

INTRO

---

OPENTRACING

# ALWAYS START WITH WHY

- debugging and monitoring distributed software architectures
- debug and optimize code (performance & latency)
- root cause analysis
- service dependency analysis
- distributed context propagation



# PLAN

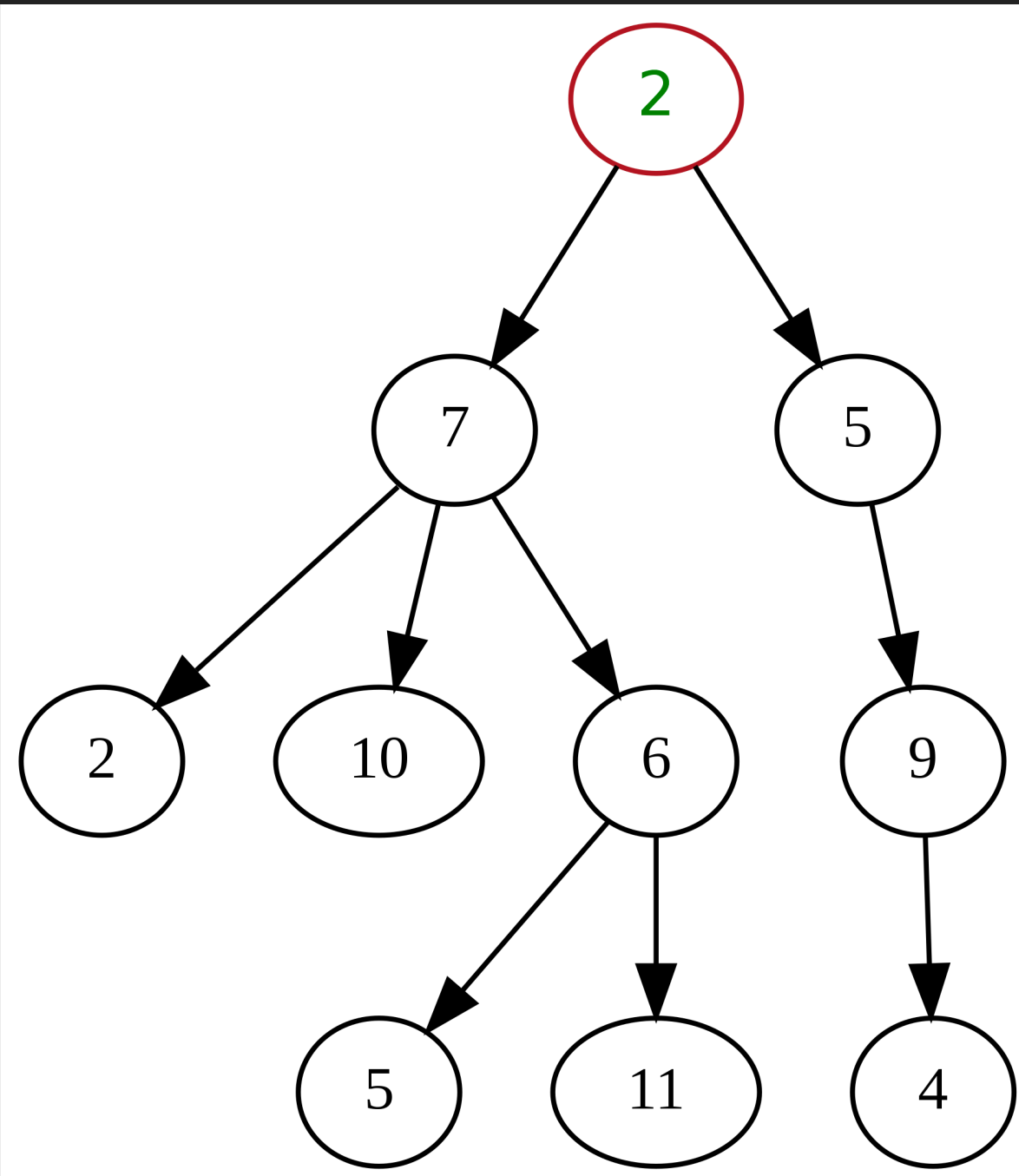
- ▶ meet the subject (theory)
- ▶ show demo (practice)
- ▶ set up server for tracing
- ▶ with interested teams apply & test



# SPECS

- ▶ what is "trace" ?

**Trace** can be thought of as a directed acyclic graph (DAG) of **Spans**.

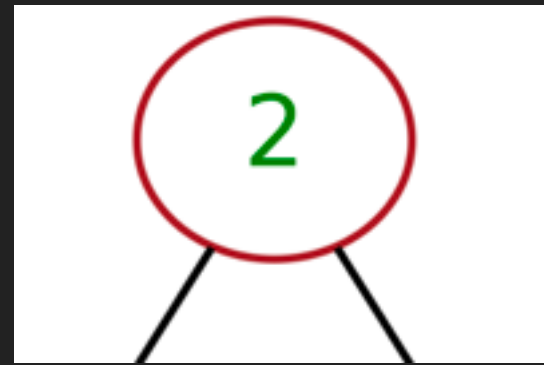
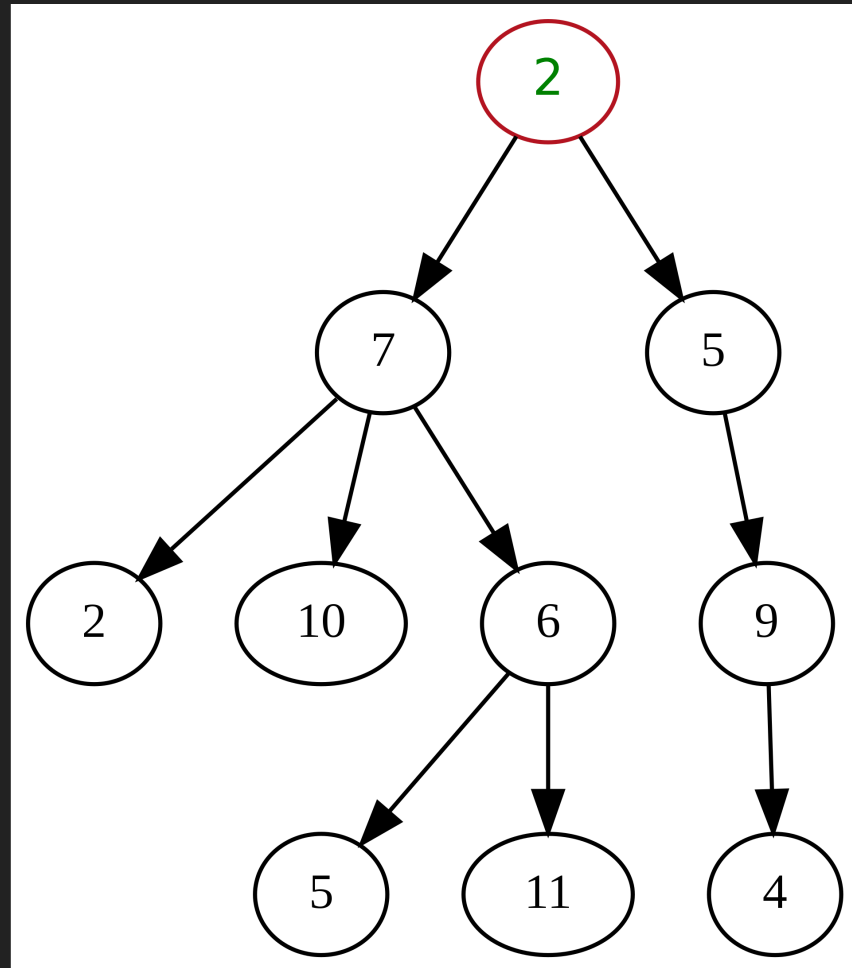


```
Emulator Nexus_6_API_28 Andr... com.example.android.architecture. Error
2019-01-24 20:25:12.755 3690-3690/com.example.android.architecture.blueprints.todomvp.mock E/AndroidRuntime: FATAL EXCEPTION: main
Process: com.example.android.architecture.blueprints.todomvp.mock, PID: 3690
java.lang.NullPointerException: Attempt to invoke virtual method 'boolean com.todoapp.data.Task.isEmpty()' on a null object reference
at com.todoapp.addedittask.AddEditTaskPresenter.createTask(AddEditTaskPresenter.java:142)
at com.todoapp.addedittask.AddEditTaskPresenter.saveTask(AddEditTaskPresenter.java:90)
at com.todoapp.addedittask.AddEditTaskFragment.lambda$onActivityCreated$0(AddEditTaskFragment.java:75)
at com.todoapp.addedittask.-$$Lambda$AddEditTaskFragment$8jpbPuFB8HguKQEvRjrqa-yQ4QE.onClick(Unknown Source:2)
at android.view.View.performClick(View.java:6597)
at android.view.View.performClickInternal(View.java:6574)
at android.view.View.access$3100(View.java:778)
at android.view.View$PerformClick.run(View.java:25885)
at android.os.Handler.handleCallback(Handler.java:873)
at android.os.Handler.dispatchMessage(Handler.java:99)
at android.os.Looper.loop(Looper.java:193)
at android.app.ActivityThread.main(ActivityThread.java:6669) <1 internal call>
at com.android.internal.os.RuntimeInit$MethodAndArgsCaller.run(RuntimeInit.java:493)
at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:858)


TODO CheckStyle Terminal Build 6: Logcat Profiler 3: Find 4: Run
```

# SPECS

## ► meet the "span"



**SPAN** - encapsulates following states:

- **name** (operation/function name)
- **start time**
- **finish time**
- set of [k => v] **tags**
- set of [k => v] **logs**
- **SpanContext** 
- **Refs** (child of, batch...)
- implementation dependent **state**
- **baggage items**

# SPECS

- ▶ meet the “tracer”



**Tracer** - defined by it's abilities (methods)

- **creates** span
- **injects** span into carrier
- **extracts** span from carrier

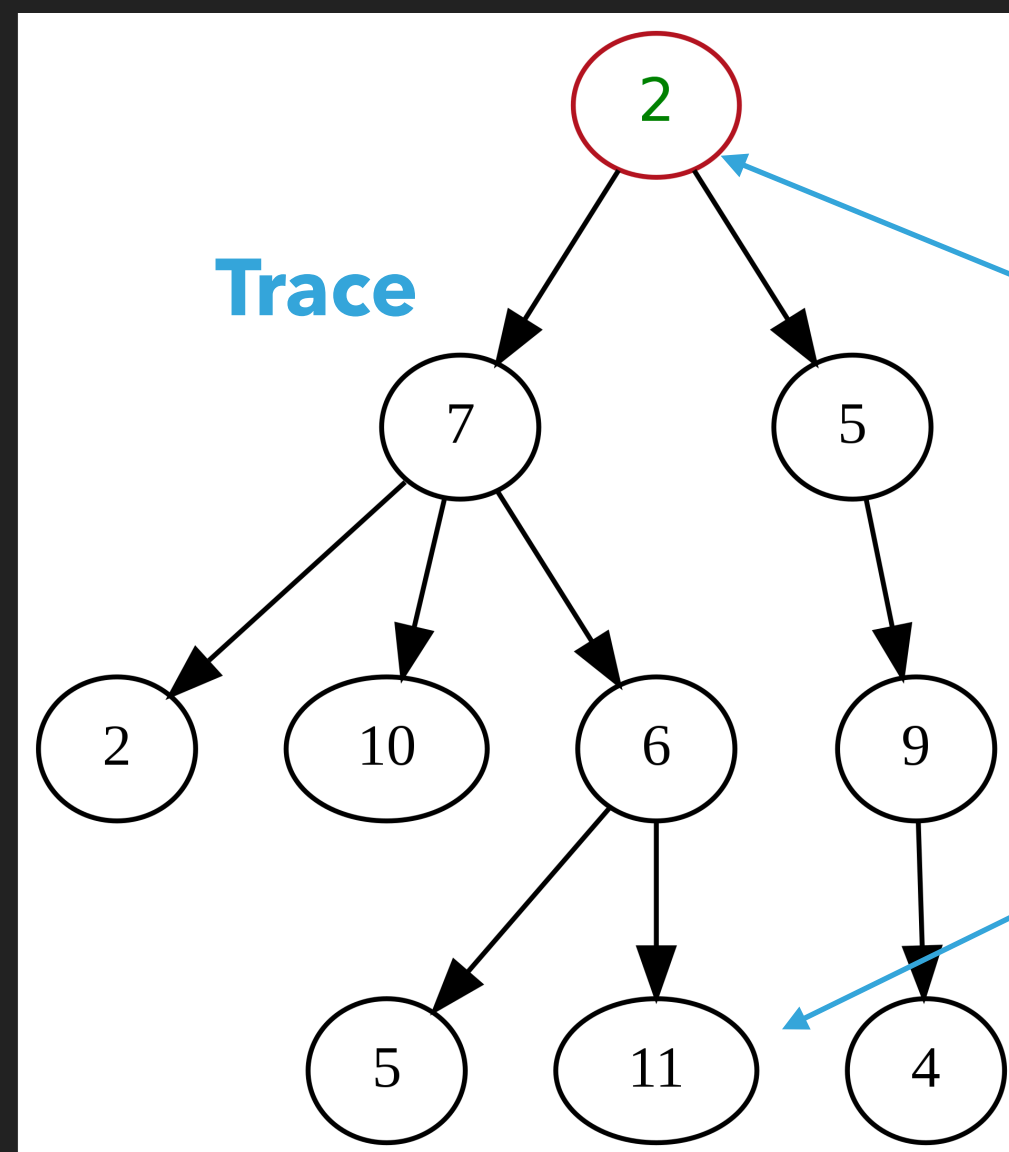


# SPECS

## TraceR (also known as GlobalTracer)



- **creates** span
- **injects** span into carrier
- **extracts** span from carrier



### Span

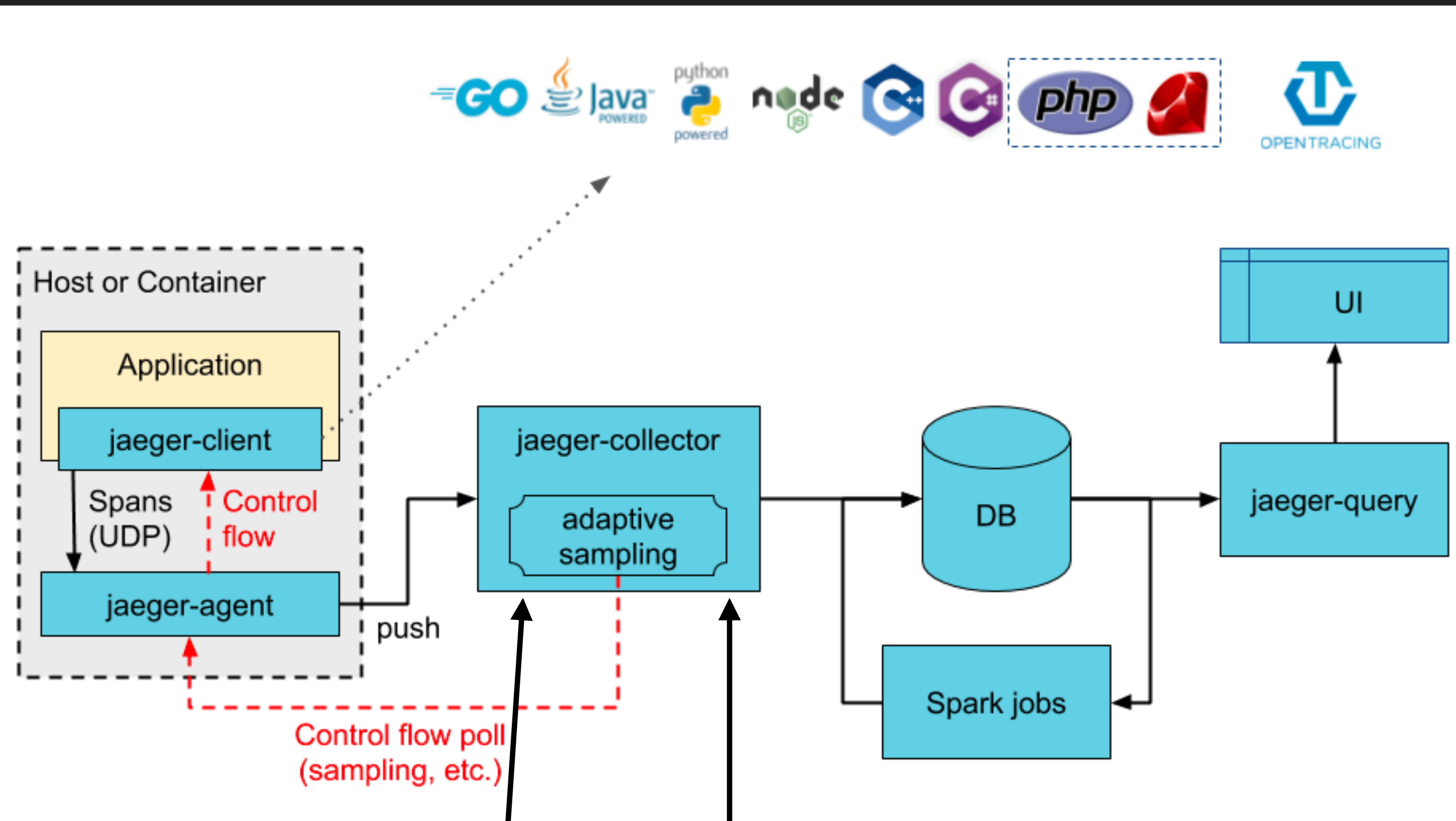
- **name** (operation/function name)
- **start time**
- **finish time**
- set of [k => v] **tags**
- set of [k => v] **logs**
- **SpanContext** →
  - implementation dependent **state**
  - **baggage items**
- **Refs** (child of, batch...)

DEMO TIME !





# JAEGER-ARCHITECTURE



**application:** any (our) application

**jaeger-client:** library to work with jaeger-api

**jaeger-agent:** daemon process that sends traces to...

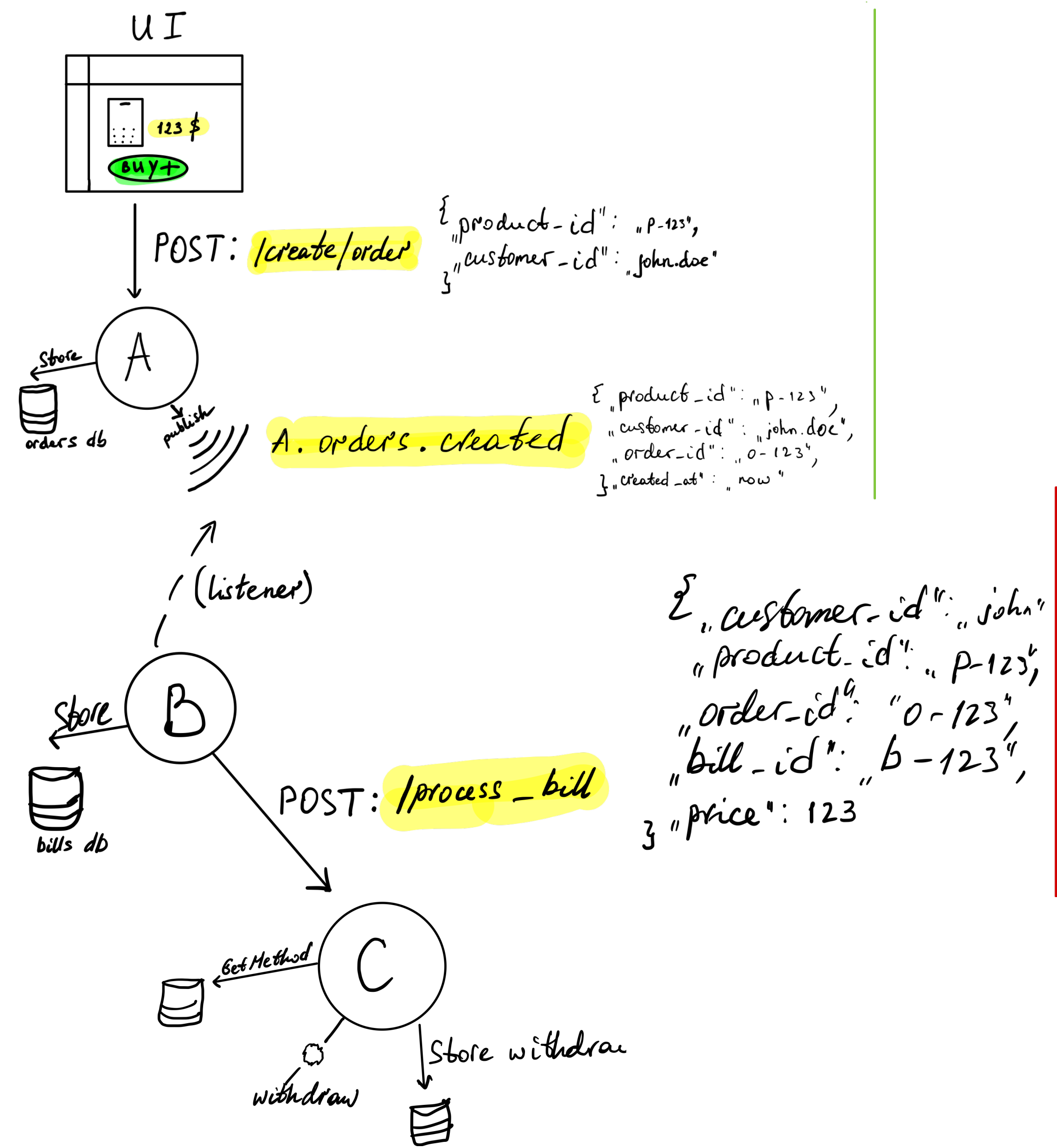
**jaeger-collector:** receives and stores spans

**jaeger-query:** serv. for fetching and viewing traces in **UI**

2) CONTAINER WITH ITS AGENT

3) CONTAINER WITH ITS AGENT

# SOA EXAMPLE





QUESTIONS ?



# LINKS

- <https://opentracing.io/specification>
- <https://www.jaegertracing.io>
- <https://medium.com/jaegertracing/embracing-context-propagation-7100b9b6029a>
- <https://github.com/yurishkuro/opentracing-tutorial>
- demo: [https://github.com/komron-m/opentracing\\_jaeger](https://github.com/komron-m/opentracing_jaeger)