Event-Driven Architecture

Topics

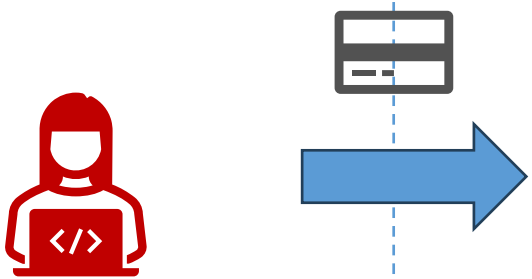
- Motivation for Event-Driven Architecture
- Fundamental concept of Event-Driven Architecture
- Request-response vs Event-Driven Model
- Event-Driven Architecture-Real-Life Example

Topics

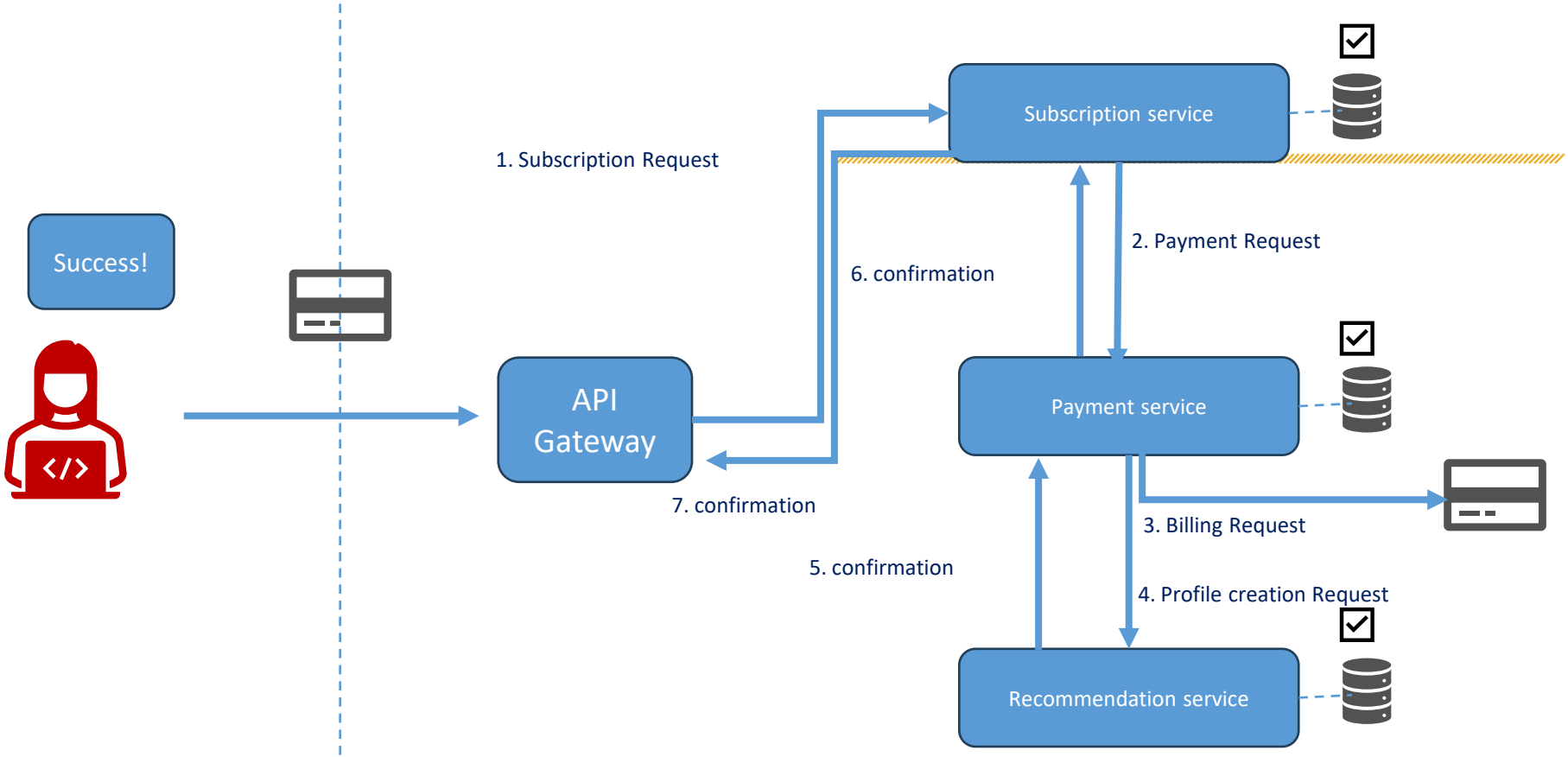
- **Motivation for Event-Driven Architecture**
- Fundamental concept of Event-Driven Architecture
- Request-response vs Event-Driven Model
- Event-Driven Architecture-Real-Life Example

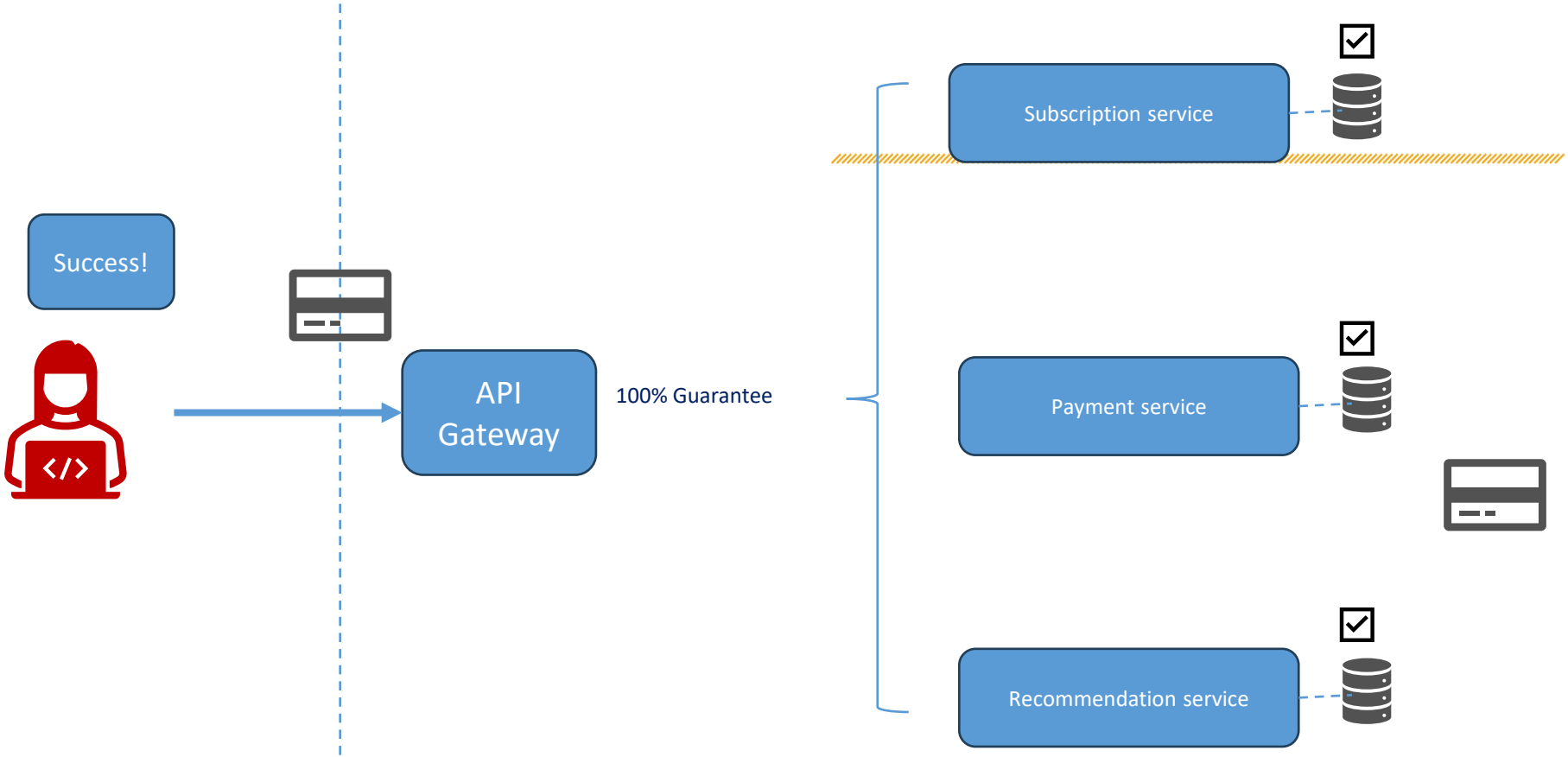
Event-Driven Architecture - Motivation

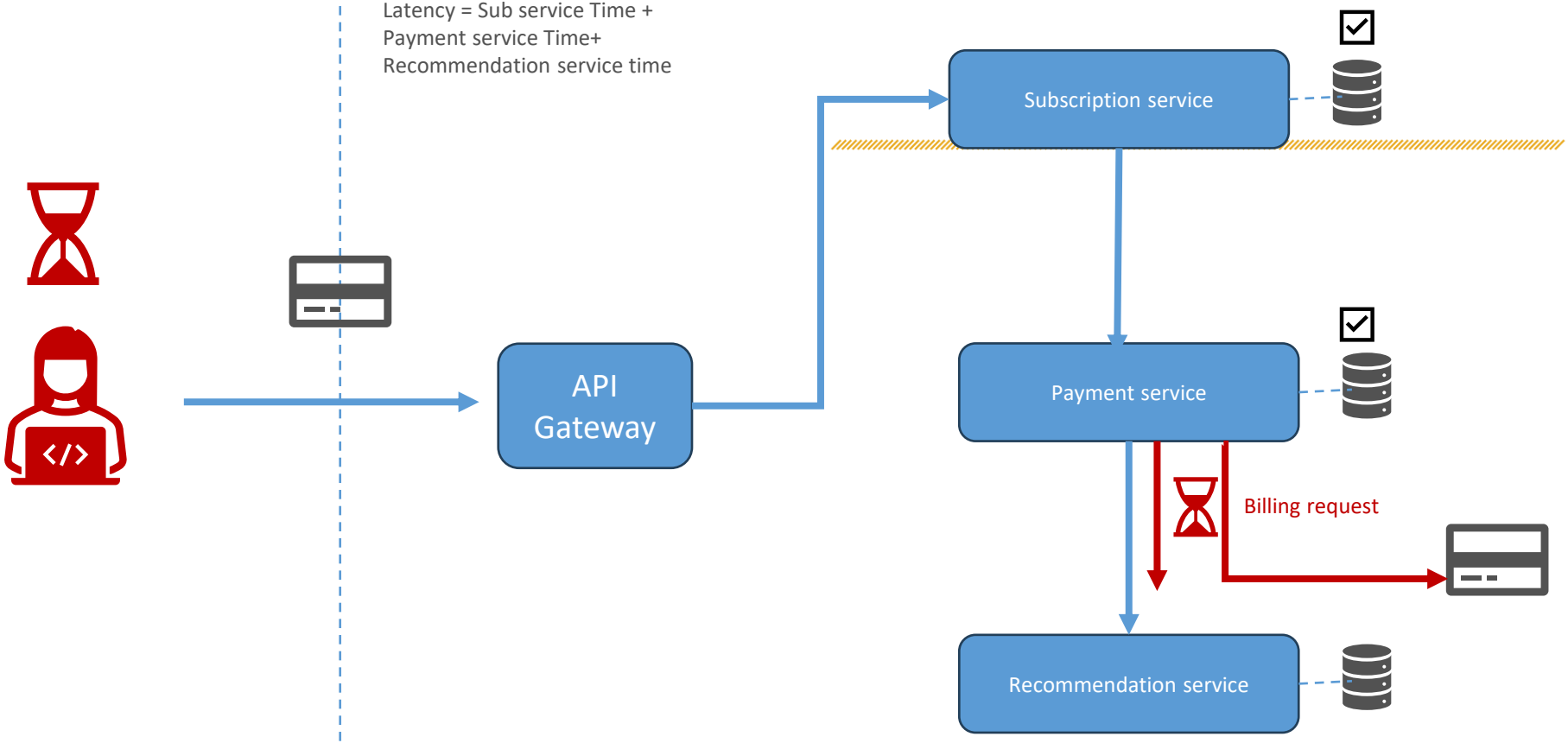
Video On Demand Subscription Service

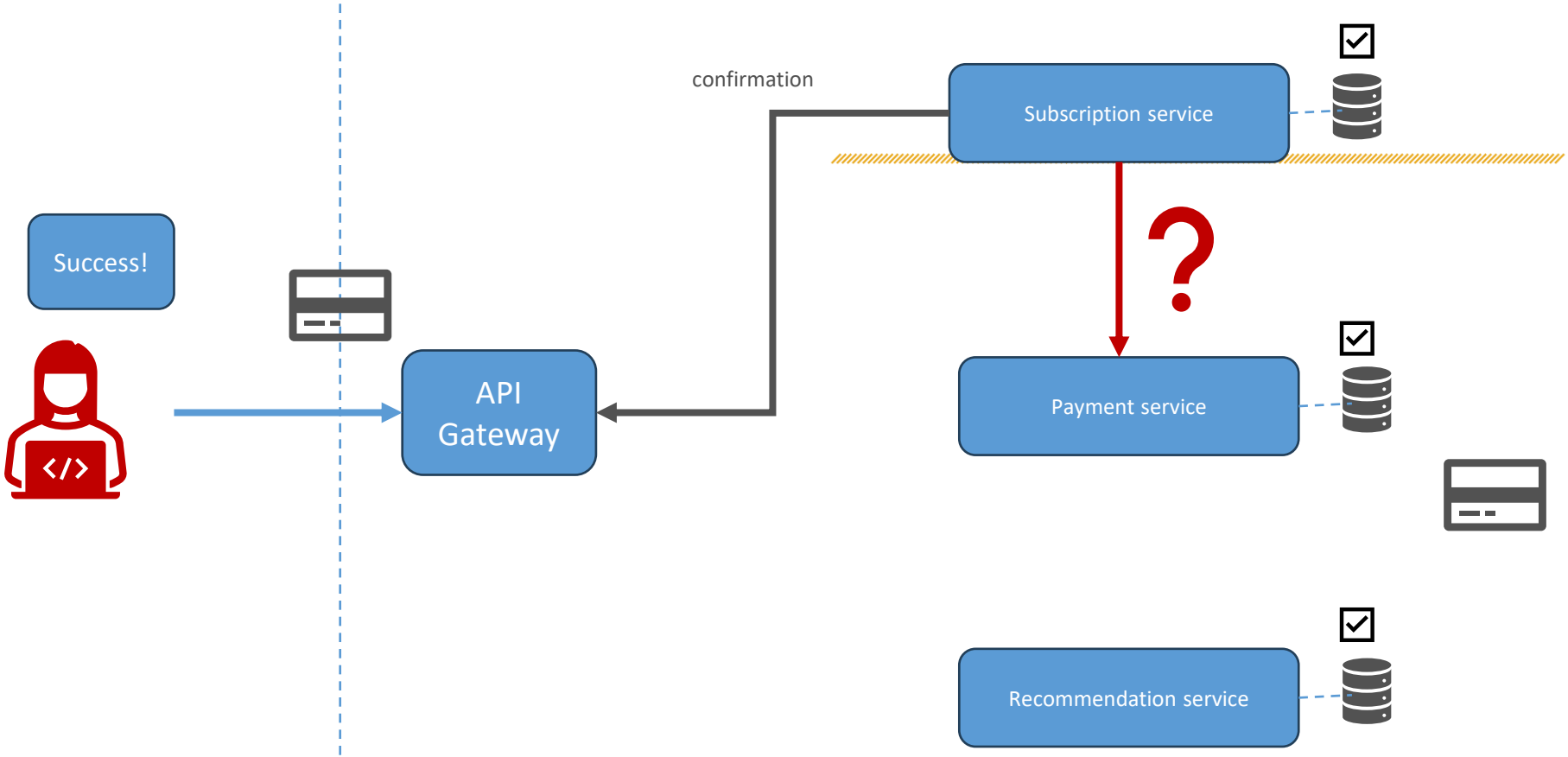


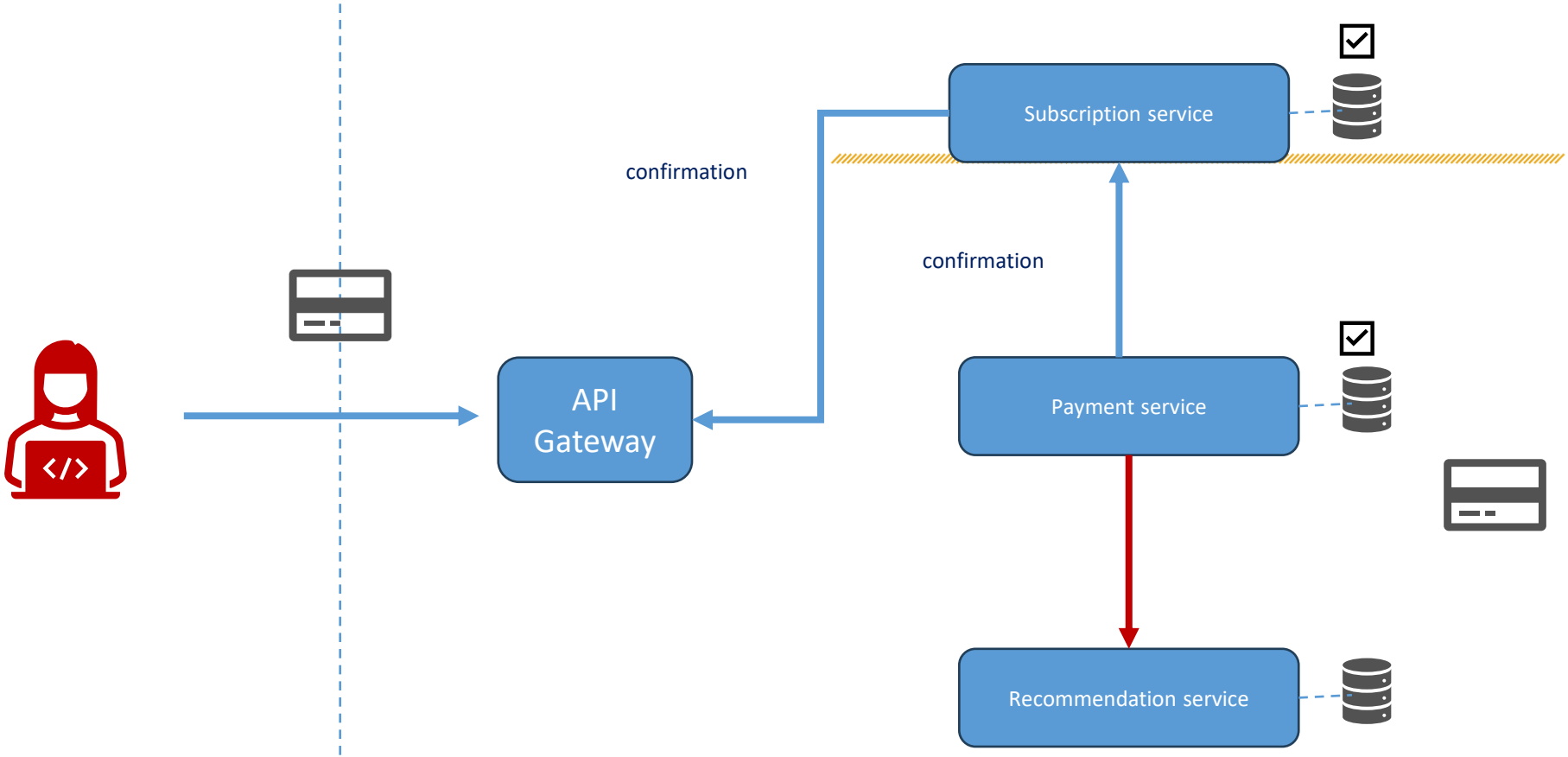
Video On demand Subscription service


Success!
API Gateway
Subscription service
Payment service
Recommendation service
1. Subscription Request
2. Payment Request
3. Billing Request
4. Profile creation Request
5. confirmation
6. confirmation
7. confirmation

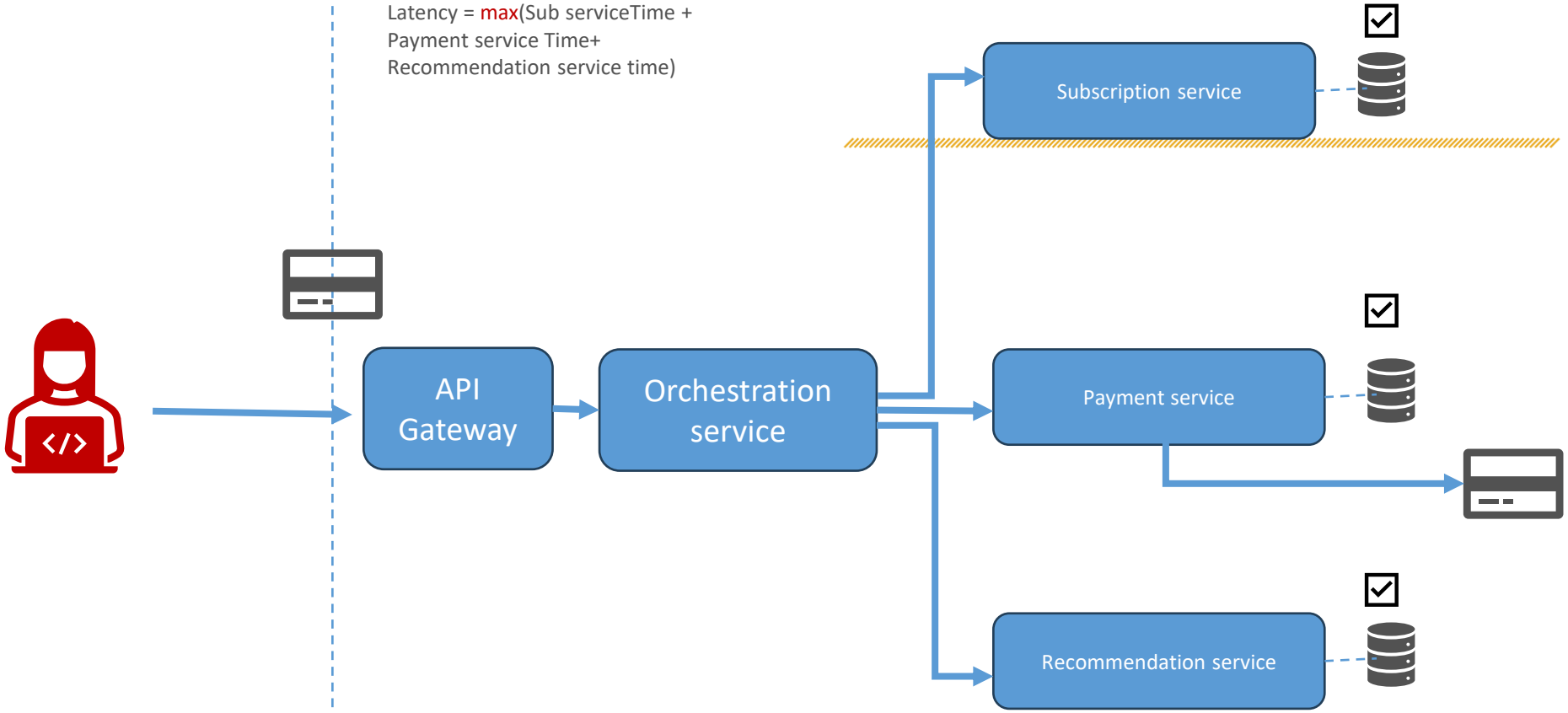

Success!
API Gateway
100% Guarantee
Subscription service
Payment service
Recommendation service

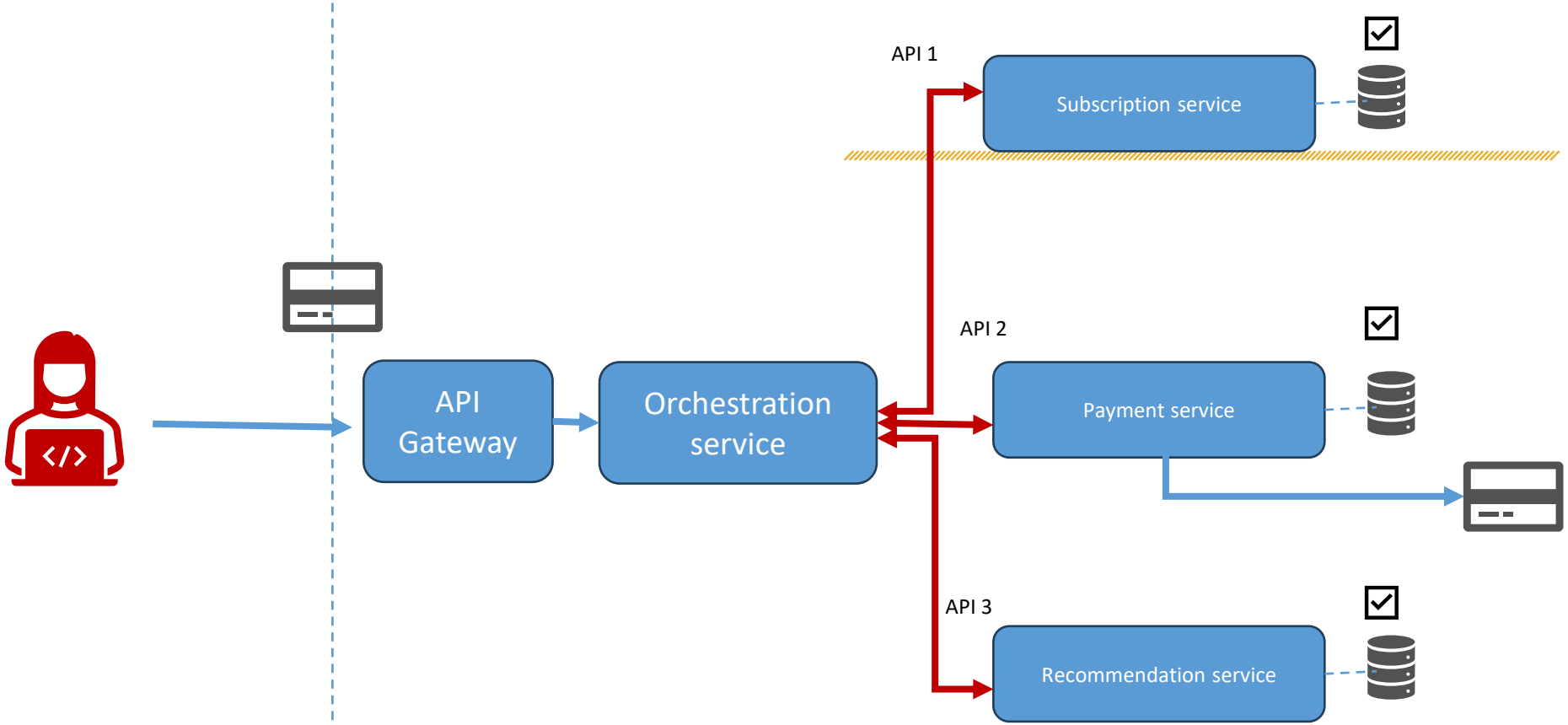

Latency = Sub service Time +
Payment service Time+
Recommendation service time
Subscription service
API Gateway
Payment service
Billing request
Recommendation service


confirmation
Subscription service
Success!
API Gateway
Payment service
Recommendation service


Subscription service
confirmation
confirmation
API
Gateway
Payment service
Recommendation service

Latency = max(Sub serviceTime +
Payment service Time+
Recommendation service time)




API 1
Subscription service
API 2
Payment service
API 3
Recommendation service
API Gateway
Orchestration service

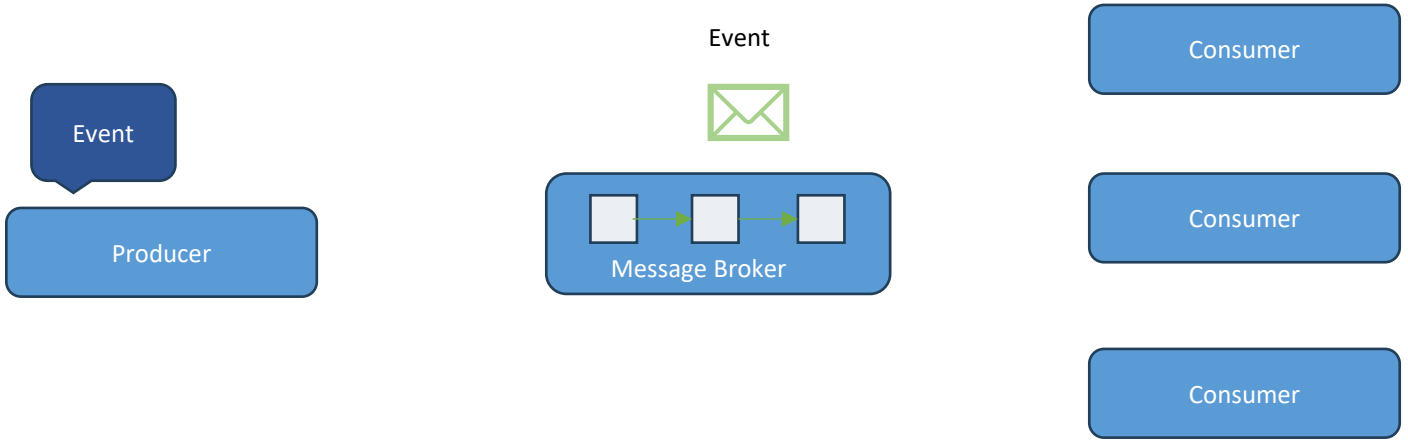
Topics

- Motivation for Event-Driven Architecture
- **Fundamental concept of Event-Driven Architecture**
- Request-Response vs Event-Driven Model
- Event-Driven Architecture-Real-Life Example

Event

- Event: Fact, Action, State Change
- Always immutable
- Can be stored indefinitely
- Can be consumed multiple times by different services

Event-Driven Architecture Participants



Topics

- Motivation for Event-Driven Architecture
- Fundamental concept of Event-Driven Architecture
- **Request-Response vs Event-Driven Model**
- Event-Driven Architecture-Real-Life Example

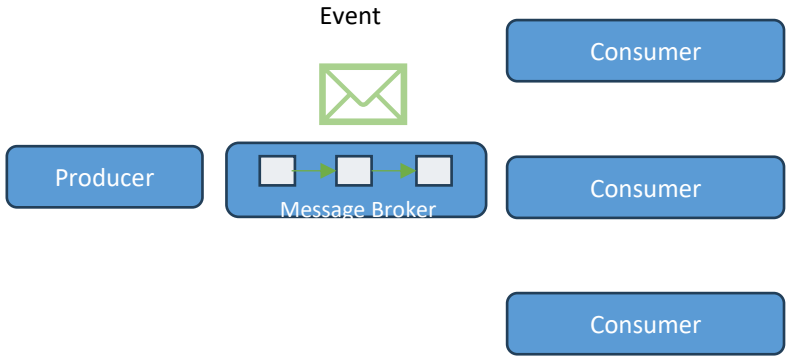
Request-Response vs Event-Driven

Request-Response Model

- Synchronous

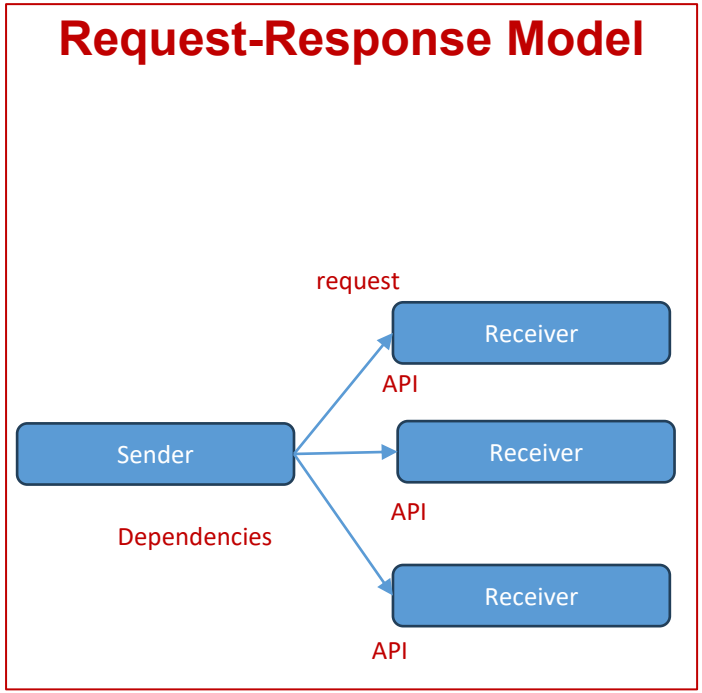
Event-Driven Model

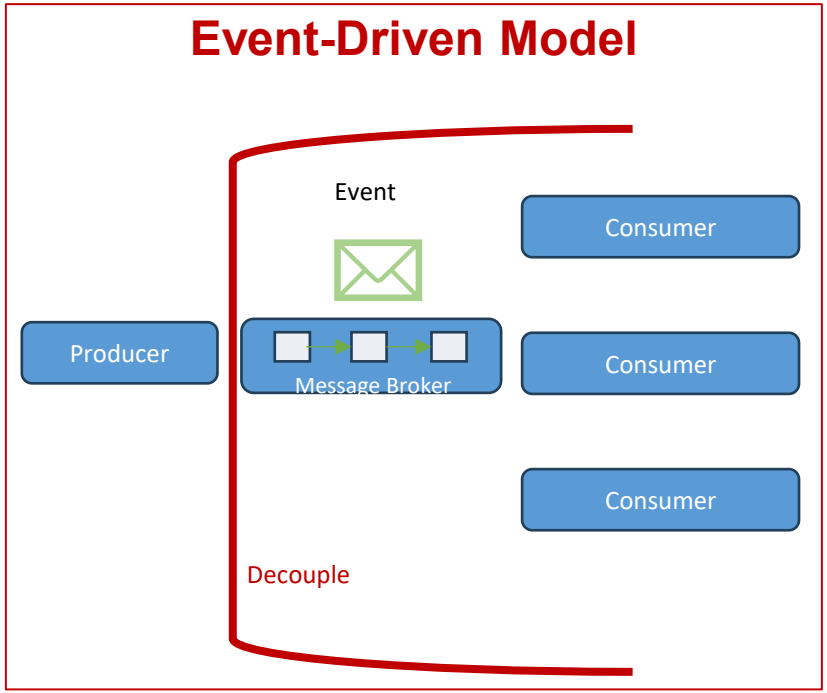
- Asynchronous



Request-Response vs Event-Driven

Inversion of control



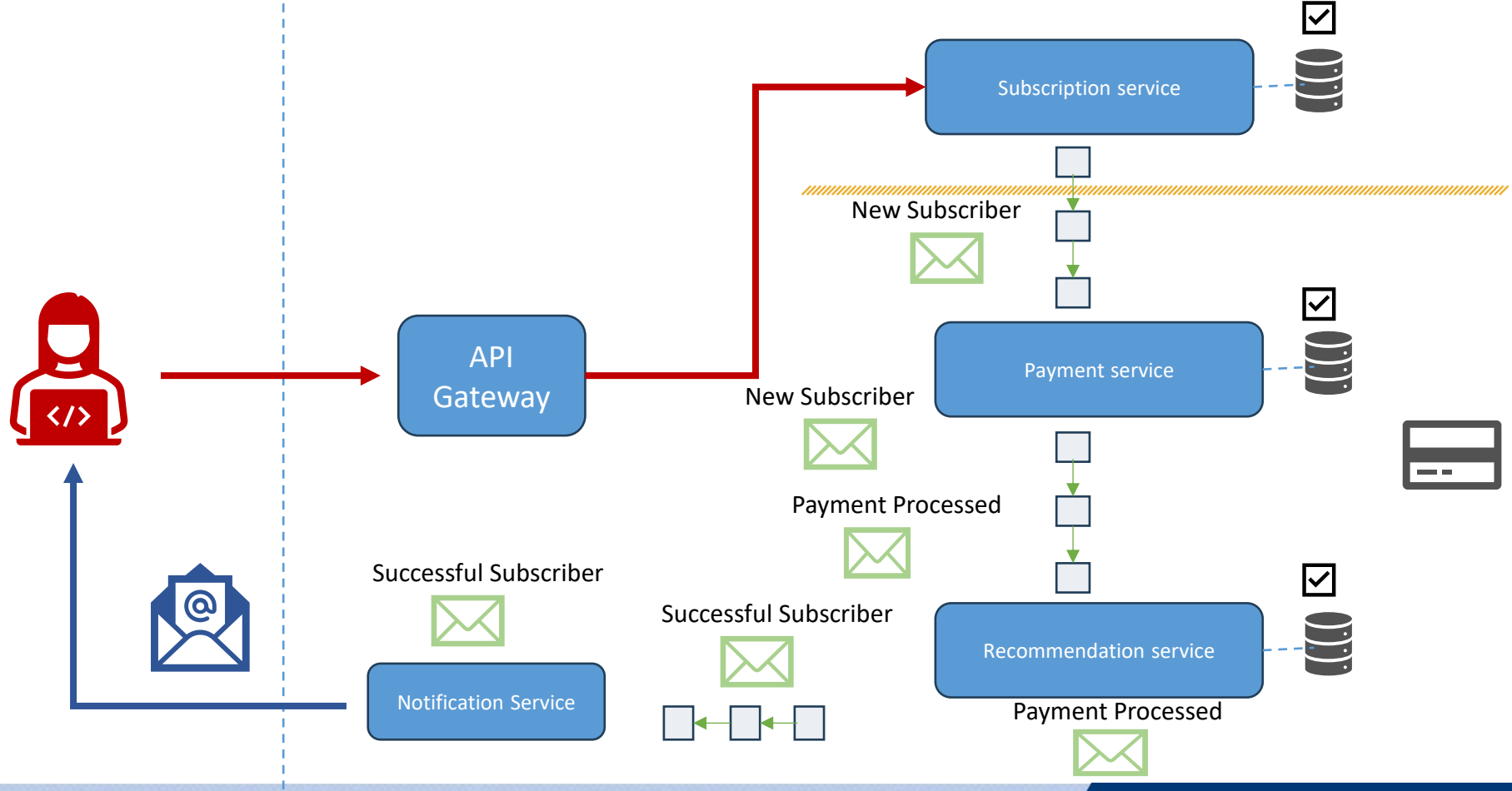


Request-Response vs Event-Driven

- Synchronous vs Asynchronous
- Inversion of Control
- Loose Coupling

Topics

- Motivation for Event-Driven Architecture
- Fundamental concept of Event-Driven Architecture
- Request-Response vs Event-Driven Model
- **Event-Driven Architecture-Real-Life Example**


Subscription service
New Subscriber
API Gateway
Payment service
New Subscriber
Payment Processed
Successful Subscriber
Successful Subscriber
Notification Service
Recommendation service
Payment Processed

The Saga Pattern

Lecture Roadmap

- Problem we're solving
- Introduction to the Saga pattern
- Two implementation approaches
- Example scenario
- Summary

The Problem

- Monolithic: Single database, ACID transactions
- Microservices: One DB per microservice
- Issue: No atomic transactions across services

ACID Transactions Recap

- Atomicity: All or nothing
- Consistency: Data remains valid
- Isolation: Concurrent transactions don't interfere
- Durability: Changes persist

Enter the Saga Pattern

- Solution: Distributed transactions via local transactions
- Each operation triggers the next
- Compensation for failures

Implementation Options

1. Orchestration-based (Workflow Management Service)
2. Choreography-based (Event-driven)

Example: Vacation Booking Service

- Payment Service
- Flight Reservation Service
- Hotel Booking Service
- Car Rental Service
- Order Service

Booking Workflow

API Gateway

Workflow Orchestration Service

1. Secure payment
2. Book flights
3. Book hotel
4. Book car rental
5. Insert order details

The payment service

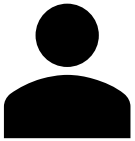
Flight reservation service

The hotel booking service

The car rental service

The order service

Compensating Operations



API Gateway

Workflow Orchestration Service

1. Secure payment
   Credit payment
2. Book flights
   Cancel flights
3. Book hotel
   Cancel hotel
4. Book car rental
   Cancel car rental
5. Insert order details
   Update order to “canceled”

The payment service

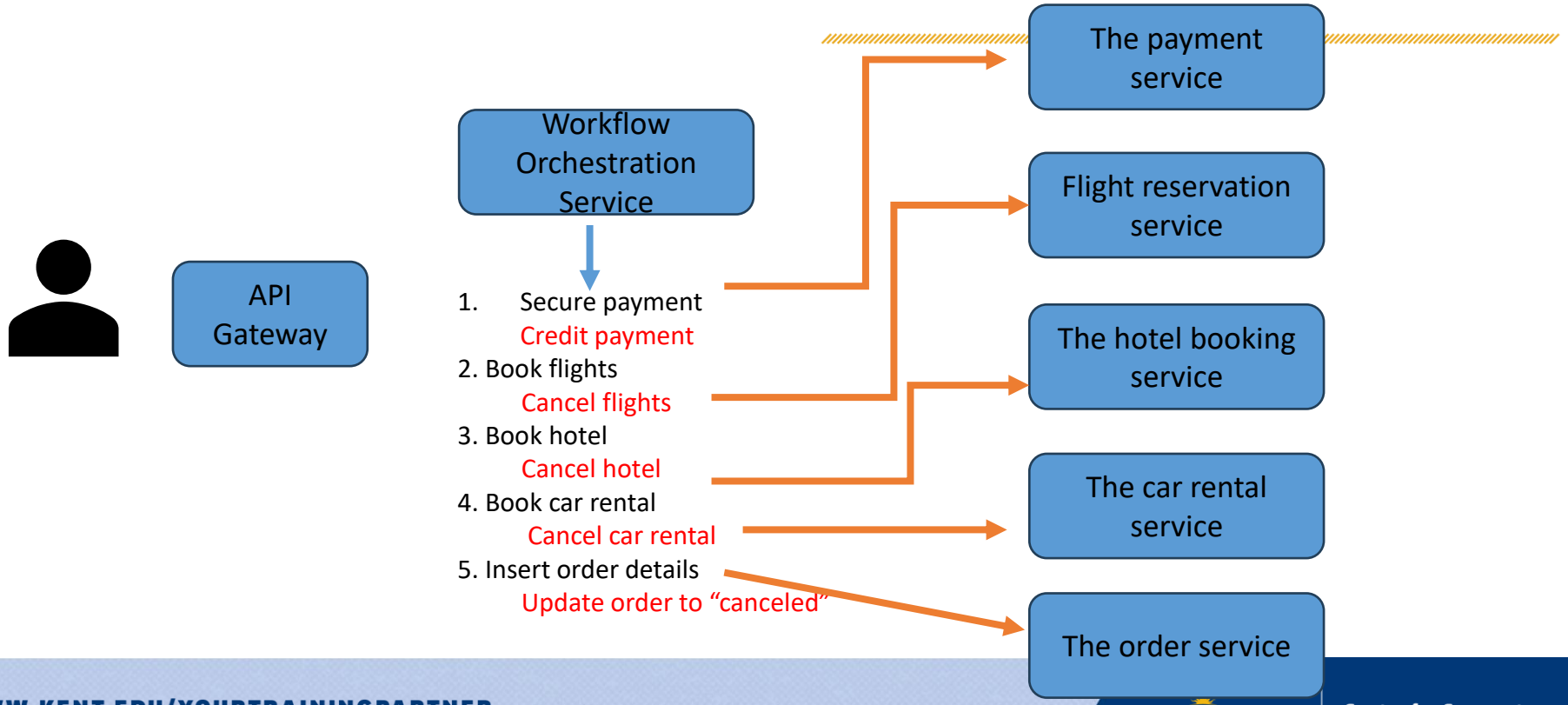
Flight reservation service

The hotel booking service

The car rental service

The order service

Compensating Operations



Orchestration Flow

- User submits booking ➜ Orchestration Service
- Executes steps in order
- On success: Confirm booking
- On failure: Compensate & inform user

Failure Example

- **Car booking fails → Orchestration Service:**
  - Cancels hotel
  - Cancels flights
  - Credits payment
  - Sends error to user

Choreography-based Implementation

- No central orchestrator
- Services communicate via events
- Each service knows next step and compensation

Choreography Flow

1. User → Payment Service
2. Payment emits event
3. Flight Service books flight
4. Hotel Service books hotel
5. Car Rental Service books car
6. Order Service finalizes
7. Notification Service updates user

Failure Example

- Car booking fails → Failure event
- Hotel cancels booking
- Flight cancels booking
- Payment refunds
- Notification Service informs user

10 min break ☺

Testing for Microservices

Testing Pyramid

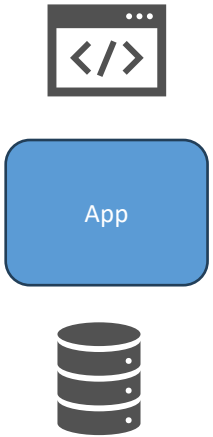
Topics

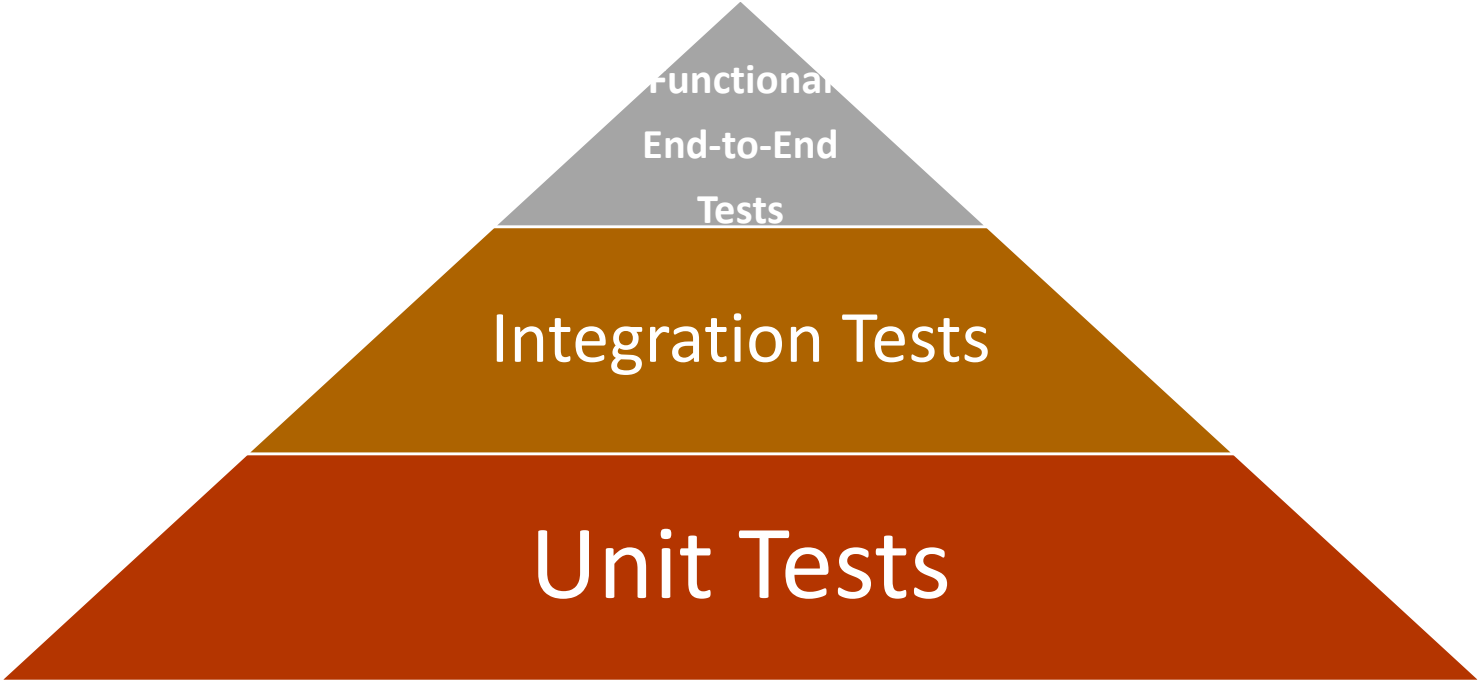
- Testing Pyramid for Monolithic Applications
- Applying the Testing Pyramid to Microservices Architecture
- Challenges of Testing Microservices and Event-Driven Architecture

Topics

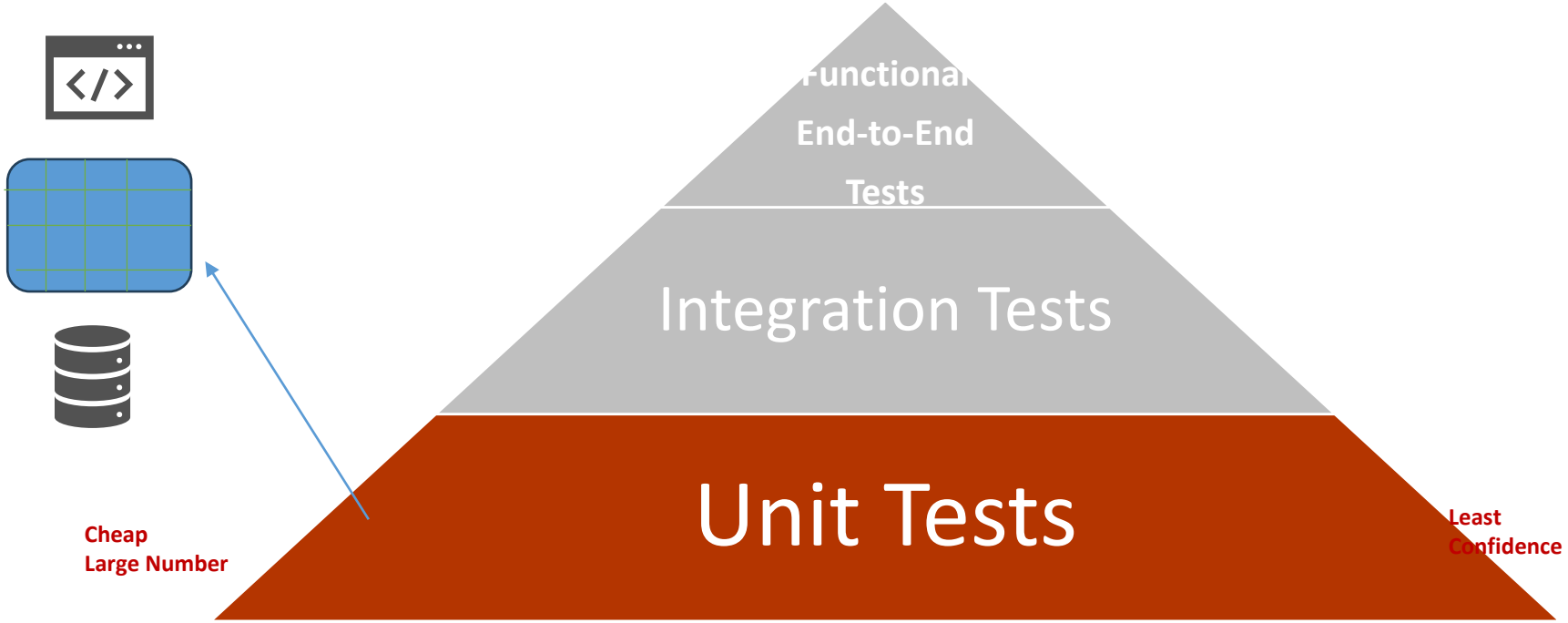
- **Testing Pyramid for Monolithic Applications**
- Applying the Testing Pyramid to Microservices Architecture
- Challenges of Testing Microservices and Event-Driven Architecture

Testing Pyramid for Monolithic Application

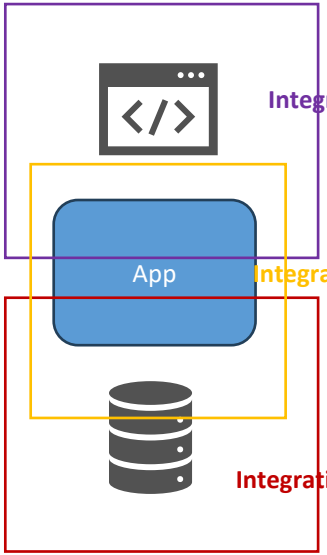




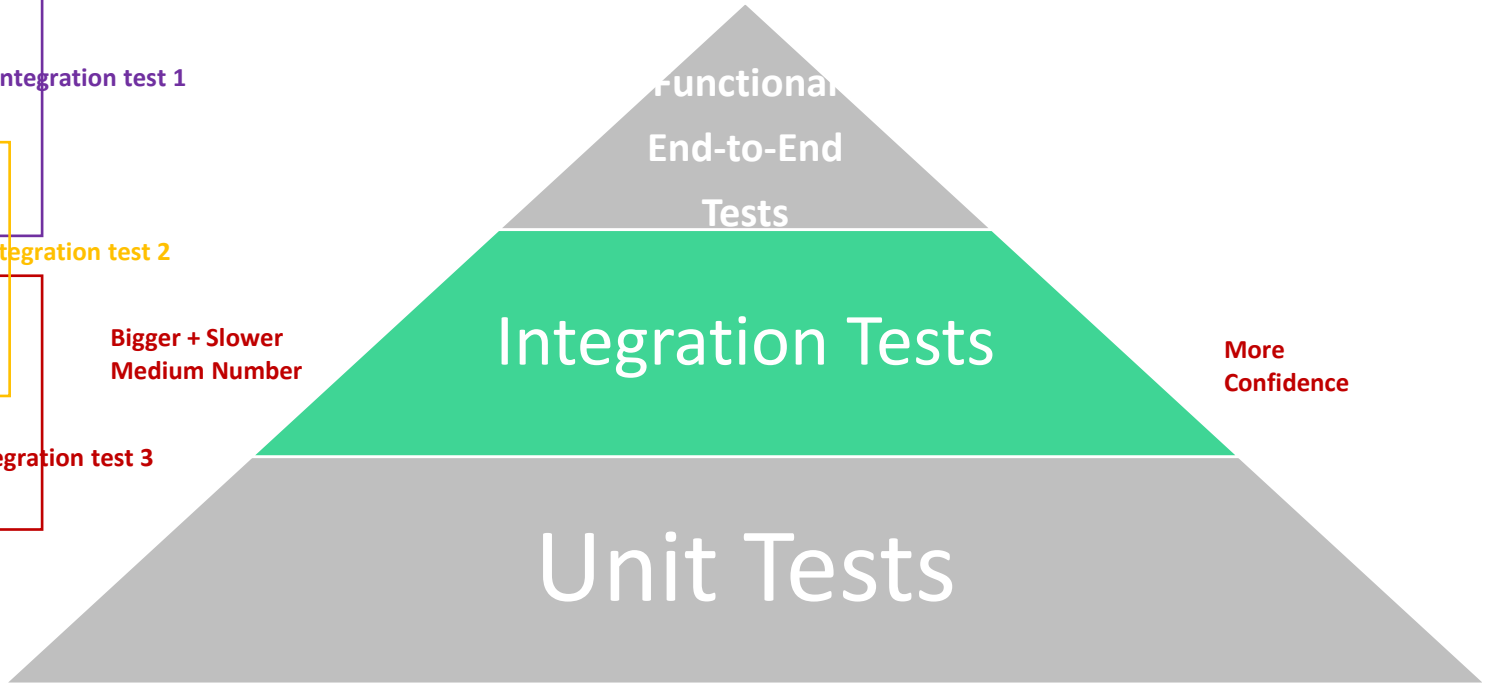
Testing Pyramid for Monolithic Application



Testing Pyramid for Monolithic Application

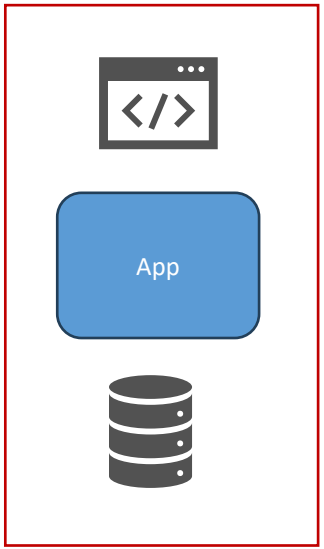


Bigger + Slower
Medium Number



More
Confidence

Testing Pyramid for Monolithic Application



User Journey 1

User Journey 2

User Journey 3

Slowest

Least Number

Functional End-to-End Tests

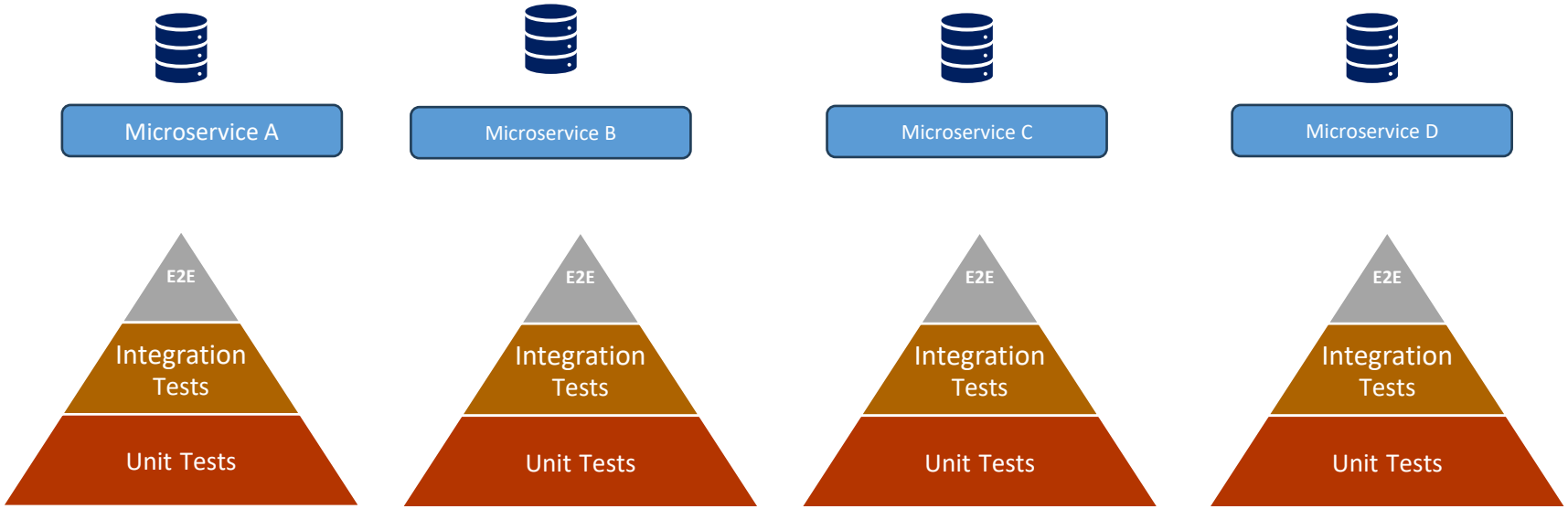
Highest Confidence

Integration Tests

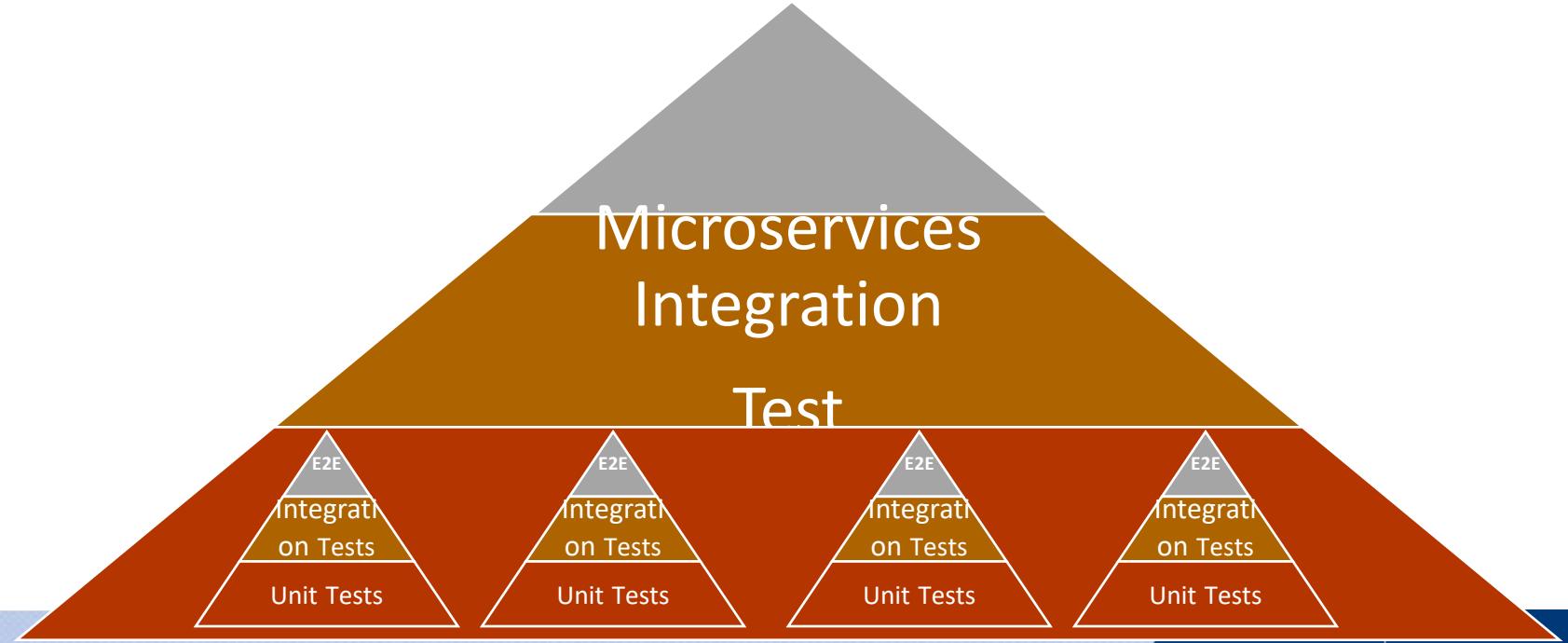
Unit Tests

Topics

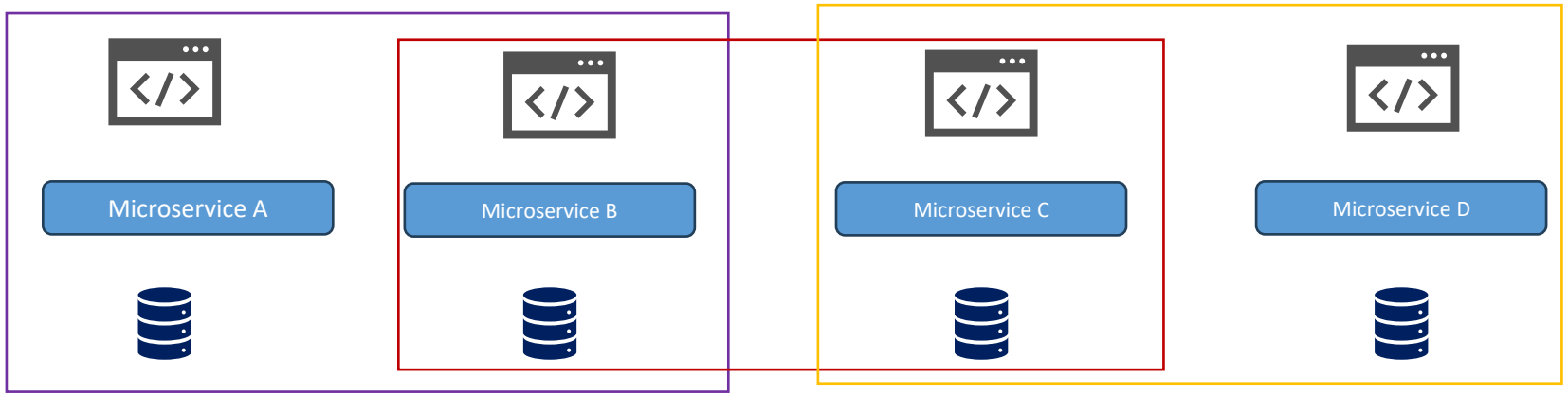
- Testing Pyramid for Monolithic Applications
- **Applying the Testing Pyramid to Microservices Architecture**
- Challenges of Testing Microservices and Event-Driven Architecture


Microservice A
Microservice B
Microservice C
Microservice D
E2E
Integration Tests
Unit Tests
E2E
Integration Tests
Unit Tests
E2E
Integration Tests
Unit Tests
E2E
Integration Tests
Unit Tests

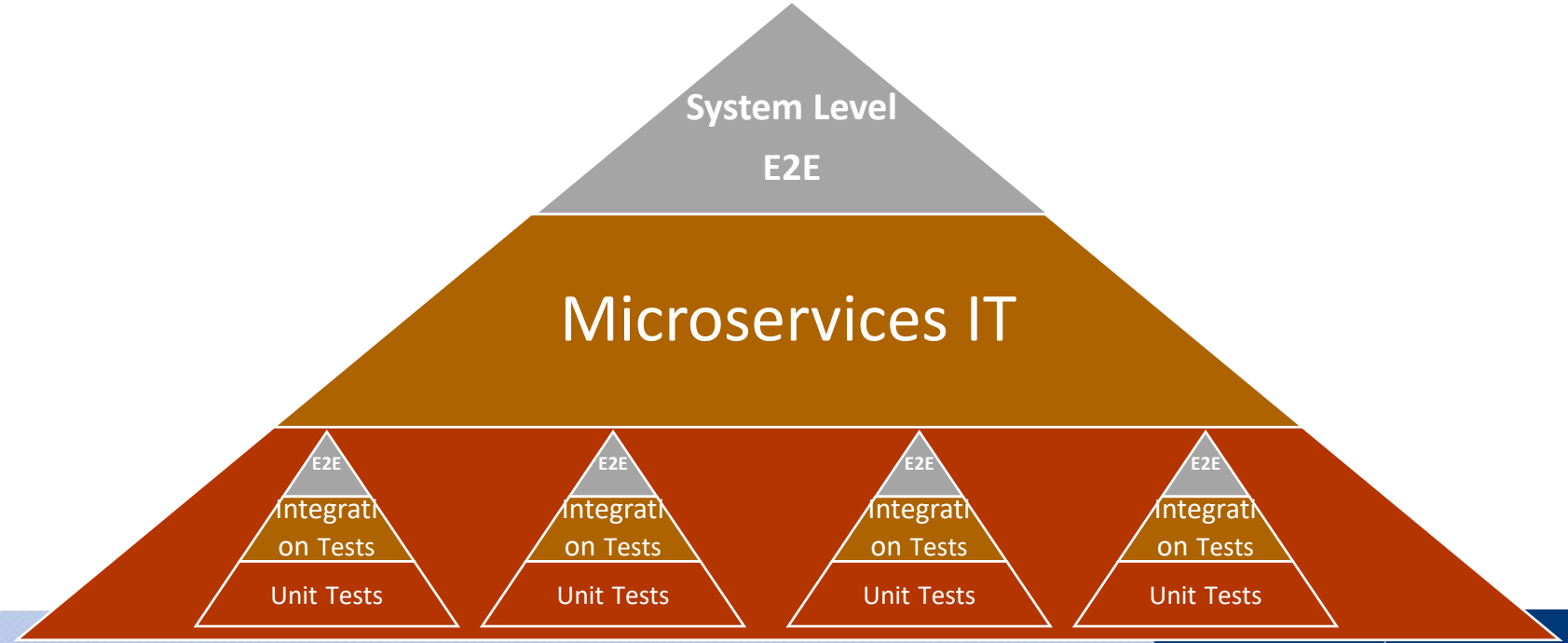
Testing Pyramid for Microservices



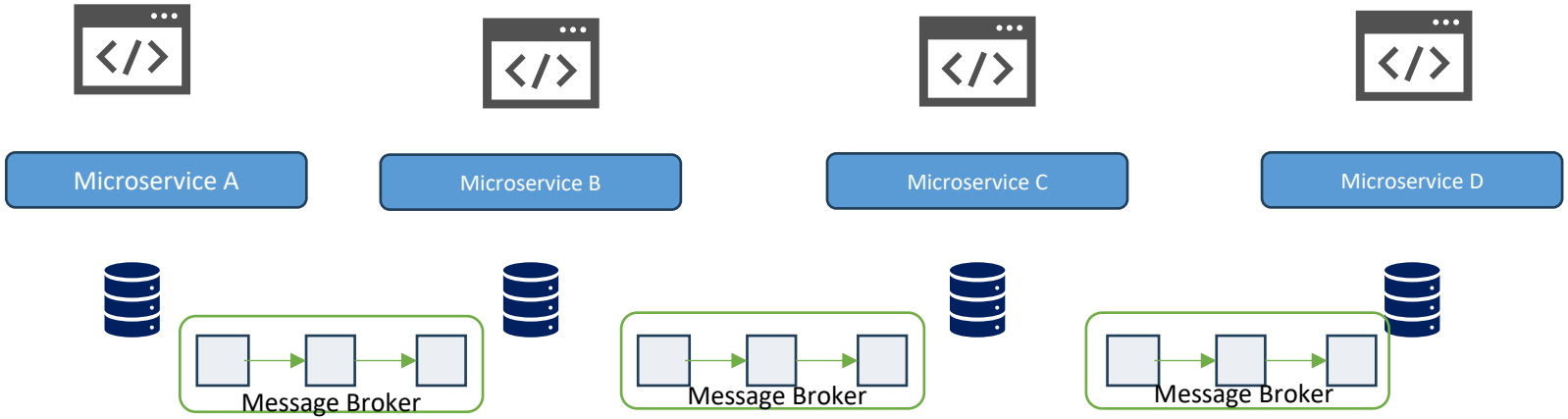
Microservices Integration Test



Microservices Functional / E2E Test



Microservices E2E Test



Topics

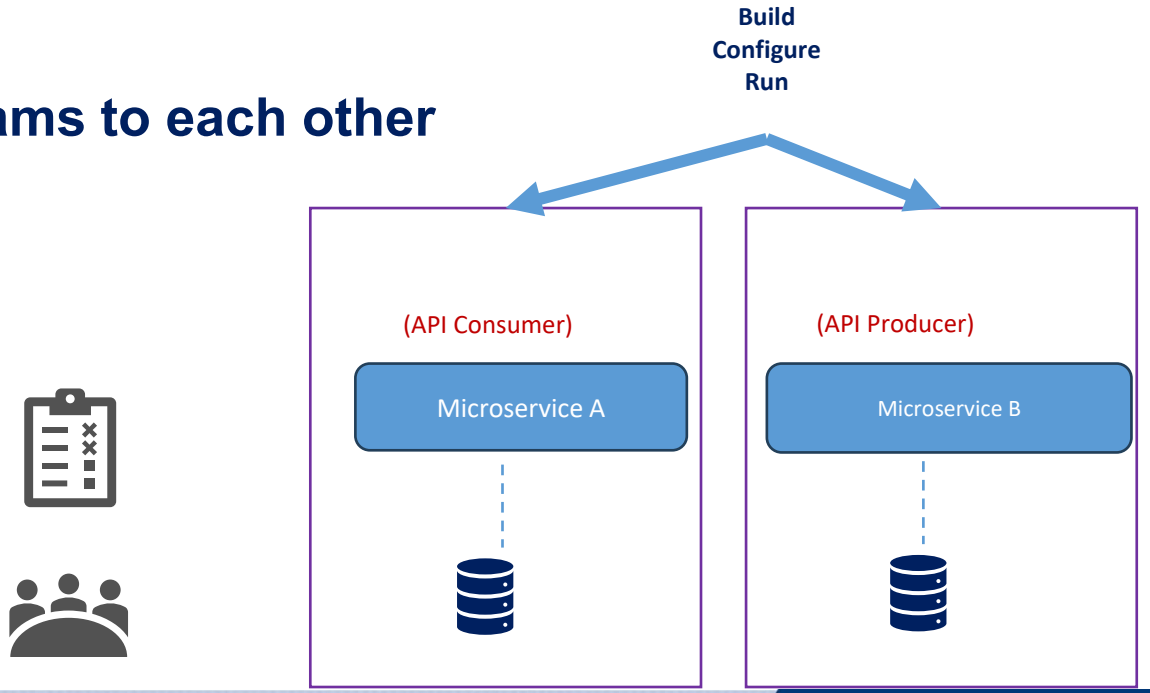
- Testing Pyramid for Monolithic Applications
- Applying the Testing Pyramid to Microservices Architecture
- **Challenges of Testing Microservices and Event-Driven Architecture**

Challenge 1 – End 2 End Tests

- **Hard to:**
  - Set up
  - Maintain
- **No clarity about ownership**
- **One team may block everyone**
- **Low confidence (ignoring failed tests_**
- **Very costly**

Challenge 2 – Integration Tests

- Difficult to run
- Tightly couples' teams to each other



Summary

- **Revisited the testing pyramid in a monolithic architecture**
- **Apply the testing pyramid to microservices**
- **Challenges of testing microservices**
  - High cost and complexity of end-to-end tests
  - Tight coupling of the integration test
  - Complexity and overhead of testing event-driven microservices

5 min break ☺

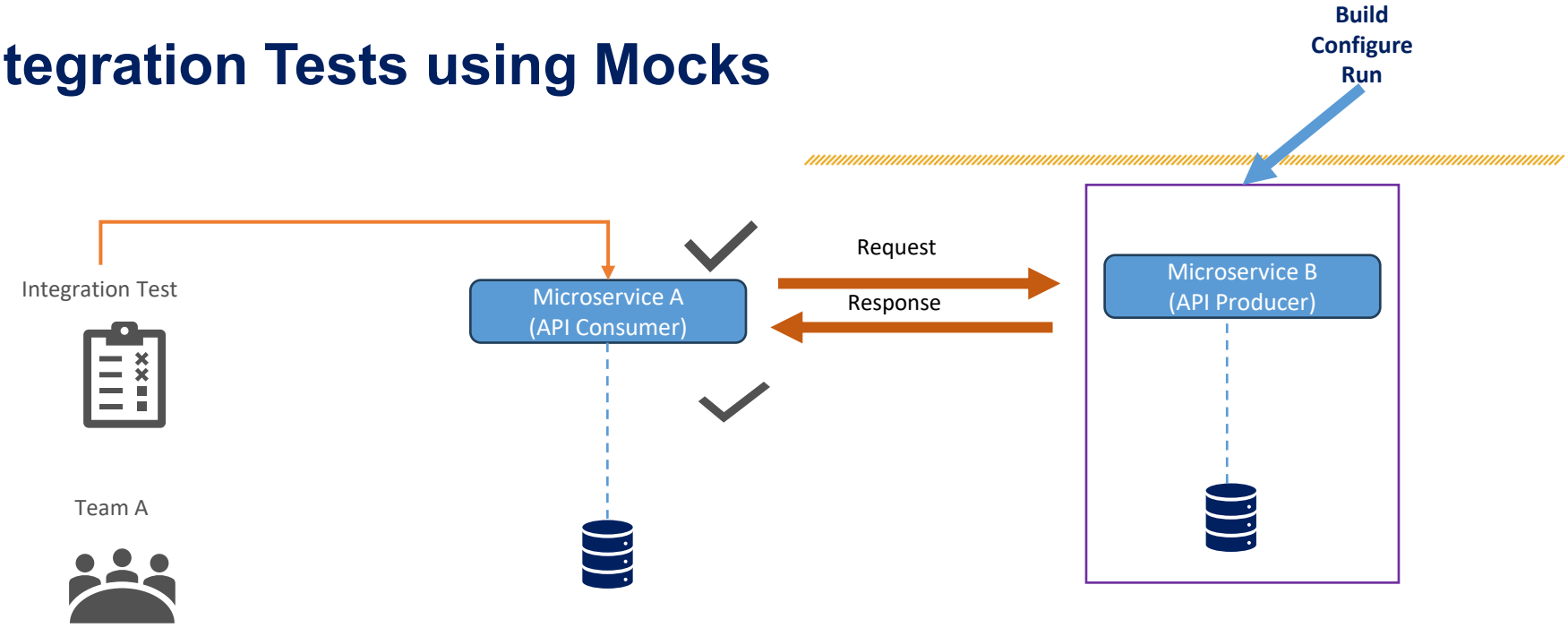
Microservices Tests

Contract Test And Production Testing

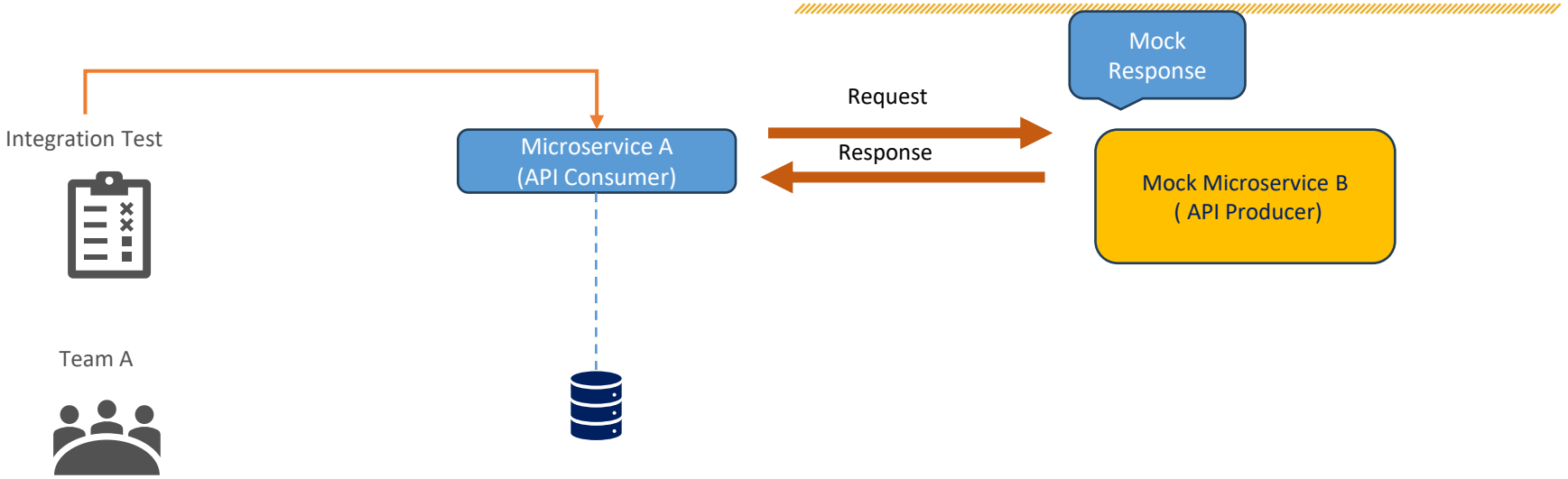
Topics

- **Integrating Tests Using Lightweight Mocks**
- Contract Tests For Synchronous Communication
- Contract Test For Asynchronous Communication
- Production Testing Using Blue/Green Deployment And Canary Testing

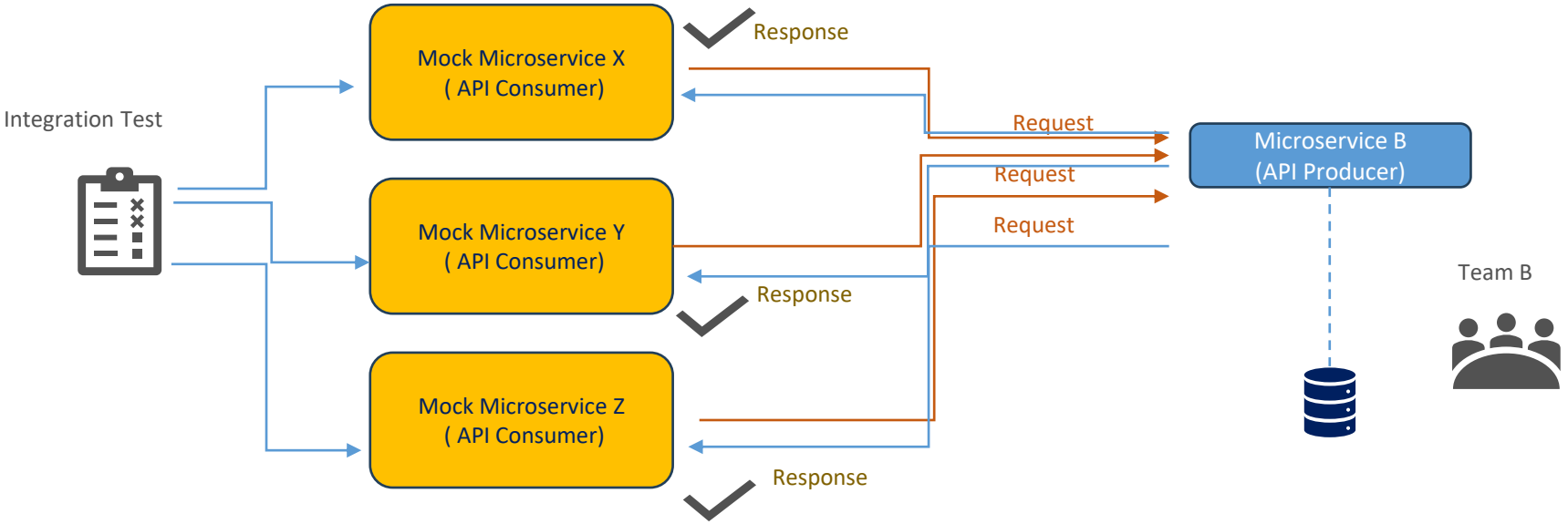
Integration Tests using Mocks



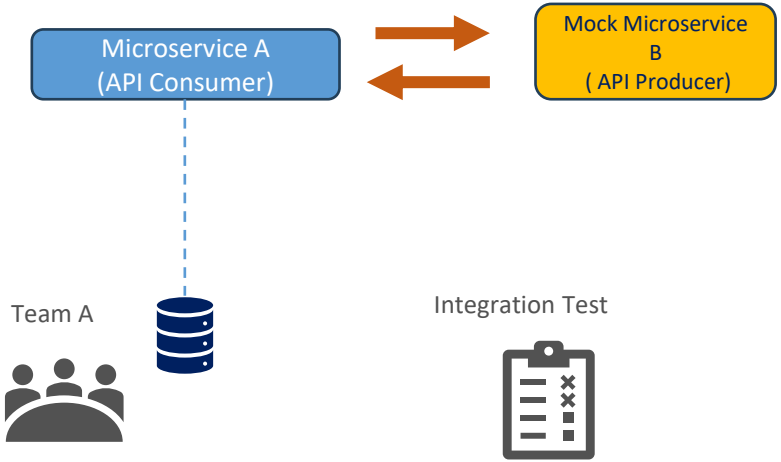
Integration Tests using Mocks

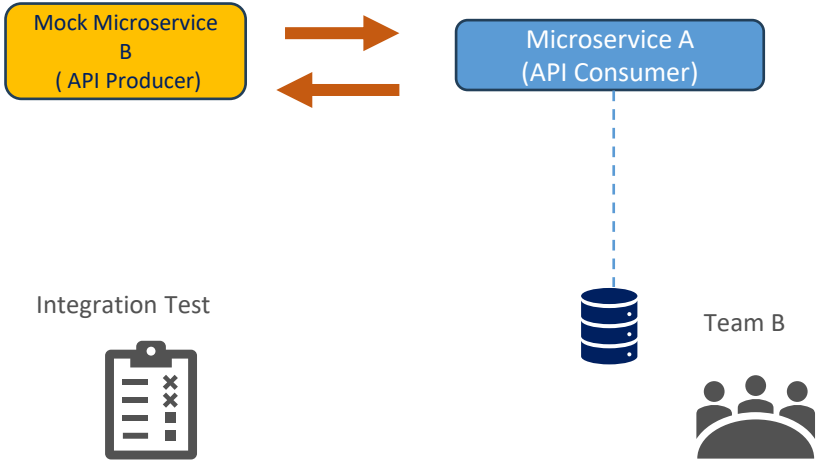


Integration Tests using Mocks



Integration Tests using Mocks

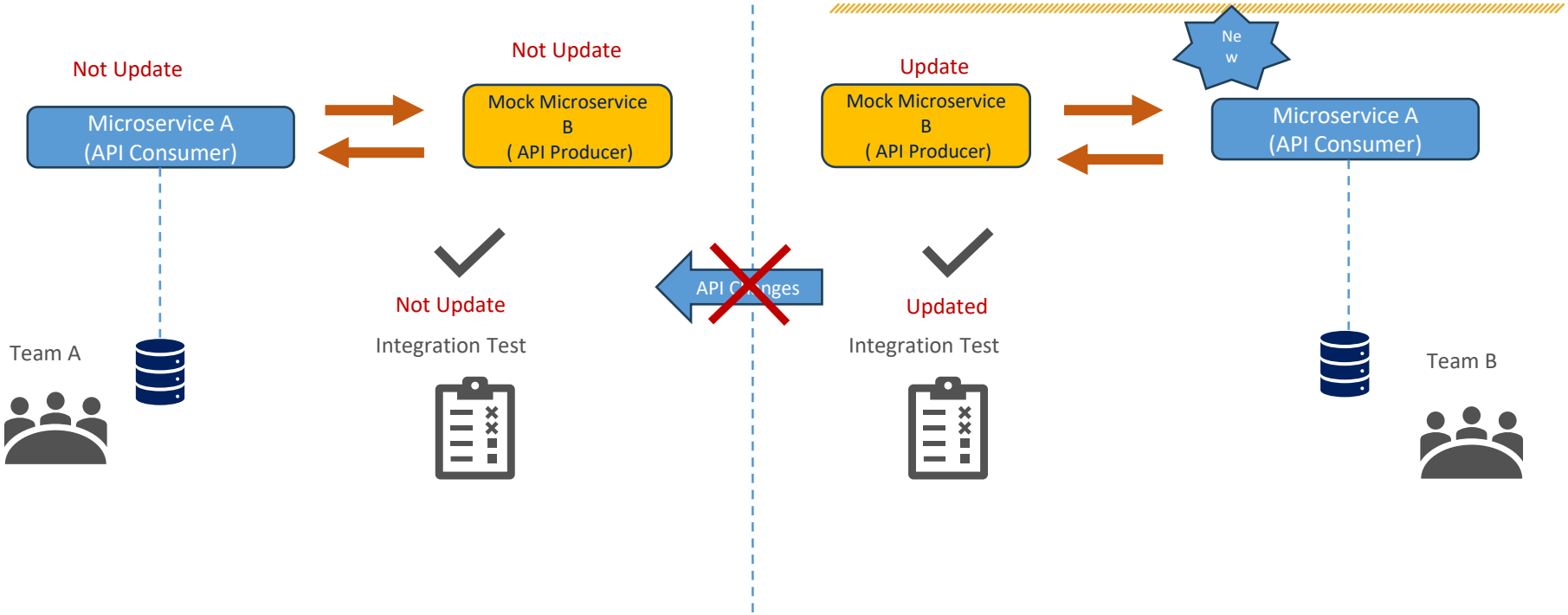




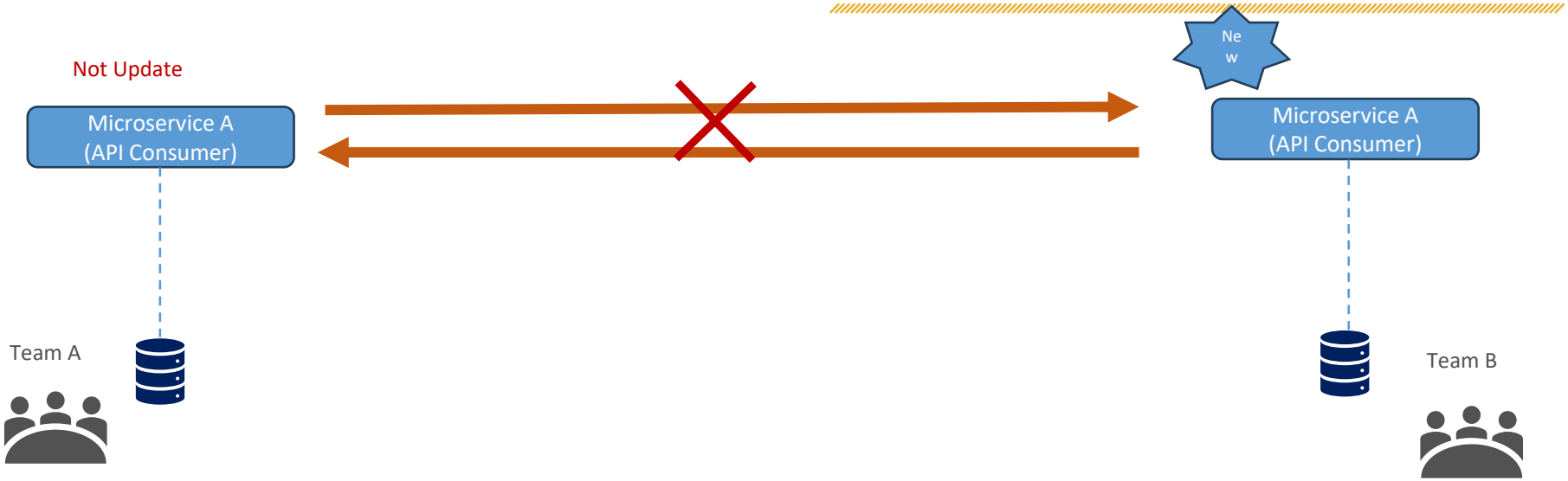
Integration Tests using Mock- Issues

- The contract between API consumer and API provider can get out of sync.

Integration Tests using Mocks - Issues

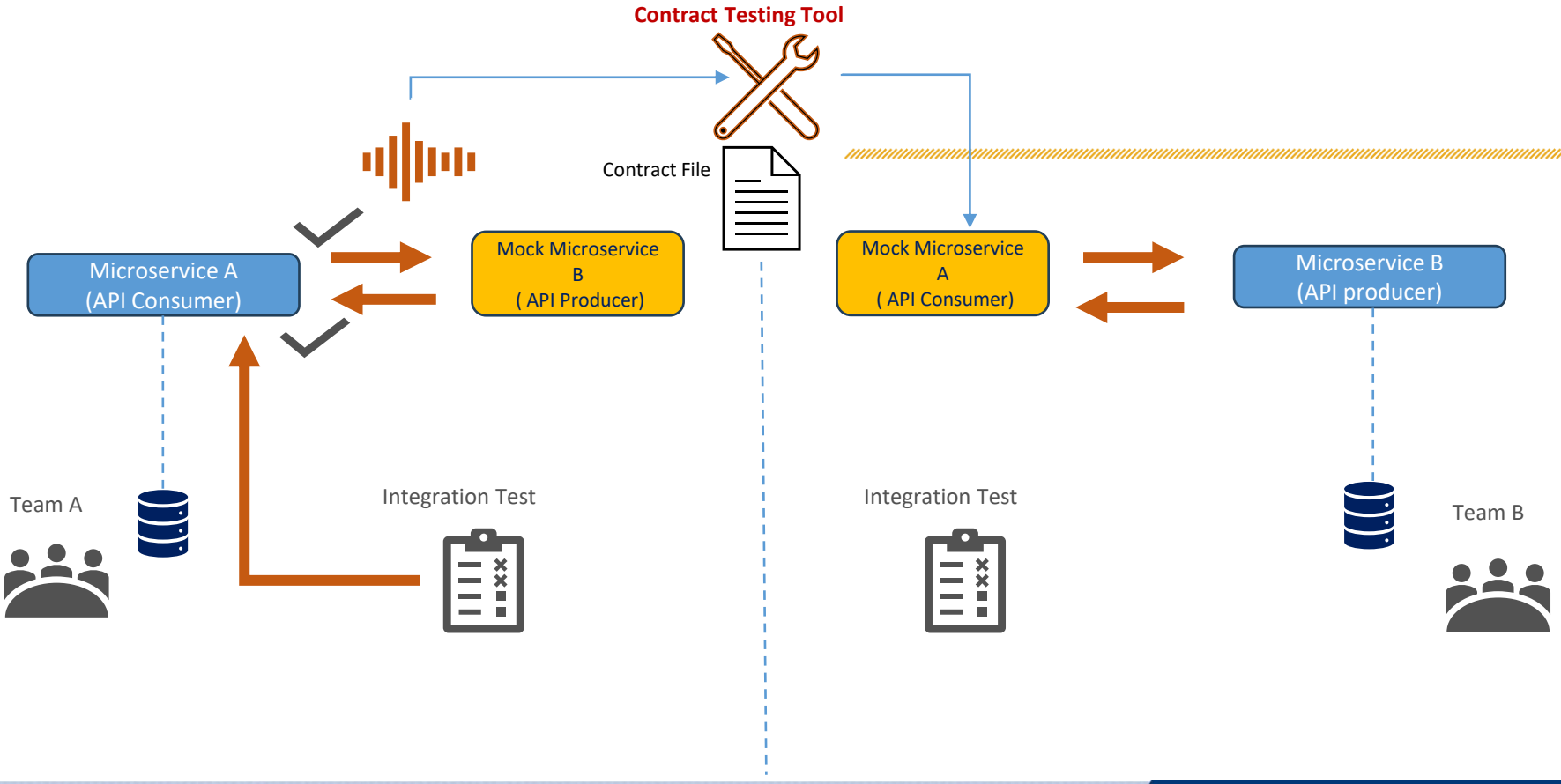


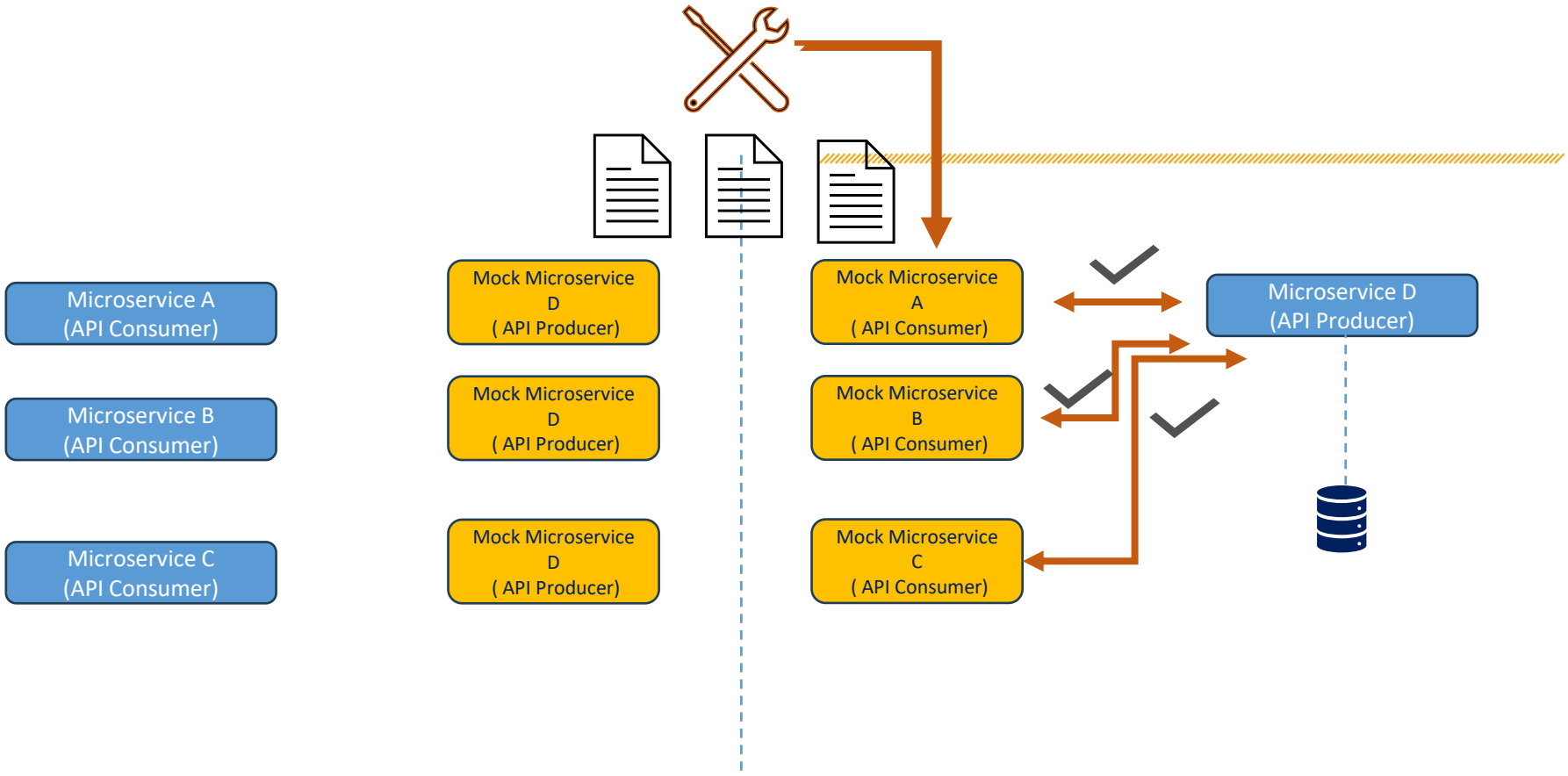
Integration Tests using Mocks - Issues



Topics

- Integrating Tests Using Lightweight Mocks
- **Contract Tests For Synchronous Communication**
- Contract Test For Asynchronous Communication
- Production Testing Using Blue/Green Deployment And Canary Testing


Contract Testing Tool
Contract File
Microservice A
(API Consumer)
Mock Microservice
B
(API Producer)
Mock Microservice
A
(API Consumer)
Microservice B
(API producer)
Team A
Integration Test
Integration Test
Team B


Microservice A
(API Consumer)
Microservice B
(API Consumer)
Microservice C
(API Consumer)
Mock Microservice D
(API Producer)
Mock Microservice D
(API Producer)
Mock Microservice D
(API Producer)
Mock Microservice A
(API Consumer)
Mock Microservice B
(API Consumer)
Mock Microservice C
(API Consumer)
Microservice D
(API Producer)

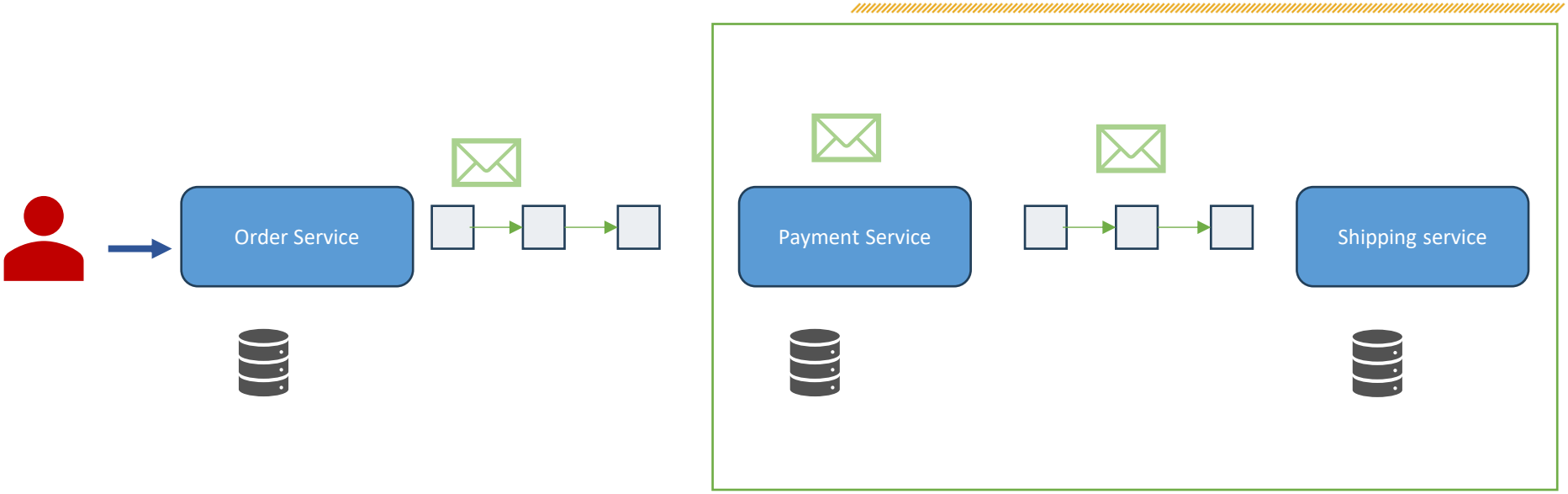
Contract Tests

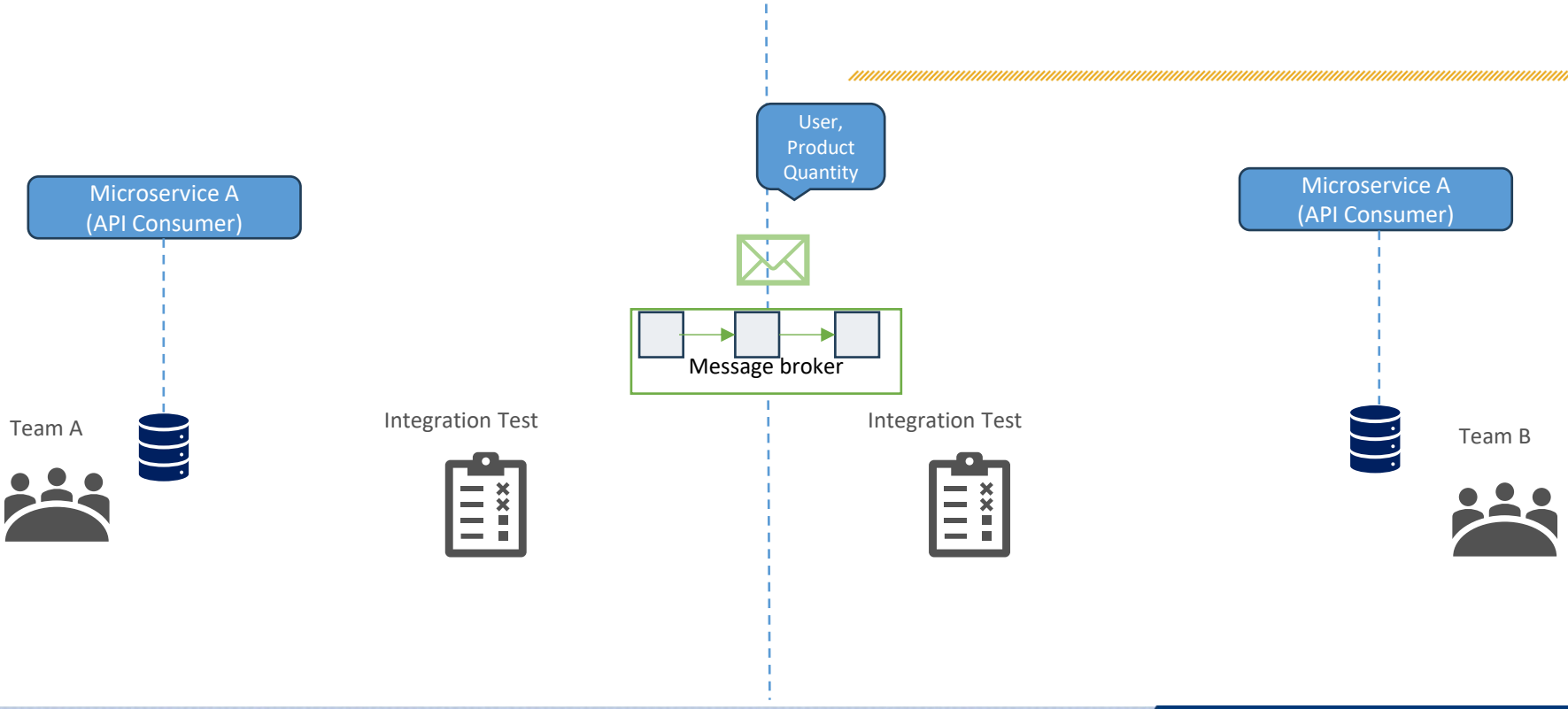
- Each Team runs its integration tests
- Contract test tools keep the contract between microservices in sync

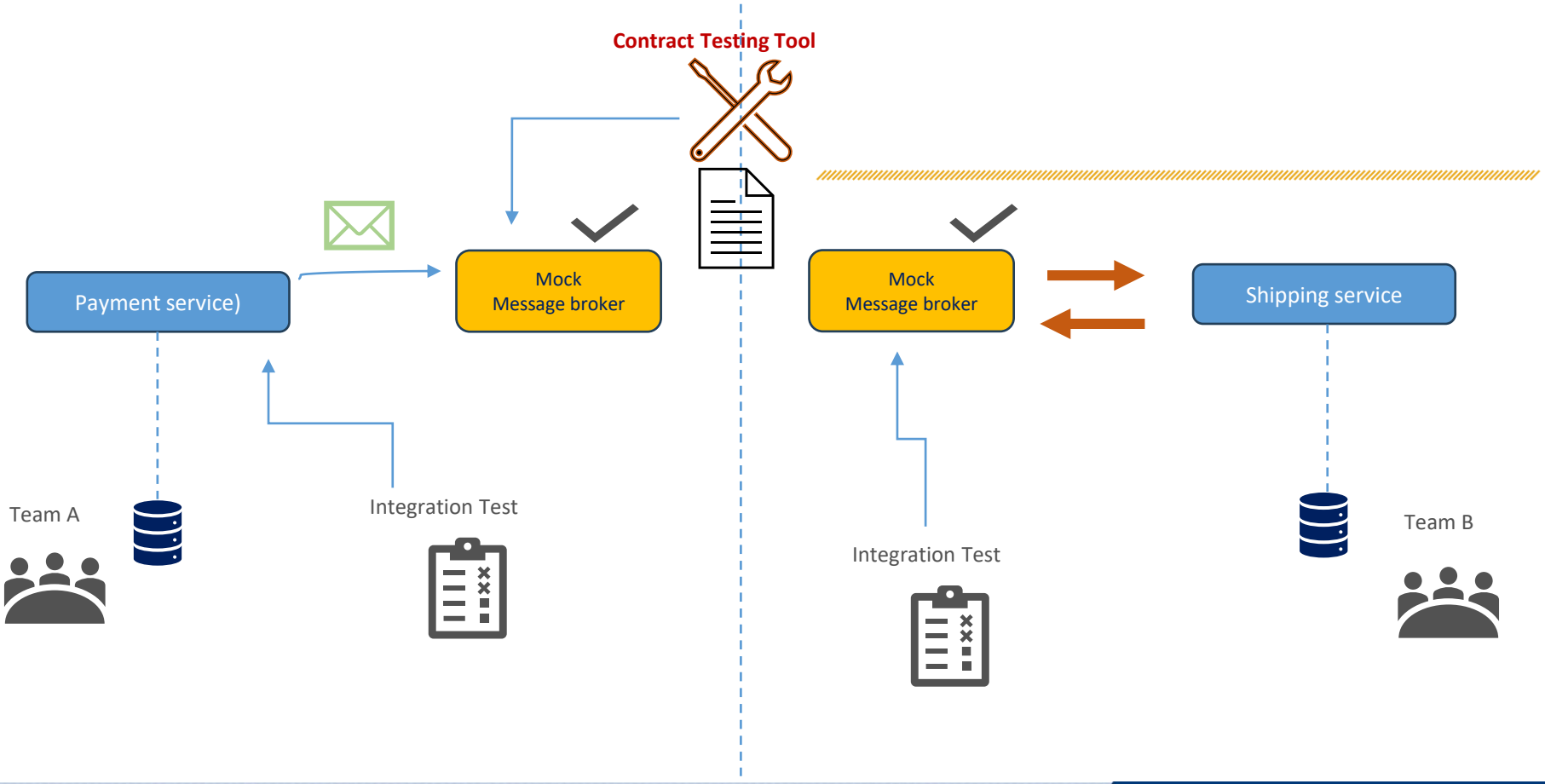
Topics

- Integrating Tests Using Lightweight Mocks
- Contract Tests For Synchronous Communication
- **Contract Test For Asynchronous Communication**
- Production Testing Using Blue/Green Deployment And Canary Testing

E-Commerce Example

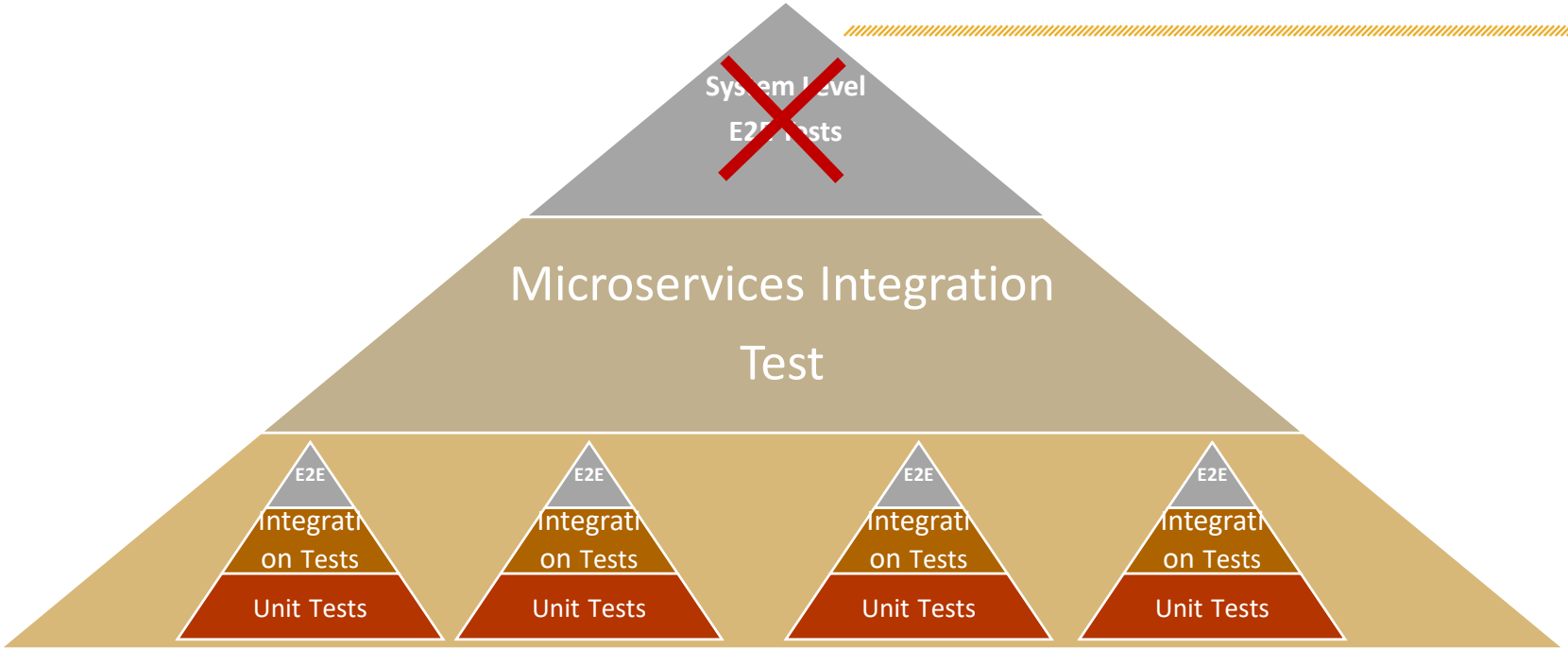



Microservice A
(API Consumer)
User,
Product
Quantity
Microservice A
(API Consumer)
Message broker
Team A
Integration Test
Integration Test
Team B


Contract Testing Tool
Payment service)
Mock
Message broker
Mock
Message broker
Shipping service
Team A
Integration Test
Integration Test
Team B

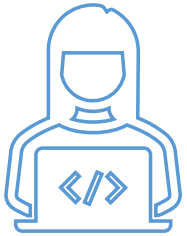
Topics

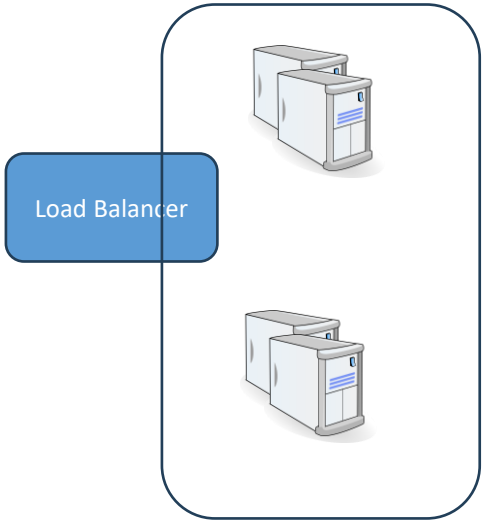
- Integrating Tests Using Lightweight Mocks
- Contract Tests For Synchronous Communication
- Contract Test For Asynchronous Communication
- **Production Testing Using Blue/Green Deployment And Canary Testing**

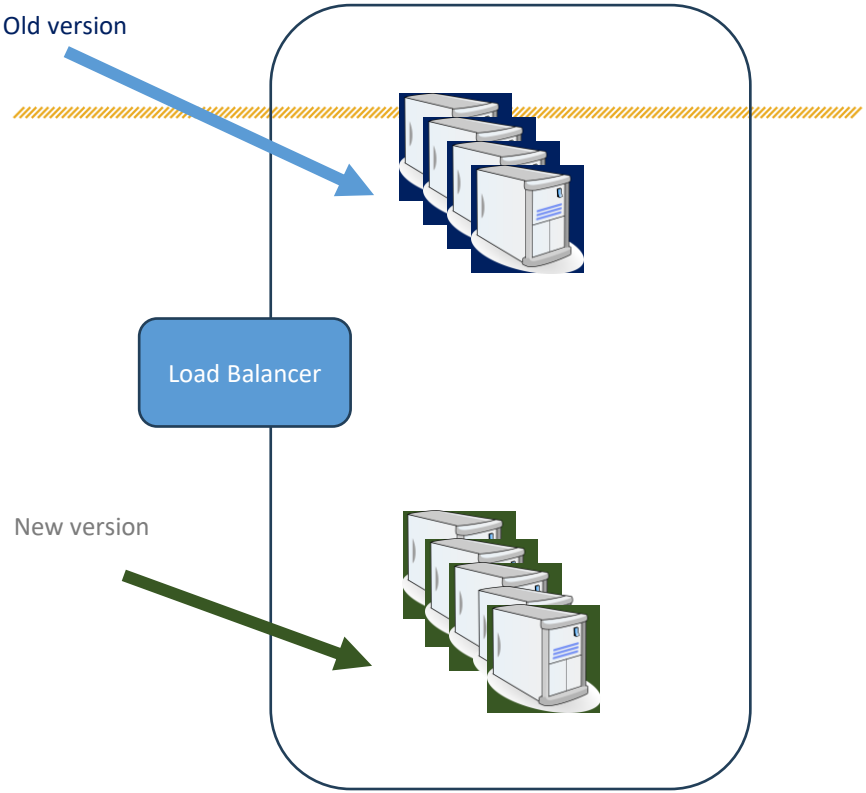

Microservices Integration Test
E2E
Integrati on Tests
Unit Tests
E2E
Integrati on Tests
Unit Tests
E2E
Integrati on Tests
Unit Tests
E2E
Integrati on Tests
Unit Tests

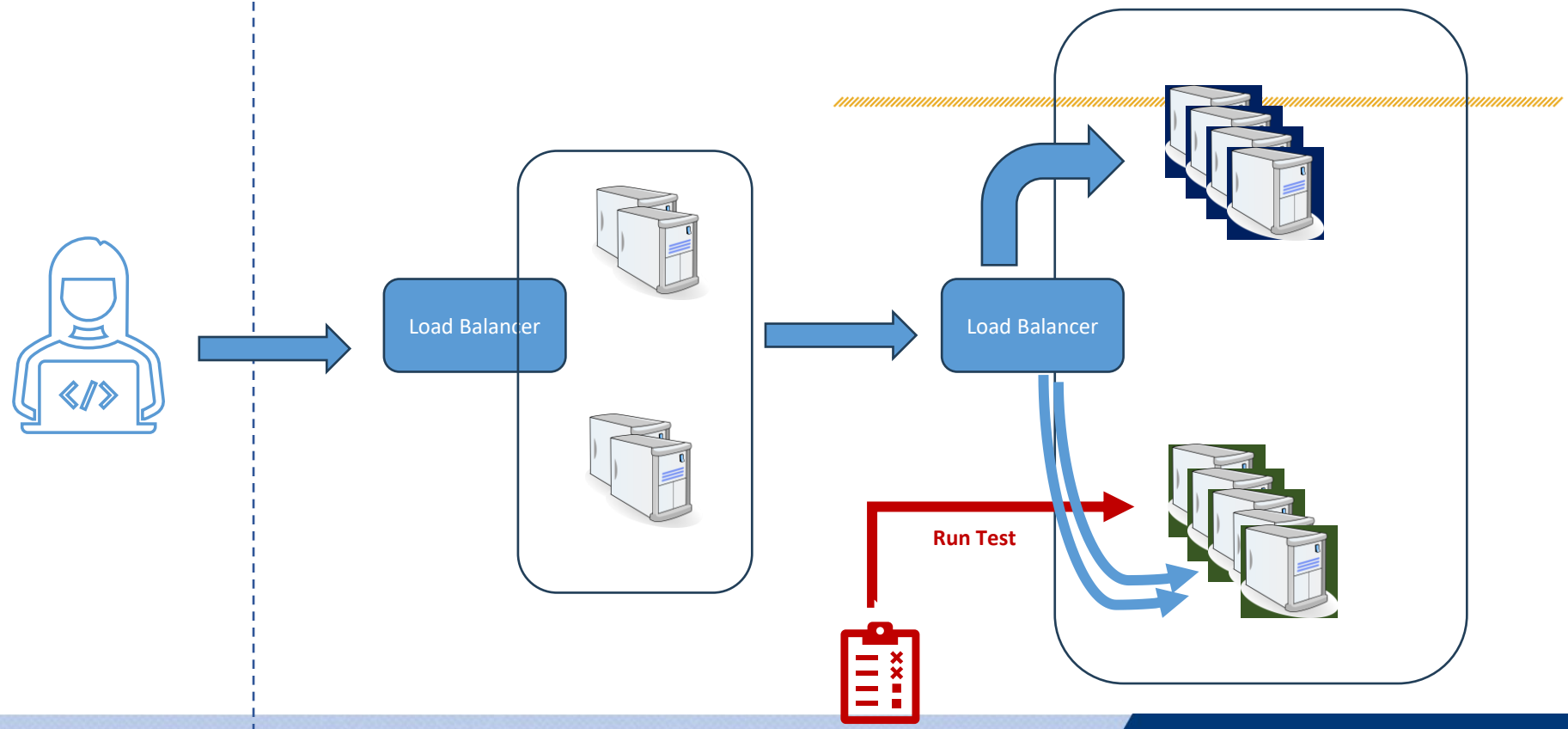
End to end Tests Alternative

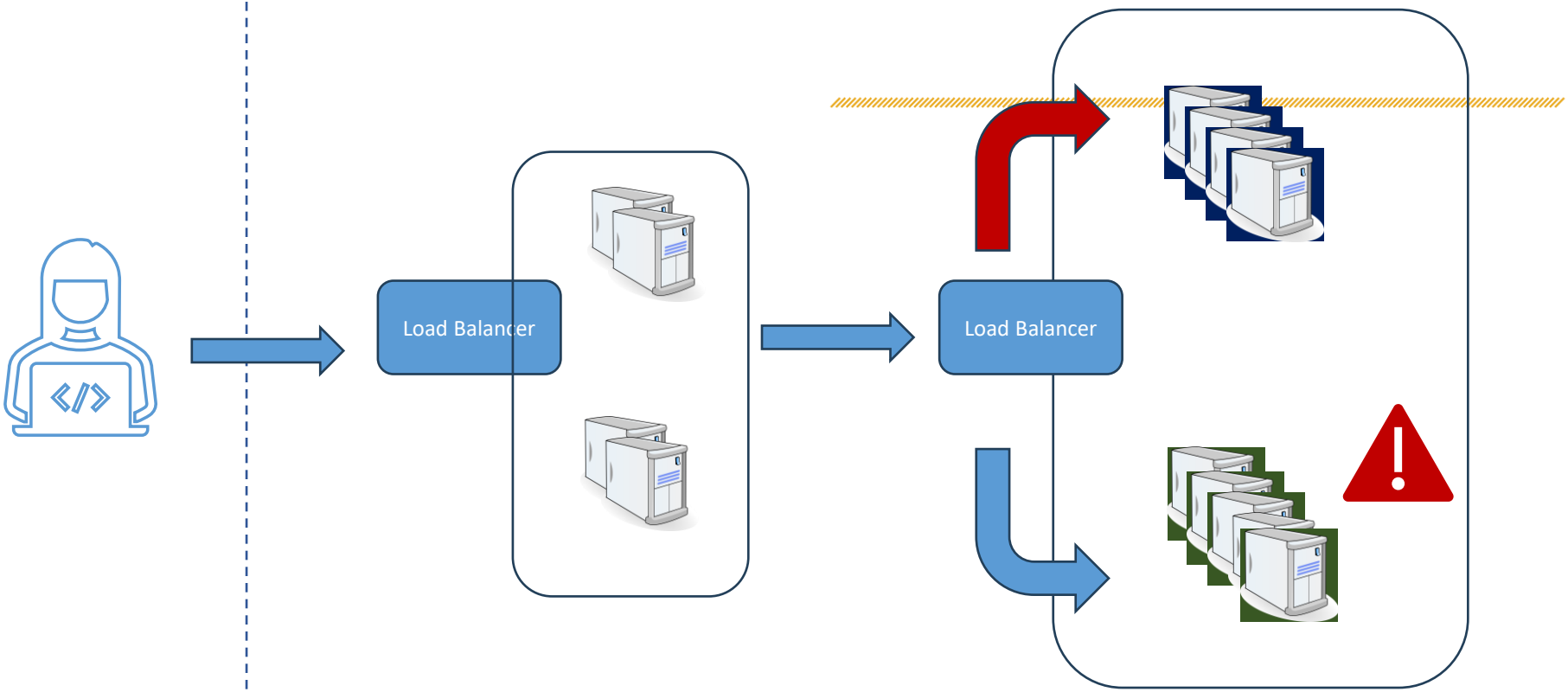
- **Testing in Production**
  - Blue-Green Deployment + Canary Testing

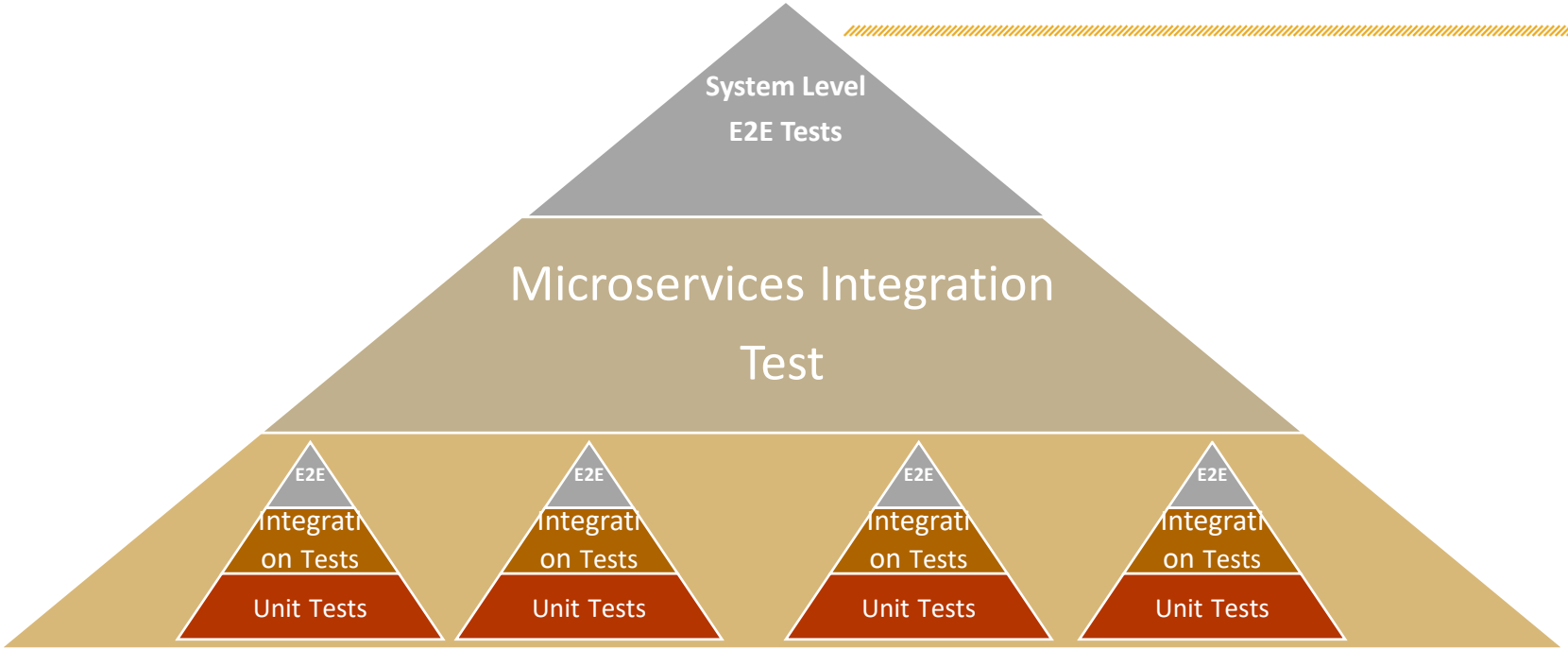



Load Balancer


Old version
Load Balancer
New version


Load Balancer
Load Balancer
Run Test


Load Balancer
Load Balancer


System Level
E2E Tests
Microservices Integration
Test
E2E
Integrati
on Tests
Unit Tests
E2E
Integrati
on Tests
Unit Tests
E2E
Integrati
on Tests
Unit Tests
E2E
Integrati
on Tests
Unit Tests

Summary

- **Address the complexity of running integration tests/end-to-end tests in a microservice architecture**
  - Using mocks for integration testing
    - Simplifies integration tests
    - Not enough to enforce API contracts
  - Contract tests keep the contract
    - Between API consumers and API producers in sync
    - Event-driven microservice
  - Alternative to end-to-end testing: Production Testing using
    - Blue-Green Deployment
    - Canary Testing

5 min break ☺

Observability in Microservices

Introduction to the Three Pillars of Observability

Topics

- What is Observability
- The Importance of Observability
- Three Pillars of Observability

Topics

- **What is Observability**
- The Importance of Observability
- Three Pillars of Observability

Observability vs Monitoring

- **Both provide tools to :**
  - Collect data
  - Give insights
  - Detect issues

Monitoring

- **Process of:**
  - Collecting
  - Analyzing
  - Displaying

  Predefined set of metrics
- **Allows us to find out if something goes wrong**
- **Don’t tell us**
  - What is wrong
  - How to remediate the problem

Observability

- **Enables us to :**
  - Debug
  - Search for patterns
  - Follow inputs/outputs
  - Get insights into the behavior of the system

Observability

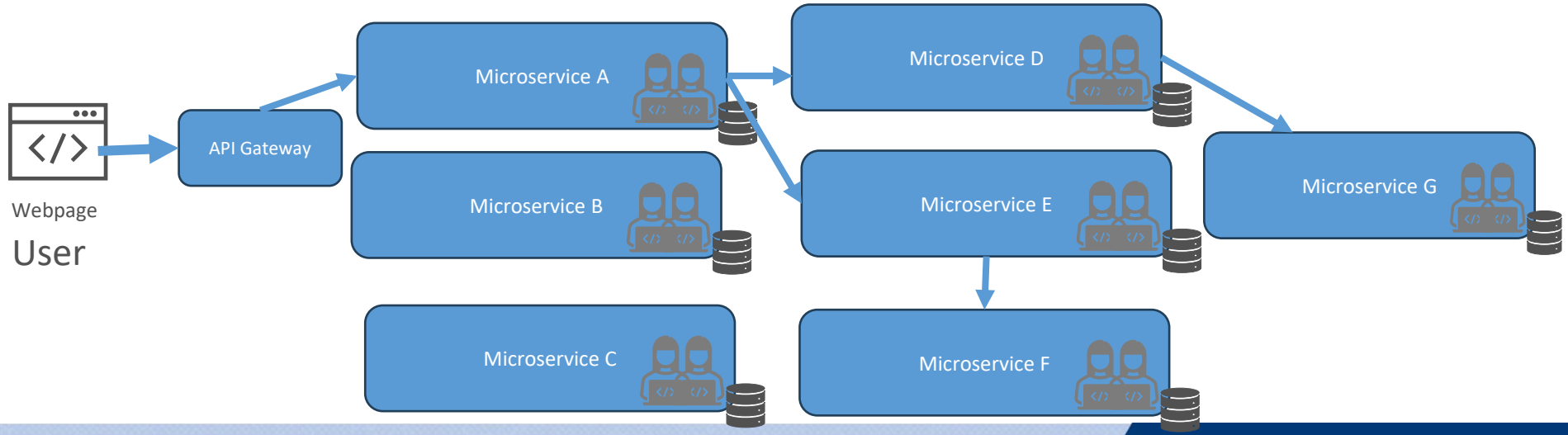
- **Allows us to follow an individual :**
  - Requests
  - Transactions
  - Events
- **Discover and isolate performance bottlenecks**
- **Point us to the source of the problem**

Topics

- What is Observability
- **The Importance of Observability**
- Three Pillars of Observability

Troubleshooting a Monolith Application

Troubleshooting a Microservice



Topics

- What is Observability
- The Importance of Observability
- **Three Pillars of Observability**

Observability in Microservices

1. Distributed logging
2. Metrics
3. Distributed tracing

1. Distributed logging

- **Appending-only files**
- **Events in :**
  - Application process
  - Container
  - Database instance
  - Server
- **Structured/semi-structured strings**
- **Metadata includes:**
  - Timestamp
  - Request
  - Method
  - Class
  - Application

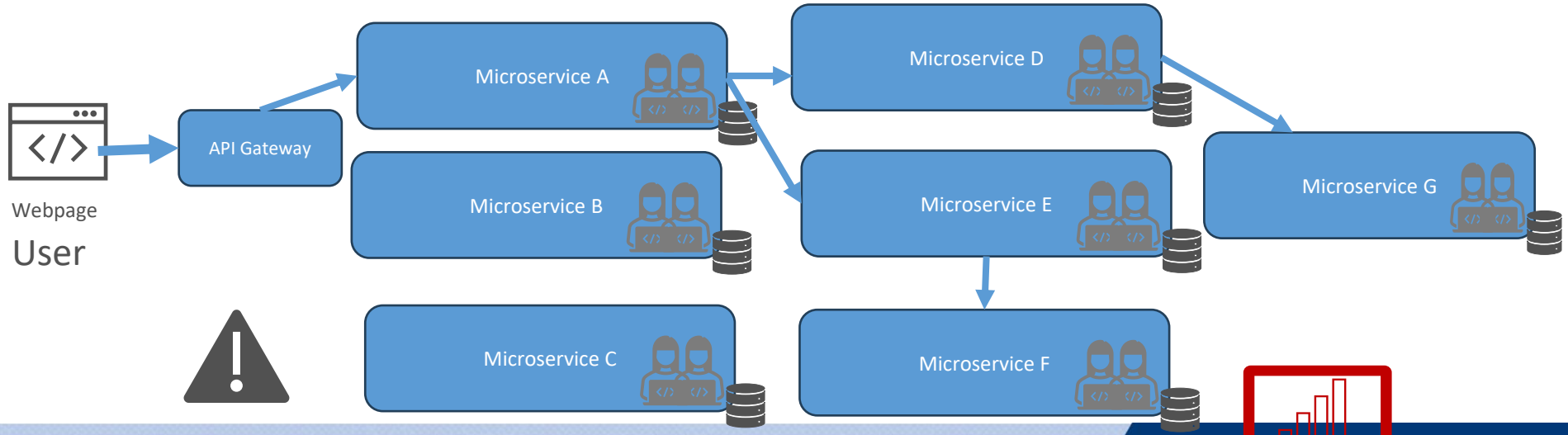
2. Metrics

- **Regularly sampled data points**
- **Numerical values**
  - Counters
  - Distributions
  - Gauges
- **Examples:**
  - Request/min
  - Errors/hors
  - Latency distribution
  - Current CPU utilization
  - Memory usage
  - Cache hit rate

3. Tracing

- **Path of a given request through several microservices**
- **Time each microservice took to process it**
- **May include:**
  - Request headers
  - Response status code

Troubleshooting a Microservice



Observability Signals

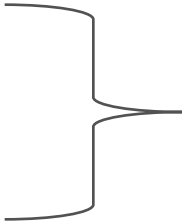
- **Help address issues by:**
  - Issuing a rollback
  - Making a Hotfix
  - Adding more instances
  - Diverting traffic to another region / data center

Microservice Observability

Distributed Logging

Observability in Microservices

- **The Three Pillars:**
  1. Distributed Logging
  2. Metrics
  3. Distributed Tracing



Get insights, debug, and find the source of the issue

Topics

- What is Logging
- Distributed Logging-Best Practices

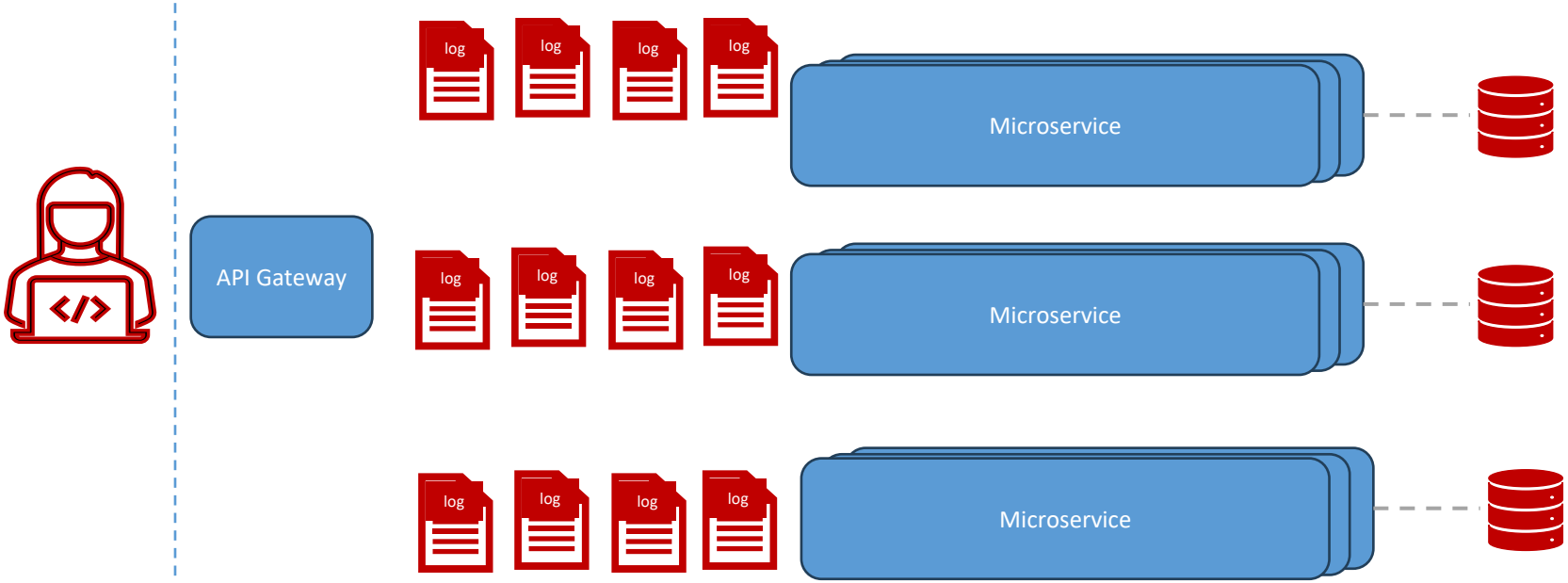
Topics

- **What is Logging**
- Distributed Logging- Best Practices

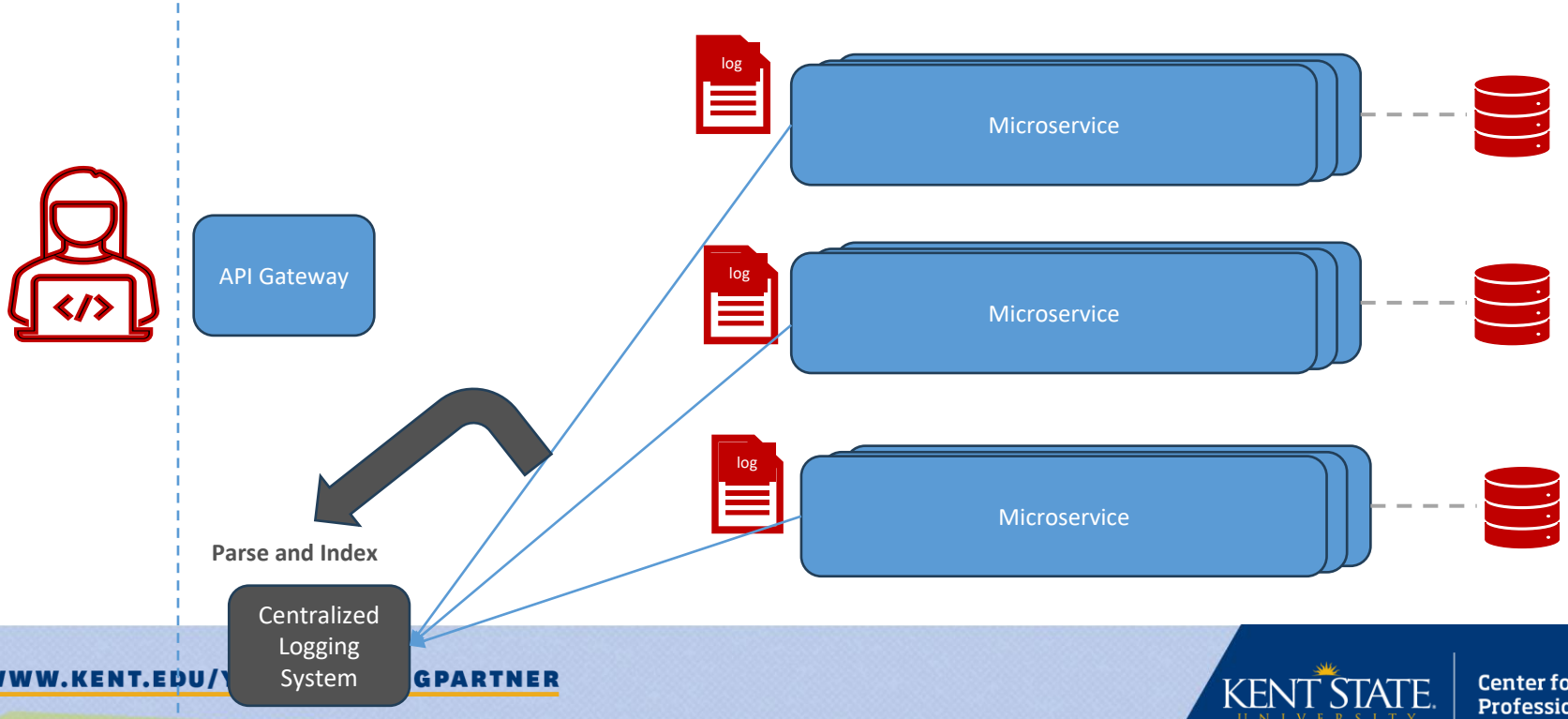
Distributed Logging

- A simple way to provide insight into the application's state
- A Logline can be:
  - Event
  - Action
  - Exception
  - Error

Logging in Microservice Architecture



1. Centralized logging System



Distributed Logging – Best Practices

1. **Centralized System**
2. **Predefined structure/schema**

- **Log lines should be:**
  - Machine-Readable
  - Human Readable

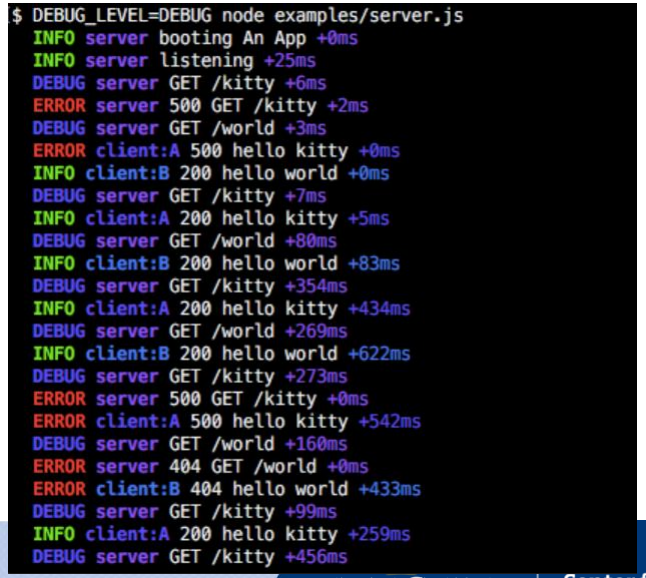
Log Structure Examples

- Logfmt
- JSON
- XML

```
{
 "info": "INFO",
 "msg": "Application running smoothly",
  "properties": {
    "status": "OK",
    "details": "No errors detected",
    "performance": "Optimal"
  }
}
```

Distributed Logging – Best Practices

1. Centralized System
2. Predefined structure/schema
3. Log Level / Log Severity
    1. TRACE
    2. DEBUG
    3. INFO
    4. WARN
    5. ERROR
    6. FATAL



Distributed Logging – Best Practices

1. Centralized System
2. Predefined Structure/Schema
3. Log Level / Log Severity
4. Correlation ID

Distributed Logging – Best Practices

1. Centralized System
2. Predefined Structure/Schema
3. Log Level / Log Severity
4. Correlation ID
5. Adding Contextual Information

Common Data Points

- Service name
- Hostname / IP address
- User ID
- Timestamp

Contextual Information - Considerations

- **Log only necessary data**
- **Do not log:**
  - Sensitive data
  - PII (personally identifiable information)

Summary

- **Learned about Distributed Logging**
- **Best practices:**
  - Centralized logging system
  - Log structure
  - Log level severity
  - Correlation ID
  - Adding Contextual information

10 min break ☺

Observability in Microservice Architecture

Metrics

Observability in Microservice

- Distributed Logging
- Metrics
- Distributed tracing

Topics

- What are Metrics?
- The problem of collecting too many metrics
- The 5 “golden Signals/Metrics”

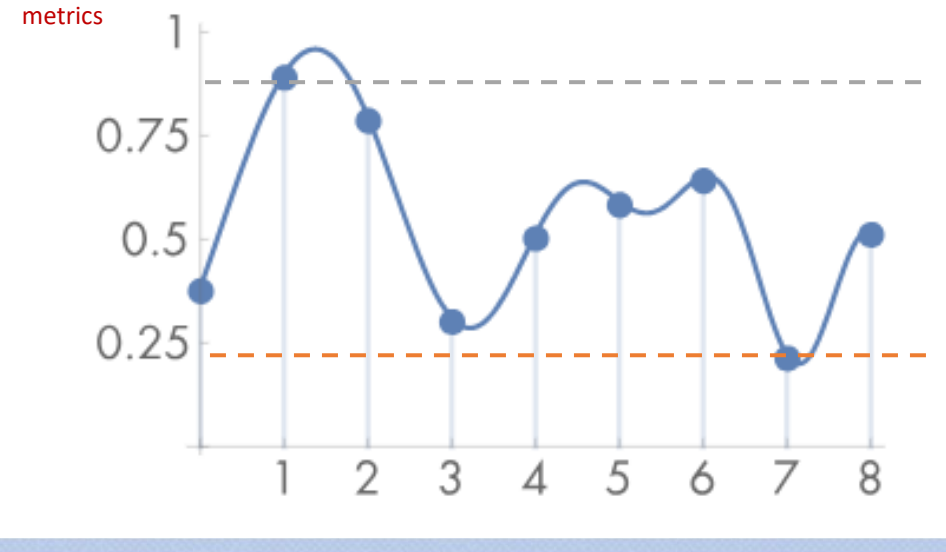
Topics

- **What are Metrics?**
- The problem of collecting too many metrics
- The 5 “golden Signals/Metrics”

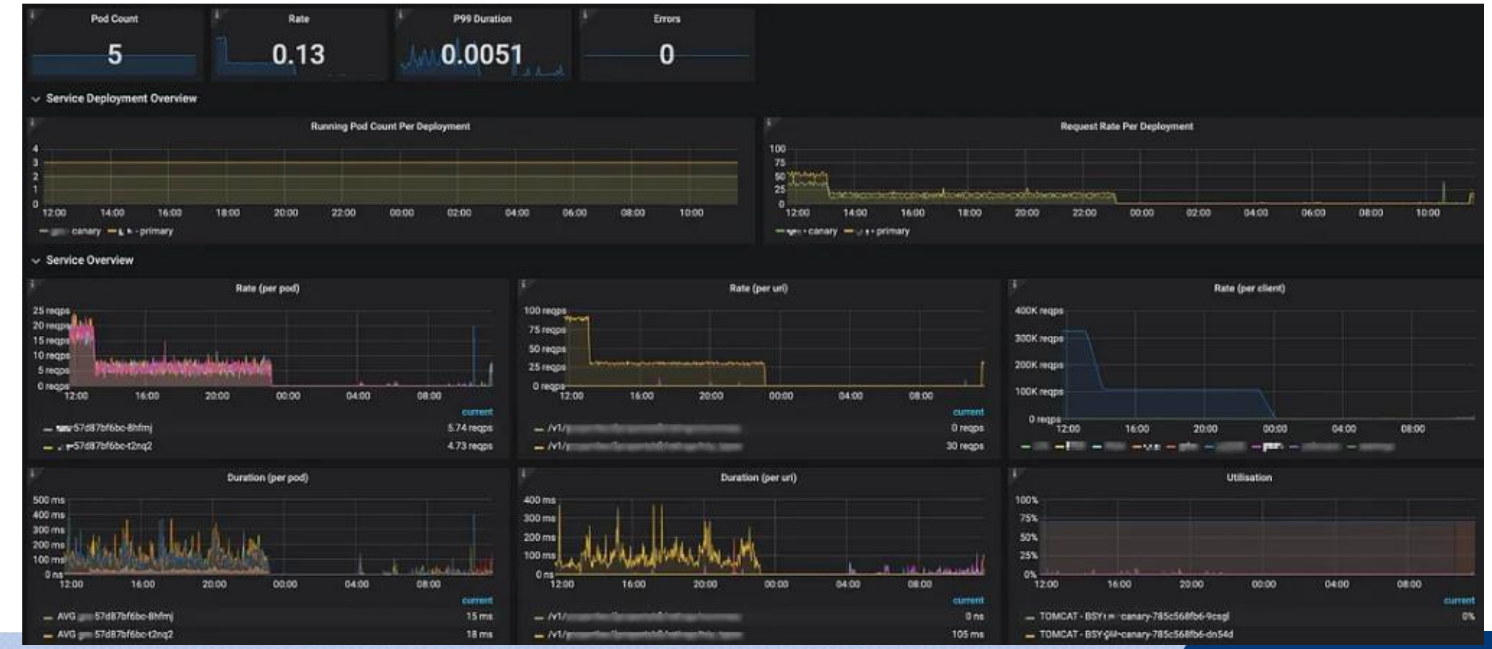
What are Metrics

- **Measurable or Countable signals of software that help us monitor the system's health and performance**

What are Metrics



Dashboards



https://medium.com/expedia-group-tech/creating-monitoring-dashboards-1f3fbe0ae1ac

Topics

- What are Metrics?
- **The problem of collecting too many metrics**
- The 5 “golden Signals/Metrics”

What Metrics Can We Collect?

- **Resources**
  - CPU
  - Memory
  - Disk IO
  - Network
  - Processes
  - …

- **Application Instrumentation**
  - Incoming Requests/min
  - Database queries/min
  - GC pause Time
  - hroughput
  - mber of threads
  - emory Usage
  - rders Received
  - Query size
  - …

Collecting too Many Metrics

- Expensive
- Information overload

Topics

- What are Metrics?
- The problem of collecting too many metrics
- **The 5 “golden Signals/Metrics”**

The Five Golden Types of Signals- References

1. Google SRE Book- “The Four Golden Signals”
2. The Use Method by Brendan Gregg

The Five Golden Types of Signals-

1. Traffic
2. Errors
3. Latency
4. Saturation
5. Utilization

1. Traffic

- **The amount of demand being placed on our system per unit of time**
- **Examples:**
  - HTTP requests/sec
  - Queries/sec
  - Transaction/sec
  - Events received/sec
  - Event delivered/sec
  - Incoming requests + outgoing requests/sec

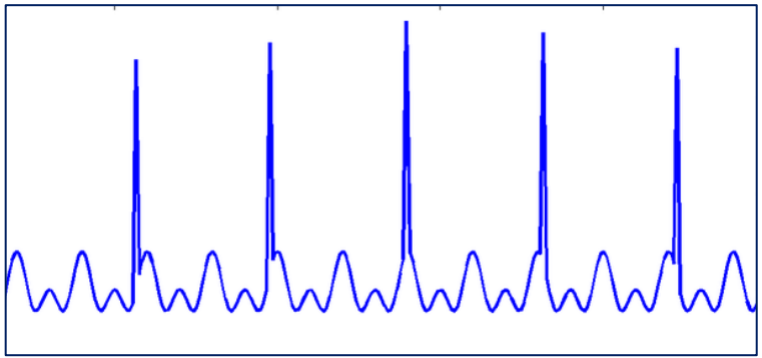
2. Errors

- **Error Rate and Error Types**
- **Examples:**
  - Number of application exceptions
  - HTTP response status code (4xx, 5xx)
  - Response exceeding latency threshold
  - Failed Events
  - Failed Delivery
  - Aborted transactions
  - Disk failure
  - …

3. Latency

- **The time takes for a service to process a request**
- **Important considerations:**
  - Latency distribution vs average

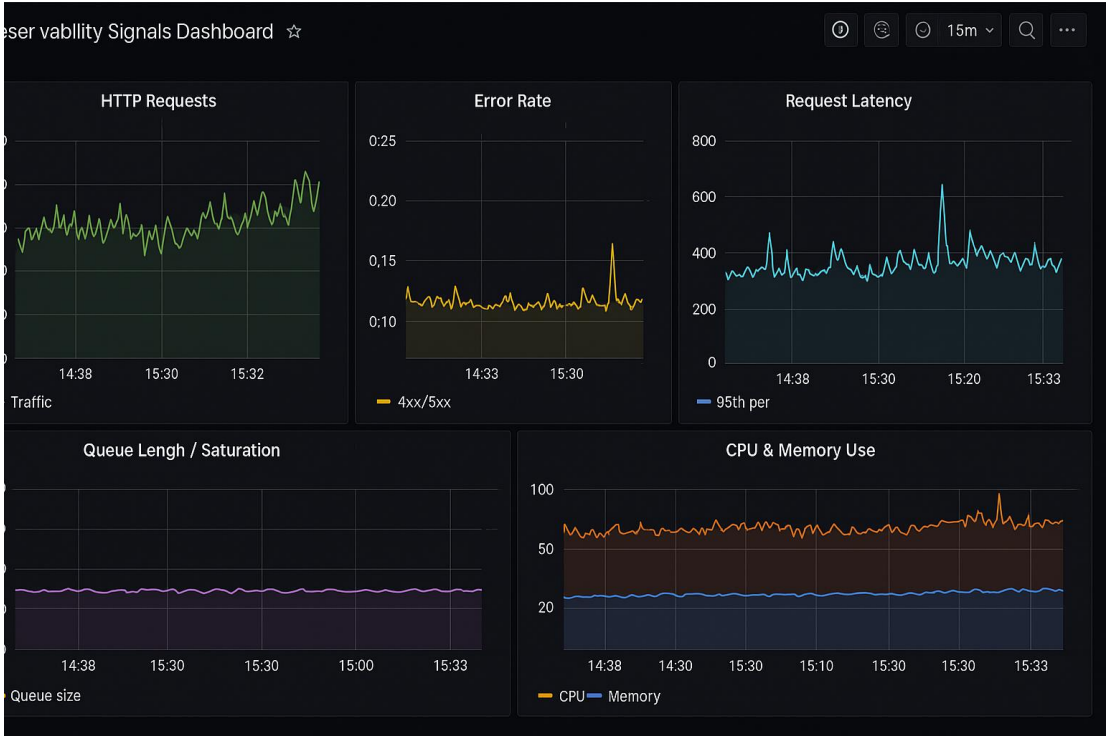
Example:
# of Req = 1000 request/min
95% processed within 50 msec
5% processed within 5000msec
Avg Latency = 300 msec
5% of the users wait 5 sec for the website to load !?



3. Latency

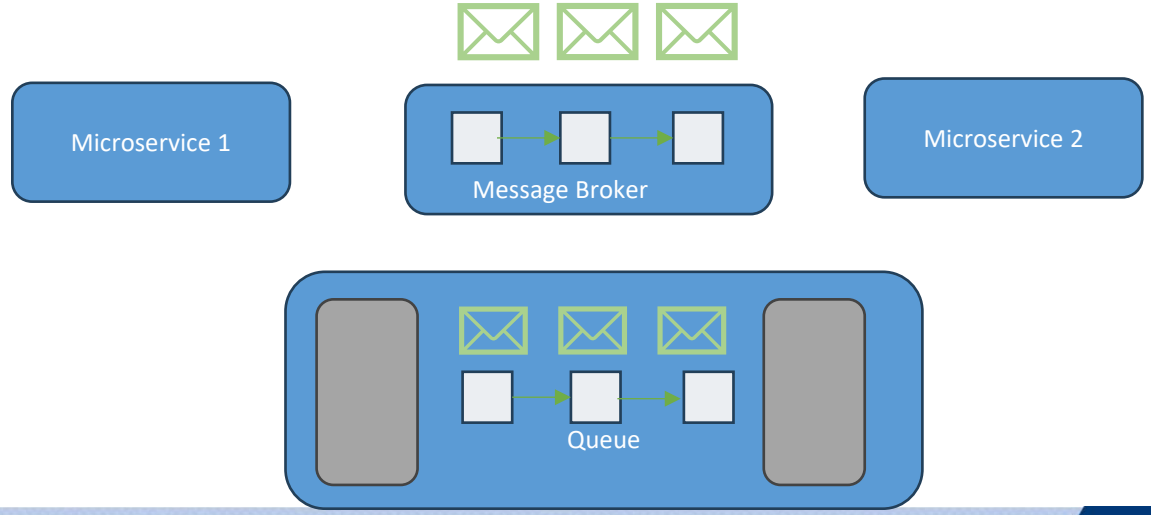
- **The time takes for a service to process a request**
- **Important considerations:**
  - Latency distribution vs average
  - Separate successful operation from failed operations

3. Latency

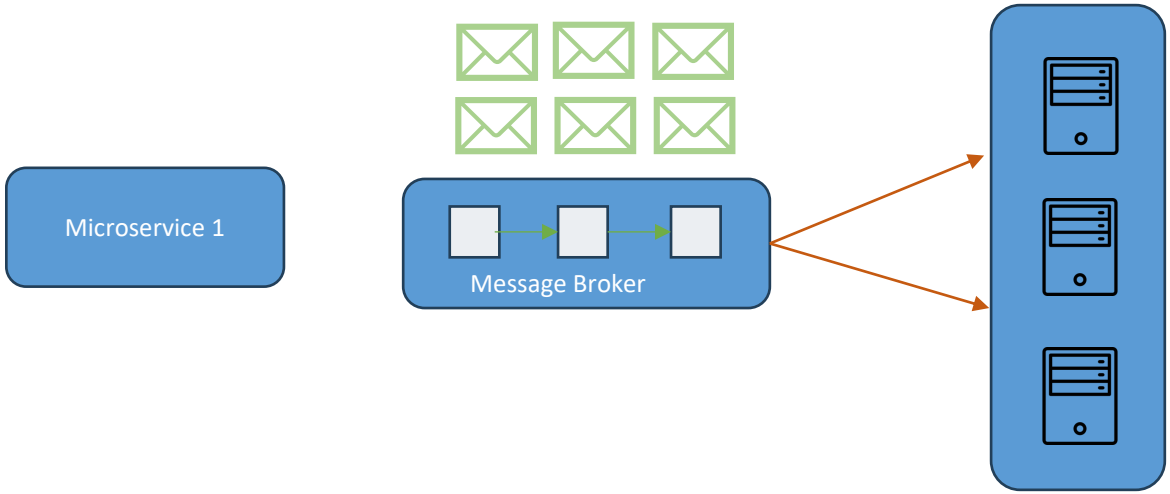

ser vabllity Signals Dashboard
15m
HTTP Requests
14:38
15:30
15:32
Traffic
Error Rate
0:25
0.20
0,15
0:10
14:33
15:30
4xx/5xx
Request Latency
800
600
400
200
0
14:38
15:30
15:20
15:33
95th per
Queue Lengh / Saturation
14:38
15:30
15:30
15:00
15:33
Queue size
CPU & Memory Use
100
50
20
14:38
14:30
15:30
15:10
15:30
15:30
15:33
CPU
Memory

4. Saturation

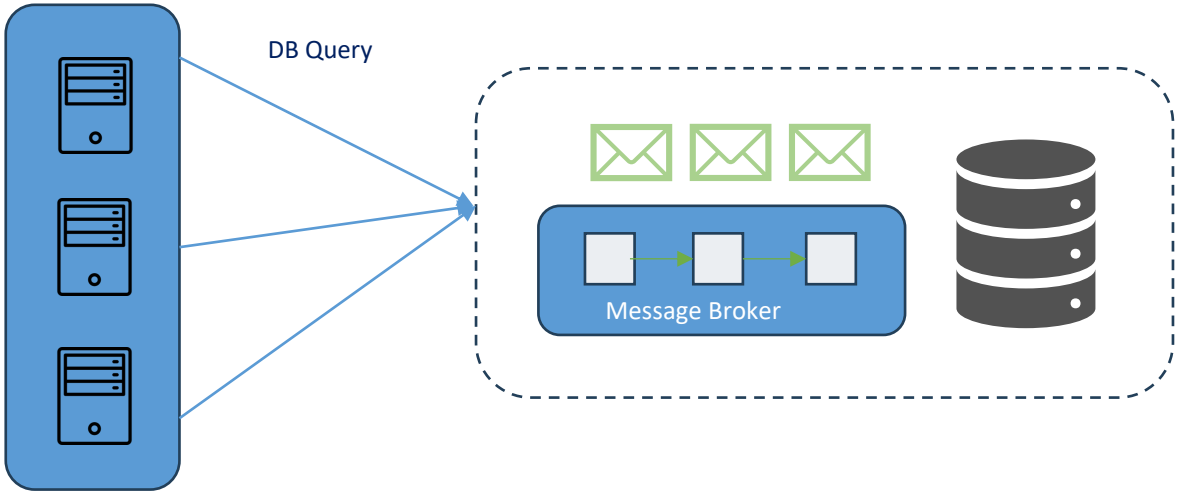
How Overloaded /Full a service/resource is.



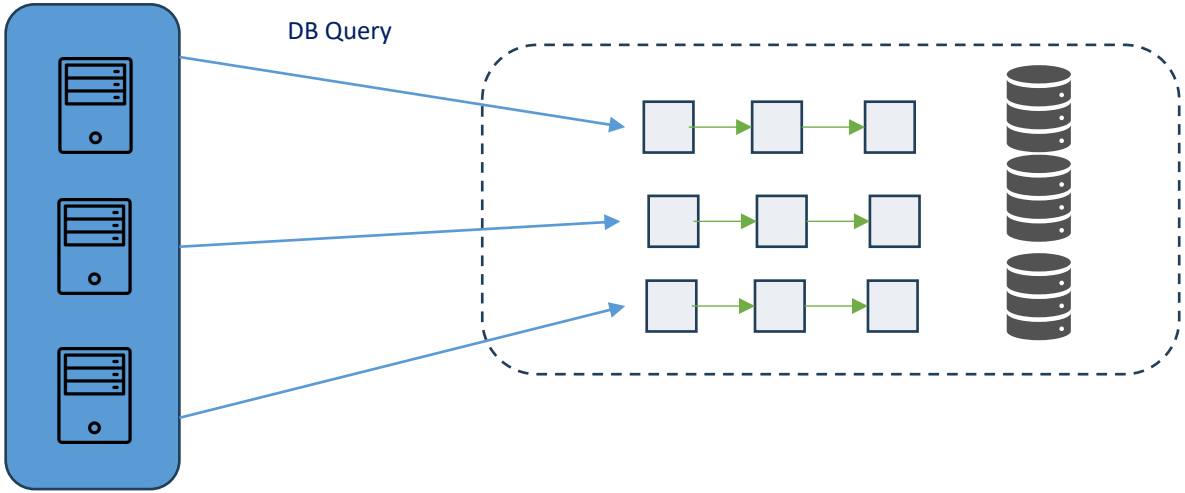
4. Saturation



4. Saturation

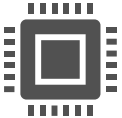


4. Saturation



5 Utilization

- How busy a resource is [0 – 100%]

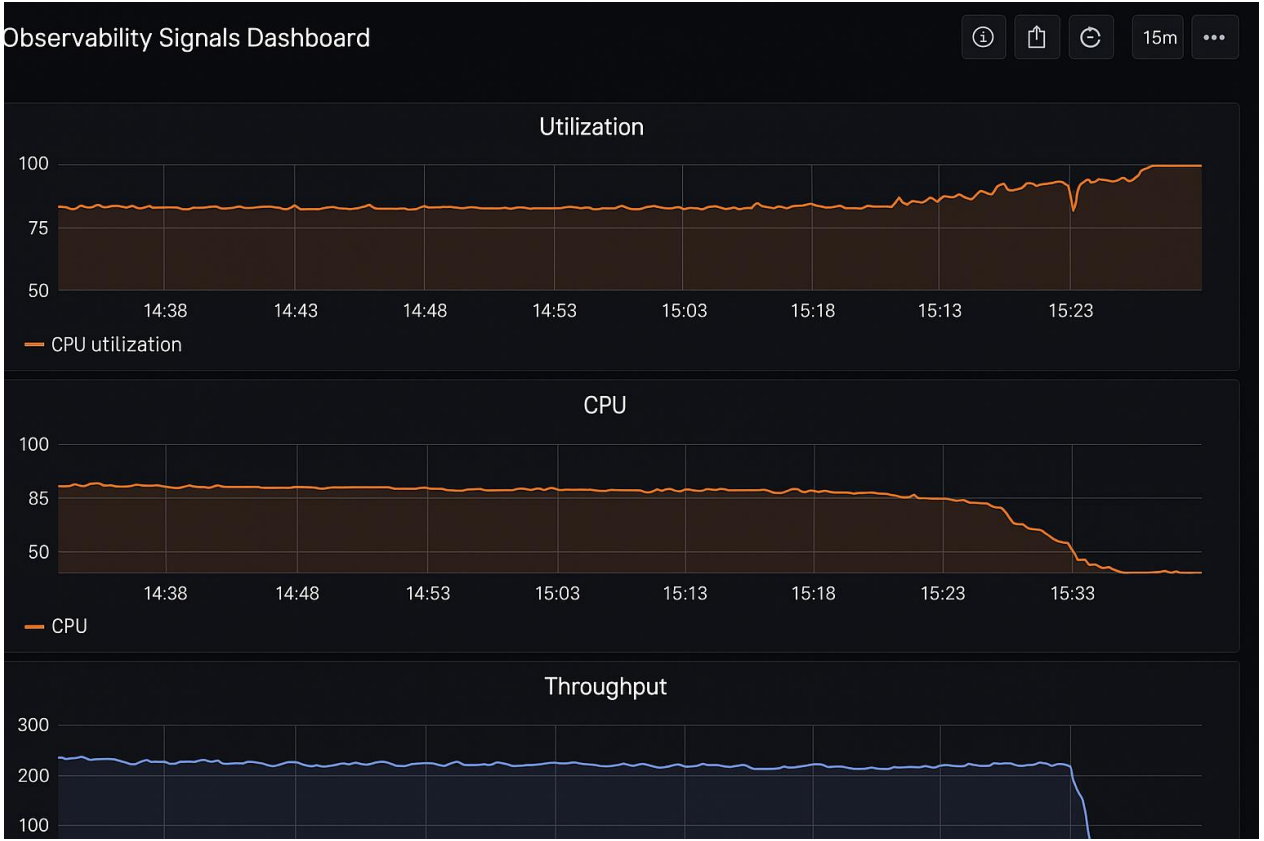


CPU

Memory

Disk

5 Utilization



Signals

- **The Golden Five Signals:**
  1. Traffic
  2. Errors
  3. Latency
  4. Saturation
  5. Utilization
- **They are the:**
  - Most common
  - Give the most value

Observability in Microservice Architecture

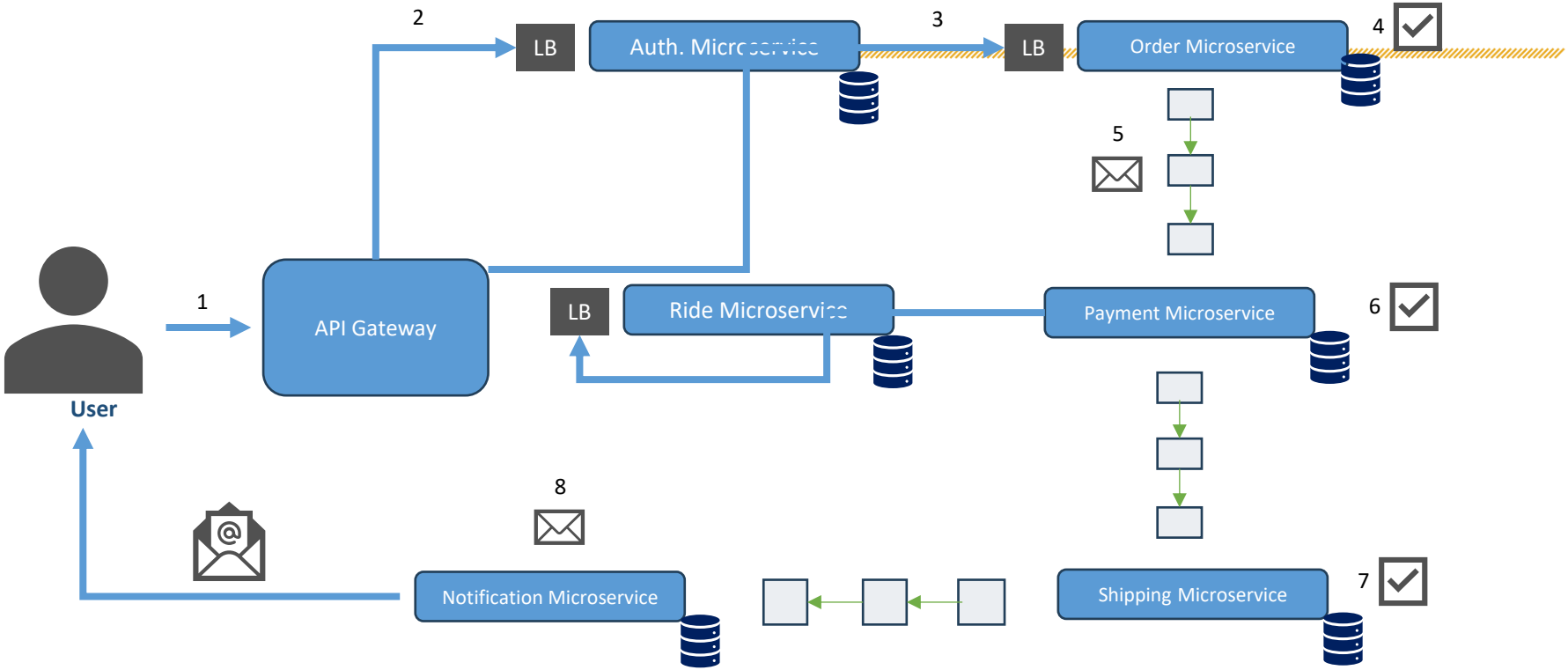
Distributed Tracing

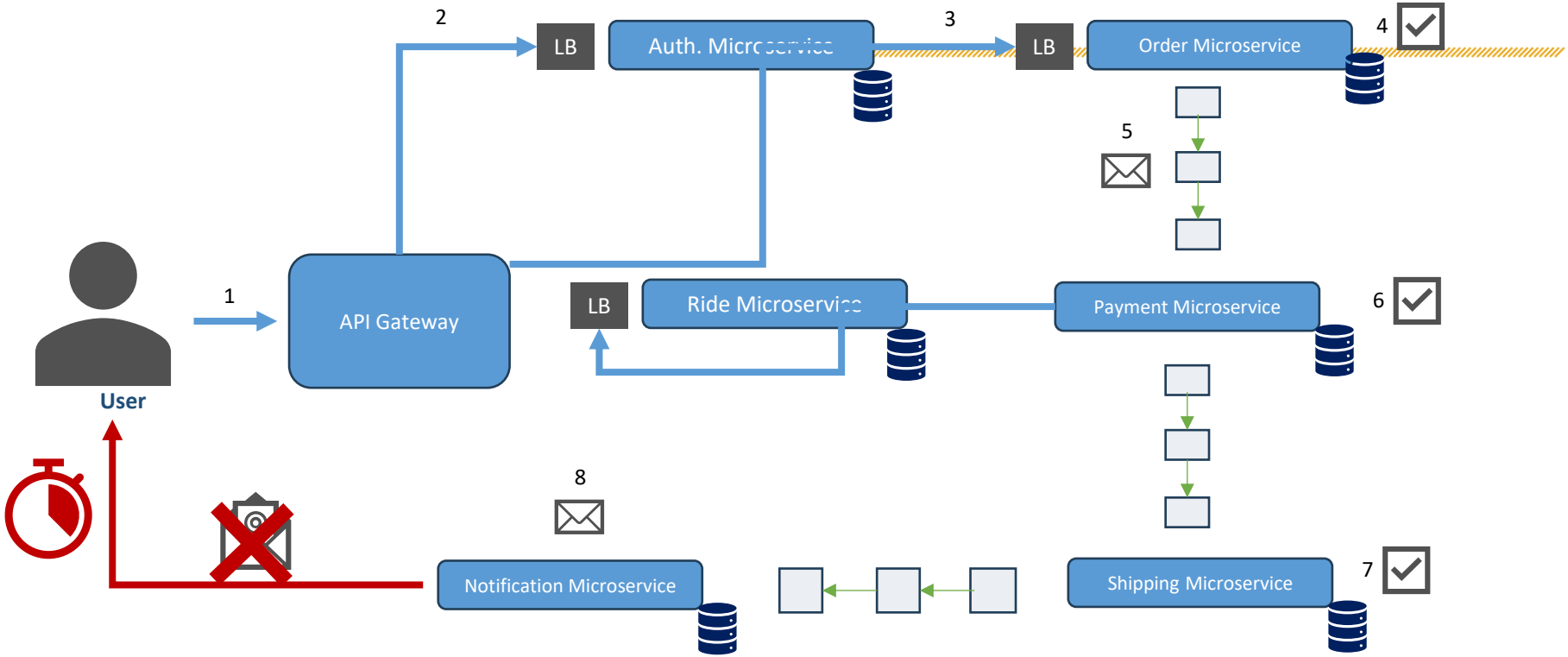
Topics

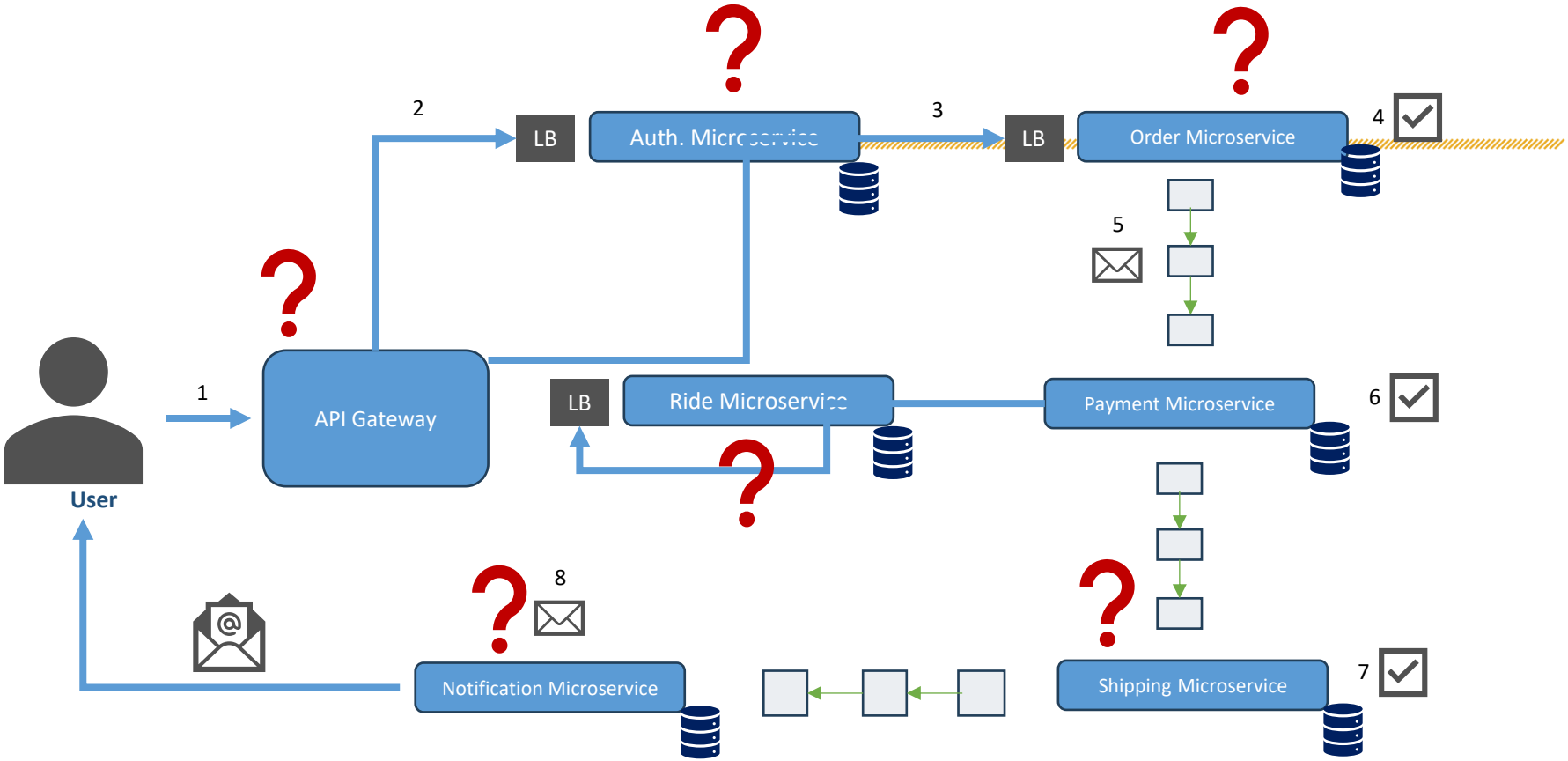
- Distributed Tracing – Motivation
- Terminology and How Distributed Tracing Works
- Challenges in Even-Driving Architecture

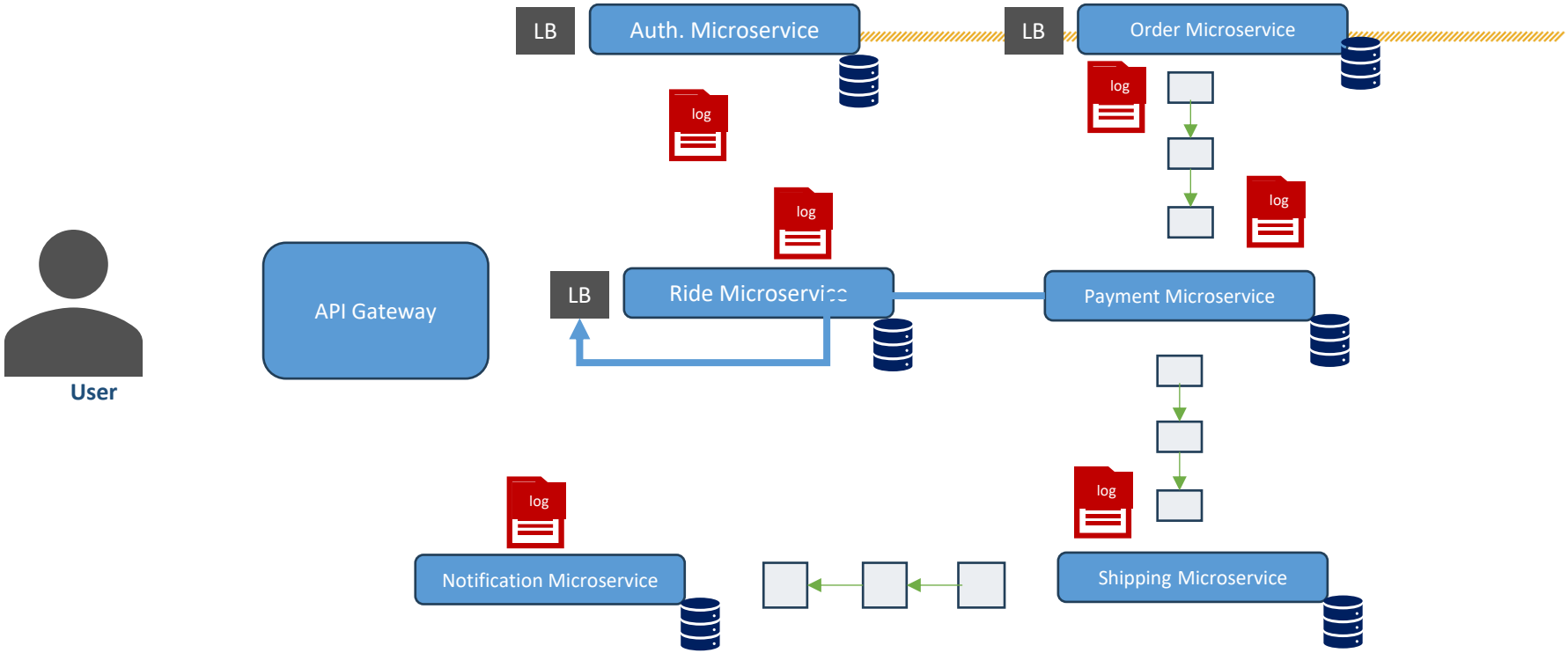
Topics

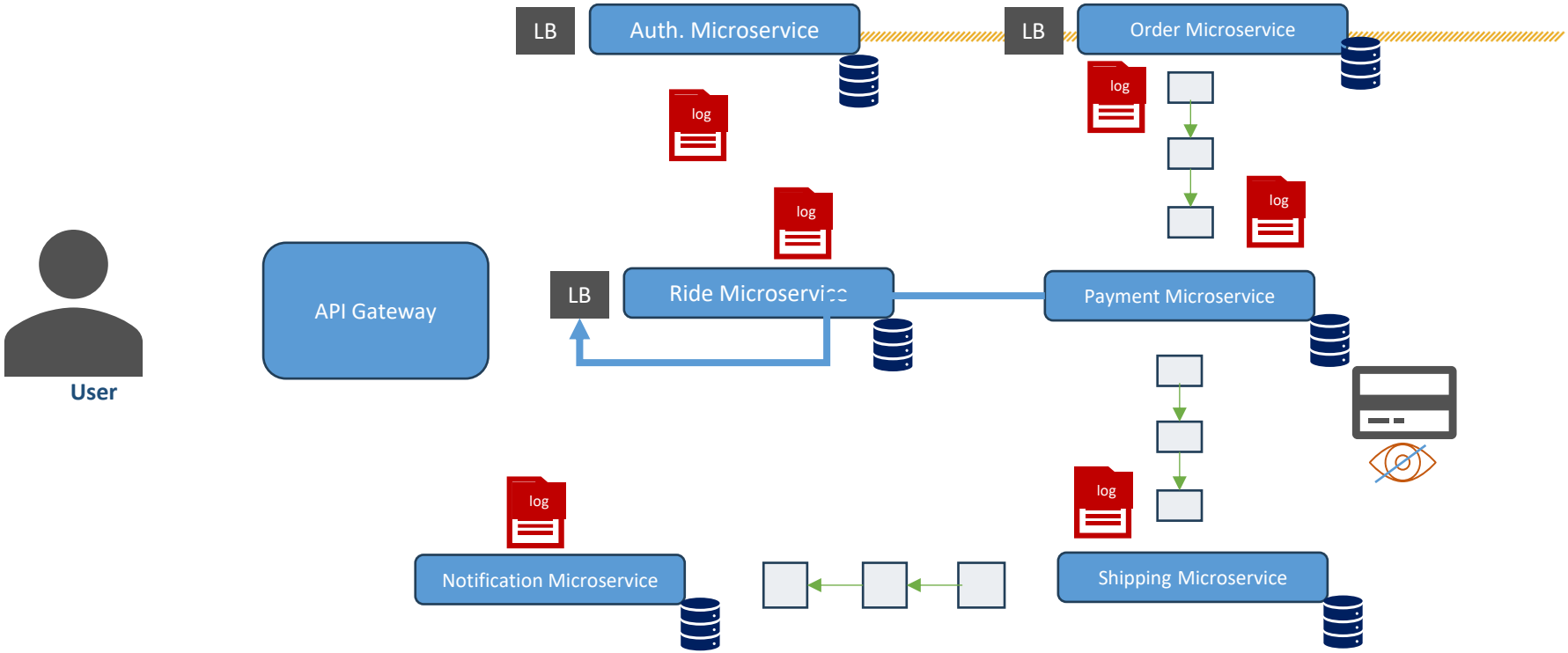
- **Distributed Tracing – Motivation**
- Terminology and How Distributed Tracing Works
- Challenges in Even-Driving Architecture


2
LB
Auth. Microservice
3
LB
Order Microservice
4
5
1
API Gateway
LB
Ride Microservice
Payment Microservice
6
User
8
Notification Microservice
Shipping Microservice
7

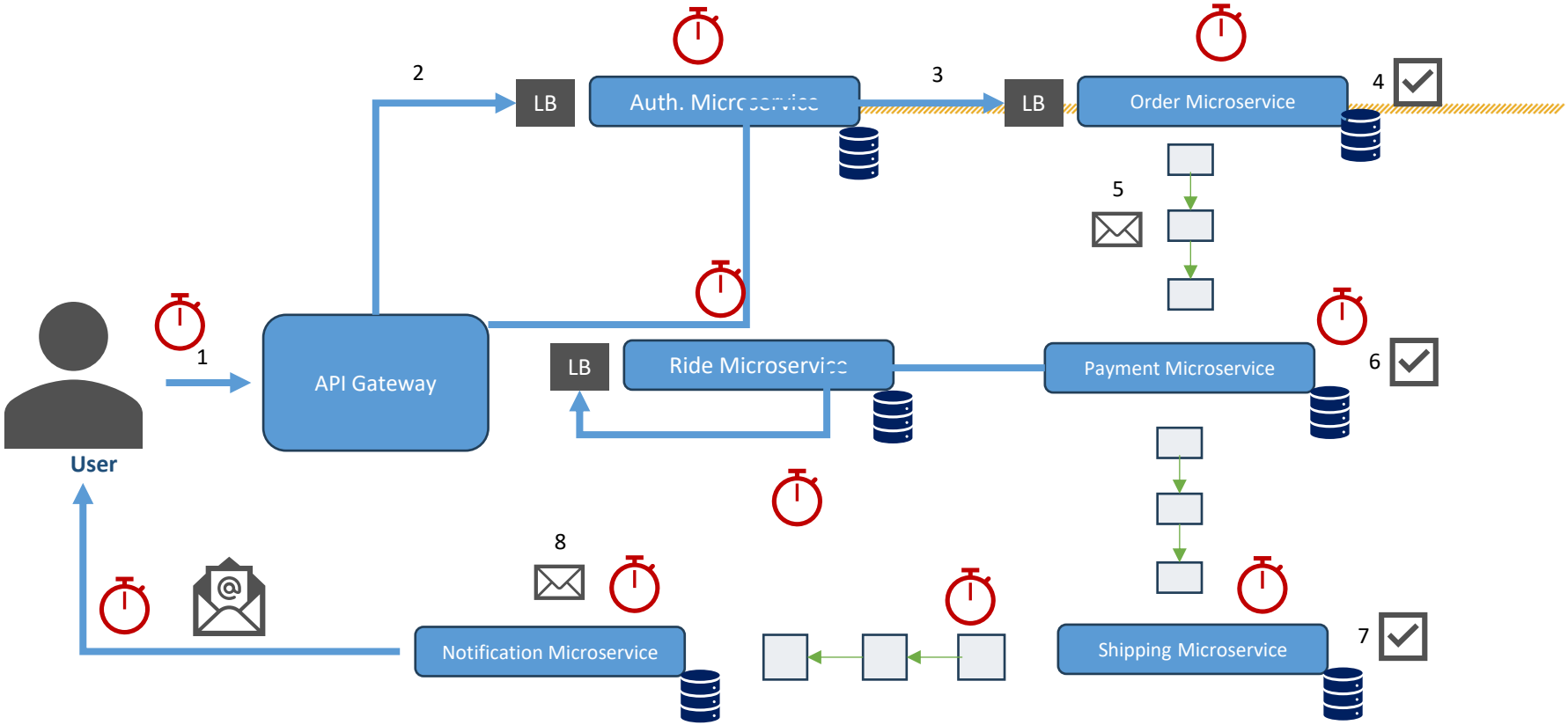

2
LB
Auth. Microservice
3
LB
Order Microservice
4
5
User
1
API Gateway
LB
Ride Microservice
Payment Microservice
6
8
Notification Microservice
Shipping Microservice
7


User
1
API Gateway
2
LB
Auth. Microservice
3
LB
Order Microservice
4
5
LB
Ride Microservice
Payment Microservice
6
8
Notification Microservice
Shipping Microservice
7

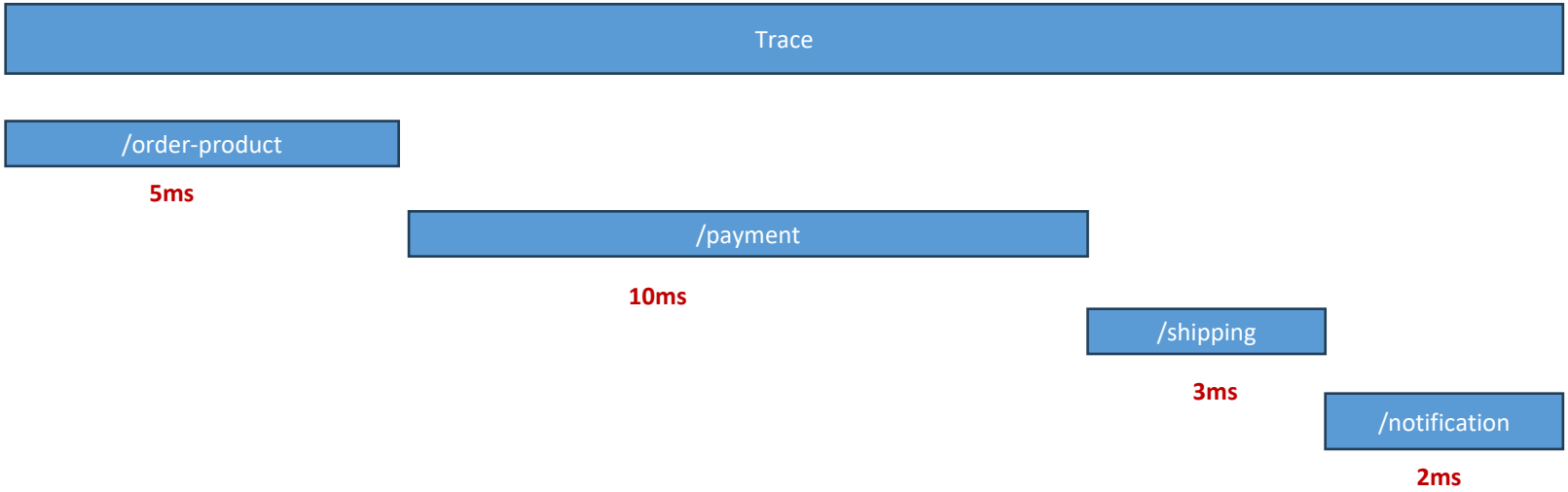

LB
Auth. Microservice
LB
Order Microservice
log
log
log
log
User
API Gateway
LB
Ride Microservice
Payment Microservice
log
log
Notification Microservice
Shipping Microservice


LB
Auth. Microservice
LB
Order Microservice
log
log
log
log
User
API Gateway
LB
Ride Microservice
Payment Microservice
log
log
Notification Microservice
Shipping Microservice

Solution: Distributed Tracing


2
LB
Auth. Microservice
3
LB
Order Microservice
4
5
1
API Gateway
LB
Ride Microservice
Payment Microservice
6
User
8
Notification Microservice
Shipping Microservice
7

Distributed Tracing Visualization



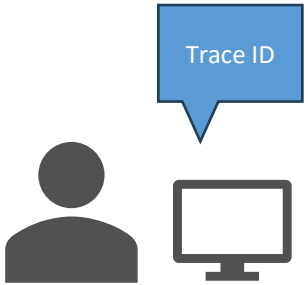
Distributed Tracing

- **Not enough on its own**
- **Helps narrow down the :**
  - Faulty component
  - Communication problem
- **We can use logs and metrics to debug further**

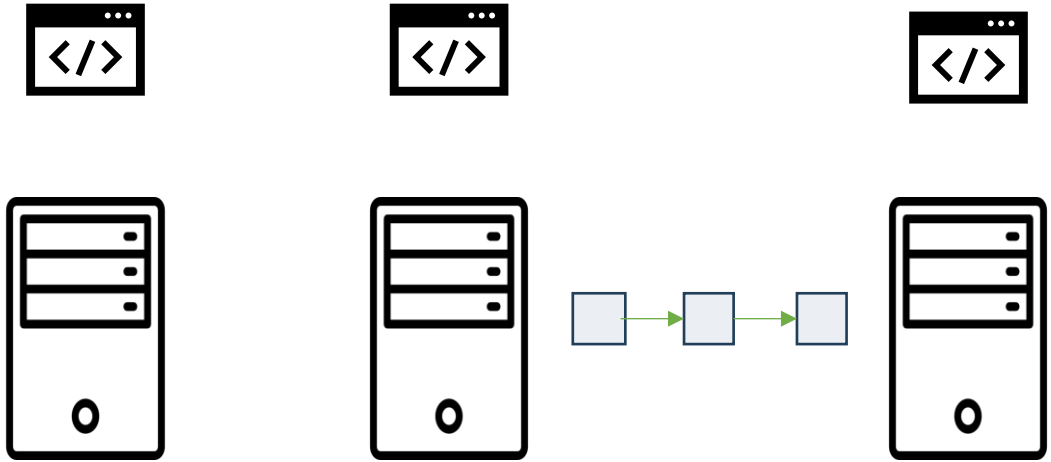
Topics

- Distributed Tracing – Motivation
- **Terminology and How Distributed Tracing Works**
- Challenges in Even-Driving Architecture

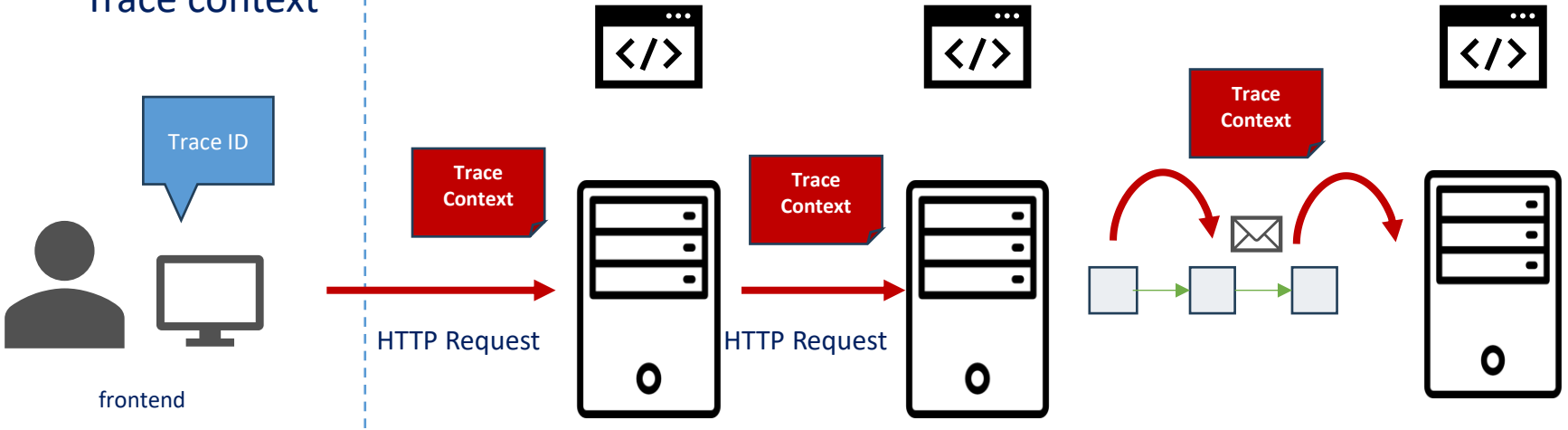
Trace context



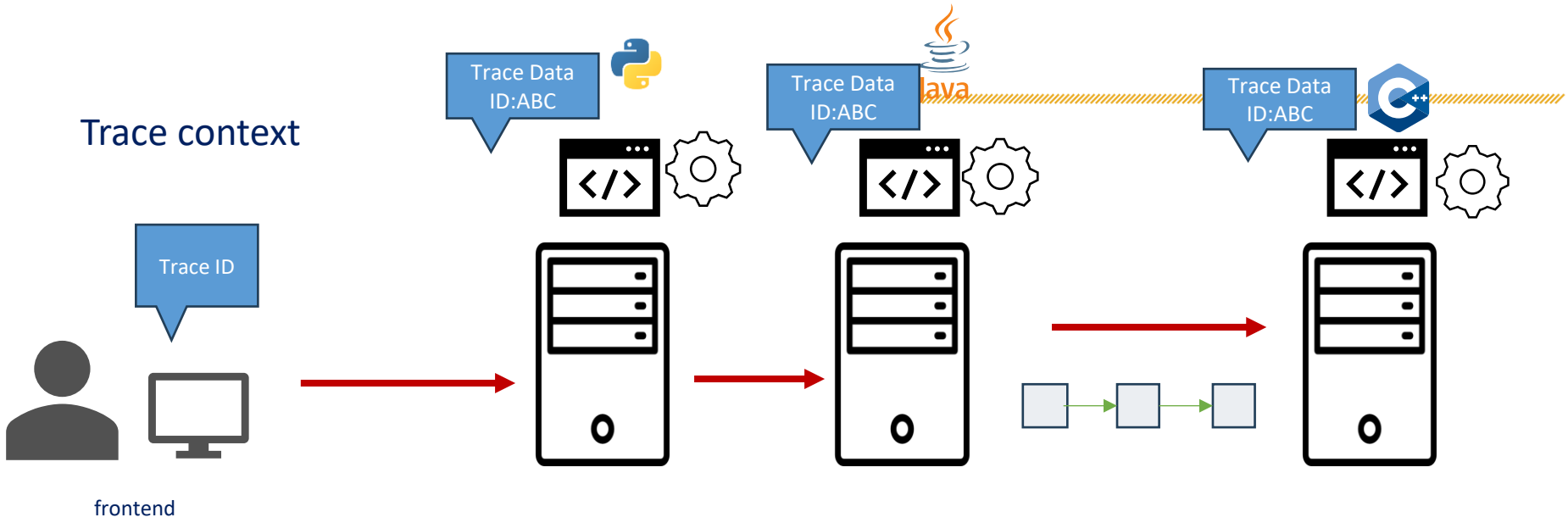
Frontend

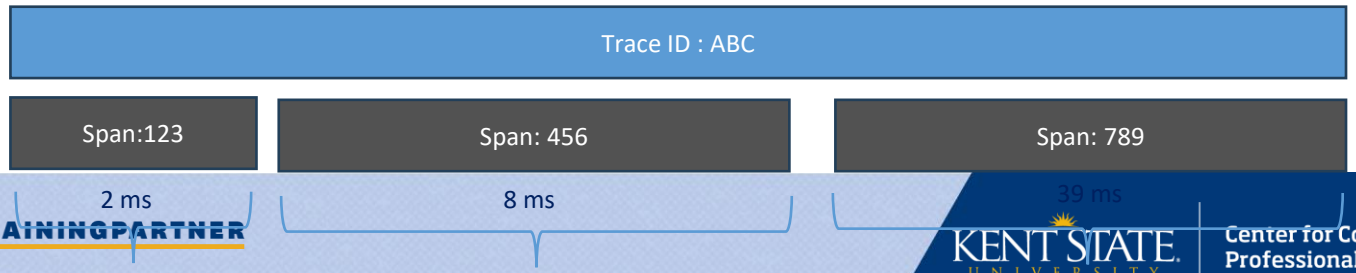


Trace context



Trace context





Topics

- Distributed Tracing – Motivation
- Terminology and How Distributed Tracing Works
- **Challenges in Even-Driving Architecture**

Distributed Tracing - Challenges

1. Manual instrumentation of code

- In most cases, No hard
- Required us to:
  - Depend on and load a library
  - Learn and manually add instrumentation code
- Otherwise:
  - Spans may be too broad
  - Missing

Distributed Tracing - Challenges

1. Manual instrumentation of code
2. Cost

Distributed Tracing - Challenges

1. Manual instrumentation of code
2. Cost
3. Big Traces/ too much data

Distributed Tracing

- A very powerful and essential debugging tool
- Important for confidence to troubleshoot issues in production