4 Reliability Anti-Patterns

ConFoo.CA 2024

DEVELOPER CONFERENCE

“*The future belongs to those who believe in ~~the beauty of their dreams~~ reliability*”
- Eleanor Roosevelt (revisited)

Air Traffic Management

Software engineer in a
safety critical domain

Ride-Hailing, Docker, etc.

Software engineering
and reliability advocacy

Google

SRE

(Site Reliability Engineer)

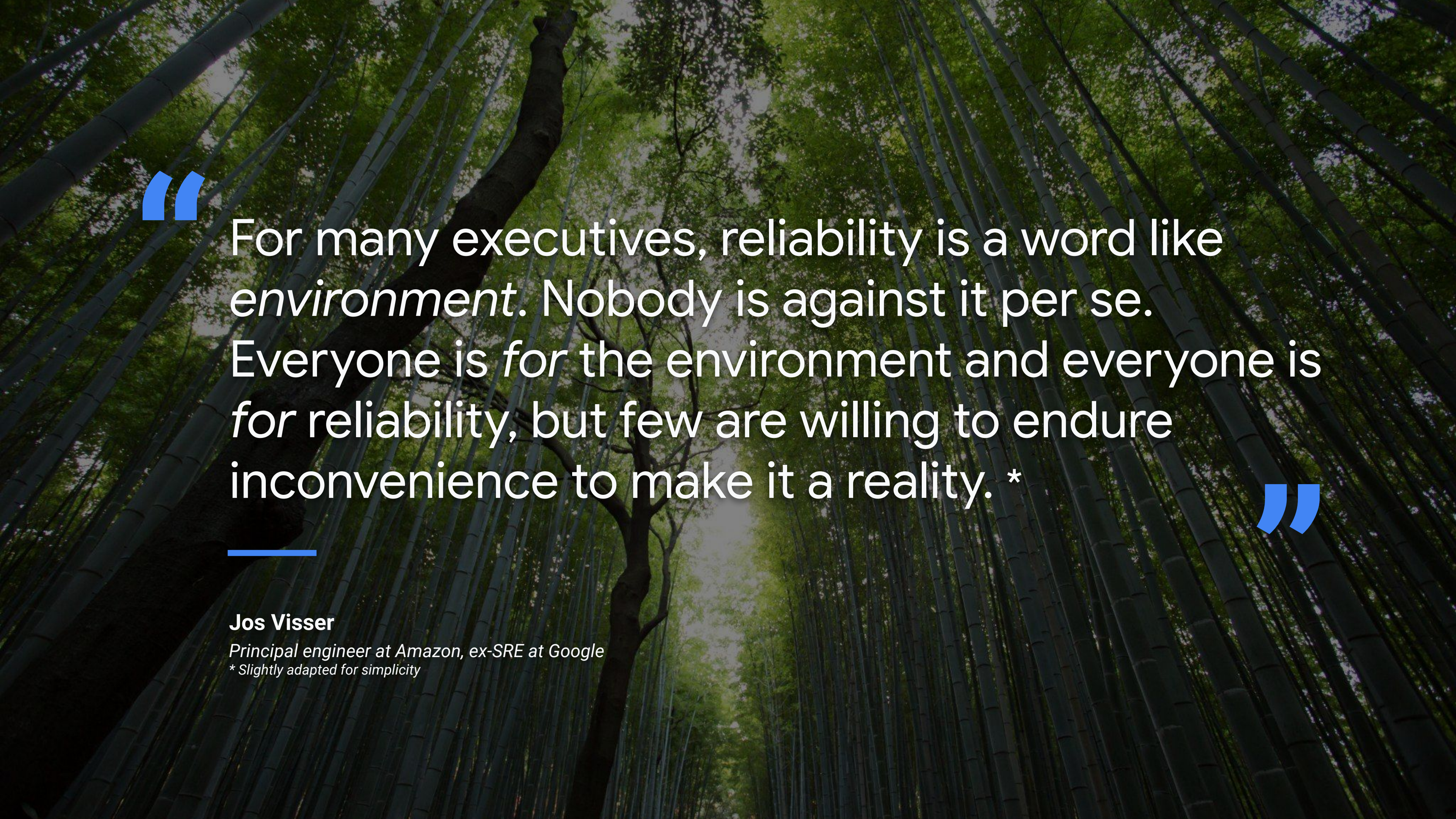
#1

Reliability Procrastination Culture

A culture of verbal acknowledgement and inaction

Reliability Procrastination Culture

A cultural mindset where organizations **avoid embracing reliability**, often due to the perceived inconvenience it may bring



> “For many executives, reliability is a word like *environment*. Nobody is against it per se. Everyone is *for* the environment and everyone is *for* reliability, but few are willing to endure inconvenience to make it a reality. *”

Jos Visser

Principal engineer at Amazon, ex-SRE at Google

** Slightly adapted for simplicity*

Some Characteristics

Reactive culture

Reactive over proactive

Short-term vision

Quick fixes over long-term solutions

Blame game

Blame over collaboration

Solutions

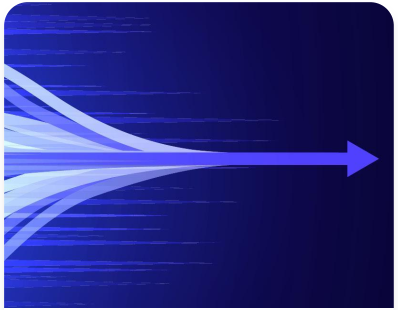
Cultural shift

- Blameless culture
- Post-mortems
- No hero
- SLOs

Solutions

SRE Culture

Dev / Ops split



DevOps

SRE

SRE: People focused on **reliability** challenges

Reliability Procrastination Culture

1.8x

Teams that excel at reliability engineering are **1.8x** more likely to meet or exceed organizational goals

Source: 2022 State of DevOps

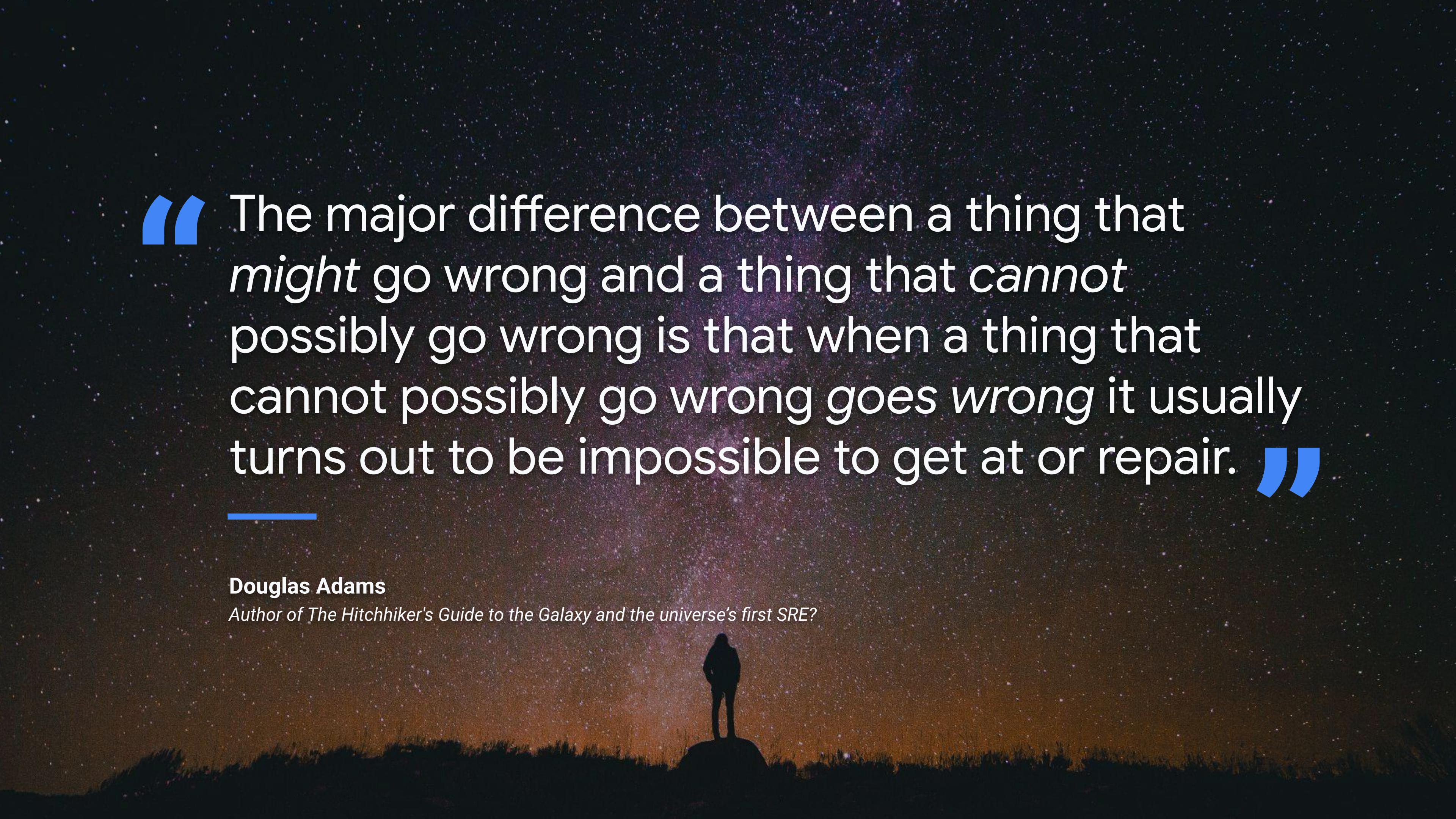
#2
Failure Denial Syndrome

A reluctance to embrace failures

Failure Denial Syndrome

A mindset that avoids or denies the **inevitability of failures** in complex systems

Story Time



> “The major difference between a thing that *might* go wrong and a thing that *cannot* possibly go wrong is that when a thing that cannot possibly go wrong *goes wrong* it usually turns out to be impossible to get at or repair.”

Douglas Adams

Author of The Hitchhiker's Guide to the Galaxy and the universe’s first SRE?

Why?

Fear of failure

People may worry about the **consequences**

Blame game

People or teams may deny failures to avoid being **blamed**

Not a lack of skills

Lack of reliability **culture**

Solutions

Organizations should treat failures as the norm

The question is **not if** it's gonna fail, but **how** it's gonna fail

> “Don't tell me how it works.
> Tell me how it breaks.”

Solutions

Design for failure

- Cattle > pets
- Crash-only software
- Don’t detect failure, but the absence of success
- Bulkhead pattern
- Graceful degradation 🔍

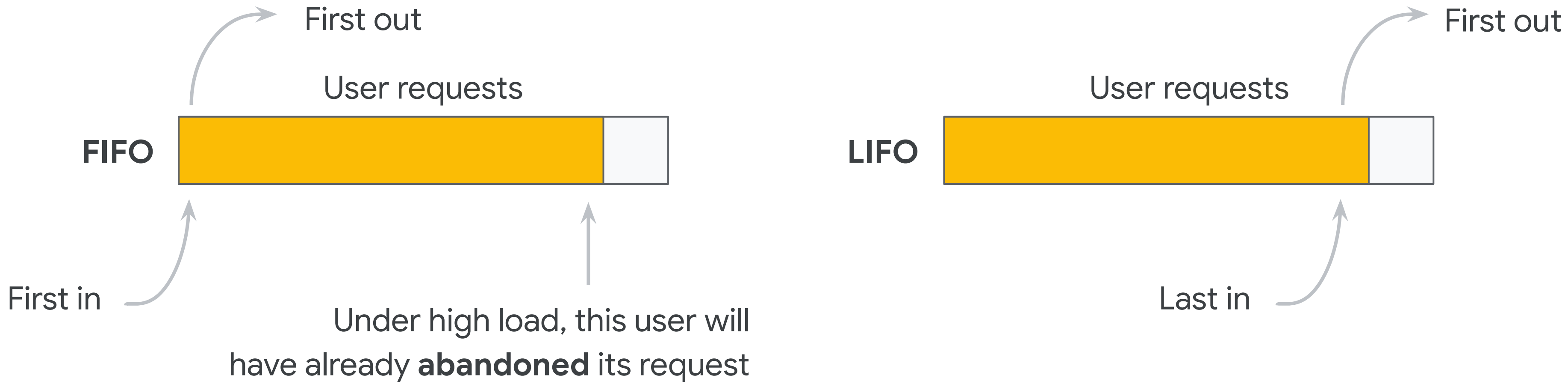
Graceful Degradation

During an unexpected event, an application can **reduce** its quality of service

Example: Load shedding

But not only!

Graceful Degradation: Facebook Adaptative Queue


First out
User requests
FIFO
First in
Under high load, this user will have already **abandoned** its request
First out
User requests
LIFO
Last in

Under normal conditions: FIFO; under heavy load: LIFO

Rationale: giving **some response** back is better than **no response** back

Failure Denial Syndrome

Failures must be the **norm**

Design for failure:

Resilient

Robust

Reliable

#3
Observability Deficiency

When observability becomes a reliability impediment

Observability Deficiency

A situation in which observability **compromises** reliability through inefficiency, blind spots, and confusion

Streetlight Effect

Streetlight Effect

Cognitive bias: when people focus on what is easily visible

Reason why many organisations fall into the "*trap*" of observability

> “Observability has now been mushed and flattened away by million of dollars of marketing spend. Now, if you ask any CIO/CTO about their observability journey, they just look confused:
>
> "*We already have observability. We have logs AND metrics.*"”

Charity Majors

CTO of Honeycomb, co-author of Observability Engineering

Observability Done Wrong

Some negative impacts

Inefficiency

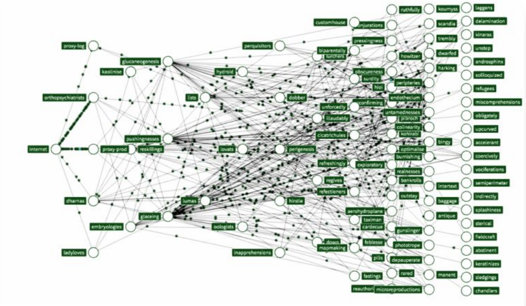
Blind spots

Misleading assessments

👉 **Erode** reliability

Let's Take a Step Back

Why do we need observability?



Complexity

Agility

Observability

First principle

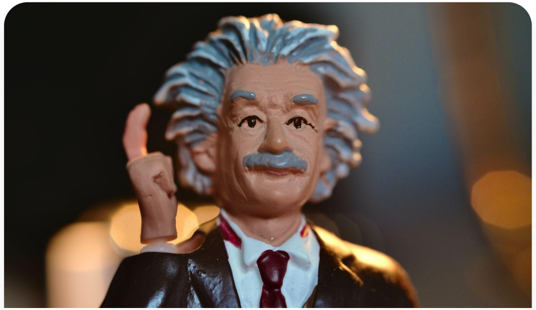
Unknown unknowns

Explorability

You Have Observability If...

You can understand **any state of your system** (no matter how novel or bizarre) by slicing and dicing **high-cardinality and high-dimensionality** telemetry data **without** needing to ship new code

Observability Deficiency



We should understand why we need **observability**

We should **promote** a culture of observability

It should stay a **moving target**

#4

Rollout Roulette

When hope becomes a deployment strategy

Rollout Roulette

The **risky** practice of deploying changes to production without an **efficient and well-defined plan**

Rollout Done Wrong

Negative impacts

Stress

Customer dissatisfaction

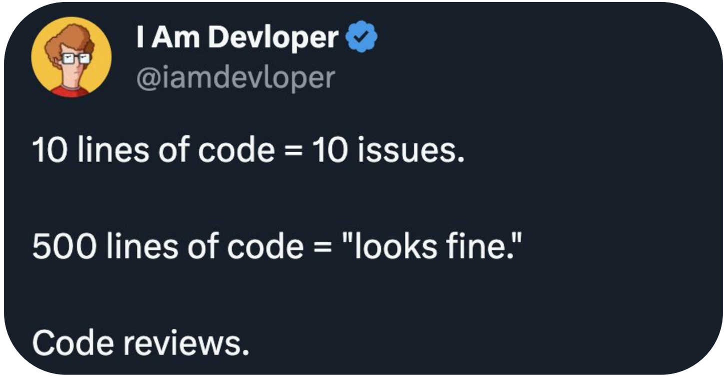
Reputation damage

Solutions

Let’s go over *some* best practices

Frequency

The more frequently we rollout, the **less change** between releases



I Am Devloper
@iamdevloper

10 lines of code = 10 issues.

500 lines of code = "looks fine."

Code reviews.

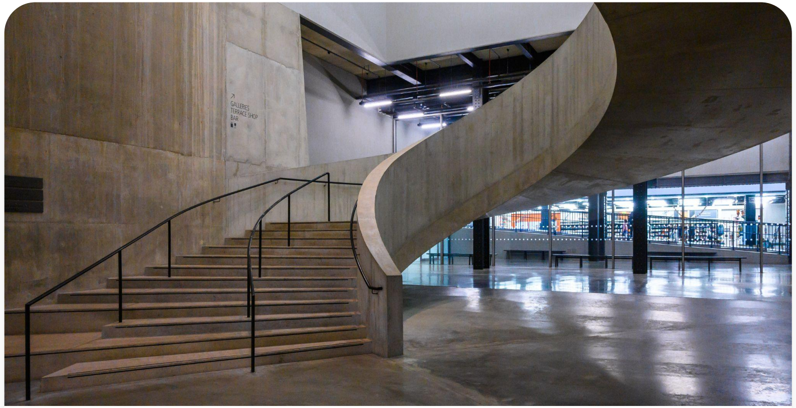
Rollout **even** if there are no changes

Canary vs. Progressive Rollout

Canary rollout

Partial and time-limited

Few production environments



Progressive rollout

Progressively increasing scope

Many production environments

Rollback

Rollout to an earlier version

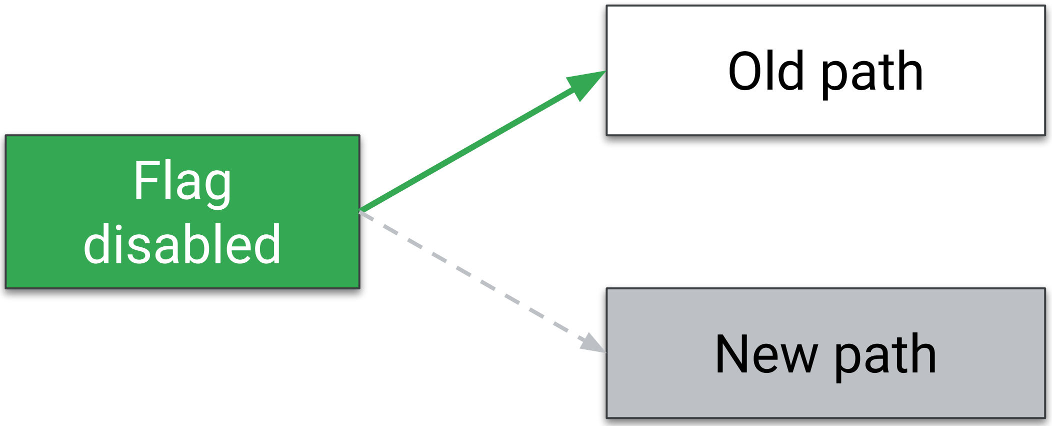
A **crucial** part of a reliable deployment strategy

Tested

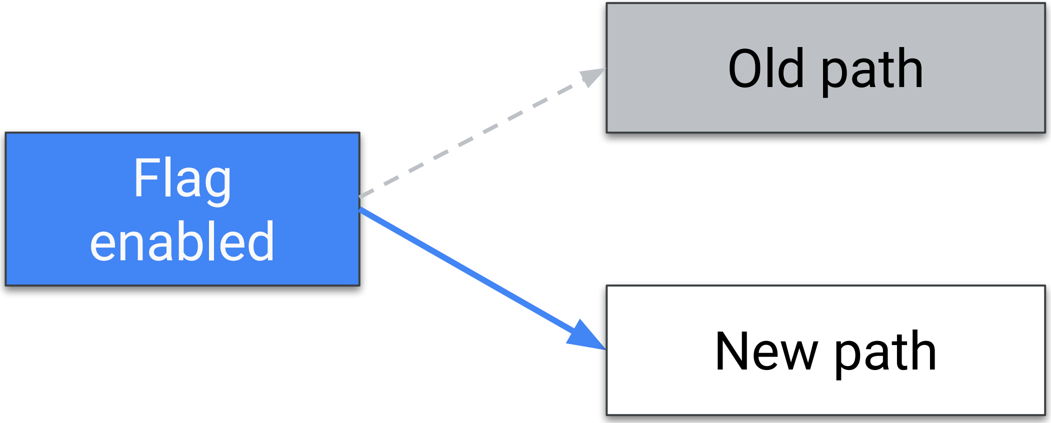
Effective

Easily accessible

Feature Flag

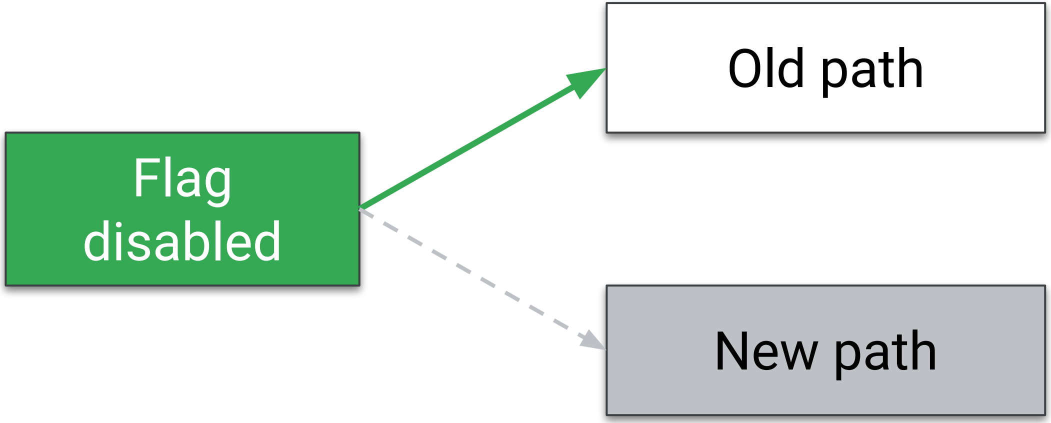


Feature Flag



Rollback?

Feature Flag



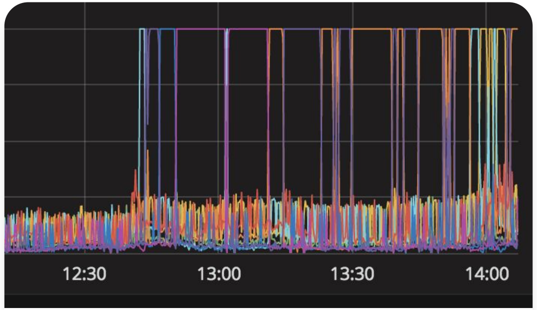
Consistency

Documented and explicit

Regular cleaning

Rollout Supervision

End users metrics



Any **behavior** changes

Rollout Roulette

Change is the first source of outages

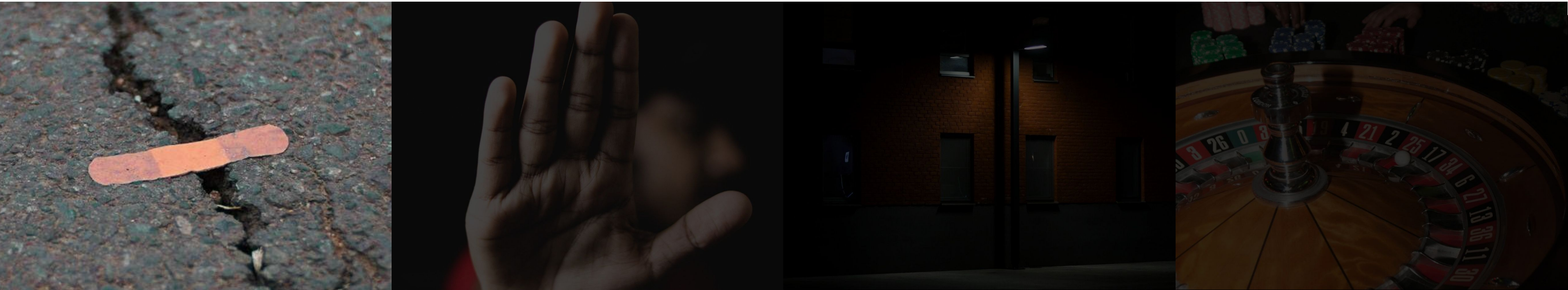
Faster is safer

Let's rely on proven industry best practices

Conclusion

We should defeat the **Reliability Procrastination Culture**

by understanding that reliability is a **force multiplier**

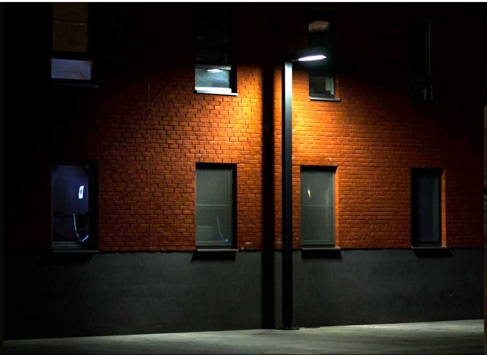


We should break free from the **Failure Denial Syndrome**

by **embracing failures**

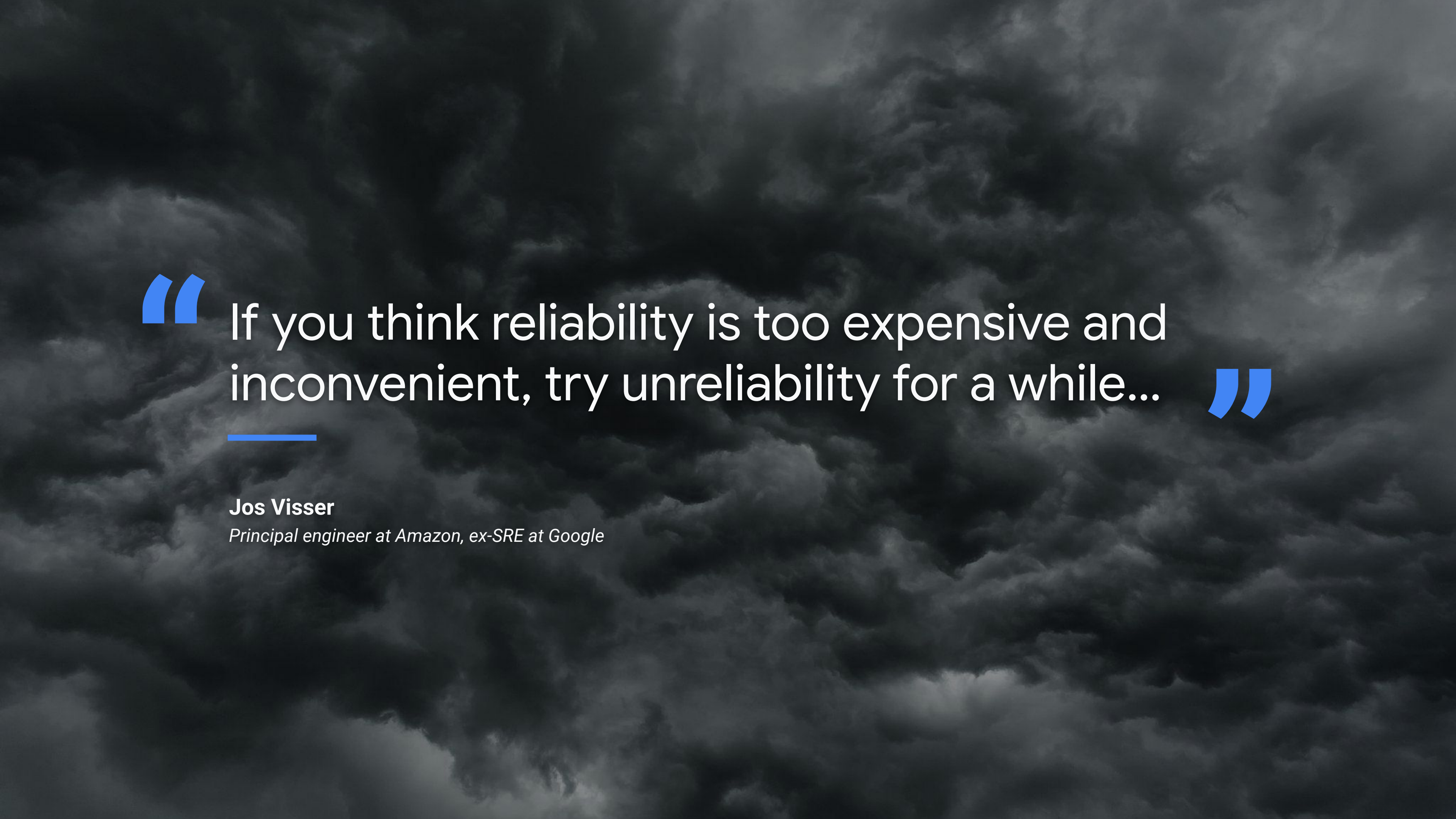
We should cure **Observability Deficiency**

by understanding **why** we need observability and how it is a **backbone** for reliability



We should defeat the **Rollout Roulette**

by building **efficient** rollout plans



> “If you think reliability is too expensive and inconvenient, try unreliability for a while...”

Jos Visser

Principal engineer at Amazon, ex-SRE at Google

Teiva Harsanyi

𝕏 teivah

teivah.io/confoo-reliability