Architectural Design

Book Referred: Sommerville 8th Edition, Chapter 11

Objectives

- To introduce architectural design and to discuss its importance
- To explain the architectural design decisions that have to be made
- To introduce three complementary architectural styles covering organisation, decomposition and control
- To discuss reference architectures are used to communicate and compare architectures

Topics covered

- Architectural design decisions
- System organisation
- Decomposition styles
- Control styles
- Reference 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.
- *An architecture is the abstract design concept of an application. Basically, a structure of the moving parts and how they're connected.*
- 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.

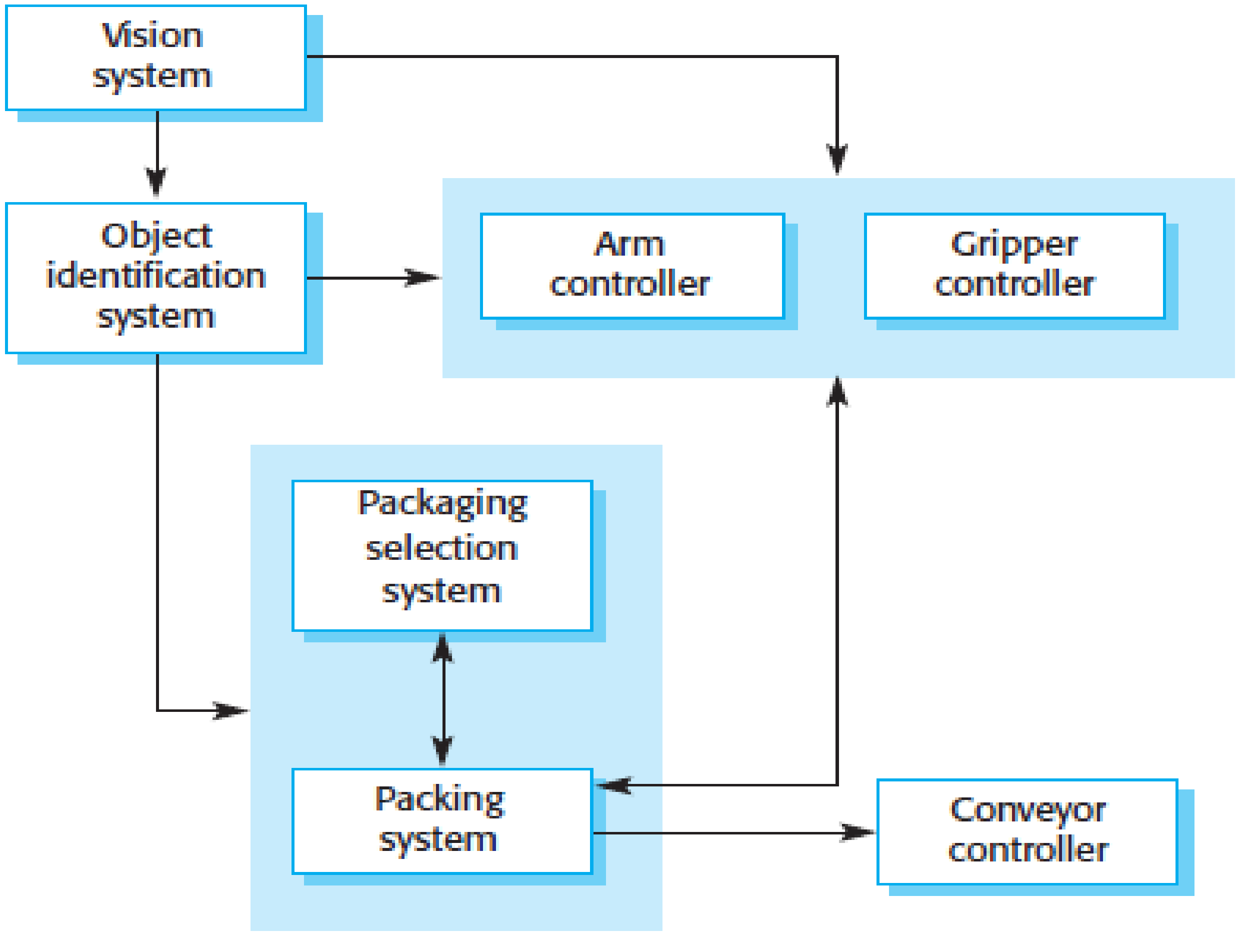
Architecture and System Characteristics

- Performance
  - Localize critical operations and minimize communications. Granularity is the extent to which a system is broken down into small parts, either the system itself or its description or observation. It is the extent to which a larger entity is subdivided. Use large-grain/coarse-grain 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, self-contained components.

Architectural Conflicts

- Using large-grain components improves performance but reduces maintainability.
- Introducing redundant data improves availability but makes security more difficult.

Packing Robot Control System



Architectural Design Decisions

- 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?

Architectural Design Document

- The product of the architectural design process is an architectural design document.
- This may include a number of graphical representations of the system along with associated descriptive text.
- It should describe how the system is structured into sub-systems, the approach adopted and how each sub-system is structured into modules.
- The graphical models of the system present different perspectives on the architecture.

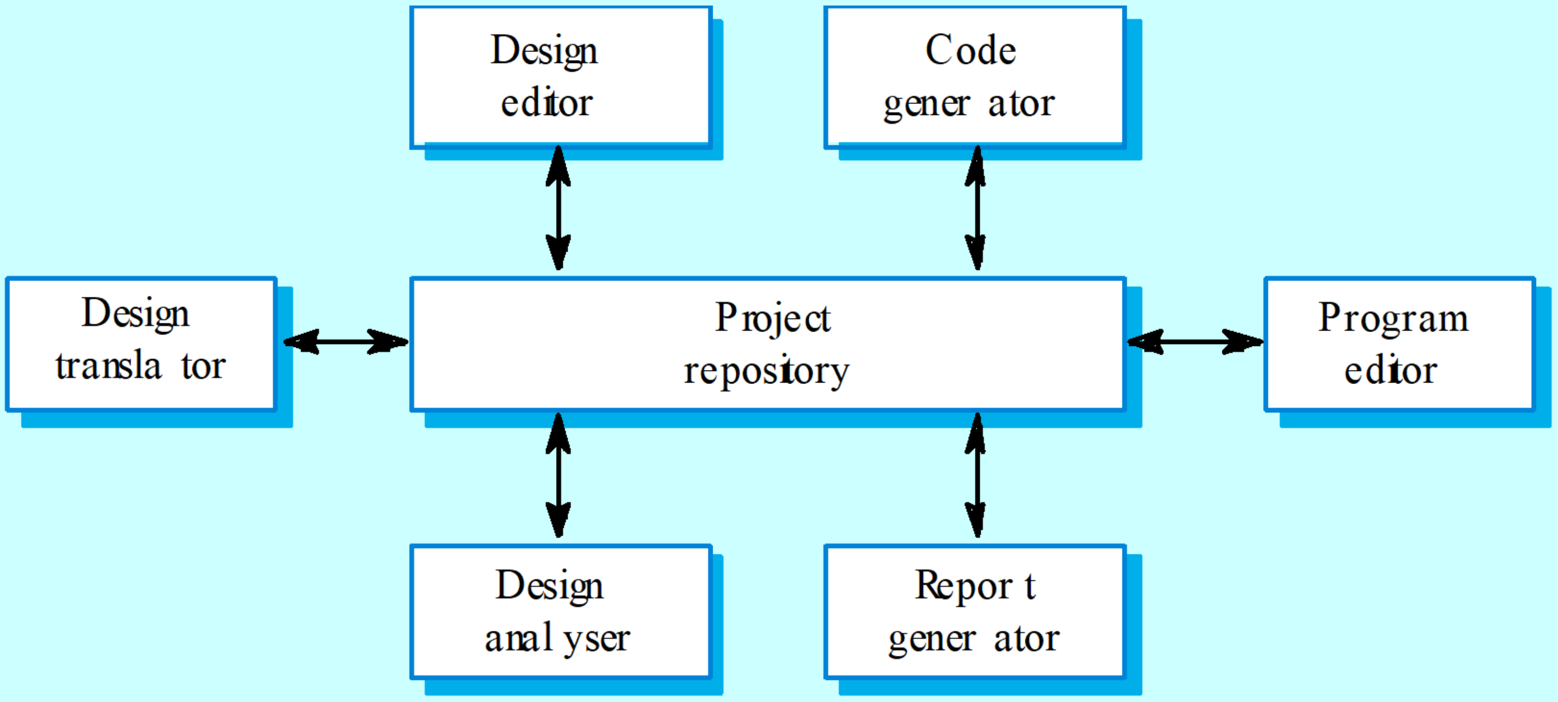
System Organisation

- Reflects the basic strategy that is used to structure a system.
- Three organizational styles are widely used:
  - A shared data repository style;
  - A shared services and servers style;
  - An abstract machine or layered style.

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 (still shared) 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.
- Examples : CASE Tool Repository, Database Management System, Management Information System etc.

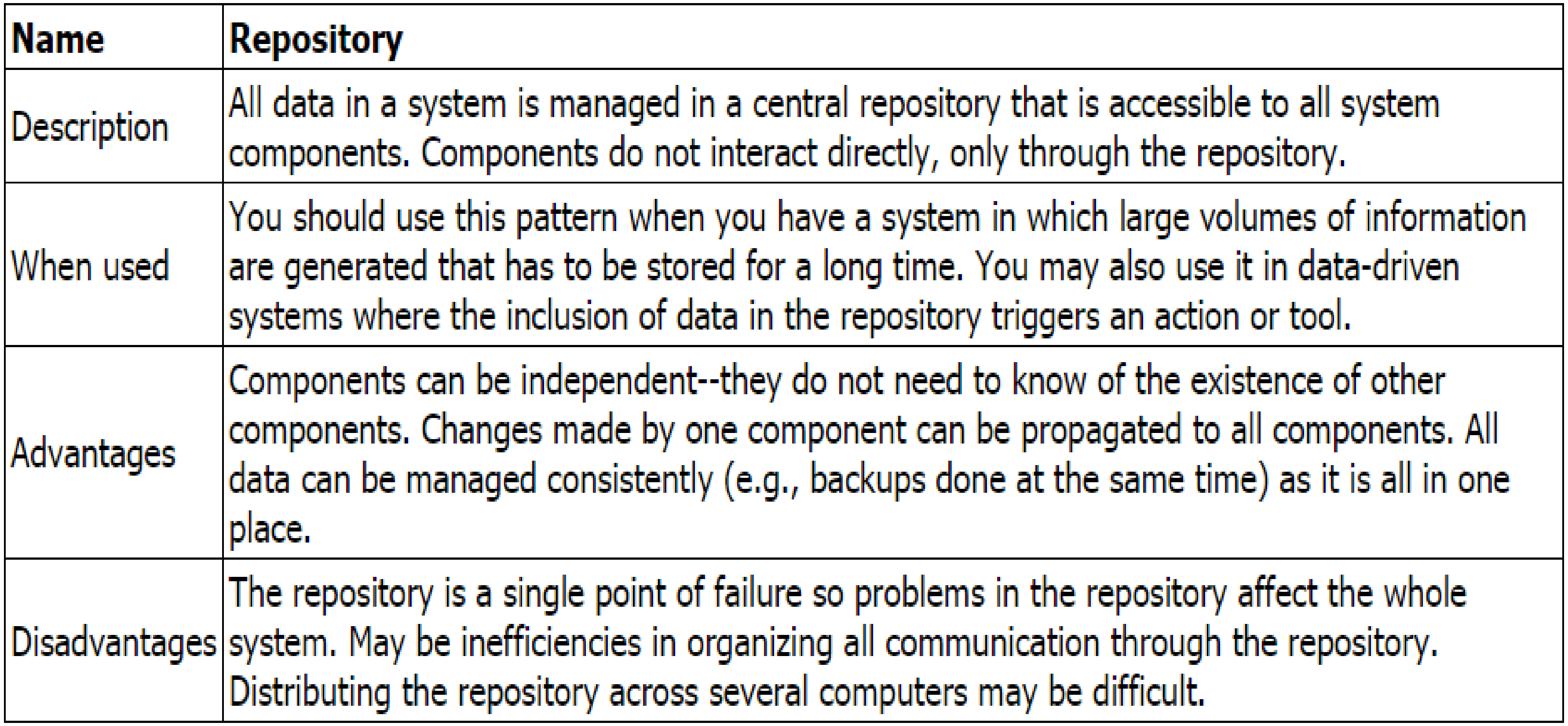
CASE Toolset Architecture



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.
- Disadvantages
  - Sub-systems must agree on a repository data model. Inevitably a compromise;
  - No scope for specific management policies;
  - Difficult to distribute efficiently.
  - Failure of repository affects the whole system.

Repository Model

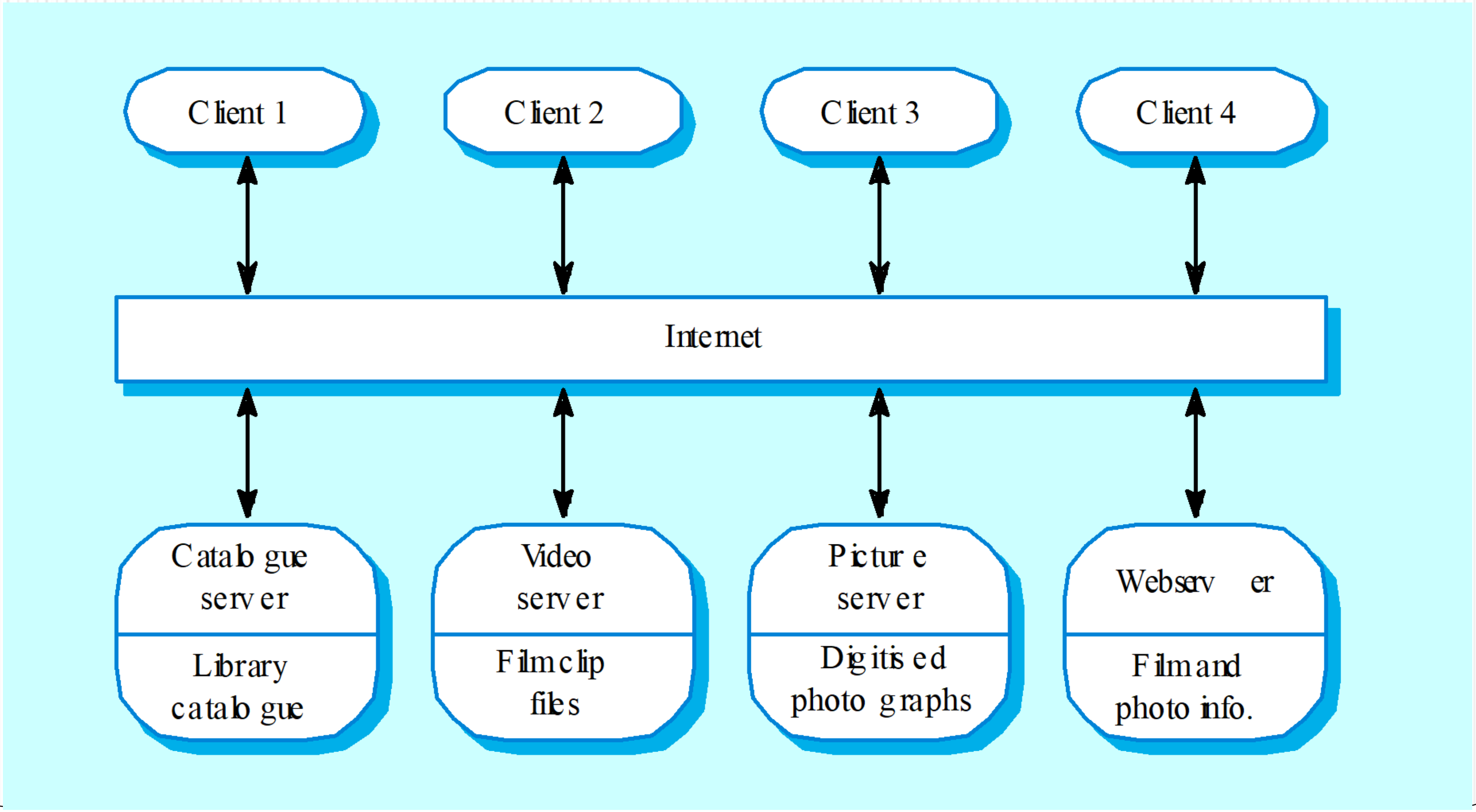


Name	Repository
Description	All data in a system is managed in a central repository that is accessible to all system components. Components do not interact directly, only through the repository.
When used	You should use this pattern when you have a system in which large volumes of information are generated that has to be stored for a long time. You may also use it in data-driven systems where the inclusion of data in the repository triggers an action or tool.
Advantages	Components can be independent--they do not need to know of the existence of other components. Changes made by one component can be propagated to all components. All data can be managed consistently (e.g., backups done at the same time) as it is all in one place.
Disadvantages	The repository is a single point of failure so problems in the repository affect the whole system. May be inefficiencies in organizing all communication through the repository. Distributing the repository across several computers may be difficult.

Client-Server Model

- 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.
- A good example of a client-server architecture is a library system that may provide multiple services to multiple students such as an article database, a book finder and an order placer.

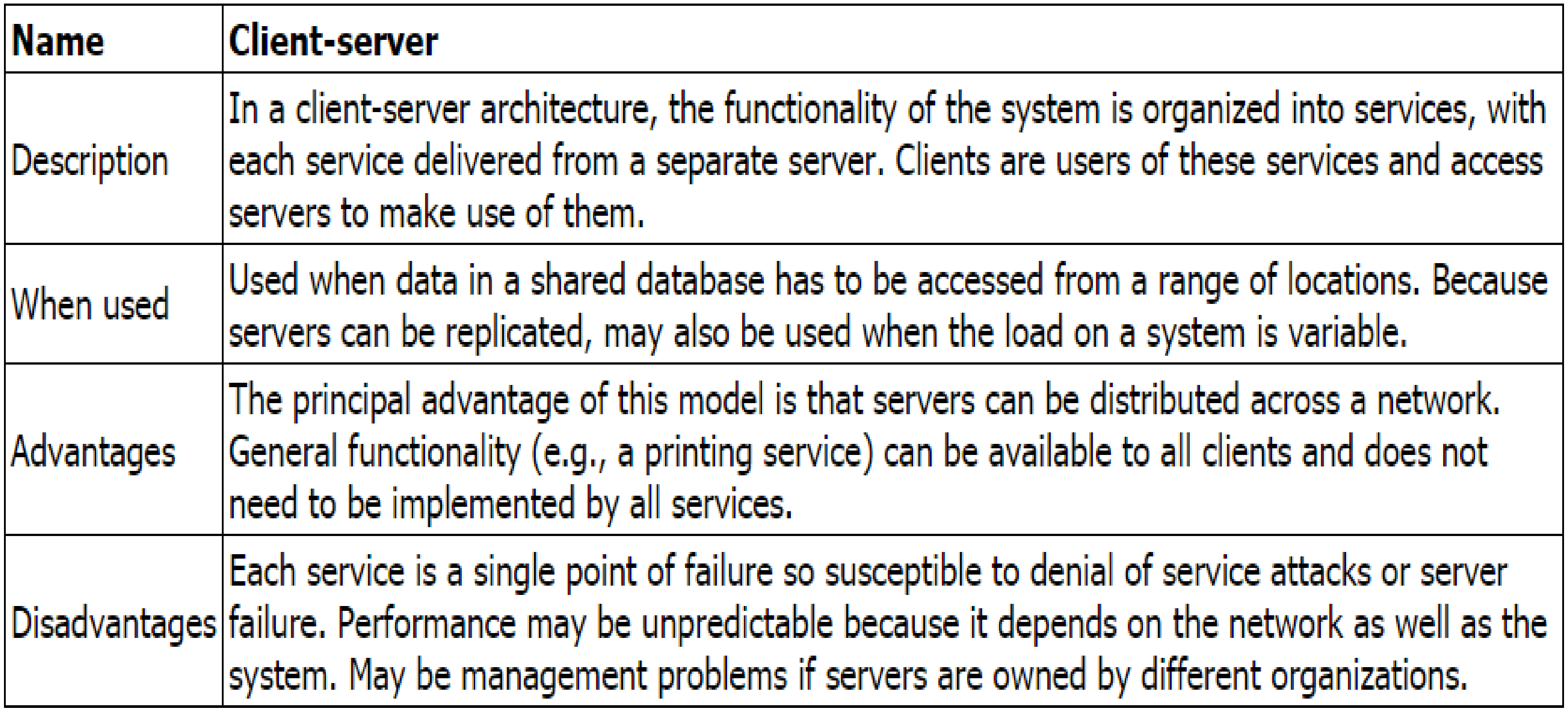
Film and Picture Library



Client-Server Characteristics

- Advantages
  - Distribution of data and services is straightforward;
  - Makes effective use of networked systems.
  - 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.

Client-Server

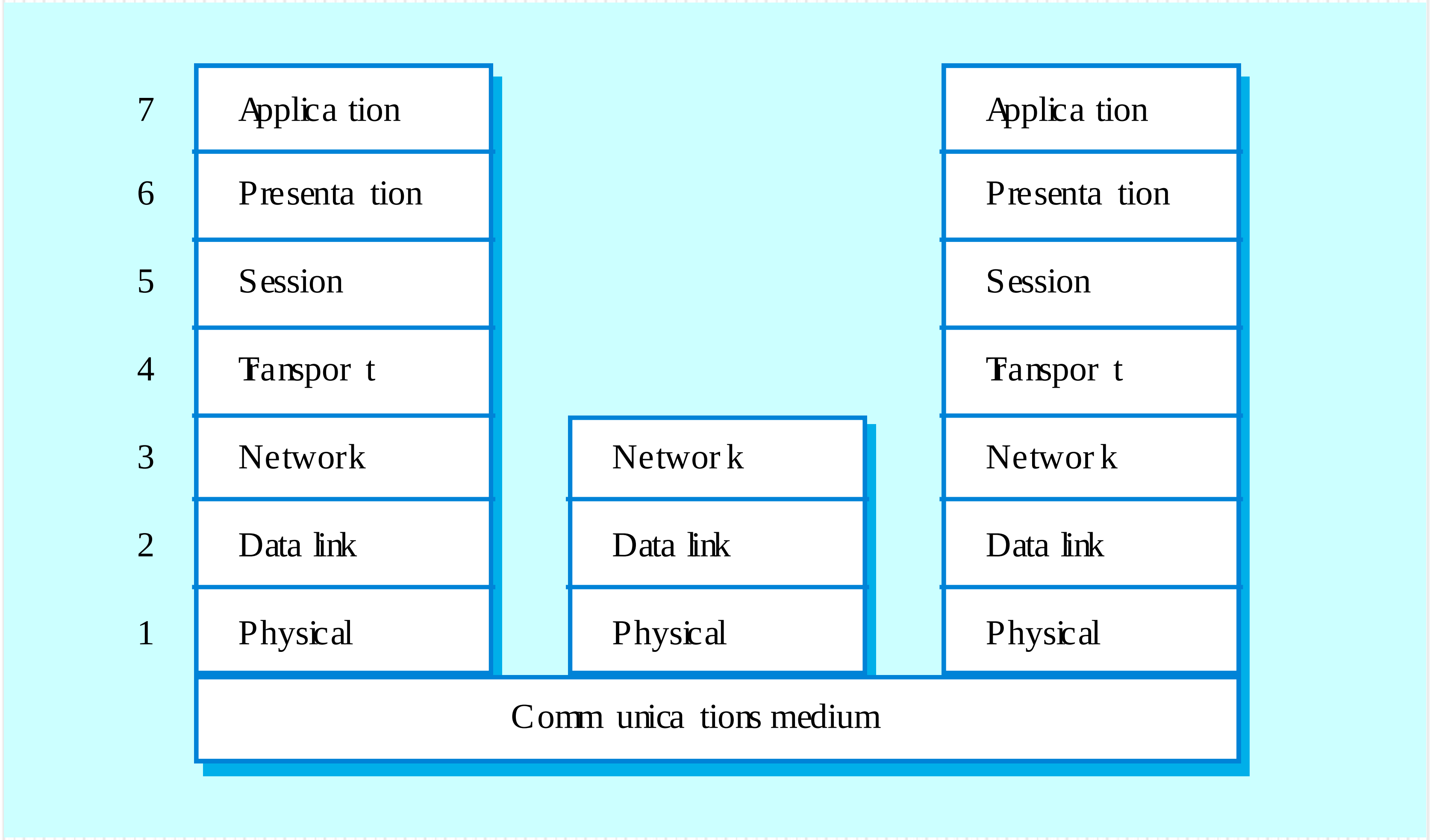


Name	Client-server
Description	In a client-server architecture, the functionality of the system is organized into services, with each service delivered from a separate server. Clients are users of these services and access servers to make use of them.
When used	Used when data in a shared database has to be accessed from a range of locations. Because servers can be replicated, may also be used when the load on a system is variable.
Advantages	The principal advantage of this model is that servers can be distributed across a network. General functionality (e.g., a printing service) can be available to all clients and does not need to be implemented by all services.
Disadvantages	Each service is a single point of failure so susceptible to denial of service attacks or server failure. Performance may be unpredictable because it depends on the network as well as the system. May be management problems if servers are owned by different organizations.

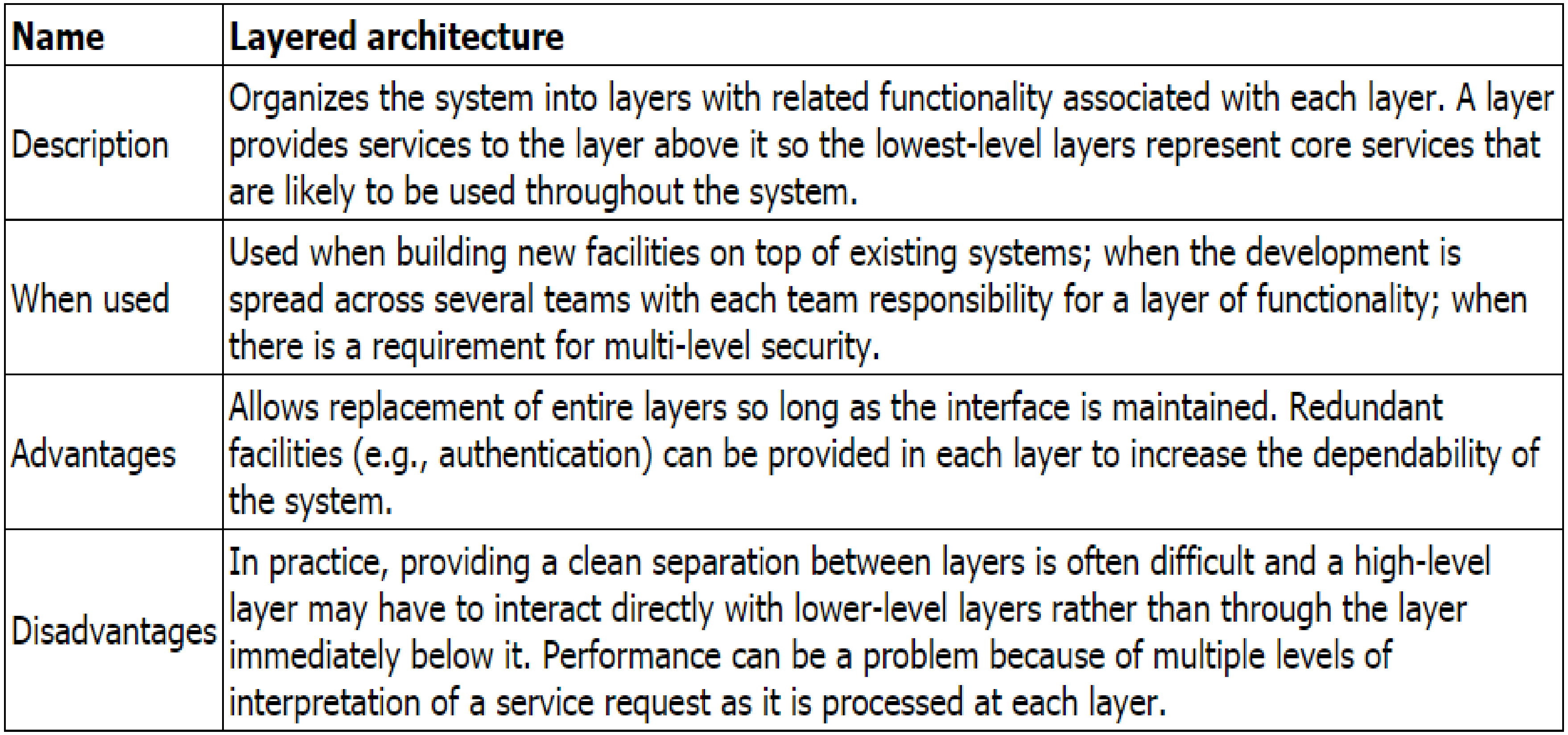
Abstract Machine (Layered) Model

- 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 artificial to structure systems in this way.

OSI Reference Model



Layered Model



Name	**Layered architecture**
Description	Organizes the system into layers with related functionality associated with each layer. A layer provides services to the layer above it so the lowest-level layers represent core services that are likely to be used throughout the system.
When used	Used when building new facilities on top of existing systems; when the development is spread across several teams with each team responsibility for a layer of functionality; when there is a requirement for multi-level security.
Advantages	Allows replacement of entire layers so long as the interface is maintained. Redundant facilities (e.g., authentication) can be provided in each layer to increase the dependability of the system.
Disadvantages	In practice, providing a clean separation between layers is often difficult and a high-level layer may have to interact directly with lower-level layers rather than through the layer immediately below it. Performance can be a problem because of multiple levels of interpretation of a service request as it is processed at each layer.

Modular Decomposition Styles

- Styles of decomposing sub-systems into modules.
- No rigid distinction between system organisation and modular decomposition.

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. Sub-systems are composed of modules and have defined interfaces, which are used for communication with 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.

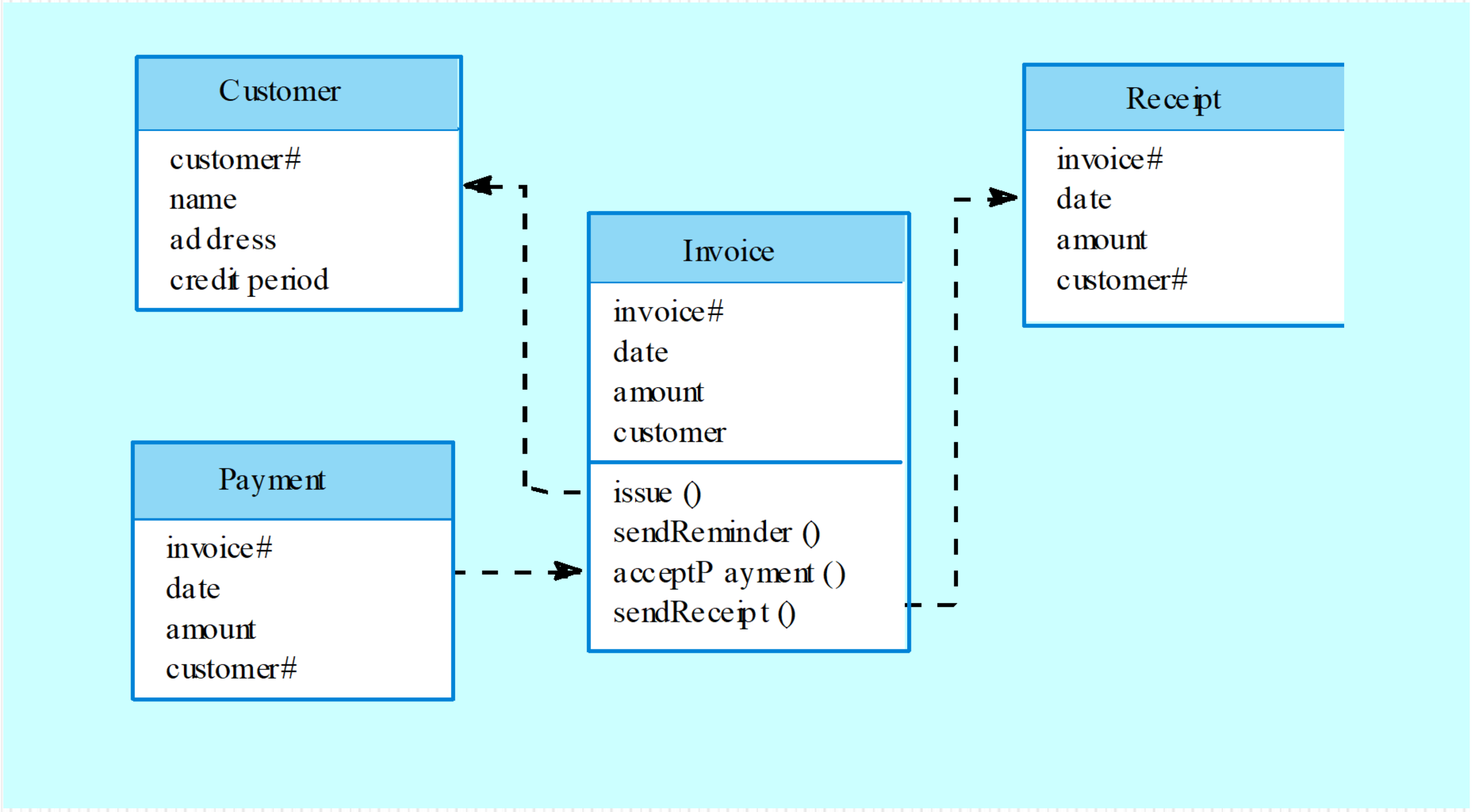
Modular Decomposition

- Another structural level where sub-systems are decomposed into modules.
- Two modular decomposition models covered
  - *Object-oriented decomposition* where you decompose a system into a set of communicating objects.
  - *Function-oriented pipelining* where you decompose a system into functional modules that accept input data and transform it into output data.

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



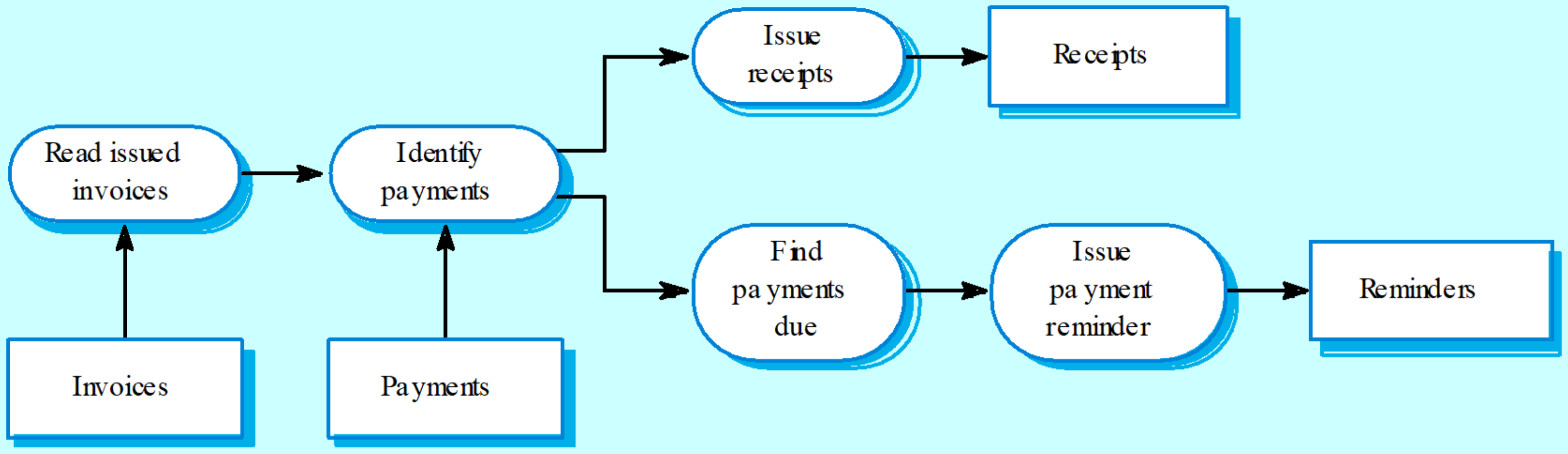
Object Model Advantages

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

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.
- Not really suitable for interactive systems.

Invoice Processing System



Function Oriented Pipeline Model Advantages

- Supports transformation reuse.
- Intuitive organisation for stakeholder communication.
- Easy to add new transformations.
- Relatively simple to implement as either a concurrent or sequential system.
- However, requires a common format for data transfer along the pipeline and difficult to support event-based interaction.

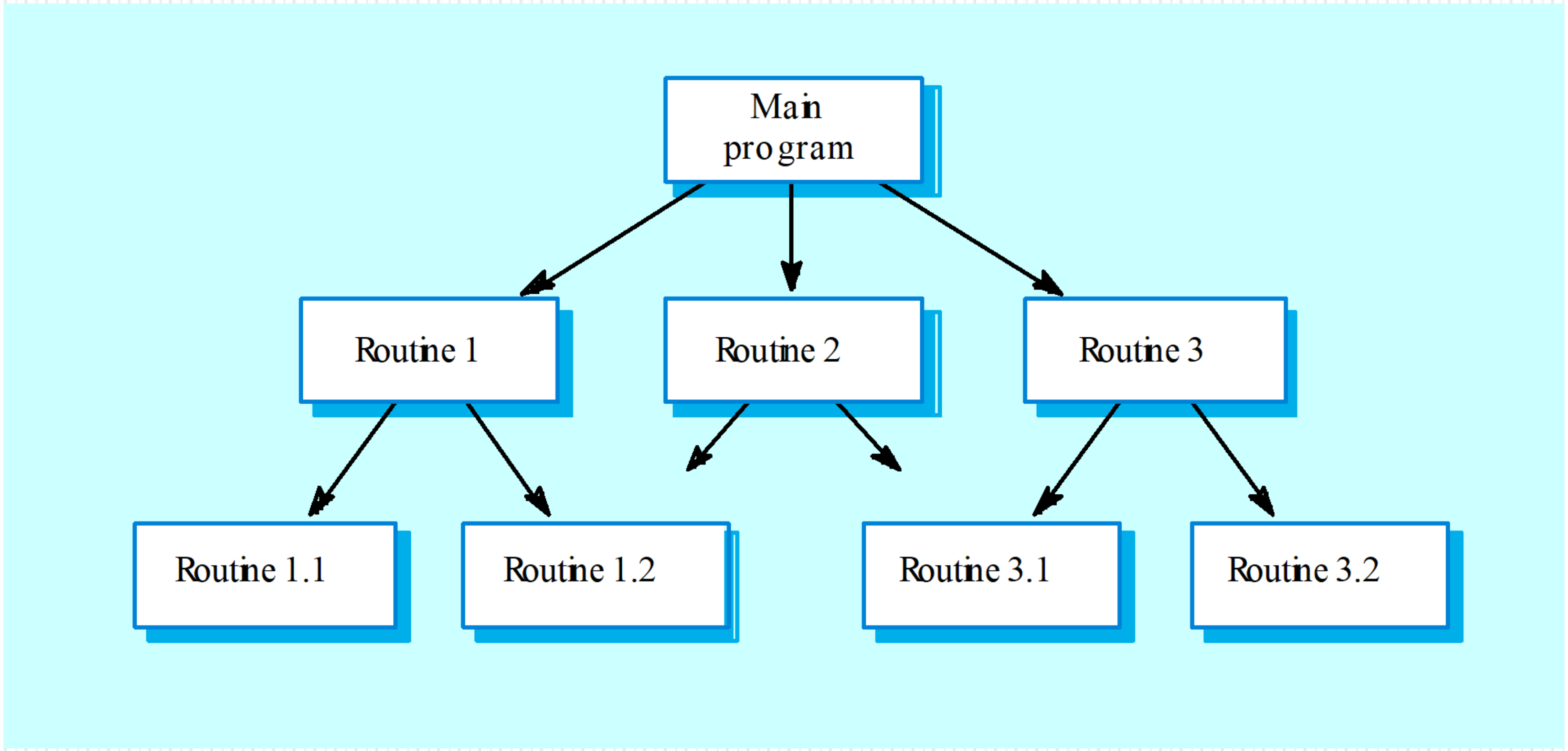
Control Styles

- To work as a system, sub-systems must be controlled so that their services are delivered to the right place at the right time.
- 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.

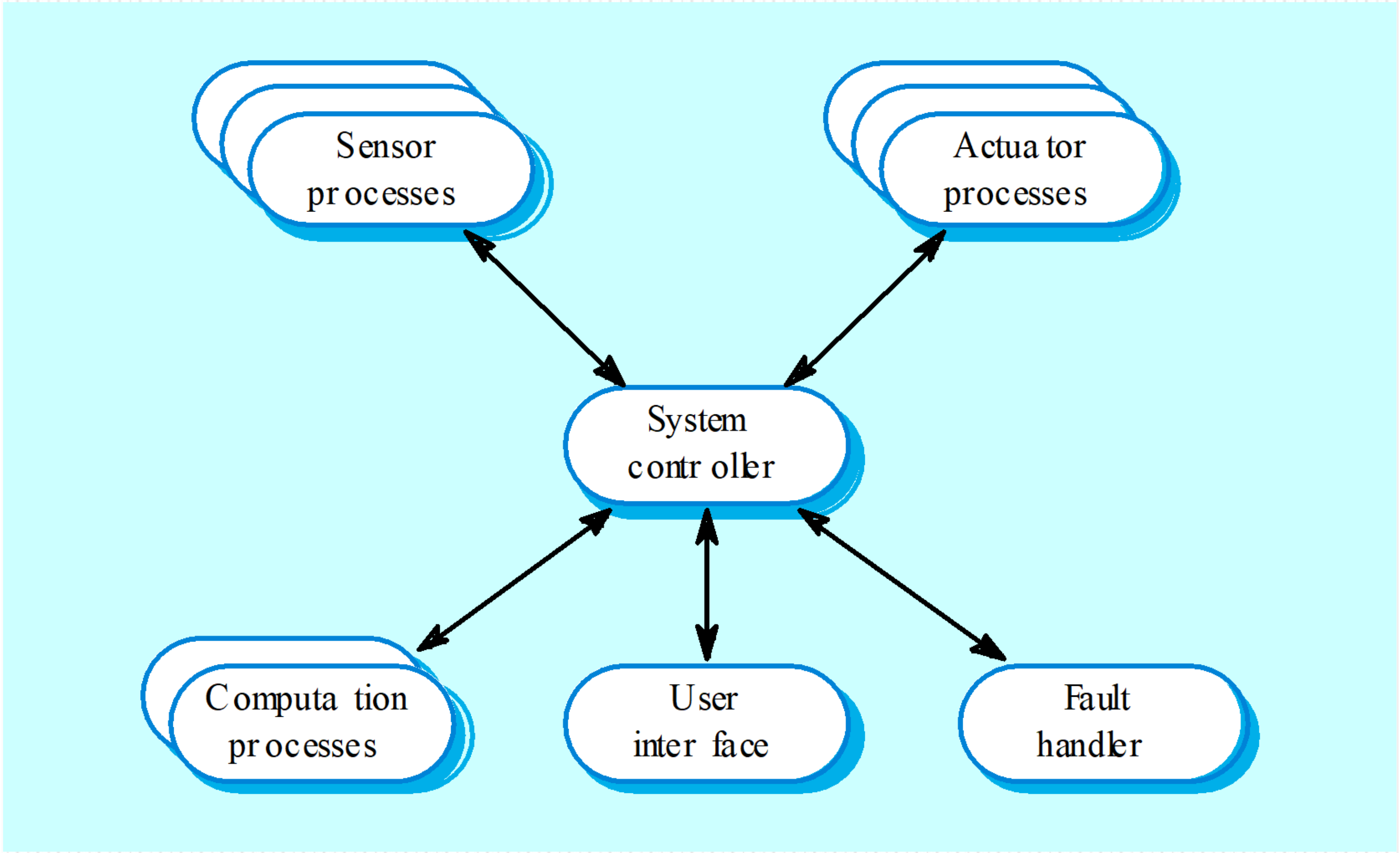
Centralised Control

- 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



Manager Model



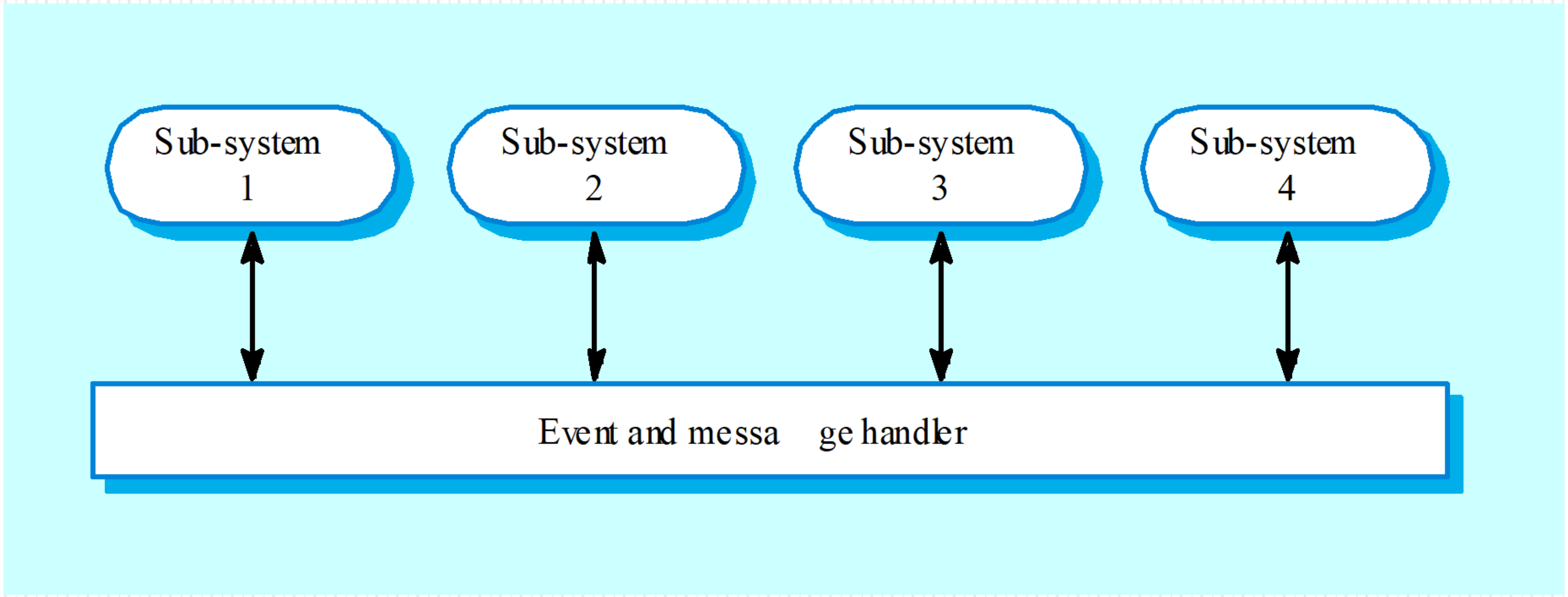
Event-driven Systems

- Driven by externally generated events where the timing of the event is outwith 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 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.

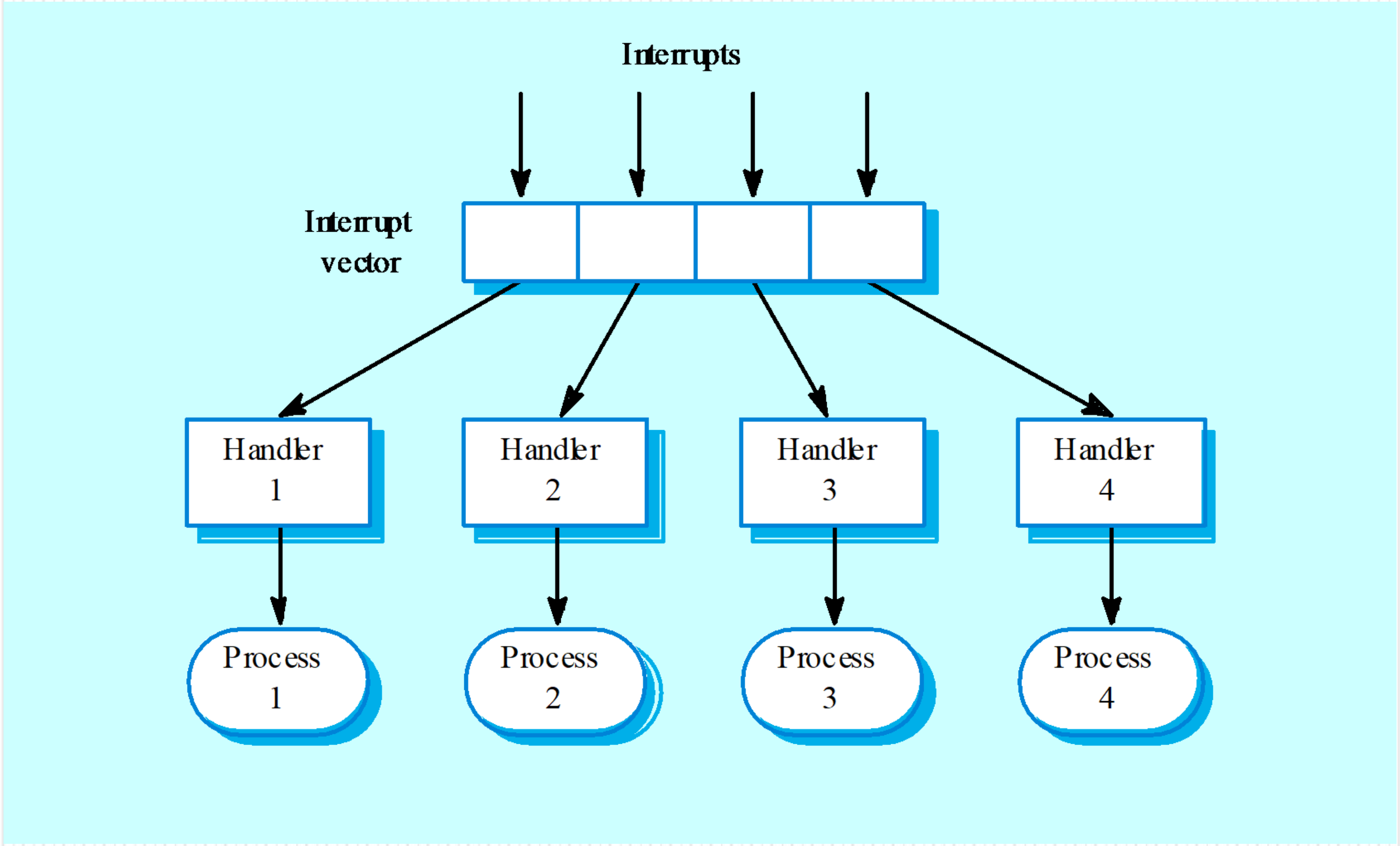
Selective Broadcasting



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



Key points

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

Thank You!!

ANY QUESTIONS??