GERMAN INTERNATIONAL UNIVERSITY

الجامعة الألمانية الدولية

Faculty of Informatics and Computer Science

Software Design & Architecture

Topics 5 & 6

Architecture Styles I & II

Dr. Ahmed Maghawry

Contents

1. Introduction

Software architecture

Refers to the high-level structure and organization of a software system.

It involves making design decisions that determine how different components and modules of the system interact with each other, how data flows within the system, and how the system as a whole achieves its intended functionality and quality attributes.

Importance of a good software architecture

- **System Understanding and Communication:**
  - Software architecture serves as a visual blueprint, facilitating effective communication and collaboration among stakeholders.
- **System Quality and Maintainability:**
  - Well-designed architecture ensures desired system attributes, reduces risks, and enables cost-effective maintenance, enhancing system reliability and longevity.
- **System Evolution and Flexibility:**
  - Architecture's modularity and flexibility allow for easier system adaptation and future enhancements, accommodating changing requirements and technological advancements.
- **Reusability and Interoperability:**
  - Architecture promotes component reuse, fostering efficiency and interoperability with external systems, reducing development time and promoting collaboration.
- **Risk Mitigation and Cost Reduction:**
  - Effective architecture identification and mitigation of risks, minimizing project failures, and controlling costs, leading to successful and cost-effective software development.
- **Performance and Efficiency:**
  - Architecture optimizes system performance, utilizing resources efficiently, and enhancing overall efficiency, resulting in a high-performing and responsive software system.

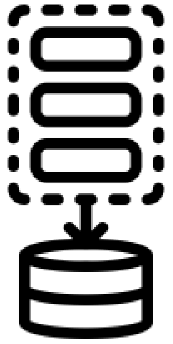
2. Monolithic Architecture

Monolithic architecture

an architecture style where an entire application is built as a single, self-contained unit.

In this approach, all the components and functionalities of the system are tightly integrated and deployed together, sharing the same codebase and database.

The monolithic architecture typically consists of three main components: the user interface (UI) layer, the business logic layer, and the data storage layer.



Advantages:

- **Simplicity:**
  - Straightforward to develop and test since all components are tightly integrated.
- **Performance:**
  - The absence of network calls between components within the monolith can result in faster performance.
- **Development Productivity:**
  - With a single codebase, developers can work more efficiently, as they have a holistic view of the application.

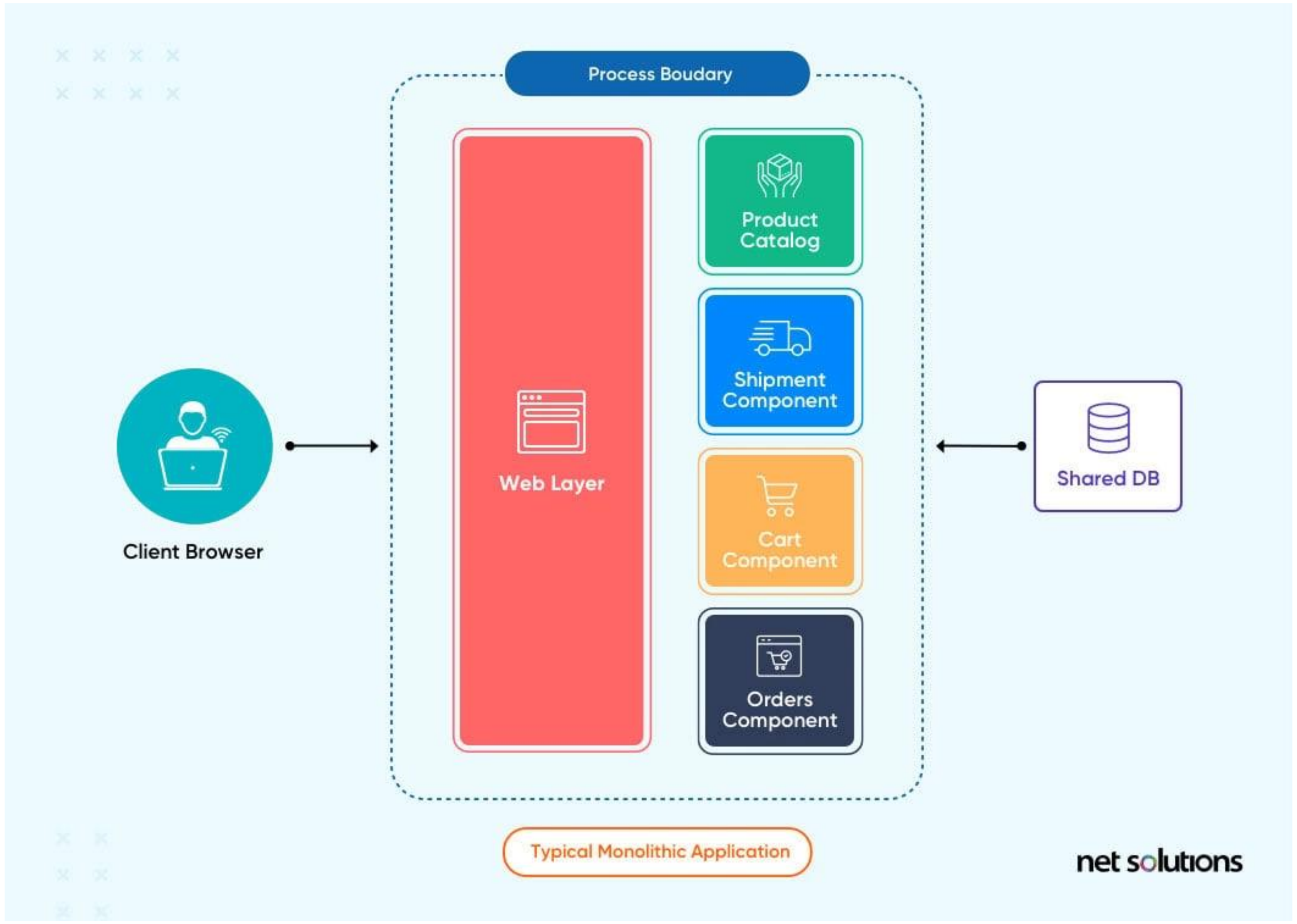
Characteristics:

- **Tight Coupling:**
  - All components within the monolith are tightly coupled, meaning they directly depend on each other and are difficult to modify or replace independently.
- **Single Deployment Unit:**
  - The entire application is packaged and deployed as a single unit, making it easier to manage and deploy but potentially resulting in longer deployment times and increased complexity.
- **Shared Database:**
  - The monolith typically uses a single shared database for all its components, making it convenient for data access and management but potentially leading to scalability and performance challenges.

Disadvantages:

- **Scalability:**
  - the entire application needs to be replicated, even if only a specific component requires additional resources.
- **Flexibility and Extensibility:**
  - Any changes may impact the entire system.
- **Technology Limitations:**
  - limit the ability to leverage new technologies or frameworks for specific components.
- **Deployment Complexity:**
  - Deploying updates or bug fixes to a monolithic system requires deploying the entire application.

2. Monolithic Architecture



[1]

3. Client-Server Architecture

Client-server architecture

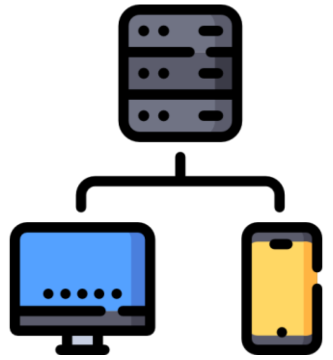
is a software design pattern where a system is divided into two main components: the client and the server. The client, typically a user interface or application running on a user's device, requests services or resources from the server. The server, on the other hand, processes these requests, performs the necessary computations or data retrieval, and sends the results back to the client.

Types of client-server architectures

- **Two-Tier Architecture:**
  - In this basic form, the client directly communicates with the server to request and receive data or services. There is no intermediate layer between the client and server.
- **Three-Tier Architecture:**
  - This architecture introduces an additional layer called the middleware or application server. The client interacts with the middleware, which acts as an intermediary between the client and the backend server, handling business logic and data processing.

Characteristics :

- **Separation of Concerns:**
  - The client and server have distinct roles and responsibilities, with the client focusing on user interaction and the server handling data processing and storage.
- **Communication:**
  - Clients and servers communicate with each other over a network, typically using protocols like HTTP, TCP/IP, or WebSocket.
- **Scalability:**
  - Client-server architecture allows for scaling the server component independently to handle increasing client requests by adding more server resources or using load balancing techniques.
- **Resource Sharing:**
  - Servers provide shared resources or services that can be accessed by multiple clients, enabling centralized data management and consistency.



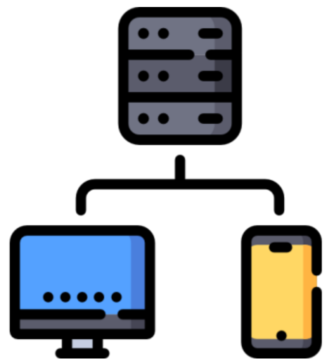
3. Client-Server Architecture

Advantages of client-server architecture

- **Scalability:**
  - The server component can be scaled independently to handle increased client demand, ensuring efficient resource utilization.
- **Centralized Data Management:**
  - Data is stored and managed on the server, promoting data consistency and eliminating data redundancy across multiple clients.
- **Security:**
  - Centralized server control allows for implementing robust security measures to protect sensitive data and resources.
- **Ease of Maintenance:**
  - Separation of client and server components simplifies maintenance and updates, as changes can be made on the server without affecting client devices.

Disadvantages of client-server architecture

- **Dependency on Server:**
  - Clients rely on the availability and performance of the server, and a server failure can impact all connected clients.
- **Network Dependency:**
  - Communication over a network introduces latency, and a slow or unreliable network connection can affect overall system performance.



4. Service Oriented Architecture

(SOA)

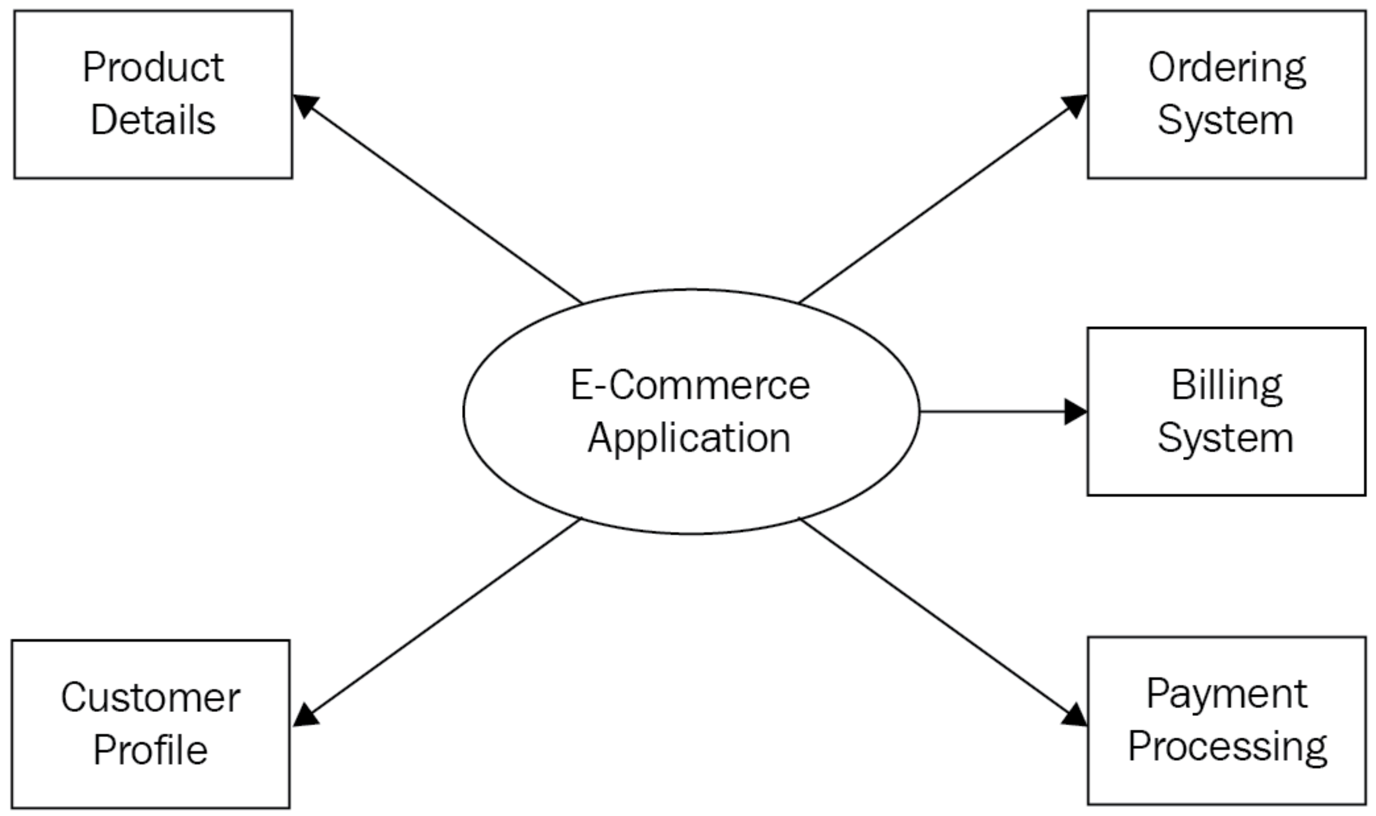
Is an architectural style that focuses on organizing software systems as a collection of loosely coupled, interoperable services.

It promotes the idea of building software applications by composing services that communicate with each other over network.

Key principles of Service-Oriented Architecture include:

- **Service encapsulation:**
  - Each service hides its internal implementation details and exposes only the necessary functionality through well-defined interfaces.
- **Loose coupling:**
  - Services are designed to be loosely coupled, meaning they can evolve independently without affecting other services. Changes to one service should not require changes in other services.
- **Service reusability:**
  - Services are designed to be reusable across multiple applications or scenarios. They can be composed and orchestrated to create new applications or business processes.
- **Service composition:**
  - Multiple services can be combined to create more complex and higher-level services. This composition can be done dynamically at runtime to meet specific business requirements.
- **Service discoverability:**
  - Services are made discoverable through service registries or directories. This allows service consumers to locate and invoke the desired services based on their capabilities.
- **Service autonomy:**
  - Services have control over their own behavior and data. They can be developed and deployed independently by different teams or organizations.
- **Service interoperability:**
  - Services can be developed using different technologies and platforms, as long as they can communicate through standardized interfaces and protocols.

4. Service Oriented Architecture



5. Microservice Architecture

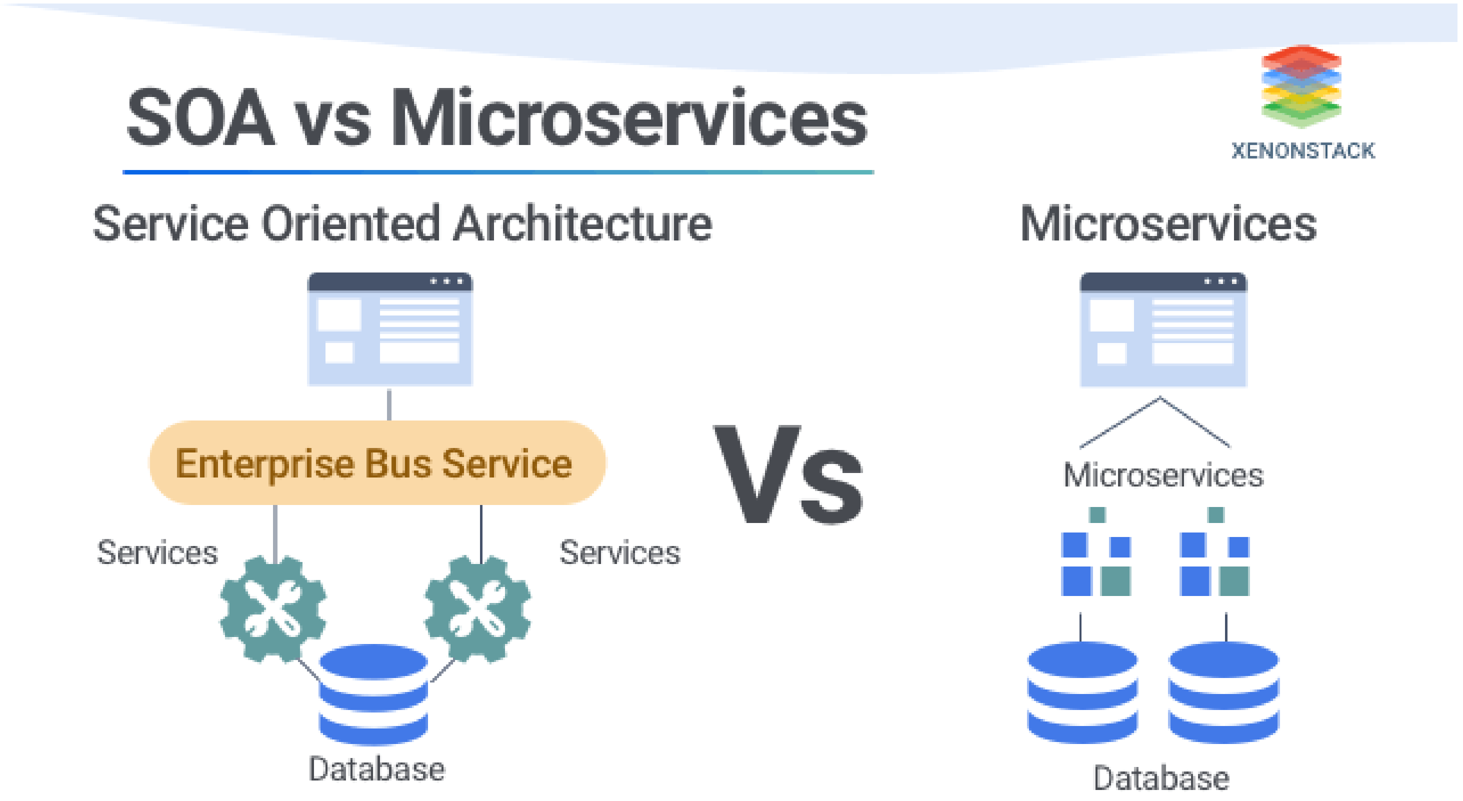
Microservice architecture

is an architectural style that structures a software application as a collection of small, independent, and loosely coupled services. Each service in a microservice architecture represents a specific business capability and runs as a separate process or container, communicating with other services through well-defined APIs.

Key characteristics of microservice architecture include:

- **Service independence:**
  - Each microservice is self-contained and has its own business logic and data storage. It can be developed, deployed, and scaled independently of other services.
- **Decentralized data management:**
  - Each microservice manages its own database or data store, ensuring that data is private to that service. Services can use different databases or technologies based on their specific needs.
- **Communication via APIs:**
  - Microservices communicate with each other using lightweight protocols such as HTTP/REST, messaging queues, or event-driven mechanisms. APIs provide a standardized way for services to interact and exchange data.
- **Autonomous teams:**
  - Microservices are typically developed and maintained by small, cross-functional teams. Each team has ownership and responsibility for one or more services, allowing for faster development cycles and easier maintenance.
- **Fault isolation:**
  - Since microservices run independently, failures or issues in one service do not necessarily affect the entire application. Services can be isolated and have their own fault tolerance mechanisms.
- **Scalability and elasticity:**
  - Each microservice can be scaled independently based on its specific load and requirements. This enables efficient resource utilization and the ability to handle varying levels of demand.
- **Continuous deployment:**
  - Microservice architecture aligns well with continuous integration and continuous deployment practices. Teams can deploy individual services without disrupting the entire application, enabling faster release cycles and feature updates.

5. Microservice Architecture



[2]

6. Event-Driven Architecture

Event-driven architecture (EDA)

is an architectural style in which systems and software components communicate and react to events that occur within the system or in the external environment. In an event-driven architecture, the flow of data and control is driven by the occurrence of events and the reactions triggered by those events.

Key concepts in event-driven architecture:

- **Events:**
  - An event represents a significant occurrence or change in the system or external environment. It can be a user action, a sensor reading, a message arrival, or any other notable activity. Events are typically represented as structured data and contain information about what happened.
- **Event producers:**
  - Event producers are components or systems that generate events. They publish events to a messaging system or an event bus, making them available for other components to consume.
- **Event consumers:**
  - Event consumers are components or systems that subscribe to events and react to them. They receive events from the messaging system or event bus and perform actions based on the event data. This can involve updating data, triggering processes, or invoking other services.
- **Event-driven messaging:**
  - Events are typically communicated between components using event-driven messaging systems. These messaging systems allow producers to publish events to specific topics or channels, and consumers to subscribe to those topics of interest. This decoupling between producers and consumers enables loose coupling and flexibility in the system.
- **Event processing:**
  - Event processing involves handling and reacting to events. It can include event filtering, transformation, aggregation, correlation, or enrichment. Event processing logic is implemented within event consumers or dedicated event processors.
- **Event-driven workflows:**
  - Event-driven architecture allows the creation of workflows or processes that are driven by events. Events trigger the execution of specific steps or actions in the workflow, allowing for dynamic and flexible process orchestration.

6. Event-Driven Architecture

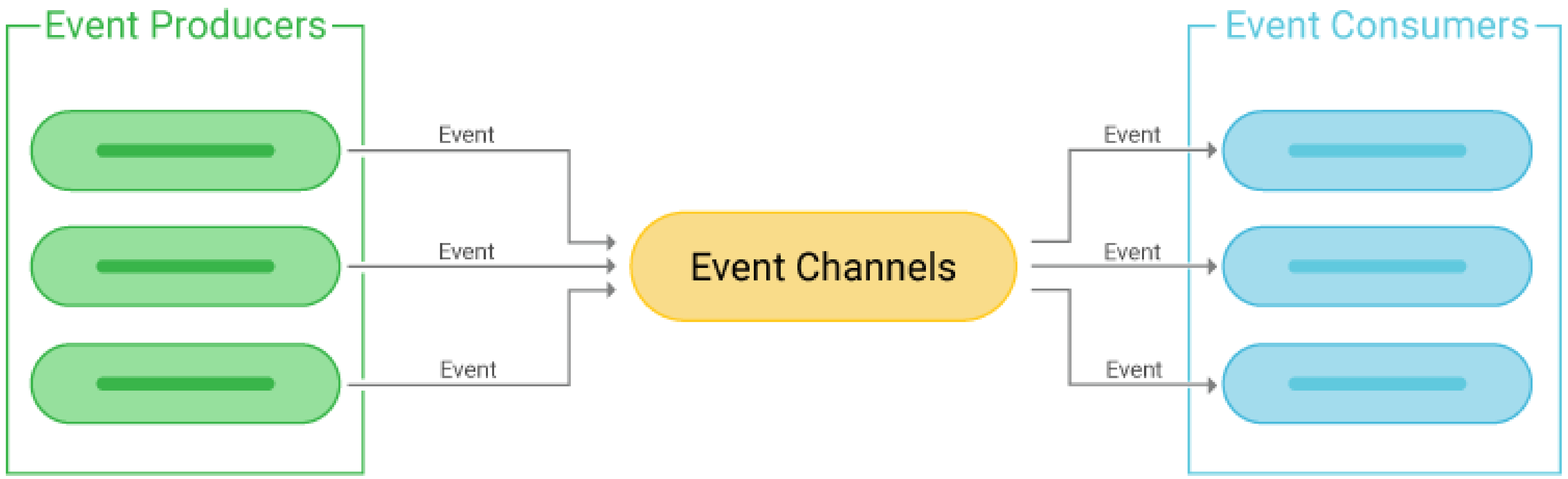
Event-driven architecture (EDA)

is an architectural style in which systems and software components communicate and react to events that occur within the system or in the external environment. In an event-driven architecture, the flow of data and control is driven by the occurrence of events and the reactions triggered by those events.

Advantages:

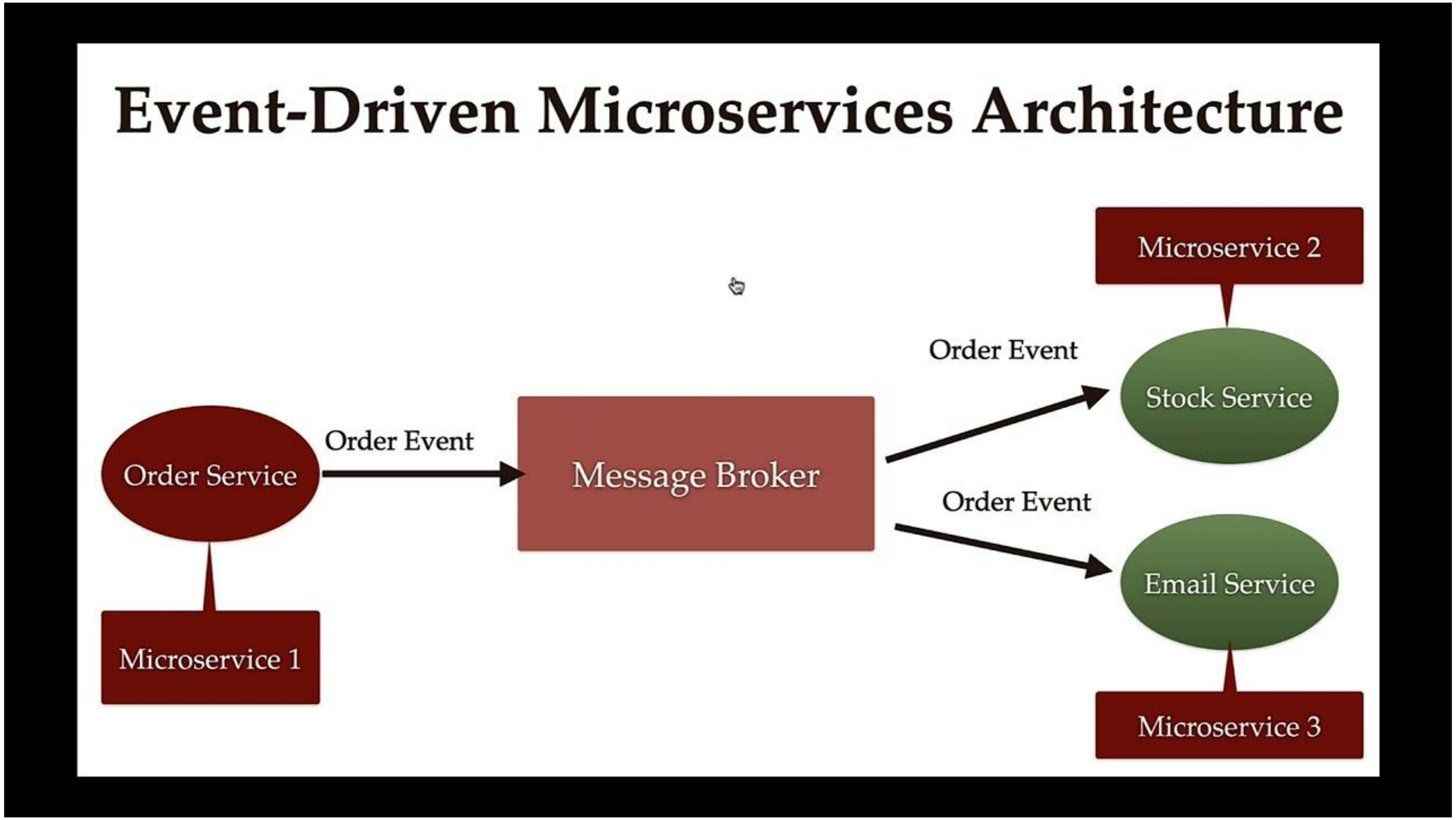
- **Loose coupling:**
  - Components are decoupled and communicate through events, enabling independence and flexibility in system design and evolution.
- **Scalability:**
  - Event-driven systems can scale horizontally by adding more event consumers or processors to handle increased event loads.
- **Responsiveness:**
  - Event-driven systems can react in real-time to events, enabling rapid and dynamic responses to changes or triggers.
- **Extensibility:**
  - New components can be easily added to the system by subscribing to relevant events, without the need for extensive modifications to existing components.
- **Asynchronicity:**
  - Event-driven systems are inherently asynchronous, allowing components to process events independently and at their own pace.

6. Event-Driven Architecture



7. Mix and Match !

Hybrid architecture styles can be designed and used according to business and technical needs!

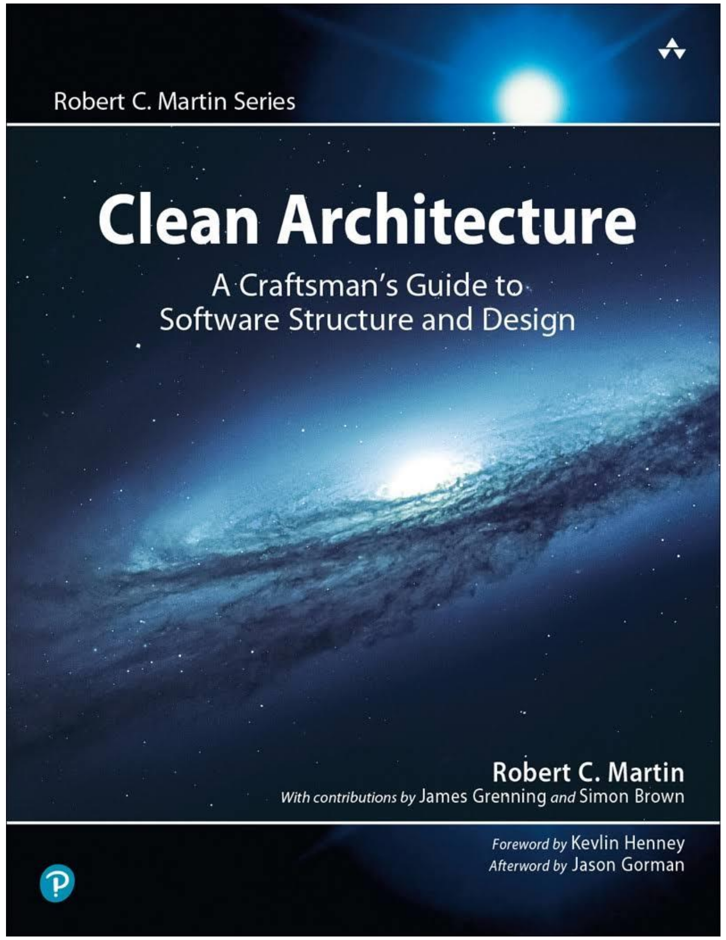


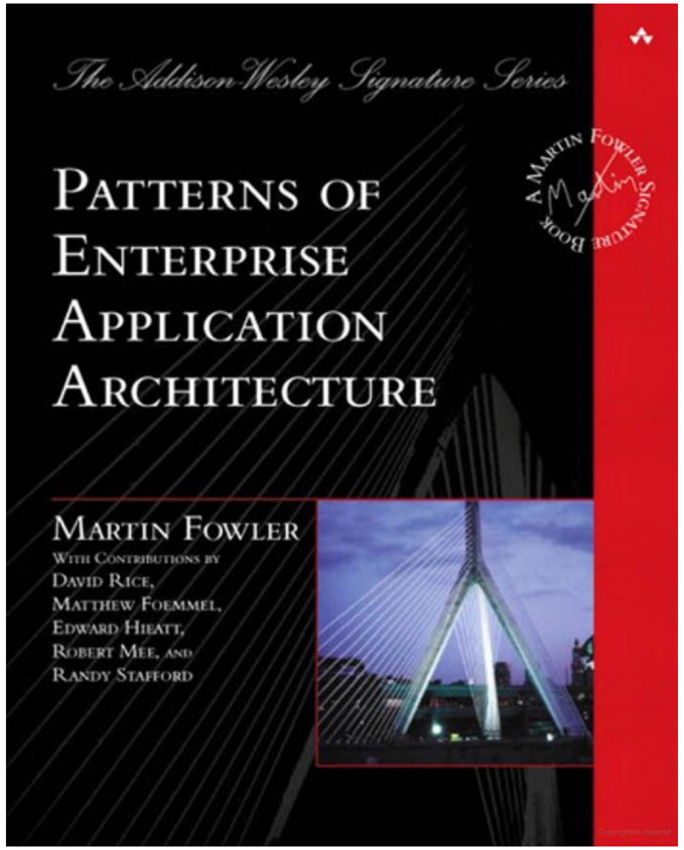
8. Conclusion

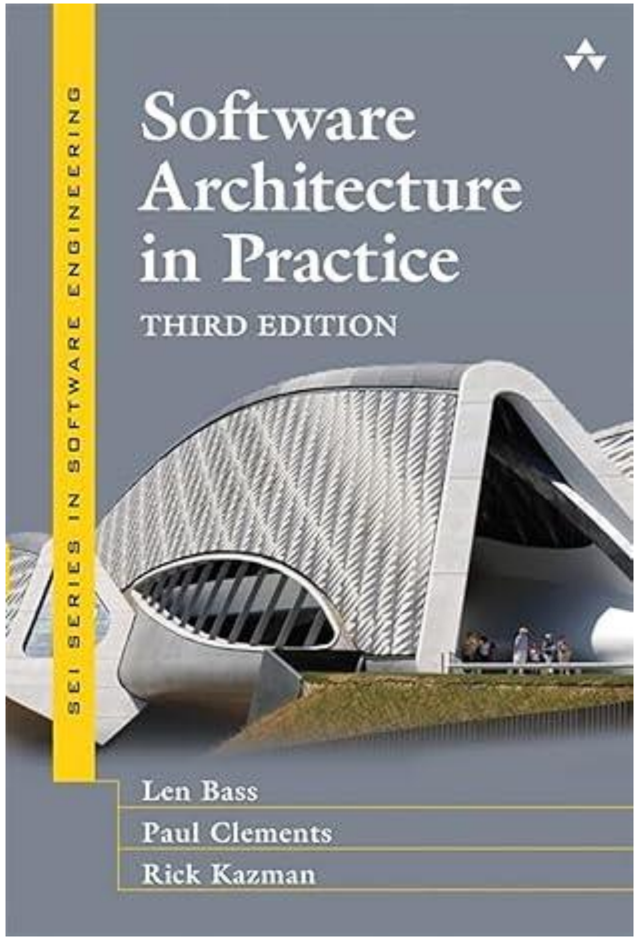
- **Software architecture plays a crucial role in designing and structuring complex software systems.**
- **It provides a blueprint for system organization, communication, and scalability.**
- **Each architecture style has its own strengths, trade-offs, and suitability for different scenarios.**
- **Monolithic Architecture:**
  - Offers simplicity in development but lacks scalability and flexibility.
  - Suitable for small-scale applications with stable requirements.
- **Client-Server Architecture:**
  - Separates client and server components, enabling centralized data processing and storage.
  - Commonly used for web-based applications and distributed systems.
- **Service-Oriented Architecture (SOA):**
  - Focuses on organizing applications into loosely coupled services.
  - Provides reusability, flexibility, and interoperability through standardized interfaces.
- **Microservice Architecture:**
  - Emphasizes the development of small, independent services.
  - Enables scalability, agility, and autonomous team ownership.
- **Event-Driven Architecture (EDA):**
  - Reacts to events in the system or environment, enabling real-time responsiveness.
  - Offers loose coupling and flexibility in handling dynamic workflows.

References

- Martin, R. C. (2017). Clean Architecture: A Craftsman's Guide to Software Structure and Design. Prentice Hall.
- Bass, L., Clements, P., & Kazman, R. (2012). Software Architecture in Practice (3rd ed.). Addison-Wesley Professional.
- Fowler, M. (2002). Patterns of Enterprise Application Architecture. Addison-Wesley Professional.


Robert C. Martin Series
Clean Architecture
A Craftsman's Guide to Software Structure and Design
Robert C. Martin
With contributions by James Grenning and Simon Brown
Foreword by Kevlin Henney
Afterword by Jason Gorman


The Addison-Wesley Signature Series
Patterns of Enterprise Application Architecture
A Martin Fowler Signature Book
Martin Fowler
With Contributions by David Rice, Matthew Foemmel, Edward Hieatt, Robert Mee, and Randy Stafford


SEI Series in Software Engineering
Software Architecture in Practice
Third Edition
Len Bass
Paul Clements
Rick Kazman

GIU GERMAN INTERNATIONAL UNIVERSITY

الجامعة الألمانية الدولية

Faculty of Informatics and Computer Science

Questions?