Learning from Incidents at Google

@stevemcghee //
smcghee@google.com

Overview

1. Google is **big** and **complex** and seems to stay up
   - Decades of incidents to learn from! :)
   - GCP Customers provide yet another layer

2. "Programatization" of LFI across a very large Org
   - Understanding **Dependencies** and **CUJs**
   - Tracking **Risks**, **Weaknesses**

3. So what? Does it work?
   - Bonus: STPA/CAST

Google is Big ~= **Scale** & Complexity

- **Search**: Billions of searches per day, 15% of each day's traffic is new
- **Google Play**: 2.5B active users (190 countries), 3B active devices
  - 1B new devices last year!
- **Cloud**
  - Mission critical systems for large enterprises:
    - banks, telcos, etc (not just tech)
  - 35 regions, 106 zones, 173 edge locations, 200+ countries
  - BigQuery: 110 TB/s, Spanner: 2 billion rps (peak)
  - >2B k8s pods / month
- Ads, Geo, News, Photos, YouTube, Waymo, Verily, DeepMind, etc etc

Google is Big ~= Scale & **Complexity**

Products := Apps, APIs, pipelines, etc

Backed by 1000s of microservices

Built by *many* eng teams

All on **Borg**, worldwide network

100s of languages, locales, jurisdictions

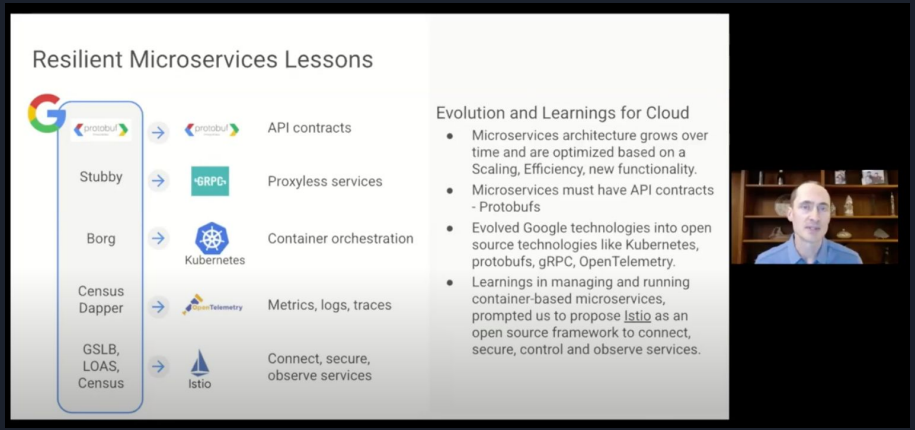
Target of state-sponsored attacks

Google is Big and **"Up"**

"Google is always up!" - anonymous

"5x9" ~= 5m downtime/year

autohealing req'd



Allen School Distinguished Lecture: Brad Calder (Google)

`Scale::Incidents` but `Scale::Impact?`

"Lots of lightning strikes in a huge forest"

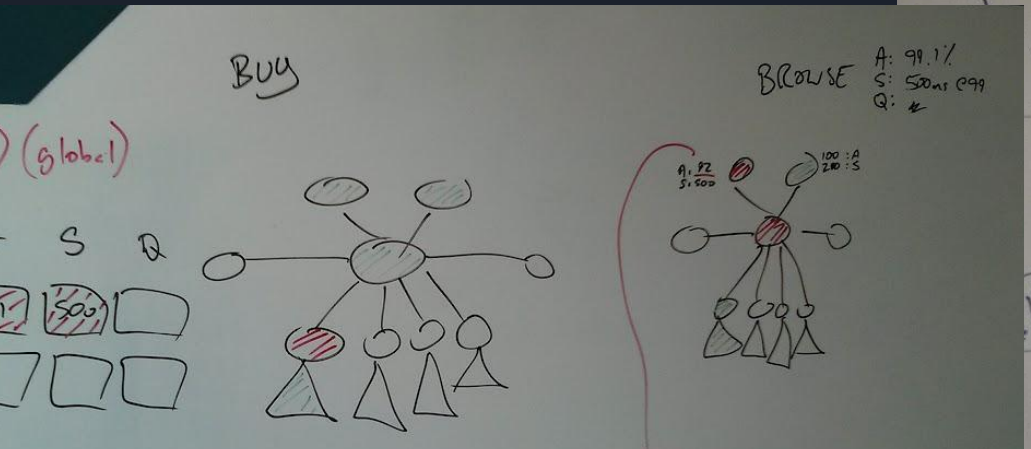
⇒ Minimize the **impact** of incidents

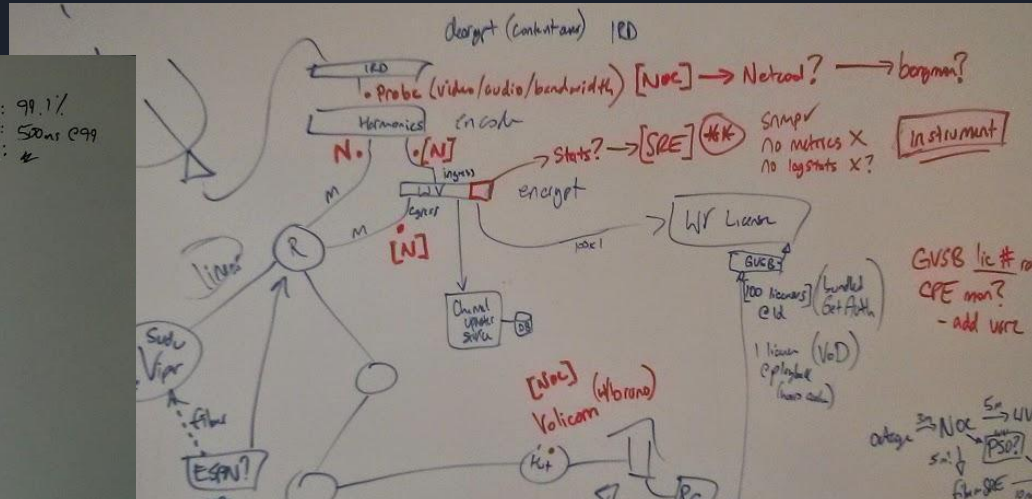
> "100% is the wrong reliability target for basically everything."
>
> **Benjamin Treynor Sloss,** Vice President of 24x7 Engineering, Google

I'm an SRE!

Google's approach to reliability. There are books!

SRE != resilience, reliability eng, systems safety
but there is **an overlap** (I think)





Yet Another Definition of SRE

Distributed systems design for **graceful degradation**

Gradual change (canary releases, experiment frameworks)

Traffic management + data replication across distributed compute

Design patterns, eg: separation of data plane, control plane, mgmt plane

Scaling up through things like partitioning/sharding/caching

Protect from data loss through backups/recovery, replication

Practice practice practice

Common tactics: draining, rate limiting, circuit breaking, etc

Focus: GCP, Customers

A platform **for products**

Cloud customers **necessarily make changes** to the system they pay for.

- Already 2 parties at minimum, **complex interactions** predicted

Mismatches of expectations may occur:

- un(der) documented behavior, emergent behavior
- failure modes, failure domains poorly understood (or explained!)
- "sunny-day engineering" (non-defensive programming)

Cultures of risk "dumping" (biz vs. dev vs. IT) still exist in Enterprise etc

Pathological	Bureaucratic	Generative
Responsibilities shirked	Narrow responsibilities	Risks are shared

DORA
DEVOPS RESEARCH & ASSESSMENT

I-sume, You-sume

"Information technology anomalies are frequently fundamental surprises. This is due to ***the difficulty in maintaining adequate mental models*** *of what is below the line, understanding how this connects to what is above the line -- crossing the line, as software systems grow in complexity and continuously change."*

STELLA report – 3.4.2

Between these two parties (Customer, Provider), **assumptions** abound.
The **Provider** must step forward:

1) recognize, understand this gap – *customer empathy*
2) improve UX of cloud products
3) produce education, documentation, advocation!

Pop Quiz! *(not graded)*

Q: Can you build **99.99%** services on **99.9%** infra?

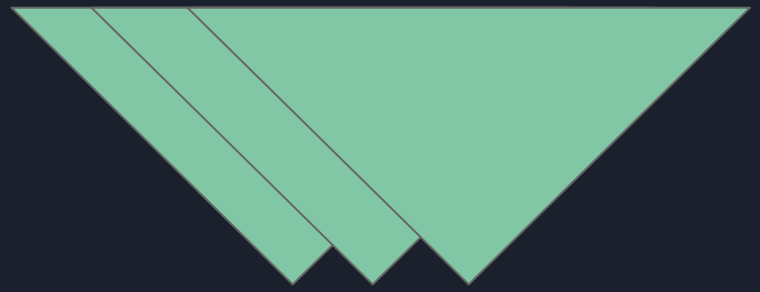
NBC Universal

Q: Can you build **99.99%** services on **99.9%** infra?

Yes.

You can build
more reliable things
on top of
less reliable things

a simple example: RAID. see: *The SRE I Aspire to Be*, @aknin SREconEMEA 2019



Component reliability:

- **Inherit reliability** from the base
- Lower levels *must be more* reliable
- "scale up"

Scalable reliability:

- Cost-effective base at scale
- Software *must improve* reliability
- "scale out"

Worked great, for a long time

Common mental model

Cloud is here, though.

(because scale, mostly)

((You can't buy more nines for your VM in Cloud))

Incidents (with Customers)

To a Cloud customer, Cloud Service Providers (CSPs) can be really great, until they aren't.

"Why did this happen?"

"Why were we the only ones affected?"

"Didn't you test for this?"

Understandable! Bad outcomes can be **brutal, scary!**

Aside: what's underneath GCP

All of Google runs on Borg, etc. Including GCP.

Search/Geo/Ads/Android/etc don't use Cloud Regions and Zones,
(They do use other similar abstractions)

⇒ So it's **not the infrastructure** that causes Cloud customer outages, then. Right?
→ Cloud **can certainly** be very reliable.
→ Services can be designed to withstand inevitable infrastructure issues.

Complexity, assumptions, misunderstandings. **It takes work**.

So, let's all just 5x9s ?

"Just run like Google," then.

But **there is no one way** that Google runs.

Customers yearn for a **simple answer**, looking to Google

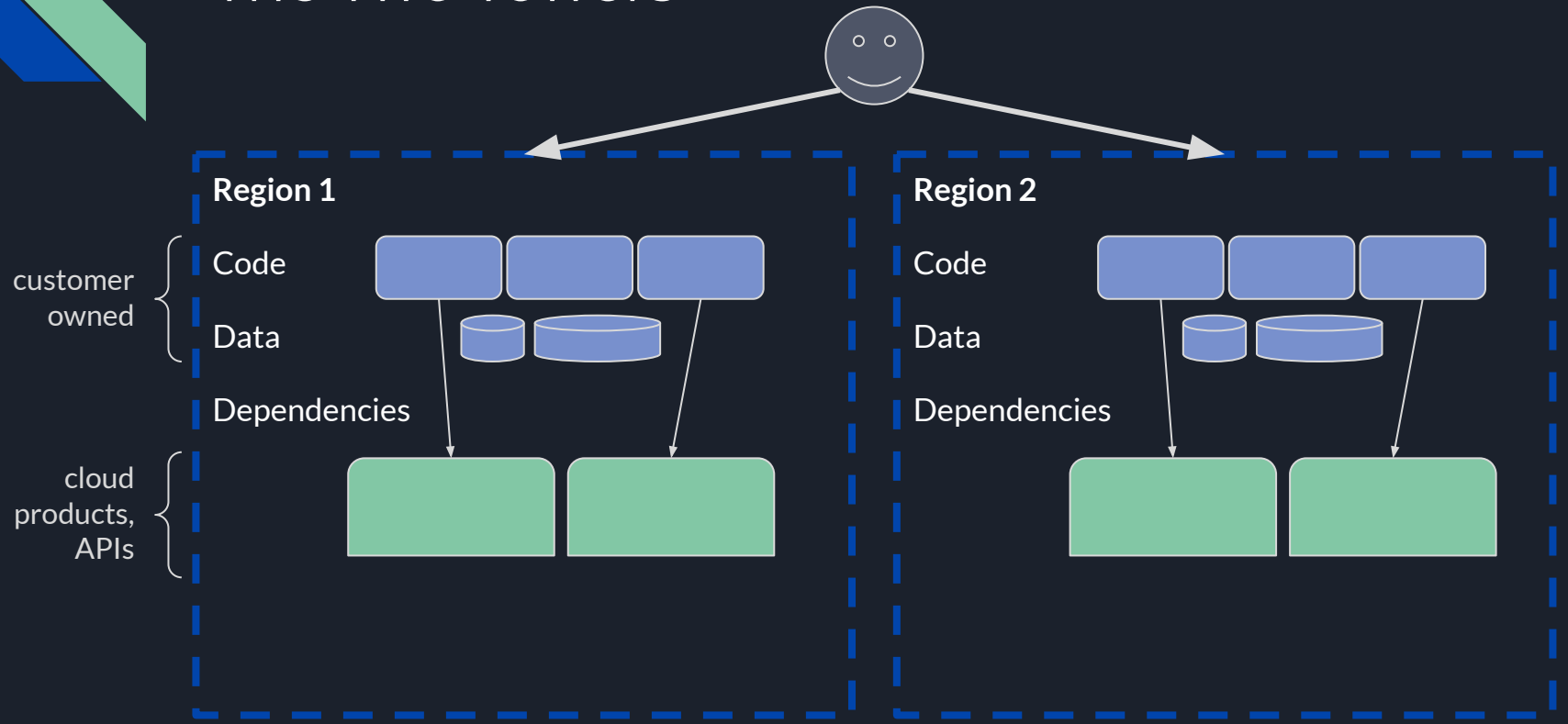
If there was one, it would be built into GCP on day one.

The secret, really:

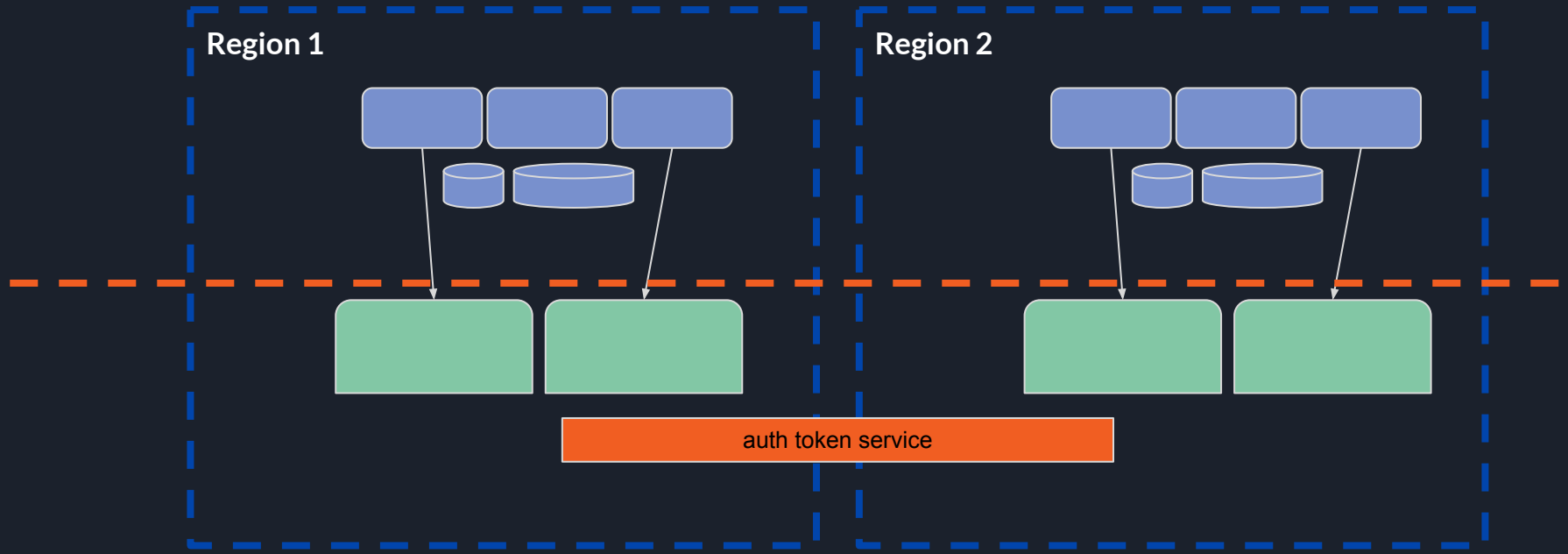
Learn from Incidents!
Apply learnings to platform, teams, process, ***repeat forever***.

Example Time!

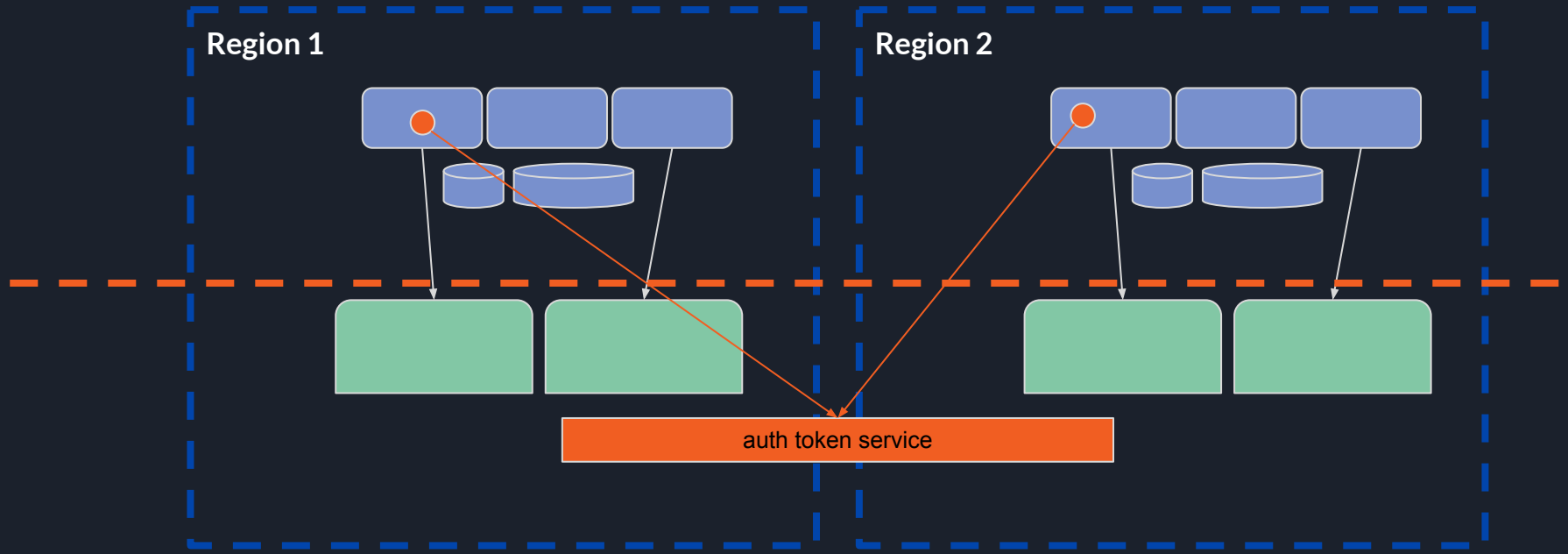
The Two Towers



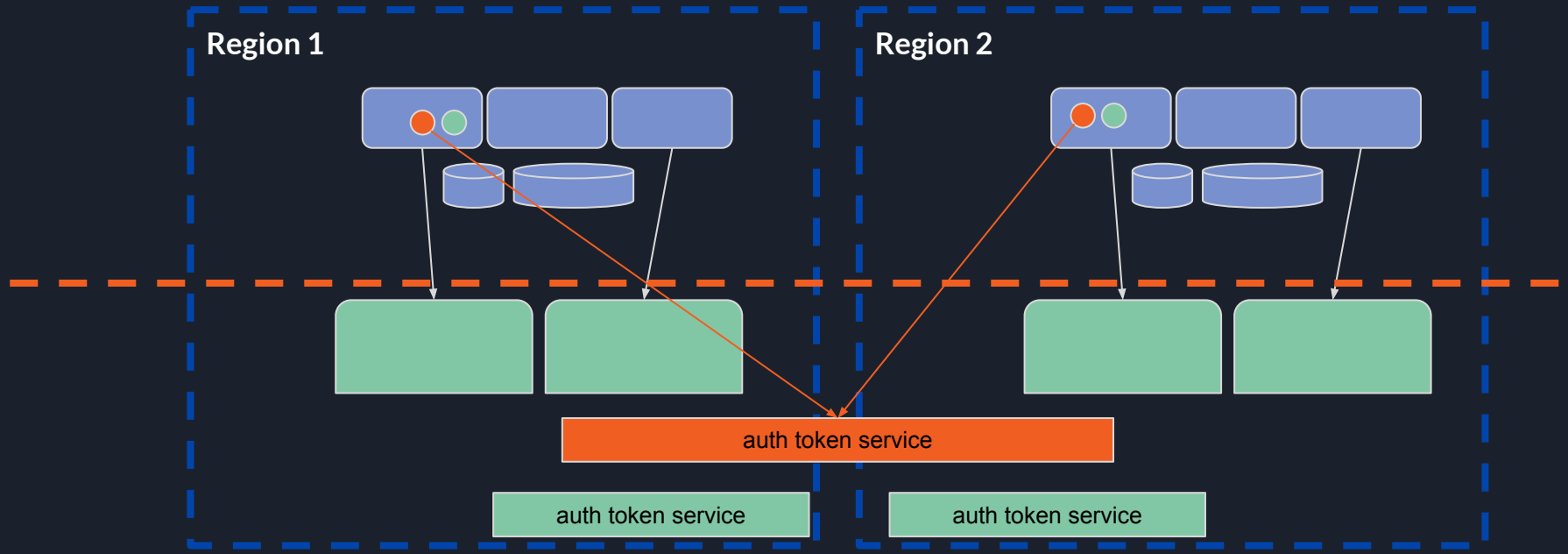
Tiny Problem (unnoticed)



Client Libraries are Hard



Versions Matter



Meanwhile...

Nobody knew all of this for *weeks*.

Single big customer, uniquely impacted

Demands investigation, followup, prevention.

> *"Never happens again"*
>
> *"Test for this"*
>
> *"Monthly updates"*

Immediate, huge response required

Measurement: Incidents, Risk

Managing, Measuring Incidents

~~Outage Management at Google~~ ("omg" !)
→ **Incident Response Management**

Tooling, comms, roles, procedure, metrics, doc templates, and

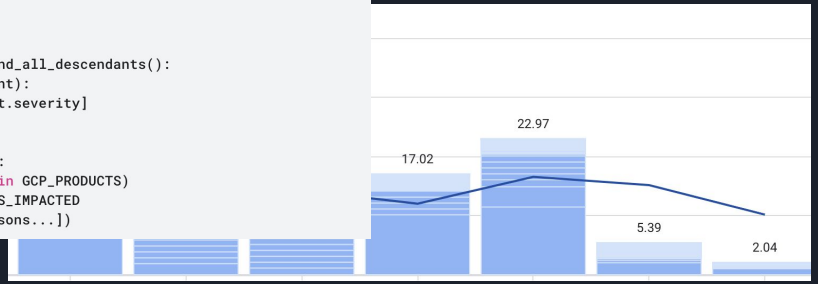
Incident impact measurement, aggregation, analysis:

like: Sev1, Sev2, …

```
OIS_WEIGHTS = {'❗MINOR':0.01, '❗❗MEDIUM':0.1, '❗❗❗MAJOR':1.0, '⚡HUGE':10.0}

def get_incident_ois(root_incident):
  total_ois = 0
  for incident in root_incident.self_and_all_descendants():
    if is_incident_a_disruption(incident):
      total_ois += OIS_WEIGHTS[incident.severity]
  return total_ois

def is_incident_a_disruption(incident):
  return ((incident.impacted_resource in GCP_PRODUCTS)
          and incident.outcome == USERS_IMPACTED
          and not [other exclusion reasons...])
```



Accuracy

Does this sizing **work**?

- "small" for Ads might be "huge" for Chrome,
- both stemming from a "tiny" incident in Infrastructure

Does consistent sizing **persist**? Adjusted later? Why?

- discovered **more impact** later
- matching **perceived effort**/urgency to reported impact

Does response **effort** match impact?

Affect **actions taken**? Teams **allowed** to be pulled in?

Identifying, Measuring Risk

Complex, Large systems : large surface area

Who identifies Risk?

- **Bottom-up**: service-owners, experts
- **Top-down**: repeated failure patterns, systemic issues
- Customer-centric: which Risks affect customers most?
  - Customer User Journeys (CUJs)
  - eg "create a cluster" or "my VM keeps running"

Risk Scoring

Initial simple scoring: **Likelihood * Impact**

- insufficient, unclear ("what does 0.12 *mean*?")
- NIST CVSS, etc

Align with **existing models** instead (eg infosec, fire safety)

~~Move Fast and Break Things~~
Define and Track Gradual Improvement

Weakness: (impact, likelihood, **asset / resource**)

Mitigation (weakness, reduction %, due-date)

Track progress per team, set OKRs, acknowledge commitments
Dashboards and rollups, reports and newsletters!

Our Problem > My Problem

Some risks aren't obvious to service owners, domain experts!
Systemic Risk is hard to identify and prioritize

Success story: FooService turndown (due to meta-analysis of many incidents)

- identify a large **Risk** in Foo backend service
- discovered significant Google-wide **dependency**
- **policy** set: track, influence, urge, incentivize
- 100k+ assets **mitigated**!

Programitization (*yawn?*)

Getting 100s of teams, 1000s of people to accomplish something … is **difficult**

now do it again and again, forever 😳

Motivation helps: existential, platform risk

Leadership matters!

Many attempts have been made, are ongoing

So What?

Does measuring actually help?

⇒ Focus our efforts, Do many things at once, Pareto

⇒ "Are we done yet?"

Unique to Google?

- Scale and Complexity
- Culture of **investigation** and **investment**
- Open to trying **new methods**

Futurework

STPA/CAST (Nancy Leveson, MIT) – a few teams so far

CAST on a postmortem ⇒ 10 new learnings, eg:

- one graph "backwards" – halted investigation
- initial impact assessment < 1% of actual

SLOs still do work well for component-level understanding
SRE is learning to look "between" the services as well

eg: Narayan Desai's beyond-SLOs work (SREcons, prodcast)

Outcomes, Real and Imagined

We don't expect incident **count** reduction
but we do expect incident **impact** (harm) reduction

So, is it working? Is **impact dropping**?

Is **this work** causing impact to drop?

What **is** working? Team are exhibiting different behavior:

- Using: Risk, Weakness, Remediations in planning docs
- User/Customer Impact as primary lens
- *Some* legitimately causal Risk reductions measurable

Learning can help **Focus**

Our teams want to **do good** by their customers.

Build empathy, understand and **solve real problems**.

A Powerful Motivator:

Being able to resolve a risk for a real customer, from end to end.

Conclusion

Managing Risks (and LFI) inside a Platform is HARD

Avoid focusing on **local maxima** (my product vs **our** platform)

- Incremental learning & improvement is key
- Continuously find new risks
- Transmit this understanding across orgs through education, shared code, enforceable policy, norms – this takes time

Don't wait for permission, but Leadership should "get it"

@stevemcghee

Thank you for your time!