How Netflix Shapes our Fleet for Efficiency and Reliability

Argha C and Joey Lynch

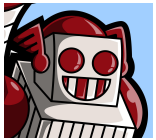
Speaker

Argha C

Staff Software Engineer
Infrastructure at Netflix

Focus on Compute and Network problems.

https://www.linkedin.com/in/argha-c/



Speaker

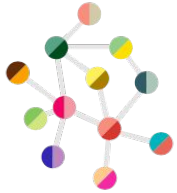
Joseph Lynch

Principal Software Engineer
Platform Engineering at Netflix

Database shepherd, compute optimizer, distributed system nerd

https://jolynch.github.io/






Agenda

Goal: Efficiency *and* Reliability

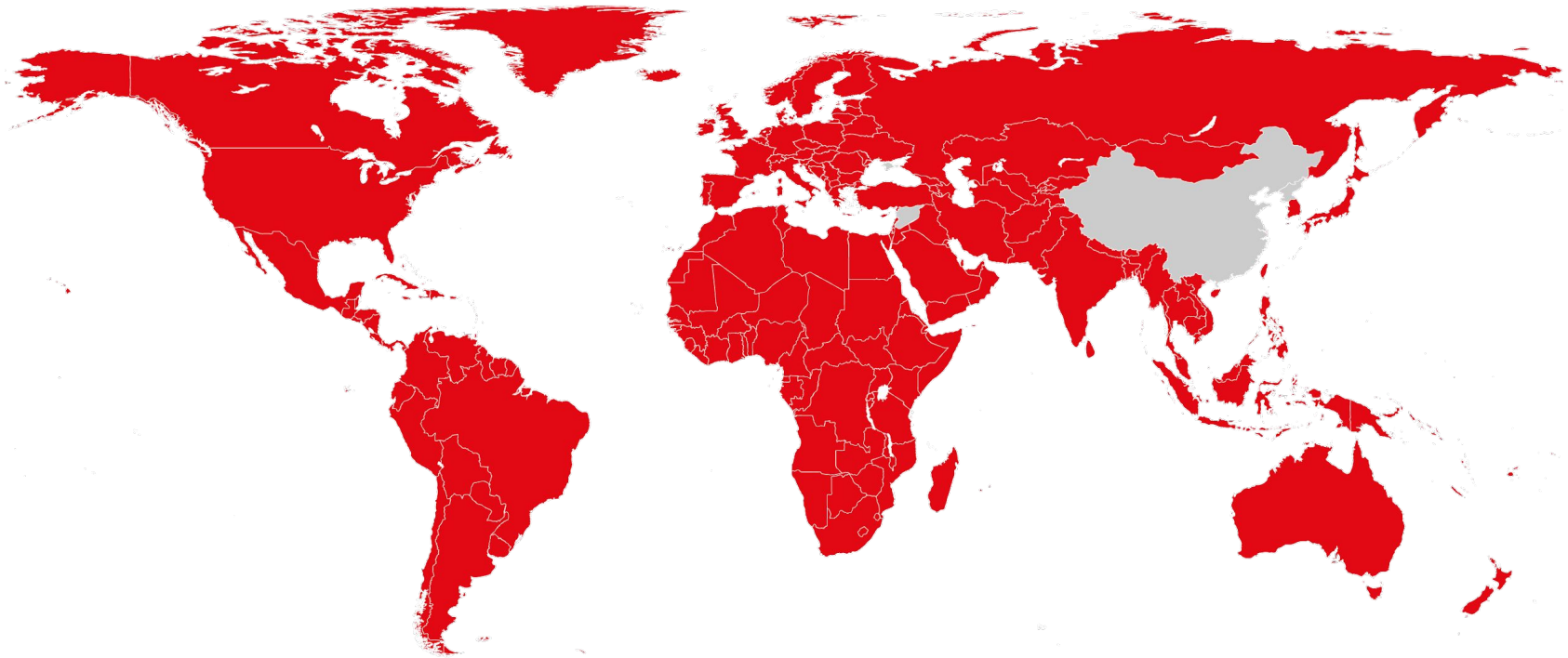
Compute **Supply**
— Capacity Planning → Fleet Planning

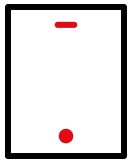
Workload **Demand**
— Workload Shape → Traffic Shaping

Balance Supply with Demand

Reliability Techniques **Allow Risk**

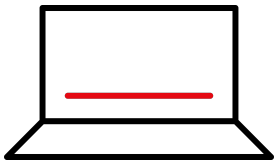
Problem - Global





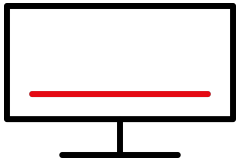
Mobile

Diverse Device Capabilities
Weaker Network
Android and IOs



PC

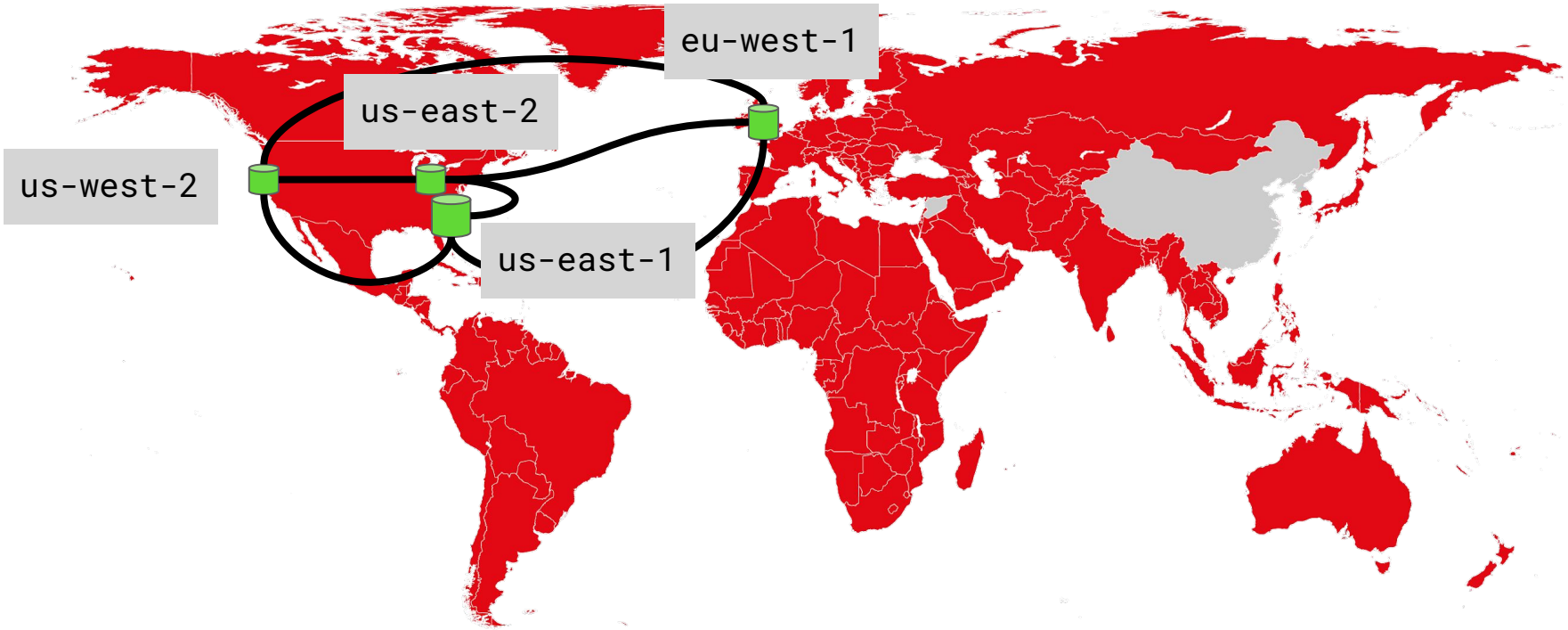
Diverse Device Capabilities
More Stable Network
Medium Resolution



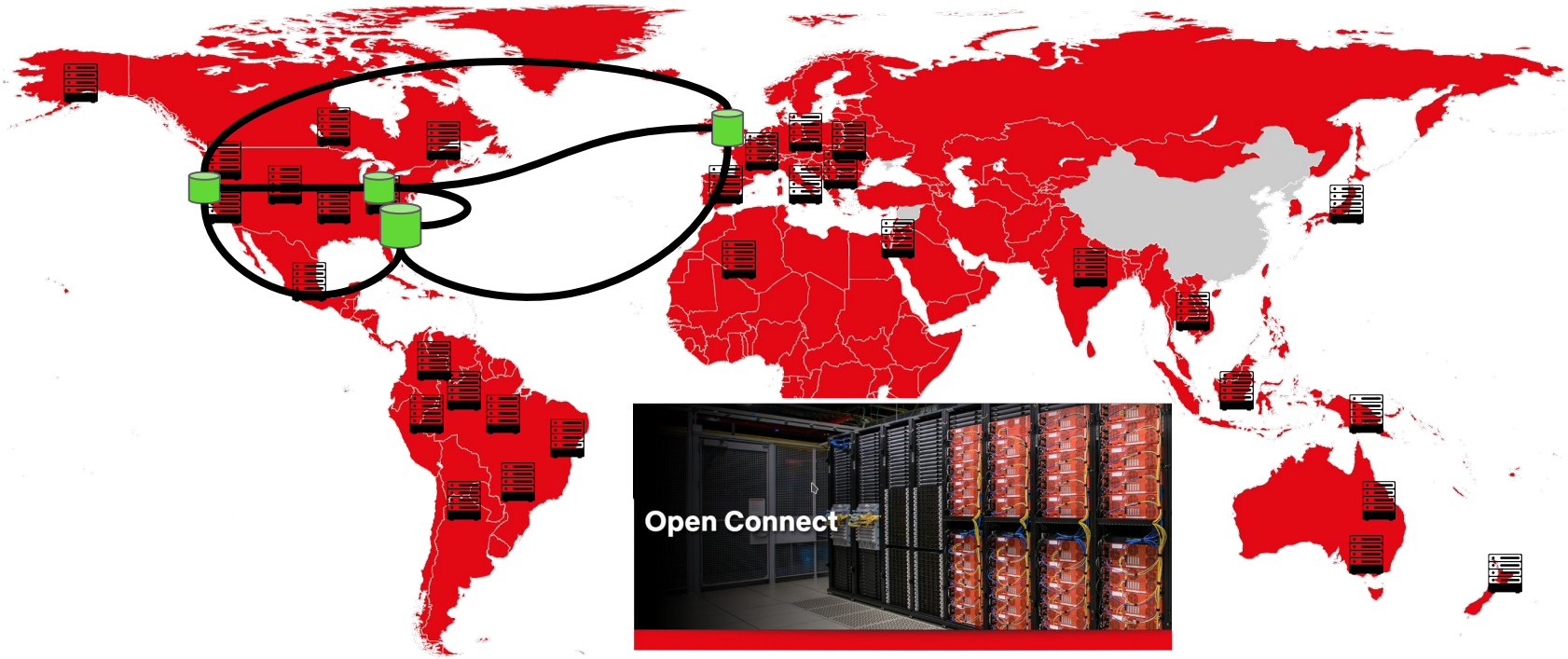
TV

Diverse Device Capabilities
More Stable Network
High Resolution

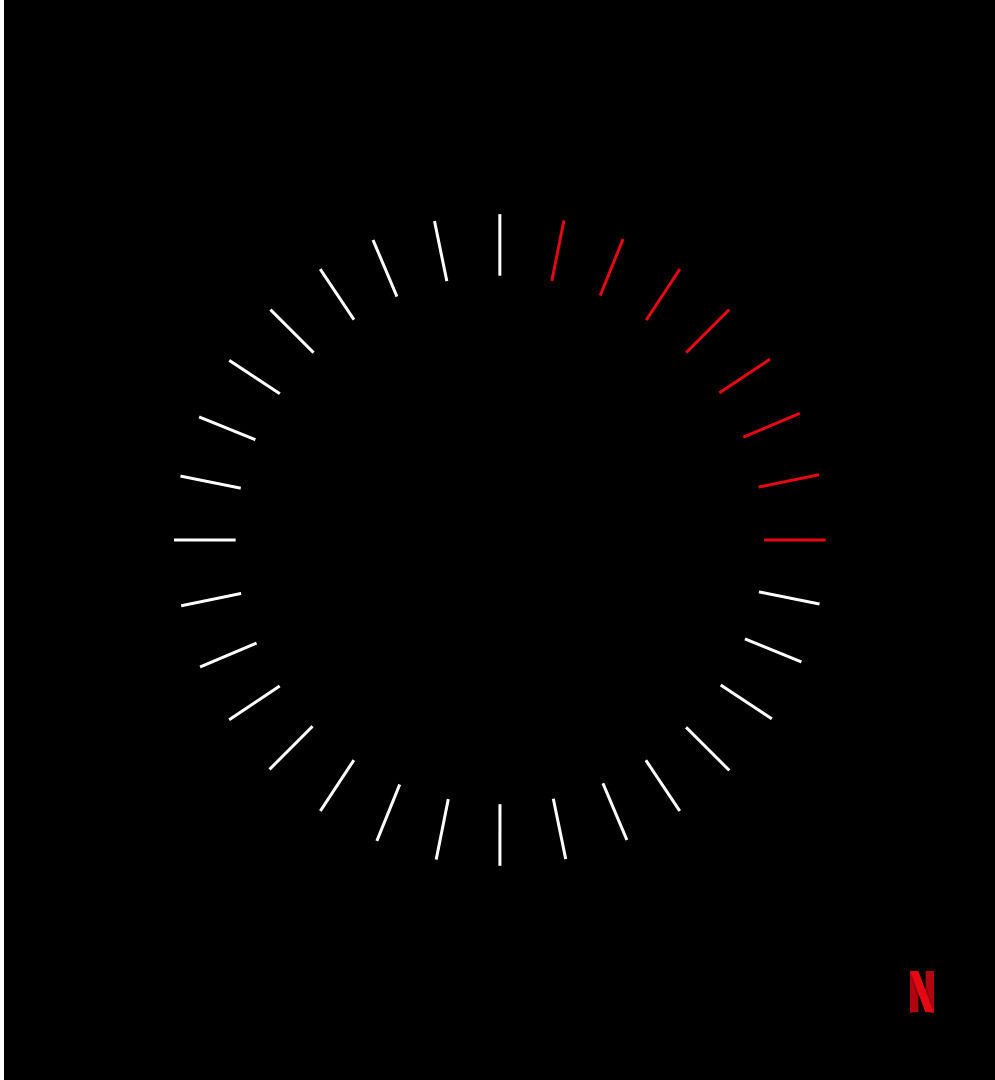
Solution - Global Control Plane



Solution - Global Data Plane



Efficiency *and* Reliability



Efficiency

Workloads create **Business Value**

Workloads create **Business Value**

Workloads have **Cost**

Efficiency

Workloads create **Business Value**

Workloads have **Cost**

Workload **Cost ∝ Resource Usage**

Efficiency

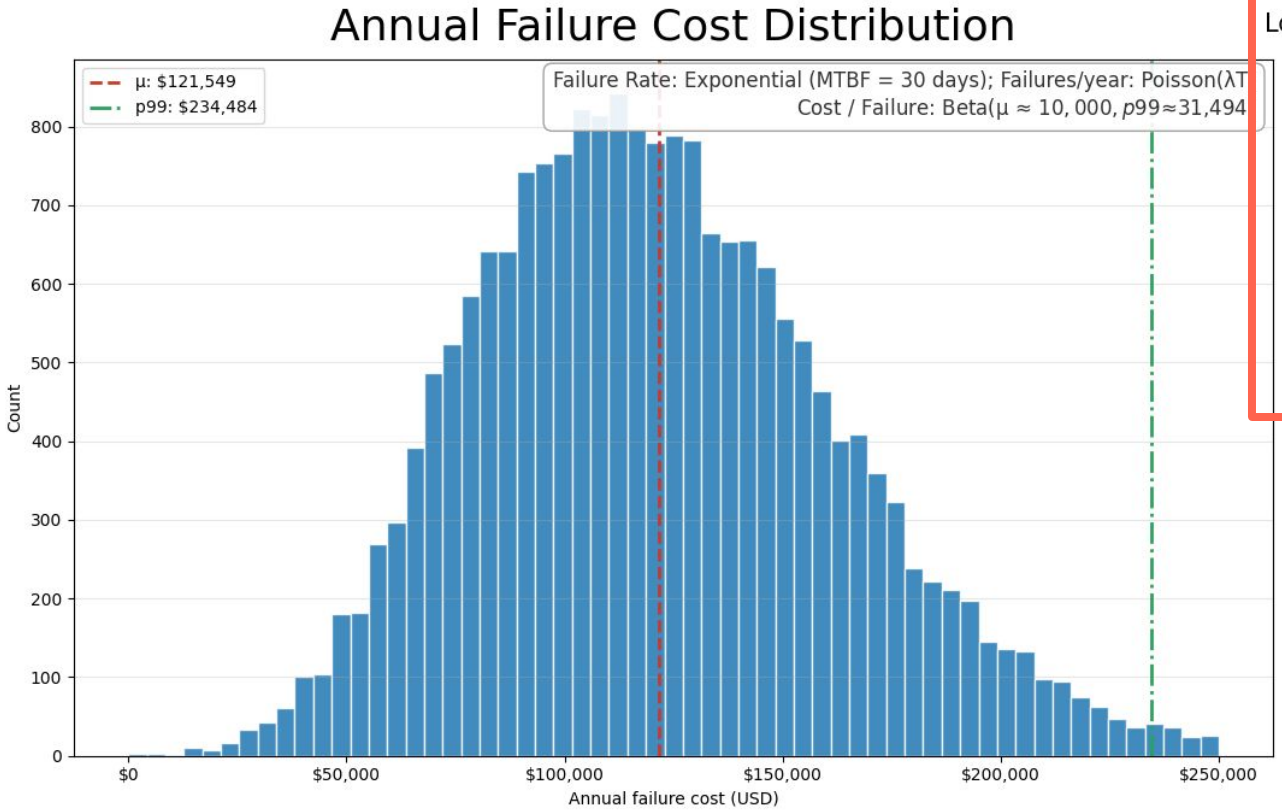
Workloads create **Business Value**

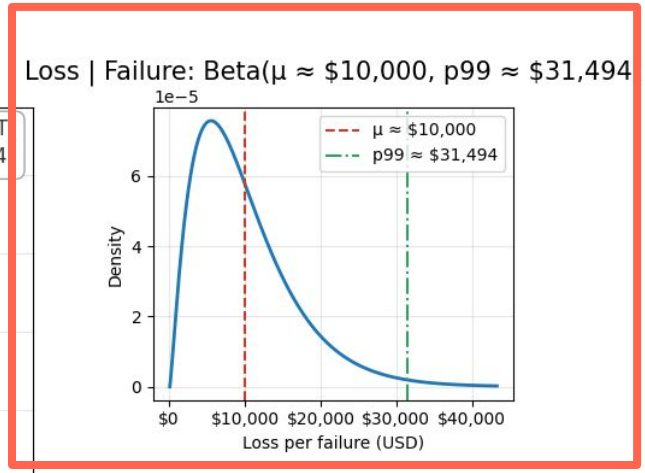
Workloads have **Cost**

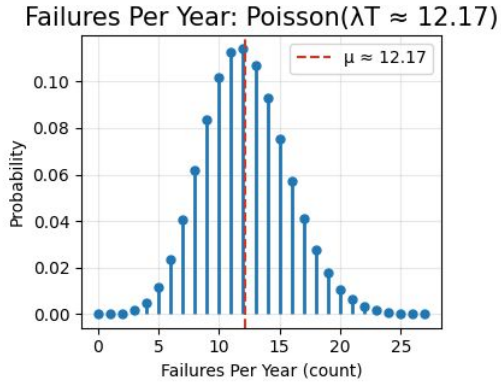
Workload **Cost ∝ Resource Usage**

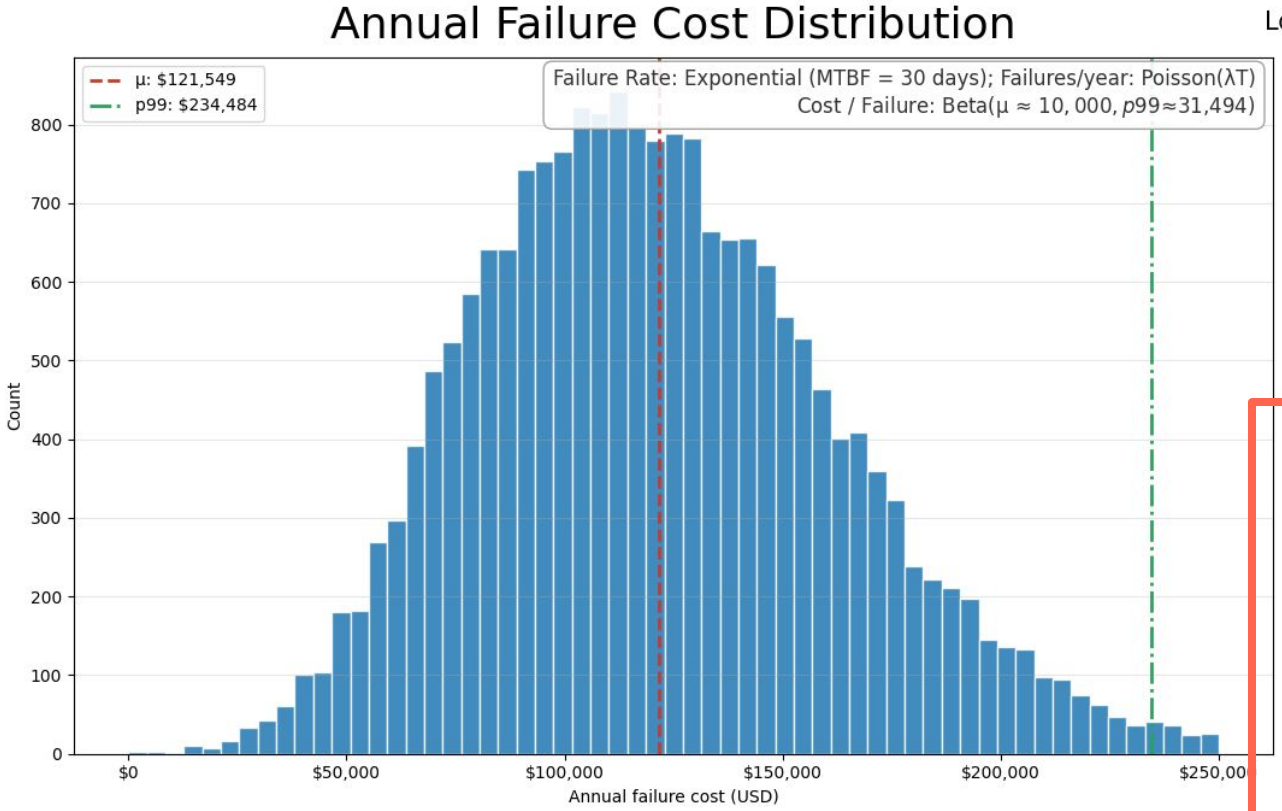
Workload **Failures have Cost**

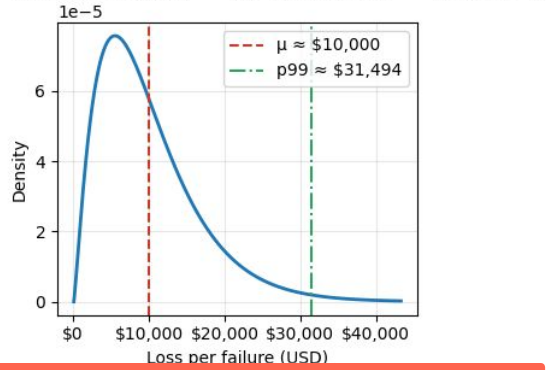
Failures have cost!

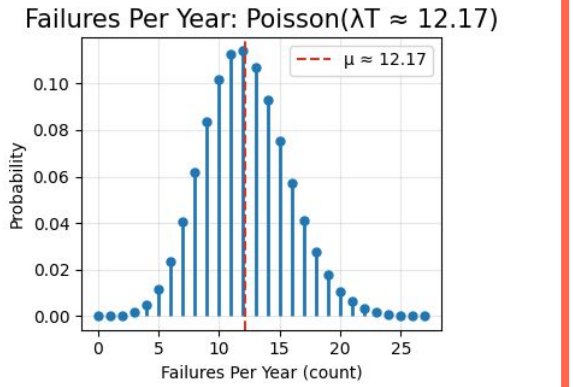




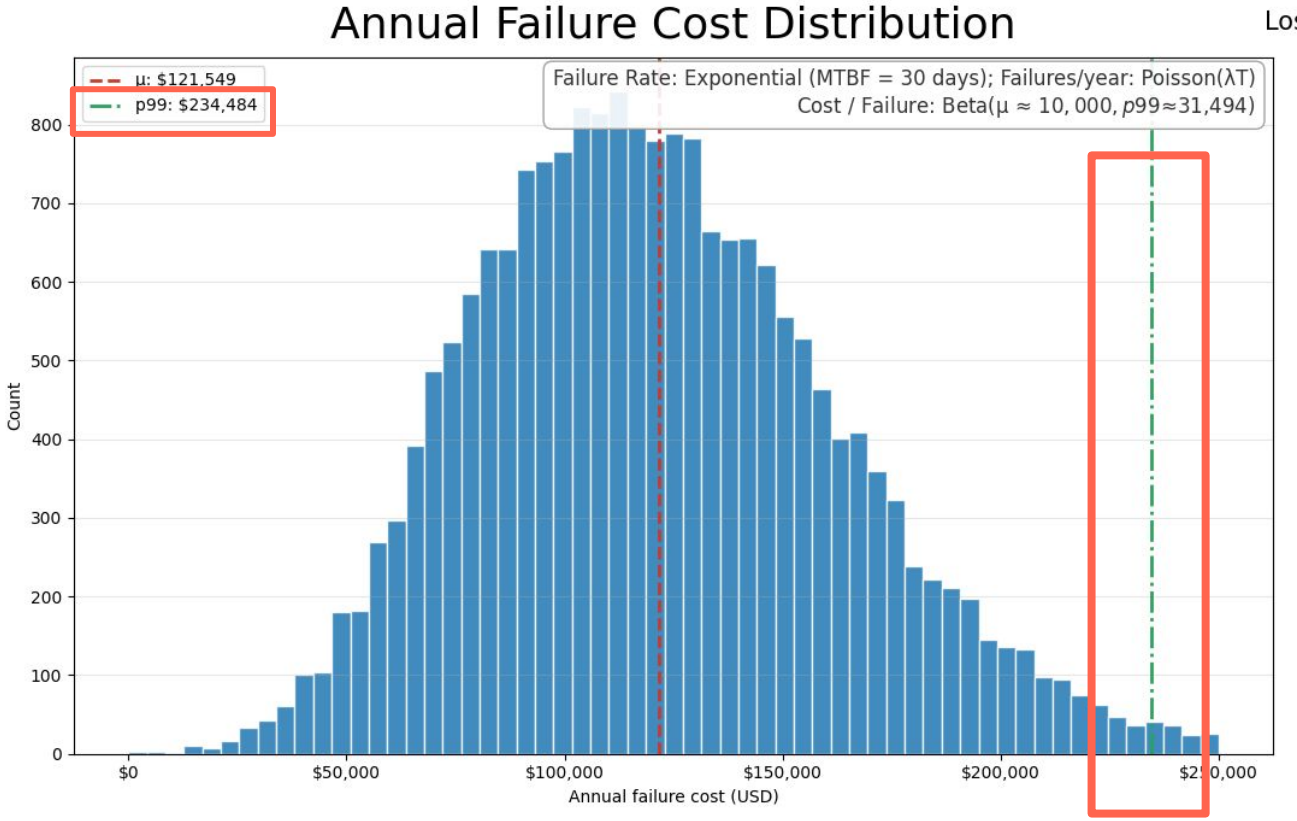


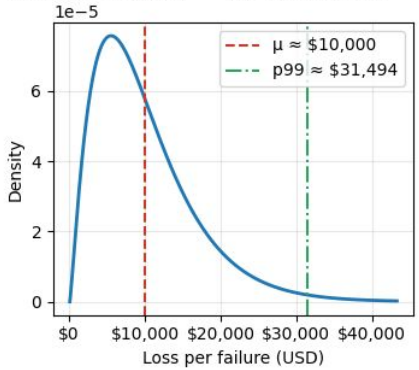


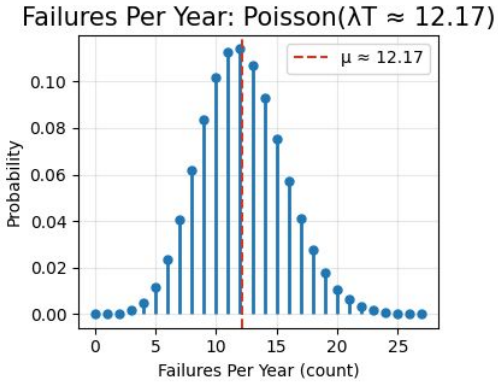




Failures happen at some rate!



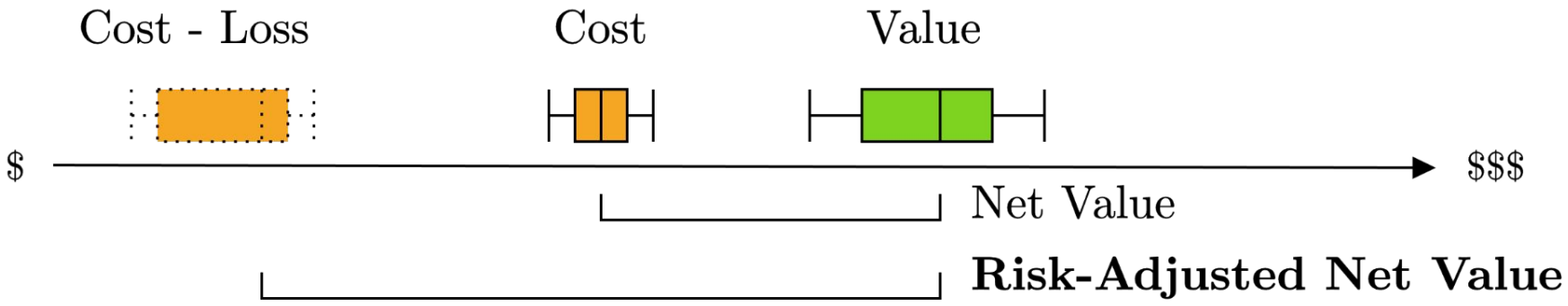




You probably care about the tail

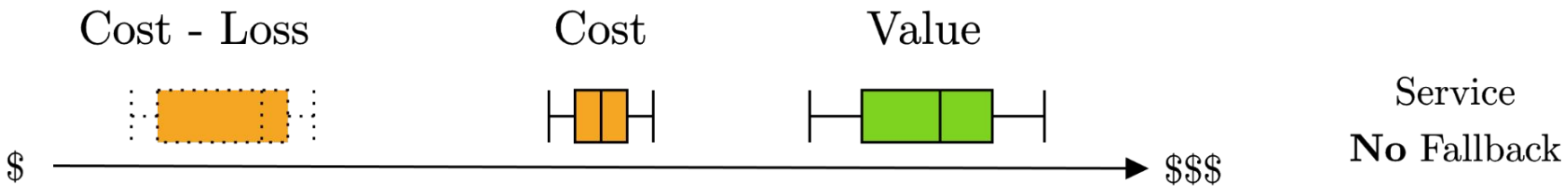
What is Efficiency?

Balance Value, Cost, and Risk



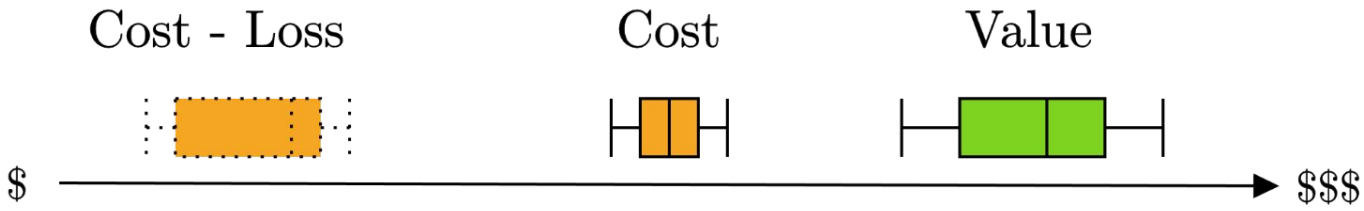
What is Efficiency?

Balance Value, Cost, and Risk

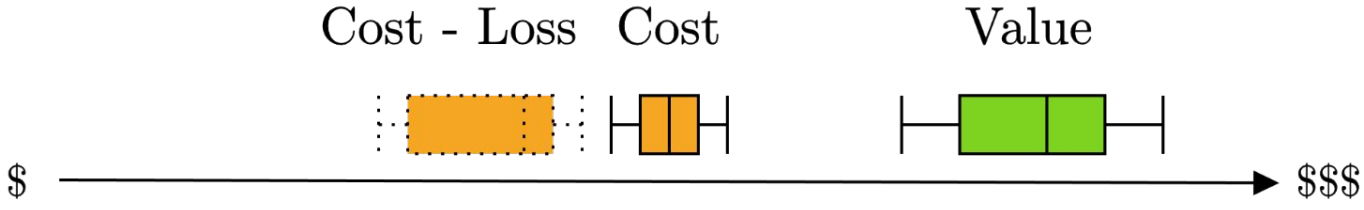


What is Efficiency?

Balance Value, Cost, and Risk



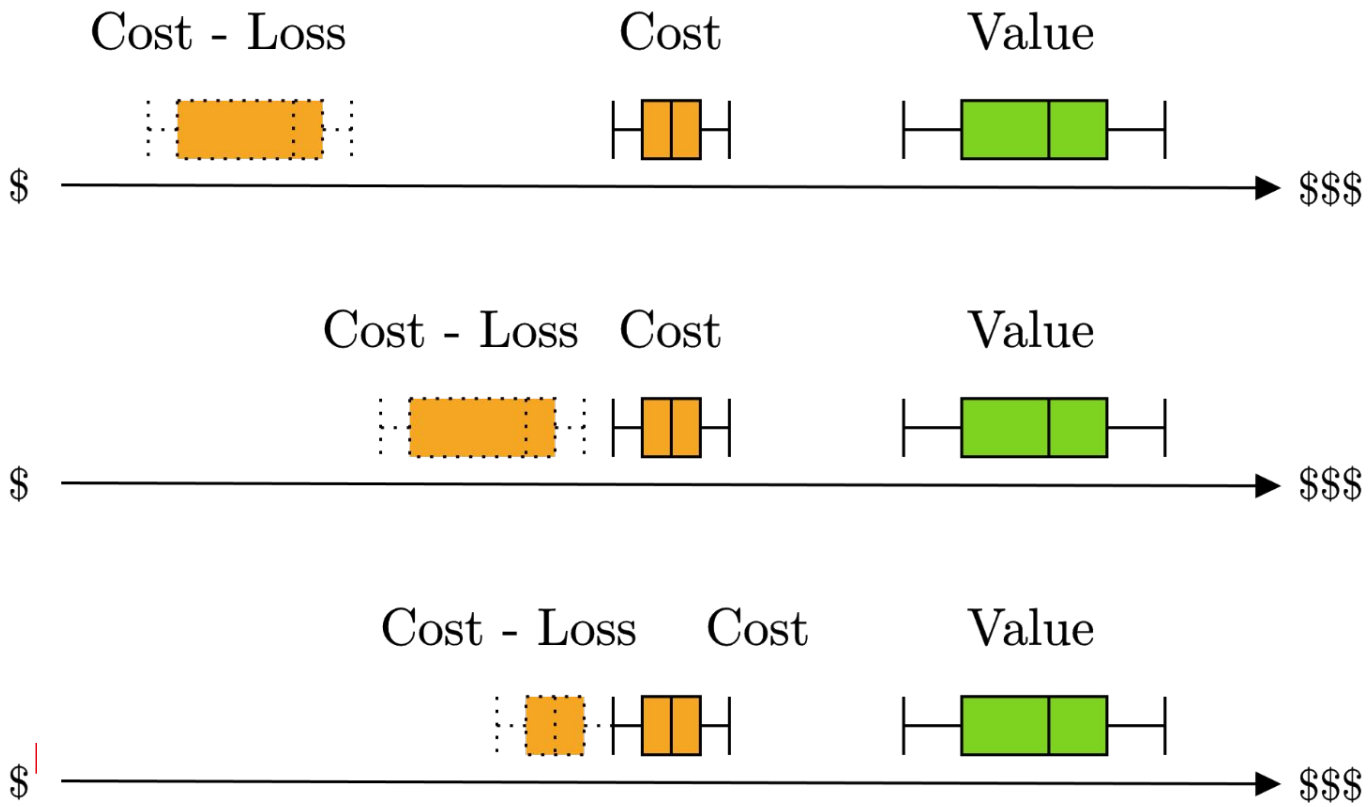
Service
No Fallback



Service
Degraded Fallback

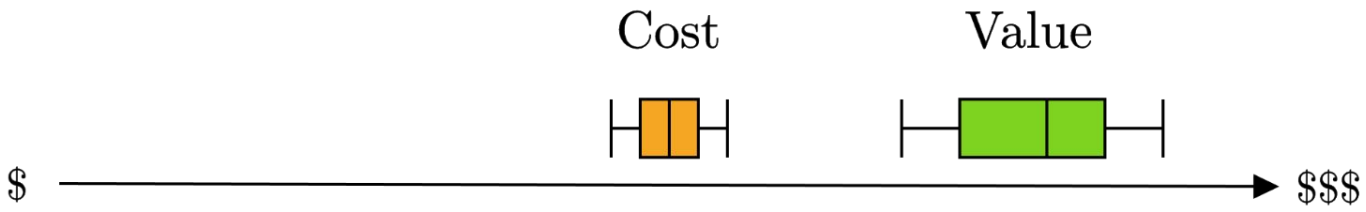
What is Efficiency?

Balance Value, Cost, and Risk



What is Efficiency?

Balance Value, Cost, and Risk



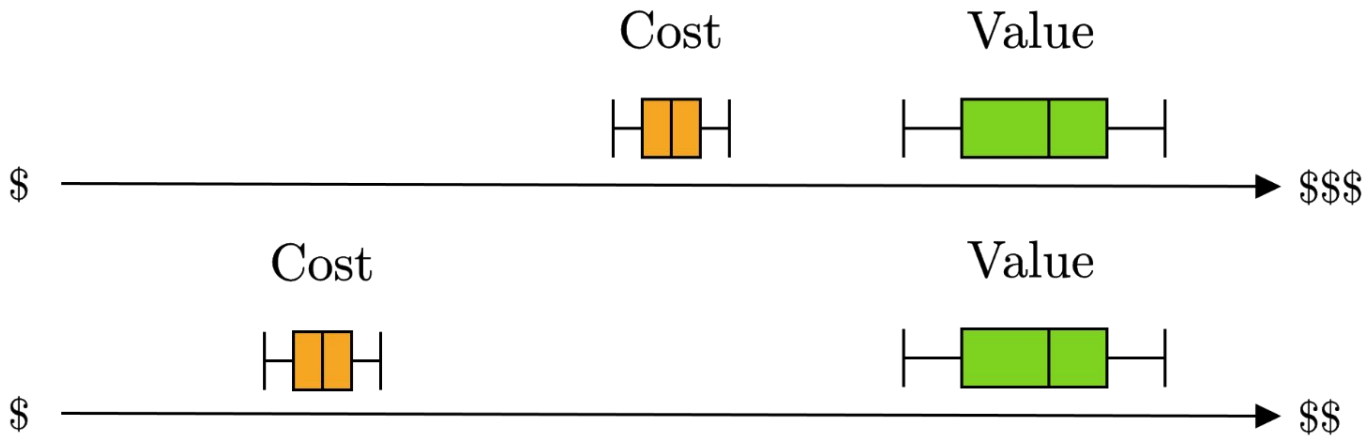
Bulk Service

OR

Batch

What is Efficiency?

Balance Value, Cost, and Risk



Bulk Service

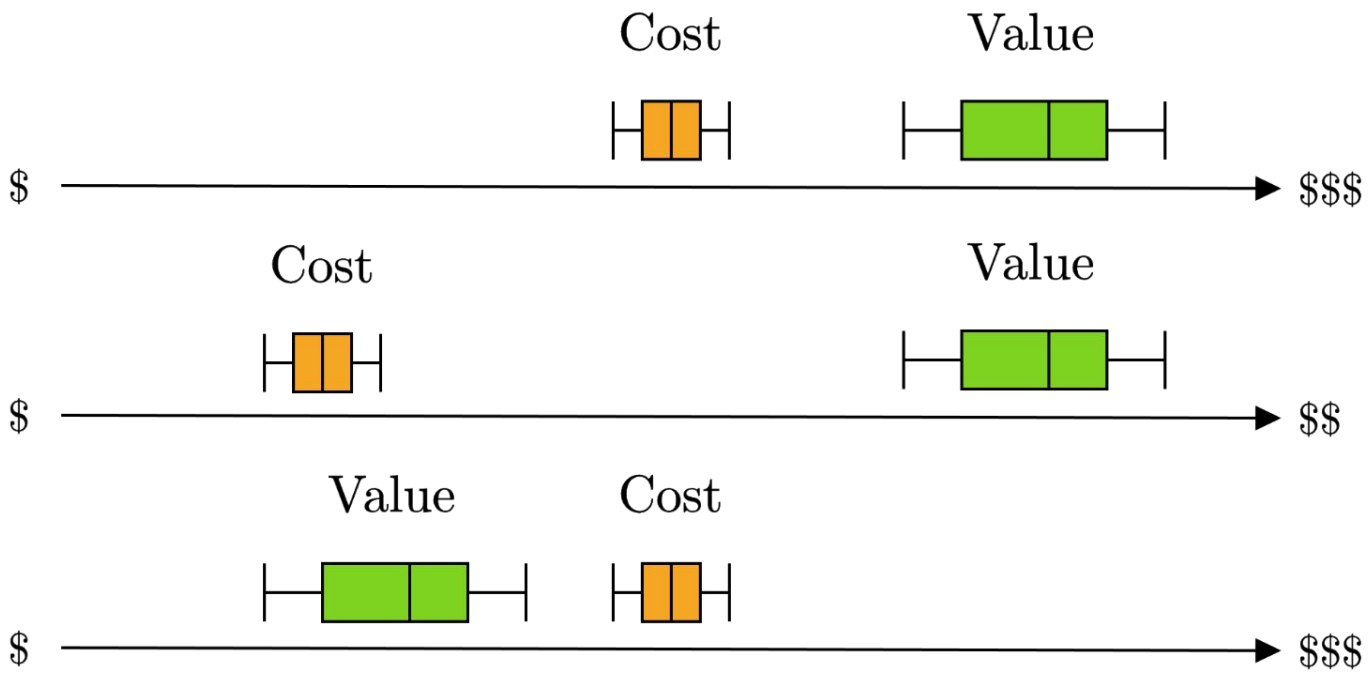
OR

Batch

Highly **Efficient**!

What is Efficiency?

Balance Value, Cost, and Risk



Bulk Service
OR
Batch

Highly **Efficient**!

Highly **Inefficient**!

System Efficiency

$$\mathbb{E}[Delay] \approx \frac{\rho}{1-\rho} \times \mathbb{E}[S] \times \left(\frac{C_S^2 + C_A^2}{2}\right)$$

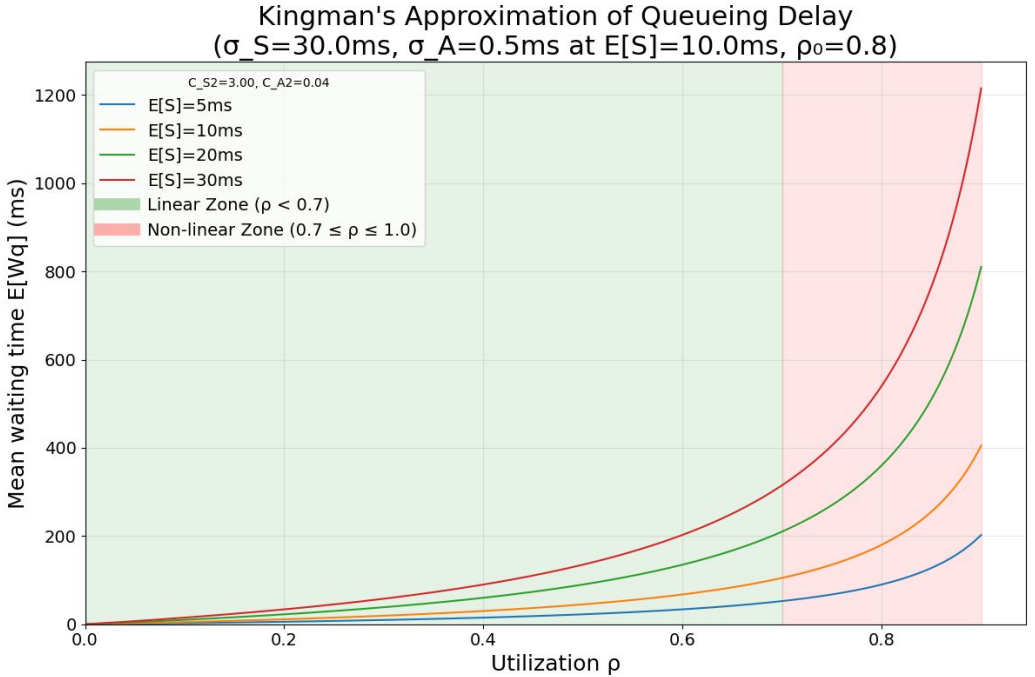
Kingsman's Approximation

ρ = *system load (utilization)*

$\mathbb{E}[S]$ = *avg svc time*

C_S^2 = *variation of svc time*

C_A^2 = *variation of arrival rate*



Mor Harchol-Balter et al. - Queueing Theory for Computer Systems Practitioners

System Efficiency

Raising Utilization is just one knob

$$\mathbb{E}[Delay] \approx \frac{\rho}{1-\rho} \times \mathbb{E}[S] \times \left(\frac{C_S^2 + C_A^2}{2}\right)$$

Kingsman's Approximation

ρ = *system load (utilization)*

$\mathbb{E}[S]$ = *avg svc time*

C_S^2 = *variation of svc time*

C_A^2 = *variation of arrival rate*

Make the Service Faster, a.k.a

Performance Engineering!

Brendan Gregg - Performance Analysis Methodology

N

System Efficiency

Raising Utilization is just one knob

$$\mathbb{E}[Delay] \approx \frac{\rho}{1-\rho} \times \mathbb{E}[S] \times \left(\frac{C_S^2 + C_A^2}{2}\right)$$

Kingsman's Approximation

ρ = *system load (utilization)*

$\mathbb{E}[S]$ = *avg svc time*

C_S^2 = *variation of svc time*

C_A^2 = *variation of arrival rate*

Make the Service *Consistently* Fast, a.k.a

Performance Engineering!

Brendan Gregg - Performance Analysis Methodology

System Efficiency

Raising Utilization is just one knob

$$\mathbb{E}[Delay] \approx \frac{\rho}{1-\rho} \times \mathbb{E}[S] \times \left(\frac{C_S^2 + C_A^2}{2}\right)$$

Kingsman's Approximation

ρ = *system load (utilization)*

$\mathbb{E}[S]$ = *avg svc time*

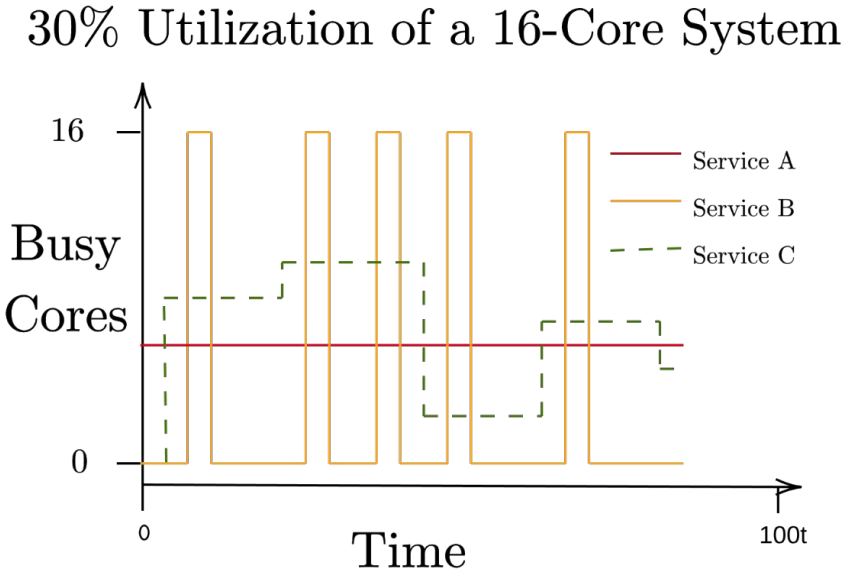
C_S^2 = *variation of svc time*

C_A^2 = *variation of arrival rate*

Balance Load (Choice-of-N)

Spread Load (Token Buckets)

Utilization is often a Lie



Kingsman's Approximation

ρ = *system load (utilization)*

$\mathbb{E}[S]$ = *avg svc time*

C_S^2 = *variation of svc time*

C_A^2 = *variation of arrival rate*

Fleet Efficiency

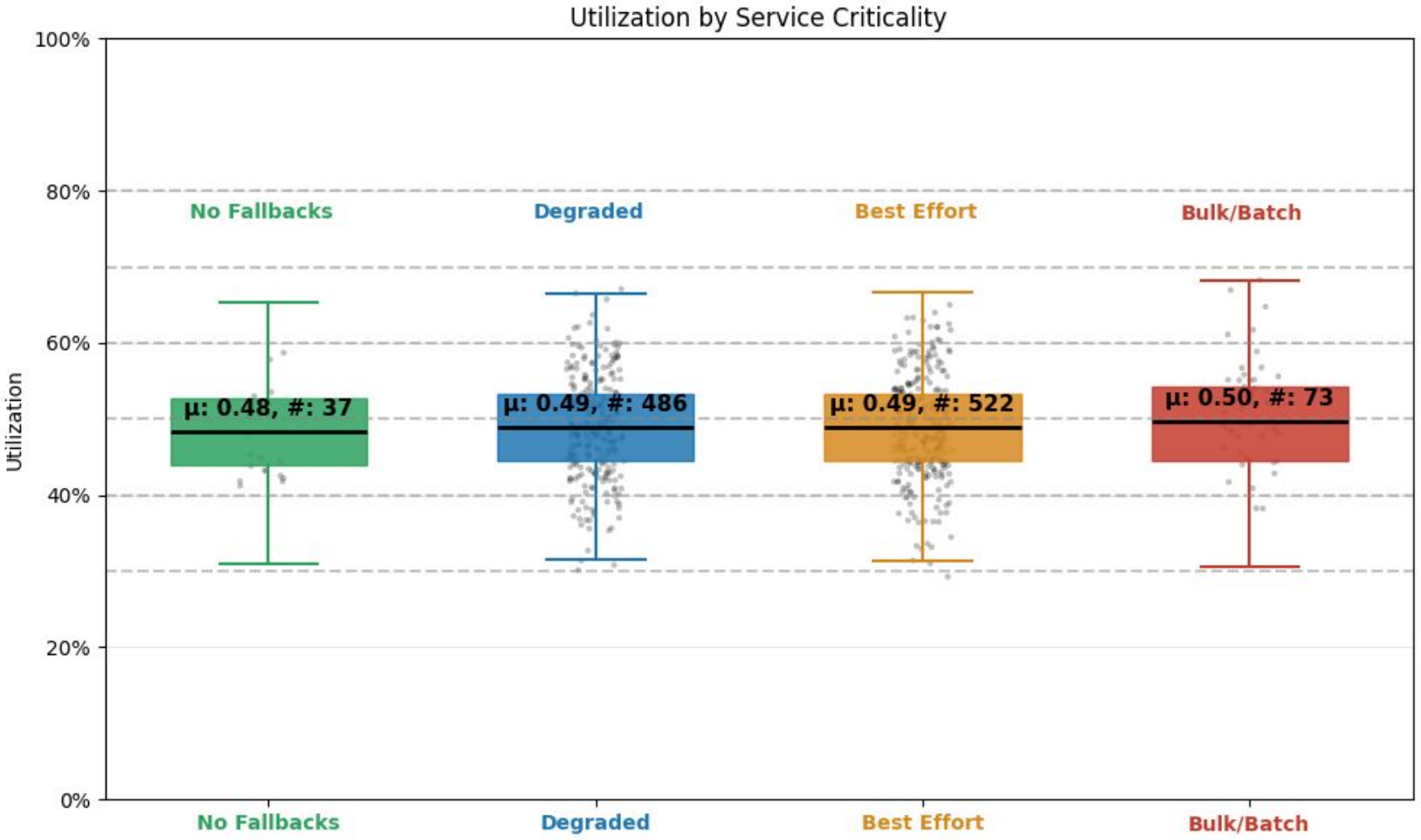
Instead of Raising Utilization, ask

Where do we spend our dollars?

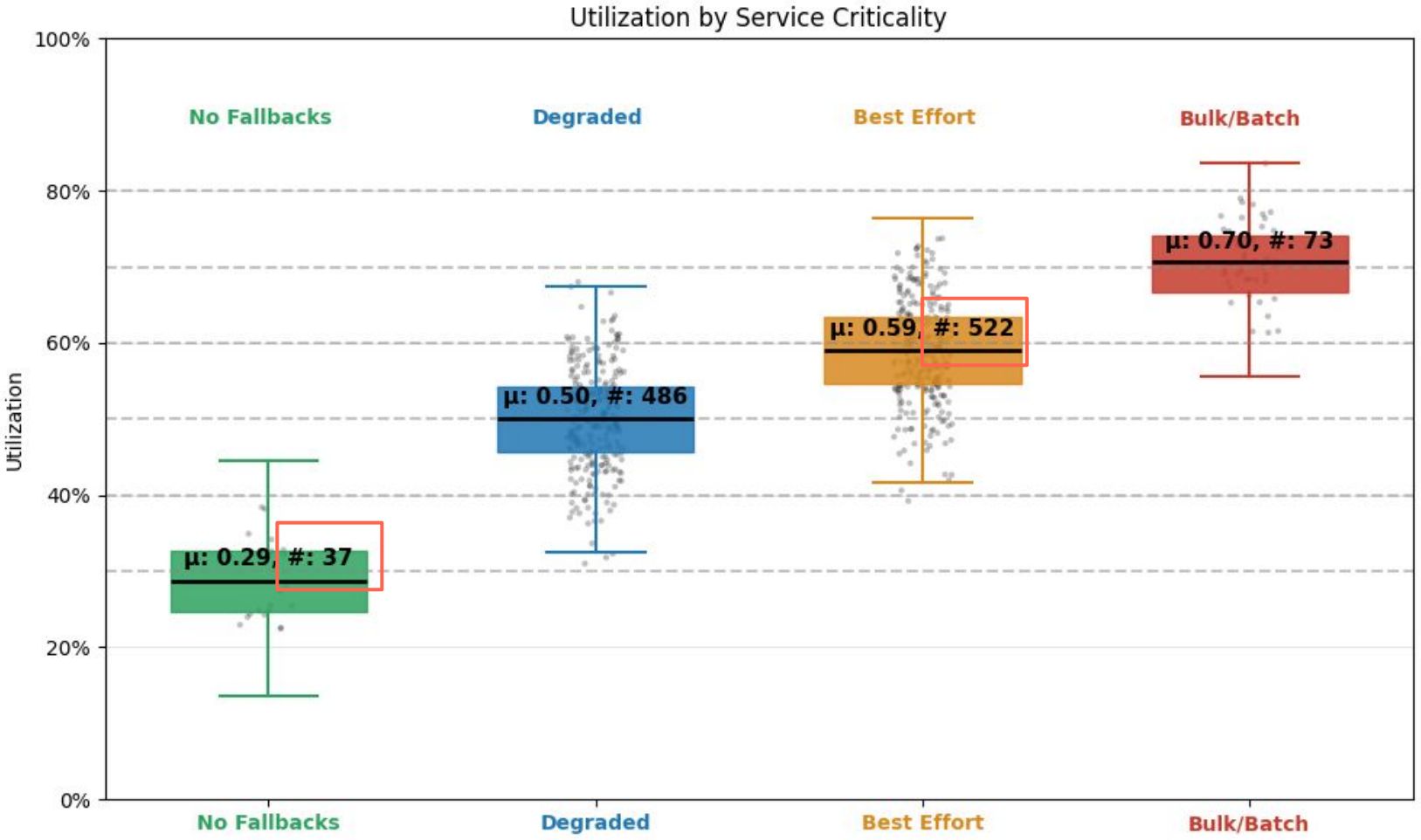
Do those dollars mitigate risk?

What lever do I pull to reduce cost?

Not Efficient Fleet



Efficient Fleet



Services - Respond with low latency

Services - Respond with low latency

Batches - Schedule with low latency

Reliability

Services - Respond with low latency

Batches - Schedule with low latency

Failures are rare in frequency (TBF)

Reliability

Services - Respond with low latency

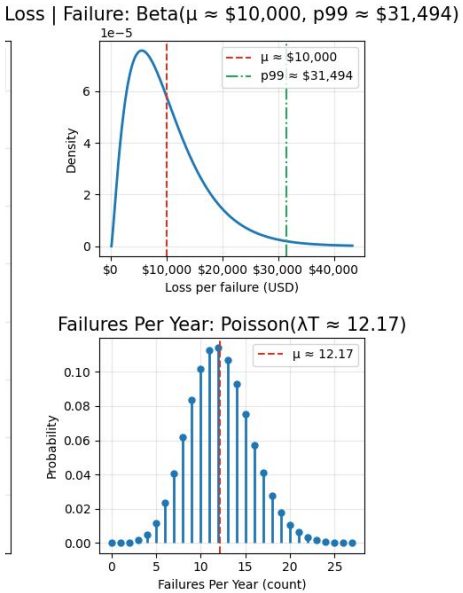
Batches - Schedule with low latency

Failures are rare in frequency (TBF)
Failures recover quickly (TTR)

Reliability

Services - Respond with low latency

Batches - Schedule with low latency



Failures are rare in frequency (TBF)
Failures recover quickly (TTR)
Failures have low impact (Blast Radius)

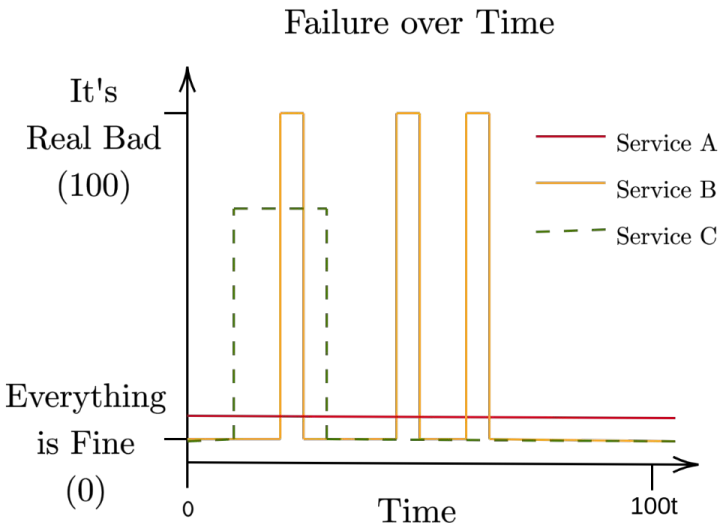
Instead of Measuring Nines, ask

How often does it fail (TBF)?

What is the impact (Blast Radius)?

How long to recover (TTR)?

Nines are Deceptive



Different Failure Modes Require Different Solutions!

	Service A	Service B	Service C
Nines	99.999	99.999	99.999
Time Between Failure (P_{05} + P_{50})	(0, 0)	(3t, 10t)	(10t, 40t)
Time to Recovery (P_{05}, P_{50})	(∞, ∞)	(1t, 2t)	(4t, 5t)
Impact of Failure (P_{05}, P_{50})	(1, 1)	(100, 100)	(50, 80)
Potential Resolution	Hedging or Tests	Load Shedding or Backpressure	Alerting or Failover or Bu

Understand
Hardware Supply

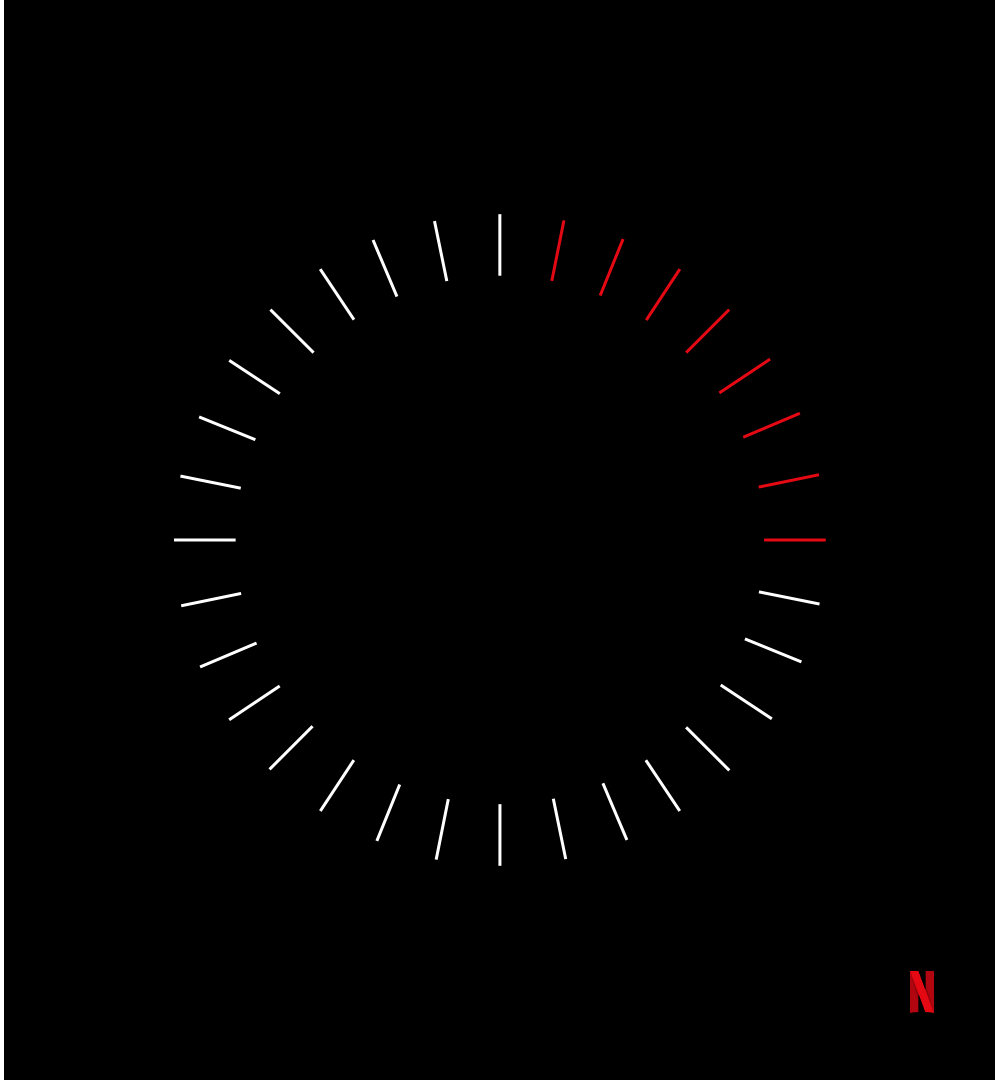
Understand
Software Demand

Balance
Compute Supply
Software Demand

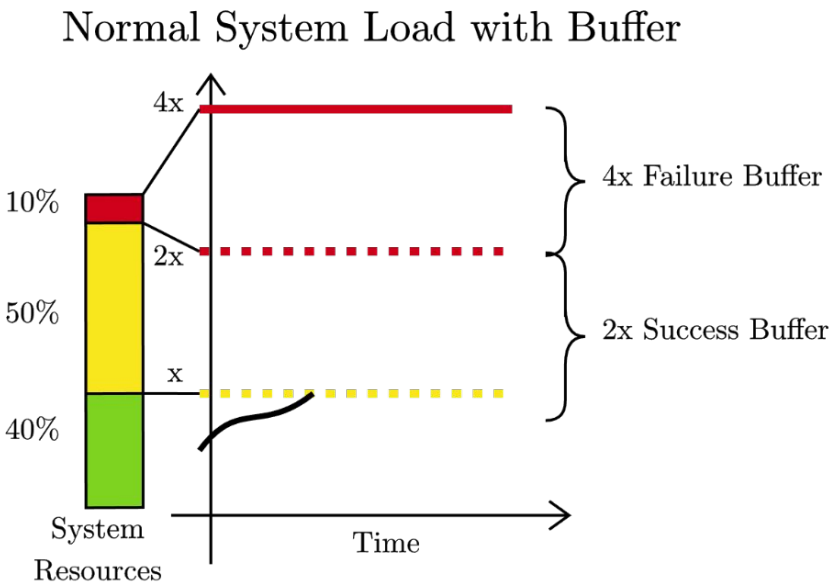
Understand Supply

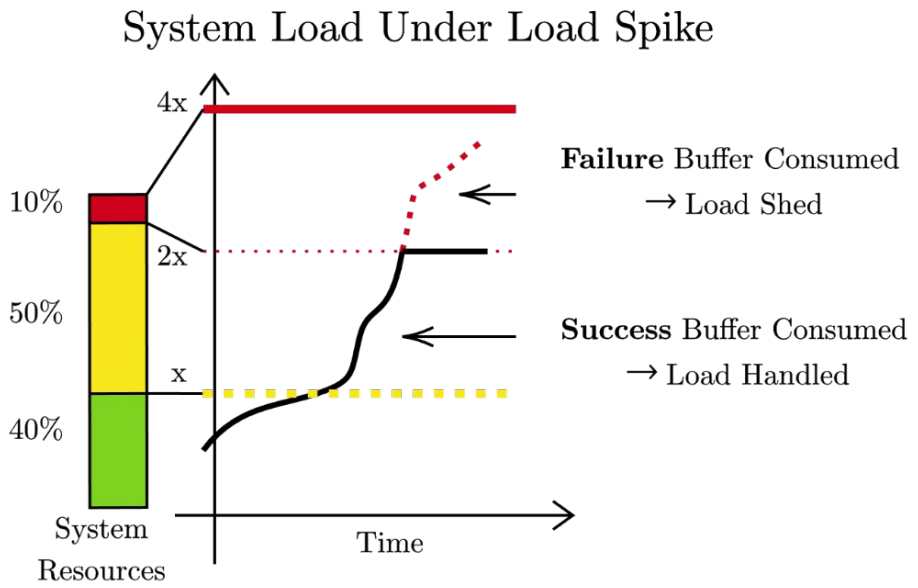
Capacity Planning

Hardware Supply Signals

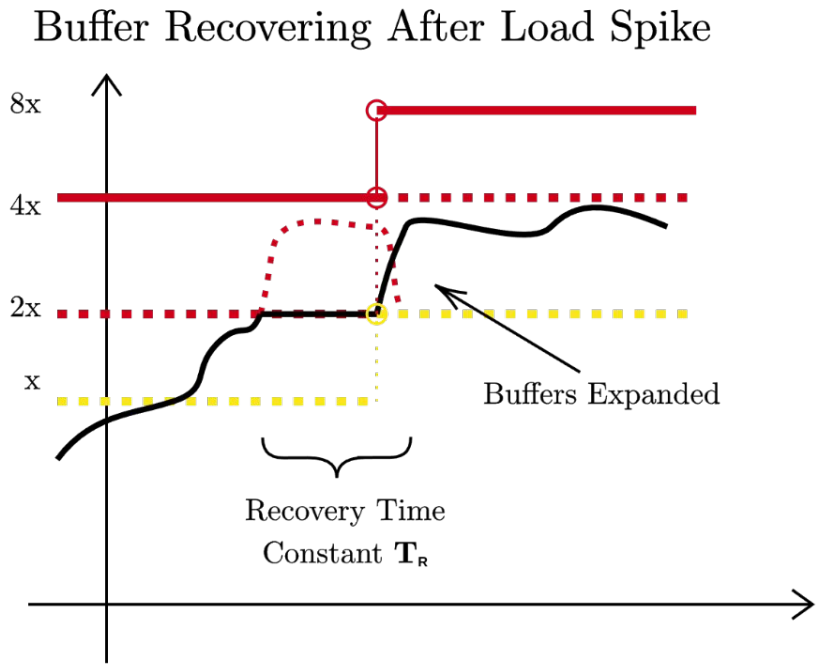


Reason about Headroom with **Buffers**





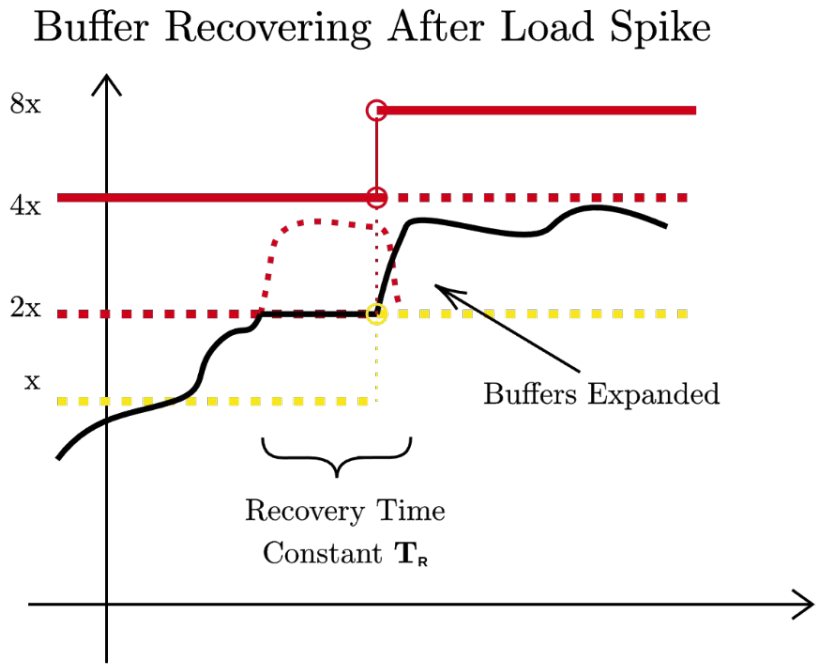
Buffer is Linked to Business Outcomes




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

Low utilization is a tradeoff!

Buffer is Linked to Business Outcomes

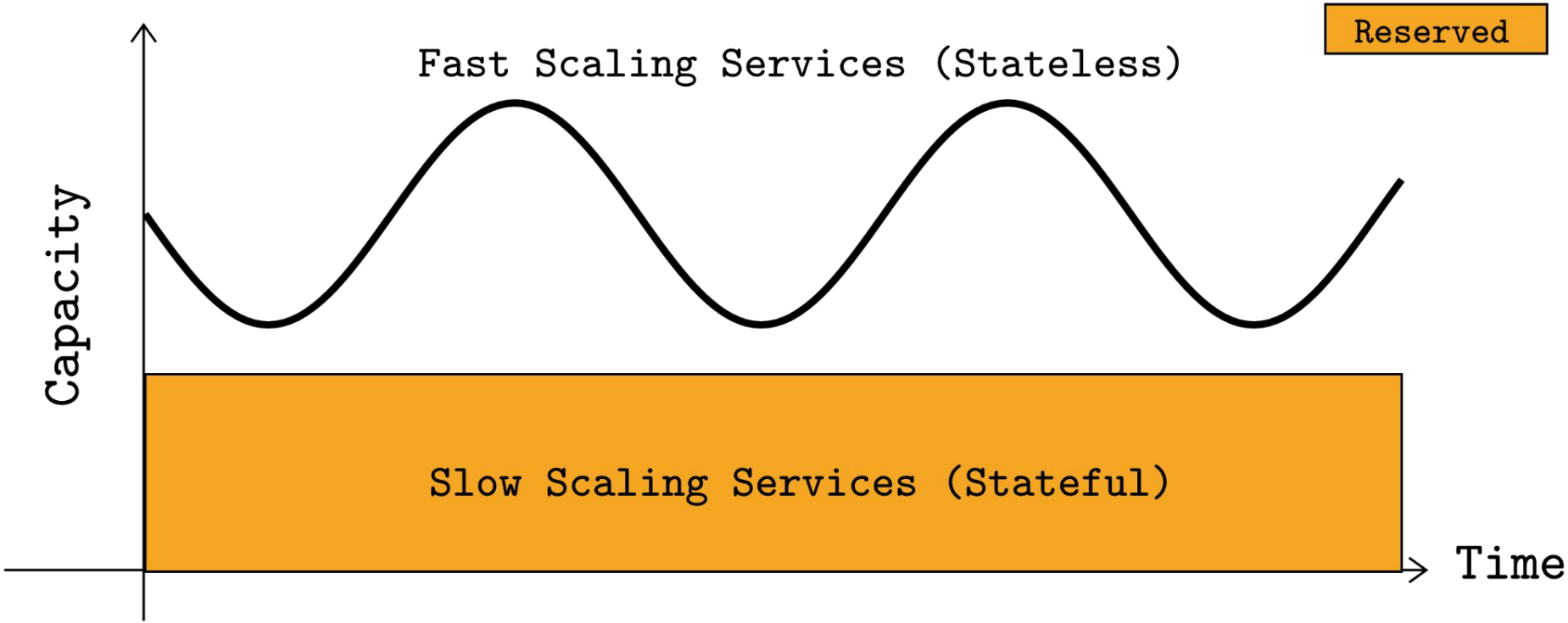



```
Buffer  = F(
        Criticality Tier,
        Business Domain,
        Recovery Time Constant
)
```

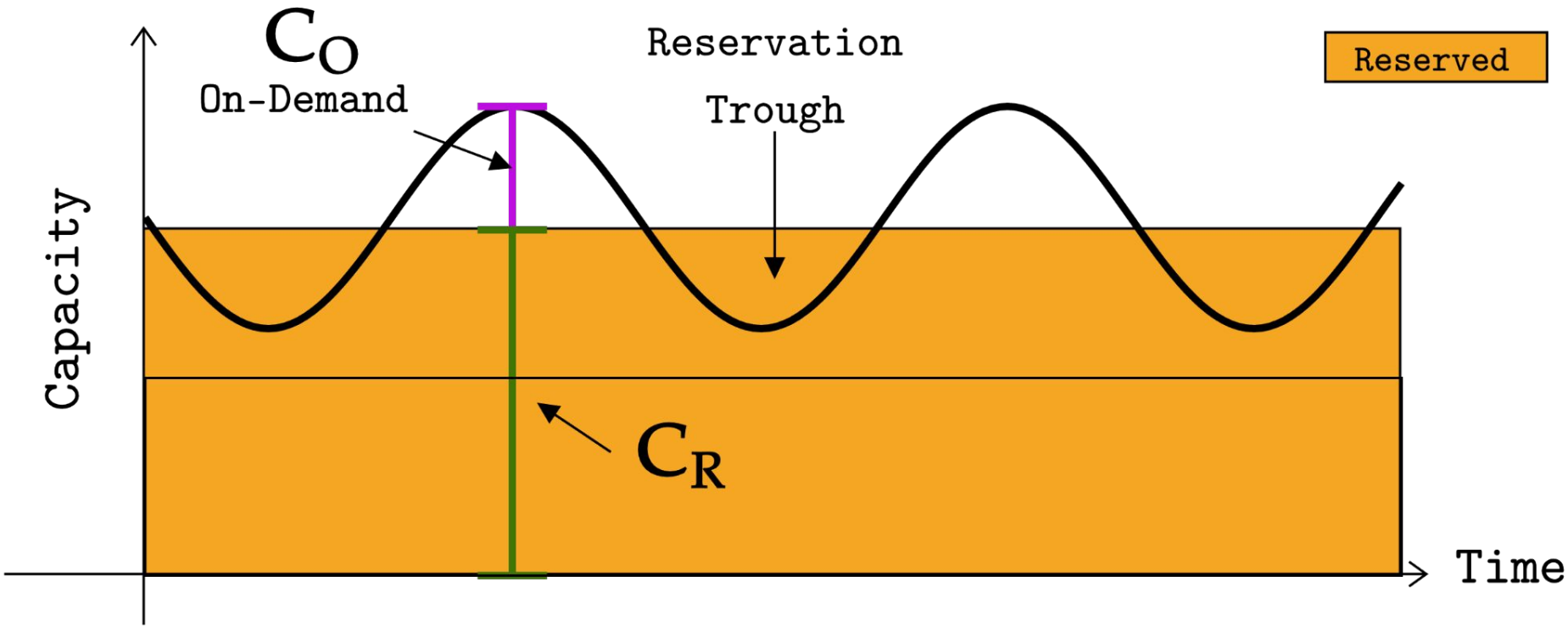
Low utilization is a tradeoff!

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

Cloud Reality #1
Still have to Plan

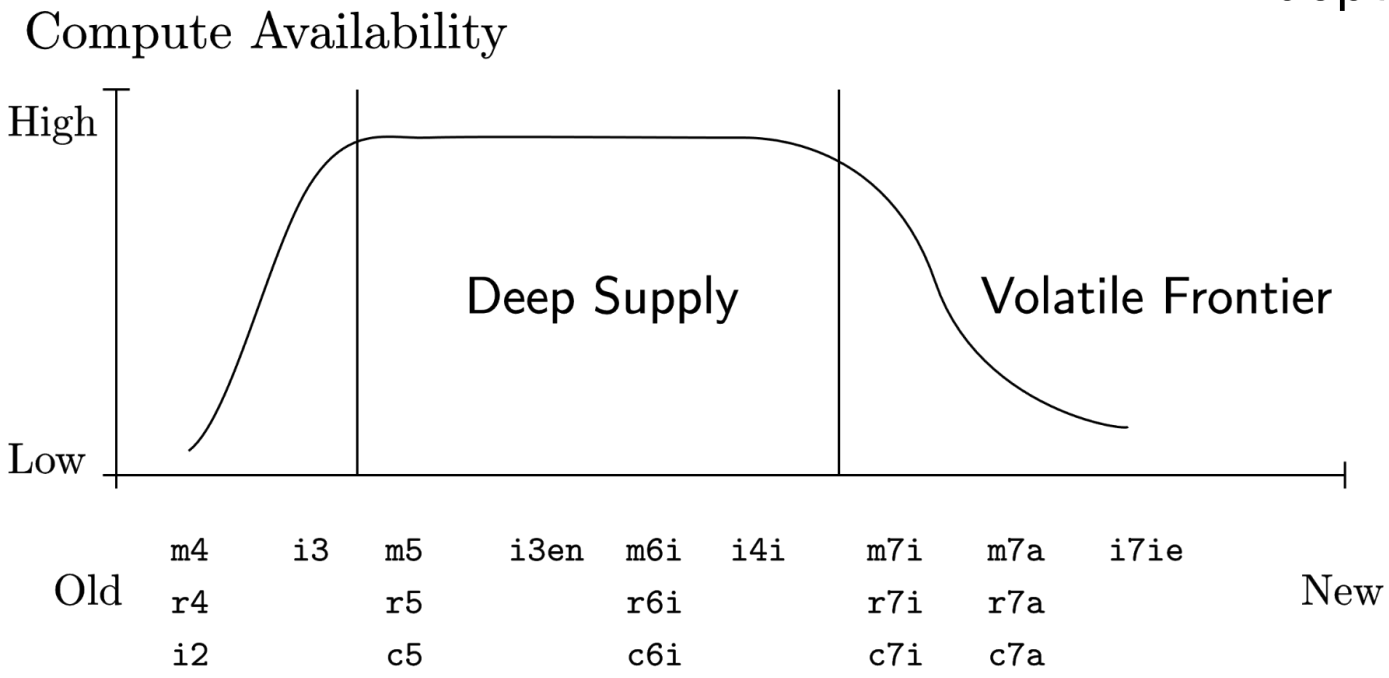


Cloud Reality #1
Still have to Plan



Cloud Reality #2
Computers have
Variable Supply

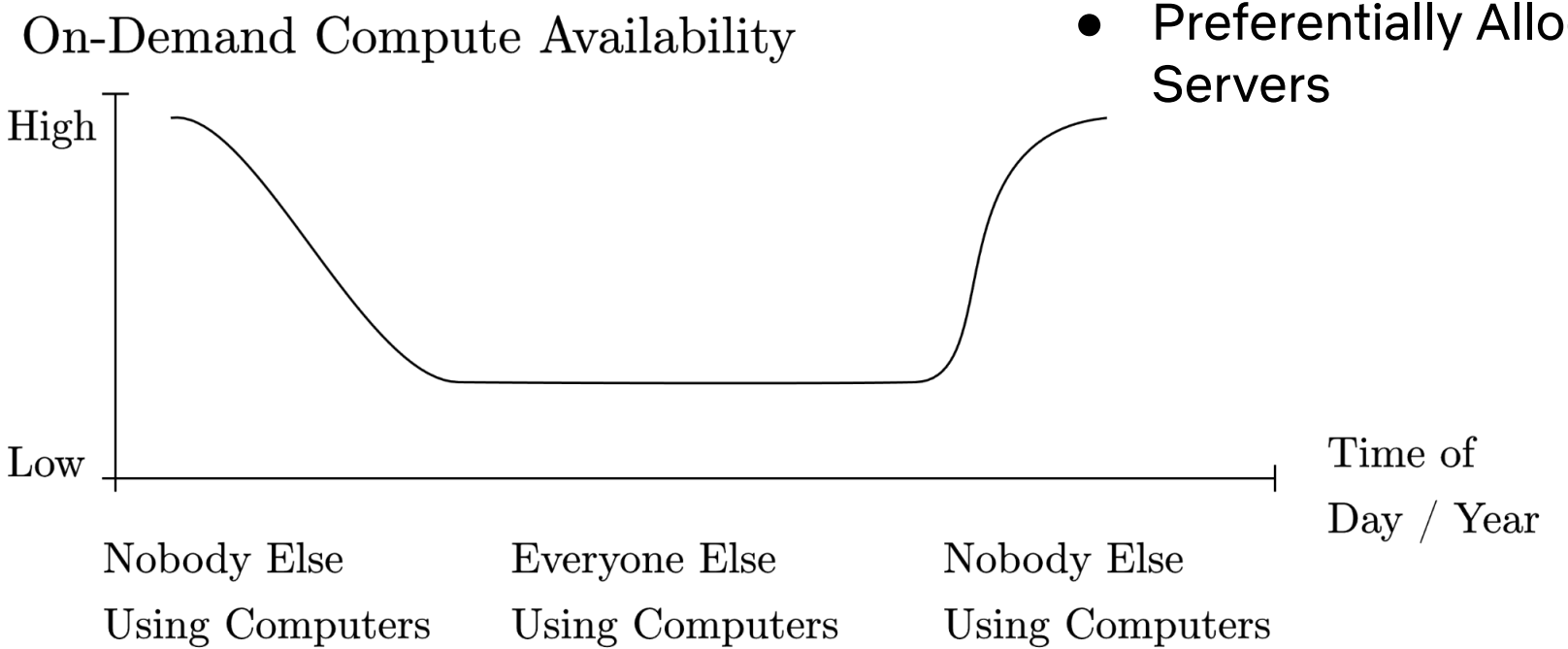
Have to *carefully*
Adopt New Shapes



Cloud Reality #2
Computers have
Variable Supply

Options:

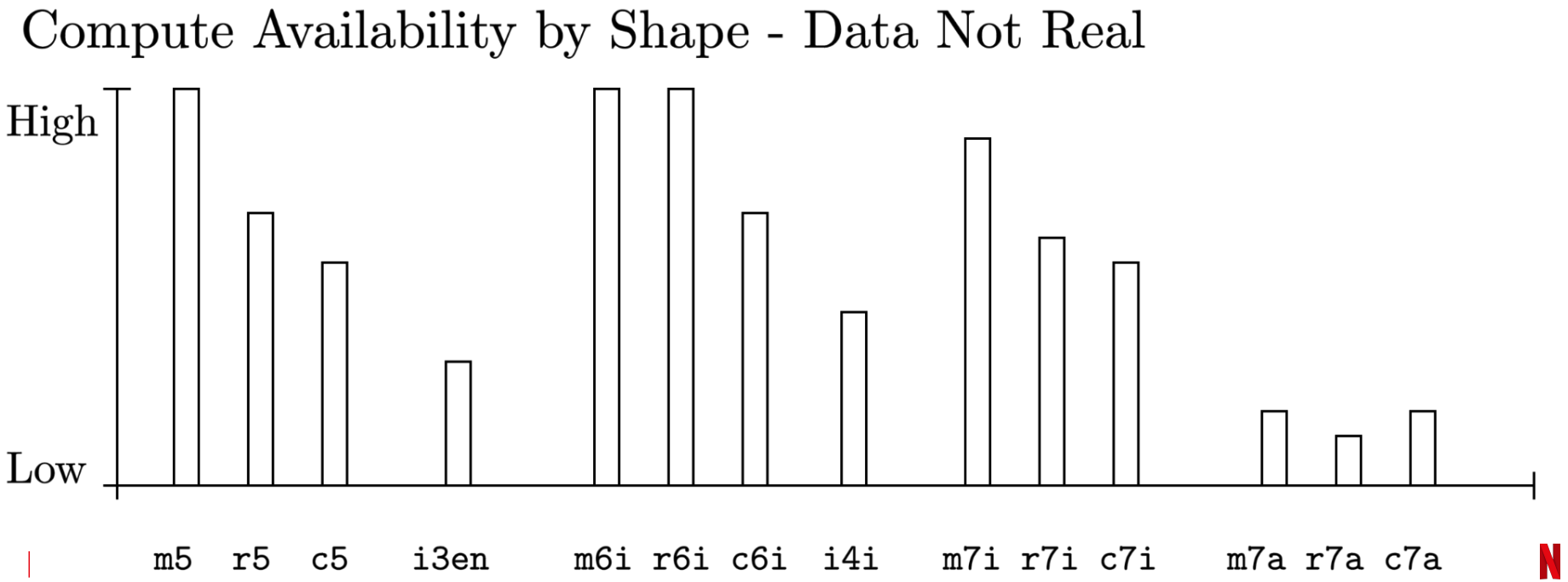
- Reservations
- On Demand Reservations
- Leverage Buffers
- Preferentially Allocating Servers



Cloud Reality #2
Computers have
Variable Supply

More flexibility == More Capacity

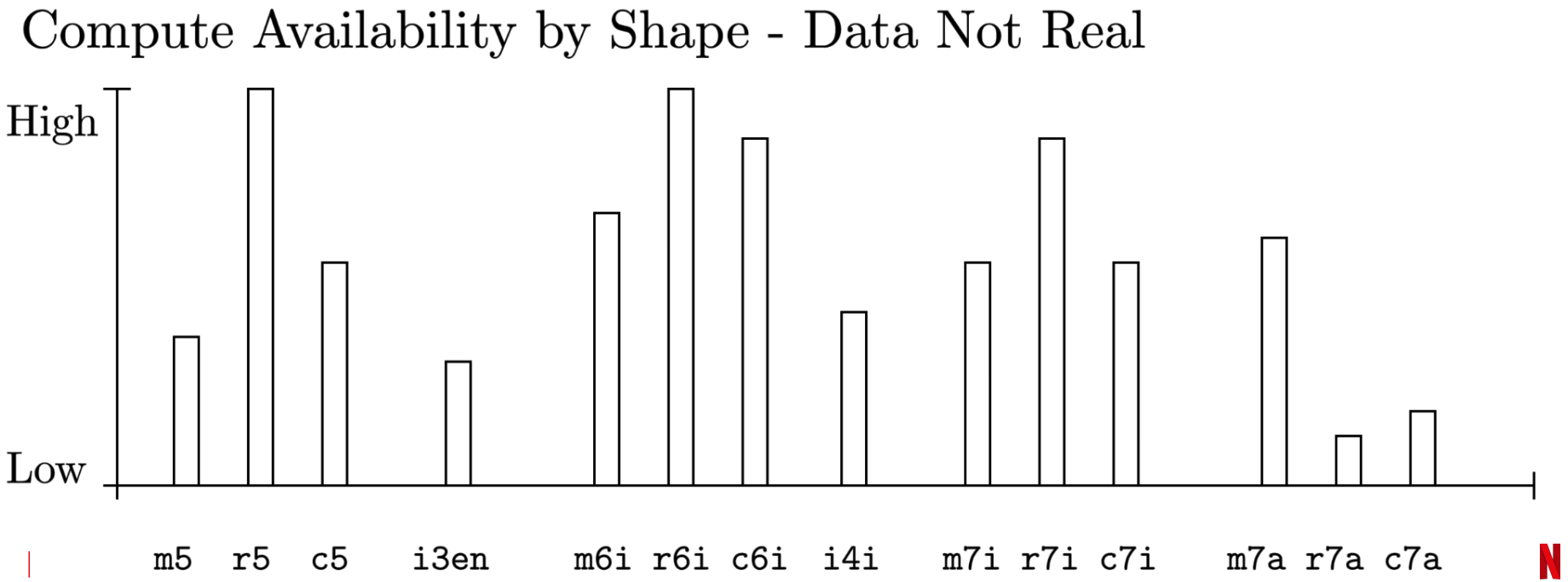
Assign workloads by resources not names



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

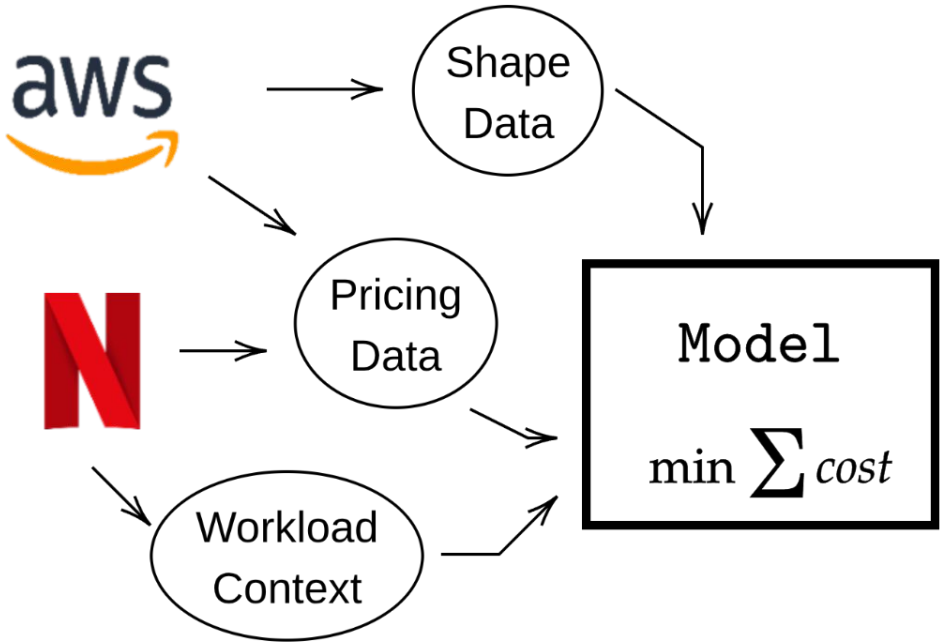
```
"m7a.4xlarge": {
  "name": "m7a.4xlarge",
  "cpu": 16,
  "cpu_cores": 16,
  "cpu_ghz": 3.7,
  "cpu_ipc_scale": 1.5,
  "ram_gib": 61.04,
  "net_mbps": 6250.0,
  "drive": null
},
```

```
"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

A generic toolkit for modeling capacity requirements in the cloud.

Capacity Model




```
shapes = [
  { shape:c7a.2xlarge
    cost: 2.0k, alt=m7a.2, r7a.2}
  { shape:c6a.4xlarge
    cost: 2.3k, alt=c7a.4...},
  { shape: m7a.2xlarge ...
```

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

Strategy: Buffer, and lots of math

```
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
)
```

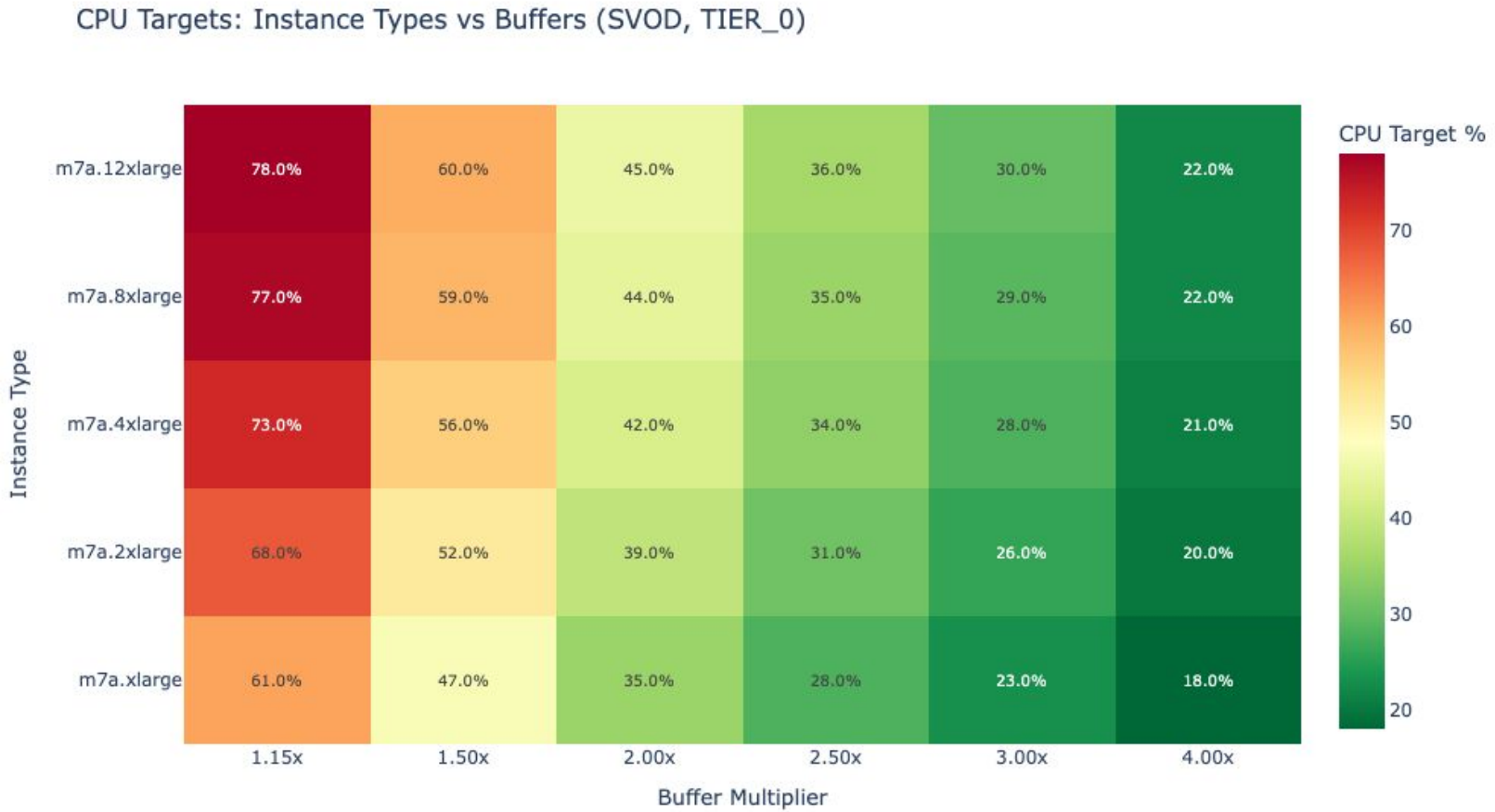
```
Least Regret Choice:
--------------------
 12 m5d.xlarge costing 8973.94

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**

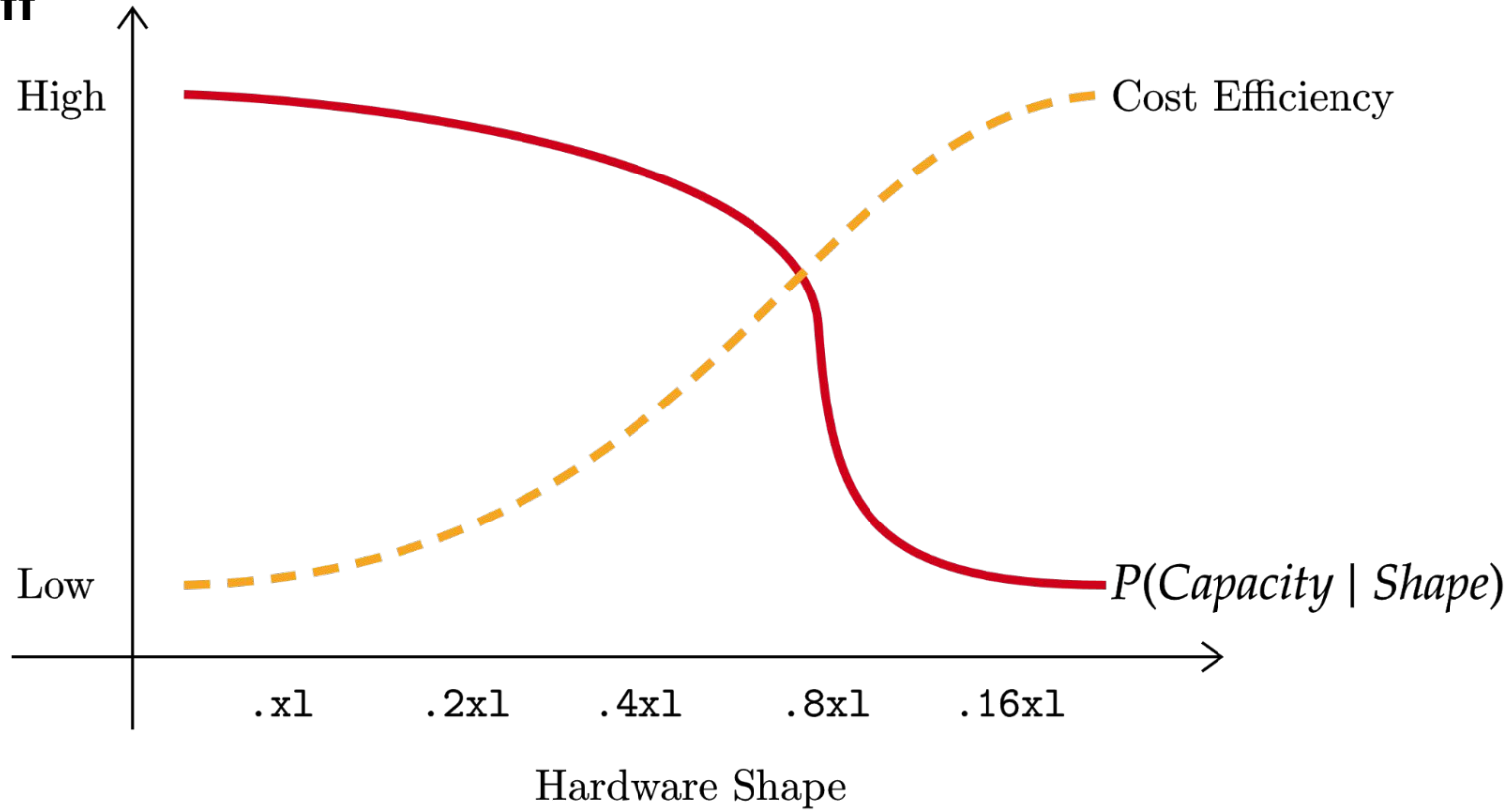
Buffer Differs By Server Type

	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

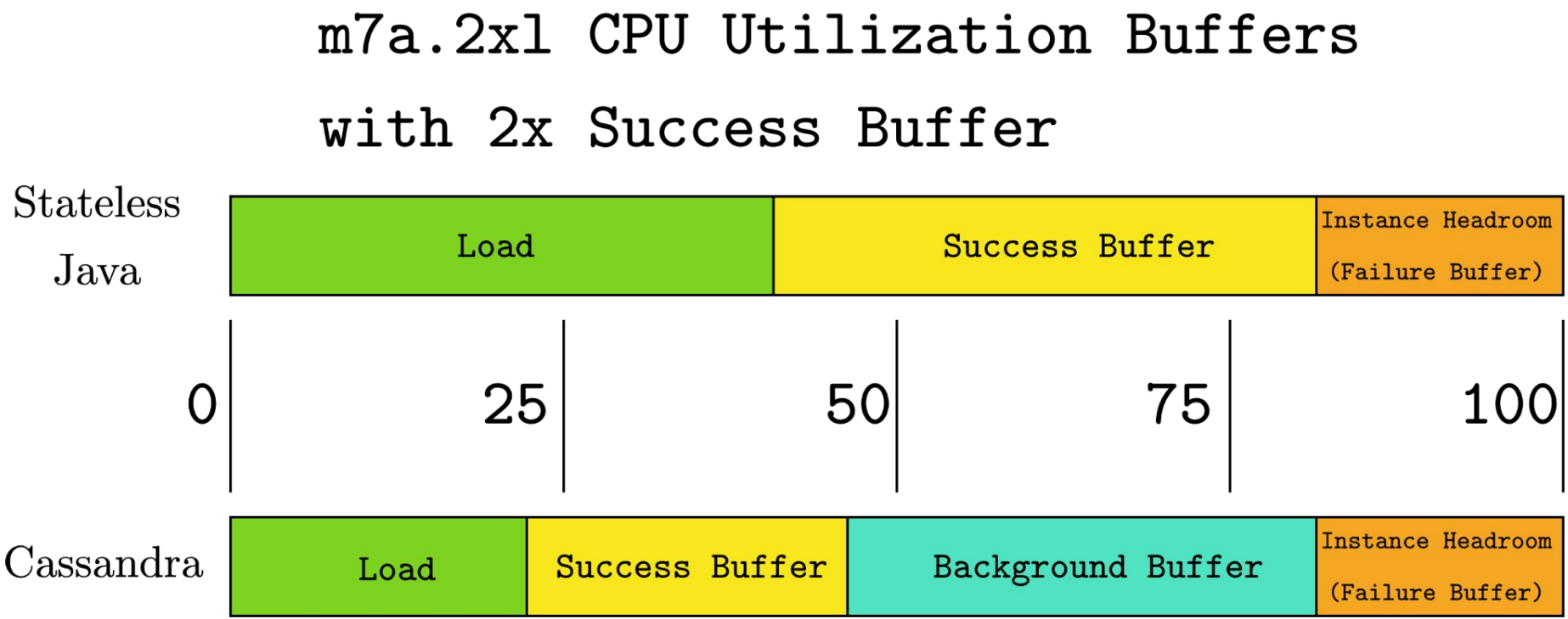
Buffer Grid


CPU Targets: Instance Types vs Buffers (SVOD, TIER_0)
Instance Type
m7a.12xlarge
m7a.8xlarge
m7a.4xlarge
m7a.2xlarge
m7a.xlarge
78.0%
60.0%
45.0%
36.0%
30.0%
22.0%
77.0%
59.0%
44.0%
35.0%
29.0%
22.0%
73.0%
56.0%
42.0%
34.0%
28.0%
21.0%
68.0%
52.0%
39.0%
31.0%
26.0%
20.0%
61.0%
47.0%
35.0%
28.0%
23.0%
18.0%
1.15x
1.50x
2.00x
2.50x
3.00x
4.00x
Buffer Multiplier
CPU Target %
70
60
50
40
30
20

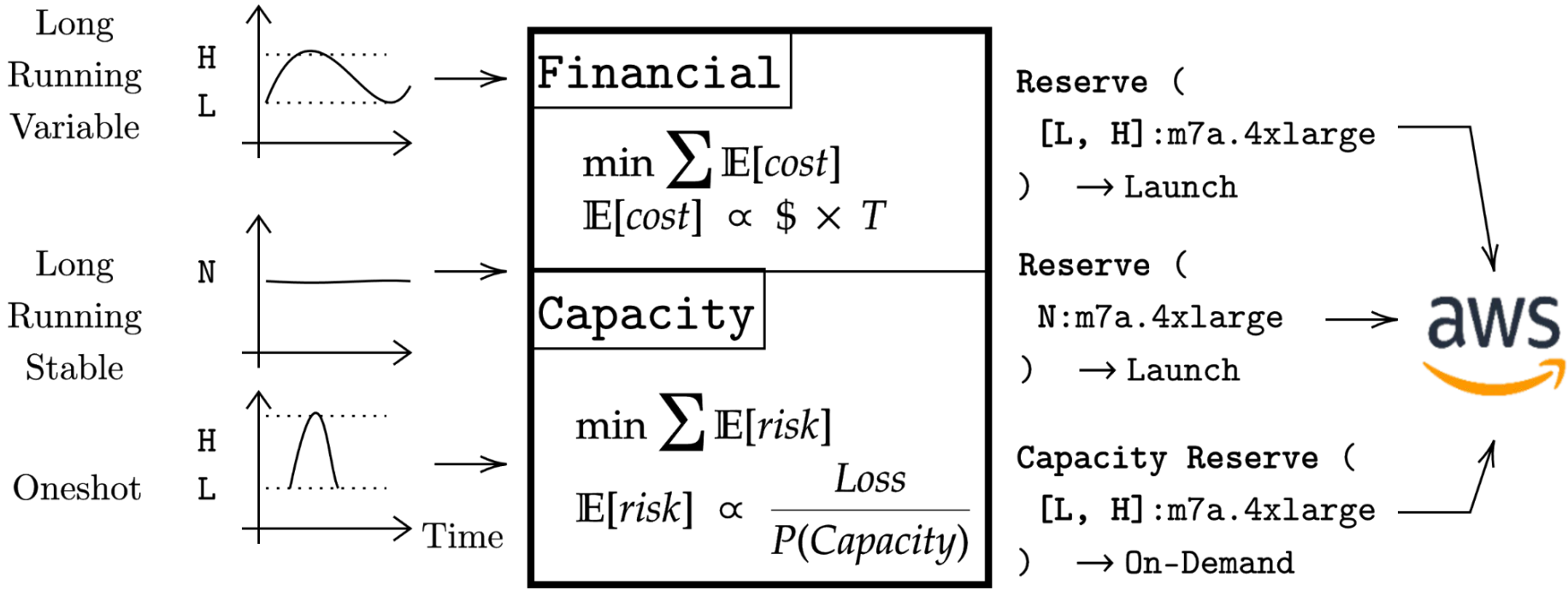
Capacity Tradeoff



Buffer Differs By Workload



Ensure Supply



Takeaways

Buffer is the relevant configuration

Capacity must be prioritized

Time-to-Recover matters

Compute is Fungible via the **Right Math™**

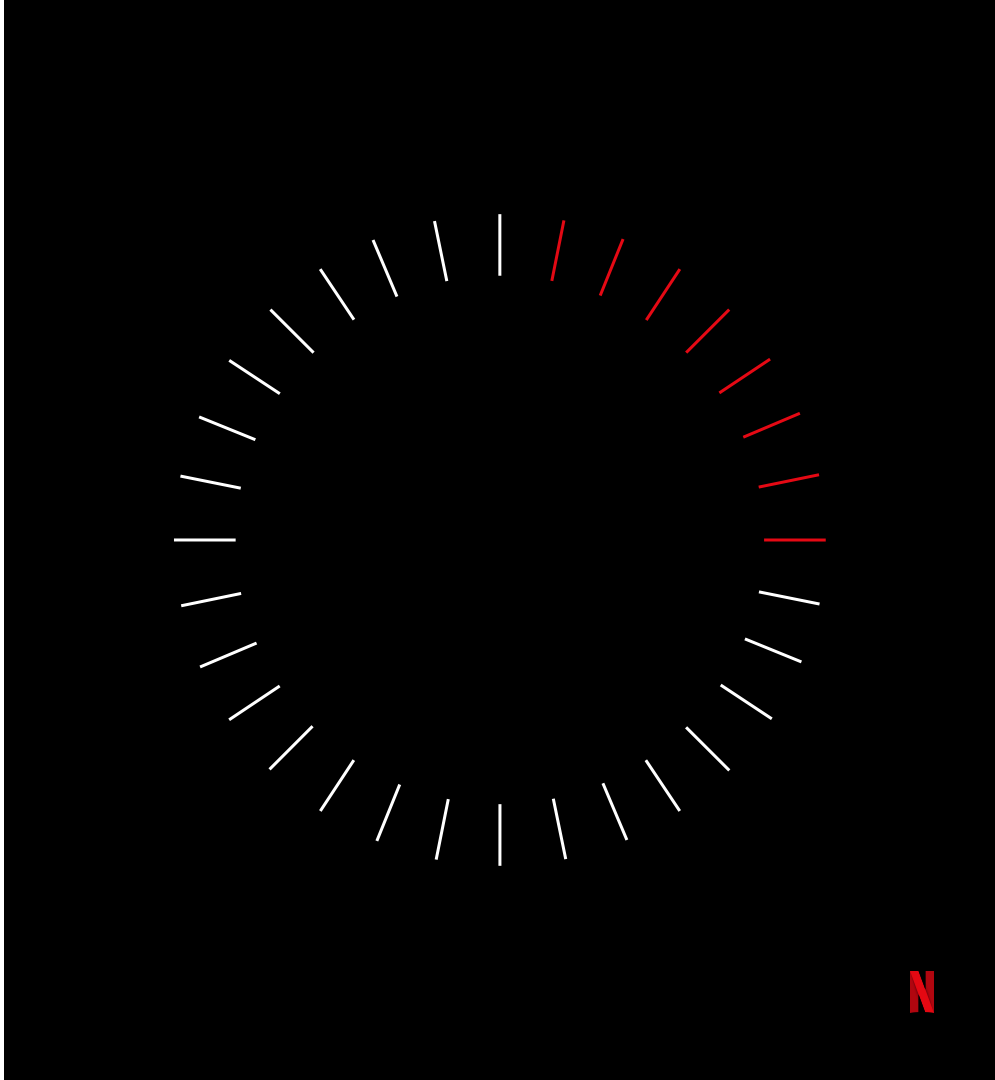
Understand
Hardware Supply

Understand
Software Demand

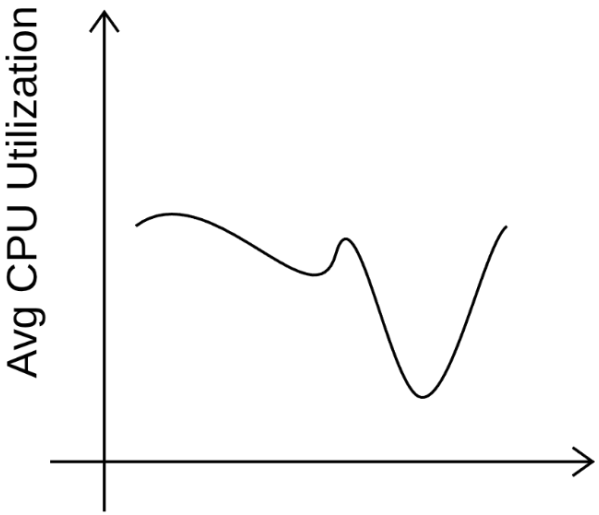
Balance
Compute Supply
Software Demand

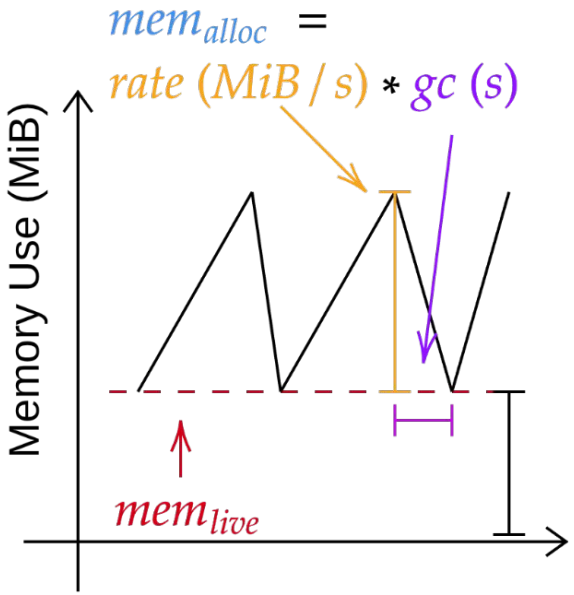
Understand Demand

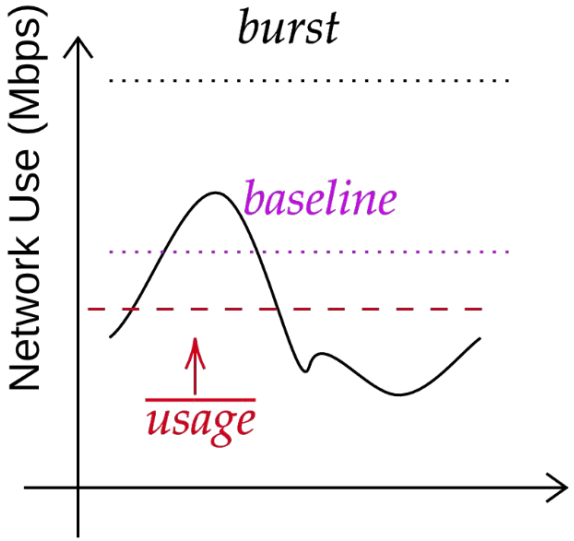
Workload Profile



Workload analysis





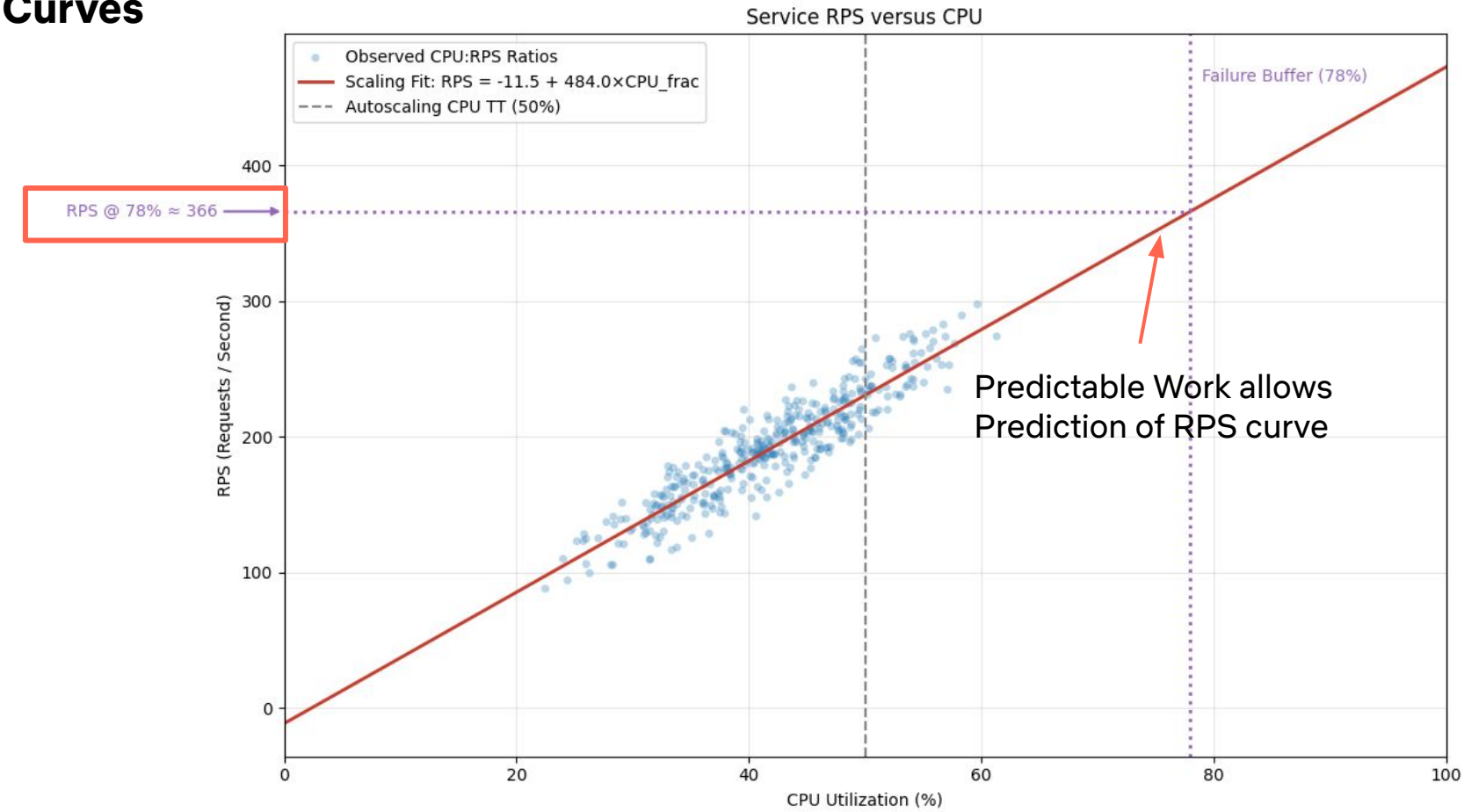


$$CPU_{util} = max(\overline{cpu\ _{util}})$$

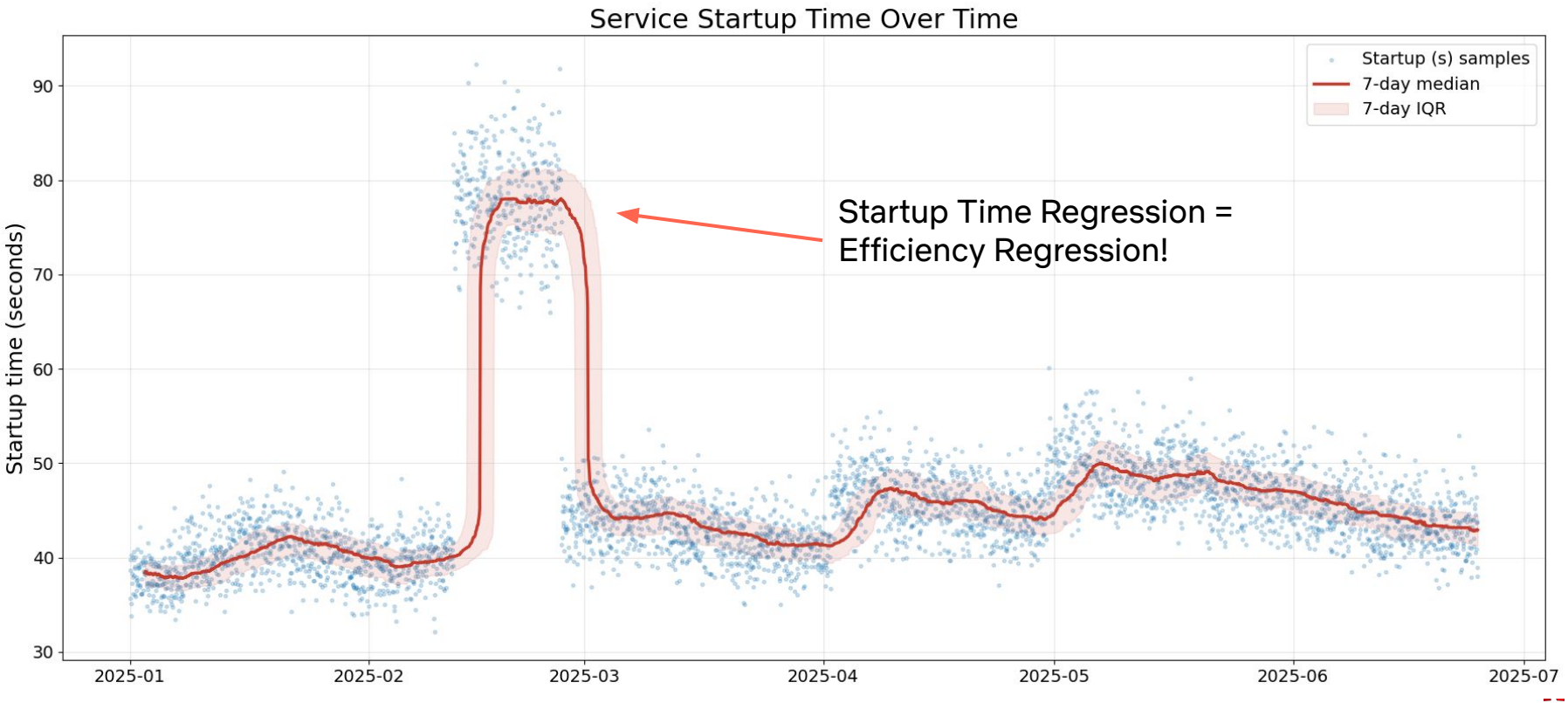
$$Mem = mem_{live} + mem_{alloc}$$

$$Net_{util} = \frac{\overline{usage}}{baseline}$$

Scaling Curves



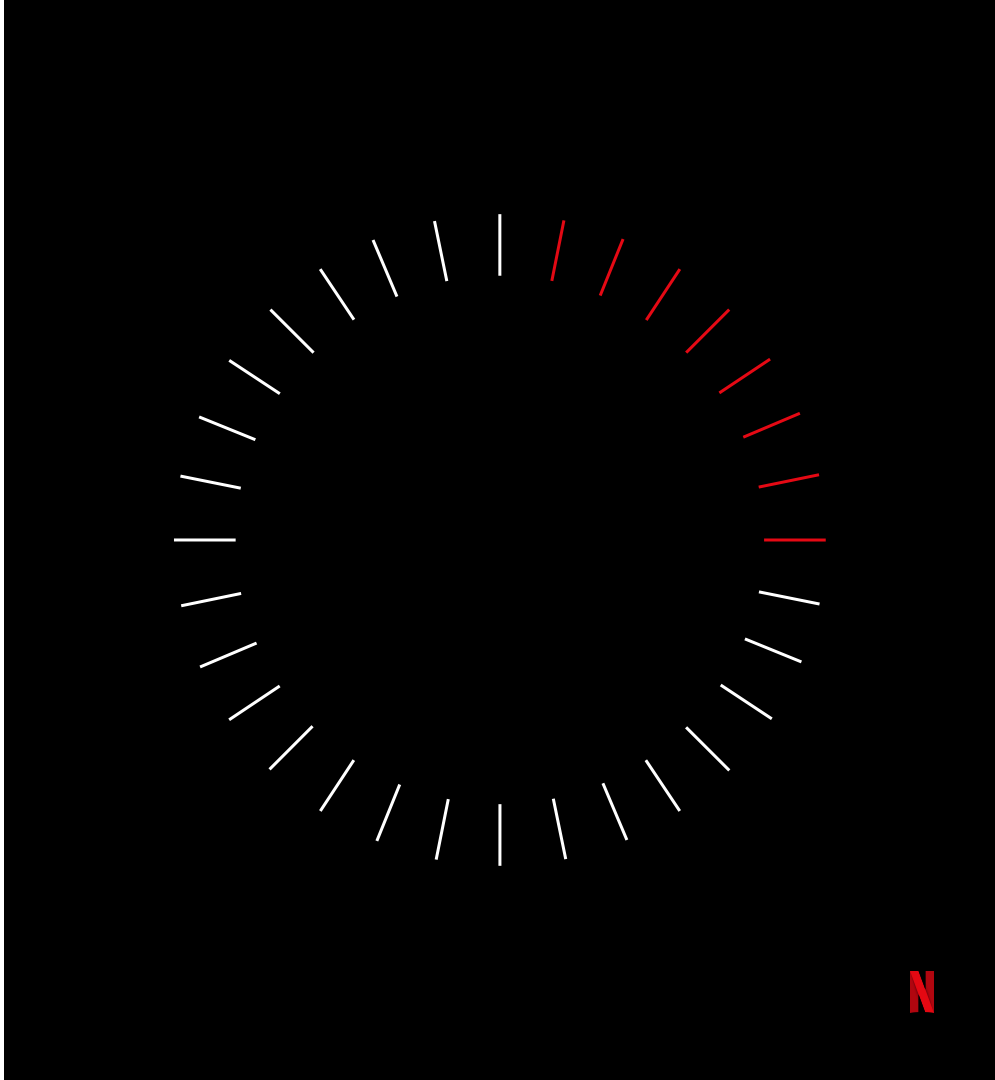
Startup Times



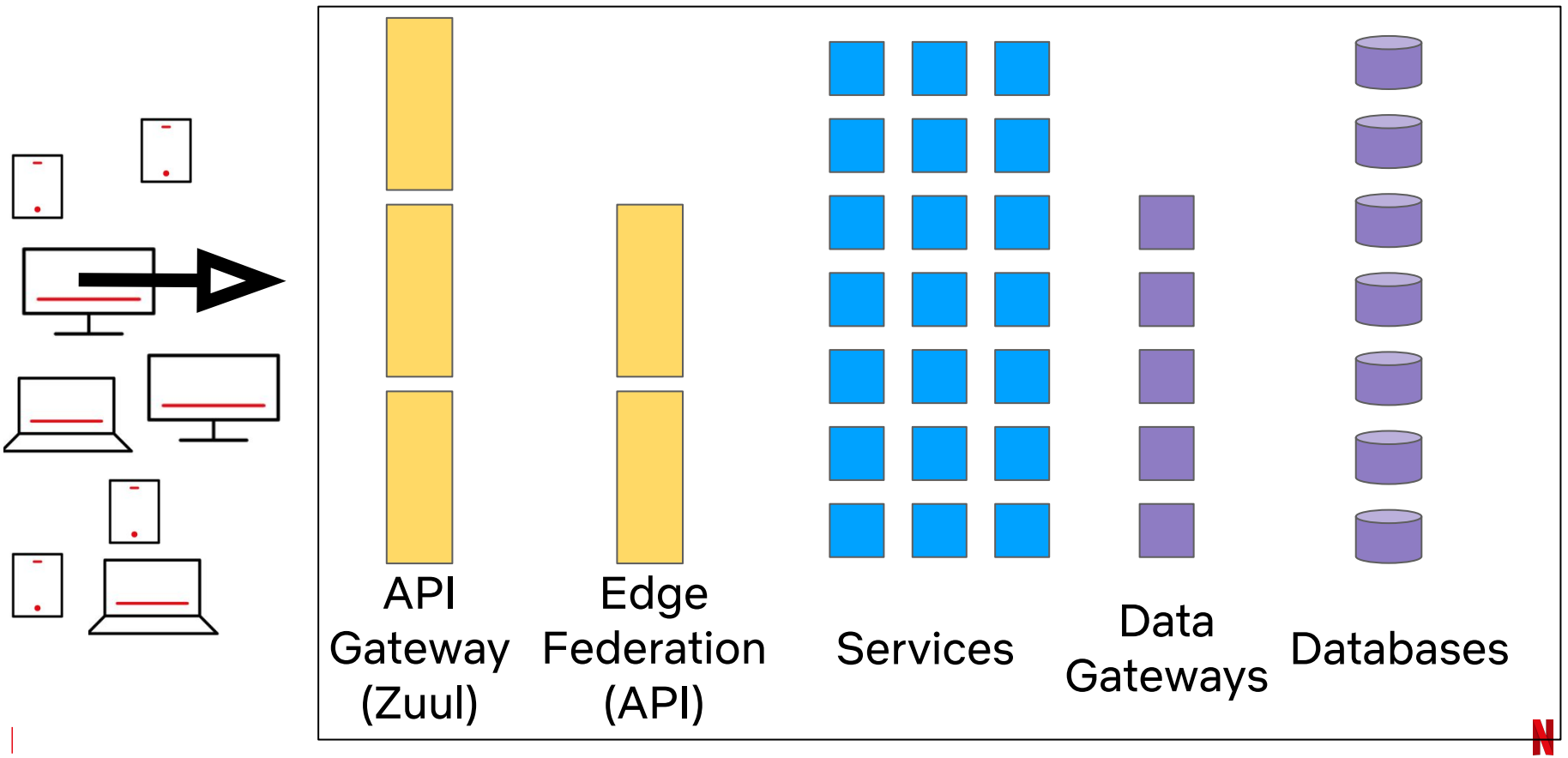
Understand Demand

Workload Profile ✅

Traffic Analysis

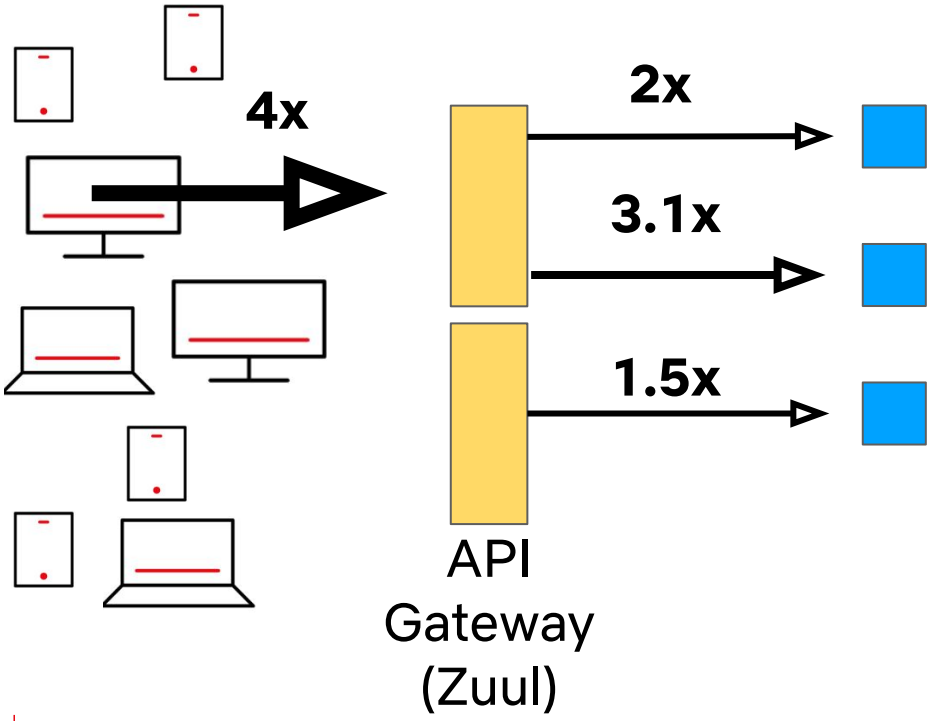


Microservices



Different for Every Service

Call Graph is not Uniform in Traffic *or* Criticality



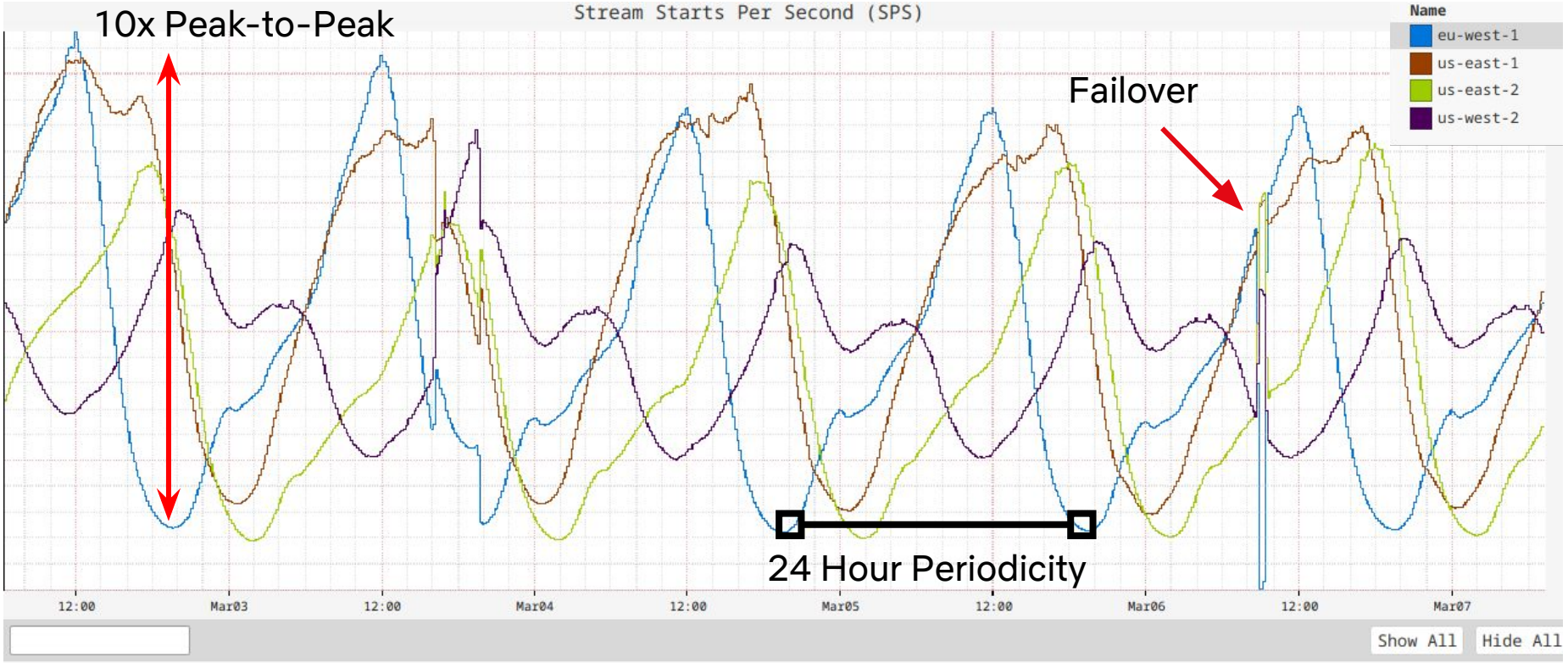
Tier 0 to Streaming Playback

Tier 1 to Streaming Discovery

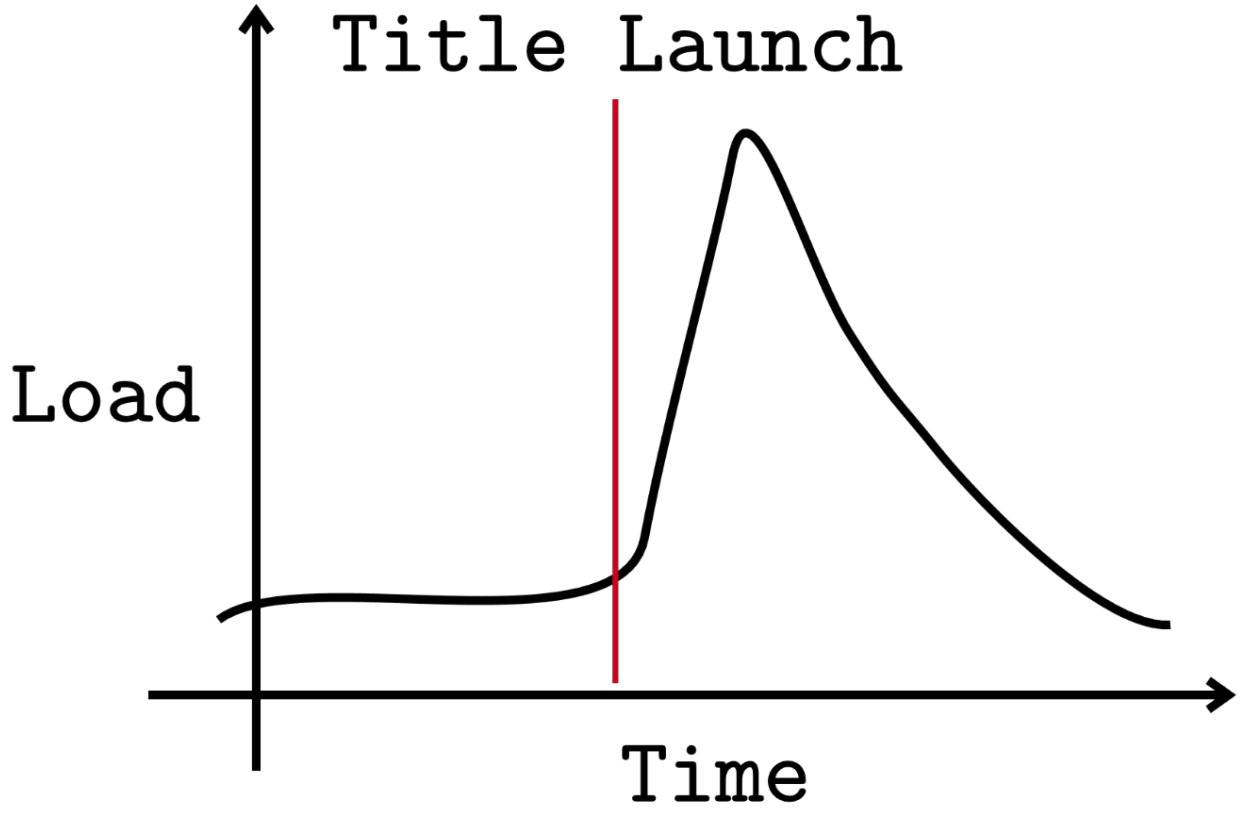
Tier 3 to Personalization

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

Variable Starts-Per-Second (SPS) Load



Content Driven Demand



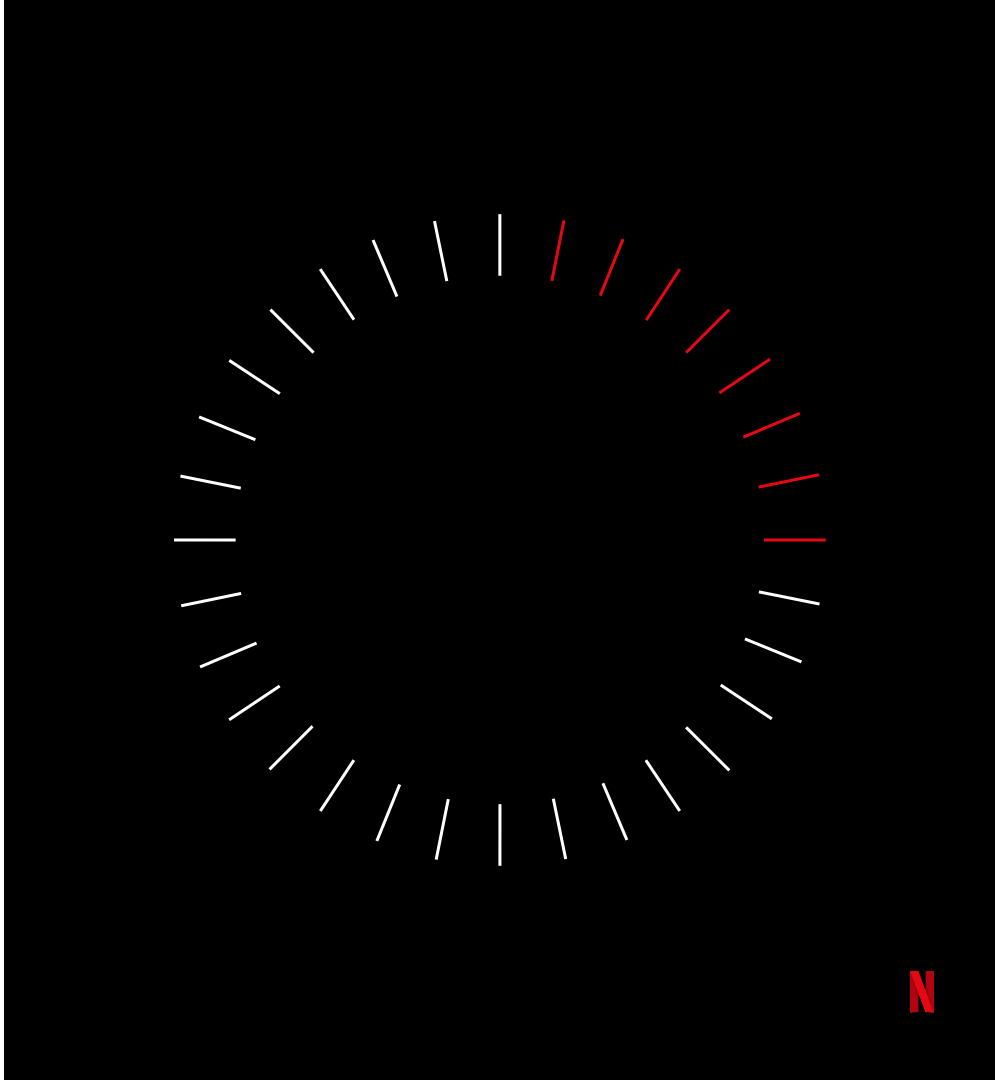
Understand
Hardware Supply

Understand
Software Demand

Balance
Manage Supply
Manage Demand

Tenets

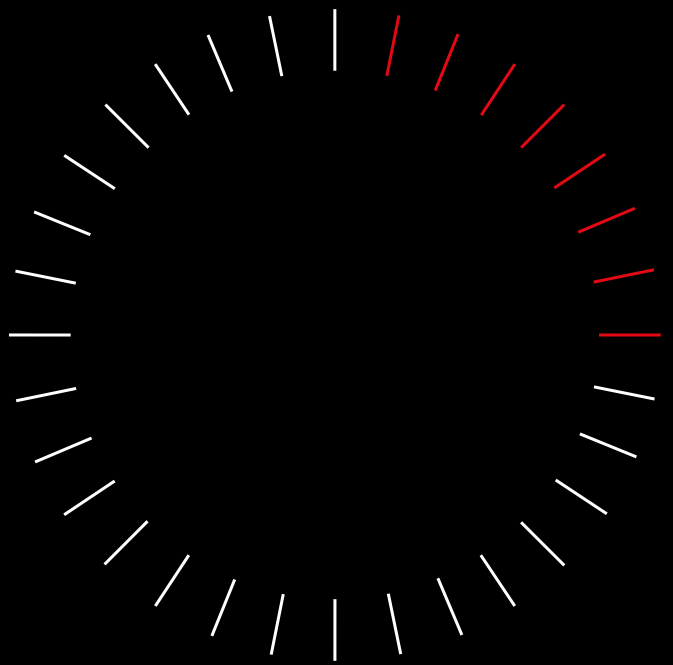
Global vs local optimization



Tenets

Global vs local optimization ✅

Ensure Buffers exist for **critical** services, when **it matters**

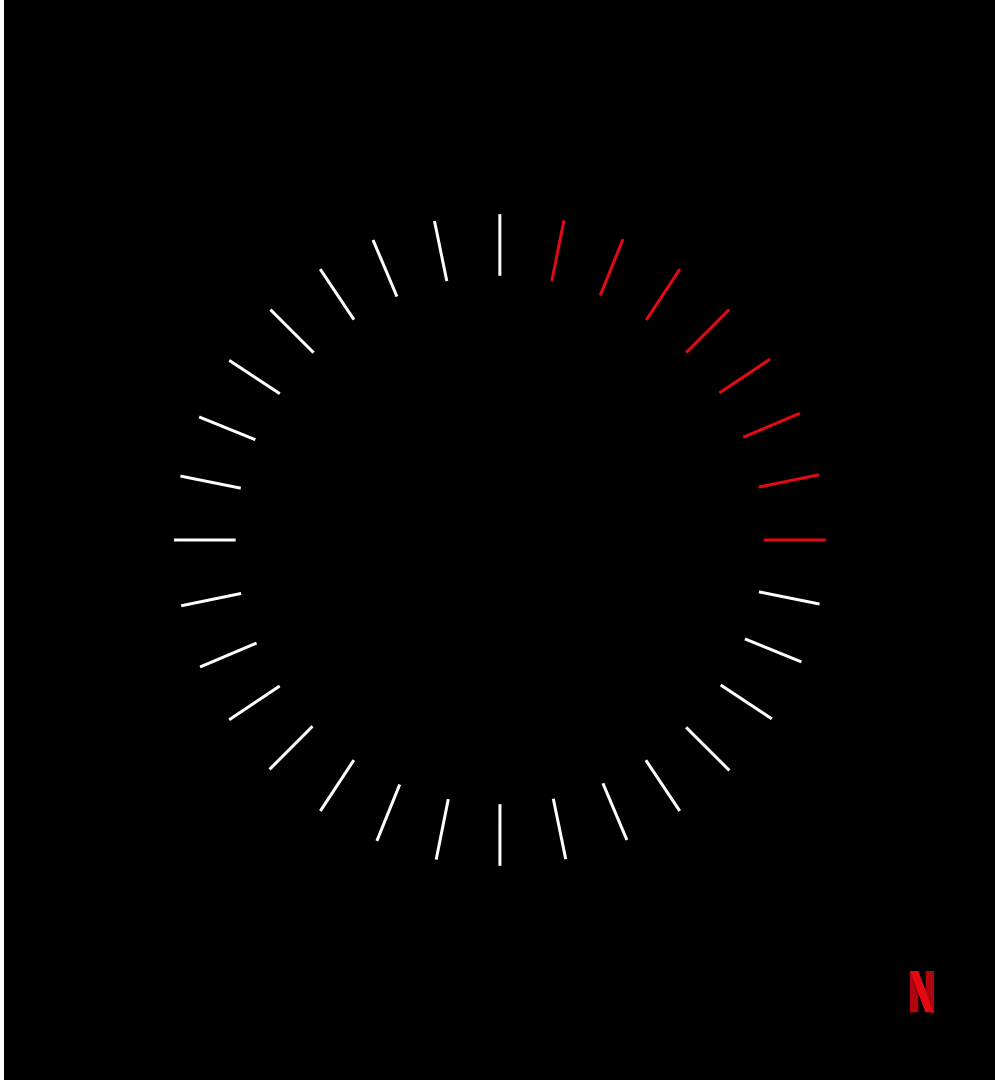


Tenets

Global vs local optimization ✅

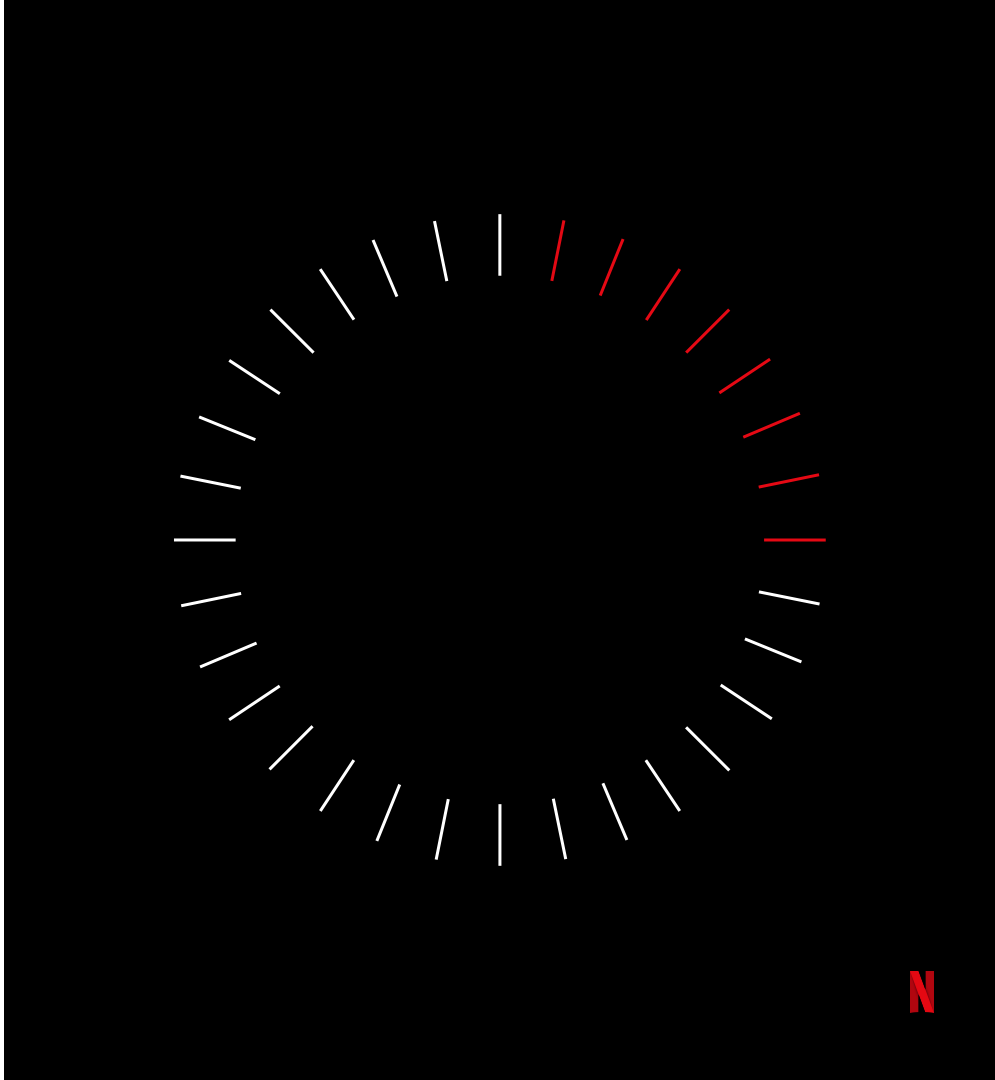
Buffers exist for **critical** services, when **it matters** ✅

Managed Supply and Demand

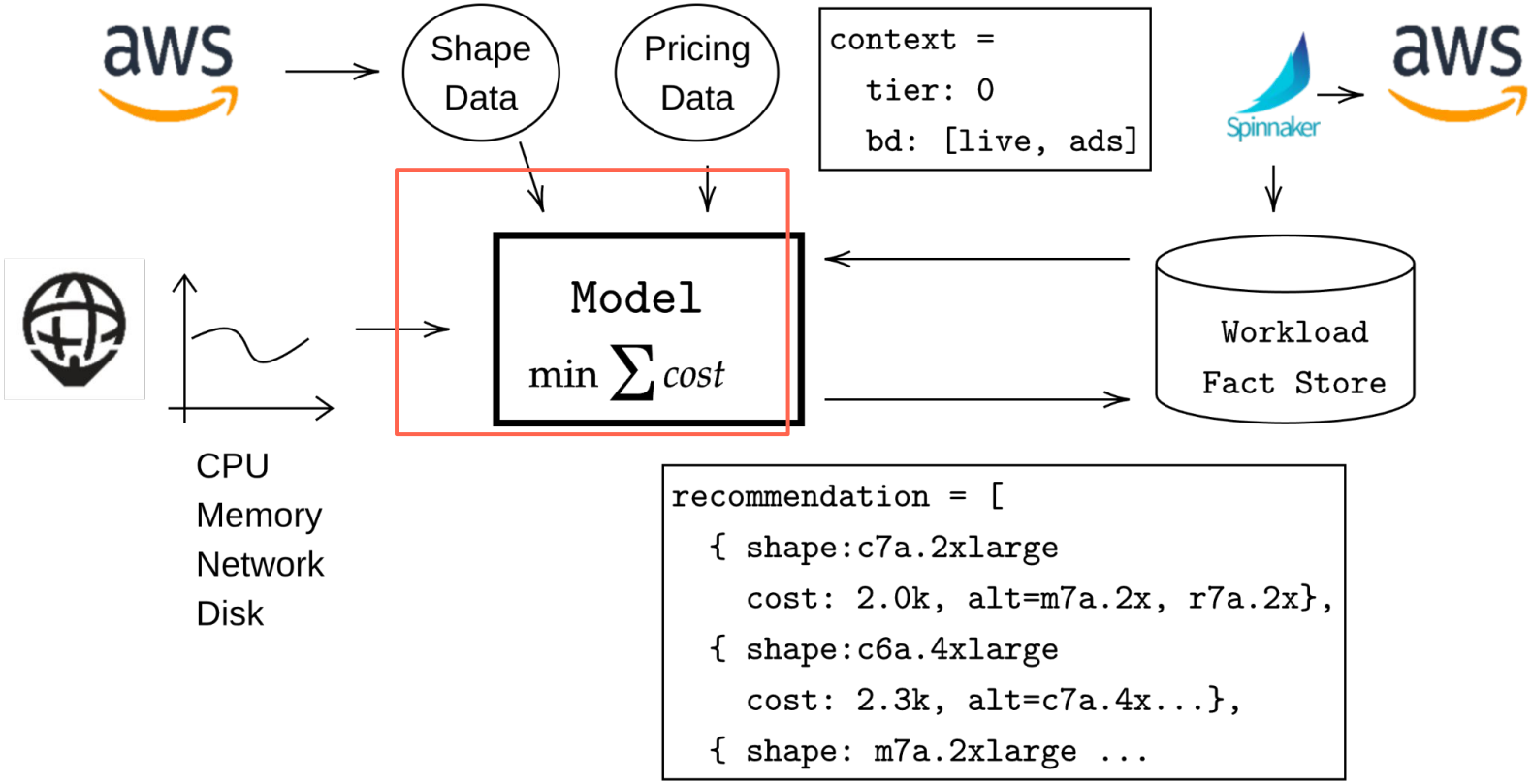


Manage Supply

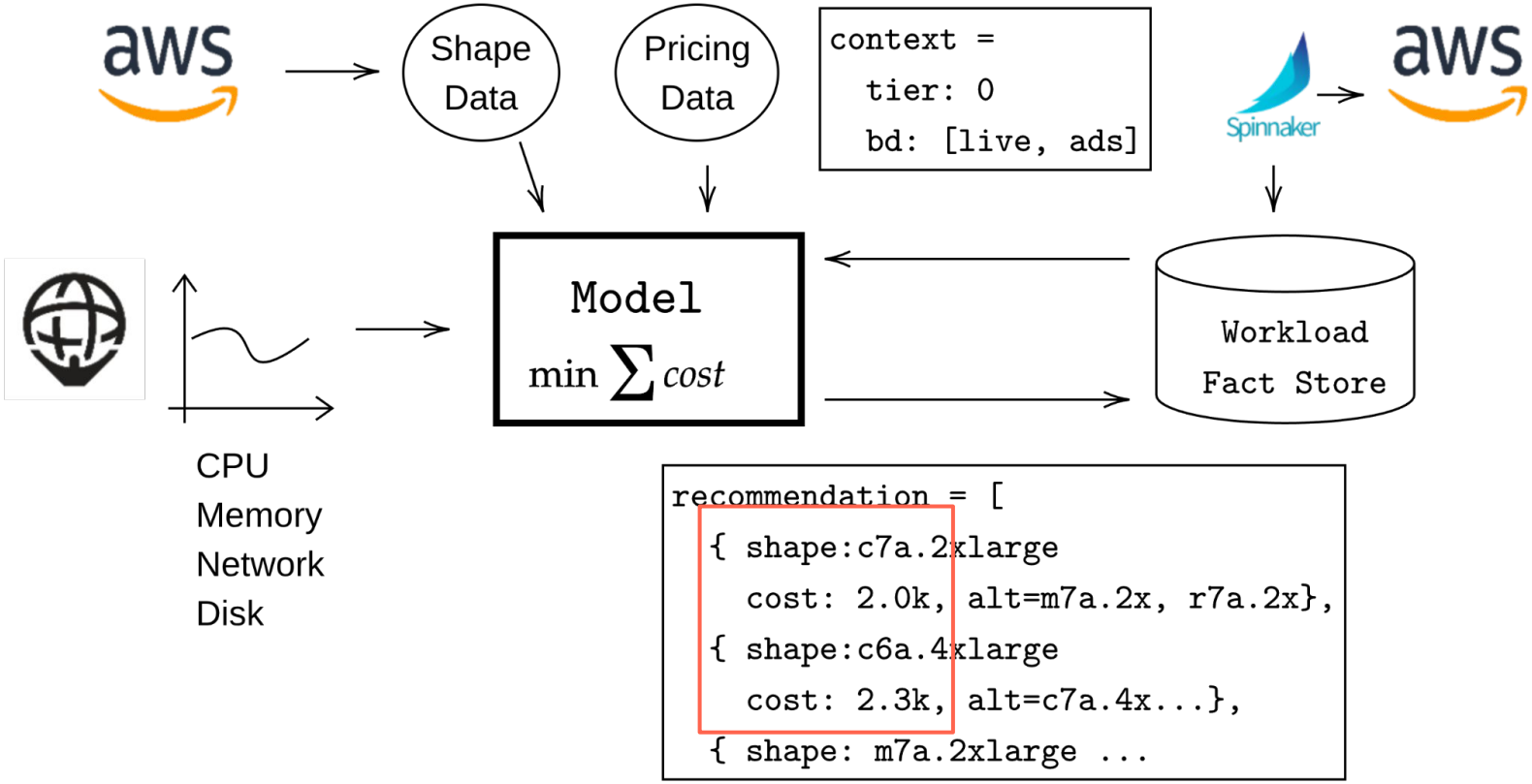
Hardware 🤝 Workload



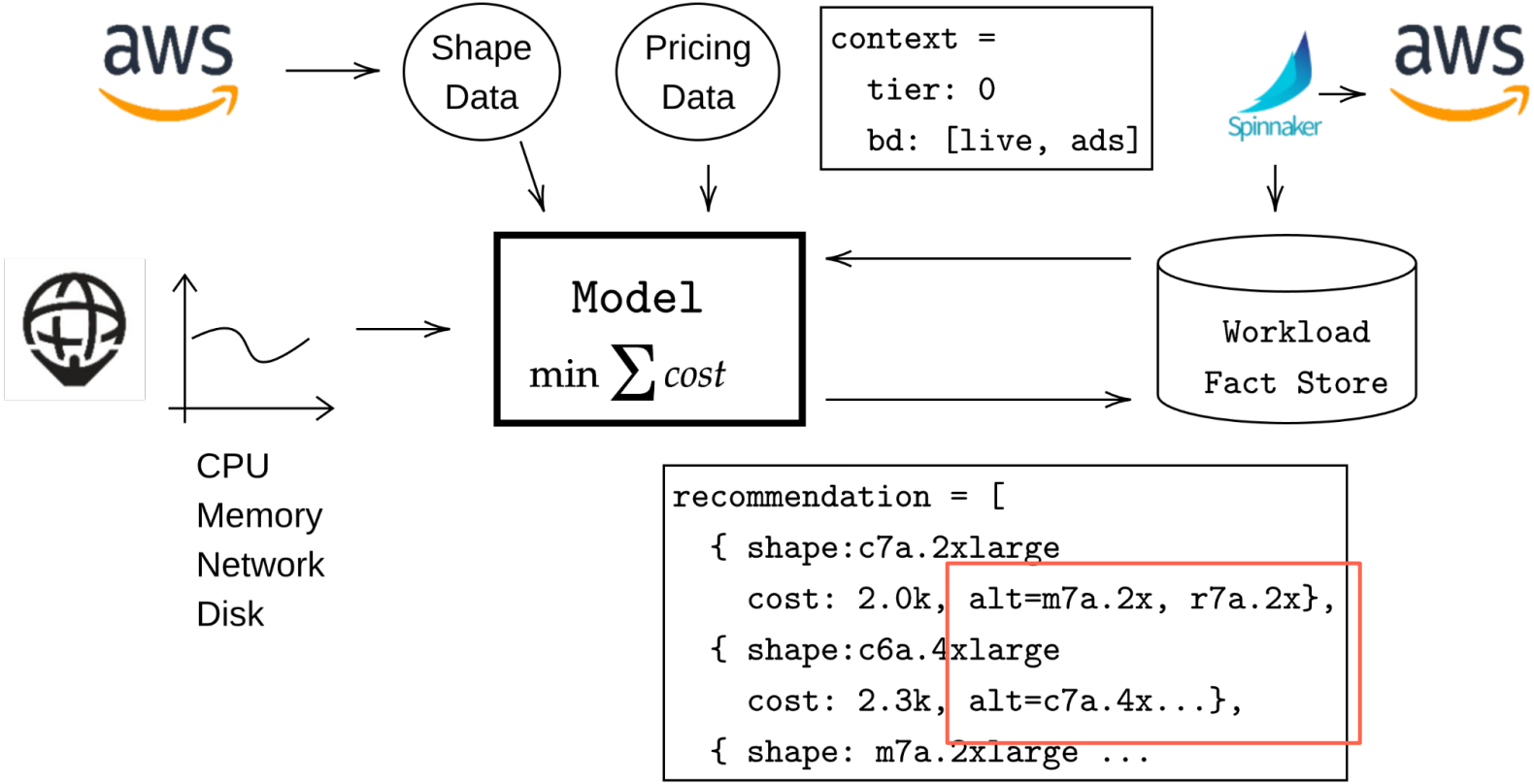
Hardware Selection



Hardware Selection



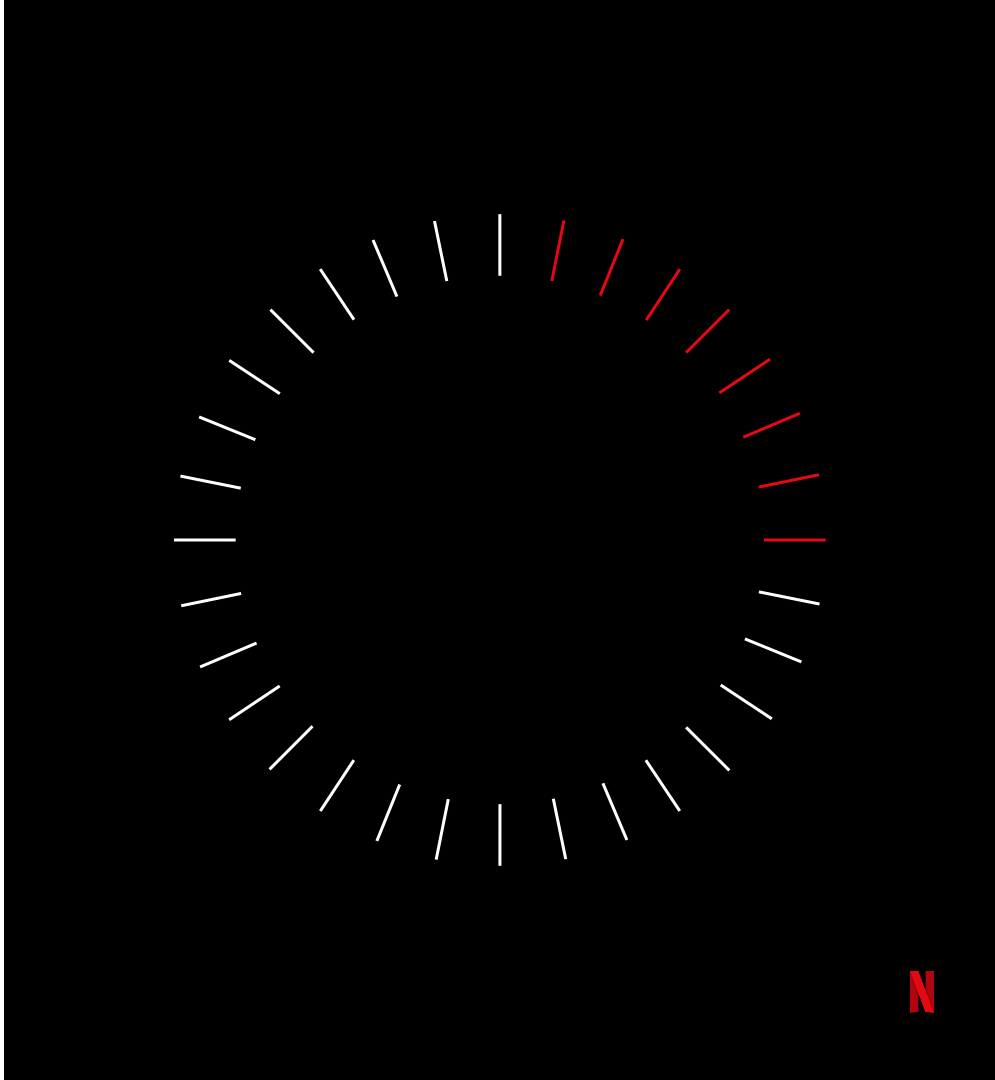
Hardware Selection



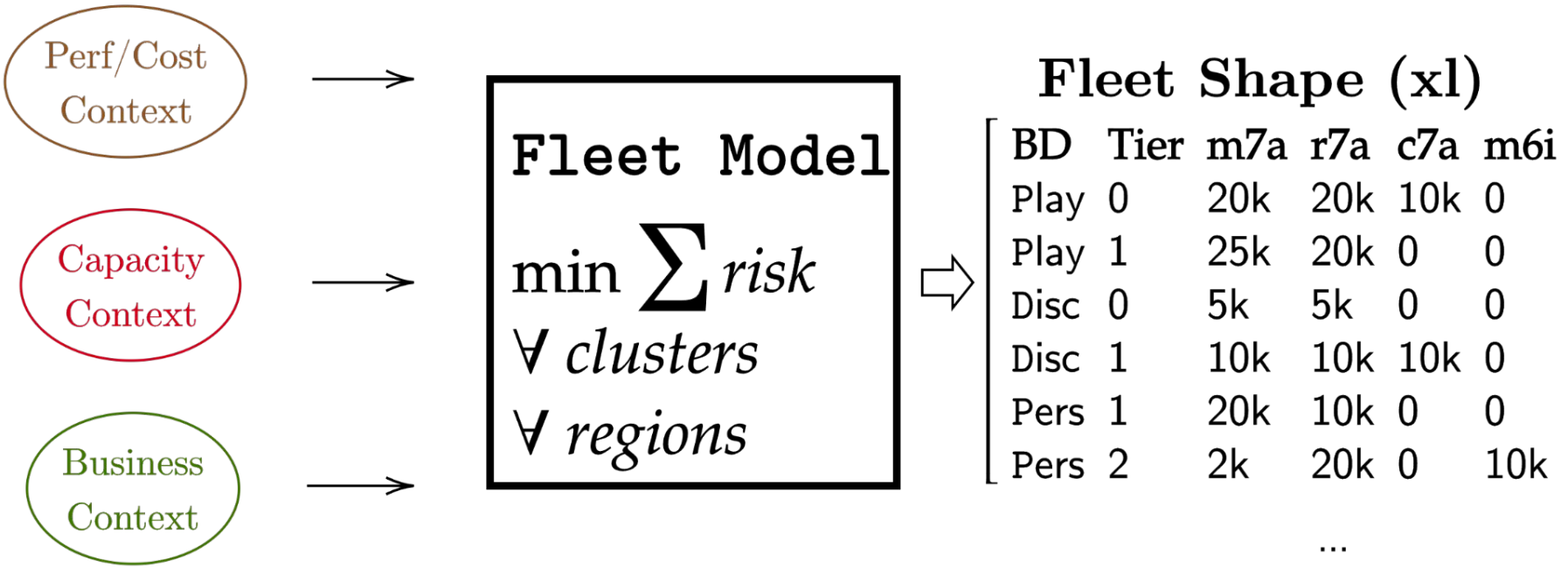
Manage Supply

Hardware 🤝 Workload ✅

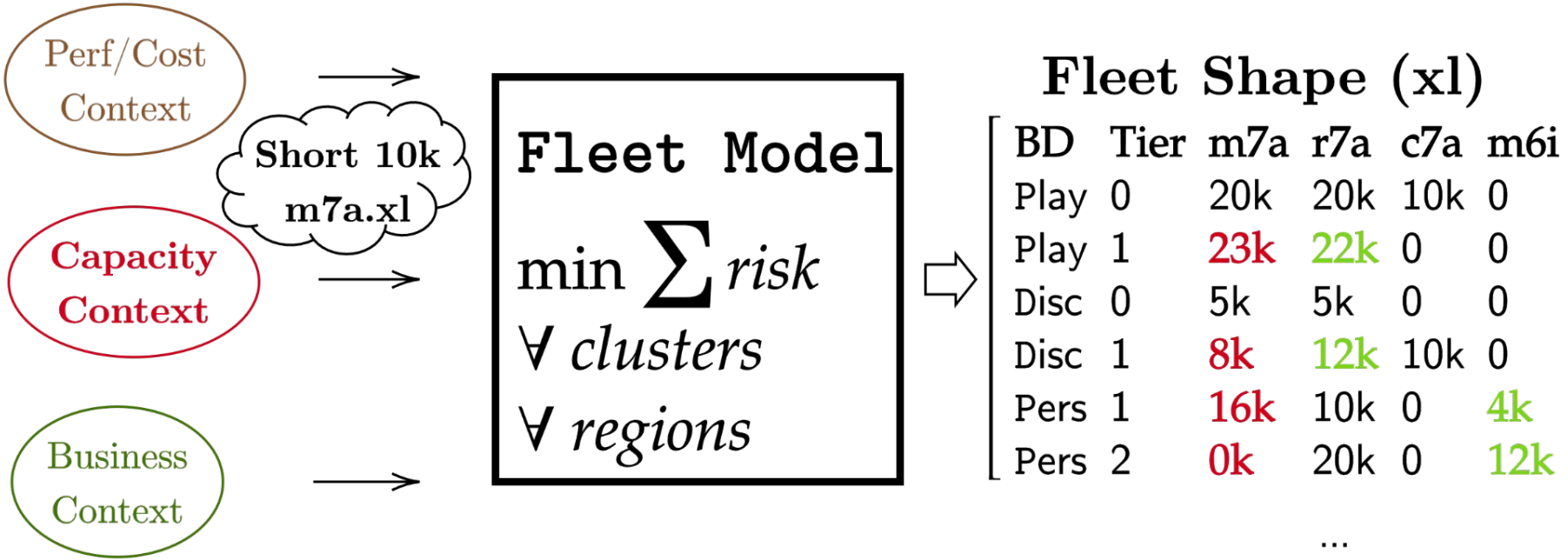
Fleet **Shaping**

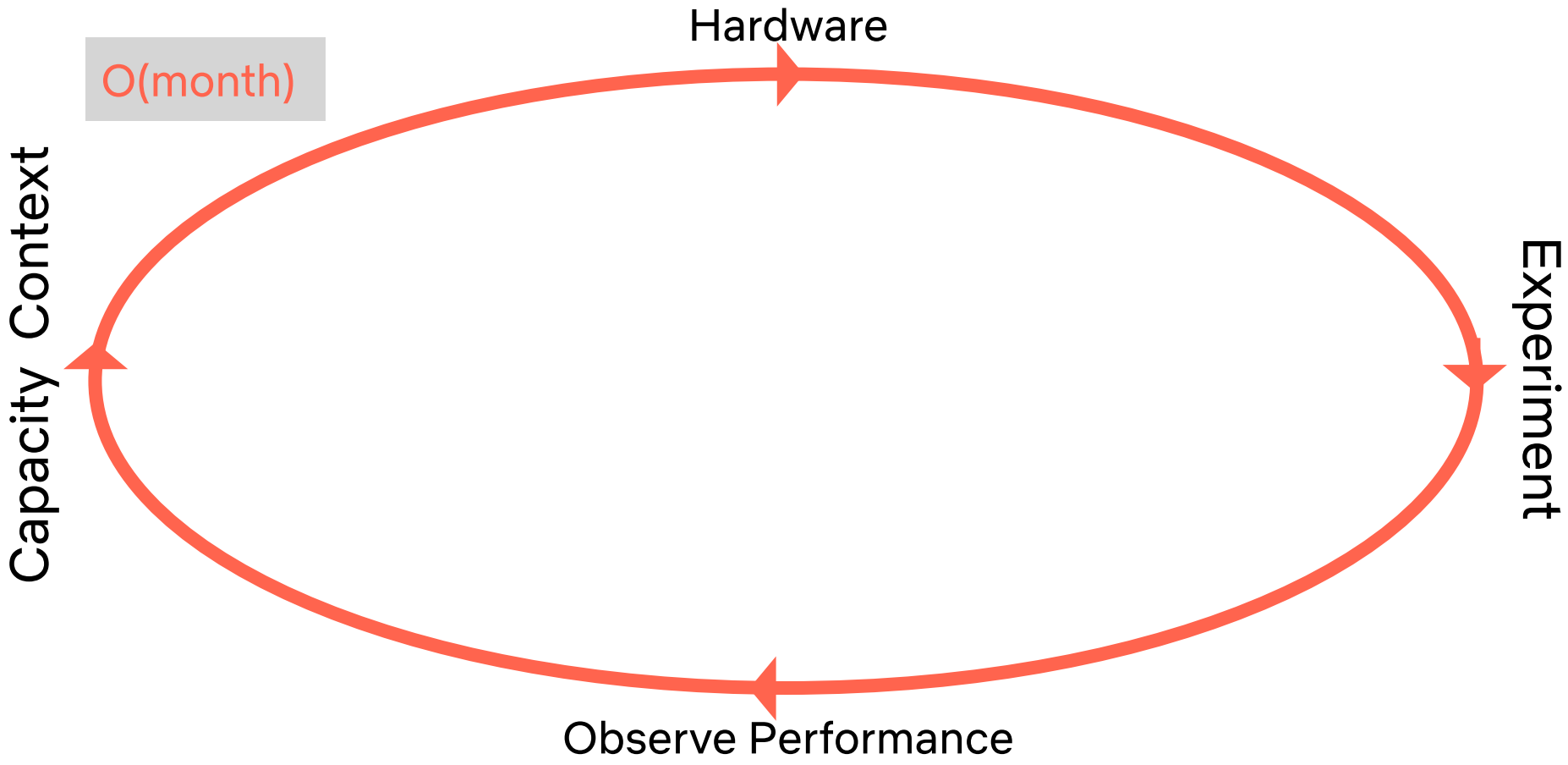


Fleet Shaping



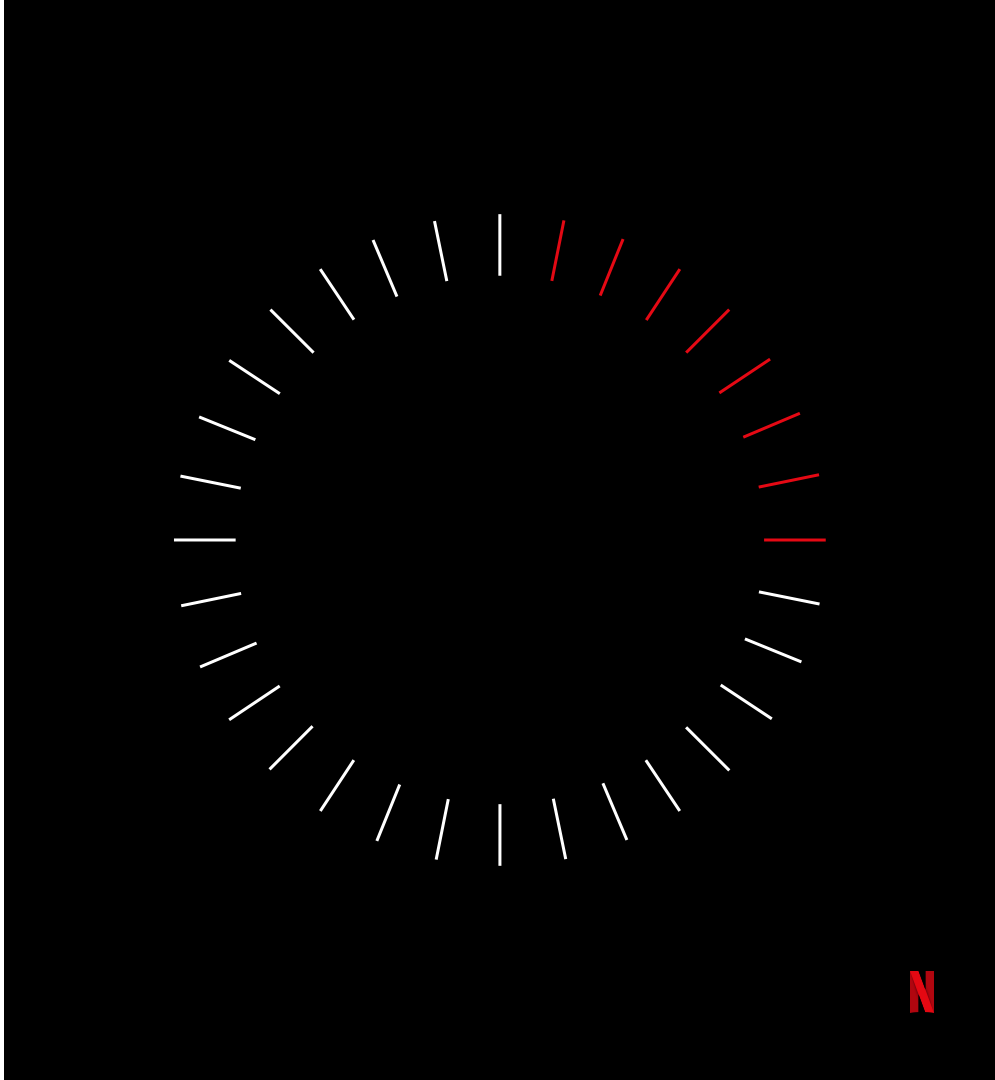
Fleet Shaping




O(month)
Hardware
Capacity
Context
Experiment
Observe Performance

Manage Demand

Pre-scaling

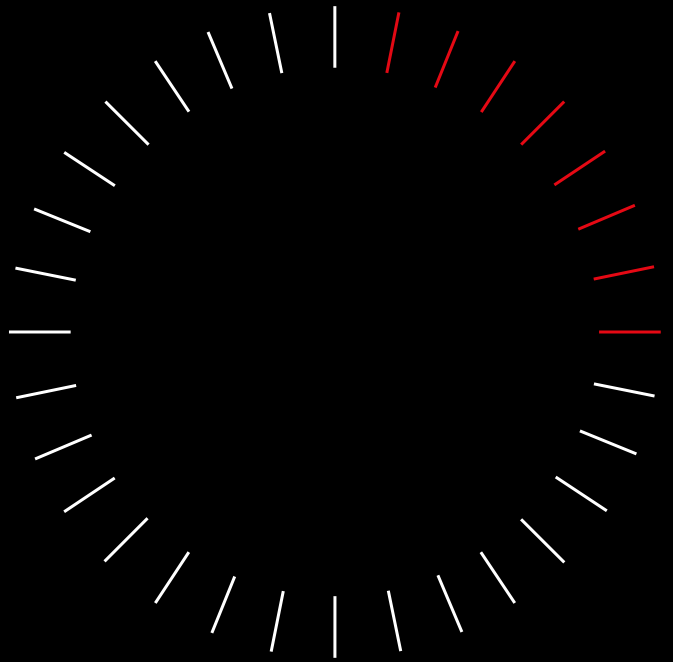


Pre-scaling

Prediction models 🔮

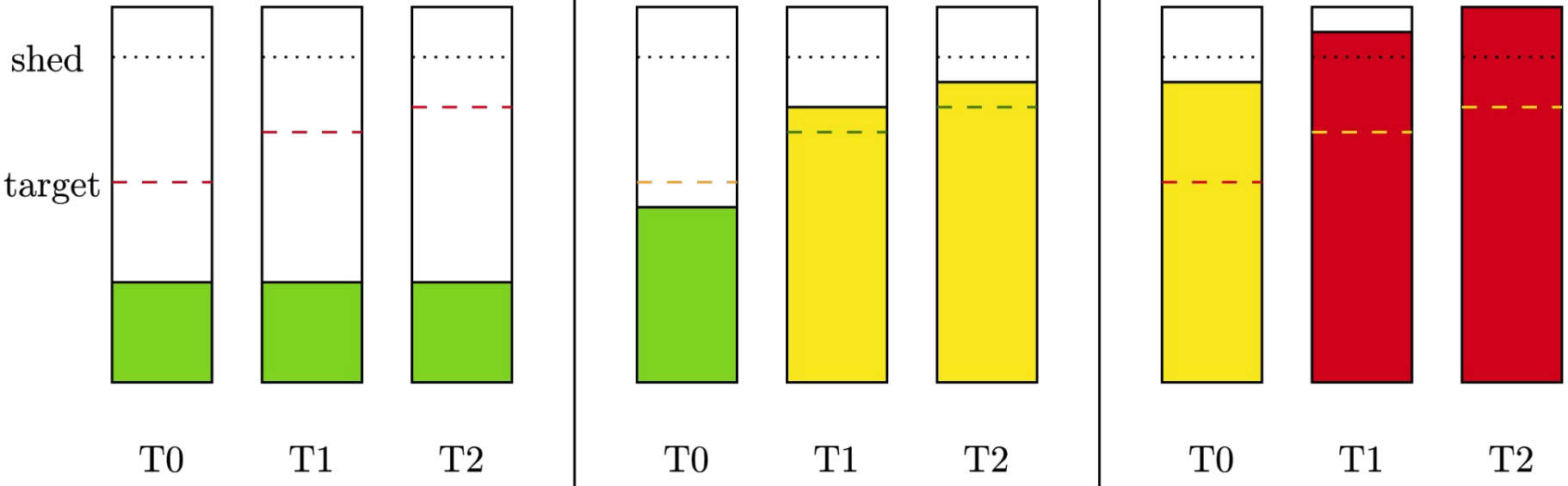
Peak load estimation (viewers -> traffic)

Automated fleet scaleup 🤖



Pre-Scale for Load

Fleet Prescale

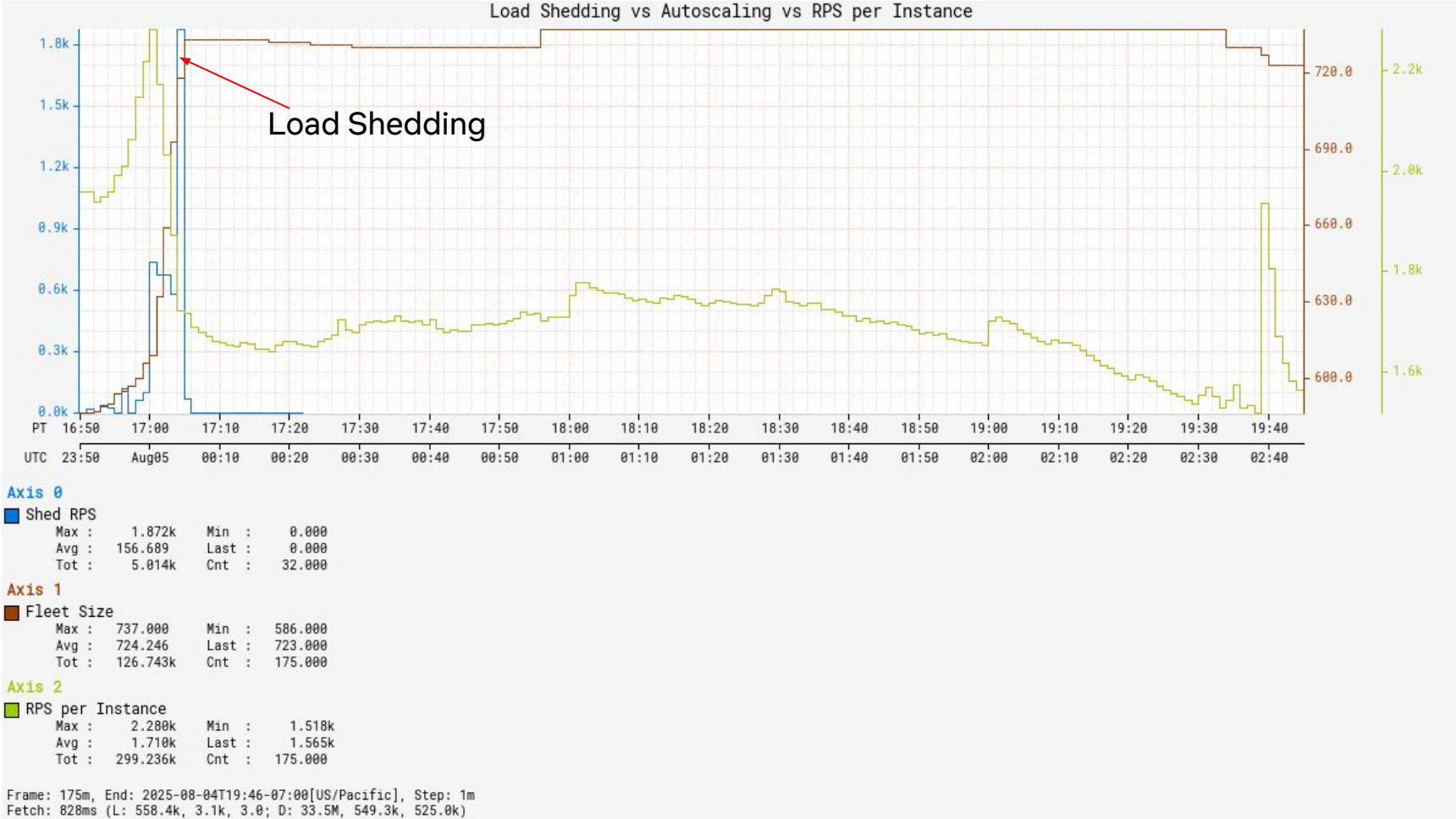


Inefficient: Overscaled

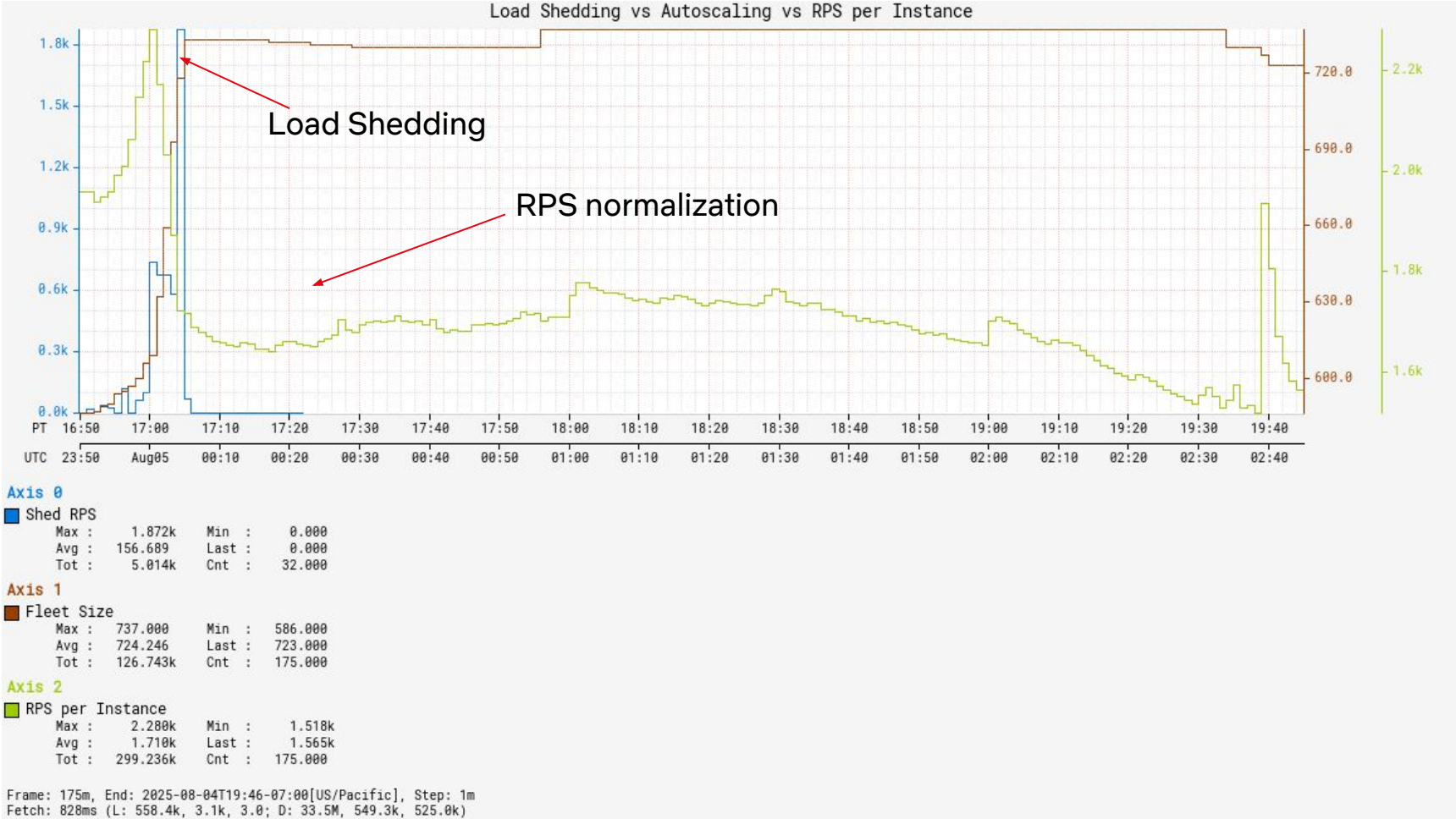
Efficient: Balance

Probably Underscaled

Fast Recovery Mitigates Pre-Scale Risk



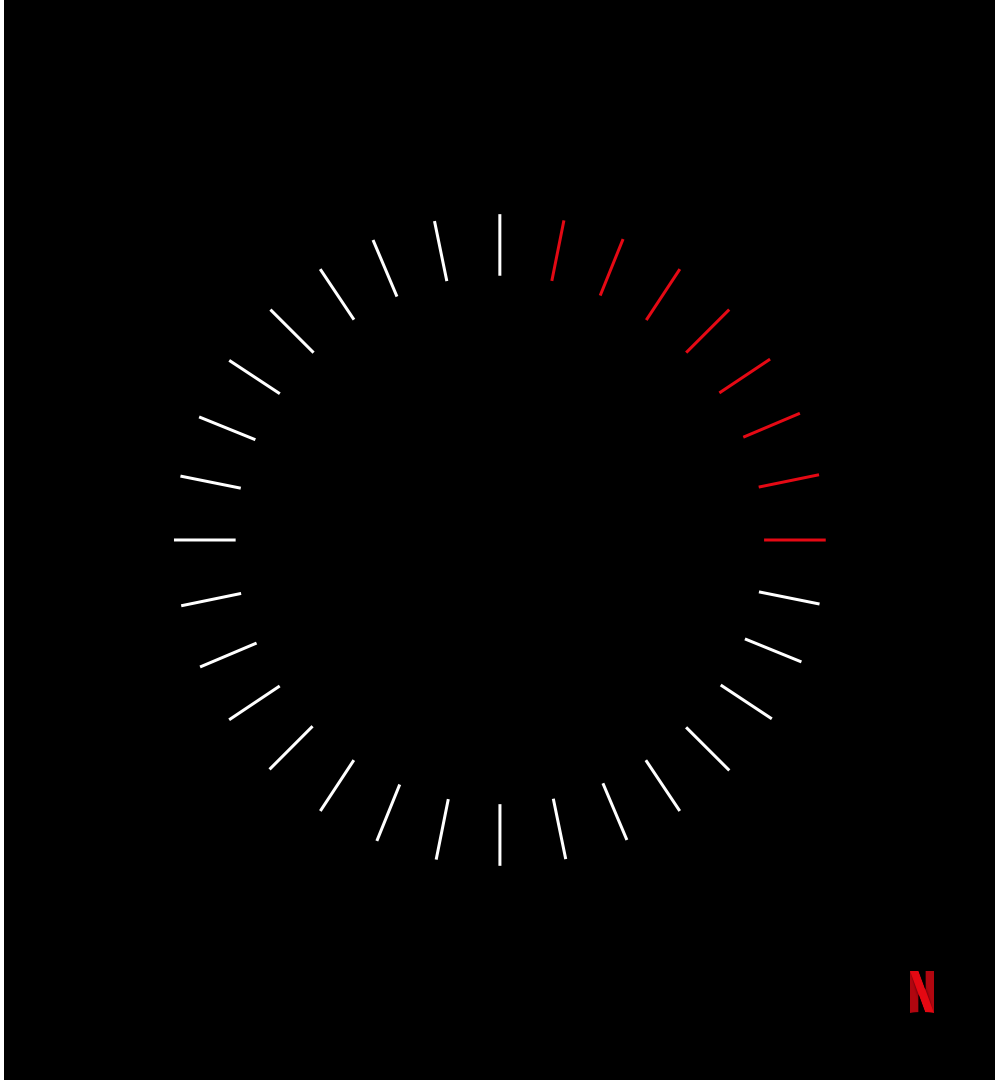
Fast Recovery Mitigates Pre-Scale Risk



Manage Demand

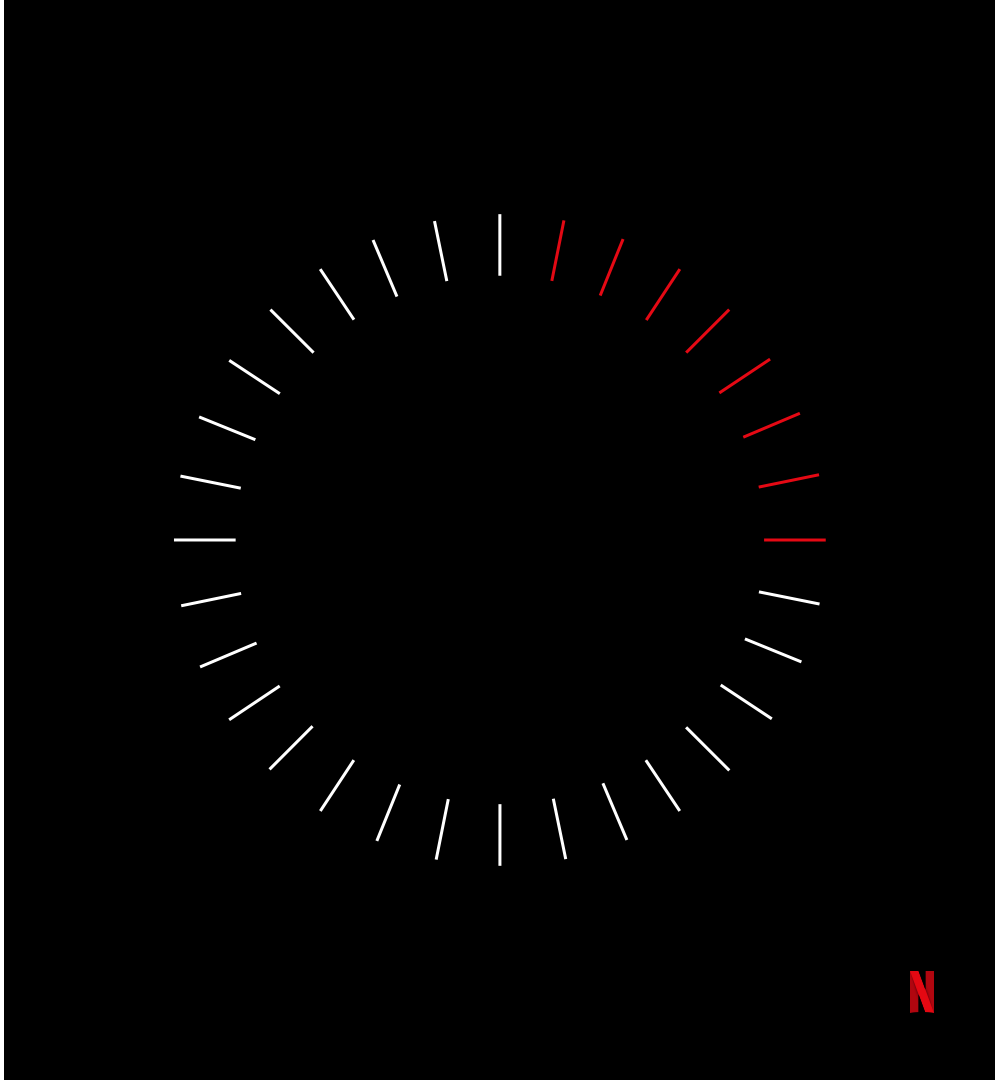
Pre-scaling ✅

Dynamic Traffic Shaping

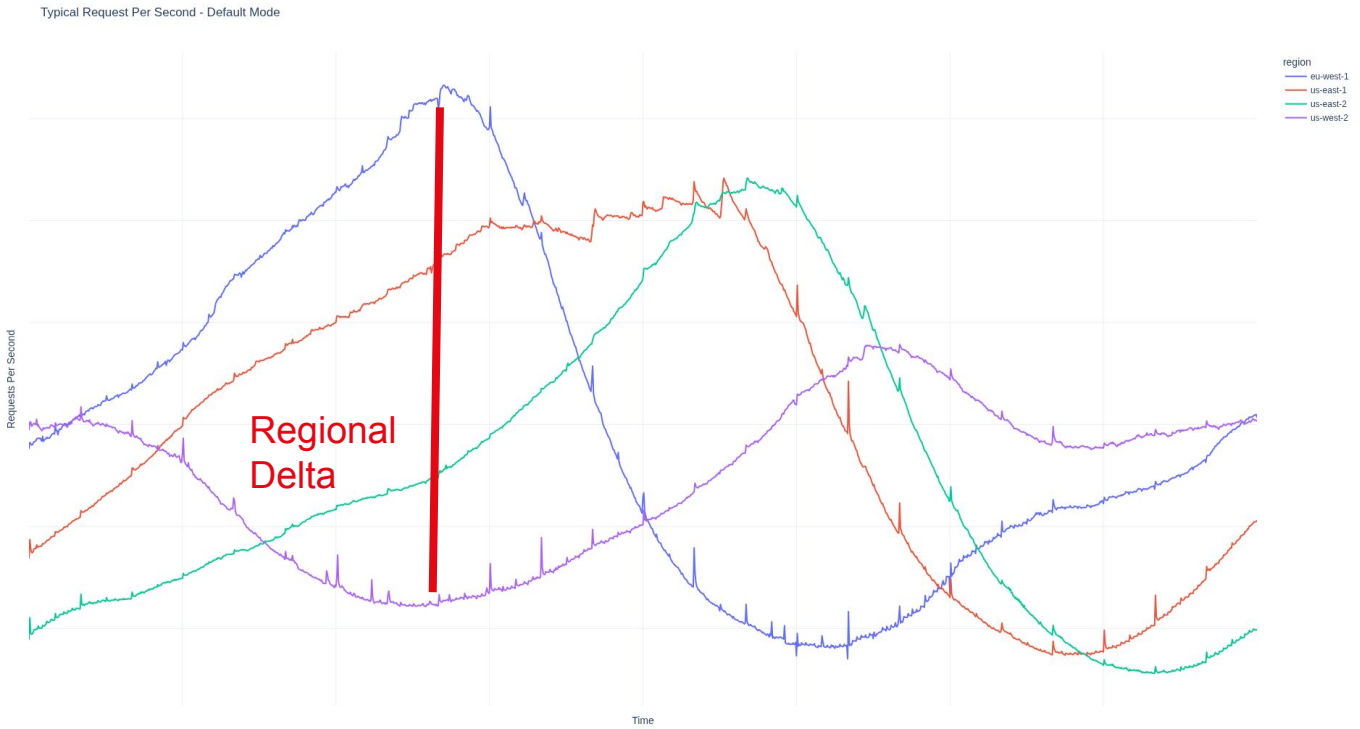


Dynamic Traffic Shaping

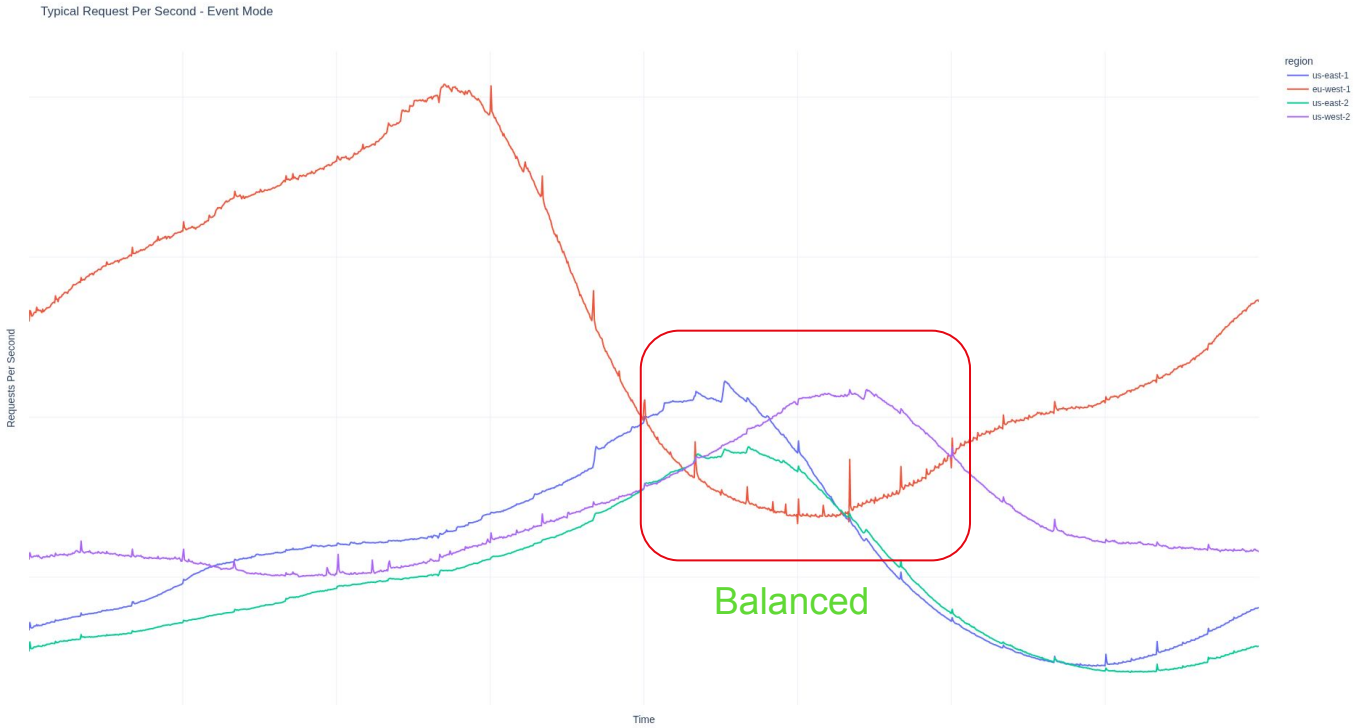
Re-distribute **existing** traffic



Default Shape



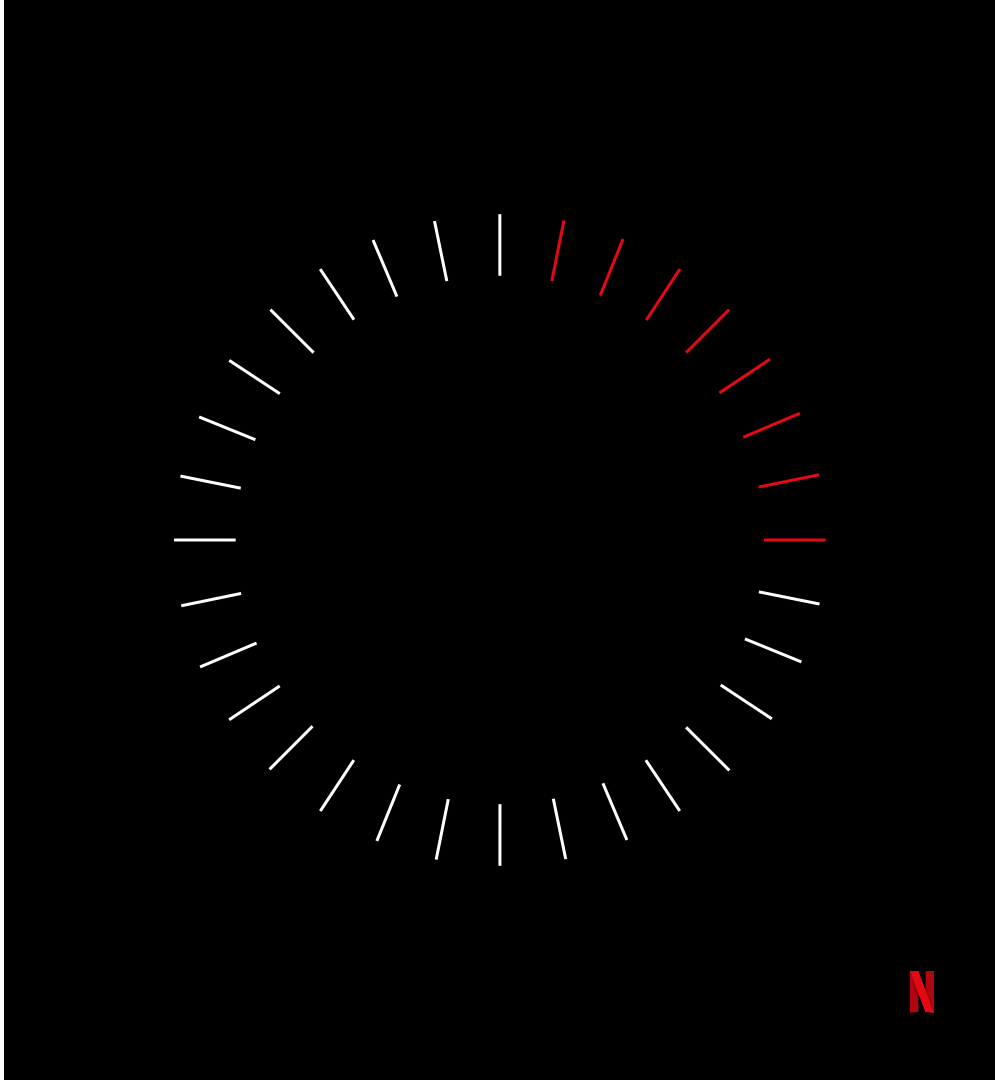
Shaping in action



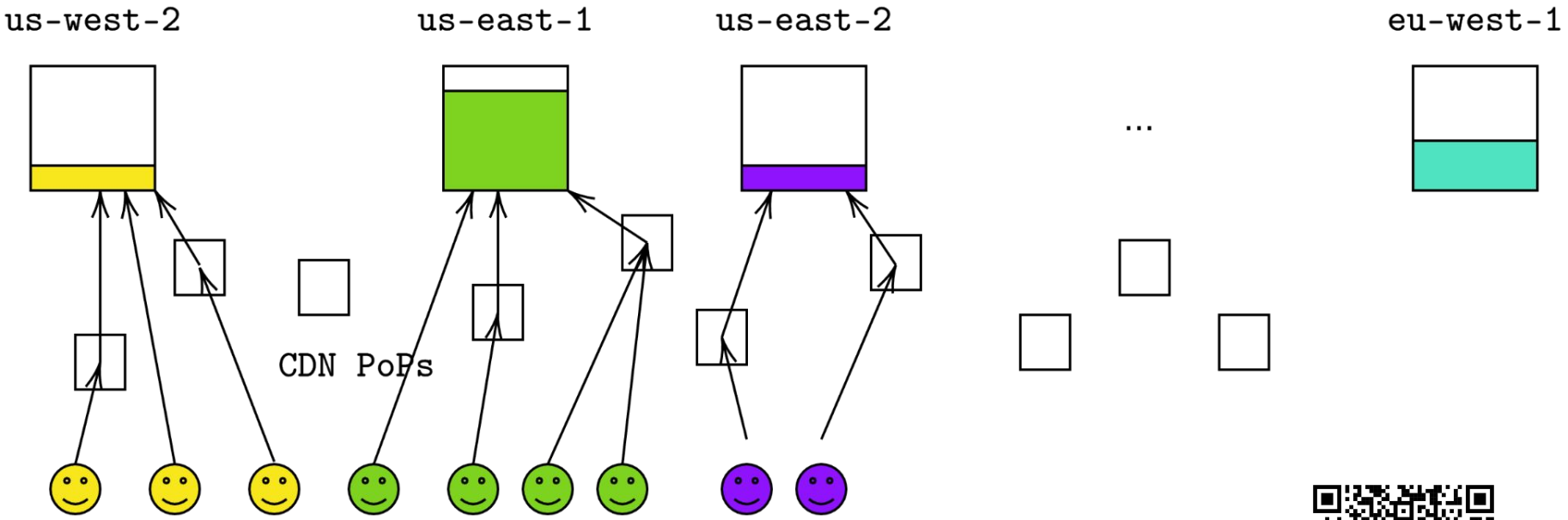
Dynamic Traffic Shaping

Re-distribute **existing** traffic ✅

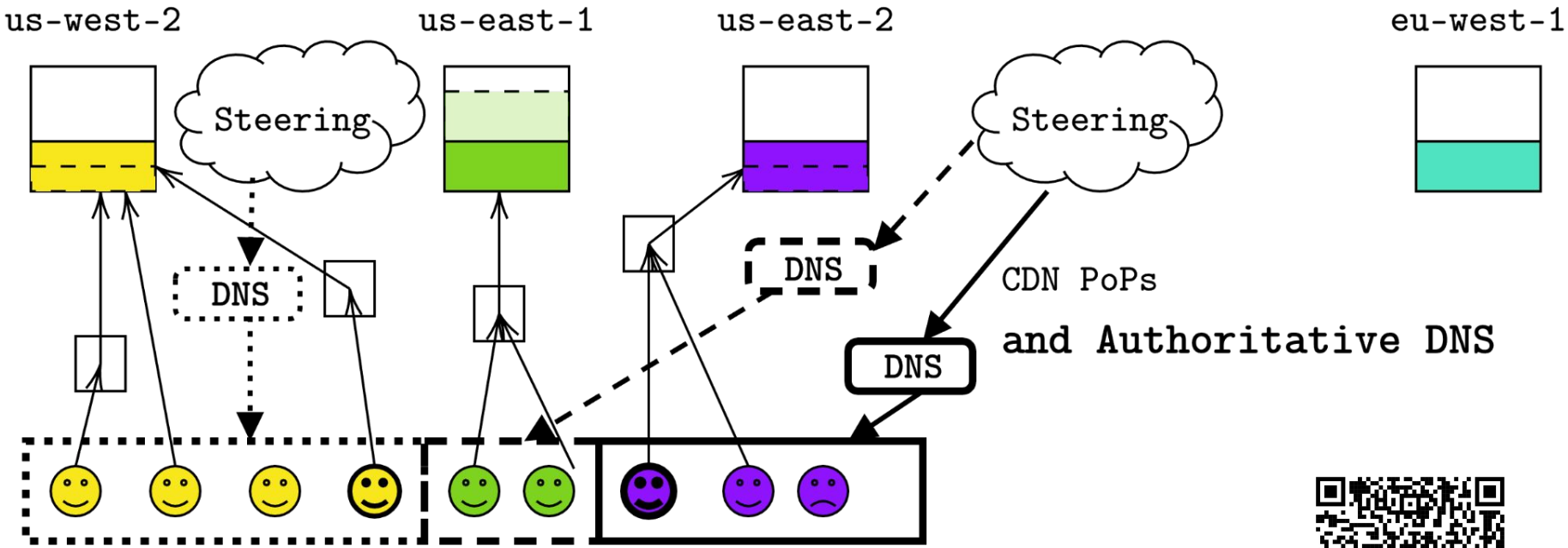
Steer **new** traffic based on latency



Netflix DNS and Steering - Typical

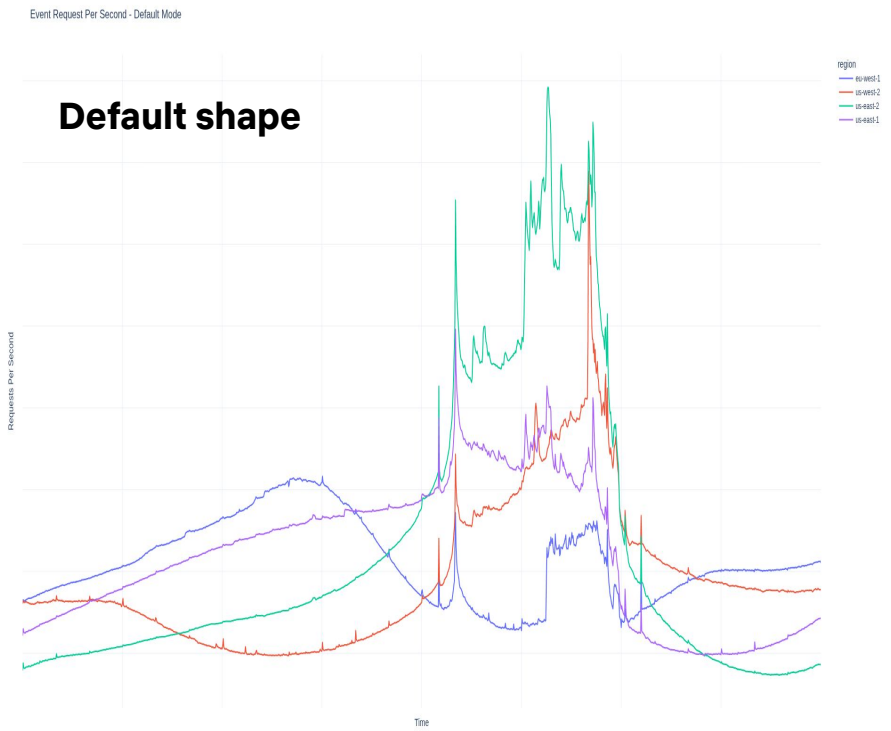


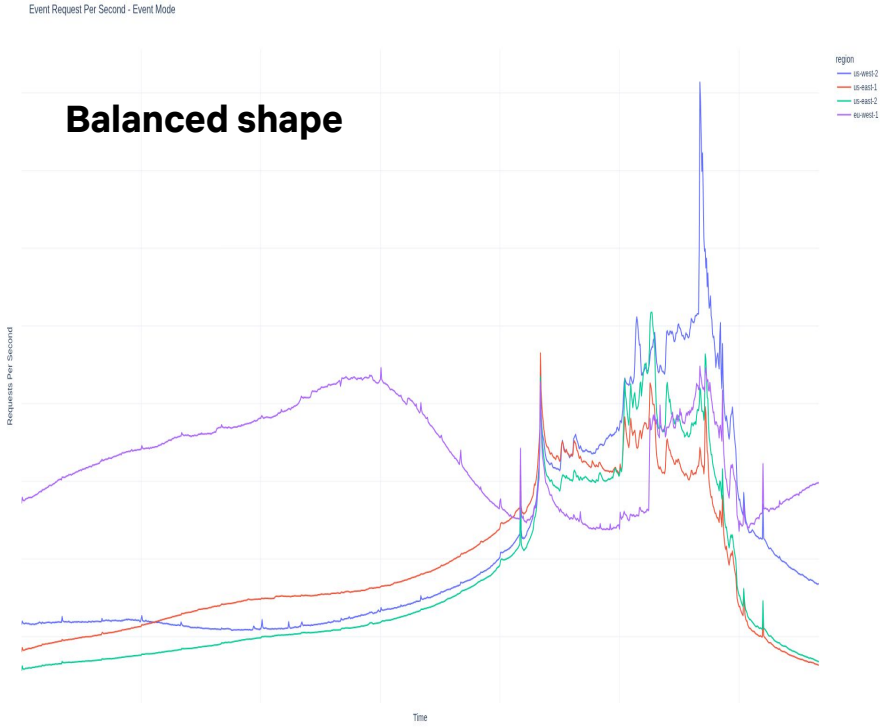
Netflix DNS and Steering - Balanced



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

Shaping effects





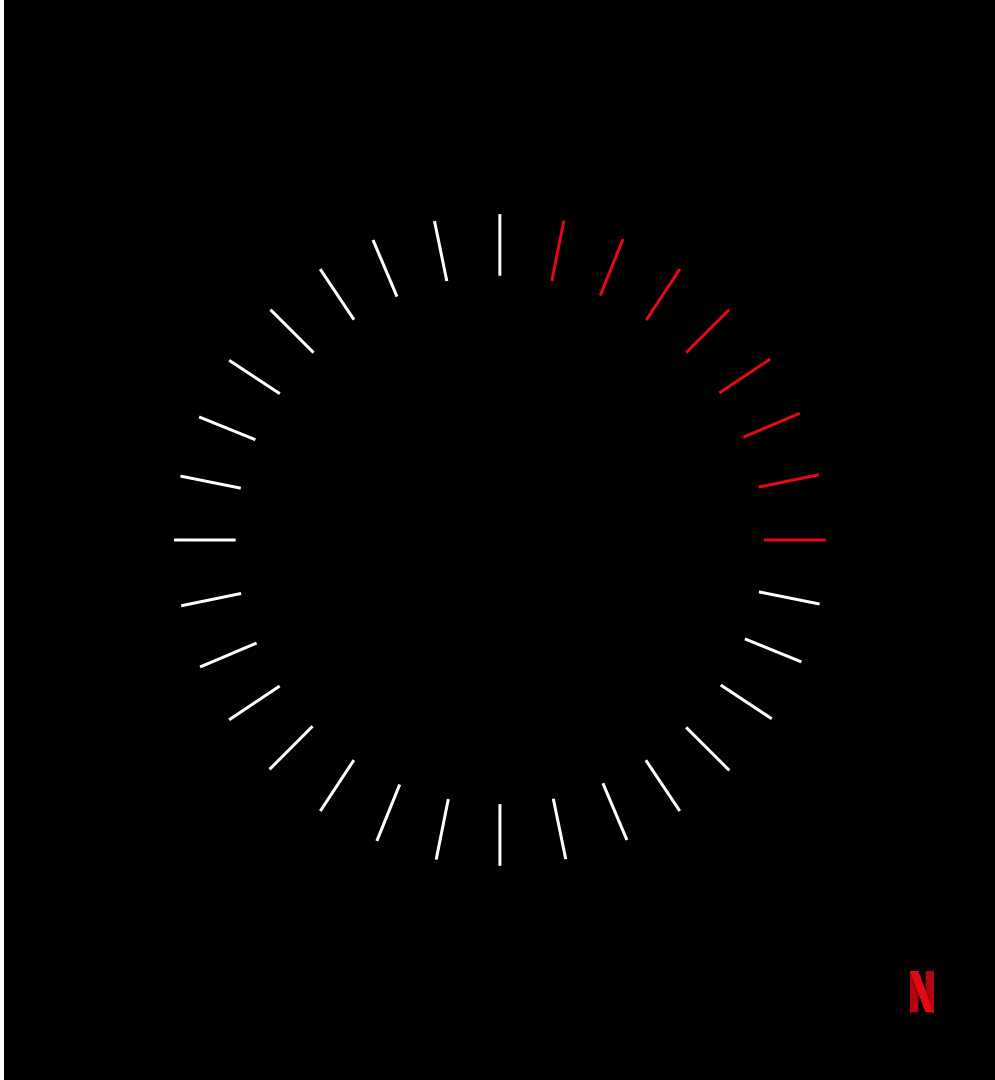
Hope is not a *Strategy*...

Manage Demand

Pre-scaling ✅

Dynamic Traffic Shaping ✅

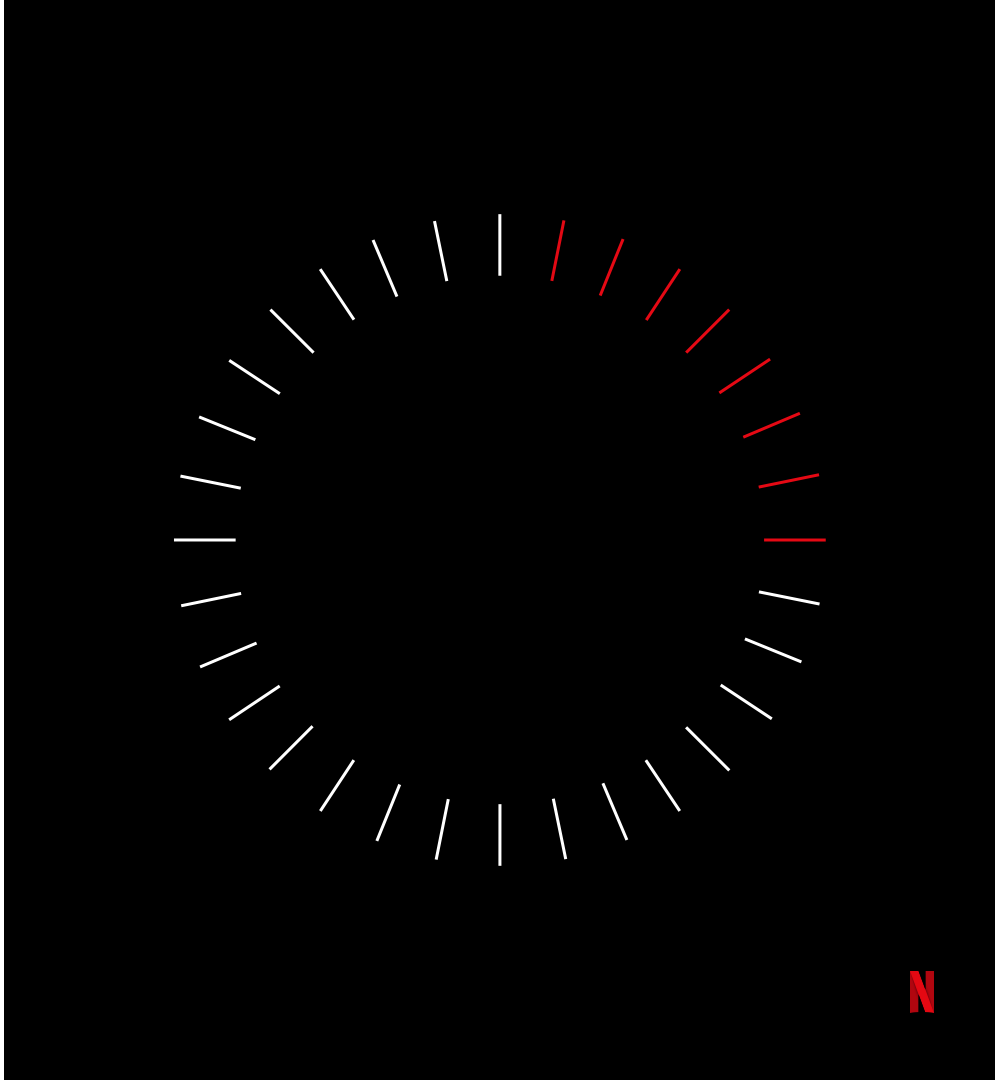
Reactive Scaling



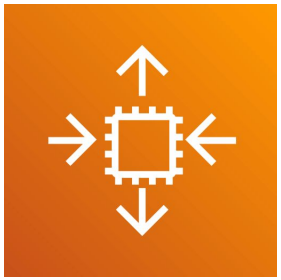
Reactive Scaling

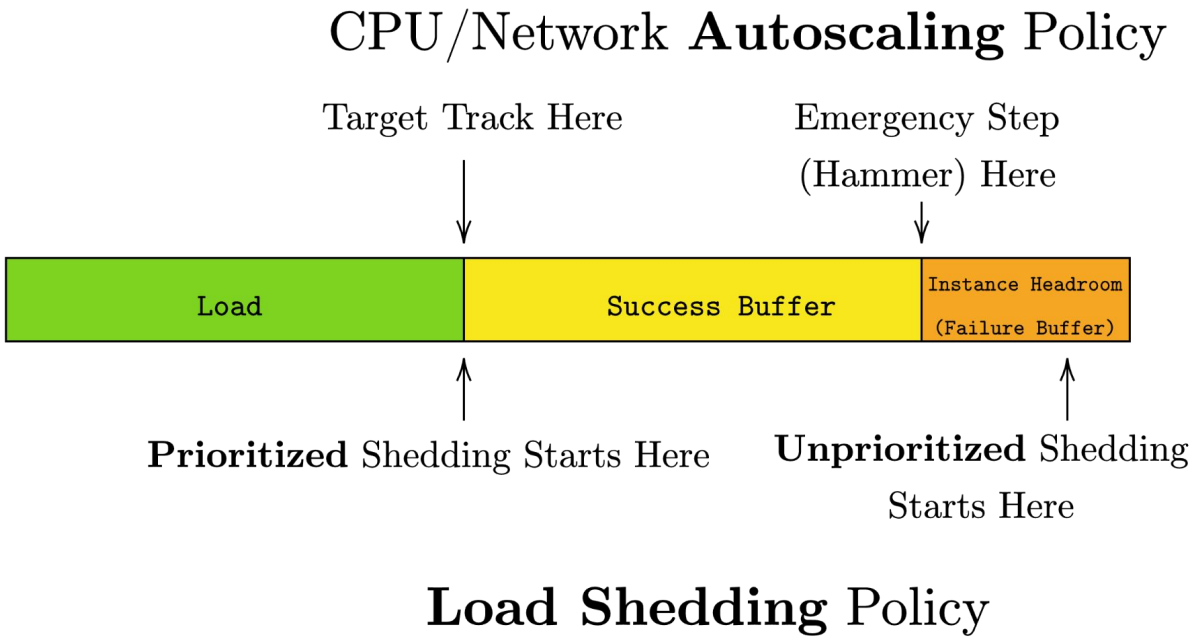
Buffers based on workload tier/domain

Mathematically derived **thresholds**

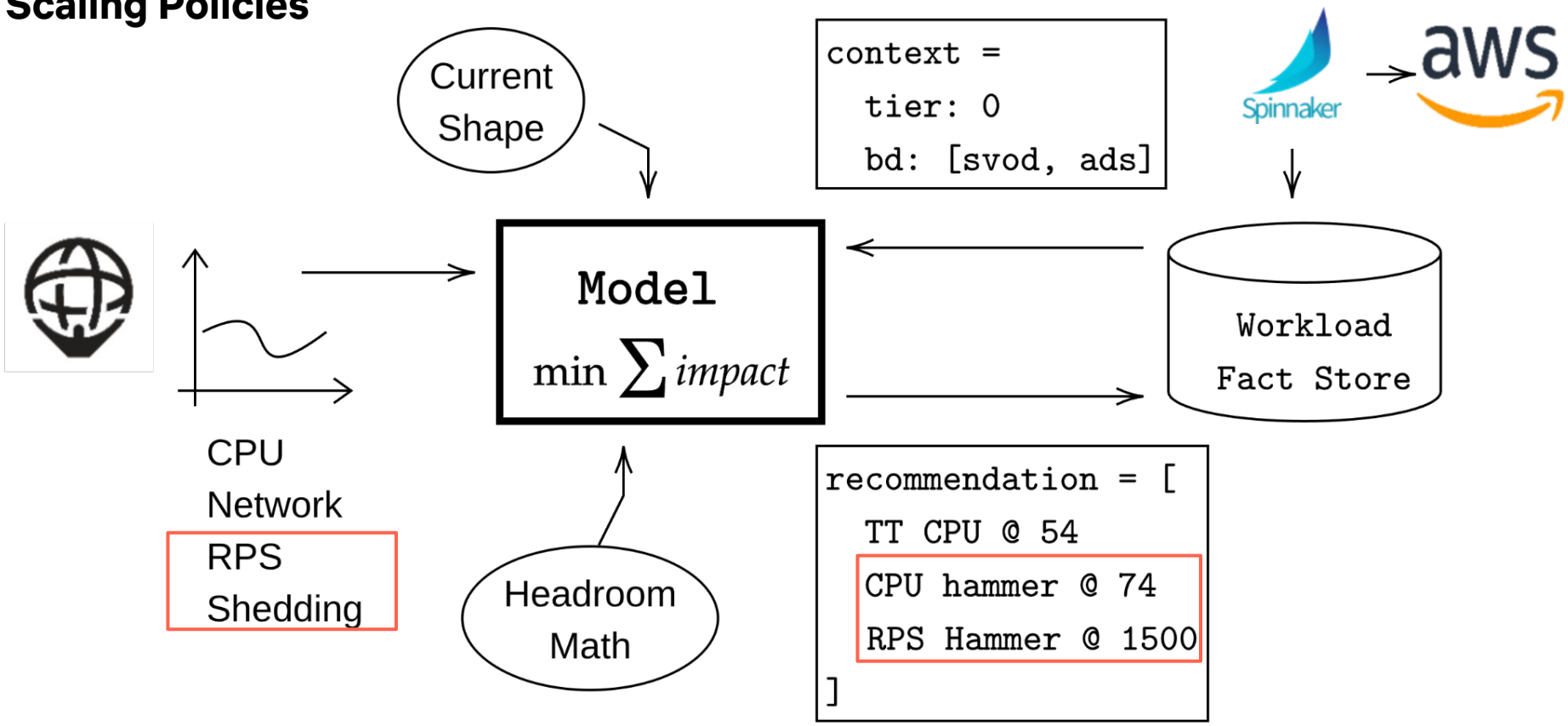


Scaling Policies

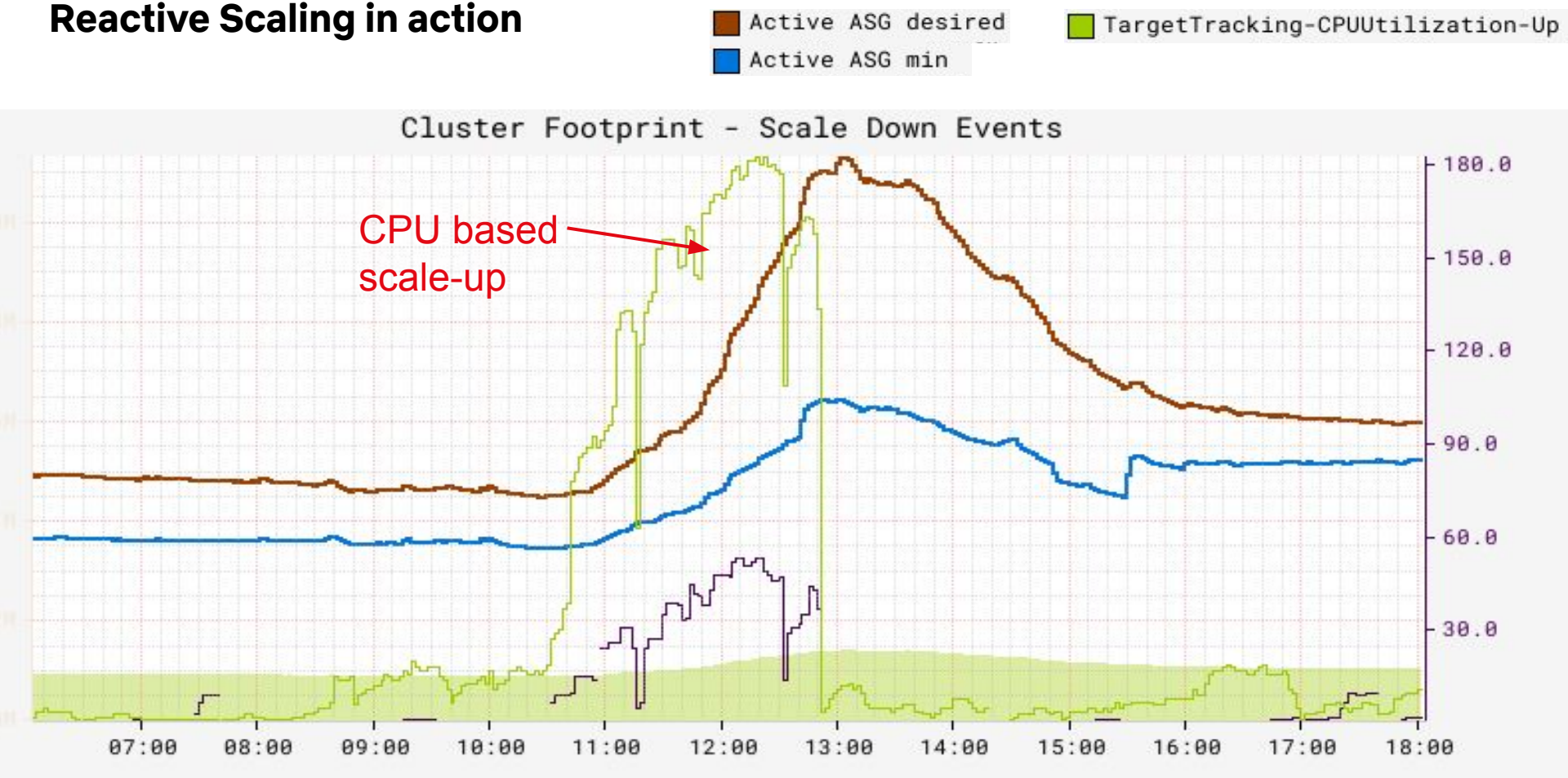




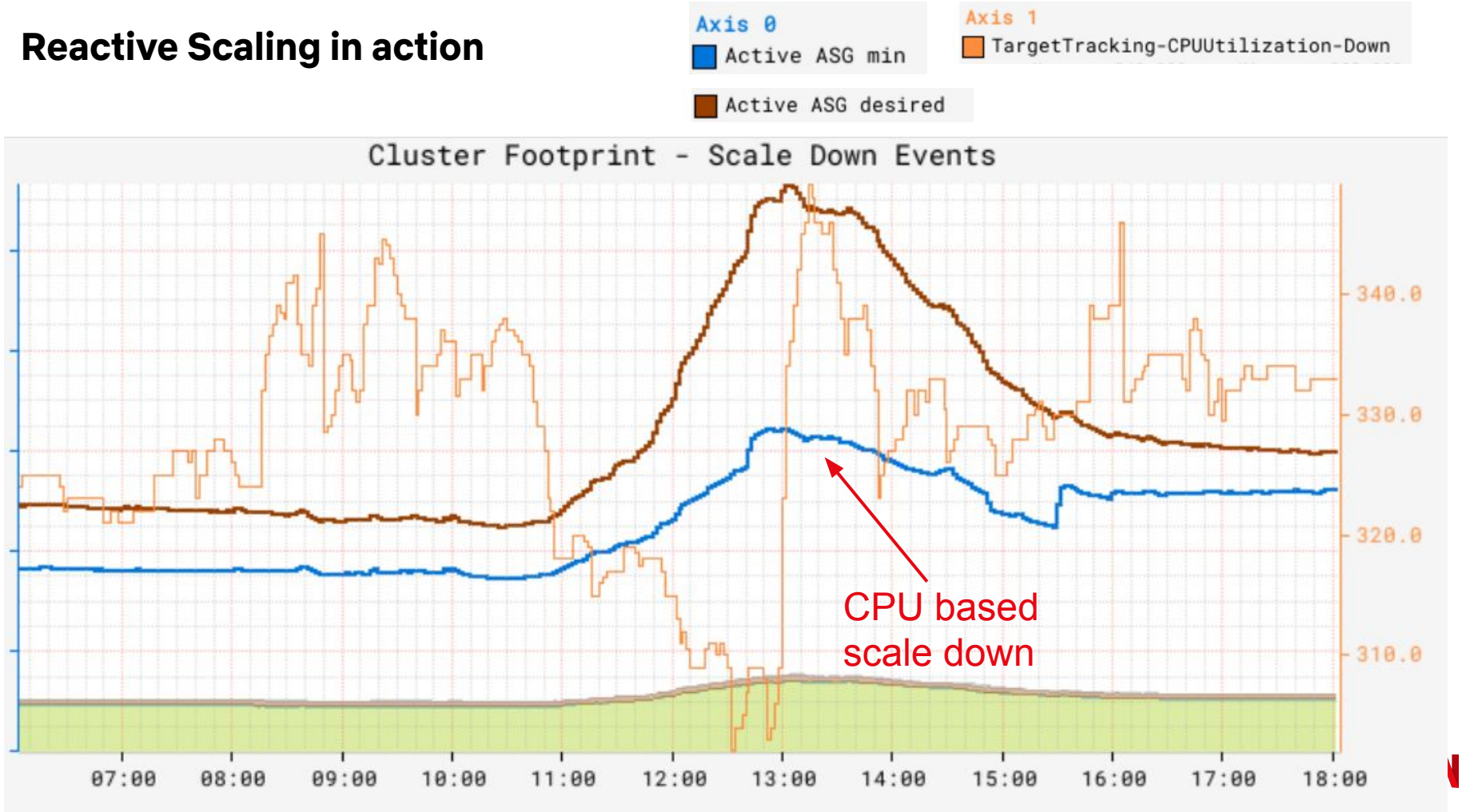
Scaling Policies

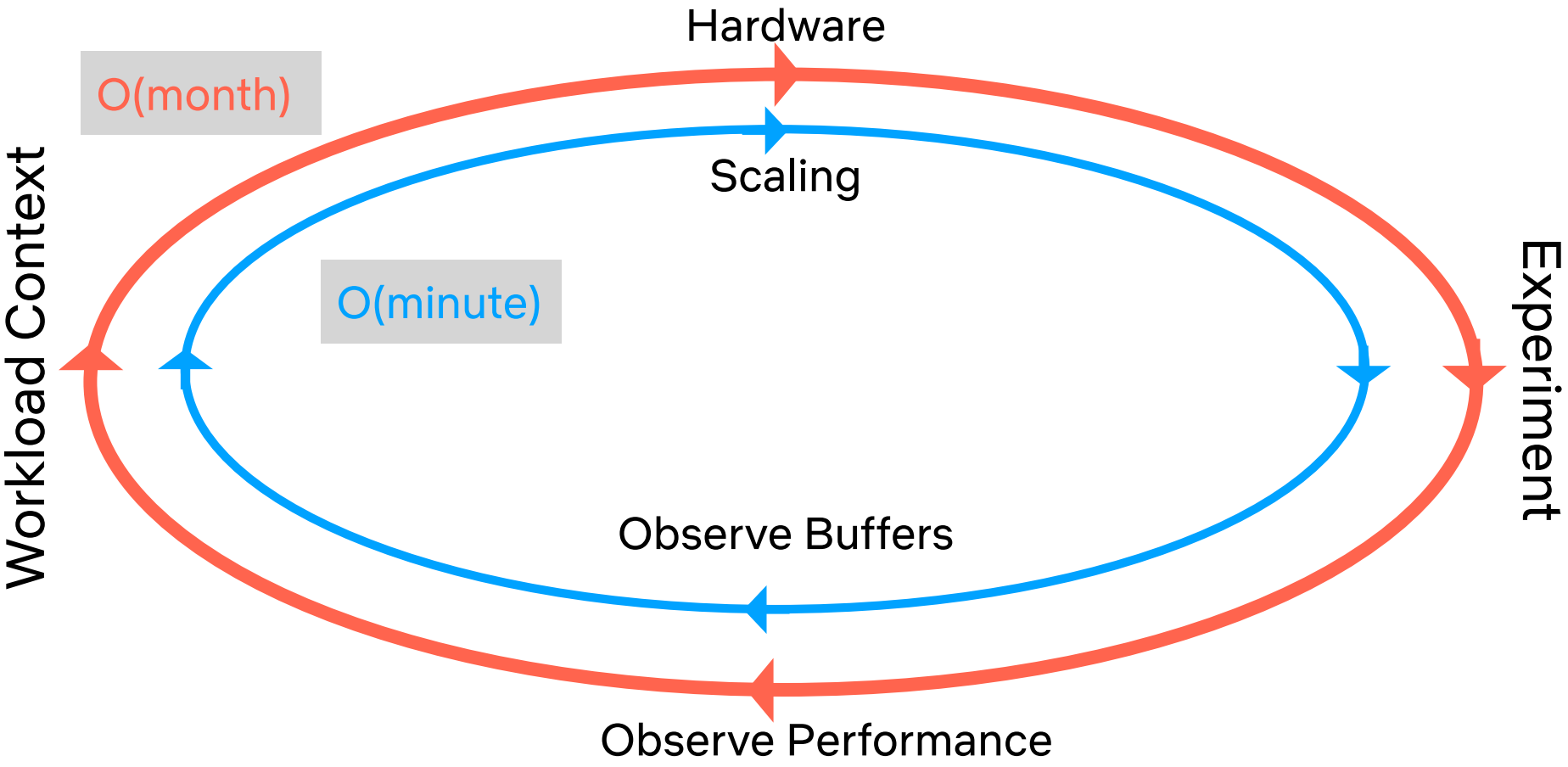


Reactive Scaling in action

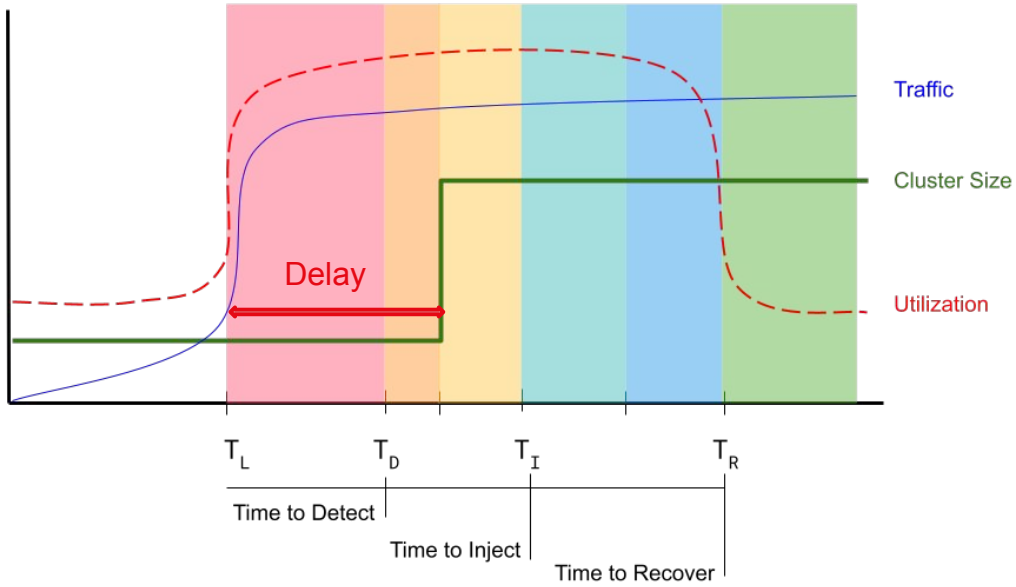


Reactive Scaling in action




Hardware
O(month)
Scaling
Workload Context
O(minute)
Experiment
Observe Buffers
Observe Performance

But....auto-scaling 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.

Does reactive *FAST* scaleup work in practice?

Tier 0 Before Tuning
10x Load Spike TTR

8-15 mins

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

Does reactive *FAST* scaleup work in practice?

~70% reduction

FAST

Tier 0 Before Tuning
10x Load Spike TTR

8-15 mins

Tier 0 *After* Tuning
10x Load Spike TTR

3-4 mins

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

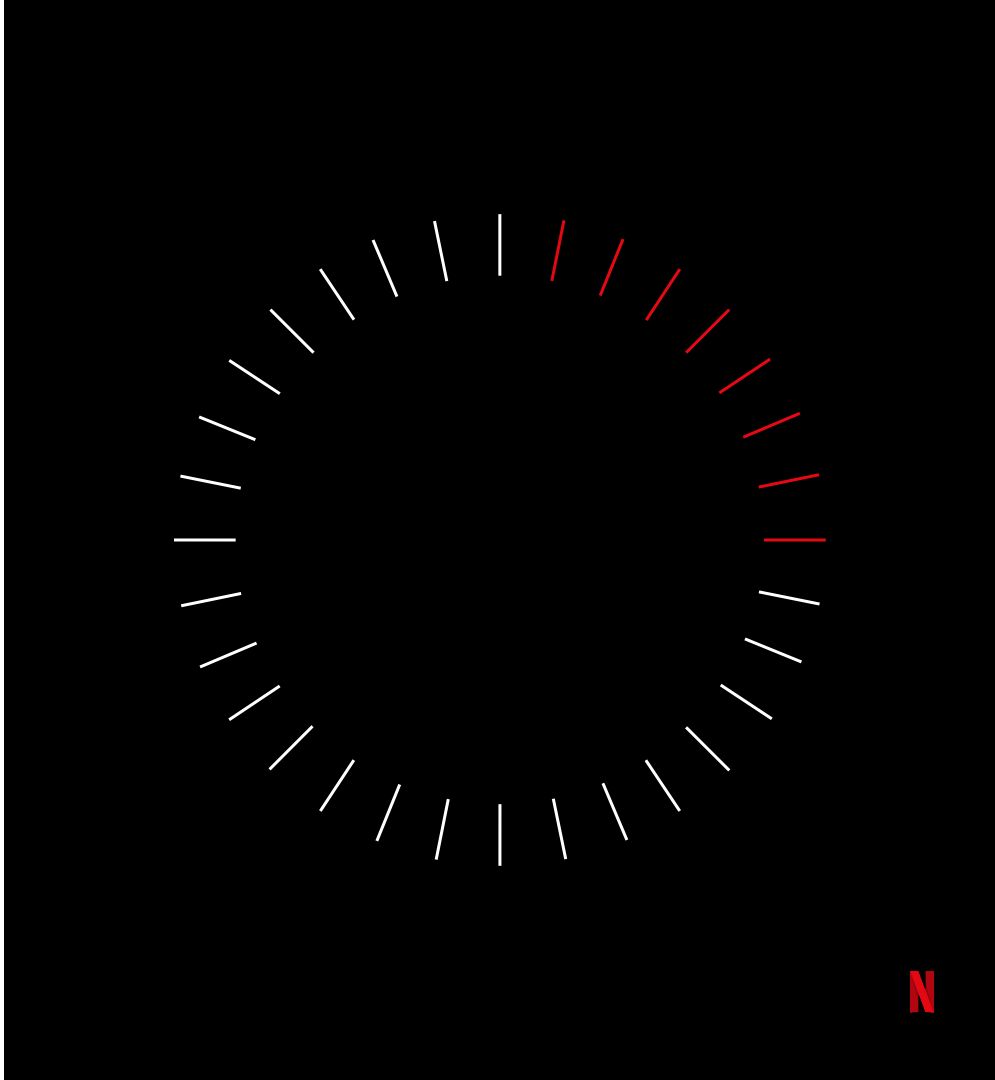
Manage Demand

Pre-scaling ✅

Dynamic Traffic Shaping ✅

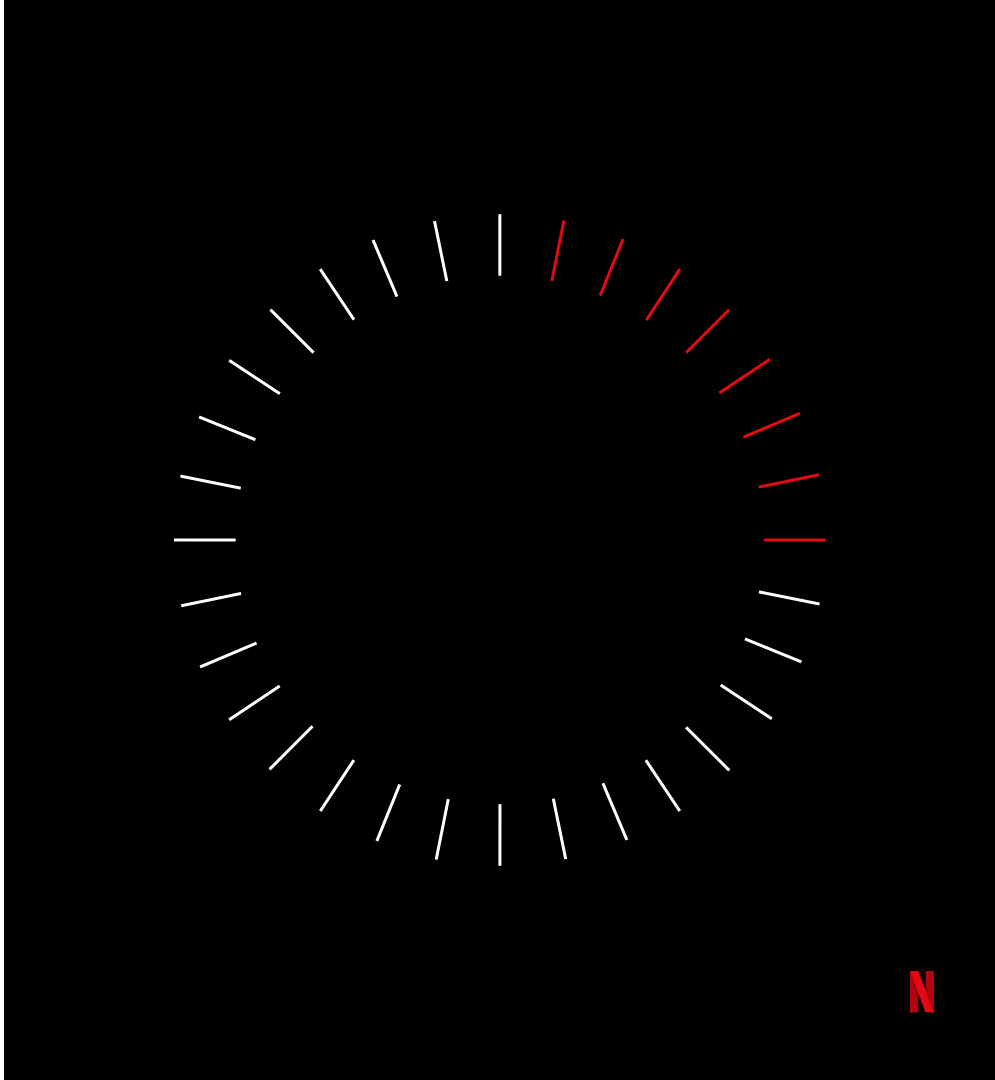
Reactive Auto-Scaling ✅

Load Shedding

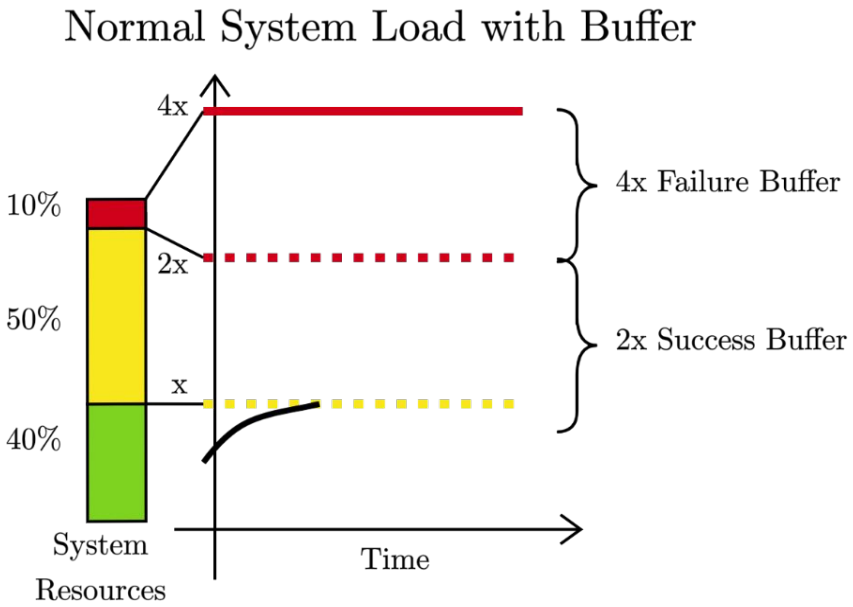


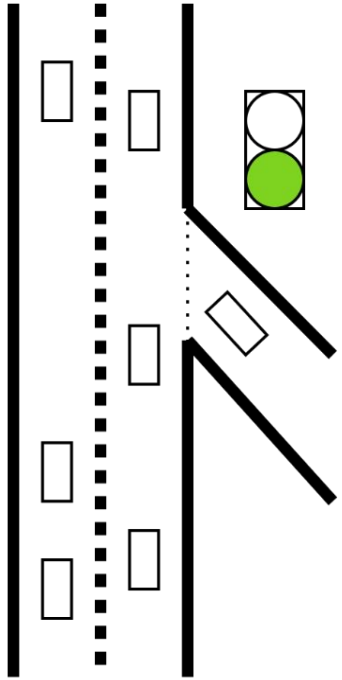
Load Shedding

Bulk load shedding



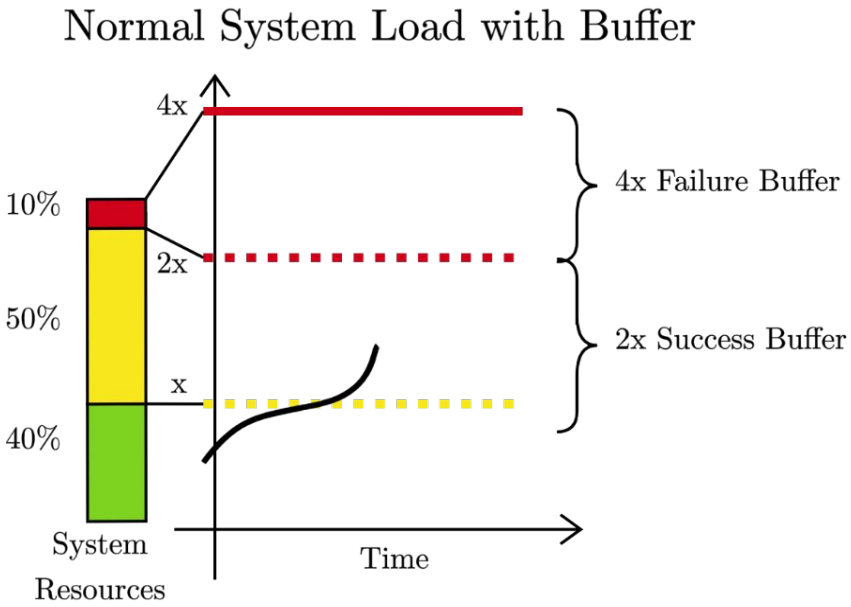
CPU Load Shedding

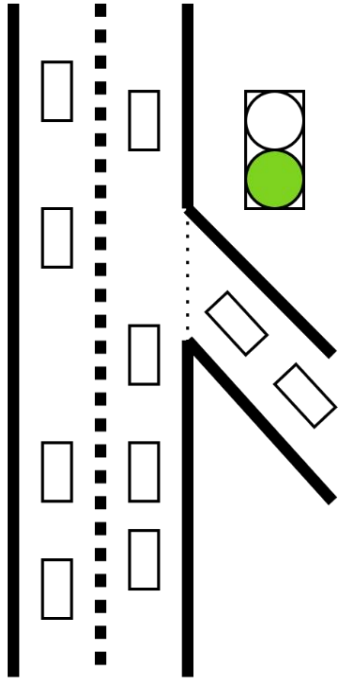




$T_{transit} = 10m$

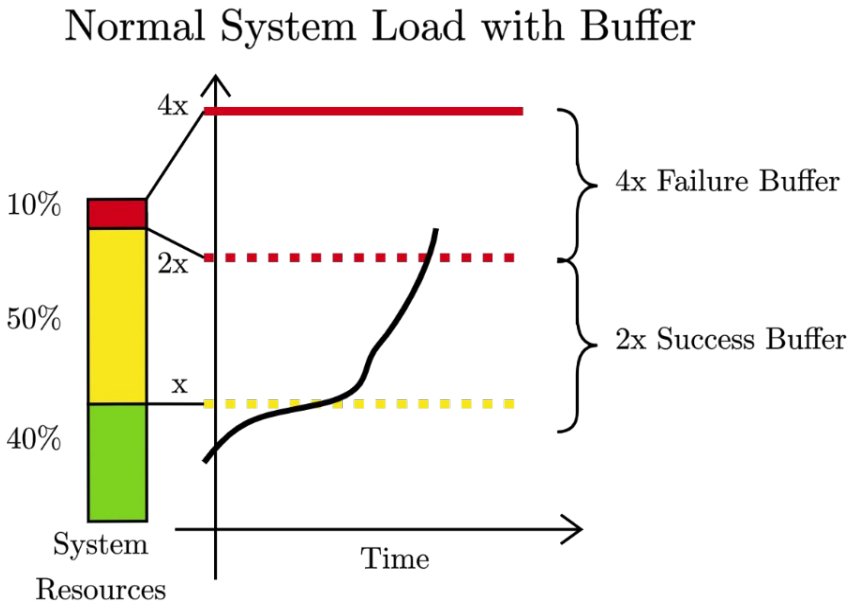
CPU Load Shedding

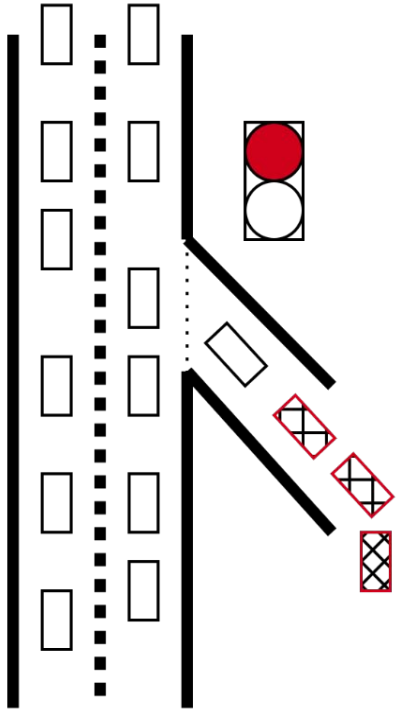




$T_{transit} = 15m$

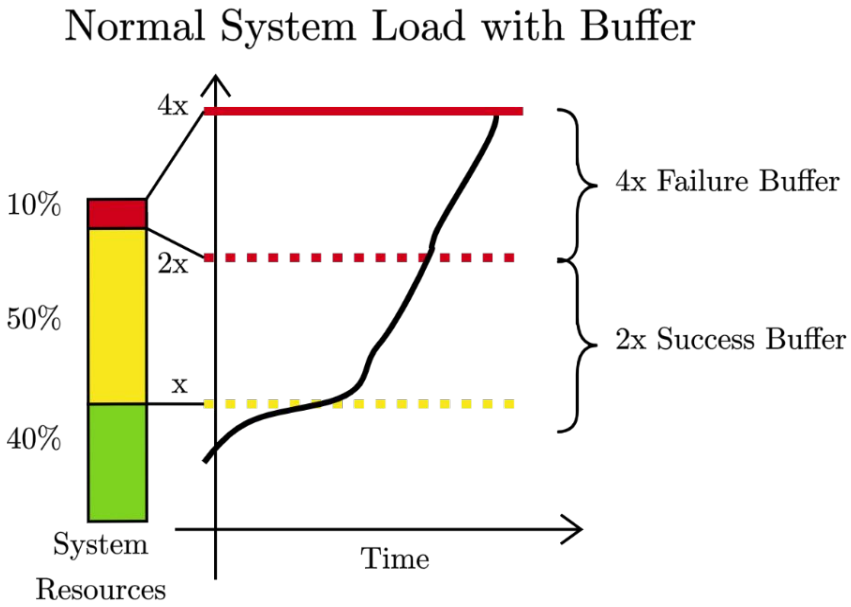
CPU Load Shedding

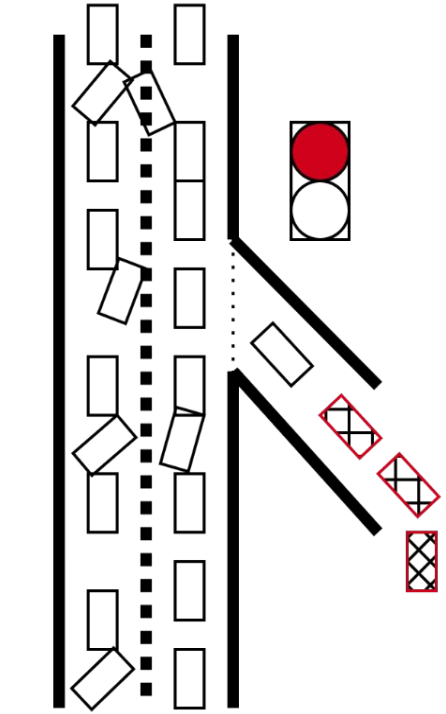




$T_{transit} = 30m$

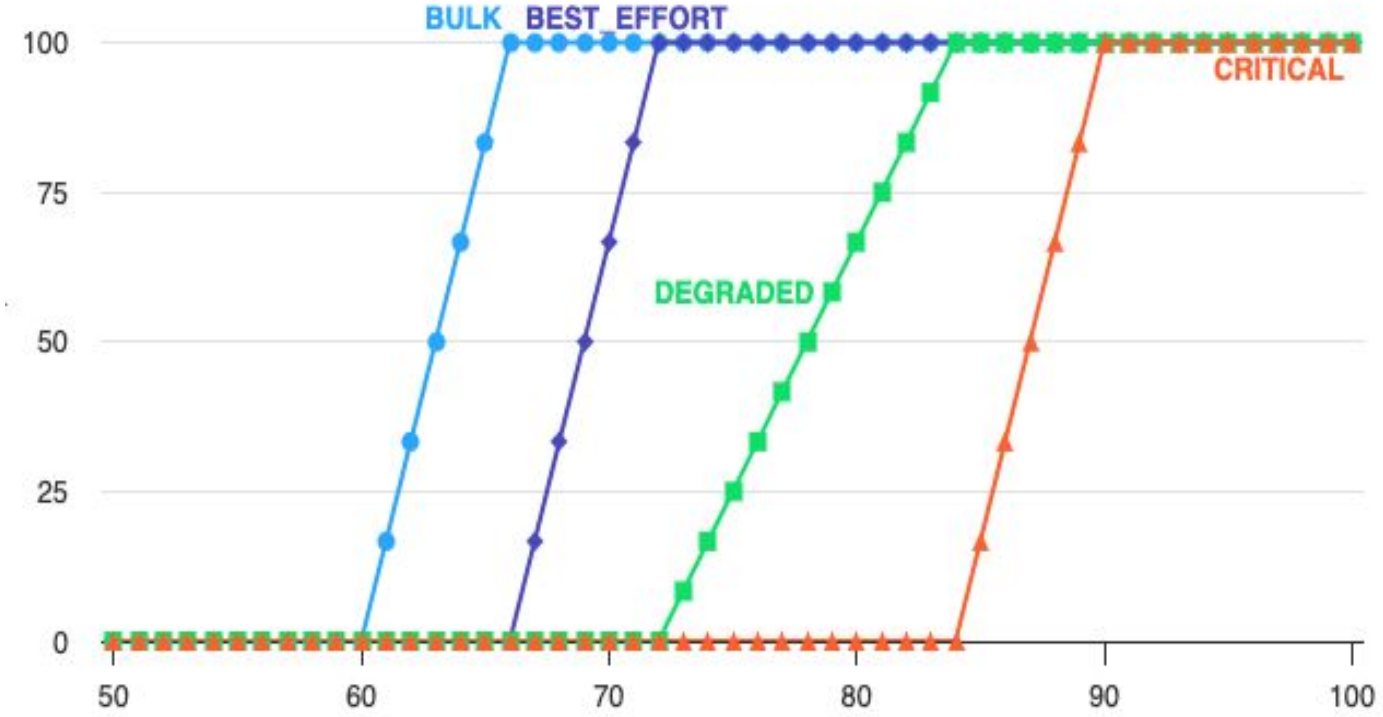
CPU Load Shedding





$$T_{transit} = \infty$$

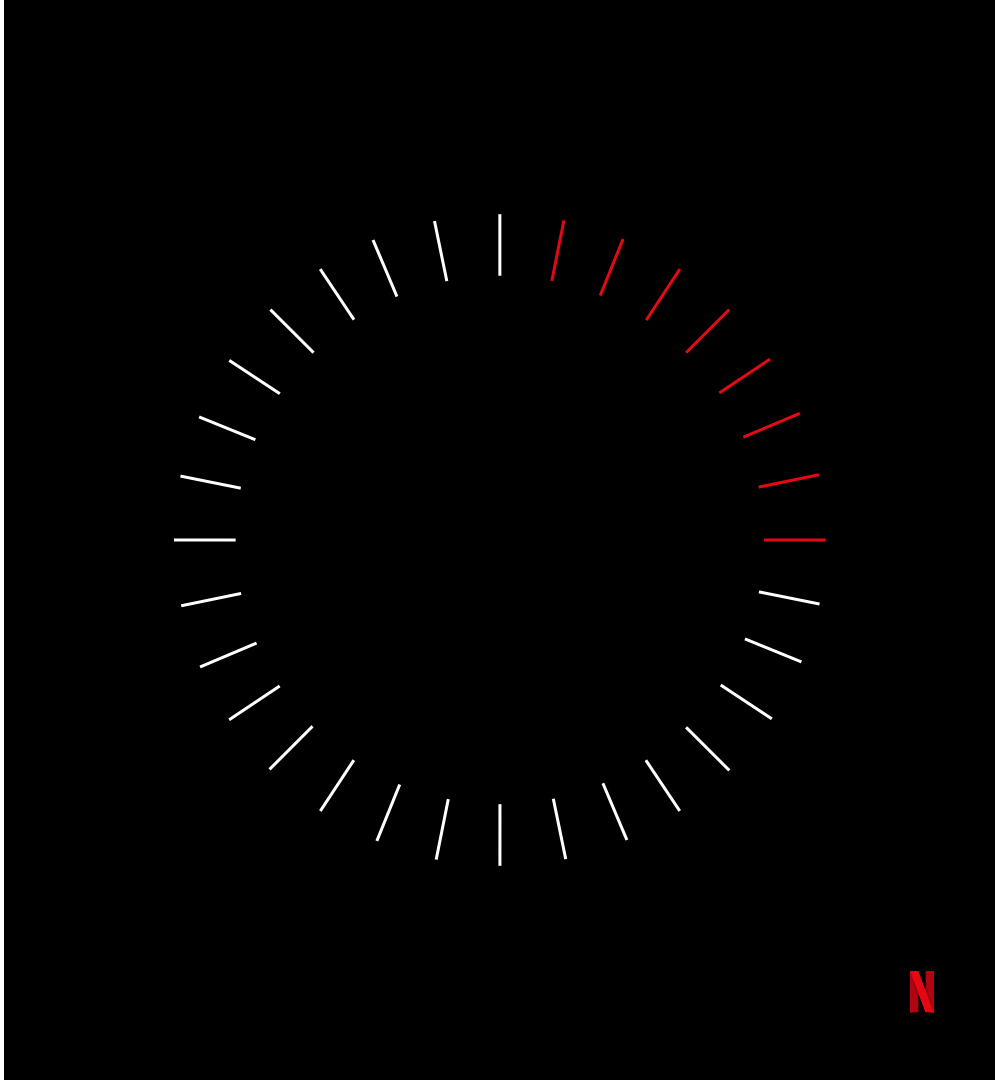
CPU Load Shedding



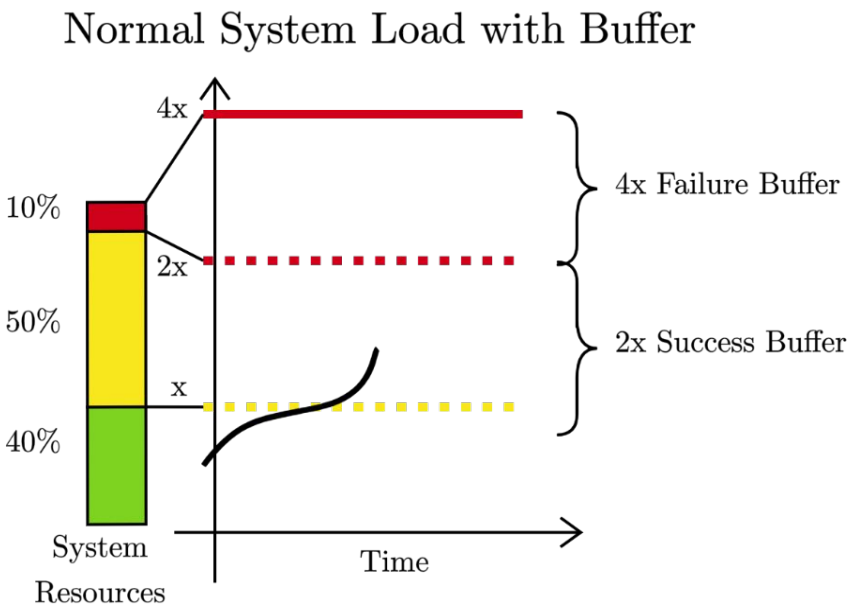
Load Shedding

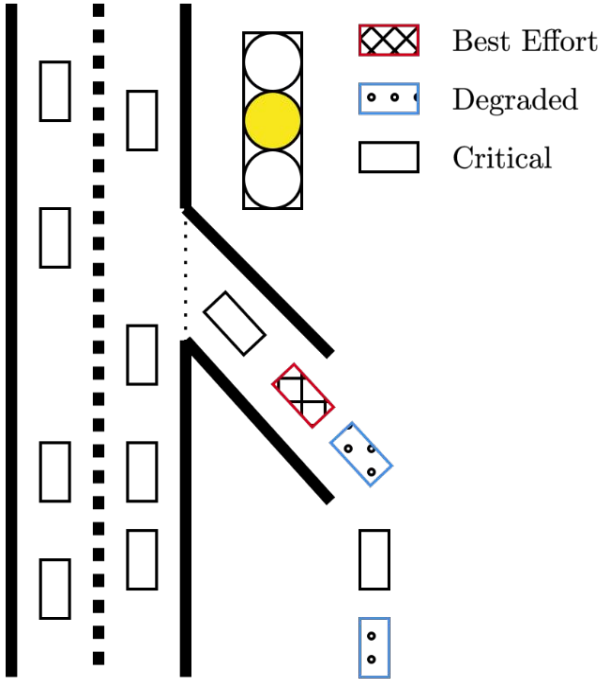
Bulk load shedding ✅

Priority based load shedding



Prioritization creates additional Buffer

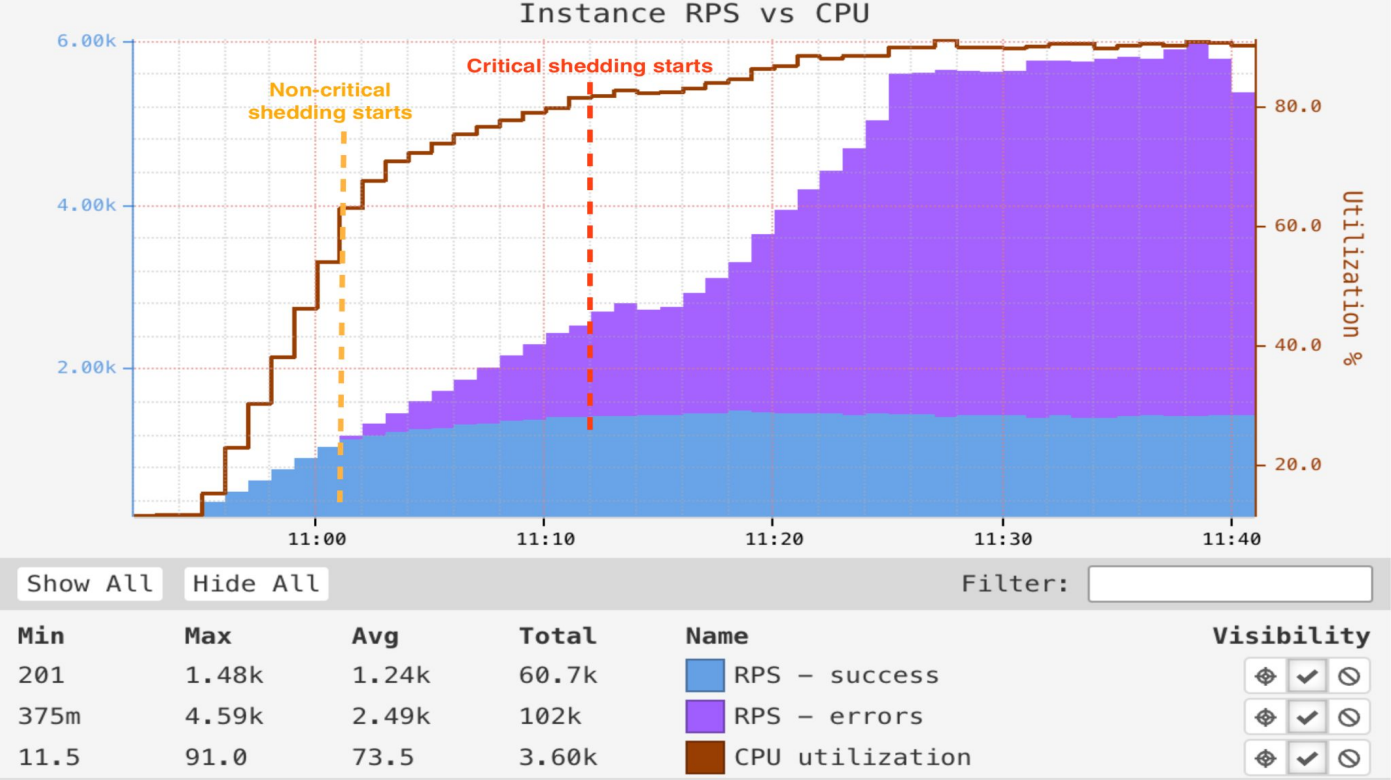




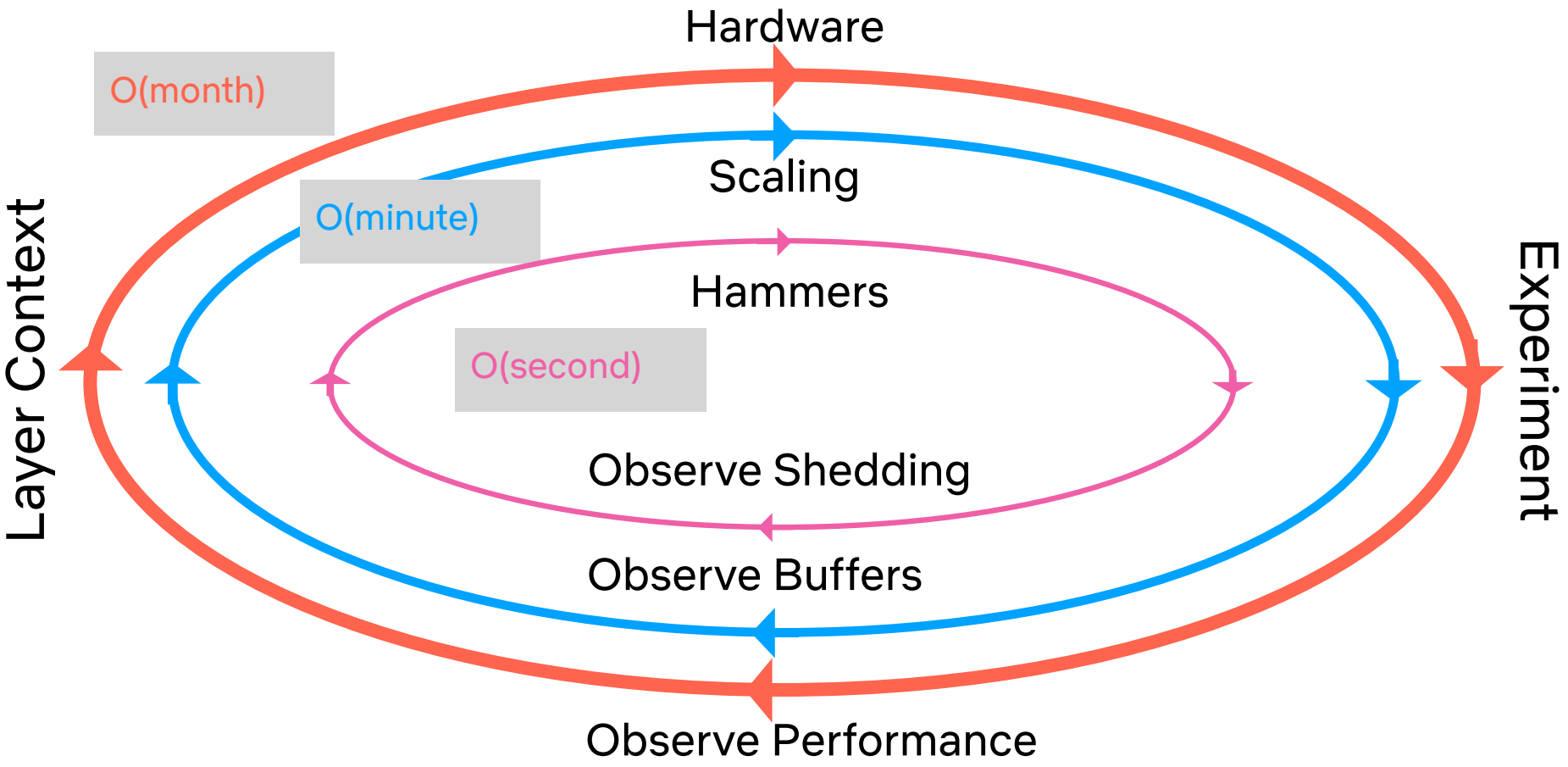
$T_{transit} = 15m$

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

Load Shedding
Priority based



QCon 2025: Anirudh Mendiratta, Benjamin Fedorka on **Service-Level Prioritized Load Shedding**

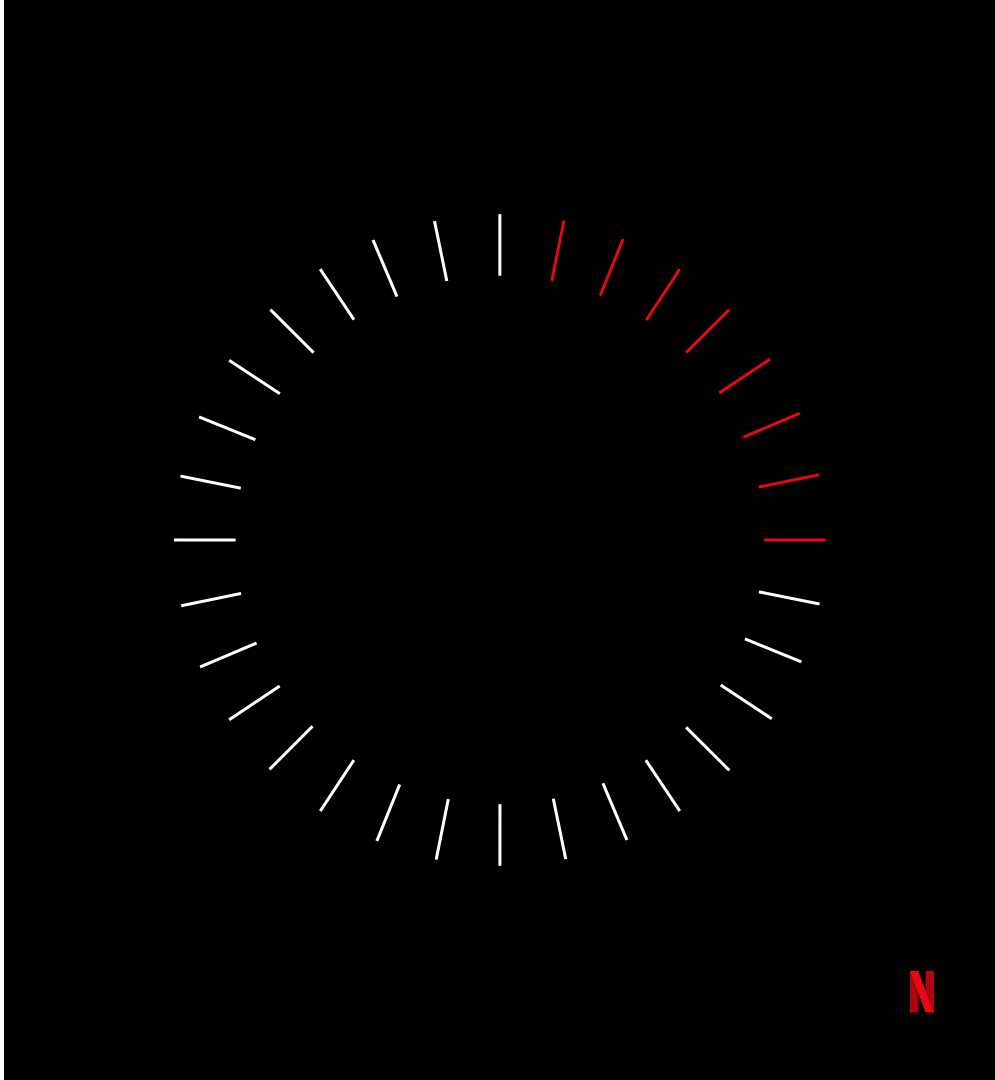

Hardware
O(month)
Scaling
O(minute)
Hammers
O(second)
Observe Shedding
Observe Buffers
Observe Performance
Layer Context
Experiment

Lesson #1

Proactive **and** Reactive levers

Pre-Ingress ✅
- Predictive Scaling
- Traffic Shaping

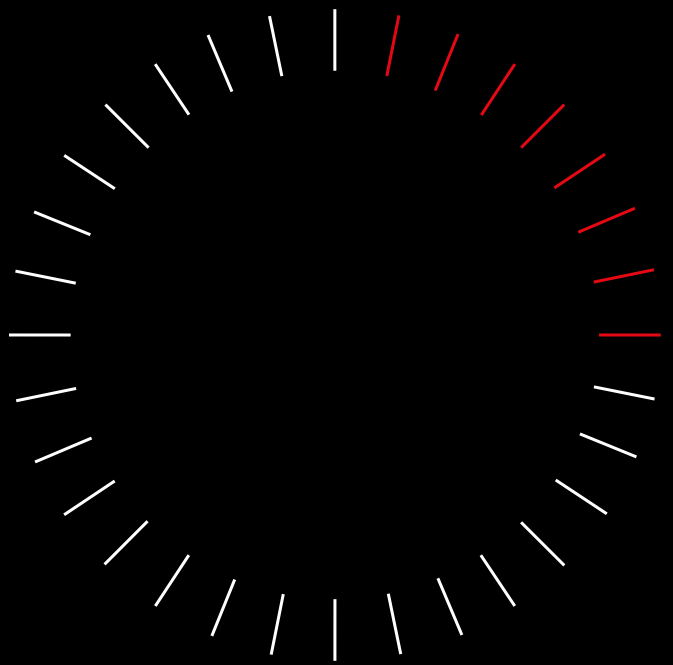
Post-Ingress ✅
- Reactive Scaling
- Load Shedding



Lesson #2

Systems Thinking..

- Raw Compute isn’t the silver bullet to buy down **Risk**
- Efficiency wins **compound**, at scale.
- **End to End** Traffic Management

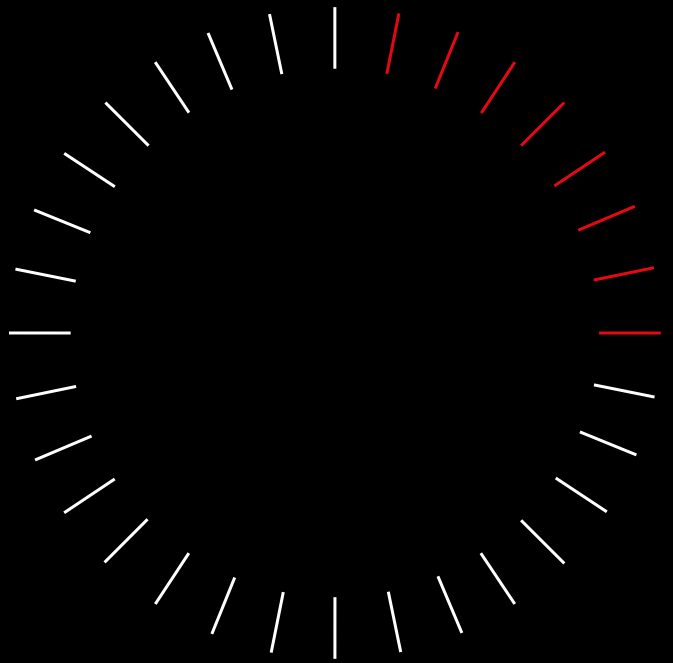


Lesson #3

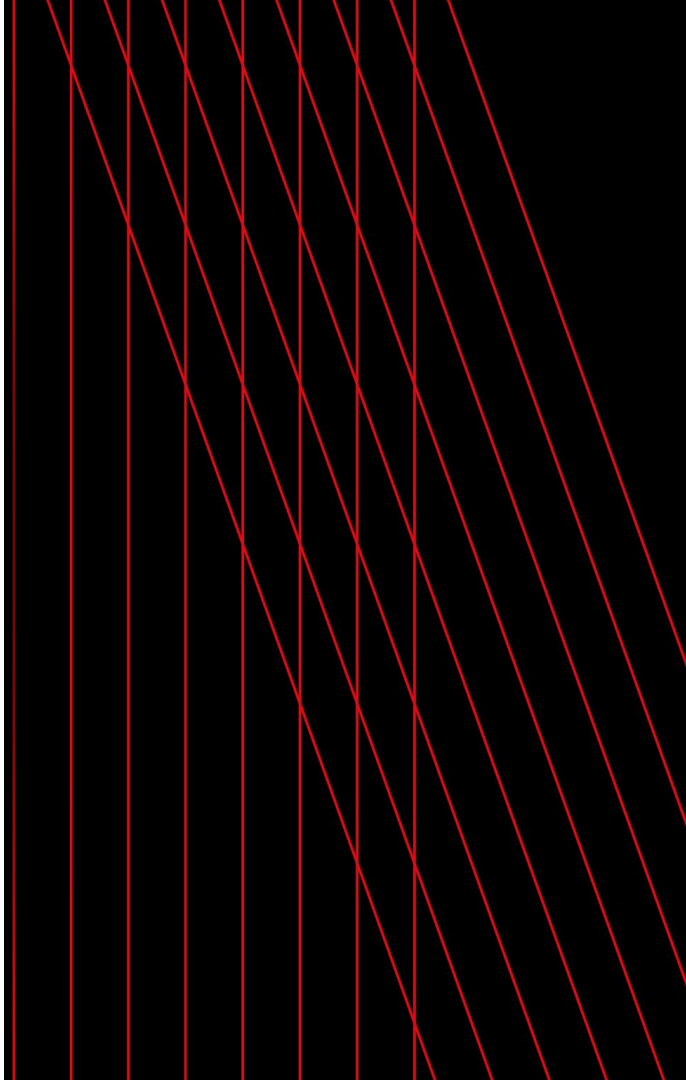
Math is our Safety Blanket™

🤝

Econ 101



Thank You



Argha C, arghac@netflix.com
Joseph Lynch, josephl@netflix.com

Please rate and leave feedback!

Argha C, arghac@netflix.com
Joseph Lynch, josephl@netflix.com

Questions?