Capacity vs Efficiency

Building a Globally Scalable Cloud Database

SRECon 2022, Sydney

Daniel Marshall <djmarshall@google.com>

Me

Electrical engineer turned SRE at Google Sydney

I work on Google Cloud Firestore

If you want to chat more about this topic, come find me after

Capacity **vs** Efficiency

Capacity – provisioning sufficient resources to handle incoming request load

Efficiency – minimising the total cost of the system

Case Study: Google Cloud Firestore

Google Cloud Firestore is a **fully scalable** NoSQL database where you **only pay for what you use**.

Corollaries

- Fully scalable ⇒ we must be able to serve any amount of traffic, at any time
- Pay for use ⇒ we pay for everything else

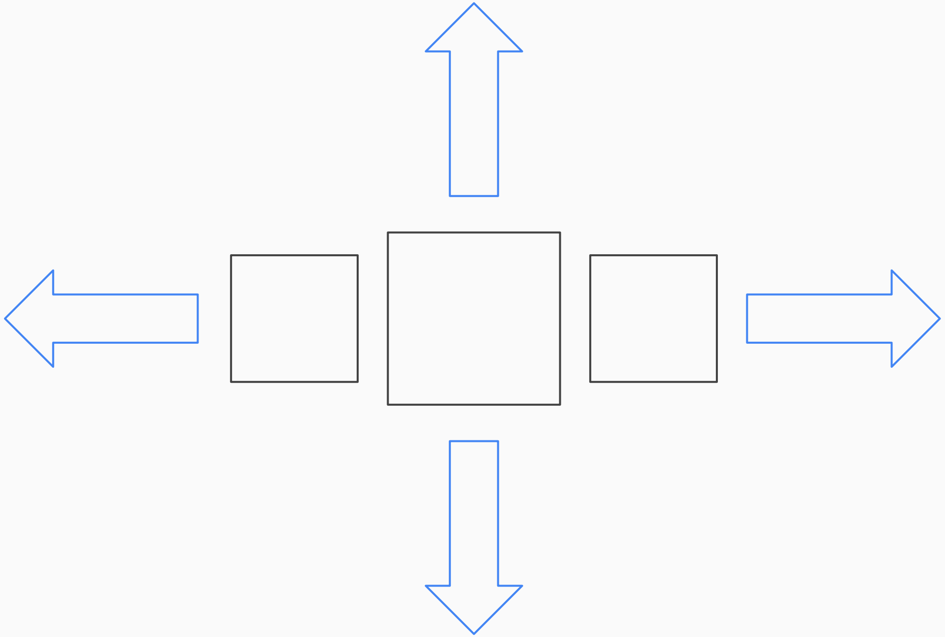
Capacity

Autoscale Everything

Autoscaling is the first step

Improves reliability **and** toil

Autoscale horizontally and vertically



Autoscaling Signals

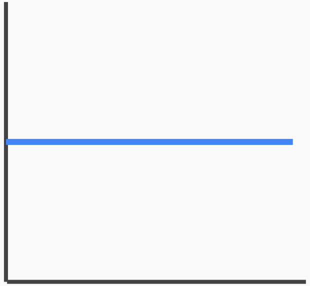
Choose one metric to scale on

Align your bottlenecks with your scaling signal

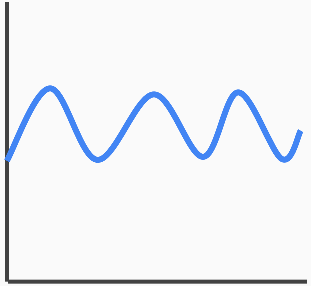
(Marginally) overprovision everything else

Traffic Patterns

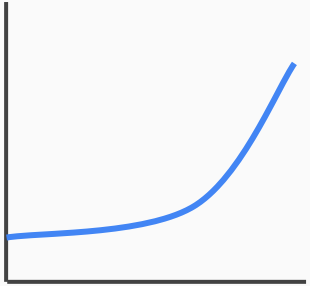
Flat



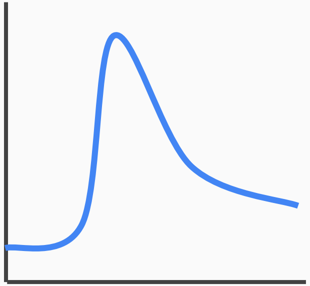
Diurnal



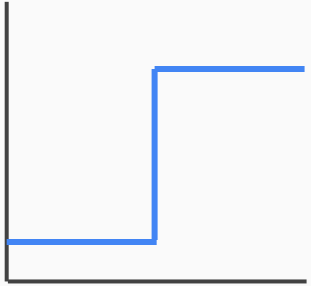
Slow ramp-up



Traffic spike

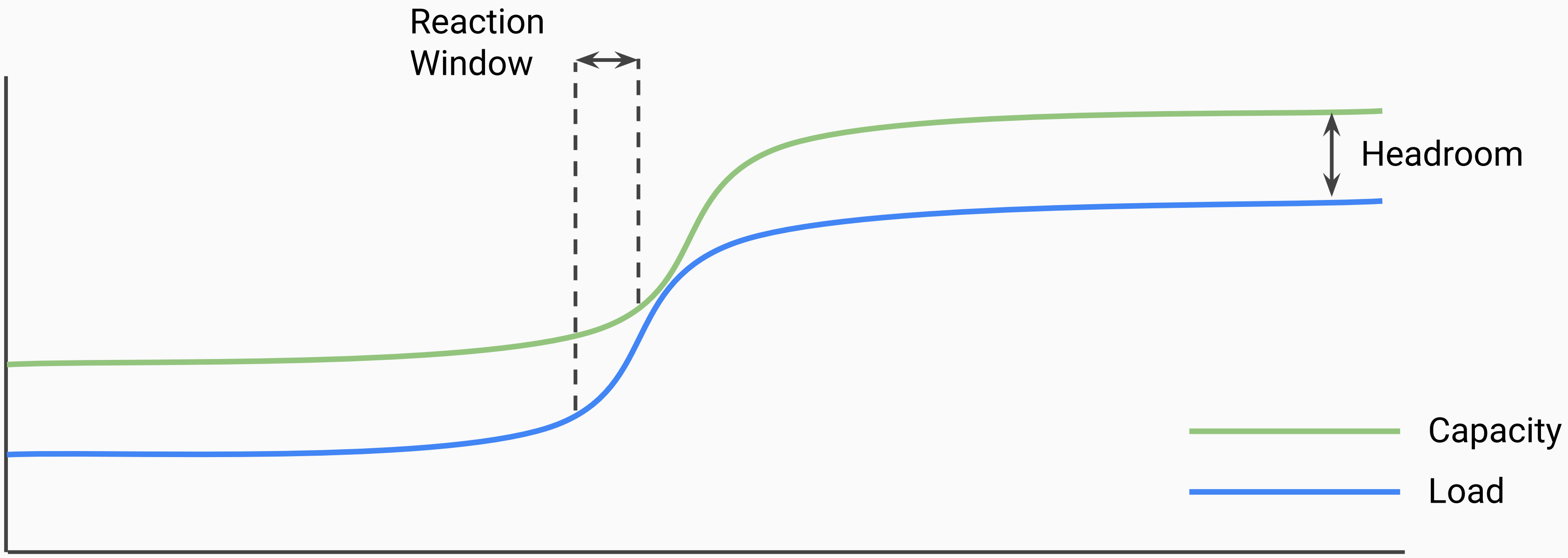


Bulk workload



Where possible, **control the ramp-up** speed

Choosing Your Headroom



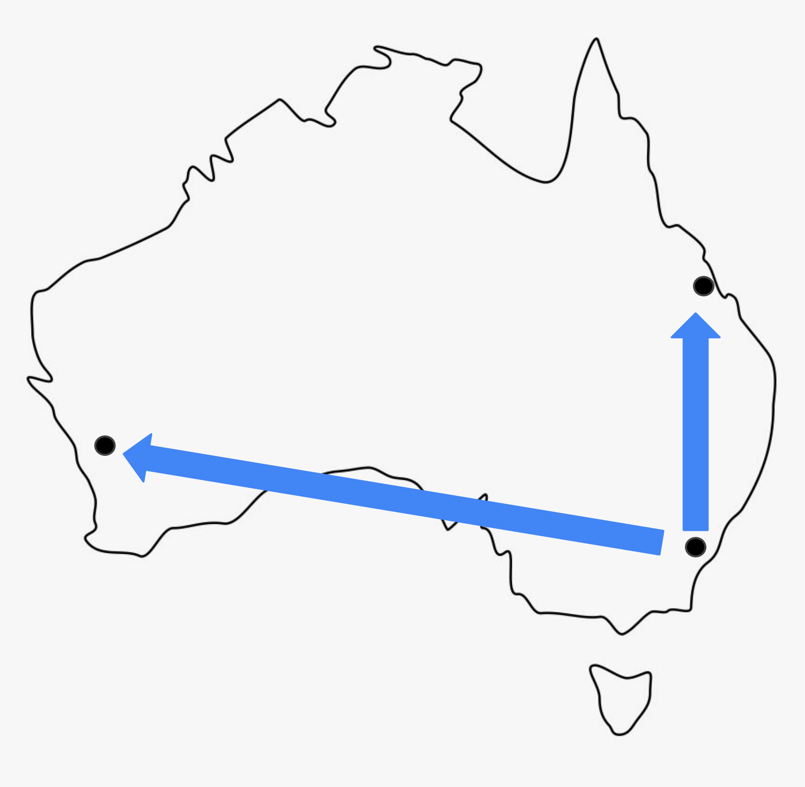
Efficiency = 100% – max(traffic_spike)

Stockout Resilience

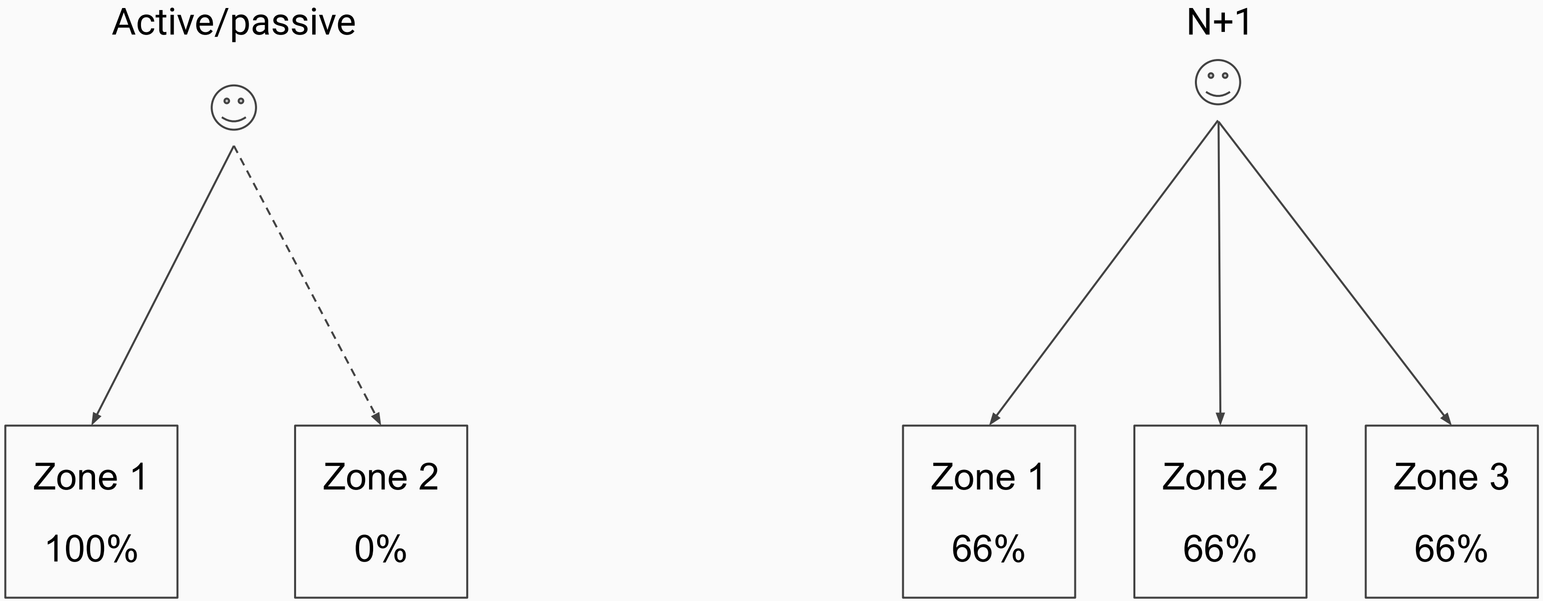
Stockout: underlying platform runs out of capacity

Use as many availability zones and regions as you can

Build capacity agility



Provisioning For Failover



Active/passive

Zone 1

100%

Zone 2

0%

N+1

Zone 1

66%

Zone 2

66%

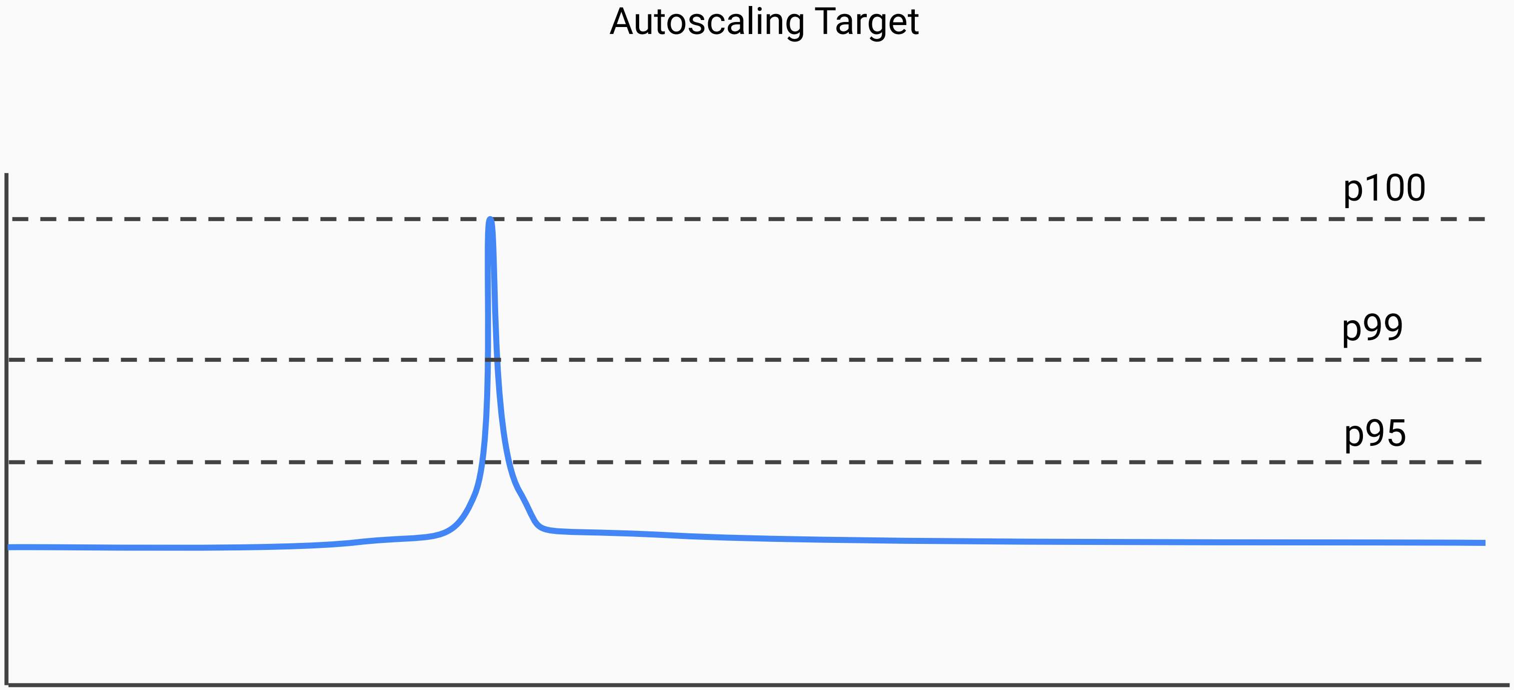
Zone 3

66%

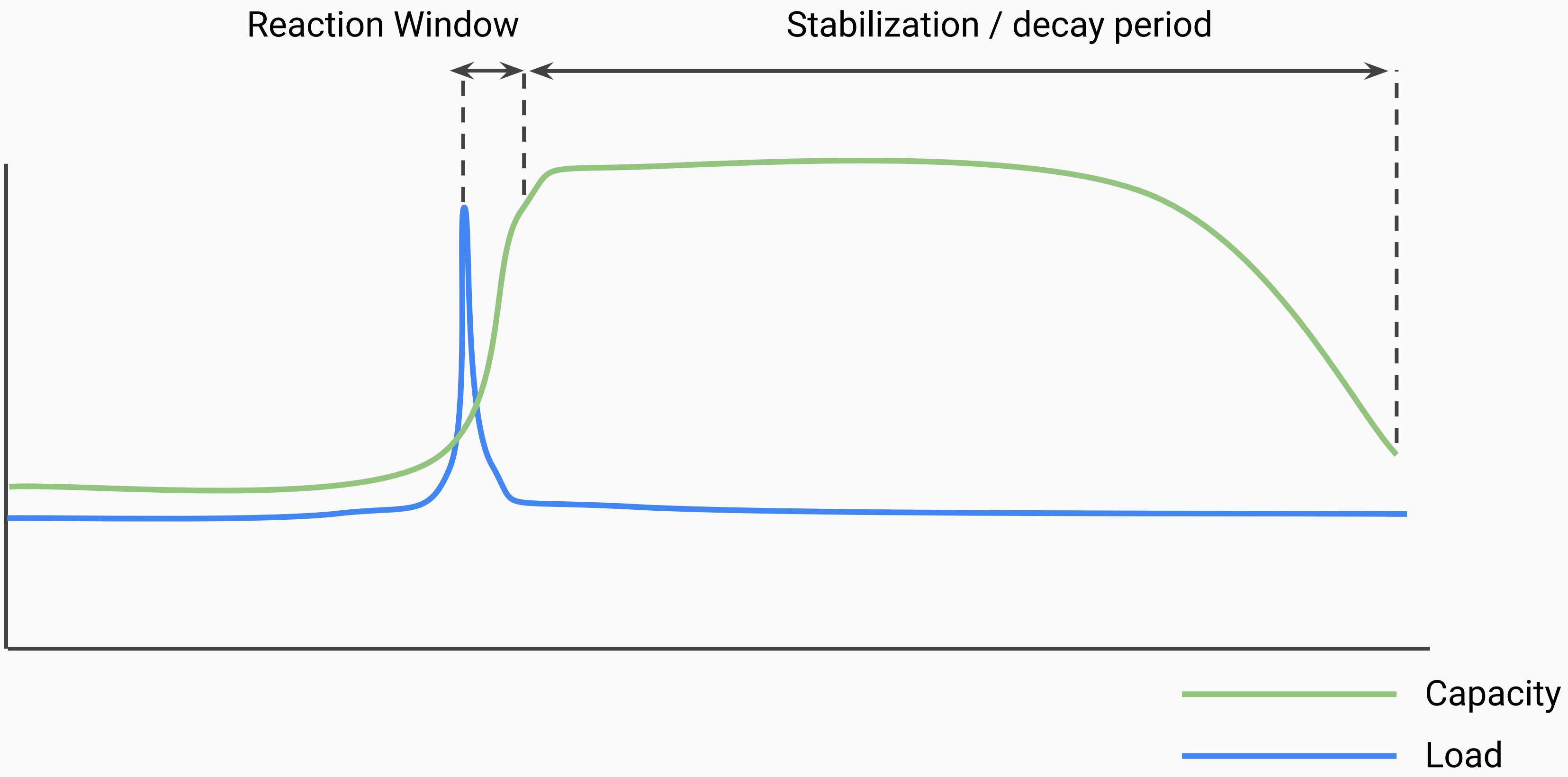
All failover models are wrong. Test your failovers!

Efficiency

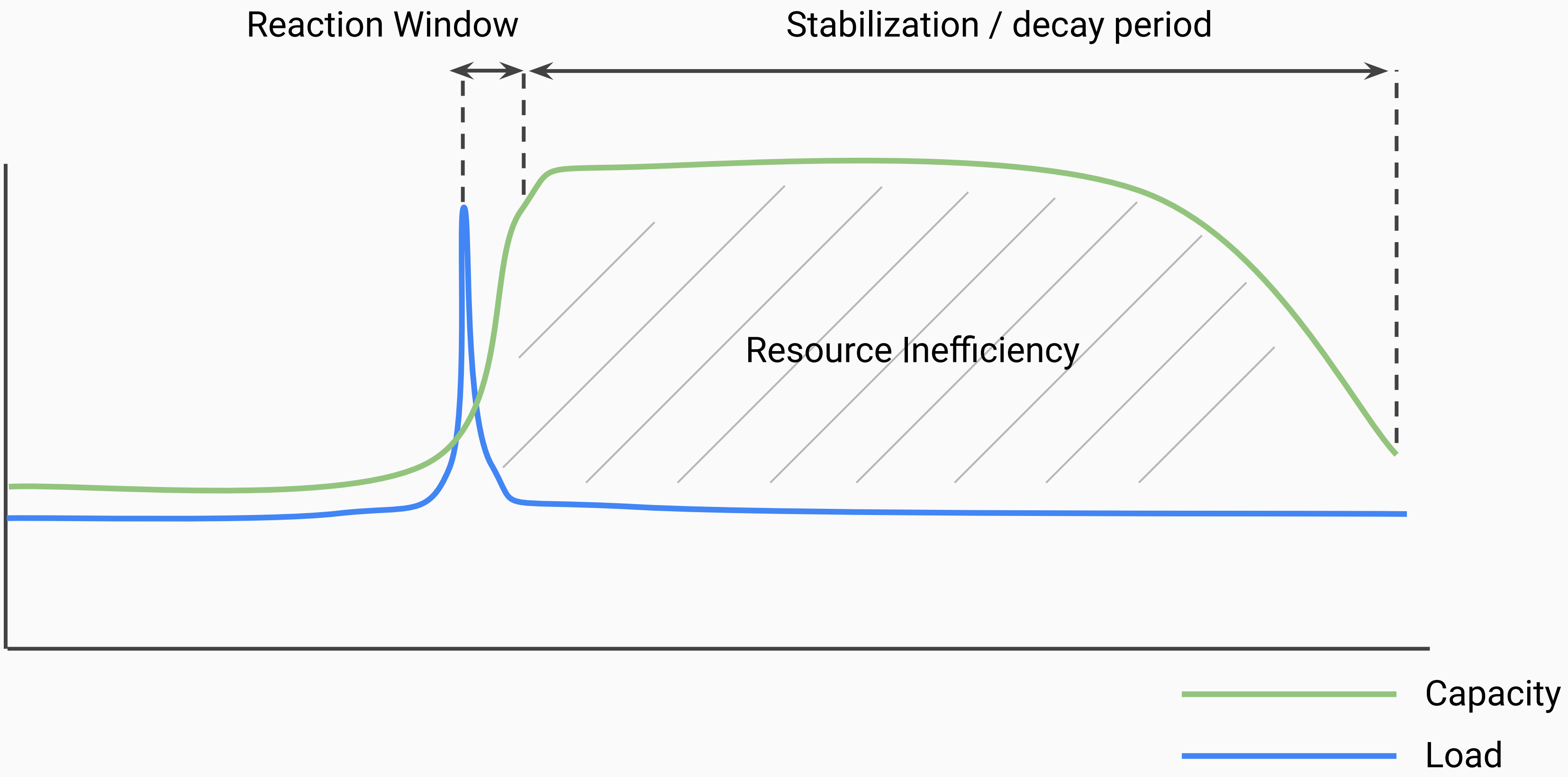
Tuning The Autoscaling



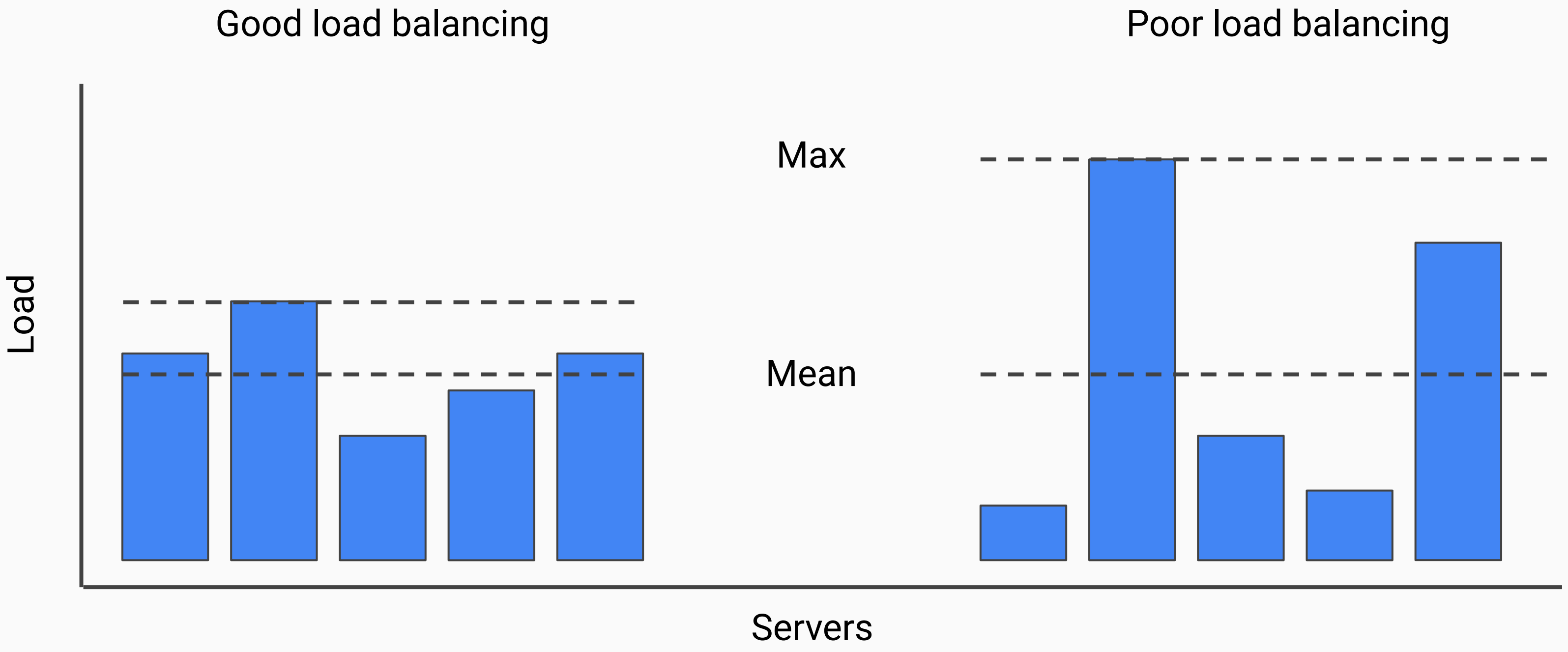
Tuning The Autoscaling



Tuning The Autoscaling



Load Balancing



Efficiency = 100% – max(traffic_spike) **– load_imbalance$_{\textbf{max-mean}}$**

Probability Distributions

$$\text{Efficiency} = 100\% - \max(\text{traffic\_spike}) + \text{load\_imbalance}_{\text{max-mean}}$$

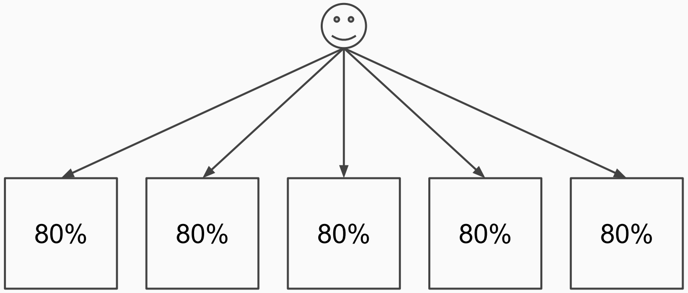
$$\Downarrow$$

$$\text{Efficiency} = 100\% - \mathbf{P}_{\mathbf{max}}(\text{traffic\_spike} + \text{load\_imbalance}_{\text{max-mean}})$$

Efficient Failovers

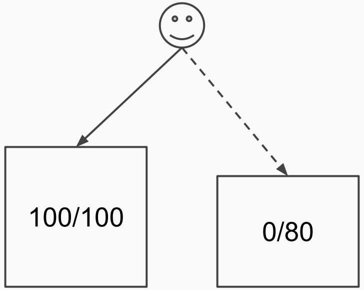
N+1: Increase the N!

- N=1 50% efficiency loss
- N=4 20% efficiency loss



Active/Passive

- Use lower SLO / spot instances
- Underprovision and scale

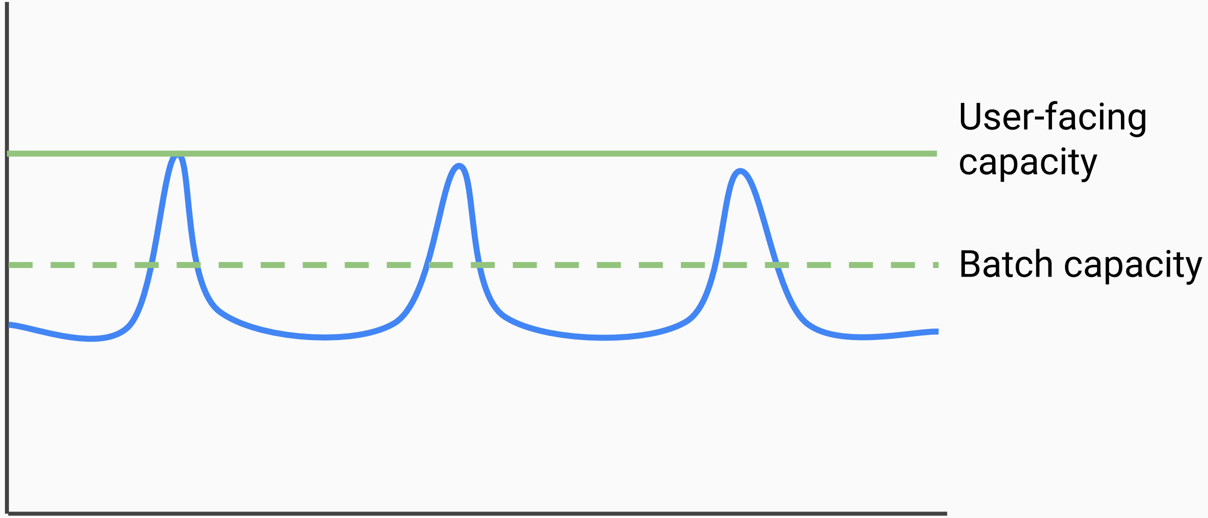


Efficiency = 100% – P_{max}(traffic_spike + load_imbalance$_{max\text{-}mean}$ + **failover_capacity**)

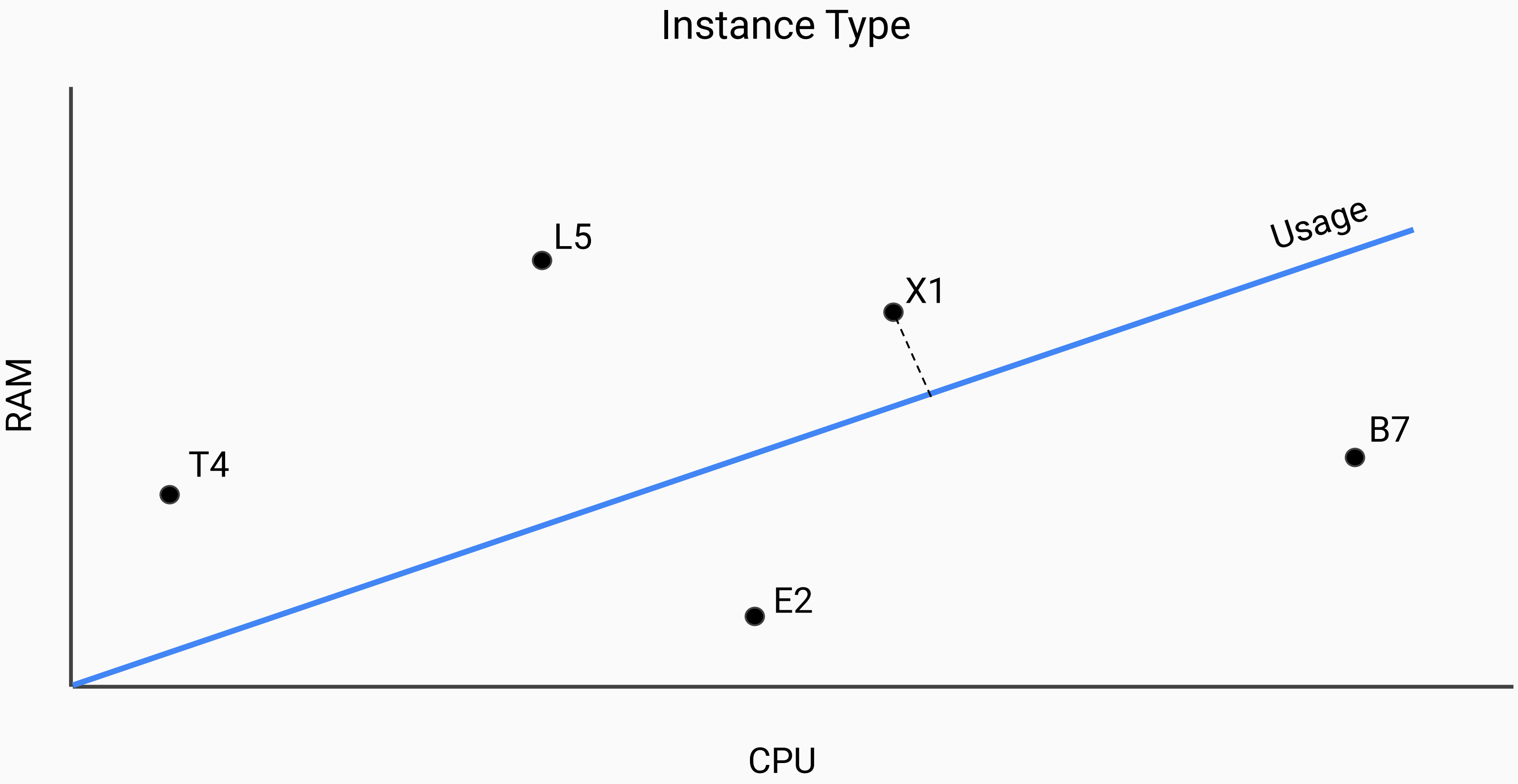
Batch Traffic

Key observation: Batch traffic is latency tolerant

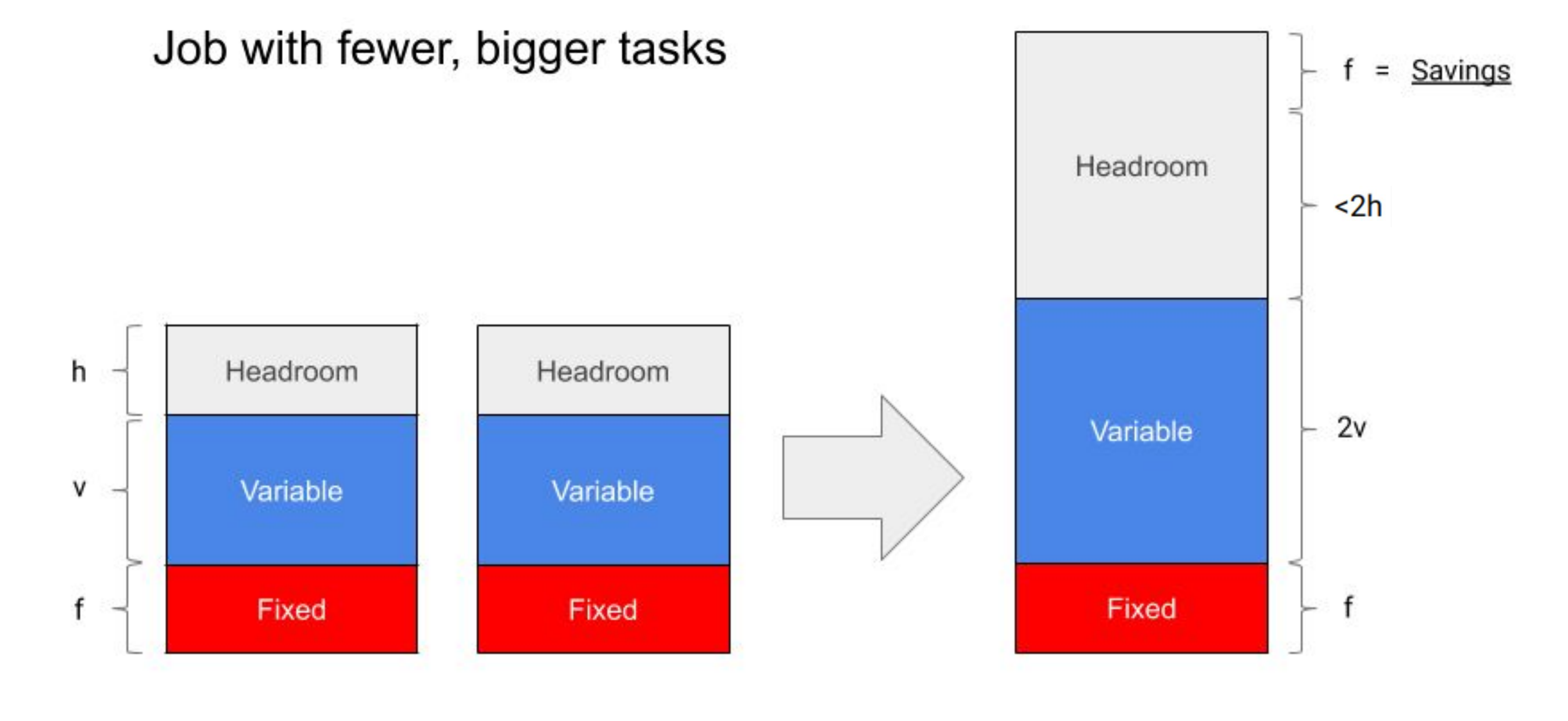
- Split it out
- Use lower SLO / spot instances
- Run it hotter



Finding The Right Shape



Bigger Is Better

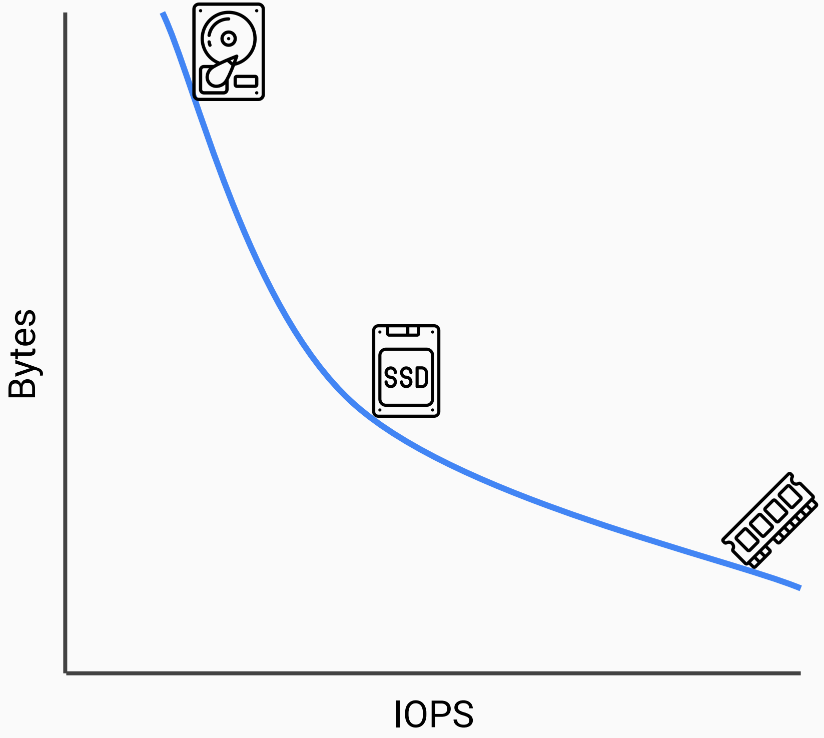


Resource Tradeoffs

Byte/$: HDD > SSD > RAM

IO/$: HDD < SSD < RAM

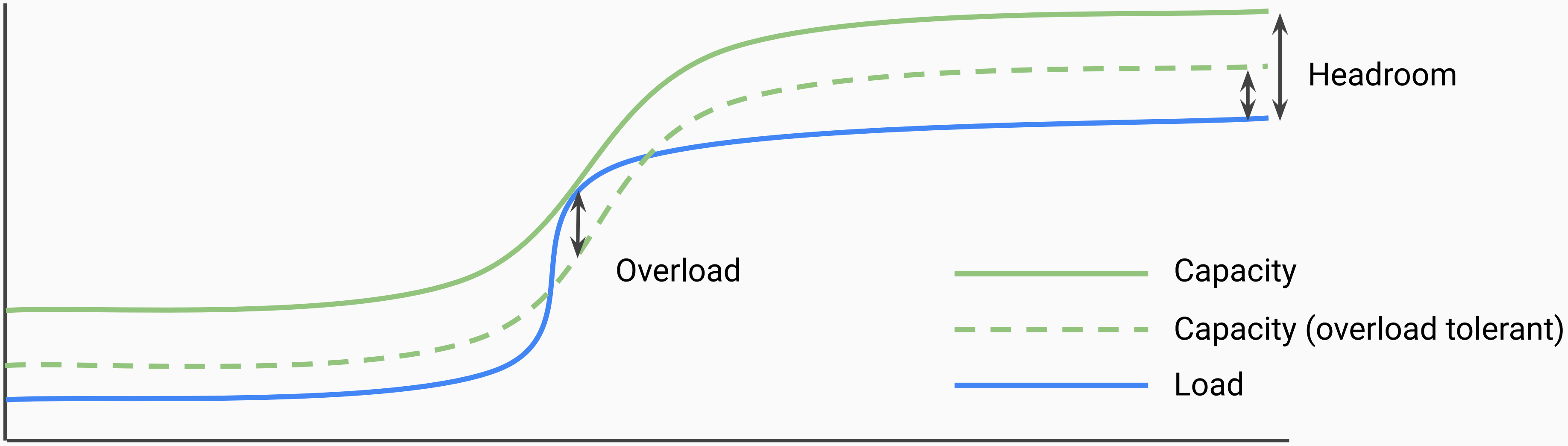
For GC languages: CPU vs RAM



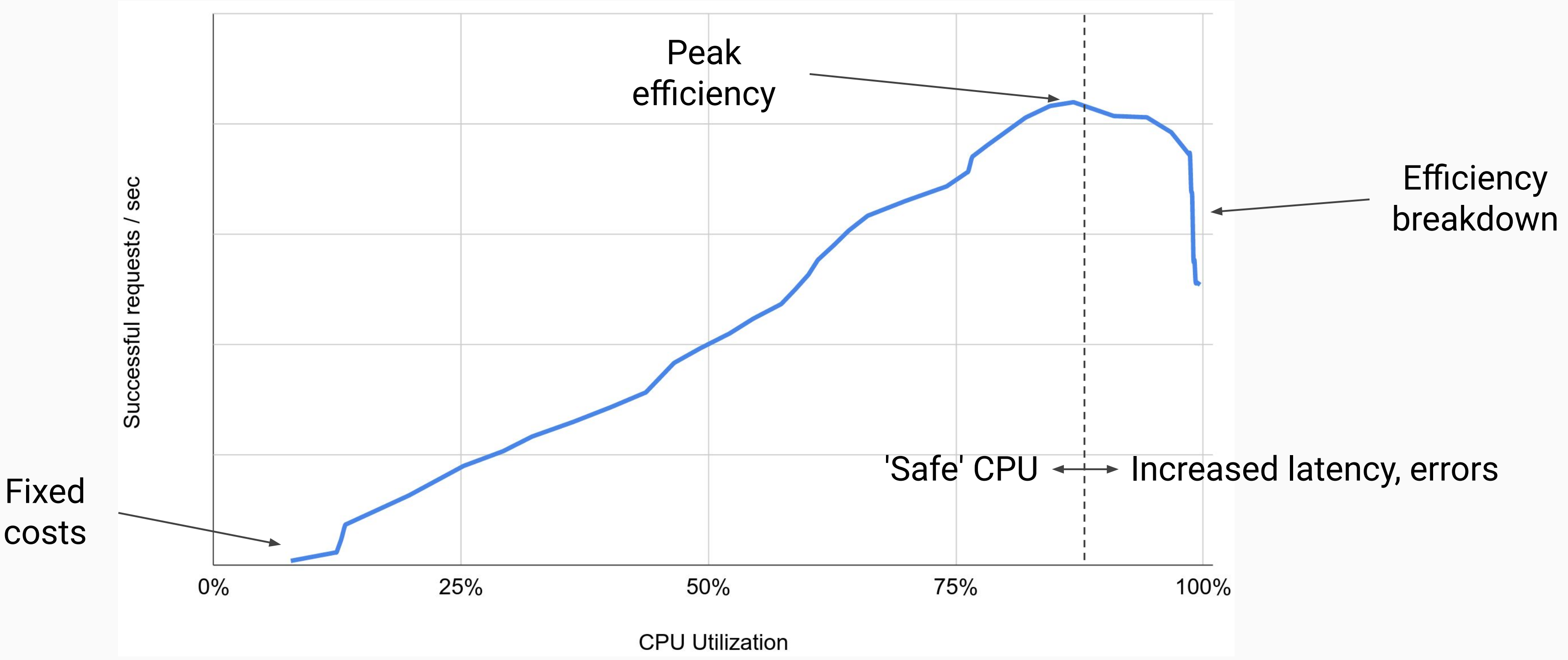
Icons by SmashIcons from www.flaticon.com

Handling Overload

Overload And Efficiency



Understand Your Limits



Peak efficiency

Efficiency breakdown

'Safe' CPU ←→ Increased latency, errors

Fixed costs

$$\text{Efficiency} = \textbf{safe\_cpu}\% - P_{\text{max}}(\text{traffic\_spike} + \text{load\_imbalance}_{\text{max-mean}} + \text{failover\_capacity})$$

Loadshedding

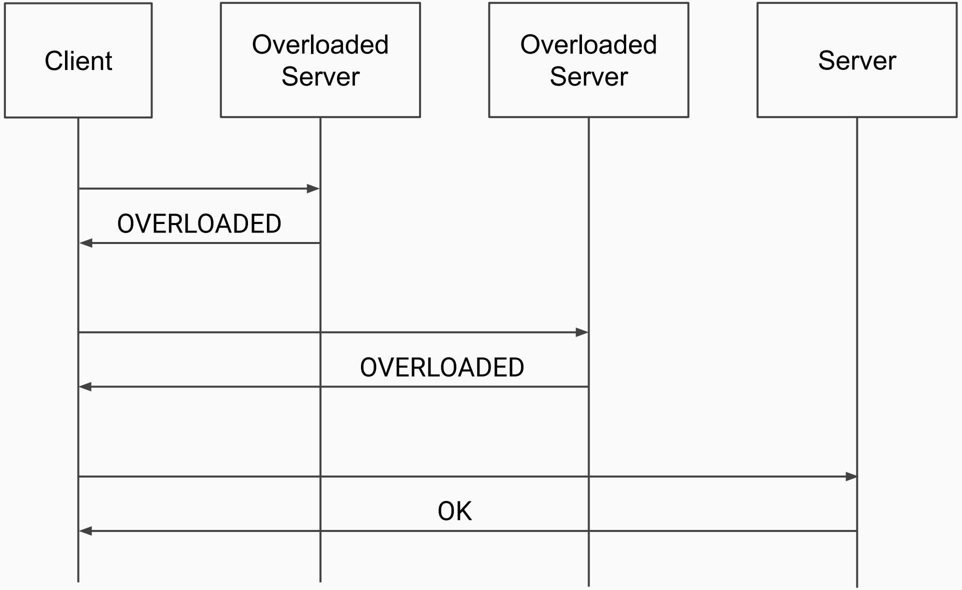
Rejecting 1 request is better than deadlining 2

Reject fast, reject early

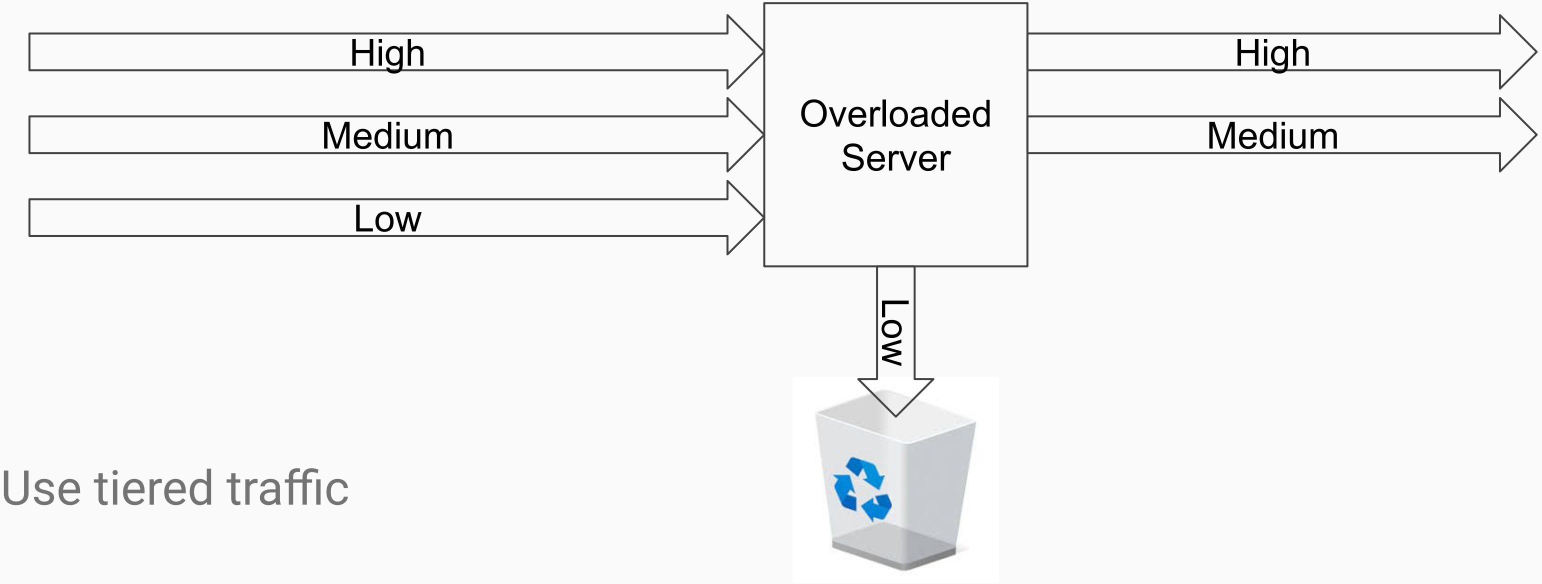
Can be done client side: throttle on latency/errors, exponential backoff

Loadbalancing via loadshedding

Loadbalancing via loadshedding



Quality Of Service



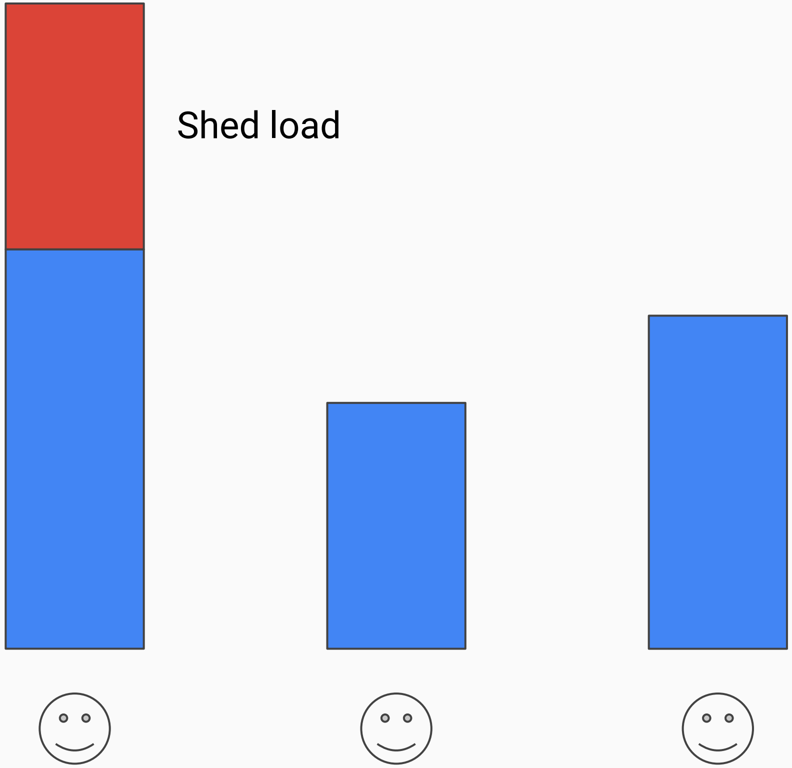
- Use tiered traffic
- Serve degraded results

Fairness Under Overload

Maintain performance for as many users as possible

Choose the right scheduler

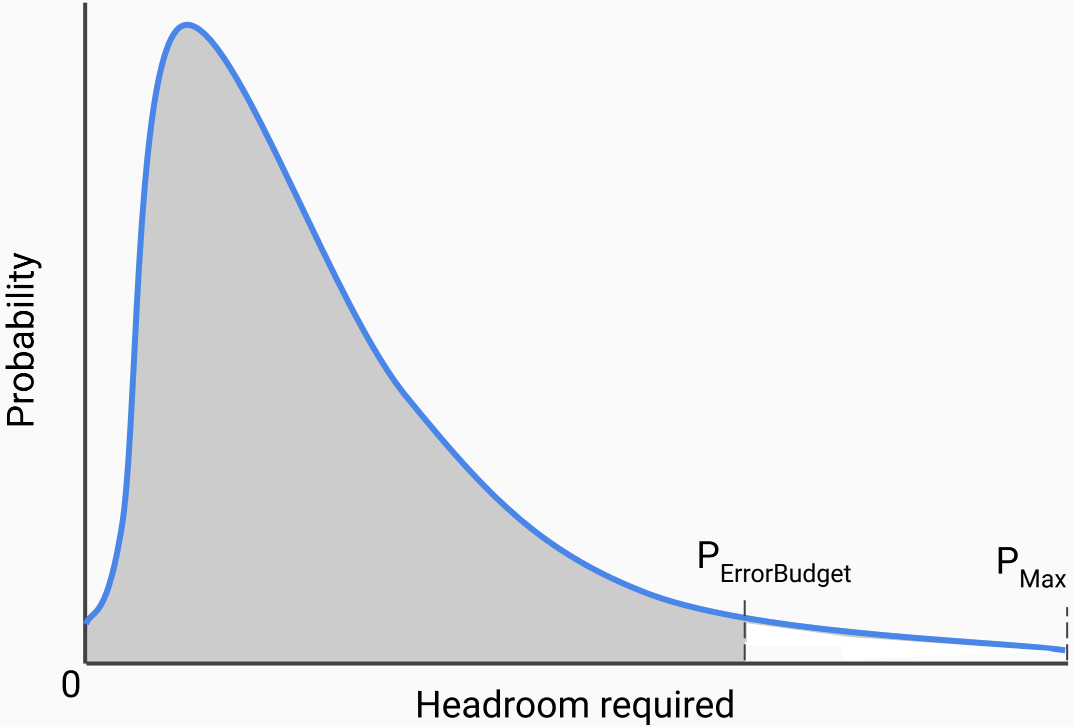
Load shed proportionally



The Efficiency Formula

$$\begin{aligned}
&\text{Efficiency} = \text{safe\_cpu\%} - P_{\text{max}}(\\
&\qquad \text{traffic\_spike}\\
&\qquad + \text{load\_imbalance}_{\text{max-mean}}\\
&\qquad + \text{failover\_capacity}\\
&)
\end{aligned}$$

$$\begin{aligned}
&\text{Efficiency} = \text{safe\_cpu\%} - \mathbf{P}_{\mathbf{ErrorBudget}}(\\
&\qquad \text{traffic\_spike}\\
&\qquad + \text{load\_imbalance}_{\text{max-mean}}\\
&\qquad + \text{failover\_capacity}\\
&)
\end{aligned}$$



Takeaways

$$Efficiency = safe\_cpu\% - P_{ErrorBudget}(traffic\_spike + load\_imbalance + failover\_capacity)$$

- Autoscale everything.
- Expand your footprint to minimise N+1 overheads.
- Test your failovers regularly.
- Understand your resource needs. Reshape your servers to fit. Trade off resources.
- Being more reliable under overload improves efficiency.

Questions?

#22apac-day2-track1