Understanding Kubernetes Through Real-World Phenomena and Analogies

Lucas Käldström - CNCF Ambassador

May 19, 2022 – Valencia

Image credit: CNCF

$ whoami

CLOUD NATIVE COMPUTING FOUNDATION AMBASSADOR

CERTIFIED kubernetes ADMINISTRATOR

Lucas Käldström, 3rd-year BSc student at Aalto University, Finland

CNCF Ambassador, **Certified Kubernetes Administrator**

and **Emeritus Kubernetes WG/SIG Lead**

KubeCon Speaker in **Berlin**, **Austin**,

Copenhagen, Shanghai, Seattle, San Diego & Valencia

KubeCon Keynote Speaker in **Barcelona**

Former Kubernetes approver and subproject owner,

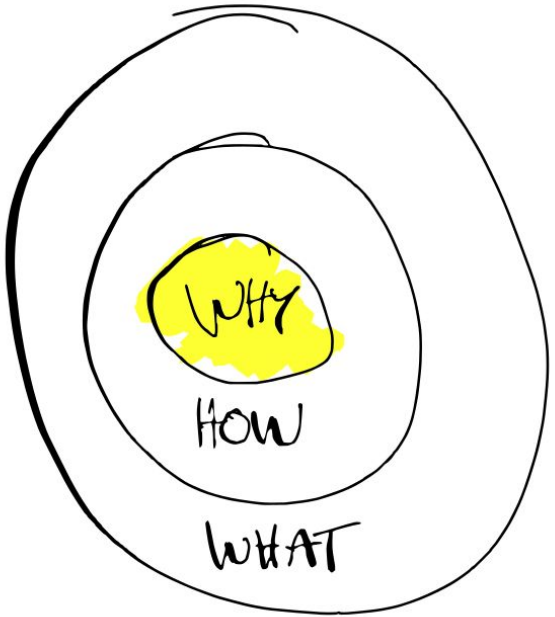
active in the OSS community for 6+ years.

Worked on e.g. SIG Cluster Lifecycle => *kubeadm* to GA.

Weaveworks contractor, **Weave Ignite & libgitops** author

Cloud Native Nordics co-founder & meetup organizer

Guild of **Automation and Systems Technology** CFO



Credits to Simon Sinek

What problem are we trying to solve?

Based on decades of experience

Large-scale cluster management at Google with Borg

Abhishek Verma[†] Luis Pedrosa[‡] Madhukar Korupolu
David Oppenheimer Eric Tune John Wilkes
Google Inc.

Abstract

Google's Borg system is a cluster manager that runs hundreds of thousands of jobs, from many thousands of different applications, across a number of clusters each with up to tens of thousands of machines.

It achieves high utilization by combining admission control, efficient task-packing, over-commitment, and machine sharing with process-level performance isolation. It supports high-availability applications with runtime features that minimize fault-recovery time, and scheduling policies that reduce the probability of correlated failures. Borg simplifies life for its users by offering a declarative job specification language, name service integration, real-time job monitoring, and tools to analyze and simulate system behavior.

We present a summary of the Borg system architecture and features, important design decisions, a quantitative analysis of some of its policy decisions, and a qualitative examination of lessons learned from a decade of operational experience with it.

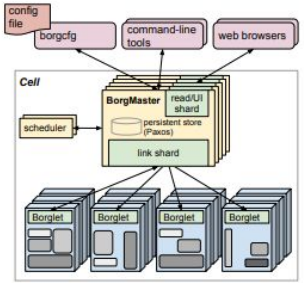


Figure 1: The high-level architecture of Borg. *Only a tiny fraction of the thousands of worker nodes are shown.*

cluding with a set of qualitative observations we have made

Omega: flexible, scalable schedulers for large compute clusters

Malte Schwarzkopf[†*] Andy Konwinski[‡*] Michael Abd-El-Malek[§] John Wilkes[§]
[†]University of Cambridge Computer Laboratory [‡]University of California, Berkeley [§]Google, Inc.
[†]ms705@cl.cam.ac.uk [‡]andyk@berkeley.edu [§]{mabdelmalek,johnwilkes}@google.com

Abstract

Increasing scale and the need for rapid response to changing requirements are hard to meet with current monolithic cluster scheduler architectures. This restricts the rate at which new features can be deployed, decreases efficiency and utilization, and will eventually limit cluster growth. We present a novel approach to address these needs using parallelism, shared state, and lock-free optimistic concurrency control.

We compare this approach to existing cluster scheduler designs, evaluate how much interference between schedulers occurs and how much it matters in practice, present some techniques to alleviate it, and finally discuss a use case highlighting the advantages of our approach – all driven by real-life Google production workloads.

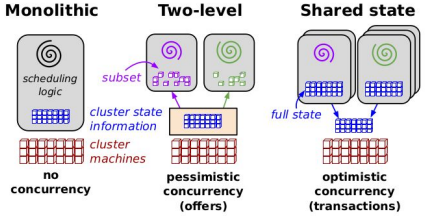
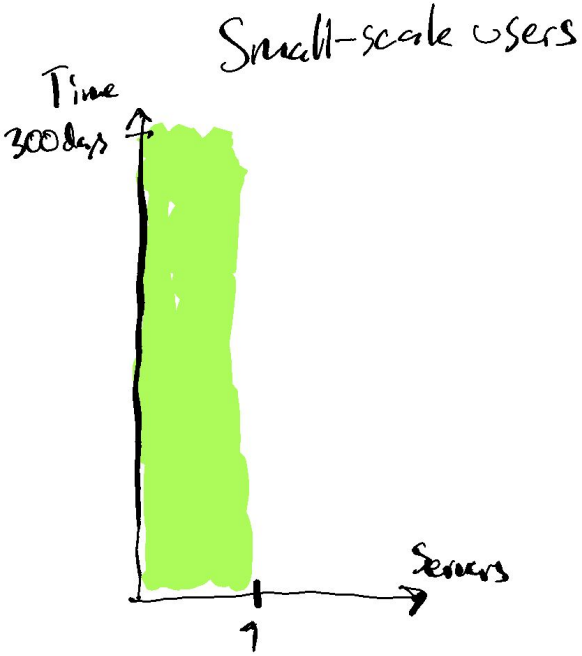


Figure 1: Schematic overview of the scheduling architectures explored in this paper.

 |


But is it inherently “too complex” for most?

Problems hiding in plain sight



It just takes longer for small-scale users to notice problems due to e.g. randomness

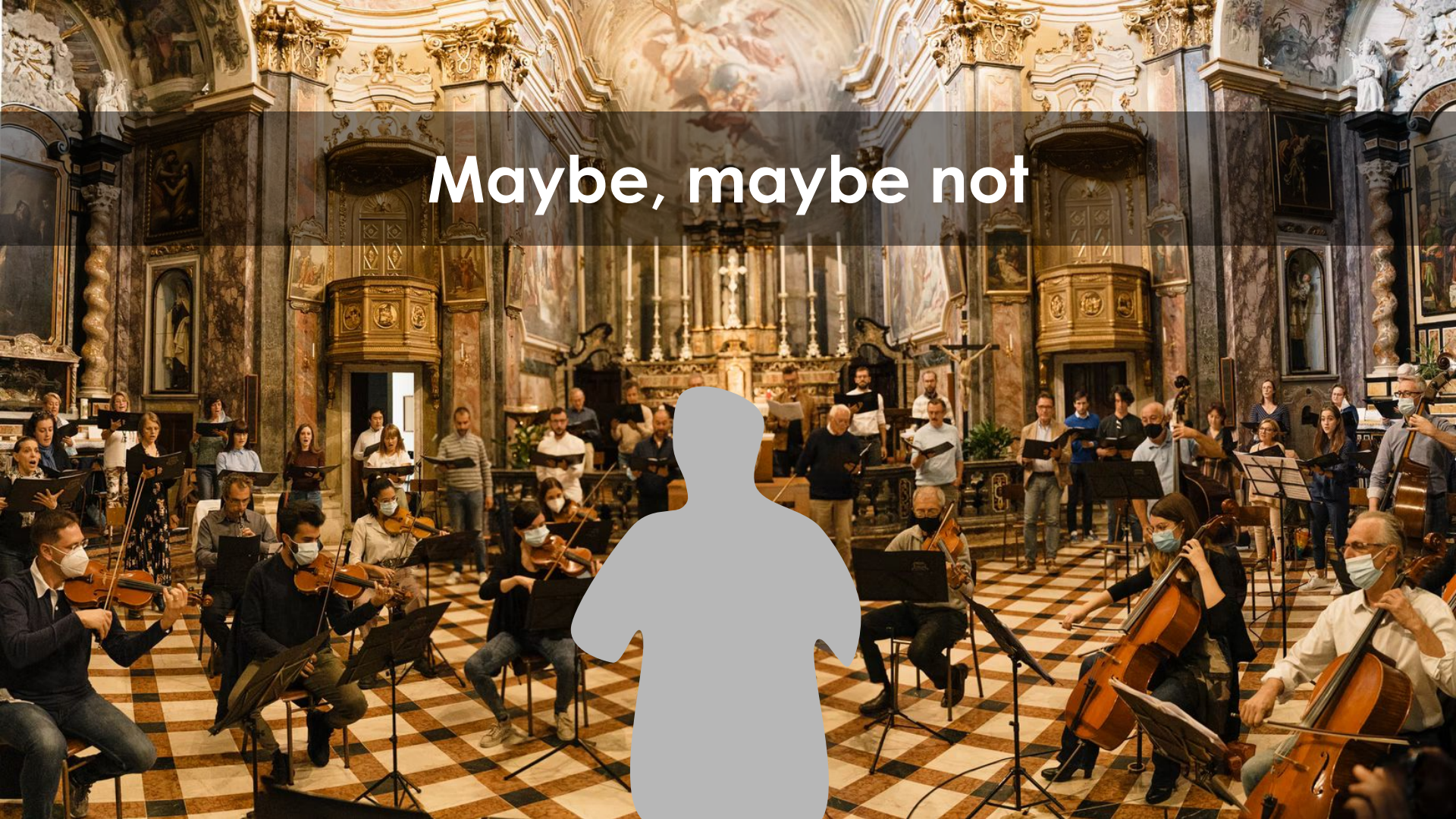

Kubernetes and the Orchestra

So… You wanna play a piece of music

But you have no conductor :(


Will the orchestra itself be able to play?

Maybe, maybe not



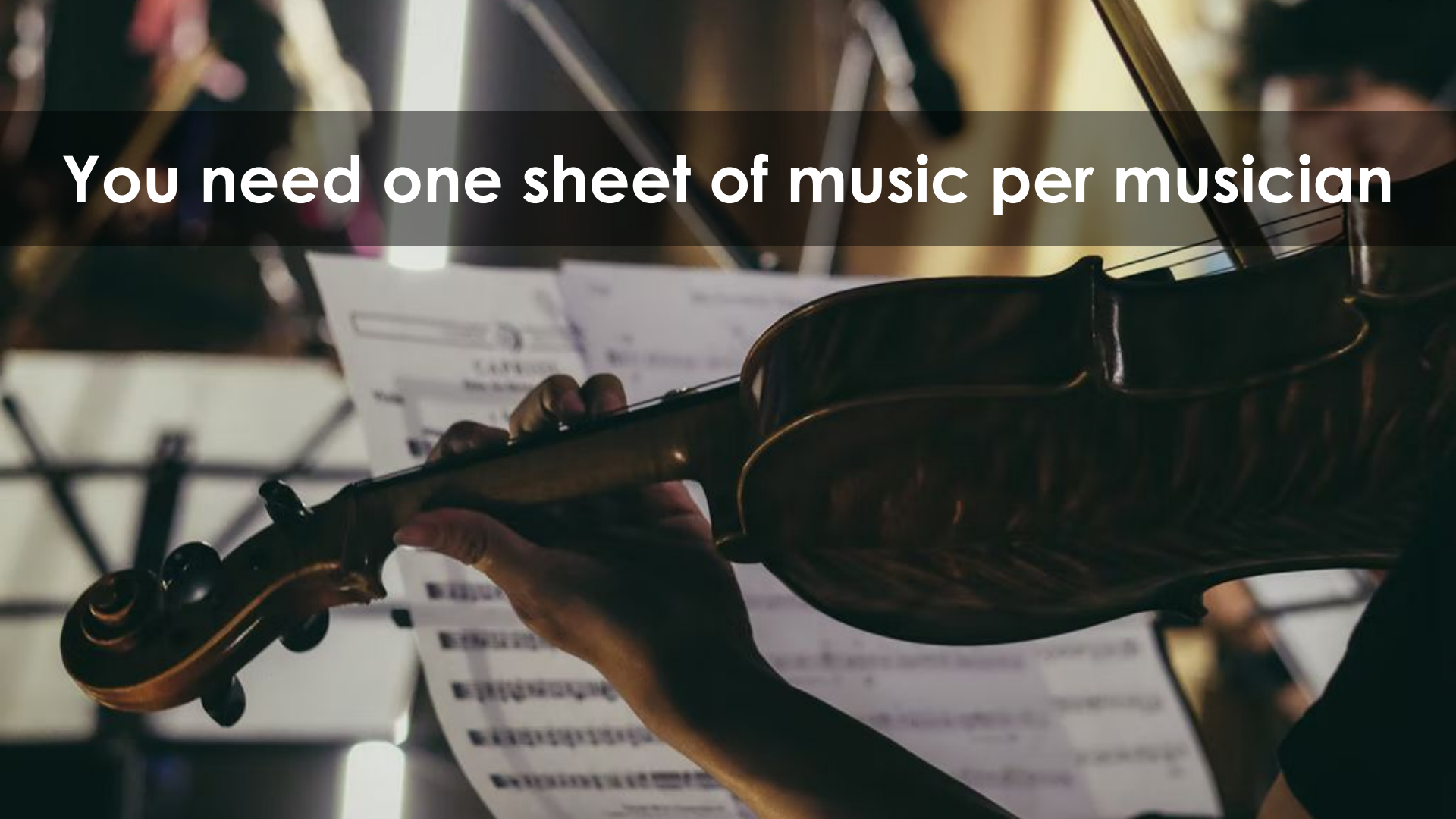
You need a conductor and good musicians

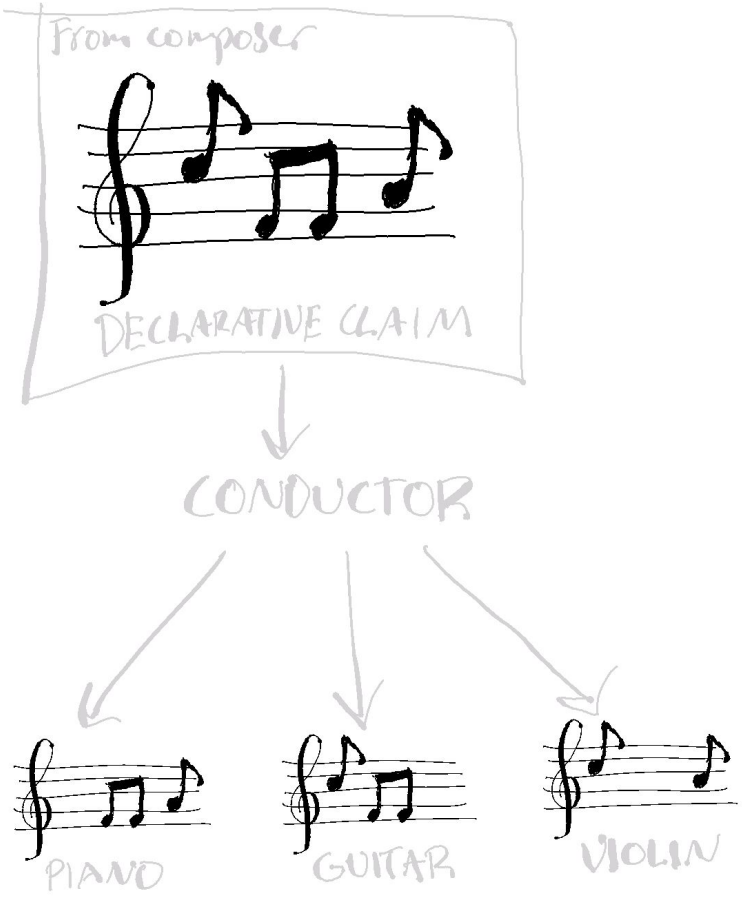
The 4 Whys:

1. The control plane is for Coordination, yet allows Improv

 |

(Joe Beda, 2017)


You need one sheet of music per musician



 |

Declarative vs Imperative

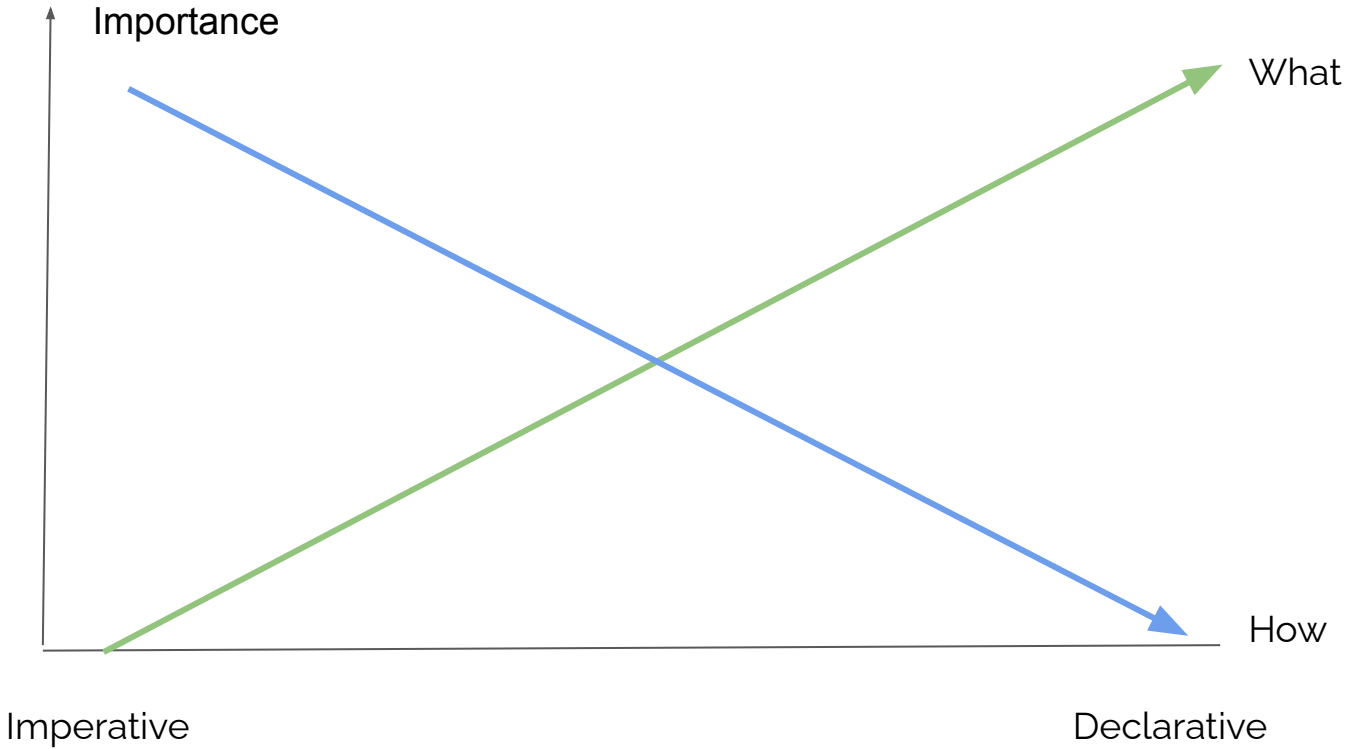
“declarative” = “making a declaration”

“declare” = “to make known as a determination” (Merriam-Webster, 2021)

Declarative: “The door is shut” (state)

Imperative: “Shut the door!” (action)

The Imperative - Declarative spectrum



Imperative – Declarative examples

C – **Imperative**

Haskell – **Declarative**

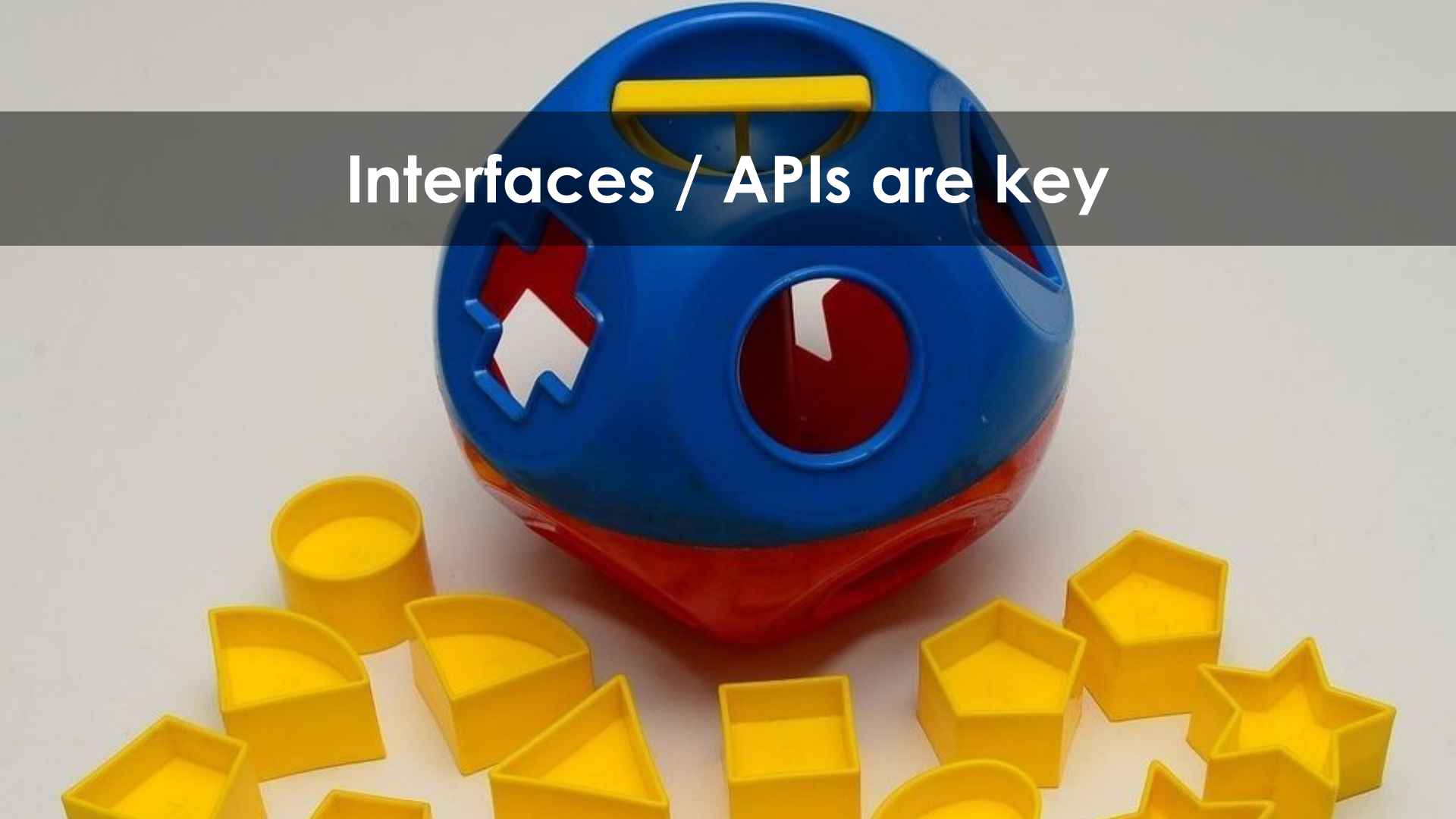
PNG – **Imperative**

SVG – **Declarative**

Manually storing data in files – **Imperative**

SQL – **Declarative**

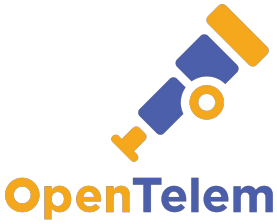
Interfaces / APIs are key

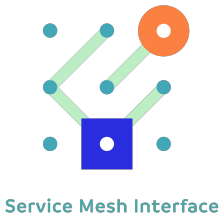


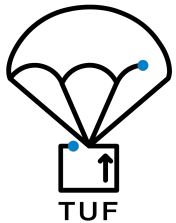
Abstraction Layers: Pluggable interfaces

Cloud Native is all about pluggable APIs forming consistent abstractions that projects can implement and/or rely on.

These CNCF/LF projects contain only a specification, no implementation:





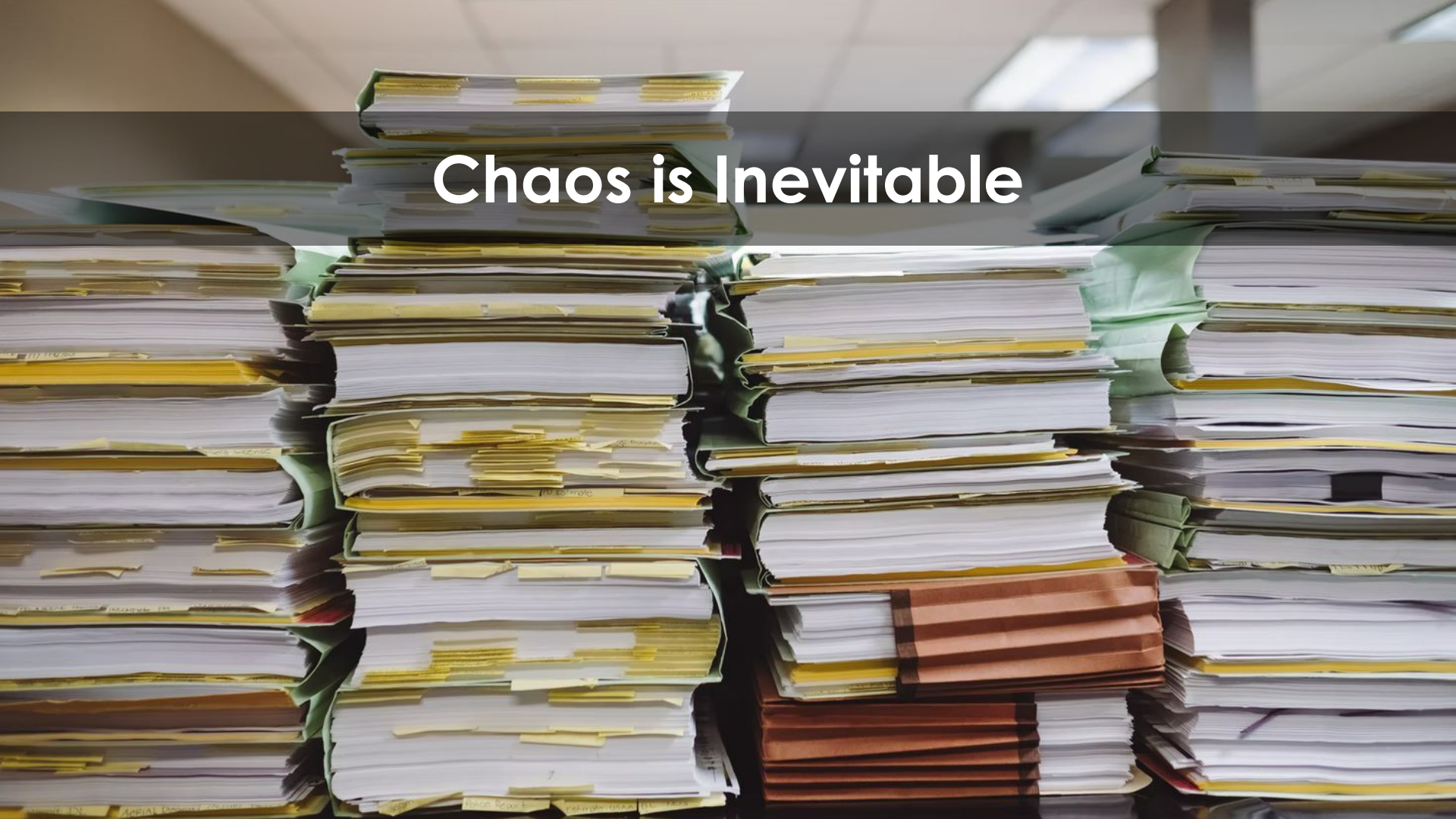


The 4 Whys:

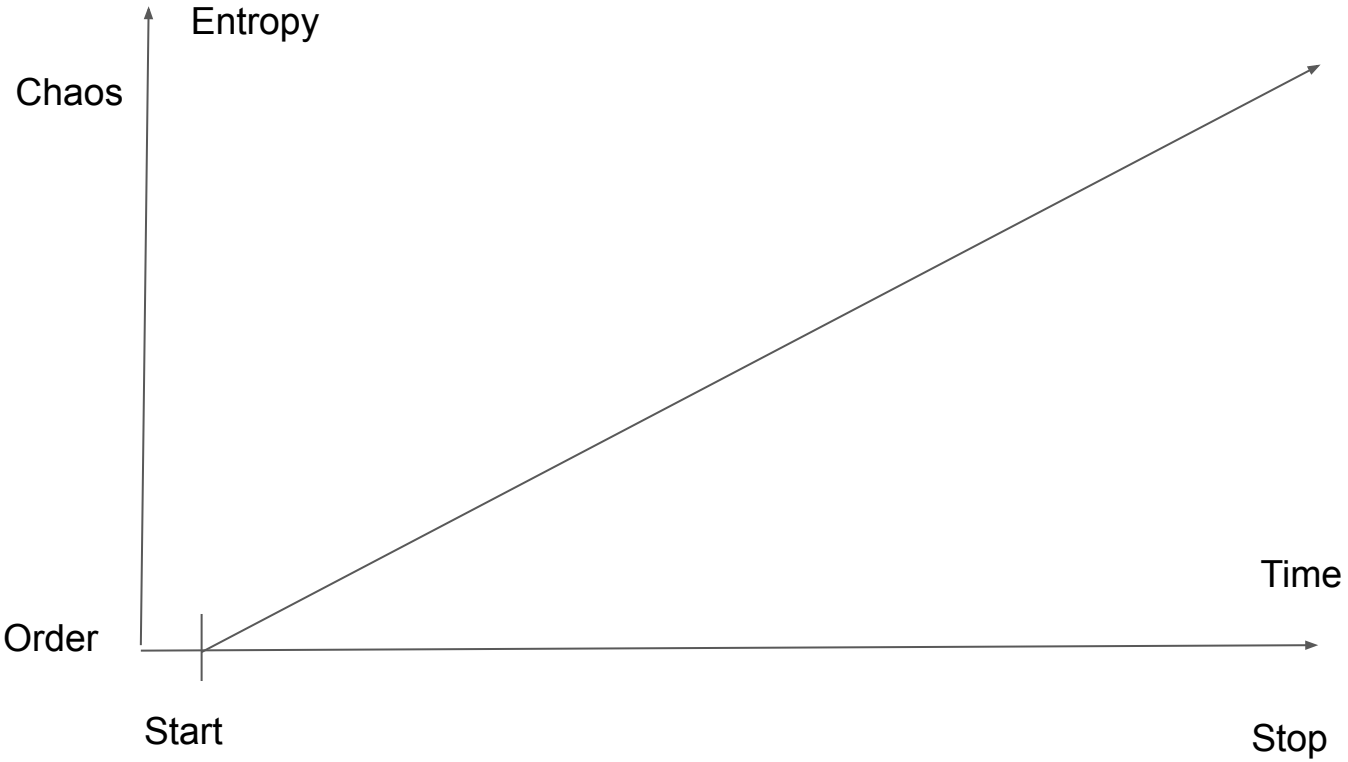
1. The control plane is for Coordination, yet allows Improv

2. **Declarativeness is for Portability and Desired State**

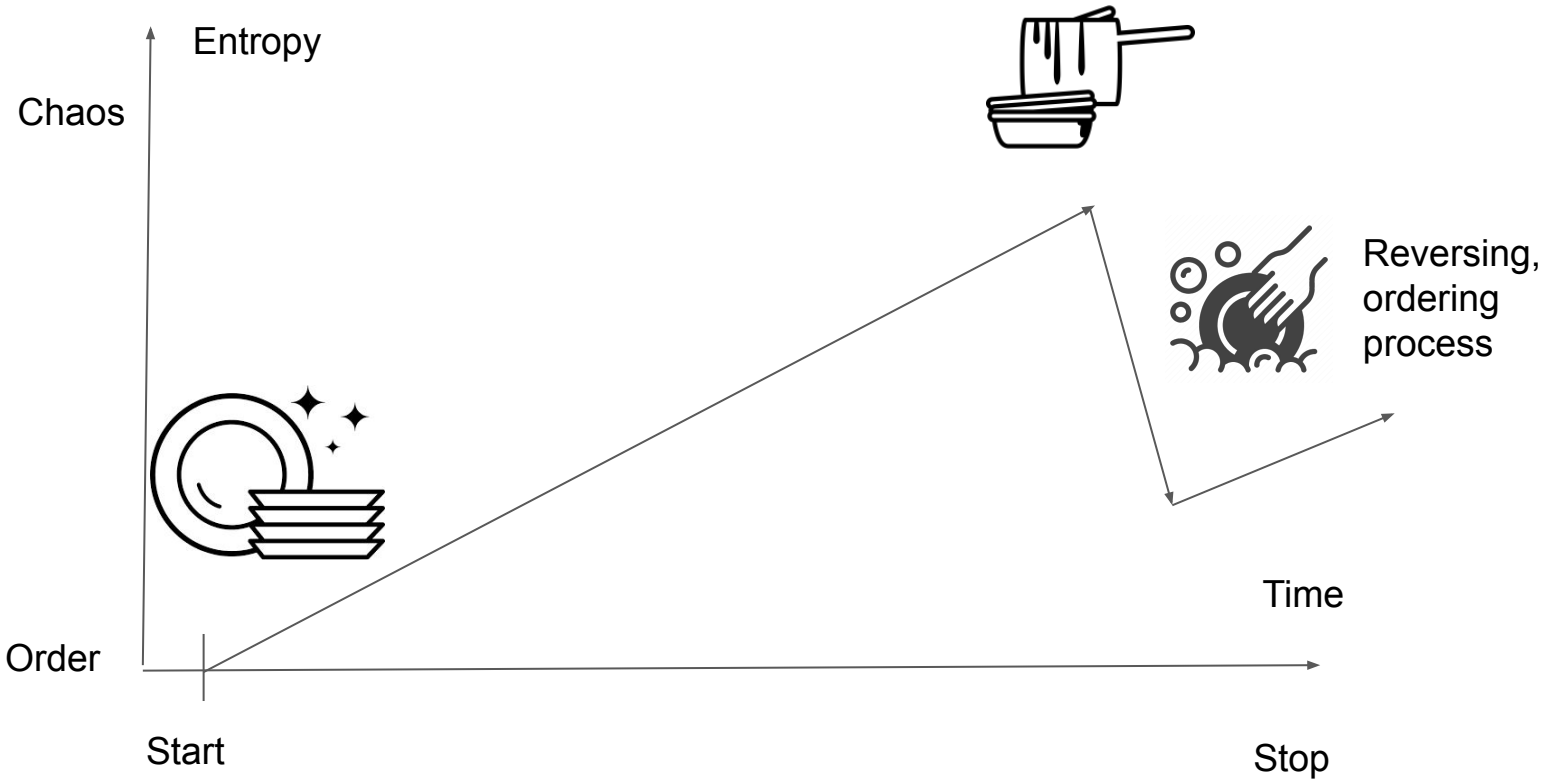
 |


Chaos is Inevitable

Entropy: Systems become less ordered



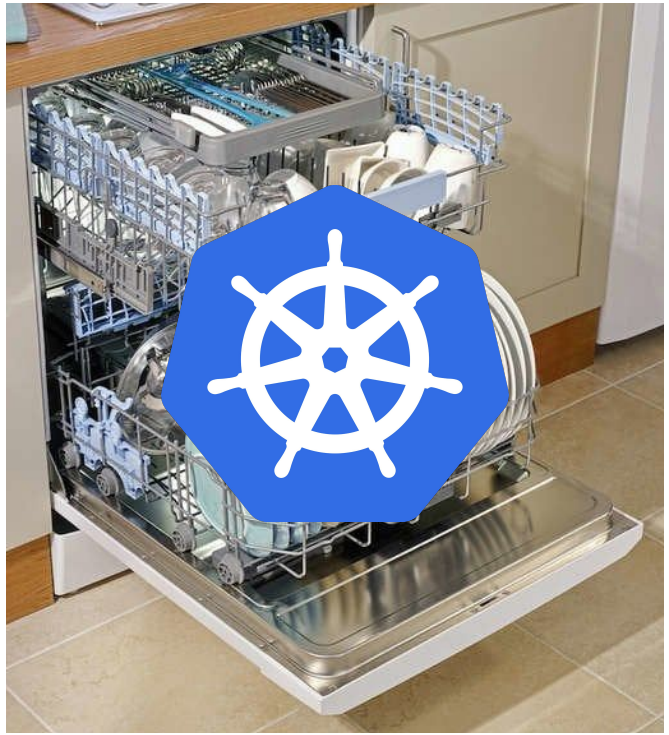
Entropy: Putting order to chaos



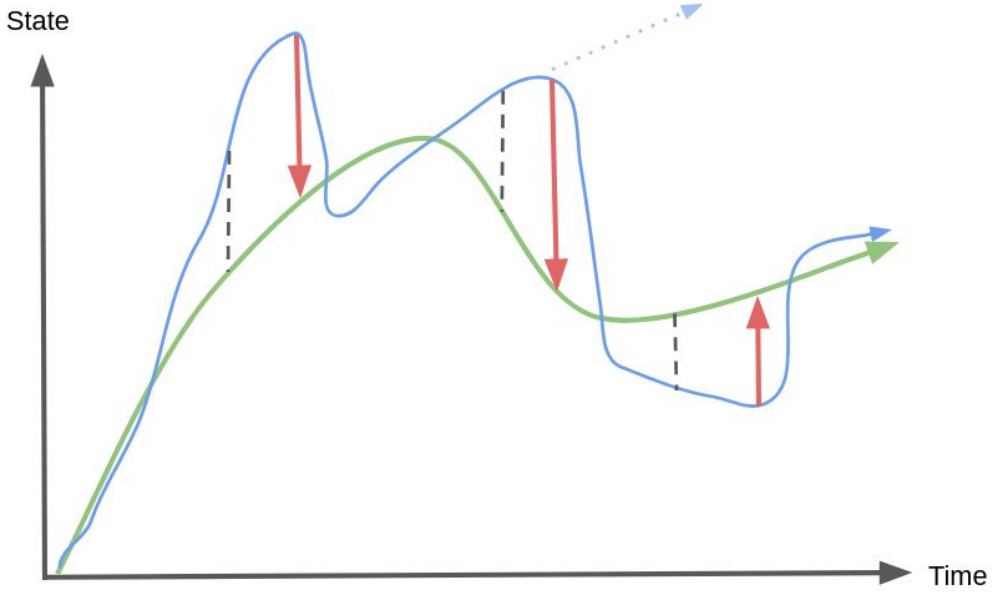
Kubernetes: The dishwasher of servers

Kubernetes: The dishwasher of servers

Kubernetes: The dishwasher of servers



Game Theory: An Infinite Game against Chaos



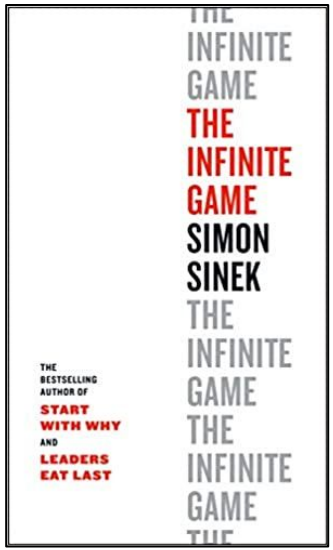
Long-term Policy, “slow”

Fluctuations, “fast”

System State, “slow” + “fast”

Corrective Action

Prediction of state if no action



 |

Key Takeaways

a) Systems are inevitably becoming **less ordered**, and thus

b) need some periodic **corrective action** to steer the course towards

c) some declared **desired state** of the system.

 |

The 4 Whys:

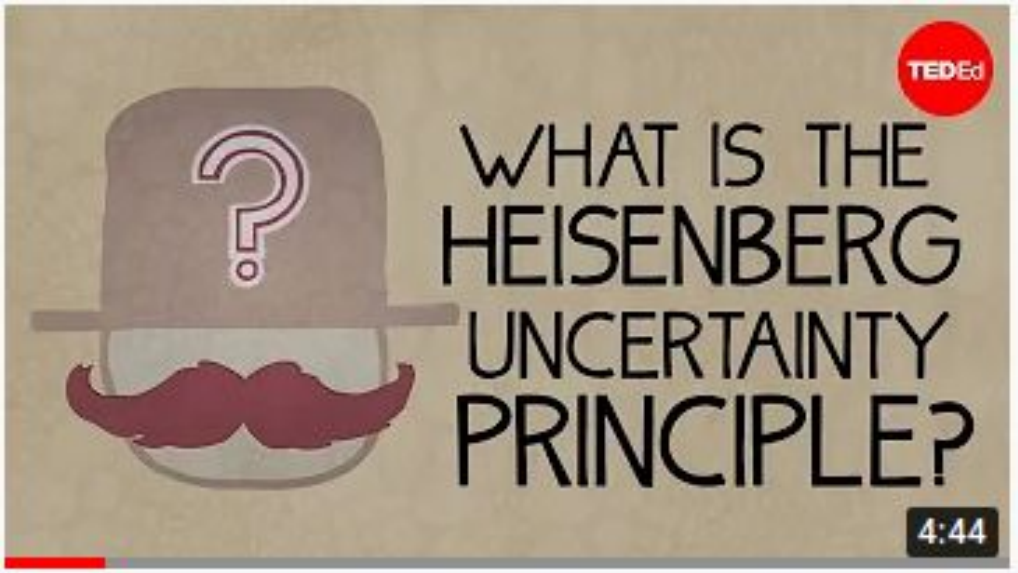
1. The control plane is for Coordination, yet allows Improv

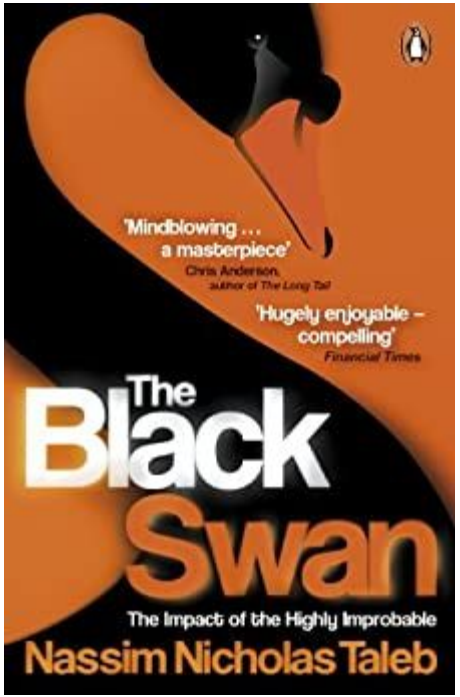
2. Declarativeness is for Portability and Desired State

3. **Periodic action is for fighting inevitable Chaos**

 |

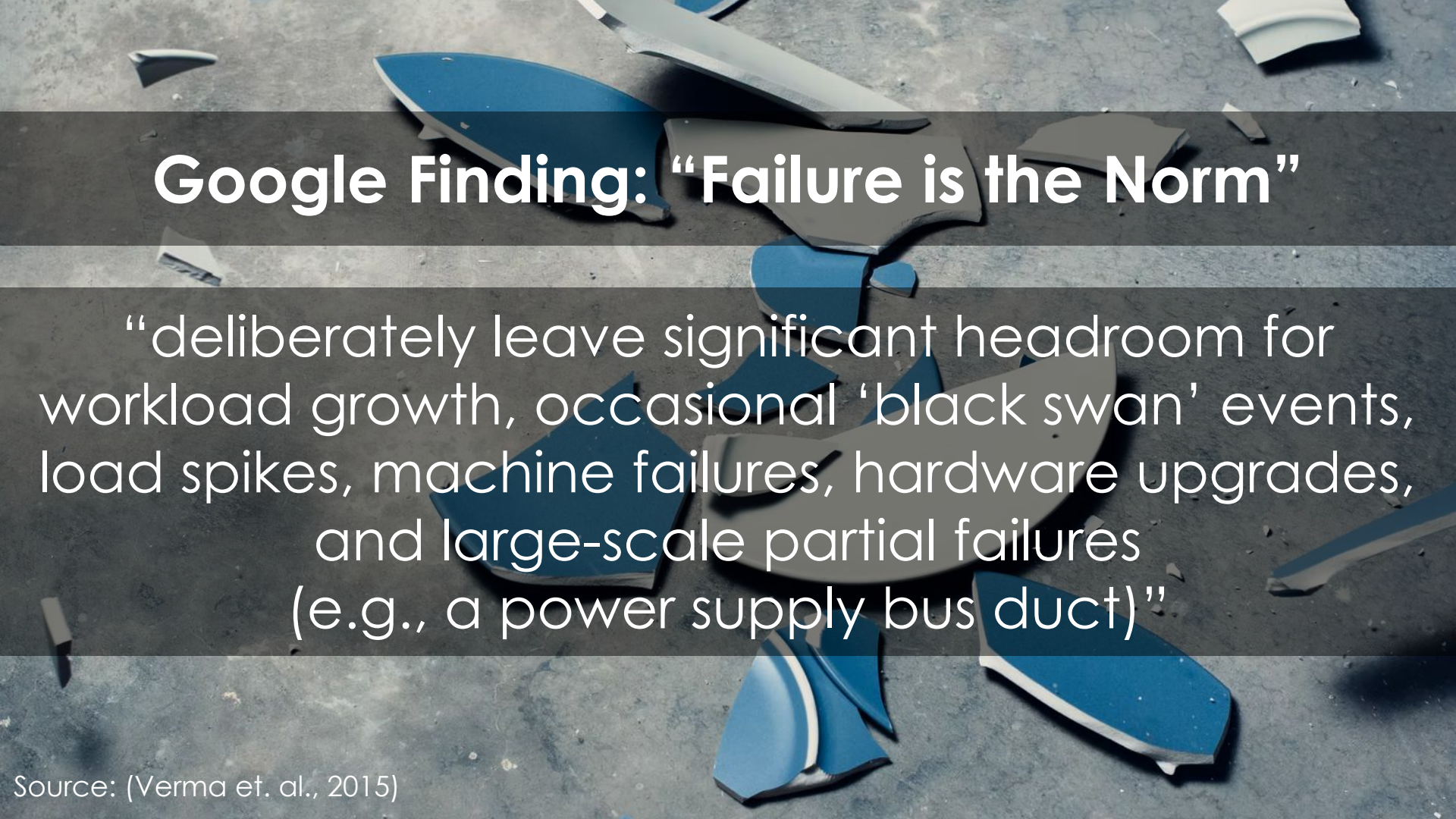
The world is much more random than we believe





 |


Google Finding: “Failure is the Norm”



Google Finding: "Failure is the Norm"

"deliberately leave significant headroom for workload growth, occasional 'black swan' events, load spikes, machine failures, hardware upgrades, and large-scale partial failures (e.g., a power supply bus duct)"

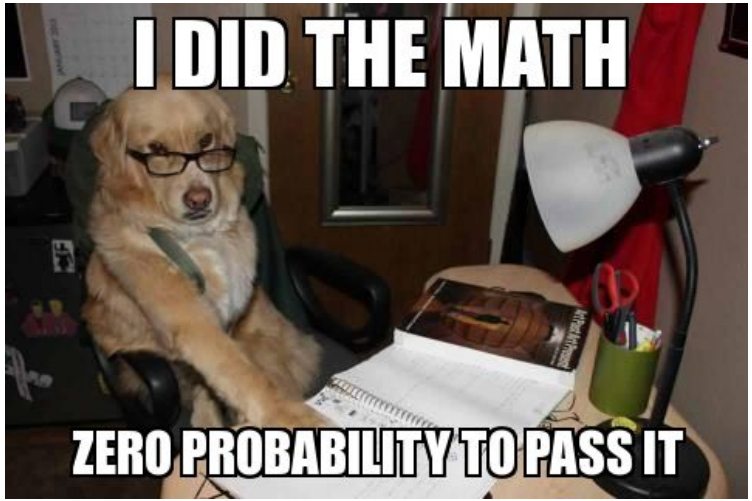
Source: (Verma et. al., 2015)

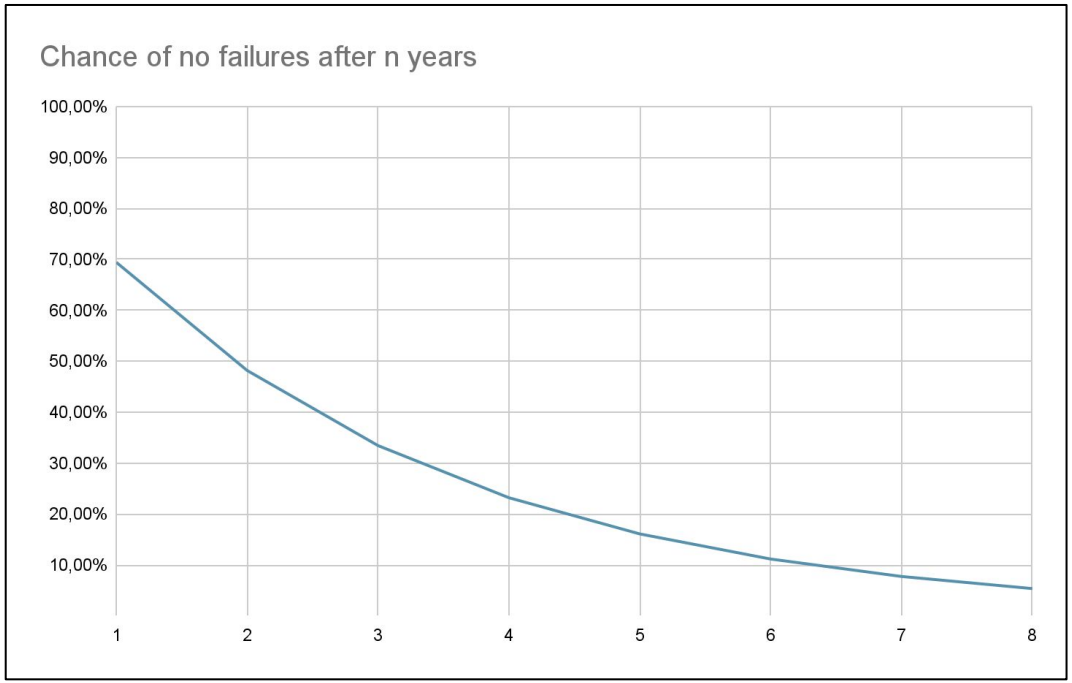
Randomness is Unintuitive

A procedure fails only once in 10000 runs (estimate)

=> 99.99% success probability

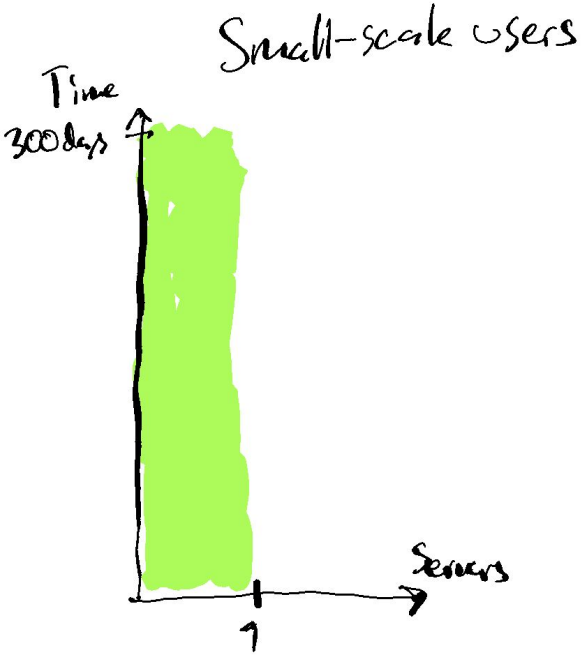
Runs 10 times a day





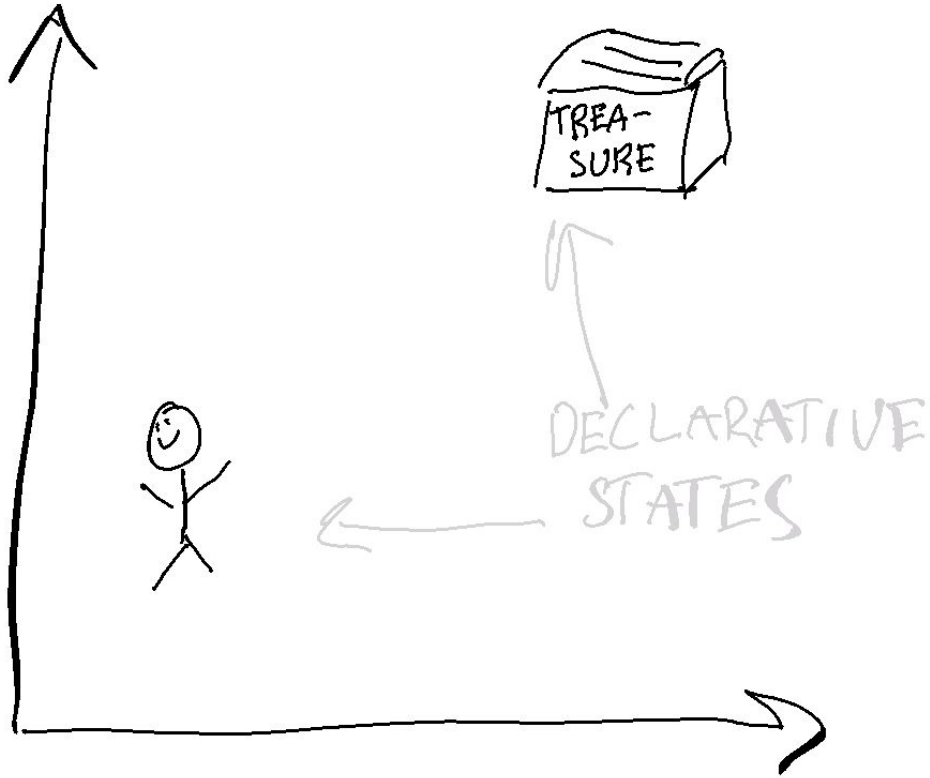
 |

Problems hiding in plain sight

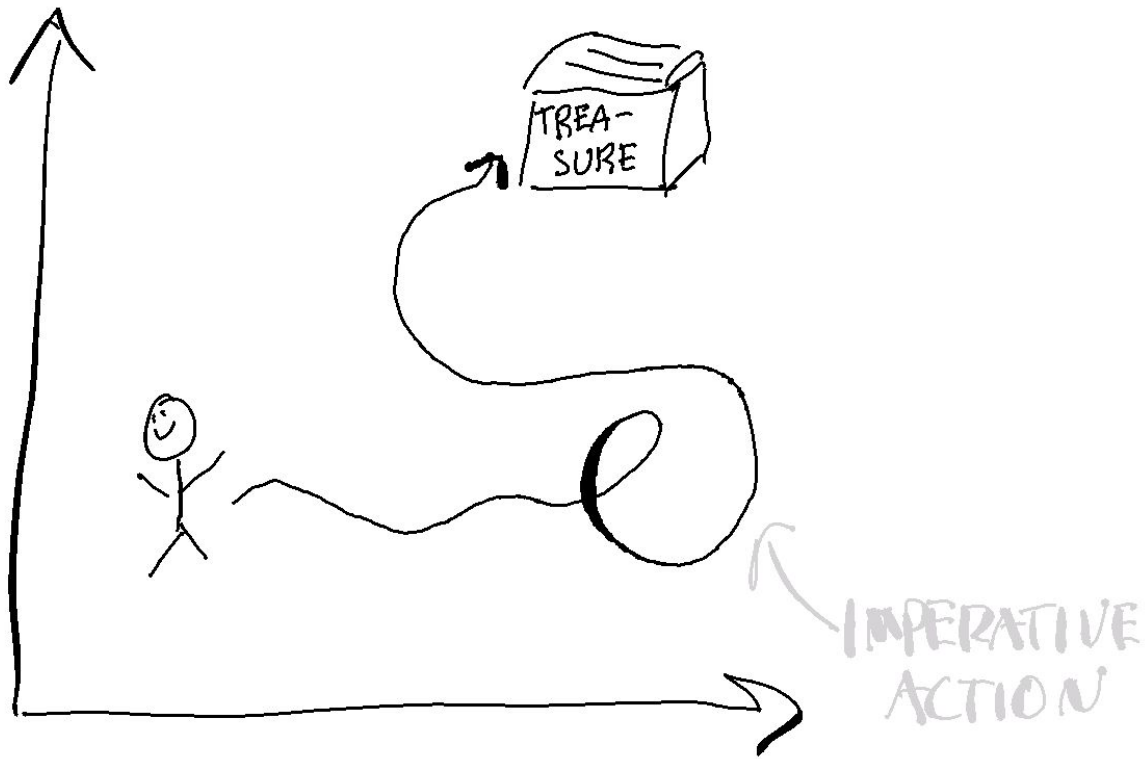


It just takes longer for small-scale users to notice problems due to e.g. randomness

Calling a Taxi is Declarative

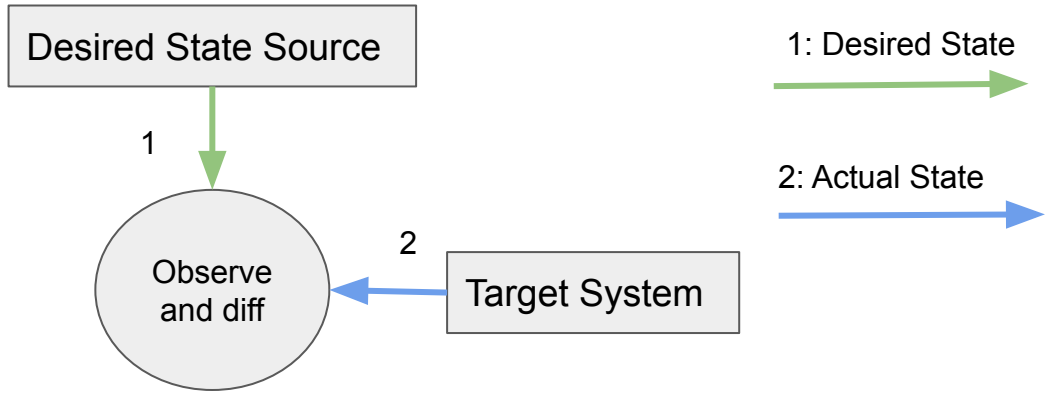


 |

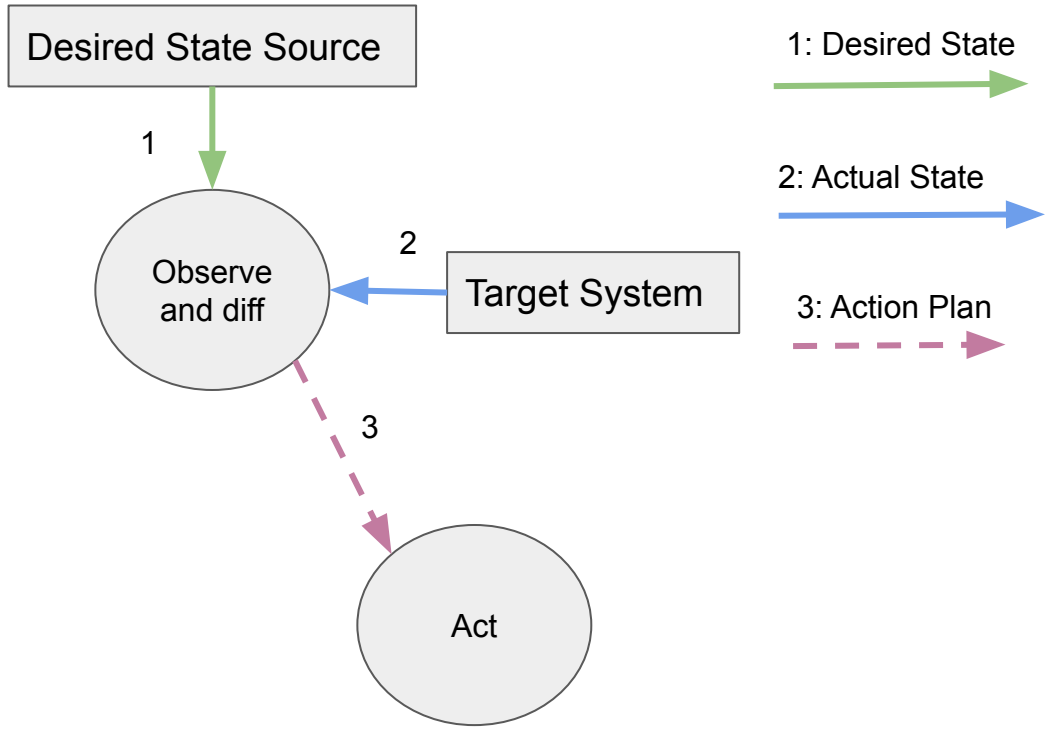


 |

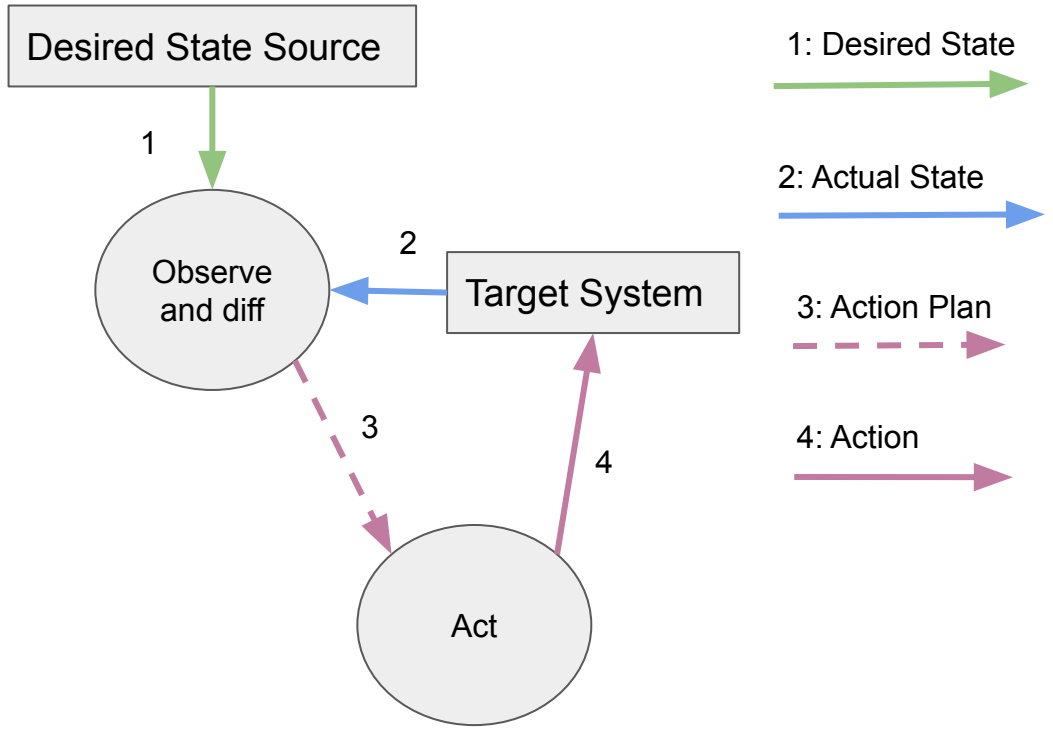
Actuators, or reconcile loops, fulfil the claim(s)



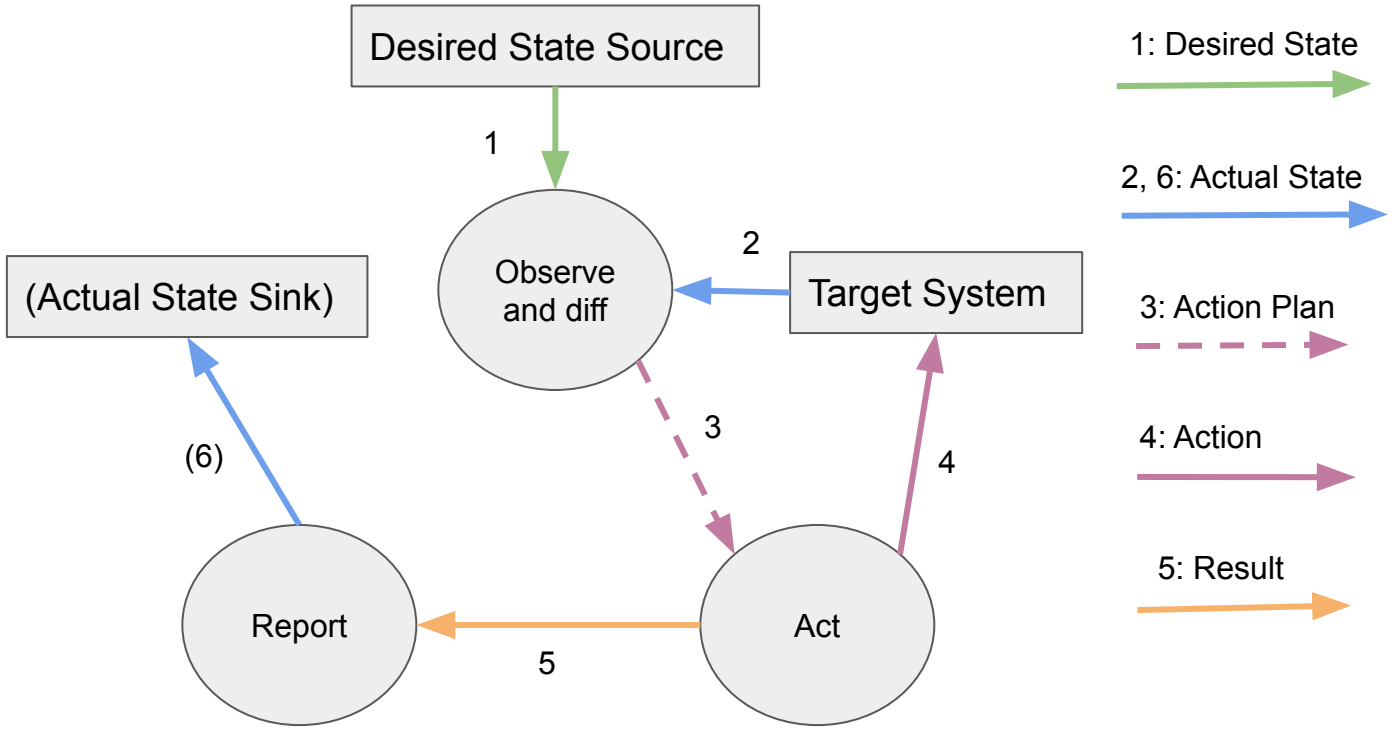
Actuators, or reconcile loops, fulfil the claim(s)



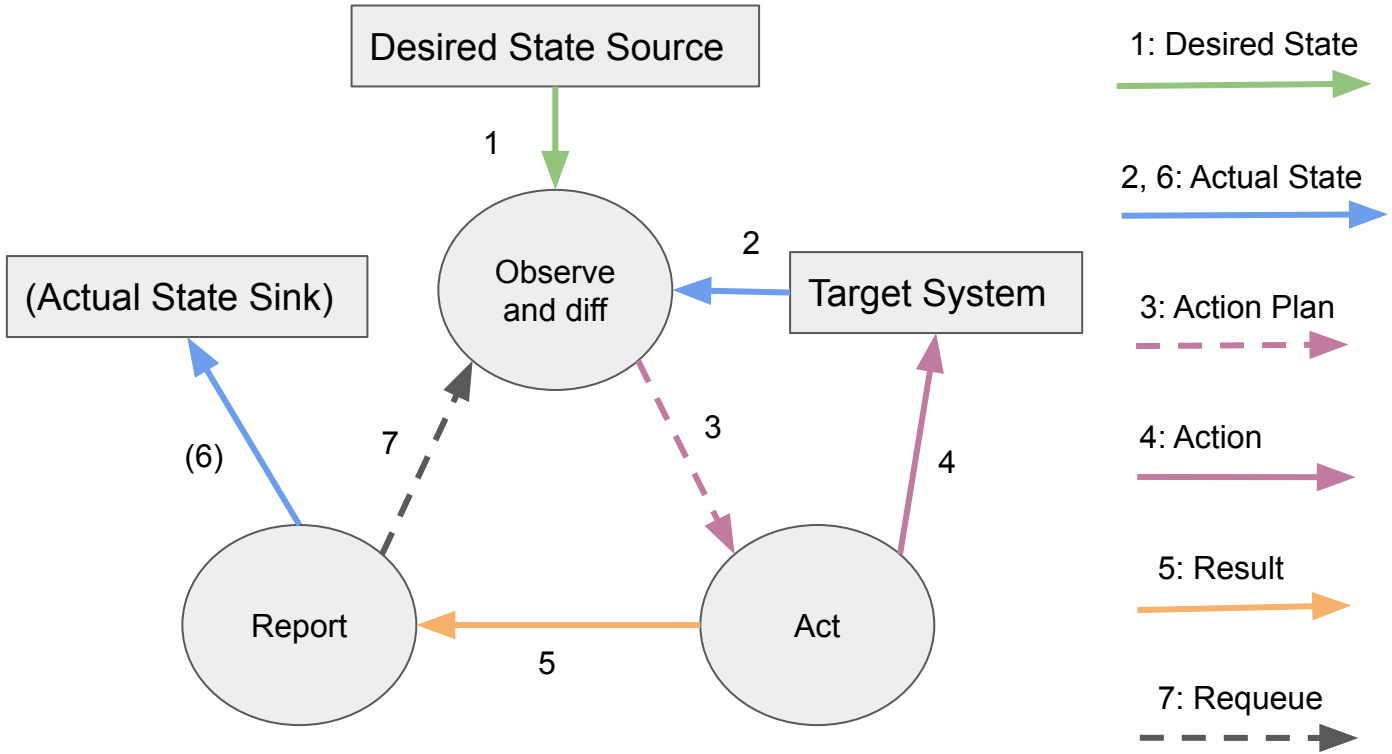
Actuators, or reconcile loops, fulfil the claim(s)



Actuators, or reconcile loops, fulfil the claim(s)



Actuators, or reconcile loops, fulfil the claim(s)



 |

A taxi driver as a reconcile loop

A taxi driver is approached by person P.

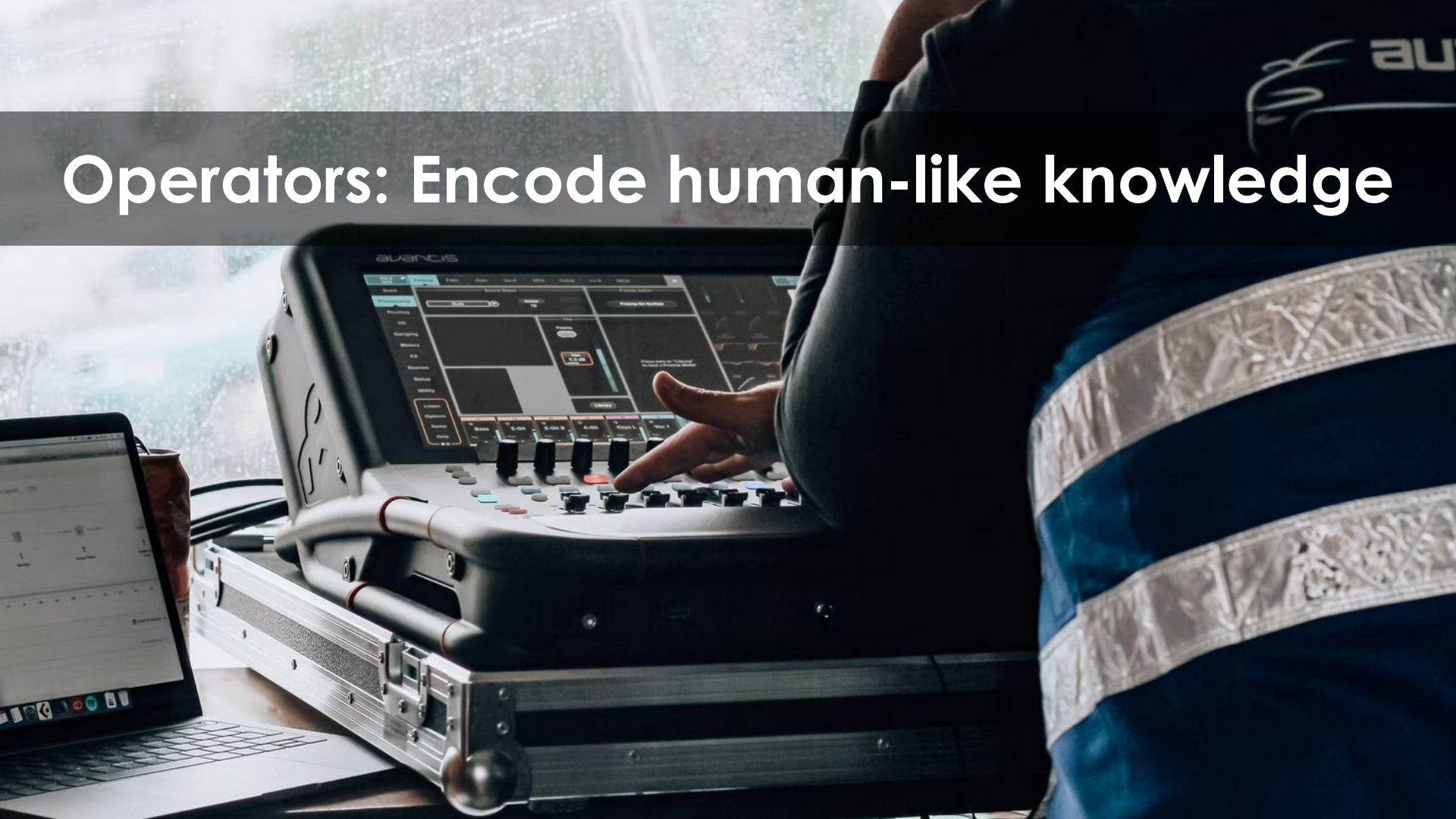
1.	Ask where P wants to go, e.g. the City	“desired state”
2.	Ask where P is, e.g. Fira Valencia	“actual state”
3.	Figure out a route plan	“action plan”
4.	Drive P from Fira Valencia to the City	“action”
5.	Was the drive successful?	“result”
6.	Photograph P for a social media update	“actual state sink”
7.	Find the next client	“requeue”

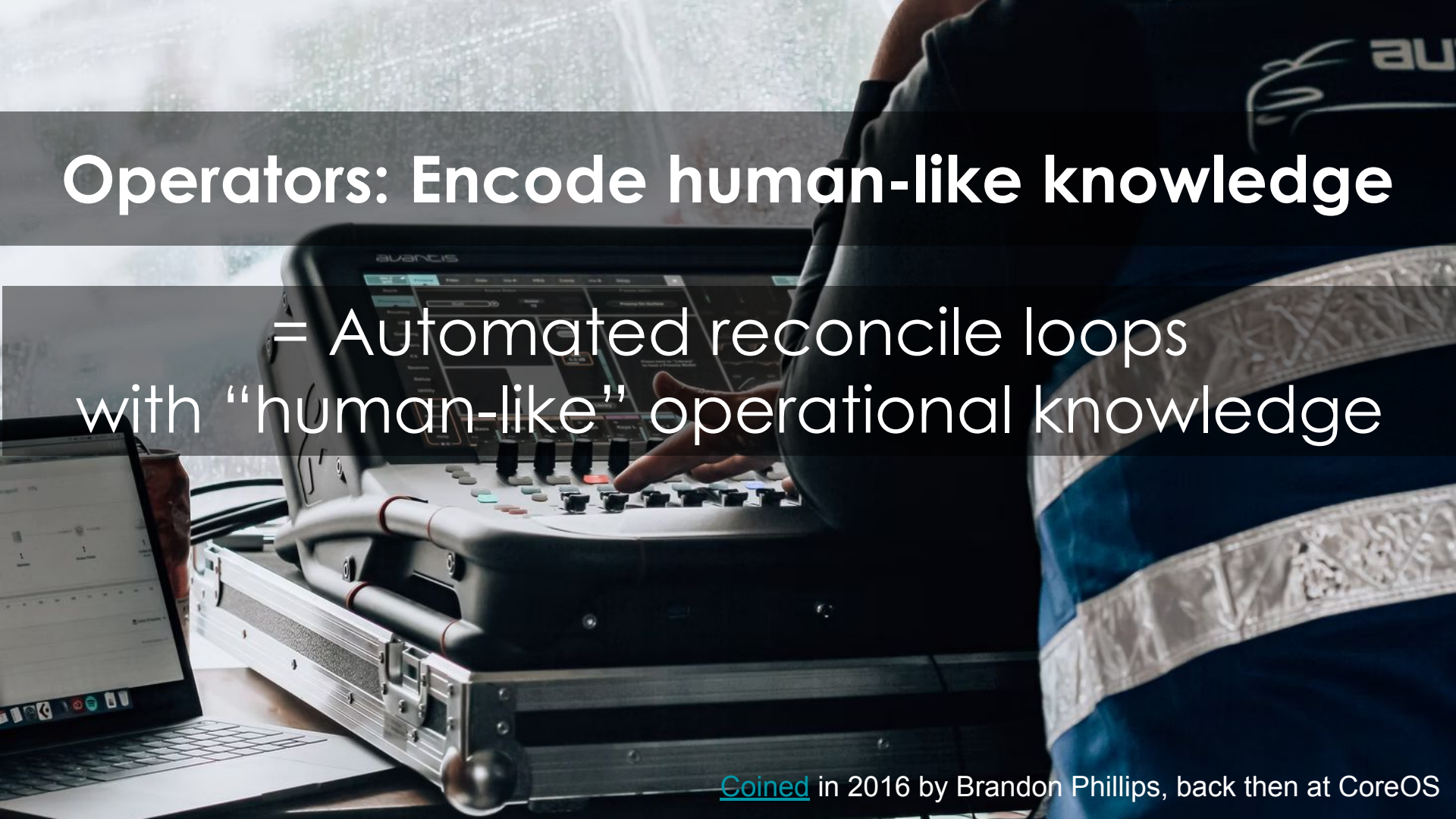
The 4 Whys:

1. The control plane is for Coordination, yet allows Improv
2. Declarativeness is for Portability and Desired State
3. Periodic action is for fighting inevitable Chaos
4. **Designing controllers for failure is for Randomness**

 |

Operators: Encode human-like knowledge



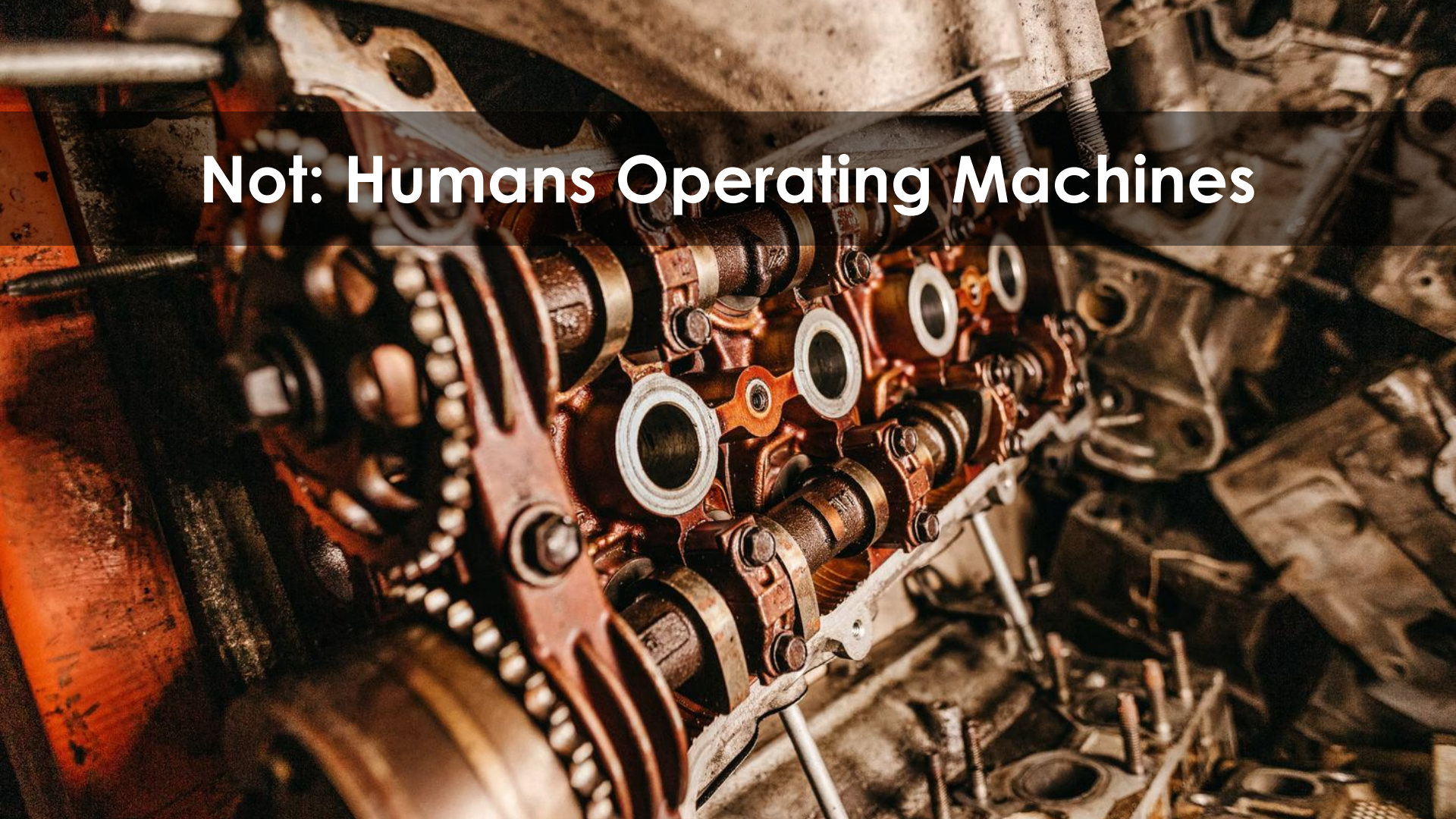


Operators: Encode human-like knowledge

= Automated reconcile loops
with “human-like” operational knowledge

Coined in 2016 by Brandon Phillips, back then at CoreOS

Not: Humans Operating Machines



Instead: Humans Operating Automation that in turn Operate Machines

Further Reading

Check out my thesis for more details!

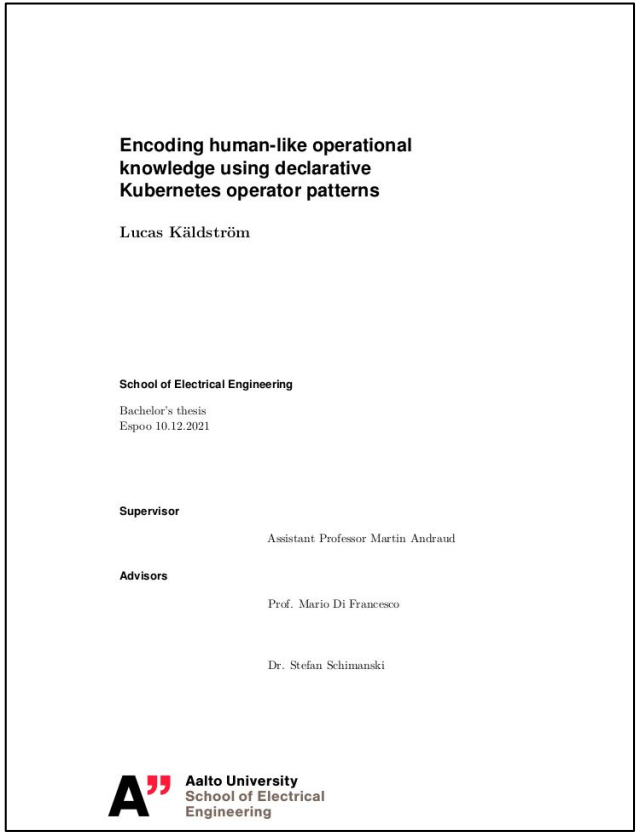
Available openly on Github:
https://github.com/luxas/research

CC-BY-SA 4.0 licensed

Encoding human-like operational

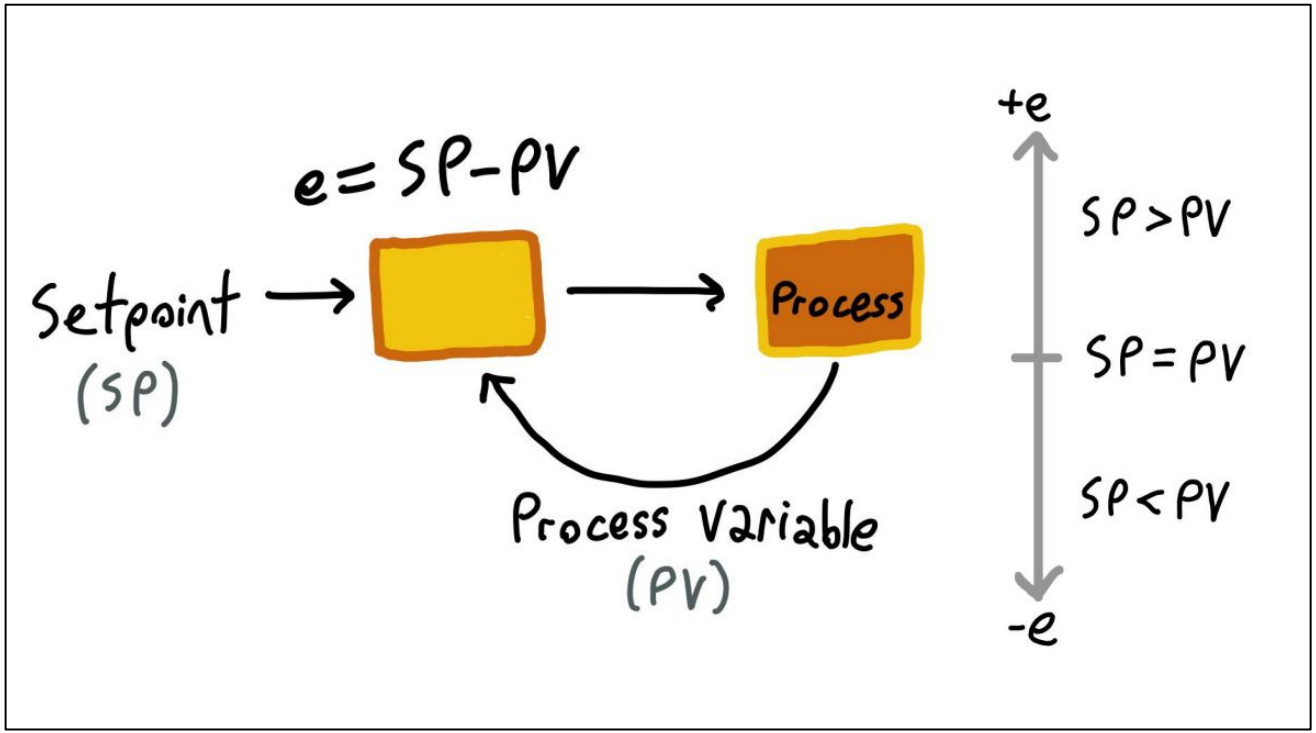
knowledge using declarative

Kubernetes operator patterns



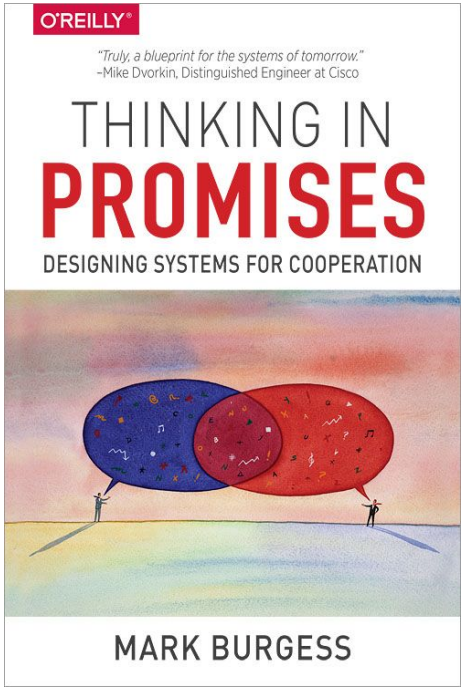
 |

Control Theory



(Vallery Lancery, QCon, 2018)

Promise Theory



[(The Kubernetes Documentary, Honeypot, 2022)]

Wrap-up: The 4 Whys:

1. The control plane is for Coordination, yet allows Improv
2. Declarativeness is for Portability and Desired State
3. Periodic action is for fighting inevitable Chaos
4. Designing controllers for failure is for Randomness

 |

Thank you!

@luxas on Github

@luxas on LinkedIn

@luxas on SpeakerDeck

@kubernetesonarm on Twitter

lucas@luxaslabs.com

Baim Hanif on Unsplash