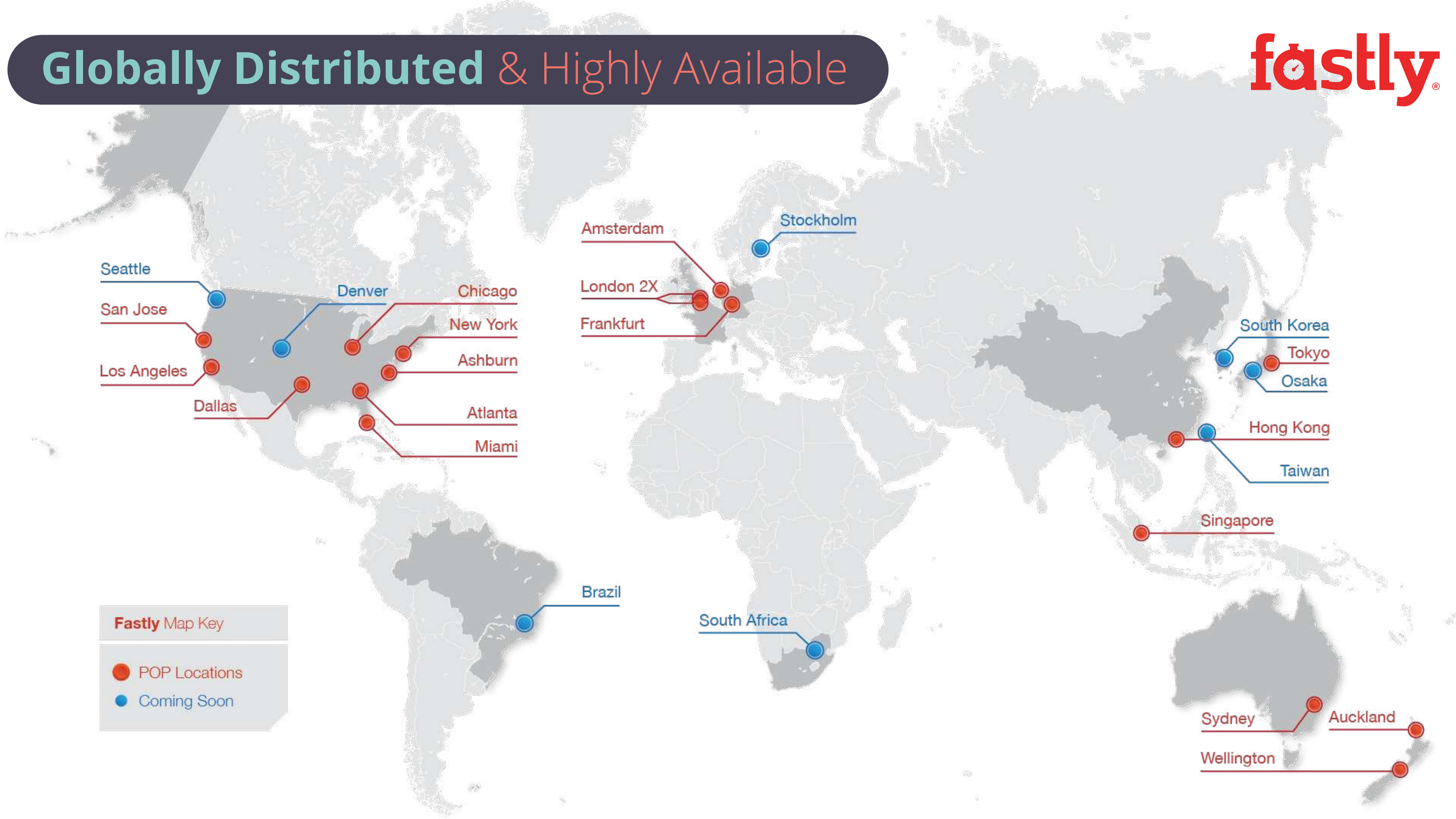
# Testing in a Distributed World


# Ines Sombra

fastly®

@randommood | ines@fastly.com

# **Globally Distributed** & Highly Available



4

# CONCLUSIONS, RANTS, & PUGS

1

# DISTRIBUTED SYSTEMS TESTING

3

# DST IN THE WILD

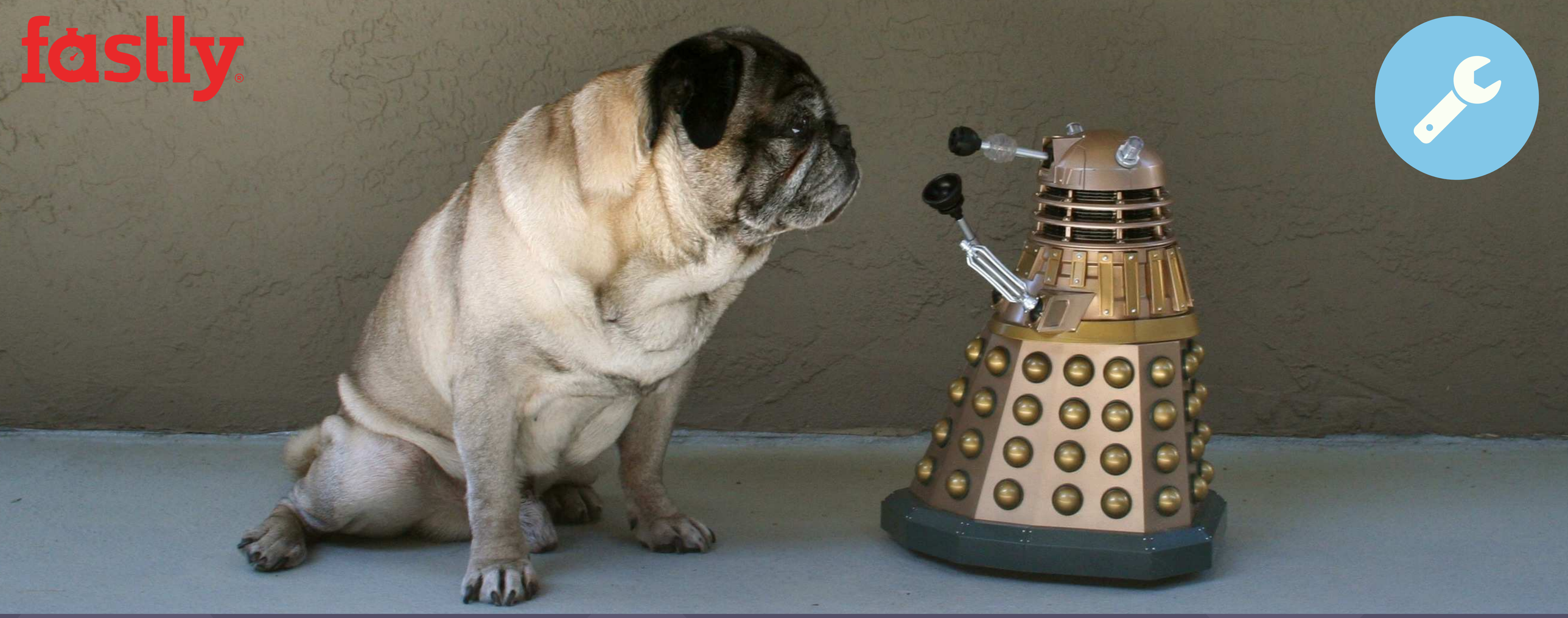
2

# DST IN ACADEMIA

fastly®

* **DST**: Distributed Systems Testing

fastly



# DISTRIBUTED SYSTEMS TESTING

# Why do we test?

*We test to gain* ***confidence*** *that our system is* ***doing the right thing*** *(now & later)*

fastly®

# Many types of **tests**

Today

## SMALL

- Unit
- Integration (maybe)

## MEDIUM

- Unit
- Integration
- System
- Acceptance

## COMPLEX SYSTEM

- Unit
- Integration
- System
- Acceptance
- Compatibility
- Fault injection
- Stress / performance
- Canary
- Regression

fastly®

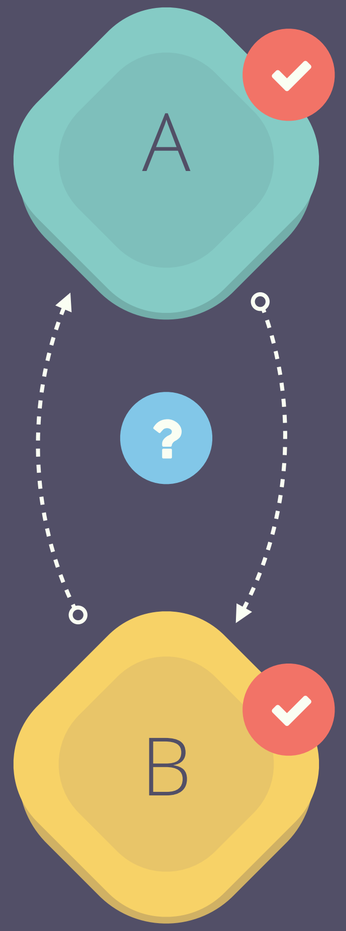

Nondeterminism
Failures
Timing
CHALLENGES OF DST
Ordering
No centralized view
Many states
Unbounded inputs
Concurrency
fastly

# Challenges of DST

*Behavior is **aggregate***

---

*Components tested in isolation also need to be tested **together***




fastly®

# Detour: Hierarchy of **Errors***


*Deadlocks*

*Livelock / starvation*

*Under specification*

*Over specification*

fastly

* Stolen from Henry Robinson's PWLSF talk

# TESTING DISTRIBUTED SYSTEMS

Difficult to approach & many factors in play

Aim to gain confidence of proper system behavior now & later

Behavior is **aggregate**

fastly®

# DST & ACADEMIA

# Formal Methods

HUMAN ASSISTED PROOFS

MODEL CHECKING

LIGHTWEIGHT FM

# “Scholarly” Testing

TOP-DOWN

BOTTOM-UP

WHITE / BLACK BOX

fastly

# Formal Methods

**NOTE: THE CHOICE OF METHOD TO USE IS APPLICATION DEPENDENT**


HUMAN ASSISTED PROOFS

CONSIDERED SLOW & HARD TO USE. *SAFETY-CRITICAL DOMAINS ( TLA+, COQ, ISABELLE )*

MODEL CHECKING

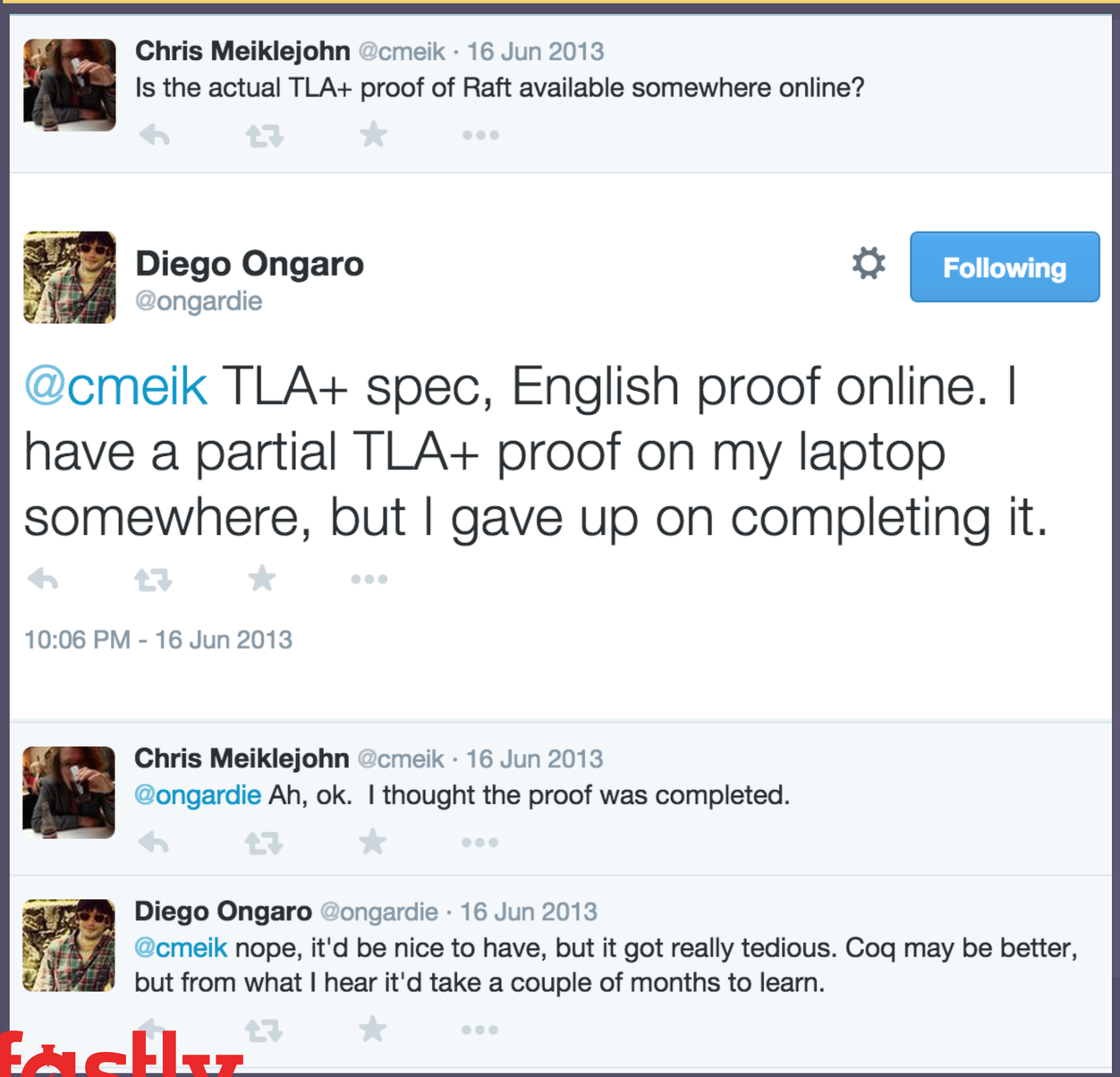
STATE-MACHINE OF PROPERTIES, & TRANSITIONS. PARTICULARLY USED IN PROTOCOLS *( TLA+, MODIST, SPIN, ...)*

LIGHTWEIGHT FM

BEST OF BOTH WORLDS *( ALLOY, SAT )*

fastly

# HUMAN ASSISTED PROOFS



**Chris Meiklejohn** @cmeik · 16 Jun 2013
Is the actual TLA+ proof of Raft available somewhere online?

**Diego Ongaro**
@ongardie
Following

@cmeik TLA+ spec, English proof online. I have a partial TLA+ proof on my laptop somewhere, but I gave up on completing it.

10:06 PM - 16 Jun 2013

**Chris Meiklejohn** @cmeik · 16 Jun 2013
@ongardie Ah, ok. I thought the proof was completed.

**Diego Ongaro** @ongardie · 16 Jun 2013
@cmeik nope, it'd be nice to have, but it got really tedious. Coq may be better, but from what I hear it'd take a couple of months to learn.

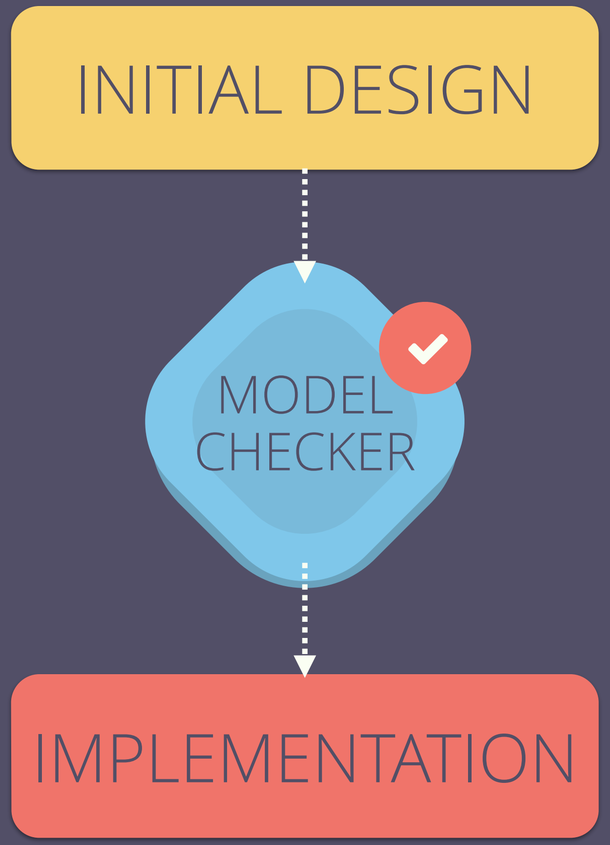
**Temporal logic of actions (TLA):** logic which combines temporal logic with a logic of actions.

TLA+ **verifies all traces exhaustively.**

Slowly making their way to industry

fastly

# MODEL CHECKING

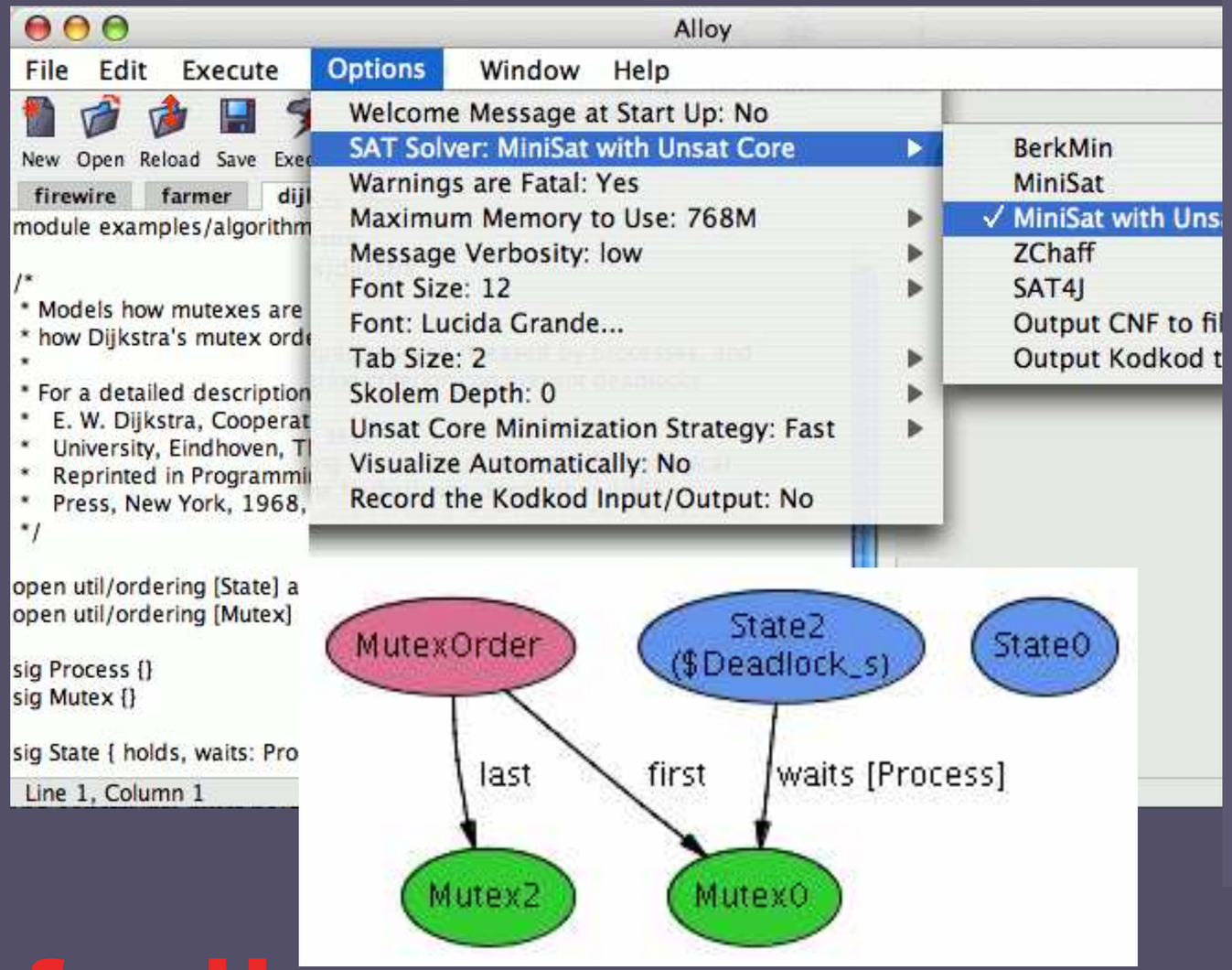



**SPIN**: Model of system design & requirements (properties) as input. Checker tells us if they hold.

If not a **counterexample** is produced (system run that violates the requirement)

ProMeLa (Process Meta Language) to describe models of dst systems (c-like)

# LIGHTWEIGHT FM







**Alloy**: solver that takes constraints of a model and finds structures that satisfy them

Can be used both to **explore** the **model** by generating sample structures, and to **check properties** by generating counterexamples.

fastly

*Developed ecosystem: Java, Ruby & more. Used a lot!*

# “Scholarly” Testing

TOP-DOWN

FAULT INJECTORS, INPUT GENERATORS

BOTTOM-UP

LINEAGE DRIVEN FAULT INJECTORS

WHITE / BLACK BOX

WE KNOW (OR NOT) ABOUT THE SYSTEM

Pay-as-you-go: **gradually increase confidence**

**Sacrifice rigor** (less certainty) for something reasonable

Challenged by large state space

fastly®

# WHITE BOX / STATIC ANALYSIS

Failures require only 3 nodes to reproduce. Multiple inputs needed (~ 3) in correct order.

Faulty error handling code culprit. Complex sequences.

**Aspirator**: Tool capable of finding bugs (JVM). 121 new bugs & 379 bad practices!

fastly


**Simple Testing Can Prevent Most Critical Failures: An Analysis of Production Failures in Distributed Data-Intensive Systems**

Ding Yuan, Yu Luo, Xin Zhuang, Guilherme Renna Rodrigues, Xu Zhao, Yongle Zhang, Pranay U. Jain, and Michael Stumm, *University of Toronto*

https://www.usenix.org/conference/osdi14/technical-sessions/presentation/yuan

| Software | % of failures reproducible by unit test |
|---|---|
| Cassandra | 73% (29/40) |
| HBase | 85% (35/41) |
| HDFS | 82% (34/41) |
| MapReduce | 87% (33/38) |
| Redis | 58% (22/38) |
| Total | 77% (153/198) |

is sponsored by USENIX.

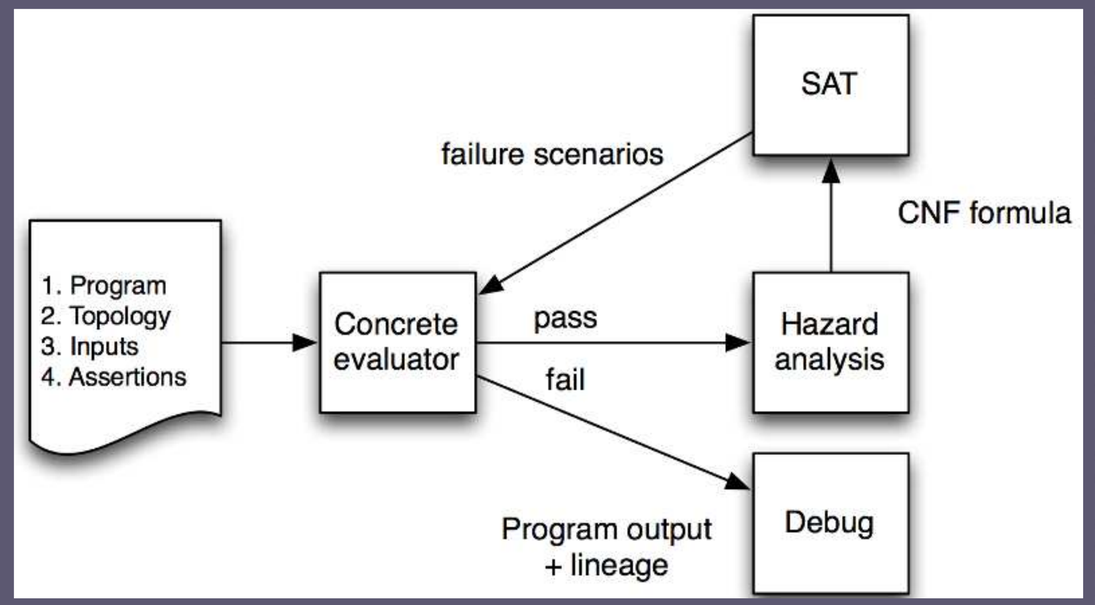
# BOTTOM-UP / **MOLLY**: LINEAGE DRIVEN FAULT INJECTION

Reasons backwards from correct system outcomes & determines if a failure could have prevented it.

Molly **only injects the failures it can prove might affect an outcome**

Counterexamples + Lineage visualizations to **help you understand why**

fastly




Lineage-driven fault injection

**ABSTRACT**

In large-scale data management systems, failure is pr
tainty. Fault-tolerant protocols and components are n
ficult to implement and debug. Worse still, choosing
tolerance mechanisms and integrating them correctly
systems remains an art form, and programmers hav
assist them.

We propose a novel approach for discovering bugs
data management systems: *lineage-driven fault injecti*
driven fault injector reasons *backwards* from correct sy
comes to determine whether failures in the execution could have
prevented the outcome. We present MOLLY, a prototype of lineage-
driven fault injection that exploits a novel combination of data lin-
eage techniques from the database literature and state-of-the-art

volving complex combinations of multiple instances and types of
failures (e.g., a network partition followed by a crash failure). Ap-
proaches such as Chaos Monkey [1] explore faults randomly, and
hence are unlikely to find rare error conditions caused by complex

hniques—
guarantee
ameworks
earch and
stic search
arios, but,
f possible

outed data
enrich the
rmal com-
principled
should be
d to mean-
o be *com-*
rexamples
*rantee* that
f possible

trategy for
s: *lineage-*
tabase lit-
h allows it
ssages that
s (from ef-
could have
than gen-

erating faults at random (or using application-specific heuristics), a
lineage-driven fault injector chooses only those failures that could
have affected a known
outcome, exercising fault-tolerance
code at increasing lev
plexity. Injecting failures in this
targeted way allows I
vide completeness guarantees like
those achievable with
methods such as model checking,
which have typically been used to verify small protocols in a bottom-

# DST IN ACADEMIA

Great research but accessibility is an issue

A few frameworks are emerging

Some disconnect but reducing thanks to OSS

fastly®

# DST IN THE WILD

fastly®

State machine specification language & compiler to translate specs to C++

Core algorithm: **2 explicit state machines**

**Test safety vs liveness mode**. All tests start in safety & inject random failures. Tests turned to **liveness** mode to verify system is not deadlocked. **Repeatable**

fastly

Paxos Made Live - An Engineering Perspective

Tushar Chandra
Robert Griesemer
Joshua Redstone

Google

June 20, 2007

**Abstract**

We describe our experience in building a fault-tolerant data-base using the Paxos consensus algorithm. Despite the existing literature in the field, building such a database proved to be non-trivial. We describe selected algorithmic and engineering problems encountered, and the solutions we found for them. Our measurements indicate that we have built a competitive system.

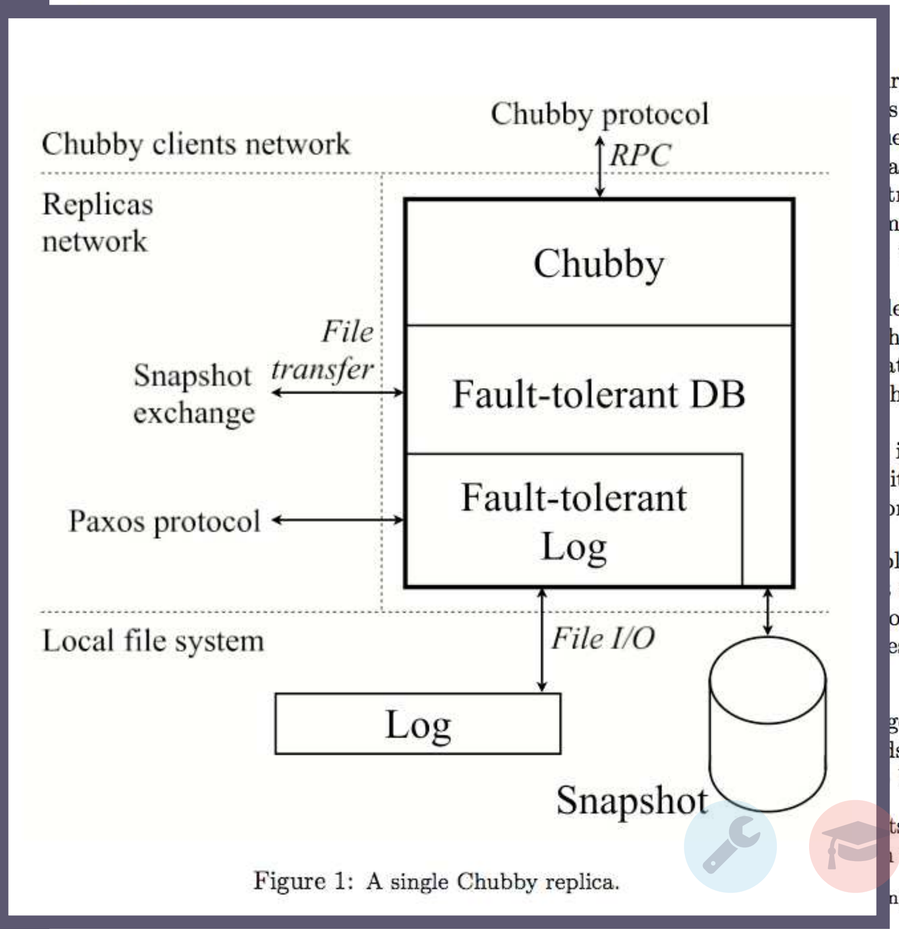



Figure 1: A single Chubby replica.

rough replication [17, 18]. A
s are mutually consistent [8,
es, it is possible to build an
ata structure, application of
tructures on all replicas. For
ne sequence of operations is
up with the same database

lerant primitives, of which a
has been studied extensively
t operate within a multitude
hm [8] has been discussed in

implements a fault-tolerant
ite the existing literature on
r a variety of reasons:

lementation contains several
that we used C++ instead
onverting the algorithm into
es and optimizations – some

gorithms (one page of pseudo
ls of lines of code. To gain
be used.

ts. H the real world
the m, bugs in its

ngs of ACM PODC 2007.

Precise description of system in TLA+ (PlusCal language - like c)

Used it in 6 large complex real-world systems. 7 teams use TLA+

Found subtle bugs & confidence to make **aggressive optimizations** w/o sacrificing correctness

Use formal specification **to teach** system to new engineers

fastly

**Use of Formal Methods at Amazon Web Services**

Chris Newcombe, Tim Rath, Fan Zhang, Bogdan Munteanu, Marc Brooker, Michael Deardeuff
Amazon.com

11th November, 2013

amazon web services™

…en using formal specification and model-…ms. This paper describes our motivation …and what has not. When discussing

…s to use. That external simplicity is built …mplex internals are required to achieve …and also to cope with relentless rapid …nched S3, our Simple Storage Service. In the 6 years after launch, S3 grew to store 1 trillion objects [1]. Less than a year later it had grown to 2 trillion objects, and was regularly handling 1.1 million requests per second [2].

S3 is just one of tens of AWS services that store and process data that 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 major challenge, as the algorithms must usually be modified in order 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.

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 depend. So, before launching such a service, we need to reach extremely high confidence that the core of the system is correct. We have found that the standard verification techniques in industry are necessary but not sufficient. We use deep design reviews, code reviews, static code analysis, stress testing, fault-injection testing, and many other techniques, but we still find that subtle bugs can hide in complex concurrent fault-tolerant systems. One reason for this problem is that human intuition is poor at estimating the true probability of supposedly 'extremely rare' combinations of events in systems operating at a scale of millions of requests per second.

> "To a first approximation, we can say that accidents are almost always th… of incorrect estimates of the likelihood of one or more things." - C. Michael Holloway, …

That human fallibility means that some of the more subtle, dangerous bugs turn out to be errors in

amazon web services™

## Applying TLA+ to some of our more complex systems

| System | Components | Line count (excl. comments) | Benefit |
|---|---|---|---|
| S3 | Fault-tolerant low-level network algorithm | 804 PlusCal | Found 2 bugs. Found further bugs in proposed optimizations. |
| | Background redistribution of data | 645 PlusCal | Found 1 bug, and found a bug in the first proposed fix. |
| DynamoDB | Replication & group-membership system | 939 TLA+ | Found 3 bugs, some requiring traces of 35 steps |
| EBS | Volume management | 102 PlusCal | Found 3 bugs. |
| Internal distributed lock manager | Lock-free data structure | 223 PlusCal | Improved confidence. Failed to find a liveness bug as we did not check liveness. |
| | Fault tolerant replication and reconfiguration algorithm | 318 TLA+ | Found 1 bug. Verified an aggressive optimization. |


fastly

Aphyr

Blog Photography Code

## Call me maybe: Elasticsearch


Previously, on Jepsen, we saw RabbitMQ throw away a staggering volume of data. In this post, we'll explore Elasticsearch's behavior under various types of network failure.

Elasticsearch is a distributed search engine, built around Apache Lucene–a well-respected Java

## Call me maybe: etcd and Consul

In the previous post, we discovered the potential for data loss in RabbitMQ clusters. In this oft-requested installation of the Jepsen series, we'll look at etcd: a new contender in the CP coordination service arena. We'll also discuss Consul's findings with Jepsen.

Like Zookeeper, etcd is designed to store small

## Call me maybe: RabbitMQ

RabbitMQ is a distributed message queue, and is probably the most popular open-source implementation of the AMQP messaging protocol. It supports a wealth of durability, routing, and fanout strategies, and combines excellent

## Computational techniques in Knossos

Earlier versions of Jepsen found glaring inconsistencies, but missed subtle ones. In particular, Jepsen was not well equipped to distinguish linearizable systems from sequentially or causally consistent ones. When people asked me to analyze systems which claimed to be linearizable, Jepsen could rule out obvious classes of behavior, like dropping writes, but

## Strong consistency models

*Network partitions are going to happen*. Switches, NICs, host hardware, operating systems, disks, virtualization layers, and language runtimes, not to mention program semantics themselves, all conspire to delay, drop, duplicate, or reorder our messages. In an uncertain world, we want our software to maintain some sense of *intuitive correctness*.

Well, obviously we want intuitive correctness. Do

## Call me maybe: Redis redux

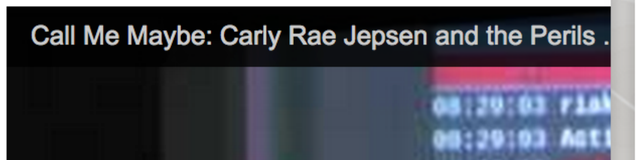
In a recent blog post, antirez detailed a new operation in Redis: `WAIT`. `WAIT` is proposed as an enhancement to Redis' replication protocol to reduce the window of data loss in replicated Redis systems; clients can block awaiting acknowledgement of a write to a given number of nodes (or time out if the given threshold is not met). The theory here is that positive acknowledgement of a write to a majority of nodes guarantees that write will be visible in all

## Call me maybe: Strangeloop Hangout



Sinc[illegible] the Strangeloop talks won't be available for a[illegible] ew[illegible] recorded a new version of the talk as a Google Hangout.

## Call me maybe: RICON East talk




## Call me maybe: Cassandra

*Previously on Jepsen, we learned about Kafka's proposed replication design.*

Cassandra is a Dynamo system; like Riak, it divides a hash ring into a several chunks, and keeps N replicas of each chunk on different

# Jepsen

# Network Partitions & DBs

# Hashicorp

Unit tests, acceptance tests, & integration tests

Shim out the network & introduce artificial partitions

Terraform spins up test cluster 20-100 nodes & more testing

Production verification & then release

Jepsen for Consul

fastly

Develop against LXCs (linux containers) to emulate our production architecture. **High setup cost**

Canary testing still falls short of giving us increased confidence

Run Jepsen for purging system

In a “seeking stage”

fastly / Avalanche

branch: master ▾ Avalanche / README.md

brucespang on May 6 initial commit

1 contributor

17 lines (10 sloc) | 0.504 kb

# Avalanche

Avalanche is a script that injects random, repeatable network faults on specific po
systems.

## Running

```
sudo ./avalanche
```

## Possible Faults:

By default, Avalanche inserts a fault with probability specified in settings.py (p_fau
the following faults is picked with the probability specified in the config file:

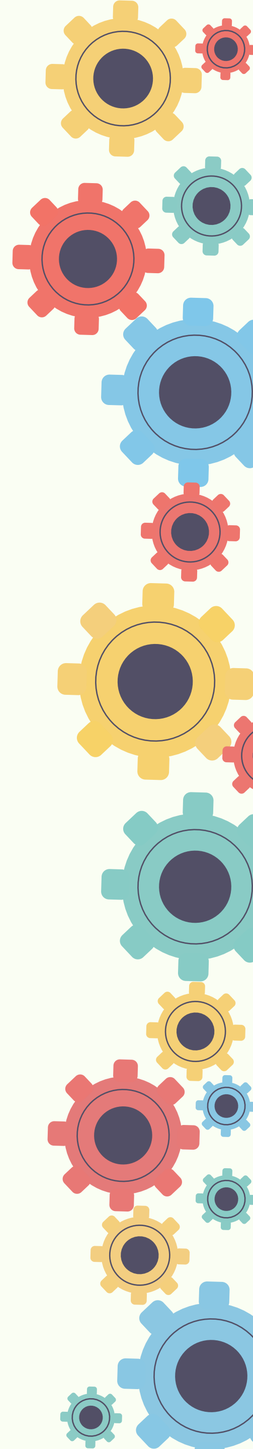
- High latency
- 100% packet loss
- Smaller percentage of packet loss
- Reorder packets

# DST IN THE WILD

Some patterns emerging

Best practices are still esoteric & ad-hoc

Still not as many tools as we should have (think compilers)



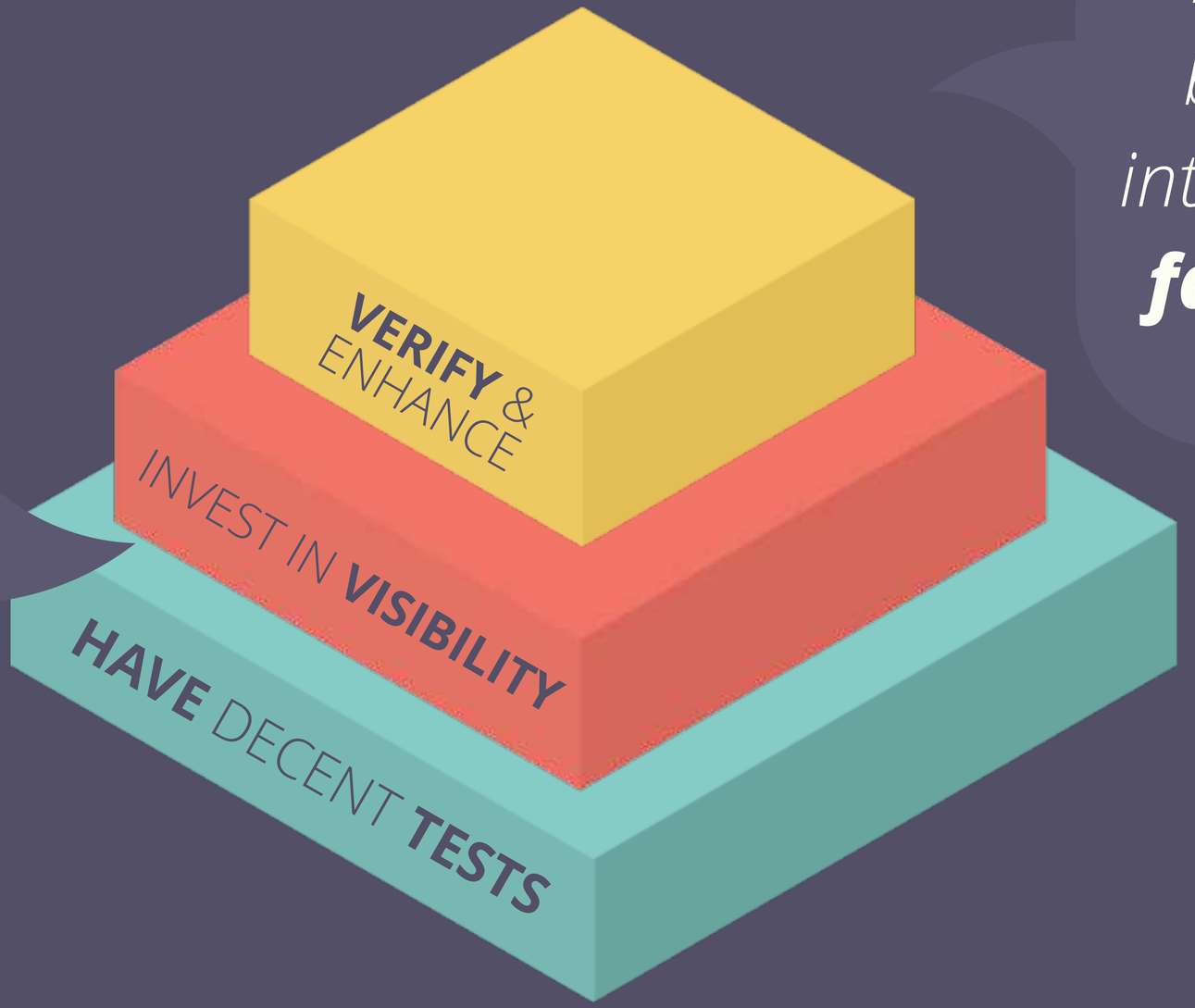
fastly

# LET'S BRING IT HOME

fastly®

# Have the basics covered

*Test the full distributed system. This means testing the* ***client***, ***system***, *AND* ***provisioning code***!



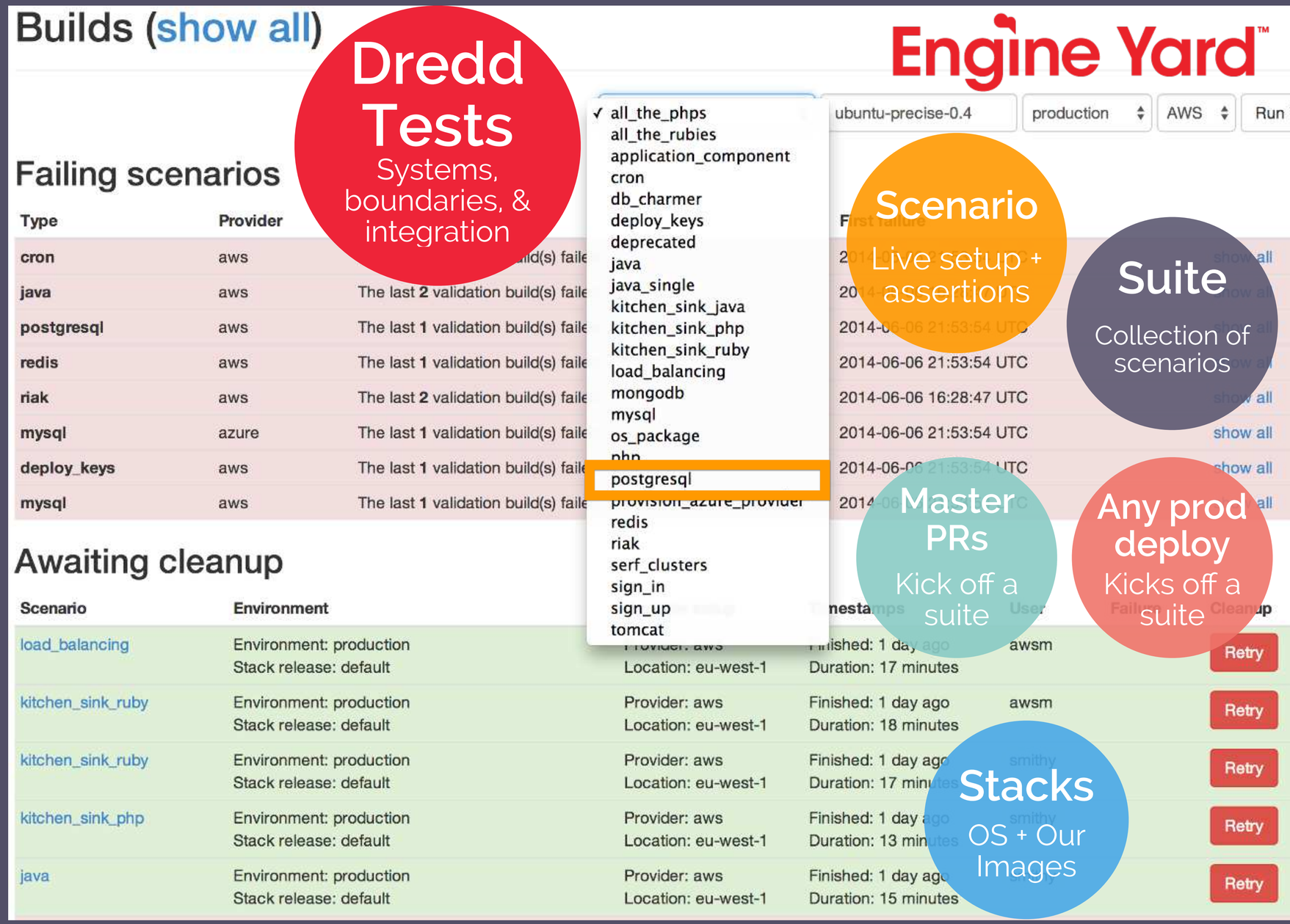

***Then*** *add behaviors, interactions, &* ***fancy stuff***

fastly

# INTEGRATION TESTS

- Run integration tests in EC2
- Mock services maybe
- Using different AMIs helped us a lot
- Can you spot the problem?

fastly



basho

GiddyUp | riak | riak_cs | riak_ee | smoke-tests

2.0.1 | 2.0.0 | 1.4.9 | 1.4.8 | 1.4.7 | 1.4.6 | 1.4.4 | 1.4.3 | 1.4.2 | 1.4.10 | 1.4.1 | 1.4.0

| Combined Coverage | centos-5-64 | centos-6-64 | fedora-17-64 | freebsd-9-64 | osx-64 | smartos-64 | solaris-10u9-64 | ubuntu-1004-64 |
|---|---|---|---|---|---|---|---|---|
| aae_test | U | U | U | U | U | | U | U |
| basic_command_line | U | U | U | U | U | | U | U |
| bucket_props_roundtrip | U | U | U | U | U | | U | U |
| bucket_types | L | L | L | L | L | | L | L |
| cluster_meta_basic | U | U | U | U | U | | U | U |
| cuttlefish_configuration | U | U | U | U | U | | U | U |
| ensemble_basic | U | U | U | U | U | | U | U |
| ensemble_basic2 | U | U | U | U | U | | U | U |
| ensemble_basic3 | U | U | U | U | U | | U | U |
| ensemble_basic4 | U | U | U | U | U | | U | U |
| ensemble_byzantine | U | U | U | U | U | | U | U |
| ensemble_interleave | U | U | U | U | U | | U | U |

# VISIBILITY

- Visual aggregation of test results
- Target architectures & legacy versions greatly increase state space
- Make errors a big deal
- Alerts & monitoring

fastly


# VISIBILITY & **LOGS**

Logs lie, are verbose, & mostly awful
BUT they are useful

**printf** is still widely used as debugging

Mind verbosity & configuration. Use different modes if tests repeatable

# VISIBILITY ACROSS SYSTEMS

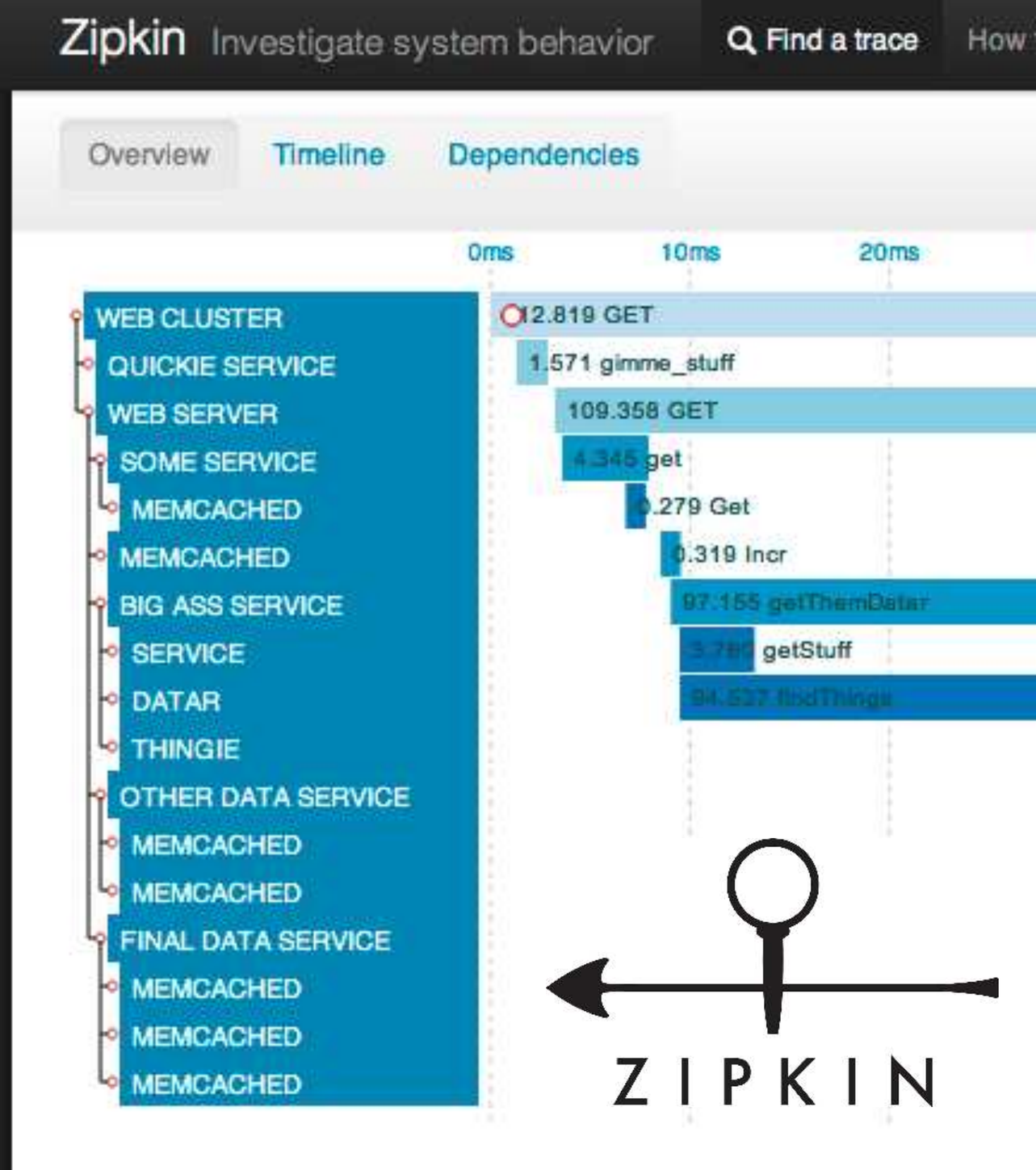
Insights into system interactions

Help visualizing dependencies before an API changes

Highlight request paths

Find the request_id

fastly



# ON ADDRESSING STATES

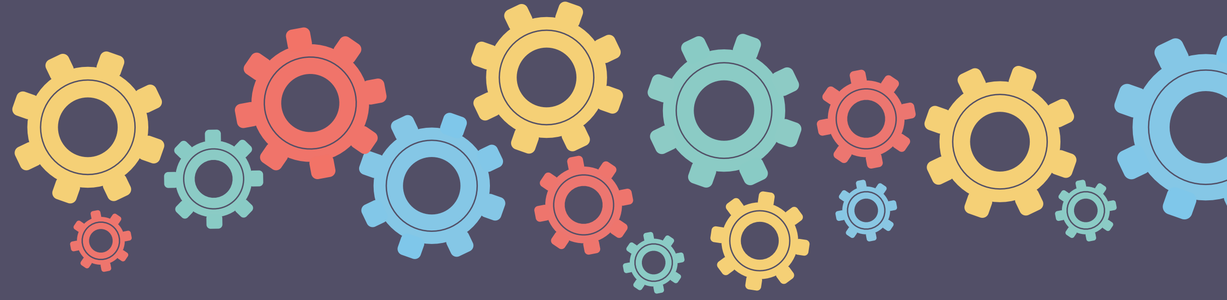
Test failure, recovery, start, & restarts (also missing dependencies)

Disk corruption happens: use **file checksums** to test to detect this

Shorter timeouts on leases in special nodes to prevent **clock drift**

fastly®

# CONFIGURATION & SETUP



Test provisioning code!

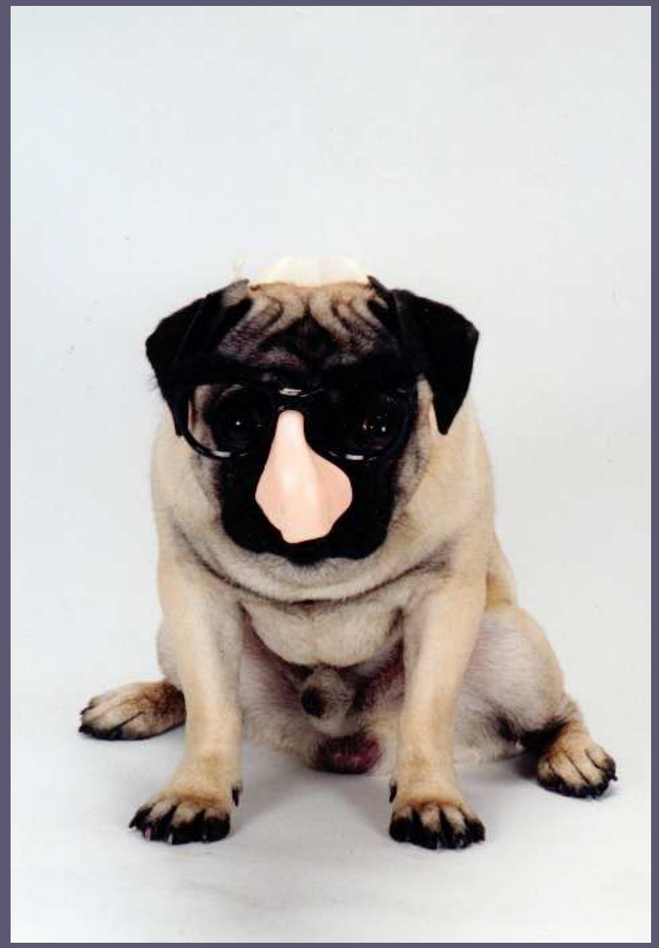
**Misconfiguration** is a common source of errors so test for bad configs

Remember OSDI paper: 3 nodes, 3 inputs + unit tests can reproduce 77% of failures

EC2 as real as it gets (Docker, LXCs too)

fastly

# ON TOOLS & FRAMEWORKS



Languages are starting to help us (go race detector)

Static analysis tools are more common

Fault injection frameworks are good but you still need understanding of **baseline behavior**

# TL;DR

## DST

Getting it right is tricky

Use multitude of methods to gain confidence

Value in testing

## ACADEMIA & INDUSTRY

**Formal Methods** when applied correctly tend to result in systems with **highest integrity**

Conventional testing is still our foundation

## WHAT YOU CAN DO **TODAY**

Test the **full system:** client, code, & provisioning

Increase tests investment as complexity increases.

Easy things don't cut it when you need certainty

Invest in **visibility** & understanding of behavior

Cost tradeoff present

fastly®

# Thank you!

*Special thanks to: Peter Alvaro, Kyle Kingsbury, Armon Dadgar, Sean Cribbs, Sargun Dhillon, Ryan Kennedy, Mike O'Neill, Thomas Mahoney, Eric Kustarz , Bruce Spang, Neha Narula, Zeeshan Lakhani, Camille Fournier, and Greg Bako.*

fastly®

https://github.com/Randommood/RICON2014

@randommood | ines@fastly.com


fastly®

github.com/Randommood/RICON2014