Google

Why is my app SLOw?

Defining reliability in platform engineering

Jez Humble, SRE, Google Cloud • sre.google • twitter.com/googlesre
Slides by Jez Humble, Narayan Desai, Brent Bryan

Site Reliability Engineering

Why is my app SLOw? Defining reliability in platform engineering

Platform engineering is all fun and games until platform customers start complaining about their apps running slowly. Is it the app code or the platform? This talk looks at how Google's Serverless SRE team detects platform-level latency regressions before users, measures the impact of regressions, and tracks performance over time. We'll discuss the limitations of SLOs in this context and how to take a statistical approach that gives a customer-centric picture of the performance of our platform instead.

Acknowledgements

Serverless SRE team: Alan Hawrylyshen, Aleksej Truhan, Anna Ayvazyan, Anna-Kaisa Pietilainen, Eric Ross, Fae Hutter, Francis Tang, Hayley Farnworth, Ib Lundgren, Inderjeet Sharma, Jez Humble, Jim Olwell, Jimmy Chen, Joan Grau, Kira Zhovnirovskii, LD Maya, Nick O'Connor, Omar Morsi, Pascal Bouchareine, Steve Jordan, Tong Yin, Will Patterson, Wolfram Pfeiffer, Yi Chen, Yuchen Ying

Kraken team: Jacob Frelinger, Jayasree Beera, Jeff Borwey, Brent Bryan, Narayan Desai, Tyler Sanderson, Sam Schneider

Also Chris Heiser, Greg Block, Eric Brewer, Jenny Sager, Kevin Tian, Niall Murphy, Nicole Forsgren

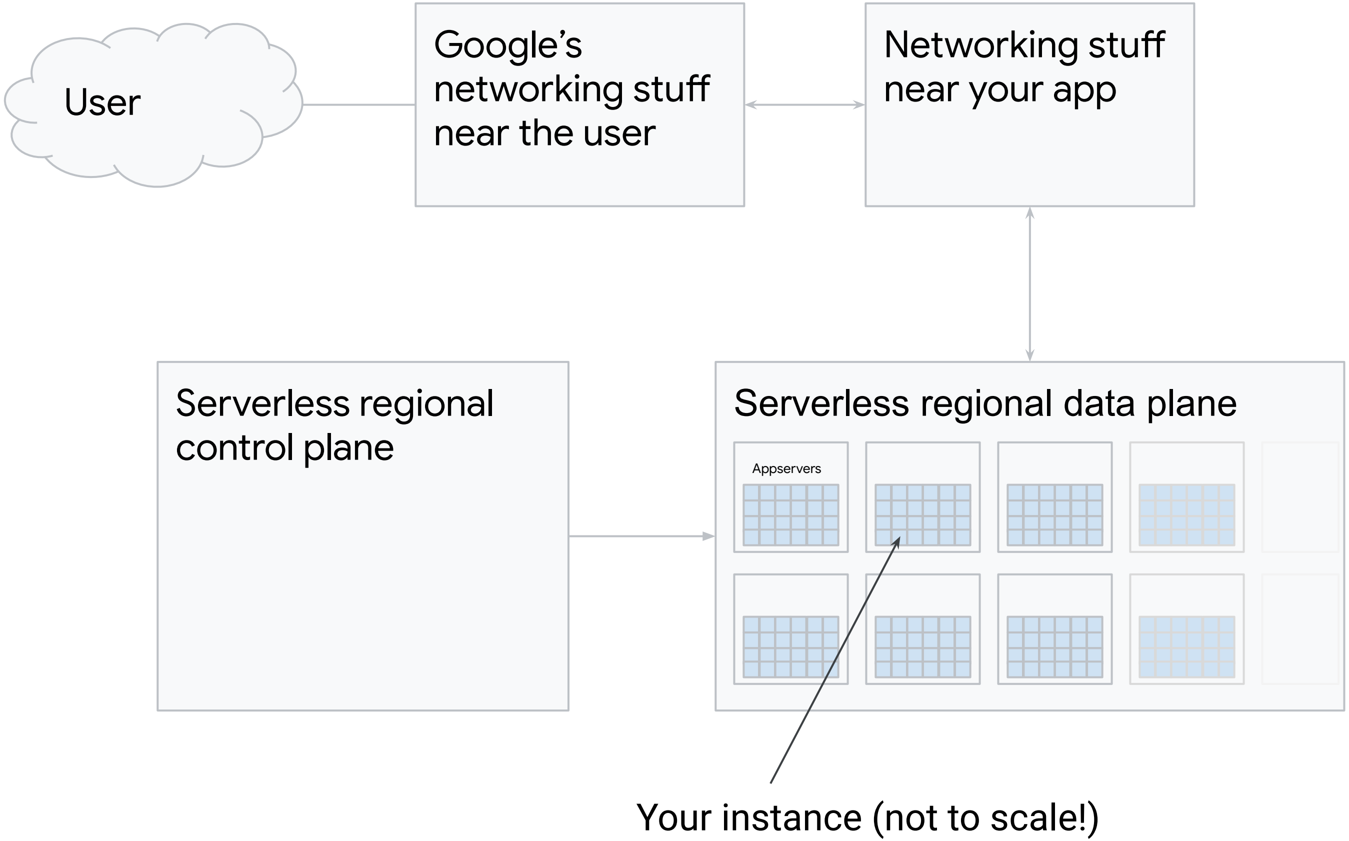
Serverless platform is amazing

Deploy containers / apps from the command line and we take care of all the infrastructure / scaling. You can scale down to zero and up to thousands of instances in seconds.

In other words, our business model is selling you the ability to apply severe stress to our platform.

It works really well!

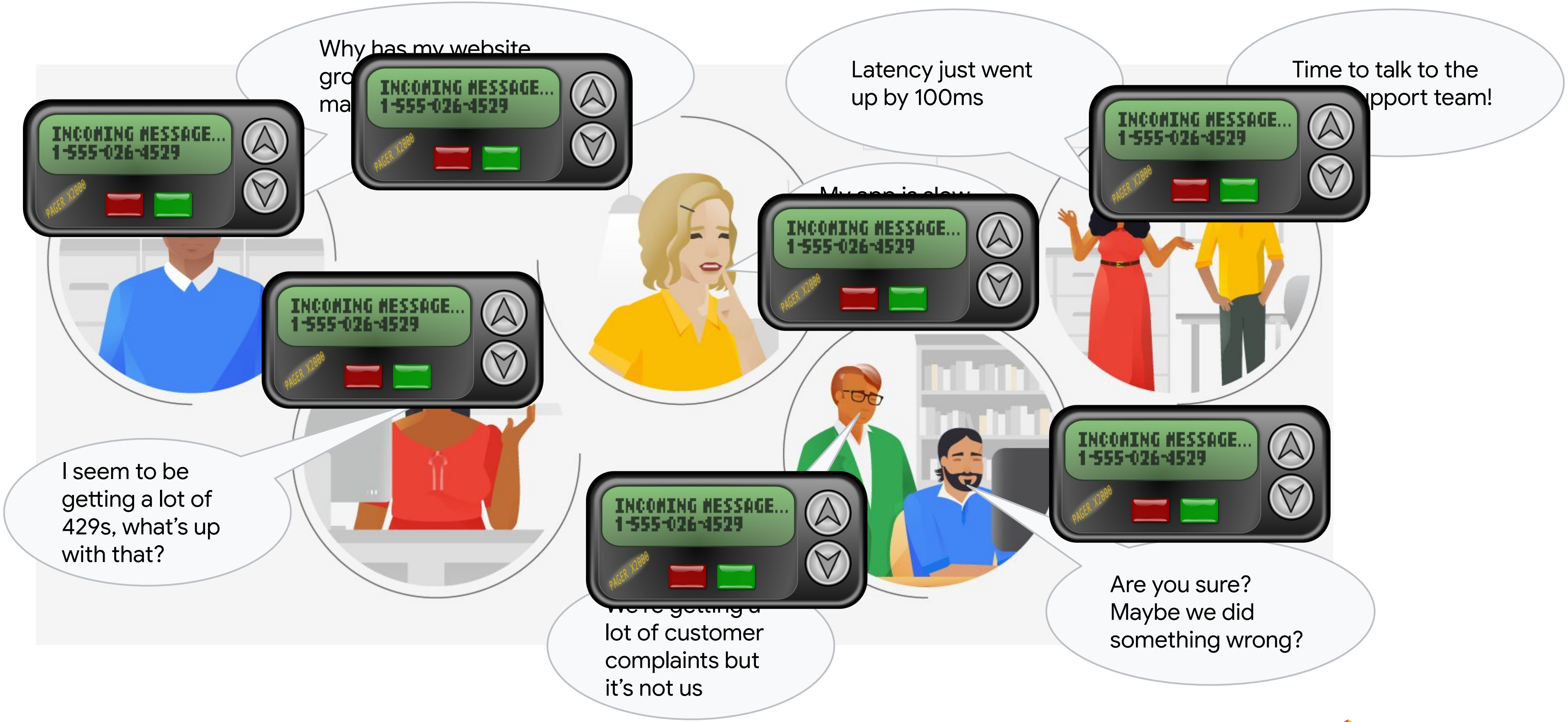
Serverless platform



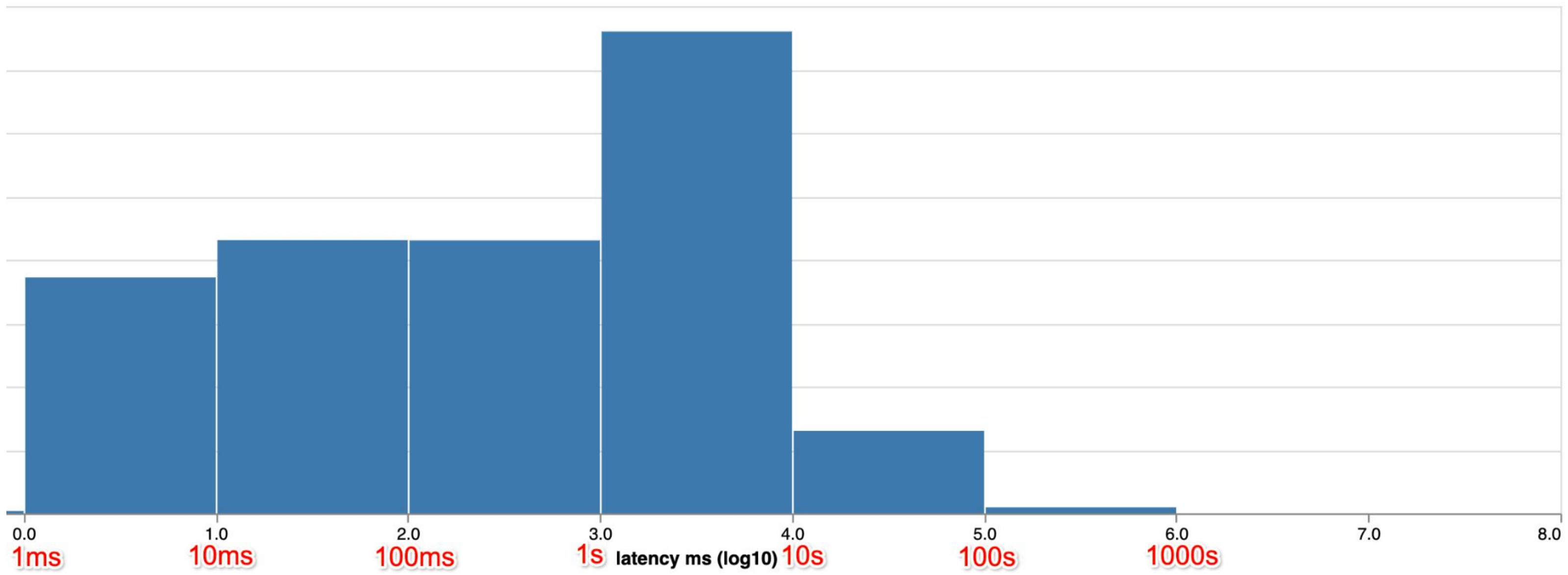
“My app is slow”

- You changed some code / config
- Change in latency/availability of dependencies
- Change in traffic patterns to your app / the platform / Google infrastructure
- Platform change
- Some config change somewhere in Google
- Noisy neighbor(s)
- DoS attack / abuse
- Suboptimal clone binpacking
- … (so many things!)

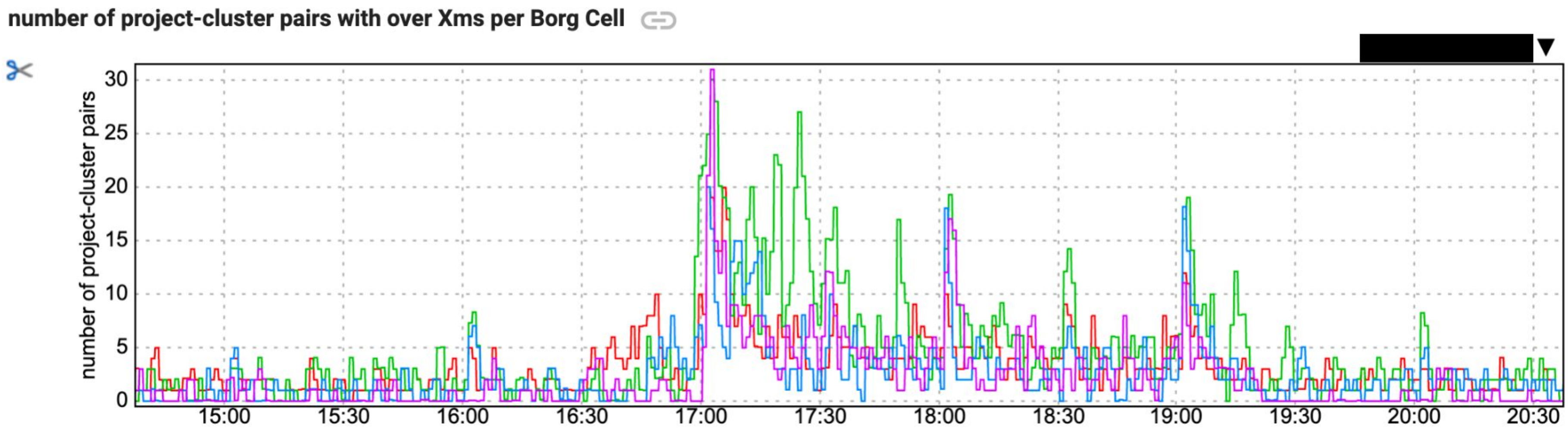
The platform is slow



Total (end-to-end) latency distribution



Request delivery latency



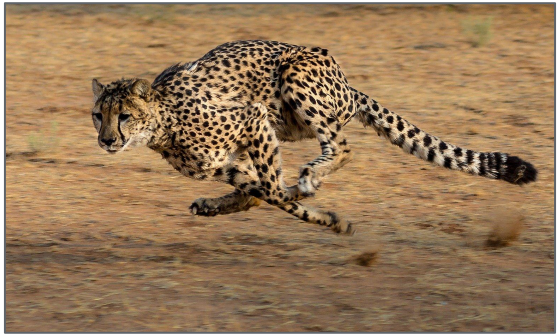
Goal

- A metric that represents the customer experience
- Combinable across projects / cells / regions
- Can be used to detect anomalies affecting multiple customers (likely platform issues)
- Computationally cheap (high QPS)
- Principle-based

Reliability

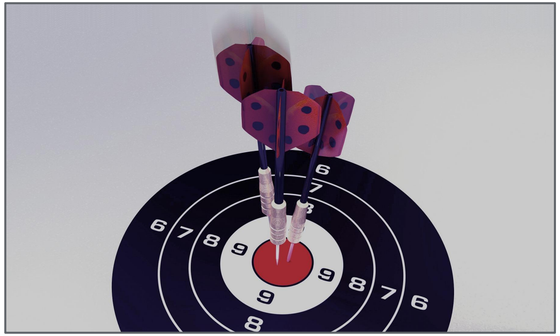
Availability

Is the service there when you need it?



Performance

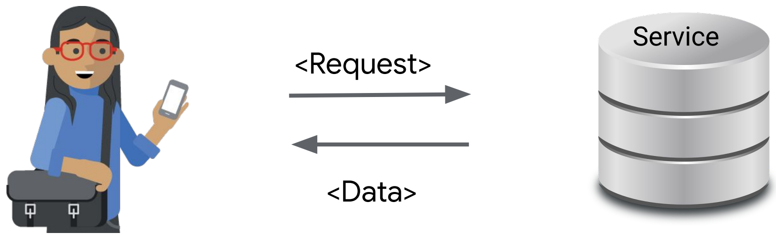
How effectively is work performed?



Correctness

Does a service do what's expected?

Reliability in Practice



Availability

- ✔ Count the number of failed requests
- ✘ 400s vs 500s
- ✘ Deadlines
- ✘ Malformed Requests
- ✘ Retries Magnify Errors

Performance

- ✔ Set P99 latency SLO
- ✔ Create Probers
- ✘ Workload dependent
- ✘ Probers are narrow

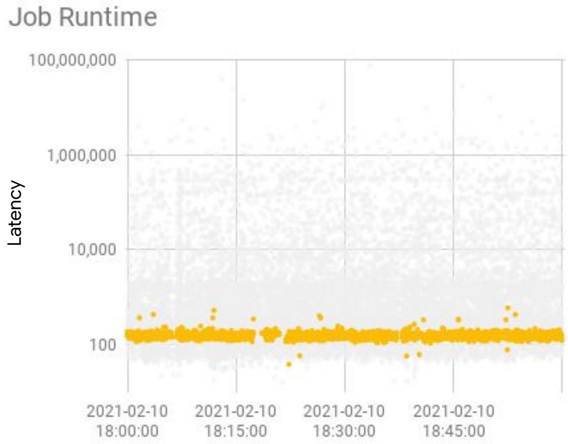
Correctness

- ✔ Lots of tests
- ✔ Canary analysis
- ✘ Limited, non-adaptive coverage
- ✘ Hope is not a strategy

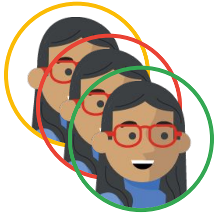
Applying to the Model

Workload → Performance ← Service

Performance ↓ Reliability



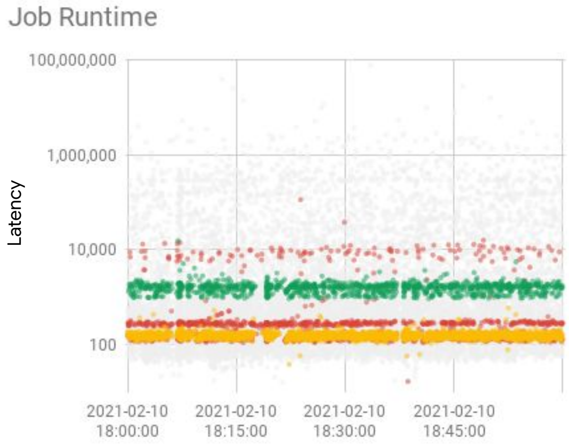
Applying to the Model



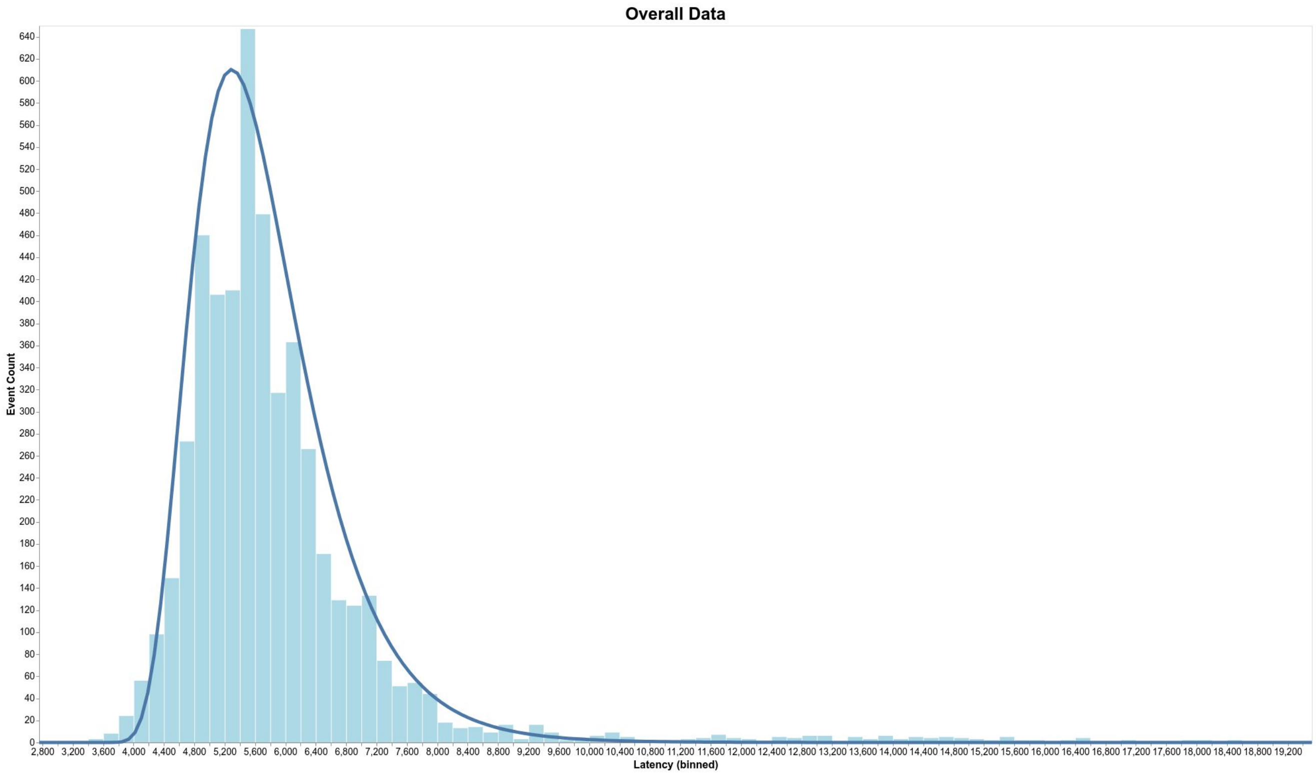
Workload → Performance ← Service

↓

Reliability



Stationarity



2σ Technique

2σ Technique

Hypothesis:

Self-Similar Workloads Should Have Consistent Performance

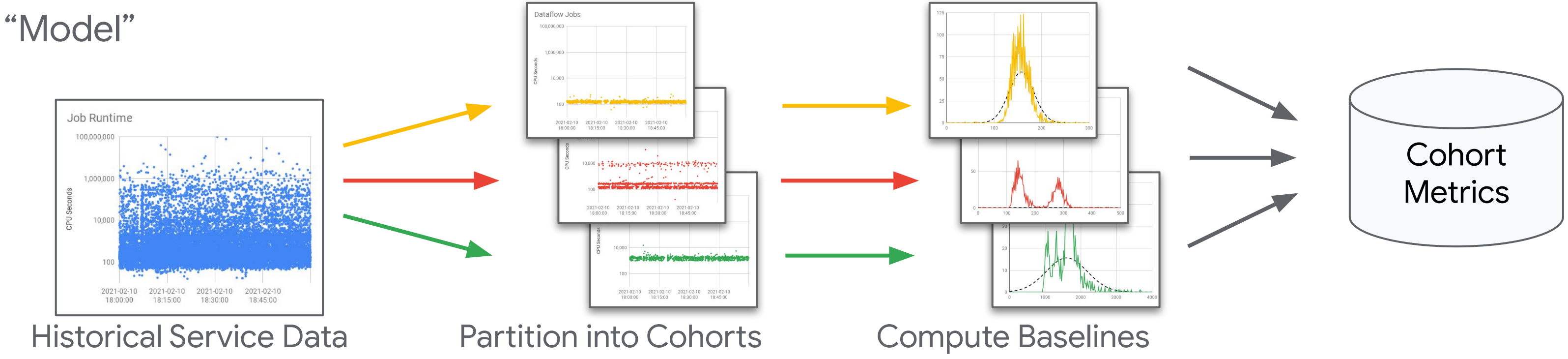
Technique Overview:

- Partition workloads into Cohorts ⟵ *Approximate Intent via Workload Features*
- Build Performance Baselines ⟵ *Estimate Distributional Form (e.g. Normal)*
- Estimate Likelihood of Delivered Performance ⟵ *Test For Stationary*

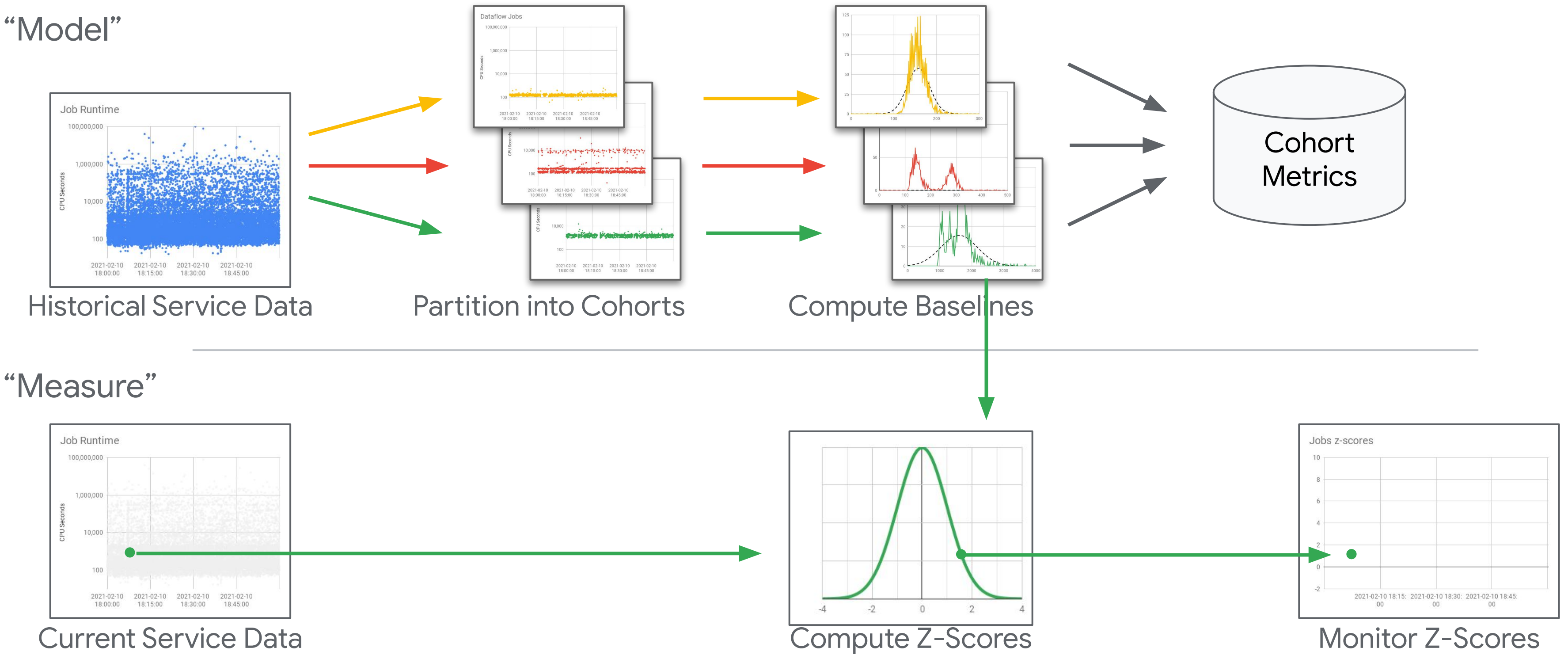
Result:

- Set of Events with Predicted Likelihoods
- Time-series of summary statistics describing concentration of extreme outliers

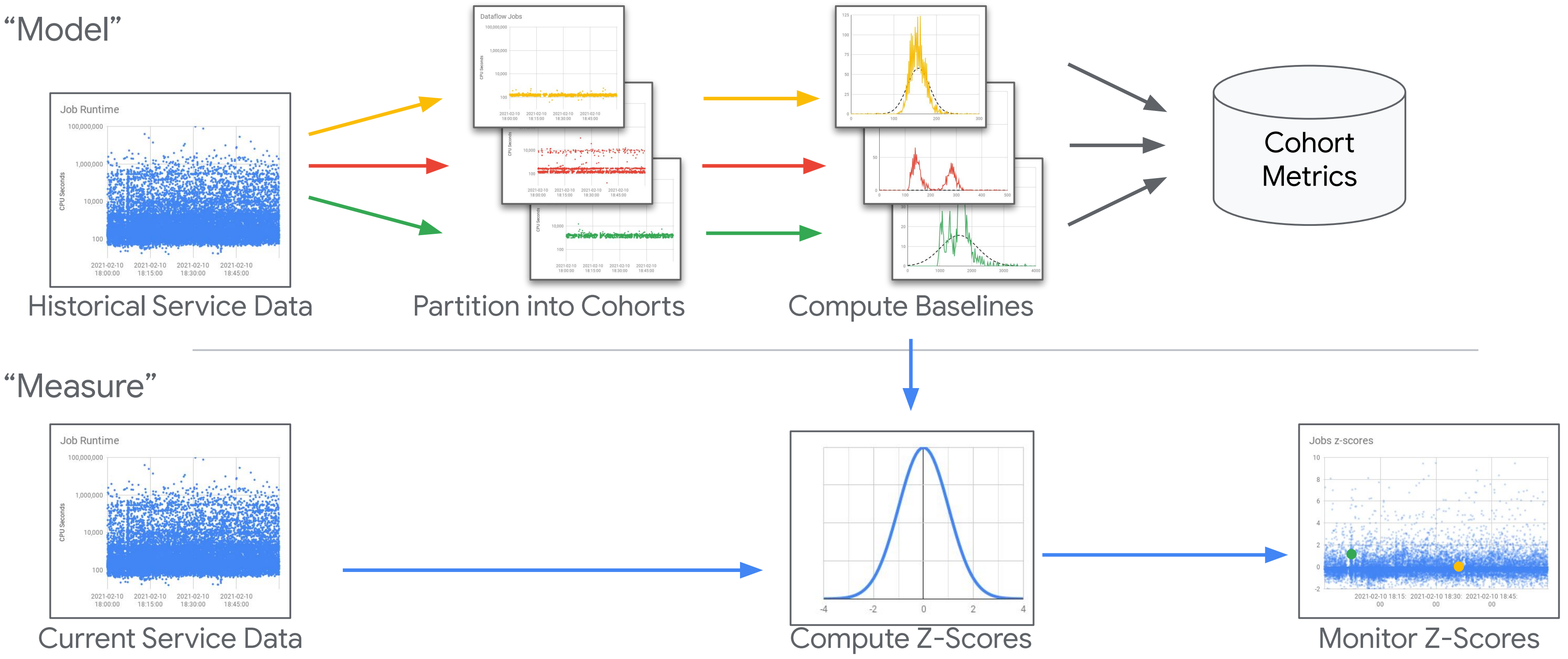
Leveraging Structure: 2σ Technique



Leveraging Structure: 2σ Technique



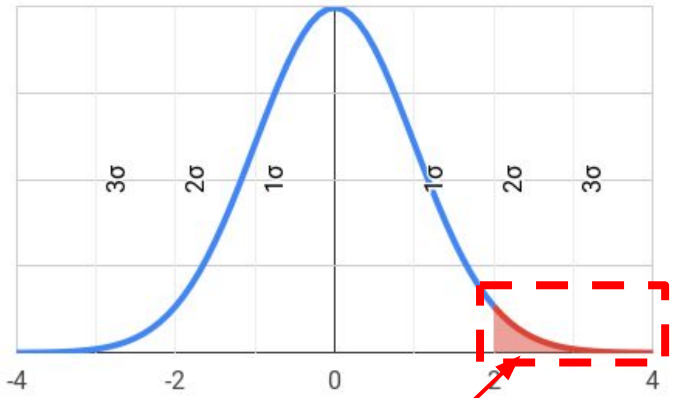
Leveraging Structure: 2σ Technique



Mechanics

Strategy:

- Aggregate z-scores across workloads
- Monitor fraction of workloads with z-scores ≥ 2, in windows
- Expect 2-5% 2σ outliers in any given window
- When >10% of workloads are >2σ, **BE AFRAID**.



Detection is based on fraction of workloads exhibiting regression

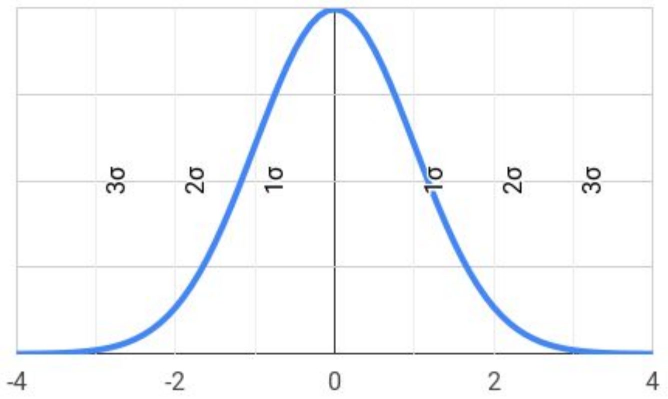
Approximations Unlock Leverage

Assume:

- Metric distributions can be approximated by parameterized distribution
- Modeling errors excluded via baseline qualification

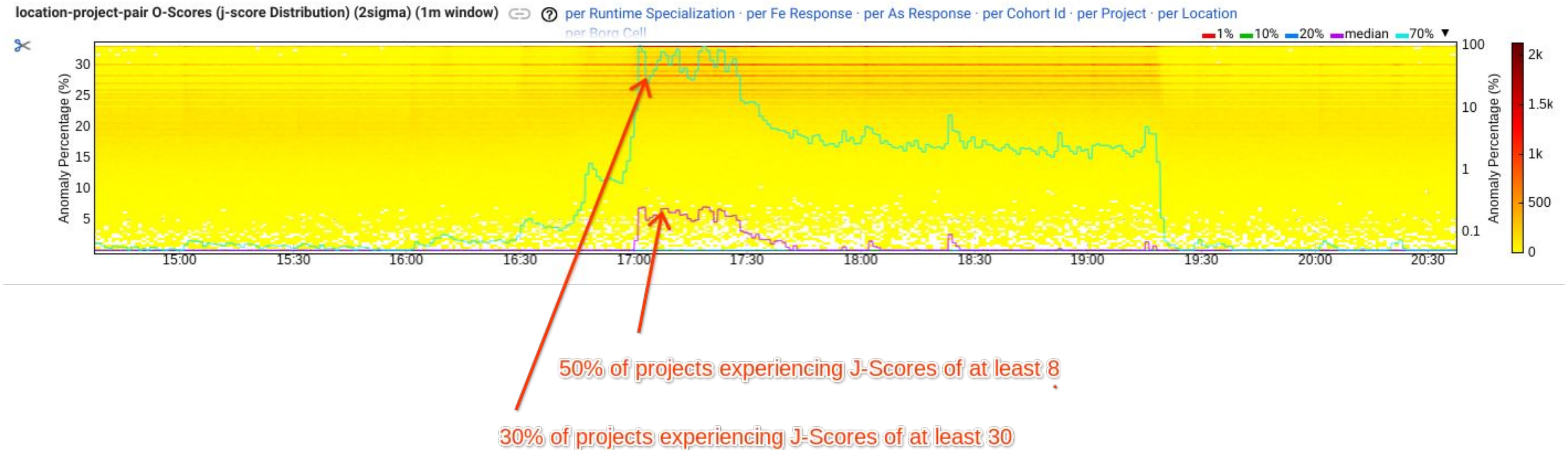
Then:

- Workload z-scores are a proxy for likelihood
- Workload performance should be IID
- Z-scores follow a standard Normal distribution
- Baseline distribution computation is "embarrassingly parallelizable"
- Z-scores are combinable (across cohorts!)

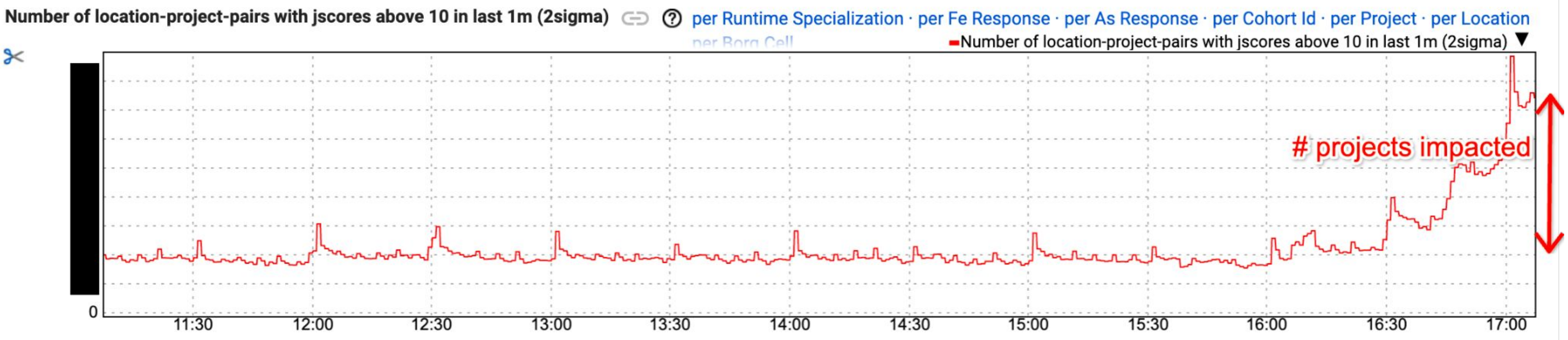


$$\mathrm{z-score} = \frac{\mathrm{obs.\ \ workload} - \mathrm{baseline\ mean}}{\mathrm{baseline\ std}}$$

Overload score



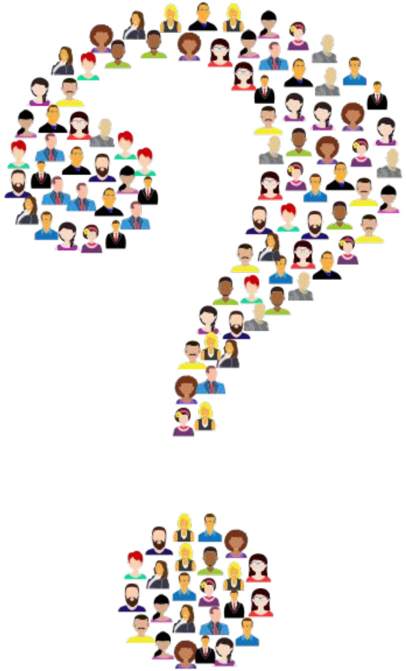
Impact analysis



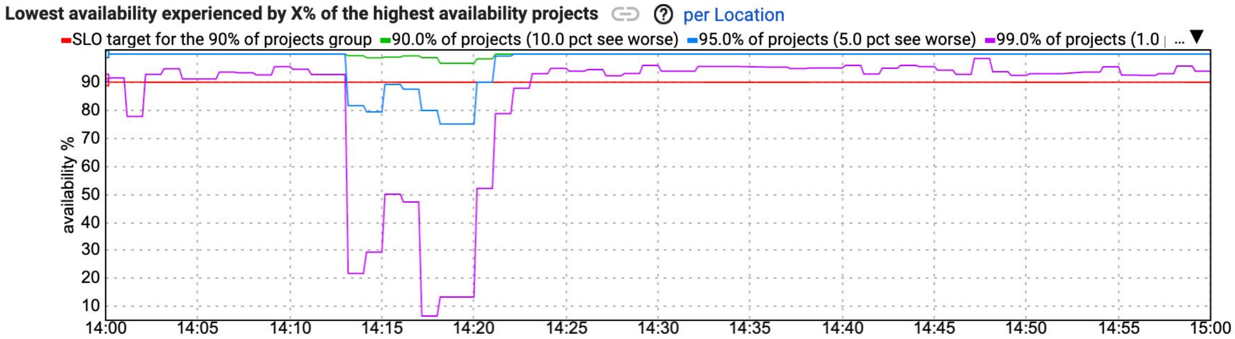
Frequently Asked Questions

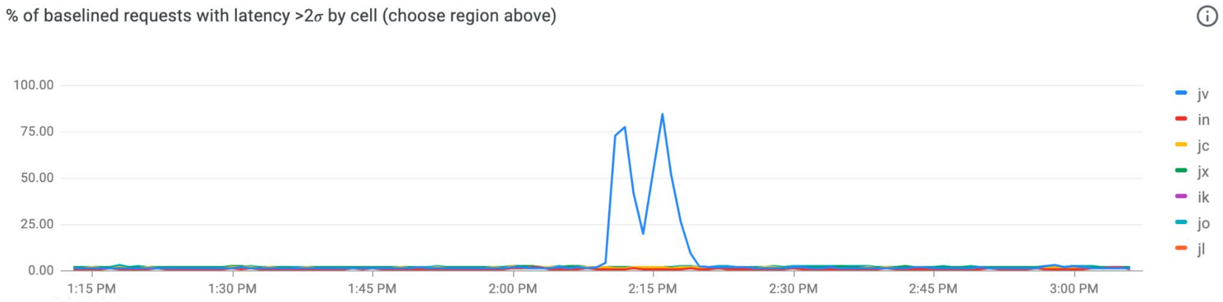
- Do performance metrics actually follow Normal distributions?
- How do you know if approximations hold?
- How do you define cohorts?
- How do deal with “singleton” / infrequent workloads?

- Ok, but does this *really* work?



Backtesting





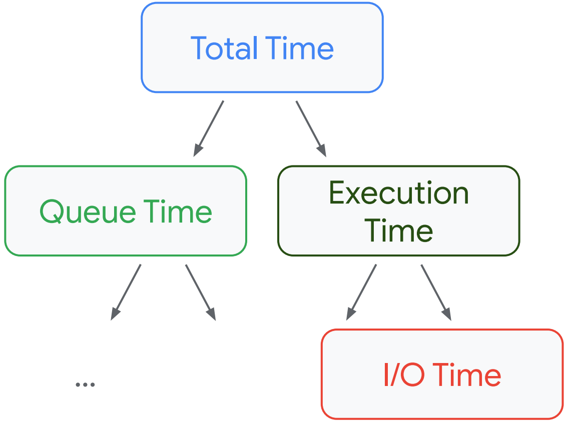
Limitations

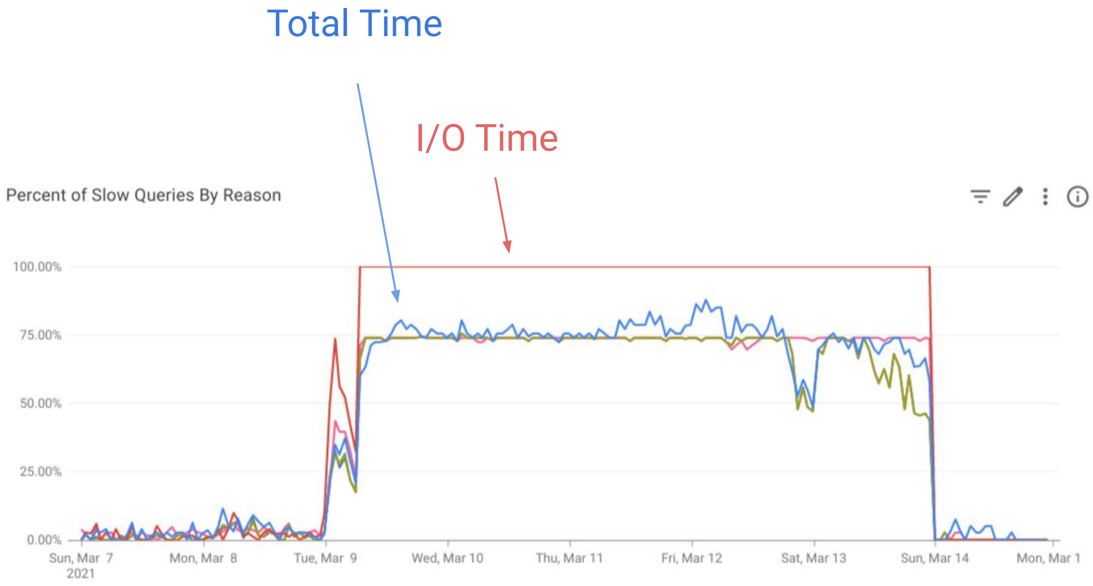
- Hard for people to interpret without first understanding stats words
- Cohort coverage ~40-60%
- Doesn’t tell you why there’s a problem (symptom-based not cause-based)*

*Note that symptom-based is a feature not a bug

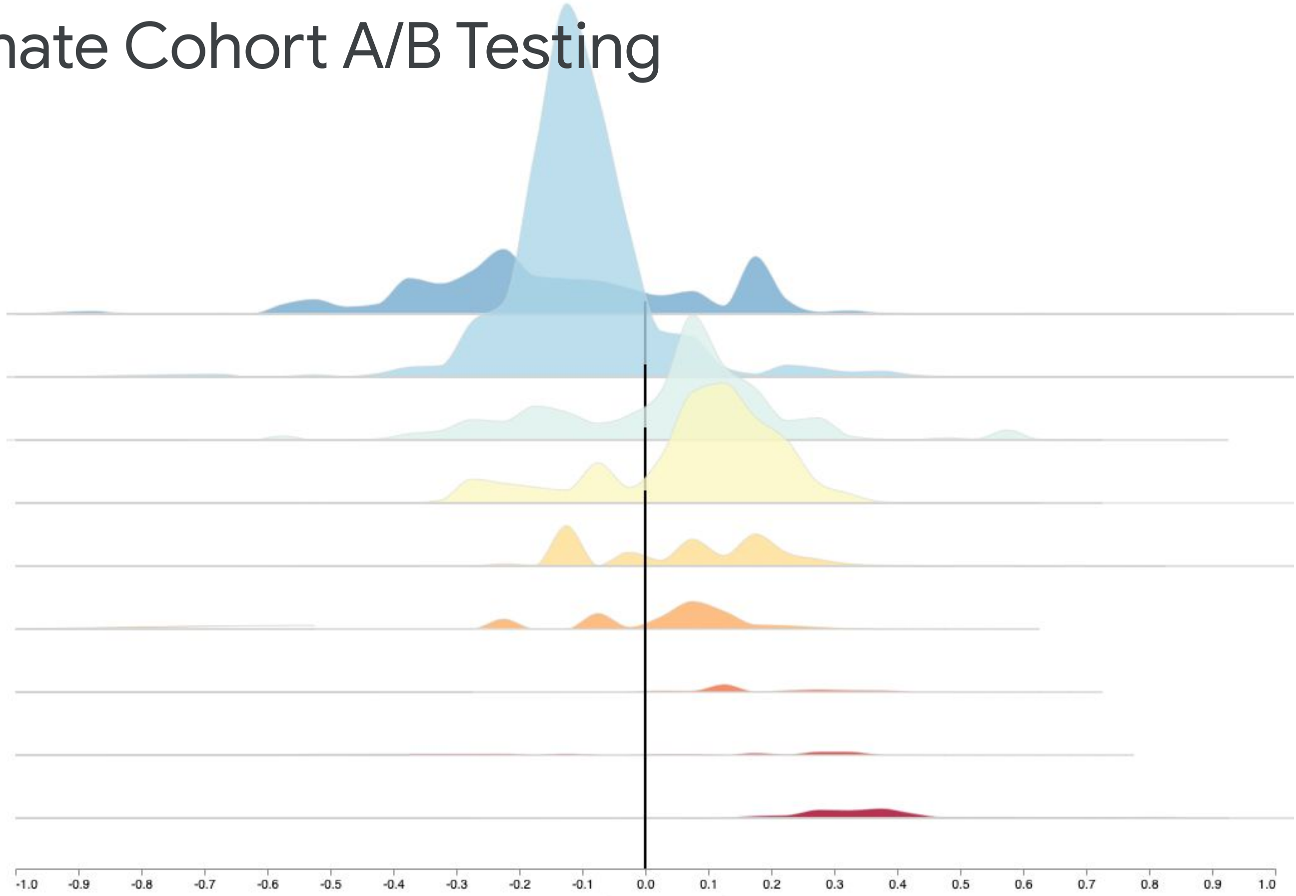
Other Applications

Streamlined Diagnosis





Approximate Cohort A/B Testing



Conclusions

Key Observations

- **We can reliably detect and measure the impact of platform regressions**

- Reliability is a shared property (between customer & service)
  - Reconstruction of end to end behavior is critical
- Metric combinability is critical for analysis
- Variability is what customers actually care about
- Distributed systems often produce decorrelation
  - We can measure it, and its absence
- Workload correlation can identify proximate causes

2σ method

- Incorporates user intent in order to model expected performance
- Tests an IID hypothesis to infer when systems diverge from expected behavior
- To produce data products that are comparable and combinable

We use these data products in order to:

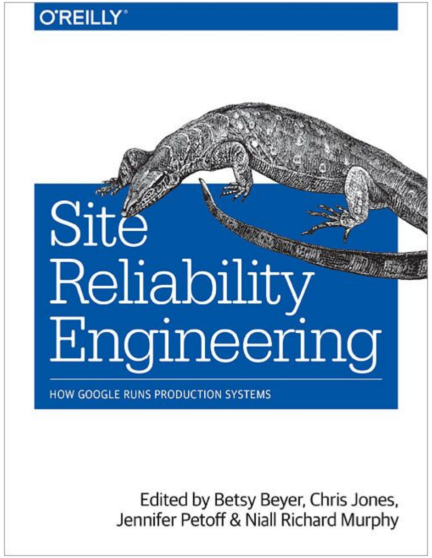
- Perform change point detection when systems diverge from expectations
- Estimate the duration, severity, and specific impact of these excursions
- Localize subsystem performance problems
- Compare relative and absolute performance over time and arbitrary workload dimensions
- Directly measure correlation across subsystems and isolation domains

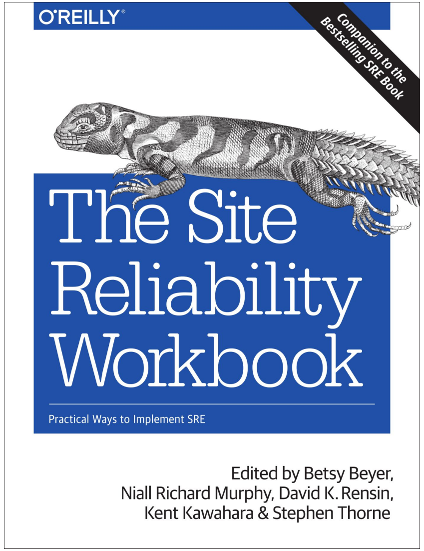
Resulting in:

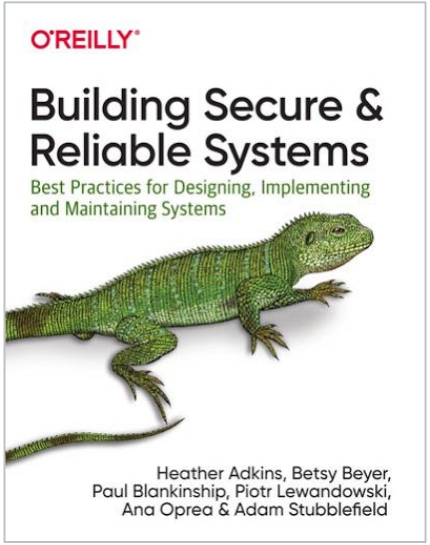
- Calibration-free insights that characterize the consistency of a system
- The ability to test system invariants continuously
- Data building blocks that can be reprocessed to answer many questions

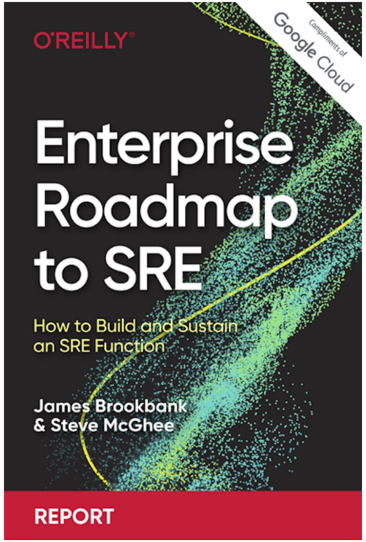
See https://www.usenix.org/conference/srecon22americas/presentation/desai

Find Google SRE publications—including the SRE Books, articles, trainings, and more—for free at sre.google/resources.









Questions