Observability-Driven Software Engineering

Wahab Hamou-Lhadj
Concordia University
Montréal, QC, Canada

Naser Ezzati-Jivan
Brock University
St. Catharines, ON, Canada

Keynote Presentation

5th International Conference on Wireless, Intelligent, and Distributed Environment for Communication (WIDECOM)
Windsor, ON, Canada
October 12, 2022

User vs. Operational Data

- **User data** describes information about users.
  - E.g. social media data, user preferences, geo-location data, images, etc.
  - Applications include marketing campaigns, fraud detection, image recognition, etc.

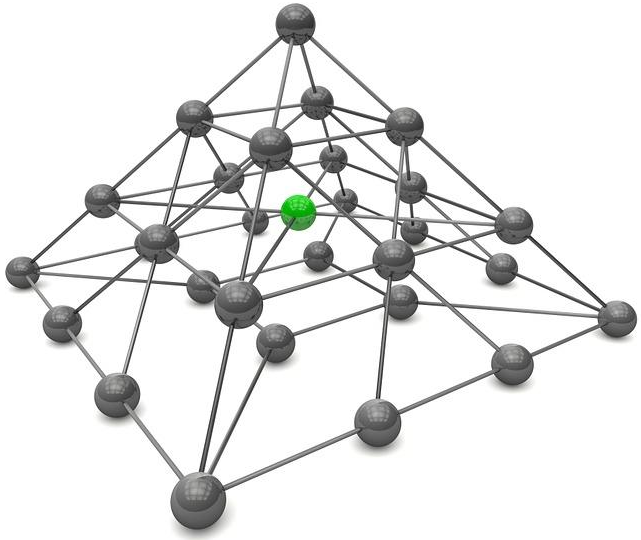
User vs. Operational Data

- **Operational (machine) data** describes information about a system (or a machine)
- It is collected automatically from devices, IT platforms, applications with no direct user intervention.
  - Useful for diagnosing service problems, ensuring reliability, detecting security threats, improving operations, and so on.

Operational Data for Software-Intensive Systems

- The proper functioning of software-intensive systems **relies heavily on operational data** to diagnose and prevent problems.

- New trends in SW dev. make this challenging:
  - Highly distributed and parallel systems
  - Micro-service architectures
  - Virtualisation and containerization
  - Device connectivity and IoT
  - Cyber physical systems
  - Intelligent and autonomous systems
  - Agile, DevOps, and continuous delivery processes

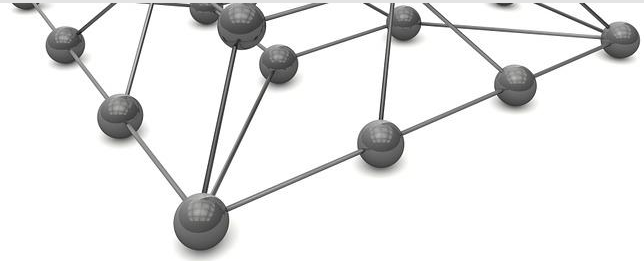


Operational Data for Software-Intensive Systems

- The proper functioning of software-intensive systems **relies heavily on operational data** to diagnose and prevent problems.

We need better runtime system analysis and fault diagnosis and prediction methods that provide full visibility of a system's internal states.

- Micro service architectures
- Virtualisation and containerization
- Device connectivity and IoT
- Cyber physical systems
- Intelligent and autonomous systems
- Agile, DevOps, and continuous delivery processes



Software Observability

- In control theory:
  - **Observability** is “a measure of how well internal states of a system can be inferred from knowledge of its external outputs” [Wikipedia]

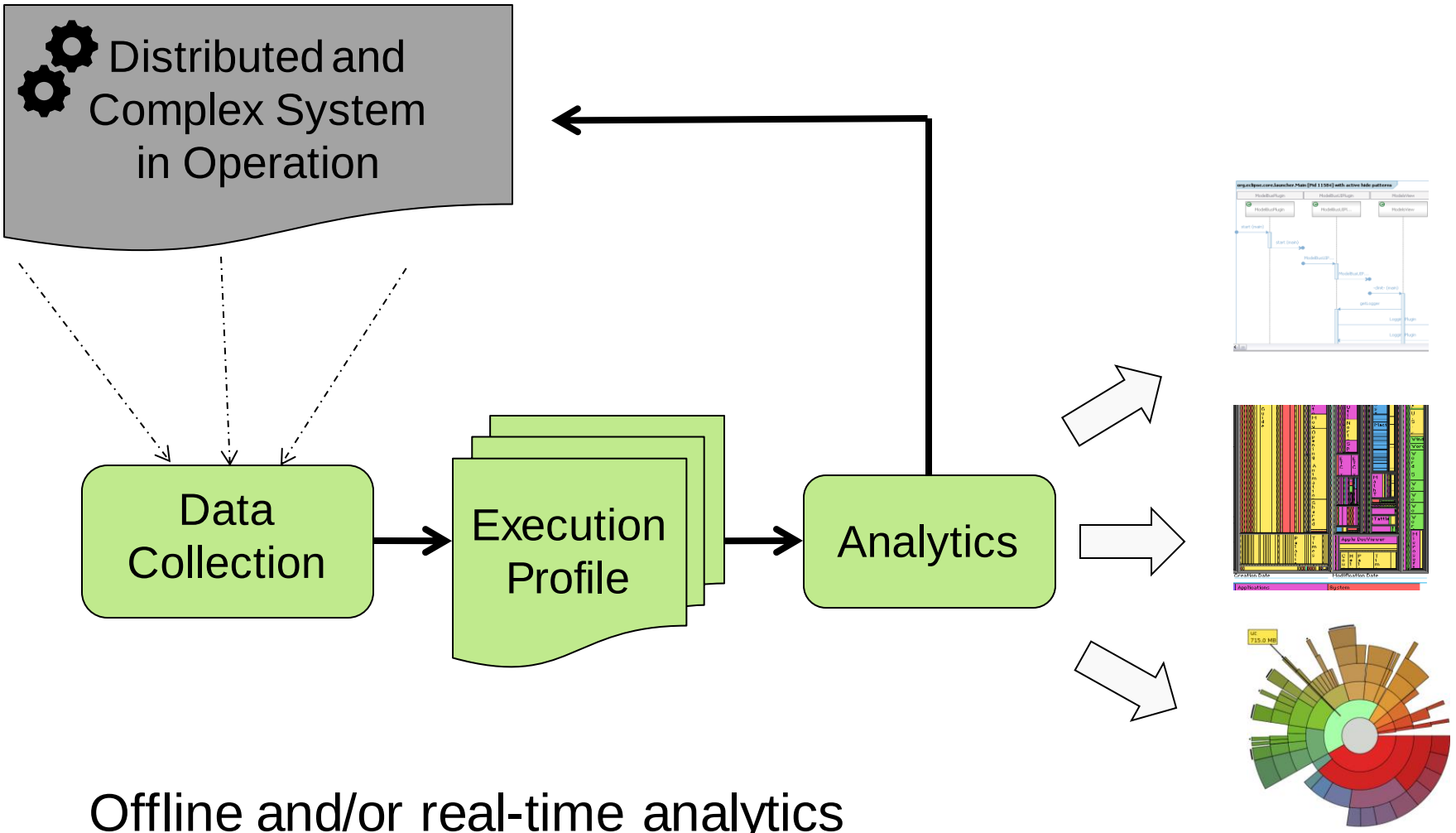
- Software Observability:
  - A set of end-to-end techniques and processes that allow us **to reason** about **what a software system is doing and why** by analyzing its external outputs.

Monitoring vs Observability

- **Monitoring:**
  - Tracks known metrics and raises alerts when thresholds are not met (e.g., 4 golden signals of Google SRE: latency, traffic, errors, and saturation)
  - Answers the question: “how is the system doing?”
  - Helps diagnose known problems

- **Observability:**
  - Answers the question: “what is the system doing and why?”
  - Enables to reason about the system by observing its outputs
  - Helps diagnose known and unknown problems

Building Blocks



Offline and/or real-time analytics

Operational Data

- **Logs:**
  - Records of events generated from logging statements inserted in the code to track system execution, errors, failures, etc.
  - Different types of logs: system logs, application logs, event logs, etc.
- **Traces:**
  - Records of events showing execution flow of a service or a (distributed) system with causal relationship
  - Require additional instrumentation mechanisms
- **Profiling Metrics:**
  - Aggregate measurements over a period of time (e.g., CPU usage, number of user requests, etc.)

Emergence of AI for IT Operations

- AIOps is the application of AI to enhance IT operations
- An important enabler for digital transformation
- Building Blocks:
  - Data collection and aggregation
  - Pattern recognition
  - Predictive analytics
  - Visualization
- Applications:
  - Fault detection and prediction
  - Root cause analysis
  - Security
  - Regulatory compliance
  - Operational intelligence

Characteristics of Logs and Traces

- **Velocity:** the data (in some cases) must be processed in real time
- **Volume:** mountain ranges of historical data
- **Variety:** captured data can be structured or unstructured
- **Veracity:** captured data must be cleaned
- **Value:** not all captured data is useful

Challenges

- **Standards and Best Practices:**
  - Lack of guidelines and best practices for logging, tracing, and profiling
  - Lack of standards for representing logs, traces, and metrics (not the OpenTelemetry initiative)
- **Data Characteristics**
  - Mainly unstructured data
  - Size is a problem
  - Not all data is useful
  - High velocity

Challenges

- **Analytics and Tools:**
  - Mainly descriptive analytics
  - Predictive analytics not fully explored
  - Mainly offline analysis techniques
  - Lack of usable end-to-end observability tools

- **Cost and Management Aspects**
  - Cost vs. benefits not well understood
  - No clear alignment of observability with other initiatives
  - Roles and responsibilities are not well defined

Challenges

- **Analytics and Tools:**
  - Mainly descriptive analytics

There is a need for systematic and engineering approaches to software observability that promote best practices throughout the entire software development lifecycle

- **Cost and Management Aspects**
  - Cost vs. benefits not well understood
  - No clear alignment of observability with other initiatives
  - Roles and responsibilities are not well defined

Observability By Design

- Bringing observability **to early stages** of the software development lifecycle.
- Defining a set of **observability patterns, best practices, and reusable solutions** to be used as guiding principles for developers.
- A **systematic approach** to tracing, logging and profiling of software systems that considers different phases of the software process.

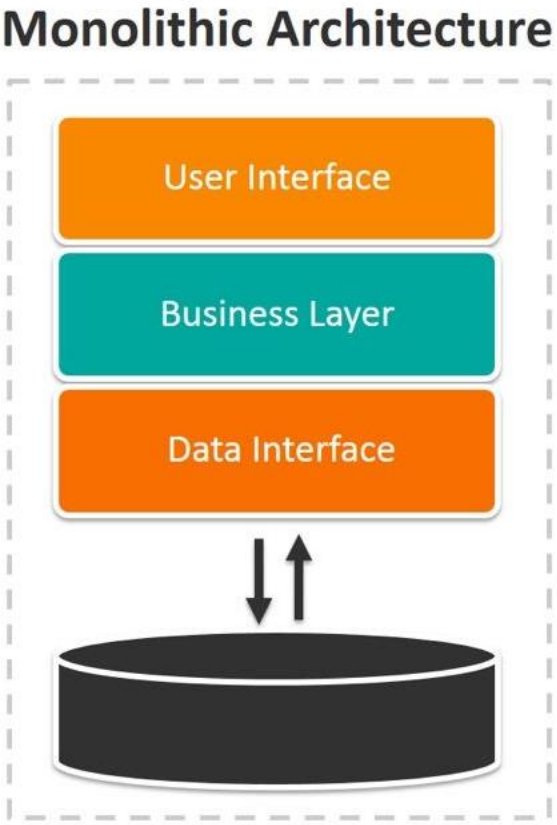
Production-Debugging Monoliths

Pre-production:

- Robust: test your few known failure modes
- Performant: benchmark, load, stress tests
- Correct: unit, integration, end-to-end tests

Production:

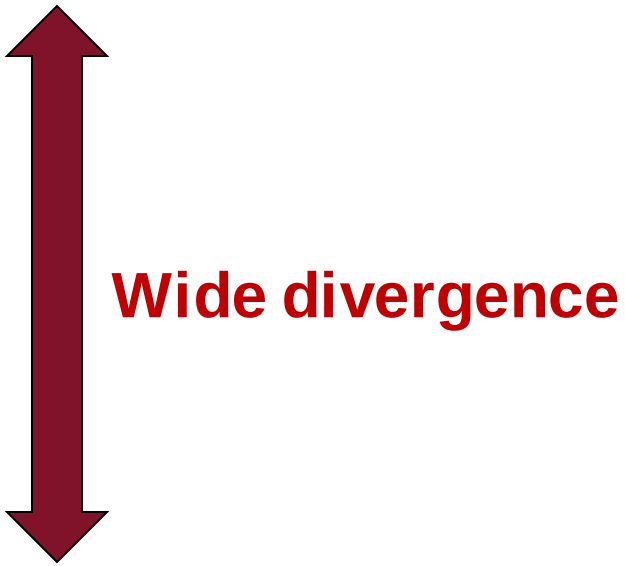
- Monitoring to detect issues (error, latency)
- Logging to troubleshoot them



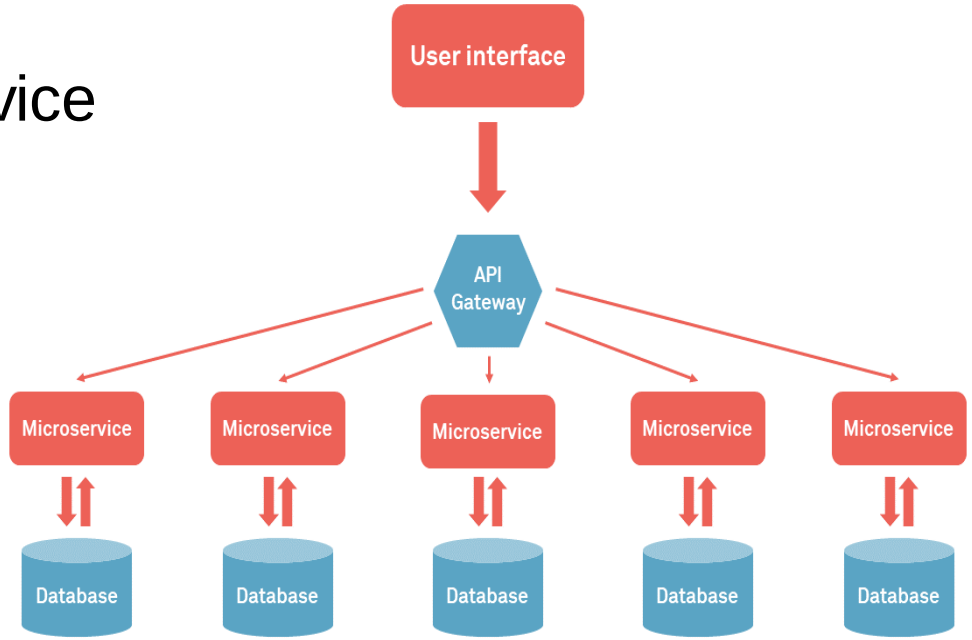
Production-Debugging Microservices

Pre-production:

- test each individual service



Wide divergence

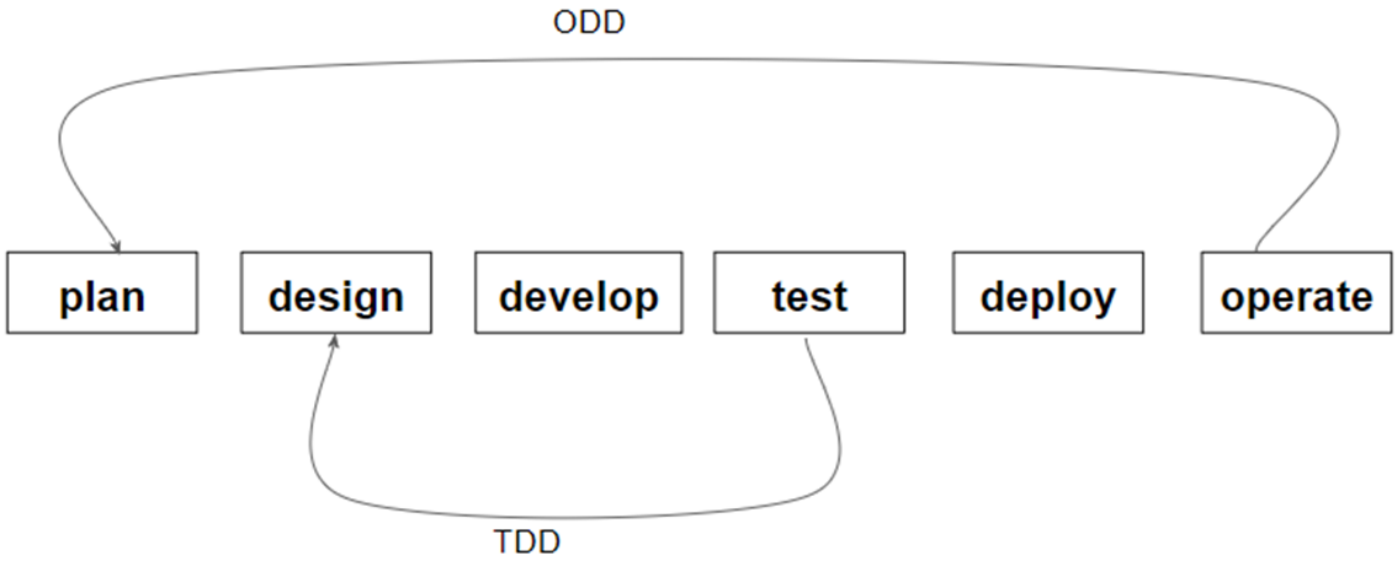


Production: (no longer replicateable)

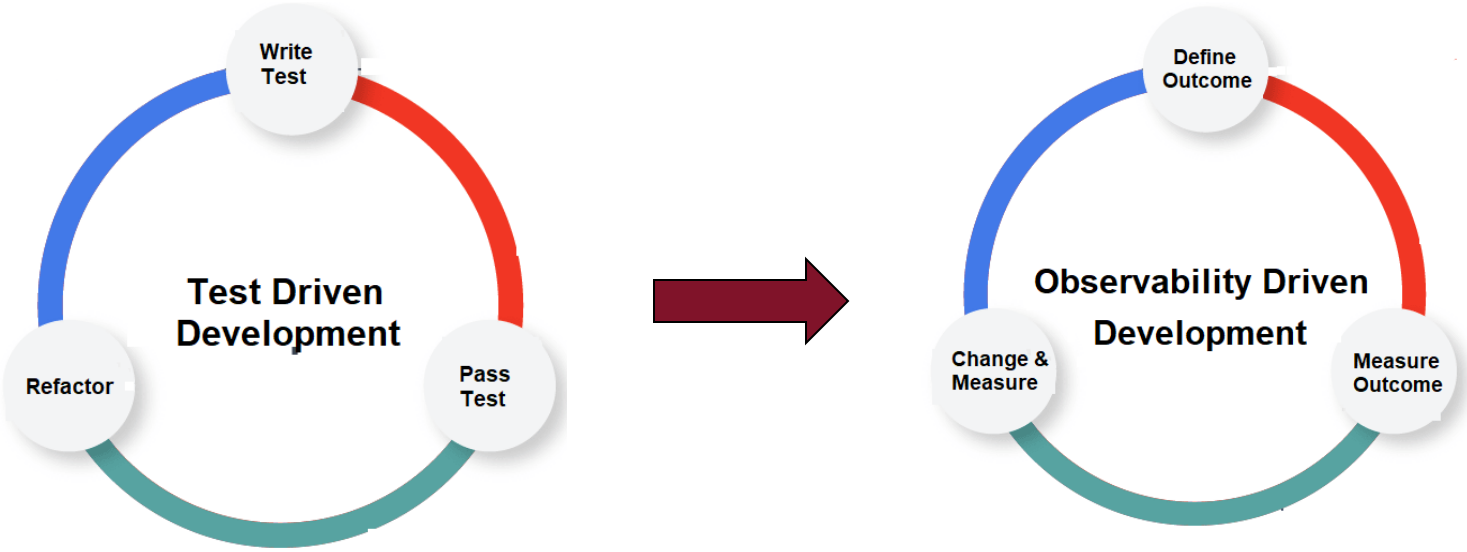
Tracing to troubleshoot issues

Observability-Driven Development (ODD)

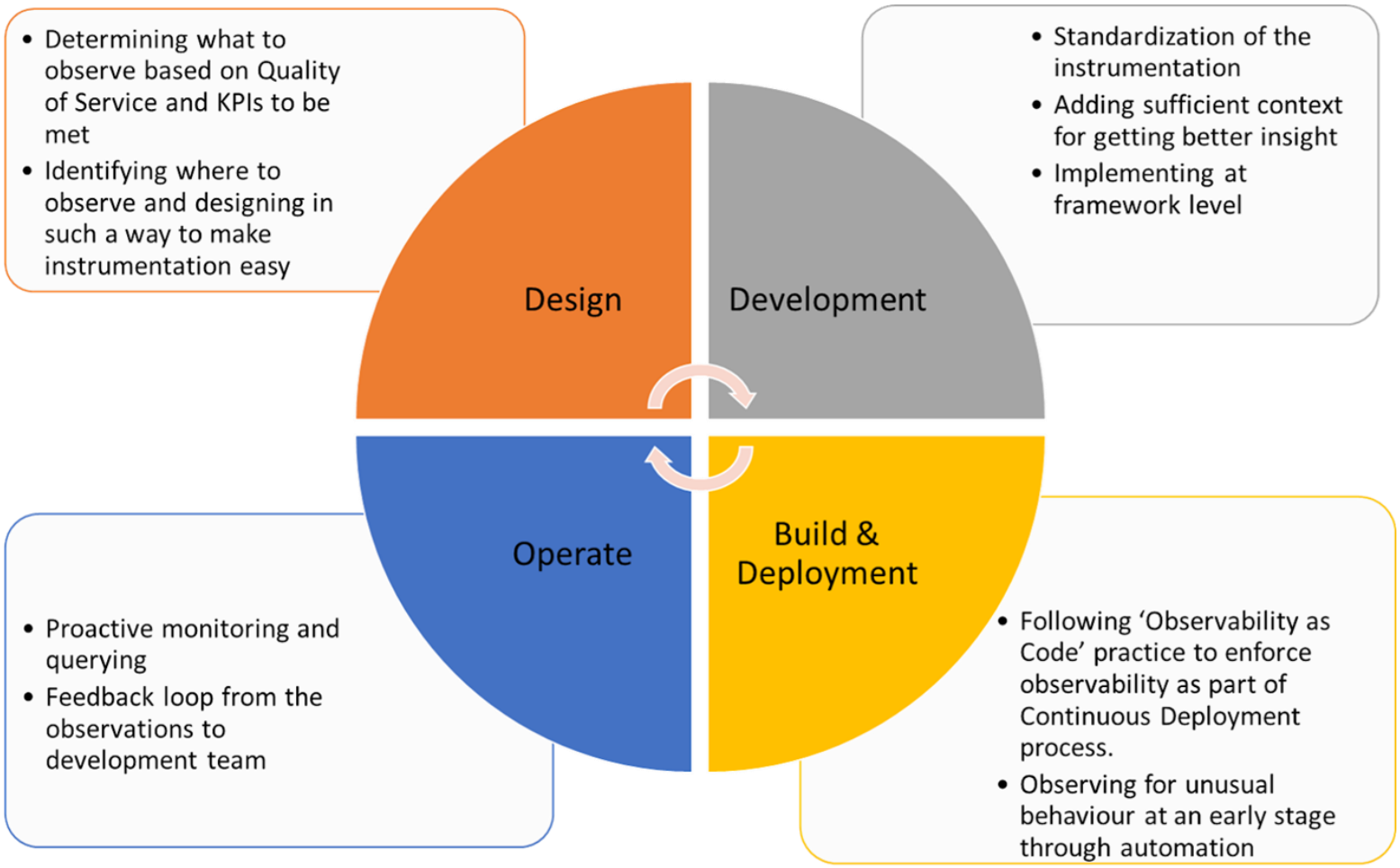
- Leveraging tools and hands-on developers to observe system state and behavior
  - Interrogating the system, not just setting and measuring thresholds and metrics for it



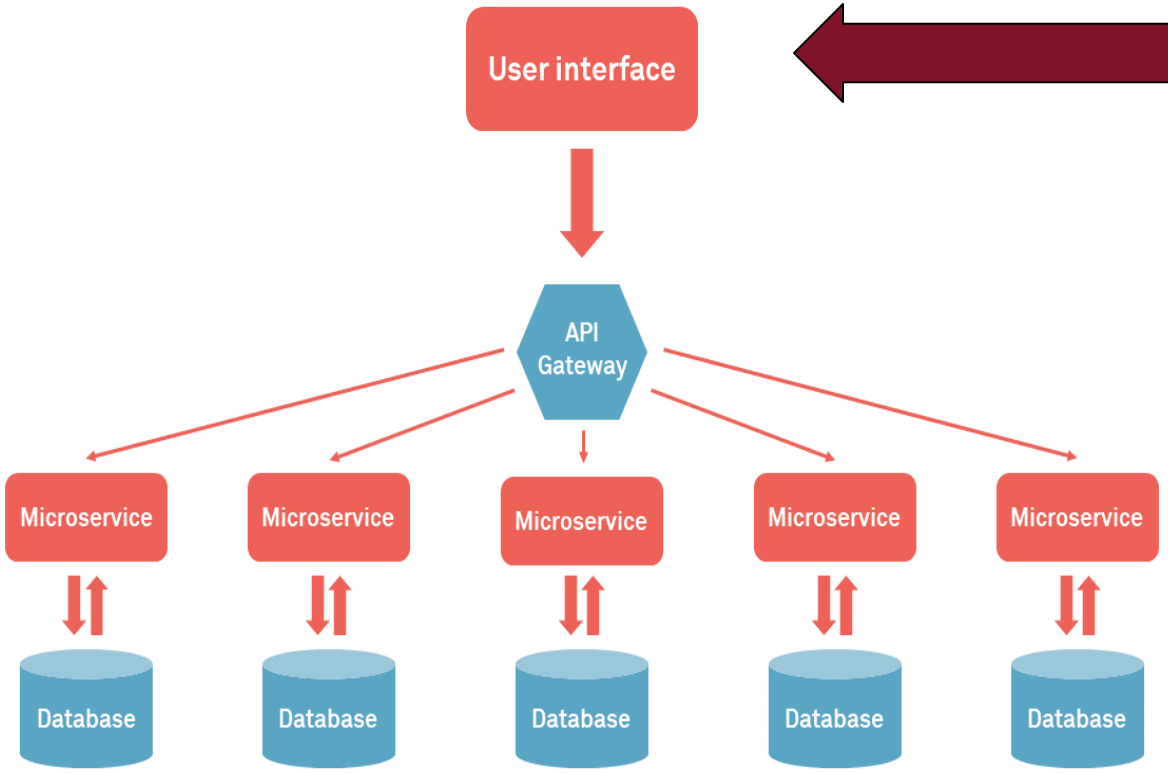
From TDD to ODD



Observability-Driven Development (ODD)



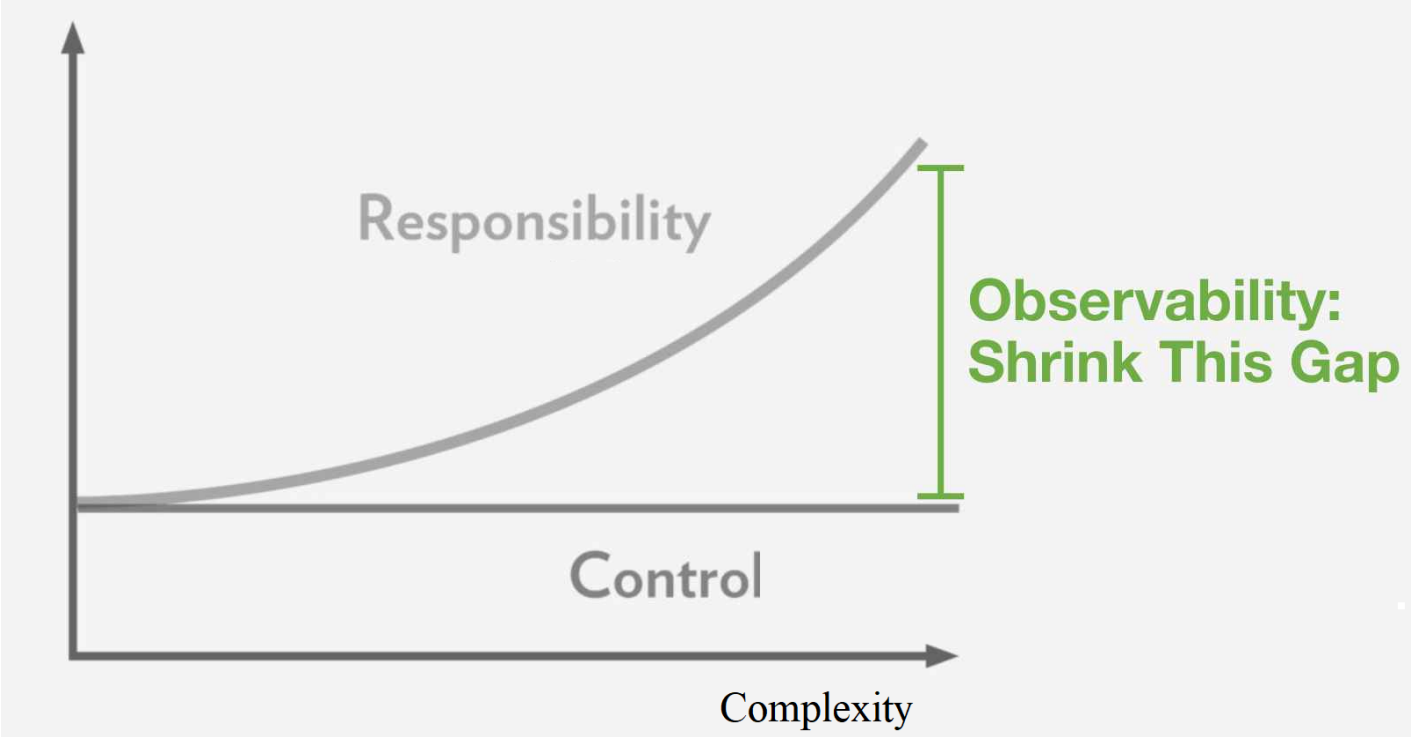
Observability-Driven Development (ODD)



What you can control

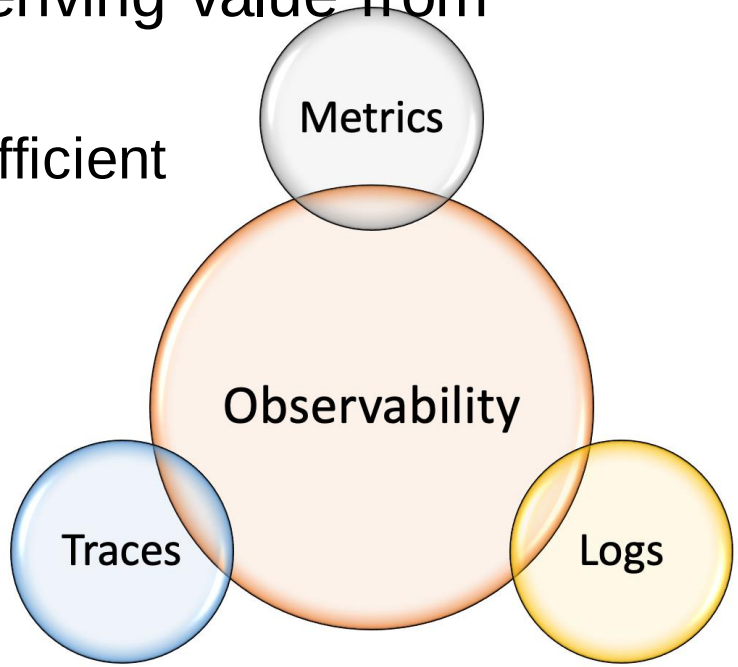
What you are responsible for

Observability-Driven Development (ODD)



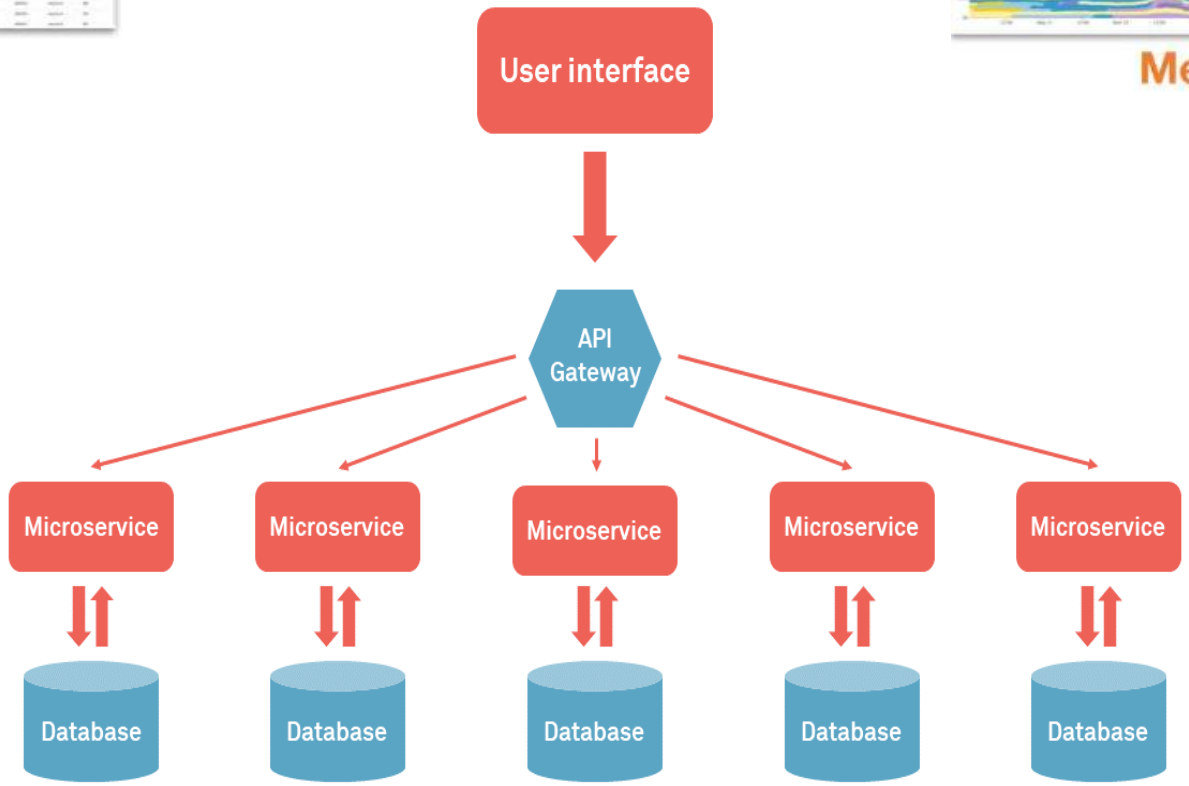
From Telemetry to Observability

- Observability is often equated with telemetry
  - "If you have metrics, logs, and traces, then you have Observability"
- Observability, is the process of deriving value from telemetry
  - Telemetry is important but not sufficient

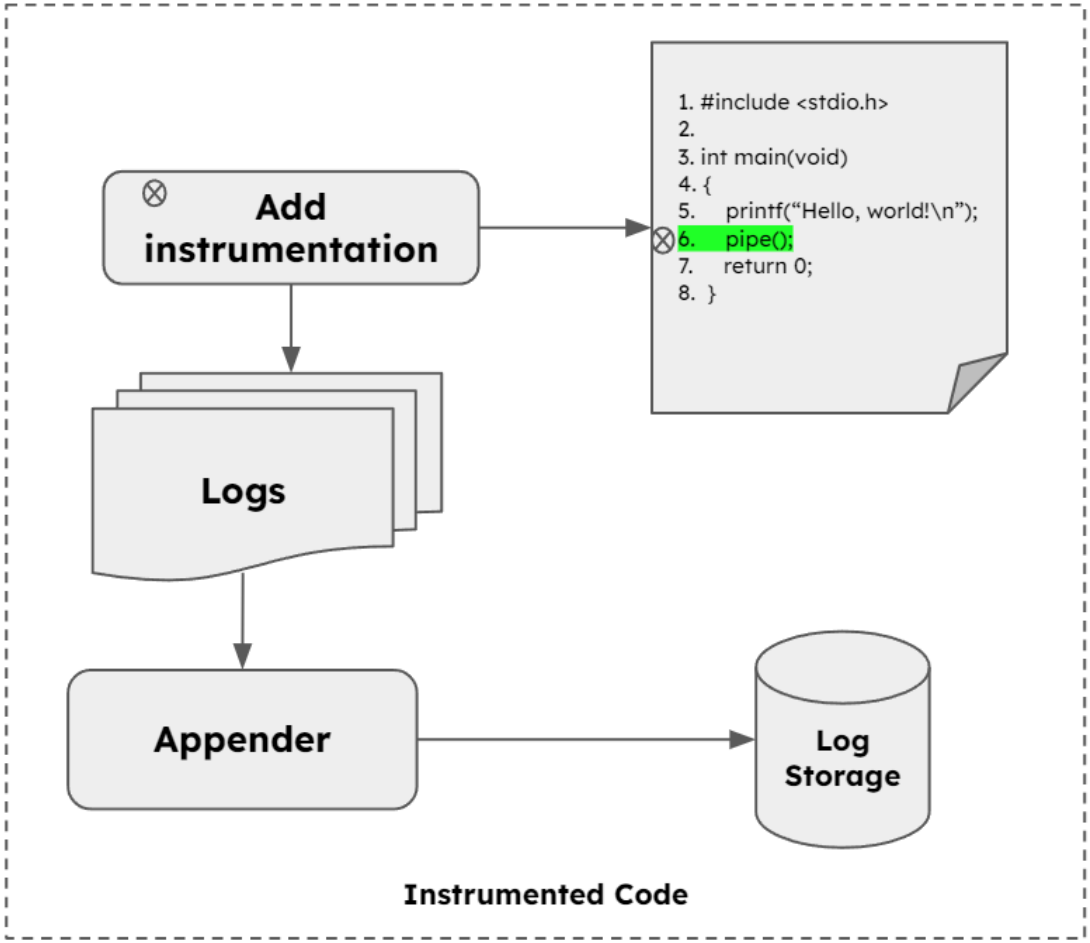


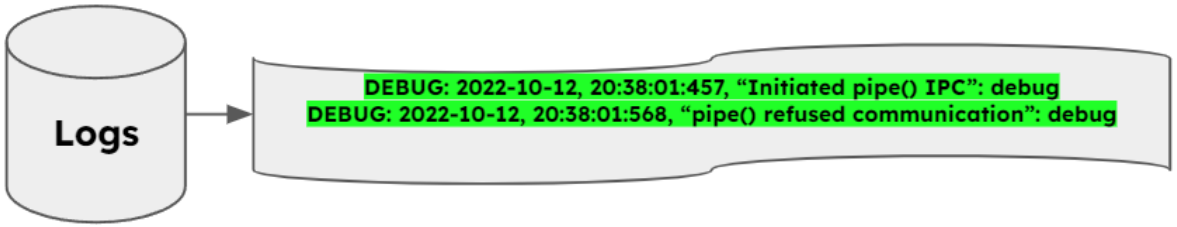
Logs

Metrics



Logs


1. #include <stdio.h>
2.
3. int main(void)
4. {
5. printf("Hello, world!\n");
6. pipe();
7. return 0;
8. }
Add instrumentation
Logs
Appender
Log Storage
Instrumented Code

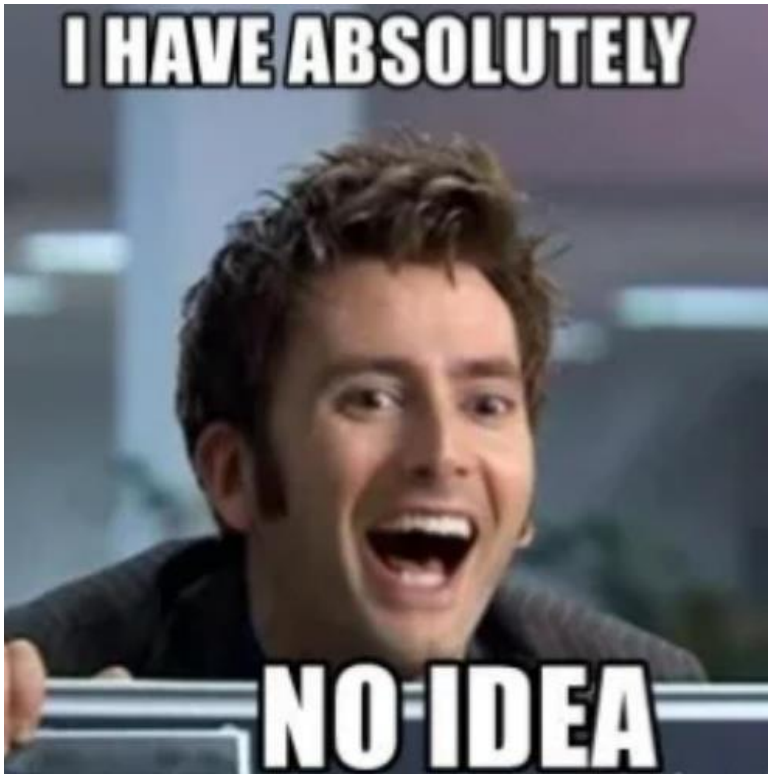

Logs
DEBUG: 2022-10-12, 20:38:01:457, "Initiated pipe() IPC": debug
DEBUG: 2022-10-12, 20:38:01:568, "pipe() refused communication": debug

Logs

Error
242

Success
1302

Metrics

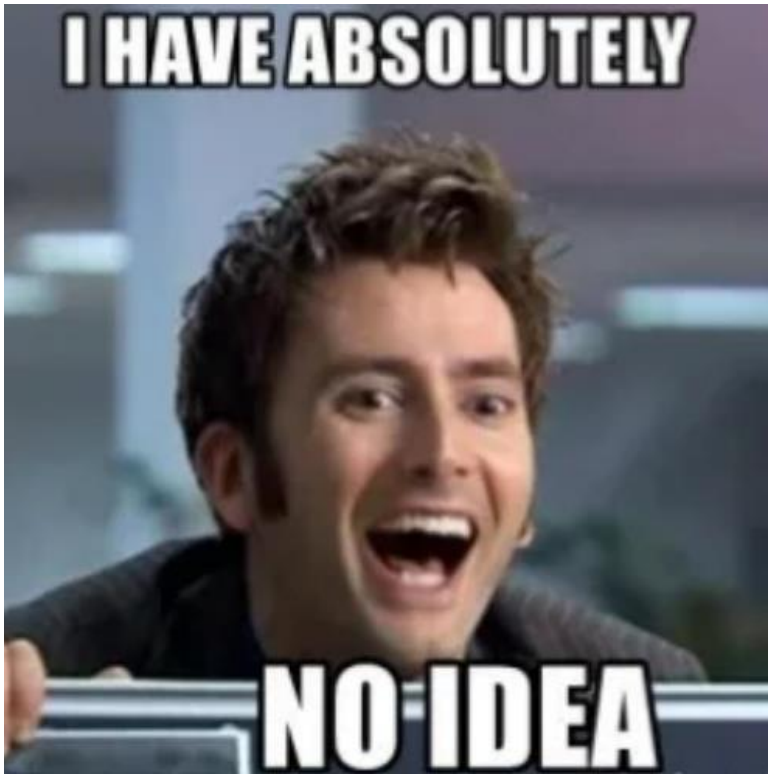


Logs

Error
242

Success
1302

Metrics



Which requests lead to that error or that status?

Logs

Traces

User interface

Traces provide Context

API Gateway

Microservice | Microservice | Microservice | Microservice | Microservice

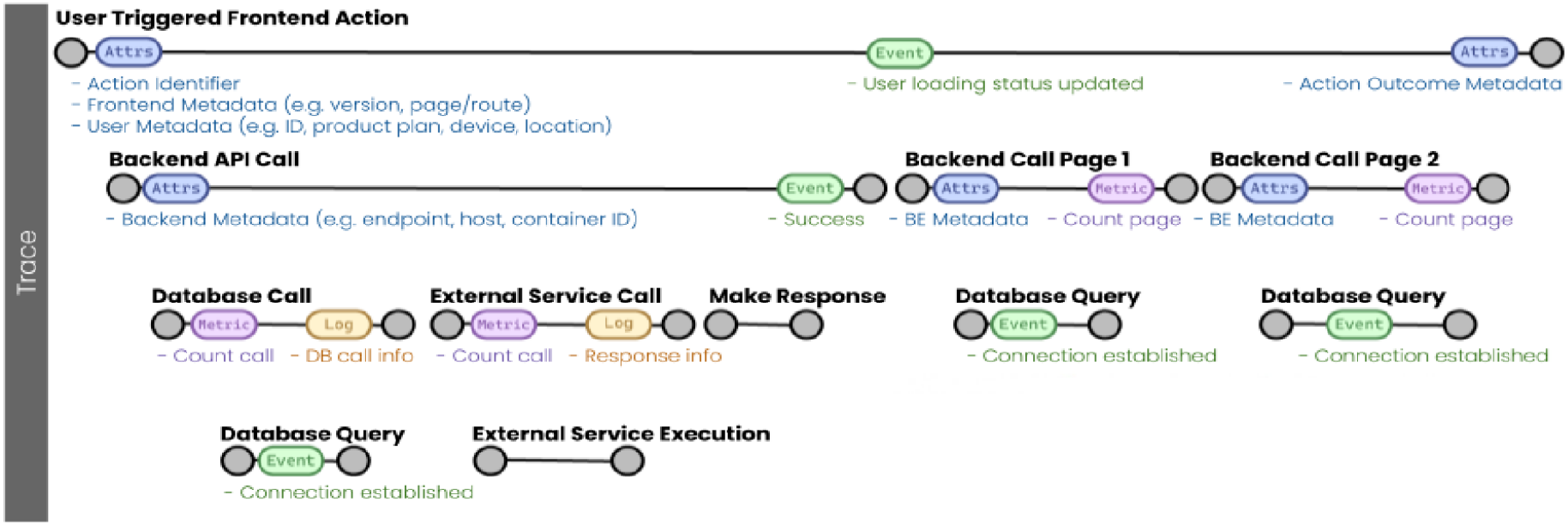
Database | Database | Database | Database | Database

Metrics

Which requests lead to that error or that status?

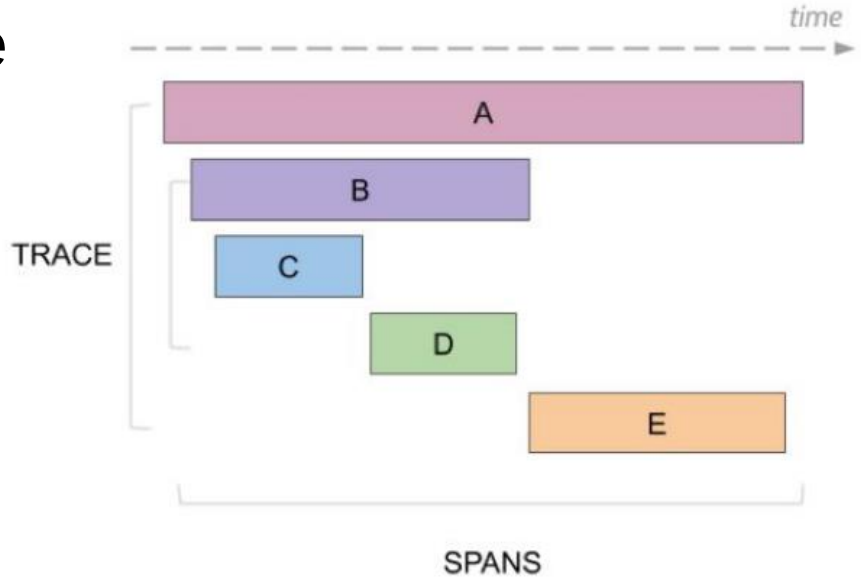
Context

- Context connects everything!



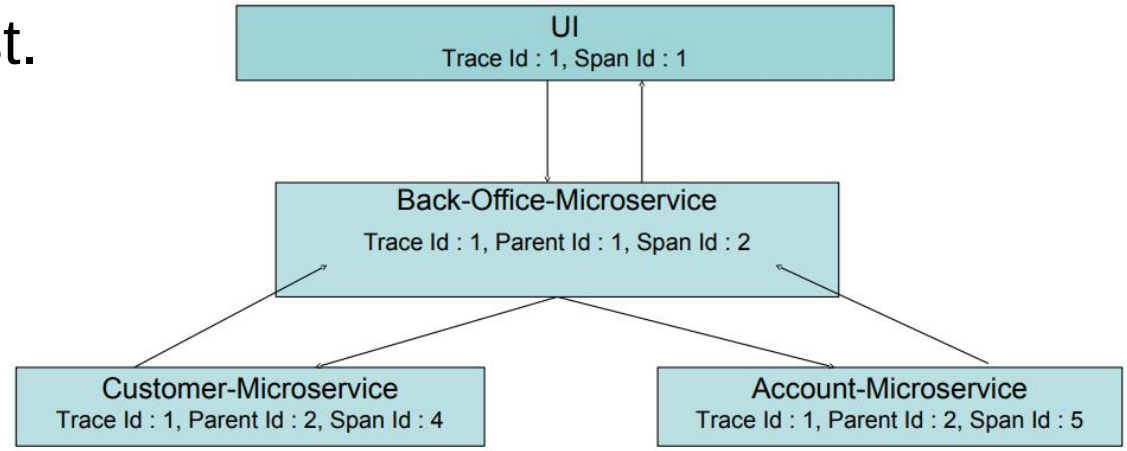
Distributed Tracing

- Trace - a trace is a tree of spans that follows the course of a request or system from its source to its ultimate destination.
- Each trace is a narrative that tells the requests story as it travels through the system.

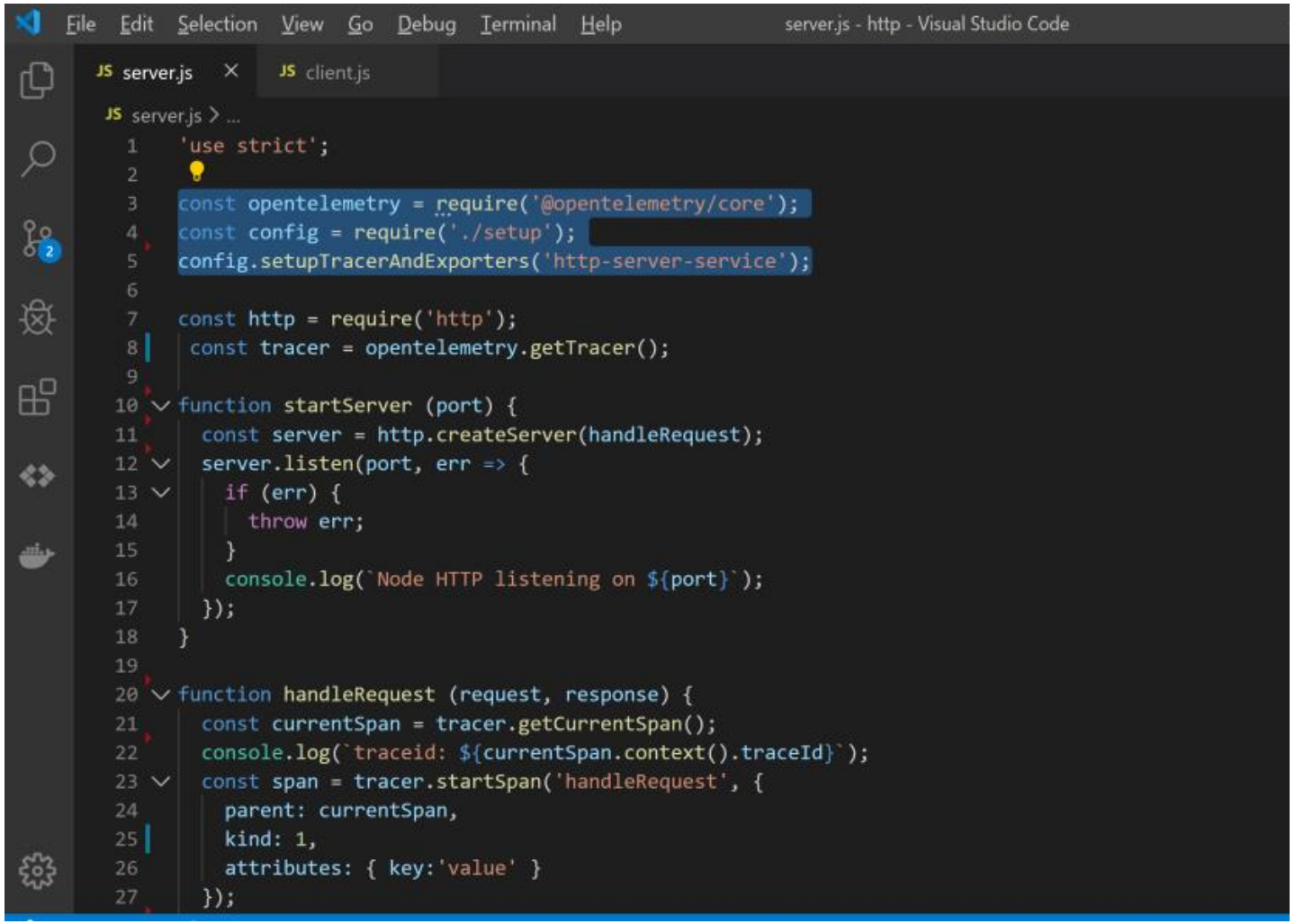


Distributed Tracing

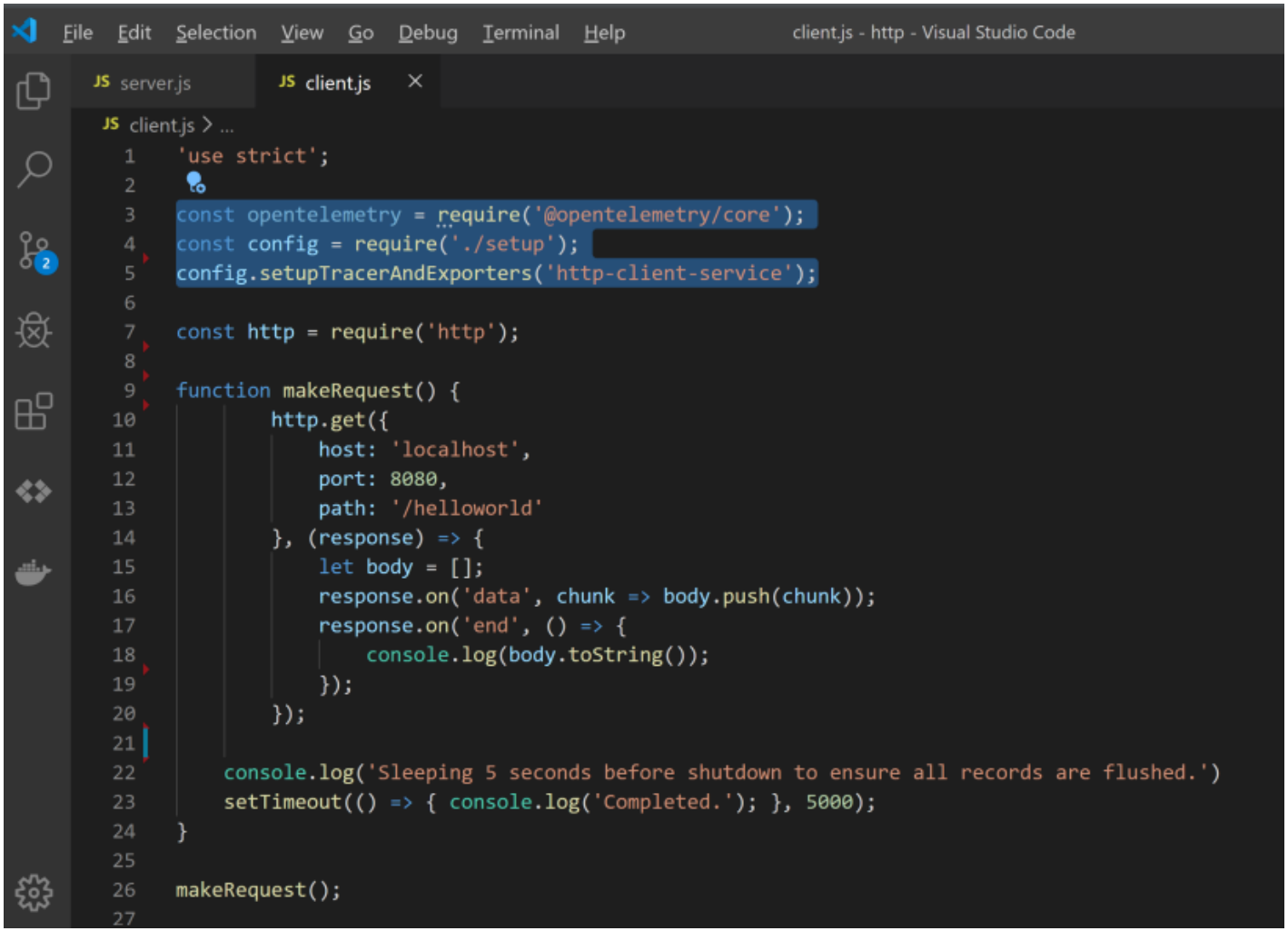
- **Span** - are logical units of work in a distributed system. They all have a name, a start time, and a duration.
- Each Span captures important data points specific to the current process handling the request.



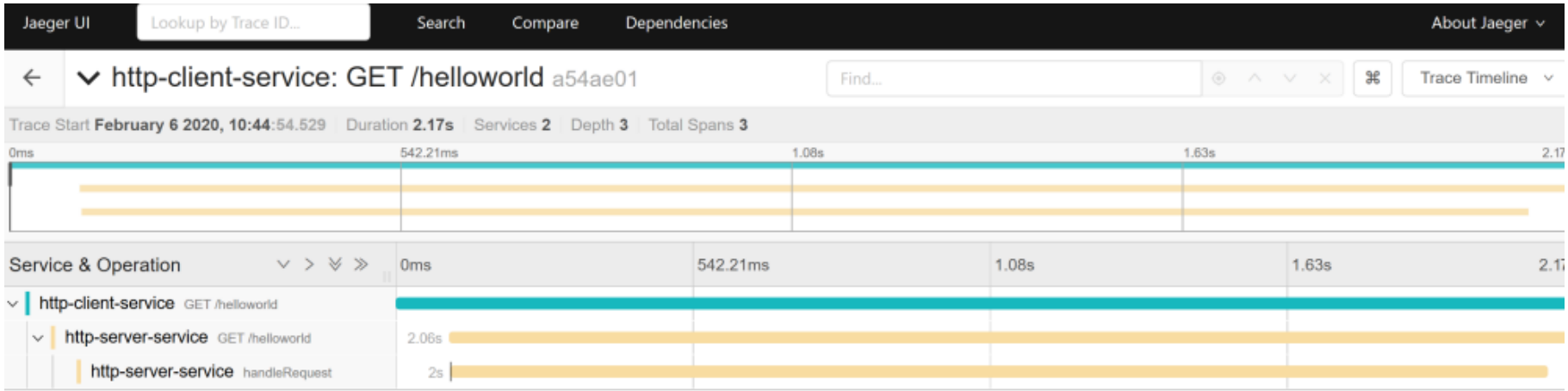
Instrumentation



Instrumentation



Visualization



A Standard Way?

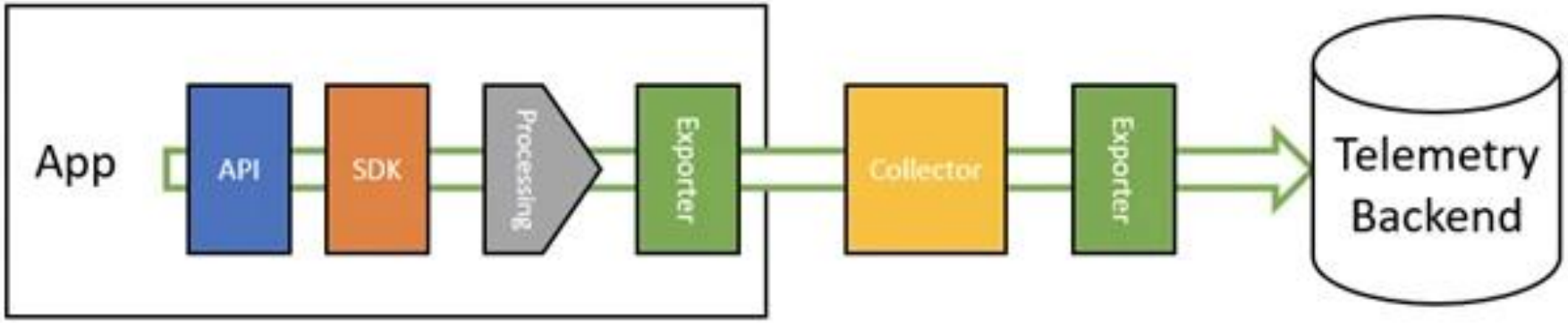
- Tracing libs in Project X do not handoff to tracing libs in Project Y
- Tracing semantics must not be language dependent
- Instrumentation must be decoupled from vendors .

OpenTelemetry

- OpenCensus:
  - Provides APIs and instrumentation that allow you to collect application metrics and distributed tracing.
  - Provides oc-service and oc-agent middleware.
- OpenTracing:
  - Provides APIs for distributed tracing with implementations provided by tracing backend vendors
- **OpenTelemetry**:
  - An effort to combine distributed tracing, metrics and logging into a single set of system components and language-specific libraries

OpenTelemetry

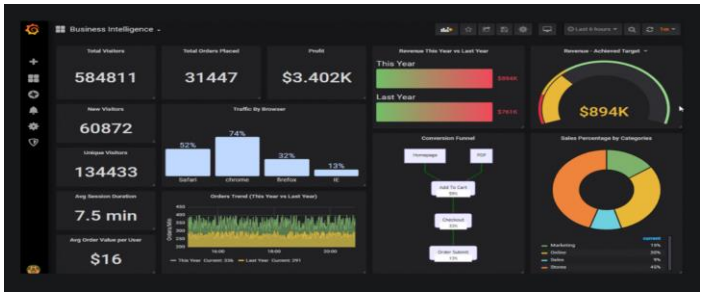
- Vendor-neutral telemetry



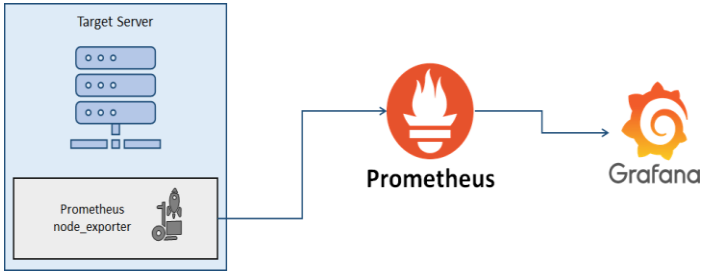
- Instrumentation
  - Changes to the application (source code or configuration)
  - "With great instrumentation comes great observability."
- Data pipeline
- Visualization & Analytics

Metric Analysis & Visualization

- Grafana
- Prometheus
- Kibana



https://grafana.com/



https://prometheus.io/docs/visualization/grafana/

https://www.elastic.co/guide/en/kibana

Observability Culture

- Observability in action!
- Before and after a problem,
- Data-driven decision making
- Educate team
- Encourage standard tools/techniques
  - Log formatting
  - Metric conventions
- Practice, share success stories, and feedback
- Measure your progress and observer your observability culture!

Contact Information

Wahab Hamou-Lhadj, PhD, ing.

Concordia University

wahab.hamou-lhadj@concordia.ca

http://www.ece.concordia.ca/~abdelw

Naser Ezzati-Jivan , PhD

Brock University

nezzatijivan@brocku.ca

http://www.cosc.brocku.ca/~nezzatijivan/