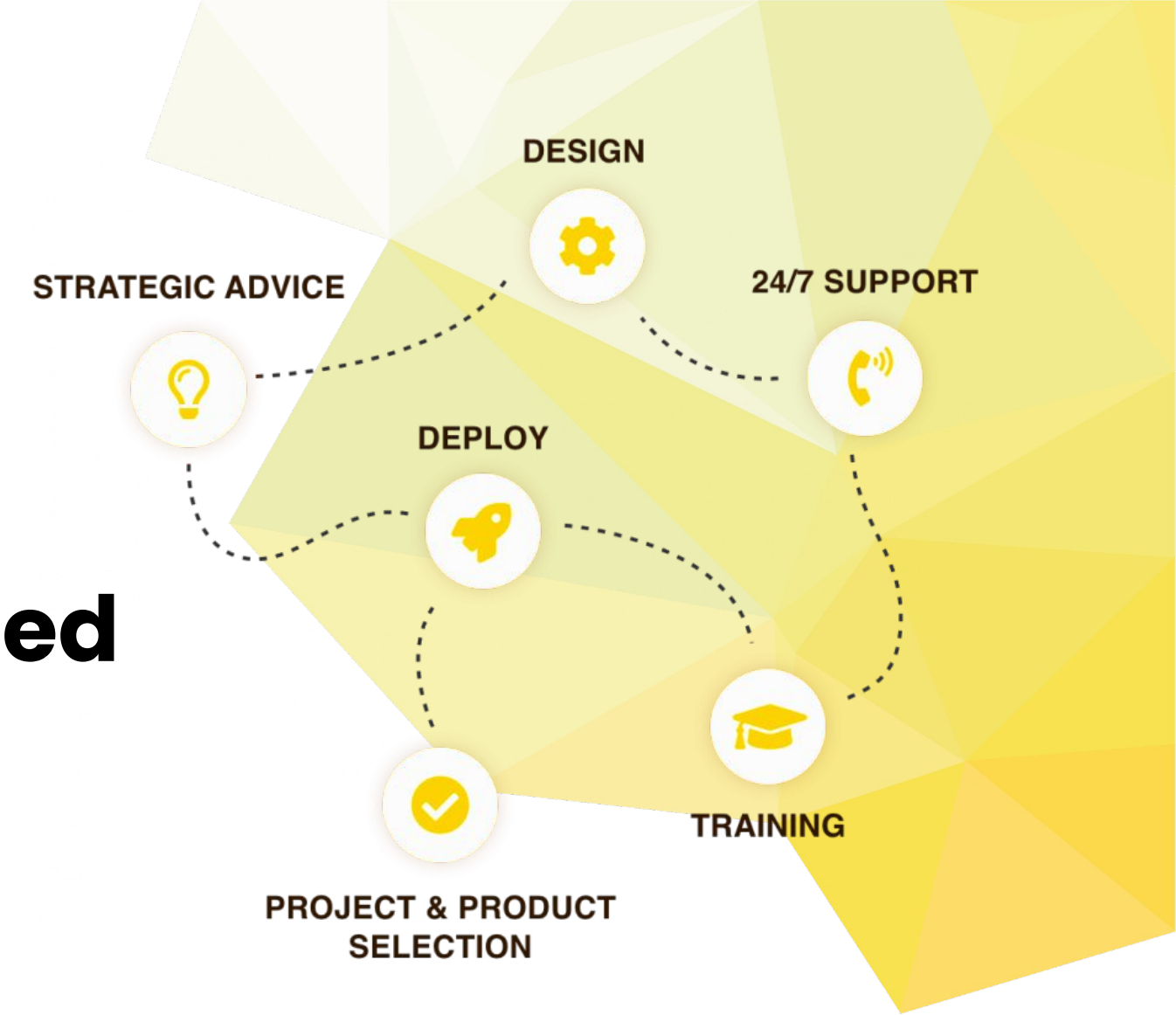
kangaroot
Linux & Open Source Solutions



Debugging Distributed Systems

Agenda

- Evolution of observability
- Distributed tracing
- OpenTelemetry
- Demo with Grafana & Elastic

Observability

Traditional monitoring

- reactive
- depends on experience and intuition of humans
- relies heavily on **metrics**
  - service oriented
  - aggregated
  - limited cardinality
- and **logs**
  - often not consistent
  - lag context

Evolution of software development

monolithic architecture

1 big team

handover from dev to ops

microservice architecture

many smaller teams

devops teams manage the complete lifecycle

Evolution of software development

- missing full picture of the entire application
- hard to keep track of all changes
- higher agility, reliability, scalability at the expense of operational complexity
- systems fail in new and unexpected ways

Without the right visibility across systems,

teams struggle to find the root cause of issues.

What is observability?

"understand the inner workings and state of your application
by interacting with external tools only,
without shipping new code,
for all possible scenarios that may or may not have happened before"

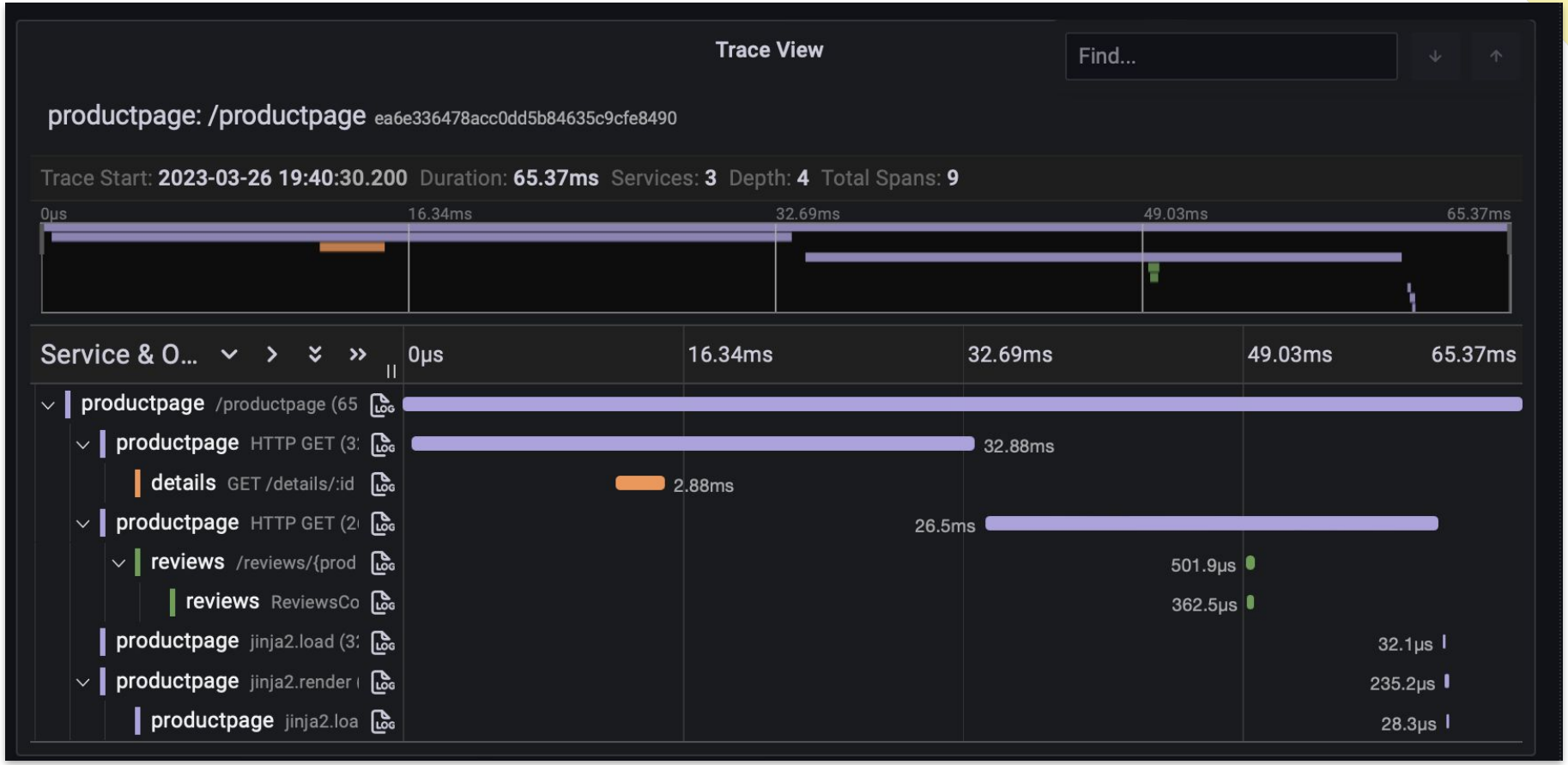
What is observability?

- predefined set of metrics or logs
- aggregated data
- approximation of overall system health
- reactive
- relies on experience and intuition of humans

- high-dimensional and high-cardinality events
- zoom in to a single request
- find the outliers
- explore arbitrary properties and patterns
- approach the investigation objectively

Distributed tracing

What is a trace?



What is a trace?

- series of rich event data generated at various points throughout a system tied together via a unique identifier
- identifier is propagated across all components
- each trace represents a unique request through a system
- each operation recorded in a trace is represented by a span

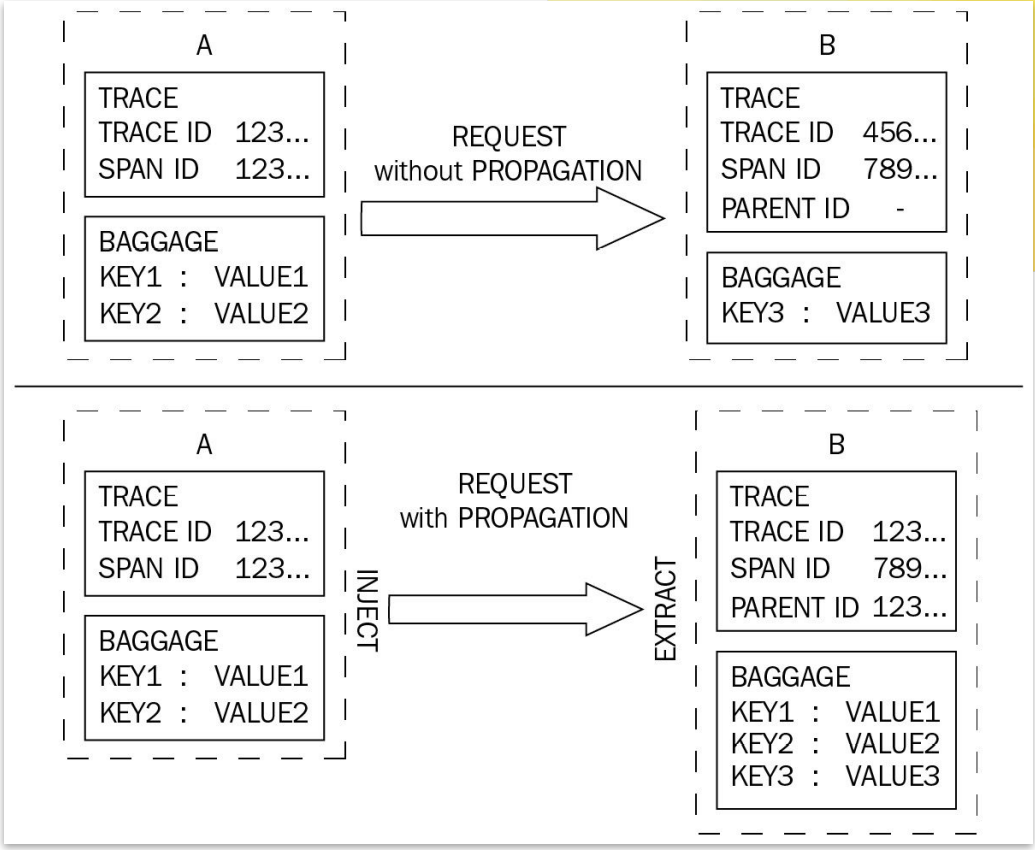
Anatomy of a span

- Name
- Parent span ID
- Start and End Timestamps
- Attributes
- Span Events
- Span Links
- Span Status
- Span Context
  - trace ID
  - span ID
  - trace flags
  - trace state for vendor-specific info

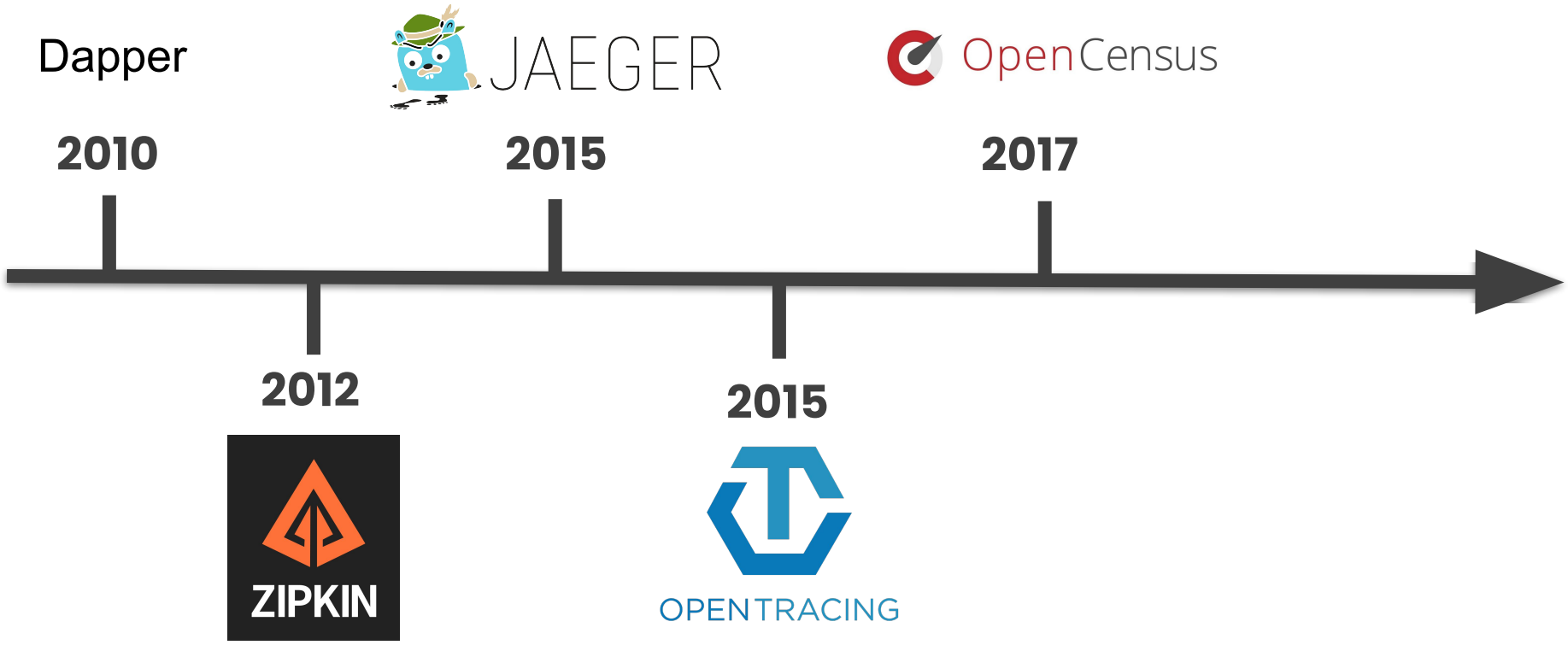
Context propagation

core concept that enables Distributed Tracing

- ◇ correlate spans and assembled into a trace
- ◇ both within a service and cross-services



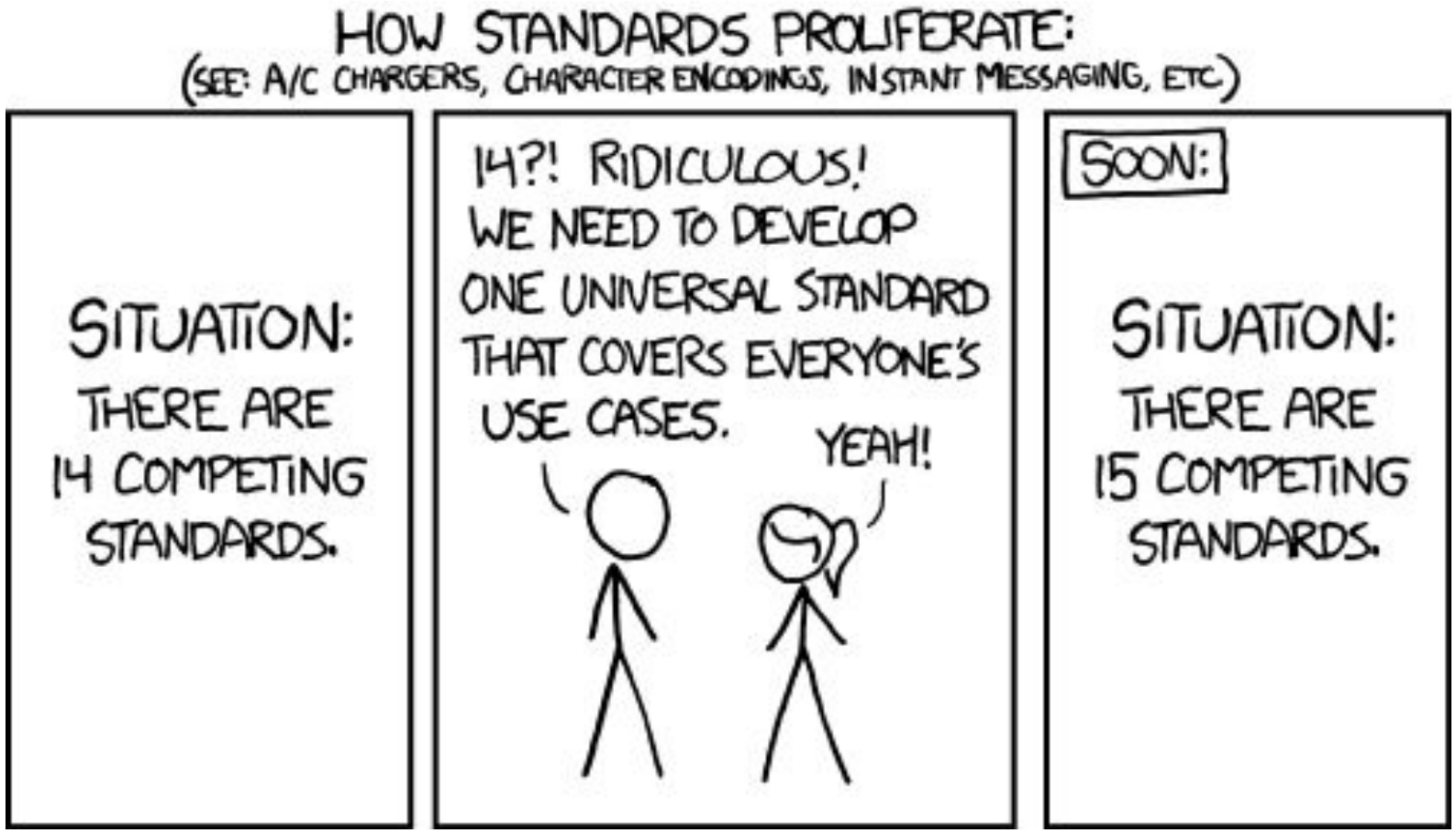
Distributed tracing



Distributed tracing

fragmented landscape

- 2 formats to send traces to backend
- 2 standards for generating traces
- daunting to know where to begin
- difficult to change tool
- libraries supporting one format or another
- no context propagation between OpenTracing & OpenCensus



(credit: XKCD, https://xkcd.com/927/)

2019: birth of OpenTelemetry

- OpenTracing
  - API spec
  - semantic conventions
- OpenCensus
  - libraries to generate traces & metrics
  - collector to output different formats
- Supersedes two existing competing standards

OpenTelemetry

aims to standardize how applications are instrumented and how telemetry data is generated, collected, and transmitted

- An open specification
- Language-specific APIs and SDKs
- Instrumentation libraries
- Semantic conventions
- An agent to collect telemetry
- A protocol to organize, transmit, and receive the data

Collector

Vendor-agnostic way to receive, process and export telemetry data

Instrumenting

configure code to emit traces, metrics and logs

- automatic: quickly gain insights into your application
- manual: embed granular observability into your code
- possible to combine for most languages

Instrumenting

Language	Traces	Metrics	Logs
C++	Stable	Stable	Experimental
C#/.NET	Stable	Stable	Mixed*
Erlang/Elixir	Stable	Experimental	Experimental
Go	Stable	Beta	Not yet implemented
Java	Stable	Stable	Stable
JavaScript	Stable	Stable	Development
PHP	Beta	Beta	Alpha
Python	Stable	Stable	Experimental
Ruby	Stable	Not yet implemented	Not yet implemented
Rust	Beta	Alpha	Not yet implemented
Swift	Stable	Experimental	In development

Instrumenting

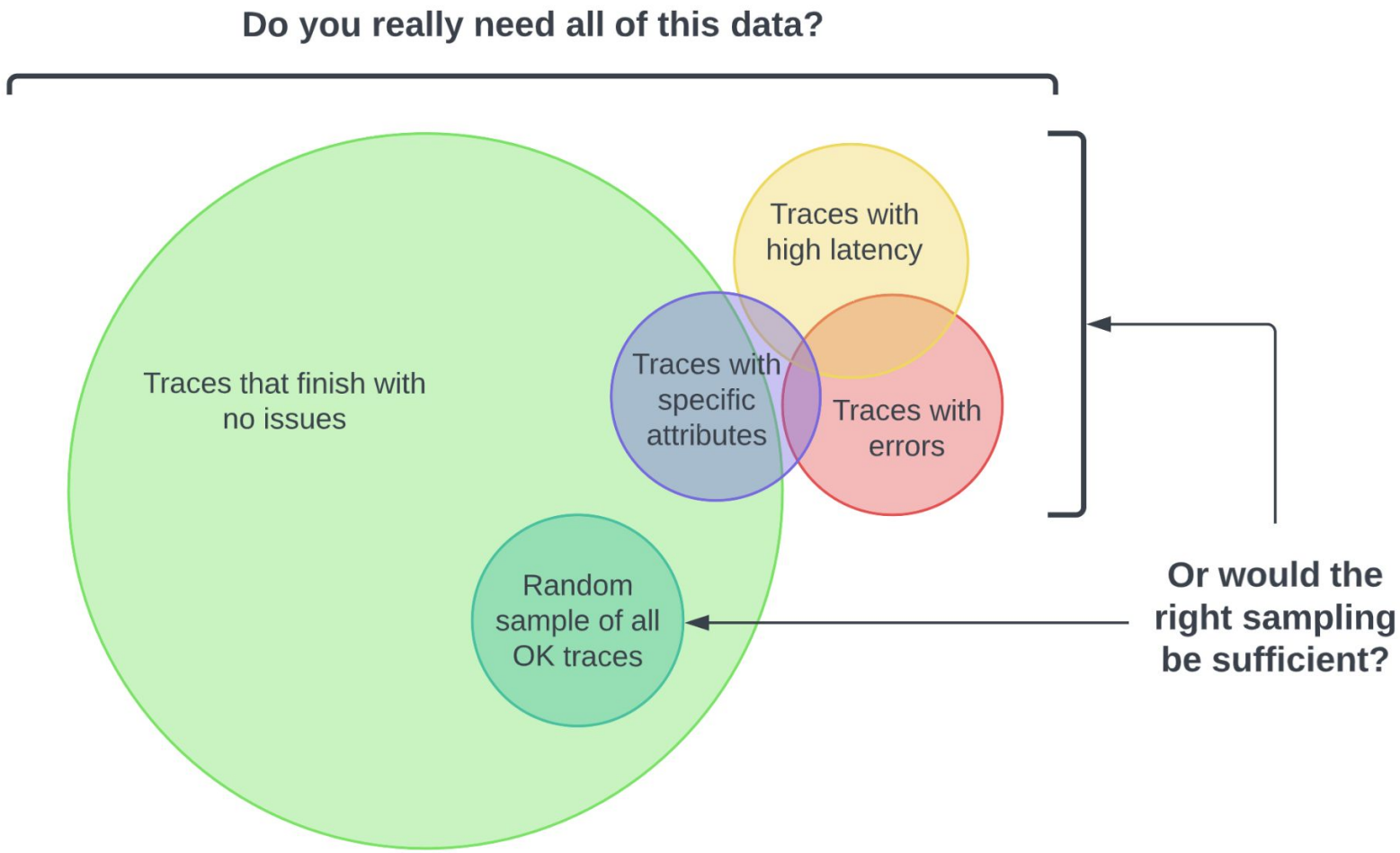
Demo

OpenTelemetry Operator

Kubernetes Operator that manages collectors and auto-instrumentation of your applications using OpenTelemetry instrumentation libraries.

- collector using Custom Resource
- auto-instrumentation using Custom Resource and annotations
  - Java
  - Python
  - .NET
  - Node.js

Sampling


Do you really need all of this data?
Traces that finish with no issues
Traces with high latency
Traces with specific attributes
Traces with errors
Random sample of all OK traces
Or would the right sampling be sufficient?

Sampling

- manage costs
- focus on interesting traces
- filter out noise

Sampling

Head Sampling

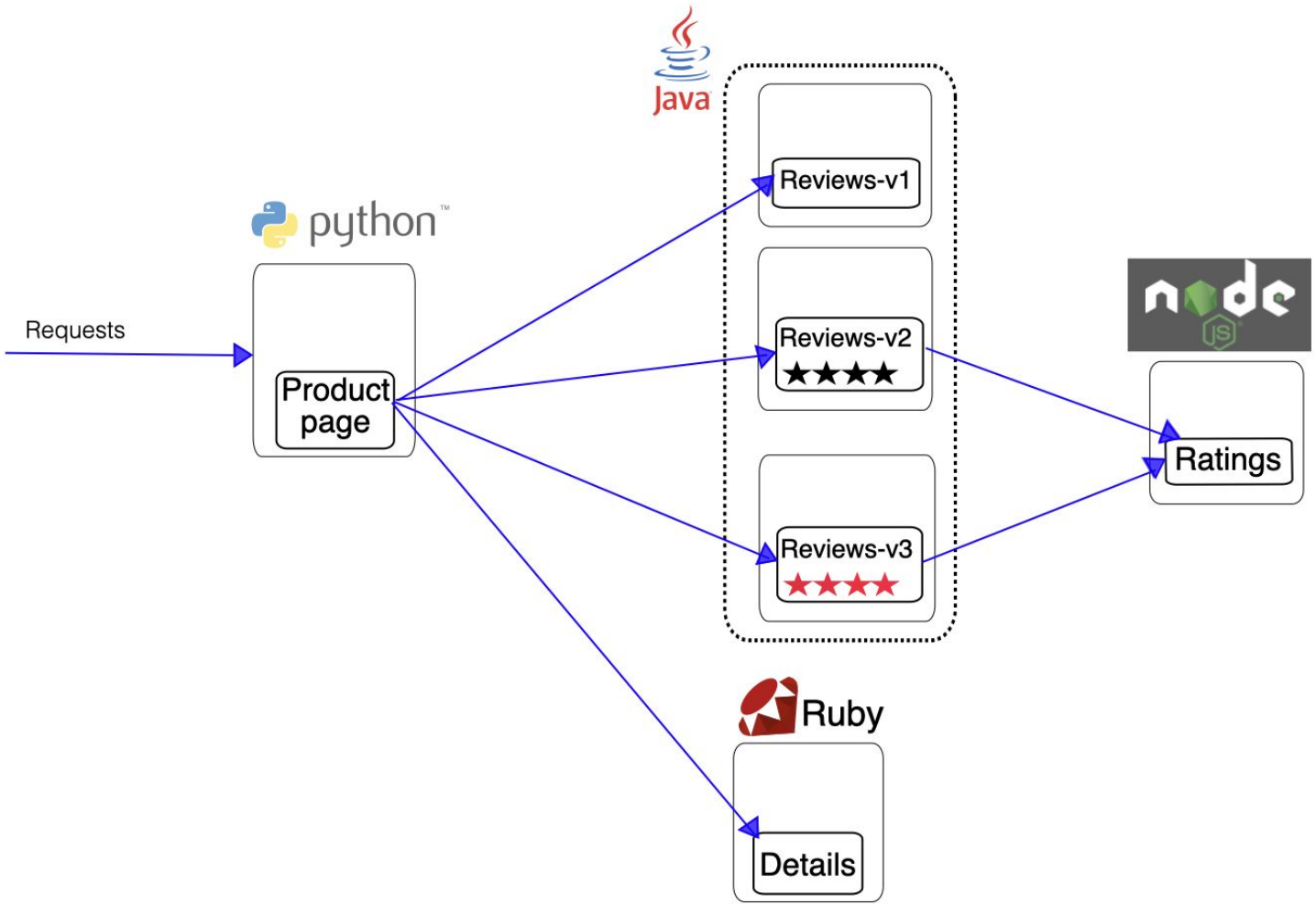
- decision is made as early as possible
- whole trace is sampled
- easy to understand and configure
- efficient

Tail Sampling

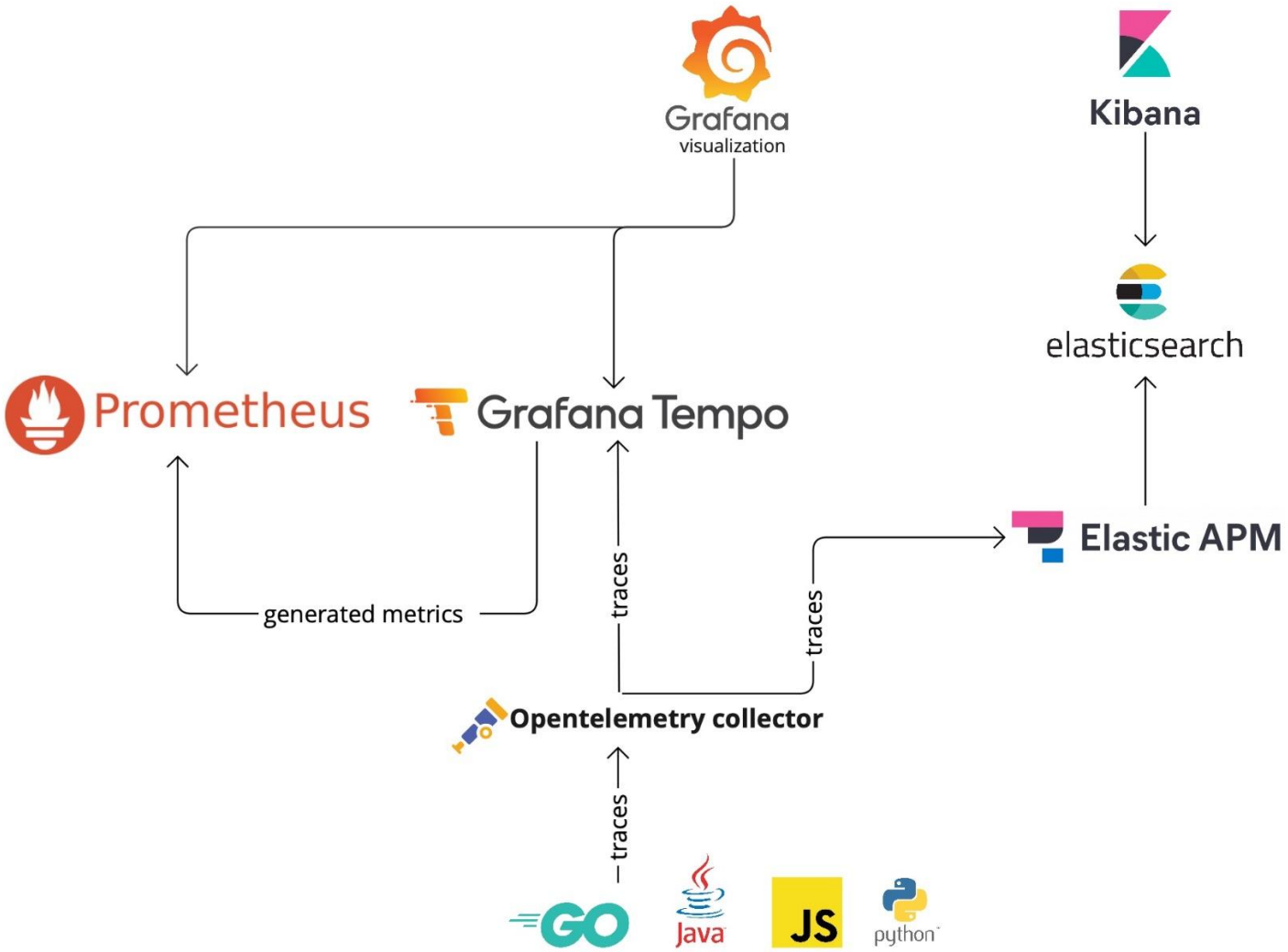
- decision takes place by considering all of the spans within the trace
- sample on specific criteria
- needed to keep the data useful
- (can be) difficult to implement
- domain of vendor-specific technology

Demo

Demo


Java
Reviews-v1
python
Requests
Reviews-v2
Product page
Ratings
Reviews-v3
Ruby
Details

Demo


Grafana
visualization
Kibana
elasticsearch
Prometheus
Grafana Tempo
Elastic APM
generated metrics
traces
traces
traces
Opentelemetry collector
GO
Java
JS
python
miro

kangaroot
Linux & Open Source Solutions

Questions?