unit3

Topics covered

- Architectural design decisions
- System organisation
- Decomposition styles
- Control styles
- Reference architectures
- User interface

Architectural design concept

- An early stage of the system design process.
- It is about decomposing of system into interacting components
- Represents the link between specification and design processes.
- Expressed as a block diagram
- It involves identifying major system components and their communications.

Advantages

- Stakeholder communication
  - Architecture may be used as a focus of discussion by system stakeholders.
- System analysis
  - Means that 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.

System properties focused in A.D

Performance

Meantime between request and response. Performance can be improved by avoiding critical operations and reducing communications between components . this is possible by building large components rather than small fine grain components

Security

Use a layered architecture with critical assets in the inner layers.

Safety

Avoid critical functionalities in small components of the system.

Availability

Include redundant components and mechanisms for fault tolerance.

Maintainability

Use replaceable components.

Disadvantages

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

➢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?
- 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?

➢System organization

- Reflects the basic strategy that is used to structure a system.
- Three organizational styles are widely used:
  - Repository model;
  - Client & server model ;
  - An abstract or layered model.

The repository model

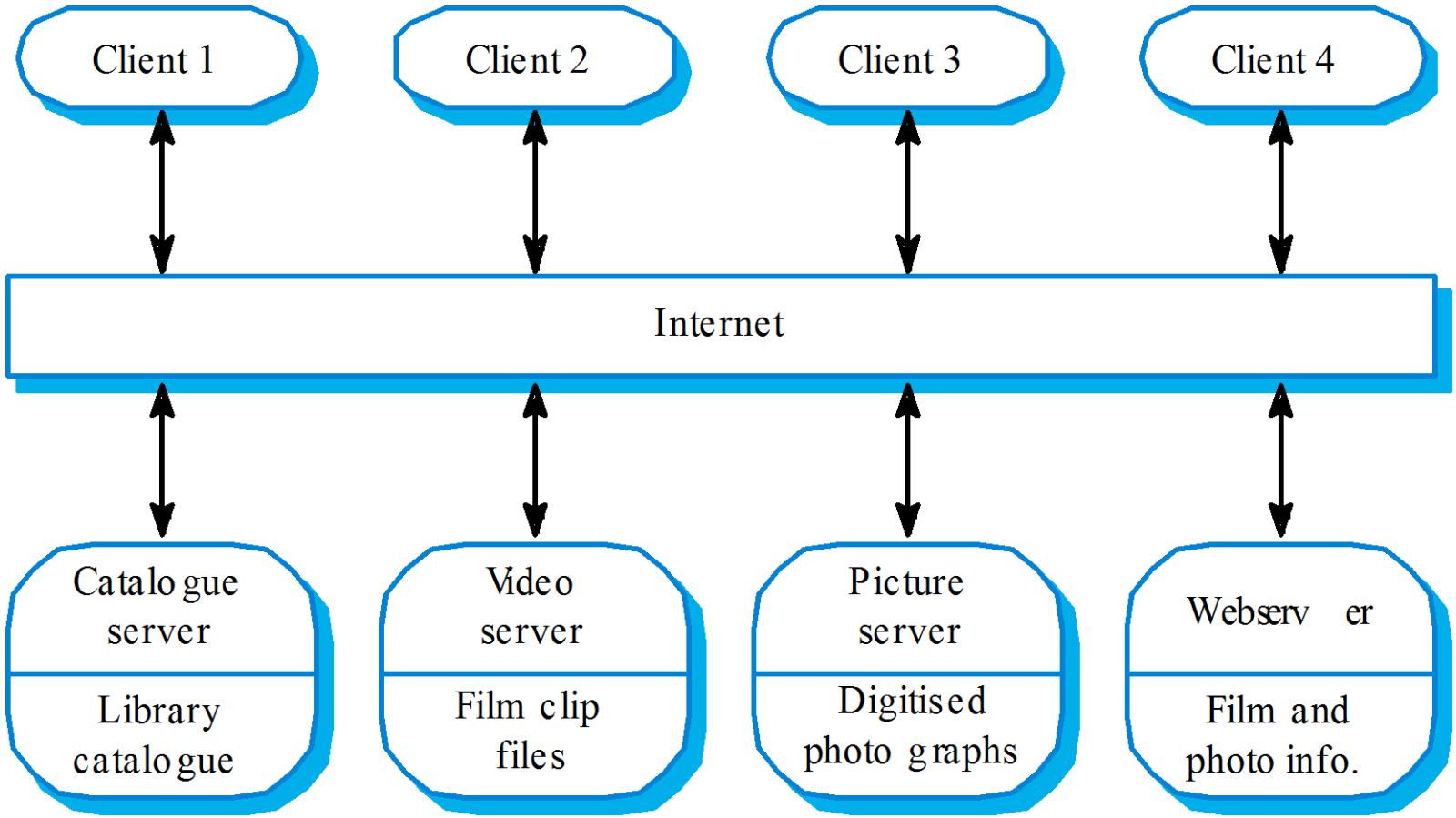
- Used when large amounts of data are to be shared.
- Sub-systems must exchange data.
- This may be done in two ways:
  - **Shared data** is held in a central database or repository and may be accessed by all sub-systems;
  - **Each sub-system** maintains its own database and passes data explicitly to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

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.
  - Data evolution is difficult and expensive;
  - No scope for specific management policies;
  - Difficult to distribute efficiently.

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.


Client 1
Client 2
Client 3
Client 4
Internet
Catalogue server
Library catalogue
Video server
Film clip files
Picture server
Digitised photo graphs
Webserv er
Film and photo info.

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 (layered) model

- Used to model the communication among 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.

➢Modular decomposition styles

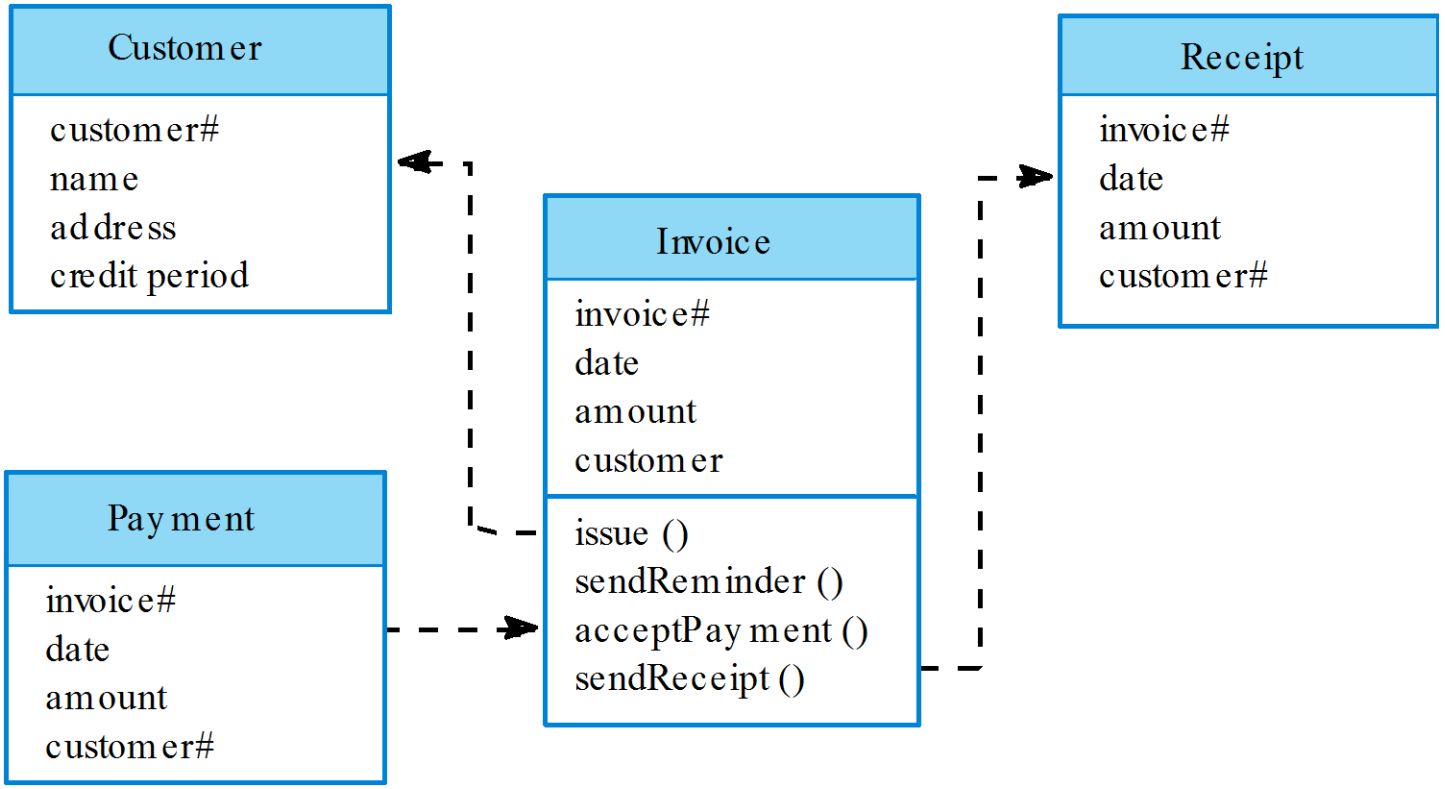
- Styles of decomposing sub-systems into modules.

- 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

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.

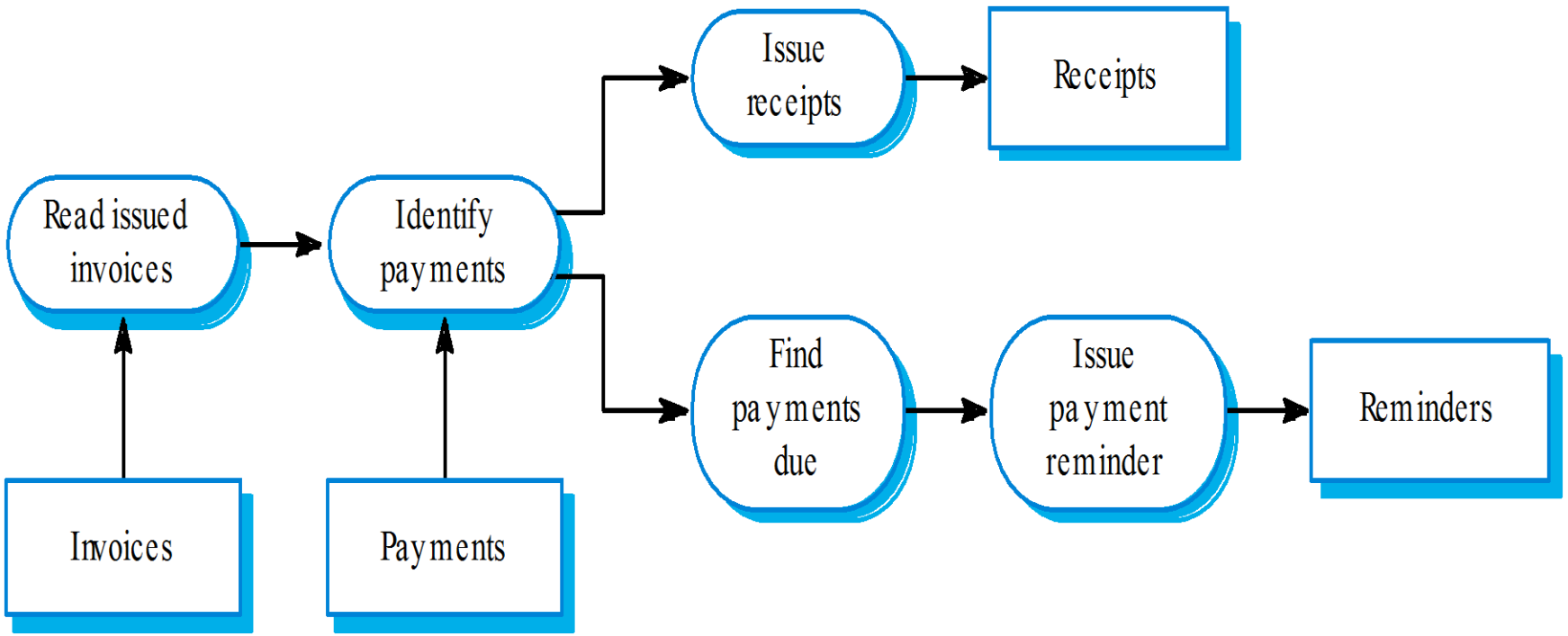
Invoice processing system



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



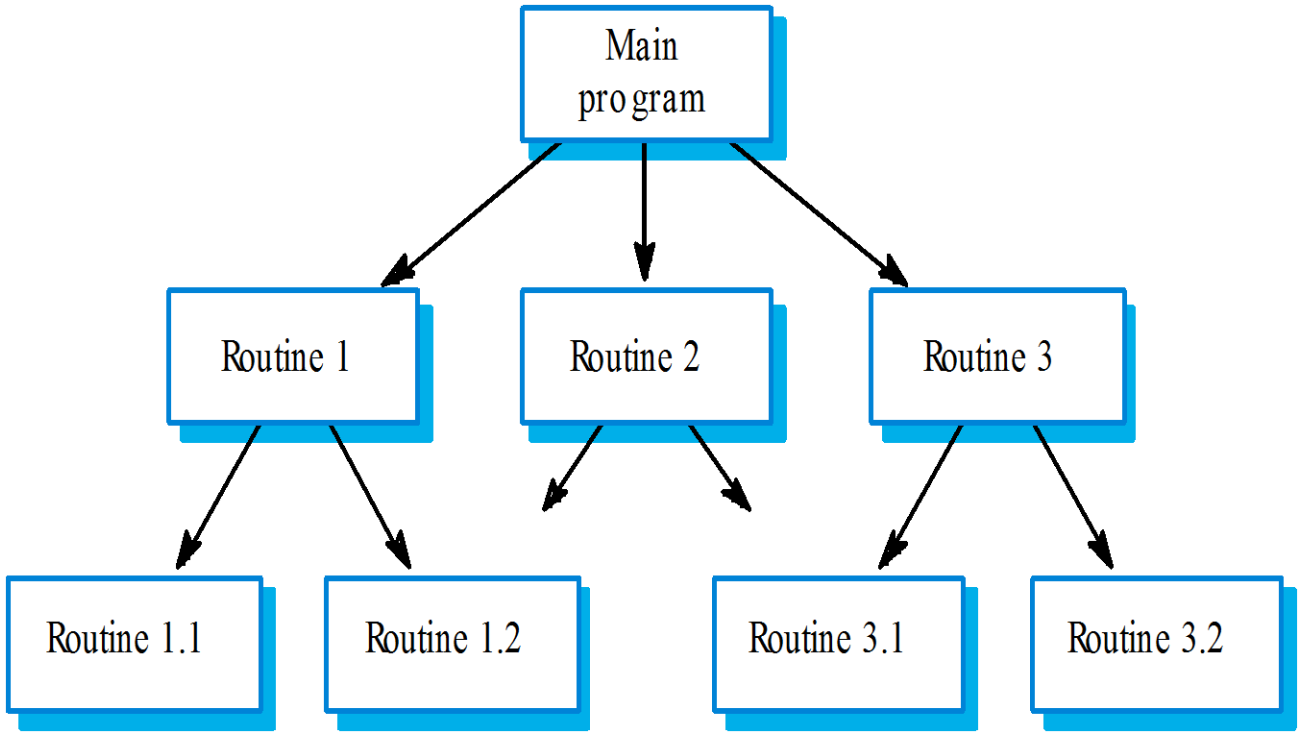
Pipeline model advantages

- 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.

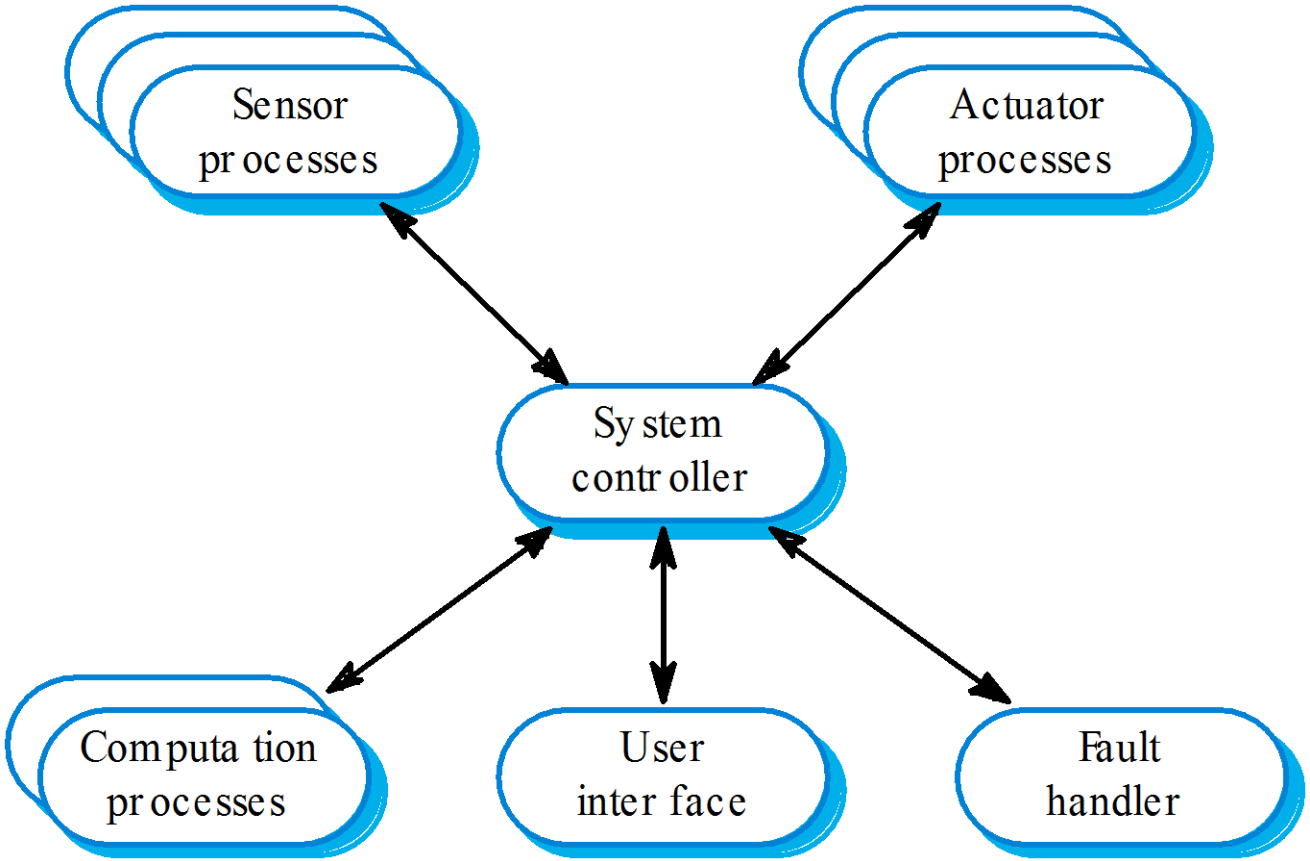
Control styles

- Are 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.
  - 2 types
  - Call return
  - Manager control
- Event-based control
  - Each sub-system can respond to externally generated events from other sub-systems or the system's environment.
  - 2 types
  - Broadcast model

Call-return model



Manager control model



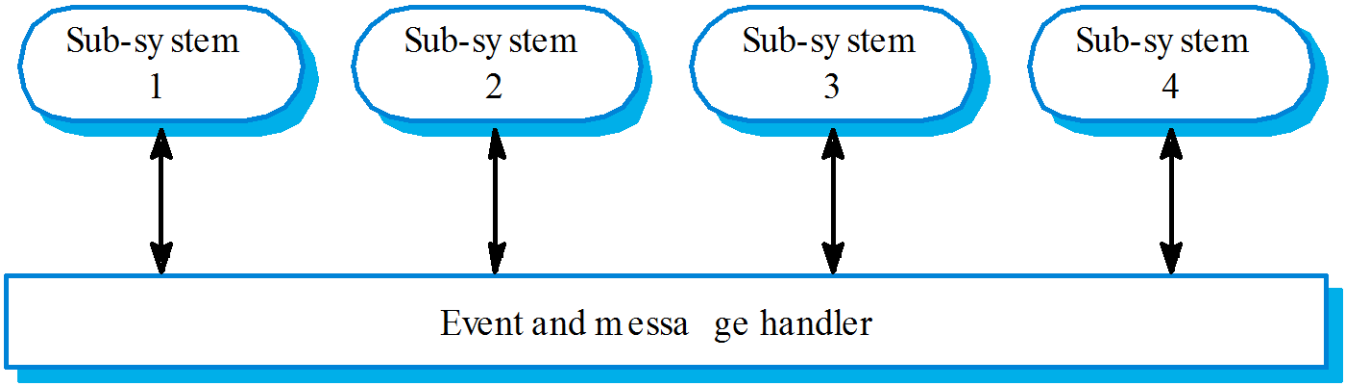
Event-driven systems

- Each subsystem is allowed to respond to the events that are externally generated by other sub-system or system environment
- 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.
- Other event driven models include spreadsheets and production systems.

Broadcast model

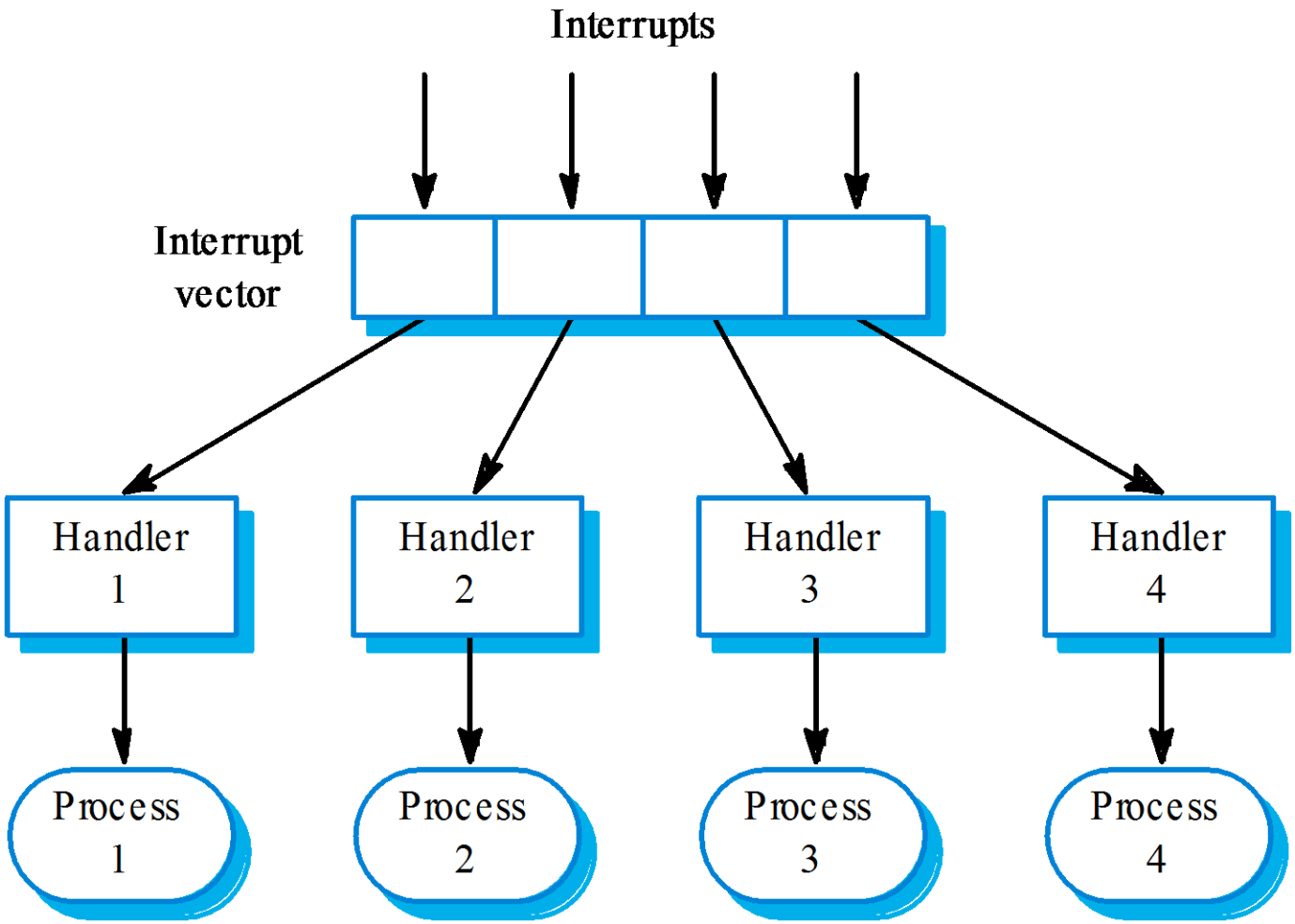
- In this event is broadcasted . any system which can respond to external event does:
- Effective in integrating sub-systems on different computers in a network.
- Sub-systems register an interest in specific events. When these occur, control is transferred to the sub-system which can handle the event.
- 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

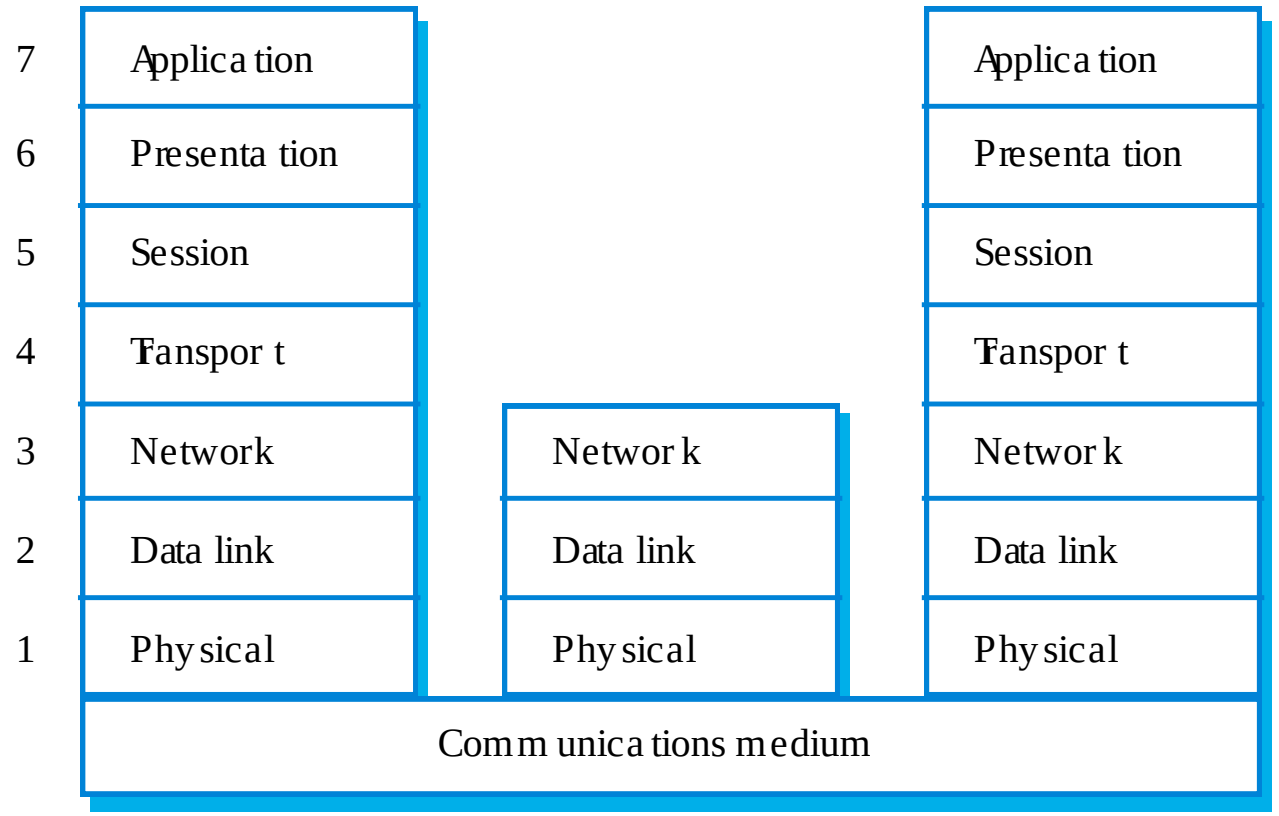
- Interrupts are identified by an interrupt handler and passed to another component for processing
- Used in real-time systems where fast response to an event is essential.
- There are known interrupt types with a handler defined for each type.
- Each type is associated with a memory location and a hardware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


Interrupts
Interrupt vector
Handler 1
Handler 2
Handler 3
Handler 4
Process 1
Process 2
Process 3
Process 4

Reference architectures

- Architectural models may be 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.
  - **Reference models** which are more abstract, idealised model. Provide a means of information about that class of system and of comparing different architectures.
- Generic models are usually bottom-up models;
- Reference models are top-down models.

OSI reference model



Application architecture

- Set of structures that defines a system
- Types
  - Data processing systems
  - Transaction processing systems
  - Event processing systems
  - Language processing systems

- Data processing applications
  - Data driven applications that process data in batches without explicit user intervention during the processing.
- Transaction processing applications
  - Data-centred applications that process user requests and update information in a system database.
- Event processing systems
  - Applications where system actions depend on interpreting events from the system's environment.
- Language processing systems
  - Applications where the users' intentions are specified in a formal language that is processed and interpreted by the system.

Application type examples

- Data processing systems
  - Billing systems;
  - Payroll systems.
- Transaction processing systems
  - E-commerce systems;
  - Reservation systems.
- Event processing systems
  - Word processors;
  - Real-time systems.
- Language processing systems
  - Compilers;
  - Command interpreters.

Data processing systems

- Systems that are data-centered where the databases used are usually orders of magnitude larger than the software itself.
- Data is input and output in batches
  - Input: A set of customer numbers and associated readings of an electricity meter;
  - Output: A corresponding set of bills, one for each customer number.
- Data processing systems usually have an input-process-output structure.

- The input component reads data from a file or database, checks its validity and queues the valid data for processing.
- The process component takes a transaction from the queue (input), performs computations and creates a new record with the results of the computation.
- The output component reads these records, formats them accordingly and writes them to the database or sends them to a printer.

Transaction processing systems

- Process user requests for information from a database or requests to update the database.
- From a user perspective a transaction is:
  - Any coherent sequence of operations that satisfies a goal;
  - For example - find the times of flights from London to Paris.
- Users make asynchronous requests for service which are then processed by a transaction manager.

Event processing systems

- These systems respond to events in the system's environment.
- Their key characteristic is that event timing is unpredictable so the architecture has to be organised to handle this.
- Many common systems such as word processors, games, etc. are event processing systems.

Language processing systems

- Accept a natural or artificial language as input and generate some other representation of that language.
- May include an interpreter to act on the instructions in the language that is being processed.
- Used in situations where the easiest way to solve a problem is to describe an algorithm or describe the system data
  - Meta-case tools process tool descriptions, method rules, etc and generate tools.

➢User interface

- Good programming technology is not enough to achieve users acceptance of software product
- It also needs good user interface else it fails in the market
- UI refers to the products where a user interacts with controls or display of the product
- A good UI design can make a product to be accepted or rejected in the market.
- If end user feel it difficult to learn and use then even an excellent product could fail

User interface design issues

Following are the few issues need to be considered while designing a good user interface.

Response time

- This is mean time between request and response of the software with desired o/p

Help facilities

- Required in 2 cases 1)when he need some info and unable to find 2)if end user is in trouble

Error handling

- Should describe the problem in words that are understood by the user

Menu and command handling

- In menu user makes selection by clicking mouse and in command user types command to give instruction to the system

UI design process

- User interface development follows a spiral process
  - Interface analysis (user, task, and environment analysis)
    - Focuses on the end users who will interact with the system
    - Concentrates on users, tasks, content and work environment
    - Studies different models of system function (as perceived from the outside)
    - Delineates the human- and computer-oriented tasks that are required to achieve system function
  - Interface design
    - Defines a set of interface objects and actions (and their screen representations) that enable a user to perform all defined tasks in a manner that meets every usability goal defined for the system

- Interface construction
  - Begins with a prototype that enables usage scenarios to be evaluated
  - Continues with development tools to complete the construction
- Interface validation, focuses on
  - The ability of the interface to implement every user task correctly, to accommodate all task variations, and to achieve all general user requirements
  - The degree to which the interface is easy to use and easy to learn
  - The users' acceptance of the interface as a useful tool in their work

UI Design principles

- **User familiarity** The interface should use terms and concepts which are drawn from the experience of the people who will make most use of the system.
- **Consistency** The interface should be consistent in that, wherever possible, comparable operations should be activated in the same way.
- **Minimal surprise** Users should never be surprised by the behavior of a system.
- **Recoverability** The interface should include mechanisms to allow users to recover from errors.
- **User guidance** The interface should provide meaningful feedback when errors occur and provide context-sensitive user help facilities.
- **User diversity** The interface should provide appropriate interaction facilities for different types of system user.