An Introduction To Distributed Tracing

Thomas Watson, @wa7son
Principal Software Engineer, Elastic

Who am I?

- Thomas Watson
- Open Source developer at github.com/watson
- Principal Software Engineer at Elastic
- Elastic APM – Node.js agent
- Node.js Core Member
- Tweets as @wa7son

glossary

Tracing

*"In software engineering, **tracing** involves a specialized use of logging to record information about a program's execution. This information is typically used by programmers for debugging purposes [...]"*

Source: Wikipedia

glossary

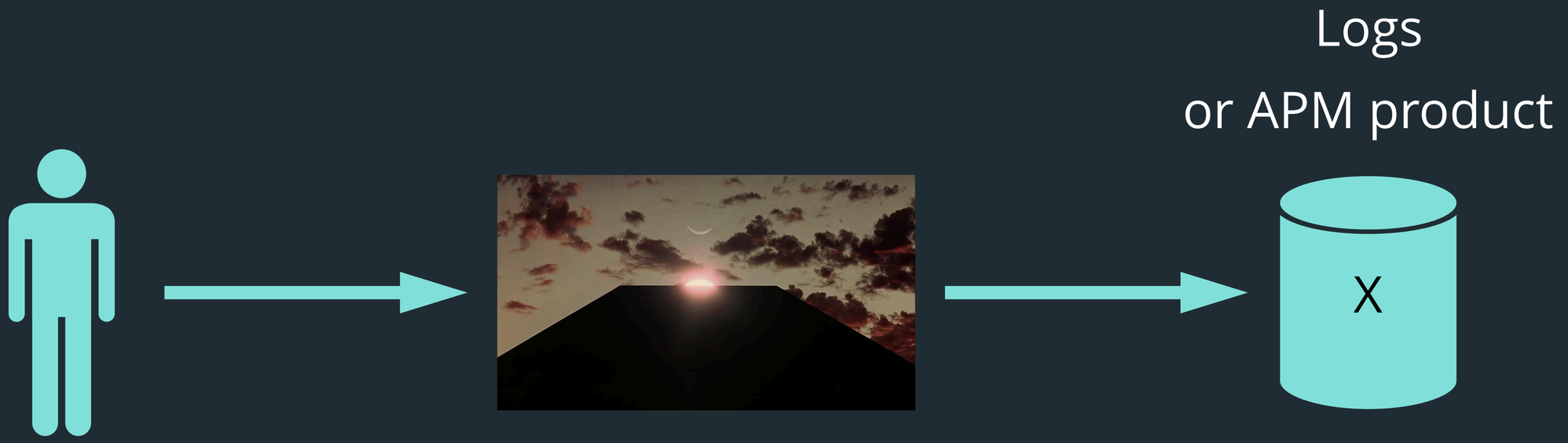
APM

Application **P**erformance **M**onitoring

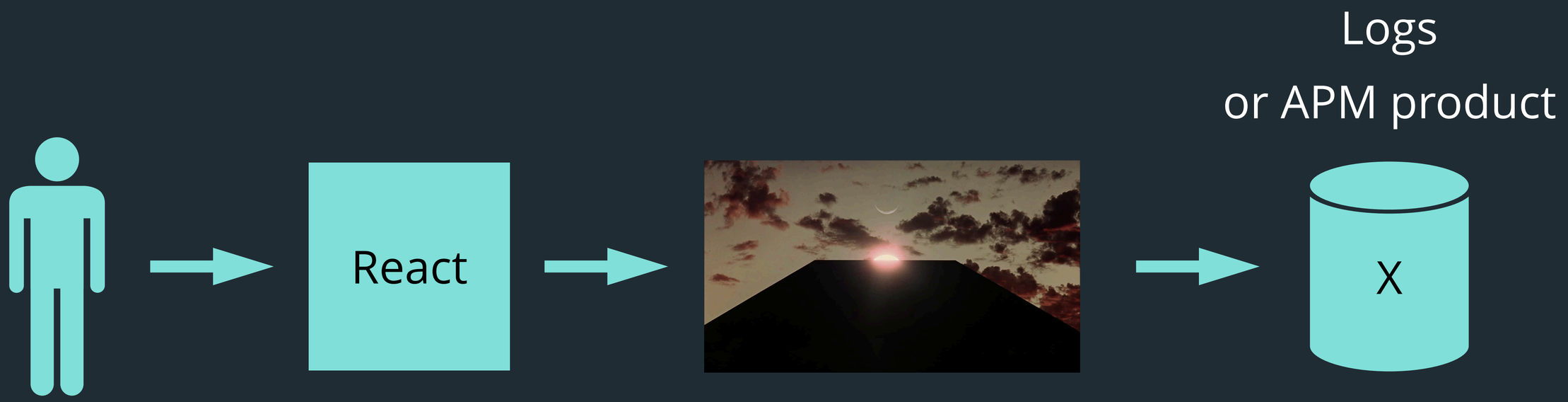
(Management)

Tracing The Monolith

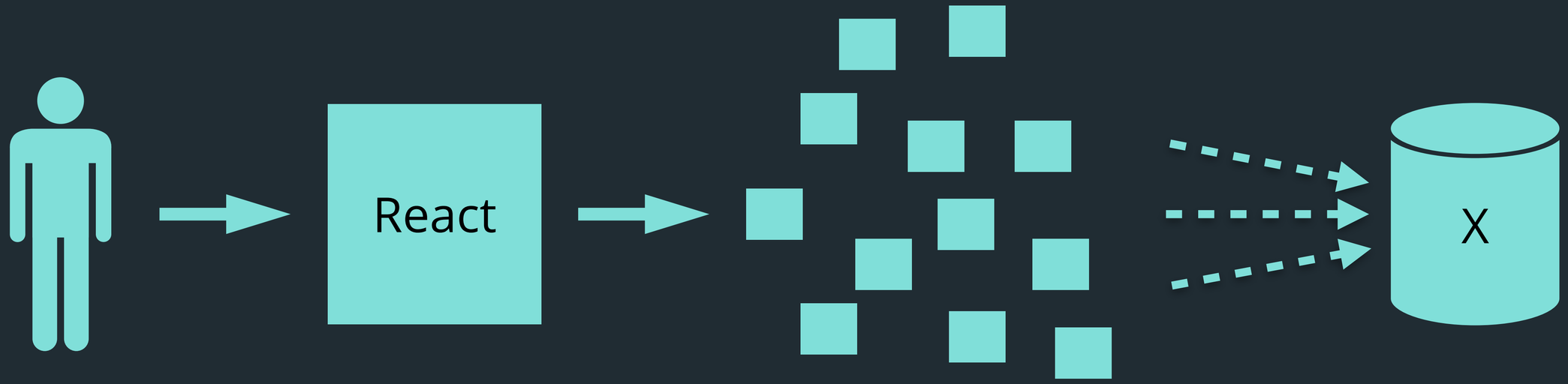
Tracing The Monolith



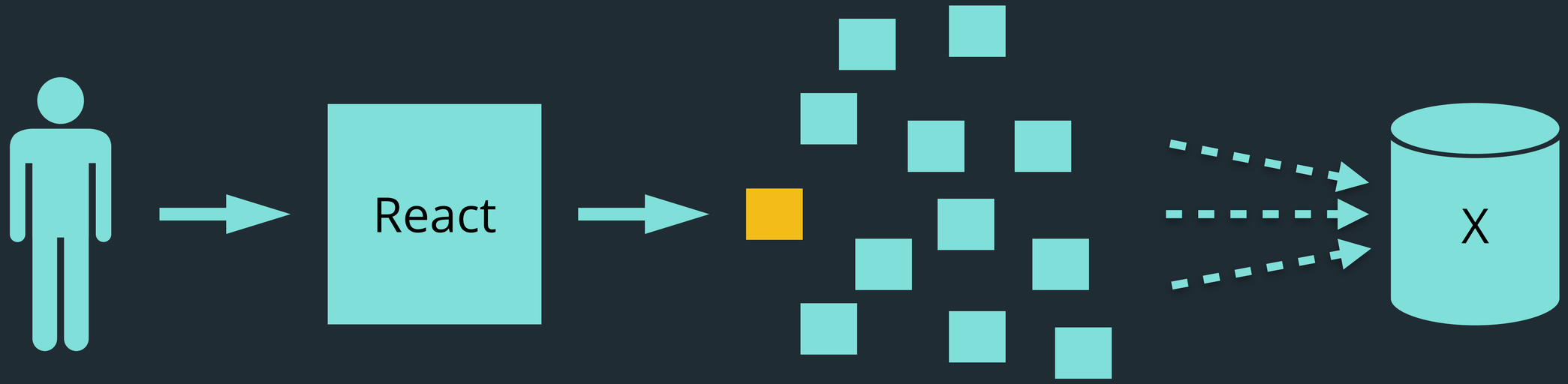
Tracing The Monolith



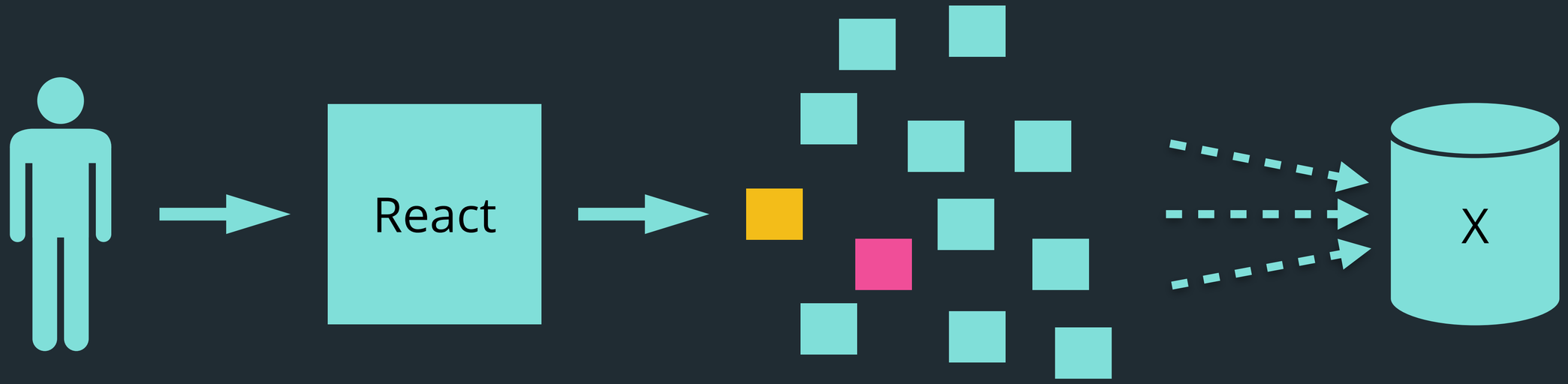
Tracing Microservices



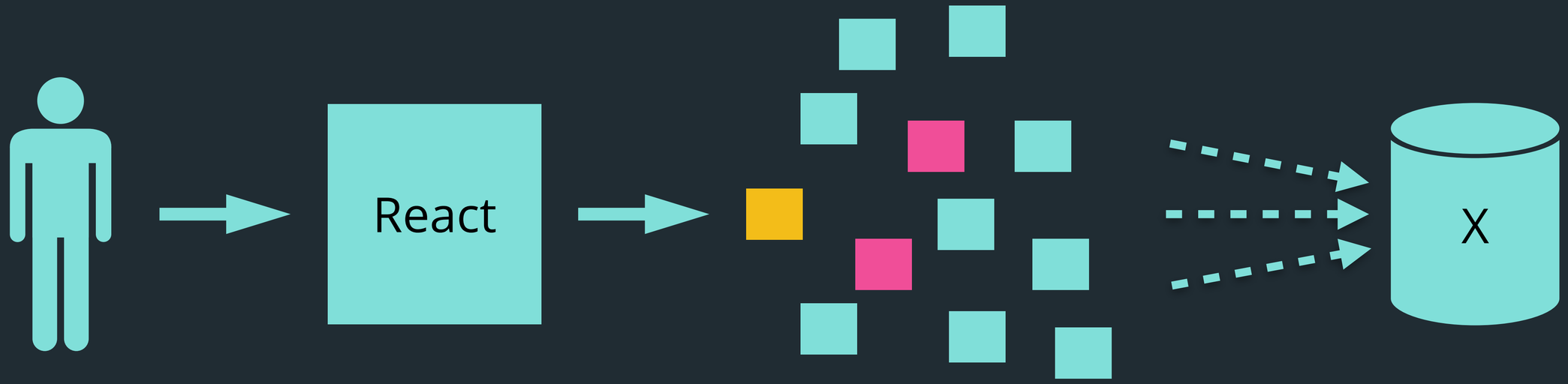
Tracing Microservices



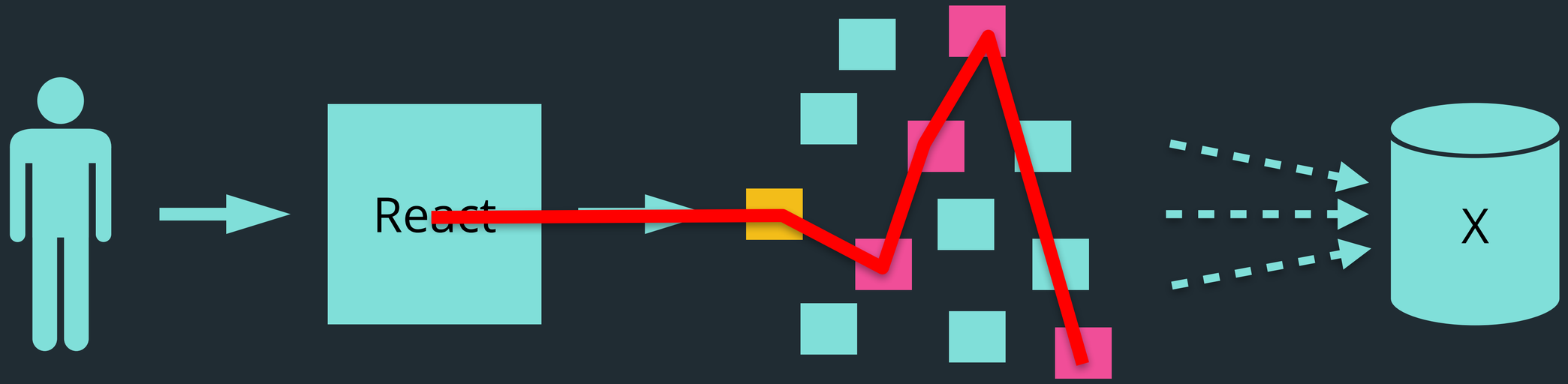
Tracing Microservices



Tracing Microservices



Tracing Microservices



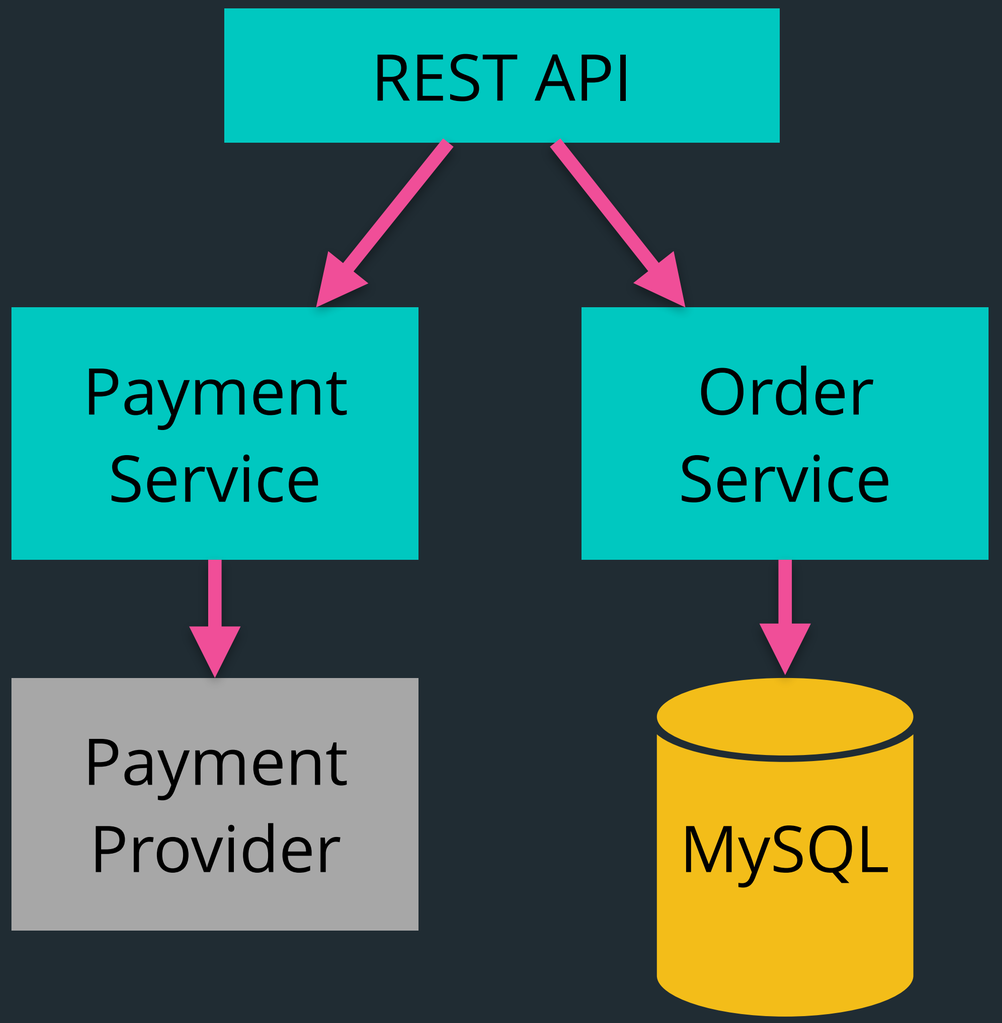
Distributed Tracing

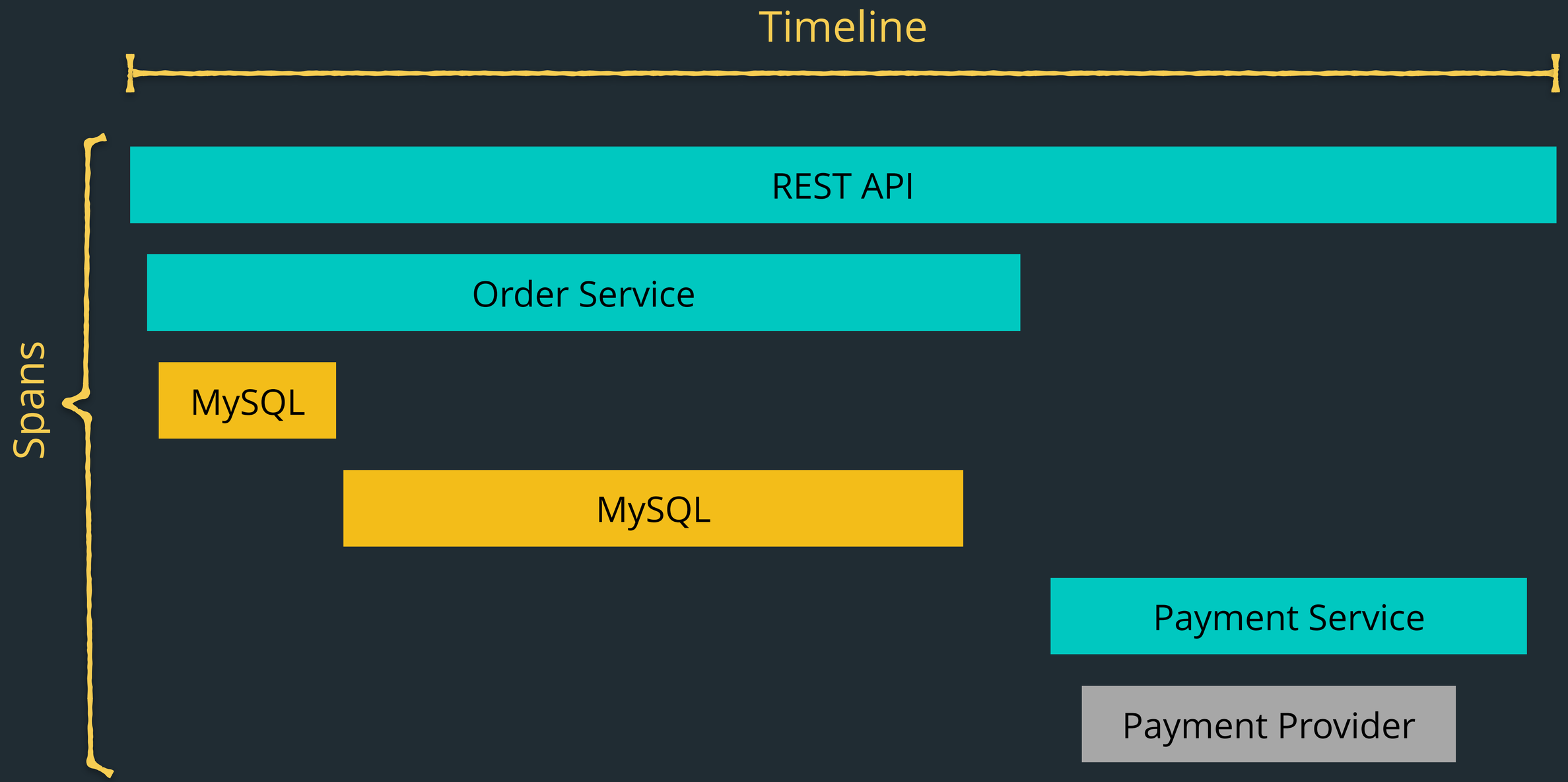
elastic

@wa7son

Agents, Clients, and Tracers

The APM library with many names

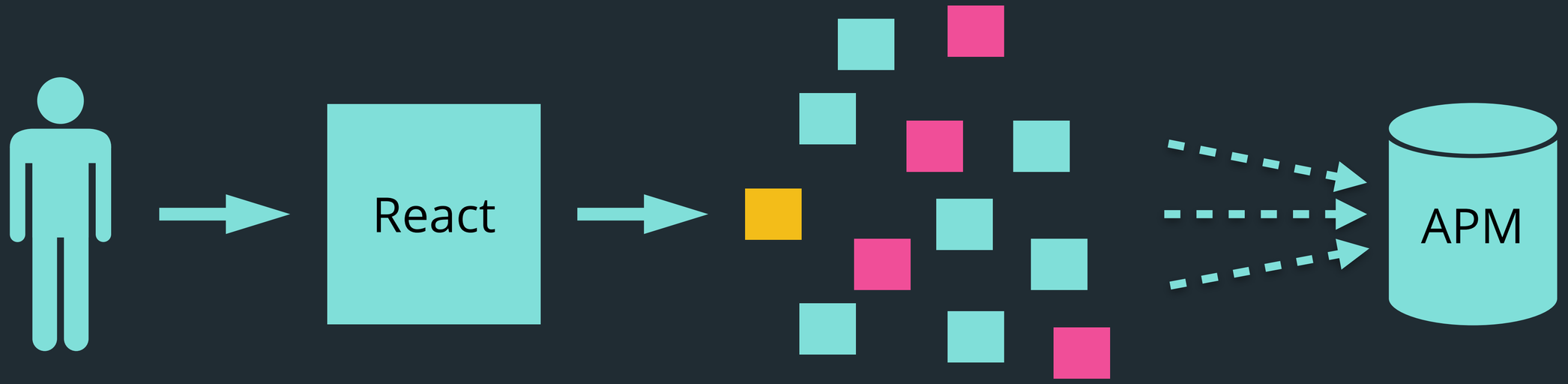

REST API
Payment Service
Order Service
Payment Provider
MySQL



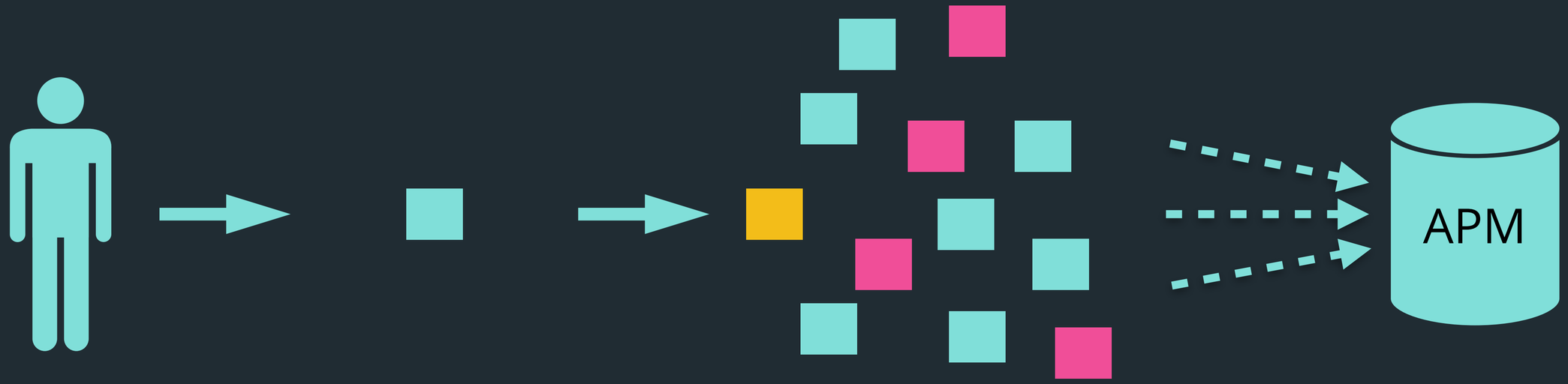
elastic

@wa7son

Tracing Microservices



Tracing Microservices



Tracing Microservices

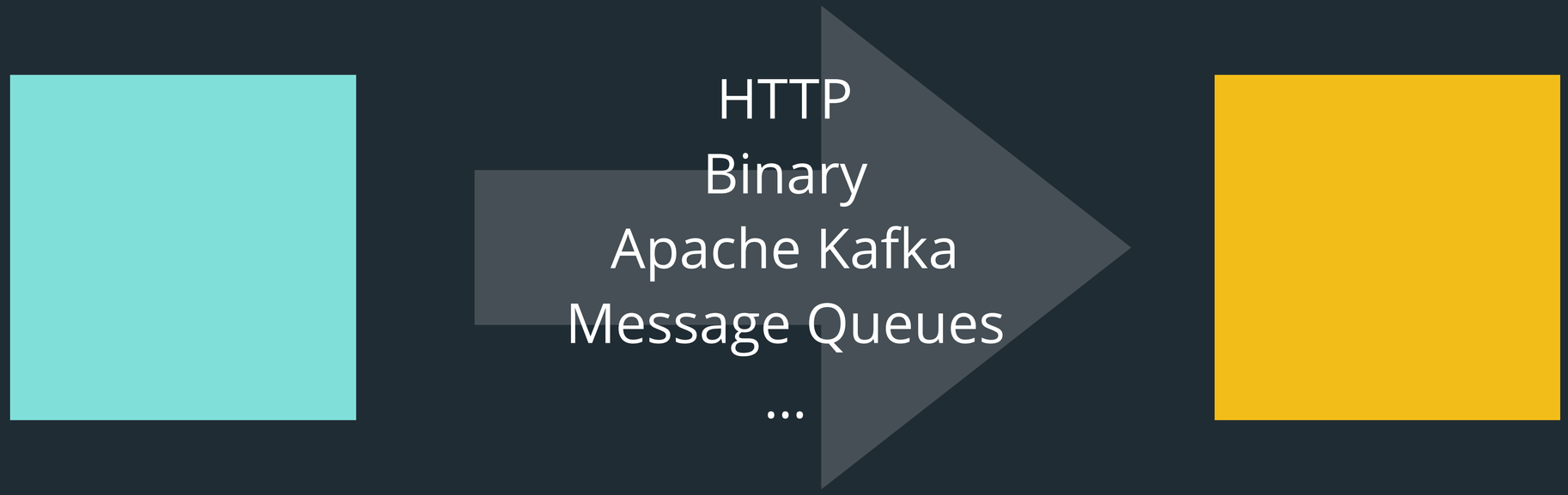

Source: Giphy

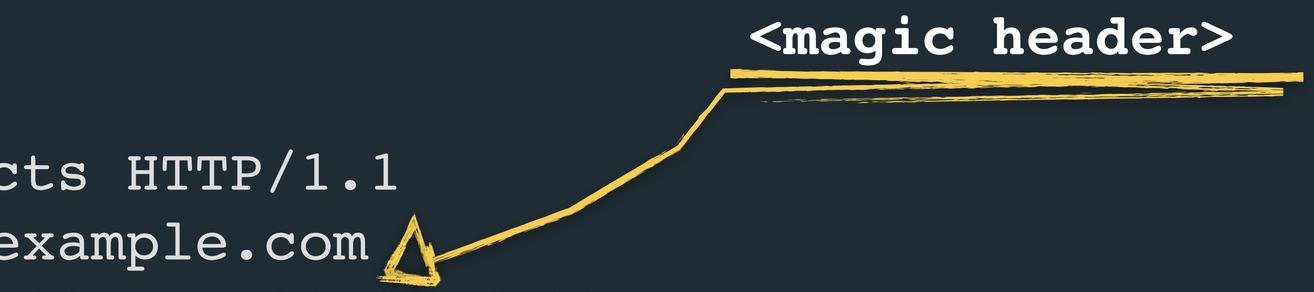
Challenges

elastic

@wa7son

Interservice Communication






```
GET /products HTTP/1.1
Host: www.example.com
Date: Mon, 29 Oct 2018 16:11:05 GMT
Connection: keep-alive
Content-Length: 0
```

```
GET /products HTTP/1.1
Host: www.example.com
<magic header>
Date: Mon, 29 Oct 2018 16:11:05 GMT
Connection: keep-alive
Content-Length: 0
```

```
GET /products HTTP/1.1
Host: www.example.com
traceparent: 00-82c5500f40667e5500e9ae8e9711553c-992631f881f78c3b-01
Date: Mon, 29 Oct 2018 16:11:05 GMT
Connection: keep-alive
Content-Length: 0
```

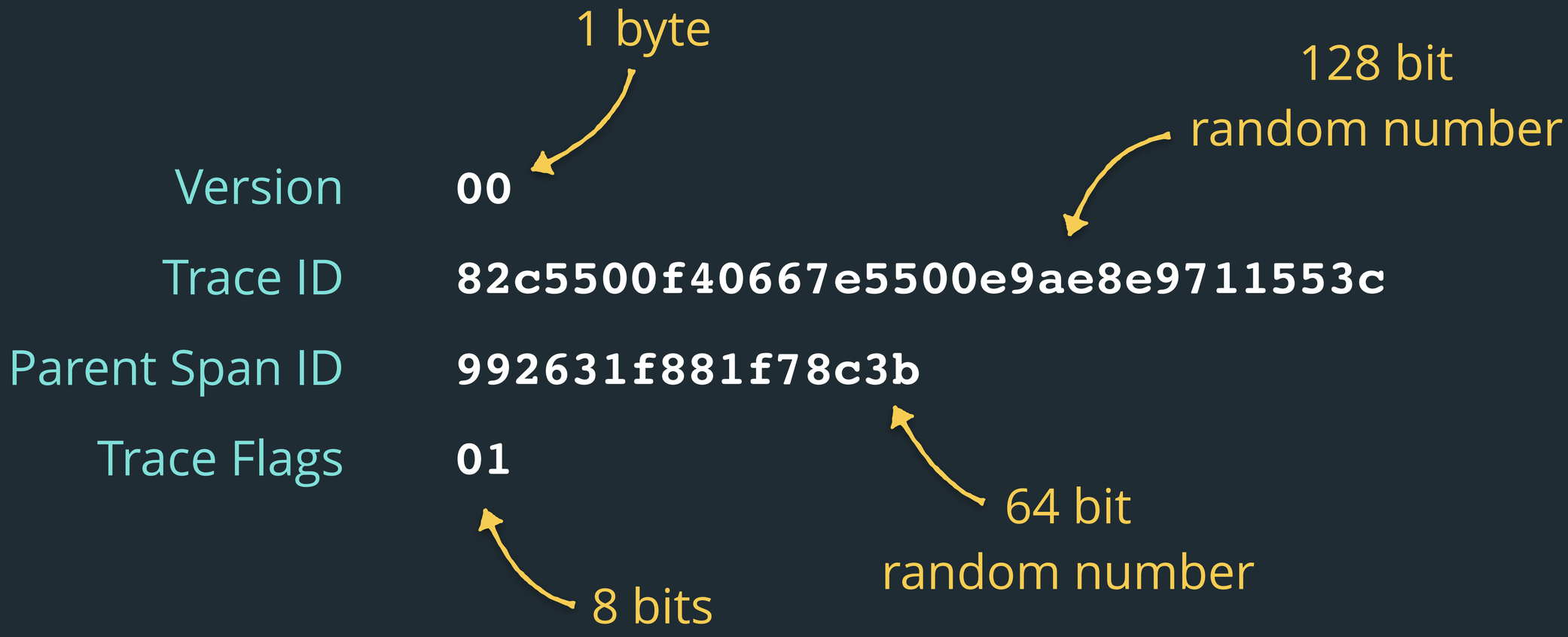
DRAFT

```
GET /products HTTP/1.
Host: www.example.com
traceparent: 00-82c55
                                                  -992631f881f78c3b-01
Date: Mon, 29 Oct 201
Connection: keep-alive
Content-Length: 0
```

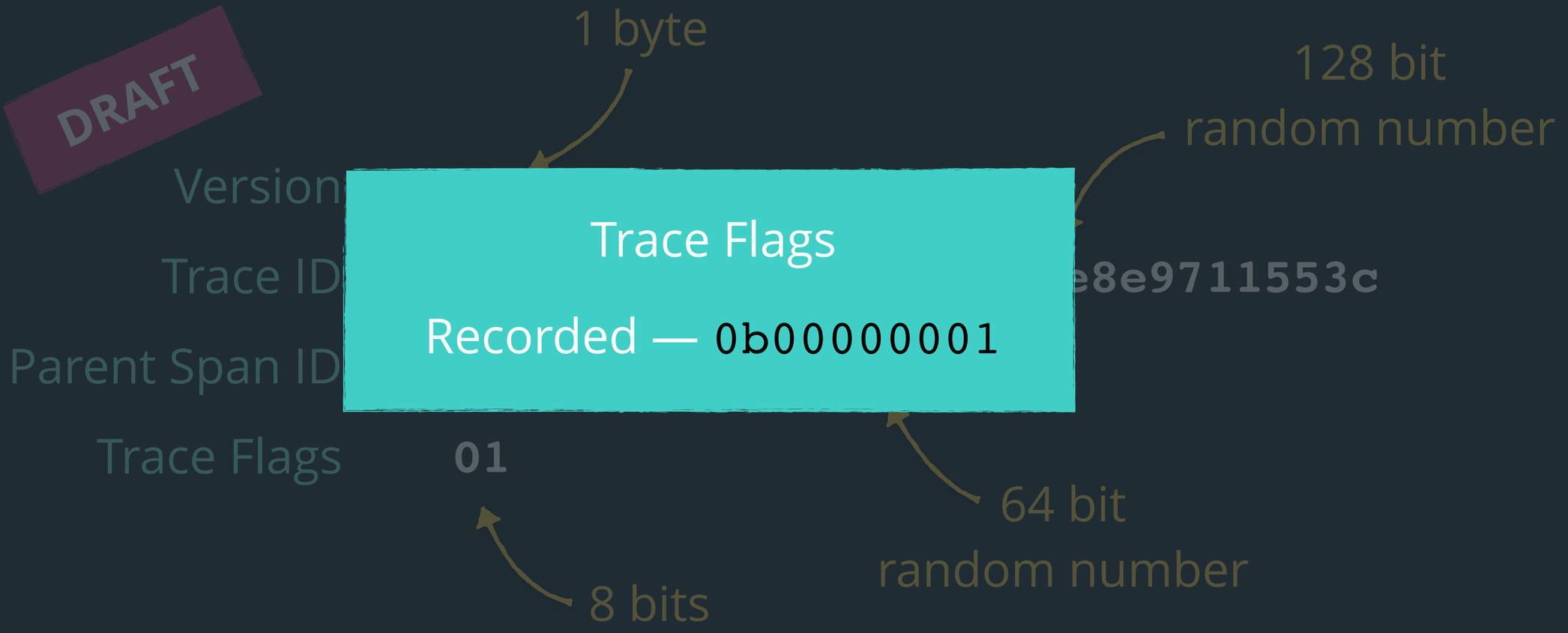
Trace Context Working Group

```
GET /products HTTP/1.1
Host: www.example.com
traceparent: 00-82c5500f40667e5500e9ae8e9711553c-992631f881f78c3b-01
Date: Mon, 29 Oct 2018 16:11:05 GMT
Connection: keep-alive
Content-Length: 0
```

traceparent



traceparent



Multi-Vendor Environments

```
GET /products HTTP/1.1
Host: www.example.com
traceparent: 00-82c5500f40667e5500e9ae8e9711553c-992631f881f78c3b-01
Date: Mon, 29 Oct 2018 16:11:05 GMT
Connection: keep-alive
Content-Length: 0
```


```
GET /products HTTP/1.1
Host: www.example.com
traceparent: 00-82c5500f40667e5500e9ae8e9711553c-992631f881f78c3b-01
tracestate: <vendorname1=opaqueValue1>,<vendorname2=opaqueValue2>,…
Date: Mon, 29 Oct 2018 16:11:05 GMT
Connection: keep-alive
Content-Length: 0
```

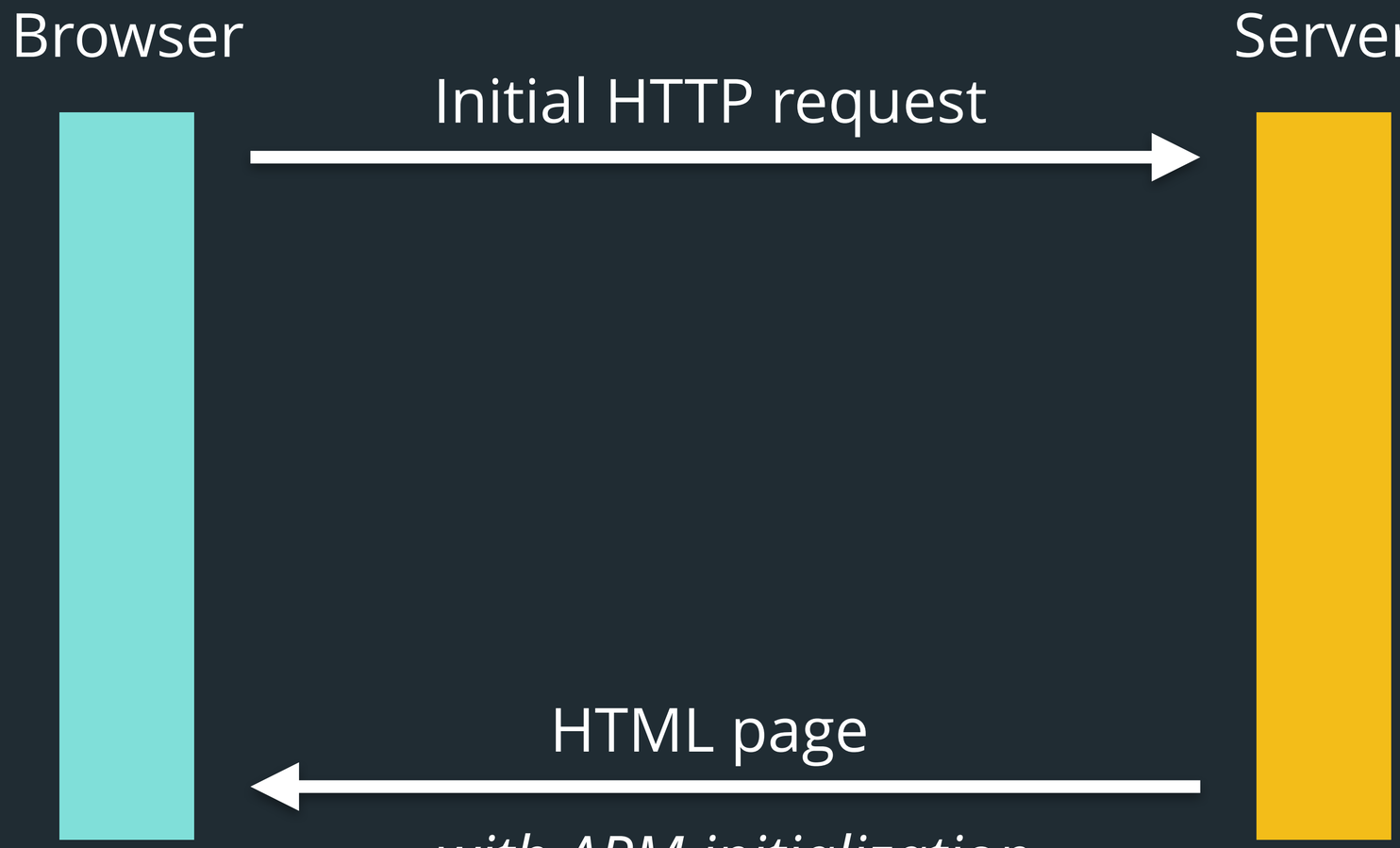
tracestate

Up to 32 unique vendors

```
<vendorname1=opaqueValue1>,<vendorname2=opaqueValue2>,…
```

Parent
(potentially vendor specific)

Bootstrapping the Trace in Browsers



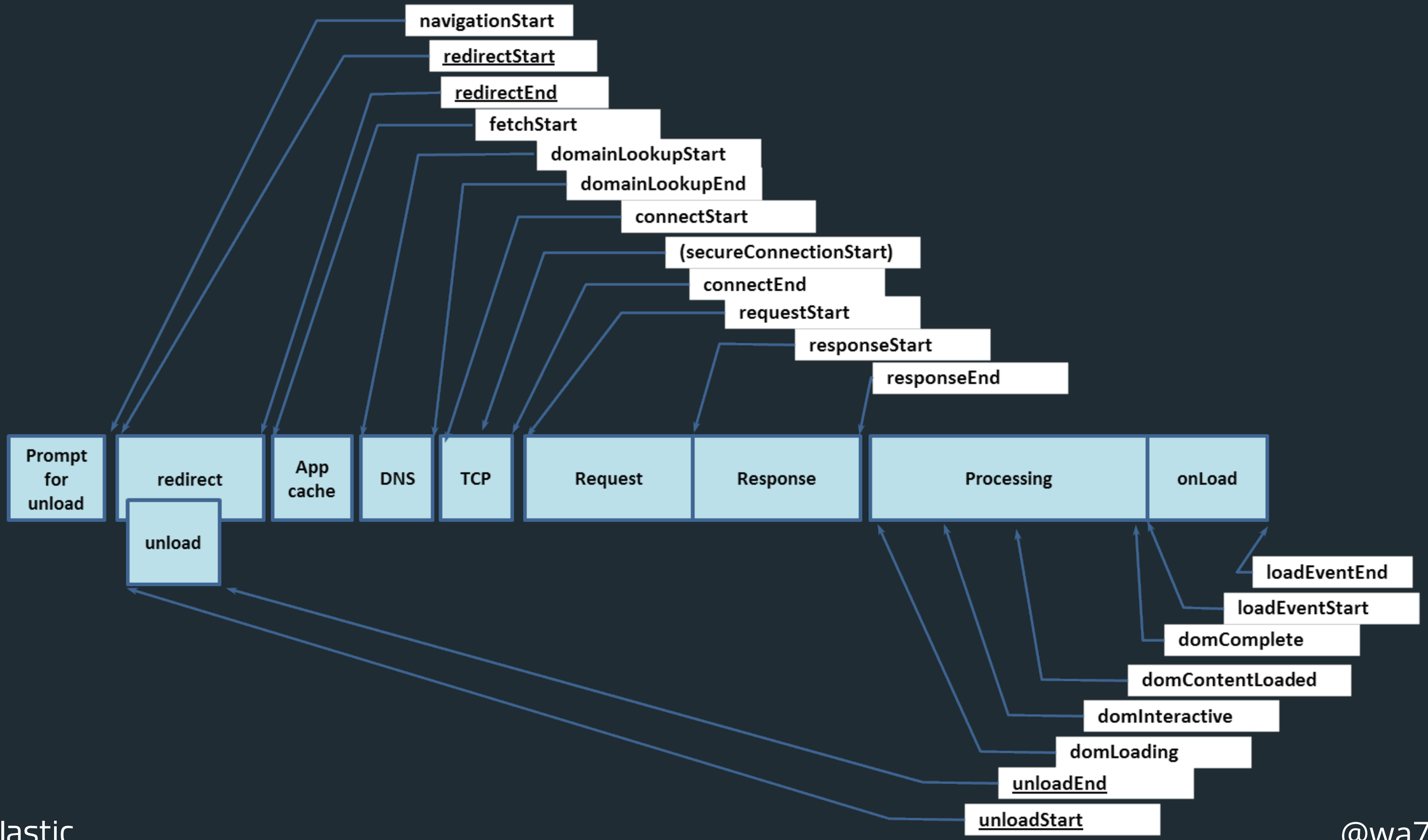
1. Generate trace id
2. Start server span
3. Generate span ID for initial HTTP request on behalf of the browser
4. Render HTML template
5. Respond to request + End server span

```
const Html = ({ body, transaction }) => `
  <!DOCTYPE html>
  <html>
    <head>
    </head>
    <body>
      <script src="/path/to/elastic-apm-js-base.umd.min.js"></script>
      <script>
        elasticApm.init({
          pageLoadTraceId: '${transaction.traceId}',
          pageLoadSpanId: '${transaction.ensureParentId()}',
          pageLoadSampled: ${transaction.sampled}
        })
      </script>
      <div>${body}</div>
    </body>
  </html>
`
```

```
const Html = ({ body, transaction }) => `
  <!DOCTYPE html>
  <html>
    <head>
    </head>
    <body>
      <script src="/path/to/elastic-apm-js-base.umd.min.js"></script>
      <script>
        elasticApm.init({
          pageLoadTraceId: '${transaction.traceId}',
          pageLoadSpanId: '${transaction.ensureParentId()}',
          pageLoadSampled: ${transaction.sampled}
        })
      </script>
      <div>${body}</div>
    </body>
  </html>
`
```

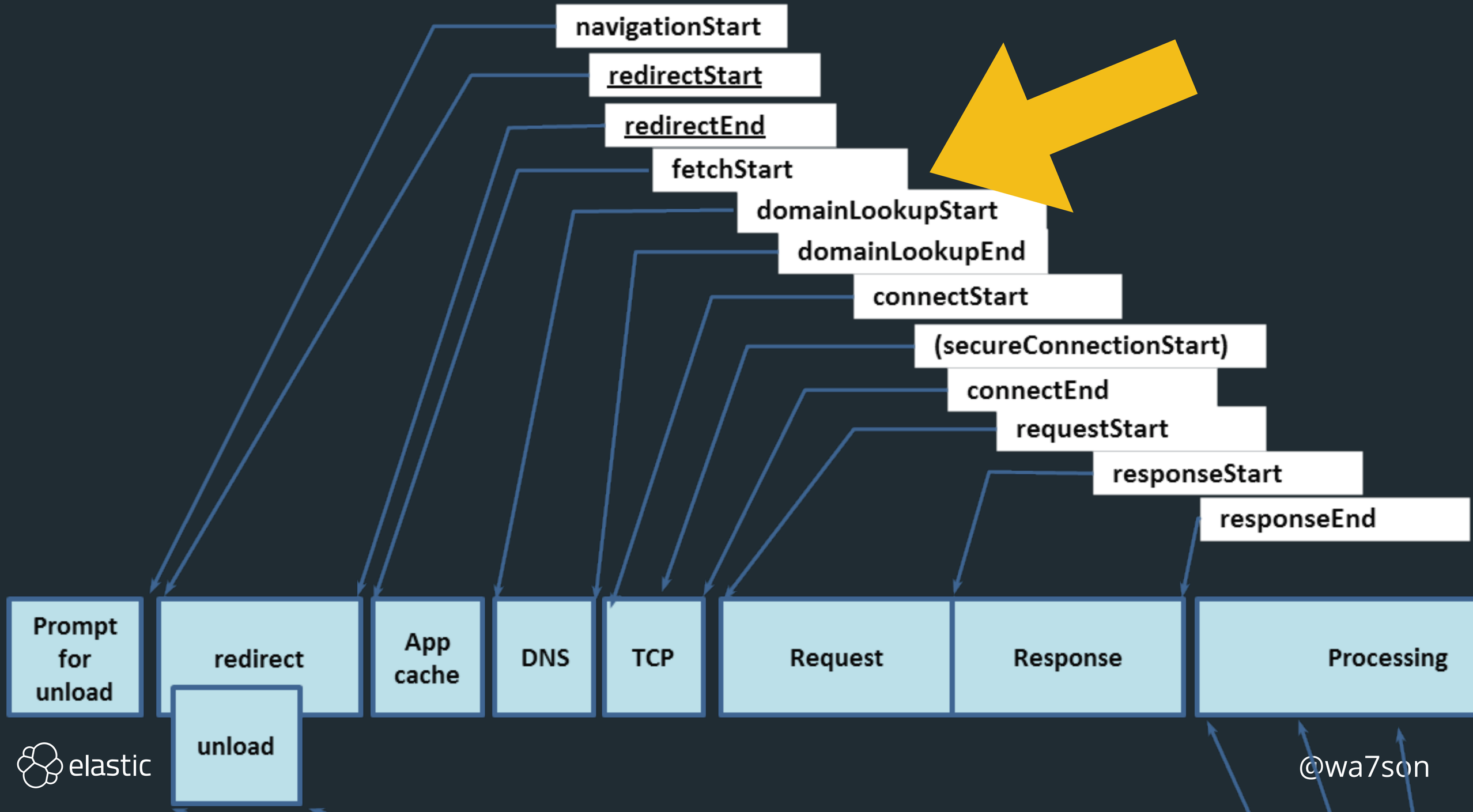
Navigation Timing API

developer.mozilla.org/docs/Web/API/Navigation_timing_API



elastic

@wa7son


navigationStart
redirectStart
redirectEnd
fetchStart
domainLookupStart
domainLookupEnd
connectStart
(secureConnectionStart)
connectEnd
requestStart
responseStart
responseEnd
Prompt for unload
redirect
unload
App cache
DNS
TCP
Request
Response
Processing
elastic
@wa7son

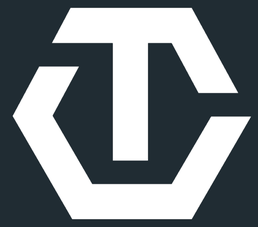

COLLECT
Don’t be this guy
ALL THE DATA
imgflip.com

Async

With great power comes great responsibility

Standards

Trace Context Working Group



OPENTRACING

elastic

@wa7son

```
const opentracing = require('opentracing')

const tracer = new opentracing.Tracer()

const span = tracer.startSpan('some work')

// Do some work

span.finish()
```

```
span.setTag('user.id', user.getId())

span.log({
  'event': ‘error',
  'error.object': err
})
```

```
// Start a new (parentless) root Span:
const parent = Tracer.startSpan('DoWork')

// Start a new (child) Span:
const child = Tracer.startSpan('load-from-db', {
    childOf: parent.context()
})

// Start a new async (FollowsFrom) Span:
const child = Tracer.startSpan('async-cache-write', {
    references: [
        opentracing.followsFrom(parent.context())
    ]
})
```

```
const headersCarrier = {}

// "Serialize" Span Context into headersCarrier
tracer.inject(
  clientSpan.context(),
  Tracer.FORMAT_HTTP_HEADERS,
  headersCarrier
)

// Add tracing headers to headers of outgoing request
Object.assign(outboundHTTPReq.headers, headersCarrier)
```

```
// "De-serialize" Span Context from inbound HTTP request headers
const headersCarrier = inboundHTTPReq.headers

const wireCtx = tracer.extract(
  Tracer.FORMAT_HTTP_HEADERS,
  headersCarrier
)

const serverSpan = tracer.startSpan('...', { childOf: wireCtx })
```

```
Tracer.FORMAT_TEXT_MAP
Tracer.FORMAT_HTTP_HEADERS
Tracer.FORMAT_BINARY
```

```
const { Tracer } = require('elastic-apm-node/opentracing')

const tracer = new Tracer({
    serviceName: 'my-awesome-service',
    serverUrl: 'https://example.com:8200'
})

const opentracing = require('opentracing')

opentracing.initGlobalTracer(tracer)

// then later...
const tracer = opentracing.globalTracer()
```

```
const { Tracer } = require('my-custom-tracer')
const tracer = new Tracer()

const express = require('express')
const expressOpenTracing = require('express-opentracing')

const app = express()

app.use(expressOpenTracing({ tracer }))

// ...
```


Source: Giphy

Deployment

elastic

@wa7son

Checklist

- Does it support the languages we’re using (Node.js, Python, Java etc)?
- Does it support “serverless” infrastructures (e.g. AWS Lambda)?
- Does it support the databases we’re using?
- Does it support the inner-service transports we’re using (HTTP, Kafka etc)?
- Does it support Real User Monitoring (RUM) for the platforms we’re using (e.g. JavaScript, React Native)?
- If the answer isn’t yes to all of the above, does it allow us to extend it easily?
- Can we access the raw trace data?
- Can we correlate the trace data with other data sources (logs, metrics etc)?

Recommendations

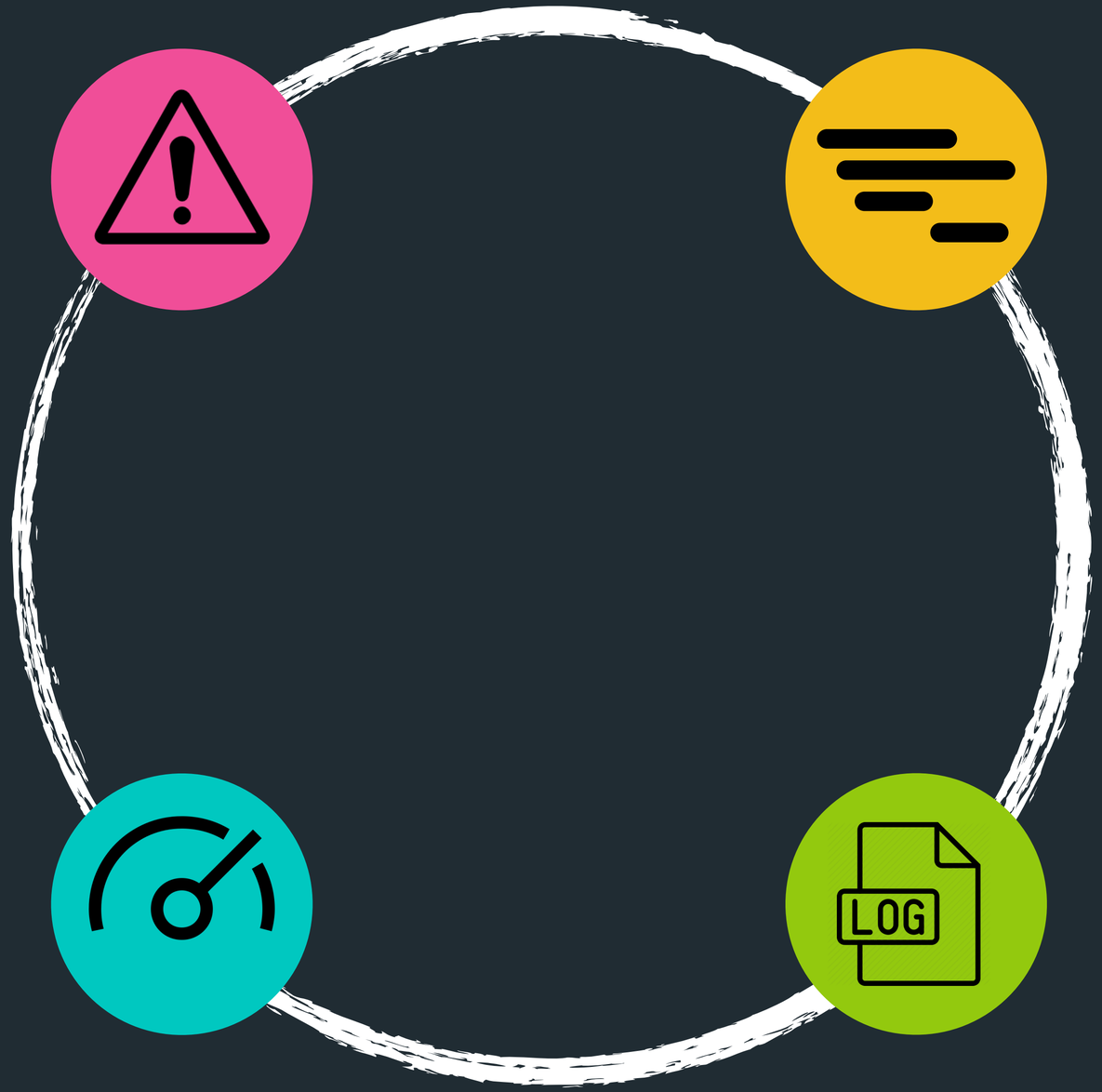
- Don’t try to instrument everything in the beginning
  - Instrument important services first

- Breaking up a monolith:
  - Add instrumentation to the monolith first if possible
  - Make instrumentation part of the process of breaking out new microservices

Data, data, data

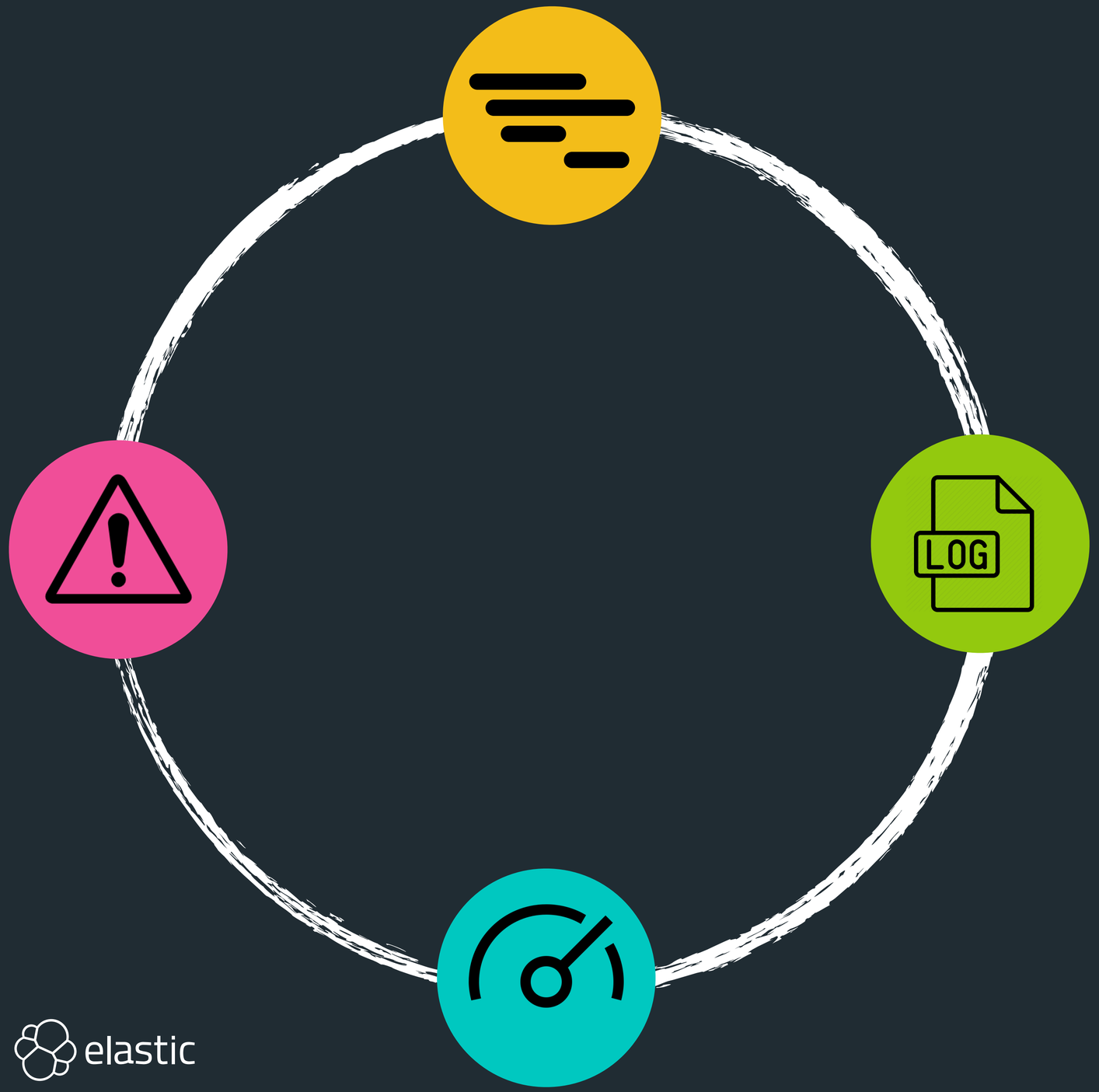
elastic

@wa7son

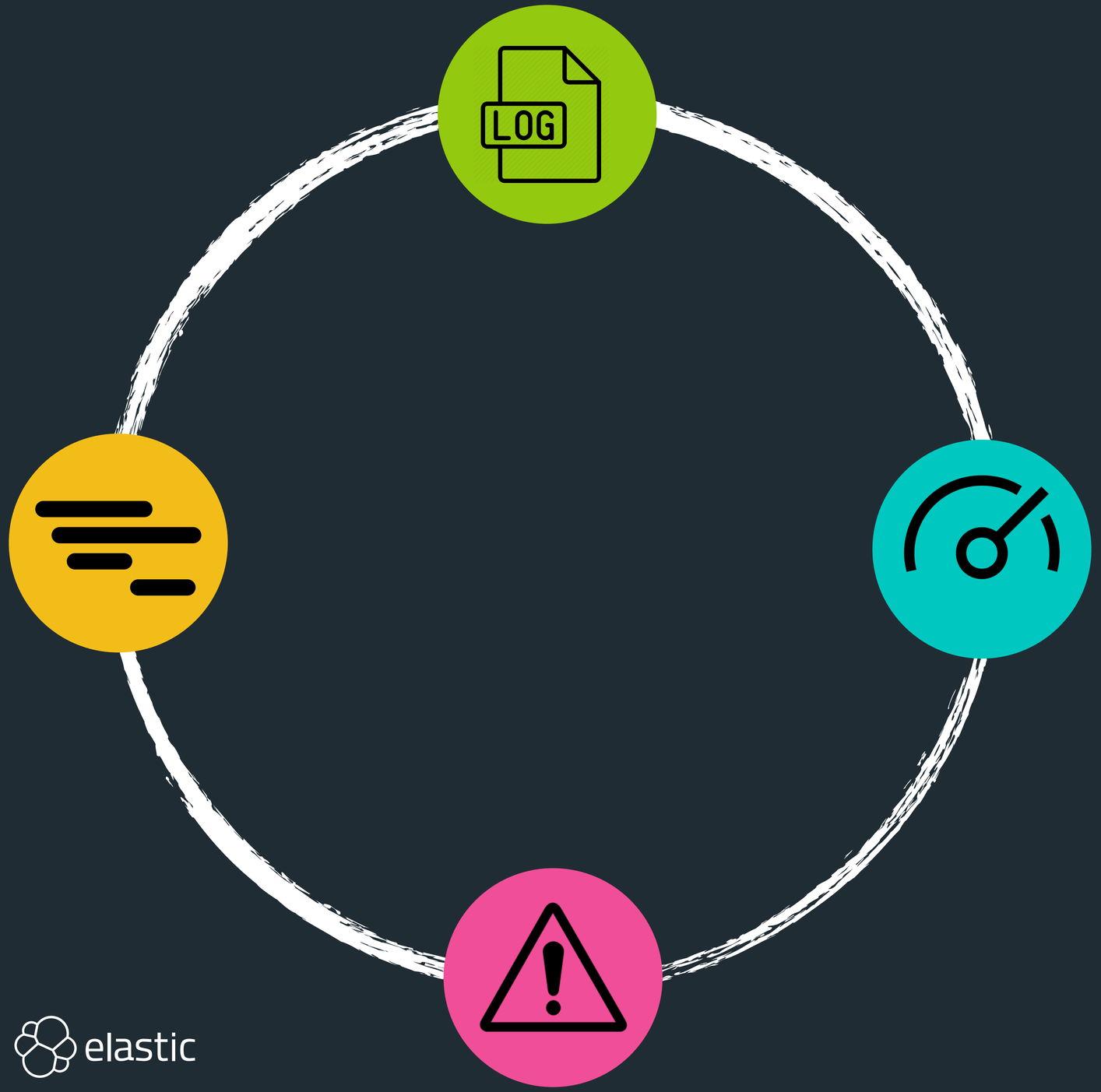


elastic

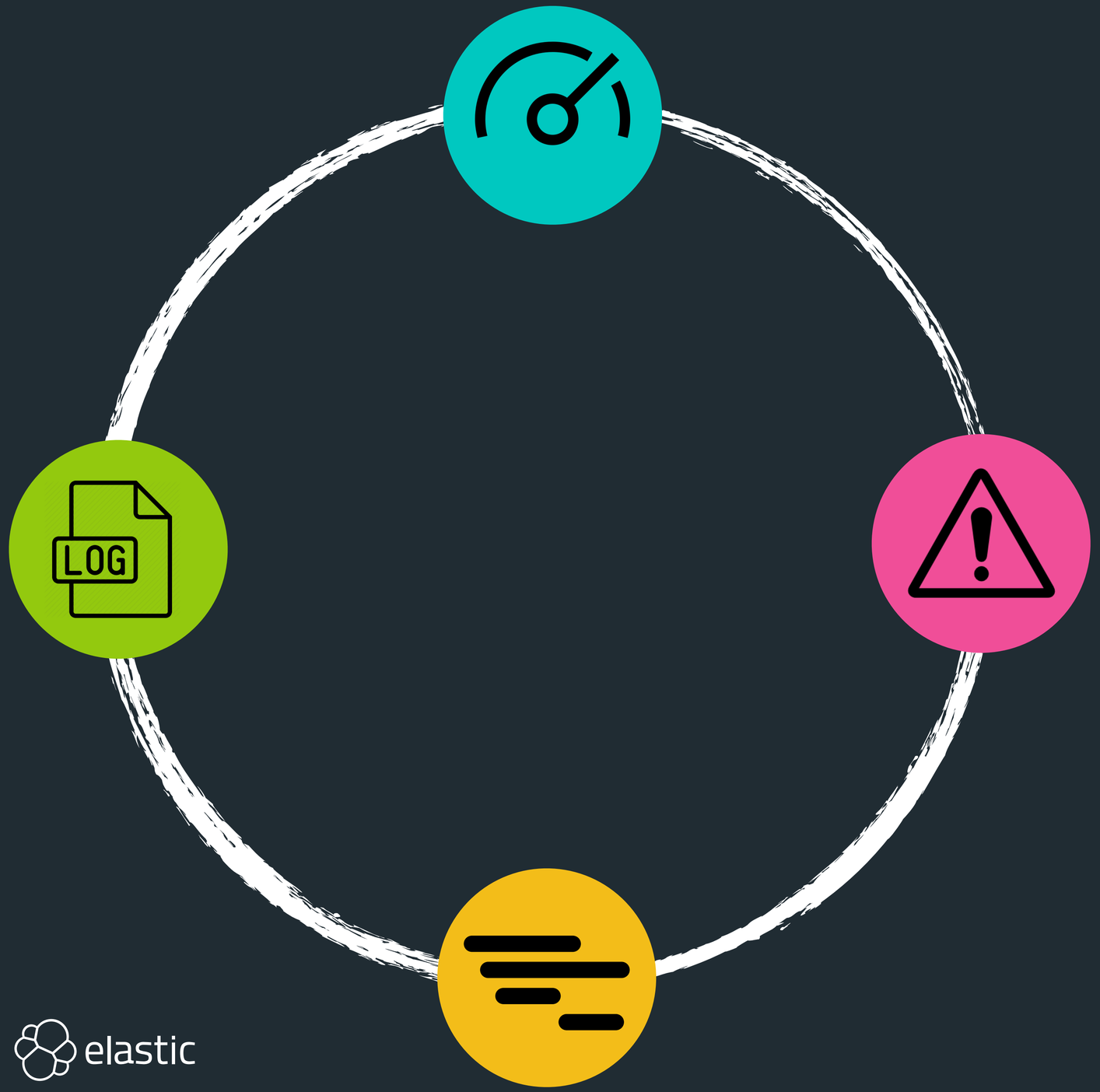
@wa7son



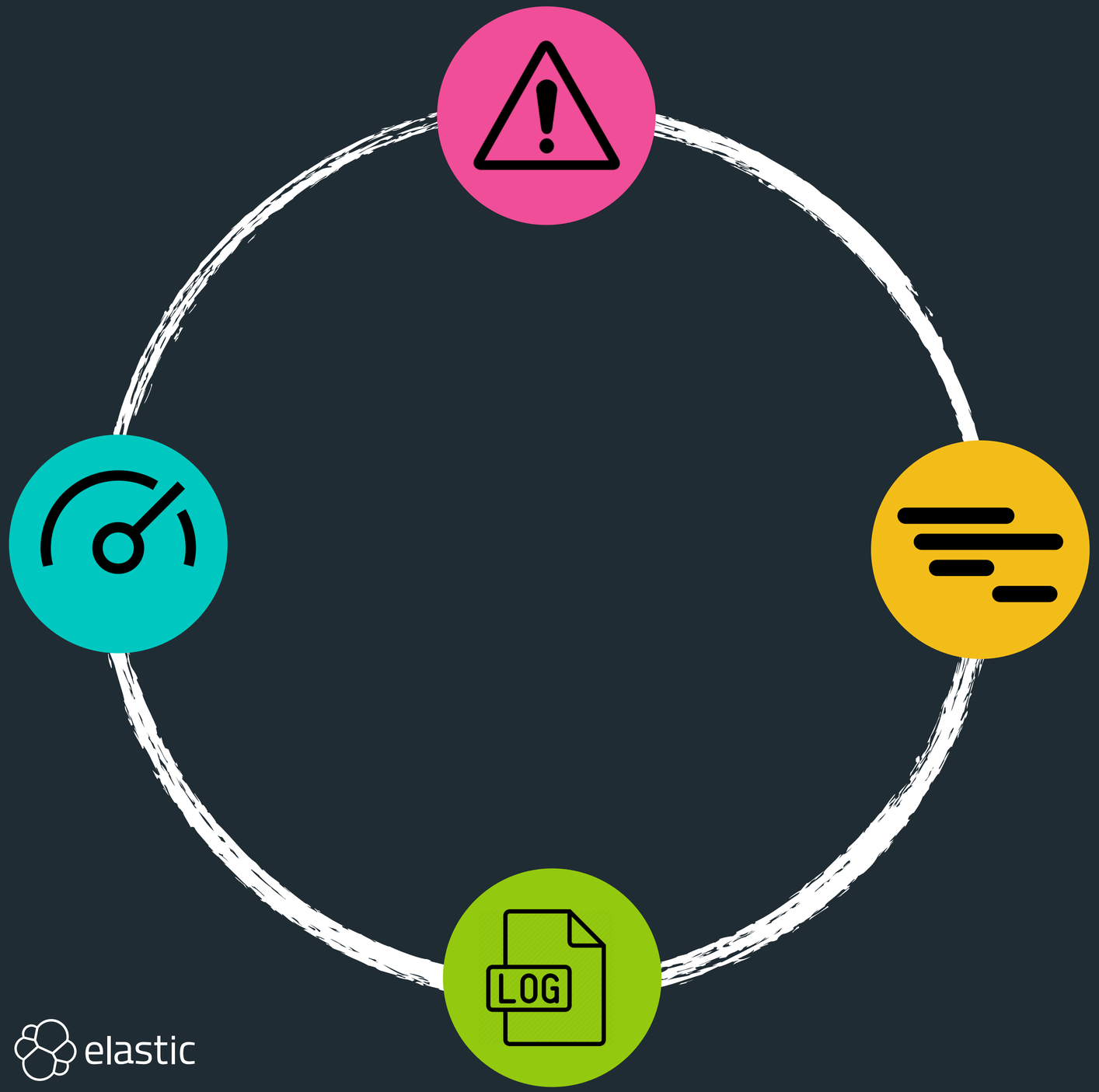
Logs



Metrics



Errors



Traces

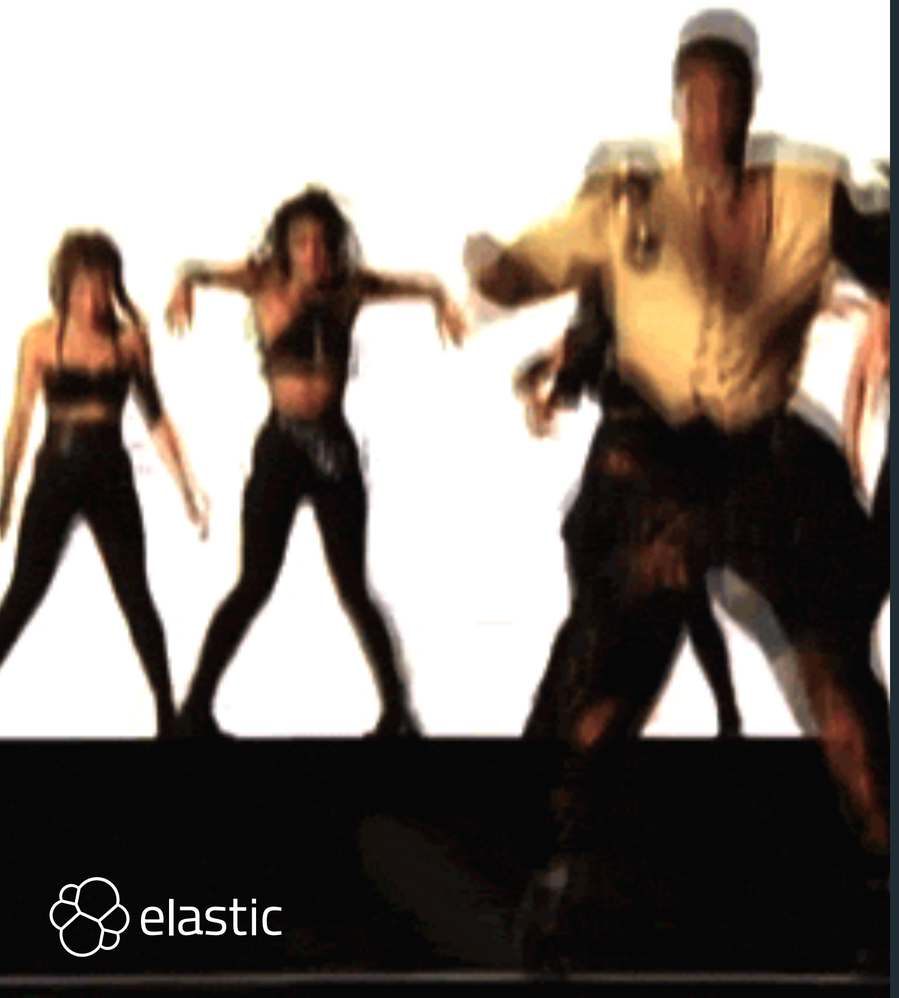
elastic

@wa7son

elastic

@wa7son

Source: MC Hammer - U Can't Touch This



Demo Time!

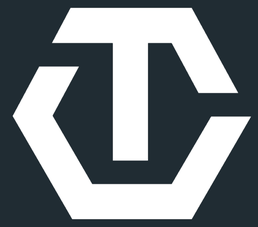
elastic

@wa7son

More information

elastic

@wa7son



OPENTRACING

elastic

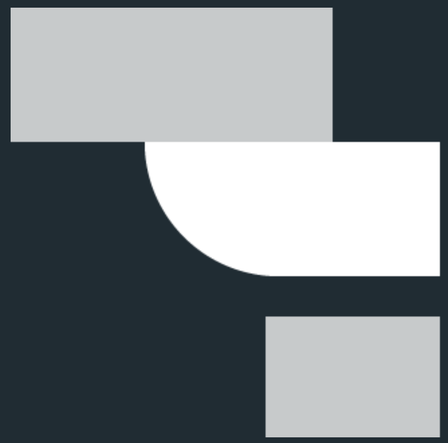
@wa7son

OPENTRACING

opentracing.io

elastic

@wa7son



apm

elastic.co / solutions / apm

@wa7son

DM’s are open ❤️

Getting involved

github.com/elastic/apm-agent-nodejs/pull/691

github.com / elastic / apm


спасибо
@wa7son
github.com/watson
elastic