KubeCon | CloudNativeCon
Europe 2022

Distributing PromQL for Fast and Efficient Kubernetes Fleet Monitoring

Moad Zardab

Senior Software Engineer

Red Hat

Filip Petkovski

Senior Production Engineer

Shopify

Agenda

- Why is PromQL difficult to scale?
- Query pushdown
- Query sharding
- Sharding in practice
- Outlook

Prometheus Overview

- Ubiquitous for Cloud Native monitoring
- Effective for real time monitoring
- Powerful query language

Ideal for **single cluster monitoring**

- Scraping across cluster boundaries is unreliable
- Relies on disk and memory for retention
- Lacking good scalability mechanisms

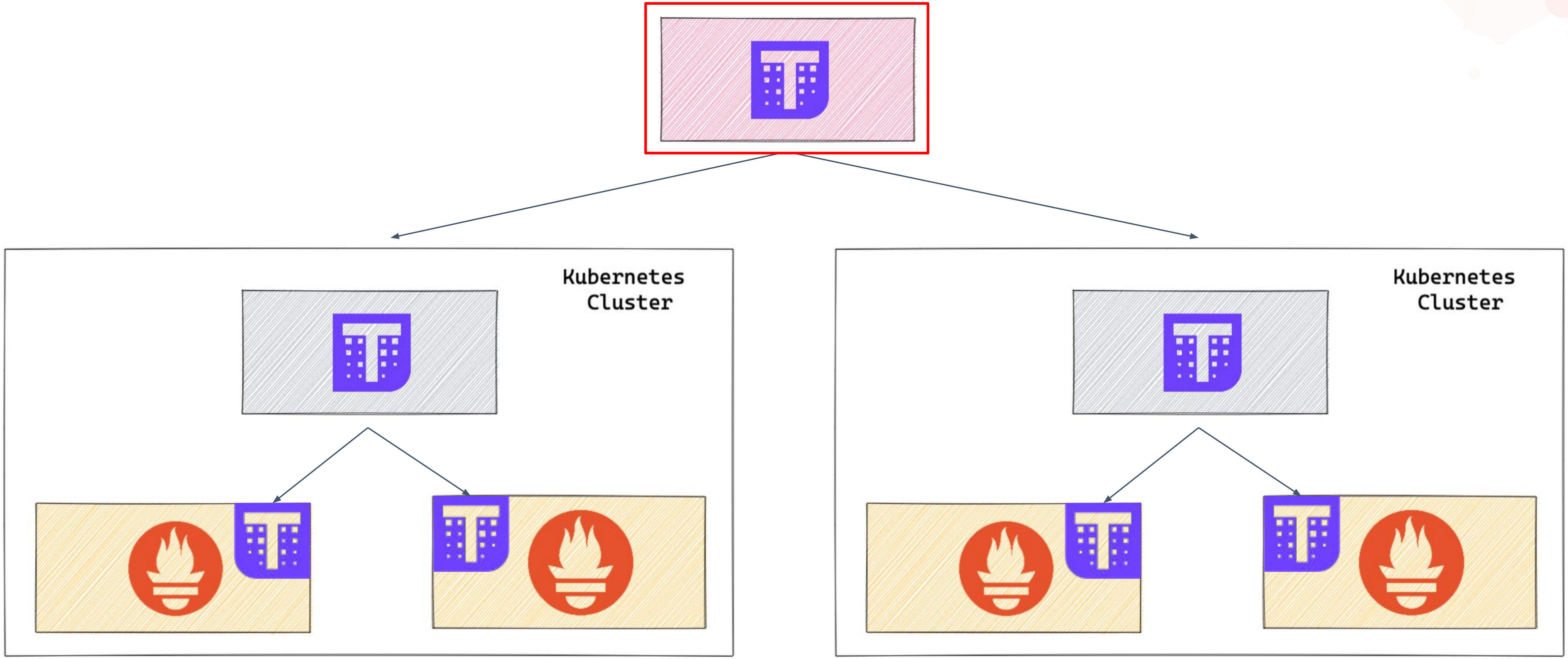
Larger Scale, Different Problems

- Projects like Thanos and Cortex gain mass adoption, enabling:
  - **Cheap**, **long term 'infinite retention'**
  - **Write replication** for redundancy
  - **Multi-tenancy** support
  - Prometheus-compliant **query API**

But…

- **PromQL engine** is still **single-threaded**
  - Needs all of your time-series data present before query evaluation
  - Exasperated by longer-term retention

Thanos Query Path Bottleneck



Bottlenecks on the with the Thanos query path?

- *Thanos-query* has to **stream all series into memory** in a single process
  - Query components need to **scale resource utilization with metric retention**
    - **GiBs worth of data over the network** and **into memory** for a **single query** that might return a single dimension (e.g. *topk, min/max)*
  - Makes component susceptible to ***OOM***

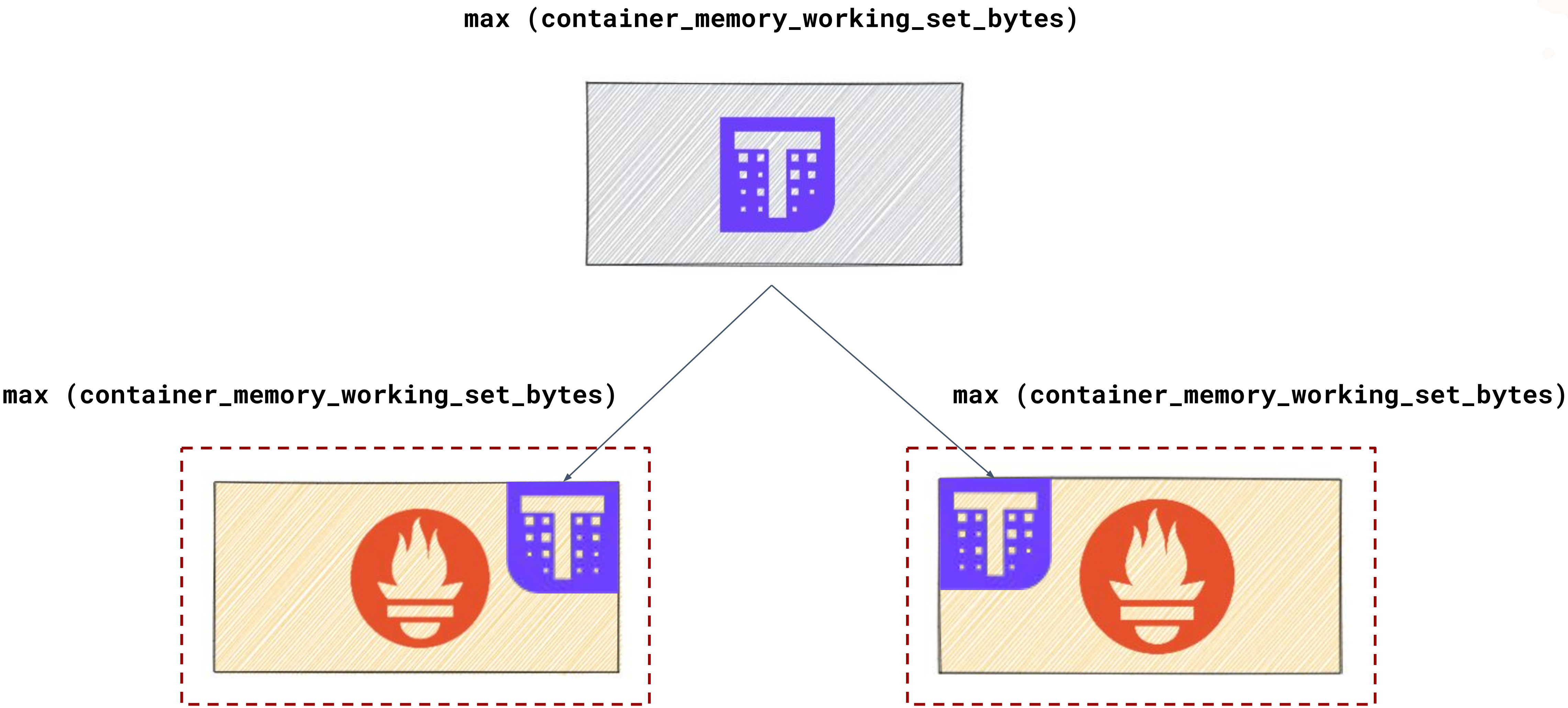
- *Thanos-query* has to fanout queries to all targets
  - Time series can be **duplicated**
  - Wasted resources on **deduplicating blocks**

The community was calling out for..

- More efficient load-balancing
- Better query component availability
- Reduce overall bandwidth required for query evaluation

Query pushdown; avoid streaming series all-together

Query Pushdown



max (container_memory_working_set_bytes)

max (container_memory_working_set_bytes)

max (container_memory_working_set_bytes)

Query Pushdown

Implemented in Thanos here https://github.com/thanos-io/thanos/pull/4917

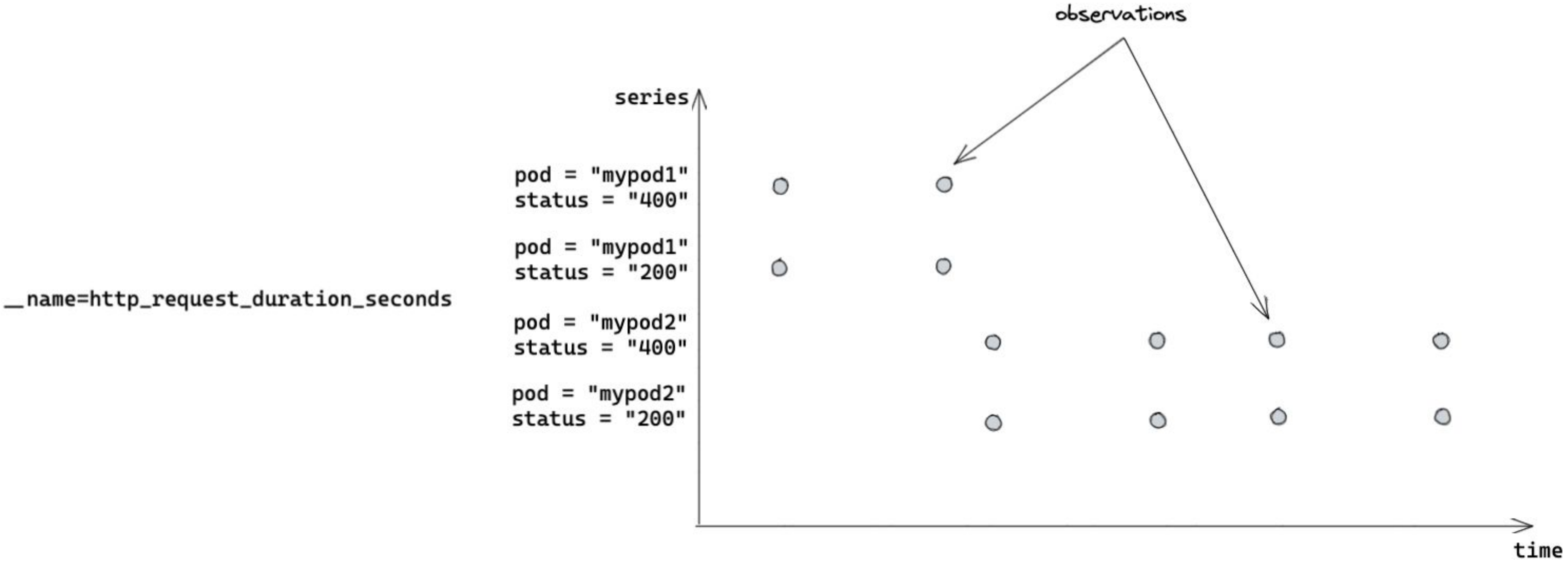
- Lower latency by
  - Processing data at-rest
  - Significantly reduced bandwidth
- Can be enabled with `--enable-feature=query-pushdown`

Limitations

- Currently applicable to: **max**, **`min`**, **`topk`**, **`bottomk`** and **`group`**
- Common operations like *`sum/rate`*, *`avg`*, *`histogram_quantile`* cannot be pushed down
- Not all store components can execute queries
- Query execution is not free

Query sharding; split a *single* query into *multiple distinct* and *disassociated* queries

Query Sharding



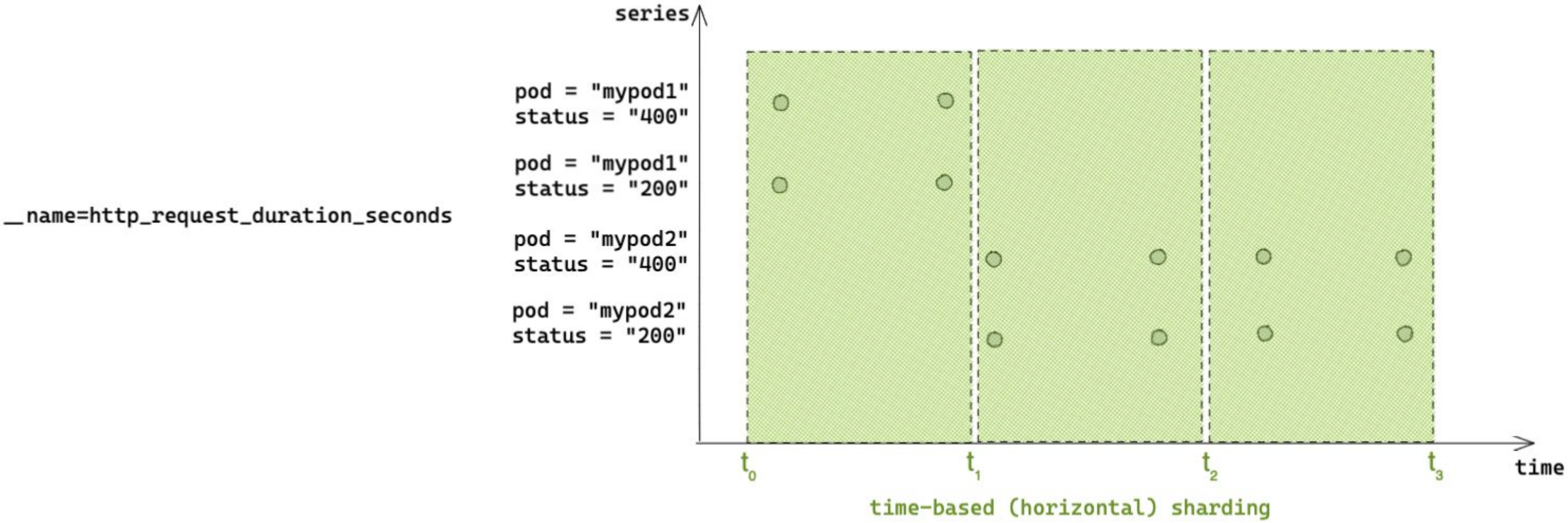
Query Sharding

Thanos already partially addresses this via **horizontal sharding**

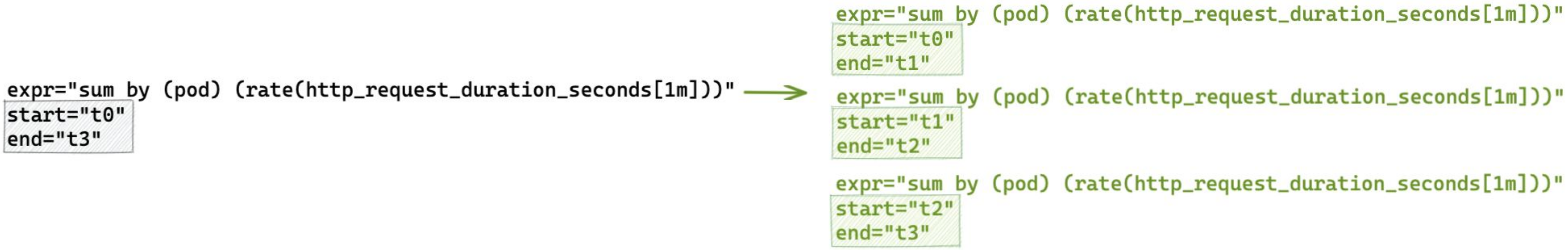
- **Time-range** query splitting
  - e.g. query spanning **four days** becomes **four, one day** queries
- **Only works** for range queries
- Even for short time intervals, metrics can still **have high cardinality**

Query Horizontal Sharding

```
sum by (pod) (rate(http_request_duration_seconds[1m]))
```



Query Horizontal Sharding



```
expr="sum by (pod) (rate(http_request_duration_seconds[1m]))"
start="t0"
end="t3"
```

```
expr="sum by (pod) (rate(http_request_duration_seconds[1m]))"
start="t0"
end="t1"

expr="sum by (pod) (rate(http_request_duration_seconds[1m]))"
start="t1"
end="t2"

expr="sum by (pod) (rate(http_request_duration_seconds[1m]))"
start="t2"
end="t3"
```

- **One query** has been split into **N equivalent range queries (***where N is the total query time range / query range interval)*
- Each query represents a **disassociated time range**
- *But…* new queries can still have **unbounded cardinality** for the given time range

How do we manage metric cardinality?

Query Sharding

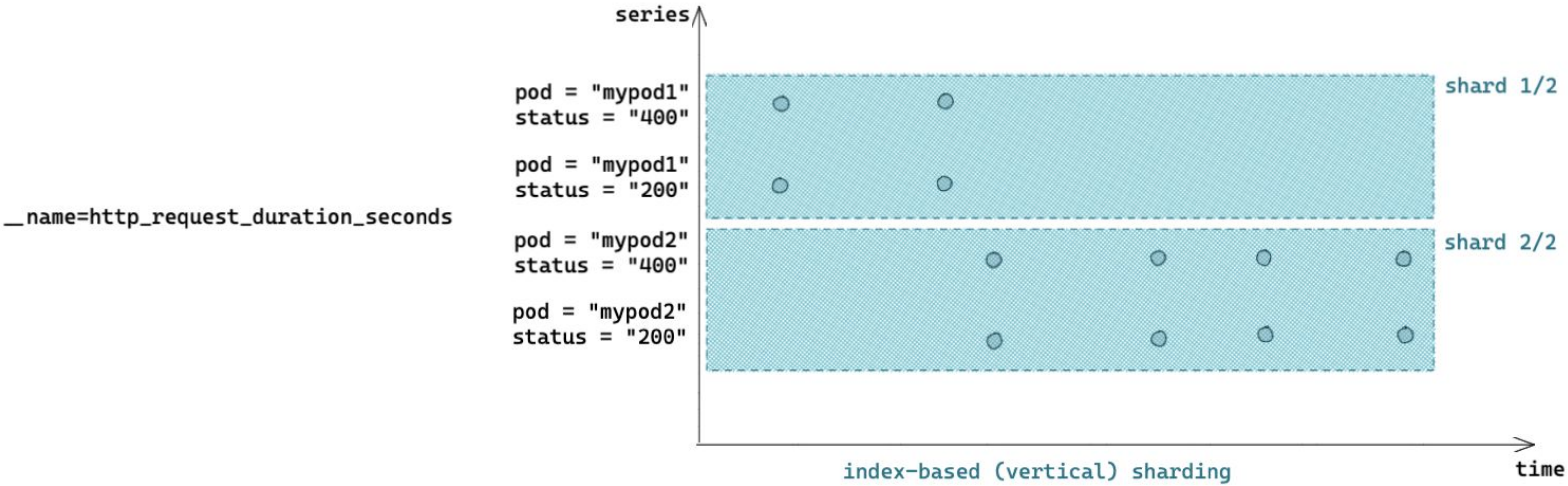
We worked on extending this with **vertical sharding:**

- Sharding based on **time series**
- Each shard is **dissociated**
- Can be implemented **without** query planning ahead-of-time
  - Leaves are responsible for **deterministically** returning sharded subset
- No changes to the Thanos write path

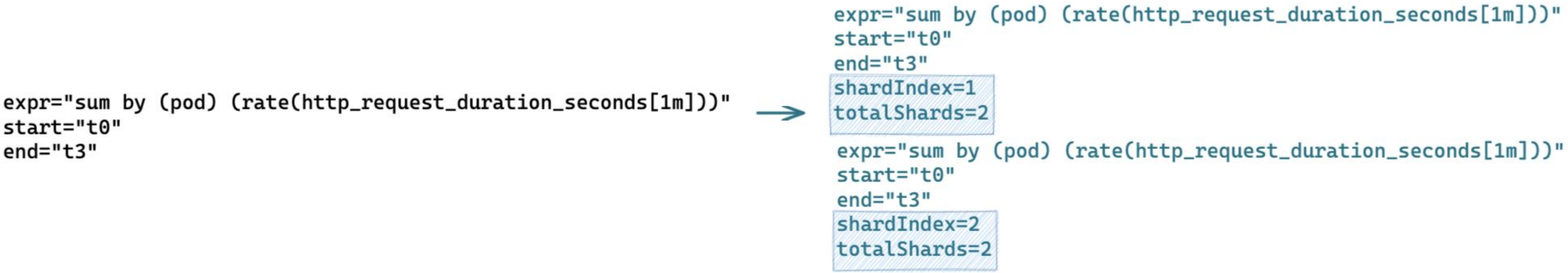
We wanted to test this first by sharding **grouped sum/rate expressions**

Query Vertical Sharding

```
sum by (pod) (rate(http_request_duration_seconds[1m]))
```



Query Vertical Sharding



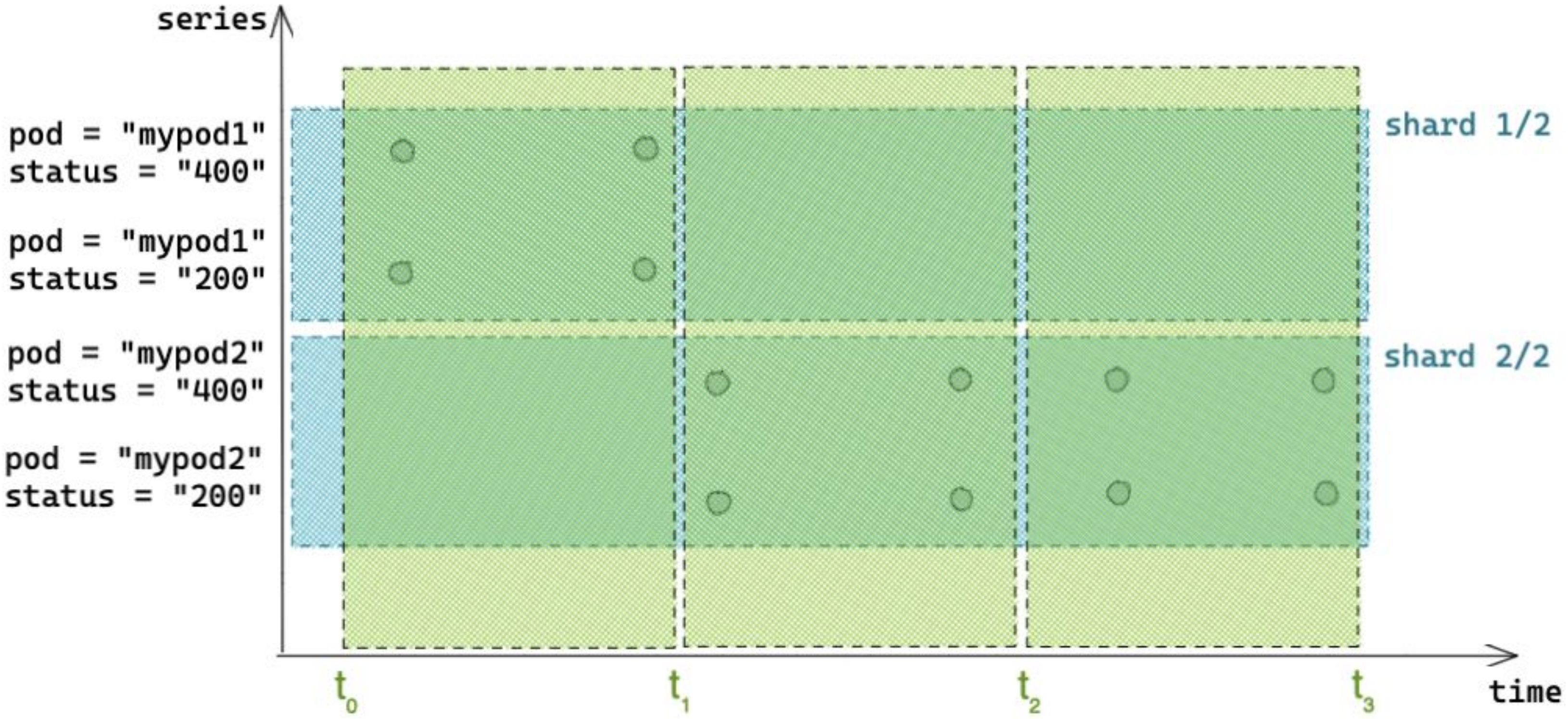
```
expr="sum by (pod) (rate(http_request_duration_seconds[1m]))"
start="t0"
end="t3"
```

```
expr="sum by (pod) (rate(http_request_duration_seconds[1m]))"
start="t0"
end="t3"
shardIndex=1
totalShards=2
```

```
expr="sum by (pod) (rate(http_request_duration_seconds[1m]))"
start="t0"
end="t3"
shardIndex=2
totalShards=2
```

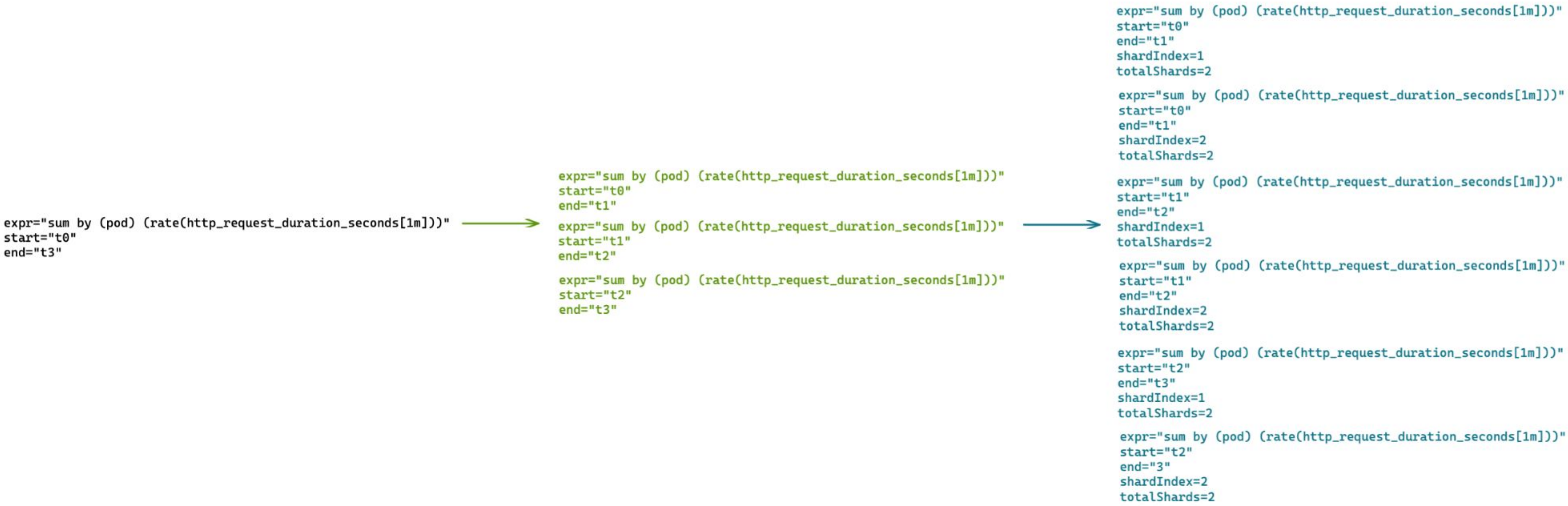
Query Vertical & Horizontal Sharding

__name=http_request_duration_seconds



combined horizontal & vertical sharding

Query Vertical & Horizontal Sharding



```
expr="sum by (pod) (rate(http_request_duration_seconds[1m]))"
start="t0"
end="t3"
```

```
expr="sum by (pod) (rate(http_request_duration_seconds[1m]))"
start="t0"
end="t1"

expr="sum by (pod) (rate(http_request_duration_seconds[1m]))"
start="t1"
end="t2"

expr="sum by (pod) (rate(http_request_duration_seconds[1m]))"
start="t2"
end="t3"
```

```
expr="sum by (pod) (rate(http_request_duration_seconds[1m]))"
start="t0"
end="t1"
shardIndex=1
totalShards=2

expr="sum by (pod) (rate(http_request_duration_seconds[1m]))"
start="t0"
end="t1"
shardIndex=2
totalShards=2

expr="sum by (pod) (rate(http_request_duration_seconds[1m]))"
start="t1"
end="t2"
shardIndex=1
totalShards=2

expr="sum by (pod) (rate(http_request_duration_seconds[1m]))"
start="t1"
end="t2"
shardIndex=2
totalShards=2

expr="sum by (pod) (rate(http_request_duration_seconds[1m]))"
start="t2"
end="t3"
shardIndex=1
totalShards=2

expr="sum by (pod) (rate(http_request_duration_seconds[1m]))"
start="t2"
end="3"
shardIndex=2
totalShards=2
```

- **One query** has been split into **N x M equivalent vertically** and **horizontally sharded queries** (*where N is the total time range/ range interval, and M is the total number of vertical shards*)
- Each subquery can be **pre-evaluated in parallel**
- Result is simply concatenation or re-evaluation of intermediate queries in the root querier

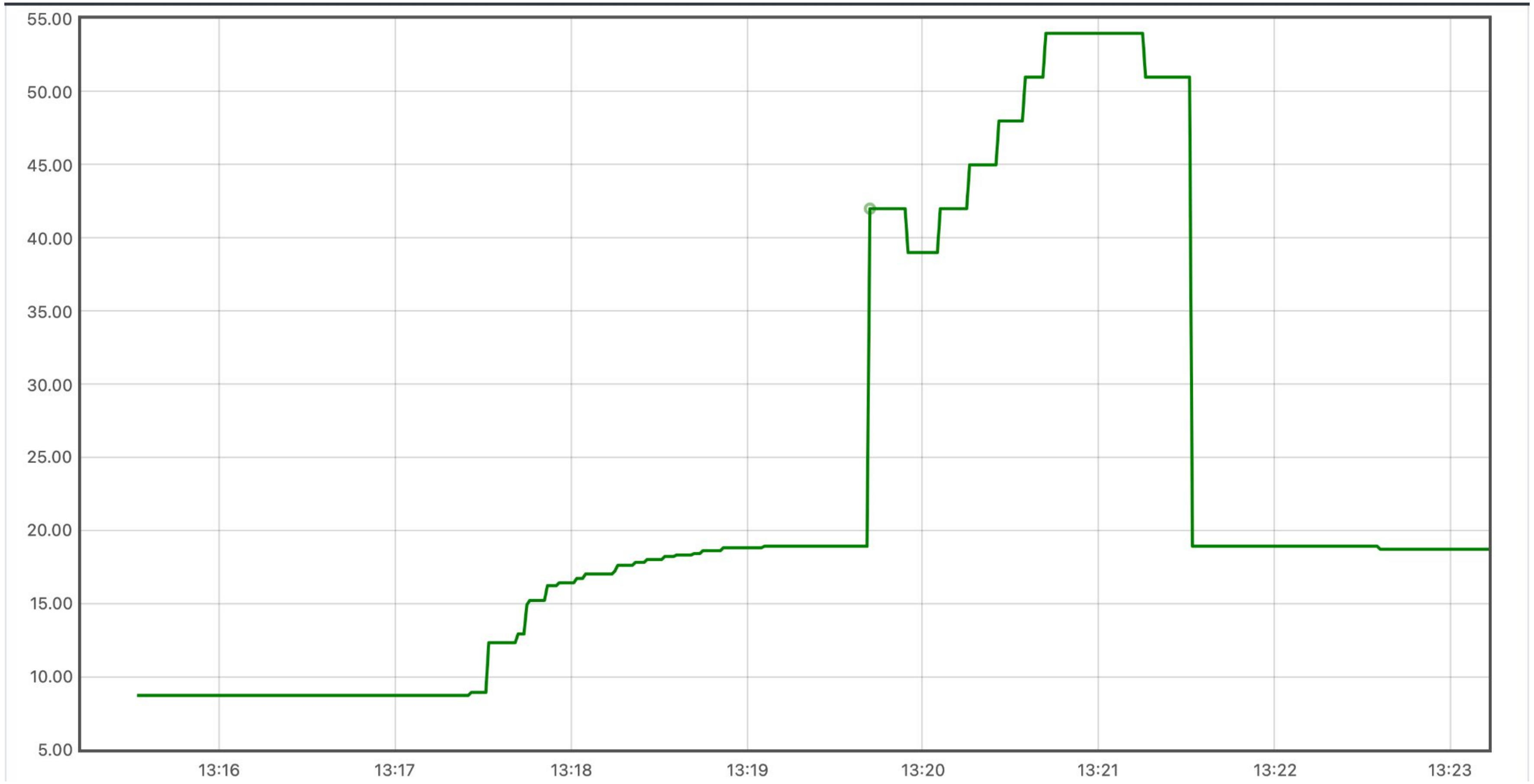
Vertical sharding performance

- Benchmark setup:
  - **100K** series, simulating **100 clusters** with **1000 pods** each
  - Executed one query consecutively for a fixed interval against 5 Queriers
  - Used a **sharding factor of 3**
  - Single node setup

- What we aimed to measure:
  - **User experience impact** (query latency)
  - Impact on **overall resiliency** (peak and average memory usage)

Query Latency (*without* sharding)

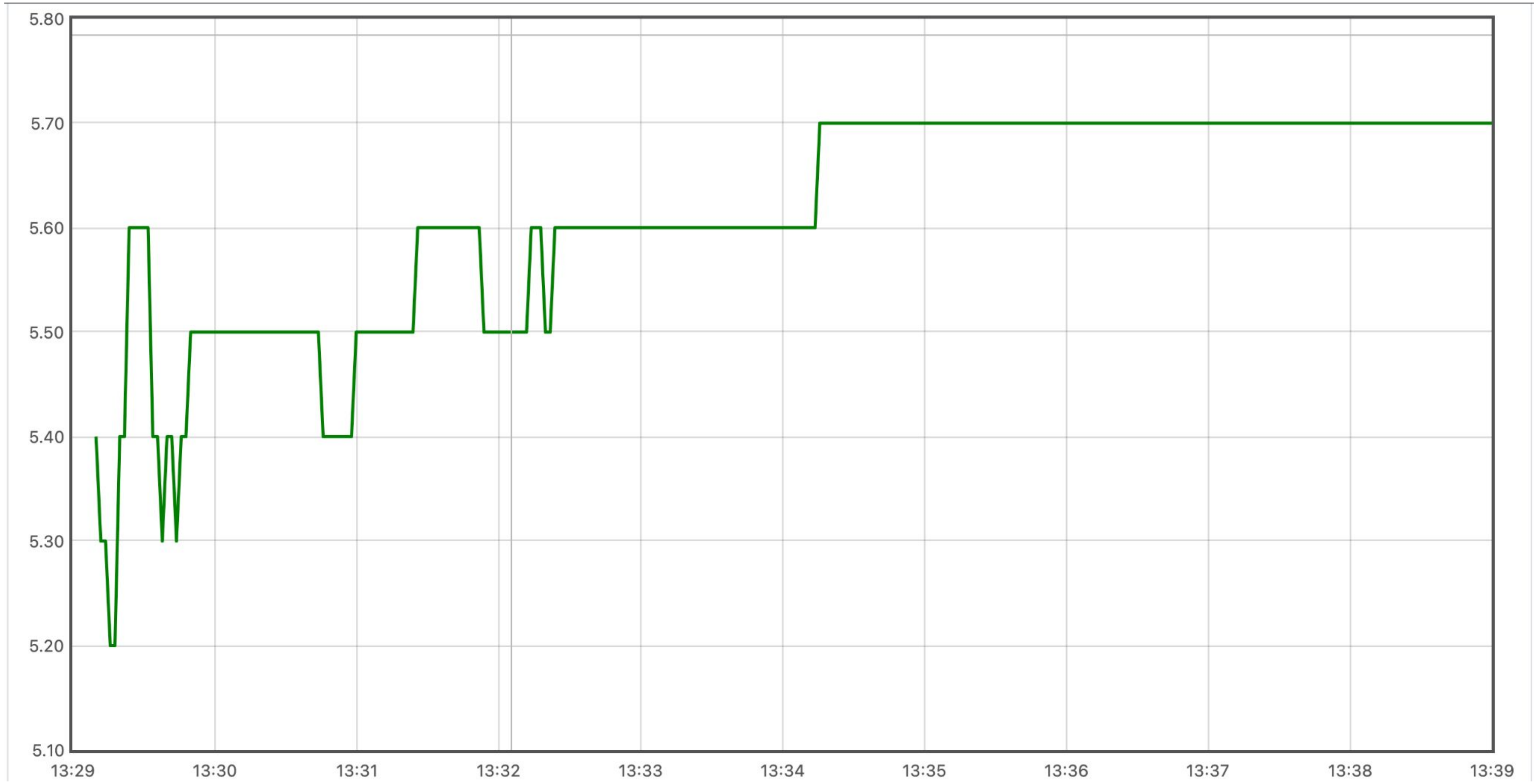
P90 latency



Start latency ~9s
End latency ~19s

Query Latency (*with* sharding)

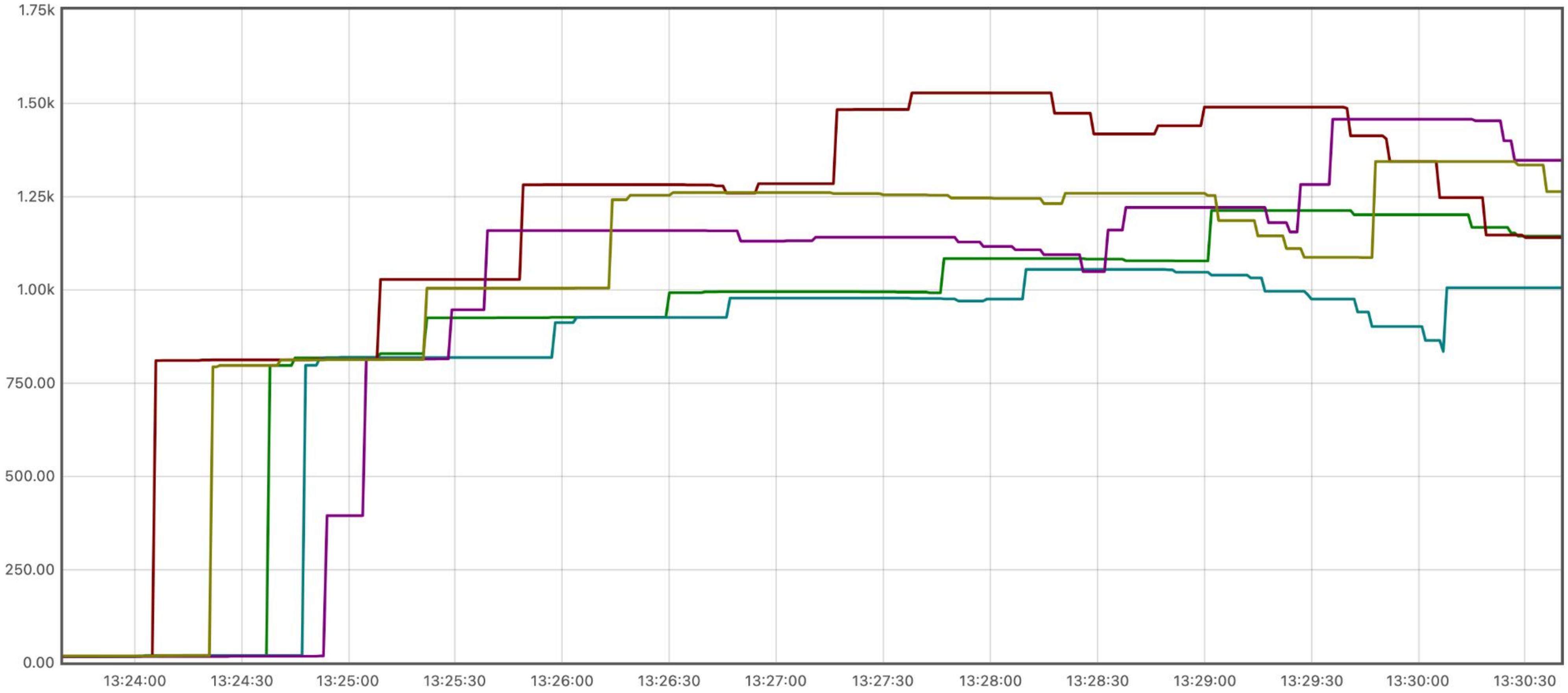
P90 latency



Start latency ~5.2s **(-43%)**
End latency ~5.7s **(-70%)**

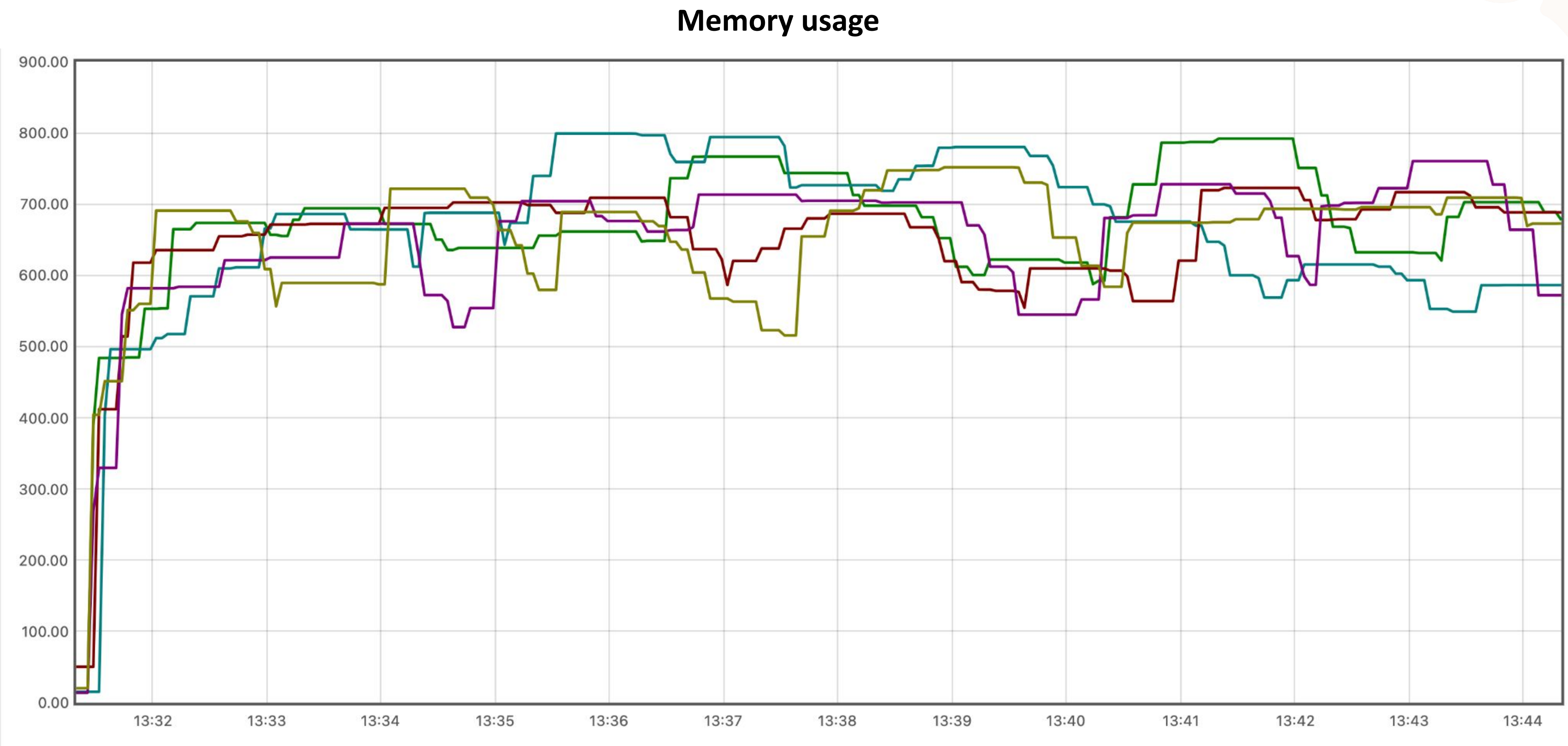
Memory Usage (*without* sharding)

Memory usage



Peak ~1.5GB
Average ~1.2GB

Memory Usage (*with* sharding)



Peak ~800MB **(-47%)**
Average ~700MB **(-42%)**

Vertical Sharding in Summary

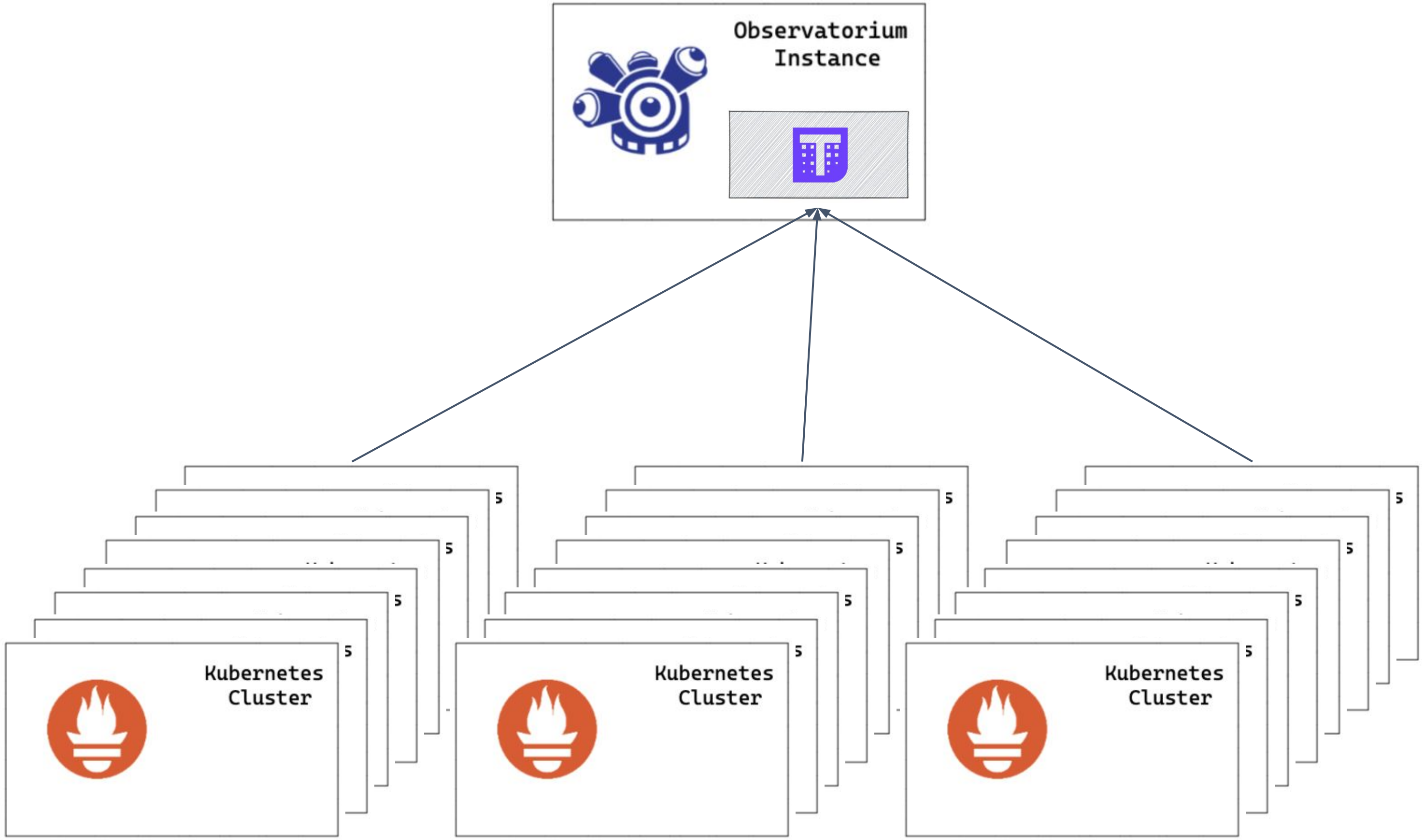
Benefits

- Greatly reduced peak memory utilization
- Query latency reduced by double digit percentage
- Applicable to instant queries (including alerting and recording rules)

Caveats

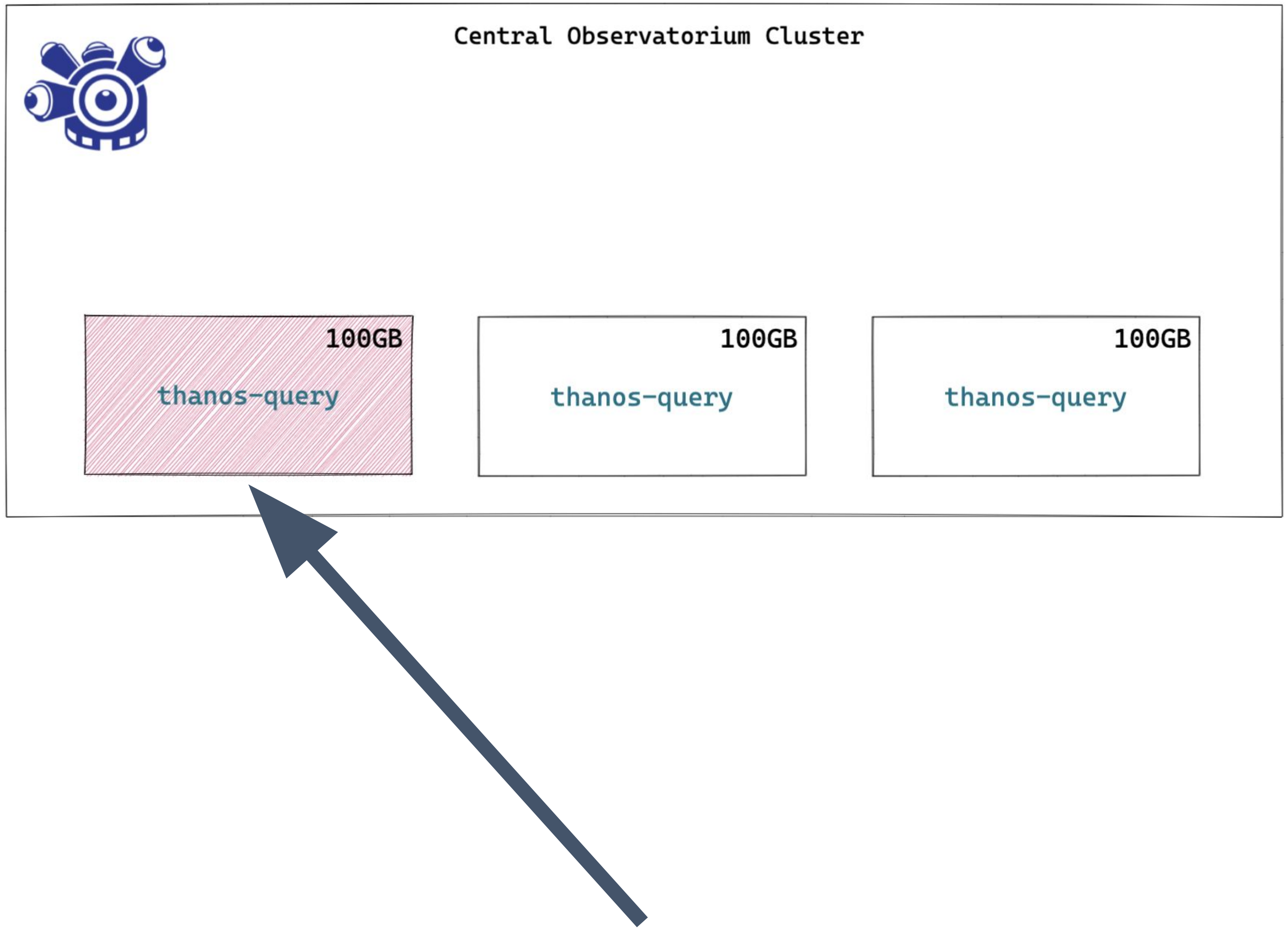
- Currently only implemented for PromQL aggregations
- Increased request volume throughout the system

Demonstration: Fleet Monitoring



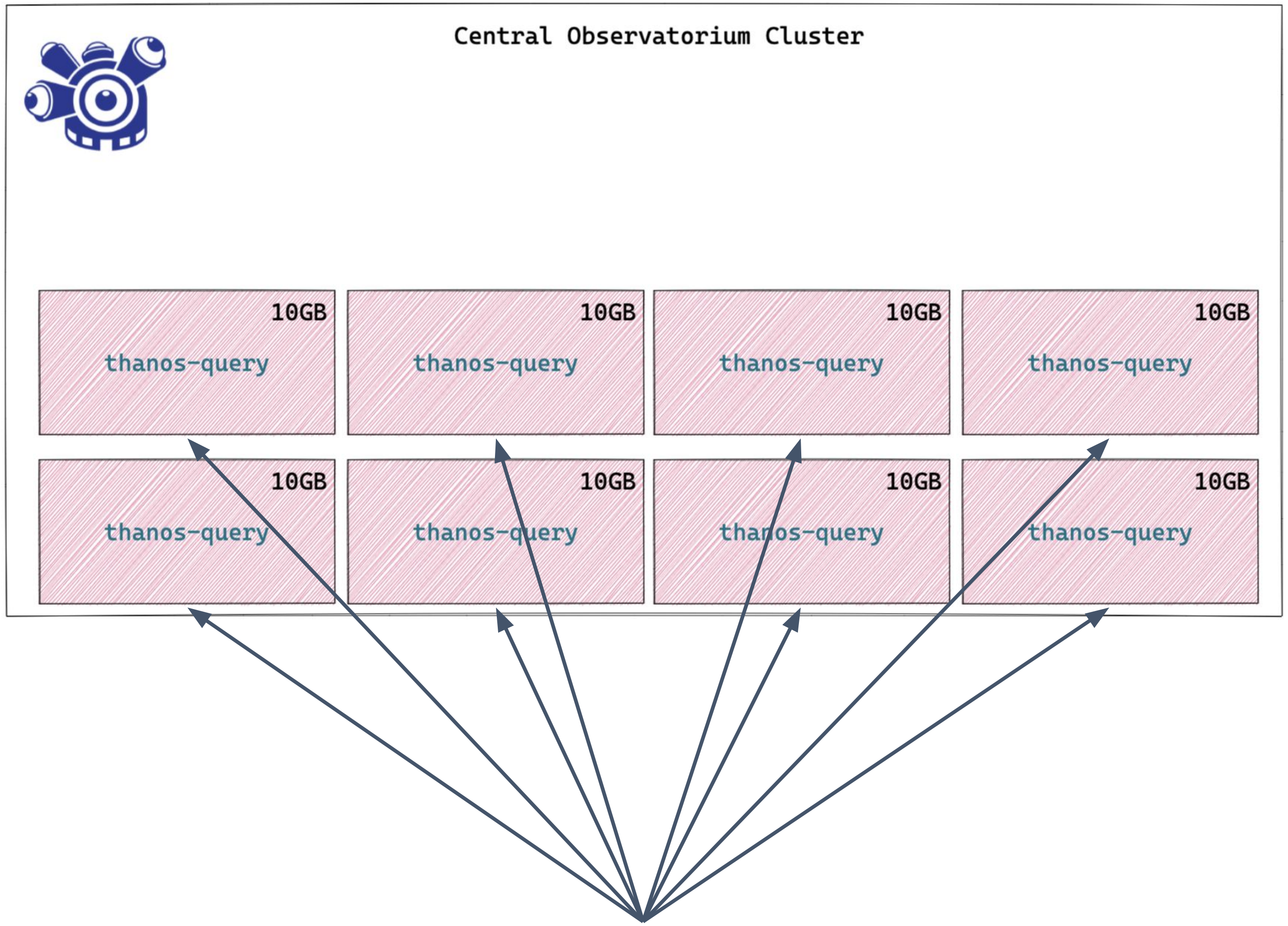
20,000 clusters, 1000 series/cluster, 30 day retention

Case Study: Observability-as-a-Service Query Load-balancing



`sum by (cluster_id) (container_memory_working_set_bytes)`

Case Study: Observability-as-a-Service Query Load-balancing



```
sum by (cluster_id) (container_memory_working_set_bytes)
```

Query Sharding Future

- Proposal in upstream Thanos https://github.com/thanos-io/thanos/pull/5350
- Reference implementation in upstream: Thanos https://github.com/thanos-io/thanos/pull/5342
- Expanding PromQL support to cover more expressions

More context:

- The Prometheus TSDB, Björn Rabenstein
- Intro to Thanos: Scale Your Prometheus Monitoring With Ease, Lucas Serven & Dominic Green
- Using Thanos to gain a unified way to query over multiple clusters, Wiard van Rij
- Thanos documentation

Thanks!

- Thanos community + maintainers

fin