



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



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.

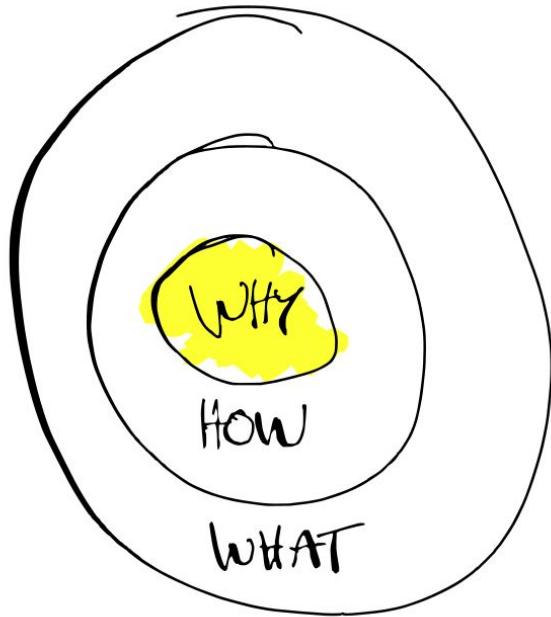
Worked on e.g. SIG Cluster Lifecycle => kubeadm to GA.

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

Cloud Native Nordics co-founder & meetup organizer

Guild of **Automation and Systems Technology** CFO





What problem are we trying to solve?

When you try to solve a problem and end up creating 10 more issues



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.

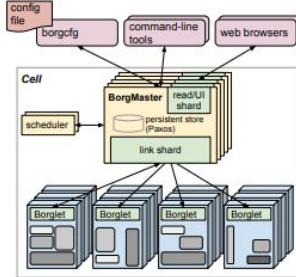


Figure 1: The high-level architecture of Borg. Only a tiny fraction of the thousands of worker nodes are shown.

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.

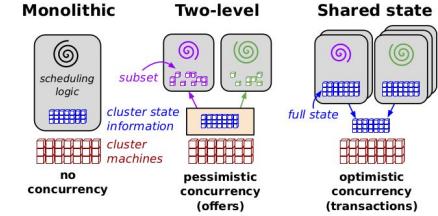


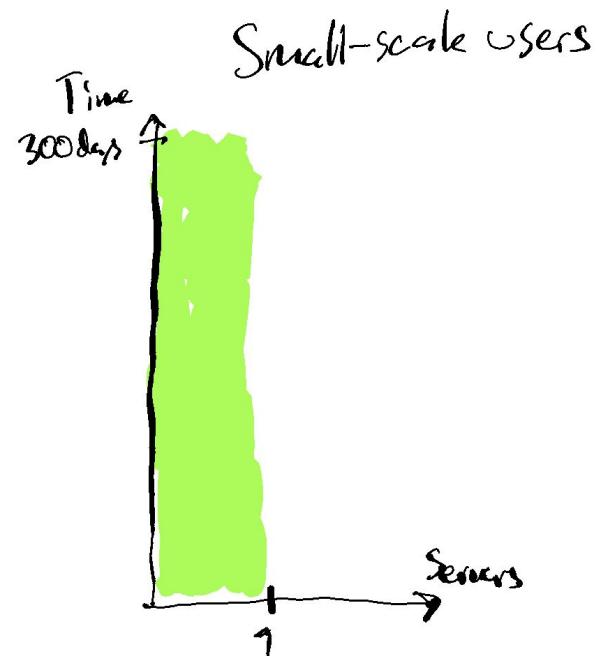
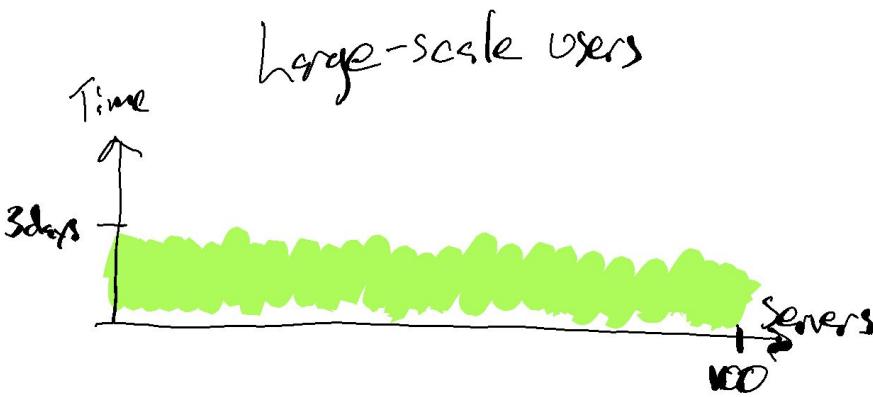
Figure 1: Schematic overview of the scheduling architectures explored in this paper.



A close-up photograph of a complex electronic circuit board, likely a patch panel or modular synthesizer. The board is densely populated with various electronic components, including several black knobs, numerous metal terminals, and small blue cylindrical components. A multitude of wires in vibrant colors—pink, orange, red, green, yellow, and purple—are crisscrossed across the board, some connected to the terminals and others trailing off. The overall impression is one of intricate, manual customization and complexity.

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

A photograph of an orchestra performing on stage. The musicians are seated in rows, playing various instruments like violins, cellos, and double basses. Large sheet music stands are positioned in front of them. The lighting is dramatic, with strong highlights and shadows. The overall atmosphere is one of a formal concert.

So... You wanna play a piece of music

A photograph of a musical performance in a Baroque-style church. The interior is highly decorated with gold leaf, intricate carvings, and colorful frescoes on the ceiling and walls. A large organ loft is visible on the right. In the foreground and middle ground, musicians in dark clothing play violins, cellos, and other instruments. A choir of about二十 people stands in the background, singing from sheet music. The floor is made of large, light-colored tiles.

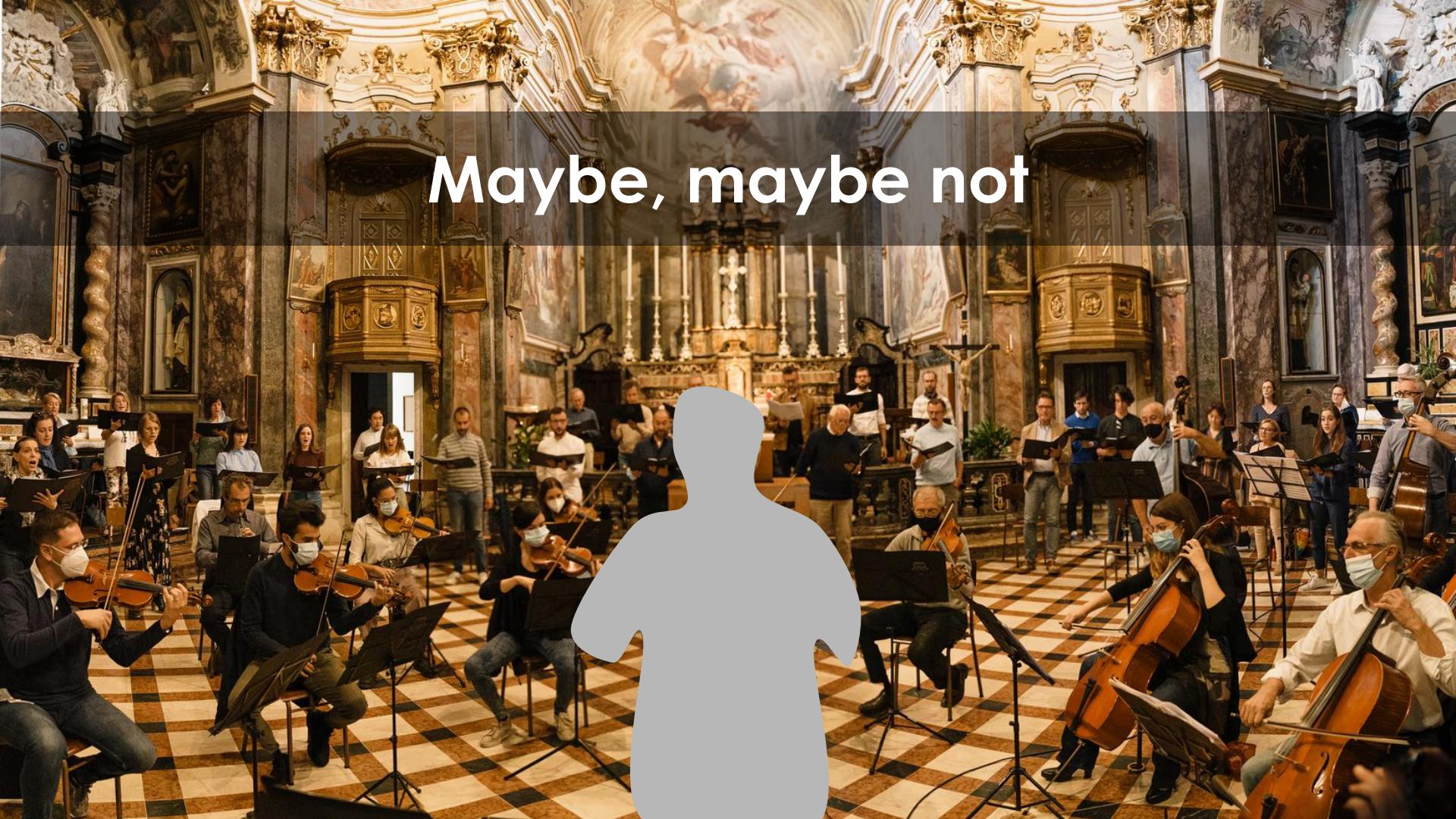
But you have no conductor :(

A photograph of a large-scale musical performance taking place inside a grand, ornate church. The space is filled with musicians, including string players, brass players, woodwind players, and a large choir, all dressed in casual contemporary clothing. They are positioned on a checkered floor, surrounded by the rich, gold-leafed architecture of the church's interior. The ceiling is painted with a dramatic scene, and various altars and statues are visible in the background. A large, semi-transparent white silhouette of a person's head and shoulders is centered in the foreground, looking towards the stage.

Will the orchestra itself be able to play?



Maybe, maybe not



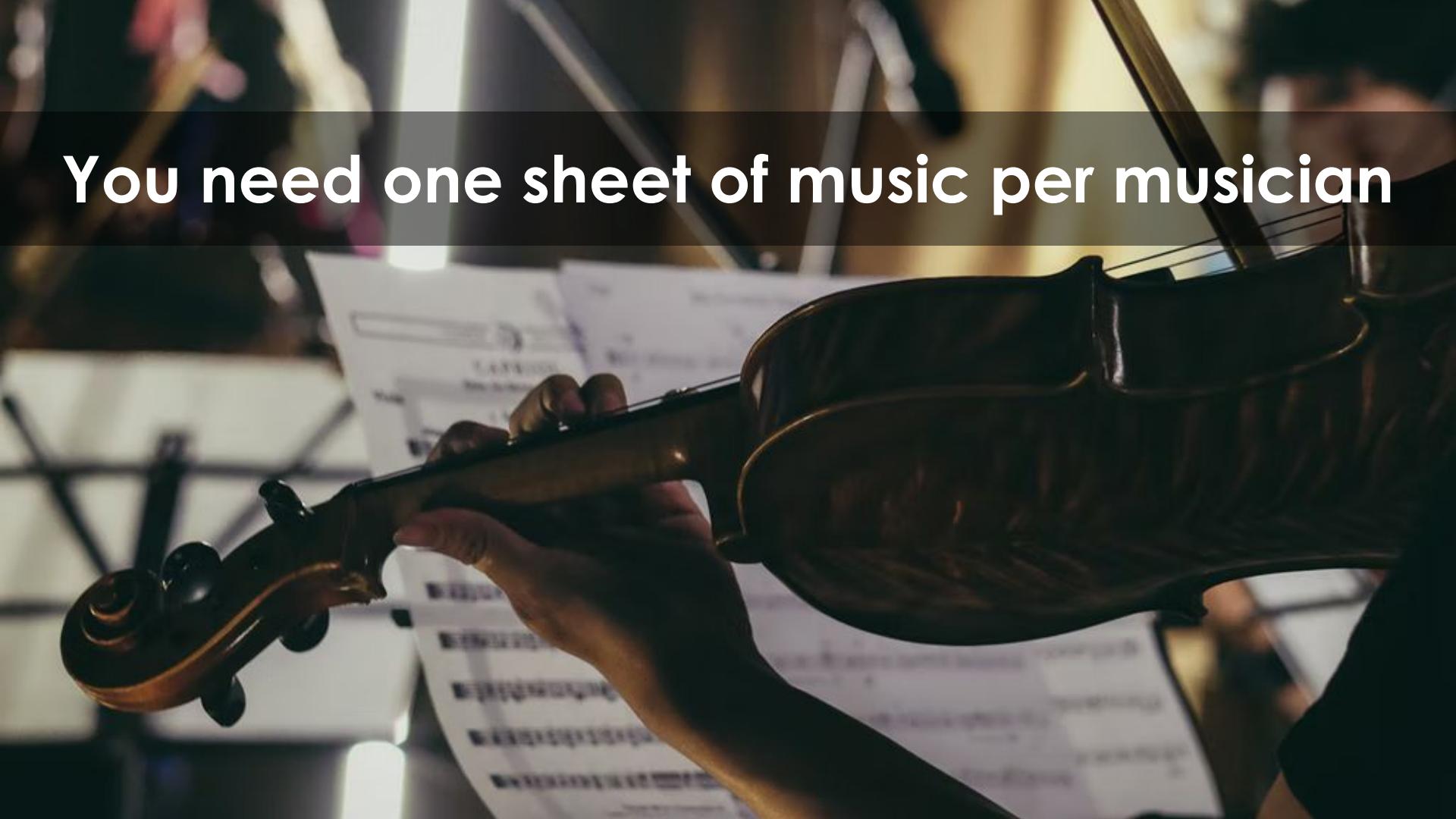
A photograph of a musical performance in a Baroque-style church. The interior is highly decorated with gold leaf, intricate carvings, and frescoes. A large organ is visible on the left, and a crucifix hangs from the ceiling. The floor is made of large, light-colored tiles. In the foreground, a conductor in a white shirt and dark trousers stands facing the musicians. The orchestra consists of violinists, cellists, and a double bass player. The choir, wearing dark clothing and face masks, is positioned behind the musicians, singing into microphones. Many people are standing in the background, some holding cameras. The overall atmosphere is one of a formal concert.

You need a conductor and good musicians

The 4 Whys:

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





You need one sheet of music per musician

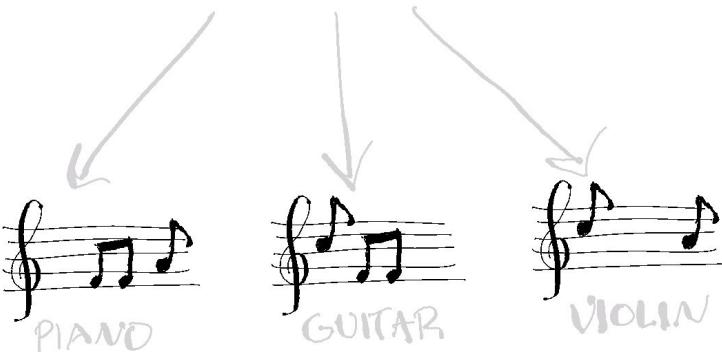
From composer



DECLARATIVE CLAIM



CONDUCTOR



IMPERATIVE ARRANGEMENT



Declarative vs Imperative

“declarative” = “making a declaration”

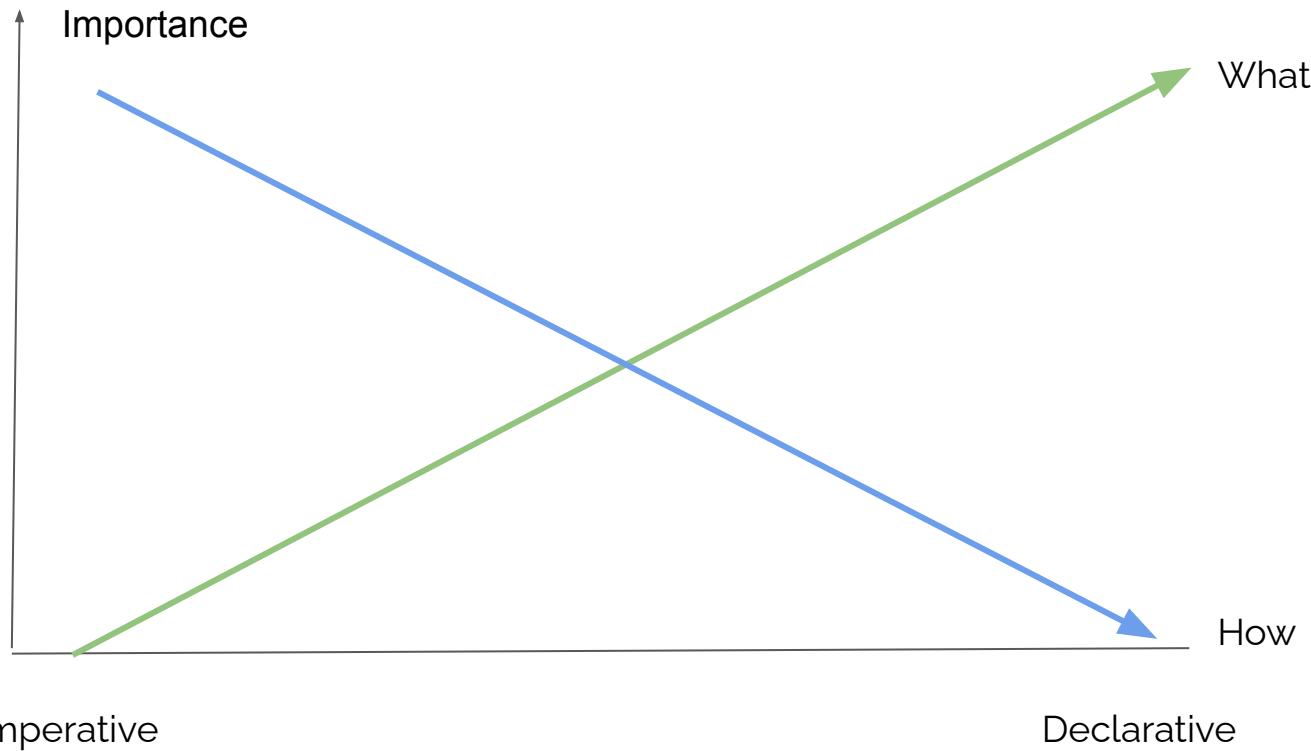
“declare” = “to make known as a determination” (Merriam-Webster, 2021)

Declarative: "The door is shut" (state)

Imperative: "Shut the door!" (action)



The Imperative - Declarative spectrum



Imperative – Declarative examples

C – Imperative

Haskell – Declarative

PNG – Imperative

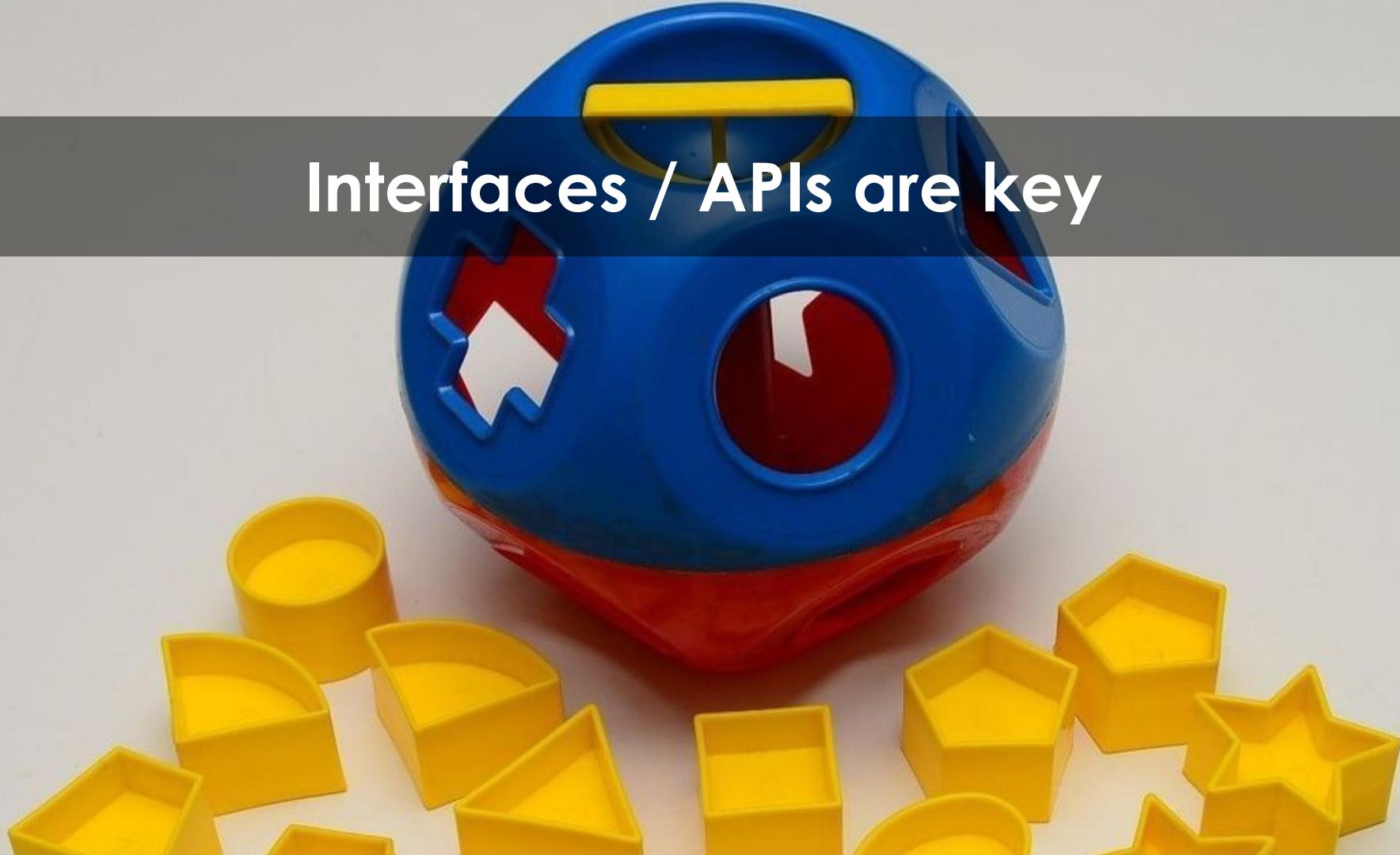
SVG – Declarative

Manually storing data in files – Imperative

SQL – Declarative



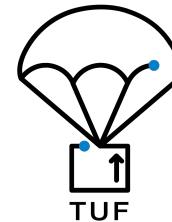
Interfaces / APIs are key



Abstraction Layers: Pluggable interfaces

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

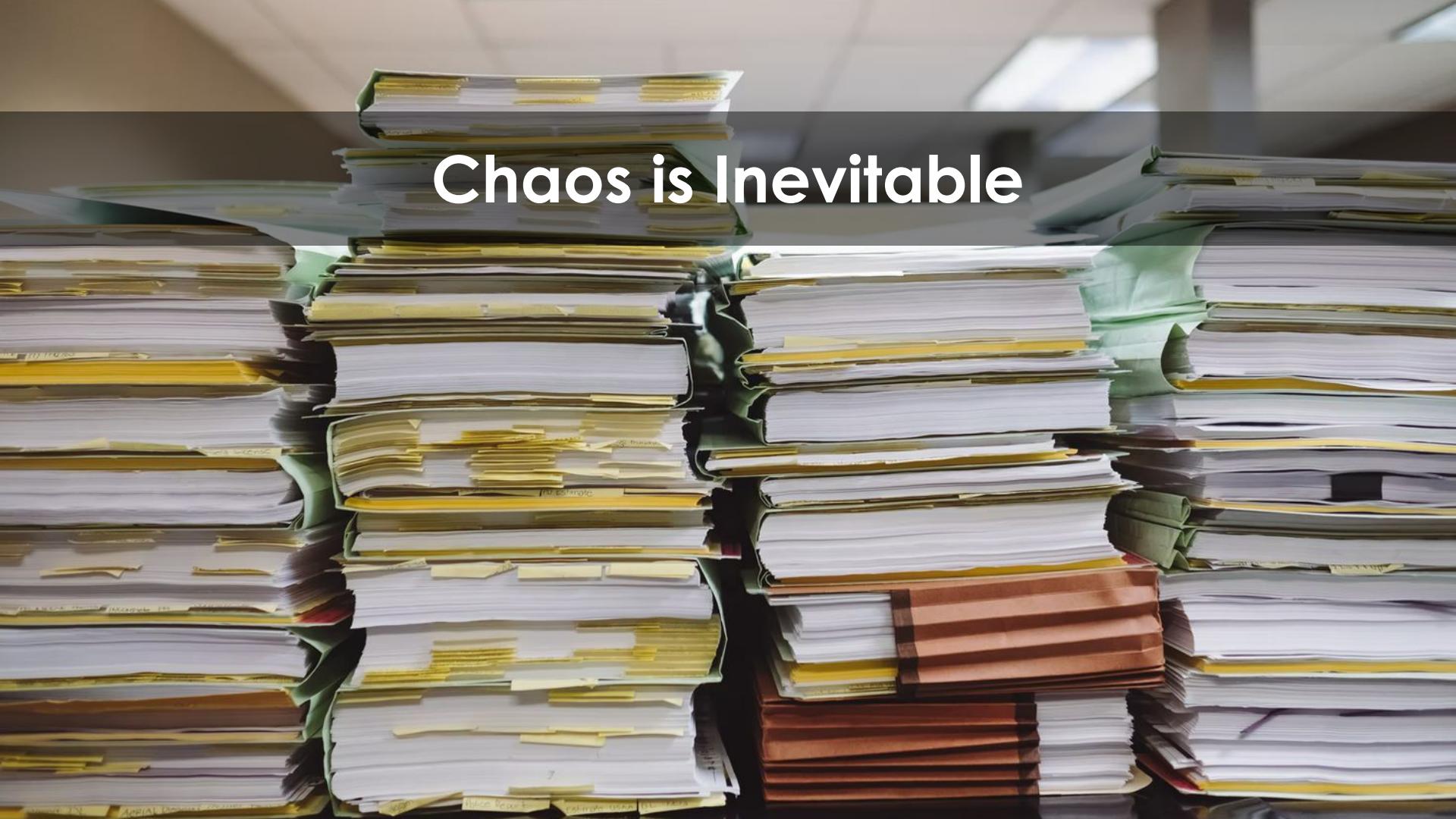
These CNCF/LF projects contain only a specification, no implementation:



The 4 Whys:

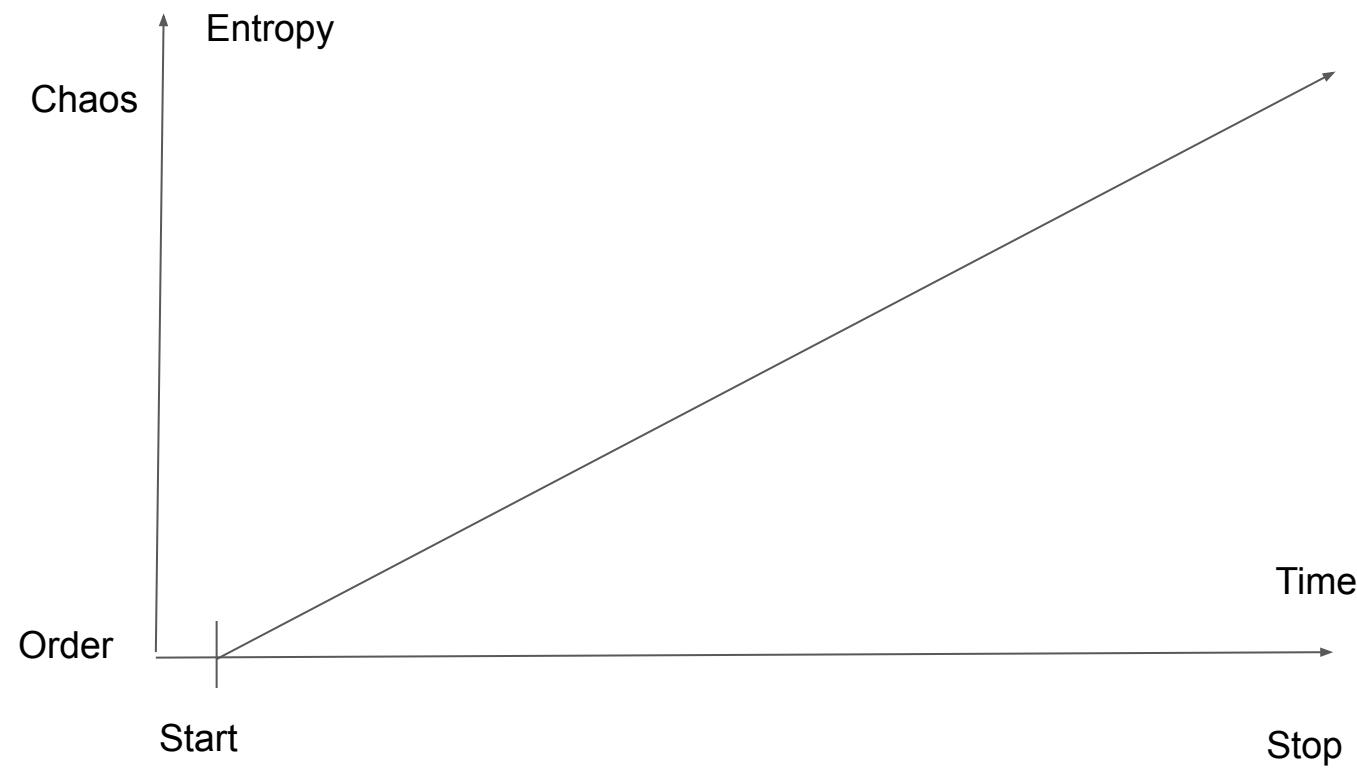
1. The control plane is for Coordination, yet allows Improv
2. **Declarativeness is for Portability and Desired State**



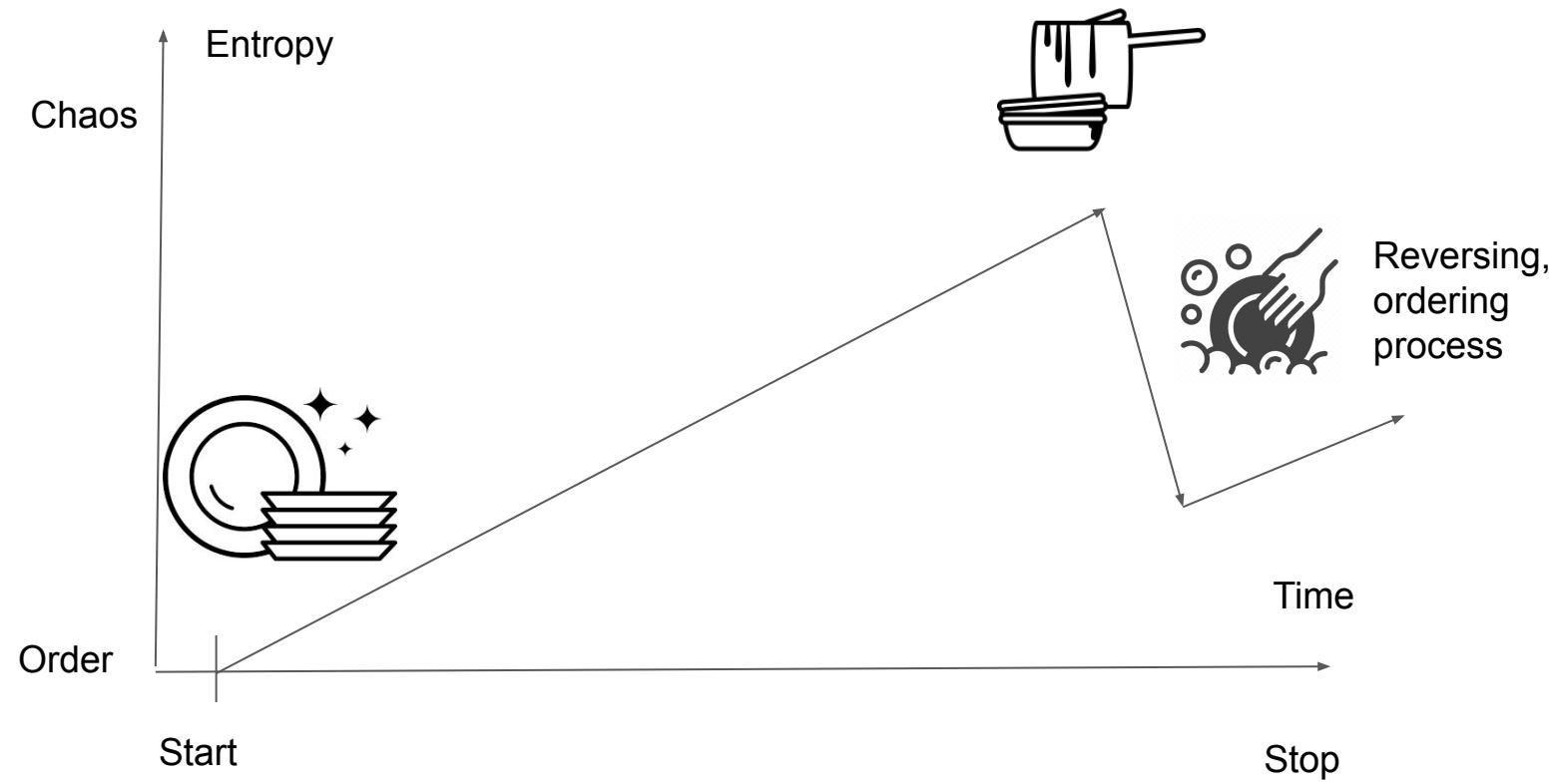
A large, sprawling stack of papers and files, symbolizing chaos and overwhelming workload. The stack is composed of numerous documents, some with yellow file tabs, others with green or blue clips. The papers are tightly packed, creating a sense of disorder and volume.

Chaos is Inevitable

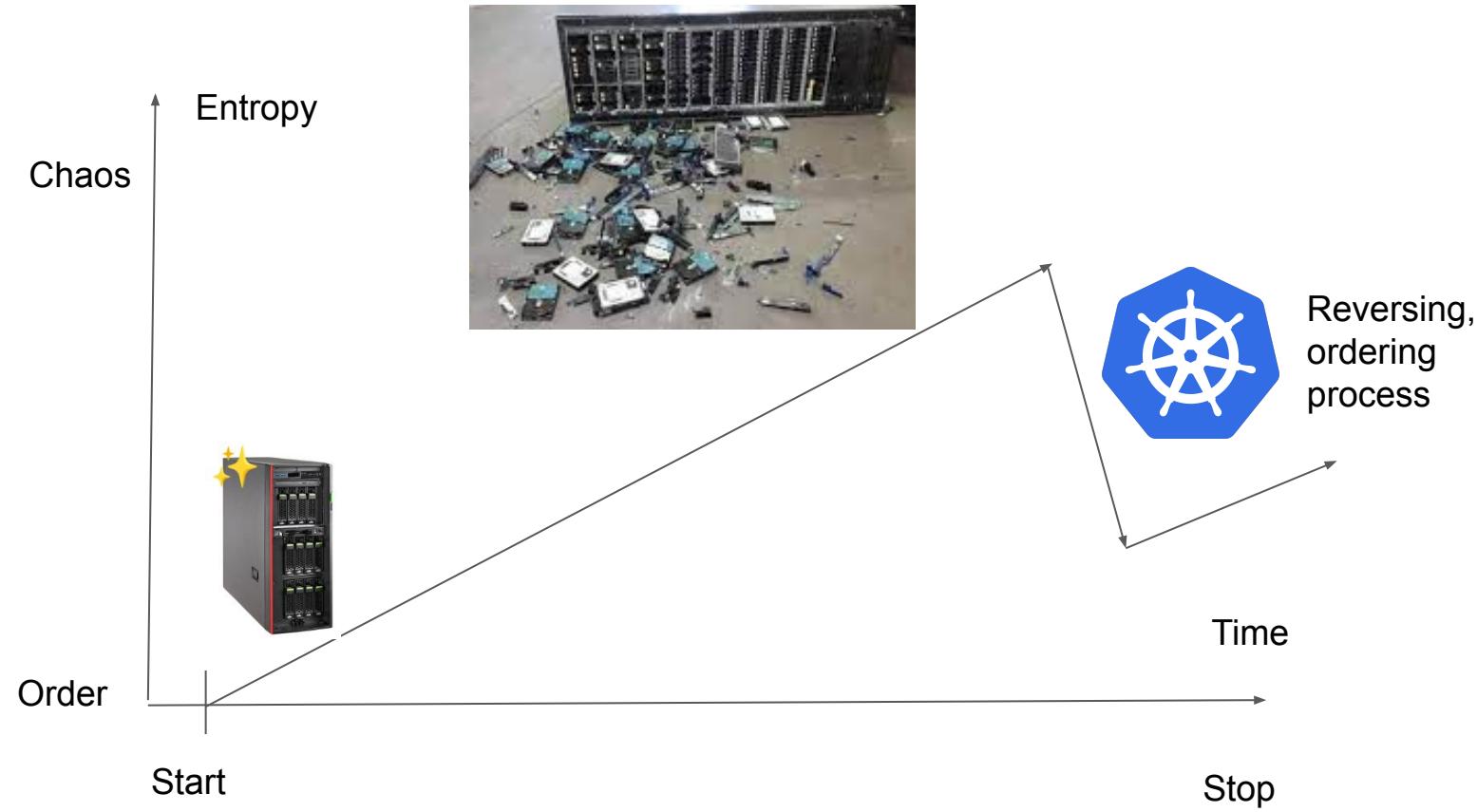
Entropy: Systems become less ordered



Entropy: Putting order to chaos



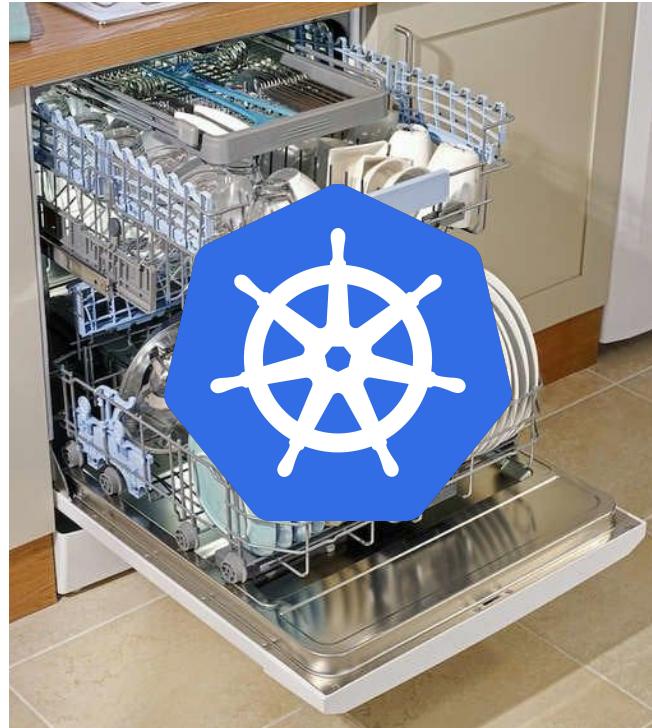
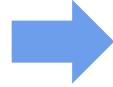
Kubernetes: The dishwasher of servers



Kubernetes: The dishwasher of servers



Kubernetes: The dishwasher of servers

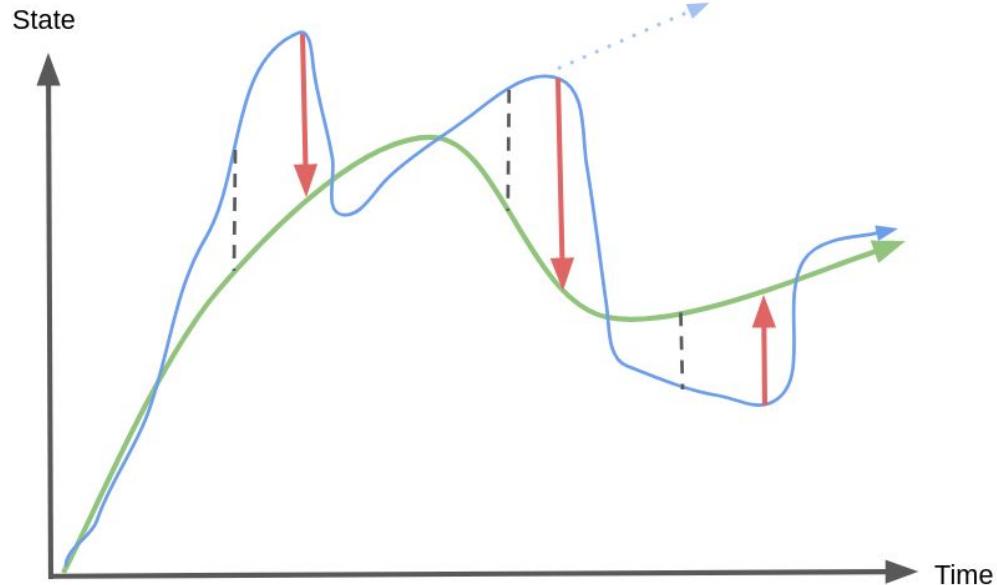




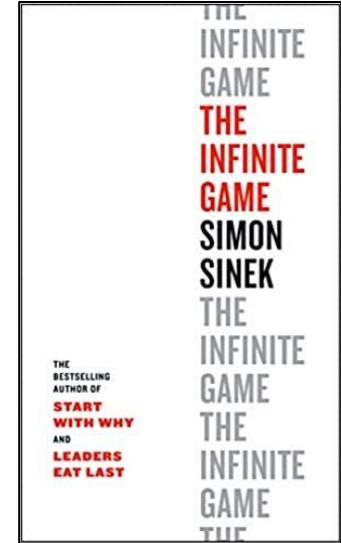
Hello, IT, have you tried
turning it off and on again?



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.

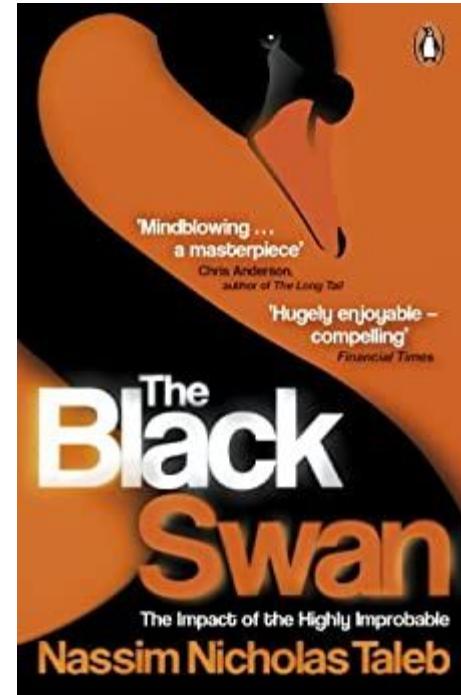
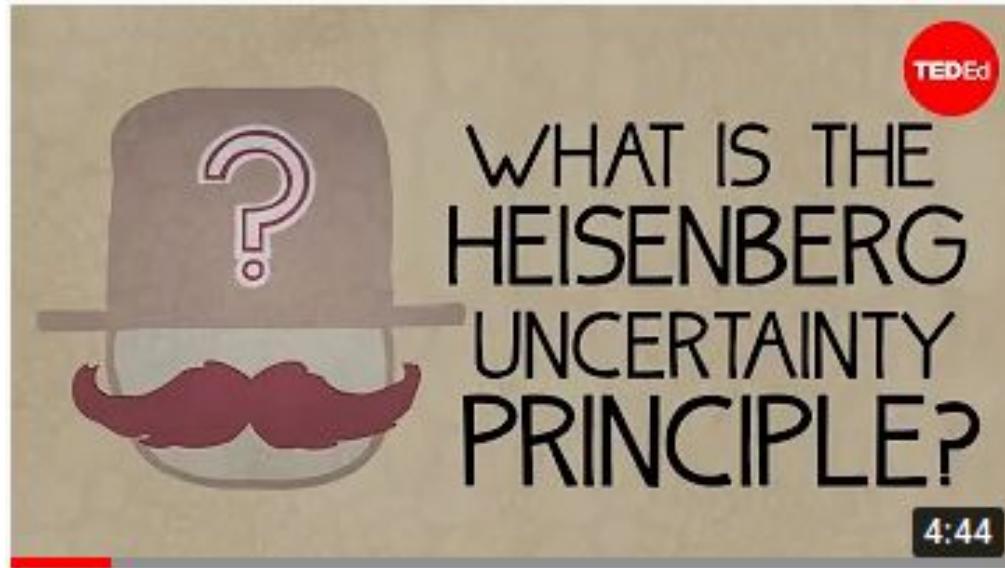


The 4 Whys:

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



A photograph showing several pieces of broken ceramic plates scattered across a light-colored, textured surface. The plates are primarily blue and white, with some pieces appearing to be from a single plate that has shattered into many fragments. The lighting creates shadows and highlights on the shards.

Google Finding: “Failure is the Norm”

Google Finding: “Failure is the Norm”

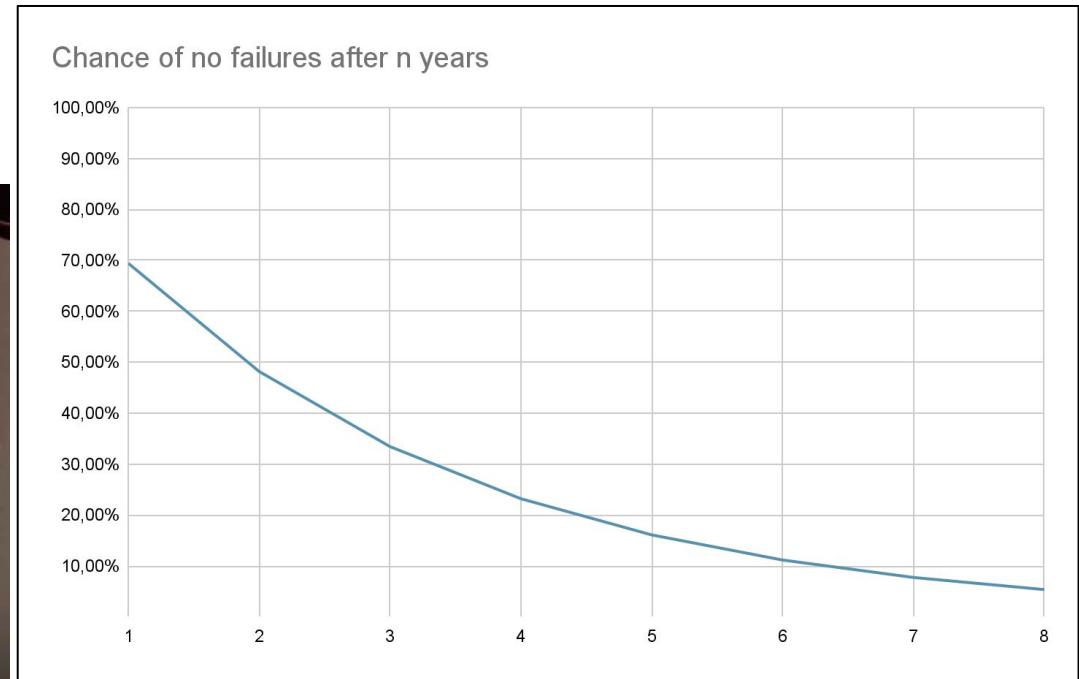
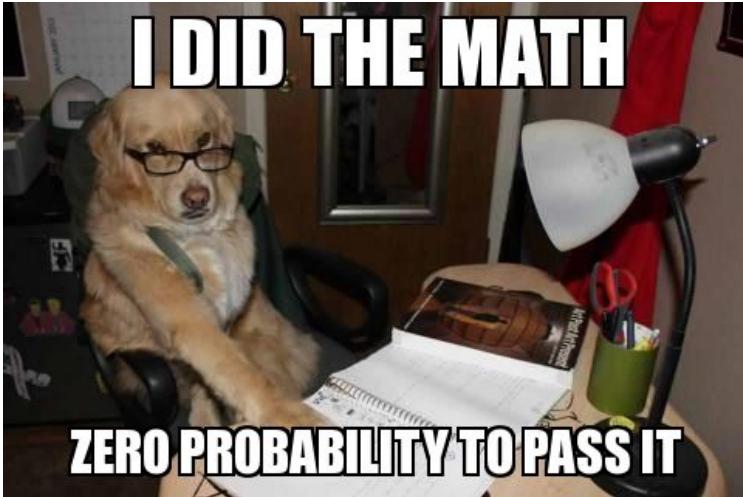
“deliberately leave significant headroom for workload growth, occasional ‘black swan’ events, load spikes, machine failures, hardware upgrades, and large-scale partial failures (e.g., a power supply bus duct)”

Randomness is Unintuitive

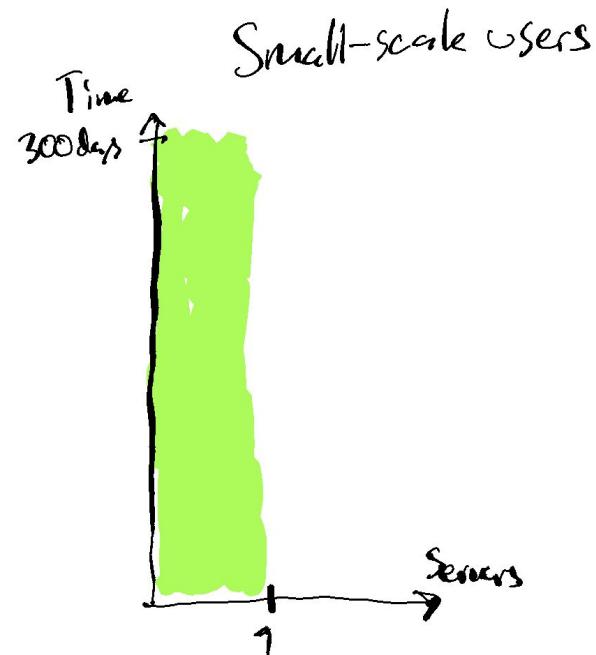
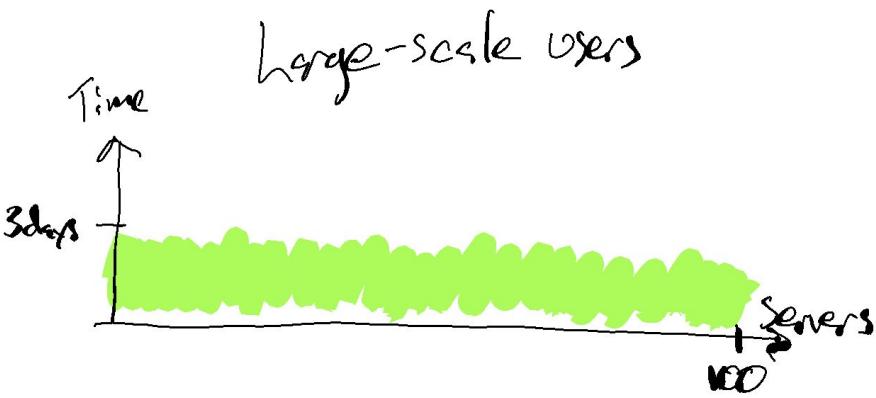
A procedure fails only once in 10000 runs (estimate)

=> 99.99% success probability

Runs 10 times a day



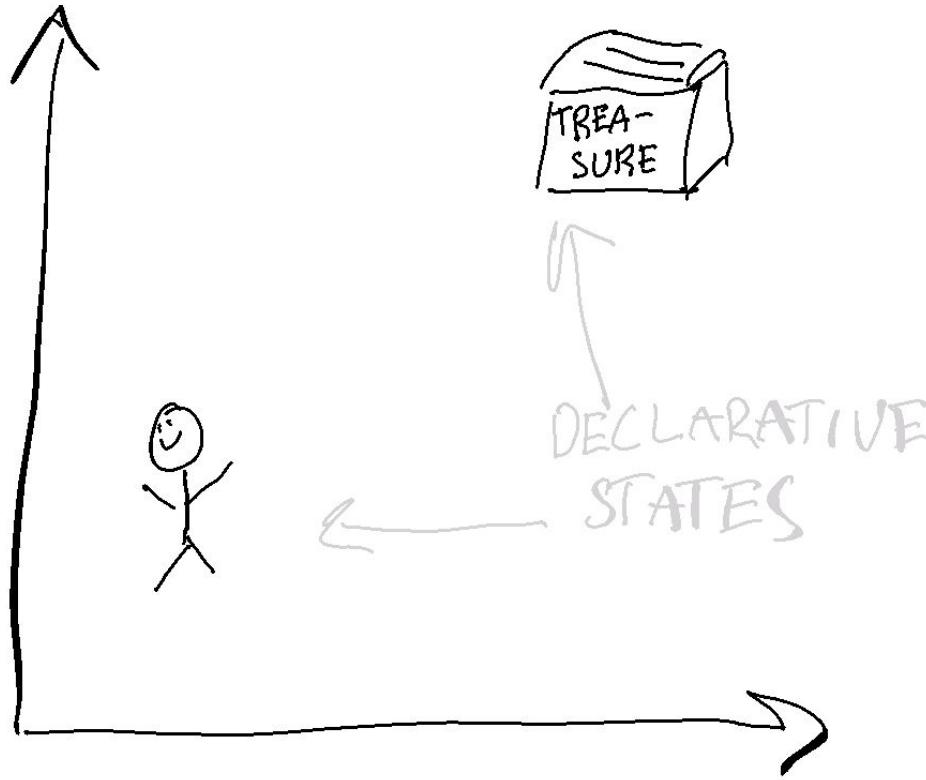
Problems hiding in plain sight

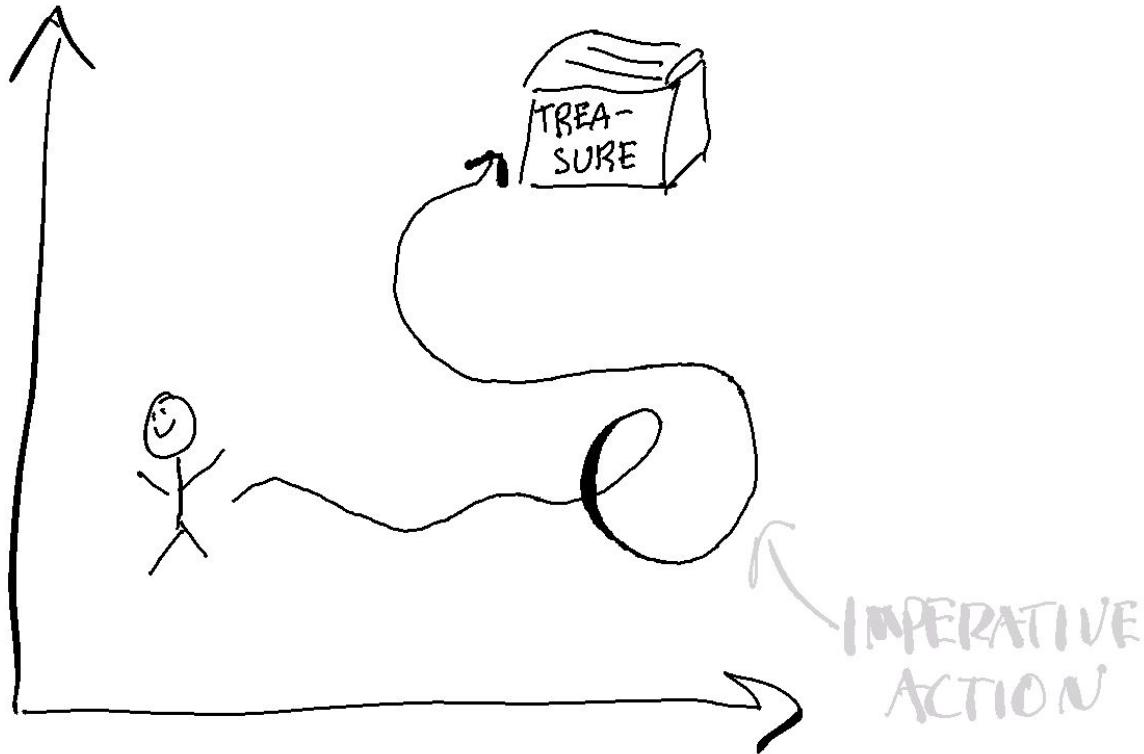


It just takes longer for small-scale users to notice problems due to e.g. randomness

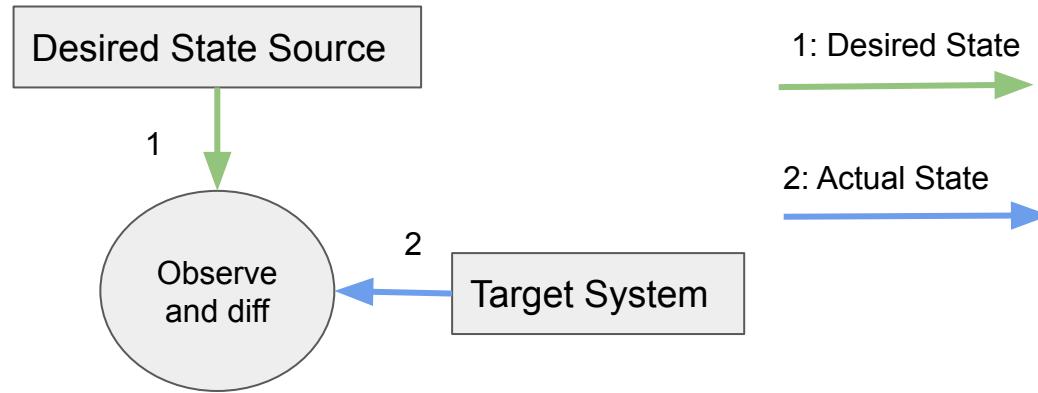
Calling a Taxi is Declarative

A photograph of two taxis at night. The taxi in the foreground is dark-colored with its yellow 'TAXI' sign illuminated. Another taxi is visible behind it, also with its sign lit up. The background is a blurred cityscape with numerous colorful lights from buildings and street signs, creating a bokeh effect.

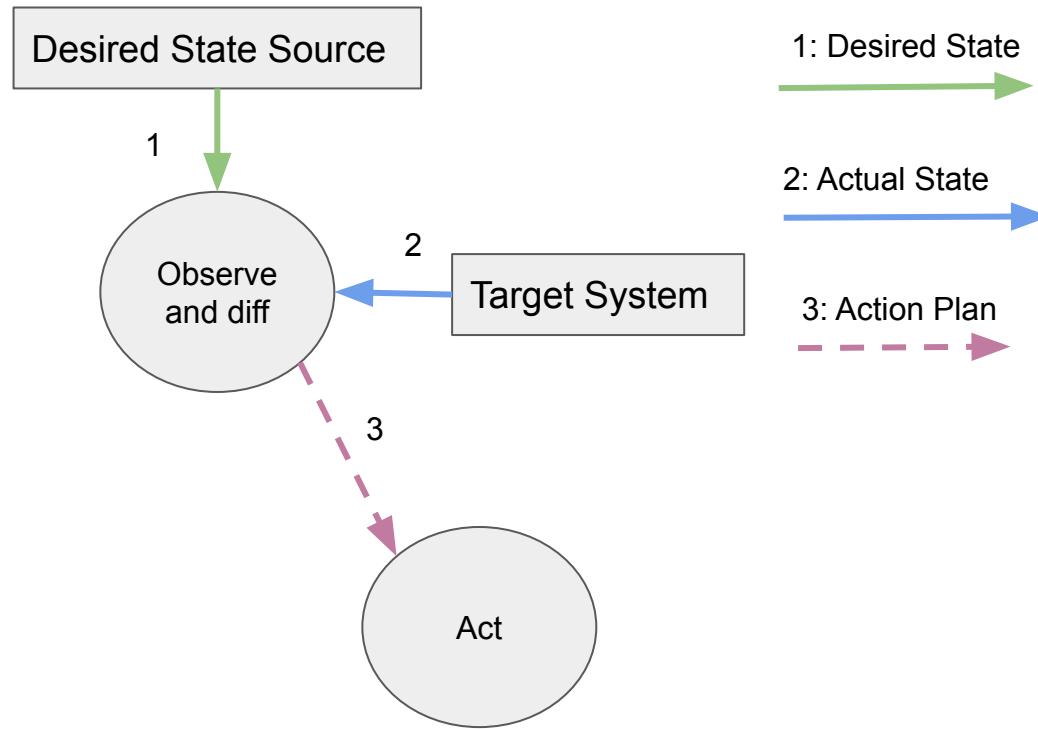




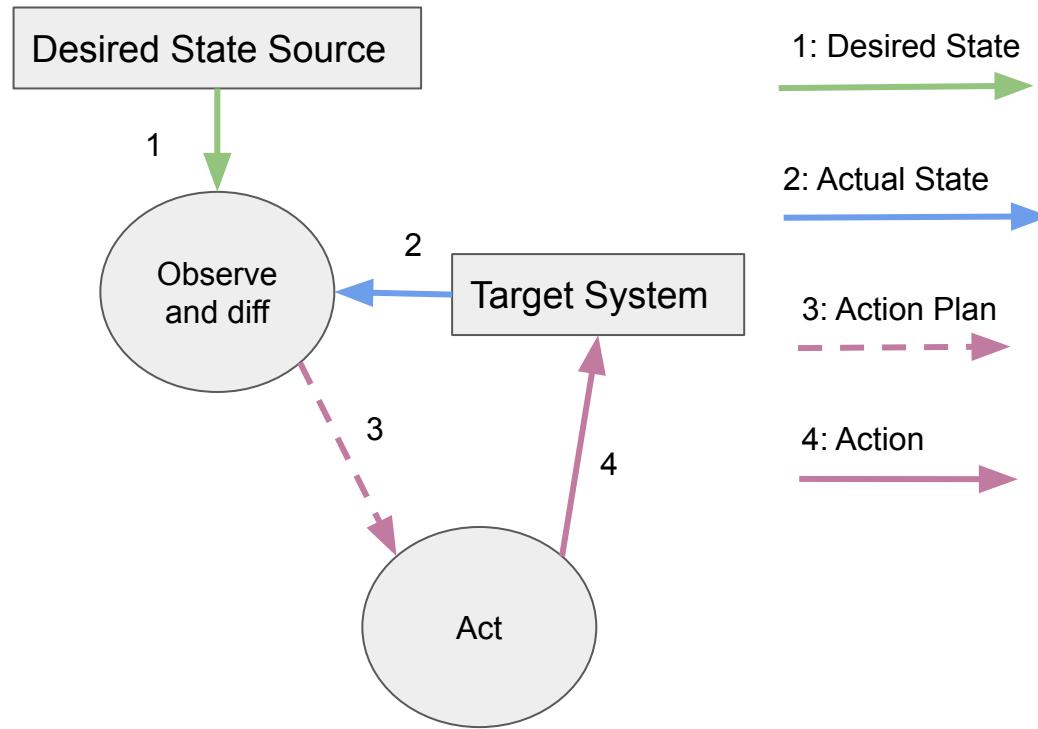
Actuators, or reconcile loops, fulfil the claim(s)



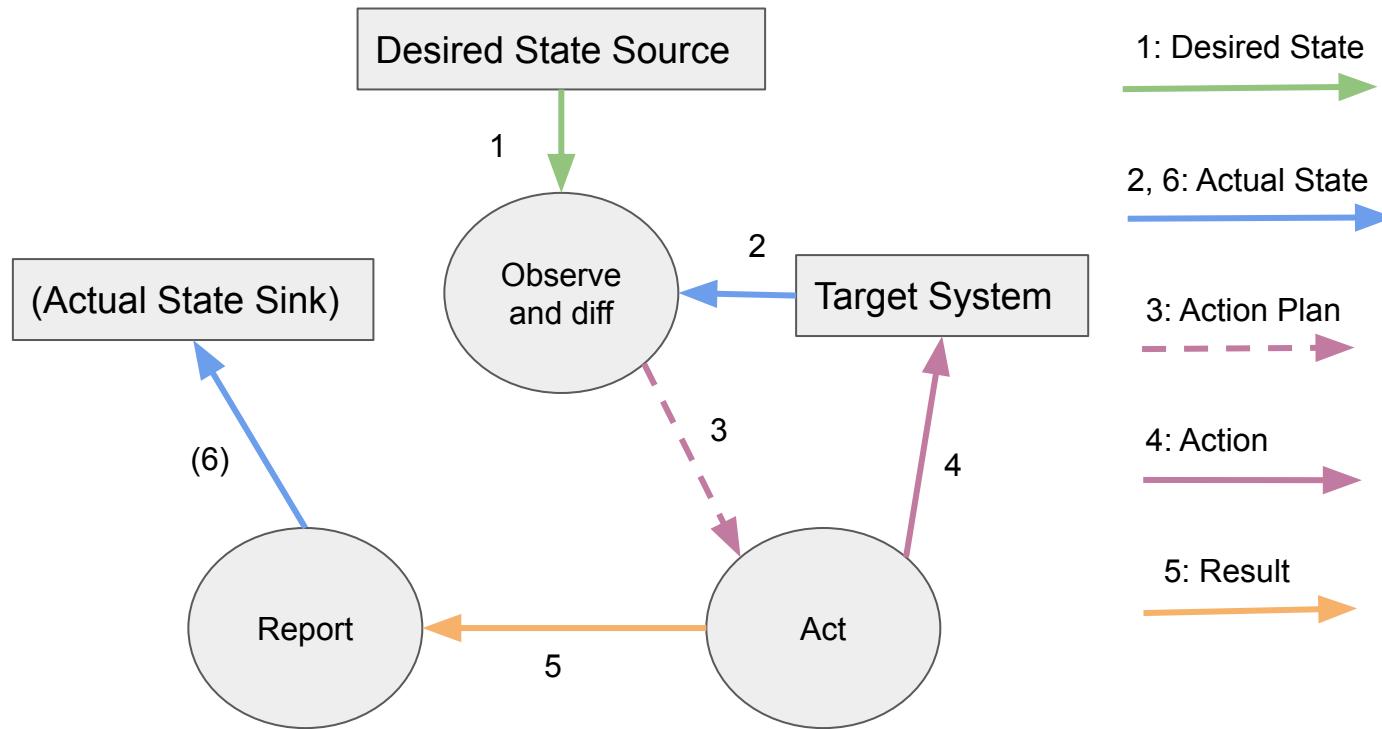
Actuators, or reconcile loops, fulfil the claim(s)



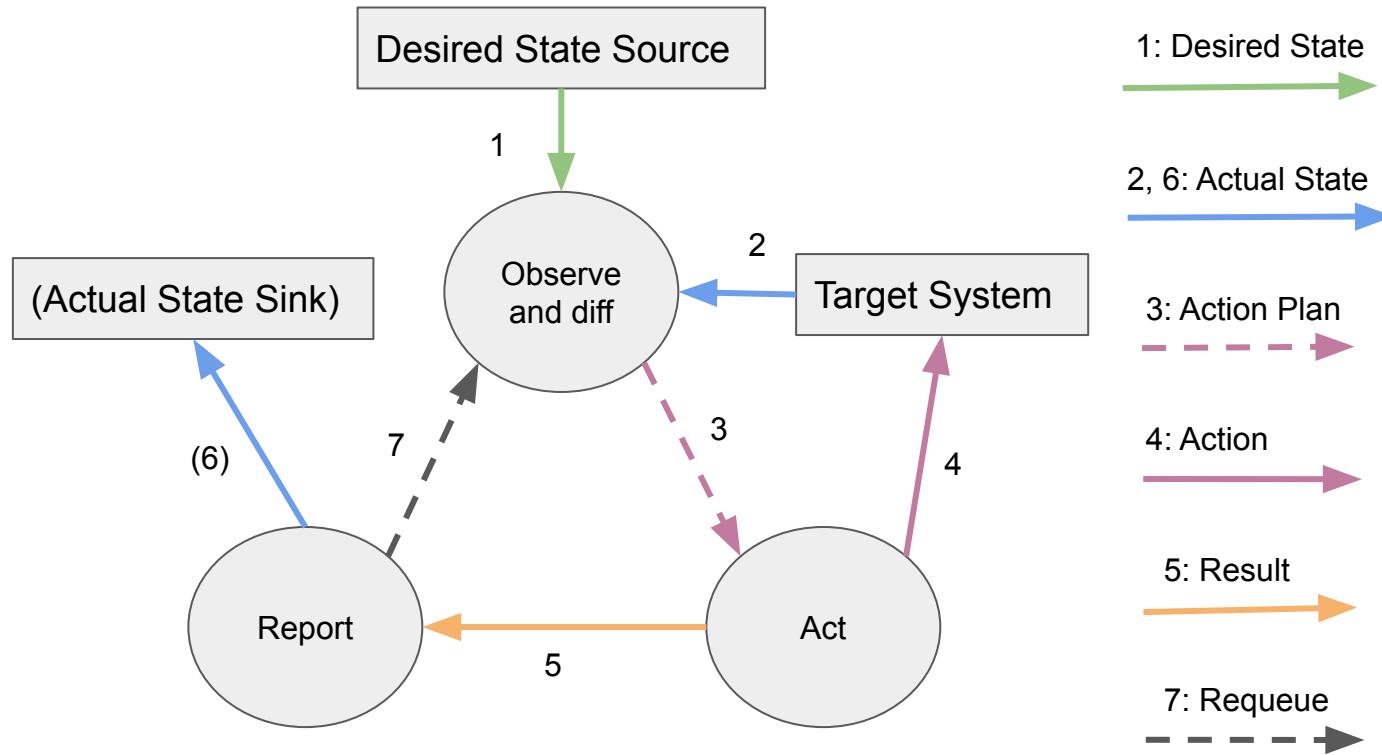
Actuators, or reconcile loops, fulfil the claim(s)



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.

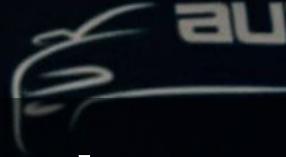
1. Ask where P wants to go, e.g. the City “desired state”
2. Ask where P is, e.g. Fira Valencia “actual state”
3. Figure out a route plan “action plan”
4. Drive P from Fira Valencia to the City “action”
5. Was the drive successful? “result”
6. Photograph P for a social media update “actual state sink”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



The 4 Whys:

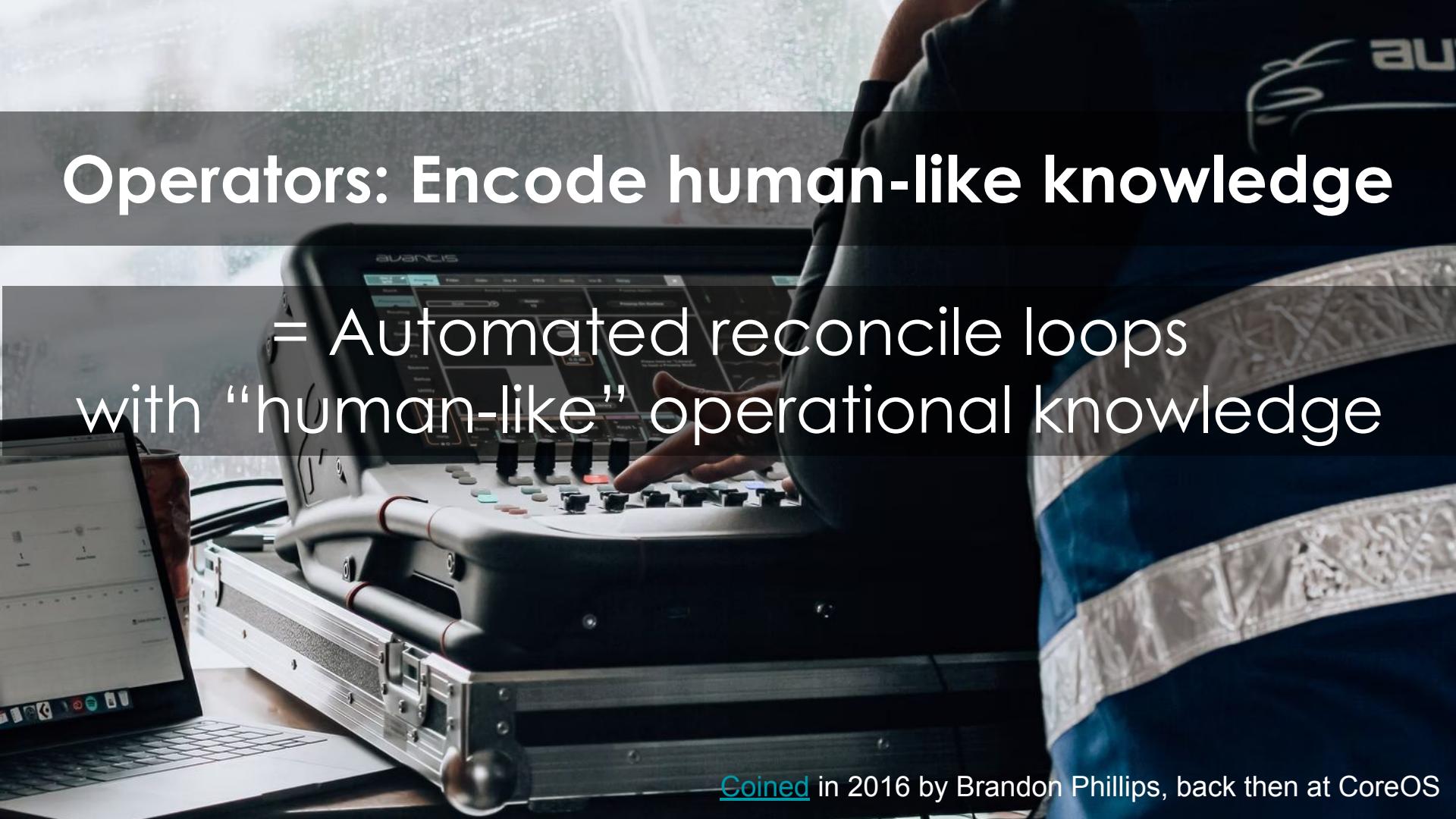
1. The control plane is for Coordination, yet allows Improv
2. Declarativeness is for Portability and Desired State
3. Periodic action is for fighting inevitable Chaos
4. **Designing controllers for failure is for Randomness**





Operators: Encode human-like knowledge

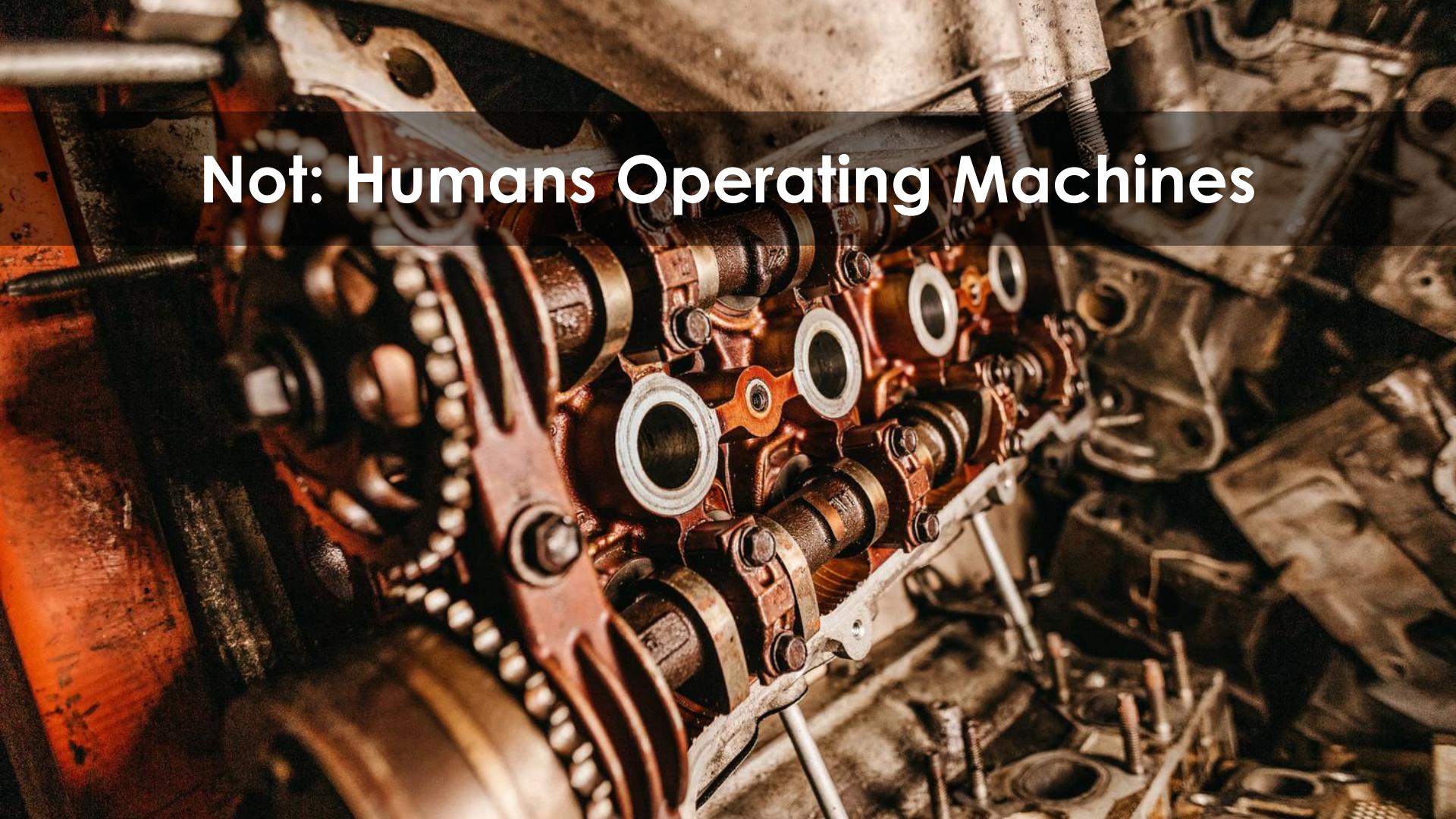




Operators: Encode human-like knowledge

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

[Coined](#) in 2016 by Brandon Phillips, back then at CoreOS

A close-up, low-angle shot of a mechanical engine's internal components. The image shows several large, reddish-brown gears and shafts, some with white protective covers. The lighting is dramatic, highlighting the metallic textures and the complex arrangement of parts. The background is dark and out of focus.

Not: Humans Operating Machines

Instead: Humans Operating Automation that in turn Operate Machines



Further Reading



CHAPTER ONE

At this, Winky howled even louder, her tortured voice a noise dribbling all down her front, though she didn't seem to be crying.

"Dobby has travelled the country for two whole years, trying to find work!" Dobby sputtered. "But Dobby hasn't found work, because Dobby was just too..."

The house-elves all around the kitchen, who had been listening and watching with intense, ill-looked eyes at these words, one, however, cried, "Good for you, Dobby!"

"Thank you, miss," said Dobby, grinning weakly at her. "But most wizards don't want a house-elf who likes paying wages. That's not the point of a house-elf, they say, and they started the door in Dobby's face! Dobby liked work, but he wants to wear clothes and he wants to be paid, Harry Potter.... Dobby likes being free!"

The Hogwarts house-elves had now started edging ever further away, as though he were carrying something contagious. Winky, however, remained where she was, though there was a definite increase in the volume of her crying.

"And then Harry Potter, Dobby goes to visit Winky, and finds out, Winky has been fired too, sir!" said Dobby delightedly.

At this, Winky flung herself forward off her stool and lay face down on the flagged stone floor, letting her tiny fist ripen it and positively ramming it into Harry. Hermione hardly dared describe her knees beside her and tried to comfort her, but nothing had made the slightest difference. Dobby continued with his now-shouting shrilly over Winky's sobs.

"And then Dobby had the idea, Harry Potter, sir? 'Why doesn't

THE EDUCATIONAL ILLUSTRATION FRONT

Dobby and Winky find work together?" Dobby says. "Where is there enough work for two house-elves?" says Winky. And Dobby looks out of a window at home, out. *Hermione* says Dobby and Dobby come to see Professor Dumbledore, sir, and Professor Dumbledore, and so on.

Dobby beamed very brightly, and happy tears welled in his eyes.

"And Professor Dumbledore says he will pay Dobby, sir, if Dobby wants payin'! And as Dobby is a free elf, sir, and Dobby gives a Galleon a week and one day off a month?"

"This is not very much," Hermione declared indignantly from the back of the common room, scowling and frowning.

"Professor Dumbledore offered Dobby ten Galleons a week, and

wouldn't off, and Dobby suddenly giving a little shout as though

the prospect of so much leisure and rest were frightening. "but

Dobby beat him down, miss... Dobby likes freedom, miss, but

he isn't wanting too much, miss; he likes work better."

"And how much is Professor Dumbledore paying now, Winky?"

Hermione asked kindly.

If she had thought this would cheer up Winky, she was wildly mistaken. Winky did stop crying, but when she sat up she was glaring at Hermione through her watery brown eyes, her whole face upping wet and suddenly furious.

"Winky is a disgraced elf, but Winky is not yet getting paid!"

she squealed. "Winky is not stuck so low as that! Winky is properly

ashamed of being fired!"

"Ahaaaa!" said Hermione blankly. "But... Winky, come out!

"Mr. Crouch who should be ashamed, not you! You didn't do

anything wrong, he was really horrible to you..."

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

**Encoding human-like operational
knowledge using declarative
Kubernetes operator patterns**

Lucas Käldström

School of Electrical Engineering

Bachelor's thesis
Espoo 10.12.2021

Supervisor

Assistant Professor Martin Andraud

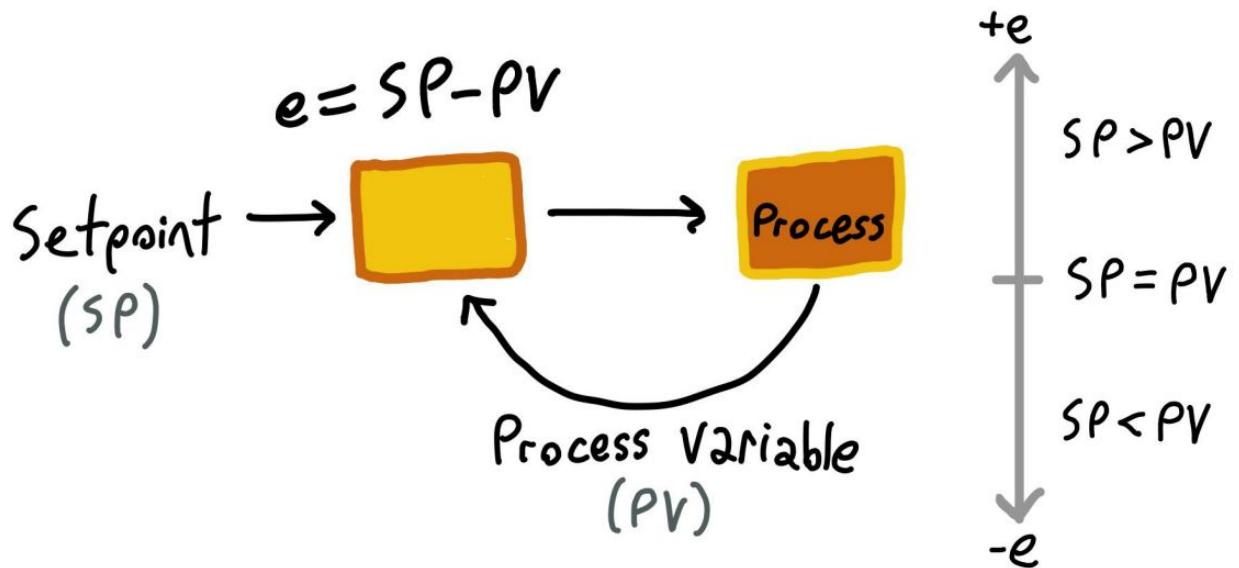
Advisors

Prof. Mario Di Francesco

Dr. Stefan Schimanski



Control Theory

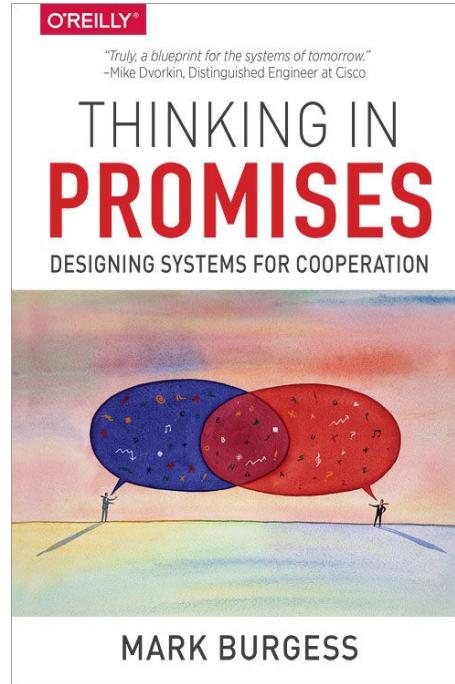


Promise Theory



So Kubernetes is built on this thing we call 'promise theory'. Even though

(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

luxas@luxaslabs.com

