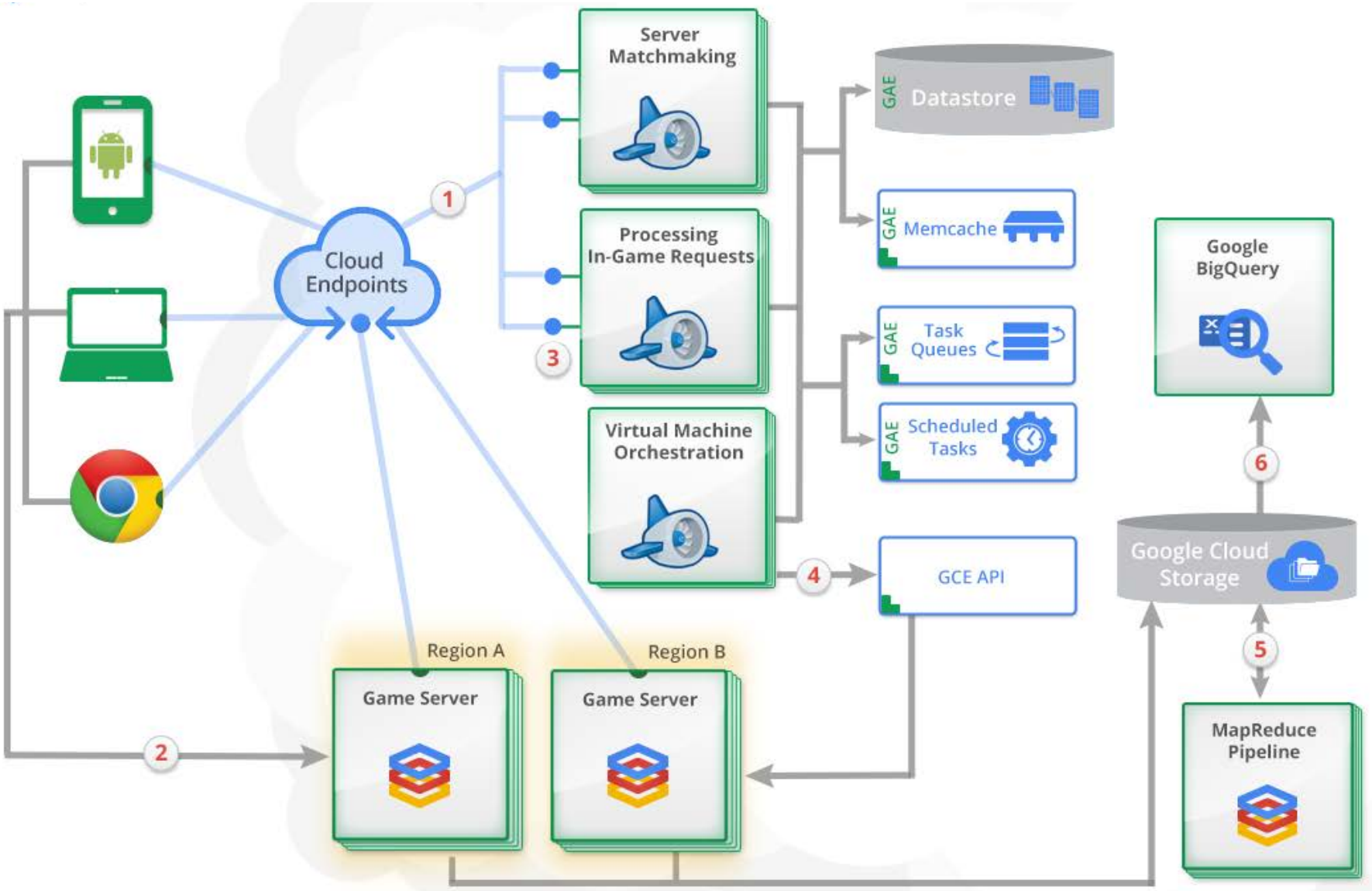
A few control issues in warehouse-scale computing

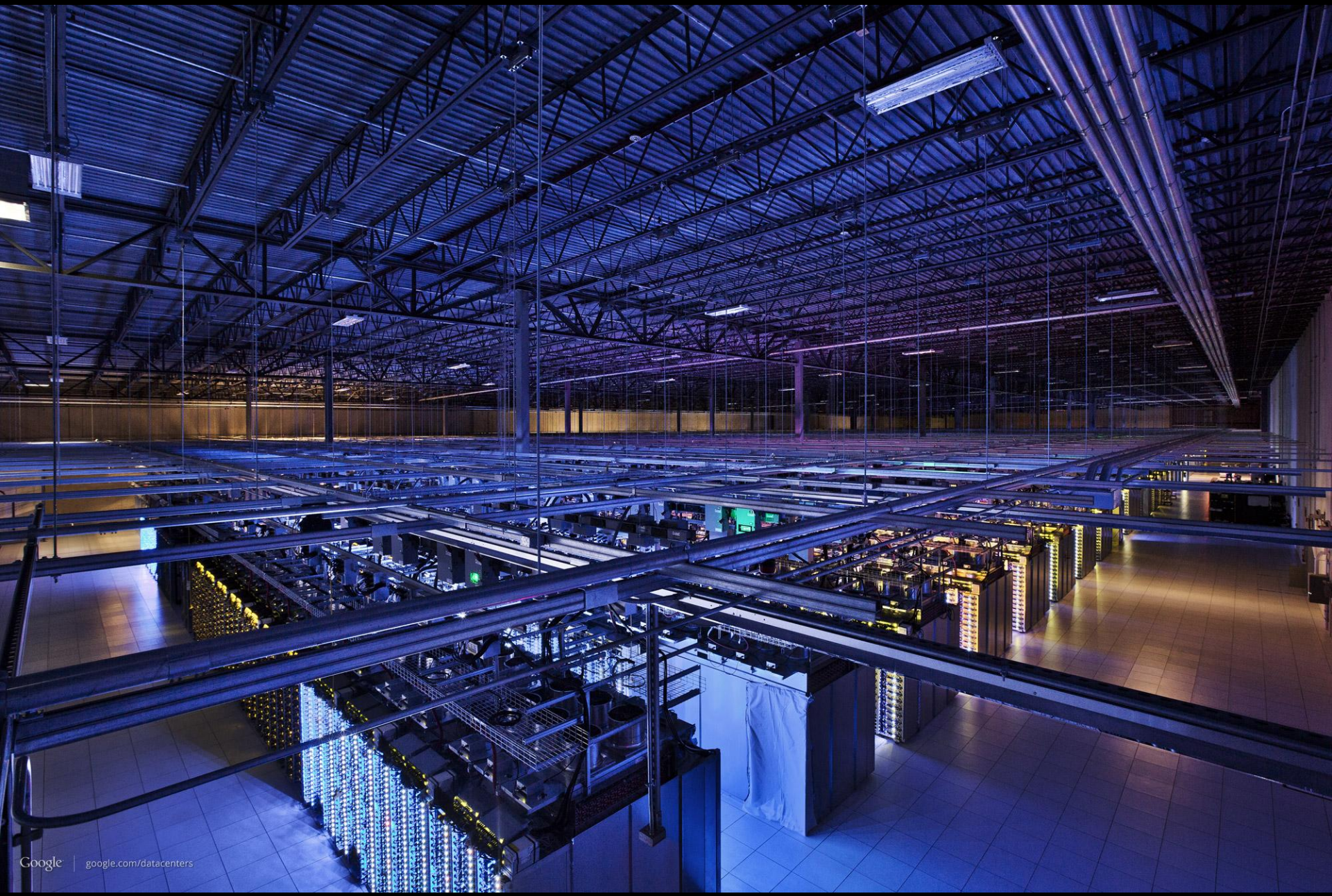
john wilkes

Cloud Control Workshop, London, UK

December 2014

Design for a game on Google Compute Platform




Google | google.com/datacenters

What's so hard? BigTable

- how big should tablets (units of data) be?
- when should they be split?
- where should they be placed?
- when should compactions happen?
- how many layers of SSTables should there be?
- thread pools: how big?
- how to prioritize traffic?
- what should be cached in RAM / on flash?
- ...

What's so hard? Gmail

A sample (very short) application stack:

- Gmail …
- uses BigTable
- which uses Colossus (GFS v2)
- which uses D (disk server)

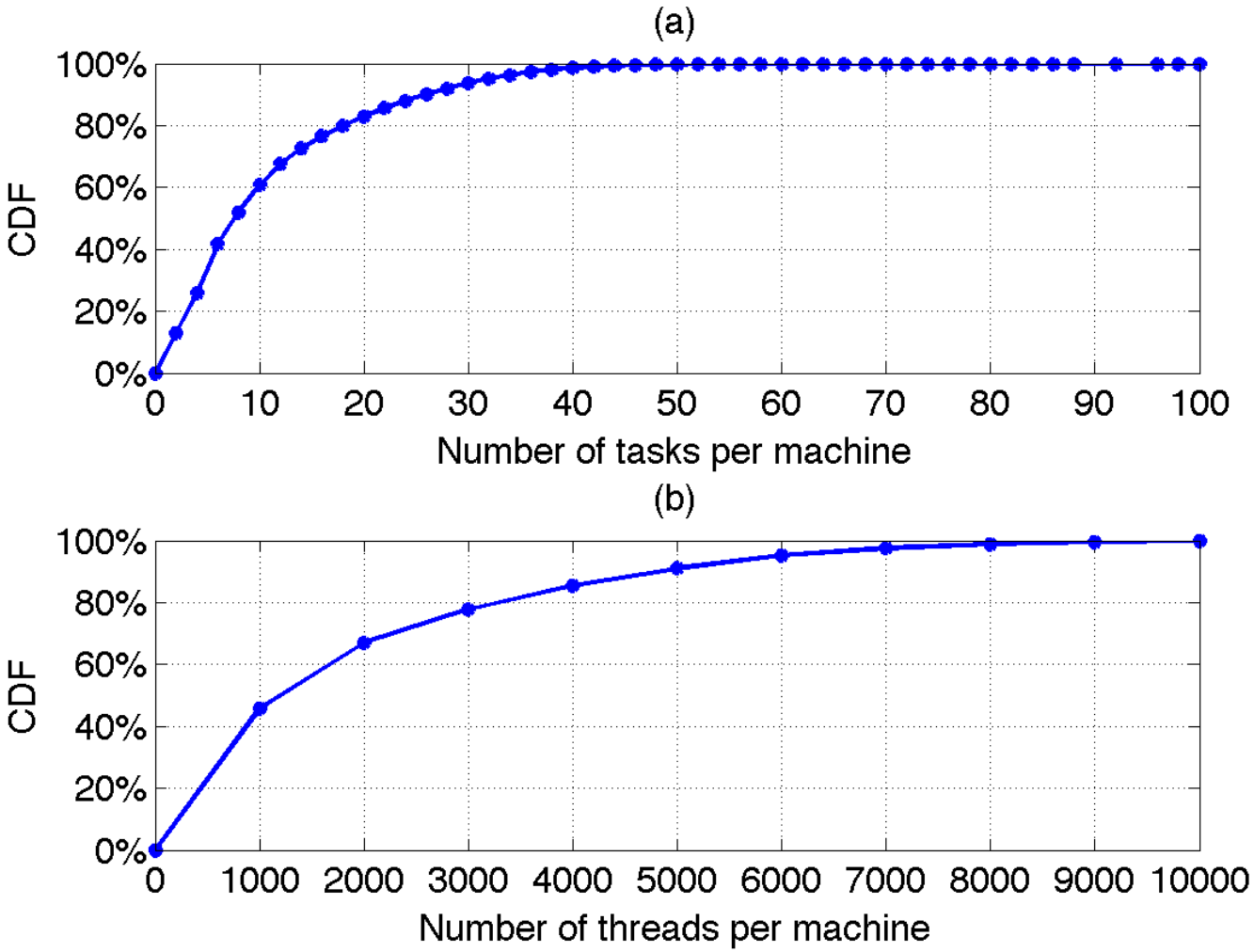
all rely on cluster manager, Chubby, network, ...


Google
google.com/datacenters

The problem

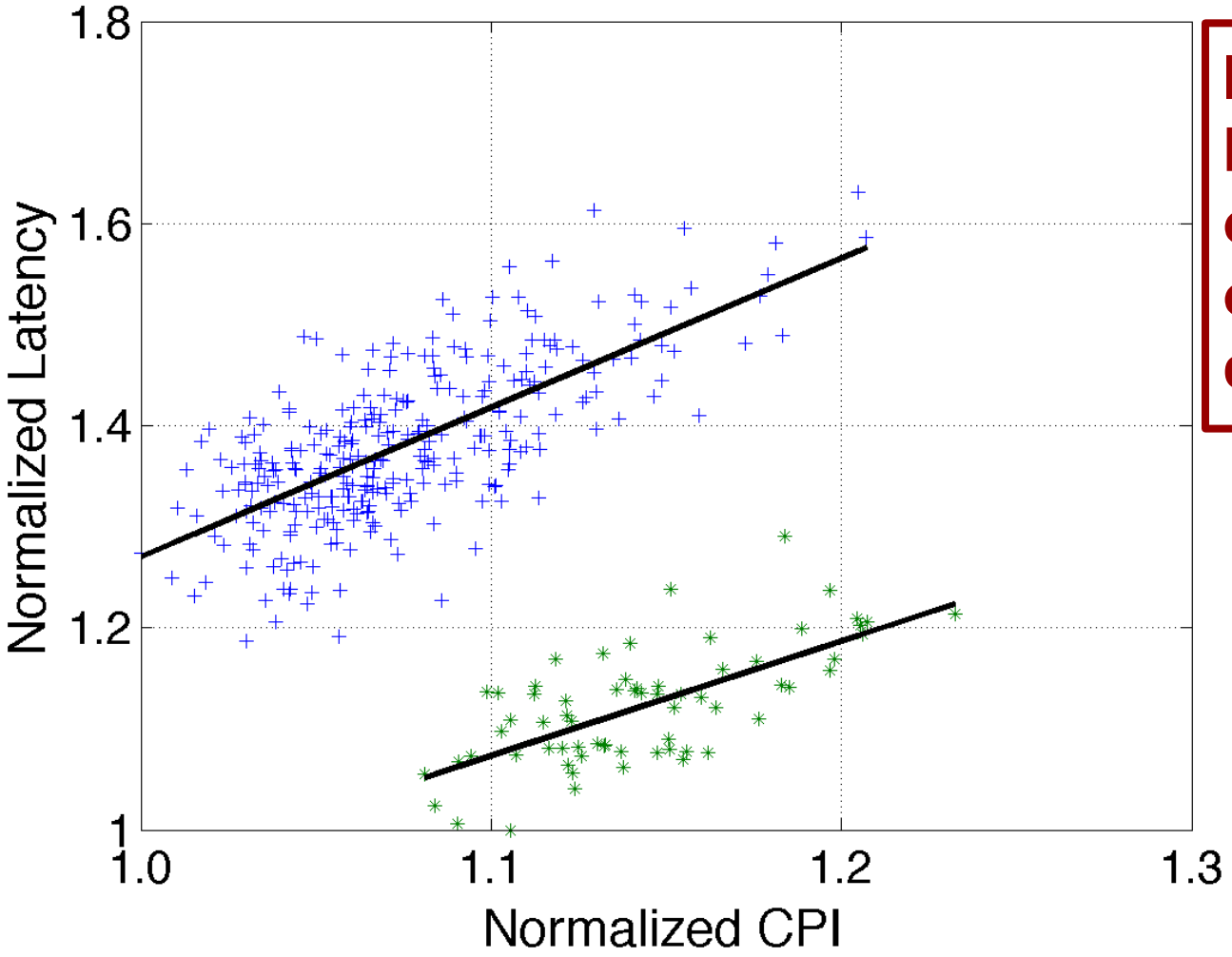
From: **CPI^2: CPU performance isolation for shared compute clusters**. EuroSys'13.

high utilization => resource sharing



The problem

resource sharing => interference



Interference happens tens of thousands of times per day

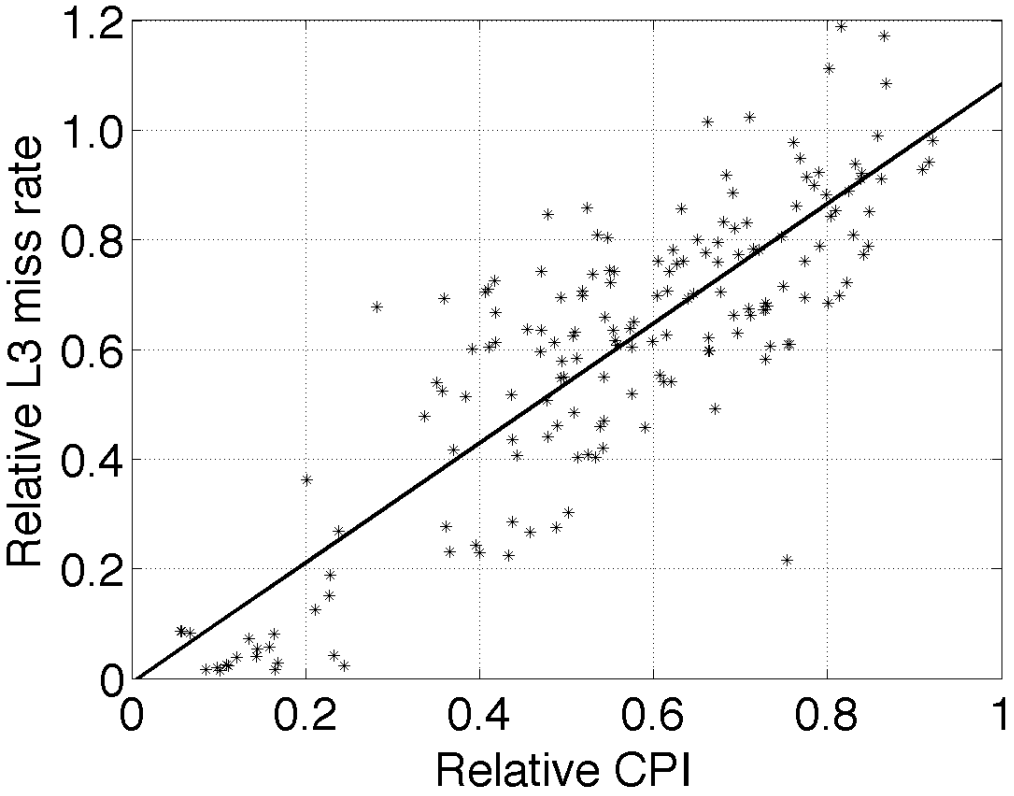
Our solution: CPI^2

a simple control system

1. Monitor *Cycles Per Instruction* (**CPI**)
2. Learn anomalous behaviors
3. Identify a likely antagonist
4. Throttle it to shield victims

Why use CPI?

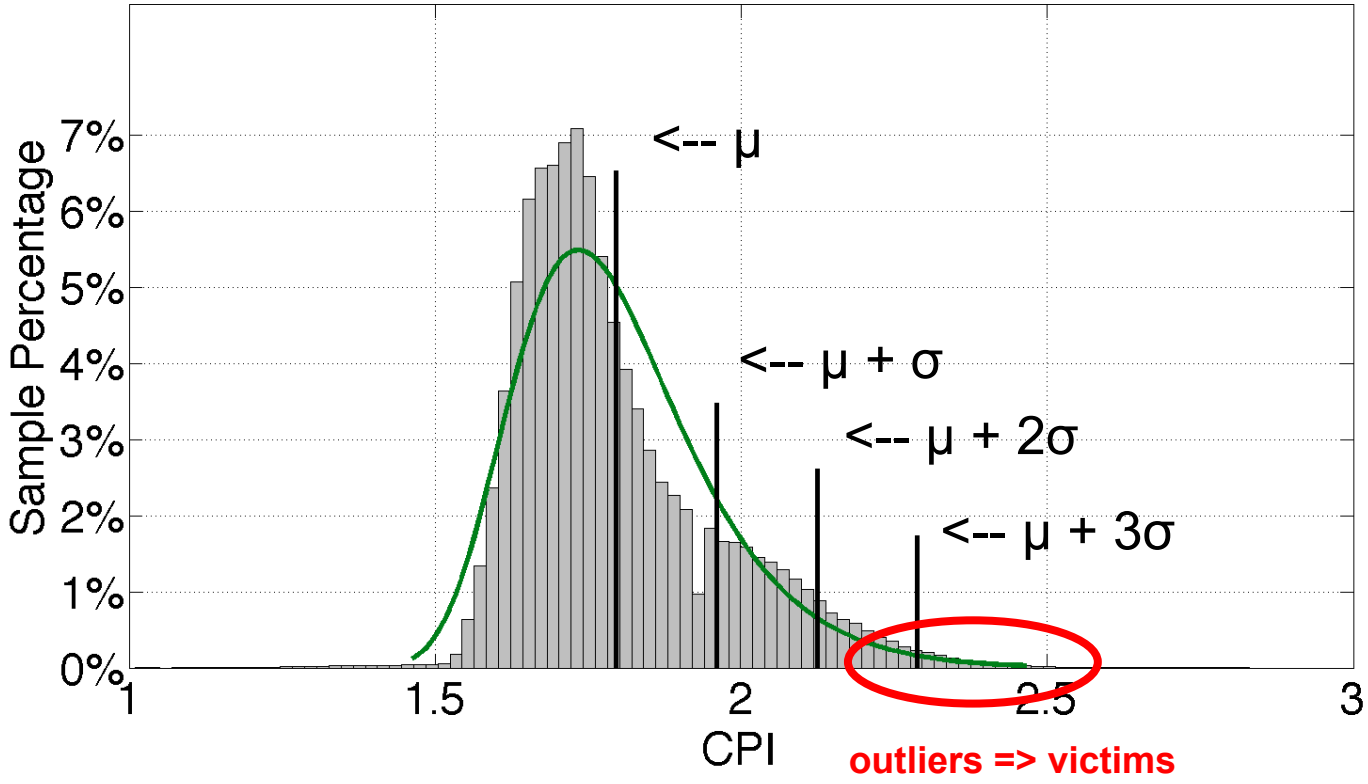
- It's cheap: < 0.1% CPU overhead, invisible to users
- It's stable (across time and space)
- It correlates well with L3 cache miss rate



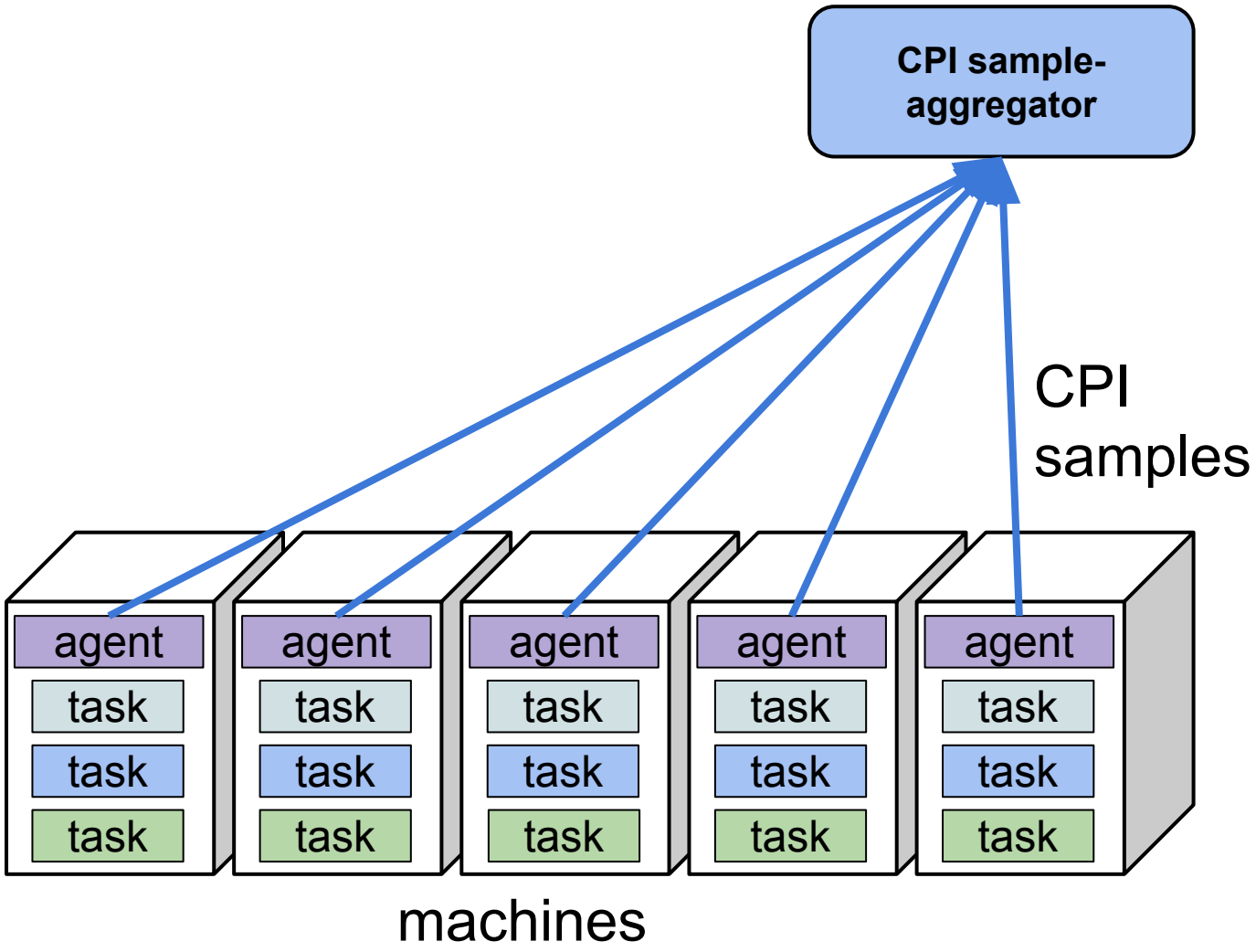
Gathering CPI

Build a CPI profile for a job

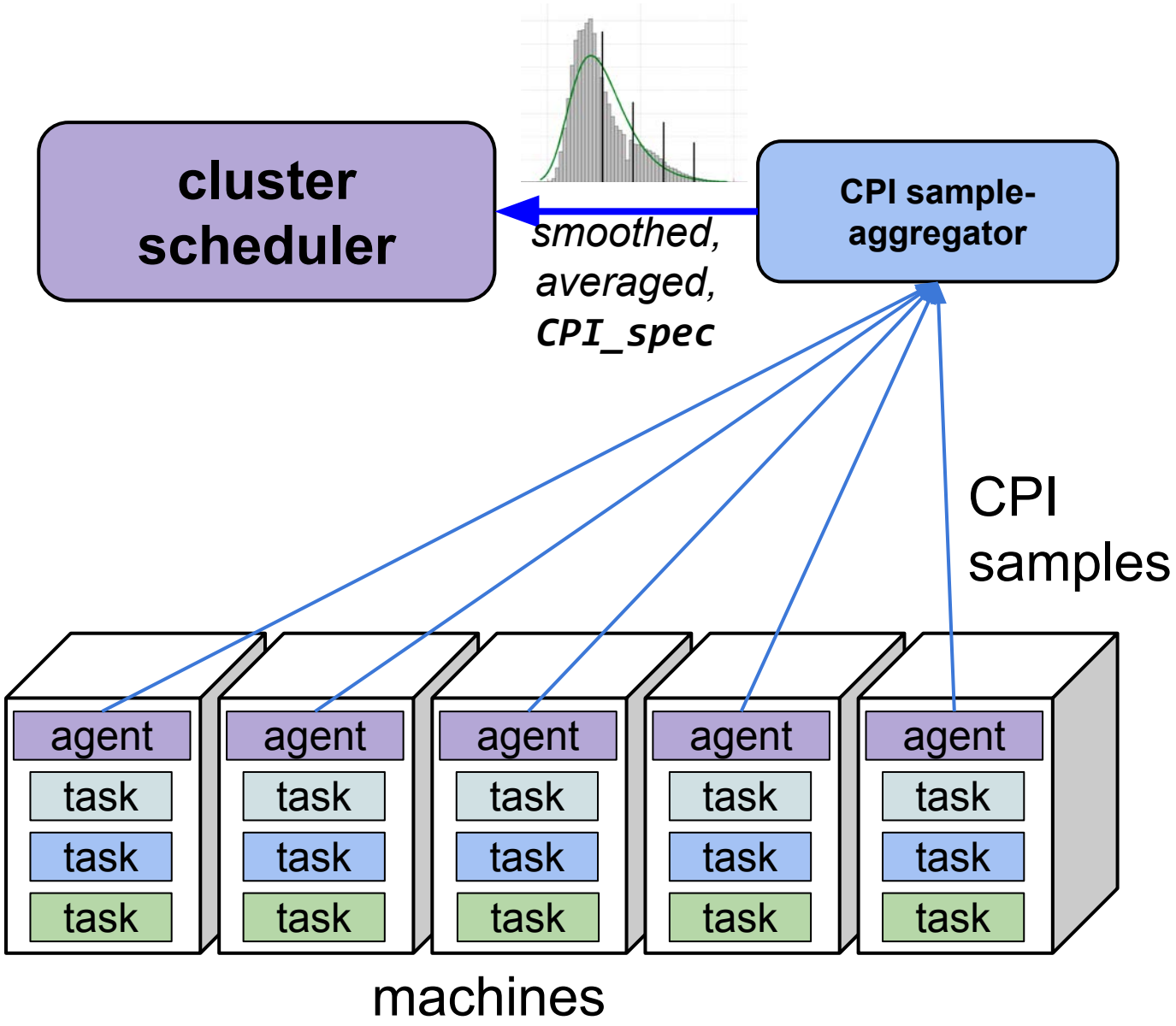
- per-cluster, per-platform
- mean (μ) & stddev (σ)



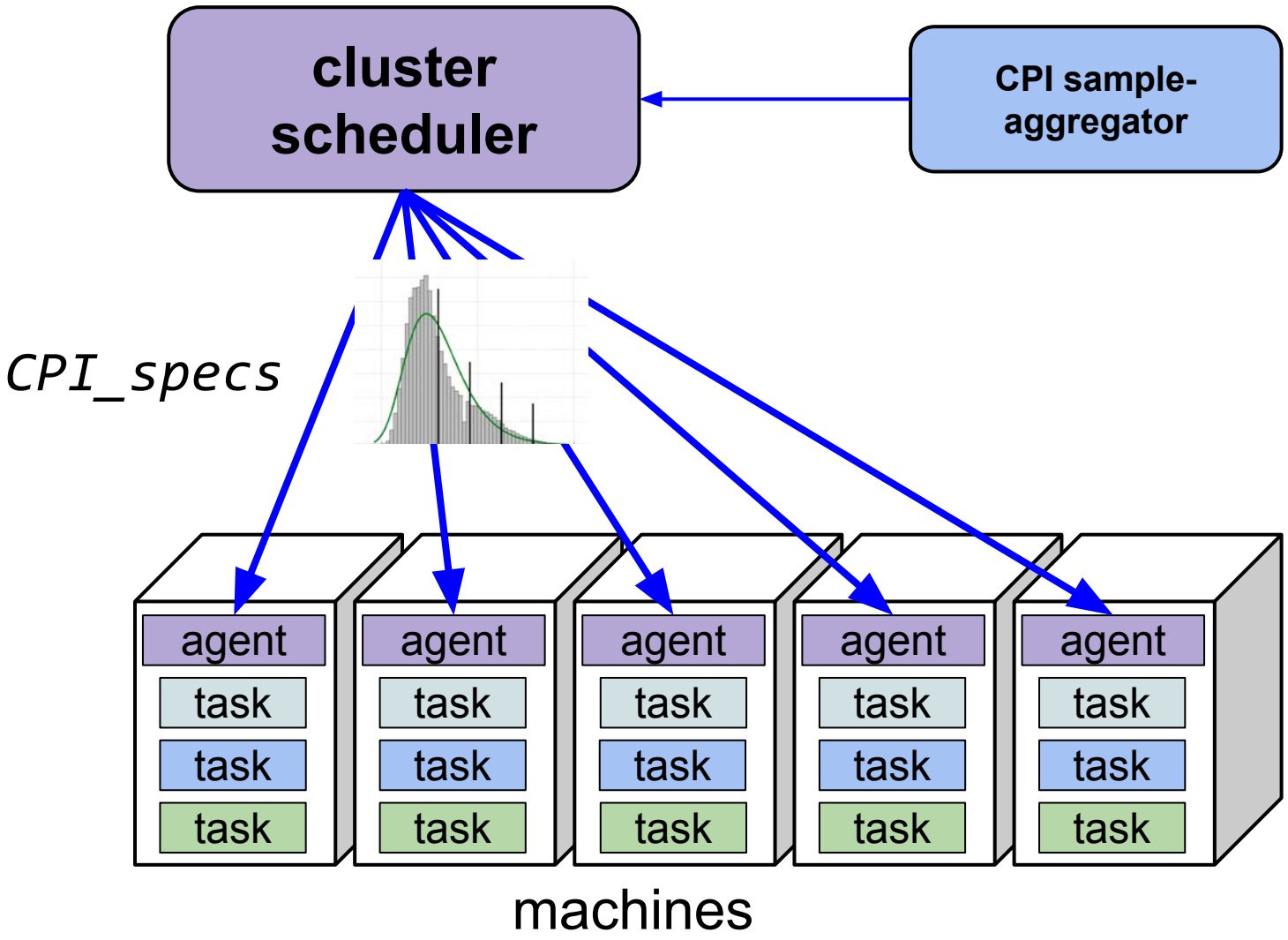
Gathering CPI



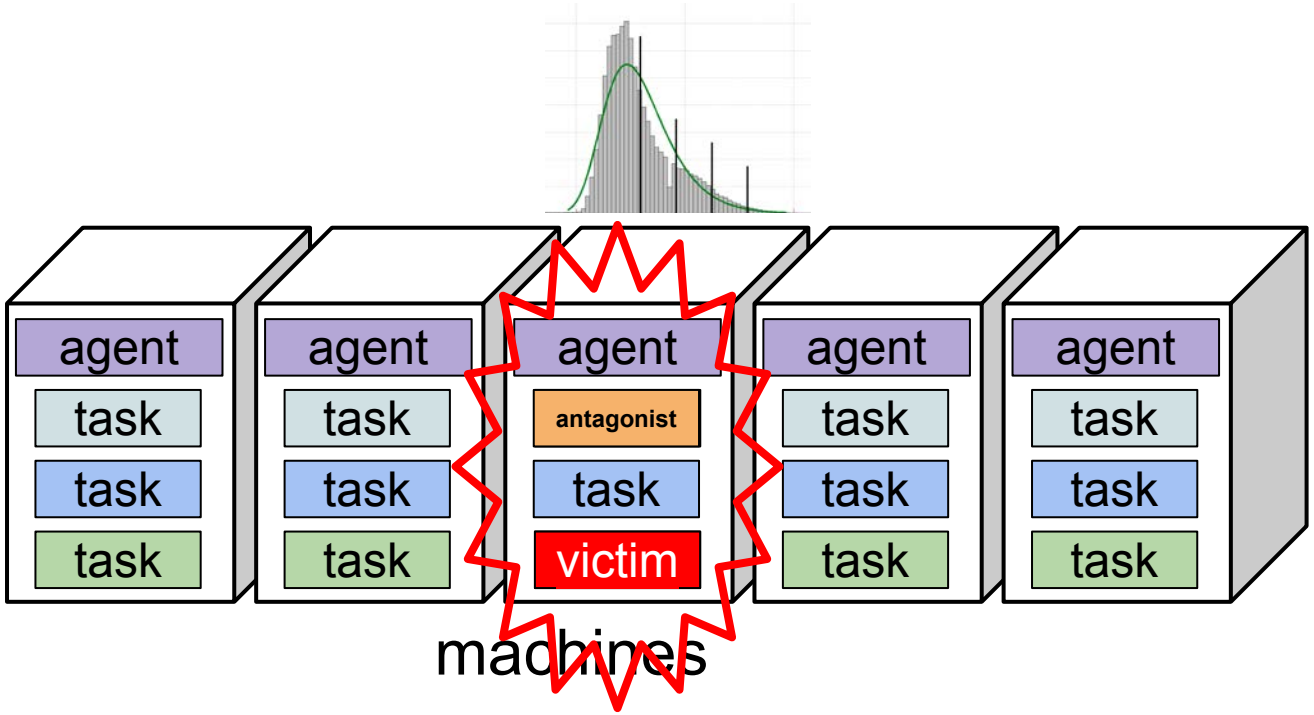
Gathering CPI



Using CPI to detect an anomaly



Using CPI to detect an anomaly



Now what?

Goal: reduce the effect of the antagonist

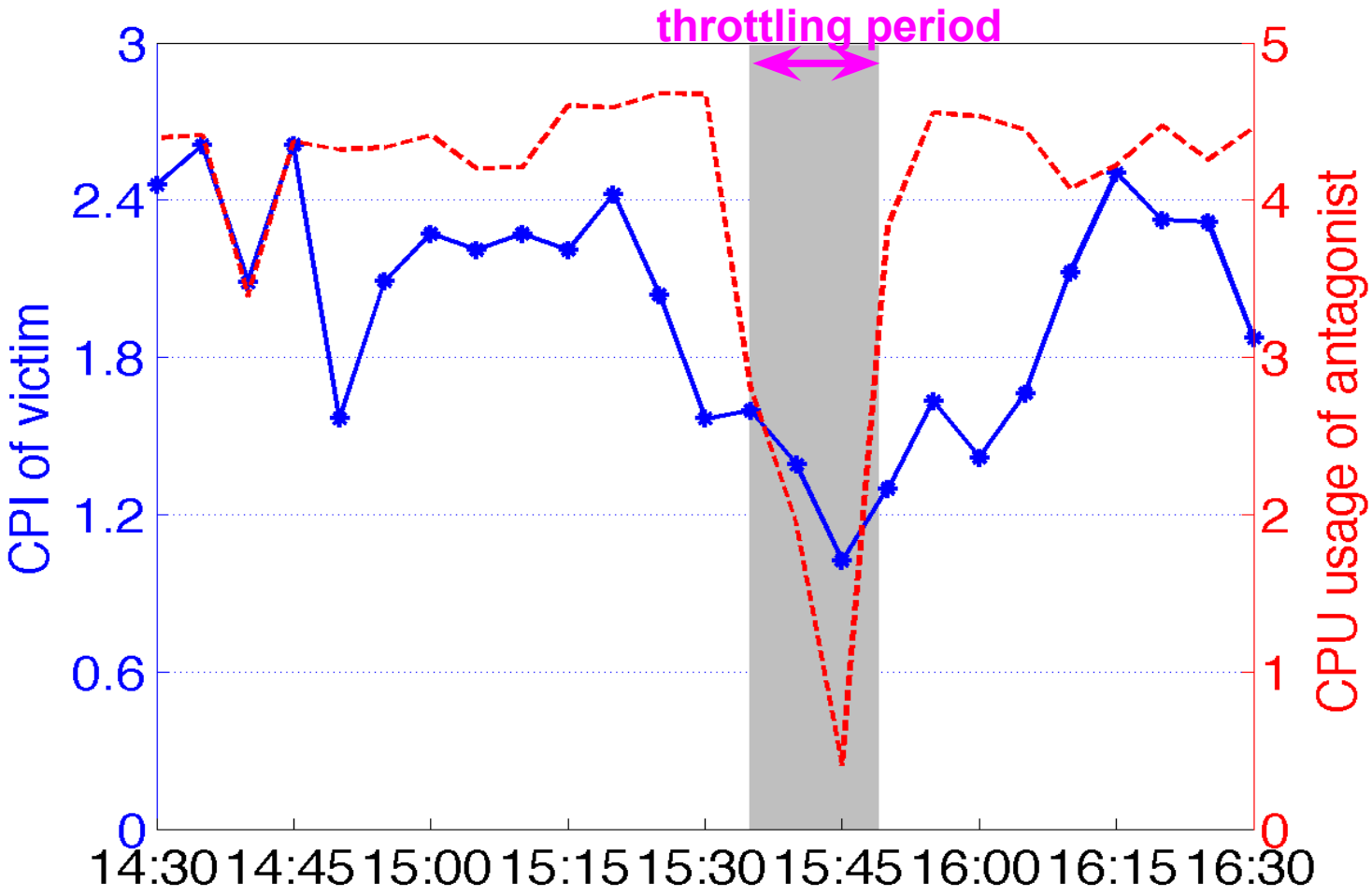
Let's **throttle** the antagonist!

- CPU hard-capping: 0.1 core for 5 minutes

Restrictions:

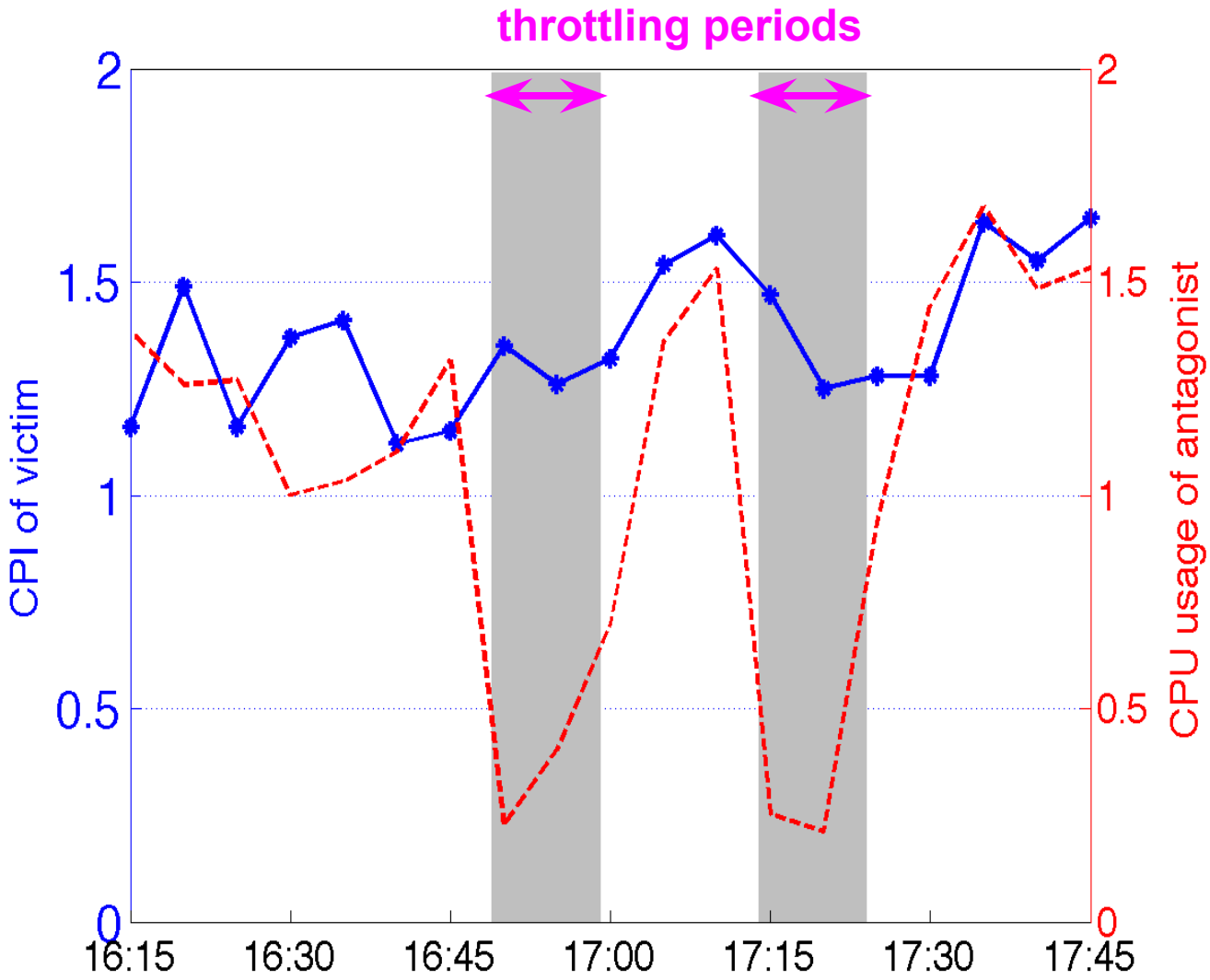
- only throttle batch jobs
- only help "important" victims

A motivating example



What could *possibly* go wrong?

A not so good example



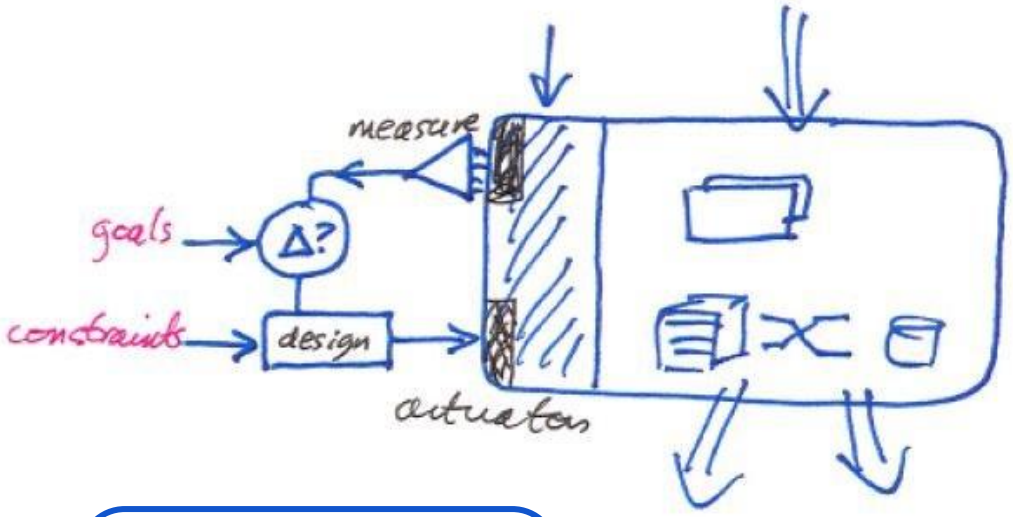
Maybe batch-only was a bad idea?
After all: LS tasks have *load balancing*

A control system to achieve:

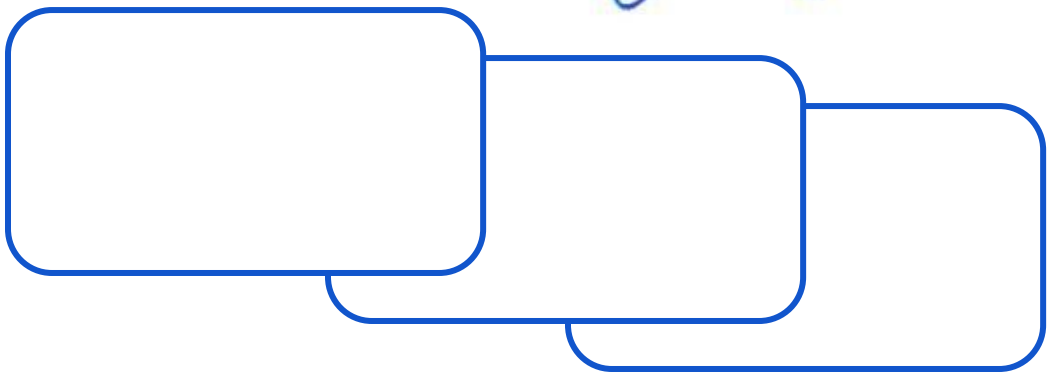
- failure tolerance (of server, of cluster)
- equal load (e.g., qps)
- equal performance (e.g., latency)

Maybe batch-only was a bad idea?

After all: LS tasks have *load balancing*



Load balancer



Service

Overload

What does your system do?

Tip: don't send all traffic to the first place on your list

Maybe batch-only was a bad idea?
After all: LS tasks have *load balancing*

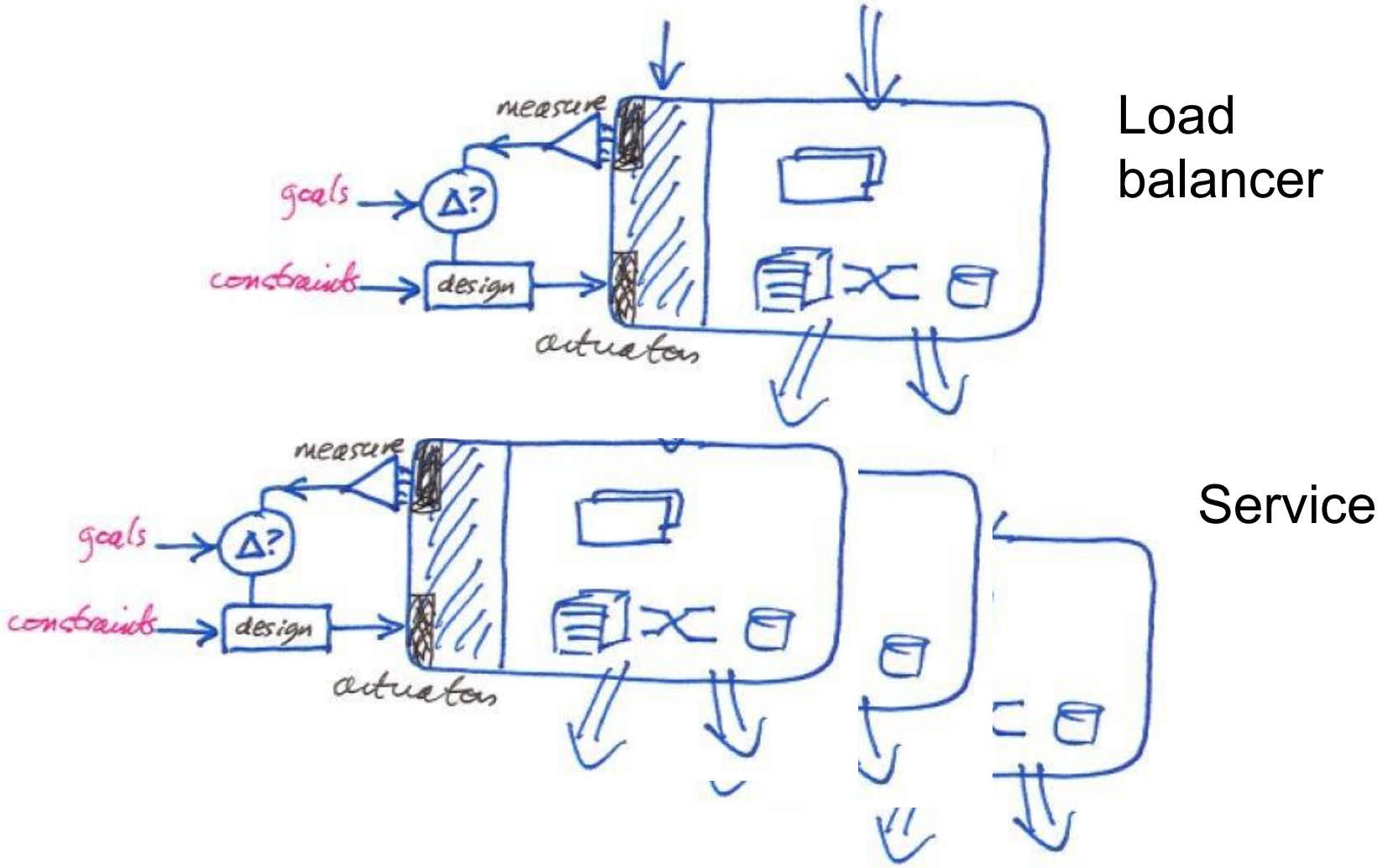
Cascading failures

1. Overload-induced outage
   - busy cluster => oops

2. No worries! Shunt load elsewhere!
   - busy cluster => much oops (repeat)
   - e.g., Gmail outage, 2009-02-24

Maybe batch-only was a bad idea?

After all: LS tasks have *load balancing*



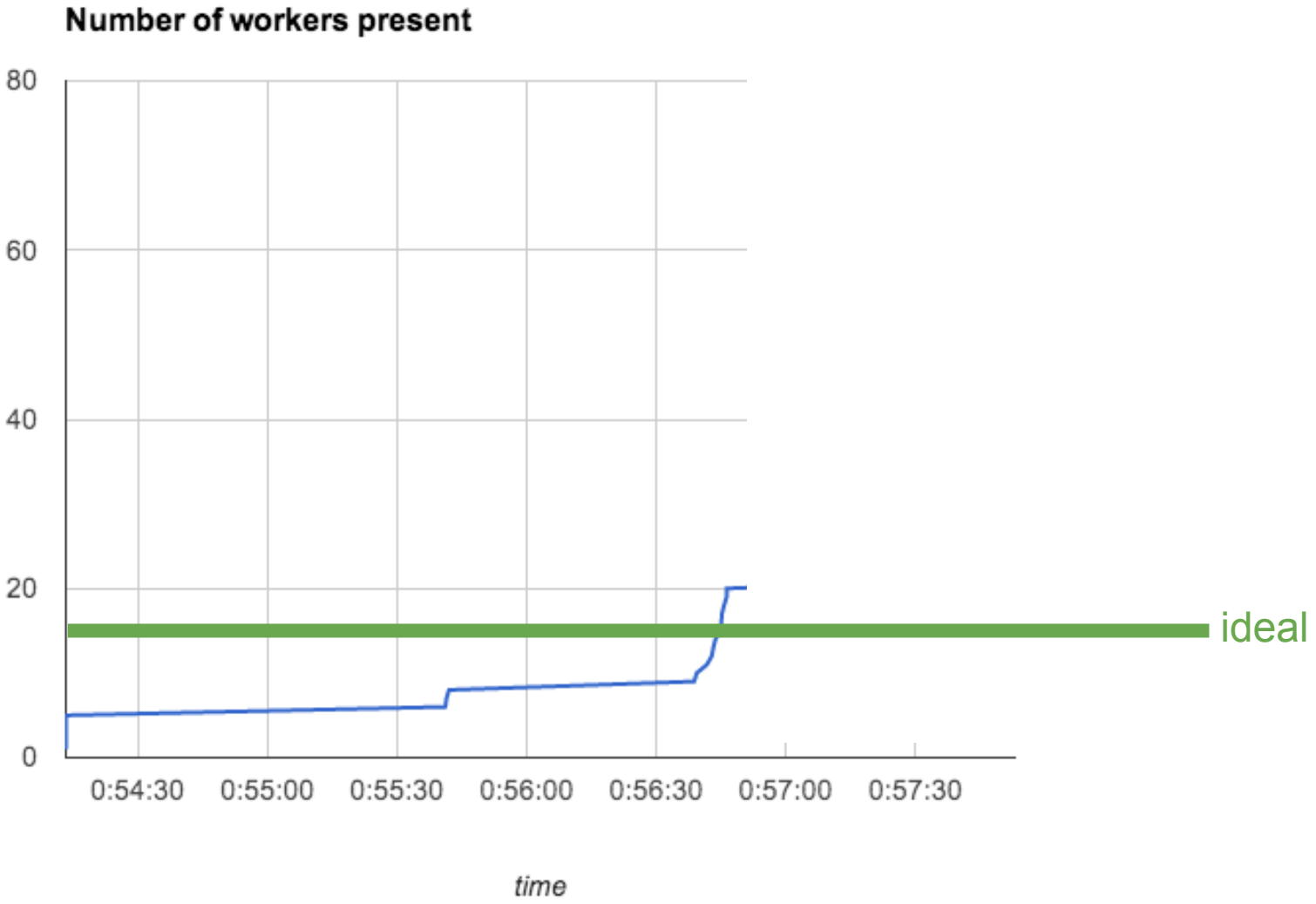
Interacting control loops

1. Load-placement
   - few-second response times

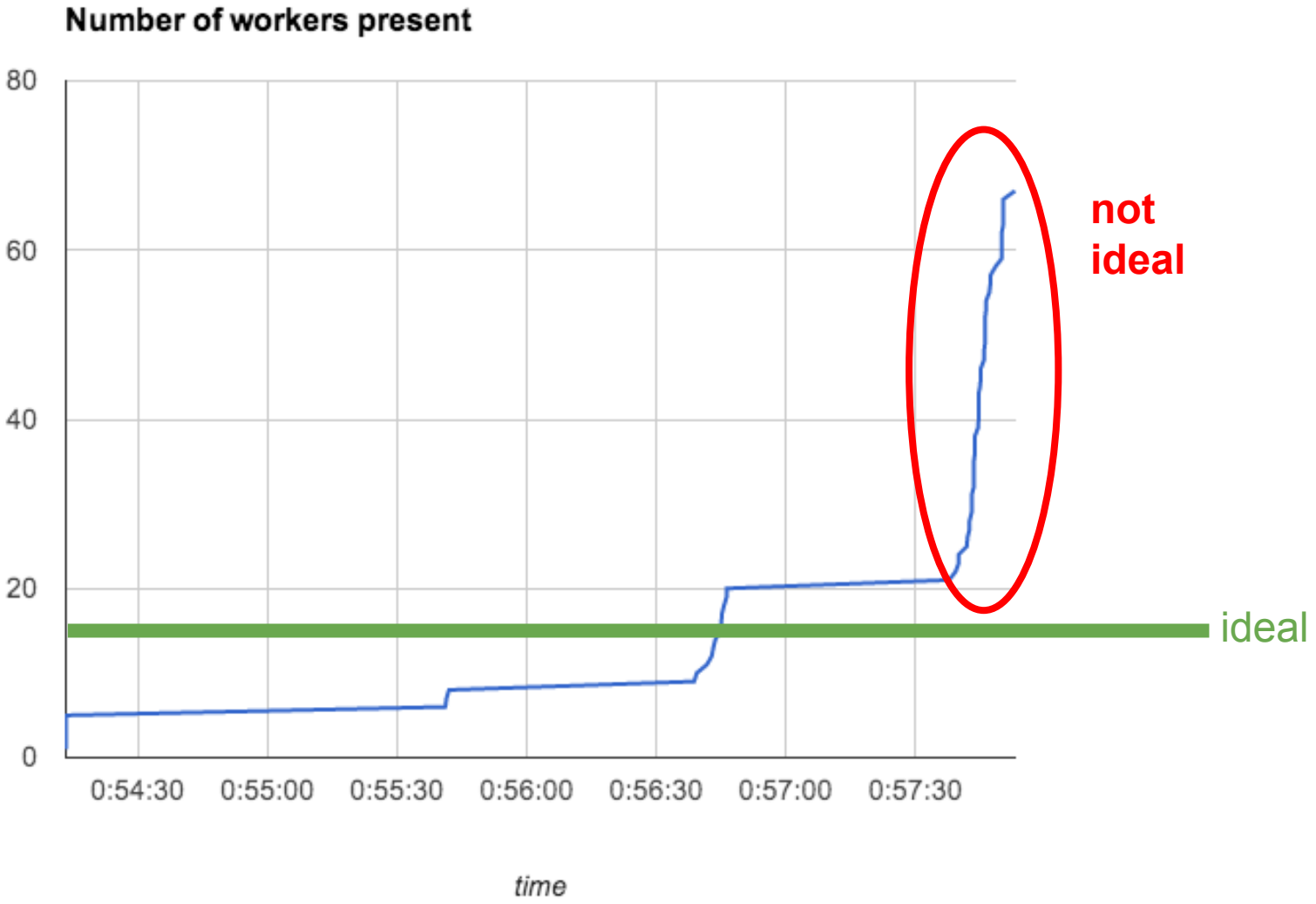
2. Number-of-workers
   - few tens-of-seconds response times

3. Add a little signalling delay ...

Auto-scaling to meet a job deadline

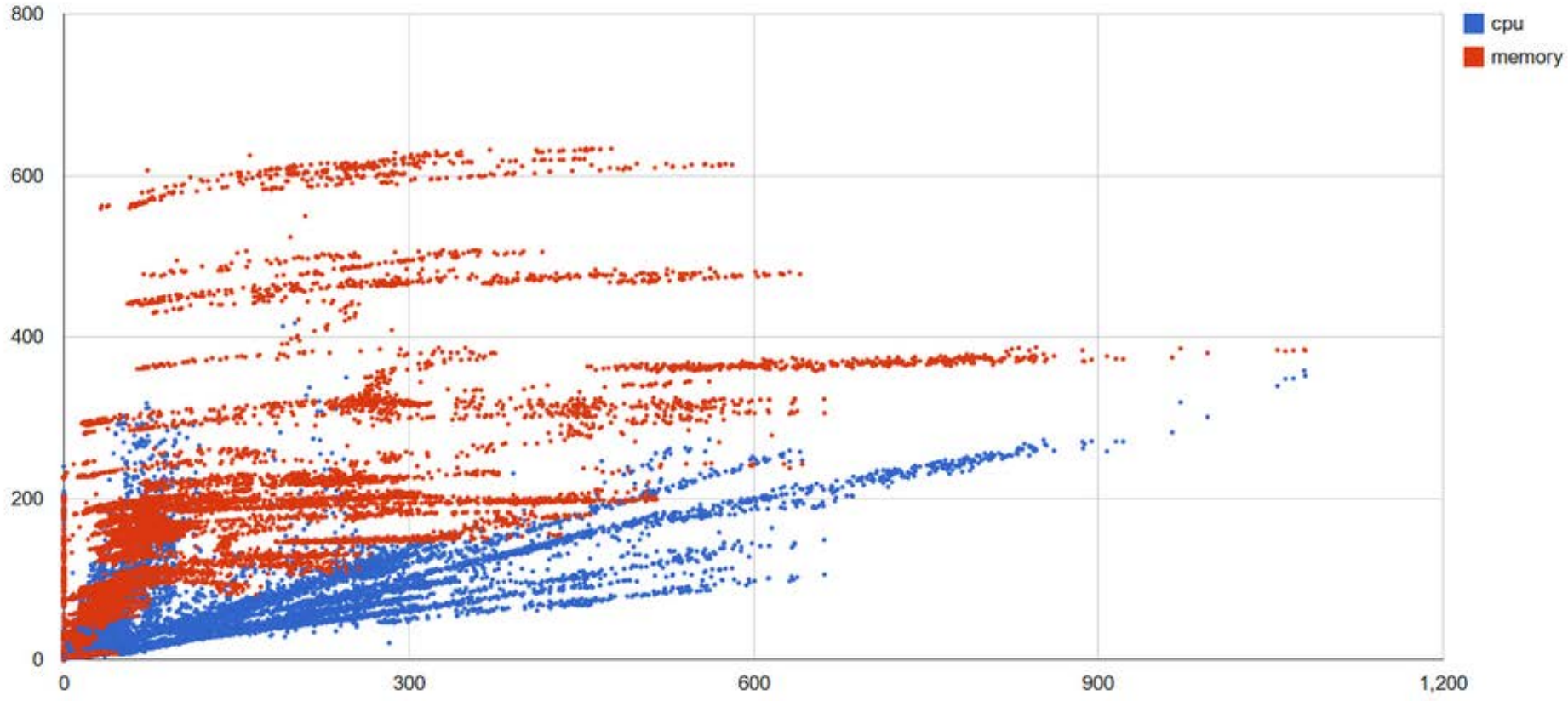


Auto-scaling to meet a job deadline



Model building is hard

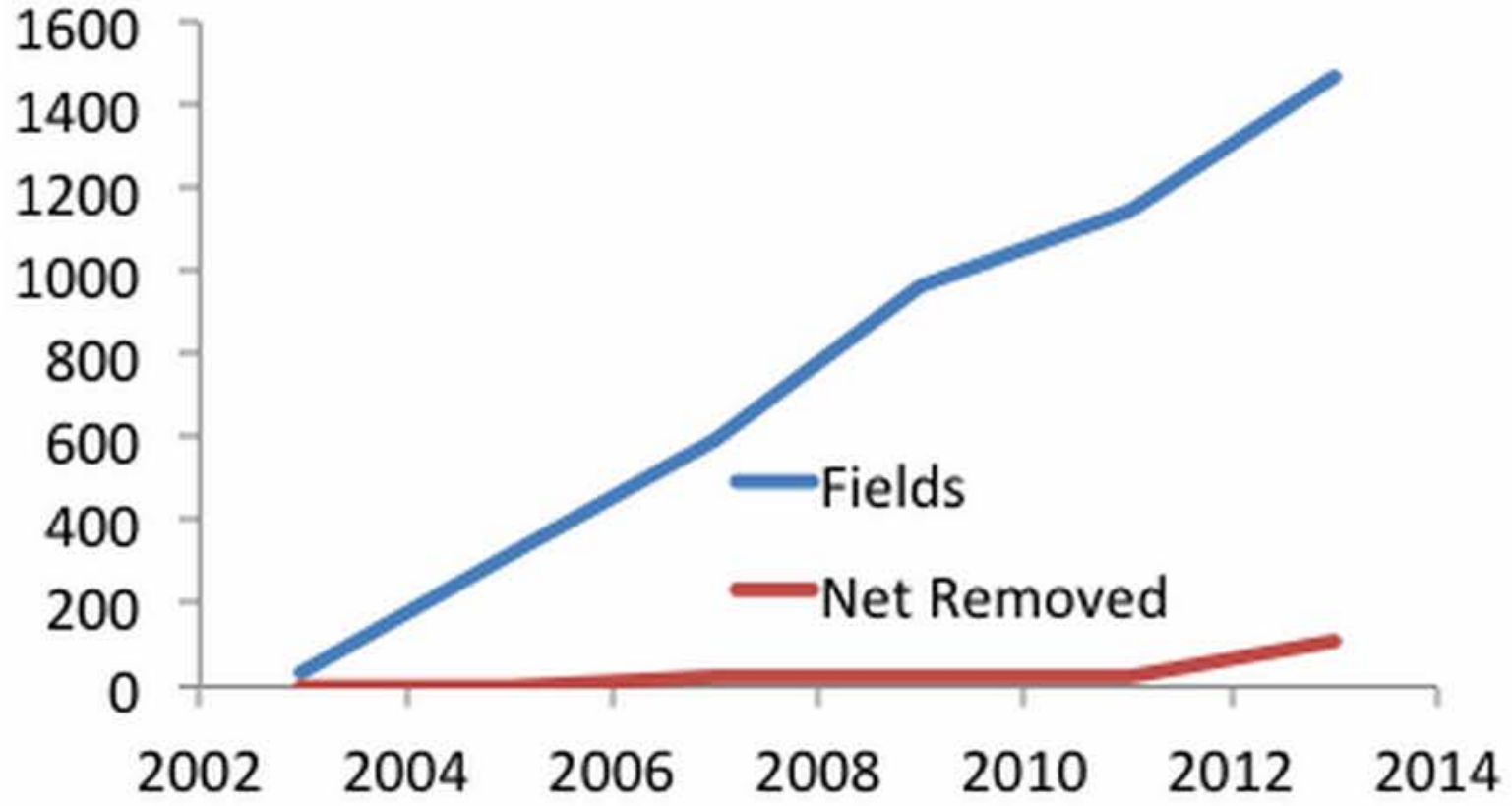
CPU, RAM usage
(arbitrary units)



Load

No worries!

Just add a few more knobs ...



GMail circa 2008

your browser

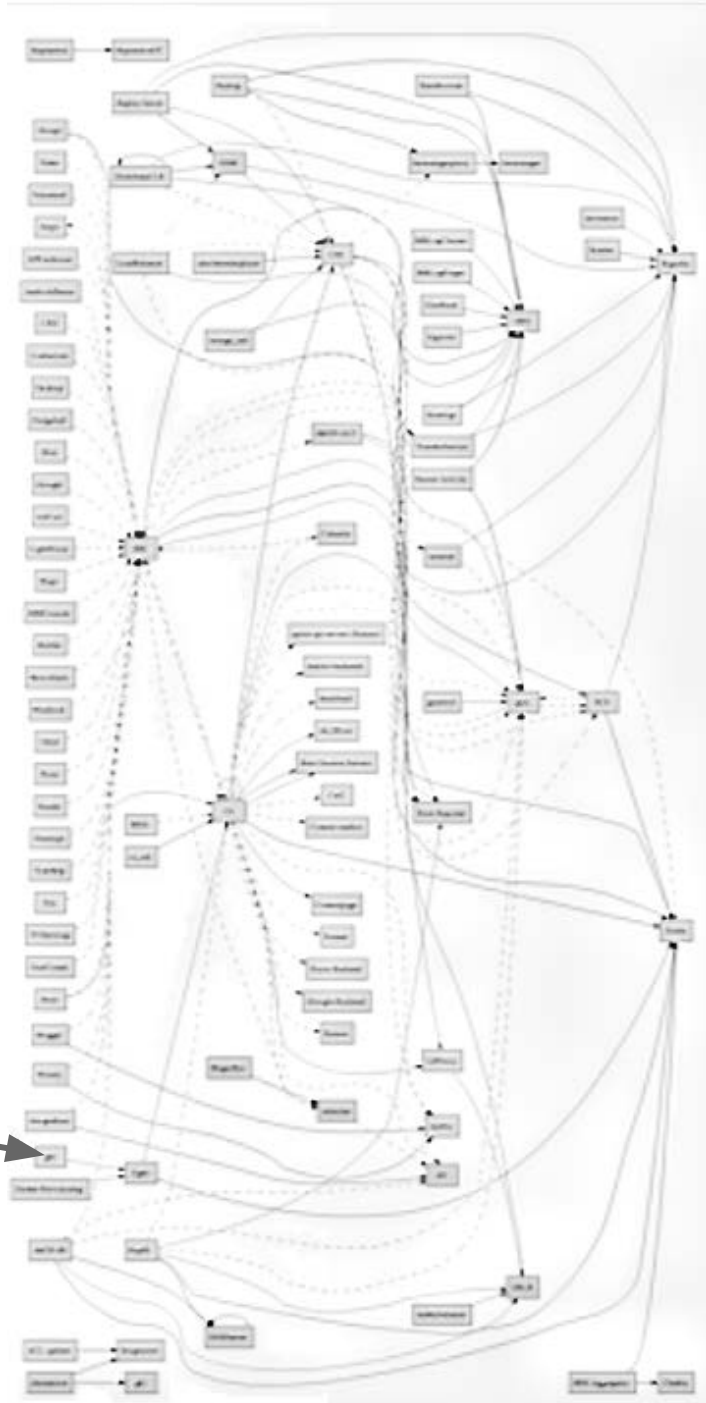


Image source: Hareesh Nagarajan

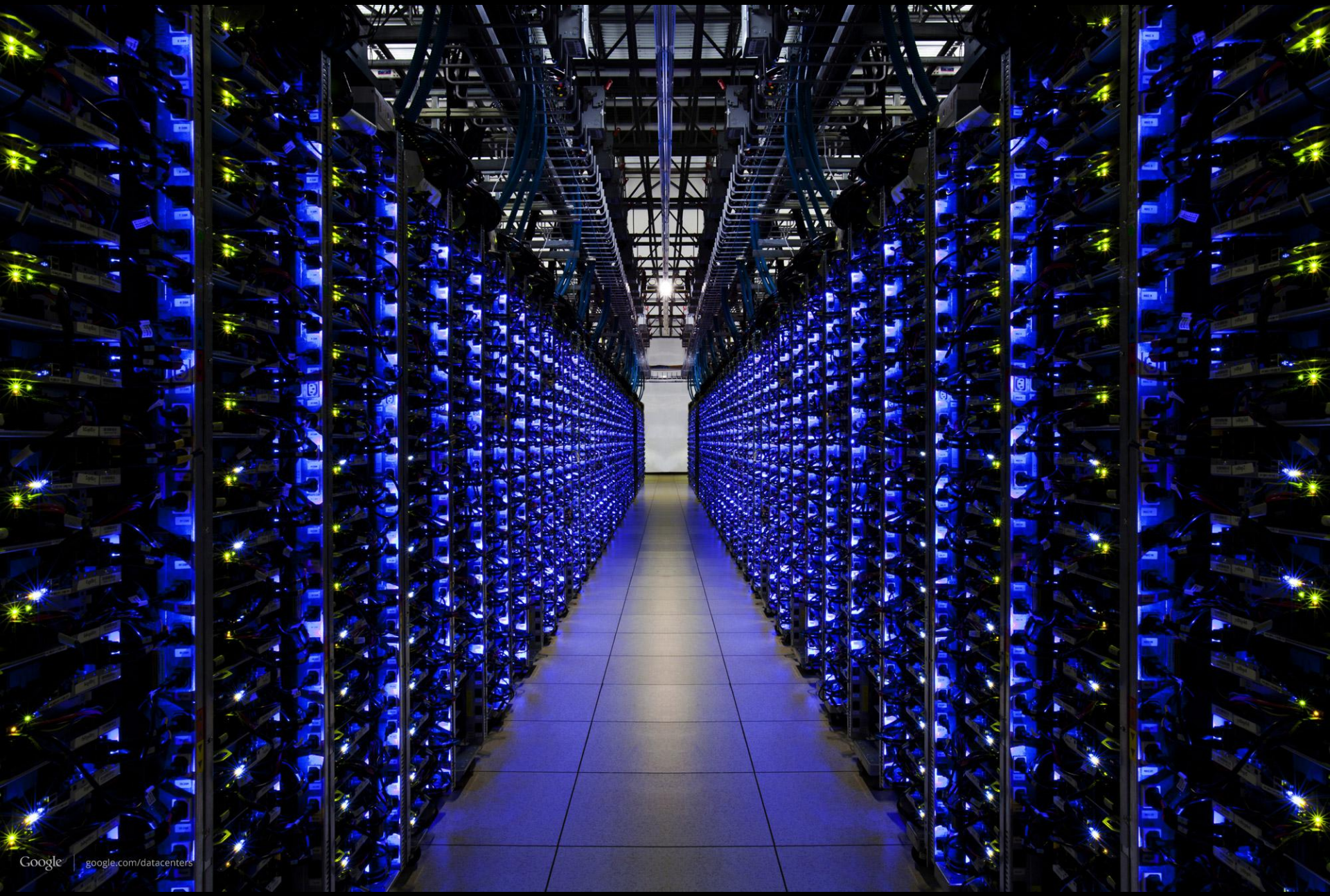
Upload malformed configuration

What does your system do?

Tip: don’t just stop working

```
umask 027
mkdir -p -m 0755 $release/usr/bin
```

“**The scariest outage ever**”
15-20% of Google's
production fleet was affected


Google | google.com/datacenters

It's 3am and your pager goes off

-- are we in trouble?

-- are we about to get into trouble?

→ what should you do about it?

Delegation is hard

be careful what you ask for

Summary

Control systems do not run in isolation

1. Do no harm
2. Make things better
3. **Assume the world <u>is</u> out to get you**

   "any sufficiently advanced incompetence is indistinguishable from malice"

   -- Grey's Law