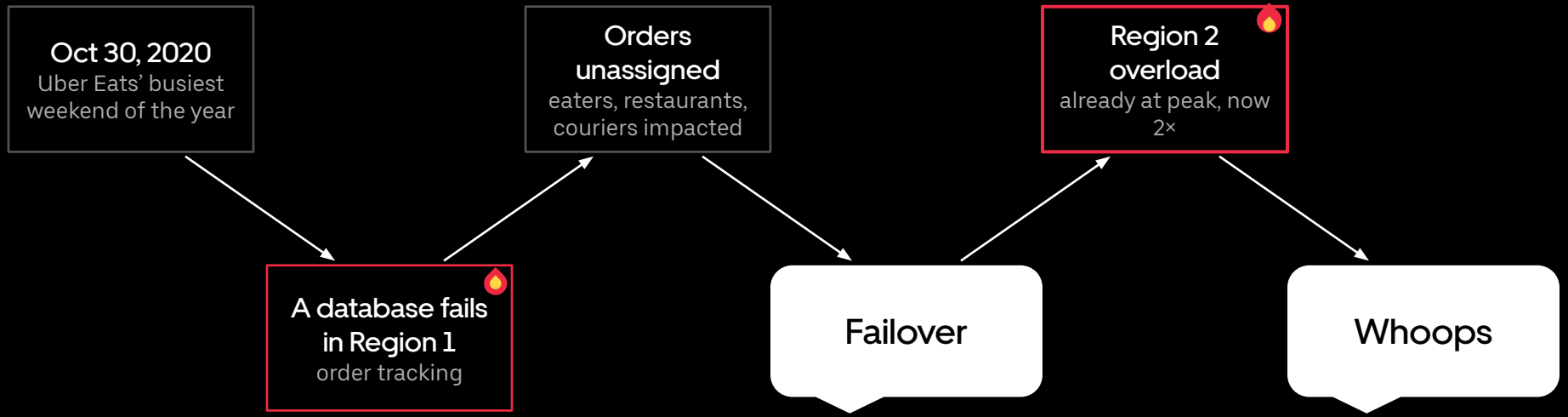
How Uber Validates Failover Resilience in Production

Johan Mena, Albert Wang
Production Engineering
Uber

A Halloween story





Johan Mena

Staff Software Engineer



Albert Wang

Senior Staff Tech Lead Manager

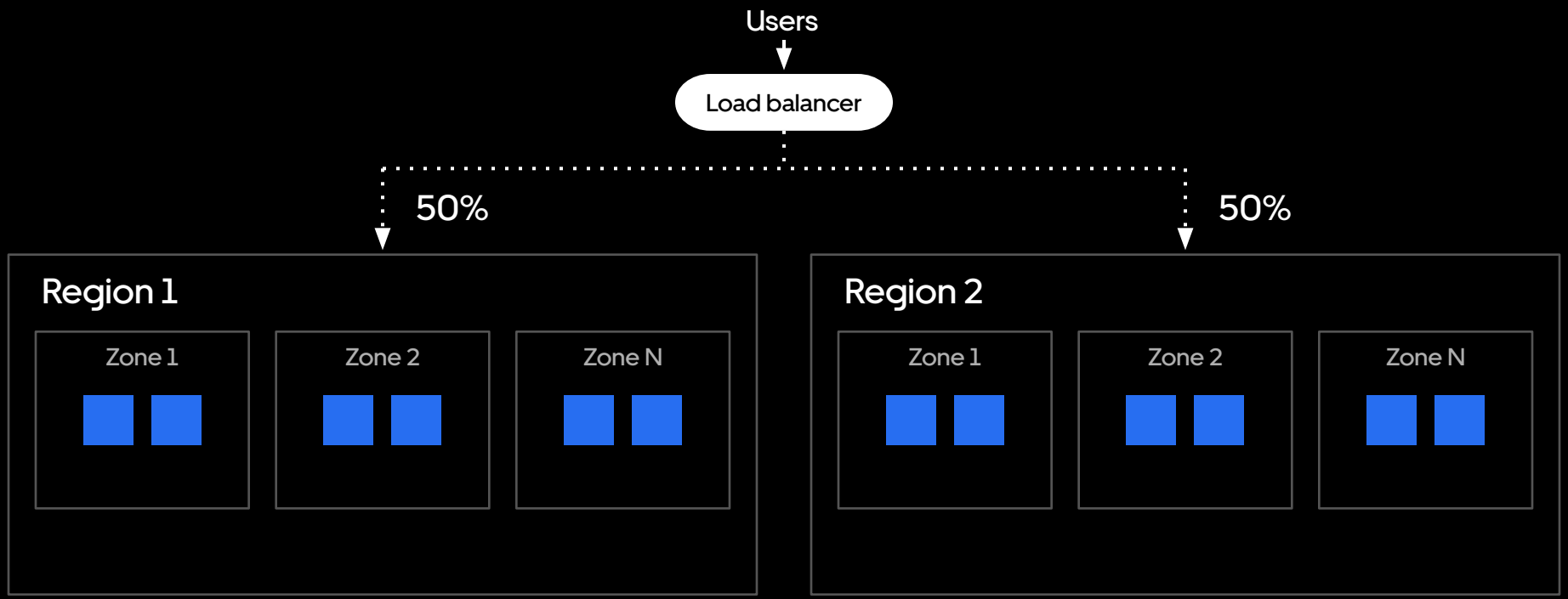
Production Engineering @ Uber

The best environment to validate failovers is production.
The only catch is, it's production.

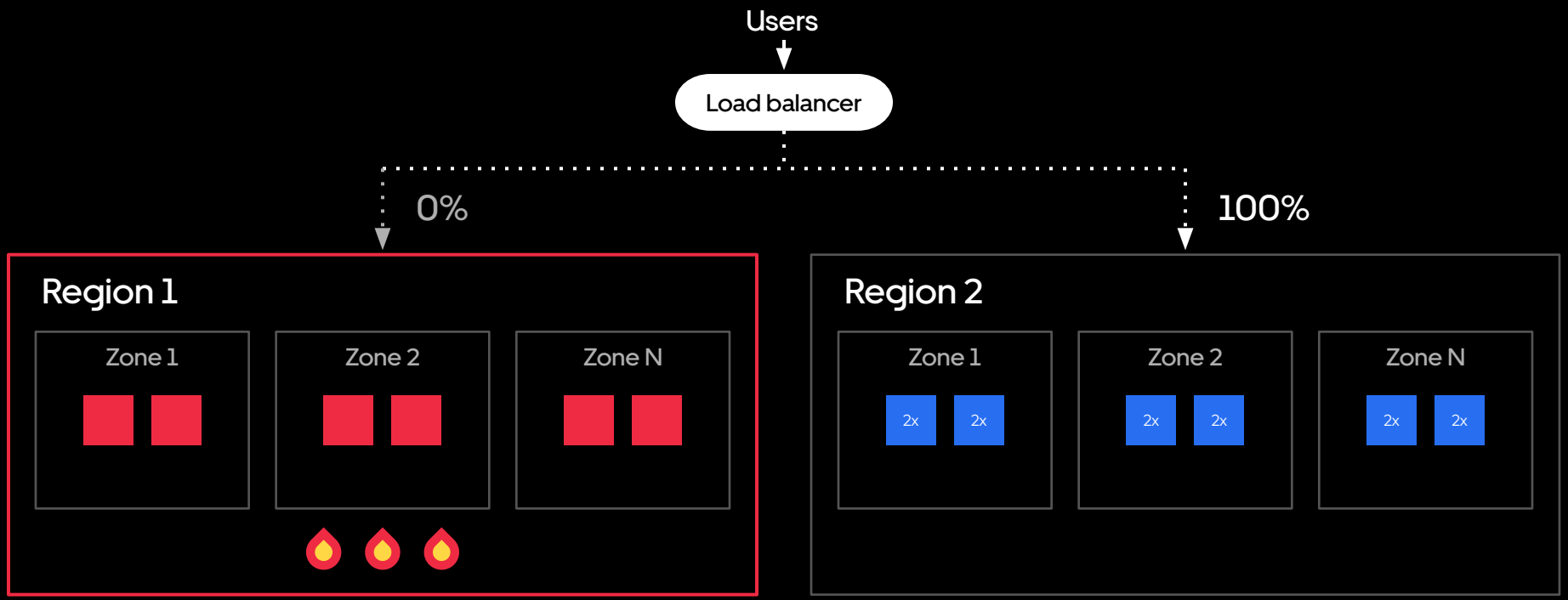
01 The failover problem
02 Load testing in production
03 Muttloader: shifting traffic on autopilot
04 Muttloader: keeping production safe
05 Impact, challenges & lessons

01 The failover problem
02 Load testing in production
03 Muttloader: shifting traffic on autopilot
04 Muttloader: keeping production safe
05 Impact, challenges & lessons

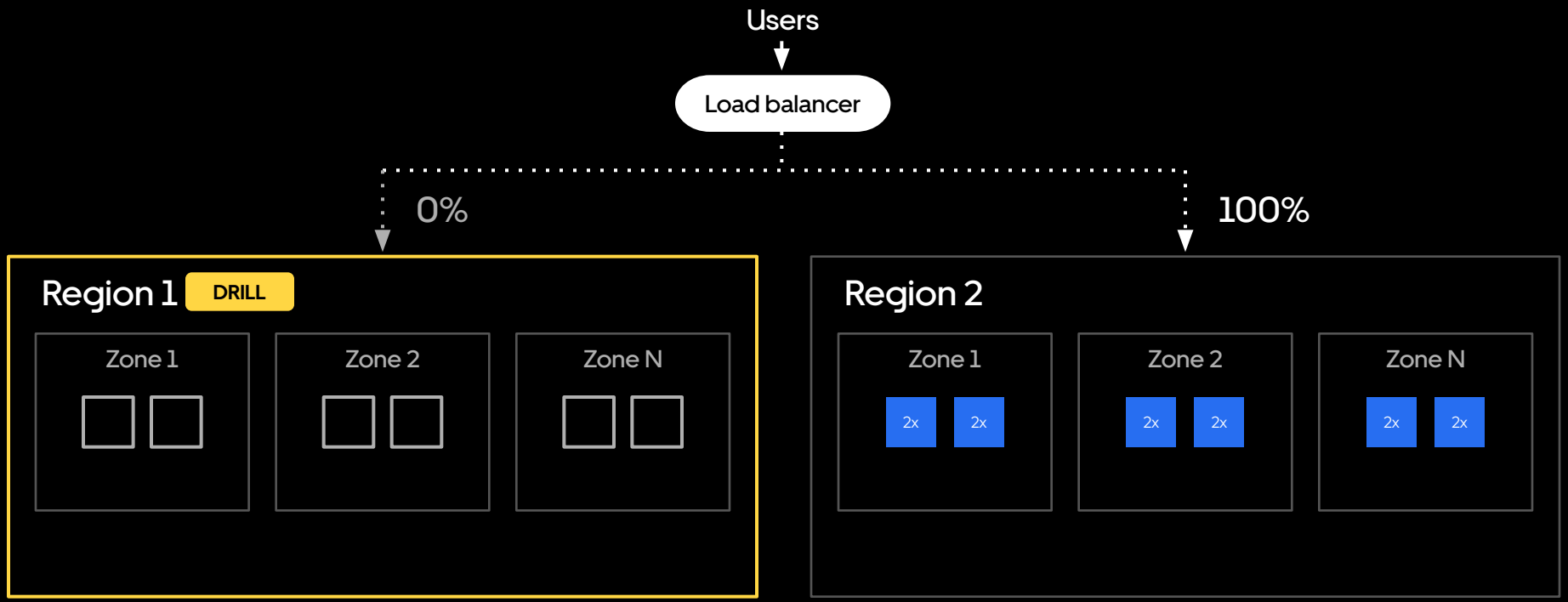
Uber's architecture



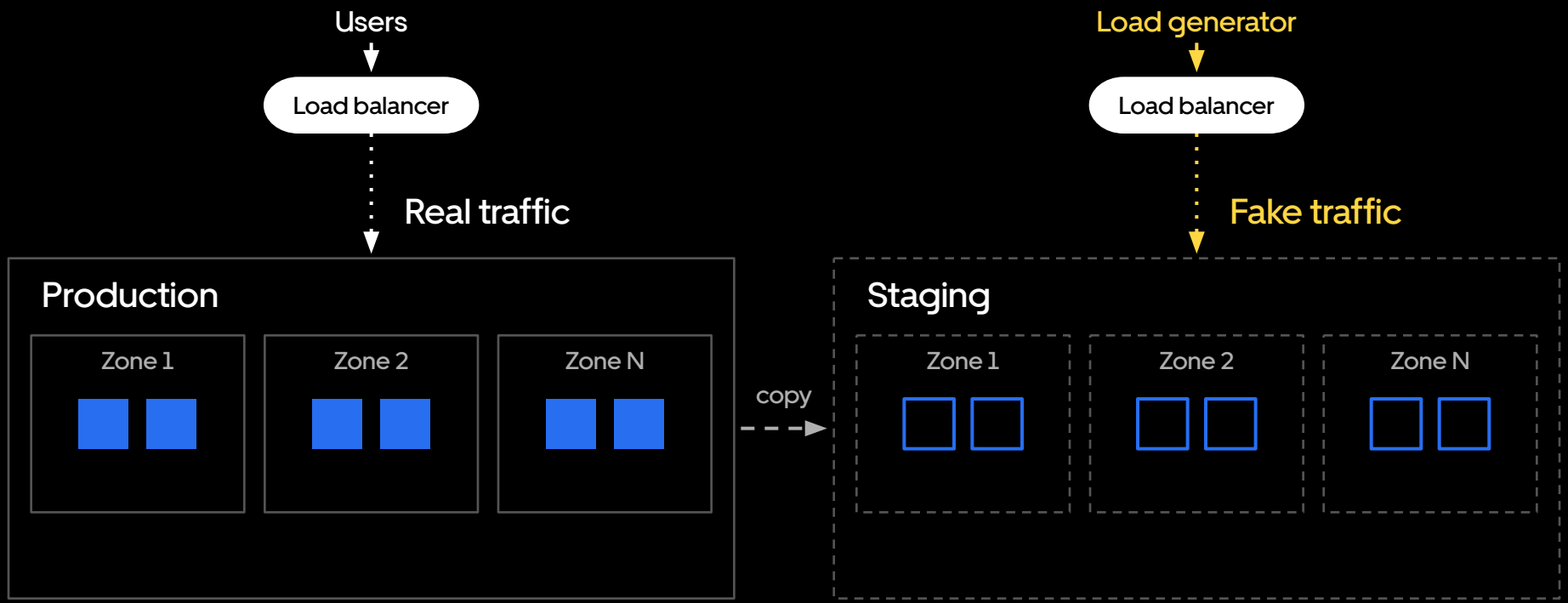
Regional failover



Production failover drills



Synthetic load tests



Testing failovers

Production failover drills

High risk

Operationally intensive (running drills)

High-fidelity results

Low compute cost

Synthetic load tests

Low risk

Operationally intensive (maintenance)

Low-fidelity results

High compute cost

01 The failover problem

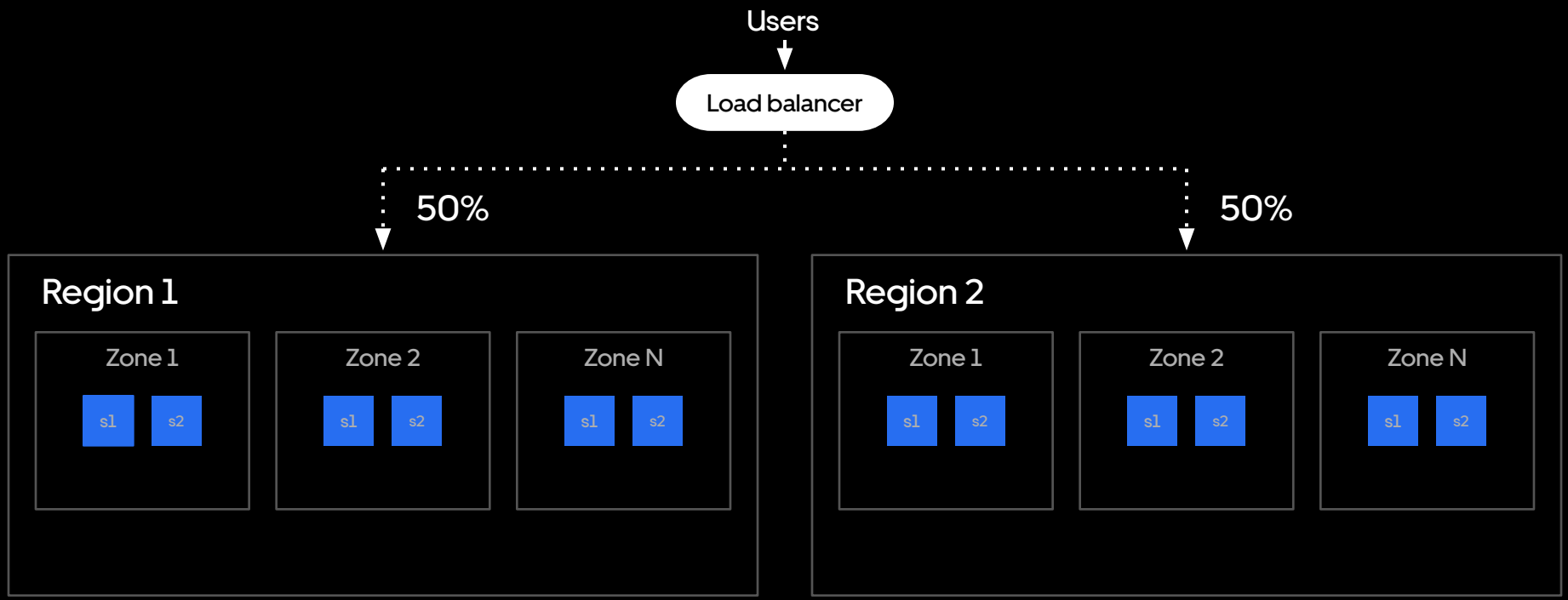
02 Load testing in production

03 Muttloader: shifting traffic on autopilot

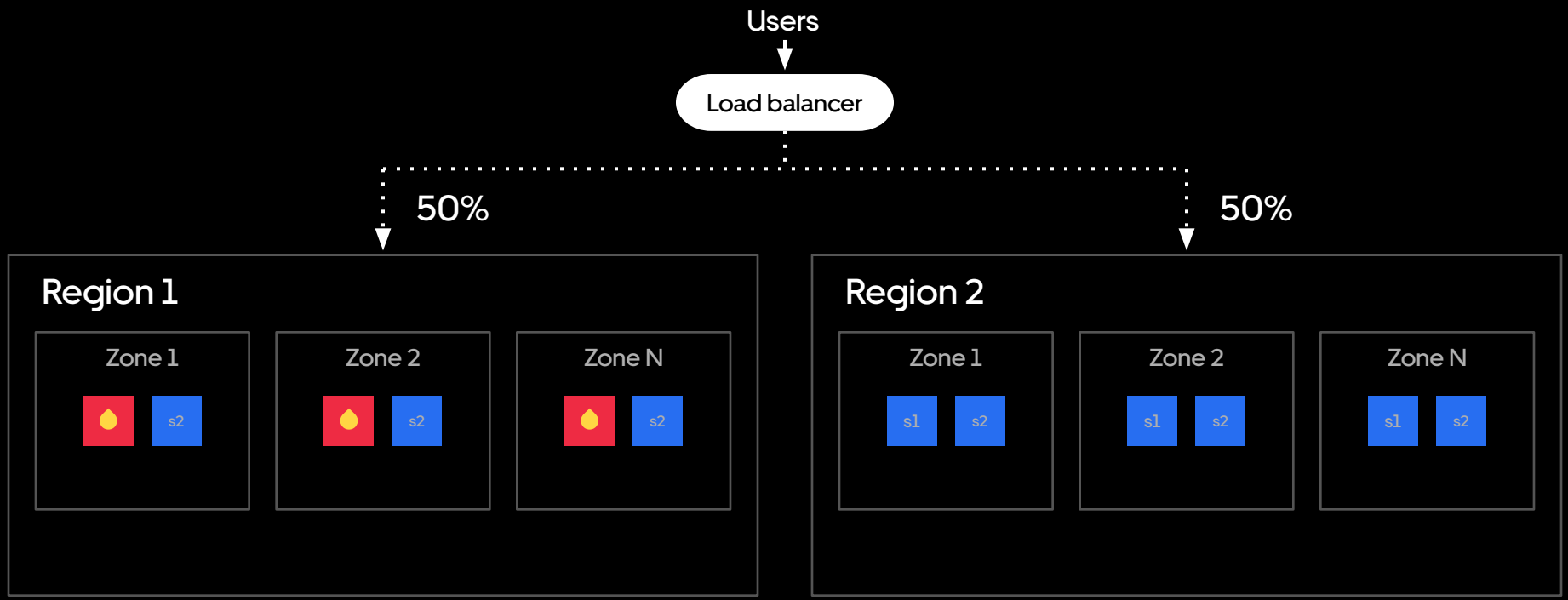
04 Muttloader: keeping production safe

05 Impact, challenges & lessons

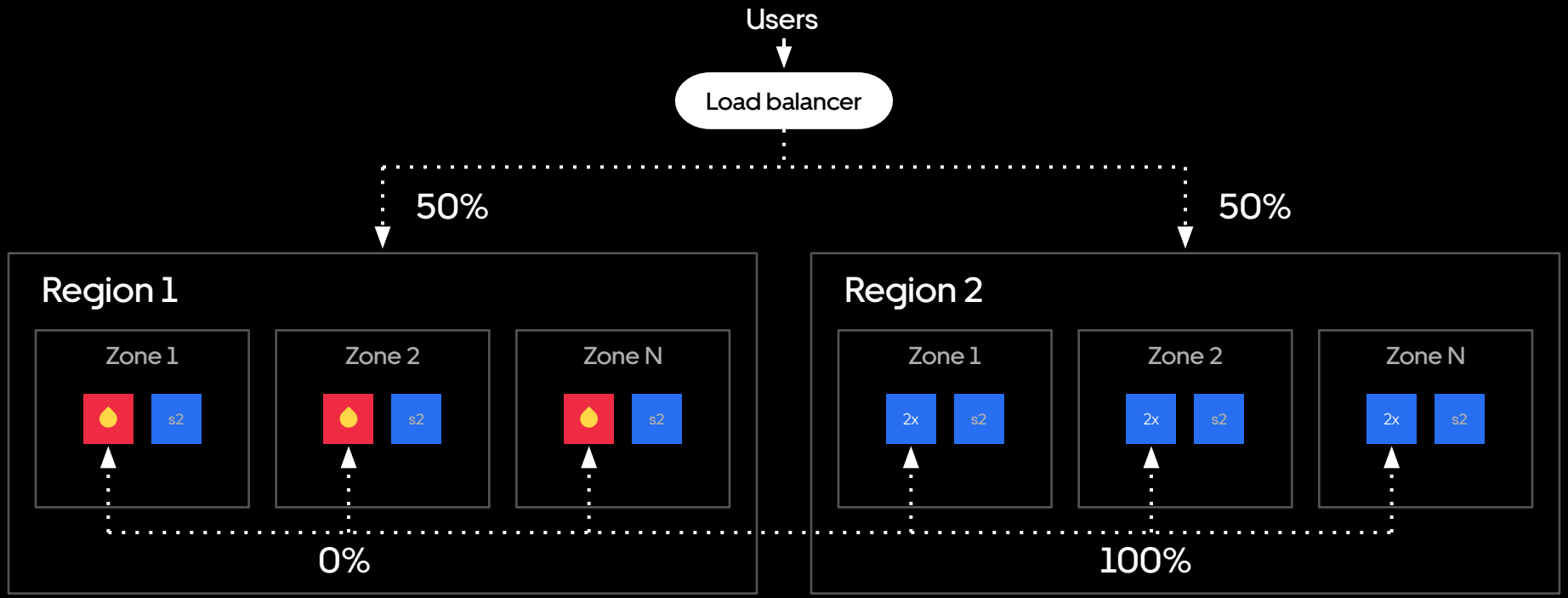
Uber's architecture



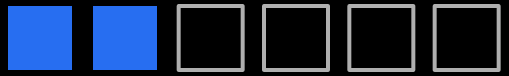
Issue: s1 fails in Region 1



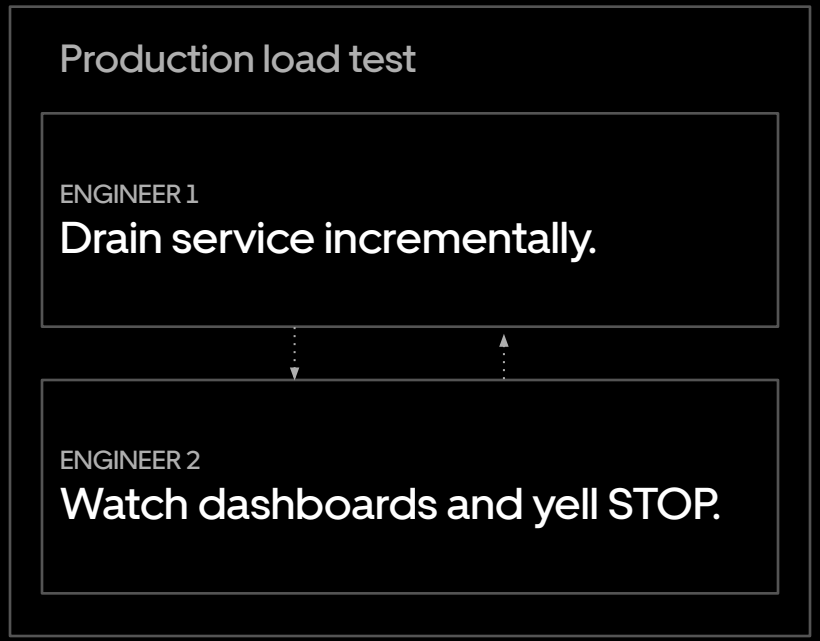
Mitigation: drain s1 from Region 1



Less capacity → more load

6 REPLICAS (■ active, □ inactive)	INACTIVE	LOAD
	1	1.2×
	2	1.5×
	3	2×
	4	3×
	5	6×

Manual load testing in production



TOIL

slow

1 hour per test

some services

only what we could afford

inconsistent

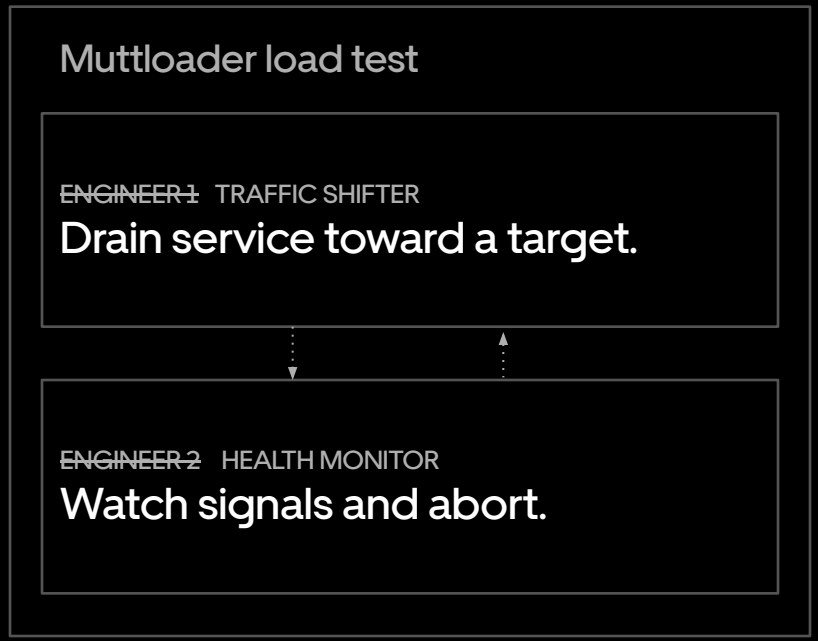
different engineer, different test parameters

hard to scale

product teams had to learn infra

01 The failover problem
02 Load testing in production
03 Muttloader: shifting traffic on autopilot
04 Muttloader: keeping production safe
05 Impact, challenges & lessons

Reducing cost of manual load testing



AUTOMATION

fast

no engineer time per test

all services

every eligible service, every week

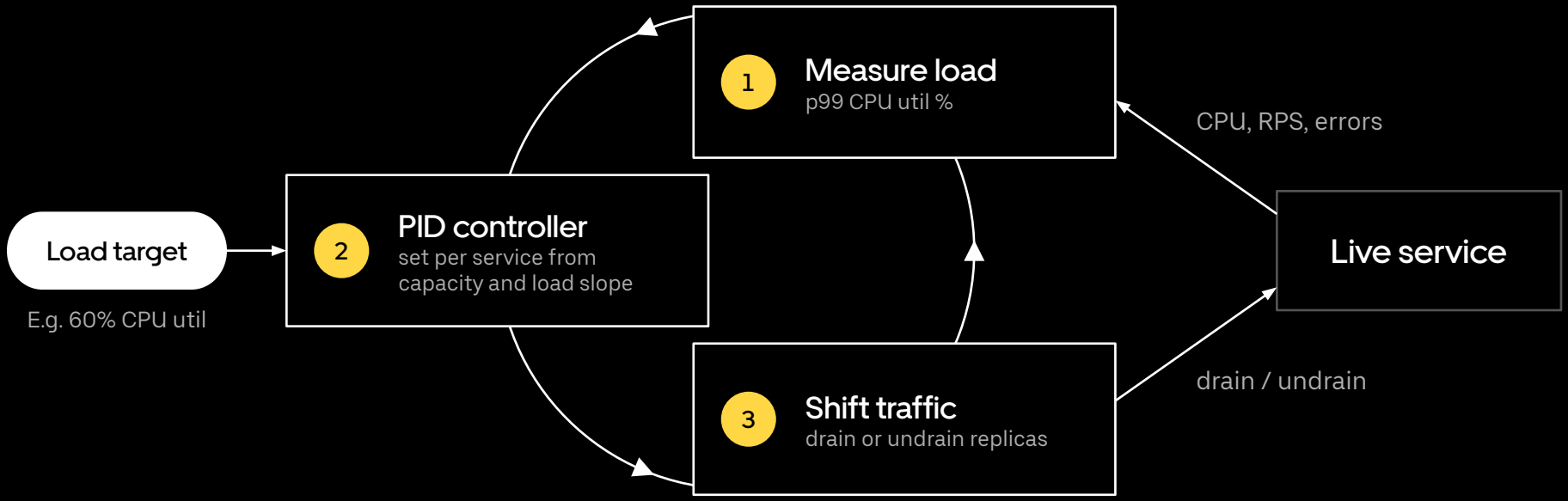
consistent

same algorithm, same thresholds

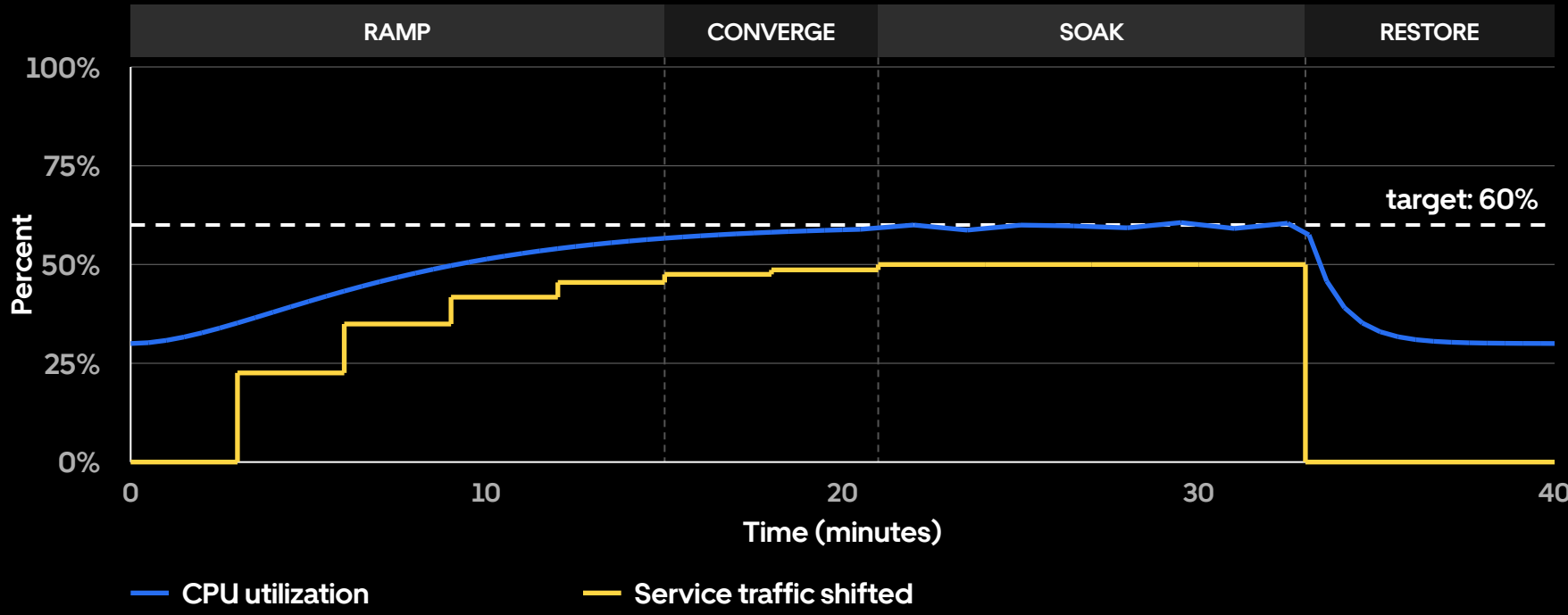
easy to scale

one API call with safe defaults

Muttloader: a closed control loop



What a test looks like



Muttloader: automated load tests in prod

Simple
set a target; the system does the rest

High fidelity
behaves as production

Zero setup
no tests to write or maintain

Frequent
runs as often as you want

Small blast radius
one service at a time

Evergreen
tests are always up to date

01 The failover problem
02 Load testing in production
03 Muttloader: shifting traffic on autopilot
04 Muttloader: keeping production safe
05 Impact, challenges & lessons

Safety is the product

Eligibility validation

only test services we can keep safe based on real-time metrics

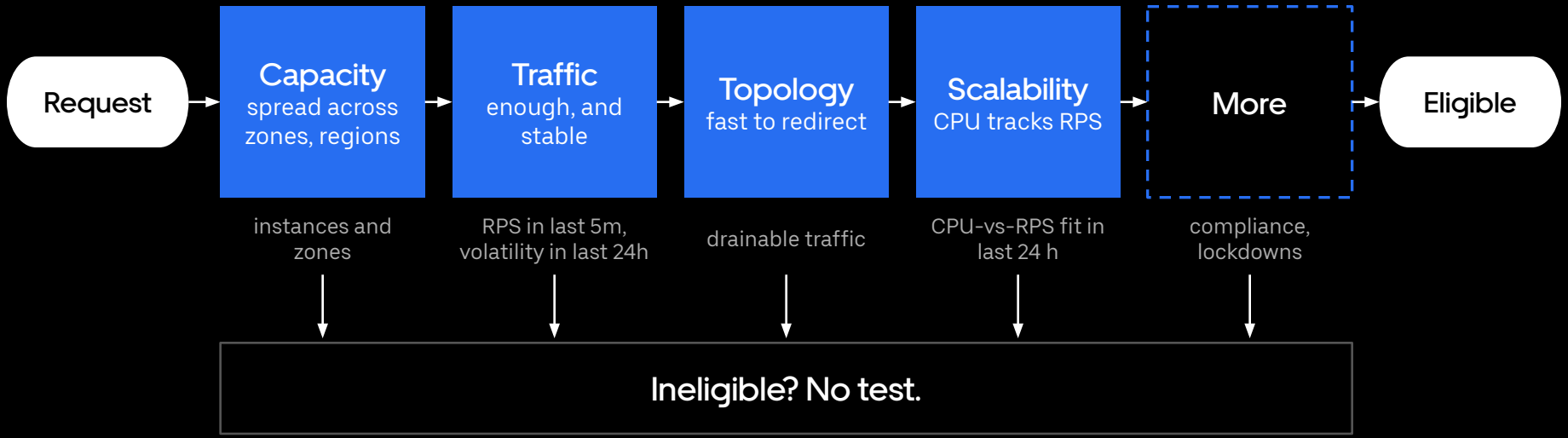
Health monitoring

monitor application, resources, and system metrics

Platform safeguards

protect production from Muttloader itself

Eligibility: checked before every test



Health monitoring: checked continuously

Application metrics

the service's own endpoint, business, and internal alerts/SLAs

Resource metrics

CPU, memory, latencies compared against known limits or pre-test baselines

System metrics

unexpected process exits, OOMs

Platform conditions

availability of critical dependencies; network degradations; active incidents

Key overload signal: request queue latency

What it is

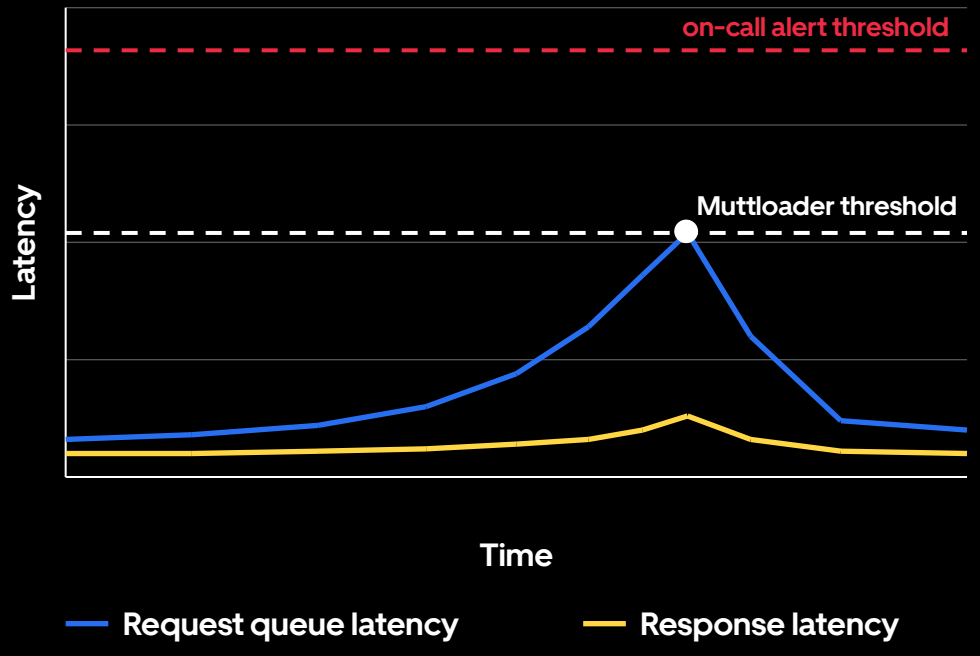
wait time in the inbound request queue

Why it matters

rises before latency and errors

How we use it

compare p99.9 over 2m vs a fixed threshold



Platform safeguards

Rollout gates

Every change load-tests our own services first.

PREVENTS

a bad change reaching production

Criticality environments

low-prio → bake → high-prio

PREVENTS

a new bug on a critical service

Operation expiration

Operations automatically removed or we get paged.

PREVENTS

a permanent load test

01 The failover problem
02 Load testing in production
03 Muttloader: shifting traffic on autopilot
04 Muttloader: keeping production safe
05 Impact, challenges & lessons

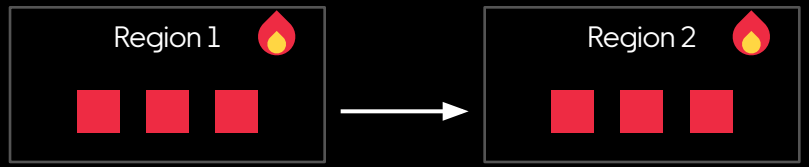
Same failover, different outcome

HALLOWEEN 2020

Region 1 fails



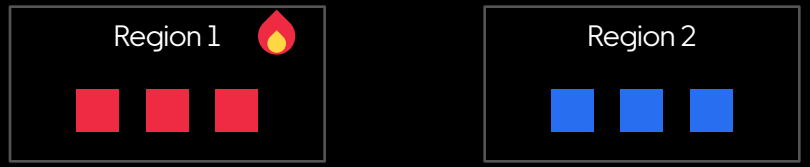
Fail over to Region 2



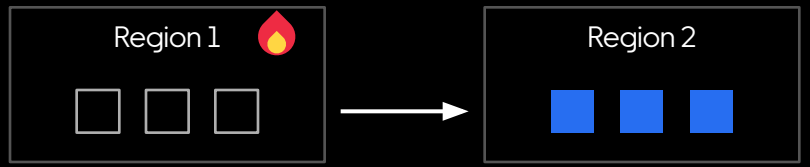
A failover overloaded the surviving region.

2026

Region 1 fails



Fail over to Region 2



No Muttloader-tested service reported capacity overload.

By the numbers (last 12 months)

28,000
load tests in production

5,300
limits found before a real failover

650
unique services load tested

700,000
CPU cores load tested

10+
tests running concurrently

50,000+
cores reduced for efficiency

What's still hard

Stateful systems
routing traffic to data

Noisy alerts
an abort is only as good as the alert

Test scheduling
dependency chains in complex graphs

Hardware differences
normalizing hardware

Memory and latency
no good target yet beyond CPU

Production drift
keeping up with deployments

Three lessons

01 The best failover validation environment is production.

02 Invest in detection and mitigation, not just prevention.

03 Reducing blast radius is key to earning trust.

Beyond failover resilience

CAPACITY EFFICIENCY

Load tests → Find limits → Reduce buffers → Save cores and $

A/B TESTS

Library safety tests

reduce blast radius of library changes

Performance tests

measure performance deltas

Race condition tests

surface hard-to-reproduce bugs

Takeaway

Production
real traffic

+

Frequency
every week

+

Safety
abort early

+

Coverage
all services



Failover resilience

Thank you

Johan Mena · jhn@uber.com
Albert Wang · albertyw@uber.com