The Hardest Problem

(and the silliest)

Marc Brooker

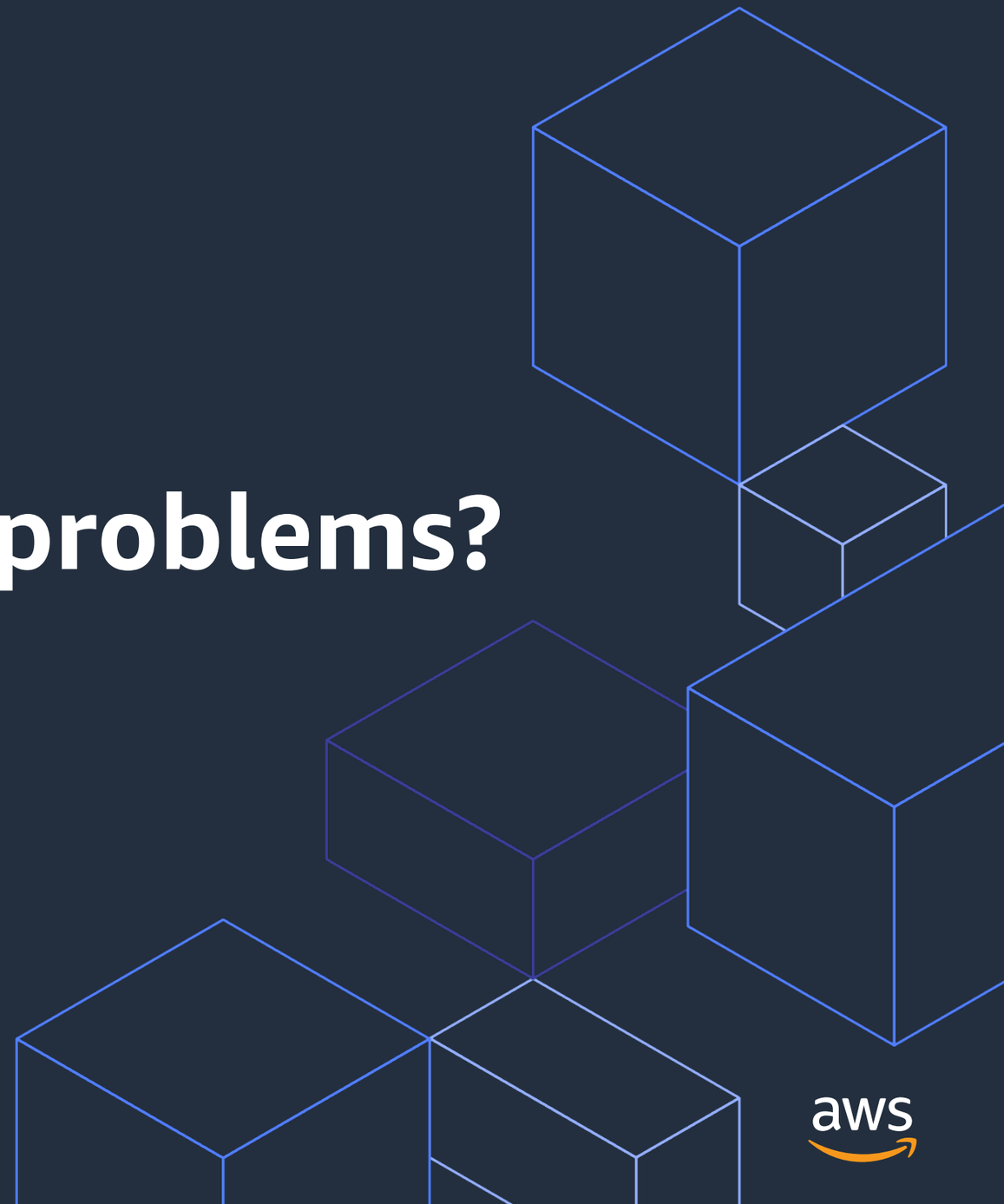
SoCC'23

About Me

- Fifteen years at AWS.
- Oncall for 15 years.
- I’ve worked on EC2, Lambda, EBS, and several other big AWS services.
- Mostly a practitioner.

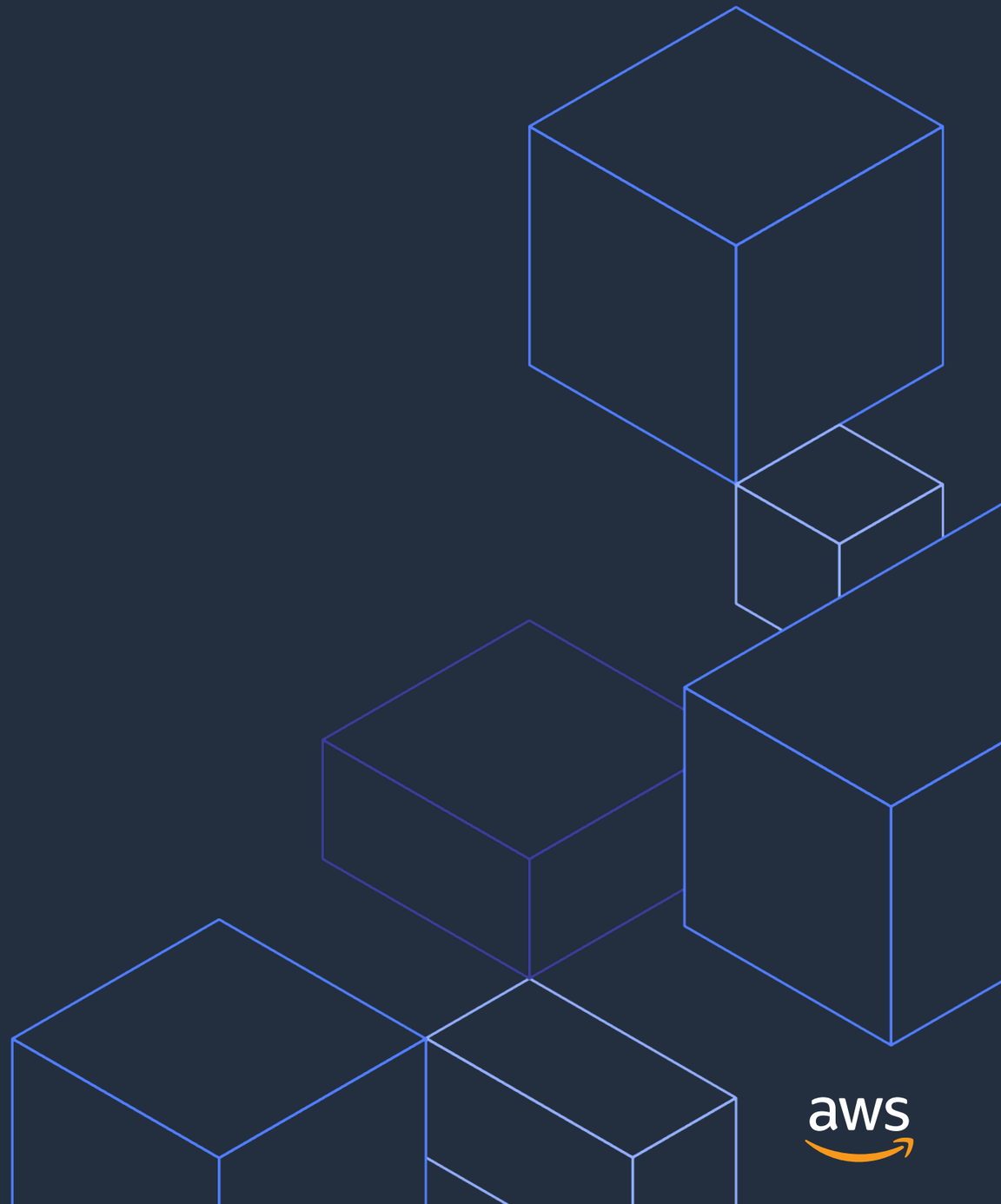
- I’ve read ~3000 cloud system postmortems, from across the industry.

What are your hardest problems?



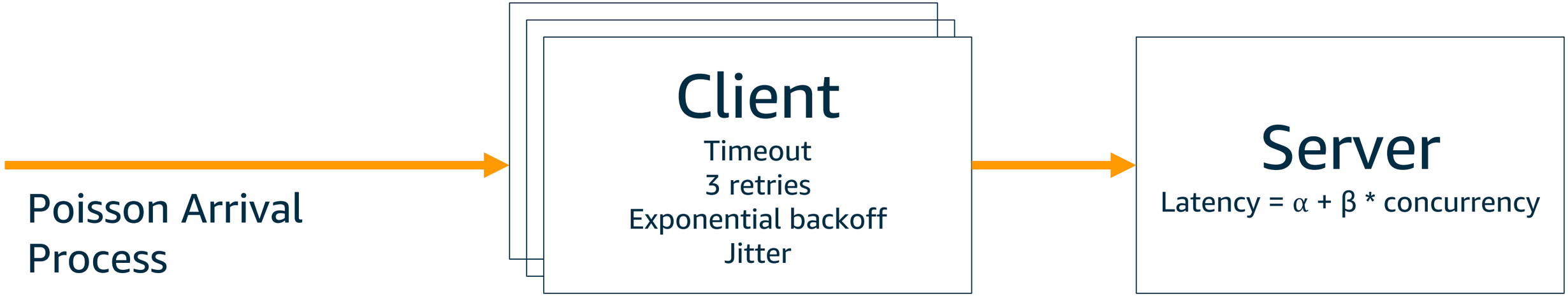
aws

The hardest problem

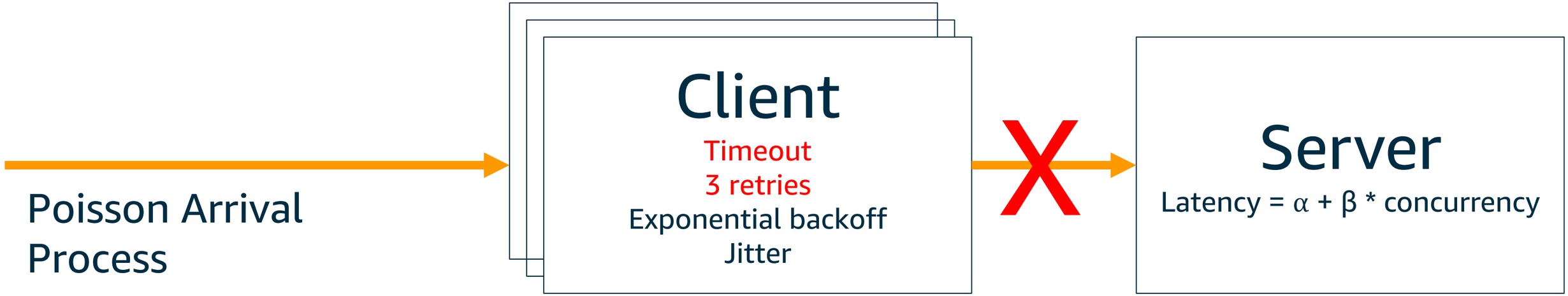


aws

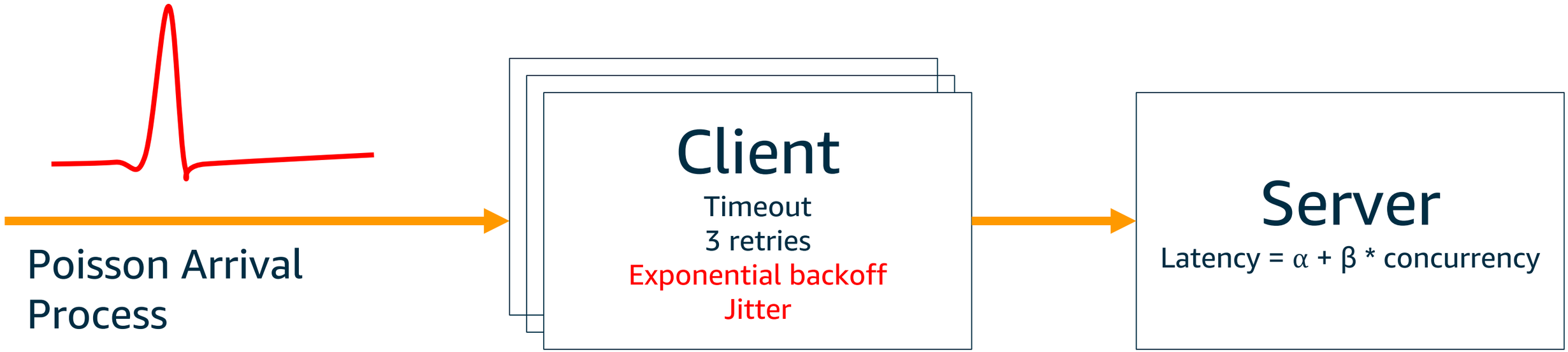
The h  t problem



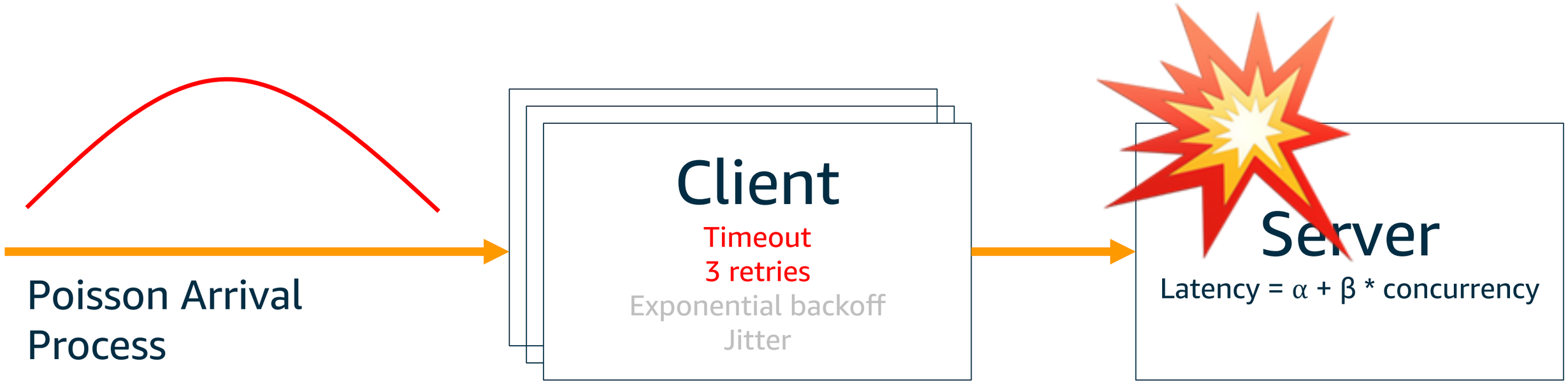
aws



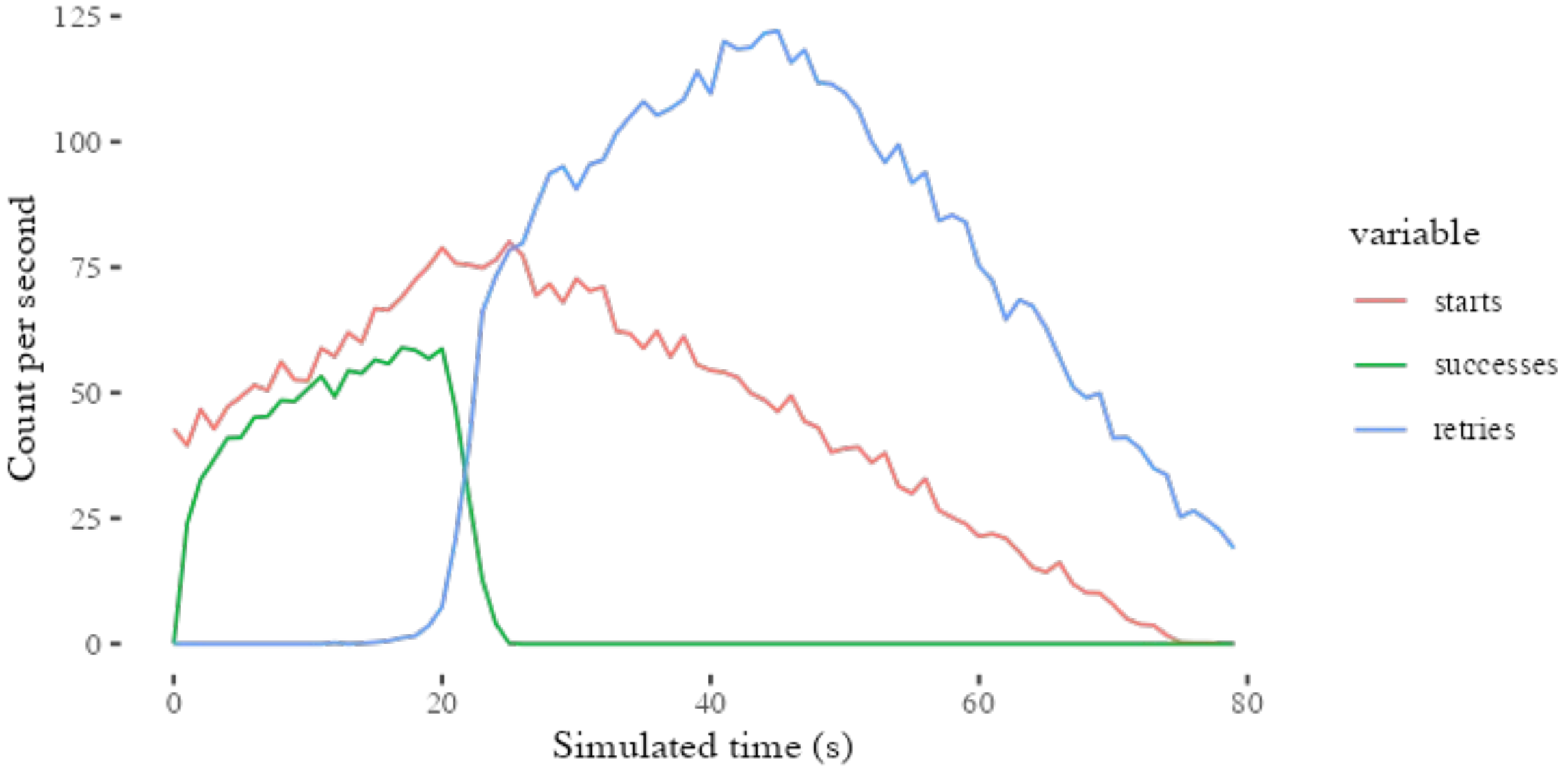
aws



aws



aws



https://github.com/mbrooker/simulator_example/blob/main/simple_collapse_sim/sim.py

aws

Can’t you just….

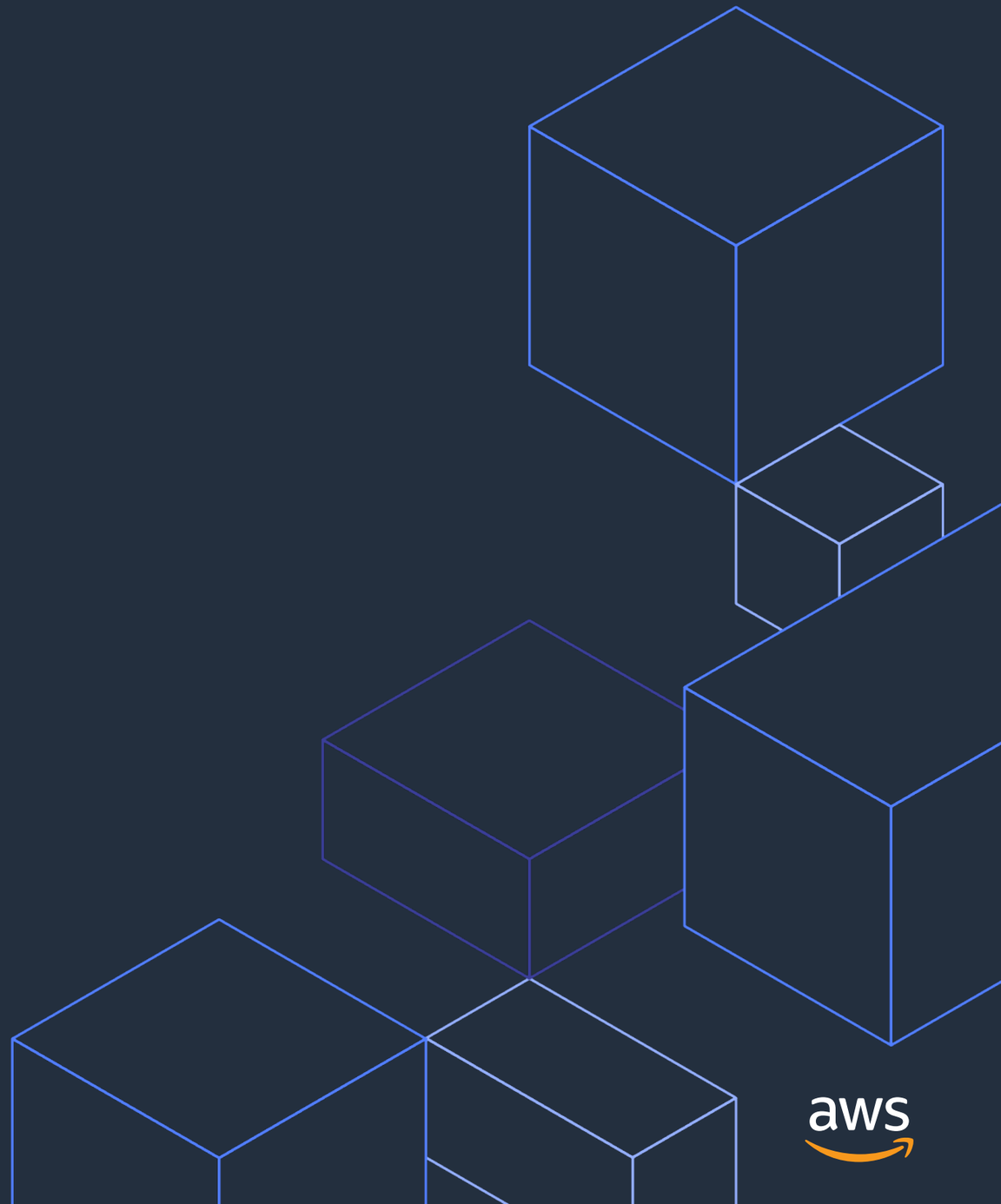
aws

They are often called Three-Phase Commit protocols. …

They have usually attempted to “fix” the Two-Phase Commit protocol by choosing another TM if the first TM fails. However, we know of none that provides a complete algorithm proven to satisfy a clearly stated correctness condition.

From Gray and Lamport, *Consensus on Transaction Commit,* ACM ToDS, 2006

The hardest problem

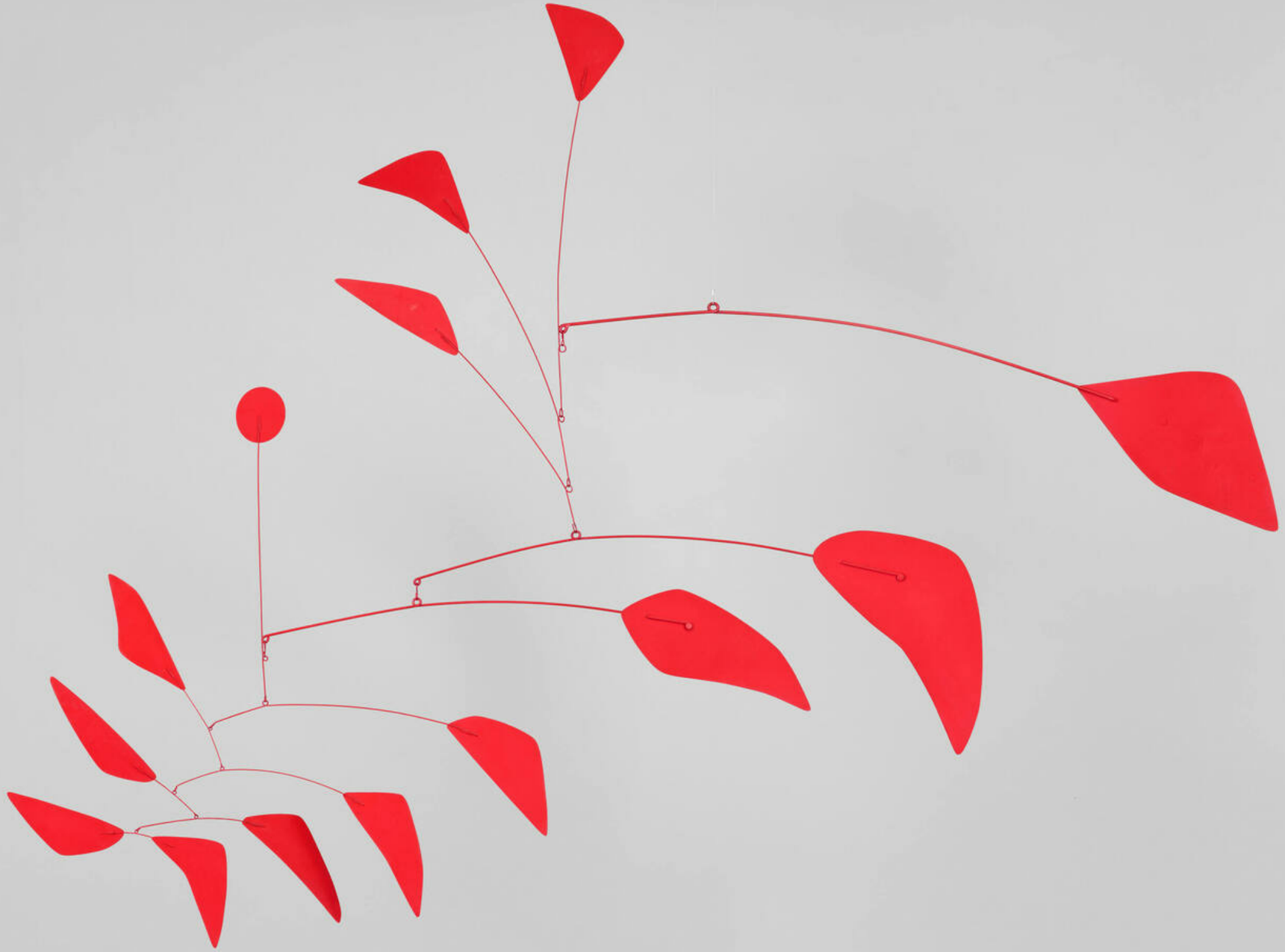


aws

Distributed systems are complex dynamical systems.

But we don’t think about them that way!

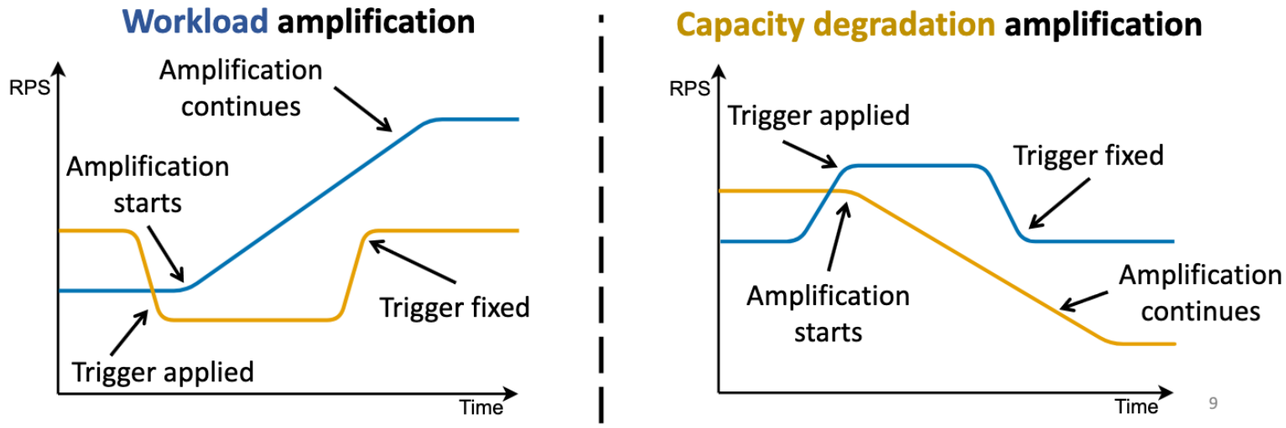
aws



Bronson et al, “Metastable failures in distributed systems”, HotOS’21
Huang et al, “Metastable Failures in the Wild”, OSDI’22

Metastability Taxonomy – Sustaining effect

- A feedback loop that keeps the system overloaded
- Two types:

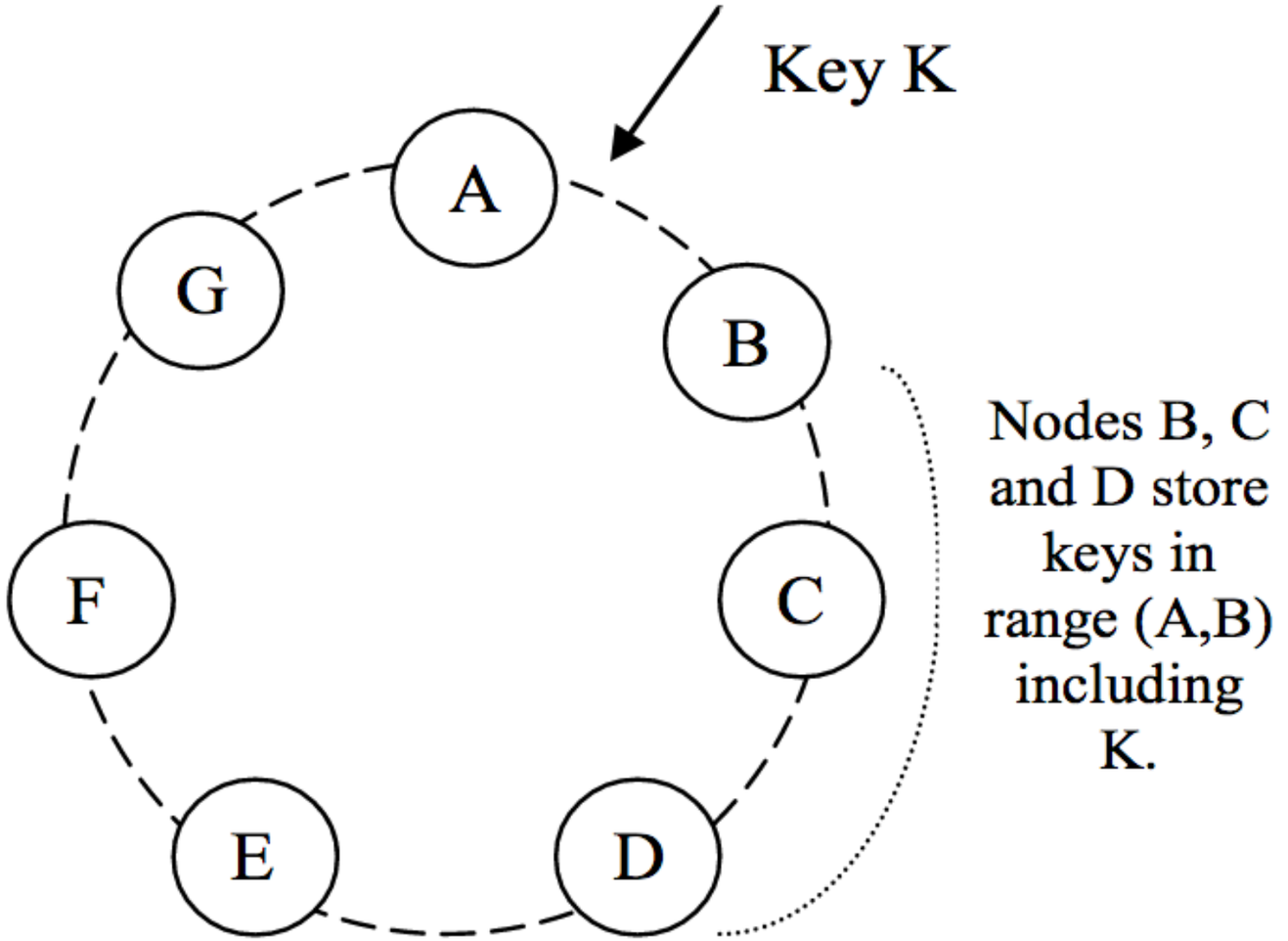


9

Metastability in the Wild – Survey

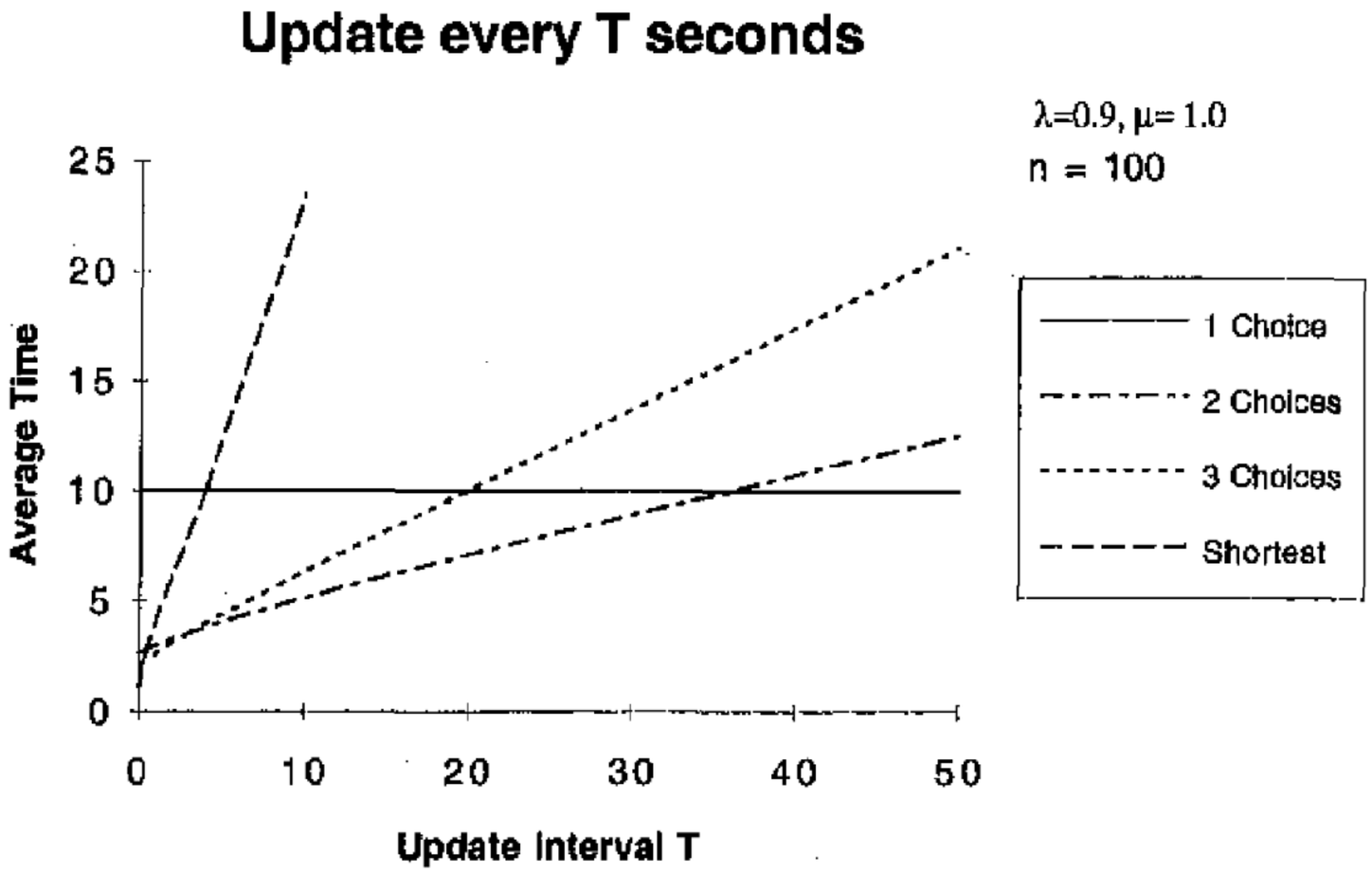
- We search through over 600 public post-mortem incident reports
  - Identify 21 metastable failures in
    - Large cloud infrastructure providers
    - Smaller companies and projects
- Can cause major outages
  - 4-10 hours most commonly
  - Incorrect handling leads to future incidents
  - An important class of failures to study

Slide from Huang, et al, *Metastable Failures in the Wild,* OSDI'22 https://www.usenix.org/conference/osdi22/presentation/huang-lexiang



Recovering from overload adds load.

De Candia et al, “Dynamo: Amazon’s Highly Available Key-value Store”, SOSP’07



What if T depends on load?

Mitzenmacher, “How useful is old information?”, 2000

Partial Solution 1: Formal Methods

aws

Lamport

What good is temporal logic?

aws

DOI:10.1145/2699417

Engineers use TLA+ to prevent serious but subtle bugs from reaching production.

BY CHRIS NEWCOMBE, TIM RATH, FAN ZHANG, BOGDAN MUNTEANU, MARC BROOKER, AND MICHAEL DEARDEUFF

How Amazon Web Services Uses Formal Methods

S3 is just one of many AWS services that store and process data our customers have entrusted to us. To safeguard that data, the core of each service relies on fault-tolerant distributed algorithms for replication, consistency, concurrency control, auto-scaling, load balancing, and other coordination tasks. There are many such algorithms in the literature, but combining them into a cohesive system is a challenge, as the algorithms must usually be modified to interact properly in a real-world system. In addition, we have found it necessary to invent algorithms of our own. We work hard to avoid unnecessary complexity, but the essential complexity of the task remains high.

Complexity increases the probability of human error in design, code, and operations. Errors in the core of the system could cause loss or corruption of data, or violate other interface contracts on which our customers de-

aws

DOI:10.1145/2699417

Engineers use TLA+ to prevent serious but subtle bugs from reaching production.

BY CHRIS NEWCOMBE, TIM RATH, FAN ZHANG, BOGDAN MUNTEANU, MARC BROOKER, AND MICHAEL DEARDEUFF

How Amazon Web Services Uses Formal Methods

S3 is just one of many AWS services that store and process data our customers have entrusted to us. To safeguard that data, the core of each service relies on fault-tolerant distributed algorithms for replication, consistency, concurrency control, auto-scaling, load balancing, and other coordination tasks. There are many such algorithms in the literature, but combining them into a cohesive system is a challenge, as the algorithms must usually be modified to interact properly in a real-world system. In addition, we have found it necessary to invent algorithms of our own. We work hard to avoid unnecessary complexity, but the essential complexity of the task remains high.

Complexity increases the probability of human error in design, code, and operations. Errors in the core of the system could cause loss or corruption of data, or violate other interface contracts on which our customers de-

Hubris
Humility
Laziness

aws

Using the Kani Rust Verifier on a Firecracker Example

Jul 13, 2022

In this post we'll apply the Kani Rust Verifier (or Kani for short), our open-source formal verification tool that can prove properties about Rust code, to an example from Firecracker, an open source virtualization project for serverless applications. We will use Kani to get a strong guarantee that Firecracker's block device is correct with respect to a simple virtio property when parsing guest requests, which may be invalid or malicious. In this way, we show how Kani can complement Firecracker's defense in depth investments, such as fuzzing.

aws

Using Lightweight Formal Methods to Validate a Key-Value Storage Node in Amazon S3

James Bornholt
Amazon Web Services
& The University of Texas at Austin

Rajeev Joshi
Amazon Web Services

Vytautas Astrauskas
ETH Zurich

Brendan Cully
Amazon Web Services

Bernhard Kragl
Amazon Web Services

Seth Markle
Amazon Web Services

Kyle Sauri
Amazon Web Services

Drew Schleit
Amazon Web Services

Grant Slatton
Amazon Web Services

Serdar Tasiran
Amazon Web Services

Jacob Van Geffen
University of Washington

Andrew Warfield
Amazon Web Services

Abstract

This paper reports our experience applying lightweight formal methods to validate the correctness of ShardStore, a new key-value storage node implementation for the Amazon S3 cloud object storage service. By "lightweight formal methods"

Using Lightweight Formal Methods to Validate a Key-Value Storage Node in Amazon S3. In *ACM SIGOPS 28th Symposium on Operating Systems Principles (SOSP '21), October 26–28, 2021, Virtual Event, Germany*. ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3477132.3483540

aws

Semantic-based Automated Reasoning for AWS Access Policies using SMT

John Backes, Pauline Bolignano, Byron Cook, Catherine Dodge, Andrew Gacek,
Kasper Luckow, Neha Rungta, Oksana Tkachuk, Carsten Varming
Amazon Web Services

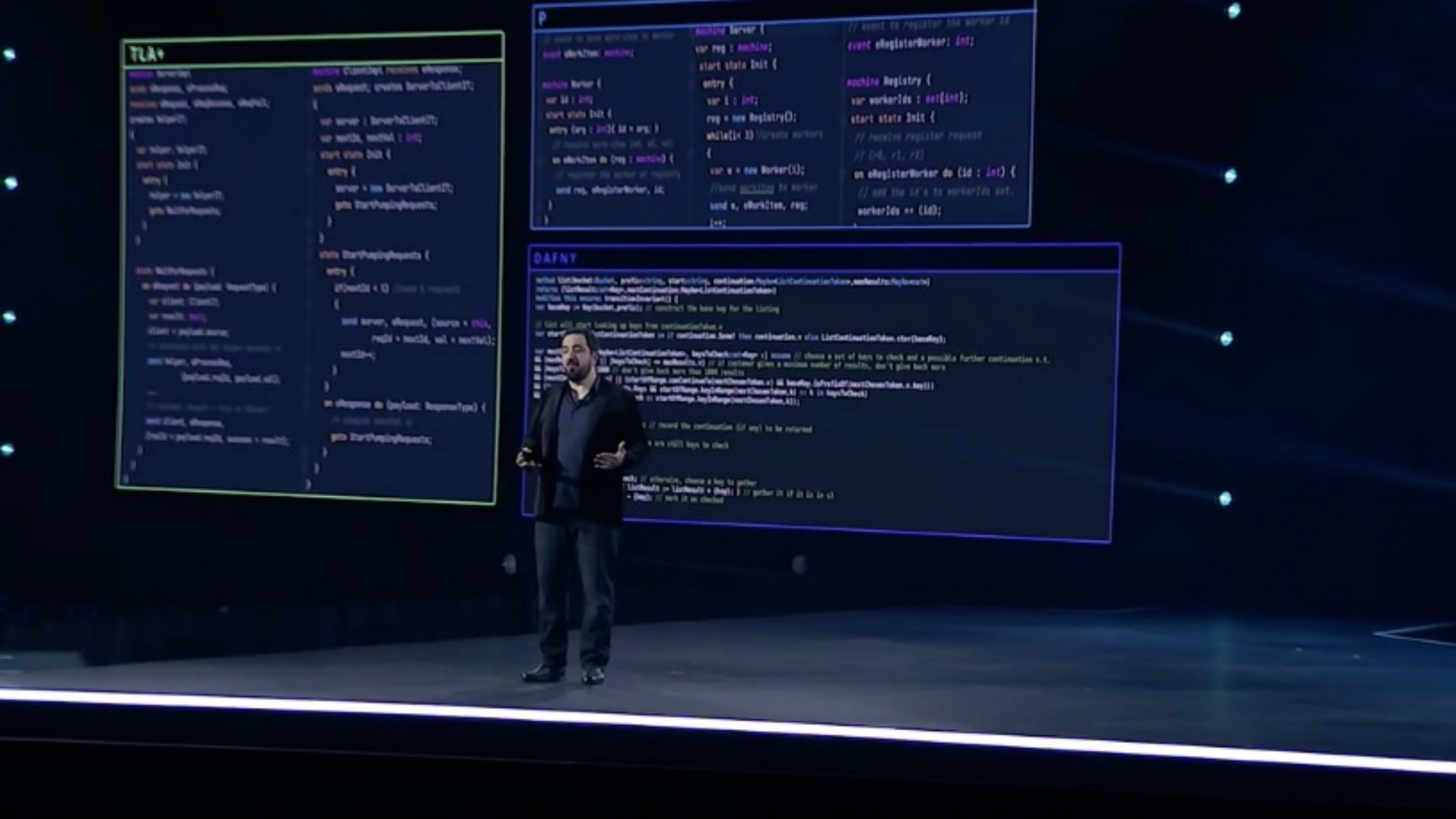
***Abstract*—Cloud computing provides on-demand access to IT resources via the Internet. Permissions for these resources are defined by expressive access control policies. This paper presents a formalization of the Amazon Web Services (AWS) policy language and a corresponding analysis tool, called ZELKOVA, for verifying policy properties. ZELKOVA encodes the semantics of policies into SMT, compares behaviors, and verifies properties. It provides users a sound mechanism to detect misconfigurations of their policies. ZELKOVA solves a PSPACE-complete problem and is invoked many millions of times daily.**

I. INTRODUCTION

In this paper, we present the development and application of ZELKOVA, a policy analysis tool designed to reason about the semantics of AWS access control policies. ZELKOVA translates policies and properties into Satisfiability Modulo Theories (SMT) formulas and uses SMT solvers to check the validity of the properties. We use off-the-shelf solvers and an in-house extension of Z3 called Z3AUTOMATA.

ZELKOVA reasons about all possible permissions allowed by a policy in order to verify properties. For example, ZELKOVA can answer the questions "Is this resource accessible by a

aws


TLA+
P
machine Server {
var reg : machine;
start state Init {
entry {
var i : int;
reg = new Registry();
while(i< 3) //create workers
{
var w = new Worker(i);
//send machine to worker
send w, eWorkItem, reg;
i++;
// event to register the worker id
event eRegisterWorker: int;
machine Registry {
var workerIds : set[int];
start state Init {
// receive register request
// (id, r1, r2)
on eRegisterWorker do (id : int) {
// add the id's to workerIds set.
workerIds += (id);
DAFNY

Safety

Liveness

[]❤️

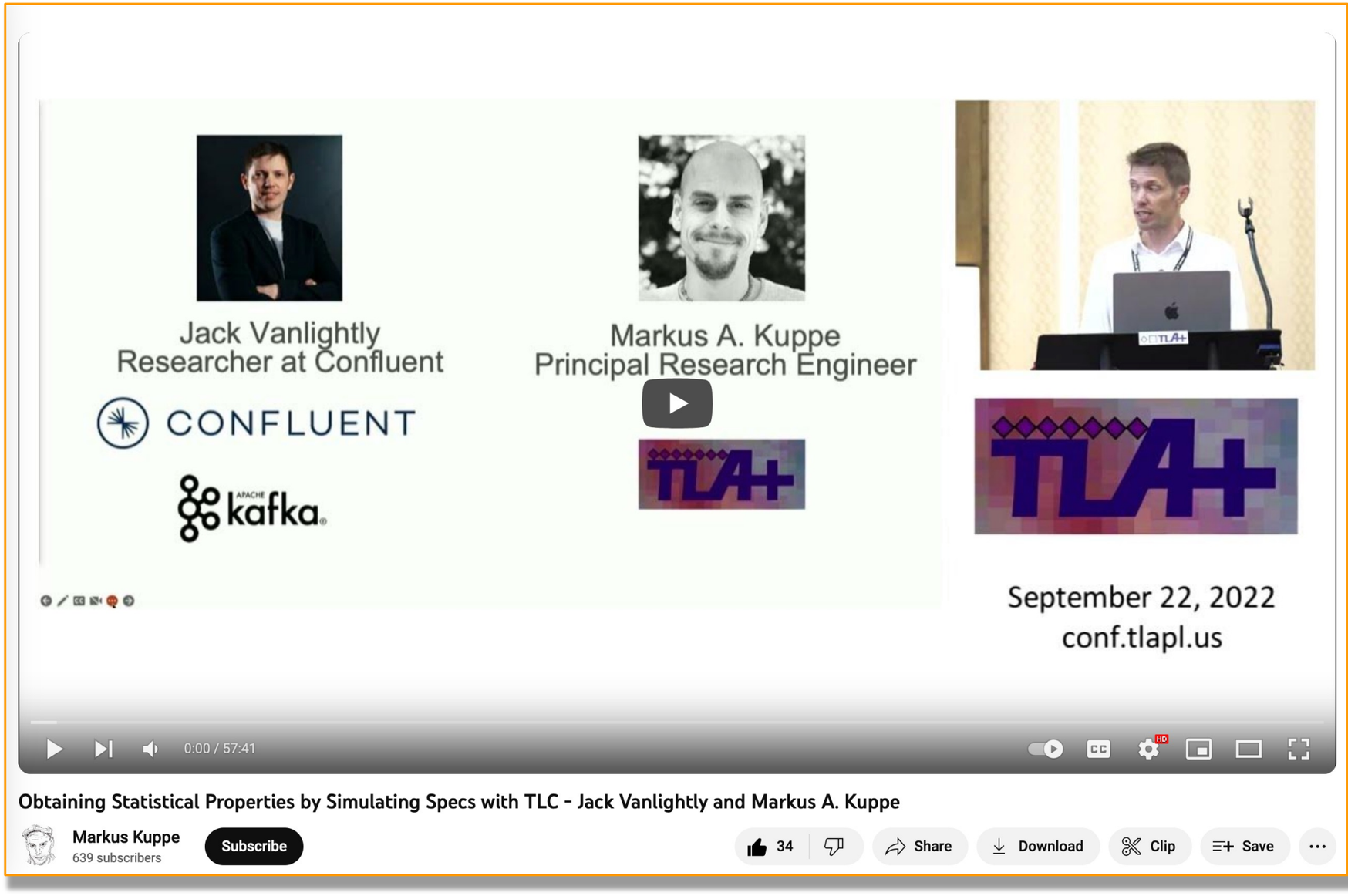
aws

Safety

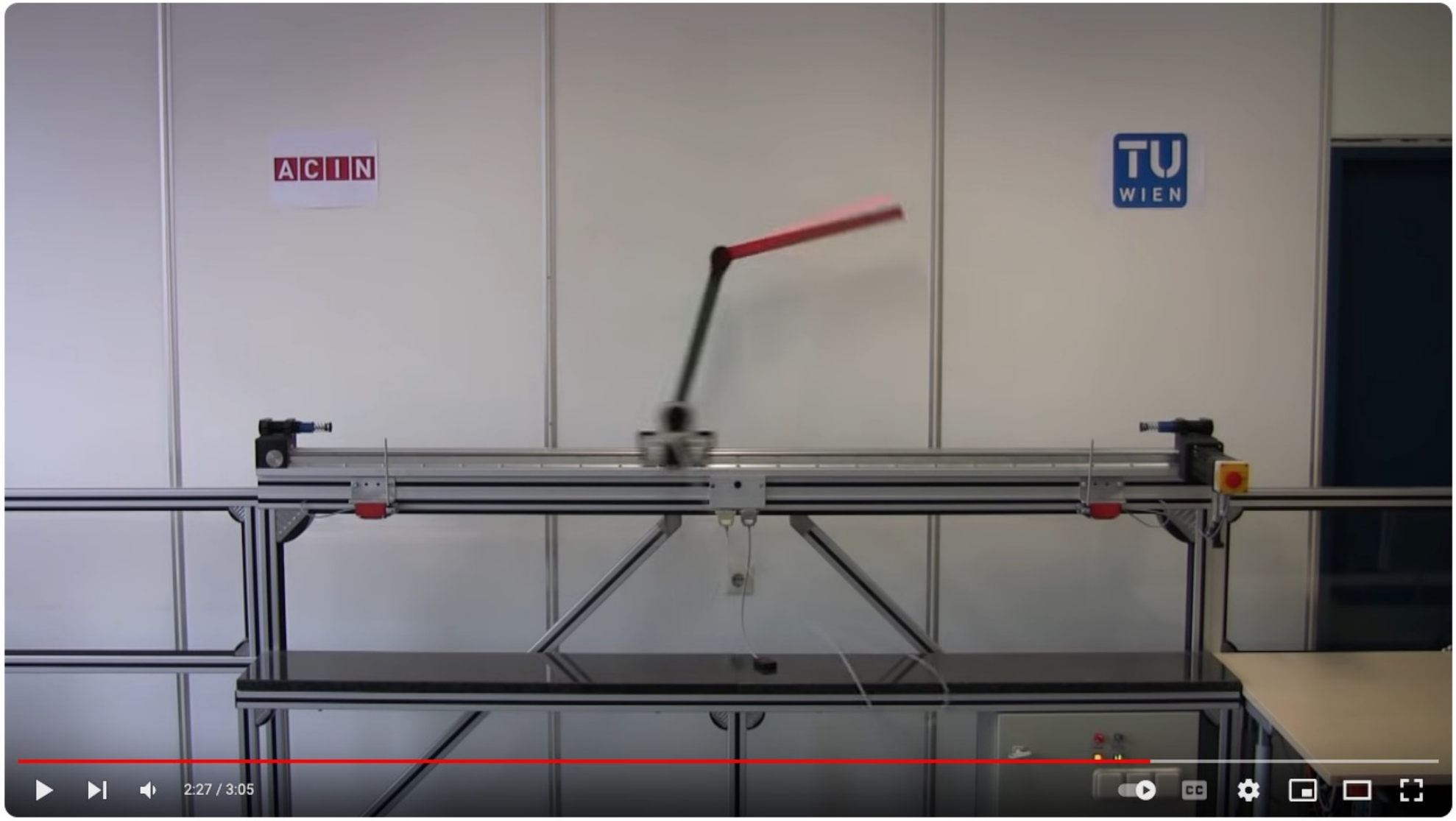
Liveness

<> 💔

aws



Partial Solution 2: Control Theory



Double Pendulum on a Cart

Tobias Glück
802 subscribers

Subscribe

2.2K Share Download Clip Save

Challenges to Applying Control Theory

- Non-linearity is common
  (doesn’t obey superposition principle)
- Non-time-invariance in ubiquitious
- Stochastic behavior is widespread

“Reasoning purely analytically about the behavior of complex stochastic systems is generally infeasible.”

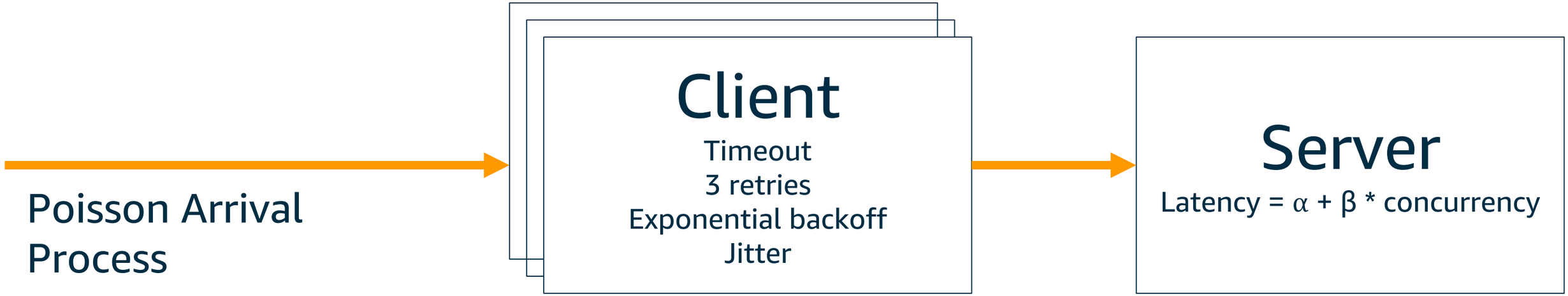
Agha and Palmskog, *A Survey of Statistical Model Checking,* TOMACS, January 2018

aws

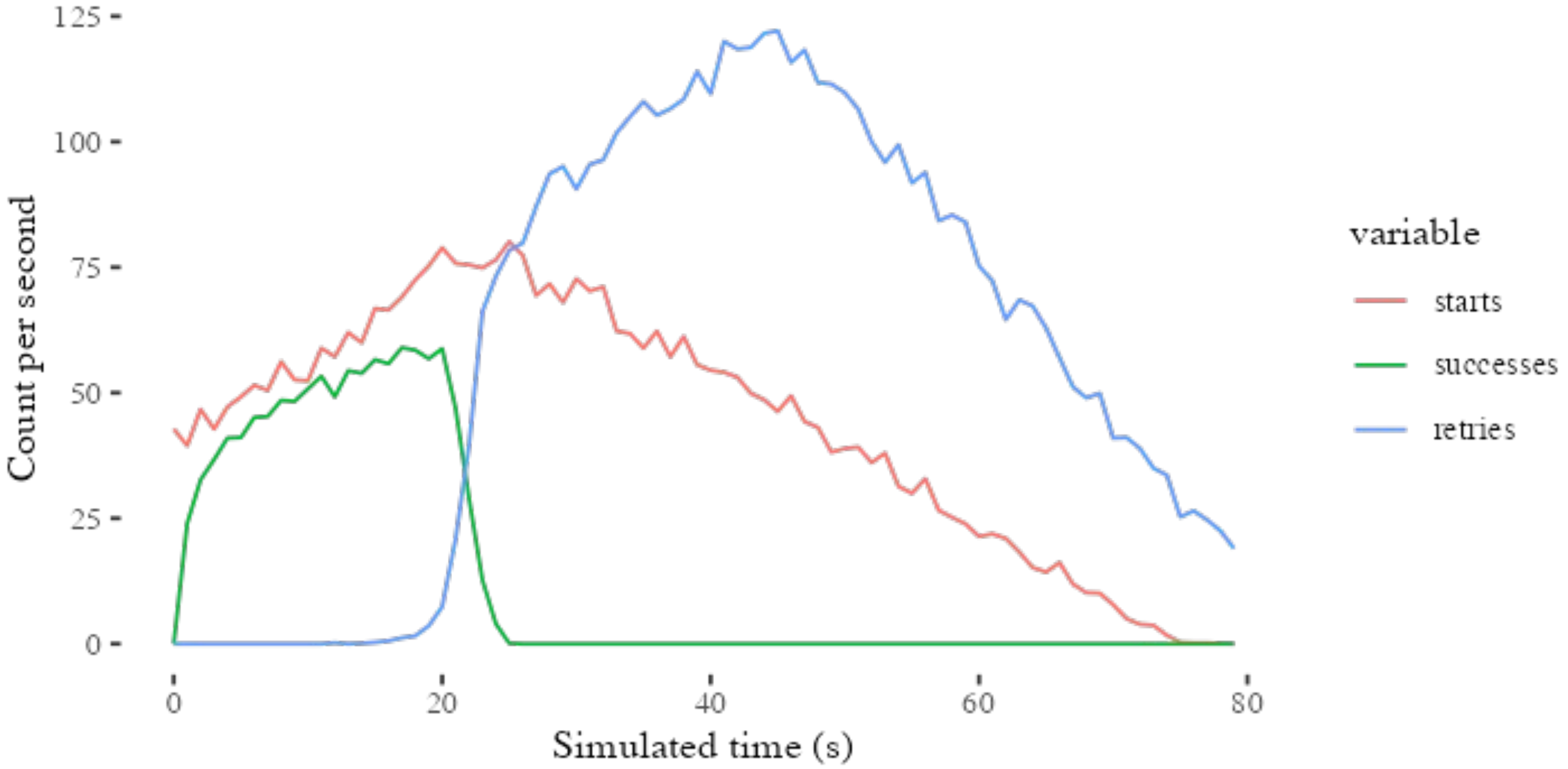
Partial Solution 3: Simulation

Simulation & Numerical Methods

Created with https://github.com/narimiran/double_pendulum

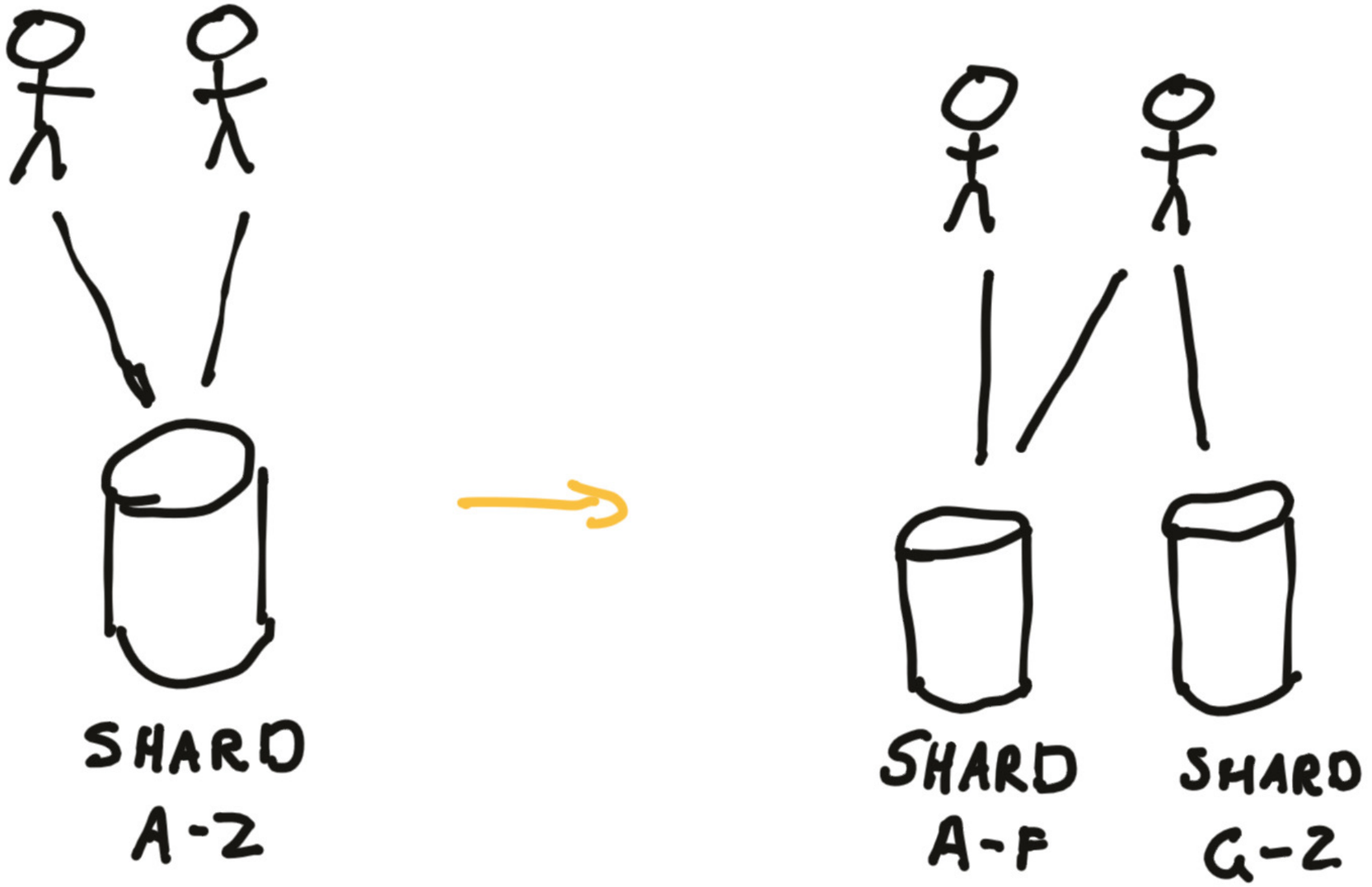


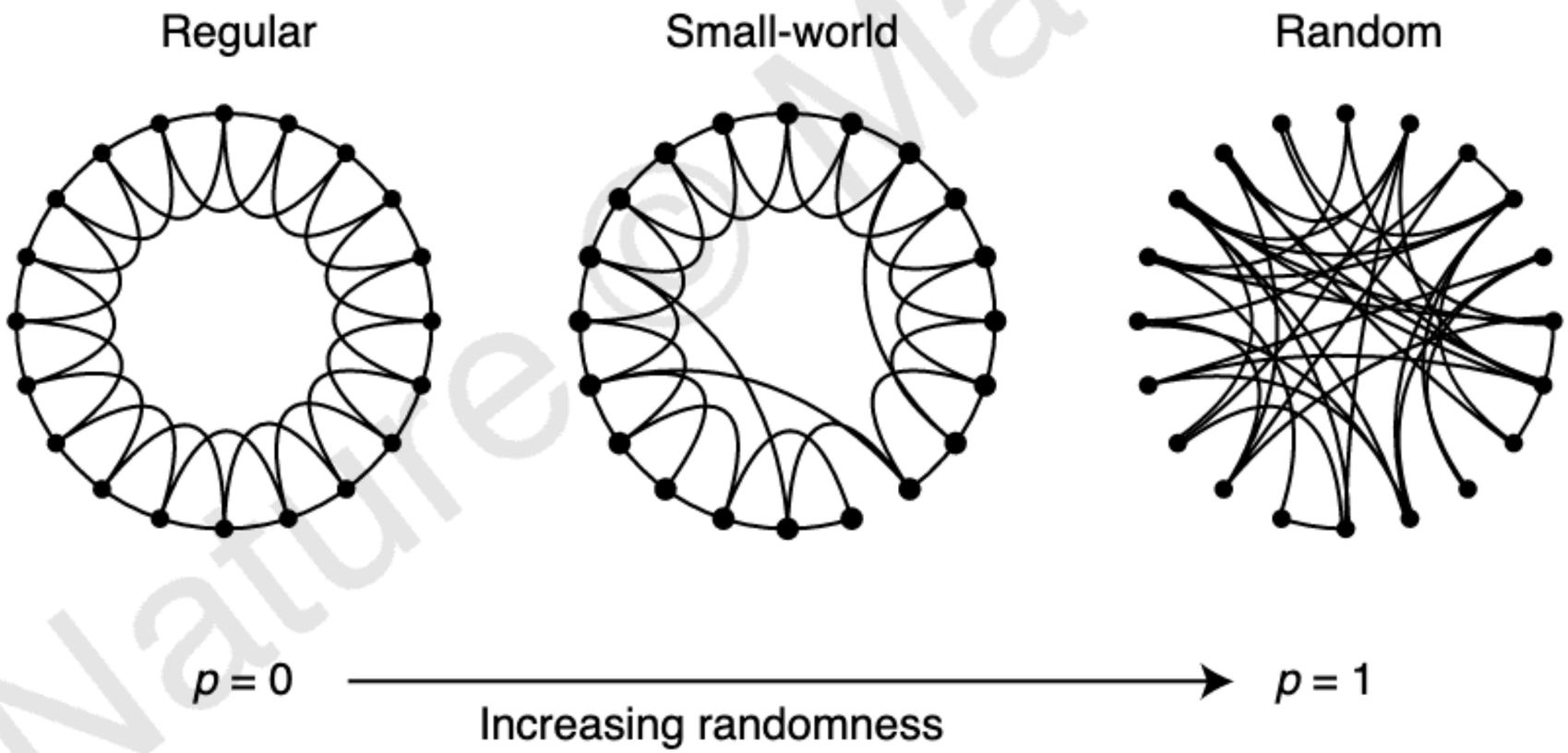
aws



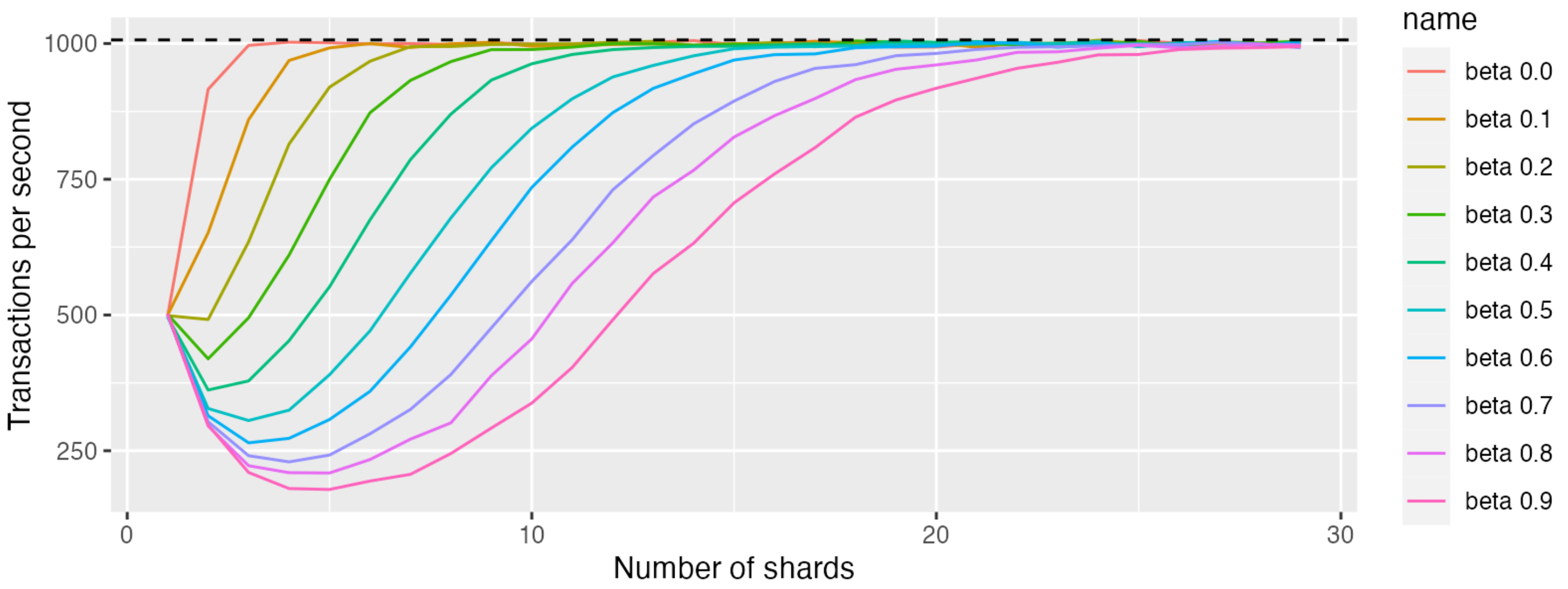
https://github.com/mbrooker/simulator_example/blob/main/simple_collapse_sim/sim.py

aws





Watts and Strogatz, “Collective dynamics of ‘small-world’ networks”, Nature, 1998



aws

~5,000 lines of Rust

but…

aws

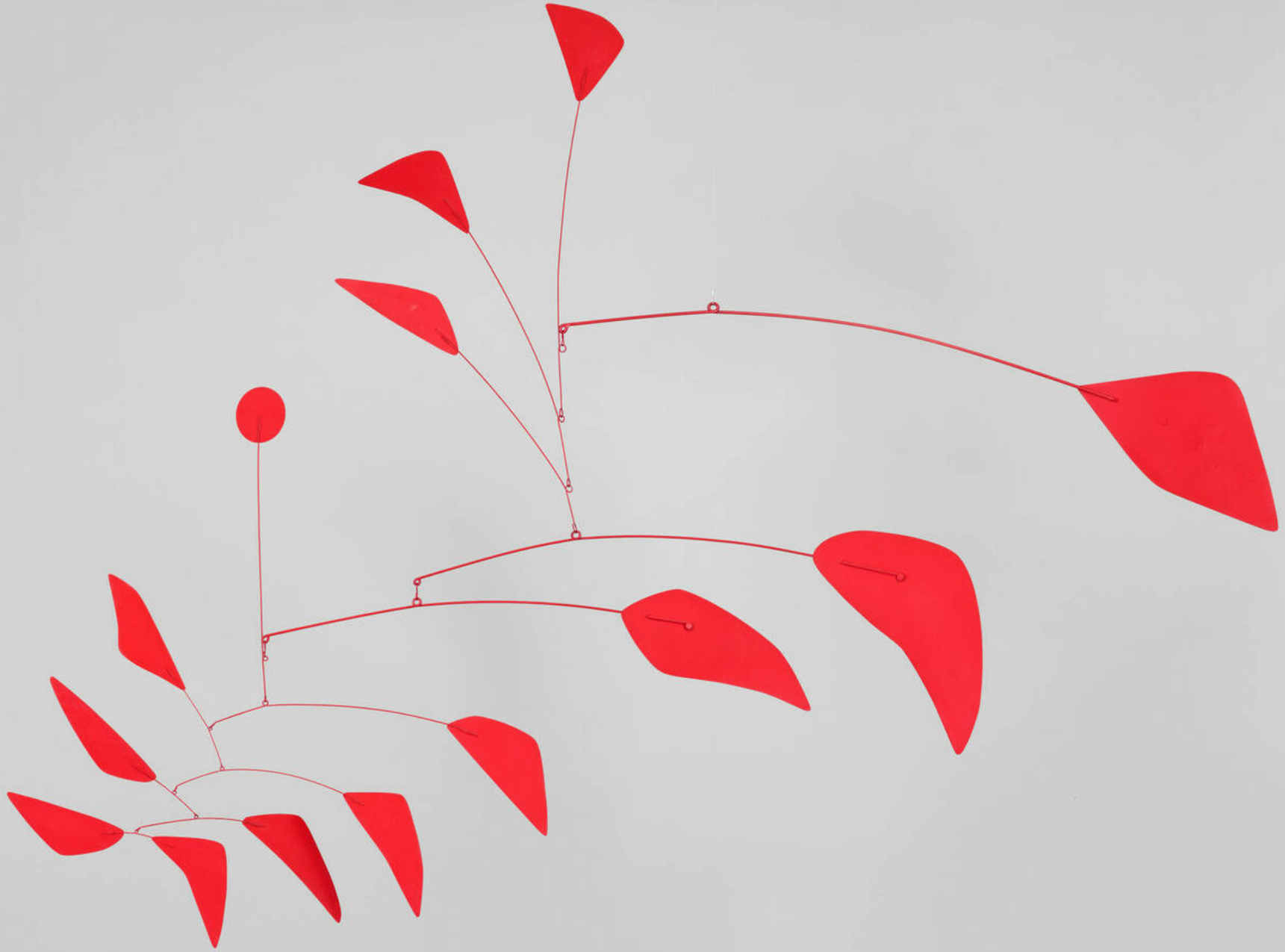
We already have a specification!

We already have a tool that searches the specification's state space!

aws

Call To Action 1:
Rethink the Foundations of Computer Science

aws



Call To Action 2:
Rethink the Systems Field’s Priorities

aws

Value stability as a design goal in systems research.

(and value security, durability, integrity, availability, more too).

aws

Call To Action 3:
Build a Stronger Foundation, Together

aws

Theory: a foundation for thinking about emergent properties of distributed systems.

aws

Tools: automated reasoning about system stability.

aws

Practices: better defaults.

A safer toolkit.

aws

Questions (or answers!)

Marc Brooker
mbrooker@amazon.com
https://brooker.co.za/blog/

aws