How to Properly Blame Things for Causing Latency

An introduction to Distributed Tracing and Zipkin

Adrian Cole, Pivotal
@adrianfcole

Introduction

introduction
understanding latency
distributed tracing
zipkin
demo
wrapping up

@adrianfcole

spring cloud at pivotal
focused on distributed tracing
helped open zipkin

Understanding Latency

introduction
understanding latency
distributed tracing
zipkin
demo
wrapping up

Understanding our architecture

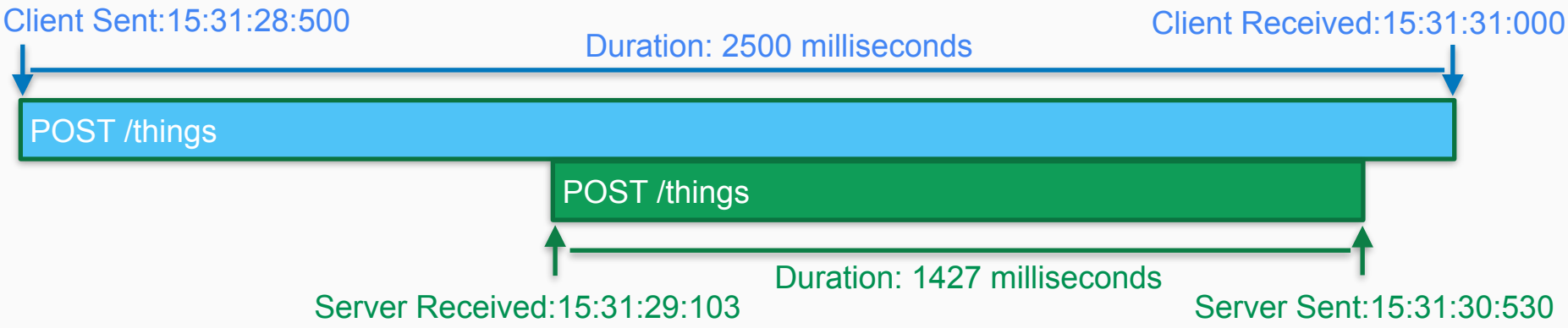
Microservice and data pipeline **architectures are a often a graph of components**, distributed across a network.

A **call graph or data flow can become delayed** or fail due to the nature of the operation, components, or edges between them.

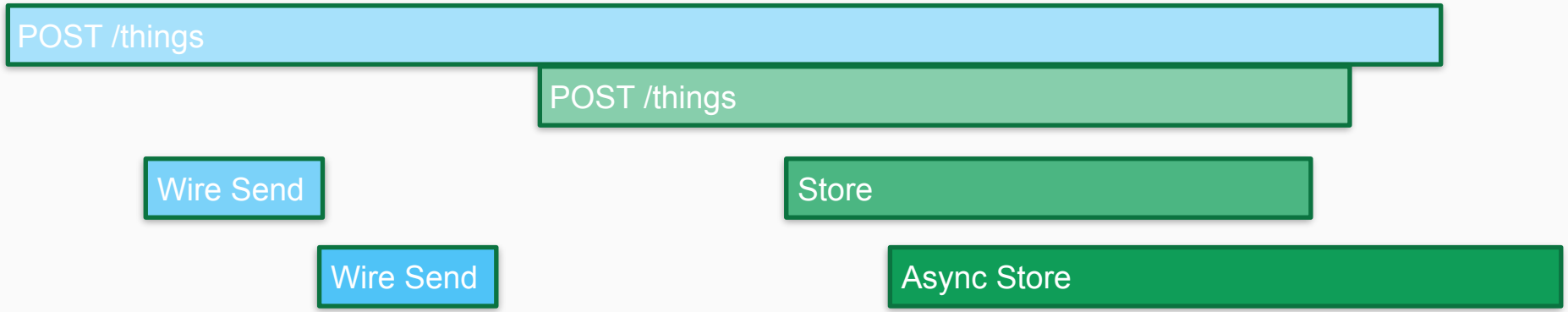
We want to **understand our current architecture** and troubleshoot latency problems, **in production**.

Why is POST /things slow?

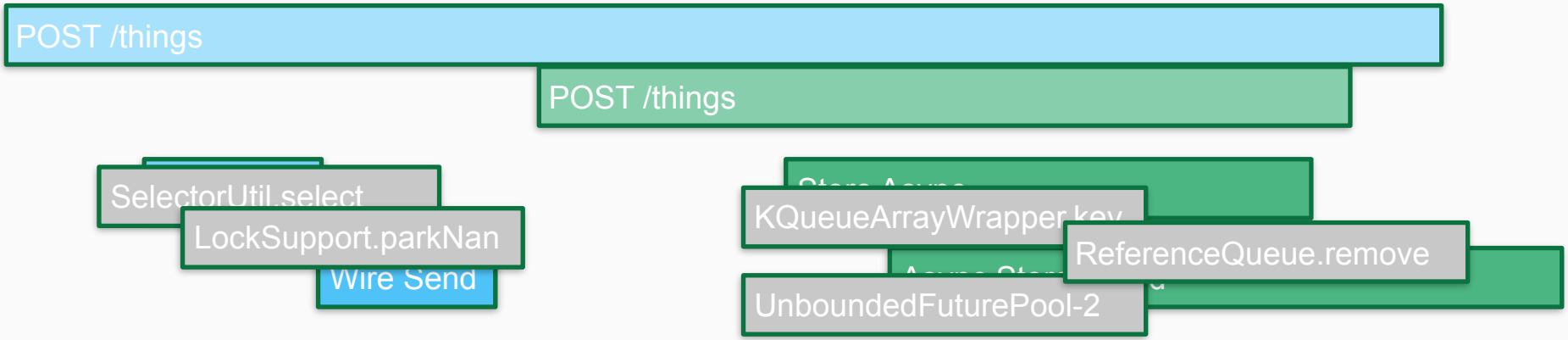
There’s often two sides to the story



and not all operations are on the critical path

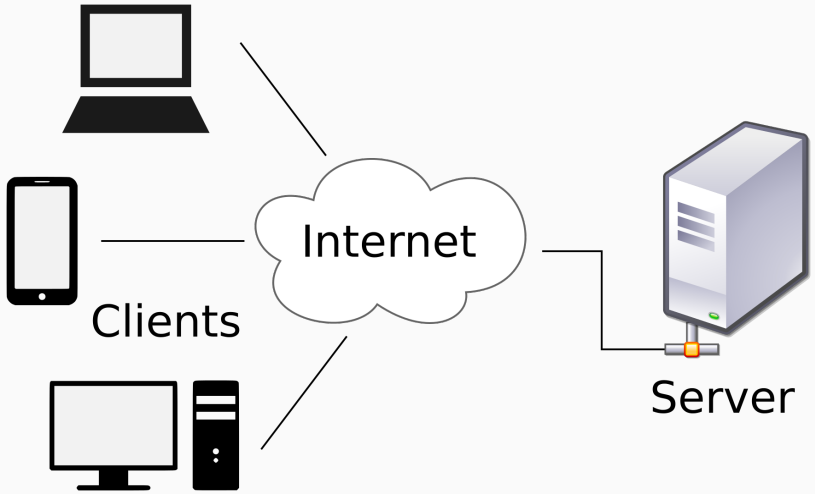


and not all operations are relevant



Service architecture isn’t this simple anymore

Single-server scenarios aren’t realistic or don’t fully explain latency.

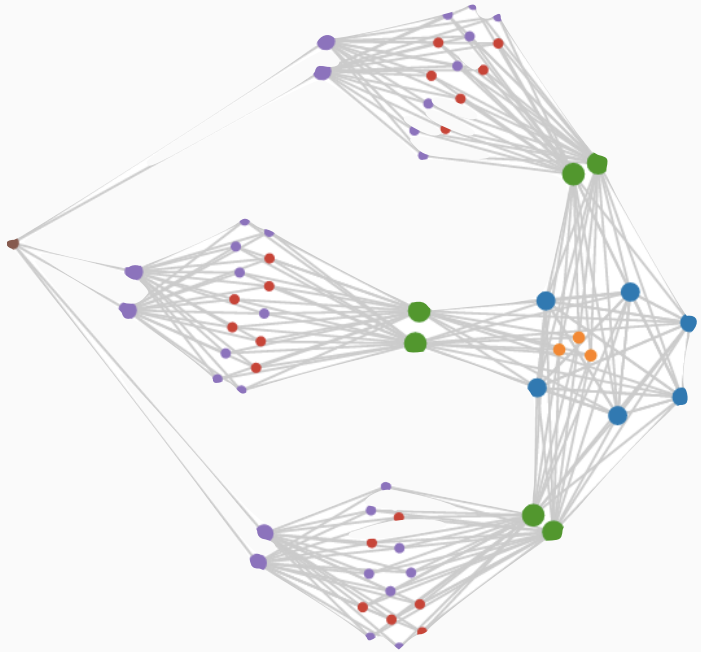


David Vignoni Gnome-fs-server.svg

Can we make troubleshooting wizard-free?

We no longer need wizards to deploy complex architectures.

We shouldn’t need wizards to troubleshoot them, either!



Distributed Tracing

introduction
understanding latency
distributed tracing
zipkin
demo
wrapping up

Distributed Tracing commoditizes knowledge

Distributed tracing systems collect end-to-end latency graphs (traces) in near real-time.

You can compare traces to understand why certain requests take longer than others.

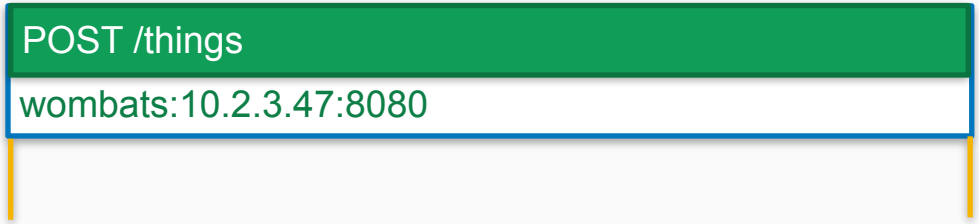
Distributed Tracing Vocabulary

A **Span** is an individual operation that took place. A span contains **timestamped events** and **tags**.

A **Trace** is an end-to-end latency graph, composed of spans.

A Span is an individual operation

Operation



Events

Server Received **Server Sent**

Tags

peer.ipv4	1.2.3.4
http.request-id	abcd-ffe
http.request.size	15 MiB
http.url	…&features=HD-uploads

Tracing Systems are Observability Tools

Tracing systems collect, process and present data reported by tracers.

- aggregate spans into trace trees

- provide query and visualization focused on latency

- have retention policy (usually days)

ProTip: Tracing is not just for latency

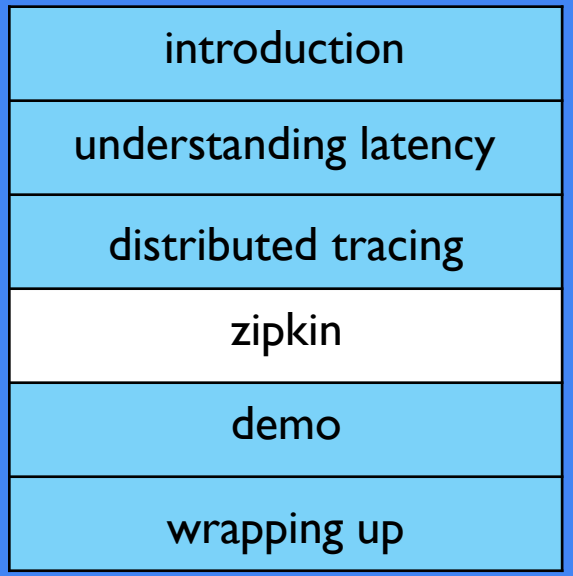
Some wins unrelated to latency

- Understand your architecture

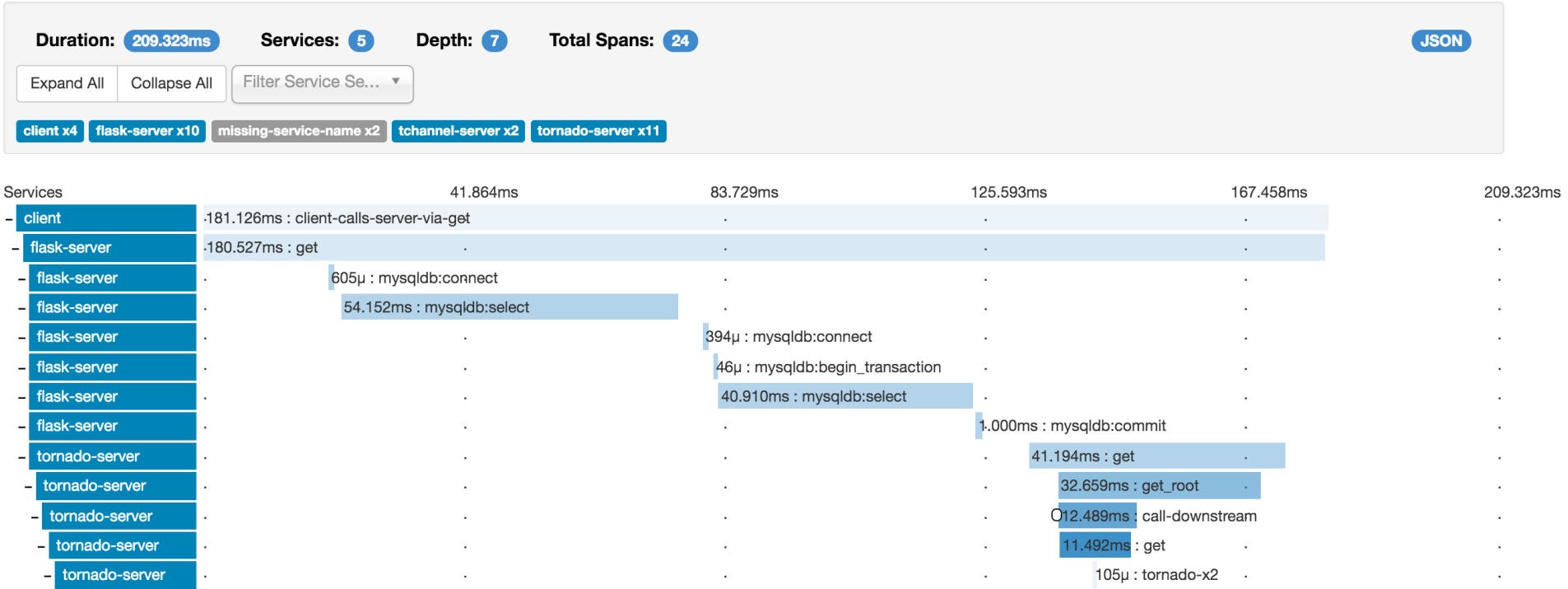
- Find services that aren’t used

- Reduce time spent on triage

Zipkin


introduction
understanding latency
distributed tracing
zipkin
demo
wrapping up

Zipkin is a distributed tracing system



Zipkin lives in GitHub

Zipkin was created by Twitter in 2012. In 2015, OpenZipkin became the primary fork.

OpenZipkin is an org on GitHub. It contains tracers, OpenApi spec, service components and docker images.

https://github.com/openzipkin

Zipkin Architecture

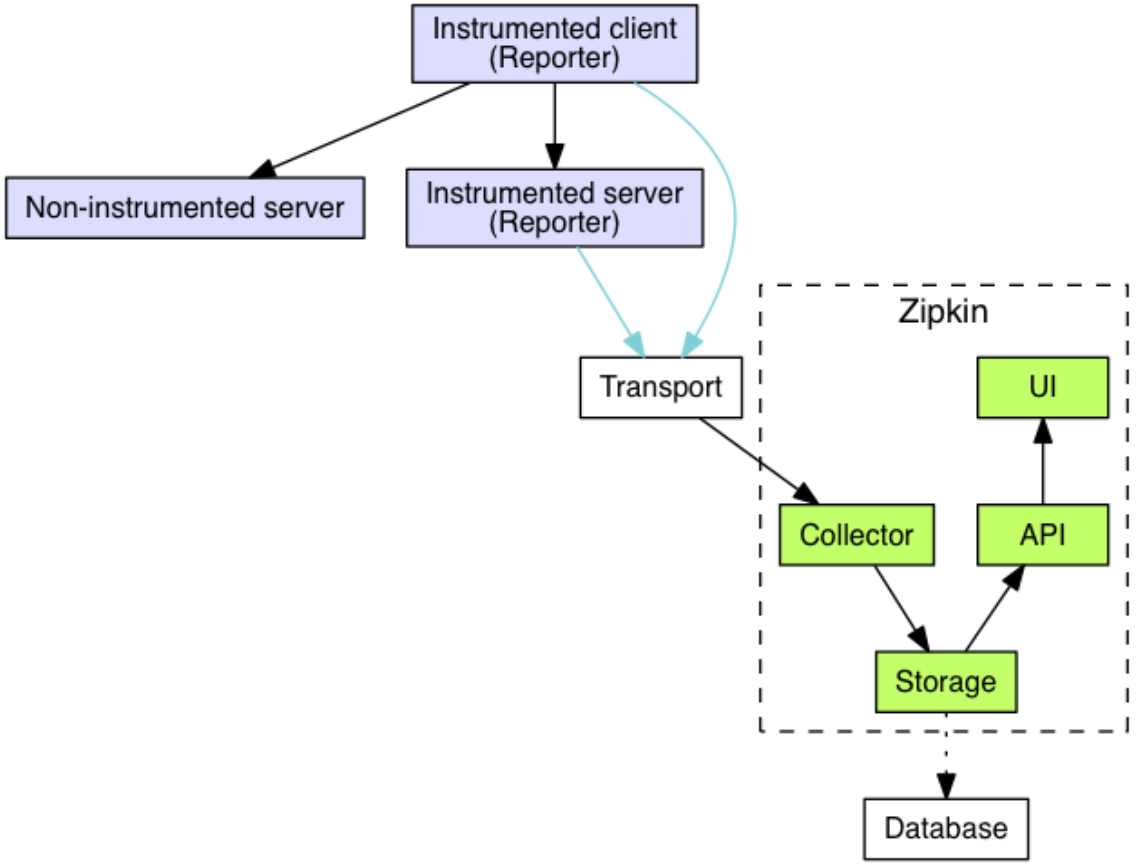
Tracers **report** spans HTTP or Kafka.

Servers **collect** spans, storing them in MySQL, Cassandra, or Elasticsearch.

Users **query** for traces via Zipkin's Web UI or Api.

Platform frameworks for Zipkin:

Bosh (Cloud Foundry)
Docker (in Zipkin's org)
Kubernetes
Mesos



Zipkin has starter architecture

Tracing is new for a lot of folks.

For many, the MySQL option is a good start, as it is familiar.

```
services:
  storage:
    image: openzipkin/zipkin-mysql
    container_name: mysql
    ports:
      - 3306:3306
  server:
    image: openzipkin/zipkin
    environment:
      - STORAGE_TYPE=mysql
      - MYSQL_HOST=mysql
    ports:
      - 9411:9411
    depends_on:
      - storage
```

Zipkin can be as simple as a single file

```
$ curl -SL 'https://search.maven.org/remote_content?g=io.zipkin.java&a=zipkin-server&v=LATEST&c=exec' > zipkin.jar
$ SELF_TRACING_ENABLED=true java -jar zipkin.jar

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v1.4.0.RELEASE)

2016-08-01 18:50:07.098  INFO 8526 --- [           main] zipkin.server.ZipkinServer               : Starting ZipkinServer on acole
with PID 8526 (/Users/acole/oss/sleuth-webmvc-example/zipkin.jar started by acole in /Users/acole/oss/sleuth-webmvc-example)
-snip-
```

```
$ curl -s localhost:9411/api/v1/services|jq .
[
  "zipkin-server"
]
```

Demo

introduction
understanding latency
distributed tracing
zipkin
demo
wrapping up

Distributed Tracing across Spring Boot apps

Two Spring Boot (Java) services collaborate over http.

Zipkin will show how long the whole operation took, as well how much time was spent in each service.

https://github.com/openzipkin/sleuth-webmvc-example

Spring Cloud Sleuth

Java

Web requests in the demo are served by Spring MVC controllers. Tracing of these are automatically performed by Spring Cloud Sleuth.

Spring Cloud Sleuth reports to Zipkin via HTTP by depending on spring-cloud-sleuth-zipkin.

https://cloud.spring.io/spring-cloud-sleuth/

Wrapping Up

introduction
understanding latency
distributed tracing
zipkin
demo
wrapping up

Wrapping up

Start by sending traces directly to a zipkin server.

Grow into fanciness as you need it: sampling, streaming, etc

Remember you are not alone!

gitter.im/openzipkin/zipkin

gitter.im/spring-cloud/spring-cloud-sleuth