Prometheus

a practical workshop v1.1

Tomer Gabel, November 2024

The Pillars of Observability

1. Logs
2. Metrics
3. Traces

The Pillars of Observability

1. ~~Logs~~
2. Metrics
3. ~~Traces~~

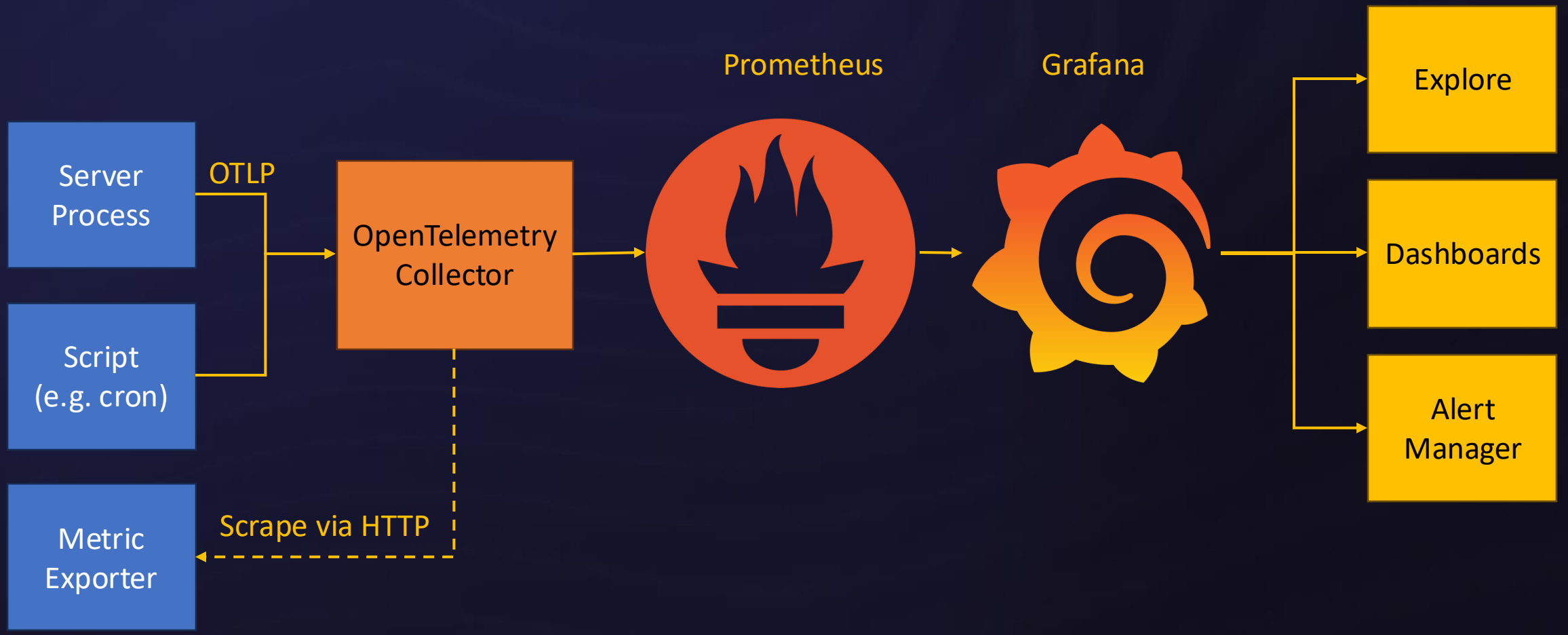
Our focus today.

Metrics:

Answer *quantitative* questions

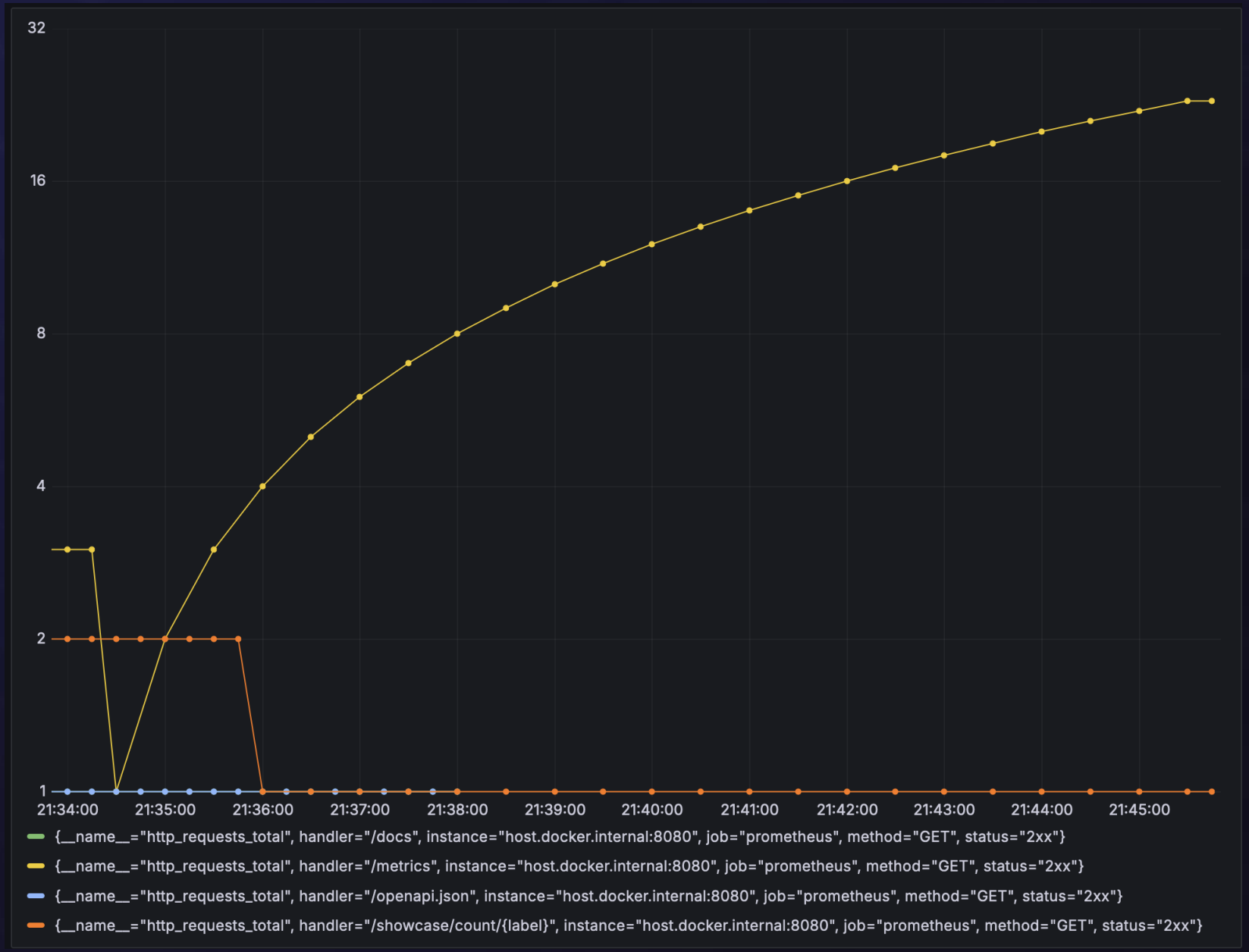
1. *How many* requests did I get?
2. *How long* did they take to process?
3. *How much* memory am I using?
4. *How much* free space is left on the disk?

Enter: Prometheus



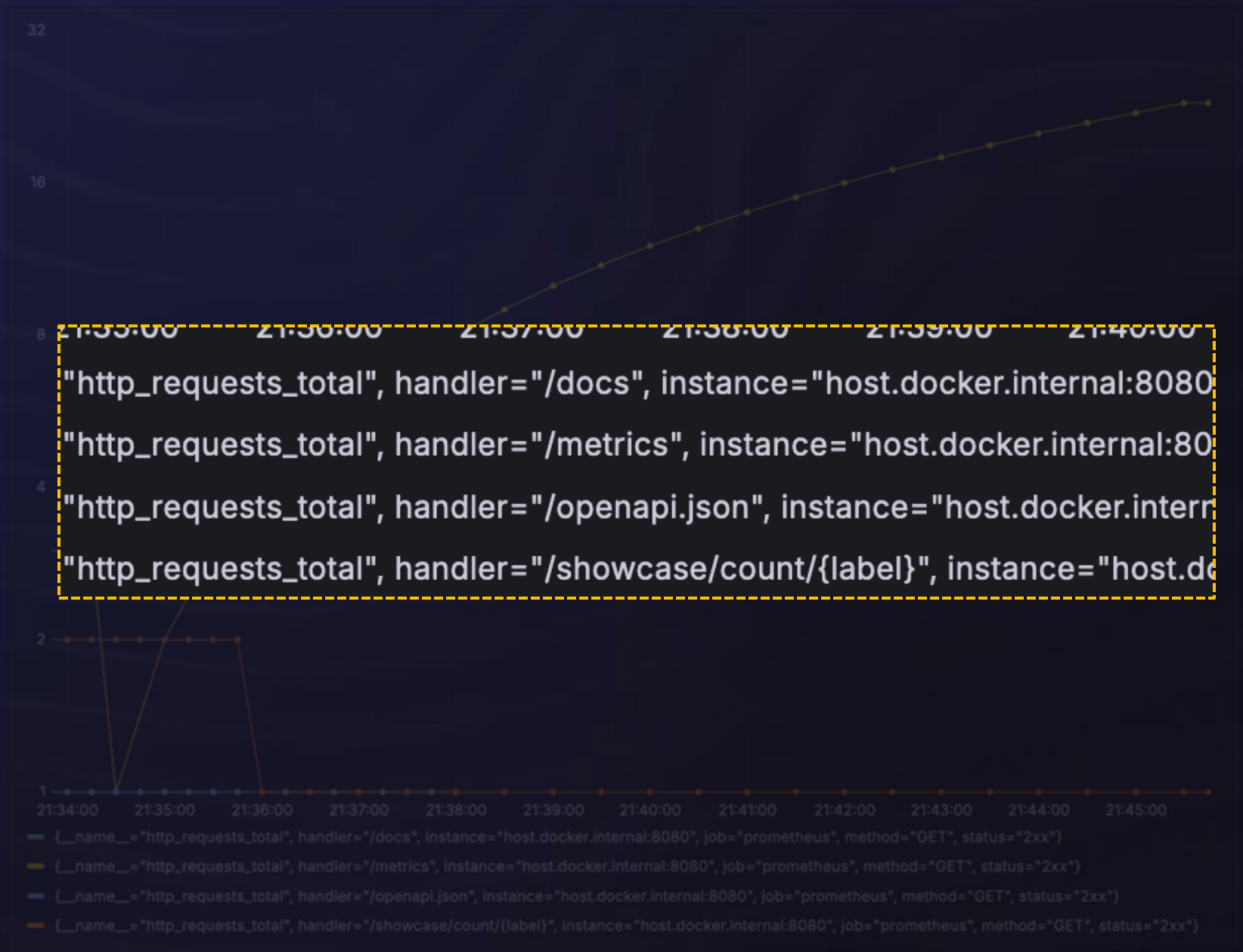
Metrics 101

1. Name
2. Labels
3. Time
4. Value



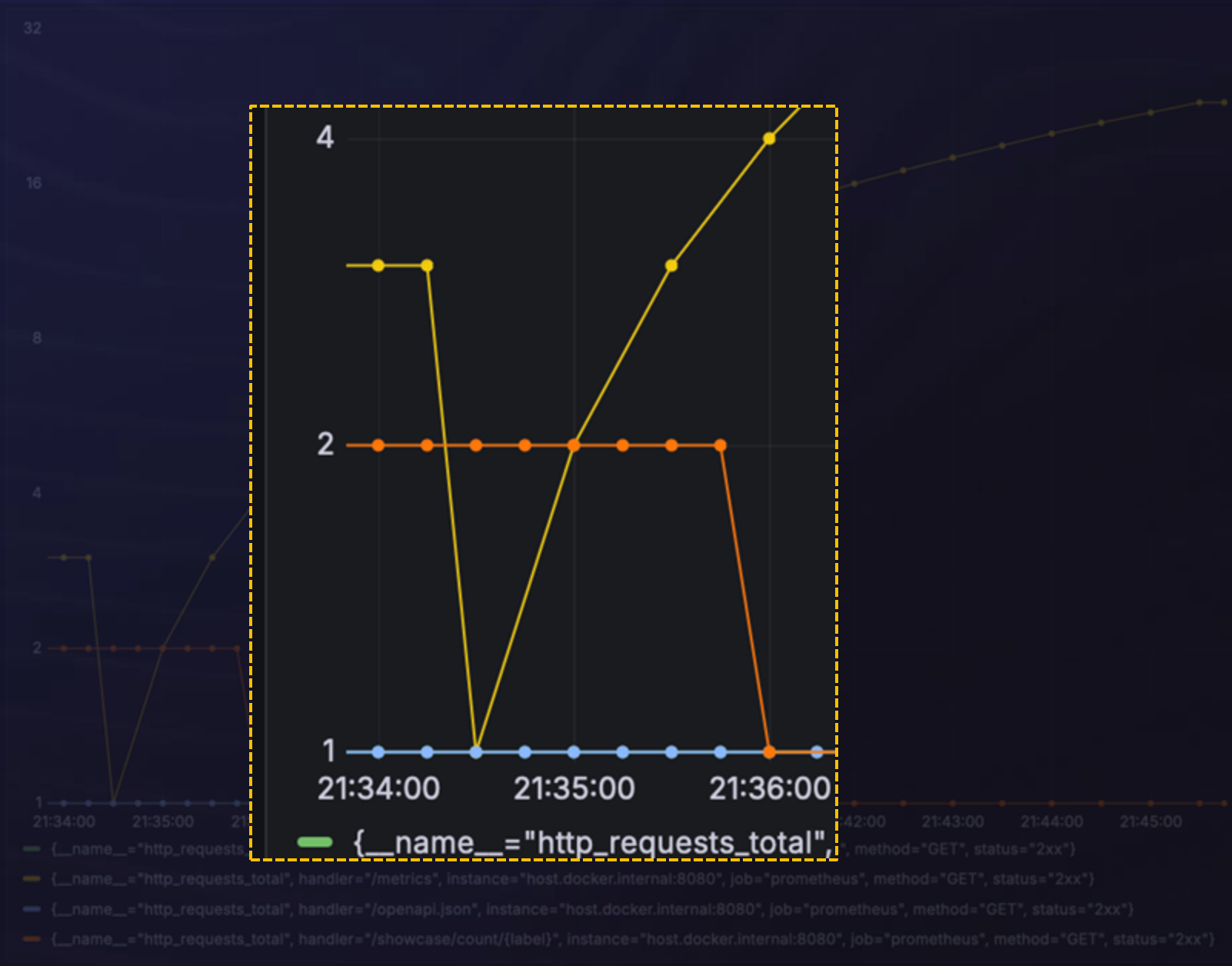
Metrics 101

1. Name
2. Labels
3. Time
4. Value



Metrics 101

1. Name
2. Labels
3. Time
4. Value



A word on cardinality

1. Prometheus is analogous to a *flat database*
   - Partitioned by *time*
   - Labels are like *indexed columns*
2. The DB size is therefore:
   - Timeframe * # labels * # *label values*

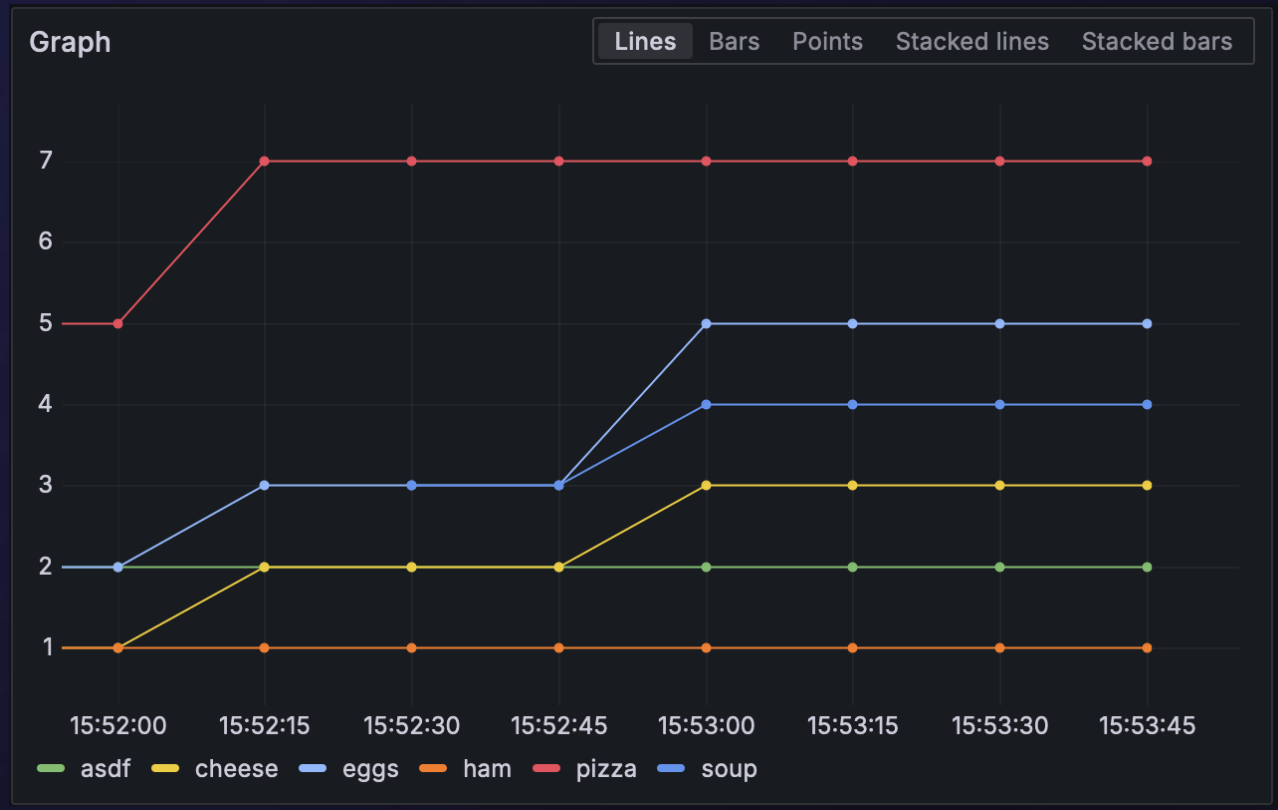
Prometheus metric types: Counter

1. As the name implies, a metric that *counts*
2. Can go up or down by any amount
3. *Stateful* – value managed by *Prometheus*
4. Examples:
   - # of server requests (by method, path, …)
   - # of events (“times a user logged in”)

Counting the Python way

```
counter = meter.create_counter(
    "my_count",
    description="Event count"
)

@router.get("/count/{label}")
def count(label: str) -> dict:
    counter.add(1, {"my_label": label})
    return {"status": "ok"}
```



Prometheus metric types: Gauge

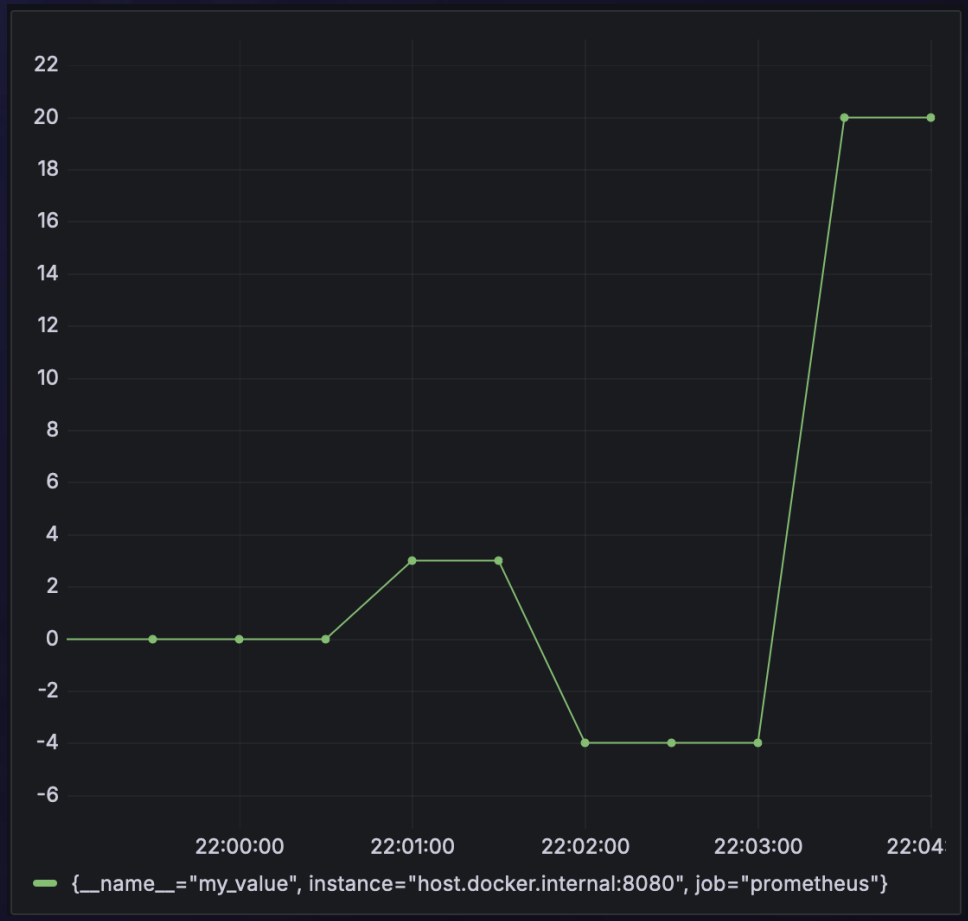
1. A metric that is sampled *on demand*
2. Each sample is *independent*
3. Common examples:
   - CPU core temperature in ℃
   - Free space on /dev/sda2

Gauging the Python way

```
gauge = meter.create_gauge(
    "my_value",
    description="Some stateful value"
)

class GaugeData(BaseModel):
    value: int

@router.put("/gauge")
def gauge_value(data: GaugeData) -> dict:
    gauge.set(data.value)
    return {"status": "ok"}
```



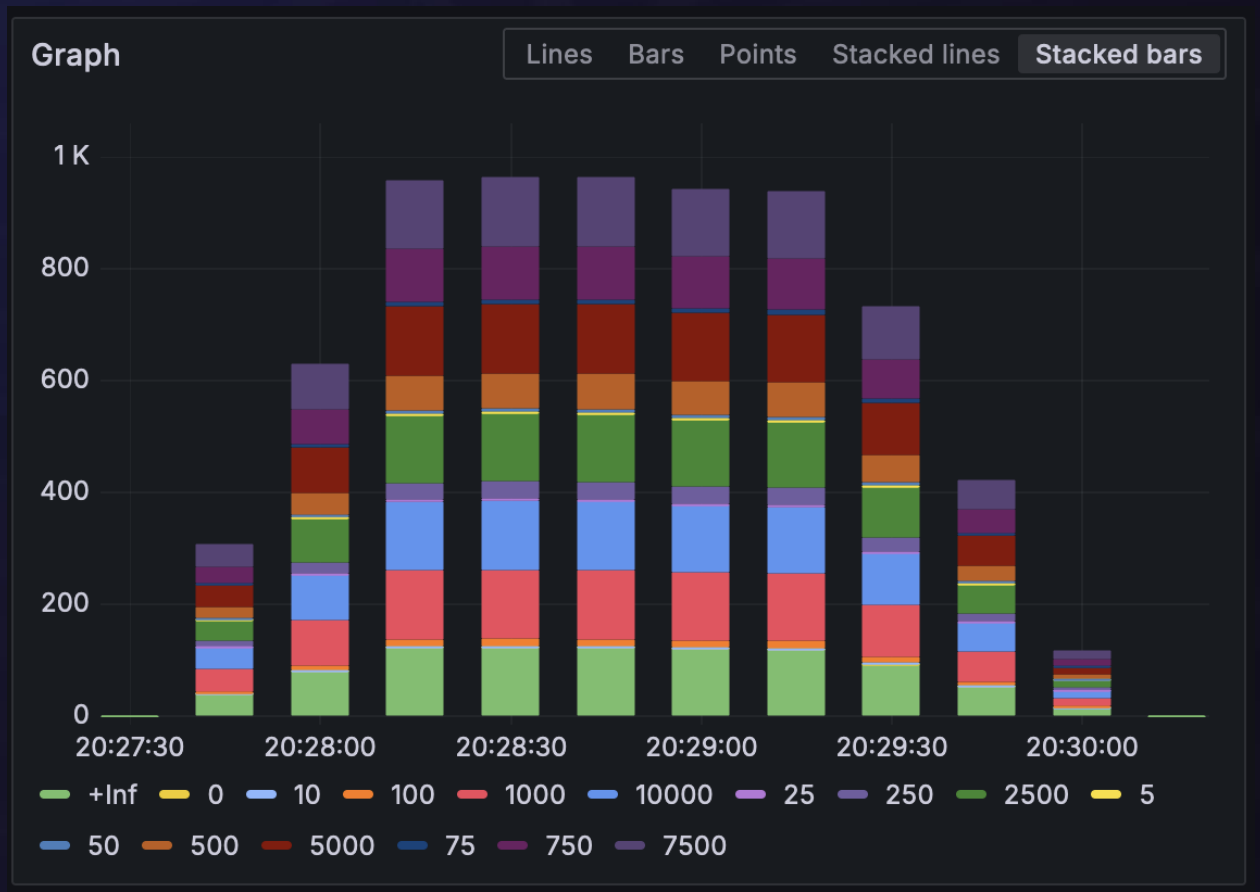
Prometheus metric types: Histograms

1. Some metrics *cannot be represented* with one value
   - "*How long* do requests to /login take?"
   - Do you mean average? Mean? 90th percentile?
2. These values prdoduce a *distribution*
3. Managed with buckets. Latency, for example:
   - 0–10ms, 10–100ms, 100–1000ms, …

Histograms the Python way

```
histogram = meter.create_histogram(
    "my_duration",
    unit="ms",
    description="Event duration"
)


@router.get("/duration/{length_ms}")
def duration(length_ms: int) -> dict:
    histogram.record(length_ms)
    return {"status": "ok"}
```



Not all histograms are made equal

Latencies vary wildly (ms vs second), as do *sizes* (payload in KB, disk in GB).

Views to the rescue!

```
duration_view = View(
    instrument_name="my_duration",
    aggregation=ExplicitBucketHistogramAggregation(range(0, 10000, 500)),
)
meter_provider = MeterProvider(
    metric_readers=[oltp_exporter],
    views=[duration_view],
)
set_meter_provider(meter_provider)
```

Lab Time

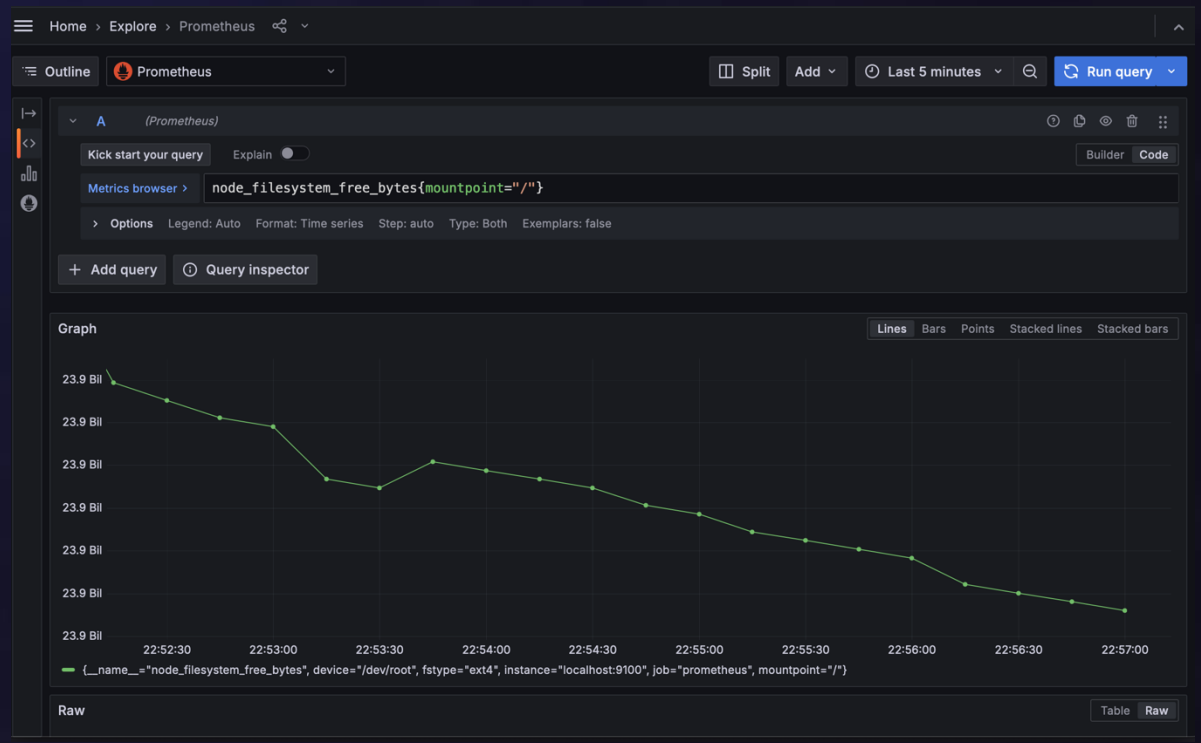
Showcase

1. F*amiliarize yourself* with the lab setup
2. *Get handsy* with Grafana
3. Open lab-showcase from the *class materials*
4. You have *30 minutes* to explore!

Part 2: PromQL

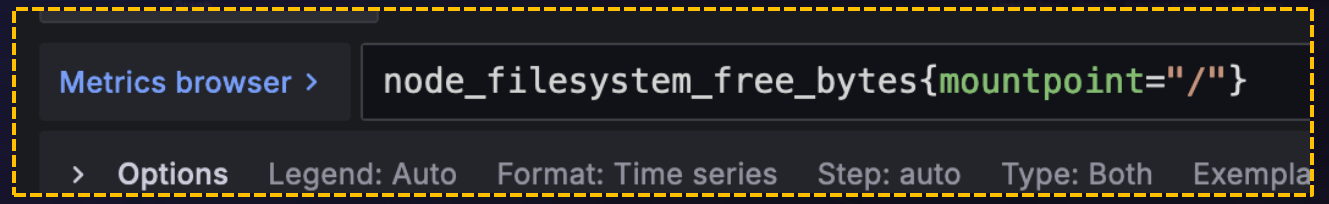
PromQL: Fear, Terror and Ruthless Efficiency

1. Extensive, powerful
2. … not entirely trivial



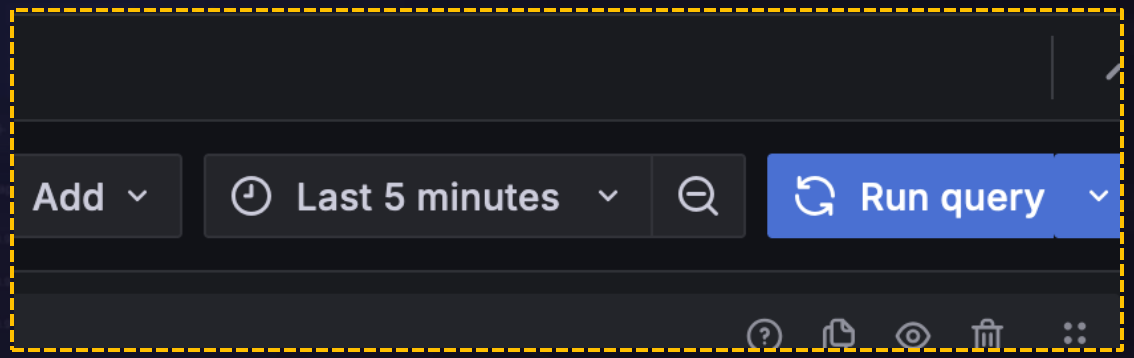
PromQL: Fear, Terror and Ruthless Efficiency

1. Extensive, powerful
2. … not entirely trivial
3. Basically:
   - A *query*
   - A *timeframe*



PromQL: Fear, Terror and Ruthless Efficiency

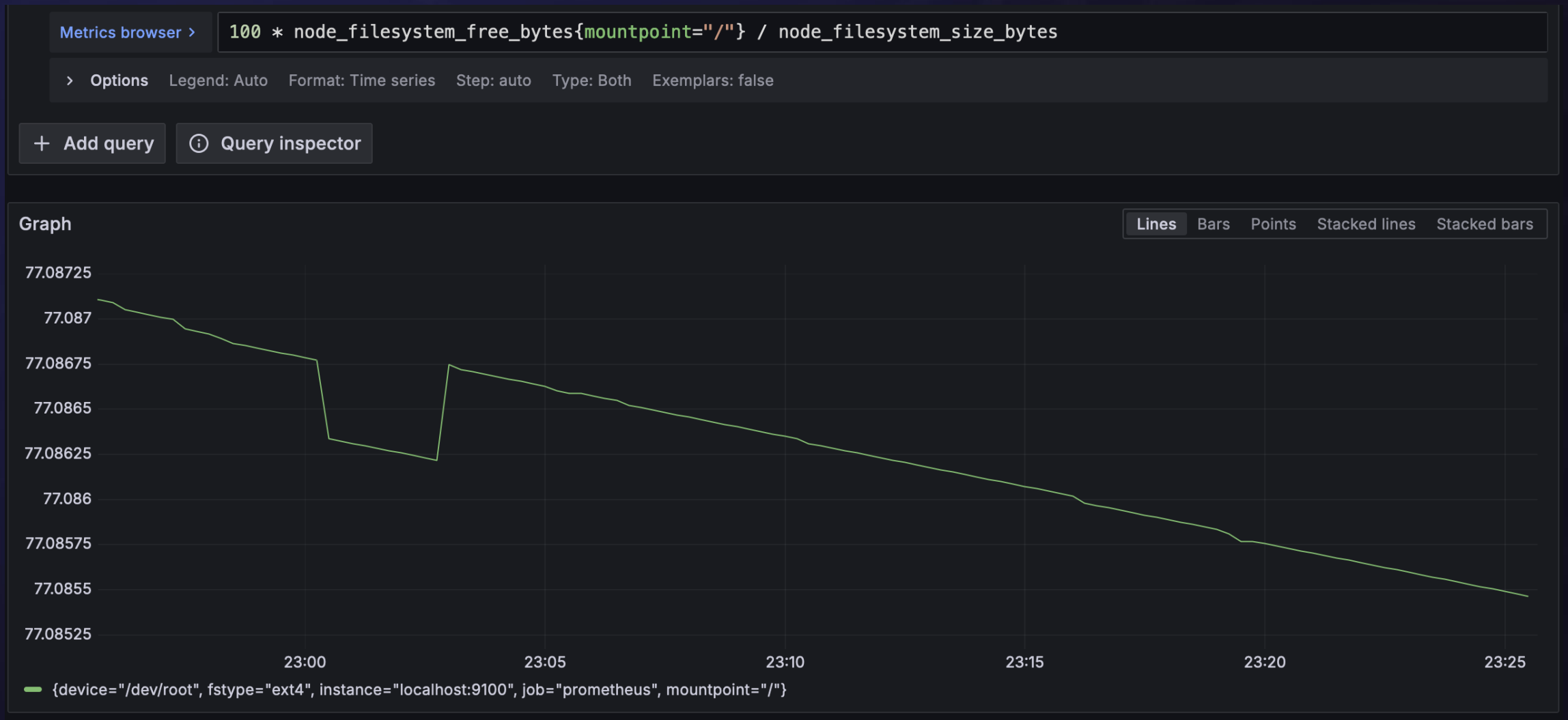
1. Extensive, powerful
2. … not entirely trivial
3. Basically:
   - A *query*
   - A *timeframe*



PromQL: Maths!

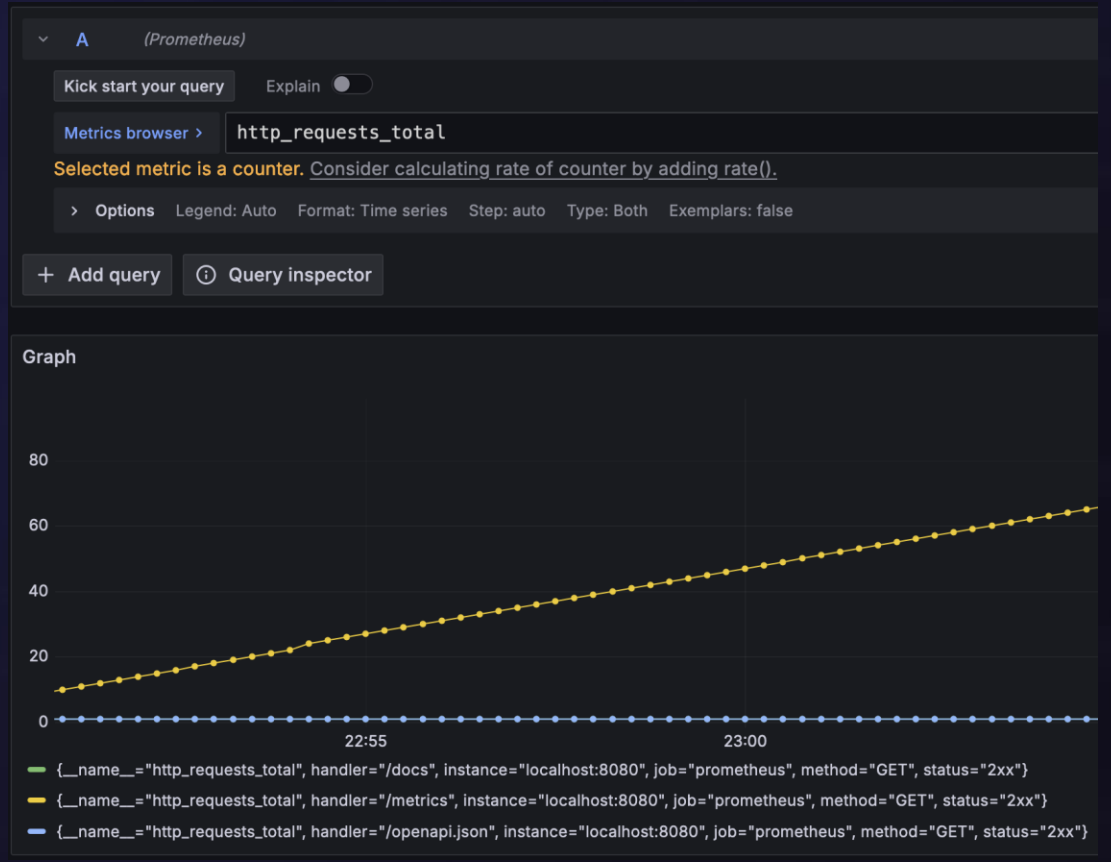
1. [Metric, labels, time] yields one value
2. Some observations require *multiple* values
3. Expressed with basic arithmetic:
   - % free disk space = *free* disk bytes/*total* disk bytes
   - % CPU = *CPU used seconds* / (# seconds * # cores)

PromQL: Maths



PromQL: Rates

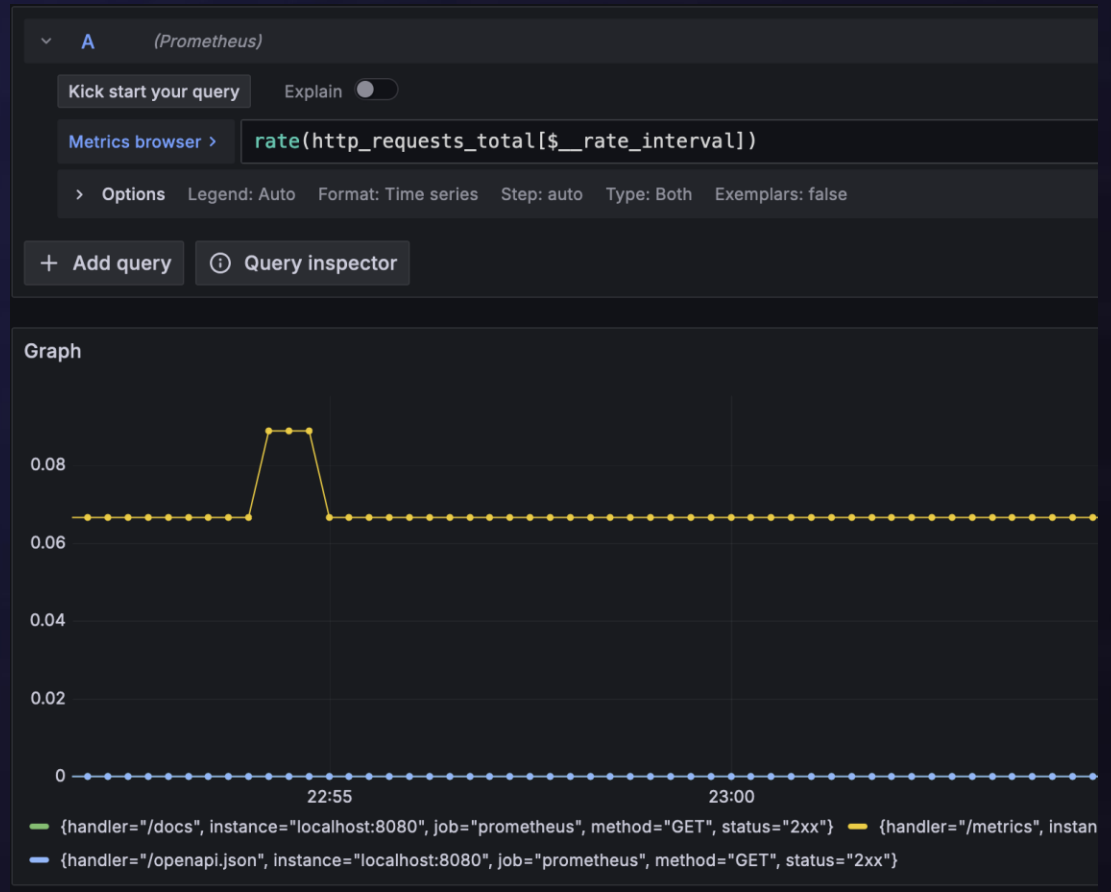
1. Counters are very useful
2. Can't use them *naïvely*



PromQL: Rates

1. Counters are very useful
2. Can't use them *naïvely*
3. We need a *rate derivation*
4. Magic incantation for the win:

   rate(metric{…}[$__rate_interval])



PromQL: Aggregations

1. Consider *http_server_duration_milliseconds_count*
   - A simple counter
   - Potentially many services, paths
2. Suppose you want the grand total?

   sum(http_server_duration_milliseconds_count)

PromQL: Aggregations

1. Consider *http_server_duration_milliseconds_count*
   - A simple counter
   - Potentially many services, paths
2. How about the 5 most used paths?

   topk(5, http_server_duration_milliseconds_count)

PromQL: Aggregations

1. What are the top 3 most time-consuming endpoints?

   topk(3, http_server_duration_milliseconds_sum)

2. How many processes run which Python version?

   count by (major) (python_info)

3. What is the highest rate of pagefaults recorded?

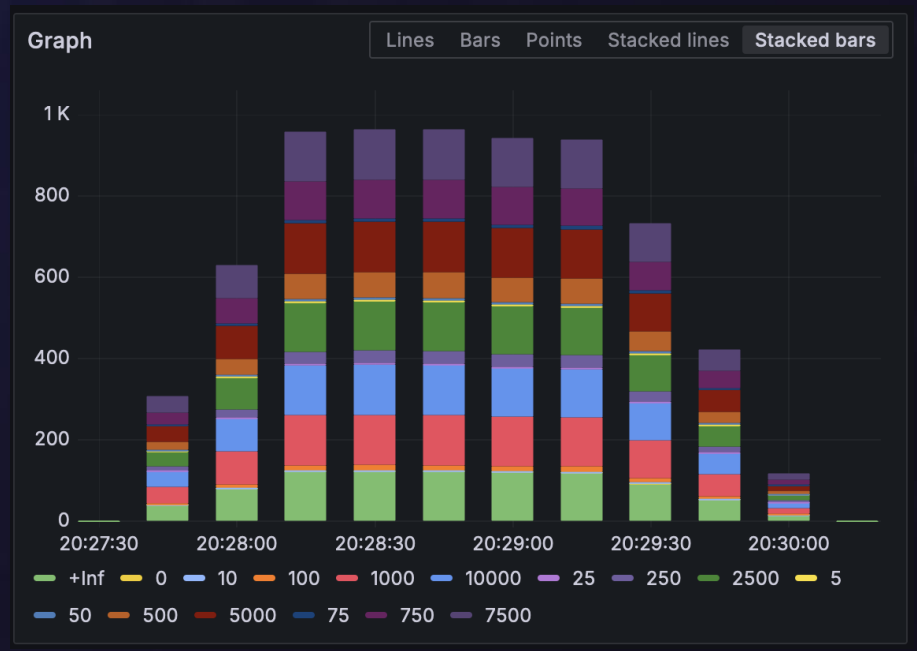
   max(rate(node_vmstat_pgfault[$__rate_interval]))

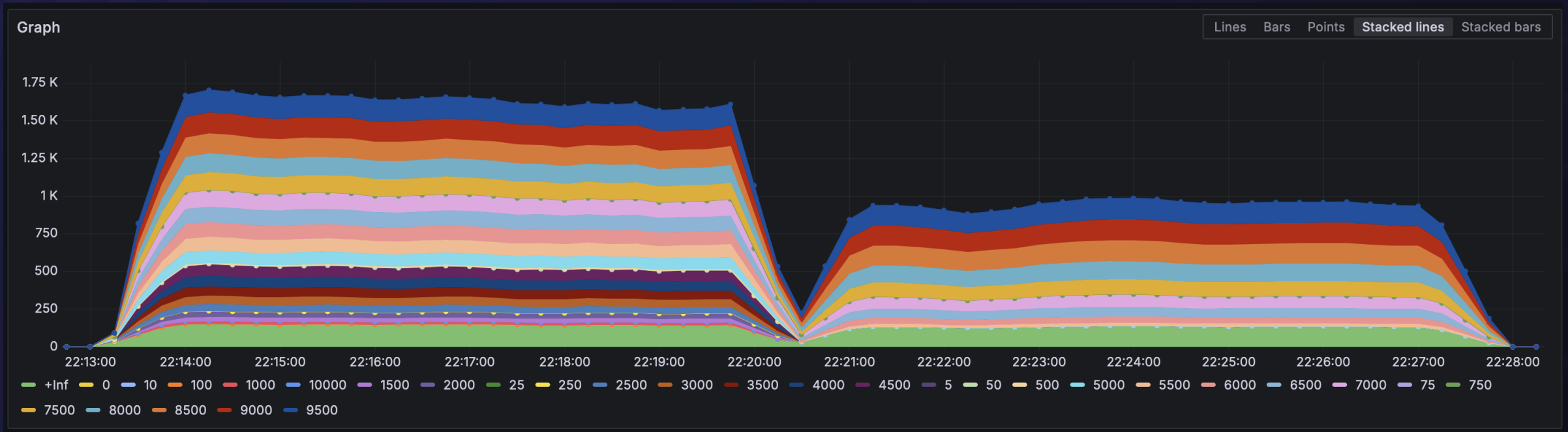
And now, what you've all been waiting for

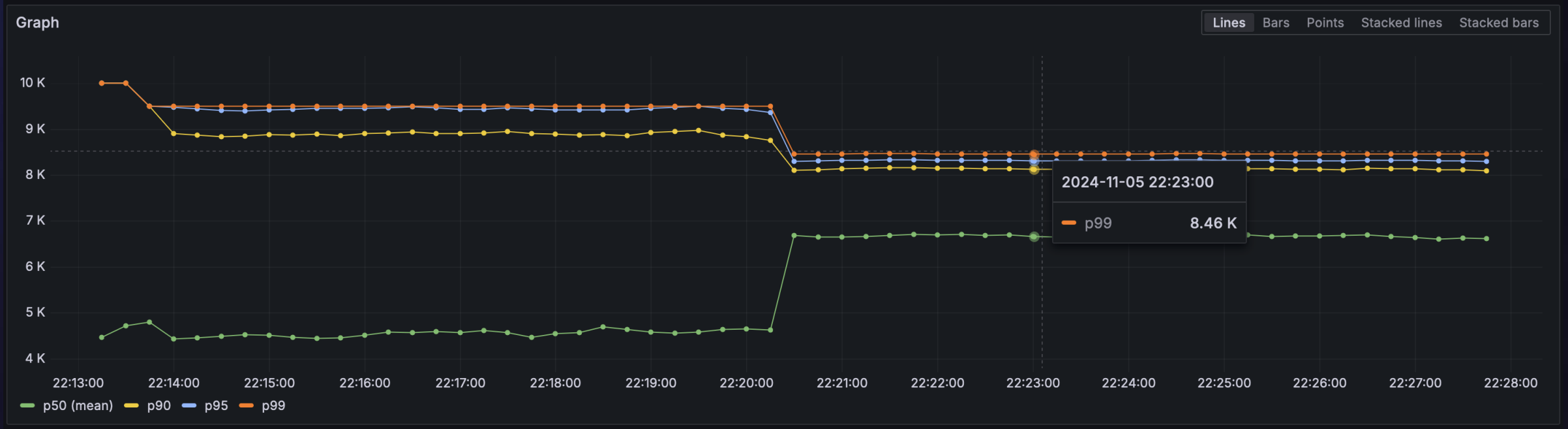
Quantiles (a.k.a. percentiles)

1. A histogram is a set of *counters*
   - 0-100ms, 100-200ms, 200-300ms…
2. What is the 95th *percentile*?
   - Linear regression FTW… but maths!
3. PromQL to the rescue:

   histogram_quantile(0.99, http_server_duration_milliseconds_bucket)




Graph
Lines
Bars
Points
Stacked lines
Stacked bars
1.75 K
1.50 K
1.25 K
1 K
750
500
250
0
22:13:00
22:14:00
22:15:00
22:16:00
22:17:00
22:18:00
22:19:00
22:20:00
22:21:00
22:22:00
22:23:00
22:24:00
22:25:00
22:26:00
22:27:00
22:28:00
+Inf
0
10
100
1000
10000
1500
2000
25
250
2500
3000
3500
4000
4500
5
50
500
5000
5500
6000
6500
7000
75
750
7500
8000
8500
9000
9500


Graph
Lines
Bars
Points
Stacked lines
Stacked bars
10 K
9 K
8 K
7 K
6 K
5 K
4 K
2024-11-05 22:23:00
p99
8.46 K
22:13:00
22:14:00
22:15:00
22:16:00
22:17:00
22:18:00
22:19:00
22:20:00
22:21:00
22:22:00
22:23:00
22:24:00
22:25:00
22:26:00
22:27:00
22:28:00
p50 (mean)
p90
p95
p99

Key Takeaways

1. Instrument early, instrument often

   There's no such thing as "too much data"

2. Beware high *cardinality*

   Aggregate, don't transact (that's what logs/traces are for)

3. Learn ye *PromQL*

   That's where the real leverage is

Lab time, again!

tomer@substrate.co.il

@tomerg

https://github.com/holograph/prometheus-workshop-service-python

Thank you for your attention

Questions?