Pimp my Traces

PRESENTER

Henrik Rexed

Henrik Rexed

Cloud Native Advocate, CNCF Ambassador

- 15+ years of Performance engineering
- Owner of IsitObservable

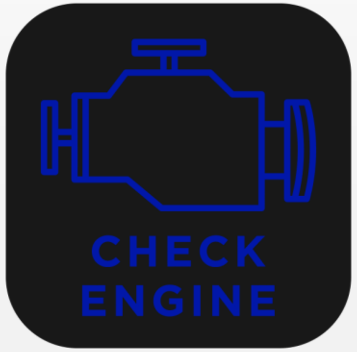
Producer of Perfbytes

If you pimp with me ....

- Reminder on the importance of Sampling
- The reality of distributed application
- We will remind the various Sampling decisions available in OpenTelemetry
  - Head Sampling
  - Collector Sampling
- Introduce Consistent probabilistic sampling
- Share Best Practices on Collector deployments
- Look at Benchmarking results

We are so cool…

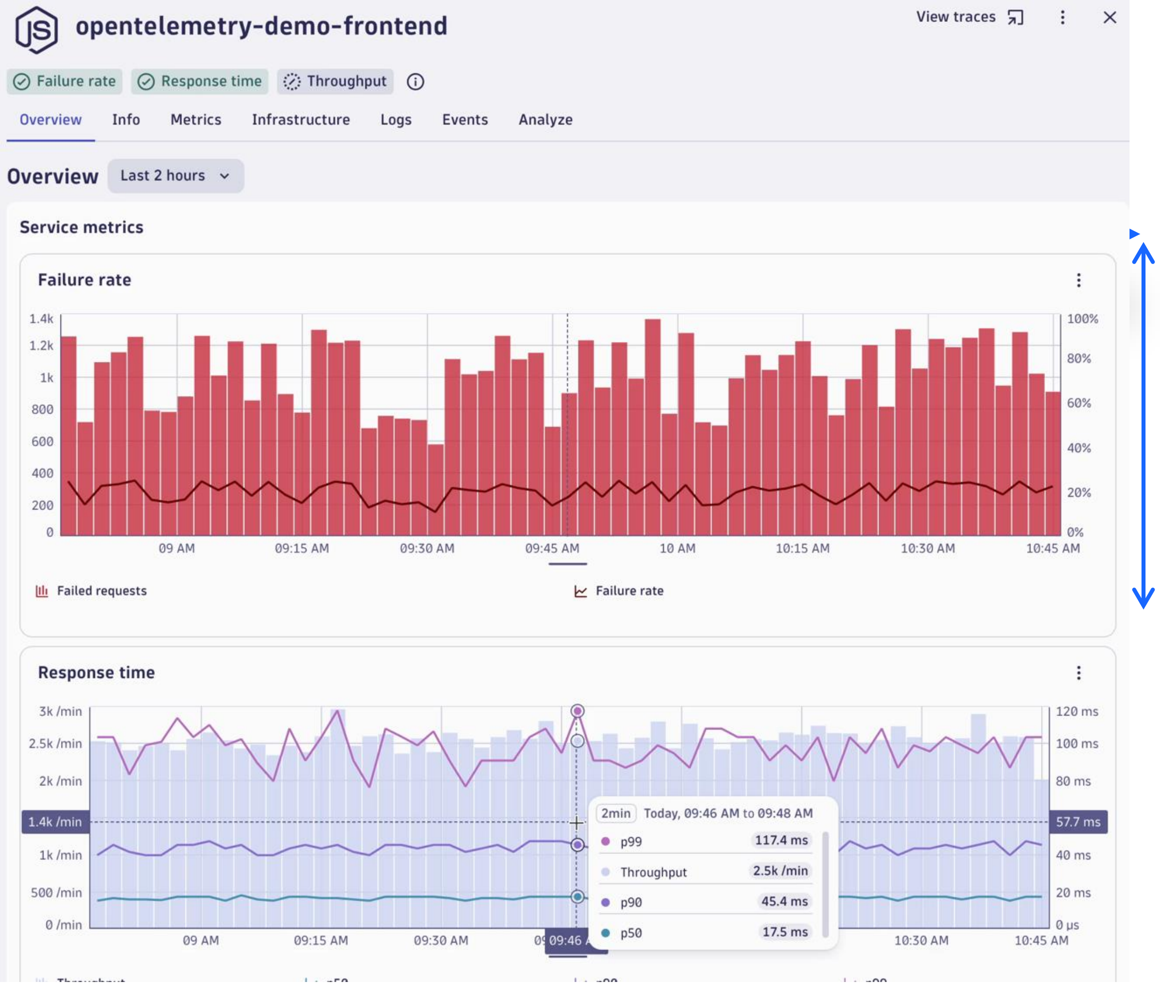
But Wait a minute…



Our engine

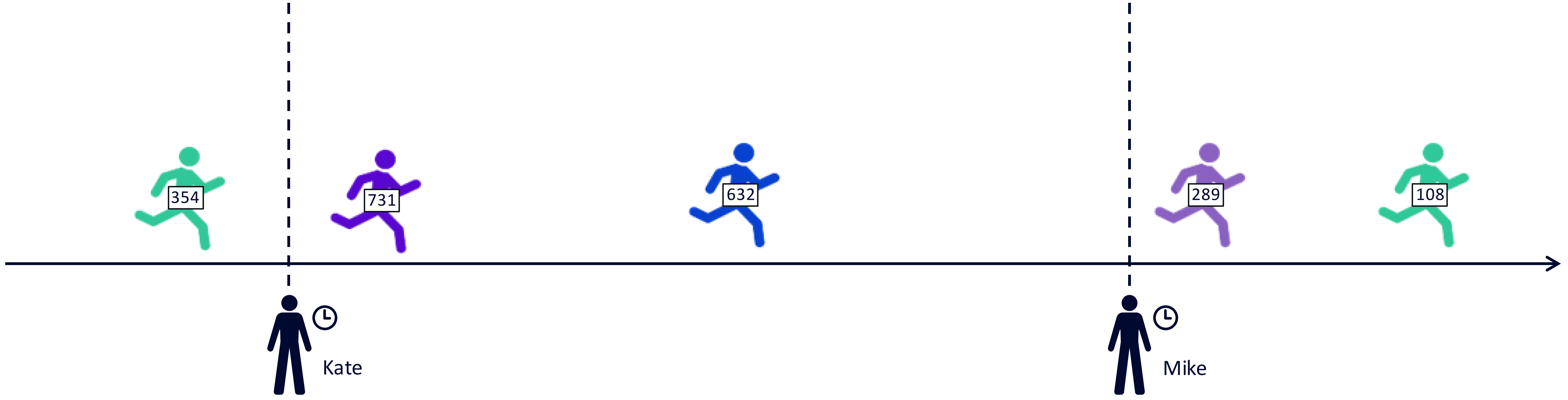
- Let me i

The Root S



Importance of Sampling

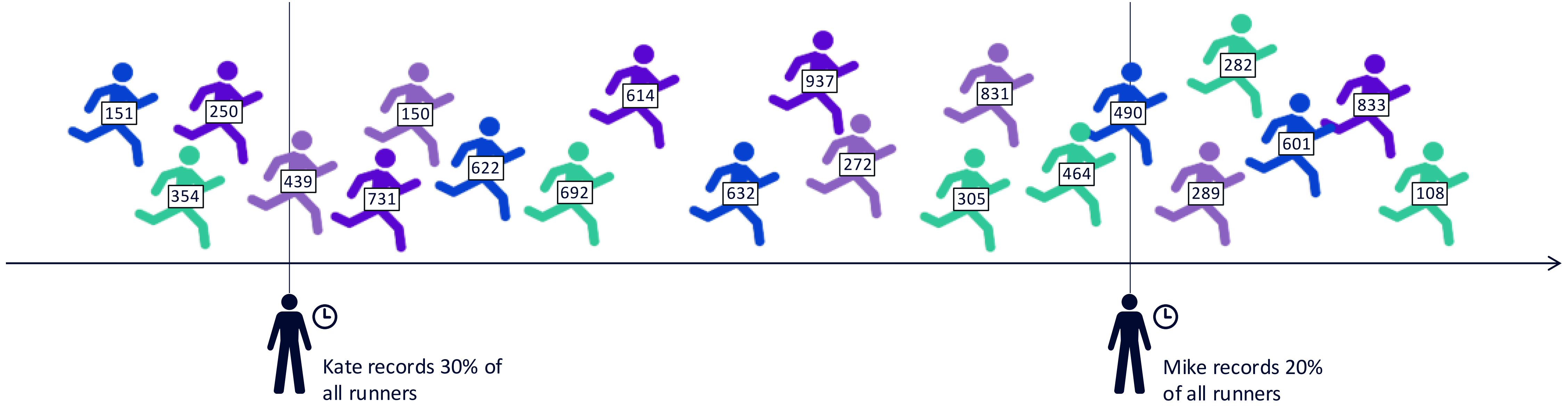
The Theory



id	time
289	10:45
108	11:31
632	12:01
731	12:14
354	13:48

id	time
108	12:59
289	13:07
632	13:46
731	14:03
354	15:40

The Reality



id	time
833	10:47
831	11:29
305	12:03
632	**12:01**
250	13:51
…	…

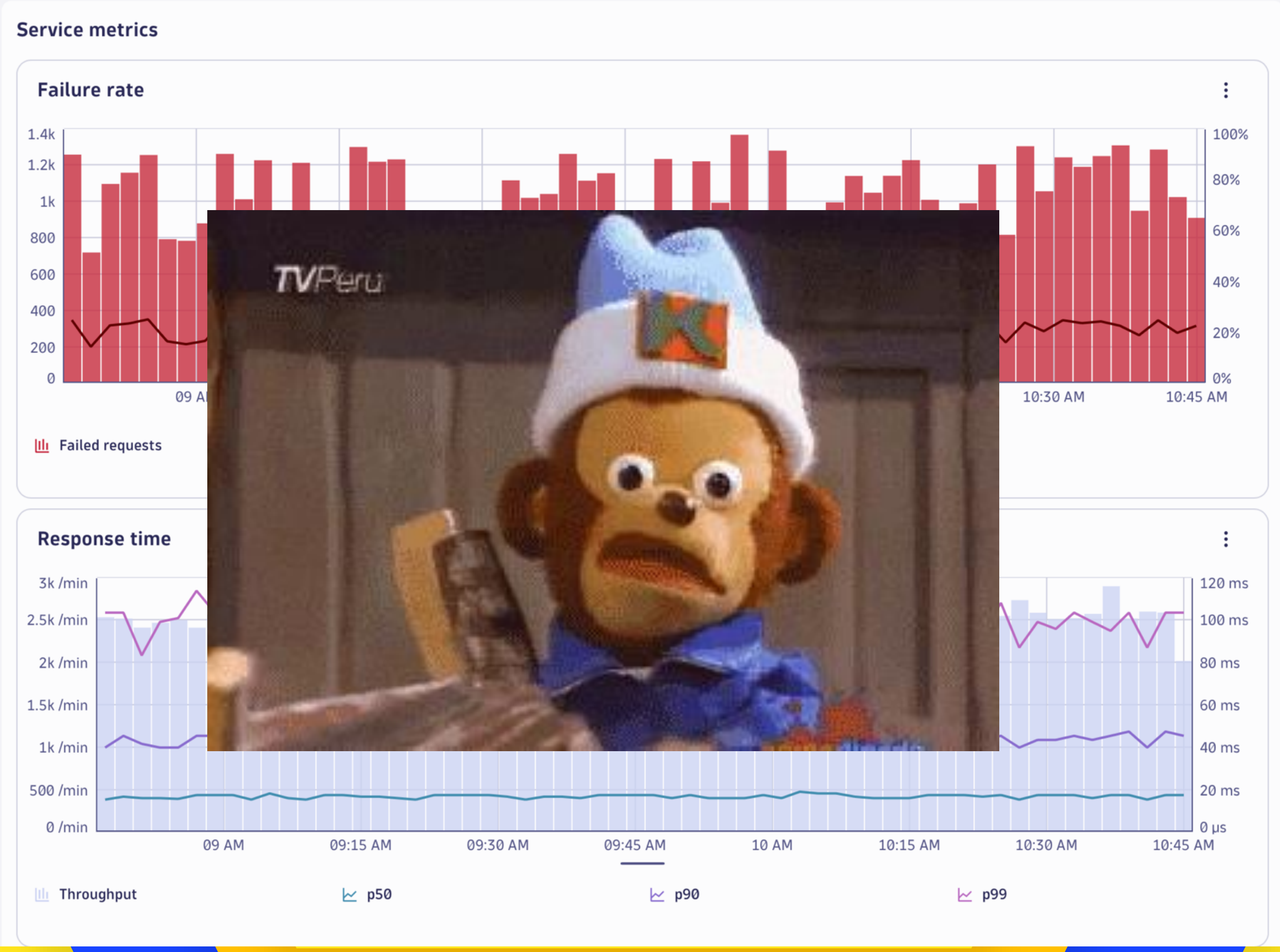
id	time
108	12:59
289	13:07
632	**13:46**
731	14:03
354	15:40
…	…

Multiplication rule of probability: **P = 0.3 x 0.2 = 6%**

Time can be calculated for 6% of all runners!

Distributed Sampling Problem

X_{12}

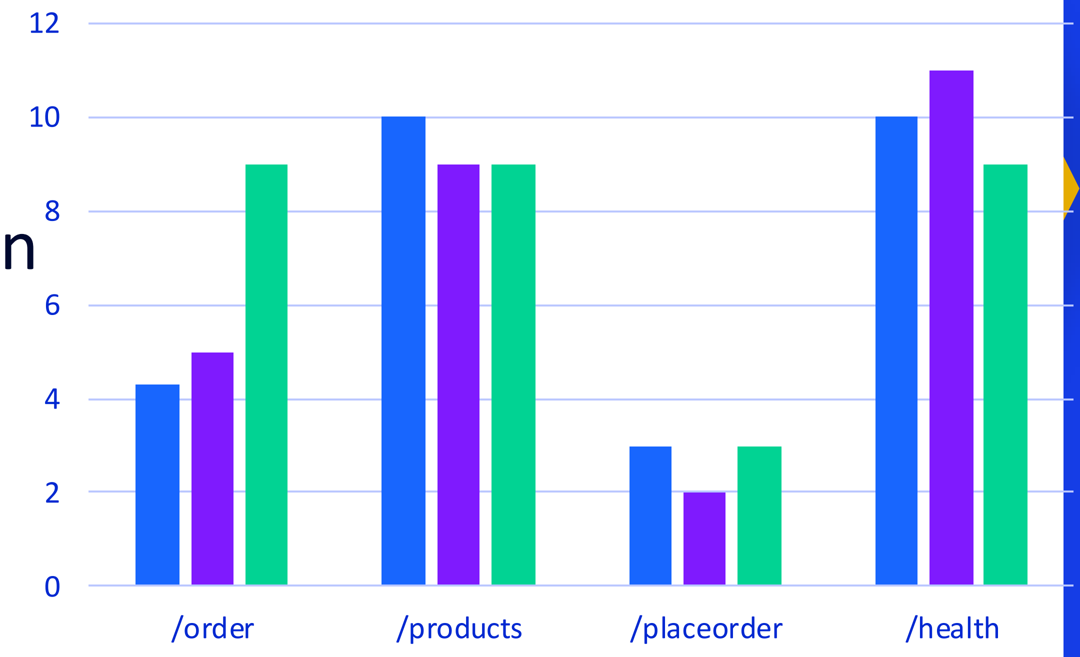


X_1

The reality

All our Endpoints are not equals

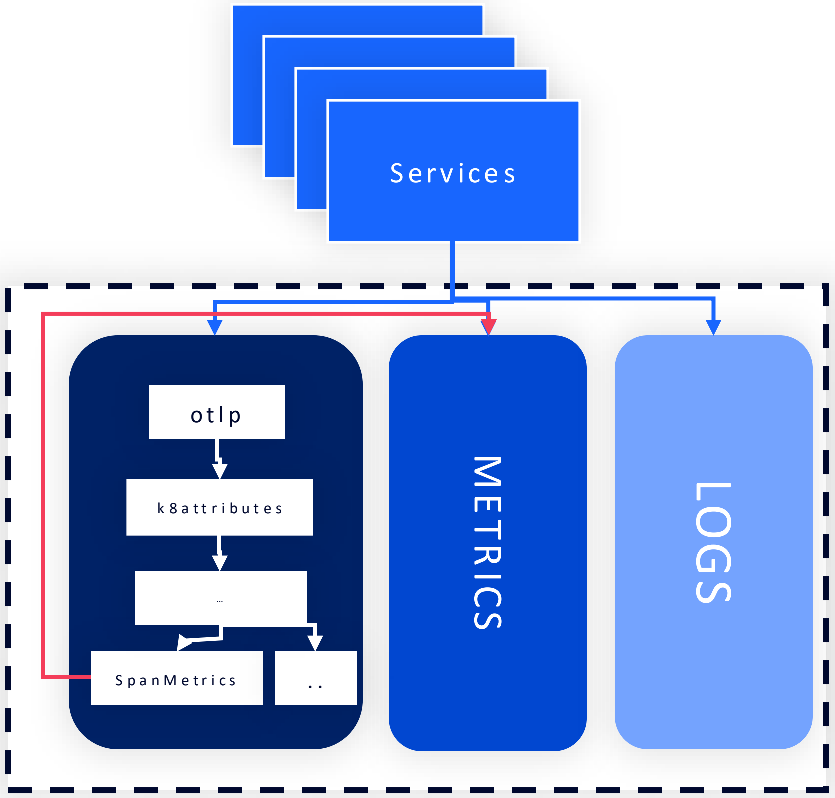
- Each Endpoints of our services are used equally
- Critical functions are often used less often
- Applying a fixed sampling will mask our critical functions



Solution

SpanMetrics Connector

- The OpenTelemetry Collector provides the spanmetrics Connector
- This connector will generate metrics out of our spans reporting:
  - The response time
  - The number of request and their status
- The spanMetrics Connector needs to be applied before any sampling decision to avoid losing details.



SpanMetrics

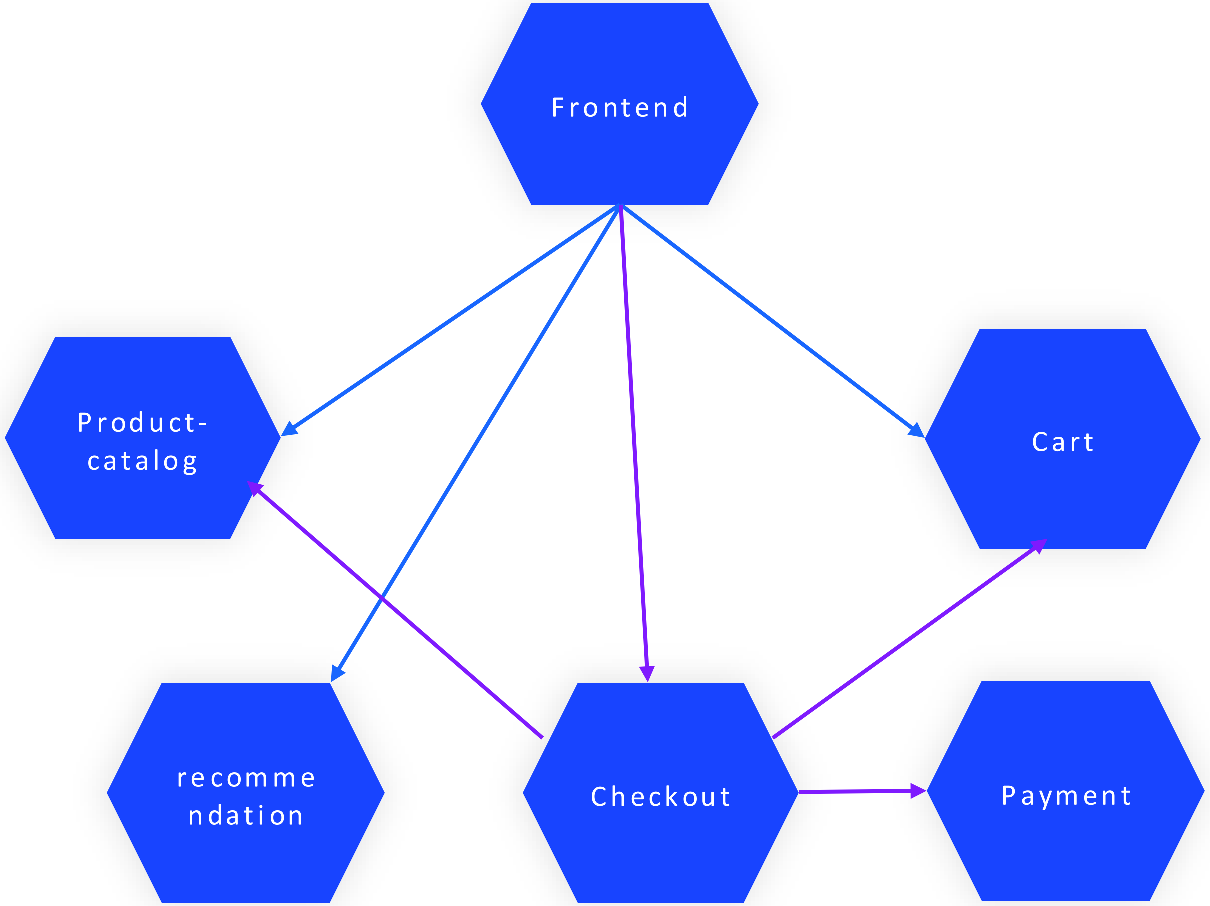
Head Sampling

OpenTelemetry Head Sampling

- AlwaysOn
- AlwaysOff
- ParentBased
- TraceidRatio

ParentBased

- ParentBased is designed for any services that are not creating the root span
- Only spans involved in a End2End transaction would be sampled
- Perfect for backend dependencies
- Limitations : losing any health Check spans, or asynchronous processes

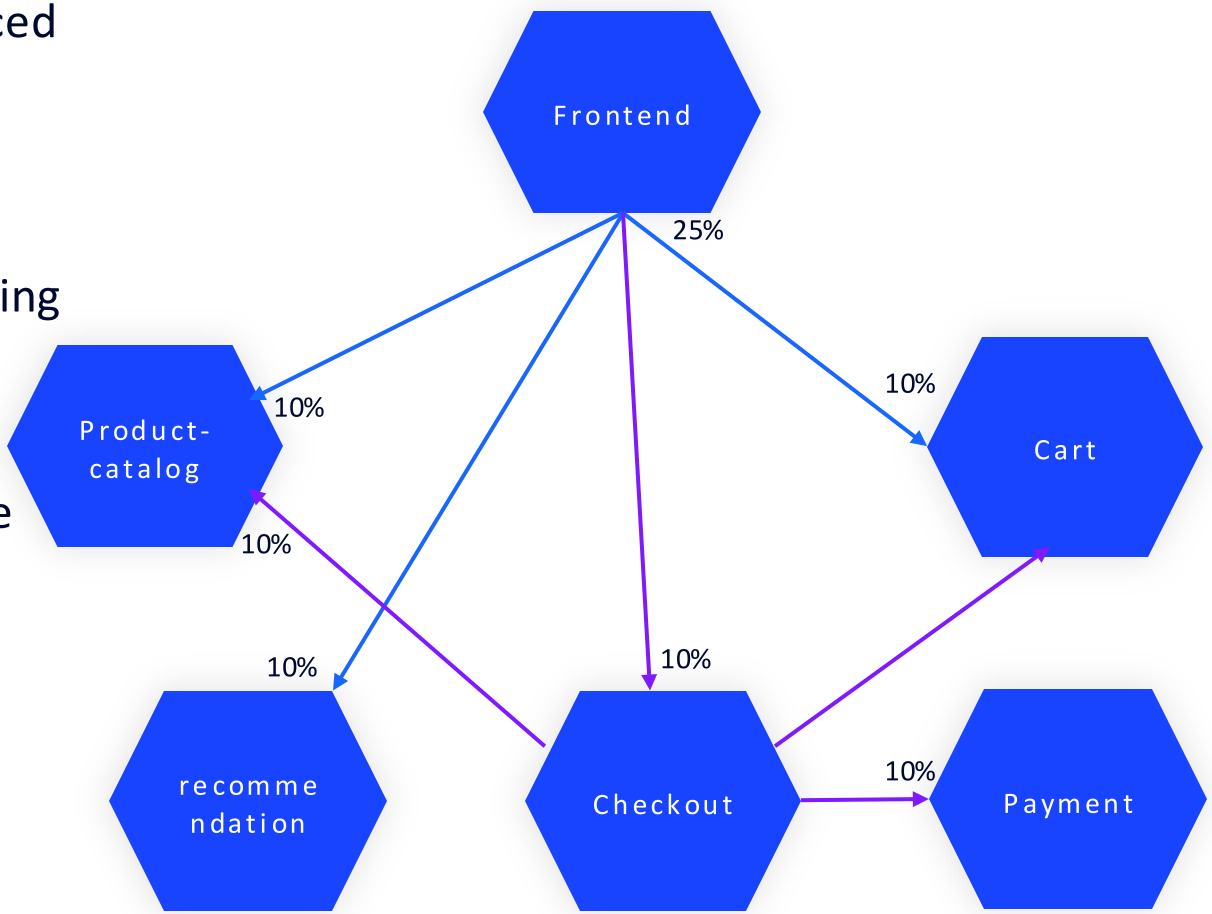


TraceIdRatio

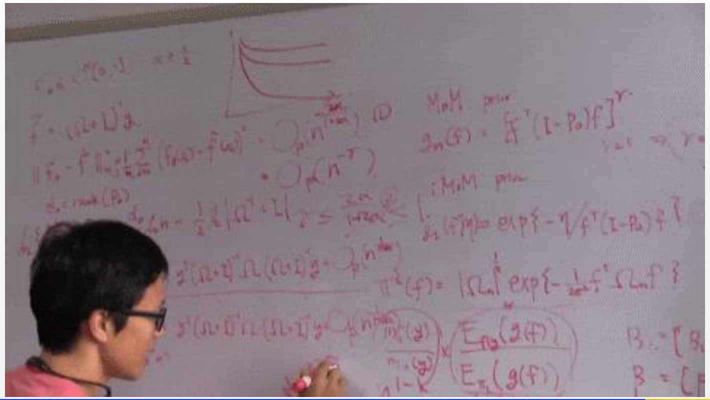
- TraceIdRation apply a fixed ratio on the spans produced by the service
- The traceContext will hold the information that one Spans has been sampled but it won't store the sampling decision
- TraceIdRatio samples the local spans produced where the tracecontext is sampled.
- Bad configuration would limit the number spans exported.

  example: to get the payment transaction

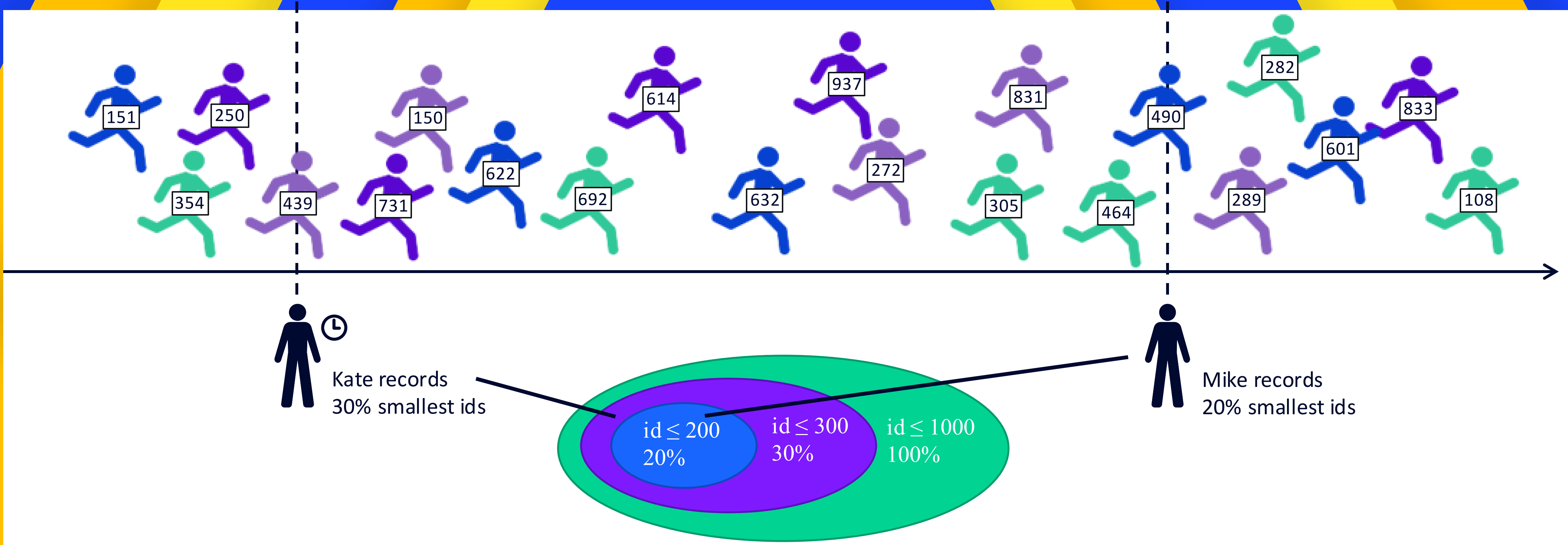
*nbspans= total*0,25*0,10*0,10 = 0,25% of total spans*



Consistent Probabilistic Sampling



Consistent Probabilistic sampling



Assumption that ids are uniformly random

Probability that Kate and Mike measure time simultaneously is given by P = min(30%, 20%) = 20%

Time can be calculated for 20% of all runners! (was just 6% with inconsistent sampling)


LA RUN'RANGE
4AZ54FEDE98D
calenrun
MAISON
NICOLAS
bouchers de père en fils
depuis
1976
METZ
FRIGO EST
TRANSPORT EXPRESS FRIGORIFI
NATIONAL ET INTERNATIONAL

Details stored in TraceState

- The other major advantages of this approach are :
  - Better sampling decisions applied within our architecture
  - The sampling decision is stored in the trace state
  - Observability Backend can rely with this details to extrapolate precise stats with the field *th*

    (Rejection Threshold)

   where th: (1 – sampling ratio) * 2^56

traceparent:

00-4bf92f3577b34da6a3ce929d0e0e4736-00f067aa0ba902b7-01

tracestate: ot=th:8;rv:9b8233f7e3a151

Collector Sampling

Sampling Processor

- Probabilistic
- Tail

Probabilistic Sampling

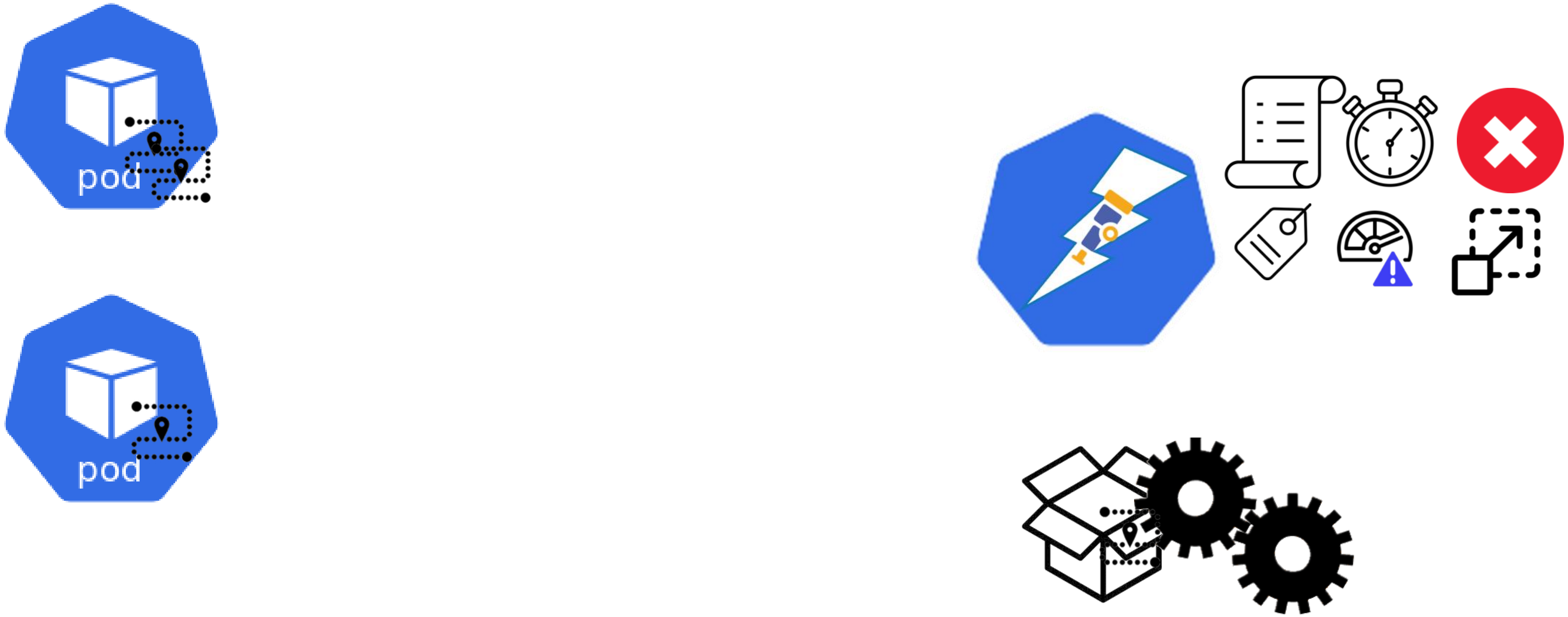
- The probabilistic processor can sample logs and traces
- Probabilistic sampling relies on “modes”:
  - Hash seed : designed for logs
  - Proportional
  - Equalizing
- Proportional and equalizing are designed for pre-sampled traces
- Proportional will set an equal ratio
- Equalizing will calculate the ratio to get an equal probability on spans

```
probabilistic_sampler:
    sampling_percentage: 15.3
    mode: "proportional"

  probabilistic_sampler/logs:
    sampling_percentage: 15.3
    hash_seed: 22
     attribute_source: "record"
     from_attribute: "foo"
     sampling_priority: "bar"
```

Sample like a pro

Tail sampling: Reflect real life



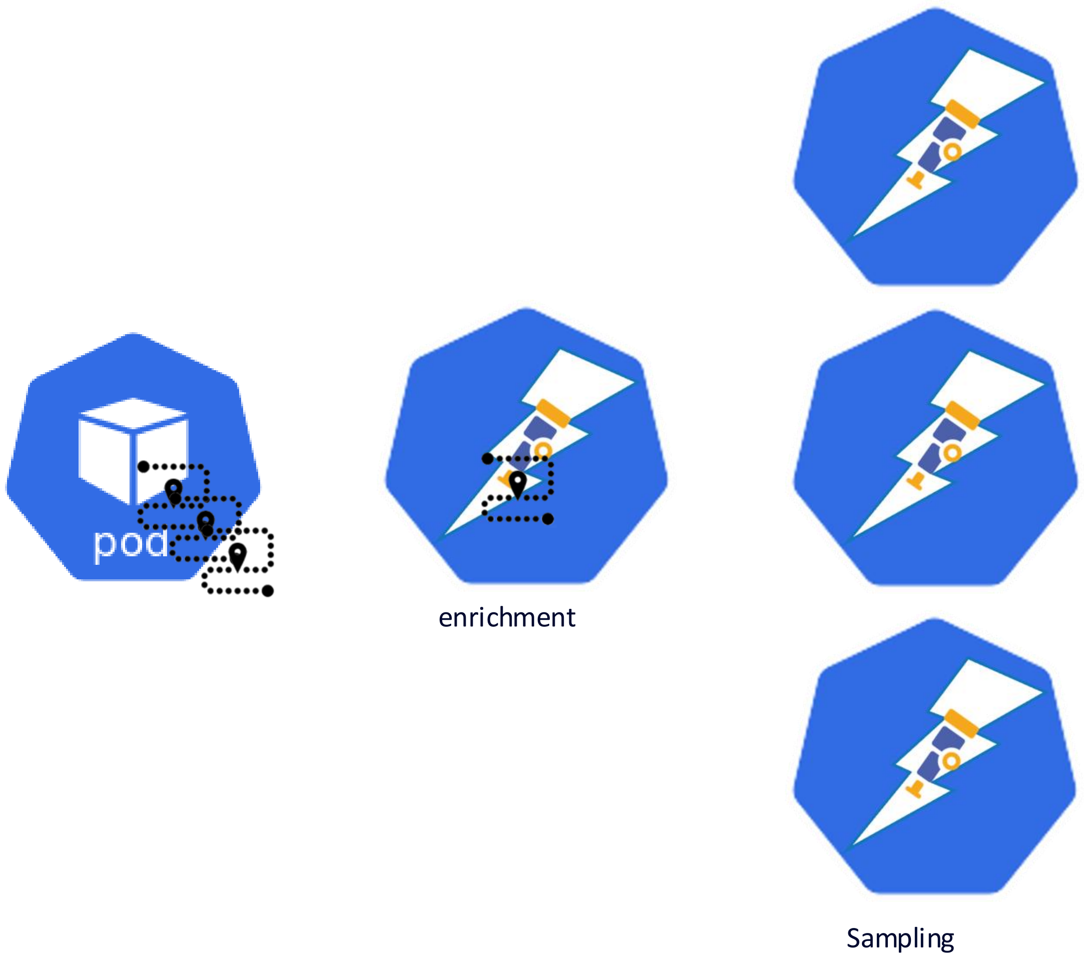
Tail sampling allow you to be very creative

- The tail sampling processor allow you to define fine grained policies based on:

- Using tail sampling requires to define precisely:
  - The time out : decision_wait
  - The number of traces to keep in memory : num_traces
  - The cache size but defining the size of sampled traces, and the non sampled cache

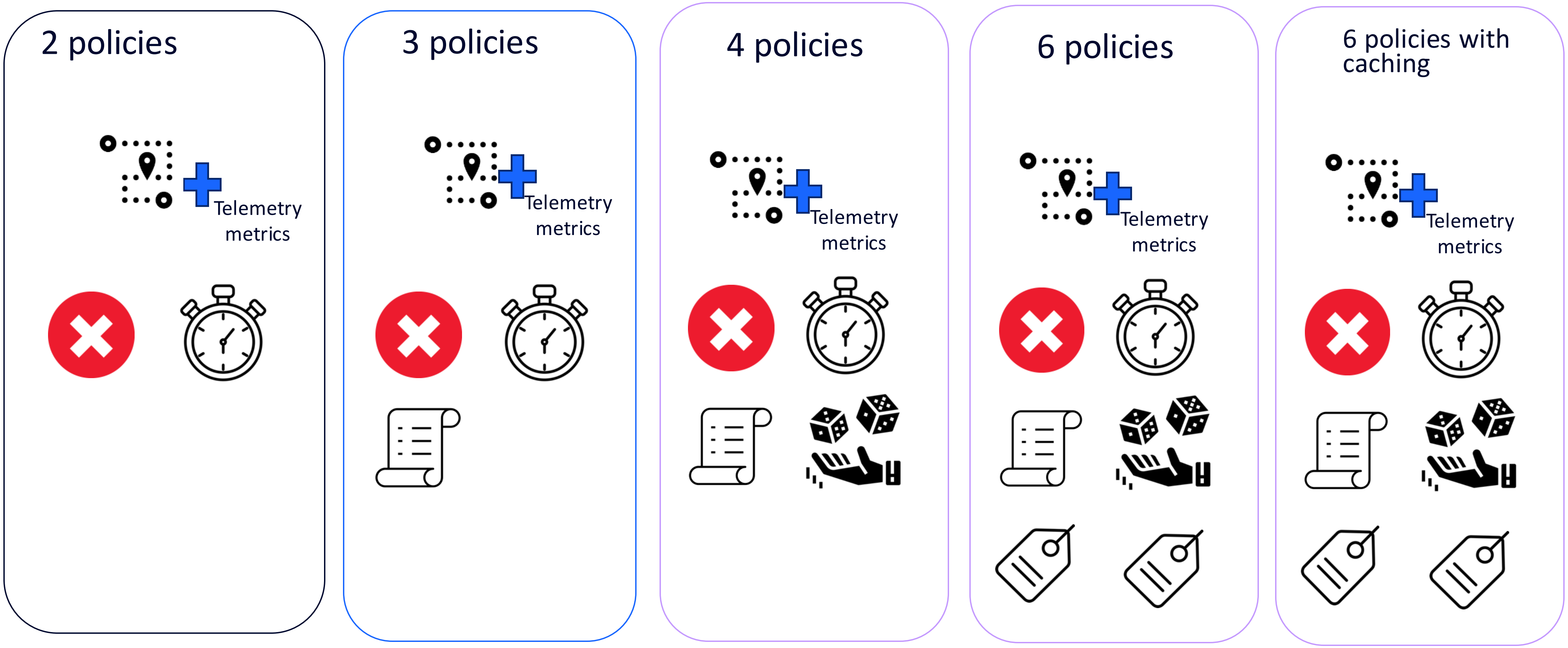
But you need do it right

- Tail sampling. Consume memory
  - Make sure to define the cache size and timeout
- Use several Collector to reduce memory usage:
  - Collector to enrich and producing spans Metrics
  - Collector to Sample and metric conversion
- Rely on the **LoadBalancer** Exporter
- Enable Sampling metrics to Identify Sampling issues

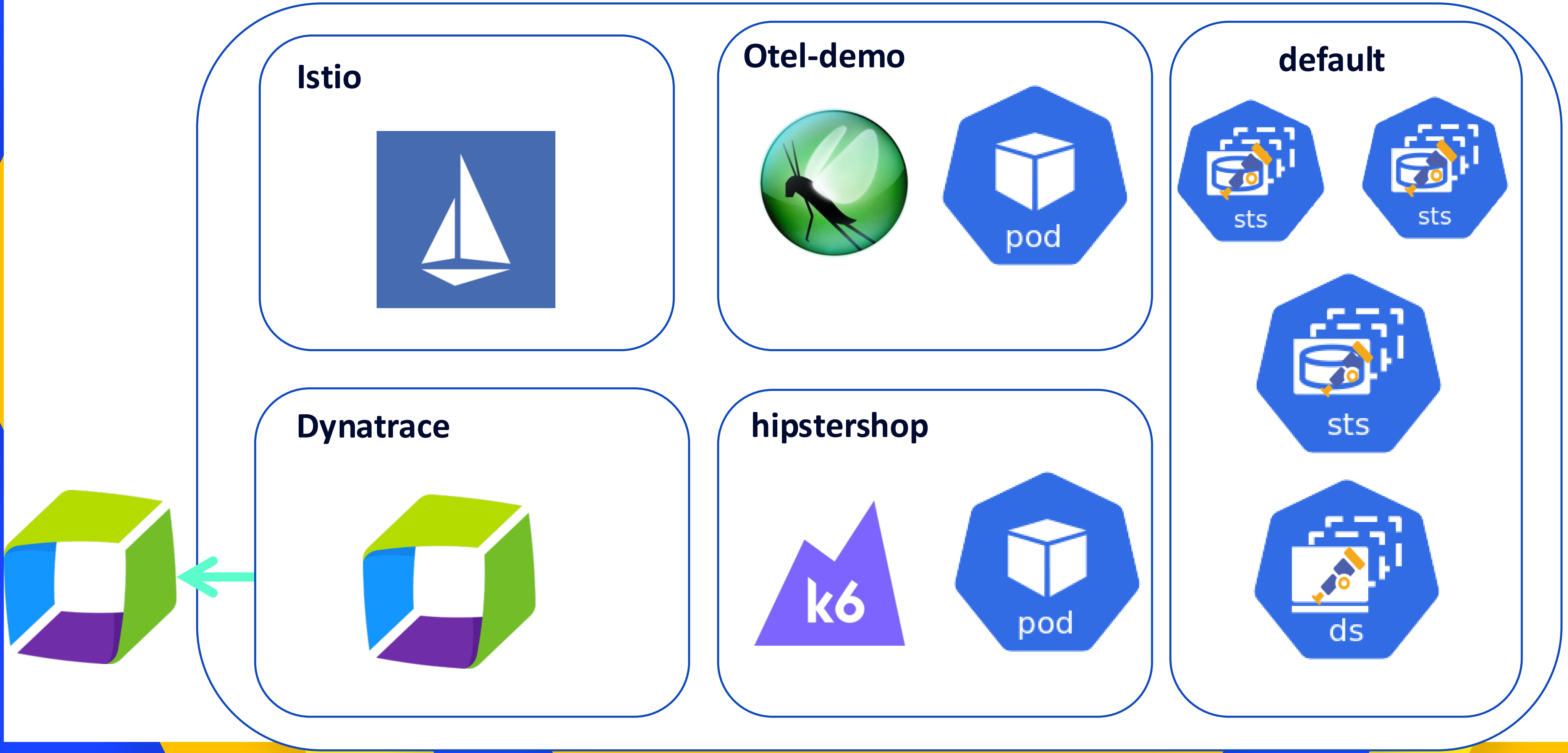


Benchmark on tail sampling

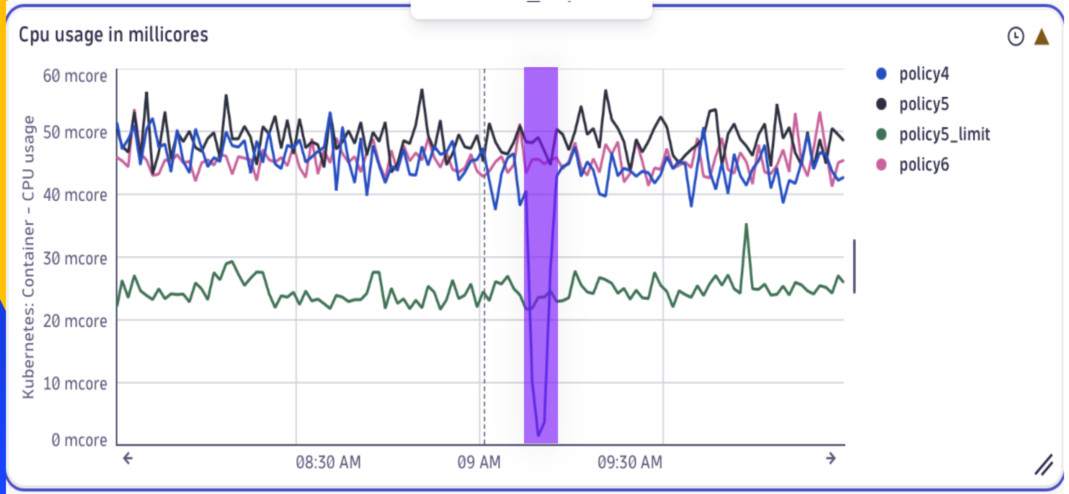
What are we comparing

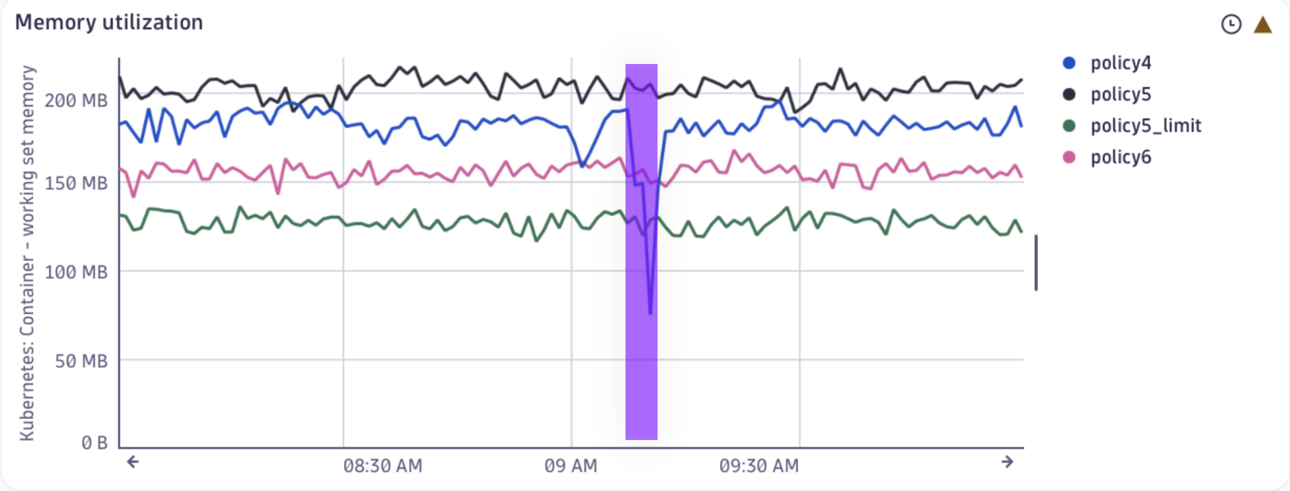


Architecture



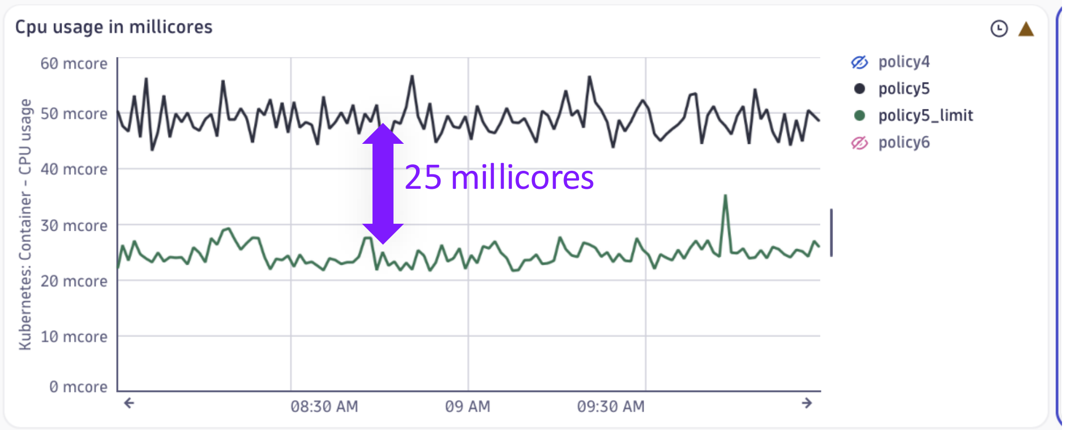
Does the number of policies impact our collectors

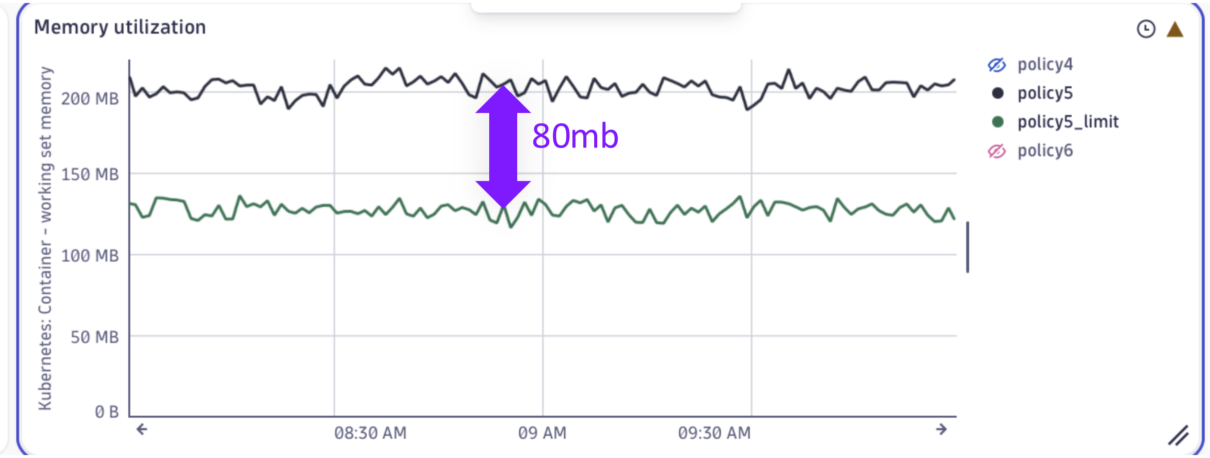




Value of caching behavior

- Defining your cache size, and the number of traces stored in memory reduce the resource usage





Take Away

Pimp My trace

Head

- Use Parent based
- Be cautions of TraceidRatio

Collector

- Use Tail Sampling
- Define precise sampling rules
- Enable Collector metrics

Architecture

- Use several collector
- Take advantage of the LoadBalancer exporter

Pimp my channel

- Looking for educational content on Observability , Checkout the YouTube Channel :

Is It Observable

Is it Observable ?