



# Conserving your Error Budget through Resilience Testing

~dylan

# About Dylan

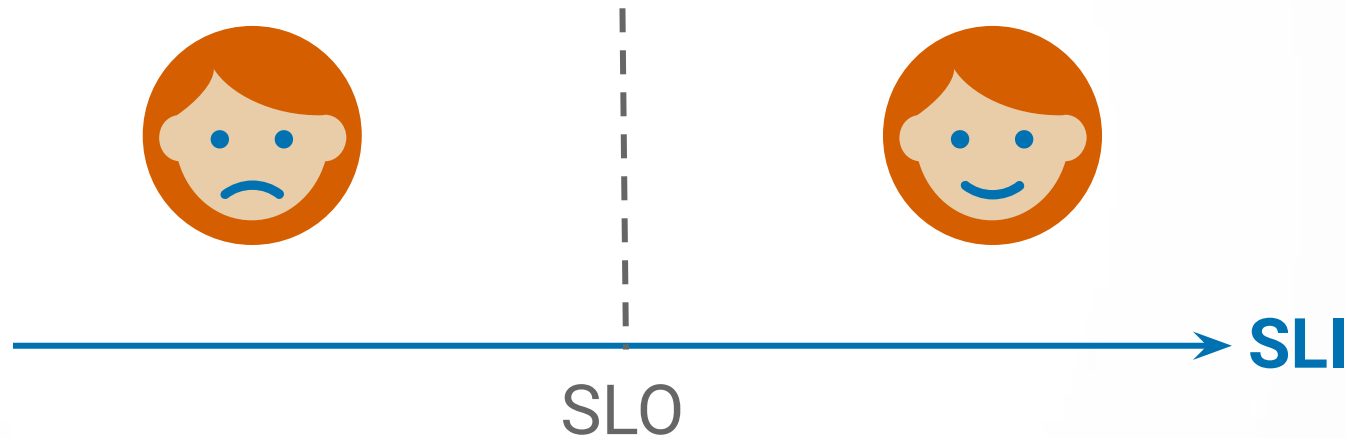
<https://www.linkedin.com/in/dviersel>

- 52 years old
- Divide my time between The Hague, Bilthoven and Amsterdam
- With Viola, 2 daughters, 3 bonus children and two cats
- Love kitesurfing and swimming
- Worked for various large and small companies as Scrum Master, Product Owner, Delivery Manager, Agile Coach, Architect, Head of Tech, Developer
- Co-founder of Perfana since 2019



# SLOs & Error Budgets

A quantitative measure of the **acceptable** level of service unreliability or downtime.



*Google - The Art of SLOs*

if the service operates within the error budget, it indicates a **healthy** balance between **reliability** and **feature development**, encouraging teams to innovate while maintaining a focus on SRE principles

# Stopping the line

Error Budgets are a way to **tolerate imperfection** (unavailability, slow performance, etc.).

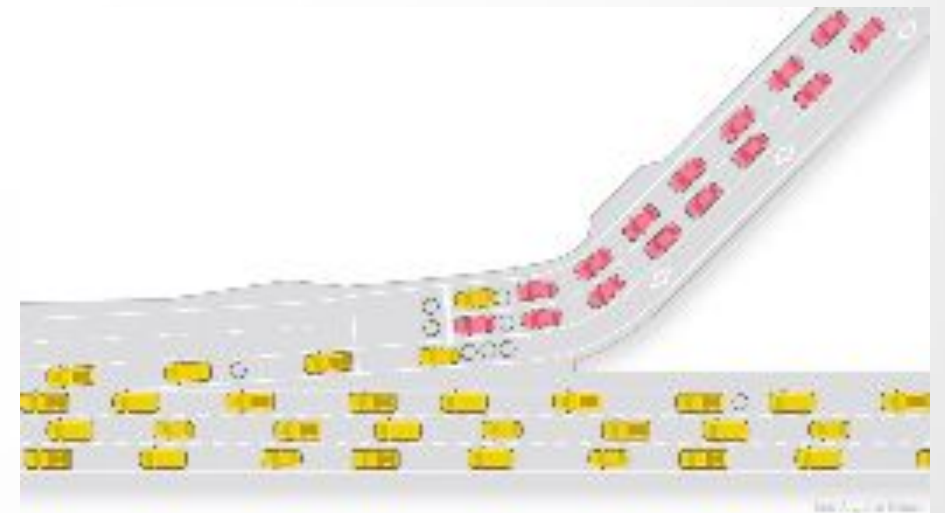
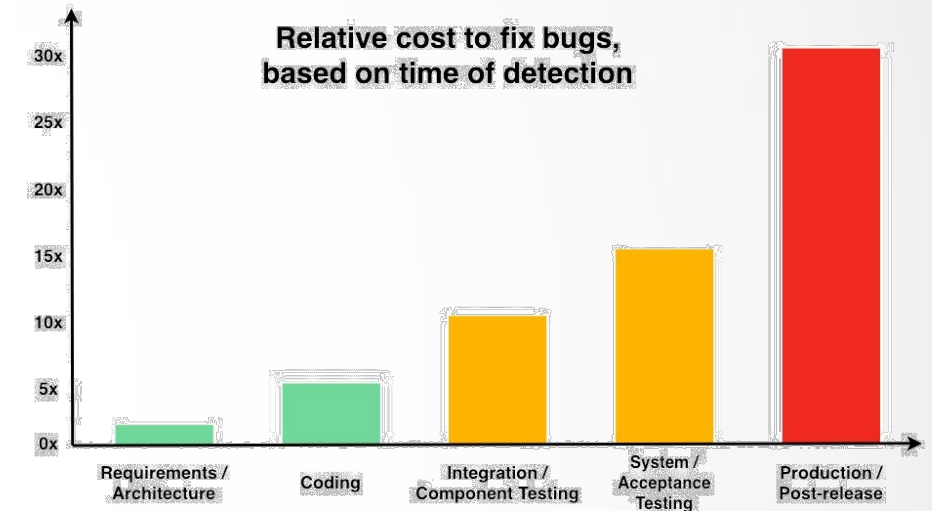
This can be used for useful or necessary stuff like:

- ✓ releasing new features
- ✓ expected system changes
- ✓ inevitable failure in hardware, networks, etc.
- ✓ planned downtime
- ✓ risky experiments

Conversely... **depleting** the Error Budget means **stopping the line**, to get back on track.

# Conservation

- Budgets are usually tight. Spend them wisely. Unforeseen things can happen.
- The cost of fixing defects rises exponentially with the time of detection.
- Production issues/defects tend to severely disrupt the flow.
- Stopping the line completely blocks the regular SDLC flow.
- Prevention is better than curing



# Shift-left AMAP

- Unit testing? Sure. 😬
- Functional testing? Maybe. 🤔
- Integration testing? Hmm.. 🤨
- Performance testing? Difficult. 😞
- Chaos testing? Hell no! 🤯

# Resilience testing

Testing focused on evaluating how well a system can recover from **crashes**, hardware **failures**, network **issues**, and **high traffic loads**

The goal is to **identify potential weaknesses** thereby enhancing overall reliability and **user satisfaction**.

# “But.. but.. but...”

“The test environment is not live-like”

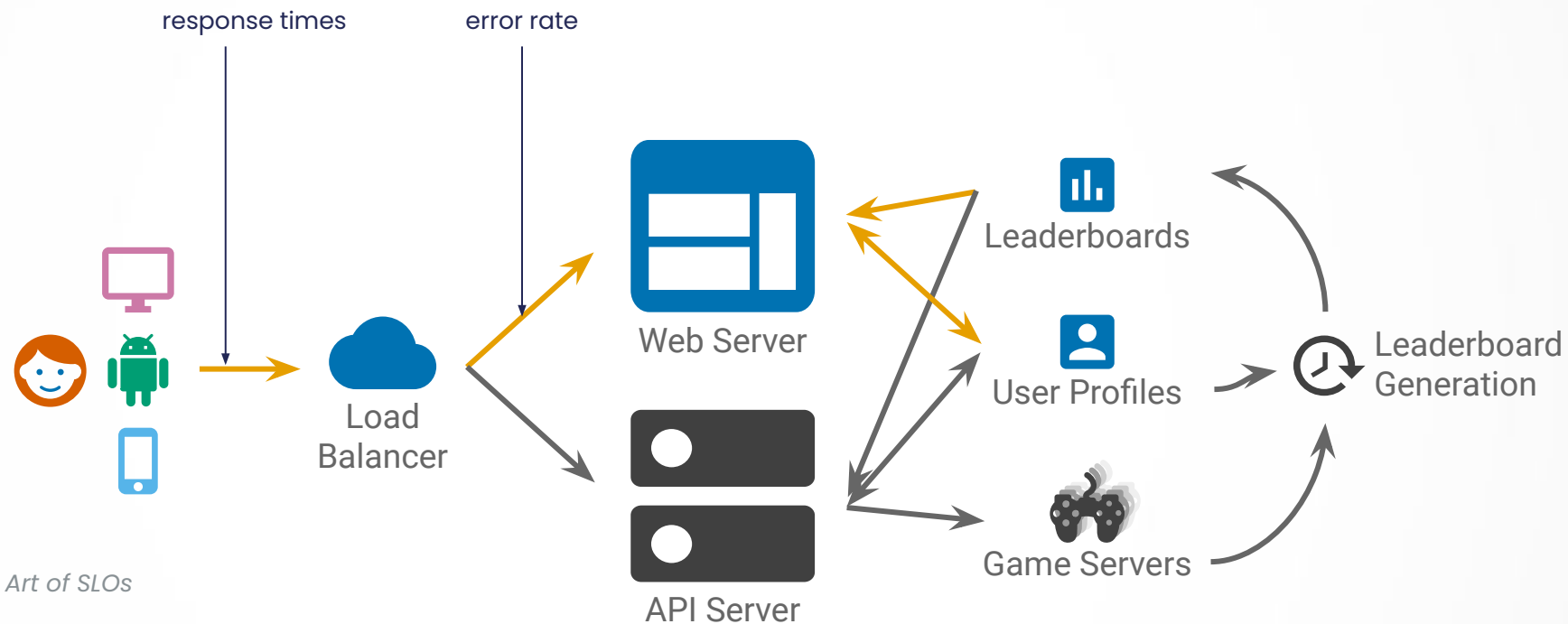
“We need a copy of production”

“(...) testing is too hard, we’ll test it in production”





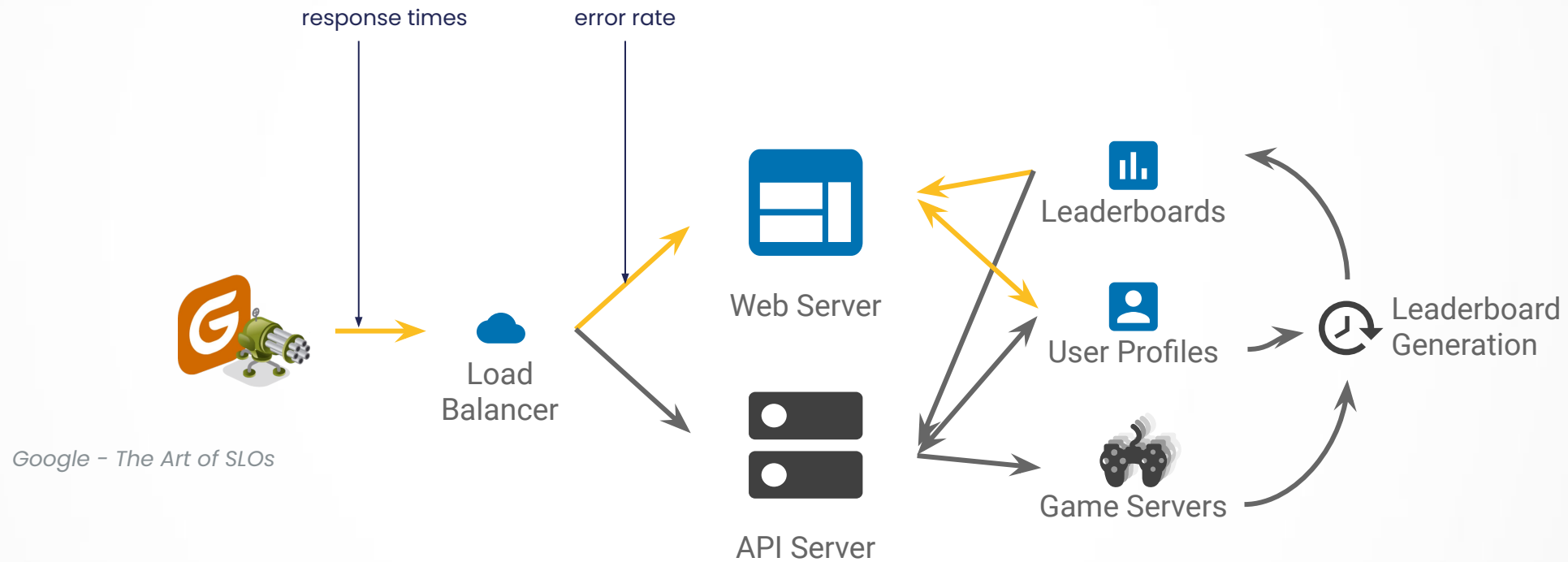
# Production setup



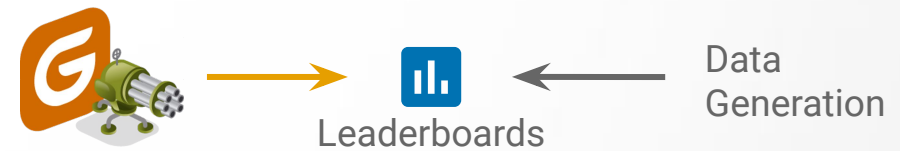
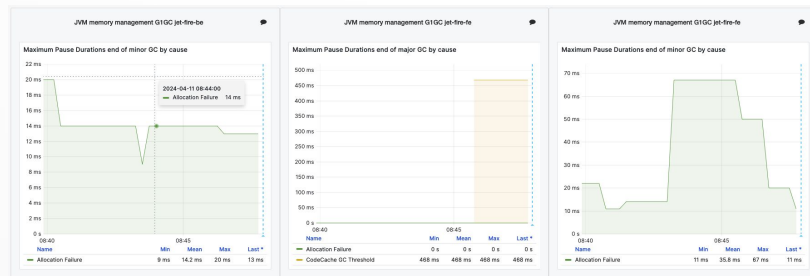
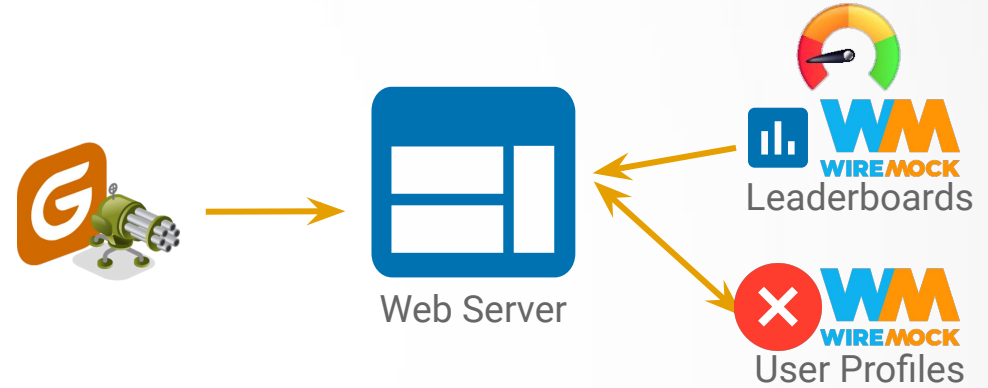
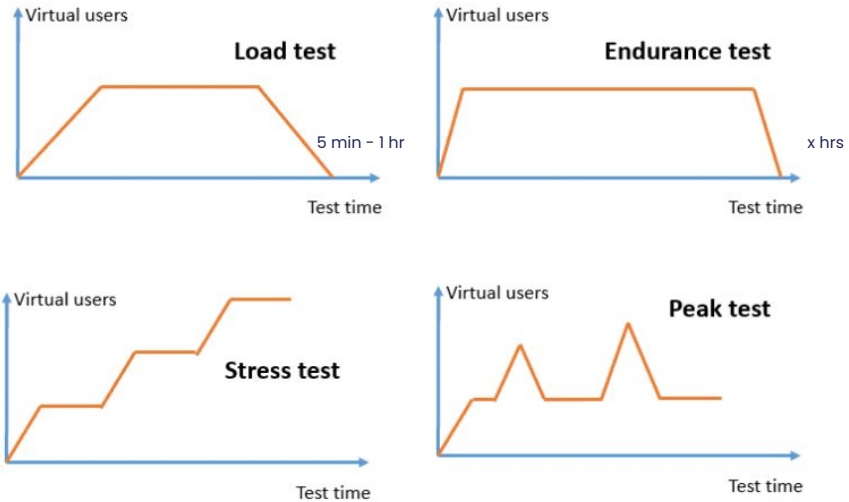
Google - The Art of SLOs

SLOs: 3–5 **leading** indicators per **user journey** that tell you whether the user is happy.

# E2E Test setup

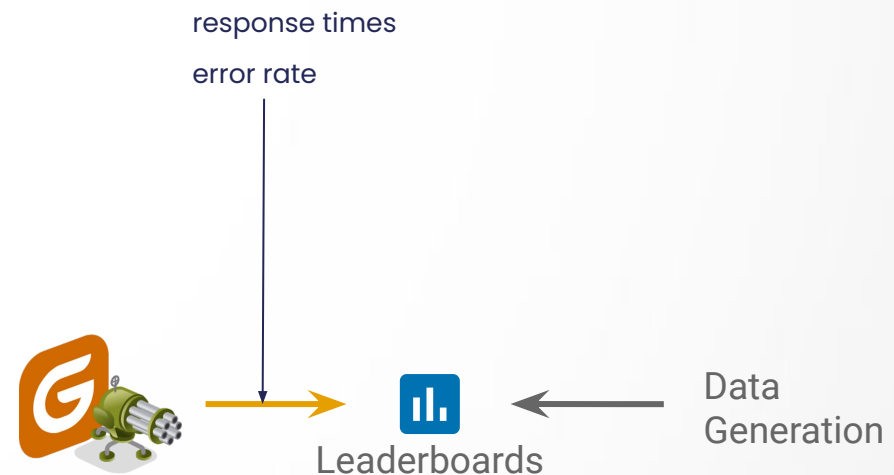
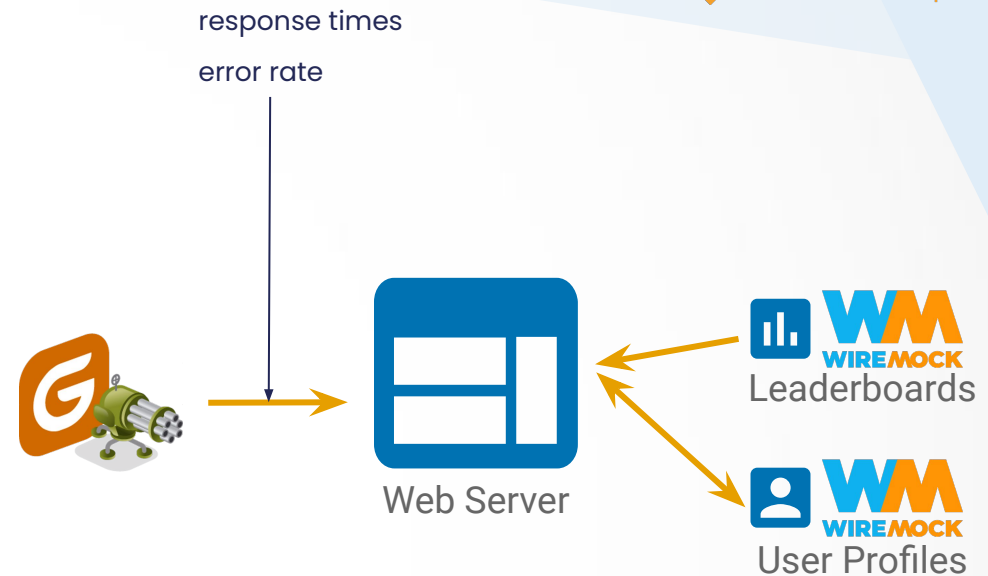


# Regression test setup



# Regression test setup

- Smaller scope  
→ isolate the user journey
- Easy to manipulate/simulate
- Run from the pipeline
- Detect regressions early



# Challenges


- Test infrastructure
- Test data
- Scope and dependencies
- Simulation and traffic shaping


# Test infrastructure

- Treat your infrastructure like cattle.
- Apply the same engineering principles as you do for production provisioning incl observability.
- Automate everything.

Disposable test environments that have known predictable characteristics and that are easy to deploy.

Does not have to be “a copy”. We’re interested in the regressions.

	<b>Pets</b> Legacy Infrastructure
Pets are given names like grumpycat.petstore.com They are unique, lovingly hand raised and cared for When they get ill, you nurse them back to health	Infrastructure is a permanent fixture in the data center
Infrastructure takes days to create, are serviced weekly, maintained for years, and requires migration projects to move	Infrastructure is modified during maintenance hours and generally requires special privileges such as root access
Infrastructure requires several different teams to coordinate and provision the full environment	Infrastructure is static, requiring excess capacity to be dormant for use during peak periods of demands
Infrastructure is a capital expenditure that charges a fixed amount regardless of usage patterns	

	<b>Cattle</b> Cloud-Friendly Infrastructure
Cattle are given numbers like 10200713.cattlerancher.com They are almost identical to other cattle When they get ill, you replace them and get another	Infrastructure is stateless, ephemeral, and transient
Infrastructure is instantiated, modified, destroyed and recreated in minutes from scratch using automated scripts	Infrastructure uses version-controlled scripts to modify any service without requiring root access or privileged logins
Infrastructure is self-service with the ability to provision computing, network and storage services with a single click	Infrastructure is elastic and scales automatically, expanding and contracting on-demand to service peak usage periods
Infrastructure is an operating expenditure that charges only for services when they are consumed	

# Test data

How to deal with:

- privacy
- relationships
- liveliness
- volume

## Strategies

- Copy production - not advised
- Generate data - easy enough for simple cases
- Use proper tools that have solved the issue of
  - Generation
  - Masking
  - Subsetting

Examples: gretel.ai, tonic.ai and others.



# Scope and dependencies

- Shift left → reduce scope
- Isolate the user journey
- Mock dependencies with a known and predictable reliability
- Simulate unreliability “events” such as slow performance or outage.





# Realistic load generation

- Use production statistics:
  - nr of concurrent users per **journey**
  - nr and type of transactions/interactions
  - nr of back end calls
  - etc
- Account for seasonal/periodic influences: end of month, holidays
- Sophisticated tooling is available
  - Record and playback
  - “Replay” production traffic and auto-create mocks. For example:  
<https://speedscale.com>

# Beware of the pitfalls

- Production SLOs often will need to be decomposed into derivative SLOs to make up for the lack of similarity, complexity, etc.
- End-to-end: Smallest possible architectural shard to deliver the “user journey”; mock and model everything that has a known (un)reliability
- The role of the PO – should be fixed by properly implementing SRE, SLOs and Error Budgets

# Finally

- Testing in production can be a valid approach as long as you are aware of the risks and willing to spend your budget on it.
- However.. Prevention is often better than curing. Shift-left.
- With proper design and implementation, resilience testing in pre-prod is not only achievable but has a number of well-known benefits
  - Fast feedback, fail fast
  - Can be repeated many times with marginal costs
- Apply platform engineering principles
  - Automate everything, cattle not pets
  - Create building blocks in shared pipelines



**Perfana helps tech teams**  
**GO FAST WITH CONFIDENCE**  
**through automated software**  
**performance validation**