OpenTracing Isn't just Tracing: Measure Twice, Instrument Once

Ted Young, OpenTracing + LightStep

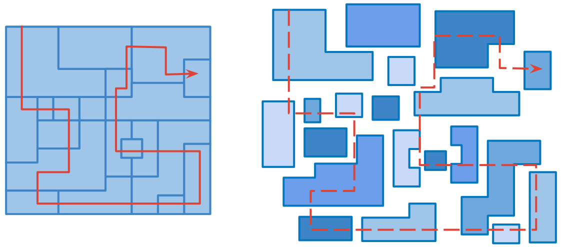
March 29, 2017
Kubecon Berlin

Part I:
Why Care About Tracing?

Microservices: 1 story, N storytellers

Microservices are here to stay: decoupled eng teams, CI, CD, etc

… but they break legacy monitoring tools: great monitoring *tells stories* about your system. Process-scoped monitoring can never do that.



How *do* you "tell stories" about a modern architecture?

Distributed Tracing: consider all requests from all services, then connect the dots

OPENTRACING

Great… So why isn’t tracing ubiquitous?

Tracing instrumentation has been too hard.

Lock-in is unacceptable: instrumentation must be decoupled from vendors

Monkey patching insufficient: instrumentation is by humans, for humans

Inconsistent APIs: tracing semantics must not be language-dependent

Handoff woes: tracing libs in *Project X* don’t hand-off to tracing libs in *Project Y*

Part II:
Enter OpenTracing

OpenTracing in a nutshell

OpenTracing addresses the instrumentation problem.

- Open and standardized API under the CNCF.
- Useful for a wide variety of instrumentation.
- Separates what you choose to instrument from what you choose to collect.
- Especially good for instrumenting OSS libraries and frameworks.

OPENTRACING

A young, fast-growing project

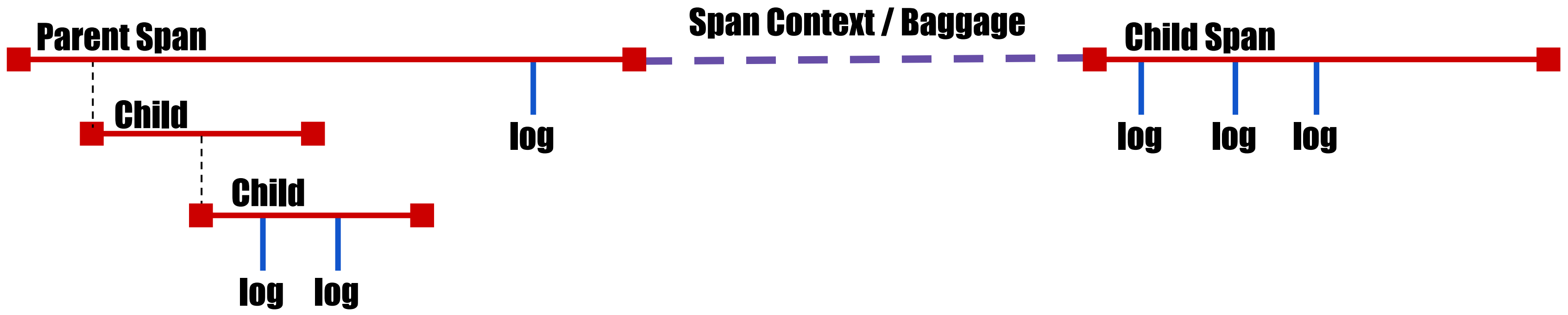
One year old! 🎂 Announced v1.0 spec in August 2016

Tracer Implementations: Zipkin, Uber's "Jaeger" Zipkin sibling, Hawkular, Appdash, LightStep, and a few smaller tracing systems

Some Companies using OpenTracing:

lyft twilio docker Joyent box Bloomberg Pinterest airbnb ZENEFITS

Opentracing Architecture



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. ZipKin, LightStep, and Jaeger. But also metrics (Prometheus) and logging.

OPENTRACING

Uses for OpenTracing

Logging - Easy to output to any logging tool, even from OSS components.

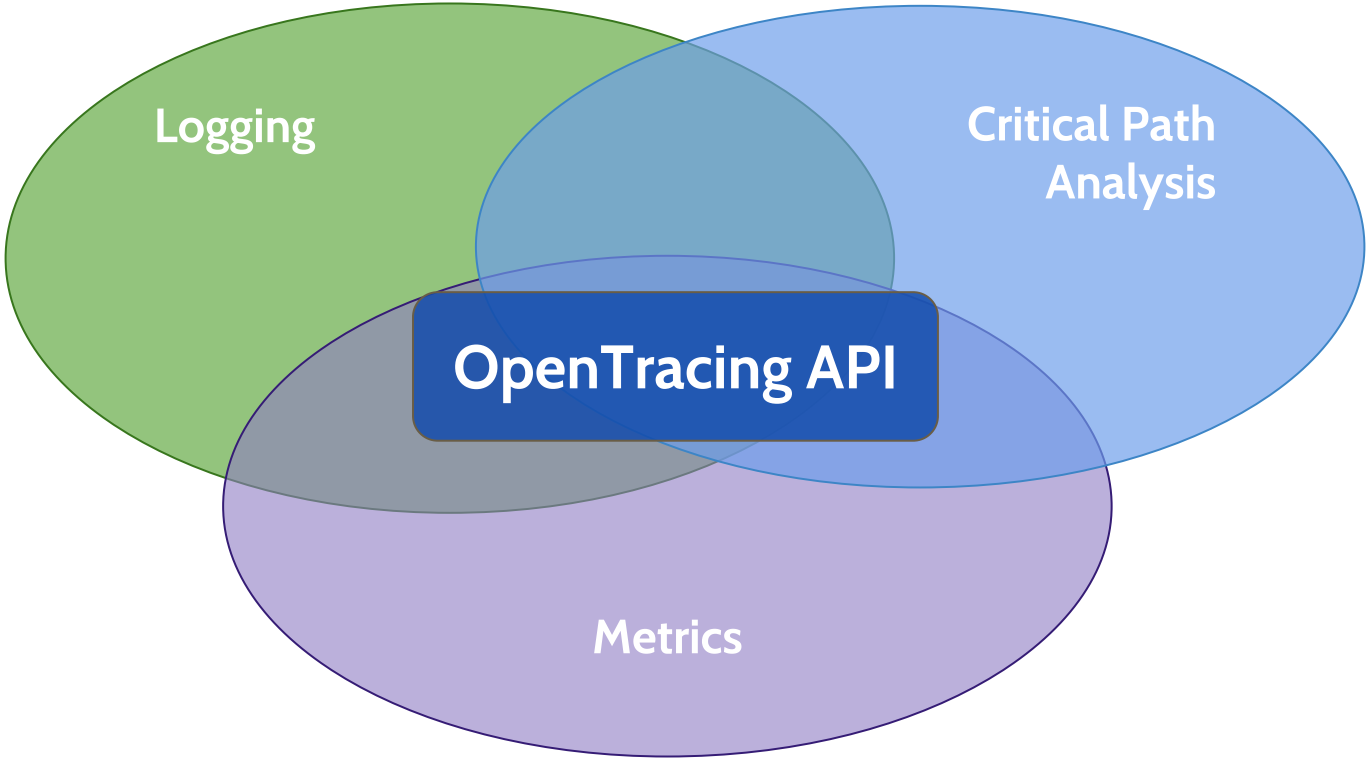
Metrics/Alerting - Measure based on tags, span timing, log data.

Context Propagation - Use baggage to carry request and user ID’s, etc.

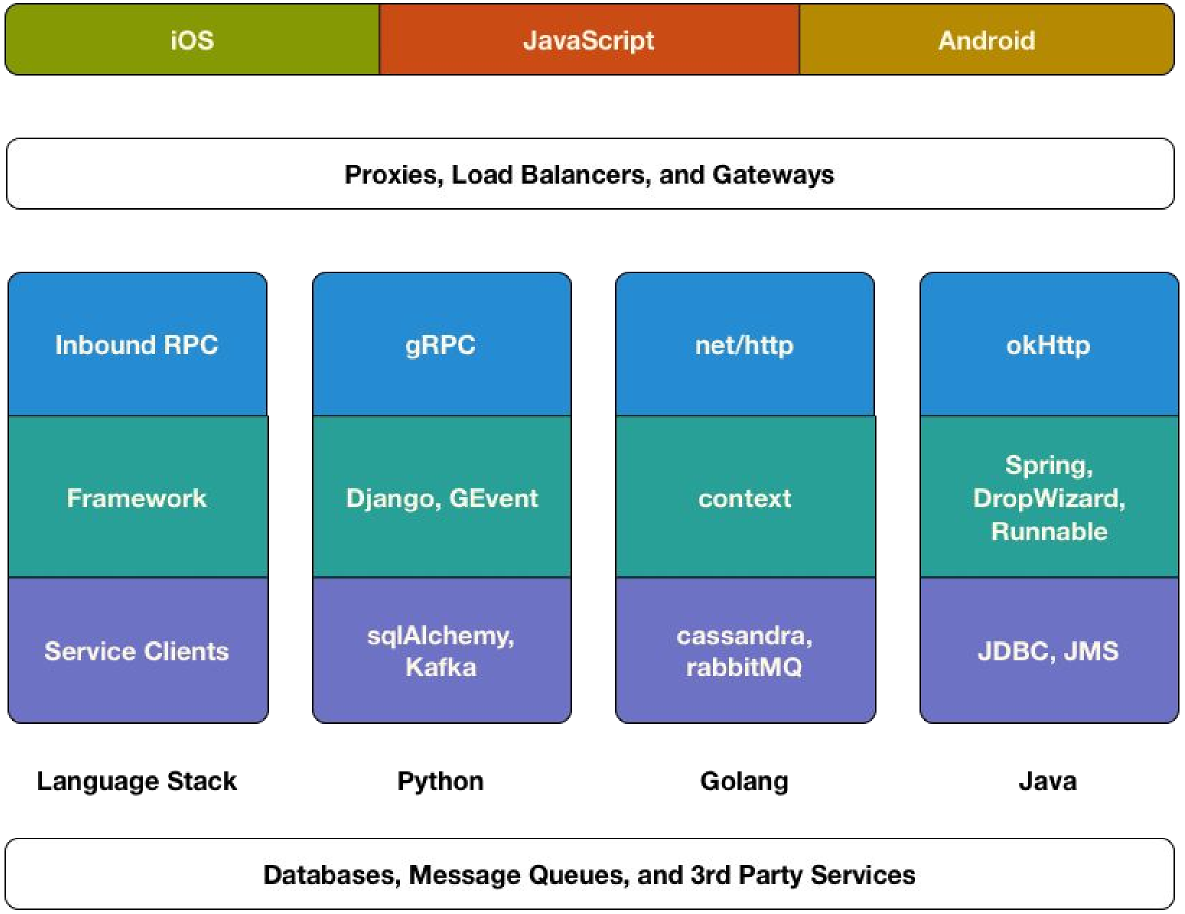
Critical Path Analysis - Drill down into request latency in very high fidelity.

System Topology Analysis - Identify bottlenecks due to shared resources.

Replaces Traditional Instrumentation



Turnkey Tracing



Part III:
Prometheus Example

Imagine a world ... with faster access to donuts

Simple Prometheus Integration

```
type PrometheusTracer struct {
  component  string
  Latency    *prometheus.SummaryVec
  ErrorCount *prometheus.CounterVec
}

func (t *PrometheusTracer) RecordSpan(span basictracer.RawSpan) {

  t.Latency
  .WithLabelValues(span.Operation)
  .Observe(float64(span.Duration))

  if _, found := span.Tags["error"]; found {
    t.ErrorCount
    .WithLabelValues(t.component)
    .Inc()
  }
}
```

Help us instrument the world

- Network Libraries and service clients
- Frameworks and runtimes
- OpenTracing multiplexers
- An OpenTracing → Prometheus bridge
- Kubernetes + OpenTracing
- OpenTracing specification itself
- Gitter: gitter.im/opentracing/public
- Github: github.com/opentracing

Distributed Tracing Salon 2017

Free Donuts. Thursday, 2:00 pm–3:20 pm. Room A08

**also, Tracing 101 (interactive), Tracing Group Therapy, Tracing + k8s, and more!*

OPENTRACING

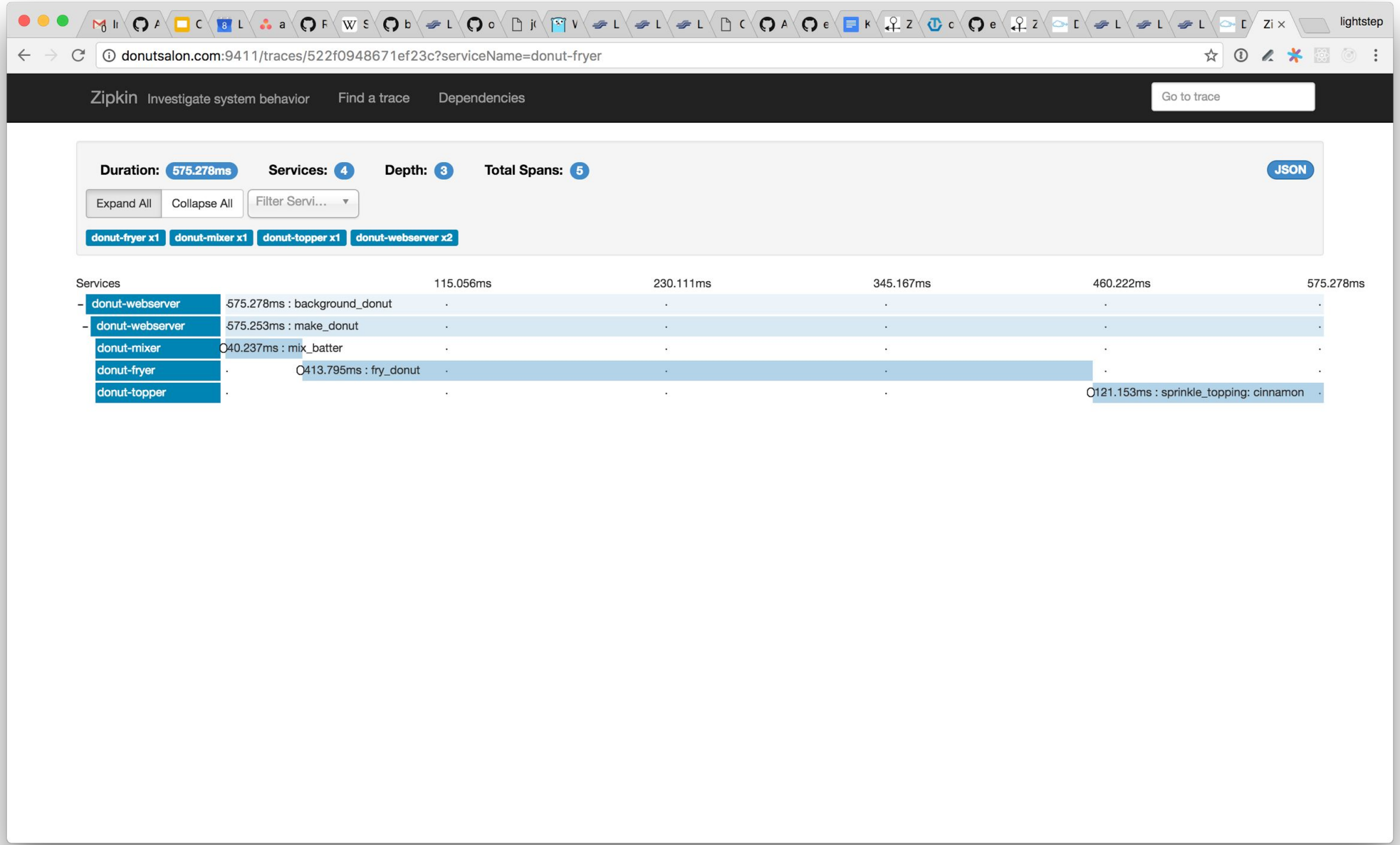
Thanks / Q&A

… and please be in touch

Ted Young
ted@lightstep.com / @tedsuooo

@opentracing

(Appendix Slides)

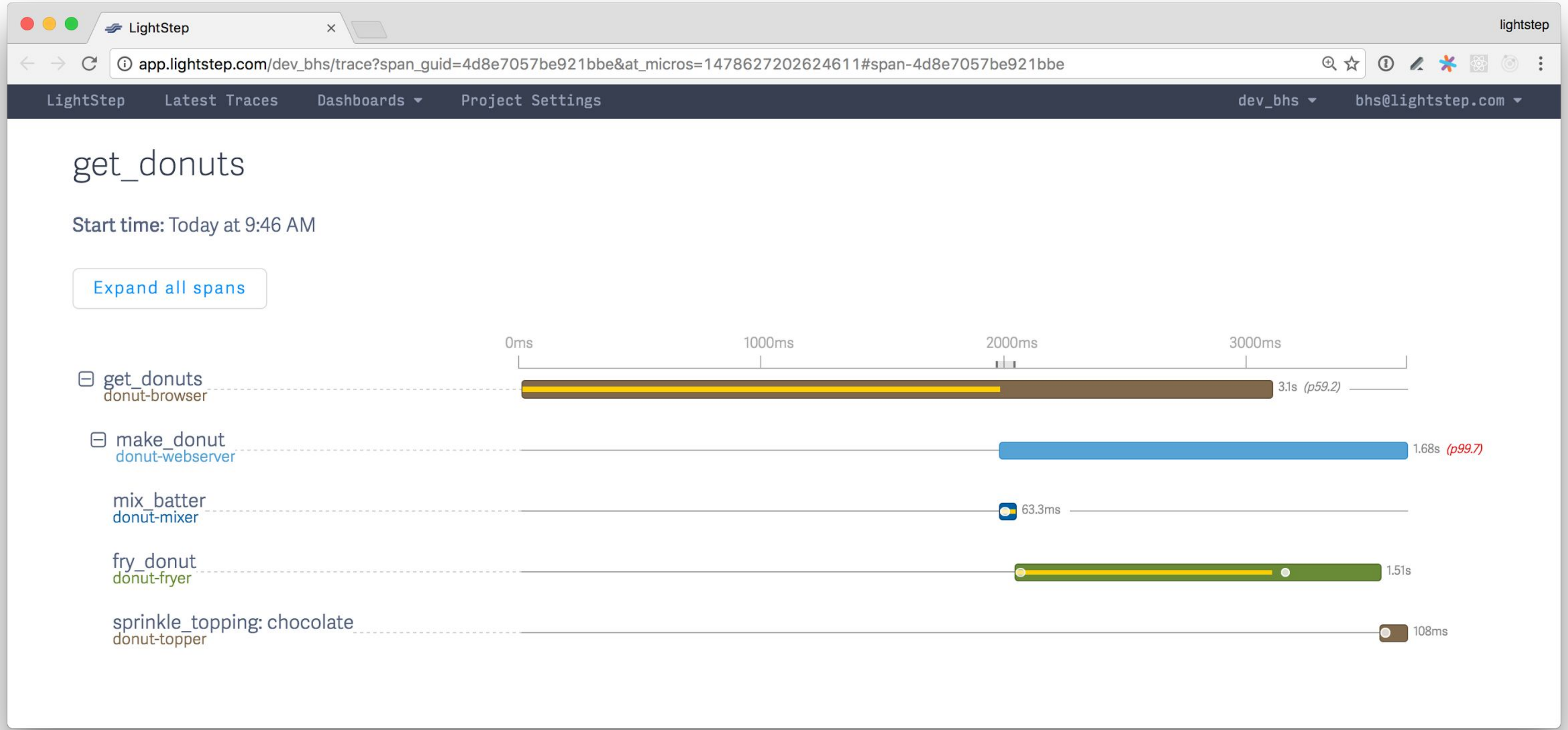


donutsalon.com:9411/traces/522f0948671ef23c?serviceName=donut-fryer

Duration: 575.278ms Services: 4 Depth: 3 Total Spans: 5

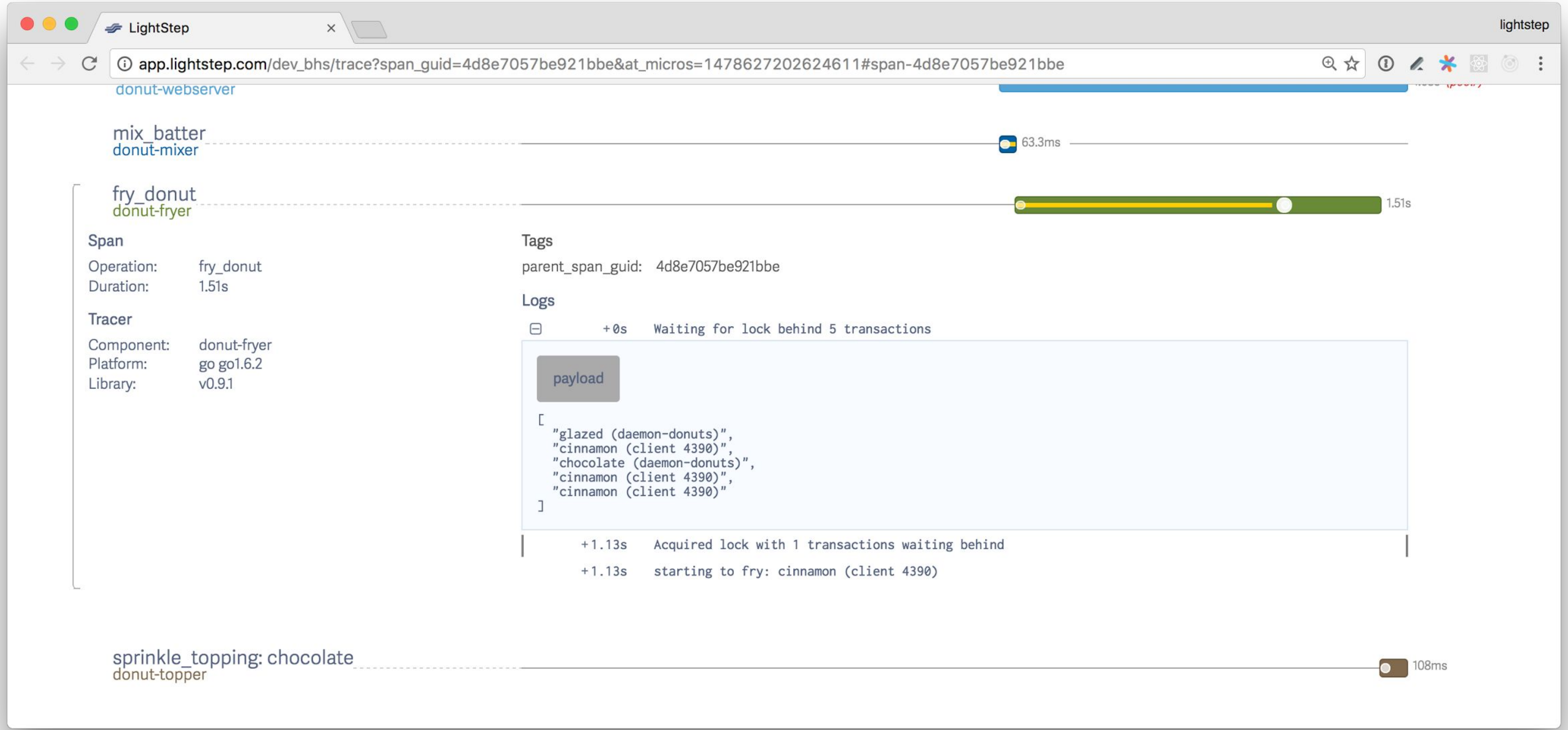
donut-fryer x1 donut-mixer x1 donut-topper x1 donut-webserver x2

```
Services            115.056ms   230.111ms   345.167ms   460.222ms   575.278ms
- donut-webserver   575.278ms : background_donut
 - donut-webserver  575.253ms : make_donut
   donut-mixer      40.237ms : mix_batter
   donut-fryer      413.795ms : fry_donut
   donut-topper     121.153ms : sprinkle_topping: cinnamon
```


LightStep
app.lightstep.com/dev_bhs/trace?span_guid=4d8e7057be921bbe&at_micros=1478627202624611#span-4d8e7057be921bbe
lightstep
LightStep
Latest Traces
Dashboards
Project Settings
dev_bhs
bhs@lightstep.com
Expand all spans

get_donuts

Start time: Today at 9:46 AM


LightStep
lightstep
app.lightstep.com/dev_bhs/trace?span_guid=4d8e7057be921bbe&at_micros=1478627202624611#span-4d8e7057be921bbe
donut-webserver
mix_batter
donut-mixer
63.3ms
fry_donut
donut-fryer
1.51s

Span

Operation:	fry_donut
Duration:	1.51s

Tracer

Component:	donut-fryer
Platform:	go go1.6.2
Library:	v0.9.1

Tags

parent_span_guid: 4d8e7057be921bbe

Logs

```
+0s      Waiting for lock behind 5 transactions
```

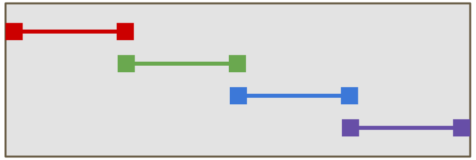
payload

```
[
  "glazed (daemon-donuts)",
  "cinnamon (client 4390)",
  "chocolate (daemon-donuts)",
  "cinnamon (client 4390)",
  "cinnamon (client 4390)"
]
```

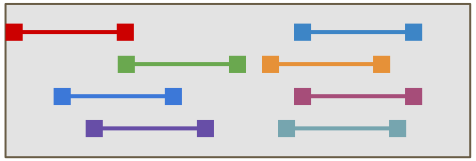
```
+1.13s   Acquired lock with 1 transactions waiting behind
+1.13s   starting to fry: cinnamon (client 4390)
```

Concurrency in Pictures

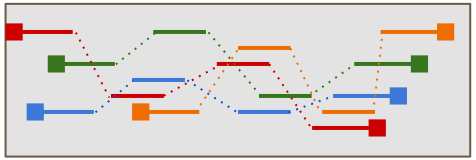
“The Simple [Inefficient] Thing”



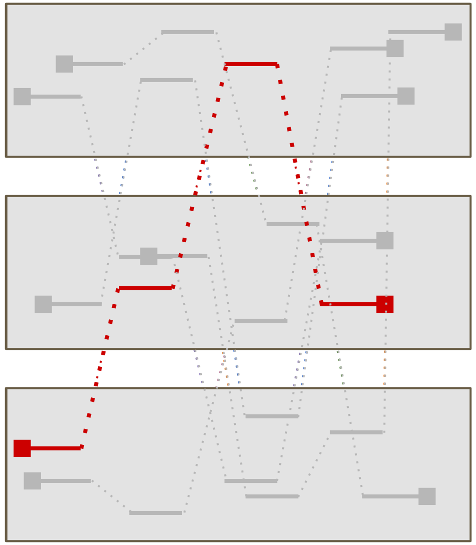
Basic Concurrency



Async Concurrency



Distributed Concurrency



The OpenTracing data model

Tracer	Span	SpanContext
Typically one per process StartSpan(): where every Span begins 0 or more “References” (e.g., parents), identified via SpanContexts Injecting SpanContexts into “carrier” propagators Extracting SpanContexts from “carrier” propagators	Start and Finish timestamps Zero or more key:value “tags” (usually for filtering and/or aggregation) Zero or more timestamped key:value logs (usually for, well, logging) Set/Get Baggage(*) Get SpanContext	TL;DR: the “nodes” in the DAG Read-only access to Baggage Mostly opaque; this is where implementations store span_id, etc (No timestamps!)

IPC propagation <u>without</u> tight coupling!

Instrumentation: Wrap an IPC data structure with an OpenTracing “carrier”

```
    carrier := opentracing.HTTPHeadersCarrier(httpReq.Header)
```

Instrumentation: Pass a SpanContext and the carrier to Inject()

```
    tracer.Inject(currentSpan.context(), opentracing.HTTPHeaders, carrier)
```

Tracer Impl: Confirm the type of the SpanContext

```
    zipkinSpanContext, ok := (ZipkinSpanContext)SpanContext
```

Tracer Impl: Use the Inject() format to determine how to encode data in the carrier

```
    if format == opentracing.HTTPHeaders {
        carrier.Put(“X-B3-TraceId”, zipkinSpanContext.HexTraceId())
        … etc …
    }
```

OPENTRACING

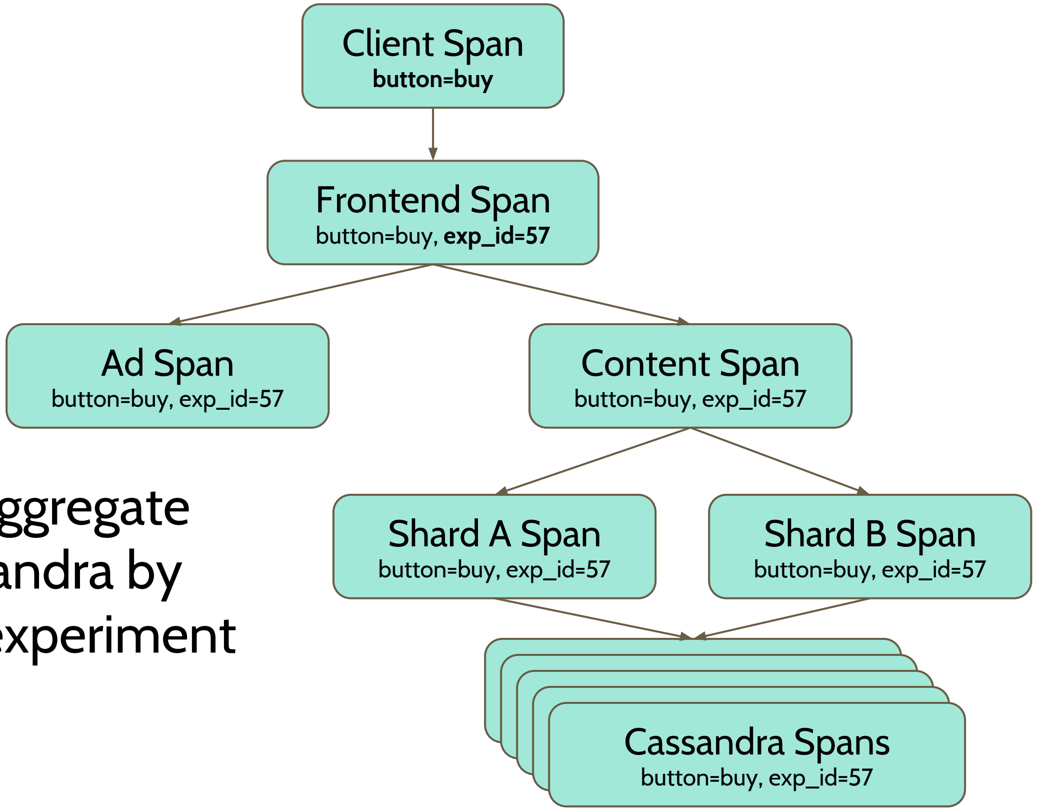
Pick your battles

OpenTracing scope

	Standard instrumentation APIs for...			Standard encoding formats for...	
Benefit / Feature enabled by standardization	(1) span management	(2) inter-process propagation	(3) active span management	(4) in-band context encoding	(5) out-of-band trace data
Tracing API consistency across platforms	**Required**	**Required**	Helpful	N/A	N/A
Keep instrumentation deps small for OSS projects	**Required**	**Required**	N/A	N/A	N/A
Avoid lock-in: easily switch all services from tracing vendor A to tracing vendor B	**Required**	**Required**	Helpful	N/A	Helpful

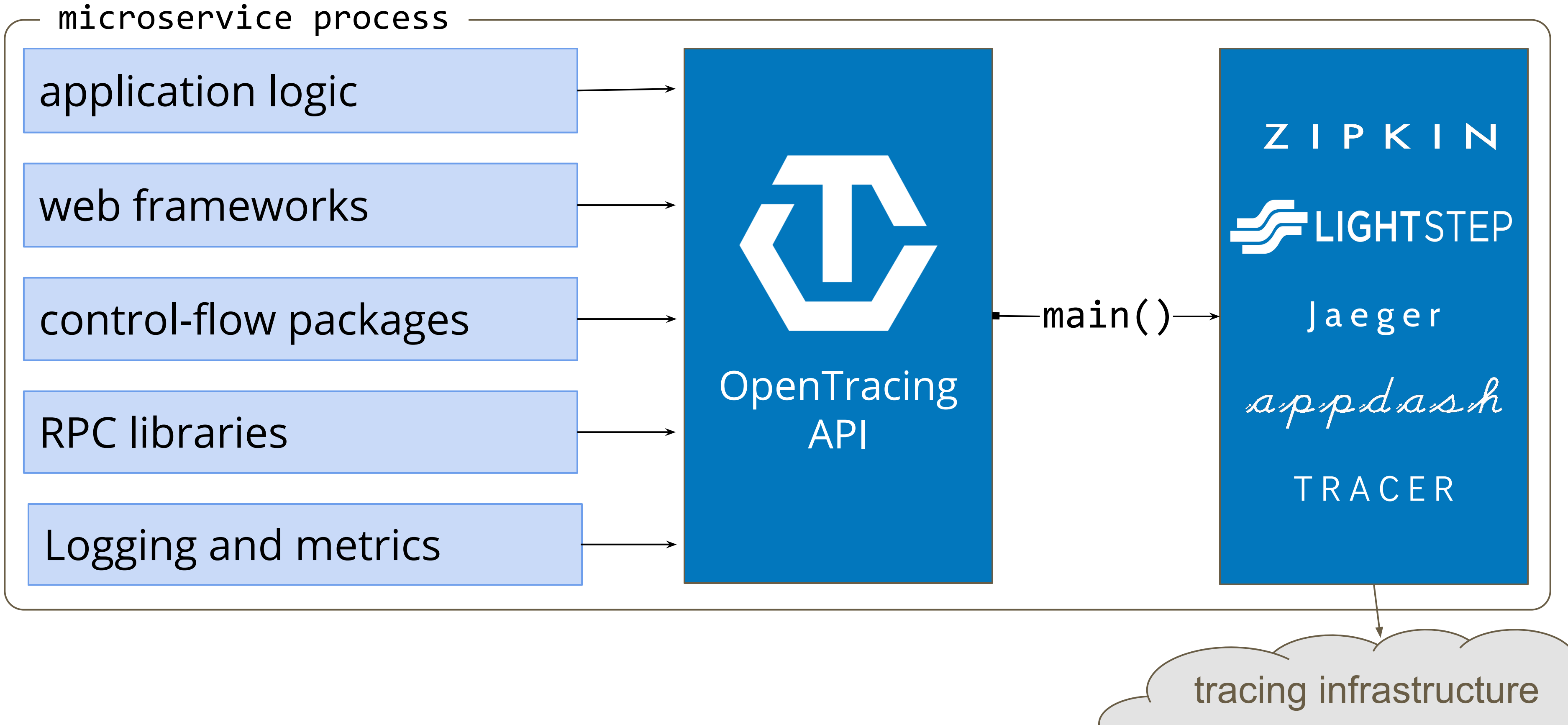
OPENTRACING

More about Baggage (see the PivotTracing paper)



Problem: how to aggregate disk writes in Cassandra by "button" type (or experiment id, etc, etc)?

OpenTracing architecture



microservice process
application logic
web frameworks
control-flow packages
RPC libraries
Logging and metrics
OpenTracing API
main()
ZIPKIN
LIGHTSTEP
Jaeger
appdash
TRACER
tracing infrastructure