Observability is for the Frontend, Too!

Gaining insights through the browser with Open Telemetry

Justin Castilla

@Justincastilla.bluesky.social

Senior Developer Advocate, Elastic

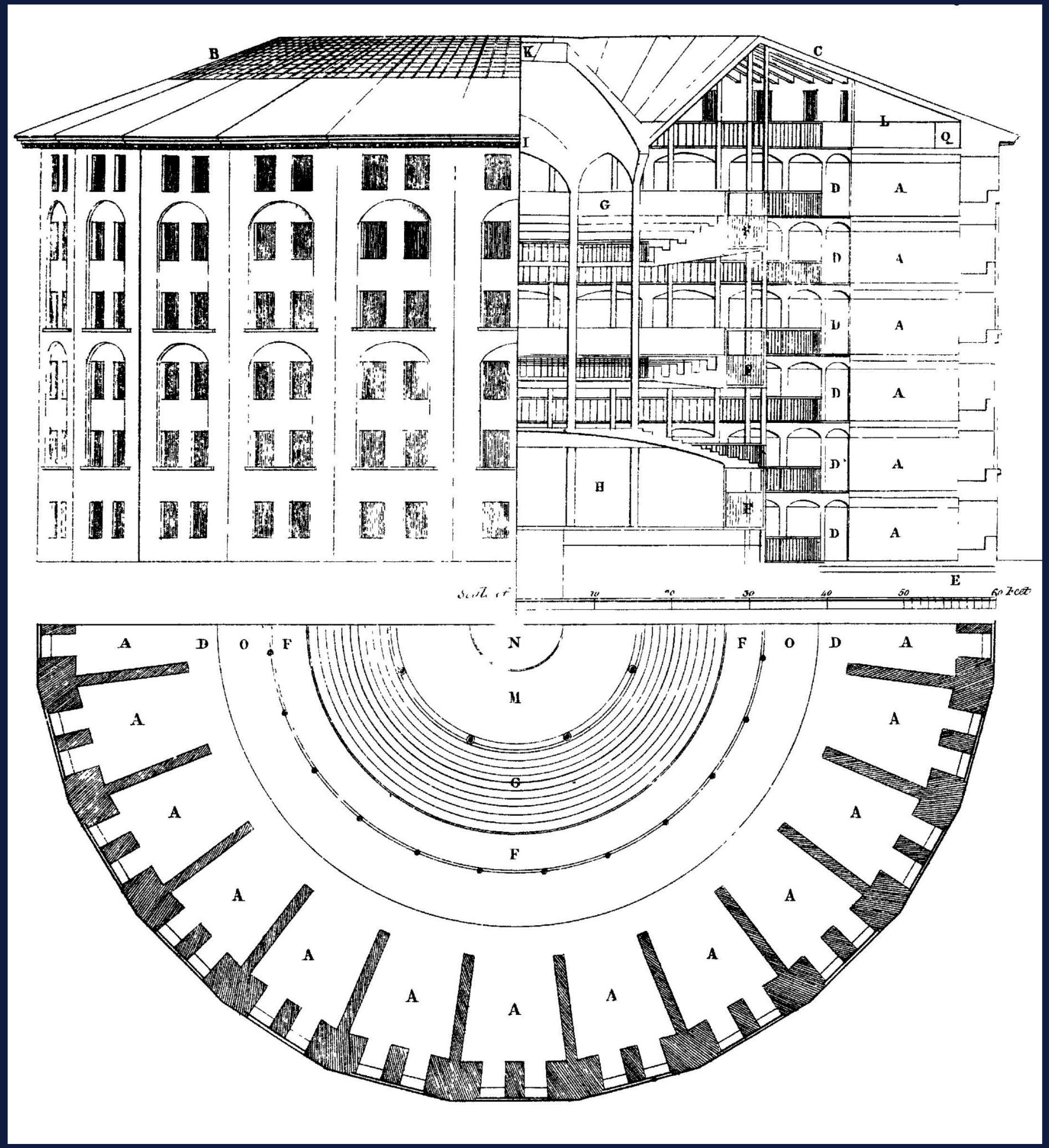
https://github.com/justincastilla/vanil
la-fullstack-otel

This talk is aimed for developers who may have some experience with observability but haven’t yet worked with browser implementations.

https://github.com/justincastilla/vanilla-browser-otel

Talk Agenda

- What is Observability?
- Why in the browser?
- How?
  - Manual
  - Automatic
  - Hybrid
- Demo Time!
- Reflection



What is Observability?

Collection, **aggregation**, and **dissemination** of **telemetry** (metrics, logs, traces, and profiling) within an application or service

Reveals **pathways** and **timelines** of processes as they **travel** through your codebase

Very big in backend application management

Why Observability in the Browser?

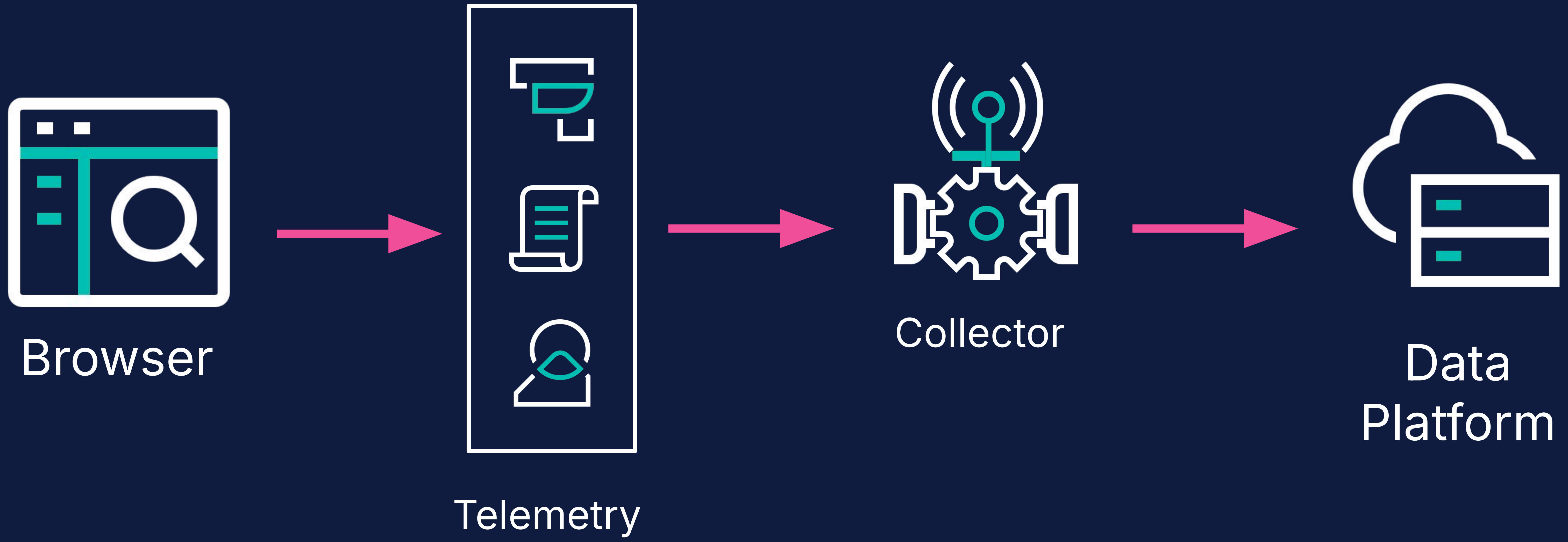
Frontend **latency**, unknown **delays**, unexpected **behavior**, and poor UX can't be solved with backend traces alone

Errors in the front may propagate errors in the back

RUM (Realtime User Monitoring) can boost your UX updates with your own metrics and data points.

Provide a **complete** picture of your data

Observability in the Browser



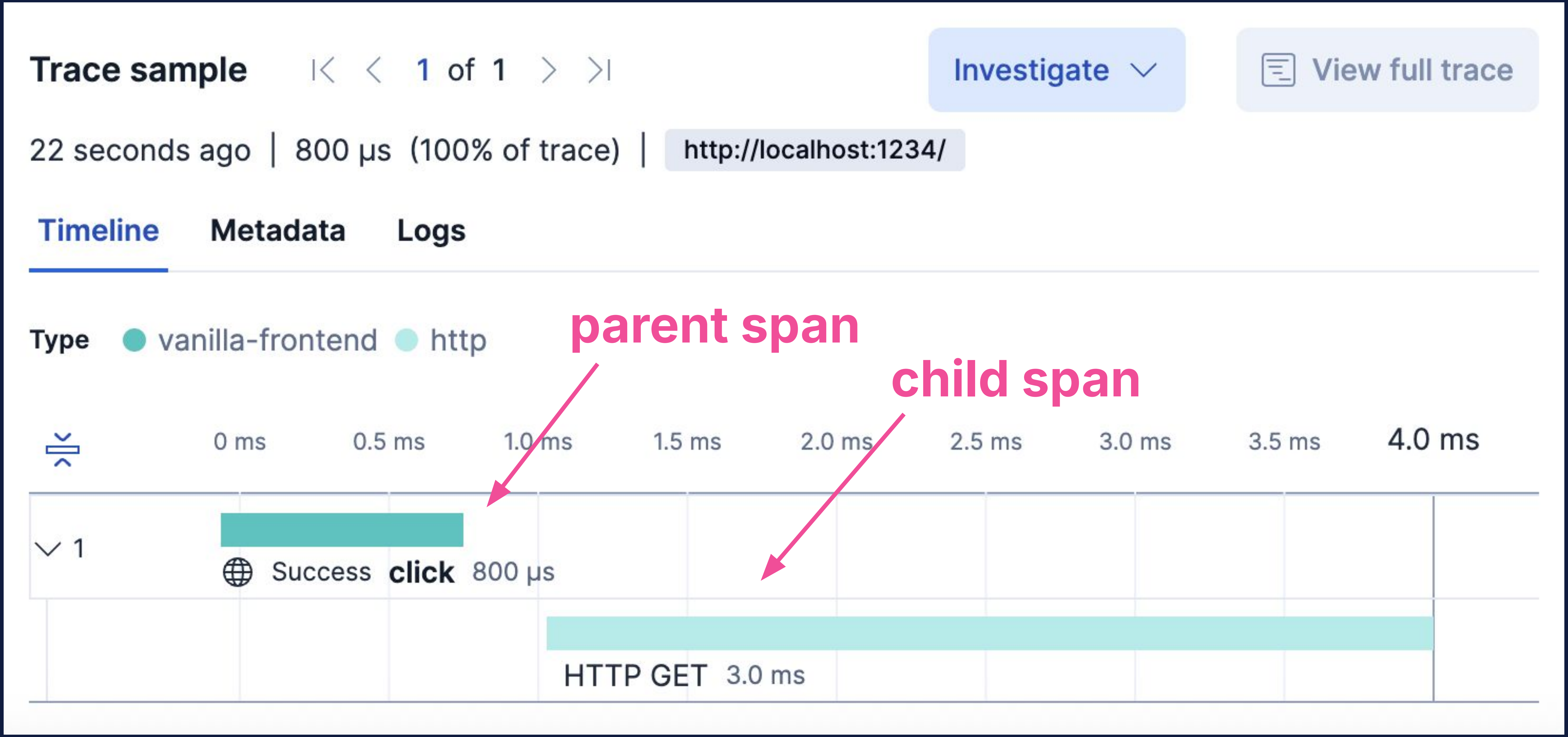
Observability with Traces

A **trace** is a record of the end-to-end path of a request through your application, showing how different components—like browser events, network calls, and backend services—worked together to fulfill it.

A **span** is a subset of a trace, encompassing a logical unit of traversal.

A span may have a **parent** or **child** span, all under a parent trace.

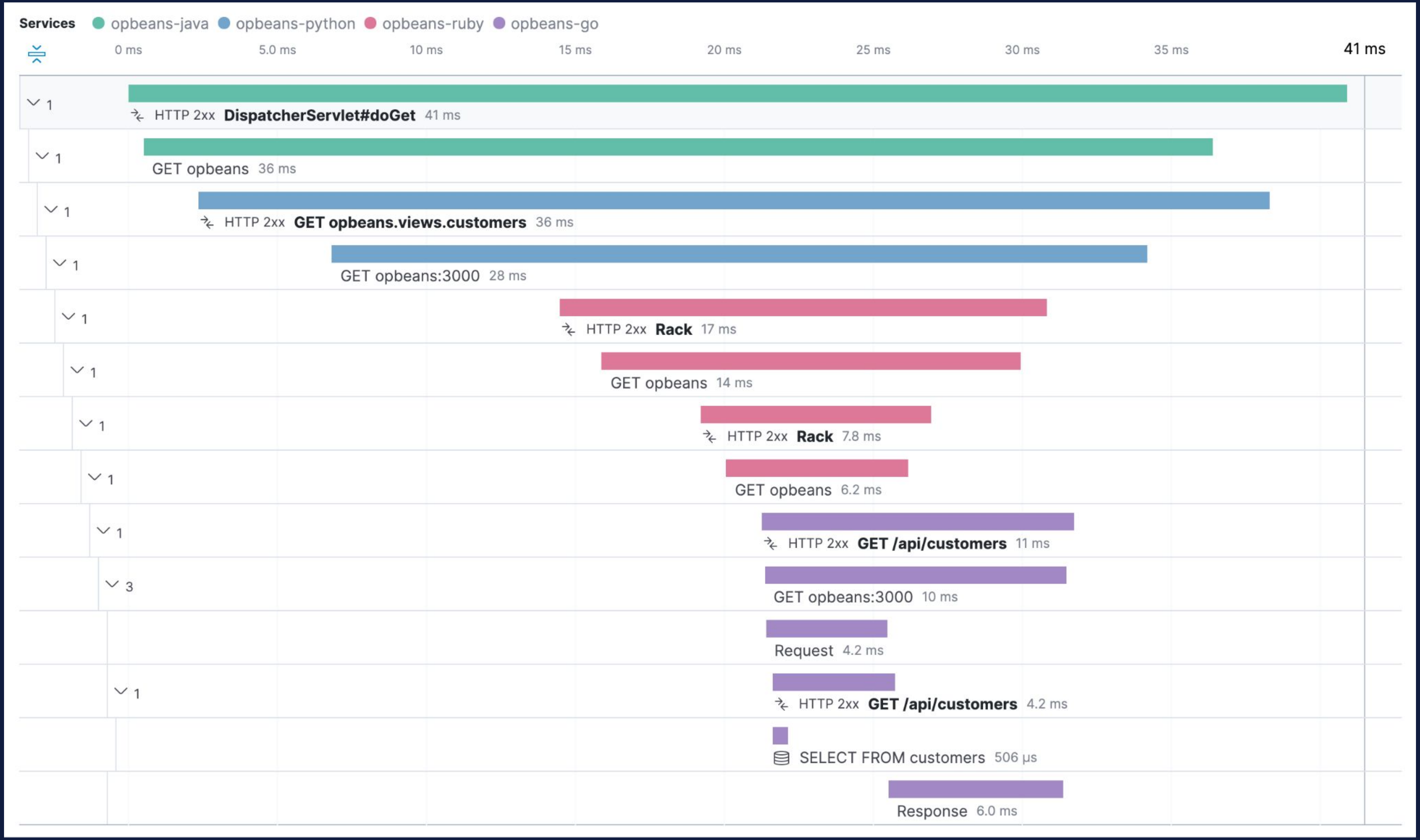
Observability with Traces



Trace sample |< < 1 of 1 > >| Investigate   View full trace

22 seconds ago | 800 µs (100% of trace) | http://localhost:1234/

Observability with Traces



Services ● opbeans-java ● opbeans-python ● opbeans-ruby ● opbeans-go
0 ms
5.0 ms
10 ms
15 ms
20 ms
25 ms
30 ms
35 ms
41 ms

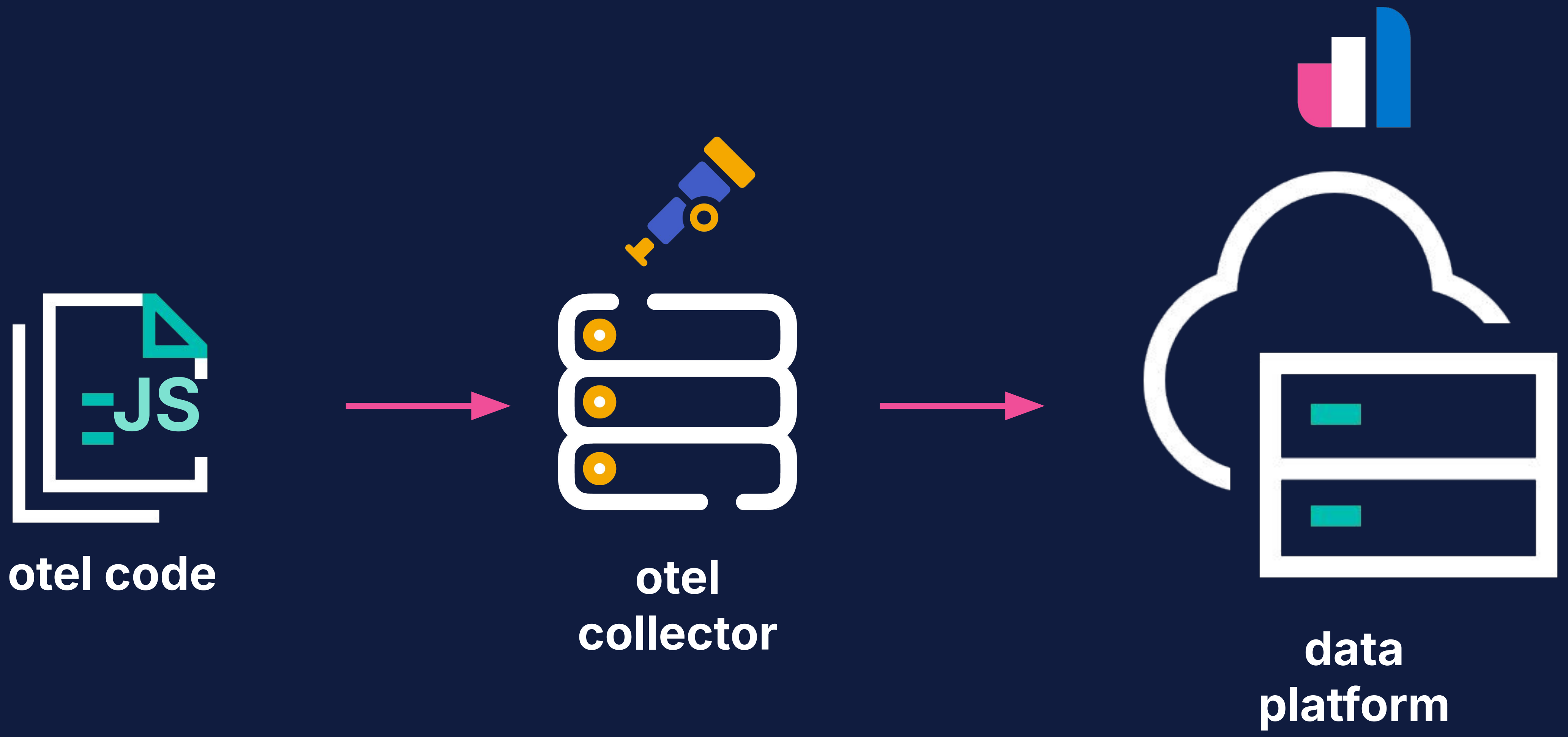
HTTP 2xx **DispatcherServlet#doGet** 41 ms
GET opbeans 36 ms
HTTP 2xx **GET opbeans.views.customers** 36 ms
GET opbeans:3000 28 ms
HTTP 2xx **Rack** 17 ms
GET opbeans 14 ms
HTTP 2xx **Rack** 7.8 ms
GET opbeans 6.2 ms
HTTP 2xx **GET /api/customers** 11 ms
GET opbeans:3000 10 ms
Request 4.2 ms
HTTP 2xx **GET /api/customers** 4.2 ms
SELECT FROM customers 506 μs
Response 6.0 ms

Observability implementation

Integrating traces and observability in the frontend:

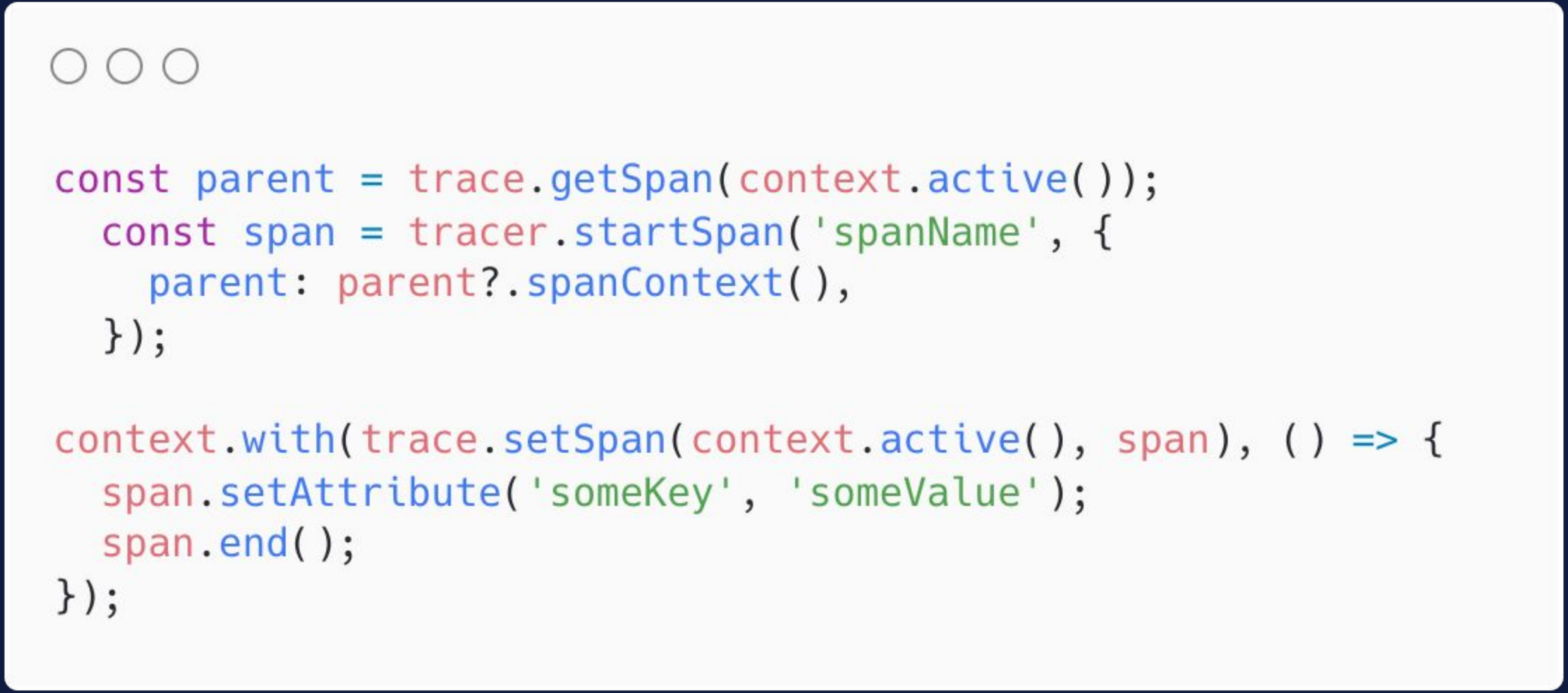
- @opentelemetry packages installed in app.js
- Node.js implementation
- Next.js & React integration
- Django and Flask ❤️s OTel, too

Observability implementation



Observability implementation

Manual Instrumentation - you create the spans

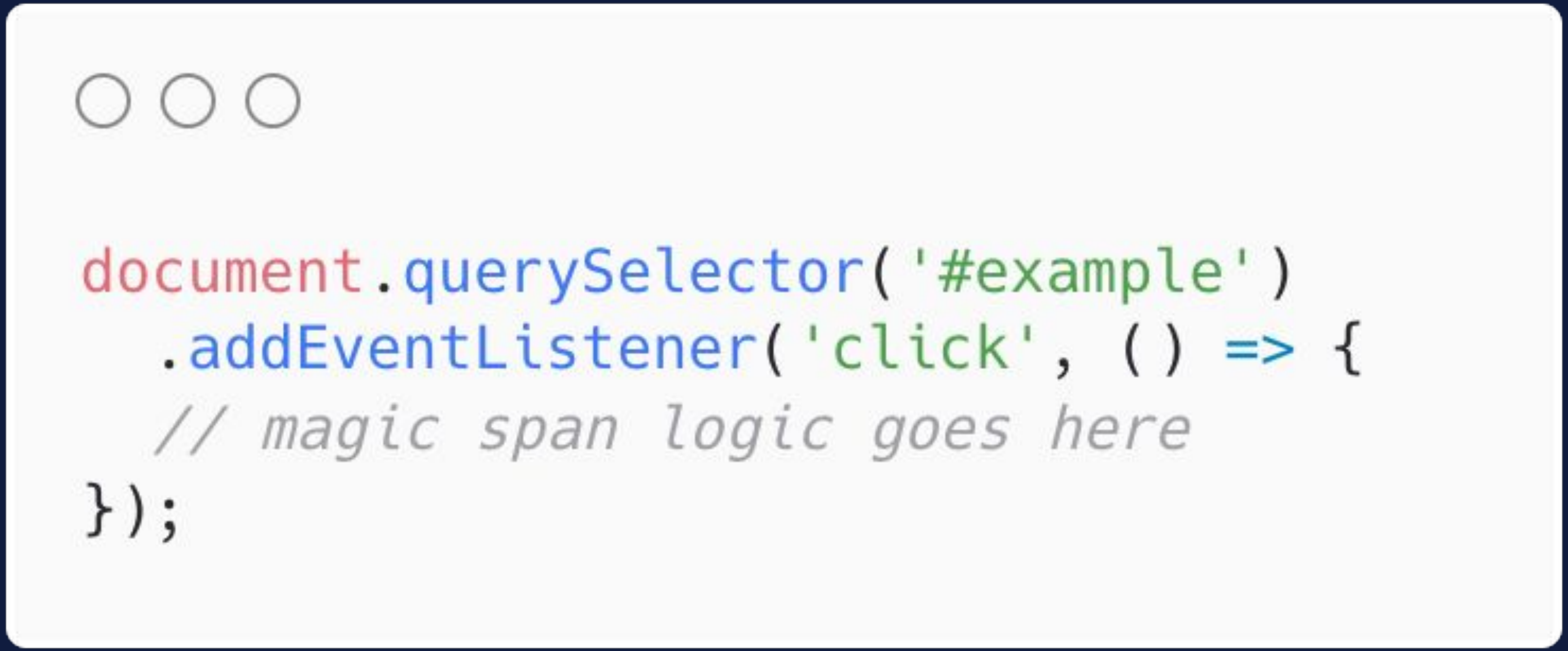


```
const parent = trace.getSpan(context.active());
  const span = tracer.startSpan('spanName', {
    parent: parent?.spanContext(),
  });

context.with(trace.setSpan(context.active(), span), () => {
  span.setAttribute('someKey', 'someValue');
  span.end();
});
```

Observability implementation

Manual Instrumentation - you create the spans



```
document.querySelector('#example')
  .addEventListener('click', () => {
  // magic span logic goes here
});
```

Observability implementation

Manual Instrumentation - you create the spans

But...

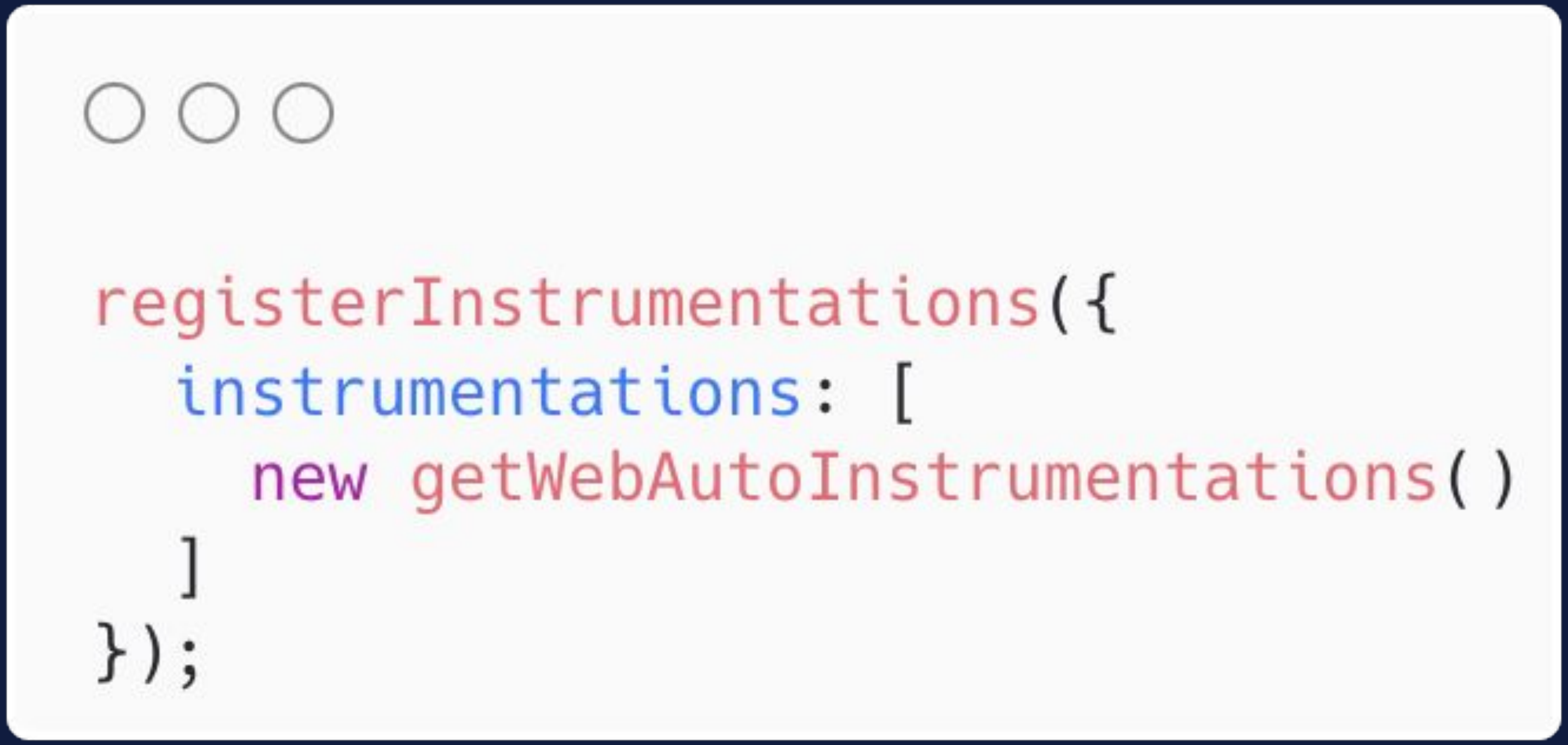
Observability implementation

Automatic Instrumentation - set it and forget it!

@opentelemetry/auto-instrumentations-web

Observability implementation

Automatic Instrumentation - set it and forget it!



```
registerInstrumentations({
  instrumentations: [
    new getWebAutoInstrumentations()
  ]
});
```

Observability implementation

Automatic Instrumentation - set it and forget it!

But…

Observability implementation

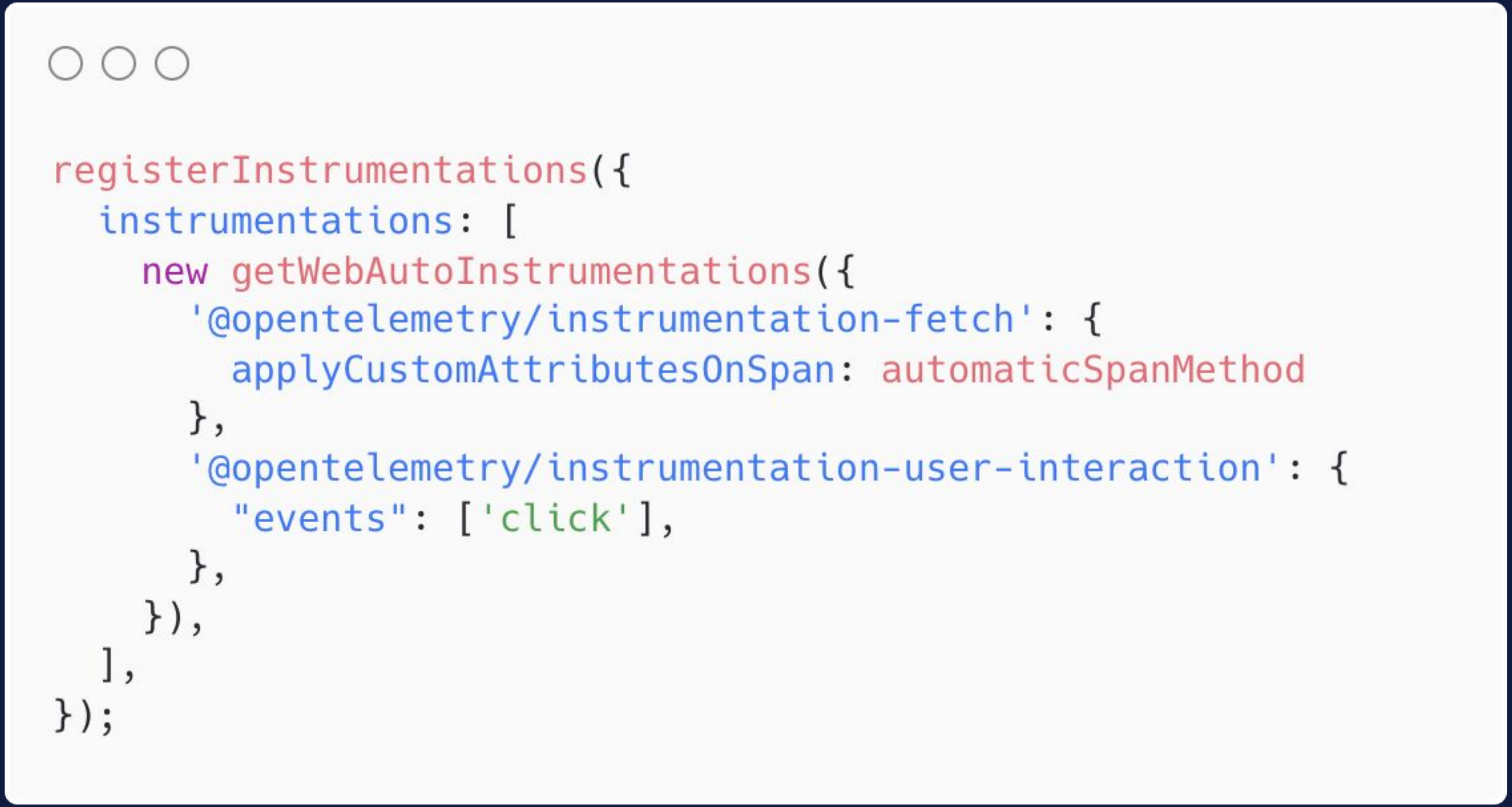
Automatic Instrumentation - set it and forget it!

@opentelemetry/auto-instrumentations-web

- @opentelemetry/instrumentation-document-load
- @opentelemetry/instrumentation-fetch
- @opentelemetry/instrumentation-user-interaction
- @opentelemetry/instrumentation-xml-http-request

Observability implementation

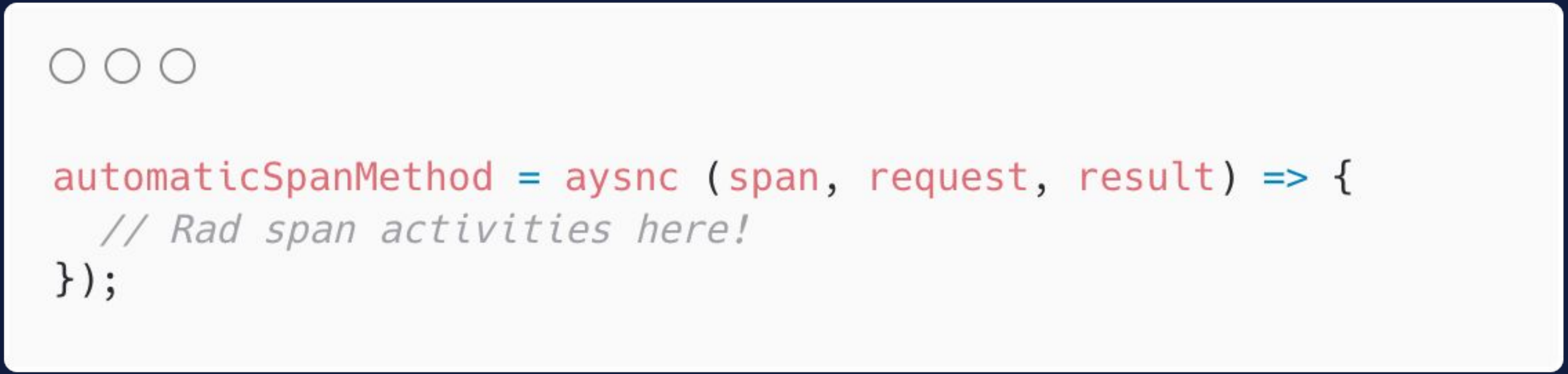
Automatic Instrumentation - set it and forget it!



```
registerInstrumentations({
  instrumentations: [
    new getWebAutoInstrumentations({
      '@opentelemetry/instrumentation-fetch': {
        applyCustomAttributesOnSpan: automaticSpanMethod
      },
      '@opentelemetry/instrumentation-user-interaction': {
        "events": ['click'],
      },
    }),
  ],
});
```

Observability implementation

Automatic Instrumentation - set it and forget it!



```
automaticSpanMethod = aysnc (span, request, result) => {
  // Rad span activities here!
});
```

Other options:
ignoreUrls, requestHook, ignoreNetworkEvents, measureRequestSize

Observability implementation

BONUS! Web-vitals instrumentation

Measure Core Web Vitals information for Realtime User Metrics (RUM)

Largest Content Paint (LCP): measure of time required to unload and load the necessary DOM data to display the largest content of a page.

Cumulative Layout Shift (CLS): measure of how often the layout shifts in the webpage load for the user. (we're looking at you, recipe pages)

Interaction to Next Paint (INP): a page's overall responsiveness to user interactions by observing the latency of all human interactions throughout the lifespan of a page visit

Observability implementation

OTel Collector

- Receive telemetry data from the browser via http
- Optionally process or transform it
    - CORS is handled
- Export it to a data platform using the OTLP exporter.
- Decouple instrumentation from backend observability systems.

Observability implementation

Data Platform

- Store incoming telemetry data
- Provide indexed search of observability history
- Create dashboards, alerts, and anomaly detection rules
- Expose an API for extended use of telemetry

Observability Demonstration

Demo time!

Observability Considerations

Should I do it this way?

Probably not.

Observability Recap

Frontend UI benefits from Observability (with OTel)

- active support for most common frameworks
- highly customizable to grow with you
- completes the journey of your application's usage path
- no tethers to a third party application
- most of if not all of the third-party services you use support Open Telemetry clients.
- RUM is rolled right into the process

Thank you!

https://github.com/justincastilla/vanilla-browser-otel