Capgemini

22nd Sept 2023

MICROSERVICES ARCHITECTURE AND PATTERNS

Expert Connect Session

Speakers

Shivaraj Mulagund

Architect
Europe C&CA

Shweta Sonawane

Architect
NA

AGENDA

- Update on India Java Community
- Overview of Microservices Architecture
- Challenges with Microservices Architecture
- Microservices Patterns
  - Communication
  - Data Consistency
  - Reliability
- Summary
- Feedback & Quiz

GOAL

LEARN | UPSKILL | NETWORK

COMMUNITY UPDATES

Manjula Samuel

Level 2 Certified Sr. Architect
India Java Community Lead

INDIA JAVA COMMUNITY - UPDATES

	Description	Progress
Quiz Capsules	Quiz allowing individual participation on SpringBoot microservices topics	Season 1 of 8 Episodes has been launched Episode 5 of Season is completed (500+ Participation)
Expert Speak Sessions	Provides an opportunity for associates to **connect with SME in a 1:1 session** and ask questions on a requested technical topic	Monthly Activity First session has been successful with 950+ participants.
Knowledge Sharing Session	Provides an opportunity for our Associates to know the **success stories** from people of successful engagements and learn the best practices from them.	Session being planned on 28th Sep Integration Transformation in Toll
Certification (OCEAN Assessments)	• OCEAN Assessments for Java tracks • Identify professional Java certifications and add in preferred Java certifications lists	Overall Beginner is 77% Overall Practitoner is 75% Pilot phase of Masters is in progress

WHO AM I?

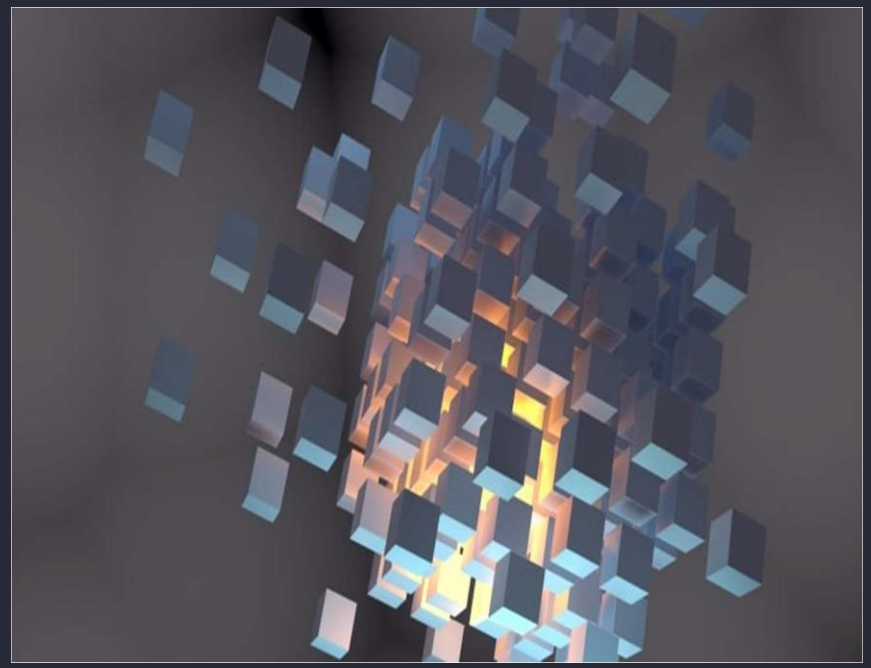
- 14+ years of IT Experience
- Role - Technical Architect
- Certified Architect (L1)
- Expertise in
  - System Design
  - Java & J2EE Development

Shweta Sonawane

OVERVIEW OF **MICROSERVICES**

What is Microservice Architecture?

It is an architecture style which structures a set of services, where each service serves only one business need.



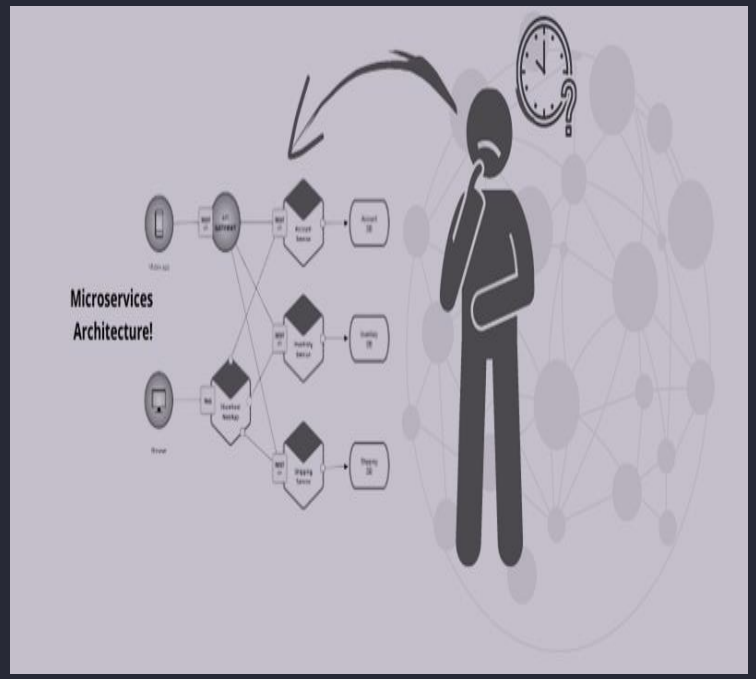
Key Attributes

- Each microservice is a small, loosely coupled distributed service.
- Each microservice is a mini-application that has its own architecture and business logic.

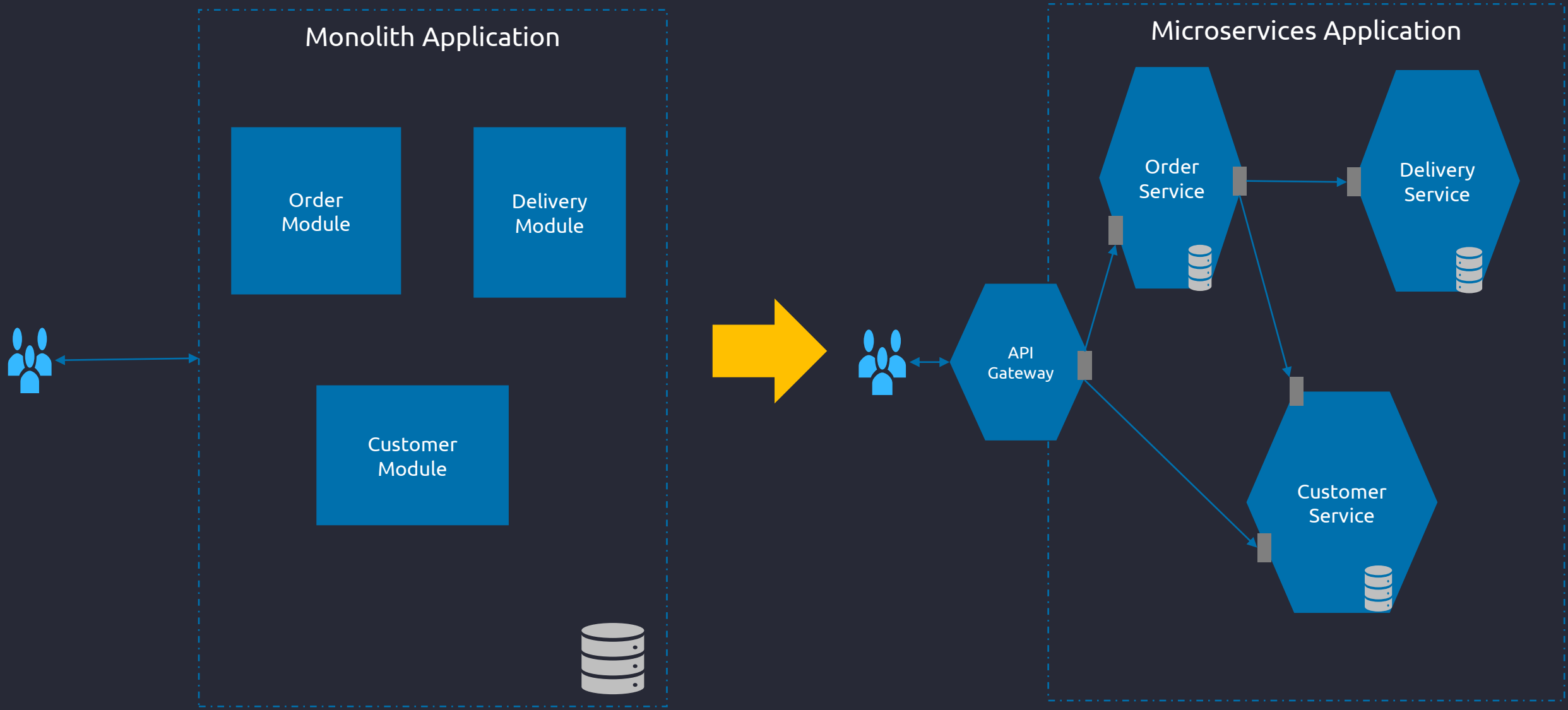
OVERVIEW OF **MICROSERVICES**

When should we choose microservice architecture?

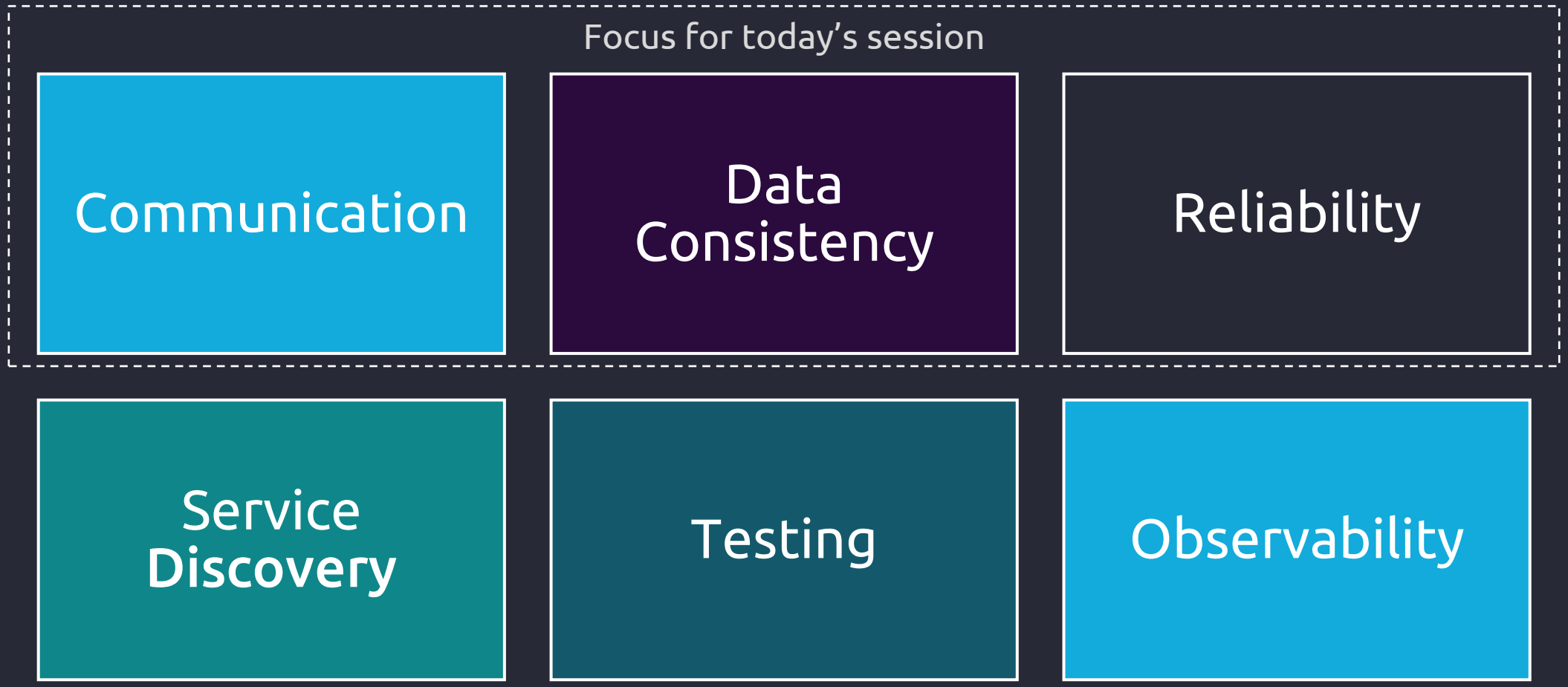
- Complex and Large-Scale Systems
- Agile Development and Continuous Deployment
- Scalability and Performance
- Team Autonomy and Flexibility
- Integration with Third-Party Systems



ONLINE FOOD DELIVERY SYSTEM



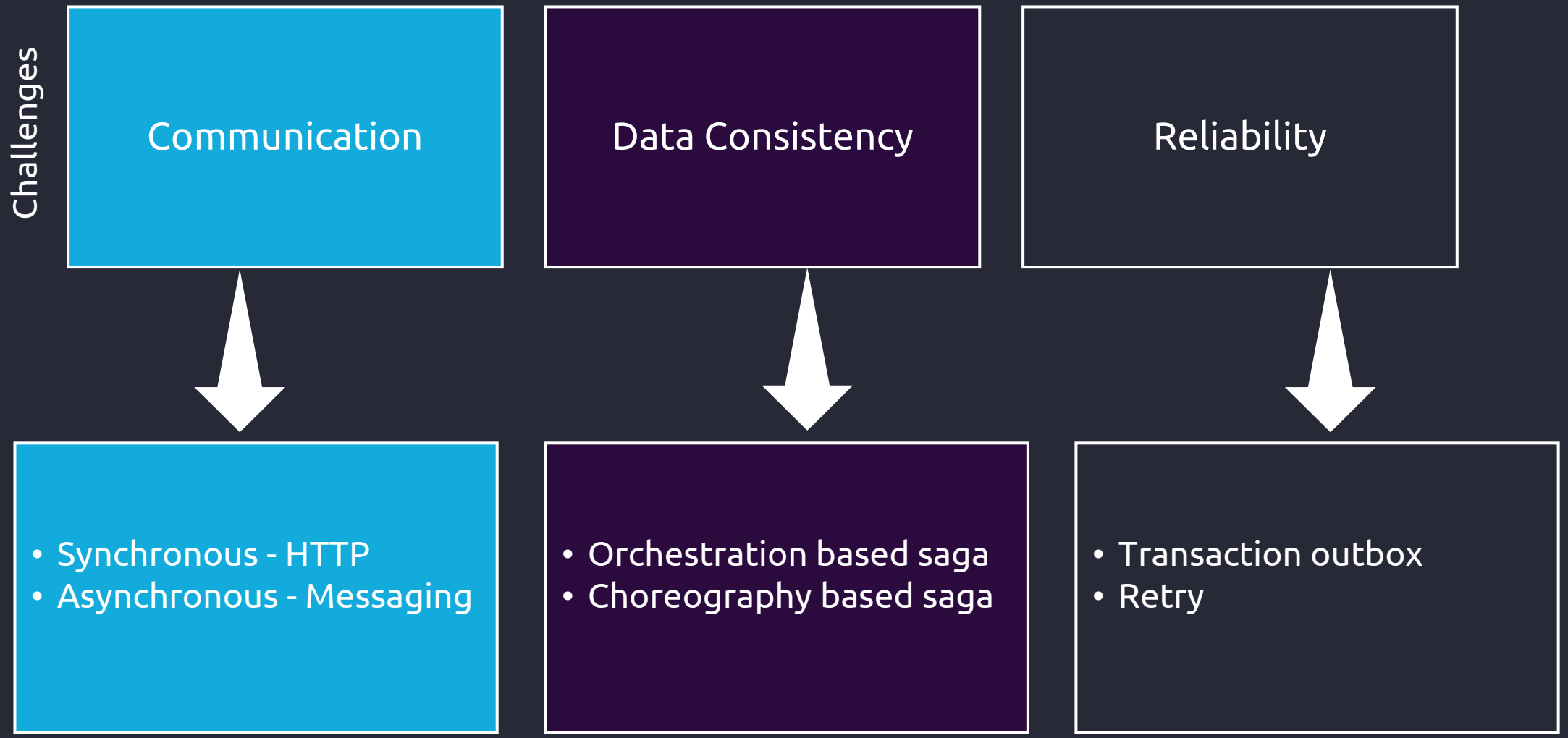
CHALLENGES WITH MICROSERVICES ARCHITECTURE



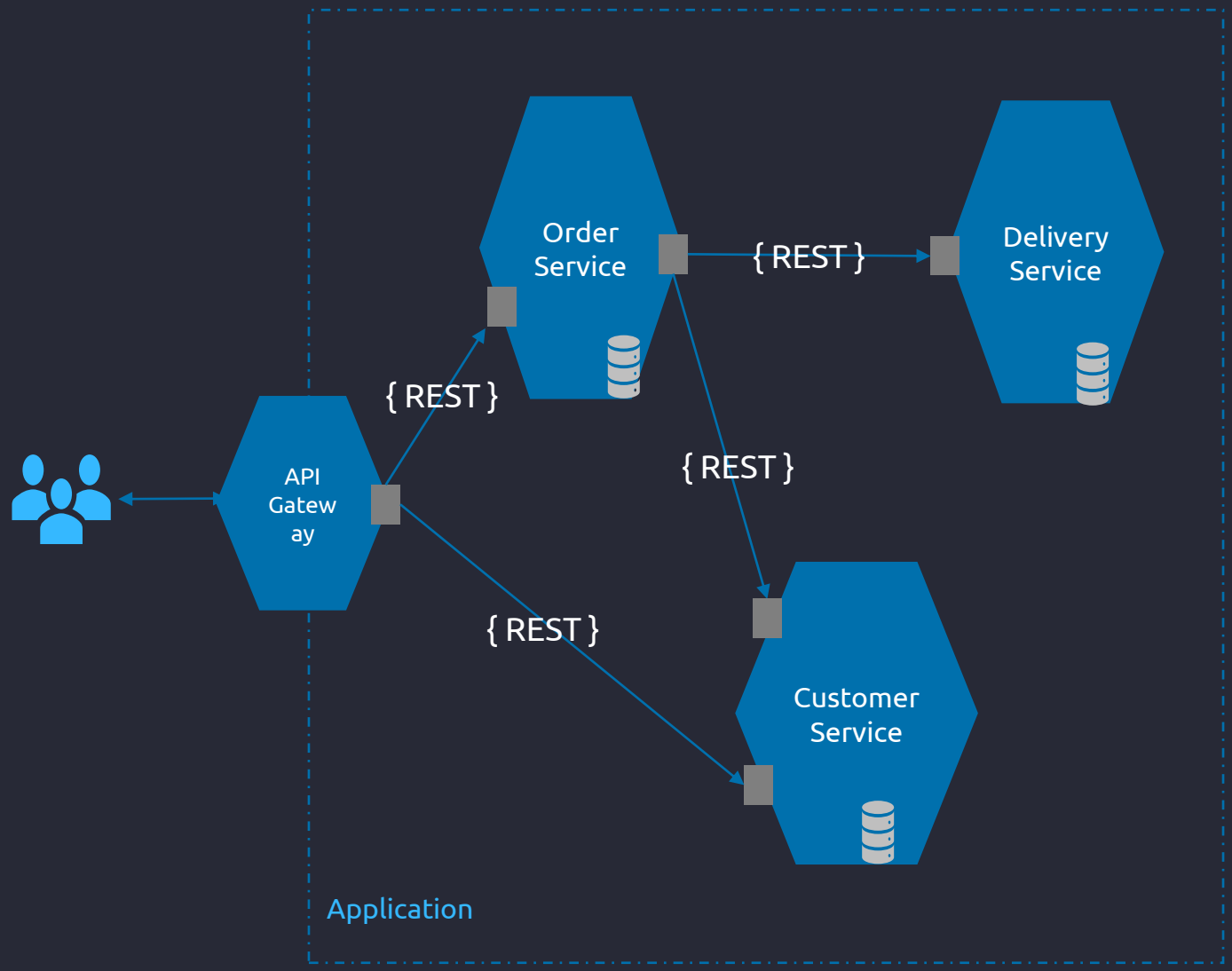
and many more

MICROSERVICES ARCHITECTURE PATTERNS

An **architectural pattern** is a general, reusable resolution to a commonly occurring problem



1. COMMUNICATION PATTERNS – SYNCHRONOUS (HTTP)



Pros:

- Simple and familiar
- Request/reply is easy

Cons:

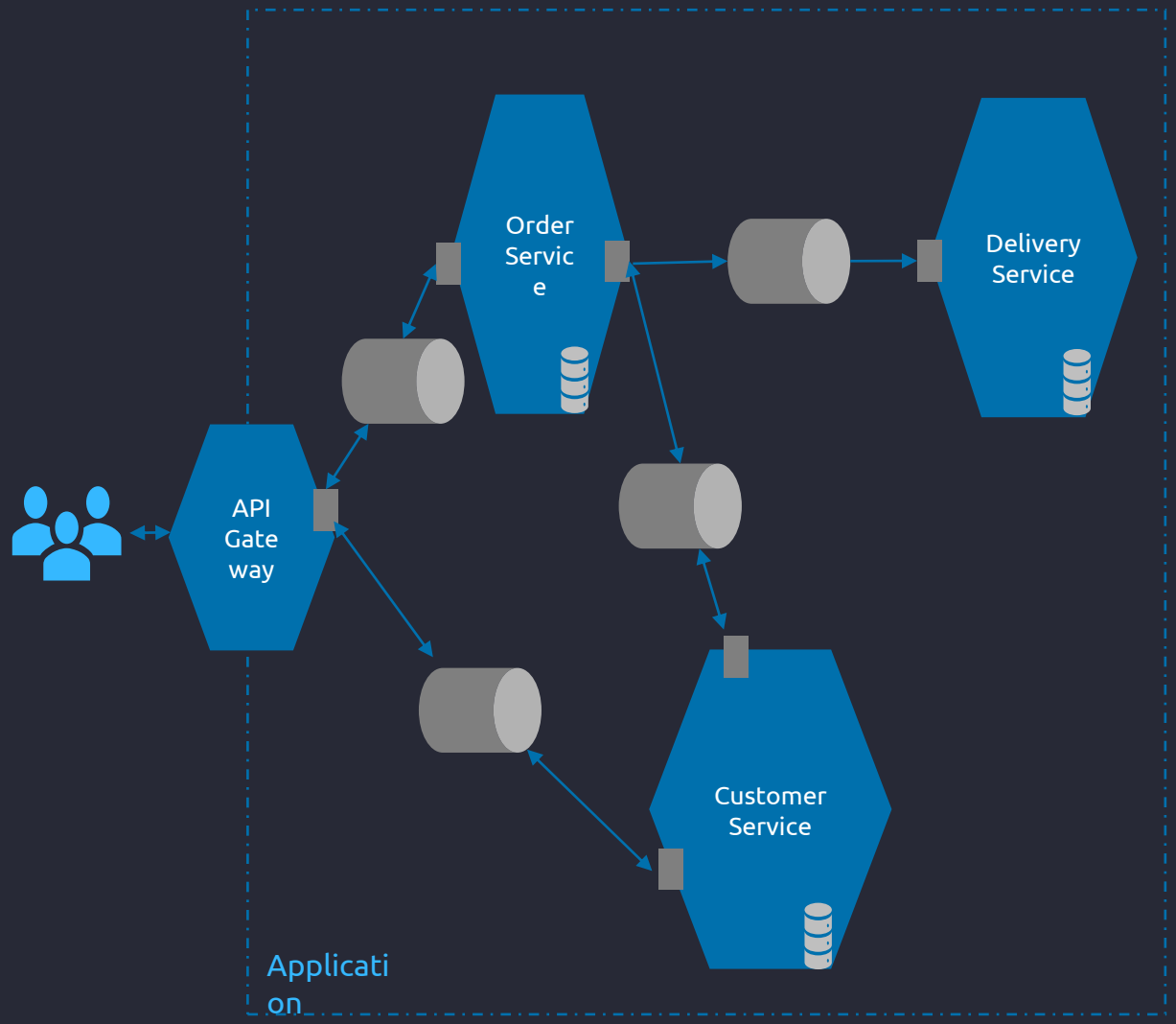
- Tightly coupled and Blocking
- Reduced availability
- Do not support other interactions types such as notifications, publish/subscribe

Technologies:

- REST, Spring

```
@Autowired
private RestTemplate restTemplate;

public Order createOrder(OrderRequest orderRequest) {
  HttpEntity <String> httpEntity = new HttpEntity<OrderRequest>(orderRequest);

  return restTemplate.exchange("http://online-food-app.com/orders",
    HttpMethod.POST, httpEntity, Order.class).getBody();
}
```

1. COMMUNICATION PATTERNS – ASYNCHRONOUS (MESSAGING)



Pros:

- Loosely coupled
- Supports interaction types such as request/reply, publish/subscribe, notifications etc
- Improved availability

Cons:

- Additional complexity due to messaging system which must be highly available
- Request/reply interaction is more complex

Technologies:

- Kafka, RabbitMQ, Active MQ, Spring Cloud streams

```
public Order createOrder(long consumerId, OrderDto OrderDto) {
    ...
    orderRepository.save(order);
    eventPublisher.publish("order.exchange", order);
    ....
    return order;
}
```

```
spring:
  ....
  cloud:
    function:
      definition: ....
    stream:
      rabbit:
        bindings:
          orderEvent-in-0:
            consumer:
              .....
      bindings:
        orderEvent-in-0:
          destination: order.event
          group: customer
          .....
```

AGENDA

- ✓ Update on India Java Community
- ✓ Overview of Microservices Architecture
- ✓ Challenges with Microservices Architecture
- ✓ Microservices Patterns
  - ✓ Communication
  - Data Consistency
  - Reliability
- Summary
- Feedback & Quiz

WHO AM I?

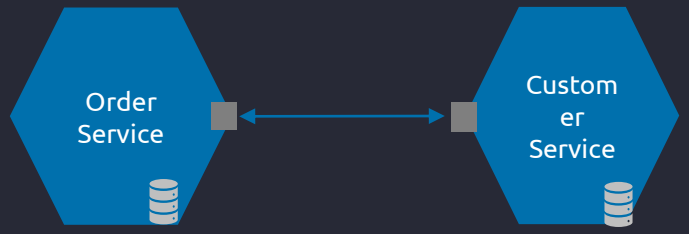
- 15+ years of IT Experience
- Role
  - Technical Architect
- Certified Architect (L1)
- Experience in
  - System Design and Microservices Architecture

Shivaraj Mulagund

2. TRANSACTION MANAGEMENT

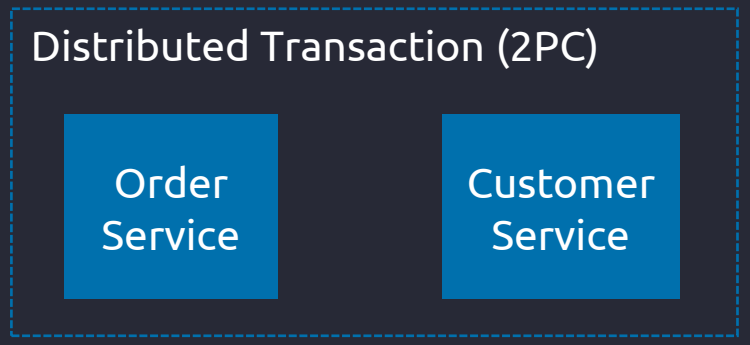
What is a transaction?

Sequence of multiple operations done on a database as single logical unit of work



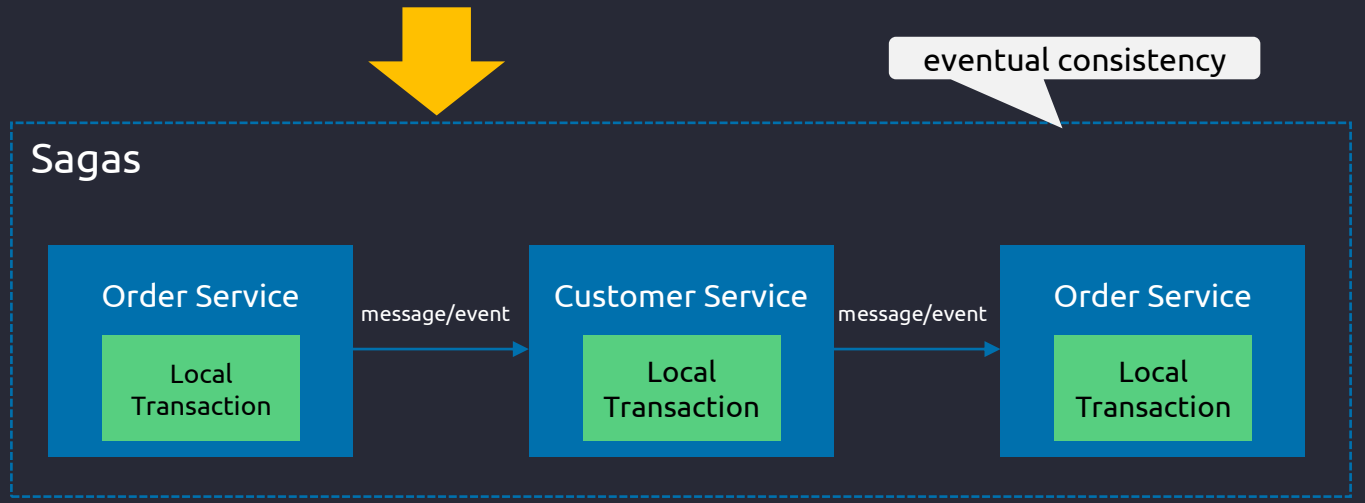

```
BEGIN TRANSACTION
...
SELCT ORDER_TOTAL FROM ORDERS WHERE CUSTOMER_ID = ?
SELCT CREDIT_LIMIT FROM CUSTOMERS WHERE CUSTOMER_ID = ?
INSERT INTO ORDERS .....
....
COMMIT TRANSACTION
```

How do we handle this? 2 Phase Commit? Or Something else?



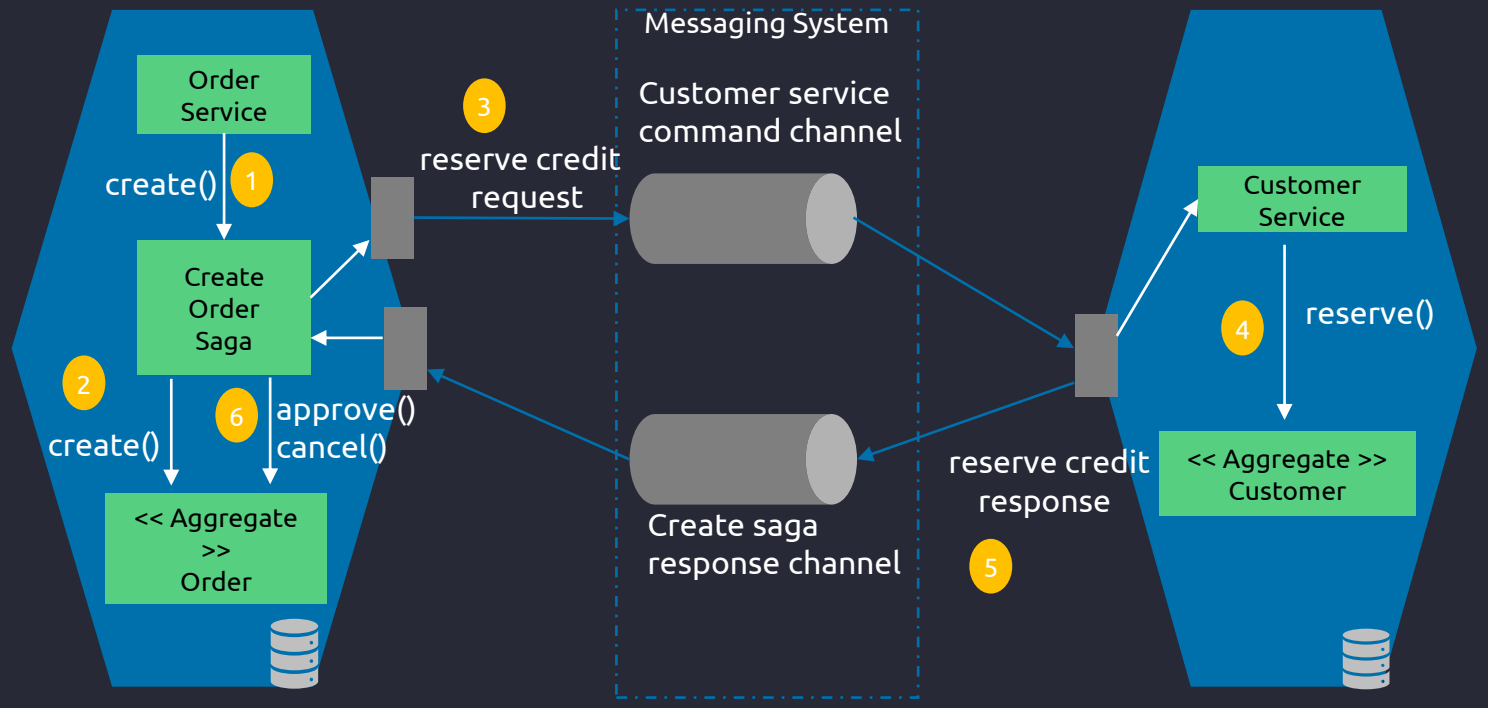
Pros:
- When strong consistency required

Cons:
- Latency
- Single point of failure
- Participant dependency



Picture reference: https://microservices.io/patterns/data/saga.html

2. ORCHESTRATION BASED SAGA PATTERN




```
public Order createOrder(long consumerId, OrderDto OrderDto) {
    ...
    orderRepository.save(order);
    eventPublisher.publish(order);

    CreateOrderSagaState data = new CreateOrderSagaState(order.getId(),
      new OrderDetails(...));
    createOrderSagaManager.create(data, Order.class, order.getId());
    ....
    return order;
}
```

Cons:

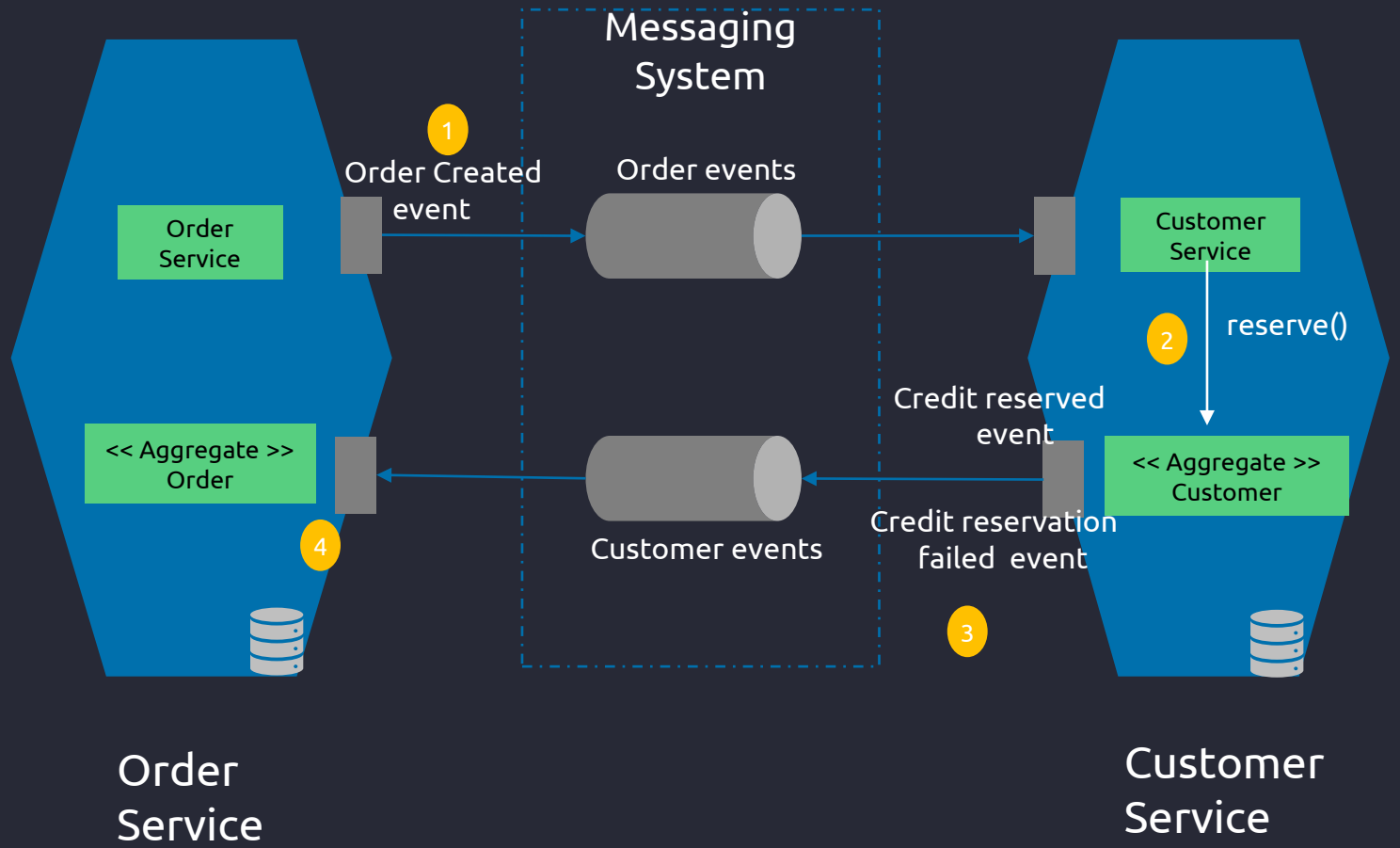
- Complex when there are multiple transaction to be taken care
- Transaction related code in business logic

Order Service

Customer Service

Picture reference: https://microservices.io/patterns/data/saga.html

2. CHOREOGRAPHY BASED SAGA PATTERN

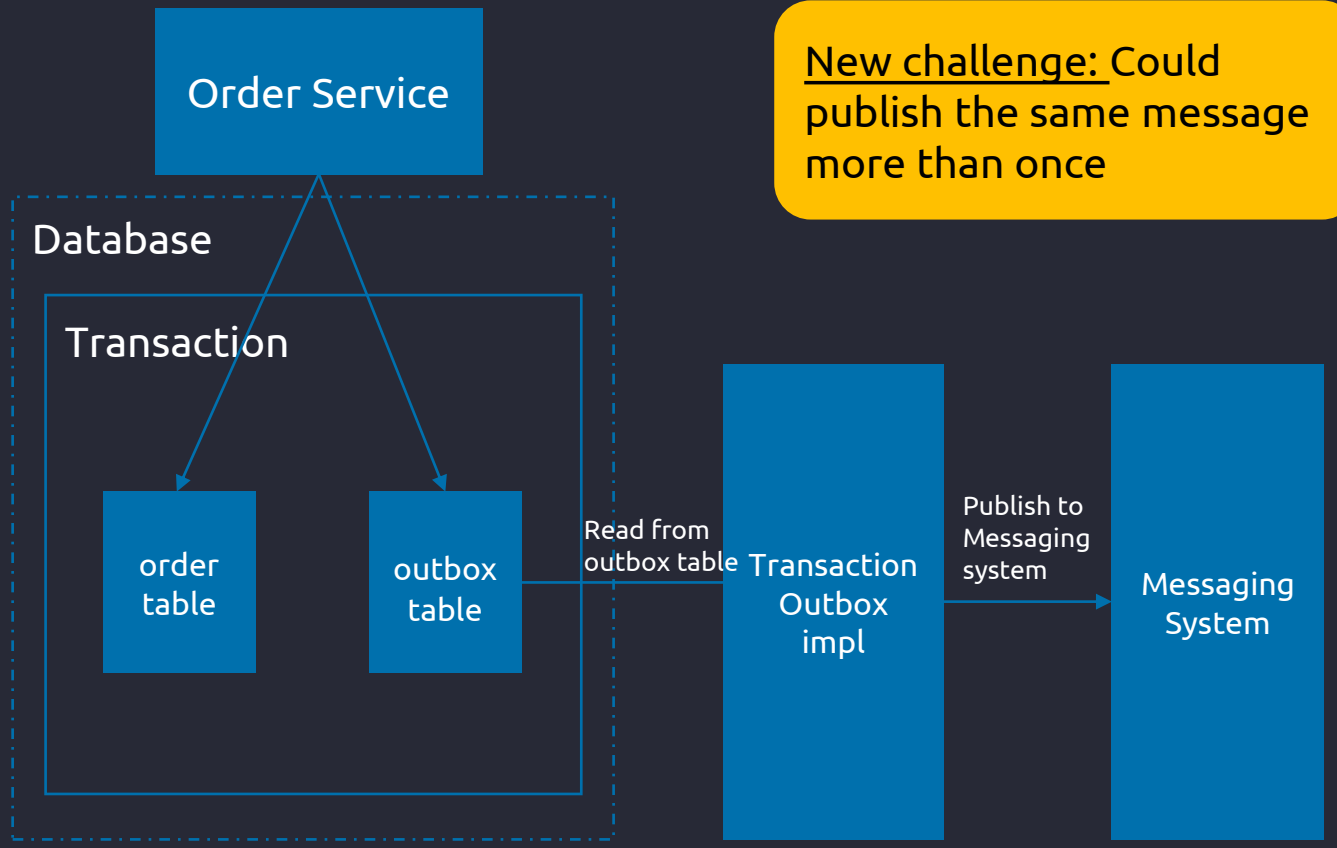



```
public Order createOrder(long consumerId, OrderDto OrderDto) {
    ...
    orderRepository.save(order);
    eventPublisher.publish(order);
    ....
    return order;
}
```

New challenge: Atomic update of DB and publishing message/event

Picture reference: https://microservices.io/patterns/data/saga.html

3. RELIABLE MESSAGING - TRANSACTION OUTBOX PATTERN



New challenge: Could publish the same message more than once

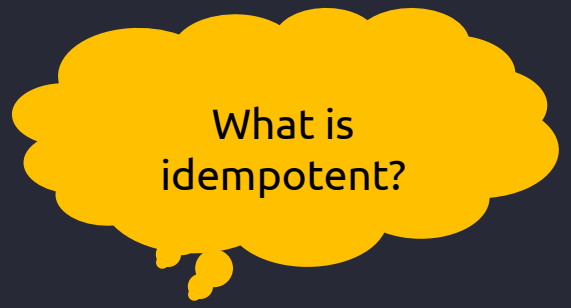
```
public void publishMessage(...) {
 ....
 orderRepository.save(order);
 this.transactionOutbox
      .schedule(..)
      .sendMessage(String exchange, OrderDto orderDto) ;
}

public void sendMessage(String exchange, OrderDto orderDto) {
  this.streamBridge.send(exchange, MessageBuilder.withPayload(orderDto).build());
}
```

Technologies:

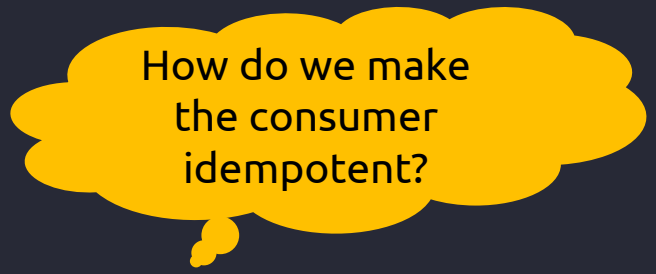
- Opensource libraries for RabbitMQ and Kafka

Picture reference: https://microservices.io/patterns/data/transactional-outbox.html

IDEMPOTENT CONSUMER



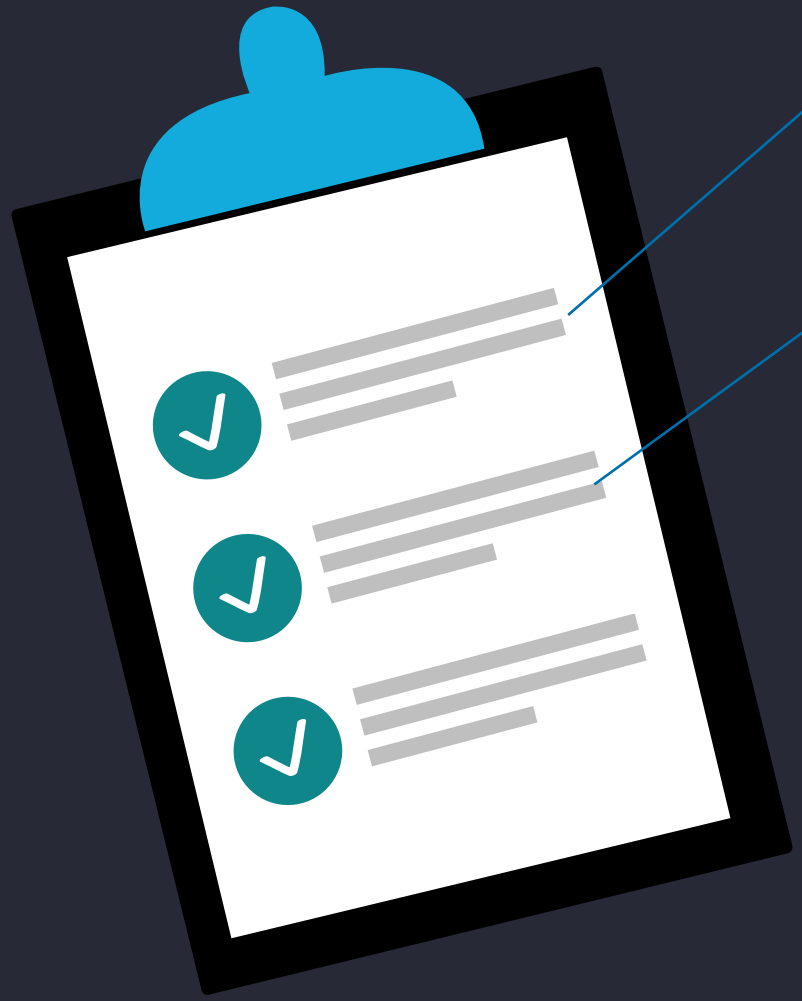
when the same request is sent more than once it produces the result as if it is consumed only once



- Some consumers are naturally idempotent.
  Ex: Updating address
- Others need to be made idempotent
  Ex: Debiting an account

- Keep track of the messages consumed and discard the duplicate messages
- Either we can store in a separate PROCESSED_MESSAGES or store with business entity

SUMMARY



- Consider the requirements and assess the problems that we want address to see if microservices architecture address them
- Always evaluate the pros and cons of each pattern and choose according to the requirements
- Take the infrastructure availability into consideration
- Consider operational complexities
- No perfect pattern which fits in all situations

Capgemini

THANK YOU

Quiz?

(https://kahoot.it/)

Feedback

HTTPS://FORMS.OFFICE.COM/E/MK7X9JI0VK

Feedback @ Expert Connect Session Sept 2023 | India Java Community

Sept 22, 2023

We would like to thank you for your participation. As a next step, we seek your feedback to understand how we can further improve such sessions in future.
For any clarification, please connect with Agarwal, Rajesh (rajesh.b.agarwal@capgemini.com)