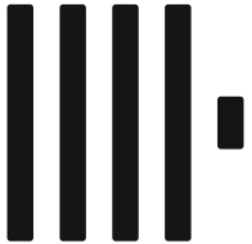
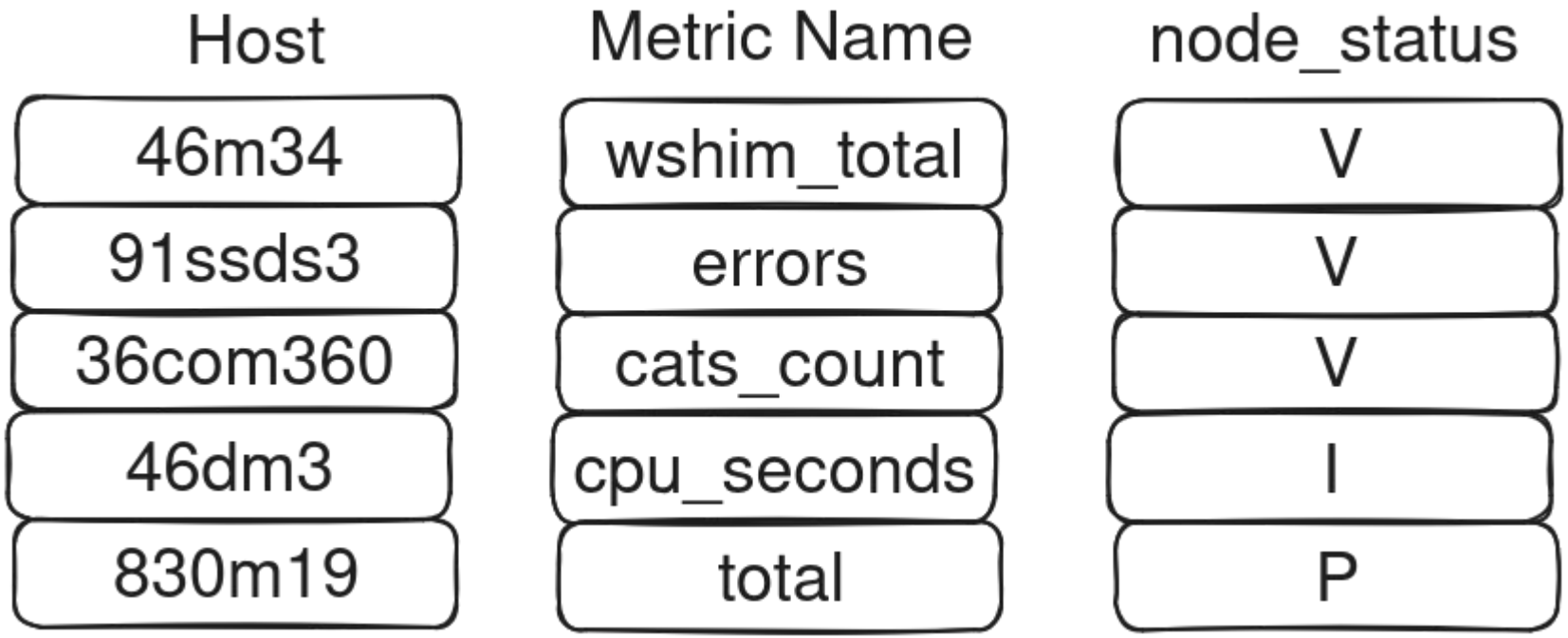
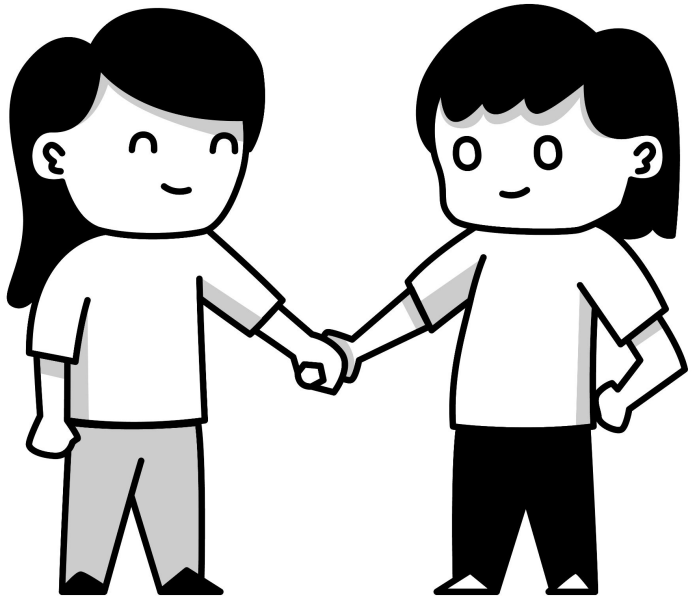
# Experiments in Backing Prometheus with Clickhouse

**Colin Douch, Observability @ Cloudflare**



ClickHouse


Host
46m34
91ssds3
36com360
46dm3
830m19
Metric Name
wshim_total
errors
cats_count
cpu_seconds
total
node_status
V
V
V
I
P



# What if we could have ***both***

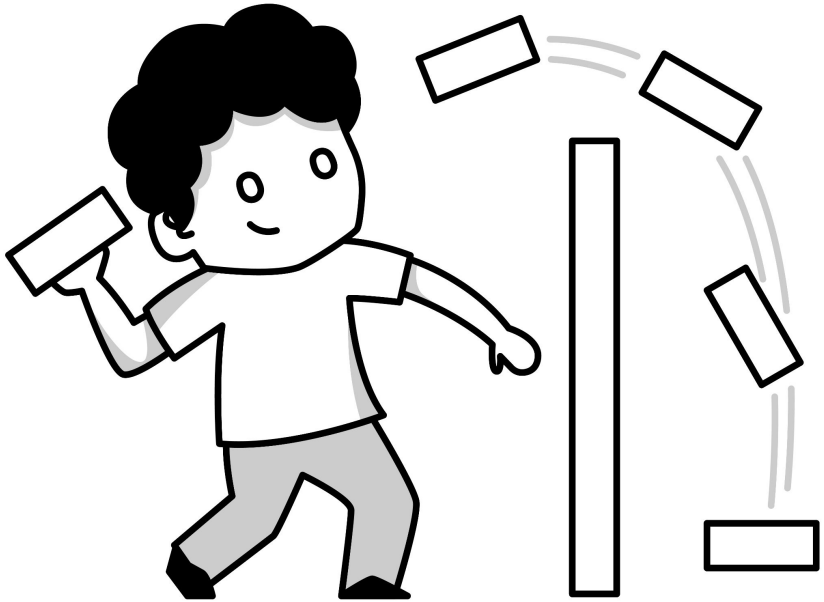
# A Quick Introduction

Hi! I'm Colin

Observability Lead @ Cloudflare

Has bad opinions

## But how?



**Remote write**

Is where the magic happens

CLOUDFLARE

# It turns out that implementing remote write it quite trivial

https://github.com/sinkingpoint/prometheus-clickhouse-bridge

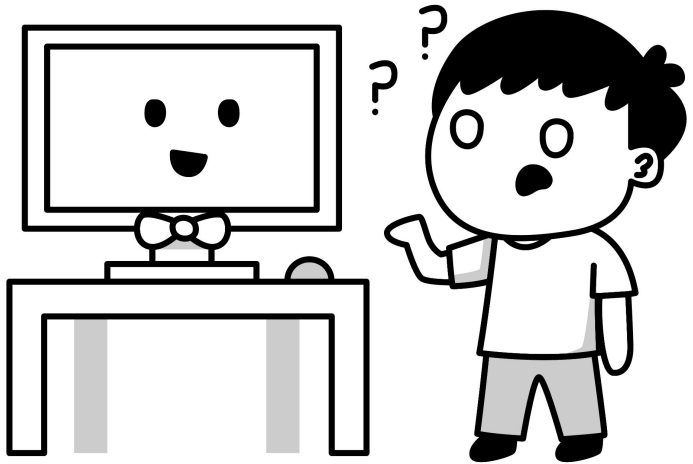
CLOUDFLARE

For queries, we need to implement ***remote read***

https://github.com/sinkingpoint/prometheus-clickhouse-bridge

CLOUDFLARE

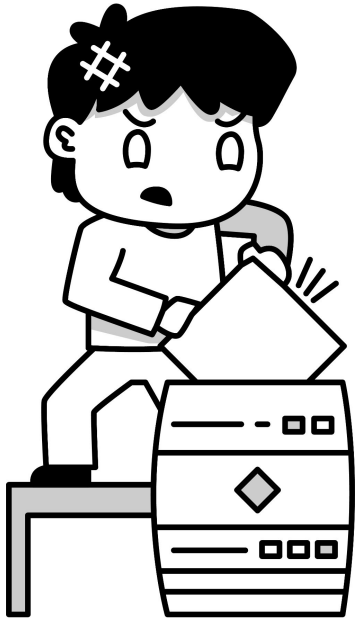
# Welcome to Experiments in Backing Prometheus with Clickhouse



# Testing Environment

**Let’s be good scientists**

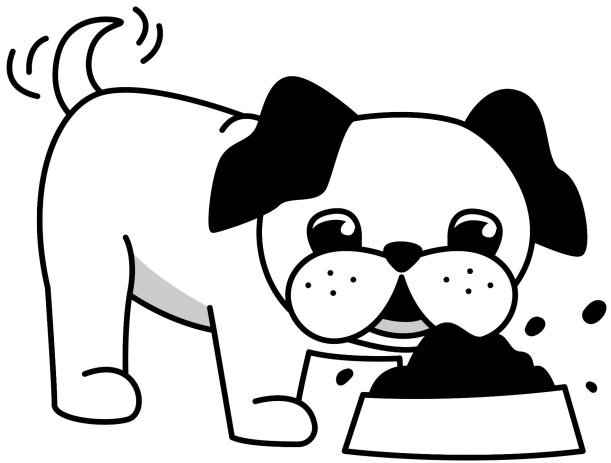
## Hardware



- 92 Cores x 1.9GHz
- 2.4TB NVME Drive
- 256 GB of RAM

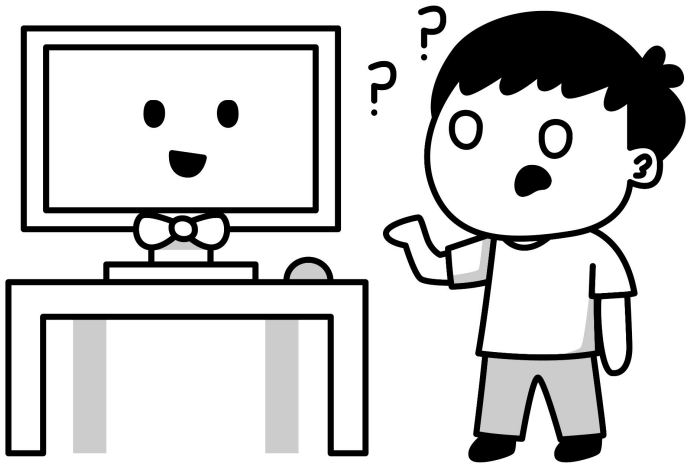
## Timeseries Data

24 Hours of production metrics, across 672 individual services with a wide variety of usage patterns



But... These are already **completed blocks**...

https://github.com/sinkingpoint/replay



# 7 days worth of production queries

## Query Buckets

- Small - 1134 Queries
- Medium - 2475 Queries
- Large - 475 Queries

CLOUDFLARE

## Simple Queries:

```
rate(errors[2m])
```

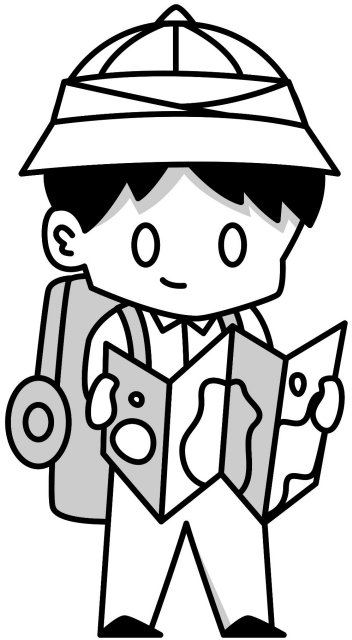
## Medium Queries:

```
rate(errors[2m]) /
rate(total[2m])
```

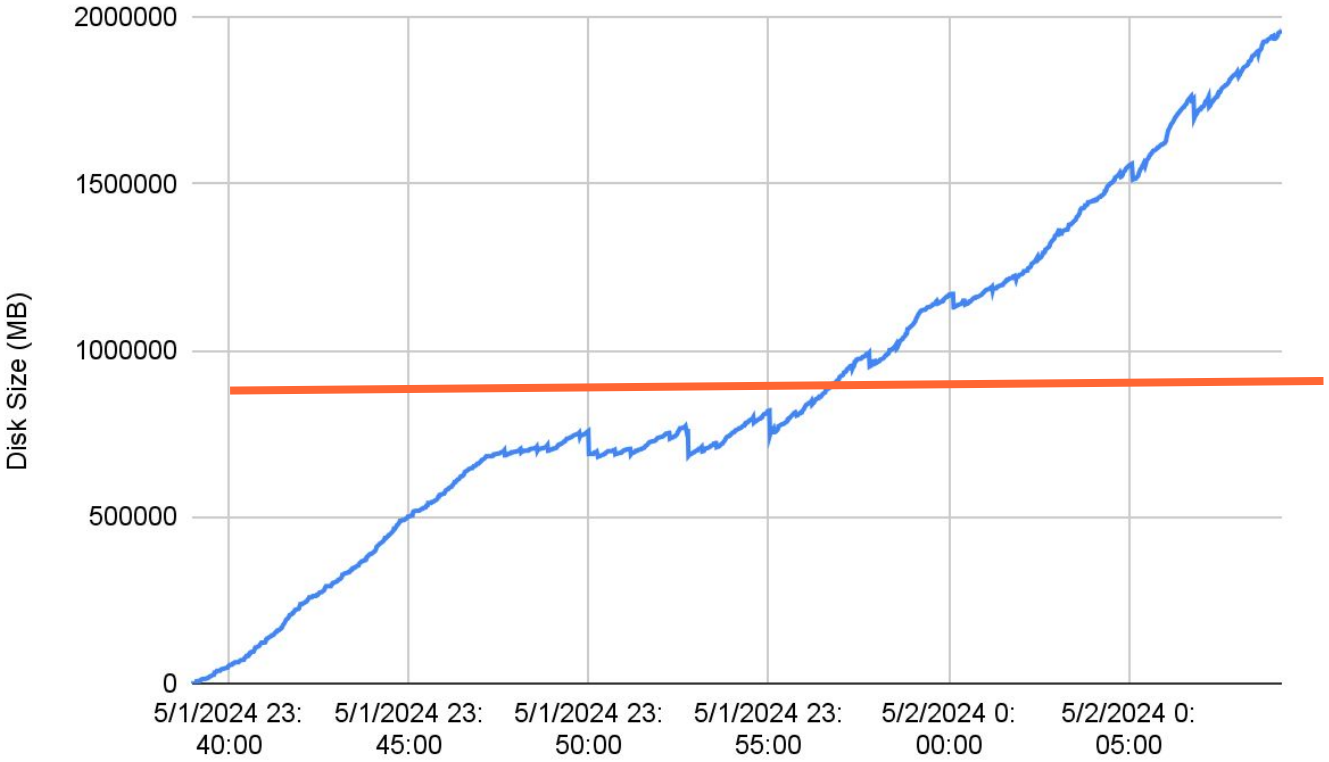
```
(-(((((avg_over_time(snmp{direction="in",snmptype="transit"}[24h])
    or
  avg_over_time(snmp{direction="in",snmptype="transit"}[24h]))
    and on (name)
  metadata{environment="prod",is_mcp="1"})
    - on (description, name) (
  (avg_over_time(snmp{direction="in",snmptype="transit"}[24h])
    or
  avg_over_time(snmp{direction="in",snmptype="transit"}[24h]))
    and on (name)
  metadata{environment="prod",is_mcp="1"})))) *
  (((
  avg_over_time(snmp{direction="in",snmptype="transit"}[24h])
    or
  avg_over_time(snmp{direction="in",snmptype="transit"}[24h]))
    and on (name) metadata{environment="prod",is_mcp="1"})
   - on (description, name) ((
  avg_over_time(snmp{direction="in",snmptype="transit"}[24h])
    or
  avg_over_time(snmp{direction="in",snmptype="transit"}[24h]))
    and
  on (name) metadata{environment="prod"}))))) ^ 0.5
```

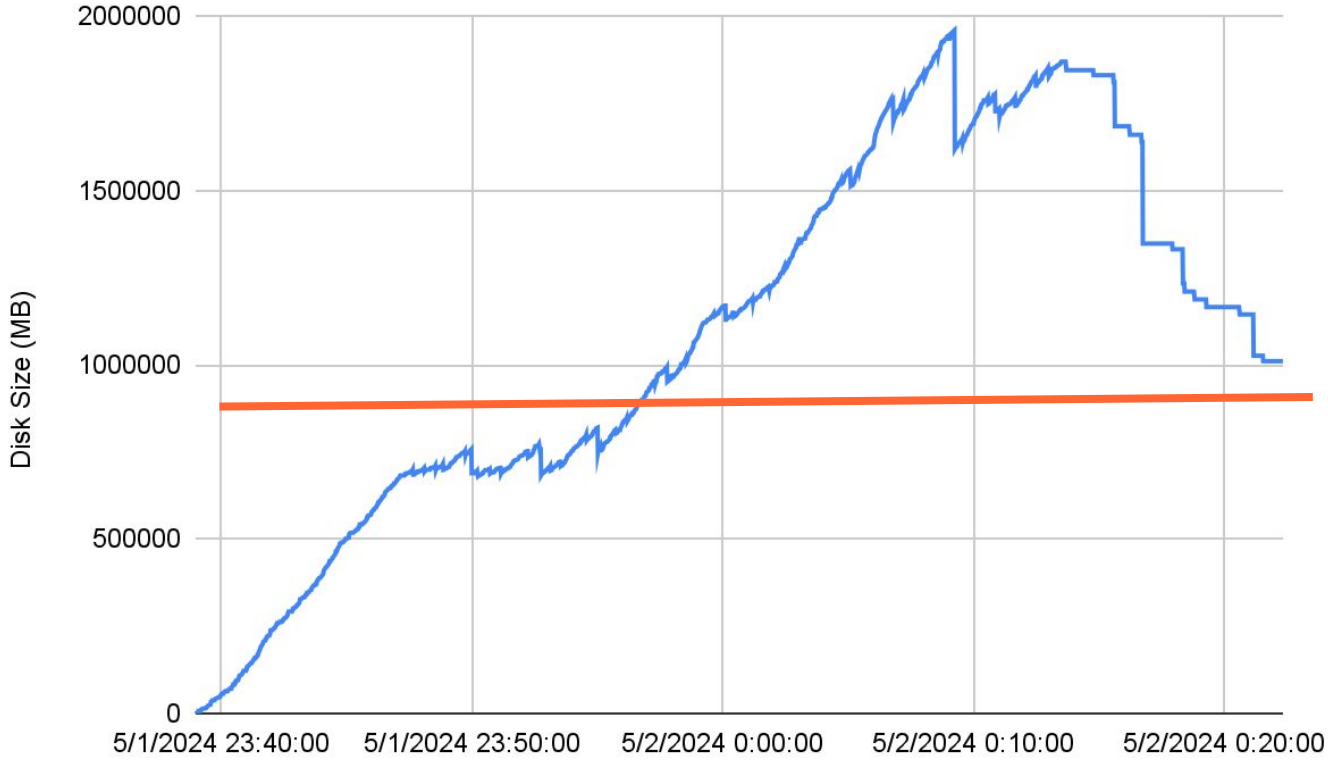
# Intrepid first steps

## Let’s see if we can pull some numbers

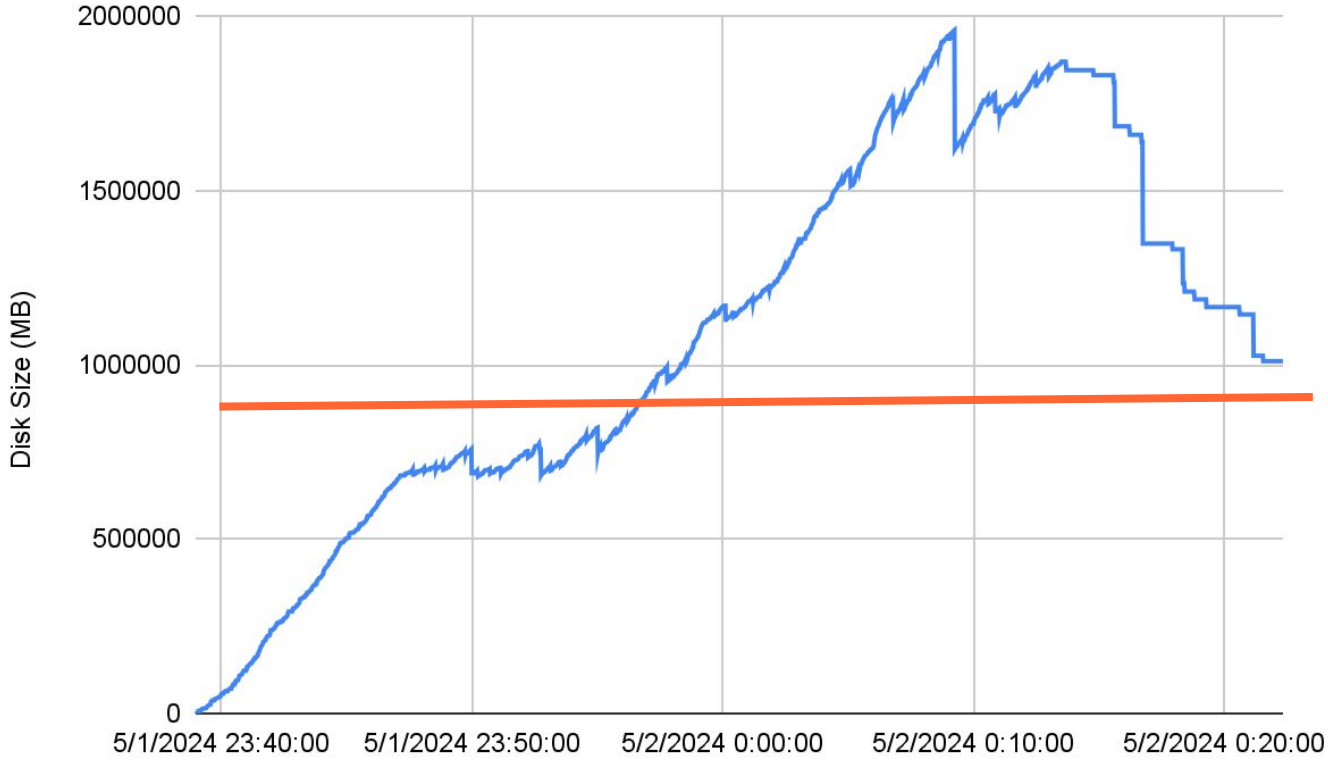


```
CREATE TABLE IF NOT EXISTS metrics (
    timestamp DateTime CODEC(DoubleDelta, ZSTD),
    name String CODEC(ZSTD),
    labels Map(String, String) CODEC(ZSTD),
    value Float64 CODEC (DoubleDelta, ZSTD),
) ENGINE = MergeTree() PRIMARY KEY (name, labels, timestamp);
```


2000000
1500000
1000000
500000
0
Disk Size (MB)
5/1/2024 23:40:00
5/1/2024 23:45:00
5/1/2024 23:50:00
5/1/2024 23:55:00
5/2/2024 0:00:00
5/2/2024 0:05:00

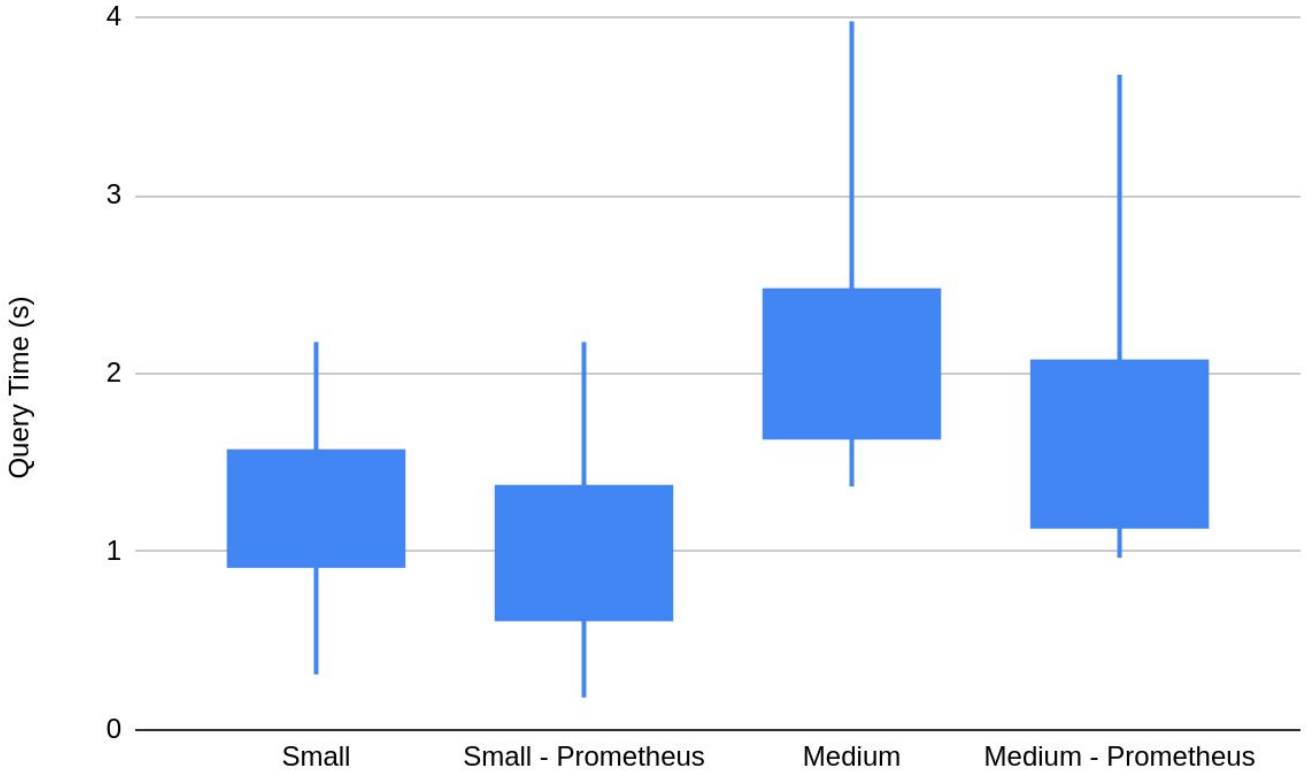

2000000
1500000
1000000
500000
0
Disk Size (MB)
5/1/2024 23:40:00
5/1/2024 23:50:00
5/2/2024 0:00:00
5/2/2024 0:10:00
5/2/2024 0:20:00

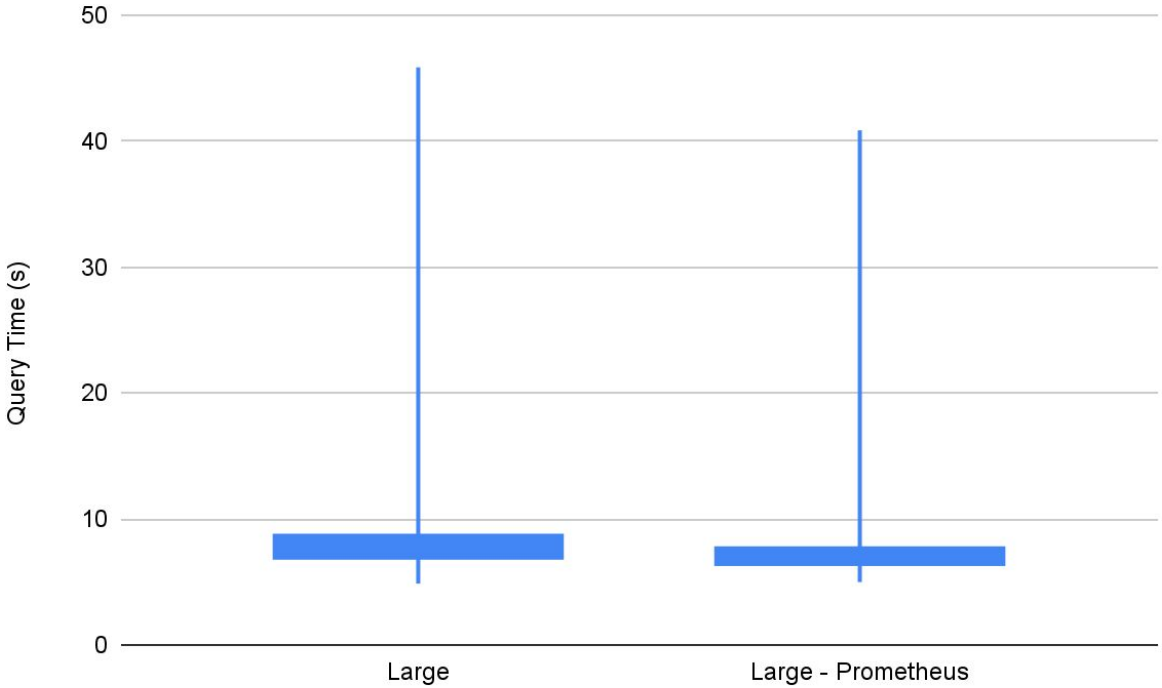
```
CREATE TABLE IF NOT EXISTS metrics (
    timestamp DateTime CODEC(DoubleDelta, ZSTD),
    name String CODEC(ZSTD),
    labels Map(String, String) CODEC(ZSTD),
    value Float64 CODEC (DoubleDelta, ZSTD),
) ENGINE = MergeTree() PRIMARY KEY (name, labels, timestamp);
```


2000000
1500000
1000000
500000
0
Disk Size (MB)
5/1/2024 23:40:00
5/1/2024 23:50:00
5/2/2024 0:00:00
5/2/2024 0:10:00
5/2/2024 0:20:00

CLOUDFLARE

| Small Queries | Medium Queries | Large Queries |
| --- | --- | --- |
| Median Query Time: | Median Query Time: | Median Query Time: |
| 0.386s | 0.734s | 1.623s |


Query Time (s)
4
3
2
1
0
Small
Small - Prometheus
Medium
Medium - Prometheus

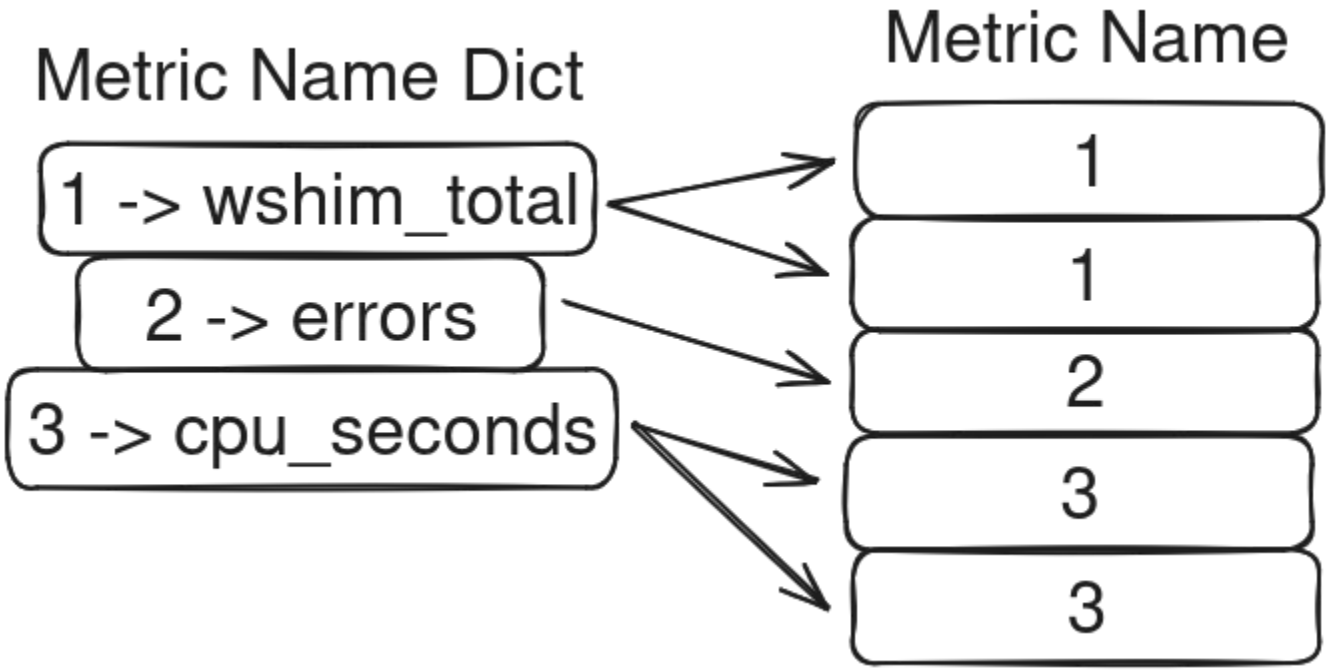

50
40
30
20
10
0
Query Time (s)
Large
Large - Prometheus

# Low Cardinality Fields

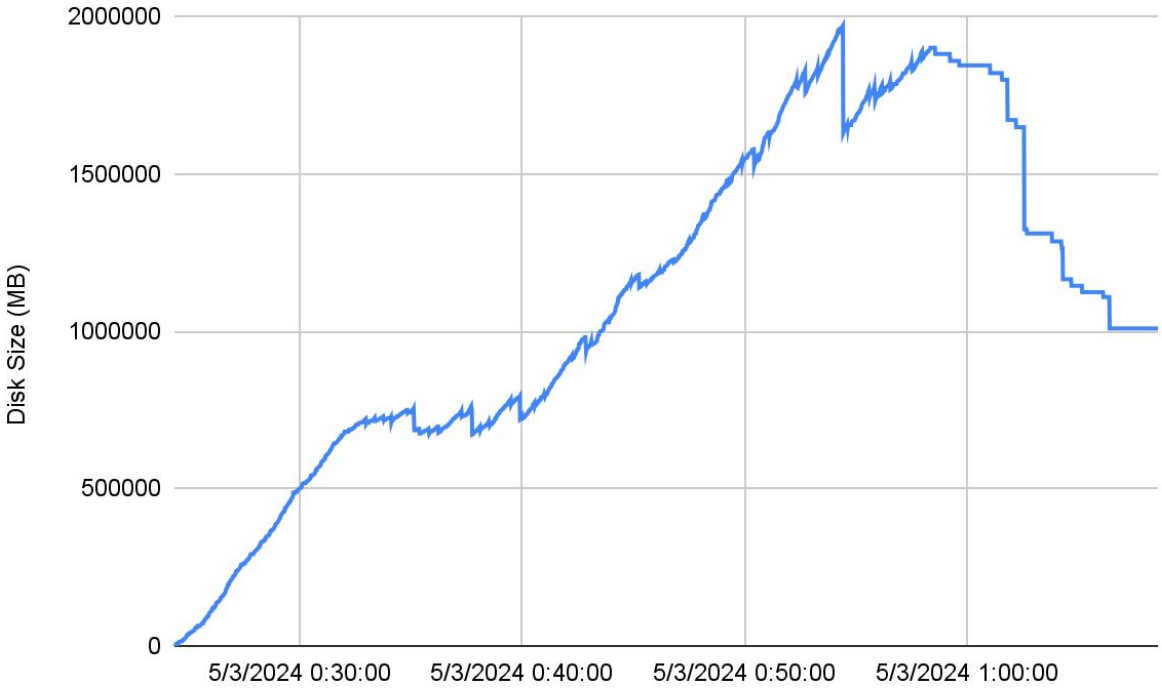
**Wait… Don’t we want *high cardinality* fields?**

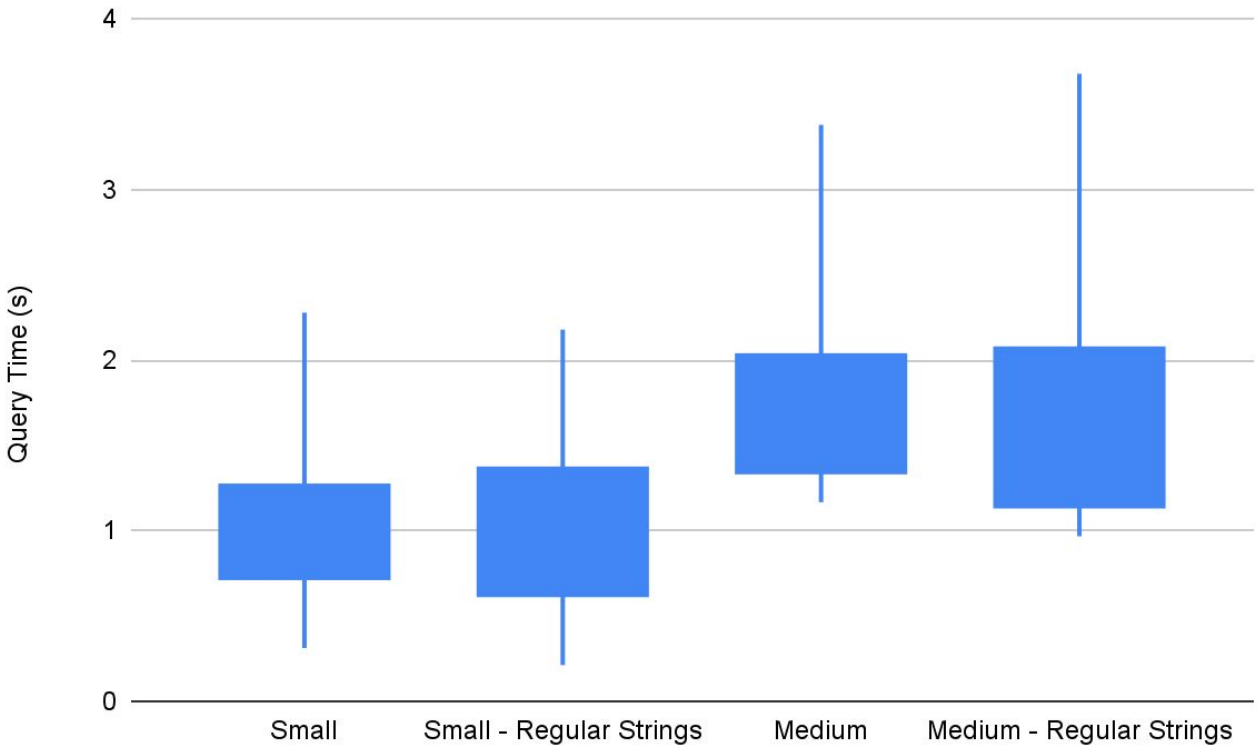
CLOUDFLARE

Low Cardinality to Prometheus is ***vastly*** different to Low Cardinality to Clickhouse


Metric Name Dict
1 -> wshim_total
2 -> errors
3 -> cpu_seconds
Metric Name
1
1
2
3
3

```
CREATE TABLE IF NOT EXISTS metrics (
    timestamp DateTime CODEC(DoubleDelta, ZSTD),
    name LowCardinality(String) CODEC(ZSTD),
    labels Map(LowCardinality(String), LowCardinality(String)
CODEC(ZSTD),
    value Float64 CODEC (DoubleDelta, ZSTD),
) ENGINE = MergeTree() PRIMARY KEY (name, labels, timestamp);
```


2000000
1500000
1000000
500000
0
Disk Size (MB)
5/3/2024 0:30:00
5/3/2024 0:40:00
5/3/2024 0:50:00
5/3/2024 1:00:00

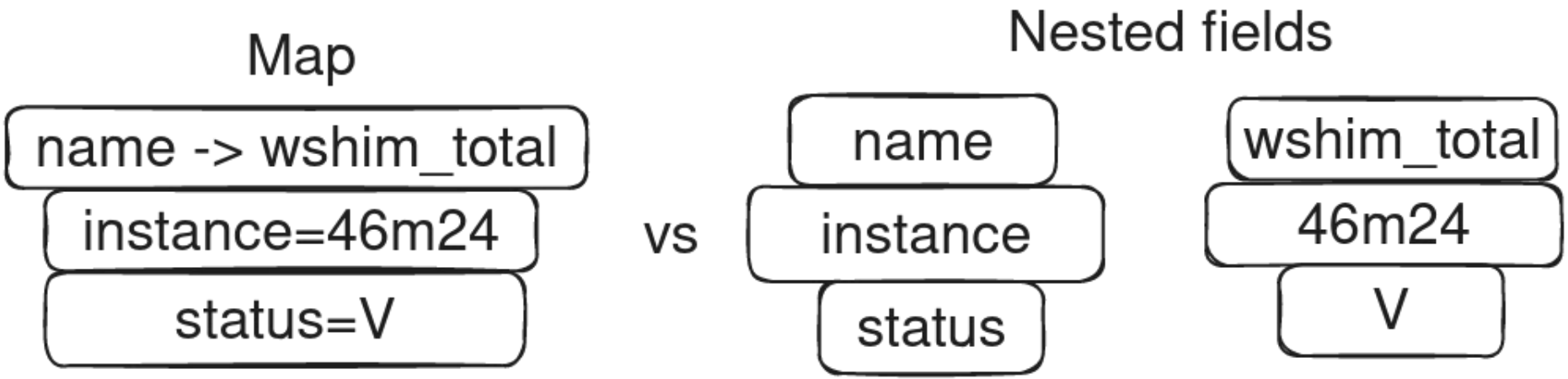

4
3
2
1
0
Query Time (s)
Small
Small - Regular Strings
Medium
Medium - Regular Strings

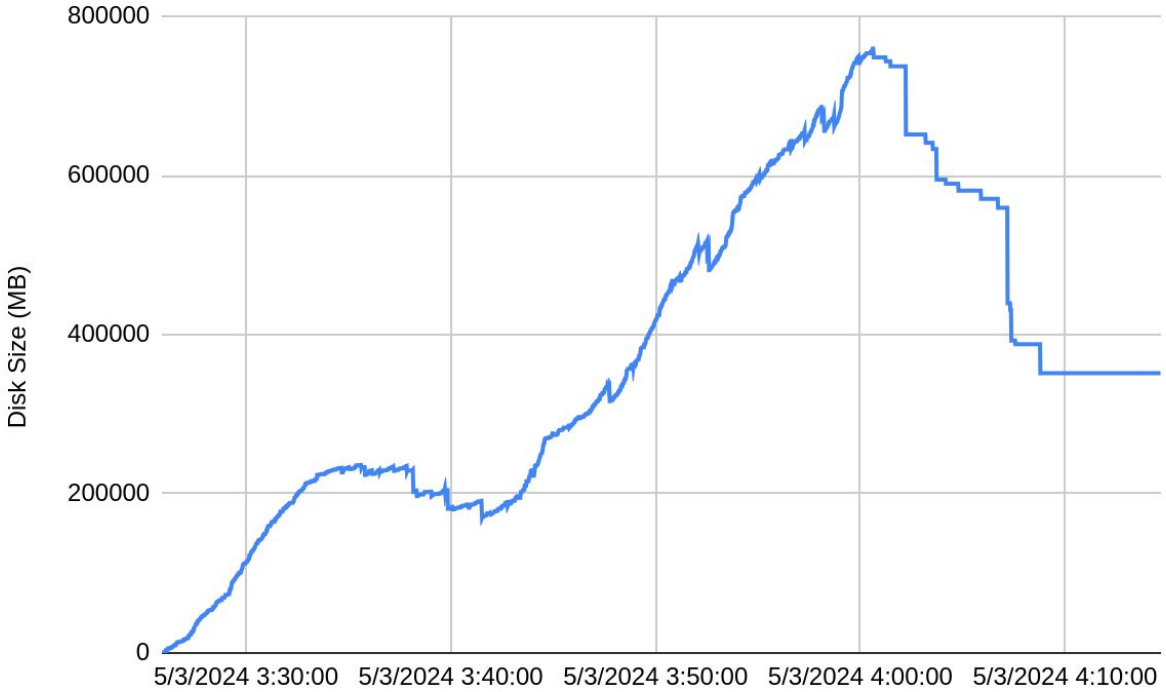
# Nested Fields

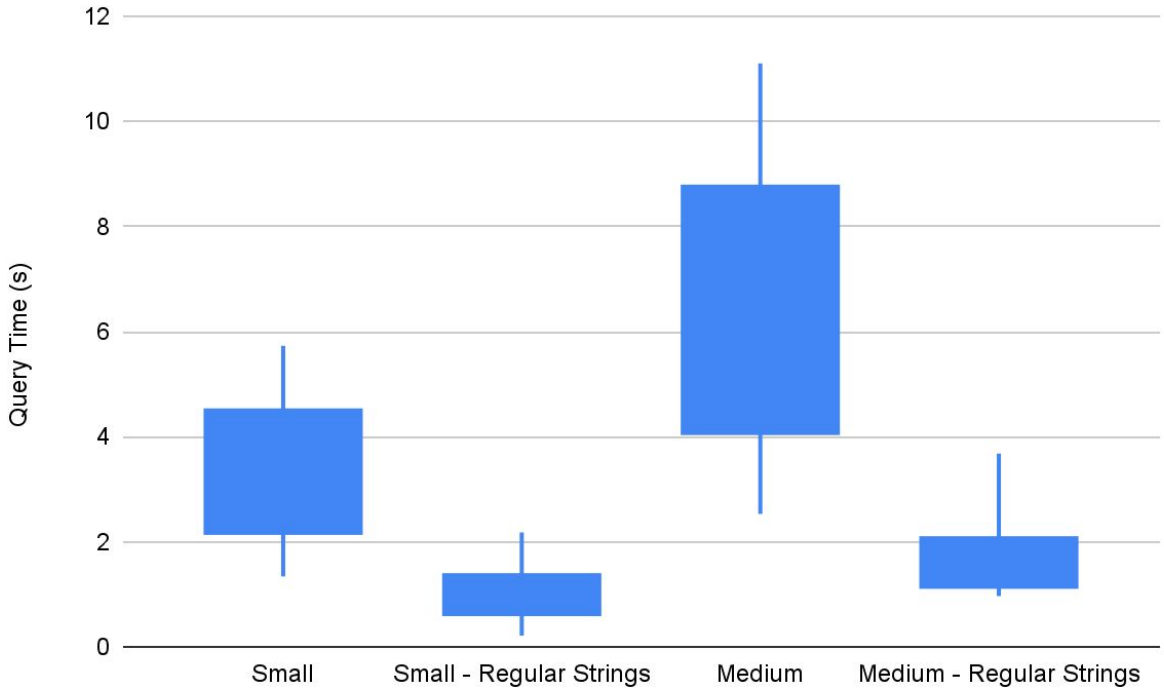
**Maps aren’t the only way to store KVs**

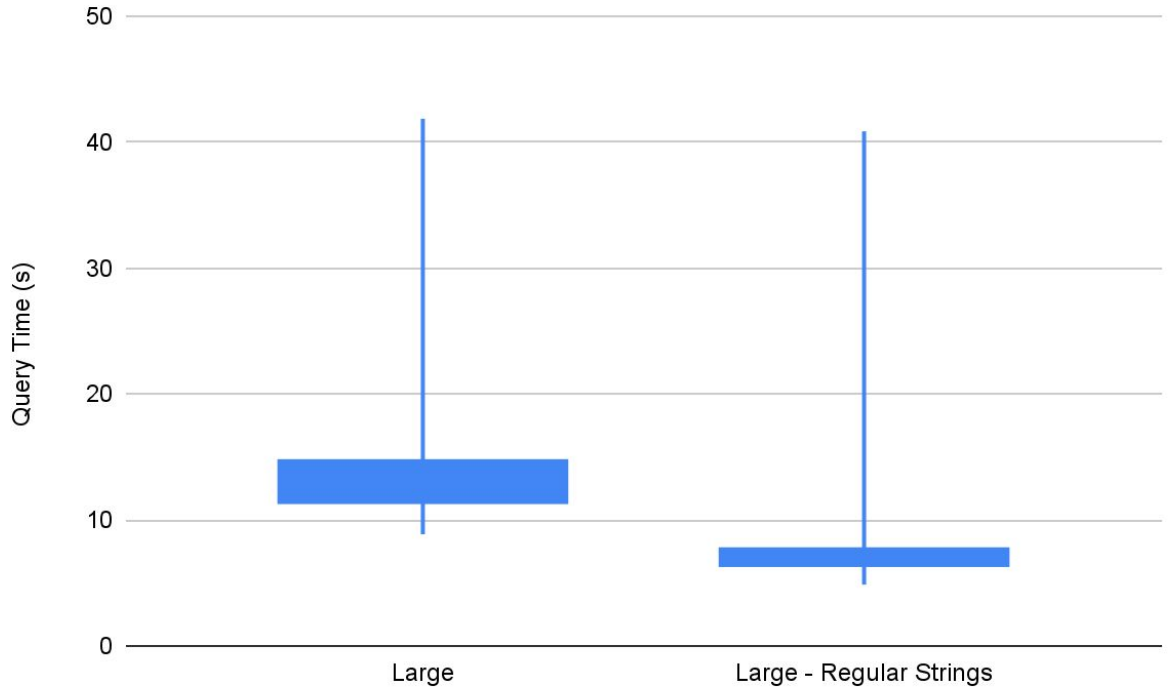
CLOUDFLARE

```
CREATE TABLE IF NOT EXISTS metrics (
    timestamp DateTime CODEC(DoubleDelta, ZSTD),
    name LowCardinality(String) CODEC(ZSTD),
    labels Nested (
        key String,
        value String
    ),
    value Float64 CODEC (DoubleDelta, ZSTD),
) ENGINE = MergeTree() PRIMARY KEY (name, labels, timestamp);
```


Map
name -> wshim_total
instance=46m24
status=V
vs
Nested fields
name
instance
status
wshim_total
46m24
V

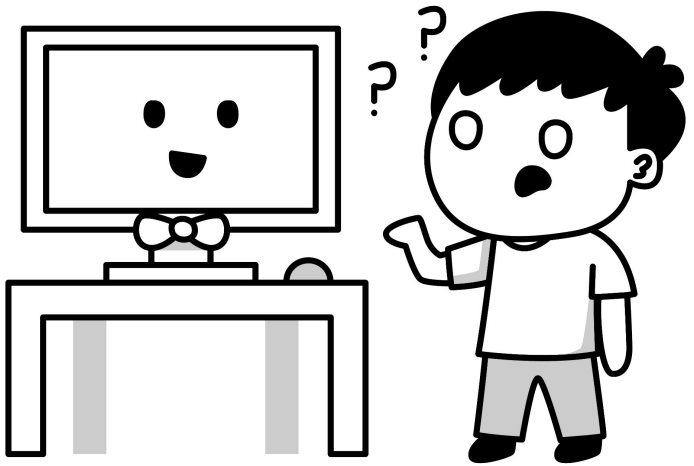

800000
600000
400000
200000
0
Disk Size (MB)
5/3/2024 3:30:00
5/3/2024 3:40:00
5/3/2024 3:50:00
5/3/2024 4:00:00
5/3/2024 4:10:00

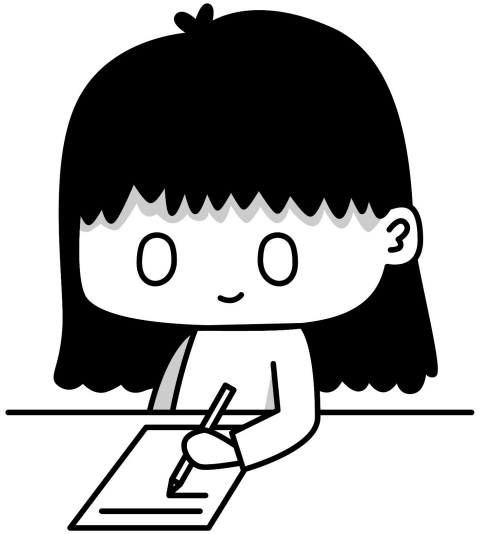

12
10
8
6
4
2
0
Query Time (s)
Small
Small - Regular Strings
Medium
Medium - Regular Strings


50
40
30
20
10
0
Query Time (s)
Large
Large - Regular Strings

# Prometheus Encodings

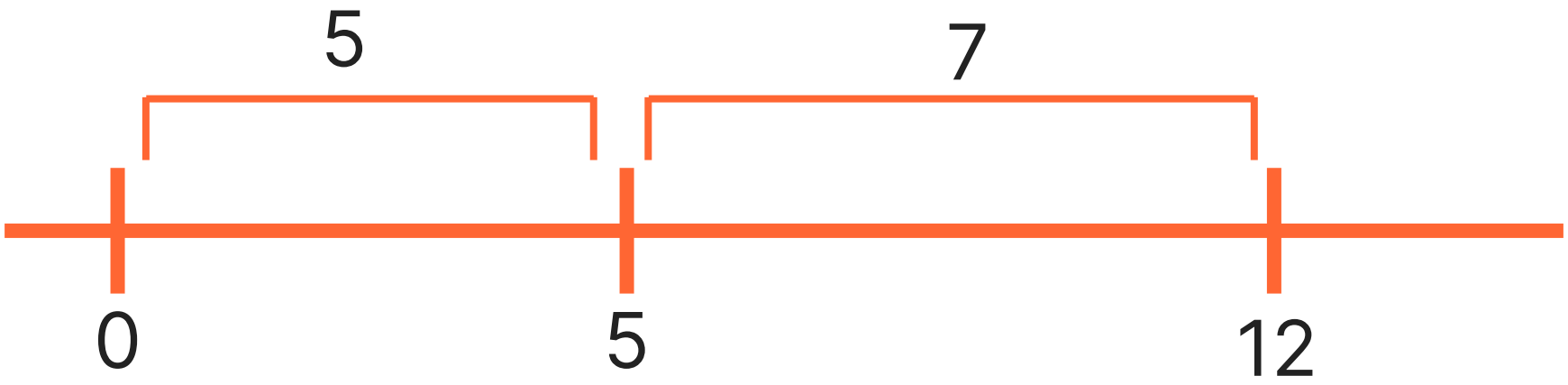
**How does this Prometheus thing work anyway?**



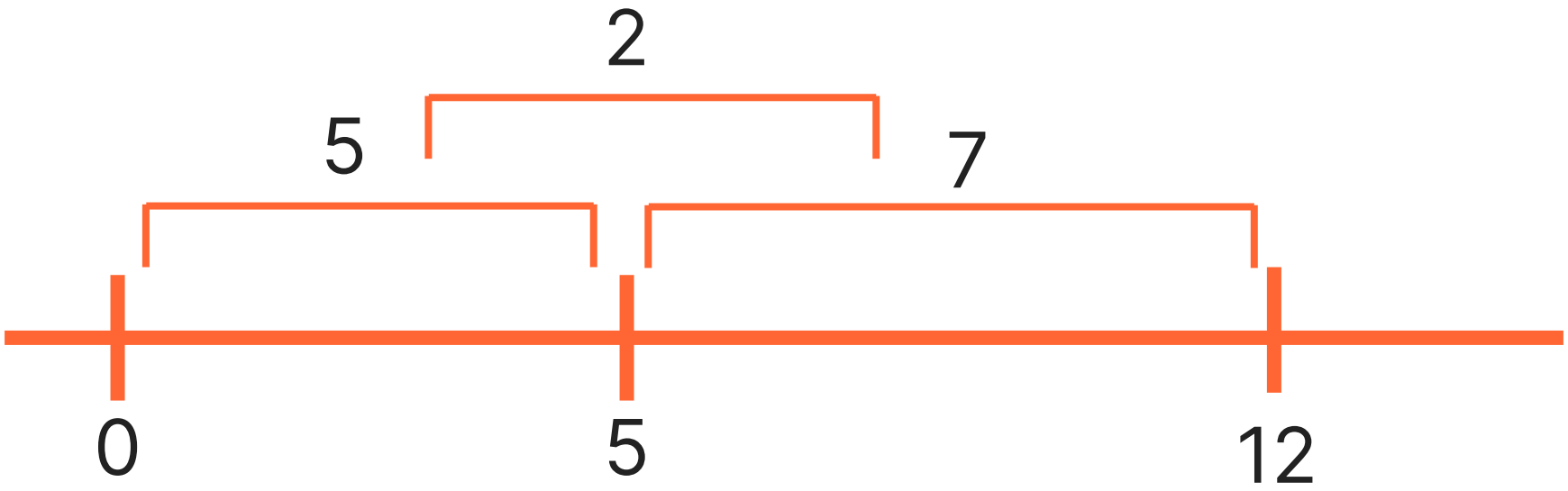


# Who here knows how Prometheus encodes data?

# Type 0: Delta Encoding

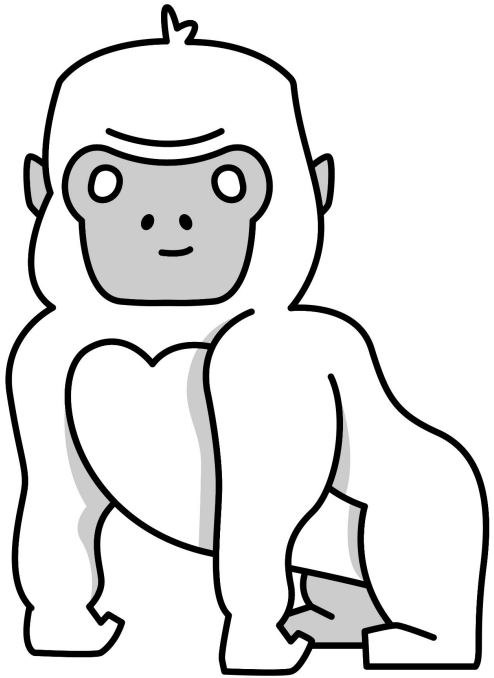


# Type 1: *Double* Delta

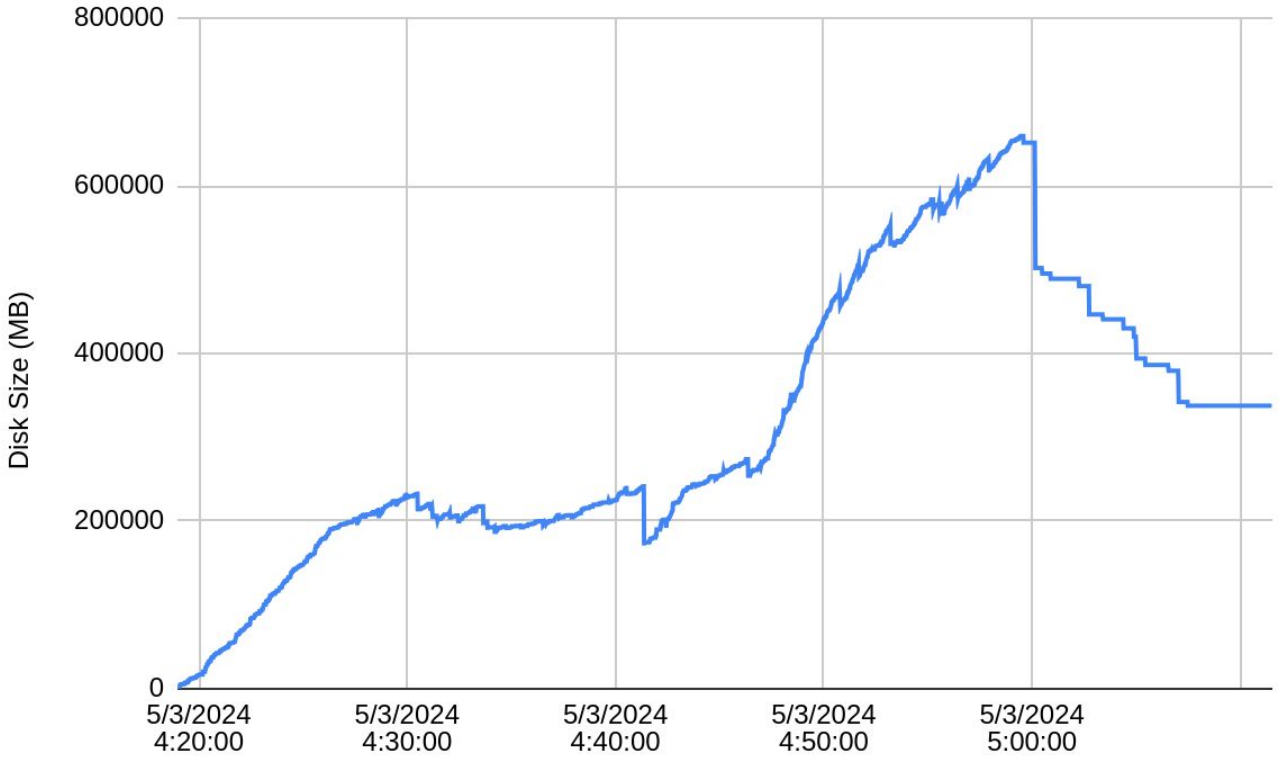


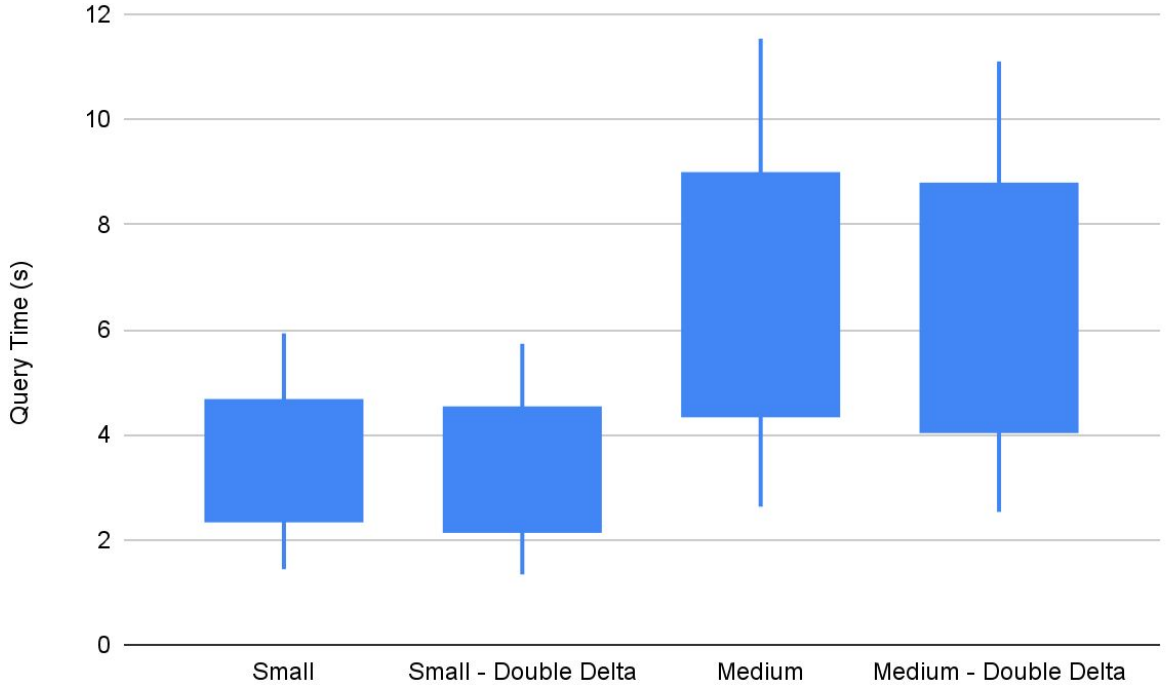
CLOUDFLARE

# Type 2:
# *Gorilla Encoding*



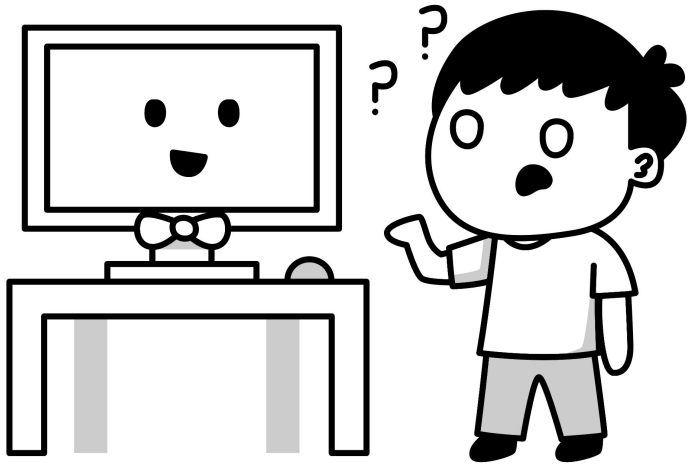
```
CREATE TABLE IF NOT EXISTS metrics (
    timestamp DateTime CODEC(DoubleDelta, ZSTD),
    name String CODEC(ZSTD),
    labels Nested (
        key String,
        value String
    ),
    value Float64 CODEC (Gorilla, ZSTD),
) ENGINE = MergeTree() PRIMARY KEY (name, labels, timestamp);
```


800000
600000
400000
200000
0
Disk Size (MB)
5/3/2024 4:20:00
5/3/2024 4:30:00
5/3/2024 4:40:00
5/3/2024 4:50:00
5/3/2024 5:00:00


12
10
8
6
4
2
0
Query Time (s)
Small
Small - Double Delta
Medium
Medium - Double Delta

# Query Speeds

**We've made it small, let's make it fast**



## What's actually being slow?

Lots of **ors**:

```
errors_total{instance=~"41m3|41m4|42m5"}
```

Lots of **prefix matches**:

```
errors{instance=~"41.+"}
```

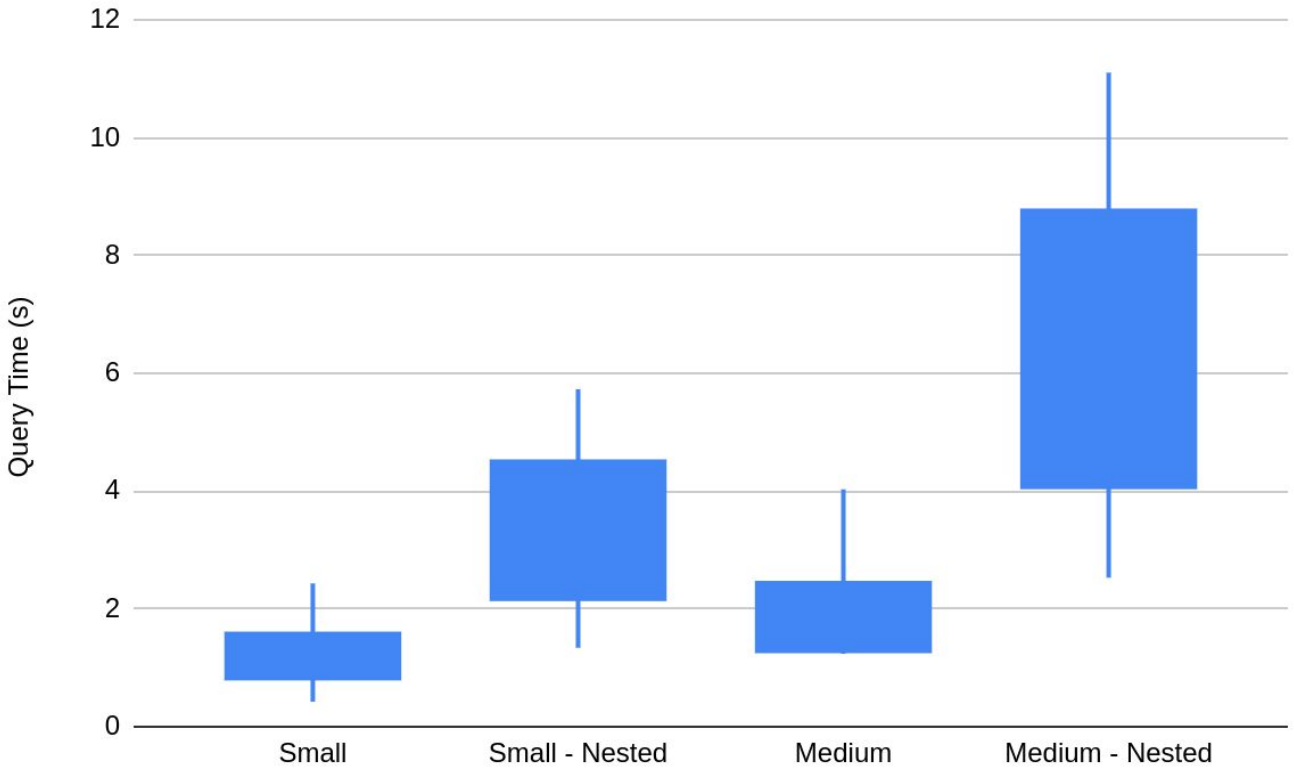
Some **contains**:

```
errors_total{instance=~".+m.+"}
```

Very few actual regexes

CLOUDFLARE

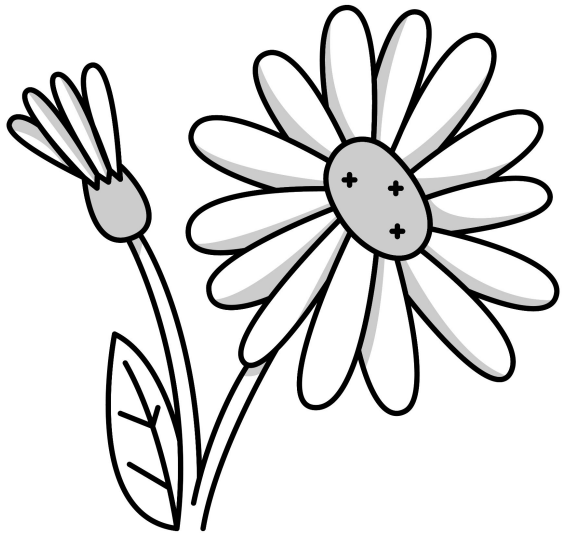
```
instance=~"41m3|41m4" -> instance='41m3' or instance='41m4'

instance=~"41m.*" -> startsWith(instance, '41m')

instance=~".+m.+" -> like(instance, '%m%')
```

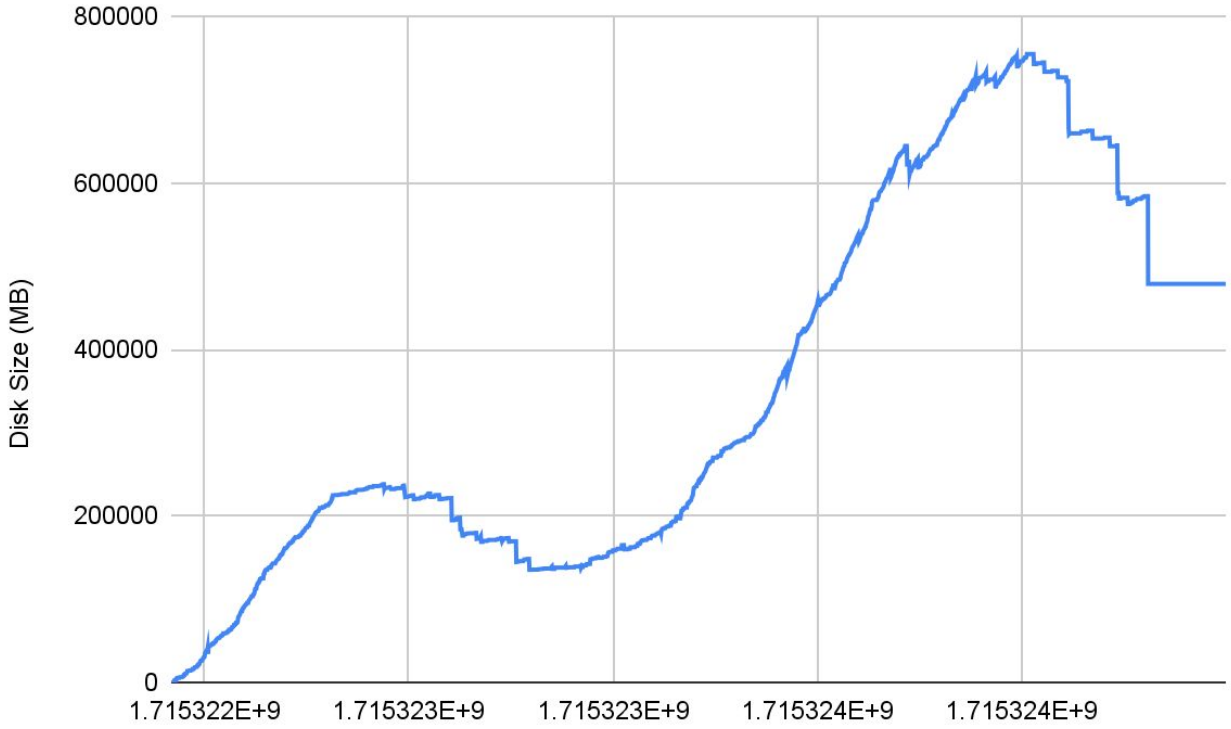

12
10
8
6
4
2
0
Query Time (s)
Small
Small - Nested
Medium
Medium - Nested

```
CREATE TABLE IF NOT EXISTS metrics (
    timestamp DateTime CODEC(DoubleDelta, ZSTD),
    name String CODEC(ZSTD),
    labels Nested (
        key String,
        value String
    ),
    value Float64 CODEC (DoubleDelta, ZSTD),
) ENGINE = MergeTree() PRIMARY KEY (name, labels, timestamp);
```

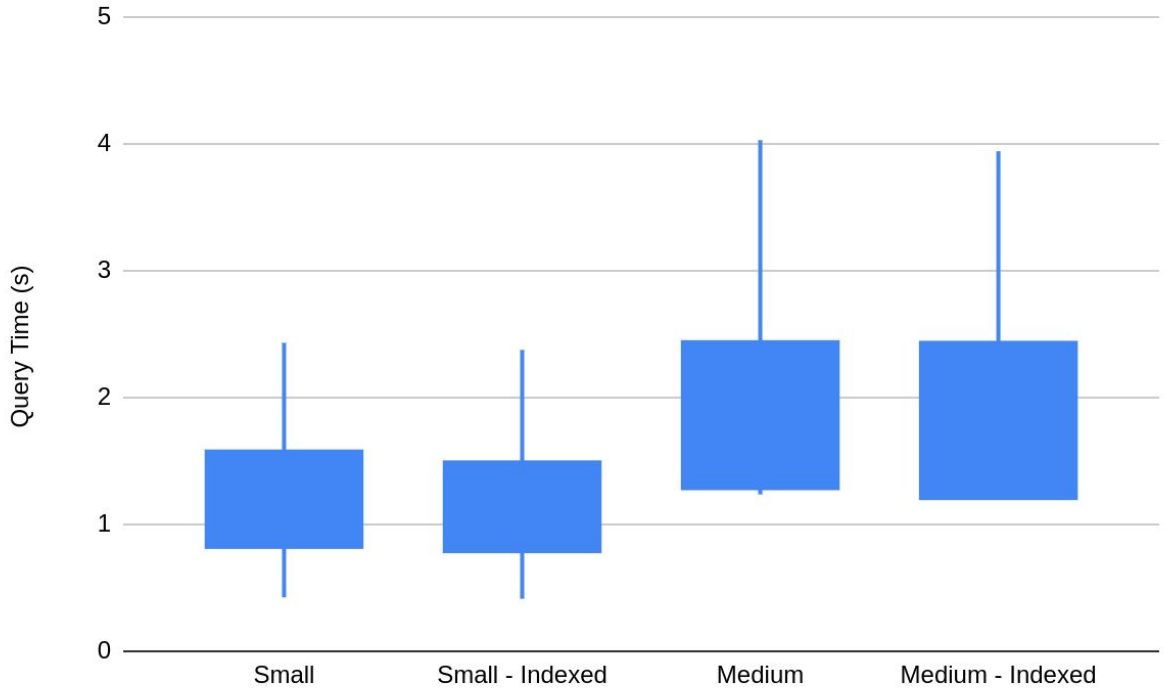
# Bloom Filter (ngrambf_v1)

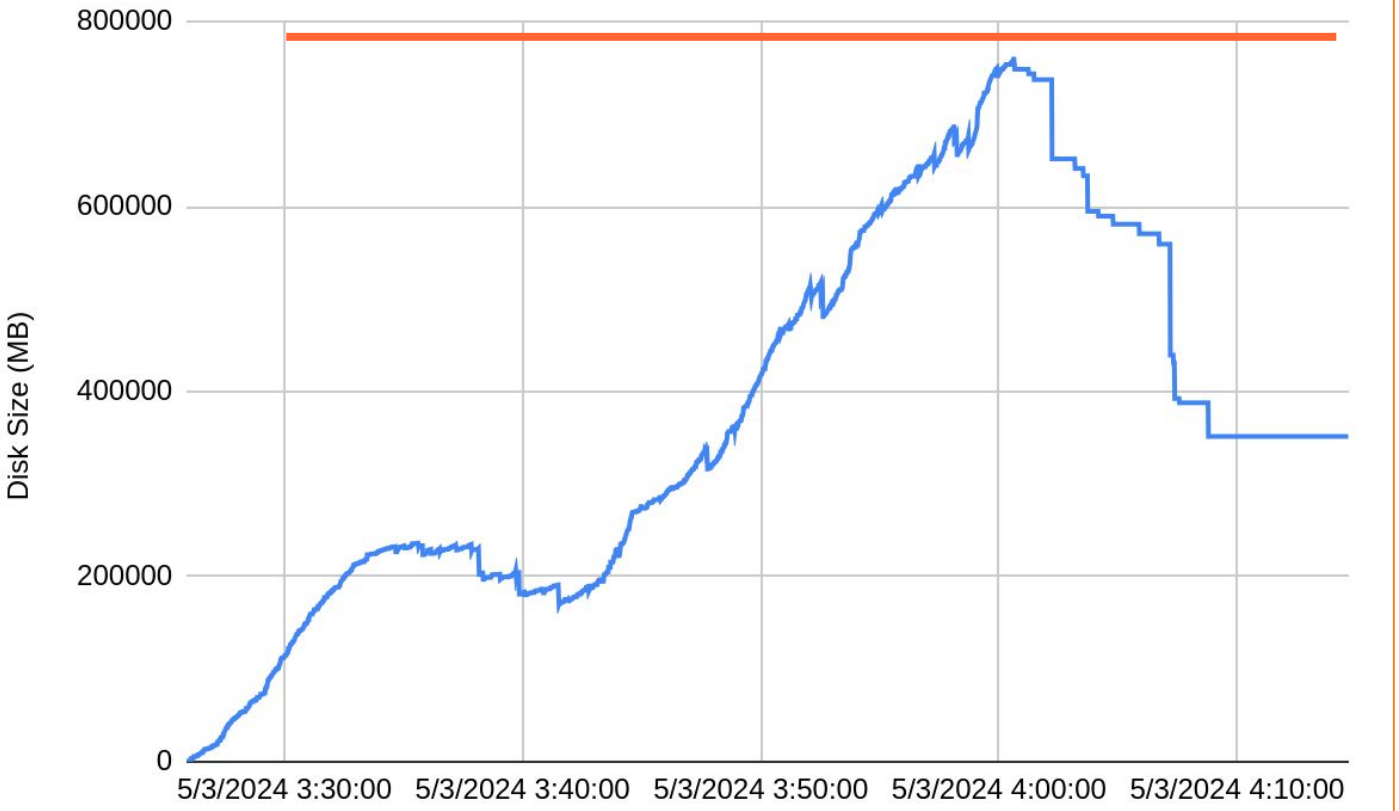
```
ALTER TABLE metrics
    ADD INDEX tags_index tags.value
    TYPE ngrambf_v1(5, 256, 3, 1)
    GRANULARITY 1;
```

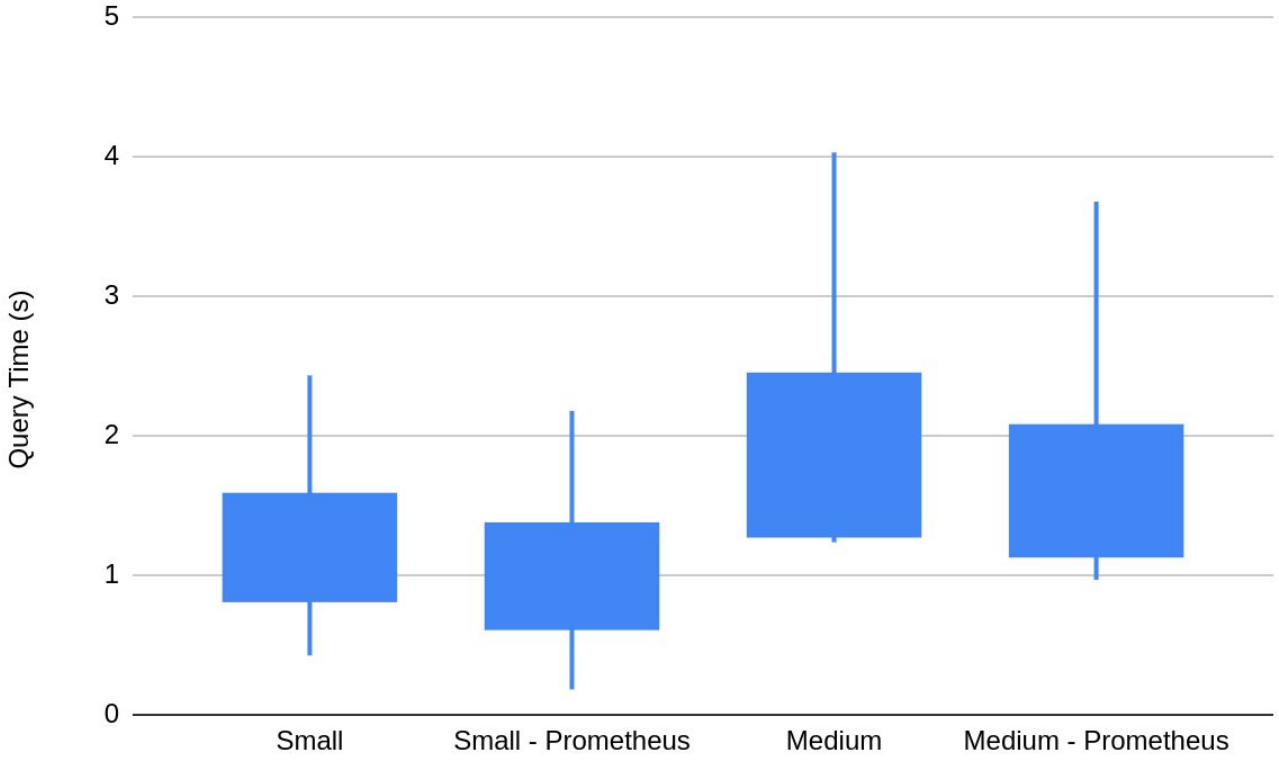



800000
600000
400000
200000
0
Disk Size (MB)
1.715322E+9
1.715323E+9
1.715323E+9
1.715324E+9
1.715324E+9

# Bloom Filter Query Times




800000
600000
400000
200000
0
Disk Size (MB)
5/3/2024 3:30:00
5/3/2024 3:40:00
5/3/2024 3:50:00
5/3/2024 4:00:00
5/3/2024 4:10:00


Query Time (s)
5
4
3
2
1
0
Small
Small - Prometheus
Medium
Medium - Prometheus

CLOUDFLARE

## Future Work

- Testing in real world scraping scenarios
- Testing with Clickhouse's "ReplicatedMergeTree"
- Splitting Labels out into their own table with a "ReplacingMergeTree"
- Tuning our indexes further
- Investigating other compression algorithms

CLOUDFLARE

# Thanks!

Art by Alicia Edwards ^

Slides ^