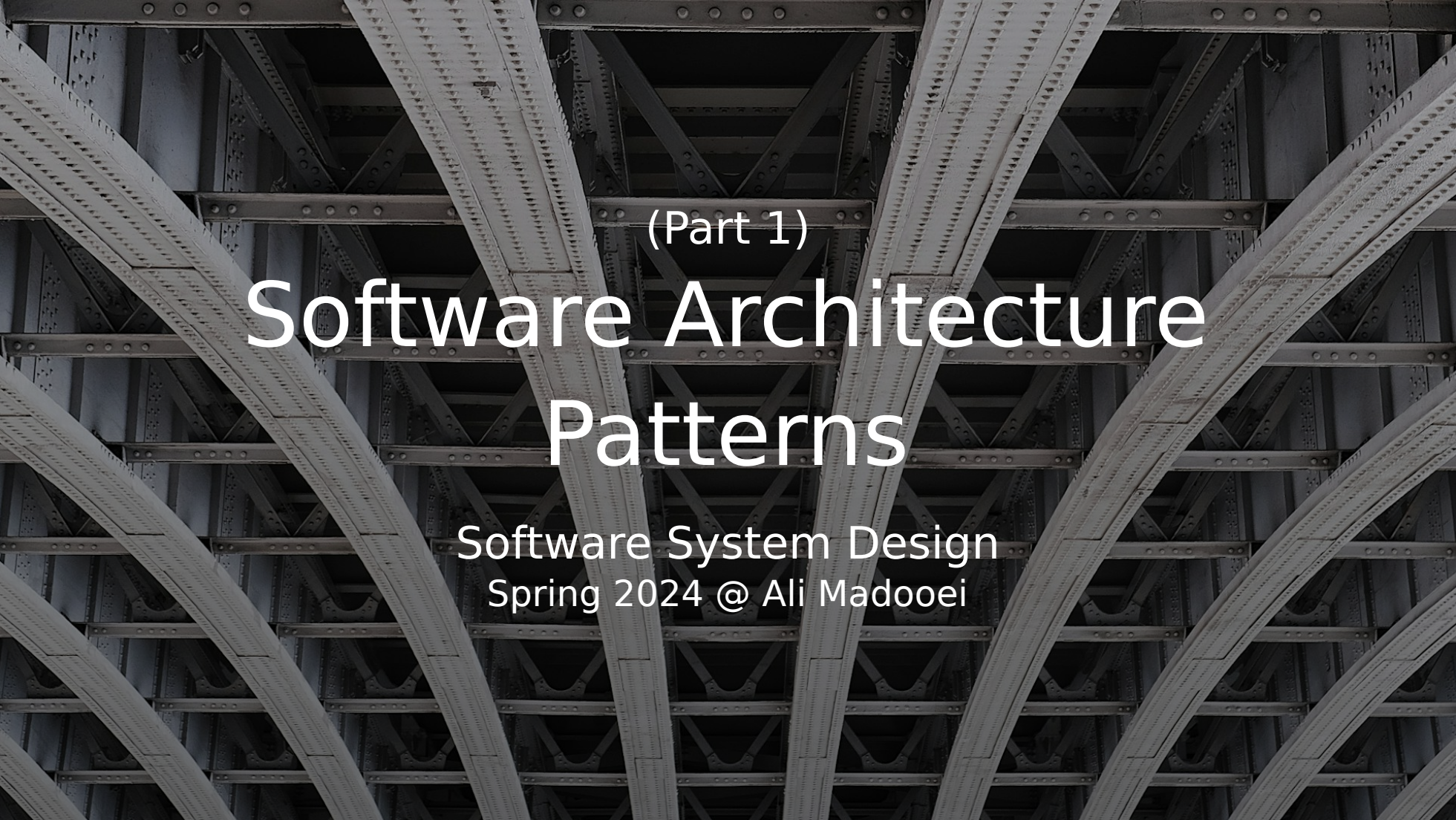

(Part 1)

Software Architecture Patterns

Software System Design

Spring 2024 @ Ali Madooei

Learning Outcomes

By the end of this lecture, you should be able to:

- Identify the key characteristics and use cases of traditional (monolithic, layered, client-server) and modern (microservices, event-driven, serverless) software architecture patterns.
- Differentiate between synchronous and asynchronous communication in distributed systems, and describe the importance of message queues and event streams in event-driven architecture.
- Evaluate the suitability of each architecture for specific use cases.

Software Architecture Patterns

Software architecture is the blueprint of a system.

- Patterns provide reusable solutions to common problems.
- Right architectural choices lead to efficient and scalable systems.

Software Architecture Patterns

Traditional Architectures

- Monolithic Architecture
- Client-Server Architecture
- Layered Architecture

Modern Architectures

- Microservices Architecture
- Event-Driven Architecture
- Serverless Architecture

Case Study: The "Posts" App

You are working on a minimalist post-sharing app. It functions like a simplified and scaled-down version of Twitter, where users can post, like, and comment on content.

LAMP Stack

LAMP stack (Linux, Apache, MySQL, PHP/Perl/Python) was a popular open-source web platform in the late 1990s and early 2000s that was used to run dynamic web applications.

- Your server runs a Linux operating system and hosts the entire application.
- On this operating system, you run an Apache web server to serve your web pages and handle HTTP requests.
- You store your data in a MySQL database which runs on the same server.
- Your application logic is written in PHP, Perl, or Python.

Monolithic Architecture

- Early days of software development
- LAMP stack is a monolithic architecture
- All components of the application are tightly coupled

Monolithic architecture, as the name suggests, is like a single large block or "monolith". In this design, everything from user interface functions to data access and storage is bundled into a singular codebase and deployed as a single unit.

Monolithic: Characteristics

- **Tightly Coupled**: All components are interconnected and interdependent.
- **Scalability**: Scaling the application means scaling the entire monolith and typically requires downtime.
  - **Vertical Scaling**: Scaling is done by increasing the resources of the server.
  - **Horizontal Scaling**: Not easily achieved due to the monolithic nature.
- **Development**: Easier to develop and test due to the centralized codebase.
- **Deployment**: Deployed as a single unit, which can be simpler.
- **Maintenance**: Changes to one part of the application can affect other parts.

Activity: Monorepo vs. Monolith

- What do you understand by the term "monorepo"?
- How does it differ from or relate to "monolithic architecture"?
- Can you have a monorepo without a monolithic architecture? Discuss!

Solution: Monorepo vs. Monolith

While a monorepo can contain a monolithic application, they are distinct concepts. A monorepo doesn’t inherently mean the software is monolithic.

- **Monorepo:** A single repository storing code for multiple projects.
  - Facilitates code sharing and reduces duplication.
  - Can contain various types of projects: monoliths, microservices, libraries.
- **Monolithic Architecture:** A single codebase with all application components.
  - All components are interconnected and interdependent.
  - Deployment involves updating the entire system.

Ruby on Rails Framework

Ruby on Rails is a popular web application framework written in Ruby that gained significant popularity in the mid-2000s.

- It follows the Model-View-Controller (MVC) pattern which is an example of a *layered architecture*.
- Rails applications are typically monolithic in nature so the entire application is bundled together and deployed as a single unit.
- However, the application logic is organized into models, views, and controllers which helps in maintaining the codebase.
- The Ruby on Rails framework became popular for its convention over configuration approach and the ability to rapidly develop web applications.

Layered Architecture

An architecture pattern that divides an application into layers to separate concerns and responsibilities.

- Each layer has a specific responsibility and interacts with the layers above and below it in a well-defined manner.
- Common layers include presentation, business logic, and data access.
- Model-View-Controller (MVC) is an implementation of the layered architecture commonly used in web applications.

"Posts" App as a Layered Architecture

- **Presentation Layer:** This would include the user interfaces, encompassing elements like post displays, user profiles, and interactions like liking or commenting on a post.
- **Business Logic Layer:** Here, we'd process user requests. It would handle functionalities like creating posts, adding likes, managing user profiles, and more.
- **Data Layer:** This is where all data is managed. It would be responsible for storing user profiles, posts, likes, comments, and any other essential data. It ensures data integrity, consistency, and efficient retrieval.

Layered: Characteristics

- **Modularity**: Each layer has a specific responsibility. This separation of concern reduces complexity and makes maintenance and updates easier.
- **Scalability**: Easier to scale (optimize) individual layers but can be challenging to scale the entire application.
- **Deployment**: Typically deployed as a single unit (monolith) but can be broken down into separatly deployable components.

MERN Stack

MERN stack (MongoDB, Express.js, React, Node.js) is a popular open-source web platform used to build full-stack web applications. It became popular in the mid-2010s.

- MongoDB is a NoSQL database used to store data.
- Express.js is a web application framework for Node.js that provides a set of features for building web applications.
- React is a JavaScript library for building user interfaces.
- The React application is typically served separately leading to the application being split into two parts: the frontend (React) and the backend (Node.js and Express.js).

Client-Server Architecture

A common architecture pattern where the client (user interface) and server (backend) are separate entities.

- MERN stack is an example of a client-server architecture where the client is the React application and the server is the Node.js and Express.js backend.
- It can be thought of as a layered architecture where the client and server are separate layers, deployed on different machines, and communicate over a network.
- The server itself can be a layered architecture (e.g., separate layers for routing, business logic, and data access).
- You can have multiple clients (e.g., web, mobile) interacting with the same server.

"Posts" App in Client-Server Architecture

- **Client (Frontend):** React-based web app.
  - Responsible for rendering UI and user interactions.
  - Communicates with the server via APIs for data and actions.
- **Server (Backend):** Express.js with Node.js.
  - Handles business logic, database interactions, and API endpoints.
  - Communicates with the database (e.g., MongoDB) for data storage.
- **Deployment:**
  - Client deployed on static hosting platforms like AWS S3 or GitHub Pages site.
  - Server hosted on Compute Engine like AWS EC2 or Heroku.

Activity: Understanding the Client

- How does the term "client" differ in context when we talk about browsers vs. web apps?
- What role does the web server play in serving frontend applications?
- How does the backend server differ from the web server, especially in a client-server model?

Solution: Understanding the Client

- **Browser Client:**
  - User’s interface to the web; makes requests for web pages.
  - Interacts with the web server to fetch and display web content.
- **Web Server:**
  - Could be as simple as serving static assets like HTML, CSS, and JavaScript files.
  - Responds to browser requests; might not handle business logic.
  - Can act as clients when interacting with the backend server
- **Backend Server:**
  - Processes application logic, database interactions, and API management.
  - Communicates with web apps (clients) and provides dynamic data.

Client-Server: Characteristics

- **Separation of Concerns**: The client is responsible for the user interface and the server is responsible for the application logic and data storage.
- **Scalability**: Easier to scale the client and server independently.
- **Deployment**: The client and server can be deployed on different machines.
- **Communication**: The client and server communicate over a network using protocols like HTTP.

Activity: Client-Server vs. Layered

- How does the client-server architecture differ from or relate to the layered architecture?

Solution: Client-Server vs. Layered

- **Client-Server Architecture:**
  - High-level division between client and server.
  - Client handles UI; server manages data and logic.
  - Focuses on the distribution of tasks.
- **Layered Architecture:**
  - Granular division into layers, especially within the server.
  - Each layer has a specific responsibility.
  - Focuses on separation of concerns.

Service-Oriented Architecture (SOA)

While client-server architecture focuses on the separation of the client and server components, Service-Oriented Architecture (SOA) takes this concept further by breaking down the server-side components into smaller, more modular services.

- In MERN stack, it is common to have separate services for database management and API routing.
- We can scale the database service independently of the API service.

Activity: SOA vs. Layered

- How does Service-Oriented Architecture (SOA) differ from or relate to Layered Architecture?

Solution: SOA vs. Layered

- **Layered Architecture**: Separate functionalities within a single application.
  - Presentation, Business Logic, Data Access.
- **Service-Oriented Architecture (SOA)**: Externalize these functionalities as distinct services.
  - Each service is reusable across different applications.
  - Communicate over network using APIs.