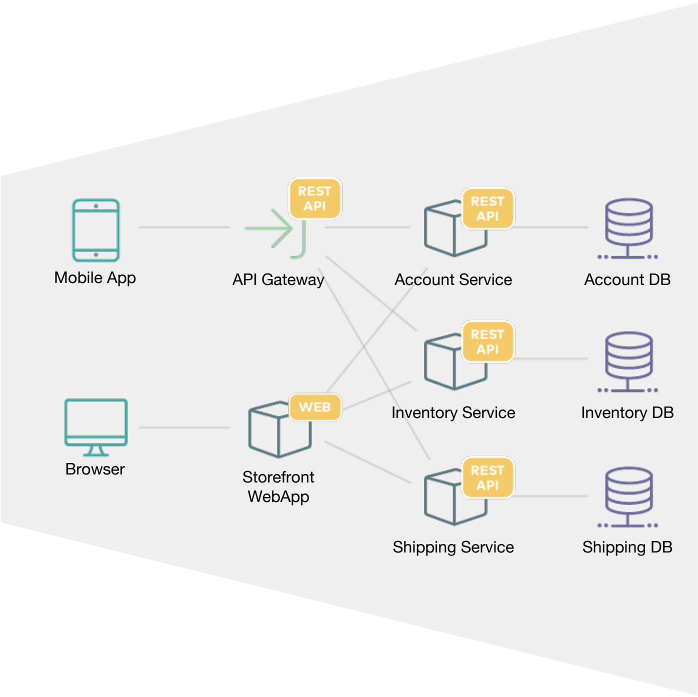
SCALYR®

Connected Systems and Distributed Request Tracing

PRESENTED BY: Dave McAllister

The cloud makes things easier, which makes things harder.

Breaking News: The World isn't getting Simpler



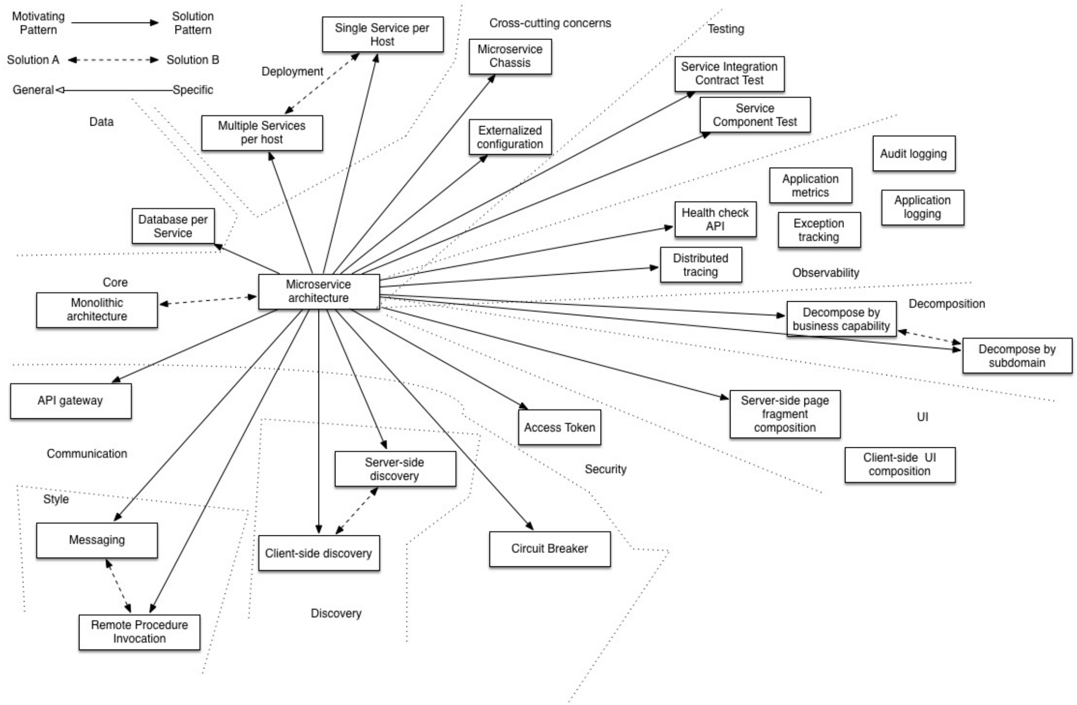


Image from https://microservices.io/patterns/microservices.html

Connected Systems are Complex Systems

“A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable.”

– Leslie Lamport, 1987

So what is Observability

Observability:

the measure of how well **internal states** of a system can be **inferred** from knowledge of its **external outputs.**

or sometimes: **basically monitoring, on Chuck Norris setting**

or: knowing the unknowable while questioning the known (?)

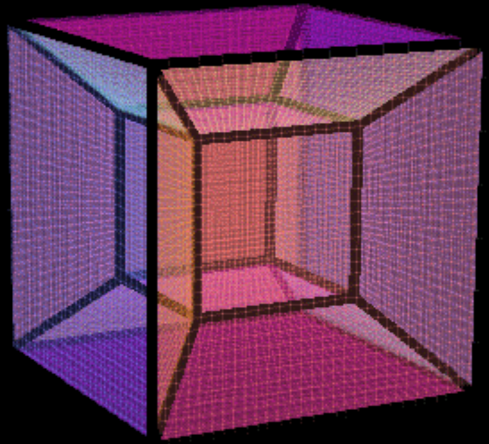
> “You see, but you do not observe.”
>
> Sir Arthur Conan Doyle
> *A Scandal in Bohemia*

Observability is not one dimensional

- Recall “Internal States Inferred from External Outputs”
  - Observability is a property of the system. Not a tool.

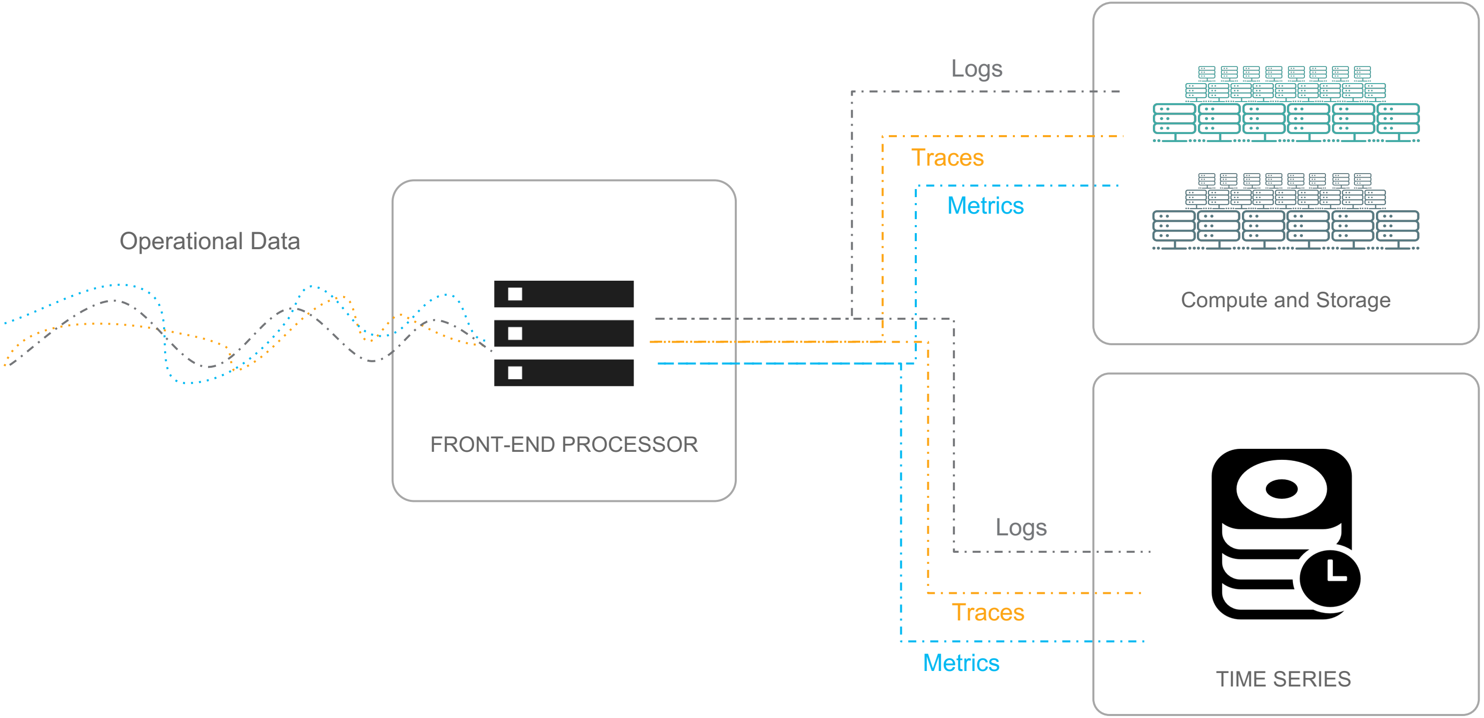
- Should consist of logs, monitoring, events/tracing
- Should include elements of metrics and time
- Should cross boundaries
  - Apps
  - Services
  - Disciplines

Anything that slows you down is bad



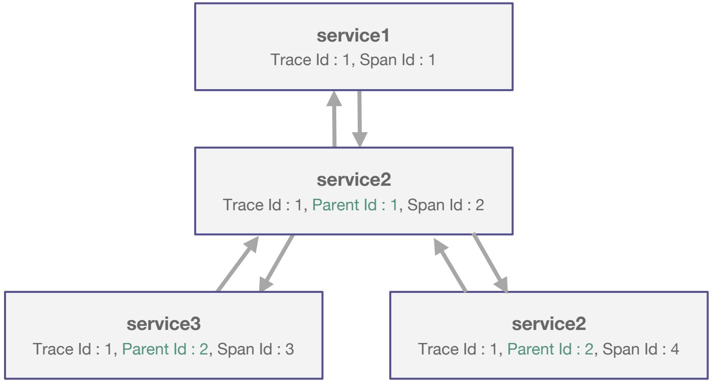
Lions and Tigers and Bears, Oh My (Or Logs, Dashboards and Tracing)

Observability is a signal to noise problem



Distributed Request Tracing

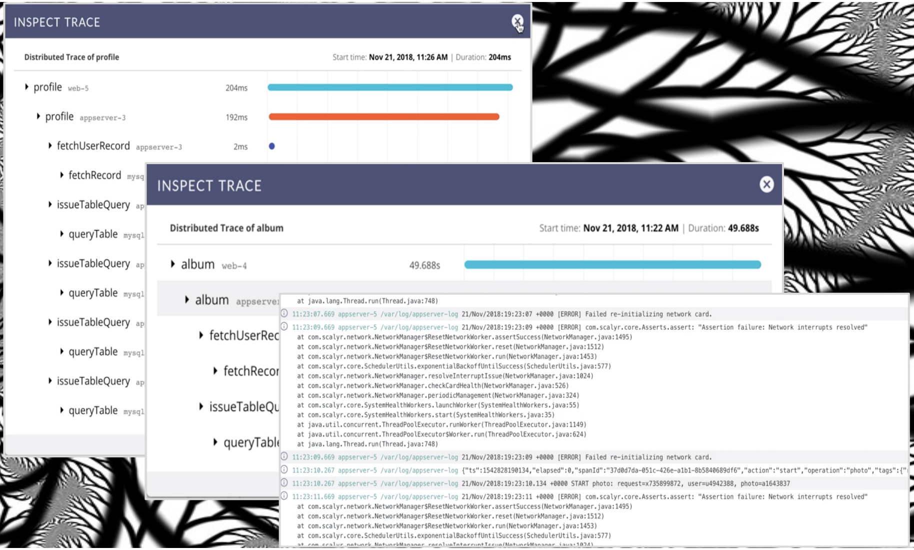
BFF: Tracing and Logs


service1
Trace Id : 1, Span Id : 1
service2
Trace Id : 1, Parent Id : 1, Span Id : 2
service3
Trace Id : 1, Parent Id : 2, Span Id : 3
service2
Trace Id : 1, Parent Id : 2, Span Id : 4

Linking tracing and its high-volume, high-cardinality data to log search
and metrics extracts even more value

Distributed Request Tracing brings events into causal order

- When was the event? How long did it take?
- How do I know it was slow?
- Why did it take so long?
- Which microservice was responsible?



```
INSPECT TRACE

Distributed Trace of profile                Start time: Nov 21, 2018, 11:26 AM | Duration: 204ms

▸ profile web-5                        204ms
  ▸ profile appserver-3                192ms
    ▸ fetchUserRecord appserver-3        2ms
      ▸ fetchRecord mysq
    ▸ issueTableQuery ap
      ▸ queryTable mysql
    ▸ issueTableQuery ap
      ▸ queryTable mysql
    ▸ issueTableQuery ap
      ▸ queryTable mysql
    ▸ issueTableQuery ap
      ▸ queryTable mysql
```

```
INSPECT TRACE

Distributed Trace of album                  Start time: Nov 21, 2018, 11:22 AM | Duration: 49.688s

▸ album web-4                          49.688s
  ▸ album appserve
    ▸ fetchUserRec
      ▸ fetchRecor
    ▸ issueTableQu
      ▸ queryTabl
```

```
    at java.lang.Thread.run(Thread.java:748)
ⓘ 11:23:07.669 appserver-5 /var/log/appserver-log 21/Nov/2018:19:23:07 +0000 [ERROR] Failed re-initializing network card.
ⓘ 11:23:09.669 appserver-5 /var/log/appserver-log 21/Nov/2018:19:23:09 +0000 [ERROR] com.scalyr.core.Asserts.assert: "Assertion failure: Network interrupts resolved"
    at com.scalyr.network.NetworkManager$ResetNetworkWorker.assertSuccess(NetworkManager.java:1495)
    at com.scalyr.network.NetworkManager$ResetNetworkWorker.reset(NetworkManager.java:1512)
    at com.scalyr.network.NetworkManager$ResetNetworkWorker.run(NetworkManager.java:1453)
    at com.scalyr.core.SchedulerUtils.exponentialBackoffUntilSuccess(SchedulerUtils.java:577)
    at com.scalyr.network.NetworkManager.resolveInterruptIssue(NetworkManager.java:1024)
    at com.scalyr.network.NetworkManager.checkCardHealth(NetworkManager.java:526)
    at com.scalyr.network.NetworkManager.periodicManagement(NetworkManager.java:324)
    at com.scalyr.core.SystemHealthWorkers.launchWorker(SystemHealthWorkers.java:55)
    at com.scalyr.core.SystemHealthWorkers.start(SystemHealthWorkers.java:35)
    at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149)
    at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)
    at java.lang.Thread.run(Thread.java:748)
ⓘ 11:23:09.669 appserver-5 /var/log/appserver-log 21/Nov/2018:19:23:09 +0000 [ERROR] Failed re-initializing network card.
ⓘ 11:23:10.267 appserver-5 /var/log/appserver-log {"ts":1542828190134,"elapsed":0,"spanId":"37d0d7da-051c-426e-a1b1-8b5840689df6","action":"start","operation":"photo","tags":{"
ⓘ 11:23:10.267 appserver-5 /var/log/appserver-log 21/Nov/2018:19:23:10.134 +0000 START photo: request=x735899872, user=u4942388, photo=a1643837
ⓘ 11:23:11.669 appserver-5 /var/log/appserver-log 21/Nov/2018:19:23:11 +0000 [ERROR] com.scalyr.core.Asserts.assert: "Assertion failure: Network interrupts resolved"
    at com.scalyr.network.NetworkManager$ResetNetworkWorker.assertSuccess(NetworkManager.java:1495)
    at com.scalyr.network.NetworkManager$ResetNetworkWorker.reset(NetworkManager.java:1512)
    at com.scalyr.network.NetworkManager$ResetNetworkWorker.run(NetworkManager.java:1453)
    at com.scalyr.core.SchedulerUtils.exponentialBackoffUntilSuccess(SchedulerUtils.java:577)
    at com.scalyr.network.NetworkManager.resolveInterruptIssue(NetworkManager.java:1024)
```

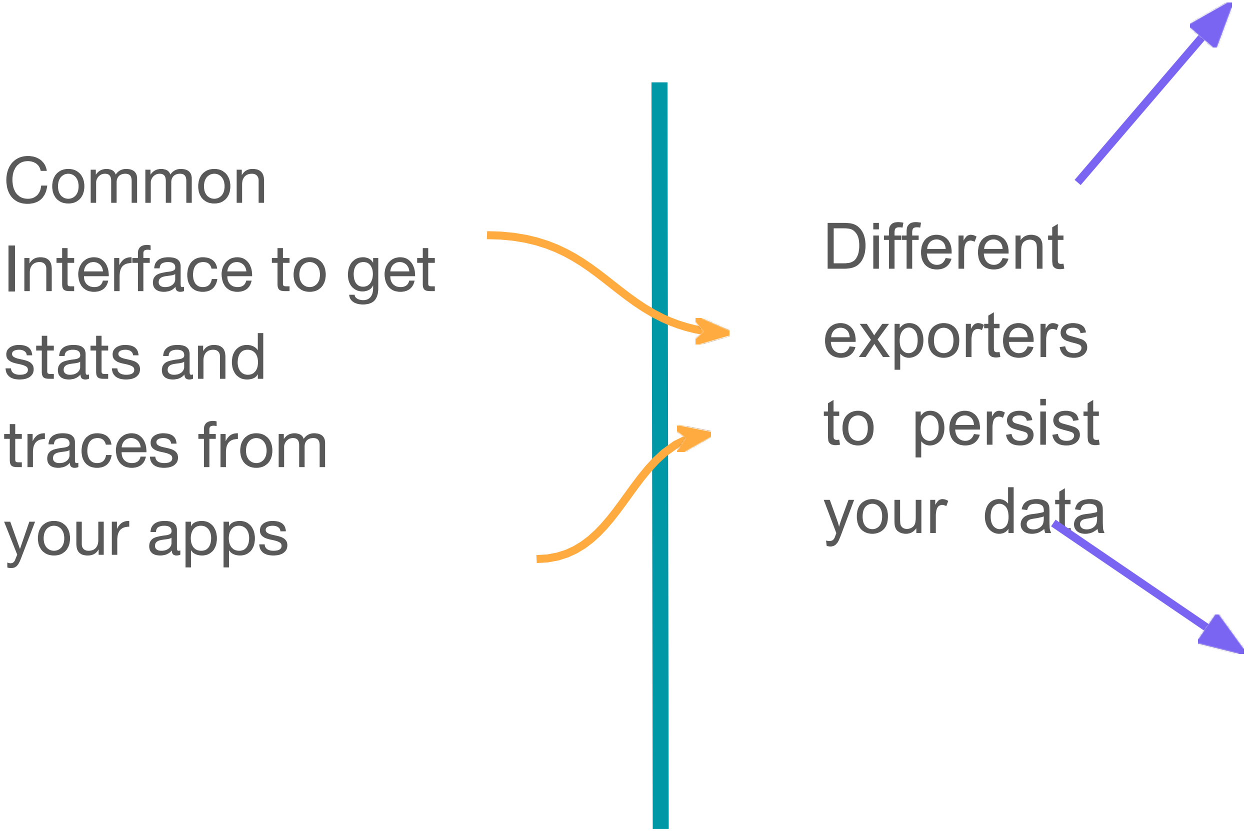
Basically this is another way of aggregating logs and metrics

Terminology

A **span** is the smallest unit in a trace

- A single HTTP request.
- A database query.
- A message execution in a queue system.
- A lookup from a key/value store.

OpenCensus: instrumentation spec and libraries by Google



Common Interface to get stats and traces from your apps

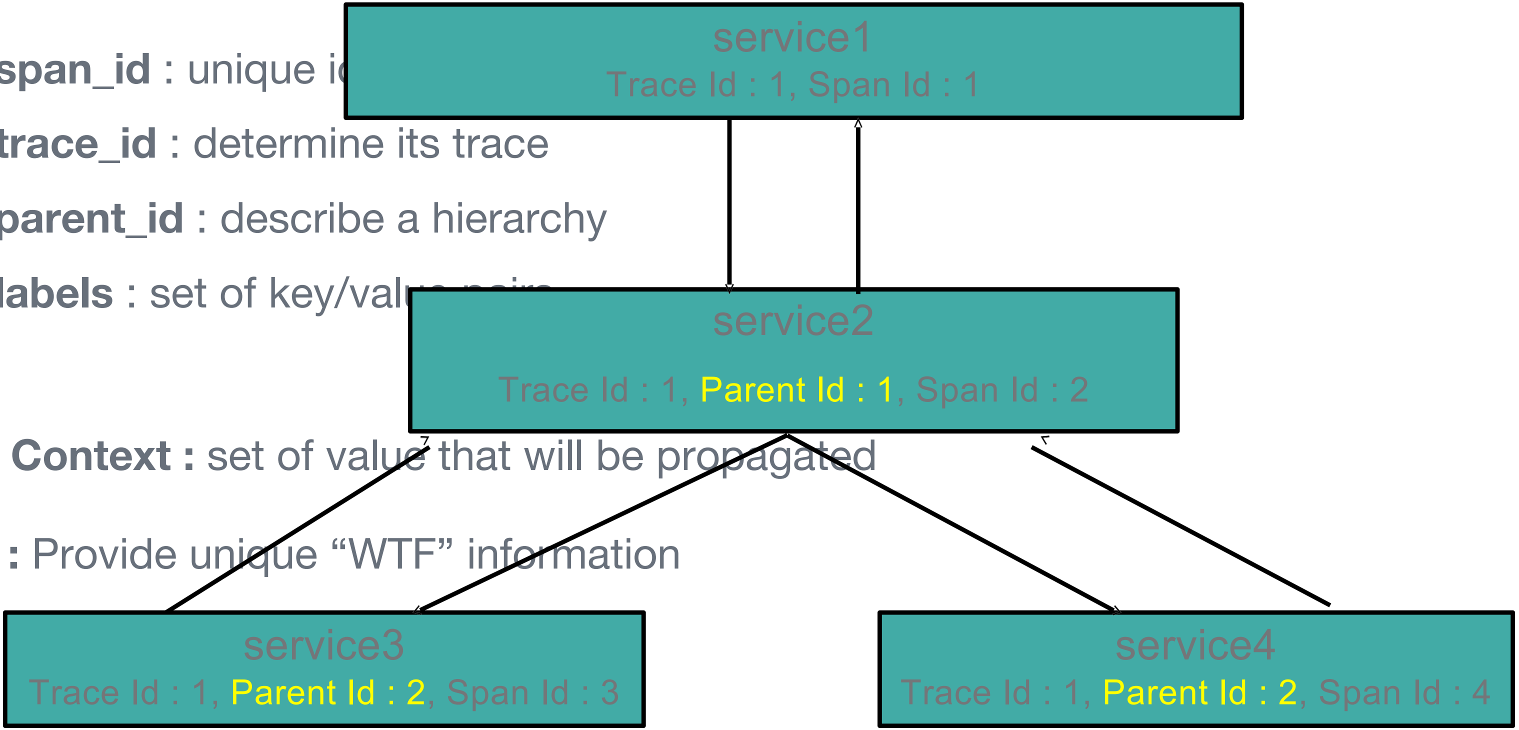
Different exporters to persist your data

Span Elements

- **span_id** : unique i
- **trace_id** : determine its trace
- **parent_id** : describe a hierarchy
- **labels** : set of key/valu

Span Context : set of value that will be propagated

Logs : Provide unique “WTF” information


service1
Trace Id : 1, Span Id : 1
service2
Trace Id : 1, Parent Id : 1, Span Id : 2
service3
Trace Id : 1, Parent Id : 2, Span Id : 3
service4
Trace Id : 1, Parent Id : 2, Span Id : 4

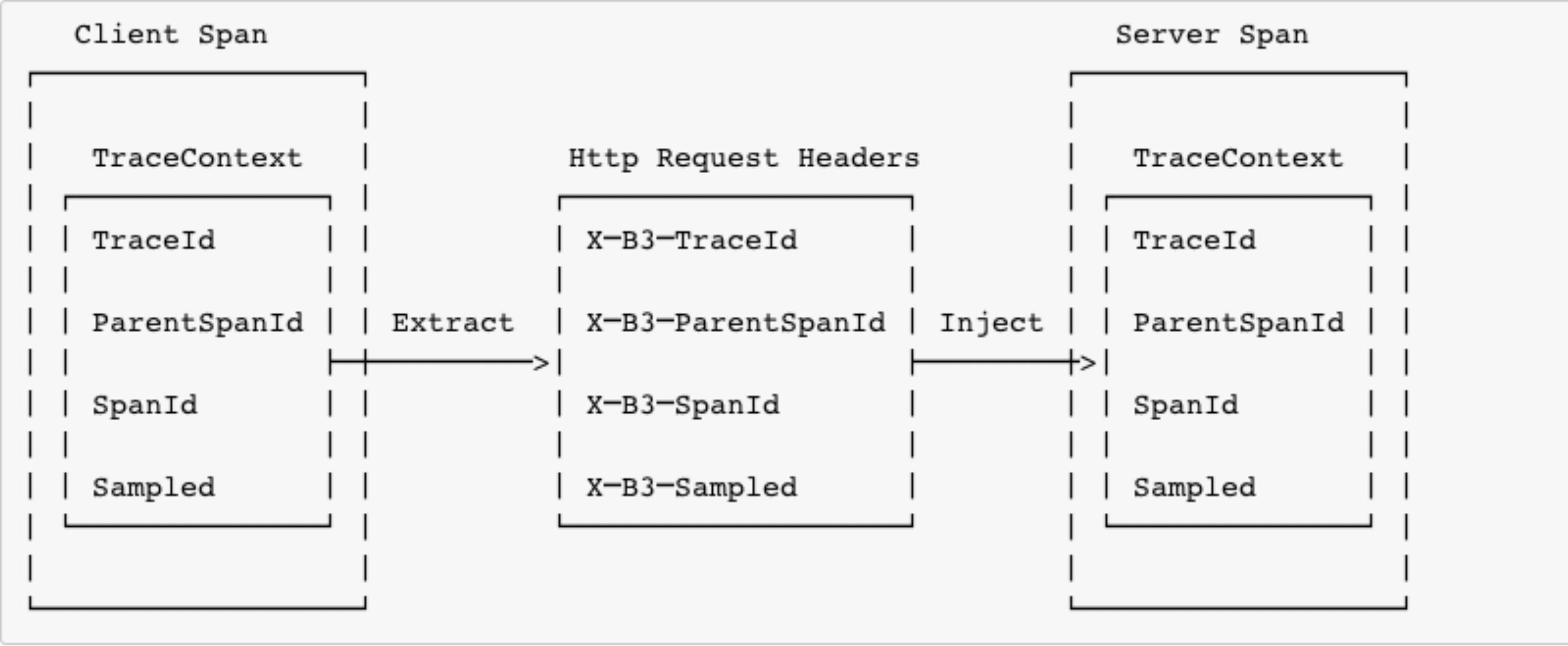
Tracers

- **Tracers** add logic to create unique trace ID
- **Trace ID is generated when the first request is made**
- **Span ID** is generated as the request arrives at each microservice
  - Tracers have instrumentation or sampling policy
- Tracers execute in your production apps

You still need logs!

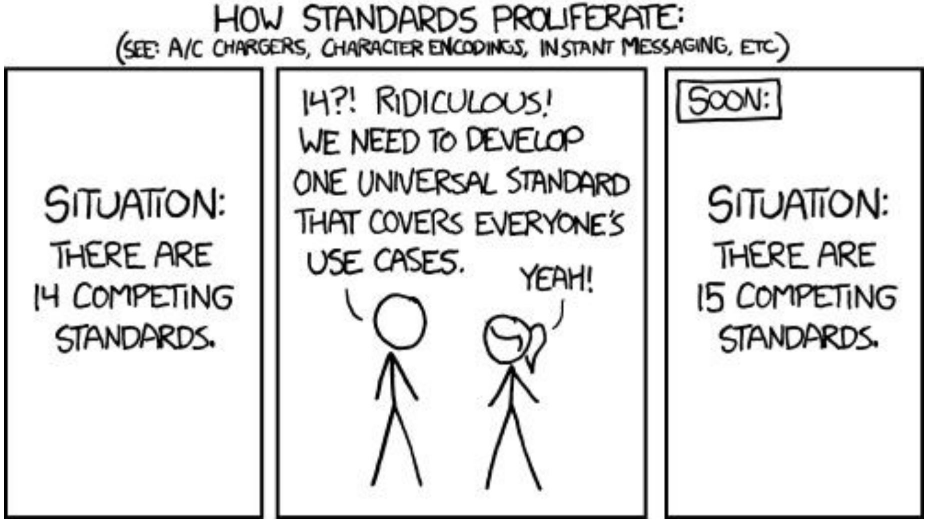
- The Original Instrumentation
- Provide unique details
- Help determine the Why, not just the what or when

B3-Propagation (original name of Zipkin: BigBrotherBird.)

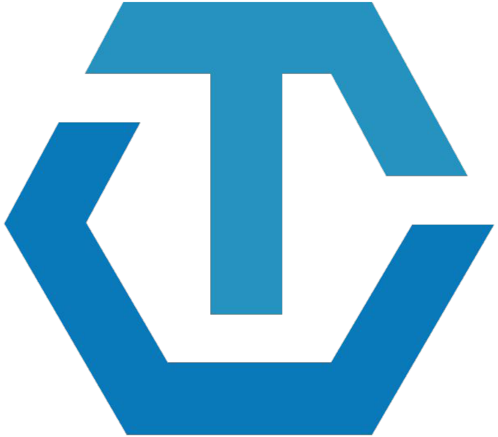


```
   Client Span                                                Server Span
┌──────────────────┐                                       ┌──────────────────┐
│                  │                                       │                  │
│   TraceContext   │           Http Request Headers        │   TraceContext   │
│ ┌──────────────┐ │          ┌───────────────────┐        │ ┌──────────────┐ │
│ │ TraceId      │ │          │ X─B3─TraceId      │        │ │ TraceId      │ │
│ │              │ │          │                   │        │ │              │ │
│ │ ParentSpanId │ │ Extract  │ X─B3─ParentSpanId │ Inject │ │ ParentSpanId │ │
│ │              ├─┼─────────>│                   ├────────┼>│              │ │
│ │ SpanId       │ │          │ X─B3─SpanId       │        │ │ SpanId       │ │
│ │              │ │          │                   │        │ │              │ │
│ │ Sampled      │ │          │ X─B3─Sampled      │        │ │ Sampled      │ │
│ └──────────────┘ │          └───────────────────┘        │ └──────────────┘ │
│                  │                                       │                  │
└──────────────────┘                                       └──────────────────┘
```

Do we need a standard?

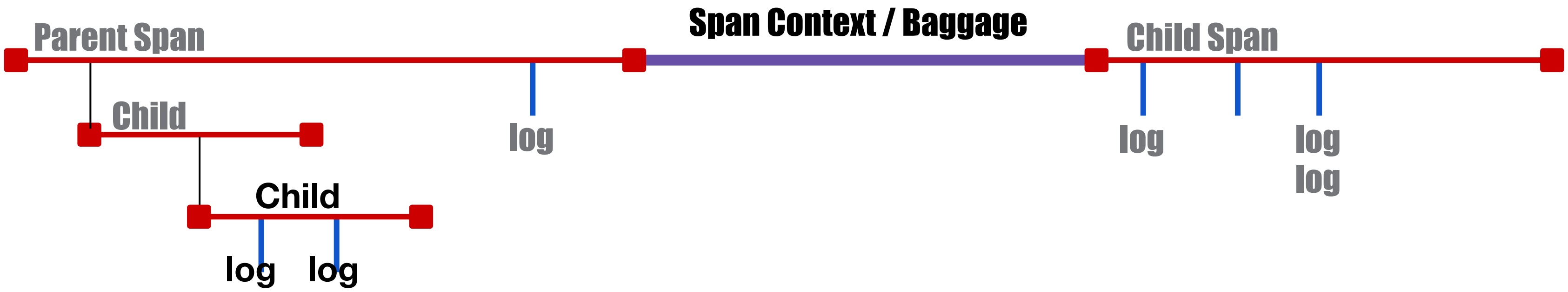

HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC)
SITUATION: THERE ARE 14 COMPETING STANDARDS.
14?! RIDICULOUS! WE NEED TO DEVELOP ONE UNIVERSAL STANDARD THAT COVERS EVERYONE'S USE CASES.
YEAH!
SOON:
SITUATION: THERE ARE 15 COMPETING STANDARDS.

- Applications can be written using different languages but at the end you need to build one single trace. We need to agree on a common standard/protocol.
- If you use a widely supported standard you can avoid vendor lock-in.



OPENTRACING

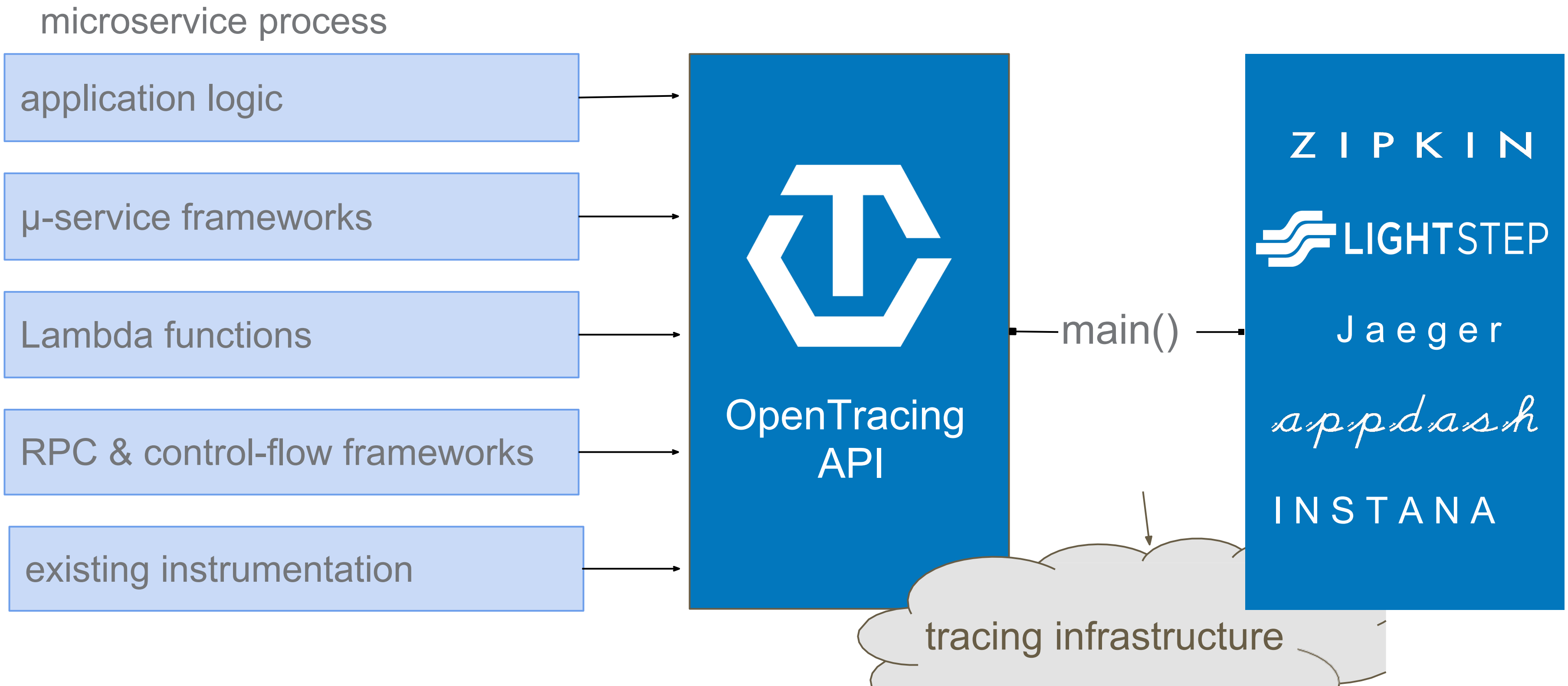
OpenTracing



Spans - Basic unit of timing and causality. Can be tagged with key/value pairs.
Logs - Structured data recorded on a span.
Span Context - serializable format for linking spans across network boundaries.
Carries baggage, such as a request and client IDs.
Tracers - Anything that plugs into the OpenTracing API to record information.

- OT engines
- Metrics (Prometheus)
- Logging

Fits to scale apps



OpenCensus

OpenCensus

Libraries for distributed tracing and metrics

- Java, Go, Node, Python, C++, C#, PHP, Ruby, Erlang
- Tracing, metrics, context propagation for every endpoint
- APIs for defining custom metrics, spans, sampling, etc.

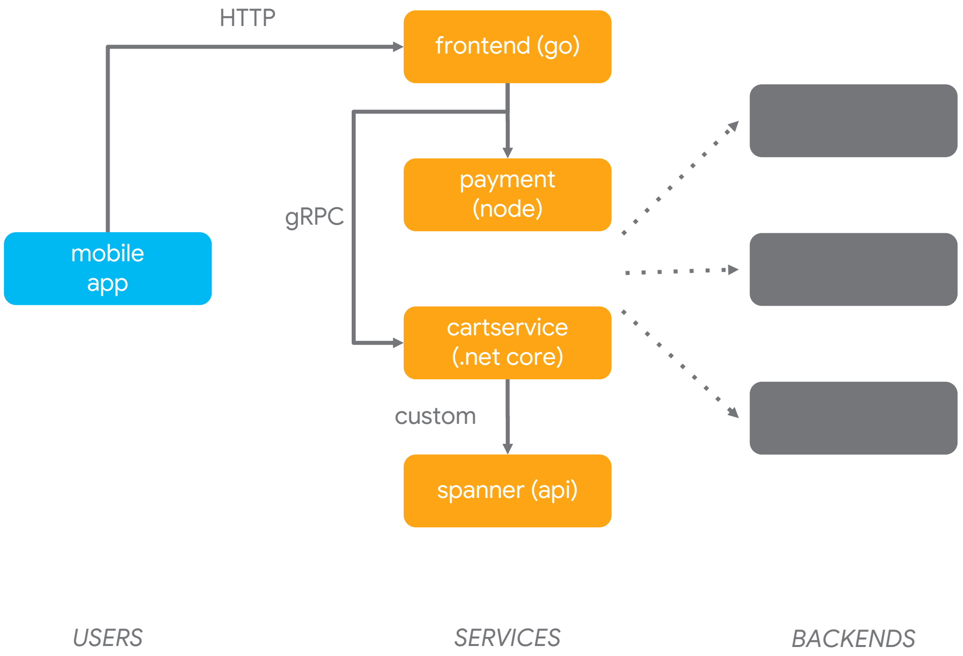
Full implementation + out of the box integrations

- Not just an API; no competing implementations
- Integrations enable automatic tracing, metrics collection, context propagation for each endpoint

Export telemetry to your backend of choice

- Send traces and metrics to Stackdriver, Prometheus, Zipkin, Jaeger, etc.
- Can export to multiple backends at once; different teams can use different tools

Example Deployment



Since last year: production ready

Java

Most mature OpenCensus support. Supports HTTP, GRPC, JDBC, MongoDB, Jetty, Serverlets

Go

Go has full API surface and supports okhttp, GRPC, SQL, Redis

node.js

Node.js has integrations with HTTP, gRPC

Python

Django, Flask, GC Client Libs, gRPC, http, MySQL, PostgreSQL,

pymongo, PyMySQL, Pyramid, requests, SQLAlchemy

Other languages

.NET php ERLANG C++

Languages feature matrix

Find more and detailed feature matrices on opencensus.io

	Java (PR)	Go (PR)	Node.js (PR)	Python (PR)	.Net	C++	Erlang	PHP	Ruby
Tracing									
Stats								□	□
Tags								□	□
Metrics					□	□	□	□	□
Context propagation									
W3C standard								□	□

Logging spec

Logs are an important signal for observability. The power of OpenCensus is to aggregate, filter and sample logs in a context of other signals.

1. Correlating logs with distributed trace context and tags and scope logs via OpenCensus tags API.
2. Having the OpenCensus agent ingest existing third and first-party logs from existing sources and send them to a backend.
3. Creating an API that developers use to write first-party log statements. This API would provide benefits versus existing logging frameworks like Log4J, but does not seek to replace them.

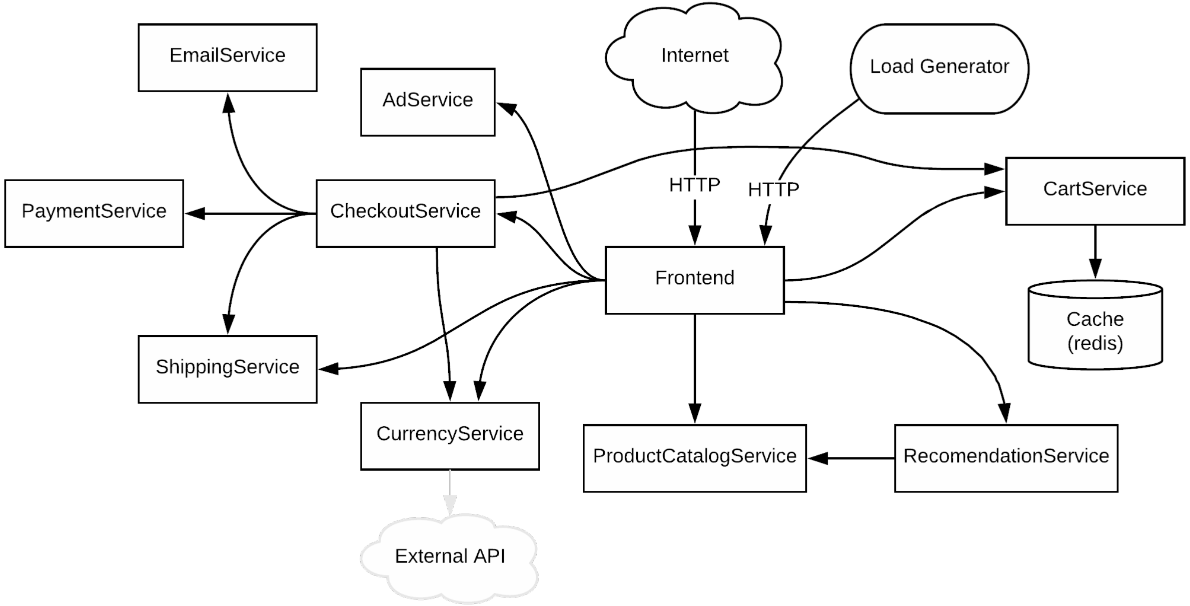
1
Correlation context on logs
2
Ingest logs into agent
3
Logging API and metadata

OpenCensus & OpenTracing merger

New name, but not a third project

- Full merger
- A single community
- A single set of integrations!
- Technical committee is overseeing API merger
- Find out more on the OpenCensus blog and at Kubecon EU

Play with it for yourself



Hipster Store: https://github.com/GoogleCloudPlatform/microservices-demo

Wnat to get involved?

- Get involved:
  - https://github.com/census-instrumentation
- Join the conversation in Gitter:
  - census-instrumentation

FAQ 1: Can I store traces for everything, everywhere?

At your own risk…

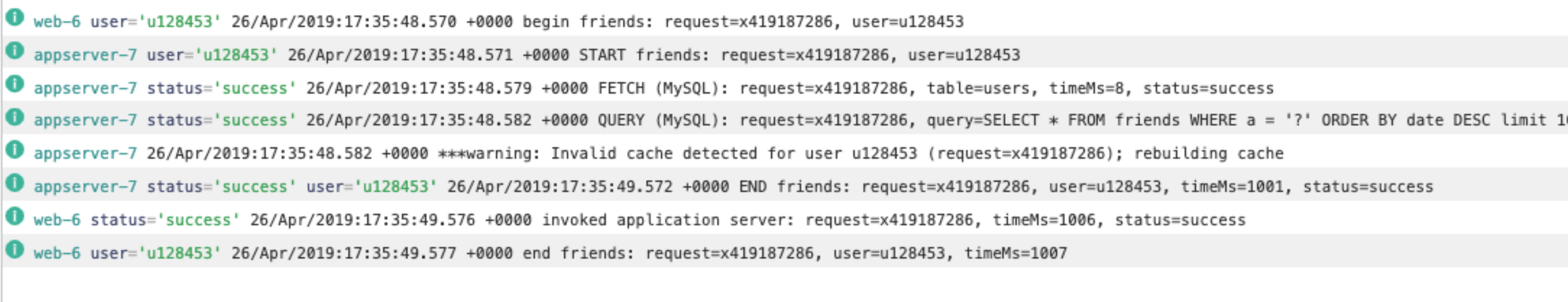
- → Really high cardinality
- → High write throughput

Databases like InfluxDB, Cassandra, MongoDB are a better option than MySQL, Postgres but it always depends on traffic and amount of data.

In the context of **databases**, **cardinality** refers to the uniqueness of data values contained in a column. High **cardinality means** that the column contains a large percentage of totally unique values. Low **cardinality means** that the column contains a lot of “repeats” in its data range.

FAQ 2: I already log stuff, isn’t that good enough?

Actually, if your logs are set for request ID’s, it’s pretty darn good



```
web-6 user='u128453' 26/Apr/2019:17:35:48.570 +0000 begin friends: request=x419187286, user=u128453
appserver-7 user='u128453' 26/Apr/2019:17:35:48.571 +0000 START friends: request=x419187286, user=u128453
appserver-7 status='success' 26/Apr/2019:17:35:48.579 +0000 FETCH (MySQL): request=x419187286, table=users, timeMs=8, status=success
appserver-7 status='success' 26/Apr/2019:17:35:48.582 +0000 QUERY (MySQL): request=x419187286, query=SELECT * FROM friends WHERE a = '?' ORDER BY date DESC limit 1
appserver-7 26/Apr/2019:17:35:48.582 +0000 ***warning: Invalid cache detected for user u128453 (request=x419187286); rebuilding cache
appserver-7 status='success' user='u128453' 26/Apr/2019:17:35:49.572 +0000 END friends: request=x419187286, user=u128453, timeMs=1001, status=success
web-6 status='success' 26/Apr/2019:17:35:49.576 +0000 invoked application server: request=x419187286, timeMs=1006, status=success
web-6 user='u128453' 26/Apr/2019:17:35:49.577 +0000 end friends: request=x419187286, user=u128453, timeMs=1007
```

Summary

- Observability requires deep insights into increasingly complex architectures
- Integrated toolsets will deliver important improvements in team productivity
- Make sure the technology you choose is able to support these requirements at the scale, performance, and cost effectiveness today’s challenges require

Questions?