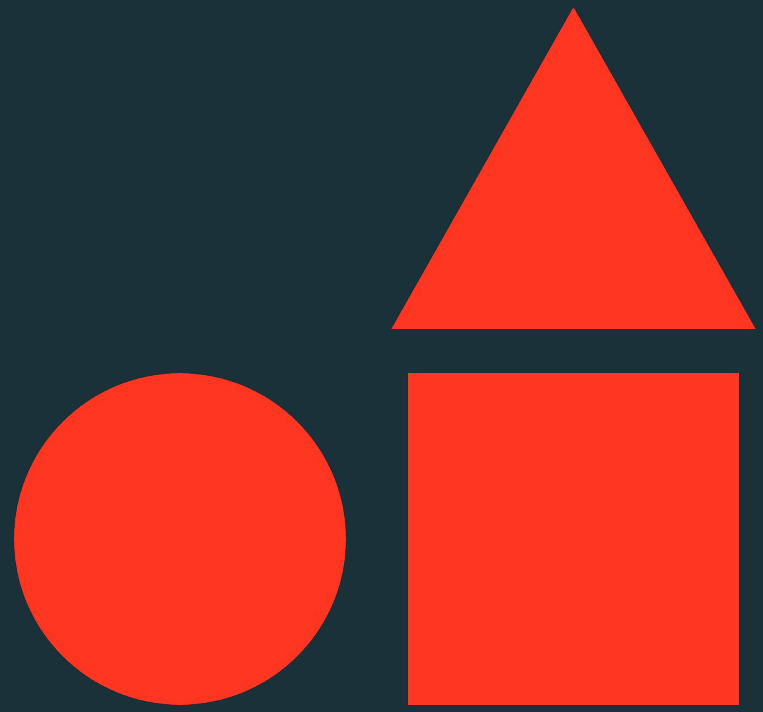
databricks

Intelligent Load Balancing in Kubernetes

A journey to fix request imbalance at scale

Gaurav Nanda, Vincent Cheng



SRECon 2026

The Databricks platform

Make data and AI accessible across the organization

What we do

- Unified platform for data + AI applications
- Creators of Apache Spark, Delta Lake, MLflow and Unity Catalog

Growth

- >65% YoY growth
- $5.4B+ revenue run rate
- 20,000+ customers

Operating at global scale

Highly distributed, latency-sensitive systems

- 3 Clouds (AWS, Azure and GCP)
- 70+ regions globally
- 1500+ clusters
- Millions of VMs launched daily

It started with SLO violations

Intermittent issues we couldn't explain!

5XX Errors

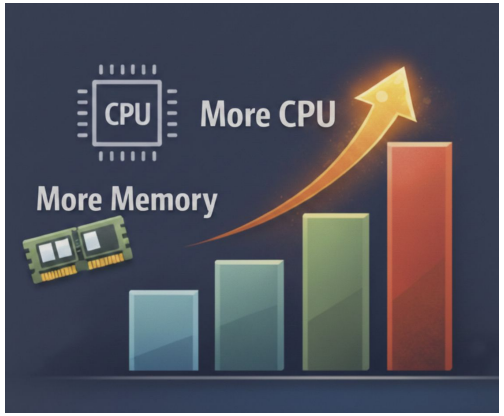
High P99 Latency

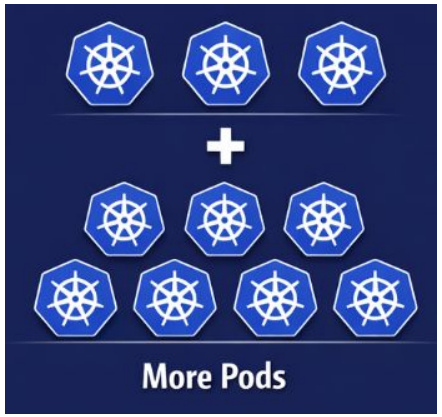
Product team’s first instinct…

Add more resources!!

5XX Errors

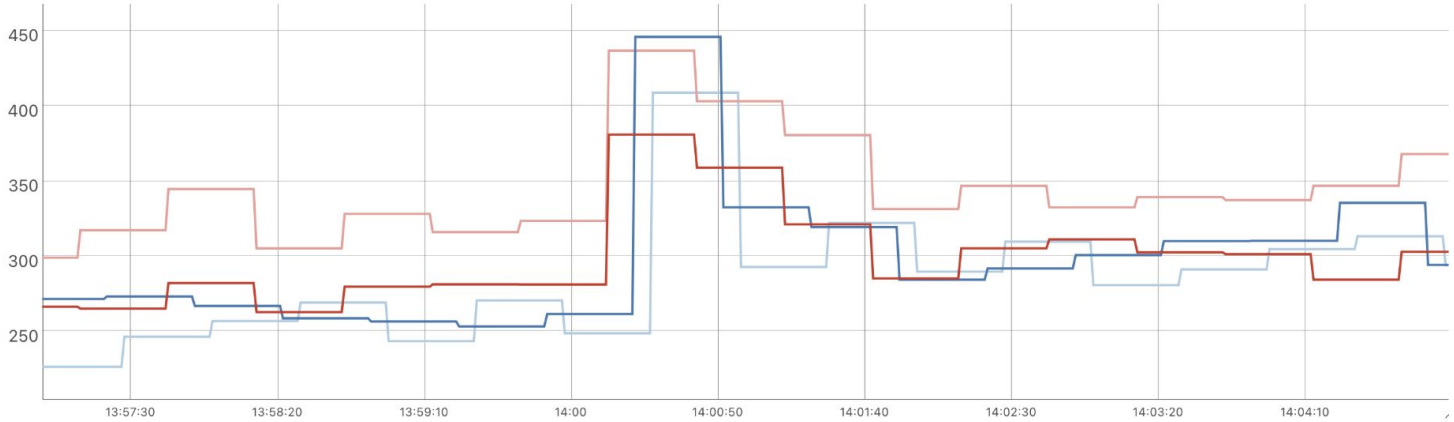
High P99 Latency



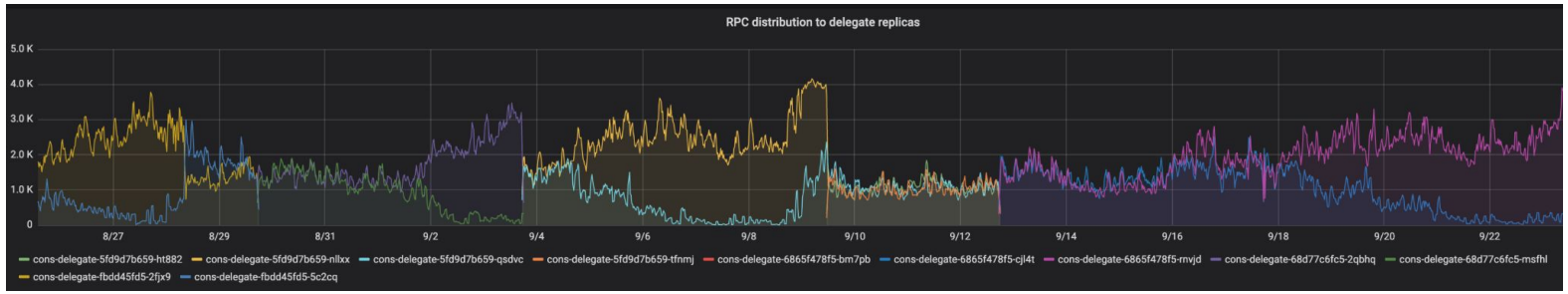


More CPU, memory, and pods!

The Problem: Identical pods, different load

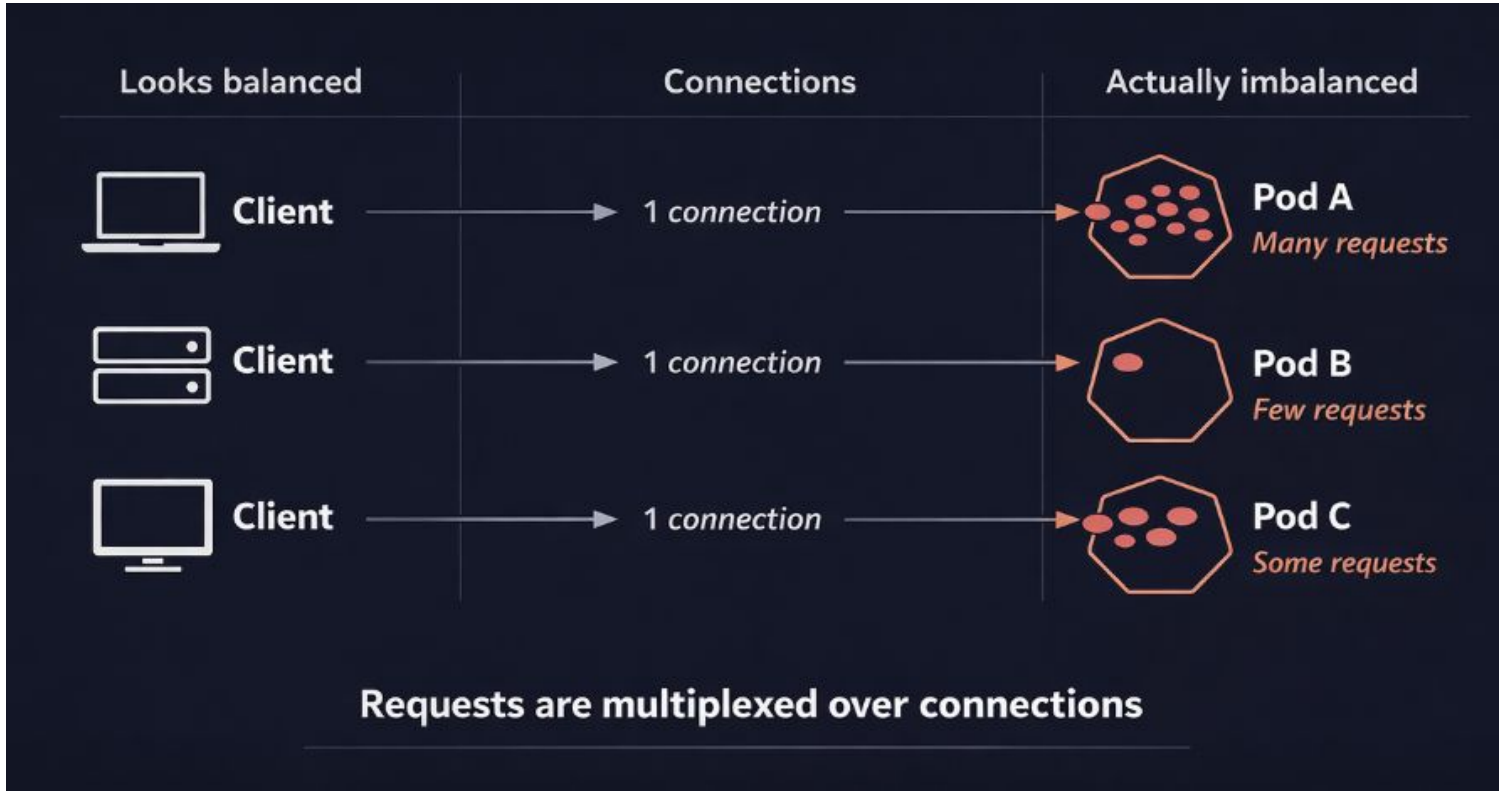


In some cases, pods saw 3–5x difference in QPS!



What actually happens

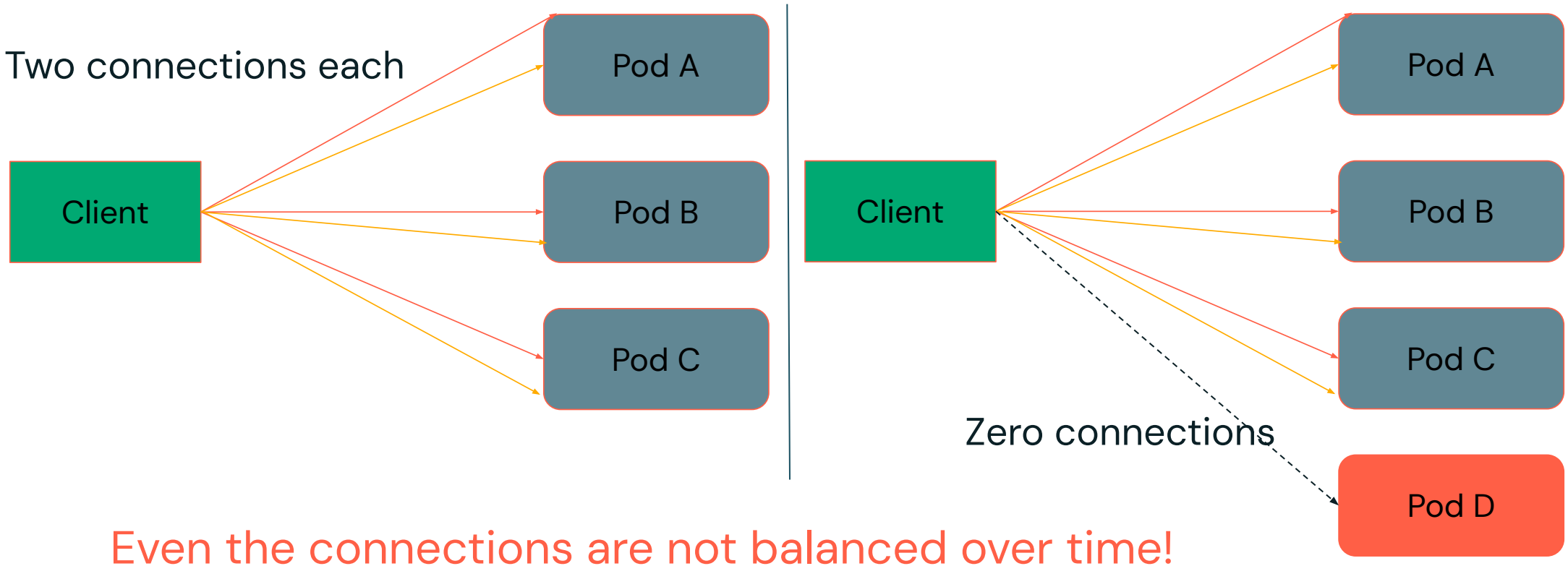
Connections are distributed. Requests are not.



- Load balancing happens at connection setup
- Most traffic is gRPC (HTTP/2)
- Requests are multiplexed over connections

And it gets worse..

Connections don’t rebalance when the system changes



Even the connections are not balanced over time!

We tried fixing it!

Some obvious approaches

Reset Connections

Force redistribution

Headless Service

Client side routing (DNS)

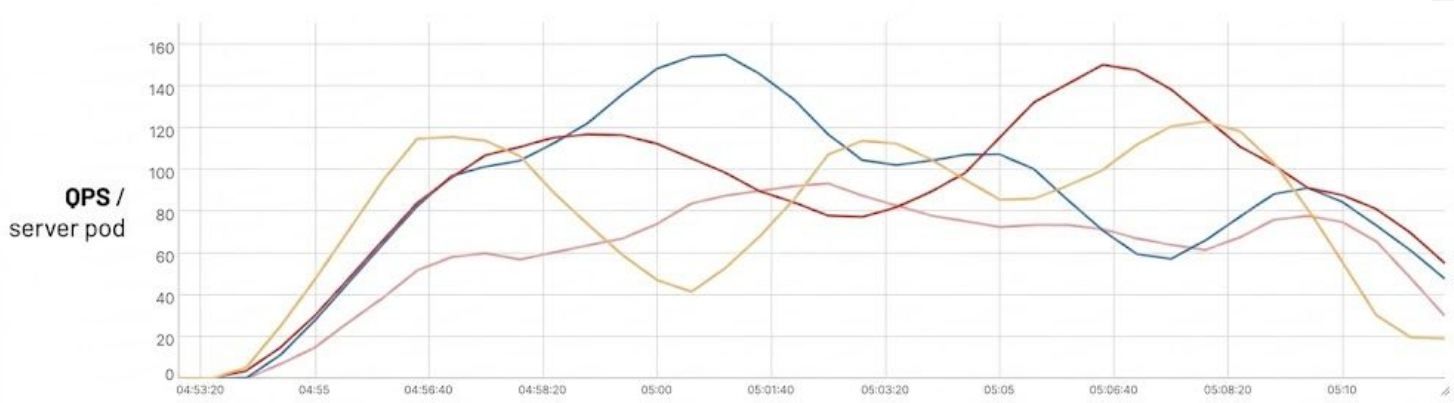
Service Mesh

Istio

Reset Connections

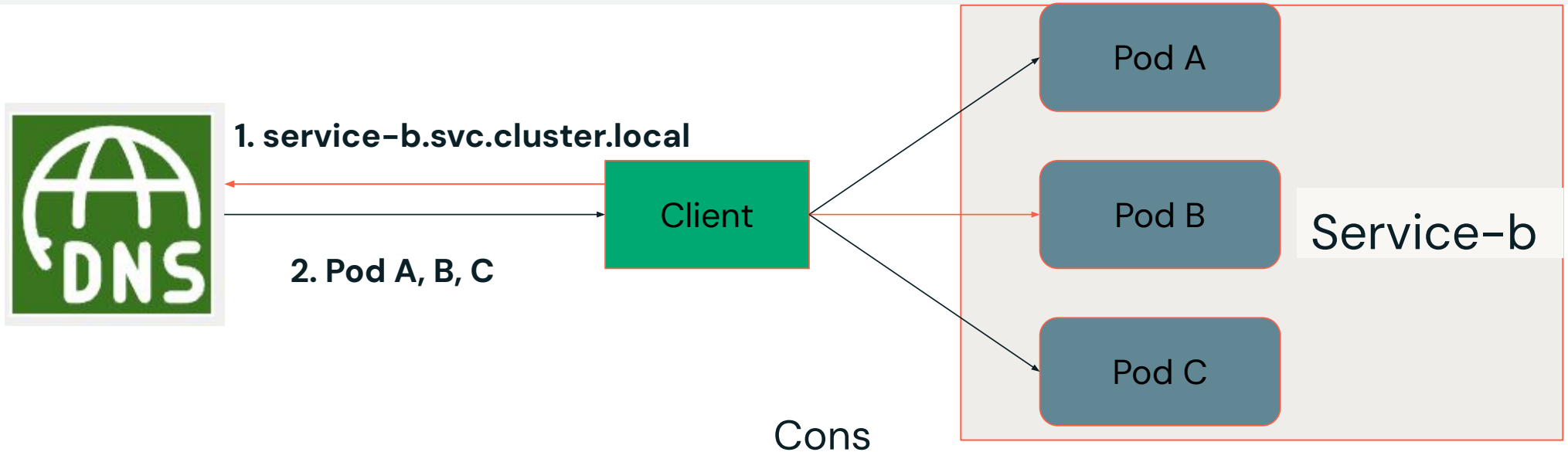
Reset connections every X requests or X time

+ Improves distribution slightly
- Requests per connection still remain a problem
- Expensive: **Higher latency and CPU overhead**
- Harder to tune



Headless Service

Client side load balancing via DNS



Pros

- Enables client side load balancing
- Per-request L7 load balancing
- Minimal effort

Cons

- DNS caching -> stale endpoints
- DNS reliability issues
- DNS as control plane - cannot support advanced use-cases

Service Mesh

- Significant operational complexity
- Additional network hop leading to higher latency

None of these fully solve it

Partial fixes, added complexity, or missing signals

Reset Connections

Force redistribution

Headless Service

Client side routing (DNS)

Service Mesh

Istio

We need something more intelligent!

What we actually needed

Requirements beyond simple load balancing

Fairness

- Uniform request distribution
- Avoid hotspots

What we actually needed

Requirements beyond simple load balancing

Fairness

- Uniform request distribution
- Avoid hotspots

Efficiency

- Minimal service discovery latency
- Minimal overhead – no proxies

What we actually needed

Requirements beyond simple load balancing

Fairness

- Uniform request distribution
- Avoid hotspots

Efficiency

- Minimal service discovery latency
- Minimal overhead - no proxies

Awareness

- Pod Load / Health
- Rich Pod Metadata - e.g: Availability Zone to minimize x-zone cost/latency

What we actually needed

Requirements beyond simple load balancing

Fairness

- Uniform request distribution
- Avoid hotspots

Efficiency

- Minimal service discovery latency
- Minimal overhead - no proxies

Awareness

- Pod Load / Health
- Rich Pod Metadata - e.g: Availability Zone to minimize x-zone cost/latency

"All pods are equal.. But some pods are more equal than others"

Building our solution

Translating requirements into implementation

Efficiency

- Minimal service discovery latency
- Minimal overhead – no proxies

- Push-based, not pull-based model
- Let client libraries make the decisions

Building our solution

Translating requirements into implementation

Efficiency

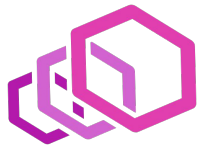
- Minimal service discovery latency → Push-based, not pull-based model
- Minimal overhead - no proxies → Let client libraries make the decisions

Awareness

- Pod Load / Health → Realtime signals
- Rich Pod Metadata - e.g: Availability Zone → Support flexible payloads

Where we got lucky

xDS: a natural fit

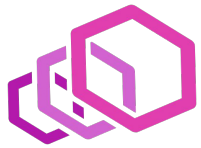


Ingress Traffic

S2S Traffic

Where we got lucky

xDS: a natural fit



Ingress Traffic

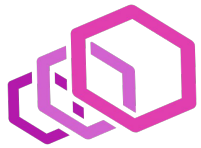
S2S Traffic

Envoy Proxies

- Native discovery protocol
- Less customizability

Where we got lucky

xDS: a natural fit



Ingress Traffic

Envoy Proxies

- Native discovery protocol
- Less customizability

S2S Traffic

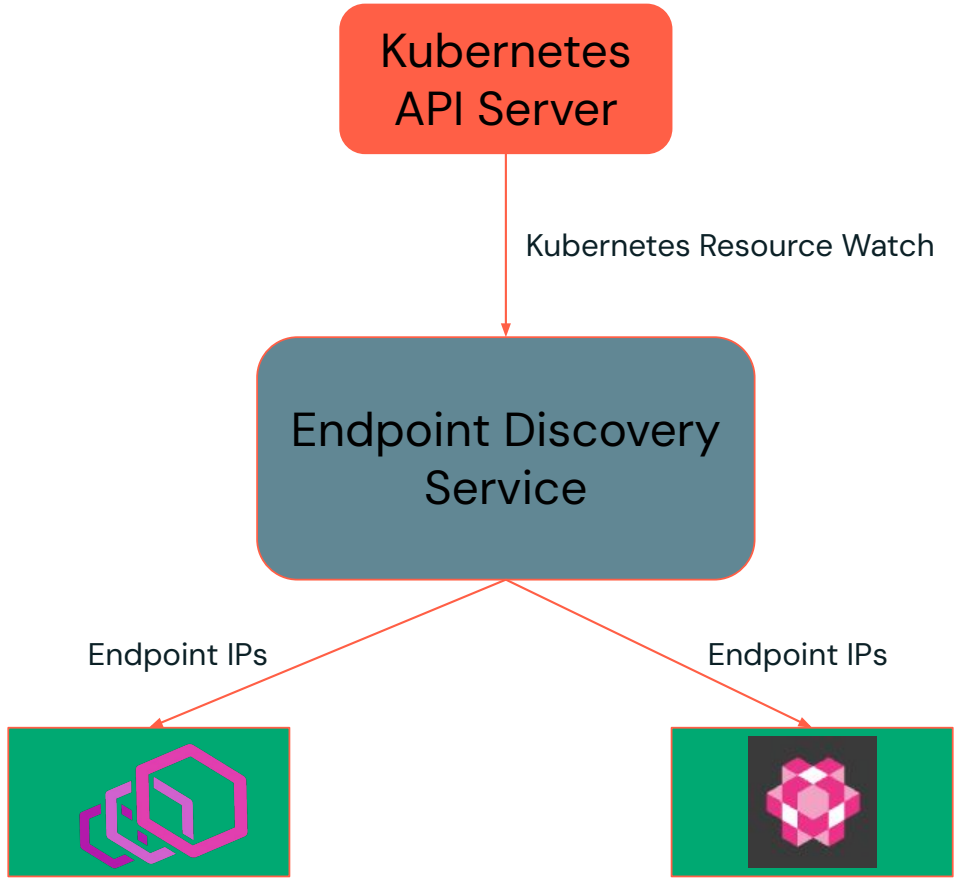
Primarily Scala services

- Uniform framework
- More customizability

Our custom control plane

Extending the pieces that were already there

Still ultimately Kubernetes under the hood!

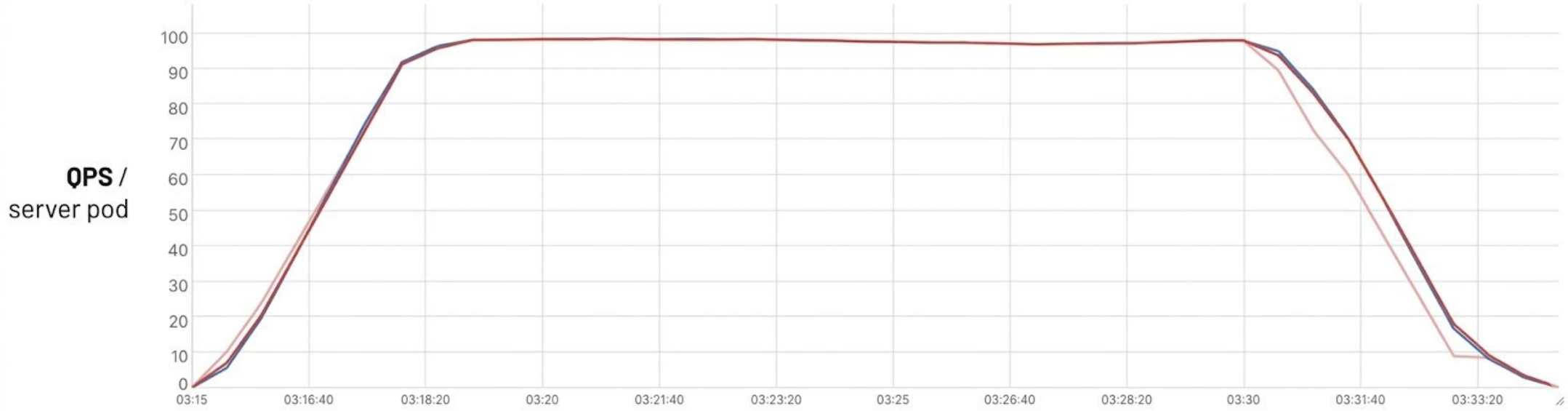


Candidate Algorithms

Starting Simple

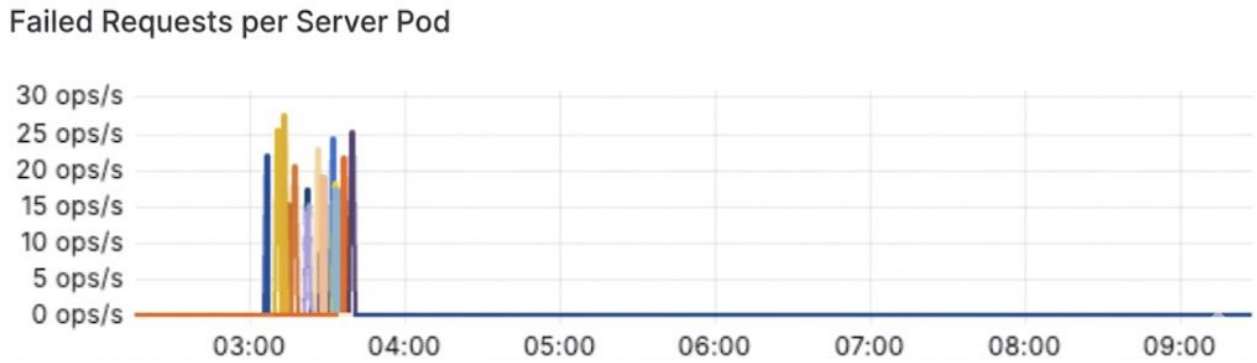
- **Round Robin**
- **Random**

Perfectly Even Distribution



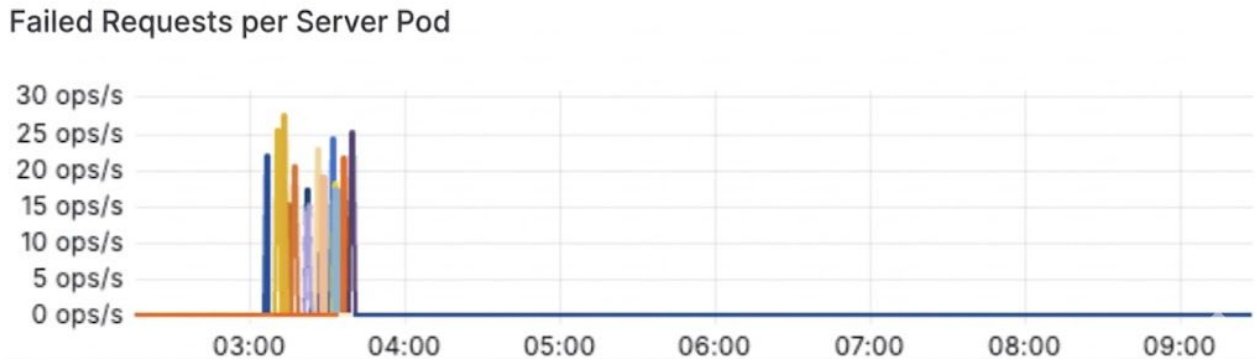
Whoops!

Where Naive Round Robin Failed to Suffice



Whoops!

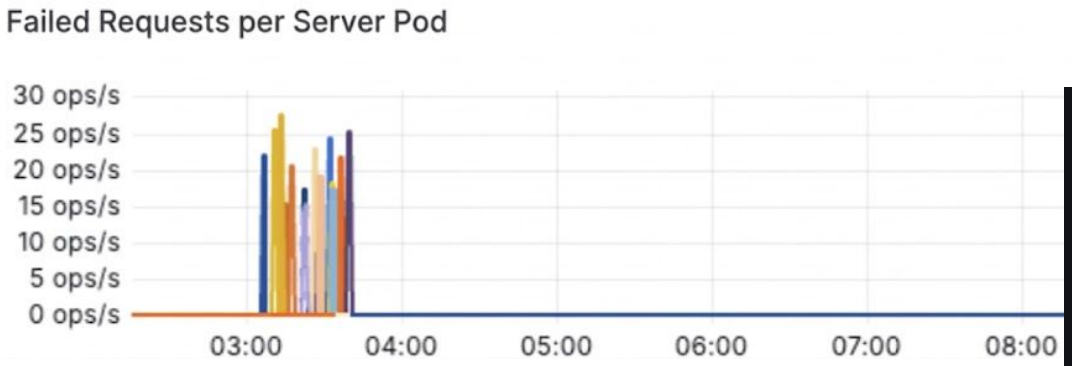
Where Naive Round Robin Failed to Suffice

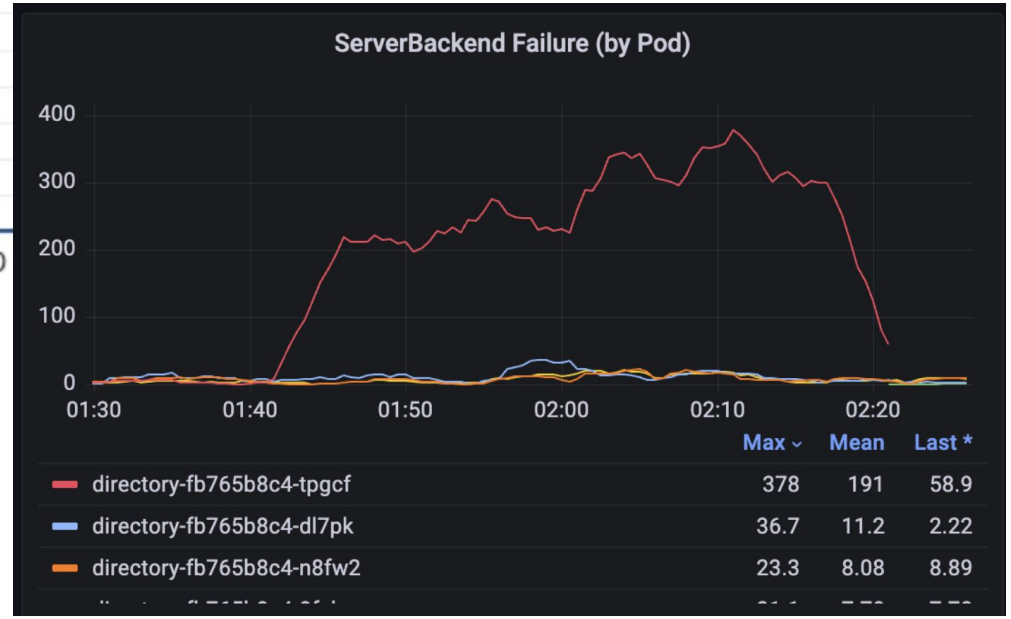


- Cold start was previously hidden by sticky connections
- Clients were now discovering pods “too fast”

Whoops!

Where Naive Round Robin Failed to Suffice





Whoops!

Where Naive Round Robin Failed to Suffice

Is Our Goal to “Equally” Distribute Requests?

One Step Towards Intelligence

- Round Robin
- Random
- **CPU load indicators**

$$\frac{100 * (1 - load\_1m / nr\_cpus)}{\sum_{i=1}^{i=n} (1 - load\_1m(i) / nr\_cpus(i))}$$

Where CPU as an indicator fell short

- Metrics Mismatch

Where CPU as an indicator fell short

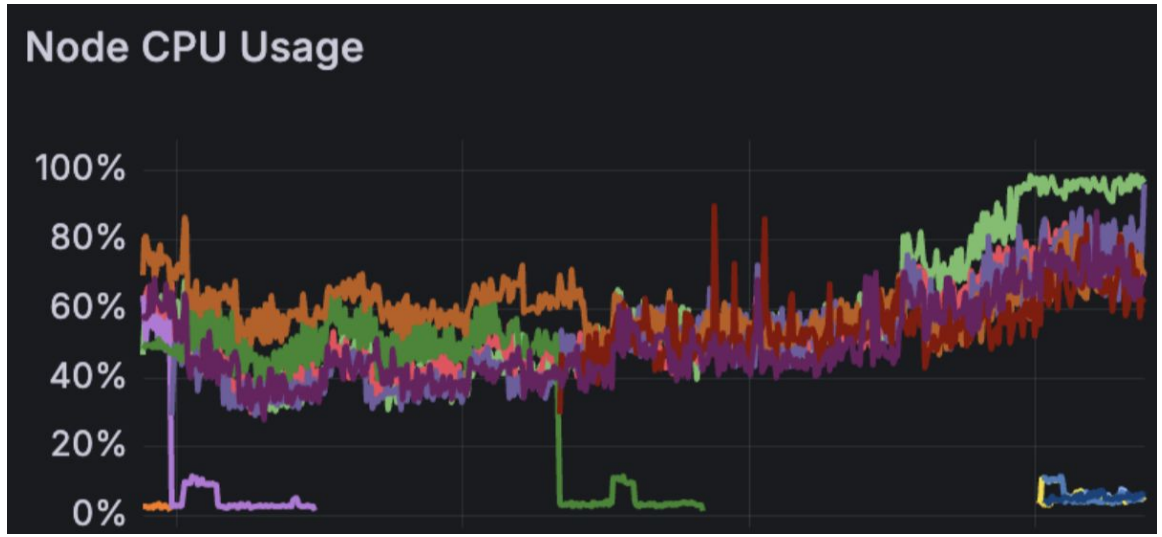
- Metrics Mismatch
  - Trailing indicator

Where CPU as an indicator fell short

- Metrics Mismatch
  - Trailing indicator
  - Realtime signal frequency misalignment

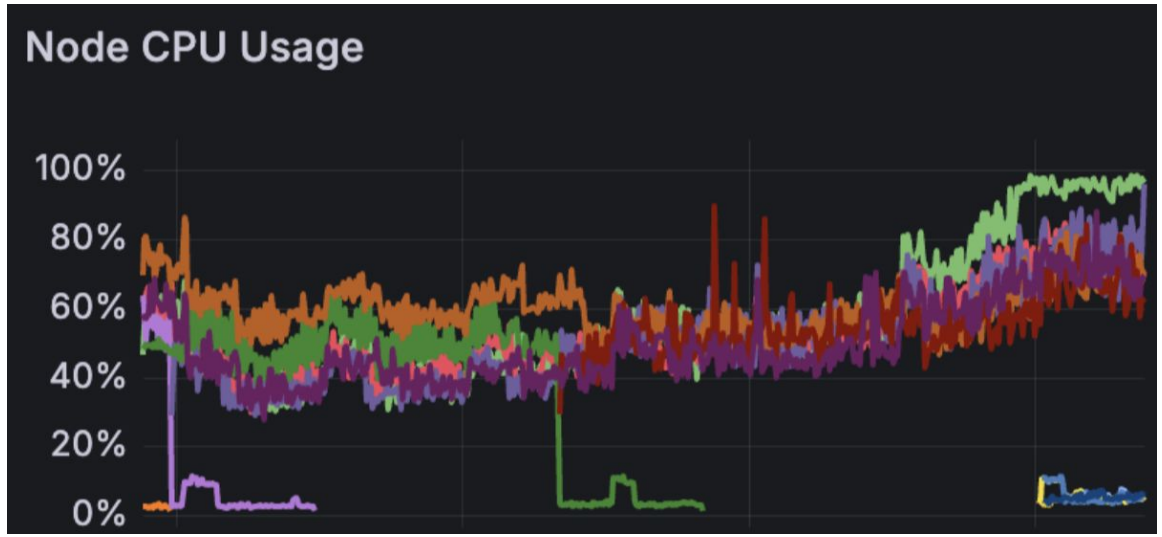
Where CPU as an indicator fell short

- Metrics Mismatch
  - Trailing indicator
  - Realtime signal frequency misalignment
- Noisy Neighbors



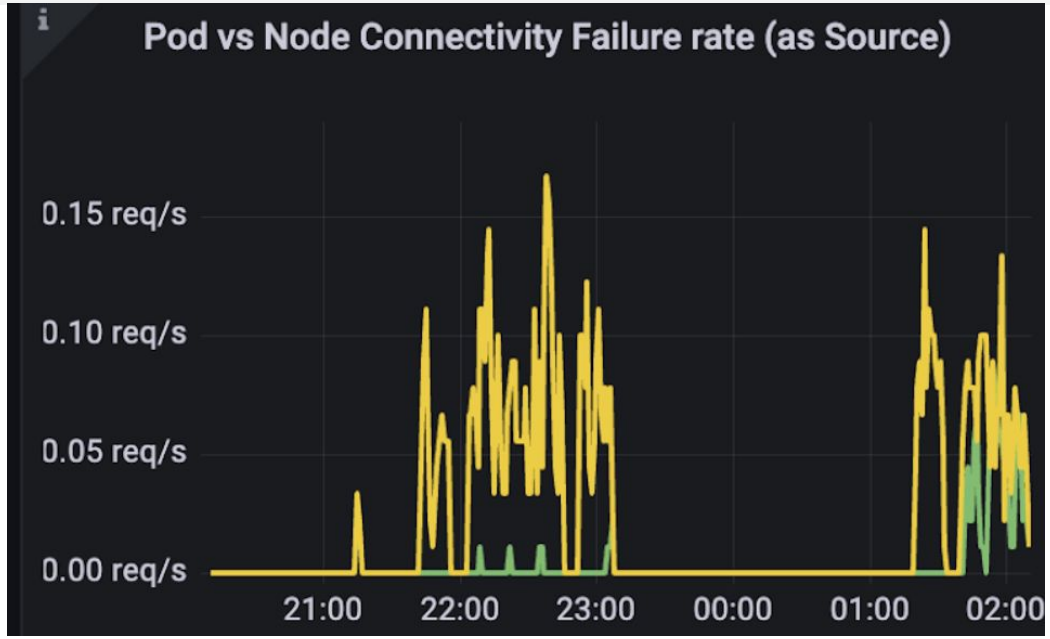
Where CPU as an indicator fell short

- Metrics Mismatch
  - Trailing indicator
  - Realtime signal frequency misalignment
- Noisy Neighbors



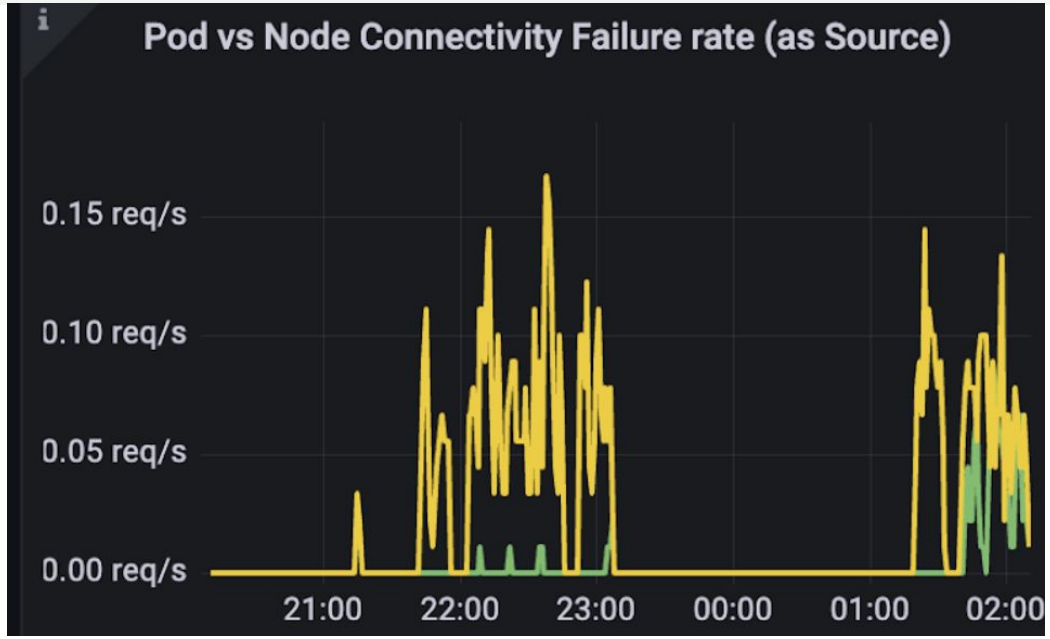
Where CPU as an indicator fell short

- Metrics Mismatch
  - Trailing indicator
  - Realtime signal frequency misalignment
- Noisy Neighbors
- Unrelated Source of Overload



Where CPU as an indicator fell short

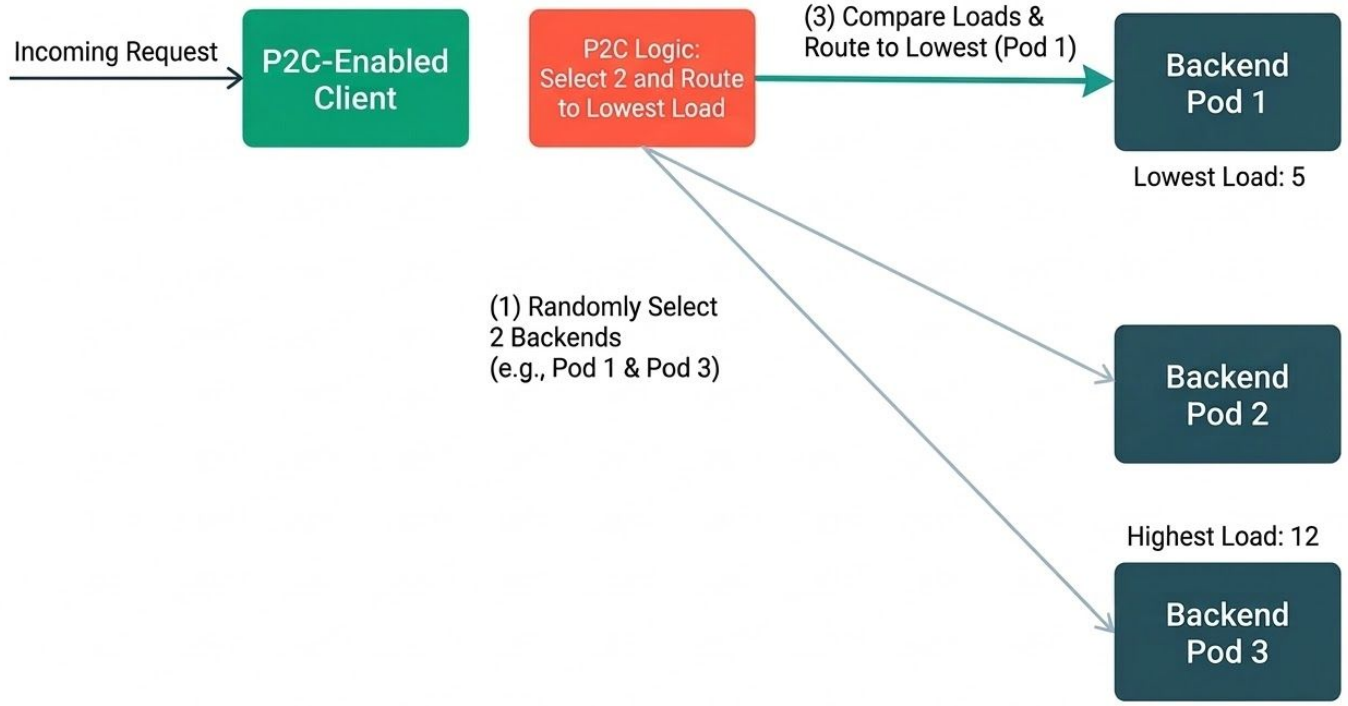
- Metrics Mismatch
  - Trailing indicator
  - Realtime signal frequency misalignment
- Noisy Neighbors
- Unrelated Source of Overload



CPU failed to capture everything we wanted to optimize for

Our Solution

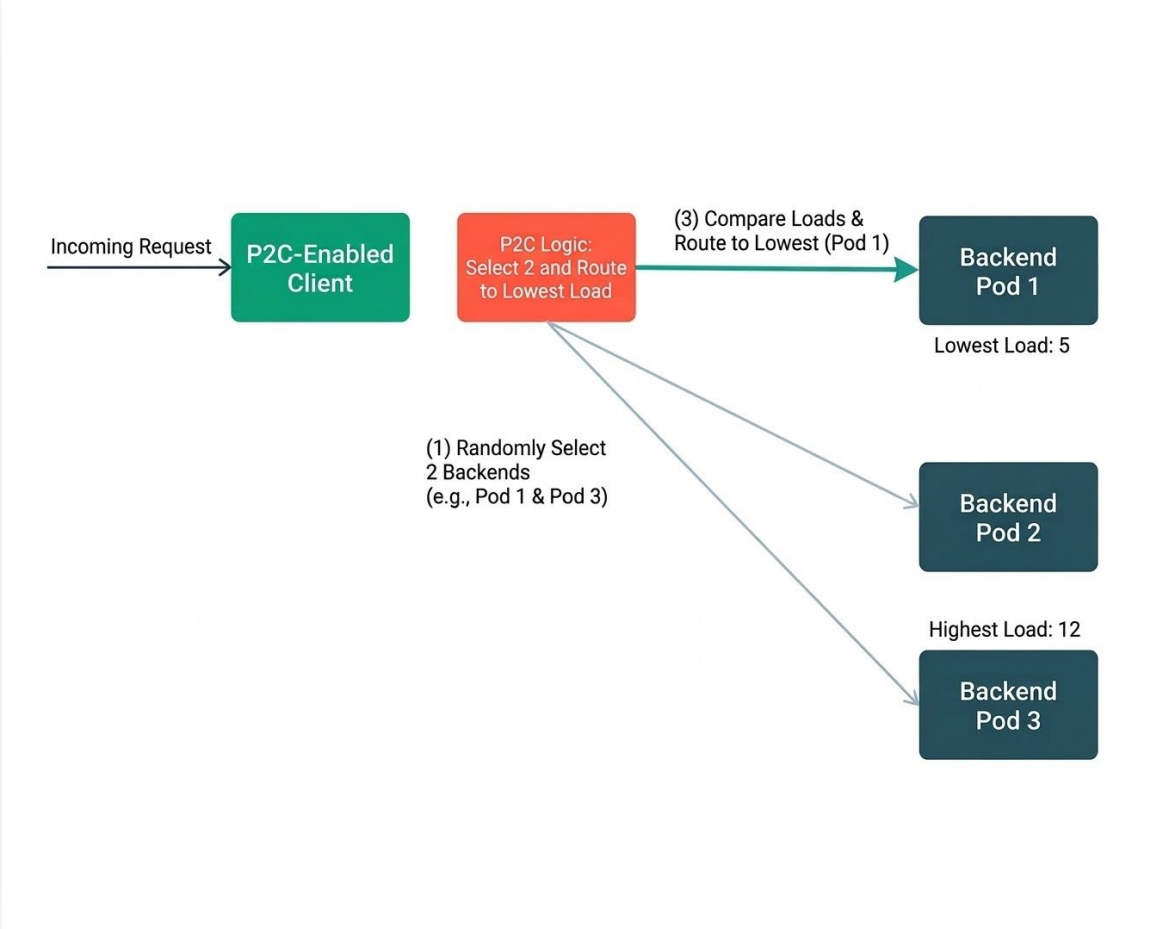
Power of Two Choices



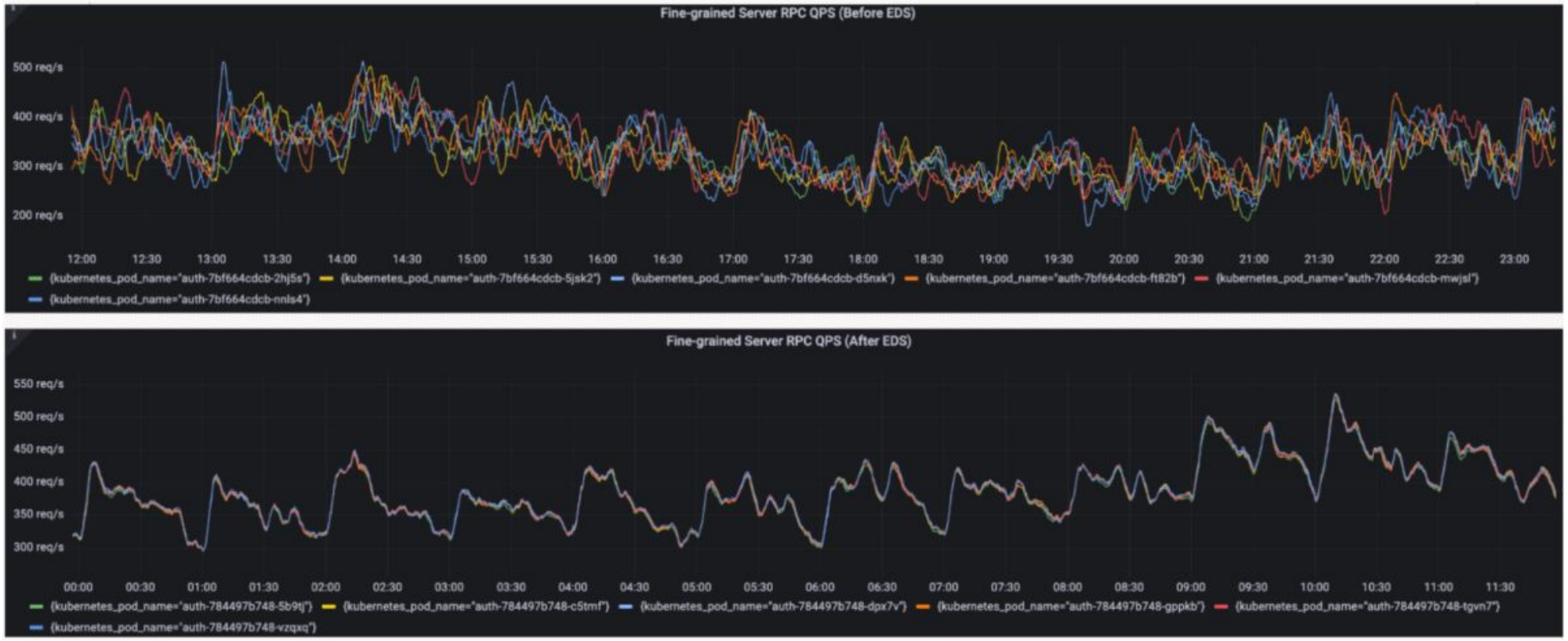
Our Solution

Power of Two Choices

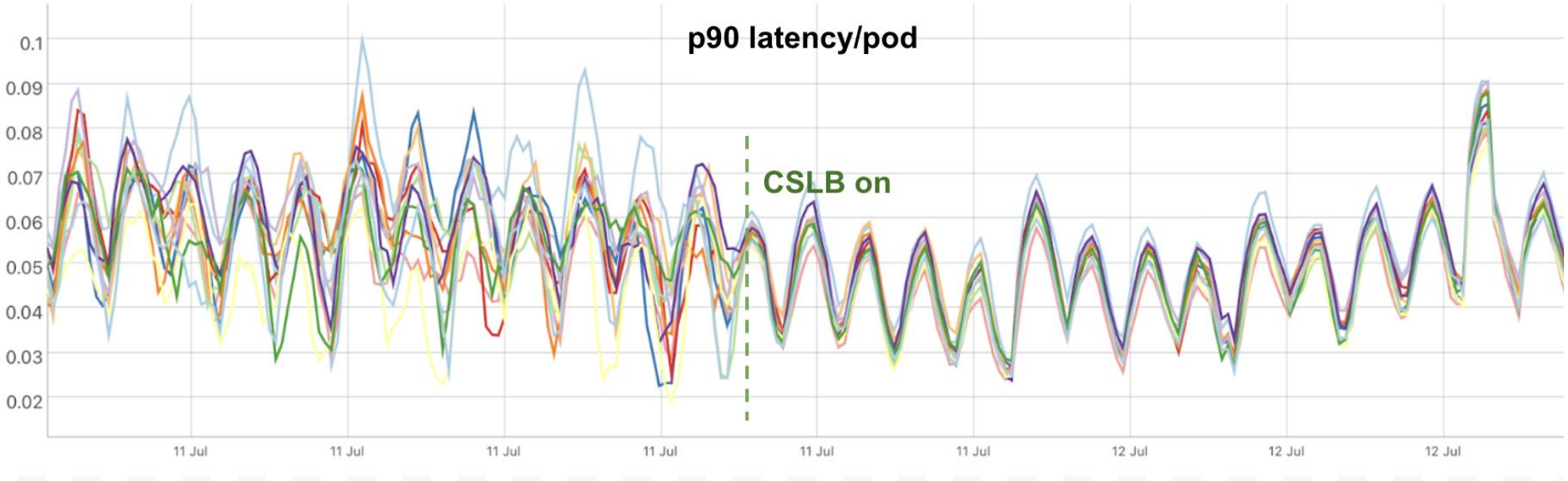
- Simpler, cheaper algorithm
- Easily extensible scoring
- Naturally avoided thundering herd



The Impact: Request Distribution

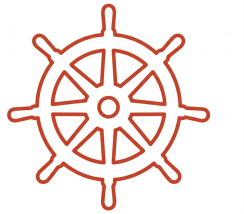


The Impact: Latency Distribution



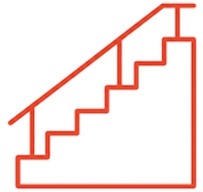
The Impact: Stability Improvements

Our Takeaways



Default Kubernetes load balancing does not scale

Understand your abstractions



Aim for incremental improvements

Q&A

Link to Blog!