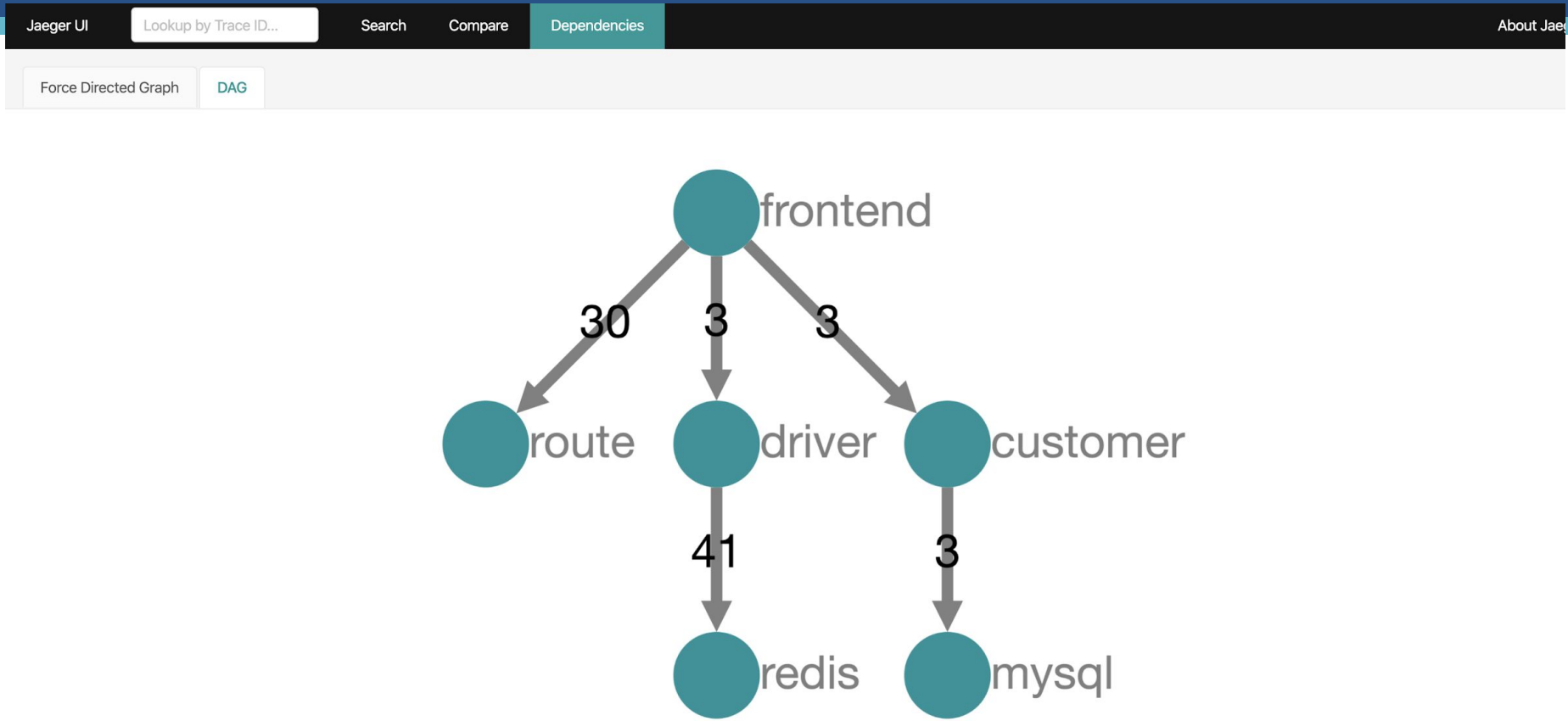
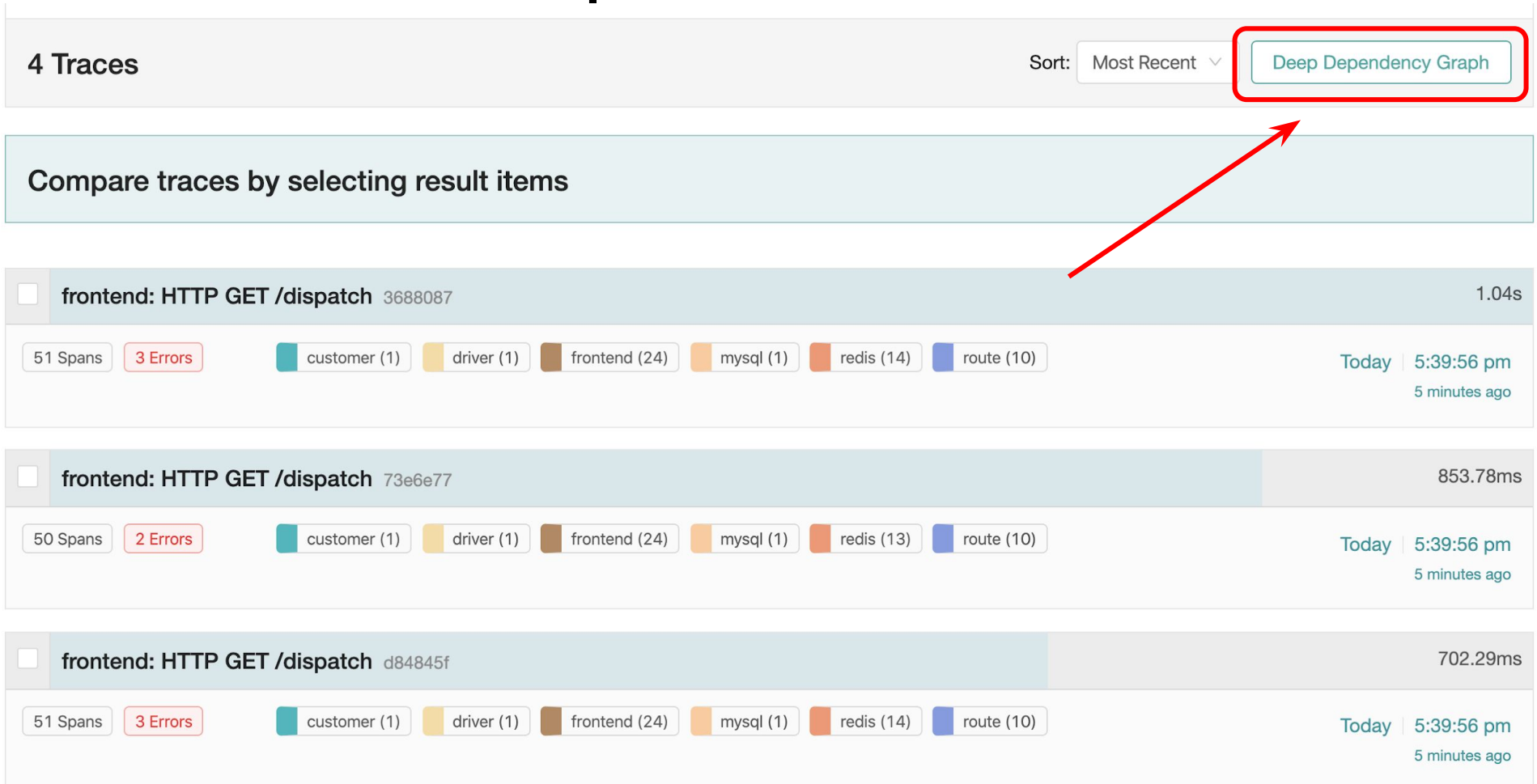
Let's look at some traces

http://bit.do/jaeger-hotrod

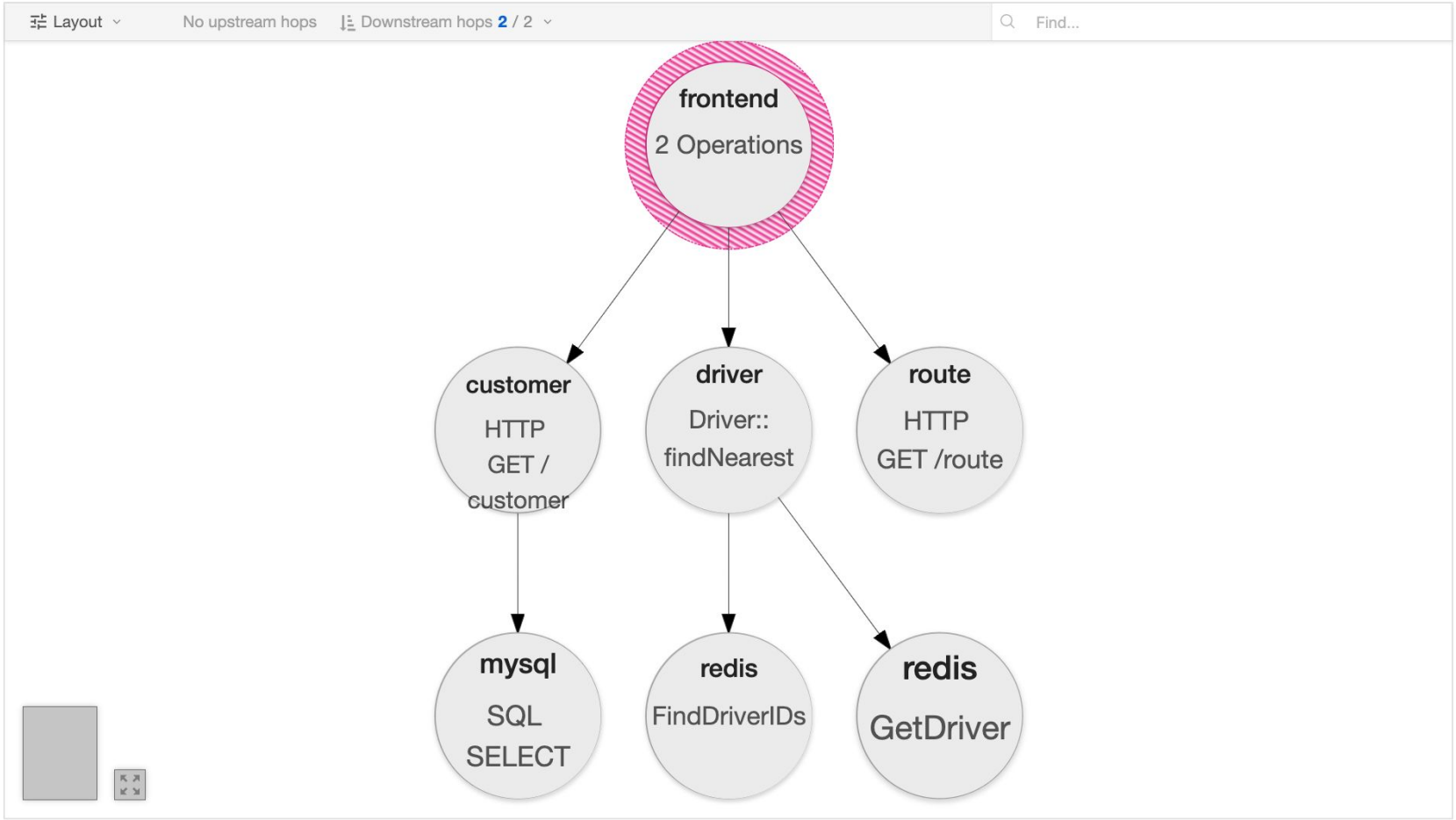
Service dependencies diagram



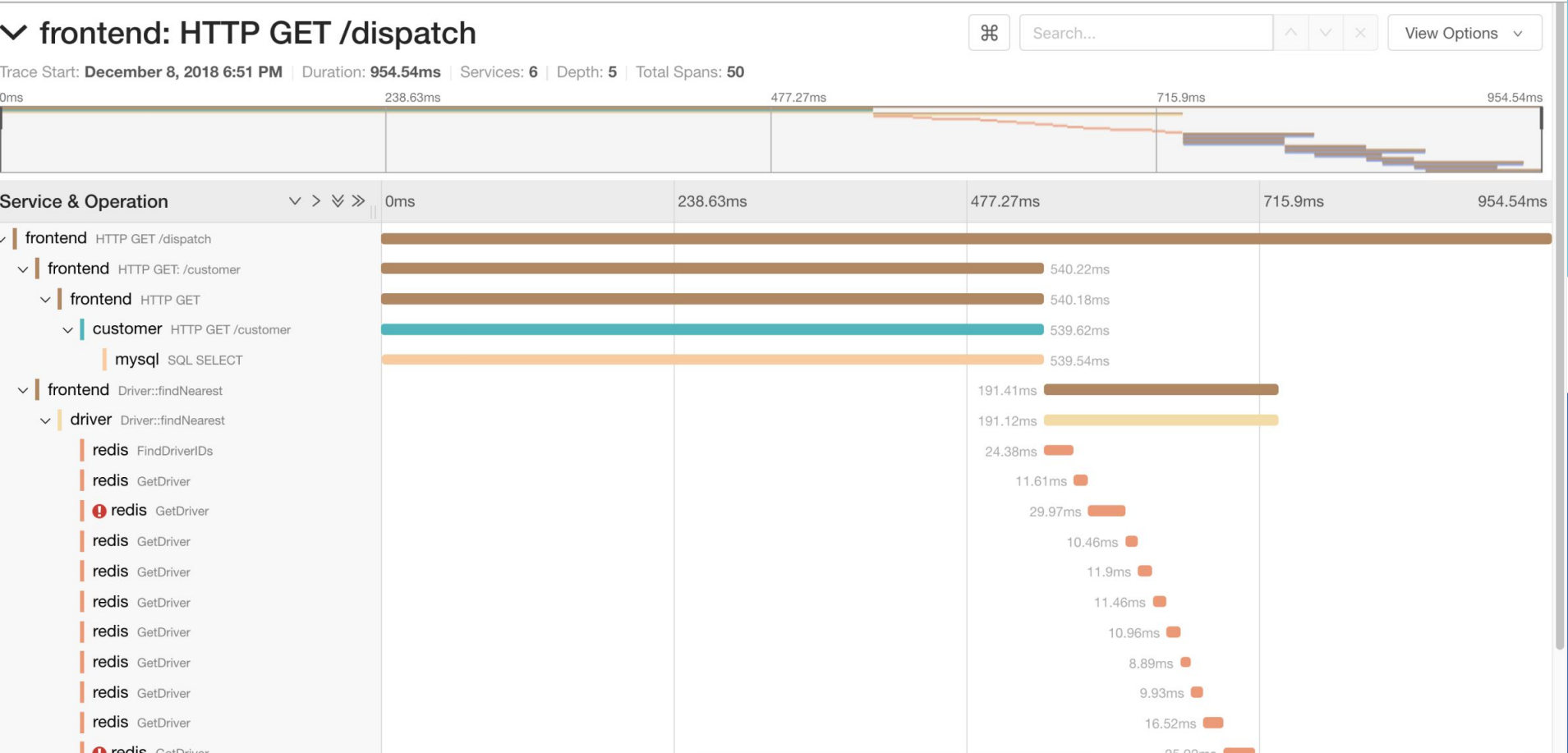
Transitive Service Graphs



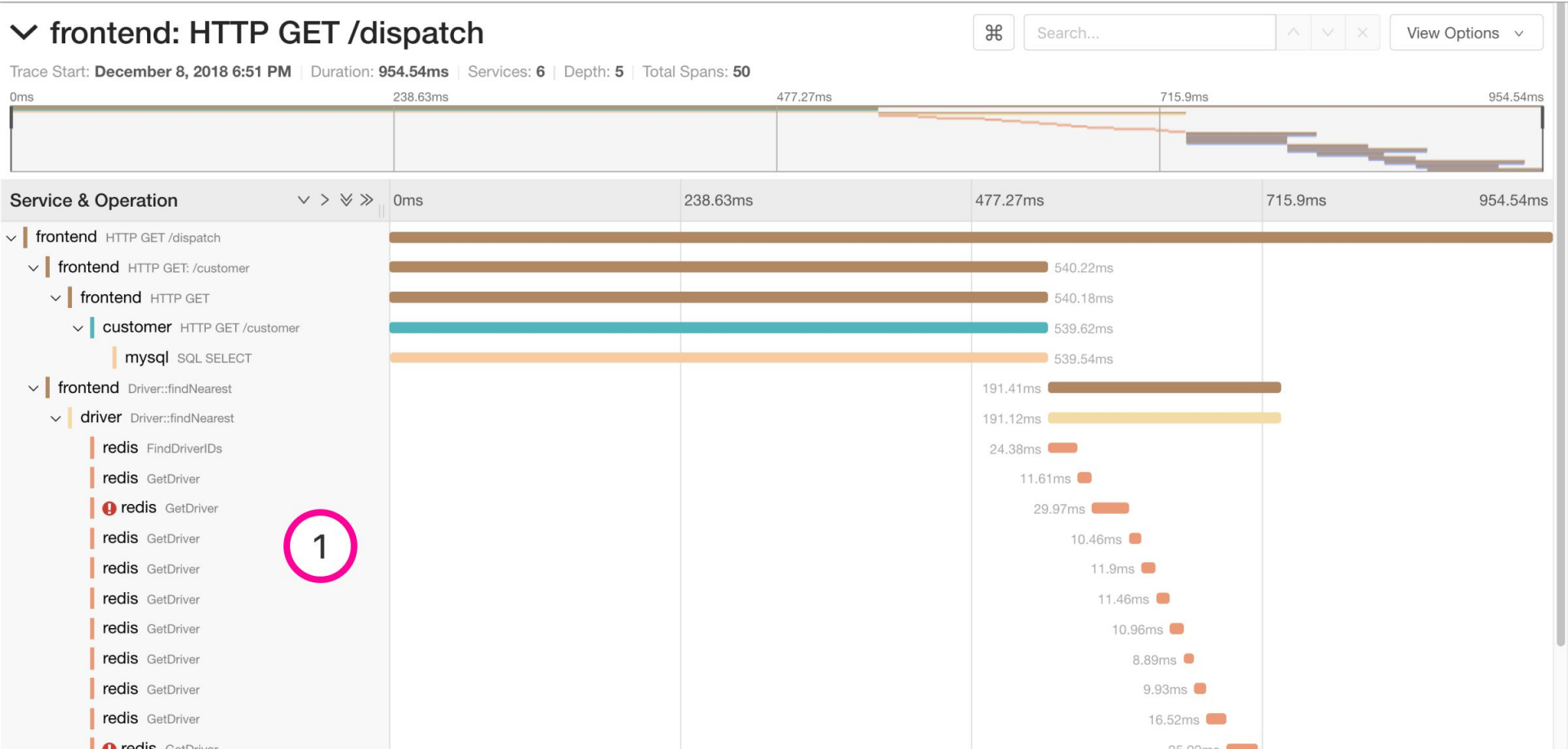
Transitive Service Graphs



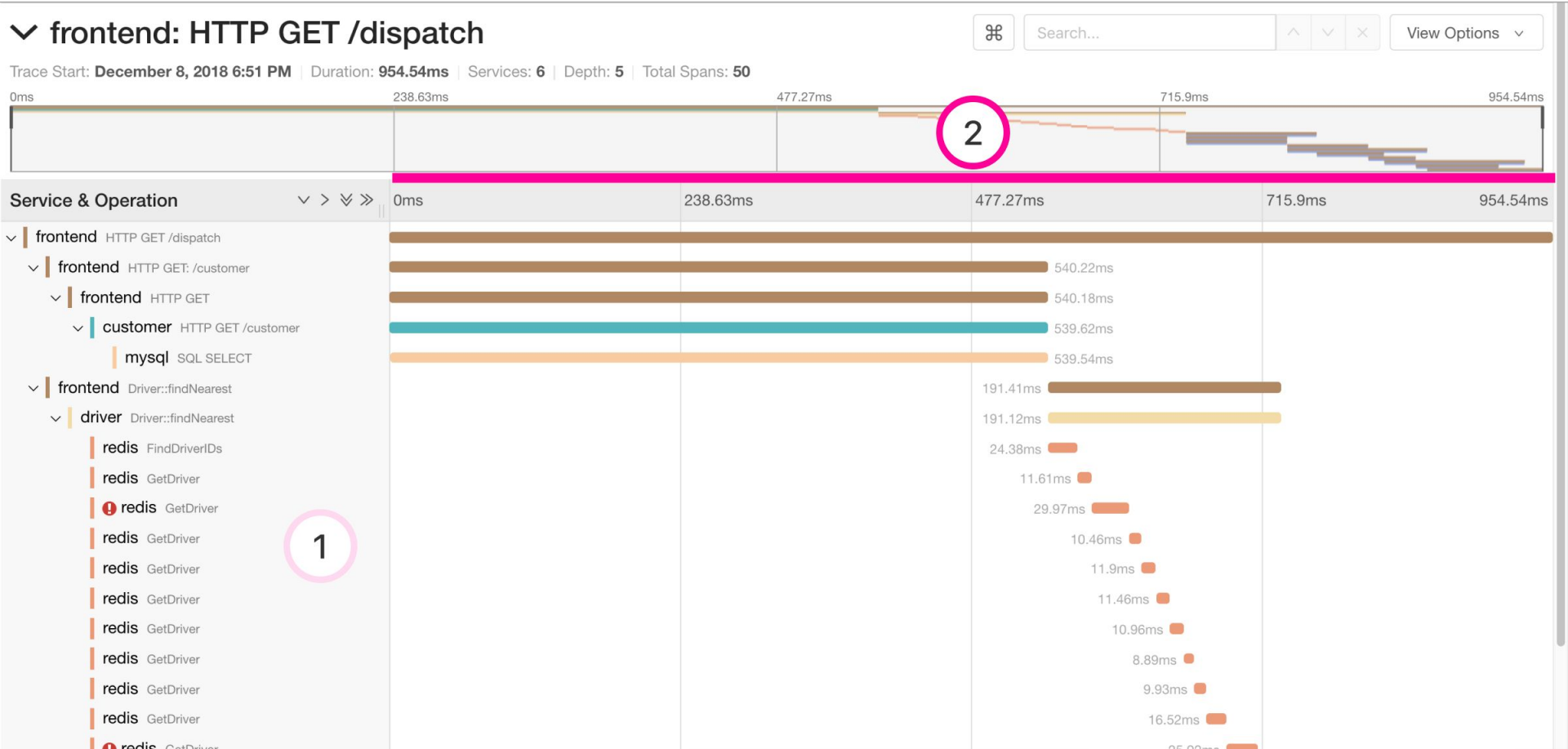
Trace timeline



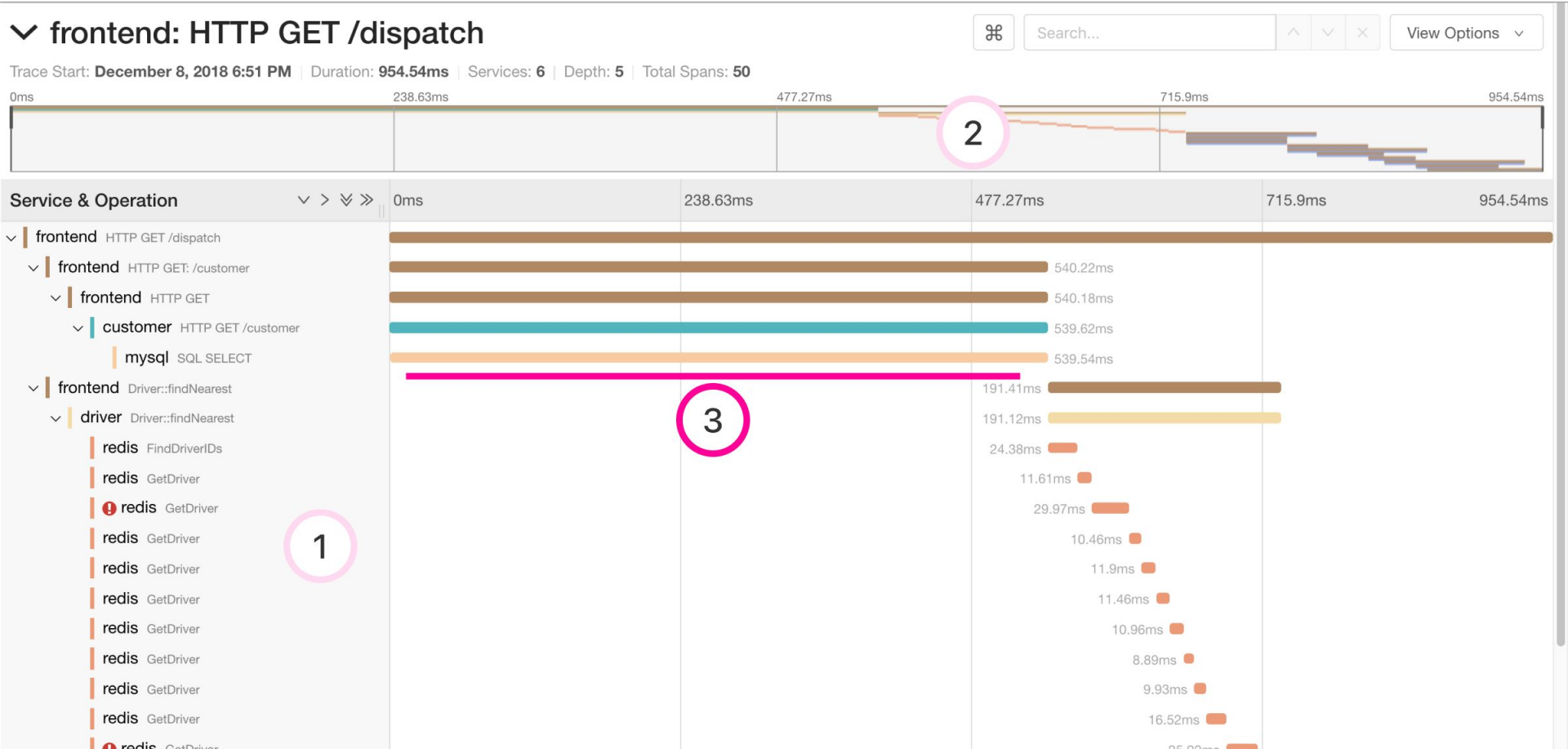
Trace timeline – Parent → Child → Grandchild



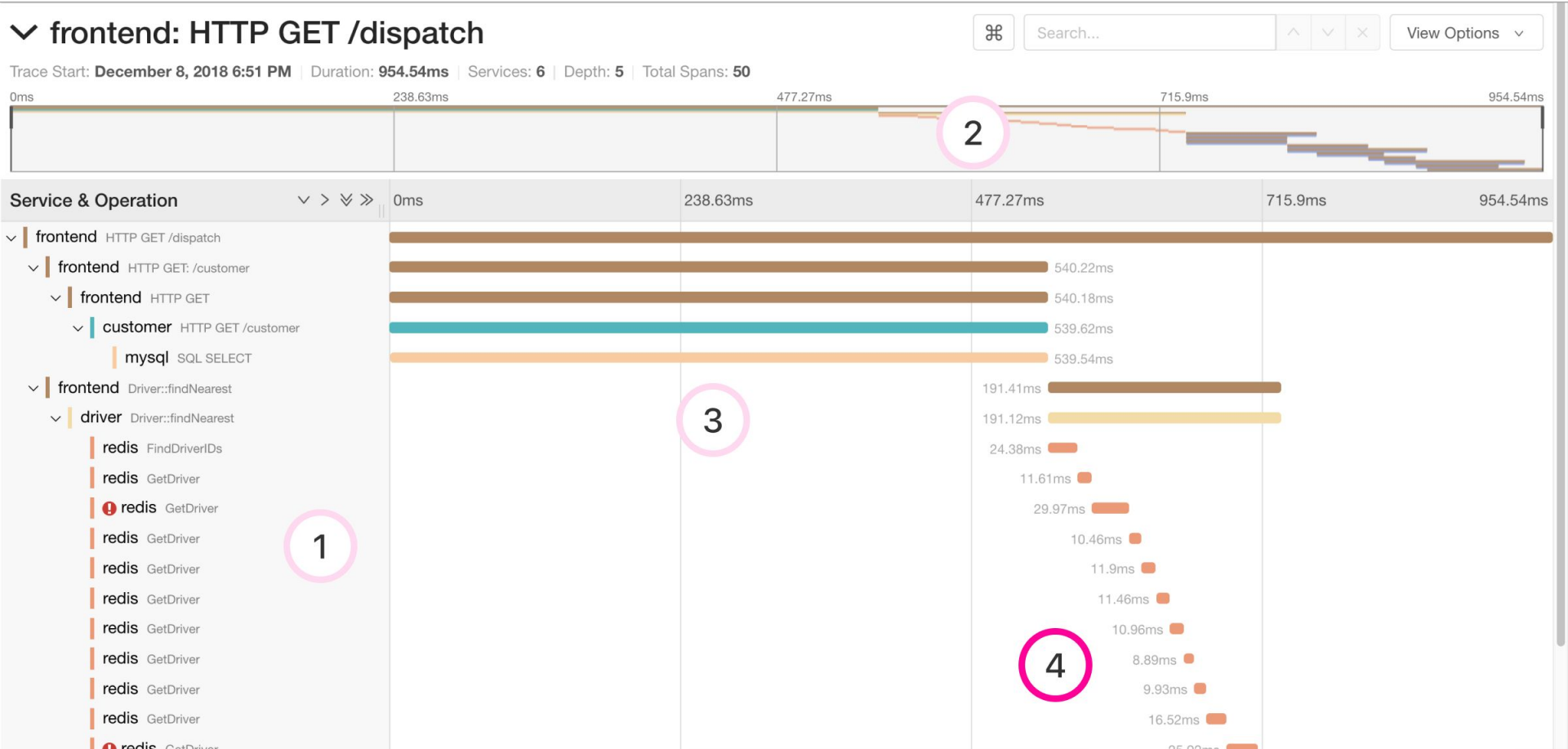
Trace timeline – Time + Mini-map



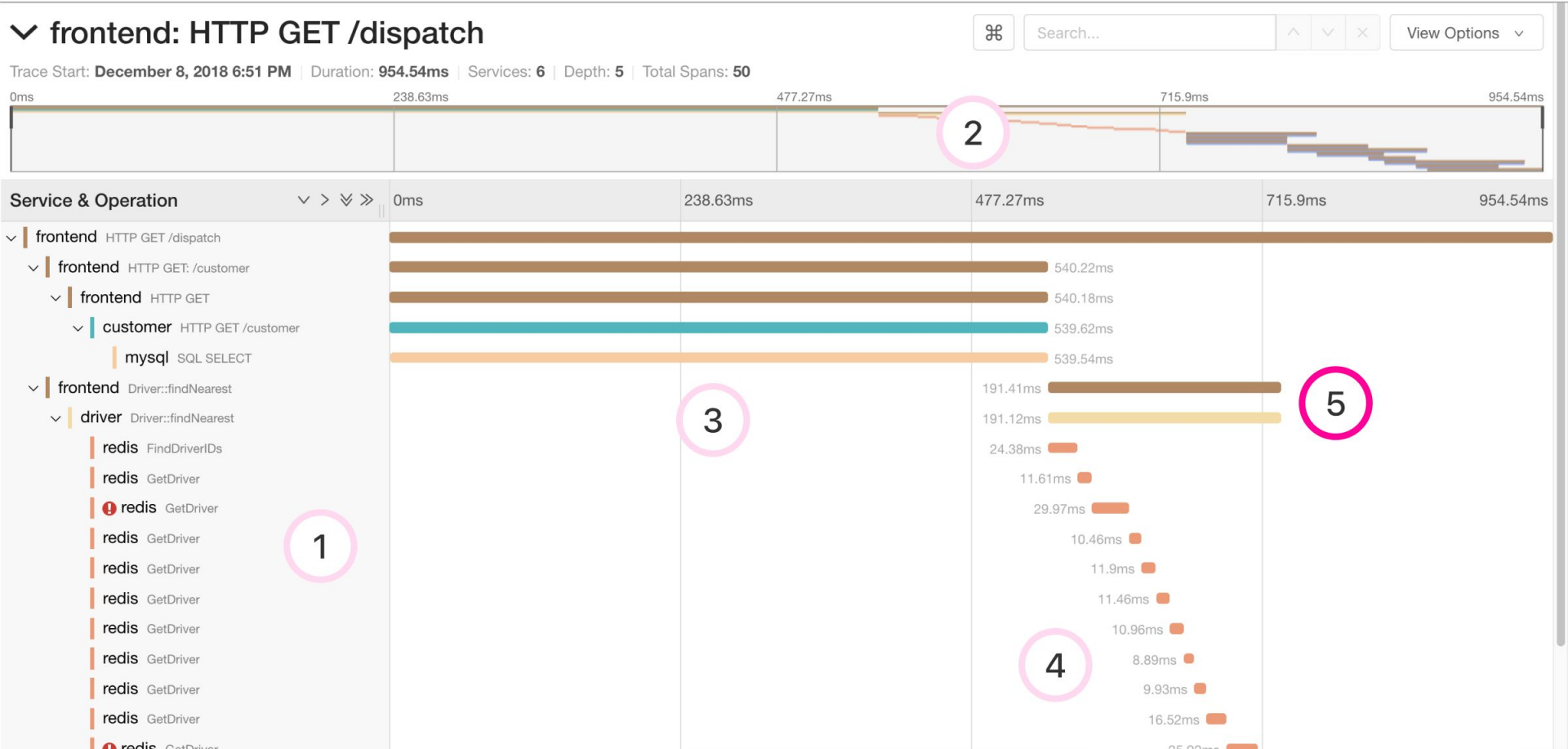
Trace timeline – A blocking operation



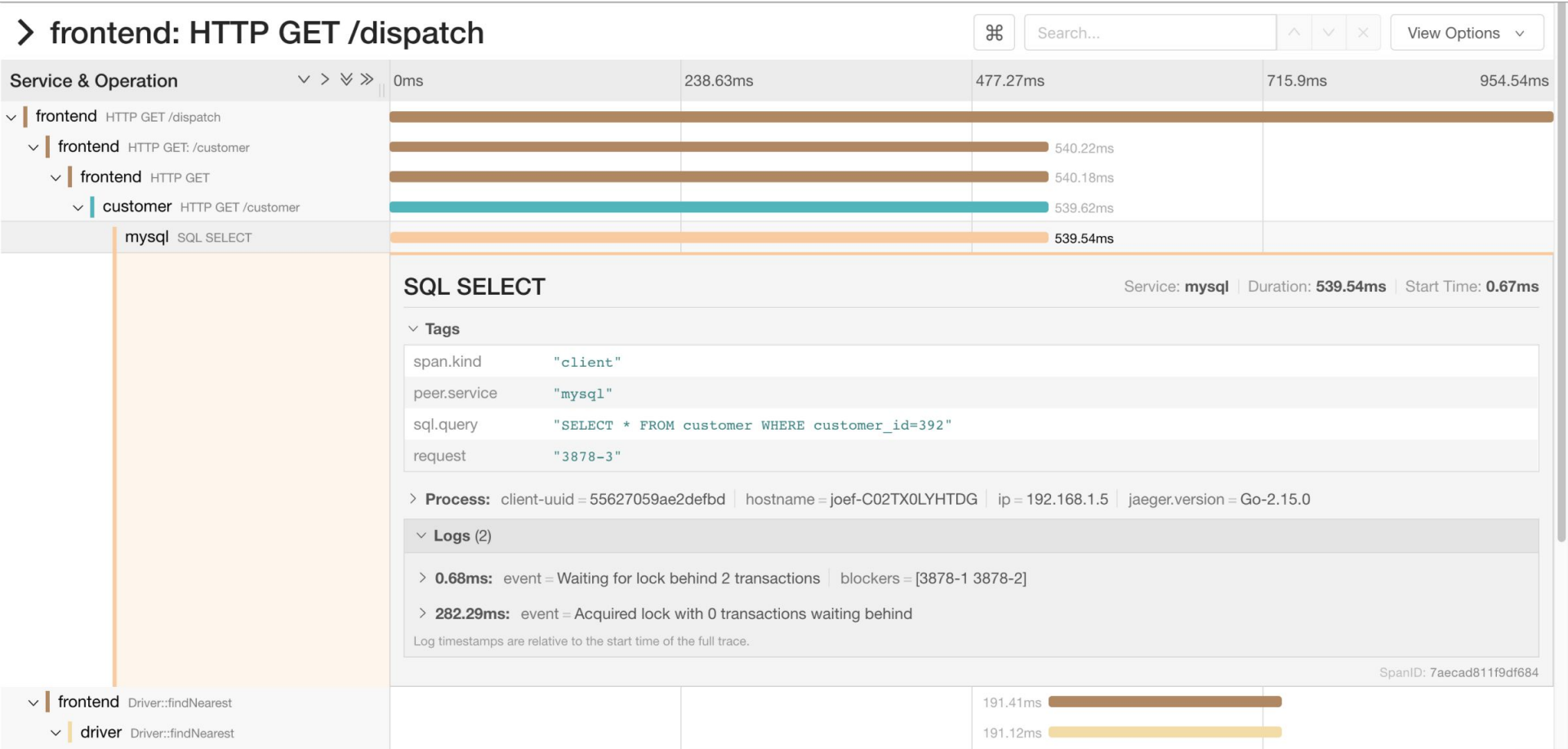
Trace timeline – Sequential operations



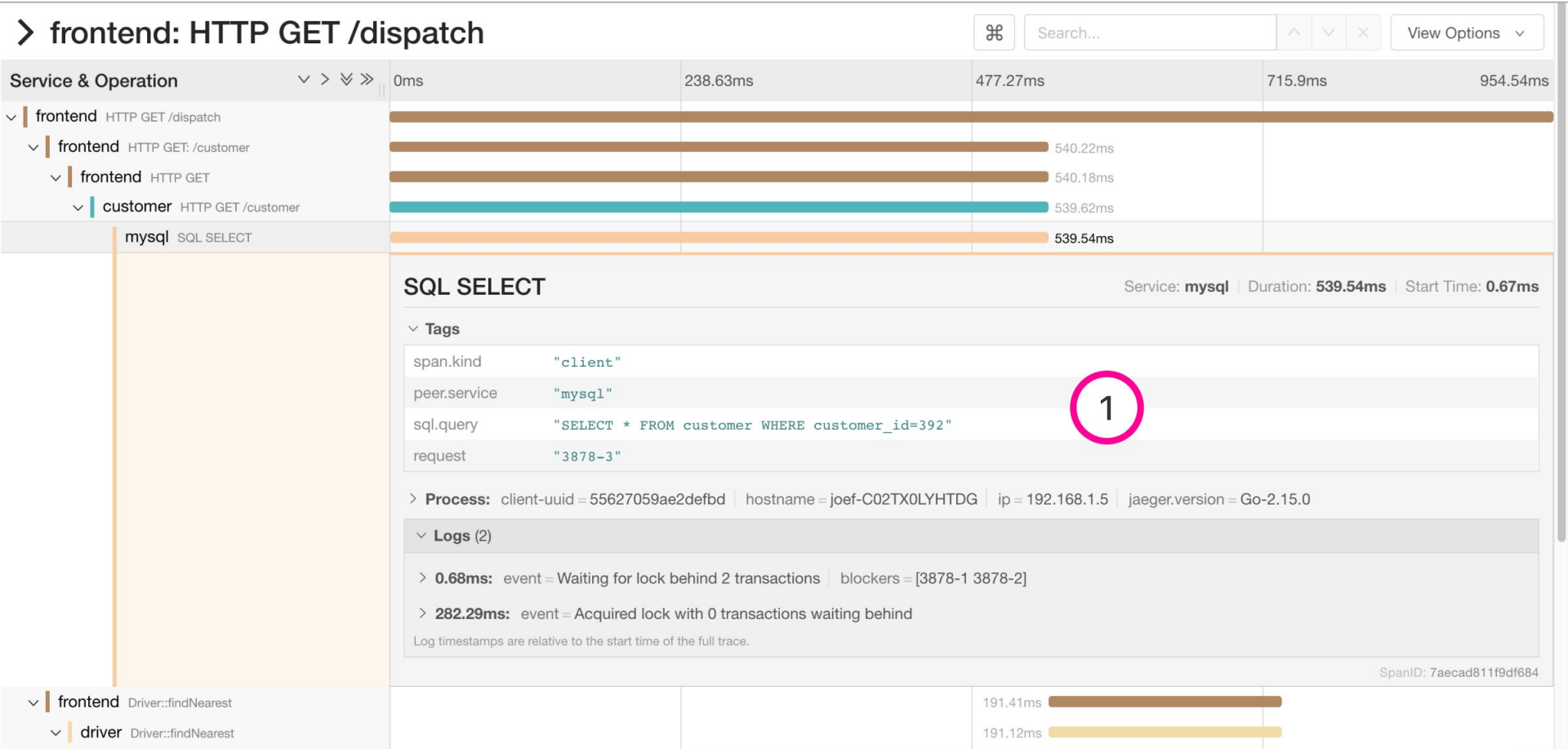
Trace timeline – Parents encompass descendents (generally)



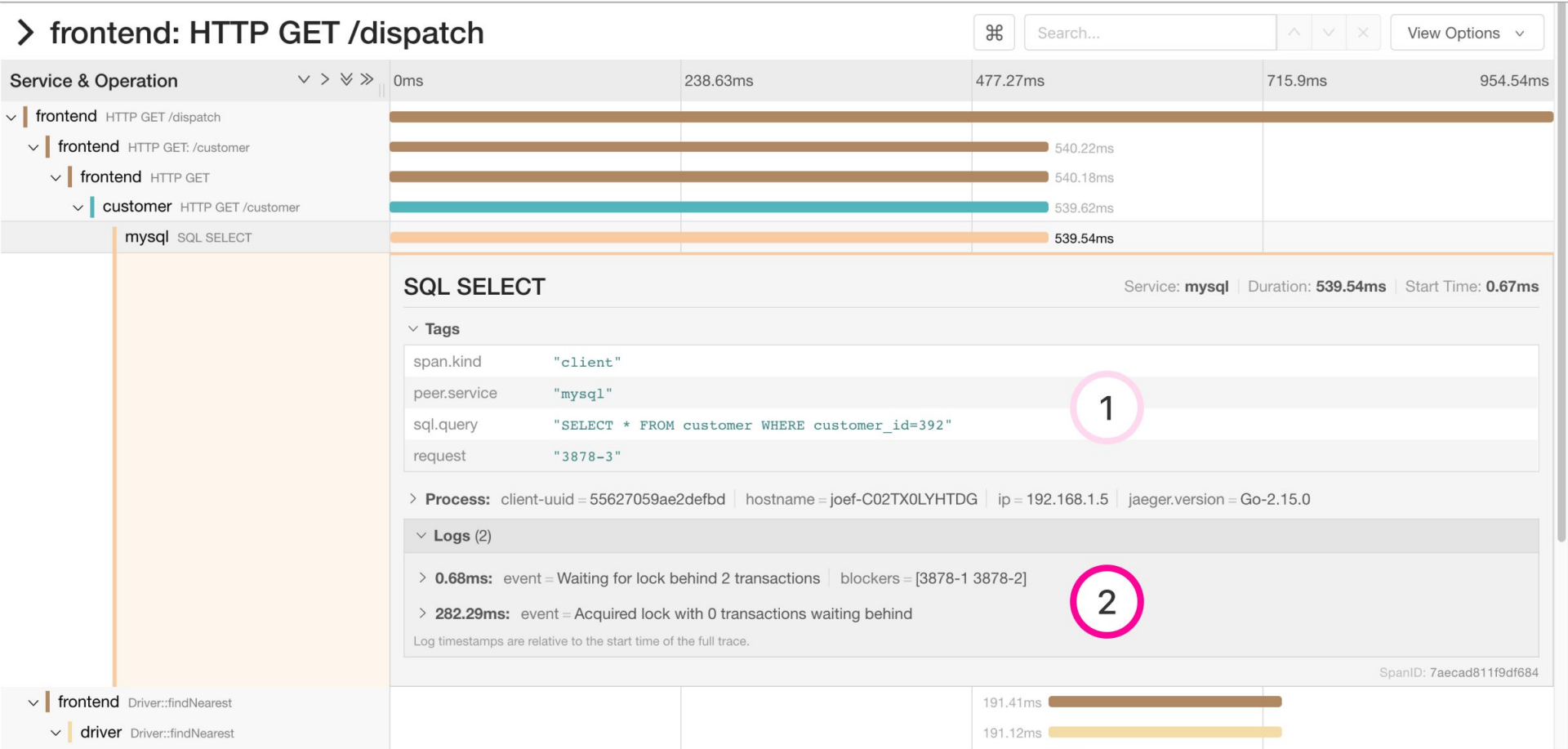
Span details



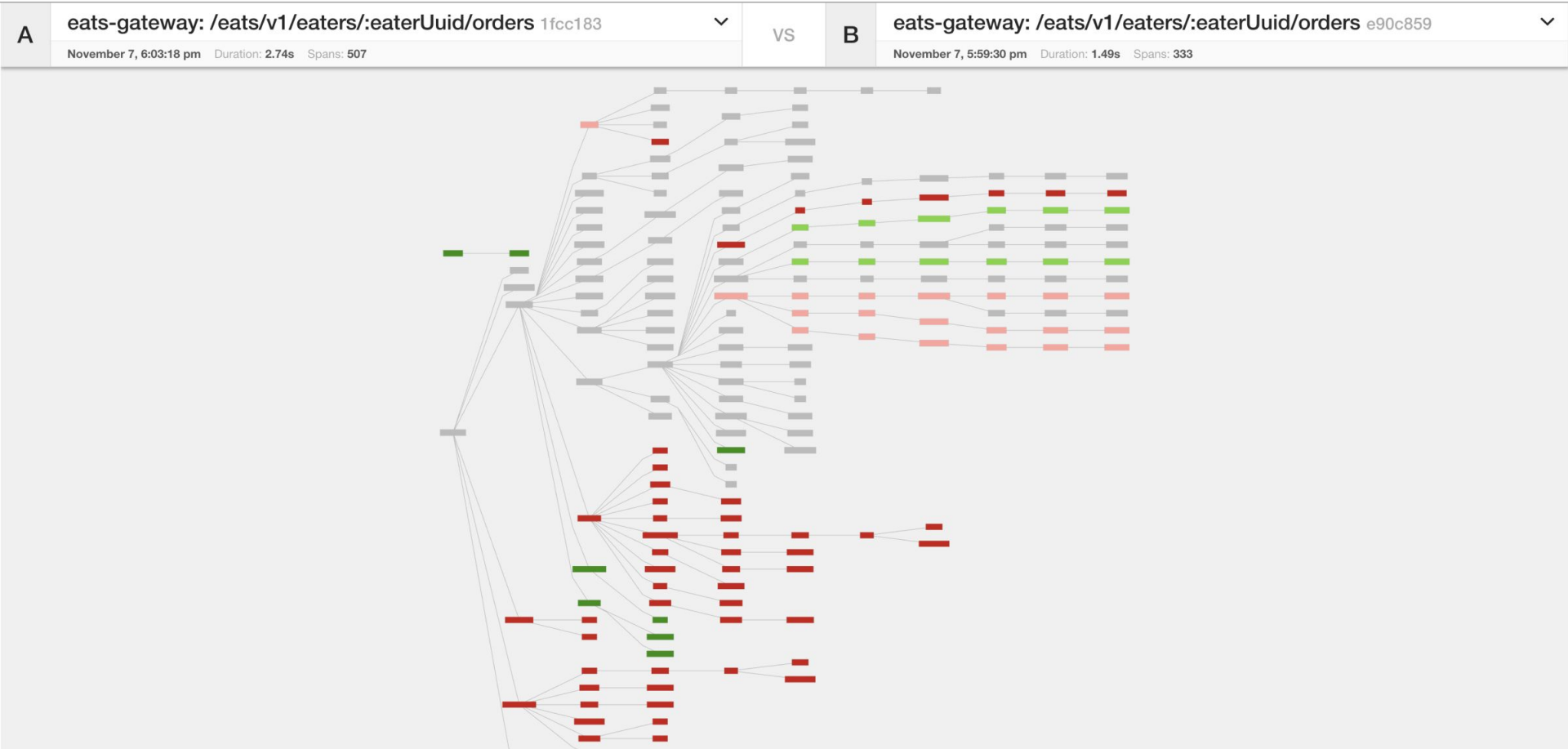
Span details – Database query



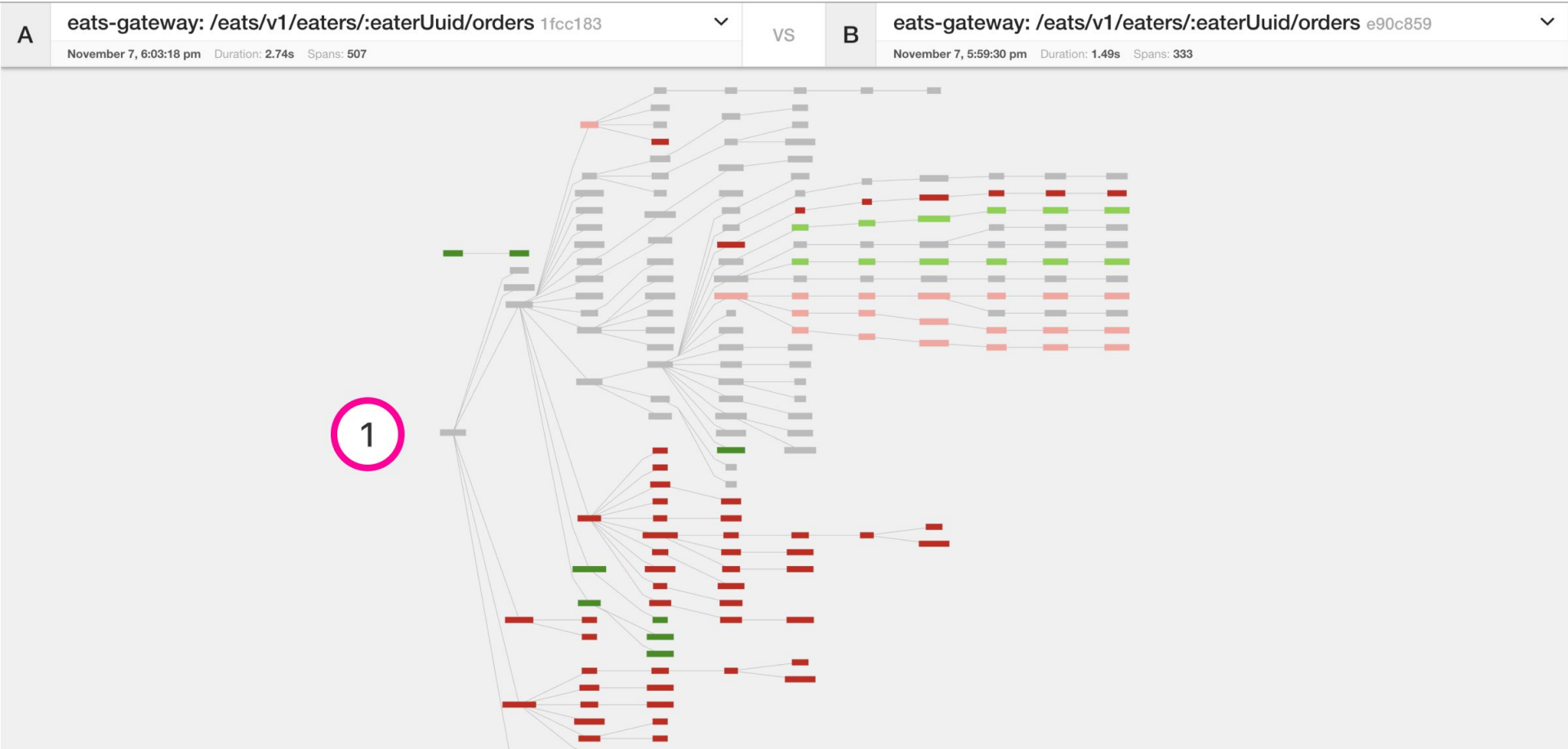
Span details – Lock contention



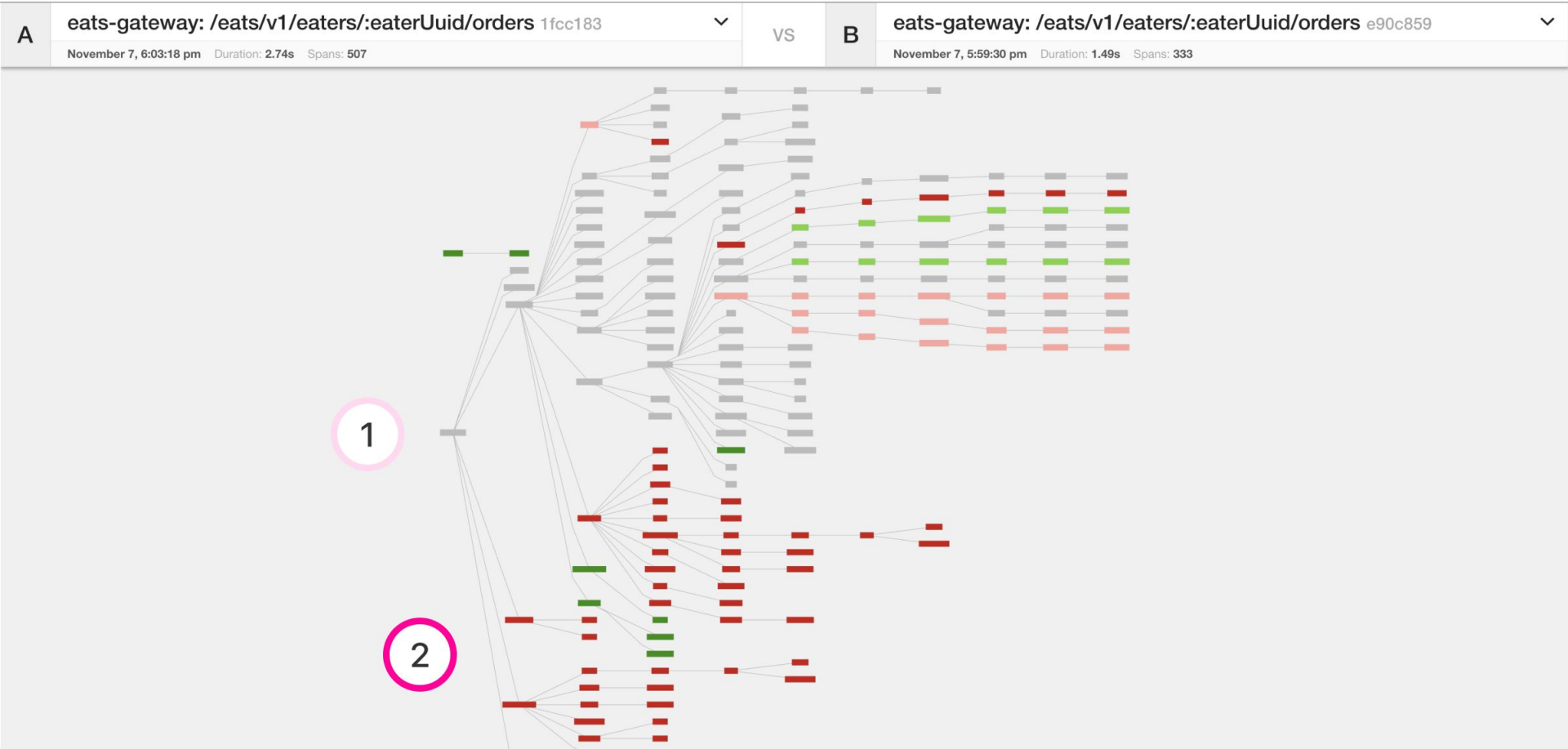
Comparing trace structures – Unified diff



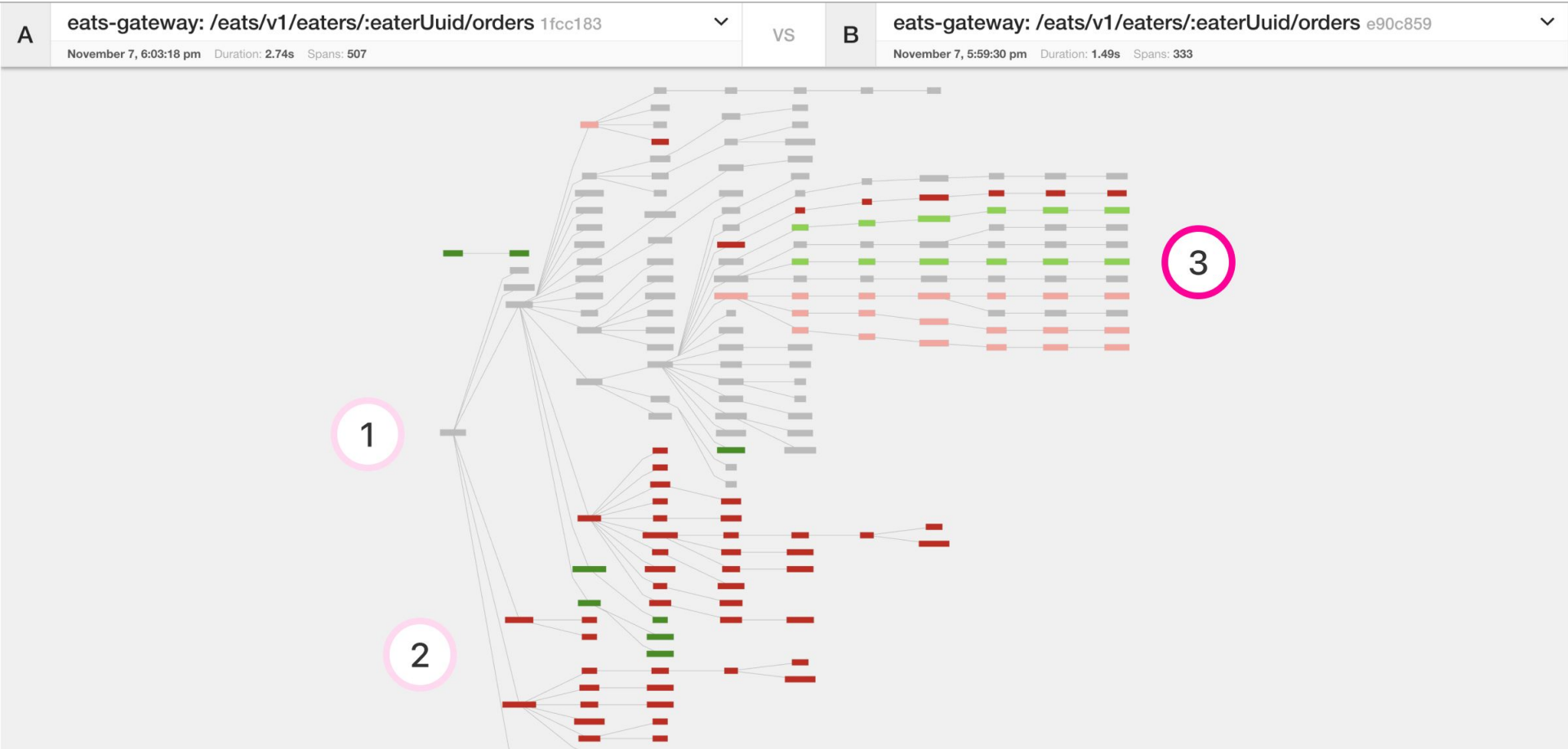
Comparing trace structures – Shared structure



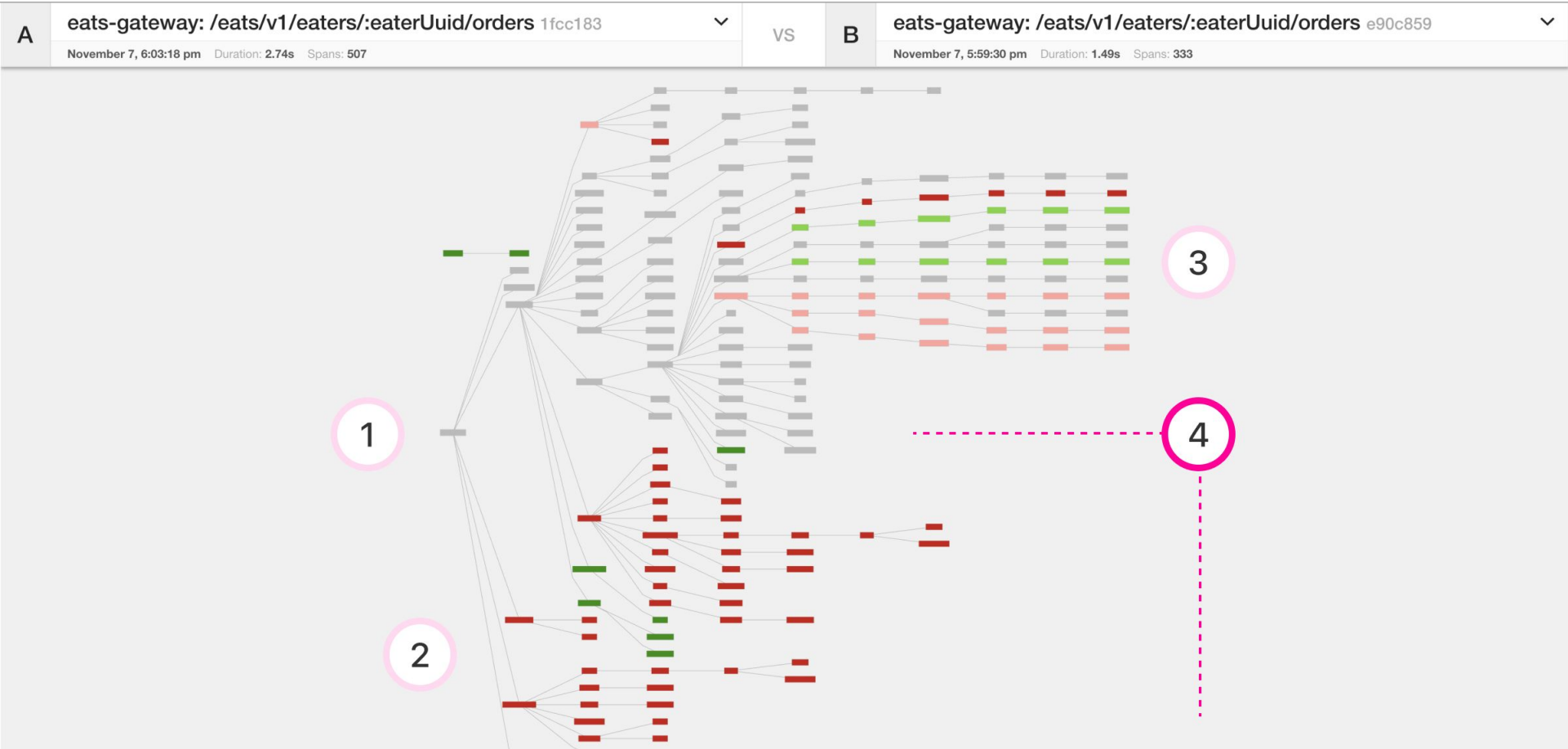
Comparing trace structures – Absent in one or the traces



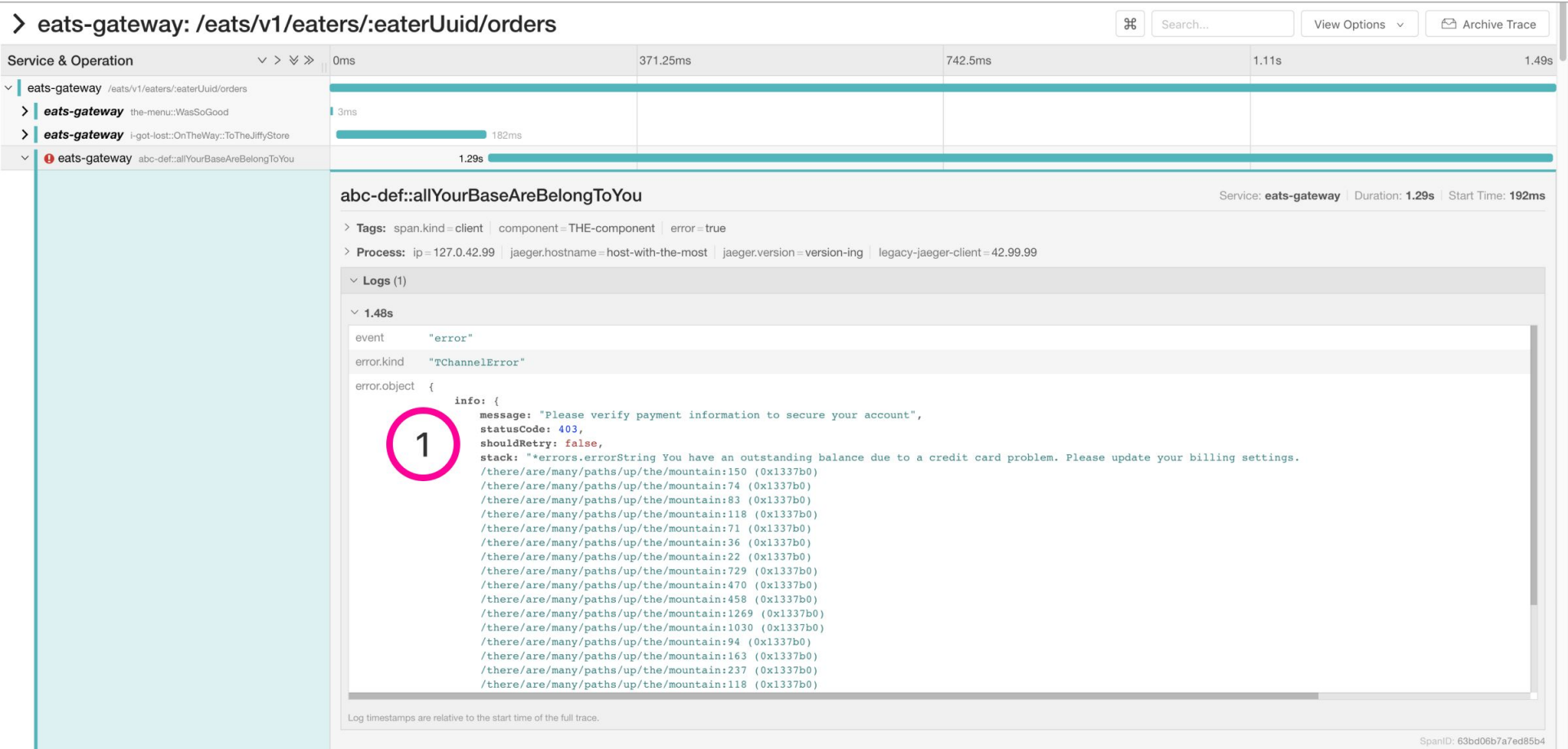
Comparing trace structures – More or less within a node



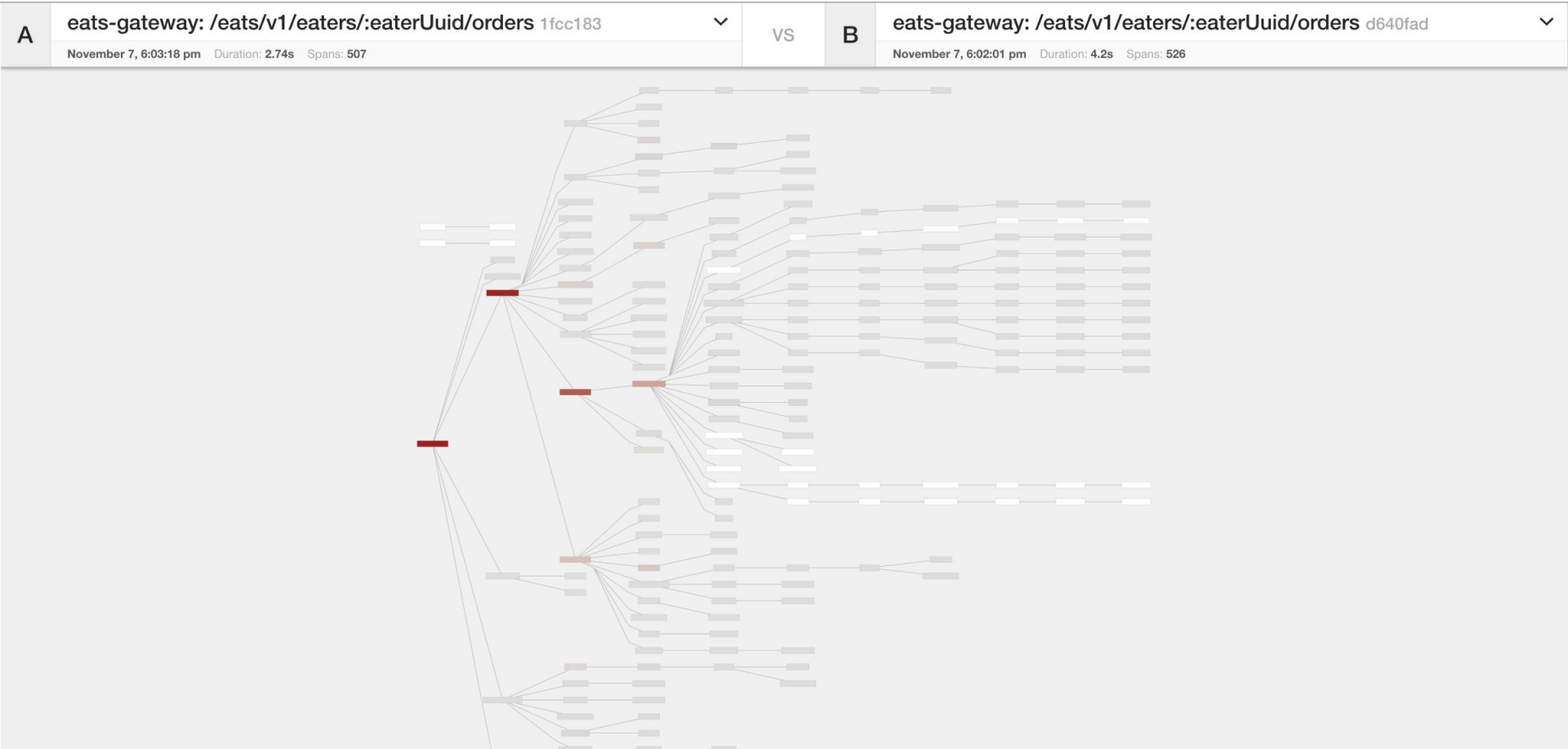
Comparing trace structures – Substantial divergence



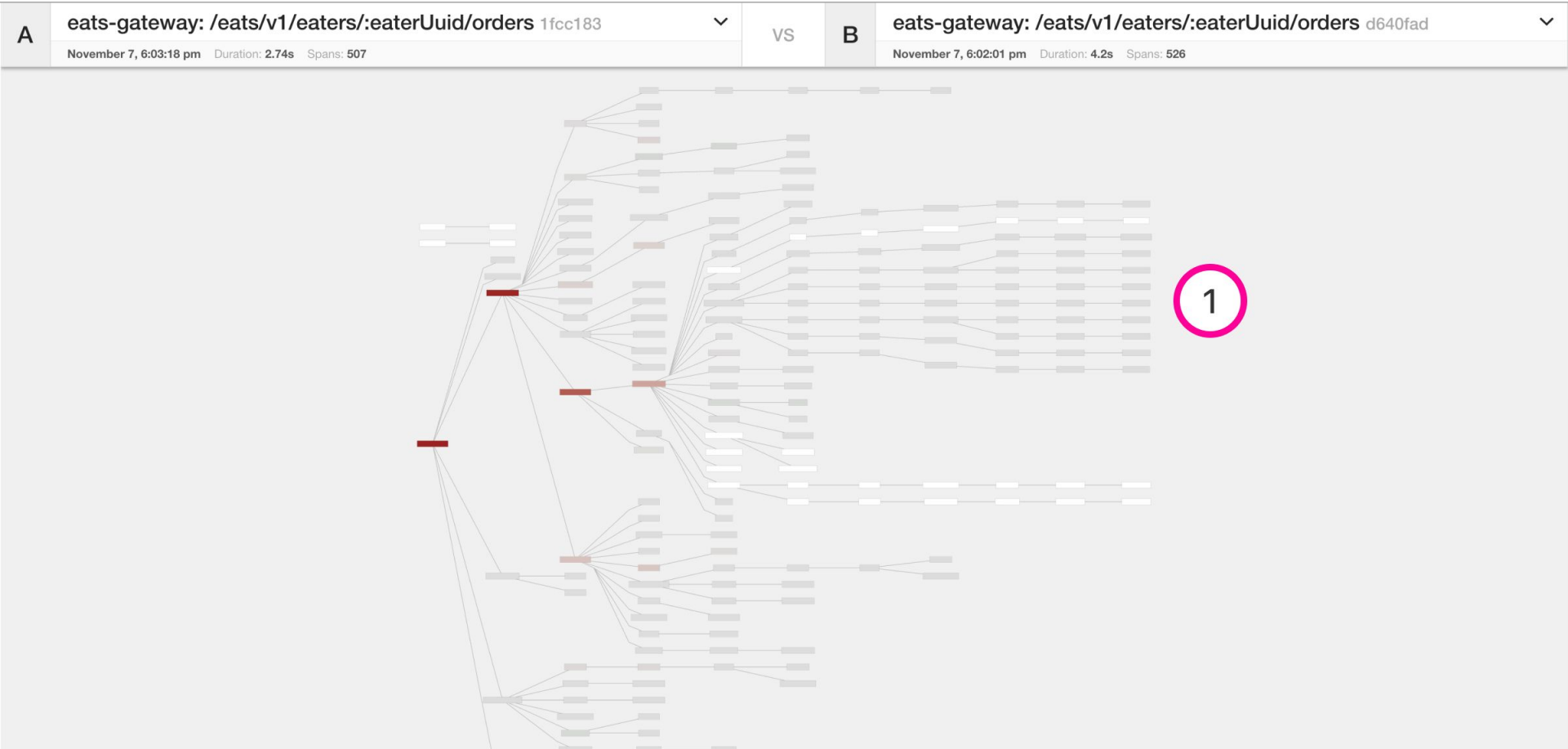
"You have an outstanding balance…"



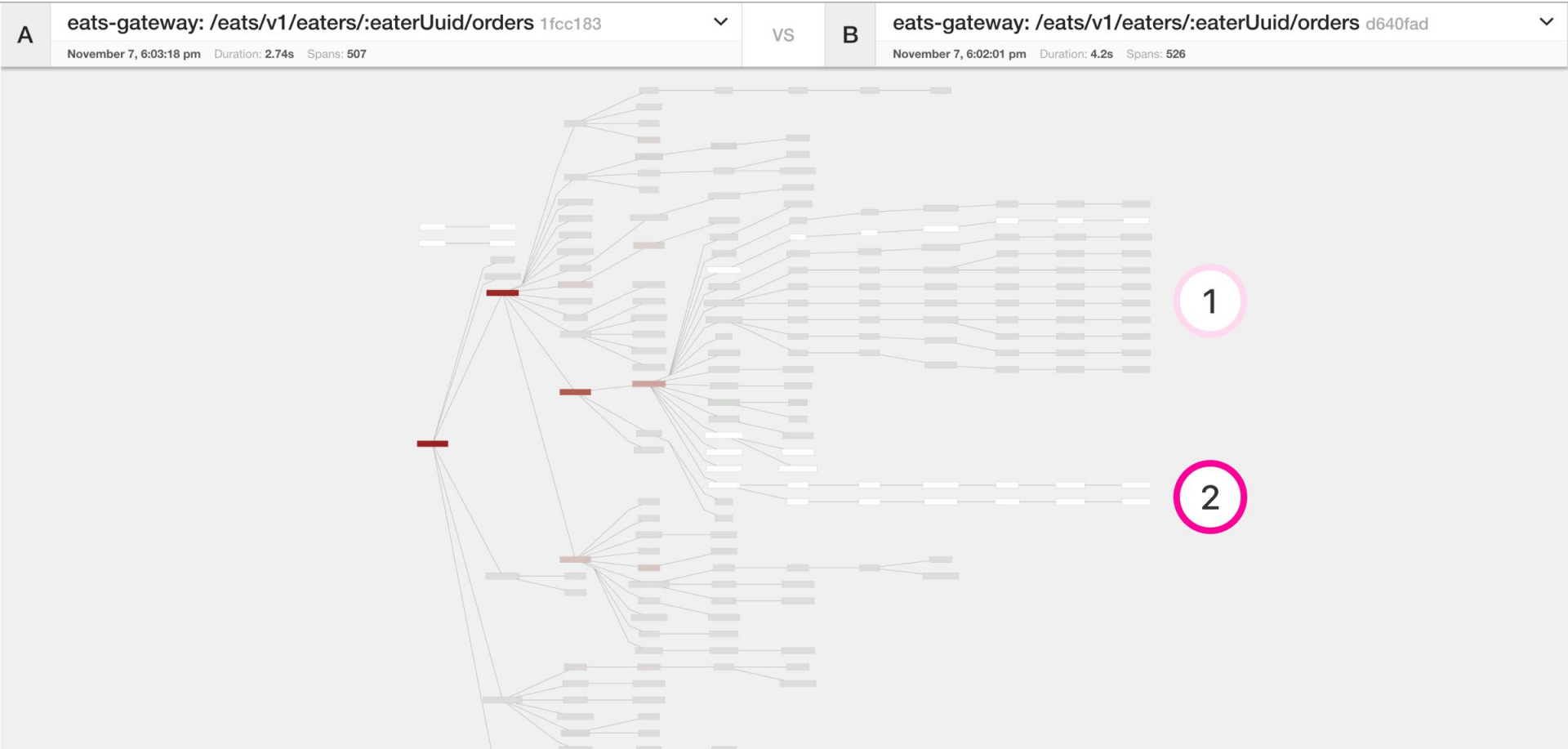
Comparing span durations



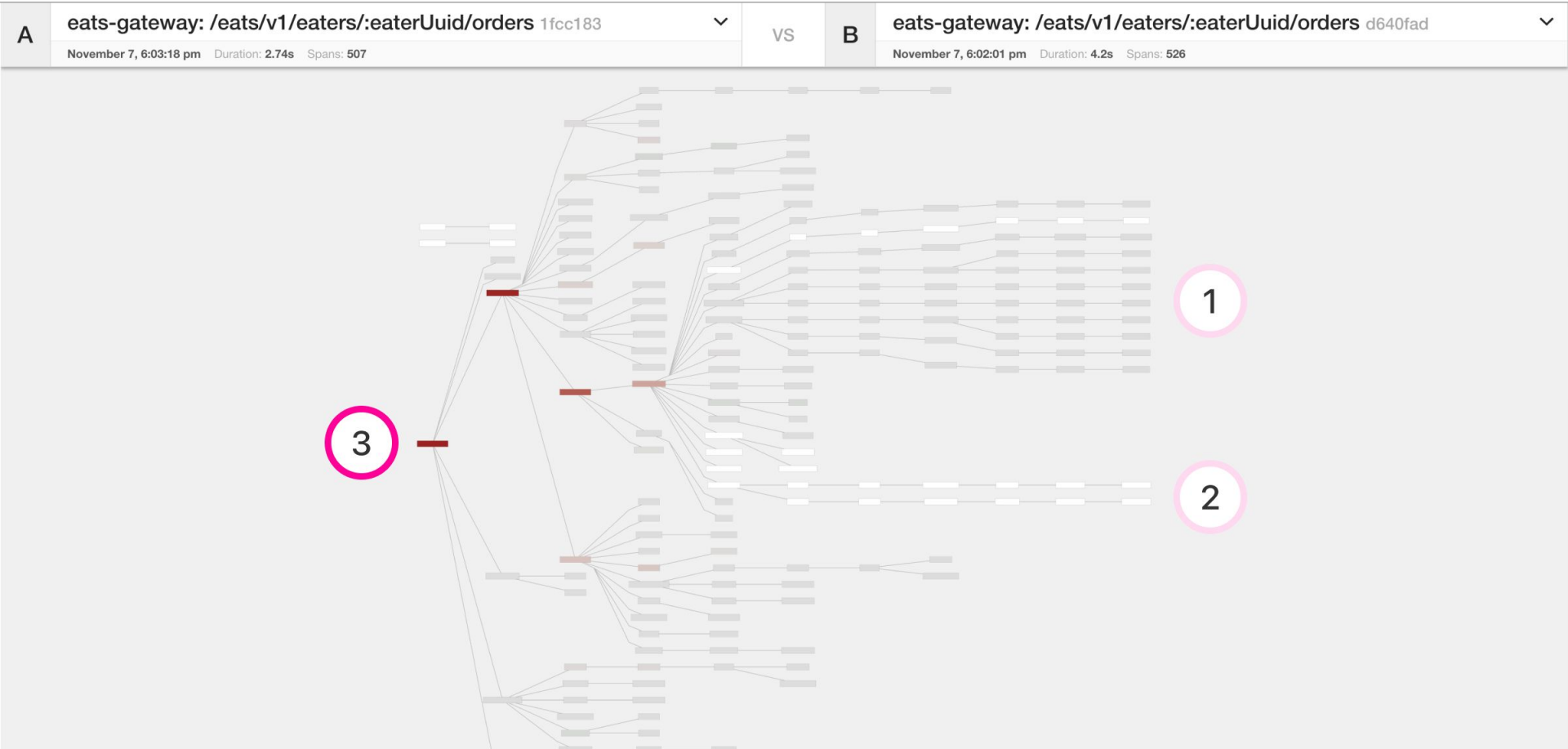
Comparing span durations – Similar durations



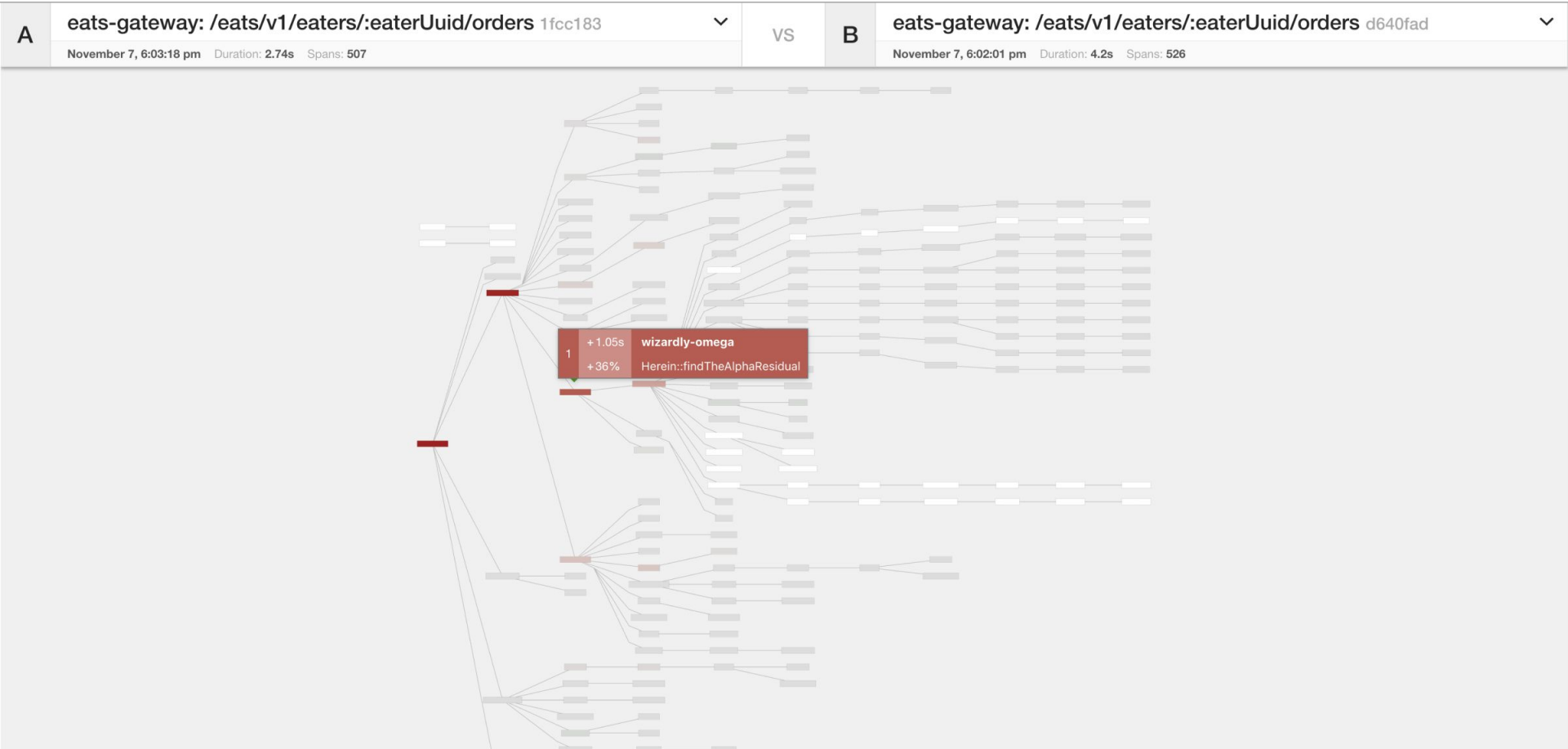
Comparing span durations – Nodes that aren't shared



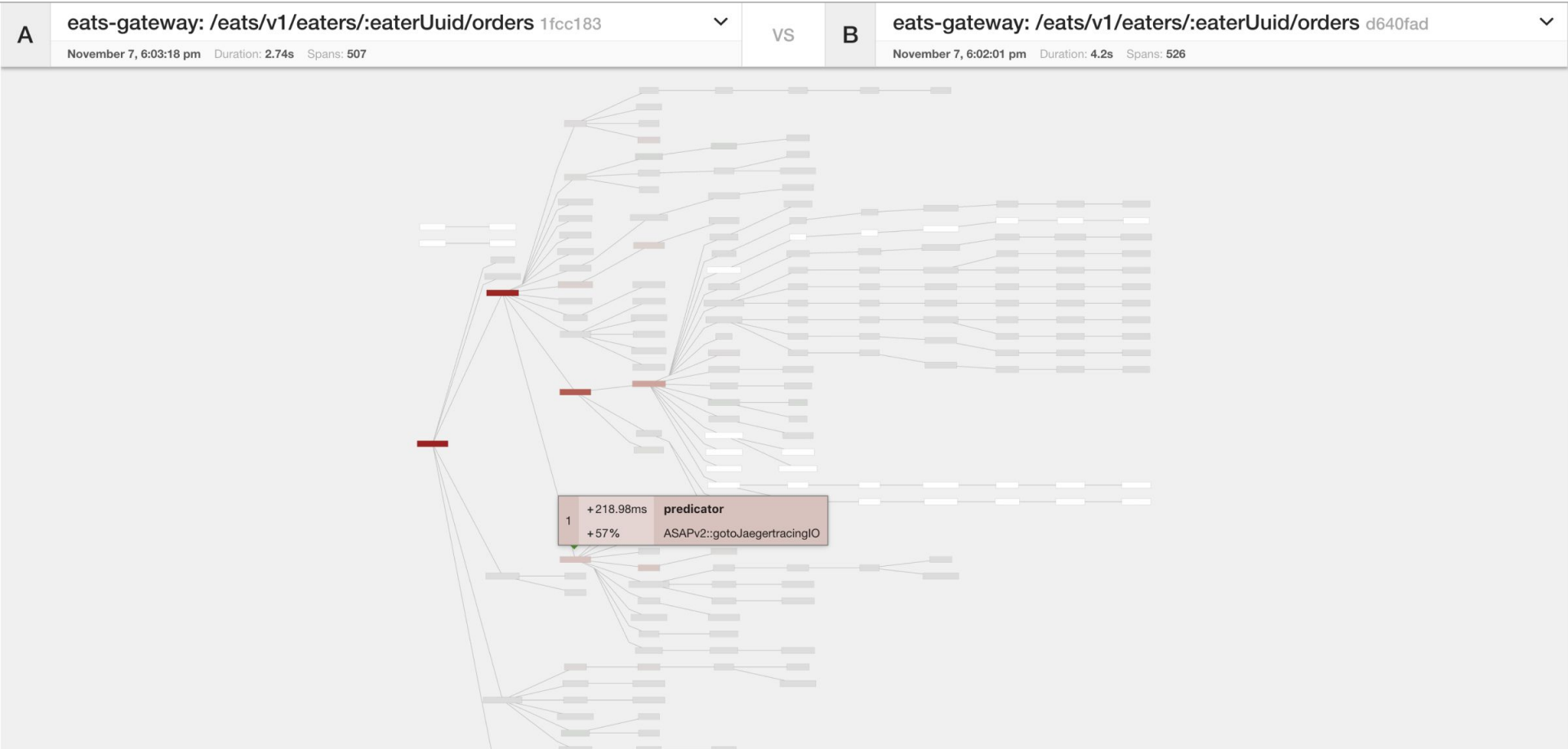
Comparing span durations – Follow the slower nodes



Comparing span durations



Comparing span durations



Graph Visualizations

Gantt chart is not great for traces with many 100s of spans

- Trace Diffs
  - Compare two traces
  - Compare one trace against a group of traces (coming soon)
- Trace Graph
  - Call graph visualization with mini-aggregations
  - Showing paths rather than individual RPCs

Graph Visualizations

- Surface less information
- Condense the structural representation
- Emphasize the differences
- Distinct comparison modes simplify the comparisons

Distributed Tracing Systems

distributed transaction monitoring

performance and latency optimization

root cause analysis

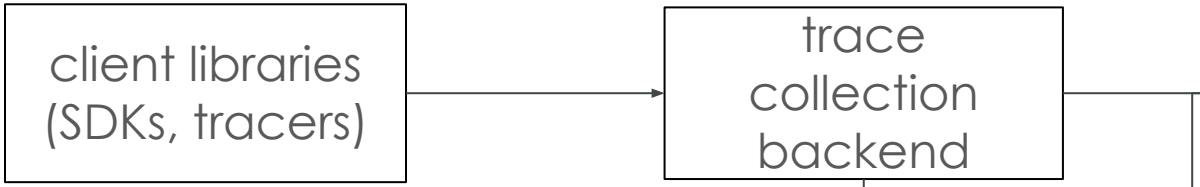
service dependency analysis

distributed context propagation

Jaeger

Architecture

Jaeger, a Distributed Tracing Platform



Instrumentation not included

Jaeger project does not provide instrumentation!

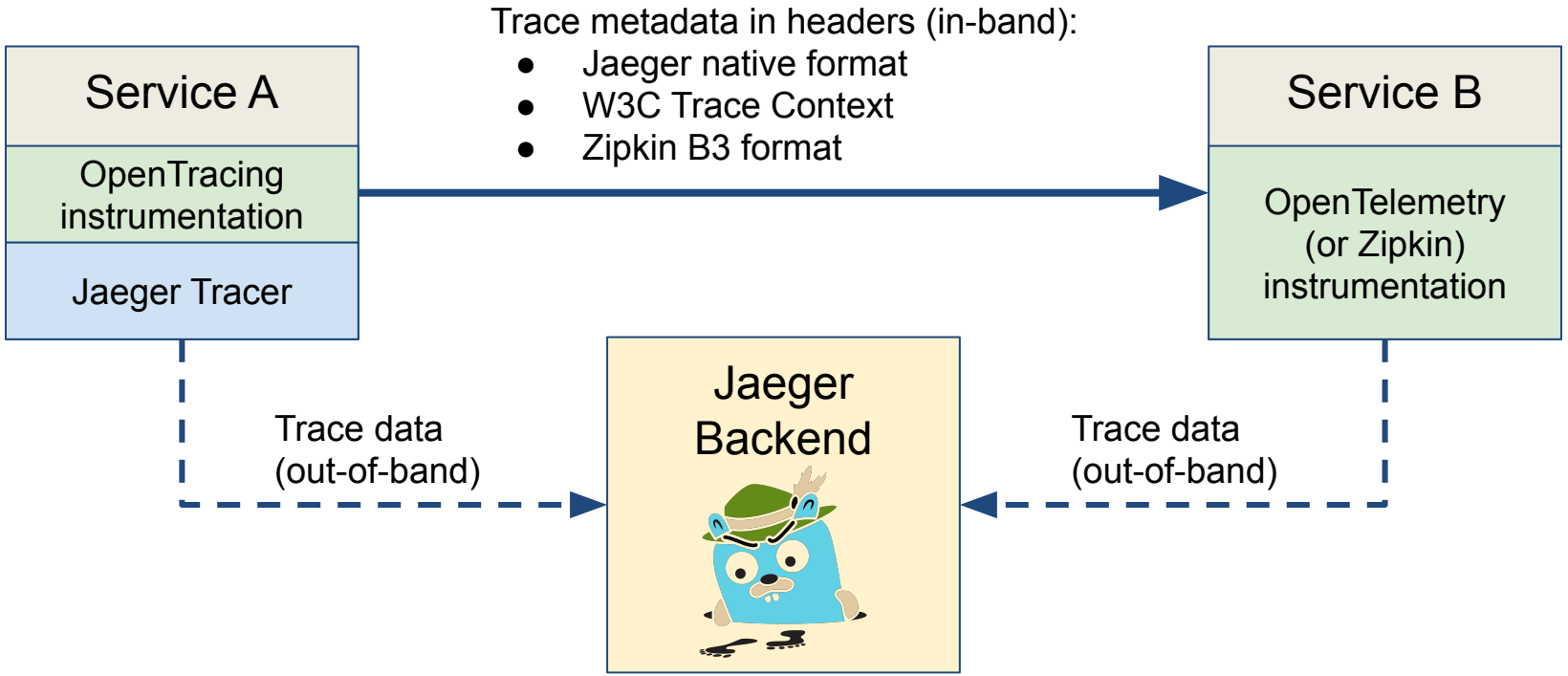
Use OpenTracing or OpenTelemetry.

Jaeger - /ˈyāgər/, *noun*: hunter

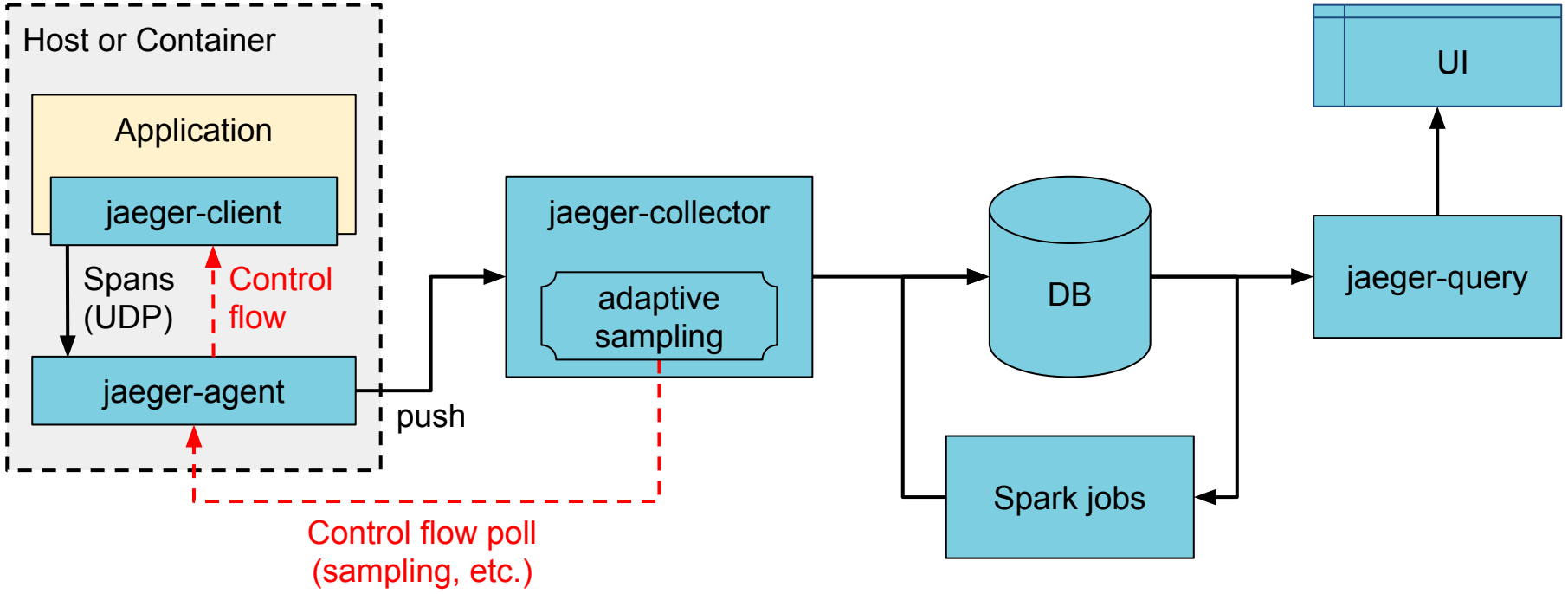
- Inspired by Google's Dapper and OpenZipkin
- Created at Uber in August 2015 (blog)
- Open sourced in April 2017
- Joined CNCF in Sep 2017 (as incubating)
- Graduated to top-level CNCF project

  Oct 2019 (CNCF announcement)

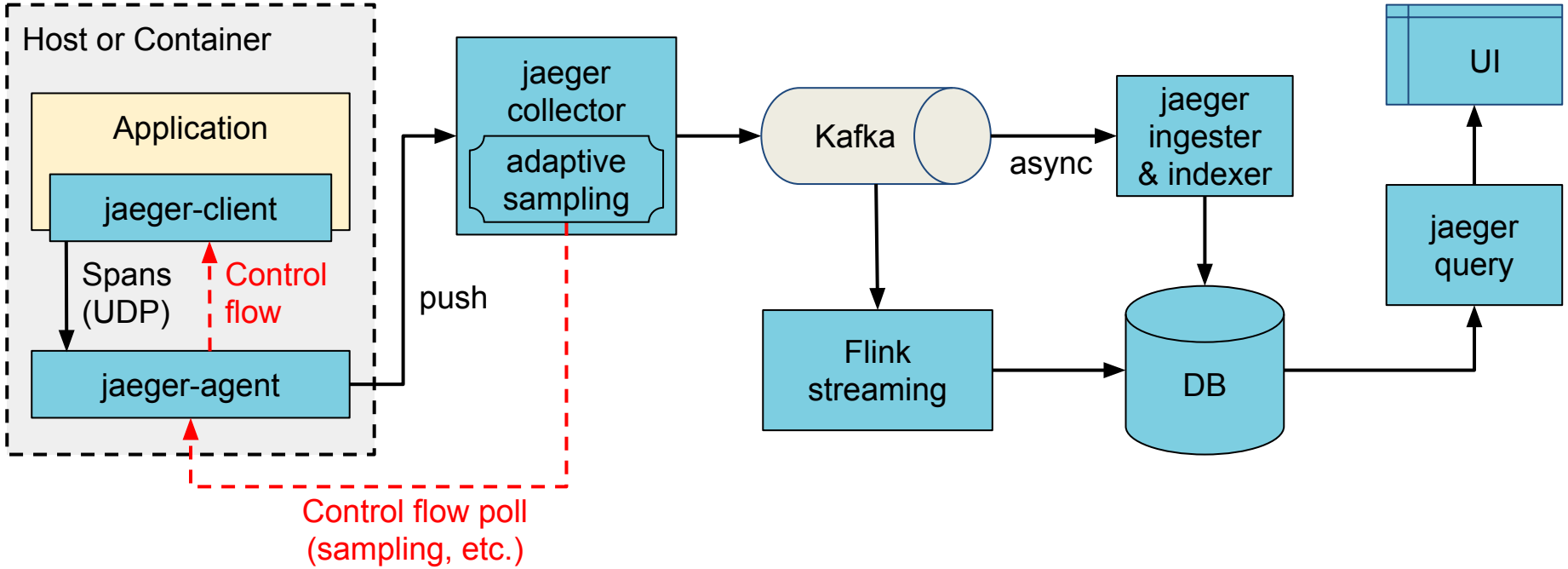
Jaeger and your system



Architecture 2017: Push



Architecture now: Push+Async+Streaming



Technology Stack

- Go backend
- Pluggable storage
  - Cassandra, Elasticsearch, badger, memory
- React/Javascript frontend
- OpenTracing Instrumentation libraries
- Integration with Kafka, Apache Flink

OPENTRACING

Go

Zipkin Compatibility

- Clients
  - Zipkin `B3-***` headers for context propagation
  - Interop between Jaeger-instrumented and Zipkin-instrumented apps
- Collector
  - Zipkin Thrift, Protobuf, and JSON v2 span format
  - Use Zipkin instrumentation (e.g. Brave) to send traces to Jaeger
- Kafka

Jaeger

And Sampling

Why do we sample

- Tracing data is quite rich (2 spans per RPC)
- Saving everything incurs large storage costs
- Performance overhead from instrumentation

Head-based (upfront) sampling

- Sampling decision is made at the start of the trace propagated in the trace context
- Minimal perf overhead when trace is not sampled
- Easy to implement, supported by Jaeger SDKs
- Can easily miss rare anomalies/outliers

Head-based sampling in Jaeger

- SDKs can be configured with different samplers (probabilistic, rate limiting, etc.)
- SDKs default to "remote" sampler that polls the actual sampling strategy from the backend, which allows centralized configuration
- Configuration is per service & endpoint

Tail-based (post-trace) sampling

- Sampling decision is made at the end of the trace
- Can be much more intelligent, based on observed latency, errors, etc. Can catch anomalies.
- Requires temporary storage of all traces
- Incurs perf overhead even for traces that may be later discarded

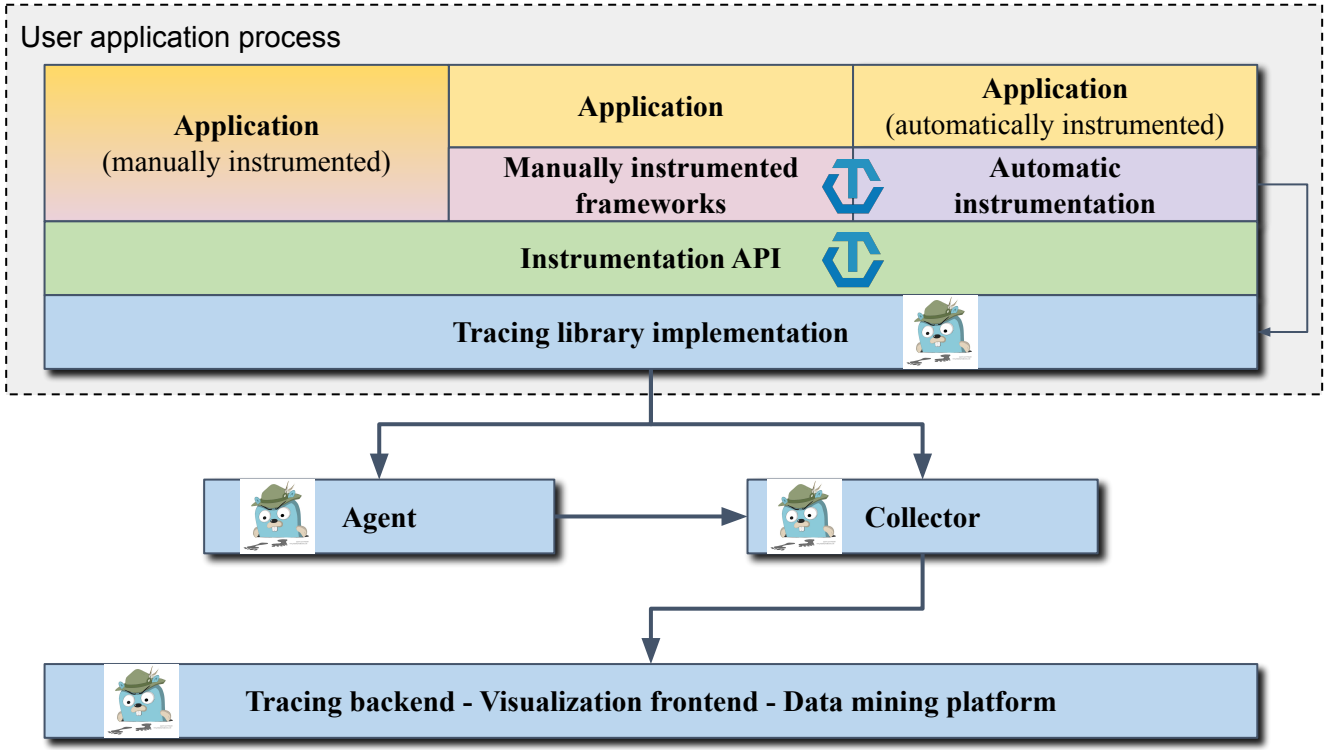
Tail-based sampling in Jaeger

- Supported in `jaeger-opentelemetry-collector`
- Configurable sampling rules: latency, certain tags
- Single-node mode only, multi-node sharded solution will be available in the future

Jaeger

And OpenTelemetry

Jaeger with OpenTracing



OPENTRACING

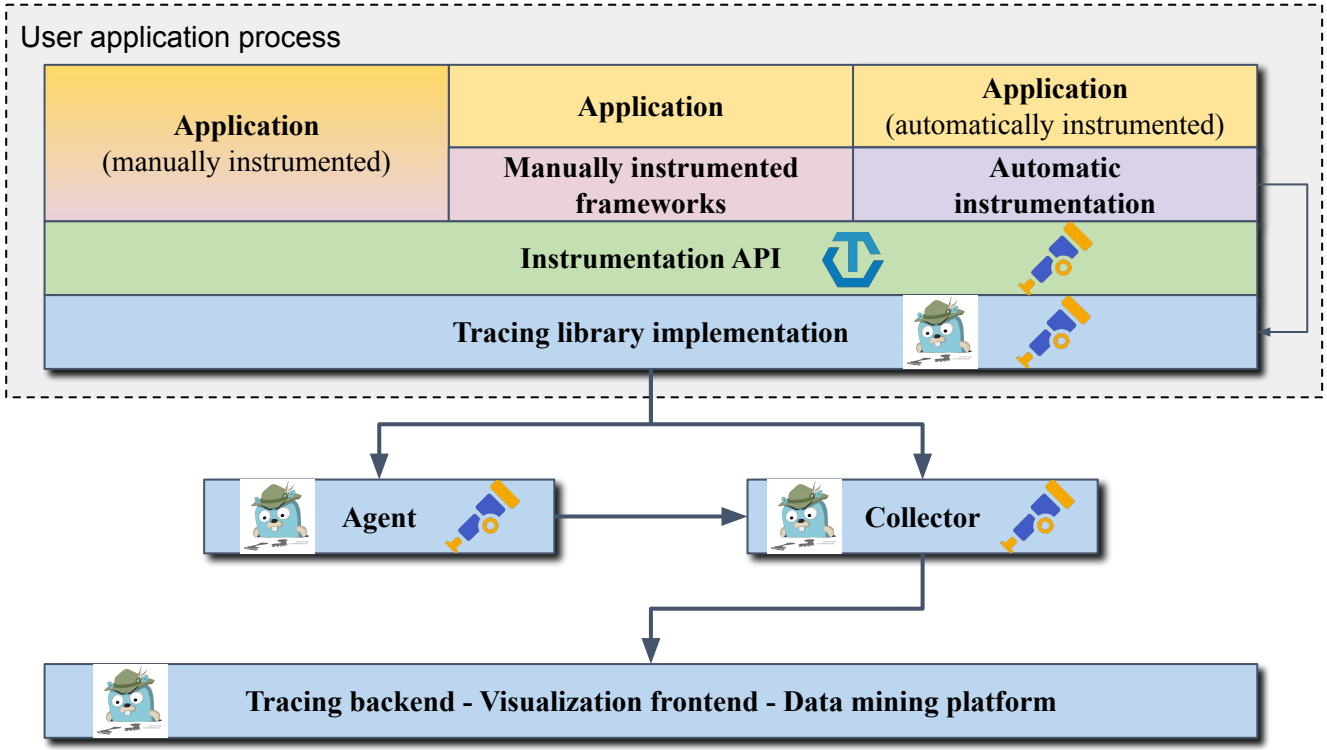
opentracing contrib

API

Jaeger components on OpenTelemetry

- OpenTelemetry Collector is written in Go
- We built Jaeger-specific versions
  - Have the same capabilities as upstream OTel
  - With Jaeger extensions, e.g. storage
- We're converting Jaeger storage implementation to OTel data model for better compatibility

Jaeger with OpenTelemetry



OpenTelemetry

contrib

API

SDK

Agent &
Collector

New Features

- Kubernetes Operator
- Badger storage
- Storage plugins: Couchbase, InfluxDB
- Visual trace comparisons
- Security improvements
  - TLS with gRPC, Kafka, Elasticsearch

Documentation Website

- Releases & Downloads
- Architecture
- Deployment
- Command line options
- Client features

To learn more

- Jaeger Deep Dive - Wed, Aug 19, 13:45 - 14:20

https://jaegertracing.io

Getting in Touch

- GitHub: https://github.com/jaegertracing
- Chat: https://gitter.im/jaegertracing/
- Mailing List - jaeger-tracing@googlegroups.com
- Blog: https://medium.com/jaegertracing
- Twitter: https://twitter.com/JaegerTracing
- Bi-Weekly Community Meetings

Learn More

Website: jaegertracing.io/

Blog: medium.com/jaegertracing