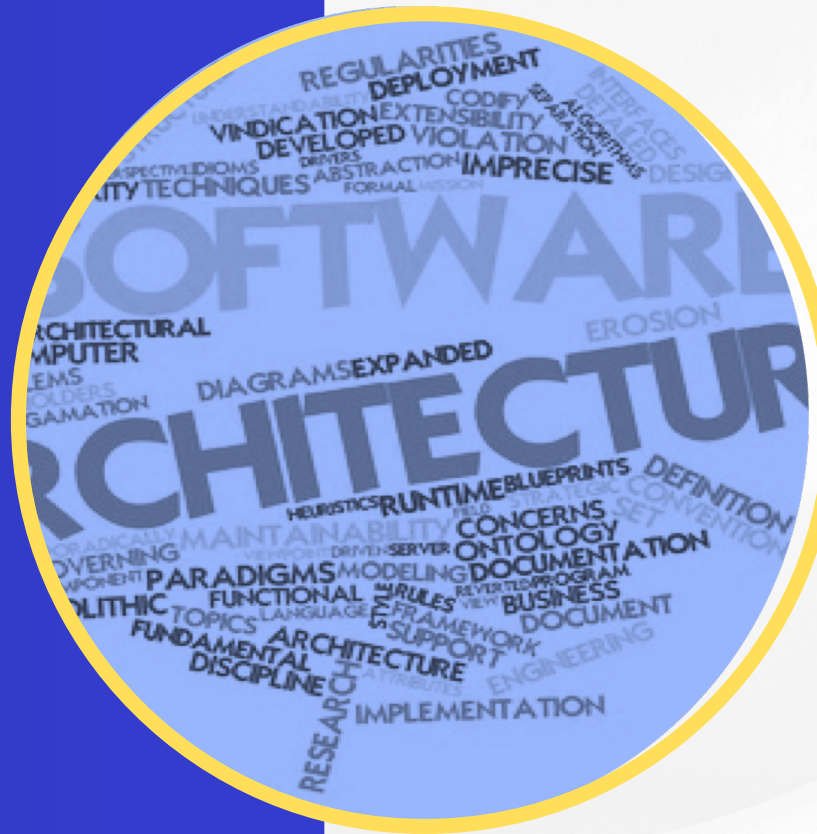
University of Tripoli – Faculty of Information Technology

Software Engineering Department

Software Architecture & Design

ITSE411



y Marwa Solla

Software architecture and design

for modern large-scale systems

Lecture 5:

Architectural styles(patterns)

What We Learn In This Lecture

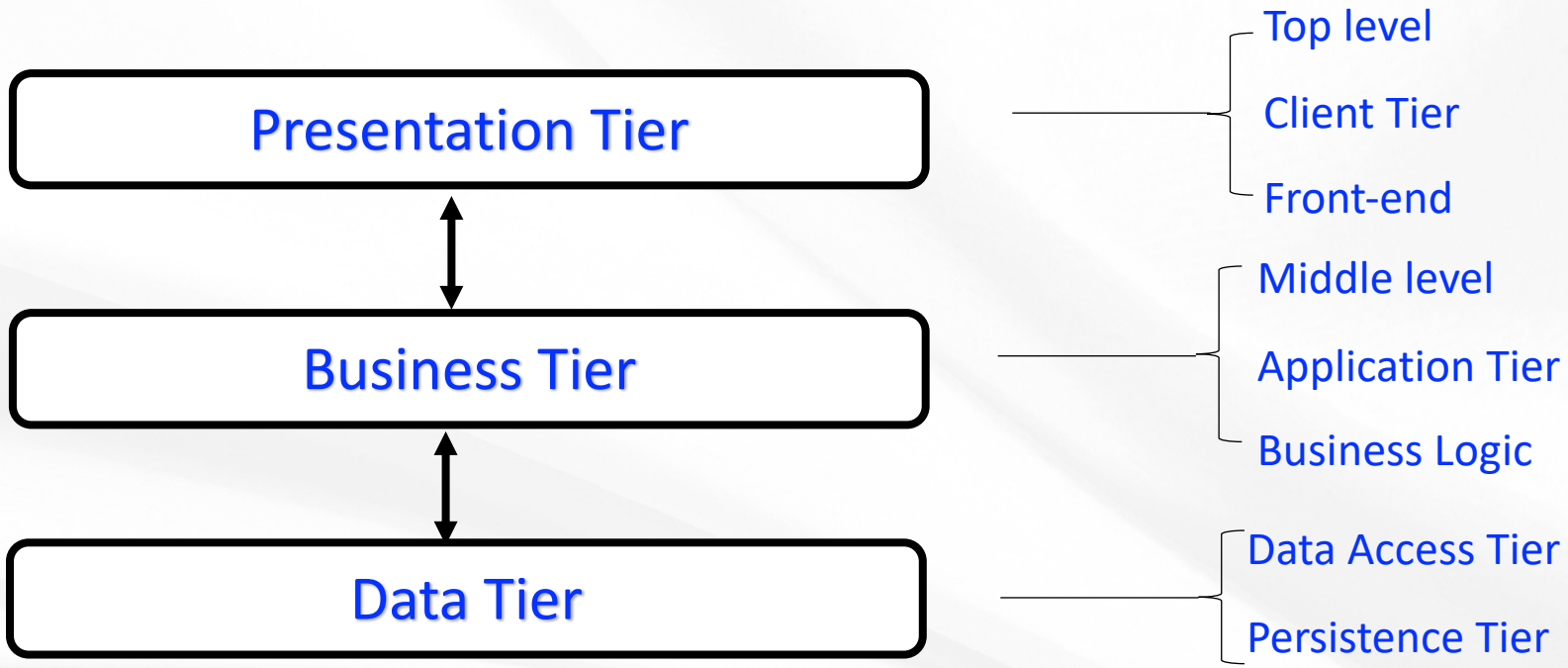
- Architecture styles
  - ✓ Monolithic architectures
  - ✓ Distributed architectures

3–Tier architecture style

- **Three Tier architecture style**
  - **Three Tier** architecture is one of the most common architectural styles.
  - Three–tier architecture is a well–established software application architecture that organizes applications into three logical and physical computing tiers: the presentation tier, or user interface; the application tier, where data is processed; and the data tier, where the data associated with the application is stored and managed.

3-Tier architecture style

- The main benefit of three-tier architecture is that because each tier runs on its own infrastructure, each tier can be developed simultaneously by a separate development team, and can be updated or scaled as needed without impacting the other tiers.



3-Tier architecture style

1. Presentation tier

   - The presentation tier is the user interface and communication layer of the application, where the end user interacts with the application. Its main purpose is to display information to and collect information from the user.
   - This top-level tier can run on a web browser, as desktop application, or a graphical user interface (GUI), for example. Web presentation tiers are usually developed using HTML, CSS and JavaScript. Desktop applications can be written in a variety of languages depending on the platform.

3-Tier architecture style

2. Business tier

- The business tier, also known as the logic tier or middle tier, is the heart of the application. In this tier, information collected in the presentation tier is processed – sometimes against other information in the data tier – using business logic, a specific set of business rules. The application tier can also add, delete or modify data in the data tier.
- The business tier is typically developed using Python, Java, Perl, PHP or Ruby, and communicates with the data tier using API calls.

3-Tier architecture style

3. Data Tier

- The data tier, sometimes called database tier, data access tier or back-end, is where the information processed by the application is stored and managed. This can be a relational database management system such as PostgreSQL, MySQL, MariaDB, Oracle, DB2, Informix or Microsoft SQL Server, or in a NoSQL Database server such as Cassandra, CouchDB or MongoDB.
- In a three-tier application, all communication goes through the application tier. The presentation tier and the data tier cannot communicate directly with one another.

Advantages of Three–Tier Architecture

- **Faster development:** Because each tier can be developed simultaneously by different teams, an organization can bring the application to market faster, and programmers can use the latest and best languages and tools for each tier.
- **Scalability:** Each tier can be scaled independently to handle increased load or resource requirements.

Advantages of Three–Tier Architecture

- **Flexibility**: Changes in one tier do not necessarily affect the others, providing flexibility in development and maintenance.
- **Security**: The separation of layers enhances security by restricting direct access to the data tier and enforcing access controls through the application tier.

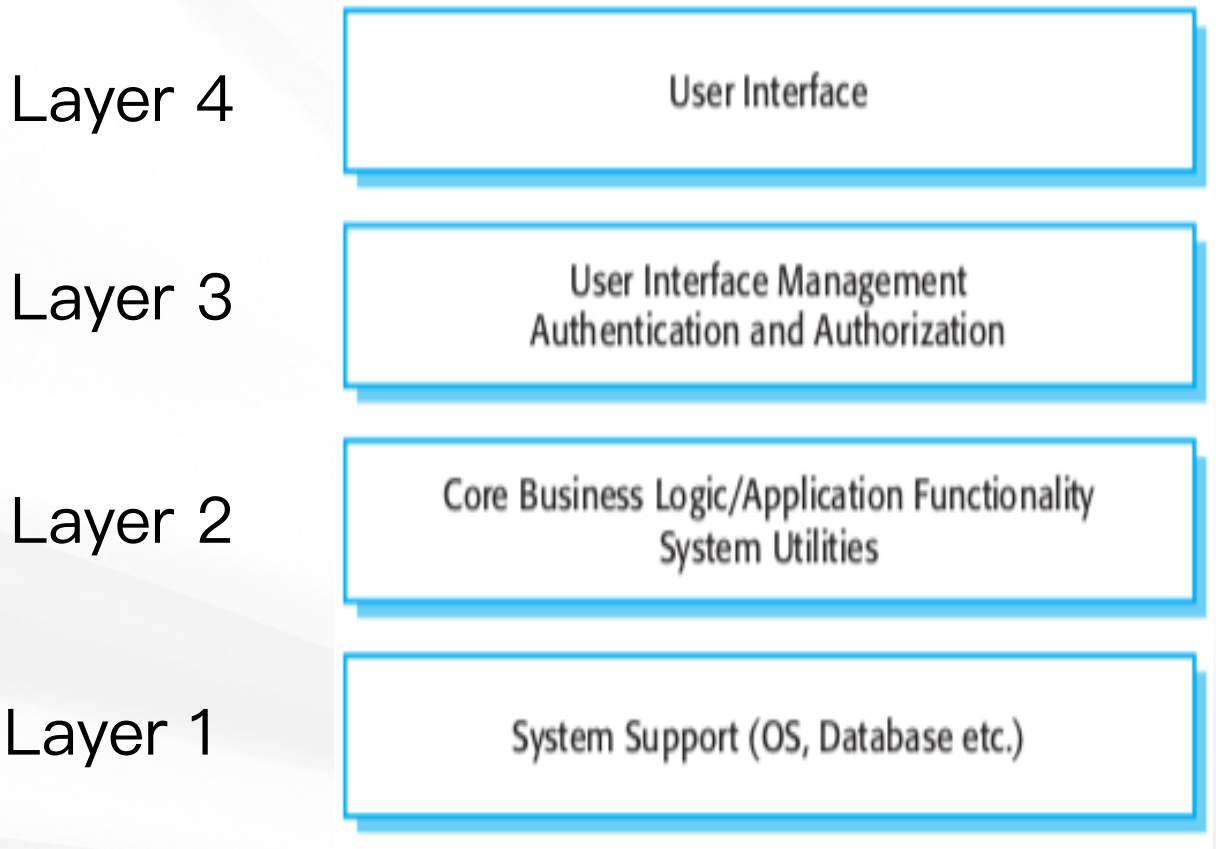
Layered architecture style

- **Layered architecture style**
  - Layered architecture known as the n–tiered architecture style, is one of the most common architectural styles.
  - Organizes the system into layers with related functionality associated with each layer. A layer provides services to the layer above it so the lowest–level layers represent core services that are likely to be used throughout the system.

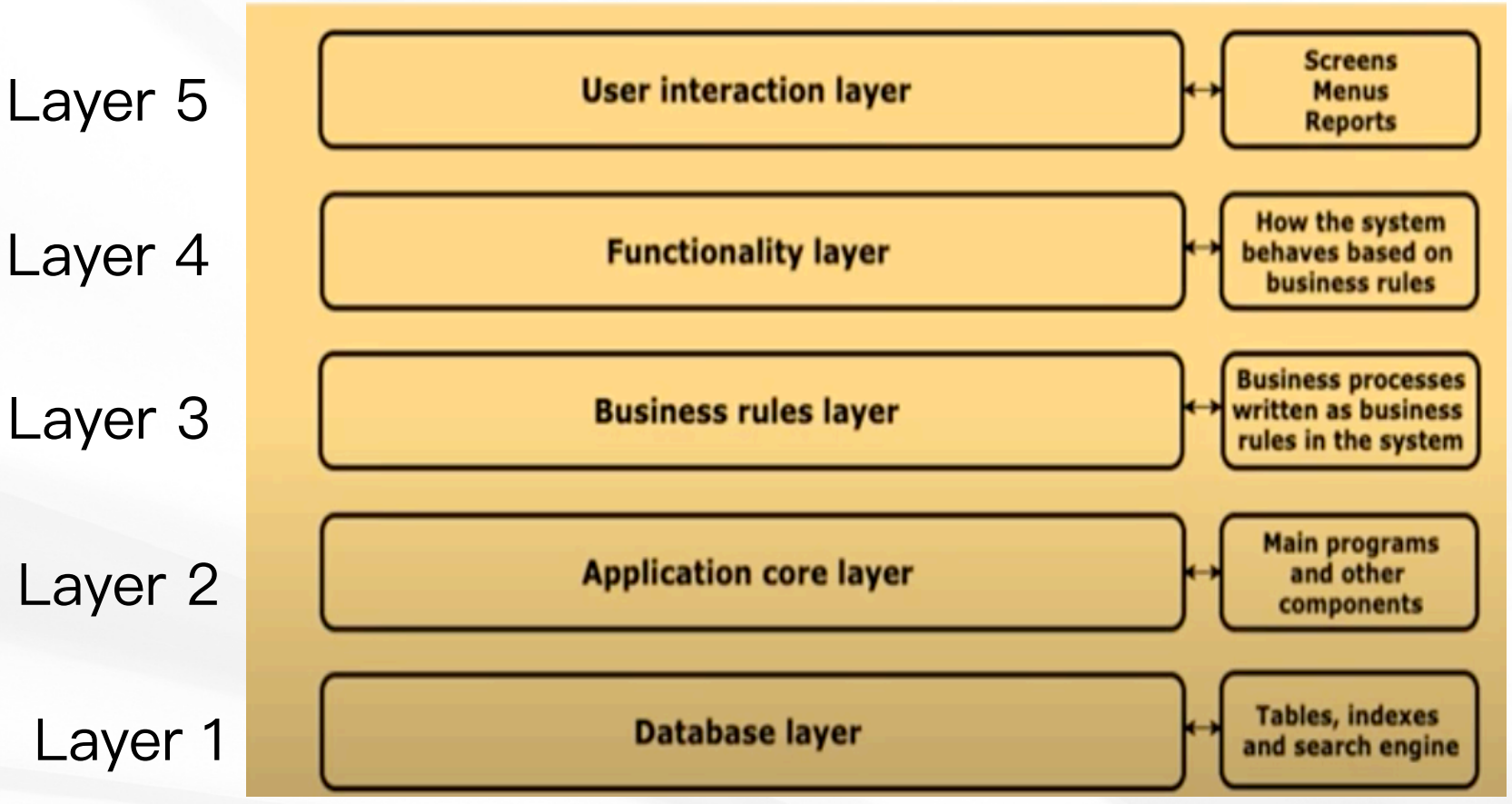
Layered Architecture

- The idea behind Layered Architecture is that modules or components with similar functionalities are organized into horizontal layers.
- Used when
  - ✓ building new facilities on top of existing systems.
  - ✓ when the development is spread across several teams with each team responsibility for a layer of functionality.
  - ✓ when there is a requirement for multi–level security.
  - ✓ Only for big applications.

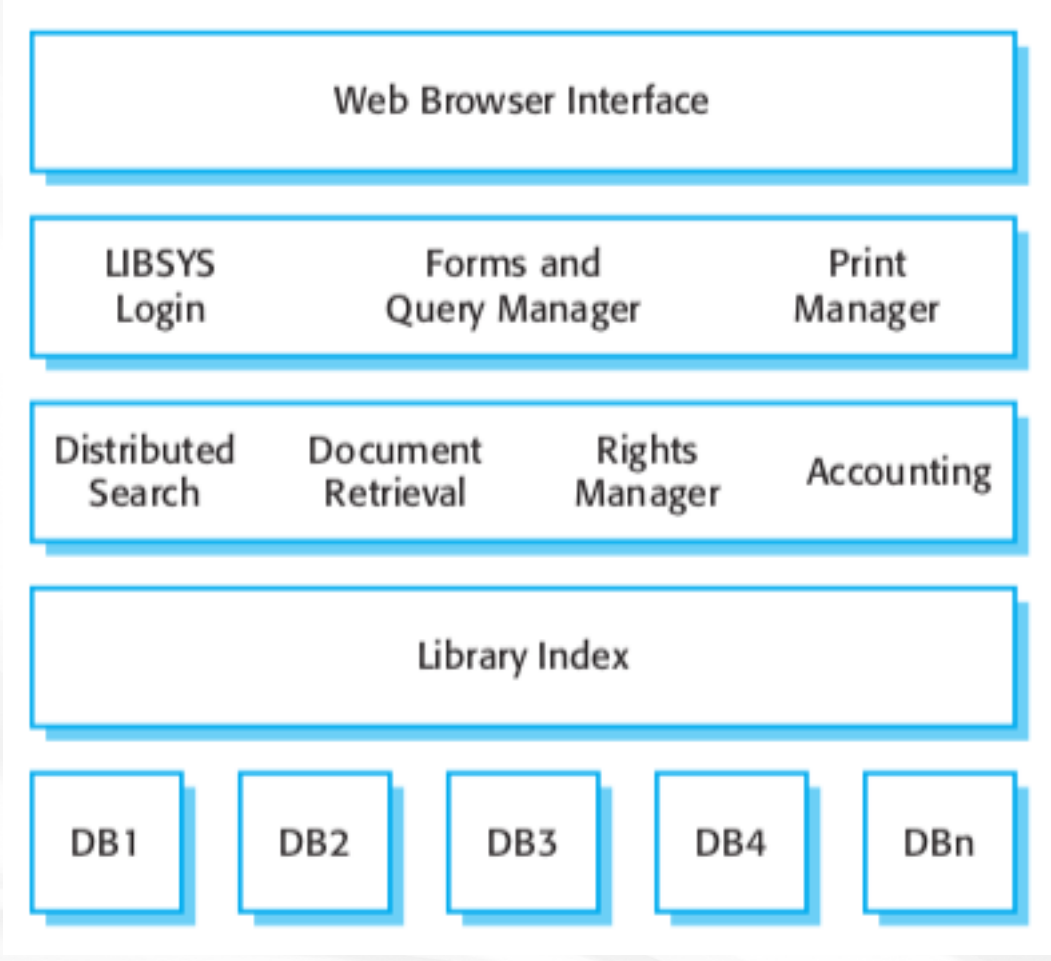
A generic layered architecture



A generic layered architecture



Example: The architecture of the LIBSYS system



Layered Architecture

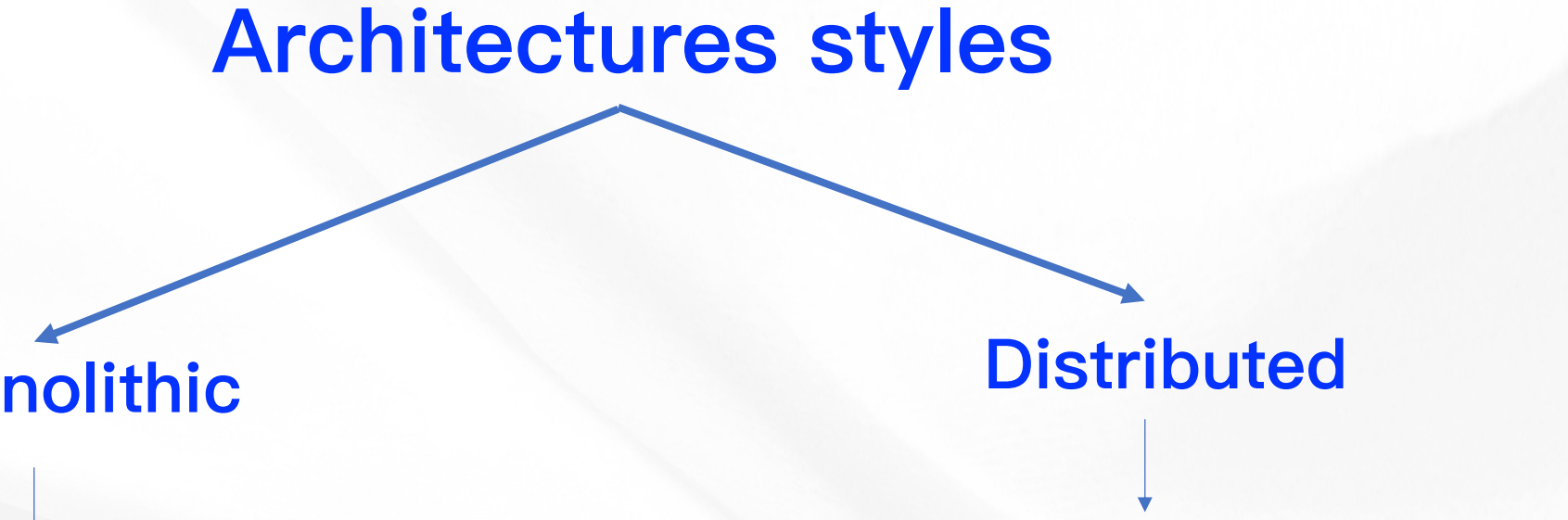
Advantages

- Allows replacement of entire layers so long as the interface is maintained.
- Redundant facilities (e.g., authentication) can be provided in each layer to increase the dependability of the system.

Disadvantages

- In practice, providing a clean separation between layers is often difficult and a high–level layer may have to interact directly with lower–level layers rather than through the layer immediately below it.
- Performance can be a problem because of multiple levels of interpretation of a service request as it is processed at each layer.

Architectures styles

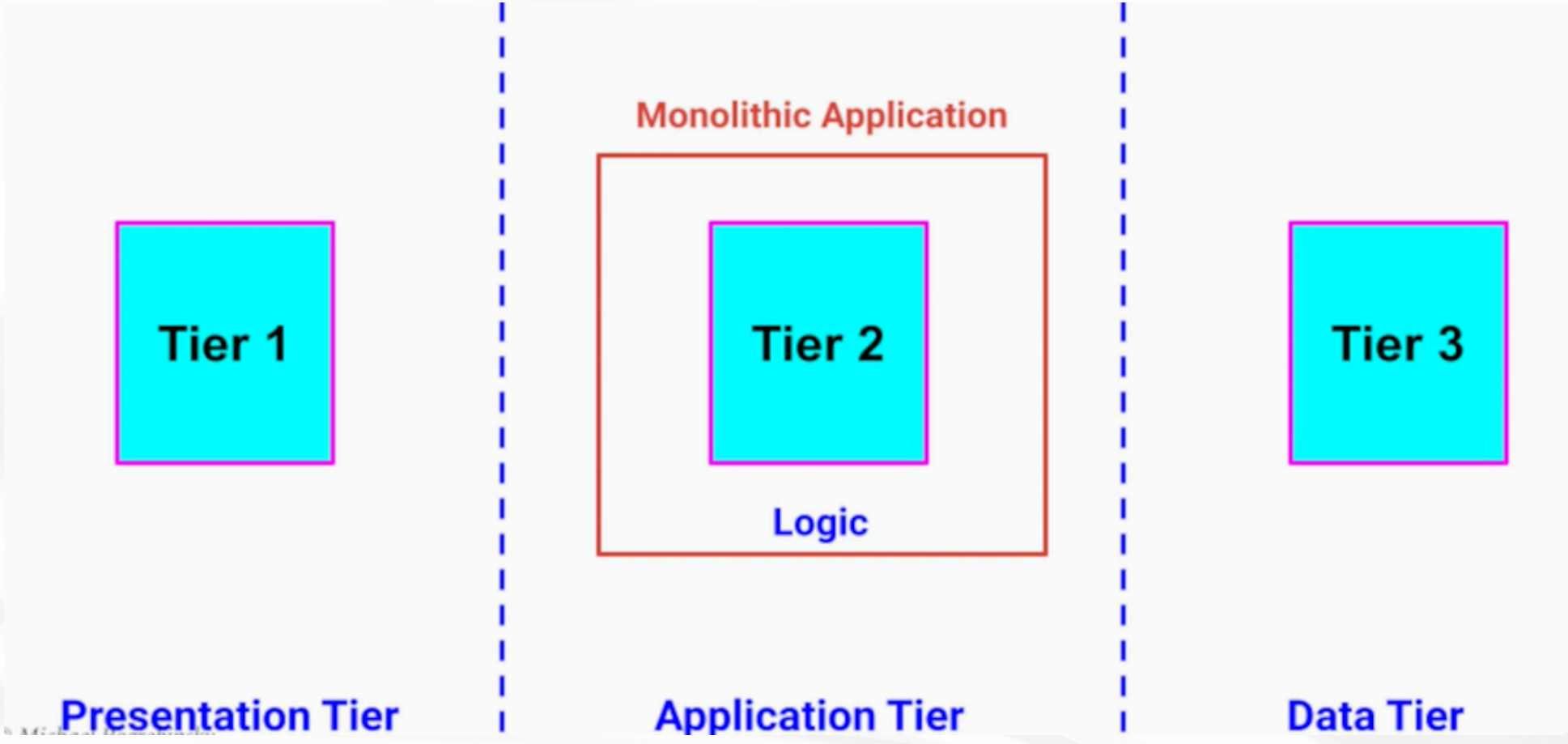


Monolithic architecture

- Monolithic architecture, in contrast to microservices architecture,
- It is an approach where an entire software application is built as a single, tightly integrated unit.
- Tight Integration: all the components and modules of the application are interconnected.
- Single Codebase: The entire application, including its functionalities and features, is developed within a single codebase.

Microservices architecture VS Monolithic architecture (3 Tier architecture)

All the business logic was concentrated in one single service in the application tier



Monolithic architecture

Monolithic Architecture – Advantages

Monolithic architecture is perfect for:

- Small teams
- Small and non complex codebase

Monolithic Architecture – Disadvantages

As the size and complexity of our codebase grow, it becomes difficult to:

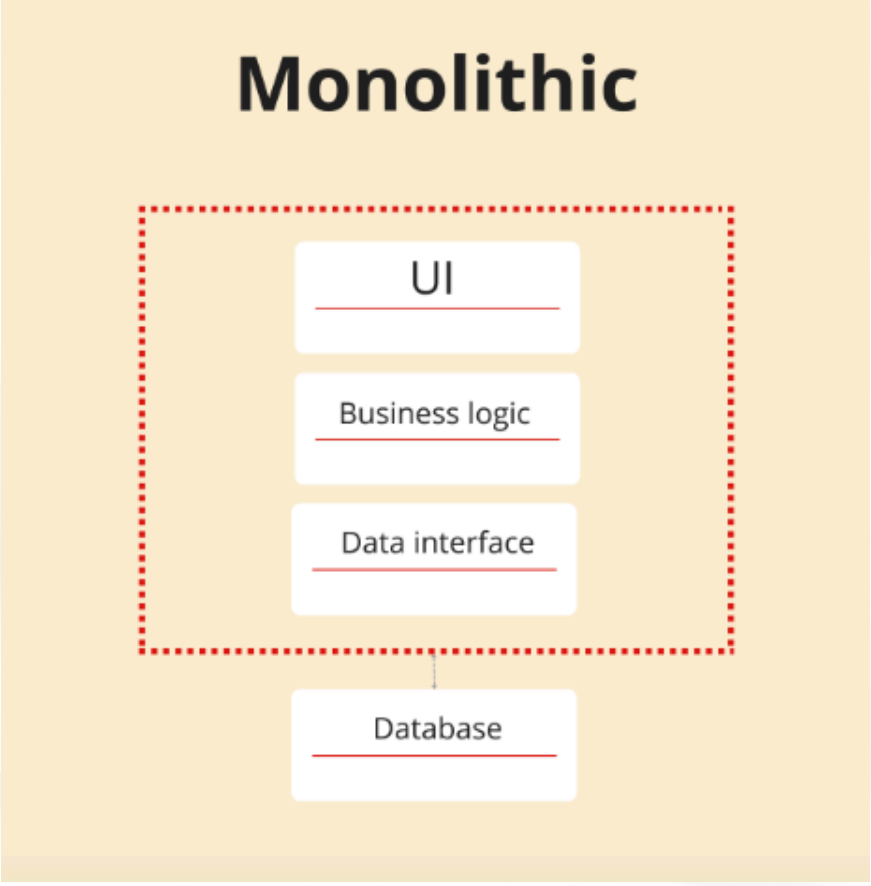
- Troubleshoot
- Add new features
- Build &Test
- Load in IDE

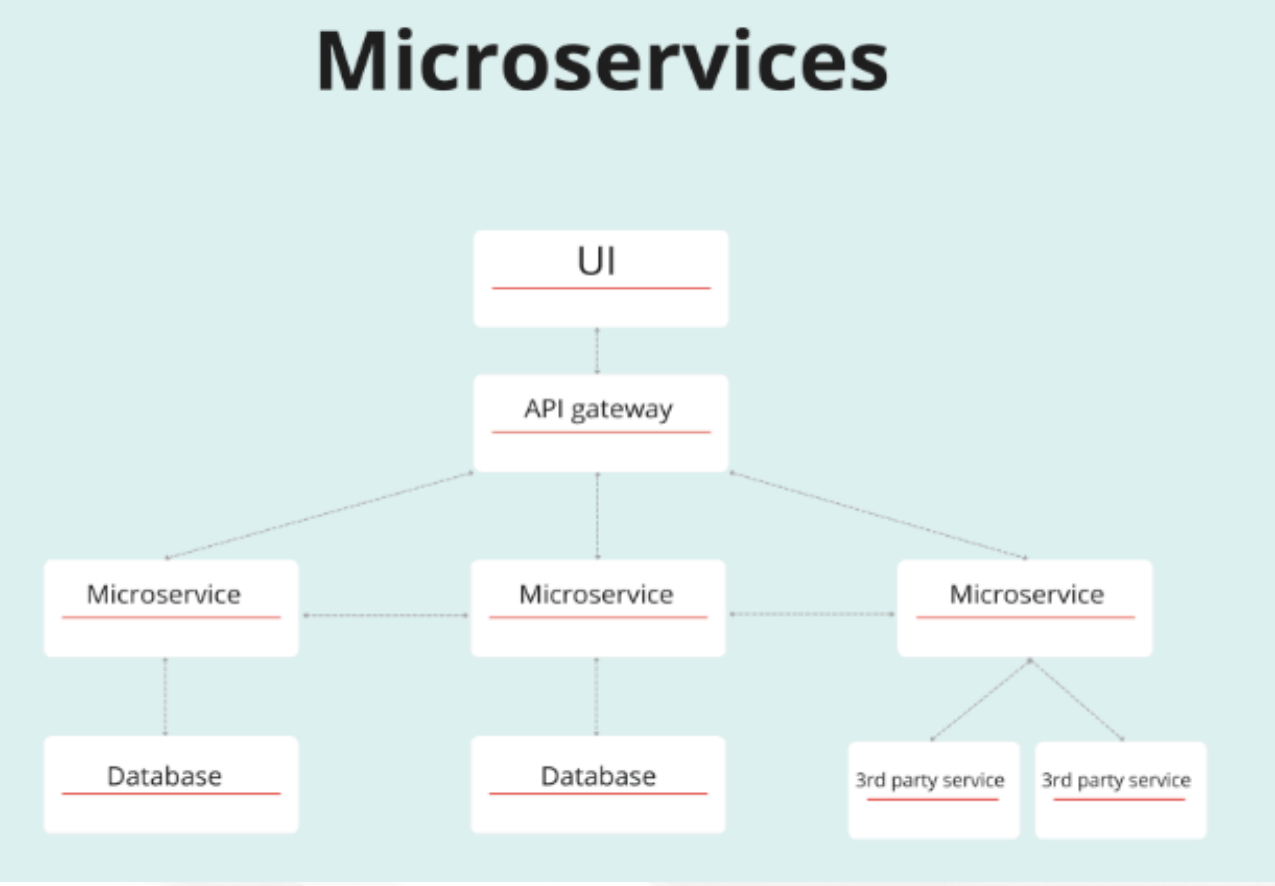
Monolithic Architecture – Disadvantages

We have problems in organizational scalability because:

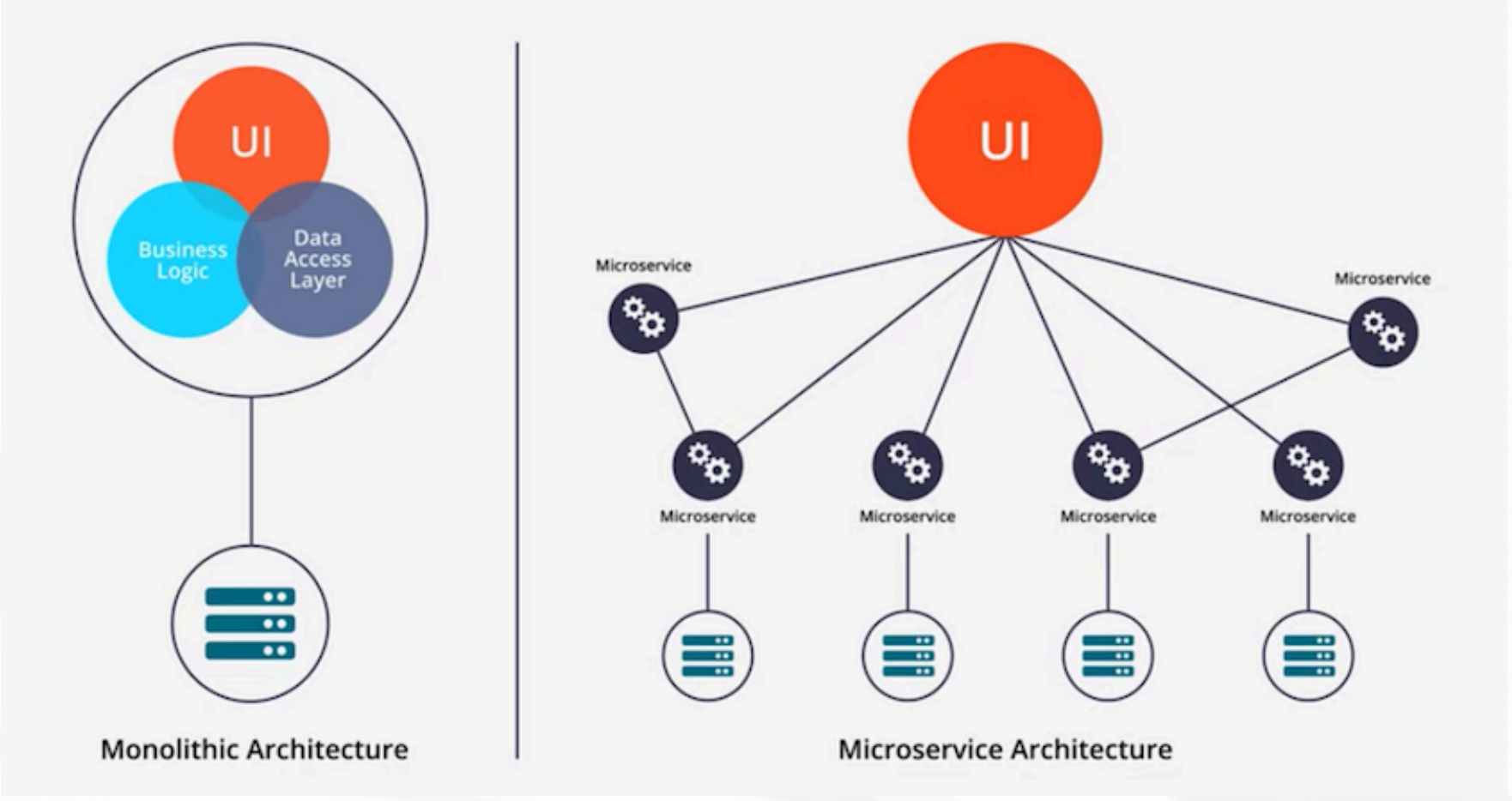
- The more engineers we add to the team, the more code merge conflicts we get
- Our meetings become larger, longer, and less productive
- Once we start seeing these problems, we should consider migrating our architecture towards Microservices

Microservices architecture VS Monolithic architecture (3 Tier architecture)





Microservices architecture VS Monolithic architecture (3 Tier architecture)



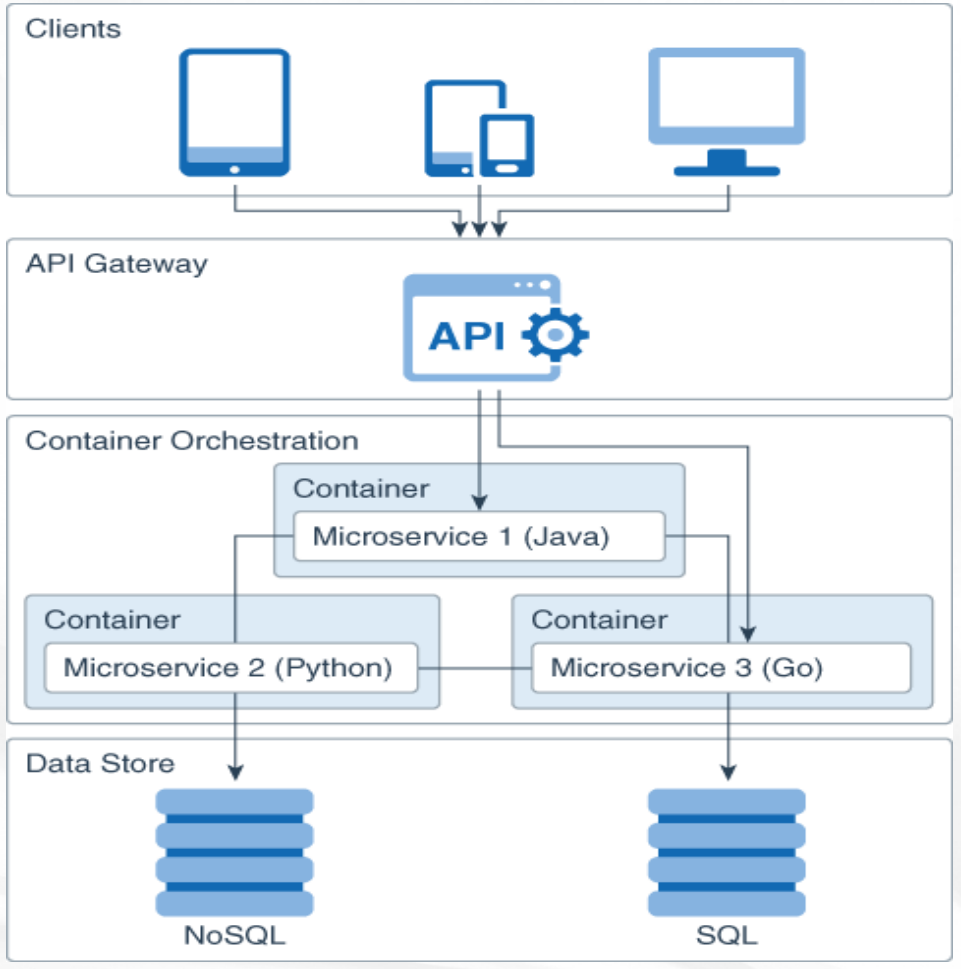
Microservice architecture

- Microservices Architecture organizes our business logic as a collection of loosely coupled and independently deployed services
- Each service is owned by a small team and has a narrow scope of responsibility.
- Microservices architecture is an industry-proven method of building applications out of separate distributed modules (microservices) each focused on one business function or service.

Microservice architecture

- These modules use APIs (often RESTful) and API gateways to connect with other services, client apps, and other applications in the organization.
- The API–driven approach enables loose connections and weak dependence between components.
- Each module in a microservices application has its own business logic, database.

Microservice architecture



Why Microservices?

1– Down–Time

If Microservice–A is down that does not mean Microservice–B will be down also.

2 – Deployment

The deployment will be easier and easier compare with the Monolithic structure.

3 – Using more than one programming language

You can use PHP for Microservice–A on the other you can use Java for Microservice–B.

Why Microservices?

4– Easy to test

When you use Microservice that is mean small functionality you will be testing.

5– Load Balancing

It's more and more important we can benefit when we use a Microservices style structure in our system. Balanced distribution traffic on our services.

Microservice architecture – Advantages

1. Smaller Codebase
2. Better Performance and Horizontal Scalability
3. Better Organizational Scalability
4. Better security (Fault Isolation)

Microservice architecture – Advantages

Smaller Codebase:

Benefits of Smaller Codebase are:

- Development becomes easier and faster
- Codebase loads instantaneously in our IDE
- Building and testing becomes easier and faster
- Troubleshooting/adding new features becomes easier
- New developers can become fully productive faster

Microservice architecture – Advantages

Performance and Scalability – Benefits

- Instances become less CPU intensive and less memory-consuming
- Services can be scaled horizontally by adding more instances of low-end computers

Fault Isolation – Benefits

- If we have an issue in one of the services, it is easier to isolate it and mitigate the problem

Microservice architecture – Advantages

Organizational Scalability – Benefits

- Each service can be independently
- Developed
- Maintained
- Deployed
- by a separate small team
- Leads to high throughput from the entire organization

Event–driven Architecture

Event–driven architecture is a style that organizes a system as a set of loosely coupled components that communicate and coordinate through events. These events can be anything from a user action, a sensor reading, a database update, or an external trigger.

Key Components of EDA:

1. **Event:** An event represents a meaningful change or occurrence within a system. It is a lightweight, immutable.
2. It is an immutable statement of fact or change.

Event–driven Architecture

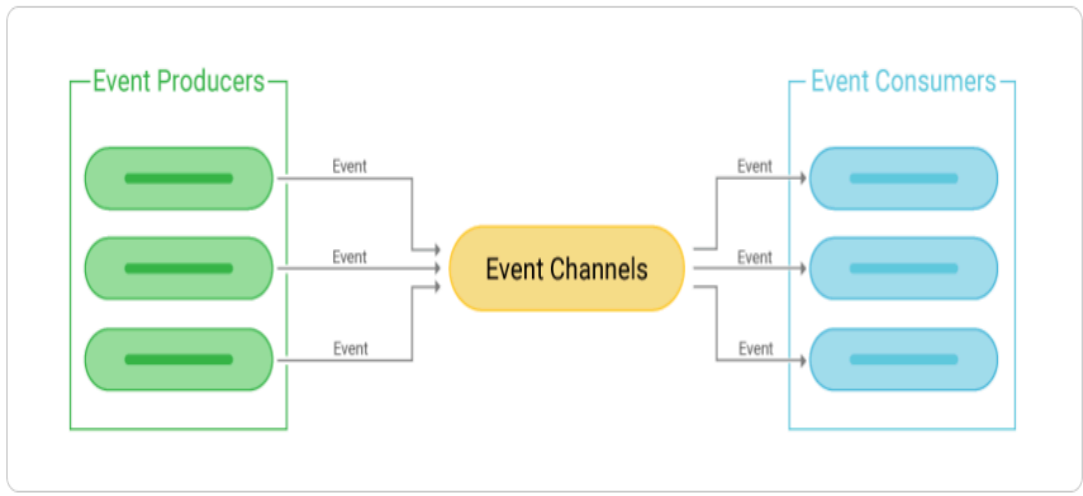
Events – Examples

- Fact Events
- User clicking on a digital ad
- Item being added to a shopping cart
- Change Events
- Player of a video game
- IoT device (vacuum cleaner)

Event–driven Architecture

Types of Event:

- Temporal Events
- State Events
- Invocation Events
- Notification Events
- Error Events



2. Event Producer(Publisher): This component is responsible for generating and emitting events. It could be a user interface, a sensor, or any other source that initiates an event.

Event-driven Architecture

3. Event Consumer (Subscriber): The event consumer is the component that listens for and reacts to events. It could be a service, a function, or any module designed to respond to specific events.

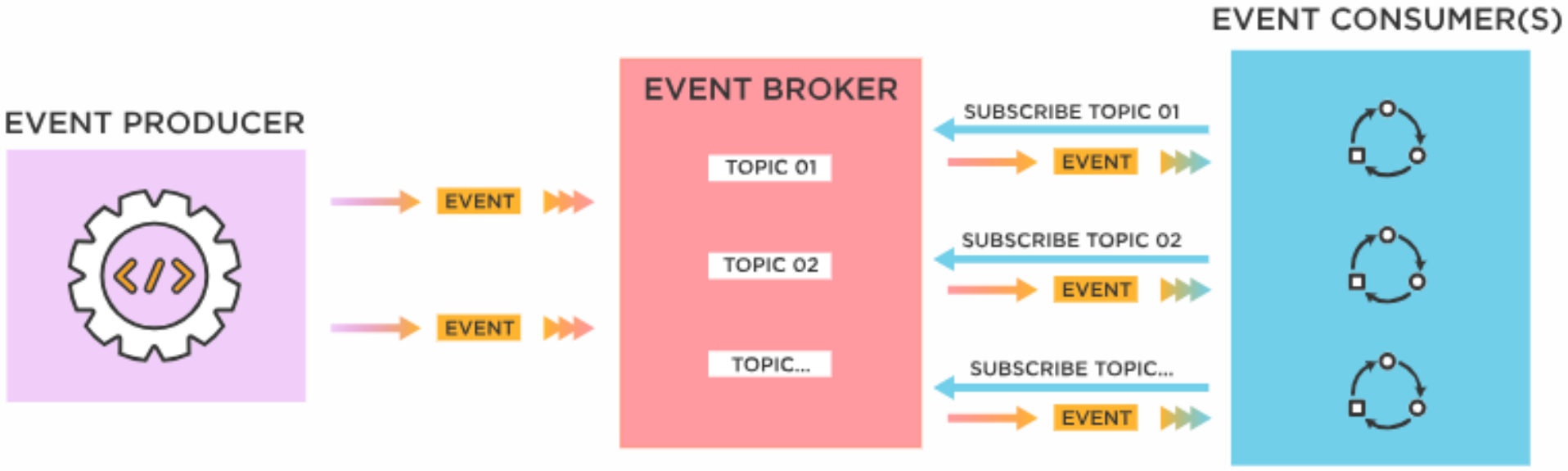
4. Event Broker(Message Broker): The event broker acts as an intermediary, ensuring that events are routed from producers to consumers efficiently. It can be implemented using various messaging systems such as Apache Kafka, RabbitMQ, or AWS Lambda.

5. Event Router: In more complex event-driven systems, you can have an event router that directs events to their respective event consumers based on predefined rules or routing logic. This component helps manage the flow of events within the system

Event–driven Architecture

TIP:

Ensure your event model in an event–driven system contains sufficient and clear information about the event that occurred. This enables consumers to quickly respond to events without the need to request additional data.



Event-driven Architecture

Event-driven architecture is commonly used in various applications, including:

- Real-time data processing
- Microservices-based systems
- IoT (Internet of Things) applications
- Systems that require high levels of concurrency and responsiveness

Event–driven Architecture– Examples

1. E–commerce: EDA can power real–time inventory updates, order processing, and personalized recommendations, enhancing the shopping experience.
2. IoT (Internet of Things): EDA is ideal for IoT applications, enabling the handling of vast streams of sensor data and immediate responses to events like equipment malfunctions or environmental changes.
3. Financial Services: In the world of finance, EDA can be used for real–time fraud detection, trade execution, and market data analysis.
4. Social Media: Social networks use EDA for real–time updates, notifications, and content distribution. Such s Twitter

Benefits of Event-Driven Architecture

- Loose Coupling and Flexibility: EDA promotes loose coupling between components, allowing them to operate independently. Components can evolve, scale, or be replaced without affecting the entire system, making the architecture more flexible and adaptable to changes.

- Scalability and Performance: EDA enables horizontal scalability by distributing the workload across multiple components that can handle events concurrently. This scalability allows systems to handle high volumes of events and provide better performance under varying loads.

Benefits of Event-Driven Architecture

- Real-time Responsiveness: By leveraging events, EDA enables real-time responsiveness to changes and triggers actions immediately. Components can react to events as they occur, enabling quick decision-making and timely responses to business needs.
- Event Sourcing: Events capture the sequence of actions and state changes in the system, facilitating event sourcing. Event logs provide a historical record of system behavior, enabling audibility, debugging, and replaying events for analysis or testing.

Software Architecture Patterns

Summary:

- Model view controller architecture (MVC)
- Client server architecture
- There Tier architecture
- Layered architecture
- Microservices architecture
- Event driven architecture

Software design patterns

- Software design patterns are general reusable solutions to common problems that occur in software design. They represent best practices for solving certain types of problems and provide a way to structure code to enhance its clarity, flexibility, and maintainability.
- Design patterns are often categorized into creational, structural, and behavioral patterns.

Software design patterns

Design patterns Types:

- Creational Patterns: These patterns deal with the process of object creation. Examples include Singleton, Factory Method, Abstract Factory, etc.

- Structural Patterns: These patterns focus on the composition of classes or objects. Examples include Adapter, Decorator, Bridge, etc.

- Behavioral Patterns: These patterns define the ways in which objects interact and communicate. Examples include Observer, Strategy, Command, etc

Software design patterns

Creational Patterns	Structural Patterns	Behavioral Patterns
▪ Factory Method ▪ Singleton ▪ Builder ▪ Prototype ▪ Abstract Factory	▪ Adapter ▪ Decorator ▪ Bridge ▪ Composite ▪ Proxy	▪ Observer ▪ Strategy ▪ Command ▪ Iterator

The End