Grafana Labs

Monitoring Latencies
How fast is your REST service?

Fabian Stäber

Devoxx Belgium 2022

Fabian Stäber

Engineering Manager at Grafana Labs

Prometheus team member; maintainer of the Prometheus Java client library

@fstabr

Contents

Part 1
(demo)

Part 2
(slides)

```
long start = System.nanoTime();
try {
        // do something
} finally {
        long duration = System.nanoTime() - start;
        observe(duration);
}
```

Contents

Part 1 (demo)

- Response time vs service time
- Load test backs off when server is at its limit

Part 2 (slides)

- Average & Maximum
- Percentiles
- Histograms
- More Histograms

Evaluation Criteria

- Time Interval
  - Ad-hoc at query time, or predefined in the instrumentation library?
- Aggregation
  - Across multiple instances of a service?
- Practical Value
  - Opinionated

Average

Total time serving requests

```
increase(http_server_requests_seconds_sum[5m]) /
increase(http_server_requests_seconds_count[5m])
```

Total number of requests

You cannot calculate an average of averages.
But you can sum up total times and total counts.

Average

- Time Interval
  - Ad-hoc at query time, like [5m] (*)
- Aggregation
  - Yes (*)
- Practical Value
  - Available by default in Spring Boot (Micrometer)
  - Affected by outliers. Example: 1000 calls take 20ms each; 1 client hangs and times out after 30s → average = 50ms

---

(*) if the instrumentation provides sum and count individually.

Maximum

http_server_requests_seconds_max

↳ As provided by Micrometer

max(http_server_requests_seconds_max**)**

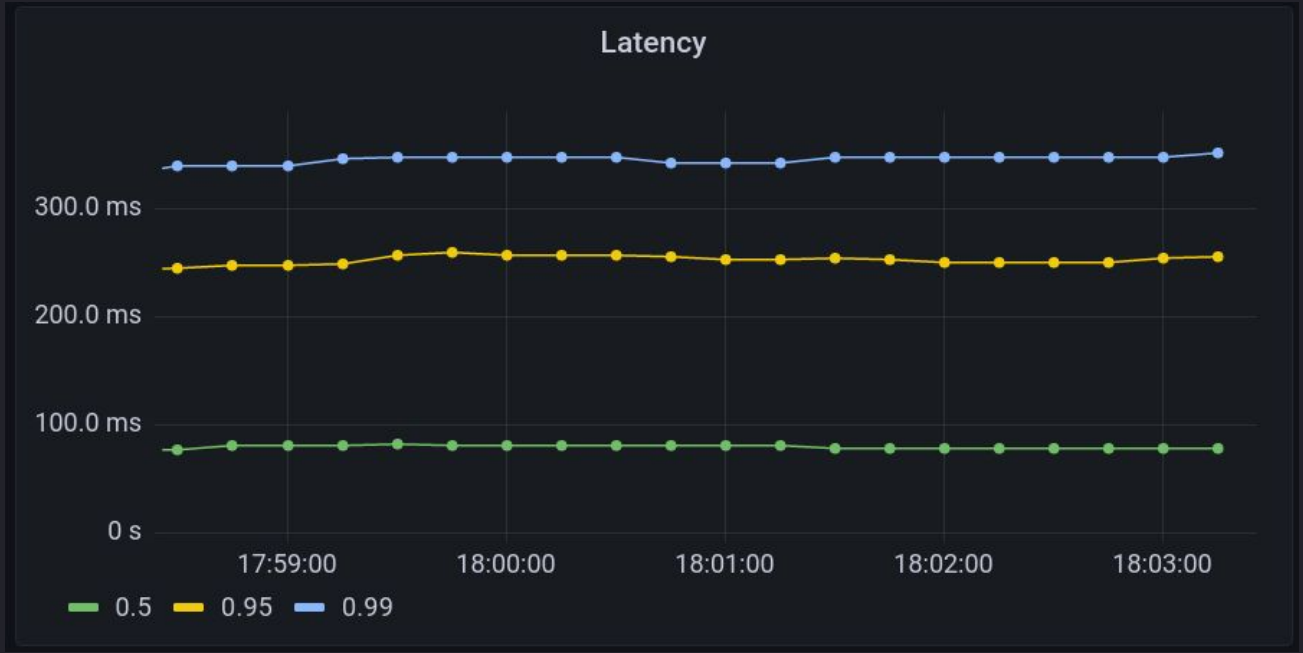
↳ Aggregation

Maximum

- Time Interval
  - Predefined in the instrumentation library
  - Spring Boot (Micrometer): Sliding window, default 3 minutes, moved forward every 1 minute
- Aggregation
  - Yes
- Practical Value
  - Good for load tests
  - Limited value for production:
    - If you are on-call, do you want to be paged when a single outlier takes long?
    - Outliers often caused by client issues, nothing's wrong with your REST service.

Recommended talk: "How NOT to Measure Latency", Gil Tene, 2015

Percentiles

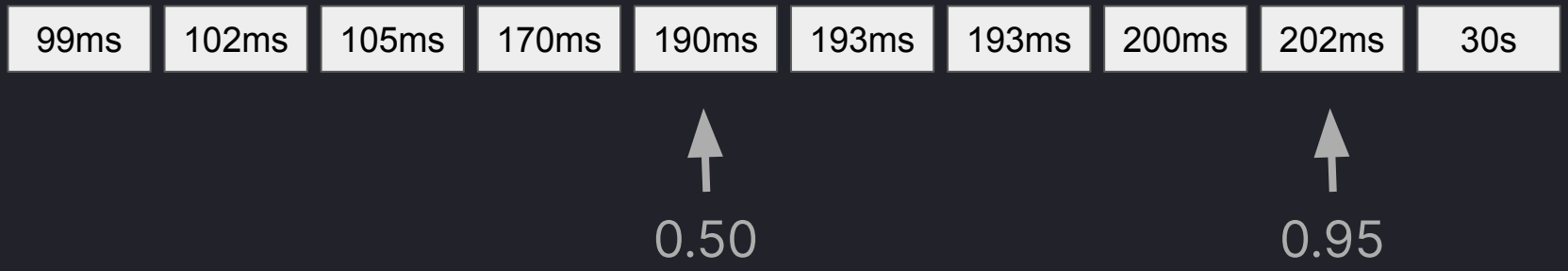


Prometheus client_java does not directly support max,
but you can use the 100th percentile for max and the 0th percentile for min.

Percentiles

Expensive:

Sorted list with all observations.

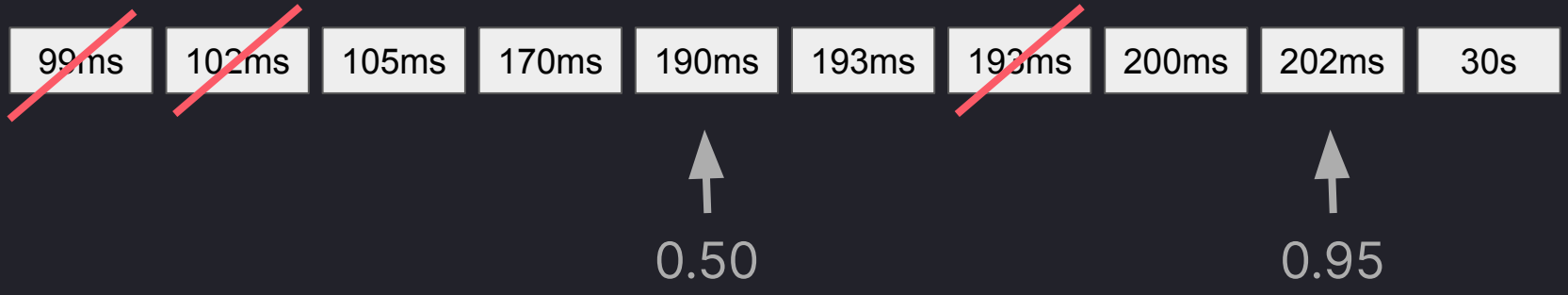


Percentiles

CKMS algorithm:

Allow some error to save space.

```
Summary.build()
    .quantile(0.5, 0.01)
    .quantile(0.95, 0.005)
    .quantile(0.99, 0.001);
```



G. Cormode, F. Korn, S. Muthukrishnan and D. Srivastava.
Effective Computation of Biased Quantiles over Data Streams

Percentiles

	.quantile(0.95, 0.005)	.quantile(0.95, 0.005) .quantile(0.99, 0.001)	.quantile(0.5, 0.01) .quantile(0.95, 0.005) .quantile(0.99, 0.001)
1,000	139	152	200
10,000	43	56	90
100,000	60	70	103
1,000,000	92	102	132
10,000,000	27	40	68
100,000,000	27	36	67

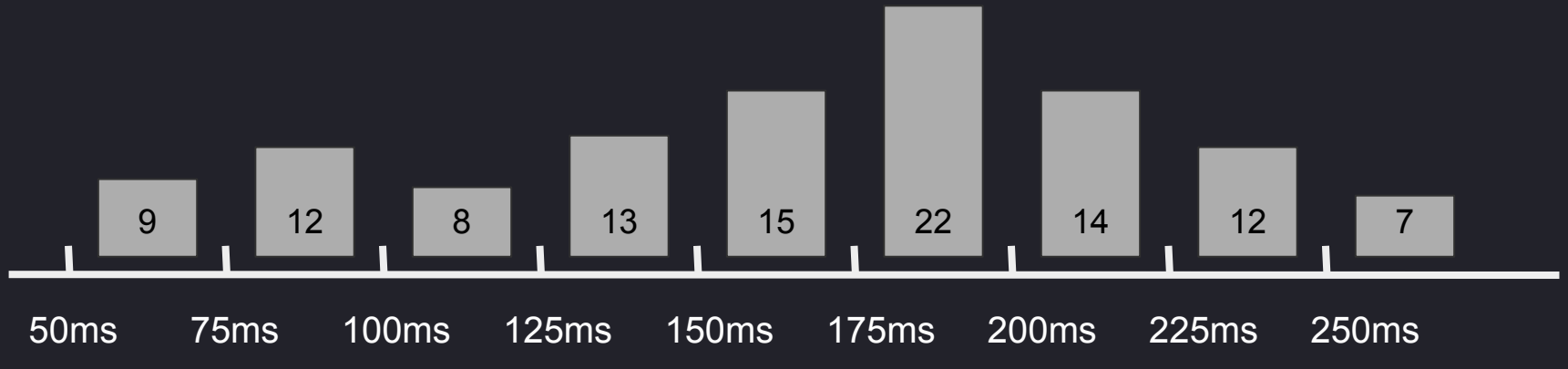
https://grafana.com/blog/2022/03/01/how-summary-metrics-work-in-prometheus/

Percentiles

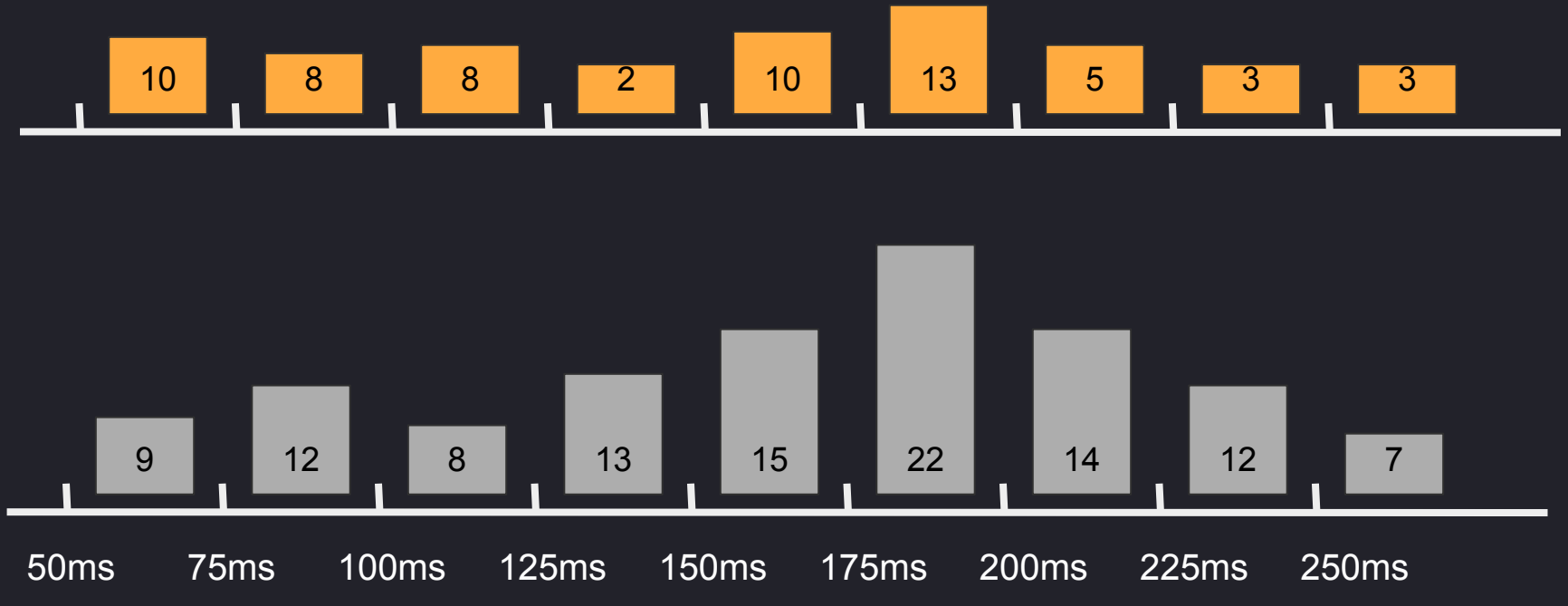
- Time Interval
  - Predefined in the instrumentation library
  - Prometheus: Sliding window, default 10 minutes, moved forward every 2 minutes
- Aggregation
  - No
- Practical Value
  - Percentiles are awesome, also good for alerting
  - Lack of aggregation is a bummer

You cannot calculate an average of averages.
You cannot calculate an average of percentiles.

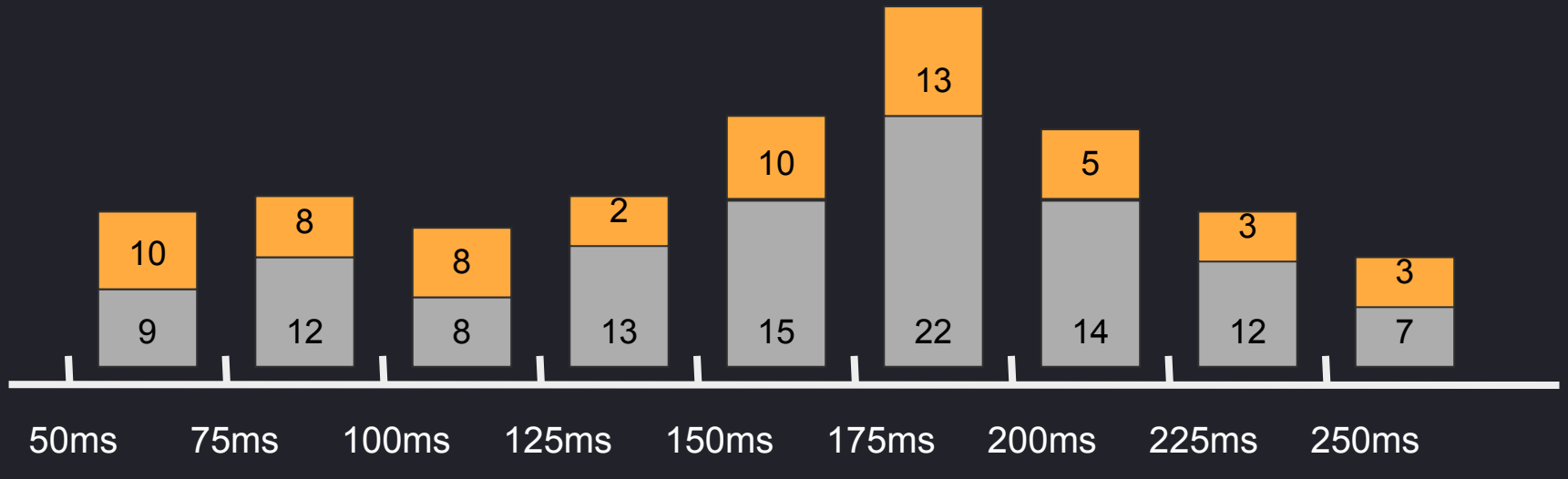
Histograms


9
12
8
13
15
22
14
12
7
50ms
75ms
100ms
125ms
150ms
175ms
200ms
225ms
250ms

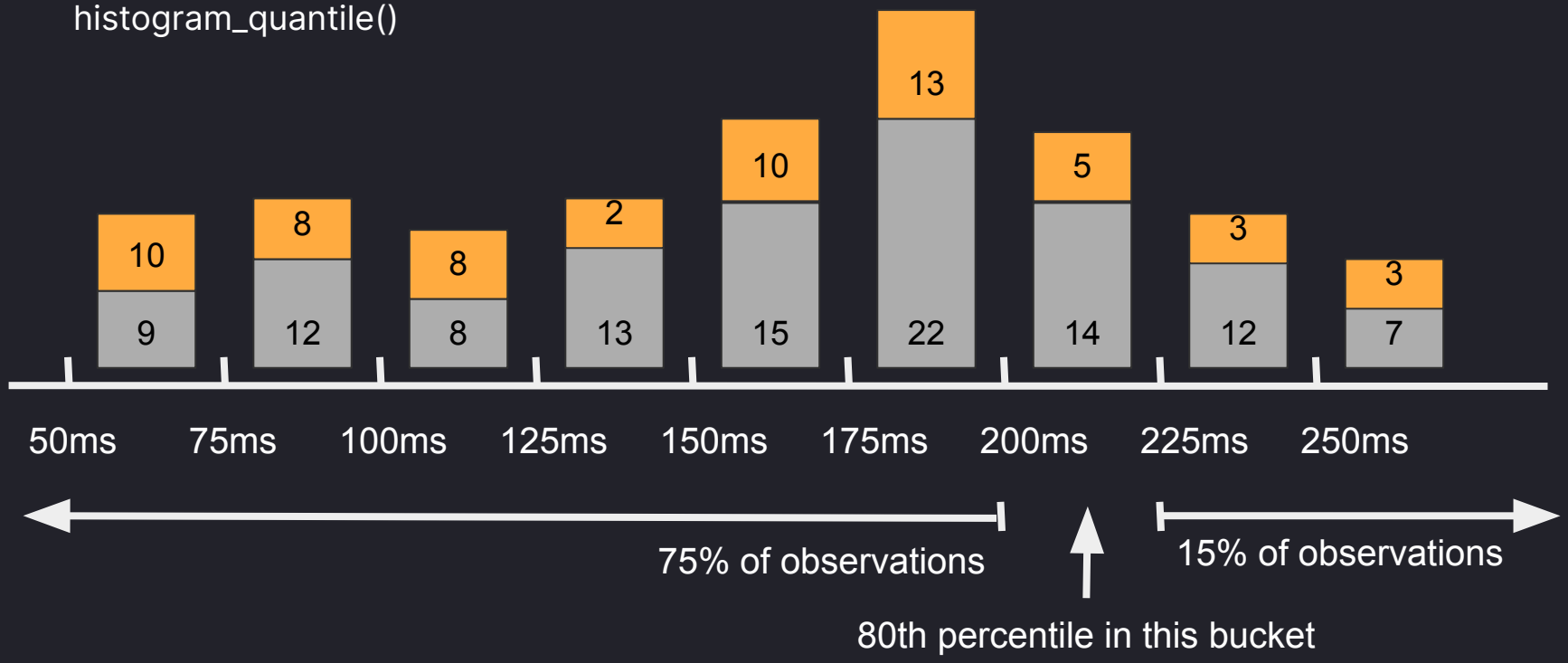
Histograms


10
8
8
2
10
13
5
3
3
9
12
8
13
15
22
14
12
7
50ms
75ms
100ms
125ms
150ms
175ms
200ms
225ms
250ms

Histograms


10
9
8
12
8
8
2
13
10
15
13
22
5
14
3
12
3
7
50ms
75ms
100ms
125ms
150ms
175ms
200ms
225ms
250ms

Histograms


histogram_quantile()
10
9
8
12
8
8
2
13
10
15
13
22
5
14
3
12
3
7
50ms
75ms
100ms
125ms
150ms
175ms
200ms
225ms
250ms
75% of observations
15% of observations
80th percentile in this bucket

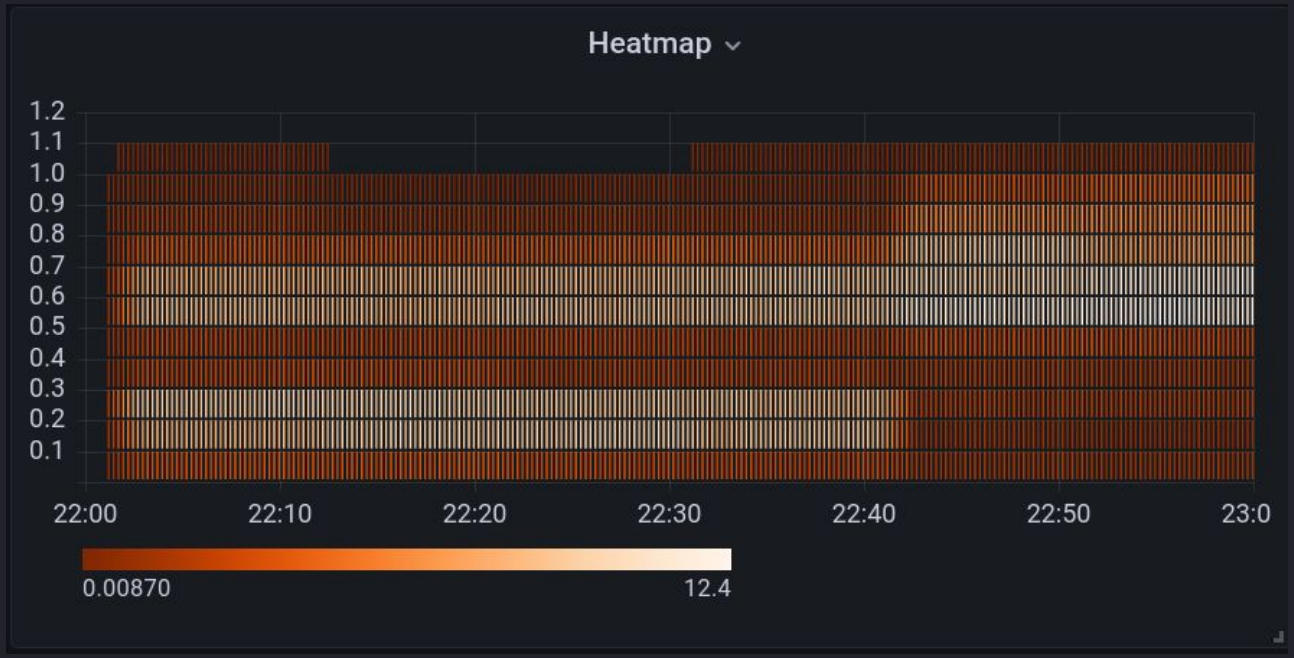
Histograms

rate(http_request_duration_seconds_bucket**[5m])**

Histograms

histogram_quantile(0.95, rate(http_request_duration_seconds_bucket[5m]))**)**

Histograms



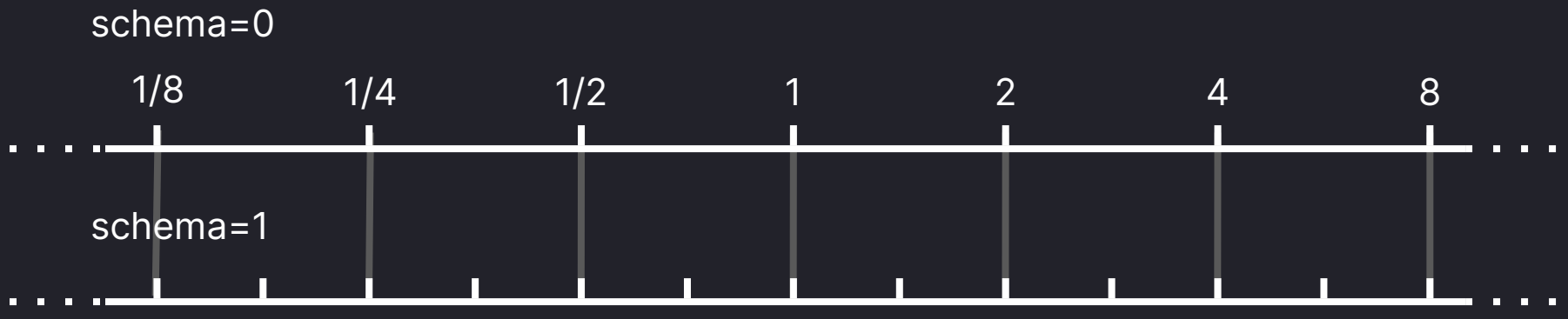
rate(http_request_duration_seconds_bucket**[5m])**

Histograms

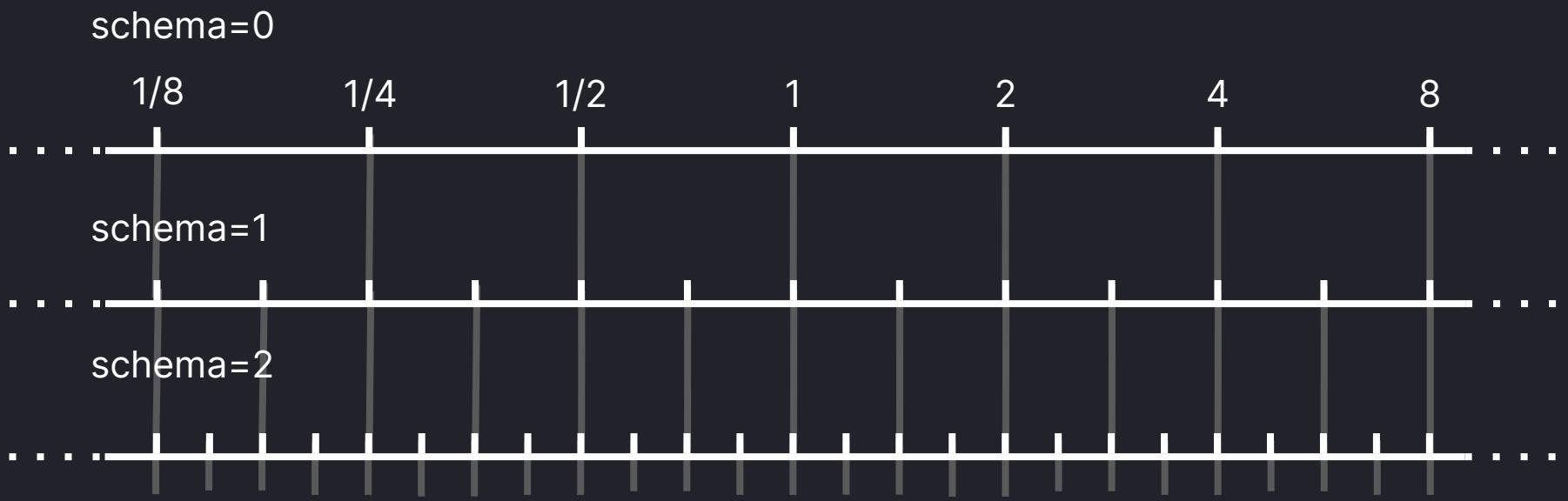
- Time Interval
  - Ad-hoc at query time, like [5m]
- Aggregation
  - Yes
- Practical Value
  - Awesome, but …
  - … you have to define reasonable bucket boundaries.

Sparse / Native / Exponential Histograms

Sparse / Native / Exponential Histograms



Sparse / Native / Exponential Histograms

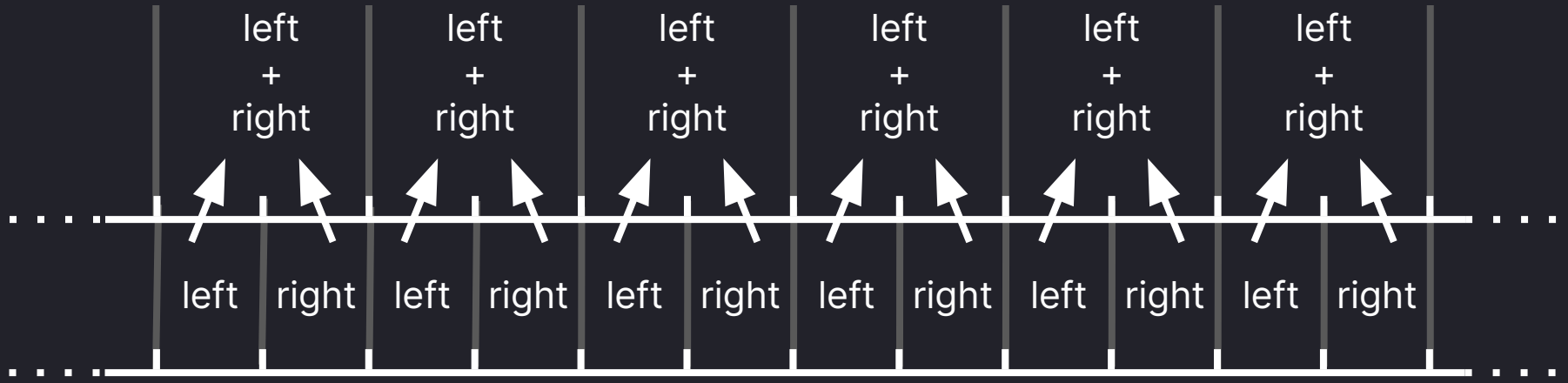


... up to schema=8

Sparse / Native / Exponential Histograms

schema	Max percentile error relative to the observed value (assuming harmonic mean)
0	33%
1	17%
2	9%
3	4%
4	2%
5	1%
6	0.5%
7	0.3%
8	0.1%

Sparse / Native / Exponential Histograms



Configure with max number of buckets:
→ If max number is reached, schema is reduced, two buckets collapse into one.

Sparse / Native / Exponential Histograms

- Time Interval
  - Ad-hoc at query time, like [5m]
- Aggregation
  - Yes
- Practical Value
  - No need for explicit bucket boundaries.
  - Can derive percentiles with guaranteed error limit.
  - It’s new. There’s little practical experience yet.

Contents

Part 1 (demo)

- Response time vs service time
- Load test backs off when server is at its limit

Part 2 (slides)

- Average & Maximum
- Percentiles
- Histograms
- More Histograms

Reference: Java Metric Libraries

- Dropwizard Metrics

- Micrometer (Spring Boot, Quarkus, ...)

- Prometheus Java client library

- OpenTelemetry Java SDK

- Microprofile Metrics 5.0

Dropwizard Metrics

- Average
  - Pre-calculated averages, no separate sum and count.
- Maximum
  - Yes, as part of Histogram
- Percentiles
  - Yes, as part of Histogram
- Histograms
  - No. There is a data type named histogram, but it exposes only percentiles, no histogram buckets.
- Sparse / Native / Exponential Histograms
  - No

Micrometer

- Average
  - Yes, using sum and count as shown in this presentation.
- Maximum
  - Yes
- Percentiles
  - Yes, using HdrHistograms internally.
- Histograms
  - Yes
- Sparse / Native / Exponential Histograms
  - No, but I assume Micrometer will add support if these histograms become popular in Prometheus

Prometheus Java Client Library

- Average
  - Yes, using sum and count (Summary in Prometheus terminology)
- Maximum
  - Yes, as the 100th percentile (of a Summary metric)
- Percentiles
  - Yes, using the CKMS algorithm described in this talk
- Histograms
  - Yes
- Sparse / Native / Exponential Histograms
  - Yes, currently in the sparsehistogram feature branch.

OpenTelemetry Metrics Java SDK

- Average
  - Yes, count and sum are part of Histogram
- Maximum
  - Yes, part of Histogram
- Percentiles
  - No (?)
- Histograms
  - Yes
- Sparse / Native / Exponential Histograms
  - Yes, but exposition in Prometheus format still a TODO.

Microprofile Metrics 5.0

- Average
  - Yes, count and sum are part of timers
- Maximum
  - Yes, part of timers
- Percentiles
  - Yes
- Histograms
  - Not in 5.0, planned for 5.1
- Sparse / Native / Exponential Histograms
  - No