Motivation

- Today, big systems are often replaced with microservice architectures.
- Microservices bring a lot of benefits: scalability, low coupling, seamless updates, etc.
- Yet, some problems, easily solvable in the old paradigm, become much more complex in microservice architectures.
  - How to determine a root cause of an error?
  - How to measure performance of a distributed system?
Tracing

• Tracing is an attempt to solve these problems
• Many different tracers are available:
  – Zipkin, Jaeger, LightStep and many others
• Tracers help to see the causal relationships between different processes in a distributed system
• Tracers also serve as distributed loggers
Different available tracers have different APIs. Yet similar semantics.

Using that API in your code makes you dependent on a particular tracer provider.

OpenTracing tries to mitigate this problem:
- A common API.
- A common unifying set of concepts.
- Tracer implementation can be easily changed.
Traces and Spans

- Spans represent separate operations: contiguous segments of work
  - e.g. a remote procedure call
- A trace is a collection of causally related spans
  - More precisely a directed acyclic graph of spans
- Two main types of relationships between spans:
  - ChildOf
  - FollowsFrom
Spans as objects

- A span contains:
  - An operation name
  - Start and end timestamps
  - A set of span tags
    - Key:Value pairs
  - A set of span logs
    - Timestamped Key:Value pairs;
      - Represent logged events
    - References to related spans (via their SpanContexts)
- SpanContext contains the information about span that crosses process boundaries.
```python
import opentracing

def say_hello(hello_to):
    hello_str = 'Hello, %s!' % hello_to
    print(hello_str)

tracer = opentracingtracer

def say_hello(hello_to):
    span = tracer.start_span('say-hello')
    hello_str = 'Hello, %s!' % hello_to
    print(hello_str)
    span.finish()

from lib.tracing import init_tracer

tracer = init_tracer('hello-world')

def say_hello(hello_to):
    with tracer.start_span('say-hello') as span:
        span.set_tag('hello-to', hello_to)
        hello_str = 'Hello, %s!' % hello_to
        span.log_kv({'event': 'string-format', 'value': hello_str})
        print(hello_str)
        span.log_kv({'event': 'println'})
```