# Observability before and after Service Mesh

# Agenda

- Microservices and the fallacies of distributed computing
- Observability
  - Logs, traces and metrics
  - How have we been solving this so far?
- Service Mesh
  - Data and Control plane
- Istio
  - Control plane components
- Demo

# About us!

## I am **Matheus Moraes**

Software Engineer @**sensedia**

Java, NoSQL and Microservices enthusiast

# whoami

## I am **Tiago Angelo**

Software Engineer @**sensedia**

Java, Microservices and Golang enthusiast

# Fallacies of distributed computing

- The network is reliable
- Latency is zero
- Bandwidth is infinite
- The network is secure
- Topology doesn't change
- There is one administrator
- Transport cost is zero
- The network is homogeneous

# Observability

**What is causing it?**

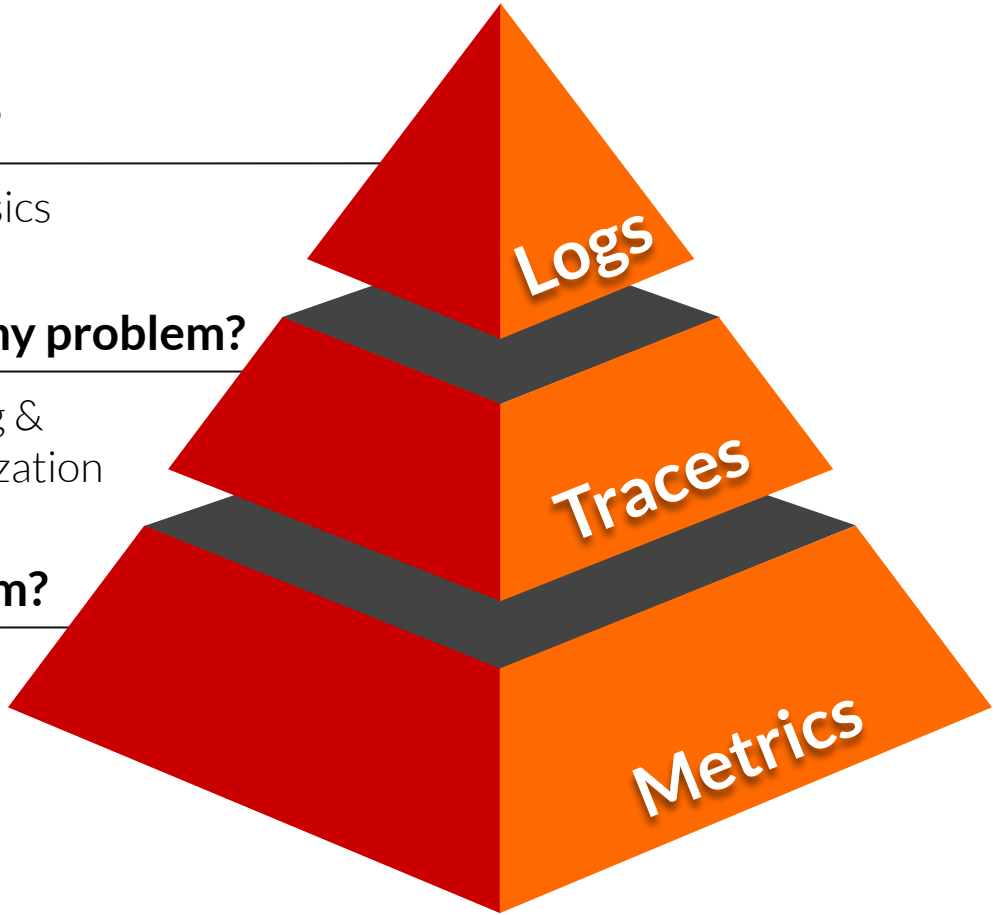
3 Root Cause & Forensics

**Where exactly is my problem?**

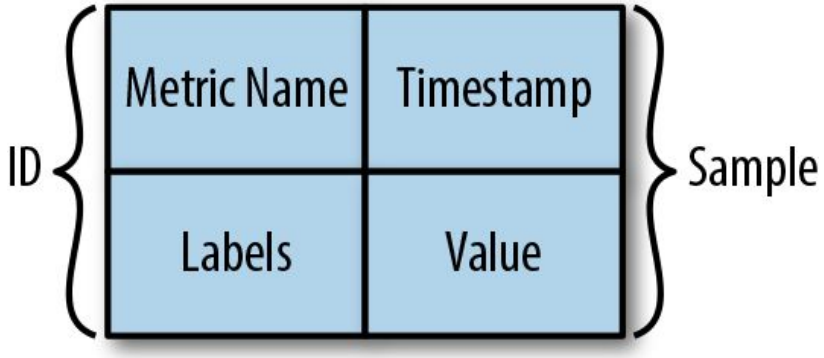
2 Cross-Service Debug & Performance Optimization

**Do I have a problem?**

1 Dashboarding, Trending & Problem Detection



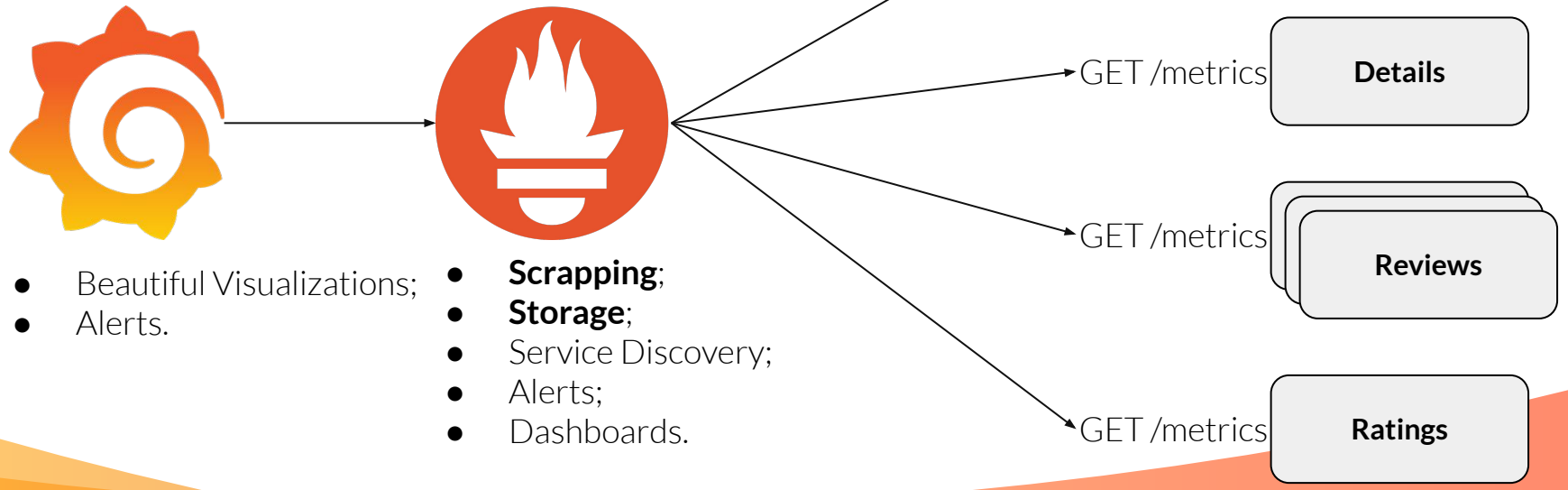
# Metrics




**R**ate
**E**rror
**D**uration

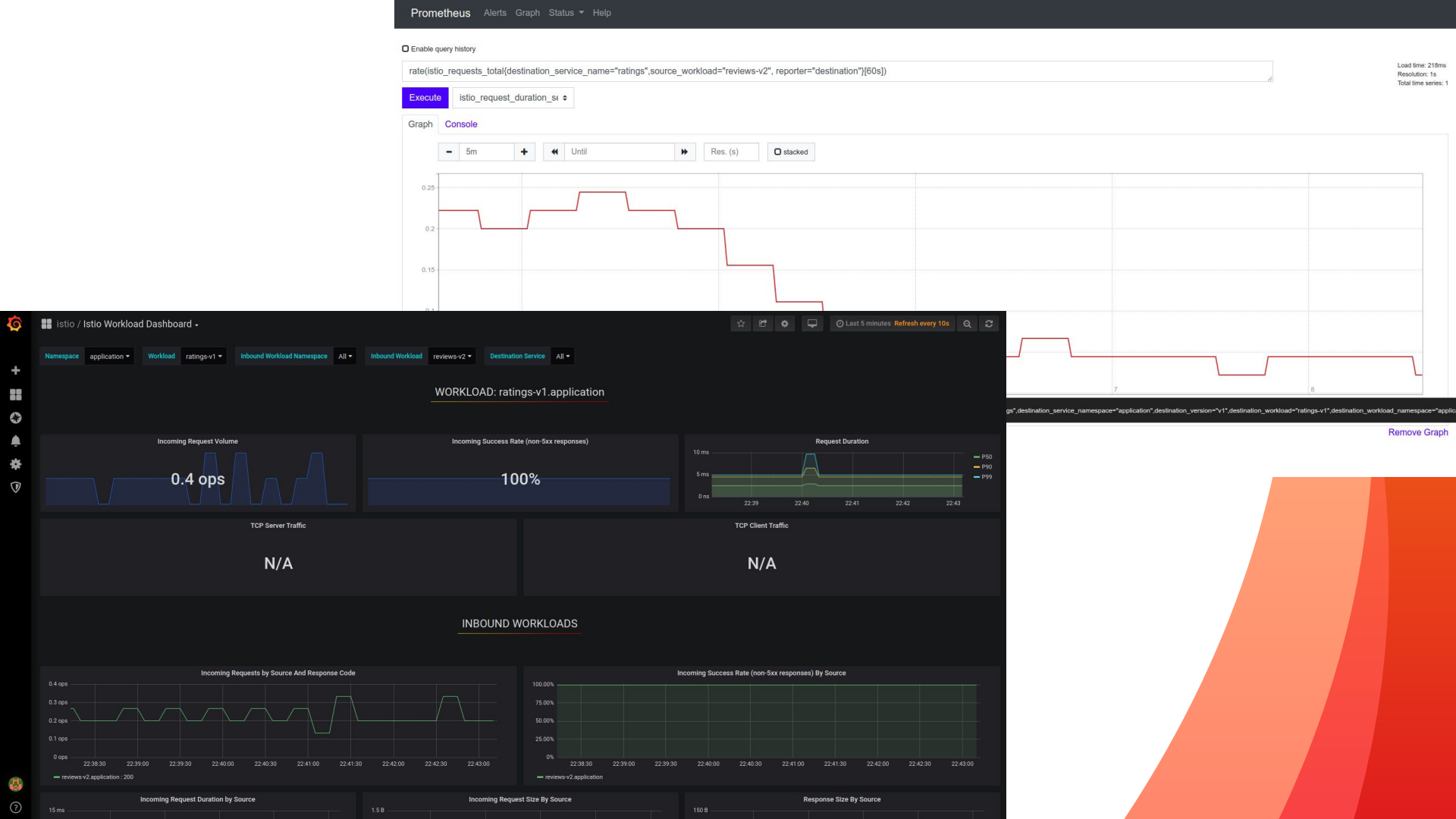
```
reviews_http_requests_total{env="prod",method="POST",code="200",type="infra"} 647.0 R
reviews_http_requests_total{env="prod",method="POST",code="400",type="infra"} 74.0 E
products_searches_category_total{env="prod",category="BOOK",type="business"} 152.0
ratings_http_request_seconds_count{env="prod",type="infra"} 77.0
ratings_http_request_seconds_sum{env="prod",type="infra"} 34.97 D
```

# Prometheus & Grafana




- Beautiful Visualizations;
- Alerts.

- **Scrapping**;
- **Storage**;
- Service Discovery;
- Alerts;
- Dashboards.


Prometheus
Alerts
Graph
Status
Help
Enable query history
rate(istio_requests_total{destination_service_name="ratings",source_workload="reviews-v2", reporter="destination"}[60s])
Load time: 218ms
Resolution: 1s
Total time series: 1
Execute
istio_request_duration_se
Graph
Console
5m
Until
Res. (s)
stacked
0.25
0.2
0.15
gs",destination_service_namespace="application",destination_version="v1",destination_workload="ratings-v1",destination_workload_namespace="applic
Remove Graph
istio / Istio Workload Dashboard
Last 5 minutes
Refresh every 10s
Namespace
application
Workload
ratings-v1
Inbound Workload Namespace
All
Inbound Workload
reviews-v2
Destination Service
All
WORKLOAD: ratings-v1.application
Incoming Request Volume
0.4 ops
Incoming Success Rate (non-5xx responses)
100%
Request Duration
10 ms
5 ms
0 ns
P50
P90
P99
22:39
22:40
22:41
22:42
22:43
TCP Server Traffic
N/A
TCP Client Traffic
N/A
INBOUND WORKLOADS
Incoming Requests by Source And Response Code
0.4 ops
0.3 ops
0.2 ops
0.1 ops
0 ops
22:38:30
22:39:00
22:39:30
22:40:00
22:40:30
22:41:00
22:41:30
22:42:00
22:42:30
22:43:00
reviews-v2.application : 200
Incoming Success Rate (non-5xx responses) By Source
100.00%
75.00%
50.00%
25.00%
0%
reviews-v2.application
Incoming Request Duration by Source
15 ms
Incoming Request Size By Source
1.5 B
Response Size By Source
150 B

# Spring Actuator & Micrometer

Just by adding the Spring dependency:

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
<dependency>
        <groupId>io.micrometer</groupId>
        <artifactId>micrometer-registry-prometheus</artifactId>
</dependency>
```

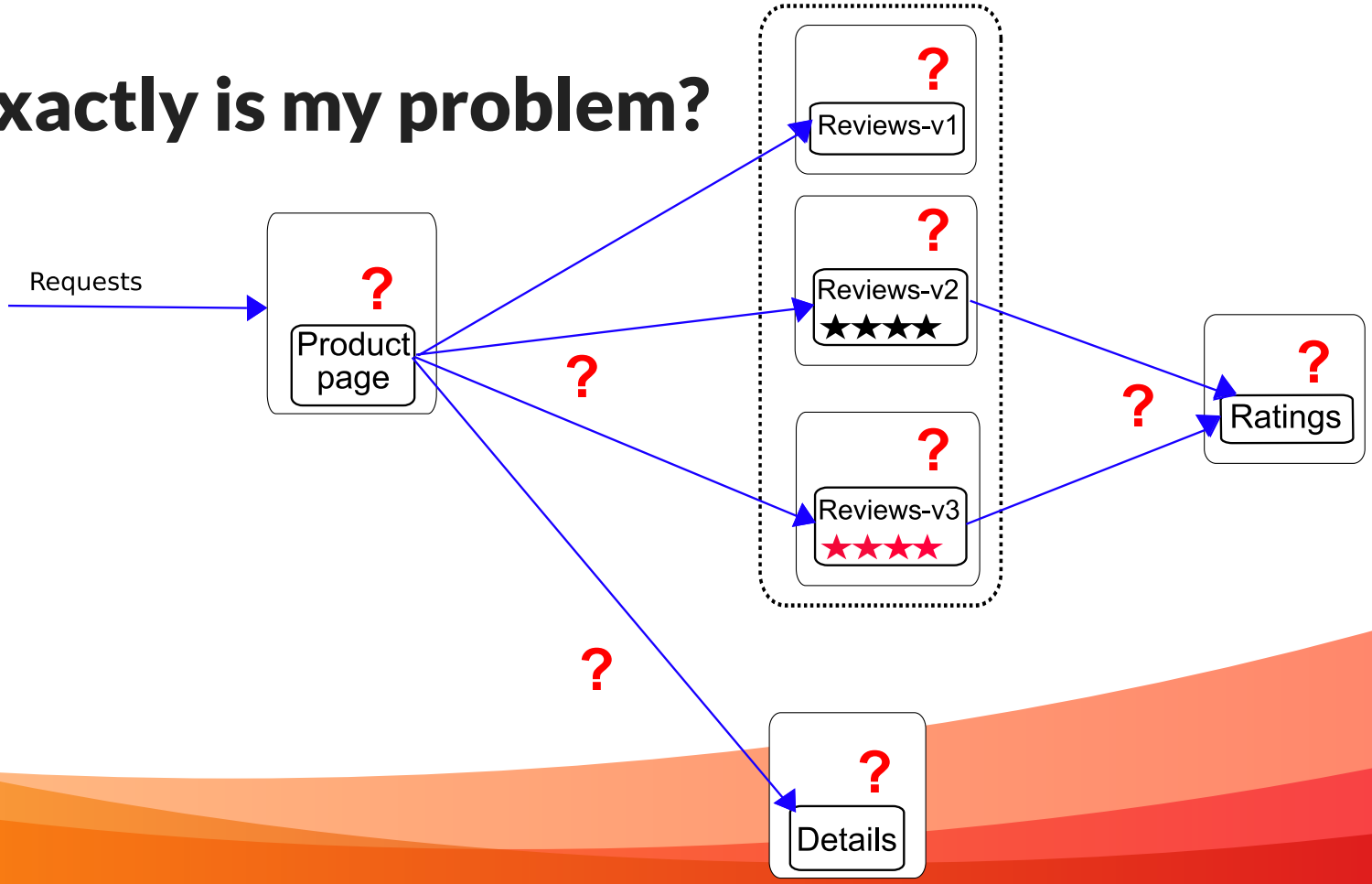
We automatically gain production ready endpoints:

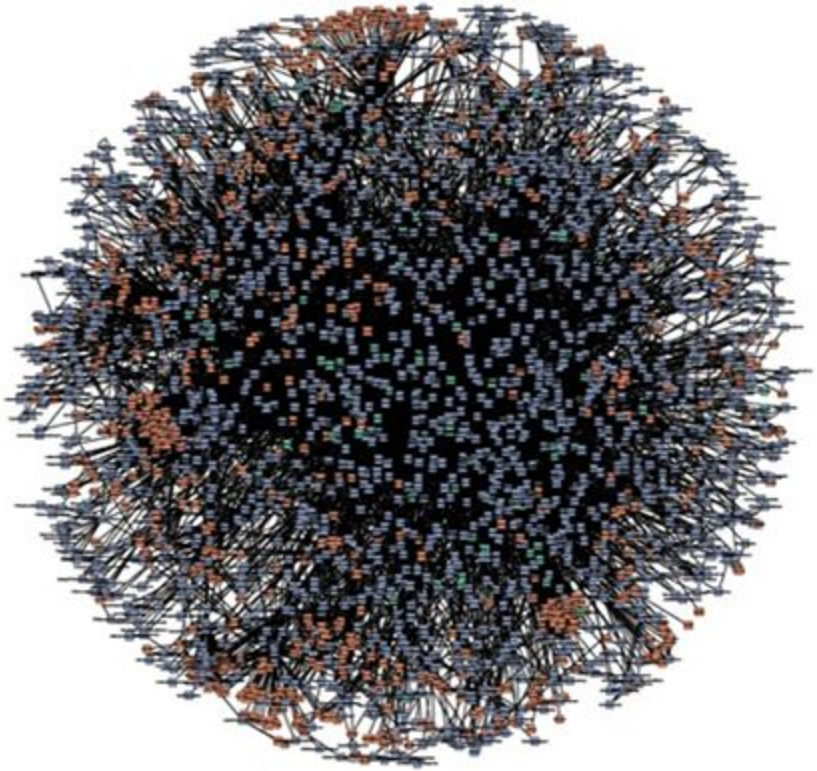
- /actuator/health
- /actuator/info
- /actuator/metrics
- /actuator/trace

- **/actuator/prometheus**

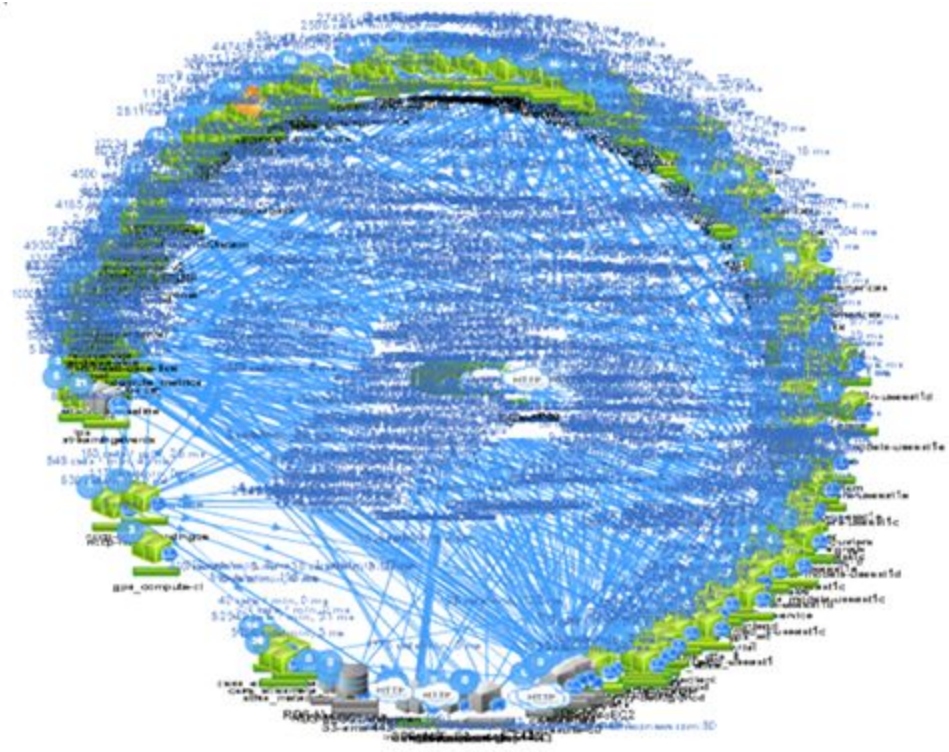

Distributed Tracing

# Where exactly is my problem?



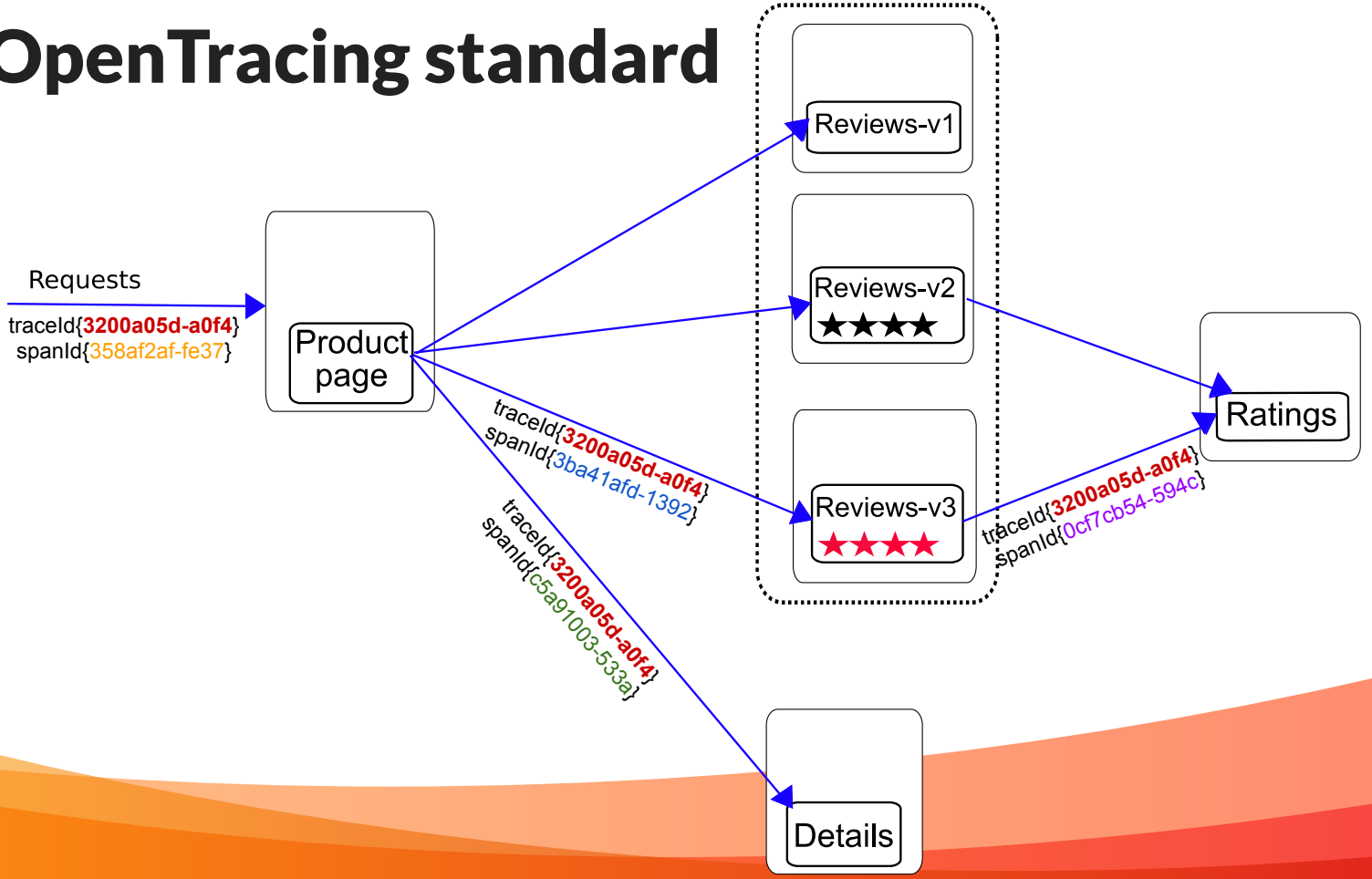


amazon.com



NETFLIX

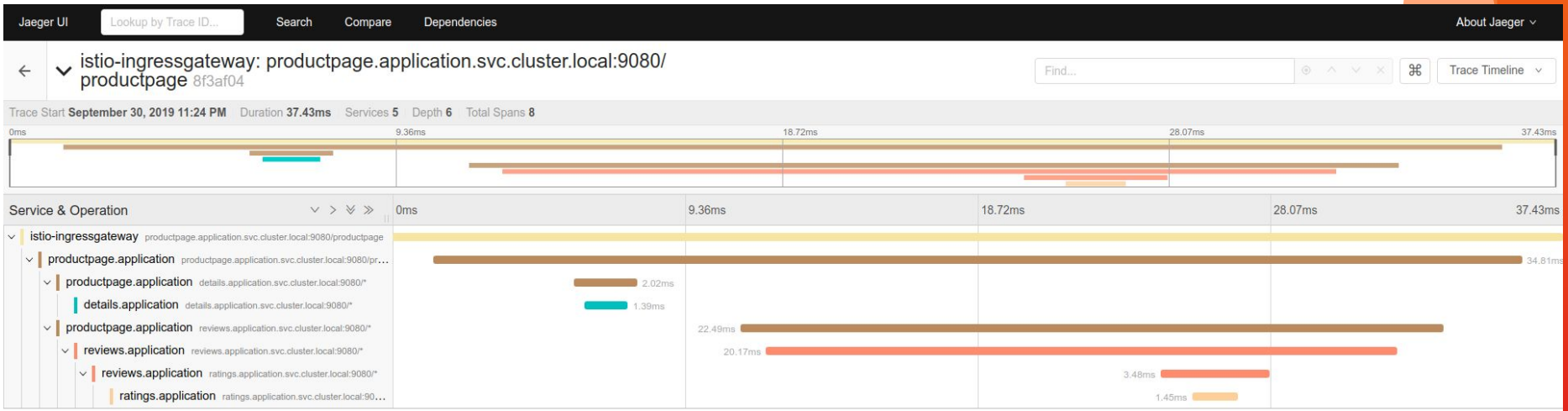
# The OpenTracing standard



# The Distributed Tracing System

Find performance issues quickly through a graphical view

JAEGER



# Spring Cloud Sleuth

Just by adding the Spring dependency:

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-zipkin</artifactId>
</dependency>
```

We automatically gain:

- trace and span instrumentation;
- logging;
- send traces to the distributed tracing system.


Logs

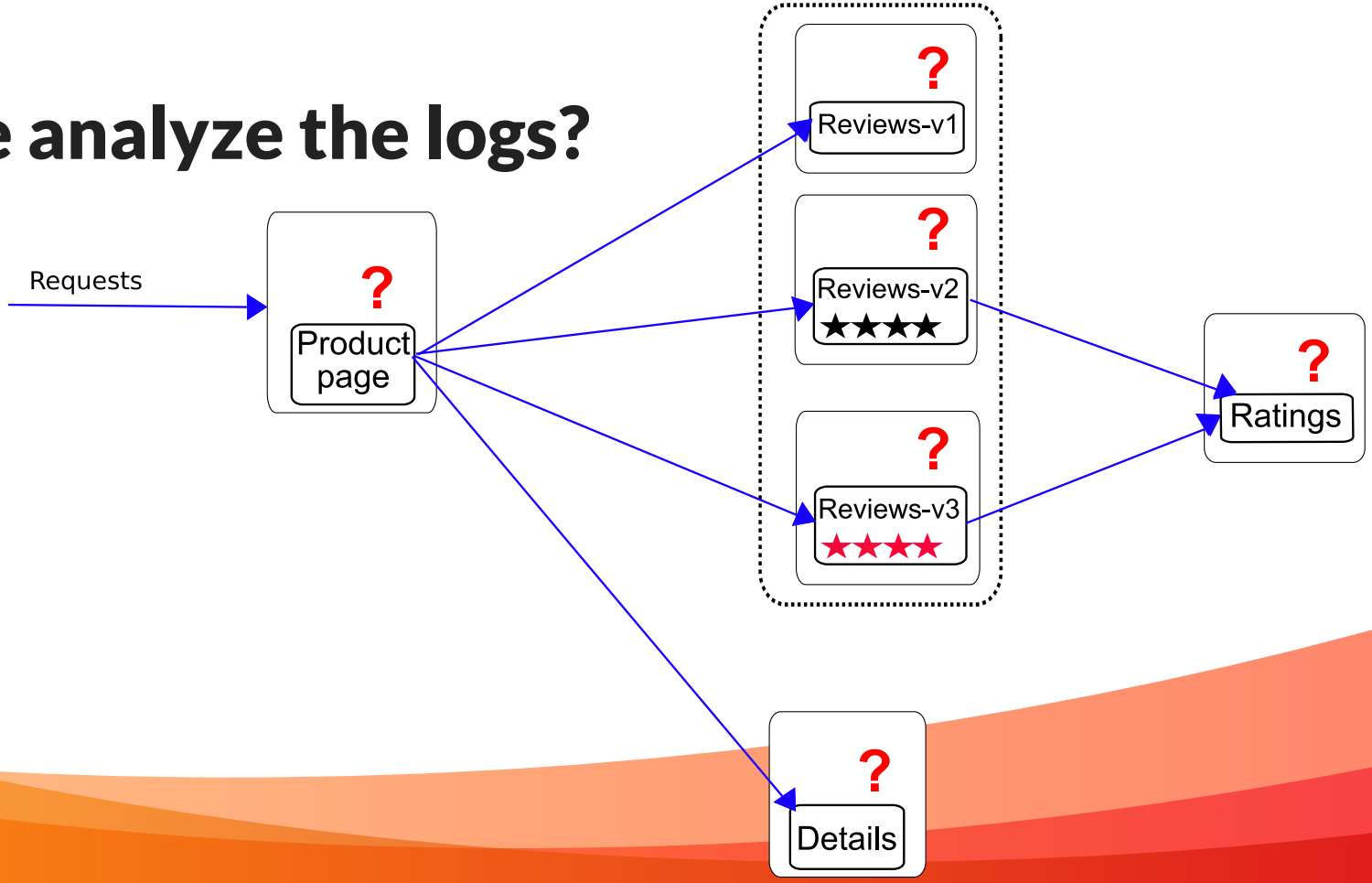
# Logs

**W**ho
**W**hat
**W**hen
**W**here
**W**hy

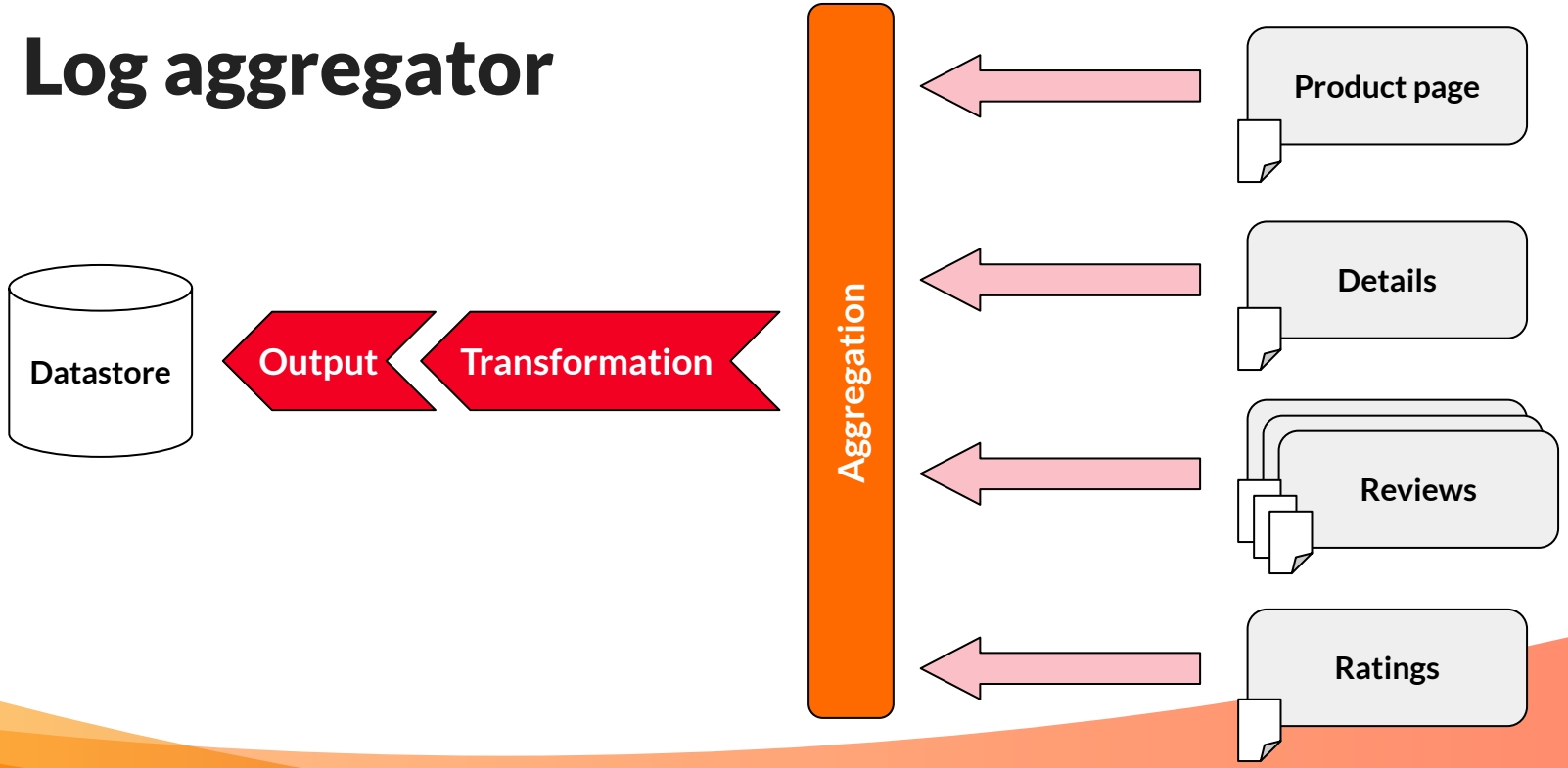
INFO
DEBUG
WARNING
ERROR

```
[INFO][2019-10-10 00:51:48][de4c1b04-9ca1][c.s.domain.service.ProductService] - finding product details by id 140708

[WARN][2019-10-10 00:52:02][de4c1b04-9ca1][c.s.domain.service.ProductService] - product details non-cached, calling details service

[ERROR][2019-10-10 00:52:03][de4c1b04-9ca1][c.s.domain.service.ProductService] - error when calling /details/140708
org.springframework.web.server.ResponseStatusException: 404 NOT_FOUND
```

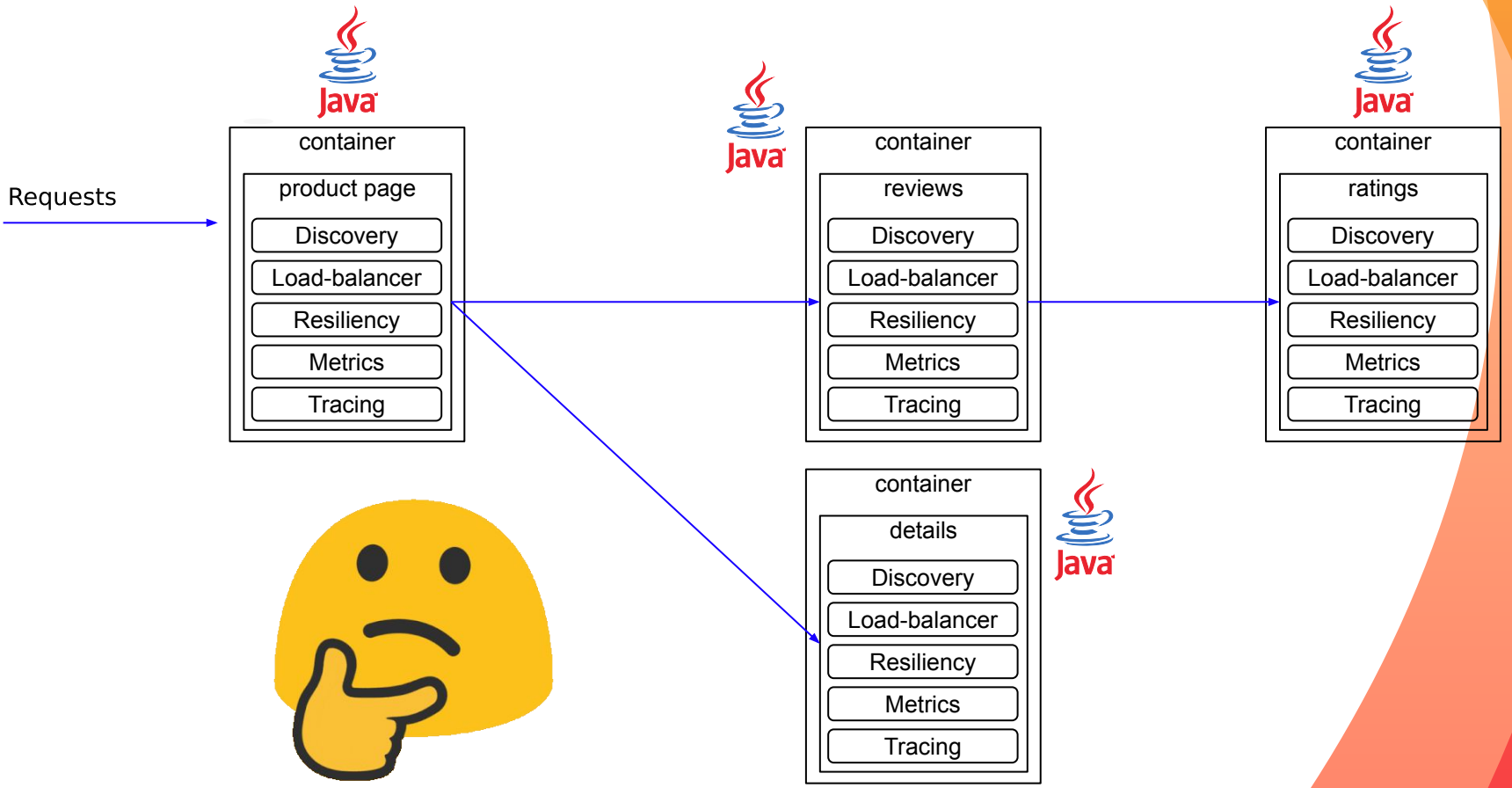
# Where analyze the logs?



# Log aggregator



# But... What's wrong with that?

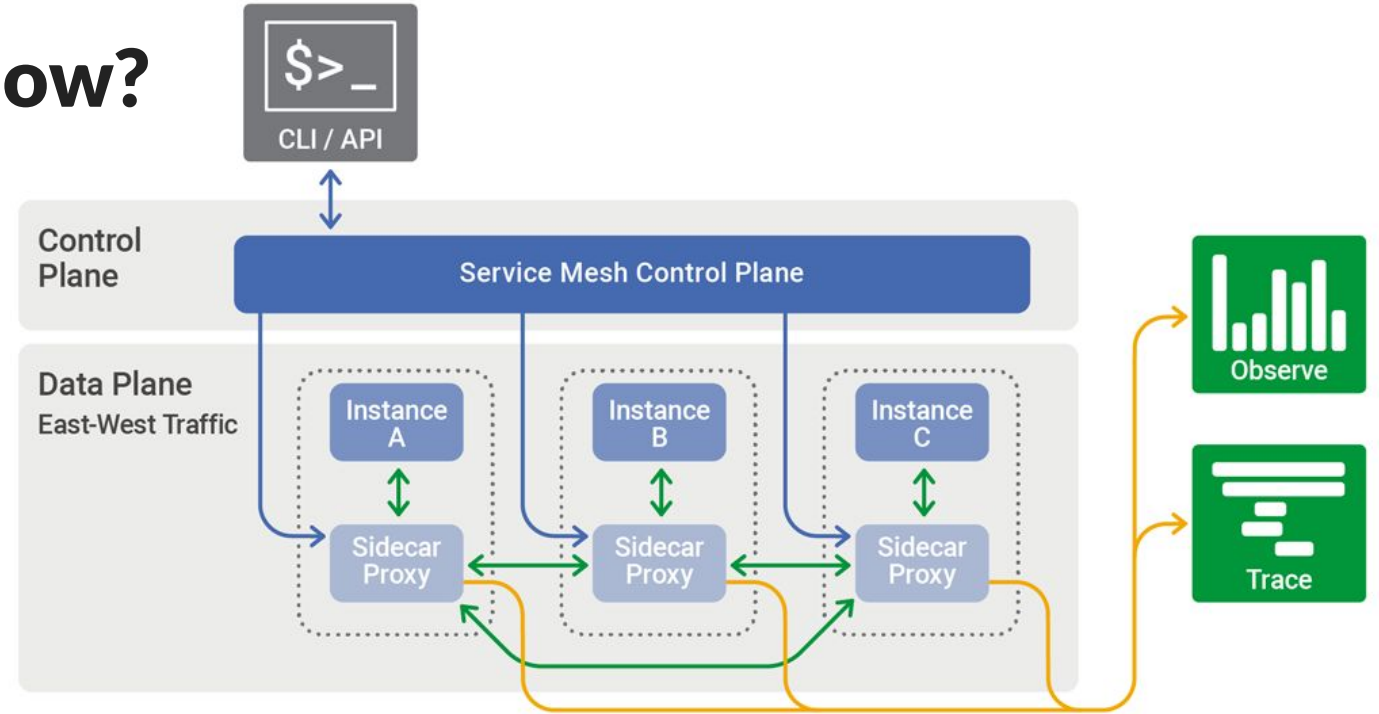



Service Mesh

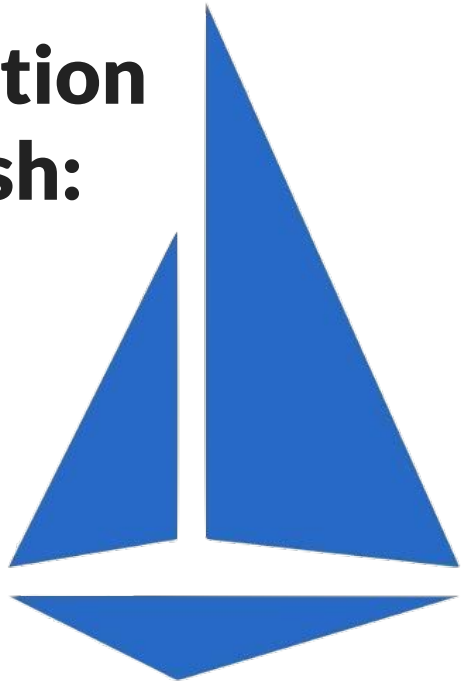
# What can we do with it?

- East/West Traffic Control
- Service Discovery
- Routing
- Security
- **Observability**

# How?


$>_
CLI / API
Control Plane
Service Mesh Control Plane
Data Plane
East-West Traffic
Instance A
Instance B
Instance C
Sidecar Proxy
Sidecar Proxy
Sidecar Proxy
Observe
Trace

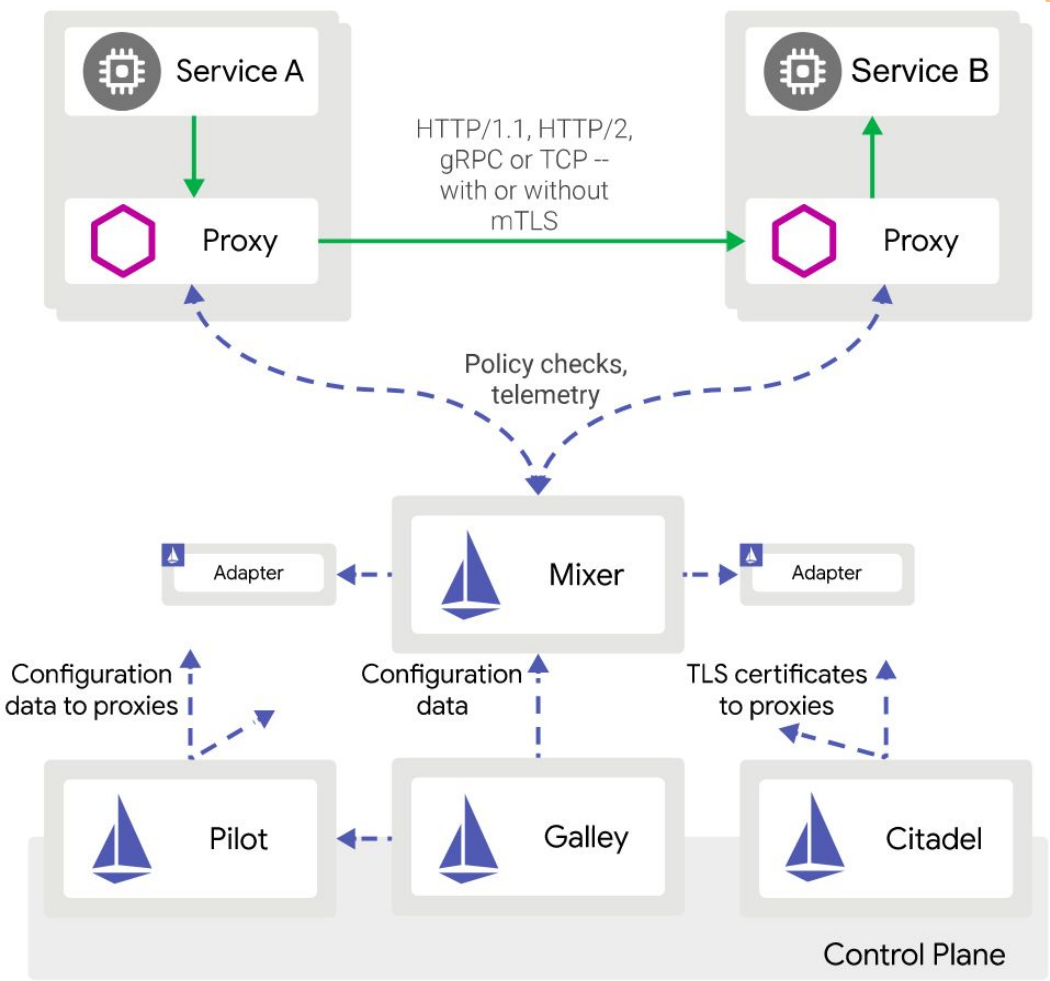
# An implementation of a service mesh:

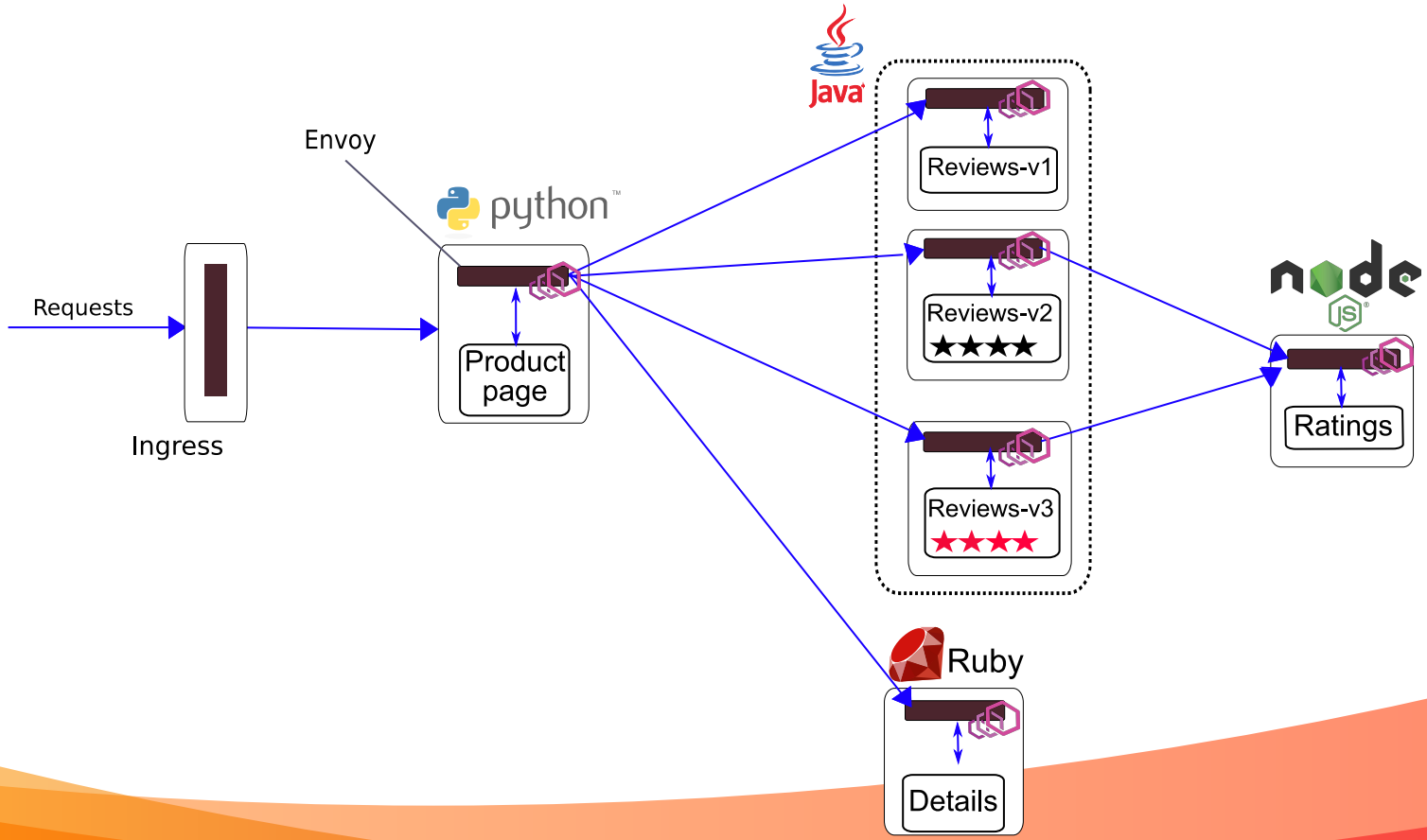


ISTIO

# Components:

- Data-Plane
  - Envoy Proxy
- Control-Plane
  - Pilot
  - Galley
  - Citadel
  - **Mixer**

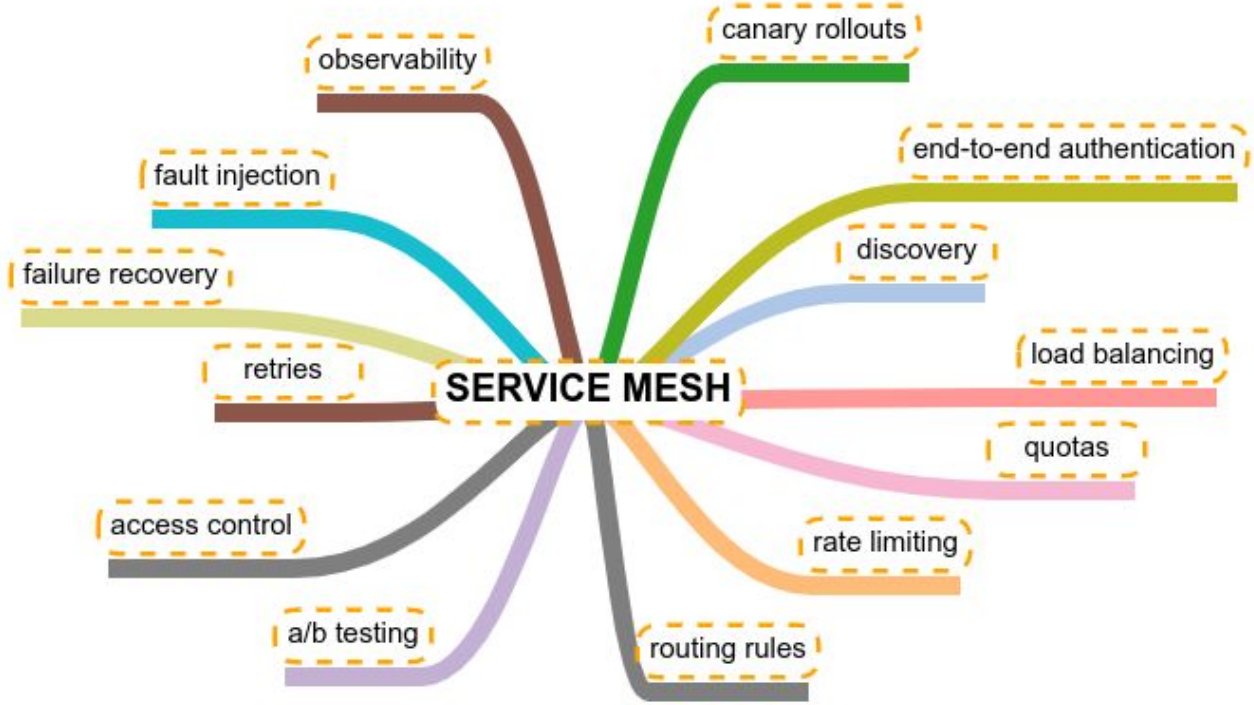



Envoy
Requests
Ingress
python
Product page
Java
Reviews-v1
Reviews-v2
Reviews-v3
node
Ratings
Ruby
Details

# Demo time! 😈

https://youtu.be/6uJ_3hw8GnA

# It's NOT JUST about observability



# Thanks!

## Any questions?

You can find us at:

 mfariam

 kurtisangelo

 matheusfm

 angelokurtis

 matheusfm

 tiagoangelo