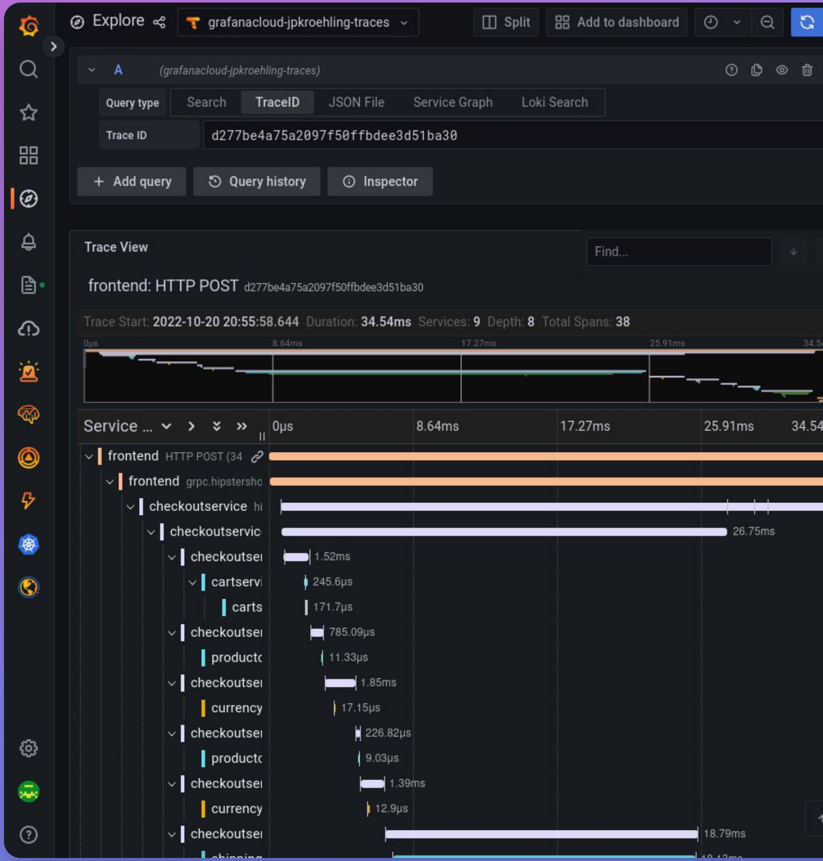
Grafana Labs

Demystifying Sampling

Juraci Paixão Kröhling
Software engineer
@jpkrohling



grafanacloud-jpkroehling-traces

A (grafanacloud-jpkroehling-traces)

Query type: Search | TraceID | JSON File | Service Graph | Loki Search

Trace ID: d277be4a75a2097f50ffbdee3d51ba30

frontend: HTTP POST d277be4a75a2097f50ffbdee3d51ba30

Trace Start: 2022-10-20 20:55:58.644 Duration: 34.54ms Services: 9 Depth: 8 Total Spans: 38

```
frontend HTTP POST (34
  frontend grpc.hipstersho
    checkoutservice hi
      checkoutservic            26.75ms
        checkoutsei             1.52ms
          cartservi             245.6μs
            carts               171.7μs
        checkoutsei             785.09μs
            productc            11.33μs
        checkoutsei             1.85ms
            currency            17.15μs
        checkoutsei             226.82μs
            productc            9.03μs
        checkoutsei             1.39ms
            currency            12.9μs
        checkoutsei             18.79ms
```

Speaker

Juraci Paixão Kröhling

Software Engineer

Agenda

1. Sampling?
2. Logs and sampling
3. Metrics and sampling
4. Traces and sampling
5. Bonus: Profiles and sampling
6. Questions and answers

Speaker

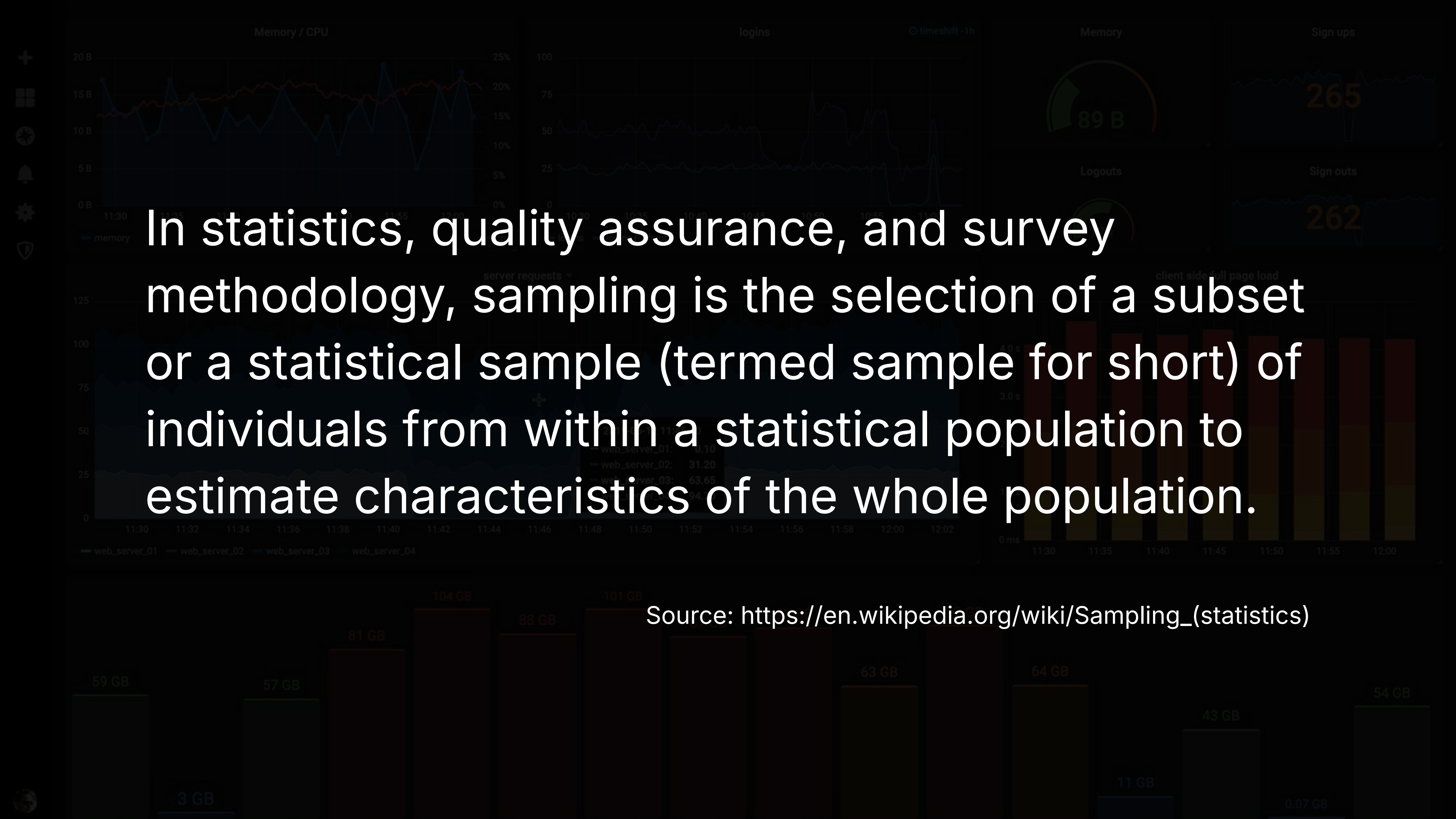
Juraci Paixão Kröhling

Software Engineer

About me

- Software engineer at Grafana Labs
- Governance Committee member for the OpenTelemetry project
- Cloud Native Computing Foundation (CNCF) Ambassador
- Maintainer of modules for OpenTelemetry Collector
- Jaeger emeritus maintainer
- OpenTracing emeritus maintainer

Sampling?



In statistics, quality assurance, and survey methodology, sampling is the selection of a subset or a statistical sample (termed sample for short) of individuals from within a statistical population to estimate characteristics of the whole population.

Source: https://en.wikipedia.org/wiki/Sampling_(statistics)

“

Sampling is the technique used to reduce the amount of telemetry data that is generated or stored, while retaining representativity.

Sampling and Logs

Sampling for logs

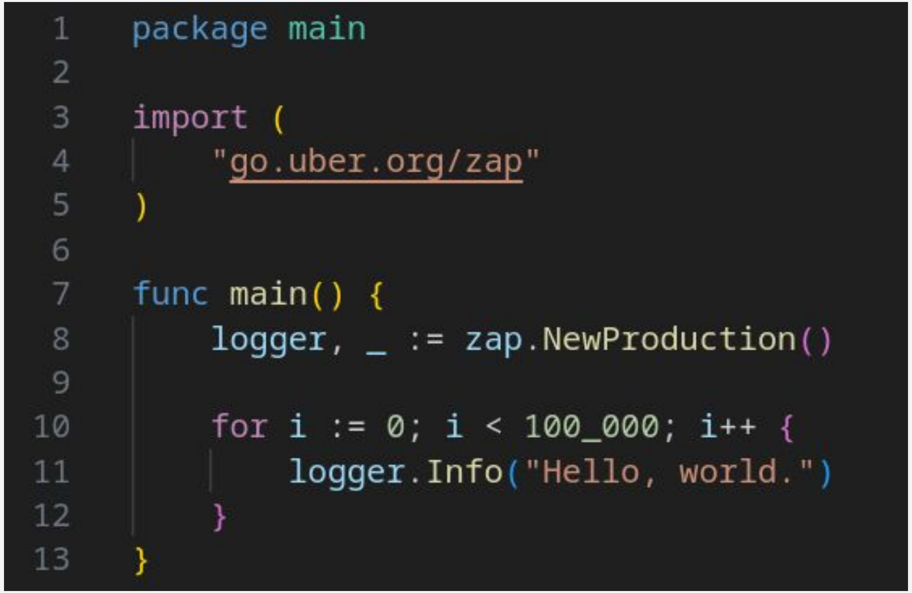
- Log levels!

Sampling for logs

- Log levels!
- Rate-limiting is the mostly used technique
- Usually done by the instrumentation library
- Prevents log flooding, especially under error conditions
    - Such as flooding the logs with stack traces that are currently happening for all incoming requests

Sampling for logs

- Go with Zap



```go
package main

import (
    "go.uber.org/zap"
)

func main() {
    logger, _ := zap.NewProduction()

    for i := 0; i < 100_000; i++ {
        logger.Info("Hello, world.")
    }
}
```

Sampling for logs

- Go with Zap

```
> go run . 2>&1 | wc -l
1099
```

Sampling for logs

- Java with Log4j

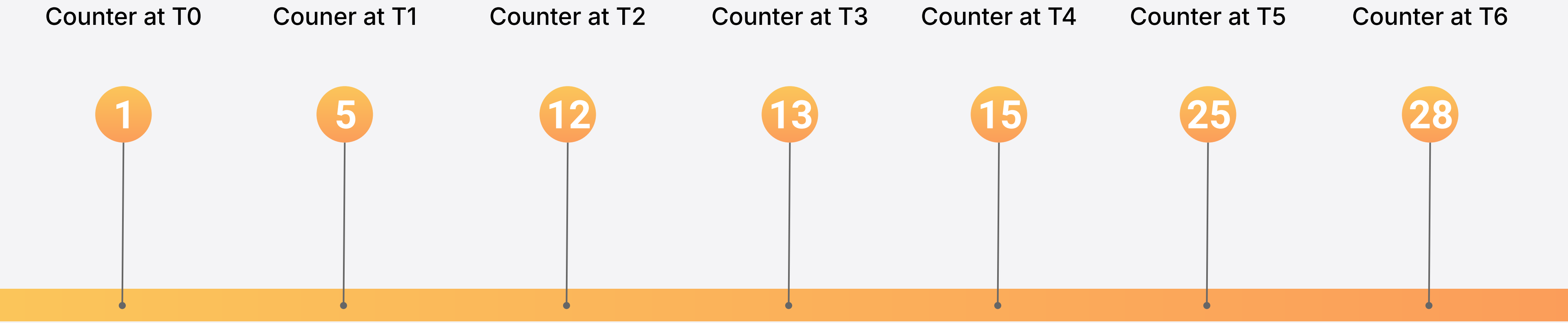
```
<BurstFilter level="INFO"
rate="16" maxBurst="100"/>
```

Sampling and Metrics

Sampling for metrics

- Downsampling
  - Typically on older data, to have longer retention in exchange for granularity
- Decreasing cardinality
  - Reduce the number of dimensions, have fewer time-series

Sampling for metrics - Downsampling



Counter at T0

1

Couner at T1

5

Counter at T2

12

Counter at T3

13

Counter at T4

15

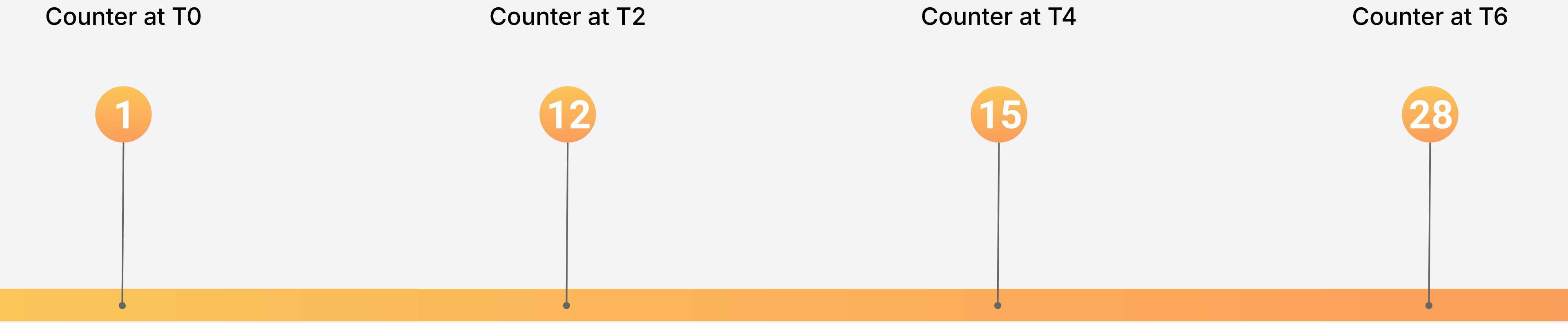
Counter at T5

25

Counter at T6

28

Sampling for metrics - Downsampling



Counter at T0

1

Counter at T2

12

Counter at T4

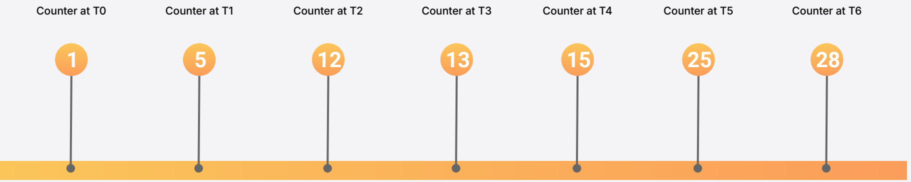
15

Counter at T6

28

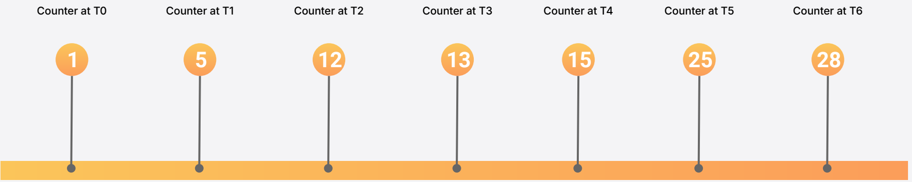
Sampling for metrics - Reduce cardinality

order_count{cluster="prod", userid=123}



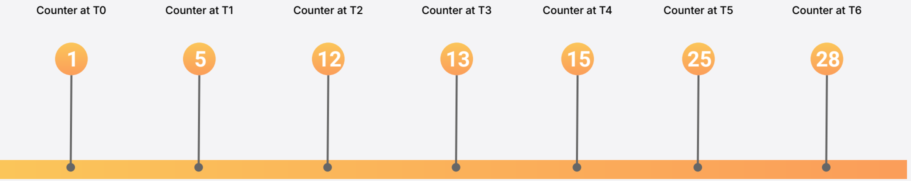
Counter at T0	Counter at T1	Counter at T2	Counter at T3	Counter at T4	Counter at T5	Counter at T6
1	5	12	13	15	25	28

order_count{cluster="prod", userid=456}



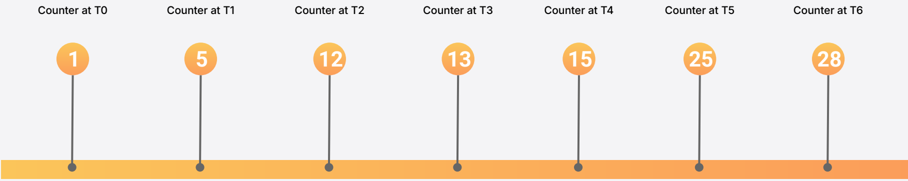
Counter at T0	Counter at T1	Counter at T2	Counter at T3	Counter at T4	Counter at T5	Counter at T6
1	5	12	13	15	25	28

order_count{cluster="test", userid=789}



Counter at T0	Counter at T1	Counter at T2	Counter at T3	Counter at T4	Counter at T5	Counter at T6
1	5	12	13	15	25	28

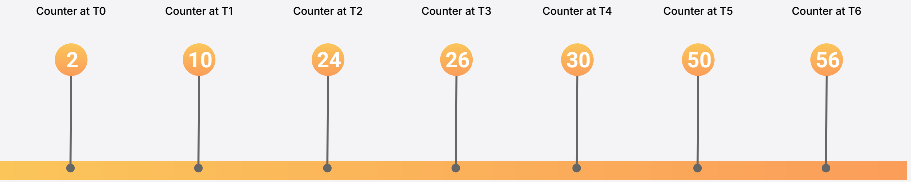
order_count{cluster="test", userid=321}



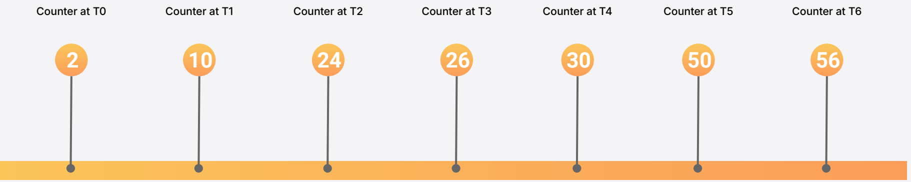
Counter at T0	Counter at T1	Counter at T2	Counter at T3	Counter at T4	Counter at T5	Counter at T6
1	5	12	13	15	25	28

Sampling for metrics - Reduce cardinality

order_count{cluster="prod"}



order_count{cluster="test"}



Sampling and Traces

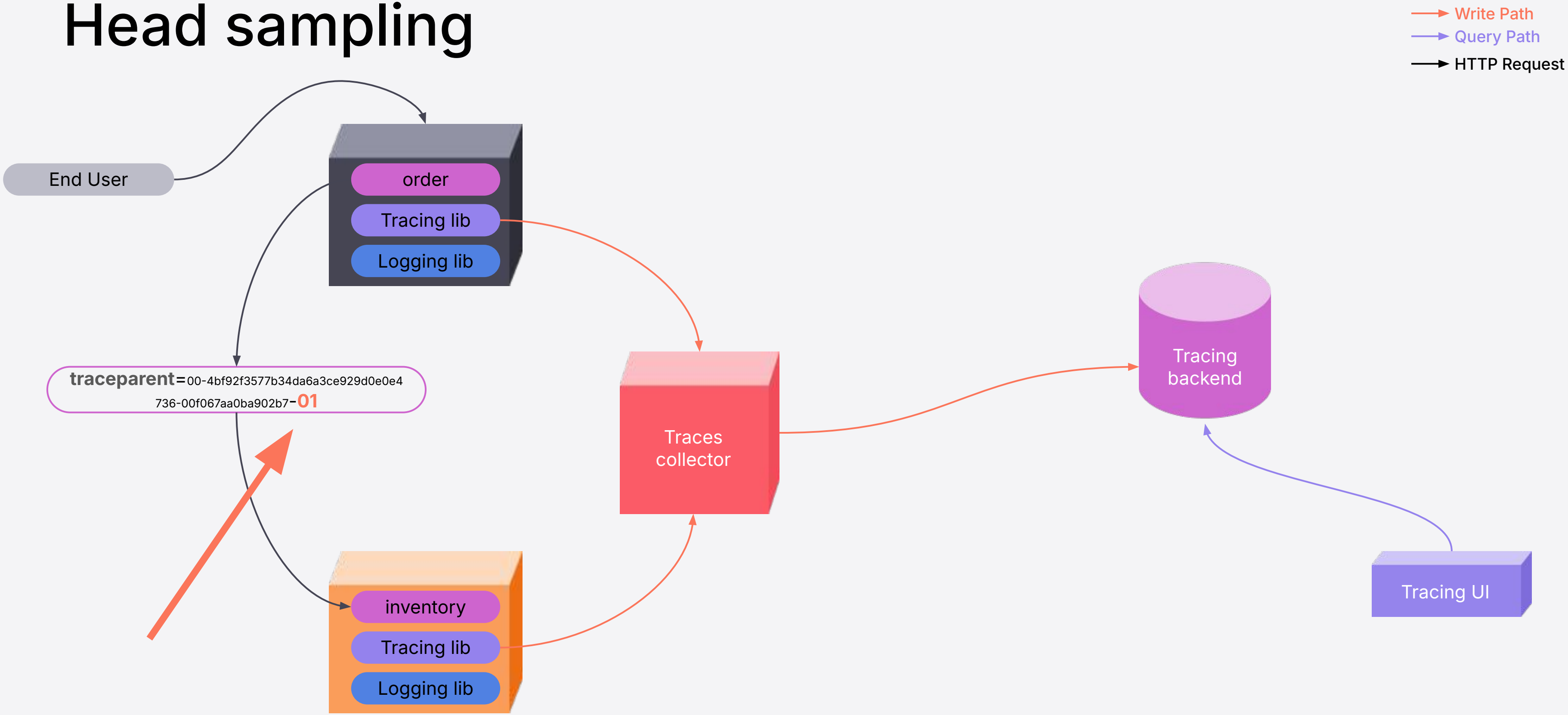
Sampling for traces

- Different sampling strategies
  - Head sampling, at the beginning of the transaction
  - Consistent sampling, based on characteristics shared by all spans in the same trace (like the traceID)
  - Tail sampling, once the transaction has been finished
  - Out-of-band sampling, after the data has been persisted
- Different sampling policies
  - Probabilistic
  - Rate-limiting
  - Based on attributes
  - Adaptive
  - Based on trace characteristics
    - Was there a span in error?
    - Did the trace take longer than a threshold?

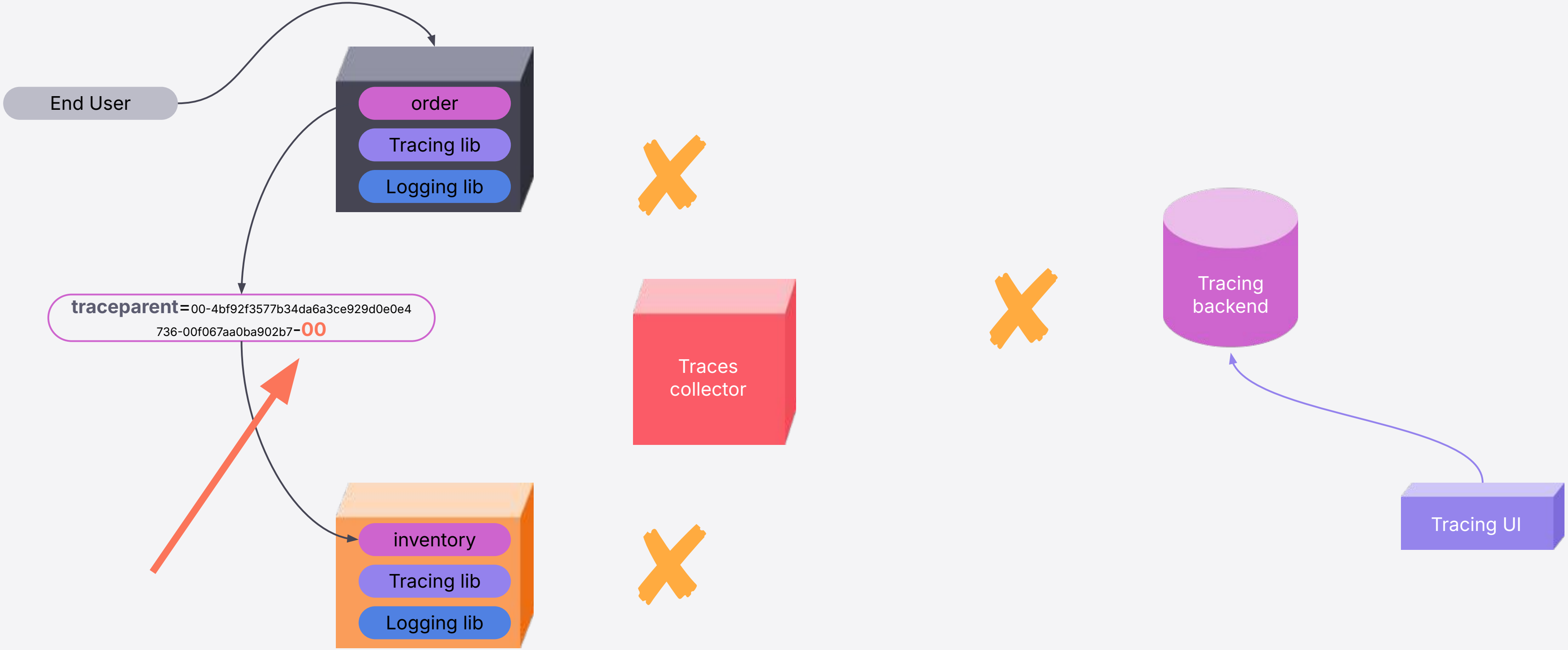
Head sampling

- Usually probabilistic
- Low network usage
- Low memory usage
- Simple to use
- Hard to do it consistently among services
- Hard to extract statistics out of it
  - Probability of 10% != 10%
  - Hard to know what was the probability at the time the span was created
  - How to extrapolate from highly different data?

Head sampling



Head sampling

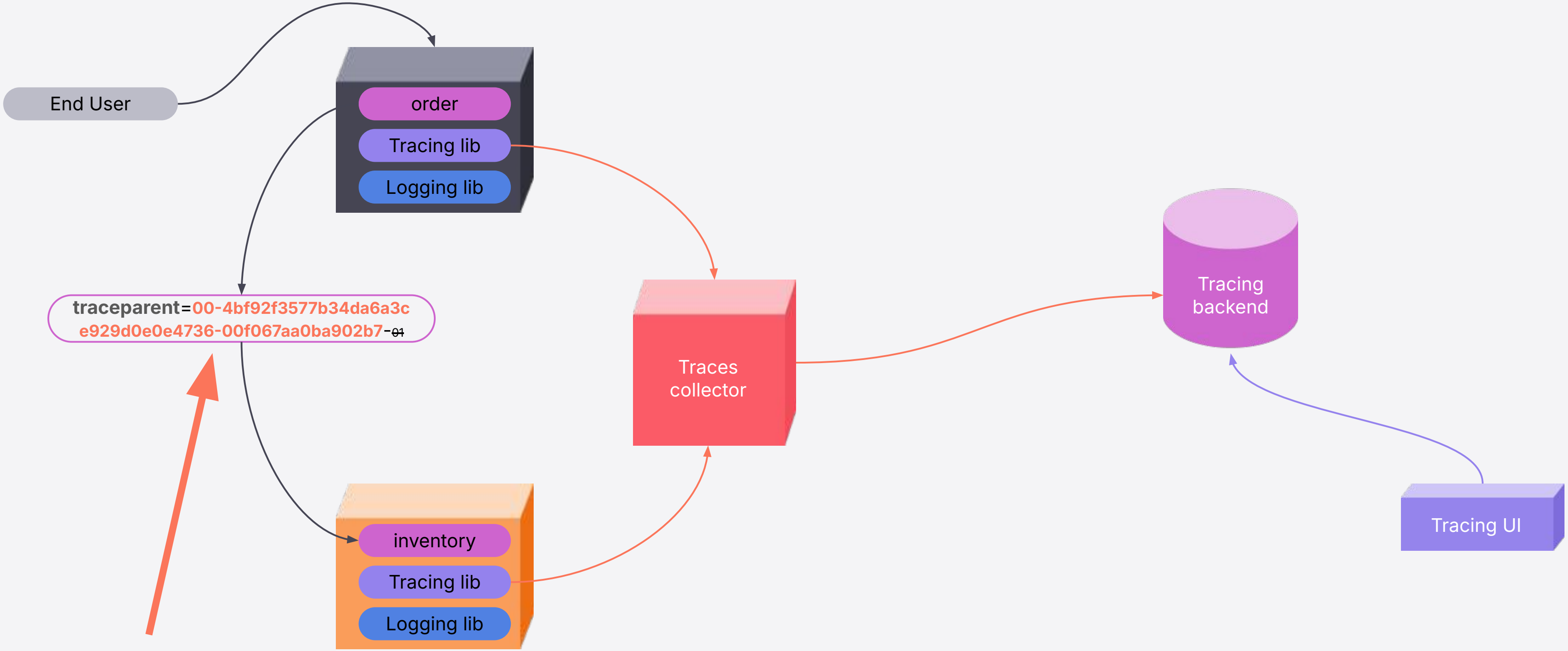


Consistent sampling

- Same decision for the spans on the same trace
    - Trace ID Ratio on OTel SDKs, for instance
    - Probabilistic sampling processor on OTel Collector, for instance
- Easier to do it consistently (one place to configure)
- More network usage
- Requires a central collection point
    - Comes with an ownership cost: engineering, computing, monitoring, ...
- Hard to extract statistics out of it

Consistent sampling SDK

Write Path
Query Path
HTTP Request



End User

order
Tracing lib
Logging lib

traceparent=00-4bf92f3577b34da6a3c
e929d0e0e4736-00f067aa0ba902b7-~~01~~

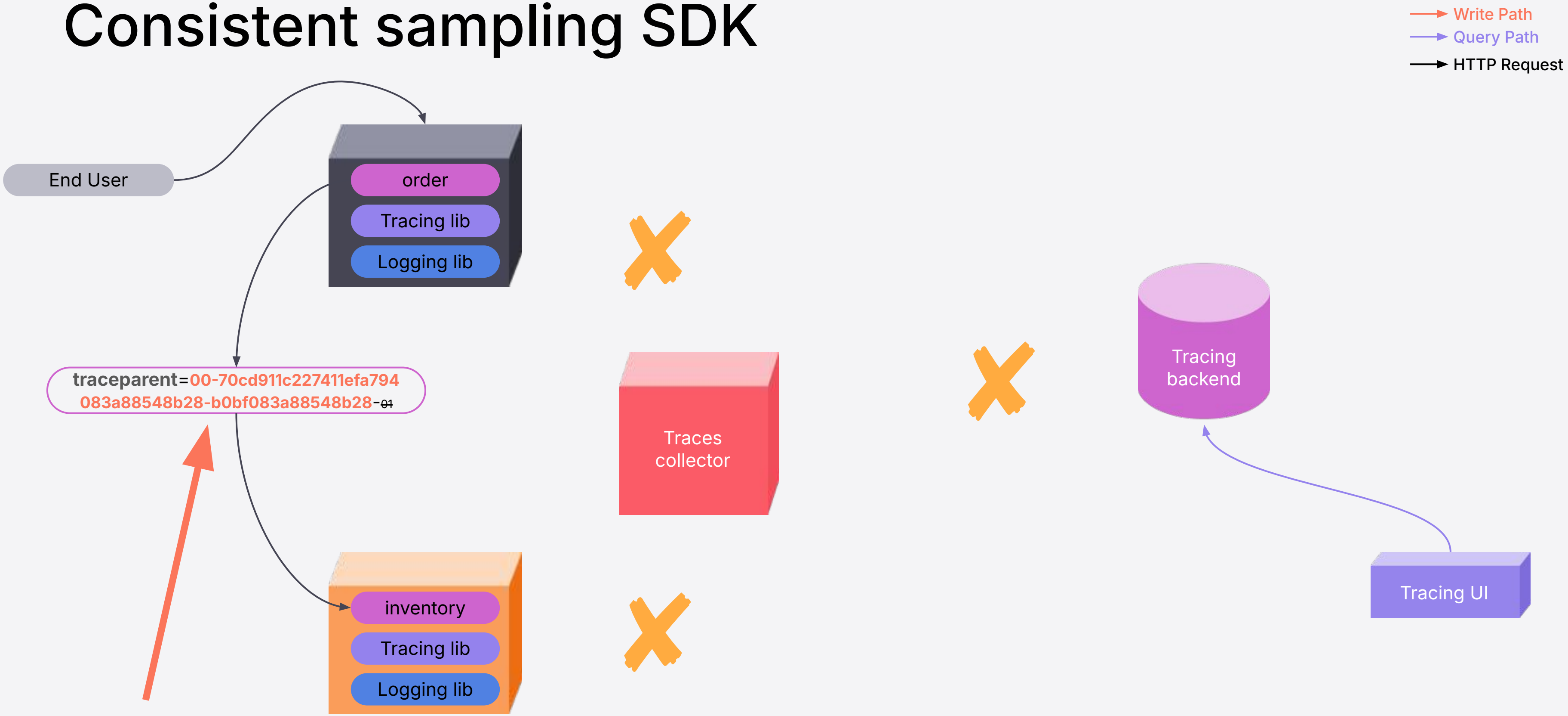
inventory
Tracing lib
Logging lib

Traces collector

Tracing backend

Tracing UI

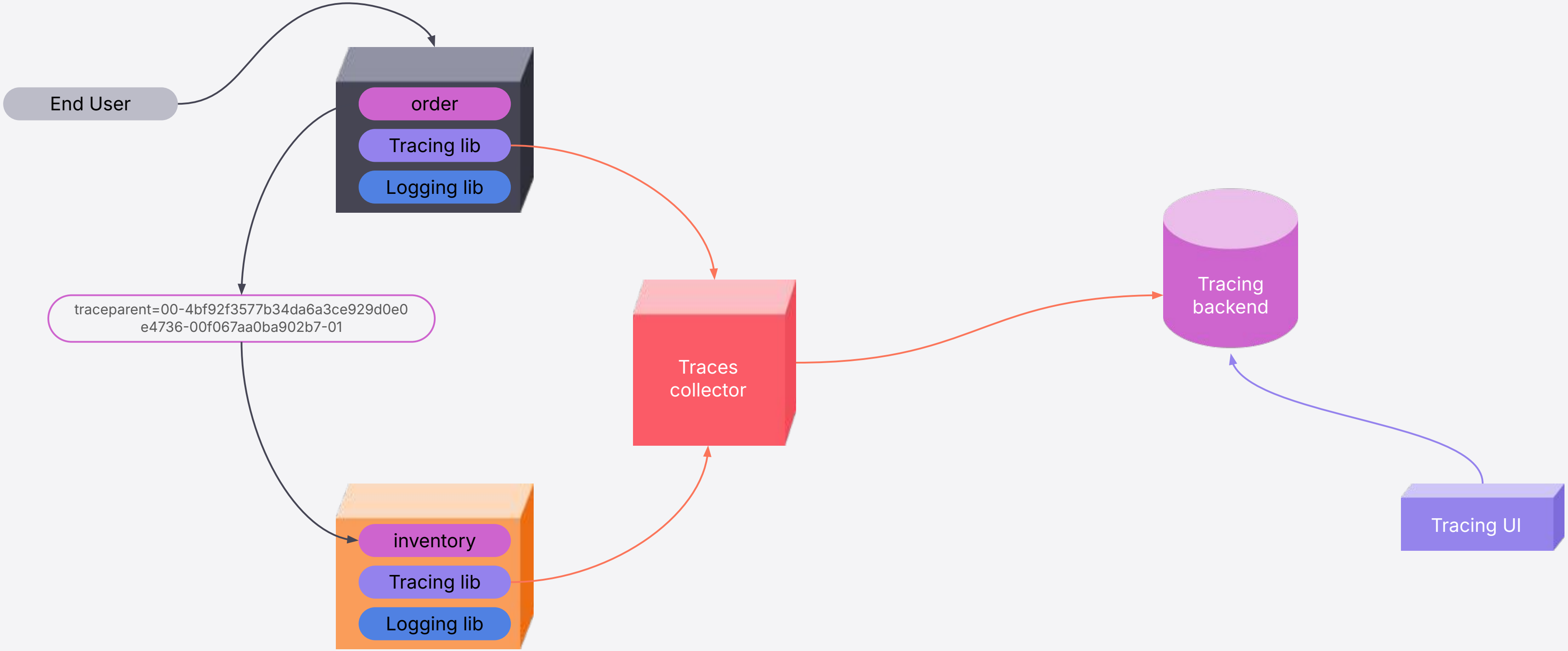
Consistent sampling SDK



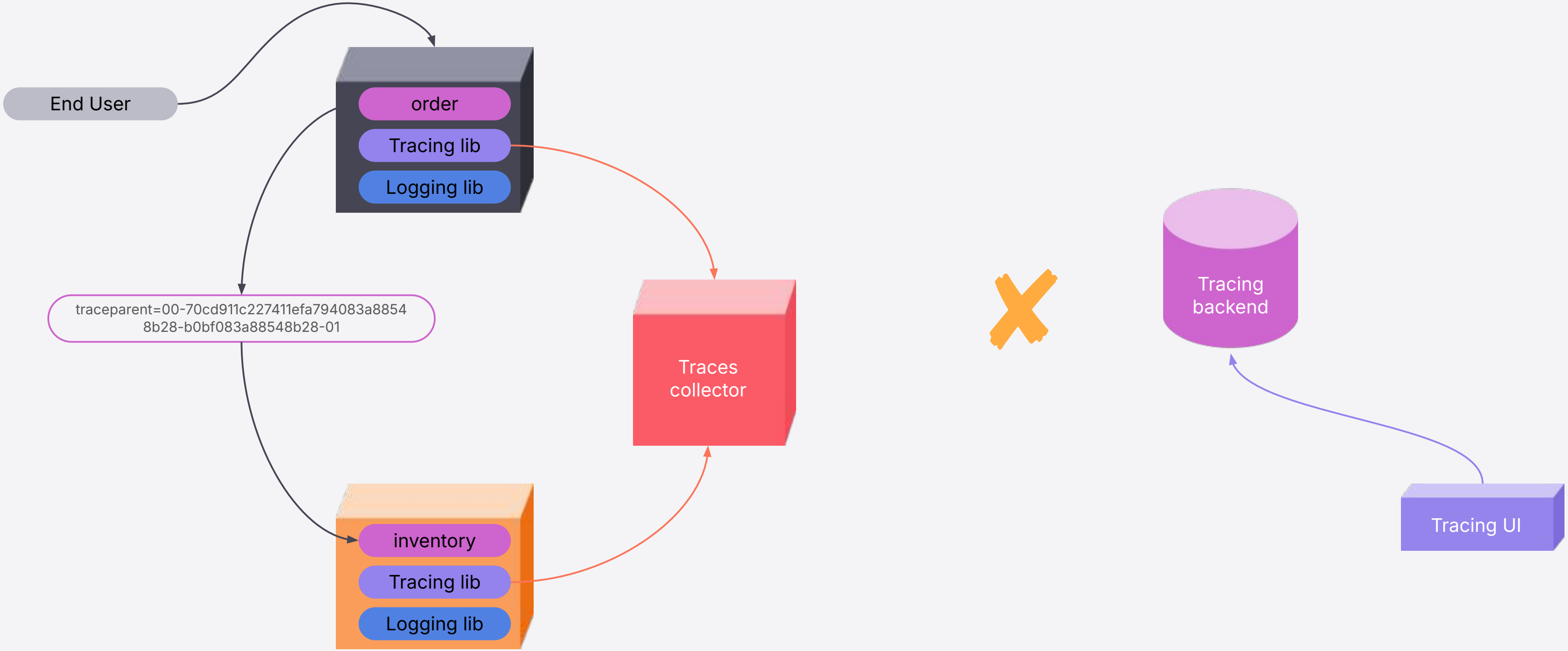
traceparent=00-70cd911c227411efa794
083a88548b28-b0bf083a88548b28-~~01~~

Consistent sampling - Collector

Write Path
Query Path
HTTP Request



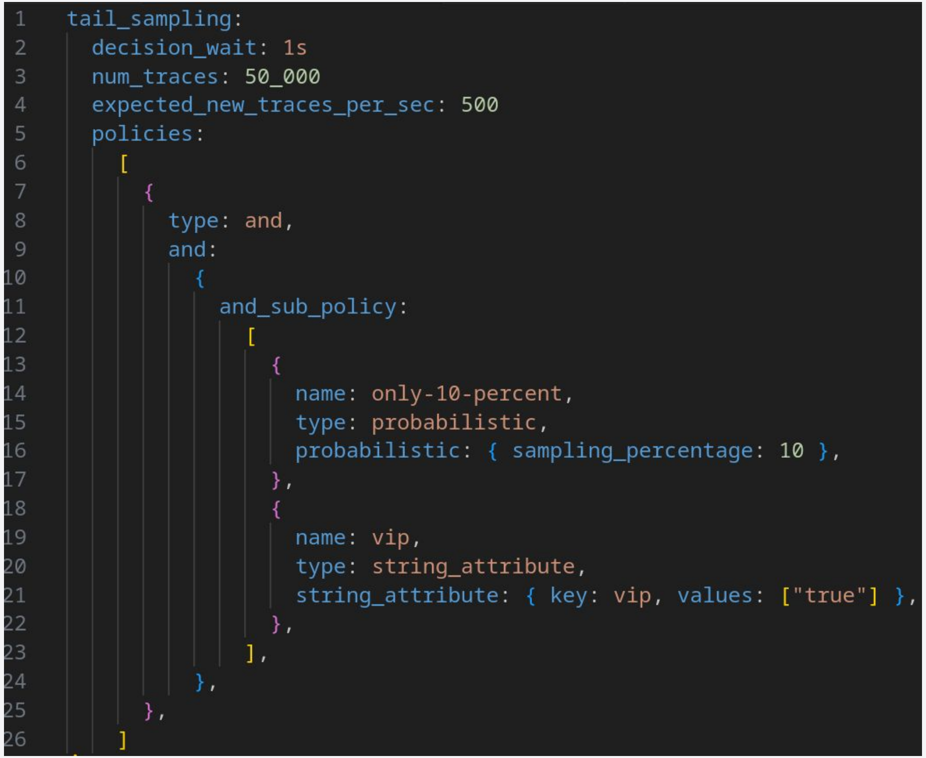
Consistent sampling - Collector



Tail sampling

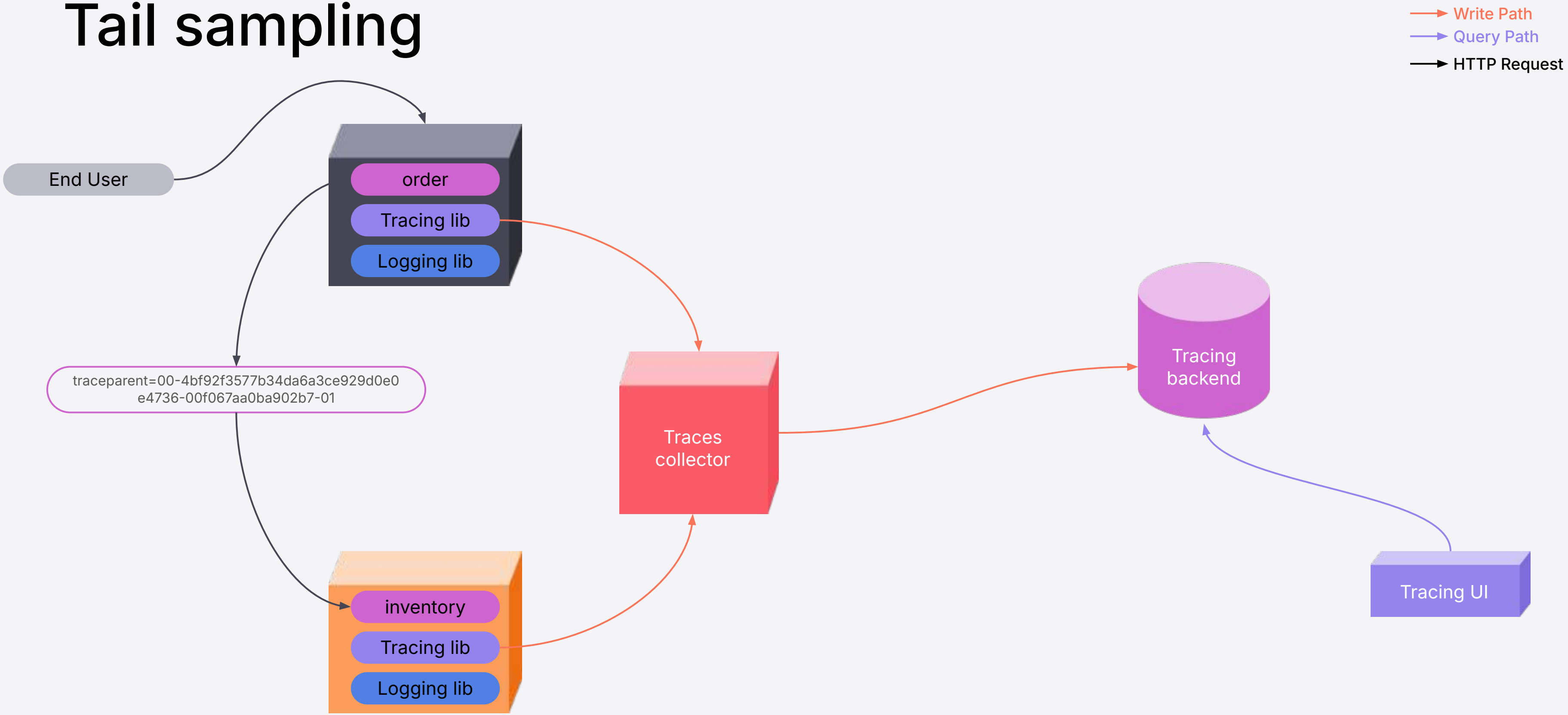
- Highly interesting traces
- Allows for complex use-cases
- Requires some effort to scale
  - A load-balancer that can use the traceID to make the decision
- Higher cost of ownership
  - Higher resource consumption
  - Likely to have multiple instances of it in production
  - Likely to have another layer of collector for load-balancing
- Even harder to extract statistics out of it
  - What was the policy that caused a trace to be sampled? How many were discarded?

Tail sampling



```
tail_sampling:
  decision_wait: 1s
  num_traces: 50_000
  expected_new_traces_per_sec: 500
  policies:
    [
      {
        type: and,
        and:
          {
            and_sub_policy:
              [
                {
                  name: only-10-percent,
                  type: probabilistic,
                  probabilistic: { sampling_percentage: 10 },
                },
                {
                  name: vip,
                  type: string_attribute,
                  string_attribute: { key: vip, values: ["true"] },
                },
              ],
          },
      },
    ]
```

Tail sampling



Write Path

Query Path

HTTP Request

End User

order

Tracing lib

Logging lib

traceparent=00-4bf92f3577b34da6a3ce929d0e0
e4736-00f067aa0ba902b7-01

inventory

Tracing lib

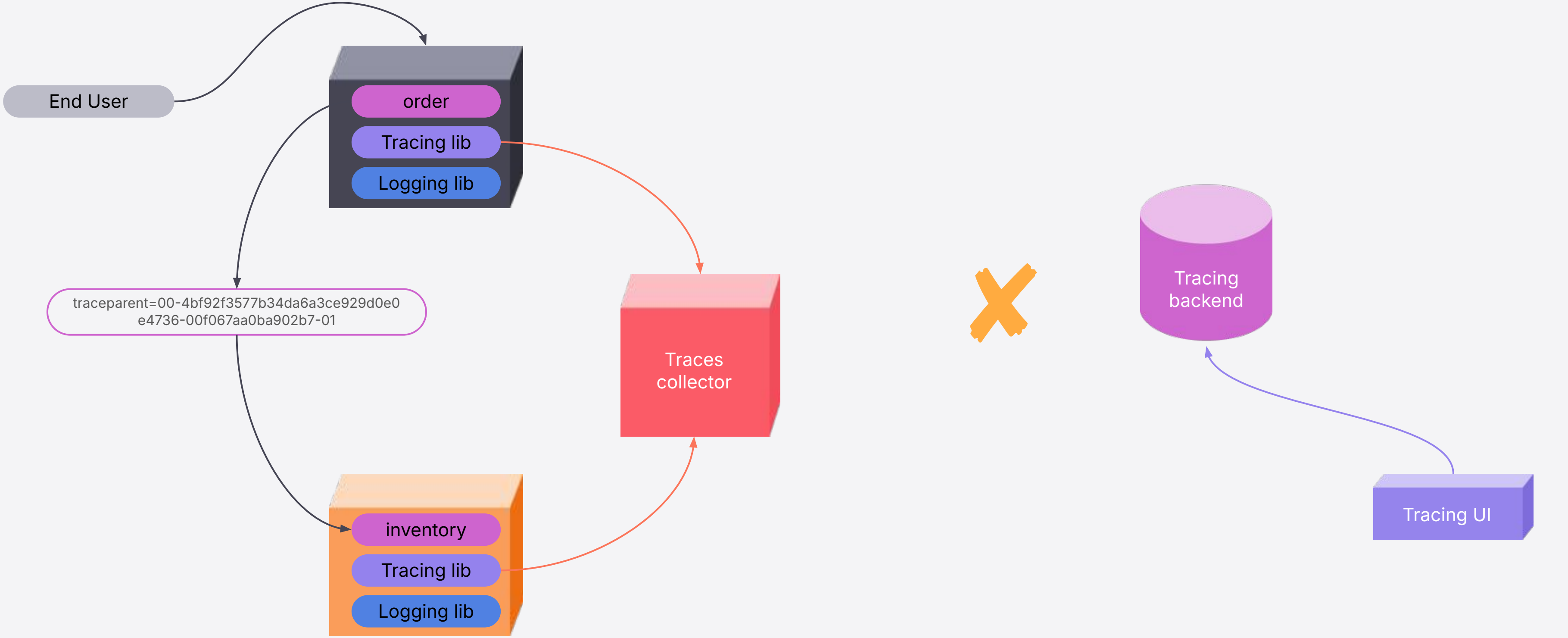
Logging lib

Traces collector

Tracing backend

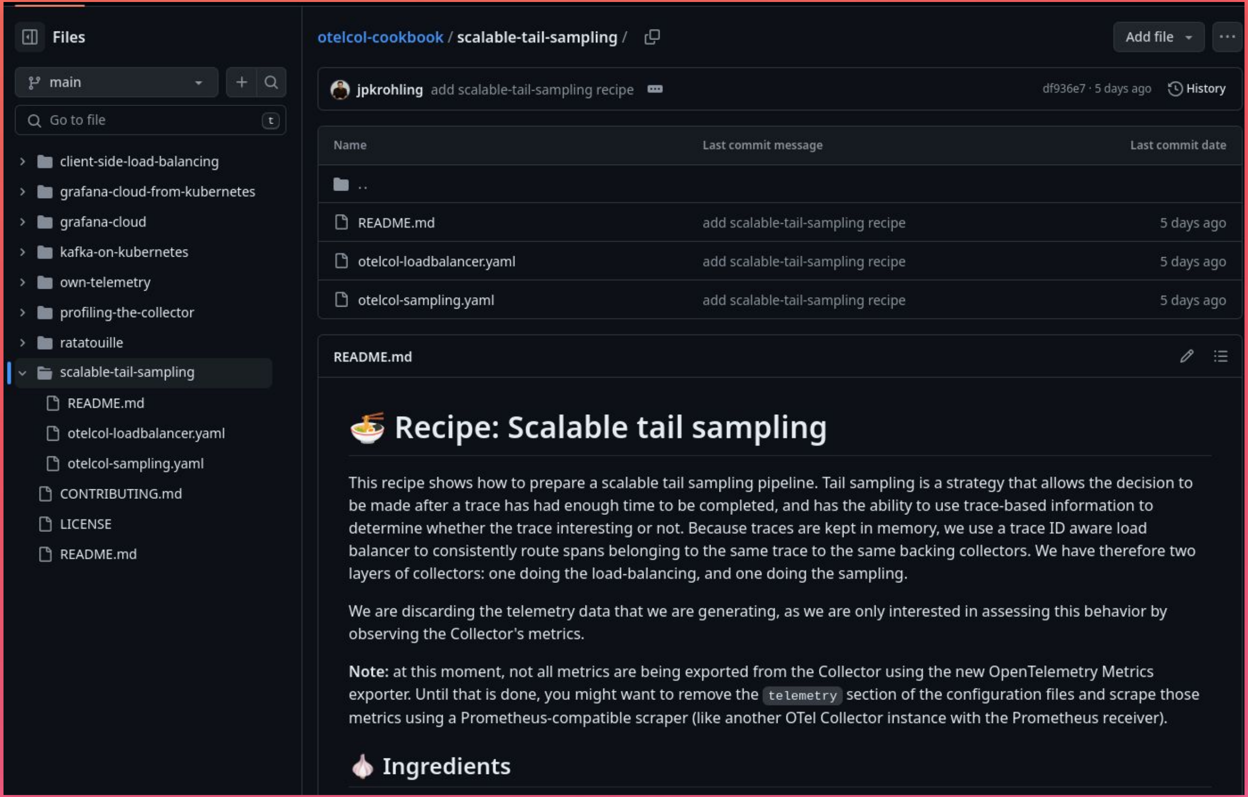
Tracing UI

Tail sampling



@jpkrohling

Hint: jpkrohling/otelcol-cookbook



jpkrohling add scalable-tail-sampling recipe df936e7 · 5 days ago

Name	Last commit message	Last commit date
..		
README.md	add scalable-tail-sampling recipe	5 days ago
otelcol-loadbalancer.yaml	add scalable-tail-sampling recipe	5 days ago
otelcol-sampling.yaml	add scalable-tail-sampling recipe	5 days ago

README.md

🍜 Recipe: Scalable tail sampling

This recipe shows how to prepare a scalable tail sampling pipeline. Tail sampling is a strategy that allows the decision to be made after a trace has had enough time to be completed, and has the ability to use trace-based information to determine whether the trace interesting or not. Because traces are kept in memory, we use a trace ID aware load balancer to consistently route spans belonging to the same trace to the same backing collectors. We have therefore two layers of collectors: one doing the load-balancing, and one doing the sampling.

We are discarding the telemetry data that we are generating, as we are only interested in assessing this behavior by observing the Collector's metrics.

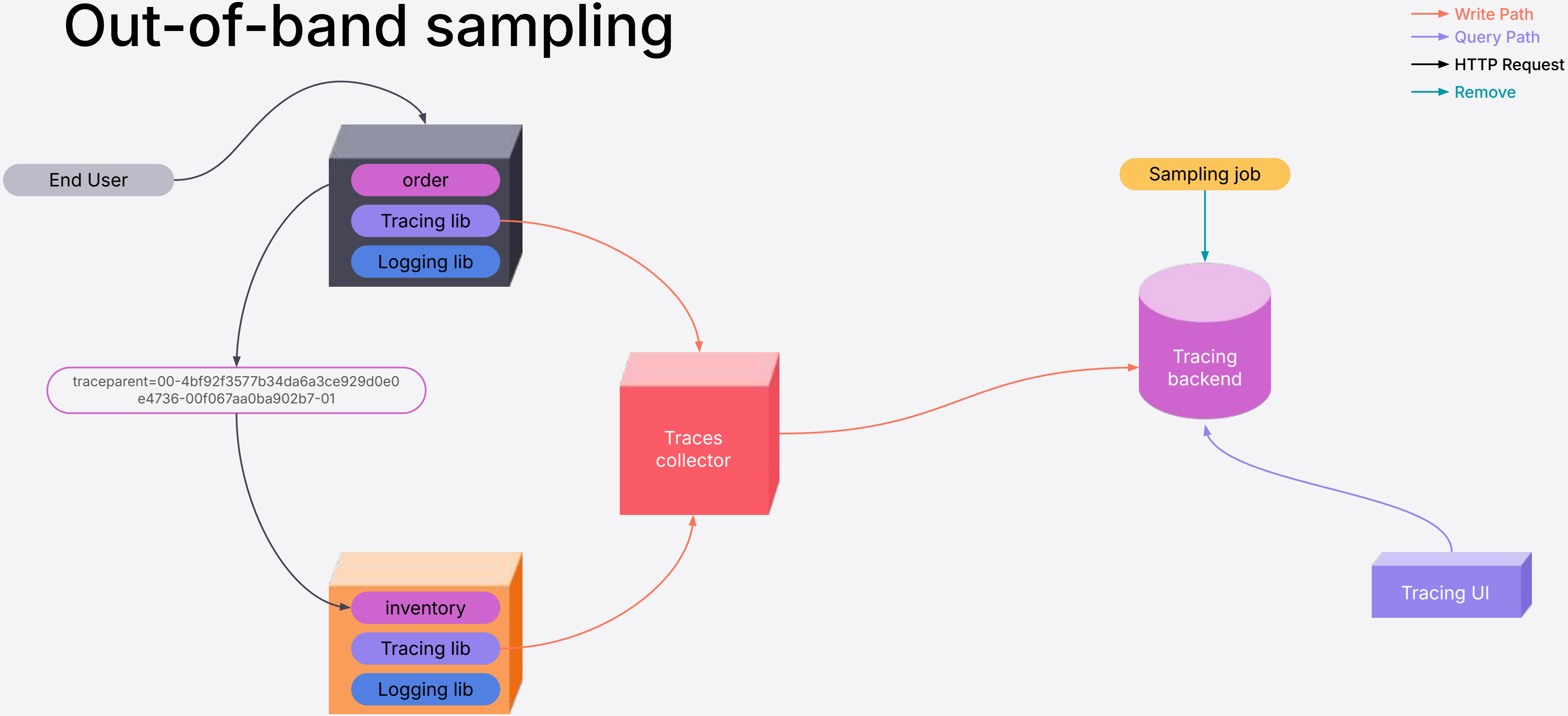
Note: at this moment, not all metrics are being exported from the Collector using the new OpenTelemetry Metrics exporter. Until that is done, you might want to remove the `telemetry` section of the configuration files and scrape those metrics using a Prometheus-compatible scraper (like another OTel Collector instance with the Prometheus receiver).

🧄 Ingredients

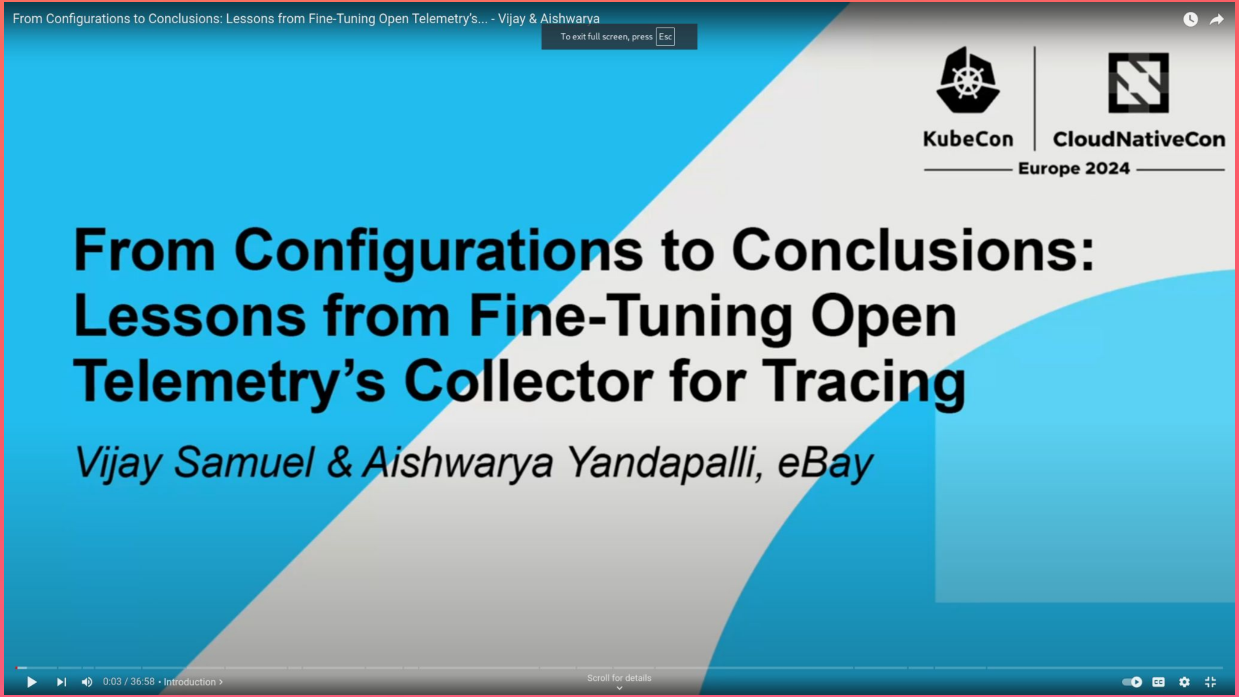
Out-of-band sampling

- Ingests everything, removes later
    - Perhaps with two stores
        - full data for a short period of time
        - sampled data for higher retention
- High-throughput systems
- Simplified architecture
- Requires development of custom components
- Statistics are possible!

Out-of-band sampling



Tip:



From Configurations to Conclusions: Lessons from Fine-Tuning Open Telemetry's... - Vijay & Aishwarya

To exit full screen, press Esc

KubeCon | CloudNativeCon
Europe 2024

From Configurations to Conclusions: Lessons from Fine-Tuning Open Telemetry's Collector for Tracing

Vijay Samuel & Aishwarya Yandapalli, eBay

0:03 / 36:58 • Introduction ›

Scroll for details

https://youtu.be/RSJwv1jOdTg

Bonus: sampling and Profiles

Sampling for profiles

- Profiling frequency
- Profiling time window
- A fraction of the instances of a service

Q&A time!

Grafana Labs

Obrigado!

Grafana Labs

@jpkrohling