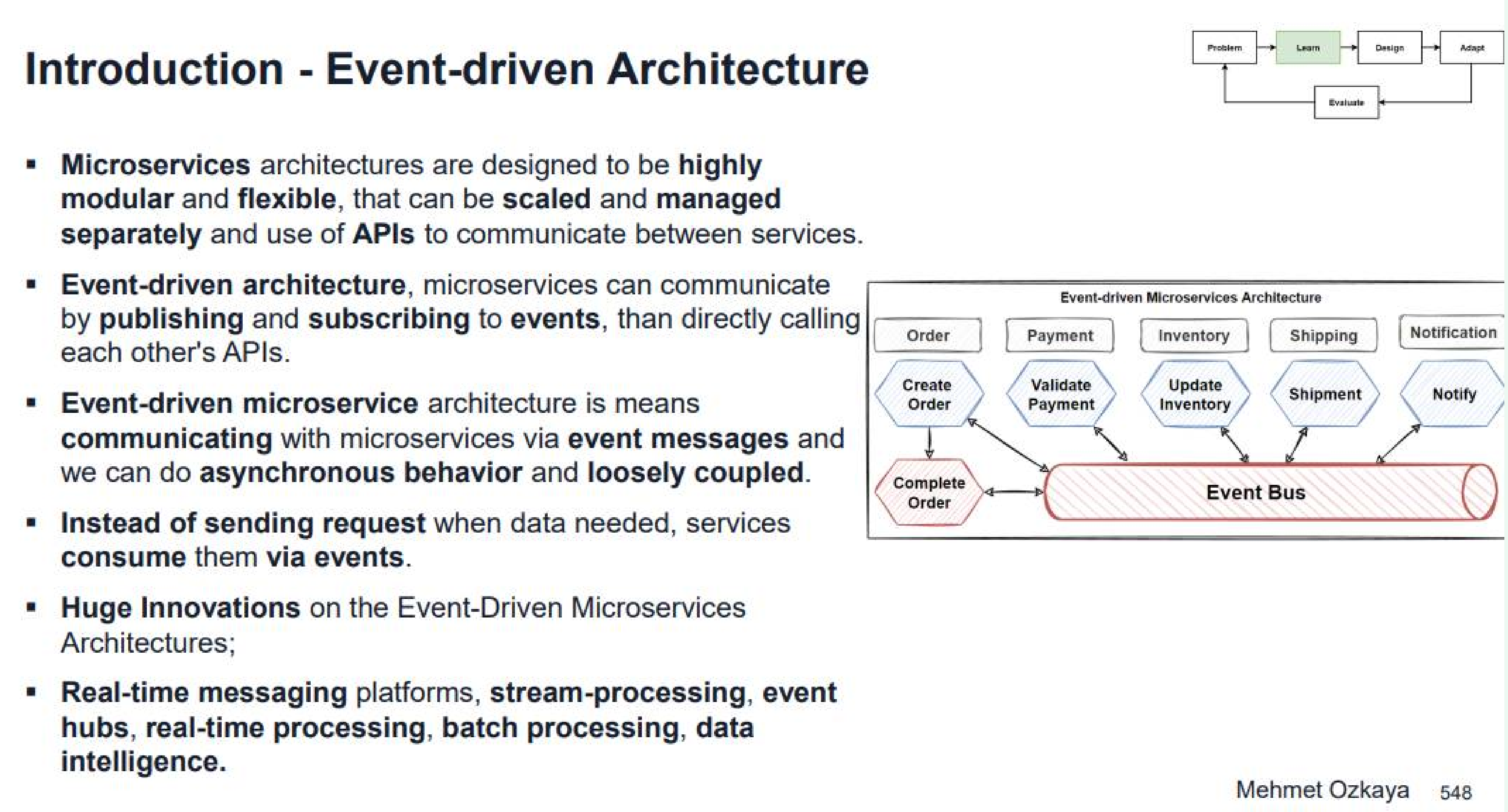
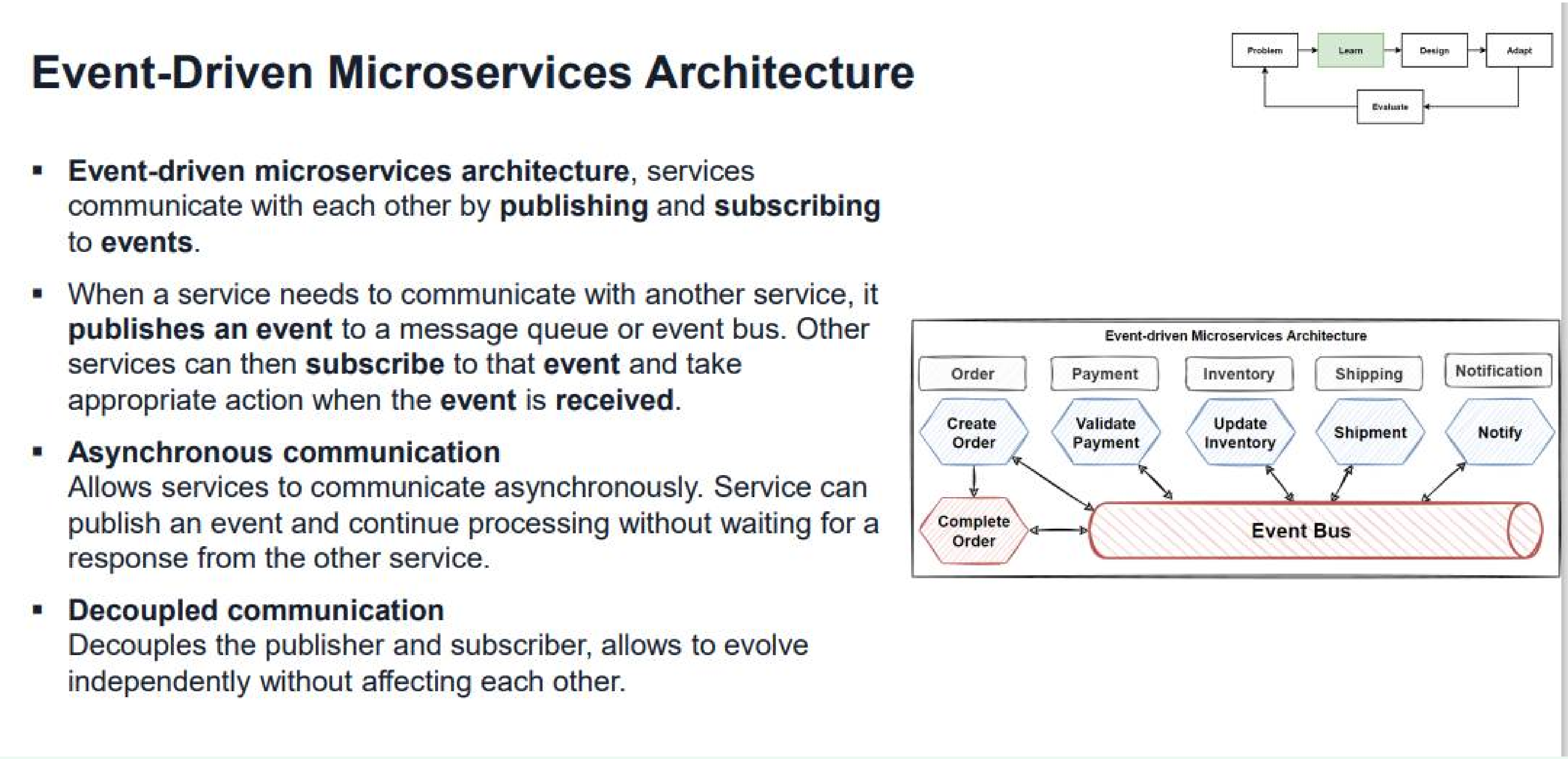
# Event Driven Architecture Overview



## Introduction - Event-driven Architecture

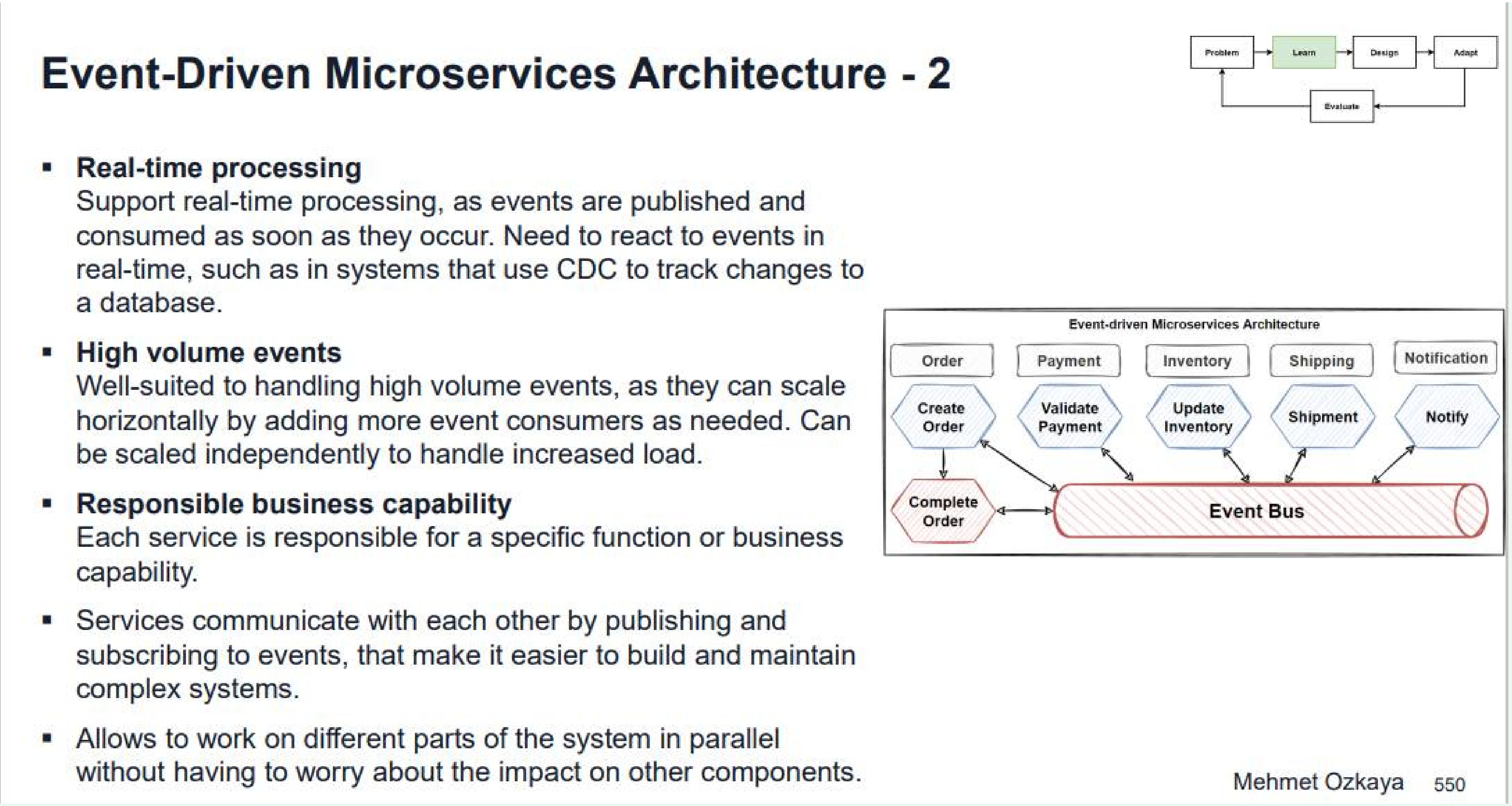
- **Microservices** architectures are designed to be **highly modular** and **flexible**, that can be **scaled** and **managed separately** and use of **APIs** to communicate between services.
- **Event-driven architecture**, microservices can communicate by **publishing** and **subscribing** to **events**, than directly calling each other's APIs.
- **Event-driven microservice** architecture is means **communicating** with microservices via **event messages** and we can do **asynchronous behavior** and **loosely coupled**.
- **Instead of sending request** when data needed, services **consume** them **via events**.
- **Huge Innovations** on the Event-Driven Microservices Architectures;
- **Real-time messaging** platforms, **stream-processing**, **event hubs**, **real-time processing**, **batch processing**, **data intelligence.**

Mehmet Ozkaya 548



# Event-Driven Microservices Architecture

- **Event-driven microservices architecture**, services communicate with each other by **publishing** and **subscribing** to **events**.
- When a service needs to communicate with another service, it **publishes an event** to a message queue or event bus. Other services can then **subscribe** to that **event** and take appropriate action when the **event** is **received**.
- **Asynchronous communication**
  Allows services to communicate asynchronously. Service can publish an event and continue processing without waiting for a response from the other service.
- **Decoupled communication**
  Decouples the publisher and subscriber, allows to evolve independently without affecting each other.



## Event-Driven Microservices Architecture - 2

- **Real-time processing**
  Support real-time processing, as events are published and consumed as soon as they occur. Need to react to events in real-time, such as in systems that use CDC to track changes to a database.
- **High volume events**
  Well-suited to handling high volume events, as they can scale horizontally by adding more event consumers as needed. Can be scaled independently to handle increased load.
- **Responsible business capability**
  Each service is responsible for a specific function or business capability.
- Services communicate with each other by publishing and subscribing to events, that make it easier to build and maintain complex systems.
- Allows to work on different parts of the system in parallel without having to worry about the impact on other components.

Mehmet Ozkaya 550

Kindly explain in details related to Even driven Architecture

The images you've uploaded provide an overview of event-driven architecture in the context of microservices. Here's a detailed breakdown based on the content:

### Event-Driven Architecture in Microservices

Event-driven architecture is a powerful approach used in microservices that allows systems to communicate through events rather than direct API calls. This enables high scalability, loose coupling between services, and supports asynchronous communication.

**Key Concepts:**

1. **Microservices Architecture**:
    - Designed to be modular and flexible.
    - Microservices are independent services that can scale and be managed separately.
    - API communication is typically used between these services.

2. **Event-Driven Architecture**:
    - Microservices communicate by publishing events and subscribing to events, rather than directly calling each other's APIs.
    - This architecture promotes **asynchronous behavior** and **loose coupling** of services.

3. **Event-Driven Communication**:
    - Instead of sending direct requests for data, microservices publish events to an event bus or message queue when certain actions occur (like creating an order).
    - Other services subscribe to these events and react to them without needing to know the source of the event.

4. **Event Bus**:
    - An **event bus** or a message queue is used to broadcast events. In this architecture, services publish their events to the event bus, and any subscribed services pick up and process them.
    - For example, after an order is created, an event is published that other services (like payment or inventory) listen to. When the event is received, they trigger appropriate actions such as validating payment or updating the inventory.

**Benefits of Event-Driven Microservices:**

1. **Asynchronous Communication**:
    - Microservices can communicate asynchronously, meaning that the publishing service doesn't have to wait for a response from the subscriber services. This allows the system to handle multiple requests concurrently and scale effectively.

2. **Decoupling**:

   - The publisher and subscriber are decoupled, allowing independent evolution of services without affecting each other. This isolation between services reduces interdependency.

3. **Real-Time Processing**:
   - Events are processed in real-time, allowing systems to react instantly as events occur (e.g., validating payment right after an order is created).

4. **High Volume Event Handling**:
   - This architecture is well-suited to handling high volumes of events. Services can scale horizontally by adding more event consumers as needed, improving scalability and load handling.

5. **Handling Parallel Business Capabilities**:
   - Each service is responsible for a specific business function (e.g., handling orders, payment validation, etc.), and they can operate in parallel without impacting other parts of the system.

**Key Components:**

1. **Event Bus**:
   - Central component where events are published.
   - Acts as a mediator between services by handling the flow of messages.
   - It helps manage message queuing and delivery, enabling event-driven communication between microservices.

2. **Publishers**:
   - Microservices that generate and send events to the event bus (e.g., an "Order" service that publishes an event when a new order is created).

3. **Subscribers**:
   - Services that subscribe to specific events and react when those events occur (e.g., "Payment" or "Inventory" services that react to the "Order Created" event).

**Real-Life Examples:**

- **E-commerce platforms**: When a customer places an order, an event is generated. The payment service, inventory service, and shipping service subscribe to this event and react accordingly. The payment service handles the payment, the inventory service updates the stock, and the shipping service starts processing the shipment.

- **Notification systems**: After an order is completed, a notification service can subscribe to the event and send the customer a confirmation email or SMS.

In summary, event-driven architecture makes systems more modular, scalable, and able to handle real-time events in a more flexible, loosely-coupled manner. This architecture is particularly useful for large-scale, distributed systems, allowing each microservice to work independently and asynchronously.

ChatGPT can make mistakes. Check important info.