SKILUP SM DAYS

by: DevOps Institute
ADVANCING THE HUMANS OF DEVOPS

The Subtle Art of Distributed Tracing

Michele Mancioppi
Technical Product Manager
Instana
@mmanciop

Agenda

- Distributed Tracing in a Nutshell
- How Distributed Tracing Works
- What's in a Distributed Tracing Solution
- Lessons Learned building Instana

Michele Mancioppi

Technical Product Manager

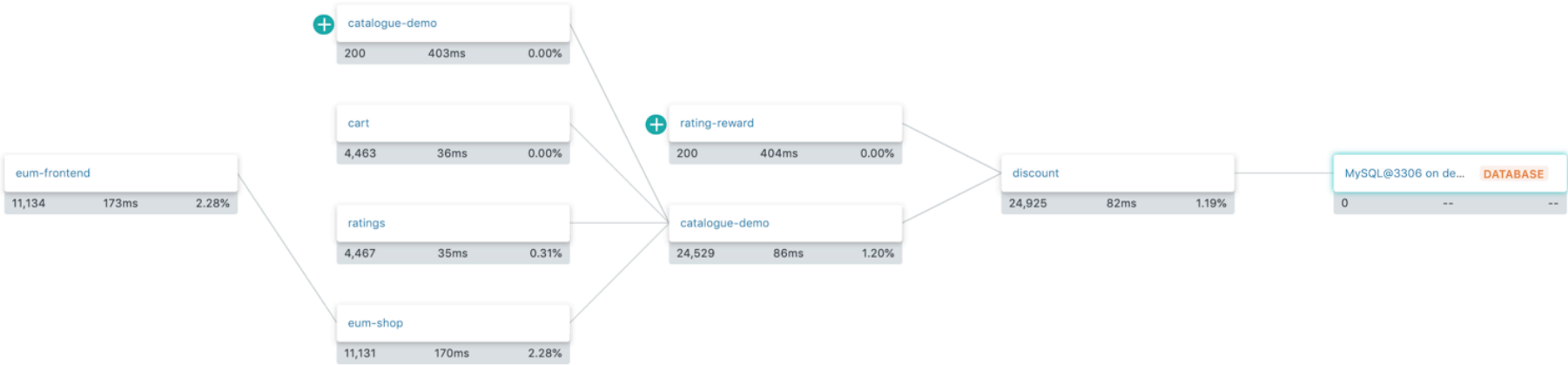
@mmanciop

Michele Mancioppi serves as Product Manager for agent, distributed tracing, Cloud Foundry and VMware Tanzu. Michele holds a Bachelors and Masters degree in Computer science from Università di Trento and a PhD in Information Systems from Tilburg University, the Netherlands.

SKILup DAYS

Everyday in microservices land

An everyday microservice

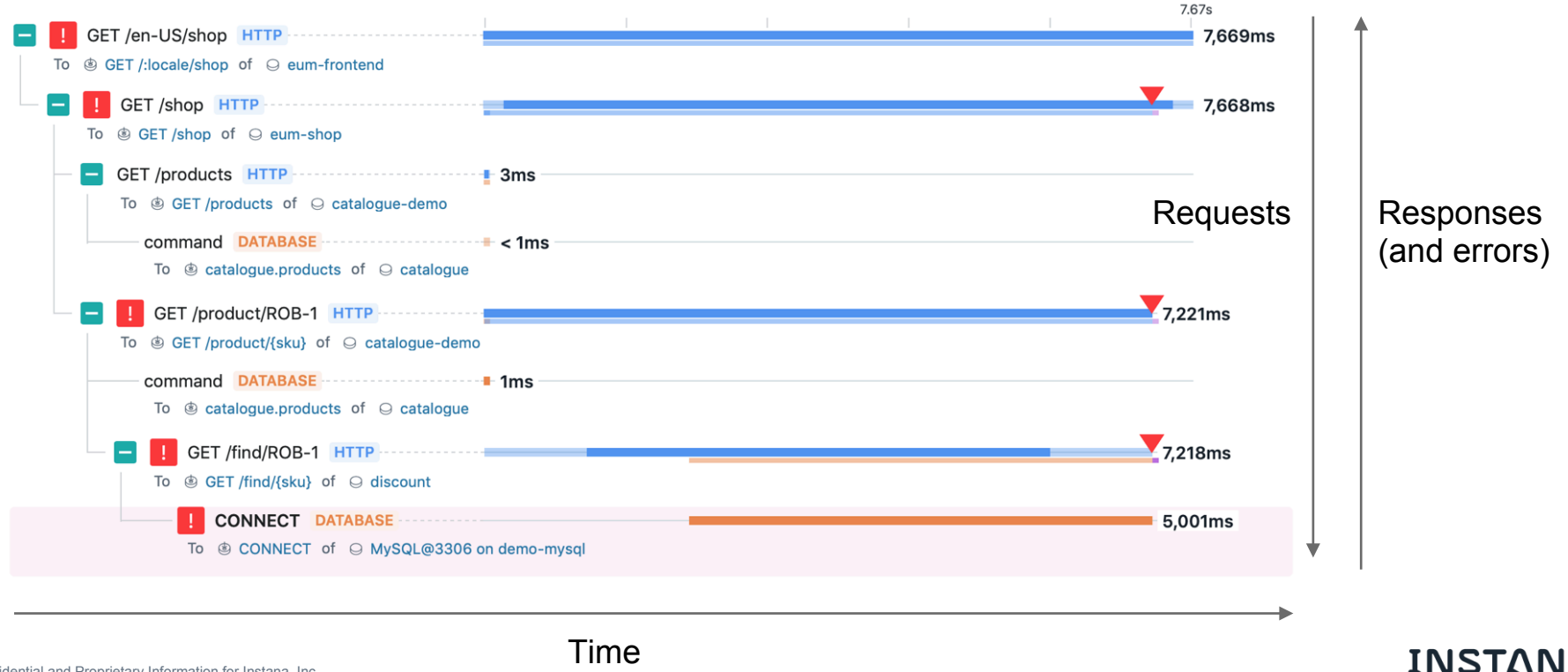


Requests

Responses
(and errors)

INSTANA

What is the origin of the issue?



INSTANA

Distributed Tracing in a Nutshell

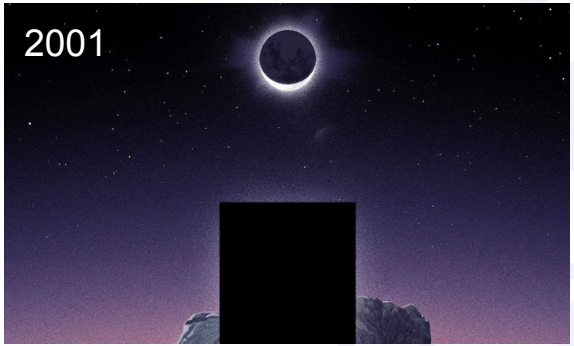
Distributed Tracing in a Nutshell

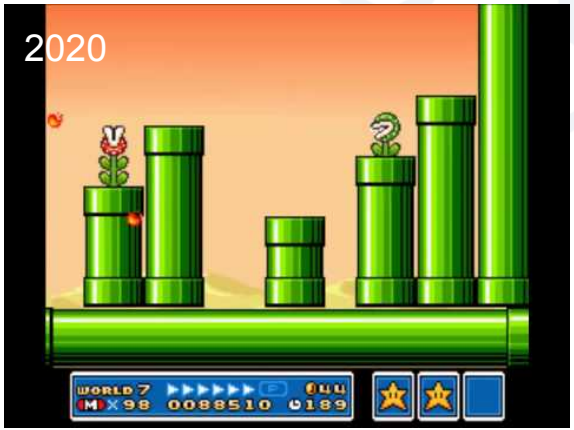
Distributed tracing is the practice of automatically collecting data about:

1. what one or more systems ...
2. … perform individually and in concert ...
3. … to serve a request

Why Distributed Tracing Matters

- Distributed complexity is increasing:
  - At network level
  - At versioning level
- Distributed complexity is:
  - Often harder to debug than monolithic complexity
  - Requires more people to be involved

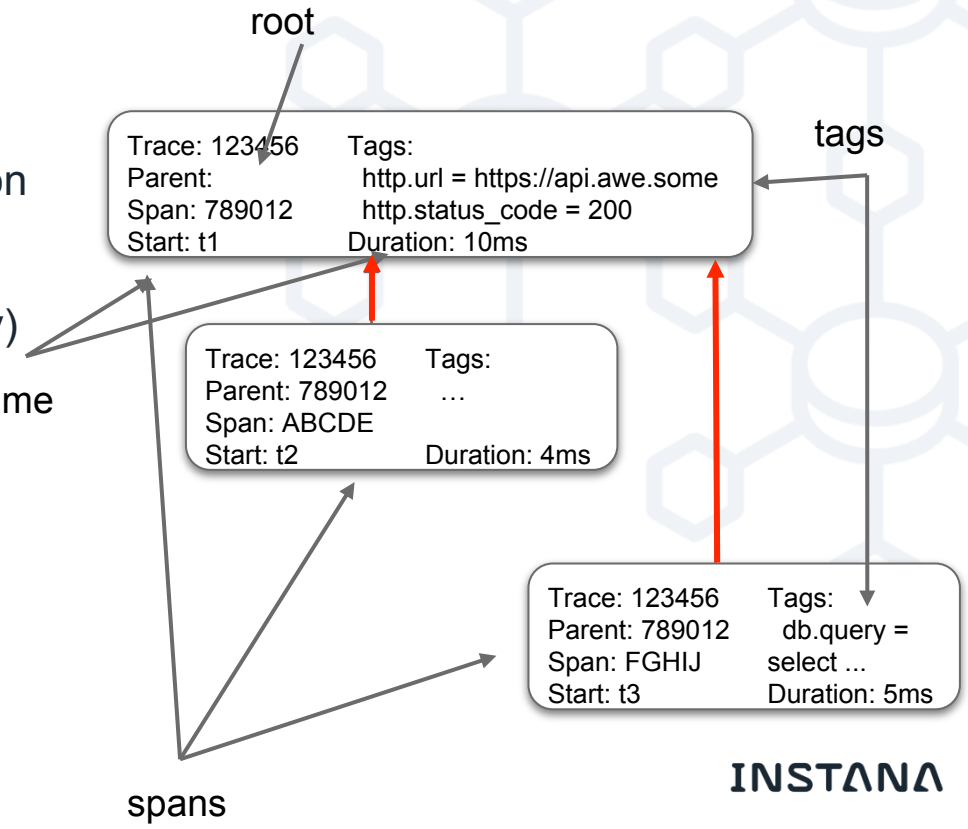




How Distributed Tracing Works

Traces, Spans and Tags

- A trace is a group of spans
- Spans are:
  - Logs with a start time and a duration
  - Related to one another via parent-child relationships (often 1-to-many)
    - The root span has no parent
  - Annotated with tags like `http.status_code`

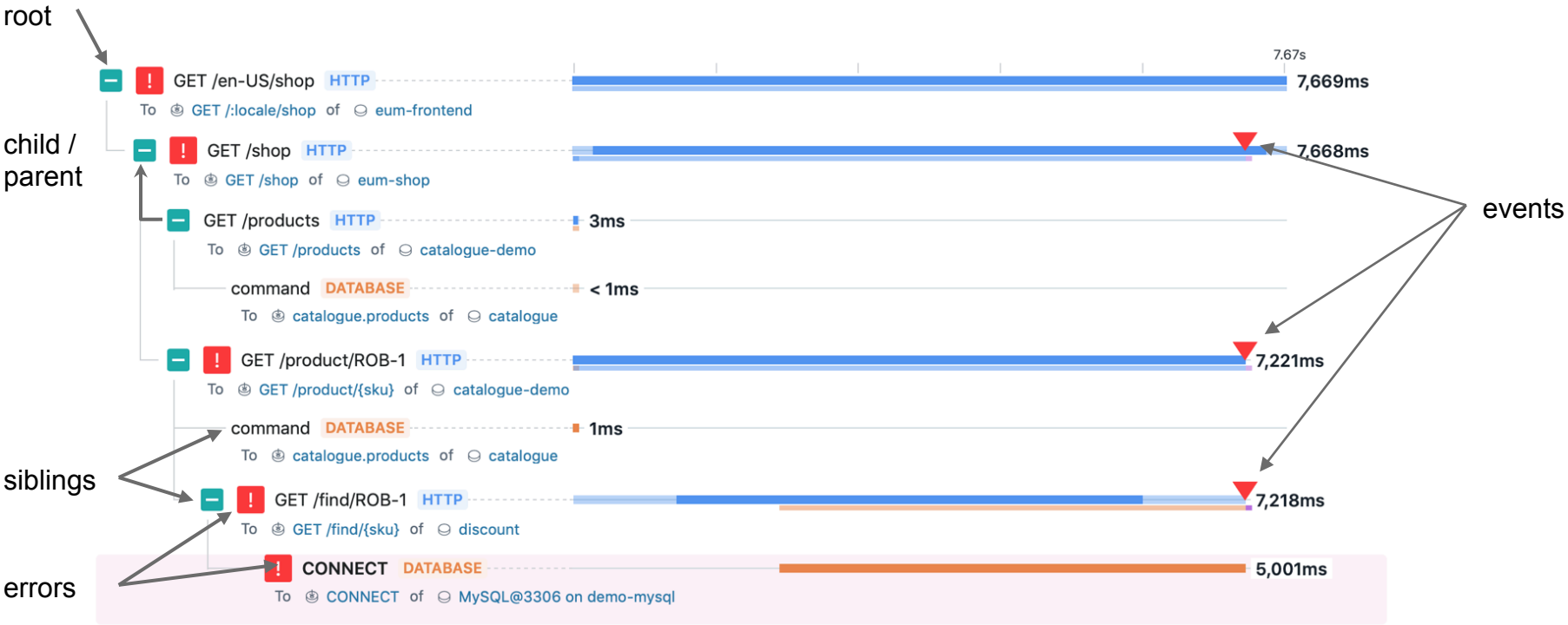


INSTANA

Trace Context

- Trace context keeps track of:
  - Trace
  - Current span
- Trace context is updated within a process with the latest span
- Trace context is propagated over messages, e.g.:
  - HTTP Headers
  - JMS metadata
  - gRPC metadata

Anatomy of a trace



INSTANA

Trace Analysis

Baselining

- Rate or requests / Flow
- Response Time
- Errors
- Structure

Troubleshooting

- What (happened)
- Where (the error occurred)
- When (the error happened)
- Who (was affected)

Alerting

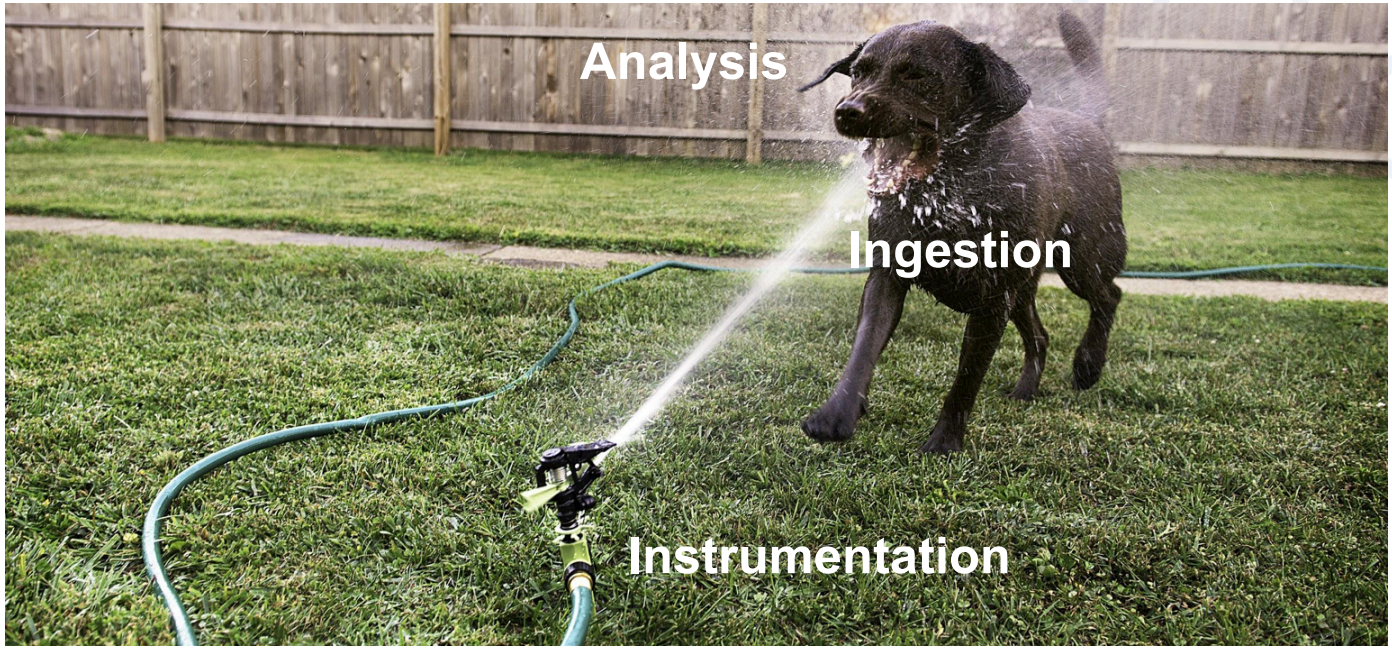
- Relevance
- Root cause

System exploration

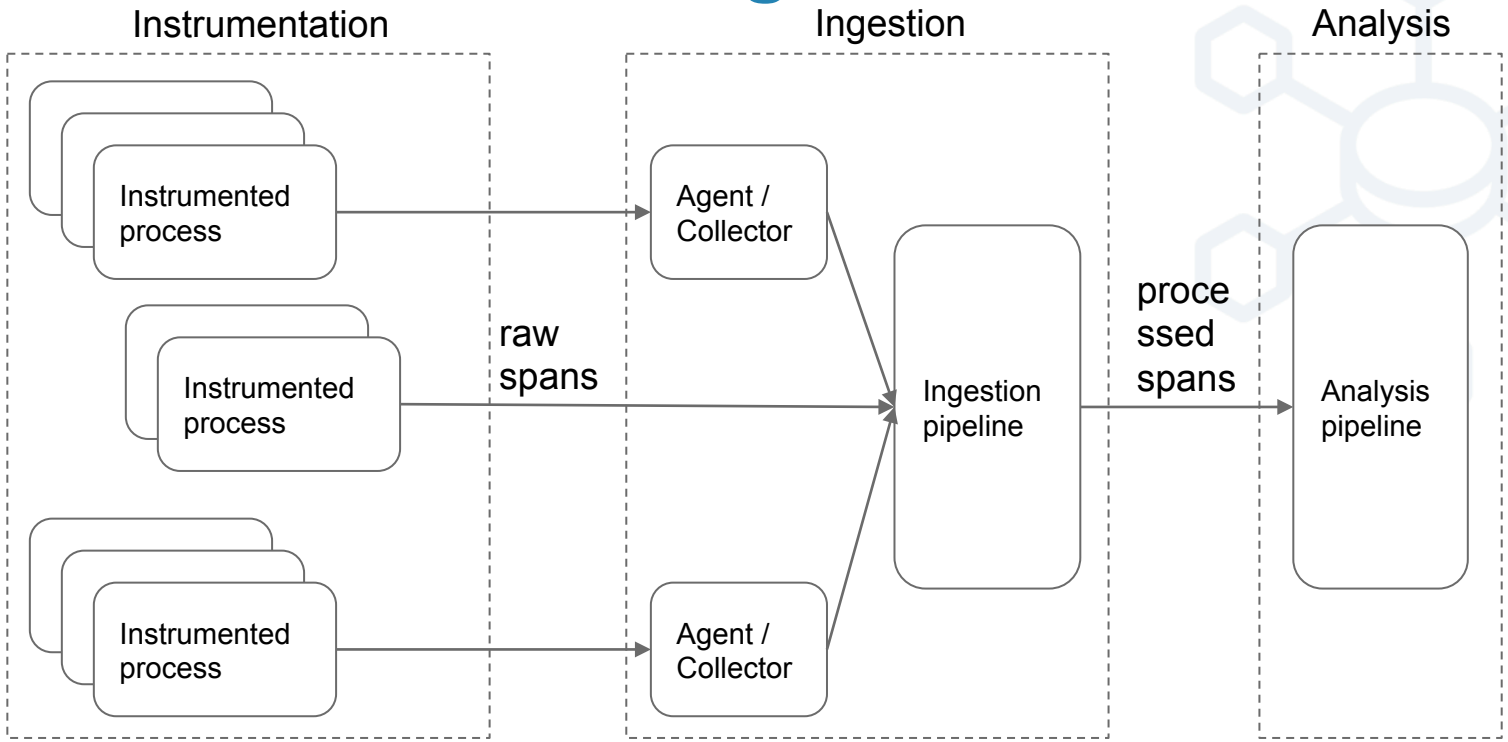
- “De-facto” architecture mapping

What’s in a Distributed Tracing Solution

What's in a Distributed Tracing Solution?



What's in a Distributed Tracing Solution?



INSTANA

Instrumentation

Instrumentation is code to produce tracing data at runtime

1. Instrumentation logic:
    a. Programmatic
    b. Automatic
2. Instrumentation delivery:
    a. Built-in
    b. Drop-in

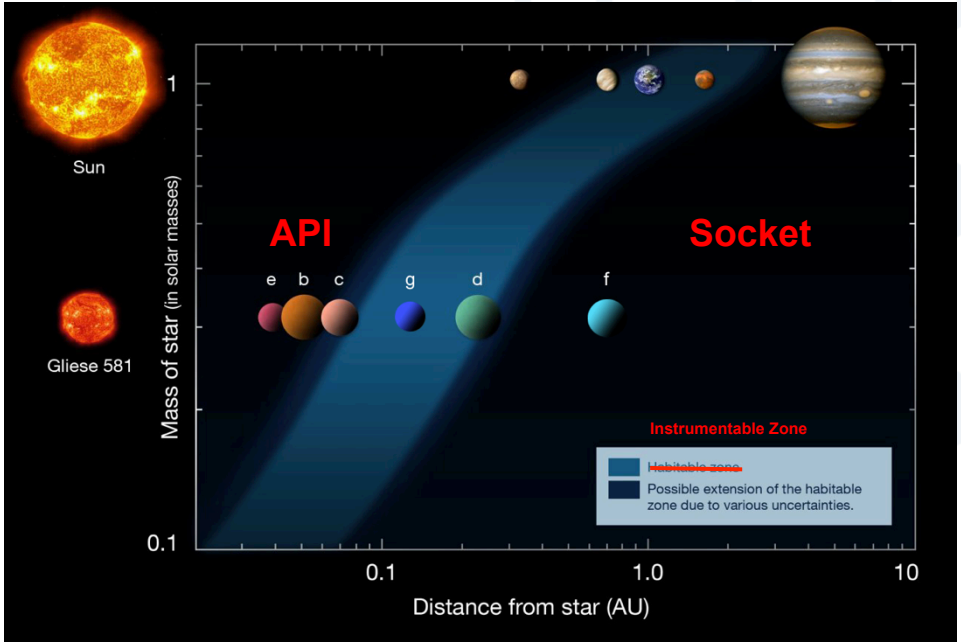
```
with ot.tracer.start_active_span('rabbitmq', child_of=tscope.span,
    tags={
        'exchange': Publisher.EXCHANGE,
        'sort': 'publish',
        'address': Publisher.HOST,
        'key': Publisher.ROUTING_KEY
    }) as scope:

    headers = {}
    ot.tracer.inject(scope.span.context, ot.Format.HTTP_HEADERS, headers)
    app.logger.info('msg headers {}'.format(headers))
    publisher.publish(order, headers)
```

Precision: Where to instrument?

Close to the API vs close to wire ?

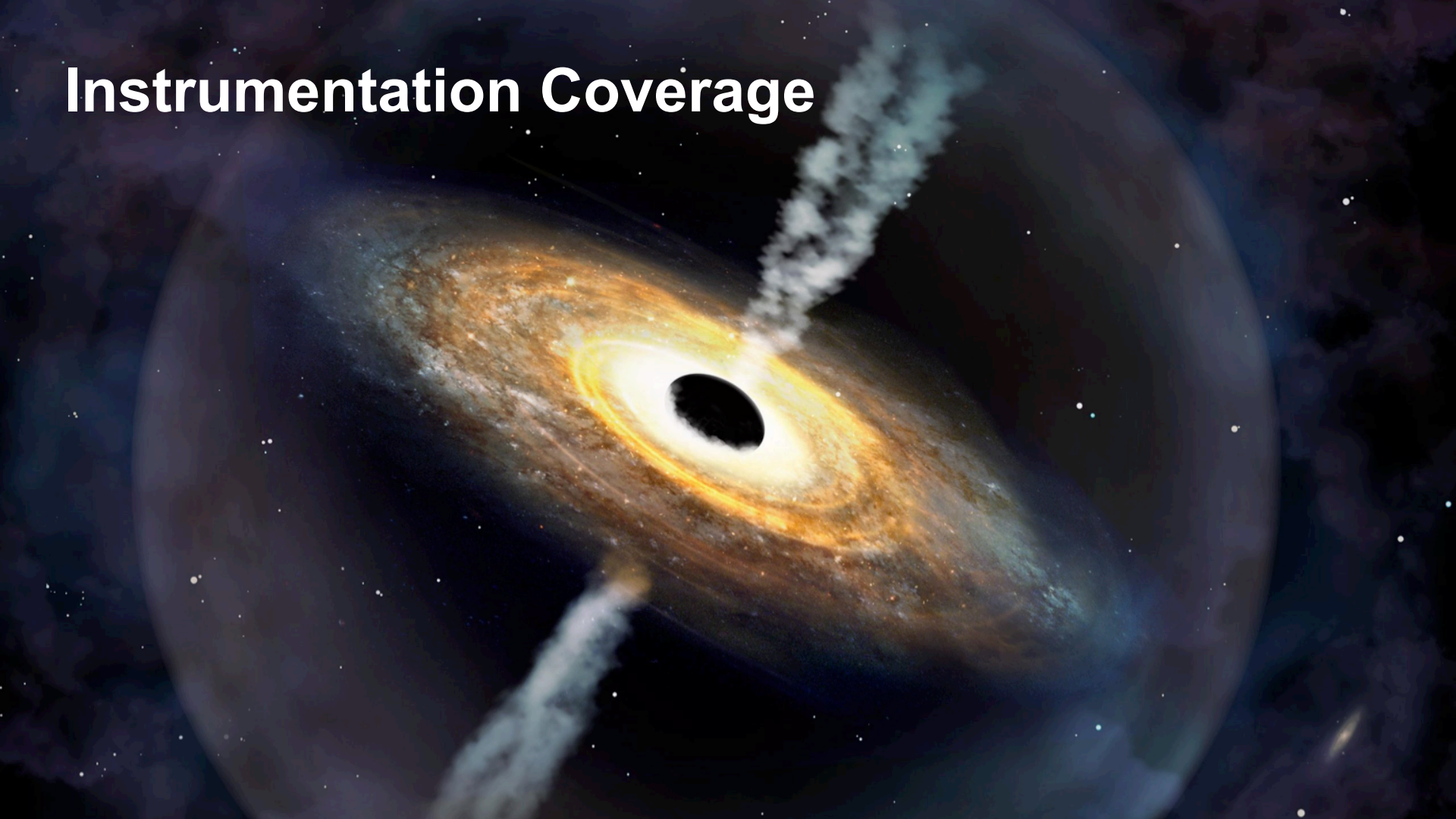
Goldilocks zone of instrumentation:

- Client errors
- Stacktraces
- Network timing
- Retries



INSTANA

Instrumentation Coverage



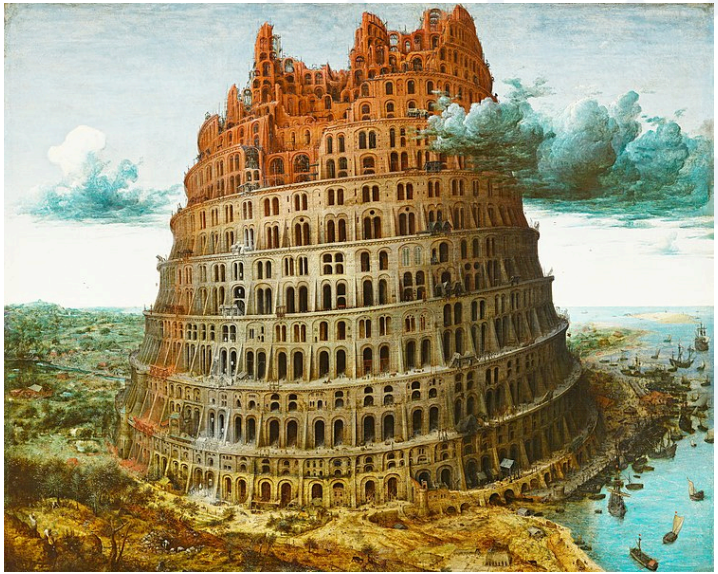
Sampling

Trace Completeness

Many Distributed Trace Contexts

Same concept of trace context, many incompatible implementations:

- B3: Zipkin, Cloud Foundry, Linkerd
- Jaeger
- Cloud Providers
  - X-Amzn-Trace-Id
  - X-Cloud-Trace-Context
- W3C Trace Context!



Metadata Consistency

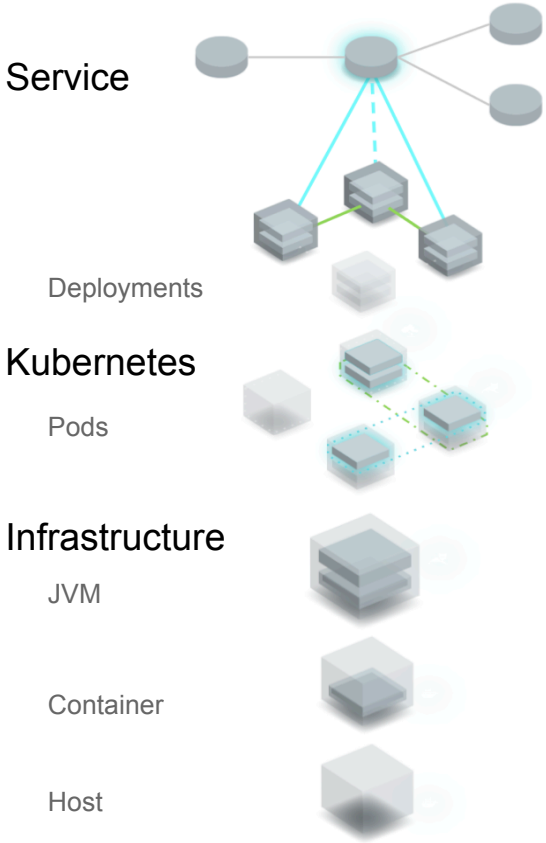
Tags have *semantics* for analysis

- Different names, same thing
  - Fewer hits in Analysis
  - More complex queries
- Same name, different things
  - False positive / negatives
  - Much harder to troubleshoot complex issues

Contextual Knowledge

Where is the span coming from

- Same software, multiple stacks and deployments
  - Dev, QA, Prod
  - Multi-cloud
- Different software versions, same stack
  - A/B testing
  - Red/Black deployment



Instrumentation Overhead

Direct overhead:

- In-process latency
- Network bandwidth

Indirect overhead:

- Garbage collection
- Preventing optimization

Lessons Learned building Instana

Primum non nocere

- Flush tracing data often
- Better throw away (some) data than affect the app
- Be ***very*** wary of introducing dependencies

Some things are better not traced

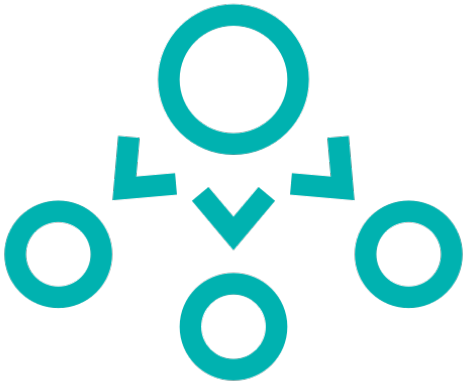
Clock skew


There is always another HTTP client

Summary

Key takeaways

1. Distributed tracing is an awesome observability tool
2. Doing distributed tracing is *FUN!*
3. Doing distributed tracing well is *HARD!*
4. … maybe you should let others provide it for you



THANK YOU!

Meet Me in the Network Chat Lounge for Questions

SKILUP DAYS