Architectural Design

Sommerville 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, modular decomposition, and control
- To discuss reference architectures that are used to communicate and compare architectures

Topics covered

- System organisation
- Modular 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.
- The output of this design process is a description of the software architecture.

Advantages of explicit architecture

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

Architecture in determining system characteristics

- Performance
  - Localise critical operations and minimise communications. Use large-grain 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.

Example: Architectural conflicts

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

Architectural design decisions

- Architectural design is a creative process so the process differs depending on the type of system being developed.
- However, a number of common decisions span all design processes.

Architectural design decisions

- Is there a generic application architecture that can be used?
- How will the system be distributed?
- 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: Three Questions

- System structuring
  - The system is decomposed into several principal sub-systems e.g. client-server, layered structure etc.
- Modular decomposition
  - The identified sub-systems are decomposed into modules
- Control modelling
  - Decisions about how the execution of sub-systems is controlled

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

Perspectives for building architectural models

- structural model that shows the major system components.
- Interface model that defines public interfaces of sub-systems.
- Relationships model that shows sub-system relationships.
- Distribution model that shows how sub-systems are distributed across computers.

System organisation

- Reflects the basic strategy that is used to structure a system.
- Three organisational styles are widely used:
  - A shared data repository style;
  - A services and servers style (client/server);
  - An abstract machine or layered style.
- These styles can be used separately or together

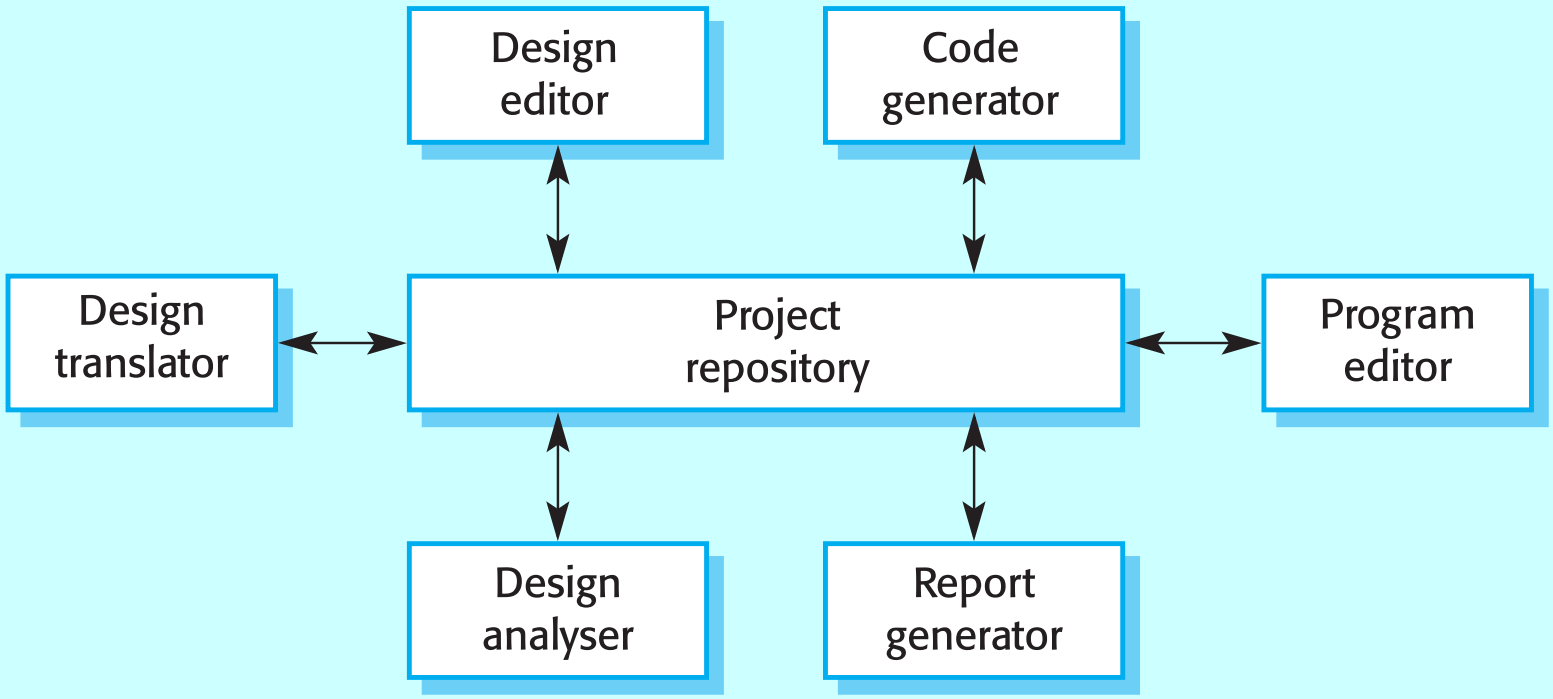
The repository model

- Two component types – data repository and data accessor
  - Data repository – provides reliable permanent storage
  - Data accessors – access data in repositories, perform computations, and may put the results back also
- Communication between data accessors is only through the repository

The repository model

- Two variations possible
  - Black board style: if data is posted in a repository, all accessors are informed; i.e. the shared data source is an active agent
  - Repository style: passive repository
- When large amounts of data are to be shared, or when data is generated and used by different sub-systems.
- E.g. database oriented systems; MIS, CASE, programming environments, C&C..

Example: CASE toolset architecture



Repository model evaluation

- Advantages
  - Efficient way to share large amounts of data;
  - Data producers need not be concerned about how the data is used.
  - Centralised data management e.g. backup, security, etc.
  - The model of sharing is visible through 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;

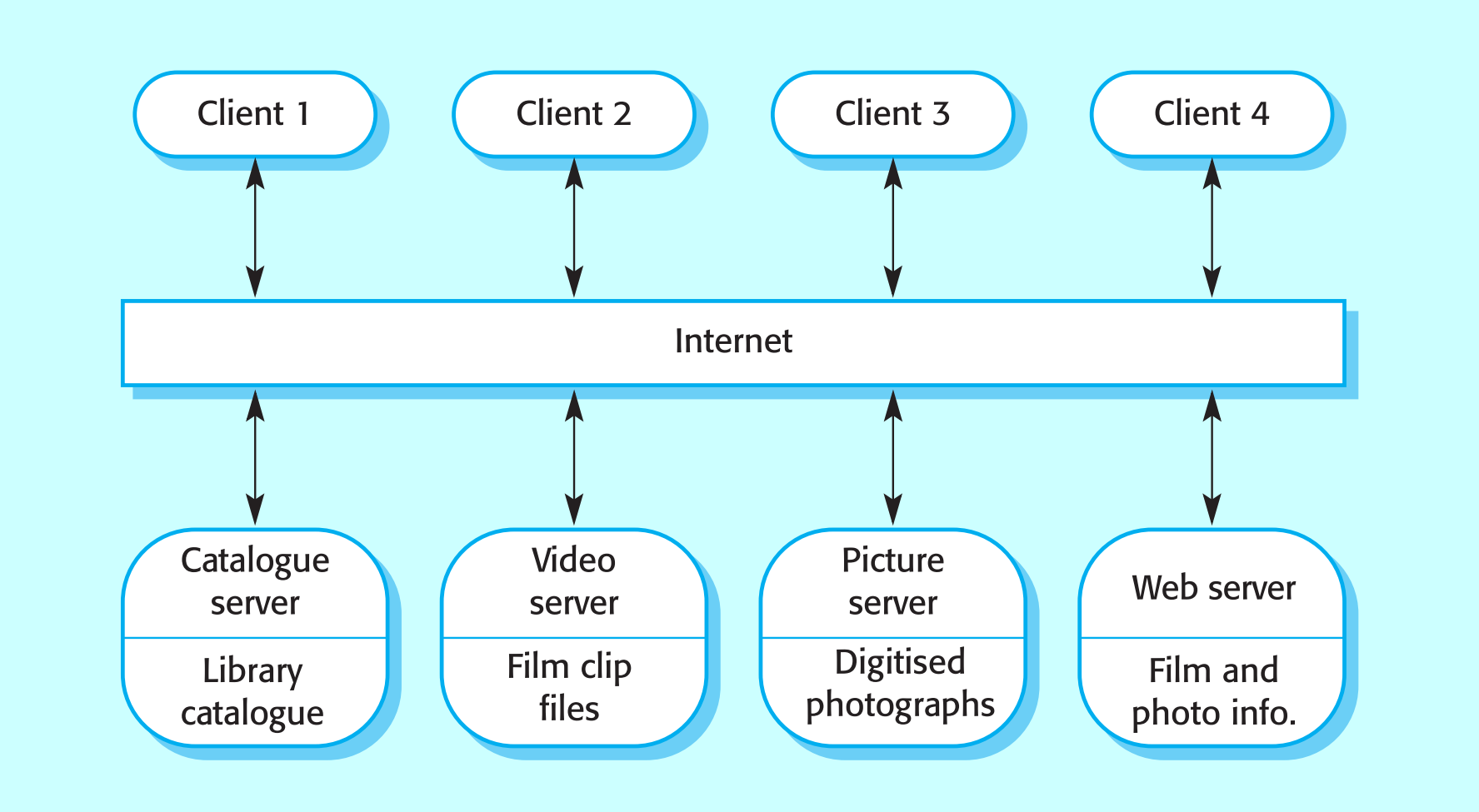
Client-server model

- Set of stand-alone servers which provide specific services such as print servers, file servers, etc. to other sub-systems.
- Set of client sub-systems which call on these services.
- Network which allows clients to access servers (if the client and server are not on the same machine).

Client-server model

- Clients may have to know the names of available servers and the services they provide.
- Servers need not know either the identity of clients or how many clients there are.
- Clients access services through remote procedure calls using a request-reply protocol such as the http protocol used in the www.
- Essentially, a client makes a request to a server and waits until it receives a reply.

Example: Film and picture library



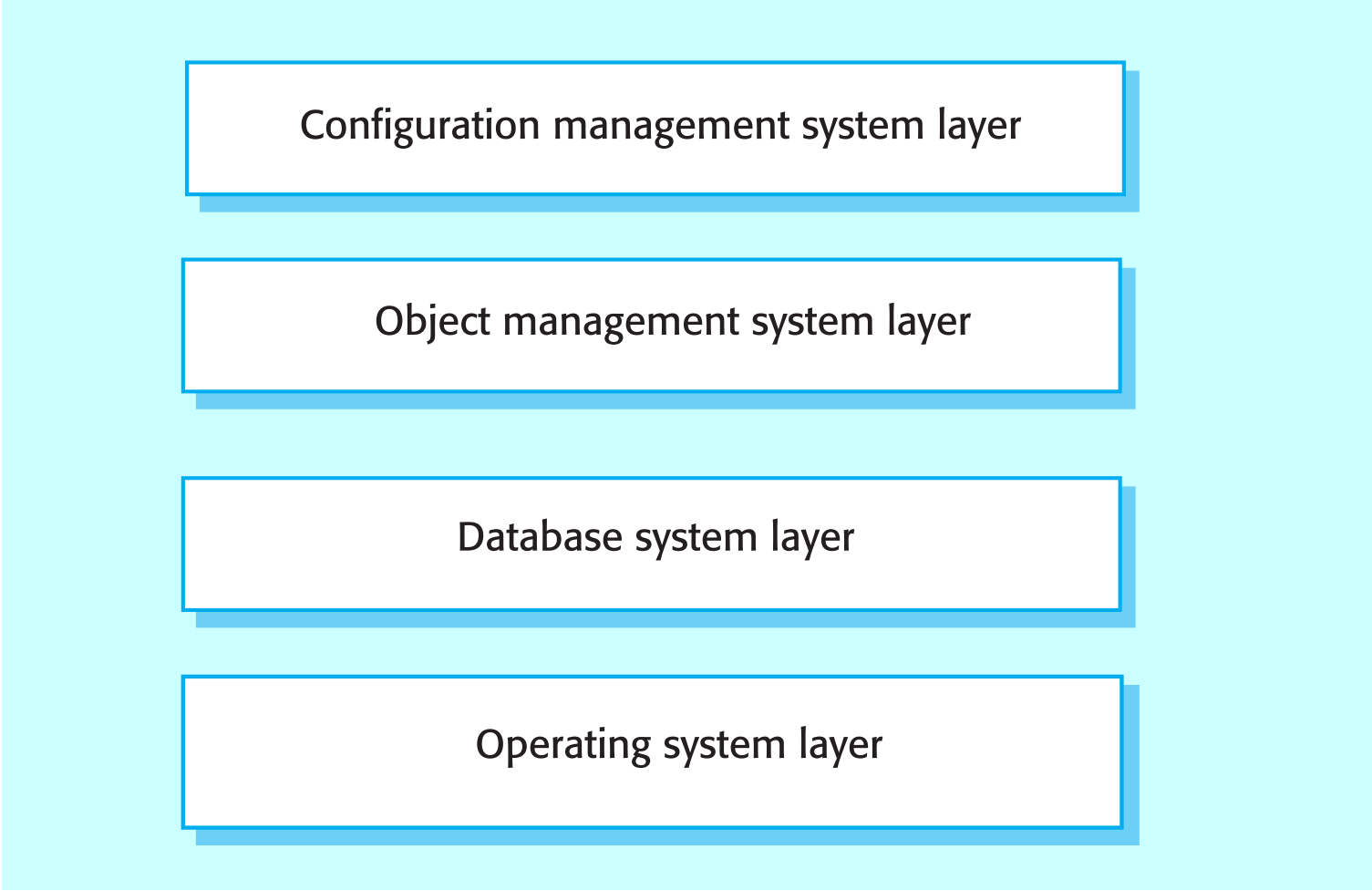
Client-server evaluation

- Advantages
  - Distribution of data is straightforward;
  - Makes effective use of distributed resources.
  - Easy to add new servers or upgrade existing servers.
- Disadvantages
  - No shared data model across servers, so specific data models may have to be established in each server to optimize its performance
  - 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 (layered) model

- 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 (some top level layers may need to access bottom layer functionalities directly).

Example: Version management system



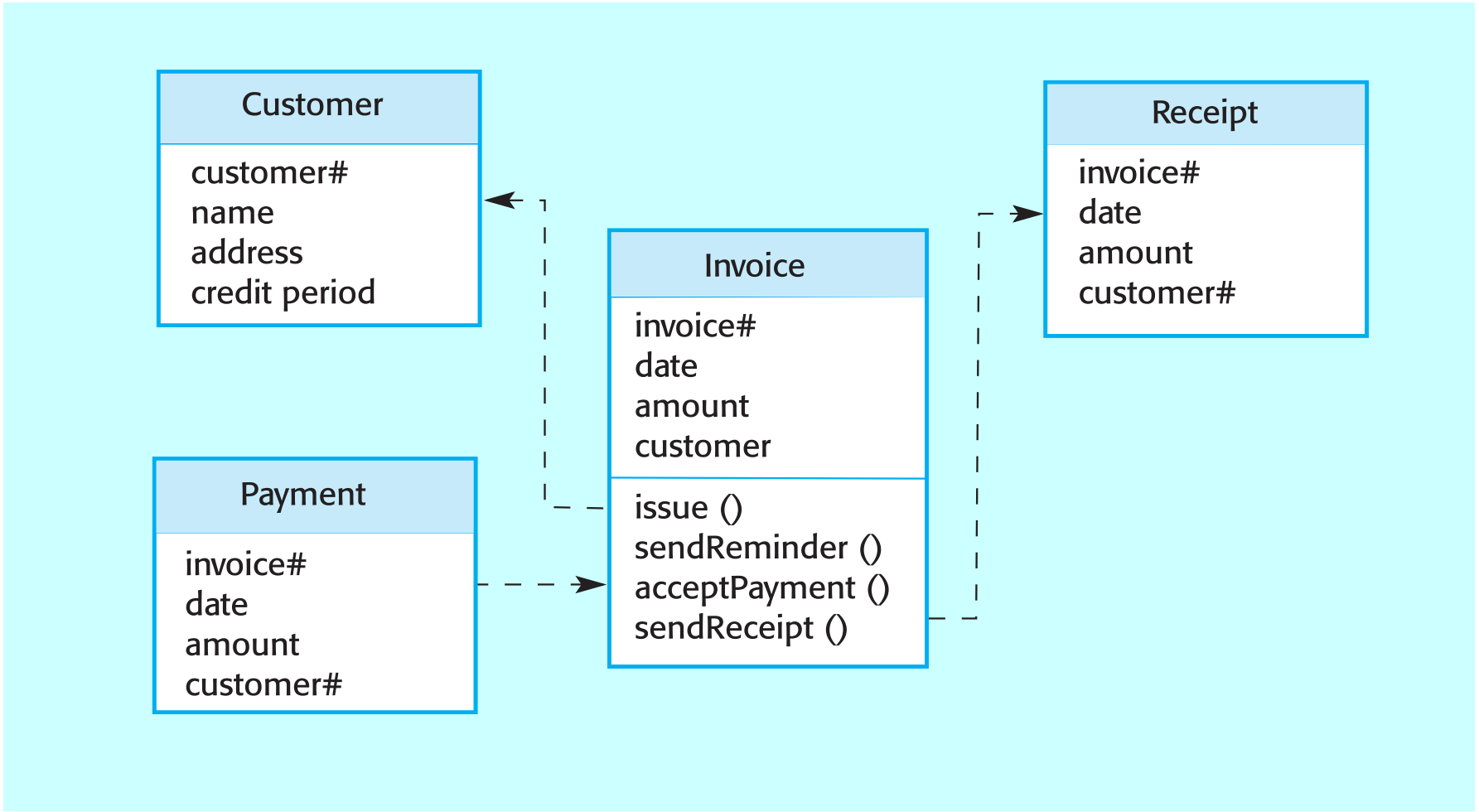
Modular decomposition styles

- Styles of decomposing sub-systems into modules/sub-systems
- Two modular decomposition models covered
  - An object oriented decomposition model where the system is decomposed into interacting objects;
  - A function-oriented pipeline model where the system is decomposed into functional modules which transform inputs to outputs.

Object oriented decomposition

- Structure the system into a set of 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.

Example: Object model of Invoice processing system



Object model evaluation

- 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
- complex entities may be hard to represent as objects.

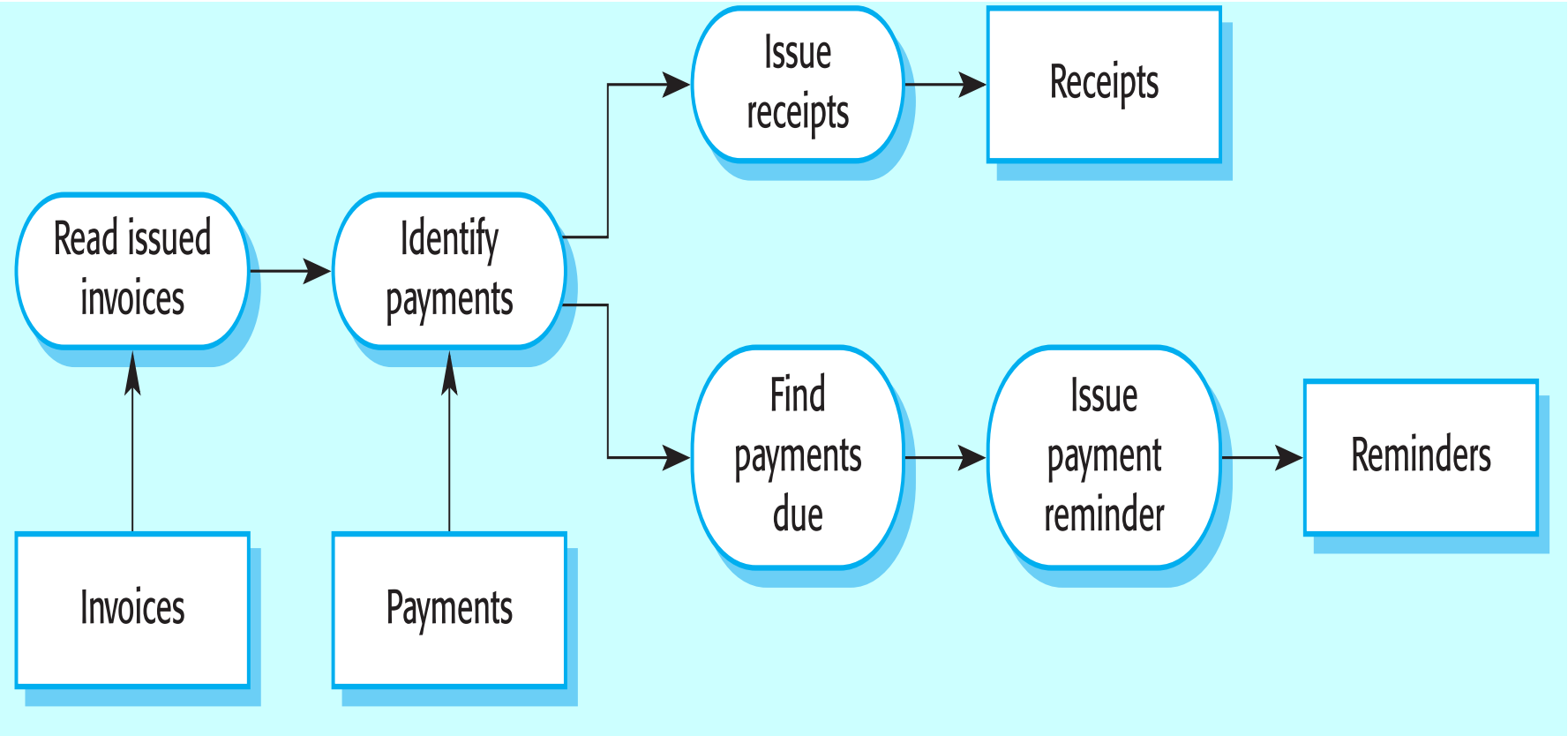
Function-oriented pipelining

- Functional transformations process their inputs to produce outputs.
- The transformations may execute sequentially or in parallel
- The data can be processed by each transform item by item or in a single batch
- Also referred to as a pipe and filter model

Function-oriented pipelining

- When transformations are sequential with data processed in batches, this architectural model is called a batch sequential model.
  - Common architecture for data-processing systems such as billing systems

Example: Pipeline model of Invoice processing system



Pipeline model evaluation

- 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
- Difficult to support event-based interaction.
- Not really suitable for interactive systems.

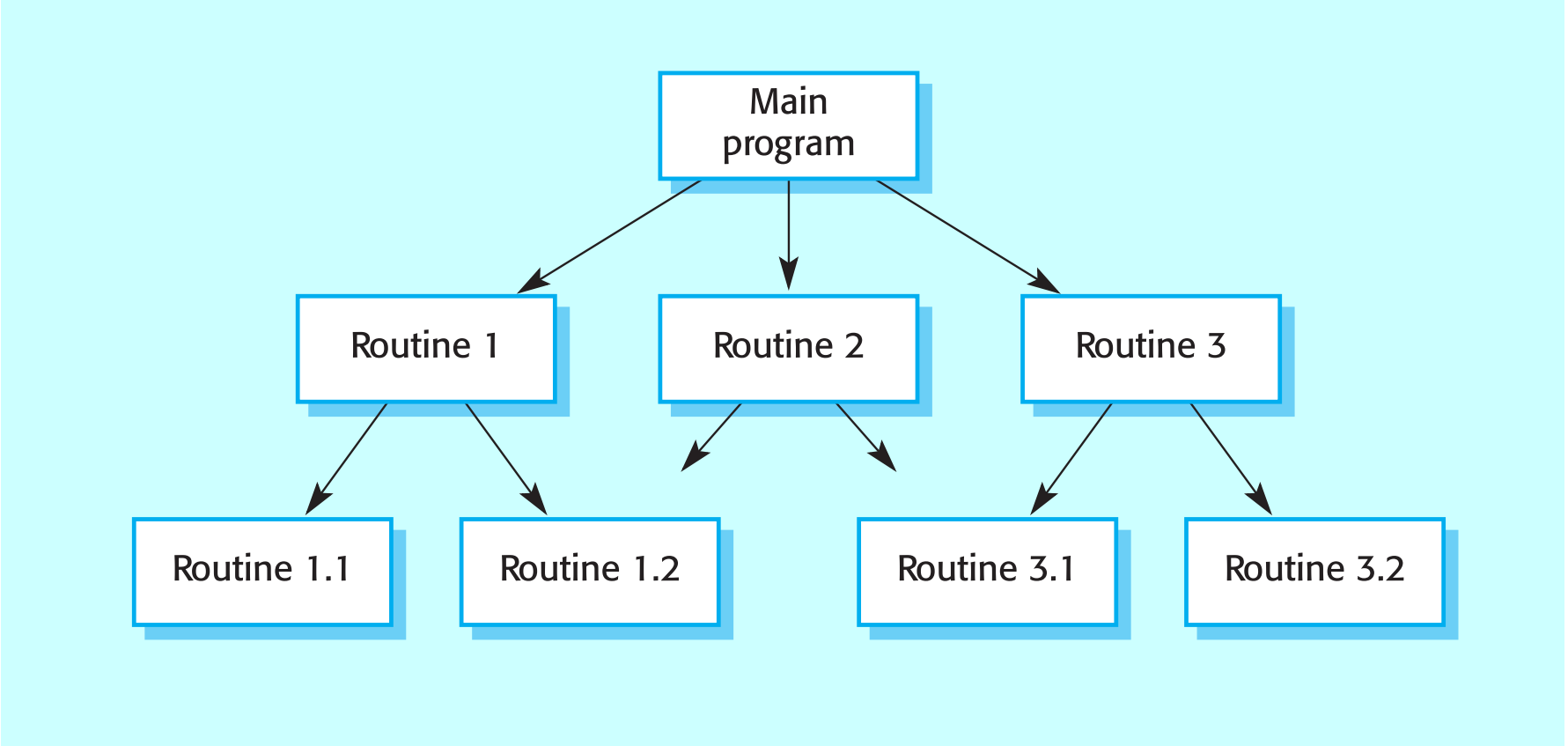
Control styles

- Concerned with the control flow between sub-systems.
- Centralised control
  - One sub-system has overall responsibility for control and starts and stops other sub-systems.
  - It may devolve control to another sub-system but will expect this control responsibility returned to it
- Event-based control
  - Each sub-system can respond to externally generated events from other sub-systems or the system's environment.

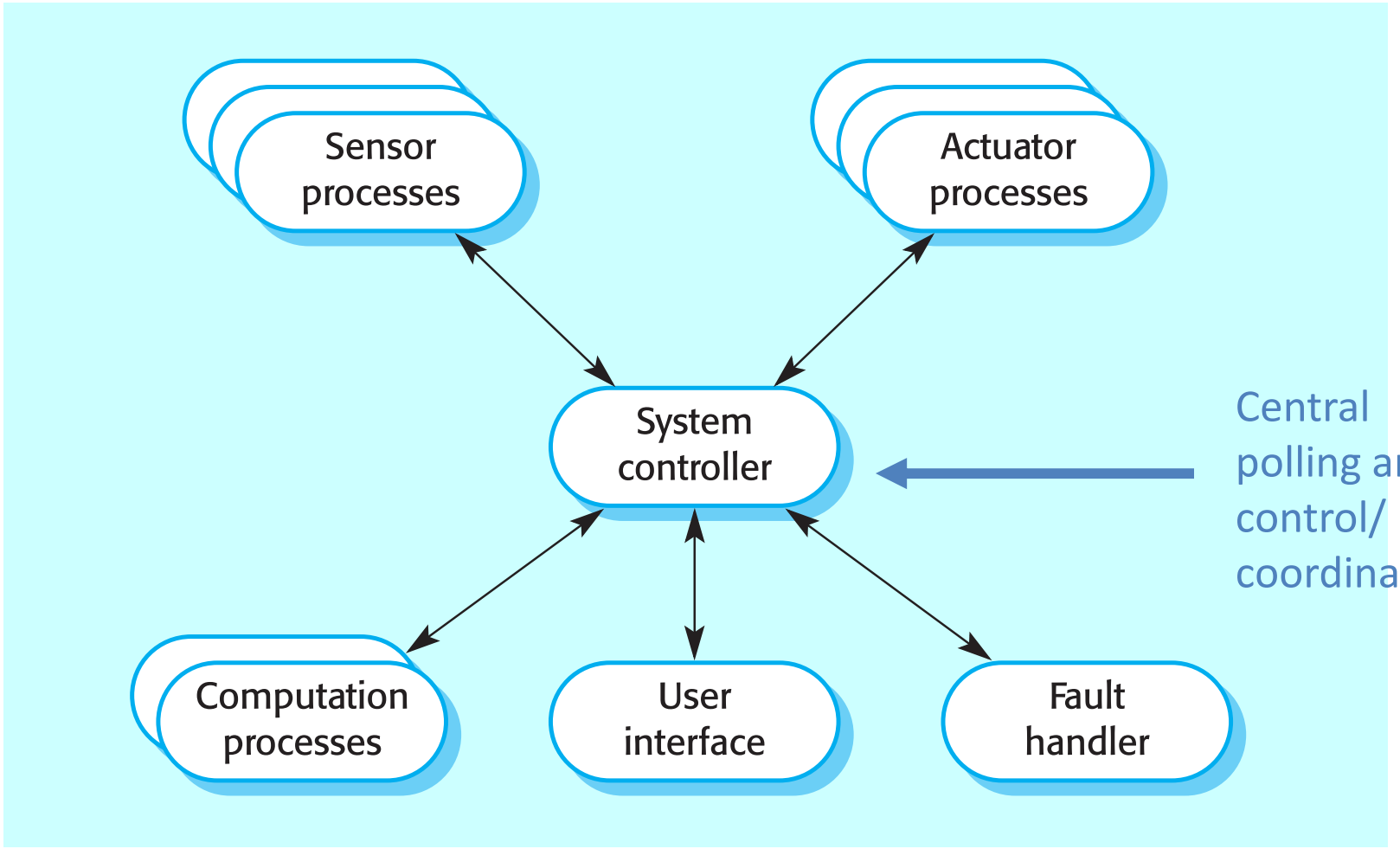
Centralised control

- Two classes, depending on whether the controlled sub-systems execute sequentially or in parallel.
- Call-return model
  - Top-down subroutine model where control starts at the top of a subroutine hierarchy and moves downwards through sub-routine calls.
  - Applicable to sequential systems.
  - Used in C, Pascal, Ada etc.
- Manager model
  - Applicable to concurrent systems.
  - One system component controls the stopping, starting and coordination of other system processes.
  - The controller loops continuously, polling other processes for events or system state changes.

Call-return model



Example: Manager model of real-time system control



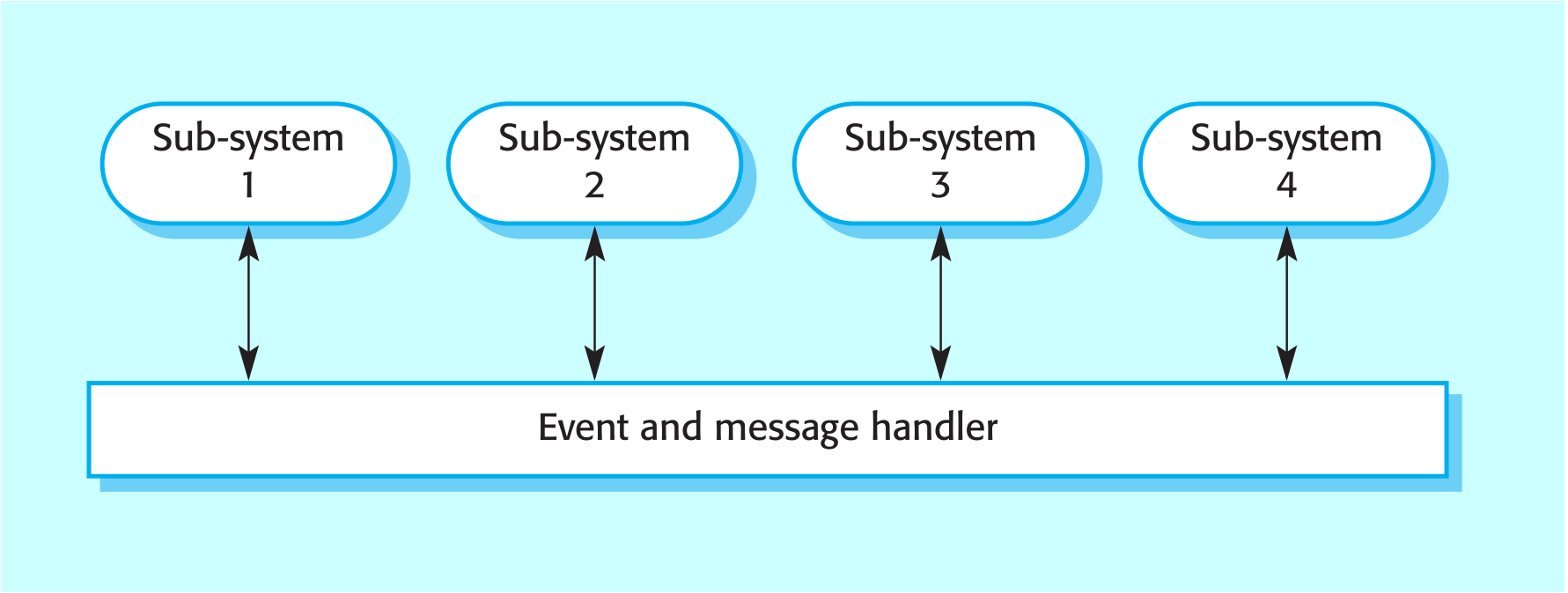
Event-based control

- While centralized control responds to system state, event-based control is driven by externally generated events where the timing of the event is outside 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.
  - Interrupt-driven models. Used in real-time systems where interrupts are detected by a central interrupt handler which passes them 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 generating the event don't know if or when the event will be handled.

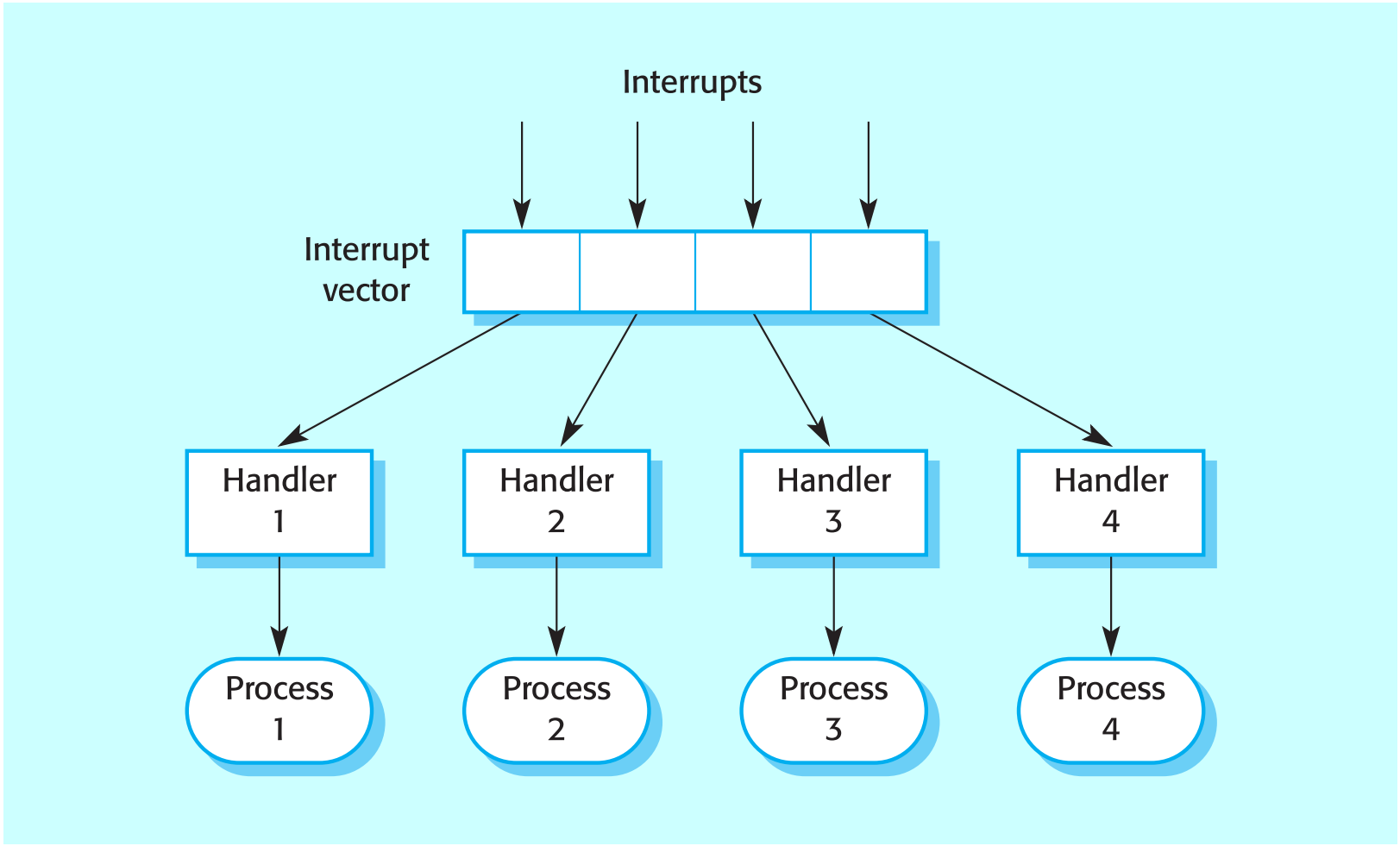
Example: Selective broadcasting



Interrupt-driven model

- 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.

Example: Interrupt-driven control



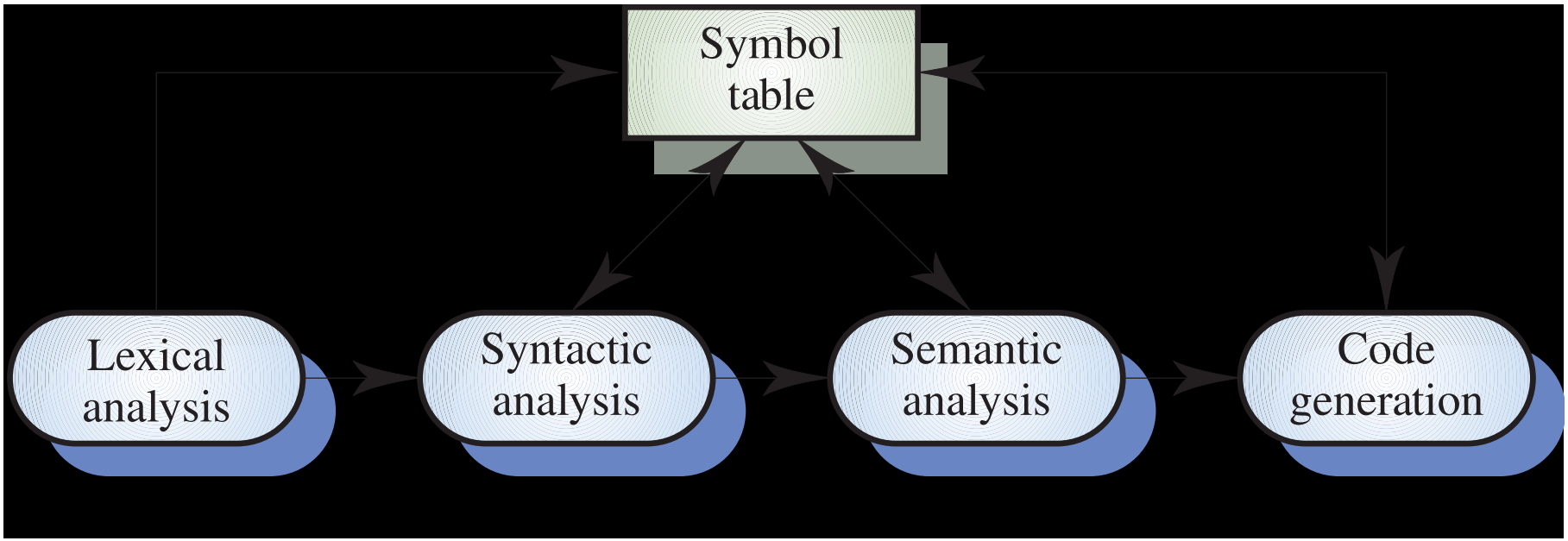
Domain-specific architectures

- Architectural models which are specific to some application domain
- Two types of domain-specific model
  - Generic models which are abstractions from a number of real systems and which encapsulate the principal characteristics of these systems. May be reused directly in design.
  - Reference models which are more abstract, idealised model of a larger class of system. Provide a means of information about that class of system and of comparing different architectures. Not normally considered a route to implementation

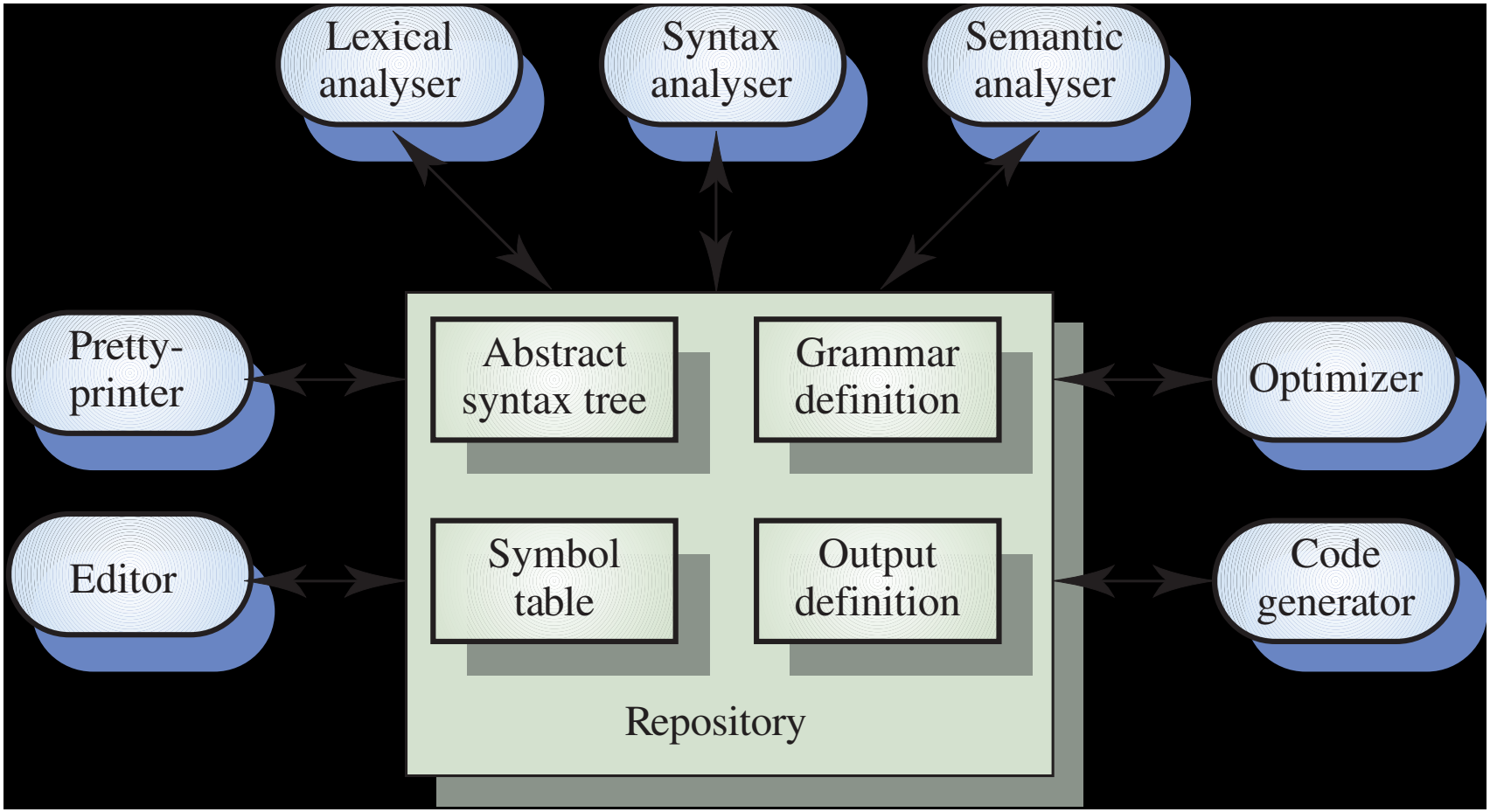
Generic models

- Compiler model is a well-known example
  - Lexical analyser
  - Symbol table
  - Syntax analyser
  - Syntax tree
  - Semantic analyser
  - Code generator
- Generic compiler model may be organised according to different architectural models

Compiler model



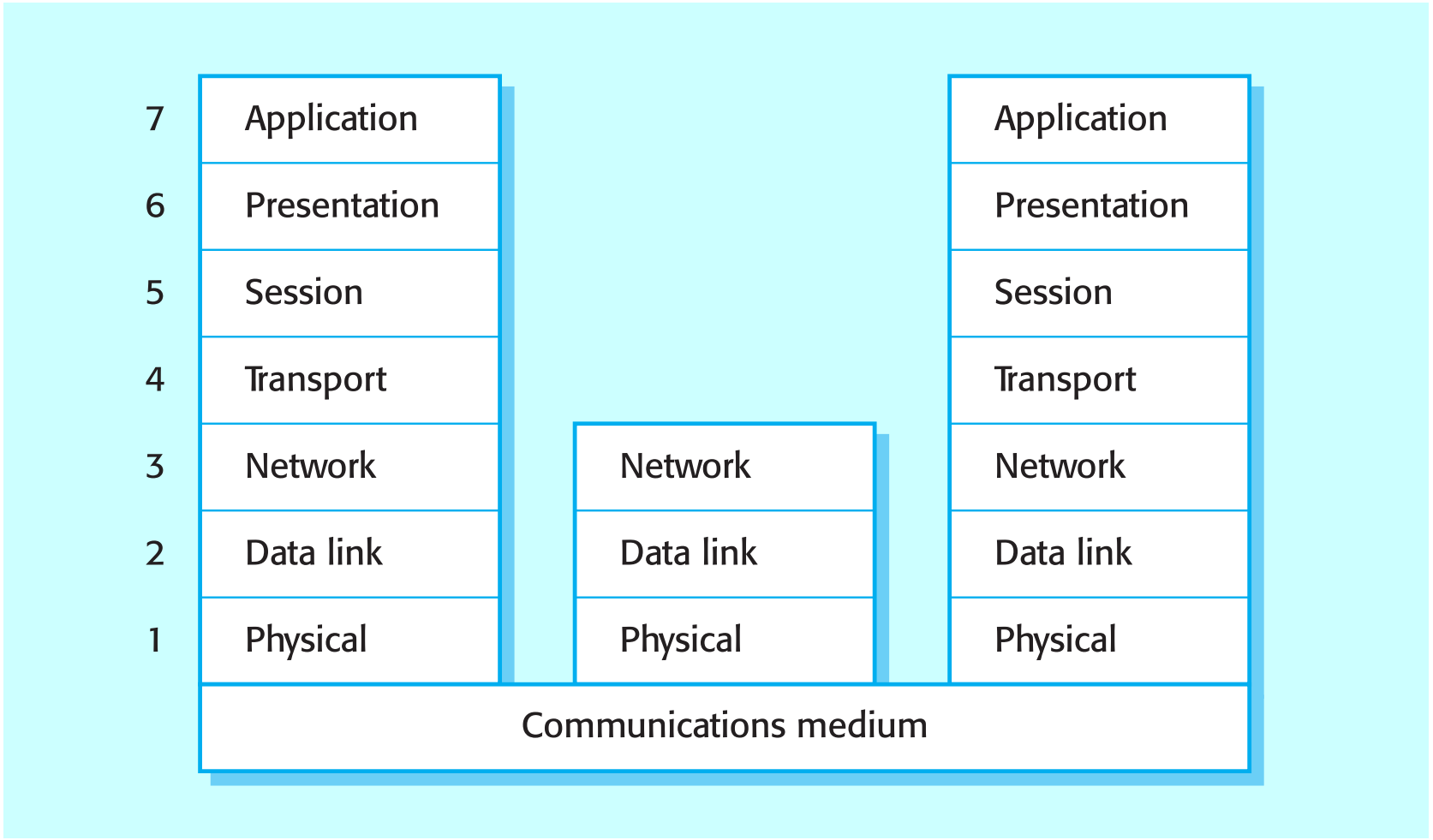
Language processing system



Reference models

- Reference models are derived from a study of the application domain rather than from existing systems.
- May be used as a standard against which systems can be evaluated.
- OSI model is a layered reference model for communication systems (not easy to implement because of problems in ensuring non-functional requirements)

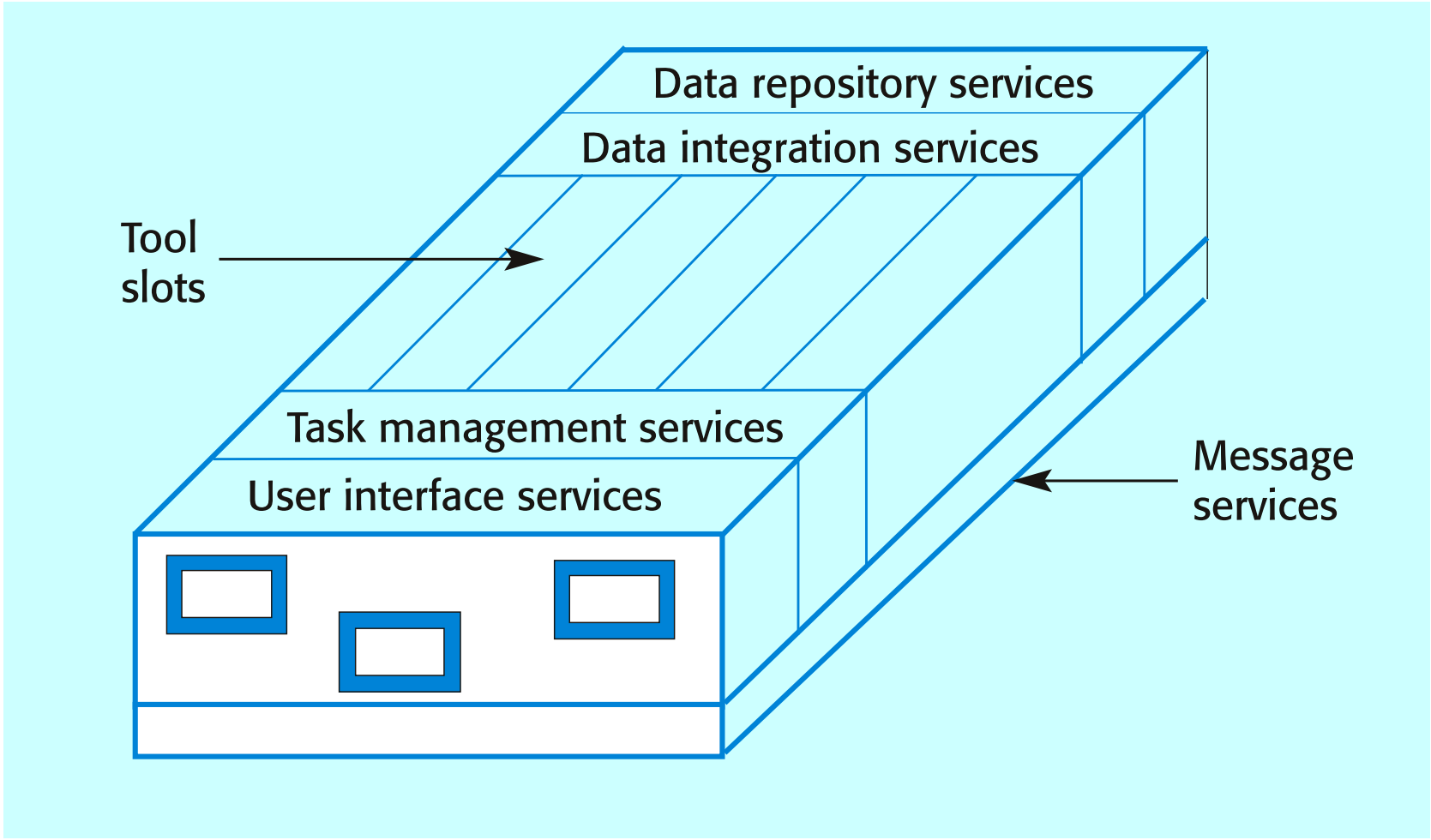
Example: OSI reference model



Example: ECMA Case reference model

- Data repository services
  - Storage and management of data items.
- Data integration services
  - Managing groups of entities.
- Task management services
  - Definition and enaction of process models.
- Messaging services
  - Tool-tool and tool-environment communication.
- User interface services
  - User interface development.

The ECMA reference model



Self Study

- Read IEEE-1471

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.

Key points

- Modular decomposition models include object models and pipelining models.
- Control models include centralised control and event-driven models.
- Reference architectures may be used to communicate domain-specific architectures and to assess and compare architectural designs.