Querying millions to billions of metrics with M3DB's index

FOSDEM 2020

@roskilli

Previously M3 tech lead at Uber, creator of M3DB.

CTO at Chronosphere.

Member of OpenMetrics.

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)

Problem

1. Increasing number of regions, containers, k8s pods, tracking deployed version - (cardinality!)

2. Metrics can have arbitrary number of dimensions

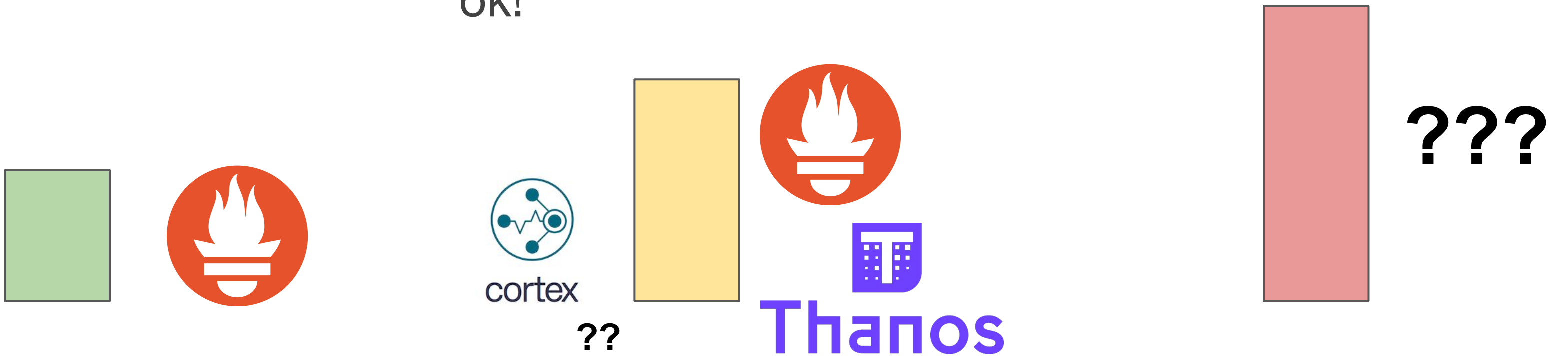
3. Building compound index is expensive

Adding more metrics at organizations

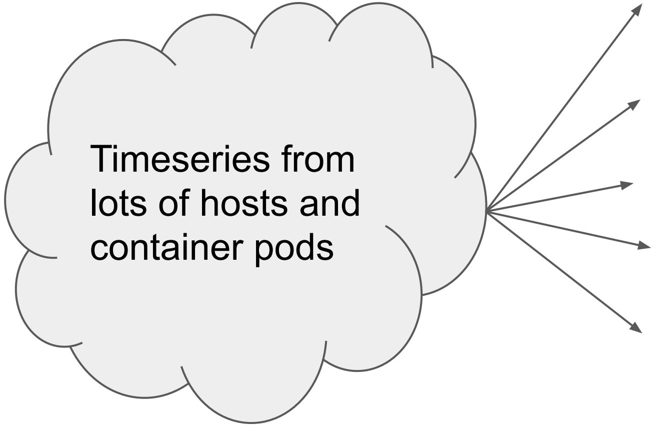
1. We have monitoring, it’s awesome and developers are happy with standardized metrics mostly.

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.

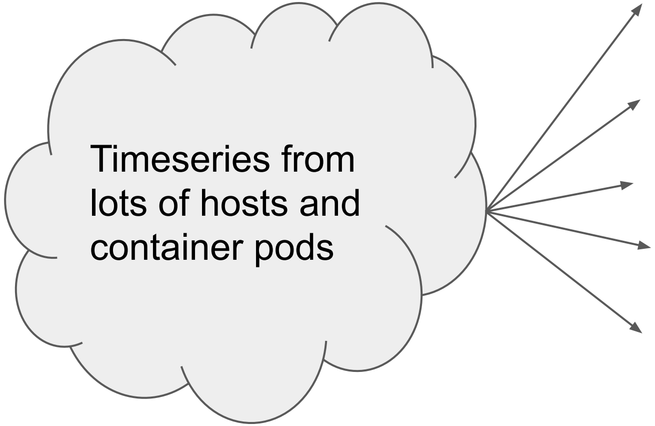


Timeseries



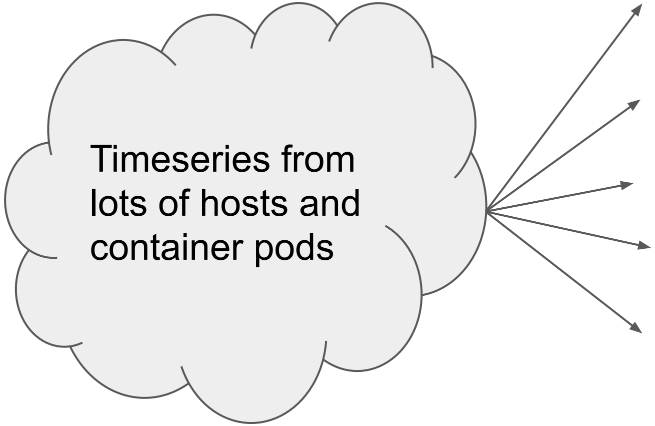
ID	Timeseries
1	__name__=cpu_seconds_total, pod=foo-123abc
8	__name__=memory_memfree, pod=foo-123abc
33	__name__=cpu_seconds_total, pod=foo-456def
44	__name__=memory_memfree, pod=foo-456def
45	__name__=cpu_seconds_total, pod=bar-768ghe
58	__name__=memory_memfree, pod=bar-768ghe
	… millions .. and if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

Aggregate metric cpu_seconds_total



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

cpu_seconds_total and pod=foo-(.+)



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

Need high flexibility and speed

1. Any arbitrary set of dimensions/labels can be specified for filtering

2. Ideally speed is sub-linear

Timeseries column lookup

1. Secondary lookup using prefix ordered table

Labels	Timeseries ID (fingerprint)
__name__=cpu, pod=foo-123abc	1

2. Secondary inverted index

Label	Label value	Timeseries IDs
__name__	cpu	1, 2, 3
pod	foo-123abc	1
	foo-456def	2
	bar-123abc	3

ID	Column key	Col value
1	**__name__=cpu, pod=foo-123abc**	{t=...,v=...} ➡️
2	**__name__=cpu, pod=foo-456def**	{t=...,v=...} ➡️
3	**__name__=cpu, pod=bar-123abc**	{t=...,v=...} ➡️

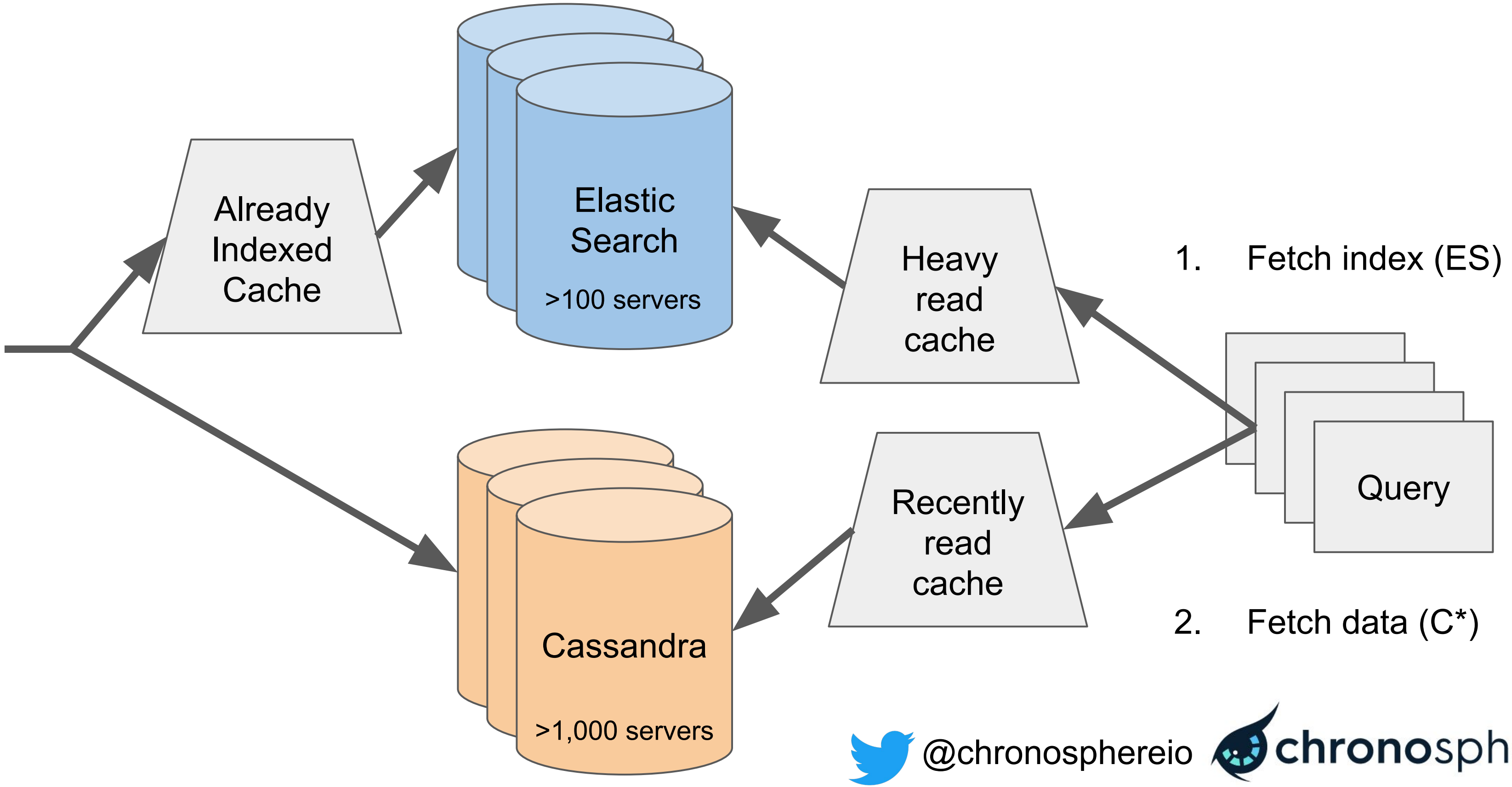
Ways to keep timeseries index/data

1. **Index and data live separately**
   Lookup and returning timeseries data across processes, typically making network request between the two operations.

2. **Index and data live together**
   Lookup next to timeseries data, send data back directly once matches index query.

M3 storage evolution (pre-open release, 2015)

v1

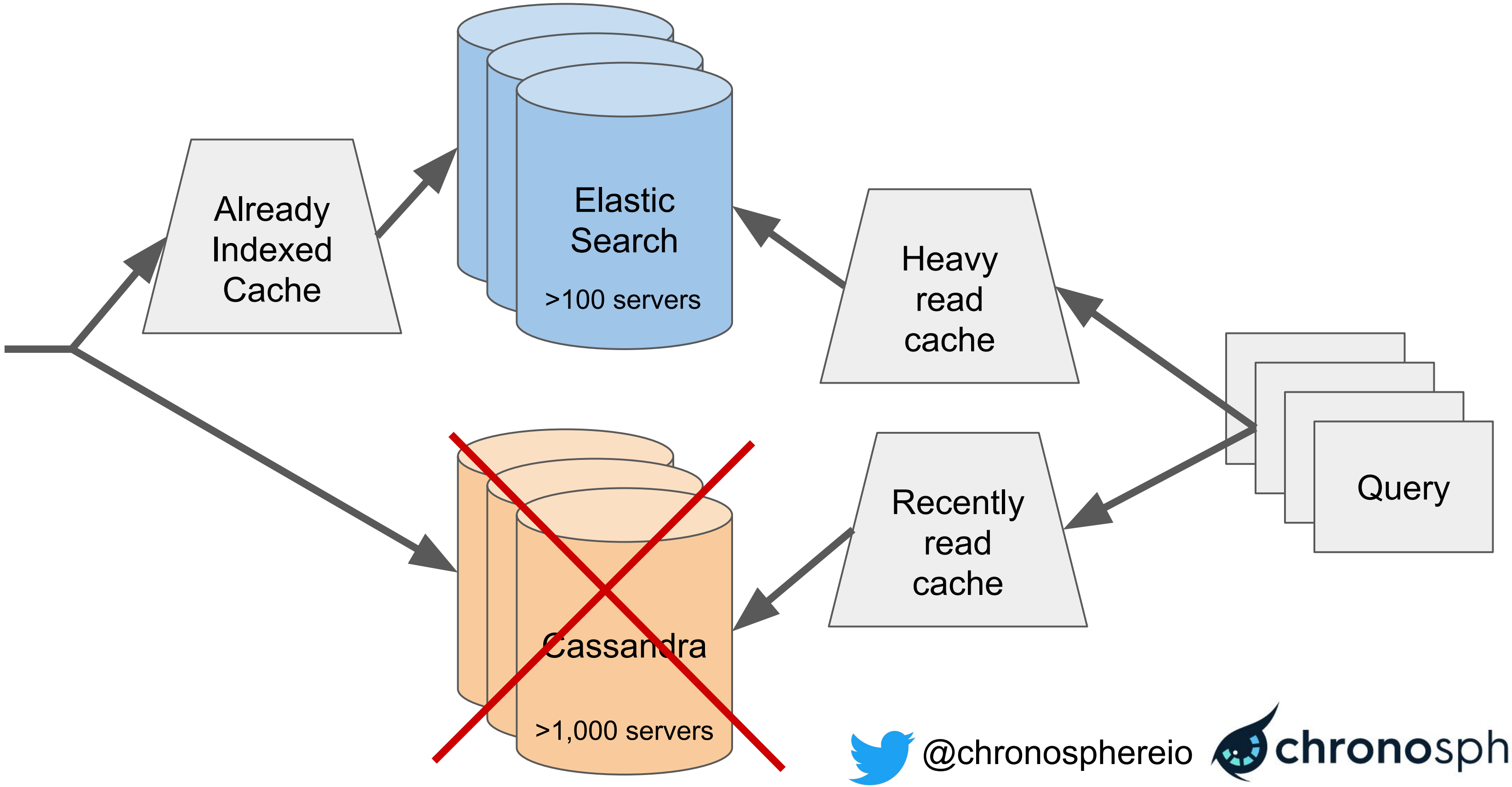


1. Fetch index (ES)
2. Fetch data (C*)

@chronosphereio chronosphere

M3 storage evolution (pre-open release, 2016)

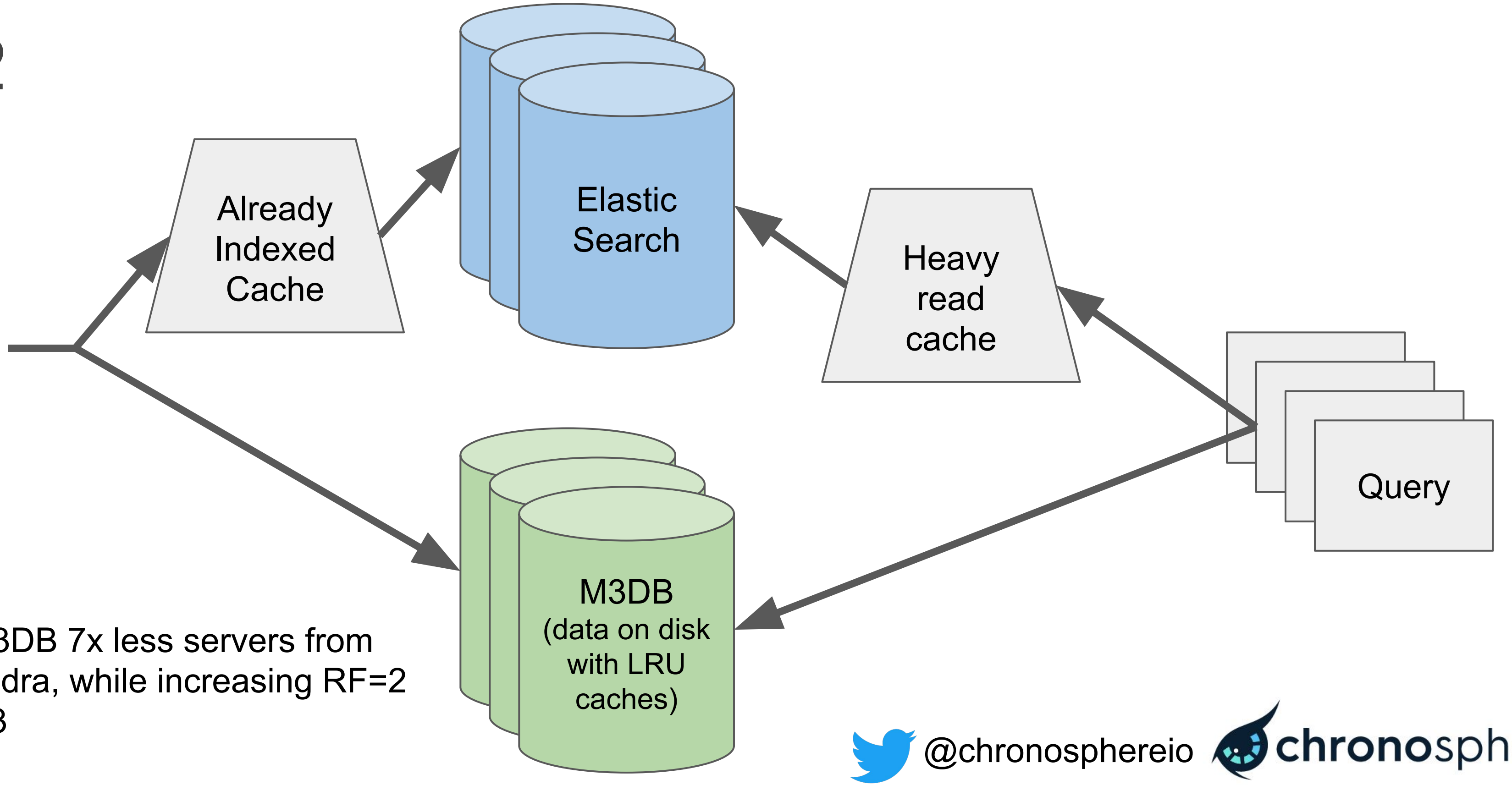
v1



@chronosphereio chronosphere

M3 storage evolution (pre-open release, 2016)

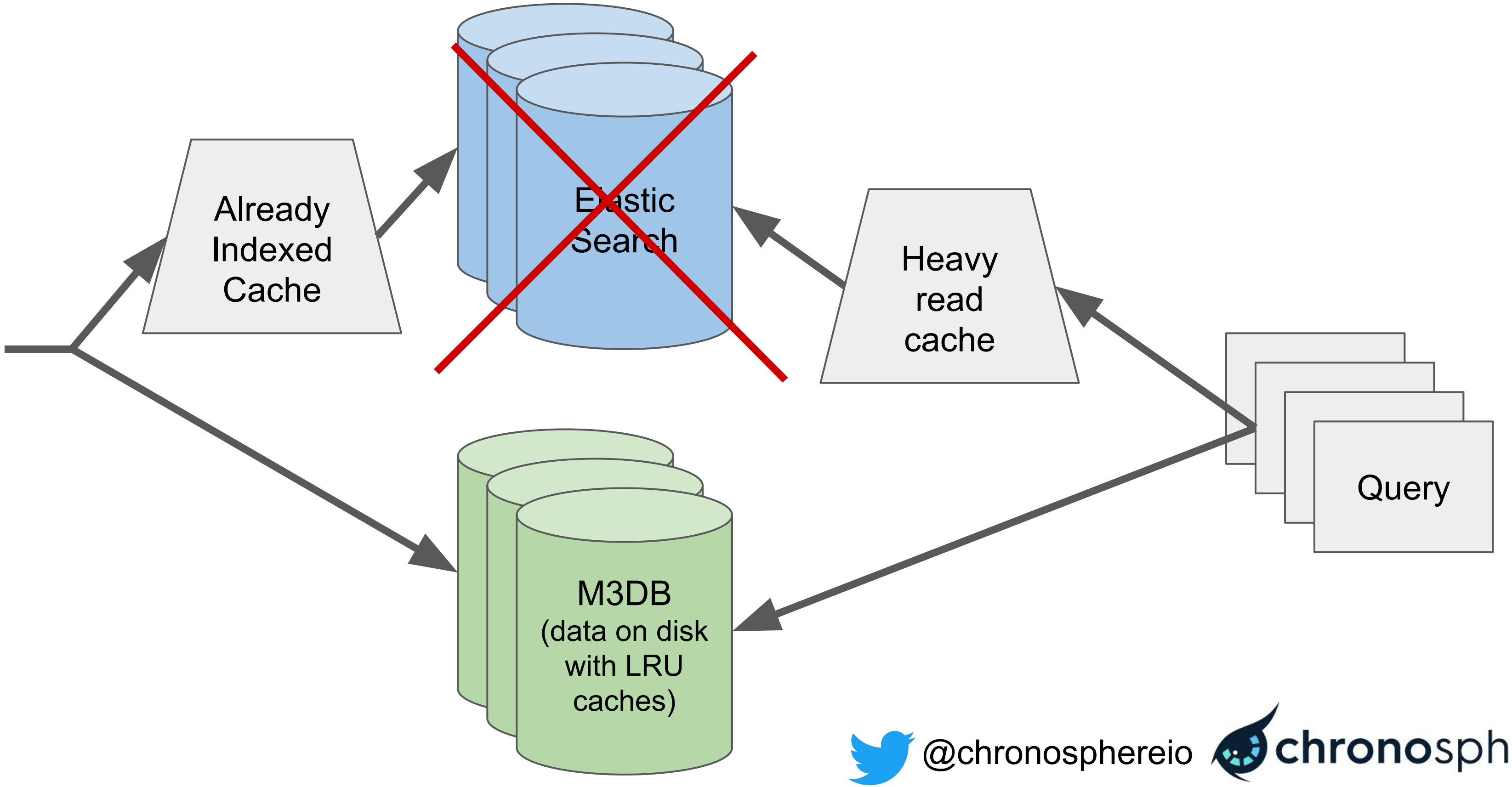
v2



With M3DB 7x less servers from Cassandra, while increasing RF=2 to RF=3

M3 storage evolution (pre-open release, 2018)

v2

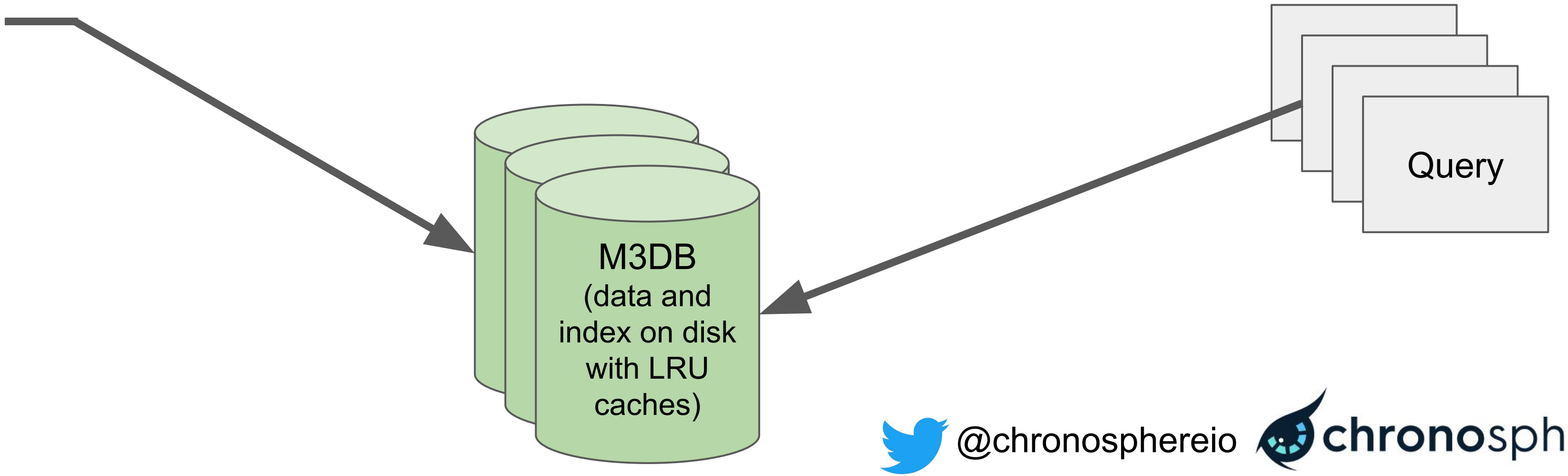


@chronosphereio chronosphere

M3 storage evolution (open release, 2018)

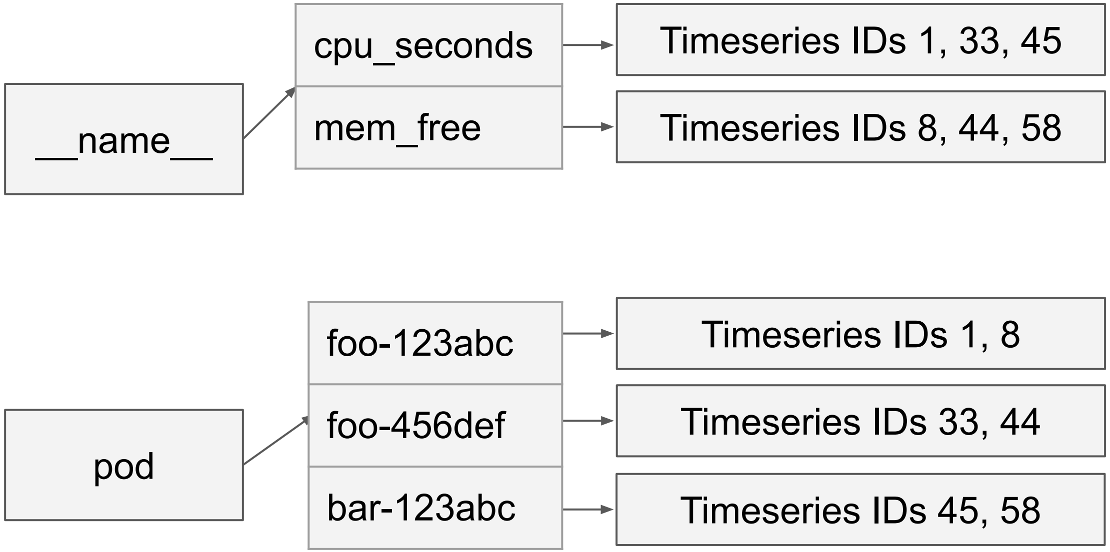
v4

All read/write caches for data/index now in M3DB nodes



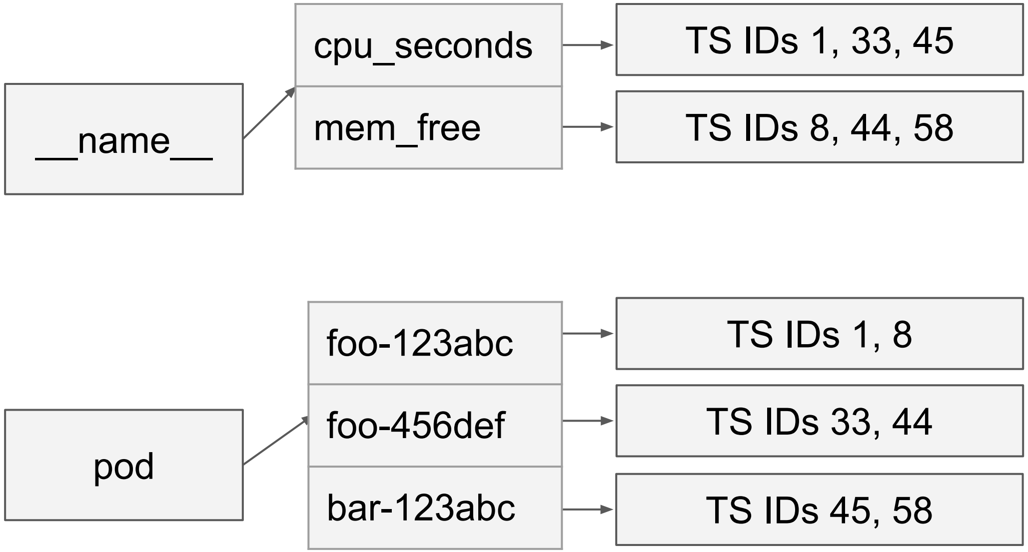
Inverted index w/ Prometheus

https://github.com/prometheus/prometheus/blob/master/tsdb/docs/format/index.md



Inverted index w/ Prometheus

https://github.com/prometheus/prometheus/blob/master/tsdb/docs/format/index.md



ID	Timeseries
1	__name__=cpu_seconds, pod=foo-123abc
8	__name__=mem_free, pod=foo-123abc
33	__name__=cpu_seconds, pod=foo-456abc
44	__name__=mem_free, pod=foo-456abc
45	__name__=cpu_seconds, pod=bar-123abc
58	__name__=mem_free, pod=bar-123abc

Inverted index w/ Prometheus

Labels (name and distinct values entries)

```
┌────────────────┬──────────────────┬───────────────────┐
│ len <4b>       │ #names <4b>      │ #entries <4b>     │
├────────────────┴──────────────────┴───────────────────┤
│ ┌───────────────────────────────────────────────────┐ │
│ │ ref(value_0) <4b>                                 │ │
│ ├───────────────────────────────────────────────────┤ │
│ │ ...                                               │ │
│ ├───────────────────────────────────────────────────┤ │
│ │ ref(value_n) <4b>                                 │ │
│ └───────────────────────────────────────────────────┘ │
```

Inverted index w/ Prometheus

Postings/Timeseries IDs

```
┌──────────────────────┬──────────────────────┐
│ len <4b>             │ #entries <4b>        │
├──────────────────────┴──────────────────────┤
│ ┌─────────────────────────────────────────┐ │
│ │ ref(series_1) <4b>                      │ │
│ ├─────────────────────────────────────────┤ │
│ │ ...                                     │ │
│ ├─────────────────────────────────────────┤ │
│ │ ref(series_n) <4b>                      │ │
│ └─────────────────────────────────────────┘ │
├─────────────────────────────────────────────┤
```

Inverted index w/ Prometheus

Matching label values

https://github.com/prometheus/prometheus/blob/38d32e06862f6b72700f67043ce574508b5697f0/tsdb/querier.go#L417-L451

```go
vals, err := ix.LabelValues(m.Name)
...
var res []string
for _, val := range vals {
    if m.Matches(val) {
        res = append(res, val)
    }
}
...
return ix.Postings(m.Name, res...) // Merges postings/timeseries IDs together
```

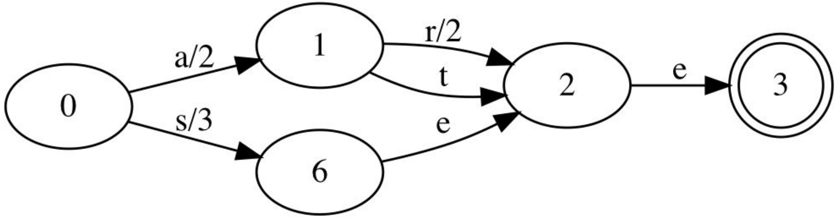
Inverted index w/ M3

1. Inverted index more similar to ElasticSearch & Apache Lucene.

2. Instead of **storing distinct label values** with associated postings, instead stores distinct label values **in FST (Finite State Transducer)**.

3. Instead of **storing postings/timeseries IDs** as integer sets (one after another), instead stores using **Roaring Bitmaps (compressed bitmaps)** for fast intersection (across thousands of sets).

What is an FST?



Like a compressed trie.

Good overview and some examples at https://blog.burntsushi.net/transducers/

Searching data set of wikipedia titles is more than 10x faster than grep.

This matters when you have billions of metrics, i.e. Uber with 11 billion metrics.

Demo

https://github.com/chronosphereiox/high_cardinality_microbenchmark

Disclaimer: This is only testing one part of much bigger systems, mainly to support architectural choices not for real world performance.

Thank you, questions? Come say hi

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

Learn more (release 0.15.0 coming soon):

- Slack https://bit.ly/m3slack
- Mailing list https://groups.google.com/forum/#!forum/m3db
- GitHub https://github.com/m3db/m3
- Documentation https://m3db.io
- Chronosphere contact@chronosphere.io

@chronosphereio chronosphere

chronosphere

Next generation monitoring platform for scale

Chronosphere provides a monitoring platform, built on M3, for today's most demanding environments. The platform stores and analyzes tens of billions of metrics in order to gain higher level insights in real time.

Try the Beta

PLATFORM CURRENTLY IN PRIVATE BETA