Resilient Software in Practice

container **retry** load_balancer integration **canarying** circuit_breaker **delivery** API pub/sub **release** build **overload** traffic_shaping **feature_flag** **load_shedding** bulkhead autoscaling **SLO** RPC distributed_lock **monitoring** microservices **redundancy** CAP health_check **throttling** pre-scaling **alerting** ACL orchestration **timeouts**

Spring Flower and Snow

Resilient Software in Practice

GoJakarta/ScaleJakarta 2019

Monang Setyawan

Agenda

1. Background
2. Google Production Environment
3. Lessons Learned
4. QA

Background

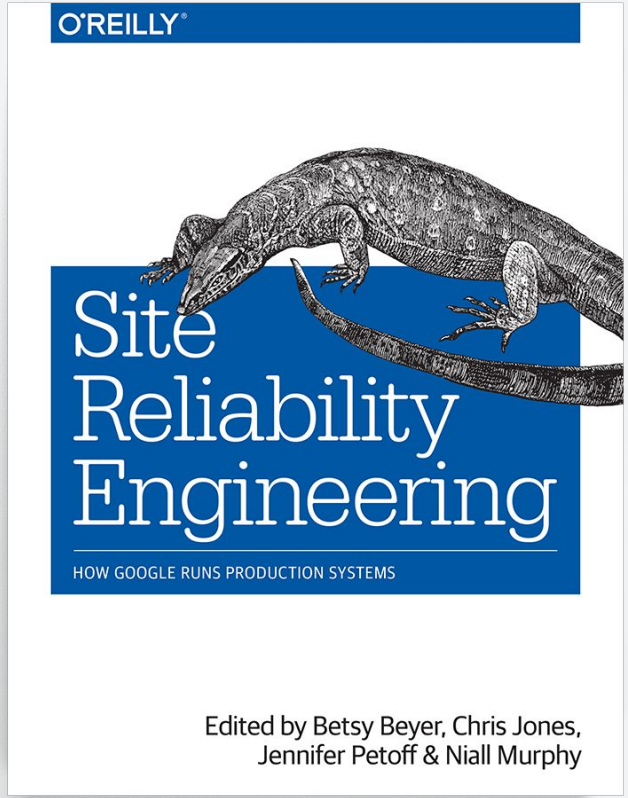
Why we are here

I’m not representing the Big G

- This presentation will self destruct in 3… 2… 1…
- I’m not trying to sell anything

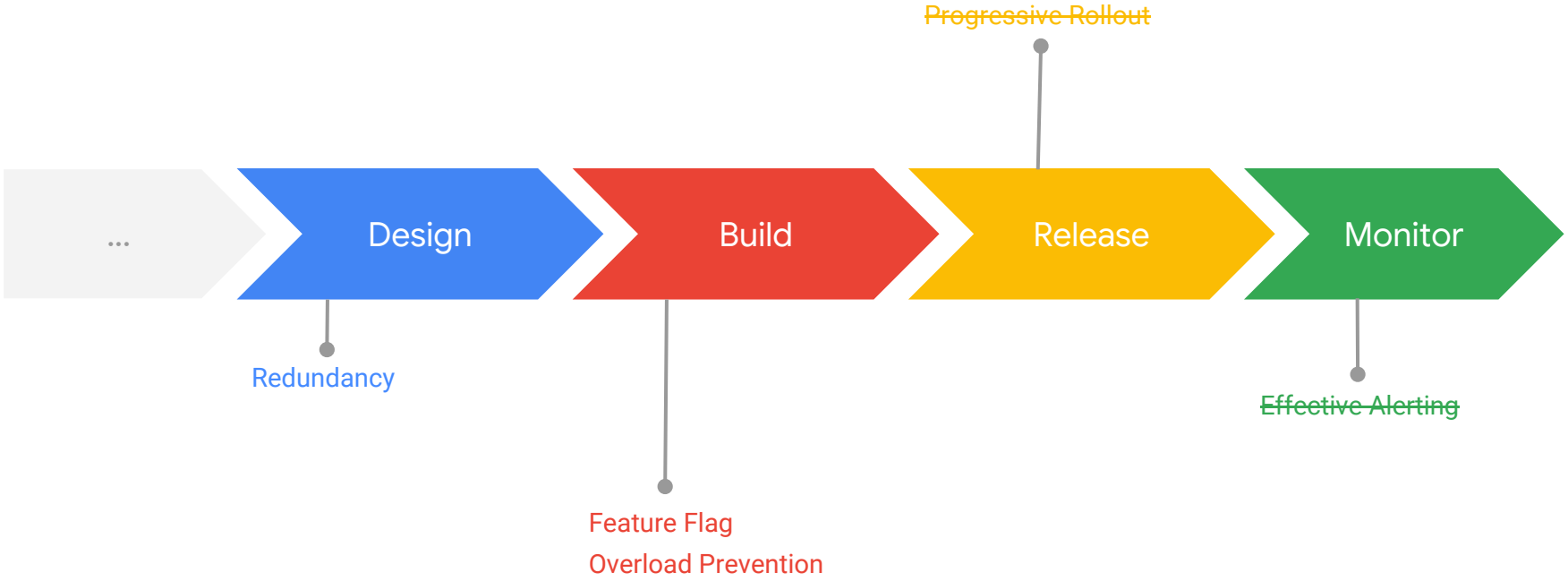
All the information is public

- Google SRE Book is awesome!
- Google Research Publications



Source: Google SRE Book

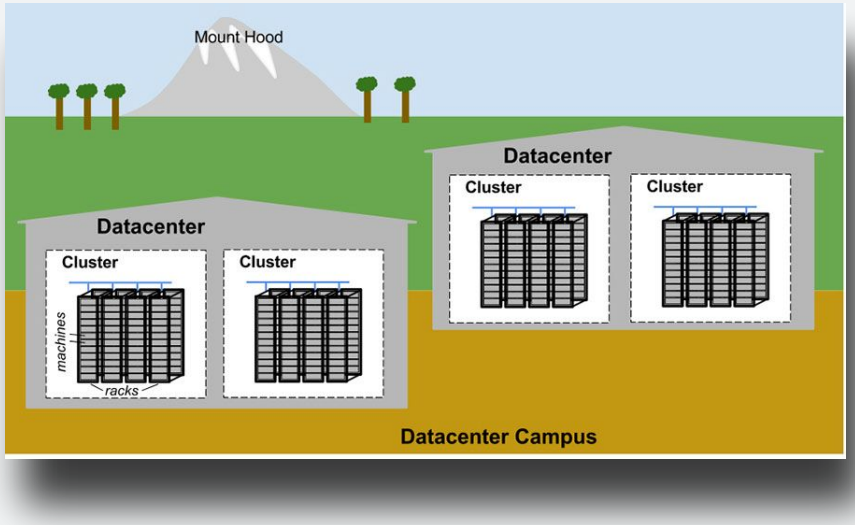
Phases


Progressive Rollout
...
Design
Build
Release
Monitor
Redundancy
Feature Flag
Overload Prevention
Effective Alerting

Google Production Environment

Hardware

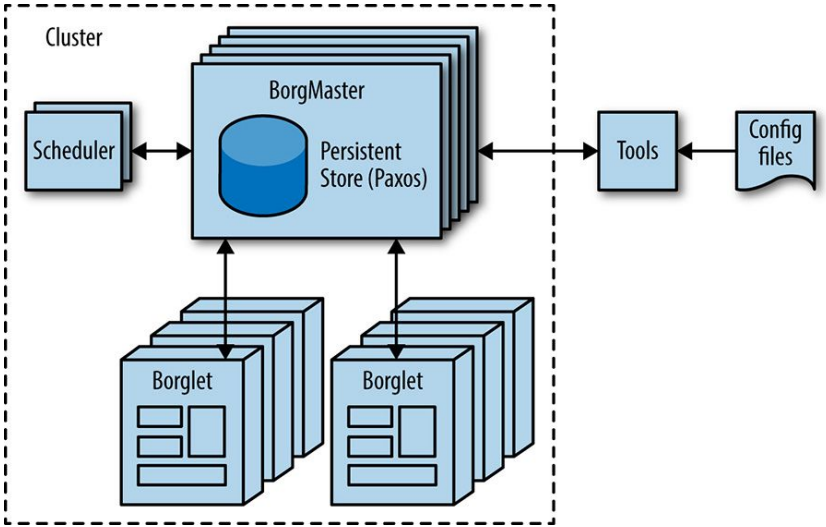
- Machine
- Rack
- Row
- Cluster (Cell)
- Data Center
- Campus
- Metro/Region
- Continent
- Planet



Source: Google SRE Book

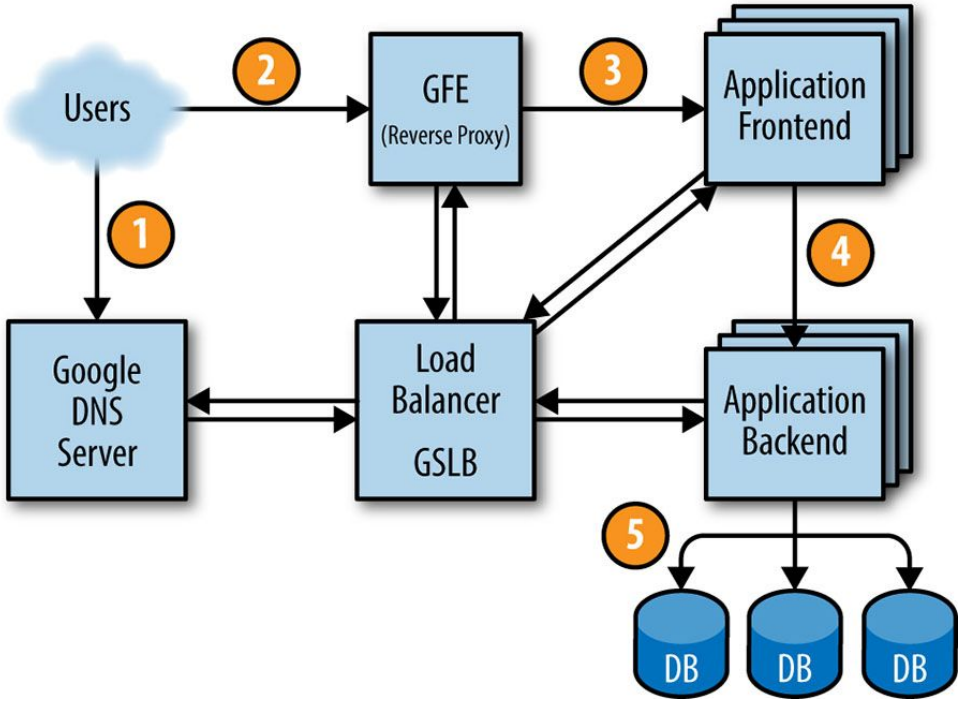
System Software

- Borg
- Job/Alloc/Task (k8s: Service/Pod/Label)



Source: Google SRE Book

Life of a Request



Source: Google SRE Book

Lessons Learned

Lesson 1: Redundancy

Why?

- Case for update
- Failures are common

At which level?

- Cluster and Region

Lesson 1: Redundancy

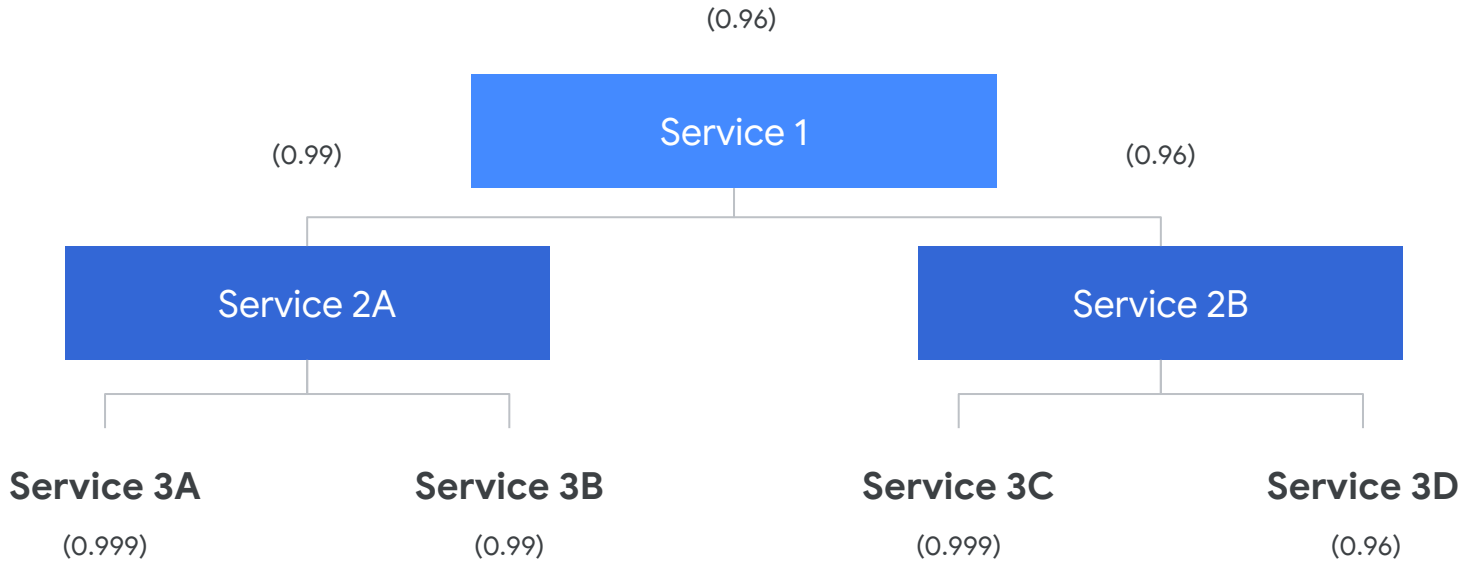
How redundant?

Define:

- Service Level Indicators (SLI)
- Service Level Objectives (SLO)
- Service Level Agreements (SLA)

> You're only as available as the sum of your dependencies.

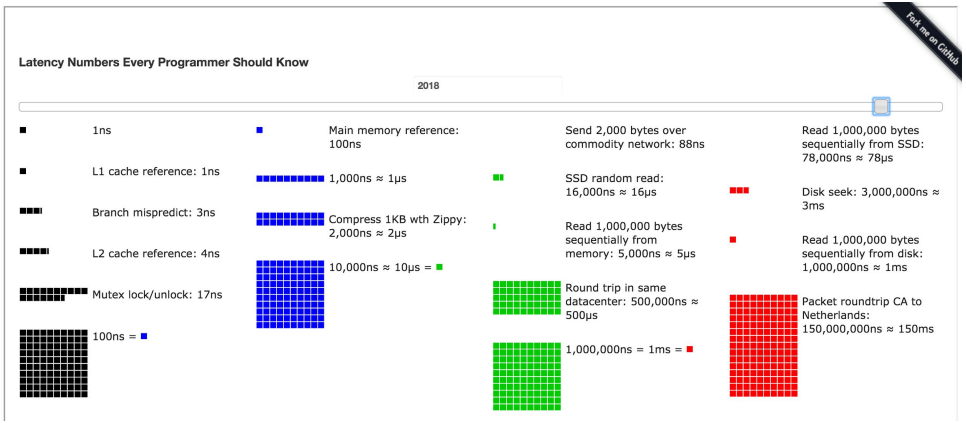
Treynor et al.,
The Calculus of Service Availability


(0.96)
Service 1
(0.99)
(0.96)
Service 2A
Service 2B
Service 3A
(0.999)
Service 3B
(0.99)
Service 3C
(0.999)
Service 3D
(0.96)

Lesson 1: Redundancy

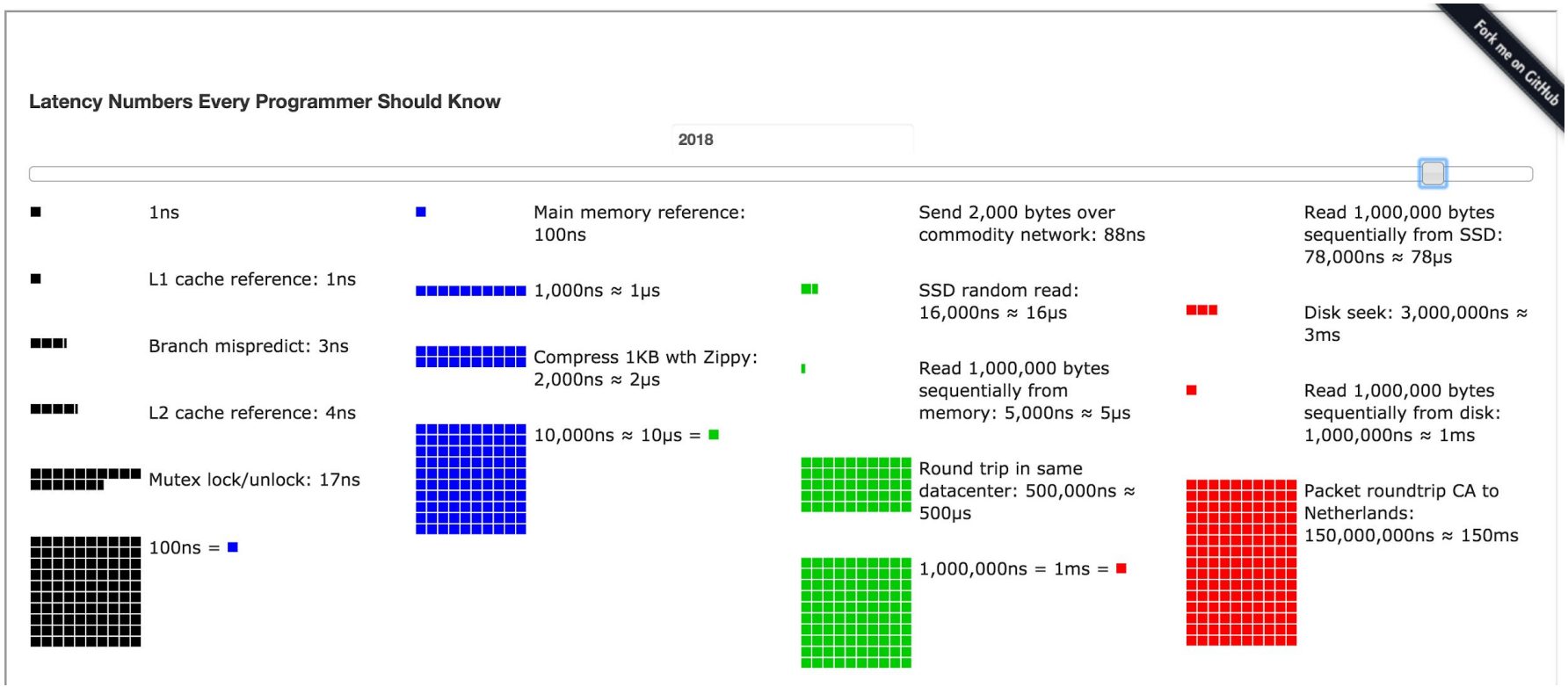
Capacity Planning - Use SLOs as Input

- Back-Of-The-Envelope Calculation (analytical)
- *or* Load Test (empirical)



Source: https://people.eecs.berkeley.edu/~rcs/research/interactive_latency.html

Latency Cheat Sheet



Source: https://people.eecs.berkeley.edu/~rcs/research/interactive_latency.html

Lesson 2: Launch with Feature Flag

Why?

- Ramping up/down new code path without release
- Some use A|B testing framework

Lesson 2: Launch with Feature Flag

How?

```
long mult(long a, long b){
  if (isFeatureEnabled("use-quantum-gpu")) {
    return qmultiply(a, b);
  }
  return a * b;
}
```

Lesson 2: Launch with Feature Flag

```
boolean isFeatureEnabled(String featureName)
```

- Can be based on request (user, cookies, location)
- Gradual rollout: use modulo operation

Ramp Up Scenario

Define key monitoring metrics.

Turn the knob to 1%.

Look at the metrics. Confirm expectations.

Metrics look good. Turn the knob to 10%.

Lesson 3: Handling Overload

Why?

Most of the time overload is inevitable

- Success disaster
- Unexpected spike
- Regression
- New service
- Erratic client behavior

Lesson 3: Handling Overload

How?

Serve degraded responses

- Cached response
- Approximated result
- Defaults

Lesson 3: Handling Overload

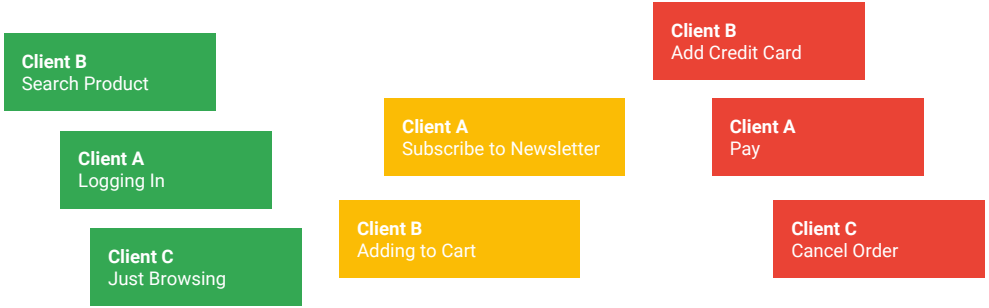
How?

Drop some requests (Load Shedding)

- Server utilization instead of QPS (throughput)
- Client quota
- Request ranking (Criticality)
- Tell client to (not) retry

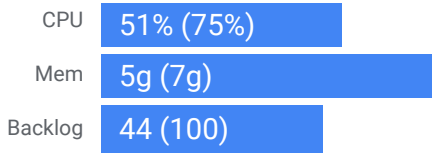
Lesson 3: Handling Overload

Requests



Server

Utilization (Cap)



Quota Remaining

Thank You

QA