Software Architecture

Some slides are adapted from Software Engineering by Ian Sommerville

Topics Covered

- Software architecture overview
- Architectural design decisions
- Architectural styles

Software Architecture

- Definitions
  - "The architecture of a software system defines that system in terms of **computational components** and **interactions** among those components" (Shaw et al., 1995)
  - "is the structure or structures of the system, which comprise **software elements**, the externally visible properties of those elements, and the **relationships** among them" (Bass et al., 2003)

Software Architecture - 2

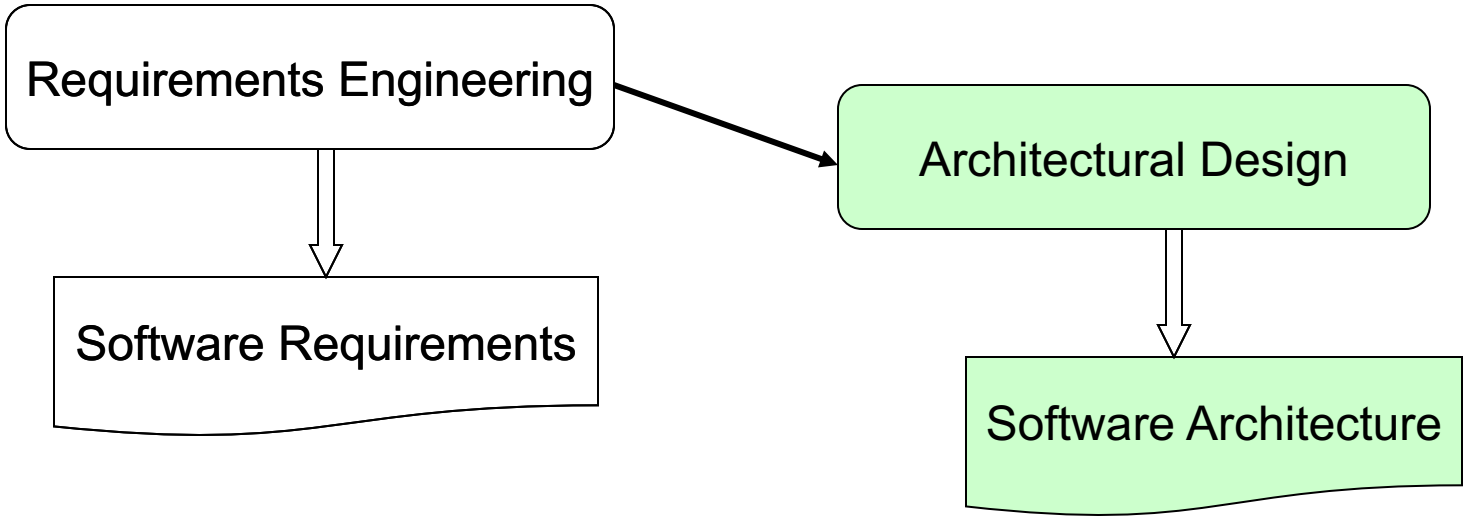
- Architectural design
  - design process for identifying the sub-systems making up a system and how they communicate

- Software architecture (document)
  - output of the design process (a description of the result from the process)

c++ là low level design

giao diện cũng là low level design

Architectural Design



Architectural Design - 2

tại sao phải chia ra nhiều phần nhỏ?

- dễ quản lí, dễ bảo trì, dễ chỉnh sửa bổ sung, dễ phân công công việc giữa các thành viên trong nhóm

- An early stage of design process
- Linking requirements engineering and design processes

  - mỗi thành phần ở mỗi máy dể phù hợp với khả năng có nhiều ng sd (khả nang scale up -> số ng dùng lớn)

- Involving
  - decomposing system into parts or components
  - identifying major system components and their communications
  - making design decisions and rationale behind decisions

Presenting Architecture

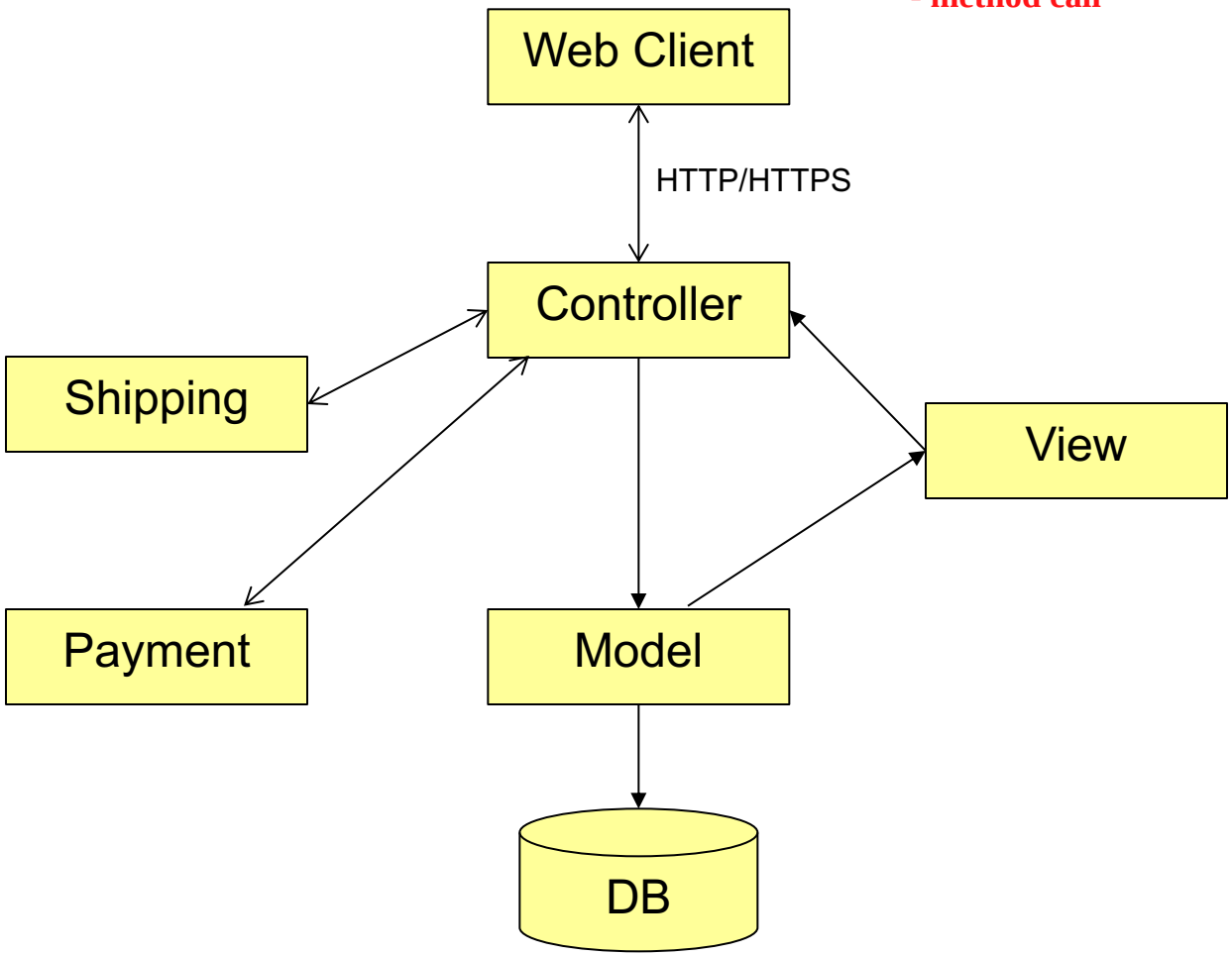
- During architectural design, system is decomposed into parts or components

- A simple representation is a block diagram presenting an overview of system structure
  - Using boxes and lines
  - Useful for communication with stakeholders and for project planning

Architecture for Hailua

có cách nào lk giữa các tp với nhau?

- method call



Group Exercise

- Discuss
  - What is software component?
  - List methods to connect components in software

Why We Need to Have Architecture?

- Stakeholder communication
  - Architecture used as a focus of discussion by system stakeholders
- System analysis
  - Analysis of whether the system can meet its **non-functional** requirements
  - Future extensions
- Large-scale reuse
  - Architecture may be reusable across a range of systems

Architecture Affects Non-functional Requirements

- Performance
  - Localize critical operations and minimize communications. Use large rather than fine-grain components
- Security
  - Use a layered architecture with critical assets in the inner layers
- Safety
  - Localize safety-critical features in a small number of sub-systems
- Availability
  - Include redundant components and mechanisms for fault tolerance
- Maintainability
  - Use fine-grain, replaceable components

nếu lỗi hoặc crash thì availability bị giảm, khi nhiều ng truy cập -> time out => giảm avail
làm sao để tăng avail?

Architectural Conflicts

- Using large-grain components improves performance but reduces maintainability
- Introducing redundant data improves availability but makes security more difficult
- Localizing safety-related features usually means more communication so degraded performance

Topics Covered

- Software architecture overview
- Architectural design decisions
- Architectural styles

Architectural Design Decisions

- Architecture is the result of design decisions made during (architectural) design
- Architects made series of design decisions during design
  - E.g., how many components, how they communicate, how components are secured, which languages used
- Elements of a design decision
  - Issues, decision, assumptions, rationale (reasons), alternatives, implications

Examples of design decisions

- Use AngularJs framework for web
- Use the cross-platform framework React Native for mobile apps
- Use MySQL for database
- Use MVC model for server-side
- Use Java for back-end/server-side components
- Daily backup

Architectural Design Decisions - 2

- Rationale behind design decisions are commonly undocumented

- Three types of undocumented design decisions
  - Decision is implicit: architects are unaware of decision
  - Decision is explicit but undocumented: architects are aware of decision but do not document
  - Decision is explicit and explicitly undocumented: reason is hidden

Architectural Design Decisions - 3

- Is there a generic application architecture that can be used?
- How will the system be distributed?
- What architectural styles are appropriate?
- How will the system be decomposed into modules?
- What control strategy should be used?
- How will the architectural design be evaluated?
- How should the architecture be documented?

Architectural Design Decisions - 4

- Example of design decision
  - Issue: system has to be maintainable
  - Decision: three-tier architecture, using object-oriented language
  - Rationale
    - with three-tier architecture, it is easy to change each tier without affecting others like interface or logic
    - OO program is easy to maintain than functional program
  - Alternatives: MVC, service-oriented architecture, n-tier

Concepts: Sub-systems and Modules

- A sub-system
  - is a system in its own right
  - its operation is independent of the services provided by other sub-systems
  - Example: Google maps embedded in an app
- A module (component)
  - is a component that provides services to other components but would not normally be considered as a separate system
  - Example: UI component on your website

Topics Covered

- Software architecture overview
- Architectural design decisions
- Architectural styles

Architectural Styles

- An architectural style describes a certain **reusable** arrangement of architectural elements
- An architectural style describes a common solution to a particular architectural problem
- Examples, MVC, n-tier, service-oriented

- Architectural style vs. design pattern
  - Design pattern: a common solution to design problem
    - Does not address structure of a complete system
    - Micro-architecture (at low level design)
  - Architecture style is at high-level design

Architectural Styles - 2

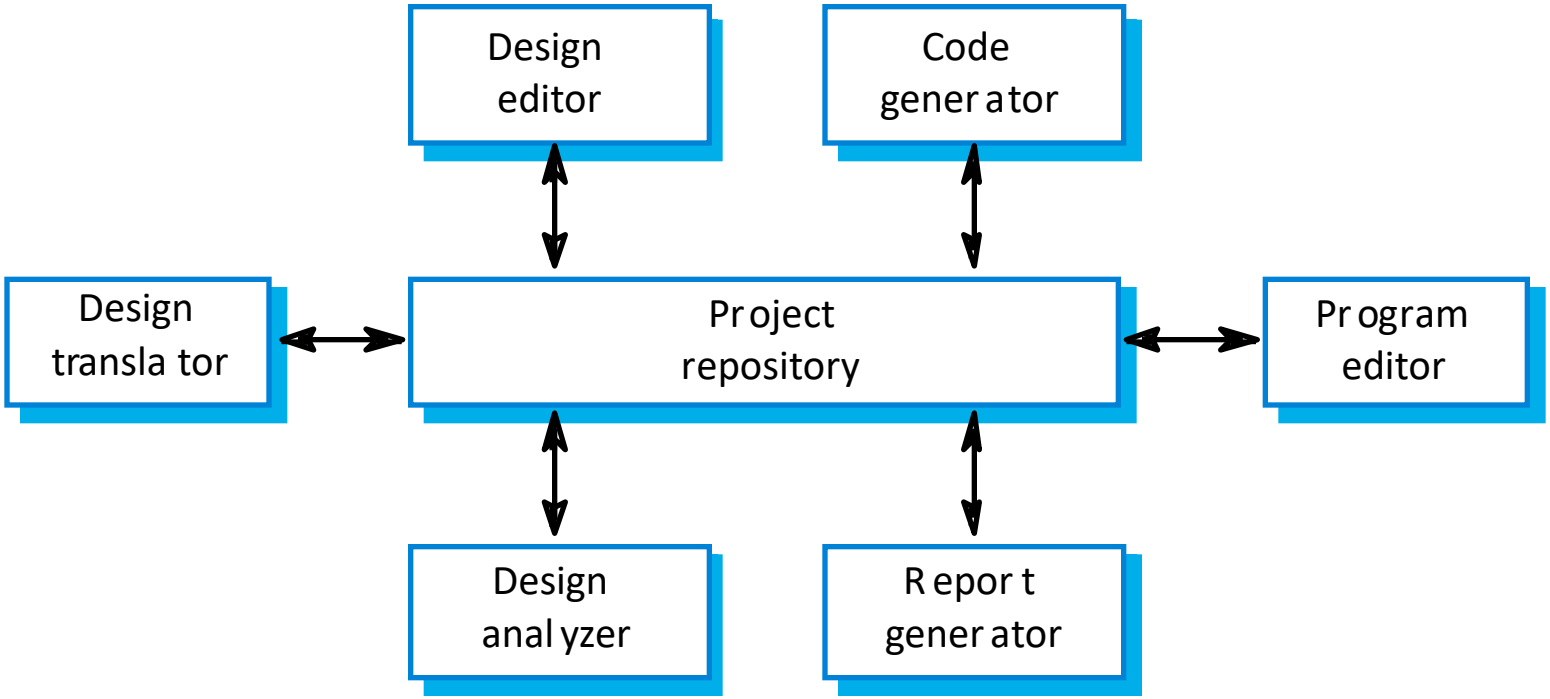
- Benefits
  - An awareness of architectural styles can simplify the problem of defining system architectures
  - **Reuse**: knowledge of how to solve a common and repeatable problem

- A style is described in form of
  - Problem: what problem the style describes
  - Context: constraints and characteristics of environment
  - Solution: how to solve the problem

Repository Style

- Sub-systems/components exchange data
  - Shared data is held in a central database or repository
  - Data may be accessed by all sub-systems
  - Each sub-system maintains its own database and passes data explicitly to other sub-systems
- Repository style is often used when sharing large amounts of data

CASE Toolset Architecture



Repository Style Characteristics

- **Advantages**
  - Efficient to share large amounts of data
  - Sub-systems need not be concerned with how data is produced
  - Centralized management, e.g., backup, security, etc.

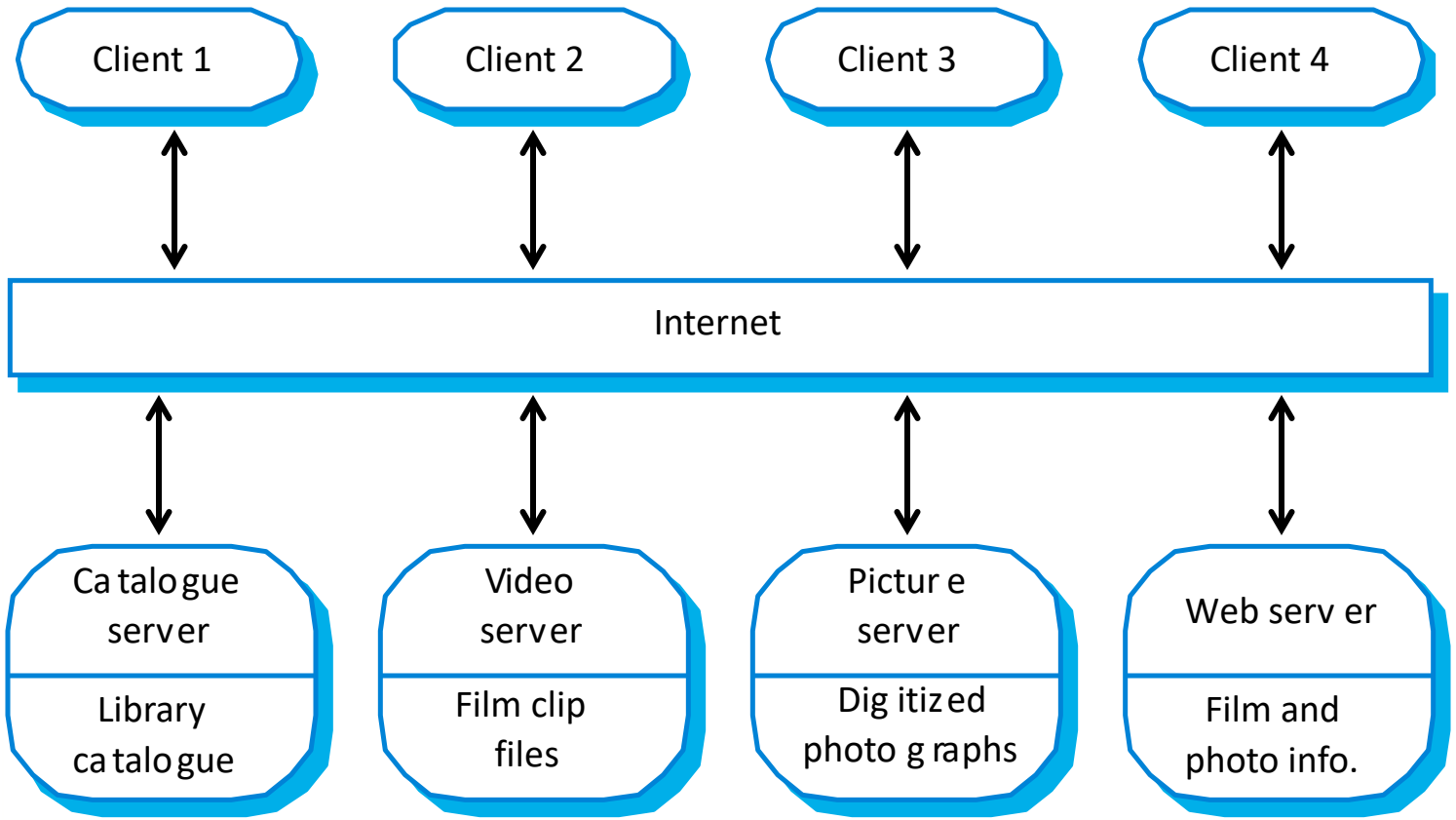
- **Disadvantages**
  - Sub-systems must agree on a repository data model
    - Inevitably a compromise
  - Data evolution is difficult and expensive
  - No scope for specific management policies
  - Difficult to distribute efficiently

Client-Server Style

- Shows how data and processing is distributed across a range of components

- Consisting of
  - Set of stand-alone servers providing specific services, e.g., printing, data management, etc.
  - Set of clients calling on these services
  - Network allowing clients to access servers

Film and Picture Library



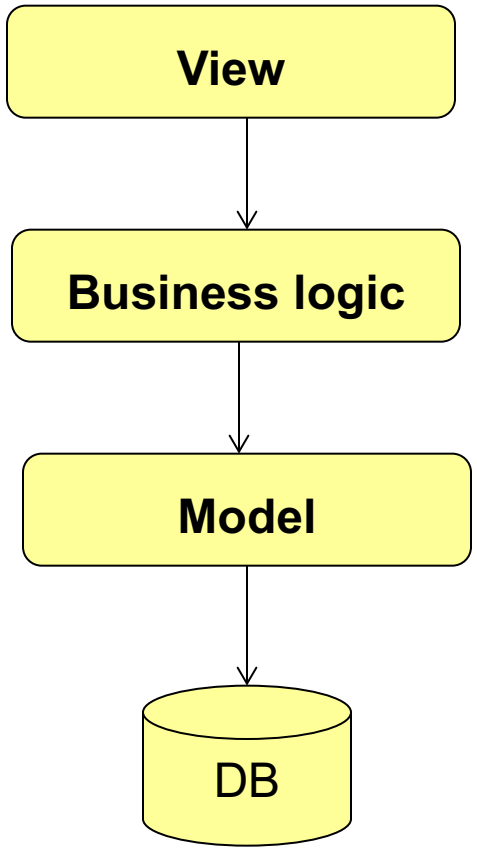
Client-Server Characteristics

- **Advantages**
  - Distribution of data is straightforward
  - Makes effective use of networked systems
  - May require cheaper hardware
  - Easy to add new servers or upgrade existing servers

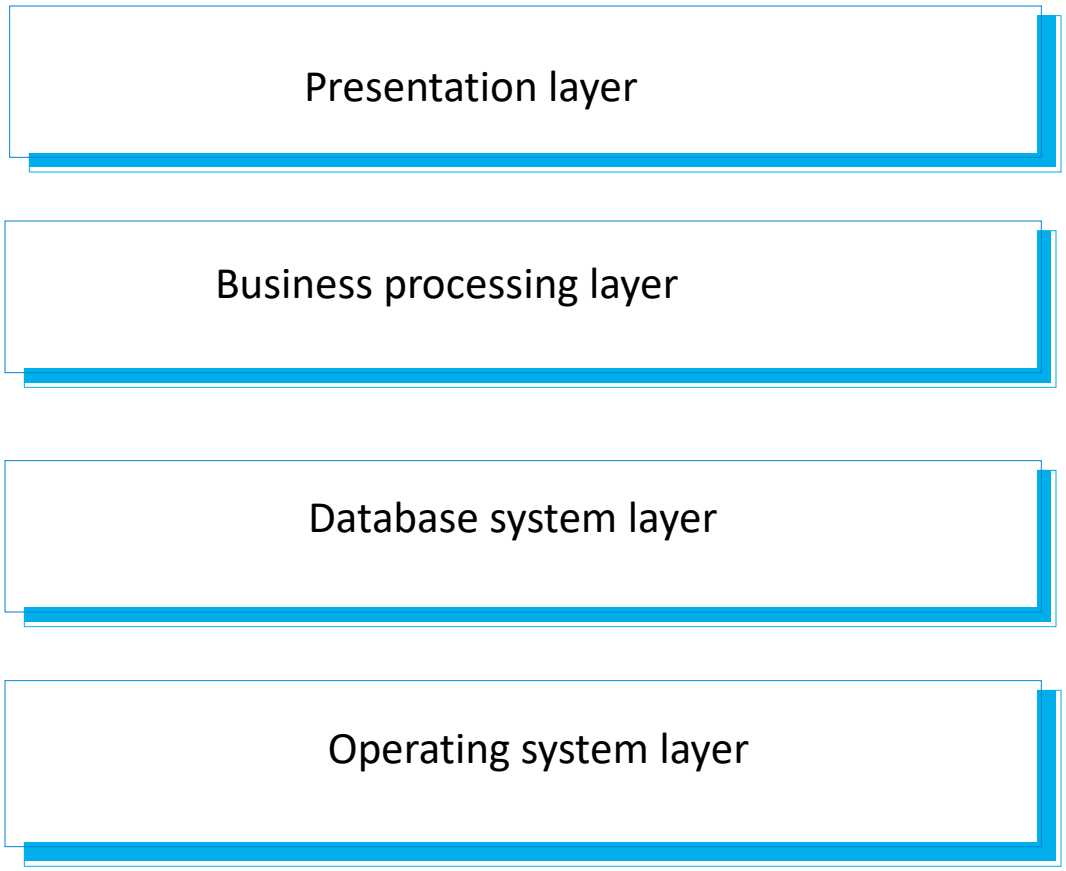
- **Disadvantages**
  - No shared data model so sub-systems use different data organization
  - Data interchange may be inefficient
  - Redundant management in each server
  - No central register of names and services

Layered Style

- Organize the system into a set of layers
  - each layer provides a set of services
- Support the incremental development of different layers
  - When a layer interface changes, only the adjacent layer is affected



Version Management System



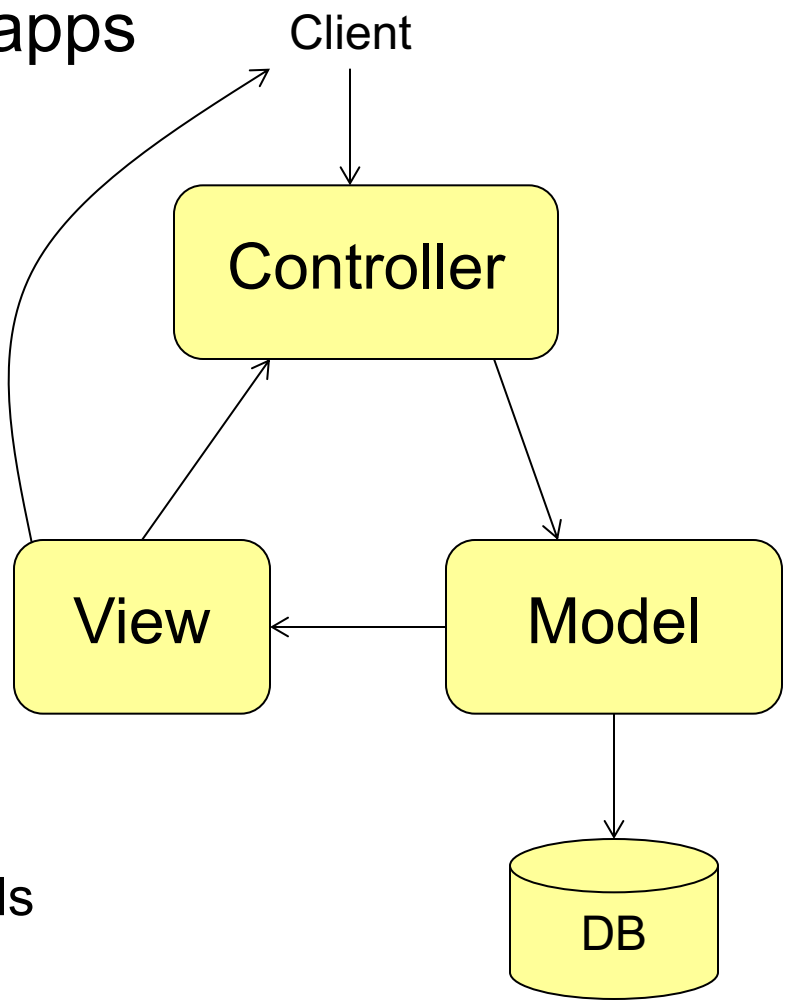
Layered Style

- Advantages
  - Separation of concerns
  - Reuse
  - Reduce impact of changes on user interface
- Disadvantages
  - Inflexible in communications among layers
  - Performance problem due to going through many layers

Model-View-Controller (MVC)

- Originally introduced for UI apps
- Now popular for Web apps

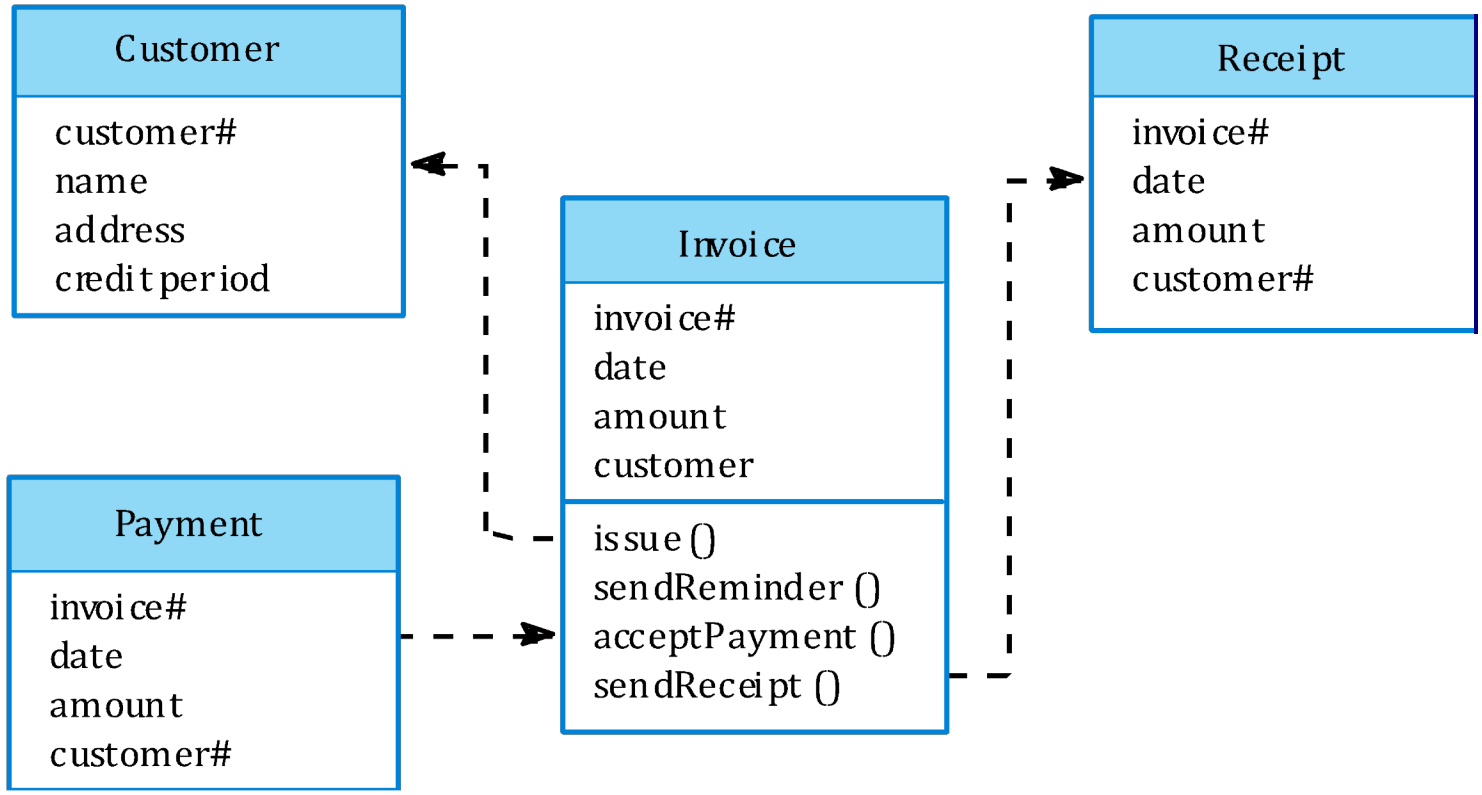
- Components
  - Model
    - Data, logic processing
  - View
    - Presentation
  - Controller
    - Accepting commands, inputs
    - Converting, passing commands



Object or Abstract Data Type Style

- Structure the system into a set of loosely coupled objects with well-defined interfaces
- Object-oriented decomposition is concerned with identifying
  - Object classes
  - Object class's attributes and operations

Invoice Processing System



Object Style

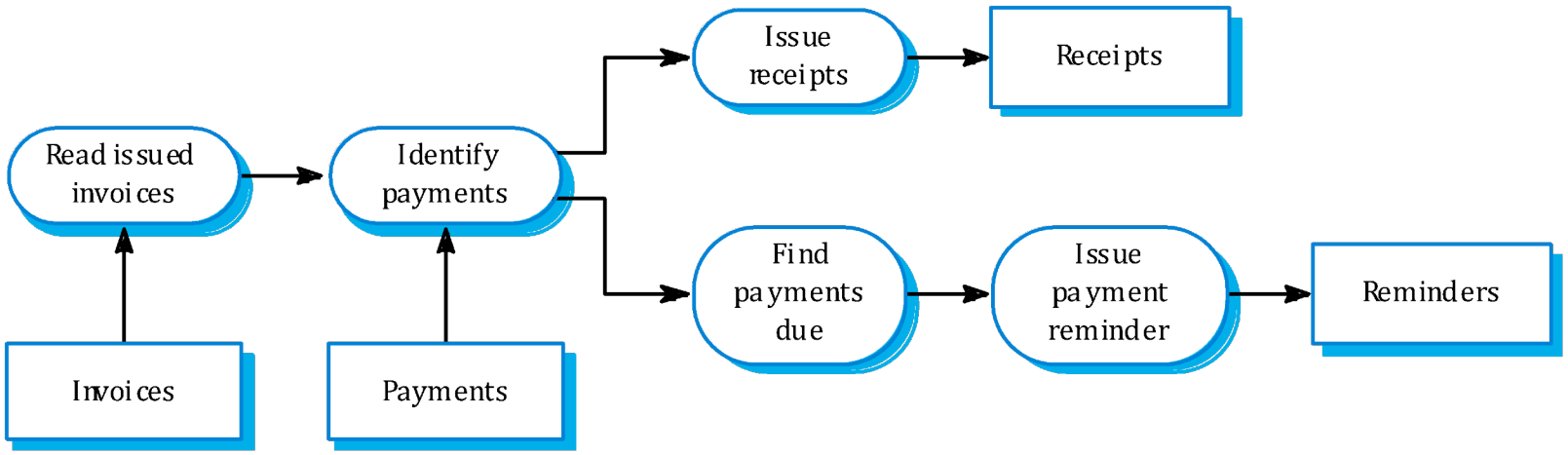
- Advantages
  - Objects are loosely coupled → their implementation can be modified without affecting other objects
  - Objects may reflect real-world entities
  - OO implementation languages are widely used

- Disadvantages
  - object interface changes may cause problems
  - complex entities may be hard to represent as objects

Pipes and Filters Style

- System decomposed into a series of computational components or **filters**
- Filters process data independently
- Data travels through filters via **pipes**

Invoice Processing System



Pipes and Filters Style

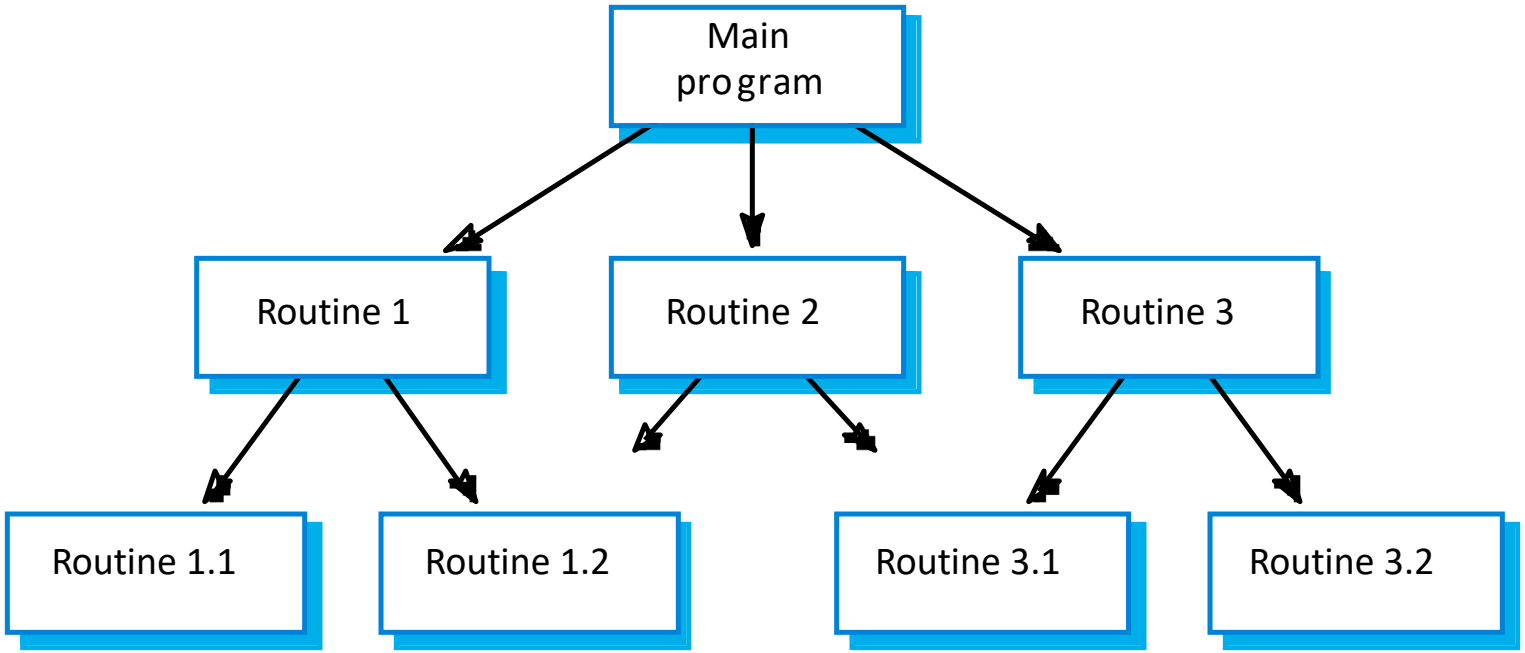
- Advantages
  - Supports transformation reuse
  - Intuitive organization for stakeholder communication
  - Easy to add new transformations
  - Relatively simple to implement as either a concurrent or sequential system

- Disadvantages
  - Requires a common format for data transfer along the pipeline
  - Difficult to support event-based interaction

Main Program with Subroutines

- A control sub-system takes responsibility for managing the execution of other sub-systems
- Top-down subroutine model
  - control starts at the top of a subroutine hierarchy and moves downwards
- Applicable to sequential systems

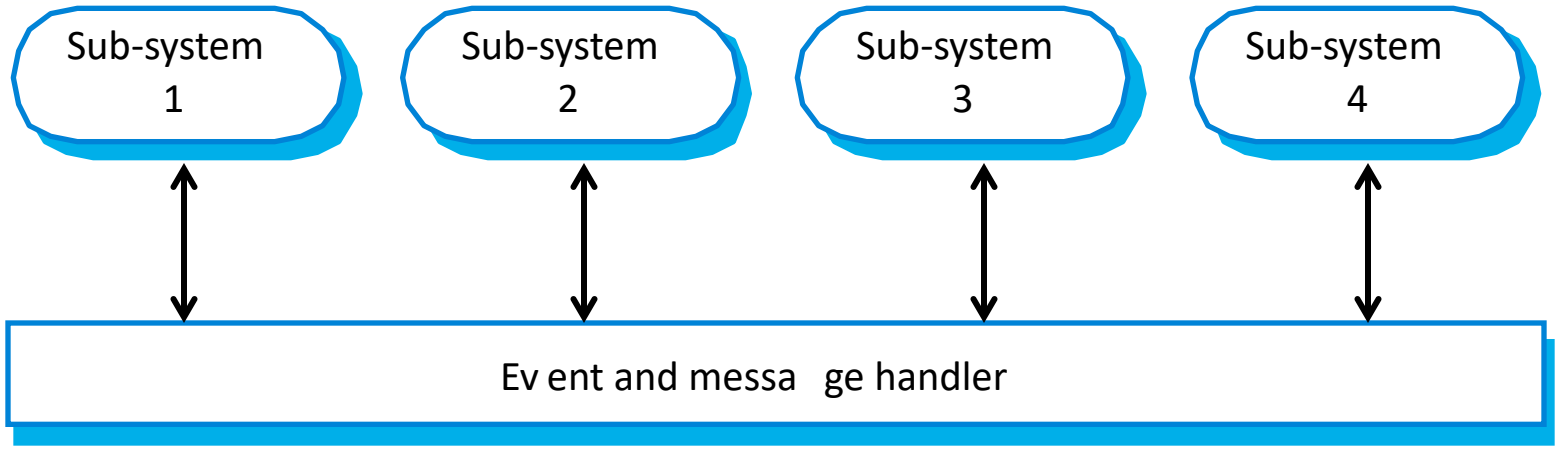
Main Program with Subroutines - 2



Broadcast Style

- Effective in integrating sub-systems on different computers in a network
- Sub-systems register an interest in specific events
  - When events occur, control is transferred to the sub-system which can handle the event
- Control policy is not embedded in the event and message handler
  - Sub-systems decide on events of interest to them
- Disadvantage
  - sub-systems don’t know if or when an event will be handled

Selective Broadcasting



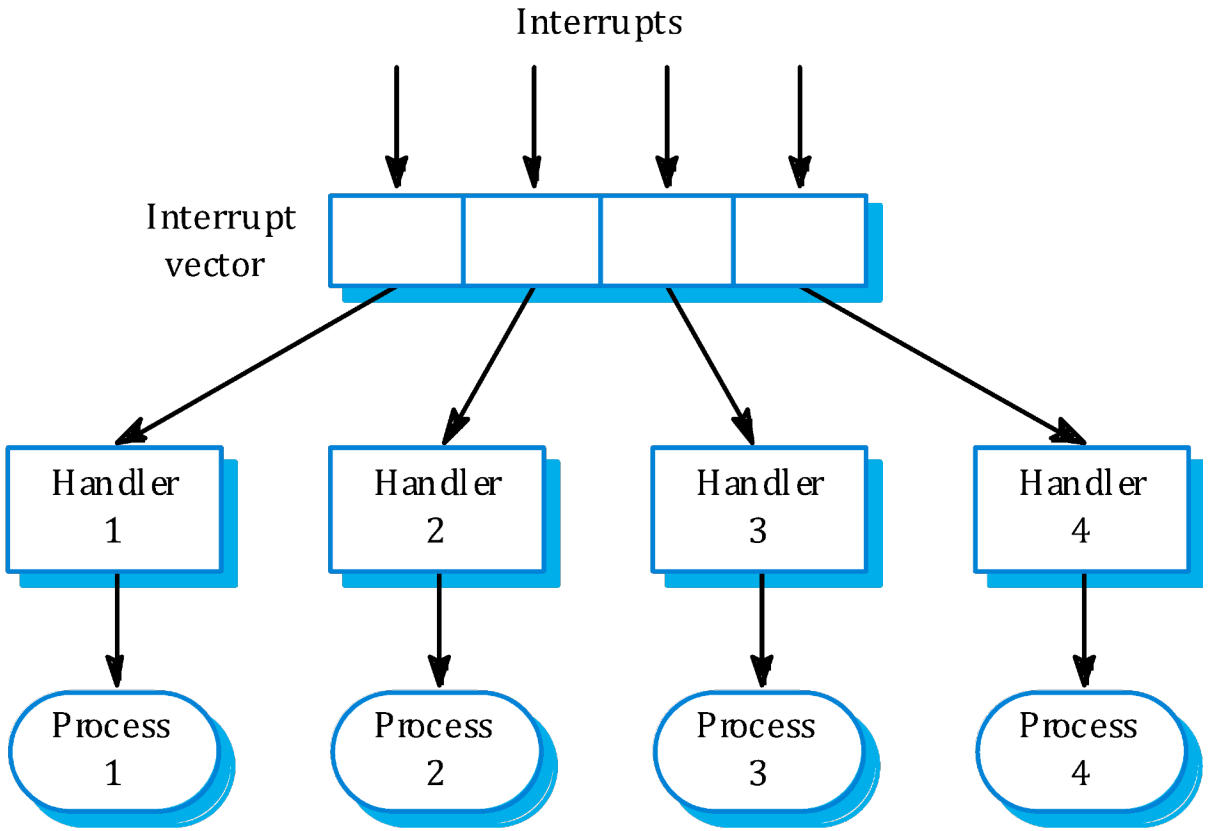
Interrupt-driven Style

- Used in real-time systems where fast response to an event is essential
- There are interrupt types with a handler defined for each type

- Each interrupt type is associated with a memory location and a hardware switch

- Disadvantages
  - Allows fast response but complex to program and difficult to validate

Interrupt-driven Control



Key Points

- Software architecture is the fundamental framework for structuring the system
- Architectural design decisions: decisions on the application architecture
- Different architectural styles are decided during the architectural design

Key Points

- Different architectural models may be produced during the design process
- Architecture Attributes
  - Performance
    - Localise operations to minimise sub-system communication
  - Security
    - Use a layered architecture with critical assets in inner layers
  - Safety
    - Isolate safety-critical components
  - Availability
    - Include redundant components in the architecture
  - Maintainability
    - Use fine-grain, self-contained components
  - Others