Faculty of ECMS / School of Computer Science

Software Engineering & Project
Architectural Design

Architectural Design

Lecture 11

Chapter 11 (6 in Edition 9)
in the course text book

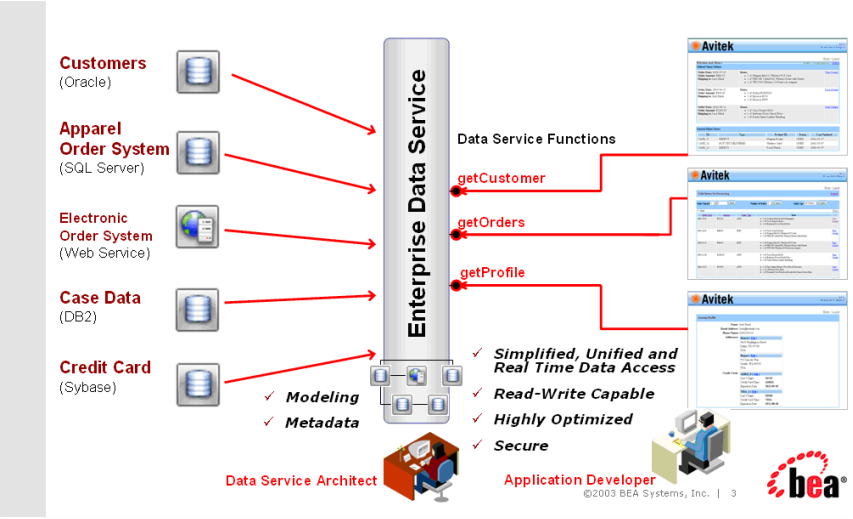
Outline

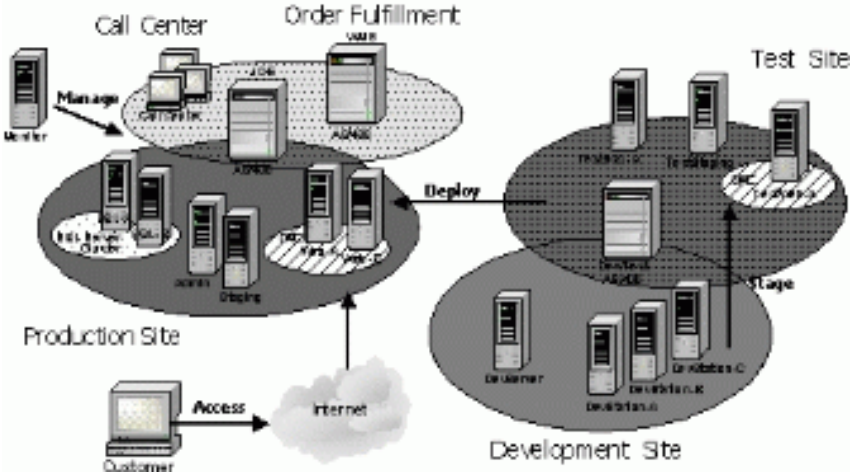
- Architectural design: what and why
- System organisation
- Decomposition styles
- Control styles

What is software architecture design

- The design process for identifying the sub-systems making
- The framework for sub-system control and communication

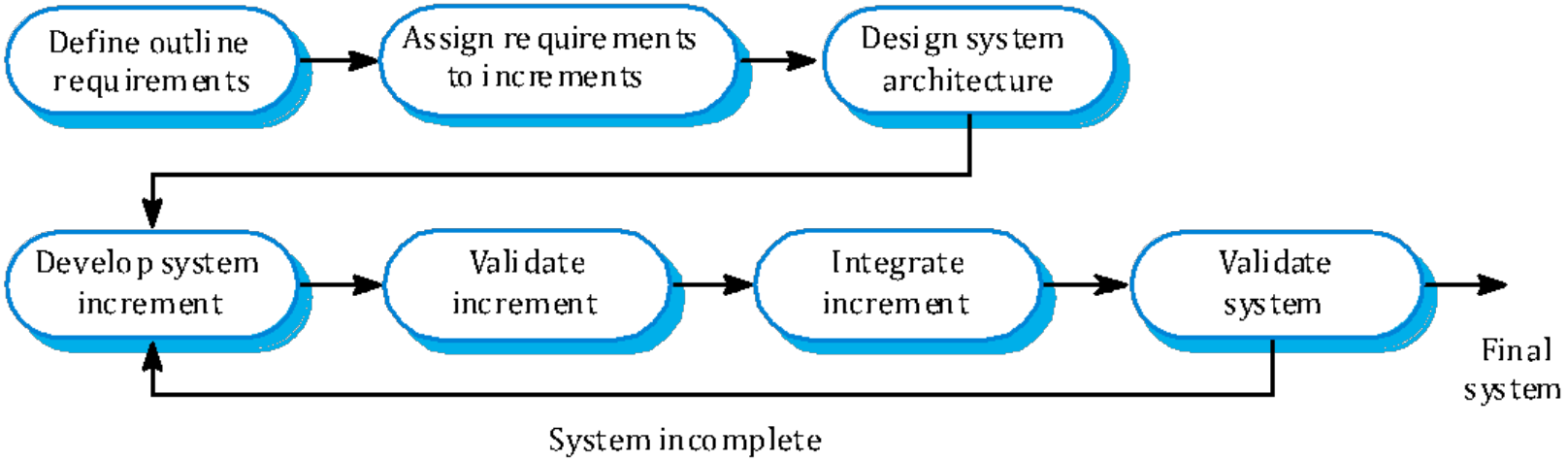
- The output of this design process is a description of the software architecture
  - Essential for large systems





What is software architecture design (Cont.)

- An ***early stage*** of the system design process
- Represents the link between the requirements engineering processes and the design processes
- It involves identifying major system components and their communications
- Should also complement projects process model



Advantages of explicit architecture

- Stakeholder communication
  - Architecture may be used as a focus of discussion by system stakeholders (e.g., clients, developers, managers)
- System analysis
  - Analysis on whether the system can meet its non-functional requirements (e.g., performance, reliability)
- Large-scale reuse
  - The architecture may be reusable across a range of systems
- Architecture design forces software developers to consider key design aspects at early stage

Choosing an architecture

- Functional requirements: statements of services that the system should provide, for example
  - Robot shall detect obstacles and return their locations
  - Host GUI shall display obstacle positions in the map

- Non-functional requirements: constraints on the services/functions offered by the system, for example
  - Performance (e.g., how fast the robot can complete the job?)
  - Reliability (e.g., how accurate the detected obstacle positions are?)
  - .....

Choosing an architecture (Cont.)

- Performance
  - Localise critical operations and minimise communications
  - Use large rather than fine-grain components
- Security
  - Use a layered architecture with critical assets in the inner layers
- Safety
  - Localise 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
  - Development of a framework

Architectural conflicts

- Using large-grain components improves performance; using fine-grain components improves maintainability
- What if both performance and maintainability are important system requirements in a system?
- Compromise solution should be applied to satisfy multiple non-functional requirements. For different parts of a system, different architecture styles could be used.

Architectural Design Decisions

- Architectural design is a creative process

- The process differs (greatly!) depending on the type of system being developed
  - The background and experience of the system architect
  - The specific requirements of the system, etc

- System architect has to make several fundamental decisions that profoundly affect the system and its development process
  - There are no one-fits-all solutions

Architectural Design Decisions (Cont.)

- Is there a generic application architecture that can be used?
- How will the system be distributed?
- What architectural styles are appropriate?
- What approach will be used to structure the system?
- 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?
- How does it work with the projects quality management, process model, team recourses?

Architectural style: a concept

- Is a pattern of system organisation (e.g., client-server, layered, component-reuse)

- An awareness of these styles, their applications, their strengths and weakness, can simplify the problem of defining system architectures
  - Which is very important!

- There is no one architecture that fits all!
  - You do have the ability to combine and create new architectures that may be specific to the project

Outline

- Architectural design: what and why
- System organisation
- Decomposition styles
- Control styles

System Organisation

- Reflects the basic strategy that is used to structure a system

- Three organisational styles are widely used:
  1. A shared data repository style;
  2. A shared services and servers style;
  3. An abstract machine or layered style.

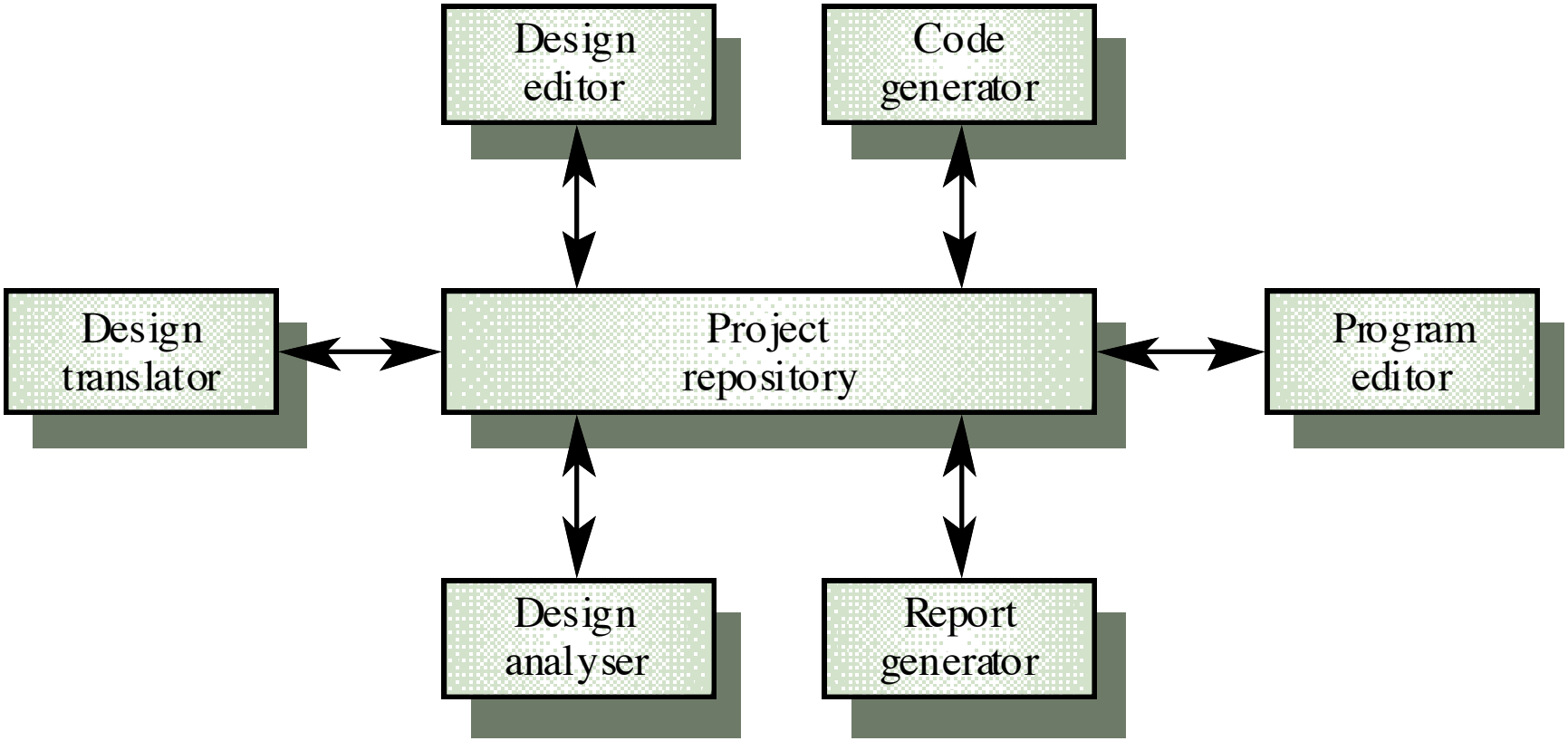
- These styles can be used separately or together

1. The repository model

- When large amounts of data are to be shared, the repository model of sharing is most commonly used
- Data generated from one sub-system are used by another

- Sub-systems must exchange data. This may be done in two ways:
  - Shared data is held in a central database or repository and may be accessed by all sub-systems;
  - Each sub-system maintains its own database and passes data explicitly to other sub-systems.

1. The repository model Example: IDE architecture



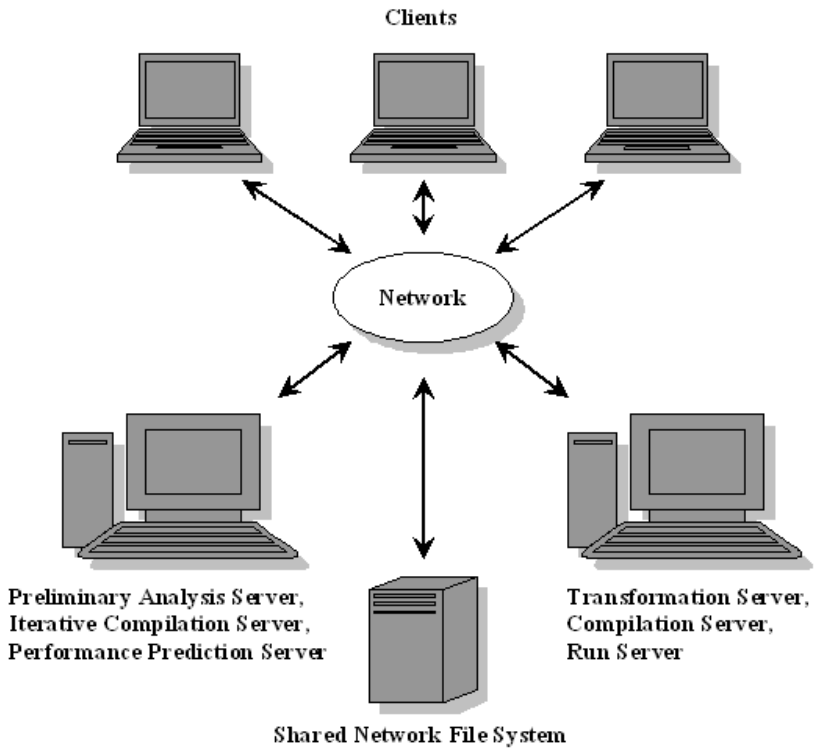
1. The repository model pros vs. cons

- Advantages
  - Efficient way to share large amounts of data among sub-systems
  - Sub-systems need not be concerned with how data is produced
  - Centralised management e.g. backup, security, auditing, analysis, etc
  - Sharing model is published as the repository schema (easy to integrate new models)

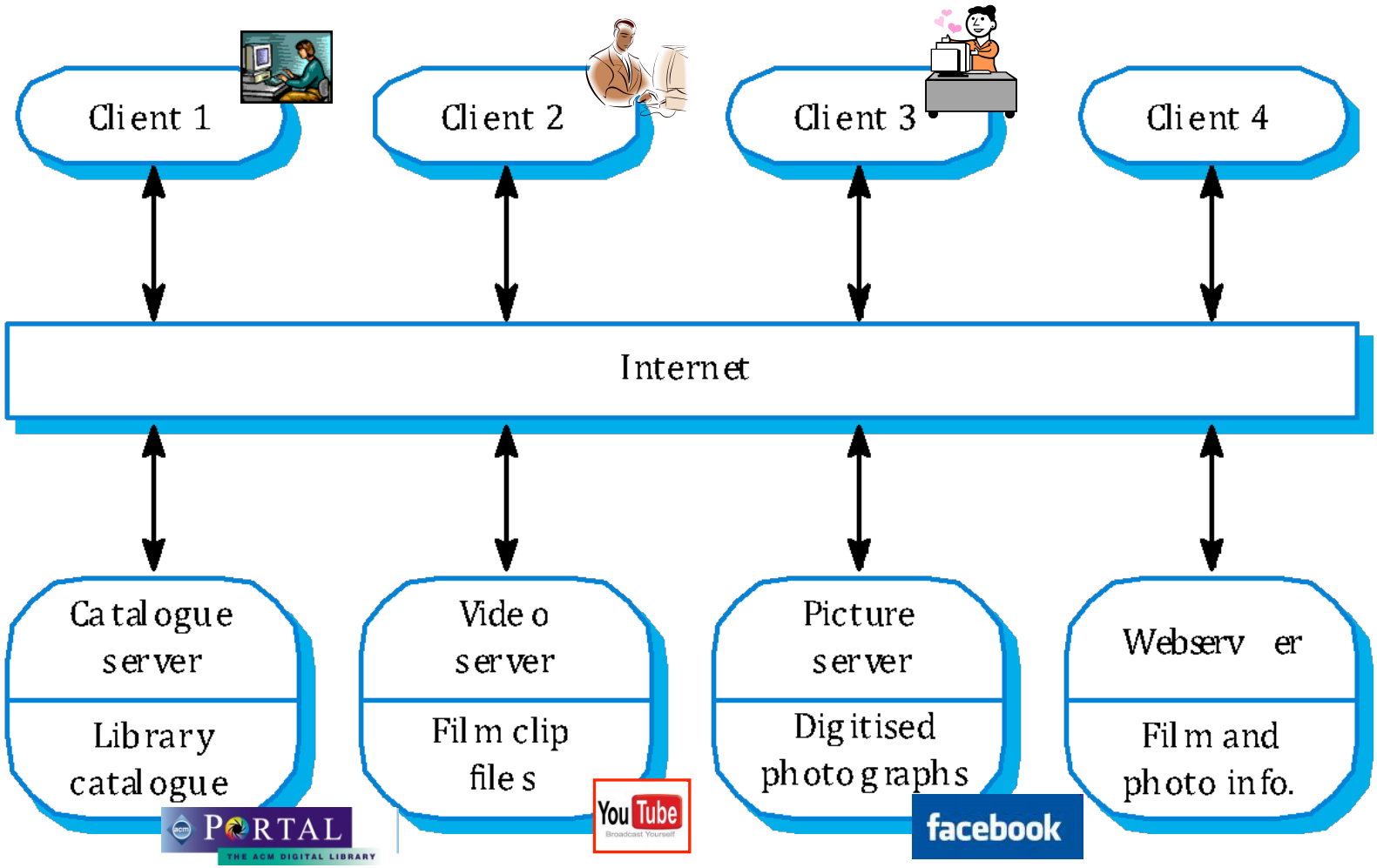
- Disadvantages
  - Sub-systems must agree on a repository data model. Inevitably a compromise
  - No scope for specific management policies
  - Difficult to distribute efficiently
  - Data evolution is difficult and expensive
    - Managing versions and stale data

2. The client-server model

- A distributed system model which shows how data and processing is distributed across a range of components
- Set of stand-alone *servers* which provide specific services such as printing, data management, online services, etc
- Set of *clients* which call on these services
- *Network* which allows clients to access servers



2. The client-server model example: online services



2. The client-server model pros vs. cons

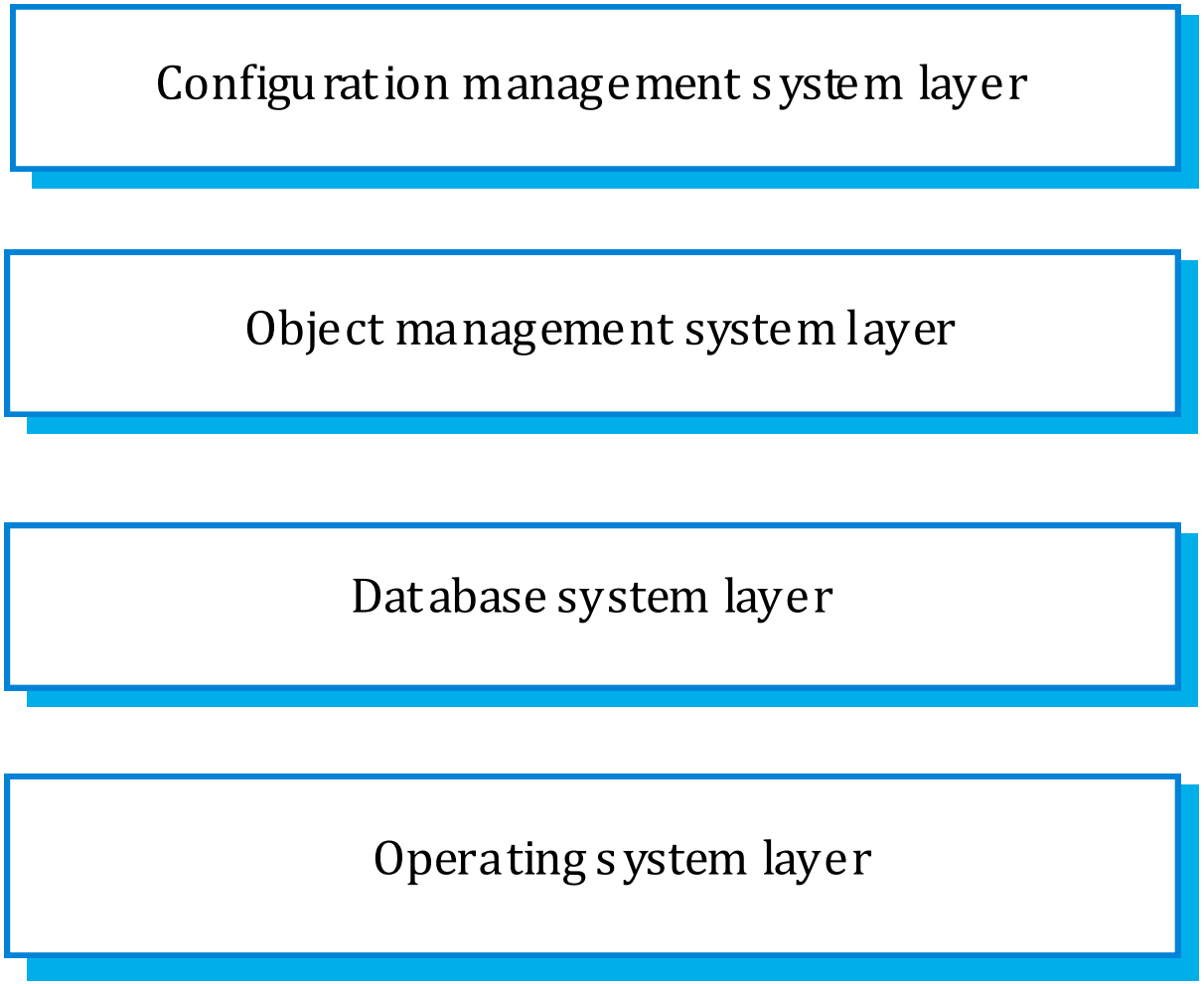
- Advantages
  - Distribution of data is straightforward;
  - Makes effective use of networked systems. May require cheaper hardware;
  - Easy to add new servers or upgrade existing servers.

- Disadvantages
  - No shared data model so sub-systems use different data organisation. Data interchange may be inefficient or incompatible overtime;
  - Redundant management in each server;
  - No central register of names and services - it may be hard to find out what servers and services are available.

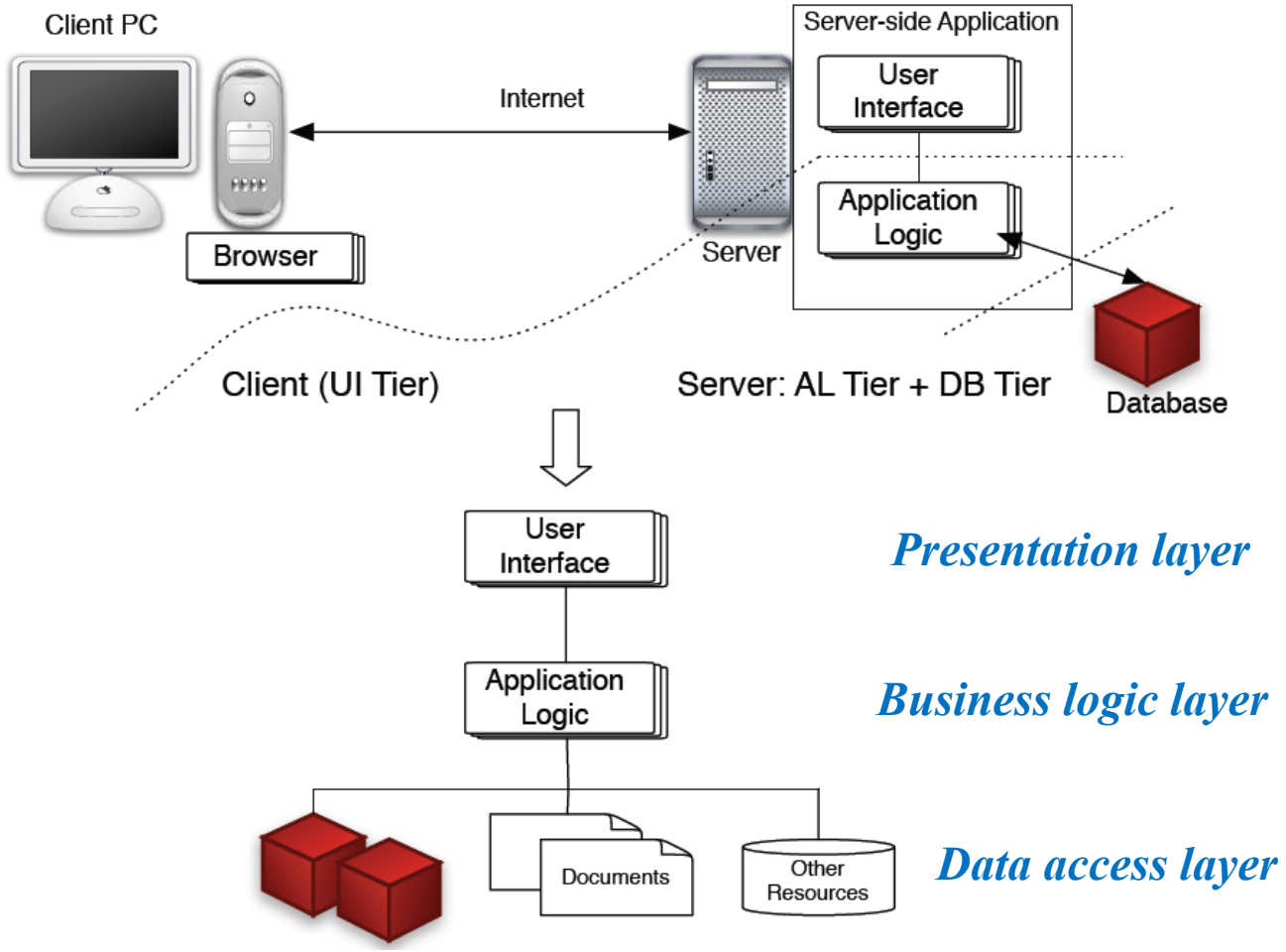
3. The layered model

- Sometimes known as the Abstract Machine model
- Used to model the interfacing of sub-systems
- Organises the system into a set of layers (or abstract machines) each of which provides a set of services
- Supports the *incremental* development of sub-systems in different layers. When a layer interface changes, only the adjacent layer is affected
  - Wrapper pattern
- *Performance could be a problem*

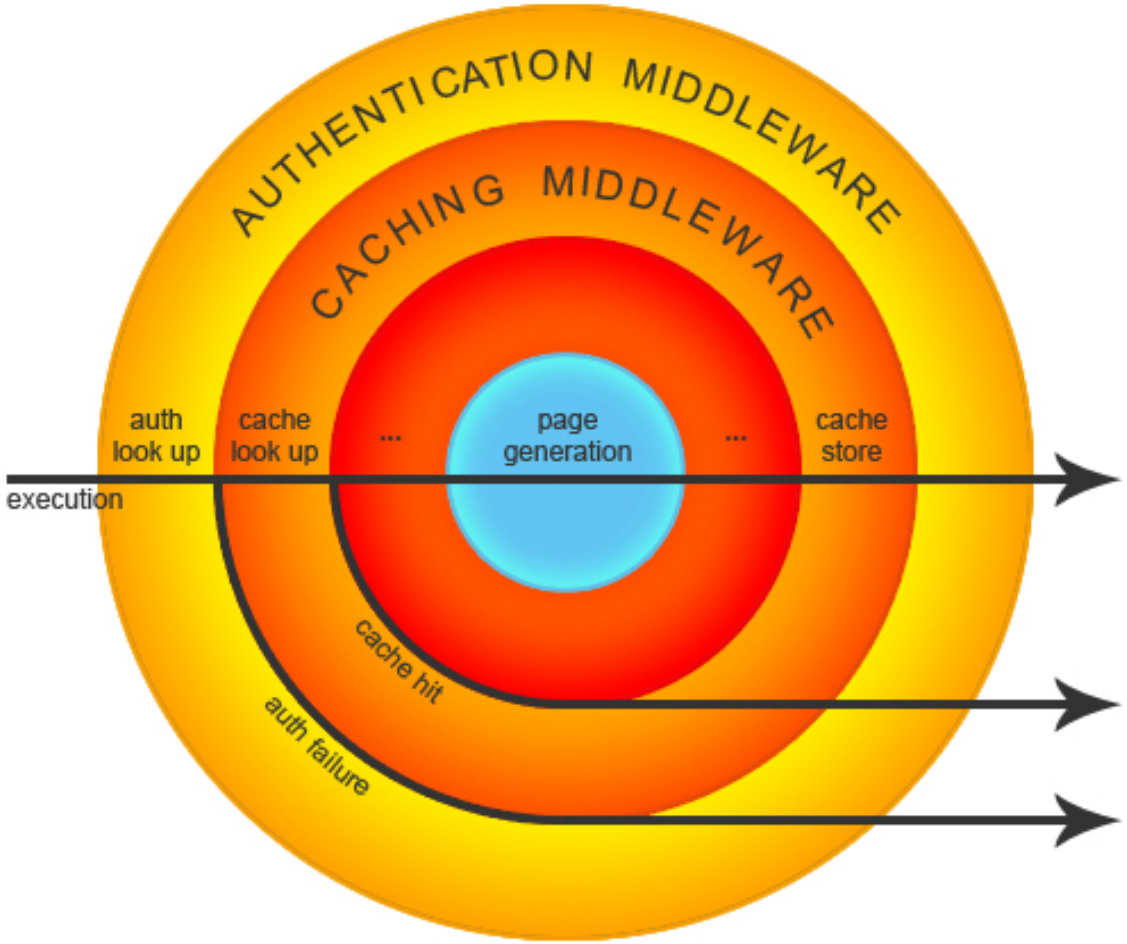
3. The layered model example: version management system



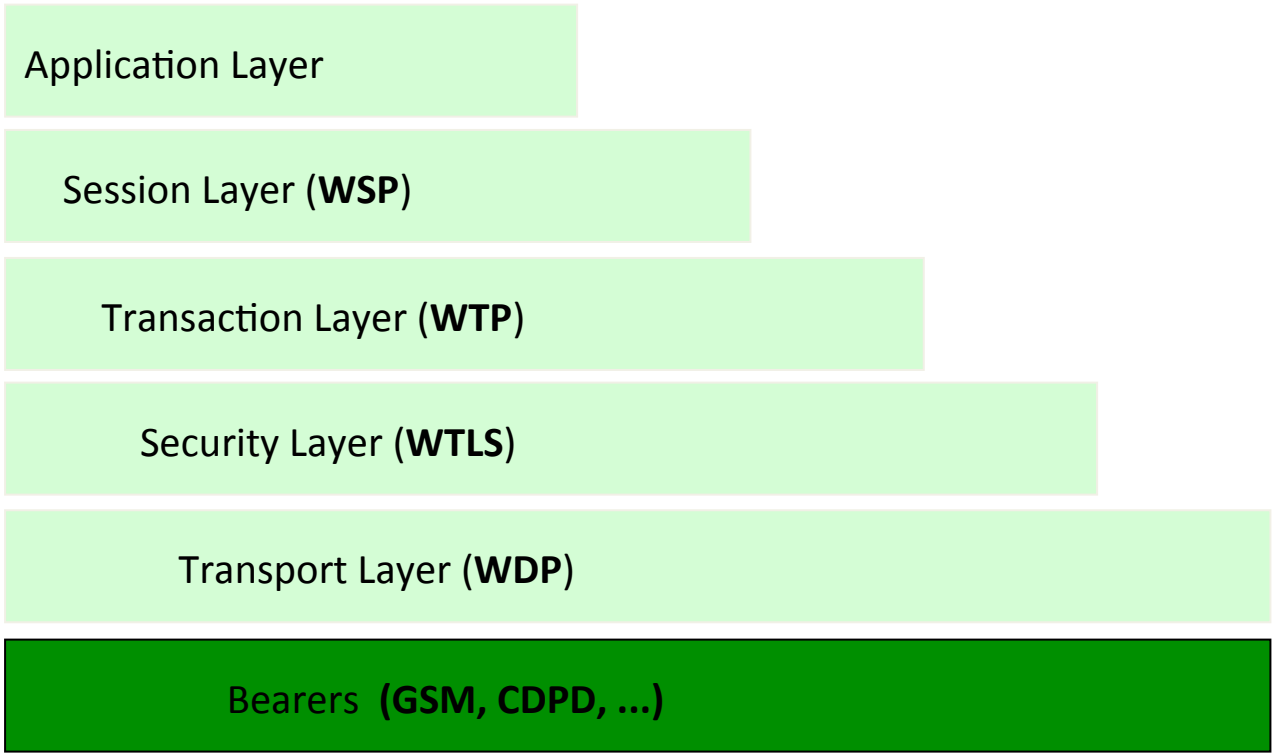
3. The layered model example: web application (Cont.)



3. The layered model example: web application stack (Cont.)



3. The layered model example: wireless access protocol (Cont.)



Outline

- Architectural design: what and why
- System organisation
- Decomposition styles
- Control styles

Modular decomposition styles

- Styles of decomposing sub-systems into modules

- No rigid distinction between system organisation and modular decomposition
  - A sub-system is a system in its own right whose operation is independent of the services provided by other sub-systems
  - A module is a system component that provides services to other components but would not normally be considered as a separate system

Modular decomposition styles (Cont.)

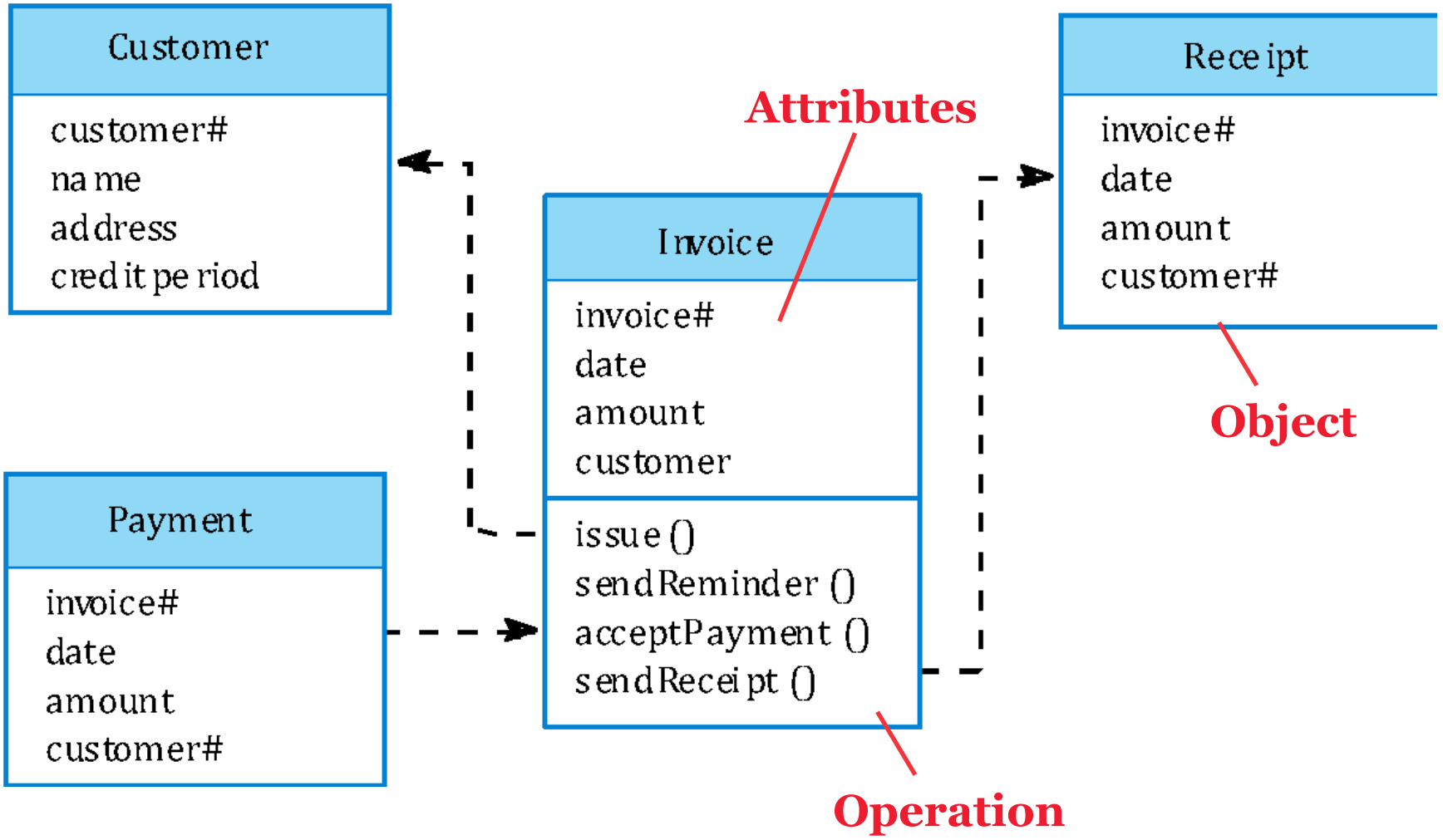
- An ***object-oriented decomposition*** model where the system is decomposed into interacting objects

- A ***function-oriented pipelining*** (data-flow) model where the system is decomposed into functional modules which transform inputs to outputs

1. Object-oriented models

- Structure the system into a set of loosely coupled objects with well-defined interfaces
- Object-oriented decomposition is concerned with identifying object classes, their attributes and operations
- When implemented, objects are created from these classes and some control model used to coordinate object operations

1. Object-oriented models example: Invoice processing system



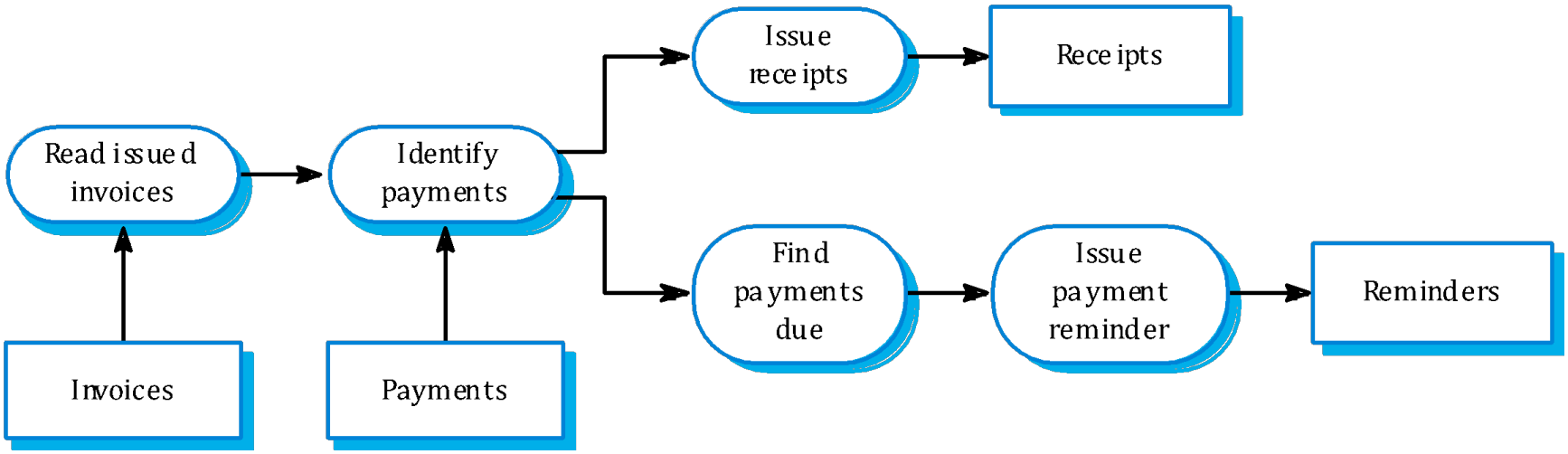
1. Object-oriented models pros vs. cons

- Objects are loosely coupled so their implementation can be modified without affecting other objects
- The objects may reflect real-world entities, easy to understand
- Can be reused
- OO implementation languages are widely used
- However, object interface changes may cause problems and complex entities may be hard to represent as objects

2. Function-oriented pipelining

- Functional transformations process their inputs to produce outputs
- May be referred to as a pipe and filter model (as in UNIX shell)
- Variants of this approach are very common. When transformations are sequential, this is a batch sequential model which is extensively used in data processing systems

2. Function-oriented pipelining example: invoice processing system



2. Function-oriented pipelining pros vs. cons

- Advantages:
  - Supports transformation reuse
  - Intuitive organisation for stakeholder communication (easy to understand, people think their work in terms of input/output processing)
  - 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 and difficult to support event-based interaction
  - Not really suitable for interactive systems (e.g., games)

Outline

- Architectural design: what and why
- System organisation
- Decomposition styles
- Control styles

Control styles

- *What control strategy should be used?*

- Are concerned with the control flow between sub-systems. Sub-systems must be controlled so that their services are delivered to the right place at the right time

- Two generic control styles:
  - Centralised control
    - One sub-system has overall responsibility for control and starts and stops other sub-systems
  - Event-based control
    - Each sub-system can respond to externally generated events from other sub-systems or the system's environment

1. Centralised Control

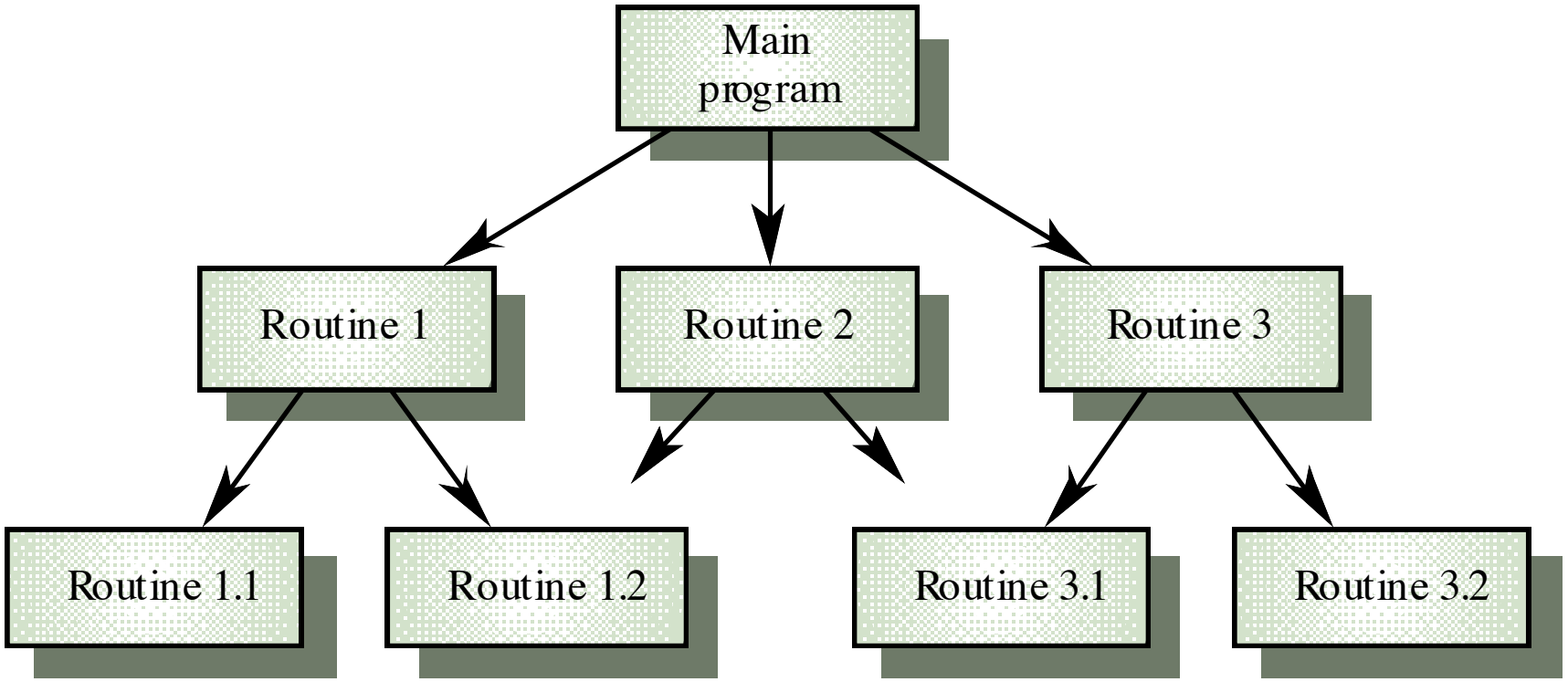
- A control sub-system (called system controller) takes responsibility for managing the execution of other sub-systems

- Call-return model
  - Top-down subroutine model where control starts at the top of a subroutine hierarchy and moves downwards. Applicable to ***sequential*** **systems**

- Manager model
  - Applicable to ***concurrent*** **systems**. One system component controls the stopping, starting and coordination of other system processes. Can be implemented in sequential systems as a case statement

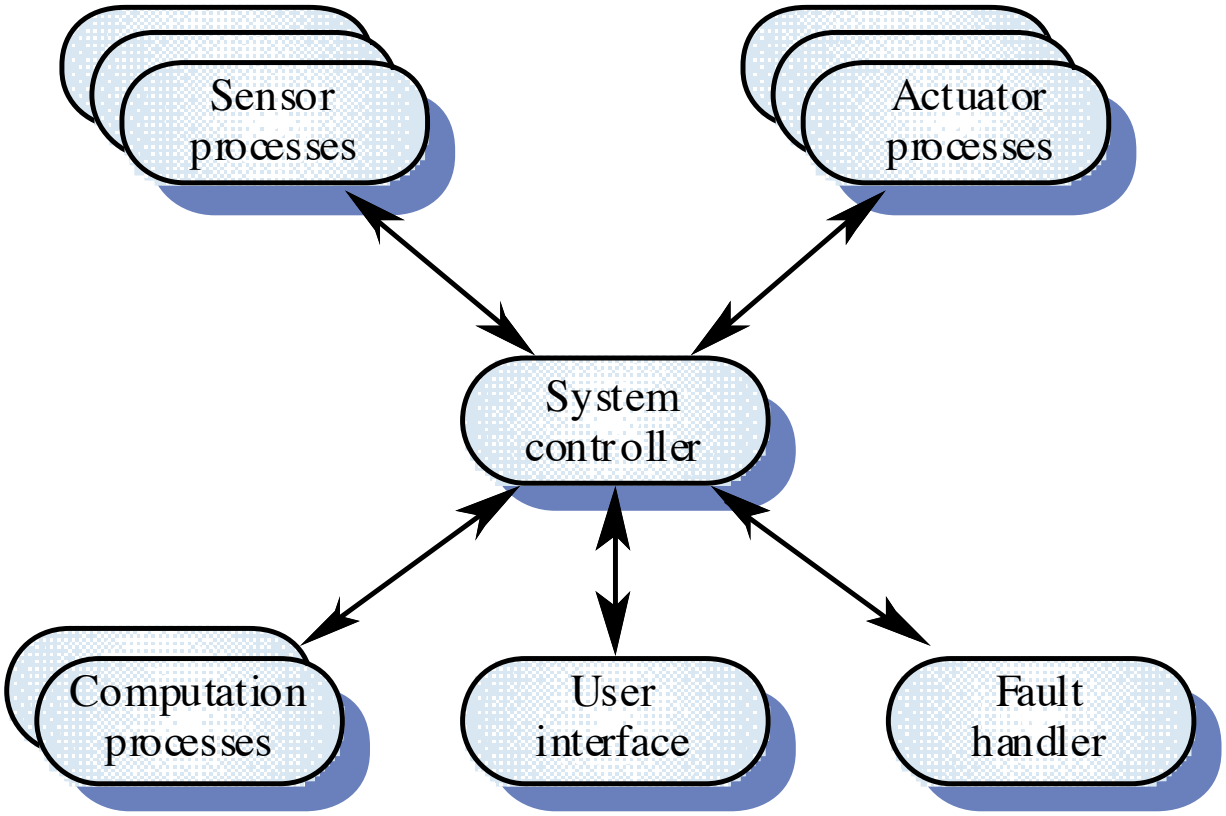
1. Centralised Control example: call-return model

- Simple to analyse control flows
- Hard to handle exceptions



1. Centralised Control example: manager model

- Often used in “soft” real-time systems



2. Event-driven systems

- Driven by externally generated events where the timing of the event is out of the control of the sub-systems which process the event

- Two principal event-driven models
  - ***Broadcast models***. An event is broadcast to all sub-systems. Any sub-system which can handle the event may do so

  - ***Interrupt-driven models***. Used in real-time systems where interrupts are detected by an interrupt handler and passed to some other component for processing

Broadcast model

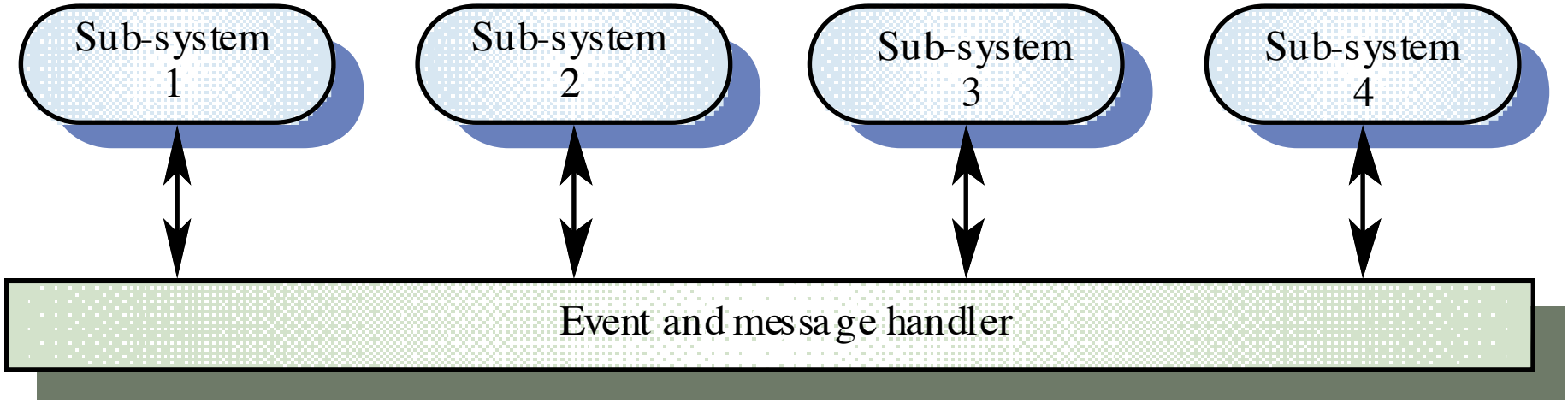
- Effective in integrating sub-systems on different computers in a network

- Sub-systems register an interest in specific events. When these 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

Selective broadcasting

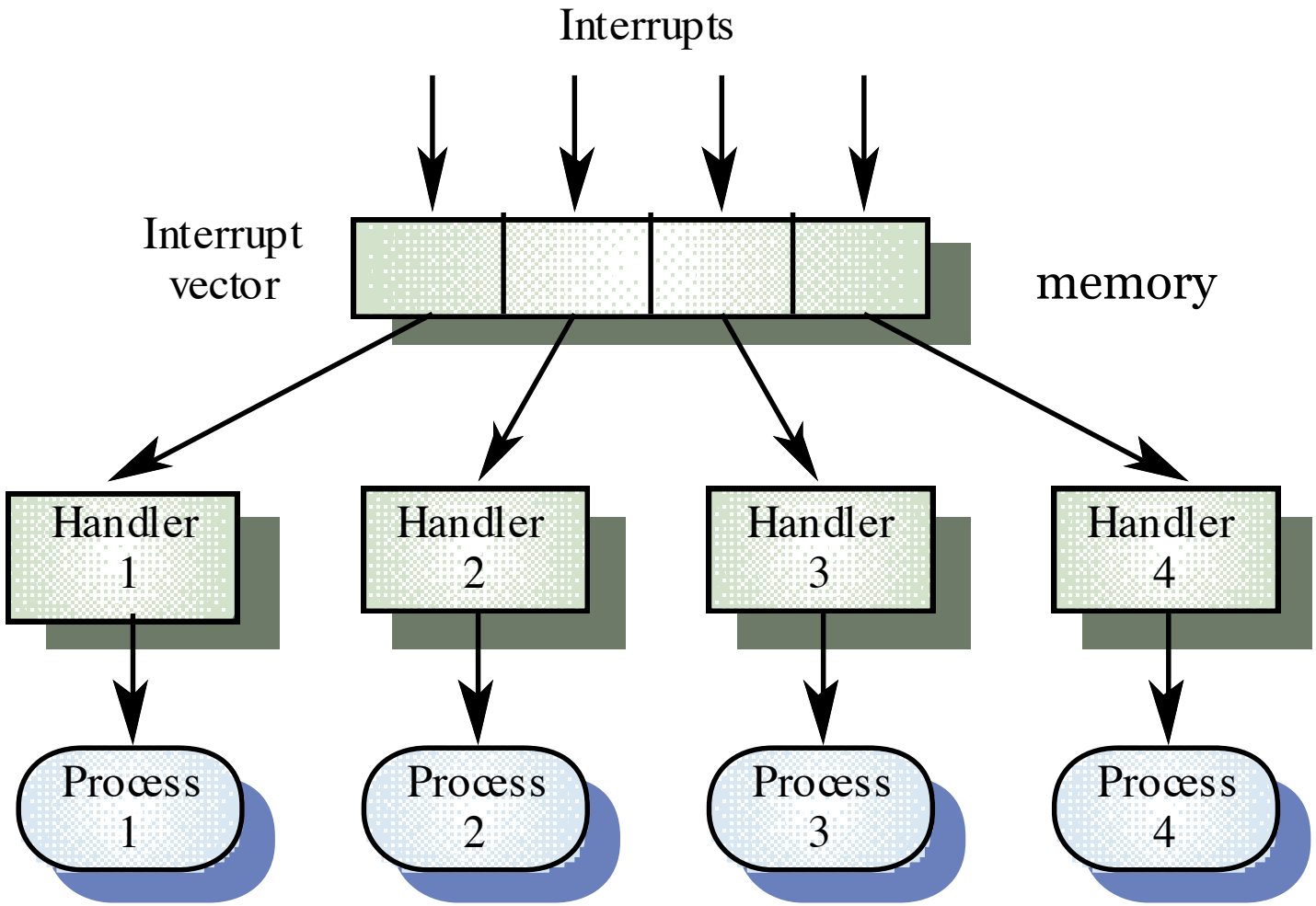
- Sub-systems can be implemented on distributed machines; easy to integrate new sub-systems
- However, sub-systems don't know if or when an event will be handled, particularly when different sub-systems register for same events



Interrupt-Driven Systems

- Used in real-time systems where fast response to an event is essential.
- There are known interrupt types with a handler defined for each type.
- Each type is associated with a memory location and a hardware switch causes transfer to its handler.
- Allows very fast response but complex to program and difficult to change.

Interrupt-Driven Systems example: interrupt-driven control



Key Points Revisited

- The software architecture is the fundamental framework for ***structuring the system***
- ***Architectural design decisions*** include decisions on the application architecture, the distribution and the architectural styles to be used
- Different architectural models such as a ***structural model***, a ***control model*** and a ***decomposition model*** may be developed
- System organisational models include ***repository models***, ***client-server models*** and ***abstract machine models***

Key Points Revisited

- Modular decomposition models include ***object*** models and ***pipelining*** models
- Control models include ***centralised control*** and ***event-driven*** models