Techniques Netflix Uses to Weather Significant Demand Shifts

Joseph Lynch

Speaker

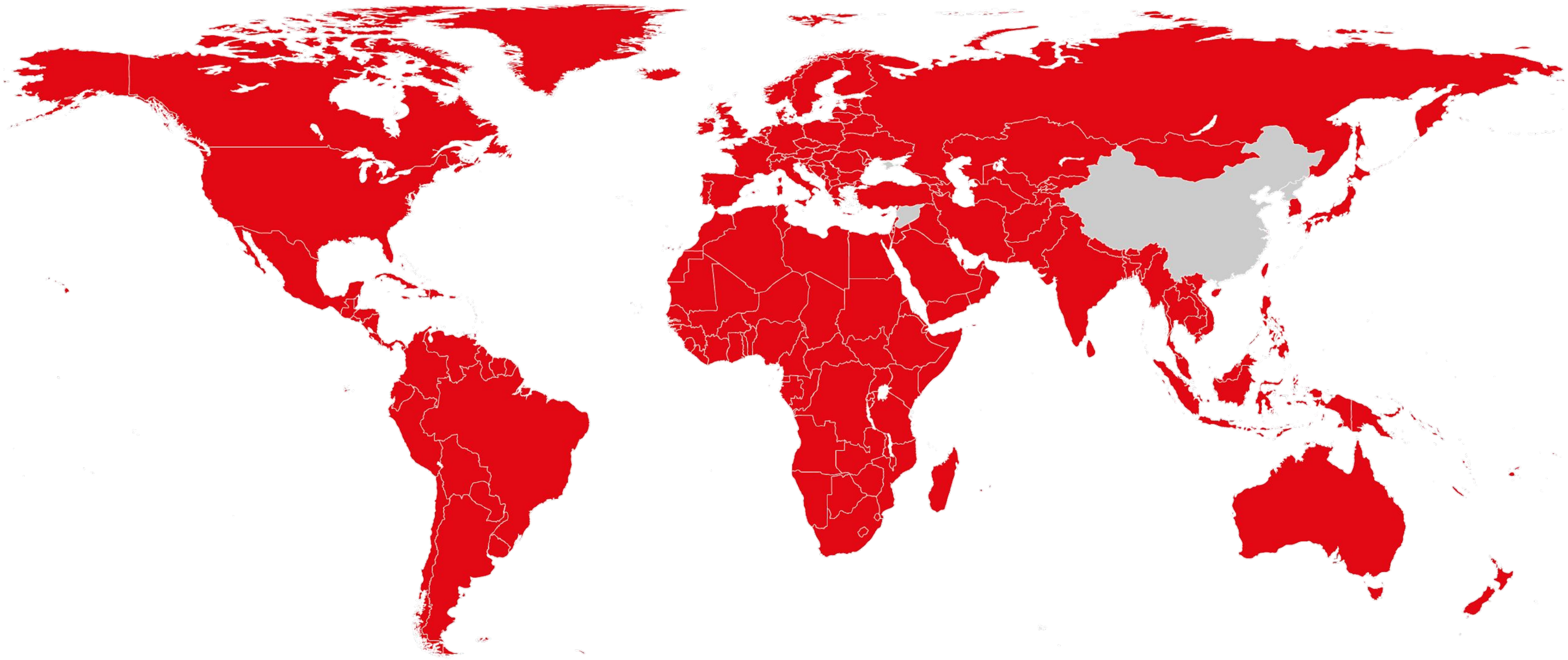
Joseph Lynch

Principal Software Engineer
Platform Engineering at Netflix

Database shepherd, compute optimizer, distributed system nerd

https://jolynch.github.io/

Problem - Global

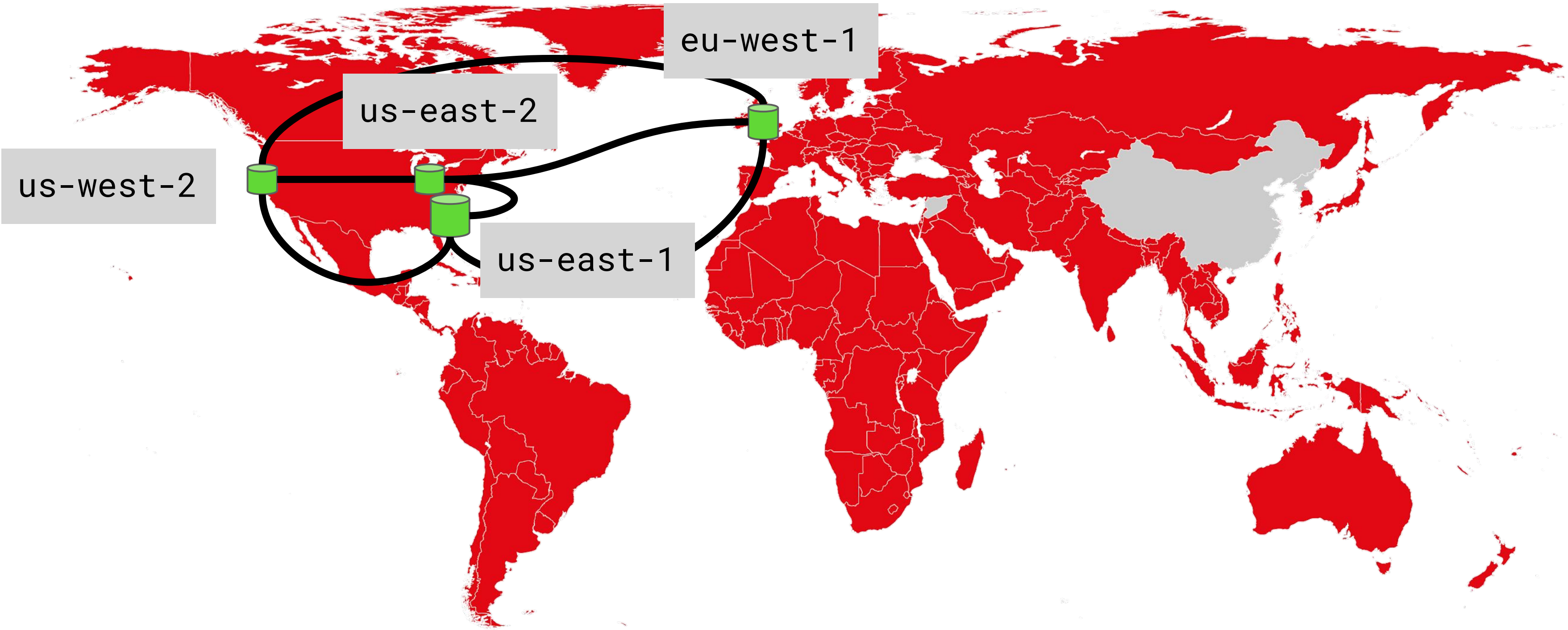


Mobile
Diverse Device Capabilities
Weaker Network
Android and IOs

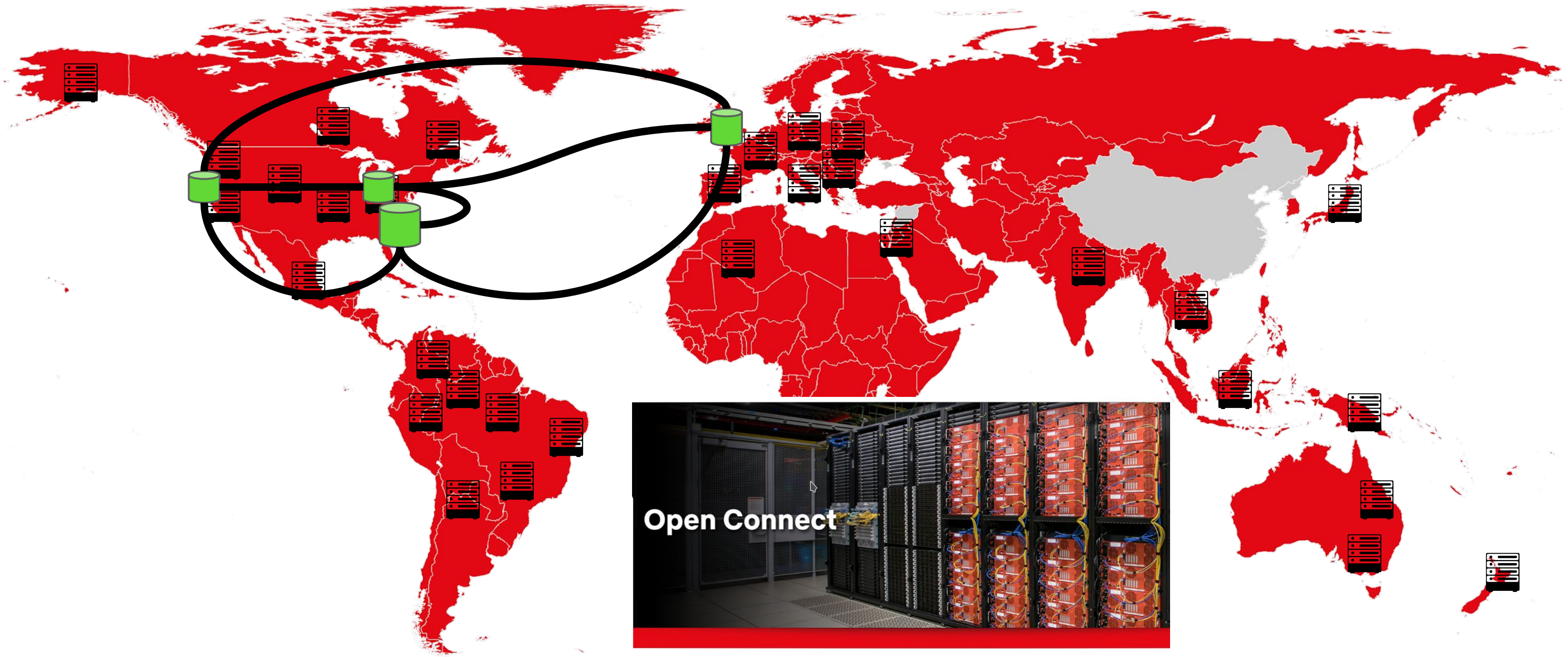
PC
Diverse Device Capabilities
More Stable Network
Medium screens

TV
Diverse Device Capabilities
More Stable Network
Large screen

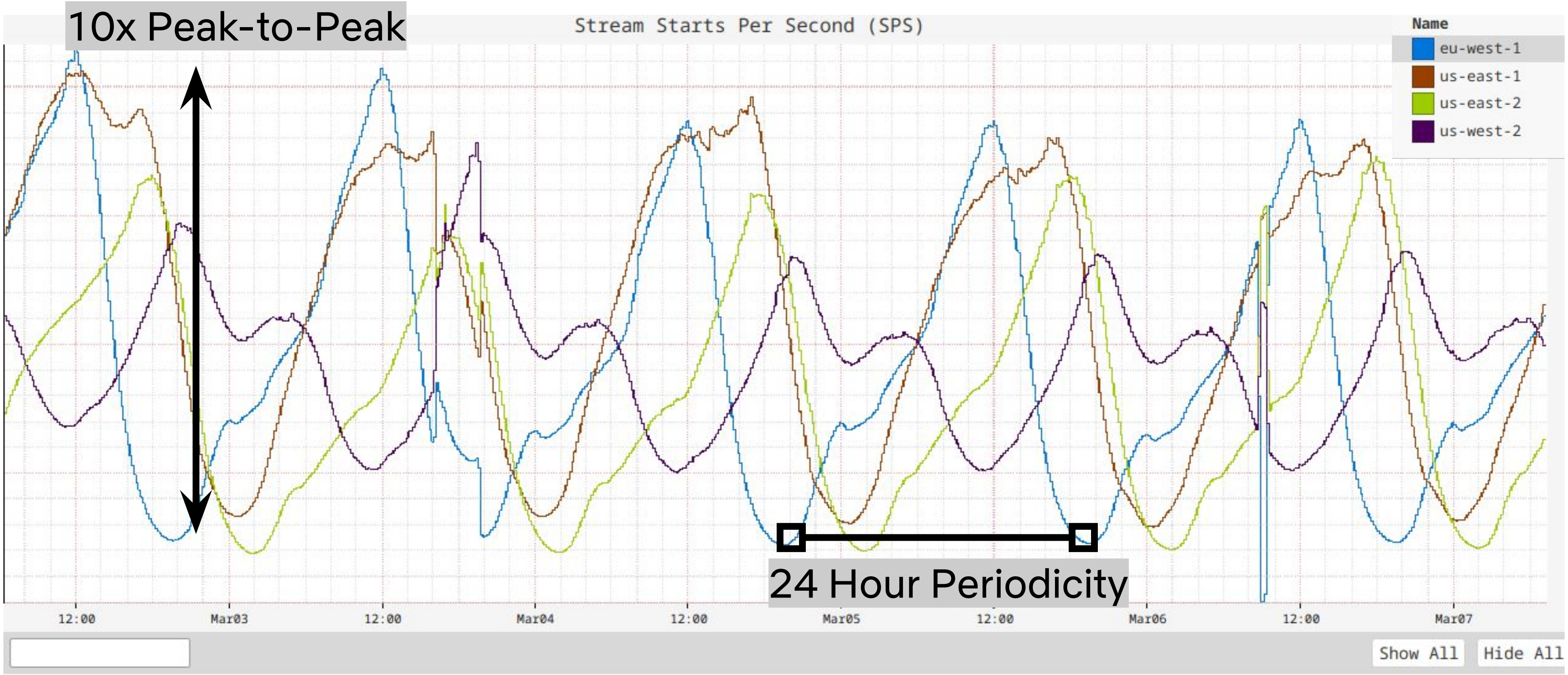
Solution - Global Control Plane



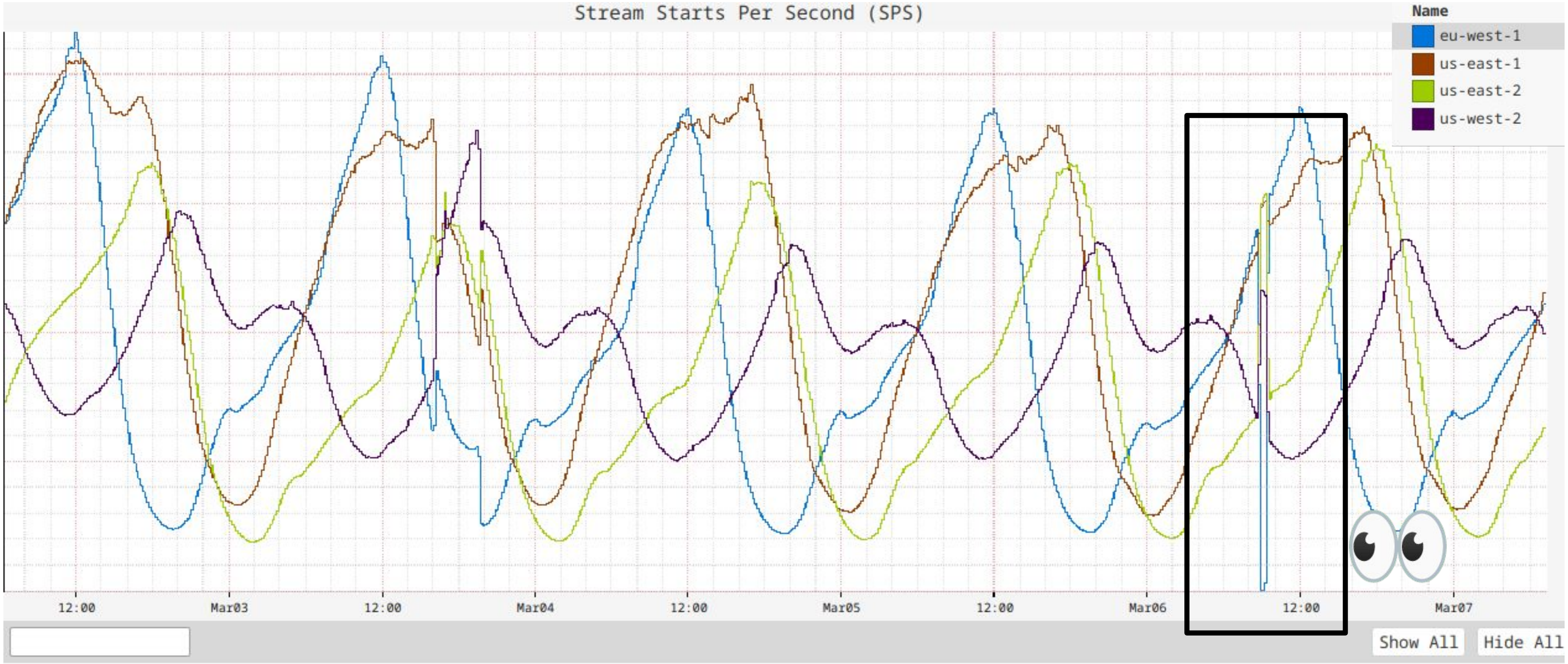
Solution - Global Data Plane



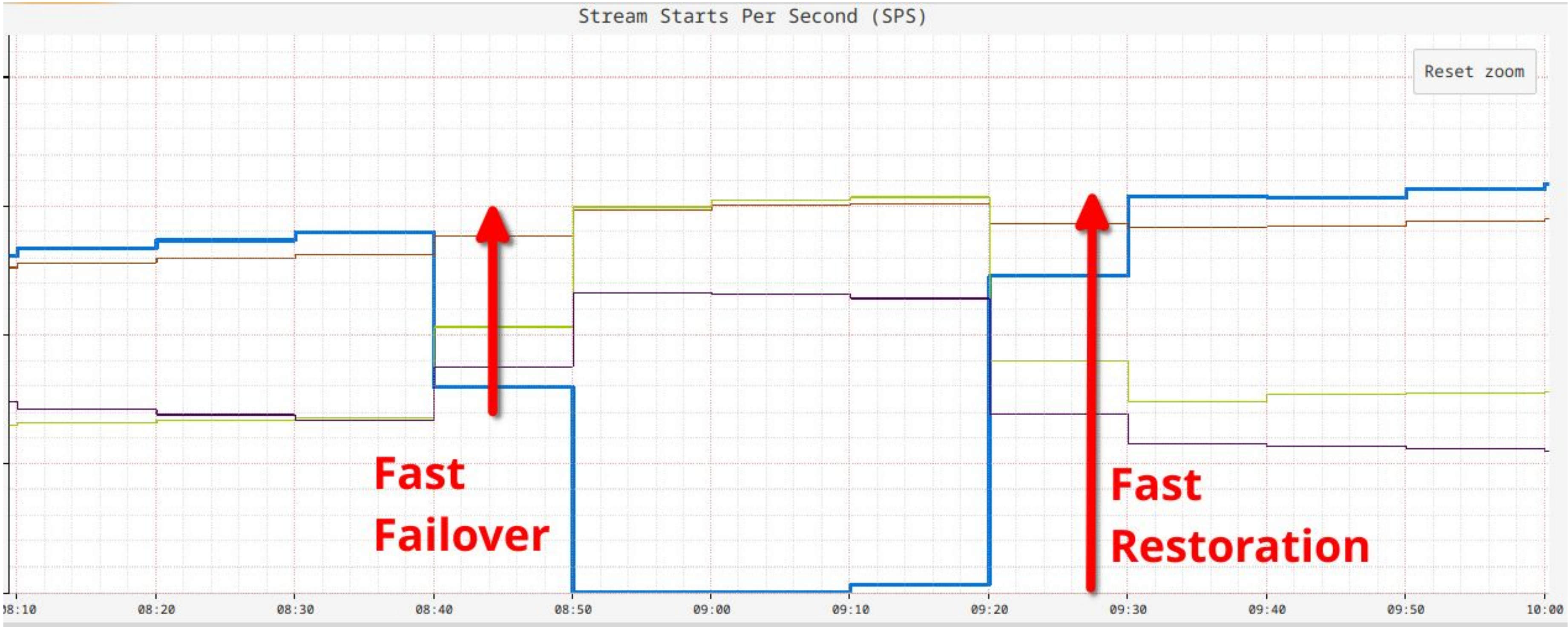
Variable Start-Per-Second (SPS) Load



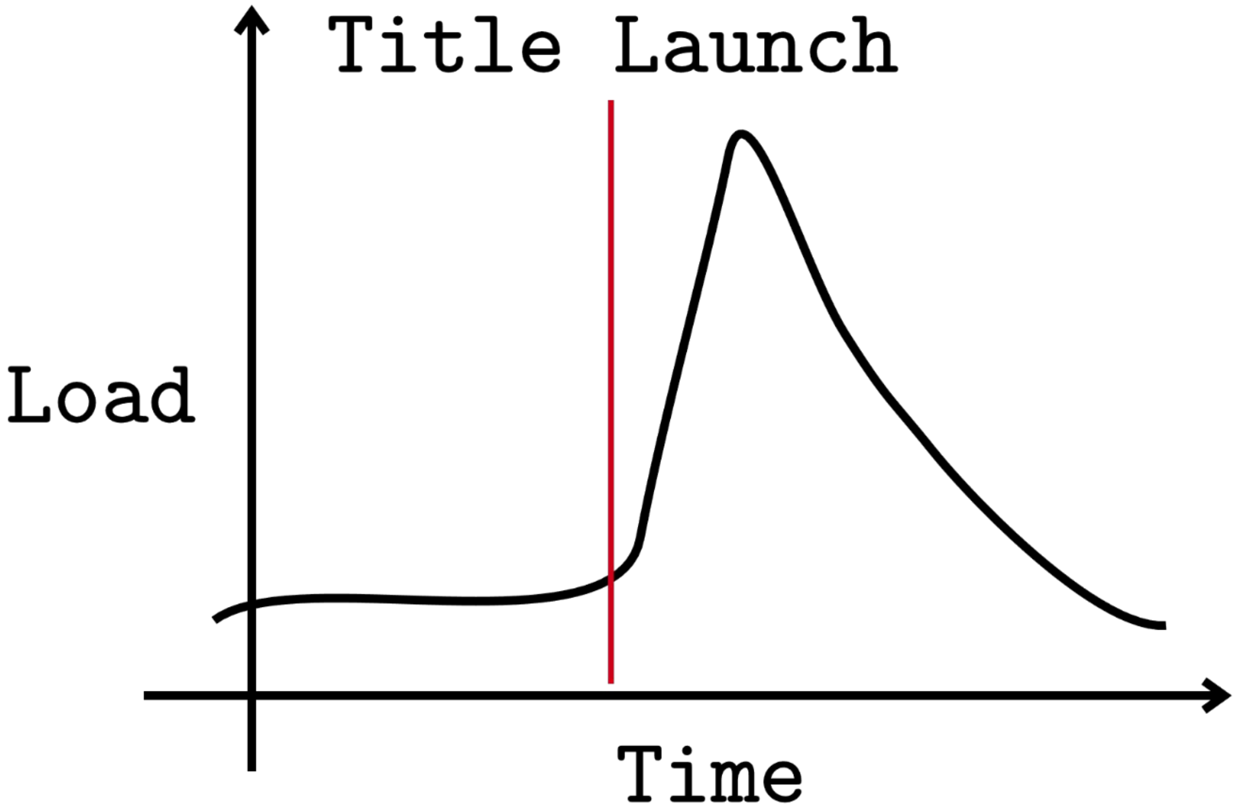
Variable Start-Per-Second (SPS) Load



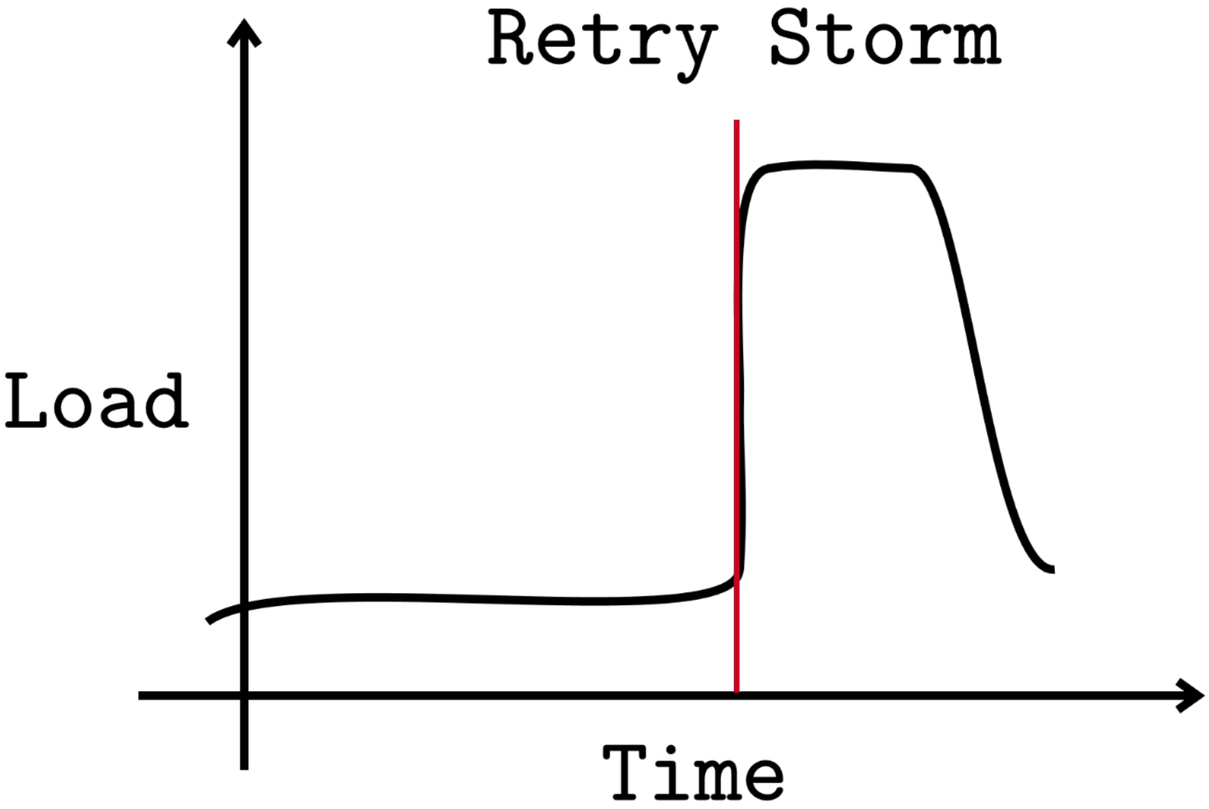
Failover Driven Demand



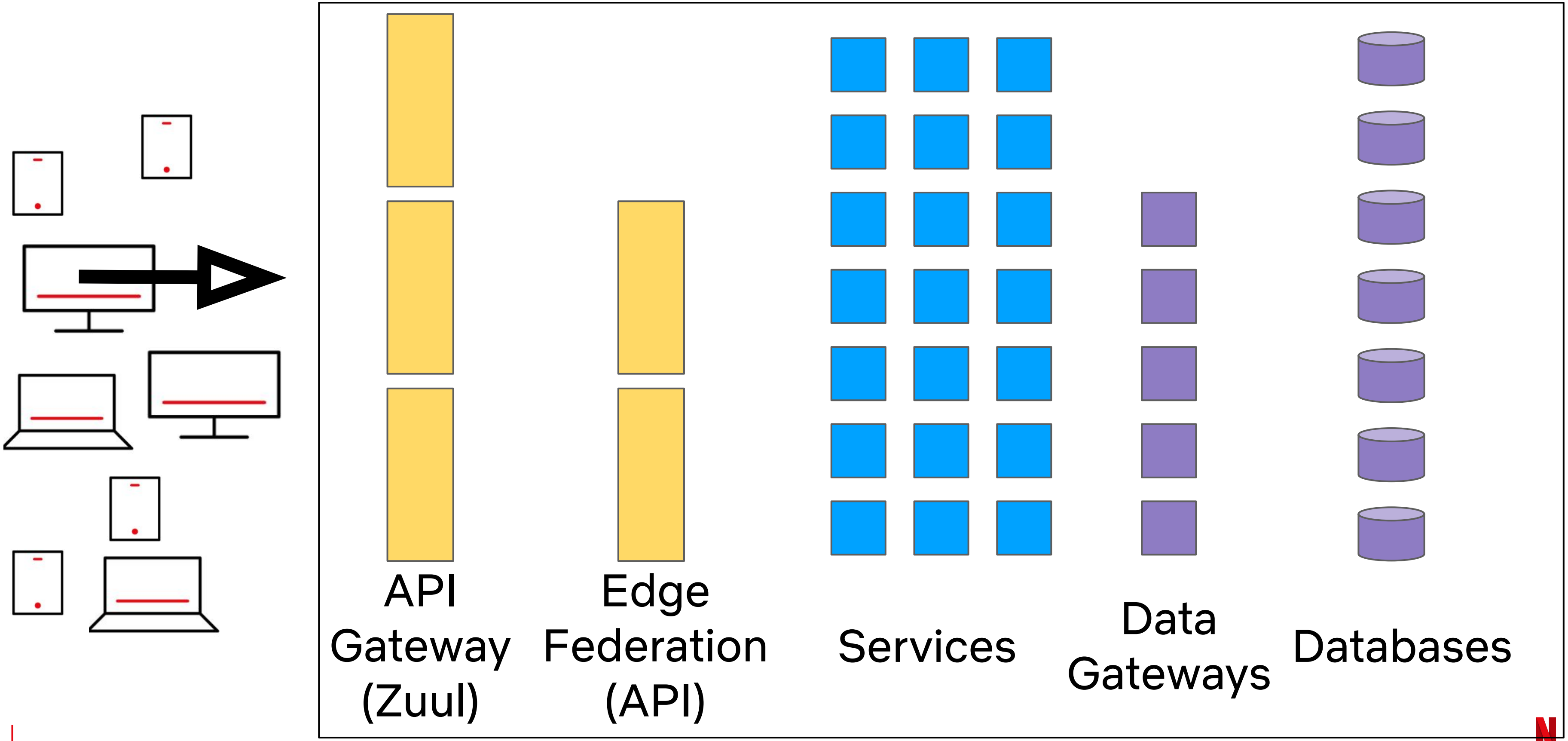
Content Driven Demand



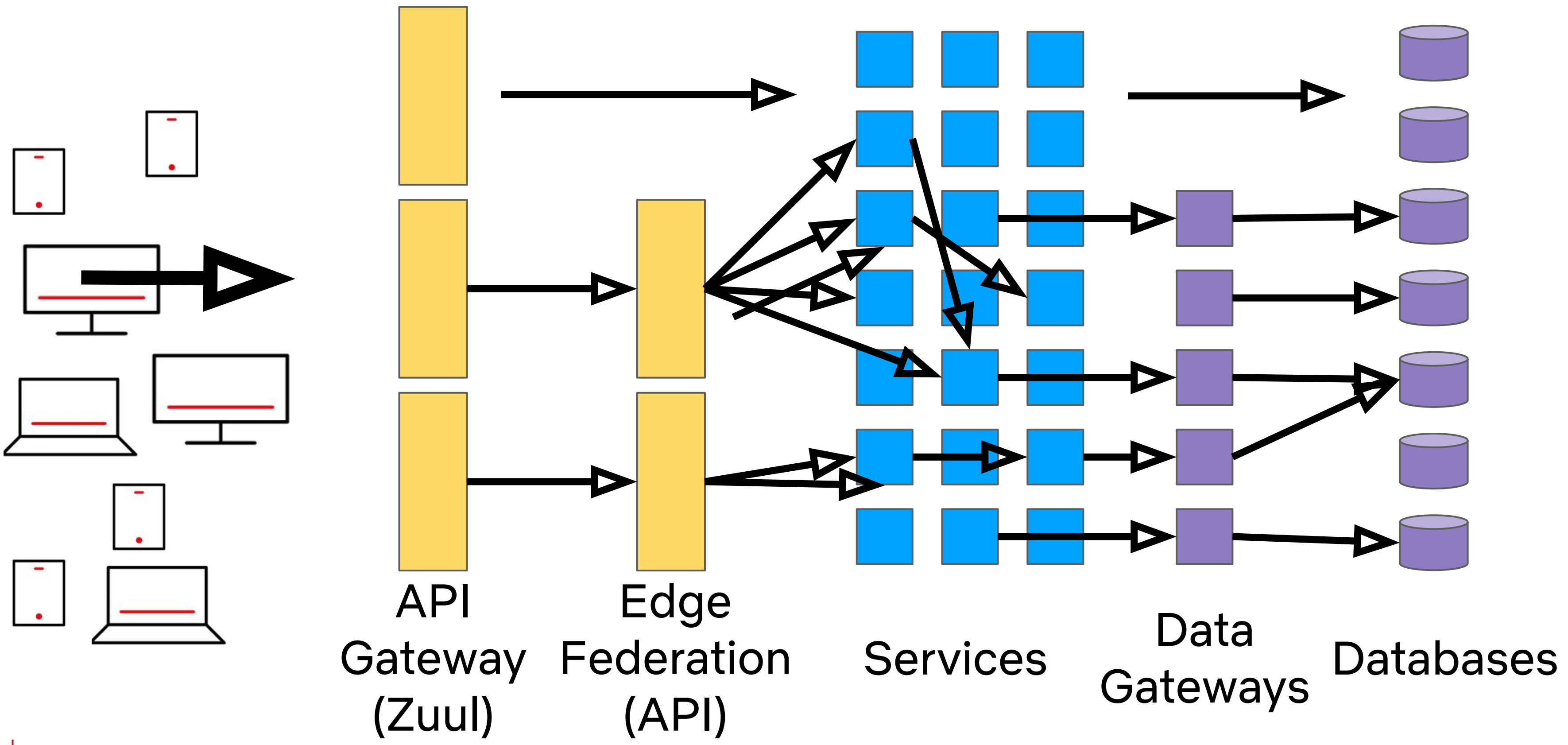
Device Driven Demand



Microservices



Microservices



Traffic **Demand**

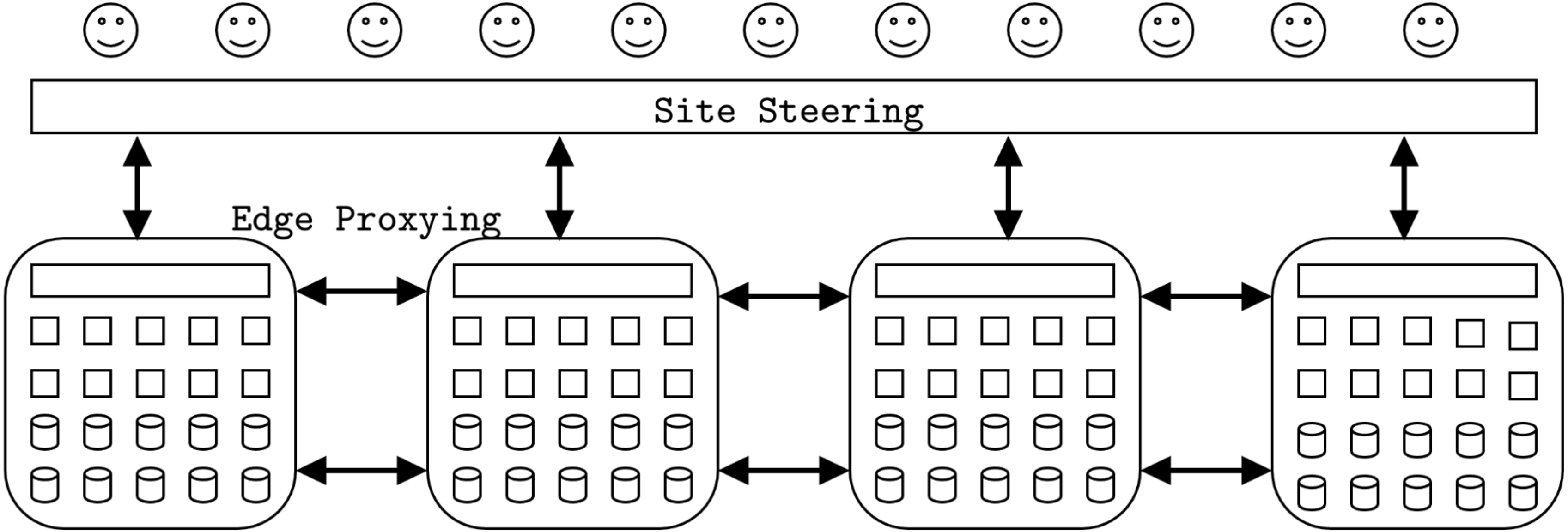
Compute **Supply**

Resilience
Techniques

Global Traffic Demand

Global architecture
Traffic balancing
Traffic shifting

Global Architecture



Full Active Database and Cache Replication

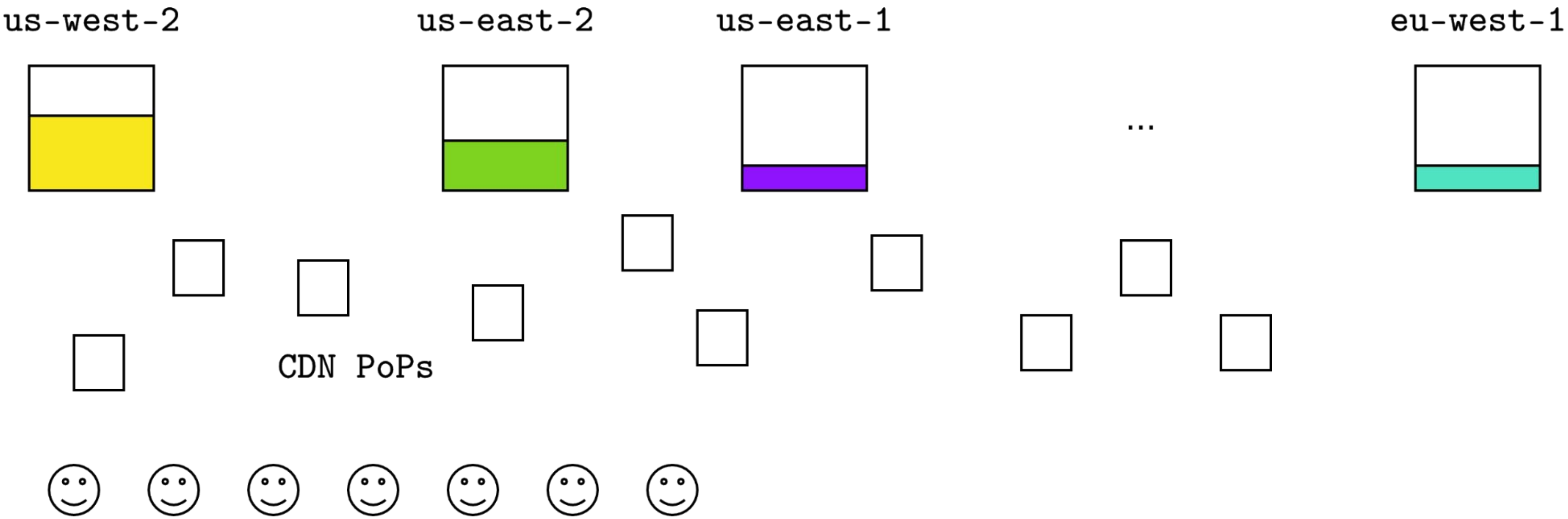
Latency

Bias users towards the **closest** Region or PoP

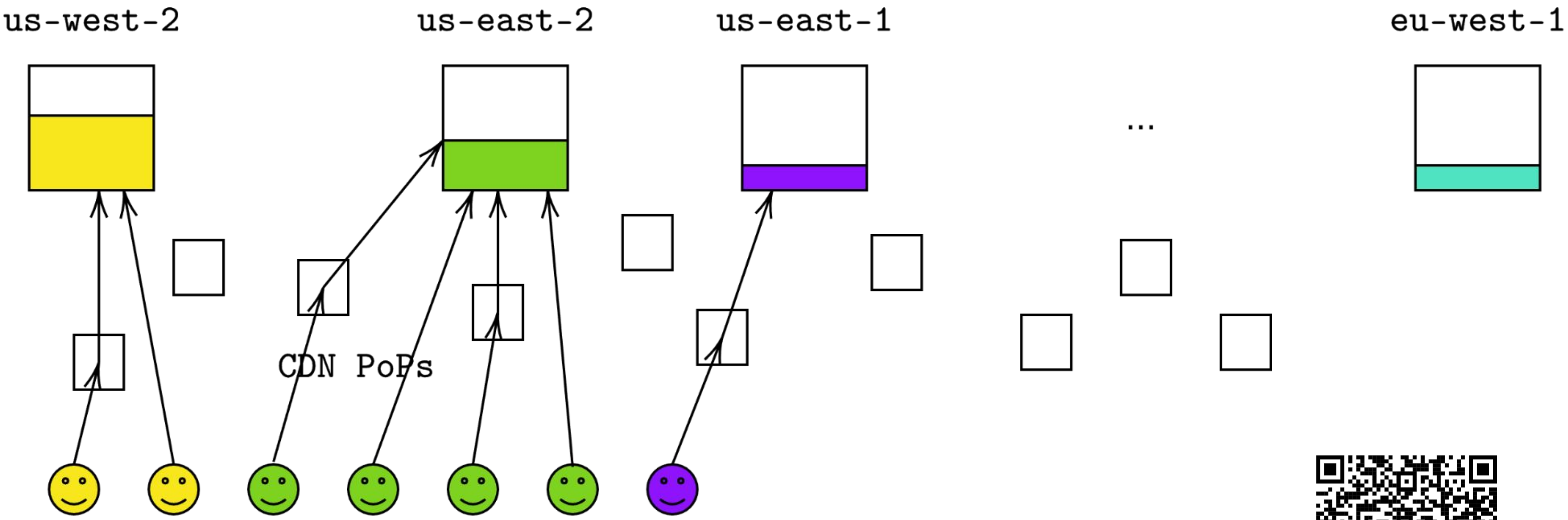
Availability

Bias users towards **least loaded** Region or PoP

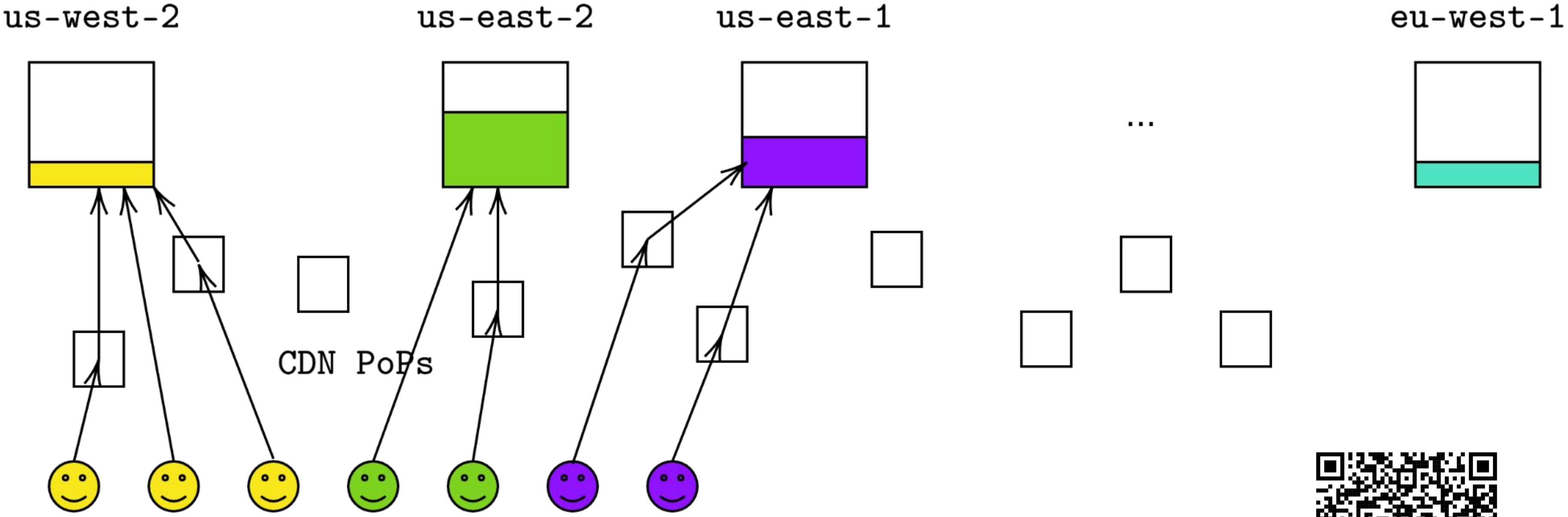
Control Plane Steering



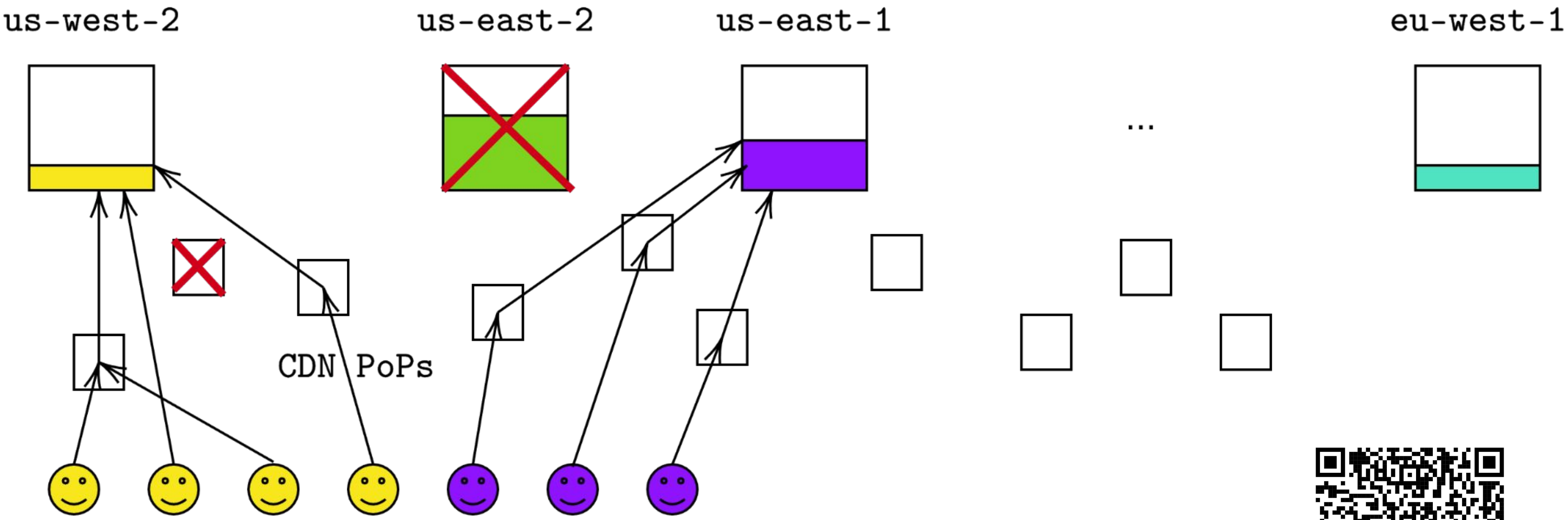
Control Plane Steering



Control Plane Steering

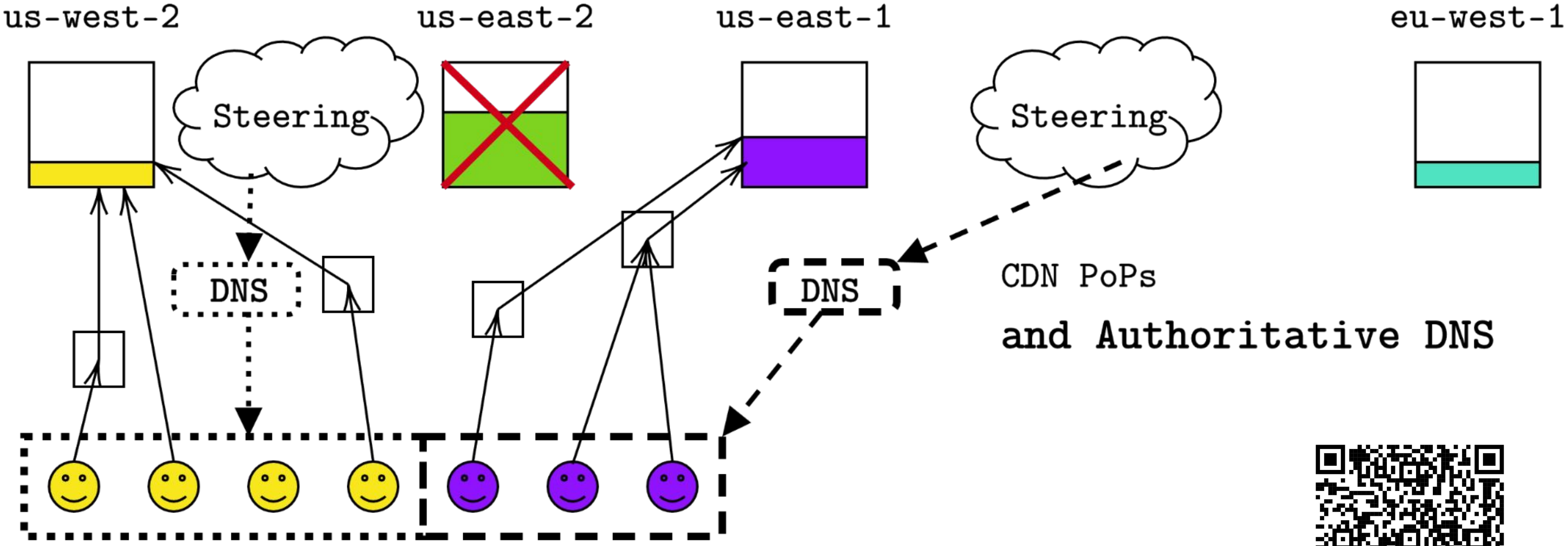


Control Plane Steering



InfoQ: Sergey Fedorov and Niosha Behnam on **Load Balancing Traffic**

Solution: Netflix DNS and Steering



Predict

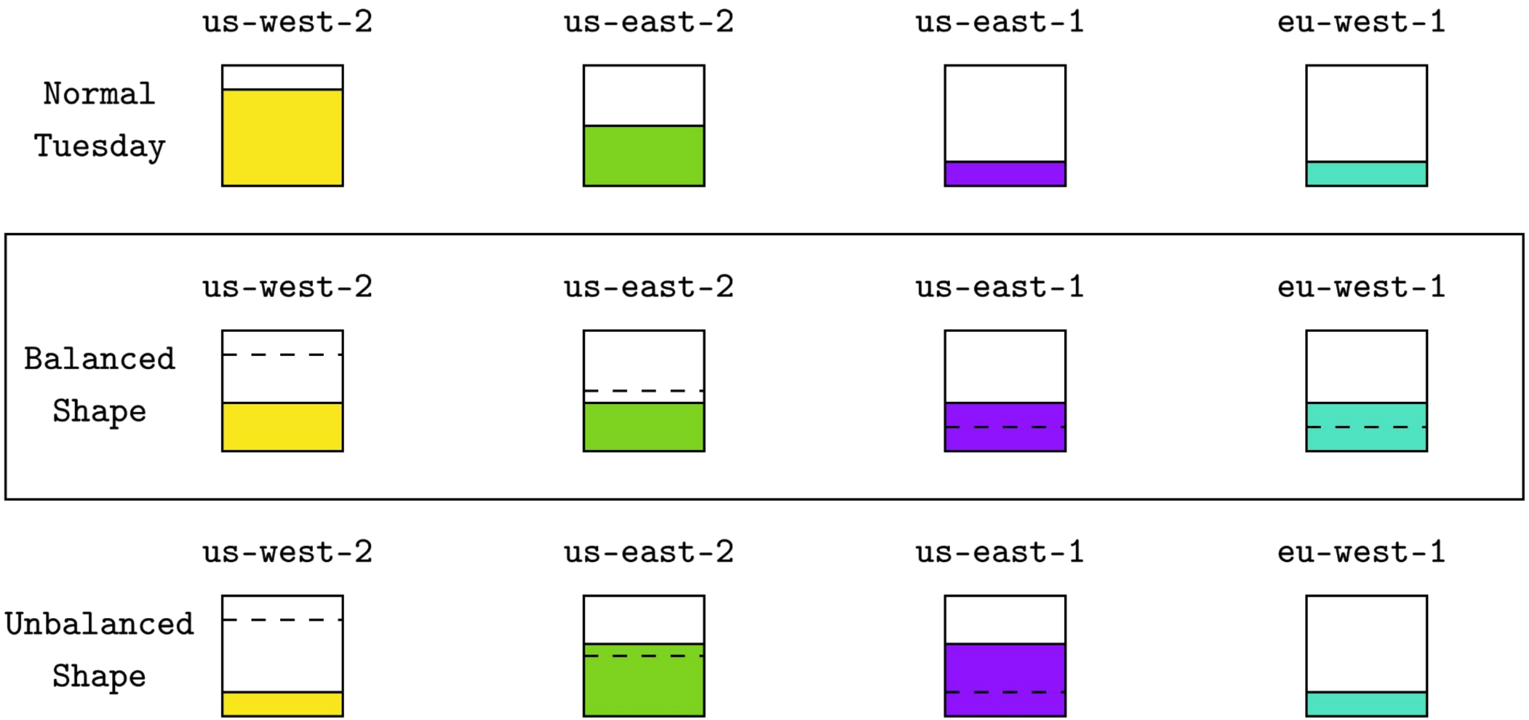
Shape traffic in anticipation

React

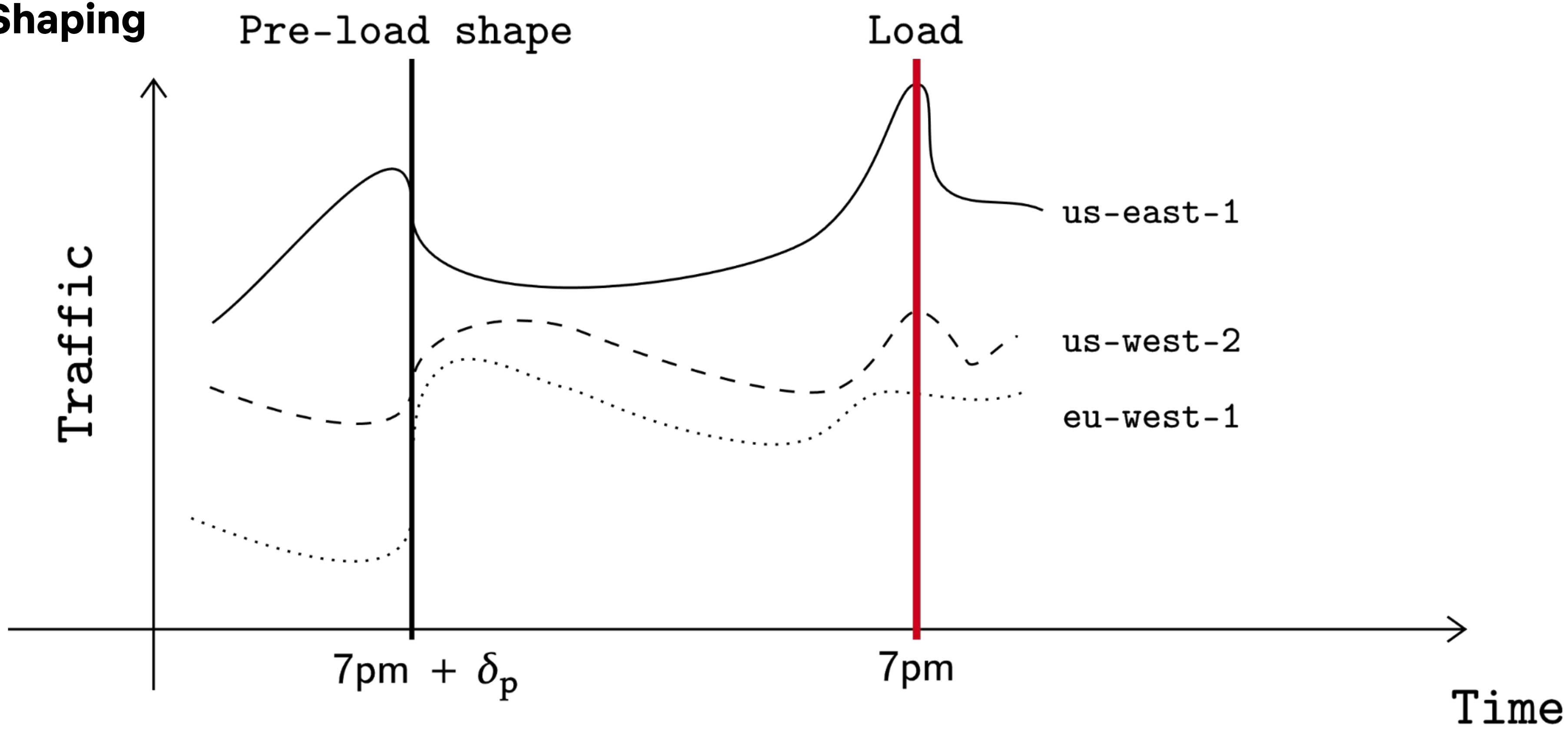
Restore balance

Predict Traffic Spikes

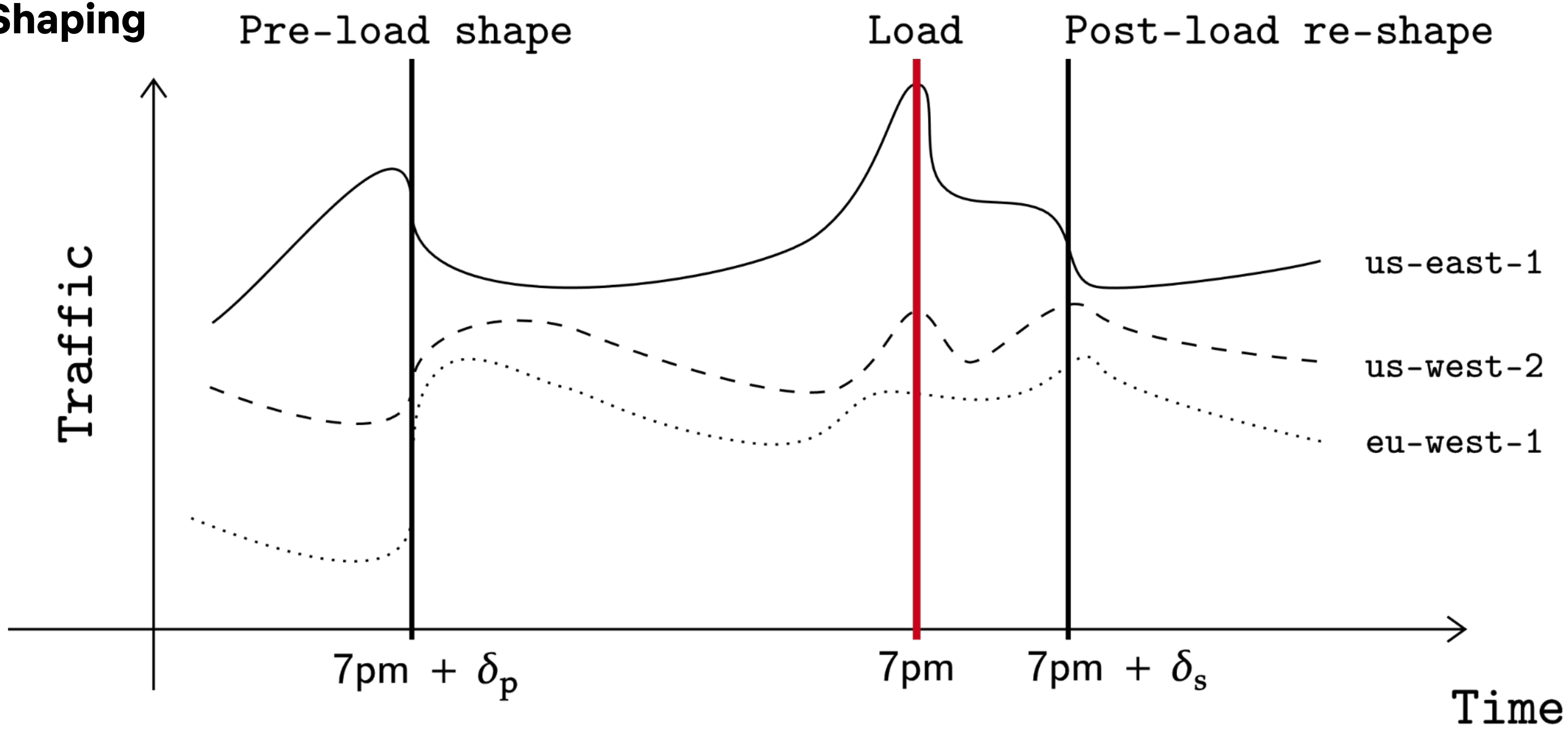
SVOD usage is reasonably predictable



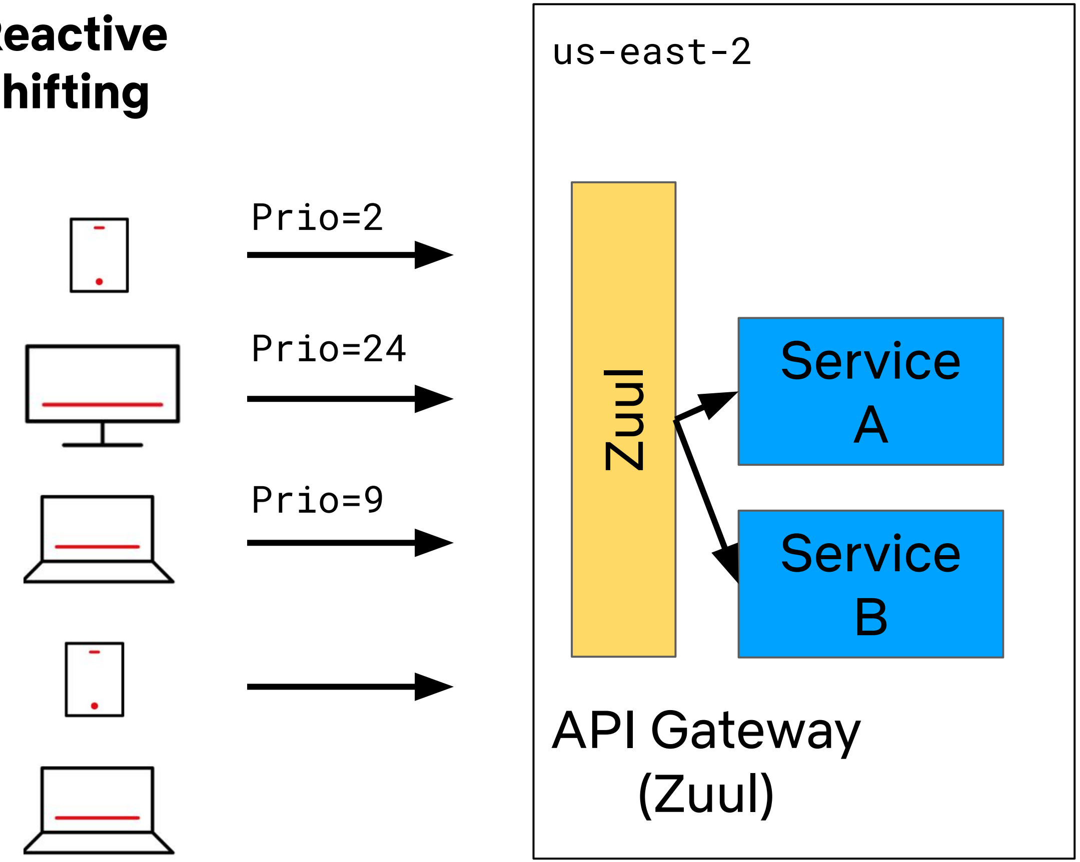
Predictive Shaping



Reactive Shaping

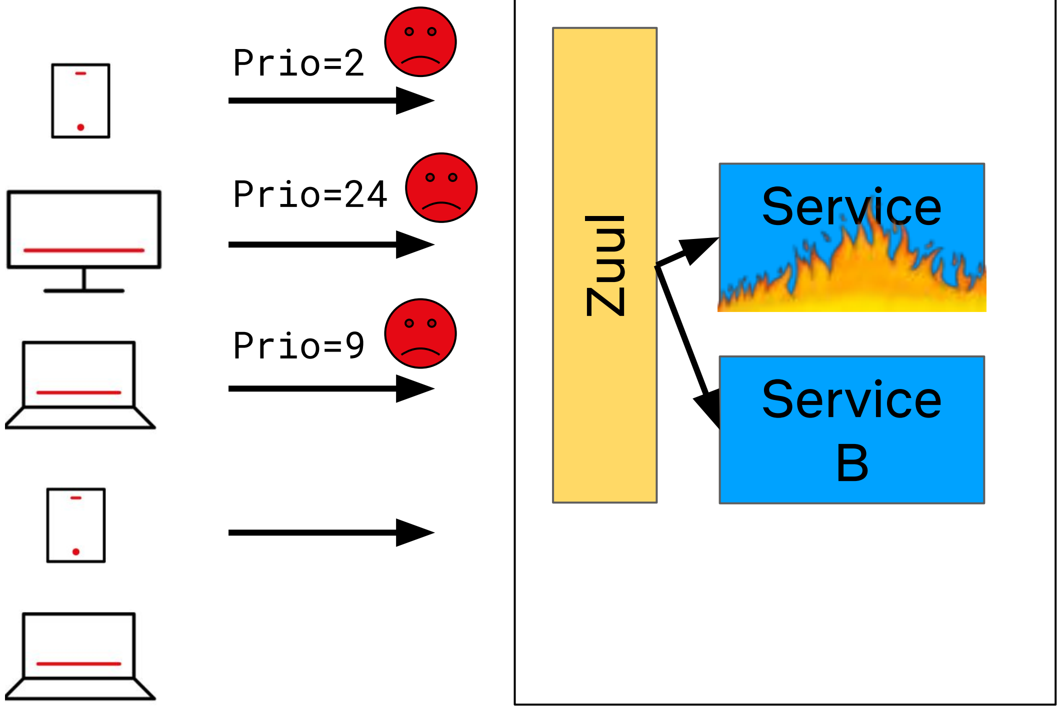


Reactive Shifting



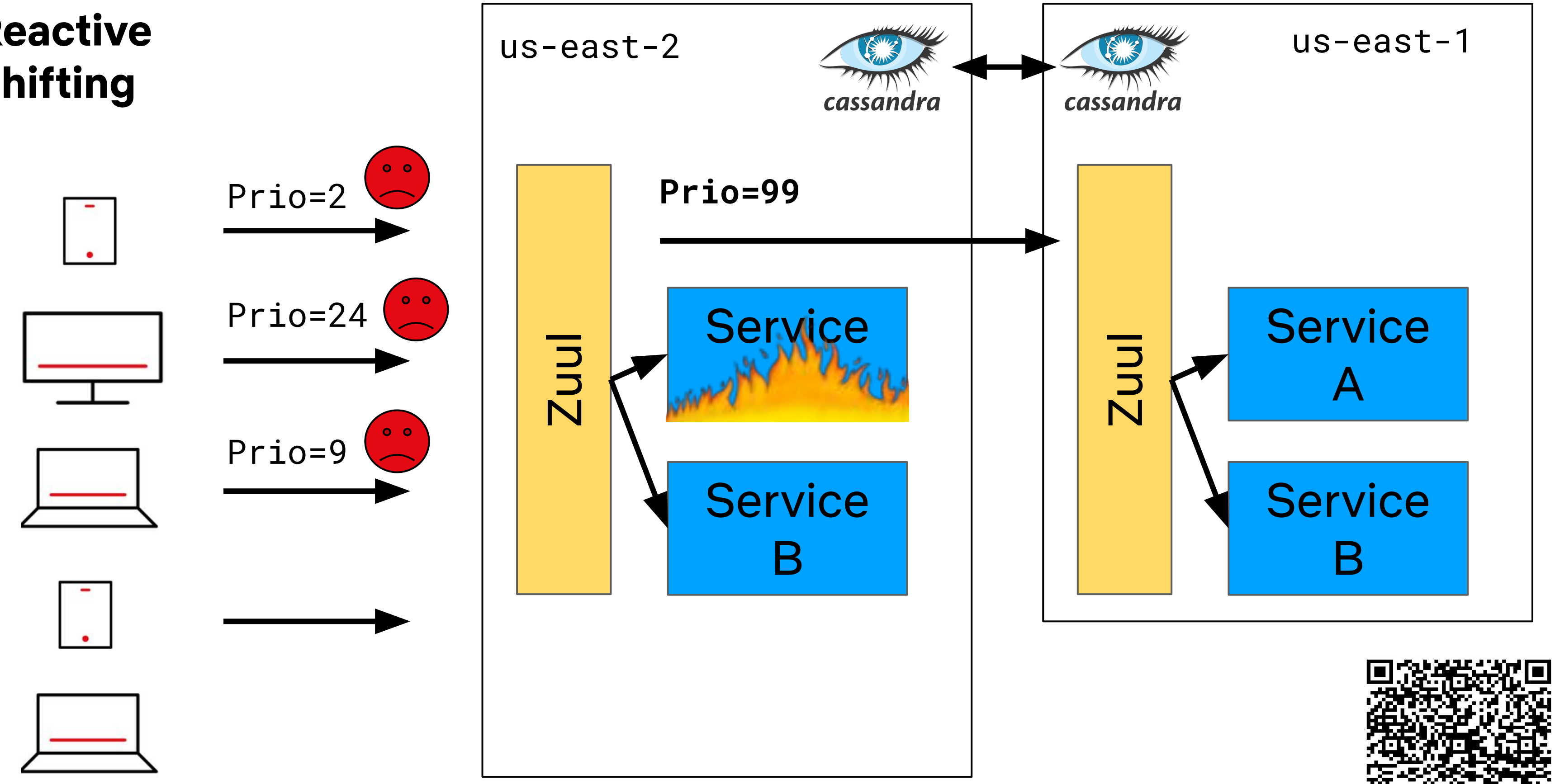
Netflix Tech Blog: Manuel Correa, Arthur Gonigberg et. al. on **Prioritized Load Shedding**

Reactive Shifting



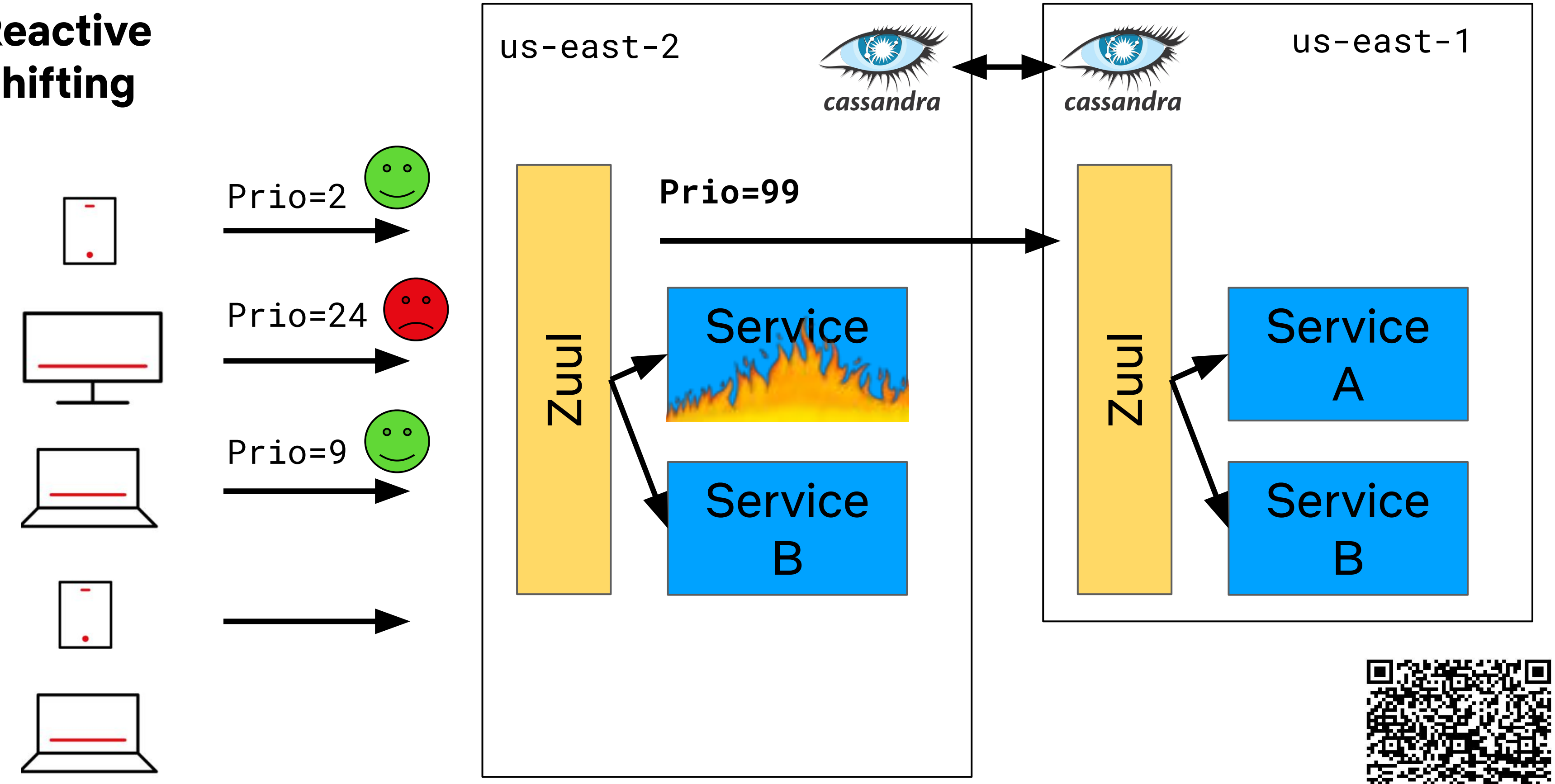
Netflix Tech Blog: Manuel Correa, Arthur Gonigberg et. al. on **Prioritized Load Shedding**

Reactive Shifting



Netflix Tech Blog: Manuel Correa, Arthur Gonigberg et. al. on **Prioritized Load Shedding**

Reactive Shifting



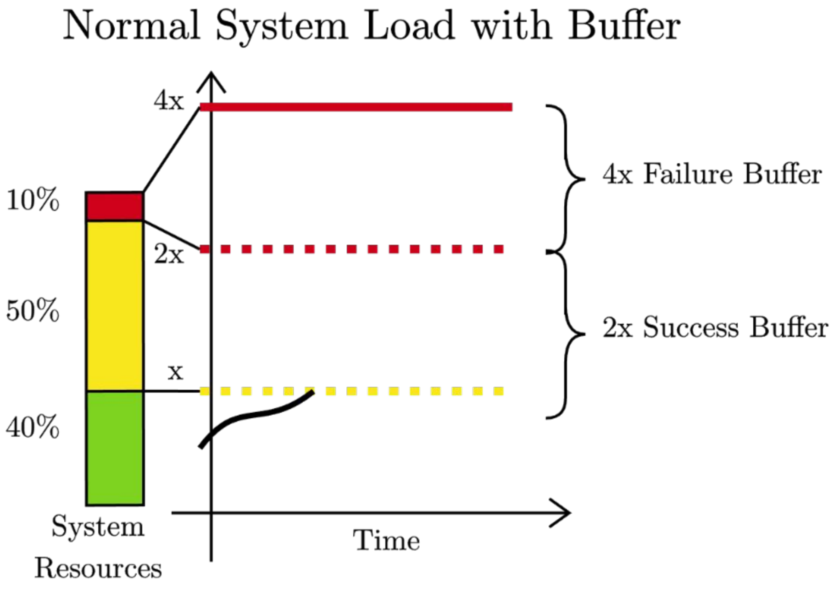
Netflix Tech Blog: Manuel Correa, Arthur Gonigberg et. al. on **Prioritized Load Shedding**

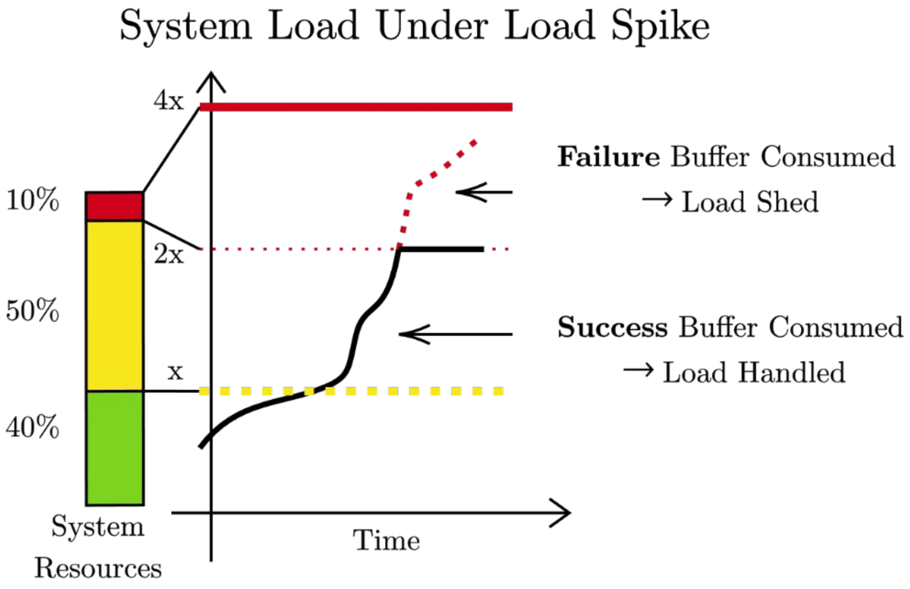
Supply of Computers

Cloud Realities
Predictive Scaling
Reactive Scaling

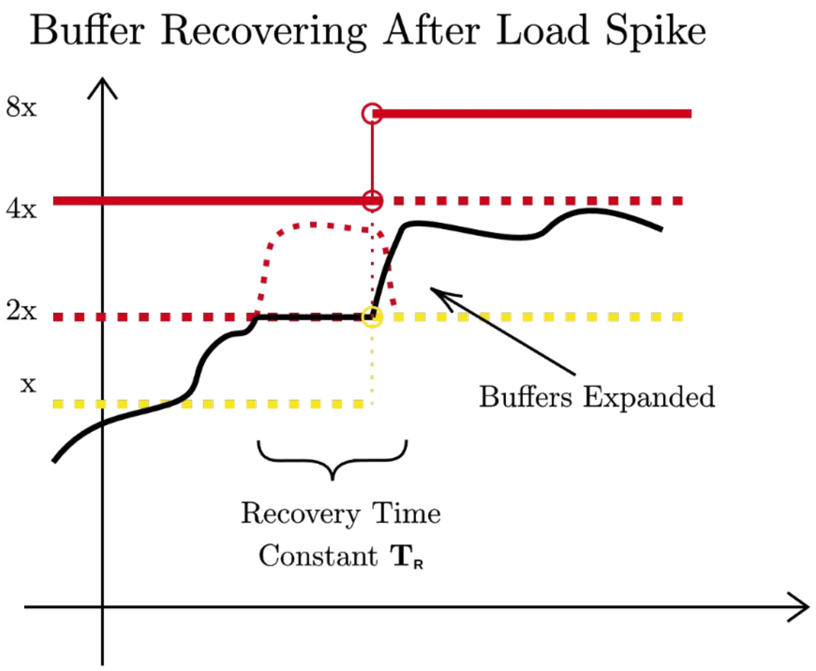

JESUS SAVES

Reason about headroom with Buffers





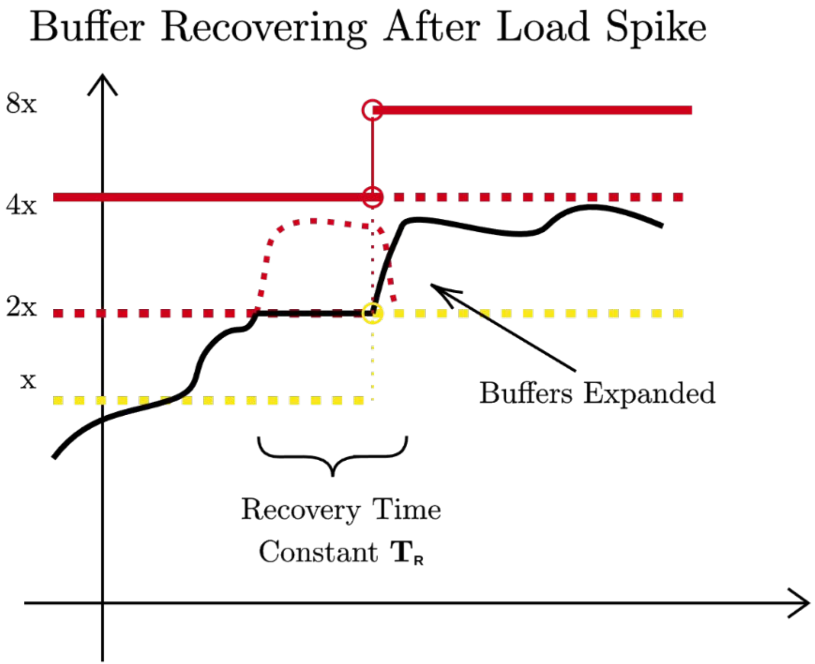
Buffer is Linked to Business Outcomes



Buffer = F(
Criticality Tier,
Business Domain,
Recovery Time Constant
)

Low utilization is a tradeoff!

Buffer is Linked to Business Outcomes

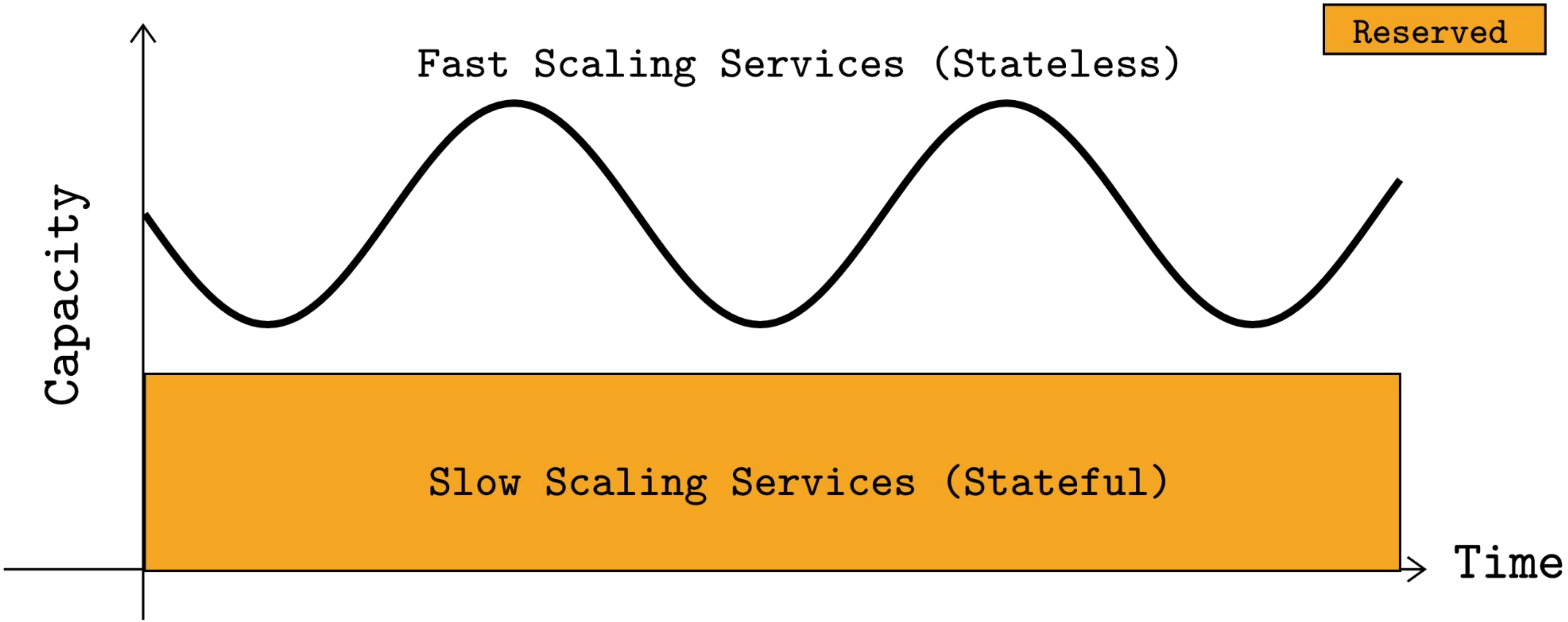


Buffer = F(
Criticality Tier,
Business Domain,
Recovery Time Constant
)

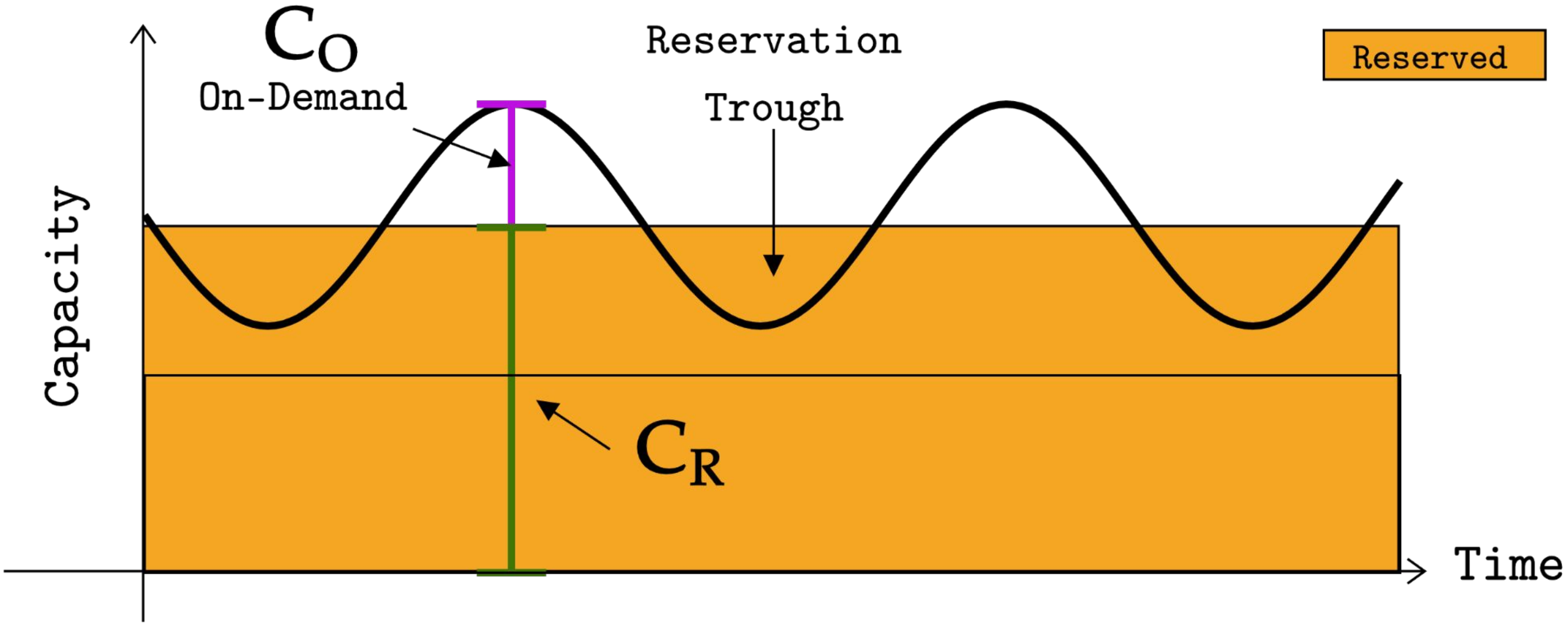
Low utilization is a tradeoff!

Fast recovering services need less buffer:
Stateless: 3-5m vs **Stateful: 30-60m**

Cloud Reality #1
Still have to Plan



Cloud Reality #1
Still have to Plan



Cloud Reality #1
Capacity Planning

It doesn't make financial sense to reserve everything

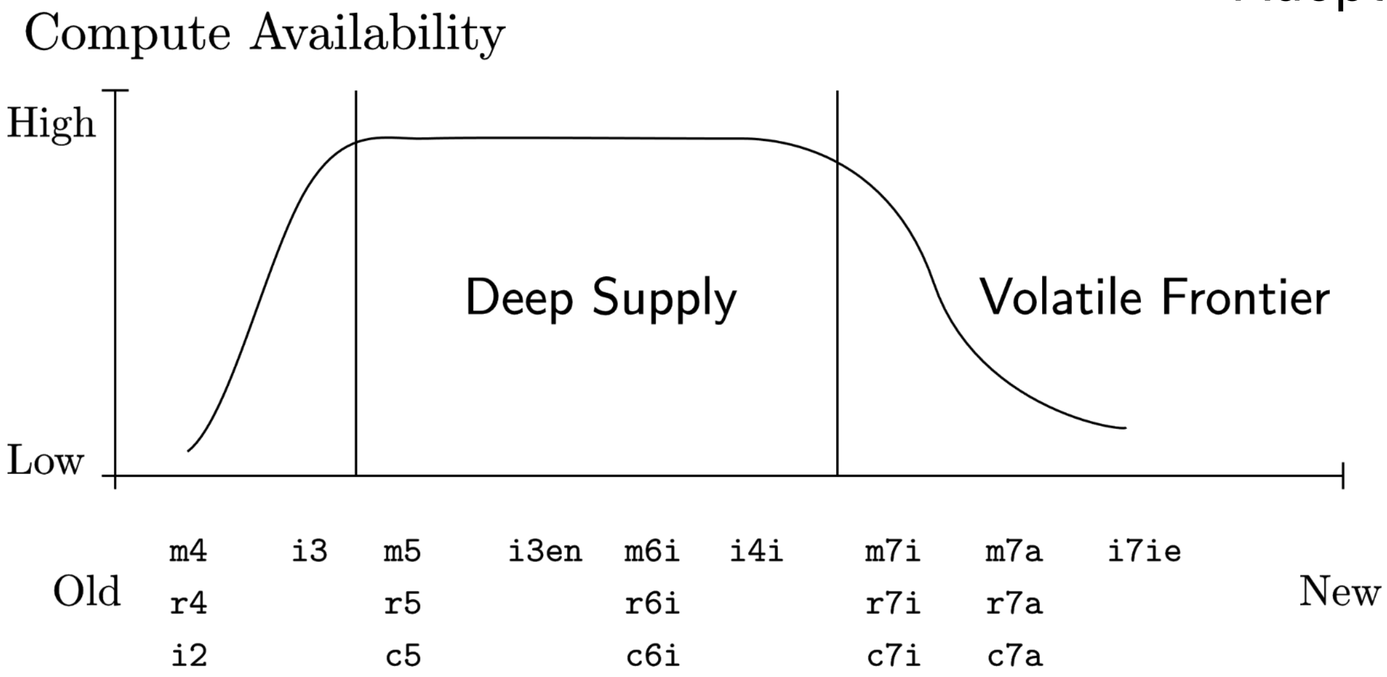
$$Capacity = C_R + C_O$$

$$C_R \propto f(time, pricing)$$

You will likely be using **On-Demand or Spot**

Cloud Reality #2
Computers have
Variable Supply

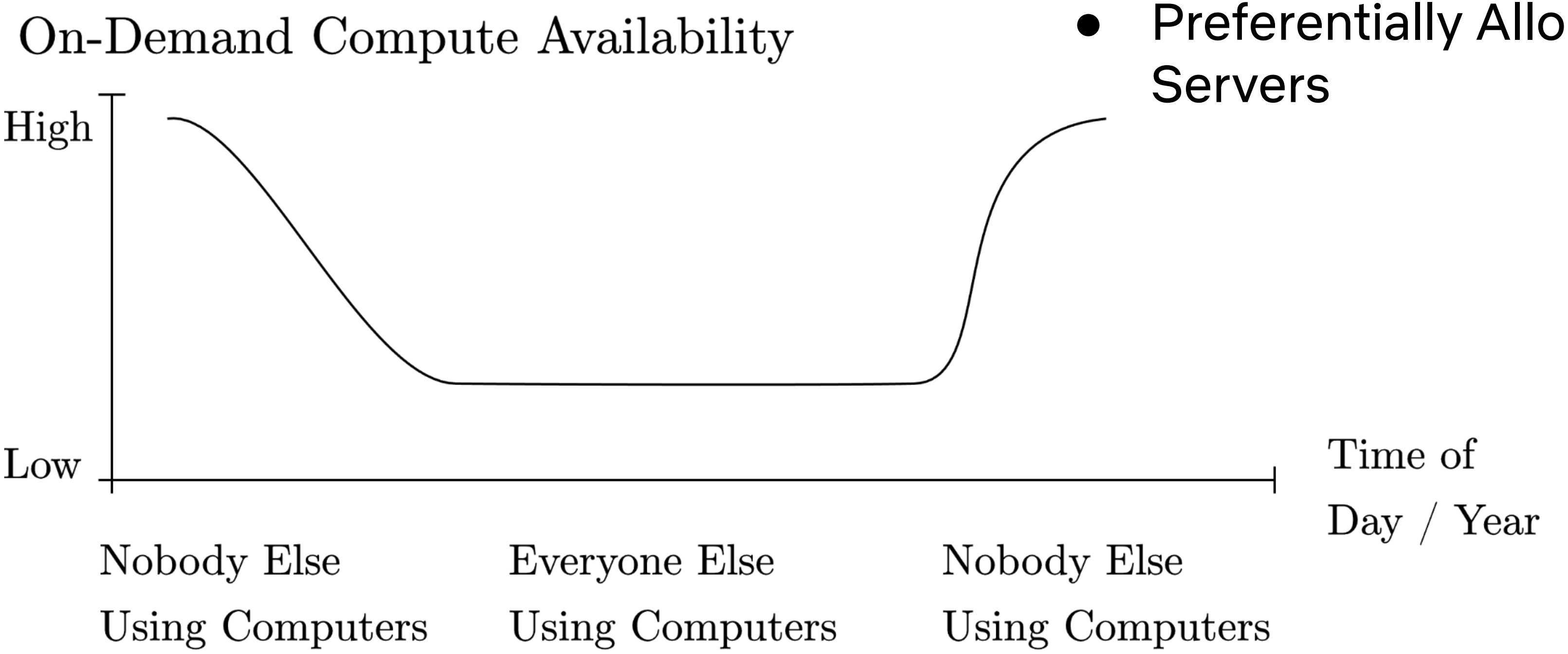
Have to *carefully* Adopt New Shapes



Cloud Reality #2
Computers have
Variable Supply

Options:

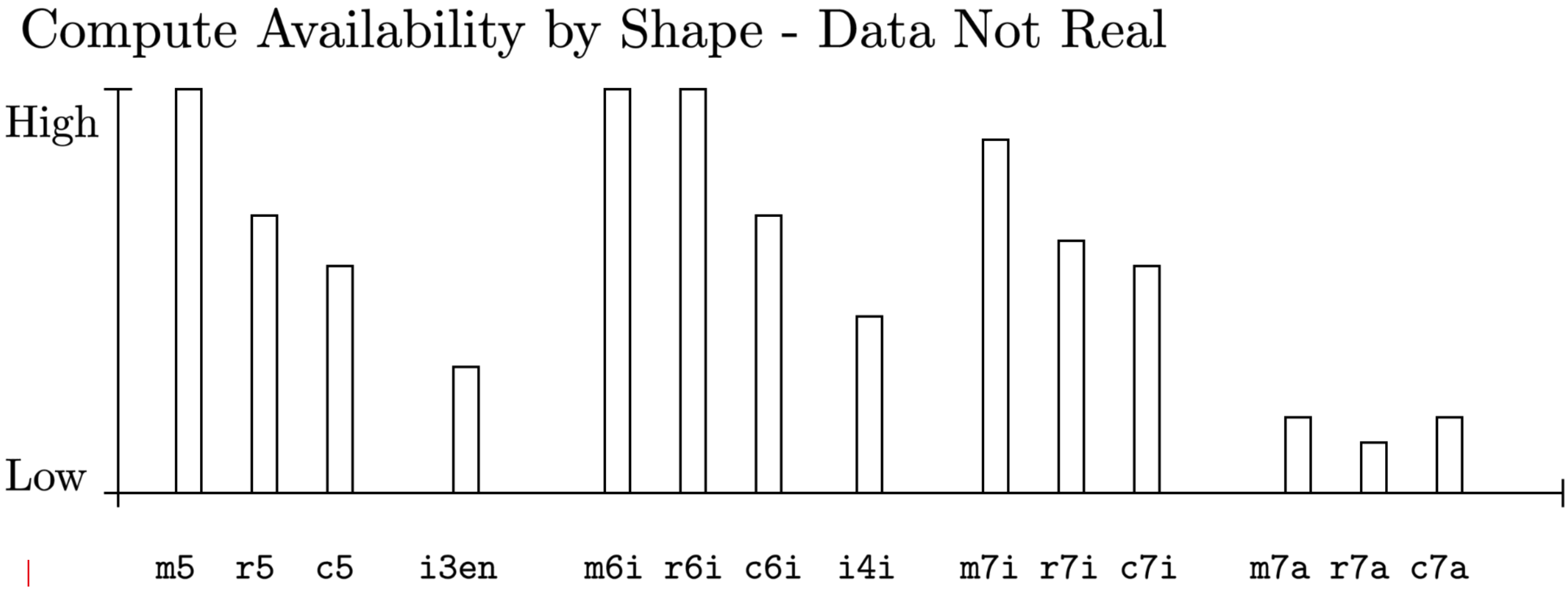
- Reservations
- On Demand Reservations
- Leverage Buffers
- Preferentially Allocating Servers



Cloud Reality #2
Computers have
Variable Supply

More flexibility == More Capacity

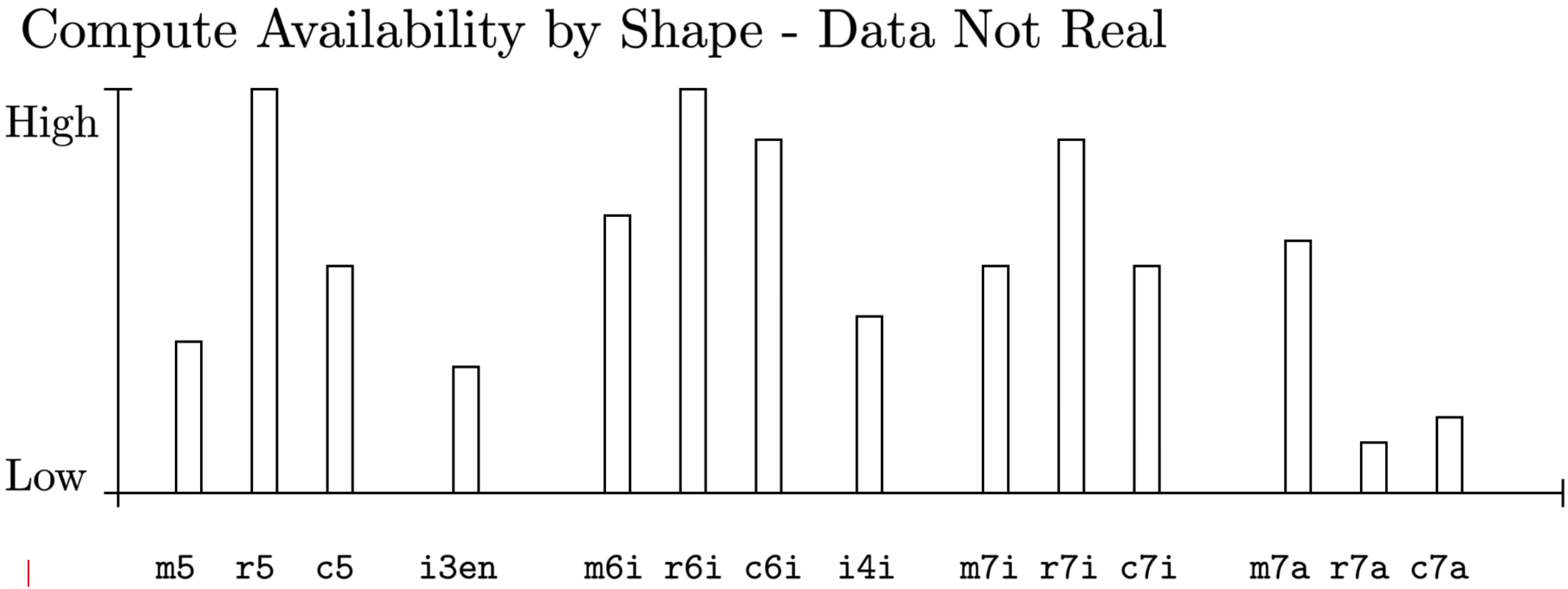
Assign workloads by resources not names



Cloud Reality #2
Computers have
Variable Supply

More flexibility == More Capacity

Assign workloads by resources not names



Cloud Reality #3 Different Servers are Meaningfully Different


```
"m7a.4xlarge": {
  "name": "m7a.4xlarge",
  "cpu": 16,
  "cpu_cores": 16,
  "cpu_ghz": 3.7,
  "cpu_ipc_scale": 1.5,
  "ram_gib": 61.04,
  "net_mbps": 6250.0,
  "drive": null
},
```



```
"m6id.4xlarge": {
  "name": "m6id.4xlarge",
  "cpu": 16,
  "cpu_cores": 8,
  "cpu_ghz": 3.5,
  "cpu_ipc_scale": 1.0,
  "ram_gib": 61.04,
  "net_mbps": 6250.0,
  "drive": {
    "name": "ephem",
    "size_gib": 885,
    "read_io_per_s": 268332,
    "write_io_per_s": 134168,
    "single_tenant": false,
    "read_io_latency_ms": {
      "low": 0.1,
      "mid": 0.125,
      "high": 0.17,
      "confidence": 0.9,
      "minimum_value": 0.05,
      "maximum_value": 2.0
    }
  }
},
```


Service Capacity Modeling

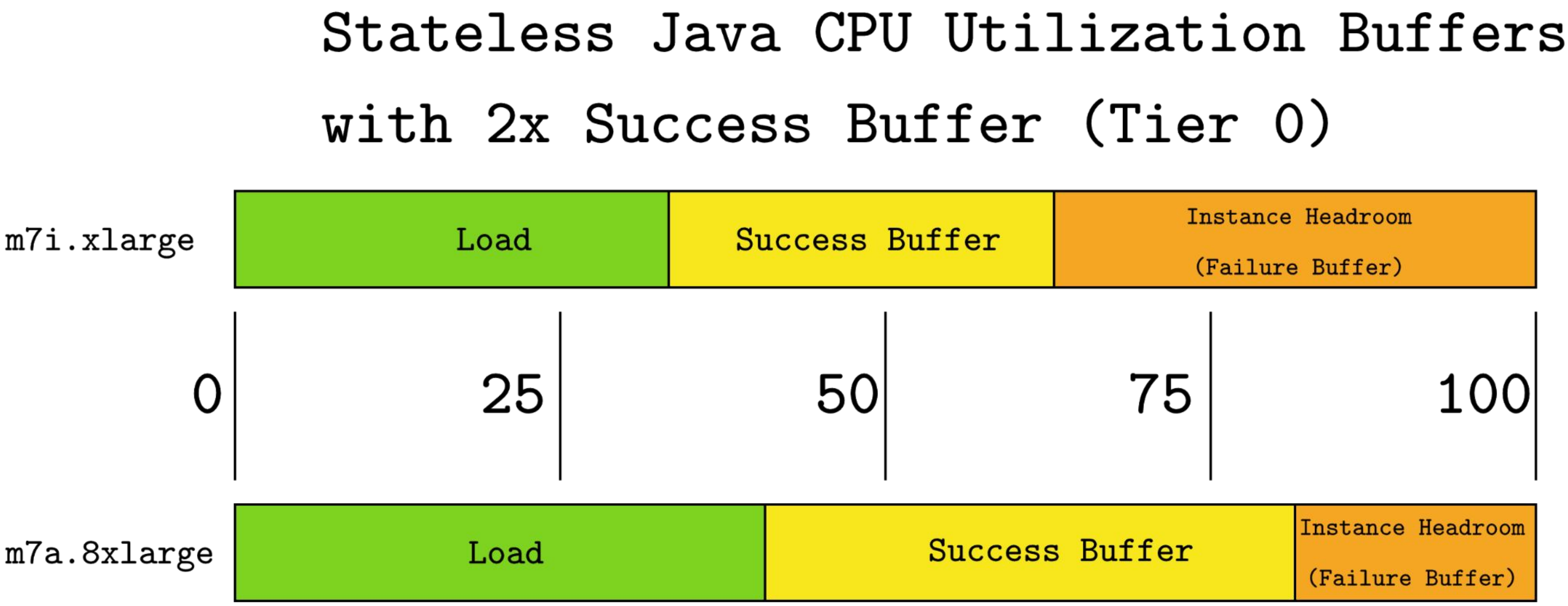
Build passing

A generic toolkit for modeling capacity requirements in the cloud.

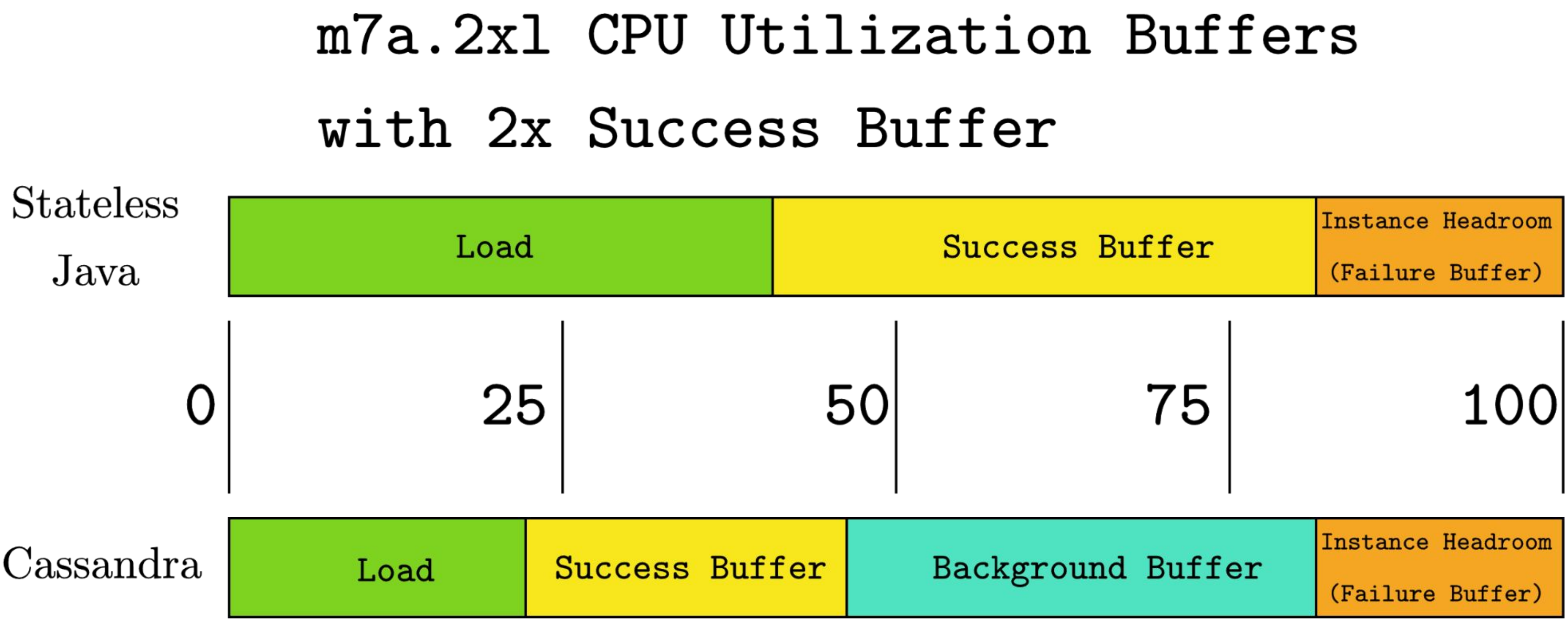
Buffer Differs By Server Type

	Name	Failure Buffer %	Tier 1 Target %	Tier 0 Target %
19	m7a.xlarge	0.70	0.46	0.35
21	m7a.4xlarge	0.84	0.56	0.42
22	m7a.8xlarge	0.88	0.59	0.44
29	m7i.xlarge	0.62	0.41	0.31
31	m7i.4xlarge	0.79	0.53	0.40
32	m7i.8xlarge	0.85	0.57	0.42

Buffer Differs By Server Type



Buffer Differs By Workload



Predict

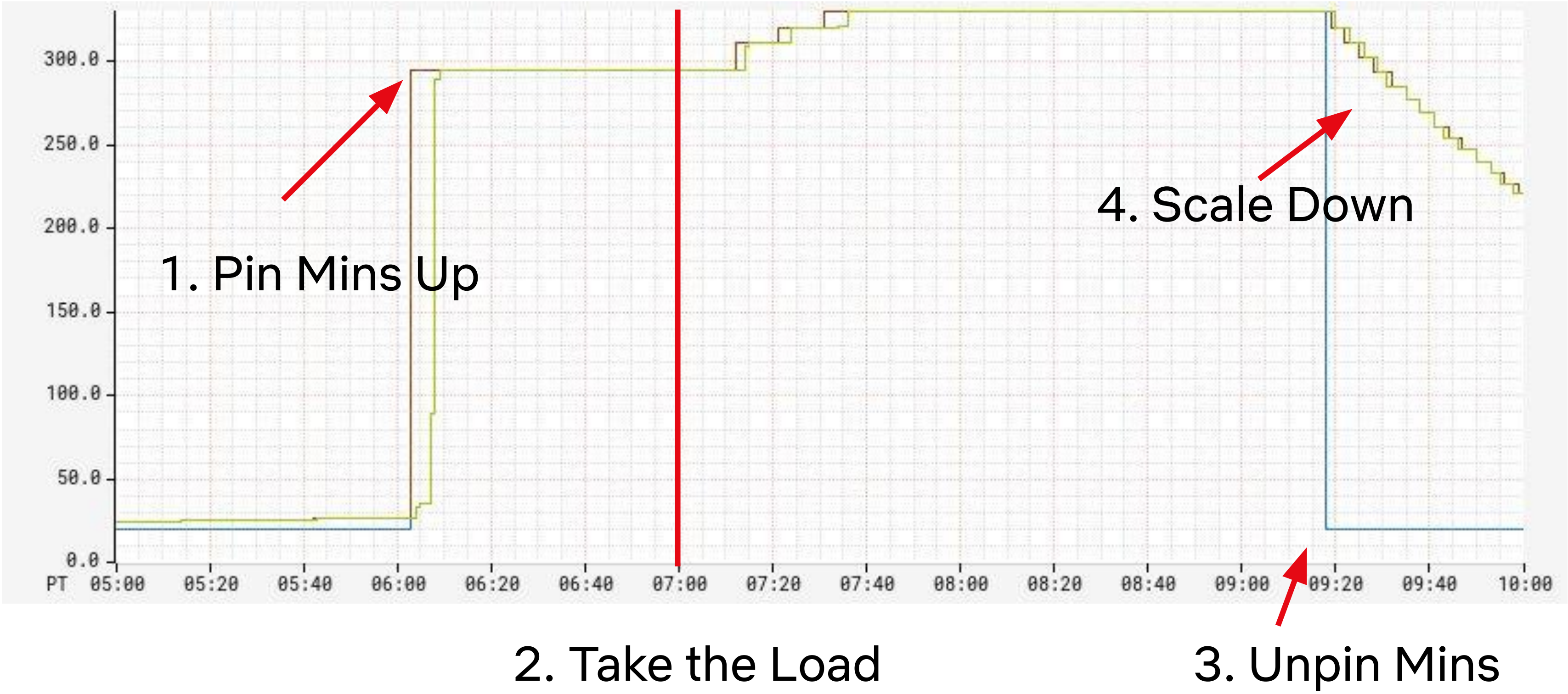
Pre-scale services for demand
Pre-scale stateful services

React

Autoscale out of trouble

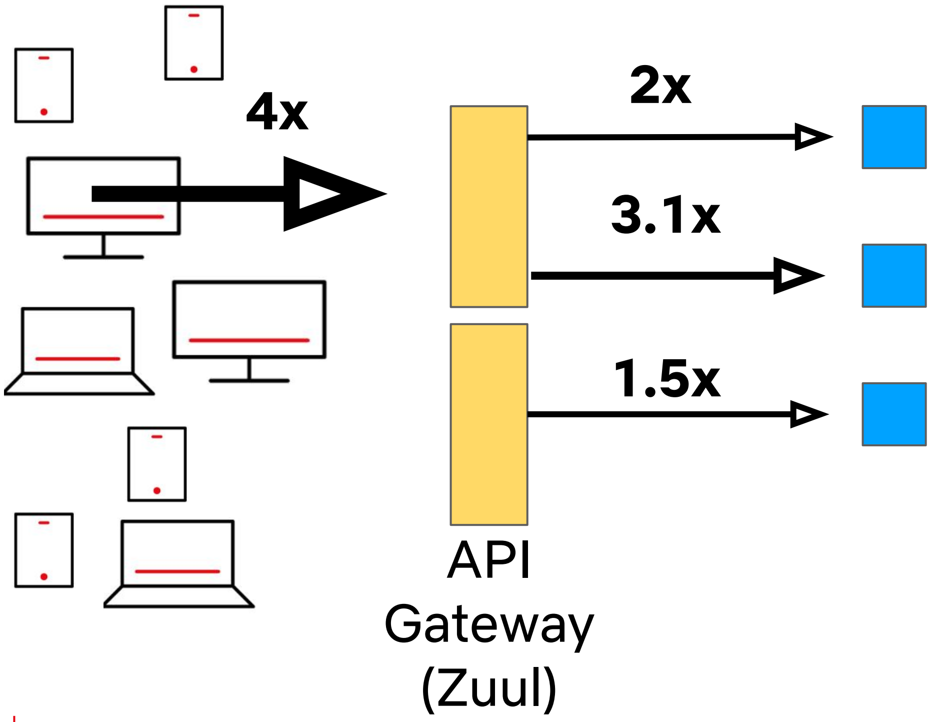
Pre-Scale for Load

Based on Projected Traffic, Pin Mins



Different for Every Service

Call Graph is not Uniform in Traffic *or* Criticality



Tier 0 to Streaming Playback

Tier 1 to Streaming Discovery

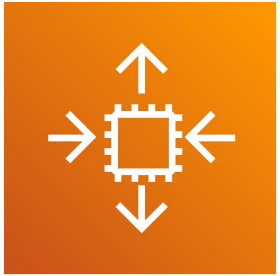
Tier 3 to Personalization

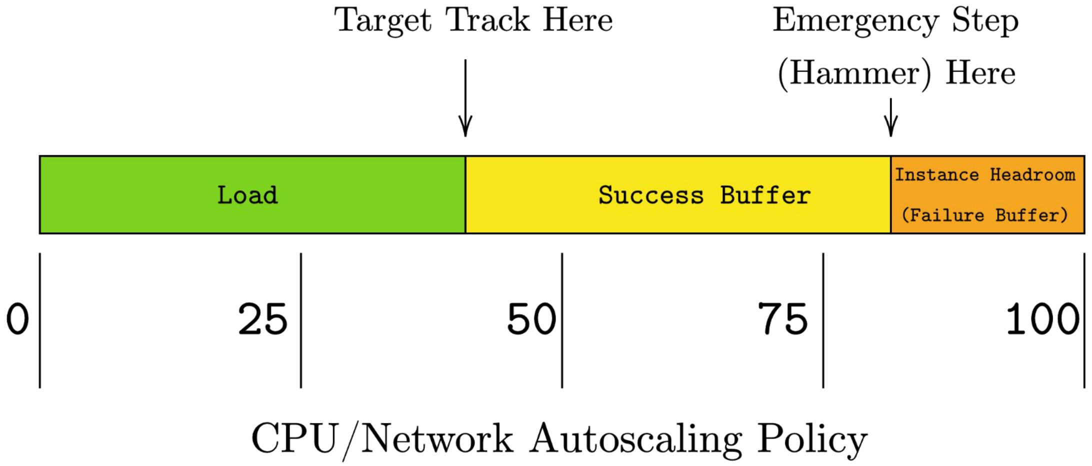
Netflix Tech Blog: Niosha Behnam on **Evolving Regional Evacuation**

What if we are wrong?

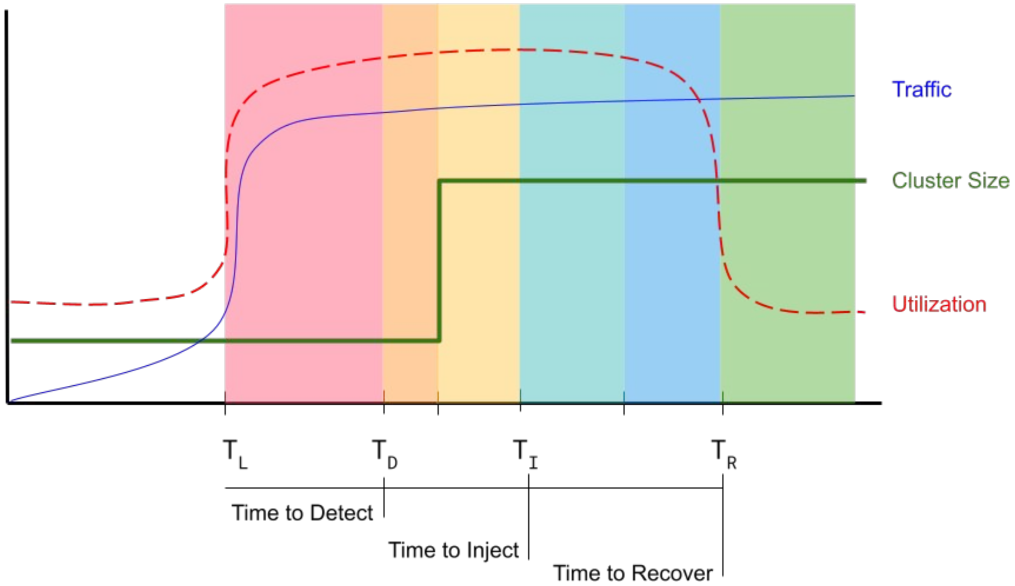
React

What if we are wrong?





Autoscaling Can be Slow

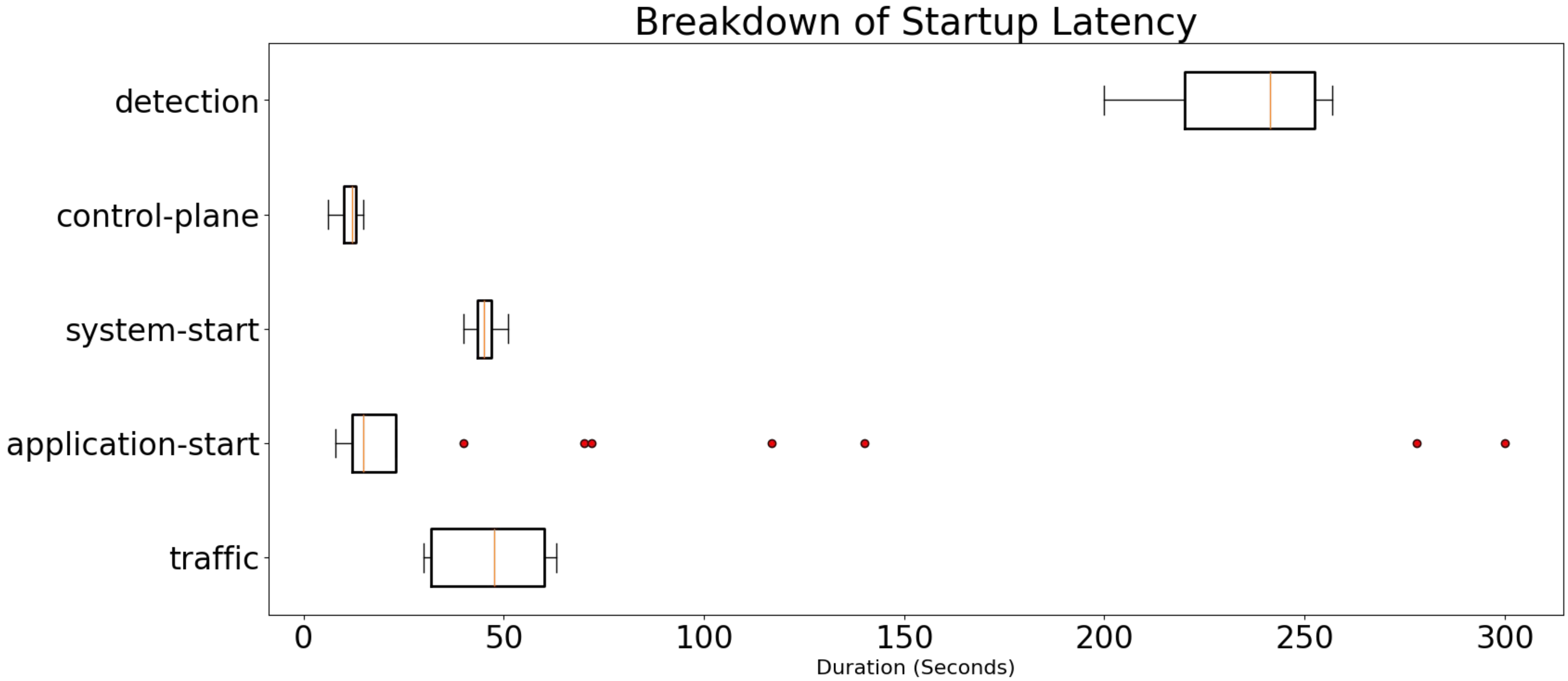


Traffic spikes at T_L causing Utilization to increase. The Cluster Size increases only after delays for Detection and Control Plane. After a delay for OS Startup, we reach the point usable capacity is injected T_I. Utilization remains high until Application Startup and Load Balancing delays allow new capacity to take traffic - then we Recover at T_R.

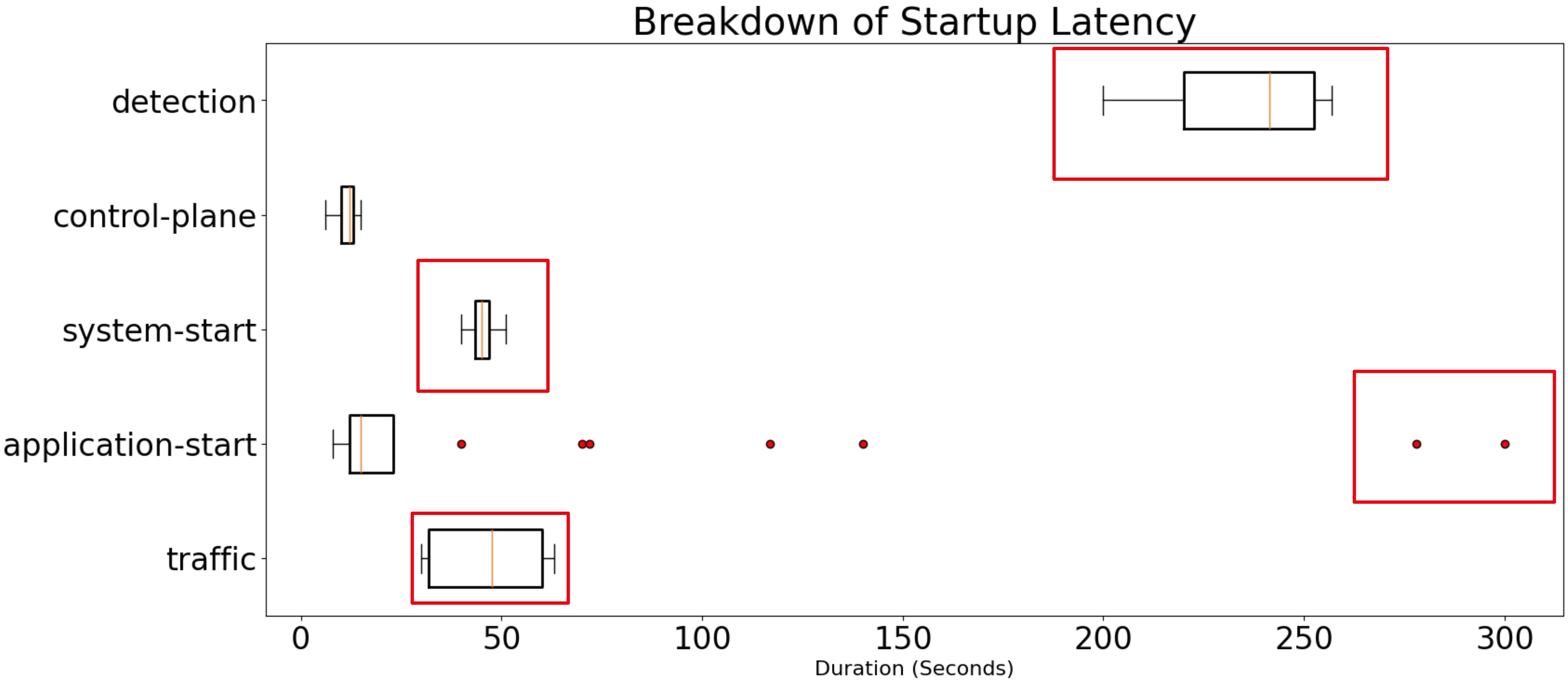
Various Sources of Latency!

Detection (Alert)
Control Plane (Hardware)
System Startup (Kernel)
Application Startup (App)
Traffic (Discovery)

Break it Down



Break it Down



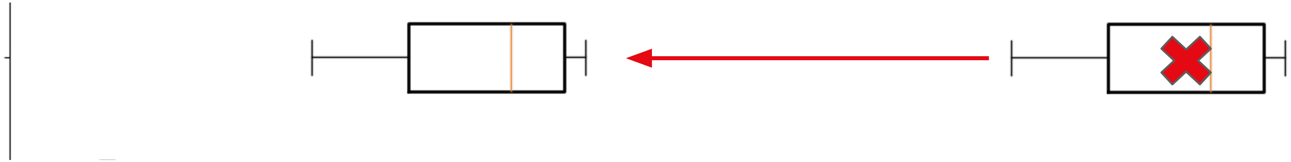
Solve Each Part

High Resolution Metrics

Observe actual start latency (cooldown)

Observe RPS/CPU and Hammer on RPS ladder.

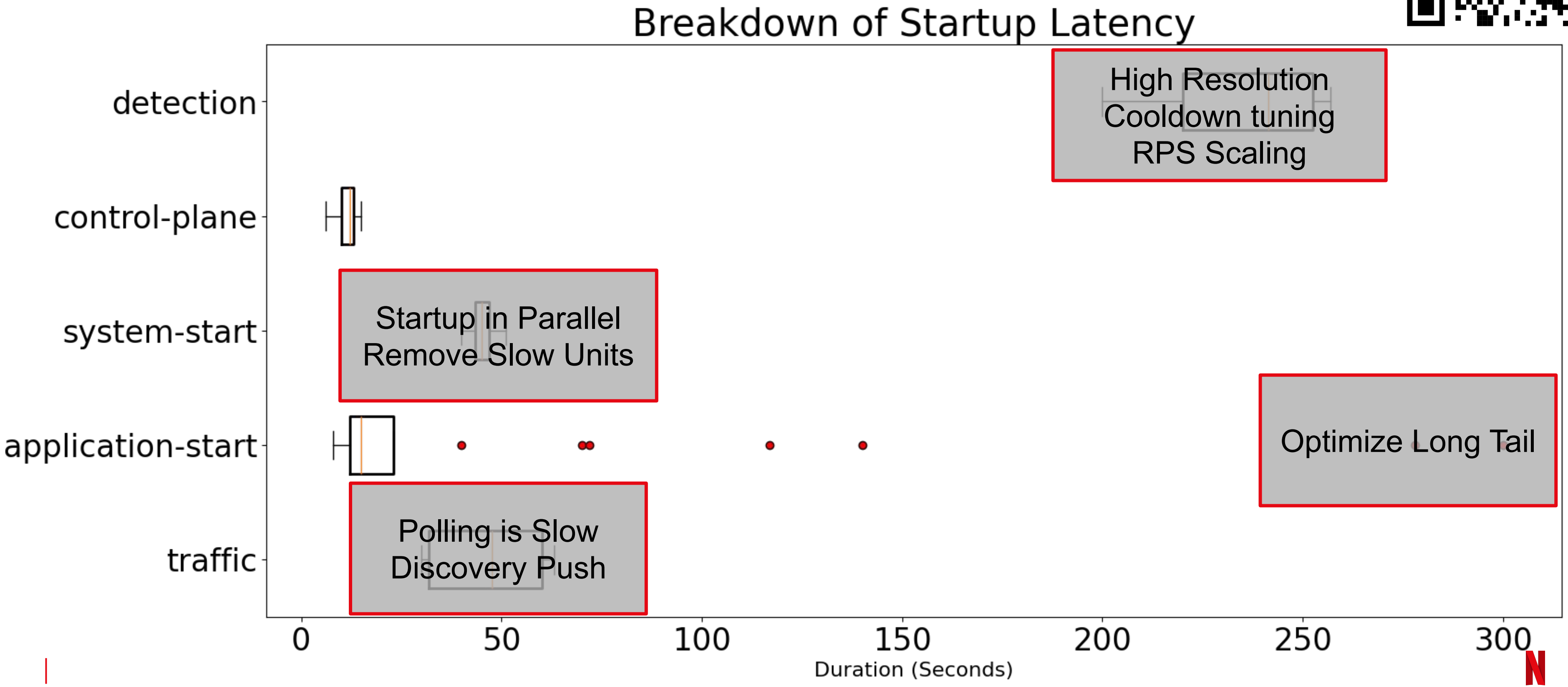
detection



AWS Re:Invent 2024: Ryan Schroeder on **Autoscaling Faster**

Break it Down

AWS Re:Invent 2024: Ryan Schroeder on **Autoscaling Faster**



Does This Really Work?

Tier 0 Before Tuning
10x Load Spike TTR

8-15M

Snail: https://commons.wikimedia.org/wiki/File:Snail.jpg

Does This Really Work?

~70% reduction!

Tier 0 Before Tuning
10x Load Spike TTR

8-15M

Tier 0 *After* Tuning
10x Load Spike TTR

3-4M

Snail: https://commons.wikimedia.org/wiki/File:Snail.jpg

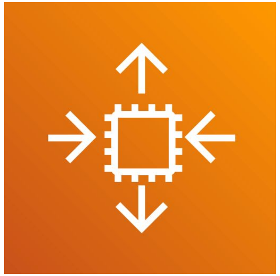
Stateless Resilience

Load Shedding
- CPU
- IO

Prioritization

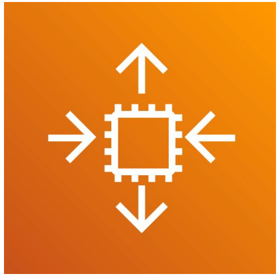
React

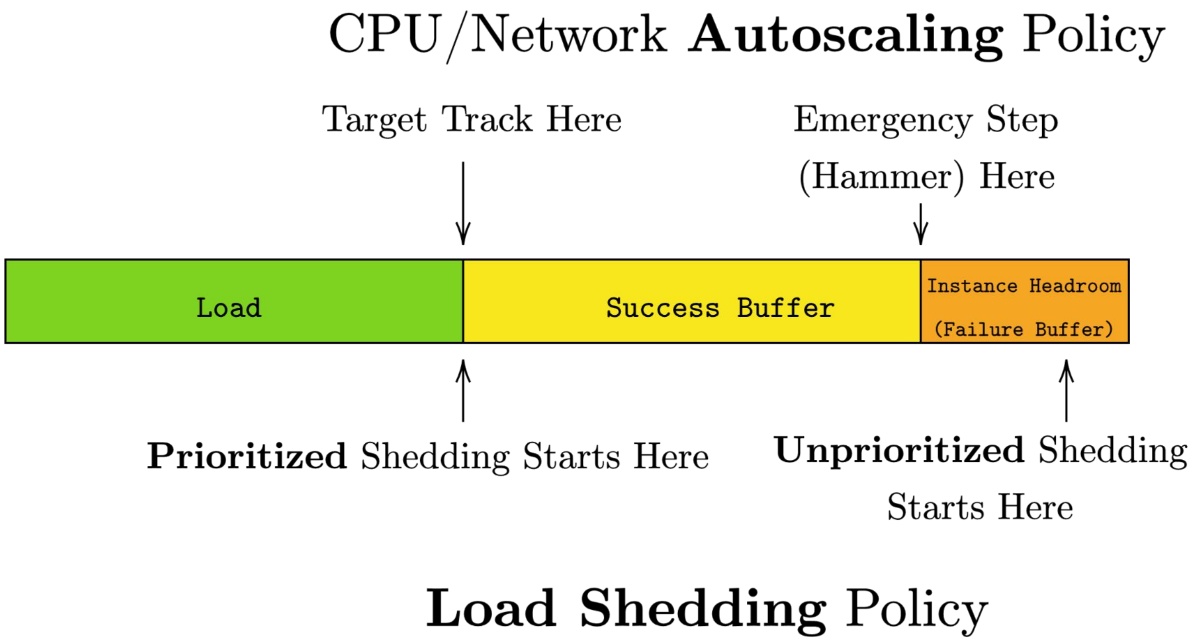
What to do *While* we are Wrong?



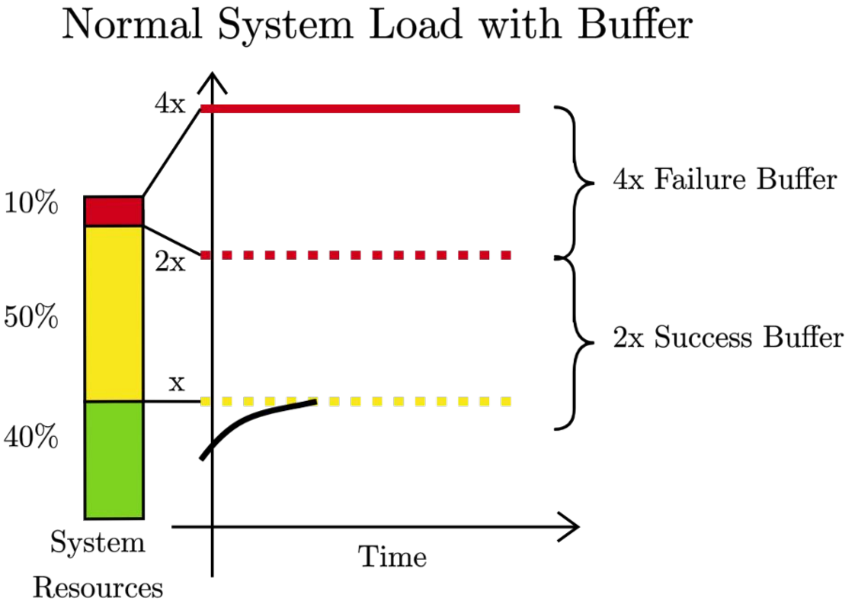
React

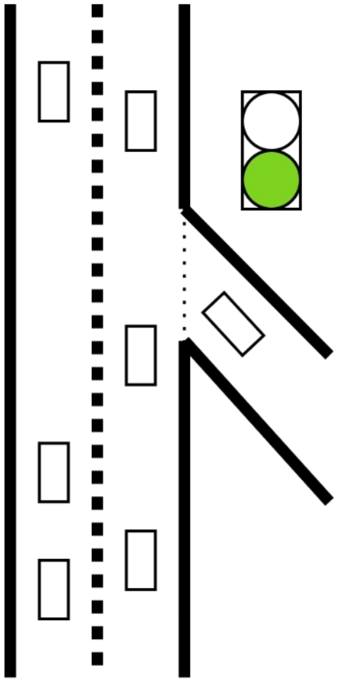
What to do *While* we are Wrong?





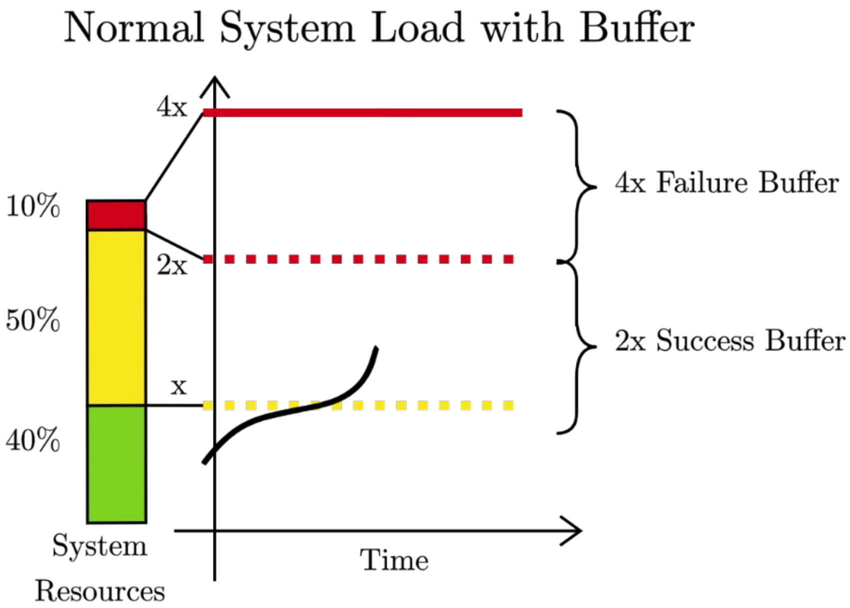
CPU Load Shedding

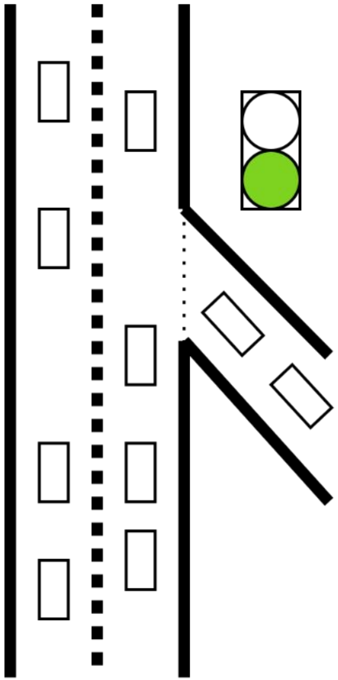




$$T_{transit} = 10m$$

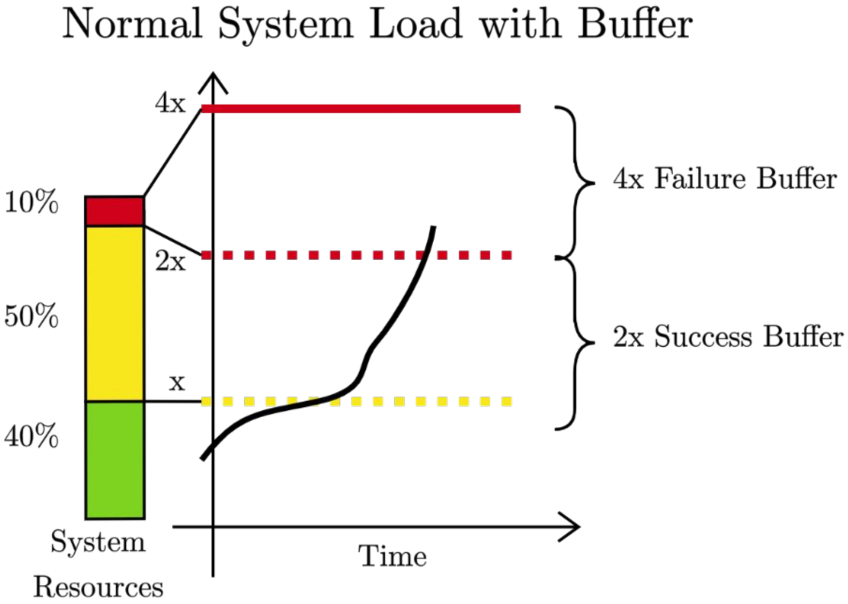
CPU Load Shedding

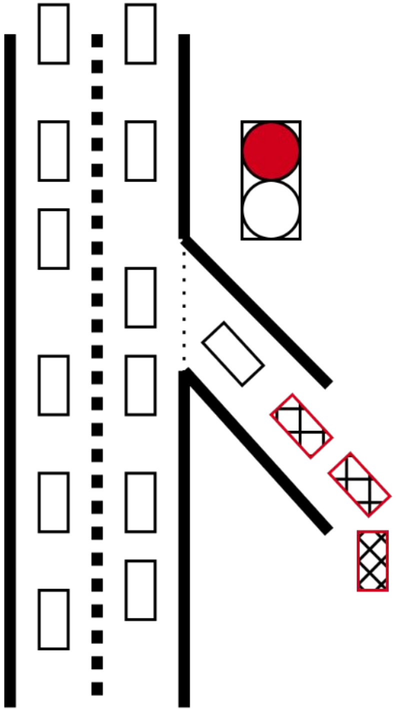




$$T_{transit} = 15m$$

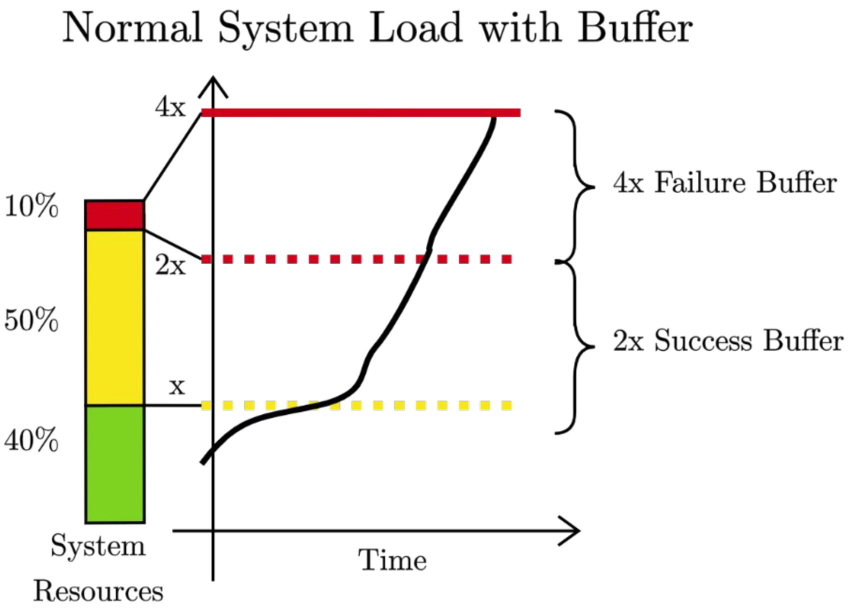
CPU Load Shedding

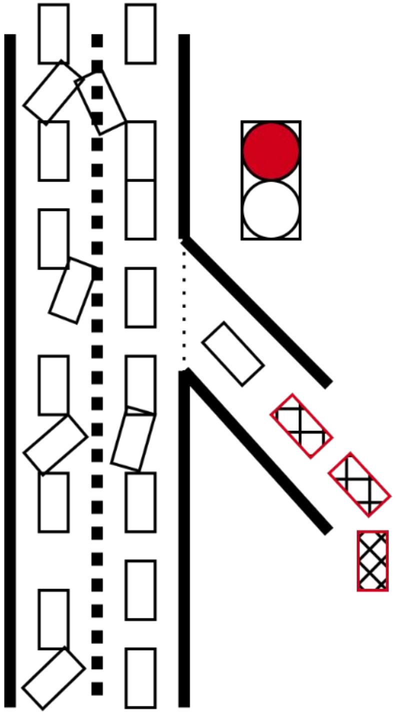




$$T_{transit} = 30m$$

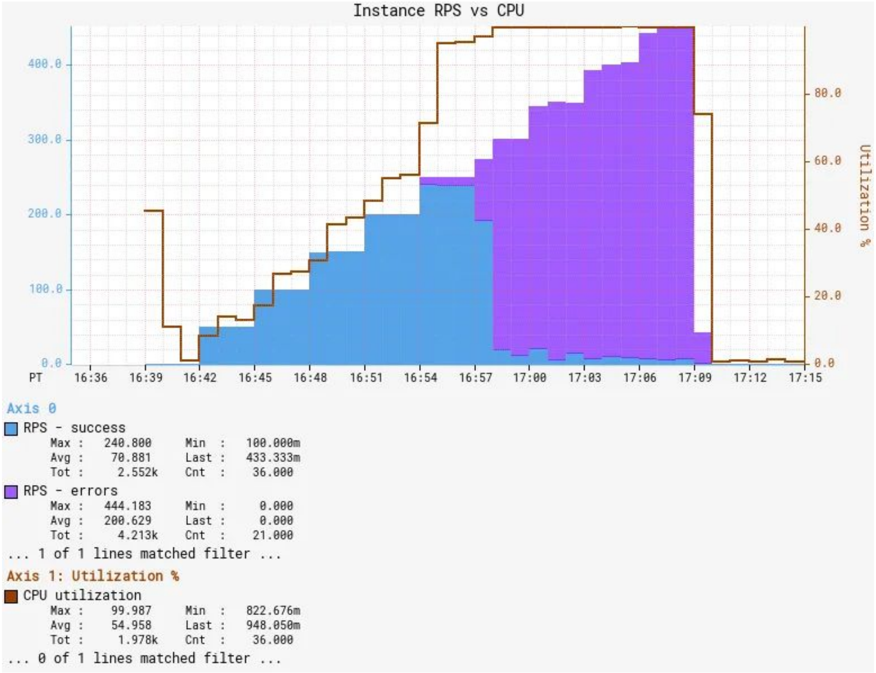
CPU Load Shedding





$$T_{transit} = \infty$$

Congestive Failure is Bad



Congestive Failure, a.k.a

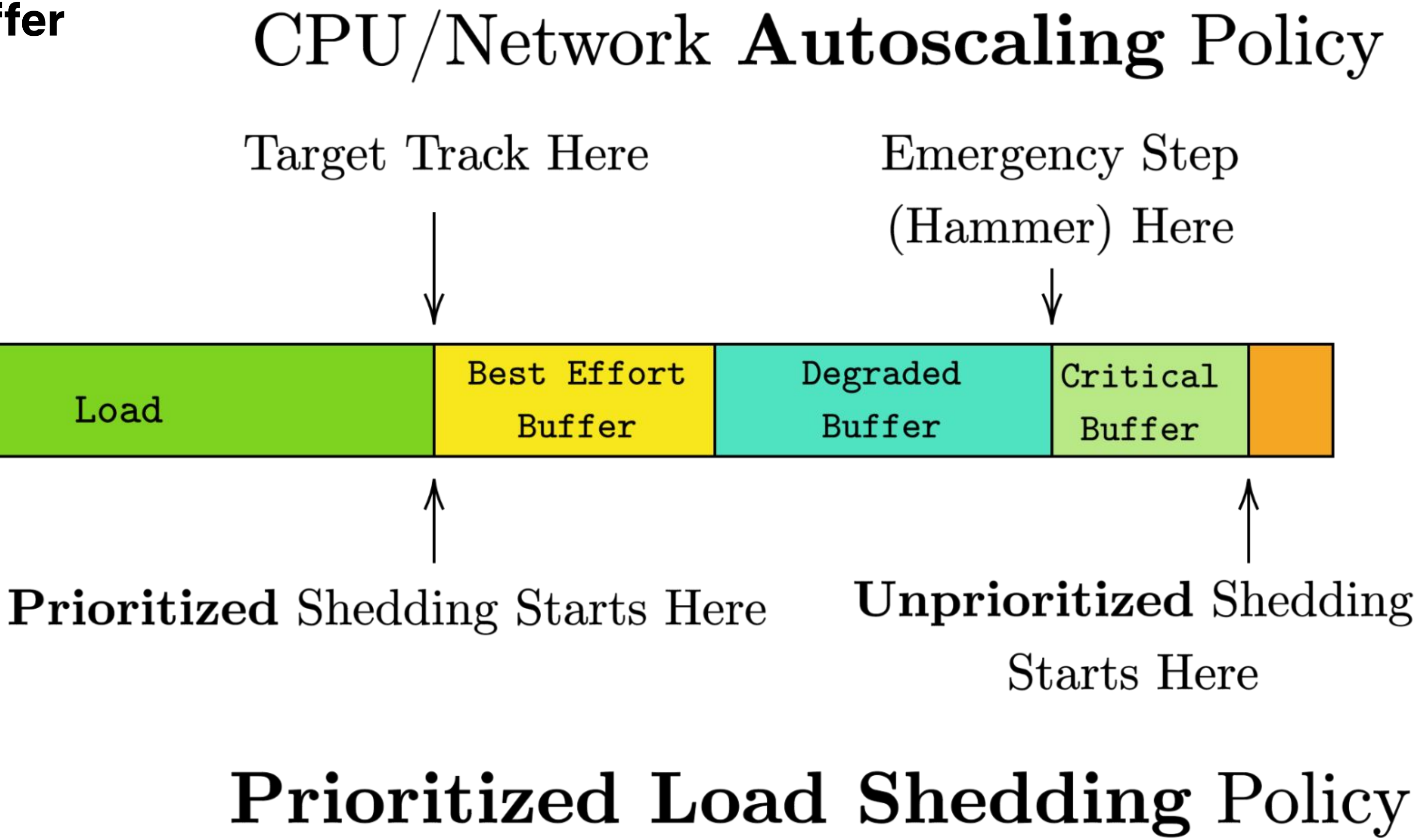
"Queueing Without Bound"

"Falling Over"

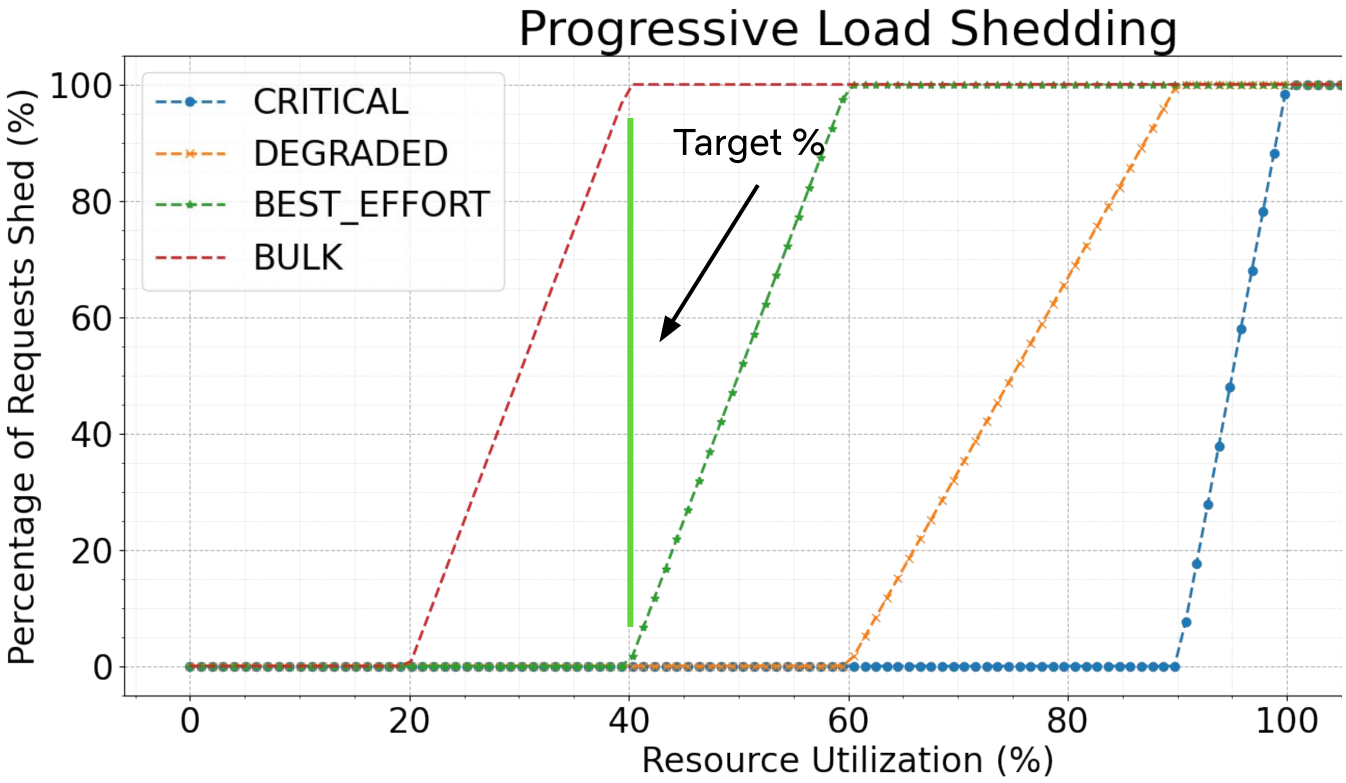
"Total System Failure"

"A real bad time™"

Prioritize Your Success Buffer



Prioritization Creates More Buffer



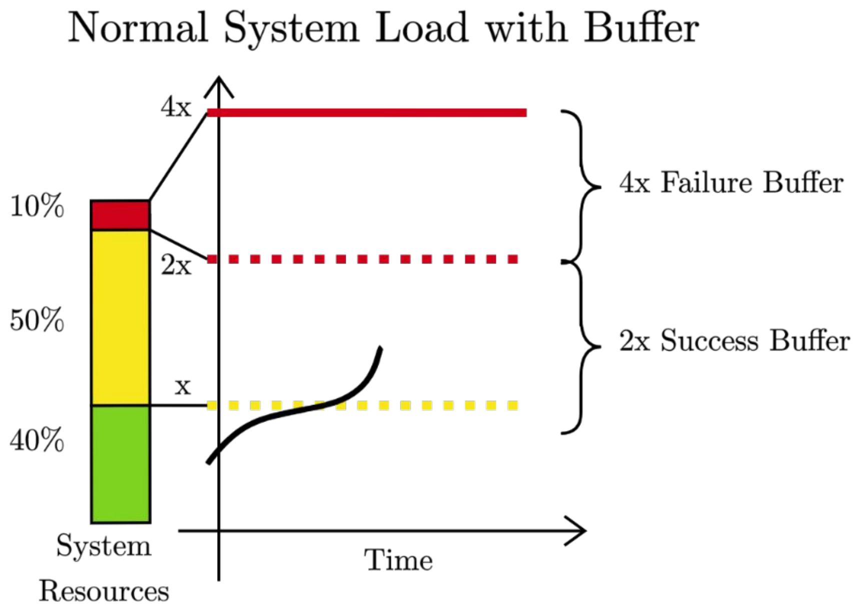
Netflix Tech Blog: Anirudh Mendiratta, Kevin Wang, Joey Lynch, et al. on **Service-Level Prioritized Load Shedding**

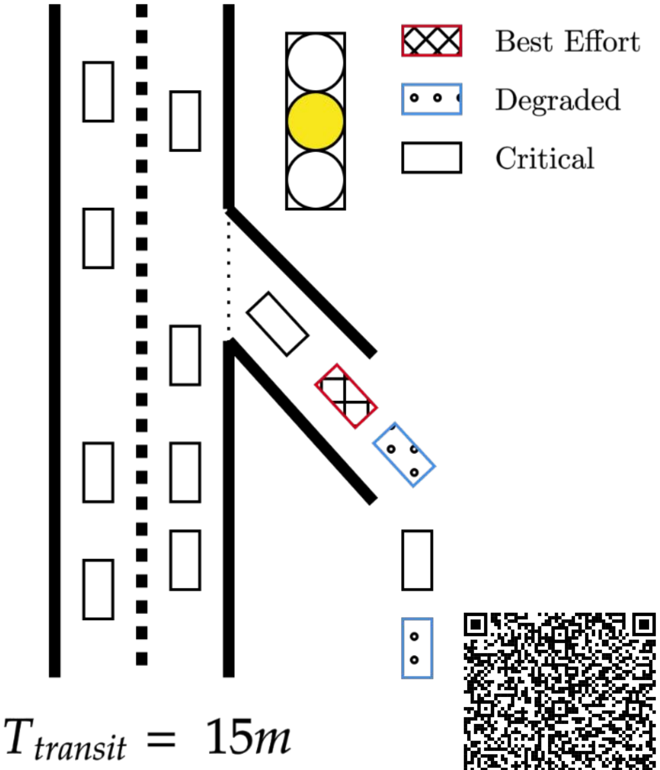
Prioritization Creates More Buffer

```
return context -> {
  Request req = context.getRequest();

  // Prioritize a particular path
  if (req.getPath().startsWith("/critical-play-url")) {
    return PriorityBucket.CRITICAL;
  }

  // Deprioritize background requests
  if (req.getParams().contains("background")) {
    return PriorityBucket.DEGRADED;
  }

  // Take the client device priority
  return getClientPriority(context.getHeaders());
}
```

Netflix Tech Blog: Anirudh Mendiratta, Kevin Wang, Joey Lynch, et al. on **Service-Level Prioritized Load Shedding**

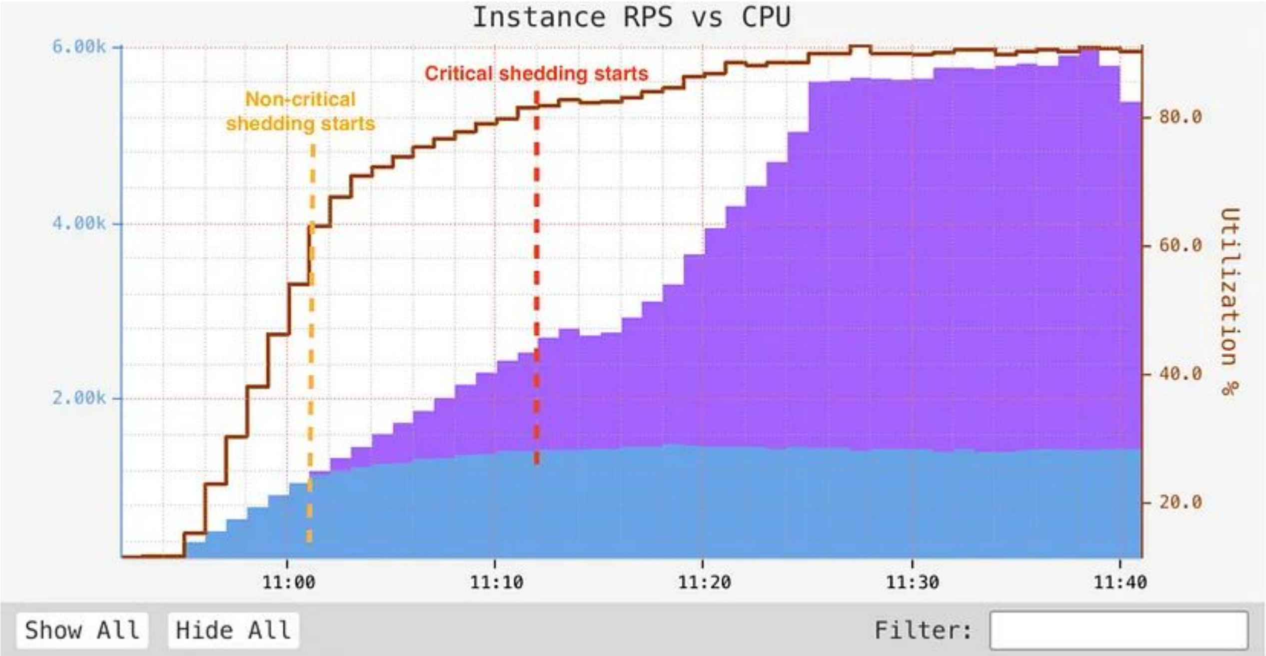
Prioritization Creates More Buffer





Netflix Tech Blog: Anirudh Mendiratta, Kevin Wang, Joey Lynch, et al. on **Service-Level Prioritized Load Shedding**

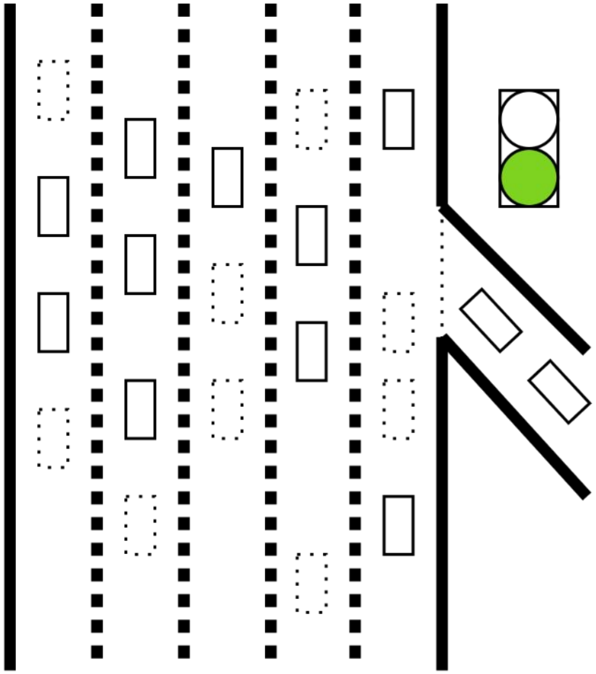
Prioritization Creates More Buffer



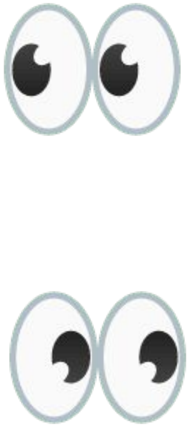
Min	Max	Avg	Total	Name	Visibility
201	1.48k	1.24k	60.7k	RPS – success	
375m	4.59k	2.49k	102k	RPS – errors	
11.5	91.0	73.5	3.60k	CPU utilization	

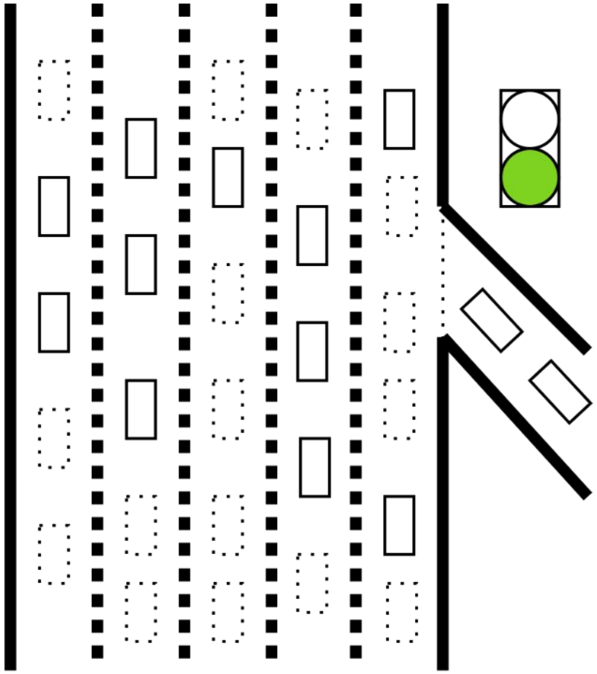
IO Load Shedding

How do we shed for IO workloads?



$T_{transit} = 15m$



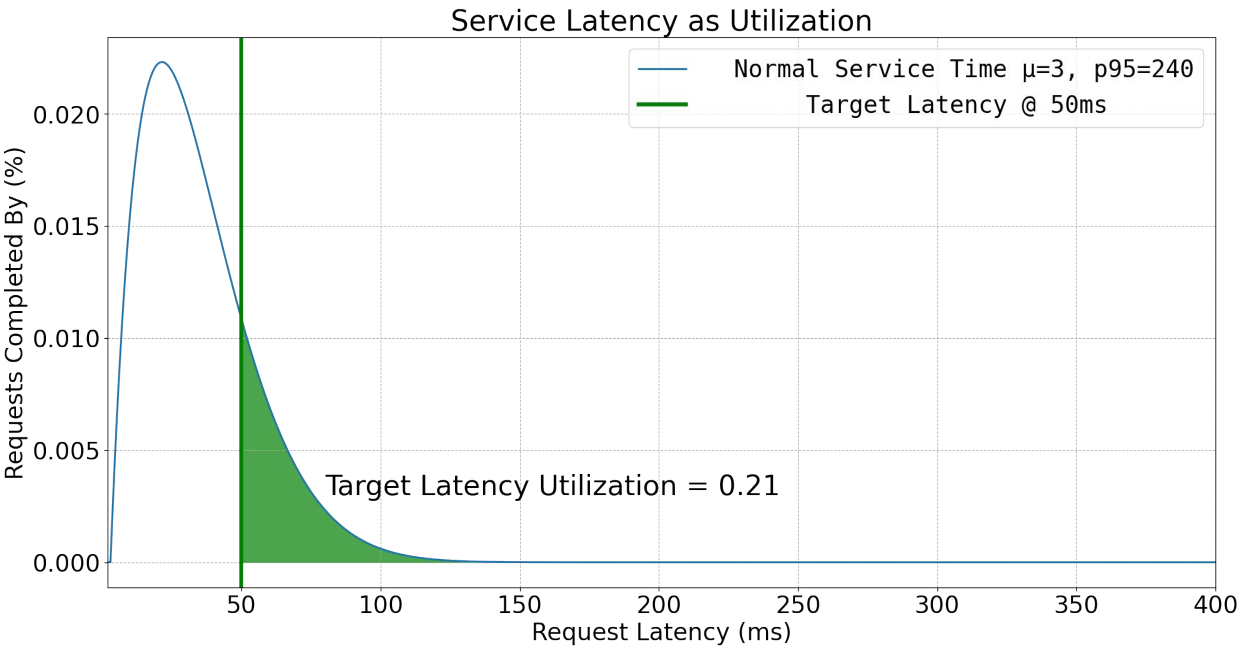


$T_{transit} = 45m$

IO Load Shedding

Use Latency Service-Level-Objective Utilization as a Proxy!

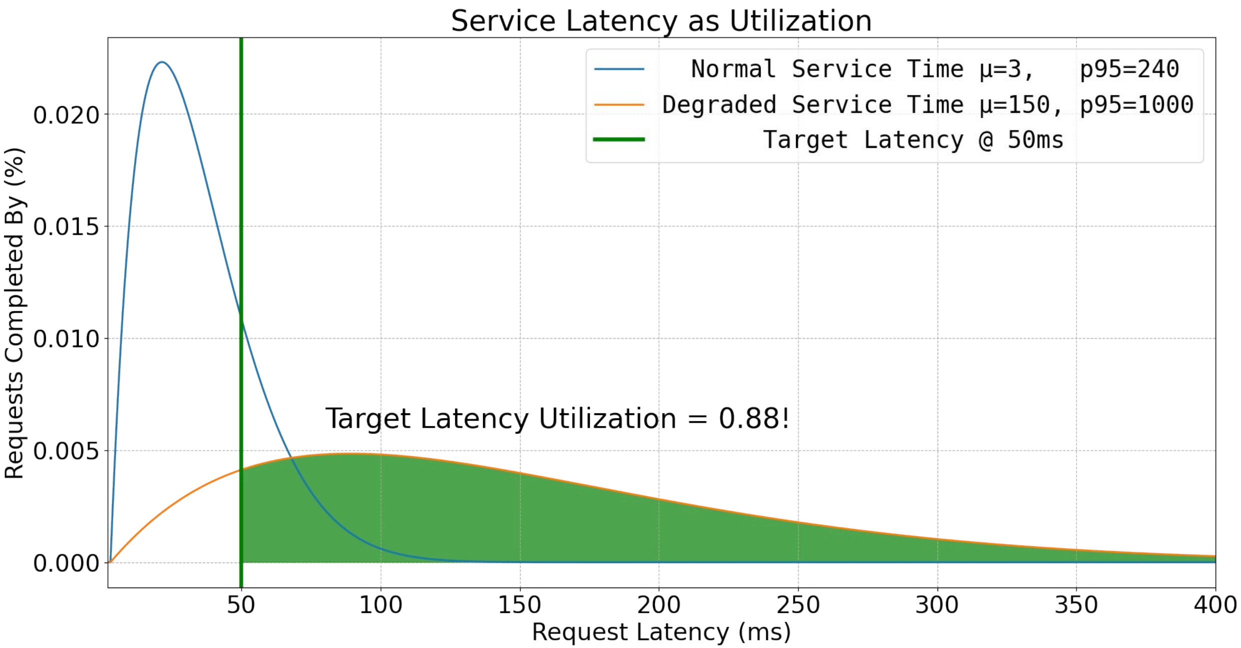
```
utilization(namespace) = {
  overall = 21
  latency = {
    slo_target = 21,
    slo_max = 0
  }
  system = {
    storage = 17,
    compute = 10,
  }
}
```



IO Load Shedding

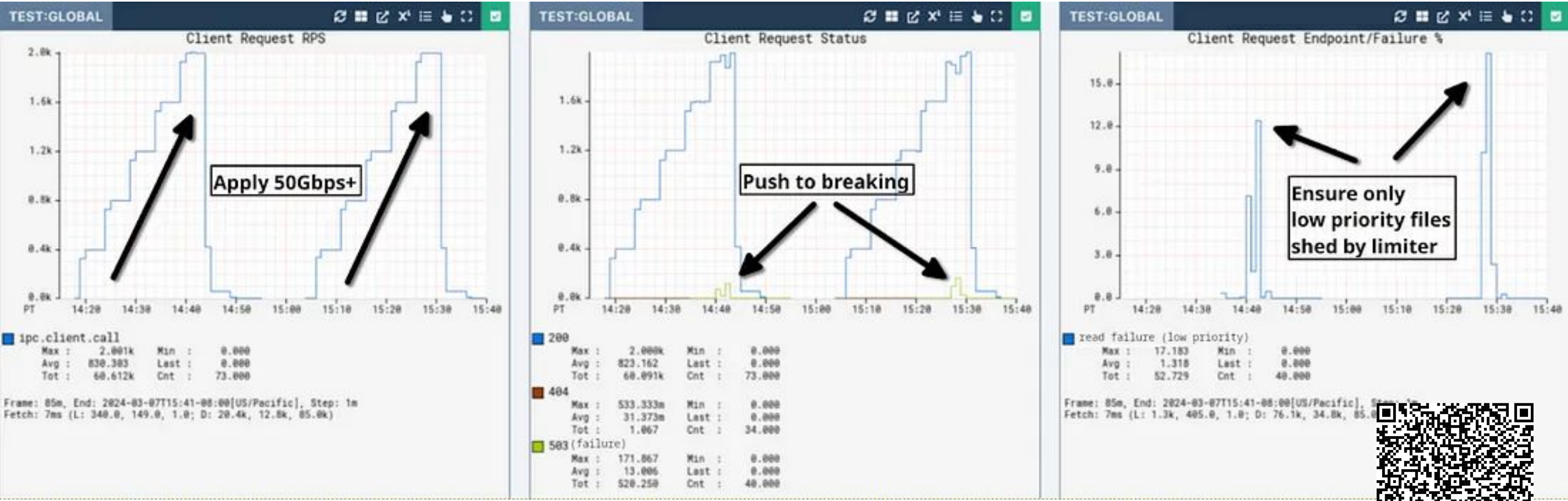
Use Latency Service-Level-Objective Utilization as a Proxy!

```
utilization(namespace) = {
  overall = 88
  latency = {
    slo_target = 88,
    slo_max = 5
  }
  system = {
    storage = 17,
    compute = 10,
  }
}
```



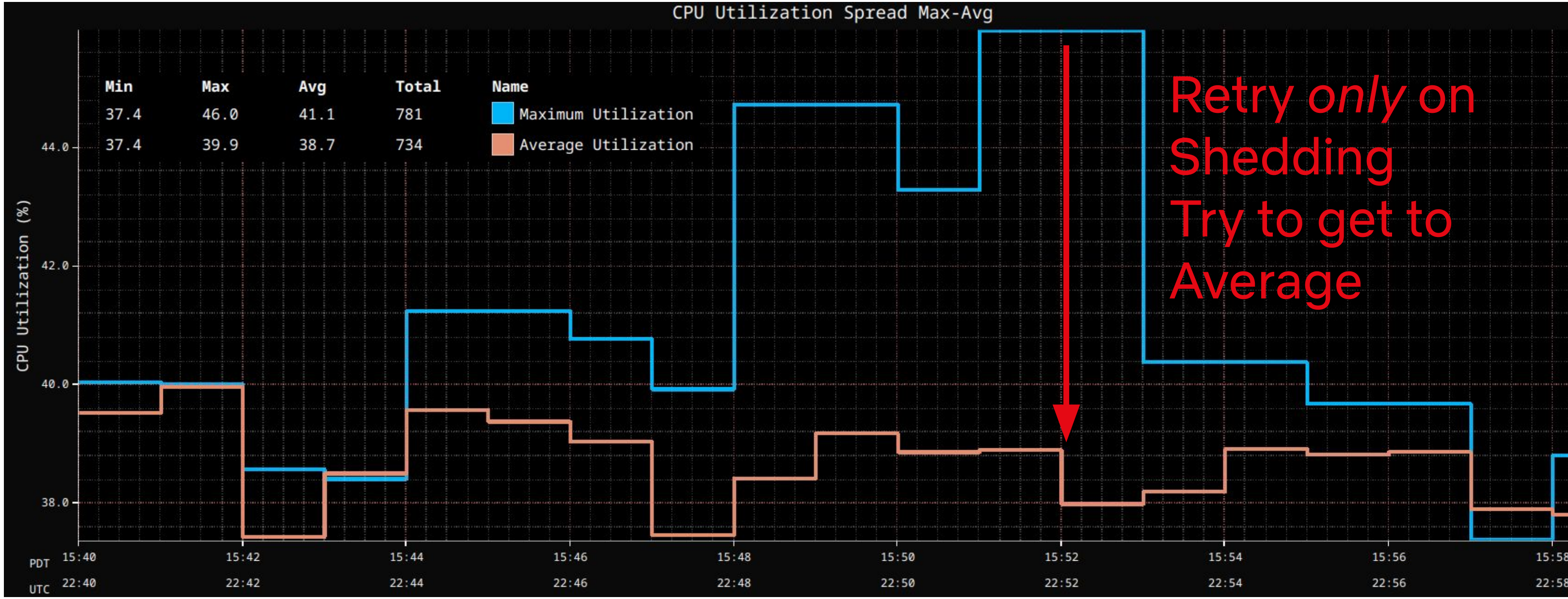
IO Load Shedding

Use Latency Service-Level-Objective Utilization as a Proxy!



Netflix Tech Blog: Anirudh Mendiratta, Kevin Wang, Joey Lynch, et al. on **Service-Level Prioritized Load Shedding**

Retries?

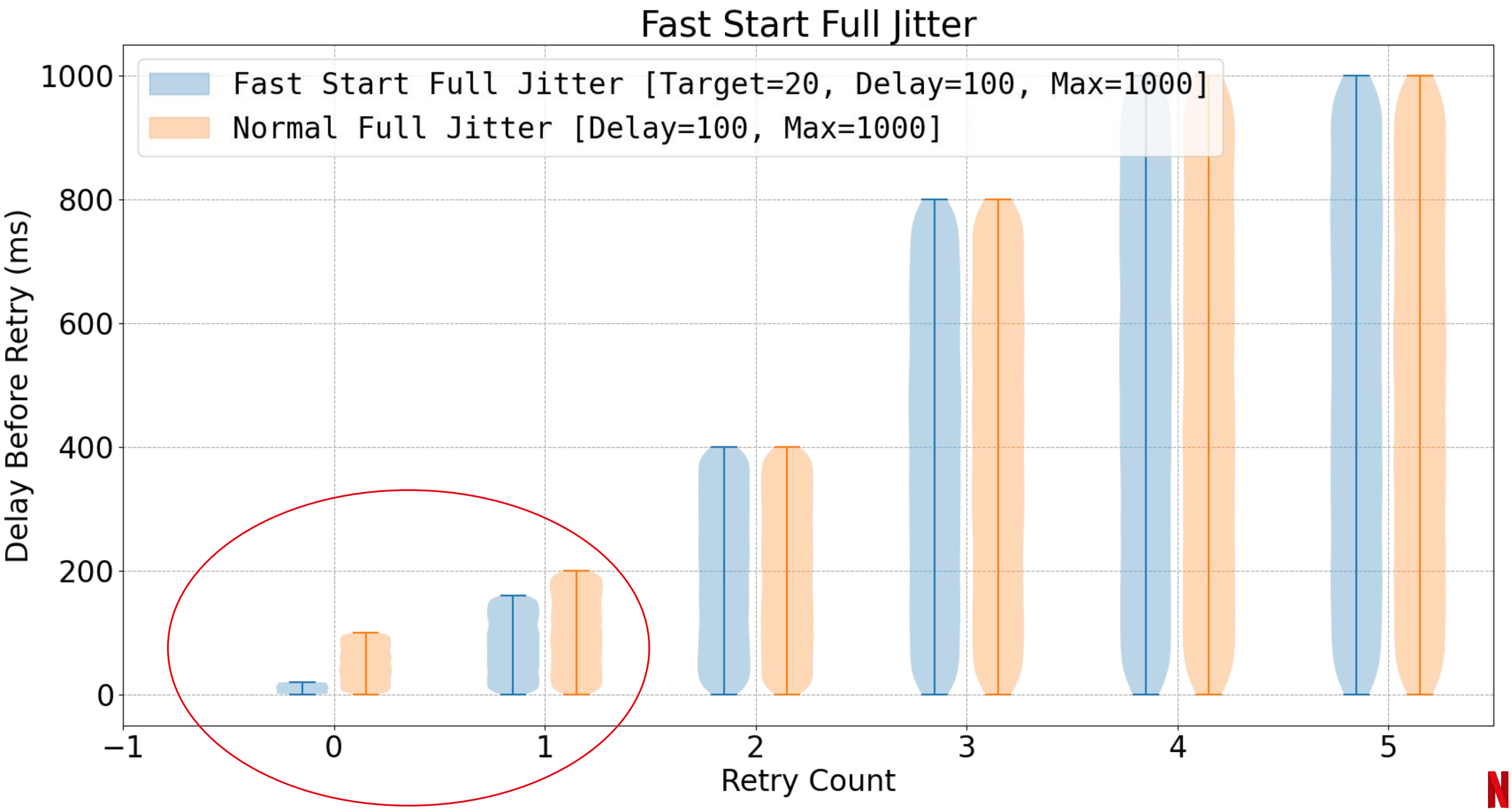


$$let\ R = \text{retry}\,\# \in [0,\ 1,\ 2,\ \ldots\ \text{retry}_{max} - 1]$$

$$\text{base}(R) = \min\left(\text{delay},\ \text{target} \times (R + 1)^2\right)$$

$$\text{retry}(R) = \text{rand}\left[0,\ \min\left\{\text{delay}_{\max},\ \text{base}(R) \times 2^R\right\}\right)$$

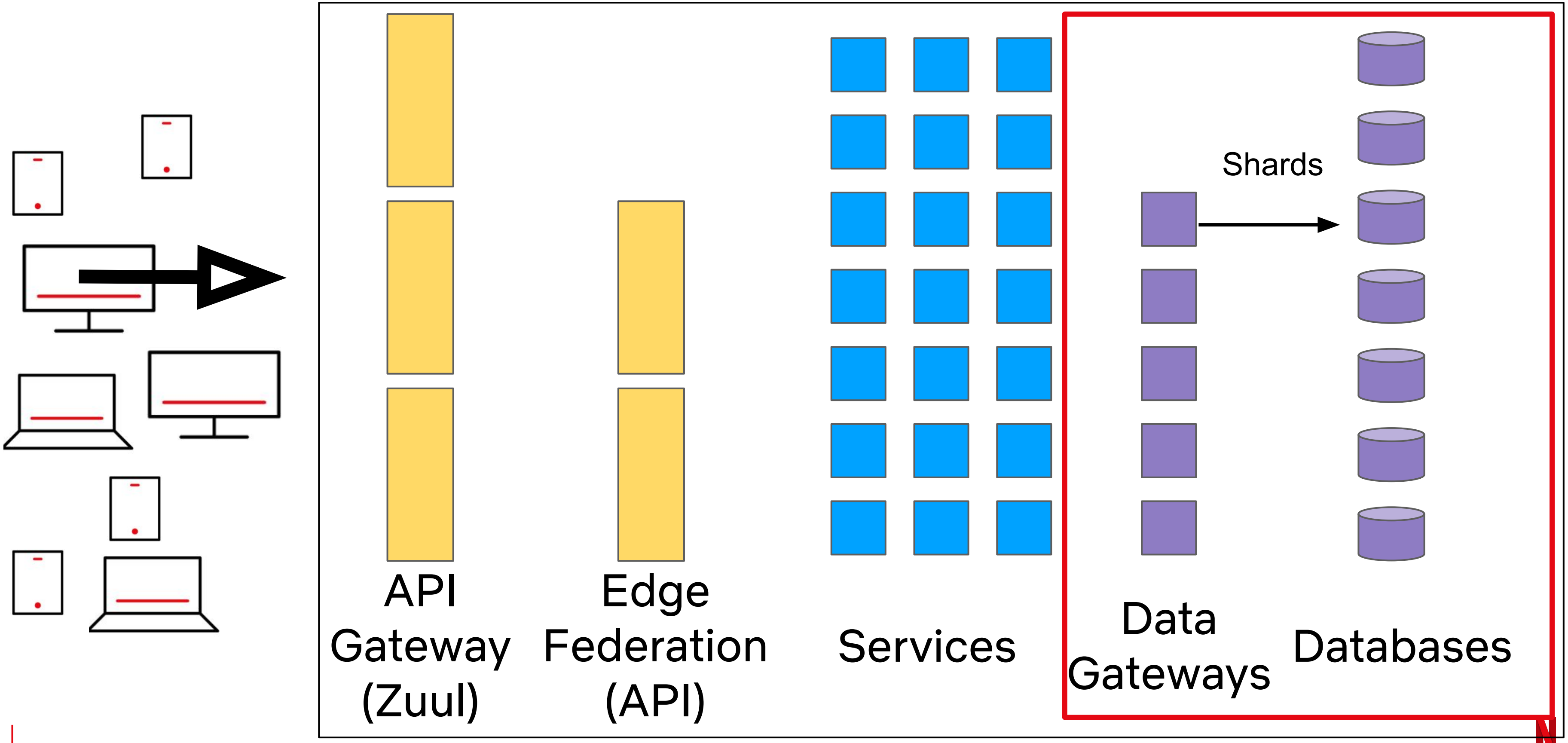
Retries?


Fast Start Full Jitter
Fast Start Full Jitter [Target=20, Delay=100, Max=1000]
Normal Full Jitter [Delay=100, Max=1000]
Delay Before Retry (ms)
1000
800
600
400
200
0
−1
0
1
2
3
4
5
Retry Count

Stateful Resilience

Capacity Planning
Data Gateways
Caching Wisely

Microservices



Capacity Plan
Stateful

Strategy: Buffer, and lots of math

```python
from service_capacity_modeling.capacity_planner import planner
from service_capacity_modeling.models.org import netflix

# Load up the Netflix capacity models
planner.register_group(netflix.models)

# Plan a cluster
plan = planner.plan(
    model_name="org.netflix.cassandra",
    region="us-east-1",
    desires=desires,
    simulations=1024,
    explain=True
)
```

```
Least Regret Choice:
---------------------
 12 m5d.xlarge costing 8973.94

All Choices
-----------
{'  6  r5.xlarge': 4,
 '  6  r5d.large': 31,
 '  6 m5.2xlarge': 2,
 '  6 m5d.xlarge': 224,
 ' 12  m5.xlarge': 132,
 ' 12 m5d.xlarge': 277,
 ' 24  m5.xlarge': 242,
 ' 24 m5d.xlarge': 54,
 ' 48  m5.xlarge': 55,
 ' 48 m5d.xlarge': 2,
 ' 96  m5.xlarge': 1}
```

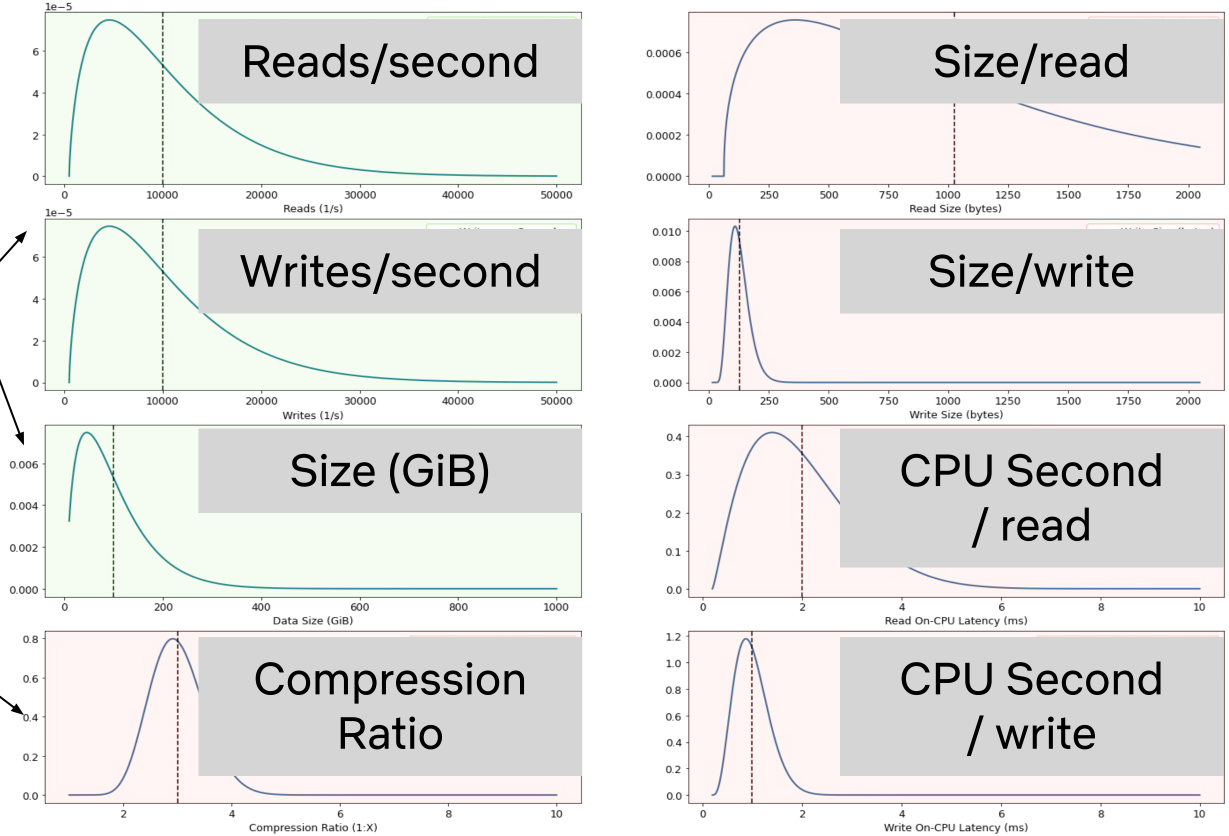
AWS Re:Invent 2022: Joseph Lynch on **Capacity Plan Optimally**

Capacity Plan
Stateful

Optimally buy computers for each workload.

From the Human

From the Model



AWS Re:Invent 2022: Joseph Lynch on **Capacity Plan Optimally**

Gateways Unlock Resilience

 # chunked writes

$x[1] \leftarrow d_1\{\text{size} = 1\text{MiB}, \ \text{token} = t_1\}$

$x[2] \leftarrow d_2\{\text{size} = 1\text{MiB}, \ \text{token} = t_1\}$

$x \leftarrow \text{Commit}\{\text{items} = [d_1, \ d_2]\} \ @ \ t_1$

 # paginated reads

$\text{get}(x) \longrightarrow \{\text{items} = [d_1], \ \text{next} = 2\}$

$\text{get}(x, \ \text{prev} = 2) \longrightarrow \{\text{items} = [d_2], \ \text{next} = \varnothing\}$

$x = d_1 + d_2$

Incremental

Gateways Unlock Resilience

chunked writes

$x[1] \leftarrow d_1\{\mathrm{size} = 1\mathrm{MiB},\ \mathrm{token} = t_1\}$

$x[2] \leftarrow d_2\{\mathrm{size} = 1\mathrm{MiB},\ \mathrm{token} = t_1\}$

$x \leftarrow \mathrm{Commit}\{\mathrm{items} = [d_1,\ d_2]\}\ @\ t_1$

paginated reads

$\mathrm{get}(x) \longrightarrow \{\mathrm{items} = [d_1],\ \mathrm{next} = 2\}$

$\mathrm{get}(x,\ \mathrm{prev} = 2) \longrightarrow \{\mathrm{items} = [d_2],\ \mathrm{next} = \varnothing\}$

$x = d_1 + d_2$

write

$x \leftarrow 1\ @\ t_1$

$\mathrm{get}(x) \rightarrow 1$

update

$x \leftarrow 2\ @\ t_2$

hedge

$x \leftarrow 1\ @\ t_1$

update visible

$\mathrm{get}(x) \rightarrow 2$

Incremental

Idempotent

QCon 2024: Joey Lynch on **Highly-Reliable Online Stateful Systems**

Gateways Unlock Resilience

 # chunked writes

$x[1] \leftarrow d_1\{size = 1MiB,\ token = t_1\}$

$x[2] \leftarrow d_2\{size = 1MiB,\ token = t_1\}$

$x \leftarrow Commit\{items = [d_1,\ d_2]\} @ t_1$

 # paginated reads

$get(x) \rightarrow \{items = [d_1],\ next = 2\}$

$get(x,\ prev = 2) \rightarrow \{items = [d_2],\ next = \varnothing\}$

$x = d_1 + d_2$

 # write

$x \leftarrow 1 @ t_1$

$get(x) \rightarrow 1$

 # update

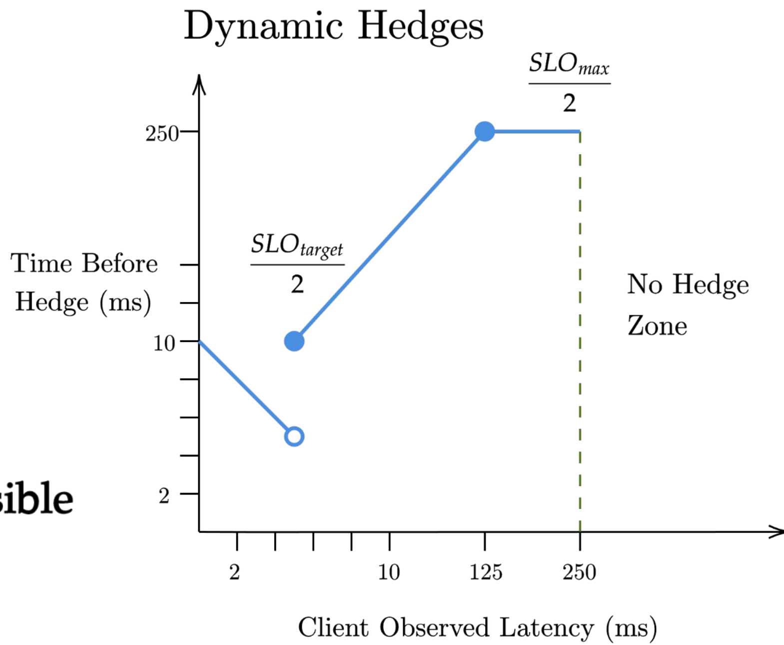
$x \leftarrow 2 @ t_2$

***hedge***

$x \leftarrow 1 @ t_1$

 # update visible

$get(x) \rightarrow 2$



Incremental **Idempotent** **Retriable**

QCon 2024: Joey Lynch on **Highly-Reliable Online Stateful Systems**

Load Shedding and Retries

Prefer limiting on server, clients implement basic safety

Load Shedding and Retries

Prefer limiting on server, clients implement basic safety

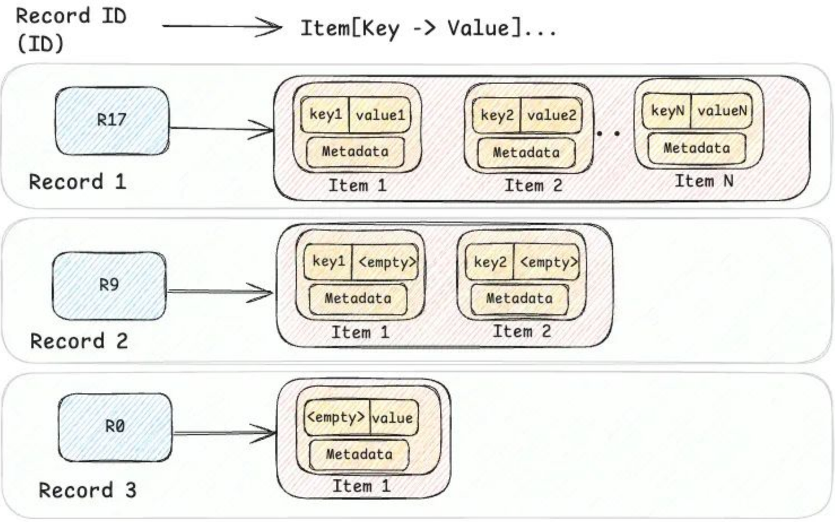
Load Shedding and Retries

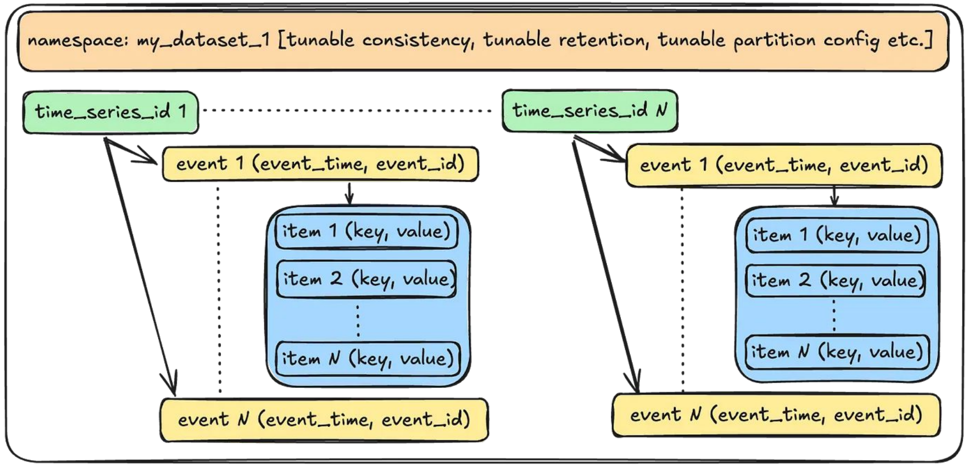
Prefer limiting on server, clients implement basic safety

Load Shedding and Retries

Prefer limiting on server, clients implement basic safety

Key-Value



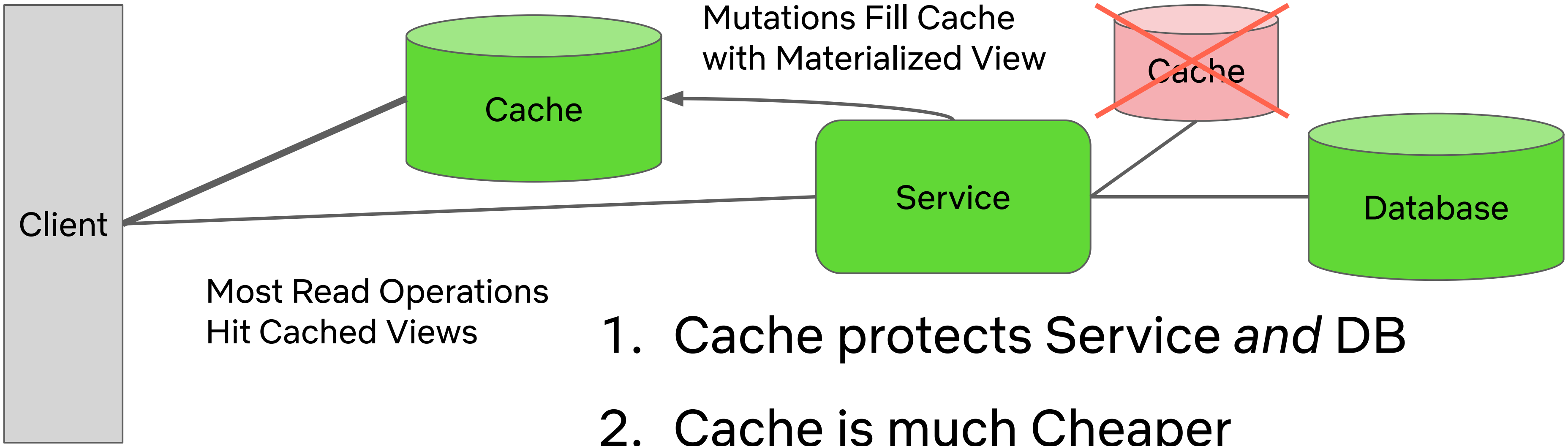


Time-Series

Netflix Tech Blog: Vidhya Arvind, Rajasekhar Ummadisetty, Rajiv Shringi, Joseph Lynch, et. al. on **Data Abstractions**

Cache In Front of Services

Cache your service, not your database!

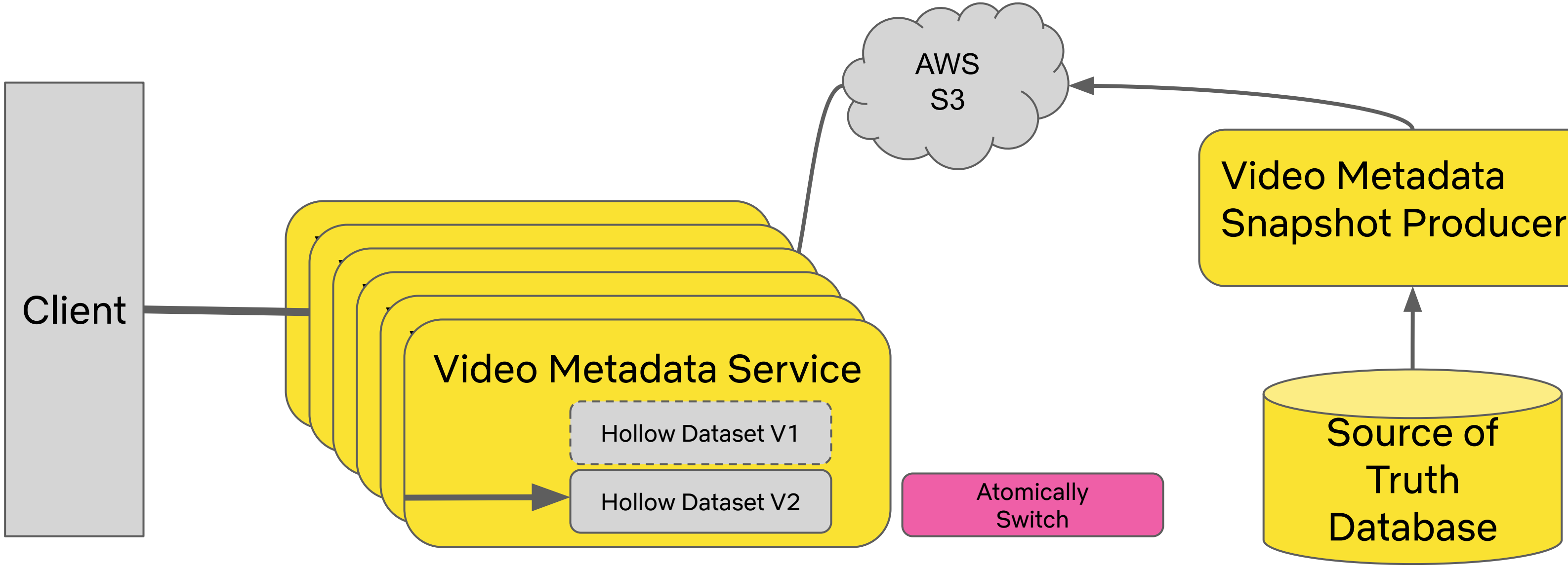


1. Cache protects Service *and* DB
2. Cache is much Cheaper
3. Cache can be Replicated

QCon 2024: Prudhviraj Karumanchi, Sriram Rangarajan on **Global Caching**

Total Cache
What You Can

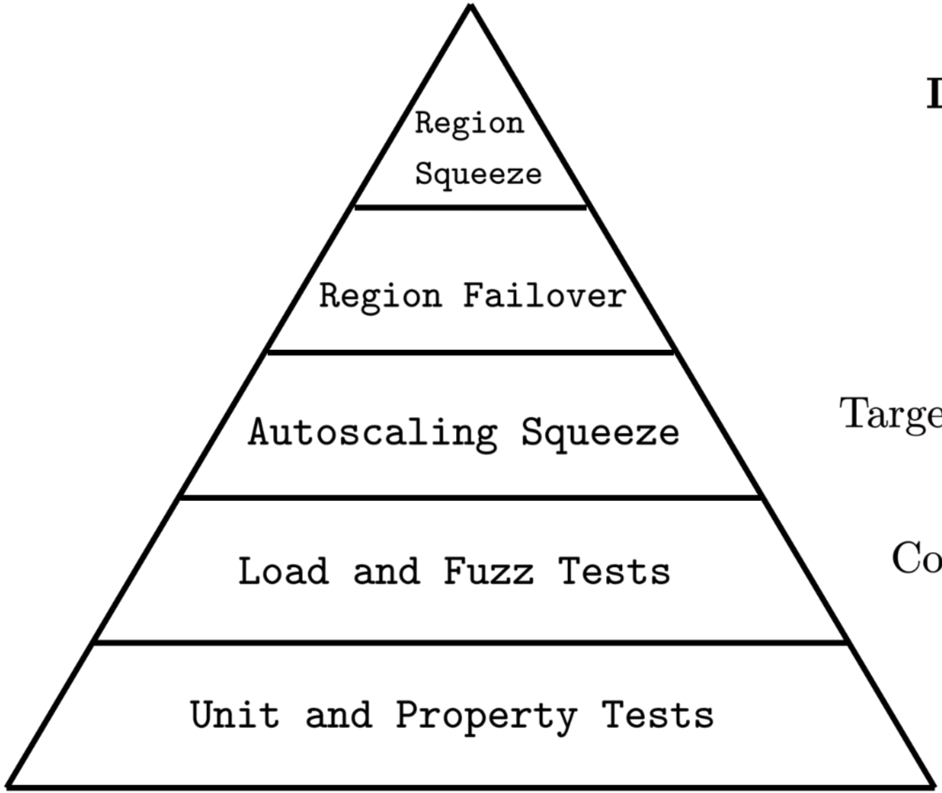
Replicated total near caches like Netflix Hollow handle Load increases easily!



Netflix Tech Blog: Ammar Khaku, Drew Koszewnik on **Total Near Caches**

Test Constantly

Properties Tests
Load Tests
Failover Tests



De-risk the Unknown with Full Global Traffic

Full System **Integration Tests**, Verify Buffers

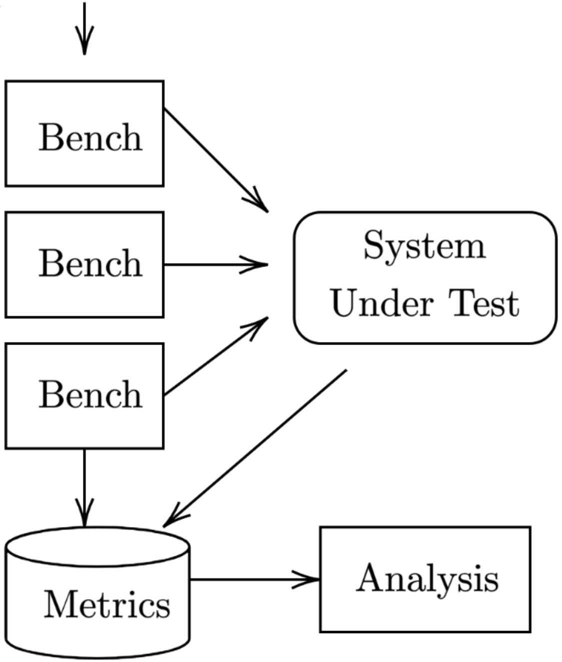
Targeted **Scaling Tests** with **Property Assertions**

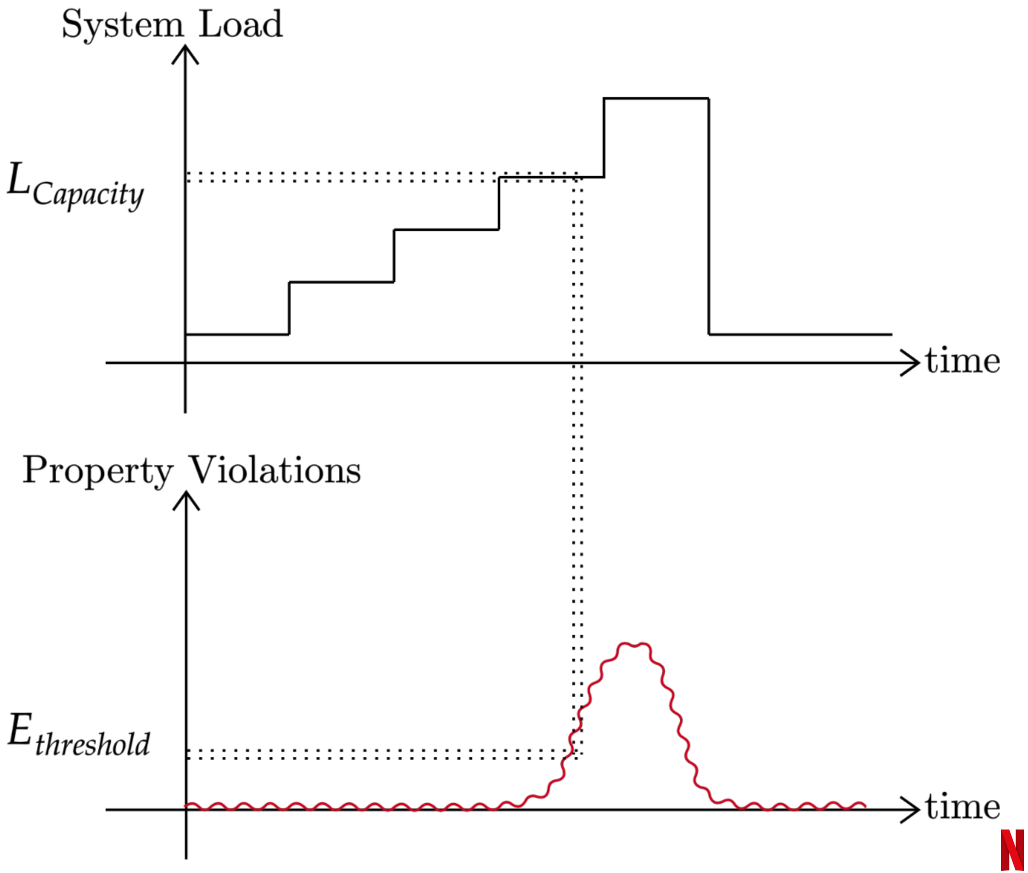
Controlled **Benchmarks** and **Randomized Fuzz**

Reliable **Unit** and **Randomized Properties**

Example Load Test

{
$\text{write}_{\text{rate}}$ = [0, 10k],
$\text{write}_{\text{bytes}}$ = [1, 4KiB], ...
}





Manage Traffic Demand

Full-Active Control of Demand
- Global **traffic shaping**
- Prioritization at the Edge
- **Fallbacks**!

Understand Traffic Flows
- Uneven Service Flows
- Different **Traffic Priorities**

Manage Quality of Service
- **Slow** often **better than Down**

Balance Compute Supply

Capacity Planning
- Compute and Workload Analysis
- Intentional **Buffers**
- **Pre-scale** when you can

Get Out of Trouble
- **Autoscale**
- **Shed Load**
- Stateful can be reliable too!

Manage Quality of Service
- **Prioritized** Shedding
- **Stale** often **better than "Down"**

Thank You.

Joseph Lynch
josephl@netflix.com
jolynch.github.io