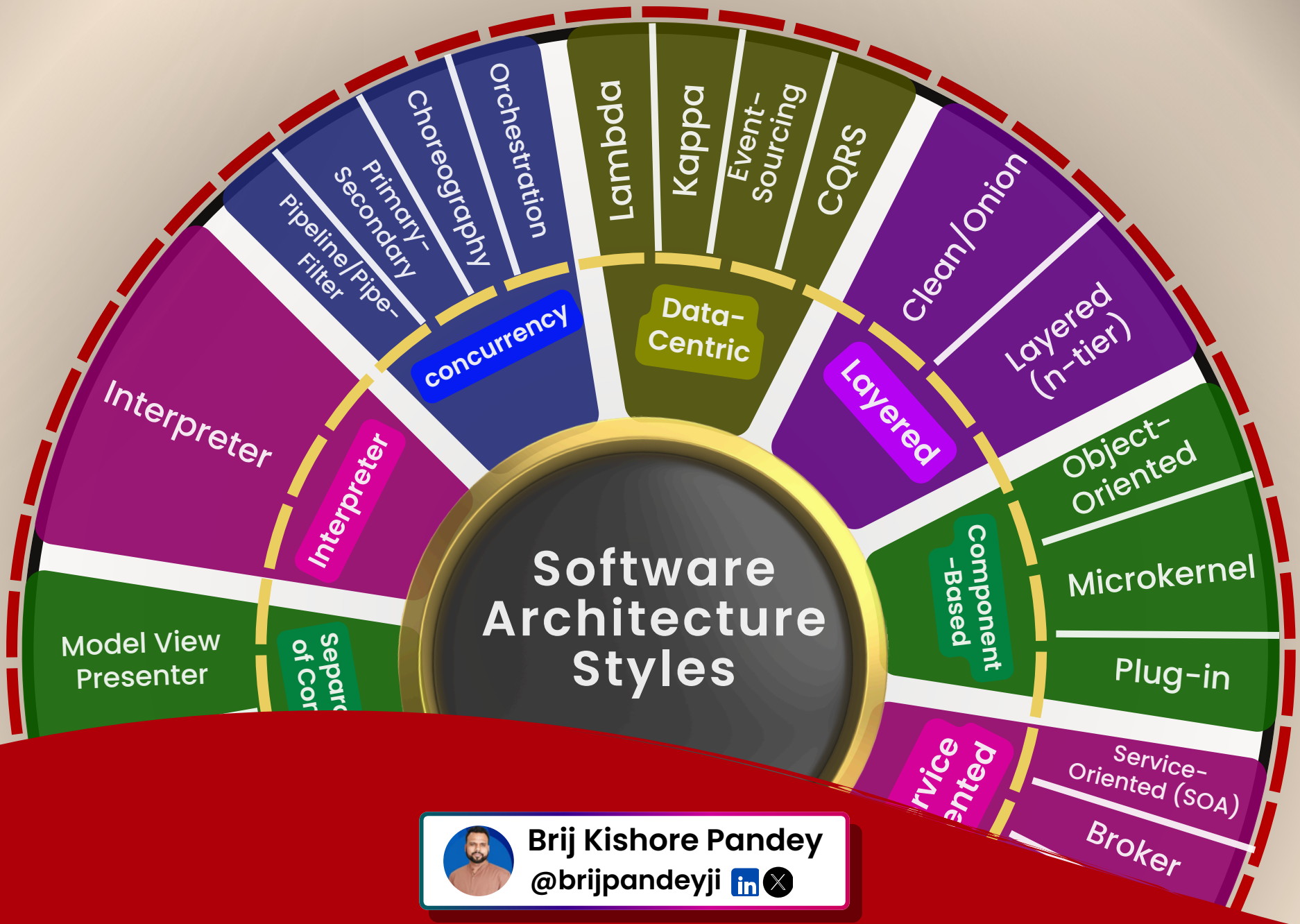


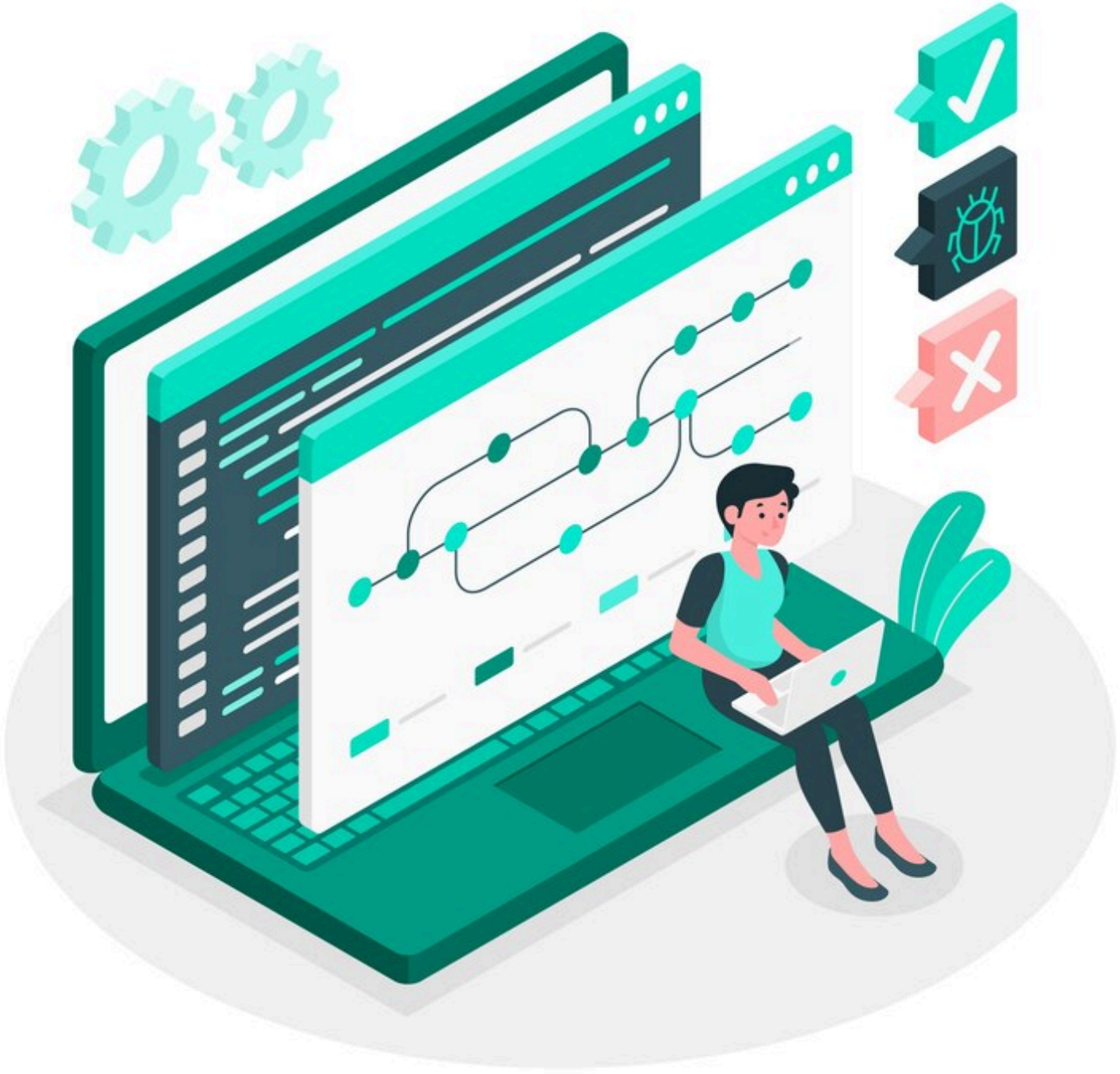
Brij Kishore Pandey
@brijpandeyji

SOFTWARE ARHITECTURE STYLES

Swipe

SOFTWARE ARCHITECTURE STYLES

A deep dive into key software architecture styles and how they shape the structure and design of systems.

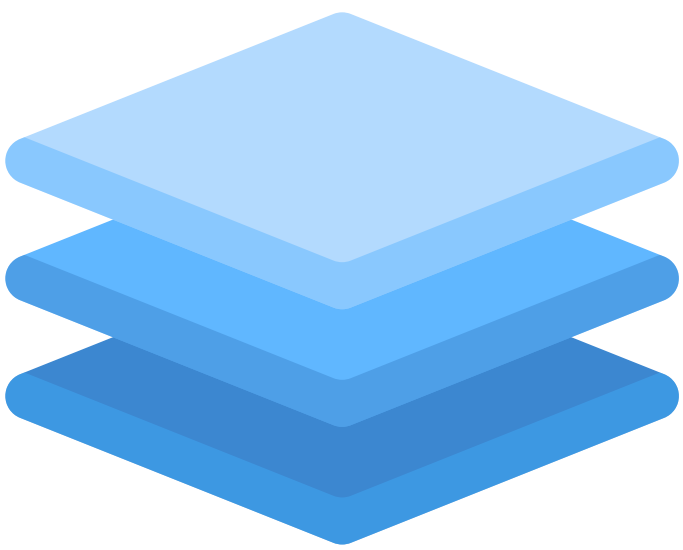


Swipe

LAYERED (N-TIER) ARCHITECTURE

What it is:

Software architecture styles provide structured patterns for organizing and designing a system. These styles define how software components interact, communicate, and work together to create a functional and scalable system.



Why it's used:

This style is ideal for applications that require a clear separation of concerns, making it easier to manage, test, and maintain. It’s commonly used in enterprise applications, where each layer can be updated without affecting the others.

Swipe

MICROSERVICES ARCHITECTURE

What it is:

Microservices architecture breaks down an application into small, independent services that perform a specific business function. These services communicate over APIs and can be developed, deployed, and scaled independently.

Why it's used:

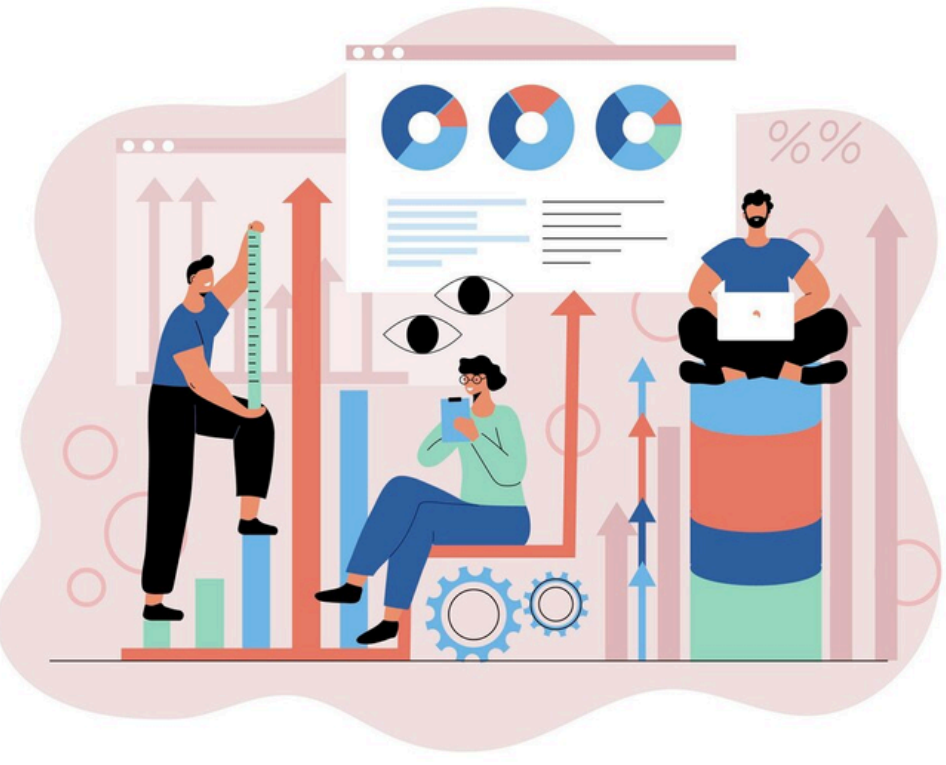
This style is great for complex and large-scale systems that need to be flexible and scalable. Microservices allow teams to work on different services simultaneously, speeding up development and deployment cycles. It also makes systems more resilient since issues in one service don’t usually affect the whole system.

Swipe

EVENT-DRIVEN ARCHITECTURE

What it is:

In event-driven architecture, components (services or modules) communicate by producing and responding to events. An event is a significant change in system state, and different components react to these events asynchronously.



Why it's used:

Event-driven architecture is highly scalable and flexible, making it perfect for systems that require real-time processing or are built to handle high volumes of incoming data. It also enables decoupled communication between services, meaning they don’t need to know about each other to communicate effectively.

Swipe

CLIENT-SERVER ARCHITECTURE

What it is:

Client-server architecture is a distributed system framework where a server provides resources or services, and a client accesses those services over a network. The client sends requests to the server, which then processes them and returns the requested data or service.

Why it's used:

This architecture is commonly used in web and mobile applications, where the server handles tasks like data storage or complex processing, and the client (usually the user interface) presents the data to the user.

Swipe

MONOLITHIC ARCHITECTURE

What it is:

A monolithic architecture is built as a single, unified unit where all the components are interconnected and tightly coupled. All functions of the system, such as the user interface, server-side processing, and database management, reside in one large codebase.

Why it's used:

Monolithic architecture is easier to develop and deploy initially, making it a good fit for smaller applications or startups. However, as the system grows, maintaining and scaling a monolithic application can become challenging.

Swipe

SERVICE-ORIENTED ARCHITECTURE (SOA)

What it is:

SOA is a design pattern where software components provide services to other components over a network. Each service is designed to be reusable, independent, and self-contained, communicating via protocols like HTTP or messaging queues.

Why it's used:

SOA promotes flexibility and reusability, making it ideal for integrating disparate systems or for applications that need to interact with other services within an organization. It is commonly used in large organizations with various departments needing shared services.

Swipe

PEER-TO-PEER (P2P) ARCHITECTURE

What it is:

In peer-to-peer architecture, each node (computer) in the system can act both as a client and a server. Instead of relying on a central server, each node can share resources or information directly with other nodes.

Why it's used:

P2P architecture is used for decentralized systems like file-sharing networks (e.g., BitTorrent), where each peer can contribute to and consume resources. It’s scalable and resilient, as the system doesn’t rely on a single point of failure.

Swipe

MICROKERNEL ARCHITECTURE

What it is:

Microkernel architecture structures an application around a core system (the microkernel) that handles the most essential processes, while additional features or services are plugged in as modules.

Why it's used:

This architecture is perfect for systems that require extensibility and flexibility, such as operating systems or systems with plugins, where core functionality remains stable while allowing easy addition of new features.

Swipe

PIPE-AND-FILTER ARCHITECTURE

What it is:

In pipe-and-filter architecture, data flows through a series of processing units (filters) connected by pipes. Each filter processes the data and passes it along to the next, transforming it step by step.

Why it's used:

This architecture is often used in data processing applications like compilers or stream processing systems, where data undergoes multiple stages of transformation.

Swipe

SPACE-BASED ARCHITECTURE

What it is:

Space-based architecture focuses on scalability by distributing both the processing and data across multiple spaces or nodes. It often uses an in-memory data grid to handle high loads by dynamically adding or removing nodes.

Why it's used:

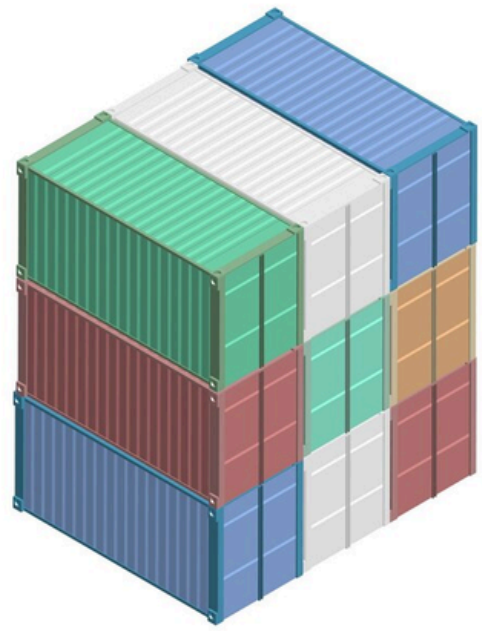
It’s particularly useful for applications that experience high and unpredictable traffic, as it enables dynamic scaling. Space-based architecture is common in e-commerce or social media platforms where sudden traffic spikes are expected.

Swipe

CLOUD-NATIVE ARCHITECTURE

What it is:

Cloud-native architecture is designed to take full advantage of cloud environments by leveraging microservices, containers, and serverless computing. It allows systems to be highly scalable, resilient, and flexible.



Why it's used:

Cloud-native architecture is perfect for modern applications that need to be scalable on demand, making it ideal for businesses moving to the cloud for flexibility and cost-effectiveness.

Swipe

CONCLUSION

Understanding different software architecture styles is crucial for designing systems that are scalable, maintainable, and flexible. Each style offers unique advantages and is suited for specific types of applications based on their complexity and requirements.

Swipe

Brij Kishore Pandey

@brijpandeyji

Follow for More