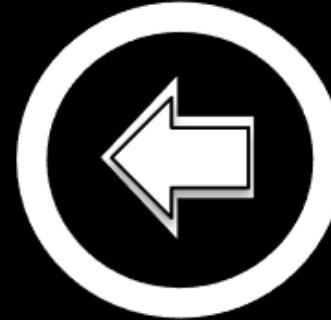


Antics, drift and chaos

Lorin Hochstein

Chaos Team, Netflix

@lhochstein



Whoops, something went wrong...

Netflix Streaming Error

We're having trouble playing this title right now. Please try again later or select a different title.



“I'm melting...melting....” by Derek Gavey is licensed under CC BY 2.0

Executes tests to warm

One day...

Add a new test

@Category(FunctionalTest.class)

Exception in thread "foo"
java.lang.ClassNotFoundException

Result: execution of unit test led to an outage

**Moral: use unit tests sparingly,
for they are dangerous**

**Moral: use unit tests sparingly,
for they are dangerous**

**Complex systems exhibit
unexpected behavior**

Failure

System failure

Outages

Incidents

LOW

W W W

h a v

What

Act I: Antics

Complex systems exhibit
unexpected behavior

— John Gall

Generalized Uncertainty Principle

Systems display antics
— John Gall

Our technologies have got ahead
of our theories

— Sidney Dekker

1. Error handling

Any large system is going to be
operating most of the time in
failure mode

— John Gall

Ding Yuan et al., Simple Testing Can Prevent Most Critical Failures: An Analysis of Production Failures in Distributed Data-Intensive Systems, OSDI 2014.

**Almost all catastrophic
failures (92%) are the result of
incorrect handling of non-fatal
errors explicitly signaled in
software**

Problems are not the problem;
coping is the problem.

— John Gall

C:\> DIR A:

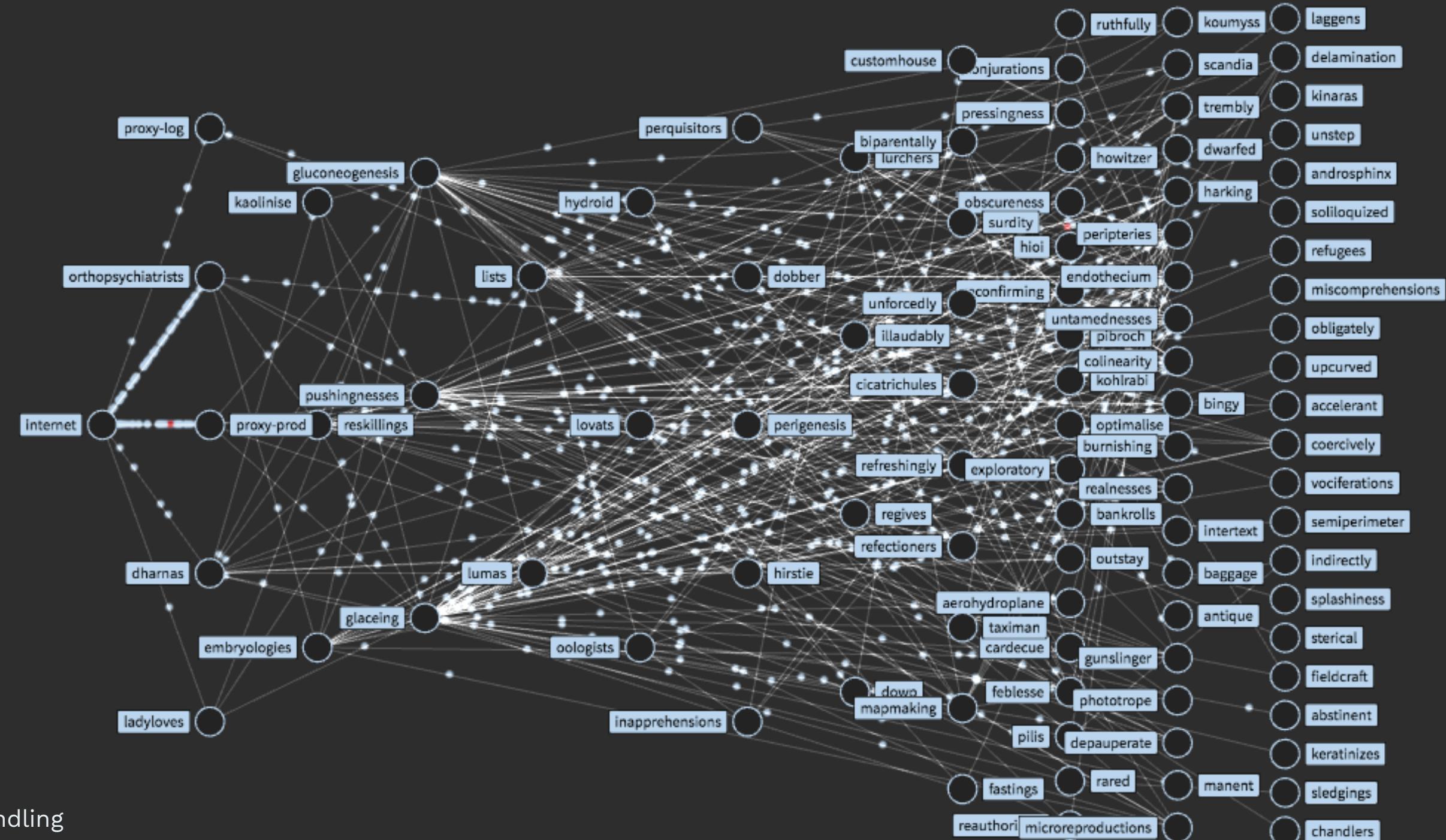
Not ready reading Drive A
Abort, Retry, Fail?

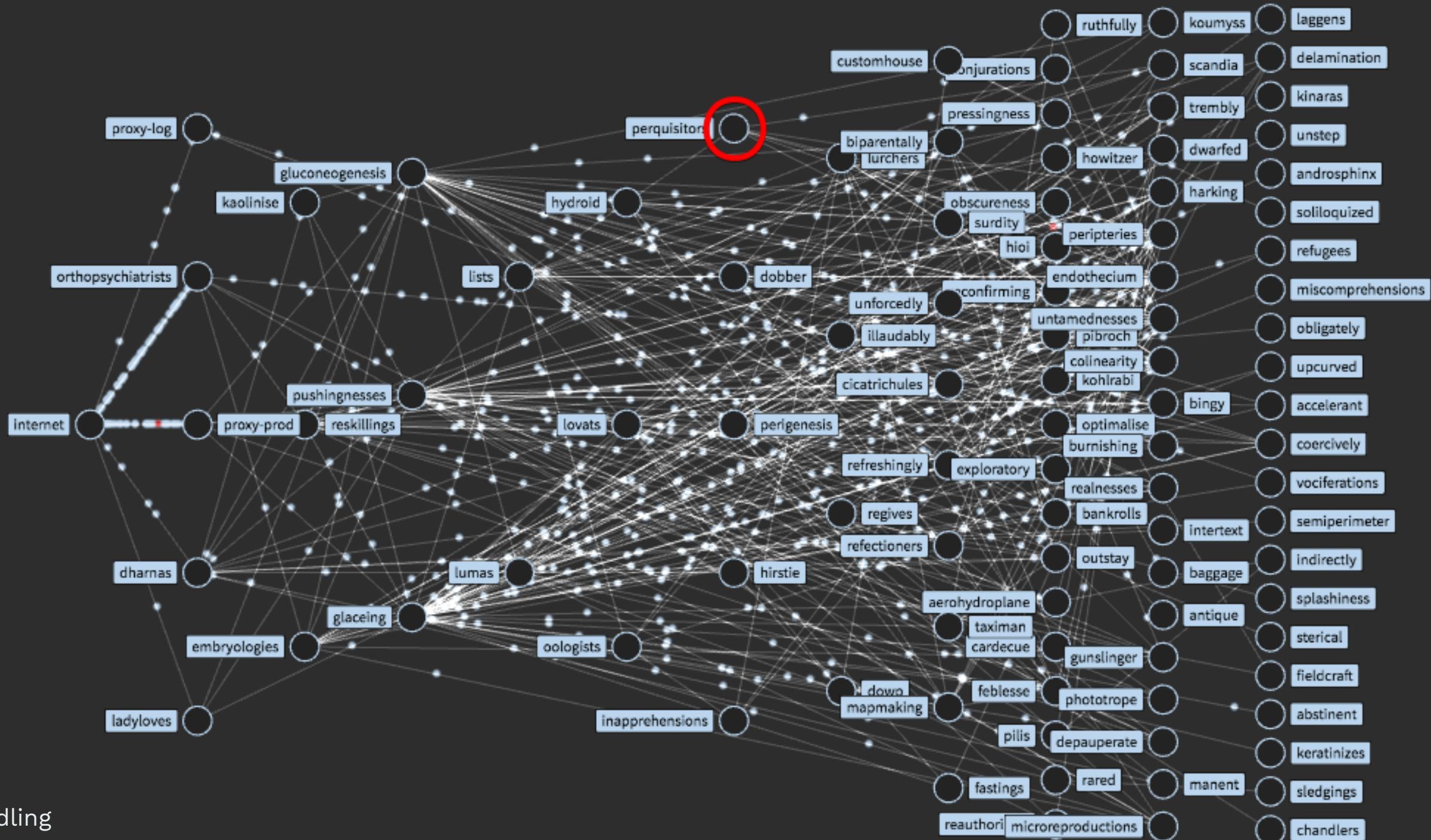
Scenario

global / us-east-1 ⚙

99 services / 2 filtered (show)

Locate Service





Service becomes latent

Clients timeout

Clients retry

Load increases

Latency increases

More clients retry



“Lightning Storm” by Benamen Benson is licensed under CC BY 2.0

The fail-safe theorem: when a fail-safe system fails, it fails by failing to fail safe

— John Gall

Zhenyu Guo et al., Failure Recovery: When the Cure Is Worse Than the Disease, HotOS 2013

Ryan Huang et al., Gray failure: the Achilles' heel of cloud-scale systems, HotOS 2017

2. Support systems

Why does Netflix need so many engineers?

Why does Netflix need so many engineers?

We're hiring! Come visit our booth!

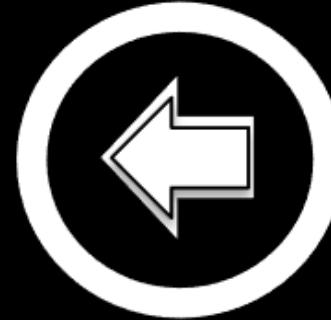
Operational fallacy: the system
itself does not do what it says

— John Gall

Harvard is really a \$40 billion tax-free hedge fund with a very large marketing and PR arm called Harvard University.

— Jim Manzi

Netflix is a monitoring company,
that as an interesting an
unexpected byproduct also
streams movies
— Adrian Cockcroft (attributed)



Whoops, something went wrong...

Netflix Streaming Error

We're having trouble playing this title right now. Please try again later or select a different title.

Antics > Support systems

Non-critical service failed

Log messages increased

Log messages sent to Kafka

Lock shared by app threads

Lock contention



“indy airshow explosion” by Paul J Everett is licensed under CC BY 2.0

Logging took down prod

AWS EBS Outage of Oct 2012

Memory leak bug in agent that monitors health of EBS servers

3. Mitigation

AWS S3 Outage of Feb 2017

S3 billing process was slow

AWS eng tried to remove some servers

Command input entered incorrectly



“by Darwin Yanque Cutipa 09” by Flickr user cticona is licensed under CC BY 2.0

Lorin's conjecture

Most major incidents will be due to

1. Unexpected behavior of a support system
2. Attempt to mitigate a non-critical incident

Recap: Antics

Antics

**Mechanisms that improve
availability (error-handling,
support systems, mitigation)
also create outages**

Act II: Drift



by Diego Delso, licensed under CC BY-SA 3.0 via Wikimedia Commons

Broken parts and sloppy devs

"Be more careful"

Drift into failure

1. Unruly technology

Software is hard to reason about

We can't model our systems

Fault-tolerance isn't composable.
— Peter Alvaro

The mode of failure of a complex
system cannot ordinarily be
determined from its structure

— John Gall

A system can fail in an infinite
number of ways

— John Gall

Pedro Fonseca et al., An Empirical Study on the
Correctness of Formally Verified Distributed Systems,
EuroSys 2017.

Formally verified component

Shim layer

Operating system

Formally verified component

Shim layer

<---- Most bugs are here

Operating system

2. Scarcity and competition

Efficiency vs thoroughness

Drift > Scarcity and competition

ETTO Principle

— Erik Hollnagel

A temporary patch will very likely
be permanent

— John Gall

3. Decrementalism

Drift happens in small steps

When is it OK to push to prod?



by Flickr user stevepj2009 licensed under CC BY 2.0

Failed canary probably OK if actual code change looks
harmless



0operator

@sadoperator

Following



me: this change is pretty small, should be fine to deploy to production

narrator: but the change was not fine to deploy to production

12:40 PM - 25 Sep 2017

Normalization of deviance

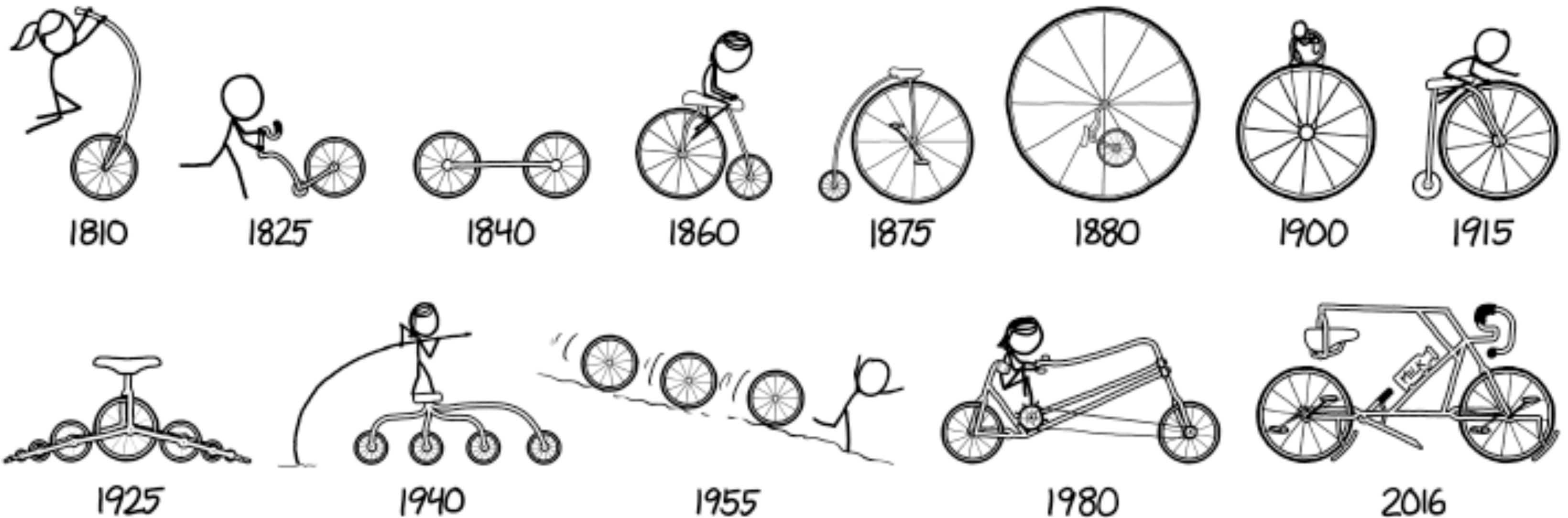
— Diane Vaughan

4. Sensitive dependence on initial conditions



by Flickr user skippy licensed under CC BY-SA 2.0

TIMELINE OF BICYCLE DESIGN



A complex system that works is
inevitably found to have evolved
from a simple system that worked

— John Gall

We make local decisions that have non-local impact



Drift > Sensitive dependence on initial conditions



Drift > Sensitive dependence on initial conditions



If URLs fails,
Playback has a fallback

One day...

traffic spike

Drift > Sensitive dependence on initial conditions



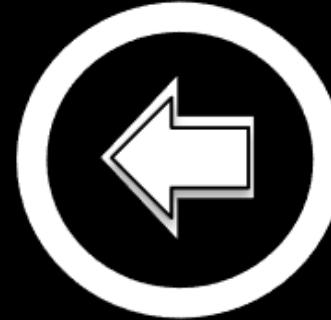
Drift > Sensitive dependence on initial conditions



- EVcache client in URLs treats timeout as **cache miss**



- EVcache client in URLs treats timeout as **cache miss**
- Playback **can't handle** missing data scenario



Whoops, something went wrong...

Netflix Streaming Error

We're having trouble playing this title right now. Please try again later or select a different title.

EVcache client defaults to treating errors as cache misses

That's the correct behavior in most usages

Structural secrecy

— Diane Vaughan

Recap: Drift

Drift

The nature of software, how people behave under resource constraints, how people make local decisions, and history all contribute to system failure

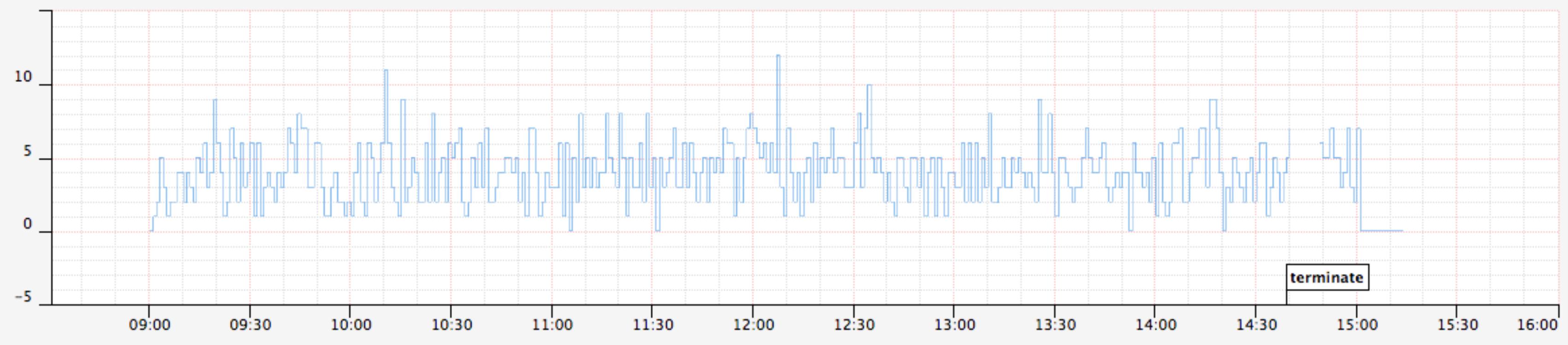
Act III: chaos



PROD:US-WEST-2



Terminations



Chaos > Chaos Monkey

Make the wrong thing harder

Chaos engineering



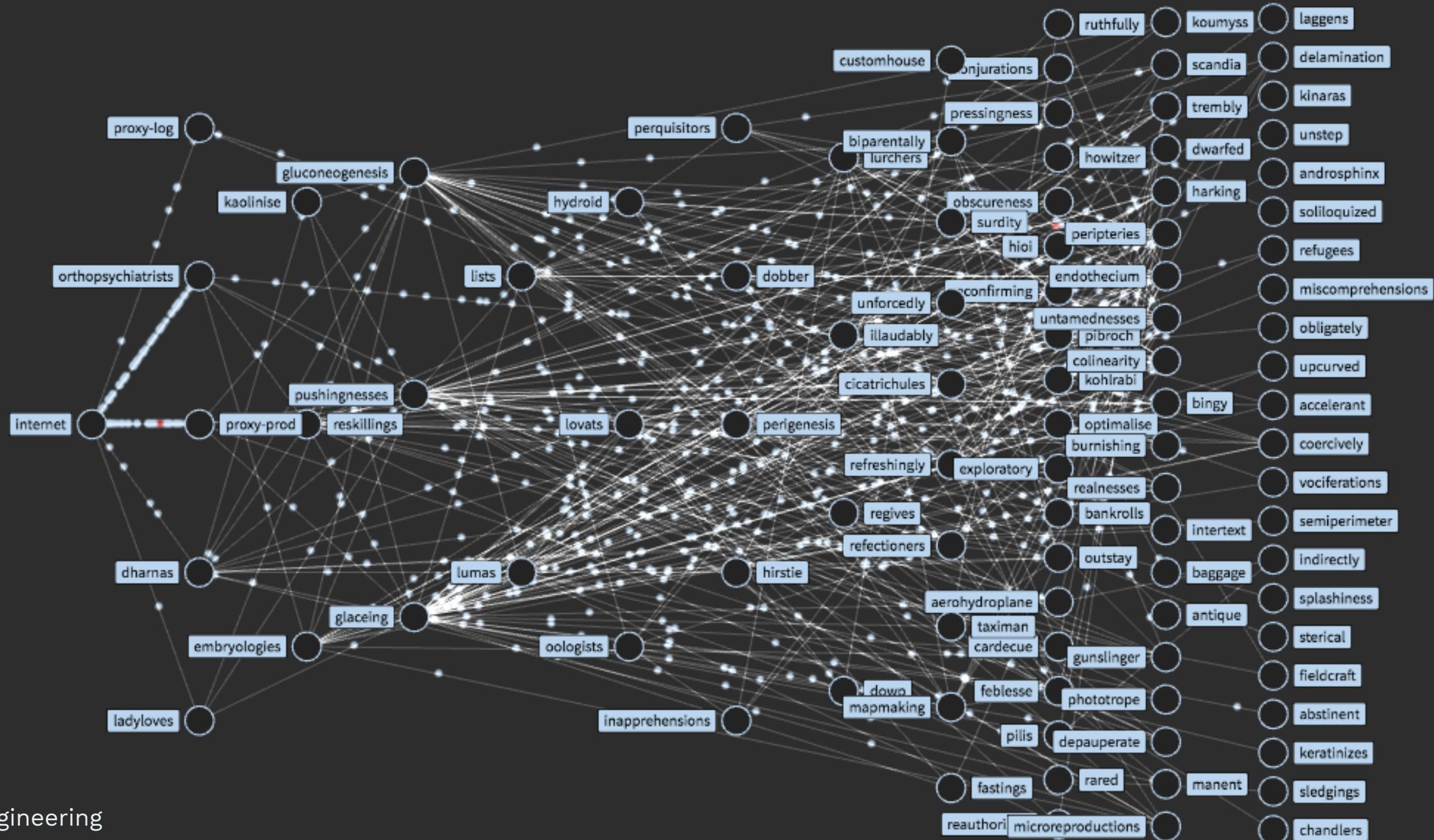
Flickr user blondyimp licensed under CC BY 2.0

Find vulnerabilities before they become outages

In production

External validity

Risk: vulnerable to failure of non-critical services

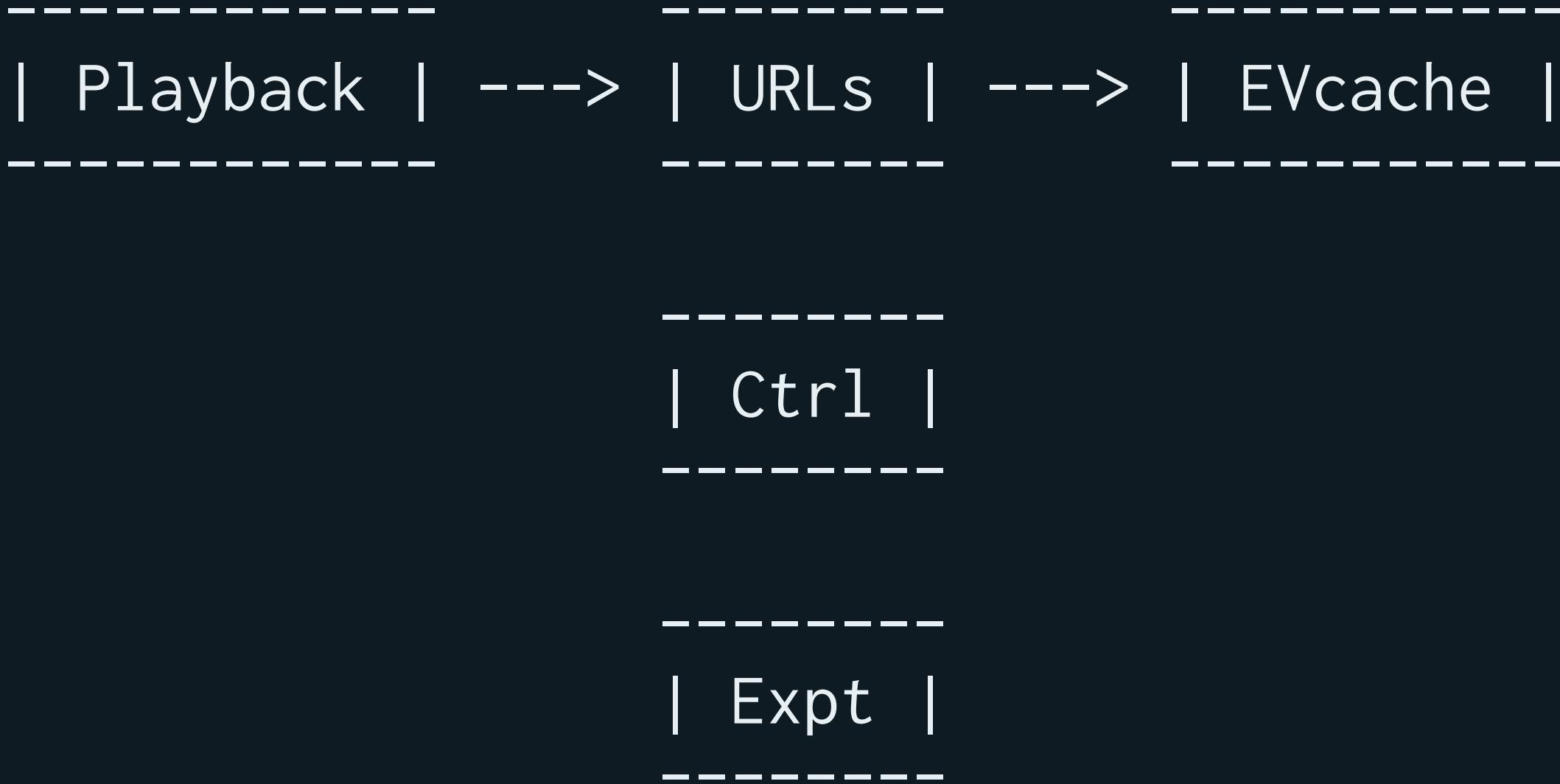








Are we vulnerable to EVcache timeouts?



1. Clone URLs cluster to make two smaller clusters



2. Route fraction of prod traffic to control and experiment clusters



3. Inject latency in calls from experiment cluster to EVcache

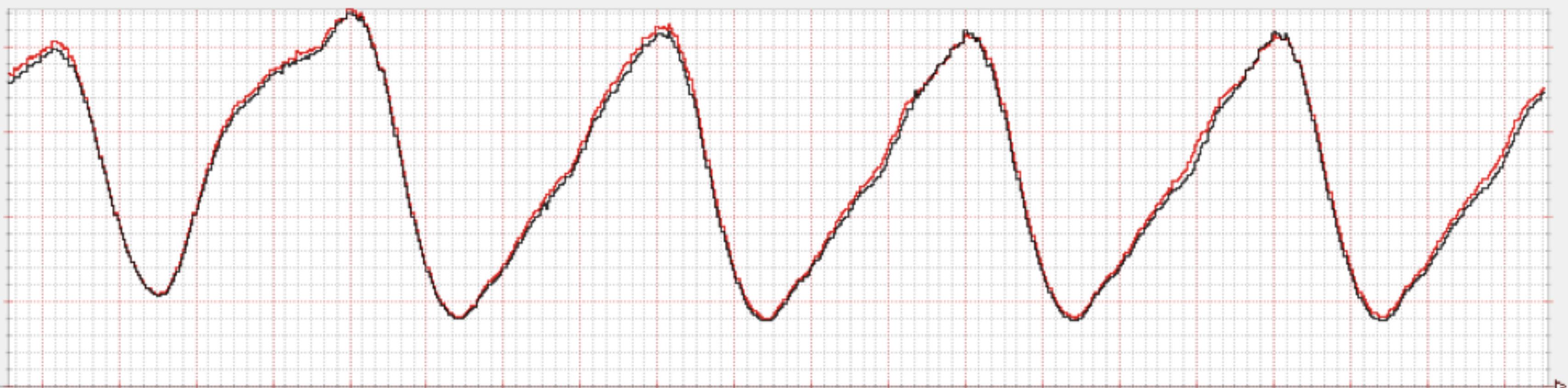


4. Measure differences between control & experiment clusters

principlesofchaos.org

1. Build a hypothesis around steady state behavior

SPS



2. Vary real-world events

Fail RPC calls

Add latency to RPC calls

3. Run experiments in production

Route prod traffic to chAP clusters

4. Automate experiments to run continuously

Integrate with deployment pipelines

5. Minimize blast radius

Route a small fraction of traffic

Stop early if impact detected

takeaways

1. Systems behave pathologically

Chaos experiments can find pathologies

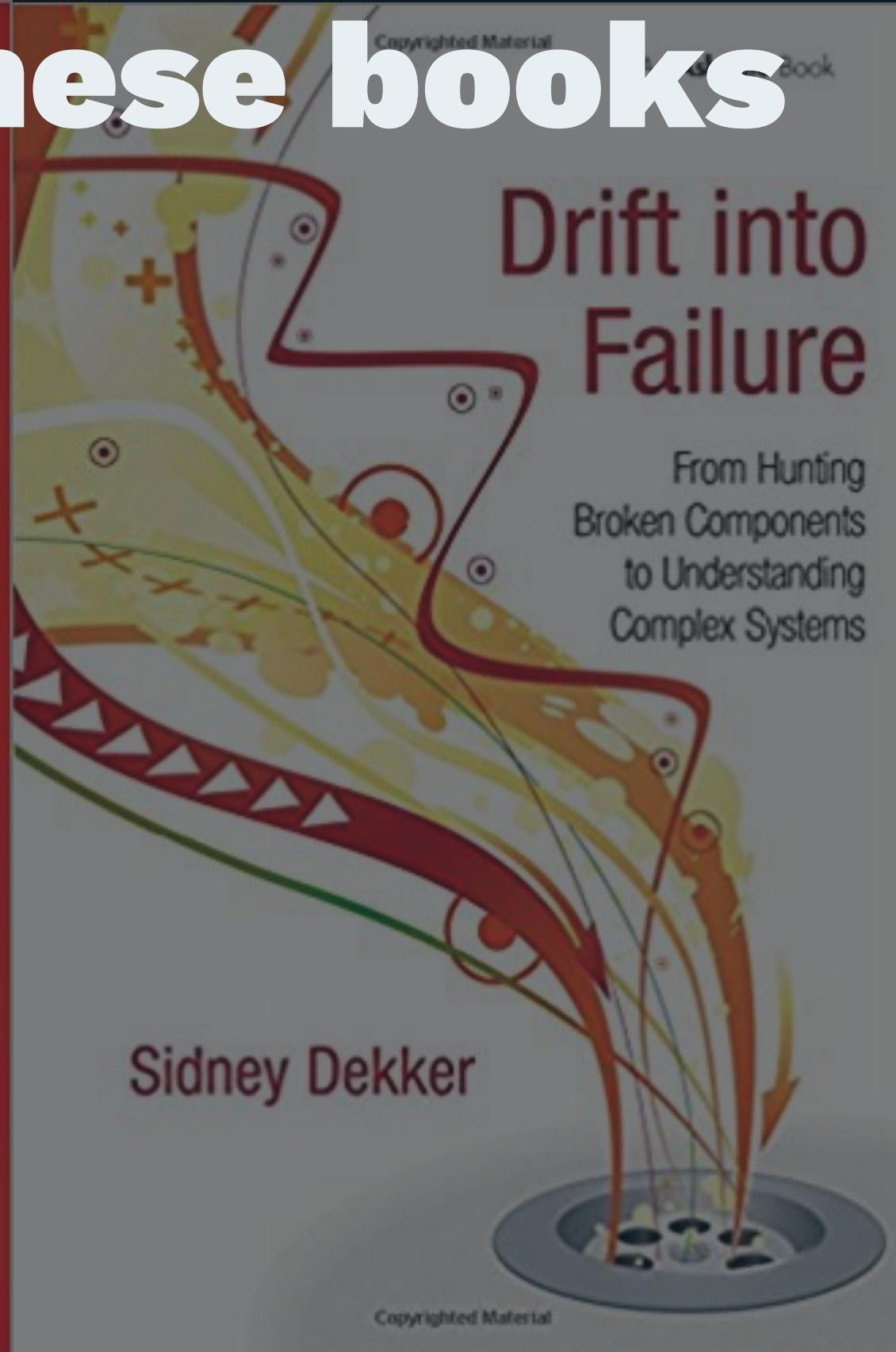
Takeaways

2. Reasonable human decisions can lead to dangerous states

Chaos provides incentives

Takeaways

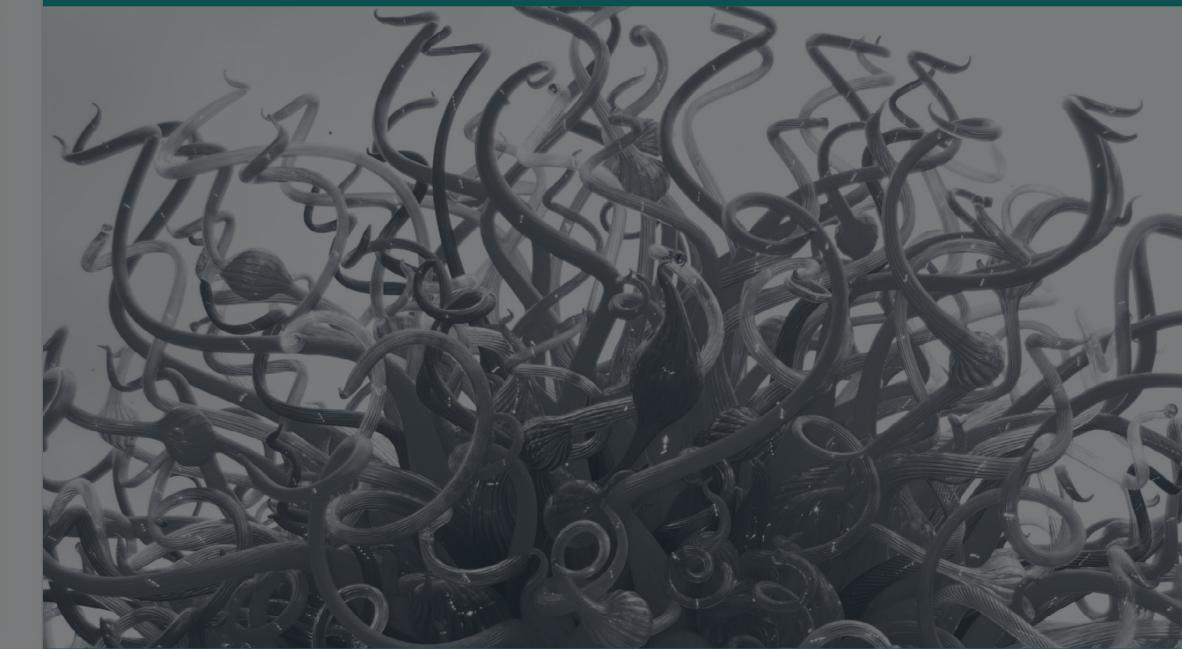
3. Read these books



O'REILLY®

Chaos Engineering

Building Confidence in System Behavior through Experiments



Casey Rosenthal, Lorin Hochstein,
Aaron Blohowiak, Nora Jones
& Ali Basiri