Conquering Microservices Complexity @Uber

With Distributed Tracing

Yuri Shkuro

SOFTWARE ENGINEER @ UBER

Agenda

- Why Distributed Tracing
- Trace as a Narrative
- Trace vs. Trace
- Traces vs. Trace
- Data Lineage
- Q & A

Yuri Shkuro

Software Engineer
Uber Technologies

shkuro.com

Founder & Maintainer
of CNCF Jaeger

jaegertracing.io

Co-founder of
OpenTracing &
OpenTelemetry

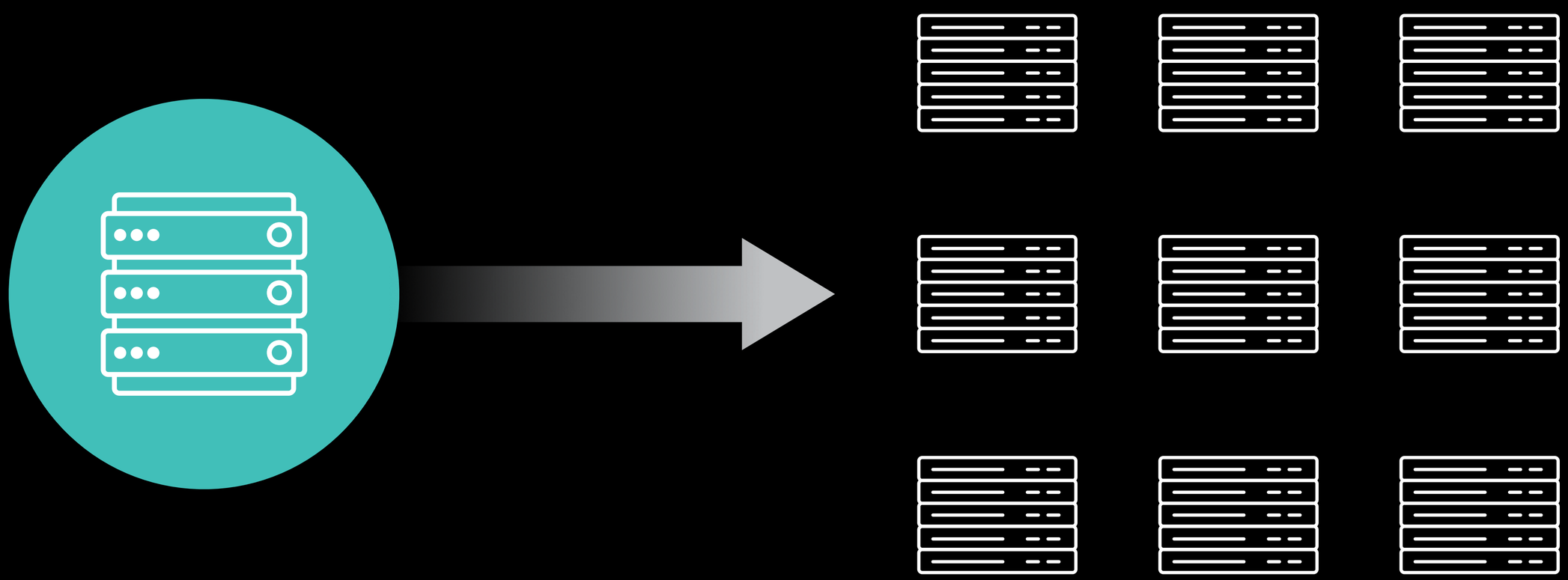
Author of "Mastering Distributed Tracing",
by Packt Publishing

Quick Poll

Why Distributed Tracing

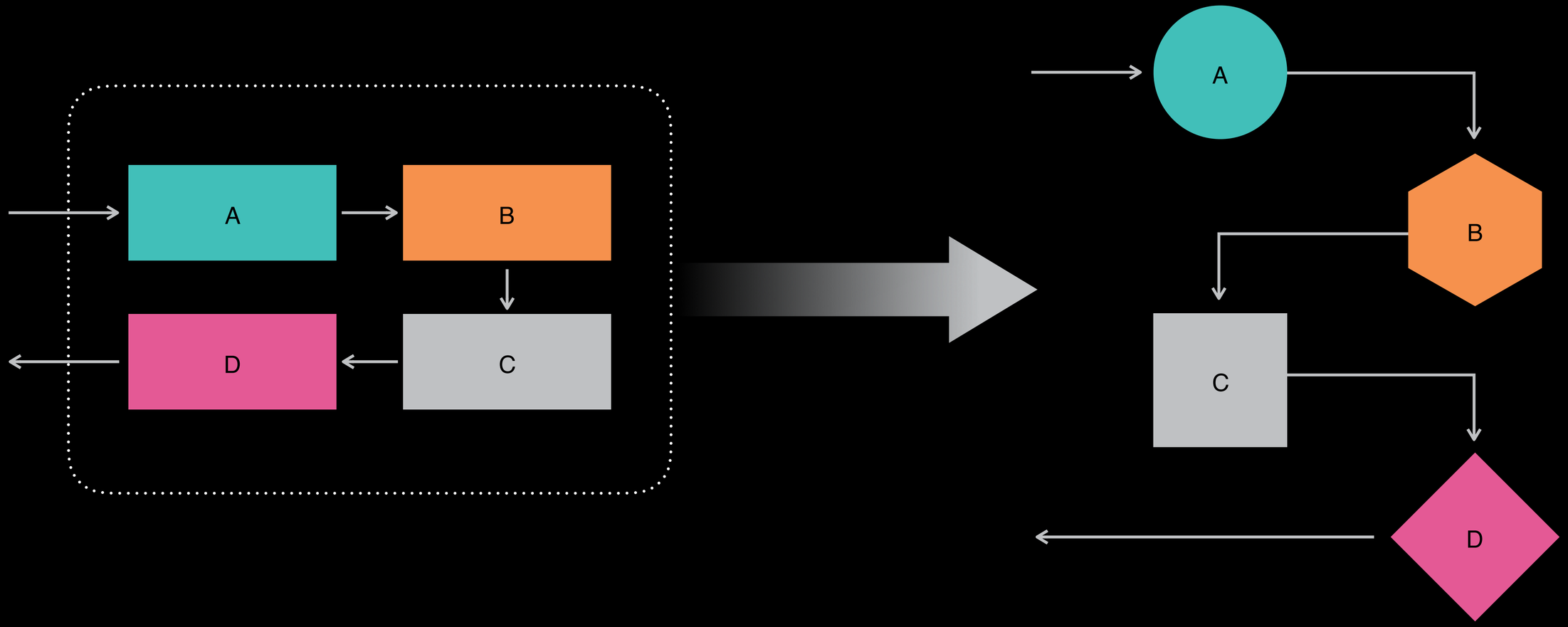
Scaling With Users

Distributed Systems



Scaling With Engineering Organization

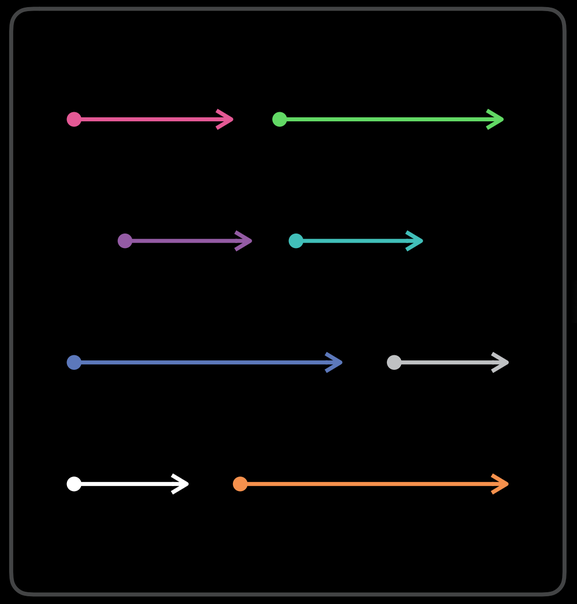
Monoliths to Microservices



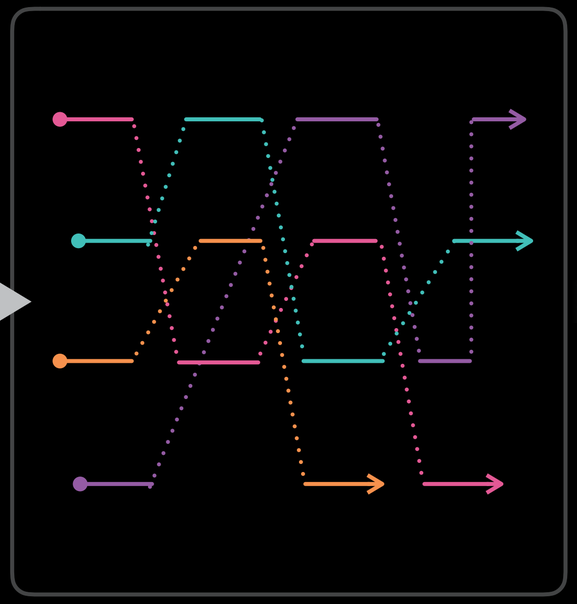
Scaling With CPU Cores

Asynchronous Programming Models, Distributed Concurrency

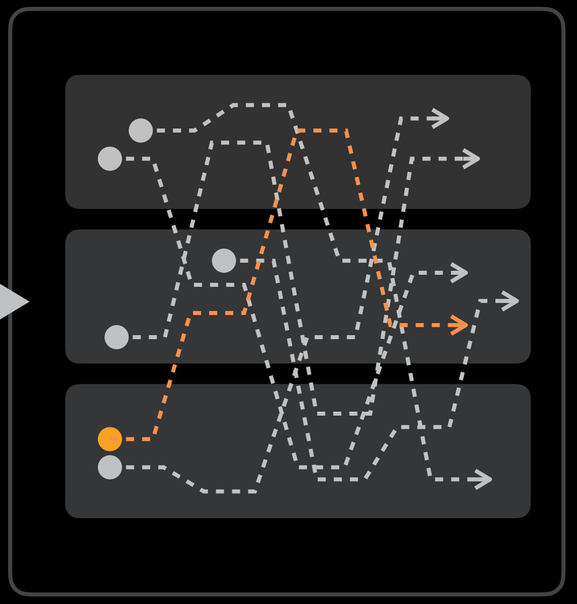
BASIC CONCURRENCY

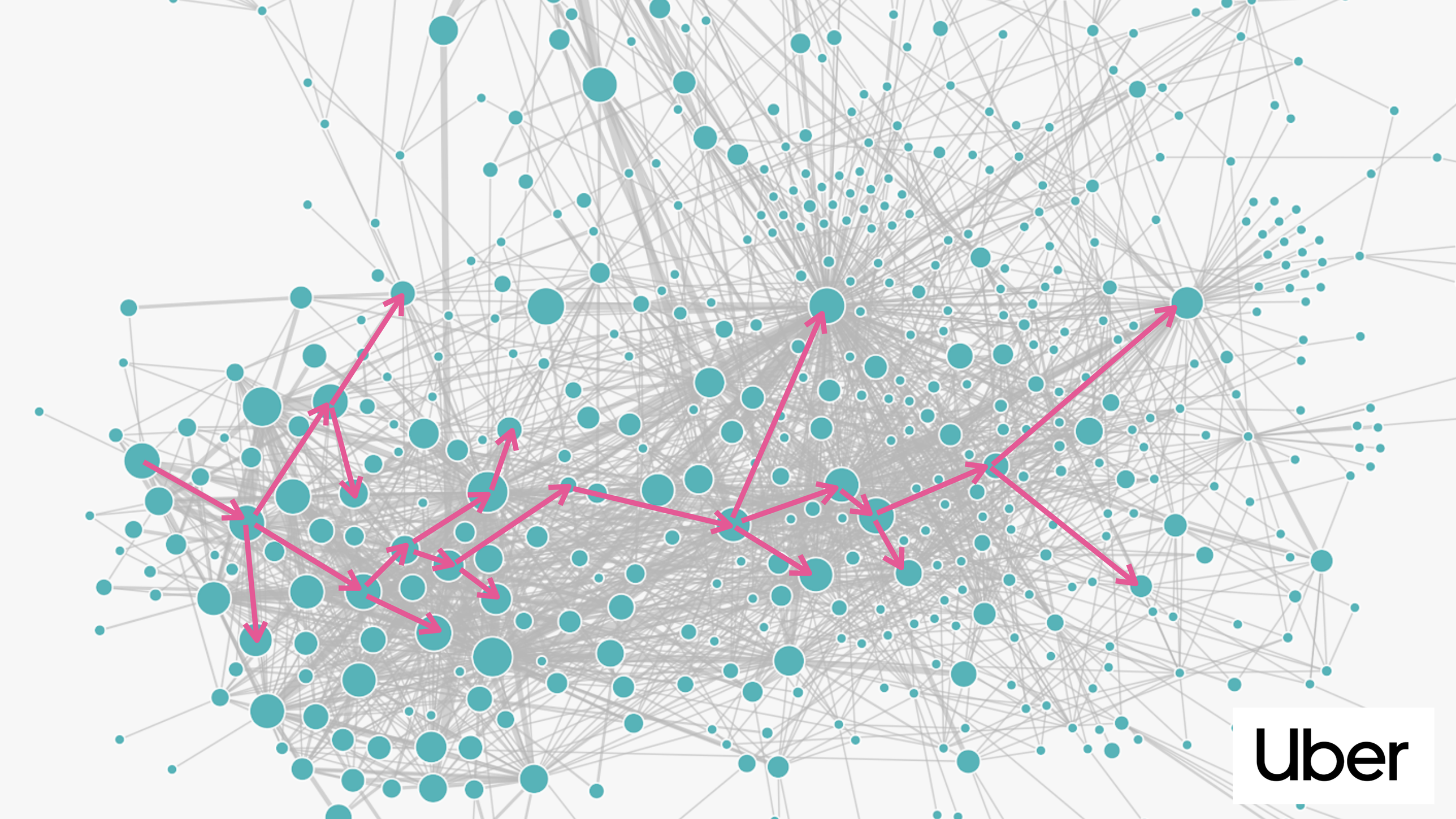


ASYNC CONCURRENCY



DISTRIBUTED CONCURRENCY




Uber

In microservices architectures
the number of failure modes
increases exponentially

Observability of
distributed transactions
is paramount!

Observability
vs.
monitoring

Observability
vs.
monitoring

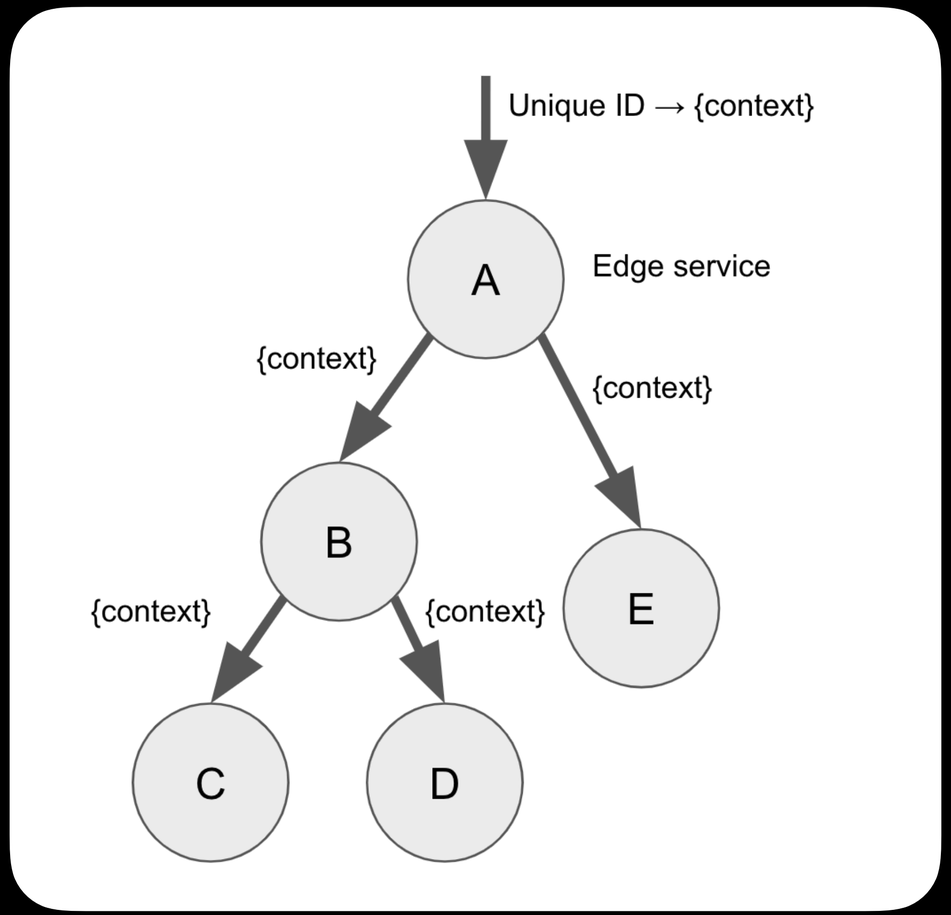
Observability

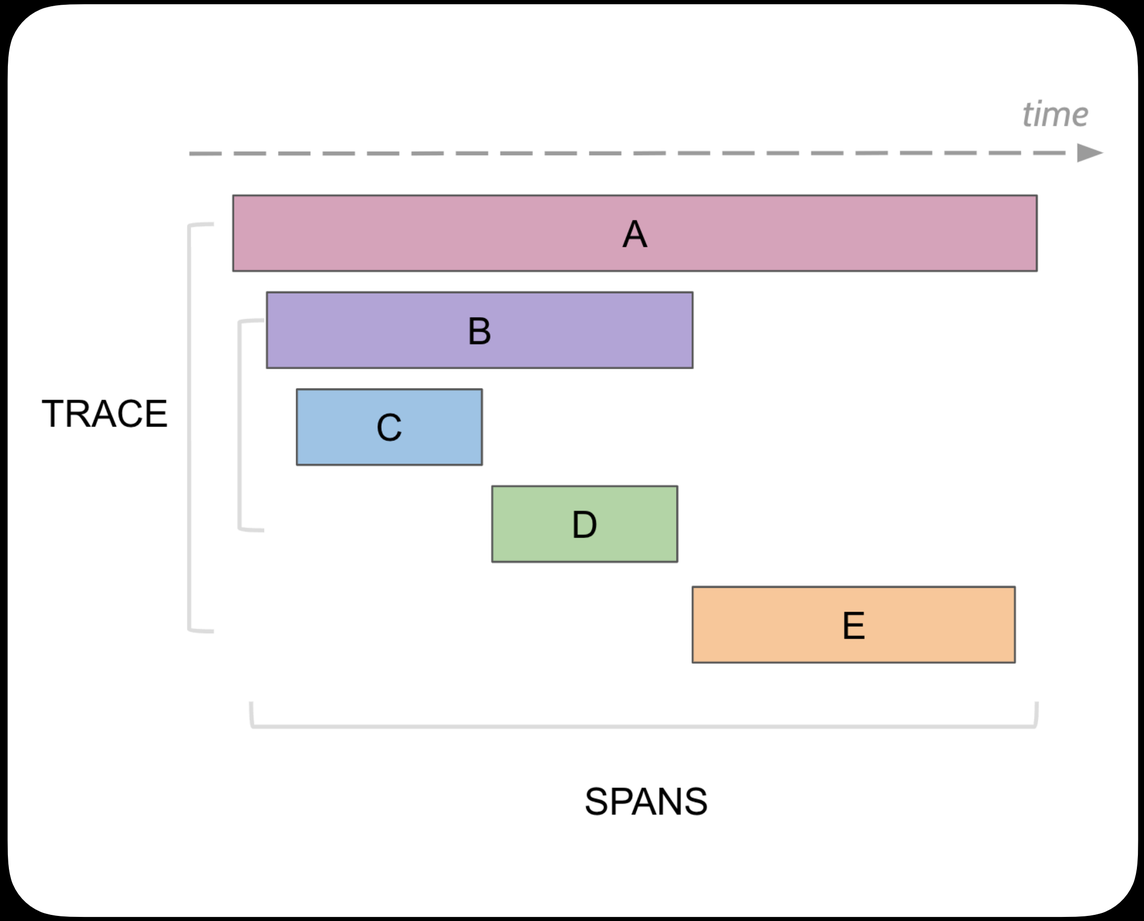
System's ability to answer questions

- Which services did the request go through
- What did every service do when processing the request
- If the request was slow, where were the bottlenecks
- If the request failed, where did the errors happen
- How different was the execution from the normal system behavior
  - Structural differences
  - Performance differences
- What was on the critical path of the request
- Who should be paged

Distributed tracing
can answer these questions
and accelerate root cause analysis

Distributed Tracing in a Nutshell

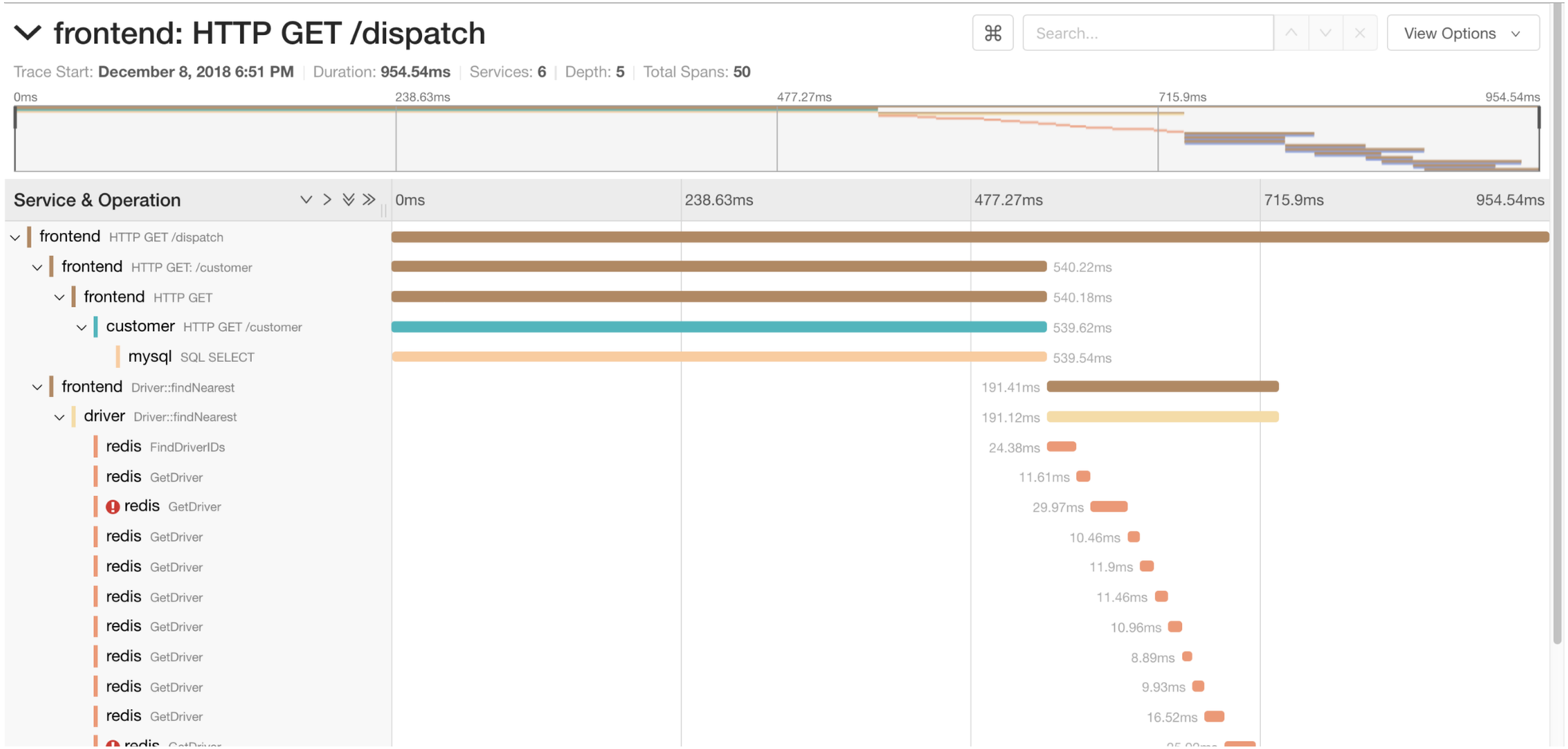




Trace as a narrative

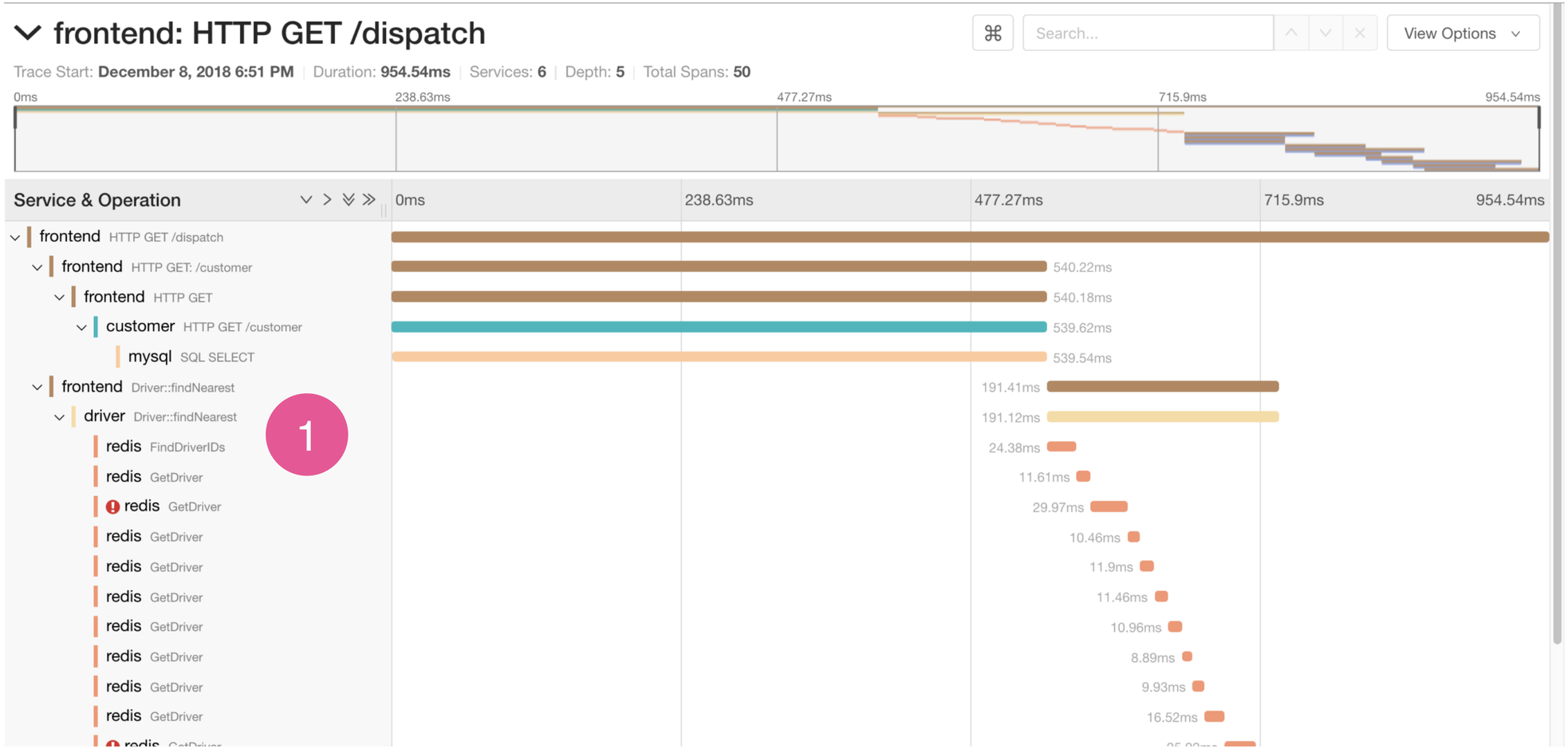
Trace Timeline

Classic trace view as Gantt chart



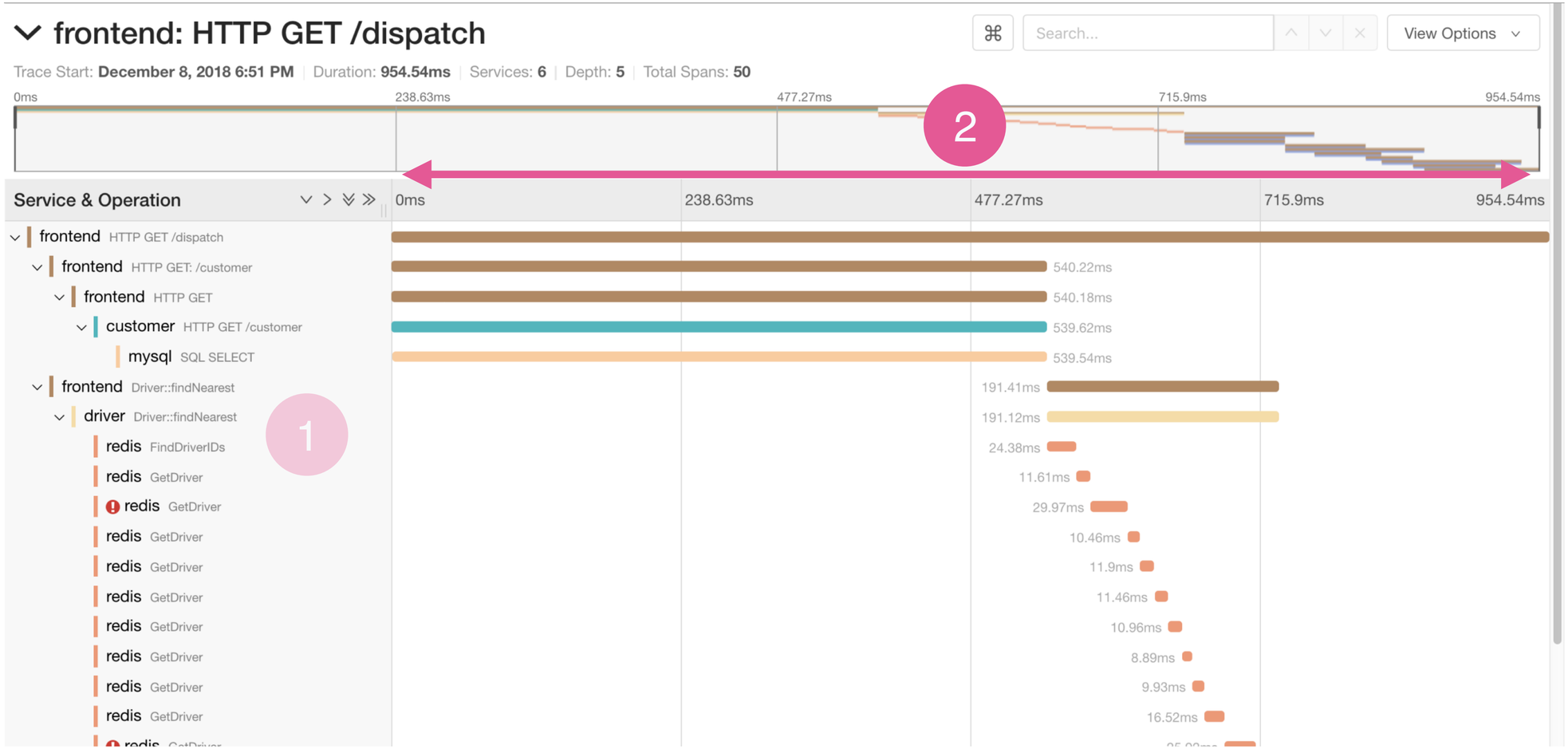
Trace Timeline

Parent → Child → Grandchild



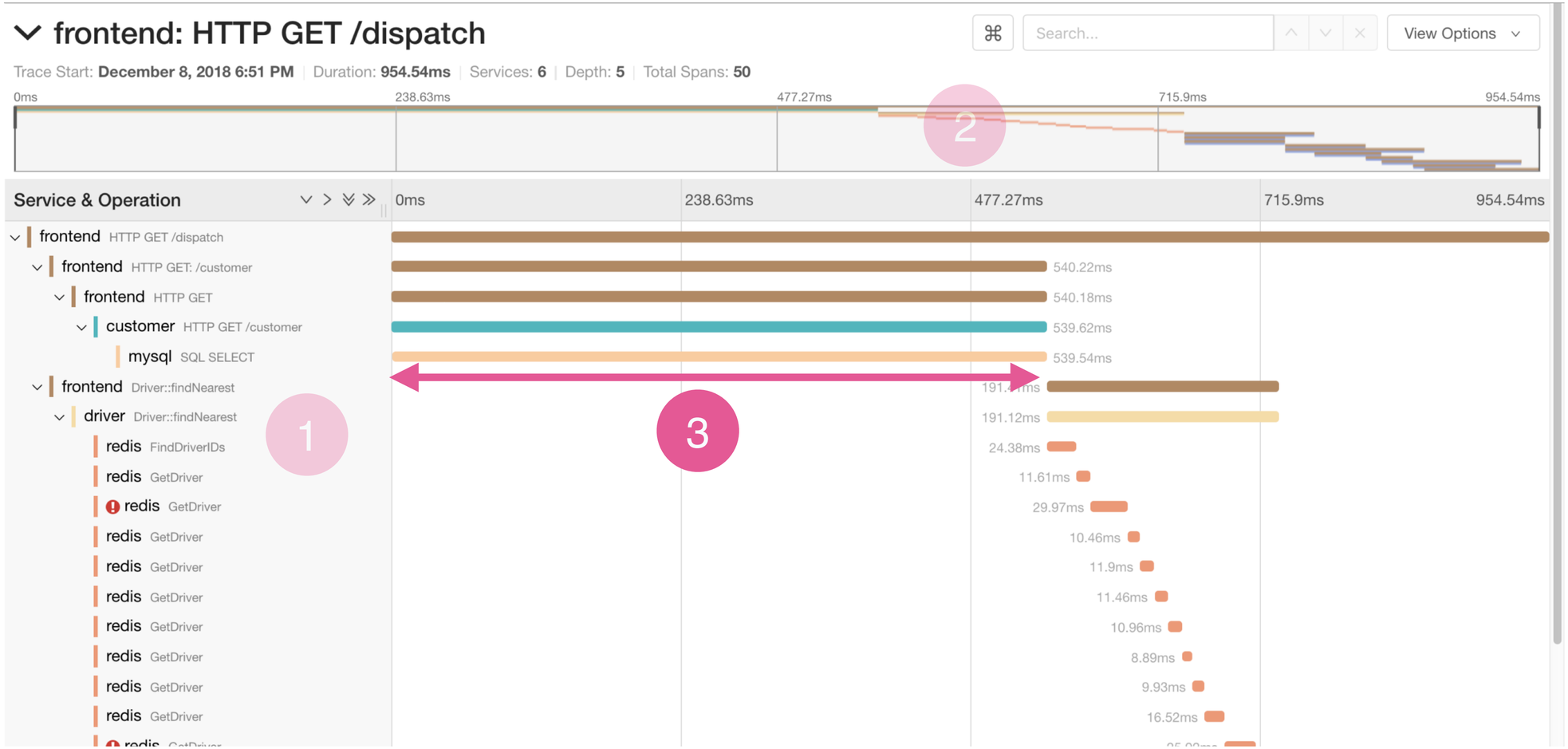
Trace Timeline

Time + Mini-Map



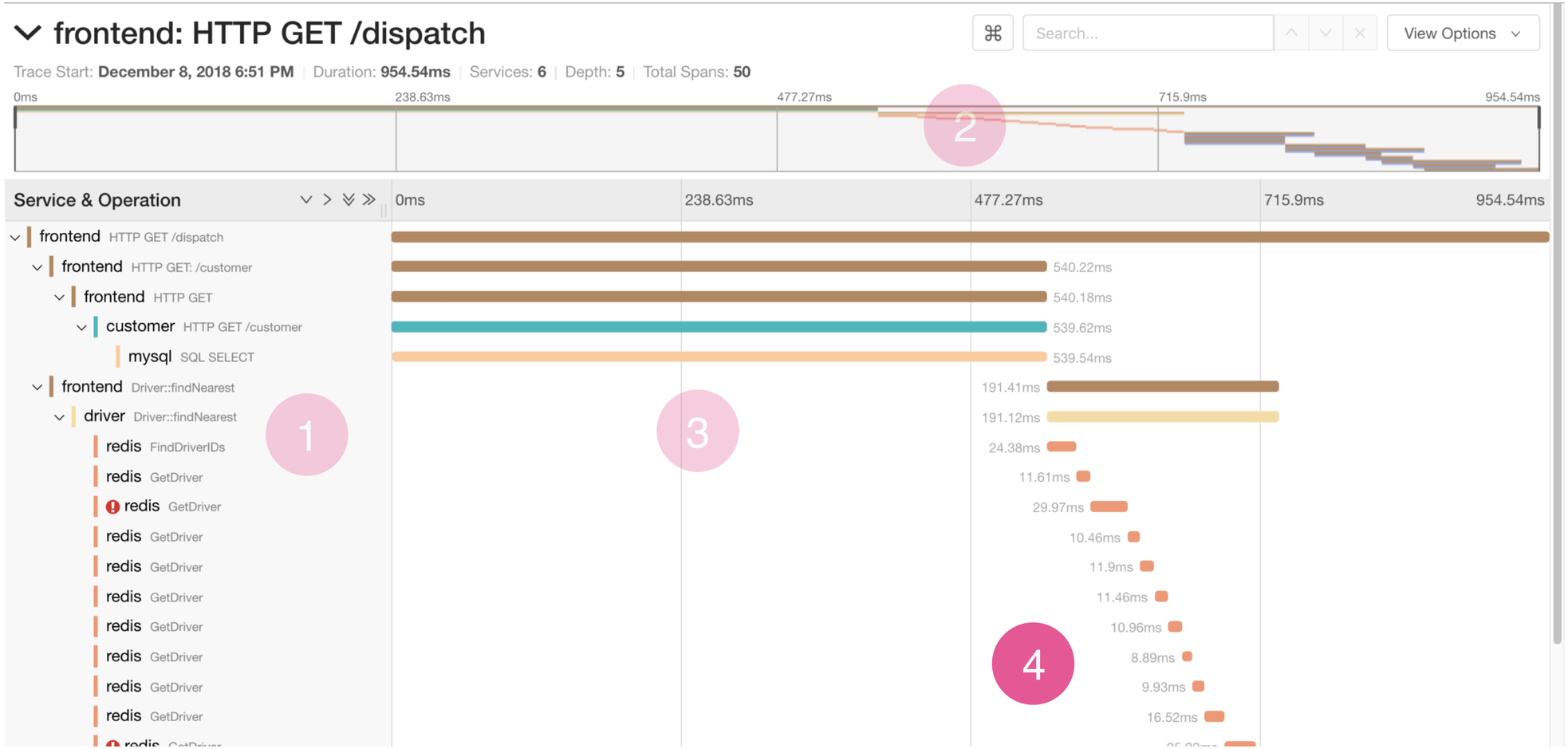
Trace Timeline

Blocking operation



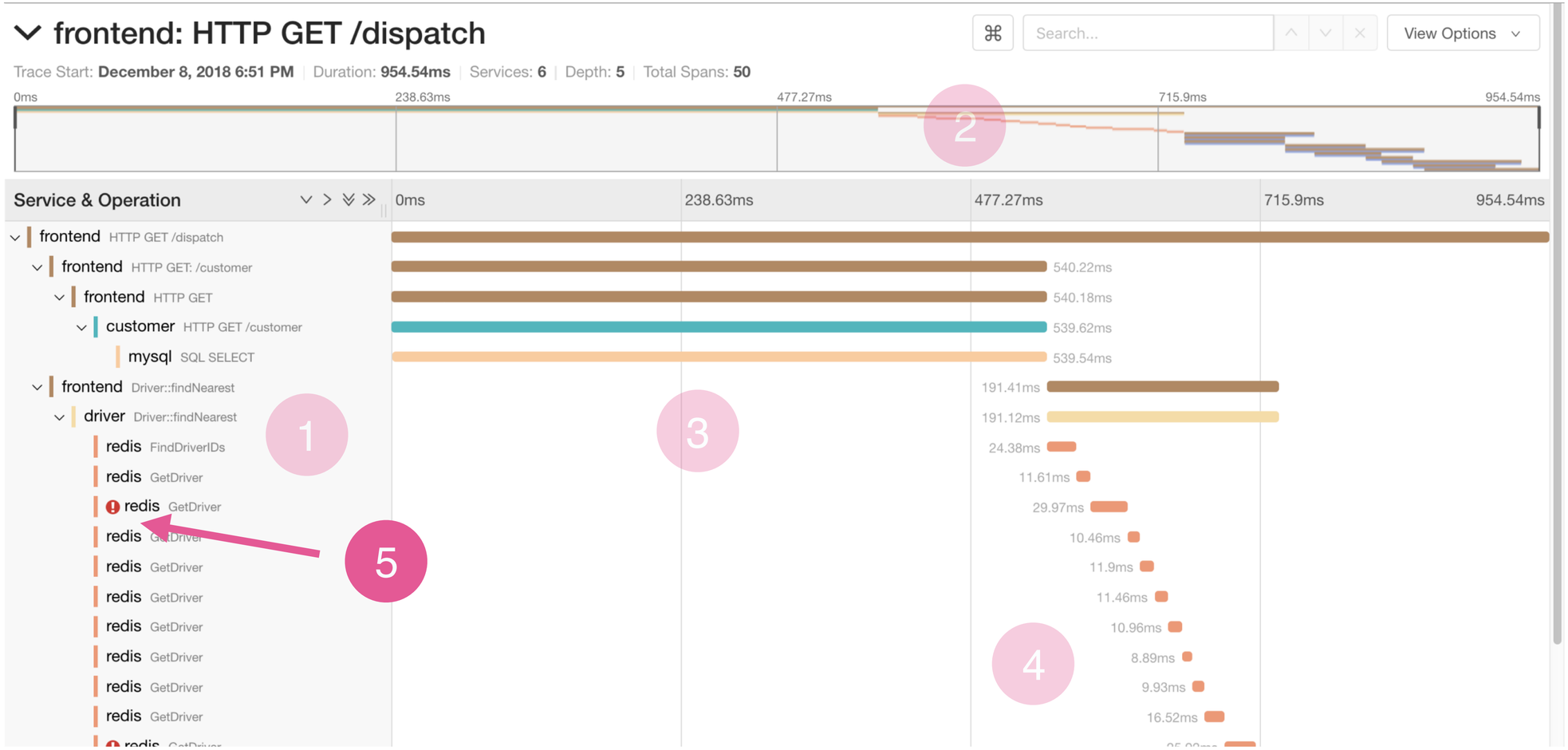
Trace Timeline

Sequential operations

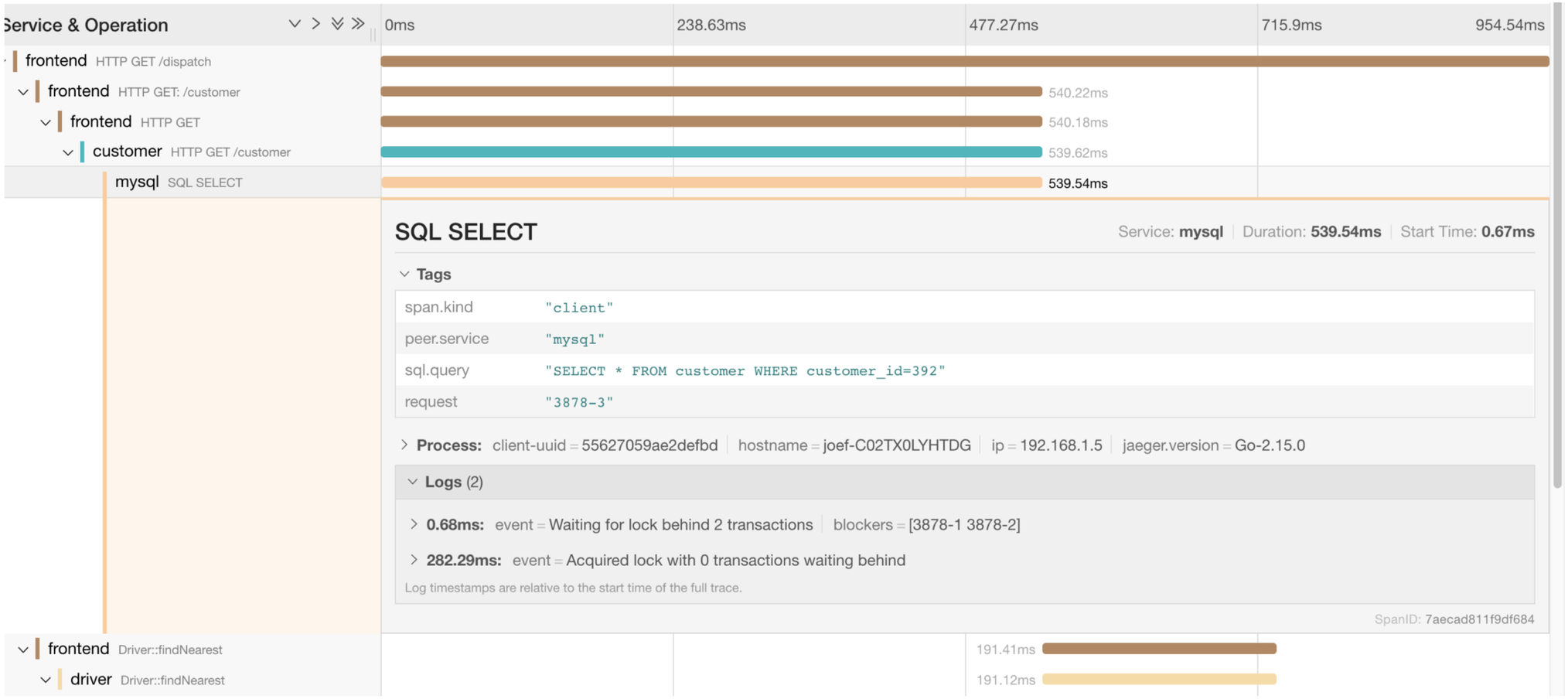


Trace Timeline

Errors

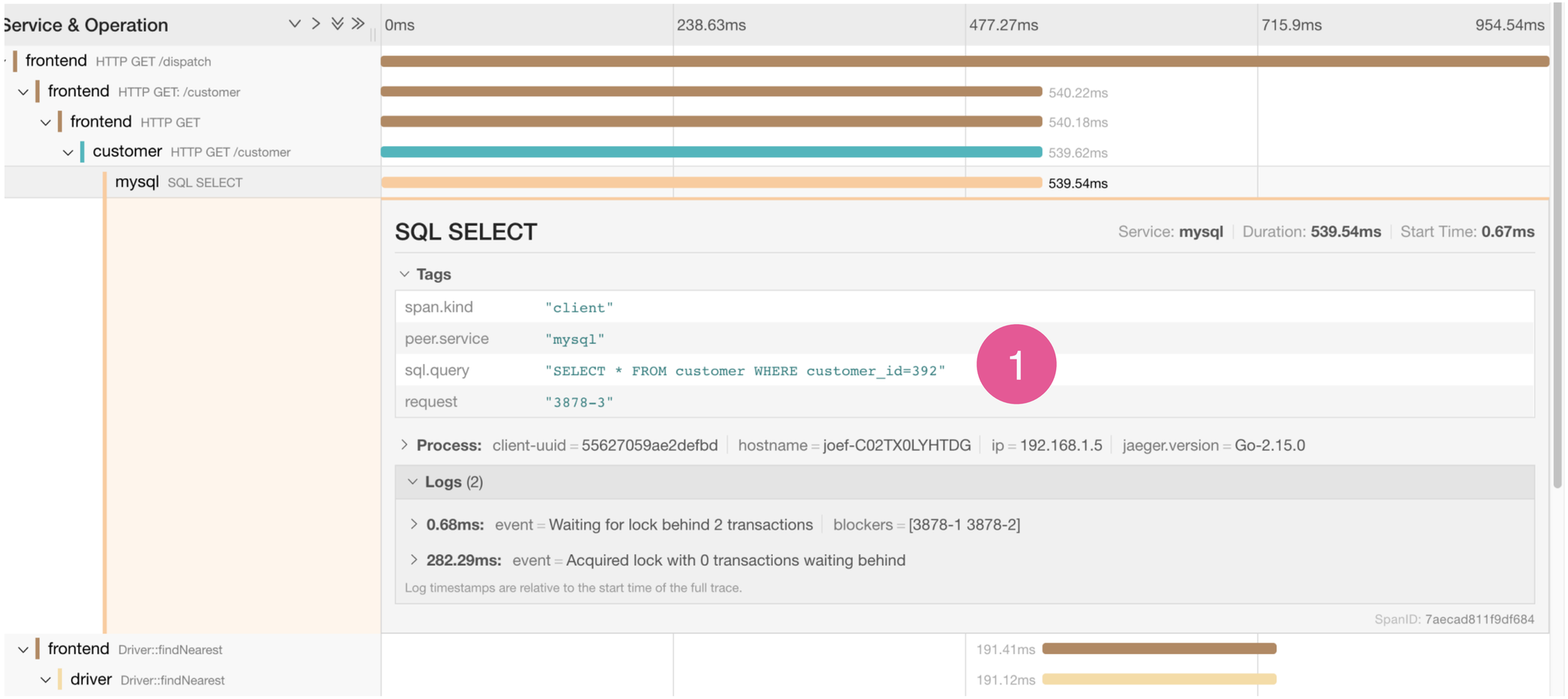


Span details



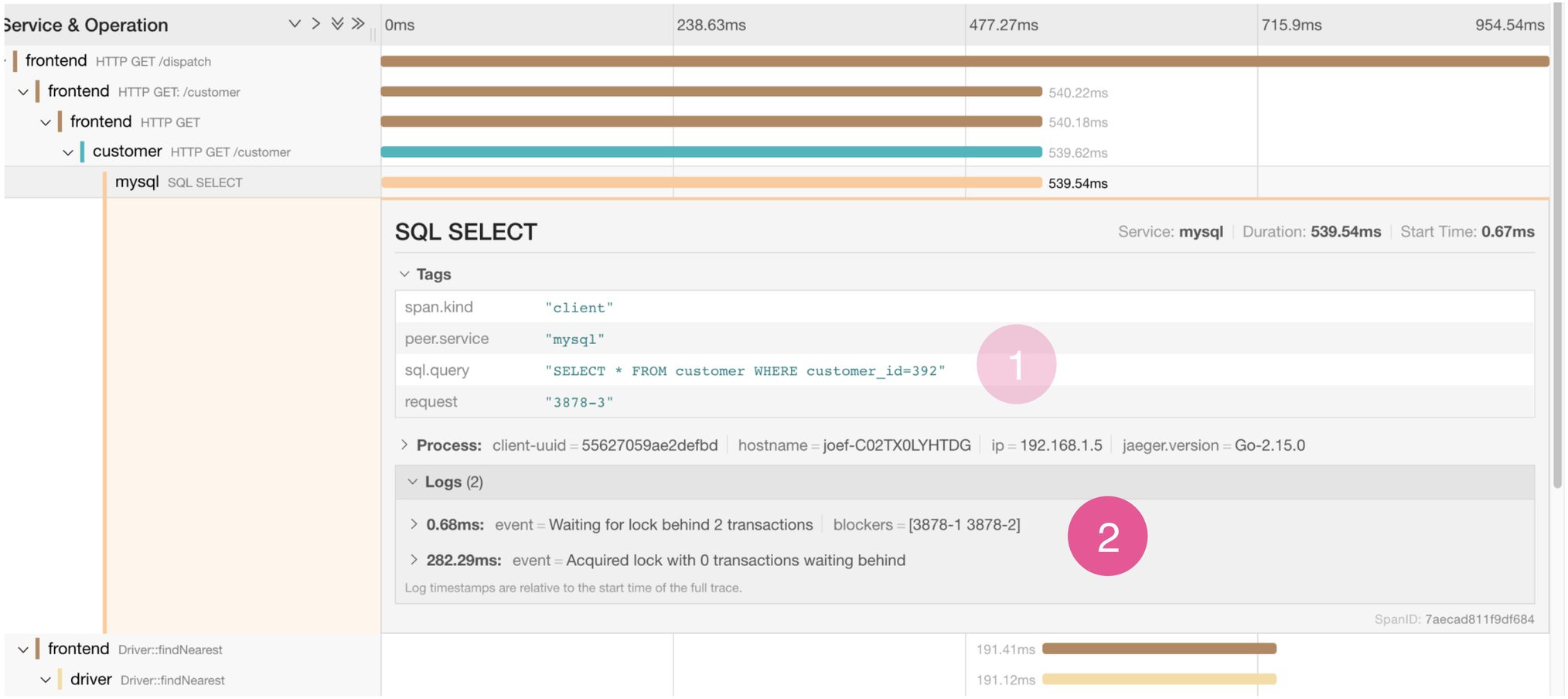
Span details

Database query



Span details

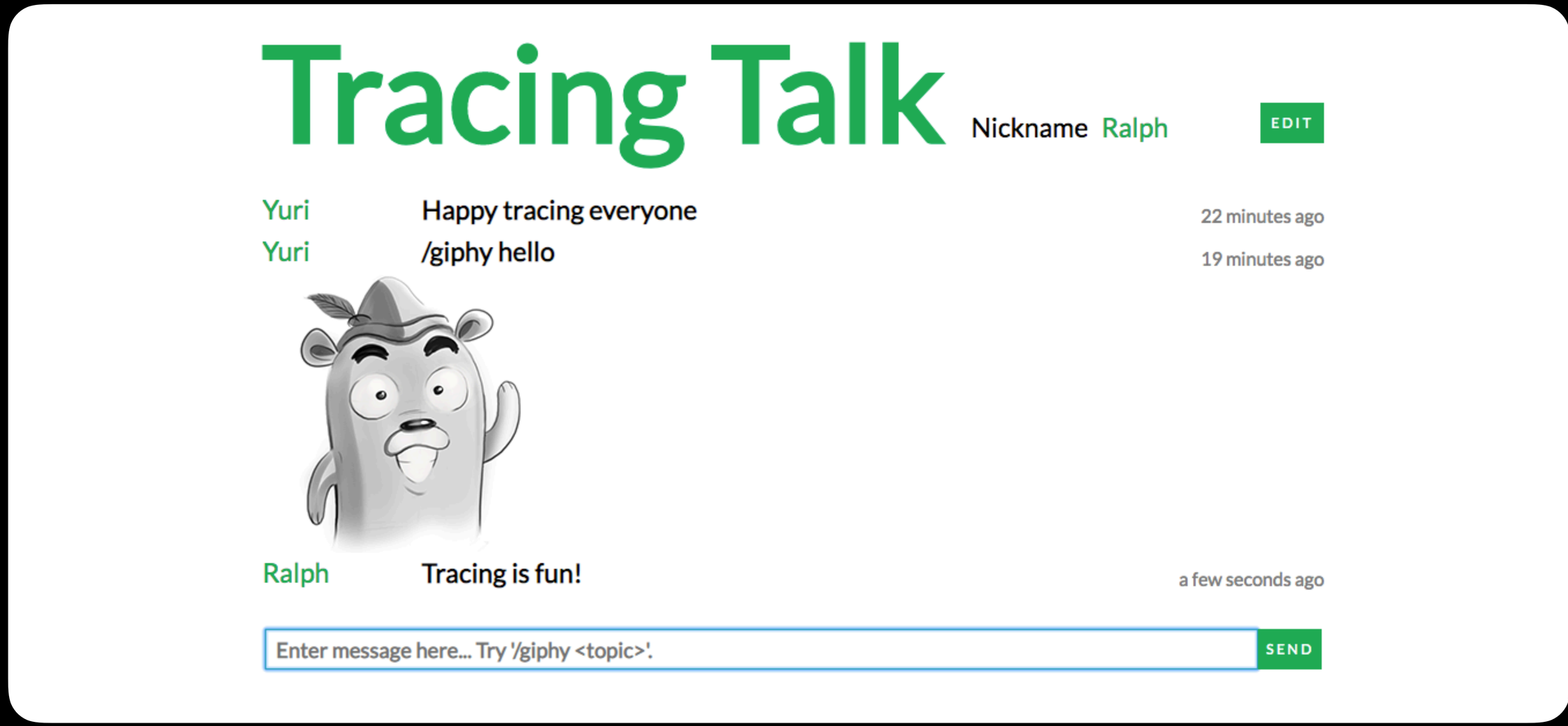
Timed events (logs)



We can also trace
asynchronous workflows

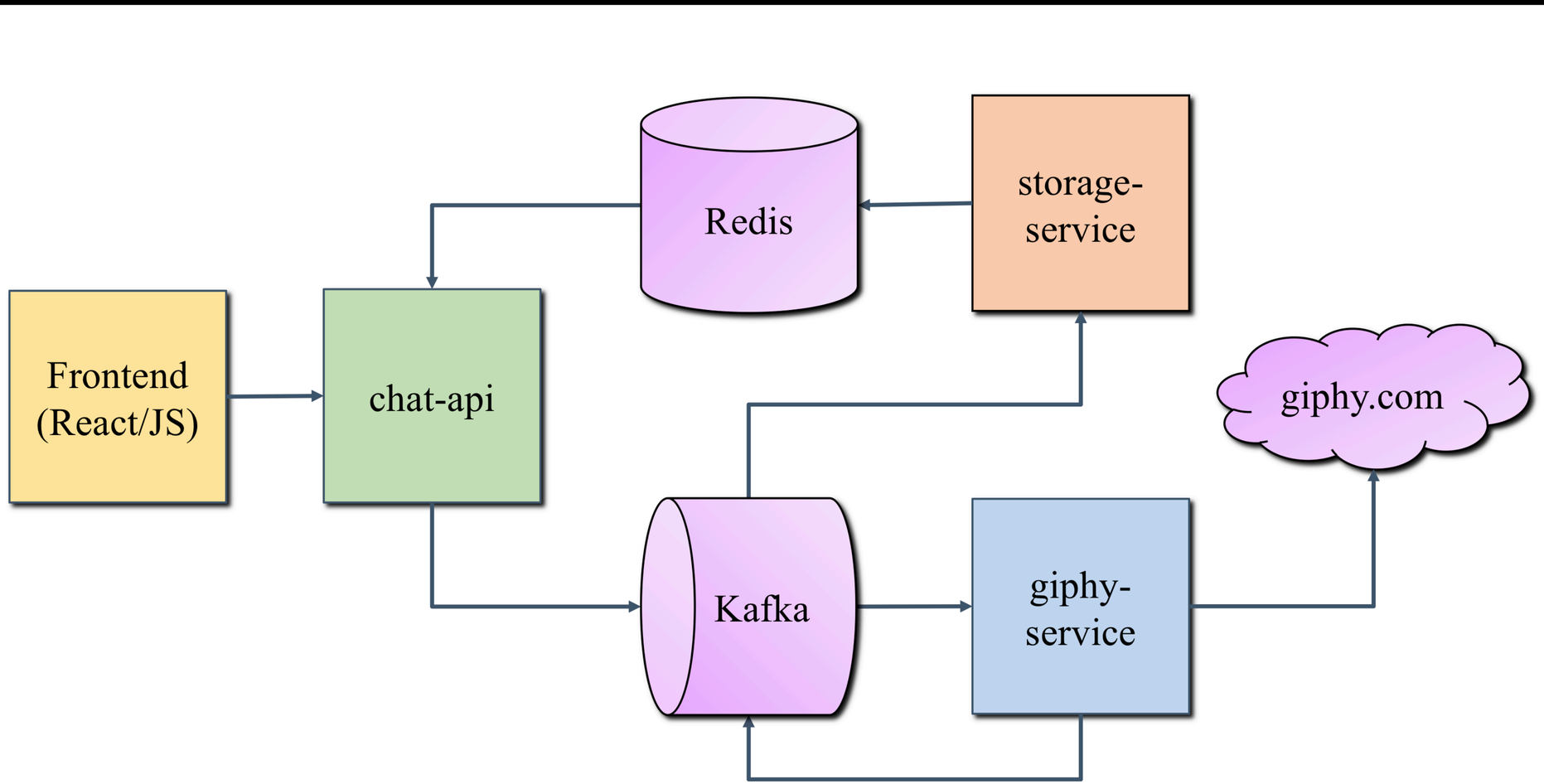
Tracing Talk Application

Mastering Distributed Tracing, Chapter 5



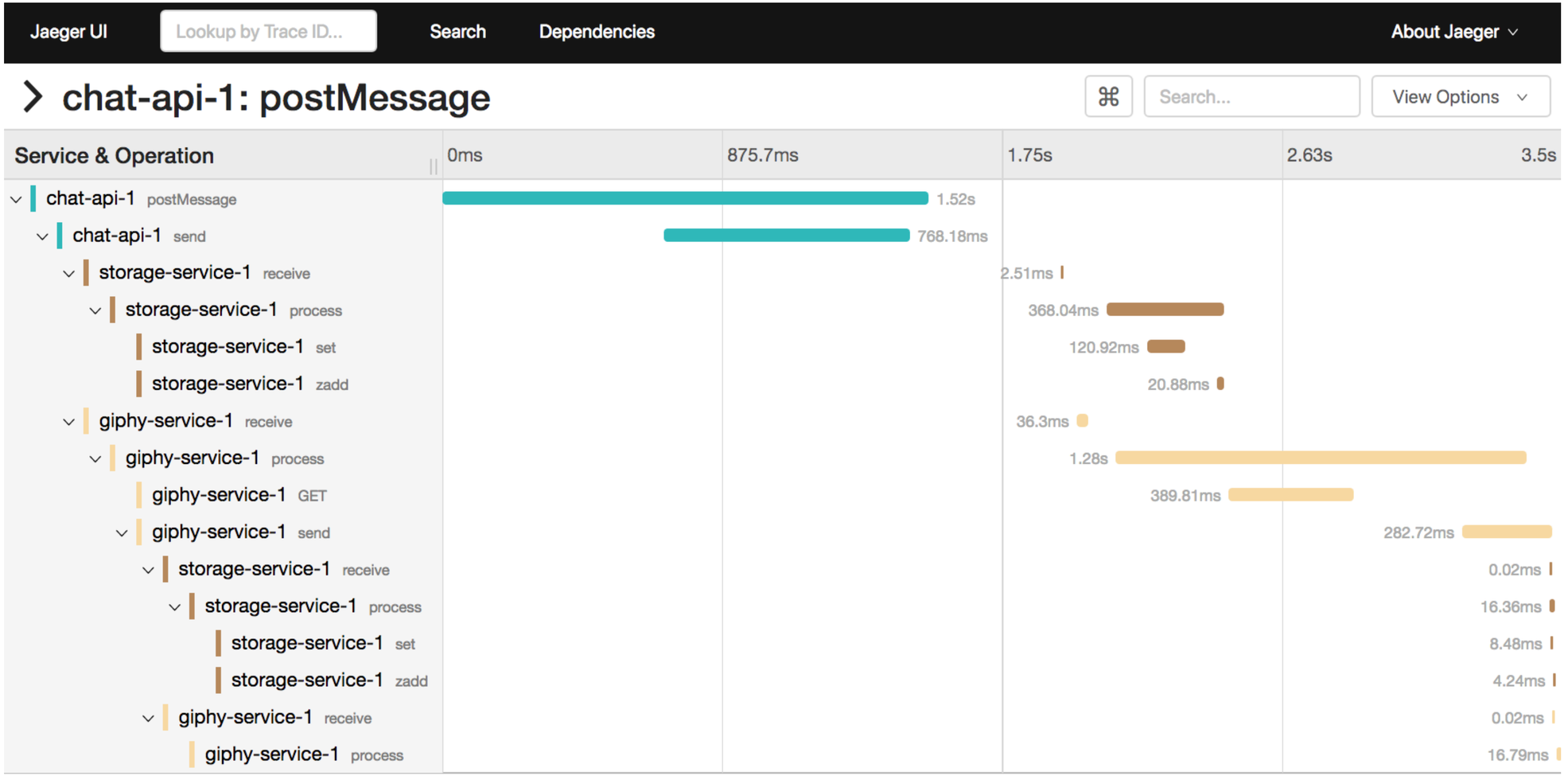
Tracing Talk Application

Architecture



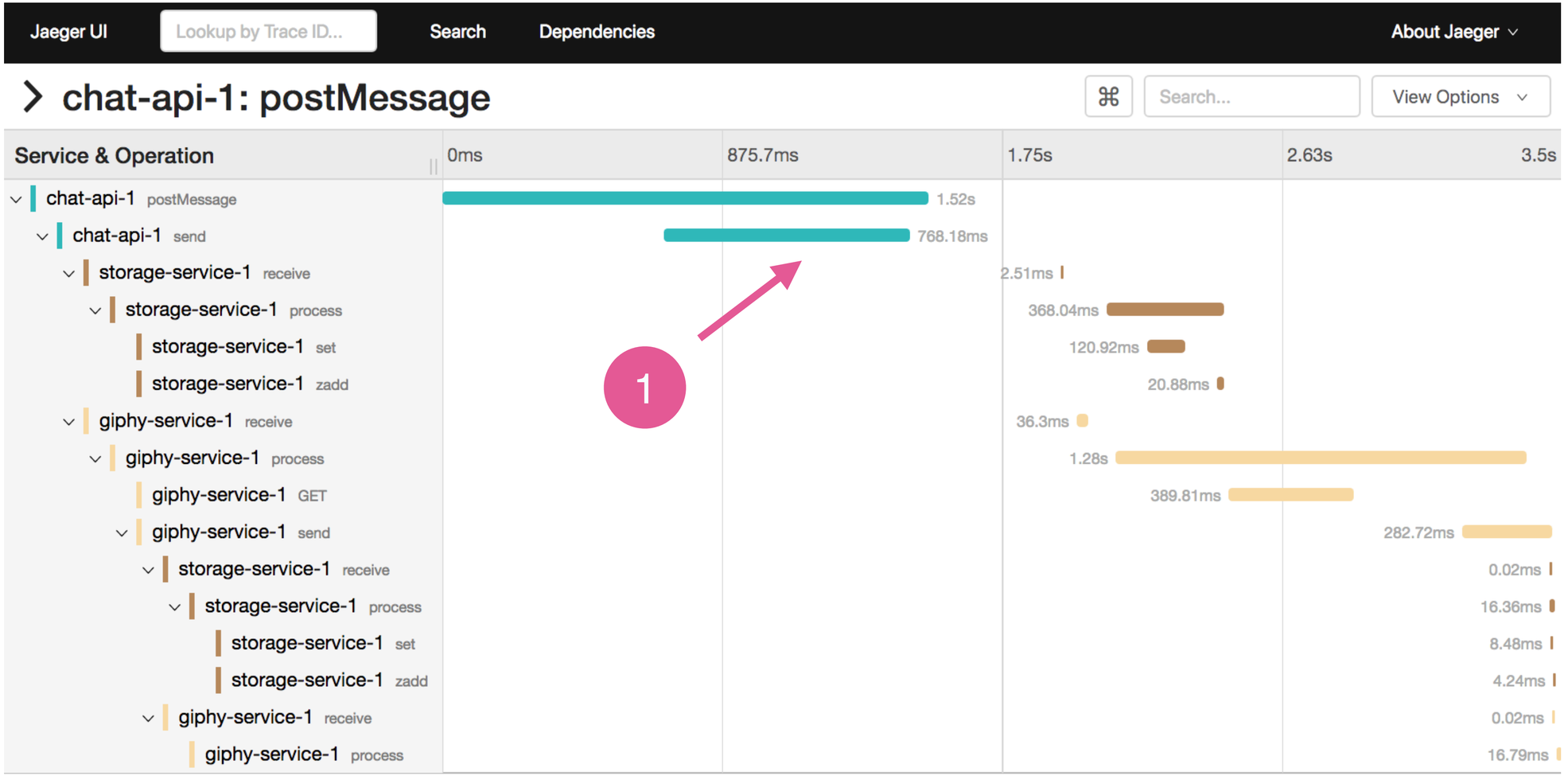
Tracing Talk Application

Request trace



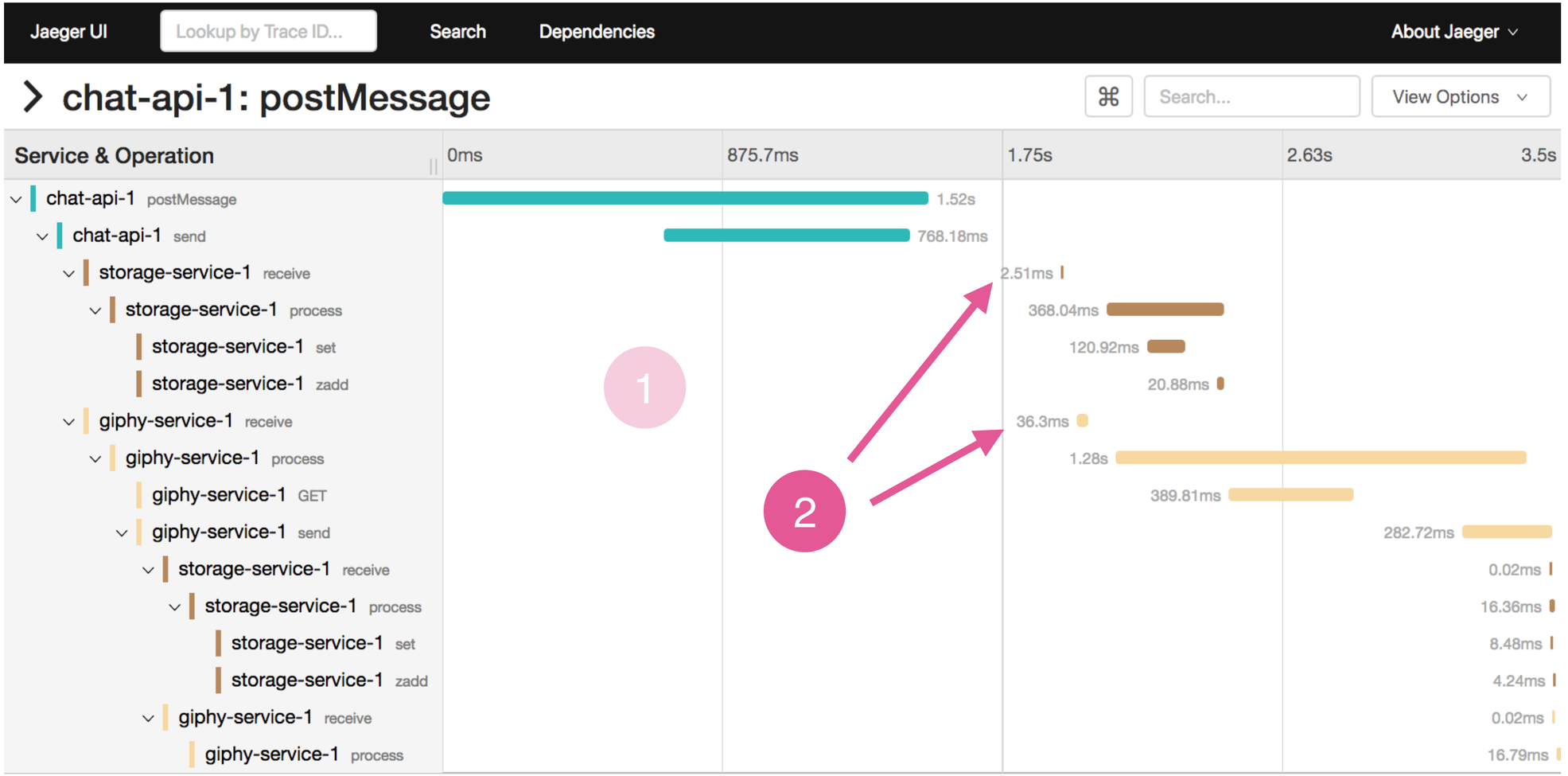
Tracing Talk Application

Message sent



Tracing Talk Application

Message received



Single Trace

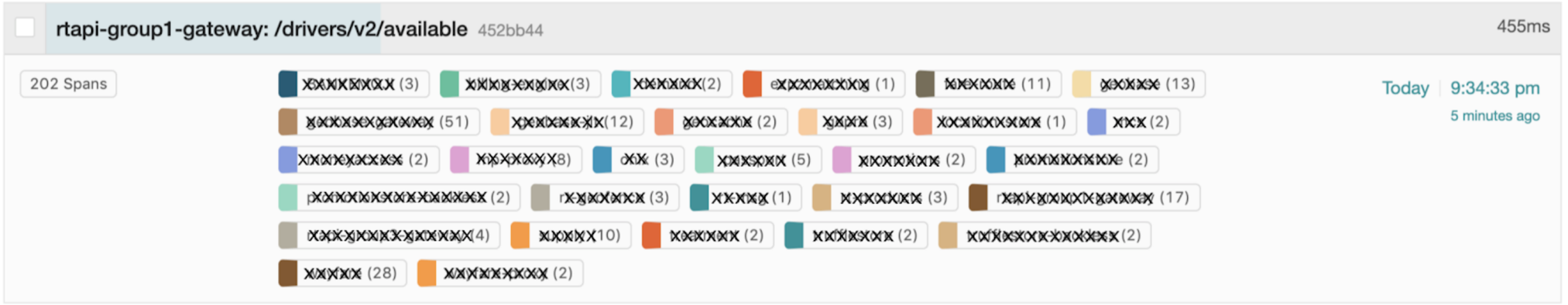
Pros and cons

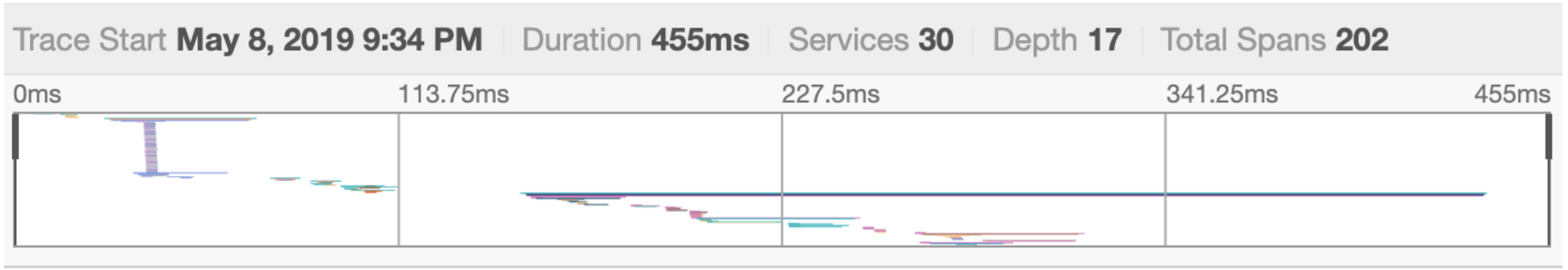
- Tells a story about a single transaction
- Allows deep contextual drill-down
- Acts as a distributed stack trace

- Tells a story about a single transaction. What if it’s an anomaly?
- One trace can be overwhelmingly complex

Too Much Complexity

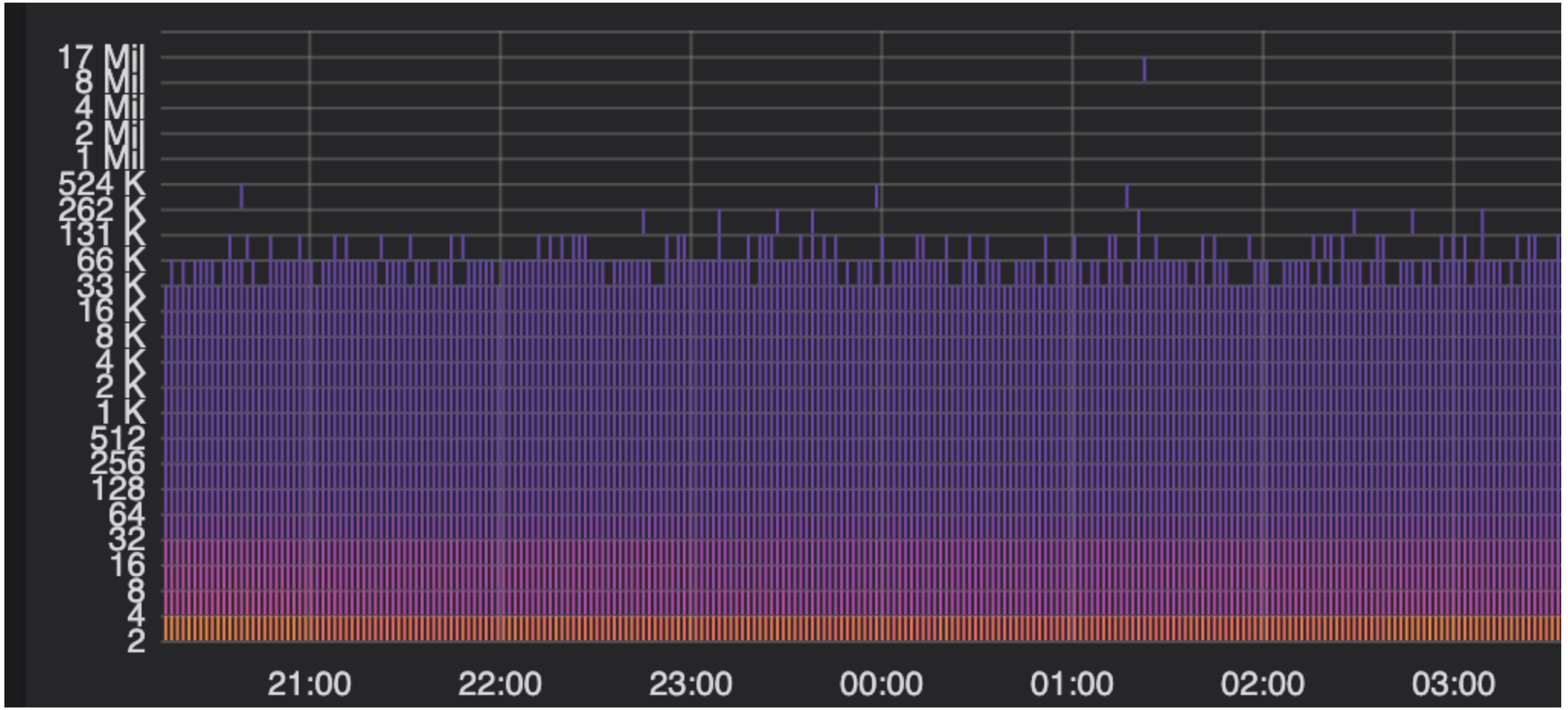
One request - 30 services, 100+ RPCs





Too Much Complexity

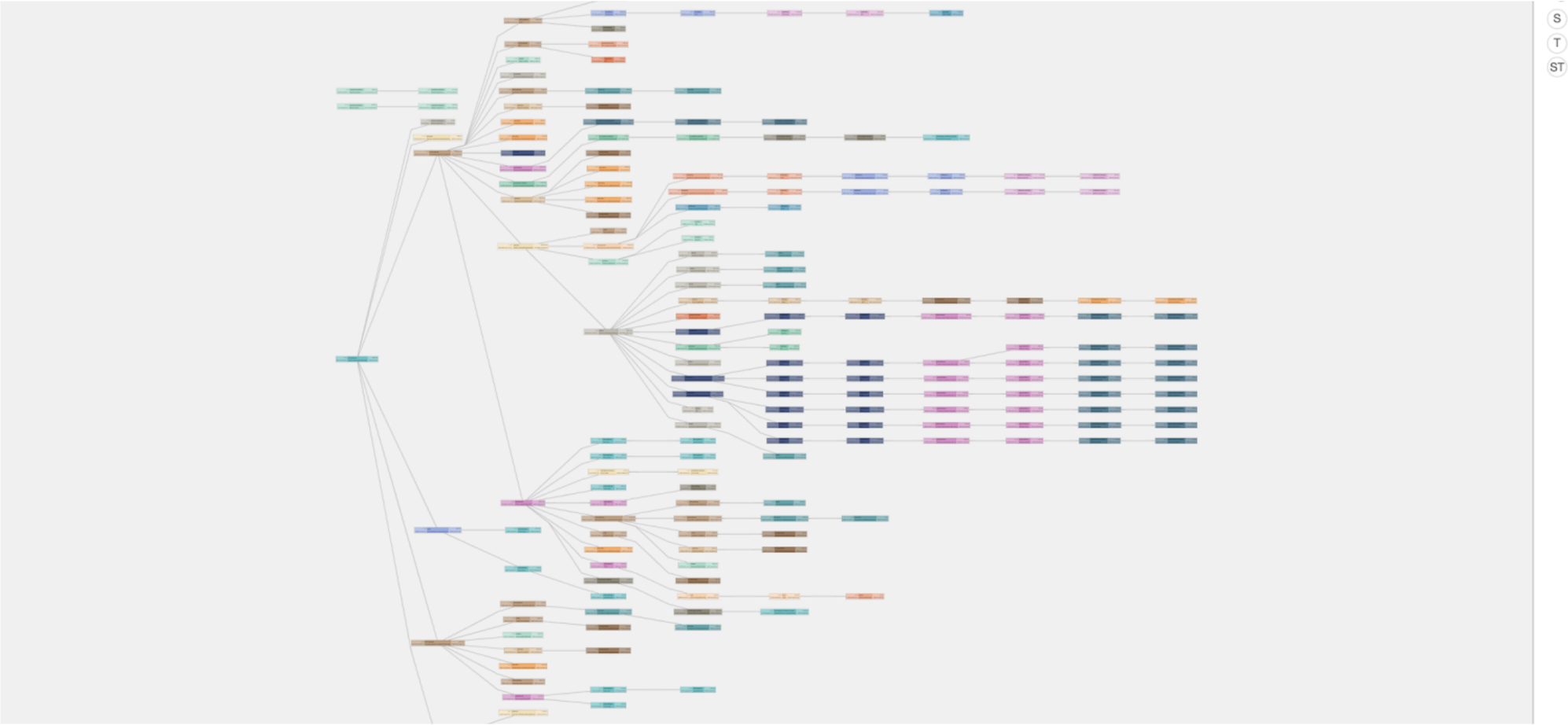
Some traces have hundreds of thousands spans



Reducing complexity by
smarter visualizations

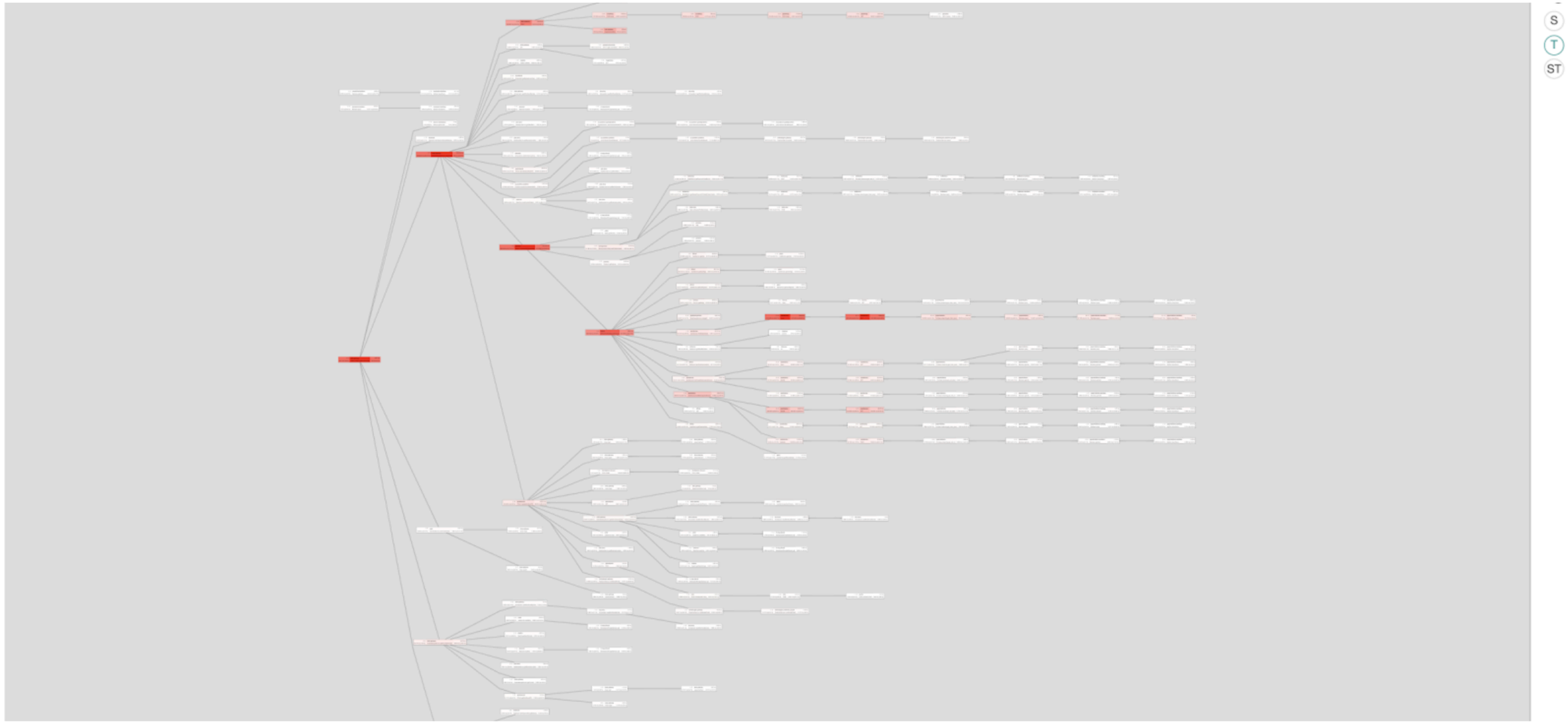
Trace graph

Time ordered, repeated edges collapsed



Trace graph

Latency heat map

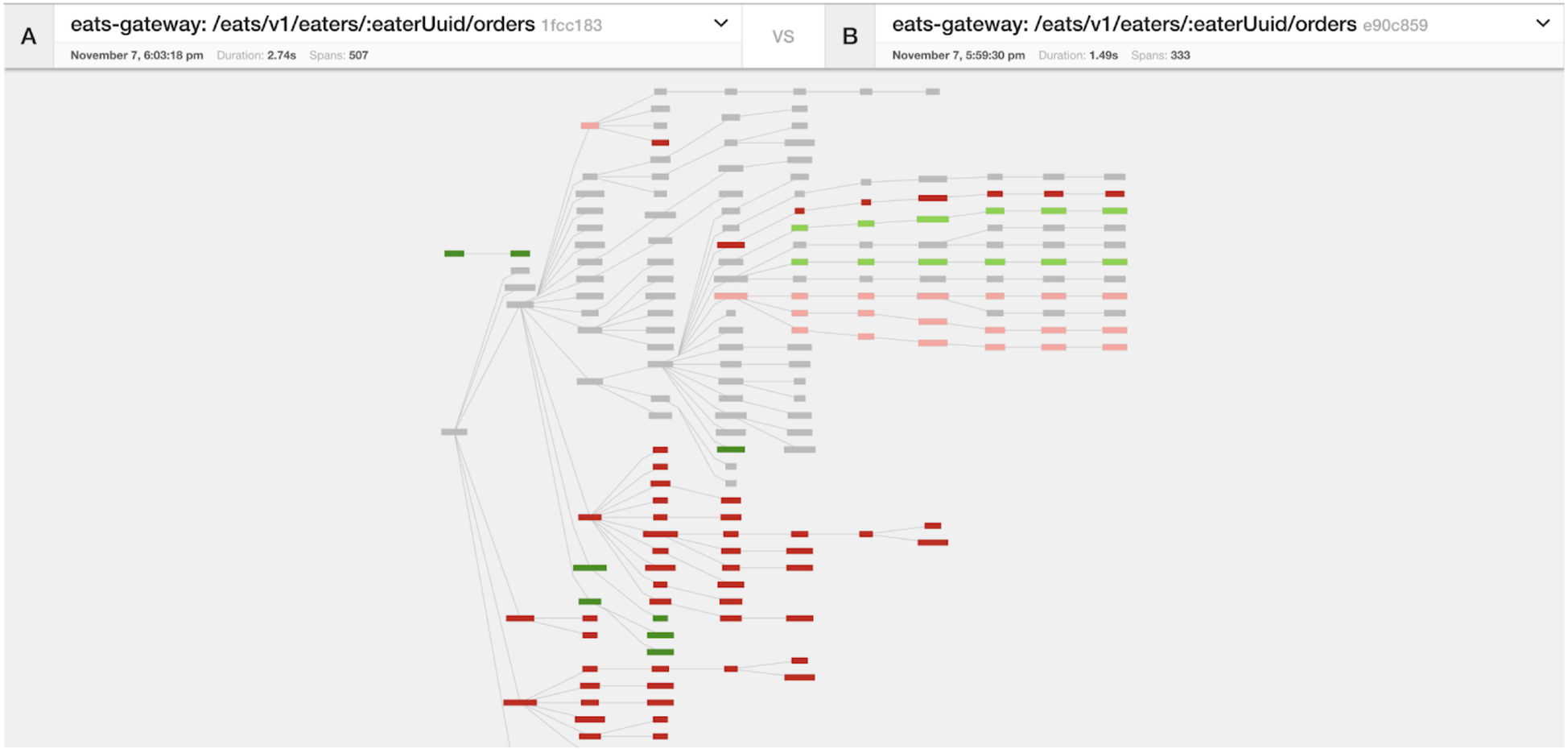


Finding anomalies is easier
when we look at differences
in performance profiles

Trace vs. Trace

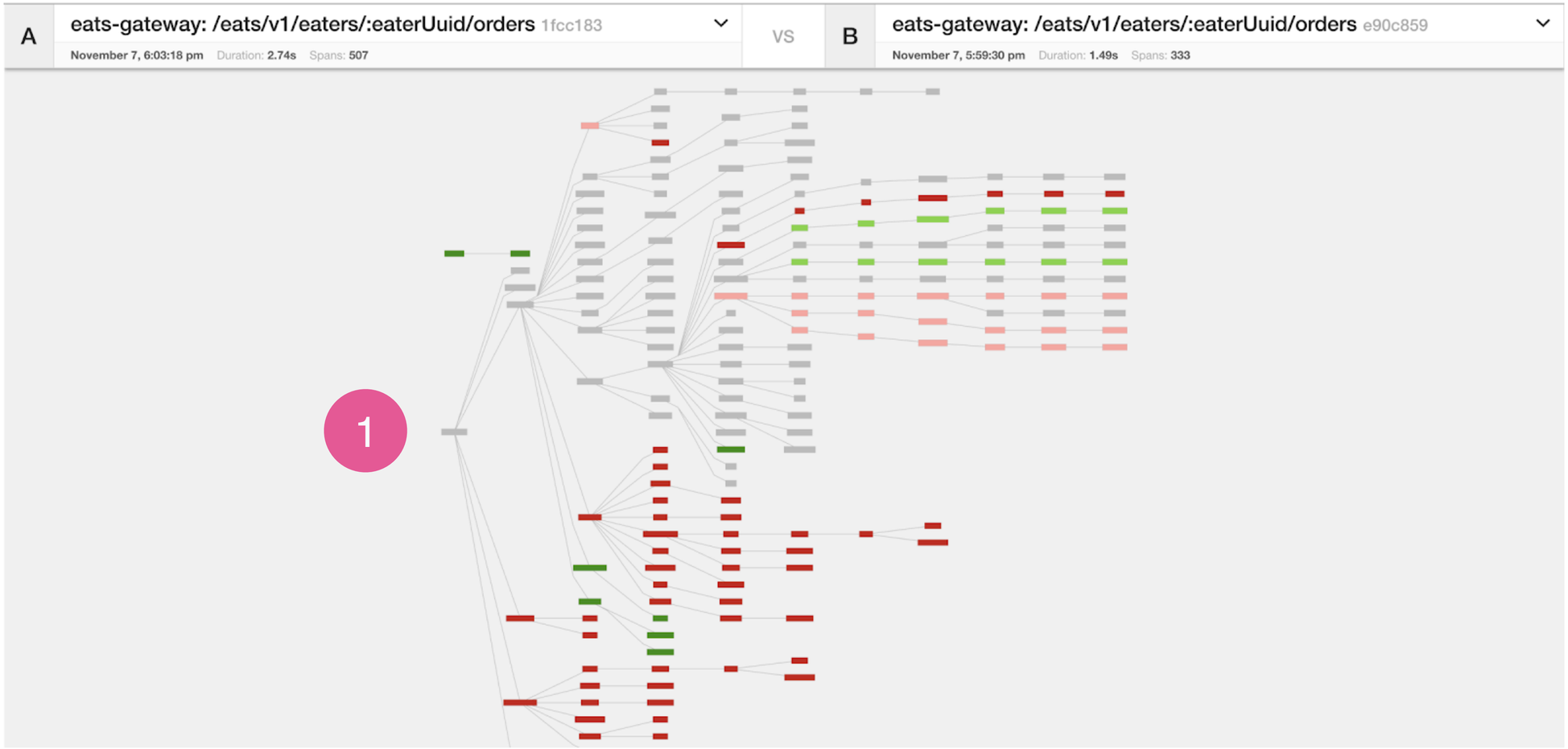
Comparing Trace Structures

Just like a Code Diff



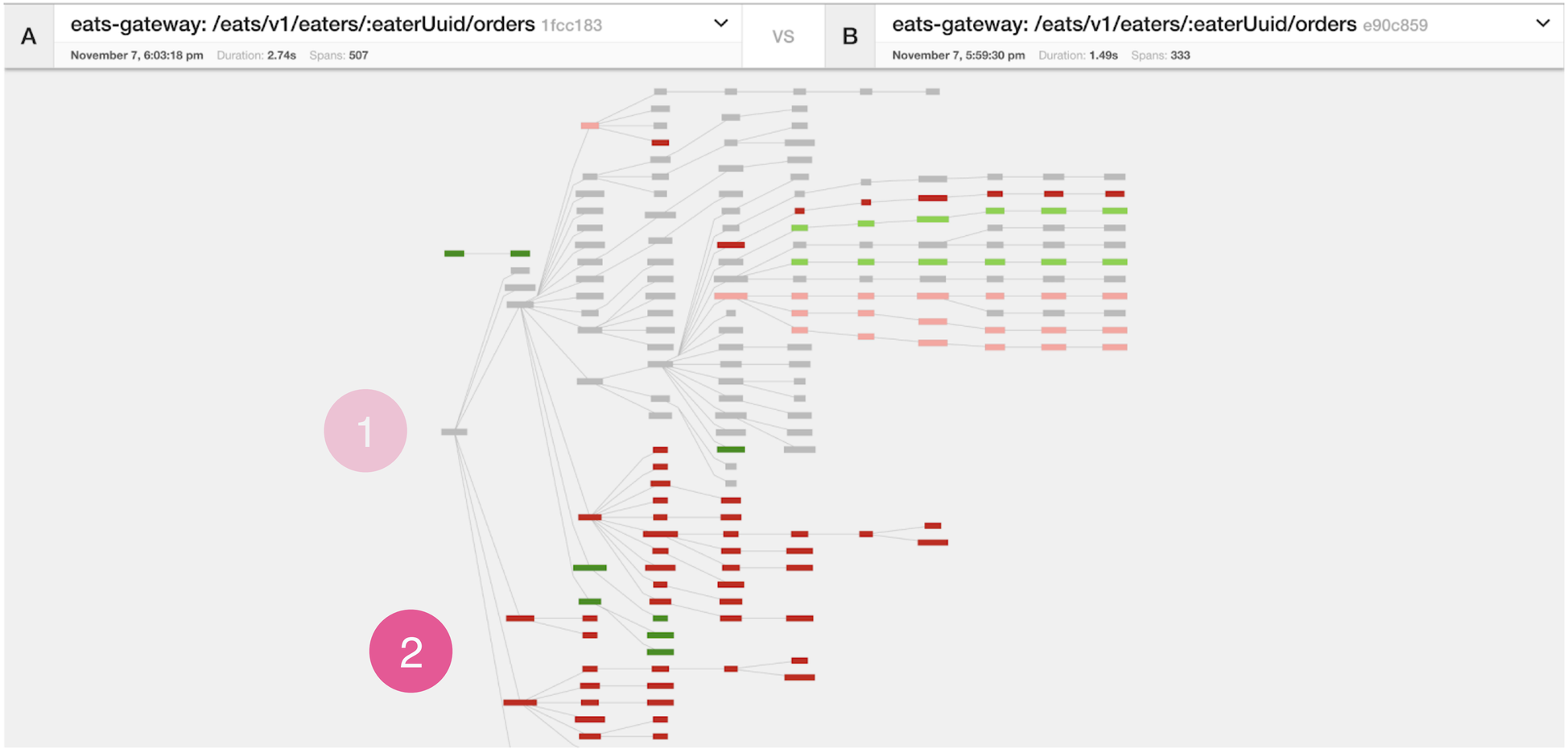
Comparing Trace Structures

Shared Structure



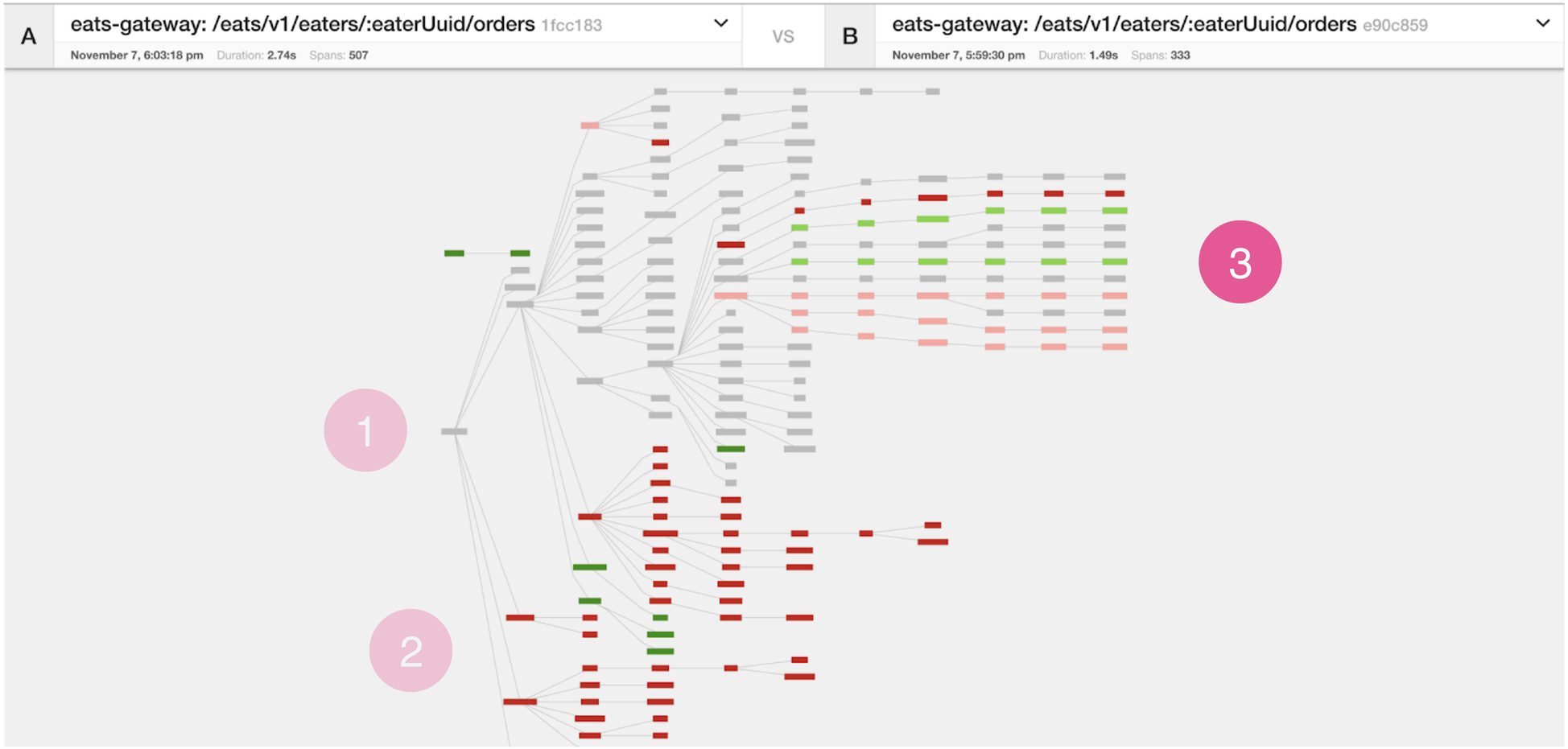
Comparing Trace Structures

Absent in One or the Traces



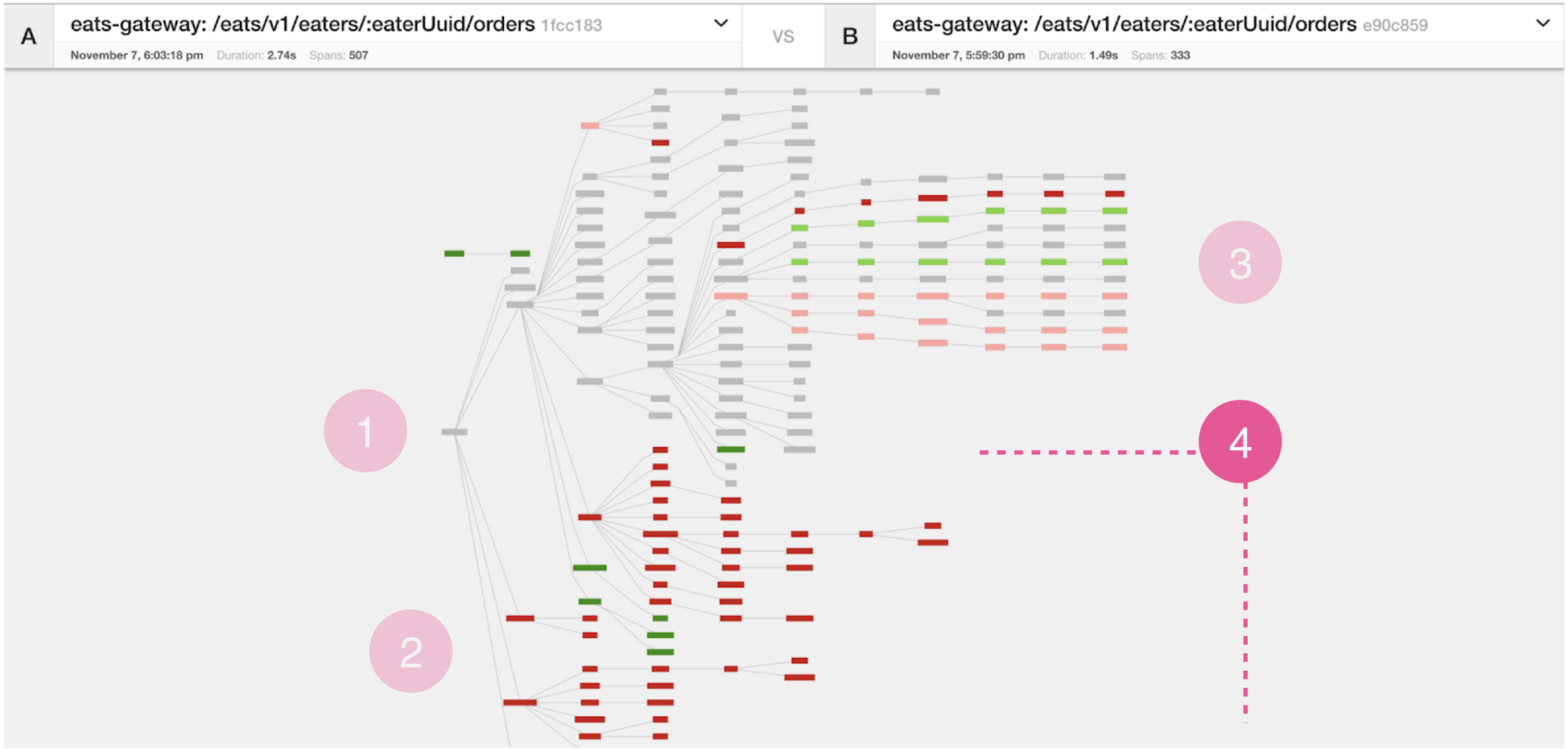
Comparing Trace Structures

More or Fewer Spans Within a Node



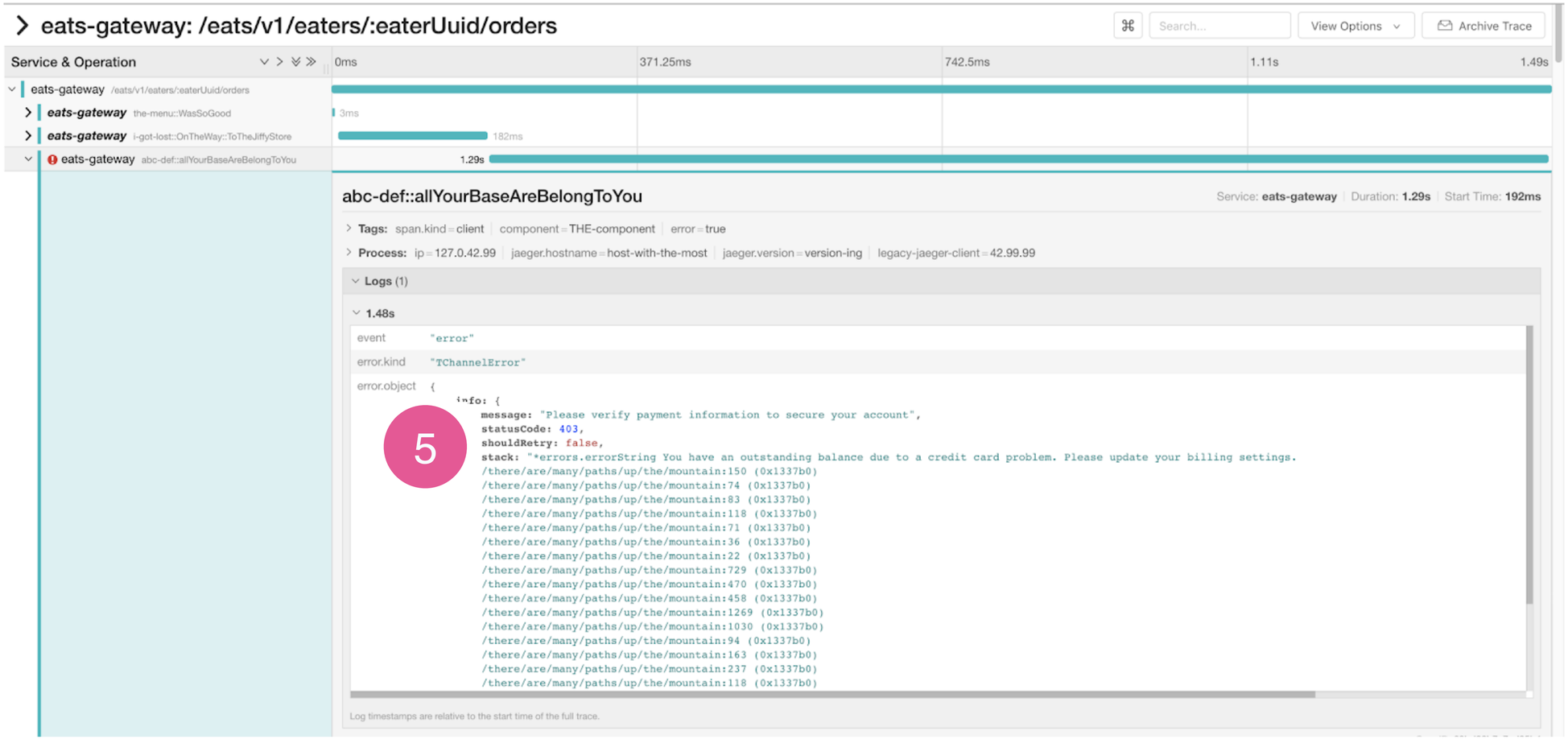
Comparing Trace Structures

Substantial Divergence



Deep Linking to Raw Traces & Spans

Error: ”You have an outstanding balance…"



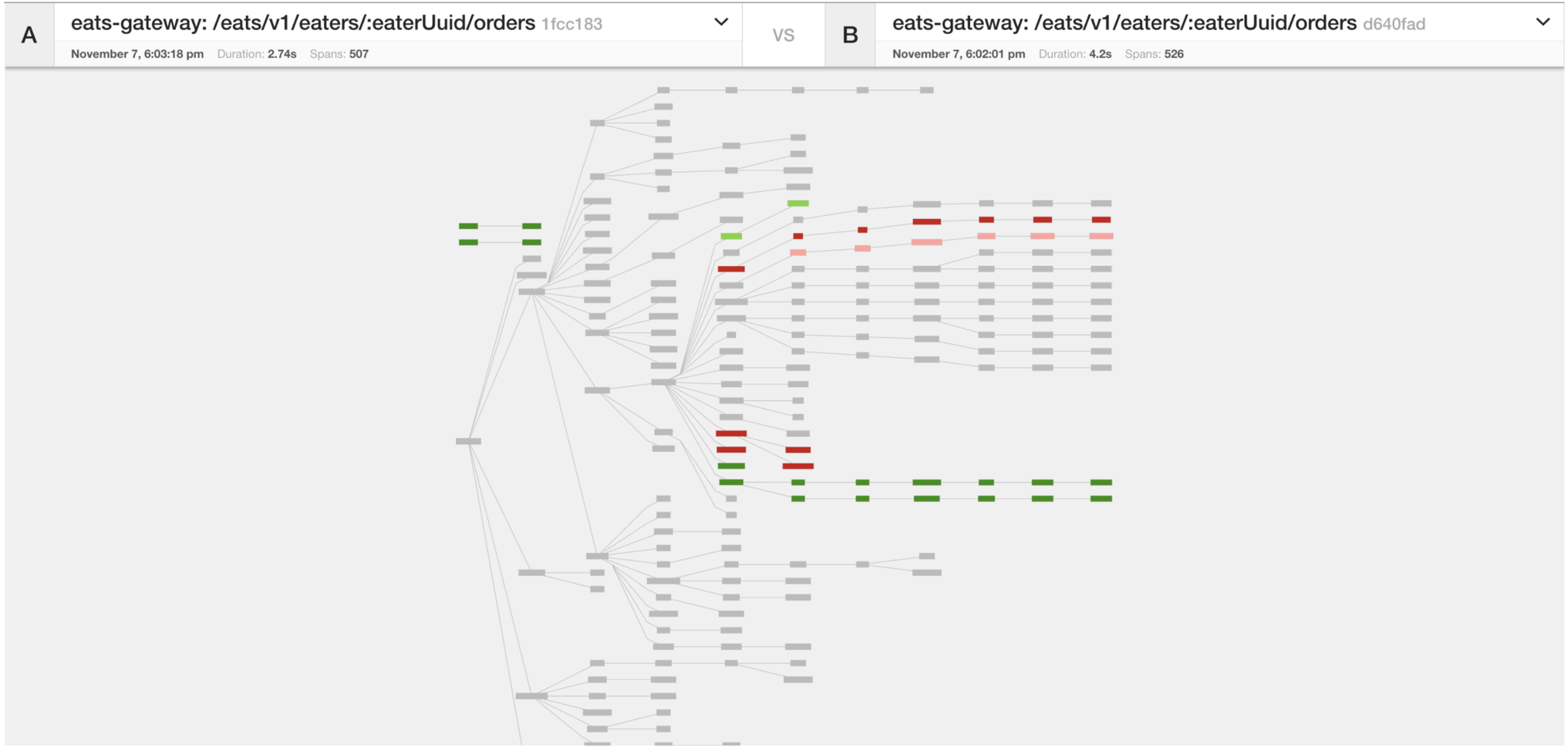
Production story

Migrating services to a nearby datacenter

Request latency doubles

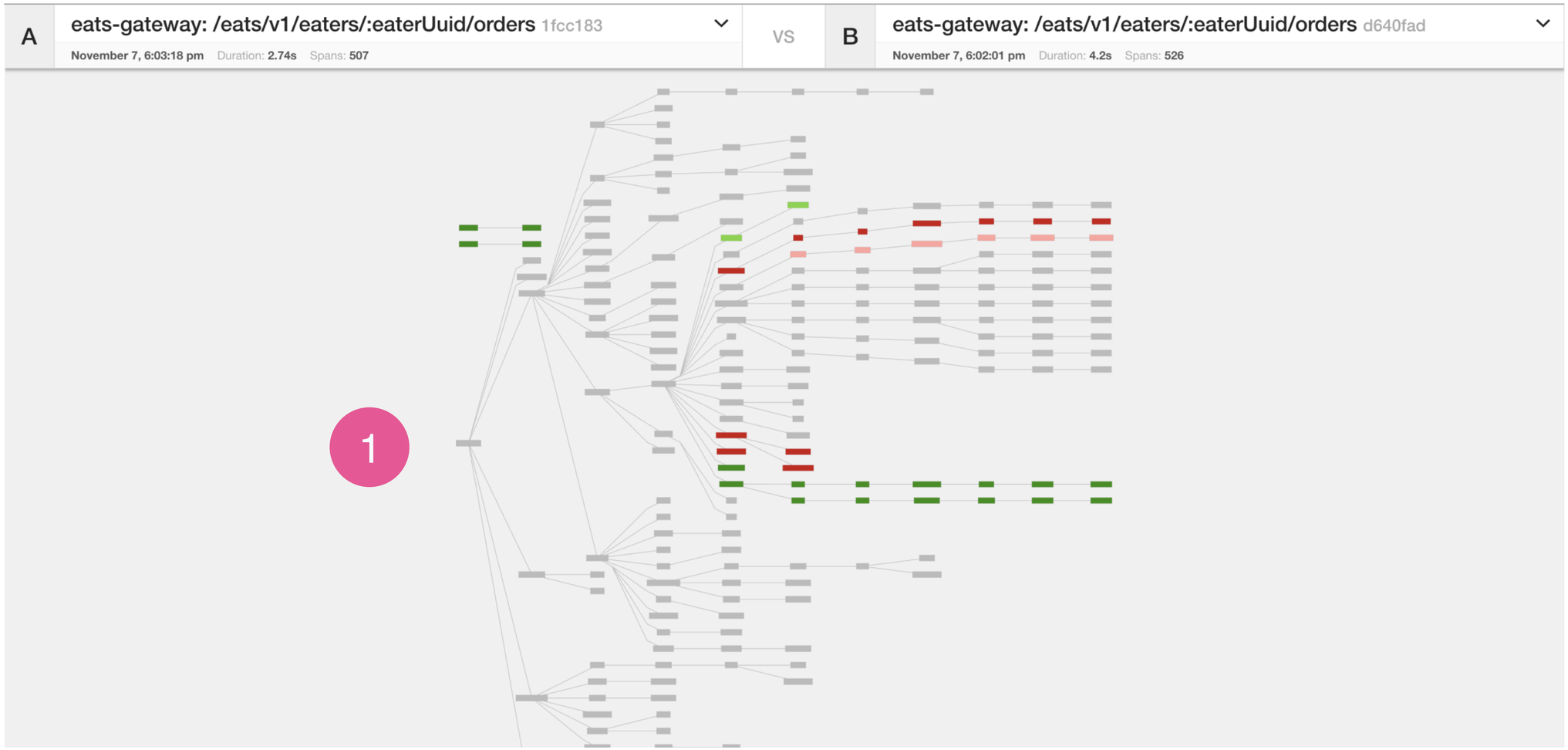
Investigating latency

Structural comparison not always useful



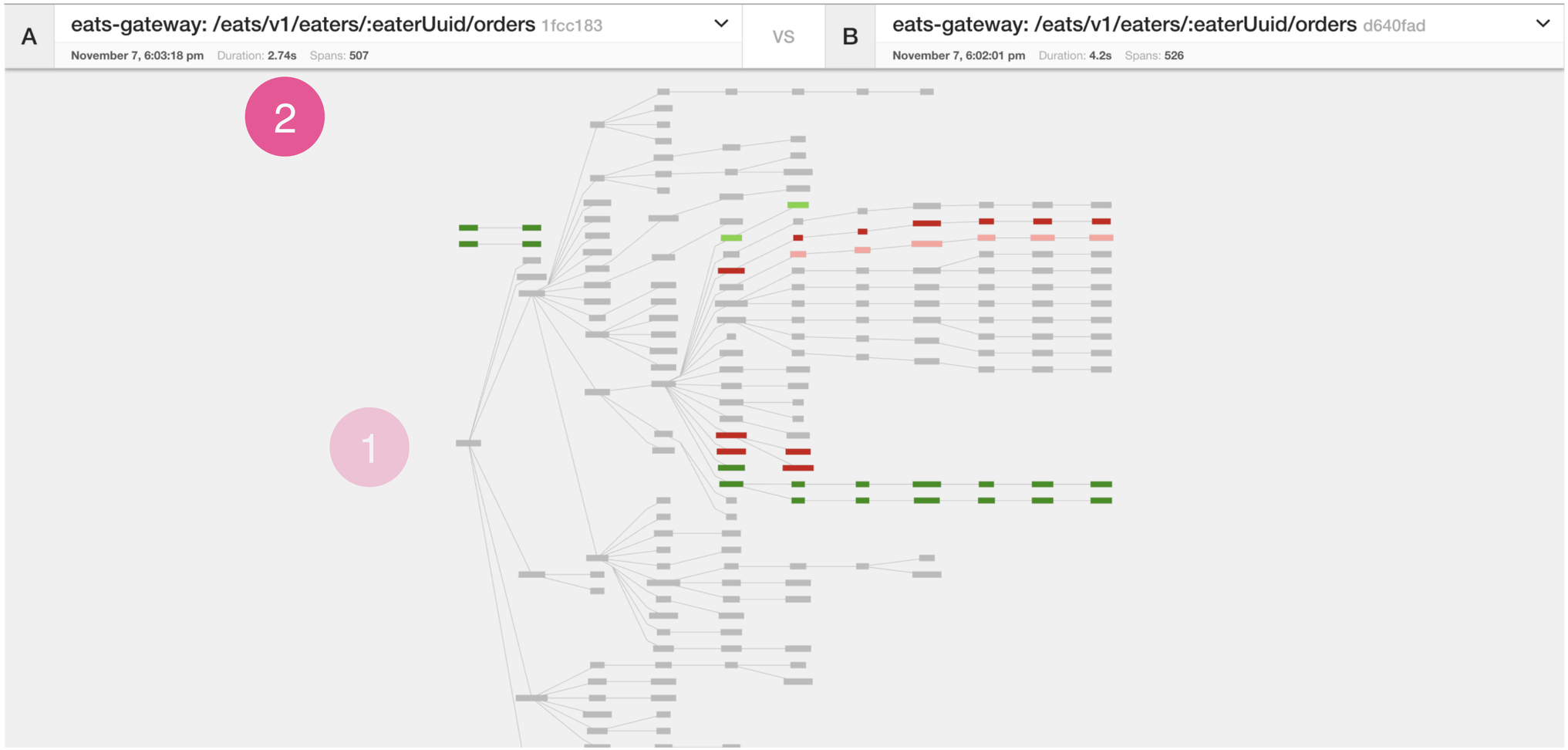
Investigating latency

Very similar structure



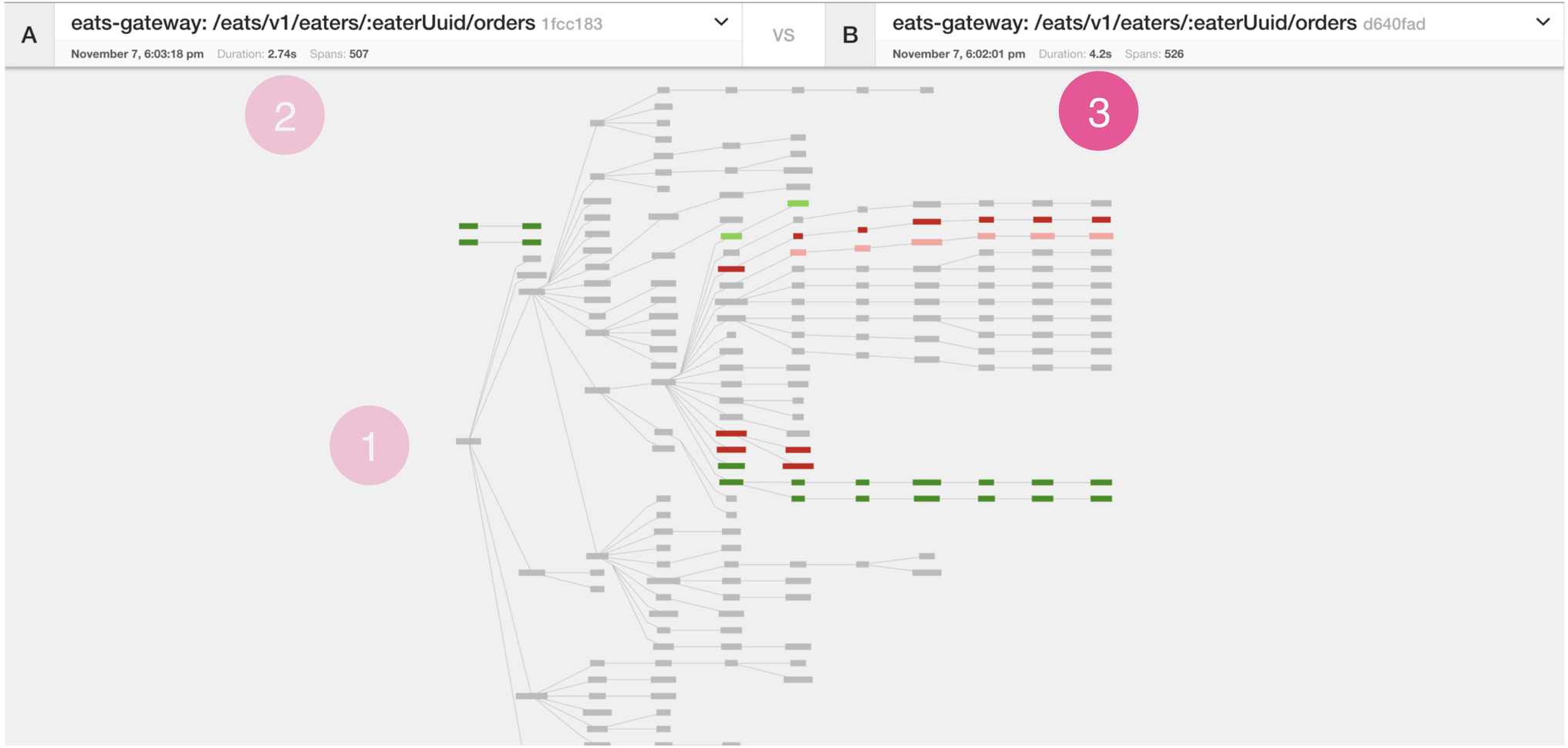
Investigating latency

Left trace 2.74 seconds



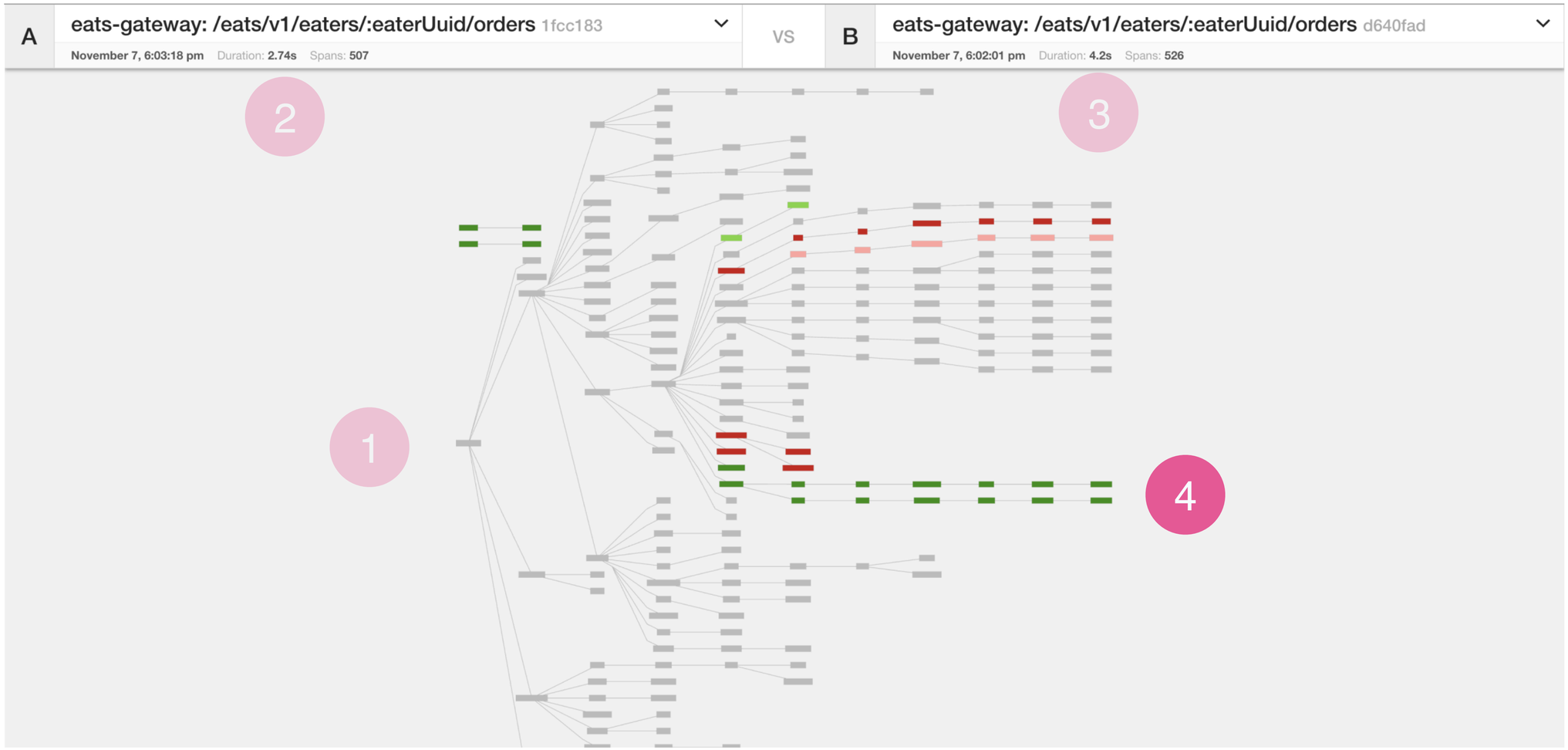
Investigating latency

Right trace 4.2 seconds



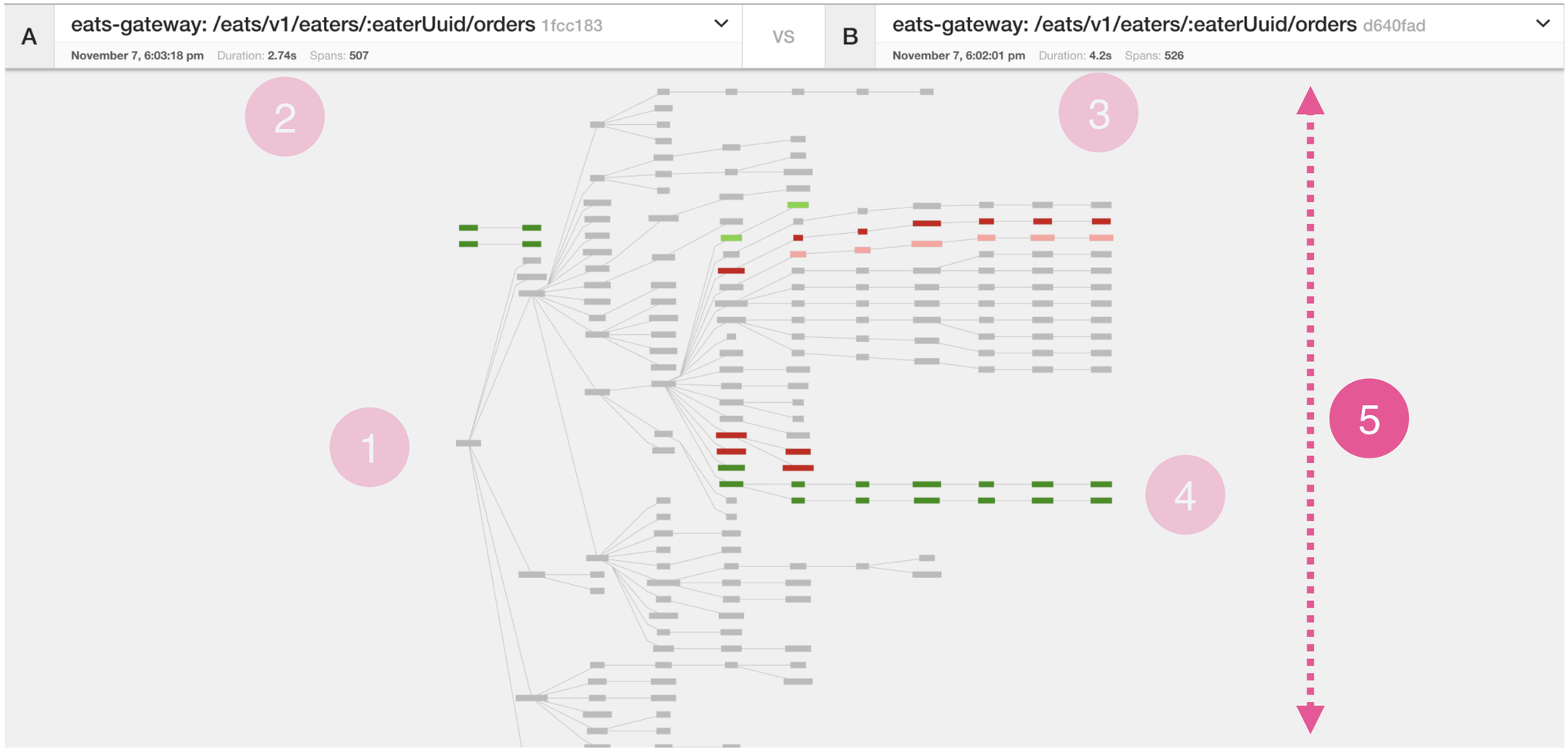
Investigating latency

Due to structural differences?



Investigating latency

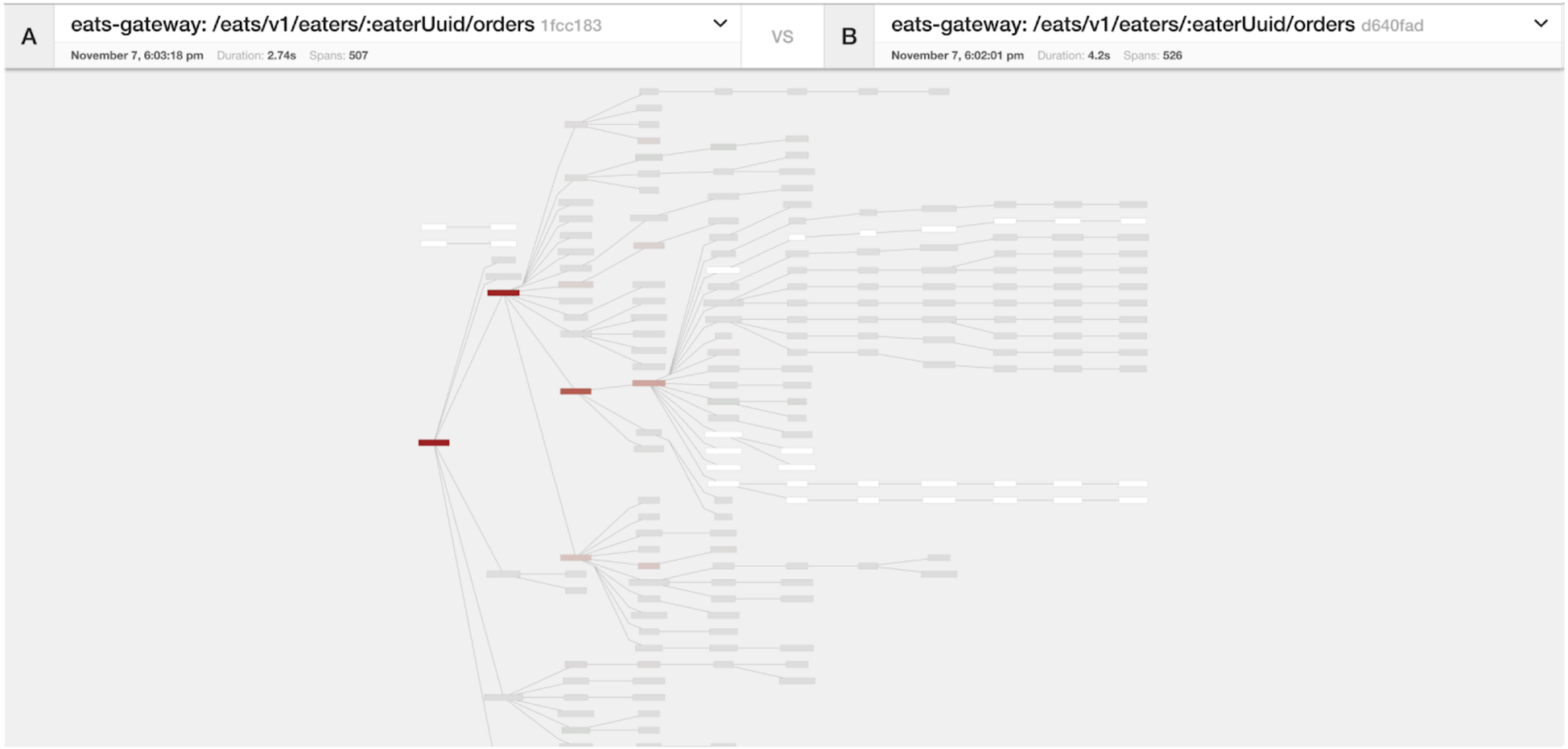
Or dispersed contributors?



Heat-maps!

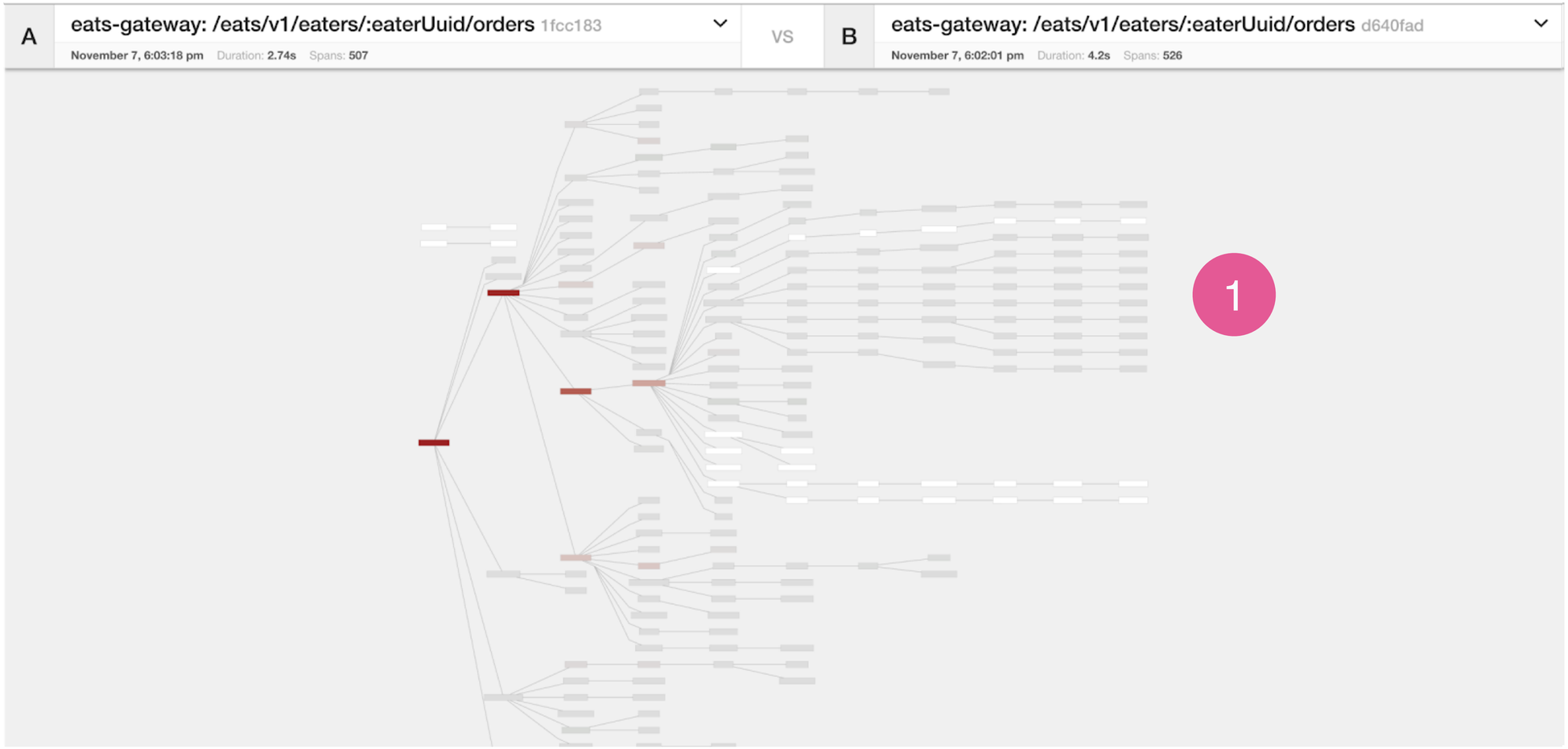
Comparing trace durations

Heat-map of latencies



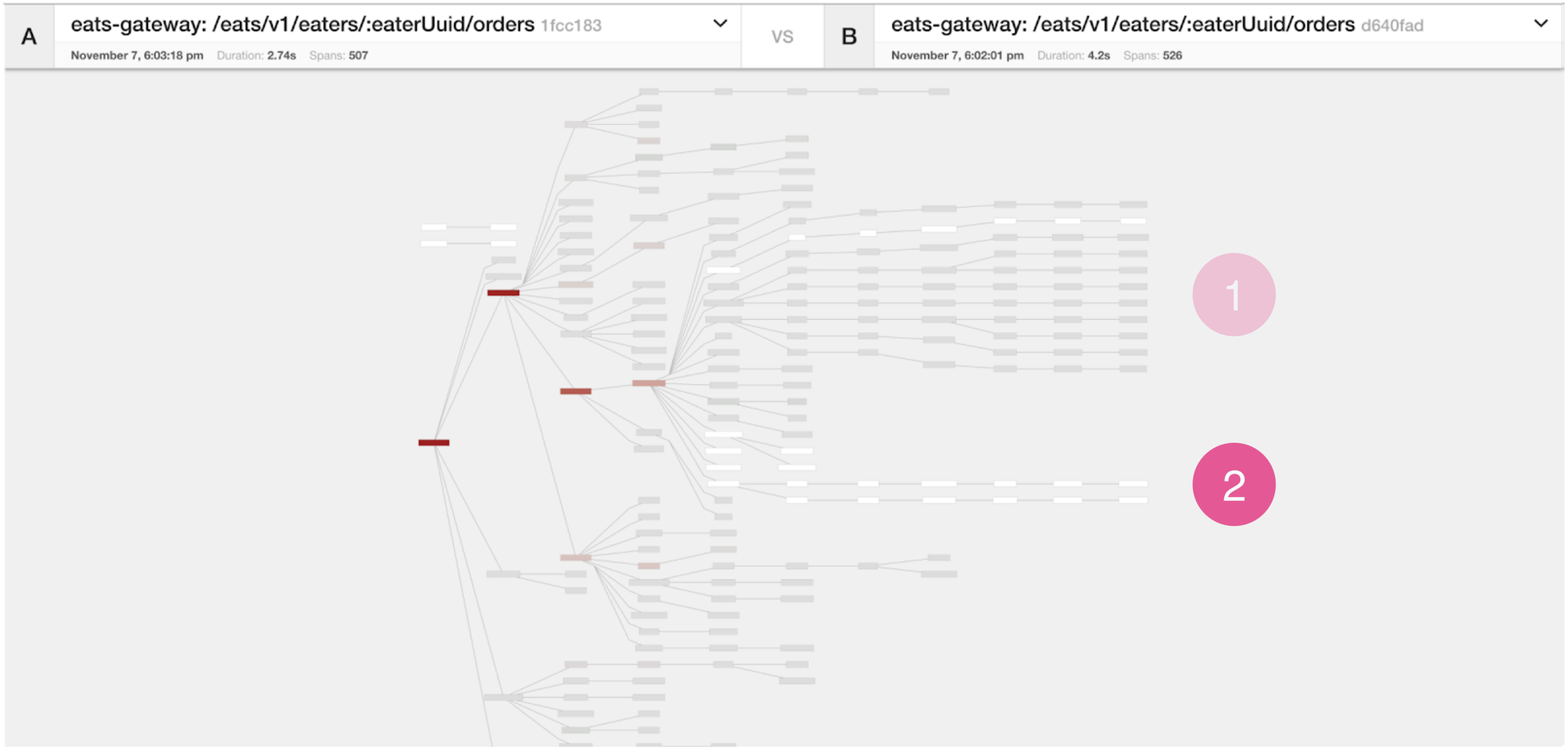
Comparing trace durations

Similar durations (grey)



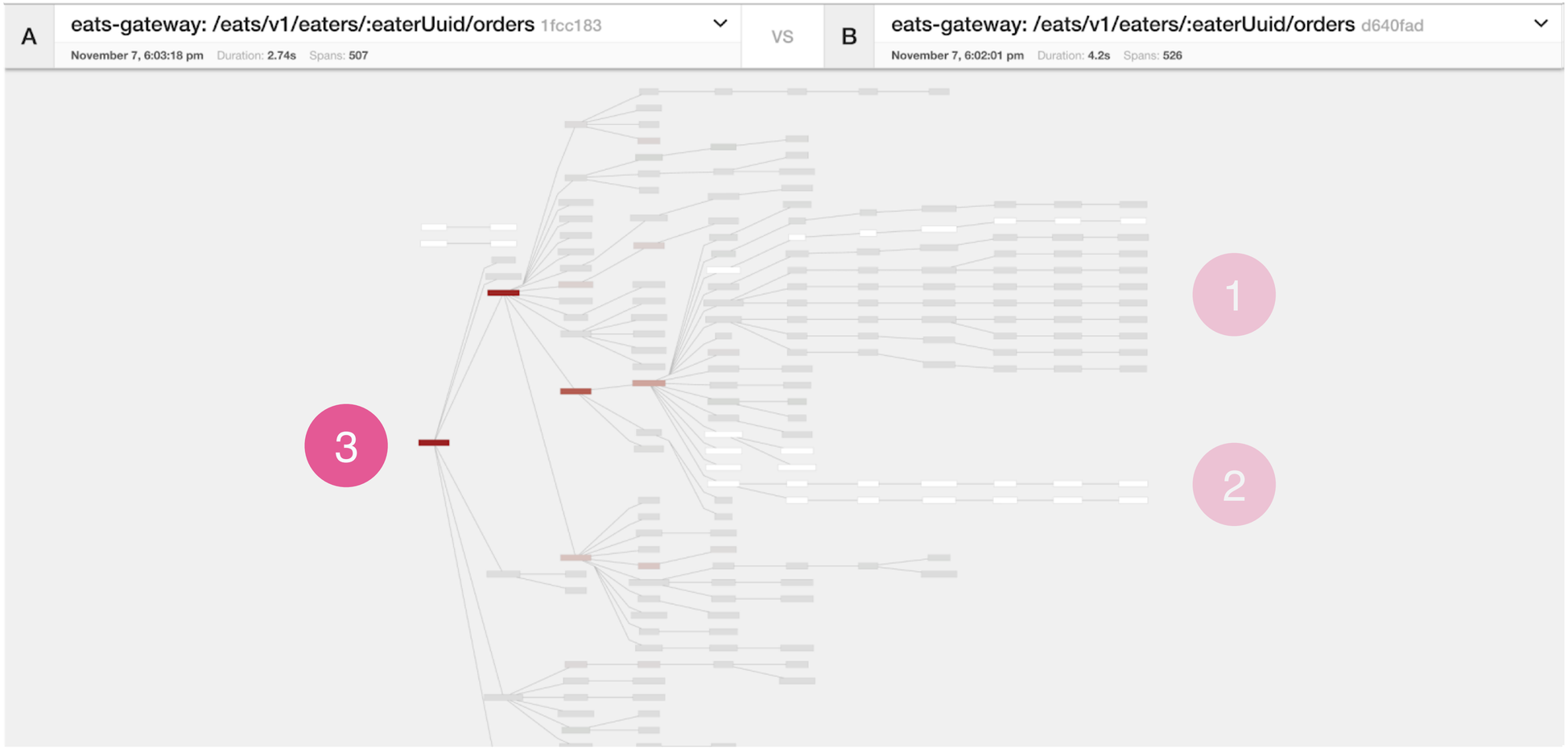
Comparing trace durations

Nodes that are not shared (white)



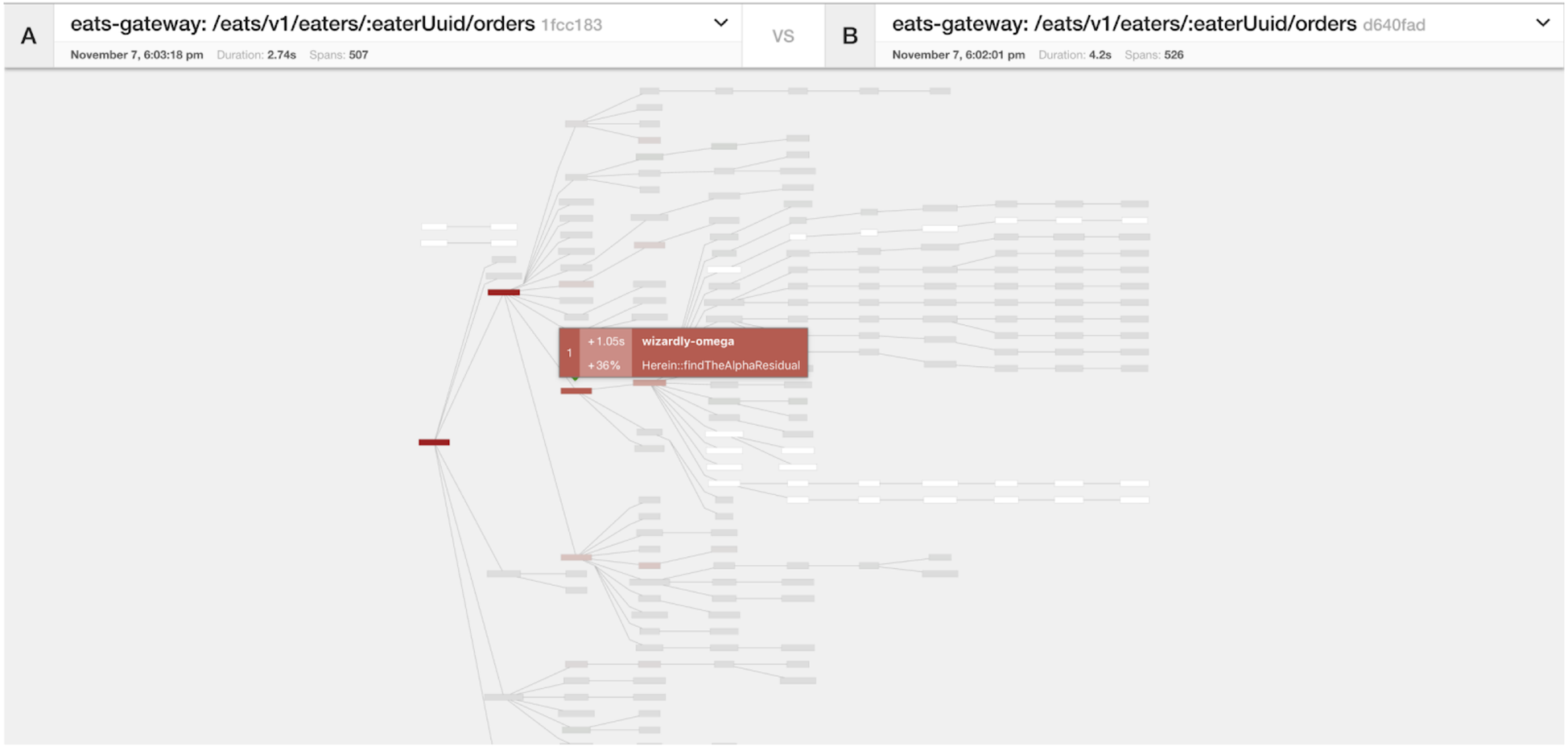
Comparing trace durations

Red heat-map for latency differences



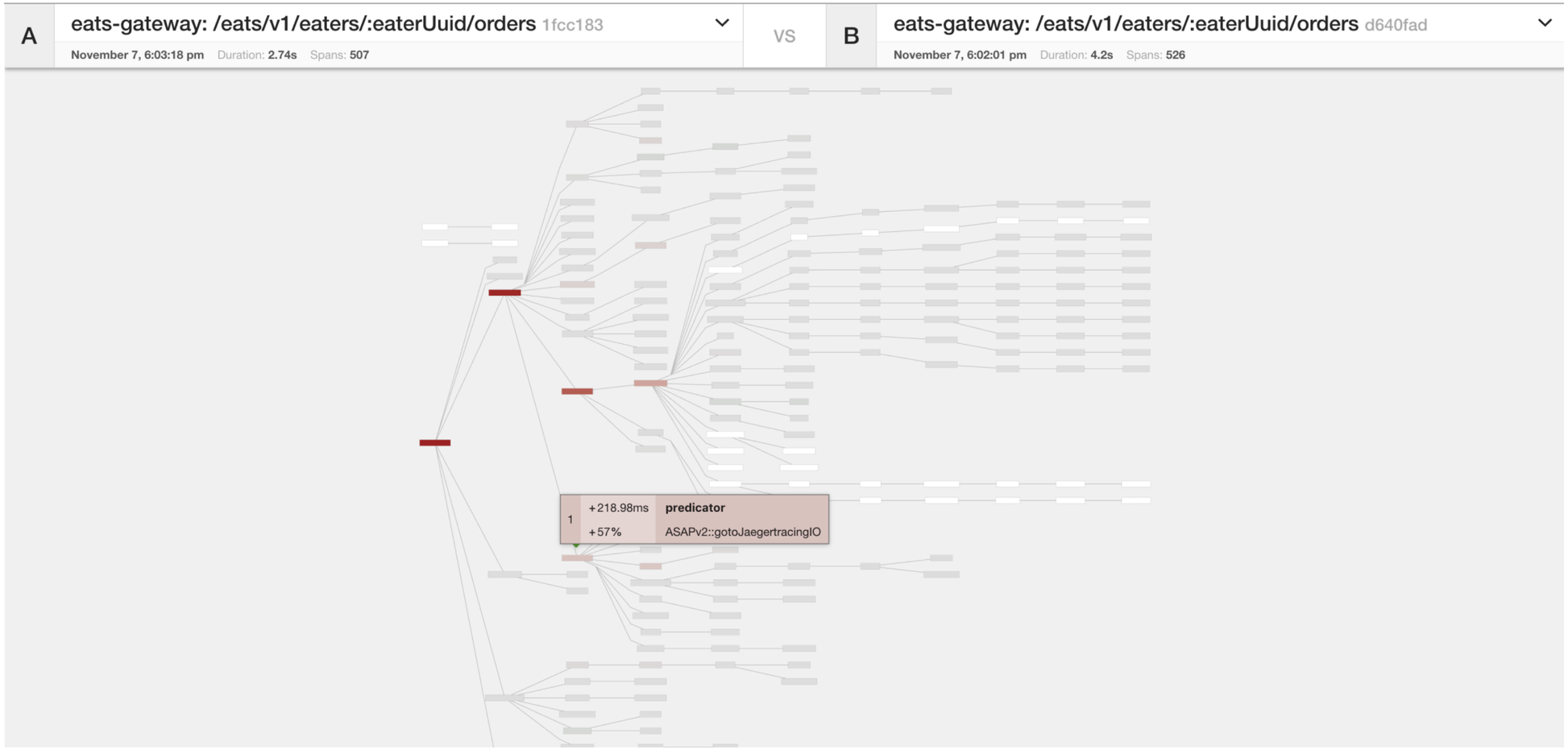
Comparing trace durations

Details on Mouse-Over



Comparing trace durations

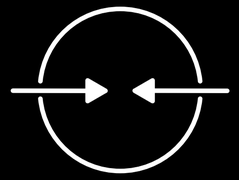
Details on Mouse-Over



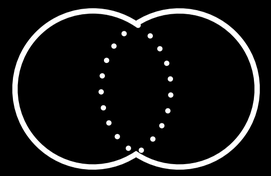
How Are These Approach Different?

Summary

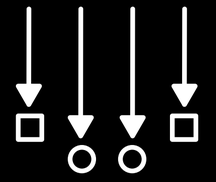
Surface less information



Condense the structural representation



Emphasize the differences



Distinct comparison modes simplify the comparisons

Challenges

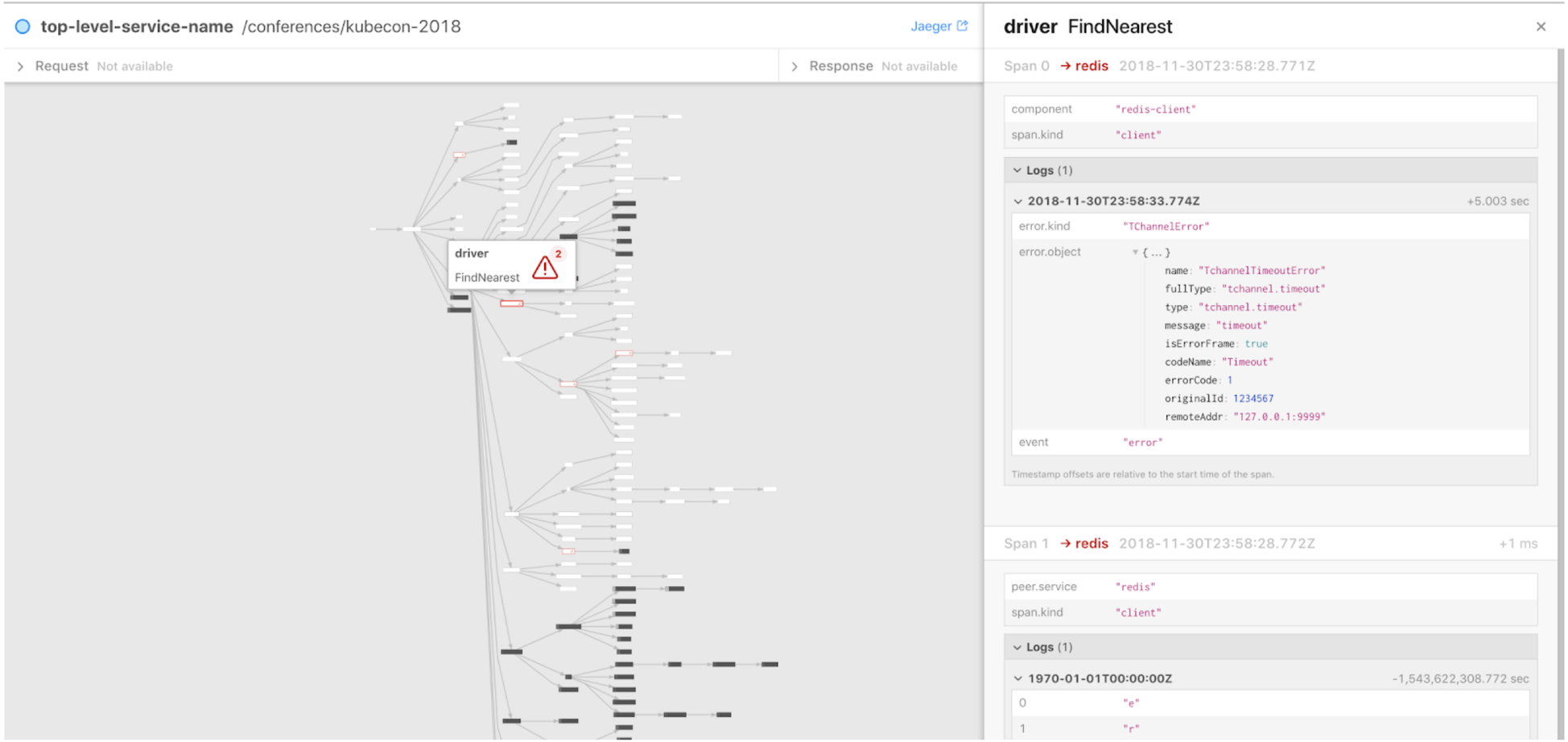
Individual traces can be an outliers.

User must find the right baseline.

Traces vs. Trace

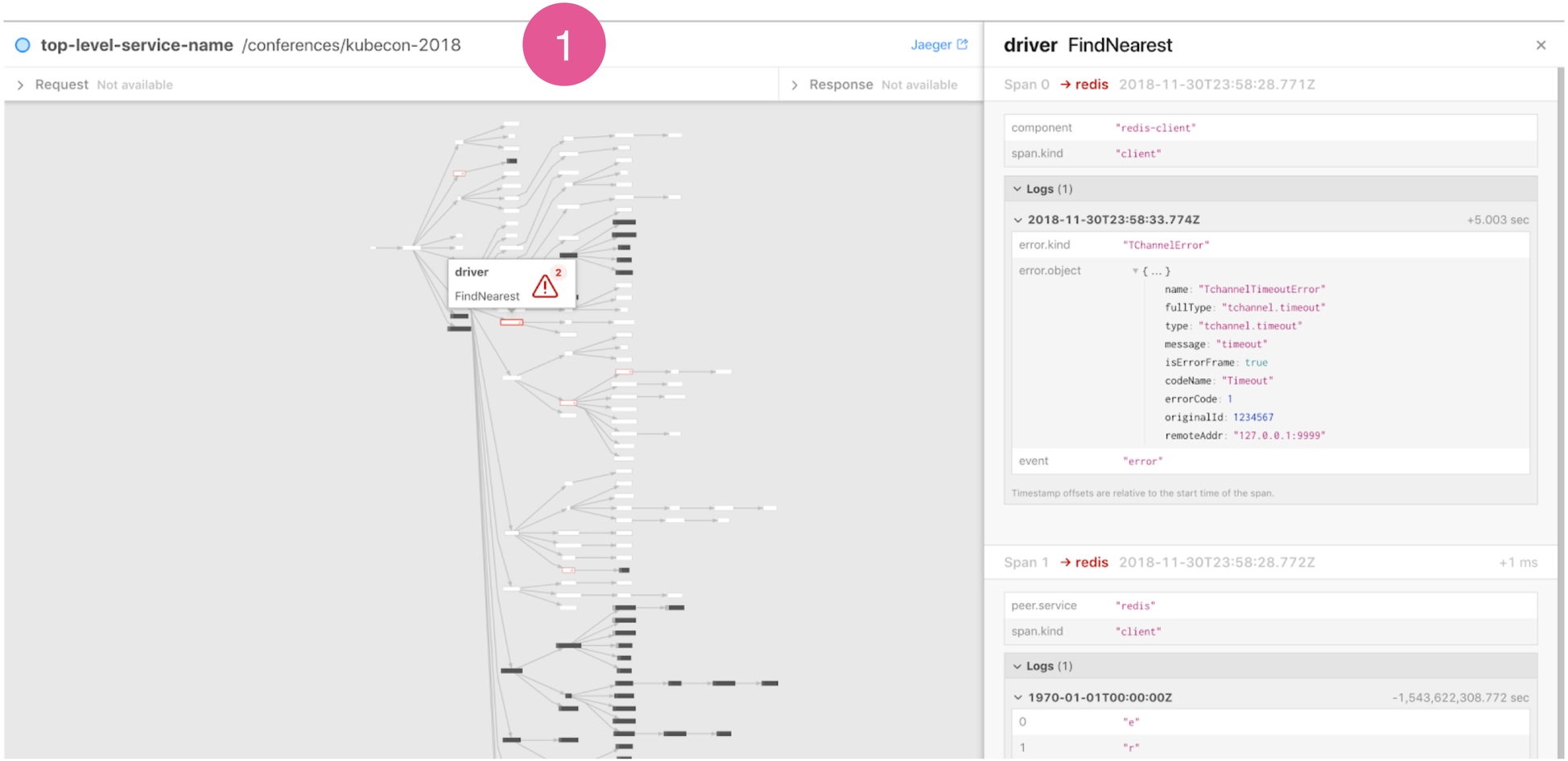
What Went Wrong?

Root Cause Analysis



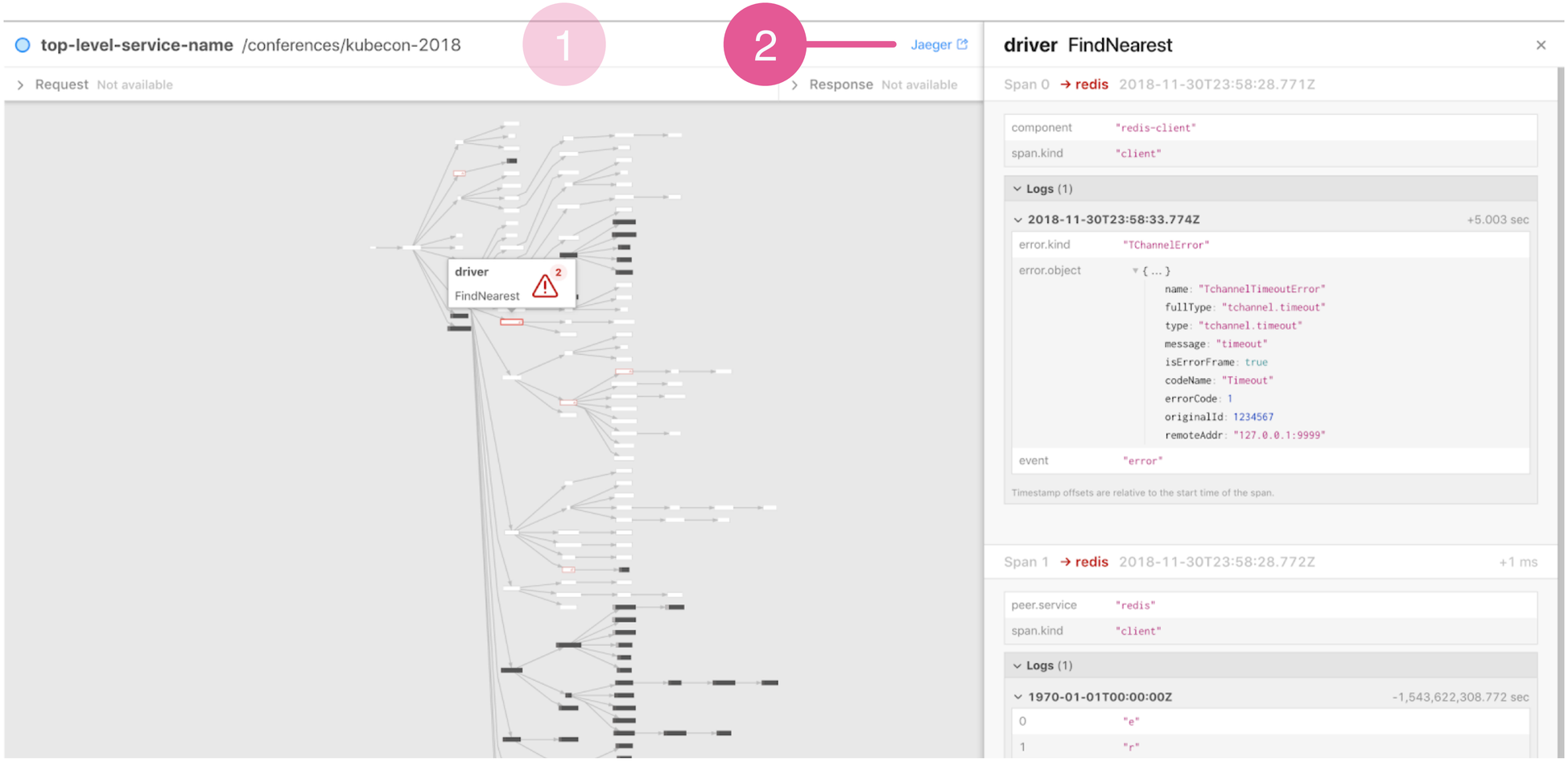
Top Level Outcome

Including Request/Response Payloads



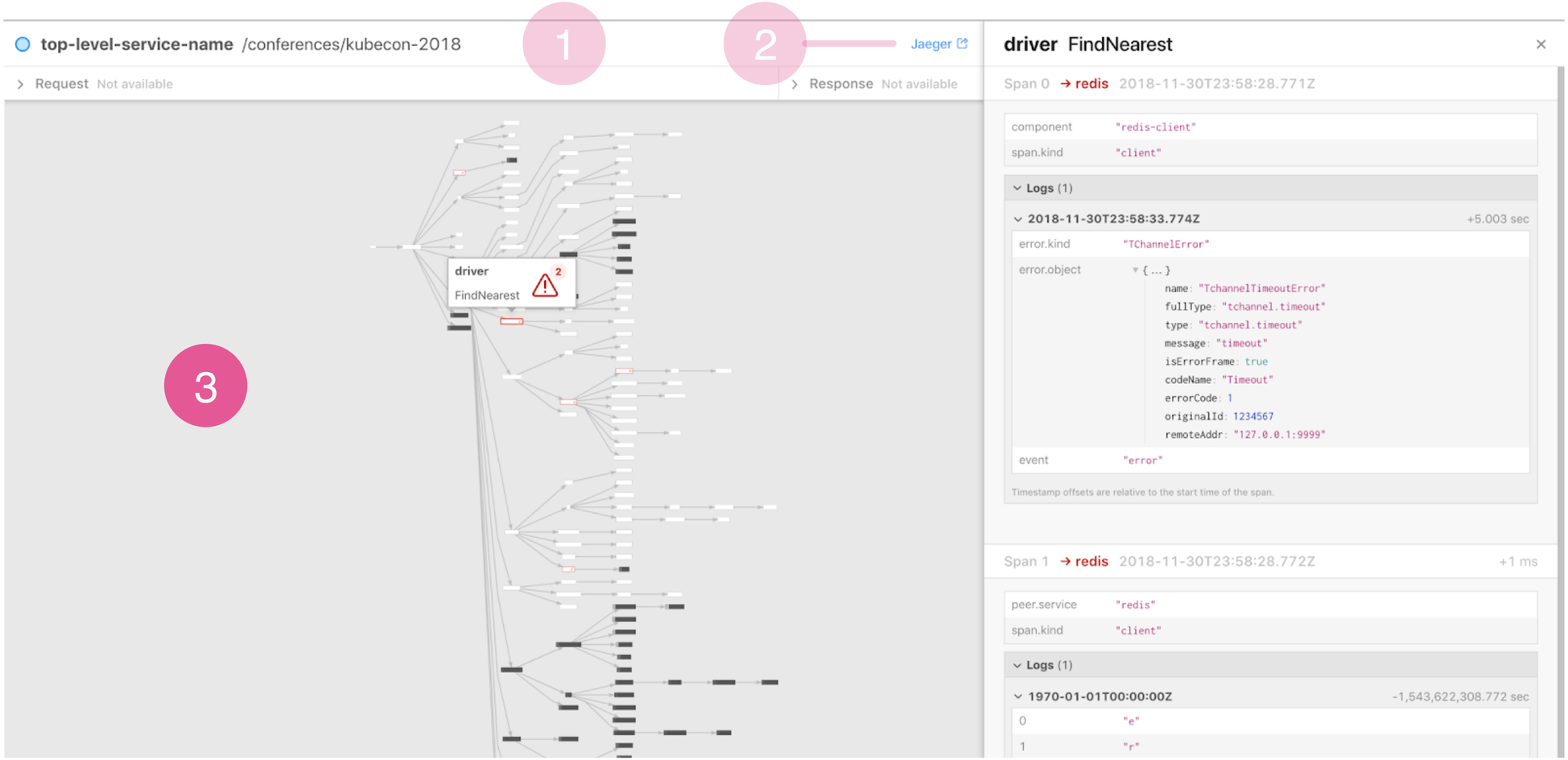
Link to the Trace

Can Always Go Back to Raw Data



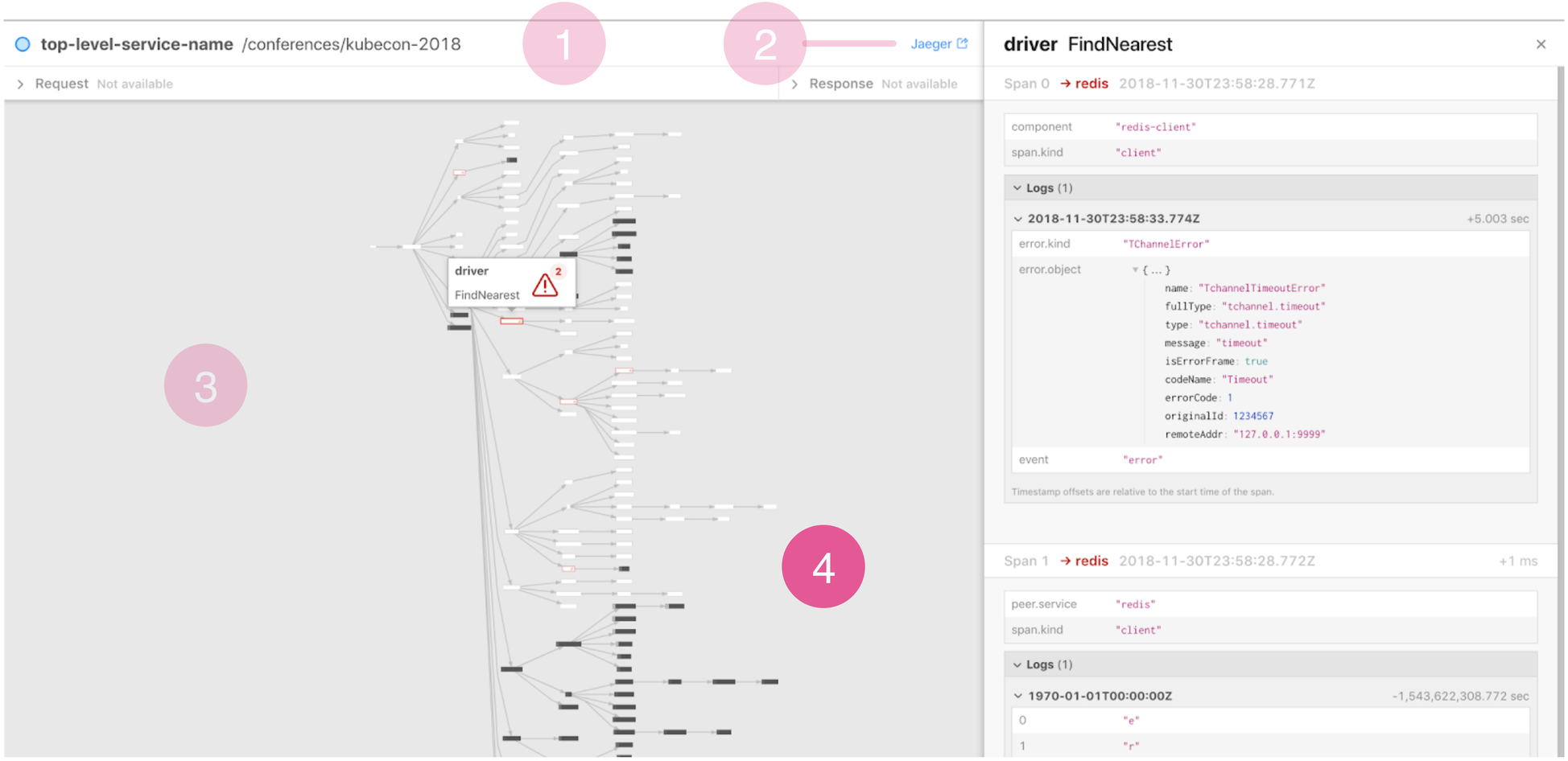
Trace Structure

Nodes Are Sorted Chronologically

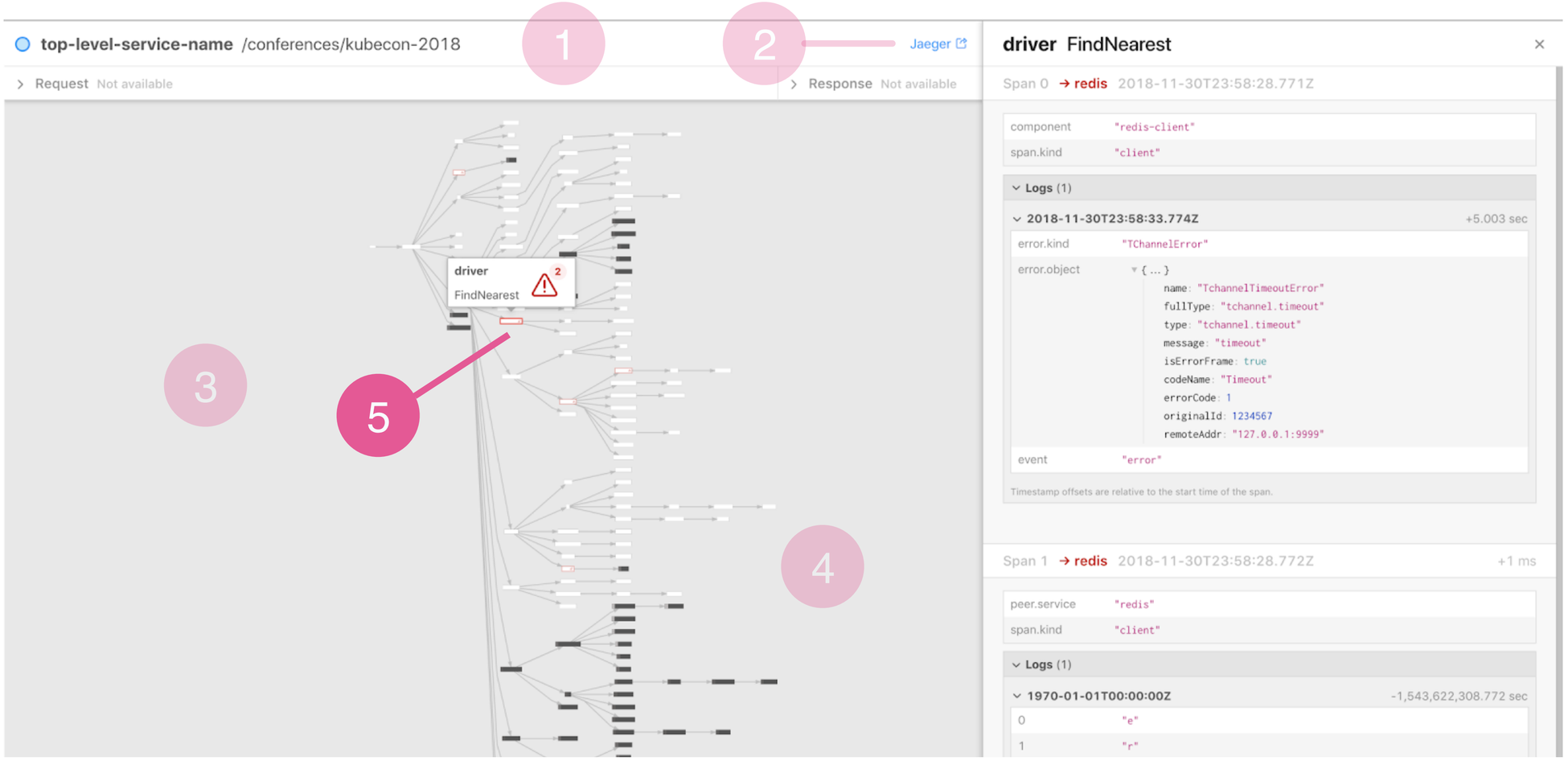


Present and Missing Nodes

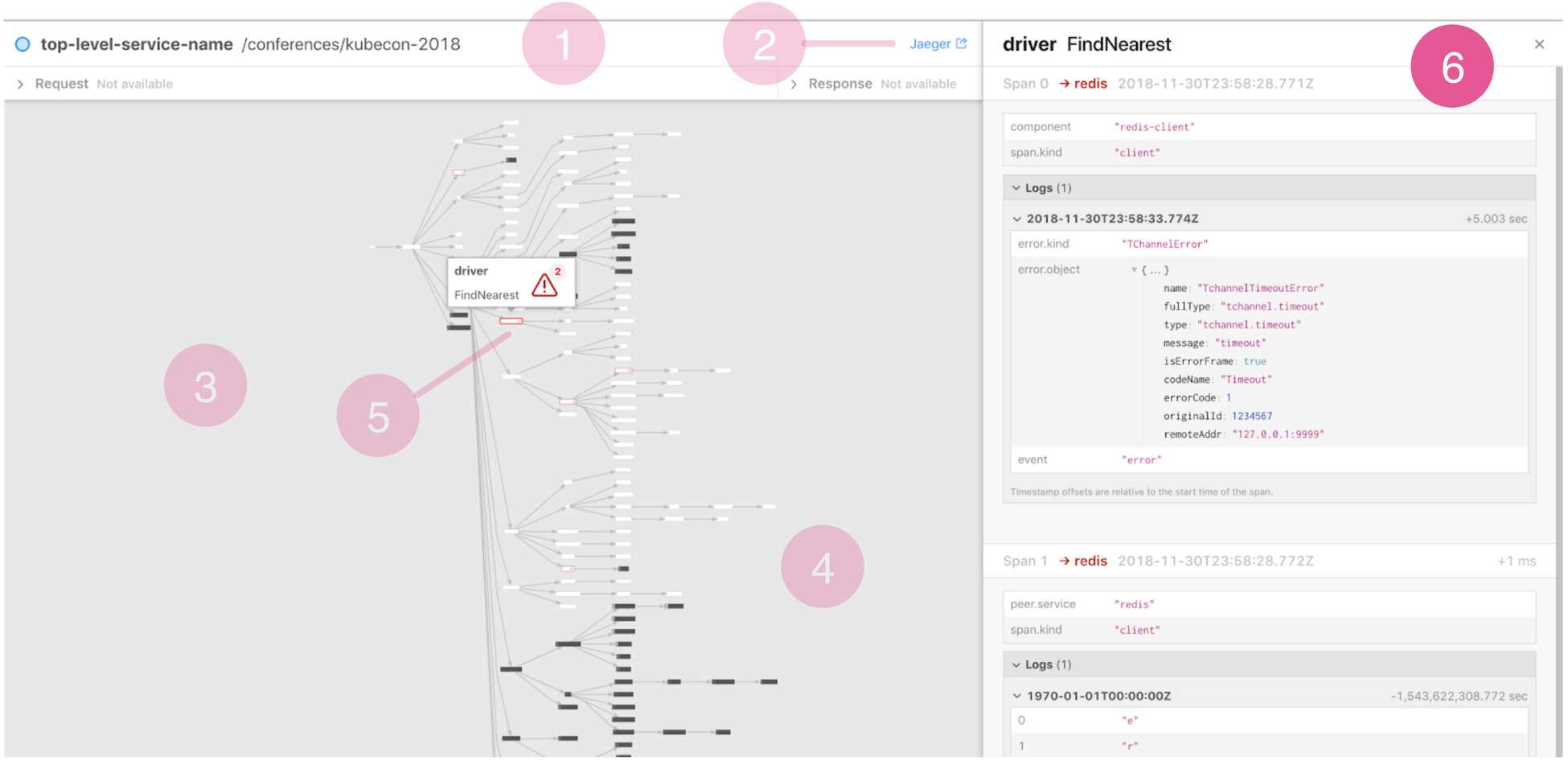
Color-Coding



A Node With Error Data

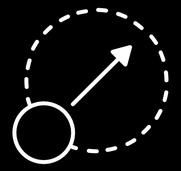


Error Data Panel



How Is This Approach Different?

Summary



Much broader context: aggregate vs. one trace

One purpose: root cause analysis of reliability issues

Tackling Data Complexity

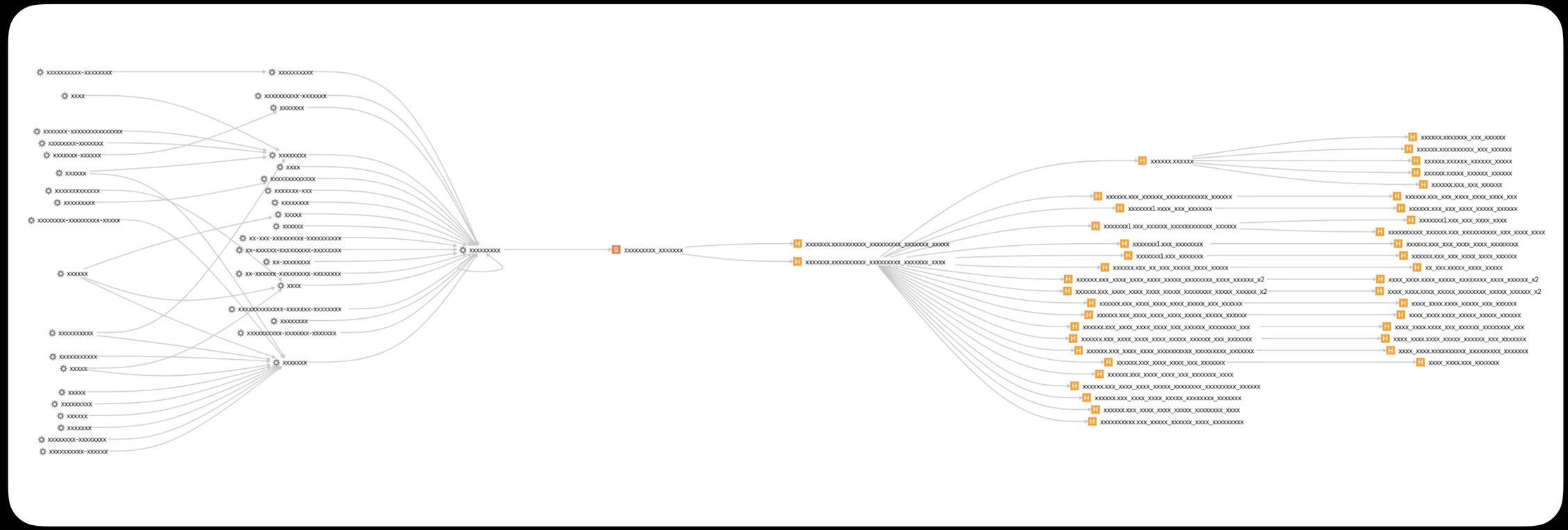
Uber is a data company

OK, and a transportation company

- Data undergoes many transformations
- More data is derived from other data
- Debugging data quality is difficult

Data Lineage

Debugging Data Quality

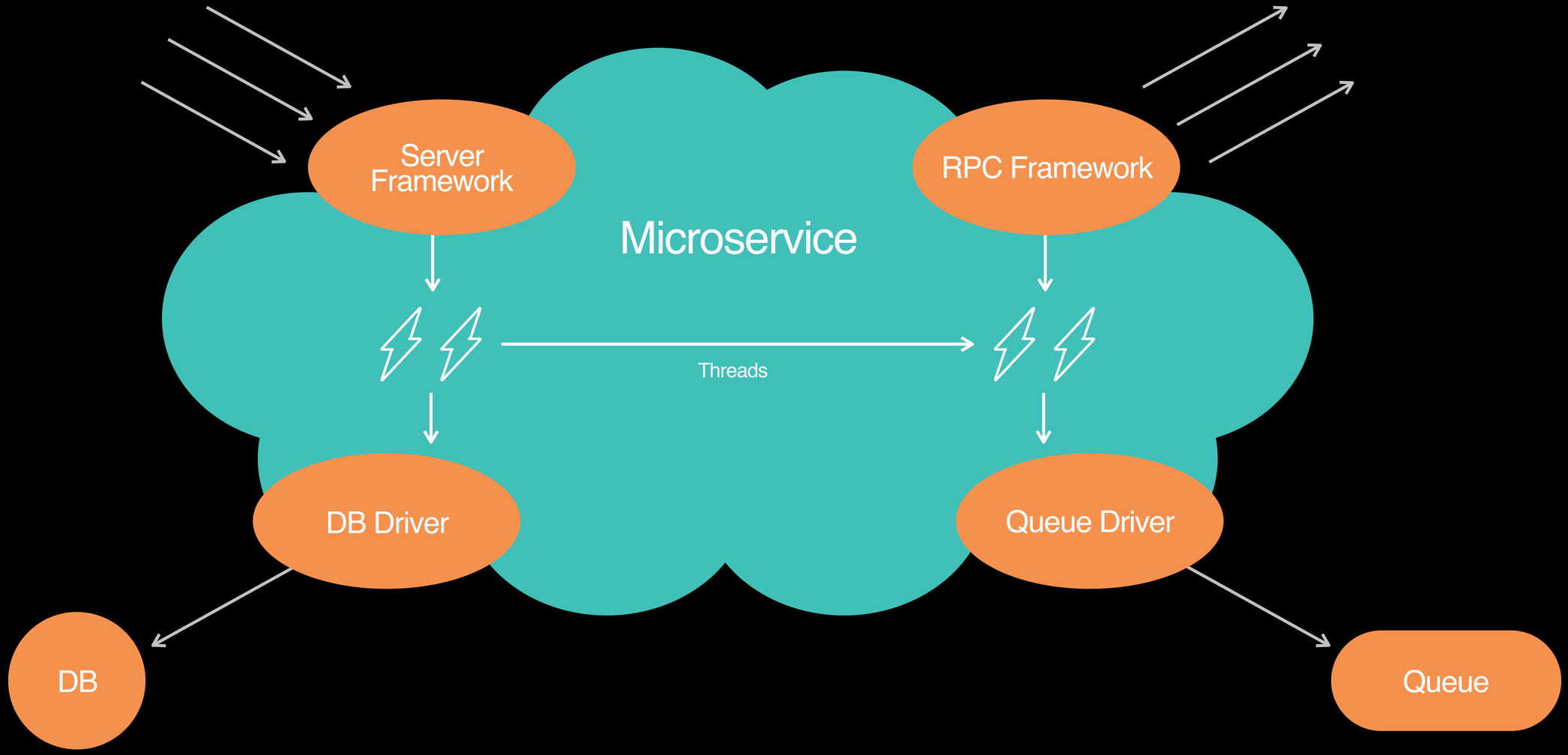


Microservices / RPCs → Streams / Kafka → Data lake / HDFS

Observability requires
high quality instrumentation.

Our Software Is Highly Composable

Often from Open Source Components



Tracing breaks if components
don’t understand each other.

Standardization Efforts

Instrumentation and Data Formats

- Effective observability requires high-quality telemetry.
- OpenTelemetry makes robust, portable telemetry a built-in feature of cloud-native software.

- Distributed Tracing Working Group
- Data formats for on-the-wire trace context & correlation-context, and out-of-band trace data.

In Summary

Distributed tracing helps us
to deal with the overwhelming
complexity of microservices

In Summary

Creative visualizations
are essential
in performance analysis

In Summary

Distributed tracing empowers
unparalleled insights
into our distributed systems

Thank You

Find me @ shkuro.com

Q&A