The Whats and Whys of Tracing

OpenTelemetry FTW

Dave McAllister

splunk> turn data into doing™

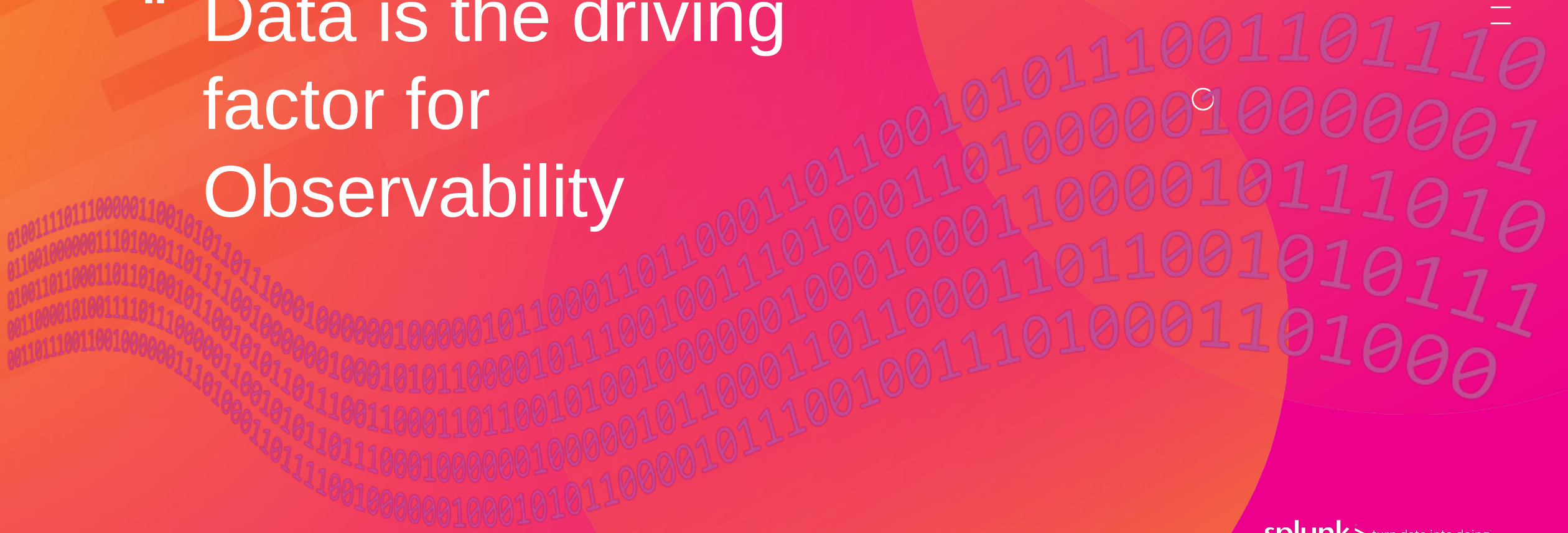
Dave McAllister

ay 115

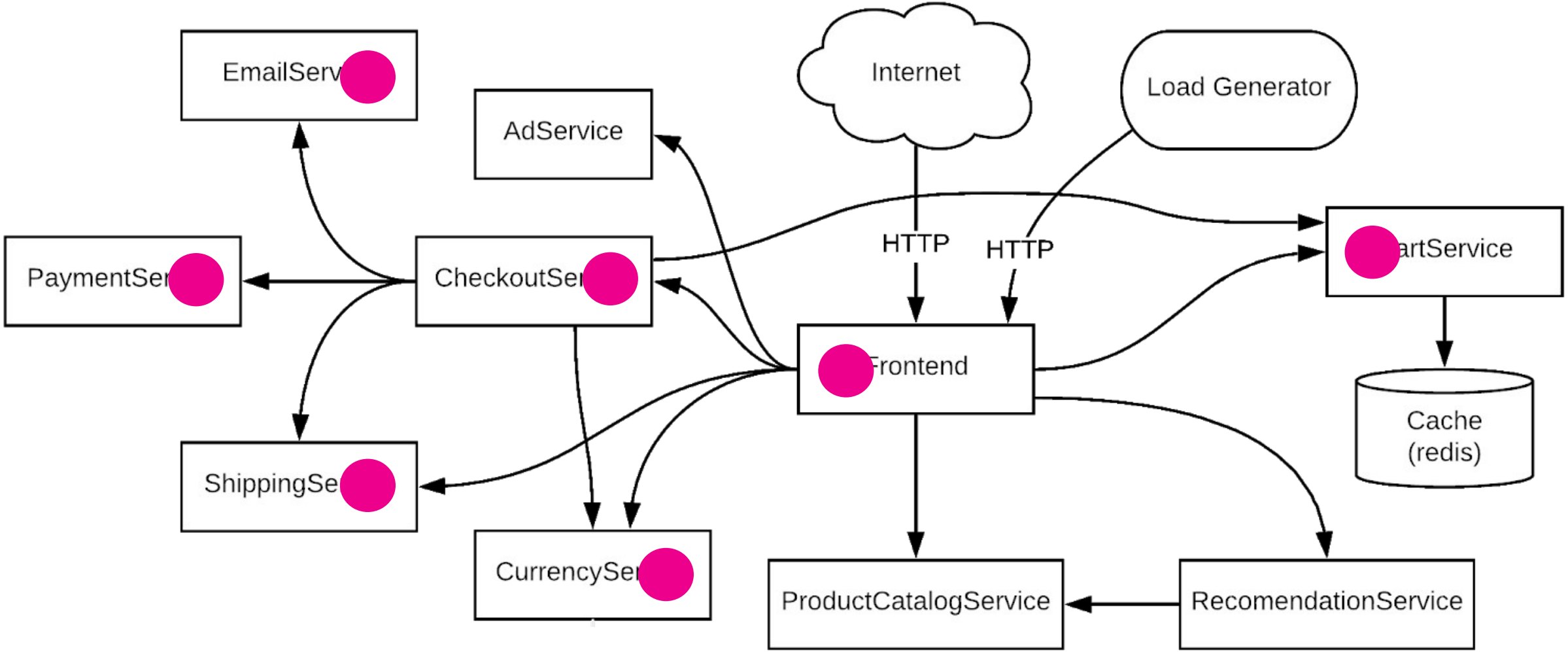
Senior Technical Evangelist

splunk>live!

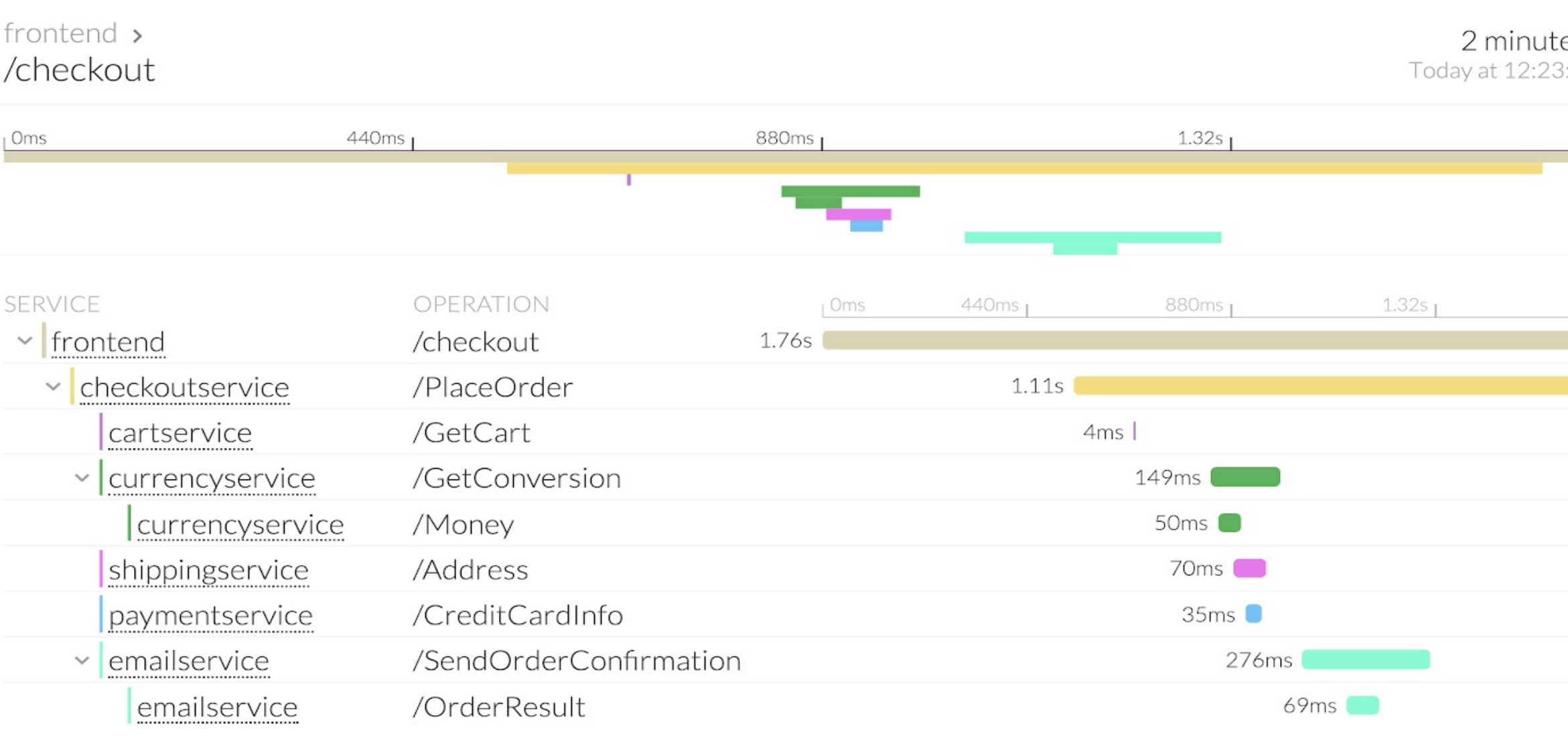
> “ Data is the driving factor for Observability



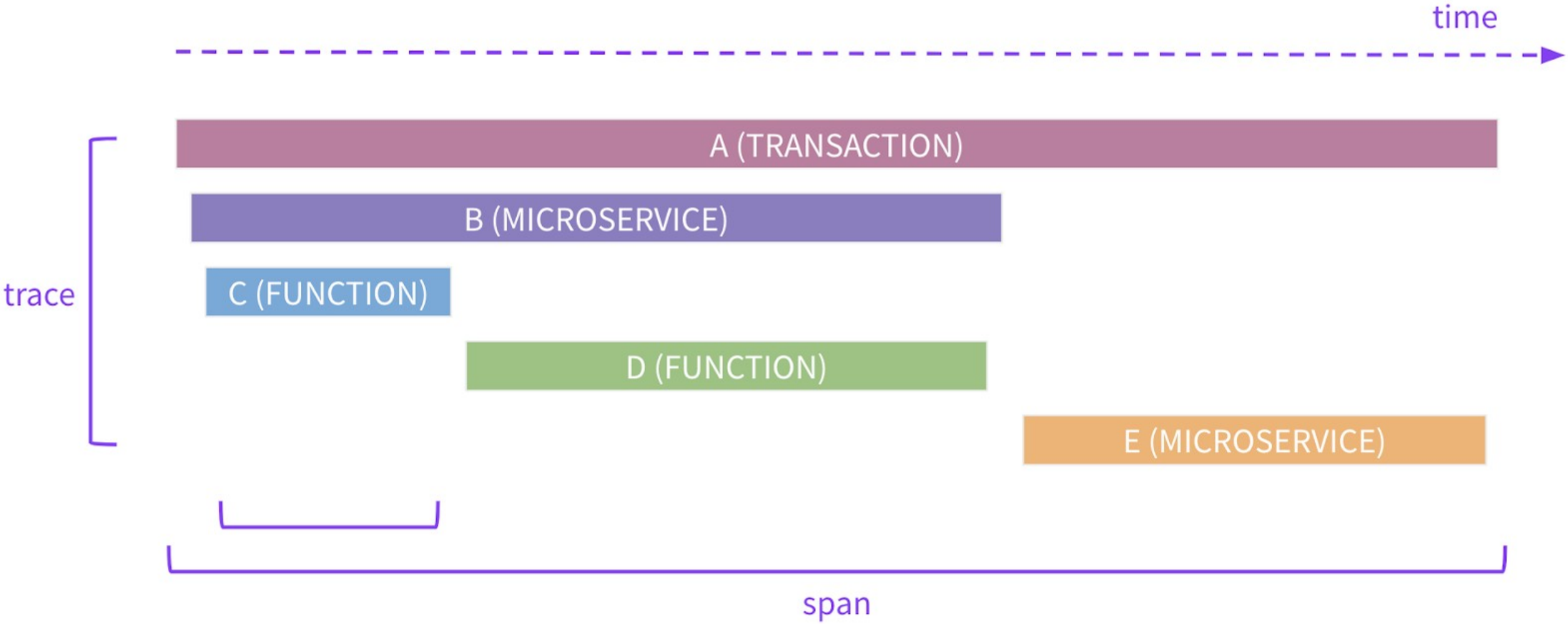
EXAMPLE MICROSERVICE ARCHITECTURE



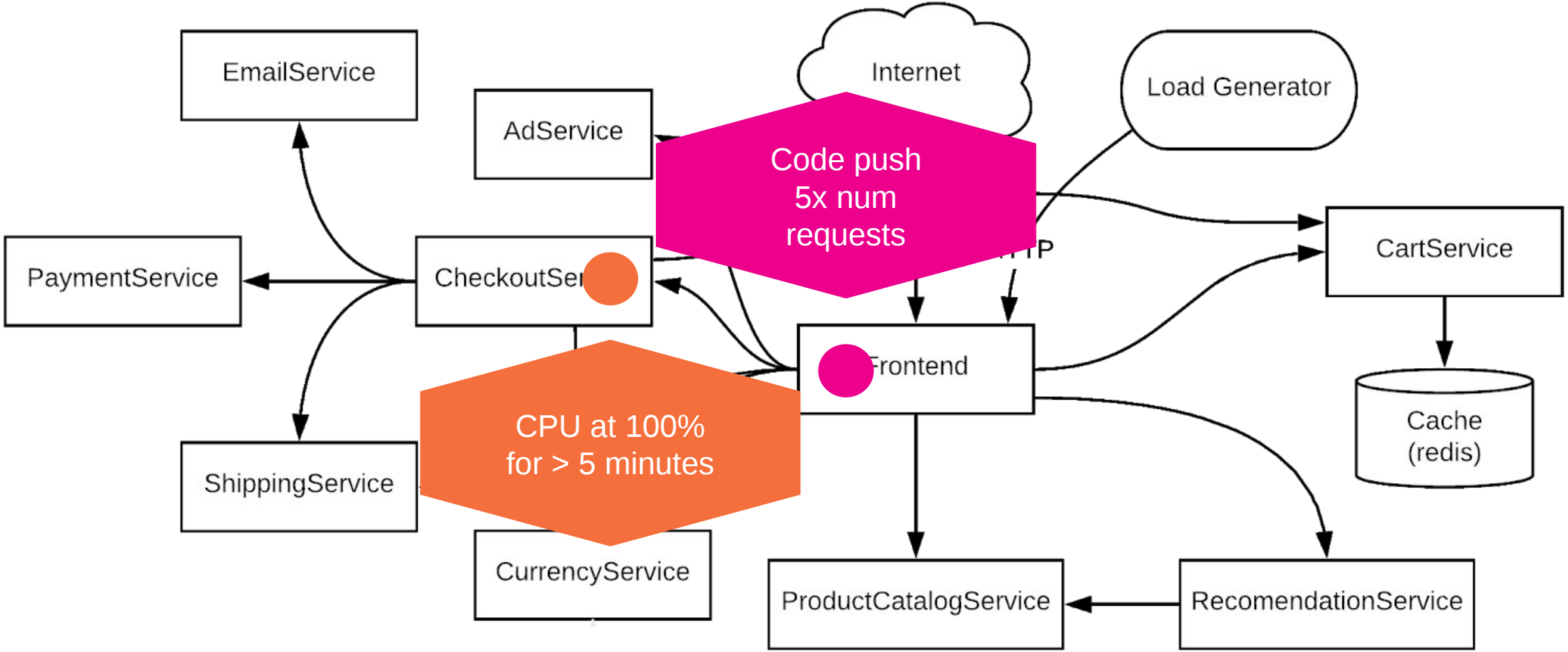
DISTRIBUTED TRACE VISUALIZATION



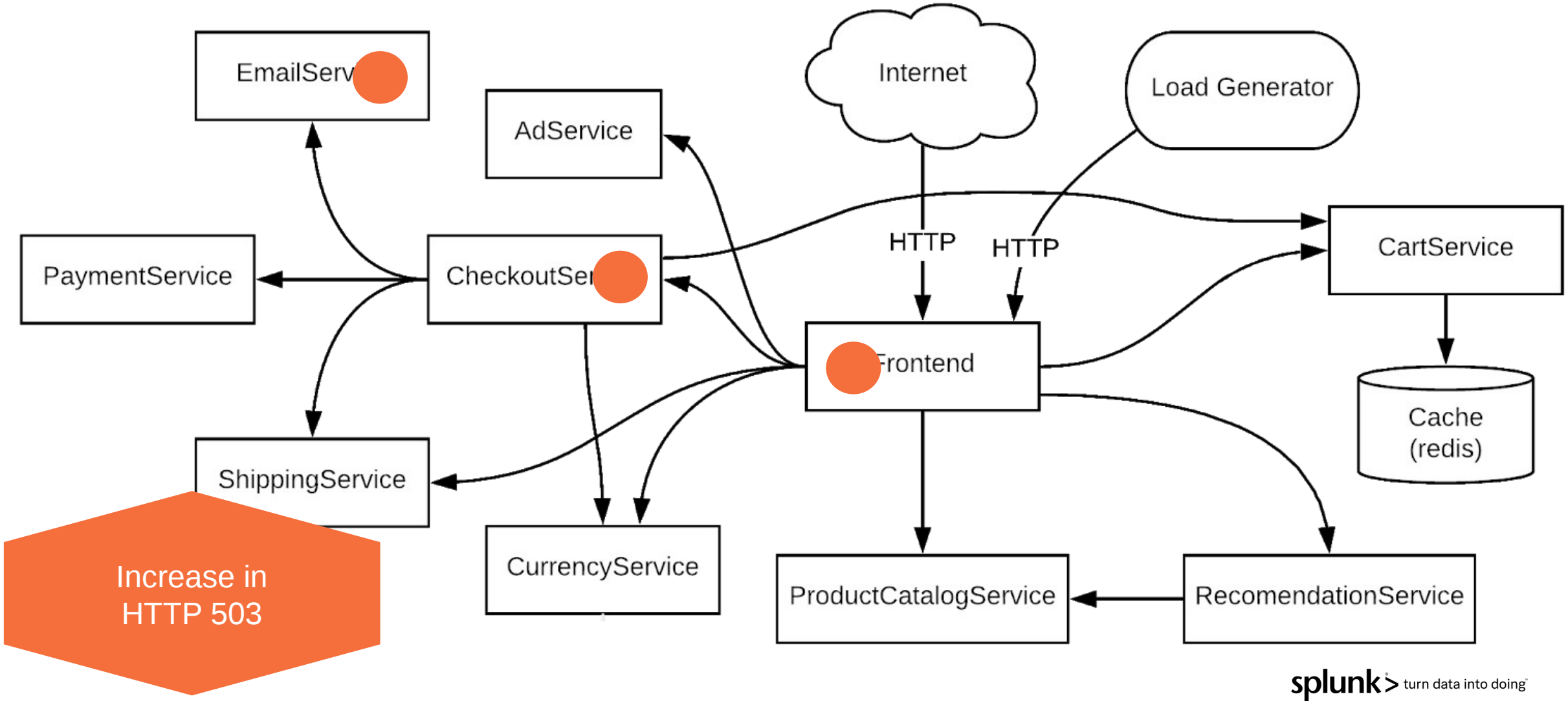
DISTRIBUTED TRACE DETAILS



METRICS IN MICROSERVICE ARCHITECTURE



LOGS IN MICROSERVICE ARCHITECTURE



> “Observability is not the microscope. It’s the clarity of the slide under the microscope.”
>
> *Baron Schwartz*

Data Collection

Standards-based agents, cloud-integration

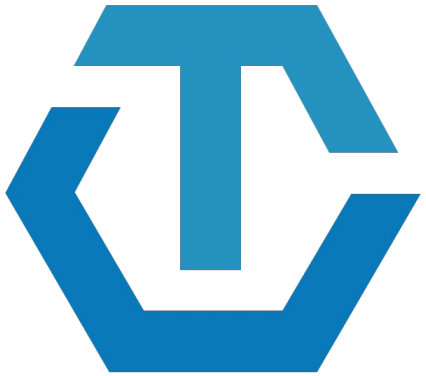
Automated code instrumentation

Support for developer frameworks

Any code, any time

No cardinality limits

What is OpenTelemetry?



OPENTRACING

+

OpenCensus

=

OpenTelemetry

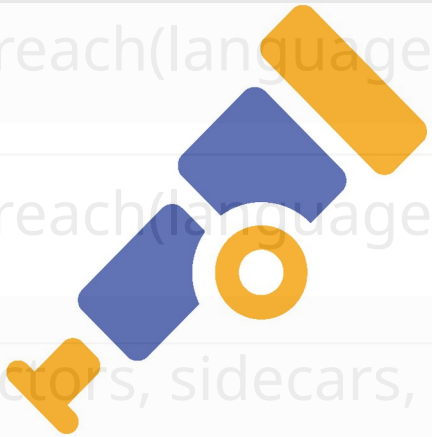
OpenTelemetry: **the next major version** of *both* OpenTracing and OpenCensus

Cloud Native Telemetry

Telemetry "verticals"

Telemetry "layers"

	Tracing	Metrics	Logs, etc
Instrumentation APIs			
Canonical implementations			
Data infrastructure			
Interop formats			



OpenTelemetry

Project Stats

CNCF DevStats

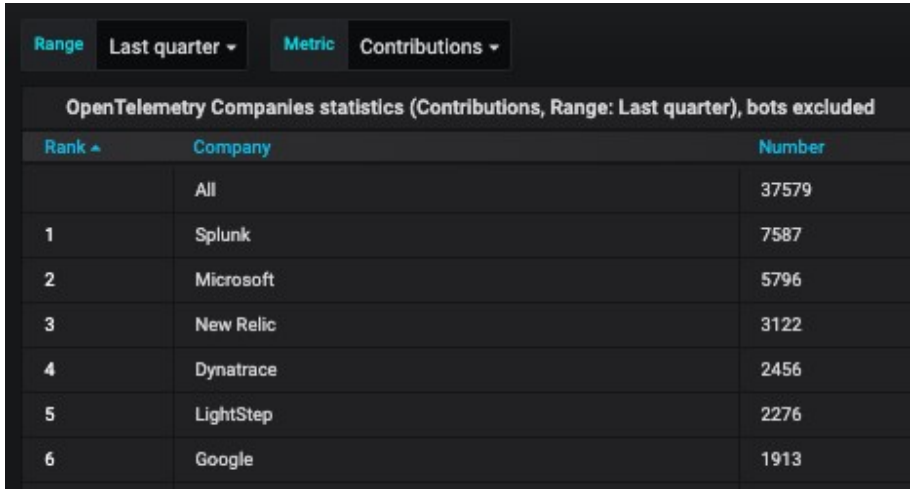
- **General:** 149 companies
- **Contributors:** 660+ unique contributors and 60K+ contributions

Community Stats

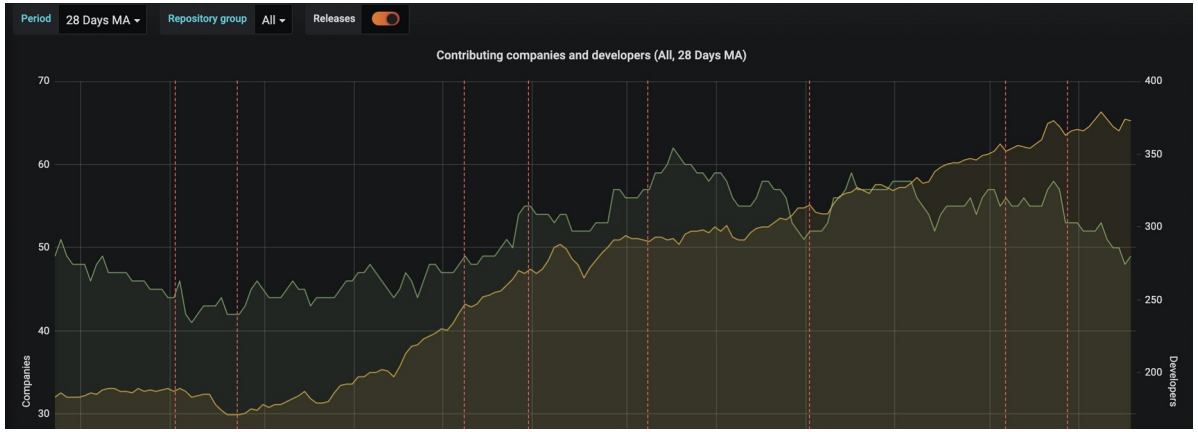
- **Cloud Providers:** Azure and GCP
- **Users (and contributors)**: Mailchimp, Postmates, Shopify, Zillow

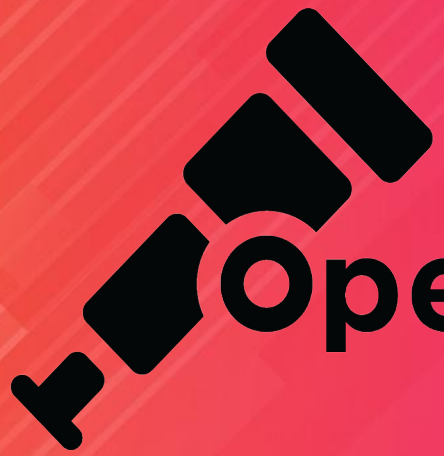
CNCF Project Collaboration

- **Fluentbit:** Potential log agent for OpenTelemetry
- **Jaeger:** Plan to leverage client libraries and collector (collector already announced)



Rank	Company	Number
	All	37579
1	Splunk	7587
2	Microsoft	5796
3	New Relic	3122
4	Dynatrace	2456
5	LightStep	2276
6	Google	1913





OpenTelemetry

is the second most active project in CNCF today!

(per CNCF DevStats)

Why do you want to trace?

splunk> turn data into doing™

What problems are you trying to solve?

A tiny subset of answers*

- Performance issues
- Mean time to resolution and detection is too high
- Metrics and logs are missing valuable context.
- More data types can provide better answers

Gaps in current observability method

A tiny subset of answers*

Difficult to correlate observed behavior with what is operating.

Difficult to collect data in different formats

No (observable) gaps and want to try something new

Goals of tracing

Questions to ask yourself and of your organization

Which teams/components would benefit the most?

How much resources are available to this effort?

What is the short term goal and long term goal?

If already in use, how is adoption? What are possible roadblocks to higher adoption?

Why tracing now?

Architecture

splunk> turn data into doing™

Components

1. **Specifications**
   a. API
   b. SDK
   c. Data
2. **Collector**
   a. Vendor-agnostic way to receive, process, and export data
   b. Default way to collect instrumented apps
   c. Can be deployed as an agent or service
3. **Client Libraries**
   a. Vendor-agnostic app instrumentation
   b. Support for traces and metrics
   c. Automatic trace instrumentation
4. **Incubating: Logging**

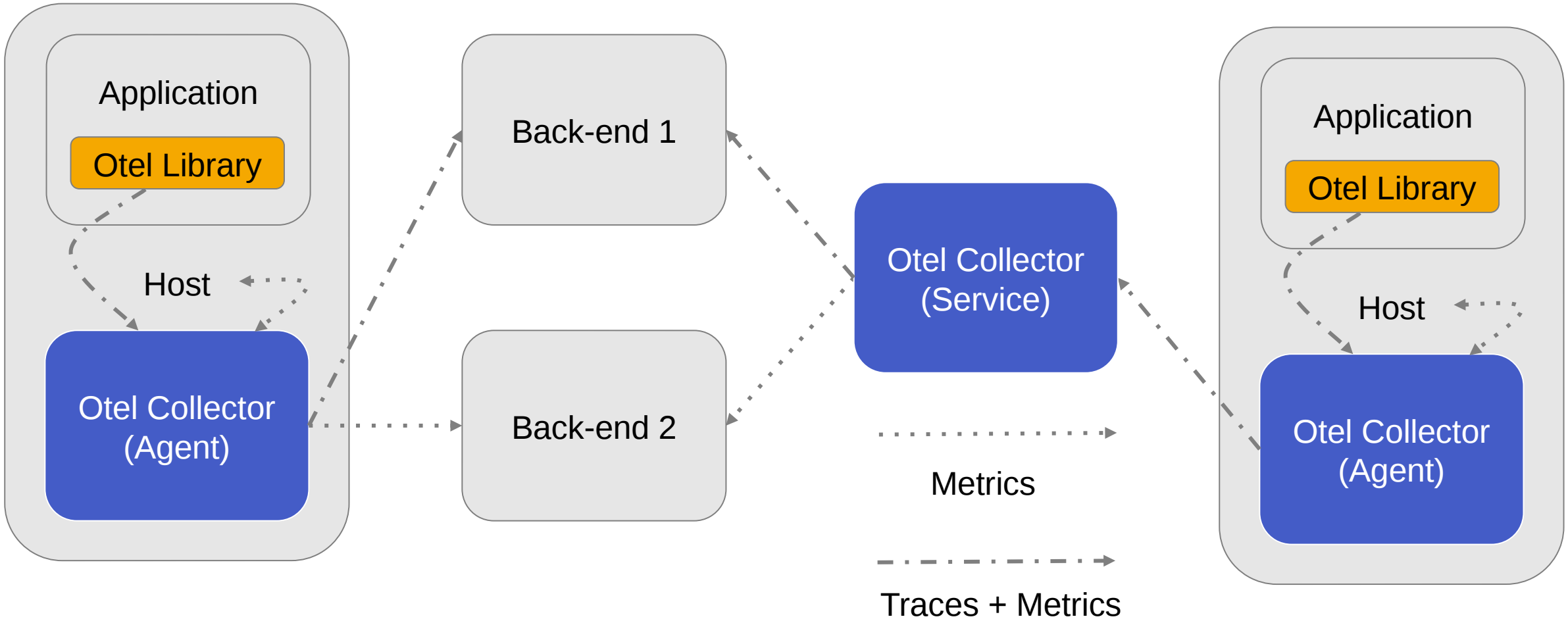
Status = Beta for Traces + Metrics:

- Collector
- Erlang
- Go
- Java (including auto instrumentation)
- Javascript (including web)
- Python (auto instrumentation planned)

Coming soon:

- .NET (auto instrumentation planned)
- Ruby (auto instrumentation planned)

Reference Architecture: OpenTelemetry



Specifications

splunk> turn data into doing™

Tracing Basics

- **Context:** W3C trace-context, B3, etc.
- **Tracer:** get context
- **Spans:** “call” in a trace
  - **Kind:** client/server, producer/consumer, internal
  - **Attributes:** key/value pairs; tags; metadata
  - **Events:** named strings
  - **Links:** useful for batch operations
- **Sampler:** always, probabilistic, etc.
- **Span processor:** simple, batch, etc.
- **Exporter:** OTLP, Jaeger, Prometheus, etc.

Tracing and Semantic Conventions

In OpenTelemetry, spans can be created freely and it's up to the implementor to annotate them with attributes specific to the represented operation.
Some span operations represent calls that use well-known protocols like HTTP or database calls. It is important to unify attribution.

- **HTTP:** http.method, http.status_code
- **Database:** db.type, db.instance, db.statement
- **Messaging:** messaging.system, messaging.destination
- **FaaS:** faas.trigger

Metrics Basics

- **Context:** span and correlation
- **Meter:** used to record a measurement
- **Raw Measurement**
  - **Measure:** name, description, unit of values
  - **Measurement:** single value of a measure
- **Metric:** a measurement
  - **Kind:** counter, measure, observer
  - **Label:** key/value pair; tag; metadata
- **Aggregation**
- **Time**

Resource SDK + Semantic Conventions

A Resource is an immutable representation of the entity producing telemetry.

All of these

- **Environment:** Attributes defining a running environment (e.g. cloud)
- **Compute instance:** Attributes defining a computing instance (e.g. host)
- **Deployment service:** Attributes defining a deployment service (e.g. k8s).
- **Compute unit:** Attributes defining a compute unit (e.g. container, process)

OpenTelemetry and Logs (Incubating!)

- The Log Data Model Specification :
  https://github.com/open-telemetry/oteps/blob/master/text/logs/0097-log-data-model.md#motivation
- Designed to map existing log formats and be semantically meaningful
- Mapping between log formats should be possible
- Three sorts of logs and events
  - System Formats
  - Third-party applications
  - First-party applications

OpenTelemetry and Logs

Two Field Kinds:

- Named top-level fields
- Fields stored in key/value pairs

Field Name	Description
Timestamp	Time when the event occurred.
TraceId	Request trace id.
SpanId	Request span id.
TraceFlags	W3C trace flag.
SeverityText	The severity text (also known as log level).
SeverityNumber	Numerical value of the severity.
Name	Short event identifier.
Body	The body of the log record.
Resource	Describes the source of the log.
Attributes	Additional information about the event.

Observability drives Evidence-based Debugging

Debugging for complex systems is iterative

- Start with a high-level metric
- Drill down and detangle based on fine-grained data/observations
- Make the right deductions based on the evidence

Collector

splunk> turn data into doing™

Objectives

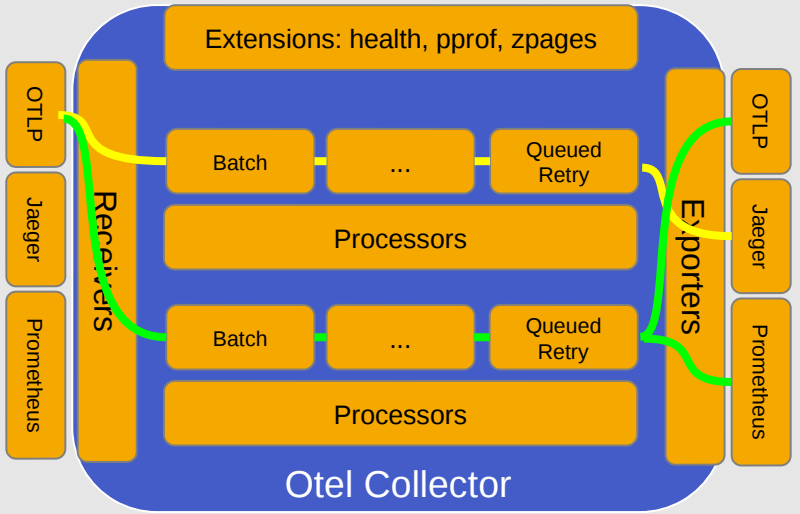
The OpenTelemetry Collector offers a vendor-agnostic implementation on how to receive, process, and export telemetry data in a seamless way.

- **Usable:** Reasonable default configuration, supports popular protocols, runs and collects out of the box.
- **Performant:** Highly performant under varying loads and configurations.
- **Observable:** An exemplar of an observable service.
- **Extensible:** Customizable without touching the core code.
- **Unified:** Single codebase, deployable as an agent or collector with support for traces, metrics, and logs (future).

But Why?

- Offload responsibility from the application
  - Compression
  - Encryption
  - Retry
  - Tagging / Redaction
  - Vendor-specific exporting
- Time-to-value
  - Language-agnostic; makes changes easier
  - Set it and forget it; instrumentation that is ready for the Collector
  - Vendor-agnostic and easily extensible

Architecture



Getting Started: Traces (Automatic)

```
java -javaagent:path/to/opentelemetry-auto-<version>.jar \
     -Dota.exporter.jar=path/to/opentelemetry-auto-exporters-otlp-
<version>.jar \
     -Dota.exporter.otlp.endpoint=localhost:55680 \
     -Dota.exporter.otlp.service.name=shopping \
     -jar myapp.jar
```

- Instruments known libraries with no code (only runtime) changes
- Adheres to semantic conventions
- Configurable via environment and/or runtime variables
- Can co-exist with manual instrumentation

WARNING: Do not use two different auto-instrumentation solutions on the same service.

Roadmap

- Rest of client libraries to beta ASAP
- Move to GA later this year for traces and metrics

- Tracing auto instrumentation for all languages
- Add initial log support (goal of beta later this year)

- Improve documentation
- Increase adoption; get case studies
- Make getting started really easy

Problem solving

Imagine being paged

What questions do you ask yourself when you are being paged?

How do you filter out noise?

How do you determine what isn’t causing an issue?

How do you determine impact?

A span for everything or bare minimum

It depends

Remember why you want to trace

Rule of thumb: Start with service boundaries and 3rd party calls

Iterative process

There is information overload

Make it easy for teams to get tracing (for free or almost free)

Next Steps

- Join the conversation: https://gitter.im/open-telemetry/community
- Join a SIG: https://github.com/open-telemetry/community#special-interest-groups
- Submit a PR (consider good-first-issue and help-wanted labels)

Thank You

splunk> turn data into doing™