

The background features a large white circle in the center, partially overlapping a light blue rectangle on the left and a light pink rectangle on the right. A dark blue shape, resembling a stylized arch or a large letter 'C', is positioned at the bottom, framing the white circle.

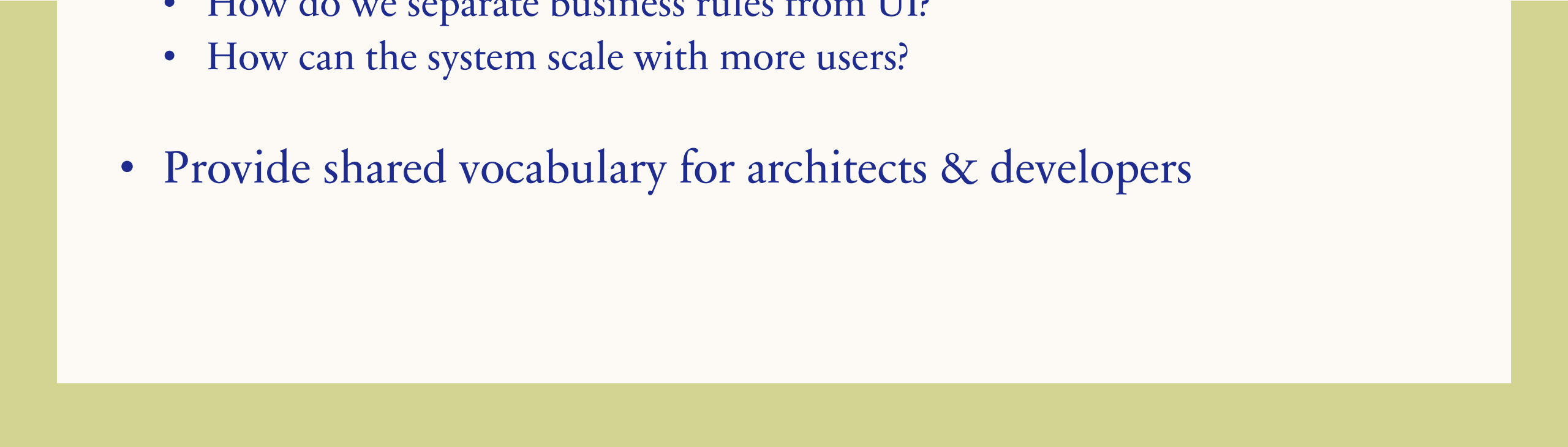
ARCHITECTURE PATTERNS

ARCHITECTURE PATTERNS

- Building software is NOT just writing code, it is to design it so that it LASTs, GROWs, and ADAPTs.
- Building needs blueprints, software needs an architecture.
- Don't start from scratch (reinvent the wheel), use PROVEN solutions.
- Architecture Patterns – templates or best practices for structuring a system.

WHAT DO PATTERNS HELP WITH?



- Answer fundamental design questions:
 - Where should logic live (client or server)?
 - How do we separate business rules from UI?
 - How can the system scale with more users?
 - Provide shared vocabulary for architects & developers
- 

CLIENT – SERVER PATTERN

- Concept: Splits system into clients (request services) and servers (provide services).
- Focus/Dimension: Deployment & Distribution (who provides services, who consumes them). Where should processing and data be located—on client or server?
- Advantages:
 - Centralized control and data management.
 - Easier maintenance and upgrades on server side.
- Disadvantages:
 - Server can become a bottleneck.
 - Network dependency affects performance.
- Examples: Web browsers (clients) communicating with web servers.

LAYERED (N-TIER) PATTERN

- Concept: Organizes system into layers, each with specific responsibilities.
- Focus/Dimension: Separation of Concerns. How can we organize code to make it maintainable and testable?
- Common layers: Presentation, Application, Business Logic, Data Access.
- Advantages:
 - Separation of concerns → easier testing and maintenance.
 - Allows independent development of layers.
- Disadvantages:
 - Can introduce overhead in communication between layers.
 - Not suitable for performance-critical systems.
- Examples: Enterprise applications, banking systems.

MODEL VIEW CONTROLLER PATTERN

6

- Concept: Separates application into:
 - Model: Data and business rules.
 - View: UI elements.
 - Controller: Handles input and updates model/view.
- Focus/Dimension: User Interface + Business Logic Separation. How do we keep UI changes independent from business logic?
- Advantages:
 - Clear separation of responsibilities.
 - Easy to modify UI without changing business logic.
- Disadvantages:
 - Increased complexity for small applications.
- Examples: Web frameworks like Django, Spring MVC, ASP.NET MVC.

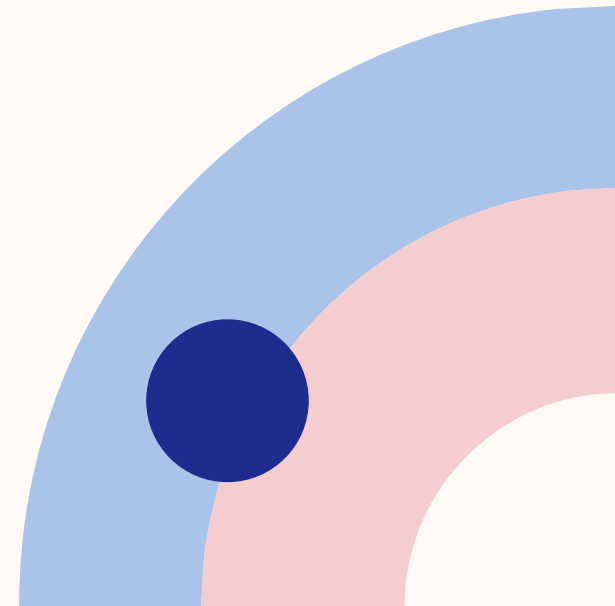
MICROSERVICES PATTERN

7

- Concept: Breaks system into small, independent services communicating via APIs.
- Focus/Dimension: Scalability and Independent Deployment. How do we scale and evolve different parts of a large system independently?
- Advantages:
 - Scalability (services scale independently).
 - Technology flexibility (each service can use different tech).
 - Fault isolation → failures don't bring down the whole system.
- Disadvantages:
 - Requires complex deployment and monitoring.
 - Network latency and distributed data challenges.
- Examples: Netflix, Amazon, Uber architectures.

EVENT DRIVEN PATTERN

- Concept: Components communicate via events (publish/subscribe).
- Focus/Dimension: Asynchronous Communication & Reactivity. How can parts of a system react to changes without tight coupling?
- Advantages:
 - Loose coupling between components.
 - Highly scalable and reactive.
- Disadvantages:
 - Debugging and tracing event flow can be difficult.
 - Requires robust event-handling infrastructure.
- Examples: Real-time analytics, IoT systems.



PIPE AND FILTER PATTERN

9

- Concept: Data flows through a series of processing steps (filters) connected by pipes.
- Focus/Dimension: Data Processing Flow. How do we structure sequential data transformations?
- Advantages:
 - Reusability of filters.
 - Easy to add/remove steps.
- Disadvantages:
 - High overhead with large data transformations.
- Examples: Compilers, data processing pipelines.



PEER TO PEER PATTERN

10

- Concept: Every node acts as both a client and a server.
- Focus/Dimension: Decentralization & Collaboration. How do we build systems without a central server?
- Advantages:
 - No single point of failure.
 - Scalability by adding peers.
- Disadvantages:
 - Security and consistency are harder to manage.
- Examples: BitTorrent, blockchain networks.



MONOLITHIC ARCHITECTURE

11

- Focus/Dimension: Unified deployment & simplicity
- Concept: Entire application packaged and deployed as a single unit
- Advantages:
 - Easy to develop and test initially
 - Simple deployment (one file, one process)
 - Good performance for small-to-medium systems
- Disadvantages:
 - Hard to scale specific parts independently
 - Small change → redeploy the entire system
 - Becomes complex and fragile as system grows
- Question it answers: How can we quickly build and deploy a simple, unified application?
- Examples: Early versions of e-commerce apps, small internal business tools

COMBINING PATTERNS

12

- Patterns are **guidelines, not rigid rules**.
- In real world projects: Use multiple patterns together for different concerns.
- **Choose** which combination of patterns meets performance, maintainability, and scalability goals?
- **Example:** Client-Server with MVC and Layered design on the server.

LINKS

- <https://www.youtube.com/watch?v=pOlA4nkkbbI>
- <https://www.youtube.com/watch?v=f6zXyq4VPP8>
- <https://www.geeksforgeeks.org/system-design/design-patterns-architecture/>