Linux systems to allow clients dealing in large amounts of data to notice the location of updates, without having to scan an entire filesystem for changes. This is a key strategy for speeding up data synchronization.*

If customers need deliveries every week, predictably, there is no need to send impositional orders that flood the delivery service: the service can serve the clients by predictable polling. On the other hand, if one is in the mode of searching for sparse business, it helps to have a service portal where impositional clients can present their needs quickly.

Small notifications can be pushed to accelerate the routing of the bulk work channel to a suitable server. This methods becomes ineffective as the traffic level of notifications rises to the queueing threshold of the recipient.

Example 141. *In a service oriented model, as compared to a vertically integrated monolithic or centralized model, scaling patterns of push and pull get translated very visibly into costs, because the vertically integrated single server model is the VLSI*[93] *equivalent of an easily replicable packaged commodity. The separated model of a collection of weakly coupled service is a free-market approach. The work of connecting the services together and securing cooperation between them grows*

However, it is worth pointing out again that the scaling of different semantic resource attributes may be multi-dimensional and involve conflicts of interest, in the manner of multi-strategy game optimization.

6.8 SERVICES

A design paradigm that has become quite popular over the past 25 years is that of services. How is this different from any other? It is quite promise oriented.

6.8.1 SERVICES DEFINED

See sections 3.10 and 4.4 of [BB14a]. The difference between promises are impositions is important. It describes whether an agent is expecting some influence from an outside

agent, or if it arrives unannounced. In this document, we can label agents in the roles of client and server. Clients and servers are *roles* that are defined as agents who make certain collections of promises.

Definition 136 (Opportunistic Client)**.** *An agent who requests r_X a service X by imposition (without having promised).*

$$C \xrightarrow{r_X} \blacksquare \quad S \tag{6.149}$$

$$S \xrightarrow{+X} C \tag{6.150}$$

$$\tag{6.151}$$

The server S may or may not promise to accept an imposition from C.

Definition 137 (Expected Client)**.** *An agent which has promised in advance that it will send requests r_X (perhaps as part of an agreement), given the existence of a promise of service X:*

$$C \xrightarrow{r_X|X} S \tag{6.152}$$

$$S \xrightarrow{+X} C \tag{6.153}$$

Definition 138 (Open or opportunistic service)**.** *A promise made by a server agent S to provide a service to clients, who may request r_X by promise or imposition,*

$$S \xrightarrow{+X|r_X} C \tag{6.154}$$

This is meant to be a take it or leave it service, open for any client impositions without prior agreement. There is unrestricted access, attempting to please everyone, in a very trusting way. In a closed service, trust is held more closely to prior promise relationships:

Definition 139 (Closed or expected service)**.** *A closed collection of promises to accept requests r_X and deliver X in response, made by a server agent S, describing a steady state, in which no impositions are accepted.*

$$C \xrightarrow{r_X|X} S \tag{6.155}$$

$$S \xrightarrow{-r_X} C \tag{6.156}$$

$$S \xrightarrow{+X|r_X} C \tag{6.157}$$

$$\tag{6.158}$$

Note that the notion of a service requires us to know about the client's behaviour, so a service is not really separable from the promises of both parties (client and server).

Comment 12 (Services and time)**.** *Service is a relationship which accumulates over time.*

The timescales for completion:

$$T_{user} \simeq T_{client} \simeq T(r_X) \gg T_{server} \simeq T(X) \tag{6.159}$$

6.8.2 SELF-SERVICE VS SERVE-TO-ORDER

It is implicit in the idea of a service that service is performed by one agent for another. So what do we mean by self-service? We usually mean that the completion of the service is driven by the client, at the imposed tempo of the client, facilitated by the server but not driven or apparently 'triggered' by it i.e. the server is decoupled from the client somehow ('help yourself'). This is not as clear a concept as it seems. If there were complete decoupling, it would not be a service interaction at all. Self-service still involves two parties.

As a starting point, we can contrast self-service with simply autonomous behaviour:

Definition 140 (Self-driven (autonomous) behaviour)**.** *An promise or imposition made by a single agent A without reference to any other agent's promises.*

$$A \quad \left\{ \begin{array}{ll} \xrightarrow{X} & ? \\ \xrightarrow{X}\blacksquare & ? \end{array} \right. \tag{6.160}$$

Lemma: It is simple to prove, from these definitions, that autonomous behaviour is not a service, because the agent A has all of the capabilities it needs internally. It does not refer to any other agent.

By saying 'self-service', we expect there to be an external agent, which provides something that an agent can opportunistically help itself to, but where the client drives the process completely, with total autonomy, and no obstacles or hindrances[94]. Thus, self-services presumes the existence of a promise to serve the client in advance, with no special negotiation.

In other words, the server plays no role in the ability to keep the service promise, other than providing perfect service on demand (a kind of guarantee). This is not a promise that can be kept in reality, so it can only be an approximation. This is a shift towards commoditization:

Definition 141 (Self-service (heuristic))**.** *The selection of pre-arranged, prefabricated, or commodity outcomes, by a client from a service that merely caches outcomes.*

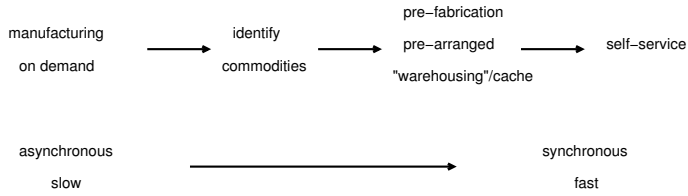


Figure 6.19: The evolution of bespoke service to self-service is a transition to commoditized synchronous delivery.

This is a shift from bespoke functionality, e.g. manufacturing on demand, to user-selection from a menu of commoditized, pre-arranged offerings. It is a decoupling of intent from process, by contrast with 'serve to order'.

We can explore the implications of this for promises:

Definition 142 (Self-service (technical)—low latency synchronous selection)**.** *Behaviour that approximates self-driven autonomous behaviour. The request r_X by a client agent C is now dependent on a menu of pre-promised outcomes of a service X made by an external agent S without further agreement:*

$$C \xrightarrow{r_X|X} S \tag{6.161}$$

$$S \xrightarrow{X} C \tag{6.162}$$

The server promise is effectively unconditional, implying no delay in delivery. The time to complete the service $T(X) \ll T(r_X)$ is much less than the time to complete the client's task, making it essentially instantaneous. This preserves the illusion that the client is entirely responsible for causing the outcome r_X. This is often interpreted to mean a low-latency synchronous service X.

Example 142 (Shopping at a commodity store)**.** *A client goes to a supermarket, where goods are promised $+X$. The goods are pre-ordered and available, open to browsing on trust, and buying them takes a negligible amount of time compared to ordering them remotely by an agreed transaction. The client experiences a self-service environment.*

Example 143 (Web and cloud computing)**.** *Cloud service providers offer a simple 'self-service' interface for rentable computers. The availability of the computers is promised in advance, allowing the client to accept the promise and make use of it. Similarly, web pages are pre-fabricated and users consume them on demand at their leisure.*

Example 144 (Tenancy, e.g. shared parking)**.** *Any rentable entity may be offered as self-service promise. If, unoccupied, parking spaces (with meter) may be empty. On a lot of rentable vehicles, a client could swipe a card and rent the vehicle immediately, rather than having to pre-order with a delivery time.*

Comment 13 (Self-service is an illusion maintained by speed (separation of timescales))**.** *Self-service is an illusion, because the service is being provided approximately continuously, by virtue of the condition $T(X) \ll T(r_X)$. Imagine going to a supermarket and 'helping yourself to the goods'. The goods do not come from within yourself, so the service is provided by the supermarket. If they close the shop, you cannot help yourself anymore. So the whole illusions rests on the availability of the service, represented by the time scale condition. The promised 'goods' have to be there 'just in time', else it doesn't work.*

What Promise Theory reveals is that self-service is not a realizable guaranteeable thing, unless it refers to autonomy, i.e an agent has everything it needs internally without requiring the assistance of another agent.

A *service to order* reverses the order of prerequisites. No service is prearranged until a request for service has been made, by promise of imposition. The client is no longer in control of the waiting time for service delivery—the service provider is now the bottleneck.

Definition 143 (Service to order)**.** *A request for service, by a client C, precedes a promise of service delivery by the service agent S:*

$$C \xrightarrow{+r_X} S \tag{6.163}$$

$$S \xrightarrow{-r_X} C \tag{6.164}$$

$$S \xrightarrow{+X|r_X} C. \tag{6.165}$$

Since there is no pre-fabrication of the outcome, this typically takes longer than a self-service, commodity promise. A client may promise its request (by advertising it like a publish-subscribe system, shared to a number of potential service providers to bid or supply) or it may impose a request on a specific agent 'out of the blue'—or any level of invitation in between these possibilities.

Example 145 (Built to order)**.** *When ordering a car or a new suit from the factory, with bespoke options and measurements, the service interaction is a longer series of interactions in which the user has to wait for the service provider to keep its promise.*

Example 146 (Just In Time delivery)**.** *This blurs the distinction between self-service and service to order. Nothing is pre-fabricated, but the timescale for delivery is minimized to be virtually synchronous. This is sometimes called lazy evaluation.*

6.8.3 MASTER AND SLAVE ROLES

The terms master and slave are promise roles, often used to characterize agents in a service relationship.

Definition 144 (Master agent)**.** *An agent considered authoritative with respect to a service: a single point of calibration that defines a notion of correctness. A mater agent is at the top of a hierarchy of dependence. A master agent does not depend on any other to define the outcome of its promise.*

It is more correct to refer to a master role in relation to a particular promise, because an agent with several roles could be be a master for one service and a slave in another.

Definition 145 (Slave agent)**.** *A slave agent is one that acts as a proxy for the intent of a master agent. In other words, it depends on the promised advice of a master agent to define the outcome of a promise.*

If an agent's sole purpose is defined by its dependencies, it is a slave, because it has no independent purpose. It has essentially one and only one life story, and it lives or dies by these dependencies.

Example 147. *A memory cache is a slave.*

6.8.4 WORKFLOW LOGISTICS: PIPELINES VERSUS SERVICES

Workflows and pipelines form the basis of the modern manufacturing ethos. A workflow forms a Directed Acyclic Graph (DAG). We send commodity components into one end of a process, described by a promise graph, and some product emerges at a pickup location on 'the other end'. Logistics, end-to-end delivery and data processing all follow this basic pattern[BB14a, Bur18].

We understand a workflow, or processing pipeline, to be a collection of agents that form several stages of a larger outcome, and where each stage depends on the outcome of a previous stage. The outcome is cumulative in some sense, and each outcome is ultimately parameterized by a history, log, or journal of what promises were kept throughout (see figure 6.20). .

A service, on the other hand, is a transactional singleton, operating in a steady state (though its promises may not be stateless), in which responses are simple functions of queries (the intent is supplied by the conditional query).

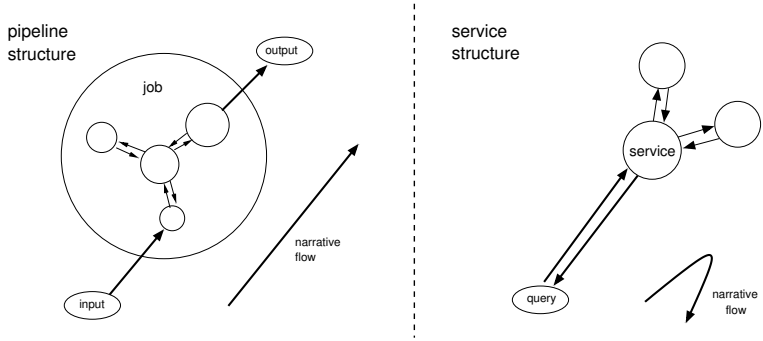


Figure 6.20: The topology of workflow or pipeline is implicitly a set of interior interactions, subordinate to a single task, which is represented by a superagent. A service, on the other and, stands independently with mainly exterior interactions.

A service performs a function continuously in response to exterior interactions. A job may be shared and coordinated between several agents, but it is unified semantically by having a singular outcome of its own rather being defined by its conditional input (see figure 6.20).

- A service accepts input and returns an outcome to the initiator of its input.
- A pipeline accepts input and returns an outcome to different agents.

The distinction between these is subtle: a pipeline can perform a service, and a service may involve pipeline processing. For example, a compilation service involves a pipeline of dependent promises whose outcome is fixed in relation to the types and conditions of inputs rather than the specific inputs.

We can define a pipeline by its agent structure. The entire job is a superagent, that may encapsulate various stages in the structure of a DAG (see figure 2.4).

Definition 146 (Workflow pipeline)**.** *A cooperative process, encapsulated as a single superagent, composed of interior sub-agents that form a directed acyclic graph. Each interior agent that forms part of the path from input to output has a* transducer *role[BB14a].*

The single exterior promise of the pipeline superagent is to produce an output whose final target state is a transformation of the input. The interior structure of a job is usually partially ordered, and entirely subordinate to keeping of this single exterior promise.

6.9 COMMUNICATION NETWORKS

As a straightforward but extended example of the promise method, we can examine network communication. This discussion is based on the paper originally written for [BBCD14]. Networks are countable systems that pass discrete packets of information. They may not may not be deterministic or conservative, as networks do drop and duplicate packets at interaction nodes. In the worst case, errors may lead to packets being corrupted or echoed, so we have to expect networking to be non-deterministic.

6.9.1 PROTOCOLS AS PROMISE RECURSION

Protocols are promises about processes or communications. Processes or messages are assembled from building blocks that combine to build standard structures.

Definition 147 (Protocol)**.** *A convention or pattern describing non-atomic interactions, that classifies them into families by similar structure or intent.*

The structure of a message or process may therefore be thought of as a promise, but made by what agent? There are in fact several kinds of agents of different roles at work in message transmission. There are, of course, the sender and the receiver of the message. These are two agents that promise the message as an entity. However, the message itself may also be considered a stream of agents that are embedded within the 'agency of the message' in full, each promising a certain kind of content, and each acting quite independently of the transmission of the promise. The message components thus have independent agency, and meaning, within the context of a message communicated between another kind of agent.

In Promise Theory, each independent part of a *message* may itself be represented as an agent that makes promises, embedded within the larger agency of a total message. This is sometimes call layering of protocols. The promise to send and deliver a message is a separate promise from message format and content.

Protocols can be very fragile structures, as with any rigid pattern. At the highest level, an error can be made by the sender in breaching the protocol's promised pattern (+), or by the recipient in failing to parse it correctly (-). At lower levels, there is also scope for mode detailed error:

- Content error: size type and content mismatch.
- Order of prerequisites breaches the standard.
- Misalignment of component boundaries leads to confusion. subagents within the message should be within promised bounds.

Error correction, or fault tolerance, methods may be applied to avoid transmission problems. There might be a significant number of promises at work in a protocol that need to be addressed:

- Recursive agency[95] or context referring to the protocol's alphabet and dictionary.
- Data type assumptions, subagent encodings, size and alignment.
- Compound clusters and groupings (recursion of grammar).
- Interpretation of data as a 'relative delta' or 'desired end state' representation.
- Sequence number or conditional element in an on-going conversation.
- Attribute advertisement (the + promises).
- Attribute matching (the - promises).
- Missing or corrupt constraint (promise not kept).
- Version of the pattern, if the pattern is evolving

Any one of these promises broken could lead to a fault in the system at the scale of the communicating end-points.

6.9.2 ETHERNET

The agents that keep promises to send and receive data are the network interfaces. For example, in the Ethernet protocol, interfaces E_i promise to label transmissions with a unique MAC addresses or string of digits.

$$E_i \xrightarrow{+\mathrm{MAC}_i | \mathrm{MAC}_i \neq \mathrm{MAC}_j} E_j \quad \forall i, j$$

When data are transmitted by an interface, the interface keeps its promises to use messages that have (destination MAC address, data). Note: the message is not a promise, the promise governs how the message is handled.

$$E_i \xrightarrow{(+\mathrm{MAC}_j, +\mathrm{data})} \blacksquare \quad E_j$$

Messages are sent 'fire and forget' as impositions on to a remote receiver. While all interfaces generally promise to accept any MAC address, (unless they block with MAC access control) only the interface whose MAC address matches the destination in the message doublet actually promises to accept the message voluntarily. Note, there is

nothing other than convention to prevent all agents from accepting the data too; this 'promiscuous mode' is used for network monitoring, for example.

$$\begin{array}{lcll} E_j & \xrightarrow{-\mathrm{MAC}_j} & E_i & \forall i,j \\ E_i & \xrightarrow{(-\mathrm{MAC}_k, -\mathrm{data}) \text{ iff } (i=k)} & E_j & \end{array}$$

Since the channel is unprotected, agents effectively promise the data to all others in scope. Moreover, all agents promise to decode the address and the data, but many will discard the results.

While this set of promises is scale independent, the assumption that every agent has to be in scope of every transmission does not scale, since it requires messages to be flooded or broadcast to every node (agent), in principle. The primary issue with raw Ethernet is that there are no ways to selectively limit the size of these broadcast domains. This makes the 'everyone please take a look at this' approach impractical.

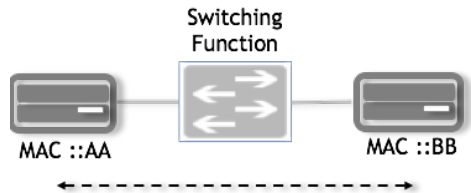


Figure 6.21: An Ethernet switching function.

In Fig. 6.21 we see two interfaces that promise MAC address 00:00:11:11:11:AA (shortened to AA) and 00:00:11:11:11:BB (shortened to BB). Suppose we wish to send data from AA to BB, then, since the Ethernet is a push-based imposition protocol, only half a contract is needed for emergent delivery, and we leave the rest to trust.

$$\begin{array}{lcll} E_{\mathrm{AA}} & \xrightarrow{+\mathrm{MAC}_{\mathrm{BB}}} & E_{\mathrm{switch}} & \\ E_{\mathrm{switch}} & \xrightarrow{-\mathrm{MAC}_{\mathrm{i}}} & E_{\mathrm{i}} & \forall i \\ E_{\mathrm{switch}} & \xrightarrow{+\text{forward MAC }_{\mathrm{BB}}} & E_{\mathrm{BB}} & \end{array}$$

In each point-to-point interaction, the agent has to formally promise to use (-) the delivery service promised by the agent giving (+). This is the algebra of binding. There is no notion of a permanent virtual circuit, as say in ATM. However, if we add handshaking, a similar story can be told about ATM, Frame Relay, MPLS and other systems.

6.9.3 INTERNET PROTOCOL

IP provides Wide Area Networking by issuing two part addressing to cope with transmission scalability. IP addresses still promise to be globally unique, but are interpreted as doublets.

(network prefix, local address)

Only addresses with the same prefix are considered in mutual scope for broadcasting, and messages addressed from one prefix to another promise to be forwarded deliberately rather than by 'flooding'. IP is thus a cooperative effort that builds on promises rather than impositions alone.

To make this work, IP needs two kinds of agent, which fall into different promise roles (see figure 6.22): *interfaces* (terminating connections), which only transmit and receive data intended for them, and *forwarders* (called routers or switches) that cooperate with multiple interfaces, and promise to selectively forward data from one interface to another between protected broadcast domain. This acts as a flood-barrier or firewall to packets promised to different prefixed networks.

To model routers, without giving up the interface abstraction, we introduce the concept of a route service (or link service), whose job it is to establish cooperative forwarding between the interfaces.

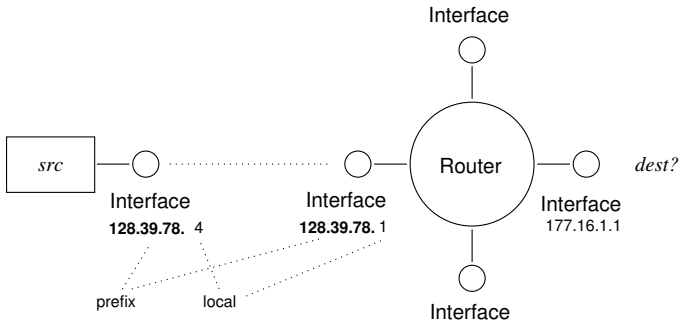


Figure 6.22: Internet promises. An end-node or leaf and its single interface promises to relay through a 'router' which is surrounded by multiple interfaces, thus connecting multiple network branches.

Consider Fig. 6.22. The source node has an address, normally written 128.39.78.4/24. As a doublet, the promises see it in two parts as $i =$ (prefix=128.39.78, local=4). We'll call this the source prefix, or, $j =$ (prefix=128.39.78, local=1) for the router interface. When a message is sent to an address with a different destination prefix, data are sent

by imposition to the interface on the router with the source network prefix (usually the 'default route'):

$$I_{source_i} \xrightarrow{+(\text{destination},\text{local}),+\text{data}} I_{\text{router}_\text{j}}$$

Each router interface j promises the connected source interfaces i to use all such packets, a priori, and to present them to the router (kernel) which keeps the following promises.

$$I_{\text{router}_\text{j}} \xrightarrow{-(*,*),-\text{data}} I_{\text{source}_i}$$
$$I_{\text{router}_\text{j}} \xrightarrow{+\text{prefix},+\text{data}} \text{Router}$$

Similarly, other interfaces connected to the router's interfaces promise to accept messages from the router that have their prefix:

$$I_{\text{source}_\text{i}} \xrightarrow{-(\text{prefix},\text{source}),+\text{data}} \text{Router}_\text{j}$$

Crucially for messages to escape from a local region, the router promises all IP interfaces to forward messages it receives on one if its own interfaces according to a set of promises which we denote 'forward'. The router interfaces, in turn, bind to this promise by accepting it.

$$\text{Router} \xrightarrow{+\text{forward}} I_{\text{router}_\text{j}}$$
$$I_{\text{router}_\text{j}} \xrightarrow{-\text{forward}} \text{Router}$$

The forward promise has the following logic:

> (1) If the prefix of the destination interface is the same as the prefix of one of the router's interfaces, forward the message onto that interface.

The remainder of the promise requires configuration with knowledge of the wider world.

> (2) If the prefix of the destination interface is known to an internal database of external knowledge, i.e. the Routing Information Base (RIB), forward the message to the interface known to lead to the desired destination.
>
> (3) Send all other message destinations to a pre-decided default interface, where we expect to reach some other router with greater knowledge of how to find the prefixed network.

Note that, like the Ethernet, this algorithm has only emergent behaviour that matches its design goal. It cannot, by direct imposition, assure a successful delivery of messages, because that requires the cooperation of potentially many intermediate interfaces and routing agents. In spite of this apparent lack of control, the Internet works demonstrably well. Trust plays a major role in operations.

6.9.4 VLAN: L2 CHANNEL CONTAINMENT

The concept of doublet addressing in IP enabled improved scalability, by black-boxing local networks, but added the cost of routing. How expensive routing is, in relative terms, is a constantly changing overhead that depends on many current technological factors. Fear of this cost tends to make datacenter traffic favor L2 solutions.

Routers were optimized for WAN delivery, so the obvious question for LAN managers was: could routing be simplified for smaller local regions without the paraphernalia needed for global routing? When packets don't have to be routed through multiple hops, i.e. when parts (2) and (3) of the forwarding promise can be ignored, a simpler form of prefixing can be used. This is the concept of the VLAN overlay. Interfaces can simply be classified, or tagged with short integer labels:

$$\text{(prefix, local address)} \rightarrow \text{(VLAN-id, MAC-address)}$$

Then we have multiplet address components again (but now with a short VLAN tag instead of a large integer prefix), for boxing off local regions. A VLAN tag signifies membership in a private logical container. As with Frame Relay, these tags have to be configured manually, so routing is a human-centric process.

The concept of a Level 2 overlay has become quite popular for its perceived simplicity in small isolated networks. It's limitations have to do with scaling of the manual configuration and broadcast domains. VLAN is a brute force routing mechanism that scales linearly with the number of addresses in a container. Containers are not localized in physical space, only in logical channel space (unlike the assumed distribution of prefixes in IP). Thus this does not address the issues of physical scaling. However, we need something that scales like $\log N$ or better (like IP).

6.9.5 VIRTUAL CIRCUITS

Virtualization of networks is a common approach to making better use of resources. In the same way we create multi-user sharing (multi-tenancy) for services, using timesharing kernels, a network operating system can perform the same sharing for internetwork communication. Virtual users need the same quasi-static promises to function as physical users: they need to have a stable identity and a stable approach to reachability. Indirection (pointer tables) allow virtual agents to maintain a fixed identity for distinguishability and reachability.

Address lookup is the basic service that enables unique *semantic* identities, i.e. names, to be mapped to unique metric identities (addresses), and distributed service processes (routing) allow context-dependent paths to be computed to route communications along virtual circuits.

Some virtual circuits formally create a persistent or even static path associated with a connection (like a client-server 'session') (MPLS, ATM, Frame relay). Others set up virtual circuits at a higher layer of abstraction (like TCP). In the case of TCP, the computation of paths from one address to another can be automated as long as addresses have global significance and are fixed. In the same of purely local addressing (e.g. Frame Relay, Ethernet, etc), circuits that span broadcast domains have to be wired based on external information by imposition of a topology—there is insufficient information promised by the agents in the network to compute paths without additional 'godlike' intervention.

In cloud computing, agents are ephemeral, often with short lifetimes from seconds to minutes. Network addresses, which are usually IP based at all layers, can only bind to agents for a short time, making traditional routing impossible. One needs separate address lookups and name-address directory rewriting in real time. This is a potentially unstable situation. The promises are thus short-lived, leading to a gaseous state for the agents.

There are several weaknesses to current approaches: many pragmatic 'quick and dirty' tie clusters to geographical regions or physical constructs like datacentres or even racks of computers. As a rule, network technologies have been designed with physical rather than virtual localization in mind, so more layers of virtualization are needed to sew together physical networks and overlay them.

Example 148 (Cluster broadcast networking)**.** *Some approaches to network virtualization create local private addresses, which cannot be routed. This includes use of the IPv4 address ranges 192.168.xxx.yyy and 10.xxx.yyy.zzz, which standard routers promise to treat as private and non-routable. In this case, name-address binding discovery can be performed by broadcasting (as in Ethernet and ARP) or by registration (as in IP and DNS).*

6.9.6 TUNNELLING ADDRESSES AND TRANSDUCER PATTERN

Embedding protocols inside one another is not the only approach to containment. One can also strip off and repackage data on different legs of a journey. To do this, one makes a transducer that converts one kind of addressing into another (see [BB14b]).

ARP is one such service that maps between Ethernet MAC addresses and IP addresses. Instead of a physical forwarding table, a logical rewriting table is maintained. When a direct ARP conversion is not possible, data are sent to the default route, which is the address of the router interface I_{prefix} to the default interface. DNS is another transducer, that maps from symbolic addresses to IP addresses.

The same principle has been applied to isolated networks, such as the reserved names-

paces 10.0.0.0 and the example.com addresses 192.1.168.0/24. IP Network Address Translation (NAT) is now been promoted from crude workaround to viable technology, extending the local IP addressing component with additional internal addressing numbers, the rewriting outgoing addresses to point to the standard IP address range. End to end addressability is not normally promised in this scheme however, so it has limited value, (however see TRIAD[CG00]) for a viable scheme for extending IPv4 in this manner.

More recently, a tunnelling approach is also being used to artificially extend Layer 2 VLAN as a stop-gap measure for a technology users who are familiar with VLAN. VxLAN, and NVGRE are encapsulations of Ethernet L2 Frames, with tunnelling over IP to enable the physical reach across multiple gateways. Addresses add a multiplet component: a Tenant Network Identifier (TNI) or Virtual Tunnel End Point (VTEP) identifier embedded parallel channels.

These schemes perform two functions: i) they increase the number of possible VLAN-like channel addresses, patching a limitation in the VLAN implementation, and ii) they allow teleportation of broadcast domains across an IP scale network, transparently of routing concerns. Thus they do not eliminate the cost of IP routing, but offer a comfortable user interface for local network administrators.

6.9.7 ADDRESSABILITY WITH SCOPE OR NAMESPACES

By introducing multiplet addressing, we draw a logical (and perhaps physical) container around a network region which hides its internals with some kind of identifier or prefix, which acts as a namespace identifier. Everything inside the namespace is local and protected. There are two principles that explain these cases.

> *Principle 1: Container multiplet addressing*: Any system that promises to support n-tuple addressability of parts, for $n > 1$, enables logical or physical containment of information, as well as log-scalable routability between the containers. $\triangle$

To transmit data across multiple (possibly embedded) containers, we typically need an address component for each logical container. Thus interfaces a_i must promise to recognize one of the components a_i and pass on all others as passenger data:

$$\text{Interface}_\text{i} \xrightarrow{\pm(a_1,a_2,\mathbf{a_i},\ldots,a_n),\pm\text{data}} \text{Router}$$

e.g. the a_i address components might include MAC address, IP address, VLAN number and VxLAN IDs. This set of addresses need to be configured and managed, either manually or by some mapping service. Some of these addresses overlap (like IP-LAN and MAC addresses).

> *Principle 2: Forwarding by multiplet address*: Forwarding of multiplet addressed data requires an infrastructure of forwarding promises by each members of each container for each address component in which all other components are ignored by other containers as payload data. △

An interfaces a_i in a given container of level i would promise to accept other components addresses components as data only to be forwarded, not interpreted, i.e. as payload with no assumed semantics. In practice some of the address components might be removed or even rewritten, depending on the encoding as data traverse container boundaries, but that is not a requirement of the principle. All of the components have a continuing logical existence. It would be enough to ignore them. Note also that intrusion detection/prevention systems sometimes break the semantics of ignoring payload.

Example 149 (Addressable scalability)**.** *The scaling of multiplet addressing is a straightforward idea. It prevents a local namespace from becoming too large for flooding or broadcasting. The size of the namespace is limited either by a fixed number of nodes accessible in one multiplet address (e.g. VLAN tag, OSPF areas, BGP AS, etc), or equivalently, by the size of a prefix in a binary encoding of the multiplet (as in IP). In the first case, there is no defined limit to how many MAC addresses can occupy the same VLAN. Scaling is throttled by physical limitations. In the latter case there is an explicit quota tradeoff between local and global from a fixed number of addresses.*

If an n bit address has a prefix of length p, this improves scaling through black-boxification of local regions. It transforms the addressing of $N = 2^n$ things into the addressing of merely $N_C = 2^p$, things globally and $n_C = 2^{(n-p)}$ things inside each of the C containers. That is $\log_{n_C}$ rather than n_C scaling.

There is also no particular reason why IP addressing has to be limited to prefix quotas. IP Network Address Translation is an attempt to extend the range of local addressing, independently of the prefix quota space to alleviate IPv4 address depletion.

Example 150 (Two networking types)**.** *Networking supports two main use-cases:*

- Content delivery *or pull requesting (asynchronous retrieval promises of the form Node $\xrightarrow{X}$ Node).*
- Signalling *or push notification (synchronous impositions of the form Node $\xrightarrow{X}\blacksquare$ Node).*

The former is a many-to-one association, for which we can employ versioning, replication (data-model de-normalization), re-direction, and delocalization (e.g. Content Delivery Networks). Point to point addressing is less important; caching is highly meaningful. The concept of Name Based Routing has been proposed to abstract away endpoint addresses[BCA+12].

For the signalling, we still need endpoint-resolution addressability, as signals cannot be cached, though they might need to be flooded. Service delivery generally involved a mixture of these two cases, which depends on the nature of the application being supported. Applications typically want to make certain promises about connectivity, security, e.g. load balancing and firewall filtering options

Application-oriented delivery suggests other forms of containment based on the logic of the service interaction. Current networking management abstractions make application specific requirements painful to configure because of lack of a consistent model for abstracting them. This brings us to the present day.

Example 151 (End-to-end service promises via proxy)**.** *To see how this could be done, we return to Promise Theory. The 'proxy' or intermediate agent pattern was described in [BB14b] abstracts the end-to-end delivery promise of a service S through some promise to handle the delivery details by proxy P to a client. Both client and server may be guests running inside various containers.*

The abstraction we would like to expose to the user is for logical services and consumers to simply make promises directly to one another (Fig.6.23), without worrying about all the intermediate agents in between. The proxy pattern shows how this can be achieved. We refer readers to [BB14b] (section 11.3) for a discussion.

Examples of the service S could be: to provide connectivity over a secure channel, to grant or deny access to data, to commit to or retrieve from storage, to provide web transport. It is important to note that a go-between might create a superficial similarity of function, but it also adds four promises and hence four possible points of failure to the equation.

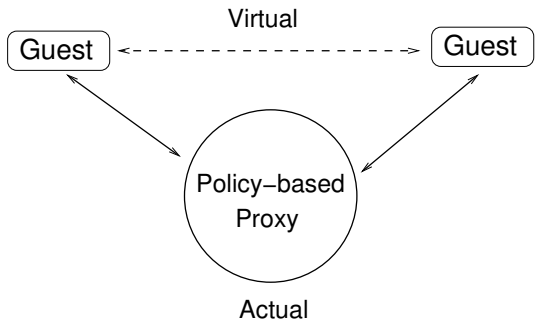


Figure 6.23: A generic proxy model that promises to mediate communication, with no internal details exposed..

In terms of the cell membrane analogy described above, we would say that the outer-membrane promises identify how others can communicate to that service, what they can communicate about, and what happens to the traffic when they communicate. Today, one would have to separate it into VLANS, Firewall rules, Load Balancing rules etc. These are details we would prefer to leave to a proxy on imposing on it a minimum of application specific requirements.

By providing this as a service, one is able to reason about applications, and how they interact with other applications rather than with the physical underlay. The membrane thus provides a level of abstraction that hides the details of the cell composition. This is how the Insieme architecture works.

By isolating promises as containers that promise to play certain roles in an application design, one can think about the datacenter as an organism. Then an organism comprises of many boxes containing multiple functions that manage their own resources based on a policy declaration.

6.9.8 MESSAGE QUANTIZATION AND JOB SIZE

In work batches, the size of a task matters. Frame or packet size plays a role in the dynamics of communication, by the interaction of timescales for the different agents' internal processes. Queueing theory suggests that short packets are preferable for agility and 'last-mile' distribution, in order to expedite packets quickly. It keep queues short and adaptable. But that could be a false economy if you need many more packets to finish the job. For mass transportation, there are economies of scale to be found by transporting in larger containers. A simple analogy can be to think about trains versus cars. Cars transport small packets of people in an agile and ad hoc manner. Their virtue lies in adaptability. Planes and trains, on the other hand, can transport numbers packets much faster, because they do not need the agility to take flexible routes. They are optimized for mass transport, and larger sizes are preferred to keep costs low. The data length of the protocol overhead is a significant cost. Efficiency of the process is improved by maximizing the payload per transaction, making the transaction longer[96].

CHAPTER 7

SCALING OF AGENTS AND THEIR PROMISES

So far, we've looked at small constellations of agents with only a few promises at a time. If agents are thought of like atoms, then we could call small clusters of them 'molecular systems': clusters of atoms that bring about new functional behaviour. In many cases, this is helpful already for deriving some insight into the workings of systems. However, the real benefits of formalization come when we look at greater scales.

Scaling of structures and influences is one of the techniques that physicists have used to address the generic patterns and broad economics of systems, both measured with energy and with money. The general rule, in a wide class of systems, is that the more you scale up, the more universal (and hence less specific) the results become. Sometimes one can characterize 'efficiencies of scale', in which costs scale more slowly than outputs. This chapter considers some of the ways in which scale plays a role in interacting agents—it's a theme that is ingrained throughout this volume, but I want to draw special attention to it here.

7.1 SCALE AND SCALING

A scale is an arbitrary approach to counting change, in space or time: the observable characteristics promised by an agent may change as we continue to resample the same agent (time) or as we move from agent to agent (space). The smallest discernable change denotes an atomic difference. A scale is a particular aggregation of that atomic difference. The more steps we aggregate, the large the scale. Aggregations of agents often effectively

make promises of their own at each distinctly separable scale—this is the lesson of complex systems and the subject of renormalization in physics.

Information science does not usually think of scaling in this way. Rather, it talk about 'scalability', or the use of parallelism to increase flow by increasing the size of a supporting system infrastructure. In other words, it concerns itself mainly with increasing dynamical throughput. This is is purely dynamical consideration; but, information rich systems, including but not limited to software, are dominated by the *semantics* of their interactions, so there is a pressing need to understand the scaling of semantics too[Bur14, Bur15a]. There are thus two meanings of scaling:

1. **Workload scaling of workload (parallel support)**.

   Scaling up a system refers to the intended increase in size or load bearing characteristics by increasing the number of agents, or by altering their promises. We compare systems based on whether they can keep a promise independently of the physical dimensions of the system, or the magnitude of the load.

2. **Universal scaling of agency (spacetime bulk)**.

   This scaling refers to a change in the way we probe or interact with a system, either in detail (small scale, short wavelength) or in bulk (large scale, long wavelength), and the effect that it has on what bulk promises are kept. This is sometimes used to make scaled models.

In 1. we are asking whether a promise can be kept for a larger load of interacting agents that still interact with the parts at the same scale according to 2. In 2. we are asking how big are the effective agents we are examining, on an absolute level, while keeping the relative sizes or dimensionless ratios the same.

Definition 148 (Functional scalability)**.** *A system possesses this property if it can change its size (number of agents) without altering its function, or changing its ability to keep promises (as observed by an external agent).*

When an architect or designer makes a scaled model, he or she is using the idea that by magnifying all of the dimensions of a building or device in proportion, there will be a predictable correspondence between the behaviour of the model and the full-size result. This is not a trivial matter; it presupposes a notion of dynamical similarity[Bur13a, Bar96, Bar03].

7.2 LESSONS FROM EFFECTIVE COARSE DESCRIPTIONS

The more we understand how systems work, the more chance we have of forming the right expectations about them, and using them successfully for their semantics as tools.

Scales thus play a unique role in understanding how systems behave.

When we are unable to separate the characteristics of a system at different scales, we fail to understand the semantics of the system, and see only the interactions of its parts. The phrase 'can't see the wood for the trees' expresses this: if all you see is trees, you can't perceive a forest or wood as an identifiable, aggregate entity, or know what role it plays in climate interactions, or global biodiversity, etc.

At each level of aggregate description, there are typically new patterns and emergent characteristics that cannot be understood from the individual parts, in the absence of their interactions. There can be new semantics associated with these (such as when a collection of components becomes a computer). In promise theory, we call these aggregations superagents. At coarser scale, dynamics become more 'universal' in their characters, i.e. we see what is common to all forests rather than what is special about one.

7.2.1 Ensemble scaling and universality of characters

System characteristics can be assessed statistically over 'ensembles' or collections of comparable systems. This is particularly useful when faced with real-world indeterminism. Sometimes we form ensembles over time, to discuss trends of change; other times, ensembles may be picked from different locations and circumstances to compare locations at the same time[97].

Some of these characteristics are famous, like Moore's law, which characterizes the development of a state by saying that the number of transistors on chips doubles every two years. This is dynamical law of scale, often interpreted semantically to mean that computing power or performance doubles every two years, which is really a different question. Similarly, there is Wirth's law[Wik17] which suggests that software is actually getting slower more rapidly than hardware becomes faster. The semantic association here is one of increasing bloat and complexity in software. Another example is how we tend to be rosy eyed about places and things we don't know very well. If you live in a place for a long time, you probe it in greater detail, and experience more of its problems. This could be formalized into a law: the longer you live somewhere, the more you complain about it.

Formal scaling relations are rarely found in information technology, because they are the more familiar tool of physicists: the physics of information systems is a relatively new area of study, motivated partly by the growth of systems to respectable scales. Recently studies exposing universal scaling in cities were carried out[Bet13, BLH+07], revealing how key performance indicators like wages, disease, services, etc., grow according to universal patterns. Cities are semantically rich information systems, as diverse as any technology. This suggests that high level patterns are universal, as we would expect on general scaling principles, and that they are likely to exist in other functional systems.

Some scaling relations expose patterns across ensembles of systems that are parameterized by size, weight, length, or some measurable dimension. Scaling is connected with dimensionless variables, such as numerical counts N. When control parameters have dimensions (mass, length, time, etc.) the control parameters are dimensionless ratios: t/T, m/μ, and so on.

Scaling relations can be compared with great generality across many systems, and correspondingly lead to only general observations that don't necessarily describe any of the samples in an ensemble well. So, while this is scientifically interesting, does any of it help us to design and build systems, or predict faults, flaws or even errors? How might the semantics at aggregate scales, and even universal scaling relations, be disrupted in a way that became anomalous?

7.2.2 DIMENSIONLESS RATIOS AND SIMILARITY IN CONTINUUM SYSTEMS

The subject of scaling in continuum models is known as dimensional analysis[Bar96, Bar03]. Our entire knowledge of classical mechanics or the physics of rigid bodies and fluids is essentially based on the understanding of scaling ratios. Could this also play a role in systems at a more general level of description? The concept of *dynamical similitude*, going back to Newton's time, expresses the idea that a real-valued continuum system can be scaled by multiplying by a scale factor. Such a scale factor applied equally to all measures of the same type cancels out of ratios with the same dimension, and thus leaves dimensionless ratios invariant. The basic observation is that, as long as we can use real numbers to characterize scales, and topologies are simple, then we can use the argument that:

$$8/4 = 4/2 = 2/1 = 2x/x \quad (\text{for any } x) \tag{7.1}$$

to claim the similarity of objects (see figure 7.1).

Example 152 (Scaling food)**.** *Suppose we make a meal for two, doubling everything could make a meal for four. But, what about the cooking time? Should we double that too? Luckily there is a methodology for working it out.*

They key insight in physics of scales is that it is dimensionless ratios of scales that determine important behaviours in the world, not lengths and times directly. This seems reasonable, as only dimensionless ratios are (manifestly) independent of the units used to measure things. It cannot be the case that a phenomenon depends on whether it is measured in centimetres or inches.

The Buckingham Pi Theorem formalizes the notion that only dimensionless scales are useful for characterizing and extrapolating behaviour (for instance, to make a scale

model). Any variable that cannot be directly compared to another in the same system of measurements is potentially a fiction of the measurement itself. The Buckingham method allows us to write behavioural relationships (like flow equations) in a form which depends on on dimensionless combinations of variables. Although this is just a rewriting exercise, it casts relationships in a form that is manifestly free of assumptions about scale. We can only say how big something is relative to a calibration reference; anything else is subjective and subject to ad hoc interpretation.

From any set of parameters, one begins by putting together all the possible dimensionless combinations. For example, in the case of an aspect ratio, a rectangle has width W and height H, both of which have the dimensions of length. So W/H is the only combination. If we add the third dimension, there is distance from the observer to the screen D, so now we can also make W/D, H/D, W/H. See figure 7.1.

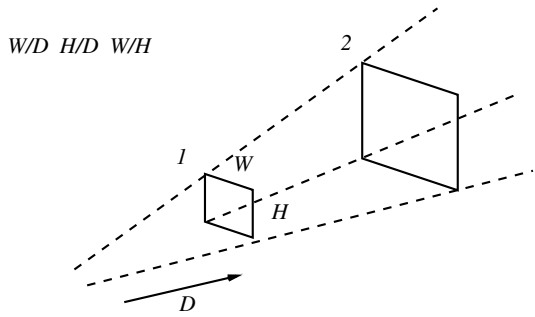


Figure 7.1: Dimensionless scaling ratios capture the qualitative measurable properties of a system as it scales, for continuous systems. For discrete systems the ratios may not exist.

7.2.3 DISCRETE AGENT SYSTEMS AND BREAKDOWN OF SCALE INVARIANCE

In a discrete system, like a graph with variable topology, the assumption of

$$8/4 = 4/2 = 2/1 = 2x/x \quad (\text{for any } x) \tag{7.2}$$

no longer holds in a simple way (see figure 7.2). Simply doubling the number of agents does not preserve the similarity of topology.

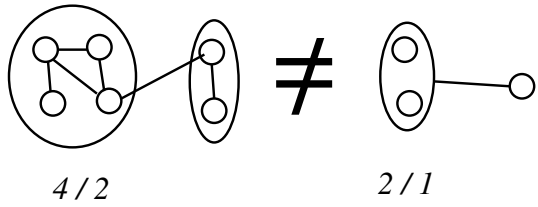


Figure 7.2: Discrete agent systems are not necessarily similar in their semantics or structure just by scaling simple ratios.

7.2.4 SCALING SEMANTICS: SYSTEM FUNCTION

Addressing semantics on the level of atomic and molecular combinations of agents is straightforward, if not intricate enough, but how could we describe systems of such great size and intricacy that it is impractical to look at every promise individually? This has been studied using promise theory in [Bur14, Bur15a].

We can learn from material science, i.e. how to describe systems without tracking every atom in a substance. Phenomena manifest at different scales with different expressions. By eliminating detail and moving to an effective coarse description, we might lose total information, but the lesson from the physics is that this doesn't necessarily matter. The laws of scaling and 'renormalization' can keep track of how small details either lose their importance at scale, or get amplified into chaos.

For example, we can say that certain crystalline structures are susceptible to cracks. Some are flexible and resilient. More practically, we can say things like: don't build a hammer out of glass as it is unsuitable. Such properties are usually learned from experience and research rather than being a priori predicted from models, though today the information technology does allow predictions. Elucidating scaled properties of abstract 'human-machine materials' is a place to start, even if we can't immediately see all of the motivations and consequences: we follow the example of material science, hoping for enlightenment down the road.

7.2.5 FLAWS IN SCALING: WHAT SEMANTICS CAN CHANGE?

Design issues that are sensitive to scaling include costs and benefits in a number of areas. Some issues are controllable (they are short range effects). Others are dominated by the environment and thus beyond control (they are long range effects).

- Maintenance.
- Replacability.

- Transport of people, materials and information.
- Latency.
- The efficiency of resource supply, harvesting of output, and maintenance come into play.

One of the things a larger system can afford is to be more tolerant of faults; this may come from having sufficient redundancy of dynamical processes to sustain functional behaviour. However, when the fault lies in an external dependence, its own redundancy may not help. So a multiply redundant flight control computer on an aircraft is no help if the plane runs out of fuel.

Offering

- Backups, dynamical multitude, redundant agents keeping the same promise for failover.
- Alternatives, semantic multitude or different promises for a different solution.

These are two ways of building flexibility and resilience into systems. They can be combined with 'mixing' or random selection (so that agents do not always bind to the same promise provider). In that way, a fault in a single location can be mitigated by varying the selected location. This introduces a new scale, e.g. the timescales for the bindings.

7.3 SCALED AGENTS (SUB AND SUPERAGENCY)

The treatment of a collection of agents as a single entity is a choice of scale, made by any observer (see figure 7.3). Agency can further be defined recursively to build up hierarchies of component parts, and their dependencies.

Definition 149 (Long and short range coupling)**.** *If dependencies are entirely self-contained, there is a natural local scaling. If dependencies remain between agents, even at a larger coarser scale, then there are long range effects.*

In [Bur14], I showed how spatial boundaries can be defined by membership to a group or role. We still have to explicate the relationship between the internal members and the structure of the whole, as perceived by an observer.

We define a collective superagent as a spacetime structure that has collective agency, i.e. its intended semantics relate to a collection of agents surrounded by a logical boundary, with collective semantics (see figure 7.3).

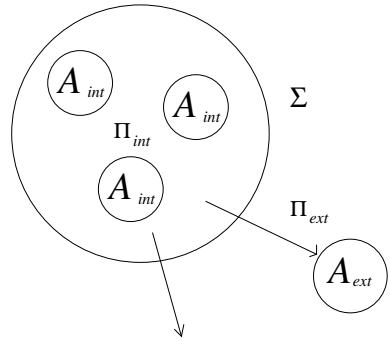


Figure 7.3: Agent structure consists of an element that makes a number of exterior promises, some of which are scalar, some vector, etc. Interior promises are invisible from the outside. For example, the subagents might promise to be trees (and not suddenly become cars), while the superagent promises to be a wood.

Definition 150 (Superagent)**.** *A superagent of size S is any bounded agency composed of individually separable agencies, partially or completely linked by internal vector promises. The bare superagent is defined by the closed graph, without any external adjacencies.*

Superagency allows us to talk about redundant 'material' bulk promises. In principle an observer could draw a line around any collection of agents and call it a cell or composite superagent. This is an assessment any agent can make, as part of its definition of an agency scale. However, it might still be of interest to distinguish special criteria by which such an arbitration might occur. In component design, for instance, the choice of boundary has often to do with the a choice interface an agent wants to interact with.

The alternatives fall into three basic categories (see figure 7.4):

(a) A membership in a group or associative role, where the central membership authority may be either inside or outside the boundary, e.g. city limits, or company campus. In this case, we are identifying a group of symmetrical agents.

$$A_{\rm host} \xrightarrow{+\rm membership} \{A_{\rm tenant}\} \tag{7.3}$$

$$\{A_{\rm tenant}\} \xrightarrow{-\rm membership} A_{\rm host} \tag{7.4}$$

(b) A total graph or collaborative role. In this case, we are identifying agents with coordinated behaviours.

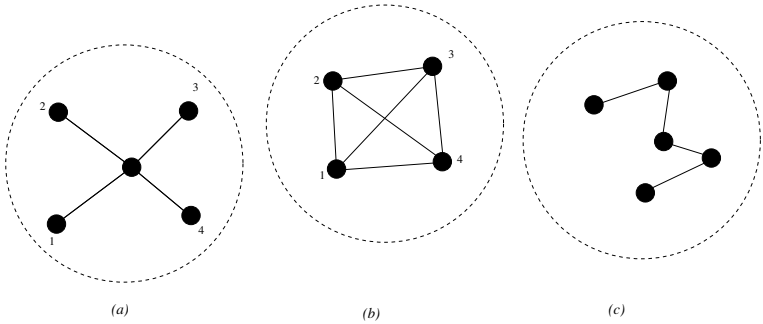


Figure 7.4: Three ways of binding collective agency. Another way is to simply make an arbitrary collection: (a) membership, (b) interaction mesh, (c) chain or dependency hierarchy.

(c) A dependency graph, path or story. In this case we are identifying dependency bindings.

If we have superagents, then obviously we can talk about subagents, where a particular system region can be decomposed into helpful parts. Lesser agents can also be residents or satellites of other agents. This leads to a hierarchy of containment by functional role.

7.4 SUPERAGENT SURFACE BOUNDARY

Superagent boundaries may be formed with different structural biases.

- Aggregations of agents, related through membership or allegiance to a single leader (leader may be inside or outside the superagent), and the connections are made through the leader as a proxy-hub.
- A cluster of agents linked by cooperative vector promises.
- Strongly cooperative agents which are inseparable without breaking an external promise. E.g. an organism made of components that are all different and non-redundant to the functioning of the whole.

Interior promises are those entirely within the surface boundary of a superagent.

Definition 151 (Surface of a superagent)**.** *The exposed* surface Σ *of the agent is the subset of interior/internal agents that have adjacencies to agencies outside the superagent.*

A superagent surface may also make new explicit promises that are not identifiable with a single component agency.

7.5 SYSTEM MODULARITY

Modularity is about how we define the semantic boundaries around different functional elements in a system[98]. Modularity is a strategy that the software industry, in particular, has made bold claims about. If we believe software doctrine, modularity leads to systems that are easier to understand, easier to maintain, and which scale better, give you an automatic place in software heaven, and contribute to world peace. Modularity has been connected to the increased efficiency, better focus and even the suggestion of easier innovation. However, some of these claims seem contradictory, and data from cooperative systems like cities offer a more nuanced view of the facts behind the doctrine[Bet13, Bur16b].

Modularity is a long standing strategy in human cooperation, related to specialization. By organizing a cooperative collaboration, the suggestion is that a specialist can focus all its efforts on becoming an expert (assuming it has plenty to spare), learning by experience, with more depth by being encumbered by less breadth. Anthropological evidence for this is also mixed[Dia97, Har11, Tai88], but there seems to be an argument for scaling: if we want to scale productivity of things like food and services to support a surplus, and free people of the need to spent all their time subsisting, then a network of specialized services does have clear advantages, which are revealed in studies of urban scaling.

Focusing on a single task requires less changes of context. This is really a claim that contention leads to wasted resources. Changing context is costly as it may involve an overhead of remembering and optimizing very different activities, not to mention the expense of adopting different tools. If the tooling is expensive, centralized specialization makes sense (see also section 5.13).

Example 153. *When technologies are expensive and immature, it takes large wealthy agents to afford them. They can provide the technology as a service. Later as the technology becomes cheap, everyone can own it as a commodity and the argument for centralization or modularity is exploded from above and below.*

For example, think of the telephone. In the beginning, users had to go to a special building to use a telephone. Later, public telephone terminals (phone boxes) were deployed at distributed locations. Today, the technology has become so cheap that everyone can have their own telephone in their pocket.

Is this complete decentralization of modules, or the elimination of modules be equalizing for everyone? Clearly the answer depends on what scale we choose to look at.

We could argue that the modules are much smaller if everyone can have their own, or that the module is all-encompassing since everyone has one (no barriers).

7.5.1 MODULES AS AGENTS AND SCALES

The standard ways of drawing technology systems are; the markitechture (layer cake) and the class diagram (DAG). Neither of these captures anything like the actual human-computer system in its environment. So how can we do better?

There are various levels of modularity.

- Non-modular monolith.
- Modular monolith.
- Service oriented.

A module may be a component or a superagent aggregation of components in a nesting hierarchy of any depth. The promises we attribute to these may have scale invariant characters (like service delivered or failed, usually the dynamical properties), but the details of semantics are usually pinned to a particular scale.

7.5.2 MODULARITY, SCOPE, AND HUMAN COGNITION

It is practically an axiom of information technology that 'modular' design is superior if not actually 'the correct' approach to system design. Books have chapters and sections, software has procedures, functions, classes, objects, etc. Retail areas of a city categorize shops by type of goods. It's very common that engineers associate modularity with specialization, but we also argue about 'failure domains' (see chapter 9), believing that modules are security barriers, and that interfaces are security checkpoints. A common feature of all the explanations of modularity is the matter of limited cognitive capacity for human engineers, rather than the dynamical limitations of the system itself. We only *trust* systems if we understand them.

Example 154 (Object Orientation and Service Architecture)**.** *Much effort went into looking for evidence to support the ad hoc hypothesis that Object Oriented (OO) Programming was superior to other forms of programming, like Service Oriented Architecture (SOA). It's one of the embarrassing episodes that reveals the immaturity of Computer Science at millenial shift. The design principals of OO and SOA were compared in [ABH06], using Promise Theoretic analysis. This revealed how OO focus on type semantics could its doctrine down blind alleys, while in fact its methods were not significantly more effective than other approaches. The insistence on modularity could event lead to*

unnecessarily fragmented program code, much harder to perceive for engineers than a longer block of procedural code. In short, the OO hypothesis seems robustly disproven today.

Containing scope is about managing cognitive limitations. Containment is an attempt to limit the impact of human action, especially human errors, because we tend to employ blunt instruments, like search and replace, to trade consistency for possible overreach. Mistakes are made unconsciously as we flirt with the limitations of our cognition.

Even carefully trained humans may be unaware that the extent of their actions can exceed the horizon of their intent. What starts as a reaction to an event or scenario ends up being a chain reaction: an 'explosion', quite out of control. This is why (even in futility) we try to limit scope. This is a trust issue. We cannot cope with too many details at once, so we trust what happens inside selected black boxes, temporarily abdicating responsibility. A system of connected parts is not isolated, by definition. We can leave tripwires to shore up cognitive deficits, but once an error or fault has occurred, it will almost certainly propagate, because the kinds of checks and control points that could stop it lead to inefficiencies. No one likes to stand in security checkpoints or passport queues, but this is what must happen to verify things we do not trust. Choosing to trust is a policy decision to waive this verification.

Modularity pins a scale, because modules are semantic. They are discriminants. They cannot be independent of scale, because they refer to the scale at which they interact, like boundary conditions, spacetime dimension, size, density.

Modularity is a hallowed goal in computer science. Recently in particular, programming culture has come to deride so-called monolithic.

7.5.3 THE LIMITS OF PROPAGATION

In factory manufacturing processes, components and processes are designed to be isolatable so that they can be manufactured according to commodity templates. This separability leads to a mirage of independence. Isolatable should not be confused with isolated. It is only when we connect the components together that the system begins to work.

Unit tests are component oriented production control. Checking that resistors, capacitors, and transistors meet their specs (plus or minus some tolerances) is only a probable prerequisite for building the actual system of intent. We can build the composite system to be tolerant of these variations, or brittle. In computer science, classic methods favour the latter, basically because of the culture of misrepresenting outcomes as precise, mutually exclusive, and deterministic. We misrepresent Boole calling them Boolean, but poor old Boole shouldn't get the blame, as he didn't believe in these sharp distinctions.

I am not a historian, but this seemed to enter with Turing, von Neumann, etc and the design of digital computers.

Without the existence of space and time to vary, nothing could be different anywhere or anywhen; so, at some level (if we want to get fundamental), spacetime lies at the bottom of everything we do. Promise theory is a kind of model of spacetime that incorporates meaning (what's the difference between a typewriter space and a parking space? Well, physics can't tell you, but computer science could. But what's the difference between a running on your laptop and running on the cloud? Software engineering can't really tell you, but physics could. So, we need a combined view, which is what promise theory allows us to construct.

7.5.4 SEPARATION OF CONCERNS

Can we separate concerns? Individual system components may contribute in different roles within the whole. Unless the different agents within a system are coordinated in intent, their correlated behaviour may not have the concerted outcome an observer might expect. In human teams, we speak of 'loyalty to the team' and 'getting behind a leader' and so on, but the same alignment of promises is needed in all components, not merely human agents. This might seem like a trivial point, but human system designers are far more likely to assume that non-human agents will be automatically aligned with the system purpose, because they were made for the purpose, that they might neglect to be certain.

Another argument for separation of concerns into discrete agents that make specific promises is to limit semantic complexity by creating a 'menu' of limited choice. This is how a restaurant limits customer expectations, and therefore makes the outcomes repeatable and predictable. Of course, this does not lead to perfect certainty[99].

- Scale of measurement
- Valuation of objects
- Separation of skills, procedures - often accompanies separation of data structures. This is a conventional choice, not a necessity.

So let's examine the idea that breaking this down into smallest possible units is the answer. Well, that can't be true, else we would all write directly to bits. There are good reasons for defining integers and reals and characters, and structs and so on. This simplifies logic. Structural data design is about caching or preempting computation.

At the opposite end of the extreme, one could create a generic datastructure and set of algorithms for doing any kind of data, e.g. JSON, YAML, SQL. This separates along spacetime lines rather than semantic lines.

Class programming asks us to alter our code for each kind of template (TYPE). Can we estimate the relative costs of separation?

- Can we count how many declarations are needed to represent a problem?
- Can we calculate how much wastage in terms of unused memory from fixed size structs?
- How many additional header files do we need to include (C library is a lump, but has many header files that add semantic complexity)
- What is the search cost of finding the right header file to include? What about dependencies between header files and libraries?

Not only separation of function, but about how it is packaged together with other things. I the C library, many functions are independent, just bundled. But we still need to include the whole thing, unless we've gone all unikernel.

7.5.5 HUMAN-COMPUTER SYSTEMS

Our ideas are coloured by what we hope for or expect of systems, not about what they actually promise. Most of these expectations are based on trust, and trust is rooted in our human capacity to assess relationships.

We judge the trustworthiness of all things by which tribe they belong to, in our minds; because it is the promises we perceive or believe in rather than the promises an agent actually makes and keeps. This capacity for humans to distort expectations is a serious issue may lead to serious functional and operational errors.

Example 155. *We might have preferences for certain programming languages, certain development frameworks, even certain programming styles, pair programming, teamwork, lean, agile, and every cliche we've every heard. But how many of these actually make clear promises, with measurable effects? CD is one example that is relatively clear.*.

We can learn something about the scaling of systemic behaviours from studies in other areas. In [Bur13a], I likened modern computers to material science. I still believe that this is a fair analogy,

If one considers technology in imagined isolation, then differences between modular compositions and black box monolithic designs are only slight, and are difficult to justify rationally. However, if we treat the entire interleaved human-computer system[Bur04a, BB14a] as a system of agents, with different properties and roles, then there is reason to see a clear distinction between what are referred to as monolithic and microservice architectures. The number of interactions points is far greater when one has transparent access the components of the system as a 'white box'.

7.5.6 SYSTEM EQUIVALENCE: DYNAMIC AND SEMANTIC INVARIANCE

We must address system scaling from two perspectives: dynamical and semantic. Rather than beginning with change, it is simpler to ask what it means to maintain properties at the same level: invariance of characteristics.

Dynamical invariance, or how quantitative behavioural performance remains the same as certain parameters change, can be applied to study the scaling behaviour of separate components, but because we know that the sum of a collection of components makes promises above and beyond the sum of the collection of promises of its components, we cannot be completely certain that invariance of components implies invariance of the collective system. *Semantic invariance*, or how functionality remains the same, is not obviously related to the size of a system. Some functions pin a system to a particular scale. For example, if we double the size of a spanner, without doubling the size of a nut or bolt, it loses its ability to perform the required function. If a talking clock, or information agent speaks at twice the speed, humans might not be able to understand it anymore. So there are natural functional limitations to scaling.

The term dynamical similarity (or similitude, in Newton's language) refers to the invariance of certain dynamical or behavioural system properties as others are changed. Dynamical similarity is a science closely connected with dimensional analysis[Bur13a]. It allows us to build scale models of systems, such as ships or aircraft to study in wind tunnels or wave machines, where the testing of a full size system would impractical in the initial stages. Dynamical scaling is particularly important during testing, because we are often interested in making a system larger to cope with greater demand, while leaving the functionality provided unchanged. This is not always possible.

Some confusion arises around describing architectures and scales, as industry practices tend to focus on specific parts of systems when they talk about 'the system'. Software developers and programmers P make promises to users U, through the results of their promises, but do not usually interact with the users directly, so users are easily forgotten. They do not interact with operations engineers directly, so operations engineers are easily forgotten. However, this does not mean that they are not part of 'the system' proper. Indeed, designers, developers, and architects are usually focused on semantic issues, while operational engineers, builders, and maintenance engineers are focused on the performance.

In order to 'scale up' a system to cope with higher demand, we might have to change certain components to scale its promises faithfully, e.g. replace humans with machines because humans cannot work fast enough, or employ robots because humans cannot reach the top shelf. This might be a naive view of scaling: just because our old idea

does not work anymore at larger scale, does not mean to say that there is not some other arrangement that could keep the same agents working in a more efficient or scalable manner, by a different process, with different costs and different tradeoffs. An alternative design might not be dynamically similar to the original one, so we should not expect it to have precisely similar behaviour, even if the user-facing promises are superficially the same. An exterior agent's assessment of the reliability of the promise keeping might be quite different, either qualitatively (semantically) or quantitatively (dynamically). Scaling a system dynamically and semantically is a subtle issue. It is not simply a question of replacing components with stronger or faster components.

7.5.7 MONOLITHIC AND CENTRALIZED SYSTEMS

Monolithic designs are usually fully integrated, imperative, impositional processes, that are driven by some input to produce some output is a quasi-deterministic fashion. The factory production line is the classic example of this: factories were designed to work like steam engines, in which humans were just commodity parts trained to keep simple-minded promises, without reference to the operational infrastructure or the end-user interaction.

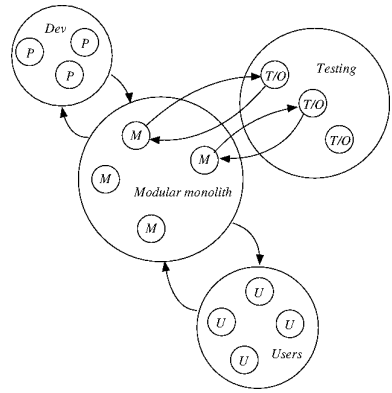


Figure 7.5: Monoliths, as defined, generally have interior modularity. A so-called centralized monolithic design has separation of concerns along stakeholder roles, e.g. developer, code, and user. The optimization is a semantic one, contrasted with a microservice split that keeps module stakeholders closer together at the cost of dynamical inefficiency.

Figure 7.5 shows a typical arrangement of the promises associated with a monolithic system. Designers, developers or programmers P who compose the arrangement of parts and promises promise to produce a set of modules M that work together. In the diagram,

the ring around the developers P implies that they work as a group, making cooperative promises that coordinate their behaviour. They make the same collective promises to the system modules. Similarly, the superagent boundary around the modules implies a set of interior promises that coordinate the behaviour of the modules. The collection of modules and their interior promises make a number of exterior promises to the users U. System testing has been separated here, so that test agents T interact only with the modules they test, not with developers or users. This picture is quite typical of a software production pipeline. It also represents the implementation process of many other kinds of system, whether cars, buildings, or food produce.

Developers usually see the system as the collection of M and tests T. Users see the system as the promises made by M, and sometimes the promises made by P about updates and repairs. Ironically, few of the agents P or U within the system see themselves as being part of the system, yet their promises and behaviours are crucial to understanding its performance. The structure of modules within $\{M\}$ must have an influence on the semantics and dynamics of the promises the collective set can keep. Even a monolithic integrated design cannot easily be made without separating semantic concerns into understandable modules. Modularity is a semantic issue, even a human issue (since semantics are for humans P and sometimes U)[100].

Modularity addresses the practical concerns of limiting the number of promises made per module. This effectively limits the size and complexity of each component too, making it easier for engineers to understand each part. However, it breaks up the total narrative of the system into smaller pieces that are then harder to reconstruct into a total narrative. The burden on programmers is now to understand the whole design. See figure 7.6. The hypothesis of modularity is that different developers can work on different modules, and that the whole story must then work together analogously to Adam Smith's 'invisible hand'. There is an unspoken presumption that either a) a design specification exists in enough detail to detail this composition of parts, or b) programmers collaborate and communicate sufficiently to cause this to emerge along side tests. In this story, the promises made to the end user cannot be fully understood without an appreciation of complete workflows throughout the system, because workflow is impositional, triggered by events and inputs from the top level. The storyline or narrative of the system is thus rigid and brittle in the sense that every part must move immediately in response to the others. There is no asynchronicity or obvious buffering.

7.5.8 SERVICE ORIENTED OR DECENTRALIZED SYSTEMS

What makes distributed service oriented systems different from monolithic systems? Figure 7.6 has no implied scale, it could refer equally well to either. It represents a necessary arrangement of interior promises between modules in order to lead to the

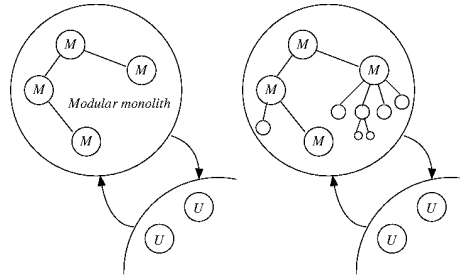


Figure 7.6: Where we place boundaries may be arbitrary, and it might depend on the promises that can be kept on either side of the boundary. A modular design may have structure that is preferred, e.g. a particular hierarchy, in order to scale the activities of the different modules. The promise arrangements are probably driven by the desire to avoid duplication of intentional behaviour (what the software industry calls 'code').

exterior promises to end users, who might themselves be localized or distributed. Should we understand M as a centralized entity, where the superagent boundary is localized and physical, or as a decentralized set of modules, where the superagent boundary is abstract and fluid? The same question can be posed for the users. Are they localized or scattered?

Designers and engineers often focus on separation of concerns through modularity as being the key difference between service oriented and monolithic. However, this does not seem to stand up to examination, given that equal modularity may be present in monolithic systems discussed above. There is plenty of modularity in so-called monolithic designs, especially in object oriented and functional code forms. If we imagine a monolithic system, like a collection of parts on the table in front of us, and increase the scale so that the parts are no longer on a table, but spread across the globe, with no other changes, then we would call the system distributed. The number of modules and the promises could be exactly the same. Clearly modularity can exist at any scale, so there must be more to service orientation than mere prejudice about distribution.

What may different about a service oriented system is the asynchronicity and tolerance of the interactions between the modules. Recall that monolithic systems are typically rigid and reactive, imitating deterministic responses like the shafts and gears of a steam engine. A service oriented approach relaxes this rigidity by queuing messages between modules (rather than forcing with rigid drive shafts). Whereas a monolithic system has only a single timescale, this service pattern results in multiple timescales in separate timelines, depending on the specific interaction patterns. The service systems may still push or impose their requests typically, but they respond asynchronously, with their own timelines.

Service orientation need not change the basic functional design of promises, modular classes or function calls significantly. However, it places bilateral queues, or asynchronous pipelines, in between the functions, reducing the brittleness of response. This has two effects.

- It increases the options for robustness, or fault tolerance, by decoupling processing schedules, and allowing for component redundancy or realtime replacement of components. Thus, if one part of a service oriented system doesn't respond, it might be possible to find an alternative service provider to keep the promise. There is an increased cost associated with this decoupling.

- Causal weak coupling introduces multiple timescales, rather than a single timescale throttled by its weakest link, because each server can run its own affairs without promising to be a slave to exterior impositions; thus, the buffering separation increases the uncertainty of response time, and computation time.

Service oriented systems thus trade rigidity and temporal predictability for robustness and flexibility with greater uncertainty and runtime cost. That cost might be worth it, however, if the risk of a complete rigid failure propagation brings down the complete system for an extended time.

7.5.9 THE MICROSERVICE HYPOTHESIS

In software architectures, a design pattern known as microservices has begun to gather attention, since around 2015, due to its association with continuous delivery methods, and scaling of whole software lifecycle at some major companies, most notably Netflix.

Microservices are more than just service oriented systems with many small services: while monolithic and service oriented designs patterns talk only about the machinery of software eschewing the user and the operational infrastructure, microservice design integrates the notion of continuity between developer, operational infrastructure, and end-user, as a closed loop human-computer system. Microservices might not be the best name for this design pattern (the name indicates that its inventors didn't realize the extent of what they were really doing, being focused on the programming rather than the whole system). The pattern suggests a more holistic integration of all the system stakeholders at the microscopic system scale: programmers P, tests T, and users U, within a common design.

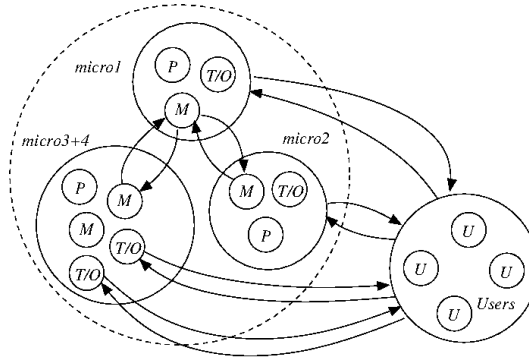


Figure 7.7: A so-called microservice architecture is not so much about centralization or decentralization as it is about mixing concerns. Different lifecycle roles are insourced. This seeks dynamical integration of all processes leading to the keeping of a functional promise: the interaction loop between promiser and promisee is minimized.

Comment 14. *Limiting the number of programmers working together, and limiting the amount of code they work on recognizes the cognitive limitations of programmers P, which is an important step in taking on the practical limitations of the human components in the system, particularly because software lifecycles are must faster than other kinds of societal systems. Updated versions are expected, because requirements are in rapid flux, while profit margins are low, meaning that focus on reliability is tackled by 'quick fix' evolutionary adaptation rather than pre-tested perfection.*

Figure 7.7 shows how one might redraw the monolithic system in figure 7.5 as a microservice design. Once again, the extent to which we can call this design centralized or decentralized is a about the scale we imagine for the diagram. The 'micro' in microservices refers to semantic or functional scale rather than physical scale. More importantly, the major superagent groups have been pulled apart and reintegrated in different constellations, emphasizing the strengths of certain interactions we would like to be tightly coupled. Each module is now closer to its own tests, and its own private developers. Microservice advocates speak of team programming.

- Programmers are tightly coupled to their modules and their user promises, leading to a close and familiar relationship that builds trust.
- A change in one location has the appearance of being more localized, so one might hope that failures could be contained within the microclusters rather than propagating throughout the whole. This hope is far from clear however. As we

know from the Byzantine influence pathways described in section 6.4 decoupling a system is much more complicated than focusing on a diagram of partial promises. Figure 7.8 shows how a common dependence on some third party exterior service, such as a directory service (DNS), a host, or the electrical power supply, can still allow a single module to bring down the rest by breaches of containment with respect to promises only represented implicitly.

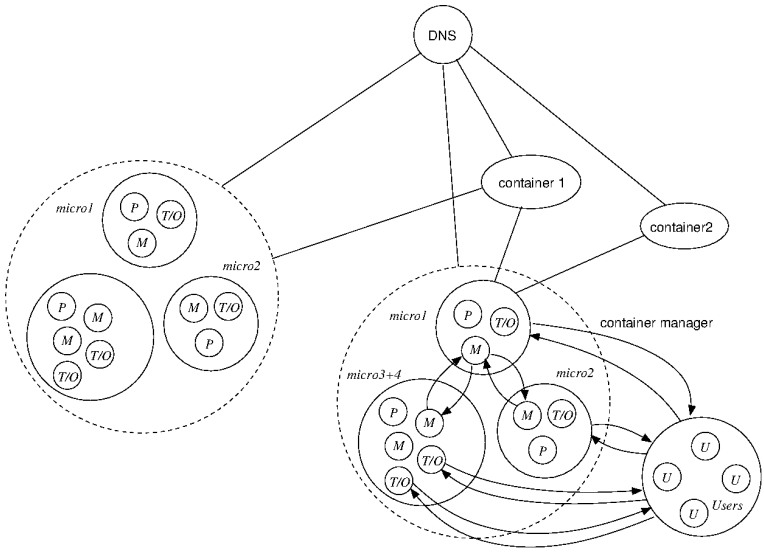


Figure 7.8: The apparent decoupling of modules in a microservice architecture is partly illusory. There are implicit promises that couple them indirectly, such as mutual reliance on directory services, software containers, hosting platforms, power supplies, and so on.

We cannot easily make claims about better containment of faults in a microservice architecture, nor can we claim that they scale better. What is most important, in my view, is that it is a rare example of a design that respects the human agents within the system. The respect for cognitive limitations, and close relationships between designer, user, and module encourages trust, and closer familiarity with all parties. This is so fundamental to the human societal modus operandi. Moreover, for a strategy of rapid repair above pre-tested perfection, it does seem well suited, as easy familiarity with all stakeholders makes communication and response potentially more efficient. The key problem with monolithic designs is the sheer weight of knowledge required to understand the cooperation and coordination of parts.

What is notably missing from the microservice promise sketch is the cooperative

promises between the developers. These have been traded out explicitly, in order to save on cognitive burden, but this must itself have a cost that remains to be evaluated. If the groups are truly independent, how do they know their resulting system will function in total? If we discount an Adam Smith or Stephen Wolfram invisible hand of emergent luck, we still need to engineer the collective promises between the teams. is a test for the entire system (what used to be the exterior promises of M in figure 7.5).

It could be that this method of stakeholder reintegration is well suited only to particular kinds of system, that are already loosely coupled enough (or well understood enough) to allow the holistic aspect of the system to be taken for granted. This remains for future studies to uncover.

7.5.10 SUMMARY: WHAT DOES MODULARITY REALLY MEAN?

The chief conclusion of applying promise theory to systems is that many of our ideas and complaints about systems are particular figments of a particular scale. The table below compares corresponding concepts for bottom-up scaling.

MACROSCOPIC, SCALED, HIGH LEVEL	MICROSCOPIC, LOW LEVEL
Superagent	Agent
Deterministic	Non-deterministic and retry until success
Synchronous	Asynchronous, retry until success or give up
Failure in exterior service promise (+)	Failure in dependent interior promises (-)
Redundant backup promise (+)	Promise to acquire multiple dependees (-)
(Un)Responsive on a long timescale	(Un)Responsive on a short timescale

Comment 15. *Modules separate interior from exterior promises, but do not necessarily halt the propagation of unwanted influence. By forcing influence through certain checkpoints, one might verify and potentially filter the passage of influence over each selectively matched $\pm$ promise channel, at the expense of creating a deliberate serial bottleneck. Parallelizing interfaces such adds the cost of calibrating and coordinating consistency. If the promises made by the modules are not constant over the timescale of the interactions, the presumed boundary of the module (a superagent region of constant promises) is inconsistent, and we cannot claim containment of dynamics or semantics: then the claims of rigorous modularity are rendered false.*

7.6 DISTRIBUTION OF STATE IN PROCESSES

Processes are networks of agents that propagate states from location to location, whether on the interior or on the exterior of a bounary. The location of those states might not be

important on a large enough scale, far enough removed from the details of the process, but—on the interaction scale—engineers are often deeply concerned about how state is distributed. This has been the subject of some controversy in software engineering, for instance[BFKT, Wig17]. This section is based on the analysis in [Bur19b].

7.6.1 INFORMAL IDEAS ABOUT STATE AND CAUSALITY

The meaning of state can be pursued on many levels. Suffice it to say that no decision process or computation can proceed without an interior dependence on some kind of state[LP97]. State basically refers to any information that characterizes a process, over some interior timescale, and may be stored anywhere within a hierarchy of agents and subagents that characterize the process. For example, a clock is a process that maintains an interior state counter.

The state concept therefore spans the full pantheon of memory what is stored in the registers of chips, to configuration files, source code, or to long term databases—but no two authors will necessarily agree on which states are the relevant ones to their arguments, or why they choose to treat one kind of state differently to another.

7.6.2 THE ROLE OF SCALE IN LOCALIZATION

Scale plays a role in localizing state. Data may be localized to a geographical region, a datacentre, a host, a container, a function, or even a register. Some authors play the game of offloading state from one location to another in order to claim statelessness; but that isn't a scale invariant characteristic. In a virtualized world of cloud computing, the meaning of being 'within' a process, entity, or agent is ambiguous—and, across the many articles written about statelessness, there is little agreement about what storage level one should be talking about. One author may call a process 'stateless' or 'immutable', meaning that all decisions except unavoidable input-output should be based on state that is frozen and held invariant before the specific execution of the process (see figure 7.9). The scale-dependence of state management, over space and time, was also the origin of the so-called 'configuration management wars'[Bur04c, TH98, Tra02, Kan03, CHIK03].

Another may consider these prior frozen configuration choices to belong to a phase of the processing itself, on a larger timescale, and the lifetime of a single process is further part of a longer meta-process, involving many clients, in which continuous delivery of upgrades to changes of dependencies are interleaved with the keeping of client promises. Authors thus cherry pick the meaning of state to suit their arguments.

It's especially important to revisit the topic of state in cloud computing, where some definitions concerning locality need to be reconsidered in a scale invariant way; cloud processes often scale elastically to some extent. Moreover, virtualization adds

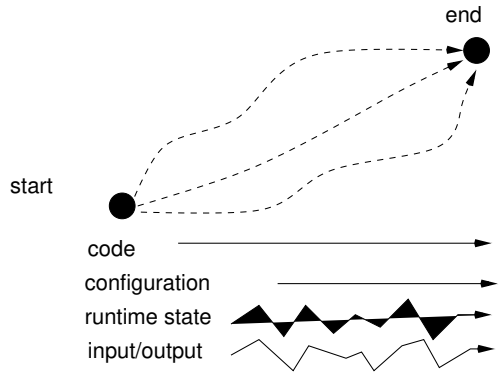


Figure 7.9: Processes exhibit state on all manner of timescales. Some state is frozen into initial state of the packaging, some is allowed to change. The main question is, over what timescale (or part of the process) does the state remain invariant?

layers to its meaning: from source code, configuration, container packaging, runtime environment, virtual machine, physical host, etc. Development is constantly jumping between the concerns of different levels: from programming, to continuous delivery, 'DevOps', configuration management, serverless, etc.

Various perspectives on these issues have been expressed over the years[SS97, Moo02, Hel07]. The Twelve Factor App[Wig17] is a widely referred to best practice manifesto, which advocates that developers should execute applications as 'one or more stateless processes', and that such apps ought to have a 'share nothing' architecture to avoid contention. Recommendations then go on to explain how necessary state can still be kept, after all, by employing 'backing services', and how caching of certain objects is 'allowed'. There is a suggestion in these principles of favouring transactional rather than continuous processing, for a particular scale and meaning of 'transaction'. So-called 'sticky sessions' that tie multiple web transactions to a particular server context and location are explicitly rejected in [Wig17]. However, if one takes an extended session to mean a 'complete' dialogue over a business process, including reliable TCP and TLS negotiations, etc, then it's no longer clear that 'stateless', as implied, has an unambiguous meaning.

Some platforms, like Kubernetes[Bur15b], have been designed with a notion of statelessness in mind, but later extended their models to include state. This suggests that state itself is not the real problem the guidelines are clawing at, but that the rejection of what is perceived as stateful behaviour is really an attempt to address concerns about localization (scale), speed (timescales), and fault tolerance (spread prevention). All of

this needs to be scaled to cope with the extended cloud of ubiquitous embedded devices.

7.6.3 POPULAR IDEAS ABOUT STATELESSNESS

A quick online search and query reveals a number of definitions about statelessness, which point a finger at state but discuss reliability. These definitions are generally tied to a single case, and generalize only by implication[101]. For example:

> 'When an application is stateless, the server does not store any state about the client session. Instead, the session data is stored on the client and passed to the server as needed.'

This definition refers to client and server roles in a two-agent interaction. Similarly:

> 'A stateful service keeps state 'between the connections' or session interactions, whereas a stateless service does not'

In this case 'between the connections' is intended to mean that access to a service is transactional and that some data persist between independent transactions when a process is stateful. The preoccupation with connection indicates the author assumes a client-server style application. Wikipedia talks only about stateless applications, which it defines to mean:

> 'that no session information is retained by the receiver'

The substitution of 'connections' for 'transaction' belies a focus on client-server computing at a particular scale, and the assumption that single exchanges are safely invariant while longer exchanges are not. That would depend on the extent to which the composition of exchanges were 'locked' (e.g. mutex locks) and the data could go missing in case of interruption. The excerpt also distinguishes the roles of sender and receiver as part of the concept, as for a protocol, implying a directional arrow from client to server. In general developers tend to focus their thinking on the preferred scale of the subtask they are working on, even as ideas about DevOps and Continuous Delivery ask them to rethink those ideas on a larger scale, for development continuity.

> 'Think of stateless as if a service is a hardware chip. All computation needs short term storage like registers and stack and maybe heap. What happens when we lose power? A service that calculates some value and returns a result can be considered purely stateless. Purely stateful would be a service that maintains state like a game server tracking scores and players in a game world.'

In this view, scale plays a key role. A short lifetime for data (as measured in the proper time of the process, rather than wall-clock time) means stateless and persistent and

reusable means stateful. As I'll show later, this view of stateless approximates the idea of memoryless systems (section 7.6.6), and very long term data that are 'invariant' over the effective lifetime of the process can be separated out and treated differently (see section 7.7.5). A refinement of this:

> 'Stateful means written to localhost
> Semi-stateful means 1:1 write over the network)
> Semi-stateless (n+1 networks relay eventual write
> Stateless means held in memory until dereferenced.'

The scale dependence becomes more evident here. The reference to 'in memory' suggests a short lived once-only usage of state, versus persistent storage again. The reference to networking is less clear: which network are we referring to? The interior host bus is a network alongside the LAN/WAN. In the past, the preference for the processor bus was about relative speeds: interior communication was much faster than LAN/WAN communication. Today, it is impossible to know whether interior host bus or exterior LAN/WAN connection will be faster. The goal of avoiding an architecture that relies on a network connection, to disk storage, or to a remote service, therfore doesn't stand scrutiny in the cloud era, as even an in-memory process memory might be paged out to disk, or retained in a hash table for extended usage. With this in mind, where exactly is the imagined line between runtime state and persistent state in the architecture?

7.6.4 PROCESS HISTORY, ENTROPY, AND TIMESCALES

Lamport was the probably the first author to appreciate the relativity of time in computer science, as a succession of causally ordered events[Lam78]. The transmission of messages, carrying causal influence plays a central role in understanding what happens in computation, both locally and in a distributed system. Processes that depend only on a current local register set, i.e. not on the recent past or the extended history of all such sets are called *path independent* or *memoryless* (see appendix)[102]. This concept will be most useful to explaining what authors are trying to express in 'stateless'.

Predictive systems can never be memoryless, for instance, because they explicitly use past experiences—not only current state—to predict the near future, involving a computation over multiple samples collected over multiple proper times[103]. Weather modelling is the archetypal case in point. Small differences in the data sets can lead to large changes in the predictions. The dependence of data processing on history is utterly susceptible to scaling arguments.

It's hard to generalize about the role of causality, because it is so dependent on the nature of interactions in a system, but I need to make a few comments on this because the integrity of data sources has been challenged in some commentaries, e.g.

[Hel16, Bon19]. Some authors have argued that we should never throw away data because it might be needed later to 'recover' from some fault. Apart from being unsustainable[104], the premise of this argument is wrong from a causal perspective; indeed, systems must eventually forget their past over some timescale. The question, again, reduces to understanding the relevant timescales. Sometimes regulatory bodies insist on data retention, for legal reasons, up to some statute of limitations. This can be factored into the policy and separated from the data that need to accessed dynamically at runtime, becoming effectively a part of a different application.

Does it matter whether we take data transaction by transaction, or in bulk as a sum of all transactions (such as an aggregate database or a file)? The process by which data arrived in the past is only relevant if it affects the promises it makes at the time of usage by another agent. For some authors, a 'database' is merely a cache for a long linear process of accretion, i.e. that the current state of a database is the sum of all past facts transacted at its entry point[Hel16, Bon19]. This is the 'calculus' view of data: by integrating differential elements from some beginning to some end one calculates the answer transaction by transaction.

> 'From this perspective, the contents of the database hold a caching of the latest record values in the logs. The truth is the log. The database is a cache of a subset of the log. That cached subset happens to be the latest value of each record and index value from the log.'[Hel16]

This quote, summarizes a linear view of data in which the current state is merely an arbitrary point is a deterministic trajectory—a classic Turing machine argument. In this view, all the causal information lies in the past. Neither of these assumptions generalizes to distributed cloud computing, as I'll show in section 5.11. The argument goes: the present is a function that does not alter the past (which is true). It's then assumed that the function is a linear function, formed from a sequence of 'deltas' or state changes that can be stored in a log and added together to yield the current state. The latter is only true for reversible linearized (memoryless) processes, localized to a single point of entry. Popular techniques of this include the use of 'actors', 'pure functions', and even mutex locks, with associated costs (i.e. without interior reads or writes to exterior data sources)[105].

Example 156 (Functional programming)**.** *The functional programming manifesto claims that programs will be deterministic and reproducible if functions are defined as following the following axioms: (i) if they are 'total functions', in the mathematical sense (i.e. they return an output for every input), ii) if they are deterministic, i.e. they return the same output for the same input (which assumes they do not implicitly rely on variant configuration, database lookups, and are immune to 'noise' over all timescales involved in the process), and iii) if they alter no exterior state other than computing their promised*

output. The composition of such objects would certainly be deterministic, but the axioms are often violated in practice, e.g. by system faults and by inattention to environmental noise. The naive view is that programs are perfectly isolated and that, if programmers do nothing, nothing will happen. In practice, there is no such isolation context for distributed systems, and it's up to programmers to explicitly perform noise correction fast enough to maintain these axioms.

It feeds the non-relativistic view that one can absolutely capture 'facts' about the source of information, which can then be preserved and treated as immutable. The error in this argument is that, as soon as a sample of data has been transported into storage, it is no longer the source view: its the observer view of the data store. One would have to transport all relevant context into the data snapshot. This may be a simple discriminator, but it's still an arbitrary view that doesn't remove the uncertainties and doesn't warrant its preservation without an understanding of its significance.

In order to recover a snapshot of state, it's argued that one should never delete any of the contributing facts in the logs. After all, in a linear system, the current snapshot is merely the balance of all previous transactions within the system; but this is simplistic. In a non-linear system, there is no such separation of process timescales, and we would still need the full past history including all leakages of noise and interleaved processes to understand the present in general, because computations are not always linearizable (see section 5.10.2). The final outcome becomes strongly dependent on the particular moment at which data were collected (a kind of 'butterfly effect')—so both the current snapshot of the database contains information that is not in the journal[106].

Even if our system is linear, and we keep all data in an eternal timeseries, searching backwards takes time, so we index data, but to do so imposes a rising cost (in energy and labelling), possibly identifying unique instances, by GUID or quasi-universal timestamps, and so on. There is a reason we aggregate data and use caches and latest summaries: to localize relevant context, and separate it from other data whose meaning has gone into the mix of entropy. What every system designer needs to consider carefully is the extent to which we flatten a dynamical process into a timeless, static database model. It's okay to do so as long as you accept the loss of a relative temporal reference, and the accumulation of entropy. It does not necessarily imply that the past is lost. Process time information is lost to entropy by design in most systems—and this is not wrong; relational databases focus entirely on static semantics of data, not on process histories. We now have a pantheon of time-series databases that focus entirely on brute force history, without attention to 'scaled semantics'. By this, I mean that timeseries typically involve many pattern scales, such as by hour, by week, by month, etc, and that these are treated as issues for post hoc analysis rather than being built into the data model in an efficient manner. There is a policy choice—a choice of semantics that can't be stipulated

in general. If we want to get systems to behave 'properly' as well as efficiently, we need to select an appropriate causal policy for what is 'proper', and be aware that these choices are inherently scale dependent in both space and time.

7.6.5 THE POINT OF USAGE

I want to make one more point about the importance of the recipient in the determination of so-called facts. Facts are a kind of promise, but it takes two agents and two promises to pass on facts: a sender and a receiver. Past facts are therefore not really as immutable as is often claimed. There is the original source value:

$$S \xrightarrow{+V} R \tag{7.5}$$

and there is the moment at which the value is used:

$$R \xrightarrow{-V} S. \tag{7.6}$$

It is this latter promise that actually passes on the information and creates an event[Bur19a]. We ask: can the promises be kept invariantly? The conditions for agent R might have changed, between the keeping of these two promises, even if S is somehow etched in stone. Promise Theory predicts that it's not the time of origin of the data that matters to its causal influence, but rather the moment at which the data are accepted into the timeline of the next agent—just as in an electric circuit with feedback, which is the inspiration for control theory.

If one dabbles in synchronous versus asynchronous processing, this may matter: if the timescale over which the behaviour of an agent changes is comparable to the timescale over which you sample data, the data basically become random variables, by the receiver's hand (not the source's). Sufficient immutability can be assured in a few ways: e.g. by assembling the states one promises to depend on before processing, to decouple independent processes. That way the processes can continue at their own rate and still avoid such issues[107]. This is what functions do in programming: automatic variables (by value) copy the value into private workspace.

There are two approaches: trusting the source and trusting the receiver.

- If one trusts the promise of invariance (immutability) of the source S (as one does in timeseries databases, as a trusted second hand source), then keeping state there becomes a policy choice and one reads information directly from that source. This second hand information replaces the source of 'truth', and is not the same. This assumes that there is also an invariant key for looking up the data, which is understood by both parties and that relationship is also constant.

- If one trusts the receiver to sample and keep the information invariant (as one does in using private local variables in programming, and in 'immutable images' in cloud computing) then policy abhors reading any new information from outside the boundary of containment. Derivative processes, like that of R, which depend on data from a source S, thus capture all their dependencies before beginning to keep any subsequent promises—freezing them and rendering them immutable as a matter of policy.

It seems, in neither case, is there any guarantee of the invariance of the data, or the ability to replay the same interactions multiple times, as that is entirely a policy decision for R. In general data come from many different sources, with conditions that are quite unequal, and merely sampling these into a trusted repository does not alter that. In fact it adds a second layer of trust, by the Intermediate Agent Law (see 7.2.2 of [BB14a]).

What matters is not whether we cache data in a database, or keep each update in a journal. What matters is whether the data can be relied upon not to change over the course of trying to use them. This often assumes implicitly that there is a single correct dependency value for each moment in time, with an ability to 'roll back', yet this notion has been debunked many times[BC11].

Coarse graining time and separating interior from exterior time: this is what we do in functional programming. It introduces the full range of process causal viewpoints.

> 'Mutable state needs to be contained.'

There is a causal twist here, in the form of Nyquist's sampling law, and Shannon's error correction law[SW49, CT91] (for a review, see [Bur19c]). If a system has knowledge of a correct state (where correct is promised as a matter of policy), then no unintended deviations from that state will be measured by an observer if restored quickly enough. We take for granted that such feedback processes are on-going at a low level of memory in all our technology at all times. The same principles may also be applied at a higher level, as maintenance procedures[108]. If problems are fixed before a fault can be sampled downstream, there will be no propagation—and the system will be invariant by virtue of dynamic equilibrium[Bur13a]. This is how data consensus works and memory error correction work, for instance.

The Twelve Factor App manifesto claims to avoid software erosion, which implies that there is should be a maintenance process at work.

In the various cloud manifestos, there has been a focus on reorganizing the maintenance process so that dependent information is embedded and assumed invariant (as in a transaction), by freezing 'golden images'. If errors are detected post hoc, due to state drift, one deletes the process, replacing it with a fresh copy (see the car example below), which is accepted as a matter of policy. Corrective actions post hoc (instead of

preventative actions) then require some kind of 'rollback' on the scale of the promised transaction. Without a preventative error correction underlying the use of the image, a post hoc correction is needed, but if the error has been observed by any process, its influence will already be too late. In a kernel or database monitor, keeping validating transactions is relatively easy (given deep memory error correction), but as the scale of interactions grows, isolation becomes less likely.

7.6.6 THE IMPORTANCE OF FORGETTING AND INDISTINGUISHABILITY

Dependency on process history adds baggage (process mass[Bur19c]), tangling up changes in dependencies with data going back in time. Memoryless processes appear 'agile' or 'cheap' to run—as far as change, creation, deletion, replacement, etc, are concerned—because they have little baggage, i.e. fewer dependencies. This allows changes to be made to them easily. Of course, that should not be taken to mean that all changes will be simply localized, with no effect on other processes.

Keeping processes agile seems to be a way to address reproducibility. Reproducibility has nothing to do with computation, per se. It has to do with trust. We build businesses and institutions on reproducibility, because it allows anyone to verify a result and repair possible errors, when something is judged to go wrong. At such a time, the idea is that we can forget a state of the system we consider erroneous, and replace it with a 'proper' policy-acceptable one.

If a process is interrupted, and some of the contributing past information is lost, we believe that this must compromise the reproducibility of the outcome. This is not necessarily true, as explained below, but let's continue. The result is that we make transactions that carry all relevant data bundled with them, and keep a copy until the transaction has successfully been prosecuted. If the transaction should fail to be confirmed, we can repeat it. The implicit assumption here is that there will be no effect on either agent (the receiver or the source) unless the transaction completes successfully. Repetition should also be 'safe', i.e. convergent to a definite outcome, not just a 'first come first serve' (FCFS) in a random walk.

If there are errors on multiple scales, we may have to go back across cumulative transactions on multiple scales to repeat the transaction. So, if one builds systems at scale on the basis of transactional determinism, we are doomed to keeping ever growing amounts of data, up to the size of the largest transaction. The cost grows in relation to history (time) not in relation to scale of parallel instances (space).

If any data are left 'floating in limbo', in extended 'stateful' sessions, it is argued that those states could be lost and data may go missing. This is not about statefulness,

but about when the data are discarded from the source. This seems to return us to a justification for the idea of never throwing away any data, discussed above, but this is not so. We simply need to preserve data until confirmation of receipt—as in reliable transfer protocols. Next we need to define the scale of that remark: on what process scale do we need confirmation of 'ok to delete'? If we treat transactions as packet by packet over a session, then a process crash could lose data. But if we treat completion as the confirmation of a promise kept that depends on the data, then scaled transactions can be constructed using locks.

Safety under repetition is the much neglected method of assuring certainty in systems. Idempotence is sometimes mentioned, but most authors think this means remembering which transactions are completed on a FCFS basis without checking for contradictions.

Example 157. *Numbering of transactions, like in TCP, is one way to maintain coherence of order, but this is not always meaningful without ad hoc assumptions. In a data pipeline, for example, you can number items, but the numbers assume that both ends have a clear sense of how the arrival of data will take place in order to combine multiple sources meaningfully. So, while a 1:N transport can be regularized by partial ordering, N:1 aggregation cannot. Numbering promises process as 'intentional' events, but random arrivals have no such coherence, making the processes non-reproducible unless the entire history is captured and used as the future source of truth. That assumed truth may not be a faithful representation of the original source processes. As always, the receiver determines the semantics of data.*

Fixed point convergence is the more economical key to reproducibility, because a fixed point is the only certain way of guiding a system with random behaviour to a known state. It requires knowledge of a future state to which the system is headed. Absolute invariant future state is simple and cheap to manage. Relative future state is fragile and susceptible to faults and cumulative errors of execution. In data pipelines, for instance, this needs to be treated very carefully (see our Koalja work, for instance[BP19]).

Example 158 (State of a car)**.** *If you crash your car, the car will be lost if it is a unique, one-of-a-kind design. But if the memory of its design (its ideal state, minus age and mileage decrepitation) is kept elsewhere, as a separate manufacturing process, then the car can be replaced—but not its runtime state, i.e. the precise details of what it was doing at the time of the crash (including its passenger inventory). If the car client is not fussy, it may overlook a few details and be satisfied with an equivalent.*

Some of the state you are happy to forget, some you are attached to. There is no fact present in the car that can tell you how to discriminate this line. We don't always need to remember the past, sometimes only a single future 'desired' state. Indeed, it's desirable to forget the past as it's just in the way—and sends you a regular bill.

Example 159. *A statute of limitations, or causality horizon. If you build on advanced boundary conditions, you need no memory. Memoryless processes have a very short horizon. A function is basically a mutex lock around a private cache of function argument values. The completion of the function is a tick of its clock at the scale of functions, and therefore a pure function is memoryless at that scale.*

We may conclude that the implicit accusation in favour of 'statelessness' is that one should push responsibility for preserving state backwards along a causal chain (into the past): onto the sources rather than the receivers. But which of these agents is the most fragile? Why should events from the past be more important or 'correct' than what happens in the future? Many will answer that 'the past determines the future', but this naive determinism. Promises about future states also contain causal information, and it is not correct once one accounts properly for scaling. So we need to return to look at the ways in which causal propagation takes place and is scale dependent.

7.6.7 SUMMARY: PROPER AND IMPROPER INVARIANTS

To summarize, we appear to have succumbed to the trap of illusory detail rather than focusing on the key question: how can a stable promise be kept? So what we seem to be struggling to express is a design decision (a promise) about which processes will be considered atomic at each scale, which is equivalent to expressing which data we are potentially willing to lose.

Assumption 5 (Promise manifesto)**.** *The central question about systems is: will the outcome of promises be invariant to the conditions under which the promise is kept or not? Does that matter?*

From this perspective, statelessness actually seems to imply a preference to use 'current state' in short-lived, ephemeral interactions, over which dependencies can be treated as approximate invariants. If all agent interactions are kept short, as measured by their own proper time, and relative to the scale of their exterior time,

$$\frac{\Delta t_{\text{interior}}}{\Delta t_{\text{exterior}}} \ll 1. \tag{7.7}$$

then a process will tend to a state of statistical invariance. This relative timescale argument could perhaps be used as a definition of 'micro' in microservice. It makes dynamical sense: it's a linearization of a potentially non-linear process. We should understand that as a design constraint. The choice enables eventually consistent outcomes over a sample set, but there may be other sample sets that have not reached the equilibrium. The best promise (no guarantee) of stability is to ensure that updates have plenty of time to reach equilibrium, by separating timescales.

Example 160. *If a dependency changes every second, and a process promises output every few seconds, there is insufficient time for the process to promise invariance. However, if changes to dependencies occur only once per year, then processes lasting a few seconds can be considered invariant in practice, by (7.7).*

There is an implicit separation of concerns in talking about state: the part of state that we care about, in the current context, and the part we don't. This suggests a natural partitioning *by policy* of scales for each relative process, rather than a universal best practice guideline. The final point about 'good enough replacement' leads us to consider the role of observability and distinguishability in deciding outcomes[Bur19a].

The issue in question seems to be: over what timescale can some form of state be considered dependable (invariant relative to the receiver), from the perspective of all stakeholders in the system? This includes at least the role of the client (when data are uploaded) and the server (receiver of uploaded data). For the remainder of the discussion, I'll therefore focus on the dynamical principles of keeping promises across a multitude of scales.

7.7 LOCALITY AND DISTINGUISHABILITY

Let's try to illustrate how a few key concepts, are behind the promises authors are trying to capture in rhetorical usage. These concepts are mainly spacetime concepts, about order, scale, and observation[Bur19a].

7.7.1 LOCALIZATION OR SPATIAL PARTITIONING

The virtue of source code modularity, for the separation of *semantic* concerns, is doctrine in computer science. Localization of process execution in space is a form of modularity too, that we call scaling of execution context—'containment' for short. Today, virtual machines and container technologies are the tools for achieving such spatial localization, erecting barriers that are supposed to limit the exchange of influence between interior and exterior. Isolation from an influence X implies causal independence of X (see section 2.4.3). In Promise Theory, bare agents that make no promises are assumed independent of all causal influence a priori.

In Promise Theory, every active or passive part of a system is an agent. The definition of an agent also defines a scale, and an isolation boundary for that scale. Elementary agents are the smallest observable scale of a system, and superagent clusters of them form larger scales, where agents work together to keep collaborative promises.

Modularity is achieved by partitioning the process into separate agents whose interactions are defined by promises.

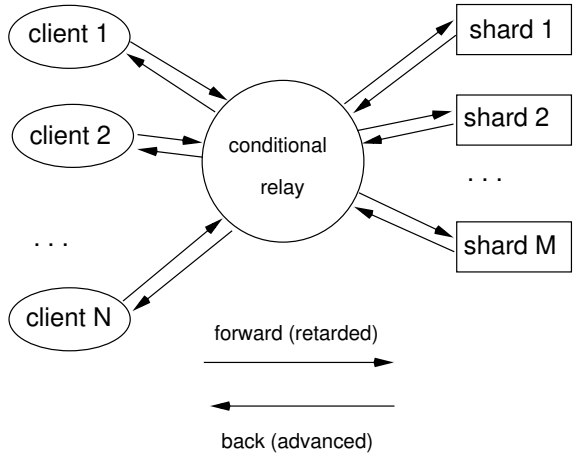


Figure 7.10: Causal influence can flow forwards or backwards along the direction of an interaction (defined arbitrarily). At the ends of an $N : M$ interaction, say from clients to servers, data may be shared or kept in entirely separate scopes, in either direction. Where nothing is shared, we often say that the data are 'sharded' or have private scope. Integrating shard implies reintroducing a shared resource in the agent that aggregates them however, so we don't escape sharing; we only delay its onset in order to acquire partial invariance of data for other processes.

Definition 152 (Partitioning of a process)**.** *A subdivision of a process agent into a number of non-overlapping subagents, whose mutual promises are exposed.*

Agents may be decomposed by space or by time if they are distinguishable by some label. In other words, if agents are numbered in order, or labelled with names or types, they can be separated into subdivisions using the labels they promise. Examples include the division of a larger process into microservices, or the partitioning of a database into shards, perhaps curated by intermediaries with a APIs between them, but the principle doesn't refer to any particular technology or set of assumptions.

Now, let's formalize the hierarchy of agents involved in representing data processing, starting with the easy parts. This helps to establish the language of promises and use of terminology.

7.7.2 SCALING OF STATE

Every memory location in a system that can record state is an agent that can promise to hold a value[109]. All states are memory agents, and partitionings lead to separation of

states that keep different promises—sometimes called 'sharding' (see figure 7.11):

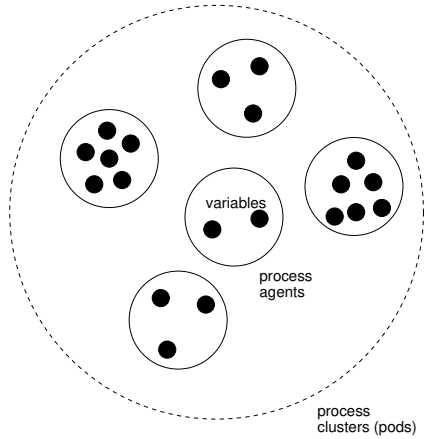


Figure 7.11: Distinguishability of agents to clients determines whether they can be partitioned or whether they form a redundant set. Agents are distinguished by the promises they make, which in turn are states of the agent. Together the states of a system form a configuration. Some states are promised to exterior agents and some have private scope. In general the scope of a promised value is contained by a certain scale, which we call a semantic boundary. Redundant agents are indistinguishable. Non-redundant shards make different promises.

Definition 153 (Variable)**.** *A key-value pair, that promises a name and a value, representable as a simple agent: promise.*

$$V \xrightarrow{+(name,value)} S, \tag{7.8}$$

within a scope S.

The internal variables of a process agent are what one normally thinks of as the state of the agent.

Definition 154 (State of a variable)**.** *Let V be any variable (or set of variables), on the interior of a process agent S, which takes values from any set of distinguishable elements X. The state of V is the value of $V \in X$, promised by the source S to any outside agent $A_?$:*

$$S \xrightarrow{+V} A_?. \tag{7.9}$$

The problem with this definition is that the state is only observable to the process agent O outside of A, if both the source S promises the information as an exterior promise, and the observer also promises to accept and use the promised information:

Definition 155 (Observable (state) of a variable)**.** *Let the state of a variable $V_S \in X$, promised by S, be sampled on any occasion by an observer O:*

$$S \xrightarrow{+V_S} O \tag{7.10}$$

$$O \xrightarrow{-V_O} S, \tag{7.11}$$

where $V_O \in X$. Then the observable state of the variable is $V_S \cap V_O$.

The role played by the observer in this definition is crucial. It underlines how relativity will play a role in all stateful phenomena[110]. If a state is persistent or invariant to order N samples, then multiple samples, by an observer O, lead to the same value for some number of samples N. It is clear that a variable, even of order 1 implies the existence of memory on that can be sampled by some process that carries information to an observer. State is an observer issue rather than a provider issue.

Definition 156 (State of an agent)**.** *Any promise made by an agent S to any other agent O, concerning the state of an interior variable V:*

$$S \xrightarrow{+V} O. \tag{7.12}$$

The total state of an agent S may consist of any number (or composition) of variables V.

In effect, each variable is a subagent member of a larger process (superagent). This tells us that the boundary where we choose to define the edge of a process plays an important role in the way we describe its behaviours (process, container, group, pod, host, etc). Moreover, since it relates to promises, whatever else it may be, the statefulness of an agent S is an *assessment* made by each recipient observer involved in promised interactions.

7.7.3 SEQUENCES OR TEMPORAL PARTITIONING

There is also localization in time: when a process's trajectory starts and ends (see figure 7.9), and whether information is fed into it only at those endpoints as immutable constants or 'invariants' of the process, or whether information is accepted into it and modifies the process as it evolves. The purpose of 'functions' in programming is to promise that only the I/O channels belonging to the function (the arguments and the return value) lead to change. This is hard to assure on a larger scale, however, as computer code

is only one in a mixture of overlapping processes whose dependencies lead to mixing. Seeking invariance of promises is the key to process stability. A process may be called *adiabatic*[Bur03] if exterior information does not alter promise definitions over the timescale of interactions that rely on it—meaning that a process's promises are invariant over the interval during which they are being kept, with no configuration changes. If process fragments are partitioned end to end, they form a sequence. If they coexist, either starting or ending at a common agent at a common time, then they may be called concurrent.

Confusion ensues for many developers when considering the origins for such change. There may be intentional change, such as a code change or a manual input of data, and there may be unintentional (hidden) change to a dependency presumed invariant. A lot of rhetoric has been exchanged around 'never touch the system and it will never go wrong', but 'if it fails, don't fix it—replace it'. These are policy decisions, not unique recipes for handling change, but they are rooted in the idea that invariance is a solid foundation for process continuity. They may have different causal outcomes.

Localization allows us to partition processes in space and time, holding certain aspects constant over the duration of a subprocess.

- Time localization leads to promise invariance for agents.
- Space locality leads to privacy of scope and non-interference.

This is a form of lock-free synchronization. Proponents of the Actor Model will find these principles familiar[HBS73].

7.7.4 DISTINGUISHABILITY, PARTITIONS, AND REDUNDANCY

In order to distinguish partitions from redundant agents from partitioned agents, all agents have to be distinguishable by the agents that interact with them.

Definition 157 (Redundant agents)**.** *Two agents A_1 and A_2 are observationally redundant if they make the same promises to an observer A_3, and the A_3 accepts the promises equally, i.e.*

$$A_1 \xrightarrow{+X} A_3 \tag{7.13}$$
$$A_2 \xrightarrow{+X} A_3 \tag{7.14}$$
$$A_3 \xrightarrow{-Y} A_1 \tag{7.15}$$
$$A_3 \xrightarrow{-Y} A_2 \tag{7.16}$$

where $X \cap Y \neq \emptyset$.

An observer that discriminates between two agents making identical promises may be called a discriminator. Such agents are the basis of all decision making based on data. In principle, discrimination at a single agent location can be made in on the basis of space (source agent) or arrival time, but each simultaneous time step sampled by the discriminator represents a new causal decision, so that must depend on how we define the scale timesteps and the discriminator itself—a distributed superagent discriminator has to be able to promise interior time coherence. As always, the scale of encapsulation over which we can assume invariance ('coherence') plays the main complicating role.

Definition 158 (Partitioned agents)**.** *Let two collections of agents* $P_1 = \{A_1, \ldots\}$ *and* $P_2 = \{A_2, \ldots\}$ *be partitions, as discriminated by an observer* A_3*, then:*

$$P_1 \xrightarrow{+X_1} A_3 \tag{7.17}$$

$$P_2 \xrightarrow{+X_2} A_3 \tag{7.18}$$

$$A_3 \xrightarrow{-Y} A_1 \tag{7.19}$$

$$A_3 \xrightarrow{-Y} A_2 \tag{7.20}$$

where $X_1 \cap Y \neq \emptyset$*,* $X_2 \cap Y \neq \emptyset$*, and* $X_1 \cap X_2 = \emptyset$*.*

Note that a partitioning is a superagent, i.e.

Lemma 33 (Partitions are agents)**.** *If agents* A_i *are scale* n*, then a partition is a superagent at scale* $n + 1$*.*

This follows trivially from the definitions, without any restrictions on what other promises the agents may make.

Example 161 (Shared nothing)**.** *'Shared nothing' agents cannot be completely redundant. They make* a priori *uncorrelated and uncalibrated promises, so they cannot promise determined redundancy; they are merely random and possibly similar. As soon as they accept a common source of calibration (by cooperative dependency on a single source), or achieve dynamical equilibration (as in data consistency protocols), they share something from* $O(1)$ *to* $O(N^2)$*. In practice, developers will overstate 'shared nothing' and allow agents to share configuration that only changes on a long timescale, making it effectively invariant, according to (7.7).*

7.7.5 INVARIANCE OF DISTINGUISHABLE PROMISES

We can now define the meaning of *invariance* of an interaction, as a process involving pairs of agents (on any scale): a source and an observer:

Definition 159 (Invariance of a promise π)**.** *An agent's (exterior) promise may be called invariant if the body of the promise is constant for the lifetime of the promise.*

Theorem 5 (Invariant promises)**.** *An agent may be invariant with respect to exterior change if and only if it contains all its dependencies and promises them constant.*

To prove this, suppose an agent A makes a promise of X to an observer O, conditionally on the promise of another agent A_D being kept, which promises a dependency D, then:

$$\pi_A : A \xrightarrow{X|D} O \tag{7.21}$$

$$\pi_{A^T} : A \xrightarrow{-D} A_D \tag{7.22}$$

$$\pi_D : D \xrightarrow{+D} A. \tag{7.23}$$

In addition, A promises its value of D to be constant:

$$\pi_C : A \xrightarrow{D=\text{const}} O \tag{7.24}$$

If we form the superagent from $\{A, A_D\}$, from the two collaborating agents, then

$$\{A, A_D\} \xrightarrow{+X|D} O, \tag{7.25}$$

$$\{A, A_D\} \xrightarrow{-D=\text{const}} O, \tag{7.26}$$

which is unconditional after A assesses the promise to provide D has been kept, i.e. $\alpha_A(\pi_D) \neq 0$. Now, since D is promised constant, $X|D \to X|\text{const}$, and this is invariant under D. If the promise π_A is made unconditionally, then $D = \emptyset$, and the result is trivially true.

Example 162. *This theorem may be considered the basis for freezing all dependencies and configurations internally in containers before execution, e.g. in Docker or using fixed images in the cloud. But it also applies to dynamical configuration engines, like CFEngine etc, where the policy for D is fixed and the promise keeping is maintained dynamically by 'self-healing' based on fixed policy. In either case, the promise may be broken because the result cannot be guaranteed. In the case of containment, one is trusting the integrity of the containment, which can only be assured for non-runtime state by making it read-only. In the case of dynamical configuration, runtime state can also be repaired by dynamical equilibrium in the presence of noise. So the required invariance is not dependent on a particular strategy. A dynamical configuration is more expensive in processing, but may prevent errors before they occur. Static containment may appear cheap, in terms of runtime process resources, but is more likely to result in exterior consequences that breach containment, because the timescale of exposure to non-corrective actions is maximal (the lifetime of the process) rather than a regular shorter maintenance interval.*

We can reduce this to a very simple expression of invariance for agents, as a whole:

Definition 160 (Invariance of an agent A)**.** *An agent promises to accept nothing from any agent.*

From the proof, we see that this is a scale dependent assertion, since we may always partition the agent internally such that one interior partition makes a promise on which the other interior partition depends, entirely within the boundary of the agent, leaving its exterior promise unconditional.

The key assumption in this argument is the absence of unintended change, by impositions, such as noise, that systems are fragile to. Many developers believe that there is no noise in systems, only the programmed change, because a lot of it has been eliminated by low level error correction[111].

As long as 'a system' of choice interacts with some other agency, it is not the total system, merely an arbitrary partitioning of it. If an promises to accept nothing, i.e. make no (-) promises, then its interior state will be invariant for as long as that promise can be kept. We may assume that this is the actual goal of systems that serve users.

7.7.6 SHARING VERSUS PARTITIONING

Partitioning is naturally the opposite of sharing. The original definition defined 'shared nothing' for databases was 'neither memory nor peripheral storage is shared among processors'[Sto86]. In the cloud era, we need a more generalized abstraction to cope with the branching technologies.

Definition 161 ('Shared nothing' agent)**.** *An agent is keeps all of its promises unconditionally (makes no assisted promises), from its own intrinsic capabilities, i.e. it makes no promises that require the assistance of another agent.*

An example would be a unikernel architecture without network service interactions and private disk. Since 'shared nothing' the default assumption of 'autonomous' behaviour for agents in Promise Theory, we see the utility of promises to describing these issues: every dependency has to be revealed as a promise to see the channels that constrain process operation. A simple consequence of defining this is that agents that play the mediating roles of hubs, switches, or routers (as in figure 7.10) for promises of any kind violate the condition above.

Lemma 34 (Hubs violate 'shared nothing')**.** *Any nodes that operate as a point of confluence, or a divergence like a switch, and connect a sharding of process messages, promises to partake in sharing and violates the assumptions of 'shared nothing' in the broader sense. Shared nothing involves promises that do not depend on one another for any of their resources. This even includes power supply at the deepest level.*

The degree of sensitivity to sharing depends on the possible variance of the dependency. If we seek to depend only on invariants, then there is only weak coupling. The shorter the timescale for variation (the more active a dependency is), the greater its effect on the system promises.

The connection between state and partitioning lies in what information is used to distinguish process agents. Process trajectories trace the evolution of causal relationships from agent to agent, at whatever scale an observer can witness. Some agents may make indistinguishable promises leading to *redundant parallelism*, or they can promise full distinguishability leading to *branching* and *switching* (decision making).

Distinguishability of promises is what enables non-shared futures, i.e. sharding and switching of process trajectories (see figure 7.10). In switching, a process selects from a set of possible futures based on the state of variable data. Each decision partitions possible outcomes into branches, or 'many worlds' futures. If branches are indistinguishable (contain only the same redundant information, both in initial conditions and runtime state) then the branching process is memoryless, and the superposition of agents acts as a single superagent on a larger scale. If they are distinguishable, the program takes on a new course.

Occasionally, different process flows merge into a single one. This happens with pull requests in software development, for example. It also happens in data pipelines where source information gets aggregated into batches. Branching (+ promises) costs nothing, but merging timelines (- promises) requires causal intervention, and the input of new information in the form of state-dependent selection criteria. Thus we do not escape the cost of a memory process by branching as long as there is a need for the branches to be merged[112].

Example 163. *In network package delivery, i.e. 'routing', for instance, the decision about which route to take is variable according to a separate parallel process, but that might be made on the same timescale as the running process from which the data arises. This is then non-linear (see section 5.10.2). It is always downstream (receiver) promises that carry the greatest responsibility and the greatest potential cost—hence the downstream principle (see section 2.4.3).*

7.8 AGENT SCALING HIERARCHIES

A full range of agent dynamics should be able to mimic all the processes of the natural world, as well as artificial behaviours based on computation. Promise theory's goal is then to explicate the semantics of these processes. As in the scaling and renormalization of physics, this leads to hierarchical ideas. However, the implications for semantics go further than those dynamical ideas, as function is often tied specifically to a fixed scale.

In the foregoing sections, I've shown explicitly how the subagencies of one (super)

agent can be contained within its boundary, and even promised to others as a resource, by emission[113]. Agency thus exists and interacts in coarse, bounded 'packages', much like Milner's notion of bigraphs[Bur14, Mil09], and all the time on top of a substrate of basic adjacency promises that we call fundamental spacetime.

7.8.1 RESOLVING INTERIOR DETAILS OF SUPERAGENT STRUCTURE DURING COUPLING

Every time an observer zooms out by coarse graining, the detail wiped out by the formation of a grain can be captured as a map called a *directory* or *index* (see discussion in section 5.8.10). Preserving this map can help an external agent to resolve and interact with the the subagencies inside a superagent's boundary[114].

This is paradoxical: in order for an agent to exchange promises with a superagent at scale M (which has no physical boundary or form other than its constituent parts), an external agent perceiving the effective promise at scale M would surely have to be directed to an available subagent to provide the directory is available to the external agent.

Example 164. *A bank superagent might promise to give you cash, but the promise still has to be given meaning and carried out by an actual bank teller. The bank itself has no interface to bind to without its subagents.*

As a fictitious boundary, a superagent could simply be imagined by the assessments of an observer, with no coordinated intent of its own. However, a promise made to or by a superagent has to be a promise made to or by its components somehow. If a promise binding is only conceptual, this might not be an issue, but if an actual transfer of information is implied, there must always be a real source and a real receiver.

Definition 162 (Transparency)**.** *An agent may be called transparent if it promises an index or directory of all its internal subagents and their promises.*

In the remainder of this section, we address how such a coupling of agency scales, given what we know about irreducible promises (see section 5.8.13), from the dual perspectives of dynamics and semantics.

7.8.2 DISTRIBUTION OR DISPATCH OF PROMISES AT SUPERAGENT BOUNDARIES

So, what happens at boundaries when a promise is made to a superagent? What happens to the information? This is not defined *a priori*, but we can promise an answer. Let's

try to answer the question dynamically first, since everything is dependent on what is dynamically possible.

When we make a new promise at scale M to a superagent, we need to understand what this means for the component subagencies at a finer-grained scale within it. Two possibilities present themselves:

- *Distribution/flooding (broadcast)*: Promise bindings made to a superagent are broadcast or diffused throughout the subagents that comprise it, spanning multiple agent locations, like the behaviour of a gas or fluid *flooding* into contact with an interface.

- *Direction/dispatch (switched)*: Promises are routed to a subset of subagents, or representative binding sites, in a solid state, making an exterior use-promise on the surface is responsible for accepting the promise. The routing can be direct from the promise to the interior subagent (if the superagent exposes its directory), or it can be made via a proxy routing agent inside the superagent (if it exposes only a gateway).

Why and how these possibilities should happen at all merits some further discussion. The details almost certainly depend on the scale and context. The generality of the questions (and the occurrence of examples in the natural and technological worlds) is what makes them most intriguing.

Example 165. *Consider an example of an extended superagent* $\{a, b, c, d\}$ *bound by some cooperative promises, which we neglect to mention here. These may occupy a space of similar extent* $\{A, B, C, D\}$*, as in figure 7.12. This scenario is a realization of many possible scenarios, e.g. a journey in many legs (plane, train, network routing), in which the promise of multi-tenant sharing of several sequential host resources forms a journey in which several hosts have to cooperate as a superagency (an inter-network cloud) (see figure 7.12). A traveller, (a tenant of the journey) has to be authorized for passage by each stage of the journey. This requires promises to authenticate credentials to be distributed throughout the path, and the collaboration of the hosting agencies in trusting the credentials.*

Example 166. *Directing or dispatching promises, through a specialized agent, is like using a reception desk, service portal, in an office or hotel. Routing of information requires the underlying adjacency infrastructure to be able to direct messages to particular addresses.*

We may now state the two methods formally, for clarity:

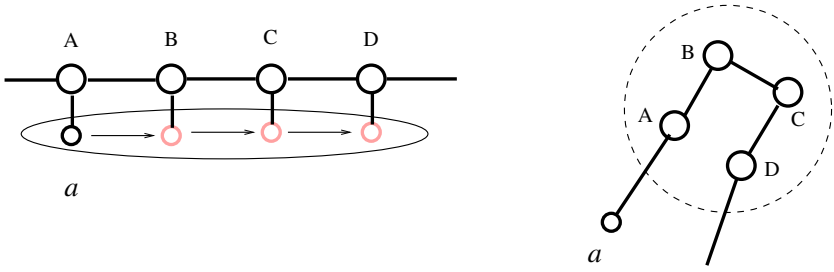


Figure 7.12: Path tenancy. The circled agents form a single superagent that occupies the corresponding space in the line above. The circles region is simply a superagent with exterior promises at its end-points. An agent binding to each site in transit can only do so at the scale of the subagents but the tenancy binding can be made as a distributive promise to the superagent.

Definition 163 (Distributive promise, at scale M (flooding))**.** *A promise made to a superagent A_s*

$$A \xrightarrow{+b} A_s \tag{7.27}$$

is assumed made to all agents within A_s:

$$A \xrightarrow{+b} A_i, \quad \forall A_i \in A_s. \tag{7.28}$$

The agents A_i voluntarily accept the promise, if they are suitable recipients, hence selecting by brute force rather than intentional labelling.

Example 167. *In information technology, flooding is used to make a 'bus architecture'. Ethernet and wireless transmission are examples.*

Definition 164 (Directed promise, at scale M (dispatch))**.** *A promise of type τ, made to the superagent, is assumed directed to a named subset of (one or more) members, on behalf of the entire superagent.*

$$A \xrightarrow{+b} A_i, \quad A_i \subset A_s. \tag{7.29}$$

The subset A_i voluntarily accept the promise, if they are suitable recipients, which we may assume is likely, given the intentional direction.

Example 168. *In information technology, dispatch to a directed address is used in queue managers, like load balancers, or memory and storage devices, to route data to a labelled destination.*

Stating these methods does not imply that they are possible in all cases. To understand whether diffusion of promise information is realizable we need to understand the small scale adjacency structure of spacetime, and its effect on promise scope.

7.8.3 TRANSPARENT ADJACENCY

Can promises, made by an exterior agent, reach all the internal subagencies in a superagent, then be comprehended and accepted? A promise made to a superagent has to be transmitted along the network of underlying adjacencies.

Both dispatch and distribution approaches to dissemination and binding assume that promises can be made directly between the subagents of neighboring superagents. The communication needed to make and keep such promises depends greatly on the network substrate of adjacency made at the lowest spacetime level. So the question becomes one about how spacetime adjacency is wired (see figure 7.13).

Example 169. *To visualize promises made at coarse grained scale, imagine a water authority that promises electricity to a town. Both these agencies are superagents composed of many subagents. Where (which agent) does the promise come from, and who receives it? What adjacency allows the promise to be transmitted?*

A generic promise made in the name of the company, depending on its legal department might make the promise. Every resident in the town is a potential recipient, as long as they can receive the information directly or indirectly, i.e. as long as they are in scope. The adjacency might be by postal communication and by water pipe.

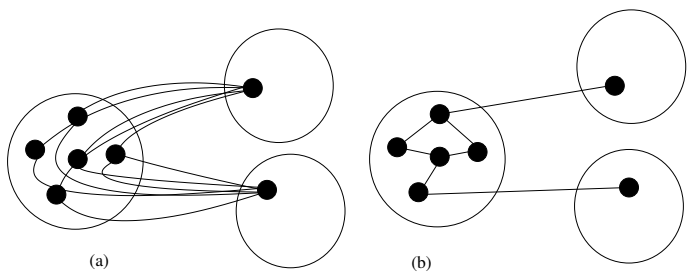


Figure 7.13: Regardless of whether promise diffusion has the semantics of flooding or directed dispatch, one is limited by the actual adjacencies that mediate communication in the space. In (a) agents are directly adjacent or 'patched' to all promisers of a particular type, and hence the promise diffuses naturally. In (b) adjacency is only to specific 'front desk' agents, who must then arrange adjacency virtually by proxy.

In figure 7.13a, the external agent promising to a superagent is directly adjacent to every subagent inside it. In figure 7.13b, the external agent only connects to a binding site. How these adjacencies come about, in practice, depends on the phase of the agents. There are two possibilities:

- Agents in a disordered gaseous state (no long range order), agents have no prior knowledge about one another without random walk meetings, and binding to one another. Thus discovery is a kind of Monte Carlo search[115], and communication is like broadcasting or flooding with messenger agents.

- Agents in an ordered phase, can assign fixed coordinate locations which can be indexed and used to access agents by design. This requires a mapping in the index between promises and locations, and adjacency between the index and each agent inside the superagent boundary. Thus an index or directory service must act as a switch, routing promises to intended destination.

  This, in turn, can be done in two ways:

  - By exchange of agent contents at the boundary, and exterior lookup
  - By encapsulation of agent contents and routing with interior lookup

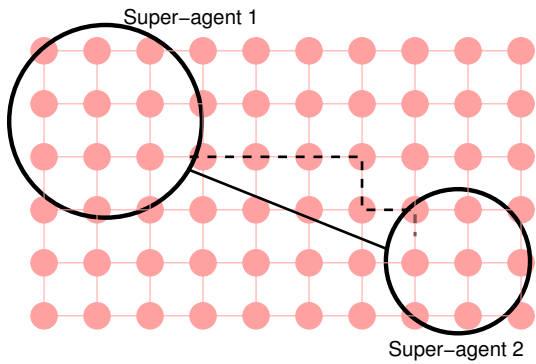


Figure 7.14: A schematic 'virtual' coarse grained view of promising, on top of the real adjacencies between subagents. If adjacency is mediated through intermediate adjacencies, the entire structure has to support the dispatch and/or flooding of messages to keep promises through the substrate of true adjacencies (whether gas or solid).

7.8.4 SEMANTICS OF PROMISE SCOPE FOR SUPERAGENCY

The semantics of scope need to be clarified during scaling, since scope represents the boundary of information about a promise's intent, and the ability to distribute a promise depends on the underlying adjacencies of spacetime. If two agents are not adjacent, they might not be able to occupy the same scope.

Lemma 35 (The scope of a promise to a superagent)**.** *Consider a promise made to a superagent S_M at scale M:*

$$\pi : A \xrightarrow[\sigma_M]{b} S_M. \tag{7.30}$$

Without the coarse graining directory $\pi_{\rm directory}(S_M)$ for superagent S_M, the scope of π is only defined to the boundary of S_M. It is not possible to say which subagents of S_M are in scope of the promise. If access to the directory is promised to the promiser:

$$S_M \xrightarrow{+\rm directory} A \tag{7.31}$$

$$\sigma_M \to \sigma_M + \sigma_{\rm directory} \tag{7.32}$$

If agent S_M promises access to its coarse graining directory, an external agent A can infer the scope of a promise in terms of the subagents of S_M.

The promise could be visible to all the subagents $A_i \in S_M$, which are adjacent to A, as well as the extra scope σ. However, a promiser can say which agents are reachable by a promise message if and only if the directory $\pi_{\rm directory}(S_M)$ is available.

The implication of this is that promises do not scale automatically by replacing an agent with a superagent: the scope of a promise made to a superagent is not necessarily distributive, because of the loss of information in coarse graining.

7.8.5 COUPLING TO A SUPERAGENT BOUNDARY (GATEWAYS AND ADVERTISEMENTS)

If all actionable agencies are concealed behind a superagent boundary, how can any promise message reach them from outside? The answer is often a role known as a gateway agent. The subagents are the agencies that must ultimately act to keep any promises of the superagent. As seen in section 7.8.3, the underlying adjacency of spacetime remains, even under the coarse graining of superagency, though it might not be visible inside a boundary, without the help of a specialized role. A promise could thus reach the subagents by flooding, or by direct dispatch of a gateway, following the adjacencies within the boundary, provided sufficient adjacency exists. Dispatch can be directed by the promiser (if transparency is granted to an external agent by access to the coarse graining

directory) or by an intermediary agency (acting as a relay gateway) within the superagent. It is assumed that there are no changes to agent semantics simply by aggregation:

Assumption 6 (Promisee autonomy is preserved in superagency)**.** *Once a promise reaches a subagent, it is up to the subagent to accept the promise or not, and behave accordingly, unless it has voluntarily promised to subordinate itself to another agent. Even if the promise is transmitted as an imposition, autonomy can never be violated.*

Thus the scaling of cooperation remains does not change the rules of autonomy; the 'voluntary cooperation' assumption of autonomous agents persists. It is up to subagents to use any promise made to them. However, this does assume that they are in the scope of such an external promise. Hence to ensure the coupling of an external agent to a superagent, transparency has to be restored or relayed. Under coarse graining, a superagent is seemingly replaced by only its boundary, possibly with a promise to access its index/directory of interior information. Promises from external agents outside the boundary can only refer to the superagent as promisee or body-tensor coordinate. How then would the internal subagents know what to do with the promise? How do they find one another?

Example 170 (Cloud computing clustering and receptions)**.** *In human organizations, the gateway between outside and inside is a reception desk, or even a secretary. This provides a single point of contact, whose role it is to route messages internally and vice versa. In cloud computing technology, users rent clusters of machines that are dispersed with complicated internal addresses, which are not exposed to the outside world. A gateway agent, often a directory service, such as LDAP, key-value store (Consul, etcd, Zookeeper), or DNS. The gateway acts as a tranducer, coupling a single visitor to a larger cluster, and routing a process on its own scale within the larger entity. Another example of a gateway is a connector plug on a device, or a power socket that connects a low level process to a larger system serving many.*

In order for one scale to couple to another scale, we can introduce the idea of a gateway—which plays the role of scale transducer (see figure 7.15): a combination of necessary and sufficient conditions for promises made *to* a superagent boundary, to be resolved without mentioning any subagencies.

Example 171 (Radio as a superagent)**.** *What part of a radio makes the exterior promise of being a radio, rather than a collection of electronic components? Whether the device is switched on or off, its function is not clear without a promise. The agency that explains this is usually the packaging of the radio, i.e. the casing. It might further be packaged in a box, but that is not a part of the radio itself. In this case, the enveloping a casing becomes an agent with a promise that tracks the superagent boundary*[116].

Definition 165 (Scale transducer). *Let A_s be a super agent at scale $M = \{A_s\}$, and let π_s be a promise made A_s, by any agency A (see figure 7.15). Recalling that the scope of a superagent includes all agents inside it, and coarse graining limits scope (see section 5.8.11), we define a scale transducer by:*

1. *A number of promises addressed to the superagent boundary, along exterior adjacencies.*

2. *One or more subagents that make exterior use-promises, within the superagent, to accept the promise made to the collective:*

$$\exists A_i \in A_s : A_i \xrightarrow{U(\pi_s)} A. \tag{7.33}$$

3. *An index for directory $\pi_{\rm directory}$ is promised externally to agents outside the superagent boundary. This provides them with information about how to address their promises.*

4. *If there is more than one agent adjacent to the superagent, i.e. more than one possible agent that can make promises with the exterior, these agents can be located through the directory.*

5. *One or more subagents in A_i play the role of gateway and dispatcher, to conditionally forward messages to interior agents with matching use-promises. This is a standard entry point for the agent, e.g. it is the agent adjacent to all exterior agencies, thus forming a skin or boundary between exterior and interior agencies.*

$$\exists A_i, A_j \in A_s : A_i \xrightarrow{U(\pi_s)} A \tag{7.34}$$

$$A_i \xrightarrow{+\pi_s|\pi_s} A_j \quad i \neq j \tag{7.35}$$

In this case, the superagent must advertise the location of one or more gatekeepers, entry-points or directory agencies that promise to relay information.

   (a) *The subagents inside a superagent boundary of a given type τ are advertised in the index/directory $\pi_{\rm directory}(\tau)$. They are symmetrical with respect to* what *they promise (promise type τ), but they might not necessarily be equivalent in* how much *they will promise (their promise bodies might differ in all details except the type). This means that the directory must advertise any promises to this effect also.*

   (b) *If a gateway is used as a proxy relay (see figure 7.15a), it must additionally make promises that select promisees from the subagents inside A_s, e.g. policies might be distributed (by flooding) or directed by dispatch.*

   (c) *Gateway agents might also need to translate between the language β_s assumed for the superagent, and the language(s) of recipient subagents β_i, as part of transducing through the opaque boundary.*

We may note that messages from an external agent might be forcibly constrained to a particular route by spacetime structure, i.e. by a limited adjacency (a bottleneck, as in section 7.8.3). Also, a gatekeeper need not be a single agent, or even a localized cluster in the role of gatekeeper. It could, itself be a fully distributed collective agency, embedded within a specialized set of subagents, and coordinated by mutual cooperation. e.g. like a cellular skin. This idea of exterior agents binding to specialized 'docking sites' leads us naturally to consider the idea of *tenancy* in semantic spacetime structures. Indeed, I'll return to this later in these notes.

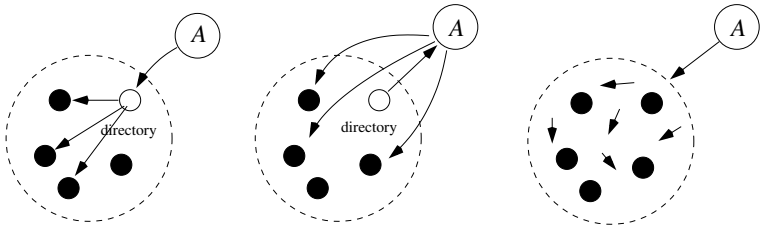


Figure 7.15: Scale transduction for incoming promises to a superagent: (a) the coarse graining directory is internal to the superagent and the external agent makes its promise to a gateway 'receptor'; (b) the coarse graining directory is exposed and makes the superagent transparent to the external promiser so that it can promise directly to the subagents; (c) there is no directory, and internal information is lost. Promises are flooded to all subagents, and may or may not be picked up. The efficiency of flooding depends on the solid or gaseous state of the superagent.

From the list of requirements for transducing promises between scales above that the following corollary applies:

Lemma 36 (Dynamical requirements for coupling between external agent and superagent)**.** *Any agent A can probe scale information in a superagent, at scales finer than its boundary, provided the superagent promises its coarse graining directory:*

$$\pi_{\rm directory}(S_M) : S_M \xrightarrow{+{\rm directory(S_M)}} A \tag{7.36}$$

$$\overline{\pi}_{\rm directory}(S_M) : A \xrightarrow{-{\rm directory(S_M)}} S_M \tag{7.37}$$

Alternatively, we can say that a coarse grained agent S_M may be made transparent by promising its coarse graining directory.

Example 172. *To couple to a single interface in an electronic device, there has to be an exterior promise to bind to (e.g. USB), and an address of the agent inside.*

Example 173. *In order to affect a nucleus within an atom (e.g. NMR), a field basically floods its promise to all subagencies blindly, hoping to excite the resonance (receptor use-promise).*

7.8.6 ADDENDUM ON SCALING OF SCALE TRANSDUCTION ITSELF: QUEUE DISPATCH

Since scale transduction is itself a dynamical process, dependent on underlying spacetime, its efficiency is also subject to scaling issues. Consider first the coupling issue from a semantic perspective, of interfacing instead. When a promise is made to a superagent boundary (as in equation (7.30)), the promise information has to go to an agent that is listening. A superagent is not such a real agent, it is only an abstraction. This suggests that, in the case of superagents, the boundary itself might be represented by an explicit agency that can perform routing and forwarding of messages between the superagent's fictitious boundary and its subagencies.

While this direct dispatched routing of promises makes sense semantically, dynamically, the idea seems contrary to the notion of scaling: to replace a scaled mass of agents by a single gatekeeper or router creates an obvious bottleneck and fragile dependence. This is the cost of coarse graining, especially in a discrete spacetime, and it suggests that the grain size should never become too large, else a superagent becomes hindered by interfacing issues.

Example 174 (Queue dispatcher)**.** *In queueing theory, a dispatcher is an agent that processes a queue of incoming messages and routes them to a service agent[Kle76, GH98]. Load balancers introduced into networks as 'middle boxes' are single-agent dispatchers to multiple subagents within an superagent of servers. These middle boxes break the equivalence of the system under re-scaling, by forcing all adjacency through a single route.*

Example 175 (Reception desk)**.** *A reception desk at a company accepts promises and information on behalf of the collective organization of subagents. An agent works at this desk to receive messages, and dispatches, routes or forwards these messages to other relevant agents using an internal directory for responsibility.*

An alternative is for the superagent boundary to promise flooding contact and allow the surface agents to coordinate internally, as redundant gatekeepers, each deciding how to resolve what happens if multiple agents receive the same message. If promises are not, by their nature, exclusive to a single gateway, then coordinating exclusivity adds N^2 complexity of promise coordination.

Example 176. *Using the coarse graining directory to give transparency, any external agent could perform its own dispatching / load sharing without loss of scalability. For example, a directory service used by software could replace 'middle box' load balancers in software, with the help of a software interface used to select a specific subagent server within the superagency of all servers.*

Let's summarize the implications of loss of scope from the past previous section.

- Routing of messages to internal agents through a single gateway breaks scale invariance.
- A scale transducer may be introduced, for mediating interactions directly by granting transparency, using a directory.
- Directory promises allow external agents to look up lists of subagents encapsulated within enabling scale transduction, i.e. a kind of microscope for crossing the semantic scale boundary.

7.8.7 ADDRESSABILITY IN SOLID STRUCTURES, AND THE TENANCY CONNECTION

To see how we can route messages back to the specific subagents, we need to understand addressing. The ability to give every agent a predictable address, and then be able to have messages forwarded to it uniquely, using that address, depends on a number of promises being kept. To illustrate this, let's construct a semi-lattice by iterating a simple asymmetric message pattern.

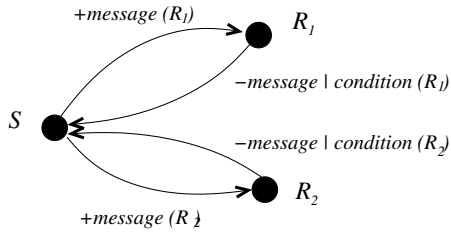


Figure 7.16: Two promise bindings, leading from a source agent S to two different recipient agents R_1 and R_2. This forms the basis of a routing structure for address decomposition, where part of an address can lead to selection or rejection of a particular route.

Let S_i and R_i be two types of roles for a set of agents A_i, and consider bindings between two kinds of promise:

- A vector promise to dispatch messages to a recipient agent R:

$$S \xrightarrow{+\text{dispatch message to R}} R \tag{7.38}$$

- A use-promise to accept messages from a source S, only if its address is compatible with the agent's conditional expression for forwarding:

$$R \xrightarrow{-\text{message}|\text{addressed to me}} S \tag{7.39}$$

Building on these two promises, we may construct uni-directional adjacency-like binding for use as a template to build larger structures.

In the figure 7.16, we see a node with two such promise bindings sporting different conditionals in different directions. Notice how the choice to forward a message from S to R is a voluntary act by S. It can send in different directions to purposely separate messages, or it can send along different paths for traffic management or load balancing. Note also that the difference between a flooding promise (sending messages to all recipients) is simply a scalar version of the dispatch promise, in which we take away a target from the promise body, i.e. without exclusivity to promisee/body vector. The result is the same, but the efficiency is compromised; efficient routing is assisted by long-range cooperation, and ultimately by long-range order.

The receiver R, has the last word in accepting a message. So no message will arrive at the wrong location no matter whether it was forwarded by targeted dispatch or broadcasted to all agents. Agents have to promise their unique identities, both so that they may be recognized by neighbours, and so that they can recognize messages directed to them.

The consequence of creating an ordered tree from these promises is to create a dumb filter, which routes messages along a unique path depending on their address.

Example 177 (Coin sorting)**.** *Coin sorting machines create unique pathways the sort and select different sized coins, allowing the to roll only one way through a maze of pathways. This is the basic principle of a semantic sorting process. The pathways for a treelike structure, and the end points of the tree all have a unique address. By placing a coin of a particular kind into the process at the root, it is like placing a message with an address (the type of coin), and having it sorted until it reaches its destination. Coins with the same address will end up at the same location.*

Armed with this tool for spatial sorting, we may iterate these promise patterns to generate semi-lattices of greater size. Having a coordinate system within a superagent boundary, for example, would allow agents to be located in a targeted manner, assuming only that they are connected. To iterate the pattern, we simple make each receiver into a source for the next iteration, and so on.

Example 178. *Figure 7.17 shows to iterated patterns formed from branching source-receiver iteration. The first (a) is a simple tree structure. Every leaf node of the tree has a unique address, and can be reached from the root by a unique path. This is the property of trees (and spanning trees).*

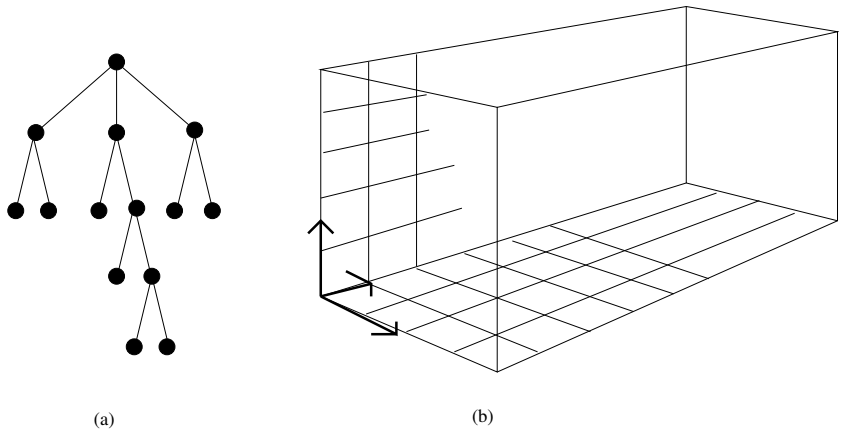


Figure 7.17: Iterating a pattern of promise bindings over a spanning tree allows unique labels to be associated with destinations in a graph. The symmetry can be a minimal tree as in (a), or, by adding redundant links it can be turned into a semi-lattice (b). Arrows are not shown for simplicity; however, both structures are uni-directional.

The second case (b) is also a tree formed from three-way branchings $S \xrightarrow{+b(R_1,R_2,R_3)} \{R_1, R_2, R_3\}$*, iterated homogeneously and isotropically. The agents then fall into a three dimensional, Cartesian arrangement, which we may call the Cartesian semi-lattice. By filling in some redundant promises from each point, one can arrange multiple routes from any node to any other, but still only in one direction (radially outwards from the origin).*

By making the promises in each of the three dimensions sort forwarding of messages according to a different component in a vector tuple, forwarding can be encoded as a purely local operation[117]*, e.g. forward only if the tuple value is greater than the current tuple address of the agent for the current lattice location. Furthermore, by completing the reverse direction, as a mirror image, with opposite semantics, addresses can also be navigated in the opposite direction, completing the lattice.*

7.8.8 CONDITIONS FOR A UNIFORM COORDINATE COVERING OF AGENTS

What are the conditions for being able to address agents using contiguous coordinates without loss of locatability? This is a slightly different question to the one about naming and promise body continuity, because it requires us to preserve the partial ordering of agents in a lattice. It is helpful to explain addressability by introducing two concepts that cover the semantic and dynamic aspects of location:

Definition 166 (Semantic addressing)**.** *An agent is said to have a semantic address if it is labelled only by a tuple of names that do not form part of an ordered pattern. Semantic addresses contain no relevant kinematic or dynamical information about an agent's location in a space.*

Semantic addresses act only as sign-posts, and a directory is needed to map which adjacency will eventually lead to the named agents. This is the approach used in Internet routing.

Example 179 (IP addresses)**.** *Internet addresses (aka IP addresses) are semantic addresses, despite being composed of numbers, because they have no requirements of spatial order. Any agent can assign itself an address with any number, and these numbers do no not imply information about where agents can be found. In order to locate IP addresses, a directory called a routing table is needed, which maps the random numbers of the addresses to physical adjacencies of the cabling. This is the function of a router or switch. The advertisement of these local directories is performed by services, which are called routing protocols (BGP, OSPF, RIP, etc).*

Definition 167 (Numeric (metric) addressing)**.** *An agent is said to have a numeric or metric address if it is labelled by a tuple of values in which each value map to a unique integer. Numeric addresses represent kinematic or dynamic information about a space.*

Agents with numeric addresses are partially ordered in a multi-dimensional lattice. The addresses form a coordinate system in the usual sense of mathematics.

In lieu of setting up a proof, I'll hypothesize this informally, as an assumed lemma, for now:

Lemma 37 (Conditions for a uniform coordinate covering of an ensemble of agents)**.** *Possible placement of fixed and ordered address labels on an ensemble of agents, in a voluntary cooperation structure include:*

- *Fixed locations.*
- *Addresses ordered by location.*
- *Promises to sort and relay messages to destination.*
- *Long range order in address promises, i.e. cooperation in behaving uniform sorting/routing to addresses.*
- *Overlapping regions of β-language relevant to address*

This does not refer to any particular topology, so it can be solved by multiple adjacency patterns. A suitable address-sorting process can be satisfied in a number of different ways, reflecting the encoding of the address in relation to the structure of spacetime. Network structures are typically tree-like, and IP addresses are prefix-based with distributed routing tables based on tree branching assumptions. Toroidal structure and Cartesian lattices using tuples are based on a pre-ordered layout, as in a warehouse, for instance.

Example 180 (Network partition)**.** *In networking, regions of a network (like the Internet) can become cut off from the rest by a loss of routing information. Because the Internet is a gas, with semantic rather than numerical (metric) addressing, each superagent boundary must contain a routing table that points to the next signpost to the destination. To prevent this from getting out of hand, a 'default route' is normally used as a wildcard (go this way to find any unspecified agent $A_?$), allowing regions to compress information about how to reach non-local agencies by handing off to centralized routing hubs.*

There is an interesting suggestion here, that semantic naming tends to favour the formation of a hierarchy in order to scale. Such a hierarchy is unnecessary for metric naming. This warrants further study.

7.8.9 EFFICIENCY OF ADDRESSING IN A SEMANTIC SPACE

Consider a network of agents, with unique names and which are all interconnected by a sufficient number of adjacencies to allow full percolation. Suppose these agents promise to cooperate in relaying messages to one another, by passing the message along one of their adjacencies until it reaches the unique name (i.e. address). How much information has to be available to each agent in order to know how to forward messages to every other agent?

The maximum size of directory information may be computed as a sum over every location, which keeps a table of every other named agent in the space, paired with the 'next hop' neighbouring agent that brings the message closer to its destination. This applies for every agent in the space.

- If every agent is independent, then every agent needs a list of all $N-1$ agents, with

  $$(\text{Agent name, direction of agent}) \tag{7.40}$$

  and a direction in which to forward. In total memory required for this information is of order $N(N-1)$ in the number of agents. If there are few agents, this is easy. If there are many, the search cost rises linearly, and the distribution of information by flooding brings high cost.

  Example 181. *This is exactly like routing in the Internet, imagining there are no network CIDR summarization prefixes, which would correspond to superagent boundaries.*

- If one can replace atomic agents with superagents, which can handle their own internal forwarding, then the amount of information one needs to exchange is less.

  $$(\text{Superagent of every node, direction of superagent}) \tag{7.41}$$

  Aggregation of clusters leads to a cost of order $(N-1)\log N$ memory. Scaling of addresses now depends on the ability to delegate responsibility to agents 'further down the line' by using superagent container names to route messages, as in the coin sorting machine or lattice. This leads naturally to hierarchical naming and routing.

  Example 182. *Postal addresses refer first to town, then street, then building, and so on. By referring to larger container boundaries first, one can delegate the detail of finding the final destination to agencies within the boundary of the superagency, e.g. the town. This assumed encapsulation comes at a price, however. It introduces inter-dependency into the end-to-end communication.*

  Today, postal addressing also now uses metric post codes, which are non-hierarchical. Given modern computational resources, a simple brute force approach can be used to look up these codes from directory information.

  Example 183 (CIDR prefixes)**.** *This is like IP routing with CIDR prefixes. Internet (IP) addresses were originally designed to reduce routing cost by aggregated along certain prefixes, originally of fixed length (called class A, B, C networks). By*

grouping addresses under a smaller number of prefix patterns, and assuming that all such addresses were contained in the same superagent boundary, routing tables could be kept small. Later, as these limited prefixes became consumed, they were subdivided into more, causing routing table growth.

- If there is a regular lattice, e.g. (x, y, z), with long range order, and tuple addresses, then the amount of information is now of order 1. It is like asking which way is 'up'? Irregularities (like holes) can be routed locally at no extra cost.

  Example 184. *For a Cartesian lattice, one knows left or right, forwards or backwards for each address because of the ordering of the integers.*

Delegation, or deferred evaluation, is an attractive idea for scaling linearizable searches, however we must note that, an agent cannot ask another agent for help without already being able to know how to reach it; so, with no basic pattern to compress by, there is no way of centralizing this routing information in the manner of a coarse graining directory.

Consider the following worst-case scenario in which every agent has a random name, i.e. a spacetime addressed by random numbers. Then, every location has to have a complete map of every other location, with zero possible compression. The need to flood all that information to all parts of spacetime adds a significant cost to promise keeping, and might exceed the capabilities of any or all agents.

Example 185. *Instead of handing out metric addresses to visiting mobile devices (as a parking lot, or hotel, would do to its visitors), the Internet hands out local semantic addresses (by DHCP), and tries to map them into its routing infrastructure. This makes sense for ephemeral gas-phase devices, but is quite inefficient for the re-purposing of solid phase agents, like virtual machine slots, or process containers.*

In practice, we see, from the stages of address scaling above, that the information is compressible only if each agent can replace a collection of addresses with a single promise. Hence to coarse grain addresses into a hierarchy of containers, without loss of information, we need to restore the information lost using a directory at each superagent boundary[118]. If we want to keep directory information small, we need *long range order* in the structural addressing promises (and presumably the adjacencies too) to enable logarithmic aggregate summarizability. Asymmetric tree structures can be adequate, but bi-directional lattices, like a Cartesian lattice, are better still[119].

7.8.10 SUMMARY OF AGENCY PROPERTIES IN SEMANTIC SPACES

The scaling behaviour described thus far allows us to 'inflate' (or scale-up) any functional arrangement of promises by substituting an arbitrary agent with a superagent composed

of subagents making similar promises. Then, one may define the exterior promises in such as way as to integrate the subagent members seamlessly to agents on the outside of the superagent boundary. There is a progression:

$$\text{agent} \rightarrow \text{superagent} \rightarrow \text{role} \rightarrow \text{subspace} \tag{7.42}$$

In other words, as an algorithm to scale given a single agent, we replace it with a black-box superagent. Then we proceed to fill it with multiple subagents that are connected to the outside agents by exterior promises. These similar subagents are symmetrical with respect to the outside, so they form a role by association. Eventually, as we scale each of the original agencies and connect them to scale the promises, what remains is a set of non-overlapping subspaces, one for each agent, embedded in a larger semantic spacetime[120].

Example 186 (Gated community)**.** *Namespaces, walled/gated communities, zones of privilege, service providers, etc are examples of agencies which scale from a single agent to collections bounded by some kind of contact surface.*

Semantic spacetime (agents) have a number of scalable properties:

- They have discrete languages of intentions, easily translatable, in order for promises to be effectively communicated.
- They can be observed and interpreted at a multitude of scales, at the behest of an observer.
- They can effectively cluster their own promises into coarse grains, through cooperation.
- The number of agencies can grow or shrink, i.e. spacetime itself grows or shrinks, as new points are added or retired.
- There are simple rules for transforming from one agency scale to another, analogous to renormalization transformations.
- Promises behave like tensors in general, with directionality.
- Causal influence is passed by vector promises, and principally through use-promises, by the principle of autonomy.
- Biological organisms offer a useful measuring stick for spacetimes with strong semantics.

So far I've focused on preserving symmetries and semantics, while piecing together the underlying connectivity of space. The asymmetry in these promises for routing to fixed addresses has a general utility, and it can be associated with the idea of *tenancy* (next section). Functionally, this asymmetry is the most important tool for making anything happen in space or time, and is worth exploring in more depth.

7.9 OCCUPANCY AND TENANCY OF SPACE

Let's now turn to a different topic: how to fill the space we've built up. A semantic space is richer in structure than its underlying connective graph so it contains information that goes beyond pure adjacency. In particular, as we add autonomous observers with their own agency, we quickly arrive at the need for agents to extend their realm of autonomous control through *occupancy* and *ownership* of resources. The question of occupancy and tenancy are thus about how we draw the boundaries of agency on a background of spatial adjacency So far we've focused on symmetry and scale in discussing agency, however strong functional semantics are a result of asymmetry, hence we must now pursue the effects of broken symmetry.

7.9.1 DEFINITIONS OF OCCUPANCY AND TENANCY

Tenancy goes beyond simple aggregate membership in a cluster. A tenant is understood to be an agent that 'occupies' or utilizes a resource or service, provided by a host, often in a temporary manner, and for mutual benefit (symbiosis). Tenants have separate identities. When we think of tenancy in every day affairs, we do not usually imagine a tenant as merging with its host, and becoming a part of it (though merger and acquisition is certainly a process one can discuss, as absorption). Tenancy is rather an association between two separate agency roles (host and tenant), each of which retains its autonomy.

To relate this to our spacetime discussion, consider the following question (which, at first glance, might seem purely facetious): *does a suit occupy space when no one is wearing it, or does space occupy the suit?* The space inside a suit is simply empty before someone climbs into it. Try replacing 'suit' with 'car' and 'wear' with 'sit inside'.

This peculiar question is closely related to the considerations surrounding the kinds of motion described in [Bur14], section 5.12. Since we are modelling space as a resource, this is not only a meaningful question, it is essential to understand what kind of volume a suit occupies. Does the presence of suit matter replace space, occupy it, attach to it, or overlap with it? These have different semantics[121].

To address some of these issues, we need to formulate definitions using promises, building up the distinctions in a rational way. Let's begin with occupancy. Its semantics

are difference from mere presence, as there is an assumption of valency. Within the scope of promise theory, we can define the following:

Definition 168 (Occupancy)**.** *An asymmetric association of one agent (the occupier) with another representing a host (the location) at agency scale M, in which the valence of a promise make by the host is reduced by its binding to the occupier. The resource R may be any scalar, vector or tensor type:*

$$\text{HOST}_M \xrightarrow{+R\#n} A_? \tag{7.43}$$

$$\text{OCCUPIER}_M \xrightarrow{-R\#1} \text{HOST}_M \tag{7.44}$$

In other words, a host makes a finite promise $+R$ to a number of agents in scope, and each occupier reduces the valency by making a use-promise $-R$

$$\mathbf{Valence}(R, \text{HOST}) = \mathbf{Valence}(R, \text{HOST}, \text{OCCUPIER}) + 1 \tag{7.45}$$

Example 187. *Our understanding of the semantics of occupancy has many possible interpretations. Here are some examples:*

- *Occupation of a territory without necessarily being there. e.g. a table reservation.*
- *Occupation of space.*
- *Occupy a car, a suit, a dress.*
- *Occupy a time slot in a calendar.*
- *Filling a space with something.*
- *In physics, bosons can occupy the same space, like voices in a song, but Fermions have exclusion, like the bodies in the choir themselves they occupy space.*

From here, we may state a basic template for tenancy for application to a variety of special cases:

Definition 169 (Tenancy). *Tenancy refers to the conditional occupancy of a location, by an agent, together with the provision of one or more services by the host, which may be considered a function $f(R)$ of resource R. These services are provided conditionally on a promise of C from the tenant:*

$$\begin{aligned}
\text{HOST}_M &\xrightarrow{+R\#n|C} A_? \\
\text{HOST}_M &\xrightarrow{-C} \text{TENANT}_M \\
\text{TENANT}_M &\xrightarrow{+C} \text{HOST}_M \\
\text{TENANT}_M &\xrightarrow{-R\#1} \text{HOST}_M \\
\text{HOST}_M &\xrightarrow{+f(C,R)|-R} \text{TENANT}
\end{aligned} \tag{7.46}$$

This is the basic template for tenancy, which may be extended by additional promises.

Example 188 (Landlord). *A landlord promises a rentable space for a single occupant $+R\#1$, conditionally on the signing a contract of terms (i.e. the promise to abide by terms and conditions) $+C$.*

$$L \xrightarrow{+R\#1, f(C,R) \mid C} A_? \tag{7.47}$$

A tenant quenches this exclusive resource, by signing up and promising the terms:

$$T \xrightarrow{+C,-R} L. \tag{7.48}$$

The terms and conditions contain a composite promise body, detailing the services $f(C,R)$ offered as part of the promise:

$$C = \{+\text{payment}, \text{termination date}, \ldots\} \tag{7.49}$$

and

$$f(C,R) = \{+\text{power}, +\text{heating}, \ldots\} \tag{7.50}$$

Tenancy is a service-like relationship between a host and a tenant. This may be contrasted with the notion of residency at a location, which is related to definition of boundaries within an observer's realm. Tenancy is a also relative concept (relative to promise semantics).

7.9.2 LAWS OF TENANCY SEMANTICS

It is basic to promise theory that we distinguish between a promise made by an agent, and the agency itself. Hence, we begin by noting that:

Assumption 7 (Promisees are independent)**.** *Promises are neither occupants nor tenants of the promisers or promisees, since they have no independent agency.*

- *Tenancy and occupancy requires two agencies to become associated.*
- *Agents can be promised, but promises are not agents (they do not possess independent agency).*

In a sense, a promise emanating from an agent seems to be attached at the location represented by the agent. However, we do not call this tenancy. A promise is a property of an agent, but it has no independent agency, thus it cannot be a tenant.

Example 189. *An agent A can be the subject of a promise, e.g.*

$$A_1 \xrightarrow{+A} A_2, \tag{7.51}$$

but it is not the promise itself, which belongs to A_1.

The semantics of the promises in (7.46) select an inherent directionality for the provision and use of a resource.

Lemma 38 (Tenancy flows in the direction of the resource being used)**.** *Tenancy flows towards the host, i.e. towards to source of the hosted resource.*

It is important to bear in mind the semantics when looking at host and tenant. Consider the following case in figure 7.18. From the perspective of a renter going directly to a hosting apartment block, the tenant

Assumption 8 (The host:tenant binding is 1:N)**.** *A host can have any number tenants, at any one time, keeping full promises, up to and including the valency of the host resource promise.*

There is an exclusivity between a tenant and a resource, which is a question of definition. Tenancy with a superagent scales like any other promise (see section 7.8.2). When we speak of a tenancy, it refers to a single relationship, even though an agent might be engaged in multiple similar tenancies.

Example 190 (Horse rider or jockey)**.** *A rider on a horse is a tenant of the horse. A rider cannot ride a herd of horses, at the same time. Moreover, the rider and horse are not joined by encapsulation, forming the embodiment of a superagent. A driver in a car however, is a tenant of the car, and is encapsulated by it. The car is a tenant of the driver's direction. Hence, while the two have independent agency, they seem to form an encapsulated superagent.*

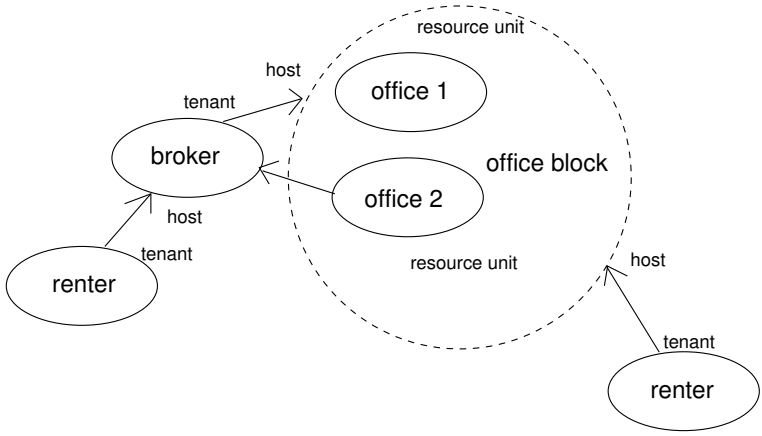


Figure 7.18: Identifying tenants and hosts correctly requires us to follow the tenancy law carefully. In each case, the arrows point towards to host resource sought by the tenant.(1) A renter may be a tenant of either an office block (providing multiple offices to multiple renters), (2) A renter may be a tenant of a broker (providing multiple client offices to multiple renters), (3) An office block or single office may be a tenant of a broker (offering multiple renters as a resource) to multiple office blocks or office.

Example 191 (OSI model)**.** *In the OSI network model, the layers from L1-L7 form a tower of dependence, in which network resources (at the bottom) are shared out between different applications and users which are tenants of the basic service. These layers farther up the stack depend on the lower layers, hence the arrow of tenancy points down to the L1 physical layer. L2 is a tenant of L1, L3 is a tenant of L2 and so on.*

Network encryption is a tenant of L3, and computing applications are tenants of the encrypted stream.

When these layers are implemented as encapsulations, the tenancy increases into the core of the encapsulation, and the host is the outside part. This seems to be the opposite of the way we are taught to think about networking, from a software engineering perspective.

Lemma 39 (Causation is partially ordered by pre-requisite dependency)**.** *Promises and intentions may be partially ordered by conditional dependencies, from the conditional promise law. This leads to a hierarchy of directional intent, for fixed semantics.*

We can distinguish tenancy from simple scaled agency by this partial ordering of tenants to hosts in the direction of a named resource. However, in most cases, the law

of complementarity of promises allows us to transform one tenancy into the reverse relationship interpreted as a different promise. In either case, the orientability of tenancy gives agents topological 'hair' which can be combed in a certain direction, as a vector field.

7.9.3 FORMS OF TENANCY

Let's look at some familiar exemplars to see how this general pattern is realized in different scenarios.

- **Club membership, or passenger with ticket**

  The issue of club membership is one where an agent associates itself as one of a group of typed agents: a vector promise directed to a specific host. The host offers the tenant a membership, and the tenant accepts the membership lease.

  $$C \rightarrow \text{membership fee} \tag{7.52}$$
  $$R \rightarrow \text{membership credentials} \tag{7.53}$$
  $$f(C,R) \rightarrow \text{benefits and services} \tag{7.54}$$

  Membership in a club is a label, i.e. a property of an agent. However, in the case that a separate agency validates this label as evidence of an association, we can view the members as guests of the hosting club. The condition C is typically some kind of subscription, the membership itself is promised with a badge or access credentials, and the additional services that accompany membership require showing of the credentials.

  If a club is exclusive, then the promise of $+R$ has finite valency, else it has infinite or unlimited valency.

- **Employment** An immediate corollary of membership is employment at an organization.

  $$C \rightarrow \text{work performed} \tag{7.55}$$
  $$R \rightarrow \text{employee status/badge} \tag{7.56}$$
  $$f(C,R) \rightarrow \text{benefits and wages} \tag{7.57}$$

  In this case, an employee is a tenant of the hosting company that pays for membership with his/her daily work. Tenancy is fulfilled by access or credentials (the company badge), and benefits include wages, lunch, travel costs, etc. Tenancy is always symbiotic, by nevertheless asymmetrical. The relative values of C and $f(C,R)$ are in the eyes of the beholders. When trading promises, what is valuable

to one party is usually not valuable to the other, else they would not be motivated to trade.

- **Privileged access (territorial access)**

  A further corollary is the use of credentials to gain access to territories, e.g. foreign visas, password entry, identity cards, etc.

$$
\begin{aligned}
+C &\rightarrow \text{identity credentials} && (7.58)\\
-C &\rightarrow \text{authentication/access control} && (7.59)\\
R &\rightarrow \text{access passport/visa} && (7.60)\\
f(R) &\rightarrow \text{territorial access/resources} && (7.61)
\end{aligned}
$$

- **Shared exclusive resource usage (multi-tenancy)**

  Now consider the case where we add a finite valency to a limited resource, as well as a condition of fair sharing. A fair sharing promise, up to a maximum valency of n, becomes an additional constraint on the host, of the form:

$$+R_j\#n \;\Big|\; \sum_i^n R_i \leq R, \tag{7.62}$$

  for each qualifying tenant TENANT_j, paying its tenancy cost C_j.

  See figure 7.19

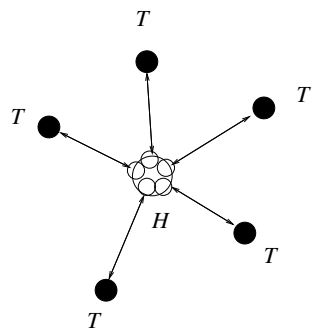


Figure 7.19: Tenancy of a singular resource by multiple agents. This is the same as membership, and containment. If multiple tenants occupy the same space, then the host effectively promises independent constituencies, so it has internal structure.

So the total promise set becomes:

$$\text{HOST} \xrightarrow{+R_j\#n \,\big|\, C_j\,,\,\left(\sum_i^n R_i \leq R\right)} \text{TENANT}_j \tag{7.63}$$

$$\text{TENANT}_j \xrightarrow{-R_j\#m} \text{HOST} \tag{7.64}$$

$$\text{TENANT}_j \xrightarrow{+C_j} \text{HOST} \tag{7.65}$$

$$\text{HOST} \xrightarrow{-C_j} \text{TENANT}_j \tag{7.66}$$

where $m < n$ by necessity due to the valencies. Services $f(C, R)$, based upon R might be subject to additional constraints, but they are also naturally limited by the constraint (7.62).

- **Representation by proxy** (spokesperson)

  In some cases a hosting agency's purpose is to be a proxy or representative for a client. This is the case for modelling agencies, writers' agents, sales representatives, public facing spokespersons, and even accountancy firms. Examples include 'Intel inside', goods on a shelf in the shop that represent their brands.

  In this case, the value added service to signing up is the representation of the tenant itself:

  $$+f(C, R) \rightarrow \text{TENANT representation} \tag{7.67}$$

  Representation or brokering for the tenant does not necessarily imply constraints on the tenant's autonomy (this depends on other promises). This is not exchange of the tenant, like sending a letter, or transporting a passenger. Notice, furthermore, that nothing promised here can prevent the tenant or host from acting as separate entities in other ways.

- **Catalysis** (special semantic environments)

  In a chemical process, some tenants need the help of a tailored environment to make a transition to a new state. A host plays the role of catalyst

  A pit-stop for tyre change, or a port/dock for loading and offloading, or repair of transport vessels.

  In the human realm, start-up labs and incubators are catalysts for companies and biological processes. The womb is a host for infant morphogenesis.

  In each case f(C,R)

Example 192 (Multi-tenancy)**.** *Users are tenants of multi-user software, logging into walled communities with login credentials. Processes are tenants of operating systems. Operating systems are tenants of computer hardware. Computers are tenants of networks and datacentres.*

7.9.4 TENANCY AND CONDITIONAL PROMISES

It should already be apparent from the definition of tenancy, in section 7.9.1, that there is a likeness between the pre-condition for tenancy (denoted C) and the resource relationship (denoted R). From the conditional promise law[BB14a], a conditional binding to provide service S takes the form

$$A_T \xrightarrow{+b} A_1, \quad \left.\begin{array}{c} A_1 \xrightarrow{S|b} A_2 \\ A_1 \xrightarrow{-b} A_2 \end{array}\right\} \simeq A_1 \xrightarrow{S} A_2 \tag{7.68}$$

Notice how the exchange of the condition has the same structure as the tenancy relationship. This is because both are examples of a generic client-server relationship, based on vector promises.

This can be formalized this further to show that a tenancy is really a conditional promise (see figure 7.20).

$$H \xrightarrow{+R} A_? \quad vs \quad A \xrightarrow{-c} D \tag{7.69}$$

$$T \xrightarrow{-R} H \quad vs \quad D \xrightarrow{+c} \tag{7.70}$$

$$H \xrightarrow{+f(C,R)|-R} T \quad vs \quad A \xrightarrow{+b|c} A_? \tag{7.71}$$

Thus the tenant is the assumed recipient of functional promises derived from the tenancy relationship, whereas in a general conditional promise this is unspecified.

Note, we shouldn't worry too much that the sign of the $+R$ maps to a $-c$, as the complementarity rule (see [BB14a], section 6.2.2) allows us to re-interpret the signs. For example, $+R$ could represent the active garbage collection of resources, while $-R$ represents quenching with resources. Similarly, $+R$ could represent employment, while $-R$ is work done to fulfill the employment moniker. In both these cases, the $+$ promise takes on the character of a receipt of service, often associated with $-$ promises.

The tenancy relationship is just an extended version of the basic client-server relationship, with the special focus on identity[122].

7.9.5 REMOTE TENANCY

If we consider the case in which tenancy is not between agents that are actually adjacent to one another, then the promises are delivered by proxy, in the sense of a delivery chain (see [BB14a], section 11.3, and figure 7.21).

When carried out via proxy, every adjacent node in a connective path through the adjacencies of the carrying spacetime becomes a possible point of failure or loss of integrity, and the cost of promising explicit integrity increases as the square of the number of agents along the path taken via adjacent agents.

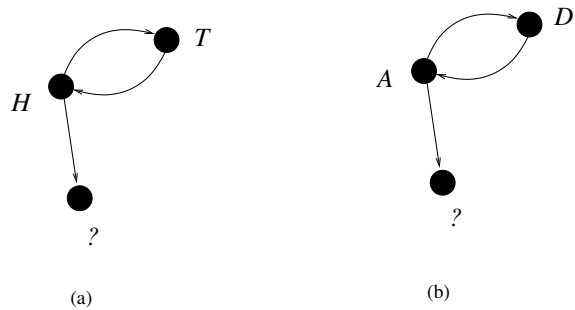


Figure 7.20: The likeness between tenancy and a conditional promise involving a third party.

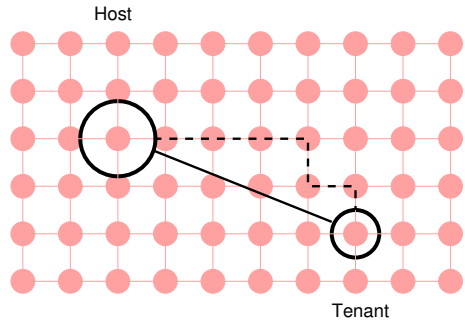


Figure 7.21: Promising tenancy virtually, over a substrate of truly adjacent intermediaries, often requires the distributive rule adds to the complications at the finer-grained scale, and scaling away these details requires implicit trust.

7.9.6 ASYMMETRIC TENANCY

The semantics of tenancy are always asymmetrical, by definition. Adjacency is usually symmetric and mutual, at least when locations are equally weighted. However, if a superior location is next to an inferior location, according to some weighted importance ranking, then the symmetry is broken, e.g. pilot fish surrounding a whale, shops surrounding a mall. Unifying locations like malls, hubs, planets are natural host-roles to shops, spokes, and satellites, regardless of their relative size, because they connect agencies into an accessible nexus. Their 'size' may be thought of in terms of their network centrality[BBCEM10], for example, which gives them semantic importance.

Lemma 40 (Adjacency is a form of tenancy, or tenancy is 'rich adjacency')**.** *By symmetrizing over the host-tenant promises, and directing unspecified promisees to mutual neighbours, we reproduce adjacency.*

$$H \xrightarrow{+R} (A_? = T) \quad , \quad T \xrightarrow{+R} (A_? = H) \tag{7.72}$$

$$T \xrightarrow{-R} H \quad , \quad H \xrightarrow{-R} T \tag{7.73}$$

$$H \xrightarrow{+f(C,R)|-R} T \quad , \quad T \xrightarrow{+f(C,R)|-R} H \tag{7.74}$$

which reduces to

$$H \xrightarrow{\pm R, f(C,R)} T \tag{7.75}$$

$$T \xrightarrow{\pm R, f(C,R)} H \tag{7.76}$$

Thus R plays the role of adjacency, and identifying $R \to$ adj and $f(C,R) \to \emptyset$, we see that adjacency is equivalent to mutual tenancy in its weakest form.

7.9.7 SCALING OF OCCUPANCY AND TENANCY

The ability to use space and time in a functional and operational way is the key to building organisms and organized processes. When we speak of scaling these semantic forms, we implicitly expect to preserve symmetries, asymmetries, and functional relationships, while inflating the overall size of a semantic space by introducing more agents. Coarse graining should then allow us to see the functional equivalence of the larger and the smaller system.

The asymmetry inherent in the ideas of occupancy and tenancy suggests that we are not generally going to see scale-free phenomena. What characterizes tenancy and occupancy is the retention of a differentiated cooperative relationship between agencies. Specific agents are bound together with intentional directionality. This contrasts with the idea of absorbing new agencies into a singular agency.

Example 193. *In a business partnership, or symbiosis, businesses or organisms retain their separate identities and work together for mutually beneficial returns. In a merger or acquisition, one company or organism subsumes the other, hoping to control it without worrying about explicit cooperation.*

The homogeneity of host-tenant semantics often play a role in the coordinated, functional usage of space. Long range order helps us to utilize space in a regular way. Without it, many aspects of space and time are simply opportunistic.

Example 194 (Parking lot)**.** *In a parking lot, the spaces need to be homogeneous in size else you might not be able to park your car in just any space. The same applies to the width or refrigerators, washing machines and kitchen appliances, block and sector sizes on disks.*

We need to account for both strong and weak couplings, homogeneity and inhomogeneity, to understand the wealth of possibilities in the world around us.

7.9.8 EXTENDING TENANCY WITH STRUCTURAL MEMORY

Homogenization is a forgetting process, while inhomogeneous differentiation encodes a memory into spacetime. In a semantic spacetime, memory is encoded through promises, and structure might refer to any one of the aggregation, residency, occupancy and tenancy candidates.

Let's contrast the ideas of scaling by absorption and tenancy more carefully. Consider the two scenarios in figure 7.22, which contrasts a symmetrical form of cooperation (a) with an asymmetric tenancy configuration (b). The solid circles represent agency scales, forming various levels of superagency.

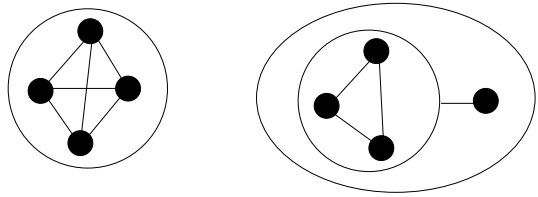


Figure 7.22: The scaling of membership can retain a memory of its process. The introduction of an agency scale can introduce artificial asymmetry. In (b), a promise binding is made to the superagent, but this has yet to be realized by a physical agency within this virtual/logical boundary.

In the first case (a), the cooperative arrangement is completely symmetric, and no information about the order in which the agents came together is retained in the structure. If we assume that the tenancy binding is based on a promise with some body X, then we may characterize this arrangement by:

$$A_i \xrightarrow{\pm X} A_j, \quad \forall i, j = 1, 2, 3, 4. \tag{7.77}$$

$$A_i \xrightarrow{\pm C(X)} A_j \tag{7.78}$$

We see exchange promises for $\pm X$, and symmetrizing coordination promises $C(X)$. In the second case, an intermediate step is apparent which singles out a distinction between the set $A_s = \{A_1, A_2, A_3\}$ and A_4:

$$A_i \xrightarrow{\pm X} A_j, \quad \forall i, j = 1, 2, 3. \tag{7.79}$$

$$A_i \xrightarrow{\pm C(X)} A_j \quad \forall i, j = 1, 2, 3. \tag{7.80}$$

$$A_s \xrightarrow{\pm X} A_4 \tag{7.81}$$

In this case, we see a memory of the structure which sees A_4 joining an already established agency. As yet, the agent A_4 is not completely symmetrical with the other three. The asymmetry of intent remains, although an observer watching how these promises are kept might not be able to tell the difference between scenarios (a) and (b). This depends on how we interpret the promise between A_4 and the superagent A_s. There are two possibilities:

- A_4 binds distributively, assuming that each of the agencies $A_1, A_2, A_3 \in A_s$ makes individual promises so as to behave symmetrically (by virtue of (7.80)), allowing:

  $$A_4 \xrightarrow{\pm X} A_j \quad \forall i, j = 1, 2, 3 \tag{7.82}$$

  The addition of these promises completes the symmetry that turns scenario (b) into (a), eliminating the memory of their inequivalence to an observer, and absorbing A_4 effectively into a cooperative entity. Regardless of how a partial observer might draw its superagency boundaries, it is now able to identify a symmetrical *role by cooperation* (see [BB14a]) between all four agents. This exercise gives us a clue about what absorption means, in a formal sense.

- A_4 binds only to a single representative of the collective A_s. This remains asymmetric, and we would consider A_4 to be a non-resident occupant or tenant of A_s. The memory of the inequivalence is coded into the intentional behaviour, by promises.

These figures illustrate how the promise configurations document the history by which an arrangement of agents was constructed, and allow an entity formed by multi-layered cooperation to retain a memory of past states.

What we see, in figure 7.22, is that adding promises can effectively remove asymmetry between host and tenant, meaning that tenancy can be eliminated by 'acquiring' an agency[123]. However, it is important to also consider scaling in which the tenant never becomes a part of the host, i.e. we maintain the strict asymmetry, as this implies no loss of autonomy between the tenants.

7.9.9 SCALING OF THE TENANCY LAW

Now, let's see how the extended scaling of internal structure in either host or tenant affects the promises made in a tenancy relationship. This scaling applies to all promises between coarse grains, not just tenancy promises.

As the number of subagents in internal structure grows, it becomes natural to consider them both as embedded subspaces of the surrounding semantic space. Such a subspace may be either a solid lattice, with long range order, a gaseous state, or forest-like (molecular).

Consider the full tenancy relationship:

$$A_{\rm super} \xrightarrow{+X\#n \mid C} A_{\rm tenant} \tag{7.83}$$

$$A_{\rm super} \xrightarrow{\pm C} A_{\rm tenant} \tag{7.84}$$

$$A_{\rm super} \xrightarrow{+f(C,X) \mid -X} A_{\rm tenant}, A_? \tag{7.85}$$

$$A_{\rm tenant} \xrightarrow{-X\#m} A_{\rm super} \tag{7.86}$$

Suppose we try to add agents in the manner of a scale perturbation. We preserve the implied roles (the tenant $A_{\rm tenant}$, and host $A_{\rm super}$), and the structure of the binding between them. What features would change, if we now attempted to scale this relationship by adding new internal structure to either host or tenant? With the further addition of $A_{\rm pert}$, assuming this adds to the valency, this becomes:

$$A_{\rm super} + A_{\rm pert} \xrightarrow{+X\#(n+1) \mid C} A_{\rm tenant} \tag{7.87}$$

$$A_{\rm super} + A_{\rm pert} \xrightarrow{\pm C} A_{\rm tenant} \tag{7.88}$$

$$A_{\rm pert} \xrightarrow{\pm C} A_{\rm super} \tag{7.89}$$

$$A_{\rm super} + A_{\rm pert} \xrightarrow{+f(C,X) \mid -X} A_{\rm tenant}, A_? \tag{7.90}$$

$$A_{\rm tenant} \xrightarrow{-X\#m} A_{\rm super} + A_{\rm pert} \tag{7.91}$$

Hence, the symmetry and internal structure of the superagent has a material effect on the binding properties in a tenancy arrangement. This is a formal scaling, but it does not really explain what happens to communicate intent across a coarse grain, as discussed in earlier.

The exterior promises of a host need to scale to provide valency n binding sites. The first issue to satisfy is the valency binding constraints, we seem to need $m \leq n$ in the tenancy promises. There are several ways this might be solved:

- One subagent is allocated to one tenant.
- Subagents host multiple tenants each.

- Several subagents working together service a single tenant.

Similarly, we have to answer the question: how does a tenant know with whom it should interact? Will tenant and host subagent be able to locate one another (see figure 7.13)? Referring to figure 7.13, and the previous discussion of promising at superagent boundaries in section 7.8.2, we ask: should external tenants bind (a) directly to independent subagents, or should they (b) go through brokers and interfaces?

Example 195. *A tenant in figure 7.23 might represent a person or a population looking for an unoccupied apartment through a fronting organization; or a car looking for a parking space in a collection of parking lots, a cubicle in an office space, and so on. While the superagencies like apartment blocks and parking lots formally promise space (valency), the potential tenant needs to locate the empty slots in order to bind to them.*

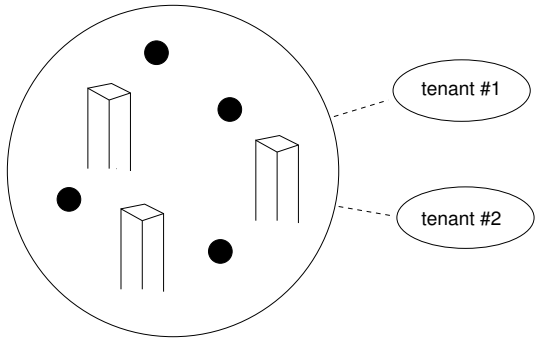


Figure 7.23: Tenants, èn masse, can contact a front organization like a website to search for housing, but still need to connect with the agency's internal components (individual landlords) that can provide actual rather than logical service. e.g. a person needs to make direct contact with the apartment block that offers apartments to rent, not merely the superagency of 'real estate' which has no physical reality. This if one tenant is asking for more service than a single host can offer, an issue becomes whether the service can be delivered in practice, in spite of the apparent size of the superagency.

7.9.10 DISTRIBUTIVE SCALING OF TENANCY RELATIONSHIPS

In order for an outside agent to form an adjacency or a tenancy binding to subagent, its has to know of the other's existence[124]. In all cases, tenant and host need to be able to locate one another, or be introduced, in order to communicate, both form a promise binding and to keep the promise. There are only two possibilities here:

- Host and tenant are directly adjacent (see figure 7.13a).
- Host and tenant can communicate with the aid of intermediaries (see figure 7.13b). This requires cooperation between the subagents.

The superagent coarse graining directory plays the key role here in making these details transparent.

Example 196 (Virtual network)**.** *Viewing virtual or switched private networks as tenants of a series of hosts that cooperate as superagency: this requires tenancy at each independent host, and coordination between them. In addition, the agencies are connected by adjacency promises between the chained carrier hosts (see figure 7.24), or virtual adjacencies via intermediaries or proxies (see [BB14a], section 11.3). The scaled tenancy relationship allows tenant spaces to appear contiguous, even though they might be distributed. Structures like overlays and tunnels act as virtual adjacencies, which rely on a substrate which we coarse grain away.*

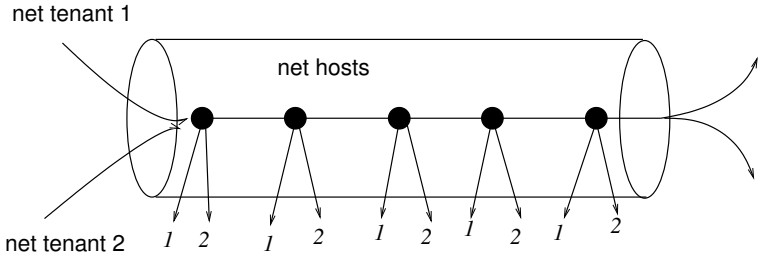


Figure 7.24: Shared network requires the distribution of the entire tenancy binding. The two hosts cooperate on handing over their resources to make a collaborative channel, while maintaining the integrity of the segmentation of tenants.

To scale tenancy on the host side, there are some basic requirements:

Lemma 41 (Distributed tenancy law)**.** *In order to distribute a tenancy to a collection of autonomous hosts subagents, there must exist:*

- *A cooperative agreement between all subagents acting as hosts to share its local resources according to the collective multi-tenancy agreement.*
- *An index for adjacency coordinates of the host subagent, if not directly adjacent.*

Example 197. *A customer of a bank can visit any branch to access their account, by mutual cooperation between the branches, else the customer has to deal directly with*

their home branch. (Remarkably, this is still an issue even in 2015, in spite of the prowess of information technology.)

Example 198. *Binding sites on cells, immune cell responses, MHC etc, allows a chance encounter for antigen to locate a cellular tenancy.*

7.9.11 THE SIGNIFICANCE OF FUNCTIONAL ASYMMETRY

The economics of tenancy (regardless of the currency used, e.g. energy, money, prestige, etc) brings tenants and hosts together, leading to an asymmetric relationship (see figure 7.25). Functionality is associated with broken symmetry: the default state of maximal symmetry has very little going for it to attach semantics to. As seen in earlier sections, pure scaling of agency, in the absence of constraints, does nothing to break symmetries, and hence leads to a maximal radial (spherical) symmetry, like a cell; however, boundary conditions from the environment break symmetry allowing functional behaviour. There are many examples of this in the world of biological organisms.

Example 199 (Reproduction)**.** *Reproductive organisms begin as a single egg, which grows and differentiates. Initially it grows symmetrically, dividing into a ball of cells, then boundary conditions from the environment during morphogenesis create preferred directions with chemical signals. This leads to dorsal-ventral asymmetry, etc. Selective apoptosis leads to further local asymmetries, with functional consequences, such a fingers. Cephalization (formation of a head) is associated with the appearance of brains in the nervous systems of organisms, and the brain is at the front where the organism meets and senses its environment.*

Example 200. *The shift from a spherical symmetry to axial and bilateral symmetries has happened prolifically in biological evolution. Axial symmetry is typically associated with orientation of an organism with respect to a flow, e.g. a jelly fish. In computational terms organisms orient along their input-output axes.*

Example 201. *A bottle (box, container) is usually a round axially symmetric structure, but this is irrelevant to its function. It does not matter whether the axial structure is symmetric or asymmetric, round, square, oblong, ellipsoid, etc. The functional shape of a bottle only depends on one end being open to be able to receive the substance it will contain. This asymmetry makes it bind with specificity with functional consequences.*

Figure 7.25 shows organisms that can be modelled as tenancy relationships with various levels of symmetry and asymmetry. The more decentralized organisms are, the more symmetrical they tend to be, and the less ‘smart’, i.e.’ with functionality that is more uncoordinated. In biology, brains are associated with ‘cephalization’, or the

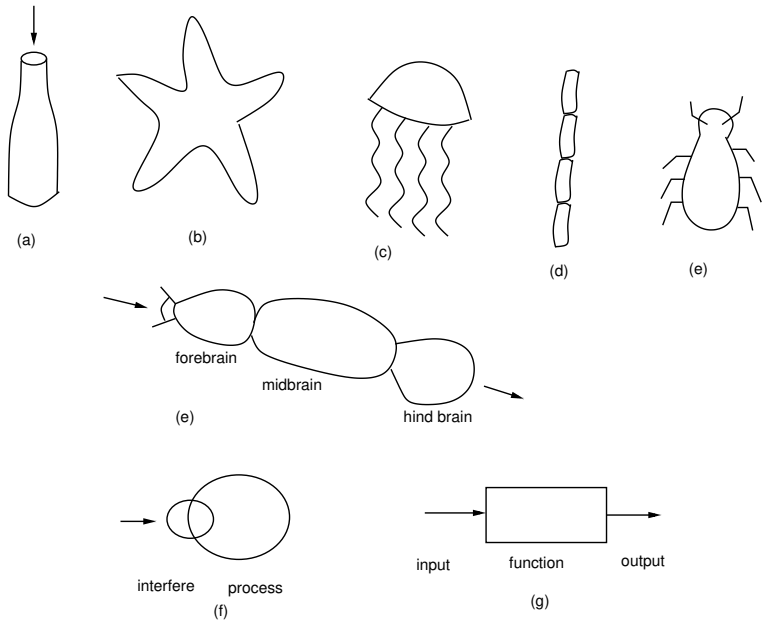


Figure 7.25: Functional behaviour is associated more with symmetry breaking than with residual symmetry. Maximal symmetry or disorder may be regarded as the default state for structures, with minimal cooperation, e.g. starfish. Cooperation with exterior forces order, and symmetry breaking, e.g. cephalization and axial symmetry.

evolution of asymmetry to deal with an axis of input-output. Segmentation along that axis (e.g. figure 7.25d,e,f), or the formation of cells or functional compartments can form through non-linear transmission of boundary information in the manner of Bénard cells, from the presence of a preferred length scale, and aids in semantic differentiation. The segments then often work together in the manner of a host-tenant relationship, binding through specific functional promises. Thus scaling of tenancy (a strong coupling regime) looks very different to the scaling of simple aggregate residency, such as how herds, swarms, and societies scale (weak-coupling regime).

Example 202 (Cells)**.** *Mitochondria and cells co-exist, with mitochondria residents of their superagent cells. Herds and flocks scale in the manner of residency, but remain loosely coupled without direction intentional 'encoded' functional cooperation.*

The scaling of intentionality tends to promote asymmetry, through specialization. This places much greater emphasis on strongly coupled, and hence fragile, semantics. It

is no longer sufficient to have safety in numbers, as in a herd or tissue design. Functional uniqueness, like the components in a radio, bring fragility[125].

7.9.12 TENANCY AS A STATE OF ORDER

The structural memory of tenancy leads to asymmetry, but the long range effects of that asymmetry depend on the phase in which spacetime elements find themselves. The phase becomes a part of the strong or weak coupling constraints.

Consider first tenancy in gaseous state. If agents that are seeking to bind in some kind of tenancy relationship are in a gaseous state, it is easier for them to rearrange and attach directly to a host binding site. Conversely, this makes their meeting more haphazard, since no agent is in a known location. Agents are free to move, so it is harder to trust their identity. On the other hand, they can move around and form adjacencies directly without the need to hand off to intermediaries.

Any spacetime location that is not fixed is in a gaseous state, as a free agent. In technology, this applies to humans, mobile phones, vehicles, satellites, etc There should be sufficient similarity between agents that they can cooperate? Else they will remain inert noble gases.

Example 203 (Internet phase)**.** *The Internet, for example, appears superficially solid, with all of its cables and boxes, but its lack of a robust adjacency relationships makes it just a slow liquid (like the amorphous solids or pitch or glass). Moreover, the lack of a fixed spatial coordinate system means that it has a cellular structure on a large scale: it has small regions of brittle coordinatization, loosely floating in a structureless soup. Names and addresses have no global significance within the scope of a symmetrical pattern, and hence have no long range predictive value.*

In contrast to simple absorption, any cooperative tenancy configuration must bring additional stability to the combined system of agents in order to persist: a symbiosis which confers a positive advantage to being a solo agent. This suggests an instability leading to the condensation of cooperation, first into amorphous liquids and swarms, and then into more rigid crystalline structures. The binding of a tenant and a host into an $H-T$ molecule is locally solid, but might be globally a gas. One could imagine a phase transition in which a bi-partite crystal was formed when scaled, with the structure of an alloy, formed from donor and a recipient.

As a general principle, flexibility and agility require fluidity; but, in order to stabilize semantics, time or change need to be eliminated. The more time plays a role, the less significant the information is. This points us towards solid structures.

Next, we consider tenancy in a solid state The solid state is familiar to us through regular spacetime structures including hotels, parking lots, warehouses, hard-disks, and

computer memory, to cite a few examples. In a solid spacetime, agents are not free to alter their adjacencies over 'time' (see [Bur14], section 5.12), so we trust their identities and relative promises more easily.

At the lowest atomic levels of agency, tenancy in a solid state must lead to asymmetry of space itself, as we'll see in addressable structures. When agents are locked into a solid structure, tenancy often leads to a large scale asymmetry, such as that seen in biological organisms. This seems to have implications for being able to pinpoint functional locations. Hence the loss of maximal, random spatial symmetry (or the rise of so-called long range order) is associated with functional behaviours, and hence functional promises. There are numerous ways in which functional asymmetry can manifest and materialize:

- Tenants and hosts are sufficient in number to be able to symmetrize on a larger scale, say in n-dimensions, and give the appearance of minimal loss of symmetry, e.g. a metallic cubic crystal, such as steel, with tenant impurities.

- Back to back, tessellating structures mixing various orientations are a possible solution to long range order however.

- Hierarchical structures which fill partial spaces, with self-similarity (fractals).

At coarser scales, where 'virtual' agencies can form tenancy bindings on top of an underlying solid adjacency substrate. The fixed locations cannot easily change and be replaced, so they are easy for other agents to trust local neighbours, however being locked into a fixed number of neighbours now mean that long range promise relationships have to be prosecuted through intermediaries (the so-called end-to-end problem discussed in [BB14a]). The presence of intermediaries means that information integrity is now in peril, and trust in non-local relationships is not automatic.

The tighter coupling of agents and reliance on intermediaries to transmit information leads to the possibility of topological defects in the structure, such as crack propagation, and sudden catastrophes. The homogeneity of a solid crystal is important, as mentioned earlier. Re-usability of space suggests quantization of resources in a 'first normal form'. The relational data normalization rule of first normal form is about the re-usability of space[Dat99, Bur04d].

Example 204 (Parking space entropy)**.** *In a parking lot, the spaces need to be the same size else you might not be able to park you car in just any space. The same applies to the width or refrigerators, washing machines and kitchen appliances, block and sector sizes on disks.*

7.10 MULTI-TENANCY, AND CO-EXISTENT 'WORLDS'

Multi-tenancy in information systems is one of the key challenges of operational effectiveness. As the custodian of the shared resource, the host has to be able to keep promises on an individual basis, while still being constrained by the behaviours of all its tenants. This means that there will be contention at the host. Moreover, the host is often the seat of mediation between the tenants and the outside world, acting as a gate-keeper, and sometimes as a barrier or firewall between them.

How the host isolates these resources from one another is one of the key issues in a functional space. One tenant should not be able to bring down another through its misbehaviour, either directly or through the host as proxy. This is the thinking behind insulated private rooms, isolated electrical circuits, multi-user systems and even virtual machines in computing. In these scenarios, the special semantic role of the host makes it vulnerable: a 'single point of failure'.

7.10.1 DEFINING MULTI-TENANCY

What are the promises that make a spacetime multi-tenant? See the example figure 7.24.

Definition 170 (Multi-tenancy at scale M)**.** *Consider a agency scale $M = \{H, T_1, T_2, \ldots, T_m\}$ for some $m > 1$. An agent H is said to exhibit multi-tenancy if a number of autonomous agencies $\{T_1, T_2, \ldots T_m\}$, called the tenants (making no a priori promises to one another) independently form a tenancy binding to H, in the role of host, for a share of a single resource R that H promises.*

$$
\begin{array}{lll}
H & \xrightarrow{+R\#n|C} & A_? \\
H & \xrightarrow{-C} & T_i, \\
T_i & \xrightarrow{+C} & H \\
T_i & \xrightarrow{-R\#1} & H \\
H & \xrightarrow{+f(C,R)|-R} & T_i \\
& m \leq n & \\
& \forall i = 1 \ldots m &
\end{array}
\tag{7.92}
$$

The issues for multi-tenancy include all those for general tenancy, and also segregation, mutual isolation, addressability of tenants, scaling of naming, sharing of resources between tenants, and the possibility of tenants sharing amongst themselves, without involving the host.

7.10.2 BRANCHING PROCESSES: SUBROUTINES, WORLDS, AND HIERARCHY

A brief digression on understanding the dynamics of aggregation of tenants around a host, introduces the notion of a *branching process* (and its inverse, the merge, confluence or aggregation process). Branching is what happens as the possible states or locations of a system fan out and increase in number from n to n', where $n' > n$, selecting a preferred direction, either over space or in time.

A branching process is an evolutionary sequence of changes in which the level segregation or multiple agency increases either in space (over distance) or time (spacelike hypersurfaces) (see figure 7.26). The branch points are associated with instabilities of the system both dynamically (bifurcations) and semantically (if-then-else reasoning).

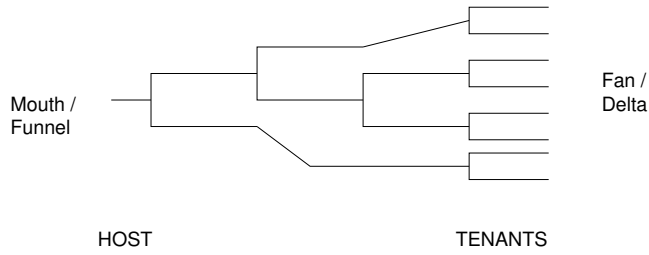


Figure 7.26: A schematic branching process, showing asymmetry of process.

Branching has an arrow of directionality from the root, along each local vector to nearest neighbour, which marks a gradient from fewer to more states. Thus as the process unravels (in time or space), the number of outcomes increases, favouring an increase in system entropy, and decreasing the likelihood of arriving at any chosen one outcome. The structure is not necessarily linear in total: it could be radial (like a Cayley tree, or snowflake), but there is a coordinate direction in which the number of locations is increasing, quasi-exponentially.

- *Branching* is associated with fine-graining, sharing from one to many, reasoning about different possibilities, and it is also about increasing specialization. Branching is a $+$ promise of the host.

- *Merging*, the inverse of branching, is associated with generalization, coarse graining, averaging over possibilities, and shared or common resources. Merging of aggregating is a $-$ promise of the tenant.

As a companion to branching, we also have a notion of *hierarchy*, which is sometimes confused with it. Similar to branching, hierarchy comes about by ranking of states. Thus, if one assigns greater importance to having more or less states, one could call a branching process a hierarchy, as the branching asymmetry implies an order from head to tail.

Definition 171 (Promise hierarchy)**.** *An iterated process, over a series of promises π_i, in which the promises π_i exhibits asymmetric semantics, with respect to the promiser and promisee. Hence the process generates a chain or sequence of ranked or ordered elements (agents), in the manner of a semi-lattice.*

Example 205 (Promise dependency and broken symmetry)**.** *For example, a promise to use (dependency, requirement, etc) is asymmetric. Promises of mutual adjacency, on the other hand, have symmetry, hence there is no preferred direction for semantics. A branching process over mutual (symmetric) adjacency does not form a hierarchy, only a tree or forest structure.*

Definition 172 (Branching hierarchy)**.** *A branching process in which there is directionality of both dynamical branching and semantic intent.*

Example 206 (Taxonomy as a symmetry breaking hierarchy)**.** *Dependency trees, and taxonomies form branching classification hierarchies.*

Because the branching of a single host into multiple tenants has the same asymmetry, clearly branching is a key process in interpreting multi-tenancy. It is the inverse of the accretion of multiple tenants to the single host. The converse of the tenancy law is that tenancy is a branching process of *host resource usage*, branching from one host into multiple tenants, and thenceforth. Semantically, branching may come about in a number of ways:

- Through a breakdown of cooperation between existing agents, leading to a *fine-graining*[126] or fragmentation of roles.
- As a spawning of new agents, increasing in number.

Dynamically, the branching can be either

- Due to a change in the receiver, e.g. a (-) use-promise no longer aggregates agents.
- Due to a change at the source, i.e. a (+) promise now differentiates agents.

Branching processes lead to proliferation of 'parallel worlds' (disconnected subspaces), and possibly a change of dynamical scaling. Keeping track of these worlds becomes a divergent problem of *knowledge management*, without a counter-process. If there is

exponential growth of agency, this may be *intractable*, in the sense of computational complexity.

Branching processes may be examined through the lenses of semantics and dynamics:

- *Semantic*: Disambiguation, tenancy, security, etc.
- *Dynamic*: Isolation, bifurcation, cell division, renormalization, etc

Figure 7.27 contrasts a few of the scale issues mentioned thus far. Notice how, at what might be perceived in one way, at one scale, could be perceived another way from the perspective of a different scale.

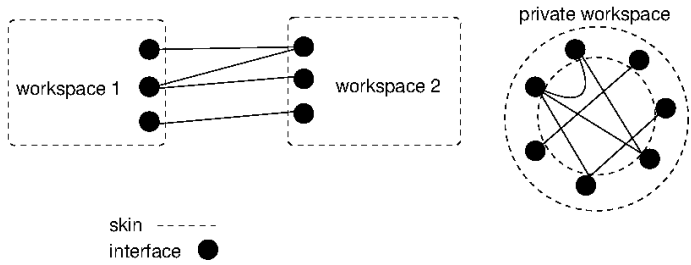


Figure 7.27: Dynamical and semantic aspects of multi-tenancy eventually merge through the semantics of scaling. This illustrates the importance of scale in the total semantic content of an observer's realm of assessment.

7.10.3 TENANT SEGREGATION, AND RESOURCE MULTIPLEXING

A host shares its resources into slots, which can be occupied by a number of tenants in a number of ways:

- Segmentation or partitioning of adjacencies in an underlying graph, or spacetime.
- Labelling of regions by promises, marking gradients (stigmergic typing of agents).
- Growing and spawning new agents.

Example 207. *Division multiplexing is a common strategy in hosting:*

- *A parking lot (host) divides up its space into slots that can be used by cars (tenants).*
- *A hard disk (host) divides its surface platter into sectors and blocks, for user-data (tenants).*

- *A time-sharing computer operating kernel (host) divides up its computational resources into processes that can be used by jobs (tenants).*
- *Memory address spaces (host) divide up addresses into pages, for occupation by process data (tenants).*

Isolation of a subspace at the host can come about by two means:

- The absence of adjacency between the tenants.
- The absence of reachability: i.e. an agent cannot be reached because it has:
  - No adjacency path/route, with or without the help of intermediaries (vector promises).
  - No name or (unique) identity, or address, to locate it by (scalar promises).

Thus, both scalar promises (e.g. names) and vector promises (e.g. adjacency) play roles in the segmentation of shared spaces.

7.10.4 MOUTH FORMATION AND GATEWAYS

The picture of spacetime described in most computer science (see, for instance, Milner's bigraphs [Mil09, Bur14]) represents a containment view of the world; such a representation is not an efficient way of addressing objects for ease of locating them. On the other hand, it is a useful way to describe the semantics of interfaces.

One of the values of the concept of multi-tenancy is the ability to have the host act as a broker for mediating contact with the tenants. We see this in many everyday scenarios:

Example 208. *Interface scenarios:*

- *A security checkpoint at the mouth (or head) of a host building, or secure area.*
- *Passport control at the airport interface of a host country.*
- *Gated access via host to locked tenant storage units.*
- *The eyes and mouth of a host organism mediate contact to the tenant organs: brain, stomach, etc.*

A tenancy leads to a form of superagency seeded on (and mediated by) a host. The host acts as a kind of moderator or proxy for certain communications with the outside world, though it might not be the only source of adjacency, if the scale of the tenancy relation is based on promises that are made over a fabric of lower-level adjacency. The axial symmetry induced by a gateway can be compared to other functional spaces (as discussed in section 7.9.11).

7.10.5 TENANCY FORMATION AND PRIVACY AS AN ADDITIONAL PROMISE

There are two questions concerning tenant segmentation and containment in a hosted superagent:

1. *Who gets to decide whether a tenant or member can join a hosting collective?* This might simply be part of the evolution of the design, or it might be a decision made by the host, or indeed all of the tenants in concert.

2. *How are tenants kept disjoint from one another, and how is access to the tenants moderated?* The connectivity involved in a tenancy with a host favours a radial symmetry between host and tenants. This can be folded (see figure 7.28) leading to the functional asymmetry.

The default assumption is that there is no cooperation between tenants:

Assumption 9 (Default null adjacency promise for tenant segregation)**.** *The default adjacency state, at scale M, between tenants is no adjacency. This is most easily accomplished in a spacetime gas phase. If tenancy is built on a connected lattice in the first instance, then this isolation might require additional promises to block adjacency. However, the latter is a losing strategy: the amount of information needed to 'lock down' every agency is too large. You need to compress the pattern into a list of exceptions.*

Independent semantics of tenants might well go beyond this simple dynamical observation however. Segregation of assets might be viewed as being an important requirement of tenancy. Naturally, this promise can only be kept by a single agent, hence only host-mediated resources can be segregated as a promise to tenants.

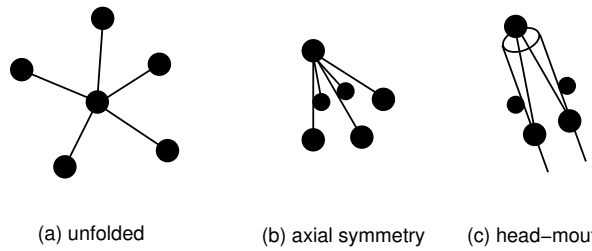


Figure 7.28: How the preferred host role leads to an axial symmetry. From spherical symmetry (a), a external flow selected a direction (b), which eventually organizes along an axis. Segmentation along the axis then marks out different levels of hierarchy in multi-stage tenancy.

If resources are mediated by the host, as is natural for this reason, it can also act as a moderator, gatekeeper, or security monitor. The point of a gatekeeper is to limit adjacency to a narrow bottleneck, or checkpoint (see figure 7.29). While this does forces a fixed scale limitation on the choke point, it simplifies the semantics of authentication. The host has to be able to verify the identity of tenants to keep its promises. Having a gatekeeper interface at the host is a simple way to do this. This helps to turn the branching process of the tenancy into a *hierarchy*, in which the host is the gatekeeper to upper levels, mediating contact to the hosts below.

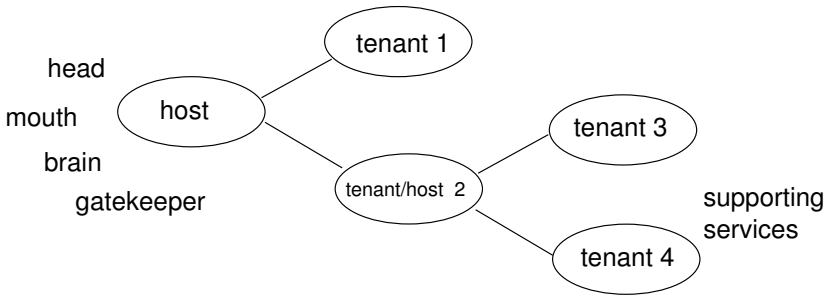


Figure 7.29: Stacking tenancy branchings leads to an oriented access hierarchy. Each branch point acts as a gateway point for accessing the tenants.

Example 209 (Lack of addressability)**.** *Many of the classic security blunders have been due to relying on the lack of addressability, in the belief that an item that cannot be named would not be accessed. This is a form of 'security through obscurity'. Systems that base isolation on prevention are much harder to police than*

As an exercise to the reader, consider what promises lead to 'secure semantics'. How are keys or addresses for accessing tenants assessed by the host, and by users? What agency plays the role of gatekeeper, if not the host? Implement a 'secure' system using conditional promises to segregate tenants.

The functional utility of the asymmetric tenancy structure thus seems to lie the following observation:

Law 2 (Hosting of input and output leads to axial symmetry)**.** *The functional arrangement of input/output mediated by the host leads to a natural head-tail asymmetry, in which the head is favoured in a hierarchy of longitudinal stages. This is known as cephalization.*

7.10.6 SPACETIME SHARING BY TENANTS: SERIAL TIME AND PARALLEL SPACE

When tenants subscribe to a resource in parallel, they share the resources of the host. This is called multiplexing in the resource domain. When a host-tenant network has net valency less than zero, tenants can multiplexed in the time domain. Over-subscription of a resource could lead to the need for time-sharing. Time division multiplexing is the way this is done in a serial queue[127].

Example 210. *Time-division multiplexing (queue-processing) of an oversubscribed service:*

- *Time-sharing of apartments.*
- *Car rental, or recycling of vehicles.*
- *Aircraft/bus passenger seating is only fixed for the duration of a journey.*

Finite-duration promises are the key to all queueing systems.

7.10.7 MULTI-STAGE MULTI-TENANCY, AND FABRICS

This section is a continuation of the iterated solid state tenancy structures, used for locating agent addresses, described in section 7.8.7. I return to the topic to emphasize that iterated multi-tenancy leads to important scaling properties, both in the information technology realm and in the biological realm. Network fabrics include toroidal networks, Clos networks, and Batcher-Banyan networks[Nar88]. This example is about the popular 2x2 Clos networks.

Example 211 (Clos network)**.** *The example of a 2-valent Clos network is instructive because it is a dynamically simple case that is increasingly used in datacentres. Semantically, it is more complicated, because it combines multi-tenancy with multi-homing (multiple hosts) in order to create a cooperative self-organizing structure. To make addressing work as a repeated, tessellating pattern, each agent needs to be both a host and a tenant for different promises.*

Today, Clos networks are intimately connected with a particular implementation involving the Border Gateway Protocol (BGP)[NLK13, Ano14]. BGP is a network route advertisement service, or gateway service, that implements a set of promises for promising route information between superagencies known as Autonomous Systems (ASN)[128].

Figure 7.30 shows the basic tenancy and dual homing. Two adjacencies upwards, from each agent in the fabric (shown in bold), connect each tenant in a lower tier to two

redundant hosts in the tier above (for resilience and load sharing). Multiple adjacencies downwards in a tier connect each agent, now in the role of host, to its subtenants. Thus there are two tenancies back to back promising resources:

The characteristic of the tree structure in a Clos network is that each branch terminates at a definite 'leaf location' with a definite and unique address. This means that every agent in the pattern knows that 'down' means a specific location, and everywhere else is 'up'. Hence, each agent engages in two cooperative relationships, framed as tenancies:

- $R_{\uparrow}$*: tenants forward messages that don't belong below me upwards for the host to aggregate and deal with. This could mean routing to one of the other parallel tenants, or it could mean sending the message out into the wider world beyond the host's boundary.*

- $R_{\downarrow}$*: hosts forward messages that belong belong me downwards to the tenant they belong to. They know which direction to send the message, because the tenancy requires this information to be paid up as part of the condition* C*.*

The tenancy boundaries thus lead to progressive layers of concentric nested agency. The agents (which are all network switches) play the dual roles of host and tenant with respect to these different services, in different layers. The tessellating pattern of all of these woven into a fabric allows it to scale to number of independent addresses that are greater than the fixed valency v *of any one host*[129].

When forwarding upwards and out, what was a host for the downward routing is now a tenant of each of the lower layer agents that provide addressing data. So the semantics of the roles are reversed depending on the process we consider.

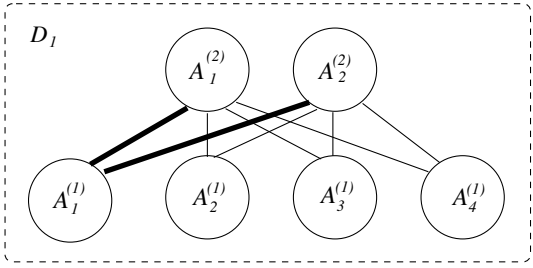


Figure 7.30: Each agent promises to form tenancy agreements with two hosts above it. Each host is a gateway to a subordinate world.

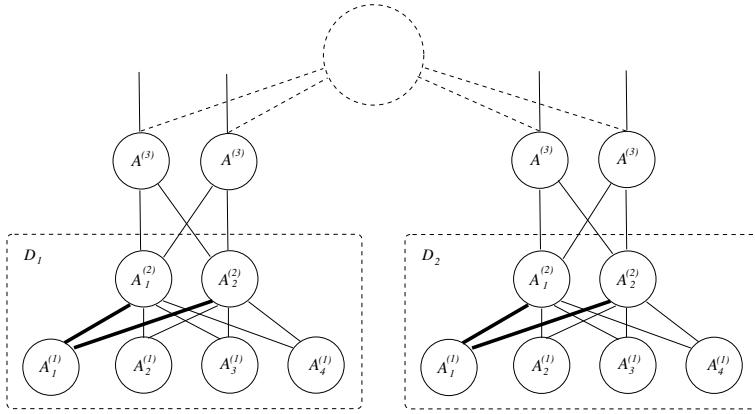


Figure 7.31: The structure can be replicated across datacentre superagents (D_n) Notice that each tenant always has two adjacencies upwards, connecting it to redundant hosts. Eventually, as a message goes up, it will reach the boundary of the Clos superagent and then the rules for regular (solid state) addressing must change.

The promises all rely on the correct positioning of the nodes to work. In practice, the Border Gateway Protocol is typically used to equilibrate that information and make sure the promises can be defined relatively, with self-organizing consistency.

Keeping a simple notation for representing a pattern this complex is a challenge; however we can illustrate the intent, along with the essential concepts of vector promises, and tenancies. I denote the ith agent $A_i^{(n)}$ in tier n, either by $T_j^{(n)}$ or $H^{(n)_i}$, depending on its role, where the indices i simply run from 1 to 2 for the dual homed hosts, and j runs from 1 to the downward valency v. Let's formalize the pattern using the multi-tenancy promises. We can label the tiers by superfix (n), and the tenants/hosts within a tier by subfix i. Without taking into account the vertical edge conditions, we can write the tessellating mutual tenancies by the pattern:

$$T^{(n)} \xrightarrow{+R_{\uparrow}^{(n)}(H_{2i-1}^{(n+1)})\#2|C_{\downarrow}} H_{2i-1}^{(n+1)} \quad \forall\, i = 1, 2 \tag{7.93}$$

$$T^{(n)} \xrightarrow{+R_{\uparrow}^{(n)}(H_{2i}^{(n+1)})\#2|C_{\downarrow}} H_{2i}^{(n+1)} \tag{7.94}$$

$$H^{(n)} \xrightarrow{R_{\downarrow}^{(n)}(T_j^{(n-1)})\#v|C_{\uparrow}} T_j^{(n-1)} \quad \forall\, j = 1 \ldots v \tag{7.95}$$

Downward forwarding assumes that an agent will only accept an address for forwarding if it knows that there is a path to the address below it. That information could be hardcoded into the pattern. In practice what happens is that BGP broadcasts this

information upwards from the bottom to the top, so that each agent tells the two agents above it (its redundant upward forwarding hosts) which names and addresses it can forward to:

$$T_{(n)} \xrightarrow{f_{\uparrow}^{(n)}(H_{1,2}^{(n+1)})\#2} H_{1,2}^{(n+1)}. \tag{7.96}$$

The summary of the parts may be written in terms of these vector promises:

$$\begin{aligned}
R_{\uparrow}^{(n)}(A_i^{n+1}) &= +\text{forward up to } A_i^{n+1} \text{ if it represents the best path (route)} \\
R_{\downarrow}^{(n)}(A_i^{n-1})\#v &= +\text{forward to best path (route)} \quad \forall\, i = 1 \ldots v \\
C_{\uparrow} &= -\mathrm{R}_{\downarrow}^{(\mathrm{n}-1)} \text{ i.e. accept the upward forwarding from below (ACL)} \\
C_{\downarrow} &= -\mathrm{R}_{\uparrow}^{(\mathrm{n}+1)} \text{ i.e. accept the downward forwarding from above (ACL)} \\
f_{\uparrow}(C_{\downarrow}) &= \text{Inform about known addresses from below (BGP)}
\end{aligned} \tag{7.97}$$

Each v-valent agent is a host to the v agents below it, and a tenant of the agents above it. The upward valency is 2. As long as we are far away from the lower edge of this pattern, each host also knows that it has two possible routes downward to its tenants also, because of the interwoven tenancy agreements. However, at the bottom edge, there is only a single adjacency to the final address.

Throughout this patterns, the (-) use-promises play the role of access control lists (ACL) for accepting data. I have suppressed most of this to avoid overwhelming with detail; however, those details are important the security and autonomy of the fabric. Without individual tenant control over its choices, the fabric becomes a homogeneous and isotropic solid state space, with long range order. Certainly, this is a good illustration of how such order is valuable, both semantically and dynamically, but it doesn't really explain how it comes about in practice between uncooperative agents.

When the Internet Protocol was designed, the routing of messages in this way was not conceived in such dense and regular spaces. The Internet was assumed to be a sparse network with clusters at the edges. Gradually, however, as the density increased, and superagent boundaries were drawn around organizations, a cooperative agreement protocol (BGP) was introduced for mutual benefit. The sole effect of this protocol is to propagate homogeneity along point-to-point adjacencies. Thus, today, a Clos fabric is implemented as a BGP multi-tenant array, just like a tenancy between two utterly independent organizations who want to cooperate in order to forward messages within a shared address space.

Although we think of the Internet as having a single global address space, there is no reason why this should be the case. What makes it true in practice is the need to cooperate between 'peers', i.e. between private superagents who can work symbiotically.

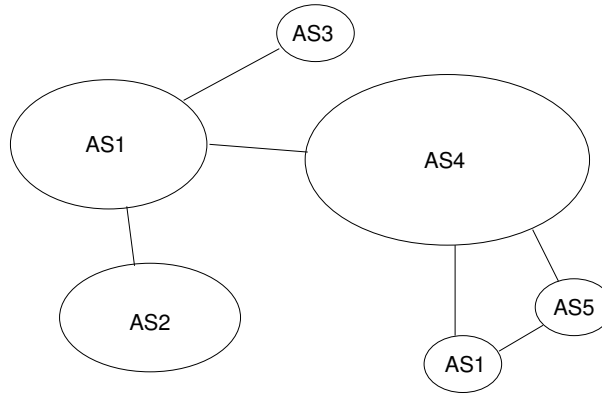


Figure 7.32: The large scale structure of agent spacetime could easily have different coordinate patches, falling into independent namespaces (called BGP autonomous systems (AS)). Each superagent (AS) can choose whether to cooperate with the community of addressing or go it alone. This has become common due to the poor scaling of the IPv4 address space, with Network Address Translation for partial splicing at the edges.

As discussed earlier, it is the breaking of symmetries that leads to functional differentiation in a structure. As a final comment on this example of a Clos regular spacetime, we can note that there is very little differentiation. So let's focus a moment on the natural residual symmetry of the structure. The functional asymmetry of the Clos network is a compelling example of how multi-tenancy orients a structure, but it has a less obvious cephalization. The structure has no obvious brain that forms a master host for the entire structure, but the head is clearly at the top or mouth of the outside world. The scaling is a symmetrical interior scaling of the superagent boundary: more of a starfish than a cephalopod.

Given such a level of long range order, and insignificant anisotropy along the up-down left-right axes, it makes sense to study the residual symmetry. On paper, we draw these networks as trees with promise adjacencies that cross over one another (see figure 7.33), as if they were in a two dimensional Cartesian lattice. However, we should not be fooled by the clumsiness of a paper drawing. The fact that the adjacencies cross one another should be a sign that this is wrong. Tree-structures have a radial symmetry, and hence the host-tenant decomposition lends itself naturally to polar coordinates, centred on the host.

If we allow the structure to untangle itself, by going to three dimensions (see figure 11.9), then its true structure begins to make more sense. First the dual hosting can be

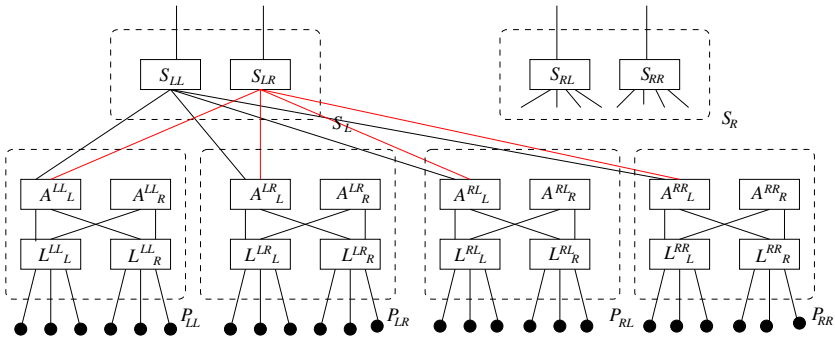


Figure 7.33: A Clos network showing redundant multi-stage multi-tenancy.

symmetrized axially, going up and down instead of just up. The then twisted pairs near the bottom of figure 7.33 can be untwisted to form rings. What one is left with is a hollow

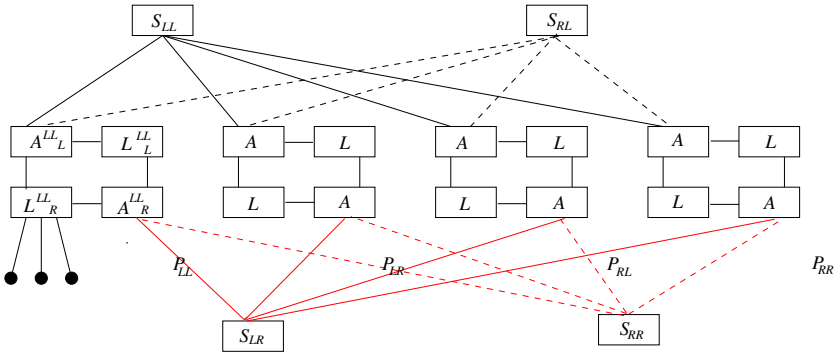


Figure 7.34: A Clos network showing redundant multi-stage multi-tenancy.

tube forming a toroidal geometry, with symmetrical up-down mouths at the top and the bottom. Thus, we have eliminated the quasi-cephalization of the structure and revealed its natural form, which has no asymmetry. It can work top to bottom or bottom to top interchangeably. The final form is shown in figure 11.10. This tendency for us to orient structures into hierarchies is a common habit in human affairs. However, it often leads to scaling issues and bottlenecking of promises.

It is interesting to think about how Software Defined Networking has tried to recentralize network controls with brain-like controllers. The annals of science would suggest

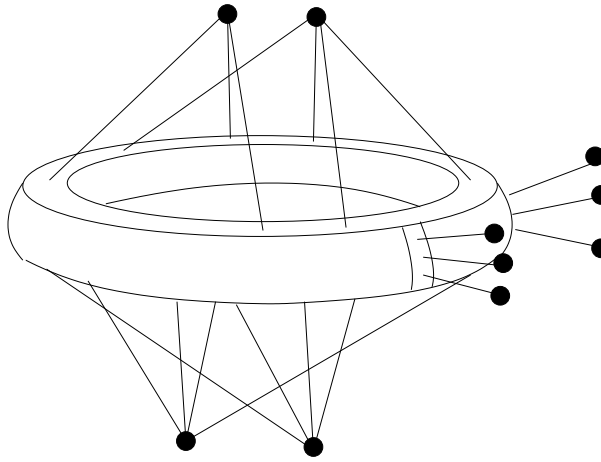


Figure 7.35: Re-folding the radial symmetry into a non-oriented axial structure with no crossing paths. All connections are 'line of sight'.

this would be associated with a natural asymmetry. But where is the asymmetry in a datacentre network fabric? Where should the brain be located in order to fulfill its roles as a rapid correlator of sensor input, and coordinator of reaction?

7.10.8 CROSS-COOPERATION

Tenants are assumed to be initially isolated from one another by default. However, we need to ask at what scale is this isolation true? The assumption might not be compatible with the underlying adjacencies of spacetime, since the tenancy promises could easily be made on top of an adjacency substrate.

Example 212. *Two students sitting an exam occupy desks next to one another, with line of sight. They make no promises to communicate, but they are physically adjacent, not isolated. Similarly, two rival companies share offices and computing resources in a hosting unit. They make no promises to interact, they might even promise not to interact, but they have an underlying adjacency making the assumption ambiguous.*

Segmentation, and non-interference rest on the default assumption about tenants that they make no promises to one another, i.e. that their only notable promises are with the host only. Sometimes separate tenancies may need to be combined. This process can be carried out following the scaling rules in earlier sections.

Example 213. *Several military units under a common command are combined to carry out a mission, e.g. different NATO country members collaborate.*

Example 214. *The section of this document that you are reading now, and the next, are separate tenants of the total host document. However, they would better be written as a single section, and their isolation can be worked around by making promises to cooperation (either through the adjacency of the host, or otherwise).*

If one tenant promises to sublet part of its space to another, by making a promise transverse to the longitudinal axis, this is ok, as it does not affect the sharing promises of the host. If the intermediary funnels resources from one tenant to the other, this could undermine the host's intentions, but it can never cannot cause the host to break its promise, thanks to the locality of promises (see figure 7.36).

However, the host might want to prevent this. It's channel for this is in setting the conditions it is willing to accept from the tenants (conditional $-C$). Thus, at best, the host can try to protect its own interests locally, and ask for the tenants to promise good behaviour. It can punish bad behaviour (tit for tat) but it cannot prevent it.

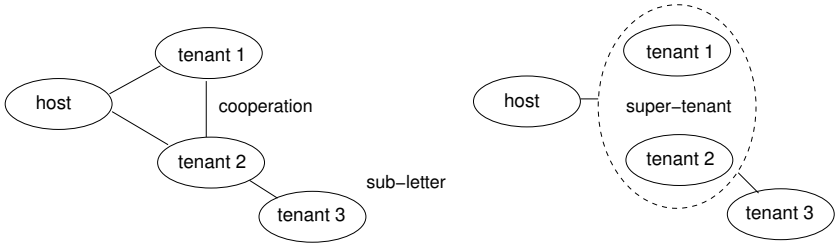


Figure 7.36: Tenants might promise to cooperate and break their isolation. They are free to do this, unless the host agreement's conditions C forbid it, in which case the host might cease to keep its own promises.

Example 215. *Choice of agency semantics are a design decision in a functional world. For commodity items like radios, television sets, watches, and so on, we design agencies to be easy to grasp by asking the question: what semantics do we wish to expose to an observer?. Consider the illustration in figure 7.37, showing two choices of agency boundaries for a common appliance. The listener of the radio only needs to see the outer surface of the agency, not its internal components. The battery is a component that needs to be changed frequently compared to the lifetime of the device, so from the perspective it might seem to be a separable piece. However, few users of a radio would want to have the battery alongside the radio as a separate entity. Thus, an agent in the role of*

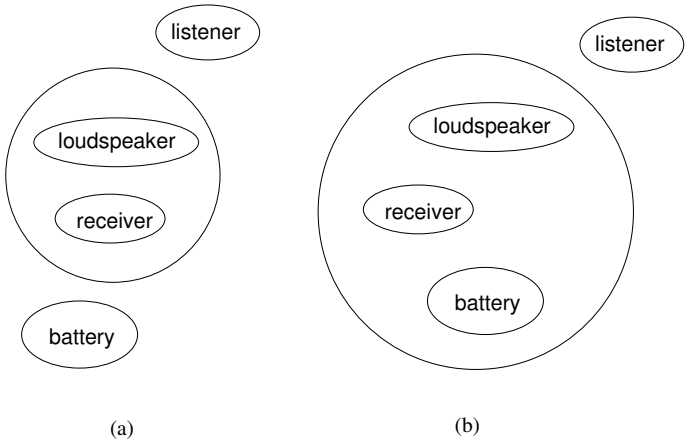


Figure 7.37: Agency scales for a radio: (a) separate battery, (b) battery included.

radio listener (appliance user), a single agency offers the preferred semantics of scale. However, another agent in the role of radio-maintainer could prefer to see the agent at a different scale, where replaceable components are exposed as separable entities.

7.10.9 SCALABILITY OF MERGERS

It is not uncommon for mergers and acquisitions of tenants to take place, in any realm of tenancy, so knowing that this can be done without violating the isolation of any other tenants is important. Following on from the subject of scaling agents in section 7.9.7, we can also consider how segregated tenants and hosts behave when they promise to act as collective entities. Does this change the tenancy relationship? From the viewpoint of the host, the addition of a cooperative agreement changes nothing. Any resources are still provided as $+R_i$ and $+R_j$ to tenants i and j respectively; thus sharing has to be re-routed by the tenants to one another. This places them in the host of mutual host for one-another, which might not be optimal in terms of semantics (it is an unwanted complication of the design). The solution for the long-term would be to renegotiate a new host-tenant relationship for the new composite entity.

The asymmetric resource tenancy shown in figure 7.36 is another example of how a semantic space *remembers* the historical process of its genesis, (recall figure 7.22). The resulting network is a non-optimal promise bindings that are not simple scalings of the intended relationships can come about by 'hot-wiring' the promises to work around the lack of scalability. Such ad hoc cooperation can solve a temporary need, but creates

a new scale pinning, through the asymmetry of the promise, which will further stifle scaling should further superagency

7.10.10 NAMESPACES AND HIERARCHIES AS MULTI-TENANCY IN IDENTITY SPACE

A namespace is a cooperative tenancy in which members are isolated from their surroundings by a gateway host. The host promises to make names of its tenants unique, usually be extending the names hierarchically under the common umbrella of its own name. This is now sometimes called 'disambiguation' in software. It is essentially the same as adding more lines to place addresses so bound location by containment within. It is a different strategy opt uniqueness than tuple-coordinatization.

Example 216. *The same street names appear in many of the local towns, often based on names of historical figures, but this is not a problem, because each exemplar can make a unique address by adding the name of the local town to disambiguate from the others.*

Similarly, the 'High Street' is a common fixture in British towns. Since most towns have a High Street, the name of the town and district can be added for uniqueness.

The term 'namespace' is a popular concept in computing, but the concept of namespaces are clearly in common and widespread use.

Definition 173 (Namespace)**.** *A namespace N is an isolated subspace of a semantic space, formed from a collection of agents. Inside a namespace, all agent coordinates and names are unique by mutual cooperation.*

$$A_i \xrightarrow{+name_i | name_i \neq name_j} A_j \quad \forall i,j \in N \tag{7.98}$$

$$A_i \xrightarrow{\pm C(name)} A_j \tag{7.99}$$

Outside the namespace, names can be made unique by transforming the names according to some bijective function, e.g. either extending the name with a boundary prefix identifier, or performing a name translation.

$$A_i \xrightarrow{+f(name_i,N)} A_k \quad \forall i \in N, k \notin N \tag{7.100}$$

$$A_k \xrightarrow{-f(name_i,N)} A_i \tag{7.101}$$

The hierarchy implicit in nesting namespaces allows us to view these as tenancy relationships, where a host is appointed as the gatekeeper mediating adjacency between the namespace and the outside:

Example 217. *Namespace examples:*

- *A family surname labels a namespace in which Christian names can be made unique relative to other families. Today, this is not a very successful scheme, as it has not scaled well to modern populations.*
- *Filesystem trees, like the Unix or Windows hierarchical directories name subdirectories and files tenants of their parents, in a forest graph structure.*
- *Recursion with local variables in functions and subroutines uses the functional closure as the host agent.*
- *Subnets and networks form a two or more level hierarchy of attachment. A layer 2 broadcast domain is a namespace, in which an IP router is the hosting gateway prefixing addresses with their subnet prefix.*
- *Taxonomy and classification hierarchies use subject categories as hosting agencies which contain subcategories.*
- *Programming class hierarchies, class member functions etc are hosted within named objects, much like a file-tree.*

7.10.11 TOPOLOGY AND THE INDEXING OF COORDINATE-SPACES IN SOLID-STATE

As we saw in section 7.8.7, addressing and tenancy are related through the notion of routing between autonomous agents. The basic asymmetry of tenancy is what allows message sorting by address to implemented by cooperation, and the ability to scale this is therefore connected to how we scale tenancy.

Branching hierarchies are the most common form of classification sorting ('disambiguation') in information technology. They follow in the Aristotelean tradition of taxonomy, widely embraced during the 19th century. It is not uncommon to shoehorn models into a tree structure out of habit. This should not be necessary, however. The challenge of any coordinate system, for address encoding, is to mimic the structure of spacetime in the naming conventions of the points. This enables predictability and emergent routing from local autonomy. Cartesian tuple-based coordinates are flexible, and are not tied to any particular origin.

Since subagent names are interior (local) to a given superagency, they can be organized as that agency sees fit. If the size of the namespace is not too large, almost any naming scheme is workable. However, as the number of internal subagents grows, the efficacy of tuples depends on the internal connectivity. This need not be a hindrance to using tuples for the following reason: tuples can always be fitted to a spacetime as a covering, with a possibly complex boundary. Put simply, even if the tuple space is not

fully populated with points in a Cartesian lattice, one can use the points that are available and ignore the others (see figure 7.38).

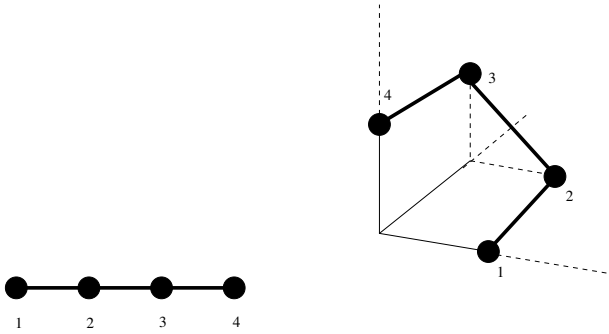


Figure 7.38: Even though the actual occupancy of the Cartesian lattice is sparse, we can use tuple coordinates with a convenient dimensionality. In this example, the only occupied points are (1,0,0), (0,1,0), (1,1,0) and (1,1,1). Using any other values is simply defined to be disallowed.

A namespace can be covered by an arbitrary set of coordinate tuples of the right size so as to span all agents in a spacetime uniquely, giving each agent its own unique ID. For, example, a tree reference

/tenant/container/subcomponent

is trivially written as a tuple:

(tenant, container, subcomponent)

The principal different between these two forms is that the order of the tuple components does not imply any particular ordering of dominance or containment. An agent can approach the identification of points by iterating over the tuple members in any order to parse the space according to the desired semantics.

7.11 APPLICATIONS OF MULTI-TENANCY

With a number of tools, principles, and concepts for multi-tenancy under our belts, we are now approaching a vision for how one could design and operationalize environments, both as isolated 'organisms', and as fully connected ecosystems of autonomous agencies.

Let's consider some examples to illustrate these points, focusing particularly on the world of information infrastructure, or what is now called 'cloud computing'. These

cases have immediate utility. We can summarize the basic principles covered so far as a number of points to cover for each analysis:

1. Determine language of promise bodies.
2. Determine the conceptual head of the organism, from input/output flow.
3. Describe segmentation or tenancy/sharing relationships.
4. Explain the role of spacetime phase (solid, gas, hybrid, etc).

In these examples, I will sketch out some outlines only, leaving the details as an exercise to the reader. Completing a full analysis of a comprehensive system would be a significant undertaking.

7.11.1 EXAMPLE: PROCESSING ELEMENT IN IT INFRASTRUCTURE

Example 218 (Cloud infrastructure)**.** *Modern IT infrastructure (what is now termed cloud computing) is built from arrays of processing nodes, storage devices, and network switching devices. There is a plethora of terminology which I will try to avoid. In this example, I want to focus only on the processing nodes. A processing node (sometimes called a 'compute node' or 'machine') generally consists of a physical computer (also called a 'server' for historical reasons). It acts as a host which promises an operating system (*$+f(C,R)$*) providing tenancy to* processes*. These sometimes consist of virtual machines or 'containers' (*$+R$*).*

The purpose of hosting these machines and containers is to support the running of software applications within the tenants. For now, I'll disregard the details of the applications and simply assume that there are isolated processes (see figure 7.39).

An array of computing machines can be thought of as a semantic spacetime, with each host representing a location. Taking the view that the hosting service is a 'front' for the internal tenants, each host may be modelled as a superagency, with internal structure consisting of the tenants. Each host has a name and an address, and the processes inside them have names and addresses too.

Let's consider the four points:

1. The language of promise bodies.

We always start by expressing promises that we know we can keep. Promising something vague or undefinable is an act of self-sabotage. Simple promises, close to the capabilities of the agents are preferred. Here, the alphabet of promises may include promises to execute programs at a certain rate (based on the CPU clock speed), transmit data, isolate processes from one another, etc.

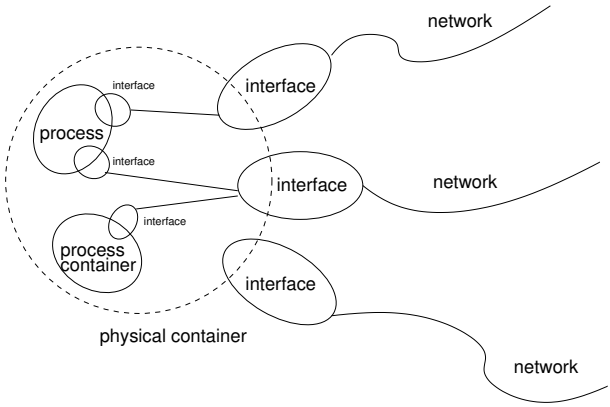


Figure 7.39: A tenant-oriented infrastructure. Notice the limited self-similarity between the process container scale and the physical container scale.

2. Determine the conceptual head of the organism, from input/output flow.

   The head of a processing node is where the interfaces to the outside world enter (see figure 7.39). These are connected directly to the kernel, and the kernel (represented by the dotted line) acts as a kind of brain for the hosting of the tenants. The tenants are the processes, which have virtual interfaces connecting them to the actual network interfaces, via the kernel.

3. Describe segmentation or tenancy/sharing relationships.

   Segmentation of the host's body is in terms of the process containers. The isolation is maintained by two-mode operation at the hardware level, and thereafter mediated by the kernel.

4. Explain the role of spacetime phase (solid, gas, hybrid, etc).

   The phase of the spacetime is undefined here. Inside a superagent, processes come and go, addresses typically change, looking like a gas. Outside the host, the Internet at large has the structure of a gas, for much the same reason. However, inside a datacentre, fabric design with regular symmetrical arrays (e.g. in leaf-spine Clos networks) and fixed networking has the structure of a solid. This looks not unlike a biological cell, with structure floating around inside and outside a boundary.

How does this scale up? Scaling can be done in one of two ways: either on the interior *of the superagent boundary of the host, or on the* exterior*:*

1. *Interior agents can be scaled by parallelizing each subagency of the superagent into a larger subagent. The number of subagents grows linearly and the superagents remains constant. This is how we build a mainframe, or NUMA-scaled cluster. Each component is made stronger by adding parallel numbers, but the structure of the design is constant. It is low level redundant scaling, which is known to lead to best effort reliability, according to the reliability folk theorem[Bur04a, HR94].*

2. *Exterior agents can be scaled by parallelizing entire hosts. This increases the total number of superagents with constant number of interior subagents. This is how we build a server farm, like a cloud environment.*

The main difference between these two is the adjacency structure, or how communication between the agents is wired.

What can we say about the host-tenant asymmetries? Within each superagent, there are actually many host-tenancy relationships working together. Referring to figure 7.39, we see a hierarchy of agency at a typical host. The stippled circle illustrates the superagent boundary of the host (running the system kernel), and the solid ellipses demark various subagents associated with it. The subagents form a variety of multi-tenant bindings, with different semantics. These are summarized in this table:

Role of Host	*Role of Tenant*
Process container	*Process interface*
Physical container	*Process container*
Physical container	*Physical interface*
Physical container interface	*Process container interface*
Physical network	*Process network*

A process container houses several process interface tenants, each of which pay by mediating a connection between the internals of the application and the process via logical network channels. The process network channel interfaces are themselves tenants of the physical network in the manner of figure 7.24.

In a similar way, but at a coarser scale, the physical interfaces are tenants of the physical host superagent, mediating physical network connections by direct analogy. Ironically, the process interfaces bind as tenants of the physical interfaces, drawing their agency from the network resource provided by the physical interface host. Such reversals are completely unproblematic, as they are simply semantics pointing towards a resource in a particular viewpoint of the collaborative ecosystem.

It is habitual in IT modelling to simply impose a hierarchy on these resource agencies, from a single viewpoint; however, the imposition of a hierarchy, without reference to the promises they make, leads to viewpoints that masquerade as authoritative, but are not.

What one learns from Promise Theory is that, for every asymmetric relation, there is a complementarity transformation which reverses the preferred interpretation and the natural ordering.

Let's focus on only one issue: addressability. Can we assign a coordinate system of names in a way that is faithful to the structure, and conducive to locating resources from any viewpoint? The possible varieties of naming of agencies are extensive, though not all are used in practice:

Agency	*Promised identifier(s)*
Physical interface	{ *MAC-addresses* }, { *IP-addresses* }
Process interface	{ *IP-addresses* }
Namespace	*Namespace umbrella name*
Physical container	*Hostname*
Physical IP-address	{ *DNS translation names* }
Process IP-address	{ *Namespace translation names* }
Process container	*Process container name*

Following industry practice, it is normal to coarse grain away several of the distinctions between these promises. This leads to problems in tracing the origins of promises made by some of the coarse grained agencies.

Well-known problems associated with design of the Domain Name Service (DNS) (especially reverse lookup) can be cited as an example of how coordinatization based on perceived containers, rather than general tuples leads to difficulty in tracing inter-agency collaborative processes. I'll leave it as an exercise to the reader to design an improved DNS service which acts as an invertible, based on the Resolvability lemma (lemma 18).

7.11.2 EXAMPLE: TENANT-ORIENTED HOSTING INFRASTRUCTURE

In the previous example, the low-level resource container (hardware) was considered to be a superagent boundary for processes running inside. However, we have great freedom to change the nature of a hierarchy in Promise Theory [130].

Functional thinking is invariably top-down. Computer science teaches top-down decomposition of problems through a logical branching process. Often this leads to trouble at lower levels due to inconsistency. The bottom-up aggregation process avoids inconsistency, and permits logical scaling, however we don't interface with systems from the bottom up. The 'cephalization' of agency around physical connectivity tends to fix attention on the host and its role as a kind of manager or brain. However, the applications (the minds of the system) are run inside the tenants, and there are more tenants than hosts. From a human perspective, then, the tenants see their concerns as a focal point, since they pay rental fees to support the host's existence as their service provider. The customer

service viewpoint thus weighs in semantically, where as the engineering viewpoint of the previous example was attractive dynamically.

Example 219. *In a computing platform, design for hosting software systems, multiple applications run as tenants of an infrastructure provided for them. This separation allows delegation with cooperation, sometimes called DevOps: tenants deal with content development, while the host deals with operational delivery.*

By analogy with the radio example 215 above, the agencies, which a user of the application would like to see, are shown in figure 7.40. The agent, in the role of user, does not want to see the internals or even the 'battery' that makes the thing work, it only cares about the surface boundary of the application, and its exterior promises. The illustration

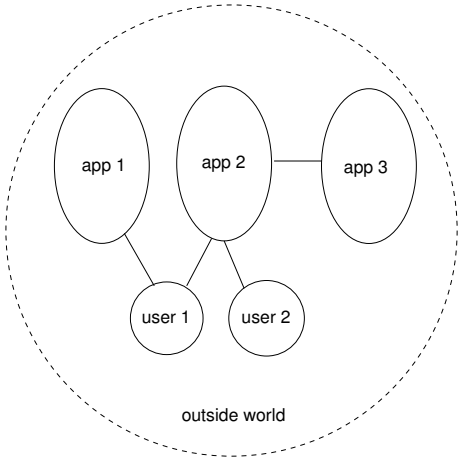


Figure 7.40: The user-application interface scale hides all details of what makes the application tick.

in figure 7.40 shows how tenants would like to conceptualize spacetime, placing their concerns as the top-most or outer-most interface. Once again, this is no limitation, since Promise Theory tells us that we have significant freedom to turn hierarchies and viewpoints upside down for convenience. With this viewpoint, the application tenant is in focus, and, from this user perspective, the application subsumes its infrastructure within the outer boundary. It wears user semantics on the outside.

The four concerns are:

1. The language of promise bodies

   This will include the exterior promises made by the application to its users, and

the interior promises made between the layers of hosts and tenants, described below.

At each scale of agency, one can imagine creating a 'compiler' from a domain specific body language to a more explicit pattern-generated explication of meaning for the lower level components.

$$\beta_M \rightarrow \beta_{M'} + \beta_M \tag{7.102}$$

2. Determine the conceptual head of the organism, from input/output flow

   The head of the organism is now the application-user interface (see figure 7.41).

3. Describe segmentation or tenancy/sharing relationships

   There are two layers of tenancy:

   (a) Application users are tenants of the application platform.

   (b) The application is a tenant of the infrastructure platform.

   The tenancies are based on approximately the following trades. For the user-application tenancy:

$$\begin{aligned} R &= \text{application account login} \\ C &= \text{user identification credentials(money?)} \\ f(C,R) &= \text{application functionality} \end{aligned} \tag{7.103}$$

   and for the application-platform tenancy,

$$\begin{aligned} R &= \text{platform login by application} \\ C &= \text{application identification credentials(money?)} \\ f(C,R) &= \text{pay by use computer, storage, network resources} \end{aligned} \tag{7.104}$$

   Maintainers of the application, and maintainers of the infrastructure on which the application resides will make different choices, depending on what semantics they wish to expose.

   At the level of computers, network, and storage, the picture might look something like figure 7.43. The infrastructure is composed of three types of agent: machine, storage, and network, which make promises accordingly. Applications are thus tenants of the platform infrastructure, and the main resources are represented the infrastructure scale in figure 7.43:

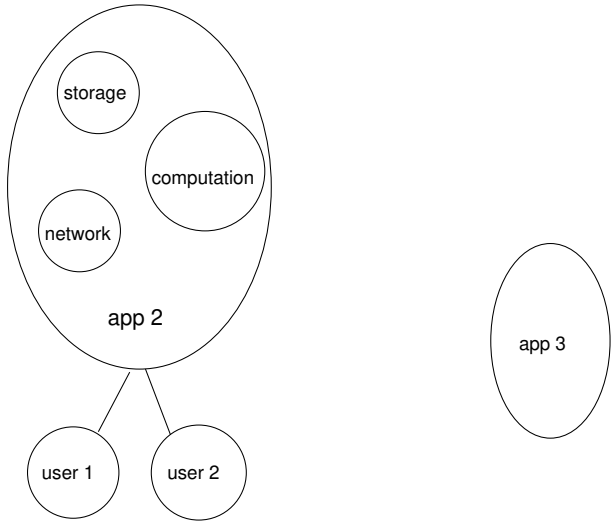


Figure 7.41: The application-platform interface exposes the resource knobs needed by application.

4. Explain the role of spacetime phase (solid, gas, hybrid, etc)

   From the application perspective, this is a single atomic agency, ready to bind to a user in a gaseous state. Inside the application, the state of the interior depends on the underlying infrastructure that powers the application.

The agency scales of interest are:

$$\text{User} : A = \{\text{application}, \text{user}\} \tag{7.105}$$

$$\text{Application} : T = \{\text{computation}, \text{storage}, \text{network}, \text{namespace}, \text{application}, \text{user}\} \tag{7.106}$$

$$\text{Infrastructure} : M = \{\text{process container}, \text{disk}, \text{router}, \text{application}, \text{box}, \text{computation}, \text{storage}, \text{network}, \text{namespace}, \text{interface}, \text{names/addresses}, \text{application}, \text{user}\} \tag{7.107}$$

Notice how, when increasing levels of detail, the higher levels are not replaced by the lower levels, since they still contain unique information (see section 5.8.13). Also, here I have introduced a semantic category scale 'box' to symmetrize over the resource providers that are interconnected by network channels (see figure 7.42).

$$\text{Box} : A = \{\text{computer}, \text{storage}, \text{router}, \text{interface}\} \tag{7.108}$$

In addition to these 'physical' agencies, there are other more abstractly defined agencies at play within an information system.

- *Names and addresses of machines and storage (IP/DNS addresses), for use by tenant name-services. These are private per application.*
- *Data from machines and storage, for use by tenant networks.*
- *Transport of data, for use by computer machines and storage (also represented as superagency 'box').*
- *Directory service lookups, for use by computers and storage, in order to locate others in their private network.*

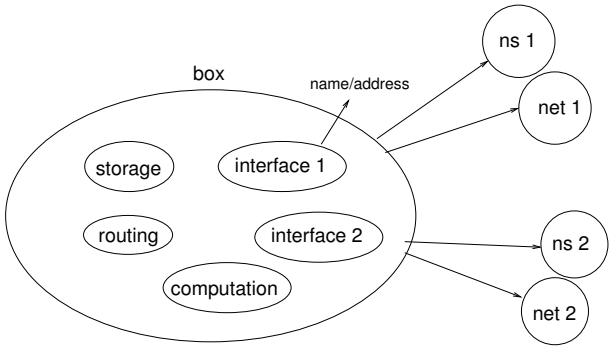


Figure 7.42: The network bindings to resource containers in a cloud computing instance. A box encloses private resources and interfaces create channels for networks to pass through from interior to exterior and vice versa.

The network binding to a resource agent has the form of a tenancy also (see figure 7.42), as a process container might be able to bind to multiple networks, in principle. An application might choose to branch its logic into multiple private channels, just as it chooses to branch sub-functions into private branches. In this reverse viewpoint, like in the previous example, applications can be considered to form private tenancies on top of the common spacetime, by using the intermediate agencies of the spacetime as proxies.

IT applications share a common infrastructure, provided for all applications by mutual cooperation. This forms a semantic spacetime, which is entirely interior to what we may call Application Space. Each application creates its own private world, as a tenant of the application itself.

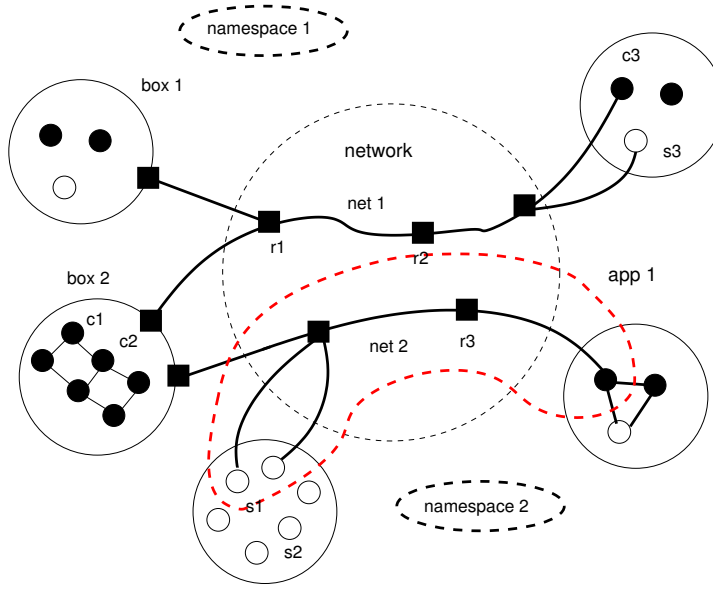


Figure 7.43: A partial IT infrastructure network, supporting multi-tenancy. Encircled clusters of agents are superagents show the tenancy scale. Inside these is the infrastructure scale: filled circles are computer nodes, unfilled circles are storage nodes, and squares are routing nodes. Network terminations at the box level are distributive to their constituent members.

Applications may thus be viewed as 'atomic' superagents, either locally or globally; thus, from a promise perspective, it is irrelevant whether the architecture is monolithic (implemented inside a closed space) or distributed (without fixed boundary). The promise abstractions are identical in both cases; the only difference is one of scale.

7.12 ABSTRACT AGENTS, AND KNOWLEDGE MODELLING

So far, most of the agencies we've discussed have been real entities, in a physical spacetime. It is also possible, even common, to construct entirely virtual 'knowledge spaces' of a more abstract kind, whose very structure is a representation of its semantics. This forms a contextual introduction for the general matters of chapter 12.

7.12.1 CLASSIFICATION, CATEGORIZATION, AND DISAMBIGUATION OF TENANTS

The branching of names into taxonomies is closely allied to the way narratives and storylines branch in logical reasoning.

Example 220 (Namespaces and hierarchy)**.** *The subsections in this section about tenancy form a hierarchy under the namespace of multi-tenancy. Trying to untwine and separate the concepts in to identify what is common and what is independent is a tricky challenge, which can quickly descend into a exercise of listing every related concept one can imagine. Partially ordering these to account for dependencies is a further challenge. This is one of the issues we struggle with when organizing information and representing knowledge.*

Classification does not only apply to idea, but also ideas we have about physical entities. Hence the power of a semantic space lies in making the distinction between pure information and information about a separate entity moot: in both cases the information has to be encoded by something physical, whether in the thing itself or in a proxy thing elsewhere.

Example 221. *Exclusive clubs are one thing, but categorization of tenants leads to explosions of new agency:*

- *Premium customers*
- *Schools for separate sex, race etc*
- *Cabin classes on airlines (economy, business, first class, etc)*
- *Car/truck parking*
- *Taxonomies of species or subject categories.*
- *Periodic table of elements.*
- *You can share a public bathroom, or you can have a private one*

Multi-tenancy is a bridge between the understanding of resources and the differential categorization of knowledge into concepts.

7.12.2 THE ECONOMICS OF IDENTITY SCALE: BRANDING

Consider the example in figure 7.44, logical reasoning creates a natural branching process that subdivides agencies into categories, often from a reductionist standpoint. Conversely, category labels act as logical hosts that unify similar contributors under a common label.

A book, such as an anthology of articles, is an aggregation of chapters, where the space R is rented out to chapters for text C. The book provides a vehicle for the chapters to be marketed under a common brand. In the same way, authors come together into categories of fiction, poetry, biography, etc, and the physical exemplars of the books share the same category, each contributing something to occupy the hosting of the category as a 'market brand'. The brand category develops relationships with an audience at a different level or scale than a single book or author can, hence there is a value to the hosting. This is sometimes called the economics of scale.

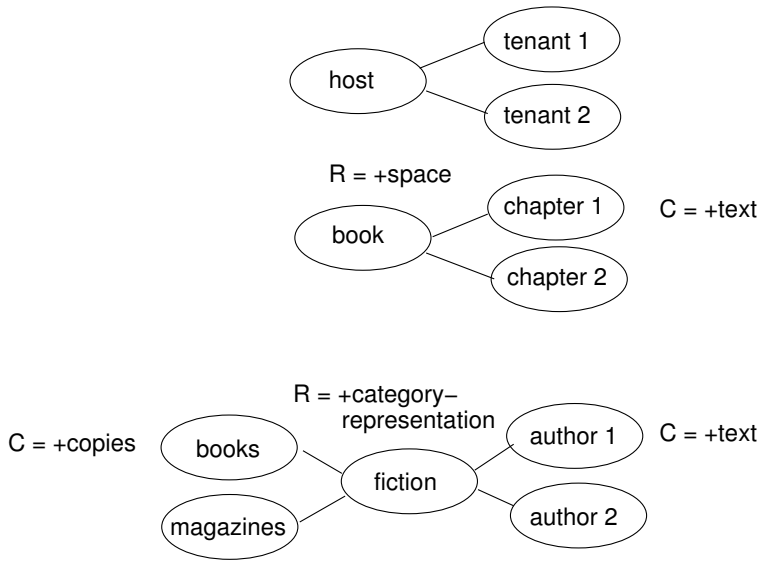


Figure 7.44: Orientation of host-tenancy roles. With host-ness to the left and tenancy increasingly to the right, we see that it is not always possible to think of tenants as being simple isolated 'things', especially when the promises they represent are more abstract. A tenant shares some space in a host, but what kind of space? A branching process might result in new tenants (+), or new hosts (-).

7.13 SUMMARY OF SCALING ISSUES

1. Scaling of the ability to discover information and promised services.

2. Exterior promise not kept.

3. Interior promise, supporting exterior promise, not kept.
4. Loss of agent transparency at scale (missing directory).
5. Distorted propagation of influence over distance (Chinese whispers).
6. Promise throughput inadequate.
7. Promise collaboration semantically deficient for the task, boundary conditions, or scale.
8. Agent weakness leading to node failure, with avalanche failure.
9. Promise takes too long at scale, because transmission scales differently to load.
10. Increased density of faults, leading to threshold 'critical mass' effect.
11. Accumulation of delays and uncertainties in multi-agent propagation.
12. Interference and diffraction of promises along different paths and cycles.
13. Unexpected (loss of) percolation of influence at higher density of agents or interactions.
14. Force density effects, attraction and repulsion of polar promises and bindings.
15. Scaling of comprehension (assessment), i.e. inability to serially process information from a bulk region.
16. Scaling of 'agility', inertia caused by the relative change in efficiency with which promise propagation occurs at scale.
17. Scaling of response to mass 'events', i.e. arrival processes (serialization).
18. Scaling of impact, i.e.

$$\text{Impact} = \frac{\text{Affected agents}}{\text{All agents}}. \tag{7.109}$$

19. Scaling of a 'fault surface' or 'attack surface' relative to system size.
20. Scaling of feedback, leading to (de)stabilization, at short range or at long range.
21. Amplification of small errors into large ones by unstable branching.

CHAPTER 8

SCALING OF PROCESS AND WORKFLOW

We can consider the various changes to promise-keeping systems independently of each other, in the time-honoured tradition of considering 'orthogonal' or independent variables one at a time. The concept of 'work' has both quantitative and qualitative connotations. Workflow, or 'throughput scaling' is a dynamical issue: it's about how we count the occurrence of outcomes with constant semantics, and is analyzed by characterizing the quantitative output of a system as a function of input (usually a task). The semantics of workflow are sometimes described by graphs or flowcharts.

Workflow may be divided into a number of process trajectories. Each workflow trajectory is serialization of effort, a one dimensional (temporal) view of a system—a queue. So—just as in calculus—we can seek to examine the behaviour of a multivariate (or multi-promise) system by keeping some variables constant as we vary others. In this way, we retain the ability to use familiar tools for orthogonal matters of semantics and dynamics.

The ability of a system to complete tasks—or keep the promise of outcomes—depends on the extent to which the task can be broken up into independent streams, which can be completed independently (i.e. interleaved in series or in parallel), and recombined into a coherent whole. Thus the topology of the system and its communication channels play a role. Some simple ideas about scalability can be understood using the flow approximation of queueing systems, in which a promise kept involves a continuous stream of data, observations, changes, etc.

8.1 DISTRIBUTED PIPELINES: FORCE DRIVEN WORK

The classical approach to industrial work is to feed material into a queue and to process or transform it in some way. When this becomes a regular task, it takes on the character of a production line. In IT, commonplace examples include the processing of data analytics, image rendering, compilation, software packaging, billing, even mass e-mailing. In some cases these processes or 'pipelines' are initiated in response to an independent process of curated changes committed by pushing a button, such as in a versioned software deployment model. In others, there are triggered by continuously arriving events that may be accumulated into batches. The optimization of a pipeline may depend on several factors, economic as well as the structure of dependencies.

For data processing, a significant point of departure occurred when single monolithic systems were replaced by plural distributed systems at much greater scale. This brought the complexity of distributed systems concepts into the mix, with all their attendant indeterminism.

The tendency for many is to think of a distributed pipeline as a single deterministic Petri net[DA94, MMS85] (i.e. a directed, mainly acyclic network of queues[Kle76, GH98, Buz73, Nel95]). But, in a distributed system, there are many complex and overlapping influences at work that make non-determinism unavoidable. There are two viewpoints:

- Triggering or 'ballistic' transactional processing model, with freely branching push semantics (see the detailed discussion in [Bur13a], for instance).
- Polled and voluntary processing with batched data collection, based on predictable pull semantics.

These two approaches may not be as different as casual inspection suggests, if one adjusts the timescales for sampling/polling to be very high, they behave in a similar way. However, in the worst case, a freely branching process, driven unconditionally by event arrivals, is essentially an uncontrolled explosion, which could consume unbounded resources. The ability to regulate flow depends on a careful understanding of 'pull'.

8.2 CONTINUITY AND DISCRETIZATION

To protect against its effects, an obvious strategy is to stabilize execution, by discretizing time and work into batches (chunks) that have a stability of their own, averaging away random variables. Further, principles for process safety, avoiding contention and uncontrolled repetition, in the scheduling of parallelized work across shared resources have been in use for decades [BS97], forming a basis for safe execution of tasks, with

convergent outcomes[Bur04b]. The present opportunity of the cloud era is to absorb these well known control structures into orchestration schedulers, of which Kubernetes currently represents the state of the art[Bur15b], and build on them as basic reliable properties. With a platform that promises these basic properties, we can treat pipelines as an end-to-end delivery problem based on trusted intermediaries[BB14a], without getting bogged down in too many technicalities.

Example 222 (High level pipeline promises)**.** *There are several aspects to a data pipeline we have to consider:*

- *Processing architectures, so called Directed Acyclic Graphs (DAG), essentially 'trees', for specifying a process representation.*
- *Deciding whether updating should push or pull semantics, for how data flow. This can potentially be optimized without user intervention, but depends principally on the nature of the data source.*
- *Whether processing uses 'windowing' aggregation for economics and scalability of processing.*
- *How intermediate caching of atomic pipeline artifacts are kept and versioned, for optimizing recalculation or resuming broken pipes.*

Figure 8.1 attempts to show some of the issues in positioning a pipeline technology solution in data processing. There are three dimensions: on the vertical axis, we have the timescale for process resolution, from manual button-push, to occasional batch jobs, and all the way up to continuous operation. On the horizontal axis, the level of smartness of inbuilt automation that takes away pain is increasing. Finally, the size of the circles indicates how self-contained the approach is. Small circles may cost users a lot of overhead in setting up. So the desirable solutions are large and far up to the top right.

8.3 EVENT DRIVEN SYSTEMS

Can a system truly be considered event driven? Certainly interrupts are an example of this, so it can be done. However, detection of events cannot be achieved without a robust and fast interior time. Interior time limits the ability to respond to events, so to some extent this is illusory. Event driven is thus something of an illusion for the cases where neglecting considerations of interior process scale can be argued.

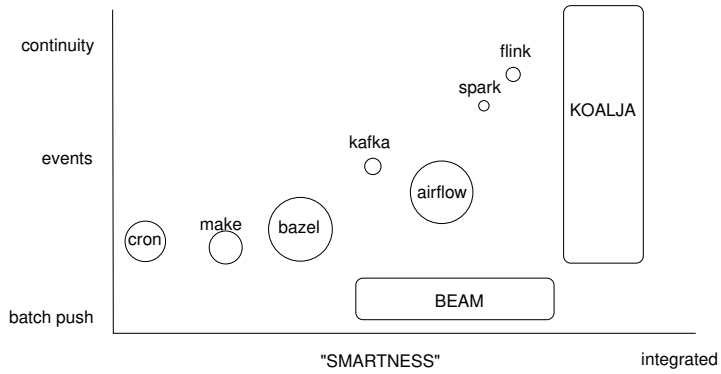


Figure 8.1: A heuristic plot of perceived effort versus efficiency involved in data processing pipelines. Push the button batch processing ranges from manual command line start to scheduled cron/at jobs. Make and Bazel add declarative semantics and dependency tracking to this, leading to DAGs. Then clusters of graphs can span multiple hosts and deal with random arrivals, and handle aggregation or windowing, from scheduled intervals to event triggered clocks. The circle sizes represent the freedom from dependency complexity, and bigger is better for taking away user pain.

8.4 EVENT-BASED WORKLOADS, ARRIVAL PROCESSES, AND QUEUES

Faults and repairs do not really happen according to a constant current, though this is a convenient simplification that helps us to gauge mean behaviour over long times. Rather, faults occur sporadically as events. Events arrive a 'random' times. A constant current is an approximation that is equivalent to a Poisson arrival process.

An arrival process is a series of random arrivals, distributed in time, with average arrival rate λ is processed serially by a server at processing rate μ. Think of rain falling (a random process of arrivals), and being led away down a drain. All we are interested in, over a steady state in time, is how much rain falls and whether we can drain it away quickly enough by introducing a sufficient number of drains (processors or servers). Thus scalability is usually discussed as throughput as a function of load. Sometimes the input is a function of a number of clients, and sometimes the output is a function of the number of servers (see fig. 8.5). There are many ways to talk about scaling behaviour.

Queueing theory deals with an abstracted model of queueing. A queue is a serial process of arrivals that is processed by a server. There are many models of queues, with arbitrary complexity, but there are some simple conclusions that summarize the pertinent

results for most purposes. The simplest queueing results are for Poisson distributed arrivals and process dispatches, but they can be generalized to other distributions. For all the mathematics, there are two results that are important here:

1. The average response time between sending an arrival into a single queue, and its successful processing by a single server is:

$$R = \frac{1}{\mu - \lambda}. \tag{8.1}$$

2. The traffic intensity λ/ρ is the dimensionless scaling parameter that controls queue behaviour.

3. For Poisson arrivals, this is called the $M/M/1$ queue, designating arrival/dispatch/servers, where 'M' stands for Markov process.

4. As the arrival rate approaches the servicing rate, the response time grows to infinity very suddenly (power law scaling), and the system becomes unstable (see figure 8.2).

5. As the queue reaches instability, the response time ceases to grow linearly and the server CPU rate becomes saturated (see figure 8.3) as processes contend over the shared CPU resource.

6. Thus the queue demonstrates the contention for throughput, when filling an interval of time with random arrival events.

A single queue can only promise its best service rate, so it can only be 'scaled up' to higher workloads by improving the service rate μ. However, if events can be handled in parallel, then we can also add more agents to promise the service, at the possible expense of making them work together. In the simplest form, queueing theory assumes that all jobs are of equal magnitude, and the system is completely homogeneous. It is a simple order of magnitude estimate. However, what it does is to expose the existence of an instability, or a critical dimensionless ratio of arrivals to dispatches that causes the linear processing of the system in time to blow up, or fail to keep its promise of a timely completion.

8.5 SCALING THROUGHPUT WITH MULTIPLE SERVERS

We can add more servers in one of two different ways (see figure 8.4). Either we multiply the whole system by a factor of n, i.e. have n queues with n servers, and arrivals enter one of these at random. This is randomly chosen parallelism, called (M/M/1)^n. The other

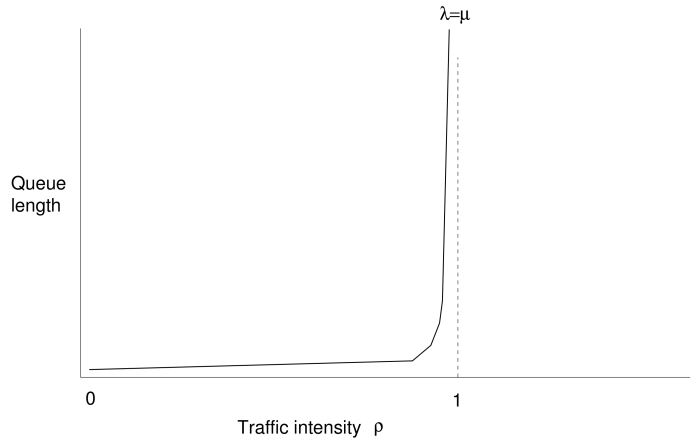


Figure 8.2: A queue behaves well when the arrival rate is far below the service rate, then it undergoes a rapid collapse into thrashing contention.

alternative is to have a single queue being dispatched by n servers working in parallel, so all arrivals enter the same line and are processed by the first available server. This is called the M/M/n queue.

It is a provable assertion that the $M/M/n$ is never worse and often faster than the $(M/M/1)^n$, because severs can be kept busy all the time by marshalling the flow. However, we should not overstate the importance of this. It was shown that a simple voluntary load balancing technique, either round robin push or voluntary server pull could perform as well as, or better than $M/M/n$[BU06, BB08] in practice, since the benefits occur mostly close to saturation. The latter is the benefit of a pull based architecture, where the servers pull from the queue when they can. Pushing data into a random queue risks entering a busy queue, while other queues are empty and idle.

The serial load-balancer or dispatcher architecture is a common design in all kinds of services. It has a basic flaw which is the serial bottleneck introduced by the load balancer. It could be avoided by direct routing of traffic to servers.[131]

8.6 HORIZONTAL AND VERTICAL SCALING OF WORKLOADS

Two scaling patterns are commonly referred to when discussing workloads: horizontal and vertical scaling. These can be understood easily from figure 8.6. If we increase

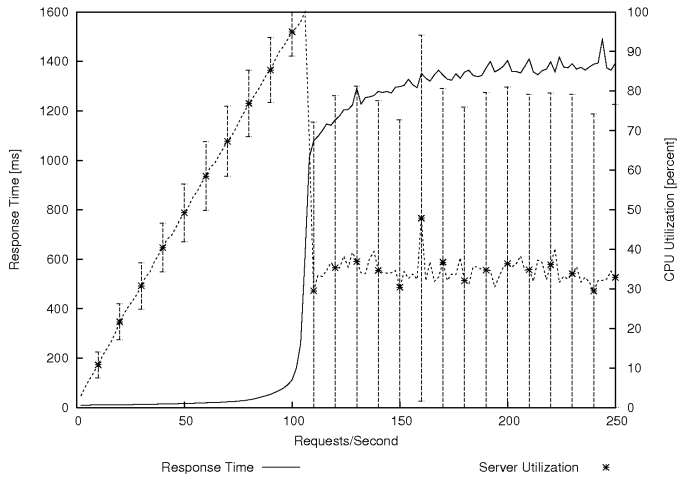


Figure 8.3: A queue behaves well when the arrival rate is far below the service rate, then it undergoes a rapid collapse into thrashing contention.

the size of a distributable workload, we have two choices to accommodate it: either increase the size of the container or handler (called vertical scaling) or to increase the number of containers or handlers (called horizontal scaling). Since each serial handler involves a processing queue, the queue length, the filling fraction of the handler agents, or their capacity utilization, is varied to place all workload fragments into an appropriately measured queue. This is sometimes called the bin packing problem[SKAEMW13].

We can formalize these definitions:

Definition 174 (Vertical scaling)**.** *Varying the capacity of a single agent in order to increase or decrease the throughput of a system. The utilization is increased by adding workload at constant capacity or vice versa, or is reduced by increasing work channel capacity.*

Definition 175 (Horizontal scaling)**.** *Distributing a workload across a cluster of agents, of constant capacity, in order to control the utilization of agents.*

The two approaches have different properties. If we now think of the handler agents as queues, we can see that filling up the queues to an economic level of utilization, without exceeding the queueing instability threshold is the key problem we face in scaling systems. Moreover, we should recall that the resource sharing queues may have

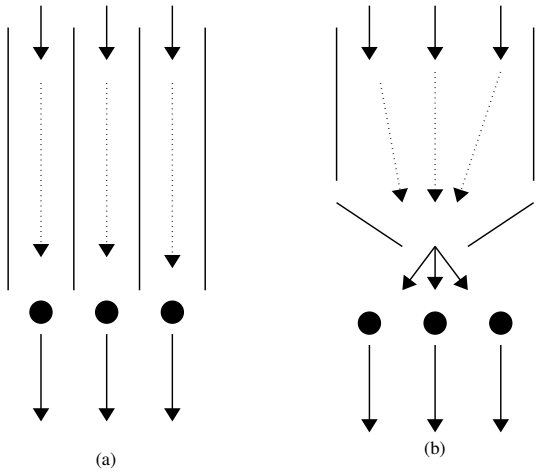


Figure 8.4: Multiple servers can be introduced in two ways: (a) by replicating the entire queue n times, and (b) by making a single queue services by n servers. Of these (b) is always has the shortest average response time, but it introduces a bottleneck and a longer single queue.

several dimensions that scale unequally, e.g. memory capacity and CPU capacity, or human brain capacity and physical strength.

Vertical scaling supposes the ability to continuously elastically increase the capacity of a handler agent to meet demand. Horizontal scaling assumes that the handler agents have a fixed commodity size, and that one simply needs to add more.

Type 2 (vertical) looks like a flexible elastic boundary, while the type 1 (horizontal) looks like an infrastructure enlargement but this is a fiction of scale.

Example 223. *In older mainframe computer designs, such as the IBM z-series mainframes, redundant CPU capacity could be pre-installed and activated on demand to elastically scale the vertical processing capacity of the mainframe.*

In cloud computing, such as Amazon Web Services, elastic scaling refers to the automated ability to add more horizontal server capacity. Because the infrastructure is prefabricated, this can be made to look effortless and like a scaled version of vertical scaling approach. The difference lies in the hidden interior details of the superagent, such as whether the network communications bus scales without bottlenecks to the workload. In practice, IT engineers think mainly about the CPU and memory, less about the network, meaning that elastic scaling is only elastic in some parameters, placing a growing burden on others. This is not universal scale-free behaviour.

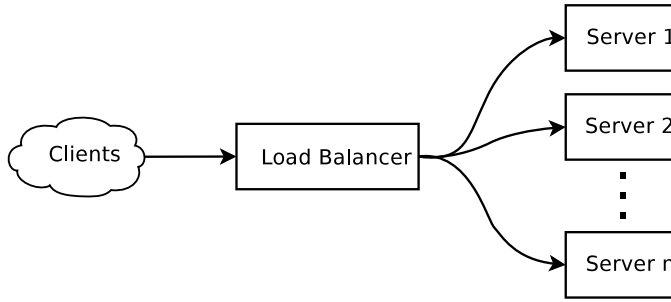


Figure 8.5: The serial load-balancer or dispatcher architecture is a common design in all kinds of services. It has a basic flaw which is the serial bottleneck introduced by the load balancer. It could be avoided by direct routing of traffic to servers.

If we create a superagent from discrete processing elements, by drawing a boundary around the agents in the lower right (horizontal scaling) corner of figure 8.6, and zoom out, then the picture looks just like the vertical scaling picture next to it. So the question of whether elasticity is a form of discrete (horizontal) or continuous (vertical) elasticity is only a matter of total scale of consideration. These are not fundamentally different. What *is* different about the discrete fixed size containers in horizontal scaling, is that the nature of the phase transition the system undergoes is different when one is at approximately the same scale as the handler agent containers. A phase transition is a change in a system from one qualitative mode of behaviour to another. The freezing and melting of ice to water are a mundane example of this. In a type 1 phase transition, there is a sudden and discontinuous change in the behaviour of a system (e.g. from hard ice to liquid water, with nothing in between). In a second order phase transition, there is only a smooth and gradual shift from one regime to another (e.g. from hard butter to soft malleable butter, eventually liquid). Once again, these are scale-dependent phenomena. As we zoom out to a level of large enough scale, all phase changes will eventually appear to be second order, as local regions of bulk scale take on the role of agents with different promises. As always, dimensionless variables determine the universality or dynamical similarity of the scale.

Example 224. *Adding a new server to a job, or a new data centre, is a horizontal, inelastic scaling. These cases might become equivalent if there are sufficient datacentres to match the servers in a datacentre, i.e. the ratio of datacentres to servers per datacentre approaches 1, but this depends on how the numbers are utilized relative to the workload. It is the scale of the workload relative to the scale of the server infrastructure that determines the behaviour.*

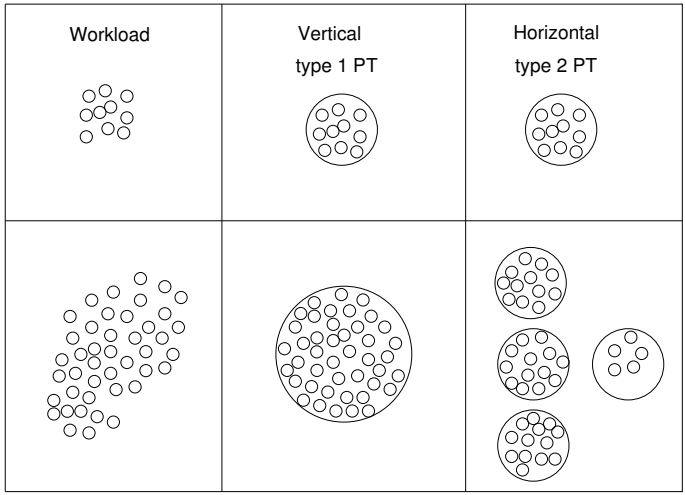


Figure 8.6: Horizontal and vertical scaling of an incompressible workload can be understood simply by thinking about how we fit the contents to containers. This is sometimes called the bin packing problem.

8.7 ECONOMIES OF PROCESSING SCALE

One of the main considerations in 'scaling up' operations is the idea that resources can be saved by consolidating (recentralizing) certain promises. This is worth some words of explanation, as both IT and economics texts misrepresent this in various ways. Economies of scale are practical adaptations that are 'illusory' in the following sense: they are about the reduction of waste, not about the amplification of intrinsic capabilities. When we see superlinear scaling (see sections from 5.15), this has to come about from the compressibility of inefficiency, because there is no free lunch.

Figures 8.7 and 8.8 attempt to illustrate how sparsely used infrastructure can be compressed by consolidation. The demand arising from four agents that are underutilized can be handled with approximately the same efficiency by a single agent with higher utilization, provided we do not get close to critical utilization of the single agent. This explanation is borne out by studies of the scaling in metropolitan environments[BLH+07, Bur16b].

For a deeper microscopic understanding of this we need to look at queueing theory in section 8.4. For now, these figures capture the intuitive picture quite well. Economies of scale transmute wasteful *horizontal scaling* into *vertical scaling*, which has a critical

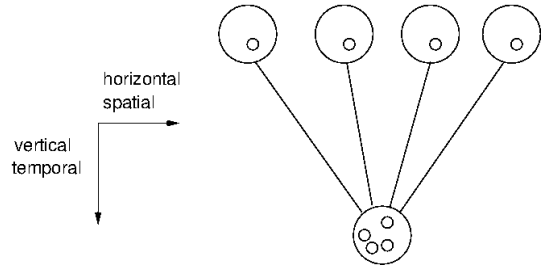


Figure 8.7: Packing or workload compression is the flip-side of queueing theory.

threshold limitation. The classic S-shaped curve on the right of figure 8.8 shows how output can be increased as a function of utilization (or equivalently dependency on demand load) up to a limit, at which point no further output can be mustered by the system. What this simple diagram shows conceals the interior trauma that happens to an agent as it approaches this critical threshold. This is a subject we need to explore in some detail in this section.

Example 225. *The bin packing of algorithms used by large scale resource managers attempt to optimize economies of scale by filling vertically to capacity, while keeping the utilization below the instability threshold.*

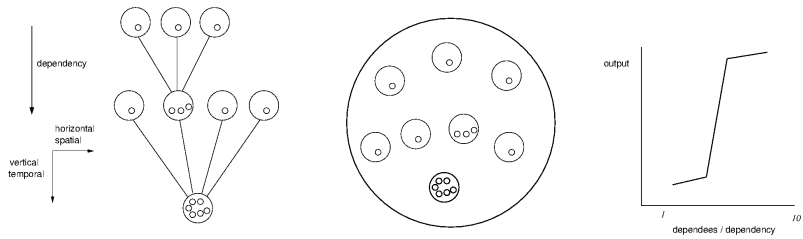


Figure 8.8: Packing or workload compression is the flip-side of queueing theory. Now the outer boundary can grow without the inner dependency growing in scale. This is an economy of scale. If we think of the dependency as the reference scale, then the size of the whole scales superlinearly, more than in proportion to the size of the dependency.

Economies of scale are about compressing spare capacity. If we are lucky, systems can grow cheaply without hitting utilization limits. Consolidation of agents is a form of data compression in information theory . If we improve infrastructure along side

increased consolidation, then we can achieve so-called superlinear scaling, in which the efficiency gain appears to grow faster than the number of agents involved. This is because increasing the number of agents must lead to linear scaling, while compression of workloads adds to the gain. The result is the appearance of amplification by better utilization.

8.7.1 AMDAHL'S LAW FOR PARALLEL PROCESSING

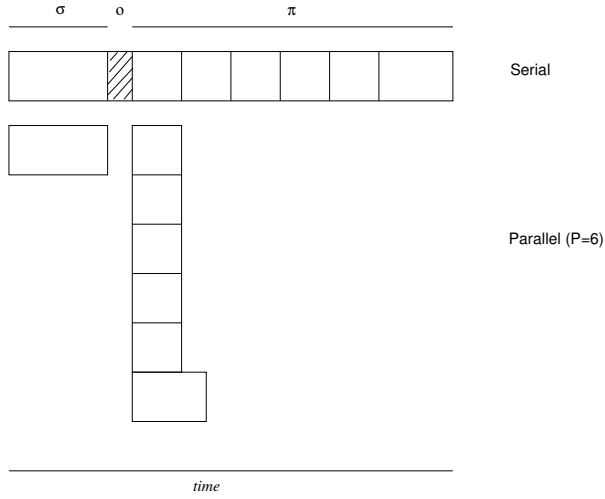


Figure 8.9: Representation of Amdahl's law. A typical task has only a fraction that is parallelizable. The serial part cannot be sped up using parallel processing or load balancing. Note that one is not always able to balance tasks into chunks of equal size, moreover there is an overhead (shaded) involved in the parallelization.

Another order of magnitude estimator is Amdahl's law[Amd67] Amdahl's law, named after computer designer Gene Amdahl, was one of the first attempts to characterize scalability of tasks in High Performance Computing, in relation to the number of processors[Amd67]. It calculates the expected 'speed-up', or fractional increase in performance, as a result of parallelizing part of the processing (load balancing) between n servers or processors as:

$$S(N) = \frac{t_{\text{serial}}}{t_{\text{parallel}}} = \frac{T(1)}{T(N)} \tag{8.2}$$

Suppose a task of total size $\sigma + \pi$, which is a sum of a serial part σ and a parallelizable part π, is to be shared amongst N servers or processors and suppose that there is an

overhead o associated with the parallelization (see fig. 8.9). Amdahl's law says that:

$$S(N) = \frac{\sigma + \pi}{\sigma + \frac{\pi}{N} + o}. \tag{8.3}$$

In general, one does not know much about the overhead o except to say that it is positive ($o \geq 0$), thus we write

$$S(N) \leq \frac{\sigma + \pi}{\sigma + \frac{\pi}{N}}. \tag{8.4}$$

This is usually rewritten by introducing units of the *serial fraction*, $f \equiv \sigma/(\sigma + \pi)$, so that we may write:

$$S(N) \leq \frac{1}{f + \frac{(1-f)}{N}}. \tag{8.5}$$

This fraction is never greater than $1/f$, thus even with an infinite number of processors the task cannot be made faster than the processing of the serial part. The aim of any application designer must therefore be to make the serial fraction of an application as small as possible. This is called the bottleneck of the system.

Amdahl's law is, of course, an idealization that make a number of unrealistic assumptions, most notably that the overhead of sharing a task between a number of equivalent subagents is zero. Overhead is introduced by dispatchers or intermediaries. Also, it is not clear that every task can, in fact, be divided equally between a given number of processors. This assumes some perfect knowledge of a task with very fine granularity. Thus the speedup is strictly limited by the largest chunk (see fig. 8.9) not the smallest. The value of the model is that it predicts two issues that limit the performance of a server or high performance application:

- The serial fraction of the task[132].
- The processor entropy or even-ness of the load balancing.

Amdahl's law was written with high performance computing in mind, but it applies also to load sharing for any kind of network services. It applies to superagent scaling in a promise model, up to the assumptions mentioned. If one thinks of an entire job as the sum of all requests over a period of time (just as drain-water is a sum of all the individual rain-drops) then the law can also be applied to the speed up obtained in a load-balancing architecture such as that shown in fig. 8.5. The serial part of this task is then related to the processing of headers by the dispatcher, i.e. the dispatcher is the fundamental limitation of the system.

Comment 16 (Current approximation)**.** *Another way of looking at Amdahl's law in networks has been examined in [BC03, BC04] to discuss centralization versus distribution of processing. In network topology, serialization corresponds to centralization of processing, i.e. the introduction of a bottleneck by design. Sometimes this is necessary to collate data in a single location, other times designers centralize workflows unnecessarily from lack of imagination. If a centralized server is a common dependency of n clients, then it is clear that the capacity of the server C has to be shared between the n clients, so the workflow per client is*

$$W \simeq \frac{C}{n}. \tag{8.6}$$

We say then that this architecture scales like $1/n$, assuming C to be constant. As N becomes large, the workflow per client goes to zero which is a poor scaling property. We would prefer that it were constant, which means that we must either scale the capacity C in step with n or look for a different architecture. There are two alternatives:

- *Server scaling by load balancing (in-site or cross-site) $C \to Cn$.*
- *Peer to peer or mesh architecture (client-based voluntary load balancing) $n \to 1$.*

8.7.2 GUNTHER'S UNIVERSAL SCALABILITY LAW FOR PROCESSING

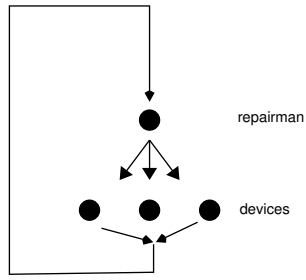


Figure 8.10: The machine repairman model is a queue where N parallel producers of traffic are handled by a single queue for repair.

In a similar vein to Amdahl's law, there is Gunther's Universal Scalability Law[Gun93, Gun08], which was proven to be derivable from a Machine Repairman model assumption (see figure 8.10). Like Amdahl's law, it assumes that the processing is due to multi-agent processing, and is careful to define scalability rather than scaling. It generalizes the

service capacity speedup function $S(n)$ there by adding the cost of coordinating the n servers in the service pool, which is of order $n(n-1)$. This corresponds to the overhead term in 8.3.

$$S(n) = \frac{n}{1 + \alpha(n-1) + \beta n(n-1)} \tag{8.7}$$

In principle there could be higher order terms too, but these become decreasingly relevant to performance (they may affect semantics). For $\beta = 0$, this reduces to Amdahl's law. The denominator terms describe, a constant term that scales with the parallel concurrency of the numerator; an order n term, with coefficient α, which describes contention between the servers; and an order N^2 term, with coefficient β, which describes consistency (coherency) and network coordination between the parallel servers[133], i.e. the cost of equilibration of cached data (see figure 8.11). We see sub-linear scaling due to the cost of coordination.

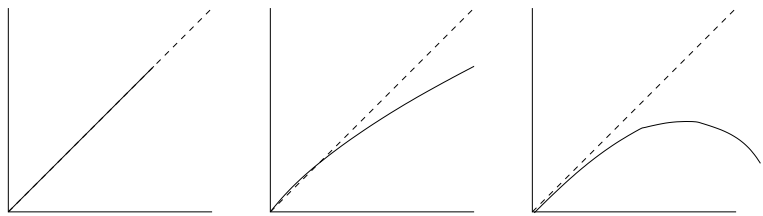


Figure 8.11: The Gunther universal scaling law for its control parameters. (a) linear scaling, (b) cost of sharing resources and diminishing returns from contention, (c) Negative returns from incoherency and the cost of equilibration.

Interestingly, superlinear scaling has been identified with negative values of α the model. These negative values are effective values in the scaling relation that are fed by recovery of an inefficiency elsewhere[134]. The finiteness of system resources seem to indicate, however, that initial superlinearity may have to be paid back with a quadratic term later, which indicates that the superlinear graph must cross back into a sublinear and eventually degrading performance cost for large superagent clusters. Thus allowing large superagent scaling in a system could itself be a design flaw[135].

8.7.3 EFFECTIVE POWER LAW SCALING FROM AMDAHL'S AND GUNTHER'S LAW

The Amdahl and Gunther scaling relations are workload scalings, not universal scaling relations. Let's consider how we could derive an approximate universal scaling relation from these. This cannot be exact, since universal scaling requires a continuum description, rather than a discrete agent model. Nonetheless, it's interesting to probe the matter from a scaling perspective.

The focus on relative 'speed up' is a point of view that is mainly of use in parallel computation. If we want to know how the time fraction (speed up) scales for as a function of the number of processors N, then we can compare N with γN. Then, we can write, for $\gamma > 1$

$$\frac{T(\gamma N)}{T(N)} = \frac{\sigma + \frac{\pi}{\gamma N}}{\sigma + \frac{\pi}{N}} \tag{8.8}$$

$$= 1 + \frac{\left(\frac{1}{\gamma} - 1\right)\frac{\pi}{\sigma n}}{1 + \frac{\pi}{\sigma n}}. \tag{8.9}$$

Let $\delta = \frac{D}{D+H}$, where $D = 1$ and $H = \frac{\pi}{\sigma n}$, then, approximating as a binomial expansion,

$$\frac{T(\gamma N)}{T(N)} = 1 - \left(\frac{\gamma - 1}{\gamma}\right)\delta. \tag{8.10}$$

$$\simeq \left(1 - \left(\frac{\gamma - 1}{\gamma}\right)\right)^{\delta} \ldots \tag{8.11}$$

$$\simeq \gamma^{-\delta}. \tag{8.12}$$

Thus we have an approximate power law fit for large N compared to π/σ, and we can write

$$T(N) \simeq T_0 N^{-\delta}. \tag{8.13}$$

i.e. there is a marginal relative economy of scale for small N, which decays to an essentially scale invariant constant result. If we allow, like Gunther, for the presence of equilibration, or mesh coherence effects, then we could use the form

$$T(N) = \sigma + \frac{\pi}{\gamma N} + \kappa N, \tag{8.14}$$

where κ represents linear time take to poll each of the worker agents. This is the case where replication and consistency are required. With this extra term, we have

$$\frac{T(\gamma N)}{T(N)} = \frac{\sigma + \frac{\pi}{\gamma N} + \kappa\gamma N}{\sigma + \frac{\pi}{N} + \kappa N} \tag{8.15}$$

$$= 1 + \frac{\left(\frac{1}{\gamma} - 1\right)\frac{\pi}{\sigma} + (\gamma - 1)\frac{\kappa}{\sigma}N^2}{1 + \frac{\pi}{\sigma} + \frac{\kappa}{\sigma}N^2}. \tag{8.16}$$

Now the behaviour doesn't separate cleanly, and there are two regimes of approximate power law scaling, with something more messy in between. Using the same procedure as before, we get an anomalous term for $\kappa \neq 0$:

$$\frac{T(\gamma N)}{T(N)} \simeq \left(1 - \left(\frac{\gamma - 1}{\gamma}\right)\right)^{\delta} + \frac{\kappa(\gamma - 1)N^2}{\pi + \sigma N + \kappa N^2} \tag{8.17}$$

$$\simeq \gamma^{-\delta} \quad (\kappa \ll \sigma, N\text{small}) \tag{8.18}$$

$$\simeq \gamma \quad (\kappa > 0, N\text{large}) \tag{8.19}$$

$$\tag{8.20}$$

So for large N, with $\kappa > 0$, we have simply

$$T = T_0 N, \tag{8.21}$$

i.e. the scaling cost becomes linearly worse with increasing size.

When we compare these results to a spacetime scaling, it will become apparent that this takes the approximate form of a scaling law in a one-dimensional spacetime $D = 1$, with Hausdorff dimension $H = \pi/\sigma n < 1$. This indicates that a serial workflow, with some parallelism, is essentially a one dimensional problem, with some fractal complexity in its trajectory due to parallelism[136]. Interestingly, as the parallelism increases, the duration of the fractal dimensionality shrinks to nothing. Thus the large N limit for serial processing tends to squeeze the degrees of freedom in the system. We can compare this result to the case of general spacetime involvement in section 5.15.

What kind of promises should agents make in order to support expectations?

8.8 DATA PIPELINES AGAIN

We want to study the composition of different pipeline use-cases to abstract necessary and sufficient promises its components need to keep. We're interested in defining the distinct configurations and characteristic for what is essentially a network of queues, based on messages sent from senders S to receivers R, in the topology of a directed acyclic graph (DAG).

8.8.1 TOPOLOGY OF WORKFLOWS

Users want to focus on an outcome storyline with their processing, and suppress concerns that are not germane to that narrative. All reasoning and programming is essentially storytelling, which is why it is important to daily business—we want to keep the story they are trying to sell in focus, and not be drowned by technicalities of modern infrastructure.

Although we call it a pipeline, the structure of a workflow is not necessarily a linear chain, but may have branching points and confluences. Even the simplest DAG 'storyline' has a tributary structure, like a river accumulating inputs from multiple independent side channels (see figure 8.13):

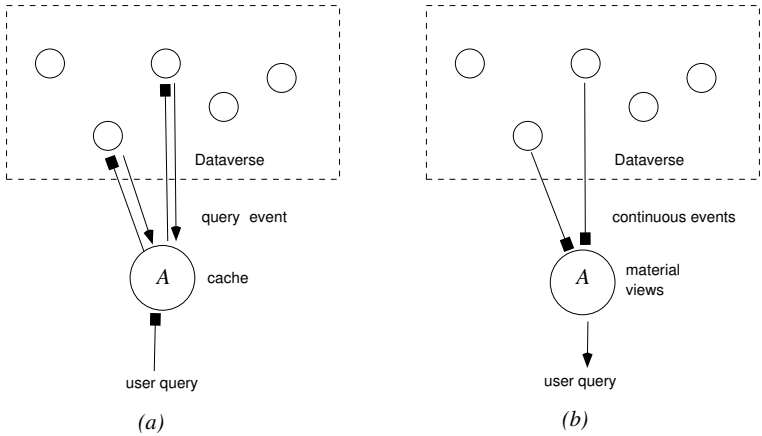


Figure 8.12: Data 'pipelines' may work in one of two modes: (a) a direct query (a downstream agent imposes a request) to upstream distributed data sources, with latent response (like intentionally fetching water from a well), or (b) in the context of a continuously updating stream (promise) imposed by upstream, received into a cache are always available (unintended rain collection into a reservoir). In both cases, to unlock a flow, users initiate a data query from a programming API to move the workflow narrative forward.

That's because every stage relies on dependencies, like the software, execution platform, and helper services (storage, DNS, etc). These all impact the result to some degree. In the weakest case, they may simply add to latency. In the worst case, they might affect the nature or semantics of the outcome, contending through covert channels of shared resources.

8.8.2 INTENTIONAL AND UNINTENTIONAL SOURCES

Data collection is one of the central constraints on a pipe. The material that goes into a pipeline may arrive involuntarily (unintended like rain), or it may be collected voluntarily (intended, like fetching water from a well), see figure 8.12. So, there are two ways to initiate work:

- Intentional workflow (scheduled workflow)

- Unintended workflow (random event stream processing) similar to a 'service'[137].

We tend to focus on the DAGs in scheduling, because the DAG tells the storyline, and follows the intent. But it is not the whole story. There are service dependencies along side, which (in some sense) are even more critical as they must be highly available and changes may conceal influences on the result that become unplanned and unintended. Classic examples are DNS, storage, etc. (see figure 5.248). These side channels usually dominate the management task in deploying a data processing job.

8.8.3 STAGING AND CACHING INTERMEDIATE RESULTS

As data 'events' enter a system, they undergo a sequence of operator transformations resulting in intermediate output states that become the inputs for the next stage, following the DAG (see figure 8.13). These intermediate results form checkpoints in the computation, and can be cached for repeated use. Any kind of intentional search for criterion matching will lead to a significant delay. Some of these intermediate stages

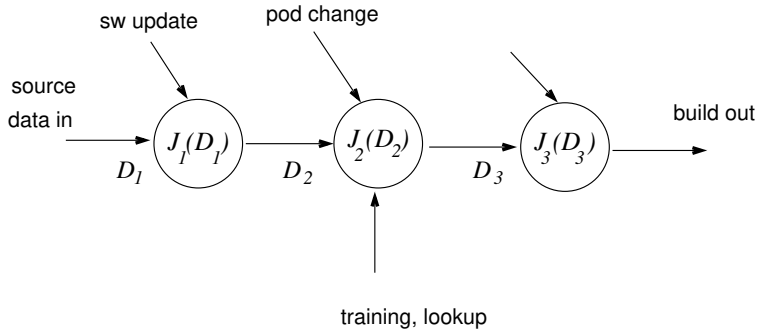


Figure 8.13: Channels and side-channel change. The flow of inputs in a pipeline is a river with tributaries, and each stage is a subcluster.

(as mentioned in figure 8.12) may be reusable for updates and fault repair. It allows processes to be picked up from where they left off, rather than being started again from scratch, following an interruption[138].

Buffer caching also brings greater regulation of the stream by allowing stages downstream to pull work from the pile at their own rate (as the rain comes in, it fills up cascading wells that are empties according to a scheduled process, restoring an artificial determinism to the computation). Formally, the sequential processing makes these cached versions into a queue, but it might not be a strictly ordered queue (a database of arrivals is sufficient). A pull approach can also be used to query the initial sources in some cases,

e.g. IoT devices at the edge of the network, can be polled rather than expecting them to push data according to an awkward perception of 'realtime', forcing network congestion. So, for result retrieval, there are two architectural choices (see figure 8.12),

- Poll
- Accept,

which represent different uses of a common caching strategy as an intermediary store.

8.8.4 TIMESCALES IN COOPERATIVE CHAINS AND PIPES

Every system's behaviour is constrained by its intrinsic timescales. Figure 8.12 defines two timescales:

- If users make explicit requests, from downstream to the source, then updates are requested on demand (which defines a timescale of user interest).
- Alternatively, the source events might drive processing updates on their schedule, imposing them without forewarning (which defines another timescale).

Push from upstream and pull from downstream (advanced and retarded boundary conditions) represent different agencies, with different intended schedules. So there are already multiple timescales on our radar:

- ΔE, the interval between events and data changes at source.
- ΔU, the interval between user requests.

and in between there are many more characteristic timescales that relate to the processing of different stages and dependencies. These timescales get distorted by processing for many reasons. The most common reason is the time it takes to fetch or compute a result from the data.

In all pipelines, there are two sets of 'users' for the pipeline (input suppliers and output consumers), and processing may be driven by actions from both ends (see figure8.12) e.g. in a build pipeline, commits might be batched for compilation once per day, but that could be accelerated if some user requests a latest image. Some examples of how these two models look in terms of technology implementation are shown in the table below.

CASE	SAMPLING PROCESS	CACHE RESULT	QUERY
SENSOR IOT	POLL SENSOR/GW	DB	(NO)SQL
CD BUILD	COMMIT CHANGES	DOCKER/HELM	START INSTANCE
ML TRAINING	TRAINING PROCESS PUSH	ANN/DL PIPE	MATCH QUERY

8.8.5 ARRIVAL PROCESSES AND EVENTS

Using a minimal amount of promise theory[BB14a] we can state and describe specifications of the system, in terms of a collection of agents that make certain promises to one another, in order to apply basic principles of distributed information and understand scaling, and verify their completeness.

Let A_i be a collection of agents capable of data processing. We shall sometimes refer to agents by the roles S_i and R_i for sending and receiving messages respectively. Subscripts i refer to different agents or system components. Subscripts n will refer to different samples or messages, at different times.

To characterize the arrival of data, from a set of sources S_i, we imagine that a number agents with independent intent feed data to one or more receivers. There are two ways this can happen. The first is by imposition:

$$S_i \xrightarrow{+D_i}\!\!\blacksquare\; R_i, \tag{8.22}$$

or 'push' of data D_i from the source, leading to unpredictable (involuntary) arrivals. The second, alternative approach, is for an agreed delivery time to be promised:

$$S_i \xrightarrow{+D_i} R_i, \tag{8.23}$$

according to a promised schedule.

8.8.6 TIME MEASUREMENT AND SO-CALLED 'REALTIME'

Before we can begin to talk about 'realtime' response, we have to be clear about whose time we are talking about. A distributed system is full of clocks that tell their own time.

System time is marked by changes of state, in the system state machine. The pipeline model described thus far leads to three distinct definitions of system time:

- ΔE_i or the event arrival interval from source S_i. In an 'interrupt' model, a single event $E^{(i)_n}$ is a tick of the receiver's clock, i.e. an entry in its log, and there the time between events is always 1 because each event is the tick itself. If the receiver has sampling (polling) semantics, then we can define it to sample at each interval Δt_i according to some independent clock.

- Δt_i or source sampling interval for events $E^{(i)_n}$ from source S_i, according to an independent interior clock of the receiver (e.g. input buffer for channel i). By Nyquist's law, $\Delta t_i \leq \Delta E^{(i)}/2$ in order to capture the events.

- ΔT is the aggregation interval, or the time between samples of the input channels. If there are N stream channels, then by Nyquist's law, $\Delta T \leq \Delta t_i/2N$ in the

ideal case of no loss, where all channels are sampling at the same rate and are approximately balanced. If events are sparse, according to T's clock, then it can get away with sampling at a slower rate to resolve arrivals.

The processing rates of the agents sending and receiving data need to scale according to their ability to respond. To avoid data loss, Nyquist's theorem tells us that we we need to ensure approximately:

$$\Delta T \ll \Delta t_i \ll \Delta E_i, \tag{8.24}$$

for the pipeline's relative timescales.

When we talk about 'realtime', we expect to be talking about t_i, but we can only describe T, so we have to aware of the latency ΔT_i on each of the channels, which might vary with T in a non-linear way. So the best notion of 'realtime' processing we can give is

Definition 176 (Realtime pipeline)**.** *A pipeline in which timescales satisfying the inequality (8.24) and with bounded latency of the order*

$$\frac{|B|}{\mu_T} \sim 1, \tag{8.25}$$

where $|B|$ is the data size of an average batch B, and μ_T is the processing rate in data per tick of clock-time T.

Note, this tells us something about the overhead incurred by not batching data. Suppose there is an approximately constant latency per batch. if the batch is small, we need to collect more and this there is greater percentage of overhead[139]. If the batches are large, the overhead is small, but the data may lag behind what we expect might be 'now' at the sensory end of the dataverse. At some critical batch size, the cost of data collection crosses a threshold at which there is no point reducing batch sizes because the overhead limits the rate at which they can be collected. So latency or overhead are the fundamental limitation of 'realtime'. Latency starts with sensors, and is added to by network latency, and collection processing latency. The former is not under our control, the latter can be managed to some extent.

8.8.7 DYNAMICAL SCALING: CONVOLUTION OR CONFLUENCE OF STREAMS

Systems scale vertically and horizontally. We can't assume that the intervals between arrivals are balanced across parallel channels in their average properties, or congruent

in their general pattern of arrivals. Using the timescales above, we can define an arrival process as a sampling, polling, or collection buffer. We assume a set of finite messages $D_1, D_2, \ldots$, which are what we call 'data'. These may be considered independent, in the sense that they make independent promises, describe independent facts, or have otherwise independent intent, so that they can be processed without dependency on any other messages.

Data transmission involves the binding of two dual processes: arrival and sampling. The data arrive as events $E_1, E_2, \ldots$, etc, which collectively constitute an arrival process[GS01]:

Definition 177 (Arrival process)**.** *Let D_n be a finite, independent data message, imposed from a sender S onto a receiver agent R:*

$$S \xrightarrow{+D_n} \blacksquare\, R, \tag{8.26}$$

A sampled random variable that forms a distribution of interarrival times $\Delta t_1, \Delta t_2, \Delta t_n \ldots$, over jobs or messages M_n, where Δt_n are measured in ticks Δt of the interior clock of agent A.

and need to be sampled or observed and accepted:

Definition 178 (Sampling process)**.** *We assume that the receiver agent promises to receive and accept the job requests:*

$$R \xrightarrow{-D} S. \tag{8.27}$$

If this promise is persistent, and is kept at a regular interval Δt, it may be called a sampling process, with Nyquist sampling rate $1/\Delta t$.

Arrival and sampling processes must coexist for there to be data transfer.

8.8.8 JOBS AND BATCH JOBS

Let $J()$ be a job function that computes a result from data D. A (primitive) job can be defined as single event, i.e. given a sampled arrival D_n:

Definition 179 (Job). *A job is an execution of the transformation function $J(D)$ that depends on the arrival of D_n*

$$A \xrightarrow{+J(D_n)|D_n} A', \tag{8.28}$$

where we use the standard notation for conditional probability and promises $|D_n$ to indicate dependency. The processing time to absorb the data D_n may be written $T_P(J, D_n)$ ticks of R's clock. In queueing theory language, this relates to the average service rate $\mu \sim 1/T_P$.

By the conditional promise law, this conditional promise can only be realized in the presence of a counter-promise to accept the data from its source:

$$A \xrightarrow{-D_n} S, \tag{8.29}$$

The scaling of the throughput follows the Universal Scaling Law (an extension of Amdahl's law for queues)[Gun93, Gun08].

8.8.9 IMPOSITION AND PROMISE SEMANTICS

If data sources can promise data in a predictable way, this allows delivery promises to be kept and for certainty (predictability as a service) to propagate along the chain. When data are imposed at ad hoc intervals, they do not arrive with a predictable schedule, and we cannot make any promises about delivery.

Lemma 42 (Imposed arrivals). *If data sources make no promise about data delivery, the receiver cannot promise when it can complete its job, i.e. the ability to make a conditional promise*

$$R \xrightarrow{+J(D)|D} A \tag{8.30}$$

without a promise about the condition is impossible:

The proof follows by the conditional promise law[BB14a], since the receiver cannot promise when it will receive or accept delivery, and cannot promise the results itself:

$$\text{Missing:} \quad R \xrightarrow{-D_n} S \tag{8.31}$$

$$\text{Missing:} \quad R \xrightarrow{+D_n} S \tag{8.32}$$

and thus cannot quench the dependency in (8.30) in any way[140].

Push driven systems are sometimes associated with *reactive* or *event driven* systems—though this can be misleading (see section 6.7.1). Pull is the fundamental sampling; it requires active polling of the queue. It optimizes message transfer according to the downstream capabilities.

8.8.10 SEMANTIC SCALING OF DATA: BATCH AGGREGATION

Dynamically aggregations always involve coarse time graining, but time is not the only variable by which to aggregate data. Specific semantic queries, based on intent, may use any key or label attached to data in a stream to sort and collect. The formation of batches is therefore a semantic issue, not only a dynamical one.

Definition 180 (Data frame)**.** *A data 'point' may actually be a vector, matrix, or general tensor of values, acting in the role of a single event, whose schema has a particular* type *label. (Something like a 'particle' in physics.)*

Definition 181 (Distributed data frame)**.** *A data frame that is physically represented across a distributed collection of nodes with private memory.*

The scaling of data from a single point or event arrival, into to a stream is key to the end to end delivery of data through the pipeline chain. The atomic unit of data is an event D_n.

Definition 182 (Batch of data)**.** *A finite collection of data messages (events), identified by 'name', i.e. some unique label, e.g.*

$$B = \{D_3, D_6, D_9\}. \tag{8.33}$$

Traditionally, the term 'batch' has conjured the idea of a large amount of data that takes many hours to process, but the size of a batch need not be limited to large sets. The size of a set is a variable that can be optimized on any number of criteria. Micro-batches are common in streaming engines because they bring additional simplicity and a reduction of overhead, or because they help to avoid out-of-order delivery.

We could plausibly define a batches as aggregations based on different criteria, e.g. the result of a SQL query set, or a collection of arbitrary events up to a minimum length, in general computations need the specific promises made by the data. For example, it would not be appropriate to muddle data semantics together in the same stream, e.g. accept climate data in order to draw conclusions about consumer shopping habits. The names of data packets, in the broad sense, label what they promise, and these refer to separate channels. Multiple channels may share common *dynamical* resources, but maintain *semantic* distinctions.

Definition 183 (Batch Job)**.** *A batch job is an execution of the transformation function $J(B)$, on a data batch B:*

$$S \xrightarrow{+J(B)|B} A = S \xrightarrow{+J(D_1,D_2,\ldots)|D_1,D_2,\ldots} \tag{8.34}$$

A batch job may or may not be sensitive to the interior structure of B. In other words, the function $J()$ might operate only on the aggregate of the data, or it may need to preserve the interior order and structure of the data in some way to compute its result. A very simple batch job is to aggregate data according to a particular label, e.g. by name, by location, etc, and return a representative value such as an average.

The criteria for collecting data points together into a batch can be based on any semantic label, used to distinguish them. Note that a batch job embodies the intent to use a specific set of messages, which in turn assumes that the set can be identified uniquely.

Some functions, such as averages, may be implementable and incrementally updatable, on any size of data batch. One might be able to show that their asymptotic behaviour will not be dependent on the order of arrivals. This is not true of all functions, however (see section 8.8.12).

8.8.11 SCALED FORWARDING, BY WINDOW AND BATCH

Once data have been aggregated according to some semantic label, we need to ask: when is a sufficient amount of processing complete, in order to trigger the next stage of the pipeline? This is a windowing policy question. Each stage may have a criterion for completeness χ_i, which is equivalent to a batch size $|B|$ at each stage. The minimum batch sizes at each stage may be quite different. This also depends entirely on the nature of the batch functions J_i.

At the source edge of the pipeline, each 'sensor' event triggers a response from the pipeline receivers, but this event-by-event response will not scale effectively (or semantically) over long times or aggregated over many sources. In most cases, an meaningful update to a pipeline result will depend on a minimum amount of new data. Some threshold policy is required to make this ad hoc choice.

Rephrasing the question: each chain is an ordered set of prerequisite dependencies. As we aggregate batches, we need to define what constitutes a tick of the pipeline's next clock. Note, each aggregator needs to be significantly faster than its predecessor's output schedule, but since this is not sustainable, the outputs of aggregated data need to 'save up' or buffer events so that they can be outputted collectively as a single event to the next stage, usually with some *data summarization*.

8.8.12 ORDERING OF ARRIVALS AND BATCHES

If functions are symmetrical in their arguments, there is no particular need for in order delivery; such jobs therefore form a Markov process[GS01]. However, if the order of data matters to computation, because the function is non-linear, then a total ordering (sort) function $O(\cdot)$ may be implied to the aggregation of data. Since this requires buffering, it

limits the rate at which results can be promised.

$$S \xrightarrow{+J(B)|B} A = S \xrightarrow{+J(O(D_1,D_2,...))|D_1,D_2,...} \tag{8.35}$$

Labels based on 'arrival times' (according to the receiver's clock) cannot necessarily be attributed any significance, since there is no causal relation between arrival time and intent on the part of the sender. The ability to meaningfully order events on a single clock depends on the kind of source represented by the previous stage in a pipeline: from a direct sensory source, to a parallelized processing cluster. The former represents a definite source of causal ordering, while the latter could be shrouded in entropy. Arrival order is not an invariant property, the ballistic approach is the root of uncertainty, along different paths.

- In general it only makes sense to preserve the order of events recorded at a single point in the network. In some cases, it might make sense to arrange for a shared clock service for a number of agents that belong together, and are approximately collocated, and define the order by the shared clock time of the cluster (see figure 8.14).

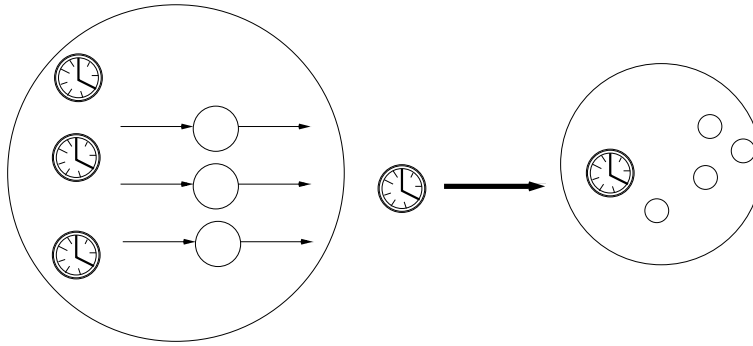


Figure 8.14: Is there a single clock whose time can be called representative for all subprocesses?.

- When data are aggregated into a single cache, such as a database, the order or data may be lost. The arrival times at the database can be used to order them as a single stream. Or, if there are already ordering labels on the data, these can be labelled additionally with the module from which the data are sourced, to keep independent clocks separate.
- The sensitivity of computations to ordering discrepancies and inaccuracies depends entirely on the nature of the transformation function. The order of data

arriving from sources that are meant to be compared congruently, such as transactional changes, weather, and other causal models, will be most expensive to define correctly. In scaling computations, the most resilient computations will be those that converge to a single attractor, without a strong dependence on order.

When searches, either from a cache, or applied directly to remote sensor gateways, are curated into a log or file stream, the streams should be kept separate, as they may have different latencies, clock calibrations, and so on.

Data are effectively engaged in a race from sources to collectors. In a parallel cluster, slow nodes might lead to an effective reordering of arrivals, and behave like late data. So chaining along different paths can distort apparent causation. This is why we need desired end state with conditional dependencies. One could try to measure the relative ordering of events at the input, and preserve this throughout a distributed computation. However, this is not a trivial matter, and inevitably leads to queuing and latencies, which could themselves become unstable. Approximations based on best-effort synchronization of clocks are often used. This is only meaningful if the 'real time' data source is localized in space.

8.8.13 SCALING OF ORDER AND CHAOS

Scaling theory (coarse graining) tells us that scaled macroscopic outcomes should not be sensitive to the precise spacetime properties (including the order) of events at the microscopic level. An exception may have to be conceded for highly non-linear computations where sensitivity to small variations does not average out and decouple, indeed they may be amplified. However, in that case, we have an intrinsic instability (deterministic chaos), and only a policy choice can resolve a definition of correctness[LLG$^+$09, LGZ$^+$14, ABC$^+$15], by trading off sensitivity against speed, accuracy, with no guarantee of certainty.

8.8.14 WINDOWS AND BATCHES

A discretization of time is often referred to as a window. Windows can be defined according to a variety of semantics, e.g. they may have a fixed schedule (calibrated to a reference clock) that covers the entire timeline in batches, or they can slide along as new events arrive, so as to maintain a fixed amount of data that always includes the latest.

Definition 184 (Window process)**.** *A collection of data events, ordered according to their arrival in a single queue Q (see fig 8.15), beginning at some*

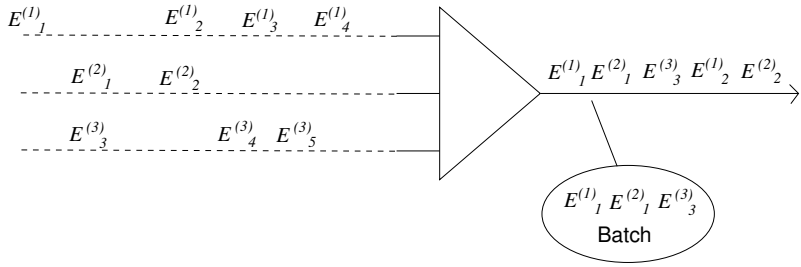


Figure 8.15: The aggregation of events into a stream may have ambiguous semantics, especially when there is aggregation from parallel sources. There are three distinct versions of time on the inputs, and a aggregated time stream at the output. The windowing concept has to take care of this, especially if the stream promises 'ordering correctness'. A queue cannot have simultaneous events, but it can label consecutive events with a coarse grained time. Some number of events from the aggregate stream constitutes a 'batch' to be processed as a unit.

8.8.15 COMPLETENESS

Does the receiver of the data have a complete set of its dependency requirements to propagate a result? If a result can be computed for every arrival, then every input event can propagate independently through the pipeline. This might be expensive however, e.g. in the case of a blockchain transactions are deliberately accumulated to a minimum window size (called a 'block') before computing the resulting hash, because it would be prohibitively expensive to recompute for only a single transaction. See figure 8.16 for some alternatives to defining windows.

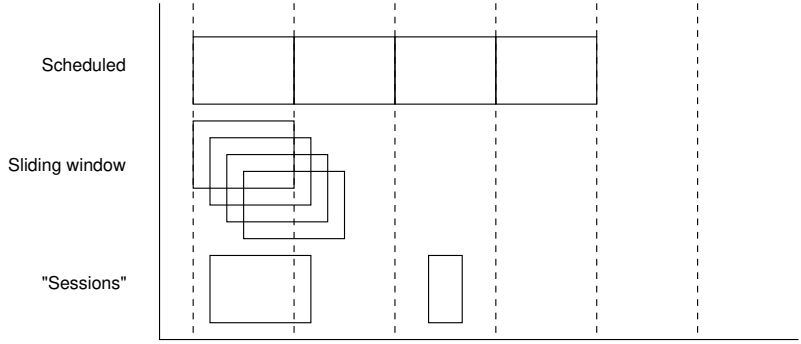


Figure 8.16: The framing of data batches or windows in sampler time.

8.8.16 "CORRECTNESS"

Correctness is a matter for interpretation. Like responsibility, it is up to the recipient of a result to assess correctness, as it has no invariant meaning. Some papers seem to assume correctness tacitly means order-preserving. Completeness, accuracy, order preservation, absence or presence of features, could all be considered measures of correctness.

Each data arrival constitutes a new versionable change. The batch/windowing policy determines which are treated as separate versions, and which are part of a bulk update. Pipelines therefore promise 'versioned' data results, based on versioned input and dependency-versioned processing, through a number of stages (see figure 8.13). If there are lots of data, we may need parallelism to divide and conquer, e.g. like Map-Reduce, so the stages may be multiple sub-clusters, like a chain of scaled services.

8.9 SMART HUMAN SPACES

We've spent a lot of time focusing on principles and analyses, using examples from the world of technology—yet, some of the most enduring and important systems of our age lie in our communities, from buildings to workplaces and cities. These are the very fabric on which society rests. In recent years, the concept of 'smart buildings', 'smart cities', and even 'smart homes' have arrived, with an emphasis on employing data driven features to enhance traditional services. Smart building designers (architects) have come somewhat further, utilizing new techniques for energy saving by integrating human and mechanistic processes under a common aegis. We can generalize all of these under the banner of 'smart spaces', and even as examples of semantic spacetime.

Smart spaces can, in principle, be imagined at any scale, from microscopic materials, smart molecules and disease technology, to urban spaces and even entire global economies. The challenge to extend 'smart design' is to adapt equip them with the ability to promise time-varying adaptation. There are two visions one might imagine for 'smart spaces' :

- To enhance the experience of people living in or around smart spaces.
- To create autonomous, specialized functional spaces (like factories, farms, or organisms) that are more self-sufficient.

In either case, the meta-goal is to bring a positive benefit to society at large. However, we have a choice about what this means. It could mean a community of smart individual units (putting individuals first), or it could mean a smart superagent, i.e. a scaled singular unit (putting the city first). The preference for either of these is often cultural. In 'Western civilization', individual benefit is the overriding focus, whereas in 'Eastern civilization'

the collective good is often the focus. There are many complex issues at play. The most important consideration is the *scale* at which systems interact with one another. Both viewpoints have a place, but can they be aligned simultaneously?

8.9.1 WHAT IS SMART?

Smart is a subjective assessment, of course. We give it, based on our perspective, to things and people we feel bring us value or save us some cost, in a timely manner (see figure 8.17). What does smart mean? Each individual observer decides whether they experience something as smart, thus the scale at which someone observes the world matters to their assessment.

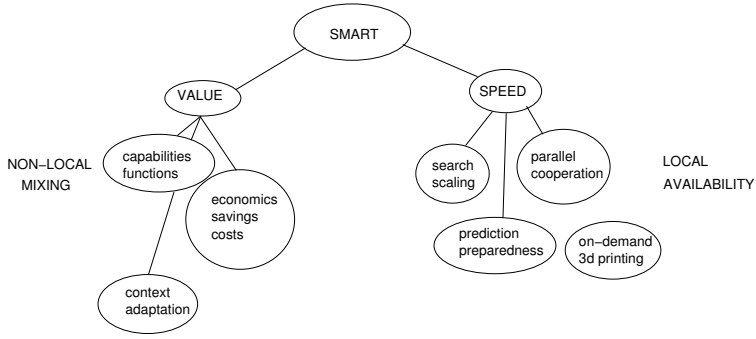


Figure 8.17: What is smart? A mixture of timeliness and fitness for purpose.

We perceive things or processes as helpful only relative to a context, i.e. are they fit for *present* purpose? The use of context implies that *sensing* of context is involved, and some feedback, influence or decision that is based on the result. A thing or process acts arrives 'just in time' to be useful in that same context (too late is not smart). Smart may also mean efficient, to the point, i.e. concise, 'not too much noise'. The final point concerns diversity. Diversity can offer different contexts for self-service adaptation. In a community or ecosystem, multiple competing agencies can evolve, and not all their functions and services will be assessed as fit for every purpose. As environments fill up with competing offers, too much competition could bring contention and become more of a burden than a help.

So, what makes a space or a city smart? From a Promise Theory view of the world, we always ask:

1. What or who are the operational agents involved?

2. What promises do they make to one another, in what context?

3. What are the important scales for their interactions?

Scale always plays a key role in the understanding of systems. Traditionally we think of cities and geographical communities, but with modern communications, this is a less obviously useful definition. When a city's resident moves beyond its geographical domain, does he or she leave it behind, or carry it with them? In other words should we say that a city is a sum of agents (fixed or mobile) or of places? For mobility, the concept of self-service infrastructure (see section 6.8.2) is a key theme in enabling agents within a space to be autonomous, to save time and eliminate dependencies that slow down 'Just In Time' experience. This is both more direct (less distortion of intent) and quicker. There also has to be a balance between focus and variety. Intent is focused. If there is too much variation (noise) it washes out the ability to focus on a specific intent. This is related to the issue of modularity.

8.9.2 SMART SPACES AS SYSTEMS

There is a lot of evidence that what happens in cities and spaces has universal characteristics at different scales, and may even be described in terms of knowledge based patterns[ea77]. Analysis begins—as always—by asking: what are the significant processes for functional behaviour in a space? What agents (people, things) are in play, in the space we are studying, at the scales we care about? What promises are made by each agent? How do the spaces affect or arrange for economies of scale? How does the space, as a whole, support innovation? What are the characteristic timescales, the dependencies, and network relationships allow them to happen? Transportation? Communication? How is knowledge represented in the space? What can the space learn about itself and about its surroundings? Where do we consider its physical and semantic boundaries to lies? The usual hypothesis is that smart spaces are a result of good use of information and knowledge. Knowledge can be stored in several ways and at all scales, from the relative positions of buildings where people to collaborate, to the patterns of wear in a park lawn, to the writing in books[141] Smartness may lie in hardware or software (and their ability to address timescales), in design, in persistent structures (like buildings), or in persistent culture.

Example 226 (Smart space agents and scales)**.** *Agents, which make promises, are functional entities that can operate at any scale. All agents are proxies for human intent, but the human intentions may be represented and enacted,* in situ*, by arbitrary* THINGS*: People, Machinery, Systems and processes, Buildings and structures, Transport systems, Animals, and so on. Each of these acts relative to its environmental* interaction scale*.*

Example 227 (Learning versus canned solutions)**.** *To make a space smarter, with the aid of technology, we need to ask: how do we embed a technology, at the right place, at the spacetime scales of the processes that can bring value 'just in time'? i.e. Location, timescale, function (cached/canned/productized decisions).*

Regardless of whether we want to think of a coherent space as an organism or a community, a space is a theatre in which activity takes place (e.g. a city is where human activity takes place); it is a catalyst for activity, and a framework for human purpose. Only when we step aside, as observers, does the occupied space become an organism for independent activity. We can imagine such independent spaces within the boundaries of a city too, e.g. factories, wildlife preserves, etc. The boundary of a system is not necessarily the same thing as the boundary of the region it occupies[142]. What's common to all of these cases is the idea that each space fulfills functional roles. A good place to start is to look at what emergent processes and functional roles occur in cities (and their smaller spaces).

8.9.3 PURPOSE OF A SPACE

No space, home, or city is an island, nor does it exist in a bubble. External, environmental forces shape cities and other spaces as much as what happens within them. Specialized curated spaces bring together agents as systems, in functional ways, for combinatoric effect—relative to their environment and context. Any large space forms a mosaic of subcultures, which develop their own languages, jargons, habits, etc. The size of these is shaped by the Dunbar hierarchies[Dun96, ZSHD04]. Why do communities form? Reasons to cluster together, in networks, begin with self-protection (safety in numbers) but extend to innovation and trade:

- Shelter, security, protection, minimize contact or risk perimeter[143].
- Supply and demand of traded goods and services with outside.
- Economics of resource sharing and specialization (optimization)[144].
- Stability: mixing and redundancy and robustness.
- Mixing: putting agents together as a marketplace for trade (including ideas)[145] .

Resources may come from outside a space (e.g. the region around a city), and waste is dumped outside it. The ability to grow and streamline operations on demand (from within and without) is important to any organism or community. This requires *functional scalability*, i.e. the freedom to change the size of a system without altering its function. It is sometimes assumed that shared resources must be centralized, or made into 'silos'

(which are convenient for productizing and branding identity), but smarter systems based on distribution and Just In Time delivery may do a better job. We should also note that centrality is scale-dependent. That which is central at one local scale may be decentralized at a wider scale (see figure 8.18).

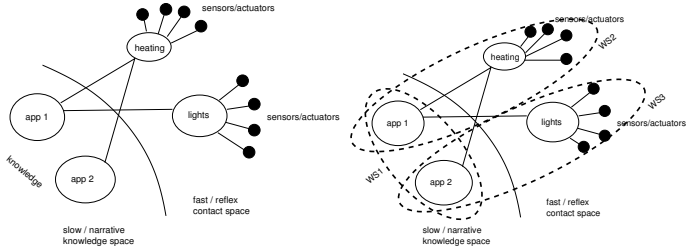


Figure 8.18: Applications may live close to the edge, or in a more central place, to fit with the fast reflex responses, or the slow knowledge based coordination. The right hand side shows separate workspaces overlaid, with applications, sensors, and physical regions integrated inside a virtual perimeter.

8.9.4 DISCOVERY AND CONNECTIVITY

Parts of a space may benefit from being close together. How agents find one another is the basis for how they can collaborate. There is both static design, and there is mobile collaboration. If a part of town is the nightlife/shopping, etc, it benefits from being clustered into only a few districts. This is a static longterm planning decision. Food trucks, circuses, raves, festivals, etc, can coordinate and organize for mutual benefit. Information technology has a clear role to play in exposing such services[146].

Example 228 (Virtual proximity, constraint or autonomy?)**.** *If we are tied to a specific location, we are dependent on agents perhaps unnecessarily. Virtualization of services frees agents to interact without being tied to a location. Applications and services are increasingly available to us on our personal electronic devices. Although we expect the nature of the interaction to evolve over time, this tele-presence facility is a great time saver. We take it for granted that these agents will be enhanced by software, but exploiting information can be done in many ways—even by being in the right place at the right time. Communities and sub-communities form virtual boundaries within a city, and districts, e.g. Chinatown, shopping districts, suburban communities, clubbing areas, etc.*

Example 229 (Smart taxation)**.** *Many taxation and personal health issues are connected*

to our place of residence. What happens when we are no longer within our home geo-political domain? This suggests the need for global digital taxation protocols between countries, supporting micro-payments.

Example 230 (Interaction scale)**.** *At what scale should a city enable interactive learning and resource sharing through communication? Between residents? Between companies? Between cities? All of these scales can benefit, but they are distinct.*

8.9.5 PARTICIPATION: A SENSE OF PURPOSE

Automation makes a society efficient and relieves humans from the burden of emulating machinery. But, as humans, we need to be involved in the stages we care deeply about. We would not want a machine to pick up a new born baby, or prepare a special meal for friends, without direct supervision. It is not possible or practical to program all the individual concerns into a simple instruction set. Participation favours dignity, which favours self-worth and human happiness.

Promises on a societal scale that help to bring about participation are: inclusiveness and transparency in politics and governance; comprehensible laws, rights, and civic governance; public services and help desks and service oriented government.

Citizens may not need physical proximity, but we know well that when there is an empathy barrier (like a windshield or telephone) between us, many misunderstandings and upsets occur.

Example 231 (Transparency or action?)**.** *In China, for example, government is not very transparent, but the users of government can report problems as 'bugs to be fixed', and prompt solutions may be offered. In the West, there is greater transparency but often little response to complaints once a choice of government has been made.*

As we move into a future where the basis for employment and purpose is undermined by our own success at innovation, we need to rethink how our living environments support our emotional needs, not only our physical requirements. Smart cities of the future might be more like smart exhibits, encouraging participation. Cultivating a sense of dignity. We don't need physical proximity, but we know well that when there is an empathy barrier (like a windshield or telephone) between us, many misunderstandings and upsets occur.

Example 232 (Global citizens)**.** *The world is globalized, but taxation is localized. This causes many difficult issues for travellers and for governments. The ability of pay electronically in multiple currencies, and relative to what services we consume could simplify the fair redistribution of wealth considerably. Taxation as a service rather than an obligation glued together by complex treaties.*

Example 233 (Wealth redistribution)**.** *What markets value is not necessarily what society values, so some processes can be independently self-sustaining; other desirable features of society need to depend on the support of high earners. Rechanneling of funds, by policy, is thus a part of a stable collective society. This is the philanthropic purpose of taxation. Smart taxes can also have a stabilizing effect on fickle markets.*

Some issues of smart adaptation can be carried with us as individuals, no matter where we might be, while other issues are tied to a location. Systems and processes take advantage of what they can when they can.

8.10 POWER CONSUMPTION

No system can exist in the physical universe without exchanging energy amongst its components and expending work. In spite of the theoretical possibility of so-called 'reversible' machinery that—in principle—doesn't dissipate energy, the reality is that not all the activity within a system can be harnessed for a functional purpose. That said, some technologies (including Information Technology) operate quite brazenly on the ability to dump electric current to ground, where it ends up as waste heat.

A direct analogy is the way manufacturing relies on disposable materials, like plastics and paper packaging. When a system is designed to consign part of its energy and material to the garbage dump, there is a potentially missed opportunity to recycle that loss. Loss is a thermodynamic reality, but it can be handled well or poorly.

For so-called RC networks, or resistive-capacitive systems, which dominate electronics, power can be calculated approximately using the basic proportionalities. Where computers and human systems exchange information, electronic systems work by exchanging electric charge Q. This is related to the capacitance C, or storage capacity, and the voltage V impetus V, by:

$$Q = CV \tag{8.36}$$

Current is a flow of charge:

$$I = \frac{dQ}{dt} = C\frac{dV}{dt}, \tag{8.37}$$

and voltage and current maintain a steady state through resistive materials in the relation,

$$V = IR, \tag{8.38}$$

which is known as Ohm's law. Voltage is the source impetus that drives change in a circuit, and its time dependence can be represented by a Fourier superposition of frequencies:

$$V(t) = \int df\ V(f) \exp(2\pi i f t). \tag{8.39}$$

For a single frequency (an unusual, but simple case), this is just

$$V_f(t) = V_0 \sin(2\pi f t). \tag{8.40}$$

Finally, the power (which is energy dissipation per unit time) is given by:

$$W = IV = I^2 R \sim f^2 R C^2 V^2 \tag{8.41}$$

$$\propto f^2 \tag{8.42}$$

There is not single frequency, but a distribution of different frequencies. The larger the content of high frequency changes, the more energy is converted into heat, through resistive heating (friction). Now, information technology works by switching energy stored in capacitances from 0 to 1 at the frequency determined by the CPU's operating frequency. Today, this switching rate is of the order of Giga-Hertz or 10^9 switches per second. What this means is that, when information technology is computing something, the power consumption is approximately proportional to the spectrum of frequencies at which information changes.

Example 234 (Power consumption in a human)**.** *Being alive involves a heartbeat, and a minimum rate of heartbeats ticking over. So our bodily functions expend energy on a scale of hundreds of Watts, and we have to feed ourselves to maintain this. The faster our hearts beat, the more energy we convert into heat, and the more energy we have to consume. The eat we produce heats up our houses and workplaces, and is ultimately lost to the atmosphere.*

Example 235 (Power consumption in a datacentre)**.** *In a rack of computers, power consumption starts at a basic and constant level just by switching on the computers—having them tick over like an engine or heartbeat. This is quite a high cost, and any additional processes tend to be small corrections to that initial cost Each time we start a process that alters memory, we generate heat that grows as the square of the frequency of change. All running programs therefore generate heat. The more computers doing this in parallel, the more heat grows in proportion to the number of processes. Multi-core computers have the capacity to generate heat faster, and thus consume and waste energy faster. Modern datacentres can power off or suspend computers to reduce the baseline cost drastically.*

From the relations above, we can see how to reduce energy wasted by processes. If we don't change stuff or data configurations in storage $C = 0$, then there is no need for a current or a workflow. If we reduce the speed of change f, we can save a lot. This is why many CPUs allow frequency scaling today in low power modes. But these reductions are not easy. Even passing data through a network costs energy because the data don't just

flow freely like waves through the atmosphere, they pass through switches and amplifiers all of which involve temporary storage or dissipative switching. Keeping or assessing any promise,

$$S \xrightarrow{\pm b} R, \tag{8.43}$$

involves energy too, due to the sampling of the outcome state, to know whether or not the promise has been kept. The faster this sampling rate (or Nyquist frequency), the more energy grows as the square of the frequency.

Example 236 (Radical system redesign)**.** *There are many ways in which one could imagine saving power in computer infrastructure. Avoiding data transportation by using computation that's close to the source of the data, is one way. This is so-called 'edge computing'. Using optical technologies that avoid resistance $R = 0$ is another. Optical switching and computation is a technology that is developing slowly but surely. Replacing all present computer designs with optical computers in the future might (or might not) decrease the wasted heat. This is hard to judge, as we know that energy has to be dissipated when work is done—unless it can be stored in very large reservoirs, which are potentially expensive and impractical. This is one reason why we dump waste into oceans and into landfill—these are reservoirs that are cheaply available. The problem is that they are not isolated, so there are side effects.*

Today, most heat generated as a byproduct of activity, especially computing, is simply wasted. In the most advanced factories and datacentres, waste heat may be captured by liquid pipes designed to store the heat and use it for something else. There is no difference, in principle, between a power station that captures heat energy from coal or oil, and pipes collecting heat generated by second order processes in factories and datacentres—only the willingness to handle the waste responsibly. None of our mobile phones, television sets, cookers, or coffee machines have any such ability to capture and use their wasted heat energy. The initial cost of doing so is generally assumed to be higher than what businesses might recapture by doing so. In other words, the financial incentives for efficiency are low. This contributes, amongst other things, to the Information Technology industry as being one of the largest energy consumers in the world—exceeding even the transport sector (air travel, shipping, etc). Awareness of this energy cost is not widespread, and the may layers of virtualization we rely on the create the magic of technology makes it difficult to appreciate for end users.

Efficiencies of scale are possible, as in the discussions in earlier chapters, but these are rather complex. Moreover, as technologies become more efficient (and thus cheaper) it often drives greater usage, which counteracts the savings. This is sometimes called Jevon's paradox.

Example 237 (The utilization argument)**.** *One of the arguments for cloud computing is that by packing as many processes into computers as possible, so that none are idling, none of the basic cost of ticking over is wasted. This leads to a reduction is waste. This is only an argument for greater efficiency, not for less waste. A similar argument is used by airlines and shipping, for packing planes and ships as full as possible to extract maximum use from the unavoidable cost of waste.*

The difference between making the most of unavoidable waste and reducing energy waste altogether is a difficult one, partly because human activity is proportional to the growing population as well as to our living standard and growth of the economy. The modern capitalist economy is basically a pyramid scheme that needs constant growth to make sense, so our very raison d'etre is to consume more and more energy. This systemic issue, above all others, is one that needs serious attention in the coming years.

Centralization enables economies of scale, in principle, only by pointing to a Single Point of Action (remedy for a Single Point of Failure, if you like), so that—if savings can be made for the central provider—then these can be made quickly available to all users. It's the $O(1)$ versus $O(N)$ network effect in action again. But it only works if the central provider of key services promises continuous improvement and optimization—not merely passing on its shared costs to clients.

CHAPTER 9

FAULTS, ERRORS, AND FLAWS

Having spent some time on the dynamical aspects of processes, in the promise language (which doesn't intentionally suppress their semantics), it's time to turn to process semantics properly. In this chapter, I want to take the intended purpose of system components more seriously. Not all intended purposes are designed by a thoughtful creator—some purposes evolve opportunistically—yet, we can speak of purpose, nonetheless, as long as we can speak of the alignment of what is offered (+) with what is used (-), then we can speak of intentionality and purpose in an impartial manner.

The three F's of anomalous behaviour are one of the main reasons we study systems. Our (often unspoken) intent is to ensure that systems keep certain promises, so that they behave predictably, we can trust them, and use their behaviours as a part of a composition in the scope of a larger context. If promises are known and stable, it should be possible to define and diagnose faults when they occur. The literature of fault diagnosis is broad and varied yet little of it addresses faults formally in terms of specifications. Fault Tree Analysis, for example, considers only system pathways that are considered to be problematic. This is an extensive analysis pioneered, by the nuclear industry with probability assessments used to rank failure modes.

The nomenclature of problem states and trajectories of a system remains somewhat vague in spite of such a literature. Here, I'll refer to 'the three F's' of systems: faults, errors, and flaws, which are concepts with somewhat different semantics.

9.1 RELIABILITY AND TRUST

Reliability is the quantitative assessment of how well an agent, on some scale, keeps a promise. The tradition of quantitative science compels us to use numbers where we can,

but we don't always have data on which to base estimates of reliabilty. Promise Theory tells us that semantics will also have a role to play in assessment. Trust is a related concept, which is combines semantics (or qualitative judegements) with dynamical (or quantitative judgements). In the absence of data, humans generally offer a default belief about the trustworthiness of another agent. This is steeped in psychology—people with shifty eyes feel untrusrtworthy, we might trust beautiful people, etc. As time an experience goes by, successive interactions (of the (non-)cooperative game [Axe97, Axe84] adjust that default belief up or down to form a kind of Bayesian probability of 'learned' experience about promise keeping. Rumour, and other public discourse, is another mechanism by which we update our assessments, quantitatively or qualitatively. A single piece of evidence that is semantically important can wipe out our trust in an agent or process. . The relationship between trust, evidence, probability, and experience is more complicated than elementary probability theory or statistics would suggest. This means that quantitative measures of traditional reliability are of limited value in assessing systems, across the spectrum of possible failures. We return to this in the next chapter.

Non-human agents may also effectively exhibit a kind of trust, for example—in adjustments to their sampling rates. When observed promise keeping seems uncertain, an agent might increase its sampling to monitor more closely, or reduce when no change occurs for a long period of time. It would be wrong to think of trust and reliabilty as entirely human or machine judgements. As always, the value of Promise Theory lies in its ability to put a common umbrella over all kinds of agent—to focus on what matters about processes without prejudice.

Example 238 (Security terminology)**.** *In the field of computer security, the term trust is used with a confusing meaning. If you trust something or someone, it's a sign that you are willing to accept it without testing it first. In security, trust is used to mean a credential acquired by testing, i.e. a confirmation that you have already tested or validated the promise, and that you assume it will therefore always be kept. This leads to several confusions: first, it rejects the idea that trust is a positive cost saver, built on relationships; second, it perpetuates a myth that trust is binary—that, once established, trust will not be broken, leading to a false sense of security.*

9.2 DRIFTING OUT OF PROMISED ALIGNMENT

The concept of drifting into failure has been popularized by Sidney Dekker [Dek11]. He makes a simple point, which we can rephrase in terms of promises. Agents, who make promises, may eventually come to alter their behaviour:

- An agent, on one side of a promise binding, might change its promise without

informing the other explicitly.

- The fidelity of the agent, for keeping its promise, might waver.

In either case, a promise binding may—over some timescale—become compromised. If we start with two agents, S and R:

$$S \xrightarrow{+b_S} R \tag{9.1}$$

$$R \xrightarrow{-b_R} S \tag{9.2}$$

then we know that the effective transfer is $b_S \cap b_R \geq \emptyset$. If either b_S or b_R drift away from alignment, then the overlap between them may tend to the empty set, until finally there is no cooperative binding at all. This is not a fault, per se, in either agent, but a fault in their alignment—in the binding.

9.3 SEMANTICS OF SYSTEM ANOMALIES

In terms of the formalized view of promises, we can ascribe reasonable meanings to the terms, under the general heading of anomalies.

9.3.1 ERRORS (OF EXECUTION)

An error is a term that applies to processes or actions, not to outcomes.

Definition 185 (Error (of execution))**.** *An* action *or change whose outcome is assessed not to comply with the promises it makes, by some agent.*

There must be a promise, as a precondition, in order for there to be an error. The promise defines an desired outcome, and plants a measuring stick for the outcome of the promise to be assessed by some observer. An error of execution may or may not lead to a fault condition, depending on the resilience of the system and the degree of detail to which it is inspected by assessing agencies.

The theory of measurement errors[Top72, Die02] distinguishes between two types of error:

- *Random errors*: Unpredictable variations in the keeping of measurement promises.

- *Systemic errors*: predictable biases distorting the keeping of promises. These are often related to miscalibration of observation, relativity and misalignment of understanding, by an observer, therefore we cannot deal with these in an impartial way.

It is known that human-related variations are often Gaussian, or normally distributed, but other system influences lead to variations that are often asymmetrically distributed [BHRS01].

9.3.2 AGENT ACCURACY OR FIDELITY

From the notion of an error, we can define a general term for the reliability of an agent, in carrying out actions related to promise keeping.

Definition 186 (Agent fidelity (accuracy))**.** *The degree to which an agent performs actions predictably, and in keeping with promises, i.e. without error.*

In this form, it no longer matters whether the role is played by a human of a machine: an agent (whether human or machine) that makes frequent errors may be said to act with *low fidelity*, as it is not able to keep its promises, even though it intends to. Moreover, by grounding the assessment with a specific promise, we help to eliminate room for speculation about intended outcomes.

9.3.3 FLAWS OF DESIGN (FITNESS FOR PURPOSE)

We must also consider the notion that a promise itself was ill conceived, i.e. it was assessed to be of low value, relative to the goals of the system.

Definition 187 (Design flaw)**.** *A promise that is assessed to be inappropriate for the intended outcome, mistaken in its aims, or misaligned with collective goals, by some agent.*

We sometimes talk about a system being 'fit for purpose'. This does not necessarily mean that its intent was misplaced, but that the model of promises was incomplete or insufficient to cover the dominant expectations and eventualities.

Comment 17 (Expecting too much)**.** *With the exception of errors of execution, which we may ascribe to low fidelity of component agents, errors occur often for the simple reason that our expectations exceed the quality of our intentions. In other words, we tend only to form appropriately detailed expectations about what 'should have been intended' only after the horse has bolted (*post hoc*), when it is too late. Thus we can still learn from failure.*

As systems operate, in unpredictable environments, the number of possible states they can occupy grows, by interaction with the environment, to areas where agents have not made promises. Thus agents find themselves without guidance. In other words, systems are incompletely specified, or work on incomplete information.

9.3.4 FAULTS (ANOMALOUS STATES)

Faults are *conditions* or *states* within a system that do not match our expectations, or comply with their promise.

Definition 188 (Fault)**.** *A state or condition, in which the system does not match the promised or desired state. This leads to unfulfilled expectations. In other words, a condition in which an assessment (sample) of the system fails to keep its promises.*

A fault may be arrived at by an error of execution, or in the absence of error because of a design flaw. We might never make something truly fault free, because design flaws can always be hard to foresee. There is thus a strong practical need to build systems that are *fault tolerant*, rather than fault free.

Definition 189 (Fault tolerance)**.** *An agent may be called tolerant if it continues to keep its promises, within acceptable tolerances, even though it depends on promises from other agents, that may or may not be kept.*

Note that, once again, without a prescribed set of promises for an intended state, there is nothing to define when a fault has occurred[147].

Example 239 (Buffers for absorbing variation)**.** *Faults propagate when agents that depend on them are affected by them. Buffers are a way for agents to absorb variations. Buffers are used in mechanical systems (fenders, air cushions, etc), in financial systems (cash flow buffers for unexpected expenses), in allowed leeway for systems in motion, and so on.*

The occurrence of system faults is an extensive and involved topic that is the subject of whole texts (see, for instance [Nat98], [HR94], [NRC81] and [SS03], [DH06]). System faults fall into three main categories, by source:

- *Random faults*: unpredictable occurrences or freaks of nature, usually distributed in a random fashion, leading to an incorrect outcome.
- *Emergent faults*: the system exhibits semantics that it was not designed to promise. These usually come about once a system is in contact with an environment, or through intra-networking at scale.
- *Systemic faults*: faults which are repeatedly caused by logical flaws of design, or insufficient specification.

Comment 18 (IEEE standard anomalies)**.** *The IEEE classification of computer software anomalies ([IEE]) includes the following issues: operating system crash, program hang-up, program crash, input problem, output problem, failed required performance, perceived total failure, system error message, service degraded, wrong output, no output. This classification touches on a variety of themes all of which might plague the interaction between users and an operating system. Some of these issues encroach on the area of performance tuning, e.g. service degraded. Performance tuning is certainly related to the issue of availability of network services and thus this is a part of system administration. However performance tuning vis of only peripheral importance compared to the matter of possible complete failure.*

Many of the problems associated with system administration and maintenance can be attributed to input problems (incorrect or inappropriate configuration) and failed performance through loss of resources. Unlike many software situations these are not problems which can be eliminated by re-evaluating individual software components. In system administration the problems are partly social and partly due to the cooperative nature of the many interaction software components. The unpredictability of operating systems is dominated by these issues.

9.3.5 DISCOVERIES (CLASSIFICATION ANOMALIES)

Does this cover everything? No: what if we have not had the foresight to even make a promise, because we weren't expecting an issue? In this case an outcome is merely something that we identify ad hoc, with a lack of understanding or incomplete information about the system. A fault is then a design flaw—an oversight. We could call the observed effects, events, anomalies, surprises, or even discoveries of new phenomena.

Definition 190 (Phenomenon discovery)**.** *An state of a system that was unexpected because no promise was made by the affected agency concerning the observed outcome.*

Surprises, like this, might not be possible to measure directly, since there is no *a priori* basis by which to describe it.

9.3.6 THE USEFULNESS OF THESE DEFINITIONS: THE MATTER OF DESIGN

We have the terms:

- Errors for actions that were not carried out as promised (transition anomalies).
- Faults for states we arrive at that were not as promised (state anomalies).

- Flaws for the suitability of the intention itself (assessment anomalies).
- Discoveries for unexpected occurrences or outcomes that do not match any promise (classification anomalies).

Having rigid definitions of intent makes it easy to follow a formal method, even if the definitions are not perfect. With the definitions above, we can:

- Push all of the ambiguity around the *source* of flaws onto a single idea: the 'design' of a system that reflects its fitness for purpose. This allows for systems that are 'complex', where causation cannot be attributed to any single element or agency.
- Push all of the ambiguity about faults onto the keeping of promises by one or more agents. We do not have to prejudge the reason why a promise was not kept.
- Push all of the ambiguity about error onto the fidelity of the agents within the system. Errors may or may not be considered the 'cause' of faults[148].

This is a highly pragmatic separation of concerns. We focus attention on issues by saying what we mean with promises. With these definitions, we are better equipped to answer these questions:

- What kinds of conditions can arise within a system?
- Are they desirable?
- If yes or no, were they intended, i.e. the results of errors?
- How could undesirable states be prevented or tolerated?

Key to answering this question is another question: how much coverage of a system's behaviour can be documented or promised? In other words, how much confidence do we have in our ability to determine the outcome? Conversely, how much is simply *ad hoc*, or even beyond control?

9.4 INSTABILITY AND THE LIMITS OF PROMISES

Stability is the perhaps the single most important notion in reliability engineering. No matter how simple or complex a system is in its makeup, the key to predicting outcomes is its stability. The ability for a system of any size to represent something consistently depends on its stability.

Definition 191 (Dynamic stability)**.** *The insensitivity of a promised outcome to perturbations from any source.*

Definition 192 (Semantic stability)**.** *The insensitivity of a promised outcome to variance in interpretation from any source.*

Our ability to determine outcomes within a system can only be related to the promises it makes only if it is both sufficiently isolated from external couplings, and it is stable to perturbations through the remaining couplings. There are essentially two cases we have to consider:

- A system is sufficiently stable for promises to be kept most of the time, and may be used in a functional role.
- A system is not sufficiently stable and cannot be assigned any reliable meaning.

In the latter case, making promises seems to be futile, but there is still a role for them: even though promises can't be kept reliably, they define a basis set of outcomes against which to measure and compare the system's behaviour.

Sometimes instability is couched as a 'complexity' issue. In so-called 'complex systems', non-linearities introduced through strong couplings (dependencies) amplify small deviations, causing divergent or unpredictable behaviour. One can try to constrain or protect against this divergent behaviour, but it becomes a competitive game of information and speed[149]. It is unclear whether complexity itself necessarily plays a role in disallowing stable outcomes; however, complex systems are likely to probe states that were not anticipated or planned for, and hence one could not easily expect that promises about them would be kept.

The existence of a documented promise enable any agent, in principle, to probe and detect when this is happening. If single promises cannot be kept, over time, the source of trouble is easily localized to the agent making the promise, and its dependencies. On the other hand, a situation in which no promises can be kept (i.e. one is constantly having to repair state) is likely a situation of major instability, at the system level, not merely bad luck in a dud component.

9.4.1 COULD ALL PROMISES BE KEPT AND STILL YIELD UNPREDICTABLE OUTCOMES?

Promises are not guarantees, and they may not cover all possible outcomes. Could all of the promises offered to explain a system's behaviours be kept, but still leave room for inexplicable outcomes? The answer would seem to be yes, because we can only make promises about scenarios we anticipate.

A promise could be so ambiguously or vaguely formulated that it is actually worthless, because its outcome cannot be assessed e.g. 'We promise to protect the public from

harm'. How shall we measure this? A 'dumb agent' may promise to follow a set of rules to the letter of the law, but fail to fulfill its intended goal: 'We followed the regulations but people still died'. In this case, the intermediary step of promising to follow impositional rules adds to the ineffectiveness of the promises; it is easier to assess when agents promise what they will do, rather than what they won't do.

Consider the following examples.

Example 240. *Imagine a 3 dimensional system that you measure only in a two dimensional slice: you will be constantly surprised by what seems to pop out of thin air. Many systems of agents make promises collectively, as if they were a single entity[Bur15a]. For example, a radio, or television makes product promises about delivering entertainment, while the individual components promise electrical properties. A box of Corn Flakes makes a broad marketing promise from the box, representing everything inside it, while the individual flakes make quite different promises.*

Example 241. *A car is a collection of parts all of which might keep their promises. The sum of these promises does not imply the promise made by the whole, i.e. to be a car. Could all of these promises be kept, and yet still not yield the outcome of transportation? Imagine if the car is lifted off the ground by a crane, none of the promises are violated, but it does not succeed in fulfilling its function of motive transport. In this case, the reason is that there is a tacit assumption that the car will be on a road, yet this coupling to the environment is crucial. This is an example of overlooking something that is too obvious to mention (though tyre manufacturers might disagree).*

Promises refer to qualities that are either assumed to be under our partially deterministic control, or stable and repeatable, in spite of noise. The challenge we face, then, is what to do about the aspects of systems that we don't know about, and cannot make any kind of promise about. Systems of incomplete information are the potentially risky.

Law 3 (Fitness for purpose and incomplete information)**.** *A system of agents may exhibit no faults, and no errors but still have unexpected outcomes.*

In this case, one may only argue that the system is *flawed* in its (possibly designed) fitness for assumed purpose, because insufficient promises were proposed to cover the outcome. The proof is simple: imagine a system that makes no promise (as in evolutionary systems without clear selection criteria). Such a system cannot have faults or errors, by definition, as there is no standard to measure against. In this case every outcome is unexpected, for the same reason.

9.4.2 CATASTROPHES, EPIDEMICS, AND CRITICAL PHENOMENA

Because interaction is the source of dependency in systems, topology is a significant factor in promise keeping. Highly connected agents become hubs for epidemic spreading of failure.

Definition 193 (Single point of failure (again))**.** *Any agent within a system whose failure to keep a promise would result in immediate propagation of a fault throughout the system, i.e. the failure of external system promises.*

Compare this to definition 129, in section 6.5.3. A single point of failure traditionally means a special node, which, if removed or disabled, would cause a complete stoppage in the functioning of the system. The definition in terms of promises becomes more precise. Thus we have the associated notion of mission criticality:

Definition 194 (Mission critical (sub)systems)**.** *A subsystem that is a hard dependency within the larger system, and whose failure would result in an immediate negative consequences.*

The aim in any mission critical system is for any fault to have a manageable impact on the system, leading to (at worst) a temporary *degradation* of service rather than *interruption* of service.

Propensity for fault propagation is another effect that is understood from network science. Network *centrality* is a characteristic of an agent that measures its potential for propagating influence to other agents. *Percolation* is another critical phenomenon, whereby a system attains a critical density of dependencies such that it becomes possible for information (including faults) to travel all the way through a system from end to end.

9.4.3 INTRINSIC STABILITY: CONVERGENT OUTCOMES

Instability is associated with non-linearity, or amplification of effect. The key property for intrinsic stability, however, is not whether or not a perturbation of a system leads to a predictable outcome once, under particular circumstances, but whether the outcome is both reproducible and within tolerances. For this, even determinism is not necessarily enough. We also need *convergence* (see figure 9.1). Convergence to a so-called *fixed point* is a form of stability[Bur04a].

Convergence of outcomes is a sublinear response. This is the antidote to instability and variability.

Example 242 (Explosion)**.** *Think of an explosion. An explosion is caused by a deterministic set of promises being kept between molecular agents that are brought together.*

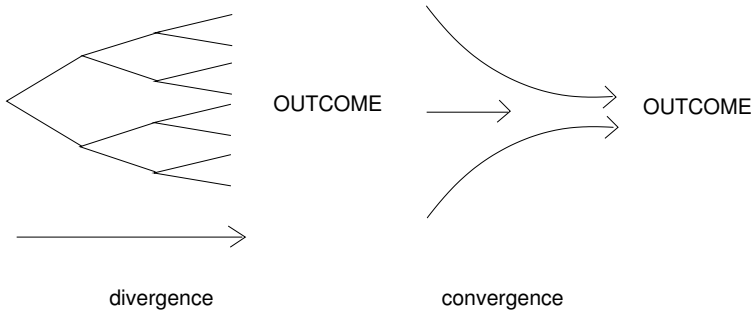


Figure 9.1: Divergence is the transition from state where the system is concentrated around a small number of states, to a large number of states. Convergence is the opposite: a concentration around a focused intended end state.

The system is both unstable because the outcome grows exponentially in response to the perturbation of being 'lit', and yet it develops entirely deterministically. The result is that there is too much information arriving too fast to process and contain, hence the system is dominated by the explosion and the normal semantics are wiped out. We can only associate stable semantics with convergent outcomes that are of finite scope (i.e. local in some sense).

The lesson of this section is that systems, which are designed by an intentional process, have to be designed so as to isolate and promise requisite stability, if they are to have predictable outcomes that users can rely on. That which is not promised, with the trustworthy intent to keep promises, is not to be depended on[150].

9.5 AGENT RESPONSIBILITY, CAUSAL MEMORY

It is often said that human error[Rea90, Dek06, D$\ddot{9}$6] accounts for the bulk of failures and outages. This is an empty statement, because ultimately all decisions or oversights can be traced back to a human, making humans an easy default target. By abstracting both humans and machines proxies simply as 'agents', which make promises, we shift from a potentially prejudicial account of individual capabilities (and indeed liabilities), to impartial *roles*, together with an assessment how well they are performed.

In trying to attribute blame, we have to understand that:

- Agents may or may not be acting with complete agency, unconstrained by circumstances beyond their control.

- The effect or 'memory' of an action taken by an agent does not last forever, because system interactions mix information sources reducing their impact; so, there is time limit on how long one might be considered responsible for a decision or action. Complex (non-linear) systems 'forget' the information that was input at some time more quickly than linear ones. Influence of an agent is scaled down by mixing, and eventually becomes dominated by new factors.

Both of these apply to humans and machines alike. The consequence is that no single agent, whether human or machine, has a decisive or lasting effect on total system behaviour, in general. The impact of an agent on outcomes has to be analyzed on a case by case basis.

From a Promise Theory perspective, whether an agent is human or not is relevant only insofar as it implies certain capabilities to keep promises. If one is impartial, then ultimately all that matters is whether promises were kept or not. Thus promises allow us to maintain impartiality while incorporating more subjective and qualitative issues into system description.

9.6 AGENT FIDELITY AND FAULTS

The accuracy with which agents are able to keep promises may depend on many factors, but ultimately such factors do not matter to the outcomes. This is why we are able to automate human processes using machinery. As long as they keep the same core of promises, surrogates and proxies may be exchanged for human labour. Most processes start out as manual human interventions, but given a sufficient understanding of the important promises, system agents can often be replaced by proxies that make the same promises, allowing humans to step aside.

Comment 19 (Humans, step aside and let the system prosper)**.** *The focus on outcomes rather than imperative procedures, implicit in a promise viewpoint, means that one may optimize promise keeping without trying to imitate the means of implementation that would be suitable for humans.*

Example 243 (Fidelity factors)**.** *Human fidelity might be affected by anxiety, stress, eating and drinking practices that dull awareness, etc. System fidelity is ensured by the availability of redundant backups, which are pre-integrated into the system (downstream principle). The predicted stability of all promises (both give and take) play into the final assessment of system fidelity.*

9.6.1 ACCURACY OF ACTIVE COMPONENTS (INTENTIONAL AGENTS)

It is tempting to delve into agents' characteristics, as we think anthropomorphically. Machines tend to offer better consistency (fidelity) in pursuit of simple unambiguous tasks, but not necessarily higher correctness, since correctness depends on the design's ability to behave in advancement of the system's major goal, in all contexts, something that requires awareness. Humans are currently superior in adapting to contexts (expected and unexpected). Machines do not notice changes of context unless they have been explicitly designed to do so; also, they are not aware of the consequences of their actions. However, we do not need to make any of these points, if we have promised outcomes in sufficient detail. The nature of agents is in the realm of speculation.

- **High fidelity agents** H_i: agents which keep their promises consistently, and with a high level of certainty.

- **Low fidelity agents** L_i: agents which fail to keep promises reliably.

The tautological nature of these definitions shows that we cannot assume that machines will perform better than humans. Such an assessment depends on many factors, e.g.

- Dumb machines are likely to be successful when isolated from unpredictability. *Machines are often act with high accuracy/fidelity during simple repetitive tasks, without adaptation. Humans may be high fidelity problem solvers, and designers, when working in short durations, within their field of expertise and interest.*

- Humans are likely to be successful when interested, practiced, and well rested. Humans are more likely to be susceptible to fatigue. *Humans typically exhibit low accuracy/fidelity agents when subjected to stressful conditions, such as those mentioned in section 2.8, but unreliable software, hardware, or exposure of unpredictable environmental interference can also make mechanistic agents unreliable, if they fail to adapt in a situation that requires adaptation.*

In between these extremes, lies a spectrum of smart cooperative systems, that are essentially partnerships between humans and machines. The so-called Internet of Things is a scenario where this will become of prime importance.

From the comments above, we see that sudden *system change*, outside the limits of fault tolerances, whether planned or unplanned can bring about situations where agents perform inaccurately, leading to propagation of errors and faults, instead of the isolation and repair.

9.6.2 PASSIVE ASSESSMENT OF INTENDED OUTCOMES

Let's call error and faults 'anomalies', or undesirable situations within the context of a system. They are spanned by the two categories of system behaviour:

- Semantics (qualitative (human subjective) interpretation) - correctness
- Dynamics (quantitative (objective) measures) - focus or targeting

The role of Promise Theory is to make these two aspects of the system part of a common framework, hence promises can blur the distinction between them.

Semantics always operate within the constraints of dynamics (systems cannot be asked to do impossible things).

- **Semantic (qualitative) anomalies** (distortion or botching of an intended outcome)

  This is an anomaly that can be traced to a human interpretation, since it requires cognition, and/or a specification of intent. Semantic anomalies can happen:

  - By direct interaction of a human within the system. e.g. a wrong choice, based on analysis, perhaps because the system has become too hard to fully comprehend.
  - Through the programming of intent by proxy, e.g. in software, or a machine design-flaw, relative to its purpose.
  - Unintended emergent effects identified by interpretation.

  Promise theory separates promises to provide (+) and accept (-) from one another. Semantic anomalies can exist in both:

  - (+) What is offered or executed is incorrect, ambiguous, misaligned with intent, or of the wrong type, etc
  - (-) What is received is misinterpreted or misunderstood, or expects the wrong type.

  Possible alleviations to semantic faults/errors include:

  - We can try to keep systems simpler, within the realm of comprehensibility. Or we can try to take humans our of the system altogether, and remove the need to comprehend anything except a constraint model.
  - The system is difficult to trust if it is non-linear, or unstable.

  - Semantic averaging over multiple agents, e.g. pair programming, dual-agents for confirmation (redundant flight computers, human supervision (co-pilot), etc) by equivalent agents (e.g. coworkers of similar skill level).

- **Dynamic (quantitative) anomalies** (measurable parameters fall outside acceptable limits)

  A failure to meet performance expectations, perhaps during unusual circumstances. Failure to meet an SLA. Reach a capacity limit in hardware or software that throttles the behaviour, or leads to a crash. An agent misses a target and fails to achieve the desired goal.

9.7 PROMISES AND THEIR RELATIONSHIP TO FAULTS

A promise is an intention that is documented as a tuple of components.

$$\pi \quad : \quad \langle S, R, b(\tau, \chi_\tau) \rangle, \tag{9.3}$$

where S is a sender of a promise, R is a receiver of a promise, and b is the promise body, which has type τ and constraint χ_τ. Promises are not directly observable, but their outcomes can be assessed, relative to the states of a given agent.

$$A(\pi) \quad : \quad \langle S, R, b(\tau, \chi_\tau) \rangle \to 0, 1 \tag{9.4}$$

$$E(\pi) \quad : \quad \langle S, R, b(\tau, \chi_\tau) \rangle \to [0, 1] \tag{9.5}$$

where 0 corresponds to 'not kept' and 1 corresponds to kept. The usual interpretation of a fault is the observation that an assessment, made by some agent, $A(\pi) = 0$, or that the average assessment over time $E(\pi)$ falls below some policy threshold. But how is this mapping made? What factors affect the assessment of a promise?

While physics and statistics focus primarily on describing change (dynamics)of entities (particles, atoms, etc), promise theory attempts to unify three essential parts of agents within a system: dynamics, semantics and context (see fig. 9.2). These aspects form complementary views of system descriptions (see table 9.1)

The table and figure 9.2 lay out the essential aspects of systems along three main axes, and their combinatorics. Promise-oriented reliability extends the expressivity and ambitions of the statistical component models in several ways:

- It shifts the attention to successful interactions between agents, rather than independent working state of agents. Hence it represents and acknowledges delocalization of responsibility.

ASPECT	COLLOQUIAL	REALIZATIONS
SEMANTICS	MEANING	ASSUMPTION, INTERPRETATION OUTCOME, QUALITY, VALUE, GRAPHS, PATHWAYS, STRUCTURE, CONSTRAINTS
CONTEXT	SELECTION	STATE, AWARENESS, CONDITIONS, ENVIRONMENT, MEASUREMENT, PHASE-SPACE
DYNAMICS	CHANGE	QUANTITY, CHANGE, TRANSITIONS, SCALARS, VECTORS, PROBABILITIES, SCALE

TABLE 9.1: THE THREE ASPECTS OF AGENCY.

- It allows us to model smarter agents that have memory and can adapt to different contexts.

- It quantifies even qualitative aspects of system behaviour, adding *dimension* or *measure* to the assessment of working state: how well, how fast, how according to intent (specification)?

Definition 188 describes a fault as a condition in which the actual state of the system does not match the promised (desired) state. In many cases, we depend not only on single promises, but promise bindings ($\pm$ neutrality) for cooperation. In other words, instead of thinking 'component is working', we think: is the component playing its role to all the stakeholders who rely on it? This does not only mean those components that are directly connected to it, but also those, which might be far removed from it, and still rely on its behaviours, unlike in the classical component *in situ* model. The promise arrows are virtual relationships, not physical connections.

9.8 THE BASIC PROMISE FAILURE MODES

Having identified the ways to define faults and errors consistently in terms of promises, we can consider how the failure to keep certain promises can lead to these conditions.

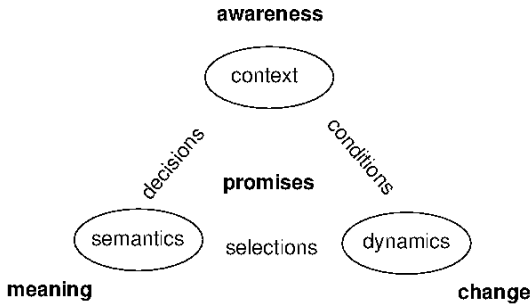


Figure 9.2: The three aspects of systems for a unified treatment. In between context and interpretation, lies decision-making, to guide system dynamics. Between context and dynamics, we find the state or condition of the environment, to which we attribute semantics. We attach meaning to change by selecting favoured outcomes, in a given context. Finally, we attribute meaning to current state and desired change by making promises about desired outcomes, i.e. we document what we intend.

9.8.1 STANDALONE AGENT PROMISES

Promises to self, to no one in particular, or to purely passive stakeholders who do not act or depend explicitly on a promise, represent the simplest declarations of intent (see fig. 9.3). Such promises may assessed to be either be kept or not kept, by any agent in scope, i.e. who knows about the promise.

Each promise acts as a local 'measuring scale' for assessing system characteristics along different 'axes', like in a coordinate system.

Example 244. *If a person promises to tie their shoe laces, brush their teeth and wash behind their ears, then we have three axes on which to measure and compare the state of person.*

Each promise type becomes an axis for measurement: a possible degree of freedom for agent behaviours. Assessments from kept to not kept (either on a binary or on a continuous scale), allow us to collect data meaningfully about whether a system is measuring up to its intended goals. Without any promise, we cannot measure the success or failure of a system, we can only study patterns of behaviour[151].

The simplest reliability question we can ask of a system is thus:

- Was any promise made relating to observed behaviour?
- Was the promise kept?

Answering these questions help to determine whether

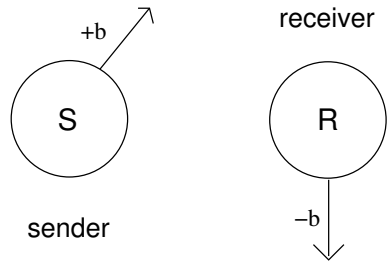


Figure 9.3: Single agents may make promises to give (+) or accept (-) something from another agent. Note that it is the sign of the promise that denotes direction of flow, while the arrow is always the direction of intent, originating from the responsible agent

- The system is behaving as designed, but it does not currently align with its stated goals (design flaw).
- The system is behaving as designed, but an observer's can't or won't accept the behaviour (inconsistent or mismatched promises).
- The system is behaving contrary to design (simple assessment of fault).

Promises act as calibrators for fault detection. In the traditional black box model, all of these modes of failure (above) are random faults, and the cause of the fault is unknown. We cannot resolve in which specific agency the misalignment with intention lies. But, by making explicit promises that document both qualitative and quantitative constraints, we introduce a measuring stick for the design of a cooperative system.

Example 245 (Security breach)**.** *A condition that is not easily defined as fault in the classical theory of reliability is a security breach. In a promise system we are able to define what specific promised condition was violated to lead to a security breach. In this way, security just becomes a promise, and a breach is only a fault or a particular kind of promise not kept.*

Comment 20 (Monitoring and alarms)**.** *Promise keeping also applies to all forms of measurement and monitoring for fault detection. Measuring without a stated promise is a waste of time, unless you are a researcher or investigator in search of clues, because the monitor doesn't know what to look for. In most IT monitoring systems, if any promises are made at all, they are usually ad hoc time series thresholds that are set by gut feeling, not aligned with system design promises.*

9.8.2 TIMING OF PROMISE KEEPING AND ASSESSMENT, AND THE NYQUIST SAMPLING FREQUENCY

A promiser that tries to keep its promise too late has not kept its promise, and the promisee interprets this simply as a fault. Responding within a timely fashion might be considered part of an extended protocol or handshake.

Using Information Theory, we can apply Nyquist's sampling theorem, in reverse, to infer that the information propagation density needs to be at least twice that of the dependency sampling, in other words: in order to keep a continuous, time-varying promise, when the promisee uses the promise at a frequency ν, the promiser will need to ensure its accuracy at the Nyquist frequency of 2ν in order to track the changes.

Law 4 (Nyquist frequency for promise maintenance)**.** *A dependency should maintain or update its current state at (at least) twice the rate at which the promise changes are used as a dependency to avoid a perceived fault.*

System with periodic sampling can optimize this maintenance cycle for rapid repair. This changes the relative economics of redundant, replacement, versus repair, discussed in chapter 11.

9.8.3 COVERAGE: BEHAVIOURS THAT ARE PROMISED AND NOT-PROMISED

Systems exhibit two kinds of behaviours:

- Intended or promised behaviours (what we normally call agency)
- Unintended behaviours (disrupted or emergent behaviours, from environmental noise and interruptions), which are *ad hoc* as there is no promise associated with them.

So a system is either guided or not. If it is not guided, there is not much we can say about its behaviour except by watching and trying to learn. If a system that does not make promises exhibits apparently predictable behaviour, we might infer that it does in fact make a concealed promise. Sometimes we believe in promises that have not been made, and speak about emergent behaviours.

Emergent behaviours are observed behaviours that resemble intentional behaviours, and may recognized and named, but where no promise was actually made[BB14a]. These are typically collective behaviours, but taking into account scaled super agents, we might not be able to tell the origin of emergence.

Definition 195 (Emergent behaviour)**.** *Observed behaviour, which is consistent with the existence of a promise, but has not been promised explicitly.*

In promise theory, we represent all behaviours by transition *trajectories* through the possible *states* available to agents[152]. These trajectories may be largely explained to be the correlated outcomes of promises, perhaps with corrections caused by noise or implementation errors. The promises might not drive the trajectories, but they constrain the observed behaviours depending on the degree to which they can be kept. This is what we mean by reliability.

Assumption 1 (Systems are non-deterministic but constrainable)**.** *All systems should be considered non-deterministic, regardless of their design, since they interact with environments which cannot be predicted. Agents act non-deterministically, and they keep and receive promises non-deterministically. Thus, they cannot be predicted precisely, but their behaviours can be narrowed or focused into an acceptable range of values.*

Non-deterministic systems experience the arrival of events or 'random' (non-modelled) occurrences, some of which are acceptable to system policy and some of which are not. Events which are not acceptable may be called faults.

9.8.4 DESIGN FLAWS RESULTING FROM MISSING PROMISES

Promises are often made conditionally, in limited contexts. Sometimes a system designer does not make promises to cover all the possible contexts, and situations arise in which no promise was made. Agents in the system have no guidance on how to behave. The behaviour of the system is them *undefined*, and we cannot anticipate what will happen. Generally, as the scale of a system becomes large (large N, large size, etc) then we should expect more of these behaviours, simply because we have more and more incomplete information.

Comment 21 (Security exploits)**.** *Many security exploits are based on the absence or mismatch of promises covering unexpected conditions.*

9.8.5 FAULTS IN COMMUNICATION

The most basic flaw in a cooperative system is the lack of a common language for mutual understanding (lingua franca). If agents, whether humans or machines, cannot make themselves understood to one another then it is not possible to make promises, or to assess their outcomes. Without promises, there cannot be expectation and cooperation will be ad hoc. Similarly, if there is only partial understanding, then errors or interpretation can lead to faults.

Comment 22 (Lingua franca)**.** *The language of a promise must be shared between the promiser and the promisees. It need not be a spoken or written language: any kind of symbolism that conveys meaning will do, e.g. the shape and placement of a door handle, the form of a chair, the on/off symbol on an electrical device, standard road signs, etc., are all examples of languages for communicating intent.*

Example 246. *External circumstances may put agents into a mode where they lose their ability to understand the language of their surroundings. For humans, loss of comprehension gets gradually worse in high stress environments. For machines, there can be sudden and catastrophic breakdown of understanding as a result of changing the context of a single component, e.g. a version mismatch. Moreover, while humans can sometimes adapt to change, machinery (especially that which is designed with the assumption of a protected environment) is particularly exposed to error arising from intolerance of change.*

These scenarios result in a breakdown of the assumptions on which a system was built.

- An exchange relating to a promise, which cannot be understood, cannot be received as a promise by another agent.
- Agents may only understand part of a promise, if their dictionary of language or terminology is incomplete[Bur15a].
- Words with multiple meanings (homonyms) can lead to misinterpretations of what is promised.
- Shared or unstated assumptions (that which is taken for granted) might not be as universal as one expects.

Example 247 (IT system languages)**.** *Language is a basic part of information technology. Common languages are assumed in protocols, software versions, user experiences and interfaces, encrypted messages, procedures, and icons.*

9.8.6 SHARED ASSUMPTIONS

Trusted assumptions and implicit promises are a common cause of system design flaws. Assumptions about what is common knowledge can allow significant compression of communications.

Example 248 (Fuel types)**.** *If an driver (part of a car system) believes that all cars run on petrol/gasolene, whereas a particular vehicle runs on diesel, an attempt to keep the*

promise to refuel the vehicle could result in a fault. If the car does not make a clear promise about what kind of fuel it can use (-fuel), then it might bind to any promise of type (+fuel). Today the fuel nozzle receptors for petrol and diesel are different, making much clearer promises about what type of fuel is expected.

This is a trust issue. For example, the assumption that, once set, a system property is immutable and is relied upon not to change could save considerable monitoring communication. If the assumption is violated, them it becomes a straightforward failure mode: in this case, a design flaw, since assumptions are usually not promised.

Agents do not have to all share the same language, but they need to be able to communicate their without errors of comprehension.

Example 249 (Power supply)**.** *Across large parts of the world, the electrical power is 240 volt A.C. In the North America and Japan, it is 110 Volts A.C.. The power outlets have very different shapes and connecters. The changes in power are sudden and often take travellers by surprise. This has led to many destroyed electrical devices.*

Example 250 (Character encoding)**.** *A document written in a character encoding such as ASCII, EBCDIC, or Unicode is not compatible with a document written in a different encoding, and appears to be garbage.*

Example 251 (Protocol)**.** *If one takes away all punctuation from a text, it becomes quite difficult to read, and may result in fatal misunderstandings. Take the well-known joke of the panda who enters a bamboo restaurant with an extra comma, and comes out angry with blood on his hands, having read: "Panda eats, shoot and leaves" instead of "Panda eats shoots and leaves".*

9.9 AGENT INTERACTIONS

An autonomous agent does not depend on any other; it has everything it needs, and is immunized against faults in other agents' promises. This is the value of autonomy. There is also value, however, in delegating to specialist services, which brings with it dependency and risk of fragility.

9.9.1 COOPERATION FAULTS ARISING FROM NON-NEUTRAL PROMISE BINDINGS

If we can assume language is mutually compatible, then faults can still arise from incomplete intent. Promises have polarity:

- + promises express the intent to provide e.g. service delivery
- - promises express the intent to accept, e.g. access control rights

Both are pre-requisite for the expectation of service propagation from one agent to another.

Rule 1 (Promise graphs should be promise-neutral)**.** *Promise bindings always consist of a (+) promise and a (-) promise:*

$$S \xrightarrow{+srv} R \qquad \textit{Service is promised} \tag{9.6}$$

$$R \xrightarrow{-srv} S \qquad \textit{Service is expected, and will be used} \tag{9.7}$$

The promise sum is neutral i.e. $\pm srv \simeq 0$.

A mismatch between positive and negative promises indicates a semantic flaw of design or implementation. If it does not currently exist (it might be accidentally true, but not intentionally true) then it will likely become a problem.

- **No + promise**: User expects a promise to be kept, but no promise was given.
- **No - promise**: A promise was given, but it was never used.

This provides a simple syntactic check, based on the model of promises.

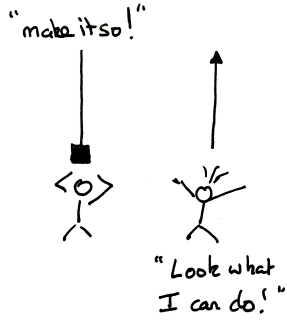


Figure 9.4: A cartoon comparison between imposition (left) and promise (right). Impositions are unexpected exterior intentions, based on presumption of state we don't know. Promises self-realized and local, based on locally available knowledge of state.

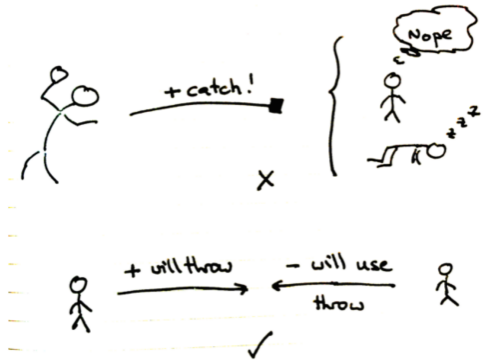


Figure 9.5: The implication of a promise (below), as opposed to an imposition (above), is a state of readiness rather than surprise.

9.9.2 FAULTS IN INTERACTIONS BETWEEN AGENTS

When a stakeholder or recipient R depends on the result of a promise sent by S, a system dependency has failed (see fig. 9.6). R must assess that its expectations of cooperation, by S, have not been met. The figure shows the contrasting cases for cooperation. The

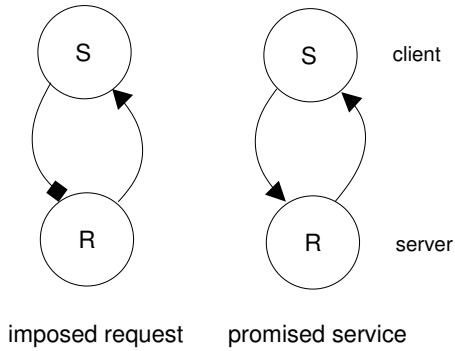


Figure 9.6: A dependency of one agent on the promise of another leads to broken cooperation.

bindings may be:

$$S \xrightarrow{+b}\!\blacksquare\; R \tag{9.8}$$

$$R \xrightarrow{+b} R \tag{9.9}$$

i.e. S imposes its request for $+b$ onto the recipient R, and R answers with a promise to provide $+b$. This represents the usual tradition 'push button' thinking in technology. An imposition is understood to come from outside the agent. It contrasts with the voluntary promise relationship, where the promise always comes from within the agent:

$$S \xrightarrow{+b} R \tag{9.10}$$

$$R \xrightarrow{-b} R \tag{9.11}$$

i.e. S promises a service $+b$ to R (voluntarily, and without imposition), and R promises to accept it $-b$. What could be the reasons for non-cooperation for impositions? Here are some proposed failure modes:

1. S's imposition request was not sent.
2. S's imposition request was not received (S and/or R were available).
3. S's imposition request was not understood, or interpreted correctly.
4. R's promise to reply was not made, perhaps because the imposition was not received.
5. R's promise to reply was made but not kept.
6. R's promise to reply was made but not kept in time.

For promise relationships we have a symmetrical situation:

1. S's promise to deliver was not made, or was made but not kept.
2. S's promise was not received, or was not understood, or interpreted correctly.
3. R's promise to accept was not made, or was made but not kept.
4. R's promise was not received, or was not understood, or interpreted correctly.

Since promises are the sole responsibility of the agent making them, neither agent depends on the other for the making of its promise.

- The imposition version of this has the form of a client-server system (push).
- The promise version of this has the form of a publish-subscribe (pull).

Clearly, the number of failure modes is huge compared to what we are usually willing to invest in prevention or repair. This makes the design of systems that are fault tolerant seem attractive both as a practical and a potentially cheaper option.

9.9.3 SERIAL REPAIR VERSUS PARALLEL FAILOVER VERSUS FAULT TOLERANCE

We have options to maintain the continuity and availability of a system: repair quickly (within the serial queue), to failover to another supplier (redundancy), or to fail gracefully without a result from the dependency (fault tolerance). Seeking the balance between these different strategies is potentially a conflict of interest, representable as a type II model[Bur04a] or strategic game. We examined some basic cases using traditional structure function analysis in chapter 10.

There are three basic patterns:

- Modular dependency, or serial dependence of one module on another, such as software packages.
- Replica sets, or parallel redundancy to give failover in case of fault.
- Retry, or soft failure with tolerance.

We can define these cases explicitly.

Definition 196 (Serial dependency)**.** *In a flow-based, transactional system, an error introduced by a dependency can affect the dependent agent, and all later agents. Once lost, information cannot be recovered. This is a high risk delegation.*

Definition 197 (Redundant dependency)**.** *If a serial dependency has several alternative sources on which it can rely, it is less likely to be in a situation where one of them is unavailable, provided it promises due diligence in using the available services. This is a mitigation of risk.*

Definition 198 (Tolerance)**.** *In a promise-based system, an error introduced into a serial chain can sometimes be absorbed, mitigated or eliminated by later actions, provided authoritative (template) information suitable for error correction can be kept locally.*

In the flow case, changes are relative, and we can prove that error correction is Byzantine or impossible. In the promise or policy system, changes are absolute, and we can define error correction by reset or zero-operation[BC11].

Example 252 (Fault tolerance - bullet proof vests)**.** *A bullet proof vest is an attempt at fault tolerance to fault impositions. A redundant failover alternative would be to rely on a second agent, and accepting the loss of the first. Realtime repair would be to patch up the shot agent in time for the next altercation.*

The sampling rate for agents working together plays a key role in causality and error propagation[BD07, Dis07]. All events that happen within a single sampling resolution are simultaneous, according to the next agent in a chain, hence rapid repair can lead to unnoticed errors. This is the approach used in memory CRC, processor errors, CFEngine, etc. Once faults escape confinement, they cannot be undone.

9.9.4 REDUNDANT ALTERNATIVES—MITIGATING A SERIAL DEPENDENCY

By reversing the signs in figure 6.11, we can average out unreliabilities, as one does in data analysis by repeating measurement samples. Agents can also measure the variability and warn about inconsistency. Unlike pure data values, semantic averaging may be considered unacceptable, when it has not been allowed for.

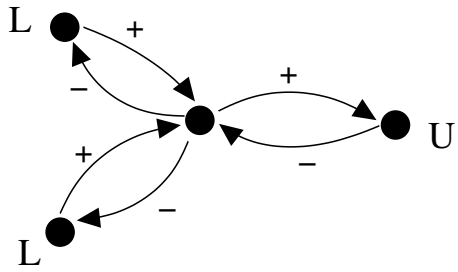


Figure 9.7: Convergence by natural aggregation and averaging

1. The multiple sources S_i all promise service $+d$ to the intermediary, within their accuracy tolerances.

2. Intermediary/relay I, promises to accept the values and select one, if suitable for meeting the dependence $s|d$.

$$S_1 \xrightarrow{+d} M \tag{9.12}$$
$$M \xrightarrow{-d} S_1 \tag{9.13}$$
$$\ldots \tag{9.14}$$
$$S_n \xrightarrow{+d} M \tag{9.15}$$
$$M \xrightarrow{-d} S_n \tag{9.16}$$
$$M \xrightarrow{+s|d} R \tag{9.17}$$
$$\tag{9.18}$$

Now I has secured sufficient redundancy to ensure that one of the promises will be kept. However, how do we know that the sources S_i are consistent? I needs a policy to select $+d$ from one or more of the agents, and possibly combine them. If errors are random, a majority or average should suffice. If errors are systematic, and repeatable,

What is not clear from this simple argument is whether the two alternative sources are exactly equivalent, or as good as one another. This is not readily expressible in a classical component analysis; in a promise model, we have the ability to look at what alternatives promise and measure expectations relative to those promises.

It remains up to the aggregating agent (user) to deal with inconsistencies. Even if we work very hard to build a system that has consistent redundancy, the promises cannot be guaranteed, so it becomes the responsibility of the selector or aggregator to keep promises consistently even when the dependencies vary unpredictably.

9.9.5 SEMANTIC FAULT TOLERANCE BY AVERAGING - REQUISITE DIVERSITY VERSUS REDUNDANCY

In the previous section, we considered how to bolter the reliability of a single kind of promise. Reliability strategies may also be characterized as a form of statistical averaging of the system dynamics. Putting all your eggs in one basket is a fragile strategy, so we can mitigate loss by either maintaining the availability of a single promise or by offering alternative courses of action. A mixed strategy (in the sense of game theory) is a fault tolerance strategy based on different promises—alternative trajectories rather than a strengthening of one.

9.9.6 SERIAL FAULT TOLERANCE: ADDING MARGINS FOR ERROR

Rather than propagating faults forwards, fault-tolerant agents can, in fact, absorb them, providing ‘shock absorption’ for errors, and correcting them to within the limits of their own tolerances (see fig. 9.8).

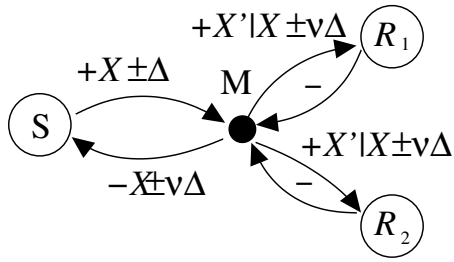


Figure 9.8: An error at the head of a chain of dependencies can be absorbed by increasing accepted tolerances from $\Delta \to \nu\Delta$.

1. S promises service X, within its accuracy tolerance $X \pm \Delta$.
2. Intermediary/relay M, promises to accept the value in the range $X \pm \nu\Delta$, where $\nu > 1$.
3. Since $\nu\Delta > \Delta$, I accepts the service from S, and promises X' to R, hence there is recovery from a broad range of failures.

The assumption of a dependence means that there must be limits on ν too. If M could manage without the service from S, then why depend on it at all? Perhaps the tolerance of no service from S could be maintained by caching past results from S, with eventual expiry.

This shows that fault tolerant agents can keep promises, even when connected in series, by absorbing errors and faults upstream.

9.9.7 TOLERANCE OF SERVICE INCONSISTENCY DURING SELECTION FROM REDUNDANT PARALLEL ALTERNATIVES

The difference between agents might not simply be in performance or data (dynamics), but also in the details of behaviour (semantics). The equivalence of the two sources S_1, S_2 of a promised dependency assumes that the sources are sufficiently alike to keep the promise needed by the intermediary I. But what if the agents have different tolerances, accuracies or limits to their capabilities? We have to ask: how sensitive is the recipient to exact similarity?

Related to statistical averaging of data sources for resilience, software engineering sometimes wants to have global consistency. Dealing with inconsistencies in semantics requires a new policy: e.g.

- Picking a majority/quorum (Paxos etc) - sometimes called multi-master
- Picking the most recent value (vector clocks).
- Checksum or certificate verification and rejection

If we want system *fault tolerance* rather than potential for failing to reach majority agreement, it's clear from the foregoing example that we should tolerate inconsistency rather than try hard to ensure it, as downstream absorption is still the most robust strategy for keeping the total promise.

Comment 23 (Consistency)**.** *There are fundamental reasons why strong consistency is not achievable in the general case. However, the idea of a quorum between servers, i.e. a minimum number of sources that must agree on an outcome in order to make a valid inference, is a popular methodology, especially in databases.*

All kinds of arbitrary and artificial strategies to claim truth are used for quora: e.g. using an odd number of servers (at least three) so that there can be a majority. None of these guarantees that the majority result is actually 'correct', unless we can verify what correct is promised to mean.

9.9.8 CONVERGENT LOCAL REPAIR

Repair is an agile method of fault tolerance. If a fault develops in theory, but has not been tested in practice, then there is a window of opportunity in which it may still be kept. Thus a sufficiently rapid repair can be an effective way of keeping promises. Some systems, e.g. agile fighter aircraft, are designed to ride the edge of stability in this way, as it affords them far greater maneuverability on a shorter time-scale than a more stable design would.

By embedding compressed information about intended state into a system throughout, repairs can be made in real time[Bur03, Bur04a, SW49]. If agents along a chain experience errors that can be self-corrected (e.g. Shannon error correcting theorem, Hamming codes, etc), a feedback loop may be applied to keep the total promise of the system, within a time $T_{\rm repair}$ (MTTR). This requires the presence of a model (and thus potentially an agency responsible for providing and updating the model also).

Inline repair, or agents in the chain, can be appropriate or even cheaper or faster than redundancy if $T_{\rm repair}$ (MTTR) is less than the time it takes to acquire, aggregate and select a value from redundant sources (see section 9.9.4).

Example 253 (Detailed balance with CFEngine)**.** *The general strategy introduced and used by CFEngine is to keep all agents as close to a policy state as possible at all times, then hope for fault tolerance.*

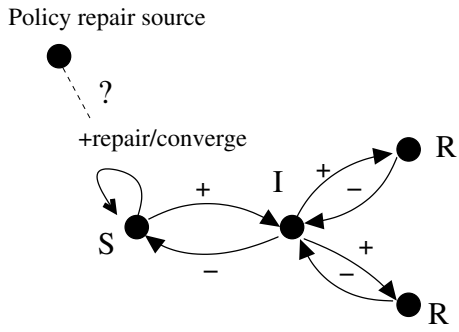


Figure 9.9: Convergence by feedback mitigates faults at the head of the chain, before amplification

Comment 24 (Continual repair is continual delivery of change)**.** *Information re-injected into the system could be:*

- *Fixed information: repair to original state.*
- *New information: course correction, repair to improved state.*

The absorption of errors is the essence of system testing *before release. This kind of tight loop is the approach used in continuous delivery[HF10].*

9.9.9 PARTIALLY ORDERED PROMISES

A chain or sequence of ordered promises is a fragile structure. Ordered promises form chains of dependencies. If a single promise fails, all subsequent dependees must also fail.

9.10 FAULT PROPAGATION AND FAILURE DOMAINS

Many engineers believe that *modularity* prevents the propagation of faults. To isolate an agent from a dependency one must withdraw all promises which use that dependency—abstaining might be an effective strategy for the spread of consequences, but it also invalidates its purpose. Locality may help to limit the propagation of a faults, if causes are themselves modularized, and downstream clients make appropriate promises to recover (see section 2.4.3). The term ‘fault domain’ is widely used to imply a kind of semi-permeable membrane that prevents faults from having consequences beyond a certain perimeter. Security perimeters (firewalls) are a common example. Such barriers may

select only specific messages from a wider set, but they cannot prevent the propagation of influence unless there is independence of the promises made by those modules.

A modular system may, on the other hand, help to localize the source of a fault if the chain of causal outcomes leave traces in the states of agents as they propagate.

Modules are sometimes referred to as 'failure domains'. The implication is that such imaginary regions of a system help us to localize issues of causality. Using promise theory, we can try to answer whether this claim has any validity. Failure containment is about limiting the propagation of influence through a system of interest. We hope to prevent the transmission of influence by creating so-called 'boundaries' of domains, but what we really mean is interfaces that act as *filters*.

Coupling between modular components occurs through conditional dependency, and dependency implies strong coupling. A change in the keeping of a promise by one agent affects the dependent agent. To avoid such strong coupling, one may try to disperse it by introducing redundancy (in parallel). A failure in the depender will propagate into a failure in the dependee. Boundaries that truly isolate parts of a system are therefore fictions of manufacturing. Once a system is assembled, it no longer has dynamical separation. Such arbitrary contextual boundaries are always porous to dynamical behaviour: we like to conjure the illusion of protection for our own sense of security, but every connected system is by definition connected. Barriers might classify certain areas, making certain promises, but few systems bother to promise what they will not allow. True isolation would mean breaking the system's coordination, and therefore its cooperative functionality. Thus, the claim that modularity leads to isolation of failures is ultimately bogus.

Example 254. *The attempt to lock down computers for security has often been at odds with their usability. The United States 'STIGS' security recommendations for government and military were widely reputed to render personal workstations actually unusable.*

The purpose of semantic modules is to serve as a reminder of context. The promises made by each module offer information to those interacting with it, that help them to adapt their own behaviour. Humans have limited cognition. We can keep in mind a handful of things, and remember only handfuls of stack levels in a thought process. However, the systems we build far exceed these cognitive capacities, and so we must deliberately and modestly erect protections against our own weaknesses. This inability to trust human comprehension under complex circumstances, especially the wider impact of changes we make under local circumstances, it is practical to imagine 'modules' as voluntarily limitations of our own causal reach: either reminding us of the limits of a context, or trying to prevent others from imposing their own influence on us.

Example 255. *When releasing a pesticide, there is always a worry that it will travel*

beyond the field where it was needed and lead to side effects. When antibiotics are used in animals, there is evidence that these drugs find their way through the food chain into far reaching parts of an ecosystem. When making a global search and replace in a document, we may try to limit context by keeping different data of different context or intent in different files. Search and replace does not normally traverse file containment.

Module boundaries, then, are principally to be understood as reminders of human context. They are semantic boundaries, and semantic boundaries are not boundaries for dynamics. If we put a fence around a single tree in a jungle, do we expect to contain the spread of a parasite? If we draw a circle around a collection of cubicles in an office space, or give certain members red sweaters to label them, do we prevent the the sudden death of these coworkers from affecting the rest of the business?

Labelling of contexts, or filtering promised outcomes, leading to partial isolation is something that depends on both the (+) and (-) promises between a promiser and a promisee.

- A module is a superagent boundary. This does allow us to limit and define the main interior and exterior promises. The definition of interior and exterior promises allows us to adapt a design for weak coupling, e.g. through redundancy.
- Does modularity enable fast repair? You can replace as a hot spare, or have fast failover, fast repair, but that is not the same as developing a solution that scales in parallel. Architecturally (cooperatively) these are quite different designs. There is some evidence that rapid repair has some advantages over redundant design, but it all depends on the timescale at which you can repair the system.
- Availability (CAP or no CAP) is about a sampling rate. System time is not a continuous quantity. MTTR is really MSTR or mean number of samples lost before repair. So if we can work very fast to correct the flawed operational envelope of a system faster than the sampling rate, we will never notice the bumps. This is how fighter jets and space flights are designed. They are designed to deal with unpredictable instability, but they have computers correcting the flight parameters at a rate that is much faster than the perturbations they experience.

Example 256 (Semantic confusion)**.** *Does this help to localize the error? That depends on how they are called and instrumented.*

```
string.h
strings.h    - not helpful
sys/string.h
```

these things were written at a time where compiler memory was expensive. So there is also a question of optimizations intruding on semantics.

9.11 INNOVATION WITH AND WITHOUT INTENT

The capacity of a system to 'innovate' or invent new emergent states that were not designed by intent appears to also be a network phenomenon. Occasionally systems seems to invent new possibilities. We sometimes call these bugs, because the outcomes were not intended. The term emergent phenomenon is also used.

9.11.1 BUGS AND EMERGENT BEHAVIOUR

Although engineers and designers strive to make systems keep certain promises, only security and safety engineers usually try to ensure that certain outcomes *don't* happen. This underlines a key conflict of interest between system builders and security and safety engineers.

Definition 199 (Bug)**.** *An outcome of a deliberately designed intentional system that was not intended.*

In the definition of a bug, we once again see that, without a promised intent, we cannot define a bug, only a surprise. This is quite similar to the definition of an emergent behaviour[BB14a]. Emergent behaviours are those that seem to exhibit intentional behaviour, but for which no promise has been made.

We would expect, as the scale of an intentional system grows, its ability to behave predictably diminishes somehow, as unforeseen pathological interactions become harder to predict. However, rationally, this depends on the degree of interaction not on the size per se.

9.11.2 SURPRISES: EXPLORATION AND INNOVATION

While systems often get more general and statistically predictable at scale, the opposite is also possible. As one mixes different parts together in a network of interactions, new phenomena occur. This turns out to be fundamental to the way ideas are generated. Although novelty can exist at all kinds of levels of sophistication, the mechanism for novelty is probably the same at all levels. It involves Darwin's key observation: mutation, mixing and selection.

To put this in a promise theory language: independent agents go off and make changes to existing promises, then then return and recombine their efforts and select the

best of the many experiments. This is branching and merging. It is sexual selection. This form of networking is the essence of a primordial soup. Perhaps it is even how new ideas form in a brain, we don't really know. It is how evolution comes up with new ideas, however: sexual mixing combined with environmental selection constraints are a form of thinking that leads to improved processes. This is innovation.

Example 257 (Open source innovation)**.** *Open source innovation revolves largely around versioning repositories where contributors branch off copies of the current state of an idea, then modify it with atomic 'commits' to match their own experiences. Later, these changes might be merged back into the main timeline of the software, selecting those atoms from multiple experimental branches that seem to succeed best. This is sexual innovation.*

9.11.3 MIXING AND SEPARATION OF CONCERNS: INNOVATION AND MUTATION

Unintended creativity is not something we always want in systems, but it can emerge at scale. We might consider innovation a security threat to stability, or a beneficial source of opportunity. Sometimes it is useful: this is why we build teams and companies and have universities and centres of excellence. If you throw active agents together, new things will emerge[153]. This is the essence of how creative innovation begins in a network[154] . Innovation is a balance between a promise of mutation (+) and selection (-). Intentionality lies on top of the virtual and even physical connectivity of whatever effective network or spacetime connects a system. Within that bound, an agent's multi-tasking, or timesharing determines limits on the number of interactions it can have with others. Interactions do not always happen directly through explicit links. They can be mediated by third parties. This is what is meant by a catalyst.

Definition 200 (Catalyst)**.** *A 'third party' agent whose mutual interactions with other parties lead to an effective transfer of influence between them.*

This is sometimes called a stigmergic or covalent interaction. For example, someone knits their friend a sweater. The sweater is seen by someone as they are waiting for a bus at the bus stop, and they ask to buy the sweater. Next week the same thing happens with a scarf. Information has been transmitted via the catalyst of the bus stop.

Formally partitioned cells are not necessarily independent, as they compete within shared boundary conditions. For example, cells might compete for the same resources, or mediators and brokers might delegate shared resources. In either case, these constraints implicitly connect otherwise intentionally separate cell networks together (weak coupling). This is why security breaches are so much more likely than many expect. These

'covert channels' bridge networks. Promising to exclude something is difficult, because it relies on knowledge of impositions, which are (by definition) unknowable.

9.11.4 FORCES AND SPECIALIZED ROLES

A hypothesis of promise theory is that one may define a notion of force for agents, which is attractive when there is economic advantage, and repulsive for economic disadvantage[155]. The formation of superagents thus comes about, for economic reasons[BF07a], by the value of collaboration. If the promises are unconditional, superagents will be localized. If they are conditional, the clusters are ordered and may thus be distended or even distributed.

- Agents, which make the same kinds of promises of same polarity, tend to repel one another.
- Agents, which make complementary (binding) promises of opposite polarity tend to attract one another.

If the hypothesis is true, one may expect behaviours such as the following:

- Dependences are held together by promise bindings.
- Competing services tend to spread out in the system region, unless they are held together by something more important.
- Distributed competitors, may cluster around a common dependence, such as shared infrastructure hubs, e.g. malls, districts.
- Chains of transport agents, bound together by conditional promises, in relay configurations.

This suggests that agents, which play the same role, will tend to move apart unless they are held together by a promise of cooperation. In promise theory, a specialized role characterizes a pattern of agents that make similar promises. By specializing specific tasks to specific agents, each agent can be more focused in learning and adapting, but acquires an additional cost of cooperation proportional to some positive power of the cluster size.

Example 258 (Data replication and cache consistency)**.** *Consider the problem of data replication. A specialized promise, like a database, may need to work together with others to serve a wide area. Since they make the same type and polarity of promise, they tend to distribute around the region. However, they also need to cooperate to ensure that they are working together rather than against one another, with the same data.*

This consistency comes at the price of promising coordination. This mutual affinity for cooperation tends to bring them closer together, so they will find equilibrium orbits, where the promise forces are in balance. The resulting collaboration cluster of agents may be considered a superagent.

Superagency collaboration is a *short range* interaction (between interior promises) [Bur15a], in the sense that it is a direct agent to agent interaction.

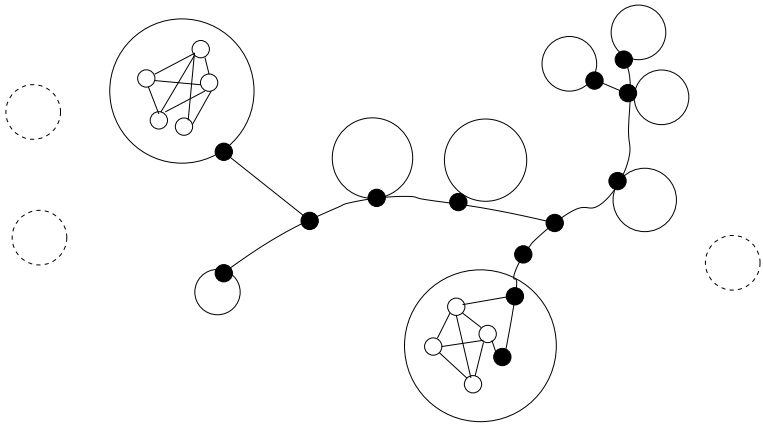


Figure 9.10: The geometry of superagents may fill space in different ways. Infrastructure that interconnects other agencies is a superagent in its own right, involving linear or approximately linear cooperation between member agents. Under preferential attachment, agents N_I tend to cluster around the infrastructure agency, leaving a few N_0 padding out the spatial volume. The circles around the subagents may be considered infrastructure binding the agents together.

The notions of attraction and repulsion are wired into our imaginations in terms of spatial concepts. We are used to electromagnetic and gravitational forces, like 'pressure', which seem unambiguous because we observe that a very large scales, where the affinities achieve a deterministic quality. At much smaller scales, where atoms and individual agents interact, these 'forces' are less ballistic and more probabilistic in nature. A promise model belongs at this low level scale, where forces should not be considered too literally as if they were classically deterministic.

Even without an embedding spacetime to describe vector directionality, we can speak of agent affinities, like the interactions described in molecular chemistry, where spacetime plays no real role. With a physical volume to embed a graph of promise-keeping interactions, geometry ties range to distance, but in a virtual network (which includes transport of messages by intermediate carrier), short range interactions can

also be disseminated over a longer effective range, by adding cost or latency (such as in telecommunications).

9.11.5 PROMISE NETWORKS THAT PERCOLATE

Specialized promises naturally lead to small molecular clusters of agent 'atoms'. They seldom span large areas, because promises act like short range interactions, which is also why superagents can be considered quasi-atomic black box agents (see figure 9.11). General survival promises are common to most agents, and pertain to the most general kind of infrastructure in systems: power, food, air, water, etc. These are a minority of promises that are ubiquitous.

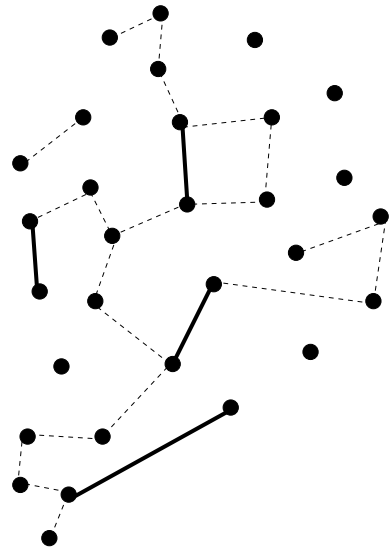


Figure 9.11: No single type of promise binding (dark lines) leads to percolation of value in the promise graph. However, with conditional dependency, and sufficient diversity and homogeneity, there can still be effectively close to N^2 links whose value converts into a common currency.

If we let N_τ be the number of agents that consume a promise of type τ, then we expect the class of τ related to survival to be of the order $N_{\text{survival}} \simeq N_I$, in the meaning of the city model. For all other types, $N_{\text{other}} \ll N_I$.

However, we'll see in section 5.15.7 that long range interactions are also needed to explain the scaling exponents for cities. The size of the effective network is not therefore given by the adjacency matrix of the underlying infrastructure network, but rather by the

typed promise graph.

It is useful to recall the definition of a promise network (see [Bur15a]).

Definition 201 (Promise adjacency matrix)**.** *The directed graph adjacency matrix which records a link if there is a promise of any type τ, and body $b_{ij}(\tau)$ between the labelled agents.*

$$\Pi_{ij}^{(\tau)} = \left.\begin{array}{ll} 1 & \text{iff } A_i \xrightarrow{b_{ij}(\tau)} A_j, \\ 0 & \end{array}\right\} \quad \forall\, b_{ij}(\tau) \neq \emptyset \tag{9.19}$$

The adjacency is the effective topology of the spacetime network, as far as the agents are concerned. The link-occupancy of this matrix, for a given promise type, is a linear sum whose value is generally much lower than that of the total possible mesh of interactions. Thus, for any promise type τ,

$$\sum_{i,j=1}^{N_I} \Pi_{ij}^{(\tau)} = N_\tau(N_\tau - 1) \ll N_I^2, \tag{9.20}$$

Note that an agent can make a promise to itself too, so the upper limit could be written N_I^2.

The value-percolating connectivity or degree of a node

$$\Pi_{ij} = \sum_\tau \Pi_{ij}^{(\tau)}, \tag{9.21}$$

$$k_i \simeq \sum_j \Pi_{ij}. \tag{9.22}$$

We can also write this in terms of the direct valuation of the promises, in terms of the actual matrix of promises π_{ij}[Bur15a]:

$$k_i \simeq \sum_j v_C\left(\pi_{ij}\right). \tag{9.23}$$

where v_C is the value of the promise as calibrated and assessed by a common central agency (see appendix B.2).

Agents can keep multiple promises, or multiple types, 'simultaneously' over a given timescale T, by multiplexing their time at a rate that is much faster, i.e. $\gg 1/T$ to avoid the queueing instability. On the assumption of sufficiently sparse packing:

$$\sum_\tau \sum_{i,j=1}^{N} \Pi_{ij}^{(\tau)} \leq N(N-1). \tag{9.24}$$

To understand the output of a promise network, we care more about the assessments of which promises were kept than the number of promises that were made (see appendix

B.2). Each agent assesses promises individually. and they may not agree. However, to compare to city statistics, we may assume that an statistical bureau agency has been appointed by the city to calibrate these assessments $\alpha_{\text{official}}(\pi_\tau)$ according to a single scale. Promise-keeping is an average over time. Provided the sum time to keep a promise, for all τ, for each agent, is much less than each time interval of the assessments, we can reduce $\alpha(\pi)$ to a frequency 'probability'. Another way of saying this is: provided the cost of keeping the promises is less than the budget of each agent.

These estimates are maximal. The size of a functional cluster is not really related to any of these graphs, because there are semantic constraints. Functional behaviour is a strict limitation, which leads to very sparse graphs. To gauge an average measure of the total economic impact of all functional interactions, we have to assume:

- The functions are successful in driving an economy.
- The density of implicit interactions is quite high, else a given output Y will not be represented by an average mesh density.
- There are some long range interactions that make the partially connected graph totally connected on average, even if only at a low level.

In reality, the city might be partitioned into quite independent regions, leading to a modular reducible form[BBCEM10]. It guided by preconditional bindings between agents. So, if one imagines the network that delivers output Y, it may be some maximally quadratic polynomial of N_I, related to the structure function of the network.

$$\sum_\tau \sum_{i,j=1}^{N} \Pi_{ij}^{(\mathrm{Y})} = \text{Poly}(N_I) \leq N(N-1). \tag{9.25}$$

If i, j run over all the individual agents within city limits, then these matrices are sparse and fragmented for each τ. Only with sufficient diversity of promise types will the sum graph for a city output Y have sufficient connectivity to form a process that generates output algorithmically. The same will be true in IT infrastructure for microservices and library components.

We have to add an assumption of sufficient diversity in the types of promises to our assumptions for a city, so that the average of all of them The deficiencies of certain skills were suggested by [AHF$^+$14] as a reason for lower than expected scaling.

CHAPTER 10

CLASSICAL RELIABILITY THEORY

The fidelity with which agents, i.e. system components, keep their promises is one of the classic subjects of statistical analysis. It's called reliability theory. Although that point of view is quite limited, many books have been written about it, and it's important to establish a connection with that literature and the language of Promise Theory adopted here. In the traditional approach to reliability, a system is considered to be a collection of components with some probability of failure. So-called *structure functions* are used to replace detailed functional semantics with a bare minimum model of dependencies between black box components. Components are assumed either to be working or not working. Reliability is then quantified by an imagined probability that components are independently in a working state.

In this chapter, we look at the basic construction of the classical method, and show how it maps into promise theory, in a simplifying limit. Since we aim to go beyond the classical results, we need to see how its generalization reduces to the same result, with the same assumptions.

10.1 THE LIMIT OF PERFECT COOPERATION

The classical approach to system modelling was constructed out of statistical theory, with a particular aim in mind. The idea was not that component probabilities could be assessed individually for specific systems, but rather that they could all be modelled by common results from generalized Poisson statistics. It is known from manufacturing

processes, that component lifetimes often follow a Poisson distribution, and hence this can be used to make generic predictions about likely time before a failure occurred.

Classical reliability theory is a blunt tool for making broad estimates: all details are lumped together into a single value, represented by the structure function. Strategies for increasing the probability of being in a working state could be addressed by reducing dependency, or increasing redundancy of the components.

10.2 THE ASSUMPTIONS

The reliability of systems of components is traditionally studied through the so-called *structure function*[Nat98, HR94]

$$\phi(x) : x \to \{0, 1\} \tag{10.1}$$

which expresses the variable dependence of components on one another, and an expectation of being in a working state (basically a frequency probability or reliability). When faults occur, its value is reduced, as a component variable goes from 1 to 0. It is computed following the basic laws of probability in series and parallel. Components are independent, and their cooperation is implicitly encoded by the structure function's form. The structure function approach is characterized by:

- Maximal impartiality and suppressed semantics.
- Being generic and unspecific in its assessments.

As a consistency check on this extreme end of the spectrum of interpretation, we shall demonstrate, in this chapter, how a promise theory view reproduces the main results of the classical theory, before going beyond them. This helps to show where the limitations lie.

Comment 25 (How shall we interpret probabilities in reliability theory?)**.** *Probabilities are used in different ways. Here we are using them as estimators for the fraction of samples over which a system is in a working state, i.e. as an approximate scale factor for reliability. The values are not computed or measured, or even guessed normally. One is more interested in how combinations of the probabilities are larger or smaller for the sake understanding how the likelihood of being in a working state is affected by combinatorics.*

10.3 QUANTITATIVE RELIABILITY—TRADITIONAL APPROACH

In the following sections, we repeat some of the standard results about component analysis from reliability theory.

Given how these diagrams resemble flow diagrams for electrical components, it is natural to think of this as implying flow, but this is somewhat misleading. For the serialization, there is a transmission of service from left to right in each image, only insofar as what happens at the right hand side implicitly depends on what happens on the left hand side. However, the components' faults are treated as independent, so the failure of one component does not necessarily transfer additional load to another. They allow for that kind of effect, we need to study the examples more closely. Promise Theory is helpful here because it documents these relationships atomically.

10.3.1 CONDITIONAL PROMISE LAW (DEPENDENCY)

To compute dependencies, we need to recall the law of conditional promises. This law defines what it means for an agent to treat promises made conditionally in a fashion consistent with the realities of what the agent can be responsible for. An agent may only make a promise about its own behaviour.

Dependency enables delegation and specialization, but it also brings potential fragility to any cooperative system. Delegation is a strategy that can be used well or poorly. It is modelled with conditional promises.

The conditional promise law says that a promise, conditional on a dependency, cannot be assessed as a true promise unless the dependency is promised and accepted by the promiser of the conditional promise. Thus, the promise to deliver a package by FedEx:

$$\text{Shop} \xrightarrow{+\text{deliver}|\text{FedEx}} \text{Customer} \tag{10.2}$$

cannot be assessed by the customer without the full construction:

$$\text{Shop} \xrightarrow{+\text{deliver}|\text{FedExService}} \text{Customer} \tag{10.3}$$

$$\text{Shop} \xrightarrow{-\text{FedExService}} \text{Customer} \tag{10.4}$$

$$\text{Shop} \xrightarrow{-\text{FedExService}} \text{FedEx} \tag{10.5}$$

$$\text{FedEx} \xrightarrow{+\text{FedExService}} \text{Shop} \tag{10.6}$$

$$\tag{10.7}$$

i.e. the Shop must not only promise that it will deliver a package, but that it will accept

the services of the FedEx agent in order to do so. This construction forms the basis of conditional chaining of components in systems.

10.3.2 SERIAL DEPENDENCY OF COMPONENTS

In the component structure view, the diagrams are symmetrical and reversible, even when there is an implicit arrow of flow. in a promise formulation, the asymmetry of causal direction is clear. In the serial case (see fig 10.1), agents A AND B have to work, and the probability

$$P(A \textbf{ AND } B) = P(A)P(B|A). \tag{10.8}$$

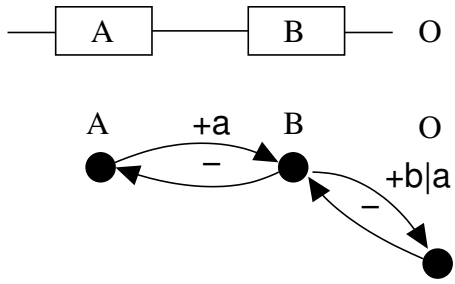


Figure 10.1: Serially dependent components and promises. Note how the causal flow direction is encoded into the promise version but is only assumed in the structure function.

If A and B are independent, this is simply $P(B|A) = P(A)P(B)$. Or in terms of the structure function, using the convention $P(A) = p_A$, etc, we have

$$\phi_{AB} = \phi_A \phi_B \tag{10.9}$$

$$E_{\rm all}(\phi_{AB}) = p_A p_B, \tag{10.10}$$

i.e. the expectation of all parties that this arrangement is in a working state is the product of probabilities of the components being in a working state. The general rule for combination becomes:

$$E_{\rm all}(\phi_{\rm tot}) = \prod_i p_i \tag{10.11}$$

In terms of promises, we have:

$$A \xrightarrow{+a} B \tag{10.12}$$

$$B \xrightarrow{-a} A \tag{10.13}$$

$$B \xrightarrow{+b|a} O \tag{10.14}$$

i.e. A promises $+a$ to B, B promises to use that, and furthermore B promises conditionally b given that is has received a promise of a. And the end-of-chain observer's O's expectation that the promise from B is kept is:

$$E_O(B \xrightarrow{+b|a} O) = P(b|a) = p_B p_A, \tag{10.15}$$

giving an alternative derivation.

10.3.3 REDUNDANT COMPONENTS—ALTERNATIVE HANDLERS

For alternative components (see fig. 10.2), there is redundancy, e.g. pilot and co-pilot, master and backup, etc. Although they are positioned in parallel, they do not necessarily complement one another in terms of performance throughput[156]. Either or both of the components should work, so the reliability frequency 'probabilities' combine:

$$P(A \textbf{ OR } B) = P(A) + P(B) - P(A \textbf{ AND } B) \tag{10.16}$$

The last term only applies if both components are in play, in which case there is some over-counting where the frequencies overlap. Alternatively:

$$\begin{aligned} P(A \textbf{ OR } B) &= \textbf{ NOT } ((\textbf{ NOT } P(A)) \textbf{ AND } (\textbf{ NOT } (P(B))) \\ &= (1 - (1 - p_A)(1 - p_B)) \\ &= p_A + p_B - p_A p_B \\ &\equiv p_A \coprod p_B \\ &\equiv P(A) \coprod P(B). \end{aligned} \tag{10.17}$$

The exclusive expectation that only one and not the other is working is:

$$P(A \textbf{ XOR } B) = P(A) + P(B) - 2P(A \textbf{ AND } B), \tag{10.18}$$

i.e. excluding the overlap region altogether. The promise version of this (see figure) has

$$A \xrightarrow{+a} O \tag{10.19}$$

$$B \xrightarrow{+b} O \tag{10.20}$$

$$O \xrightarrow{-a} A \tag{10.21}$$

$$O \xrightarrow{-b} B \tag{10.22}$$

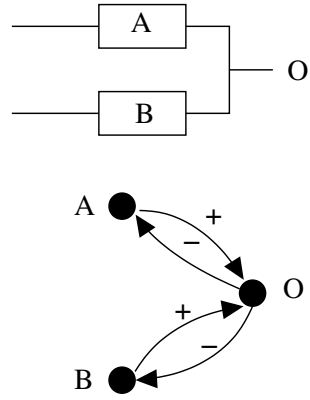


Figure 10.2: Parallel components / promises

The observer's estimate of the reliabilities for promise keeping is:

$$E_O\left(A \xrightarrow{+a} O\right) = p_A \tag{10.23}$$

$$E_O\left(B \xrightarrow{+b} O\right) = p_B \tag{10.24}$$

By treating A and B as a coarse super-agent, we may write:

$$O \xrightarrow{-(a \text{ OR } b)} \{A, B\} \tag{10.25}$$

whence , the expectation of being able to keep the promise

$$E_O(\{A, B\} \xrightarrow{+(p_A \text{ OR } p_B)} O) = p_A + p_B - p_A p_B, \tag{10.26}$$

$$= p_A \coprod p_B. \tag{10.27}$$

and we recover the result in a different way.

10.4 COMBINING DEPENDENCY AND REDUNDANCY

The structure diagrams, with their combination of serial and parallel arrangements tempts us to think about electrical circuits and continuous flow, but this may be misleading. Systems may involve discrete events or continuous operation.

10.4.1 REDUNDANT ARRANGEMENT OF DEPENDENT SERIAL SUB-SYSTEMS

Duplicating whole systems in parallel is a form of redundancy at the high level, e.g. a server and a backup server, or a route and a backup route. Each system may consist of sequences of components, like the dotted boxes in figure 10.3.

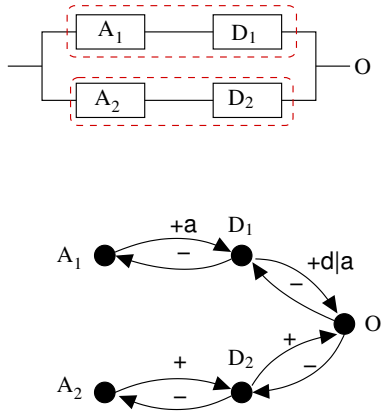


Figure 10.3: Redundant 2-stage serial chains (dotted) in parallel

Taking the simplest example of a system of two serial agents A_1 and dependent series component D_1, in parallel, with a complete clone A_2, D_2, the structure functions or reliabilities have the form:

$$\begin{aligned} \phi &= \phi((A_1 \textbf{ AND } D_1) \textbf{ OR } (A_2 \textbf{ AND } D_2)) \\ &= \phi(A_1)\phi(D_1) \coprod \phi(A_2)\phi(D_2) \end{aligned} \tag{10.28}$$

Defining the reliability expectations:

$$\phi(A_1) = P(A_1) = x_1 \tag{10.29}$$

$$\phi(D_1) = P(D_1) = x_2 \tag{10.30}$$

$$\phi(A_2) = P(A_2) = y_1 \tag{10.31}$$

$$\phi(D_2) = P(D_2) = y_2 \tag{10.32}$$

Here, by analogy with the cases above, the $P(A)$ etc, may be viewed as the probability or reliability fraction with which the agent A keeps its promise. We only need to describe precisely what (and to whom) that promise is. The arrangement of components in the

structure diagram implies to whom the promise is made, and on whom it depends, but it does not matter too much what promise is being kept. This is built into the assumptions of such classical reliability treatments. With that, we can simply say that the total structure function is:

$$\begin{aligned} \phi &= x_1 x_2 \coprod y_1 y_2 && (10.33)\\ &= x_1 x_2 + y_1 y_2 - x_1 y_1 x_2 y_2 && (10.34) \end{aligned}$$

Also, we can supplement the promise semantics to complete the functional description that is absent in the structure viewpoint:

$$\begin{aligned} A_1 &\xrightarrow{+a_1} D_1 && (10.35)\\ D_1 &\xrightarrow{-a_1} A_1 && (10.36)\\ D_1 &\xrightarrow{+d_1|a_1} O && (10.37)\\ O &\xrightarrow{-d_1|a_1} D_1 && (10.38) \end{aligned}$$

Similarly, an analogous set of promises is true for the parallel system A_2, D_2. The two systems come together only at the choke point of the observer:

$$\begin{aligned} O &\xrightarrow{-d_1|a_1} D_1 && (10.39)\\ O &\xrightarrow{-d_2|a_2} D_2 && (10.40) \end{aligned}$$

From which O interprets, by coarse graining:

$$O \xrightarrow{-d_1|a_1 \ \mathbf{OR} \ -d2|a_2} \{D_1, D_2\}. \qquad (10.41)$$

Note how the implicit assumption of aggregation and coarse graining away differences between the branches is documented explicitly in the promise version of this formula. From this, it would infer the reliability of the system to be

$$\begin{aligned} E_{=}\left(\{D_1, D_2\} \xrightarrow{+d_1|a_1 \ \mathbf{OR} \ +d2|a_2} O\right) &= P(d_1|a_1) \coprod P(d_2|a_2)\\ &= x_1 x_2 \coprod y_1 y_2. && (10.42) \end{aligned}$$

This reproduces the result using the promise theory. Once again, the promise viewpoint doesn't alter the structure of the problem. It only adds the functional documentation of intent. This becomes more significant as the promise collaboration graphs become more complicated.

Comment: it is easy to see why there is still room for improvement in this design: the point of system recombination is itself a possible point of failure, with everything riding on a single consumer O.

10.4.2 A SINGLE SYSTEM MADE FROM FULLY PARALLELIZED (REDUNDANT) COMPONENTS

The natural extension to removing a single point of failure is to make each functional component in the system independently redundant (see figure 10.4). This keeps all the failure probabilities independent and one expects the result to be smaller. Indeed, there is a well known theorem that proves this to be the case (see the folk theorem in volume 1).

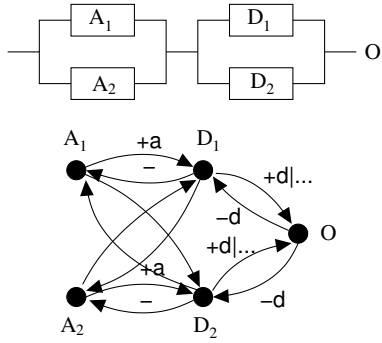


Figure 10.4: Serial chain of parallel components

From the structure function, we now have:

$$\phi = (\phi(A_1) \textbf{ OR } \phi(A_2)) \textbf{ AND } (\phi(D_1) \textbf{ OR } \phi(D_2)) \tag{10.43}$$

Once again. defining the reliability expectations:

$$\begin{aligned} \phi(A_1) &= P(A_1) = x_1 && (10.44)\\ \phi(D_1) &= P(D_1) = x_2 && (10.45)\\ \phi(A_2) &= P(A_2) = y_1 && (10.46)\\ \phi(D_2) &= P(D_2) = y_2 && (10.47) \end{aligned}$$

This becomes simply;

$$\phi = \left(x_1 \coprod y_1\right) \left(x_2 \coprod y_2\right) \tag{10.48}$$

Now let's do it from the promise perspective. The promise diagram takes the form of a

coupled set of agents. From the two row of diagram:

$$A_1 \xrightarrow{+a_1} D_1 \tag{10.49}$$
$$D_1 \xrightarrow{-a_1} A_1 \tag{10.50}$$
$$A_1 \xrightarrow{+a_2} D_2 \tag{10.51}$$
$$D_2 \xrightarrow{-a_2} A_1 \tag{10.52}$$

and the bottom row:

$$A_2 \xrightarrow{+a_2} D_2 \tag{10.53}$$
$$D_2 \xrightarrow{-a_2} A_2 \tag{10.54}$$
$$A_2 \xrightarrow{+a_1} D_1 \tag{10.55}$$
$$D_1 \xrightarrow{-a_1} A_2 \tag{10.56}$$

Now there are two stages of OR aggregation: each of the D_i agents, and finally the observer O. The D_i aggregations take the form:

$$D_1 \xrightarrow{-(a_1 \ \mathbf{OR} \ a_2)} \{A_1, A_2\} \tag{10.57}$$
$$D_2 \xrightarrow{-(a_1 \ \mathbf{OR} \ a_2)} \{A_1, A_2\} \tag{10.58}$$

Thus, D_i promise to O:

$$D_1 \xrightarrow{+d_1|(a_1 \ \mathbf{OR} \ a_2)} O \tag{10.59}$$
$$D_2 \xrightarrow{+d_2|(a_1 \ \mathbf{OR} \ a_2)} O \tag{10.60}$$

By further aggregation, this is observed as:

$$O \xrightarrow{(-d_1|(a_1 \ \mathbf{OR} \ a_2)) \ \mathbf{OR} \ (-d_2|(a_1 \ \mathbf{OR} \ a_2))} \{D_1, D_2\} \tag{10.61}$$

Hence, O evaluates the reliability of this promise, with the coarse graining assumption made explicit once again:

$$E_O\left(\{D_1, D_2\} \xrightarrow{(+d_1|(a_1 \ \mathbf{OR} \ a_2)) \ \mathbf{OR} \ (+d_2|(a_1 \ \mathbf{OR} \ a_2))} O\right)$$
$$= \left(x_2(x_1 \coprod y_1)\right) \coprod \left(y_2(x_1 \coprod y_1)\right) \tag{10.62}$$
$$= (x_1 \coprod y_1)(x_2 \coprod y_2) \tag{10.63}$$

which reproduces the earlier result.

10.5 THE FOLK THEOREM FOR REDUNDANT FAULT TOLERANCE

The redundancy folk theorem[Nat98, Bur04a], of classical reliability theory, states that a single system in which every component is duplicated for redundancy is at least as reliable as duplicated systems offered in parallel (see figure 10.5). This follows from the lemma:

$$\phi(x \coprod y) \geq \phi(x) \coprod \phi(y) \tag{10.64}$$

for any $\phi(\cdot)$. When applied recursively, this purely mathematical inequality implies that low level redundancy is the most reliable approach in terms of fault coverage. The assumptions of this are based entirely on probabilities however.

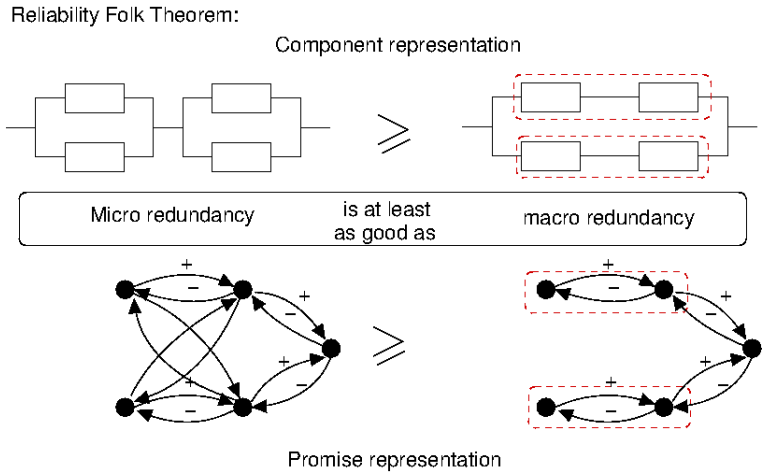


Figure 10.5: Illustration of the redundancy folk theorem.

Although the probabilities satisfy the inequality without prejudice, a single system with every component made redundant might not be the cheapest approach to build. For example, we might not know how to make the system work with that design. Alternatively, the system represented by $\phi(x)$ might be a cheap commodity, making an array of cloned systems cheaper than a single system of higher quality.

By examining the promise graph for low level redundancy, in figure 10.4, we see that the number of promise required to make all necessary interactions complete, exposing all hidden assumptions, is twice the number for the low level redundancy case, thus the

chance of a system keeping all of the design promises is greatly reduced. This works against the 'truth' of the theorem.

In spite of the shortcomings, we can now write the folk-theorem result in a promise notation. For want of a better shorthand, and out of respect for tradition, we could express the promise lists in the simplest analogous form as:

$$E\left(\pi(\langle A_1, O, b_1\rangle \coprod \langle A_2, O, b_2\rangle)\right) \geq E\left(\pi(\langle A_1, O, b_1\rangle) \coprod \pi(\langle A_2, O, b_2\rangle)\right). \tag{10.65}$$

This shows that it is purely as result about structural expectation. We could add to this a cost function $C(\cdot)$, which has the opposite result:

$$C\left(\pi(\langle A_1, O, b_1\rangle \coprod \langle A_2, O, b_2\rangle)\right) \leq C\left(\pi(\langle A_1, O, b_1\rangle) \coprod \pi(\langle A_2, O, b_2\rangle)\right). \tag{10.66}$$

The proof is trivial by counting interaction lines and associating a non-zero cost to each promise. We can add to this the cost of repairing components, or replacing components, which is the same in both cases, unless the components themselves are aggregated into commodity blocks for cost savings.

The lesson from this exercise is that, in any system of significant intricacy, there are no simple rules for optimizing reliability. The statistical theorems are interesting as guidance, but they do not offer a panacea for design choices.

10.6 FAULT TREES

A natural extension of the component approach to reliability is Fault Tree Analysis (FTA), widely used in the nuclear power industry. It suffers from the same limitations as the examples above, but allows for more sophisticated structures. It takes a simple hierarchical view of system dependency, that is quasi-deterministic, and hence assumes sufficient isolation from other effects.

The main purpose of FTA is to enable computation of relative reliability. It fails to model semantics in a way that is related to the structure of the system. To address these concerns, we need to go to a graphical method: promise theory.

10.7 QUEUES AND DETAILED BALANCE

Queues are the balance between arrival processes and service or renewal processes. If we imagine the arrival process is the arrival of faults, then we can measure system

rejuvenation statistically using simple flow rates. This was approach was used to discuss scaling in [BC04]. This approach can give some indication about timescales and fault tolerance limits, using the queueing instability[Bur04a].

10.8 WHAT IS THE LIMIT OF PERFECT COOPERATION?

The promise representation has the ability to represent more aspects of a cooperative system of component parts than the structure function approach. However, both give the same result in the limit in which promise relationships are mutually established with 100 percent certainty. The limit implies that:

1. Both sides agree to cooperate.

2. Cooperation implies propagation of influence with unit probability.

We can now work on generalizations that stray from this limit. In general, this means that the likelihood of a working system will be estimated lower than the classical limit, but the important part is not the numbers (which are only estimates), but rather how the scaling of probability is affected by different issues with greater specificity.

10.9 WHY NOT LOGIC AND RULES?

In Western culture, we have a strong belief in rules, so much so that it pervades our thinking. We stop at red lights, and walk at the green man. We speak of 'laws of nature' as if the world were following a rigid plan; but, when we say 'exception to the rule', we really mean 'exception to the norm'. Do we stop at stop lights because we have to, or because it's a good idea to comply with conventions?

The problem with this trust in rules and obligations is that they distort our understanding of reality. Systems do not have to comply with our expectations. It's the other way around: we try to build systems that are stable enough to allow us to make rules by observing their normal behaviour. But then, we come to believe that what is normal, in one context, is actually a generally applicable rule, or even 'truth'. This is where flaws begin. Imposing rules and logical assertions onto systems is an interesting exercise for the purpose of testing, but it does little to promote understanding. Turning this upside down, a promise viewpoint replaces rule and obligation with likelihood and promise, to describe our design understanding as best effort.

10.10 SHORTCOMINGS OF CLASSICAL RELIABILITY THEORY

Classical reliability theory was addressed to the description of machine components in which the fixed meanings of components were assumed to be correctly composed to deliver a workable outcome, for simple networks. The only question remaining was: does any component stop playing its part? Consequently, it was never intended to encompass human issues, qualities, or to represent changing systems.

In any statistical theory, the assumptions and semantics are constant in order to have parity when measuring data. The functional behaviour of a system must therefore be constant too, unambiguous and inevitable. The problem with this view is that functional utility only makes sense relative to certain contexts or circumstances. In modern systems, humans and machines interact in integrated systems and a purely statistical theory is insufficient to describe system reliability. This suggests, too, that probability as an estimator of likelihood is not the right interpretation, but probability as a scale factor for relative weight in a cooperation is.

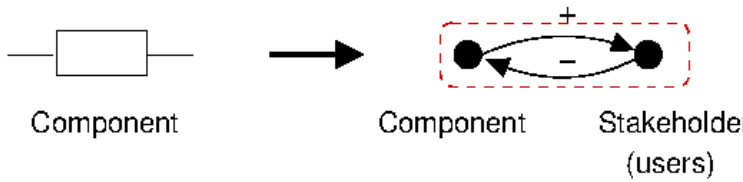


Figure 10.6: The shift from traditional statistical reliability theory to a promise view. Promises represent a shift attention from standalone components to interactions with users.

Let's qualify some of the shortcomings of the traditional approach. Using the structure function to represent the system behaviour has limited expressive power:

1. **It focuses on components as lumped 'roles' instead of the integrity and appropriateness of their interactions**. No component does anything by itself. Thus even if all components are working, if they fail to interact properly, the system will not work. Every interaction with uni-directional propagation of influence involves two promises; one to provide and one to accept. Thus, any dependence can fail in one of two different ways:

   - By sender/provider/source promise not kept (+).
   - By receive/accepter/sink promise not kept (-).

All transmission of intent, every act of cooperation, involves two independent agencies. The two parts are independent promises, located at independent agencies, but they form dependent failure modes: the receiver cannot receive if the sender has not sent, so it is dependent. This is not accounted for in the basic fault theory, where it is assumed that systems work by imposition / obligation.

2. **Perfect cooperation without situational awareness**. Because it lumps everything into single probabilities, combined deterministically, it effectively assumes that cooperation and adaptation of agents plays no role in its effective functioning. The agents play their roles without question or self-awareness. This could be appropriate for 'dumb agents', such as manufactured components, whose behaviour does not depend on their perception or awareness of their situation, but it is not adequate for human-machine collaborations.

3. **Limits and tolerances**. The structure function does not easily account for what happens when the sum of impositions on a component exceeds the component's capacity to perform. Capacity limits are not included in the simple expressions for structure, but they are a key part of information theory and transmission of influence.

   When a limit is exceeded, one of two possibilities must take place:

   - A queue must build up at sender or receiver.
   - Certain impositions or promises to use must be abandoned.

4. **Scale anomalies**. As workload is scaled to higher levels, we want to understand how the fixed scale limitations of individual components impacts the total system. Bottlenecks could be mitigated by employing parallel processing, not merely redundancy, but this causes a potential confusion between parallelism and fault redundancy, which serve different purposes.

Fortunately, the results of the previous chapter can easily be improved on by developing the promise approach, while keeping the simplicity of the method.

10.11 SUMMARY

In this chapter, we've used the classical approach of treating components as black boxes. They are assumed to work (or not) with a certain probability. We do not model *how* they work together, only that they are connected through dependencies and with redundant alternatives. This offers some rough statistical insights about the ability to repair parts within the design.

Although we need approximate methods for summing up statistical reliabilities at fixed semantics, we also need to be able to ask deeper questions related to functional design and changing semantics. What if the semantics of a system suddenly changed, while it continued to function? That would also be a fault, in which semantics no longer represented intent. These are the issues that relate to human involvement in systems, and thus represent a crucial perspective, from the design of machines as proxies for our intentions, to our own interactions with them.

CHAPTER 11

RECOVERY, REPAIR, REPLACEMENT, RESILIENCE

When agents interact, they may not be prepared for everything that will happen. To begin with, they may find themselves in a state in which they are indisposed and unable to respond on the appropriate timescale—whether for reasons of inadequate resources, interior faults, flaws, and so on. This makes their full range of outcomes uncertain. Beyond this basic level of uncertainty, there may also be events that cannot even be promised. These are especially aggregious, especially when they are catastrophic, lead to the a failure to keep promises, and even the destruction of agents.

The possible weaknesses of systems' promises were discussed under the heading of the three F's: faults, flaws, and errors. The complementary topic of how one primes or restores a system to a condition of intended functionality is the subject of this chapter.

Promises help us to clarify matters here too. Promises document intent—the degree of semantic alignment between agents taking part in processes, and when we assess the counts of successful outcomes, these document patterns of *past* dynamics. Promise Theory adds to traditional descriptions the implication that agents are never merely passive components—driven by the imposition of exterior forces—they must, in fact, have autonomous interior processes in order to respond. This has some surprising implications for matters of design responsibility, as discussed earlier. The interior 'heartbeats' of agent processes can transform, absorb, and even initiate outcomes. This is not the usual point of view in process modelling, and yet it has important consequences for how we design systems, in order to be stable and resilient, recoverable and self-sustaining.

11.1 AGENT READINESS—EXPECTATION AND INTENT TO PREPARE

The 'R' which is more usually discussed in the case of catastrophe is 'risk'. Risk is a heuristic assessment of the possible impact of an already expected outcome, assumed to be unwanted and hopefully unlikely. It cannot be the assessment of an unexpected outcome: by definition, we can't assess what can't be predicted[157]. Of course, some events (like natural disasters or force majeure) could be predicted in principle, but we don't bother in practice due to their low rate of occurrence[158]. Promise Theory offers a simple insight into the success of system processes: preparedness (in the form of prior promises) leads to greater certainty about behavioural outcomes, and thus a better understanding of changes and desired outcomes than the more traditional and mercurial idea of imposition. Imposition, as we recall, offers no chance for an agent to prepare, and thus the outcome of impositions is more uncertain than that of promised interactions. Promises signal the intent to prepare for a constrained or more programmed outcome.

Expected or unexpected, does it make a difference? What do we mean by that? What are the semantics of being prepared? When agents have accepted a promise, about the behaviour of another agent, we may assume that they are better informed, and primed to be ready for what is going to happen—at least to the extent that their interior and autonomous capabilities admit. A promise acts as an 'invitation' to prepare. When something is imposed onto agents, they are—by definition—unprepared, and may have to muster a response in real time.

In section 9.2, we remarked that drifting of agents' promised intentions may lead to misalignment. Another way of saying this is that promise drift leads to unfulfilled expectations. This is an *active* rather than a passive use of expectation, e.g. I expect you to wash your hands. It implies imposition of an obligation. In the binding,

$$S \xrightarrow{+b_S} R \tag{11.1}$$

$$R \xrightarrow{-b_R} S \tag{11.2}$$

we know that the effective transfer is $b_S \cap b_R \geq \emptyset$. But, what if b_S gradually drifts, or is reduced in scope, so that R has the intent b_R to use much more than S is offering? In that case, the mere promise to use b_R might eventually become a demand for more than S is willing to promise. Then the (-) promise effectively transforms into a (-) imposition, i.e. a *requirement* or a *demand*. The imposition of usage, by demand, can lead to stress. In a human interaction this may even lead to hostilities.

Example 259 (Free software promises)**.** *Imagine some free software, with a license of usage 'as is'. It makes no particular promises, but a user assesses that it solves a*

problem that fulfills a particular need—say, to convert pounds into kilograms. Over time, the software provider deprecates that particular functionality, or moves it somewhere inconvenient, so that the user no longer assesses the software to keep its apparent promise. The user then sues that company for damages.

Traditionally, statistics is the empirical field of study for coarse or average behaviours. Statistics concerns the frequency with which outcomes fall into certain anticipated classes. Statistics are related to probabilities for future outcomes in the special case of steady state behaviours, i.e. when one expects the regular patterns observed over past history to remain the same in the future. The patterns may be stochastic rather than deterministic, but we need a steady state. This leads to the characteristics of ordinary behaviour. Part of that ordinary behaviour might include random failures, such as in the Poisson processes of chapter 10.

11.2 TALES OF THE UNEXPECTED

A more interesting, and critical question, is what happens to a system when it is influenced *unexpectedly* by an external agent? At such a moment, probabilities are of no use whatsoever, because the conditions under which they promise to provide an accurate picture of expected behaviour have been violated. We speak of *anomalies*. The lack of a quantitative scale can be distressing to those of us coached in the quantitative sciences—who med (mis)led to believe that anything worth describing would be described by numbers, because 'numbers don't lie'. It's typically in so-called 'complex systems' that unexpected conditions occur. The reason is that complex systems generally have too many possible outcomes to include in the scope of a model. They have 'power law' behaviour, i.e. behaviours with long tails that cover almost unlimited possibilities, albeit with tiny and incalculable probabilities.

In physics, the concept of a perturbation or disturbance is used as simple model for defining the response of a system to external influences—whether expected or unexpected. As long as we can define the impact, we should be able to define the response, based on the promises we know about the system. Of course, if we are not sure what promises have been made, this in intractable—which is hopefully encouragement to know our systems well. This is particularly important for the discussion of stability. Let's try to generalize the purely quantitative notion of an influence from physics to a the language of Promise Theory.

Definition 202 (Perturbation)**.** *The imposition of a request by an agent S for response by an agent R:*

$$S \xrightarrow{\pm\Delta_S} \blacksquare \quad R \tag{11.3}$$

In other words this is analogous to a 'demand' for change.

The response to a perturbation may be said to be a causal consequence, by virtue of the conditional dependence on it:

Definition 203 (Response)**.** *A conditional promise made after a prerequisite imposition to provide it:*

$$R \xrightarrow{-(\pm\Delta_S)} S \tag{11.4}$$

$$R \xrightarrow{+\Delta_R \mid \Delta_S} S. \tag{11.5}$$

See also the response function of section 6.4.13.

Notice how this definition of seems backwards in the language of promise theory. The initial imposition of Δ_S has not power to affect change; the response Δ_R is given voluntarily but in response. It was unexpected, and thus interrupts whatever interior plans or processes were ongoing. In classical physics, a perturbation is an external force—in the classical model, the mere imposition is enough to guarantee a deterministic response, though in quantum physics that is no longer true. In a locally autonomous picture, an external influence can only yield an effective force if accepted: it depends on the actual response of the affected agent. We say this is voluntary, as a matter of definition.

The state of the agent is an internal matter, but it affects the ability of the agent to sample, receive, and send information, as part of a Shannon information channel. We can define an agent to be prepared if its acceptance of perturbations precedes their arrival:

Definition 204 (Preparedness)**.** *An agent R which makes a promise to accept a promised or imposed change Δ from an agent S is prepared with respect to the promise body and agent pair $\langle \Delta, S \rangle$*

$$R \xrightarrow{-\Delta} S. \tag{11.6}$$

Lemma 43 (Expected and unexpected events)**.** *Impositions that arrive before acceptance are unexpected. All other impositions are expected.*

Thus, when a perturbation is received, we are interested in the ability of an agent to 'handle' it, which might mean to make use of it, or simply to ignore it. Perturbations

might be not be material to the operation of a system, e.g. a dent in a vehicle doesn't affect its ability to transport things:

Definition 205 (Ignorable perturbation)**.** *A perturbation whose effect is within a tolerance band of acceptable values, and leads to no conditional response.*

Perturbations, which are ignored, have no causal impact on the system's principal promises. The type of ignorable promises can be regarded as a 'symmetry' of the system. Such symmetries play a role in the meaning of resilience for systems[Coc19].

11.3 PROMISE CONTINUITY UNDER PERTURBATION

The ability to continue promising and keeping promises is central to what we mean by robustness and resilience. What's missing from most accounts of resilience are reliable definitions that can be applied across a range of contexts. Some informal clues to the meaning may be found in [HWL06]. Let's try to make some clearer statements about the various R's.

Note that, by the basic axioms of Promise Theory, we can only really speak about the continuity of a promise binding, because both (+) and (-) promises can be unkept—and a promise that is not used for binding is of no interest, as it has no causal effect[159]. After a fault, which leads to an interruption of promise-keeping, continuity can be restored by a number of strategies. Some common words used in this regard include the following.

Definition 206 (Recovery)**.** *A process by which a system moves from a fault state to a non-fault state, i.e. all agents in the system of interest move to a state in which their exterior promises are kept.*

Recovery may involve repair or replacement of parts.

Definition 207 (Repair)**.** *A process by which an agent performs recovery or replacement of interior agents and promises in order to recover a promise-keeping state in its exterior promises.*

Definition 208 (Replacement)**.** *A process by which an agent R, which depends on a promise π from an agent S_1, accepts an equivalent promise π from an alternative agent S_2. a dependency, so that reliance on S_1 is exchanged for a reliance on S_2.*

Some engineers talk about 'hot replacements' in which redundant services are available all the time as a matter of redundancy—as opposed to the case where a system must be taken offline to perform a 'cold replacement'.

An understanding of the avoidance of these courses of action leads us to define a notion of continuity under conditions that are either internal or external to an agent, on any given scale. Like all promise concepts, the definitions of resilience and robustness are relative to a context in which they find themselves. Agents cannot be intrinsically robust or resilient, because the perturbations and continuity they experience are—by definition—assessed through interactions with other agents.

Definition 209 (Robustness)**.** *An assessment about the capacity of an agent or superagent to absorb, tolerate, and survive perturbations imposed or promised by interior or exterior agents. Robustness, as defined, concerns an agent's invulnerability to change.*

Robustness implies that an *agent* is sturdy and constant in its purpose, keeping its integrity and its promises, without having to rely on exterior agents.

Going beyond mere survival, we may speak of *resilience of a process* that involves the processes that maintain agents and their promises. Resilience is an assessment of stability and continuity in the qualitative and quantitative details of promises kept as part of the an on-going process. This assumes more than a mere tolerance of perturbations by a particular agent—but the distinction is subtle and perhaps not very crucial.

A resilient system is one that continues to keep certain promises, even in the presence of failures and perhaps even flaws. Faulty dependencies can be bolstered by redundancy or repair, the envelope of possibility might be wide enough to tolerate even simple design flaws. There may be many possible failure modes that test different aspects of resilience.

Definition 210 (Resilience)**.** *When a promise π is invariant under perturbations: i.e. agent and promise may be called resilient if they can withstand, absorb, or recover from a perturbation before another agent can observe a change.*

Resilience implies both dynamical and semantic stability, i.e. continuity by dynamical means, regardless of perturbations imposed or promised by interior or exterior agents on which A depends conditionally for its own promises. Resilience is an assessment concerning the behaviour of an agent or superagent A that envelops a process—a property of processes rather than of static agents. It implies invariance over a specific critical timescale. Like robustness, resilience can be compromised by both interior failures and exterior perturbations.

11.4 PROMISE DISCONTINUITY, RISK, AND IMPACT

When there is an intermission in promise keeping, observed by some reliant agent A, which accepts and makes use (-) of the offers (+) from source agents S within a system, it will fail to satisfy its function unless it can be repaired before an agent makes use of it, or the loss is negligible because redundant agents can keep the promise instead. This tells us that there are two causes of risk: the fidelity of agents in keeping their promises, and the dependence on other agents (and thus their fidelity). Undocumented environmental conditions affecting promise keeping may be absorbed into the fidelity of the agent by the downstream principle. An agent A that relies on an agent S to promise X to O, is responsible for accepting the promise from S, as well as supplementing it with its own:

$$A \xrightarrow{+X|D} O \tag{11.7}$$

$$A \xrightarrow{-D} S \tag{11.8}$$

Because the *possibility* of a promise not being kept (regardless of any assessment of probability) depends on these factors, the risk of an impact on promise keeping propagates and accumulates, somewhat like eigenvector centrality and our model of trust (see [BB06]).

The assumption of continuity of agents' operating conditions is (perhaps regrettably) almost a default assumption in system design, even in the most testing of circumstances—that's partly because we can't predict the unexpected. The traditions of reliability (see chapter 10), and the assessment of risk, attempt to forge a link between the assessment of risk and evidential probabilities. However, probability is a useless tool for risk assessment, because it is merely a characterization of what *can* be expected, based on past evidence. No one can predict the unexpected. What we can do instead is to assume the worst and consider the cost of avoidance or repair.

A simple scale by which to assess risk is thus to use an estimated cost avoiding, repairing, replacing, a broken system—a measure of the impact on the system. This is more embedded in the context of each system proper, as it makes use of costs and efforts that are known from its interior processes. Measures like Mean Time To Repair (MTTR) are only a surrogate for that greater impact of the loss of a function—which still remains to be characterized. Models such as Poisson probability distributions of component failure are of little use once a failure occurs.

Principle 10 (Risk assessment)**.** *Assess and plan for promise discontinuity by considering the processes (promises, timescales, and costs) required to remedy faults.*

The central aim in design of a functional system is to avoid catastrophes (in the mathematical sense), i.e. suddenly discontinuities in promised behaviour. A system's

response needn't be technically 'smooth', in the sense of calculus, but it should be of a proportionate magnitude, and on the normal and within the characteristic timescales of its regular promises in order to absorb the event. The possible timescales for perturbations should therefore be studied and, by Nyquist's law, one should seek to maintain interior processes on a timescale of at least twice as fast as that.

This consideration of risk is another reason why the principle of separation of scales is important. The ability for systems to absorb and tolerate perturbations is therefore the key to resilience. Tolerance is related to aggregation: the ability to transduce a fast timescale into a slower one.

11.5 SCALING RESILIENCE IN PROMISE NETWORKS

Resilience is thus the ability to absorb and tolerate perturbations, whether they can be predicted or not. A realistic attempt at this requires some effort to understand the context and environment in which systems intend to keep their promises.

System design principles for scaling promises—like serialization, parallelization, centralization, decentralization, etc—have a generic impact on system resilience. When we scale up systems from the bottom up, using agents in a promise-oriented model, the bulk properties of systems look very much like the properties of matter in material science. The characteristics of a sheet of metal is not directly related to the characteristics of the atoms of which it's comprised. If we apply this to what we mean by resilience, the same observation applies. There is an abundance of ideas in popular literature about what constitutes 'resilience' for different kinds of systems and agents. Some authors use arguments to try to discount certain kinds of agents (e.g. humans or machines) as being unsuitable for certain kinds of task—as we discussed under the topic of agent fidelity (see section 9.6). While such considerations might be warranted on pragmatism in special cases of immediate urgency, this feels both unjustified and irrelevant to the bigger picture.

We can begin discussing these issues from standalone atoms, building up through molecules, to chains, and multi-dimensional crystals and fabrics. All if these structures are in use in technological and human systems today.

Example 260 (Computational pipelines)**.** *Flow charts or pipelines are like circuitry in which messages are exchanged. Each connection between components, with specialized tasks, can be represented as promises or impositions. Each link assumes a causal direction for progress, from input to output. This model is a facsimile of a process, which is why flowcharts are so popular for simple programming and algorithm design. In a pipeline, several components might feed into a single component. How do we know if it*

will be able to cope with the load implicit in this? What promises and impositions enable this to work reliably?

11.5.1 SCALING AND RESILIENCE

The scaling of promise keeping is naturally related to the question of resilience (see chapter 7). A system that can always keep its promises is resilient by any measure.

Some authors have tried to argue against intrinsic stability, as this may lead to inflexibility or a lack of agility, rendering systems unable to respond to challenges that are unforeseen. Resilience certainly has to do with timescales. If a system is quick on its feet, so to speak, then it can correct in realtime to avoid catastrophe.

Interior transparency is not necessarily sufficient to determine failure modes but it might be helpful or harmful, depending on context. The intrinsic avoidance of failures under perturbations depends on stability, not on transparency. Transparency could expose weaknesses for malicious agents to exploit. Our best shot for stability is to keep systems constrained to linear or sublinear (convergent) behaviours. Our best strategy for speed of response is the be able to resume normal promised function within a timescale that is equal or less than the timescale over which perturbations arise.

Lemma 44 (Resilience as scale invariance)**.** *A system which is resilient to perturbations of any size must be be independent of the size of perturbations, and therefore must be scale free.*

Since no system is truly scale invariant (even if properties can be extended indefinitely, the cost of running the system must rise, so we can only ever achieve partial scale invariance by ignoring certain issues), there is no fully resilient system.

Example 261 (From hardware to software)**.** *Software engineers sometimes argue that by moving from hardware design to software, using only standardized and generic Personal Computer hardware, they remove the shackles of hardware allowing any problem to be solved. This is incorrect, because it ignores the intrinsic scales that are implicit in the generic hardware dependencies. We can never remove a dependence on scale, we can only trade one scale variance for another. By introducing a giant computational cloud, one can eliminate a certain restriction on capacity, but only at the cost of additional latency, power consumption, and interaction. Scale invariance is an incomplete characterization, and thus so is resilience.*

11.5.2 BULK PROPERTIES OF MATERIAL FABRICS

Our characterizations of systems are usually at a high level. Classical component analysis would like to reduce these characterizations to concrete failure probabilities based on

construction, but experience suggests that people pay little attention to such detailed levels of characterization. Instead, we hand wave using effective characterizations on a macroscopic scale. The analogy discussed in [Bur13a] is to compare to the engineering we carry out based on bulk materials, with their macroscopic properties and failure modes.

We can define a few bulk material properties, by analogy with material science [Gor68]. Having 'concrete' definitions (sorry) does help to be more precise about the properties we are trying to express.

Definition 211 (Brittleness or rigidity)**.** *A property of promises and promise bindings, implying the inability for a promise binding to be modified by continuous or plastic deformation, i.e. given any promise*

$$S \xrightarrow{b} R, \tag{11.9}$$

all changes of the form $b \to b + \delta b$ result in $b \to \emptyset$, i.e. the collapse of the promise, and hence a promise binding.

Rigidity builds on the notion of co-dependence, introduced in chapter 4.5. A body is rigid when the interior clocked processes of all the parts move in lockstep. This rigidity makes them brittle over every interval of time, because a change is spread to every sub-agent on the interior of the body at the 'same (exterior) time'. Failures therefore appear sudden to external observers. For this reason, brittle materials can normally only change by such catastrophic failure modes, including crack propagation, in which chains of promises are broken like a zipper unravelling. A common synonym for brittleness is:

Definition 212 (Fragility)**.** *A less formal synonym for brittleness, usually applied to systems.*

In response to fragility the following term was coined by Taleb [Tal12].

Example 262 (Antifragility)**.** *This term has been used to express the capacity for systems to learn and adapt to perturbations, e.g. by reinforcement, growth, or redundancy or remembering configurations like memory foam.*

The counterpoint to brittleness is plasticity.

Definition 213 (Plasticity)**.** *This is a property of promise bindings, reflecting the ability for the equilibrium point of offer and acceptance promises to be altered in a continuous manner, in response to some applied influence. For example, given:*

$$S \xrightarrow{b} R, \tag{11.10}$$

a third party T

$$T \xrightarrow{+\delta b} S \tag{11.11}$$

$$S \xrightarrow{-\delta b} R, \tag{11.12}$$

or simply S *itself causes* S *to alter the parameters for* b*:*

$$S \xrightarrow{b+\delta b \mid \delta b} R \tag{11.13}$$

and R *admits sufficient tolerance of this to accommodate the change:*

$$R \xrightarrow{-(b \pm \delta b)} S. \tag{11.14}$$

Plasticity requires the cooperation and tolerance of both S *and* R *with the influence of the third party* T*.*

The corresponding resistances or costs of brittle and plastic changes are described by:

Definition 214 (Stiffness)**.** *Resistance to (or cost of) plastic deformation.*

For example, resistance might be the result of there being a multitude of agents that need to be influenced (resistance proportional to the number). Or it could be related to the capacity to keep the promise (it might take longer as b increases), etc.

Definition 215 (Strength)**.** *Resistance to brittle failure.*

The size of δb needed to break a promise binding describes the tolerance of the binding to increase or decrease in the overlap between sender and receiver of a promise binding. Compressive strength and tensile strength are two different measures, in material science, corresponding to the ability to increase or decrease the quantitative measure of a promise.

Plastic deformation is usually an irreversible change, but if there is a restoring influence (as in the case of restorative maintenance processes) which maintains a detailed balance, then we can speak of elasticity:

Definition 216 (Elasticity)**.** *The ability to recover from plastic deformation.*

There are more properties we might describe for bulk materials formed from superagent collaborative structures.

11.5.3 RIGID BODIES AND FRAGILITY

Superagents that are strongly coupled are rigid. The model of entanglement or co-dependence, described in section 4.5, is key here. When agents depend on one another mutually:

$$A_1 \xrightarrow{+b_1(b_2) \mid b_2} A_2 \tag{11.15}$$

$$A_2 \xrightarrow{+b_2(b_1) \mid b_1} A_1, \tag{11.16}$$

their promises are bound to change in lockstep on the timescale of the interior clock of the entanglement promises. This is also the exterior time between the two agents, and is thus always slower than the interior time of either agent. For agents exterior to the composite agent $A_1 \oplus A_2$, changes to both are instantaneous.

Now, suppose that an exterior boundary agent imposes a change in either b_1 or b_2, assuming that the agents would accept such a change:

$$S \xrightarrow{+\delta b}\blacksquare\ A_1 \tag{11.17}$$

$$A_1 \xrightarrow{-b} S \tag{11.18}$$

$$A_1 \xrightarrow{b_1 \to b_1+\delta b \mid \delta b} S, \tag{11.19}$$

then now both agents' promises are shifted by the same amount due to their rigid co-dependence:

$$A_1 \xrightarrow{+b_1+\delta b \mid b_2+\delta b} A_2 \tag{11.20}$$

$$A_2 \xrightarrow{+b_2+\delta b \mid b_1+\delta b} A_1. \tag{11.21}$$

Rigidity means that bodies whose components are entangled or co-dependent, with respect to a property b, in this way, effective move as a single entity, with all parts co-moving in a shared state of unison, with respect to perturbations δb. This needn't be true for all perturbations. For example, hitting a ball with a cue might induce it to move as an entity, but spraying it with blue paint only affects the outside edge.

11.5.4 STRESS AND STRAIN IN PROMISE NETWORKS

We can generalize the idea of a stress, as used in material physics, to accommodate more general promises, both qualitative and quantitative. If a load is an invited perturbation, i.e. one for which a provider is properly dimensioned, then a stress is an uninvited perturbation, i.e. a 'demand' imposed on an agent:

Definition 217 (Stress)**.** *The imposition of a request for a change Δ, by an agent R onto a potential provider S:*

$$R \xrightarrow{-\Delta_R}\blacksquare\ S \tag{11.22}$$

The response to a stress may be a deformation of a promise, which (by analogy) we can call a strain

Definition 218 (Strain)**.** *The response to an imposed request for a change Δ, accepted by an agent R onto a potential provider S:*

$$S \xrightarrow{-(-\Delta_R \cap \Delta_S)} \blacksquare \quad R \tag{11.23}$$

$$= S \xrightarrow{+\Delta_S} R. \tag{11.24}$$

We can give more quantitative substance to these measures by using the model of a queue as a processor of discrete transactions. Strain increases the as the waiting time increases, or the rate of processing decreases. Strain reaches a maximum when the response time R tends to infinity: Stress increases as the arrival rate grows:

$$\text{Strain: } \sigma \sim R \simeq \frac{1}{\mu - \lambda} \tag{11.25}$$

$$\text{Stress: } \epsilon \sim \lambda \tag{11.26}$$

From Little's Law (see volume I, chapter 12), we know that, if the queue length or work queue $L = \langle n \rangle$, and $T_W = R$ is the waiting or response time, for an arrival process with average rate λ and average service rate μ (and $\rho = \lambda/\mu$) are given, then

$$L = \lambda T_W. \tag{11.27}$$

Using the fact that, for an M/M queue, we have

$$T_W = R = \frac{\langle n \rangle}{\lambda} = \frac{1}{\mu - \lambda} \tag{11.28}$$

$$L = \langle n \rangle = \frac{\rho}{1 - \rho}, \tag{11.29}$$

then we can write

$$\rho = (\mu R)^{-1} L, \tag{11.30}$$

whence comparing to the Young's law of stress and strain, where σ is the stress (or tensile pressure), ϵ is the strain (or relative extension), and E is a constant called Young's modulus,

$$\sigma = E\epsilon, \tag{11.31}$$

we can make the analogous identification,

$$\sigma_{\text{eff}} \leftrightarrow \rho \tag{11.32}$$

$$E_{\text{eff}} \leftrightarrow (\mu R)^{-1} \tag{11.33}$$

$$\epsilon_{\text{eff}} \leftrightarrow L. \tag{11.34}$$

This has an intuitive feel to it: the length of the queue is the measurable strain, and the traffic intensity is the pressure under which the strain is felt. The modulus of proportionality is a constant of order unity related to the average server efficiency.

An impulse or shock is certainly an imposition. Sudden impacts are impositions, while expected loads may constitute promises. On the other hand, statistically stable populations of impositions can be predictable too on an average level, so the relevant part of signalling intent by promises lies not so much in whether the intent at the source was announced in advance or not, but rather whether the receiver of a load is prepared or unprepared to promise a response.

11.5.5 STRESS CONCENTRATIONS IN WORKFLOWS

The model of workflow, in which a succession of promises forms a dependency chain or 'circuit' can be analyzed simply as a queueing system: i.e. an arrival based model of discrete events. In bulk aggregations of agents, a key issue is how promises are kept by superagents, where the sharing of a burden occurs on a larger scale. For example, a bulk material (like steel) is formed from structures that contain may atoms working together; biological tissue is made from many cells sharing certain burdens in parallel. The influences that are imposed at an external boundary are not all channelled through a single agent on a small scale, like a funnel, but are shared somehow between all of the parts that make up the whole. How this sharing takes place is a key scaling issue that includes both qualitative (semantic) and quantitative (dynamical) aspects.

Two network structures are particularly significant in the scaling of workflows (see figure 11.1):

- Places where promises fan out to a number of (either mutually exclusive or coexisting) alternatives, sometimes called a load sharing or balancing structure.
- Places where promises come together like a funnel that aggregates dependencies at a single agent location.

We can call these converging or diverging lenses in the promise graph, because they either focus or defocus the trajectory of outcomes in the workflow. From the perspective of stress and strain, a fanning out offers relief from stress, while a convergence funnel leads to a stress concentration (a bottleneck). The latter often leads to fragility along interfaces and boundaries, especially where the promises are brittle.

We can analyze these situations, on a superficial level, as another application of the replication scenario discussed in chapter 3, and reproduced here for convenience in figure 11.2. This describes the basic semantics of the connections. Similarly, we can use the simple dimensional scale arguments of queuing theory to estimate the stresses

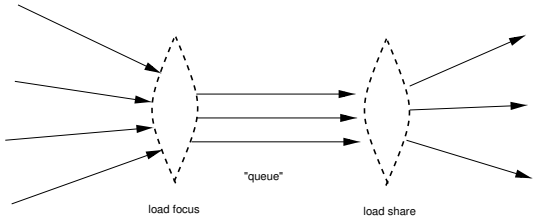


Figure 11.1: Focusing signals from space, trading space and time by queueing and load sharing.

and strains. The essence of bulk promise-sharing is easily encapsulated in the queueing model (see volume 1 of this treatise) for m servers, e.g. the $M/M/m$ queue. The reason is straightforward: a queue is a processing pipeline for discrete countable events and may be addressed as a continuum approximation too in order to deal with scale. The connection between these lies in the choice is a distribution, such as a Markov model. The model consists of potentially many stages, serially one after the other; however, these fall into three main roles, shown in figure 11.2. A source stage forms the initial boundary condition, where causation begins its trajectories. The replica stage acts as a transmission line if the number of parallel agents is constant. It can also act as a load balancer for the next layer if there is a fanning out to a larger number of agents that can share the burden of dependency.

We can quantify this relationship in a flow approximation using queueing theory (see volume I). If the total stress on the left if λ, and there are m agents to share the load (in the manner of an $M/M/m$ queue, then the acceptable strain is reflected in the response times of the servers, each of which receives a share λ/n. As we know, from the queueing instability, critical behaviour ensues as λ approaches $\lambda/\mu \geq m$ from below.

The picture extolled in these simple dimensional estimators can be extended to apply to form chains and other crystal lattice structures (see section 11.5.5). In practice the quantitative estimators are of less importance than the semantic considerations built into the specific promises. That's because queueing systems have to be operated far away from their critical region of brittle failure or collapse in order to be effective and maintain the assumption of 'flow'.

Our best chance of avoiding brittle shattering is *fault tolerance*, which can absorb cracks before they propagate. This is like putting impurities into metals to disperse stress concentrations.

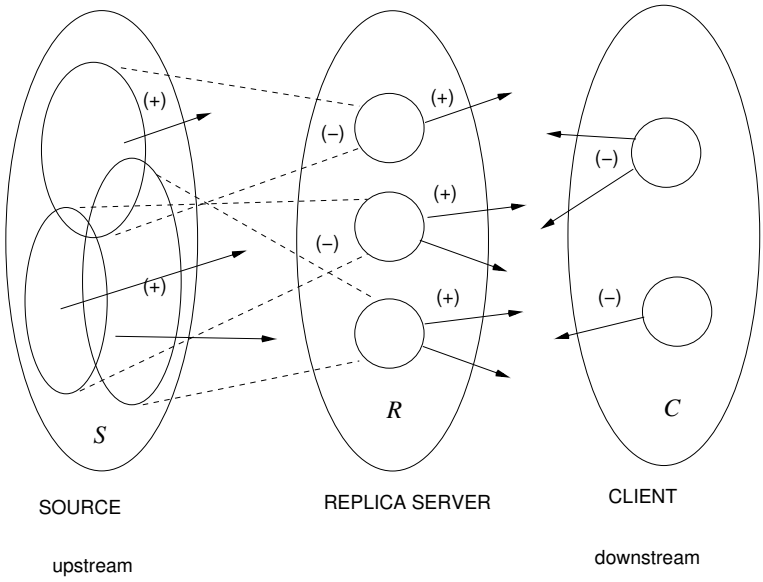


Figure 11.2: The basic configuration for sampling data through intermediaries.

11.5.6 LOAD-BEARING STRUCTURES

Resilience of infrastructure is often based on a large scale cooperation of smaller components. Material science is a perfect example of this, which illustrates the scaling principle—namely that superagents can make promises and form composites of their own at each scale. Figure 11.4 shows the classical depiction of the beginning of crack formation in a crystalline material. The absence of a single agent reduces the the number of agents on an intermediate later for load sharing from left to right. If the stresses are shared evenly by passing on the additional burden to the remaining agents, this might be enough to tip one agent after the other over the edge of what it is able to promise, leading to a domino cascade failure.

Example 263 (Transduction layers)**.** *Drivers and transducers that try to bring a kind of uniformity to the variety of non-standardized devices and software interfaces are amongst the most common places for bugs in software, because the lack of clear or standard promises about the format of the data means that drivers and transducers are often guessing, or 'screen scraping'. This is has dubious semantic stability.*

Interaction is what makes systems non-trivial. Interactions may be cooperative,

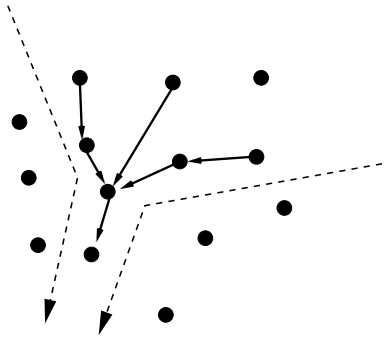


Figure 11.3: Queueing funnels may lead to signal congestion and delivery delays.

non-cooperative, or even competitive. Many faults arise along boundaries, because boundaries are, by definition, non-redundant. Cooperative structures may themselves act as dependencies at the foundations of systems, providing the glue that holds large scale cooperation together.

Example 264 (Infrastructure as latent dependency layers)**.** *Utilities such as electricity, water, and Internet are needed by every home and business, but they are generally considered to lie outside of the zone of interest for those entities. Similarly, in chemistry atoms bound by ionic and covalent bonds play a role in forming molecules and crystals that promise bulk properties. In an assessment of stability, the integrity of those structures may be key to keeping load-bearing promises.*

Example 265 (Network fabrics)**.** *The history of redundant networks goes back to the beginning of the Internet. The Internet architecture was, itself, designed to avoid single points of failure typified by hierarchical tree-like organizations. So-called Clos networks, designed for non-blocking telephony described patterns of material resilience, for forwarding conversations at a structural level[Clo53]. These networks have been rediscovered in connection with routing fabrics for large datacentres [Ano14, NLK13].*

The basic idea is shown in figure 11.5. By doubling agents and their connections in a mesh, redundancy of pathways is secured. Single failure can be disguised by a failover, and traffic can be balanced to avoid saturation of any particular node, provided the total throughput capacity is not exceeded. The number of agents in the layers can be varied; it needn't be one to one, as shown in figure 11.6.

The effect of combing multiply redundant layers is to introduce a material robustness to a entire structure, which is not promised by any single component. The individual

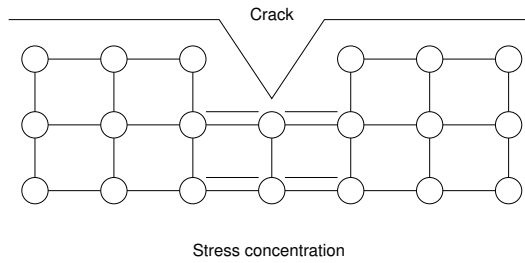


Figure 11.4: A load bearing structure with a stress concentration due to a broken or missing promise. A direct comparison with a discrete promise-based system is beyond the scope of this document, but the roles are easy to identify. The depression, around the missing agent, acts as a funnel for stress applied from left to right. Each vertical column of agents acts as a load-sharing interface and server battery for the next column. A crack implies fewer agents to transmit the same stress leading to cascade failure above a threshold, analogous to the queueing instability.

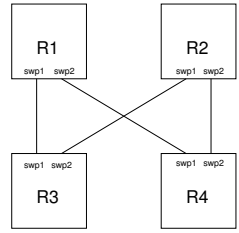


Figure 11.5: 2x2 Clos pattern.

components, combined with the way their promises distribute the load together can bring a resilience to the structure—potentially increasing the total capacity, or merely ensuring complete stability. The traditional way of drawing the networks is in a Cartesian embedding, with crossed wires, shown in figure 5.6. Although compelling to certain engineers from an algorithmic viewpoint, figure 11.7 is an abomination from a symmetry viewpoint. Crossed wires leads to a tangled topology that makes datacentre organization extremely complex, error prone, and difficult to maintain. The cost of cabling is such network installations dominates the costs of the infrastructure, and cable failures are major debugging issues. Using the simplest of transformations, we can unpick the tangled geometry to expose its natural form. The first step is shown in figure 11.8 Applying this separation throughout, for the 2x2 network, leads to an disentangled form, shown in figure 11.9.

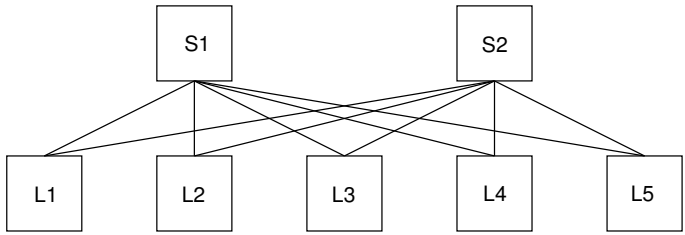


Figure 11.6: 2x5 Clos pattern.

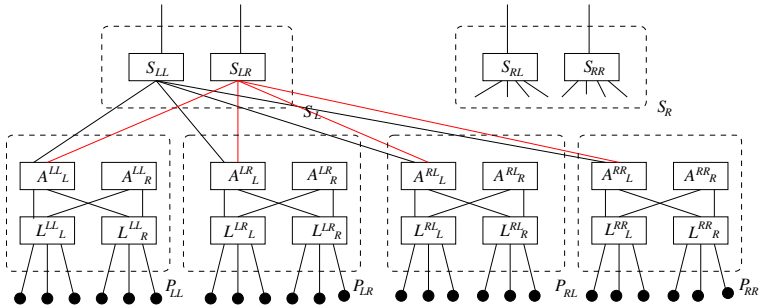


Figure 11.7: A redundant left-right 2x2 Clos network, shown in a normal top-down geometry.

Finally, adopting a radial geometry, as shown in figure 11.10 allows the square geometry to be turned into a fully line-of-sight network built with straight lines. In a future vision of networking, these channels might even be replaced by direct laser pathways, avoiding expensive cabling altogether. The key importance of the Clos structures is that—by distributing the promises of the network amongst a number of independent agents, with their causally independent interior processes—one avoids any single point of failure amongst the components. The result is a super-component on a larger scale. As we know from the redundancy folk-theorems, such low level redundancy is always preferable to higher level failover mechanisms, say at the level of services. On a different level, the two are merely scaled versions of one another, so the distinction may well be moot when implemented throughout.

11.5.7 SERIAL REPAIR NETWORKS (EPIDEMICS)

When a system depends on a serial dependency, cascading failures may have to be repaired by fixing one stage at a time to bridge access to the next fault. It's tempting to

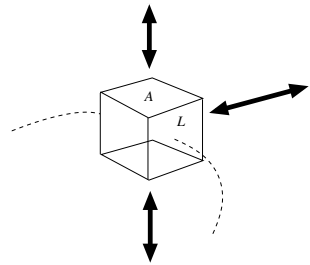


Figure 11.8: Exposing the symmetries of the 2x2 Close network to form a natural radial structure.

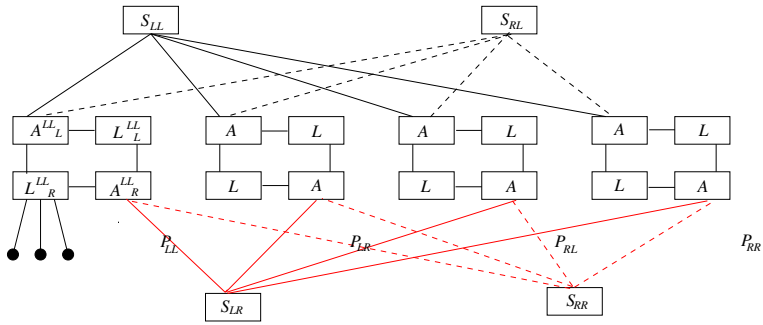


Figure 11.9: The redundant 2x2 Clos network unpacked into an extra dimension (from left-right to up-down) by symmetry.

use the model of network flow to think about the problem, but we should also realize that influence propagates through networks even without the delivery of some kind of package. Fixing holes in a road, for instance, has this structure because you have to fix the first hole to reach the next, and so on. In other words, change can propagate like a wave, such as when a disease spreads. The interaction between semantics and dynamics is complicated here[160] and involves (as usual) the ratio of interior agent process timescales. Where repairs need to be carried out, from within a network, spanning multiple agents, then this takes on a diffusion like character.

The example above of the Clos network fabrics offers an interesting case study, as these networks are both staged (critically dependent) and yet have interior redundancy. This leads to a particular kind of robustness, without necessarily bringing resilience. Redundancy can only go so far in keeping promises. If redundancy is only to avoid immediate failure then it must transfer stress to back up routes. If those routes are also

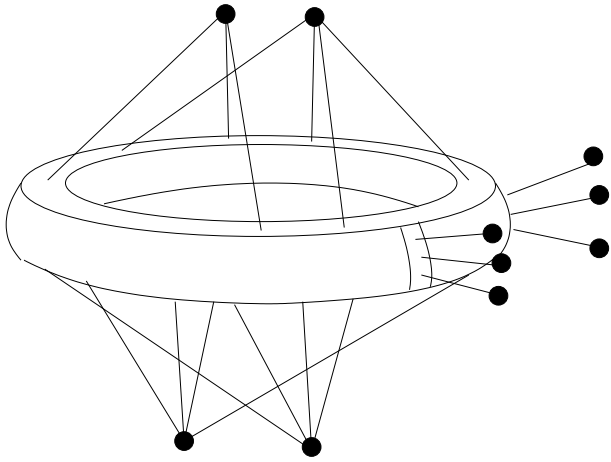


Figure 11.10: A radial embedding geometry for the Clos network shown in figure5.6.

used in normal production, alongside routes that have failed, the redundancy doesn't protect against stress concentrations, and crack formation. Rather it performs a kind of stabilization by entropy of mixing—rendering agents effectively indistinguishable from one another by subordinating them to a bulk promise. This is a key role of subordination: agents voluntarily give up their distinctive freedoms to become 'just another brick in the wall'.

11.6 RECOVERY BY RENEWAL PROCESSES

Prevention of fault states is a difficult task, even for a stable and predictable system. Seeking a quick recovery from fault states is generally a more viable strategy for system continuity, in the long run, since this will be necessary at some time anyway.

Law 5 (Busy and stable is predictable)**.** *Detection of anomalous behaviours is haphazard except in the busiest and most stable systems. Only busy systems have statistical stability to define the normal envelope of behaviour. Only stable busy systems are regular enough to define anomalies cleanly.*

A predictable norm makes anomaly detection of faults easiest. Sometimes a system that is only quasi-stable or weakly unstable can operate from intervention to intervention, but it may never reach a steady state. This is much how a jet fighter works—by continual

renewal or recovery. The continual process of intervention might be called countervention or the principle of equilibrium by detailed balance. This is how we handle chaotic behaviours like unpredictable weather events, and so on.

Example 266 (The fighter and the passenger jet)**.** *A passenger jet promises to fly you in a straight line from start to end, without any daring acrobatics. It is designed to fly in a stable configuration, quite static. There are few changes of intended course during a flight, so it can afford intrinsic stability against its main threat: the weather. A fighter jet, on the other hand, expects threats to come much faster from attack. It needs to be much less stable in order to be able to adapt more quickly. Fighter jets are constantly struggling to escape from losing control by adapting very quickly to a changing flight envelope. Too much stability would be a threat to them. This continuous adaptation is highly expensive, requiring computer control to act quickly, but it enables maneuvers that would be impossible for a passenger plane.*

Realistically, recovery is always a better strategy than prevention, because we need recovery anyway. So, if we design systems to be intrinsically recoverable rather than being intrinsically stable, we can fend off vulnerabilities both dynamically and semantically as long as we don't allow too large an envelope of possible states to wander about in.

11.6.1 AGILITY FOR AVOIDANCE AND RECOVERY

The question of timescales is—as usual—central to understanding and maintaining systems, whether human, machine, or hybrid. Prevention of stress and ultimately faults is only possible if one knows exactly what the desired (zero strain) state of a system is at all times—i.e. when systems are essentially fixed or very slowly varying in their intent. A focus on strict compliance with static goals leads to brittle systems, impeding flexibility and adaptation in the fact of contextual challenges.

Example 267 (Why kids don't fall on the ice)**.** *If an athlete can dance on her feet, she can stay upright even if the ground moves beneath her. When kids run in the snow, with careless abandon, they are constantly unstable in a mode of rapid continuous correction. If an old person walks rigidly on the ice, stiff and defensive, a sudden unexpected slip means he'll break something, because he is unable to react and correct quickly.*

Definition 219 (Agility)**.** *The ability to adapt a promise body to accommodate changing circumstances in 'realtime', i.e. on the same timescale as perturbations from the environment.*

11.6.2 CAN AGENTS BE REPAIRED OR REPLACED TO IMPROVE RELIABILITY?

A traditional argument for eliminating errors from a system, especially human errors, is to replace human workers with machinery, or to transform humans into procedural machines. As already mentioned, the culture of incomplete system design tends to result in the diagnosis of 'human error', when we have been too lazy to design fault tolerances and contingency promises. This suggests that simply replacing humans with machines might reduce errors of execution, increase speed and thus perhaps reduce faults, but cannot address design flaws, which embrace both of the latter.

Paradoxically, the more we deploy automation within human-computer systems, the more skilled human involvement we generally need to plan and guide the collaborative processes, and interpret the contextual learning. Humans, on the other hand, are certainly good at learning and improving the coverage of promises, as a work in progress, and thus they have a natural skill within a system for *continuous improvement*. This points to the fallacy of looking at the nature of the agents themselves, rather than focusing on whether or not they keep their promises. The most unlikely agents can sometimes do the best job[161].

11.6.3 PREVENTION IS PERFECTION, BUT REPAIR IS REALISTIC

We cannot prevent faults and errors from occurring, especially in regard to complex and even conflicting semantics, such as when obligations and impositions are expected to dominate. In modern software systems, for example, there are just too many conflicting semantics in play, making mismatches of intent likely. In industrialization, we work hard to eschew the semantics so that only mechanical repeatable behaviour can be maintained in a steady state, but this kind of inflexibility only works for very limited circumstances. One aims for separation of concerns to allow sufficient isolation to enable this stable situation.

More generally, our best hope then is to be able to repair systems quickly enough that they will lead to catastrophic outcomes. This can be supplemented with the design of safe failure modes—i.e. states which are themselves temporarily stable, like a holding pattern, while analysis and recovery can be performed.

Lemma 45 (Quick repair is indistinguishable from avoidance)**.** *The consumer of a promise that samples its dependency on a timescale of once every t seconds, cannot detect a failure and repair fluctuation that occurs on a timescale much faster than this.*

This follows from Nyquist's theorem, or from the Maintenance Theorem discussed in volume 1[Bur03]. Quick repair is the only way to handle unpredictable outcomes,

because prevention assumes that you know how the failure is going to occur. What you can't foresee, you can't promise to prevent. Moreover, repair can be automated by using fixed point convergence, to perform at a high enough speed.

This might not just be about keeping the existing set of promises, because that set might not be an adequate envelope for containing the system's behaviours.

11.6.4 MAINTENANCE BY DETAILED BALANCE

Stability is a statistical characterization that takes time to reach equilibrium. It's a form of 'learning' with respect to environmental perturbations and boundary conditions. Equilibrium, on the other hand, is a steady state property which requires the effective isolation of the system over timescales for the 'relaxation' of the system to a steady state. The issue of scales—in particular timescales—is never far from system behaviour. This is not the story one learns in computer science, or in human management, yet this is the central lesson of this treatise.

The more traditional dogma in system design is that of 'rules based' order, based on command transactions, which leads practitioners erroneously down the path of impositions and deontic reasoning—known to be flawed[Bur05b]. This works in highly isolated circumstances, where there is an effective firewall against environmental uncertainty, but it leads to lazy and presumptuous thinking. Transactional rules alone are inadequate to explain outcomes in systems (see section 11.6.5). Systems may gravitate towards *fixed points* (see sections 11.6.9 and 11.6.10) by design, and may respond to boundary conditions or information that arrives from the exterior of a system[162].

It serves as a useful reminder to the principles discussed as part of the Maintenance Theorem in volume 1 of this treatise. That theorem describes how discrete transition systems can be repaired by either reversible transactions under complete causal isolation, or by convergence to a policy defined fixed point if repairs can be made at the Nyquist rate. Because there is a competition between agents in a scaled system, and because choices are discrete, the strategies essentially form an array of alternatives that can be mixed like a strategic game, or a type II theory (see volume I)[Bur04a]. The essence of continuity is a balance between perturbations and course corrections or state repairs. This takes the form of a game between faults versus maintainers, e.g. perhaps a kind of Tit for Tat tournament of episodes that hopefully lead to long term stability. The language of game theory helps to look at this problem through a formal lens (see volume 1), as does the language of stability under Nyquist sampling.

Example 268 (Steady course in stormy weather)**.** *To keep the same semantics, in a changing environment, we have to adapt.*

Example 269 (System upgrades, releases, bugs and fixes)**.** *Some motivations for system design upgrades include:*

- *Adjustments to promises due for fitness for purpose*
- *Adjustments to implementation algorithms for keeping promises.*
- *New promises to increase the coverage of maintenance and feedback.*

11.6.5 TRANSACTIONAL 'ATOMIC' CHANGE AND LOCKING

The ability to turn back time by undoing the processes that brought about the current state of affairs is a compelling one that computer science tries to engineer by the technique of *critical sections*, *monitors* or *isolation kernels*. You can always try to retrace your steps—to undo the changes that led to the current configuration of time. Then you are faced with the confusing idea of whether you are going forwards in time to undo stuff, or whether that means that time is going backwards.

Example 270 (Undo)**.** *If you wanted to go back in time, and you hadn't taken the precaution of memorizing every snapshot in your cosmic database, you could try to reverse or undo all the differential changes across the entire universe since a certain calendar time (hit CTRL-Z repeatedly on your life computer and see what happens). On a computer, the 'undo' operation is achieved by remembering all of the deterministic and relative actions and reversing them, within a protected environment.*

Example 271 (Reversibility in physics)**.** *In physics, we believe we have this information compressed in a form of memory about processes that we call 'Equations Of Motion' that form part of the 'laws of physics'. Many of these rules are reversible, as discussed earlier, so we can use them in reverse to predict backwards—in principle. It's not simply a case of throwing the universe into reverse gear of course—to do that you would need precise knowledge of every subatomic process in full detail, and have the technology and force to reverse them all deterministically. When you finally undid everything, you would have to set every particle on its trajectory again (except for yourself and your fellow time travellers), and then continue in a forward direction again as before.*

One way to achieve closed conditions of temporary isolation and thus metastability is to build a transactional model. The essential feature this enables is *reversibility*, or the 'undo' operation. To define transactional integrity we need to discuss how intended change and unintended system change can be tracked and recorded in order to maintain complete information about a system's states over time. We shall distinguish a the

concept of a *journal* of transactions (for intended change) from that of a history (which includes actual changes intended and unintended).

To model intended versus actual change, we introduce the notion of a journal, inspired by the notion of journaling in filesystems. A journal is a documentation of changes applied to a system intentionally, noting and remembering that—in real systems— this can be different from what actually takes place.

Definition 220 (Journal $\hat{J}$)**.** *A complete, ordered sequence of all input symbols passed to an automaton α^* from an initial time t_i to a final time t_f is called the automaton's journal $\hat{J} = (\alpha^*, t_i, t_f)$. Each symbol α corresponds to a change in system state $\delta_\alpha q$. A journal has a scope that is known to the user or process that writes the journal. A journal change δJ involves adding or removing symbols in α to J, and adjusting the times.*

A sequence of transactions, forming a journal, can itself behave as a transaction, as long as there is sufficient isolation from external environment. One way to achieve this is to apply Shannon's error correction theorem, or the Maintenance Theorem as a sequence of desired end-states (see section 11.6.10). Shannon's theorem states that one can always achieve and error free outcome, with long enough time to apply repairs of coding. Each transaction becomes an effective symbol in a *robust alphabet* of change. This requires an absolutely brittle process, with no deviations from determinism, and total isolation. This is both the benefit and the weakness of transaction design in information systems.

To understand imperative and transactional change, we start with a boundary condition, which may be called the snapshot state of a system Ψ_0. We then impose a series of changes Δ_i to this as a sequence. Let S be a system agent (or superagent) and M be a management agent. Viewed as a series of promises made between these two, we start with the initial snapshot state:

$$S \xrightarrow{+\Psi_0} M \tag{11.35}$$

$$S \xrightarrow{-\Delta_i} M \quad \forall i \tag{11.36}$$

$$M \xrightarrow{+\Delta_1} \blacksquare\, S \tag{11.37}$$

$$M \xrightarrow{+\Delta_2} \blacksquare\, S \tag{11.38}$$

$$\ldots$$

Notice that S has to promise to accept arbitrary changes Δ_i in advance, making itself vulnerable to possible attack. If something goes wrong, we have to recreate the snapshot state in order to recover, and the order of the transactions has to be promised by M. This is the 'congruence' approach[Tra02]. In the convergent rollforward approach, the transactions are turned into a set of conditional promises that can only be applied with

prerequisites in place (this adds a kind of transactional safety in reverse: instead of relying on a snapshot state, it relies on a desired outcome).

An improvement on this haphazard trusting approach is to verify each change imposed, by communicating back assessments of the changes $\alpha(\Delta_)i$, and making successive impositions dependent on the confirmation, to invoke a protocol of prerequisites:

$$S \xrightarrow{+\Psi_0} M \tag{11.39}$$

$$S \xrightarrow{-\Delta_i} M \quad \forall i \tag{11.40}$$

$$M \xrightarrow{+\Delta_1|\alpha(\Delta_1)} S \tag{11.41}$$

$$S \xrightarrow{+\alpha(\Delta_1)|\Delta_1} M \tag{11.42}$$

$$M \xrightarrow{+\Delta_2|\alpha(\Delta_1)} S \tag{11.43}$$

$$S \xrightarrow{+\alpha(\Delta_2)|\Delta_2} M \tag{11.44}$$

$\ldots$

It's usual to define transactions in terms of atomicity and consistency, but this inevitably means referring to a specific process which is non-scalable. Here we can define the concepts more simply using invariance of promises, at any scale:

Definition 221 (Transaction at scale T)**.** *A transaction is the promise of an invariant sequence of messages $M_1, M_2, \ldots, M_T$, of length/number T, accepted by a process agent A, whose memory of the messages is also invariant over the sequence, and contains all the data needed to keep the conditional promise*

$$A \xrightarrow{+X|M_1,M_2,\ldots,M_T} \tag{11.45}$$

In other words, the agent A doesn't let go of the information from its cache until it is acknowledged by the receiver. With this definition, we do not presuppose any model or scale for the meaning of a transaction. As long as the transacting agent is invariant over the completion of its promised task, and the data require no dependencies.

The virtue of this definition is to make such transactions repeatable, as all the conditions of the transaction are self-contained, and thus invariant. Put another way, transactions turn messages into scalable autonomous (super)agents, without exterior dependencies beyond their promised scale T.

The resources are effectively blocked or locked with mutual exclusion. Failures on a large enough scale can still wipe out all the information of the transaction, but this adds some assurance of invariance if the data survive the transaction.

Lemma 46 (Transactions are repeatable)**.** *Any valid transaction at scale T leads to a repeatable process, given the same message and conditional promise.*

Notice that the process is only memoryless if $T = 1$, i.e. we choose a particular scale, but the all important invariance is scale independent. Also note that it's important to distinguish between events and transactions, which many authors fail to do. The invariant properties of transactions are not shared by arbitrary messages, so favouring a transactional system is not the same as favouring a message or event driven system.

The model of integral transactions was introduced to try to establish control and certainty over change, by means of artificially induced isolation. It is widely used for financial exchanges, and database writes. Transactional thinking is very common in information technology. A transaction is a promise of locally deterministic change that relies on two main conditions:

- Exclusive process isolation for an interval of time.
- Atomicity of changes, i.e. all change happens within the interval of isolation.

The scaling of transactional systems often additionally relies on a specific ordering of changes, i.e. serialization of changes, as is usual in databases. This effectively breaks transactional semantics.

Lemma 47 (Composition of transactions)**.** *The composition of transactions is not a transaction unless they are supplemented with the conditions for a transaction on the scale of the sequence.*

This is a straightforward consequence of exterior composition, since transactions assume constant exterior conditions. Once the conditions for being on the inside of a transaction are ceded, that promise of invariance is given up.

None of these conditions are generally valid in a distributed system[163], making errors due to incorrect assumptions rife.

Example 272 (Transactional security)**.** *There are certain 'consistency protocols' (e.g. Paxos, Raft, Zookeeper, Chubby, etc) that claim to handle transactional changes consistently in distributed systems, but these apply typically only to very simple data values not to distributed state or interactions. Moreover, they have a finite response time to implement, and they are not fault tolerant. The assumption of process isolation is precarious and fragile.*

We tend to think that banking systems require this kind of consistency, although banks and credit card companies regularly handle fault tolerance in transactions with buffer pools, as this is an innate part of dealing with unpredictable requests. The transactional nature only helps to generate a narrative about who did what first, often as a basis for charging fees and fines.

Example 273 (Maintenance windows)**.** *Planned downtime, maintenance windows, or repairs on the ground, enable transactional isolation by taking system parts out of service for planned maintenance. This might result in taking down the whole system, or systems might rely on reduced capacity or 'hot wiring' a bypass (e.g. in heart surgery) to work around the issue.*

11.6.6 ROLLOUT AND ROLLBACK

Planned change goes through a decision lifecycle. In a distributed system, the moment of committing to the change is sometimes called the rolling out of the change. To distinguish them from rigorously implemented local transactional systems, we define two concepts that respect widespread beliefs about change at the level of systems:

Definition 222 (System rollout)**.** *A distributed intentional change Δ implemented in a quasi-transactional manner. This may or may not involve the suspension of a system operations.*

Definition 223 (System rollback)**.** *A distributed change Δ^{-1}, relative to an intentional change Δ, implemented in a quasi-transactional manner. This may or may not involve the suspension of a system operations.*

Transactional rollback is the inverse of an atomic 'transaction commit' operation, i.e. it is supposedly an atomic 'undo' operation[164].

The concepts defined above mimic the notion of transactional commit and rollback in locally isolated monitors, such as computer supervisors and database engines, but the usage does not satisfy the isolation requirements for proper transactional behaviour.

The popularity of transactional thinking has led to a similar popularity of rollback thinking: when a mistake or unforeseen error is committed, one simply needs to roll back to the previous state. A formal proof has been given to show that this is impossible in general[BC11].

Law 6 (Rollback is unreliable)**.** *Rollback cannot be promised in an open system. Congruences that may contain unknown symbols cannot be undone.*

Desired end state approaches are the natural solution to this issue.

Example 274 (Version control reversals)**.** *Version control systems use the same basic technique of approximate isolation of state with content branches, though it often goes awry, due to the realities of imperfect isolation. This is because changes have dependent consequences, arising from sampling the faulty states during the non-atomic changes. We cannot control time is a distributed system.*

11.6.7 STAGING, TESTING, AND SAFETY NETS

The staging of changes, to produce phased rollout of pseudo-transactions, is essentially another term for pipelining or workflow with a few transactional constraints added. Refer to figure 11.2. Phased deployments of change are sometimes assumed to bring a greater resilience to a working system, by minimizing the risk of losing system integrity. There is no theoretical reason why this should be true.

A phased rollout is simply the imposition of a change as a number of transactional stages. That means the conditions for transactional security are applied to each stage instead of to the whole process. This could minimize the scope of the change, or it might lead to partially consistent states. In other words, limiting the scope of 'rollouts' can limit the risk of unintended side-effects, but it cannot necessarily predict all of them. By making a deployment of change into a transaction, only the affected states are protected, not states that depend on the changes.

Problems may occur only when a certain scale is reached. Phase rollout effectively limits the amplification of possible faults by limiting $|R|$ in equation 6.135, see the discussion of gain in section 6.5.1. However, what one cannot see from the instantaneous response/gain is the network effect of extended fault propagation.

Transactions are often treated as a panacea—as a way to transform change into do-undo sequences. This is naive at best. Even if changes are made transactionally, the unintended consequences may only be visible over an extended time, long after the transaction has completed—requiring a broader plan for clean-up and recovery. Transaction locking does not apply to all state in a distributed system, because there is no monitor of all resources (kernel, hypervisor, etc) that has jurisdiction over everything.

The arguments for transactions are thus of limited value in systems with dependencies. Such effects can only be estimated by understanding the timescales relating to dynamics of the system; this is difficult to predict when systems reach a high degree of complexity, such as near instability.. During that time interval, intermediate states can arise preventing the possibility of a rollback, even if it is still possible to recreated an approximation to transactional behaviour[BC11].

Example 275 (Push-imposition and rollback)**.** *A change is made to a firewall accidentally and erroneously shutting it down. A computer gets infected. The change is rolled back so the firewall is replaced, but the server is already infected, hence the fault persists.*

There is an obligation to remove your shoes before entering houses in Japan, Norway, etc. If you are unaware, you might unwittingly make a perceived error in following this imposition. To correct the error, you remove your shoes, but now you have to clean up the dirt you already walked into the house. Thus a rollback to the approved state does not undo the problems associated with erroneous push.

By limiting the scope, one increases the chances of finding instabilities, errors, and provoking faults in the larger system while still maintaining a partially working system but this is not guaranteed, as critical instabilities could still be provoked. See section 11.6.7.

Example 276 (Strategies for software deployment)**.** *Some common observations about software systems are worth mentioning:*

- *Fault tolerance is always a preferred option for system integrity, absorbing cracks before they propagate. Making rigid systems with brittle conditions that have to be met only leads to states of non-compliance that bring down the ability to promise the system.*

- *When depending on inputs, validation testing of input for tolerance or rejection is a strategy for avoidance of fault states, though a rejection may be perceived as a different kind of fault state. Testing is the asking of the question: are all promises kept? Without clear promises, we cannot test a system except by unwarranted expectations. An observer can impose their wishes onto a system, but (as we know) this is futile. Systems don't often adapt to the wishes of users on a short timescale. In software, for instance, testing covers various issues related to promises:*

  - *Semantic tests (acceptance of promises).*
  - *Dynamics tests (execution and scale tests for promise-keeping).*
  - *Stress tests: Monte Carlo search methods, using 'chaotic' conditions, induced failures etc.*

- *When parallel services can be offered, A/B testing (of version A alongside version B) is a useful way of assessing the response of the system to live perturbations under real conditions.*

- *Faults that have already propagated to other agents cannot be undone without significant cleanup, and perhaps never. The time-window in which a semi-complete transaction allows partial state to propagate is a highly significant scale for live systems.*

11.6.8 INTENDED CHANGE BY PUSH AND PULL

As a brief addendum, we can comment on promise and imposition as 'push' and 'pull' protocols. Changes that are *pushed* from a single agent are in an unknown state by the

instigator until each agent being imposed upon reports its state. There is no global atomicity, and no plausible rollback (imposing state through a noisy channel with crosstalk is a low fidelity operation).

Changes that are *pulled* by a set of agents always maintain a known distributed state by the instigators. There is limited local atomicity, and limited chance of repair in case of local error. The distributed consequences of a correctly implemented local change might still be incorrect at a larger scale. In this case, the effect is emergent and cannot be rolled back.

11.6.9 CONVERGENT REPAIR AND 'ROLLFORWARD'

A more reliable approach to a predictable outcome, in any circumstances is to reject the notion of 'undo' operations altogether. Such operations are intrinsically fragile, but they can be replaced by something more robust. Convergent repair was discussed in chapter of volume 1, section 10.4. One way to understand it is as an integrated test-observe-repair loop, sometimes called continuous delivery in software circles.

Consider an agent that gets its stability from another agent. This applies to many software systems, and it applies to thermal and chemical equilibria. Let S be the source (or reservoir) of 'desired state', and the receiver R be the agent that seeks to maintain its continuity. The receiver accepts to subordinate its state to the directives promised by S:

$$R \xrightarrow{-\psi_S} S, \tag{11.46}$$

and R promises to report its state to S

$$R \xrightarrow{+\psi_R} S \tag{11.47}$$

$$S \xrightarrow{-\psi_R} R. \tag{11.48}$$

As long as the state is within acceptable limits Ψ, S makes no impositions to interfere with R, but if the state deviates, then it imposes a strict target:

$$S \xrightarrow{\emptyset \mid (\psi_R \in \Psi)} R \tag{11.49}$$

$$S \xrightarrow{+\psi_S \mid (\psi_R \notin \Psi)} R. \tag{11.50}$$

Notice that this is not a relative correction, but an absolute reset. The scale of the reset is not important, though the assumption is that it can be made invisibly to agents that depend on R in some way so as to not cause disruption.

Example 277 (Convergent operator resets)**.** *The approach used by CFEngine[Bur98, Bur04c, Bur03] was to apply this method independently to every microscopic atomic configuration property of a computer system, by regular sampling and repair of state, thus*

minimizing changes over their expected timescale. The assumption here was that these minimal interventions would go unnoticed by other processes maintaining continuous compliance. The uncertainty with this lies in knowing whether the system might actually spend small amounts of time in non-compliance states over the time it takes to converge to Ψ_S.

Example 278 (Congruent rebuild resets)**.** *The CFEngine approach was criticized by Traugott[Tra02] who argued for a maximal reset on the scale of a computer. The computer was inactive while being re-imaged and rebuilt, attempting to ensure that the system was never in a non-compliant state after fault detection and retirement. This disadvantage of this approach is that it mandates complete interruption of all processes on the interior of the computer.*

Example 279 (Container resets)**.** *The congruent approach was adopted for container technologies, like Docker, on a more granular level of a single process and its dependencies, by a redefinition of what 'atoms' could be changed for the era of cloud computing.*

These three examples are all equivalent methods, but applied on different scales. The smallest scale used by CFEngine was one of minimal intervention and fastest repair. The maximal scale, used in the congruence of whole machines, was one of firewalling an entire set of related processes. The container scale is a compromise between the two.

The congruent or imperative command-based approach, with blocked state execution, is like the use of *mutex locks* in programming. Ideally one tries not to block execution for a long time, else other processes will be affected. The advantage of applying the method to microscopic or atomic changes is that each state can be defined in approximate isolation along with a very simple method for repair. As soon as the combinatorics of atomic transactions becomes significant, the complexity of computation grows. This problem is known to be NP and PSPACE complete[SC07, BK07].

Many programmers and human managers still believe that state changes only occur in systems when they make intentional changes. This neglects faults, environmental and contextual changes, and so on. The result is that many software systems and businesses are unprepared for the unexpected.

Comment 26 (Change is not only intended)**.** *During forensic analysis of problems, it is often assumed that changes occur in a system by deliberate intentional action. This is not always the case. Both intended and intended changes can be instigated by agents. Moreover, changed that come from environmental impositions can also affect systems, particularly when they have not been designed to maintain their state in the face of such changes.*

11.6.10 REPEATABILITY AND FIXED POINTS

The method in the previous section builds on the properties of mathematical property of 'fixed points' of transition maps. The goal for functional systems, as tools, is to strive for repeatability and predictability. Repeatability is not assured simply by repeating chains of impositions or commands from a common starting state. This is because the composition itself might be affected by unmodelled and unrecognized influences[BC11]—which is what makes the strategy of 'rollback' inapplicable in general (see section 11.6.6).

It should now be clear that this is about the larger goal of arranging invariance over process conditions, i.e. dependencies. The surrogates that often stand in place of this, such a statelessness, and causal ordering, are themselves non-invariant characteristics and should therefore be avoided.

A common mistake is to try to assure invariance by acting 'only once' (the FCFS random walk approach to state, rather than the determined fixed point). For example, in the delivery of a transaction. We might number transactions, like TCP sequence numbers, and tick them off a checklist as they are completed. This leads to a growing process memory (a stateful process). It can be replaced by a memoryless local process using advanced causation.

Advanced causation (treating the end state as a fixed point) has many uses, e.g. for desired state policy enforcement. Systems whose interior states are changing may not have homogeneous transitions across different replays, but a choice of a fixed attractor is equivalent to inline error correction.

Relying on thing that happen only once is a non-invariant procedure (changing the sampling timeout can change yes into no). Messages may be repeated or lost, and isolation from interference is not a promise that can be kept easily (process isolation is often the first thing violated by intrusions and security exploits). If we seek a deeper level of safety, it makes sense to rely not on the keeping of promises that are fed as data, but on the characteristics that are more likely to be preserved, such as convergence to fixed points[165]. The surest means to achieve repeatability is the maintain the promises on a timescale shorter than that at which they are sampled. This is the Nyquist sampling theorem in action.

Advanced propagation determines based on a desired state x_D

$$x_{\text{end}} = f(x_{\text{any}}) \tag{11.51}$$

$$x_{\text{end}} = f(x_{\text{end}}) \tag{11.52}$$

We see that the final value is insensitive to the initial value, which is in strict opposition to the functional idea of past forming immutable facts. The immutable fact lies in the definition of the function itself, which refers to an 'inevitable' future state.

The outcome is idempotent when it reaches its final state, not after a certain number of transactions 'once only' has been reached[Bur4 a, Bur04c]. The approach is what the immune system does, and was used famously in CFEngine[Bur95, Bur04c] and later configuration tools[166]. It's also the approach used in pull requests, and GPS locators. The processes are designed to favour a predetermined outcome. The outcome will only become an event in the agent's future, and will only be observable as a future event by other agents that depend on it.

On the interior of a process, a fixed point of a chain satisfies conditional promises:

$$\begin{aligned} A &\xrightarrow{+X_p|X_i} A' \\ A &\xrightarrow{+X_p|X_p} A'. \end{aligned} \tag{11.53}$$

The more familiar retarded process is a Markov chain, to some order, and has no deterministic end state unless the agents keep their promises perfectly, which is essentially impossible to promise.

11.6.11 THE PHASE AVERAGING TRICK FOR NOISE REDUCTION

Noise is unintended signalling, which is received (-) by an agent, and interpreted as a message. Noise is difficult to deal with once it arrives. Without a guide to the correct information, a receiver is at the mercy of the noise. The approaches for noise elimination are identical to the strategies for any other repair: quick repair versus redundant failover.

- In band error correction was studied by Shannon. His error correction theorem showed that, with suitable channel encoding, a part of a channel capacity could be given over to the elimination of noise with arbitrary accuracy, at the expense of a loss of transmission rate[SW49, CT91].
- If we can introduce redundant parallel channels that absorb errors with the opposite sign, then they can be cancelled on recombination. This is difficult for semantic errors, but easier for dynamical errors. This is the trick used in making balanced line audio cables. This is a dynamical averaging technique, by ensemble resilience.

11.7 SECURITY

The topic of security is one that's rarely integrated into discussions of system design, in spite of its importance. It's more commonly viewed as an added extra, which is unfortunate as this goes along way towards mystifying the topic, leading it to be associated with the somewhat arcane and mysterious worlds of cryptography and trickery. Nevertheless, security can be defined straightforwardly using the notion of promises. Readers are

reminded to review the basic concepts rights, permissions, authority, and sharing (the opposite of privacy) in chapter 2.

11.7.1 DEFINING SECURITY

It's unusual for security to be defined alongside ordinary system concerns. Perhaps we are blind to security issues because we are trained to think in terms of deontic concepts of obligation: what must or must not happen, what is allowed or disallowed, instead of thinking in terms of what we can and cannot promise[167]. The tradition of obligation thinking has blunted our senses to obvious issues. Privacy is a simple example of this. The absence of a promise to share something, for example, is not a promise of an absence of sharing! This lack of attention to casual uncertainty lies behind simple mistakes on all levels, and it tends to trip up designers, especially in Information Technology, where people have been taught a naive and comforting idea about Boolean logic.

We begin by defining security in as uncontroversial a way as possible. The Oxford English Dictionary asserts:

Definition 224 (Security)**.** *The term security is used in a number of ways:*

- *As an adjectival state, it refers to the state of being free from danger, threat, or risk, i.e. avoiding dependencies that may become non-beneficial to an agent's own integrity for promise-keeping.*
- *As a noun, it refers to a guarantee of fulfillment, e.g. a certification or attestation to an outcome—implying a promise of no risk, or an obligation to deliver.*

Clearly security is an aspirational state rather than a realizable goal. Even if a complete specification could be promised, meeting all stakeholders' expectations, promises are not guarantees, and rely on dependencies over which we have no control. Indeed if we start from the axiom that an agent can only make a promise on behalf of itself, then any promise of security could only be made by an agent about itself, i.e. concerning personal security—yet the subject of the promise concerns external things (threats, dangers, etc). Promising security amounts to a declaration against self-harm—often this is more of a self-deception than a realizable promise.

As a noun, 'a security' (such as the usage in finance for bonds and collateral obligation papers) is only an obligation (which is thus likely to be unsuccessful), we take it as an axiom that no agent can supply guarantees (promises are not guarantees), since the law of autonomy tells us that two parties are always involved in delivering a cooperative outcome.

Let's point out some flaws of reasoning that lead to incomplete promises being made. Perhaps the most basic error is in believing oneself to be in control of a system: that which is not designed doesn't occur. There are always degrees of freedom in systems that we cannot account for, unless we can arrange a significant degree of isolation from outside influence. This becomes increasingly difficult as the 'size' of a system grows, as measured by the system surface, i.e. the agents that can interact with the exterior.

Theorem 6 (Absence of a promise is not promise of absence)**.** *The absence of a promise of b is not a promise of the absence of b.*

The proof of this is trivial.

$$A \overset{b}{\not\to} A' = A \overset{\emptyset}{\to} A' \neq A \overset{\neg b}{\longrightarrow} A' \tag{11.54}$$

The absence of a promise is no promise at all. This simple observation (make so many times through history) seems laughable, and yet it catches us out repeatedly.

Example 280 (CFEngine classes)**.** *The state classifiers in CFEngine that promise evidential properties about a contextual environment have often been confused with boolean variables, but as promises they cannot be booleans. This does not mean that one can't use them for Boolean reasoning, as long as one understands this simple point. A promise of* `Monday` *acts as though it were a boolean truth statement. If this promise is not given (because it's Tuesday, say) then we can easily infer that it is not Monday, i.e.* `!Monday` *is true. In this case it's because the observation of the date is always promised, and there is mutual exclusivity between the outcomes. However, if a test for a property is not promised (or the result is not accepted) then using the absence of the class as evidence of its absence is inconclusive. For example, testing whether a system is a member of the group of DNS servers (a fact that could be changing in real time) relies on a Nyquist sampling rate being sufficient in relation to the use of the fact to return to 'correct' answer. To the use of the class* `!DNSserver` *may not be treated as a boolean variable, since its status is not synchronized with immutable observations.*

11.7.2 THE DYNAMICS AND SEMANTICS OF SECURITY

Security relies on a mixture of dynamical and semantic intentions, i.e. promises. Trying to ensure the continuity of a system, as intended, is the business of security. Avoiding failures, faults, and errors is a key part of this continuity. One could easily get the (unsubstantiated) impression that the usual approach to security is to try *prevention* of fault states from occurring in the first place. But avoidance could be both expensive and lead to misplaced effort. In many cases rapid recovery is a better option.

Agility is a surprising answer to address security, which goes against general advice from experts, yet being evasive is a good defense from the unpredictable. Changing the objective of a system in realtime may lead to chaotic behaviour, but it could also save you from the unexpected. Recall example 266.

Example 281 (Closed circuit TV camera)**.** *Security cameras sweep back and forth to try to prevent intrusions by detecting them quickly. If a thief can run twice as fast as the camera sweeps he can avoid being seen by the camera—running only when the camera is pointing away. The timescales of a security process have to match those of the threat model. Prevention may therefore fail if the prevention process is not agile enough. A fast recovery, by a rapid response team can still be just as good as the prevention if it can response before any harm is caused. This is another example of the use of Nyquist's theorem.*

Realistically recovery is always a better strategy than prevention, because we need recovery anyway. So, if we design systems to be intrinsically recoverable rather than being intrinsically stable, we can fend off vulnerabilities both dynamically and semantically as long as we don't allow too large an envelope of possible states to wander about in.

Security is easy to *define* and to *police* when it comprises a set of fixed promises, i.e. when it is rigid and inflexible. This makes it easy to manage by prevention, but both vulnerable to brittle failure and unable to adapt to changing circumstances, i.e. potentially flawed and unfit for purpose. Opening security systems to realtime adaptation, such as in a mixed game strategy or even a non-linear response, opens to the potential of unpredictable and even chaotic behaviours, and moving targets, but it could ultimately maintain the promised integrity without interruption. This is the dilemma of choice system designers face. Ultimately, the optimum choice requires a careful analysis of timescales over which perturbations and responses can be expected to occur. As always we return to the fundamental observation about system behaviour: predictability comes from an attention to the proper separation of scales.

11.8 PLANNING CONTINUITY

Having a system persist in the face of perturbations is the goal of resilient continuity, even when propped up by repairs, replacements, and upgrades made 'in flight'. The metaphor of changing aircraft engines while in flight is often used for mission critical continuity. Replacements can easily become temporary self-induced faults.

Whether or not we can accept 'downtime' or periods in which the engines are not running is entirely a question of timescales. This goes back to Nyquist's theorem, which

we can restate as a basic law of continuity:

Law 7 (Continuity law)**.** *If a promised dependency is used (sampled) faster than it can be repaired, the a failure will propagate. If it can be repaired faster than it is sampled, continuity will be maintained and faults will not propagate.*

This follows essentially from Nyquist's theorem, or the Maintenance Theorem[Bur03] described in volume 1.

Faults, i.e. states that conflicts with what is promised, take several typical forms:

- States arising from logical errors.
- States arising from random errors.
- States arising from (unexpected) emergent interactions.

We can adopt different strategies for solving a problem:

- Mitigate the damage by relieving symptoms on a regular basis.
- Fix the cause of the problem at designated source, aka 'root cause analysis'.

In many complex systems, it is profitable to employ both of these. While the belief in a single 'root cause' is at odds with the network nature of most real systems, in practice this is simply a convenient definition that relies on a prescribed model of the system. If we learn anything from this volume is it surely that, without as model of a system, we are flying entirely in the dark.

When a fault arises, either spontaneously or due to an error, we generally have to resort to one of the four R's of technology: recovery, repair, and ultimately replacement for resilience. Forensic investigation (aka 'debugging') is often initiated as a meta-process to find out which promises were not kept and why. An integrated process of self-monitoring and regulation (see chapter 12) can be a way to avoid any unpredictable change.

11.8.1 DESIGN WITHOUT FLAWS

Avoiding problems before they occur would be a simple-minded way to avoid trouble. Prevention may be preferred, but—given that we are always operating with only incomplete information—it sidesteps the important potential for learning from experience in-band, as we go, and the attendant 'anti-fragility' that could result[168].

Assuming that the arrangement of components in a system is determined by its function, first and foremost, let's ask what measures we can take to reduce unreliability in the agents. A natural beginning is to ask the most basic of questions:

1. Which agent makes the promise?
2. In what context does the promise apply?
3. Is the agent robust? Does it incorporate redundancy internally?
4. Is the agent's promise adaptive or fixed/dumb with respect to context?
5. Who are the stakeholders?

11.8.2 SYSTEM ISOLATION

As mentioned in the introduction (see section 1.6), systems do not usually have simple or well-defined boundaries. Some issues might be localizable within a certain scope, say a geographical region, others may roam freely. Systems should be considered 'open' in all practical circumstances. Many of the difficulties and errors that are made about systems arise from the tacit treatment of systems as being closed, i.e. existing immutably and in isolation from external forces. However, as soon as we make use of a device or procedure, it interacts with and is altered by its environment[169]. Design flaws often revolve around the assumption that systems are only affected by intentional changes made one at a time inside a system's artificial boundary. In practice, we know that the largest perturbations inflicted on any system come from its environment—often from its users.

11.8.3 SCALING ROLES TO ELIMINATE SINGLE COMPONENT FAILURE

The lesson learned from classical reliability theory is that redundancy—on the lowest possible level—is the most effective way to avoid immediate failure, but this might only delay a more extensive failure. Once a crack has opened, it will tend to propagate. Redundancy and risk mitigation are gambles against the laws of chance. The same is true of the capacity of a system to handle loads. Load sharing is another application for redundancy. As load rises, what might seem initially redundant becomes a matter of necessity.

We thus try to design for embedded repair and replacement, by designing around *roles* rather than commodity components. Commodity components are cheap and have the promise of easy availability, but they might not themselves be robust. So we need to build systems with hierarchical redundancy built in. Every agent can be replaced by a superagent, with a geometry that covers as many failure modes as can be predicted.

Redundancy is not only a dynamical continuity, but also about semantic continuity: were procedures followed properly?

Example 282 (Dynamical redundancy)**.** *Is there a backup that keeps the same promise?*

Example 283 (Semantic redundancy)**.** *Were observations correctly interpreted? If there a backup plan if one selected outcome is fails?*

11.8.4 THE INS AND OUTS: BOUNDARY CONDITIONS

Any interacting system operates with a set of promised inputs and by promising a set of outputs. If the output is conditional on the inputs, then the promises are in the manner of a transformation matrix. Mimicking the notation of quantum theory S matrix, we can write it as a scattering matrix. The information 'in' forms a boundary condition for the transformation, and the information 'out' forms another. Dirac's suggestive notation can be used here to good effect: a transformation T is a kind of matrix multiplication:

$$\langle \text{out}|T|\text{in}\rangle = \begin{cases} A_{\text{in}} \xrightarrow{+I} T \\ T \xrightarrow{-I} A_{\text{in}} \\ T \xrightarrow{+O(I)|I} A_{\text{out}} \\ A_{\text{out}} \xrightarrow{-O} T. \end{cases} \tag{11.55}$$

The output is sensitive to the inputs, but also to the bindings in the transformation. If the inputs are misinterpreted or distorted in an unintended way by the intermediate agents of whatever process is represented by T, then the output will be affected. It's worth remembering the intermediate agent law: promises cannot be propagated through intermediate agents without uncertainty.

11.8.5 SHRINKING THE POTENTIAL FAULT SURFACE (CONTEXT)

The minimization of the 'attack surface' for a system is often used as a metaphor for systems, deriving from military terminology. This language is less appropriate for systems that are not based on entirely physical interactions. The inputs or boundary set for acceptance of influence, in the foregoing section, is an obvious place for errors to be propagated, and for new ones to occur. If there are N input agents to a transmuting superagent, then the uncertainty or potential for error is N times as large as for a single agent. This tells us that we should be wary of all dependencies. However, what happens next in the agent is more important to the causal trajectory of the system (see figure 11.11).

A convergent agent, or closure, which maps all inputs to a single fixed point attractor is an example of an extremely fault tolerant transmuter agent. No matter what the inputs, the output will be the same (e.g. CFEngine).

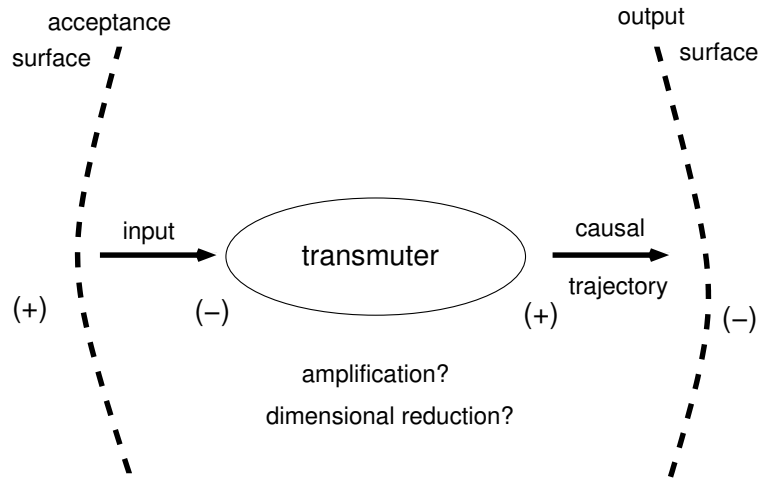


Figure 11.11: A transmuter pattern accepts input and generates modified output. The output degrees of freedom in the output might be greater or less than those at the input, depending on whether its acts as amplifier or discriminator. The potential for vulnerability depends on this amplification, not on the 'attack surface' per se.

Dependencies are inputs. System boundaries might not be easy to describe, but one can always try to reduce their potential contact with environmental ins and outs by limiting interactions. This lesson applies between every stage in a multi-stage transformation workflow, such as figure 11.2.

Example 284 (Pattern recognition and AI)**.** *In so-called Artificial Intelligence machine learning systems, discriminators are built to perform a dimensional reduction. One way to view these discriminators is as transmuters that perform a dimensional reduction from an input source with a large amount of diverse and high entropy data into a number of symbols at the output, e.g. in recognition of cats, dogs, etc from picture data. Attempting to attack the input to expose a vulnerability is harder the smaller the number of symbols at the output, as the possibilities for false matching are reduced.*

Example 285 (Spectral analyzer)**.** *The example of pattern recognition above is a special case of spectral analysis methods. In astronomy, light from distant stars or from chemical vapours is passed through a prism for separating the wavelengths that are absorbed by key materials. In this way, one can infer the composition of the source from the discrete spectral lines at the end of the transmuter. The uniqueness of the absorption spectra makes misdiagnoses unlikely.*

Example 286 (Sudo not su -)**.** *A command line operation is a concrete example of what we abstractly refer to as transformations. The Unix command* `sudo`*, which grants one time root access, is often recommended for use instead of a privileged root login. This is because it grants privilege only for the duration of the named command, preventing accidental invocation of other commands out of context.*

11.9 SUMMARY

What we have seen so far is that promises are kept and cooperation is won through networks of agents whose only link is the keeping of promises. The tolerance of a system to the unexpected is an important principle to contain the envelope of possible states that it might encounter. The concept of safety is somewhat beyond the scope of this volume, but it's clear that we are not far away from defining that too with these elementary formal considerations.

Some topics here, like transactional isolation, might be equally placed under the heading of processes and their predictability. I chose to include them here under the topics of resilience and recovery to emphasize that we should not be complacent when adopting such techniques. There are no panaceas when it comes to equipping process with armour. There is always a complex interplay between semantic intent and dynamical response.

CHAPTER 12

SYSTEM KNOWLEDGE

> "Skilled workers tend to think that it is enough to be smart. In fact this is wrong: smart people tend to be problem solvers and will happily solve the same problem many times, wasting time and effort. Moreover, human intervention is often based on panic and lack of understanding so every time someone logs onto a system by hand, they jeopardize everyone's understanding of the system. Only the self-discipline of stable procedures leads to predictability. "
>
> –The Author, CFEngine 3 tutorial (2010)

In the final chapter of these notes, I want to take a step back from the details of specific promises and look at the greater purpose of observing and describing systems. Without cognitive faculties, on some level, studying systems would be impossible–for any agent. Promise Theory is interesting because it removes the distinction between assessments made by humans and assessments made by any agent in a system, with any set of capabilities. Human assessments might be more sophisticated (of greater semantic complexity) than those, say, of a thermostat, but they are no more valid.

What higher intelligence adds to assessment is the ability to build causal narrative. Without the ability to formulate stories, based on cause and effect, there would be no organizing principle by which to discuss behaviour. The echoes of space and time lie deep within every description of phenomena, and there are meta-strategies for how we formulate narratives about them. That's a topic worthy of a book in its own right, but it's worth sketching out some of the issues in the light of the previous chapters.

12.1 SCALES OF KNOWLEDGE

Like any other information-based phenomenon, knowledge can be observed at multiple scales. We can know different things about a system at different levels of abstraction (difference semantic or dynamical scales). All knowledge of phenomena is based on a learning relationship with agents associated with those phenomena[Bur09]. Discussing knowledge fits cleanly into the notion of promises, with agents playing the role of the entities about which knowledge can be learned, promises playing the role of observable evidence, and assessments playing the role of measurement in its broadest sense.

The principle of separation of scales must therefore apply to knowledge too. This means that it's possible to discuss a subject at several levels, about promises that may be quite independent—from technical details to assessments of public significance, politics, law, and so on. This is a topic we don't have time to address here. Instead, I want to focus on the technical application of Promise Theory to address such narratives, as they may appear on any level or scale.

Example 287 (Aviation accidents)**.** *Aviation accidents invoke discussions at all levels—from the technical specifications of flight systems, to processes carried out for maintenance, in-flight operations, and eventually the public discussions (carried out largely in the dark, and based on speculation). Public discourse as well as technical specification can be couched in the language of promises as a lens through which to separate issues clearly [CL99, BB19b, BB20].*

12.2 FROM OBSERVATION TO KNOWING

The desire to expose the chains of cause and effect that underpin observable phenomena is irresistible, yet we know that the ability to observe and interpret systems is both limited and compromised by availability, relativity, signalling speeds, as well as by the noise of environment. The state of the art in observational methodology still relies principally on brute force data collection and graphical presentation, perhaps with some regression analysis. There is surprisingly little discussion about the semantics of the process[HL93, Hog95, Hel96, PS06, SBS99, DF98]. Uncertainties from sensory instrumentation are often left for human operators to untangle on their own. In this chapter, I want to seek a few principles that can be applied across hybrid systems[170].

The 'measurement problem', as it is known in physics, bedevils every corner of science in different ways. We need to ask: is there a consistent viewpoint on what's happening in a system, which can be arrived at without doing such violence to the system? The ability to know something about an adversary or a friend doesn't come by magical revelation. It doesn't happen suddenly. It grows as a process of iteration, by building a

relationship, and assessing what promises are made and kept. This is supported by two key pieces of work: Dunbar's social brain hypothesis [Dun96, ZSHD04] Axelrod's work on non-cooperative game theory [Axe97, Axe84]. These two foundations suggest the importance of understanding knowledge in terms of ongoing processes with certainty and trust that evolve with experience[171]. The topic of knowledge, then, is related to the power of observation, as well as the ability to form assessments of the behaviour of agents over different scales. This topic could easily consume several books on its own, so we suffice to lay out the basic promises and dependencies that connect with the language laid out in the foregoing chapters.

12.2.1 COMPOSITION

Systems, composed of agents, may be visible, invisible, transparent or opaque. What can we do with this? Being able to observe what happens is not sufficient criterion for being able to understand or predict its behaviours. If a system has no repeated behavioural patterns, watching it is futile—we learn nothing unless there is a pattern that can be symbolized and reduced to semantic value. The challenge of observation, then, is to go beyond data to structure patterns into alphabets and languages of change. By language, we don't necessarily imply something that is written down in words, but certainly something that can be written into symbols with an invariant meaning. Language adds some specific qualities to mere data: there is a separation of data into roles, prerequisite context, and scaled composition of concepts. Language embodies hierarchy and composability.

12.2.2 PERFORMANCE ANALYSIS

The observation of a system to determine its effectiveness and its efficiency is what we call performance monitoring. It relies not only on the ability to collect random data (which is unfortunately the standard in many systems), but to understand which measurements have a meaningful corelation with promise keeping.

Definition 225 (Performance measure)**.** *A measure, composed of qualitative and quantitative components, characterizing the rate, cost, and capacity of a system to keep its promises.*

The performance of a system might also be described as a measure of its spacetime envelope, its set of possible trajectories and their characteristics, etc. Performance requires a model to define a measurement precisely. It might therefore include simple measures like 'time to keep promises' (or Mean Time to Promises, MTTP).

Defining the effectiveness of a system is straightforward.

Definition 226 (Effectiveness)**.** *The effectiveness of a system is the measure of what fraction of its resources keep their promises over a set of measurements.*

$$Effectiveness = \frac{Promises\ kept}{Promises\ made} \tag{12.1}$$

Defining the efficiency is a different matter. When is a system not completely efficient? This implies a policy about the use of resources, which might not be knowable.

Definition 227 (Efficiency)**.** *A measure of what fraction of the resources were actually needed to keep their promises over a set of measurements.*

$$Efficiency = \frac{Minimum\ required\ resources}{Actual\ resources\ used} \tag{12.2}$$

This fraction is vaguely defined, and once again requires a model to make sense of it. For that reason, efficiency is generally ignored as an issue in systems. Time and motion studies of human systems were often used to try to improve and economize on resource usage, but this requires only relative quantification. A measure of efficiency suggests that we can offer an absolute quantifier of the envelope of resource usage. That, in turn, suggests that there would be an attractor for perfect efficiency, at which a minimum level of resources was reached. That's a speculative claim as it depends on the method and machinery used to achieve the outcome, which might not be unique. There is an link to the question of faults and design flaws here. An inefficient system might be considered one that has a fault, as it fails to keep its promises within a budget. Alternatively, a bloated system might be considered a design flaw.

Example 288 (Software bloat)**.** *Modern software is often characterized as 'bloated', implying an inefficiency. That comprises too many layers of software, unused libraries, redundant functionality, etc. A small part of some software may be fit for purpose, without the remainder needed at all. In modern parlance, the strategy of microservices or 'function as a service' has been used as a way to improve this duplicated redundancy, by centralizing software as a service to be rendered for a statistical population of users. In a similar way, the resource cost of running software on emulators and virtual machines, with many layers of virtualization and management has led to concerns about energy waste. Unikernels, which strip software down to a minimal level, have been one proposal for reducing that waste.*

Example 289 (CFEngine control envelope)**.** *The CFEngine software constrained the performance of processes using two parameters to limit resource usage and protect against Denial of Service attacks. These were called ExpireAfter (a maximum time a process should be allowed to run on order to keep an atomic promise), and IfElapsed (a*

minimum time that should have elapsed, according to a local clock, before the state of a promise should be sampled and perhaps repaired again).

12.2.3 SMART OR EFFECTIVE COGNITIVE SYSTEMS AND DUMB SYSTEMS

Cognitive systems are another name for sensory learning systems. Some readers might disagree with this characterization, but this is a useful starting point, with a minimum of assumptions. It can be scaled and extended to enhance and embellish the position. I choose it for its universality. For an extensive justification and motivation see [Bur19c], or simply take this as an axiom. Dumb systems have been the norm for mechanical automation until recently: preprogrammed systems, without sensors, enacting a simple non-adaptive program. However, increasingly human-machine hybrids are equipped to observe and receive information, from which they learn and adapt. This is the definition of a cognitive system—perhaps not a genius mega-brain, but a 'contextually smart' system nonetheless.

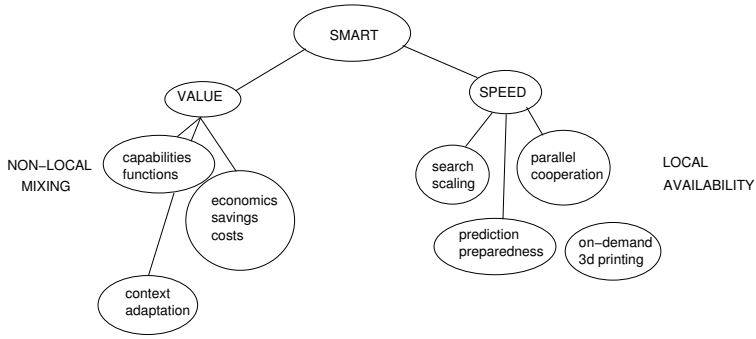


Figure 12.1: What makes a system smart or effective? A mixture of timeliness and fitness for purpose. Effectiveness and efficiency may be characterized in both quantitative and qualitative terms. Without the latter, measures have little meaning.

Smart is a vague and subjective assessment, but the term has become widely adopted as a marketing vector. We might also use the term *effective*. We give it to things and people we feel bring us value or save us some cost, in a timely manner. We perceive a thing or a process as helpful only relative to a context (i.e. fit for *present* purpose). The use of context implies that *sensing* of context is involved, and some feedback, influence or decision that is based on the result. Smart might simply mean a thing or process that arrives 'just in time' to be useful in that same context (too late is not smart). Smart

also sometimes means efficient, to the point, i.e. concise, 'not too much noise'. It's not essential to define 'smart', but it's a useful intuition in dealing with human-machine interactions.

From a promise theory view of the world, the way we always start an analysis is to ask:

1. What or who are the operational **agents** involved?
2. What **promises** do they make to one another, in what **context**?
3. What are the important scales for their interactions?

Scale always plays a key role in the understanding of systems.

Example 290 (Smart cities)**.** *Smart spaces can be imagined at any scale, from microscopic materials and disease technology to urban spaces and even economies. For static choices, we have already built these structures to the best of our abilities. The new challenge is to extend 'smart design' to time-varying adaptive cases. There are two approaches one could take for 'smart spaces'*

- *To enhance the experience of people living in or around smart spaces. (***Interactive***)*
- *To create autonomous, specialized functional spaces (like factories, farms, or organisms) that are more self-sufficient. (***Passive***)*

In either case, the meta-goal is to bring a positive benefit to society at large. However, we have a choice about what this means:

- *Community of smart individual units (putting individuals first).*
- *Smart scaled singular unit (putting the city first).*

There are many complex issues at play. The most important consideration is the scale *at which systems interact with one another. Both viewpoints have a place, but can they be aligned simultaneously?*[172]

Smart cities are an important case, because they span most scales, and allow us to see many of the cases under a single banner, so I will focus on cities as an umbrella term. We'll need to address services and promises at each scale independently.

In a cognitive system, an observer (who is often a user of the system) has a cognitive trust relationship with the system as a provider. If the provider forces the user to jump through many hoops and adapt to its case, trust will be low. This is a dilemma for security, because it implies that strict controls and rigid protocols will end up not being respected.

Too many barriers also degrade trust[Tai88] and may lead to system fragmentation. Openness (trust) on both sides of an interaction is the lowest cost configuration.

12.2.4 LEARNING SYSTEMS

The concept of learning has become loaded with expectations since machine learning technology became commonplace. It's too broad a term to define without a specific subject about which we learn[173].

Between data and semantics there is learning.

Definition 228 (Learning about π)**.** *Let S be an agent which promises a sequence source of data to a receiver R:*

$$\pi : S \xrightarrow{+d_i} R, \quad = 0, 1, 2, 3, \ldots \tag{12.3}$$

We say that R learns about a sequence d_i or its promise π, if R promises to repeatedly sample the sequence $+d$ and adjust its belief estimates $E(d)$ according to some learning (memory) function L:

$$R \xrightarrow{-d} S \tag{12.4}$$

$$R \xrightarrow{+E(d)|L(),d} ? \tag{12.5}$$

where

$$E(d_{i+1}) = L\left(E(d_i), d_{i+1}\right). \tag{12.6}$$

The function $L()$ typically represents some kind of Bayesian learning. Learning is a relationship accumulated over time, so it fits naturally with service relationship.

Definition 229 (Longitudinal (cognitive or timelike) learning about π)**.** *Collection of samples at sequential times,*

This is naturally associated with Bayesian probabilistic analysis, since the Baysian formula represents a timelike oriented process.

Definition 230 (Cross-sectional (spacelike) learning about π)**.** *Collection of semantically simultaneous samples (i.e. under approximately 'identical' conditions).*

This is naturally associated with frequentistic probabilistic analysis, which is a constant-time ensemble aggregation.

Comment 27 (Learning and time)**.** *The timescales for learning:*

$$T_{user} \simeq T_{client} \simeq T(r_X) > T_{learn\ about\ user} \gg T_{server} \simeq T(X) > T_{learn\ about\ server} \tag{12.7}$$

12.3 CONTEXT FOR ADAPTABILITY

This section is based on the discussion in [Bur16c]. Context is an assessment of the state of 'here and now', summarized from whatever sensory inputs are available. Context may refer to short-term and long-term memory of phenomena that have occurred before.

Definition 231 (Context)**.** *A summarization function $f()$ of recent sensory inputs d_i, collected from sensors S_j promised in a form that may be used to predicate (discriminate) behaviours or decisions. Context can be promised by an agent A that promises to aggregate data from sources S_j:*

$$S_j \xrightarrow{+d_i^{(j)}} C \tag{12.8}$$

$$C \xrightarrow{-d_i^{(j)}} S_j \tag{12.9}$$

$$C \xrightarrow{+f(d_1^{(1)},d_2^{(1)},\ldots,d_i^{(j)},\ldots)|d_i^{(j)}} S_j \tag{12.10}$$

The function $f()$ may be a simple Markov process, or an n-th order Markov memory process. The domain/range of $f()$ can be adapted to each case.

In order to be useful, context has to be something we can use to select decision pathways, so it can act as a kind of discriminator, turning complex inputs into simple selection outputs. It has a tokenizing, or digitizing role.

It is useful to define the two components of context in a learning system:

Definition 232 (Exterior Context)**.** *Context in which the source of data arises from outside the learning system, i.e. from sensory inputs that receive unknown impulses from the exterior environment.*

Definition 233 (Interior Context)**.** *Context in which the source of data arises from inside the learning system, i.e. from the sensing of active concepts and memories, previously integrated into a world-view.*

Interior context may become active as a result of exterior context, or it may be randomly activated (as in dreaming). So the schematic flow of reasoning is:

1. S offers (+ promises) data.
2. R accepts (- promises) or rejects the data, either in full or in part.
3. R observes and forms an assessment $\alpha_R(.)$ of what it receives.

This third and final stage is the moment at which data can be said to arrive at the receiver.

The details of a physical network are not directly relevant, but the topology of actual interactions between agents is. It depends on the promises made between pairs of agents, which therefore serve as documentation of intent. An offer promise with body $+b_i$ made by S_i to R_j is written:

$$S_i \xrightarrow{+b_i} R_j, \tag{12.11}$$

where the $+$ refers to an offer of some information or behaviour (e.g. a service). This is a part of S_i's autonomous behaviour, and the promise constrains only S_i. R_j may or may not accept this by making a dual promise, marked $-b$ to denote the orientation of intent:

$$R_j \xrightarrow{-b_j} S_i. \tag{12.12}$$

If both of these promises are given, and kept, then influence in the form of vital information about the body b will pass from S_i to R_j. In general, the offer and acceptance may not match precisely, in which case the propagated information will be the overlap (mutual information)

$$b_\cap = b_i \cap b_j, \tag{12.13}$$

in the manner of mutual information[SW49, CT91]. I'll suppose that modern systems are cloud computing systems. The elementary agents of cloud computing are *processes*, any of which may express promises about state and services. Processes are hosted at agent locations A_i, S_i, R_i etc.

I use the following nomenclature for message agents M:

- E_γ is an event, for example $L_\gamma \subset E_\gamma$ may be a line of information reported in a log or journal. Greek indices γ label information agents successive packets, i.e. $\gamma = 1, 2, 3, \ldots$.
- $\{E_\gamma\}$ or$\{L_\gamma\}$ refers to a collection of such events or lines.
- $S_i, R_i \in A_i$ refer to processes running on computers.
- $\{S_i\}$ refers to a collection of sources, etc.
- A_i refers to a process checkpoint in some kind of dataflow, which has its own interior event log and counters. Checkpoints typically make promises about their identity, location, local counter values, and intent to pass on data in the form of packets P_i, with some promised order.
- P_i refers to a data packet passed between checkpoints agents. Packets typically make promises about their identity, data content, schema, and type.

Latin indices therefore label locations, and Greek indices label events at the same location.

12.4 DEFINING THE KNOWLEDGE PROBLEM

Let's use the language of Promise Theory one last time to formulate the basics of acquiring knowledge as part of a process of observation.

12.4.1 WHAT IS INTENDED AND WHAT IS PROMISED?

There are two ways in which we use data to interrogate a number of processes:

- **Tracing**: ('During') —in band observation, in which data are sampled *intentionally* and recorded as a process unfolds to maintain 'situation awareness'. e.g. the ECG or life monitor approach to medical monitoring.

- **Diagnosis**: ('After') —out of band forensic reconstruction of a system using data one can find after an incident, where intent to comprehend kicks in only after the event has occurred: e.g. the post mortem approach to medical investigation.

Most users will try to combine these approaches, paying attention mainly when significant events occur. The automation of alarms (usually based on simple-minded absolute thresholds) tells human operators when to pay attention, at which point they have to rely on what has been traced. The promise to maintain awareness is an expensive one, and we rely heavily on our skills of reconstruction after the fact.

12.4.2 THREE PERSPECTIVES ABOUT SCALE AND RELATIVITY

Distributed processes are composed of agents that pass information in space and time (see figure 12.2). Messages or events propagate from one agent to another, and we consider the arrival of such information to be an advance in the 'state' of the distributed system, which is what we mean by the *proper time* of the process. Events that occur in parallel, as separate logical locations, know nothing about one another—they are causally disconnected and lead independent lives. The time on the wall clock or system clock is not a 'proper' time, as we'll see below[174].

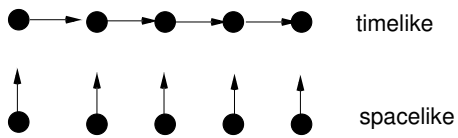


Figure 12.2: Space and time as agent parallelism and serialism respectively.

There are three kinds of story or explanation we want to be able to tell about distributed systems (figure 12.3):

1. The data traveller log. What a travelling data packet experiences along its journey, e.g. which software including version handled it and in what order?

2. The checkpoint visitor log, from key signposts around the data processing landscape. The log of what each checkpoint along the journey saw, i.e. which data packets passed through the checkpoint and what happened to them?

3. The map of combinatoric intent, i.e. the relationships between invariant elements and concepts, including the topology of checkpoints and influences, the types of data passed between them, significant occurrences, and so forth; i.e. the semantics of the data, software, and invariant qualities and quantities that summarize the processes within the system's horizon.

These viewpoints require separate data collections. Present day logging systems focus almost entirely on the second of these.

In Promise Theory, one reduces a system to a collection of agents, their promises, and their assessments. Agents include the checkpoints from which data emerge and are collected. A second layer of agents comprises the data packets that are transmitted. The promises made by these agents include communication, data compression, speed, and integrity. They may include data formats and ordered protocols. We equip different agents in the system with promises to report the information available to them to observers. I shall not be concerned with matters of authorization and permission in this paper, but rather focus on the difficulties experienced by those who are promised information.

12.4.3 DIAGNOSIS

It's up to an observer to infer something about the state and history of a system, based on what is observed. This is not as straightforward as software systems have come to assume, especially as cloud computing pushes the limits of observability. At some point, this reconstruction involves a form of reasoning—not necessarily rigid logical reasoning, but at least a process of joining dots into an acceptable story.

12.4.4 DIAGNOSTIC MESSAGES

Process tracing is a simpler problem than reasoning, because it can be constructed as a purely Markov process—at least in principle. Tracing is the construction of a totally ordered path through a set of agents. Reasoning, on the other hand, involves semantic relationships between clusters of agents that may be considered to have an invariant

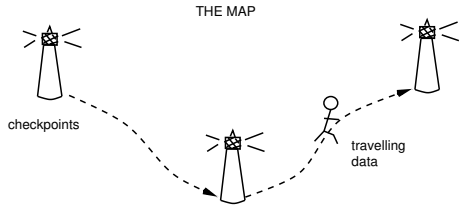


Figure 12.3: 3 Views. Travelling passport documents, versus logs of entry and exit from a checkpoint, versus the map of checkpoints and routes.

meaning, and it may combine several traces into a satisfactory explanation. According to the definitions in [Bur16c, Bur17c], I'll simply define the following:

Definition 234 (Reasoning)**.** *Reasoning is a search over a graph of ordered conceptual relationships.*

This pragmatic and unconventional definition might offend some logicians, but it's closer to what humans call reasoning than a definition based on mathematical logic. A few common issues crop up in diagnostics:

- The predictability of agents' behaviours.
- The distinguishability of agents and data messages.
- Loss of information due to mixing of origin sources.
- Reordering of information due to latency.

The problem with the first two kinds of story in the list, is the lack of a deterministic and universally defined order between the transactions of 'event driven' processes at separate source locations. The extent to which we can write down spatially invariant orders, process summaries, etc, which may be expected to persist over a timescale useful for prediction, is the essence of the difficulty in tracing causal history[175].

12.5 OBSERVABILITY OF MESSAGES

From the foregoing, we can define the concept of observability between pairs of agents. It does not make invariant sense to speak of 'observability' without some qualifications, so we can be more precise:

Theorem 7 (Observability of X at S by R)**.** *A range of promised set values X, sourced from an agent S is observable by an agent R if and only if:*

$$\pi_+ : S_i \xrightarrow{+X_i} R_j, \tag{12.14}$$

$$\pi_- : R_j \xrightarrow{-X_j} S_i. \tag{12.15}$$

$$X_j \subseteq X_i \tag{12.16}$$

Note that the criterion for observability is not a deterministic guarantee the ability to obtain a value on demand. It is essentially a property of an information channel, in the Shannon sense. There is only a finite probability of all these promises being kept, which makes observation a fundamentally non-deterministic process. There are many impediments to keeping promises in practice, not least of which the the Law of Intermediaries[176].

By assumption, agent's are autonomous and each plays a role in the collaboration required to exchange the information involved in monitoring. The definition of observability illuminates a basic dilemma in monitoring: the autonomy of agents in any distributed process (i.e. their causal independence) means that there is fundamental uncertainty about the process of observation not just its outcome. Causal independence is the very definition of a random variable. A source of signals may believe it does all it can to ensure correct transfer of information, and that any problems lie in the delinquencies of the receiver; meanwhile, the receiver believes it does all it can and trusts the source and the network in between implicitly to report with complete fidelity. The assessment of X by R_j (denoted $\alpha_j(\pi_+)$) is still a function of R_j's access and capabilities at any given sample, and may be subject to environmental interference.

12.5.1 PRELIMINARIES ABOUT INTENT

Most technologists believe that, if they design without 'bugs', they can achieve whatever outcome they desire, given sufficient resources. This is not a scalable view, so we need to be more cautious. In the standard model of queueing theory [Kle76], data are produced by a source S_i, at a rate λ_i messages per unit time, and may be processed by a receiver R_j (sometimes called a server) at a rate μ_j. The queue is unstable and grows out of control as the traffic density $\lambda_i/\mu_j \to 1$.

12.5.2 EVENTS AND SAMPLING

Events were defined in definition 51 chapter 3. The concept of events plays a major role in the language used for monitoring and data flow in IT. The arrival process for events

of a particular type may have any kind of profile, from a strongly deterministic regular signal, such as a ticking clock, to an entirely random event.

There are two different kinds of distribution that characterize events:

- Interarrival times: the distribution shows the number of arrivals that fall occur since that last arrival, i.e. Δt versus $N(\Delta t)$.
- Frequency by class or type: shows that number of arrivals that fall into a certain characteristic sample class C, i.e. ΔC versus $N(\Delta C)$, for some characteristic promise of the data.

In classical component failure analysis (see chapter 10), fault events are assumed to have interarrival times distributed as a Poisson process. This is an experimentally observed 'fact' for the distribution of faults in a wide range of systems. Messages that arrive from users of the Internet have arrival frequency profiles that can be modelled as quasi-thermal equilibria[Bur00b, Bur00a].

12.6 AGGREGATION OF SOURCE DATA

It's time to look more carefully about what aggregation of data means. In the context of causality[DS05], it matters both where and when signals come together, and to what degree information is lost by mixing. Distinguishability plays, again, a central role here. For example, the commonly used metrics of load average, CPU percentage, memory usage etc. The behaviour of a process depends on the behaviour of the platform, which in turn depends on the behaviours of the guest processes[Coc06]. These are measures of different scales, since a platform is an aggregation of processes, and so on.

When unexpected behaviour (signpost behaviour) is observed in an aggregate variable, the culprit may not even be in the same process. The relevant question may seem to be: can we obtain information about which process may have been responsible? But that is not the right question, because it could be the accumulation of many processes leading to an exhaustion of resources which actually impacts the process we are monitoring—by undermining its critical dependencies. When this is the case, we might be more interested in why scheduling policies resulted in such a confluence of demand. Obviously, there are many layers of decision behind such stress concentrations.

The causal connection between these cannot be inferred with any certainty from quantitative measurement however. One would rather expect to see a process log fail to allocate memory from within the privileged context of the process itself. Today, we wrap processes in containers that are quite opaque. In fact, processes are equally opaque when viewed through kernel metrics, because there is unfortunately little or no causal connection between the changes and any particular process of interest.

12.6.1 SAMPLING RESOLUTION (TIMESCALES AGAIN)

We need to know when data belong together and when they should be considered separate. For any collector, this is a policy decision, but it can be informed by the processes of the system. There is information in the order, content, and volume of data. If that information is squandered, it may be unrecoverable.

12.6.2 METRIC SIGNIFICANCE

Data collection is frequently abused thanks to the ease with which data can be collected. Experiments show that there is often little correlation between commonly collected quantitative metrics and actual process semantics[BHRS01]. This is a historical artifact that comes from the fact that observables were designed for timesharing, not for process monitoring. Measuring kernel metrics is something analogous to watching the weather to plan for a crop. In some cases, a change in a distant place may may trigger outcomes that result from arbitrary choices in code elsewhere. For example, if one sets an arbitrary threshold for a value, in a conditional statement, the unusual process weather originated elsewhere may push the conditional over the limit unexpectedly and lead to a discontinuous branch of behaviour. This is why understanding relativity is so important in reasoning, and why cloud computing is especially susceptible to relativistic effects.

Entropy of mixing does not usually increase relentlessly in IT systems, because new information is being added in the form of semantic labels (boundary conditions) all the time, e.g. when a particular set of images is identified as belonging to the same person, we name the set as the person; or when a sequence of command instructions leads to the same failure mode, we name the histories with the name of the failure mode. This new information adds context.

As data scale, some information is lost, and new information is added. Aggregating data and integrating over time, throwing away time information, but building maps of invariant relationships. The map of what remains distinguishable grows as more data are added, because the number of possible storylines grows as new invariants are added.

Lemma 48 (New data at all scales)**.** *Origin data are lost by coarse graining, but combinatoric selections of aggregates leads to a new degree of freedom: distinguishable routes or paths through the composite variables.*

Each story has its own semantics: the loss of event indexing leads to the addition of fewer semantically stable storylines. Distances that require distinct labelling become meaningless, but new emergent distinctions lead to new possibilities for classification. If sufficient information is retained to point backwards along to causal signposts, specific paths can be traced without muddying the global picture.

12.6.3 LEARNING AND COARSE GRAINING DEFINED

We can track the different scales of a system by seeking to separate invariants from microscopic local changes, using the principle of separation of timescales. Learning over sequences (in a timelike direction) effectively form Bayesian processes that can act as aggregate state discriminators[Pea88, Pea00].

Example 291 (Learning)**.** *The collection of data to train a statistical algorithm may take weeks or months, and involve large amounts of data that are compressed into a composite form, irreversibly. The composite aggregate is used as part of an algorithm that recognizes images on a timescale of seconds. These processes can be naturally decoupled.*

The degree of separation of timescales corresponds to what we call supervised learning (highly separated timescales for learning and using) and unsupervised learning (where timescales for learning and using are approximately the same).

Even a single unique episode may eventually become viewed as an invariant if it is not repeated and hence is never challenged, but its significance may be limited. We can only know this by learning over time. The significance of concepts grows by the frequency by which they become repeated. Thus learning and garbage collection of insignificant concepts is needed to prevent all the information from becoming noise.

For full episodic reconstruction, the invariant connections that generate process stories need to integrate with one another, like a linked list, using the a map of invariant concepts as glue. The invariants represent the aspects of processes that are not specific to a single source. See the earlier work on characterizing spacetime semantics [Bur14, Bur15a, Bur16c, Bur17c], based on earlier experiments [CB09].

All causality is a representation of Shannon's basic model of an information channel. The distinguishability of information is the key to following and tracing processes, but where does the significance of information lie? The significance of information (associated with labels to it) is necessarily diluted by scale, or the entropy of signal aggregation.

12.6.4 THE MASHED POTATO THEOREM

Mixing of signals leads to loss of traceability. Suppose you are at a restaurant and you receive some mashed vegetables. You first assume that it's potato, because that is common, but something doesn't taste quite right. There are some other vegetables mixed in. Closer inspection reveals some orangy colour (perhaps carrots, or sweet potato, etc). How could you know what was in the potato without accurate knowledge from the source?

The loss of distinguishability (entropy of mixing) tells us that we can't easily discern the content of mashed potato without the recipe because we cannot separate (classify) the parts of the signal.

Theorem 8 (Loss of distinguishability). *Let Σ be an alphabet of class categories that are distinguishable by a set of source agents S_i. Data aggregated from S_i without complete causal labelling $\sigma \in \Sigma$, from the source cannot be separated into its original categories C_σ with certainty, i.e. the promise*

$$\pi_{categ} : R \xrightarrow{+C_\sigma|data_\cap} A_? \tag{12.17}$$

is kept with equal probability for all C_σ.

To see this, we note that the aggregation of data involves promises:

$$\pi_{\text{data}} : S \xrightarrow{+\text{data}_S} R \tag{12.18}$$

$$\pi_{\text{listen}} : R \xrightarrow{-\text{data}_R} S \tag{12.19}$$

$$\text{data}_\cap \quad = \quad \text{data}_S \ \cap \ \text{data}_R. \tag{12.20}$$

and we take data to be a collection of line signals data $\sim \{L_\gamma\}$. In order for the conditional promise (12.17) to be causal, the promise of data in (12.18) have to be a reversible function of the C_σ. But if data are indistinguishable, then the C_σ must also be indistinguishable, thus

$$C_\sigma = C_\tau, \quad \forall \sigma, \tau \tag{12.21}$$

thus, the probability of discerning a signal C_σ, $p_\sigma \equiv P(C_\sigma) = p_\sigma$, and the entropy

$$S_{\text{ent}} = -\sum_\sigma p_\sigma \ \log p_\sigma \to \max(S_{\text{ent}}). \tag{12.22}$$

If one rescales all the $C_\sigma = C_\tau$ into a single category to indicate that all such categories are the same, then the entropy is zero, $S_{\text{ent}} = 0$, indicating that the information per transmission, in the mixture, is actually trivial. Thus, we must keep all labels from different sources and classifiers in order to retain useful information. This does not depend on the amount of data (or mashed potato). Moreover, if data are passed on, the dependence of a reconstruction by (machine) learning is unstable to the datasets[177]. The definition of entropy in computer processes has been examined recently to address its semantics[ZKT19].

12.6.5 SEPARATION OF CONCERNS

The practical question remains: how should one separate variant and invariant data when designing systems? This depends partly on the structure of intent and observations.

Programmers are trained to recognize what values are variable and abstract them into parameters to invariant functions.

If we think about how we formulate stories, as humans, we embed variable fragments of causal history (episodes) into larger assemblies of more invariant concepts, which provide context for reasoning. This is how experiences are organized around conceptual models. I'll come back to this in section 12.8 on model extraction.

Example 292 (Logging text compression)**.** *As a simple example, consider the generation of a log message from a typical format string in code:*

- *A separate format string is an invariant class of messages. It can be replaced by a single numerical value and looked up in a hash table to compress data.*
- *Standard data format can record format string and variables in an indexed structure with named members.*
- *Message significance level or priority (policy) - imposed + or -?*
- *Variable Substitutions in the format string are variants with respect to the message. Some of these may be invariants too (the name of a host or function), while other data have no long term significance (the time or date).*

If we compare these points to a Unix syslog message, the `glog` *library, and many more examples, it's clear that syslog satisfies none of these promises. Lines of text are basically random.*

12.6.6 RETAINING SEMANTIC CONTEXT FOR EVENTS

Concepts are the result of dimensional reduction over contextual learning sets C_i from a number of sources S_i [Bur16c]. In invariant cases, the context can be learnt by accumulation of evidence over time, because it doesn't change. In general, significance may be assessed based on a number of contextual sets C_i, so when an alert message M is reported to an agent R, this is in fact a conditional promise that depends on the context:

$$S \xrightarrow{+M|C_1,\ldots,C_\sigma\ \forall i} R \tag{12.23}$$

This means, by the conditional promise law, that the promises supplying this context to S:

$$A_i \xrightarrow{+C_i} S \tag{12.24}$$

$$S \xrightarrow{-C_i} A_i, \tag{12.25}$$

$$S \xrightarrow{-C_i} R, \tag{12.26}$$

are not available to R. The context is lost. This means R has to trust the alert and its significance as a random variable. This is no problem if the goal is to bring an unrecognized condition to the attention of an operator. However, if the goal is to perform contextualized reasoning on an aggregate scale, the graph of invariant context also needs to be promised:

$$A_i \xrightarrow{+C_i} R \tag{12.27}$$

$$R \xrightarrow{-C_i} A_i, \tag{12.28}$$

and the conditional dependencies also need to be captured:

$$R \xrightarrow{-(M|C_i,\, \forall i)} S \tag{12.29}$$

If we didn't apply this idempotently only to sparsely occurring invariant concepts, the cost of the aggregation would rise sharply. An expedient separation of scales allows the context to be contained at the sources as 'smart sensors' [Bur16c, Bur17c].

Context can be framed and localized using namespaces. Namespaces also provide unifying labels that can usually be treated as invariants in information systems. Aggregation of messages, without cataloging, indexing, or other labelling leads to ensemble entropy: the irreversible loss of structural information and contextual semantics.

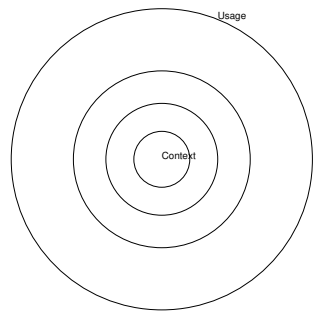


Figure 12.4: As data get propagated farther from their initial context, their original meaning is degraded, unless all context is transported with them. Each of the rings may represent an intermediate agent that may or may only promise to forward data selectively or after distortion.

Transporting too much context is a questionable idea. If the environment in which context originates is lost, then the meaning of the context is also lost, and the ability to reconstruct scenarios based on it becomes of largely forensic interest. System designers need to find expedient ways to compress context and filter it: what can remain local at the source, and what can be aggregated and assigned wider meaning? Thus is remains a

policy decision to balance the cost of preservation against the actionable usefulness of doing so.

12.7 HISTORIES: LOGS AND JOURNALS

Now aware of the issues around sequentialism and observability in distributed systems, we can tackle the first two story types in section 12.4.2. Logging of process conditions may well be the most popular and common approach to tracing in computer programs. Isolated single-agent logs are simple serial queues, or time-series databases, of varying degrees of sophistication, for keeping informative messages about what transpires in a process. This is no longer true for log aggregation unless complete causal linkage is preserved.

12.7.1 CAUSAL LINKAGE

In most shared logging services, messages are imposed by multiple process agents S_i onto a queue and are strongly ordered by a single receiver R.

$$S_i \xrightarrow{+L} \blacksquare\, R. \tag{12.30}$$

R accepts requests indiscriminately

$$R \xrightarrow{-L} S_i. \tag{12.31}$$

Individual processes can voluntarily write their own logs but this is not a common practice because the end goal of logging, in modern practice, is to aggregate all messages as 'big data' to be trawled.

Modern logging services, like Prometheus etc, provide more nuanced semantics with structured data formats that can incorporate key-values; but these trust data to be useful. They are abused greatly by programmers, who tend to dump any and all data into a stream without regard for meaning or consequence, in the hope of sorting it all out later. Logs need to promise invariant causation:

- Same text (signal) as the same interpretation.
- Information is encapsulated as transactions to show partial order.
- Every significant transaction needs to point to its previous significant event.

These principles have been embodied in a proof of concept implementation[Inc].

12.7.2 DROPPING HINTS

To record useful events, from within the meaningful context of a process, processes need an API that constrains authors to produce information that can be consumed later. For example, in the Koalja history package, based on the principles in this paper, significant events can be marked with signposts[Inc].

```
H.SignPost(&ctx,"Milestone 1...")
H.SignPost(&ctx,"Milestone 2...")
...
H.SignPost(&ctx,"Commence testing")
```

And these signposts can be detailed, using the four spacetime semantic relations from [Bur16c, Bur17c]:

```
H.SignPost(&ctx,"code signpost X").
    Intent("open file X").
    ReliesOn(H.NR("/etc/passed","file")).
    FailedBecause("xxx").
    PartOf(H.NR("main","coroutine"))
```

We shall explain this point further in a sequel[BL19].

12.8 MODEL EXTRACTION

If we pursue a concrete strategy of separating timescales and extracting invariants from the chaff of noisy variation, we can expect to infer causal and conceptual relationships over long aggregate times by learning. Learning is a process that happens across several timescales, as noted in [Bur16c]. In modern Machine Learning parlance, we would say that acquiring and stabilizing training data is a long term process, while recognition and classification is a short term process. Monitoring tries to achieve both processes as an unsupervised in band single-scale process, so it has to deal with the instabilities in band too.

The spacetime model in [Bur16c, Bur17c] allows us to define a partial ordering of semantics, represented as agents in a virtual knowledge space. The future and past cones are generated by the first two spacetime semantic relations: for generalization or scope and causal order. Ordered relationships are the most important ones because they tell the stories we seek. Data may arrive in incidental order, for a variety of reason that involve

causal mixing. We need to extract the intended order of system cooperation from the incidental or unintended order of side channels that muddle behaviour.

12.8.1 INVARIANT SEQUENCES FORM EXPLANATIONS

The invariant stories that can be generated about a process require us to first be able to promise identifiable and repeatable phenomena. In order to generate a map of invariants, we need to not only identify them, but consider what they can promise about one another, so that we may position them in relative terms. In [Bur16c, Bur17c, Bur17a, Bur17b], it was shown that we can plausibly define four coarse kinds of semantic relationship based on elementary spacetime considerations. These ought to apply between pairs of agents in any distributed system (see figure 12.5). The significance of these types is that they are all we need to algorithmically reason about different kinds of process relationship. Any more specific information may be contextual relevance, but does not affect the causal structure of a story.

- i) CONTAINS: localization in spacetime (scope of containment or ordering by scale)
- ii) FOLLOWS: order, causation (Markov processes or order by influence)
- iii) EXPRESSES: local distinction (scalar attribute at any scale)
- iv) NEAR TO: measure, distance (assessment of distance at any scale)

Notice that order is distinct from distance, i.e. direction and proximity are different concepts. These concepts are not clearly distinguished in a vector space.

The four kinds of relationship are illustrated in figure 12.5 and detailed in table 12.1. This identification of types offers an enormous dimensional reduction algorithmically for the characterization of system trajectories. Agents may promise exemplifiers, symbolic and metric discriminators, or regional classifiers. The relationship between a discriminator iii) and a classifier iv) is subtle, and is easy to promise inconsistently. The essence of expressing an attribute is to label the type, which can be combined with something else to form a union of different promises (a kind of semantic chemistry). A promise of containment, on the other hand, is a promise of belonging to a common class. These two promises therefore represent assembly versus classification of agents[178].

12.8.2 PROMISING SEMANTIC MAPS

The four spacetime semantic relationships, described in [Bur16c, Bur17c] may be assigned between pairs of concepts, originating by signal (+) or by inference (-), entirely at the behest of an observer, and according to the following ‘selection rules’:

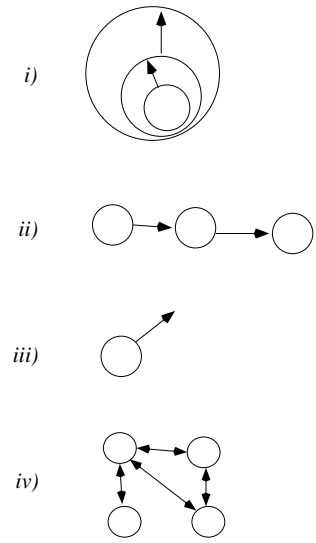


Figure 12.5: The four kinds of promise that spacetime can express: i) containment, ii) succession, iii) local attributes, and iv) proximity. Although we can distinguish different sub-types of these four, it's hypothesize that the four are necessary and sufficient for describing observable phenomena.

1. **Distinguishability**: Descriptive properties that distinguish, describe, and embellish the name of a concept are EXPRESS promise types. These are scalar promises used to explain attributes that may form compositions of attributes for aggregate 'hub concepts'.

   For example: a banana may express the colour yellow, ripeness, and sweetness. It does not express fruit or Del Monte.

2. **Generalization**: membership in classes and informal categories use the CONTAINS promise type. These express subordination to one or more umbrella concepts, and superordination to instances and exemplars of the named concept. Generalization is strictly transitive.

   Generalization is not as in taxonomy: a concept may have any number of generalizations, i.e. there is no unique typology to concepts. The utility of recognition lies in the overlapping nature of classes[Bur05a].

   For example, a banana is generalized by fruit and desserts, and has instances such as Del Monte. It does not express these as attributes.

TYPE	FORWARD	RECIPROCAL	SPACETIME STRUCTURE
ST 1	is close to approximates is connected to is adjacent to is correlated with	is close to is equivalent to is connected to is adjacent to is correlated with	contiguity PROXIMITY "near" Synonym similarity
	FORWARD	RECIPROCAL	SPACETIME STRUCTURE
ST 2	depends on is caused by follows	enables causes precedes	ordering DIRECTION "follows"
	FORWARD	RECIPROCAL	SPACETIME STRUCTURE
ST 3	contains surrounds generalizes	part of / occupies inside aspect of / exemplifies	boundary perimeter AGGR MEMBERSHIP "contains" / coarse graining
	FORWARD	RECIPROCAL	SPACETIME STRUCTURE
ST 4	has name or value characterizes represents/expresses promises	value of property property of represented/expressed by	qualitative attribute DISTINGUISHABILITY "expresses" Asymmetrizer

Table 12.1: Examples of the four irreducible association types, characterized by their spacetime origins, from [Bur16c]. In a graph representation, 'has attribute' and 'contains' are clearly not independent, so implementation details can still compress the number of types.

3. **Dependency**: promises of dependency—prerequisite or follow-up concepts are FOLLOWS type promises. They may link concepts of any type into some meaningful order, by any interpretation of the observer.

   For example, the beginning precedes the end. "One" precedes "two" which precedes "three", etc. Dependency is usually transitive, but may contain loops (in feedback cycles).

4. **Similarity**: the degree of similarity between two concepts is an assessment that may be promised by any observer, to represent a degree of similarity or closeness. This is represented by promises of type NEAR. This is an ad hoc assessment and should not be taken too seriously.

   In the case where several agents form an agreement about the metric distance relationships between concepts, these assessments may form the basis for a shared local coordinate system.

The semantics of these relationships are not automatically orthogonal to one another,

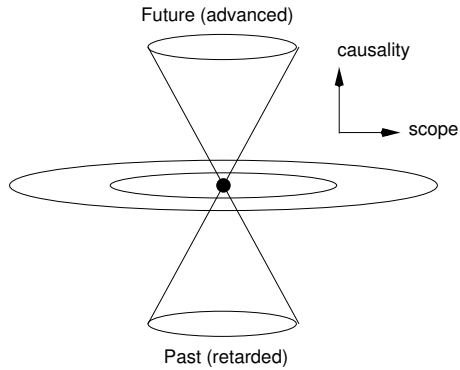


Figure 12.6: The propagation cones indicate the past and future as 'timelike' trajectories generated by the causal relationship, and semantic scale or scope of meaning accumulated in 'spacelike' directions around the average time axis in rings of increasing generalization.

so we have to maintain the incompatibility of the types by assignment. The local promises are mutually incompatible, which is to say the no two agents may promise more than one of the three types CONTAINS, FOLLOWS, EXPRESSES (or their inverses[179].

- EXPRESSES is incompatible with CONTAINS.
- CONTAINS is incompatible with FOLLOWS.
- FOLLOWS is incompatible with EXPRESSES.

The semantics are easily illustrated with an example. The concepts blue and yellow are expressed by objects that combine them as part of their identity: e.g. a blue and yellow pattern, like the Swedish national flag, and green paint may be composed of blue and yellow paint, but neither the Swedish flag nor green are generalizations of blue and yellow. The concept of colour, on the other hand does not express blue or yellow, but generalizes them as members.

The promise of proximity is slightly different:

- NEAR is potentially compatible with any of the above, since it is an informal assessment of non-locality.

  The assessment of proximity between agents may seem to imply something about the orthogonal semantics above, but this is ambiguous (see figure 12.7). For example, because proximity is a type of relationship, not a standardized metric

constraint, relations may vary in their interpretation:

$$A \text{ EXPRESSES 'close to B'} \tag{12.32}$$

$$A \text{ EXPRESSES 'close to C'} \tag{12.33}$$

$$\text{'close to B' FOLLOWS 'close to C'} \tag{12.34}$$

Together these might suggest that A, B, C all lie in a certain region and that there must therefore be a category (dotted line in figure 12.7) that generalizes all of them. That kind of inference is dangerous, because it is based on coarse inference,

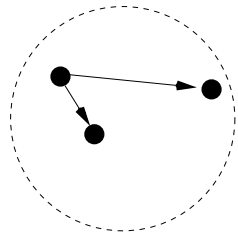


Figure 12.7: The assessment of proximity between agents may seem to imply something about the orthogonal semantics above, but this is ambiguous.

These selection rules can be applied in order to join similar objects into hubs. Each observed instance maps to a hub that can be broken down into atomic concepts by expression. Containment promises are generally learned on a much longer timescale (e.g. added by human expertise), and causal dependency promises are added by processes that generate them or observe them.

12.8.3 STORYTELLING FROM SPACETIME SEMANTICS

Once constructed, the graph may be parsed to generate stories, or automated reasoning. A reasoning process may be viewed as an expansive search along alternating (+) axes (causal outcomes that are related by generalization or exemplification by specific instance), and tempered by elimination by relevance criteria (-).

- Starting from a topic of interest, we follow promises of type FOLLOWS independently in the forward and backward directions, to explore the causal cone (figure 12.8).
- Arriving at each new concept, we follow promises to generalize and specialize the concept to find all links arising from the collective generalized concept, and follow

these along different story paths. In other words, we multiply the number of stories by conceptual associations that imply examples of the same idea—expanding the scope of meaning without going off the rails.

Such promised relationships cannot be easily found by in band machine learning techniques: this is an orthogonal and complementary method, but learning may form the basis for collapsing experience into an arrangement of similar concepts on longer timescales (see figure 12.8).

The algorithmic rules for parsing stories from the concept graph come in several forms. A conceptual reasoning search might be bounded at the start (retarded), at the end (advanced) or at both ends (causal). The first two are a form of brainstorming that ends with a single concept: 'tell me all about X'. The latter case asks for a specific explanation of 'Y given X'. There is no unique path for any search, in general. The paths most frequently trodden, i.e. have the most frequently observed transitions, or most frequently searched for concepts, become 'classical paths' and may be favoured, someone analogous to a PageRank search[PBMW98, BBCEM10].

Loops in causal relationships may be significant, so we should detect them. Some loops may be errors of identification, others may be cyclic reasoning (e.g. self-consistent ideas, like eigenvalue problems).

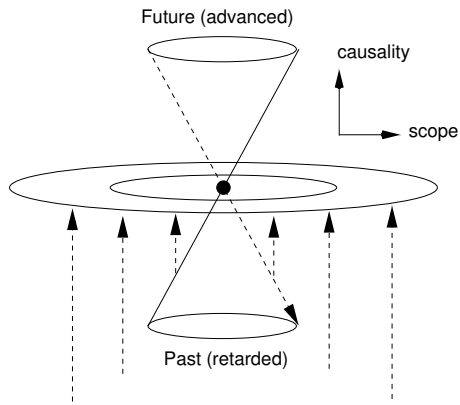


Figure 12.8: The scope of knowledge about spacelike information is accumulated as memory from past events propagated into a model of the present.

12.8.4 MODELS, SHARDING, IDEMPOTENCE, AND FORGETTING

From sampling of data at the edge of a network, to actionable insight, there is a chain of reasoning to monitoring that starts with observability and ends with the deletion of irrelevant and antiquated data:

1. Data collection,
2. Stability or convergence to fixed points,
3. Model extraction,
4. Classification into buckets,
5. Controlled forgetting.

Each of these steps plays an important role. Data collection provides the basic observability to trace systems at different scales and tell stories about them that bring valued insights. The convergence of phenomena to fixed points is an incredibly important principle in dynamics, but one that receives too little attention[180]. When systems fly all over the place, they are not telling us anything significant. It's only when they converge onto repeated patterns or stable attractors that we can build on them as part of a reliable. Models need to expose those differences. Today, there is a fascination with using machine learning to try to expose such fixed points, but the technique is only possible if there is sufficient stability. When monitoring reaches a level of maturity in IT, we will place as much value in attending to semantics as we do in recording noise today. Data that have the same semantics need not be recorded twice. Once one has identified the invariants of a system, these can be made idempotent. Model identification classifies inputs into discrete alphabets. Repeated symbols can be compressed, and if they are repeated no harm need be done if they have fixed point semantics.

For example, it is not a problem if we accidentally collect the same data twice as long as they map to the same place. Storing the same data twice is idempotent unless we are counting frequencies. Frequency counting can be made idempotent by labelling intervals. The general principle is that we should engineer data sources to permit convergence. Promise Theory reveals the mutual responsibility for information transfer between sender and receiver.

Principle 11 (Convergent data)**.** *The safest way to avoid data inconsistency is to design messages in such a way that repeated messages always map to the same location and update them without breaking a promise.*

Fixed points lead to stable models, which lead to efficient indexing of knowledge. This helps in scaling storage (e.g. in sharding), and it helps in fault tolerance. When data are recorded around proper index points, it doesn't matter if data get delivered multiple times (idempotence) or even out of order: everything will find its proper place in the end. Model extraction tells us how to compress the data into an alphabet or catalogue of meaningful and significant ideas, and therefore separate into buckets or shards.

Finally, perhaps the most important issue of all is how to forget what is no longer of value. Keeping data and even models around forever is a senseless squandering of resources and an irresponsible and unsustainable use of technology. One wonders how many of the photos now being eagerly accumulated in the cloud will be preserved in ten years' time. The same is true of monitoring data that were collected last week. If we don't understand the timescales, context, and relevance of data, then we have no business collecting it, because it cannot tell us anything of value. A policy for forgetting can usually be built into a definition of context from the start, e.g. through finite windows and sliding sets, running averages, and so on [Bur02].

Example 293 (Model based collection)**.** *In CFEngine, weekly data were mapped idempotently to a finite number of buckets marked by 5 minute intervals throughout a week, based on a prior measurement survey using autocorrelations. After a weekly period, the buckets would wrap around, like a clockface and data would update the corresponding image in the map. This approach was able to promise a limited stability of expectations, as well as automated forgetting (constant weight gradient of temporal history), and thus effective garbage collection*

What's remarkable is how few of these issues actually get any attention in the literature. One hears arguments like 'its cheap to keep all data forever'—which smacks of the sudden realization of global warming or the plastic crisis. Every time we advocate increasing something, we need to think about the balancing garbage collection process.

12.9 KNOWLEDGE SUMMARIZED

The collection of accurate data is not in question. Today, there is increasing interest capturing 'digital twin' representations of agents in the real world, with every detail available for possible inspection. No one could resist the idea of such knowledge, unless it invades their privacy. The question in technology monitoring is rather whether every detail should be centralized and whether data can be compressed without loss.

In this summary of what can be observed about distributed systems, we see that tracing events back to 'root cause' is an ill-defined problem, but tracing back a significant likely cause is indeed possible with careful labelling (especially of time). This kind of

labelling is not commonly provided in current tooling. Assuming access to data, we might hypothesize that a complete monitoring system would promise to:

- Separate timescales.
- Identify the alphabet of system invariants
- Capture local histories of instances, in the context in which they happen.
- Identify significant events at different scales and measure their invariance.
- Tools for reconstructing and backtracing of histories from local data.
- Tools for generating semantic past-future cones for causal reasoning.

Today, many IT monitoring systems transmit raw data in large quantities to a central point for analysis, without attempting to alphabetize the data before transmission. In effect, by ignoring the existence of a model (summarized by an alphabet of non-overlapping signals) one is repeatedly sending the same model over the network again and again, wastefully, and to no gain. If we can classify observations at their source, and condense them into an alphabet of signals, a vast data compression can be accomplished for both faster recognition and potentially greater semantic content. That will be the subject for a sequel [BL19].

It should be clear that nothing about the ability to trace systems enables full reversibility of state, which should be considered difficult to impossible, depending on scale[BC11]—so ultimately monitoring may be of little value. More value could be captured by building intrinsic stability into systems in the first place.

The elephant in the monitoring system is an essential attitude in the industry concerning the purpose of monitoring. Systems are only sustainable, knowable, and predictable when they seek stability—not when they labour under the burden of intrusive inspection. For a lot of practitioners there is a conflict of interest here. If we seek to measure all that is random or unstable, by oversampling and consuming resources wastefully, it will be neither stable nor sustainable. Consensus protocols, for instance, promise semantic stability, and are popular (if somewhat over-used) in software engineering[181]. They draw attention to a preoccupation with semantics in software engineering, i.e. a desire for stable qualitative outcomes, at the expense of quantitative delay. Software engineers seem not to trust concepts like intrinsic dynamical stability i.e. systems that promise to converge to predictable quantitative outcomes. Monitoring tends to treat all software as adversarial, and we put more faith in ill-designed monitoring than in an initial software design. This is a paradox that will inevitably lead to big surprises and catastrophic events.

CHAPTER 13

AFTERWORD

Bringing this collection of notes to a close, its easy to conclude that there are more than a few significant principles we can state about systems, but there remain many gaps left to discern and be filled for a fuller engineering understanding of human-machine systems. Unlike the idealized world of physics, it is more difficult to separate semantics and dynamics clearly—not only because these tend to span multiple scales, but also because the number of independent process channels is typically much larger for specified systems than for faceless 'ergodic' systems about which we can divine statistical knowledge. Systems share more with complex electronics or biology than with the mechanics of non-decript bodies.

TO THE READER

I hope the purpose of these notes—to bring together traditionally incompatible issues under a single framework—is helpful to some. I understand that I ask a lot—readers who find value in these pages are not typical 'just show me the result' engineers, nor are they 'keep my head down and publish classical papers' scientists, let alone 'let's all talk about it together' team managers; the lectures are aimed at a more ambitious audience of scientific problem solvers who strive to forge a deeper understanding—and in an age of soundbites and impatience. That will not be to everyone's taste.

Through many examples, which I've assembled and tried to track over many years of consulting and problem solving around the world[182], I think one can see, convincingly, how both dynamics and semantics can be brought together under a common umbrella, and their concerns (sometimes separable and sometimes not) reconciled in a 'simple' way. This has a value in its own right, since it has been staunchly denied by many in the

fields of both Computer Science and Physics (at least while I was climbing the academic ladder). The principles discerned can be applied to any system to understand and describe properties on all scales. Systems can be described as phenomena with intent and outcome included. It won't necessarily be easy, but it is achievable.

For me, this has been an enjoyable though exhausting effort to try to bring some order to the subject. I realize that there is a lot of scope for improvement, but that work almost certainly needs the contributions of others who are convinced of plausibilty of the mission. Perhaps these two volumes might inspire those, while I move on to expand this approach to the next compelling frontier, which is crying out for a formalism that brings together semantics and dynamics: *social systems*. From here on, it's up to a new generation of researchers to take on board the successes and remedy the gaps—to assume the mantle of developing a more integrated theoretical picture and push for future progress. There is no shortage of topics remaining, nor of details to complete in the material here.

SUMMARY

Once again, in summary, here are some of the key findings:

1. By designing systems to yield a promised outcome (by promise, not by obligation or imposition) we maximize causal predictability.
2. There is no need to separate human and non-human parts of a system: analyze them together.
3. Choosing the right agent to keep each promise should follow an assessment of fidelity building on experience rather than assumption.
4. In general we must be aware of where the boundaries of a promise are: there are different effective boundaries for different kinds of promise. If we pretend they are all the same, we will likely compromise the integrity of some.
5. We can build for self-regulation and self-protection by choosing pull-based systems over push-based systems.
6. By avoiding systems, which amplify impositions, we avoid the magnification of inevitable errors. During amplification, we can use pull based methods for stability and context.
7. We choose centralization principally for calibration and separation of timescales.
8. We choose decentralization principally for redundant robustness and resistance to change.

POSTSCRIPT: SUSTAINABILITY AND WASTE

Although I avoided a direct discussion of the issue in these notes, it seems clear to me that too many of today's human-designed systems are sluggish and obese leviathans, built on principles that have not adapted significantly to modern times or its many technological opportunities. Wasteful and vulgar examples abound, where commoditized convenience has been championed over engineering excellence. Systems and processes are in desparate need of streamlining, for environmental as well as social reasons. The waste produced by our civilization—both in disused material and heat entropy—is both indefensible and unsustainable. We see the effects of the Industrial Revolution written on accelerating climatic processes, and the next crisis will surely be the legacy of the Information Revolution. It will concern unaddressable data storage and computational waste. Software engineers receive little training about the processes that make computers work, or are aware of the enormous energy cost of wasted computation. I hope more attention can be paid to efficient and thoughtful design, in the coming decades, both in physical and virtual systems, with care and parsimony triumphing over today's brute force methods. No engineer, or citizen, can afford to take resources for granted.

ACKNOWLEDGEMENT

I'm grateful for comments and discussions with a number of people, some of whom worked with me on various projects, others who were kind enough to read my notes and formulations. These include Adrian Cockcroft, Paul Borrill, and Daniel Mezick, who gave specific comments. Naturally, I have had many important and pivotal discussions with my collaborator Jan Bergstra over the years in which we have developed Promise Theory together. I should also mention (but shall nonethelss refrain, for the sake of privacy) the discussions and examples adapted from the experiences together with a number of engineers in the field—these practical cases have driven this entire second volume. None of the above bear any responsibility for errors, omissions, lack of clarity, or interpretations contained herein.

APPENDIX A

EMPIRICAL EXAMPLES: CASES AND REMEDIES

The following is a selection of faults observed in the field for service providers, in boldface. Proposed resolutions, based on a promise-compliant architecture are described for each.

A.1 OBSERVED FAULTS AND THEIR AVOIDANCE

A common theme in these examples is the misplacement of trust in low fidelity agents. Buffers and noise reduction techniques (dynamic and semantic averaging) may be employed to reduce the risk of faults.

A.1.1 KEY PARAMETERS WERE UNEXPECTEDLY MODIFIED

Examples from networking equipment:

1. **Some switch parameters which should be OPEN are CLOSED, which was not realized until service crashed. e.g. routine backup, MAC self-learning function, etc.**

   No documented promise was made to provide these services, so no assessment of the state was made either, meaning that the state of the switch parameters are ad hoc, and not automatically repairable.

   This is common in models based on change, instead of desired outcome/state. (We make models about how we can get involved, not about what outcomes we desire.)

By using a policy model, and a self-repairing agent with regular maintenance (e.g. like that of CFEngine), because we cannot guarantee no tampering, this kind of error can be eliminated within a minimum time window. e.g., if an agent checks the state every T minutes, then the Mean Time To Repair (MTTR) is $T/2$.

2. **Some global parameters were modified by mistake and affected large scale services.**

   Strict routines for non-tampering with devices can be introduced. A barrier or buffer between humans and device states is needed so that human whims and spurious actions are isolated from high risk outcomes. The risk is higher when instructions and changes are partial or relative (context-free) changes, rather than complete repeatable outcomes. The loss of repeatability is a key problem.

   For a software system, all changes may be version controlled changes to a context aware policy that describes a clear outcome for each context. The policy is then implemented and maintained by a self-repairing agent with convergent outcome.

Comment 28 (Model based promise-making)**.** *The key principle that is violated in these examples is that a user or consumer of a service has clear expectations of a promise being kept, but no promise was actually made: ad hoc configuration changes were once by someone, without verification.*

By employing a model-based promise language, a promise can be made a kept formally and verifiably. Changes can be versioned and tracked, pre-tested, etc. This is a standard approach used for:

- *Compliance with regulations.*
- *Compliance with Service Level Agreements.*

A.1.2 WRONG MODIFICATIONS APPLIED IN BATCH

Examples:

1. **A user wants to modify the LAC attribute of a single base station[183], but actually all LAC of base stations was modified, because the user select the ALL parameters by mistake.**

   Inadequate buffer between human low-fidelity agent and amplified outcome. A high-impact control parameter has inadequate safeguards to accidental change. e.g. the pilot ejector seat on a plane should not be located next to the intercom radio button. A small mistake could lead to catastrophic consequences.

A noise reduction method is needed, based on semantic averaging. For all amplified or catastrophic changes, implemented by low-fidelity agents, there should be multiple opinions, e.g. 'dual key' launch codes, or confirmation from a second (multiple) source. This is why pilots have co-pilots, missiles have dual launch keys, etc.

2. **A user wants to disable a device in a tree model network, but selects the upper level node by mistake, so all the devices under this node are been disabled.**

   The user is implementing actions directly instead of decoupling reasoning from action. Reasoning should be slow, and deployment fast only after validation. No validatable promise was made about what devices should be active, or the agent's assessment of the state was erroneous, perhaps due to insufficient fidelity or resolution in its assessment of the context.

   Again, this is a design error of placing radically different things close to one another (control channel separation), and trusting an unreliable agent without asking for confirmation? A warning message is the minimum barrier to a fault: "warning you are about to delete ... Are you sure?" A two-factor authentication with one-time pad codes can be used to authorize such events. This is common in online banking. For high risk items, dual keys can be required.

   This could also be dealt with by fault tolerance, ensuring that an erroneous change did not have major consequences, or requiring dual-key activation.

Comment 29 (Big hammers need supervision)**.** *Big hammers or coarse changes refer to to changes that are sweeping and extensive. The key principle violated here is in coupling high impact consequences to a low fidelity agent, without verification. Spurious choices are thus amplified with high fidelity by automation (see section 6.5.3), This is like giving a child a sledgehammer to open a bottle.*

Role Based Access Control (RBAC) can set limits on what individual operators are allowed to do, with or without confirmation. Some operations are harmless and one can trust even a low-fidelity agent to implement them. When destructive operations are implemented with a high degree of amplification, but high fidelity, semantic averaging can be used (dual authorization).

As part of a version controlled language of promises, changes can be made by using a version control system (e.g. git), and filing a change request. A process supervisor then merges to offer confirmation. The code can be checked for consistency. Provided the workload on the supervisor does not lead to low fidelity verification, this is usually good enough.

A.1.3 DELETION OF KEY RESOURCES BY MISTAKE

More examples of inadequate separation of (potentially) low fidelity human agent from a dangerous outcome (misplaced trust):

1. **Deleting a route by mistake.**
2. **Delete a IP path by mistake.**
3. **Some configuration files deleted by mistake.**

   No manual change allowed. Model based change with version control is used, and the "diff" (delta change) is authorized by duplicate sources of truth before being deployed.
4. **The user deletes the wrong resources, or he did not realize there are key services on these resources.**

   No manual change allowed. Model based change with version control is used, and the "diff" is authorized by duplicate sources of truth before being deployed. If resources are in use when deleted, this can be detected from the model documentation. Knowledge-based tools and keep track of these.

Comment 30 (Dependence tracking and change impact)**.** *In a system, any change can have consequences. A map of knowledge about these dependences can be inferred from a promise model.*

A key question to ask is: why would a user end up in this situation of wanting to delete a resource? What leads to this decision? Then, what context information does he/she need to make the decision?

A.1.4 PARAMETERS OR ATTRIBUTION ARE WRONGLY CONFIGURED

Examples:

1. **The attribute of a port has been wrongly configured. It should be an optical interface, but was configured as an electrical port.**

   Low fidelity agent makes an erroneous assessment of current state, or erroneous response. Replace the agent/promise with a higher fidelity agent/promise.

   This is an error of understanding or intent. This could be due to inexperience of the human operator, or bad naming conventions. Clear naming conventions allow different components to promise their functions clearly. It should not be possible to confuse one type of component for another.

2. **An IP address been wrongly configured, so the service is directed to the wrong destination.**

3. **The ID of a device is wrongly configured.**

4. **The route map table is wrongly configured.**

   These are errors or intent. The lack of a proper model of the system, verified and pre-tested, and rolled into production without testing. Again, clear naming is important for human comprehension and recognition.

   If the configuration errors are made in the policy model itself, this suggests a lack of understanding of the system, and a lack of testing.

Comment 31 (Proper identification through classification and naming)**.** *The fidelity of human agents depends on their ability to make sense of a situation. Clearly distinguishable names and identities aid human comprehension.*

When systems process too much raw information, this leads to delays, fatigue, and loss of human comprehension. By using patterns to abstract/represent many things as a single pattern, we can make generic rules or wildcards to amplify the effect of a small amount of information (see earlier remarks about amplification).

A.1.5 CONFIGURATION OF KEY RESOURCES OR PARAMETERS IS MISSING

Examples:

1. **APS group 1022 of 106 did not get deployed, so a service packet did not send at both working tunnel and protection tunnel, so the protection tunnel lost function.**

2. **The VLAN was not divided and both two NEs connected to one maintenance LSW cause the issue and service crash.**

3. **Forgot to change IP after software upgrade, service abnormal.**

These are all examples of a lack of clear promises, implemented by high fidelity agents.

A.1.6 INCONSISTENT CONFIGURATION

Examples:

1. **The configuration on master and slave board are not consistent, the configuration on slave board is wrong. But this mistake can only be found when the switch from master to slave is happen. At this time, the service is affected.**
2. **The docking parameters between two devices are configured inconsistently. Such as frame format, Name, policy of two devices are not consistent of conflict.**

These issues can be dealt with by model-based promised policy. Fault tolerance of system inconsistency is the most important property of a system. Full consistency cannot be guaranteed at all times. It is affected by network partitions and latencies.

A.1.7 NON-OPTIMAL CONFIGURATION

Examples:

1. **Parameter is too small or too big (but still in valid interval)**
2. **Some parameters are configured too small, it's still in valid interval but have high risk. In the condition of certain business volume, it will trigger incident.**
3. **The network is configured in a non-optimal way.**
4. **VLAN is configured and split in non-optimal planning, so the service quality is affected.**

Optimization requires an understanding of how causal factors lead to desired outcomes. This can be learned over time by human analysis, by collecting data. For this, we have to combine intended state with actual state and output.

A.2 CHALLENGES

Examples:

1. **A mistake has been introduced into the network long ago, but was undetected until the service was seriously affected, leading to many incidents.**

   Manual tampering must be forbidden. Errors in the model can be verified by acceptance testing, or oversight.
2. **A wrong configuration causes bulk parameters or data be modified. Sometimes the error can be found immediately, but it will take too long to change the parameters or data back to the initial status, leading to long term service disruption.**

Agent based automation, based on policy models minimized this risk. Errors to policy itself can be addressed by testing, or consistency checklists. The move towards disposable container technologies allows quick reversal of configuration in stateless systems. A dependence on runtime state leads to fragility here.

3. **An error been introduced into the network, because of the lack of complete validation rules. But it is a huge work to work through the code to find which is lacking.**

   This is an intentional error. Validation is simpler in a promise model.

4. **Configuration errors take too long to discover.**

   Regular automated verification should be run at regular intervals that matches the likely rate of fault (MTBF).

5. **There are different vendor products, and some configuration is associated with vendor product, but some with service type. It's hard to find a solution or technology to help all products to resolve the problems.**

   A multi-vendor (open source) agent is straightforward. This is the approach used in CFEngine across dozens of different systems and vendors systems.

A.3 COMMON DATACENTRE FAILURE MODES

1. Device failures - bad batch of disks, Power Supply Units, etc.
2. CPU failures - cache corruption, math errors
3. Datacentre failures - power network, disaster
4. Routing failures - DNS, Internet/ISP path
5. Vendor diversity

A.4 SOFTWARE FAILURES

1. Time bombs - counter wrap, memory leak
2. Date bombs - leap years, seconds, epochs etc
3. Expiry - certificates expire
4. Revocation of credentials - Certificate authority failure

5. Security exploit
6. Language bugs - compile time
7. Runtime bugs - JVM, Linux, Hypervisor
8. Network bugs - routers, firewalls, protocols
9. Versioning mismatches
10. Configuration errors
11. Overload
12. Build was interrupted or failed to complete
13. Timeout

A.5 PREDICTING NEW FAILURE MODES AT SCALE

- Certain unpredictable occurrences only occur at a macroscopic scale.
- What if the 'weather' of contending resource scheduling becomes so heavy that it begins to create effective couplings though third parties? Then a macroscopic superagent can become an effective boundary condition for a microscopic process. This leads to non-linear feedback
- Second order effects become more likely, amplified at scale.

A.6 CAN WE STABILIZE SYSTEMS WITHOUT BREAKING THEM FURTHER?

Suppose we design a system based on a set of promises and assumptions. This leads to an unexpected outcome. Is it possible to say anything about how to stabilize the outcomes so that we could make/keep new promises to constrain the behaviour, without destabilizing the existing promises? This is the software bug fixing problem. There is no general answer to the question. Any change could negatively impact another exterior promise; this depends on the way the interior promises are coupled inside the relevant agent boundary.

For example, modular separation of concerns is a commonly assumed strategy here. If two parts of a system seem to be separable, we tend to assume that they are not in contact, and will not affect one another. This is simply wrong. However, we often assume

that the risk of fault propagation is reduced, which is simply naive, as it ignores first and second order interactions.

APPENDIX B

SUMMARY OF *do* AND *don't* IN SYSTEM DESIGN

Summarizing the analysis of fragilities with respect to promise keeping from the foregoing sections. This describes a mixture of general suggestions for stability of both dynamics and semantics.

Learning (i.e. so-called 'anti-fragile') systems become increasingly 'situation aware', such that predictability and robustness can grow. On the other hand, we can avoid designs that rely on memory or suffer from system memory loss, such as unstable, and non-linear designs that lead to mixing and loss of predictability.

Comment 32 (Warning, take responsibility)**.** *In the following, readers are warned to interpret "DO" and "DO NOT" not as an absolute directive, but as a guard rail suggestion. If this book advises anything, it is surely that simple rules make no sense without context and consideration of tradeoffs.*

B.1 FAULT RELATED PRINCIPLES

1. DO Reduce fault surface of system $|in\rangle$.
2. DO Reduce amplification of effect $\langle out|in\rangle$.
3. DO Reduce degrees of freedom with promised constraints.
4. DO Equilibrate degrees of freedom.
5. DO Increase sampling rate for fault detection, Nyquist frequency.

6. DO Decrease MTTR Δt.

7. DO add preventative processes to increase MTBF (increase tolerance, or slow system velocity).

B.1.1 DEVICES (MACHINE PROXY AGENTS)

1. DO Avoid operations that apply relative changes (deltas). Make all operations converge (idempotently) to a fixed desired end-state.

2. DO Make all behaviours fault-tolerant of their own states, and of dependencies. Every operation should have an outcome that is aligned with goals, despite unexpected conditions.

3. DO Make state self-maintaining.

4. DO make devices self-recovering in case of fault.

5. DON'T rely on alarms and human intervention in critical situations (see comment 9). DO pre-plan for disaster scenarios and build in safe rational responses that converge towards a desired end-state.

6. DO Avoid hard dependencies. Cache/store all pre-requisites at the point of requirement before acting to prevent unpredictable failure.

7. DON'T allow any state of the system to block a control channel to an agent (human or device).

8. DON'T (ever) assume that all devices are synchronized or in a consistent state.

9. DON'T assume availability of any device.

10. DO make agent models. DON'T make agent-less models. DO apply repairs and controls at the fault location, never by remote control (because this compounds uncertainty).

11. DO ensure that all devices and systems fail safely and predictably.

12. DO consider the need for redundancy and DO evaluate the need for multiple opinions and quorum (semantic averaging). It is most stable to be tolerant of all variation in a system.

B.1.2 HUMAN AGENTS

1. DO assess all promise proposals, in the light of all possible contexts, before making them.
2. DO review promises regularly (continuous assessment and delivery) and consider possible missing promise coverage.
3. DO change promise intent slowly, but contextual details continuously, with version control, and test validation, to trace who changed them and why.
4. DON'T use transactional thinking. Don't pretend that something is transactional if it is non-atomic and without proper isolation. Few system scenarios fit these conditions.
5. DON'T allow any state of the system to block a control channel to an agent (human or device).
6. DO consider the need for redundancy and DO evaluate the need for multiple opinions and quorum (semantic averaging). It is most stable to be tolerant of all variation in a system.

B.1.3 HUMAN-MACHINE INTERACTION

1. DO limit the focus of any interface to a clear goal to increase situation awareness.
2. DON'T provide direct (CLI or GUI) access to mission critical functions. Use policy to buffer changes through an intermediary (with authorization, verification and validation).
3. DO require confirmation of intent by compiling intentions and validating them before execution.
4. DO provide graded alarms as advisories. DON'T plan to rely on new reasoning in a crisis (see comment 9).
5. DO avoid relying on specific ordering of events or promises being kept. Fault tolerance of ordering and preconditions will avoid a major source of brittleness.

Direct action is always more risky than buffering decisions through a proxy. The aim is to allow ourselves time to evaluate reasoning before executing or acting on the decision. We want to avoid risky situations, where rapid decisions insert new instabilities. Decisions on the timescale of changing environmental events leads to non-linear coupling, and hence increases the likelihood of instability. A good design goal is to separate slow reasoning from fast pre-determined response.

B.2 THE VALUE OF PROMISES

The value of links in a network depends on the promises they make. The value of a promise is a form of assessment[BB14a] that any agent can make independently. We write an assessment of whether a promise was kept

$$\alpha_i(A_i \xrightarrow{b} A_k) \in [0,1] \tag{B.1}$$

to mean the assessment by agent A_i that the promise from A_j to A_k was kept. A valuation is an estimate of what a promise is worth to an agent. This may of may not depend on the assessment of to what extent the promise is kept or not. Every agent assesses on its own calibrated scale. If we want a common currency valuation for all parties, this has to be calibrated by a single agent according to its scale.

The interpretation of value is also an individual judgement that relies on trust, and may be based on accounting of the assessments over time (reputation)[BB06].

$$\text{My reputation} \propto \sum_{\text{you}} \Big(\text{you} \xrightarrow{-b} \text{me}\Big) \tag{B.2}$$

In words, my reputation is proportional to all the number of 'you' who (publicly) promise to accept my promised service. Even unilateral promises may have some value:

VALUATION BY X	ABOUT PROMISE	REASON FOR VALUE TO X
Me	me $\xrightarrow{+b}$ you	An reputation building investment
Me	you $\xrightarrow{+b}$ me	A service that might help me
You	me $\xrightarrow{+b}$ you	A service that might help you
You	you $\xrightarrow{-b}$ me	You need the service now

Cooperative relationships are usually based on conditional assistance[BB14a], and take the form of a conditional equilibrium:

$$S \xrightarrow{+S|M} R \tag{B.3}$$
$$R \xrightarrow{+M|S} S \tag{B.4}$$
$$S \xrightarrow{-M} R \tag{B.5}$$
$$R \xrightarrow{-S} S. \tag{B.6}$$

in words, S promises R a service, if it receives payment M; and R promises to pay M if it receives service S. In a network without trust, this is a deadlock. But if any agent trusts the other enough to go first, it is a cyclic generator of a long term relationship. Such relationships imply lasting value, as known from game theory (for a review see

[Bur13a]). Both agents also promise that they will take (-) what the other is offering unconditionally. This is a signal of trust. Valuations are not necessarily rational to anyone but the agent that makes them, and are unrelated to cost.

The economic value is that something is exchanged, which requires a binding of both + and - promises.

$$S \xrightarrow{+S} R \tag{B.7}$$

$$R \xrightarrow{-S} S \tag{B.8}$$

Both agents recognize the value of the other party, so the value exchanged is proportional their assessments that the promises were kept:

$$v_S \propto \alpha_S(S \xrightarrow{+S} R)\alpha_S(R \xrightarrow{-S} S) \tag{B.9}$$

$$v_R \propto \alpha_R(S \xrightarrow{+S} R)\alpha_R(R \xrightarrow{-S} S). \tag{B.10}$$

In a community where such transfers are made often and between arbitrary pairs of agents, standards of valuation are equilibrated, and may be exchange in league with a calibration agency (e.g. a bank or government). Thus, in a well-connected community, with a spanning infrastructure, we may posit that the value of a one-way transfer is simply

$$v_C(\Pi_{ij}^S) = c_S \alpha_i \alpha_j, \tag{B.11}$$

where c_S is the currency value of a perfect service relationship S, and α_i is an impartial assessment of the probability with which A_i will keep its promise to give or receive S.

CHAPTER NOTES

NOTES

[1]This second volume brings together terminology from a wide range of fields, and it's my hope that readers will appreciate the subtlety with which words have been defined and used. I've tried to strike a balance to preserve the intended meanings of words in common usage. These choices are not as haphazard as in some other works, so I encourage readers to pay attention to the definitions and consider them carefully.

[2]Although Logic and Category Theory have achieved arcane recognition, they have yet to succeed in offering any practical insights as they ask too rigorous a discipline of formulations. Category Theory feels more like an organizing principle, like taxonomy—a collector's hobby more than a predictive theory. In this respect, Promise Theory has come much farther in a shorter space of time. However, as a cynic, I note that it has the forms and following to appeal more to the society of mathematical snobbery more than Promise Theory does, and will therefore rise to greater heights in the academic world.

[3]Impartiality may be the goal of science, but it is often defined tautologically, or not at all.

[4]This does not mean that they can necessarily work independently of external help, only they can always choose to ignore it if they so wish.

[5]For example, all processes can be represented as Feynman diagrams.

[6]This could be accomplished virtually by aggregating multiple serial events into a single process tick (using 'subtime' transactions).

[7]In continuous systems, the state is the position and generalized velocity. In a discrete system, the state is the position and the transition probabilities.

[8]Some like to use the term rules or laws of motion, but this suggests that they have some absolute nature, which is not the case. The behaviours are emergent, but might be sufficiently repeatable to warrant capturing as a formula.

[9]The issue of independent oversight was how Promise Theory came about, from CFEngine's model of system maintenance[Bur95, Bur04b].

[10]As we say in the networking technology world 'No route to destination'.

[11]It's interesting that this kind of 'public education' service was common after the second world war, but fell into disuse in the 1980s when many of the objectives became normalized. Today, do we assume normalized behaviour is established without such education? The risk of norms decaying into instability is always there. Maintenance of promises in necessary in all systems[Bur03].

[12]Inviting someone to invite them is almost like an imposition, but through the channels of diplomatic maneuvering.

[13]British readers may take offence at the spelling of offense, which is a constant source of consternation between regional norms.

[14]It often leads to what has come to be known as Complex Adaptive Behaviour to indicate the non-linearity and fragility of adaptation.

[15]The framing of law through Deontic reasoning, or obligations is a red herring. Promise Theory shows that it is really build upon a foundation made from layers of promises.

[16]This assumes that we are talking about promises of the first kind, which are the fundamental promises.

[17]This contradicts the mandatory language of 'inalienable human rights' in law, such as the American constitution's bill of rights—whose amendments are in fact nothing more than promises by the state to its subjects. There are no fundamental rights. There are only fundamental capabilities. It makes no sense to speak of the right for humans to be able to speak about X any more than anyone could grant the right to have wings, to fly, or to breathe under water.

[18]The etymology of authority comes from the Latin auctoritas, meaning 'originator, promoter', as in author.

[19]The latter is effectively a form of voting, but not based on a democratic majority. The mandate is required for every single agent individually for cooperation.

[20]This suggests that use of simple user interfaces may not increase human fidelity in difficult circumstances.

[21]Maxwell's cog wheel model of electromagnetism is a good example.

[22]It might sound bizarre and fanciful to think of the memory of a process, but it's no more radical than suggesting the existence of hidden degrees of freedom, dimensions, or internal quantum numbers. In physics, we have always assumed that propagation of influence or energy occurs deterministically or perhaps probabilistically, but without any hidden process, but we know this to be an oversimplification. In either case, there is an effective coupling constant but a 'driving term' like a forced oscillator. A finite process is needed to absorb or transfer any normal mode from a source to a receiver. This is the essence of Nyquist's theorem. A discrete process is a computation, see Turing, von Neumann, and Chomsky[Cho59]. The longer the chain of process required to decide an outcome of emission and absorption, the more interior memory a 'location' (agent, field, particle, etc) needs—e.g. simple energy levels that encode electron processes as scalar information in the form of energy levels. Promise Theory seems to show that there is no way out of the hierarchy of processes for transmission of information. It's sampling all the way down, which implies that the origin of time is always interior to agents.

[23]In physics, locality is usually a reference to light speed propagation, which makes a number of assumptions about the primacy of light and its universal role. No such assumption will be made here; rather, we define locality as a primitive notion, associated with the definition of a semantic boundary.

[24]The speeds of waves and signals, used to transmit influence, depend on material properties that are often mainly invariant, and can therefore be assumed constants.

[25]In practice, the application that reads the database is the observer. This might be quite localized. If the application has access points all over the globe, then the problem is of greater magnitude since the inconsistencies are magnified by transport latency.

[26]The exception is in data processing pipelines where processing time may dominate for some tasks.

[27]The increasingly pervasive language of Complex Adaptive Systems leads to assertions about strong and weak coupling, but we need to be able to define those things to use them.

[28]One can define the scope of a promise is a subset of agents that can observe a promise. For convenience we may write this:

$$A \xrightarrow[\sigma]{b} A' \tag{B.12}$$

where the set $\sigma \subset \{A\}$ is the scope, which includes A'. This can further be broken down into promises of the type $\mathbf{def}(\pi)$, as discussed earlier[BB14a]. Note that A cannot decide σ, as this would violate the principle of autonomy.

[29]This illusion of a flow is maintained by a gradient of intermediate agents that accumulate samples in buffers.

[30]We see the effect of 'populism' in society today, when intentions follow unstable polls instead of convictions.

[31]This is where the turtles hide.

[32]It is not a theorem in the mathematical sense,

[33]It's sometimes assumed that the coincidence limit is about getting closer to the finest grain of structure in nature, but in fact it's the opposite—what happens when you zoom out of a system and lose the resolution to see elementary discreteness.

[34]In physics, we have often neglected to account for possible anomalies in the scaling of spacetime properties. The assumption of continuity and Lorentz invariance all the way down is an obstacle one must overcome to model elementary spacetime processes.

[35]This topic is new to me, and seems worth following up.

[36]Note, nothing prohibits there being several channels for this communication, with different characteristics. We think immediately of light messages versus entanglement messaging, for instance.

[37]In physics, we associated Long Range Order with phase transitions and symmetry breaking[Gol92].

[38]There are obvious implications about discussions in QM about where is the electron? Can it be in more than one place at a time? Physics confusion arises from the inability to discuss clear semantics and make implicit assumptions based on tradition.

[39]One can speculate whether entanglement in QM is such an out of band channel that enables a short circuiting of local.

[40]The generating functional approach to percolation in [New03] may be compared to the effective action as a generating functional for Feynman diagrams[Abb92] and exposes the irrelevance of coordinate labels x, t to essential system properties.

[41]It's flawed reasoning to assume the conditions for reversibility by default and then claim that it's physical law.

[42]Anyone who has used a version control system, in IT, understands subtime as all those moments observers of the document repository cannot see, that lead to what was committed in each observable version.

[43]Defining an equal view is not a simple matter either, unless one assumes a god's eye privileged position.

[44]It does not rule out other forms of quantization at a smaller or larger scale.

[45]Of course, in practice the 'doctor' here represents a superagent of staff working with the doctor to support the doctor's practice

[46]The increasingly pervasive language of Complex Adaptive Systems leads to assertions about strong and weak coupling, but we need to be able to define those things to use them.

[47]This is the paradox of weather forecasting: when nothing is changing, we can predict the weather easily; but, when everything is in flux, prediction is hopeless.[Bur13a].

[48]One works hard to make this point in the physics of relativity where coordinate systems get in our way of understanding spacetime processes

[49]Reversibility is something of a misunderstood concept in dynamics, especially as applied to IT system behaviours. The apparent reversibility of the machinery is sometimes used as an argument against causality, but the argument is built on a misunderstanding that ignores the boundary conditions.

[50]These ionic properties may reach outside the superagent boundary too in some circumstances, allowing oxidation, etc.

[51]If one can imagine a disconnected agent that makes no promises at all, then from there, one can imagine postulating the existence of an infinite number of empty spacetimes. These matters become technical, and we have no way of deciding whether they are real or fictional, assuming that there is a distinction.

[52]Note, this information is related to the entropy of the system, but it is not related to disorder, merely a loss of information about order.

[53]In fact, in the case of a radio, one could argue that it is the outer casing which makes the promise of being a radio, and that the other components are tenants of the outer casing agent. I'll return to the issue of tenancy in the latter part of these notes.

[54]The description of a virtual hierarchy of perimeter boundaries around resources leads to a kind of 'Gauss law' for promises made by process agents. Any promise of state expressible externally must come from interior process memory.

[55]I've argued that one should instead be guided by *The Principle Of Separation Of Timescales* if predictability and stability are the primary goal[Bur19a, Bur4 a, Bur4 b].

[56]The intent of the Twelve-Factor App manifesto seems to principally address risk and local contention.

[57]A huge amount of discussion centres on data consensus sharing protocols for server (+) promises, but almost nothing is written about the responsibility of the receivers (-) who ultimately shoulder the burden of dealing with inconsistency. For an industrial example, see [Bre18].

[58]This is an interesting example because in Newtonian mechanics, a collision may be memoryless, but the trajectory isn't. The momentum of balls is conserved and remembers the sum effect of prior collisions. This is one of many ways to illustrate how memory and state are scale dependent.

[59]This may also be used as a definition of the autonomy of the agents.

[60]This is a more precise expression of what people mean by 'immutable containers'.

[61]This is what happens, for example, when runtime incidents lead to iterated bug fixes in software and new promises are made that incorporate past states into the initial conditions of current promises (feedback loops). The process of Continuous Delivery renders such changes on the same timescale as runtime transactions, which makes sensitivity to change higher.

[62]I borrow this phrase from Paul Borrill, and elsewhere use the term 'interior time'.

[63]This is Turing's halting problem.

[64]This explains the value of in band configuration maintenance for security and safety in a live setting. Instead of relying on isolation, one hopes for isolation but validates it with a competing immunity process ('trust but verify').

[65]This distinction and its scale dependence was the basis for the configuration management wars of the 2000s. It was argued that an initial state process was required along with complete congruence of steps (requiring total isolation at a high level)[Tra02]. The converse was argued: by creating closed operations in which the outcome was assured at a low level, the dependence on exterior ordering could be relaxed (which corresponds to a non-blocking execution policy)[Bur95, Bur04c]. The latter is just a micro-encapsulation of the former. The process is the same on different scales, but the latter is 'reactive' in the sense of the reactive manifesto[BFKT].

[66]See for example the way this is used as a limitation on observability to guarantee equilibration in entanglement systems [BBKK18].

[67]This is the case for many observation and monitoring systems.

[68]This feels like a good place to remind readers about the redundancy folk theorem, which states that low level redundancy is always at least as effective than high level redundancy. See section 10.5.

[69]The most fragile is perhaps the heart because of the high degree of serial coordinated calibration needed to maintain the rhythmical action. One could imagine a different kind of pump that did not have this flaw.

[70]The authors of these studies did not use Promise Theory in their work, but that does not prevent us from retrofitting the inferences from their work.

[71]Telling you twice that we owe you 100 dollars doesn't mean that we owe you 200 dollars.

[72]It is well known that the scaling of ad hoc communications networks, where agents are distributed randomly is like $\sqrt{N}$. This is easily understood from the spatial geometry: mobile phones occupy some approximately two dimensional area, so the diameter is of order $N^{\frac{1}{2}}$; alternatively, they have average separation $d \simeq V/\sqrt{N}$, so the distance across the group is of the order $Nd \simeq \sqrt{N}$. The linearity of the process gets mixed up with the geometry of the embedding space.

[73]Note that this is not a real connectivity, which has to do with the number of nodes, but a kind of close-packing of the sparse interactions that occur between the nodes into the infrastructure stream.

[74]A simple analogy is to think of a tube of toothpaste. The toothpaste comes out in a one dimensional stream of fixed width, but we are forcing the output of a three dimensional tube through this portal, and asking: how does the amount that comes out increase with the size of the tube if we squeeze it in the same way? By fixing the cross section, we can compare different tubes, or different cities.

[75]Electrons play this role in molecular chemistry, or telecommunications in the human realm.

[76]A dependency does not just have to be discovered, but also maintained in a persistent relationship, which accumulates cost over time.

[77]If the person's path is detailed, one could include the Hausdorff dimension of the path and use $v_i^{H/D}$ as the range, as Bettencourt suggests. I'll ignore this for now, as humans do not tend to move in fractal paths, as his data suggest.

[78]Because telecommunications networks are global, it does not make sense to relate their cost to the size of the city (though this depends on exactly how we model the costs), so the cost depends more on its usage than on its extent. We simply assume that it exists and has sufficient capacity for the N_I connected residents.

[79]Messages may be sent with or without words, with body postures, or coded by melody (frequency division multiplexing) rather than representations discrete in time (time division multiplexing). These are well known in information theory.

[80]One can speculate about the reason for the size of discrete patterns used to convey meaning. Dynamical scales will ultimately place limits of the comprehension of an agent. If agents could have infinite resolution, there would be no limit to what information could be conveyed in an arbitrary promise. But a recipient has to be able to parse this information in a finite time, shorter than that which is needed to keep its promise. This suggests that information density must be finite, whatever the nature of the agents. Even a concept like 'happiness' cannot have an infinite number of shades of grey! Protein size limits the size of a gene; variety of length and time scales limit the complexity of a key used by a human or a computer, and even the density of musical notes in the scale is limited to approximately quarter tones by the size of the human ear.

[81]The latter case is like the Millikan experiment for measuring electric charge. If you look for differences, then the smallest difference may be assumed to be an elementary.

[82]There is an analogy here with local gauge symmetries, as imagined by Weyl.

[83]Why, for example, would genes be preserved in number and type across species? If that were the case, all species would eventually equilibrate into one, and what was gained by one species would be lost by another.

[84]Causation in the sense of statistical inference has received a lot of popular press in recent times, and it a subtly different issue that is more controversial than causation by intent[Pea00].

[85]It is tempting to say that the probability should be equal to 1 to speak of causation, but that is too simplistic. We might choose to classify a failure as a fault instead, depending on the arrangement of intermediate agents in the system.

[86]A proper description of these states is beyond the scope of this work. This is not a completely trivial matter. Any dynamical system, in general, is characterized by two 'canonical' quantities: q and δq at each epoch.

[87]Laws of motion are not laws, of course, but observed patterns that are kept with overwhelming regularity.

[88]Note that amplification of effect does not depend on whether a system operates by push or by pull, though the susceptibility for promises not to be kept may do. Any system that has the intent to transfer intent with amplification can propagate and magnify both intended and unintended influence.

[89]It is a general prejudice arising from our manual experience to think of push as a driving force for change, but we know from countless examples that both possibilities exist. Gravity, magnetism, 'vacuum pressure', etc are all examples of pull. Even at the elementary particle level, interactions can be framed as retarded or advanced propagation, depending on how we choose to fix boundary conditions. It is precisely this desire to fix the end state, rather than the start, which favours pull.

[90]This approach is sometimes used in systems: continuous mandatory replacement of parts to ensure correctness. It can be effective, but it is disruptive and wasteful.

[91]This is a method of dissemination used in magazine stands, bookshops, content delivery networks (CDN), supply chains, etc.

[92]This was the motivation for cloud computing[SKAEMW13, Bur13a].

[93]VLSI or very large scale integration is the strategy of compressing multiple electronic components into a single chip for commodity packaged scaling efficiency.

[94]Causally, the role of the server can be eliminated effectively by scaling, like in renormalization group (see [Bur15a]).

[95]The recursion of agency is sometimes associated with the idea of 'namespaces' in information technology.

[96]In a bus architecture, like the original Ethernet, where a single channel is shared between multiple agents that need to coordinate their activity, data length and wavelength are connected. In Manchester encoding, for instance, a wavelength means a bit of information. Frequency and wavelength play a role in this tradeoff. Data therefore has a physical size on a wire, and contention resulting from the need to wait for confirmation on a shared wire sets minimum requirements for transaction size.

[97]The meaning of 'at the same time' is itself ambiguous. Simultaneity in a system is one of the difficult concepts in relativity, as it assumes a global definition of time, which might not be possible.

[98]Why doesn't modularity consider the natural dynamical separation of system parts (such as one might discover through Principal Component Analysis, or machine learning), or the network centrality regional structure[SPB$^+$03]? The main reason seems to be that we are more concerned with naming things, and scoping out territorial claims on functionality than in actual emergent behavioural patterns. This is one reason why performance analyses lead to refactorings, as an afterthought, than forming the basis for an initial design.

[99]Everyone who ever got a multiple choice exam wrong knows that the perspective or promiser and promisee are different: what is promised and what is received are two different things. You can try to make something very simple (binary choices); but, if the receiver doesn't interpret the semantics in the same way, that countable certainty of a discrete menu is useless.

[100]In building systems, modules are designed with the end user in mind, but in software systems modules are principally designed with the developers in mind.

[101] I choose not to cite the ephemeral sources for these 'quotes' as they are easily found, sometimes paraphrased, and pulled out of longer discussions. I hope readers agree that they are representative of the state of thinking.

[102] I sometimes call them ballistic processes, because we tend to treat computation as if it behaves something like a game of billiards: the mere sending of some data provokes an immediate involuntary reaction, fully deterministic. This is used to argue for 'push' over 'pull' methods, for instance, and is completely wrong.

[103] For a definition of proper time, see [Bur19a]

[104] The argument is that disk storage is cheap today, so why wouldn't you store everything forever? There are plenty of reasons. Our current experience with the crisis over cheap plastics should be a wake up call for anyone advocating an end to garbage collection. For example the escalating power cost of storage alone is a reason.

[105] See also the approach to network data consistency taken in [BBKK18].

[106] I suspect that the underlying and unspoken aim of advocating 'stateless' and 'throw away nothing' approaches is actually to linearize systems and make them as deterministic as possible by weak coupling. Alas, the rising cost of this, in some cases, is prohibitive and ultimately unsustainable, so alternative strategies should probably be considered.

[107] This idea was the built into the design of CFEngine, a realtime maintenance tool as a safety measure, and was rarely understood by users.

[108] The relationship with the strategy used by CFEngine to define 'convergent operators'[Bur95, Bur04c], is interesting. If you deal with pure functions, you cannot have maintenance of persistent agents. You redefine maintenance as the death and rebirth of an agent, with associated loss of runtime state. Runtime state is contained mutable state. But for a memory process it affects the behaviour of the agent in 'realtime', i.e. in band of the function's I/O channel.

[109] This applies to the nodes in any state machine too.

[110] The example of observing the inconsistent state of a clock was discussed in reference [Bur19a].

[111] In band 'self healing' configuration engines are essentially noise error correction processes, on a fairly long timescale of minutes to hours, which may be too slow to maintain invariance for busy processes.

[112] This is basically the reason why Continuous Delivery advocates recommend developing software in a single branch. Contention can then be resolved in band, since software development is a largely stateful process, in spite of modularity.

[113] Agents can be exchanged, by emission from one agent and absorption by another, as the actual information of the promise body. This is how one models the exchange of forces in quantum theory, via gauge bosons like the photon, or gluons, etc.

[114] In information technology, these maps are called variously directory services, name services, indirection tables, or data indices.

[115] This is somewhat like the way the immune system binds to cell sites.

[116] I've usually drawn agents schematically as atomic dots, but the shape of an agent is not defined. Nothing prevents it from appearing as a shell.

[117] This is essentially how routers and switches forward datagrams in information infrastructure.

[118] This corresponds roughly to the design of the Border Gateway routing tables, for IP addressing, as known from the Border Gateway Protocol BGP.

[119] One can see the historical reasons why the Internet was not designed as a Cartesian lattice, but modern datacentre fabrics still have the opportunity to repair this choice.

[120] This is essentially the method of top-down decomposition used in procedural programming, but with constraints that allow parallelized scaling of execution. One could imagine programming and representation

languages that support this kind of model in many different walks of life. This is probably how we should be teaching programming, and service management, instead of the linear imperative models of today. In the final chapter, based on [Bur16c], we'll see why the linear storyline approach has its own naturalness.

[121] In [Bur14] that motion of the second and third kinds distinguish between the idea that space and its occupants are either: ii) a visitation by a separate entity, or iii) a change in the state of the same entity. In other words, does space get filled by matter or does matter transform the nature of space?

[122] The description of services in [BB14a] treats clients and essentially faceless, generic entities.

[123] This happens frequently in merger and acquisition of companies of course.

[124] This is actually a hierarchy problem. At the lowest fundamental level of agency, there is no intermediate solution to this: either agents are adjacent or they aren't. We have to assume that they can sense and discover one another somehow (see section 7.8.2).

[125] It seems likely that survival instincts and intelligence would evolve to be strongest in those species that cannot find obvious safety in numbers.

[126] The vernacular 'dis-aggregation' is common.

[127] In data communications, so-called *frequency division multiplexing* is what corresponds to normal parallel resource sharing of R

[128] The similarity to promise terminology should not be a surprise: the two ideas are very closely related. BGP predates Promise Theory by many years, but through Promise Theory it gains a special clarity that cannot be seen when focusing on its irrelevant protocol.

[129] At the time of writing, the construction of a standard switch has valency of 48 possible tenants downwards, with fixed channel capacity, and a valency of two hosts upwards, each with greater capacity than the downward channels, for allow for aggregation.

[130] An analogy might be the following: do we consider the mind to be a tenant of the body, or the body to be a tenant of the mind? Dynamically (physically) the former makes sense, but semantically the latter is a highly convenient viewpoint.

[131] DNS round robin load balancing works in this way, as a directory service. The simple average round robin balancing algorithm works almost as well as more deterministic feedback algorithms at very low cost[BU06].

[132] It is often quoted that a single woman can have a baby in nine months, but nine women cannot make this happen in a month.

[133] Paxos, Raft, and other consistency algorithms involve processes of this type.

[134] The appearance negative contention can manifest when contentions between agents for common resources are mitigated. This effect has also been observed in the scaling of cities[Bet13].

[135] It is interesting how there is a limit to social networks due to limited brain size cost, as well known from the Dunbar hierarchy[Dun96, ZSHD04].

[136] Alternatively, if we think about the problem graph theoretically, we can also say that it behaves like a $D = N$ dimensional space, and a trajectory with Hausdorff dimension $H = \pi/sigma$. In a graph, the node degree $k = N$ is the effective dimension of spacetime at the point[Bur14].

[137] We usually reserve the term 'service' for bidirectional interactions, where the results of processing are returned to the initiator.

[138] Anyone who has tried to download a large file from the Internet, and have it interrupted, knows how important it is to not repeated work unnecessarily.

[139] Typical timescales recorded for Kafka 10^7 events per second[Wam16].

[140] Each agent in a chain of processing should be aware of its neighbours' semantics in order to maximize the intended meaning in its results, but it is sufficient to known the semantics of a single directional

flow to be able to make some kind of promise about the result: one can push responsibility for outcome downstream (see figure 2.4).

[141]The so-called Internet of Things plays an obvious role in equipping human spaces with smart capabilities. Information technology has a role to play in enhancing the design of infrastructure, both in terms of control and monitoring. It is already widely used in transport systems, but many existing systems are archaic and poorly standardized. It is worth remembering that processing may be carried out by hardware or software. Programmable low level hardware can work orders or magnitude faster than high level software (e.g. programmable ASICs, IBM True North chips, etc). Any kind of specialization of space is a form of hard processing, where agents separate and gravitate to special areas by functional affinity.

Learning capabilities: computers do three things:

- Aggregate/disseminate data: the basis for learning or sharing.
- Transform data: the basis for automation / algorithms.
- Update and maintain software.

When we ask what computing can do for cities and other spaces, we are asking: what can we do with learning and automated production? Automated learning sounds nice, but it is the computer that is learning, not necessarily people! To make the computation useful to society, what is learned also has to be made available, in the right context and at the right time.

Computers are only proxies for decision-making, they do not make decisions. Decisions are made by human inputs to software or policies that become constraints on the outcomes of processes. Computation encodes policy in programs that acts as proxies and evaluate the pre-programmed decisions. No matter how much machine-learning or AI we apply, it is (human) policy decision-making that shapes the results. This can be applied to people, things, information, or software. There are clearly ethical issues with the use of automated reasoning in complex systems: what we intend is not always what we get.

[142]Regardless of where or how decisions are made, the agents that end up with responsibility for keeping promises are:

People (behaviour, habits, desires, intentions, preferences, wellbeing, health, entertainment), Land (semantics of land use, urban, green, woodland, agriculture, crops), bacteria, viruses, information (propagation, misuse, illness), companies, institutions, groups (goods, services, products), transportation and logistics (delivery systems, messengers, relocation), etc.

[143]There are many themes that one could raise under the aegis of around smart spaces. A few of them are described below, under the following major themes:

- Access to infrastructure (utilities)
- Housing and workspaces
- Configurable and adaptive structures
- Waste processing
- Balancing privacy and sympathy
- Logistics, and tracking of resources
- Sharing economy
- Competitive economy
- Maintaining diversity to sustain economy and innovation
- Public safety, health, and welfare
- Emergency and disaster response

- Skill and service availability

How do we make functional selections or decisions: What is the reason for a space?

- Pleasure,
- General use,
- Breathing room,
- Separation (focus)
- Mixing (catalysis)
- etc.

[144]Sharing allows us to minimize expensive waste, by increasing utilization of resources, but only when certain conditions are met: For an economy of scale to make sense, there are two prerequisites:

- Moderately-saturated (sparse) demand.
- Moderately-saturated supply (delivery).

Transportation (communication) is the prerequisite key to unlocking this. When supply/demand are too low, the cost of centralization (transportation) is too high for the central service. When the supply/demand are too high, the cost of queueing/waiting at the central location is too high for the clients[Bur16b, Bet13, BLH+07].

[145]Discrimination of agents for assigning identity is a key function of knowledge and reasoning. For smart spaces, it means: what is the purpose and signal represented by each spatial location, at each time? Discrimination of agents (locations), based on functional characteristics, is the way we create cooperative systems, and encode information patterns (learning). In an information rich society, agents are competing for attention, through their unique identities (brands) - this is how they establish agency in the assessment of others. It is not the brands they promise, but the perception (assessment by others) of the brands that matters.

[146]Ad hoc services and offerings can be made sense of using ordinary service registration, but spontaneous events could also be recognized by sensors, without need for manual registration (like in white/yellow pages). Distributed localization (proximity to resources) is the key to cheap scaling, and discovery helps to identify both possibilities and efficiencies. Services are not binary, they scale according to costs and returns.

> "...on no account allow the engineering to dictate a building's form. Place load bearing elements...according to the social spaces...never modify the social spaces to conform to the engineering structure..."
> –Alexander et al, A Pattern Language[ea77]

Think about the scaling of Content Delivery Networks (CDN). This is classical distribution logistics. Content-centric networking (CCN) or Name Data Networking (NDN) are also expressions of distributed supply caching, with information based search capabilities, rather than address-based location.

The idea is to combine a directory service (white/yellow pages) with an address lookup system, like a table of contents, so that we can look up by function instead of present location. This offers better support for mobility.

[147]Hand-waving heuristics or 'bad feelings' are often used to cry foul. Bad feelings are not very useful for engineering, but they cannot be dismissed either. Human intuition is mysterious, and can often sense 'danger' without being able to form a narrative to explain it.

[148]While there is low level causation at work in the system, it is not necessarily traceable as a convenient human narrative of events.

[149]The fighter jet is an example of this. In order to fly with great agility it is basically designed to be unstable. It remains in control by correcting for these instabilities with very fast computer automation. This is a case of very clever design around the edge of instability.

[150]The expression a 'black swan' event is sometimes used for rare unimagined occurrences. In this analogy, you wouldn't bother to promise that a swan is white, because it is so obvious.

[151]Science is about measurement, and it is usually unprejudiced about the behaviours of its subjects. In the case of technology, the subjects are designed deliberately to have semantics; hence that becomes part of their behaviour. A promise creates an abstract measuring scale for semantically charged, purposeful outcomes.

[152]This model can be applied whether an agent is a human or some kind of proxy, machine, device, etc. Ignoring the distinction makes sense because we humans experience any behaviour as if it were anthropomorphic or 'caused by agencies': think of 'the ghost in the machine', or 'Maxwell's daemon'. This is harmless, even practical, as long as we don't read too much into what we mean by an agent. Ultimately all agency stems from some human source, either by programming or direct control.

[153]Like the quote from the chaotician in Jurassic Park said, 'nature will find a way'.

[154]Scaling is important because it is an enabler for opportunity: sexual mixing (male), discerning selection (female), and then post mixing isolation (gestation) in which the arduous calculations work through their constrained narrative processes. Random mutation might actually be promoted by separation - isolated environments are less stable and test out mutations more quickly because in a small network, each single agent is proportionally more important (too big to fail).

[155]It does not matter here whether we consider the force to be a Newtonian deterministic force, or a probabilistic susceptibility for drifting closer, as in stochastic systems.

[156]One of the weaknesses of this simple model is that this is unclear, but we return to discuss this issue in the next chapter

[157]In recent times, the idea of 'chaos engineering' has been used, often attributed to Netflix engineers. The idea is to randomly search the space of possible catastrophes by wreaking chaos, switching things on and off, and so on, in the hope of provoking the unexpected. This is a kind of Monte Carlo search method for anticipating failure modes.

[158]Some would say 'low probability' but this is an abuse of terminology, as probability is only meaningful under expectations and assumptions of essential system stability, i.e. in which future behaviours are basically the same as past behaviours or are only slowly varying.

[159]It's possible that other agents, merely in scope of a promise, might rely on it 'unofficially', but that is an act of irresponsibility by the downstream principle.

[160]This problem was studied in [Dis07, BD07, BDS07] for autonomous self-healing repairs with dependencies added.

[161]The example of using maggots to clean a wound comes to mind. We associate maggots with the disgusting dirty flies they become, yet maggots eat only dead flesh and discriminate far more efficiently than any artificial mechanism we have yet created.

[162]The CALM principle attempts to do this for more general processes, see section 5.12.3.

[163]The use of vector clocks is common to bring a notion of consistent serialization of changes in systems, preserving order, but this is independent of the temporal consistency of global state, which is basically impossible without locking in a highly disruptive manner. Thus a promise of consistency might be given at the expense of a promise about availability. This is known as the CAP problem[Bre00].

[164]Text editors can do this as long as there is only a single journal of changes and a single user interacting with the data. Local databases can likewise maintain process integrity.

[165]This explains the value of in band configuration maintenance for security and safety in a live setting. Instead of relying on isolation, one hopes for isolation but validates it with a competing immunity process ('trust but verify').

[166]This distinction and its scale dependence was the basis for the configuration management wars of the 2000s. It was argued that an initial state process was required along with complete congruence of steps (requiring total isolation at a high level)[Tra02]. The converse was argued: by creating closed operations in which the outcome was assured at a low level, the dependence on exterior ordering could be relaxed (which corresponds to a non-blocking execution policy)[Bur95, Bur04c]. The latter is just a micro-encapsulation of the former. The process is the same on different scales, but the latter is 'reactive' in the sense of the Reactive Manifesto[BFKT].

[167]There is a tradition of using logic to try to eliminate uncertainty in reasoning, which in turn has brought an almost irrational focus to bear on logics that are quite ill-equipped to describe the real-world situations.

[168]This is what Artificial Intelligence people refer to as unsupervised learning

[169]The term immutable process is sometimes used in computing to refer to the idea that what happens inside a box, which only observes the outside world, does not affect the world outside the box, even though it affects the world inside the box.

[170]Interestingly, the collection of observational measurements from a distributed system is related to the far-more widely studied problem of data *consensus*, in Computer Science[Lam01, OO14]. The latter considers how we may distribute multiple copies of information over a wide area, with integrity of order—surely one of the most frequently revisited problems tackled in distributed systems. The fascination with consensus stems from the attempt to cling onto approximate determinism.

[171]The contents of this chapter are basically derived from [Bur17c, Bur19a].

[172]In stable predictable systems, interaction scales are not strongly coupled. Strongly coupled systems may be unpredictable and unstable.

[173]This section is based on the discussion in [Bur16c].

[174]In Einsteinian relativity, the term proper time is reserved for the time experienced by an observer about its own states, so I keep to this convention here.

[175]The main approach to determining spacetime invariance in science is by the use of statistics (aggregation or 'learning'): by accumulating multiple samples, we hope to separate what is quickly varying (fluctuation) from what is slowly varying (trend). Persistent concepts are what remains during or after a process of learning has separated these processes.

[176]See reference [BB14a]. The law of intermediaries basically says that intermediate agents cannot be relied upon to faithfully transmit promises or intent, because all agents are fundamentally autonomous.

[177]I won't consider this here, but see the signals lemma in [BL19].

[178]The extent to which we have the ability to localize a causal influence is the essence of 'root cause' analysis: the ability to contain the process within a virtual boundary which itself can make promises on a new level. This is part of the motivation for virtualization and containerization.

[179]We must be cautious and pay attention to the Promise Theory principle that just because one agent promises to contain another or be followed by another, it does not imply that the agent concerned agrees with this, and may not promise it. For example, firewalls may create a one-way glass effect that prevents the inverse from being implemented)

[180]This is a principle that I have reiterated many times since CFEngine[Bur95] to stabilize and guide us towards invariant meaning.

[181]The question of whether to invest in promising an expensive and late consensus over a coarse grain of space, or whether to expose its divergences as a feature remains a policy choice—one that currently aligns

with opposite poles of Dev and Ops.

[182] I keep extensive notebooks of observations and thinking instead of a diary.

[183] 3GPP Technical Specification TS 23.003 Numbering, addressing and identification contains a section defining the Identification of location areas and base stations, using LAI, LAC, RAI, RAC, CI, CGI, BSIC, RSZI, LN, SAI. Also a section on Identification of mobile subscribers, using IMSI, TMSI, P-TMSI, LMSI, TLLI.

BIBLIOGRAPHY

[Abb92] L.F. Abbott. *Acta Physica Polonica*, B13:33, 1992.

[ABC+15] T. Akidau, R. Bradshaw, C. Chambers, S. Chernyak, R.J. Fernndez-Moctezuma, R. Lax, S. McVeety, D. Mills, F. Perry, E. Schmidt, and S. Whittle. The dataflow model: A practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. *Proceedings of the VLDB Endowment*, 8:1792–1803, 2015.

[ABH06] D. Aredo, M. Burgess, and S. Hagen. A promise theory view on the policies of object orientation and the service oriented architecture. In *submitted to Science of Computer Programming*, 2006.

[AHF+14] E. Arcaute, E. Hatna, P. Ferguson, H. Youn, A. Johansson, and M. Batty. Constructing cities, deconstructing scaling laws. *Journal of The Royal Society Interface*, 12(102), 2014.

[Amd67] G.M. Amdahl. Validity of a the single processor approach to achieving large scale computer capabilities. In *Proceedings of the AFTPS Spring Joint Computer Conference*, 1967.

[Ano14] Anonymous. Clos ip fabrics with qfx5100 switches: Building a flexible, programmable datacenter network using layer 3 protocols and overlay networking. Technical report, Juniper Networks, 2014.

[AR97] A. Abdul-Rahman. The pgp trust model. *EDI-Forum: the Journal of Electronic Commerce*, 1997.

[Ash52] W.R. Ashby. *Design for a brain*. J. Wiley & Sons, 1952.

[Ash56] W.R. Ashby. *An introduction to cybernetics*. J. Wiley & Sons, 1956.

[Asi50] Isaac Asimov. *I, Robot*. Doubleday?, ?, 1950.

[Axe84] R. Axelrod. *The Evolution of Co-operation*. Penguin Books, 1990 (1984).

[Axe97] R. Axelrod. *The Complexity of Cooperation: Agent-based Models of Competition and Collaboration*. Princeton Studies in Complexity, Princeton, 1997.

[Bar96] G.I. Barenblatt. *Scaling, self-similarity, and intermediate asymptotics*. Cambridge, 1996.

[Bar03] G.I. Barenblatt. *Scaling*. Cambridge, 2003.

[BB06] J.A. Bergstra and M. Burgess. Local and global trust based on the concept of promises. Technical report, arXiv.org/abs/0912.4637 [cs.MA], 2006.

[BB08] R. Badonnel and M. Burgess. Service load balancing with autonomic servers: Reversing the decision making process. In *Resilient Networks and Services, Second International Conference on Autonomous Infrastructure, Management and Security, AIMS 2008, Bremen, Germany, July 1-3, 2008, Proceedings*, pages 92–104, 2008.

[BB14a] J.A. Bergstra and M. Burgess. *Promise Theory: Principles and Applications*. $\chi tAxis$ Press, 2014.

[BB14b] Jan Bergstra and Mark Burgess. *Promise Theory*. χtAxis Press, February 2014.

[BB19a] J. A. Bergstra and M. Burgess. A Promise Theoretic Account of the Boeing 737 Max MCAS Algorithm Affair. *arXiv.org/abs/2001.01543*, page 20 pages, 2019.

[BB19b] J.A. Bergstra and M. Burgess. A Promise Theoretic account of the Boeing 737 Max MCAS algorithm affair. *arXiv (submitted)*, 2019.

[BB20] J.A. Bergstra and M. Burgess. Candidate software process flaws for the Boeing 737 max MCAS algorithm, and risks for a proposed upgrade. *arXiv (submitted)*, 2020.

[BBCD14] P. Borrill, M. Burgess, T. Craw, and M. Dvorkin. A promise theory perspective on data networks. *CoRR*, abs/1405.2627, 2014.

[BBCEM10] J. Bjelland, M. Burgess, G. Canright, and K. Eng-Monsen. Eigenvectors of directed graphs and importance scores: dominance, t-rank, and sink remedies. *Data Mining and Knowledge Discovery*, 20(1):98–151, 2010.

[BBKK18] P. Borrill, M. Burgess, A. Karp, and A. Kasuya. Spacetime-entangled networks (i) relativity and observability of stepwise consensus. *arXiv:1807.08549 [cs.DC]*, 2018.

[BC03] M. Burgess and G. Canright. Scalability of peer configuration management in partially reliable and ad hoc networks. *Proceedings of the VIII IFIP/IEEE IM conference on network management*, page 293, 2003.

[BC04] M. Burgess and G. Canright. Scaling behaviour of peer configuration in logically ad hoc networks. *IEEE eTransactions on Network and Service Management*, 1:1, 2004.

[BC11] M. Burgess and A. Couch. On system rollback and totalized fields: An algebraic approach to system change. *J. Log. Algebr. Program.*, 80(8):427–443, 2011.

[BCA+12] Md. Faizul Bari, Shihabur Rahman Chowdhury, Reaz Ahmed, Raouf Boutaba, and Bertrand Mathieu. A survey of naming and routing in information-centric networks. *IEEE Communications Magazine*, 50(12):44–53, 2012.

[BD07] M. Burgess and M. Disney. Understanding scalability in network aggregation with continuous monitoring. In *Lecture Notes on Computer Science, Proc. 18th IFIP/IEEE Distributed Systems: Operations and Management (DSOM 2007)*, volume (submitted). Springer, 2007.

[BDS07] M. Burgess, M. Disney, and R. Stadler. Network patterns in cfengine and scalable data aggregation. In *Proceedings of the 21st Conference on Large Installation System Administration Conference*, LISA'07, pages 22:1–22:15, Berkeley, CA, USA, 2007. USENIX Association.

[Bel09] M. Belbin. Team roles. `http://www.belbin.com/about/belbin`, 2009.

[Bet13] L.M.A. Bettencourt. The origins of scaling in cities (with supplements). *Science*, 340:1438–1441, 2013.

[BF07a] M. Burgess and S. Fagernes. Laws of systemic organization and collective behaviour in ensembles. In *Proceedings of MACE 2007*, volume 6 of *Multicon Lecture Notes*. Multicon Verlag, 2007.

[BF07b] M. Burgess and S. Fagernes. Norms and swarms. *Lecture Notes on Computer Science*, 4543 (Proceedings of the first International Conference on Autonomous Infrastructure and Security (AIMS)):107–118, 2007.

[BF08] M. Burgess and S. Fagernes. Laws of human-computer behaviour and collective organization. *submitted to the IEEE Journal of Network and Service Management*, 2008.

[BFKT] J. Bonér, D. Farley, R. Kuhn, and M. Thompson. The reactive manifesto. https://www.reactivemanifesto.org/.

[BHRS01] M. Burgess, H. Haugerud, T. Reitan, and S. Straumsnes. Measuring host normality. *ACM Transactions on Computing Systems*, 20:125–160, 2001.

[BK07] M .Burgess and L. Kristiansen. *Handbook of Network and System Administration*, chapter On the Complexity of Change and Configuration Management. Elsevier, 2007.

[BL19] M. Burgess and W. Louth. Preserving the significance of distributed observations. *unpublished*, 2019.

[BLH+07] L.M.A. Bettencourt, J. Lobo, D. Helbing, C. Hühnert, and G.B. West. Growth, innovation, scaling and the pace of life in cities. *Proceedings of the National Academy of Sciences*, 104(107):7301–7306, 2007.

[Bon16] J. Bonér. Life beyond the illusion of the present (talk at voxxeddays, zurich). 2016.

[Bon19] J. Bonér. Serverless needs a bolder, stateful vision. *The Newstack*, Serverless, 2019.

[BP19] M. Burgess and E. Prangsma. Koalja: from data plumbing to smart workspaces in the extended cloud. *arXiv:1907.01796 [cs.DC]*, 2019.

[Bre00] E. Brewer. Towards robust distributed systems. In *Keynote, Symposium on Principles of Distributed Computing (PODC).*, 2000.

[Bre18] C. Breck. From a time-series database to a key operational technology for the enterprise: Part ii, March 2018.

[BS97] M. Burgess and D. Skipitaris. Adaptive locks for frequently scheduled tasks with unpredictable runtimes. *Proceedings of the Eleventh Systems*

Administration Conference (LISA XI) (USENIX Association: Berkeley, CA), page 113, 1997.

[BU06] M. Burgess and G. Undheim. Predictable scaling behaviour in the data centre with multiple application servers. In *Lecture Notes on Computer Science, Proc. 17th IFIP/IEEE Distributed Systems: Operations and Management (DSOM 2006)*, volume 4269, pages 9–60. Springer, 2006.

[Bur4 a] M. Burgess. *A Treatise on Systems: Volume 1: Analytical description of human-information networks.* in progress, 2004-.

[Bur4 b] M. Burgess. *A Treatise on Systems: Volume 2: Intentional systems with faults, errors, and flaws*. in progress, 2004-.

[Bur95] M. Burgess. A site configuration engine. *Computing systems (MIT Press: Cambridge MA)*, 8:309, 1995.

[Bur98] M. Burgess. Computer immunology. *Proceedings of the Twelth Systems Administration Conference (LISA XII) (USENIX Association: Berkeley, CA)*, page 283, 1998.

[Bur00a] M. Burgess. The kinematics of distributed computer transactions. *International Journal of Modern Physics*, **C**12:759–789, 2000.

[Bur00b] M. Burgess. Thermal, non-equilibrium phase space for networked computers. *Physical Review E*, 62:1738, 2000.

[Bur02] M. Burgess. Two dimensional time-series for anomaly detection and regulation in adaptive systems. *Lecture Notes in Computer Science, IFIP/IEEE 13th International Workshop on Distributed Systems: Operations and Management (DSOM 2002)*, 2506:169, 2002.

[Bur03] M. Burgess. On the theory of system administration. *Science of Computer Programming*, 49:1, 2003.

[Bur04a] M. Burgess. *Analytical Network and System Administration — Managing Human-Computer Systems*. J. Wiley & Sons, Chichester, 2004.

[Bur04b] M. Burgess. Configurable immunity for evolving human-computer systems. *Science of Computer Programming*, 51:197, 2004.

[Bur04c] M. Burgess. Configurable immunity model of evolving configuration management. *Science of Computer Programming*, 51:197, 2004.

[Bur04d] Mark Burgess. *Analytical Network and System Administration — Managing Human-Computer Systems*. J. Wiley & Sons, Chichester, 2004.

[Bur05a] M. Burgess. A tiny overview of cfengine: convergent maintenance agent. In *Proceedings of the 1st International Workshop on Multi-Agent and Robotic Systems, MARS/ICINCO*, 2005.

[Bur05b] Mark Burgess. An approach to understanding policy based on autonomy and voluntary cooperation. In *IFIP/IEEE 16th international workshop on distributed systems operations and management (DSOM), in LNCS 3775*, pages 97–108, 2005.

[Bur09] Mark Burgess. Knowledge management and promises. *Lecture Notes on Computer Science*, 5637:95–107, 2009.

[Bur13a] M. Burgess. *In Search of Certainty: the science of our information infrastructure*. Xtaxis Press, 2013.

[Bur13b] Mark Burgess. *In Search of Certainty - The Science of Our Information Infrastructure*. χtAxis Press, November 2013.

[Bur14] M. Burgess. Spacetimes with semantics (i). `http://arxiv.org/abs/1411.5563`, 2014.

[Bur15a] M. Burgess. Spacetimes with semantics (ii). `http://arxiv.org/abs/1505.01716`, 2015.

[Bur15b] Brendan Burns. How kubernetes changes operations. *;login:*, 40(5), 2015.

[Bur16a] M. Burgess. A promise theory approach to understanding resilience: faults, errors, and tolerance within systems. Technical report, Available at markburgess.org, 2015-2016.

[Bur16b] M. Burgess. On the scaling of functional spaces, from smart cities to cloud computing. *arXiv:1602.06091 [cs.CY]*, 2016.

[Bur16c] M. Burgess. Spacetimes with semantics (iii). `http://arxiv.org/abs/1608.02193`, 2016.

[Bur17a] M. Burgess. From observability to significance in distributed information systems. `https://arxiv.org/abs/1907.05636`, 2017.

[Bur17b] M. Burgess. Locality, statefulness, and causality in distributed information systems (concerning the scale dependence of system promises). `https://arxiv.org/abs/1909.09357`, 2017.

[Bur17c] M. Burgess. A spacetime approach to generalized cognitive reasoning in multi-scale learning. `https://arxiv.org/abs/1702.04638`, 2017.

[Bur18] M. Burgess. Notes of data pipelines. Technical report, Aljabr Inc., June 2018.

[Bur19a] M. Burgess. From observability to significance in distributed systems. *arXiv:1907.05636 [cs.MA]*, 2019.

[Bur19b] M. Burgess. Locality, statefulness, and causality in distributed information systems (concerning the scale dependence of system promises). *arXiv:1909.09357 [cs.DC]*, 2019.

[Bur19c] M. Burgess. *Smart Spacetime*. χtAxis Press, 2019.

[Buz73] J.P. Buzen. Computational algorithms for closed queueing networks with exponential servers. *Communications of the ACM*, 16:527, 1973.

[BW15] M. Burgess and H. Wildfeuer. Technical report, IRTF, https://tools.ietf.org/html/draft-burgess-promise-iot-arch-00.txt, Oct 19 2015.

[Car18] K. Carter. Broken promises (talk at jfokus). 2018.

[CB09] A. Couch and M. Burgess. Compass and direction in topic maps. *(Oslo University College preprint)*, 2009.

[CG00] David R. Cheriton and Mark Gritter. Triad: A scalable deployable nat-based internet architecture. Technical report, 2000.

[CHIK03] A. Couch, J. Hart, E.G. Idhaw, and D. Kallas. Seeking closure in an open world: A behavioural agent approach to configuration management. *Proceedings of the Seventeenth Systems Administration Conference (LISA XVII) (USENIX Association: Berkeley, CA)*, page 129, 2003.

[Cho59] N. Chomsky. On certain formal properties of grammars. *Information and Control*, 2(2):137–167, 1959.

[CL99] Hyoung-Kee Choi and John O. Limb. A behavioral model of web traffic. In *ICNP '99: Proceedings of the Seventh Annual International Conference on Network Protocols*, page 327, Washington, DC, USA, 1999. IEEE Computer Society.

[Clo53] C. Clos. A study of non-blocking switching networks. *Bell System Technical Journal*, 32(2):406–424, 1953.

[Coc06] A. Cockcroft. Utilization is virtually useless as a metric! In *Proceedings of Int. CMG Conference*, pages 557–562, 2006.

[Coc19] A. Cockcroft. Failure modes and continuous resilience. *Medium*, 2019.

[CT91] T.M. Cover and J.A. Thomas. *Elements of Information Theory*. (J.Wiley & Sons., New York), 1991.

[D$\ddot{9}$6] D. Dörner. *The Logic of Failure*. Basic Books, New York, 1996.

[DA94] R. David and H. Alla. Petri nets for modelling of dynamic systems — a survey. *Automatica*, 30:175–202, 1994.

[Dat99] C.J. Date. *Introduction to Database Systems (7th edition)*. Addison Wesley, Reading, MA, 1999.

[DCP17] G.M. D'Ariano, G. Chiribella, and P. Perinotti. *Quantum Theory From First Principles*. Cambridge, 2017.

[Dek06] S. Dekker. *The field guide to understanding human error*. Ashgate Publishing, Surrey, 2006.

[Dek11] S. Dekker. *Drift Into Failure*. Ashgate Publishing, Surrey, 2011.

[DEKM98] R. Durbin, S. Eddy, A. Krigh, and G. Mitcheson. *Biological Sequence Analysis*. Cambridge, Cambridge, 1998.

[DF98] D. Dasgupta and S. Forest. An anomaly detection algorithm inspired by the immune system. *Artifical immune systems and their applications*, page 262, 1998.

[DH06] D.D.Woods and E. Hollnagel. *Joint Cognitive Systems: Patterns in Cognitive Systems Engineering*. Taylor & Francis, New York, 2006.

[DHS10] F. Dowker, J. Henson, and R. Sorkin. Discreteness and the transmission of light from distant sources. *arXiv:1009.3058 [gr-qc]*, 2010.

[Dia97] J. Diamond. *Guns, Germs, and Steel*. Vintage, 1997.

[Die02] R.H. Dieck. *Measurement and Uncertainty (Methods and Applications)*. Instrument, Systems and Automation Society, Triangle Park, North Carolina, third edition edition, 2002.

[Dis07] M. Disney. Exploring patterns for scalability of network administration with topology constrants. Master's thesis, Oslo University College, 2007.

[Dow08] F. Dowker. Causal sets and the deep structure of spacetime. *arXiv:gr-qc/0508109*, 2008.

[DS05] M. Dam and R. Stadler. A generic protocol for network state aggregation. *RVK 05, Linkping, Sweden, June 14-16*, 2005.

[Dun96] R. Dunbar. *Grooming, Gossip and the Evolution of Language*. Faber and Faber, London, 1996.

[ea77] C. Alexander et al. *A Pattern Language: Towns, Building, Construction*. Oxford University Press, 1977.

[Eve56] H. Everett. *Theory of the Universal Wavefunction*. PhD thesis, Princeton, 1956.

[Fey49] R.P. Feynamn. Space-time approach to quantum electrodynamics. *Physical Review*, 76:769, 1949.

[FLP85] M.J. Fischer, N.A. Lynch, and M.S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, April 1985.

[For75] D. Forster. *Hydrodynamic Fluctuations, Broken Symmetry and Correlation Functions*. (Addison Wesley, California), 1975.

[GH98] Donald Gross and Carl M. Harris. *Fundamentals of queueing theory*. John Wiley & Sons, Inc., New York, NY, USA, 3 edition, 1998.

[GL06] Jim Gray and Leslie Lamport. Consensus on transaction commit. *ACM Trans. Database Syst.*, 31(1):133–160, March 2006.

[Gol92] N. Goldenfeld. *Lectures On Phase Transitions And The Renormalization Group*. Addison Wesley, 1992.

[Gor68] J.E. Gordon. *The New Science of Strong Materials, or Why You Don't Fall Through the Floor*. Penguin Books, London, 1968.

[GPT15] N.J. Gunther, P. Puglia, and K. Tomasette. Hadoop superlinear scalability: The perpetual motion of parallel performance. *ACM Queue*, 13(5), 2015.

[GS01] G.R. Grimmett and D.R. Stirzaker. *Probability and random processes (3rd edition)*. Oxford scientific publications, Oxford, 2001.

[Gun93] N. J. Gunther. A simple capacity model of massively parallel transaction systems. In *CMG National Conference*, 1993.

[Gun08] N.J. Gunther. A general theory of computational scalability based on rational functions. Technical report, arXiv:0808.1431, 2008.

[HA19] J.M. Hellerstein and P. Alvaro. Keeping calm: When distributed consistency is easy. *arXiv:1901.01930 [cs.DC]*, 2019.

[Har11] Y.N. Harari. *Sapiens*. Vintage, 2011.

[HBS73] C. Hewitt, P. Bishop, and R. Steiger. A universal modular actor formalism for artificial intelligence. In *Proc. of the 3rd IJCAI*, pages 235–245, Stanford, MA, 1973.

[Hel96] J.L. Hellerstein. An approach to selecting metrics for detecting performance problems in information systems. *Performance Evaluation Review*, 24:266, 1996.

[Hel07] P. Helland. Life beyond distributed transactions: an apostate's opinion. *ACM Queue (Distributed Computing)*, 14(5), 2016 (2007).

[Hel16] P. Helland. Immutability changes everything. *ACM Queue (Databases)*, 13(9), 2016.

[HF10] J. Humble and D. Farley. *Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation*. Addison Wesley, New Jersey, 2010.

[Hic09] R. Hickey. Are we there yet? Talk at QConn, 2009. http://www.infoq.com/presentations/Are-We-There-Yet-Rich-Hickey.

[HL93] P. Hoogenboom and J. Lepreau. Computer system performance problem detection using time series models. *Proceedings of the USENIX Technical Conference, (USENIX Association: Berkeley, CA)*, page 15, 1993.

[Hog95] C Hogan. Metrics for management. *Proceedings of the Ninth Systems Administration Conference (LISA IX) (USENIX Association: Berkeley, CA*, page 125, 1995.

[HPFS02] R. Housley, W. Polk, W. Ford, and D. Solo. Internet x.509 public key infrastructure: Certificate and certificate revocation list (crl) profile. http://tools.ietf.org/html/rfc3280, 2002.

[HR94] A. Høyland and M. Rausand. *System Reliability Theory: Models and Statistical Methods*. J. Wiley & Sons, New York, 1994.

[HWL06] E. Hollnagel, D.D. Woods, and N. Leveson, editors. *Resilience Engineering*. Ashgate Publishing, Surrey, 2006.

[IEE] IEEE. A standard classification for software anomalies. *IEEE Computer Society Press, 1992*.

[Inc] Aljabr Inc. Koalja history package. https://github.com/AljabrIO/ koalja-operator/tree/master/pkg/history.

[IT93] ITU-T. *Open Systems Interconnection - The Directory: Overview of Concepts, models and service. Recommendation X.500.* International Telecommunications Union, Geneva, 1993.

[Kan03] L. Kanies. Isconf: Theory, practice, and beyond. *Proceedings of the Seventeenth Systems Administration Conference (LISA XVII) (USENIX Association: Berkeley, CA)*, page 115, 2003.

[Kle76] Leonard Kleinrock. *Queueing Systems: Computer Applications*, volume 2. John Wiley & Sons, Inc., 1976.

[KMS06] T. Konopka, F. Markopoulou, and L. Smolin. Quantum graphity. *arXiv:hep-th/0611197v1*, 2006.

[Kri63] S.A. Kripke. Semantical considerations in modal logic. *Acta Philosophica Fenica*, 16:83–94, 1963.

[Lam78] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, July 1978.

[Lam01] Leslie Lamport. Paxos Made Simple. *SIGACT News*, 32(4):51–58, December 2001.

[Lay06] Richard Layard. *Happiness: Lessons from a New Science*. Penguin, 2006.

[LGZ+14] H. Li, A. Ghodsi, M. Zaharia, S. Shenker, and I. Stoica. Tachyon: Reliable, memory speed storage for cluster computing frameworks. In *Proceedings of the ACM Symposium on Cloud Computing*, SOCC '14, pages 6:1–6:15, New York, NY, USA, 2014. ACM.

[LLG+09] M. Liu, M. Li, D. Golovnya, E. A. Rundensteiner, and K. Claypool. Sequence pattern query processing over out-of-order event streams. In *2009 IEEE 25th International Conference on Data Engineering*, pages 784–795, March 2009.

[LP97] H. Lewis and C. Papadimitriou. *Elements of the Theory of Computation, Second edition*. Prentice Hall, New York, 1997.

[Mar98] F. Markopoulou. The internal description of a cause set: What the universe looks like from the inside. *arXiv:gr-qc/9811053v2*, 1998.

[MB04] G. Canright M. Burgess. Scalability of peer configuration management in logically ad hoc networks. *Network and Service Management, IEEE Transactions on*, (1):21 – 29, 2004.

[Mil09] R. Milner. *The space and motion of communicating agents*. Cambridge, 2009.

[MK05] Mark Burgess and Kyrre Begnum. Voluntary cooperation in pervasive computing services. In *LISA'05*, 2005.

[MMS85] J.F. Meyer, A. Movaghar, and W.H. Sanders. Stochastic activity networks: structure, behavior and application. *Proceedings of the International Conference on Timed Petri Nets*, page 106, 1985.

[Moo02] T. Moors. A critical review of "end-to-end arguments in system design". In *2002 IEEE International Conference on Communications. Conference Proceedings. ICC 2002 (Cat. No.02CH37333)*, volume 2, pages 1214–1219 vol.2, April 2002.

[Myr78] J. Myrheim. Statistical geometry. CERN preprint TH.2538, August 1978.

[Nar88] M.J. Narasimha. The batcher-banyan self-routing network: universality and simplification. *Communications, IEEE Transactions on*, 36(10):1175–1178, Oct 1988.

[Nat98] B. Natvig. *Pålitelighetsanalyse med teknologiske anvendelser*. University of Oslo Compendium, Oslo, Norway, 1998.

[Nel95] Randolph Nelson. *Probability, stochastic processes, and queueing theory: the mathematics of computer performance modeling*. Springer-Verlag New York, Inc., New York, NY, USA, 1995.

[New03] M.E.J. Newman. The structure and function of complex networks. *SIAM Review*, 45:167–256, 2003.

[NLK13] E. Nkposong, T. LaBerge, and N. Kitajima. Experiences with bgp in large scale data centers: Teaching an old protocol new tricks. Technical report, Microsoft, 2013.

[NRC81] U.S. Nuclear Regulatory Commission NRC. *Fault Tree Handbook*. NUREG-0492, Springfield, 1981.

[OO14] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*, USENIX ATC'14, pages 305–320, Berkeley, CA, USA, 2014. USENIX Association.

[pap19] White paper. The new rules of sampling. Technical report, Honeycomb.com, 2019.

[PBMW98] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. *Technical report, Stanford University, Stanford, CA*, 1998.

[Pea88] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgen Kaufmann, San Francisco, 1988.

[Pea00] J. Pearl. *Causality*. Cambridge University Press, Cambridge, 2000.

[PS06] A. Gonzalez Prieto and R. Stadler. Adaptive distributed monitoring with accuracy objectives. *ACM SIGCOMM workshop on Internet Network Management (INM 06), Pisa, Italy*, 2006.

[Rea90] J. Reason. *Human Error*. Cambridge, Cambridge, 1990.

[Rec97] ITU-T Recommendation. X.509 (1997 e): Information technology - open systems interconnection - the directory: Authentication framework. Technical report, 1997.

[SBS99] R. Sekar, T. Bowen, and M. Segal. On preventing intrusions by process behaviour monitoring. *Proceedings of the workshop on intrusion detection and network monitoring*, USENIX, 1999.

[SC07] Yizhan Sun and Alva Couch. *Handbook of Network and System Administration*, chapter Complexity of System Configuration Management. Elsevier, 2007.

[Sea83] J.R. Searle. *Intentionality*. Cambridge University Press, Cambridge, 1983.

[Sha40] C.E. Shannon. *An algebra for theoretical genetics*. Massachusetts Institute of Technology, Dept. of Mathematics, 1940.

[SKAEMW13] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes. Omega: Flexible, scalable schedulers for large compute clusters. In *Proceedings of the 8th ACM European Conference on Computer Systems*, EuroSys '13, pages 351–364, New York, NY, USA, 2013. ACM.

[Sor03] R.D. Sorkin. Causal sers: Discrete gravity. *arXiv:gr-qc/0309009*, 2003.

[SPB+03] T.H. Stang, F. Pourbayat, M. Burgess, G. Canright, K. Engø, and Å. Weltzien. Archipelago: A network security analysis tool. In *Proceedings of The 17th Annual Large Installation Systems Administration Conference (LISA 2003)*, San Diego, California, USA, October 2003.

[SS97] M.I. Seltzer and C. Small. Self-monitoring and self-adapting operating systems. *Proceedings of the Sixth workshop on Hot Topics in Operating Systems,Cape Cod, Massachusetts, USA. IEEE Computer Society Press*, 1997.

[SS03] M. Steinder and A. Sethi. A survey of fault localization techniques in computer networks. *Science of Computer Programming*, 53:165, 2003.

[Sto86] M. Stonebraker. The case for shared nothing architecture. *Database Engineering*, 9(1), 1986.

[SW49] C.E. Shannon and W. Weaver. *The mathematical theory of communication*. University of Illinois Press, Urbana, 1949.

[Tai88] J. Tainter. *The Collapse of Complex Societies*. Cambridge, 1988.

[Tal12] N.N. Taleb. *Antifragile: Things that Gain from Disorder*. Allen Lane, London, UK, 2012.

[TDW+12] A. Thomson, T. Diamond, S.C. Weng, K. Ren, P. Shao, and D.J. Abadi. Calvin: Fast distributed transactions for partitioned database systems. In *Proceedings of the 2012 ACM SIGMOD International Conference on*

Management of Data, SIGMOD '12, pages 1–12, New York, NY, USA, 2012. ACM.

[TH98] S. Traugott and J. Huddleston. Bootstrapping an infrastructure. *Proceedings of the Twelth Systems Administration Conference (LISA XII) (USENIX Association: Berkeley, CA)*, page 181, 1998.

[t'H14] G. t'Hooft. The cellular automaton interpretation of quantum mechanics. *arXiv:1405.1548 [quant-ph]*, 2014.

[Top72] J. Topping. *Errors of Observation and their Treatment*. Chapman and Hall, 1972.

[Tra02] S. Traugott. Why order matters: Turing equivalence in automated systems administration. *Proceedings of the Sixteenth Systems Administration Conference (LISA XVI) (USENIX Association: Berkeley, CA)*, page 99, 2002.

[VGS+17] A. Verbitski, A. Gupta, D. Saha, M. Brahmadesam, K. Gupta, R. Mittal, S. Krishnamurthy, S. Maurice, T. Kharatishvili, and X. Bao. Amazon aurora: Design considerations for high throughput cloud-native relational databases. In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, pages 1041–1052, New York, NY, USA, 2017. ACM.

[Wam16] D. Wampler. *Fast Data Architectures for Streaming Applications*. O'Reilly, 2016.

[WDSC07] F. Wuhib, M. Dam, R. Stadler, and A. Clemm. Robust monitoring of network-wide aggregates through gossiping. In *10th IFIP/IEEE International Symposium on Integrated Management (IM 2007)*, 2007.

[Wes99] G.B. West. The origin of universal scaling laws in biology. *Physica A*, 263:104–113, 1999.

[WF45] J.A. Wheeler and R.P. Feynman. Interaction with the absorber as the mechanism of radiation. *Reviews of Modern Physics*, 17:157–161, 1945.

[WG06] IEEE 1044 WG. Ieee standard classification for software anomalies, 1992-2006.

[Wig17] A. Wiggins. The twelve-factor app. 12factor.net, 2017.

[Wik17] Wikipedia. Wirth's law. Wikipedia, 2017.

[ZKT19] H. Zenil, N.A. Kiani, and J. Tegnér. The thermodynamics of network coding and an algorithmic refinement of the principle of maximum entropy. *Entropy*, 21(560), 2019.

[ZSHD04] W.X. Zhou, S. Sornette, R.A. Hill, and R.I.M. Dunbar. Discrete hierarchical organization of social group sizes. *Proc. Royal Soc.*, 272:439–444, 2004.

INDEX

About the author

Mark Burgess is a British theoretical physicist, turned computer scientist, living in Oslo, Norway. After authoring and consulting for the IT industry and holding a number of research and teaching positions, he was appointed full Professor in the field of Networks and Systems at Oslo University College in 2005, which he held until resigning in 2011 to found the CFEngine company. He is the originator of the globally used CFEngine software as well as founder of CFEngine AS, Inc, and the Research and Innovation consulting company ChiTek-i AS, which focuses on educating and assisting companies in the solution of challenging systemic problems. He is the author of many books and scientific publications, and is a frequent speaker at international events.

Mark Burgess may be found at `www.markburgess.org`, and on Twitter under the name `@markburgess_osl`.