Scaling Prometheus

How We Got Some **Thanos** Into **Cortex**

Marco Pracucci, Grafana Labs
Thor Hansen, HashiCorp

Horizontally Scalable Prometheus

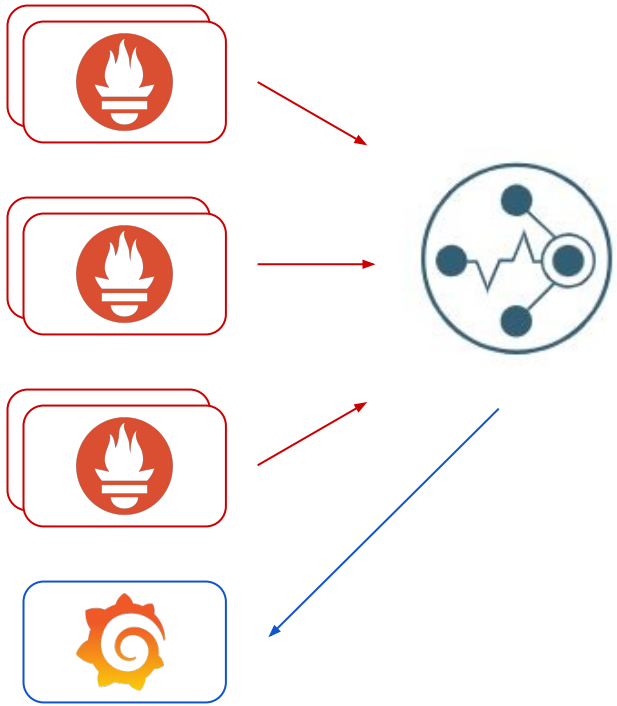
Cortex is a distributed time series store built on Prometheus that is:

- **Horizontally scalable**
- **Highly Available**
- **Durable long-term storage** (1+ year)
- **Multi-tenant**
- **Global view** across all your Prometheus
- **Blazing fast queries**

CNCF Sandbox project, applied to Incubation

Learn more at cortexmetrics.io

Horizontally Scalable Prometheus



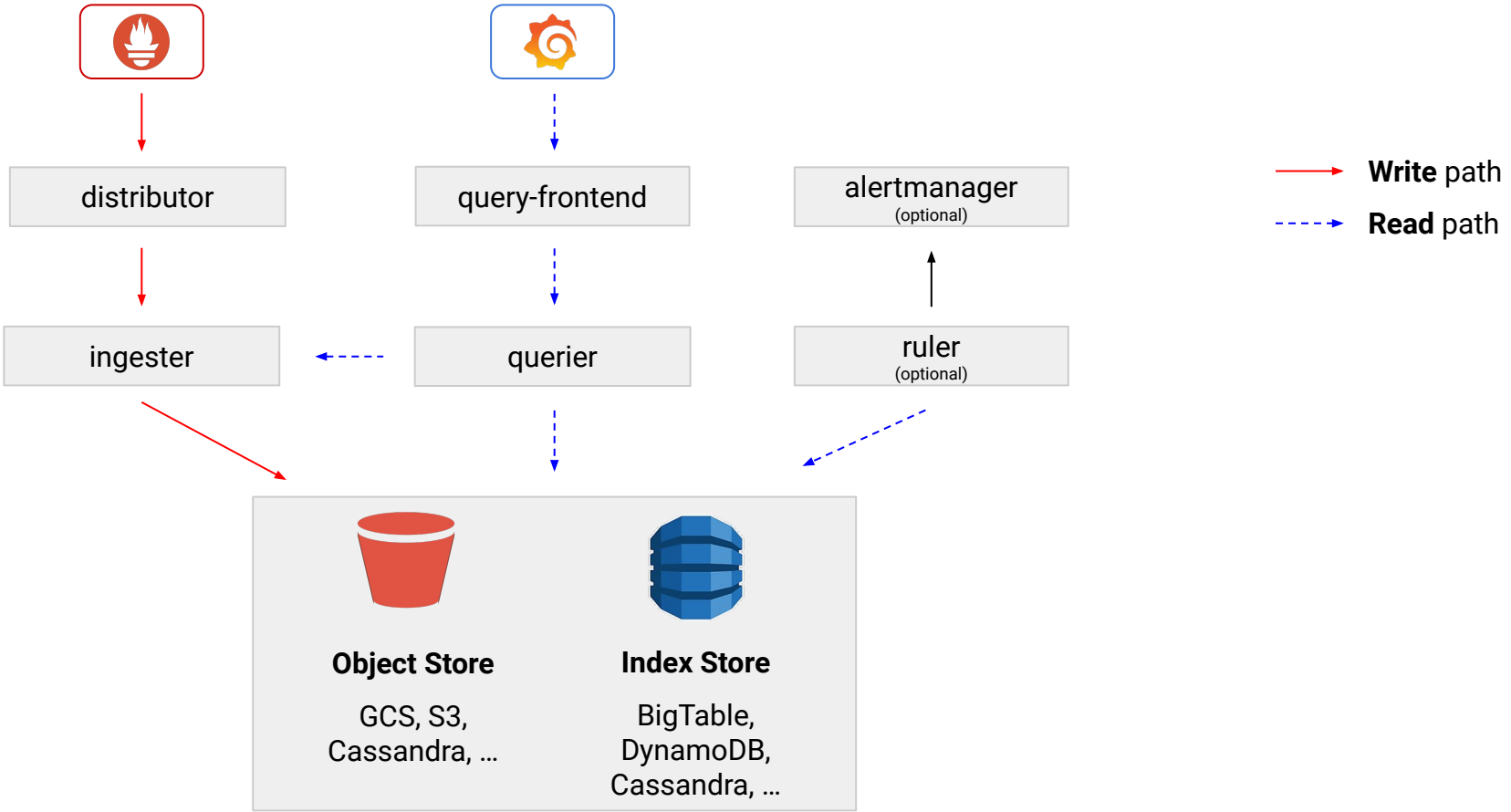
Typical use case:

- Multiple Prometheus servers in HA pairs remote writing to Cortex
- Query back metrics from Cortex

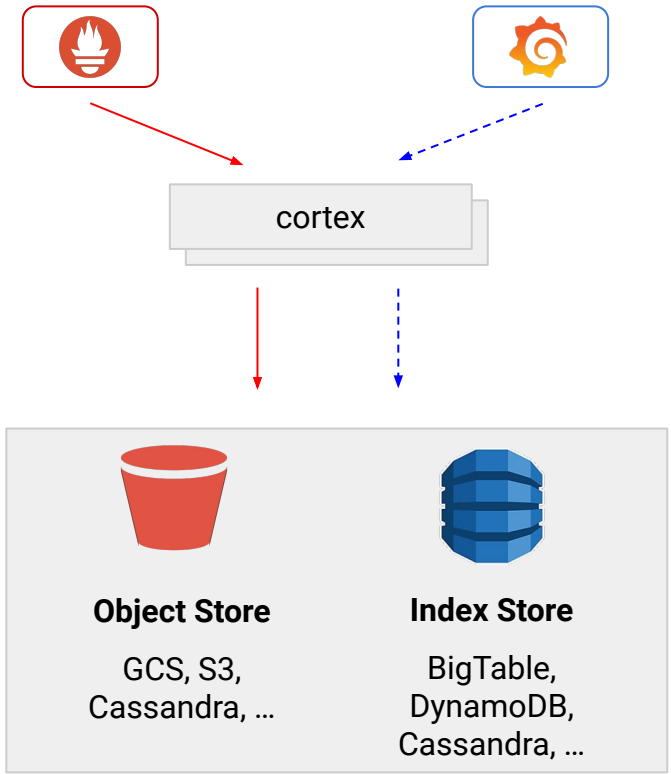
100% compatibility

Cortex exposes Prometheus API endpoints and internally runs PromQL engine

Cortex Microservices Architecture



Cortex Single Binary Mode



The **single binary mode** is the **easiest way** to deploy a Cortex cluster:

- Single binary / config / deployment
- Highly-available
- Horizontally scalable

Internally, a single Cortex process runs all microservices.

No compromises on scalability and performances

This architecture works and scales very well:

- Store 10s to 100s millions active series
- 99.5th percentile query latency < 2.5s

But, requiring both an **object store** and **index store** introduces extra operational **complexity and costs**

The rise of the blocks storage

> “ Can we **remove the index store** at all? ”

Hello!

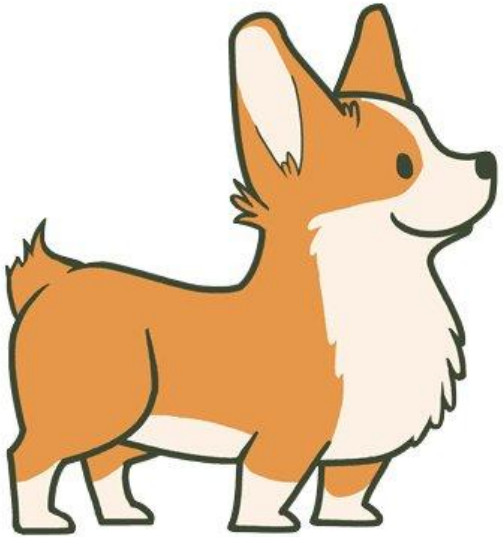
Marco Pracucci

Senior Software Engineer @ Grafana Labs

Cortex and **Thanos** maintainer

@pracucci

Hello!



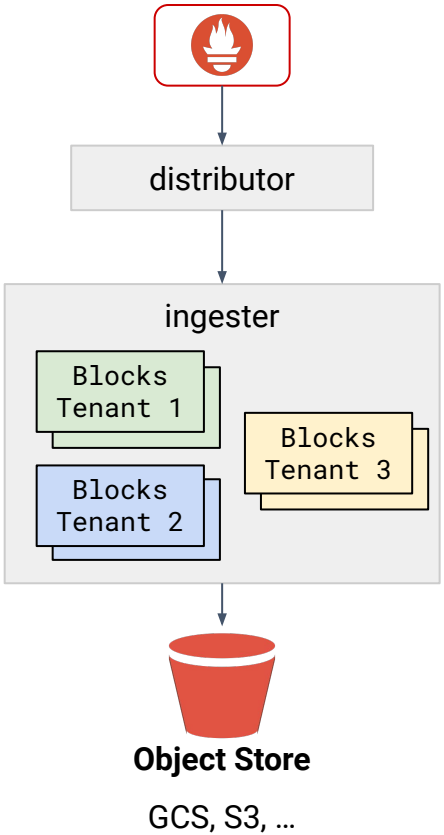
Thor Hansen

Senior Software Engineer @ Hashicorp

Cortex contributor

@thor4hansen

Cortex Blocks Storage



Block Content

- **meta.json**
- **chunks**
  512MB files containing compressed samples
- **index**

The idea:

- Store ingested samples in **per-tenant TSDB blocks**
- A block contains 2 hours of time series
- Blocks are then uploaded to the object store and queried back

Learn more: "TSDB Format"

Hold on.

Isn’t what  **Thanos** is already doing?

Instead of building it from scratch,

why not **collaborate** with Thanos?

Learn more: PromCon 2020 talk “Sharing is Caring”

Cortex Blocks Storage

Feat/blocks #1695

Merged **gouthamve** merged 25 commits into `cortexproject:master` from `thorfour:feat/blocks` on 30 Oct 2019

Conversation 83 | Commits 25 | Checks 1 | Files changed 227

thorfour commented on 24 Sep 2019 — Contributor

The motivation behind this PR was to be able to store larger sized objects into the S3 storage backend. It leverages a lot of code from the thanos project to accomplish that.

This implements ingester storage and s3 storage using the tsdb blocks and thanos shipping to s3.
Each user has a tsdb opened up under their userID, and a shipper that periodically scans that directory and uploads to s3.

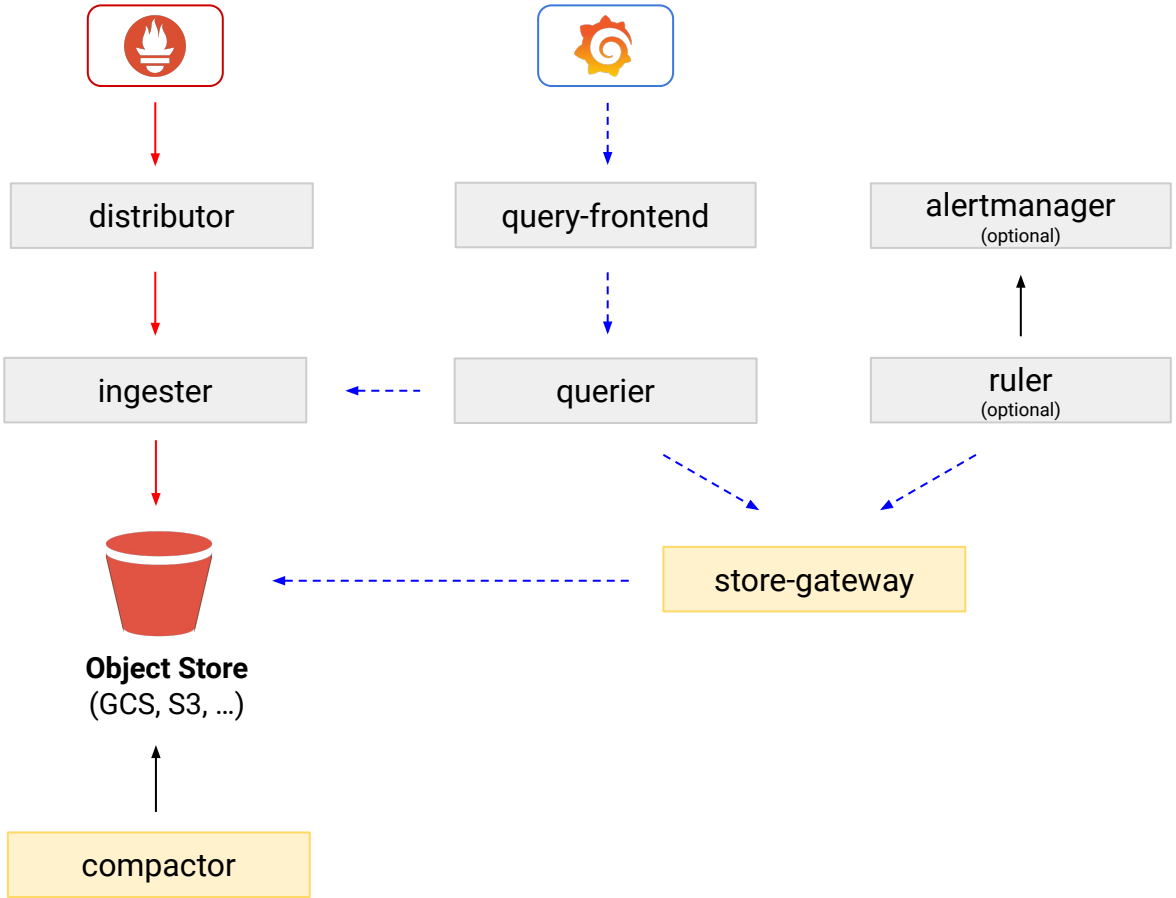
Querier creates a block querier that syncs against s3 to perform the long-retention queries.

1

With an huge help from our friends

and 9 months …

Cortex Blocks Storage Architecture

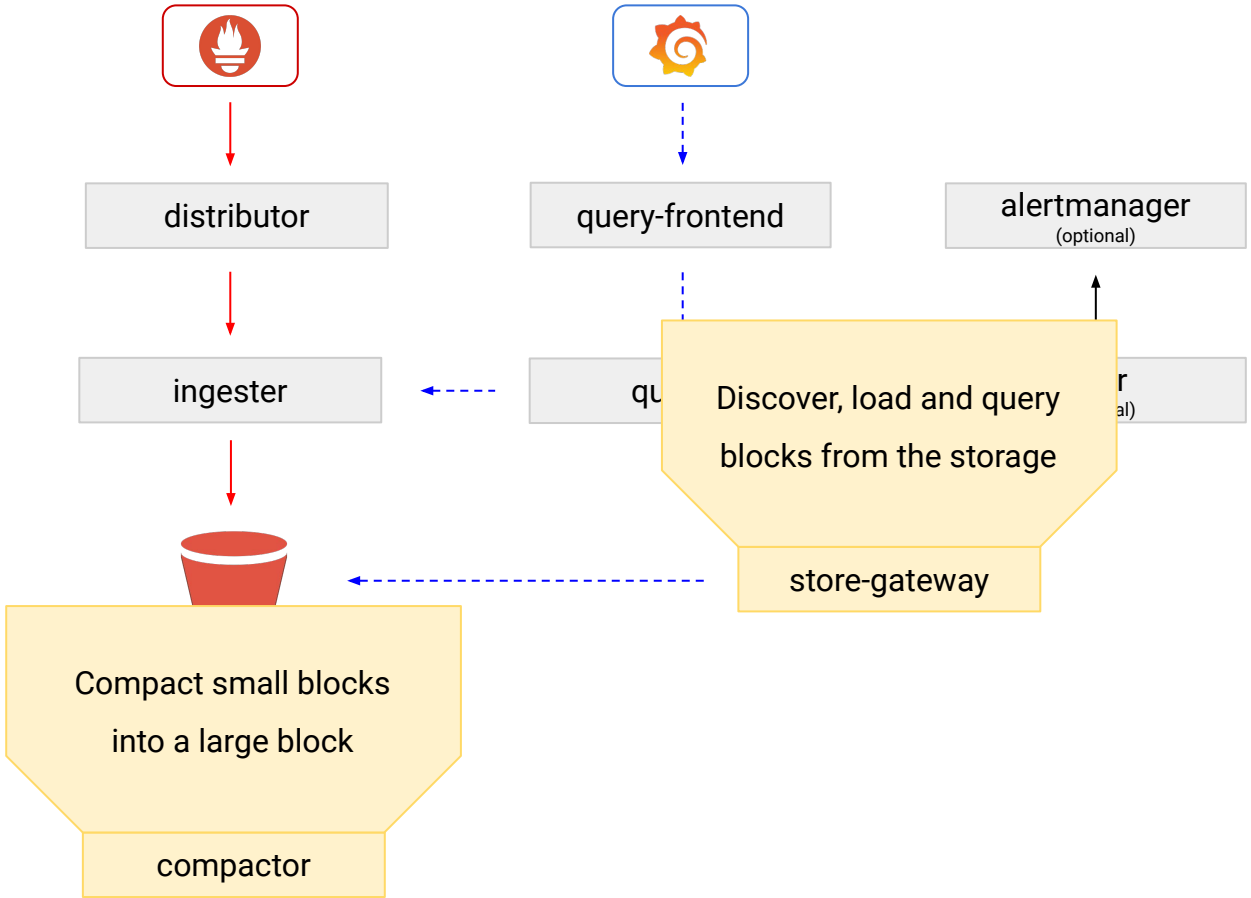


Write path

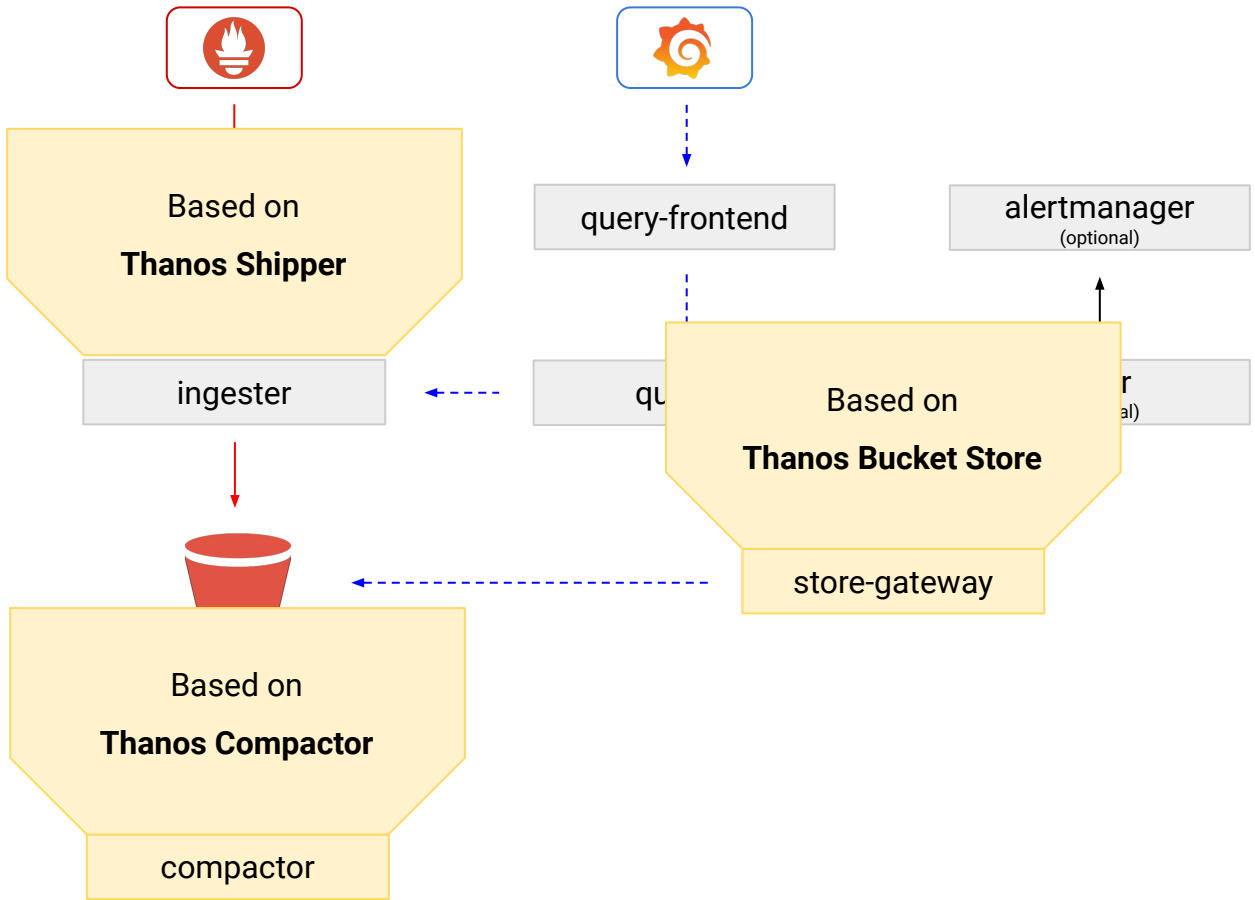
Read path

You can easily deploy Cortex in **single binary mode**

Cortex Blocks Storage Architecture

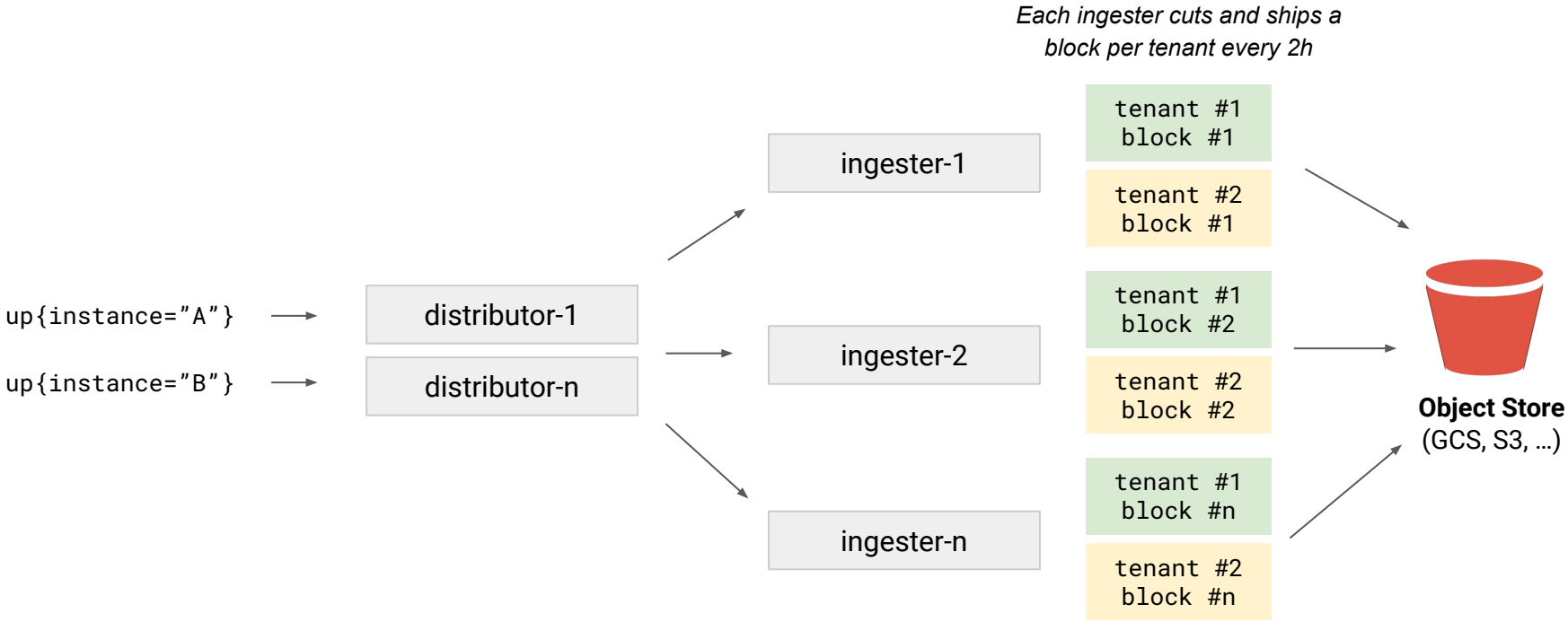


Cortex Blocks Storage Architecture



The write path

1 TSDB block per tenant per ingester every 2h.



The write path: scalability issues

Issue: the number of blocks shipped by ingesters per day is:

```
Daily blocks = num tenants * num ingesters * (24 / 2)
```

Example: 1K tenants, 50 ingesters

```
Daily blocks = 1K tenants * 50 ingesters * (24 / 2) = 600K blocks / day
```

```
1 year retention = > 200M blocks
```

The write path: scalability issues

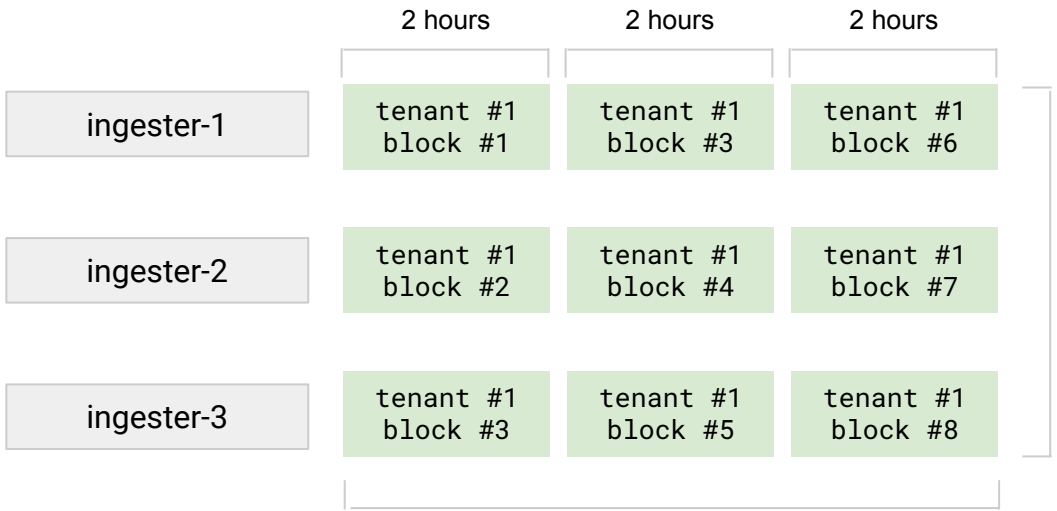
Issue: given Cortex replicates series 3x across ingesters,

samples are duplicated 3x in the storage.

ingester-1	tenant #1 block #1	`up{instance="A"} [T1,V1] [T2,V2]`
ingester-2	tenant #1 block #2	`up{instance="A"} [T1,V1] [T2,V2]`
ingester-3	tenant #1 block #3	`up{instance="A"} [T1,V1] [T2,V2]`

The write path: scalability issues

Solution: merge and deduplicate blocks with the **compactor**.



Horizontal compaction

compacts adjacent blocks
reducing the total number of blocks and total index size

Vertical compaction

merges and **deduplicates** overlapping blocks
reducing total chunks size by 3x

The write path: scalability issues

Solution: merge and deduplicate blocks with the **compactor**.

After compaction:

1 deduplicated block / day / tenant

Compactor can be **horizontally scaled**

The write path: scalability issues

Issue:

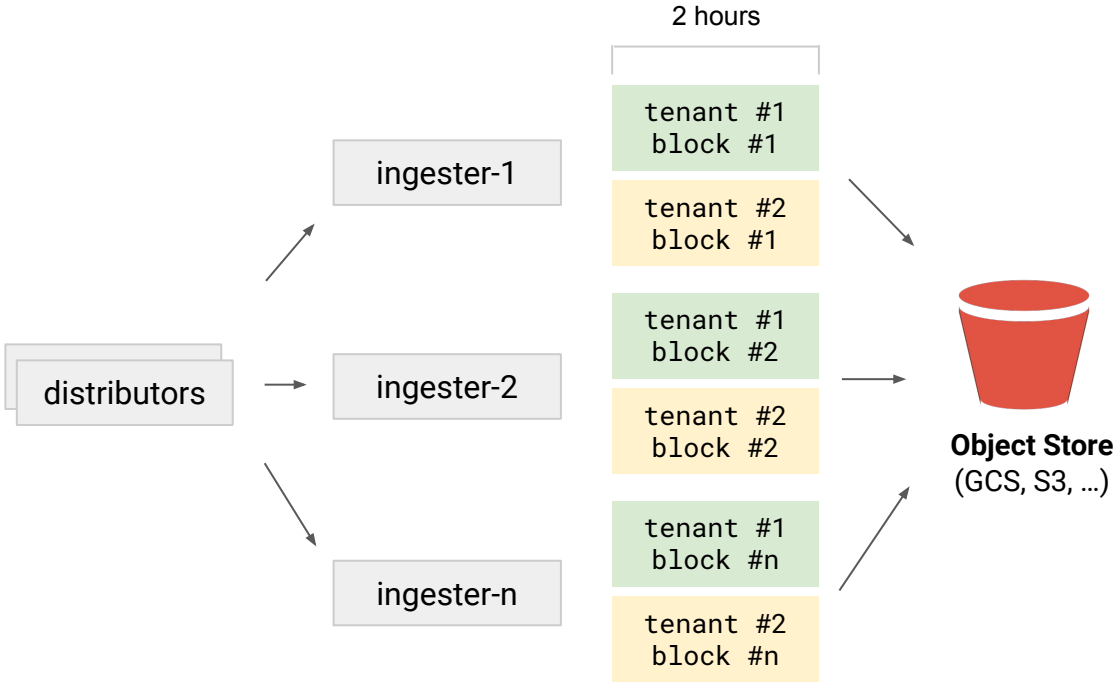
if each tenant series are sharded

across all ingesters,

we still have

1 block / ingester / tenant

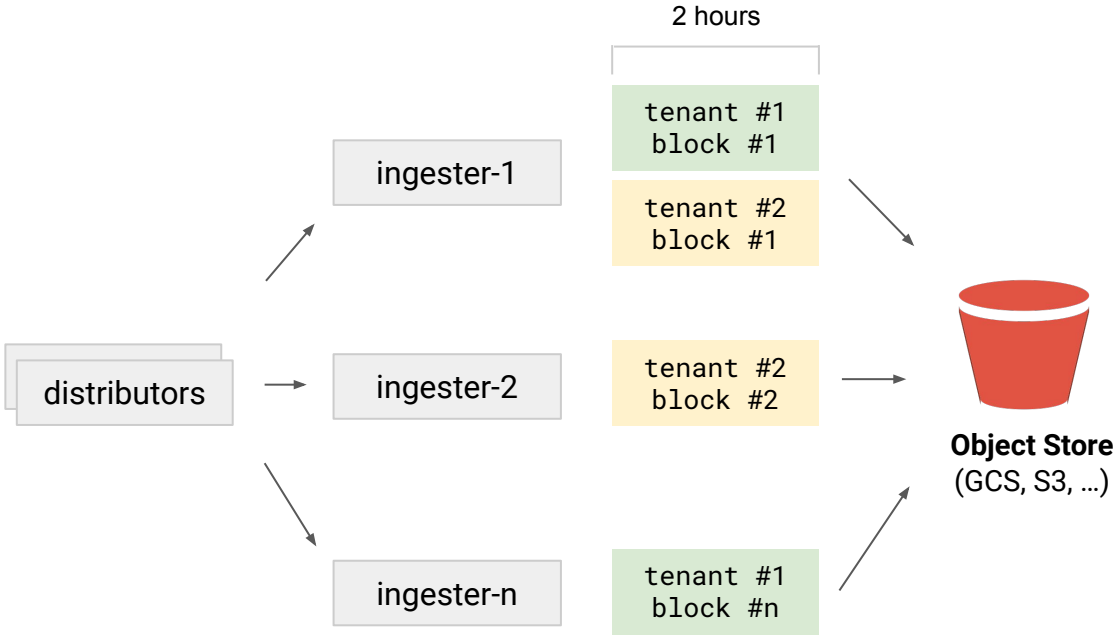
shipped to the storage every 2h



The write path: scalability issues

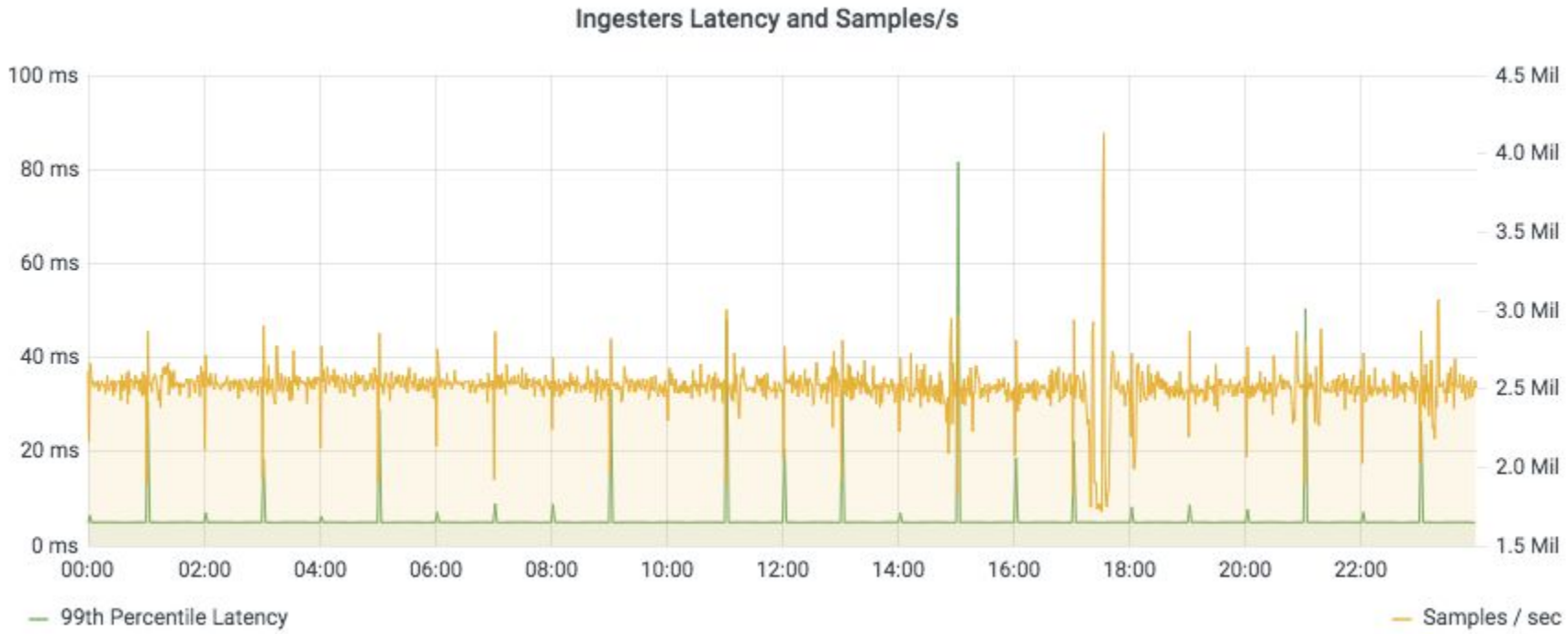
Solution: **shuffle sharding**!

Shard series of each tenant across a different **subset** of ingesters



Shuffle sharding support is experimental and under development

The write path: performances



2.5M samples / sec ingested with a 99th latency ~5ms

The read path

Ok. How do we query it back?

The read path

30M active series

200GB / day

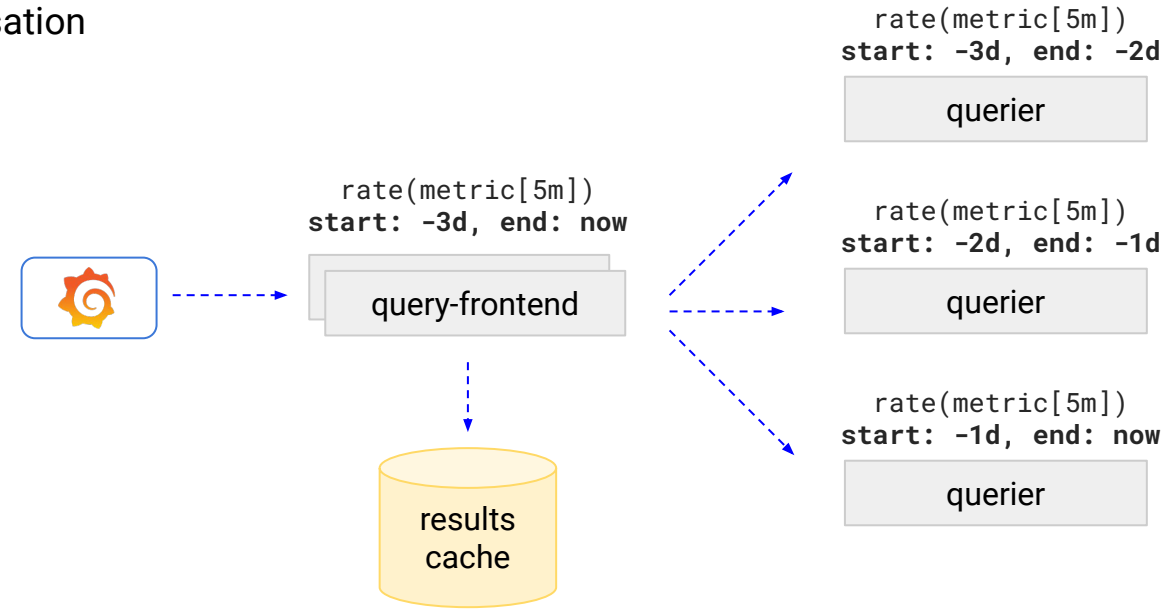
(after compaction)

70TB / year

The read path: query-frontend

The **query-frontend** provides:

- Query execution parallelisation
- Results caching



Learn more: "How to Get Blazin' Fast PromQL"

The read path: query-frontend

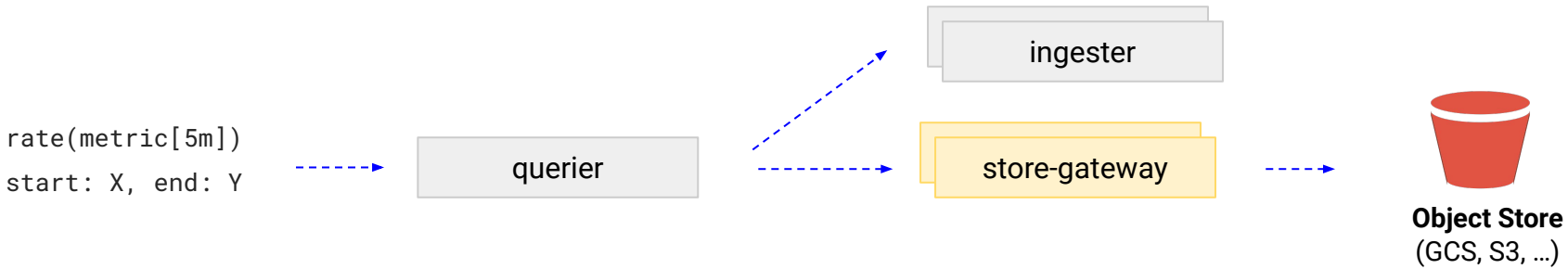
Internally, **a single query** executed by the **querier** will cover only **1 day** (in most cases).

For this reason, we **compact** blocks (by default) up to 1 day period. Results in a better parallelisation of a large time range query.

The read path: querier and store-gateway

The **querier**:

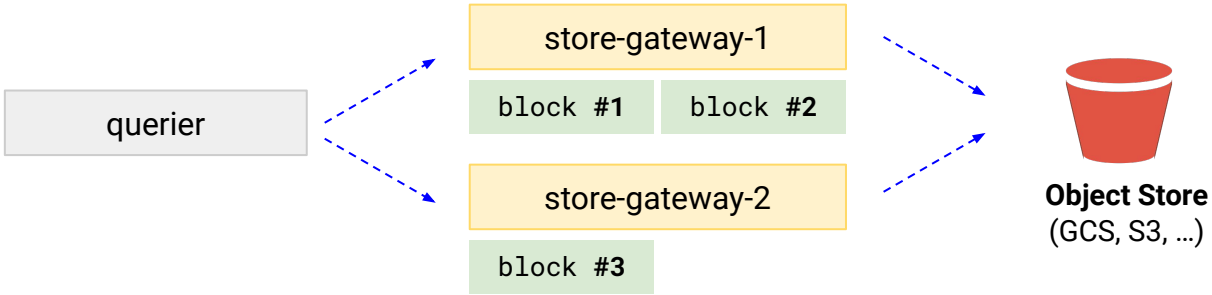
1. **Keeps** a in-memory map of all **known blocks** in the storage (ID, min/max timestamp)
2. **Finds** all block IDs containing samples within the query start/end time
3. **Query** most recent samples from **ingesters** and blocks via the **store-gateways**



The read path: querier and store-gateway

The **store-gateway**:

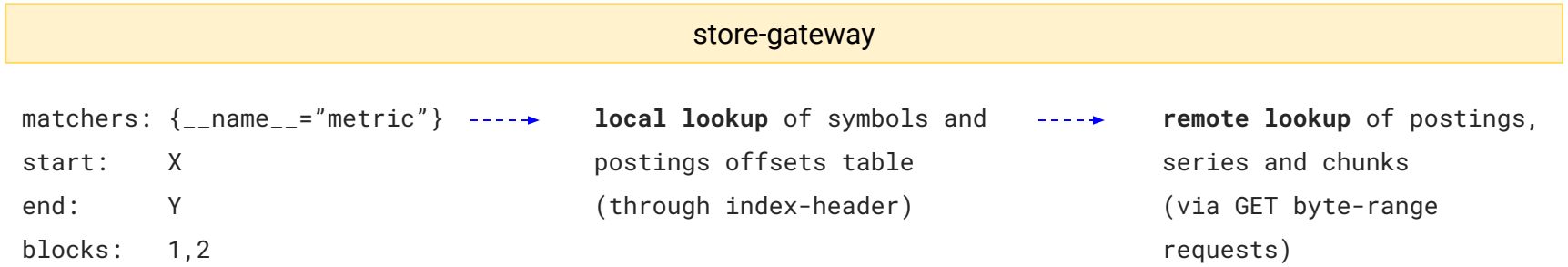
- **Blocks** are **sharded** and **replicated** across store-gateways
- For each block belonging to the shard, a store-gateway loads the **index-header** (small index subset)
- **Querier** queries blocks through the minimum set of store-gateways holding required blocks



The read path: querier and store-gateway

Inside the **store-gateway**:

- Index-header is fully downloaded
- Full index or chunks are never entirely downloaded (but lazily fetched via GET byte-range requests)



The read path: querier and store-gateway

Three layers of caching:

- **Metadata**

  Used to discover blocks in the storage

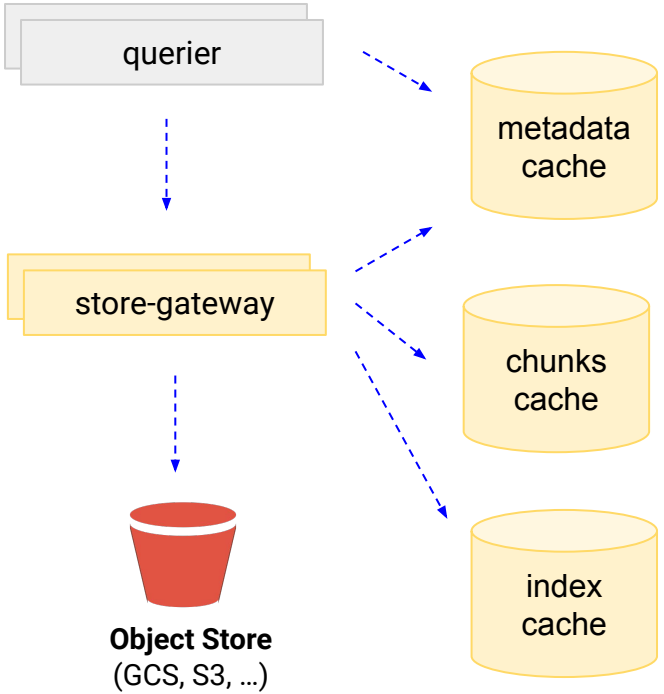
- **Index**

  Lookup postings and series

- **Chunks**

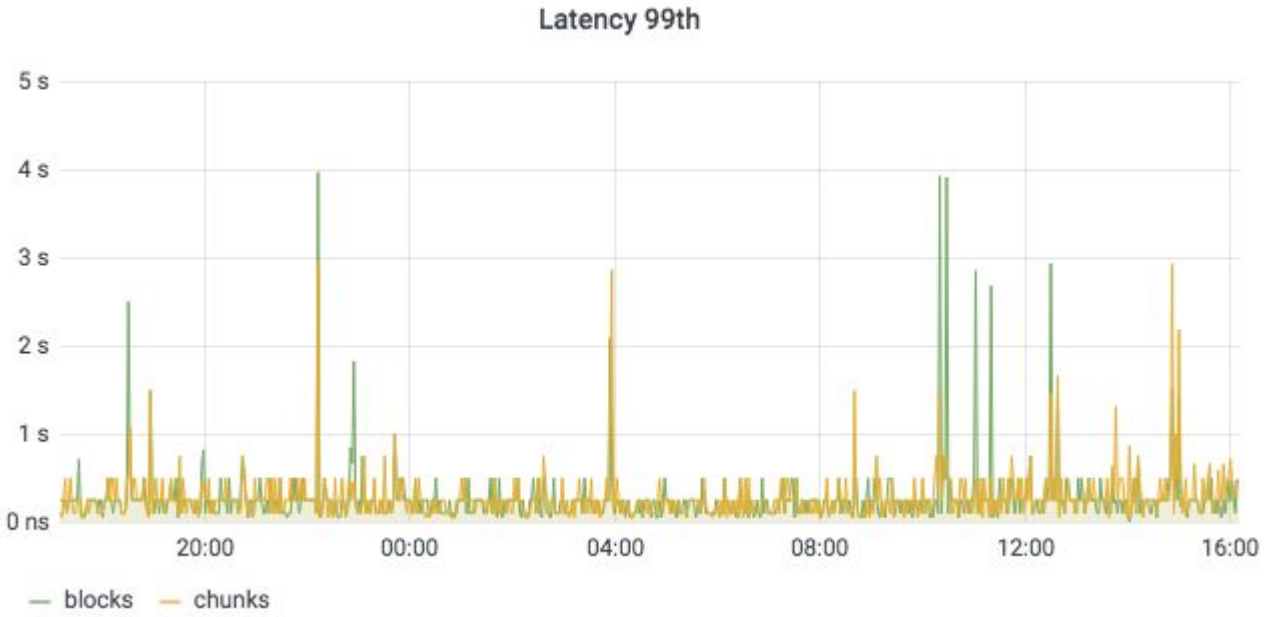
  Fetch chunks containing samples

  (16KB aligned sub-object caching)



Caching is optional, but recommended in production

The read path: performances



Two Cortex clusters ingesting same series (10M active series)

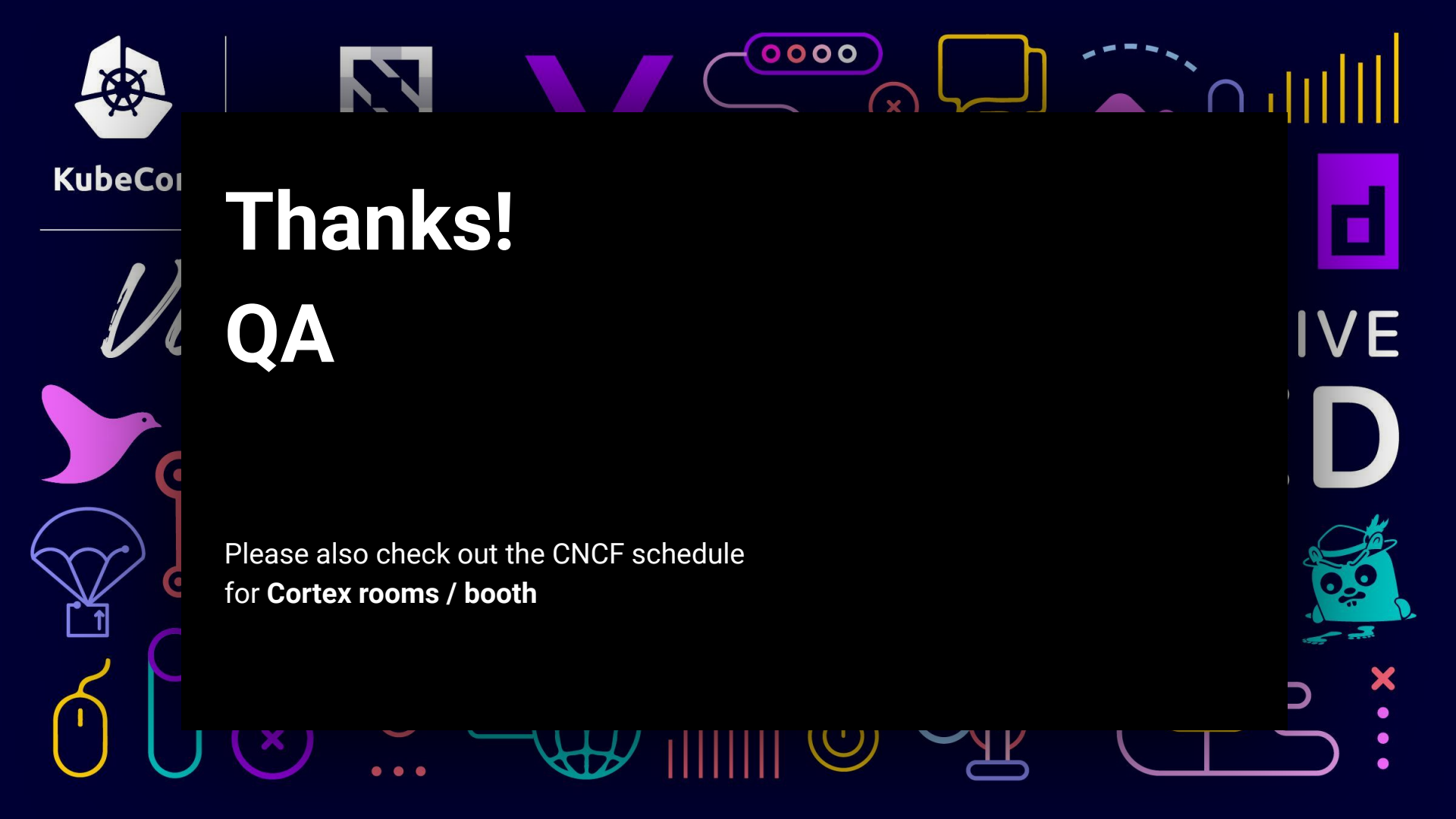
Queries mirrored to both clusters via query-tee

Blocks storage performances comparable with chunks storage

The future

Coming soon in the Cortex blocks storage!

- Even **faster queries**
  - Fully load indexes for last few days?
  - Write-through cache?
  - 2nd dimension to shard blocks?
- Productionise **shuffle sharding**
- **Deletions** (lead by Thanos community)



Thanks!

QA

Please also check out the CNCF schedule
for **Cortex rooms / booth**