Maximizing M3:

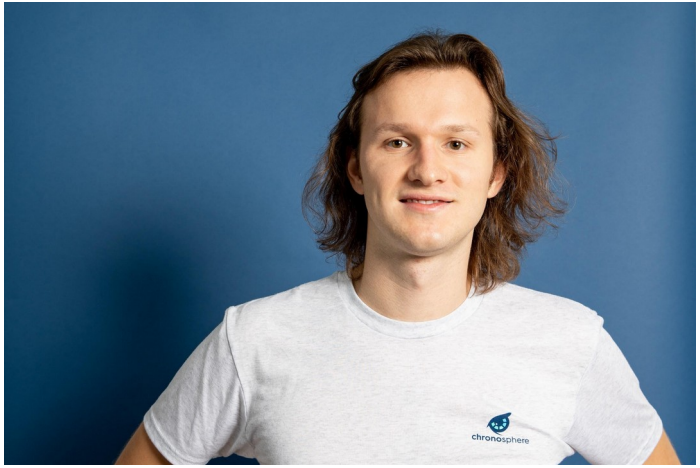
Pushing performance boundaries in a distributed metrics engine

CNCF Member Webinar 2020

/usr/bin/whoami

Ryan Allen

- Senior Software Engineer @ Chronosphere
- Formerly at Applied Predictive Technologies (APT)
- LinkedIn



chronosphere

Monitoring: what is a metric?

Schema for data you would like to collect and aggregate

Name

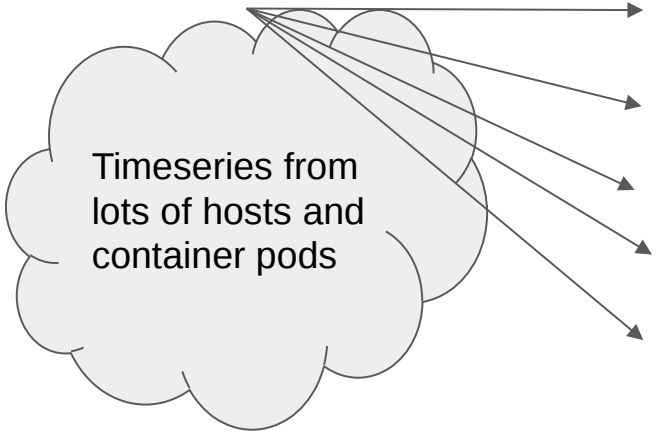
- http_requests

Dimensions/Labels

- endpoint (e.g. /api/search)
- status_code (e.g. 500)
- deploy_version_git_sha (e.g. 25149a04c)

chronosphere

Problem 1: Scale



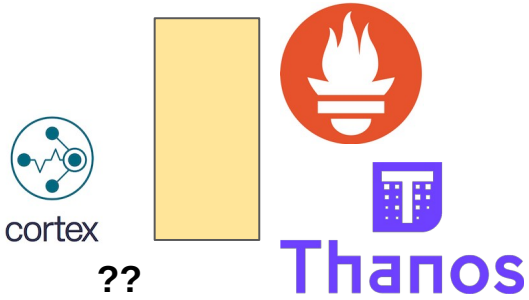
ID	Timeseries
1	__name__=cpu_seconds_total, pod=foo-123abc
8	__name__=memory_memfree, pod=foo-123abc
33	__name__=cpu_seconds_total, pod=foo-456def
44	__name__=memory_memfree, pod=foo-456def
45	__name__=cpu_seconds_total, pod=bar-768ghe
58	__name__=memory_memfree, pod=bar-768ghe
	… millions .. and if you are unfortunate... billions

chronosphere

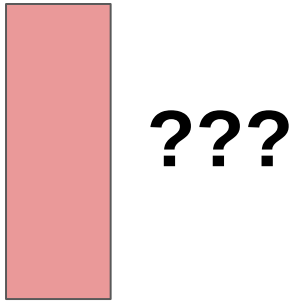
Problem 1: Scale

1. We have monitoring, it's awesome and developers are happy with standardized metrics mostly.

2. Developers put custom metrics on everything and I am deploying tons of applications in something like Kubernetes, things are ok!

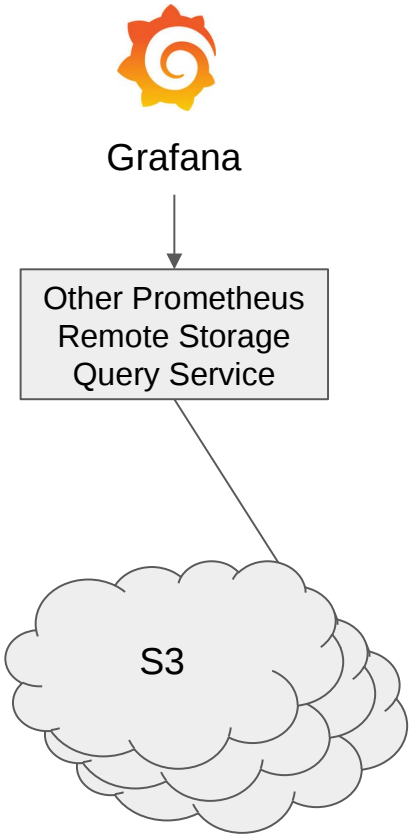


3. Things are on way too on fire, we can't manage this many things anymore, can everyone just stop please.



Problem 1: Scale

Single node execution
Read all index segments relevant to this time window and data chunks for time windows included by query, if too much index data then can't hold it entirely in memory.



chronosphere

Problem 2: Reliability

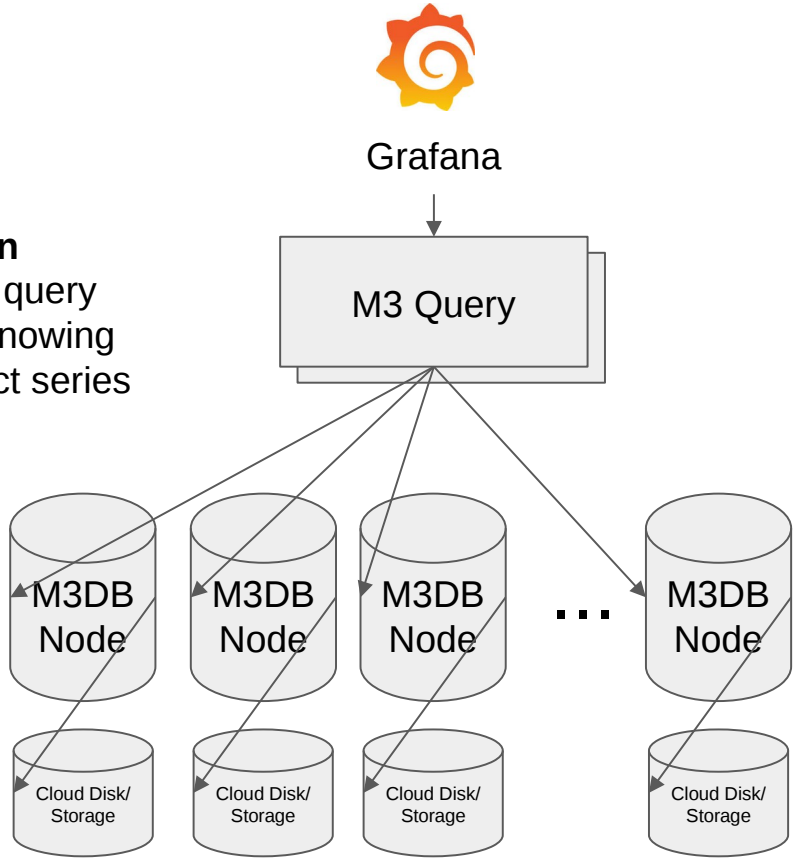
- Single node metrics solutions susceptible to outages

- Metrics monitoring infrastructure cannot go down... otherwise you lose visibility into your actual infrastructure

chronosphere

M3: Scale

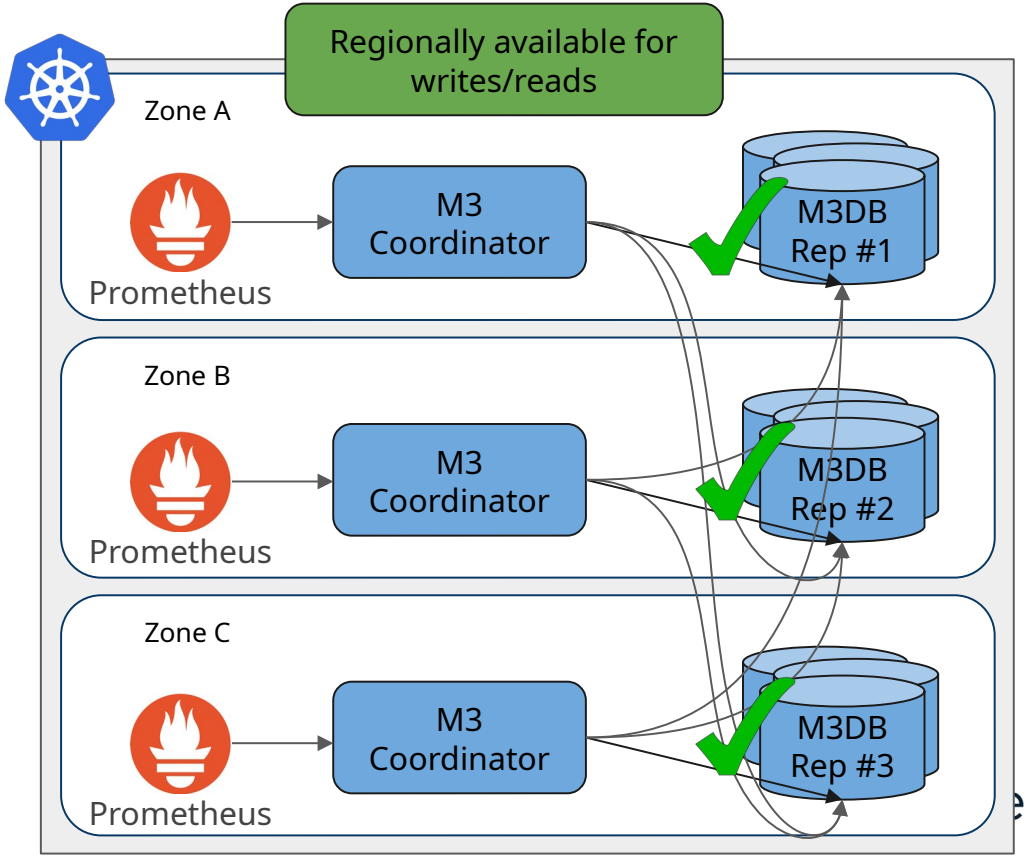
Map/reduce execution
Find metrics matching query and return in parallel knowing exactly where to extract series data from local store.



chronosphere

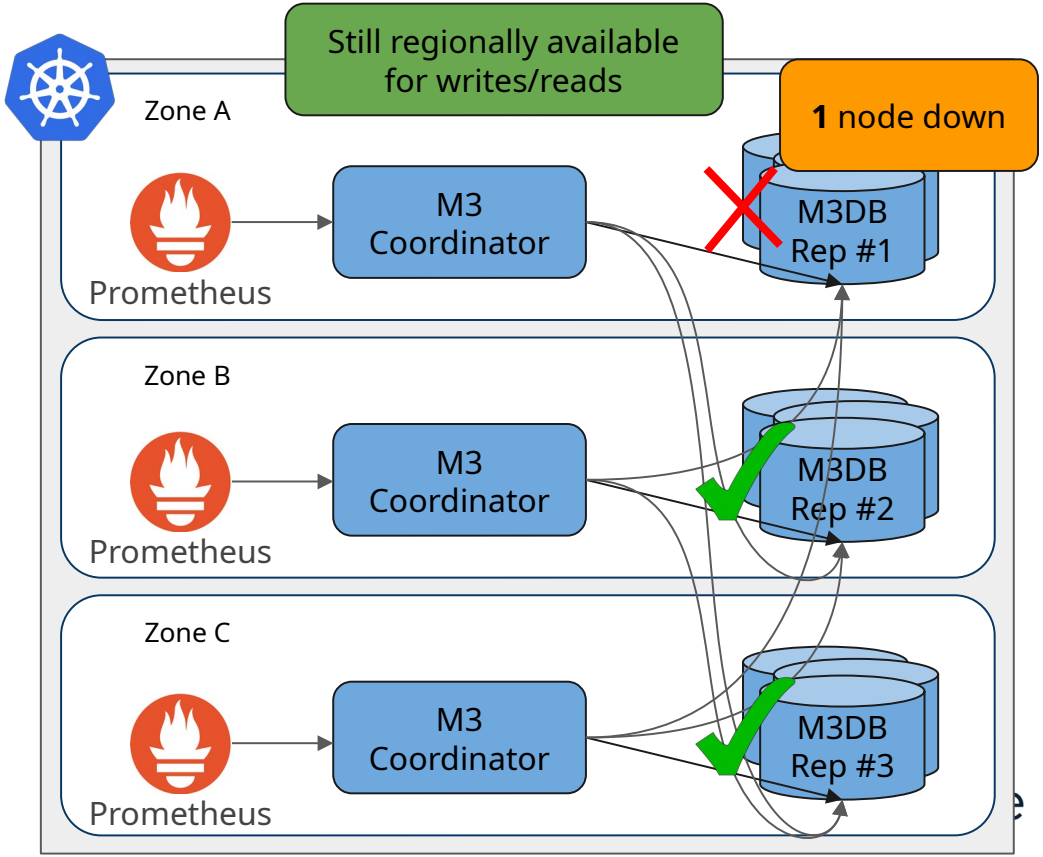
M3: Reliability

- Zero-downtime upgrades, restarts, withstand entire replica failures
- Able to be deployed multi-region and join data at query time across regions transparently
- At Uber warehouses more than 1PB of metrics data (10B series), storing 40million datapoints per second (3.4 trillion a day)



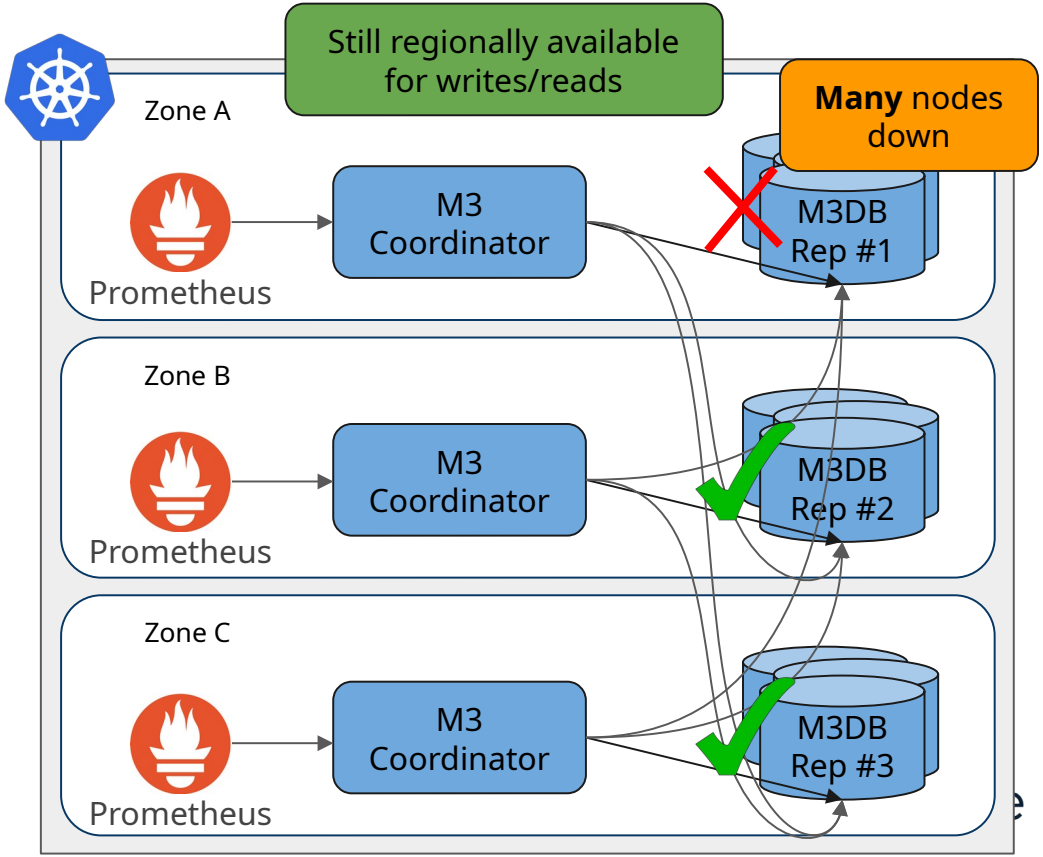
M3: Scale & Reliability

- Zero-downtime upgrades, restarts, withstand entire replica failures
- Able to be deployed multi-region and join data at query time across regions transparently
- At Uber warehouses more than 1PB of metrics data (10B series), storing 40million datapoints per second (3.4 trillion a day)



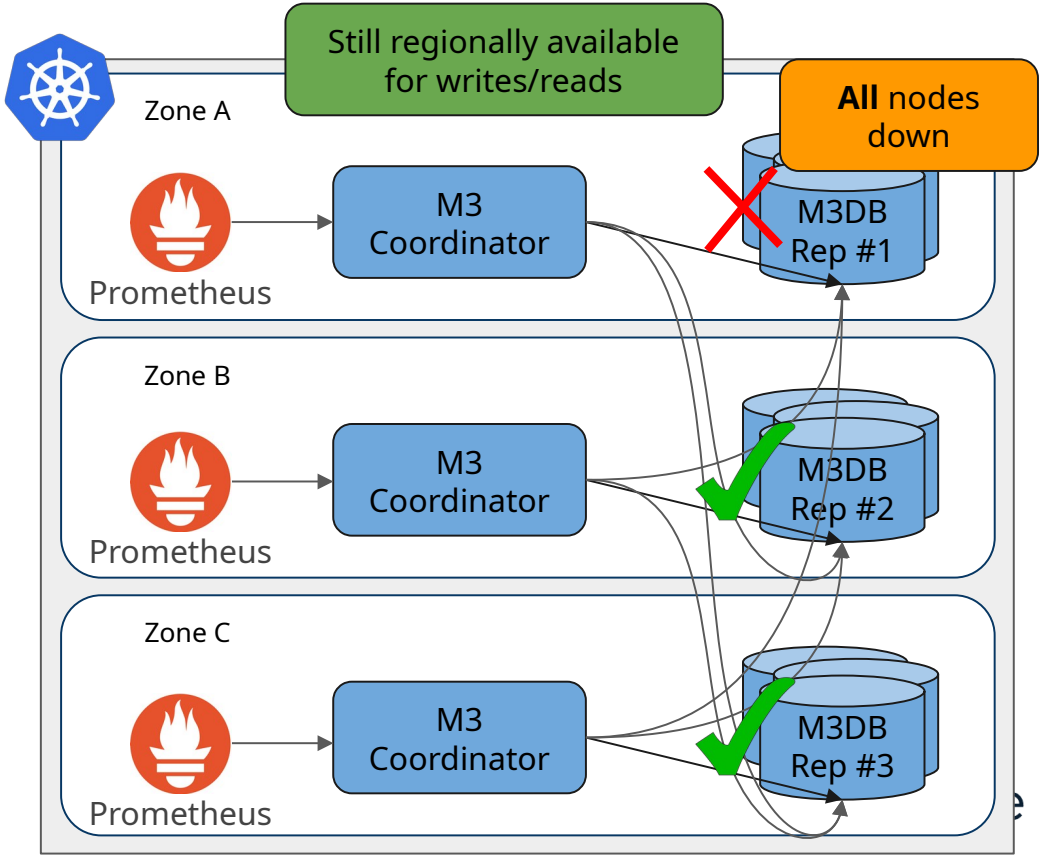
M3: Scale & Reliability

- Zero-downtime upgrades, restarts, withstand entire replica failures
- Able to be deployed multi-region and join data at query time across regions transparently
- At Uber warehouses more than 1PB of metrics data (10B series), storing 40million datapoints per second (3.4 trillion a day)



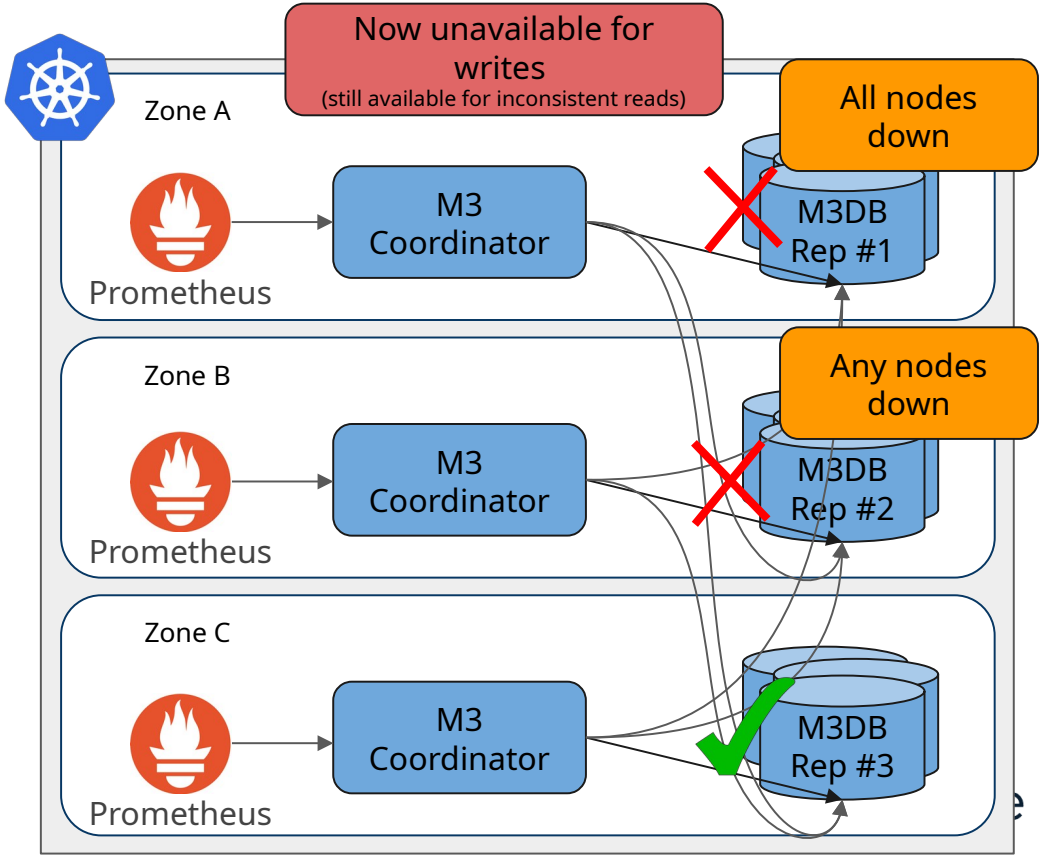
M3: Scale & Reliability

- Zero-downtime upgrades, restarts, withstand entire replica failures
- Able to be deployed multi-region and join data at query time across regions transparently
- At Uber warehouses more than 1PB of metrics data (10B series), storing 40million datapoints per second (3.4 trillion a day)



M3: Scale & Reliability

- Zero-downtime upgrades, restarts, withstand entire replica failures
- Able to be deployed multi-region and join data at query time across regions transparently
- At Uber warehouses more than 1PB of metrics data (10B series), storing 50million datapoints per second (3.4 trillion a day)



New Problem

We now have “infinite” horizontal scale via distributed storage… so now we can scale indefinitely, right??

Blocker (or at least dissuader) no longer is **PHYSICS**…

but rather **$$$**.

And also… cardinality only continues to grow with modern times

chronosphere

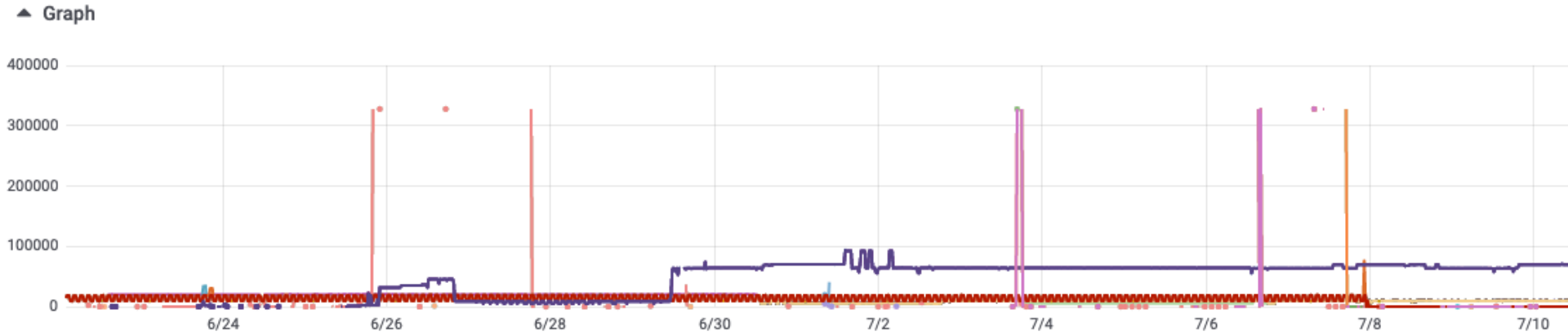
Maximizing perf / $

Maximizing metric / compute

chronosphere

Maximization Problem A: Query Load

Query load is unpredictable and spiky...



How do we optimally resource plan (and not pay just to support the rare spikes)?

chronosphere

Query Stats & Limits

- We don’t want to have to provision M3 to support **worst-case** scenarios (needless cost)
- Instead,
  - Provide engineers **controls to tune clusters** for optimal usage (i.e. pay up to the need, and no more)
  - Provide engineers **transparency into usage** and how their clusters tend to behave to inform optimal tuning

chronosphere

Limit Memory Globally

- M3 automatically surfaces histogram metrics on concurrent memory use from queries per m3db node

chronosphere

Limit Memory Globally

- Configure m3db nodes to enforce custom limits (docs)

```
limits:
  # If set, will enforce a maximum cap on time series blocks matched for
  # queries searching time series by dimensions.
  maxRecentlyQueriedSeriesBlocks:
    # Value sets the maximum time series blocks matched, use your block
    # settings to understand how many datapoints that may actually translate
    # to (e.g. 2 hour blocks for unaggregated data with 30s scrape interval
    # will translate to 240 datapoints per single time series block matched).
    value: 0
    # Lookback sets the time window that this limit is enforced over, every
    # lookback period the global count is reset to zero and when the limit
    # is reached it will reject any further time series blocks being matched
    # and read until the lookback period resets.
    lookback: 5s
```

chronosphere

Limit Memory per Query

- M3 automatically surfaces histogram metrics on per-query results sizes

chronosphere

Limiting Max Memory per Query

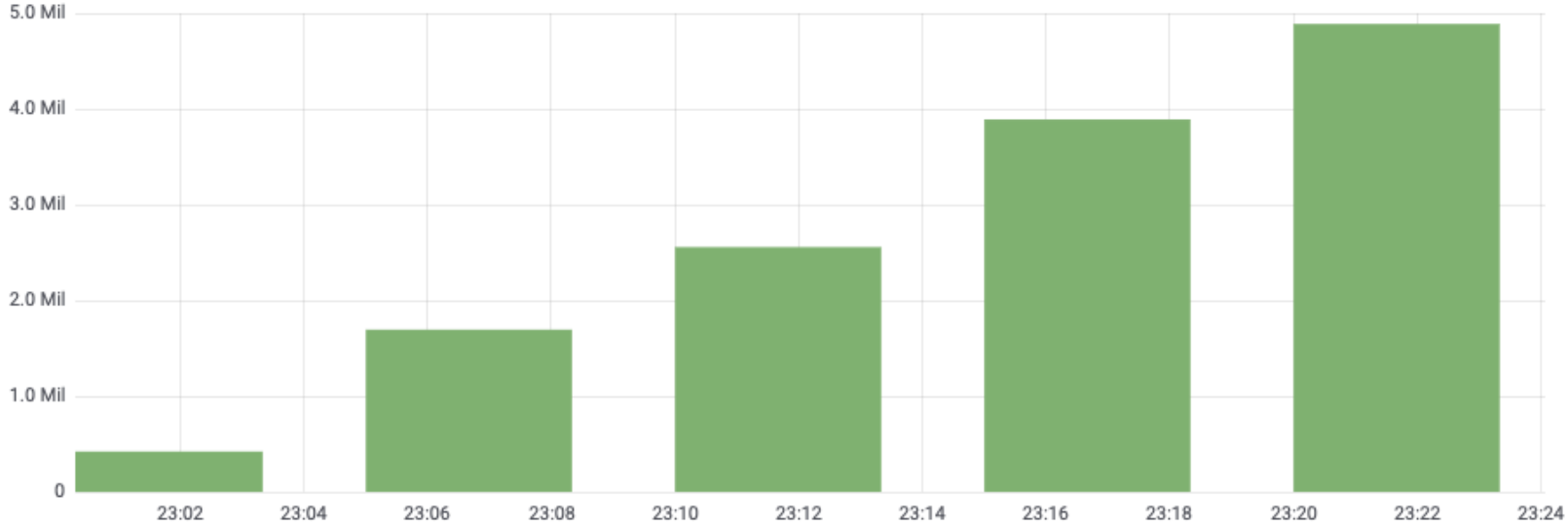
- Configure m3db nodes to enforce custom limits per query (docs)

```
limits:
  # If set will override default limits set per query.
  perQuery:
    # If set limits the number of time series returned for any given
    # individual storage node per query, before returning result to query
    # service.
    maxFetchedSeries: 0

    # If set limits the number of index documents matched for any given
    # individual storage node per query, before returning result to query
    # service.
    # This equates to the number of time series * number of blocks, so for
    # 100 time series matching 4 hours of data for a namespace using a 2 hour
    # block size, that would result in matching 200 index documents.
    maxFetchedDocs: 0

    # If true this results in causing a query error if the query exceeds
    # the series or blocks limit for any given individual storage node per query.
    requireExhaustive: false
```

chronosphere

Maximization Problem B: Write Load

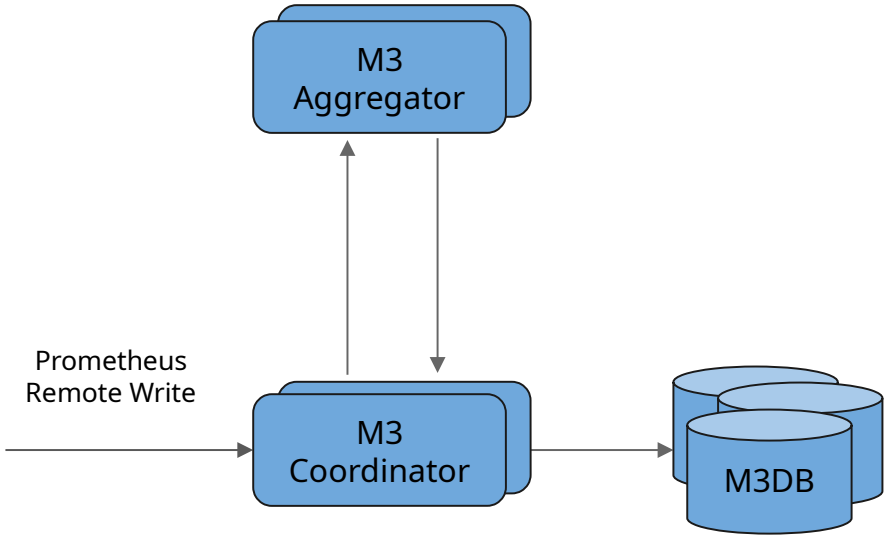
As we ingest more and more new metrics over time, how can we do so without just indefinitely paying more?



chronosphere

Aggregator Rollups

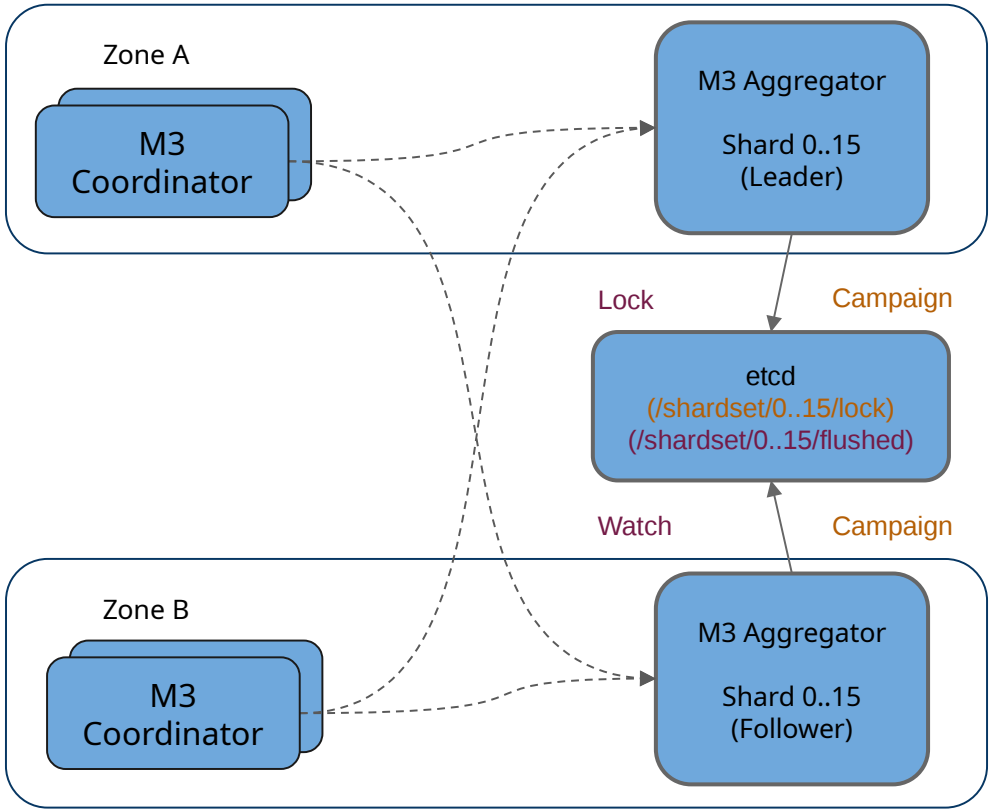
- Optionally keep all or just some metrics downsampled with streaming aggregator.
- Able to aggregate across arbitrary dimensions, e.g. ingestion time rollup rules allow for **sum() by ()** of metrics with cardinality in millions.
- Move expensive recording rules to streaming aggregation, store less and alert on very high cardinality cheaply.



chronosphere

Aggregator Rollups

- Leader election is uses keep single aggregator replica will emitting results to downstream M3DB.
- Flushed state per time window is kept so on leader failover no time windows are missed.



Aggregator Rollups

- HTTP request latency broken down by server version.
- Unique values on original metric multiply to create cardinality of 24 million series.
- Using aggregator rollup we can aggregate at ingestion time this to 240 thousand series. Now can alert and query on each HTTP route only needing to touch a few thousand series (~5 thousand).

Metric dimension	Unique values
Pod (“k8s_pod”)	~100 pods per region
Bucket (“le”)	~30 buckets
Status (“status_code”)	~10 status codes
Route (“route”)	~50 HTTP routes
Git SHA (“git_sha”)	~4 active versions
Region (“region”)	~4 regions

chronosphere

E.g. Rollup across millions of series

Stage 1:
Take delta increase

Stage 2:
Sum by dimension

Stage 3:
Create monotonic cumulative counter

Documentation:
docs.m3db.io/operational_guide/mapping_rollup

```
- name: "http_request latency by route and git_sha without pod"
  filter: "__name__:http_request_bucket k8s_pod:* le:* git_sha:* route:*"
  transforms:
  - transform:
      type: "Increase"
  - rollup:
      metricName: "http_request_bucket"
      groupBy: ["le", "git_sha", "route", "status_code", "region"]
      aggregations: ["Sum"]
  - transform:
      type: "Add"
  storagePolicies:
  - resolution: 30s
    retention: 720h
```

chronosphere

E.g. Rollup across millions of series

Stage 1:
Deltas

Stage 2:
Sum by dimension

Stage 3:
Add cumulative

```
- name: "http_request latency by route and git_sha without pod"
  filter: "__name__:http_request_bucket k8s_pod:* le:* git_sha:* route:*"
  transforms:
    - transform:
        type: "Increase"
    - rollup:
        metricName: "http_request_bucket"
        groupBy: ["le", "git_sha", "route", "status_code", "region"]
        aggregations: ["Sum"]
    - transform:
        type: "Add"
  storagePolicies:
  - resolution: 30s
    retention: 720h
```

chronosphere

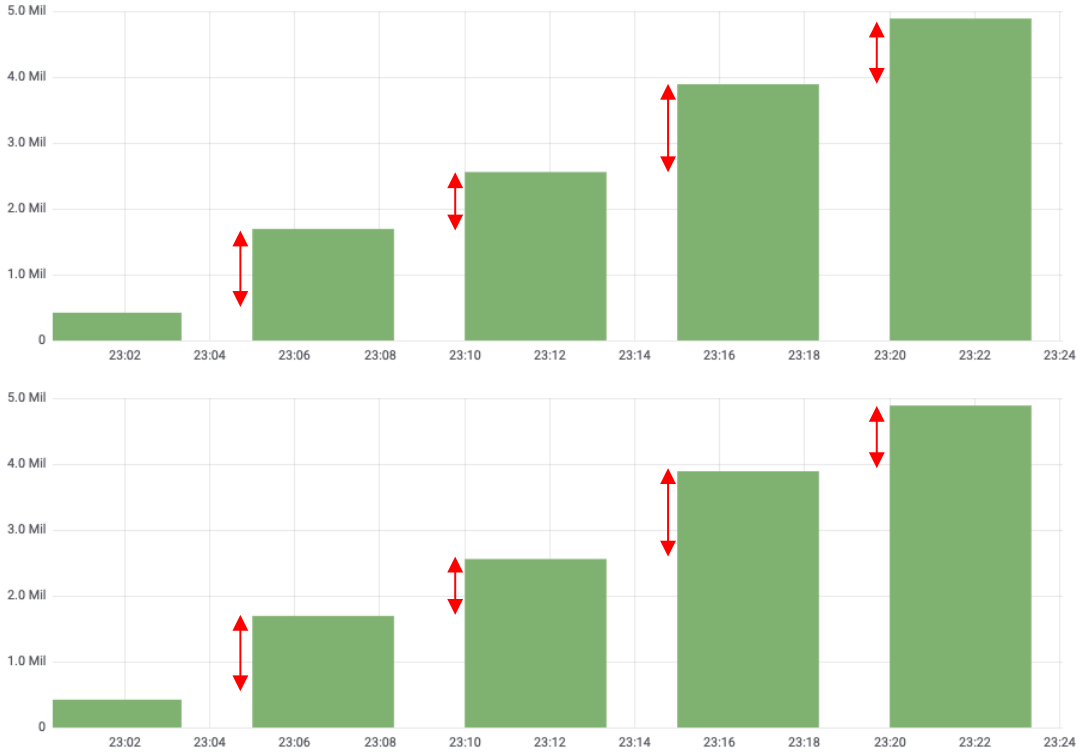
E.g. Rollup across millions of series

Stage 1:
Deltas

nosphere

E.g. Rollup across millions of series

Stage 1:
Deltas



E.g. Rollup across millions of series

Stage 2:
Sum by dimension

Sum each with same "le", "git_sha", "route", "status_code", “region”

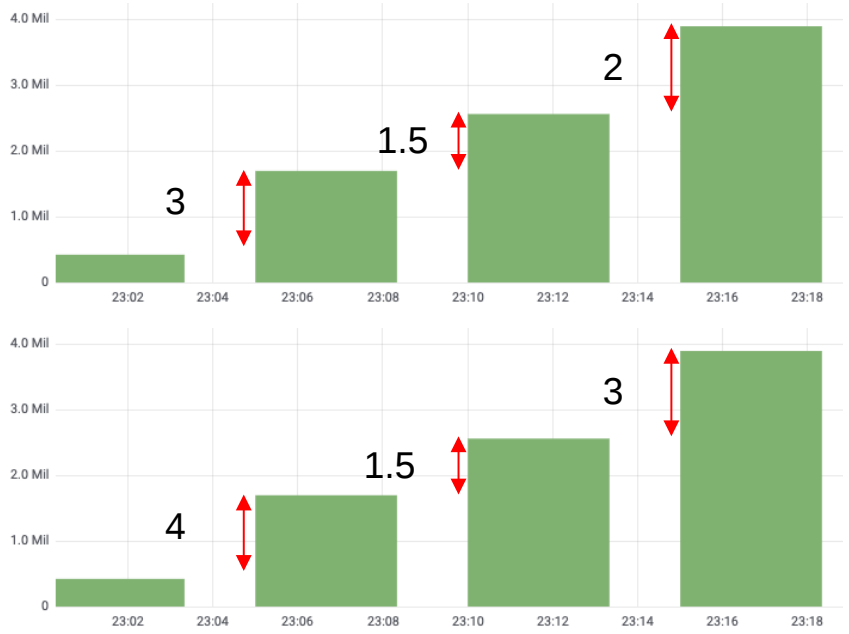
```
- name: "http_request latency by route and git_sha without pod"
  filter: "__name__:http_request_bucket k8s_pod:* le:* git_sha:* route:*"
  transforms:
  - transform:
      type: "Increase"
    - rollup:
      metricName: "http_request_bucket"
      groupBy: ["le", "git_sha", "route", "status_code", "region"]
      aggregations: ["Sum"]
  - transform:
      type: "Add"
  storagePolicies:
  - resolution: 30s
    retention: 720h
```

chronosphere

E.g. Rollup across millions of series

Stage 2:
Sum by dimension

Sum each with
same "le",
"git_sha", "route",
"status_code", “region”



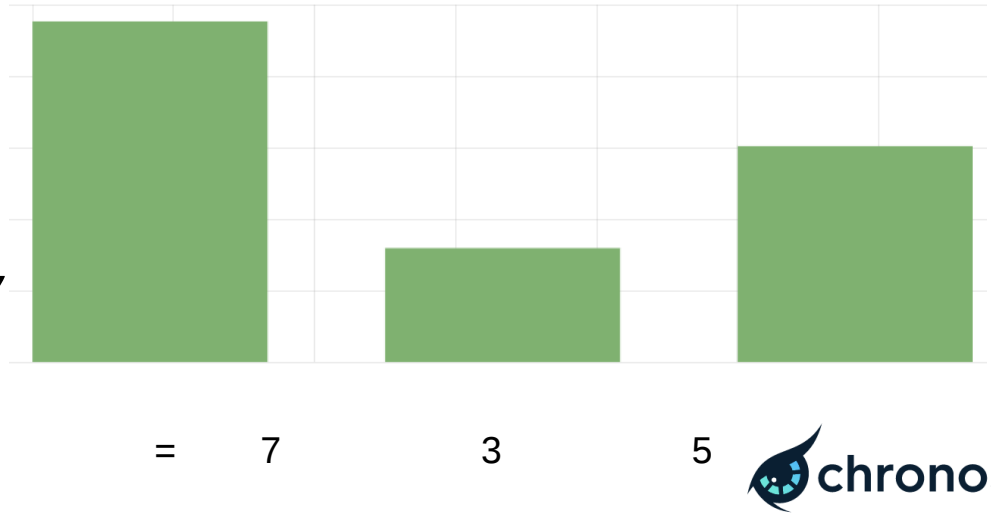
= 7 3 5

chronosphere

E.g. Rollup across millions of series

Stage 2:
Sum by dimension

Sum each with same "le", "git_sha", "route", "status_code", “region”



chronosphere

E.g. Rollup across millions of series

Stage 3:
Add cumulative

Running sum of same "le", "git_sha", "route", "status_code", “region”

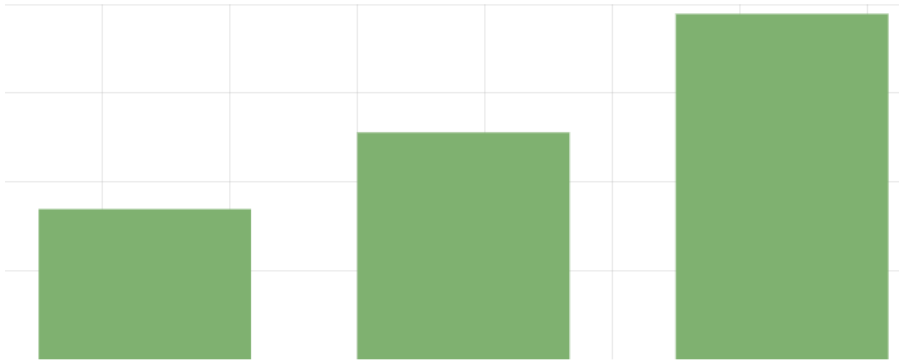
```
- name: "http_request latency by route and git_sha without pod"
  filter: "__name__:http_request_bucket k8s_pod:* le:* git_sha:* route:*"
  transforms:
  - transform:
      type: "Increase"
  - rollup:
      metricName: "http_request_bucket"
      groupBy: ["le", "git_sha", "route", "status_code", "region"]
      aggregations: ["Sum"]
  - transform:
      type: "Add"
  storagePolicies:
  - resolution: 30s
    retention: 720h
```

chronosphere

E.g. Rollup across millions of series

Stage 3:
Add cumulative

Running sum of
same "le",
"git_sha", "route",
"status_code", “region”



= 7 10 15

chronosphere

Optimization Summaries

- Query limits (READ)
  - Provides controls over *global* and *per-query* limits
  - Limits available on *series* and/or *blocks* (i.e. memory read)

- Aggregator rollups (WRITE)
  - For very high cardinality metrics avoid constant heavy reads against TSDB with recording rules.
  - Optionally drop uninteresting labels (faster queries at cheaper cost).
  - Keep ability to apply further transforms and rate interval at query time.

chronosphere

Thank you, questions?

Thank you to M3 contributors:
...@chronosphere.io, ...@uber.com, ...@linkedin.com, ...@aiven.io, ...@cloudera.com and many other great individuals!

Learn more:

- Slack: https://bit.ly/m3slack
- Office Hours (every 3rd Thursday, 11-1pm EST): signup for slot here
- Meetup: https://www.meetup.com/M3-Community/
- GitHub: https://github.com/m3db/m3
- Documentation: https://m3db.io
- Contact us: contact@chronosphere.io

chronosphere