# Introduction to Software Architecture

# Learning Objectives

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

- Understand the distinction between software architecture and design patterns
- Recognize common architectural styles
- Understand why these common architectural styles are used

# Software Architecture

Software architecture is the process of designing the global organization of the system, specifically:

- Dividing the system into subsystems
- Determining how subsystems interact with one another (when and with whom)
- Identifying how the subsystems are deployed (e.g. into same/different processes or machines altogether!)

Software architecture is high-level design:

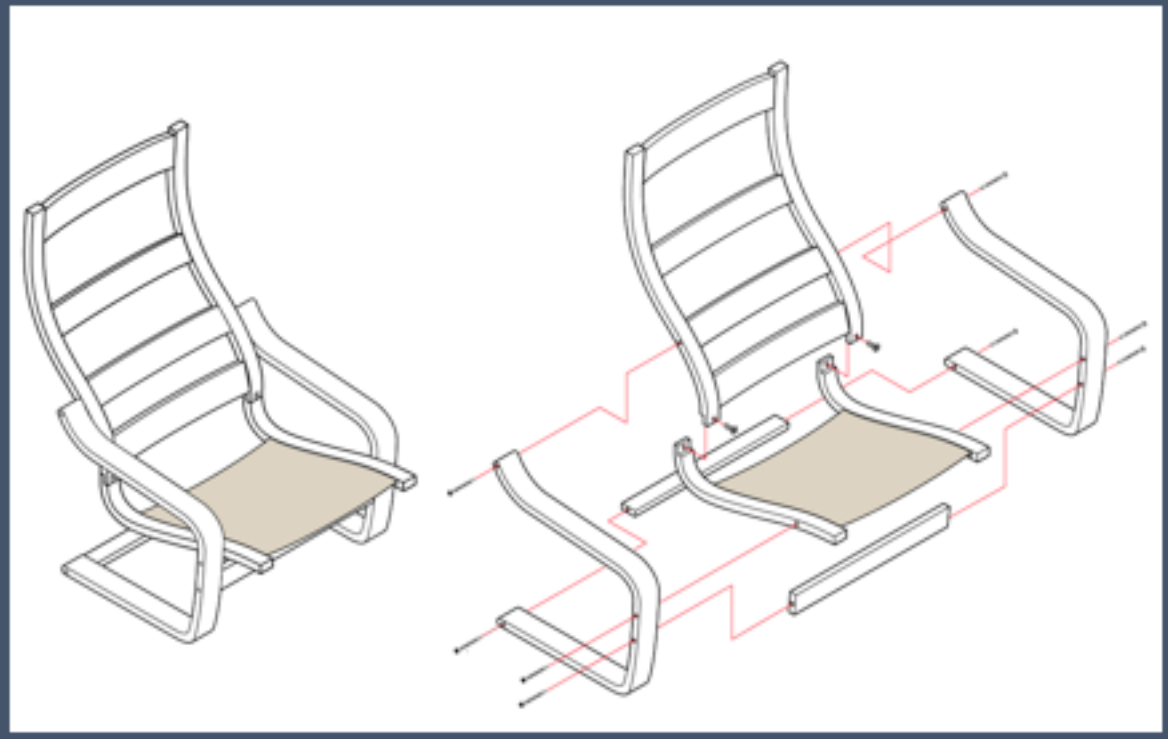
- Abstract away details of individual classes
- Subdividing system into manageable subsystems
- Considering how these pieces fit together

# Why is an architecture important?

Supports understanding of the system

Allows sub-teams or individuals to work on subsystems in isolation

Enables re-use and re-usability



# Partitioning Considerations

Hardware constraints (e.g. external database server; scalability)

Computational cost (parallelizing expensive operations)

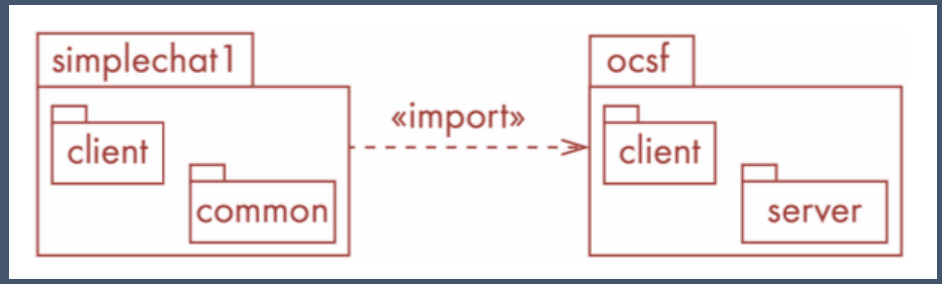
Logical divisions (e.g. UI, business logic, database, etc.)

Communication

- Intraprocess (same method, same thread, different threads)
- Interprocess (same node)
- Interprocess (different nodes)

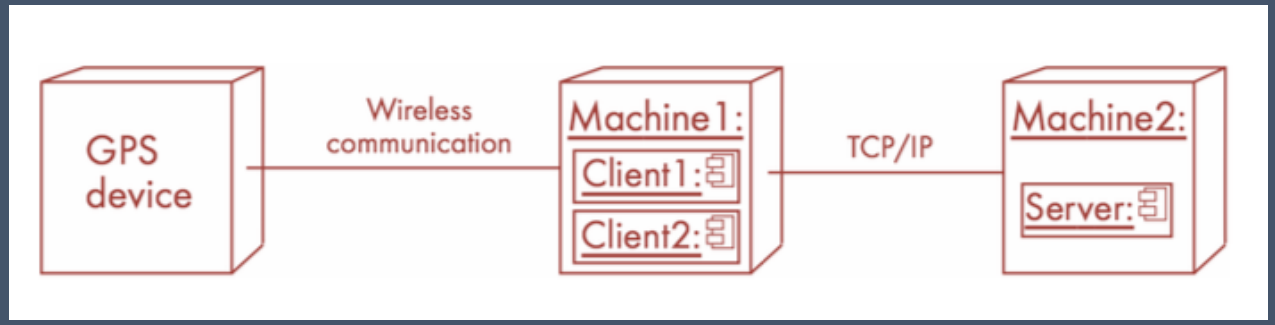
# Some UML for describing architecture too!

Package diagram
(emphasizes logical structure)




Component diagram
(emphasizes interfaces/communication between components)


Deployment diagram
(emphasizes deployment model)



# Architectural Models are judged on Stability

Stable: new features can be easily added with no or small changes to the architecture

So: an architectural model ought to be designed to be stable—this aids maintainability, extensibility, and reliability of the system in the long term

# Common Architectural Styles

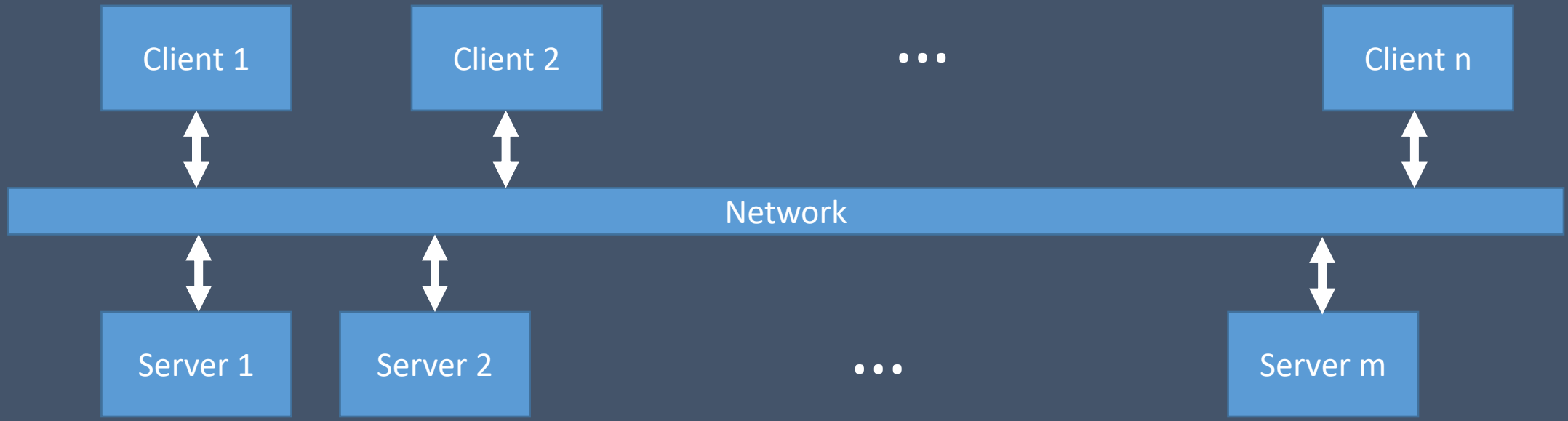
Client/server
Pipes-and-filters
Repository
Model-view-controller
Layered (three-tier, four-tier)
Peer-to-peer
Interpreter
Plugin
Component-based
Event-based
…

# Client-Server Style

Stand-alone servers

Stand-alone clients

Network connects them



# Examples

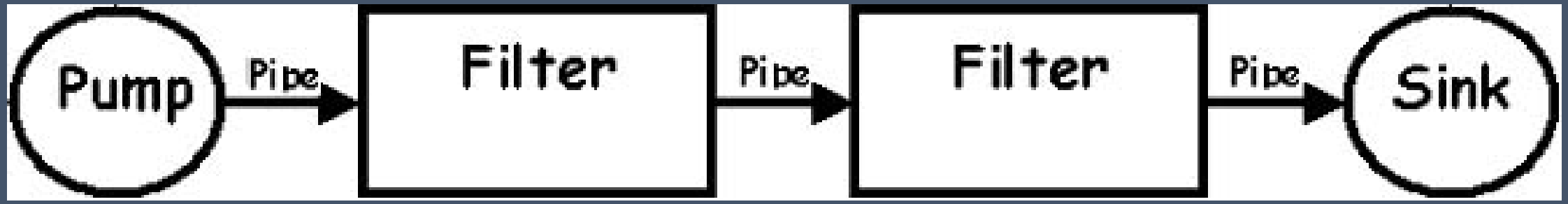
- Web Servers : Apache, Microsoft IIS
- E-mail Servers: IMAP - Internet Message Access Protocol , SMTP  - Service Mail Transfer Protocol, POP3 - Post Office Protocol V3
- Domain Name Server (DNS)

# Pipes and Filters

Pump = source

Filter = small, self-contained units that take input from a pipe, transform the input, and send it out to another pipe

Sink = something that

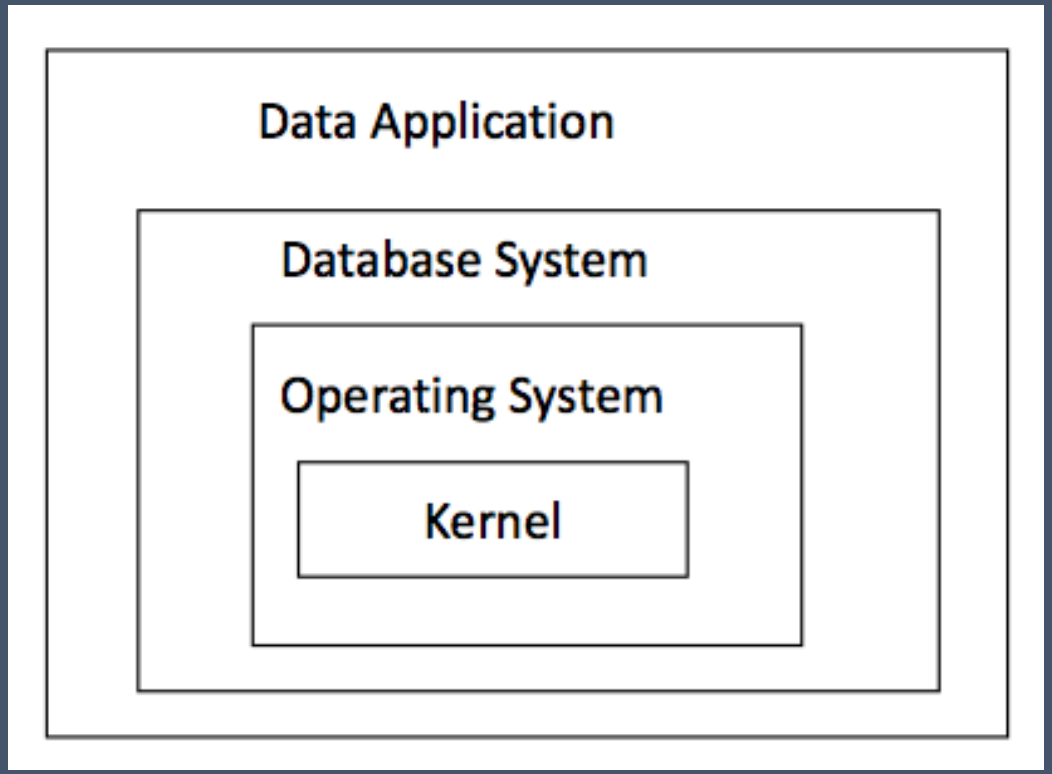


# Layered Architecture

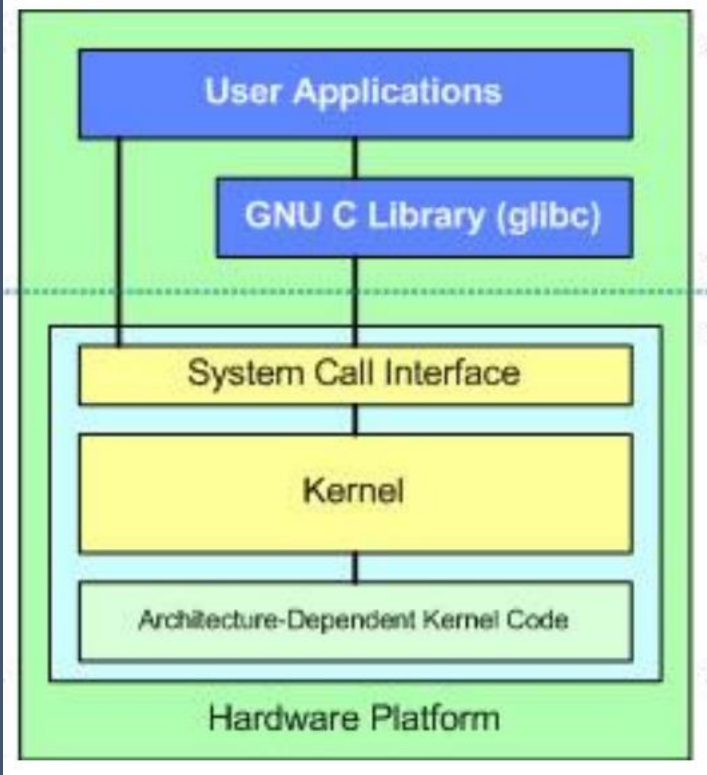
Sub-systems are organized into layers

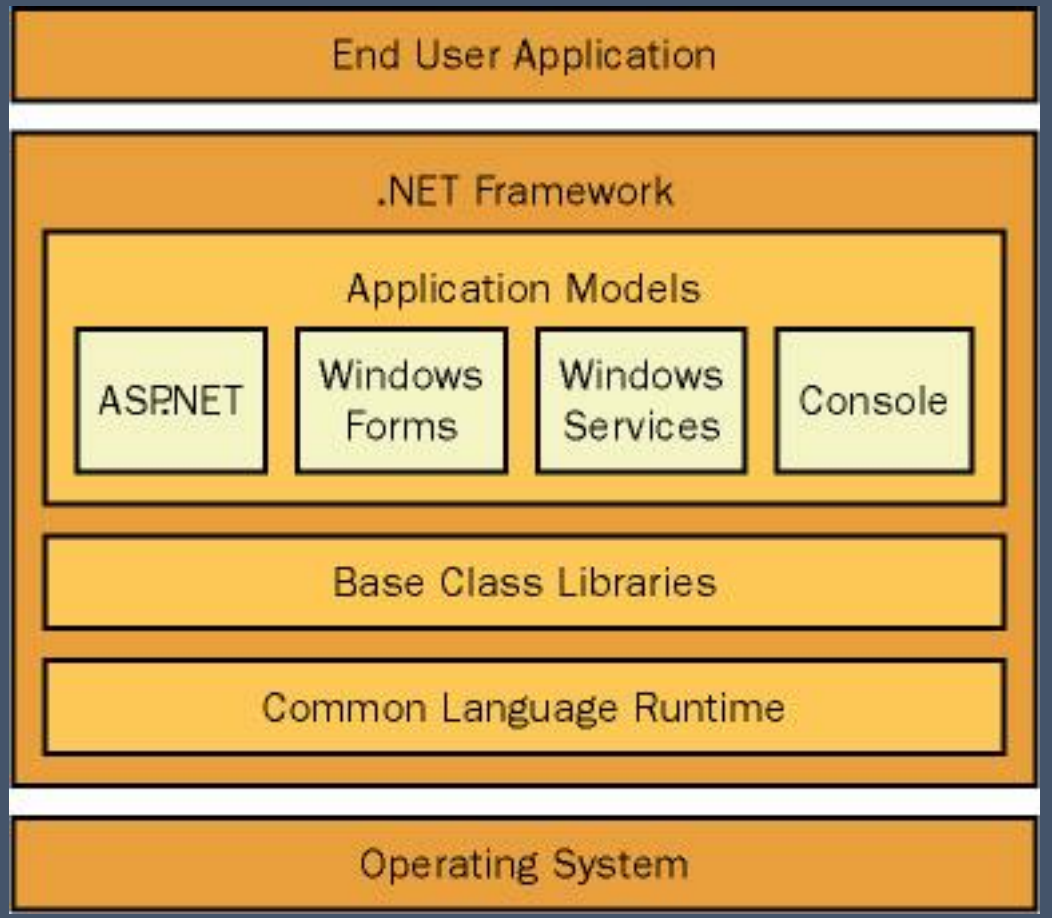
Each layer:

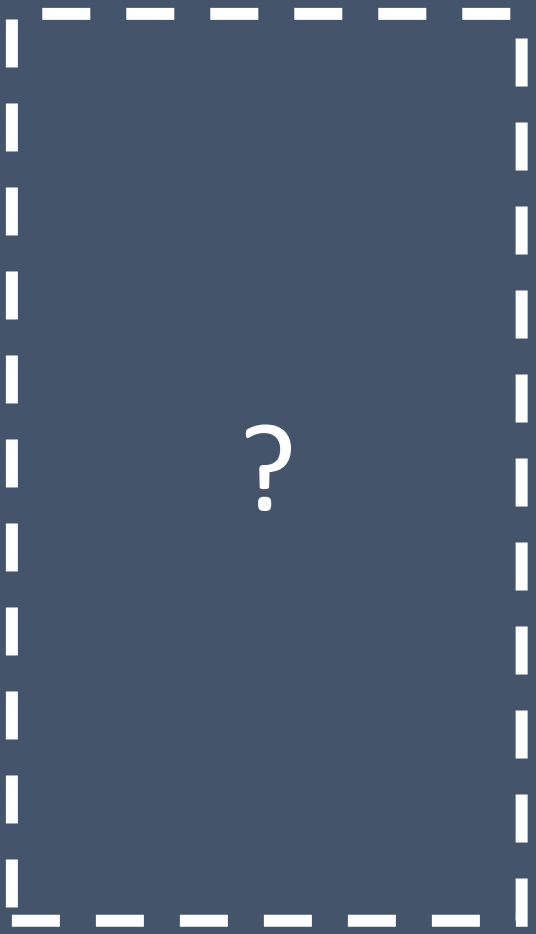
- uses the services of the layer below
- provides services to layer above through well-defined interfaces



# Layered Architecture Examples



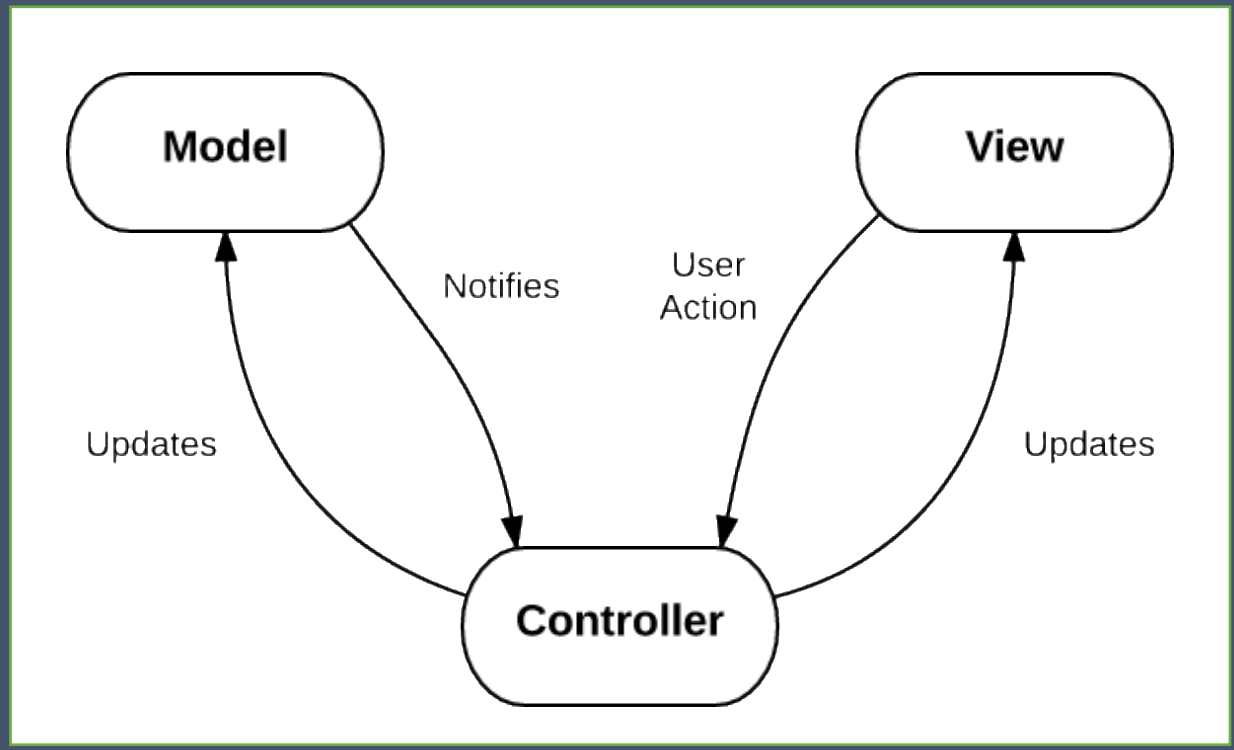
# Model-View-Controller Architectural Style

Separation of M, V and C:

**Model**: manages behaviour of data; responds to requests about state (from View), responds to state change commands (Controller)

**View**: manages display of info

**Controller**: interprets user input, and updates model and view



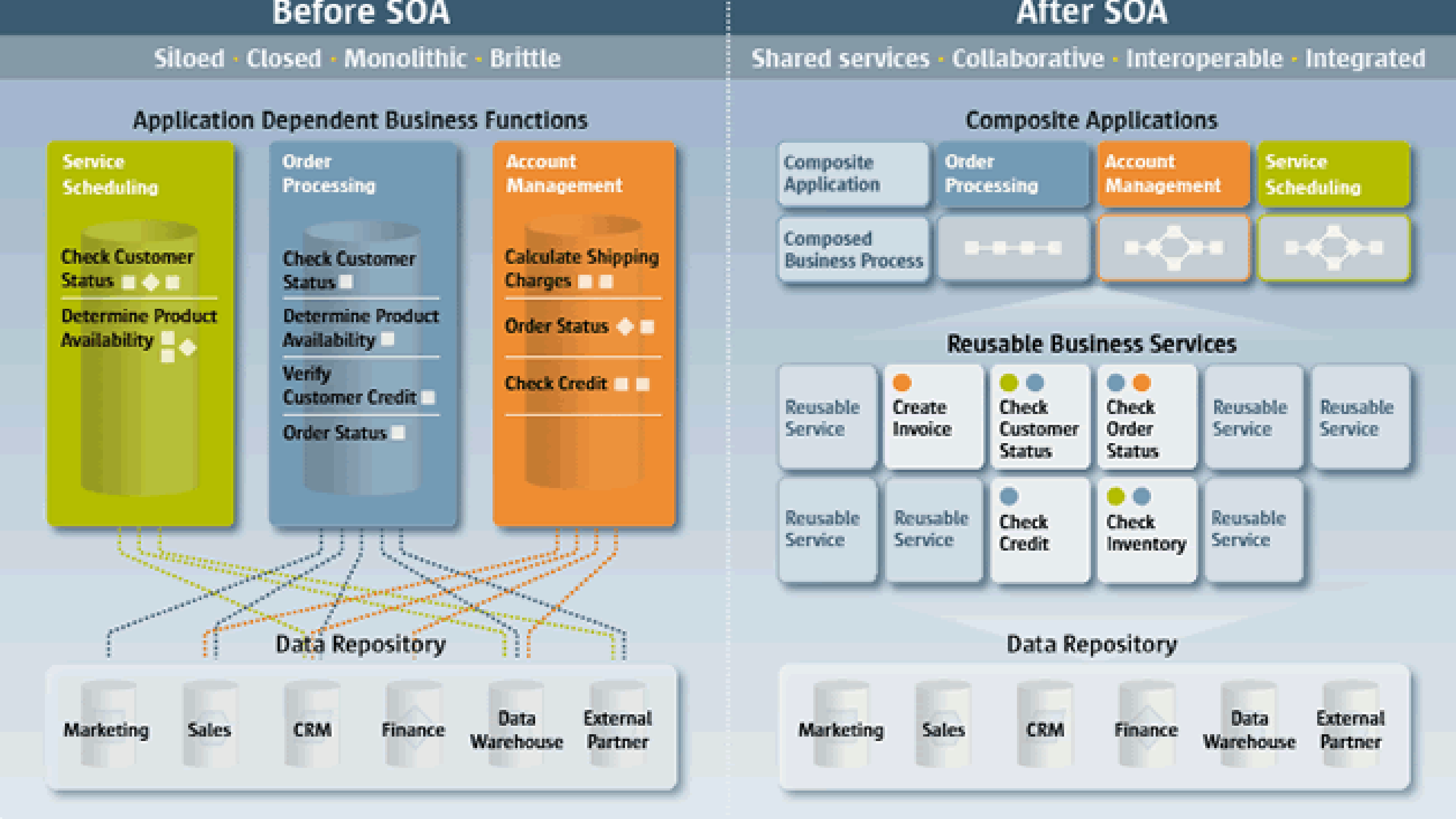
# Service Oriented Architectural Style

“New kid on the block”

Loosely-coupled, autonomous, distributed services

Close adherence to a schema and contract, not class (i.e. it’s usually about data)

Applications are then mostly about composing services together


Before SOA
Siloed · Closed · Monolithic · Brittle
Application Dependent Business Functions
Service Scheduling
Check Customer Status
Determine Product Availability
Order Processing
Check Customer Status
Determine Product Availability
Verify Customer Credit
Order Status
Account Management
Calculate Shipping Charges
Order Status
Check Credit
Data Repository
Marketing
Sales
CRM
Finance
Data Warehouse
External Partner
After SOA
Shared services · Collaborative · Interoperable · Integrated
Composite Applications
Composite Application
Order Processing
Account Management
Service Scheduling
Composed Business Process
Reusable Business Services
Reusable Service
Create Invoice
Check Customer Status
Check Order Status
Reusable Service
Reusable Service
Reusable Service
Reusable Service
Check Credit
Check Inventory
Reusable Service
Data Repository
Marketing
Sales
CRM
Finance
Data Warehouse
External Partner

# Architectural Style Overview

**Client-server:** Segregates the system into two applications, where the client makes requests to the server. In many cases, the server is a database with application logic represented as stored procedures.

**Pipes and filters:** Decompose a task that performs complex processing into a series of separate elements that can be reused

**Layered architecture:** Partitions the concerns of the application into stacked groups (layers).

**Model-view-controller:** The *Model-View-Controller (MVC)* pattern separates the modeling of the domain, the presentation, and the actions based on user input into three separate classes

**Service-Oriented Architectural style:** Refers to applications that expose and consume functionality as a service using contracts and messages.