SOFTWARE ARCHITECTURAL DESIGN, DISTRIBUTED SYSTEMS ARCHITECTURES

Topic 9

Josephine Magu MSC UON

ARCHITECTURAL DESIGN - ESTABLISHING THE OVERALL STRUCTURE OF A SOFTWARE SYSTEM

Topics covered:

- System structuring
- Control models
- Modular decomposition

Architectural Design

- Multiprocessor architectures
- Client-server architectures
- Distributed object architectures

Distributed Systems Architectures

SOFTWARE ARCHITECTURE

- The design process for identifying the sub-systems making up a system and the framework for sub-system control and communication is *architectural design*
- The output of this design process is a description of the *software architecture*

ARCHITECTURAL DESIGN

- An early stage of the system design process
- Represents the link between specification and design processes
- Often carried out in parallel with some specification activities
- It involves identifying major system components and their communications

ARCHITECTURAL DESIGN PROCESS

- **System structuring**
  - The system is decomposed into several principal sub-systems and communications between these sub-systems are identified
- **Control modelling**
  - A model of the control relationships between the different parts of the system is established
- **Modular decomposition**
  - The identified sub-systems are decomposed into modules

SUB-SYSTEMS AND MODULES

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

ARCHITECTURAL MODELS

- Different architectural models may be produced during the design process
- Each model presents different perspectives on the architecture:
  - Static structural model
  - Dynamic process model
  - Interface model
  - Relationships model

ARCHITECTURAL MODELS

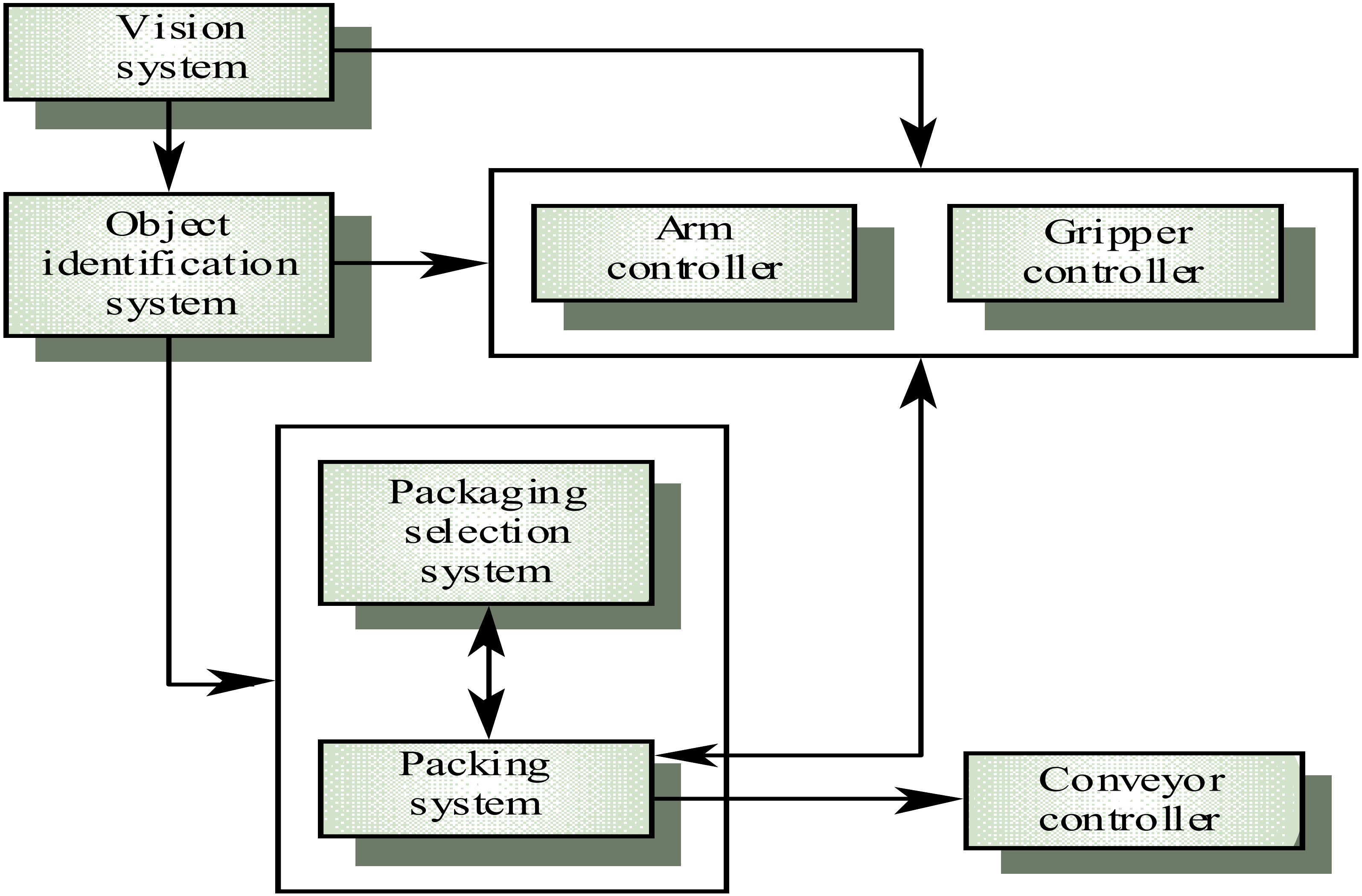
- **Static structural model** that shows the major system components
- **Dynamic process model** that shows the process structure of the system
- **Interface model** that defines sub-system interfaces
- **Relationships model** such as a data-flow model

SYSTEM STRUCTURING

Concerned with decomposing the system into interacting sub-systems

- The architectural design is normally expressed as a block diagram presenting an overview of the system structure
  - (More specific models showing how sub-systems share data, are distributed and interface with each other may also be developed)

PACKING ROBOT CONTROL SYSTEM



THE REPOSITORY MODEL

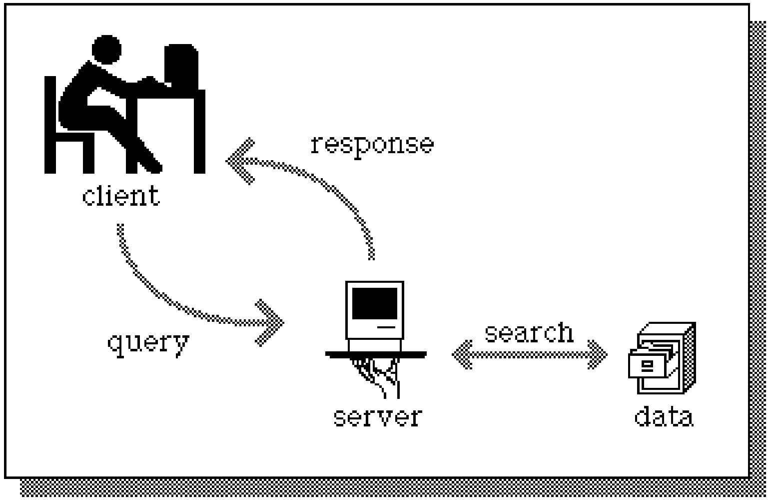
- **Sub-systems must exchange data**. This may be done in two ways:
  - Shared data is held in a central database or repository and may be accessed by all sub-systems
  - Each sub-system maintains its own database and passes data explicitly to other sub-systems
- When large amounts of data are to be shared, the repository model of sharing is most commonly used (WHY???)

REPOSITORY MODEL CHARACTERISTICS

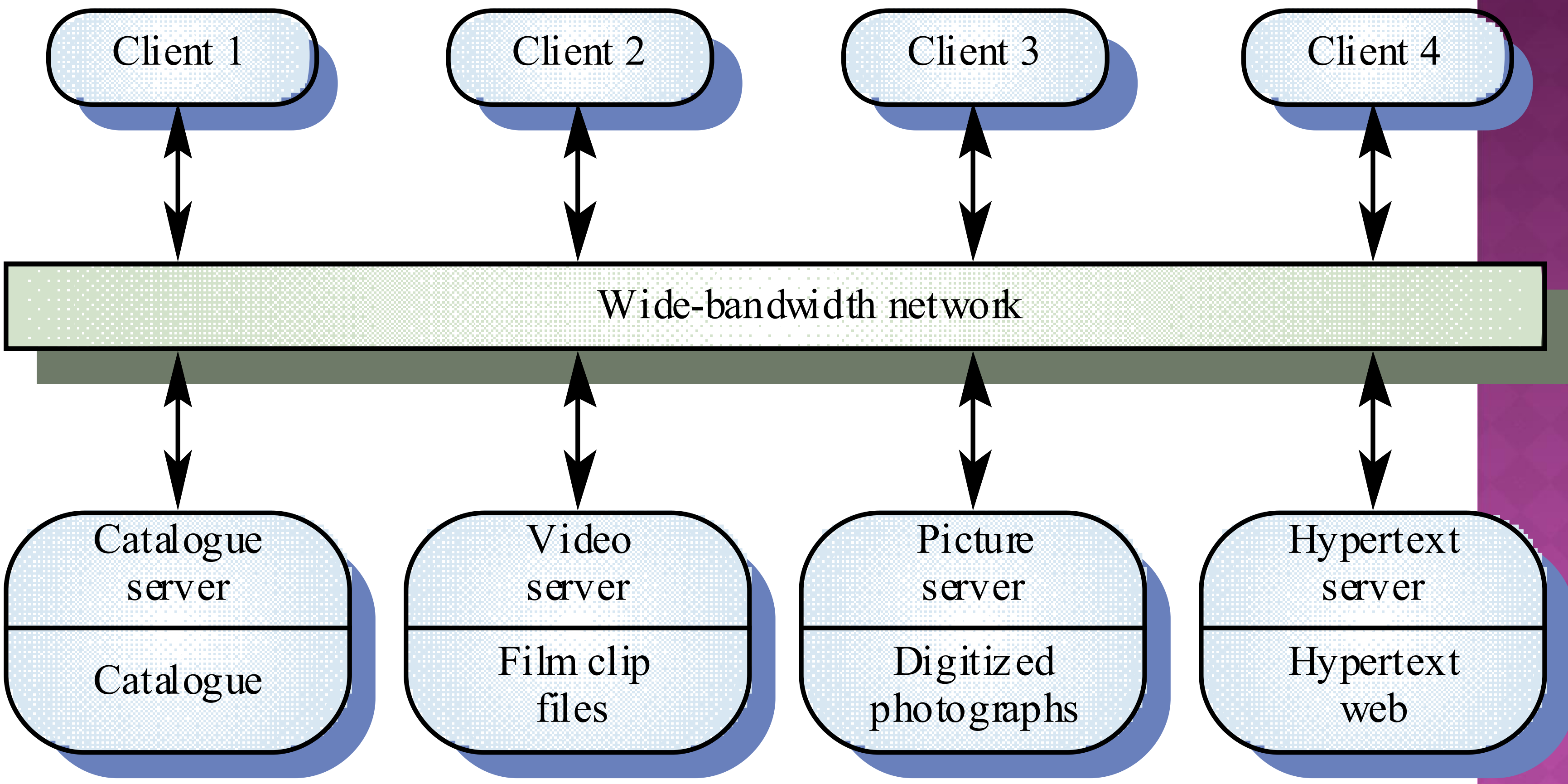
- Advantages
  - Efficient way to share large amounts of data
  - Sub-systems need not be concerned with how data is produced
  - Centralised management e.g. backup, security, etc.
  - Sharing model is published as the repository schema
- Disadvantages
  - Sub-systems must agree on a repository data model. Inevitably a compromise
  - Data evolution is difficult and expensive
  - No scope for specific management policies
  - Difficult to distribute efficiently

CLIENT-SERVER ARCHITECTURE

- 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, etc.
- **Set of clients** which call on these services
- **Network** which allows clients to access servers



FILM AND PICTURE LIBRARY



CLIENT-SERVER CHARACTERISTICS

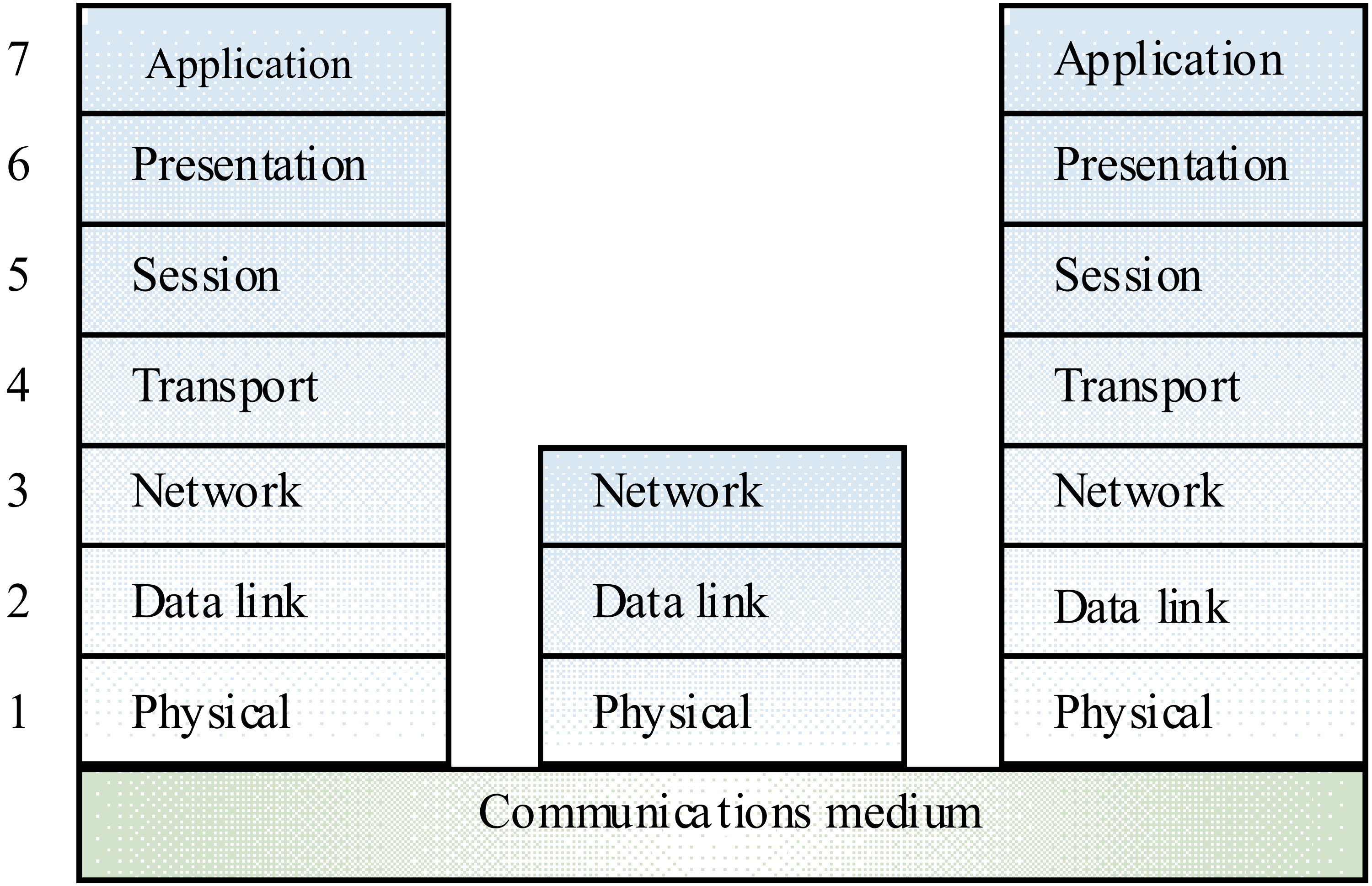
- Advantages
  - Distribution of data is straightforward
  - Makes effective use of networked systems. May require cheaper hardware
  - Easy to add new servers or upgrade existing servers
- Disadvantages
  - No shared data model so sub-systems use different data organisation. data interchange may be inefficient
  - 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

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 provide 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
- However, often difficult to structure systems in this way

ISO/OSI NETWORK MODEL



CONTROL MODELS

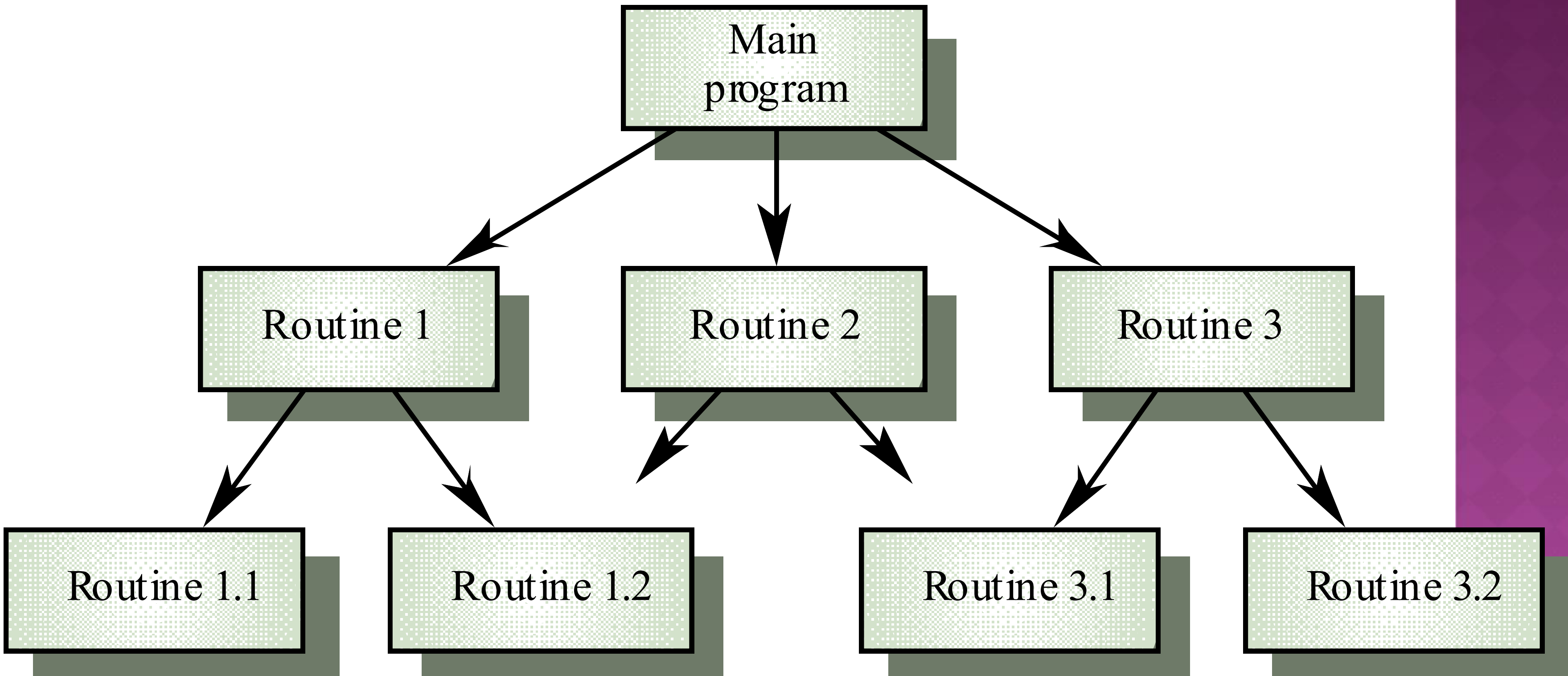
Are concerned with the control flow between sub systems. Distinct from the system decomposition model

- **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

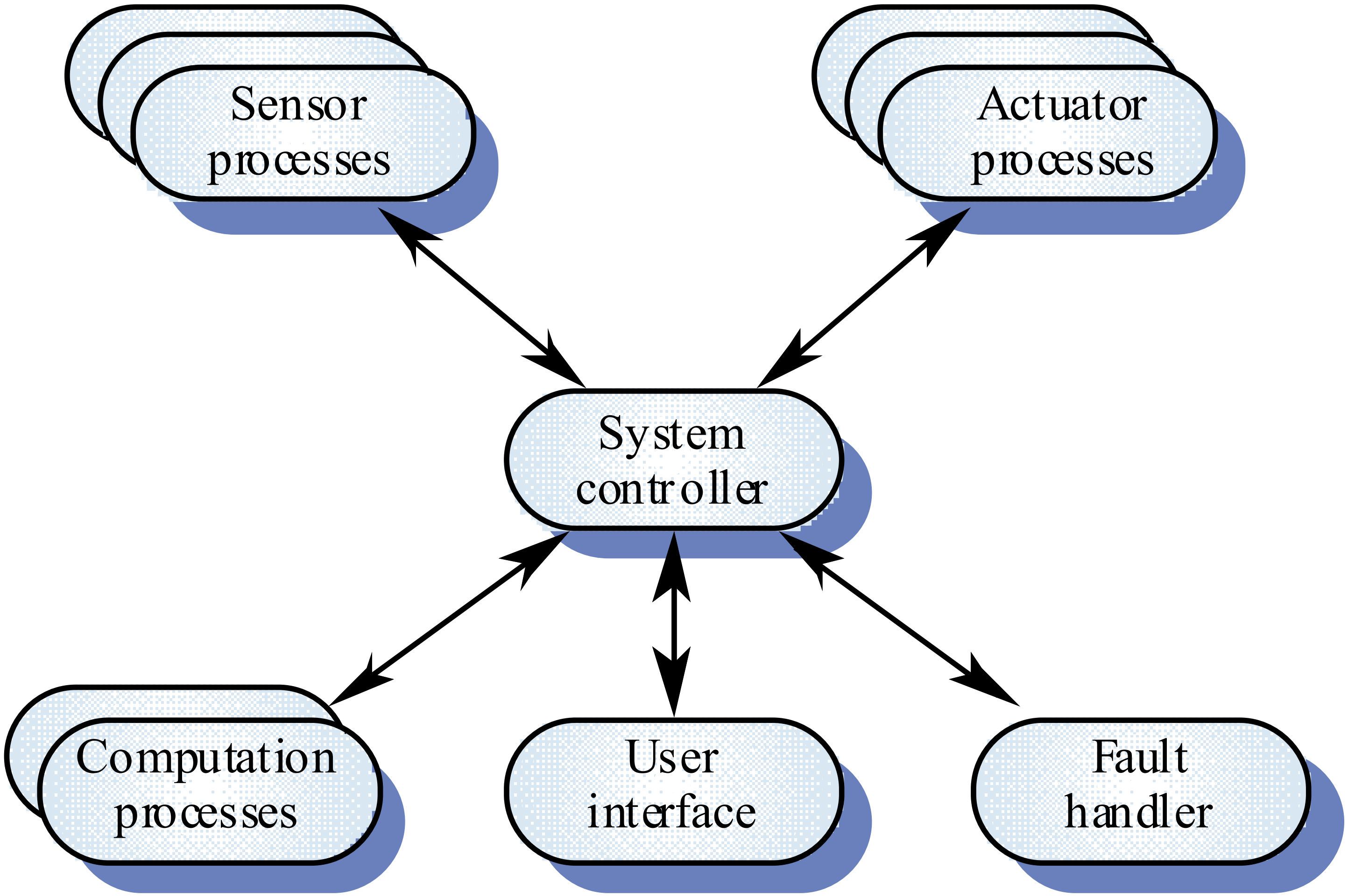
CENTRALISED CONTROL

- A control sub-system 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

CALL-RETURN MODEL



REAL-TIME SYSTEM CONTROL



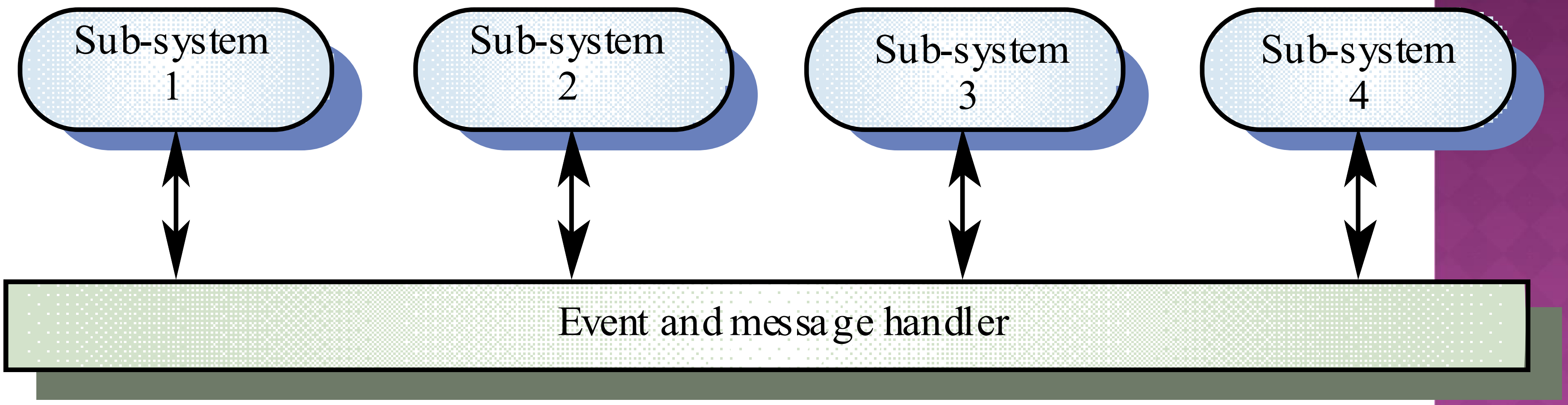
EVENT-DRIVEN SYSTEMS

- **Driven by externally generated events** where the timing of the event is out with 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
- (!!!) However, sub-systems don't know if or when an event will be handled

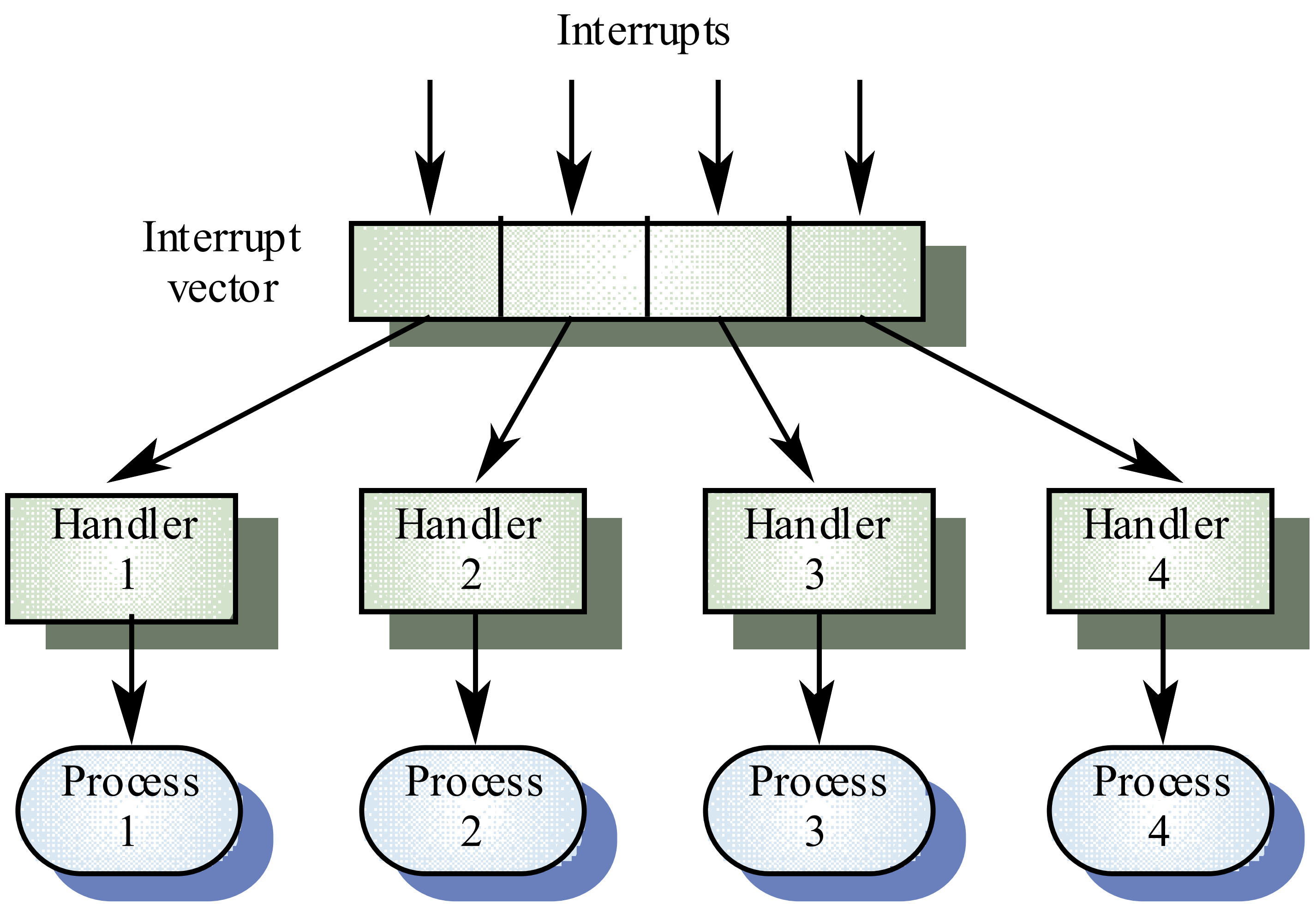
SELECTIVE BROADCASTING



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 fast response but complex to program and difficult to validate

INTERRUPT-DRIVEN CONTROL



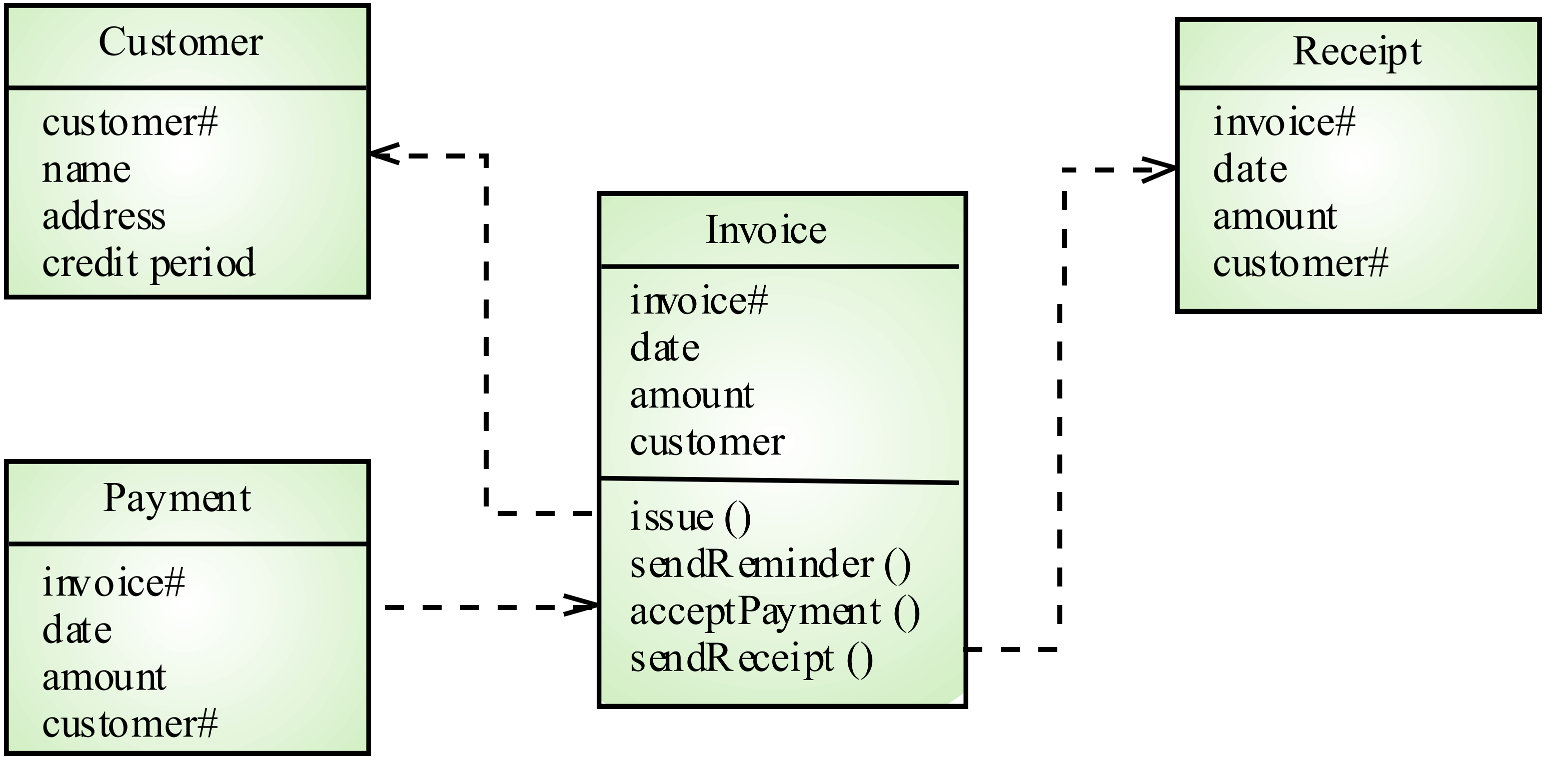
MODULAR DECOMPOSITION

- Another structural level where sub-systems are decomposed into modules
- Two modular decomposition models covered
  - An object model where the system is decomposed into interacting objects
  - A data-flow model where the system is decomposed into functional modules which transform inputs to outputs. Also known as the pipeline model
- If possible, decisions about concurrency should be delayed until modules are implemented

OBJECT 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

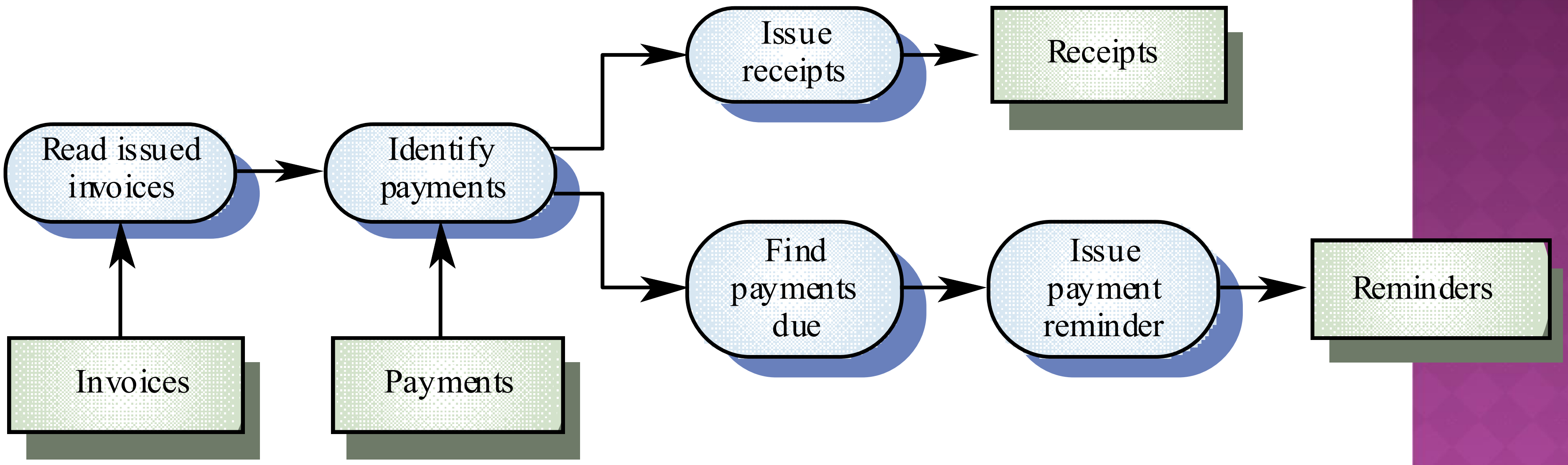
INVOICE PROCESSING SYSTEM



DATA-FLOW MODELS

- 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
- Not really suitable for interactive systems

INVOICE PROCESSING SYSTEM



DISTRIBUTED SYSTEMS ARCHITECTURES

Architectural design for software that executes on more than one processor

DISTRIBUTED SYSTEMS

- Virtually all large computer-based systems are now distributed systems
- **Information processing** is distributed over several computers rather than confined to a single machine
- **Distributed software engineering** is now very important

SYSTEM TYPES

- **Personal systems** that are not distributed and that are designed to run on a personal computer or workstation.
- **Embedded systems** that run on a single processor or on an integrated group of processors.
- **Distributed systems** where the system software runs on a loosely integrated group of cooperating processors linked by a network.

DISTRIBUTED SYSTEM CHARACTERISTICS

- Resource sharing
- Openness
- Concurrency
- Scalability
- Fault tolerance
- Transparency

Distributed system disadvantages :

Complexity

Security

Manageability

Unpredictability

DISTRIBUTED SYSTEMS ARCHIECTURES

- **Client-server architectures**
  - Distributed services which are called on by clients. Servers that provide services are treated differently from clients that use services
- **Distributed object architectures**
  - No distinction between clients and servers. Any object on the system may provide and use services from other objects

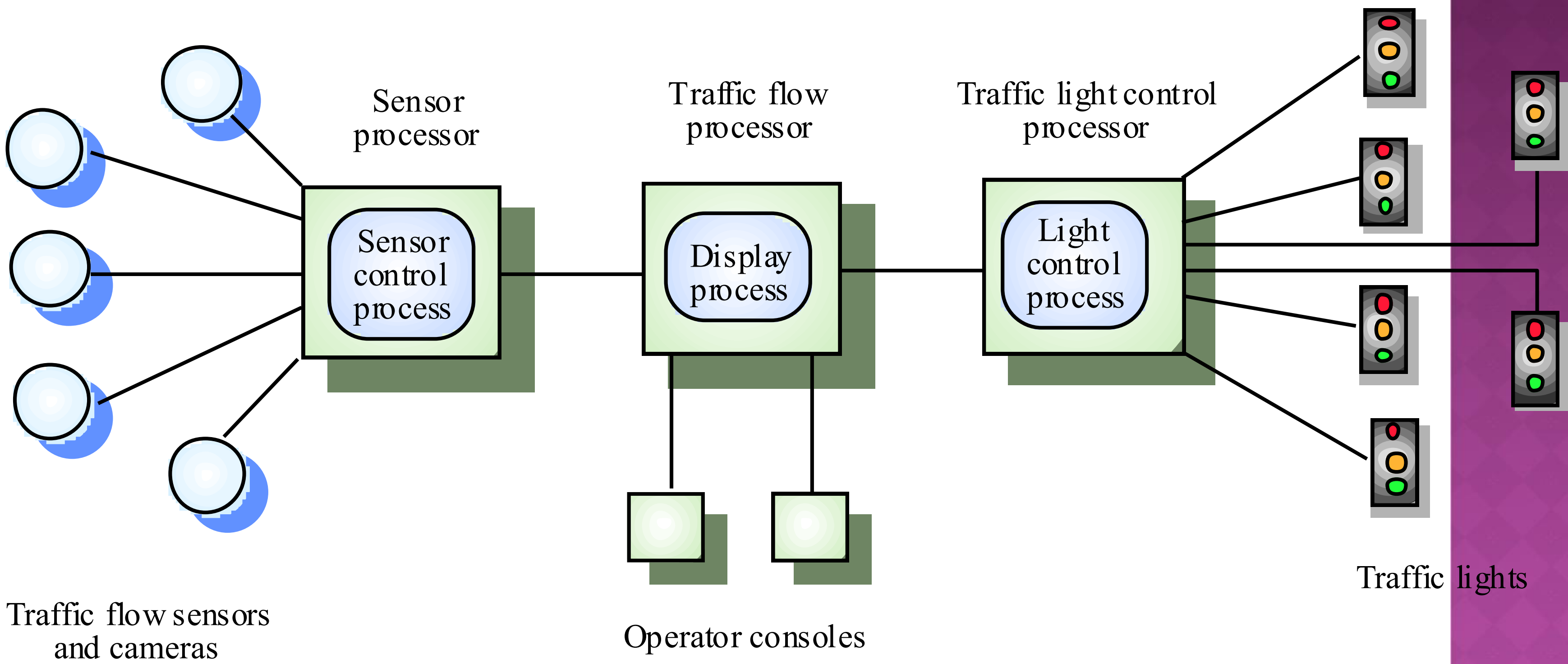
MIDDLEWARE

- Software that manages and supports the different components of a distributed system. In essence, it sits in the ***middle*** of the system
- **Middleware** is usually **off-the-shelf** rather than specially written software
- Examples
  - Transaction processing monitors
  - Data converters
  - Communication controllers

1. MULTIPROCESSOR ARCHITECTURES

- **Simplest distributed system model**
- System composed of multiple processes which may (but need not) execute on different processors
- **Architectural model of many large real-time systems**
- Distribution of process to processor may be pre-ordered or may be under the control of a dispatcher

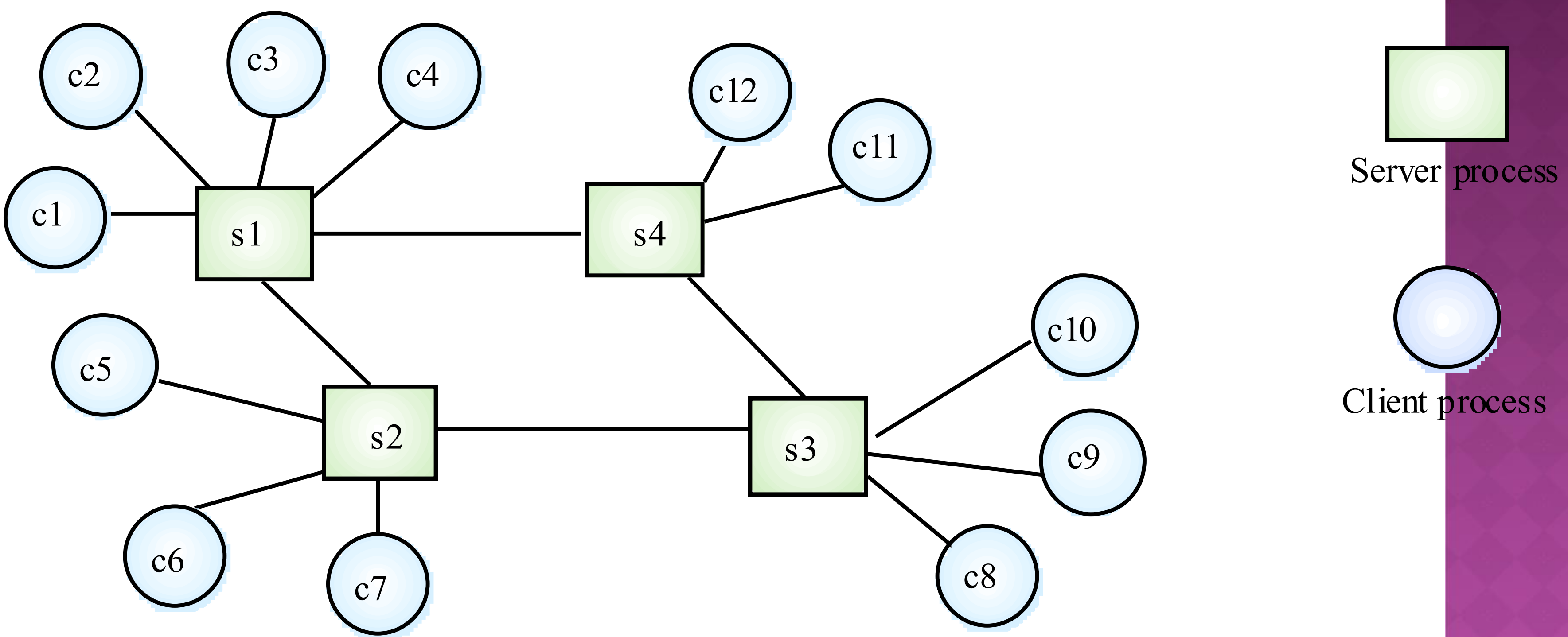
A MULTIPROCESSOR TRAFFIC CONTROL SYSTEM



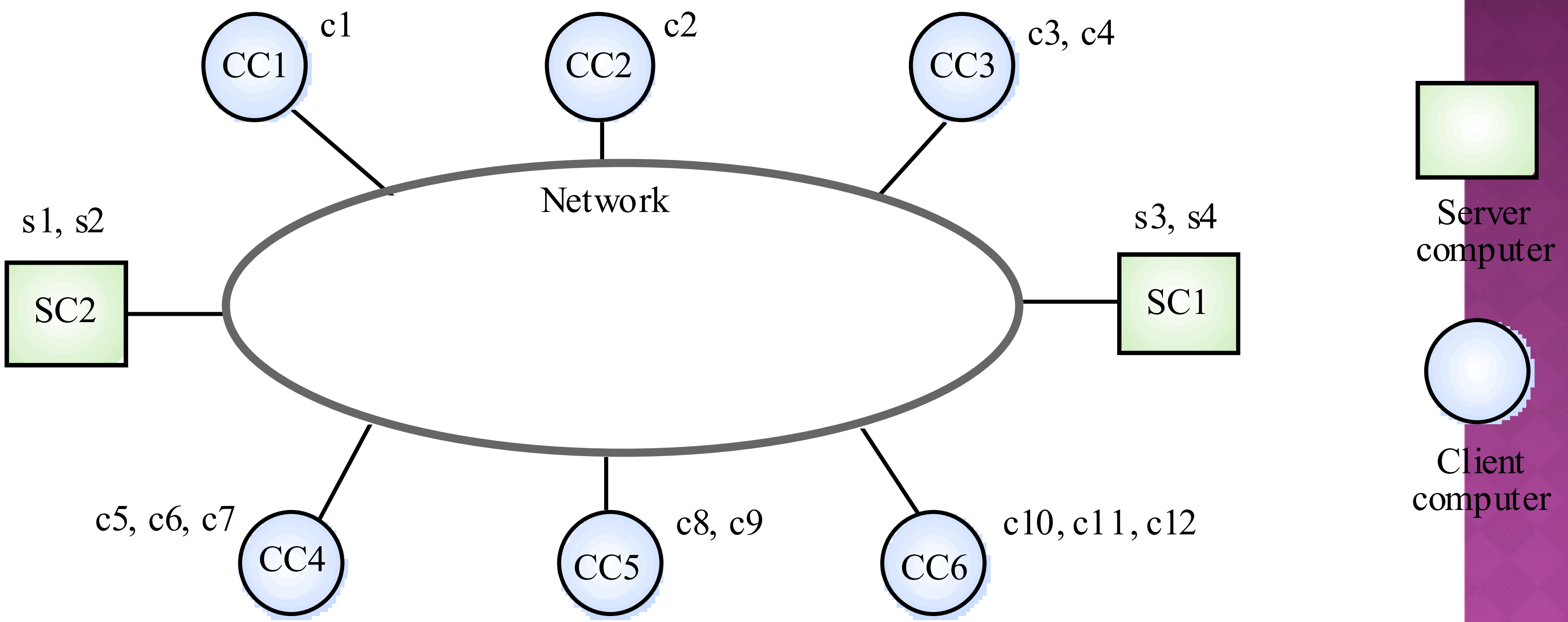
2. CLIENT-SERVER ARCHITECTURES

- **The application is modelled as a set of services** that are provided by servers and a set of clients that use these services
- **Clients know of servers but servers need not know of clients**
- Clients and servers are logical processes
- The mapping of processors to processes is not necessarily 1 : 1

A CLIENT-SERVER SYSTEM



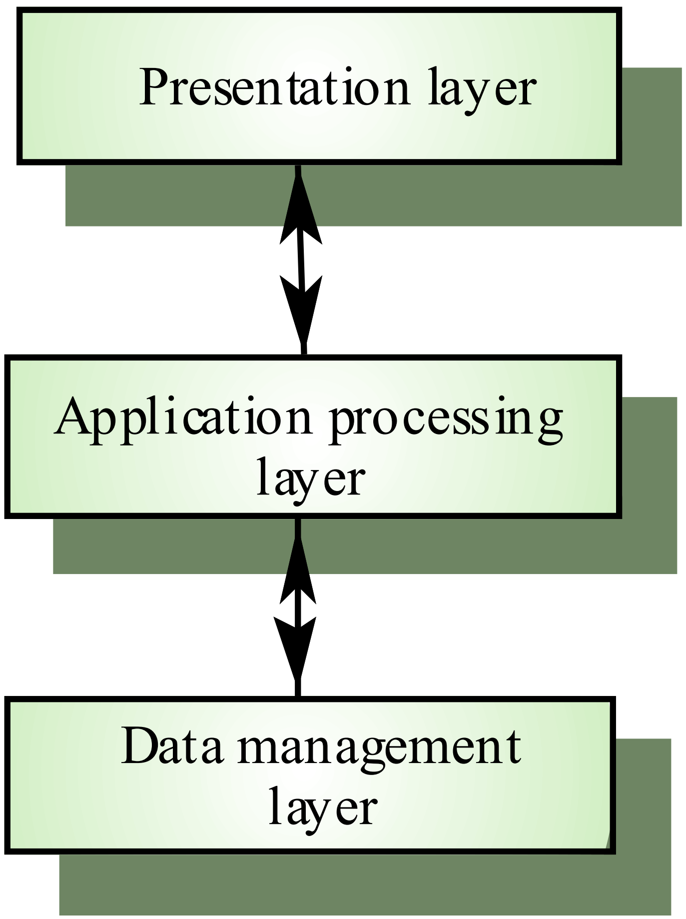
COMPUTERS IN A C/S NETWORK



LAYERED APPLICATION ARCHITECTURE

- Presentation layer
  - Concerned with presenting the results of a computation to system users and with collecting user inputs
- Application processing layer
  - Concerned with providing application specific functionality e.g., in a banking system, banking functions such as open account, close account, etc.
- Data management layer
  - Concerned with managing the system databases

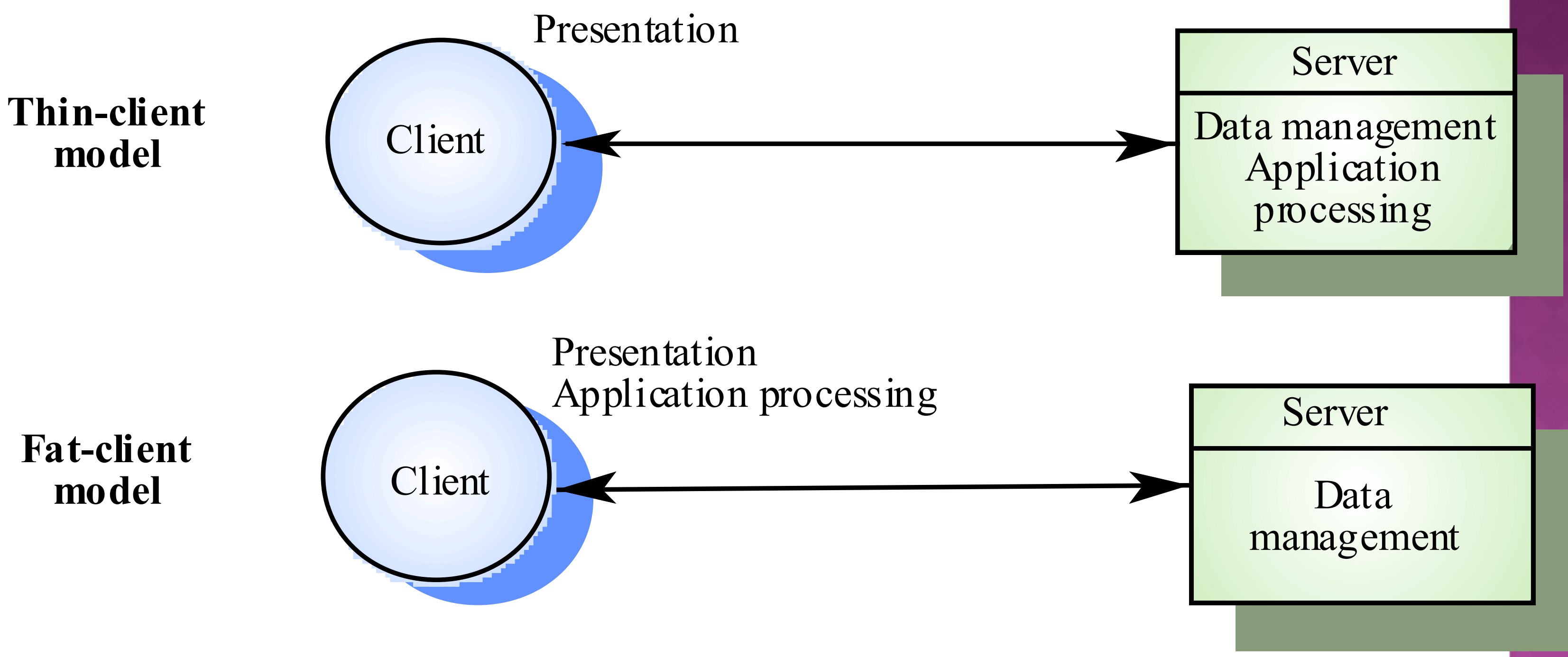
APPLICATION LAYERS



THIN AND FAT CLIENTS

- ***Thin-client model***
  - In a thin-client model, all of the application processing and data management is carried out on the server. The client is simply responsible for running the presentation software.
- ***Fat-client model***
  - In this model, the server is only responsible for data management. The software on the client implements the application logic and the interactions with the system user.

THIN AND FAT CLIENTS



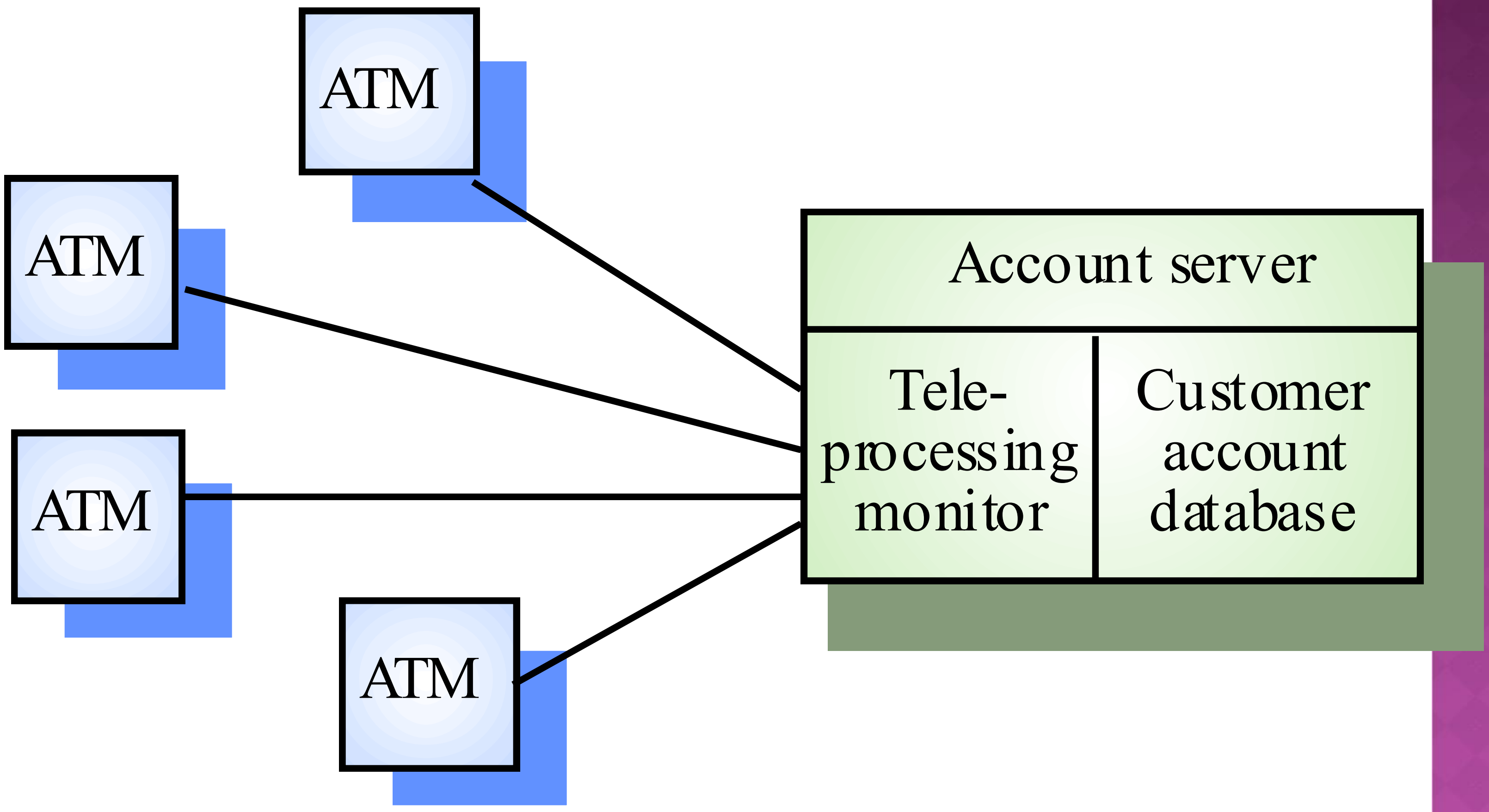
THIN CLIENT MODEL

- **Used when legacy systems are migrated to client server architectures.**
  - The legacy system acts as a server in its own right with a graphical interface implemented on a client
- A major disadvantage is that it places a heavy processing load on both the server and the network

FAT CLIENT MODEL

- More processing is delegated to the client as the application processing is locally executed
- Most suitable for new C/S systems where the capabilities of the client system are known in advance
- More complex than a thin client model especially for management. New versions of the application have to be installed on all clients

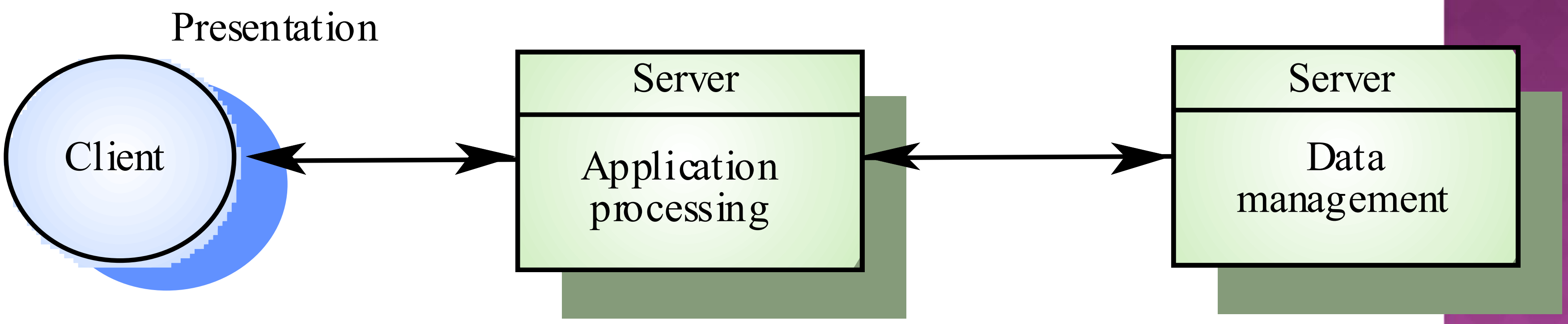
A CLIENT-SERVER ATM SYSTEM



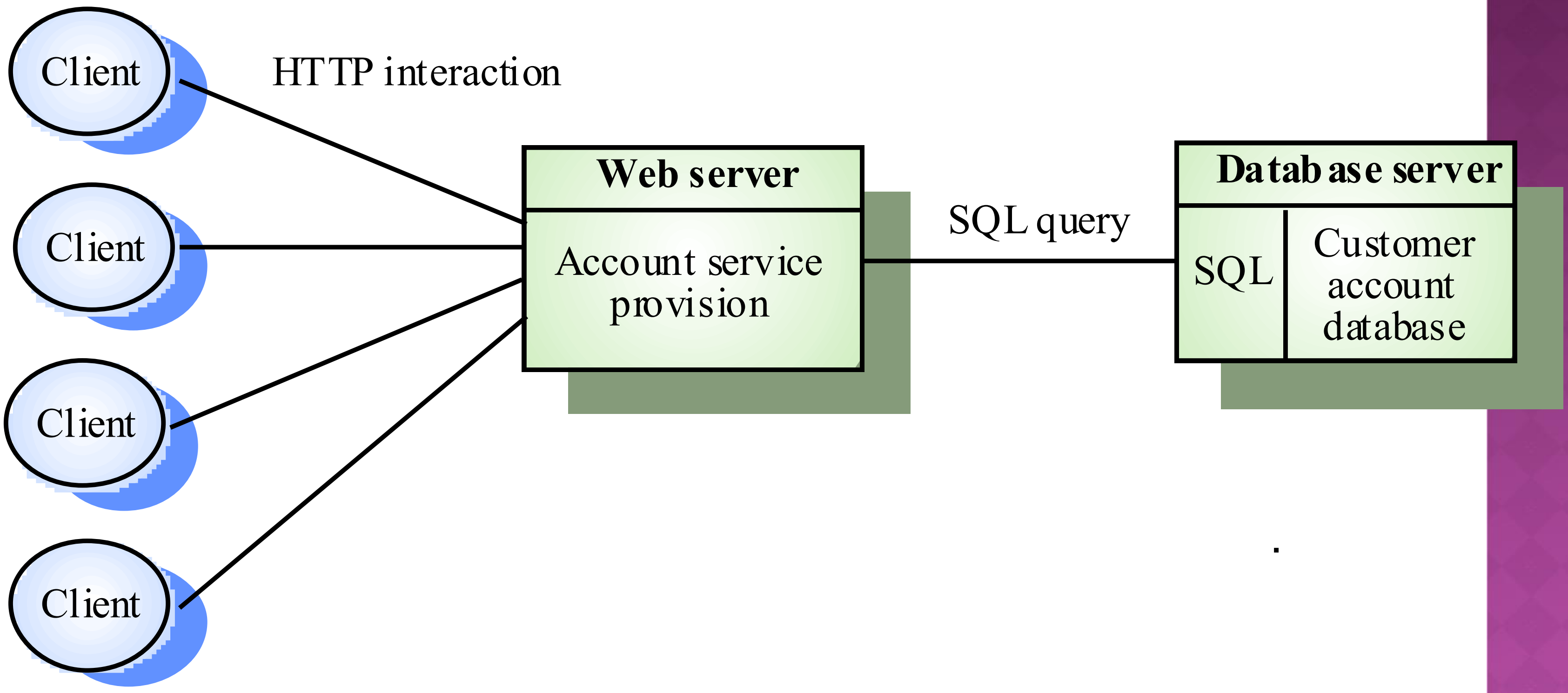
THREE-TIER ARCHITECTURES

- In a three-tier architecture, each of the application architecture layers may execute on a separate processor
- Allows for better performance than a thin-client approach and is simpler to manage than a fat-client approach
- A more scalable architecture - as demands increase, extra servers can be added

A 3-TIER C/S ARCHITECTURE



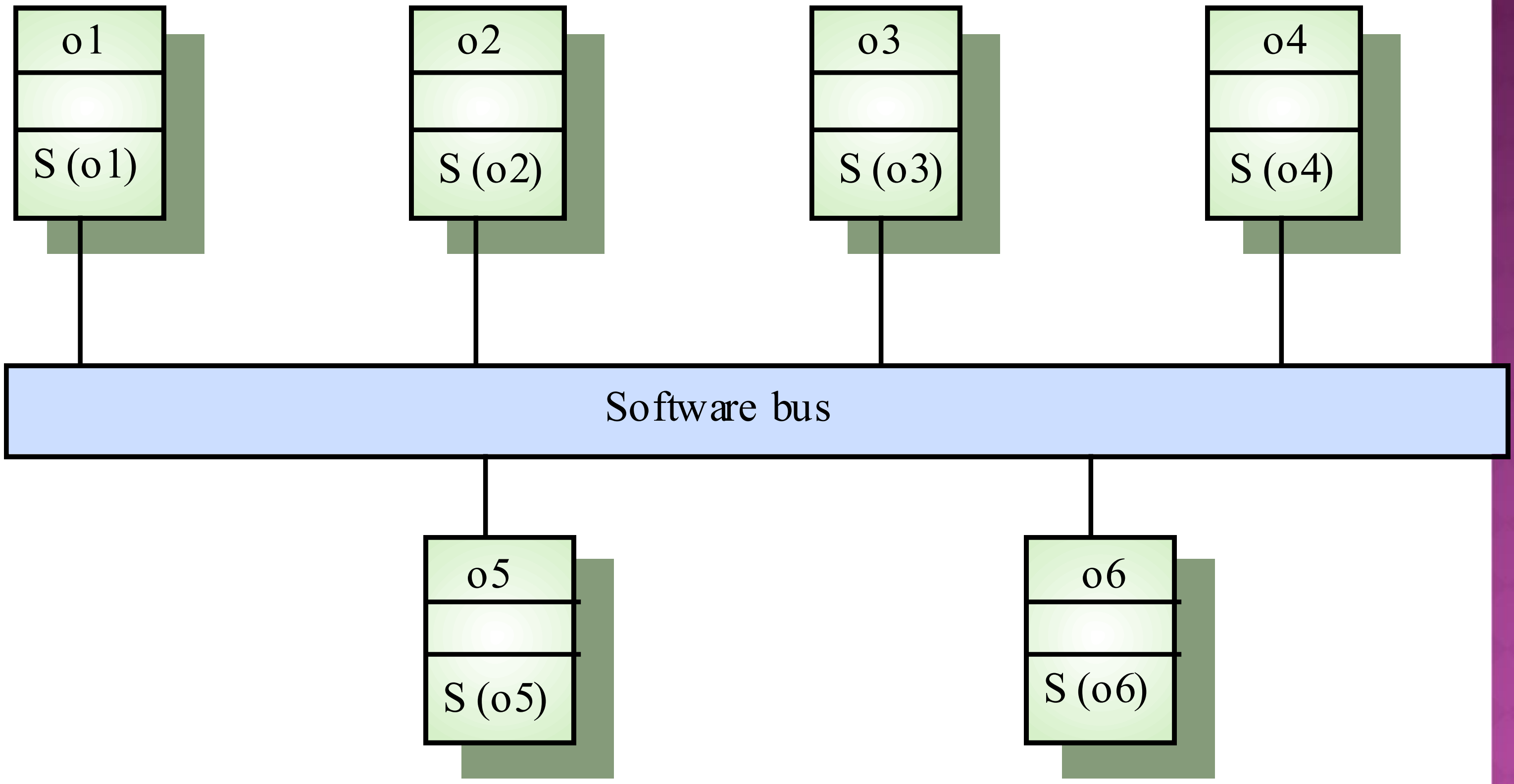
AN INTERNET BANKING SYSTEM



3. DISTRIBUTED OBJECT ARCHITECTURES

- **There is no distinction in a distributed object architectures between clients and servers**
- Each distributable entity is an object that
  - provides services to other objects and
  - receives services from other objects
- Object communication is through a middleware system called an object request broker (software bus)
- However, more complex to design than C/S systems

DISTRIBUTED OBJECT ARCHITECTURE



ADVANTAGES OF DISTRIBUTED OBJECT ARCHITECTURE

- It allows the system designer to delay decisions on where and how services should be provided
- It is a very open system architecture that allows new resources to be added to it as required
- The system is flexible and scaleable
- It is possible to reconfigure the system dynamically with objects migrating across the network as required

USES OF DISTRIBUTED OBJECT ARCHITECTURE

- **As a logical model that allows you to structure and organise the system**. In this case, you think about how to provide application functionality solely in terms of services and combinations of services
- **As a flexible approach to the implementation of client-server systems.** The logical model of the system is a client-server model but both clients and servers are realised as distributed objects communicating through a software bus

AIMS

- To illustrate the various model of exception handing
- To consider the Ada and Java models of exception handling in detail and to show how exception handling can be used as a framework for implementing fault-tolerant systems