Architectural Styles

UNSW
SYDNEY

These lecture slides are from the book "*Head First Software Architecture*",

by Raju Gandhi, Mark Richards, Neal Ford, O'Reilly Media, Inc., March 2024

Introduction to Architectural Styles

❖ Architectural Styles:

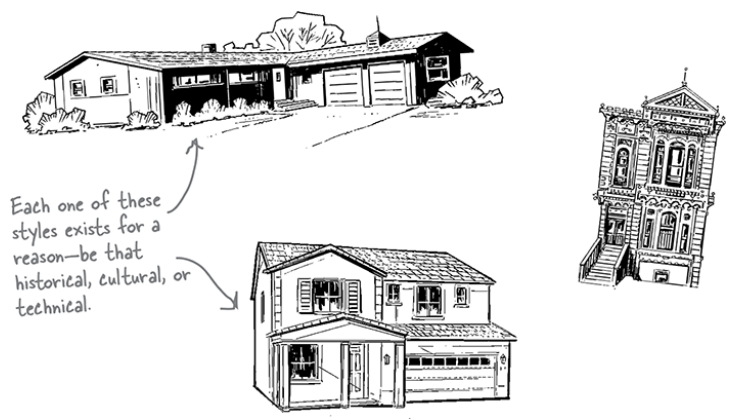
- Predefined patterns and philosophies guiding how software systems are structured and deployed.

❖ Importance of Understanding Styles:

- Facilitates better design decisions.
- Aligns software architecture with project needs.

❖ Example:

- Residential housing styles influenced by geography, climate, personal preference. Similarly, software architecture varies by project requirements.



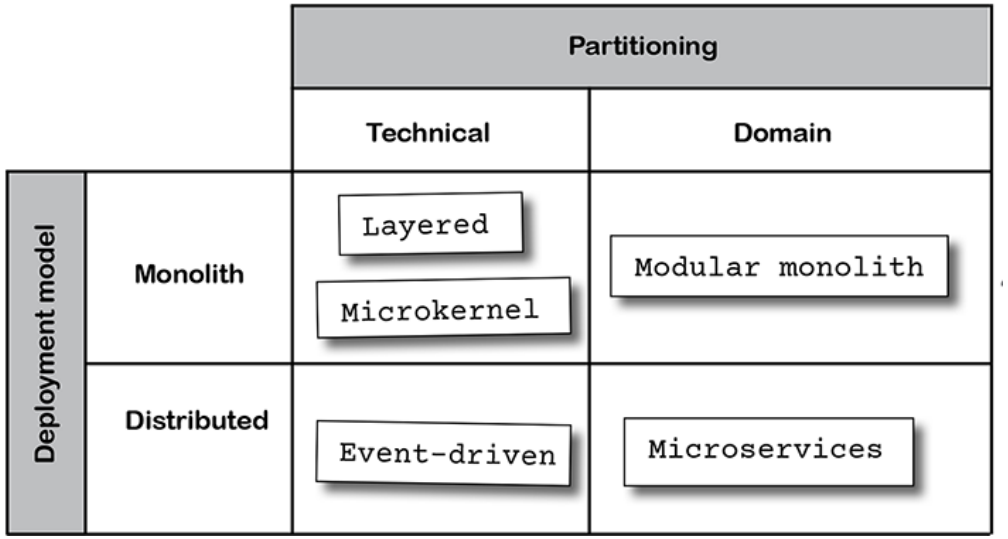
Categorizing Architectural Styles

Two main categories for architectural styles:

1. Partitioning
   - Technical vs. Domain-based.

2. Deployment
   - Monolithic vs. Distributed.

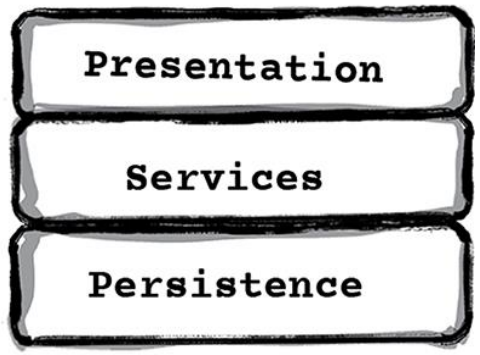
❖ Why Categorize?
   - Helps systematically analyse and select appropriate architecture.



Partitioning by Technical Concerns

Technical Partitioning:

- Code organized by functional roles or technical layers.



Characteristics:

- Clear separation of responsibilities.
- Easier specialization of teams.

Example: A standard web application:

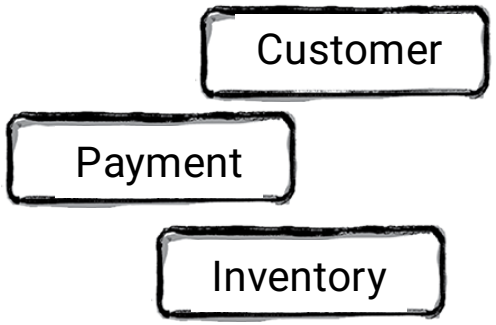
- Presentation Layer (UI);
- Business Logic Layer (Services)
- Data Persistence Layer (Database)

➢ Real-world Analogy:
Roles in a fancy restaurant (host, server, chef, busser) clearly divided by technical concerns (greeting, cooking, cleaning).

Partitioning by Domain Concerns

Domain Partitioning:

- Code organized around business domains or problem areas.



Characteristics:

- Alignment with business goals.
- Easier maintenance of related features.
- Strong domain modeling.

Example: An e-commerce platform:

- Customer Domain (user accounts, user interface)
- Inventory Domain (product catalog, stock management)
- Payment Domain (billing, transactions)

➢ Real-world Analogy:
Food court restaurants, each specialised in distinct cuisines (pizza, salads, burgers).

Comparing Technical vs. Domain Partitioning

Technical Partitioning	Domain Partitioning
Layered by technical roles	Organized by business areas
Easier for specialised teams	Aligned closely with business needs
Risk of over-generalisation	Risk of duplicating common functionalities

Example Scenario: A banking application:

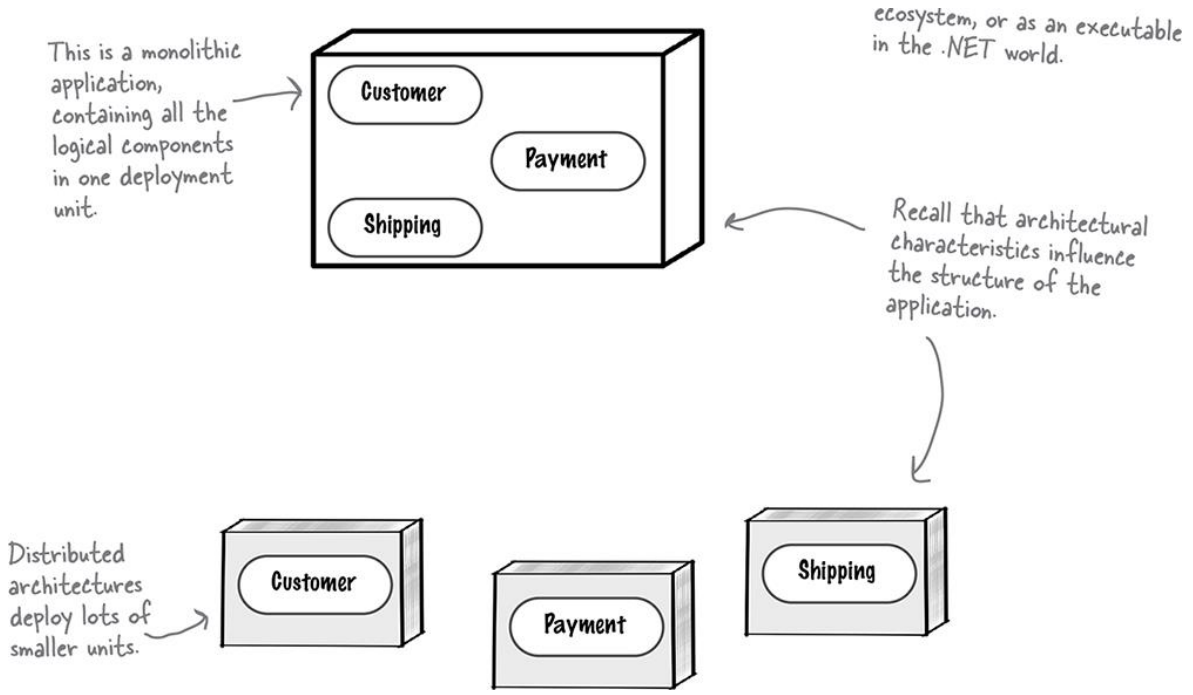
- ***Technical***: Separate teams for frontend, backend, DB administration.
- ***Domain***: Separate teams for loans, investments, account management.

Deployment Models Overview

1. Monolithic Architecture
   - Single deployable unit.

2. Distributed Architecture
   - Multiple deployable units communicating over networks.

Choice affects scalability, complexity, and cost.



Monolithic Architecture – Overview and Pros

Monolithic:

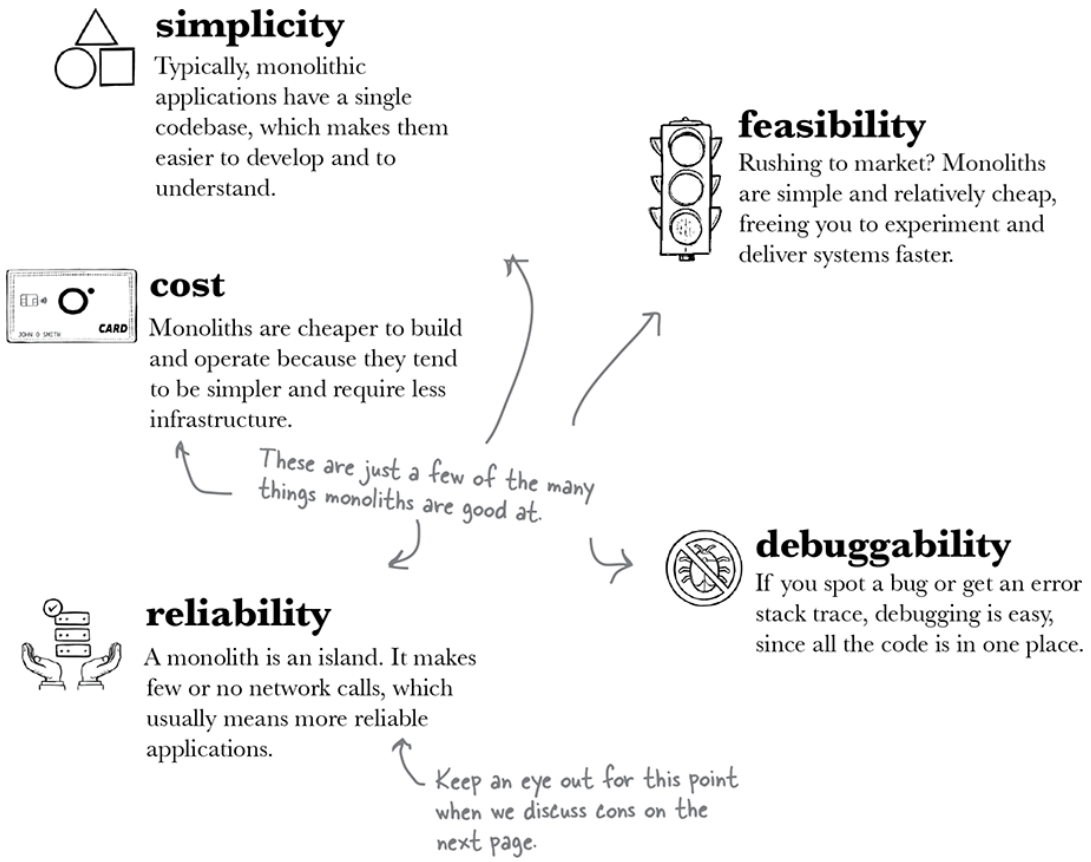
- Entire application deployed as one single executable or package.

Pros:

- Easier initial development.
- Simplified debugging.
- Lower initial deployment cost.

Examples:

- A single .jar (Java) or .exe (.NET) containing all app logic and resources.
- Smartphone as a single device doing many functions (calling, browsing, tracking).



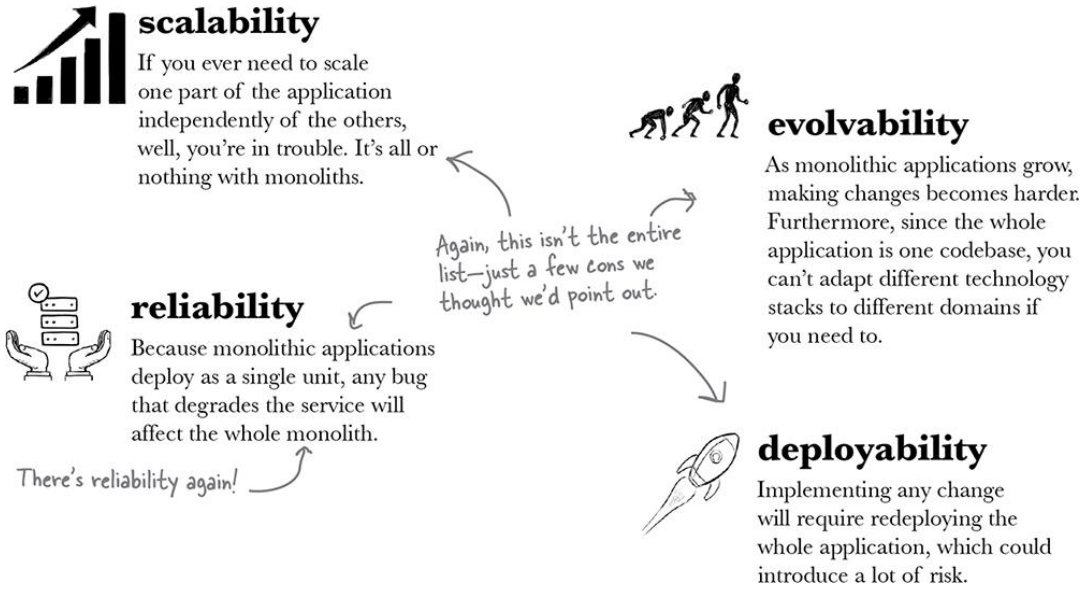
Monolithic Architecture - Limitations

Cons:

- Difficult to scale independently.
- Single bug can disrupt entire system.
- Inflexible when adapting to changing demands.

Example:

- Scaling a monolithic online store application
- Scaling means duplicating the entire application, increasing resource consumption significantly.



Distributed Architecture - Overview

Distributed:

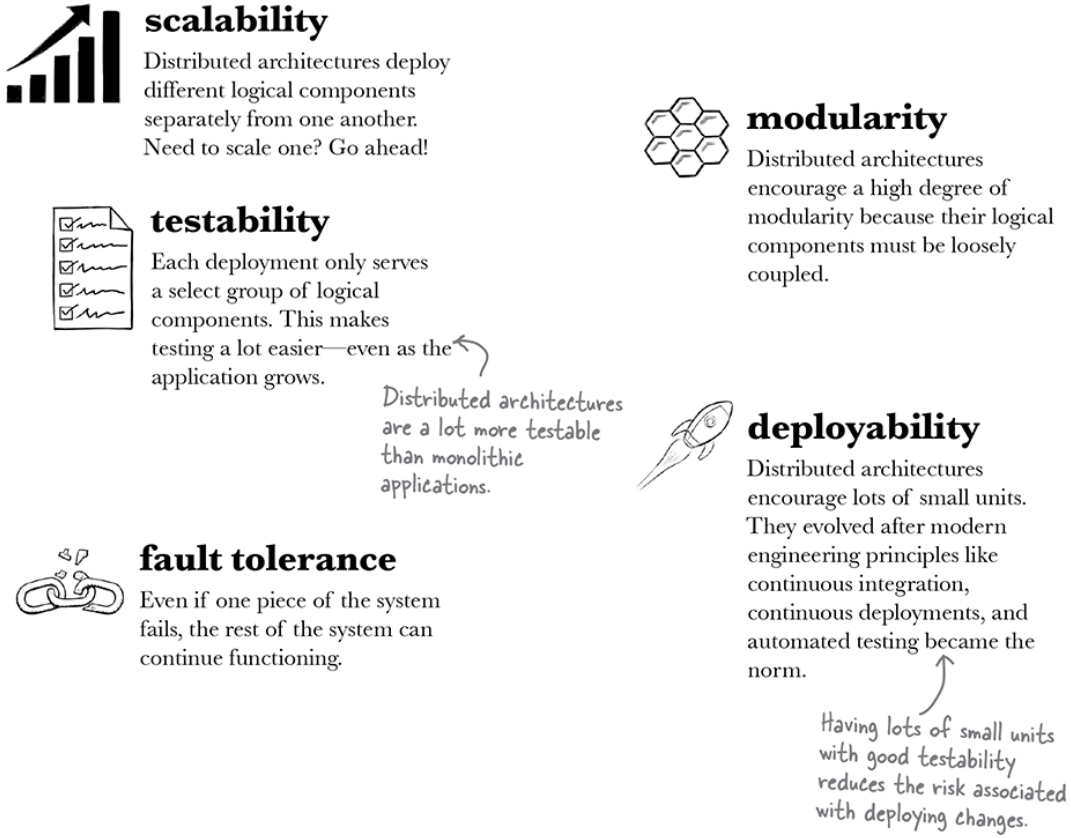
- Application components deployed separately, each as individual processes/services.

Pros:

- Independent scalability of components.
- Encourages modular design.
- Fault isolation—failures affect only single units.

Example:

- Microservices architecture for Netflix or Amazon, allowing independent scaling of services like user management, video streaming, and recommendation systems.



Distributed Architecture - Challenges

Cons:

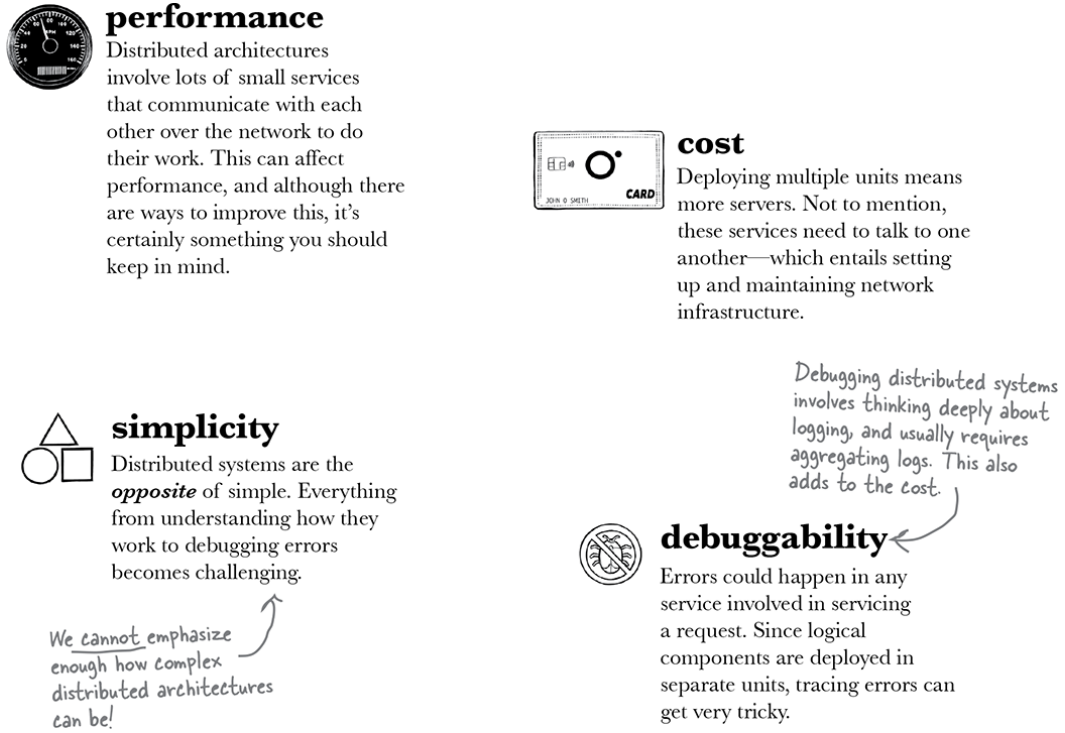
- High complexity due to network dependence.
- Increased maintenance and debugging complexity.
- Higher infrastructure and operational costs.

Example:

- Managing distributed transactions across services—complex coordination required, increased risk of partial failures.

Real-world Analogy:

- Earlier days—separate devices for GPS, web browsing, and phone calls each required separate maintenance and integration.



Comparing Monolithic vs. Distributed

Monolithic	Distributed
Simpler development & debugging	Complex system integration
Lower initial costs	Higher upfront infrastructure cost
Scaling is all-or-nothing	Individual services scalable
Single failure disrupts whole system	Fault tolerance through isolation

Discussion - Regulatory and Compliance Needs

Consider special needs like:

- Regulatory compliance (e.g., financial industry).
- Security requirements.

Monolithic:

- Easier control and monitoring in regulated environments.

Distributed:

- Can complicate compliance but increases modularity and maintainability.

Example:

- Banking systems might use monolithic for core banking due to tight regulatory controls, however distributed services for customer engagement modules.

Key Takeaways

- Numerous architectural styles exist; each with unique characteristics and trade-offs.
- Partitioning styles: Technical vs. Domain.
- Deployment models: Monolithic vs. Distributed.
- Choice of style influenced by:
  - Project goals.
  - Scalability requirements.
  - Complexity management.
  - Cost implications.