A Survey of the Open-Source Tracing Ecosystem

KubeCon + CloudNativeCon Europe
May 2, 2018

Ben Sigelman

OpenTracing co-creator, LightStep co-founder and CEO

Talk #goals

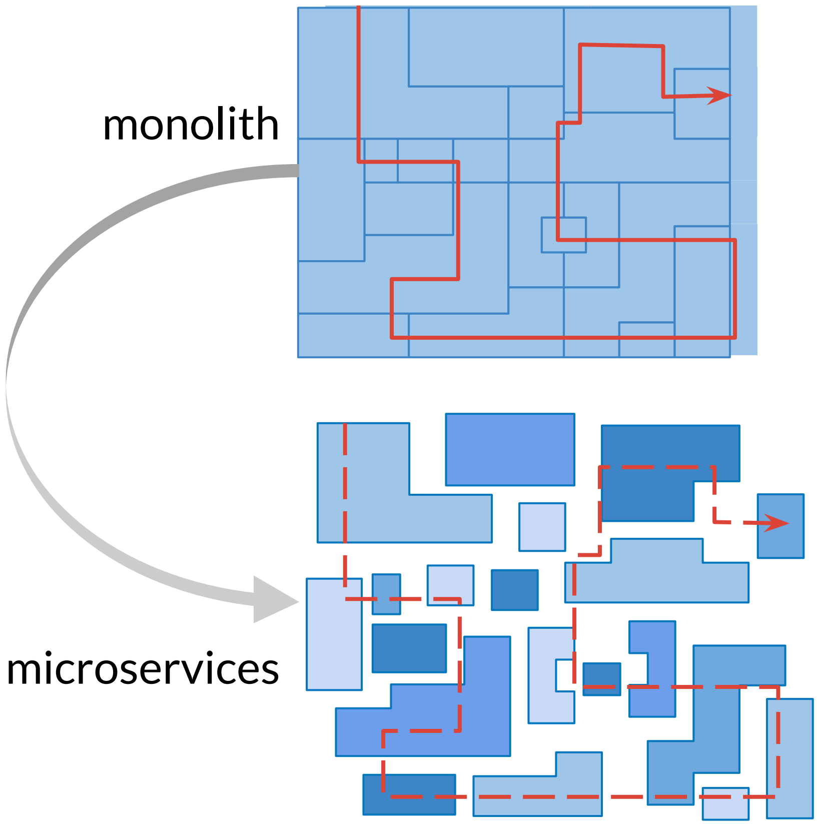
- What's distributed tracing?
- How does it compare with other monitoring tech?
- What are the moving parts?
- Why are there so many tracing projects???
- ... and what's the difference between them?

Part I
Where Distributed Tracing fits into the Monitoring Ecosystem

“All of Monitoring” (more or less)

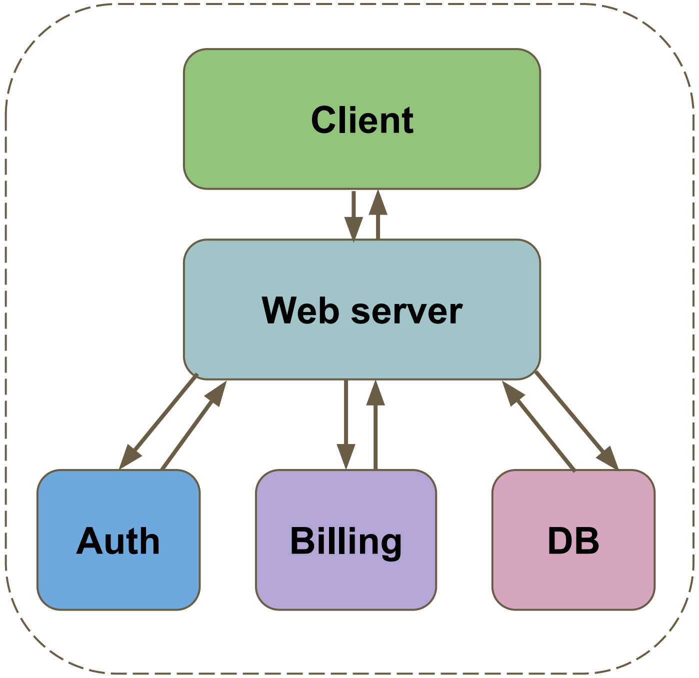
	Timeseries Statistics	**Structured Events**
Measuring Symptoms	p99 latency alerts Error ratio alerts (etc)	Exception Monitoring (Sentry) Counting relatively infrequent things like software releases (etc)
Explaining Symptoms (i.e., Root-Cause Analysis)	Most “big” dashboards Filter + GroupBy on metrics (etc)	Grepping logs Rolling up data in ad hoc ways, a la Kibana or similar *... and Distributed Tracing!*

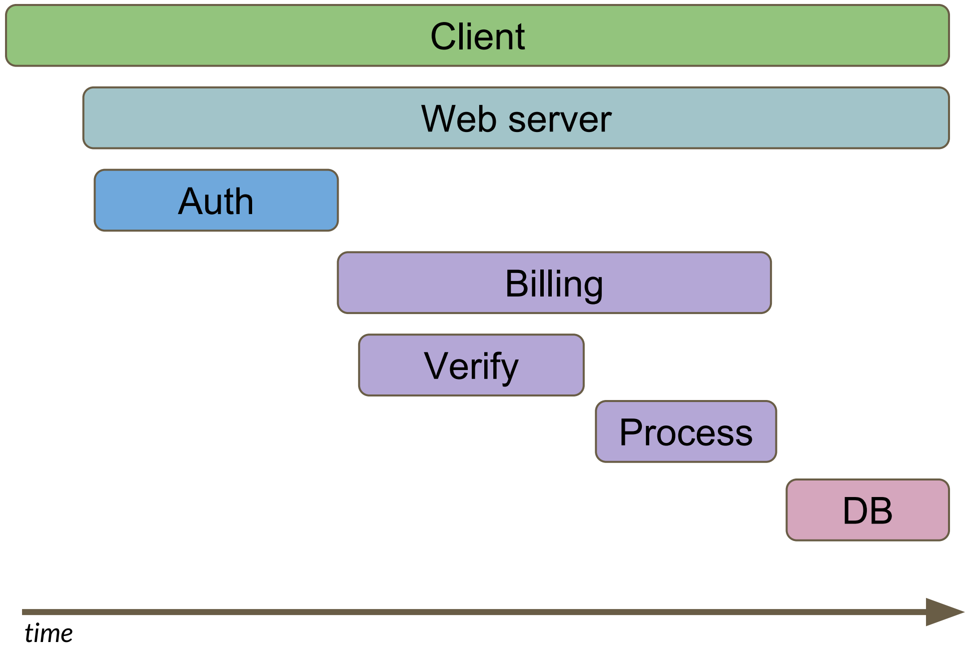
Distributed Traces: Events, Organized



- Every request touches many services
- These touches generate event data
- Must “glue” all of the per-request event data back together
- Once you’re “gluing,” you’re “tracing”

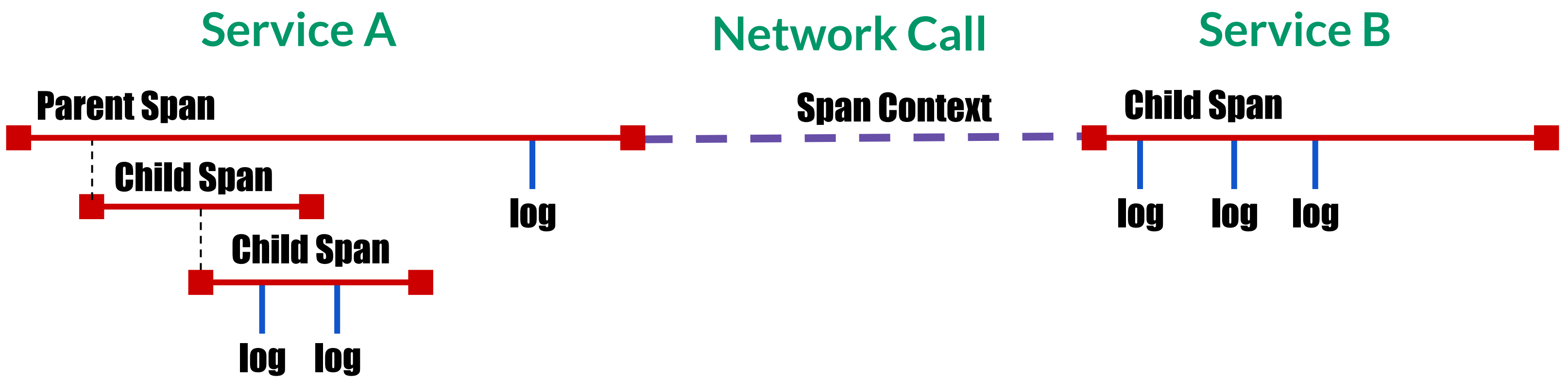
Example Request with Trace Visualization





Time context visualization

Tracing: the Mental Model



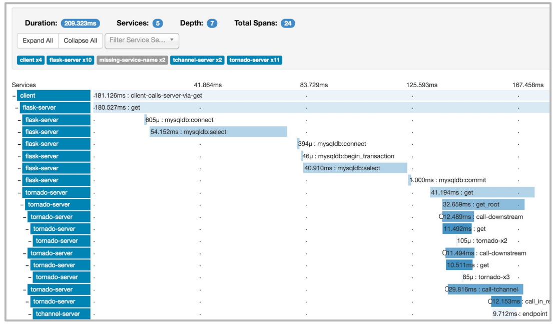
- **Trace:** A recording of a transaction as it moves through a distributed system.
- **Span:** A named, timed operation representing a piece of the workflow. Spans have a Timestamp and a Duration, are annotated with **Tags** and **Logs**.
- **Span Context:** A set of **Trace Identifiers** injected into each request, which the next service will extract and use in order to propagate the trace.

OPENTRACING

Part II

“Tracing,” “Tracing,” “Tracing,” and “Tracing”: One word, four needs

Tracing: It's About *Analyzing Transactions*

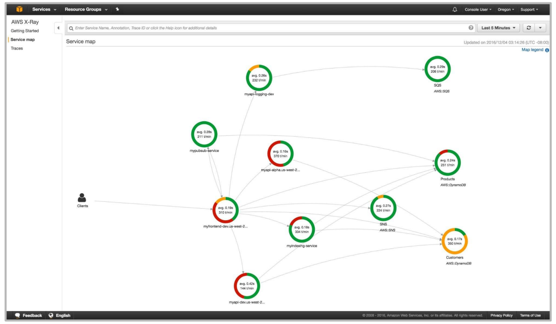


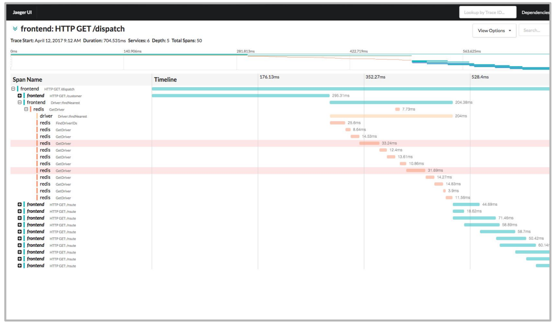
Duration: 209.323ms Services: 5 Depth: 7 Total Spans: 24

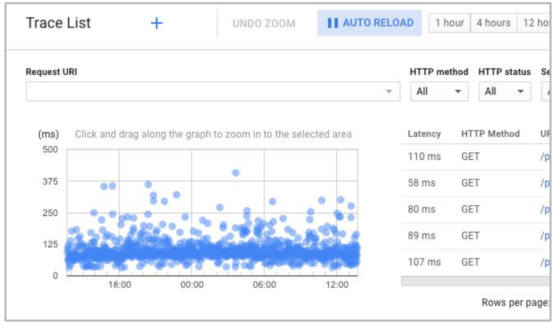
Expand All | Collapse All | Filter Service Se...

client x4 | flask-server x10 | missing-service-name x2 | tchannel-server x2 | tornado-server x11

```
Services            41.864ms        83.729ms        125.593ms        167.458ms
- client            181.126ms : client-calls-server-via-get
 - flask-server     180.527ms : get
  - flask-server    605µ : mysqldb:connect
  - flask-server    54.152ms : mysqldb:select
  - flask-server    394µ : mysqldb:connect
  - flask-server    46µ : mysqldb:begin_transaction
  - flask-server    40.910ms : mysqldb:select
  - flask-server    1.000ms : mysqldb:commit
  - tornado-server  41.194ms : get
   - tornado-server 32.659ms : get_root
    - tornado-server 12.489ms : call-downstream
     - tornado-server 11.492ms : get
      - tornado-server 105µ : tornado-x2
    - tornado-server 11.494ms : call-downstream
     - tornado-server 10.511ms : get
      - tornado-server 85µ : tornado-x3
    - tornado-server 29.816ms : call-tchannel
     - tornado-server 12.153ms : call_in_re
      - tchannel-server 9.712ms : endpoint
```







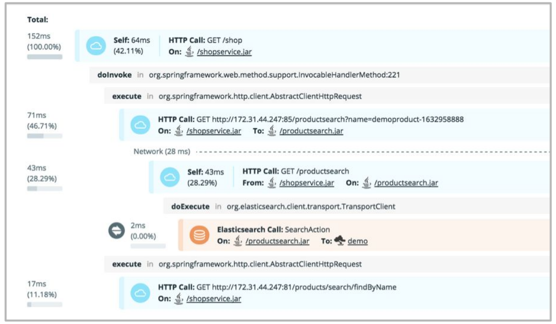
Trace List + UNDO ZOOM AUTO RELOAD 1 hour 4 hours 12 ho

Request URI | HTTP method All | HTTP status All

(ms) Click and drag along the graph to zoom in to the selected area

Latency	HTTP Method	UF
110 ms	GET	/p
58 ms	GET	/p
80 ms	GET	/p
89 ms	GET	/p
107 ms	GET	/p

Rows per page:



Total:

152ms (100.00%) — **Self:** 64ms (42.11%) | **HTTP Call:** GET /shop **On:** /shopservice.jar

doInvoke in org.springframework.web.method.support.InvocableHandlerMethod:221

execute in org.springframework.http.client.AbstractClientHttpRequest

71ms (46.71%) — **HTTP Call:** GET http://172.31.44.247:85/productsearch?name=demoproduct-1632958888 **On:** /shopservice.jar **To:** /productsearch.jar

Network (28 ms)

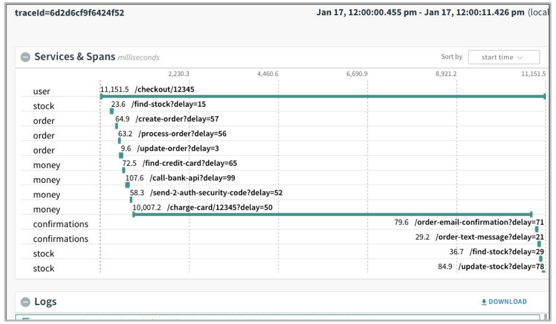
43ms (28.29%) — **Self:** 43ms (28.29%) | **HTTP Call:** GET /productsearch **From:** /shopservice.jar **On:** /productsearch.jar

doExecute in org.elasticsearch.client.transport.TransportClient

2ms (0.00%) — **Elasticsearch Call:** SearchAction **On:** /productsearch.jar **To:** demo

execute in org.springframework.http.client.AbstractClientHttpRequest

17ms (11.18%) — **HTTP Call:** GET http://172.31.44.247:81/products/search/findByName **On:** /shopservice.jar



traceId=6d2d6cf9f6424f52 **Jan 17, 12:00:00.455 pm - Jan 17, 12:00:11.426 pm** (local

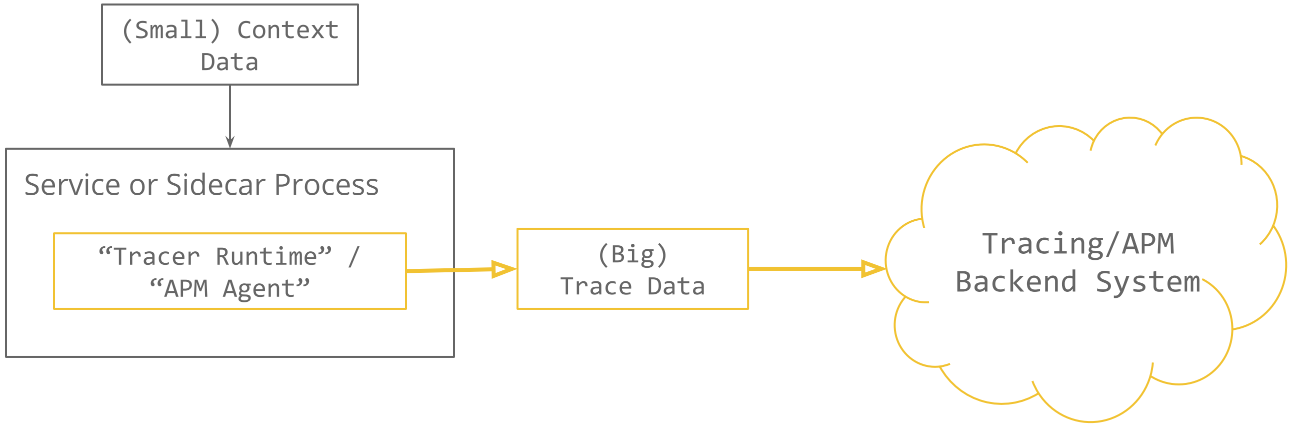
Services & Spans *milliseconds* Sort by start time

```
                2,230.3     4,460.6     6,690.9     8,921.2     11,151.5
user            11,151.5  /checkout/12345
stock           23.6  /find-stock?delay=15
order           64.9  /create-order?delay=57
order           63.2  /process-order?delay=56
order           9.6  /update-order?delay=3
money           72.5  /find-credit-card?delay=65
money           107.6  /call-bank-api?delay=99
money           58.3  /send-2-auth-security-code?delay=52
money           10,007.2  /charge-card/12345?delay=50
confirmations   79.6  /order-email-confirmation?delay=71
confirmations   29.2  /order-text-message?delay=21
stock           36.7  /find-stock?delay=29
stock           84.9  /update-stock?delay=78
```

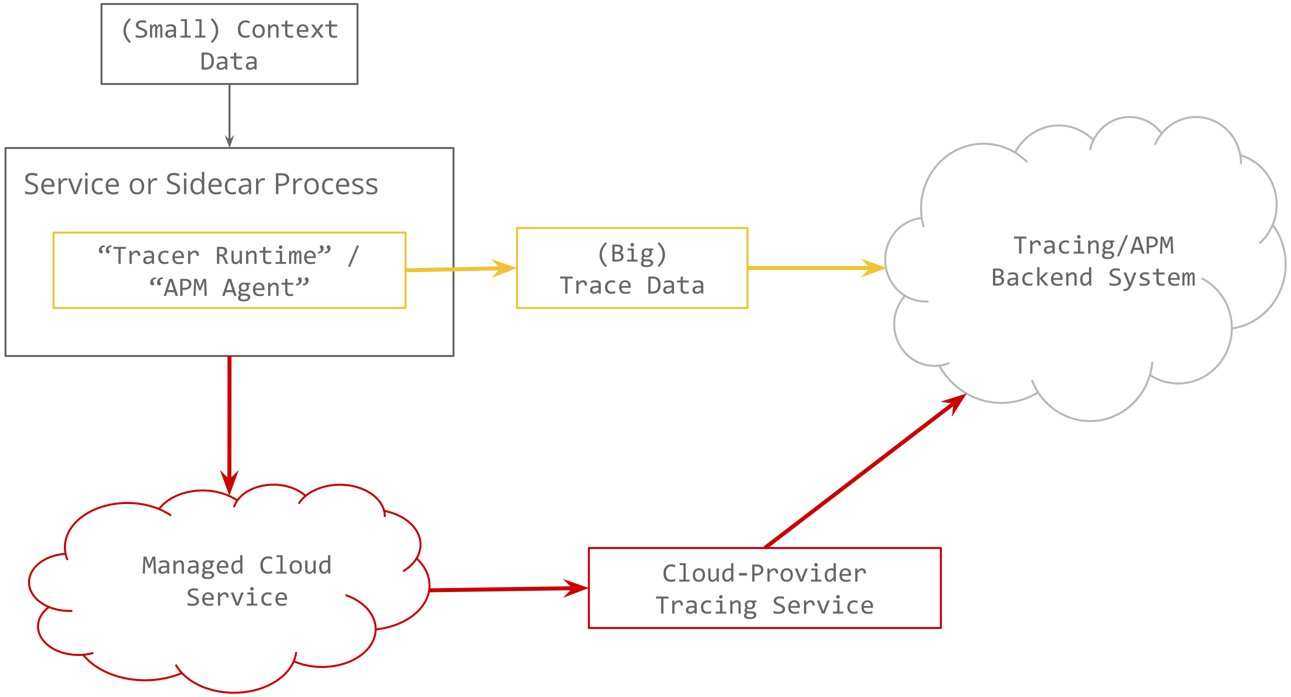
Logs DOWNLOAD

OPENTRACING

Tracing: It's About *Recording Transactions*

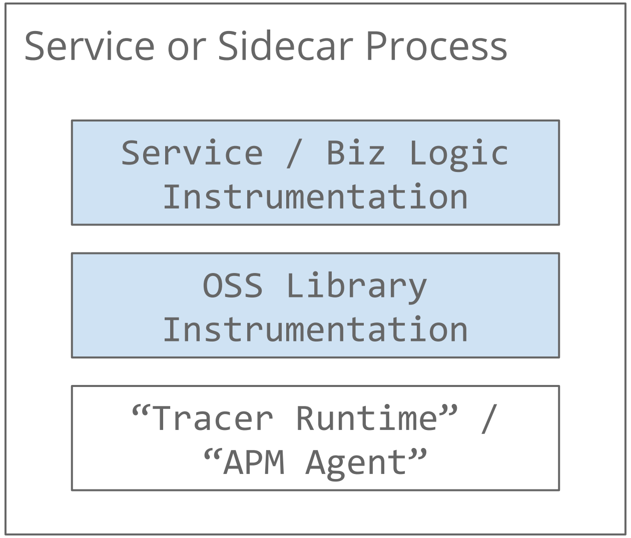

(Small) Context Data
Service or Sidecar Process
"Tracer Runtime" / "APM Agent"
(Big) Trace Data
Tracing/APM Backend System

Tracing: It’s About *Federating Transactions*


(Small) Context Data
Service or Sidecar Process
“Tracer Runtime” / “APM Agent”
(Big) Trace Data
Tracing/APM Backend System
Managed Cloud Service
Cloud-Provider Tracing Service

OPENTRACING

Tracing: It's About *Describing Transactions*


Service or Sidecar Process
Service / Biz Logic Instrumentation
OSS Library Instrumentation
"Tracer Runtime" / "APM Agent"

Tracing is (presently) all four

1. Analyzing Transactions
2. Recording Transactions
3. Federating Transactions
4. Describing Transactions

All have their place; hopefully they stay decoupled.

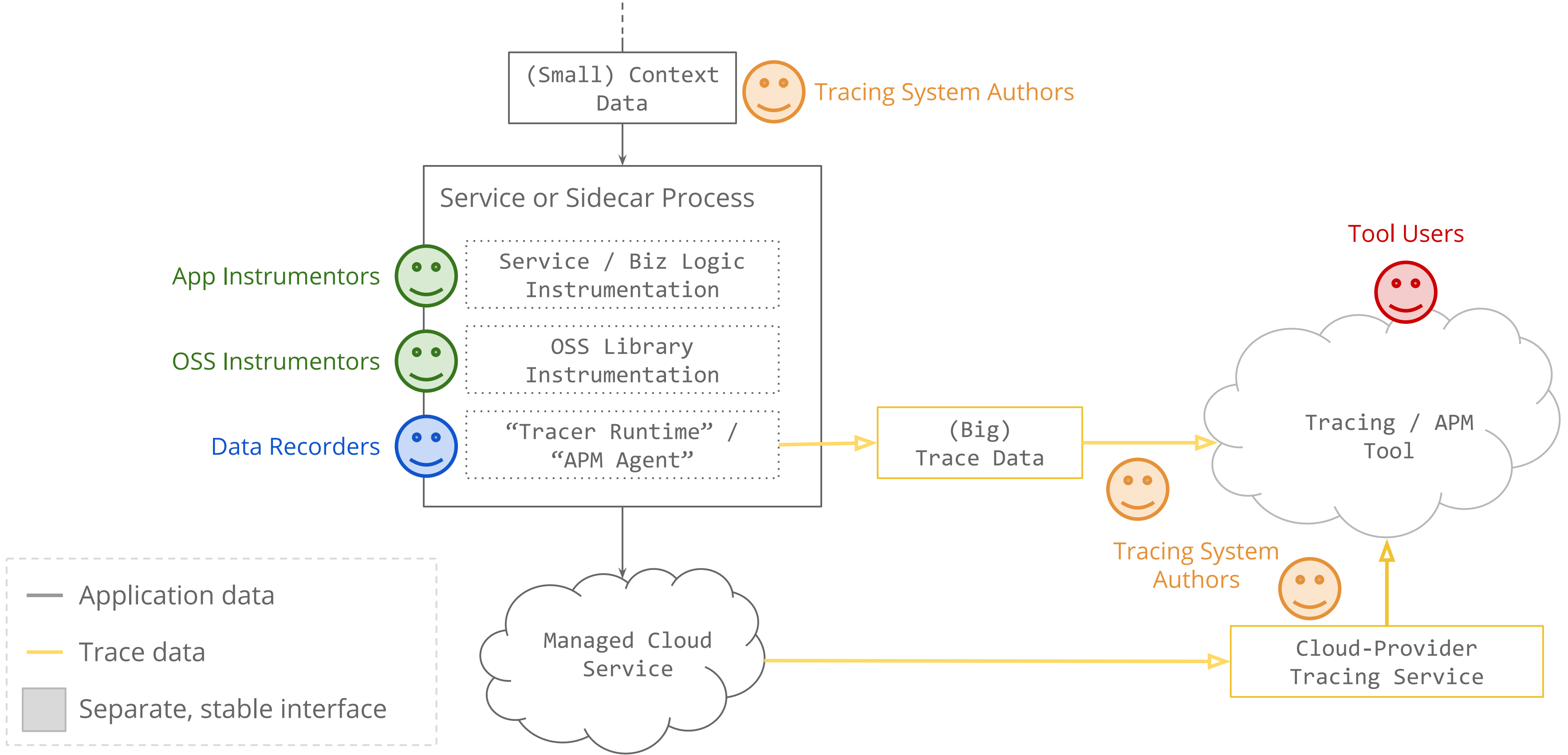
Part III

Noteworthy tracing projects

and

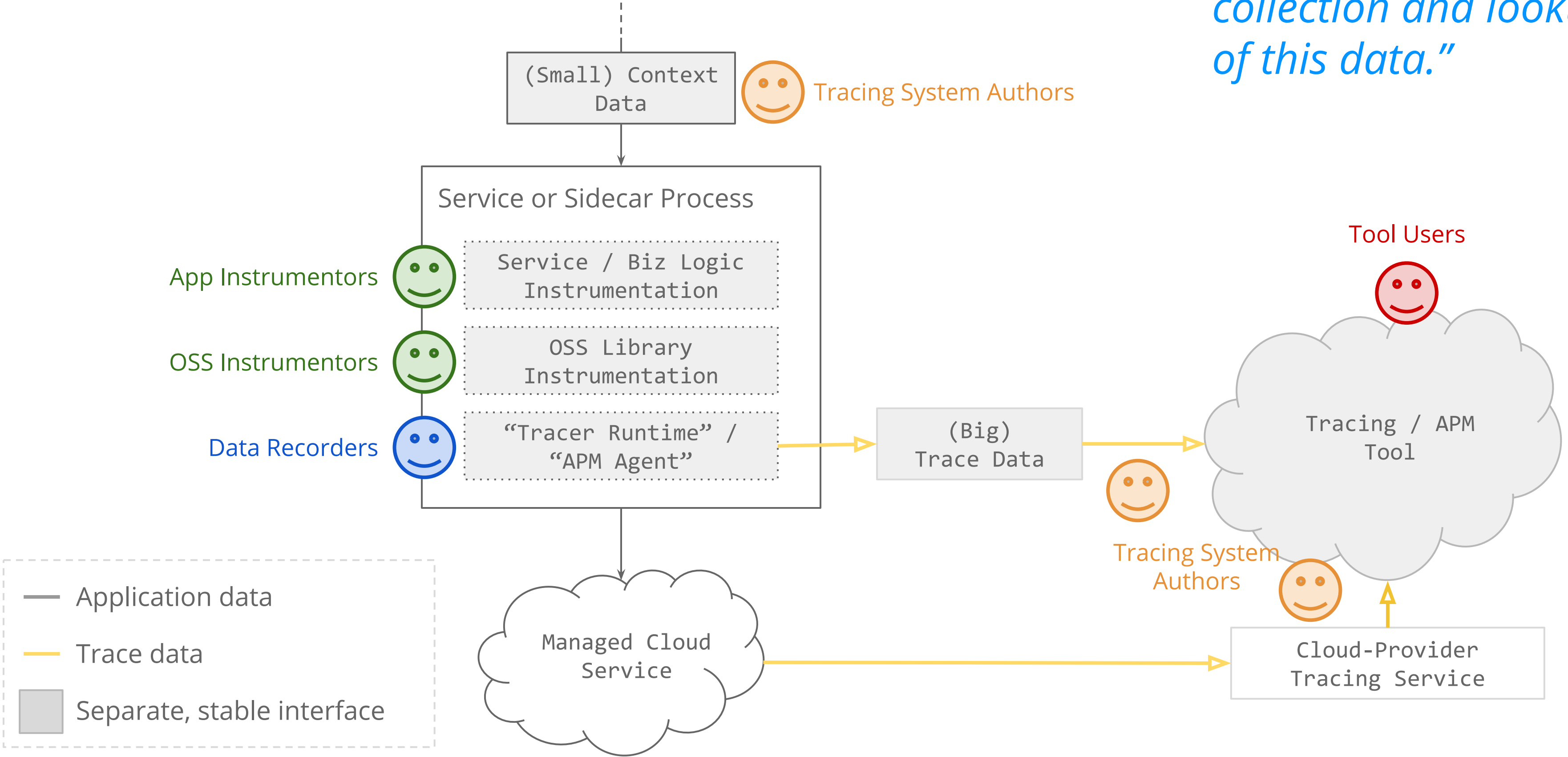
how they fit together

Composable Parts in a Tracing Deployment



Zipkin surface area

"Zipkin is a distributed tracing system. ... It manages both the collection and lookup of this data."

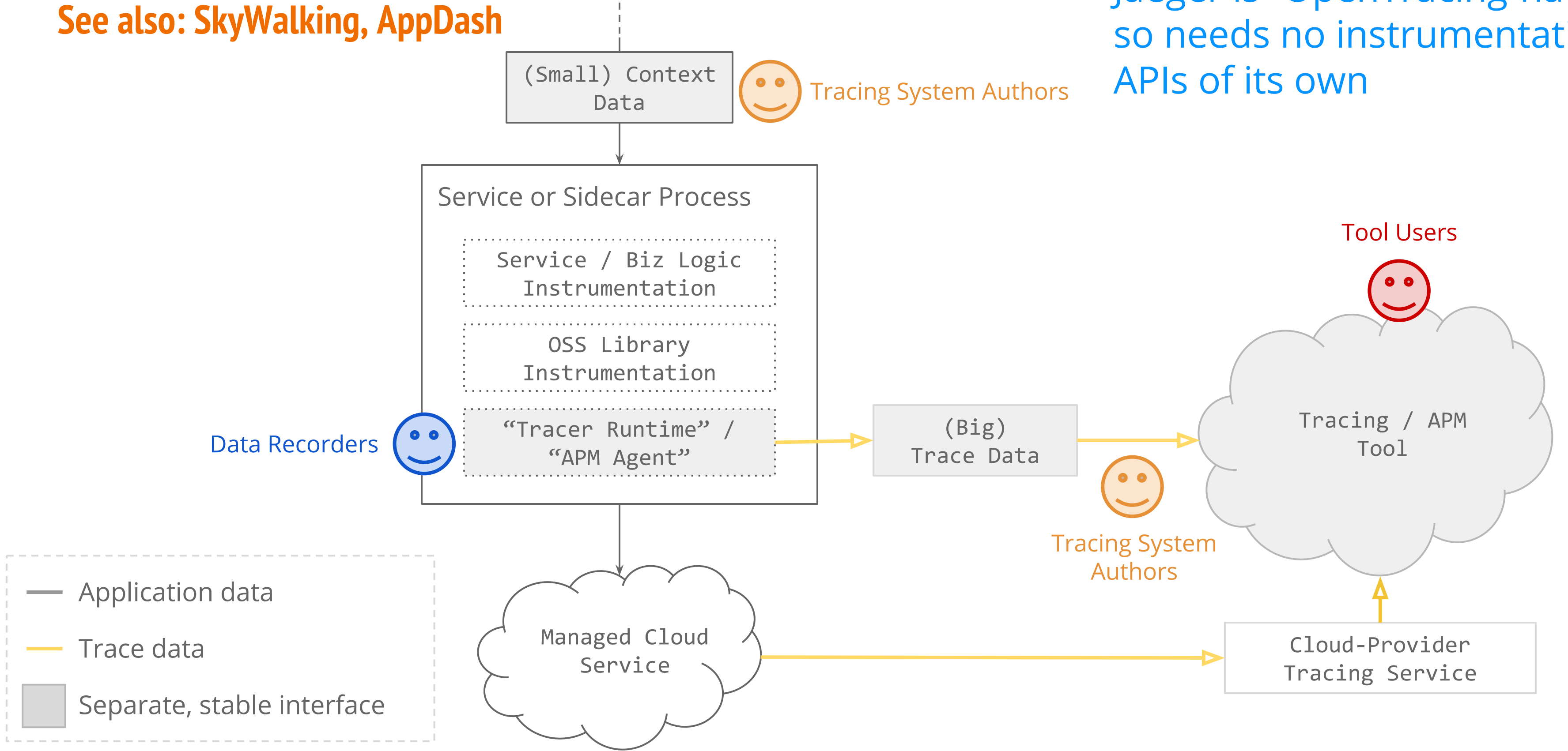


Jaeger surface area

See also: SkyWalking, AppDash

"Jaeger: open source, end-to-end distributed tracing"

Jaeger is "OpenTracing-native" so needs no instrumentation APIs of its own

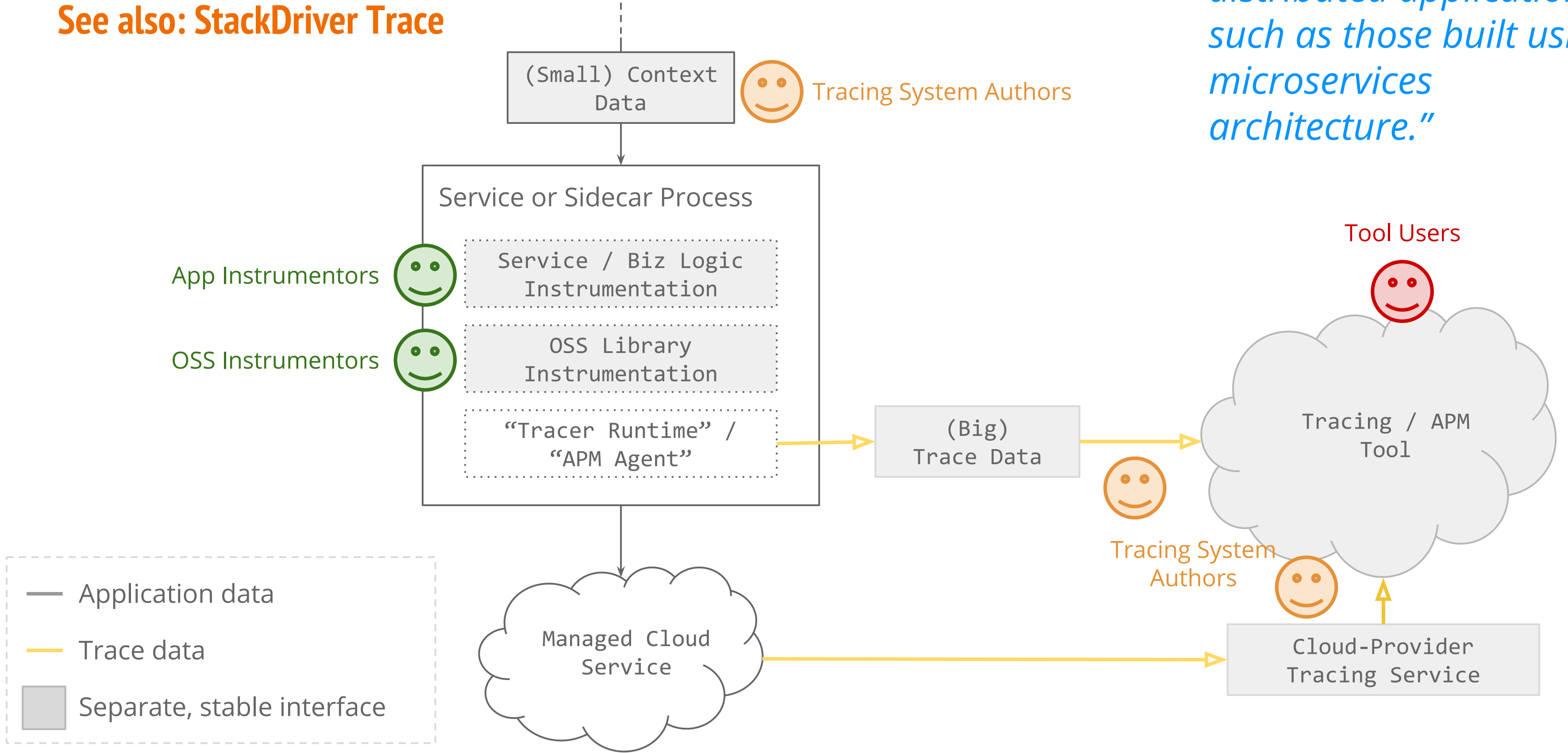


(Small) Context Data
Tracing System Authors
Service or Sidecar Process
Service / Biz Logic Instrumentation
OSS Library Instrumentation
"Tracer Runtime" / "APM Agent"
Data Recorders
(Big) Trace Data
Tracing System Authors
Tool Users
Tracing / APM Tool
Managed Cloud Service
Cloud-Provider Tracing Service
Application data
Trace data
Separate, stable interface

AWS X-Ray surface area

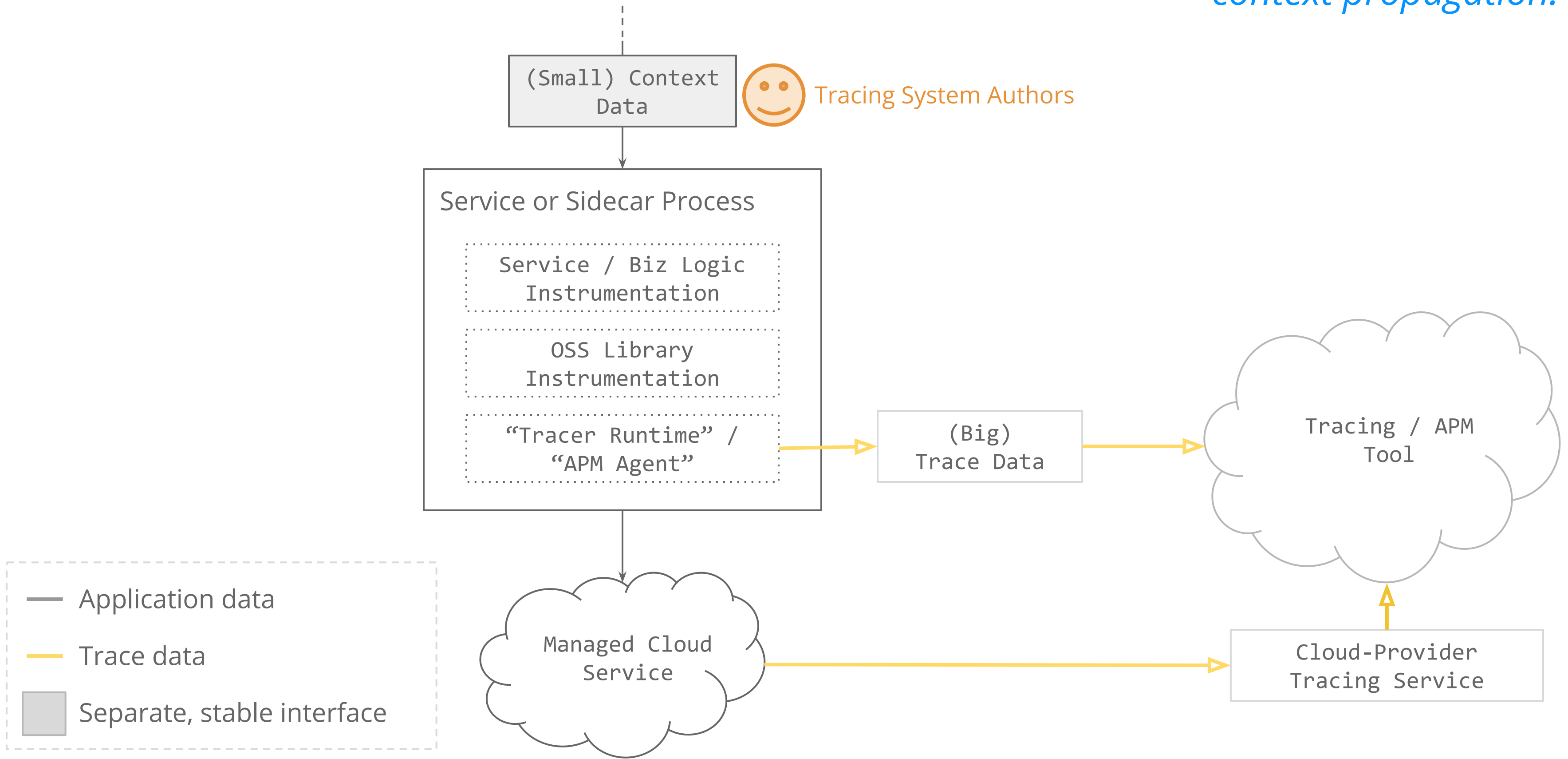
See also: StackDriver Trace

"AWS X-Ray helps developers analyze and debug production, distributed applications, such as those built using a microservices architecture."



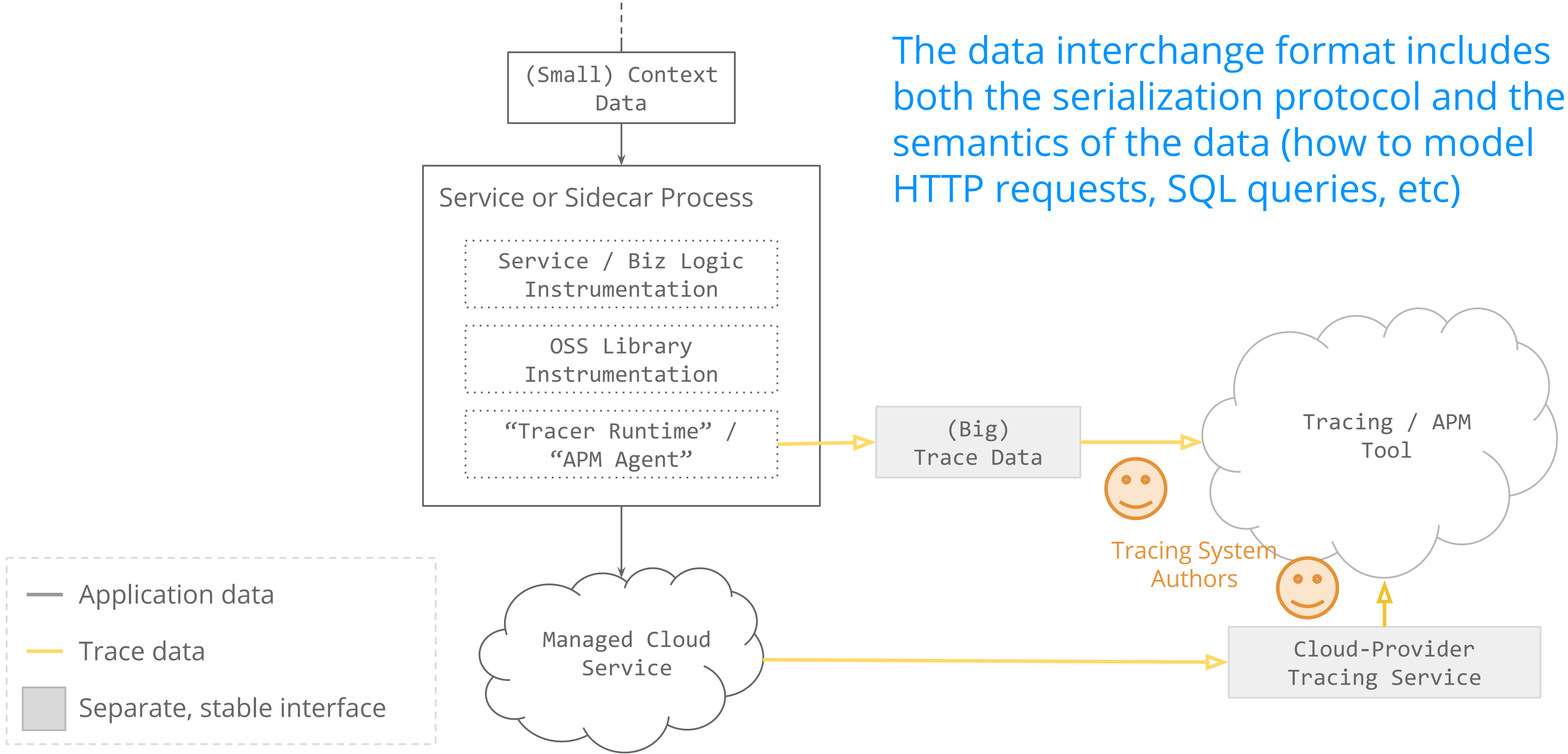
w3c "Trace-Context" surface area

"The mission of this group is to define the standard for distributed trace context propagation."


(Small) Context Data
Tracing System Authors
Service or Sidecar Process
Service / Biz Logic Instrumentation
OSS Library Instrumentation
"Tracer Runtime" / "APM Agent"
(Big) Trace Data
Tracing / APM Tool
Managed Cloud Service
Cloud-Provider Tracing Service
Application data
Trace data
Separate, stable interface

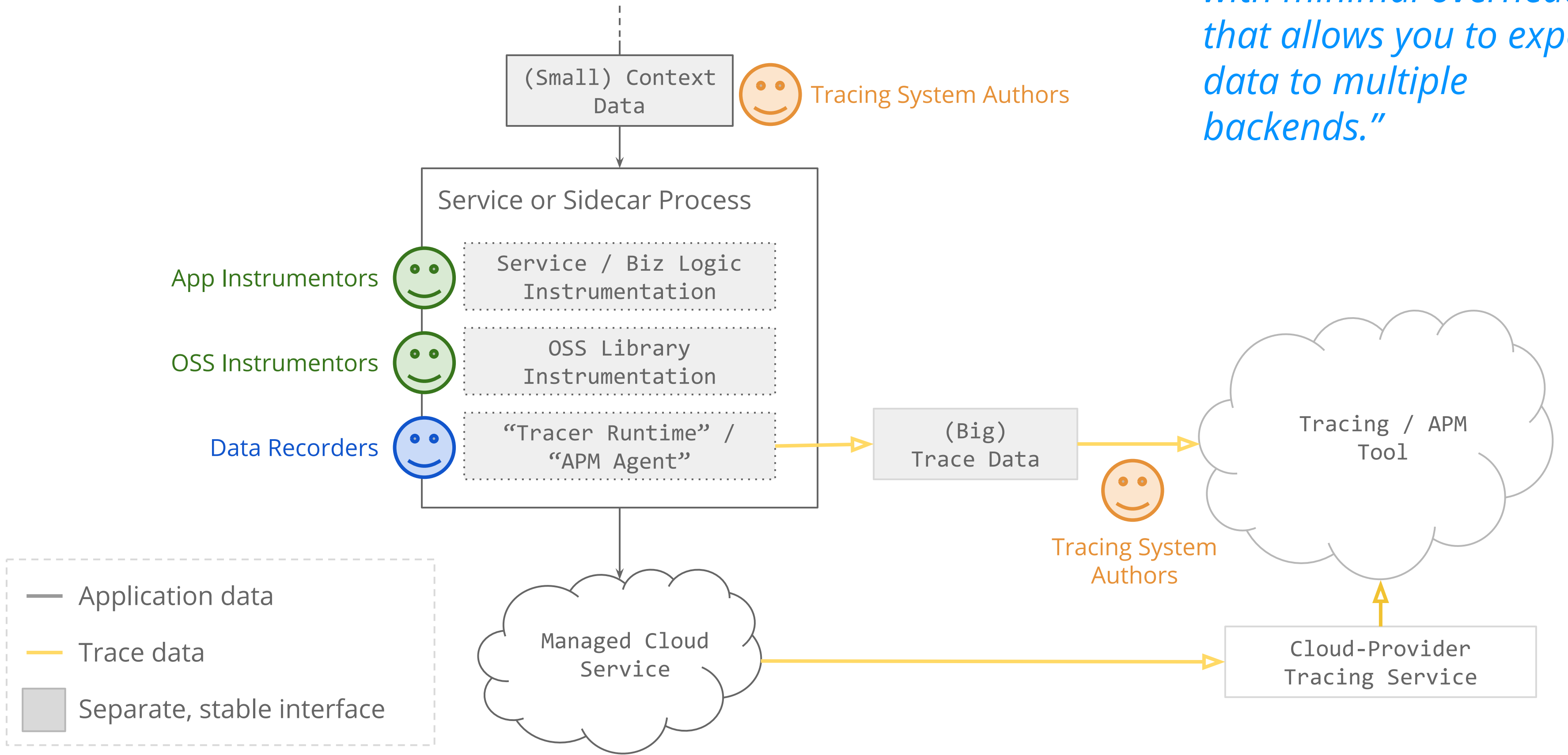
w3c "data interchange format" surface area

The data interchange format includes both the serialization protocol and the semantics of the data (how to model HTTP requests, SQL queries, etc)


(Small) Context Data
Service or Sidecar Process
Service / Biz Logic Instrumentation
OSS Library Instrumentation
"Tracer Runtime" / "APM Agent"
(Big) Trace Data
Tracing / APM Tool
Tracing System Authors
Managed Cloud Service
Cloud-Provider Tracing Service
Application data
Trace data
Separate, stable interface

OpenCensus surface area

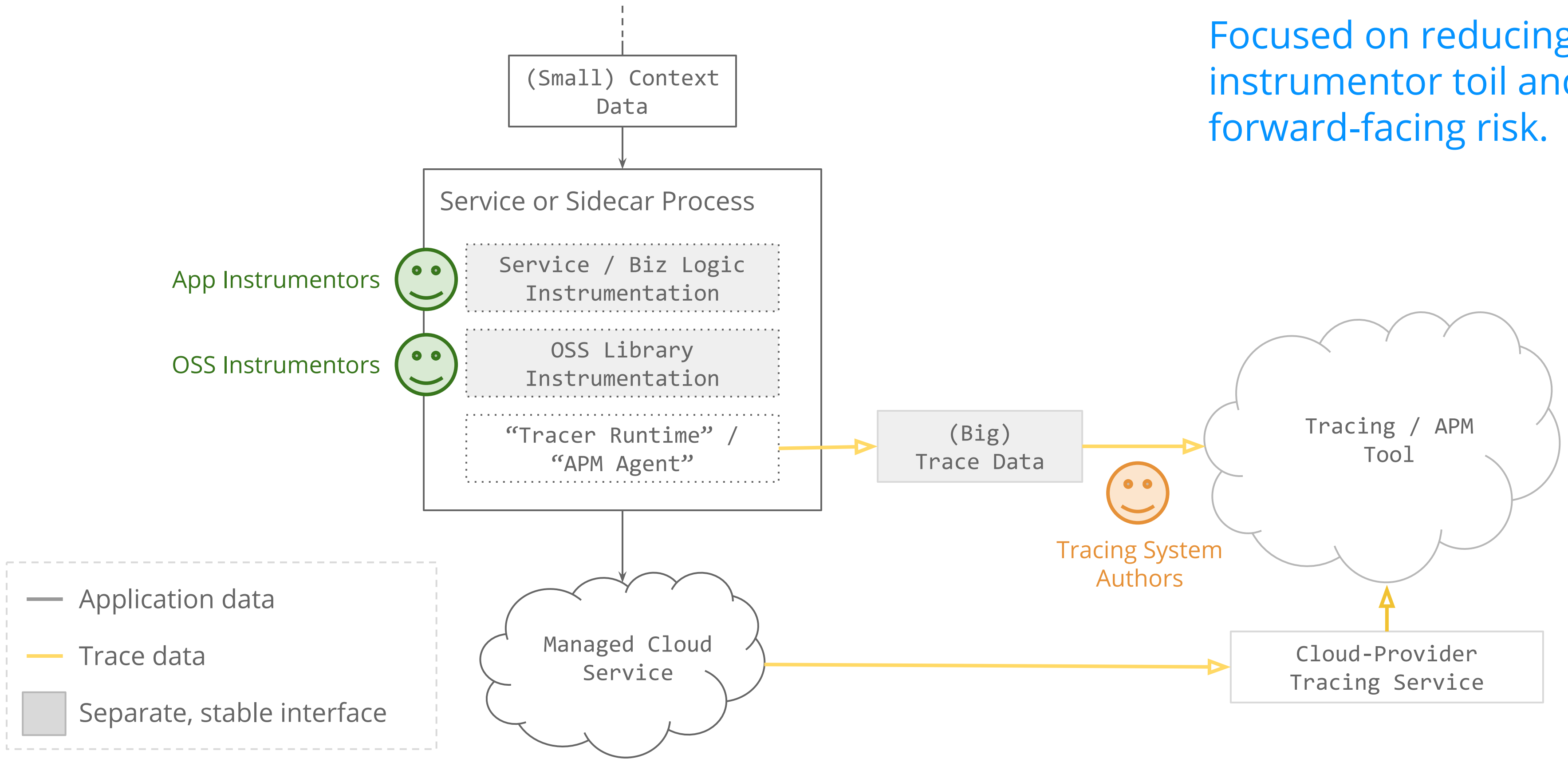
"A single distribution of libraries for metrics and distributed tracing with minimal overhead that allows you to export data to multiple backends."



OpenTracing surface area

"Vendor-neutral APIs and instrumentation for distributed tracing"

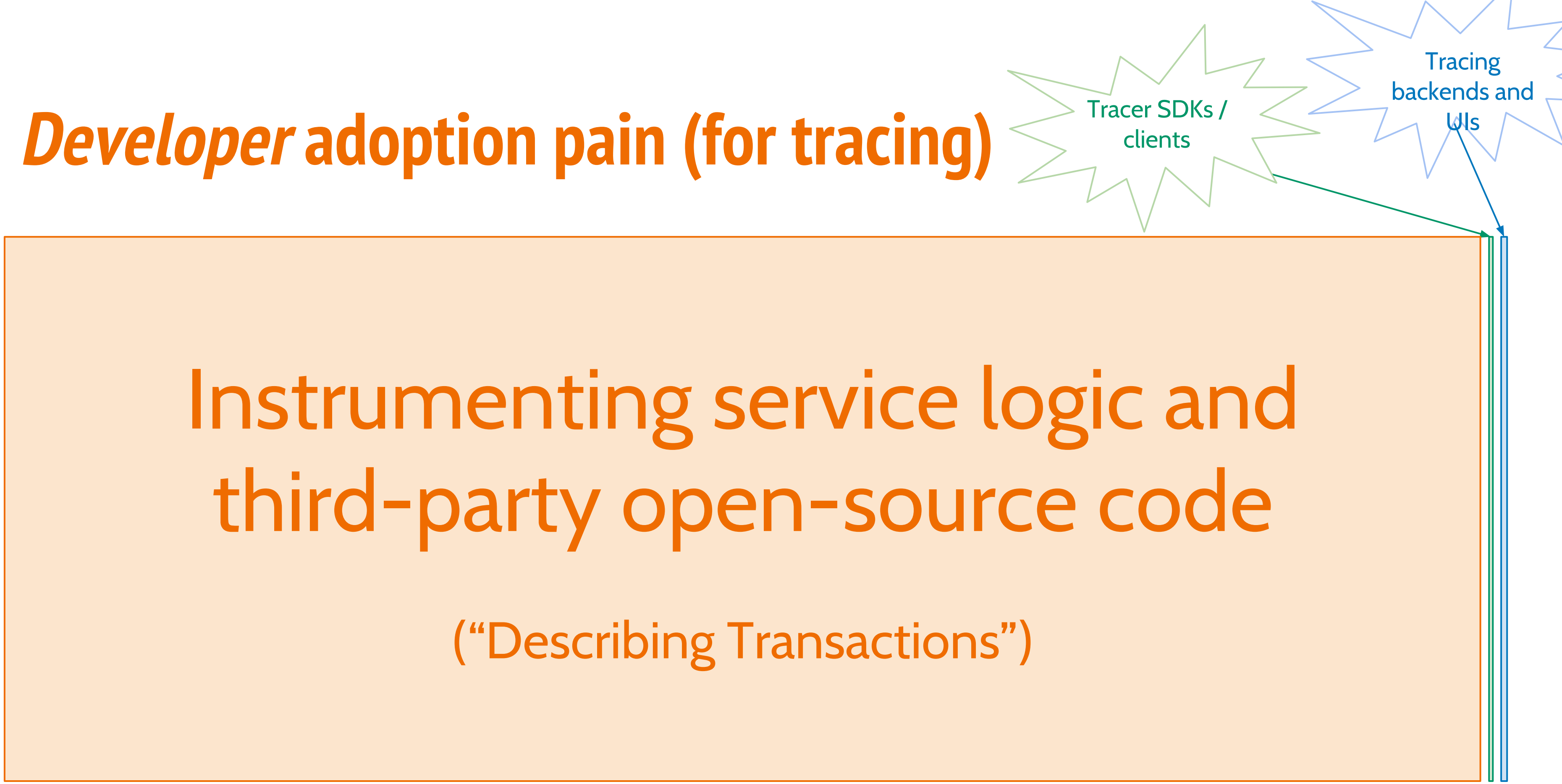
Focused on reducing instrumentor toil and forward-facing risk.


(Small) Context Data
Service or Sidecar Process
App Instrumentors
Service / Biz Logic Instrumentation
OSS Instrumentors
OSS Library Instrumentation
"Tracer Runtime" / "APM Agent"
(Big) Trace Data
Tracing / APM Tool
Tracing System Authors
Managed Cloud Service
Cloud-Provider Tracing Service
Application data
Trace data
Separate, stable interface

Part IV

The Case for Narrow Interfaces

Developer adoption pain (for tracing)


Tracer SDKs / clients
Tracing backends and UIs
Instrumenting service logic and third-party open-source code
("Describing Transactions")

One example: OpenTracing

Instrumentation scope:

- 9 languages
- ≥ 100 instrumented packages

“Tracer” diversity:

- Many Tracer implementations: OSS and commercial
- Some are not “just” tracing a la Dapper (debuggers, metrics, logging)

A small formal API means:

- A single, *decoupled* value proposition
- Smaller deps at compile-time
- Fewer surprises downstream

Another example: w3c TraceContext

Many players: 25+

Many other (tempting) adjacent problems to solve

- The "Span data format" for vendor-to-vendor interop
- OpenTracing-style APIs for propagation among other things
- More complex sampling strategies, as well as baggage, etc

The narrow scope:

- Keeps the discussion focused
- Encourages a "best-of-breed" approach
- Reduces coupling (e.g., between instrumentation and headers)

Summing up...

- Tracing is **table stakes** now
- Tracing must involve **many constituents**
- **Tracing, Tracing, Tracing, & Tracing**: not the same thing :)
- Each project exists for good reasons: **understand them**
- Not a zero-sum game!

Also: I’m rarely in Europe and want to meet you!

↓↓↓

twitter: **@el_bhs**
email: **bhs@lightstep.com**