



A practical introduction to Open Telemetry

 [@nicolas_frankel](https://twitter.com/nicolas_frankel)

In the good old days...



- Monitoring
- Lots of people looking at screens
- Alerting



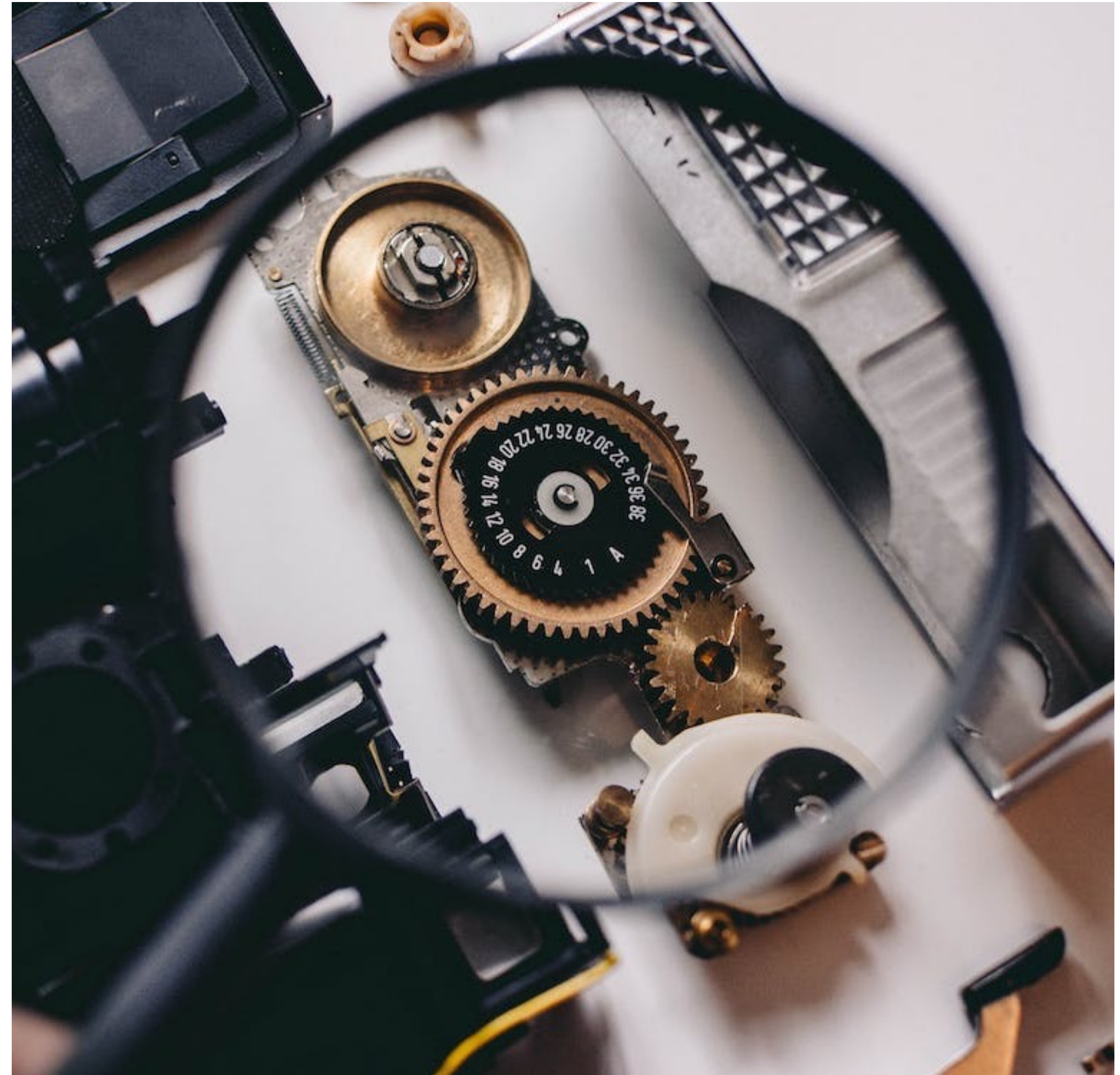
Then systems became more distributed



Observability

“In *distributed systems*, observability is the ability to collect data about program execution, internal states of modules, and communication between components. To improve observability, software engineers use a wide range of *logging* and *tracing* techniques and tools.”

-- https://en.wikipedia.org/wiki/Event_monitoring



The 3 pillars of Observability

1. Metrics
2. Logging
3. Tracing



Metrics



- System metrics
 - CPU, memory, etc.
- Higher-level metrics
 - Requests per second, HTTP status code, etc.



Logging



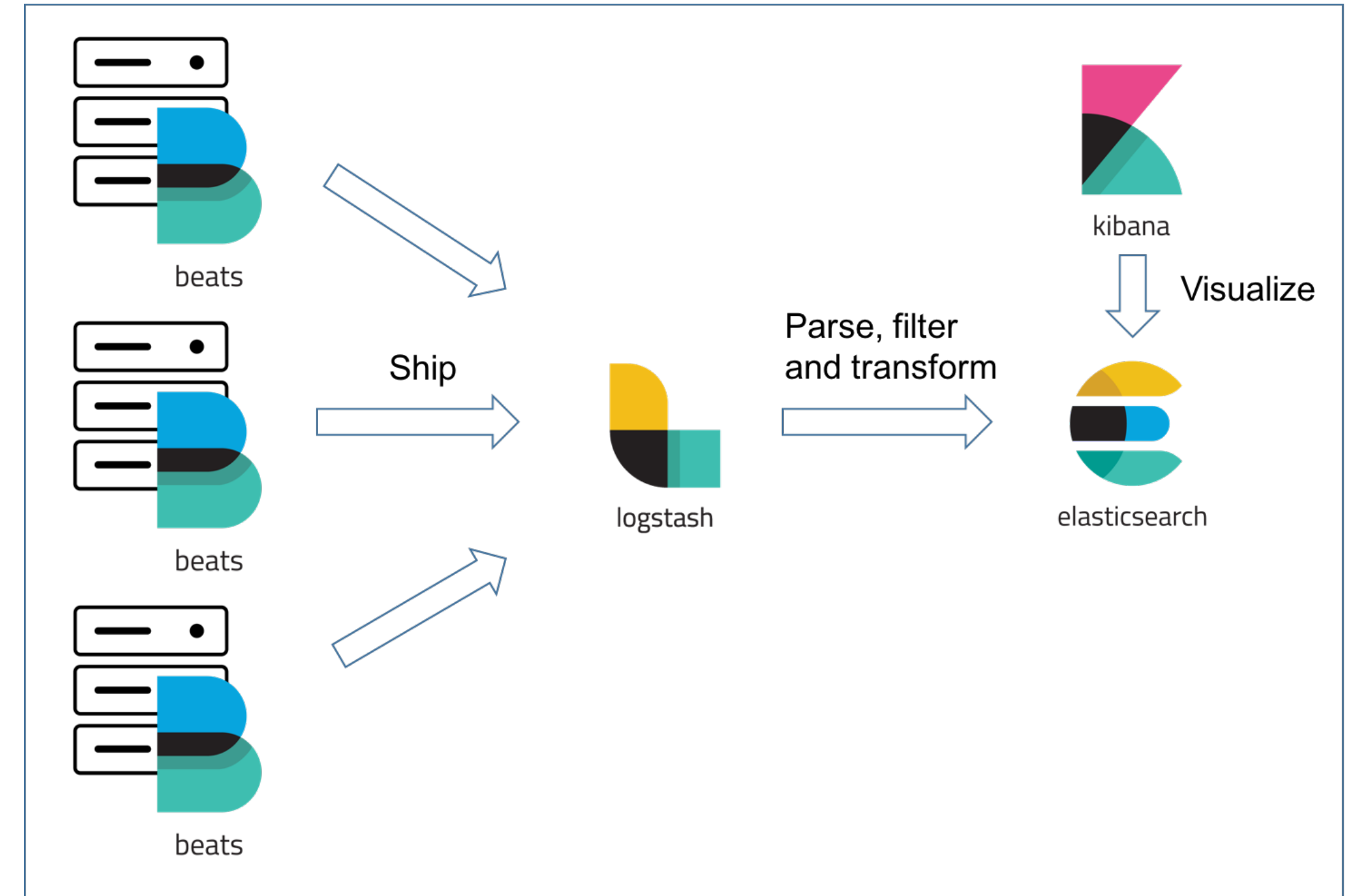
- What to log
 - Auto vs. manual
 - Sensitive data
- Logging format
 - Human readable vs. JSON
- Where to log
 - Console vs. log files
- Logs aggregation FTW



Centralized logging systems



- Get the log
 - Scrape vs. Send
- Parse the log
 - Structured vs. unstructured
- Store the log
- Search the log
- Display the log



Some centralized logging systems



Tracing



“In software engineering, tracing involves a specialized use of logging to record information about a program's execution. [...] Tracing is a cross-cutting concern.”

-- [https://en.wikipedia.org/wiki/Tracing_\(software\)](https://en.wikipedia.org/wiki/Tracing_(software))

 @nicolas_frankel



Tracing

“Set of techniques and tools that help follow a business request through multiple components across the network”

-- Me

(inspired by lots of others I don't remember the name of)



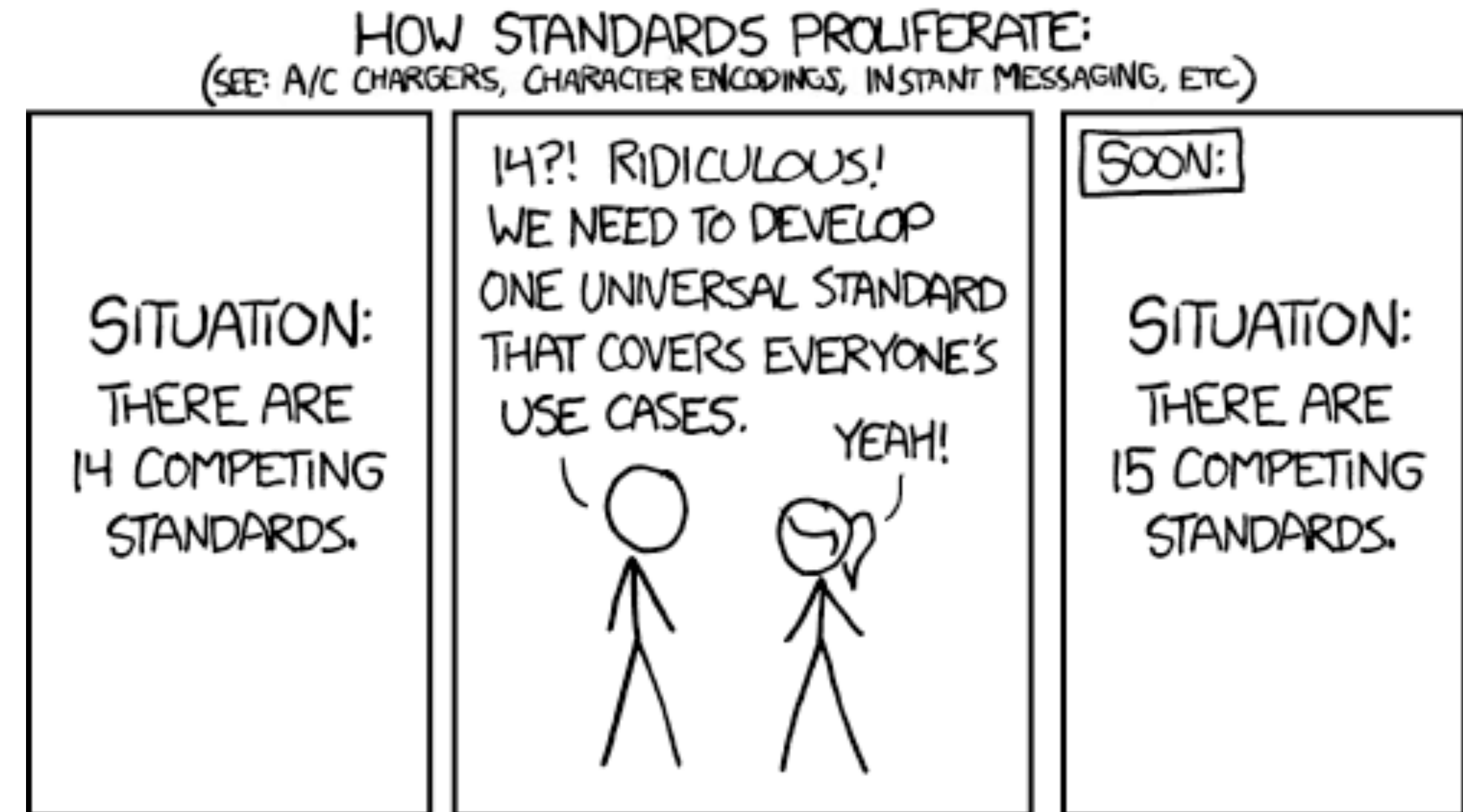
Tracing pioneers



JAEGER

The W3C Trace Context specification

“This specification defines standard HTTP headers and a value format to propagate context information that enables distributed tracing scenarios. The specification standardizes how context information is sent and modified between services. Context information uniquely identifies individual requests in a distributed system and also defines a means to add and propagate provider-specific context information.”



— <https://www.w3.org/TR/trace-context/>

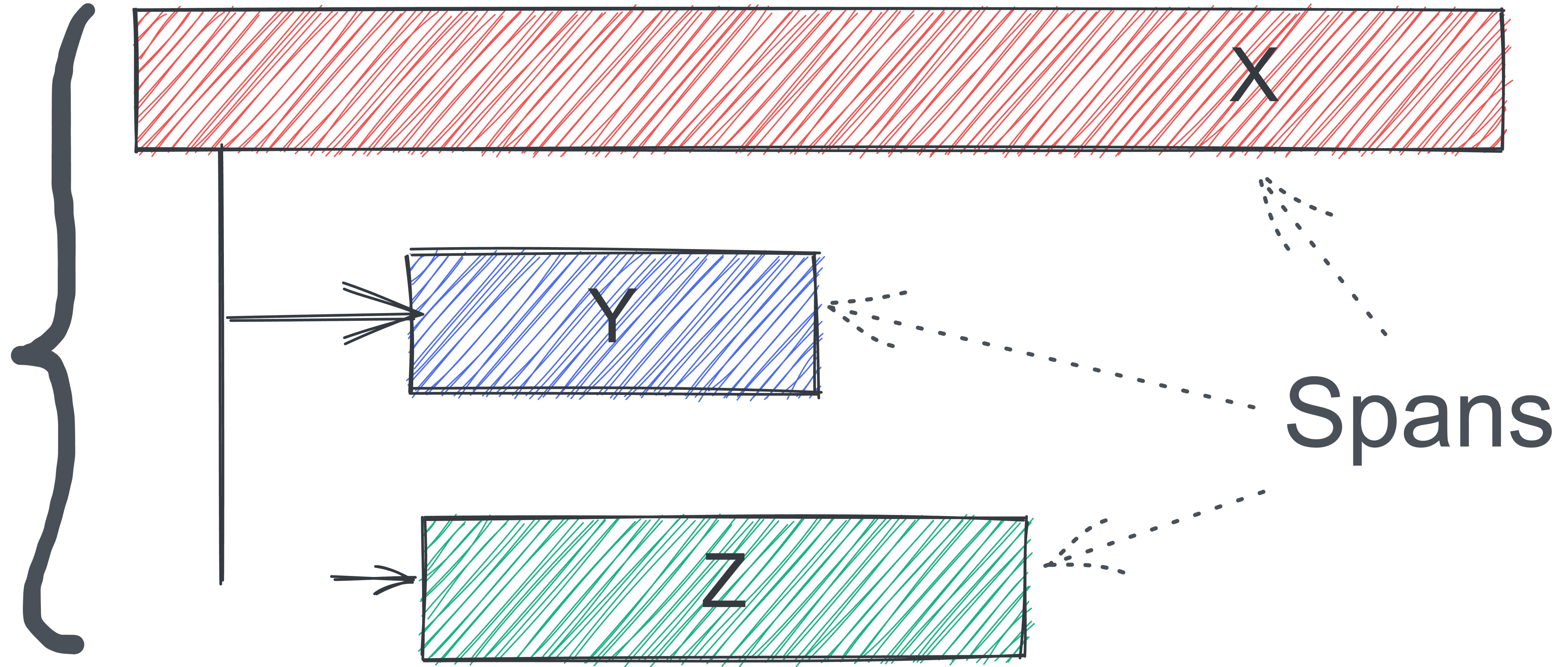
Base concepts



- Trace: follows the path of a request that spans multiple components
- Span: bound to a single component and linked to another span by a child-parent relationship



Single trace





OpenTelemetry

“OpenTelemetry is a collection of tools, APIs, and SDKs. Use it to instrument, generate, collect, and export telemetry data (metrics, logs, and traces) to help you analyze your software’s performance and behavior.”



-- <https://opentelemetry.io/>

 [@nicolas_frankel](https://twitter.com/nicolas_frankel)

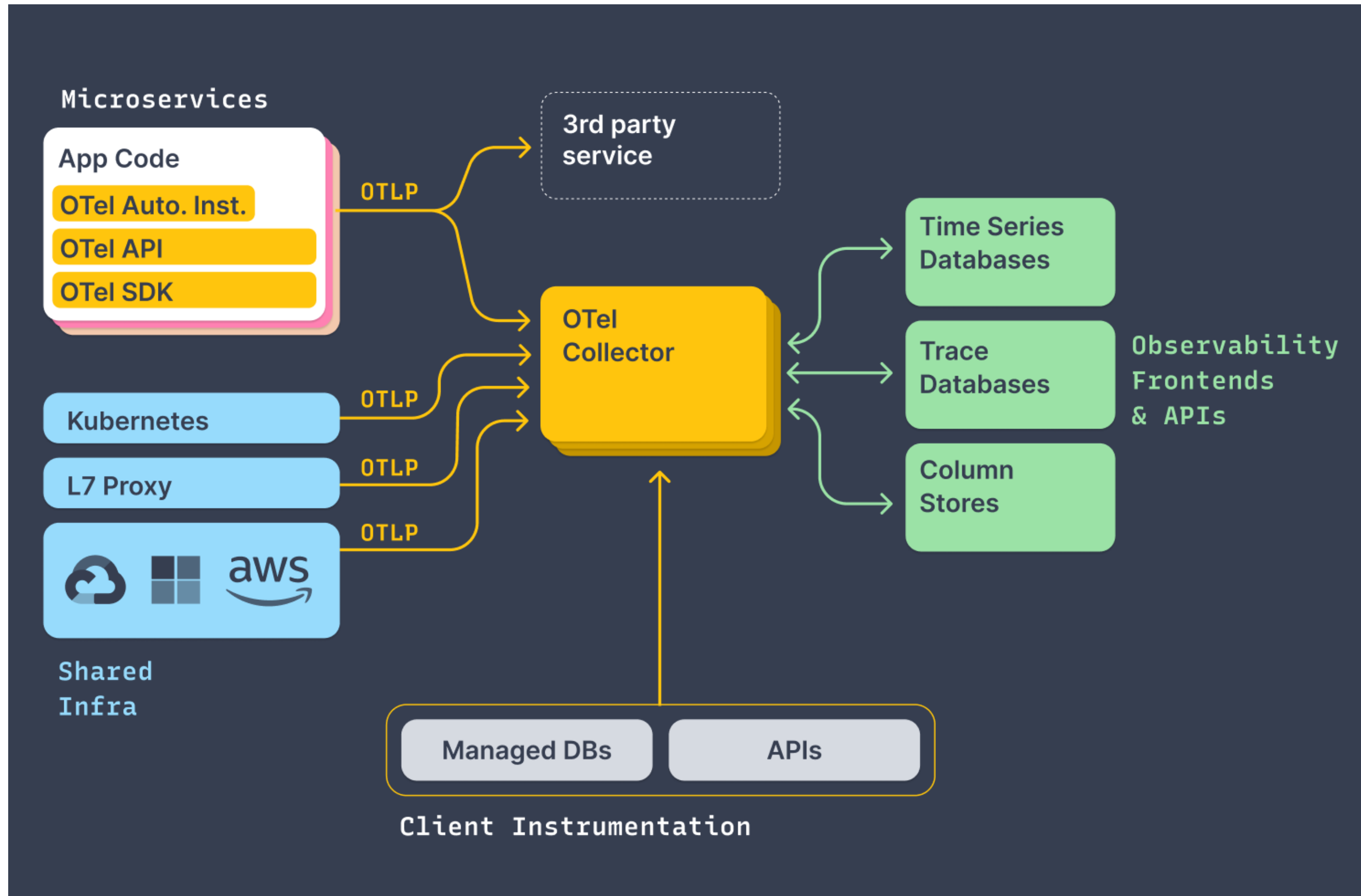
OpenTelemetry



- Implements W3C Trace Context
- Merge of OpenTracing and OpenCensus
- CNCF project
- Apache v2 license
- 1.3k followers on GitHub



OpenTelemetry architecture



Life after the OTEL collector

- OTEL provides a collector
- Jaeger and Zipkin provide compatible collectors
 - Continue using your existing tracing provider!



Auto-instrumentation vs. manual instrumentation

- Auto-instrumentation
 - Via the runtime
- Manual instrumentation
 - Library dependency + API

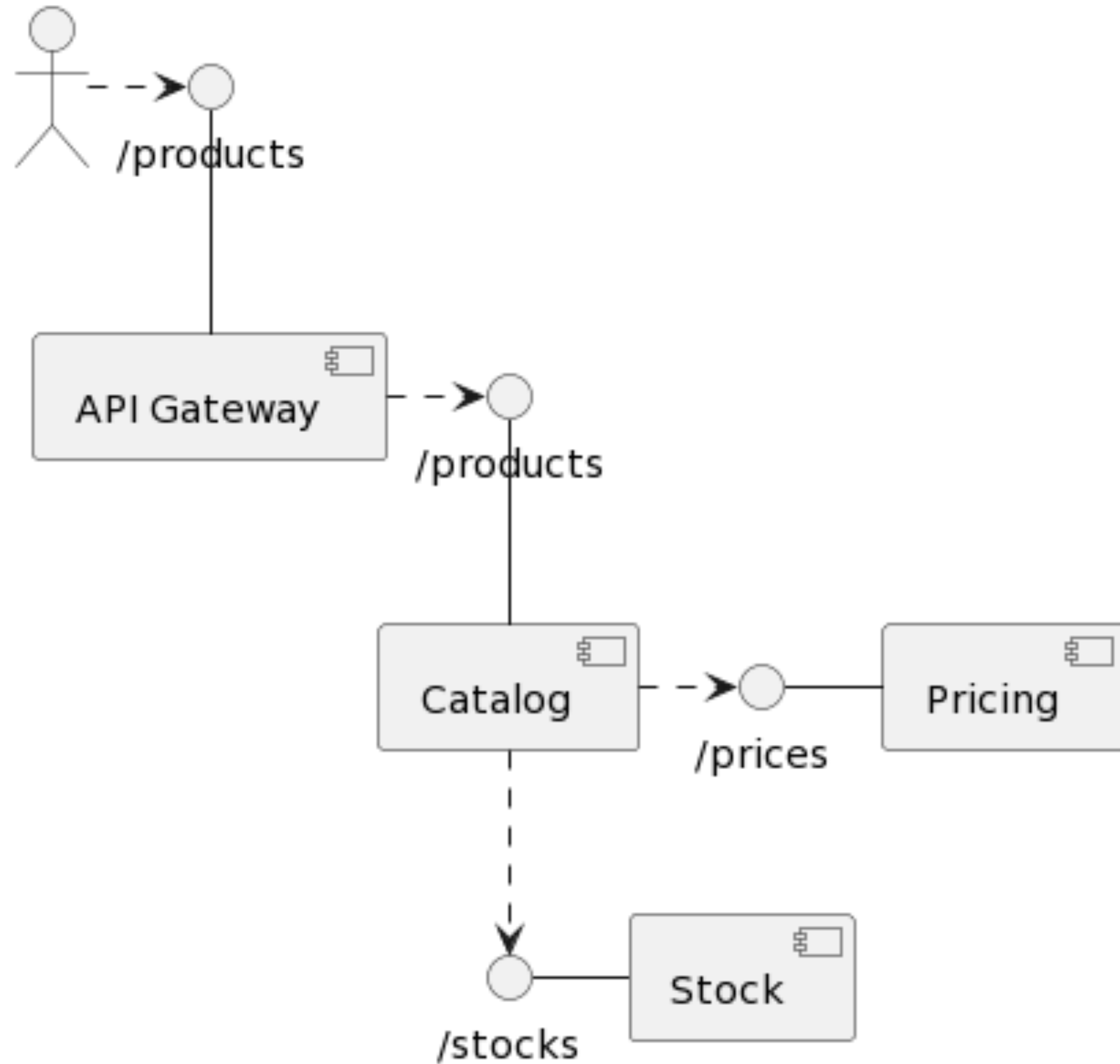


Benefits of auto-instrumentation



- Low-hanging fruit
- No coupling





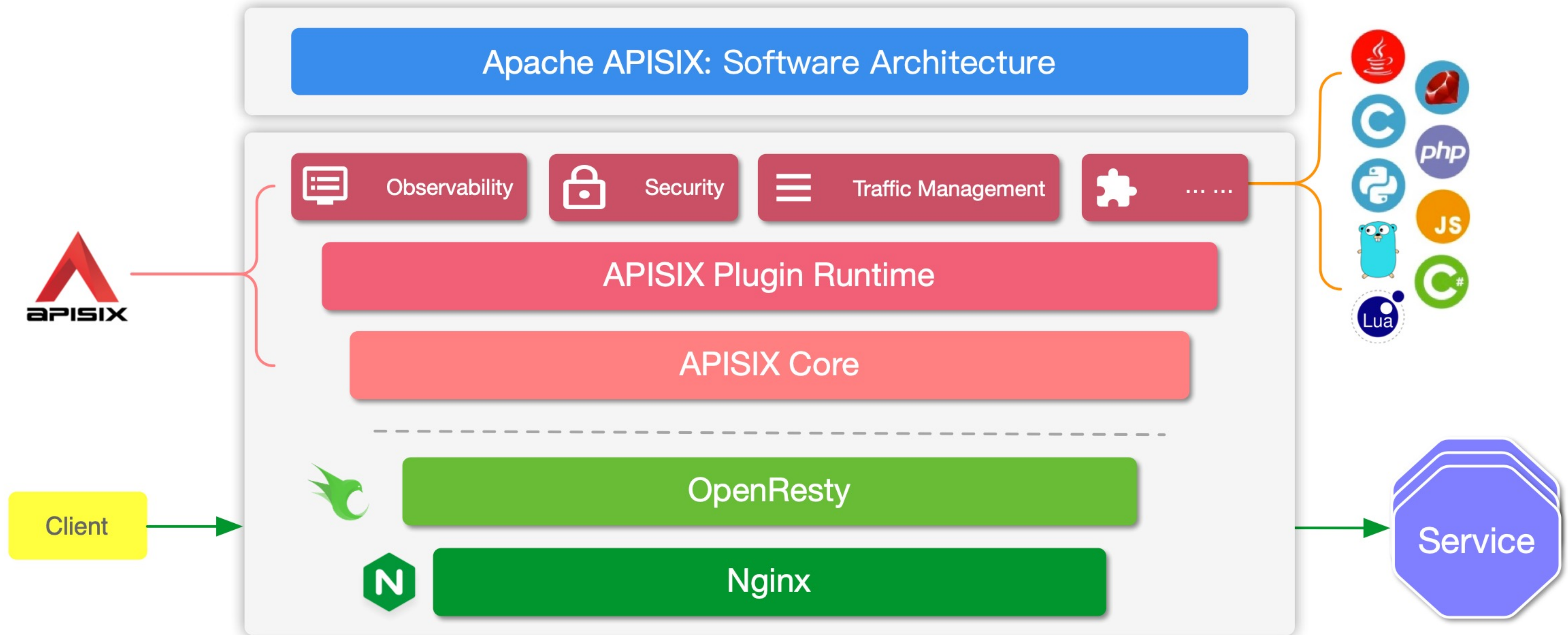
The entrypoint



- The most important part as it generates the first ID
 - Reverse proxy/API Gateway



Apache APISIX, an API Gateway the Apache way



General configuration



```
plugins:  
  - opentelemetry  
plugin_attr:  
  opentelemetry:  
    resource:  
      service.name: APISIX  
  collector:  
    address: jaeger:4318
```





Per-route (or global rule) configuration

```
plugins:  
  opentelemetry:  
    sampler:  
      name: always_on  
    additional_attributes:  
      - route_id  
      - request_method  
      - http_x-ot-key
```



JVM auto-instrumentation implementation



- Via a Java agent:
 - `-javaagent:otel.jar`
- Regardless of:
 - The language
 - The framework





JVM explicit instrumentation

- Requires the OTEL dependency
- Usage:
 - Annotations
 - API call

Annotations

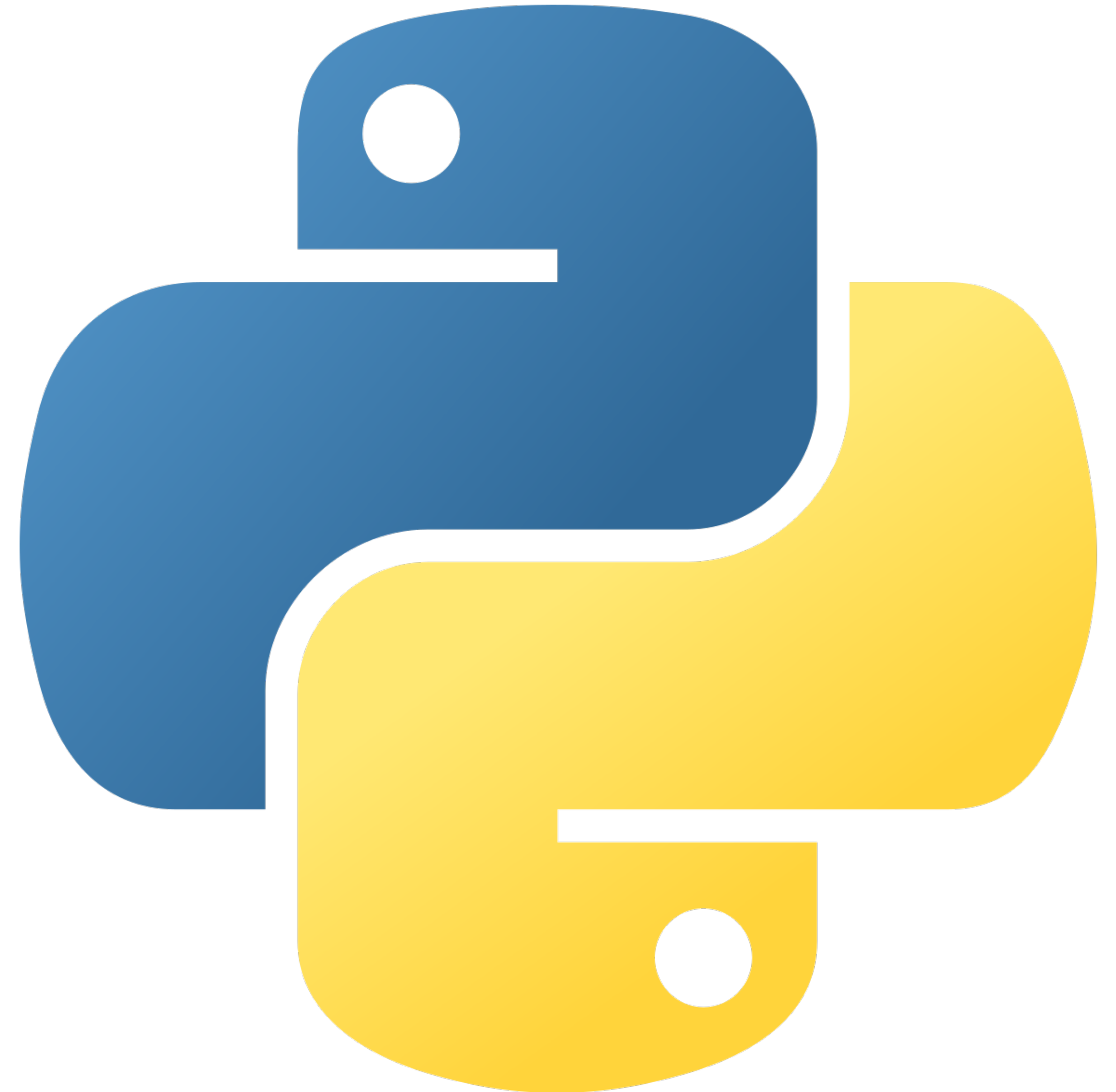


```
@WithSpan("ProductHandler.fetch")
private suspend fun fetchProductDetails(
    @SpanAttribute("id") id: Long,
    product: Product) {
    // ...
}
```

Python auto-instrumentation

- Add the OTEL dependency
- Run with the instrumentation:

```
>opentelemetry-instrument flask run
```



Explicit API



```
from opentelemetry import trace

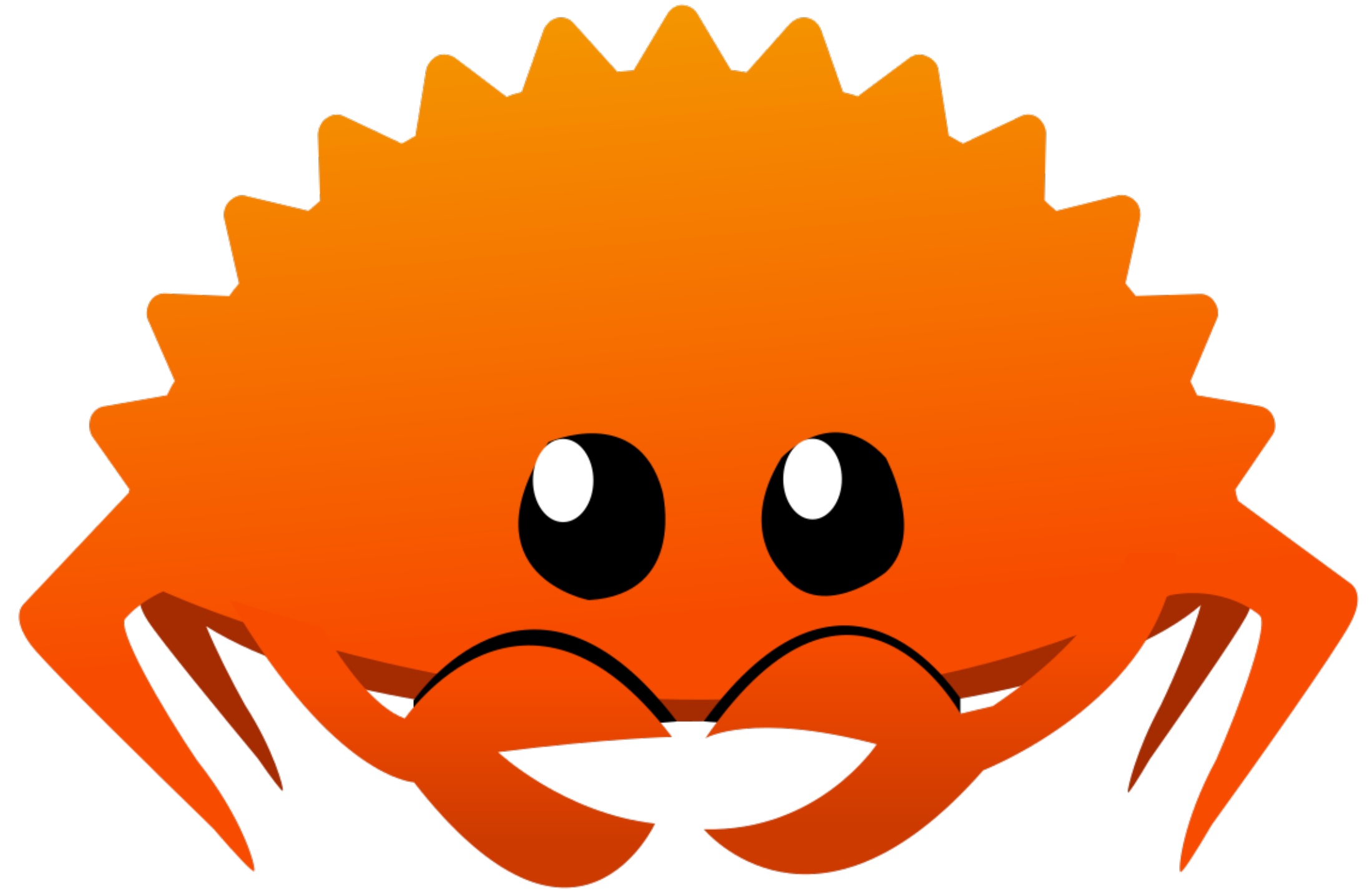
tracer = trace.get_tracer(__name__)

with tracer.start_as_current_span(
    "SELECT * FROM PRICE WHERE ID=:id",
    attributes={"id": 1}):
    #do under the span
```

Rust



- Rust compiles to native:
 - No runtime
 - Needs explicit calls





Finding the relevant Cargo dependency

- It's not trivial!

```
axum-tracing-opentelemetry = { version =  
"0.7", features = ["otlp"] }
```


Configure axum



```
let app = axum::Router::new()
    .route("/stocks/:id", get(get_by_id))
    .layer(response_with_trace_layer())
    .layer(opentelemetry_tracing_layer());
```


Thanks for your attention!

- @nicolas_frankel
- @nico@frankel.ch
- <https://bit.ly/otel-demo>
- <https://apisix.apache.org/>

