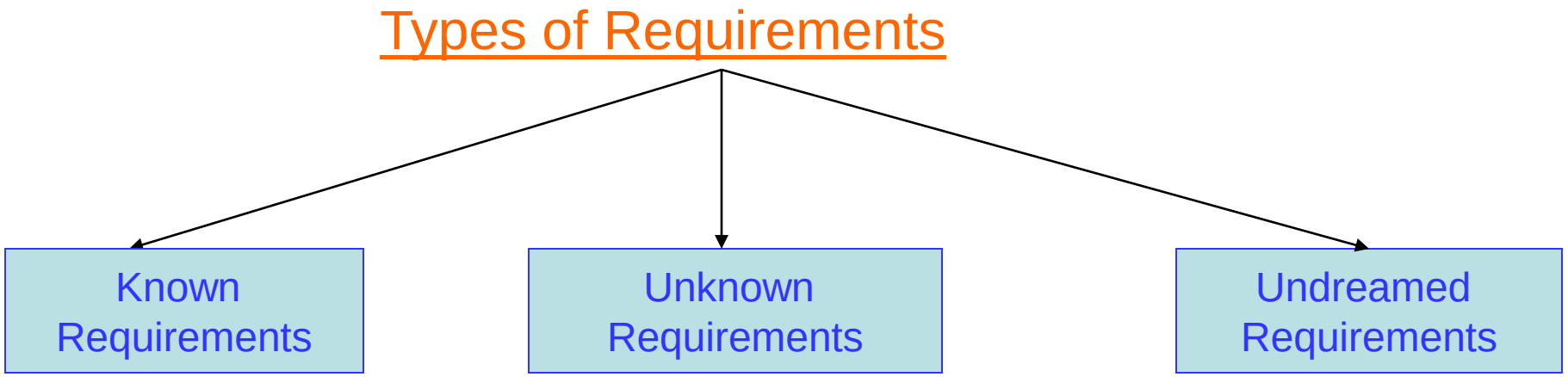
Requirement Analysis & Specification

UNIT -2

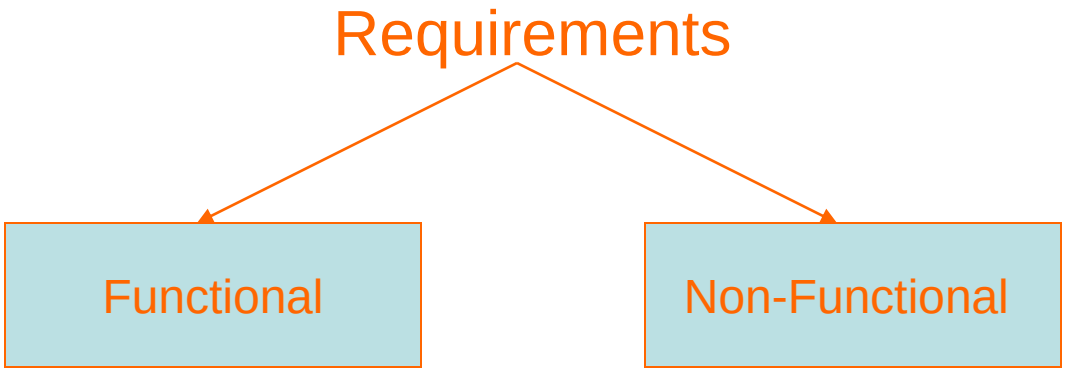
Types of Requirements



Stakeholder: Anyone who should have some direct or indirect influence on the system requirements.

--- User

--- Affected persons



Types of Requirements

Functional requirements describe what the software has to do. They are often called product features.

Non Functional requirements are mostly quality requirements. That stipulate how well the software does, what it has to do.

Availability
Reliability
Usability
Flexibility

} For Users

Maintainability
Portability
Testability

} For Developers

Types of Requirements

User and system requirements

- User requirement are written for the users and include functional and non functional requirement.
- System requirement are derived from user requirement.
- The user system requirements are the parts of software requirement and specification (SRS) document.

Types of Requirements

Interface Specification

- Important for the customers.

TYPES OF INTERFACES

- Procedural interfaces (also called Application Programming Interfaces (APIs)).
- Data structures
- Representation of data.

Feasibility Study

Is cancellation of a project a bad news?

As per IBM report, “31% projects get cancelled before they are completed, 53% over-run their cost estimates by an average of 189% & for every 100 projects, there are 94 restarts.

How do we cancel a project with the least work?

➡ CONDUCT A FEASIBILTY STUDY

Technical feasibility

- Is it technically feasible to provide direct communication connectivity through space from one location of globe to another location?

- Is it technically feasible to design a programming language using “Sanskrit”?

Feasibility Study

Feasibility depends upon non technical Issues like:

- Are the project's cost and schedule assumption realistic?
- Does the business model realistic?
- Is there any market for the product?

Feasibility Study

Purpose of feasibility study

"evaluation or analysis of the potential impact of a proposed project or program."

Focus of feasibility studies

- Is the product concept viable?
- Will it be possible to develop a product that matches the project's vision statement?
- What are the current estimated cost and schedule for the project?

Feasibility Study

Focus of feasibility studies

- How big is the gap between the original cost & schedule targets & current estimates?
- Is the business model for software justified when the current cost & schedule estimate are considered?
- Have the major risks to the project been identified & can they be surmounted?
- Is the specifications complete & stable enough to support remaining development work?

Focus of feasibility studies

- Have users & developers been able to agree on a detailed user interface prototype? If not, are the requirements really stable?
- Is the software development plan complete & adequate to support further development work?

Requirements Elicitation

Perhaps

- Most difficult
- Most critical
- Most error prone
- Most communication intensive

Succeed

effective customer developer partnership

Selection of any method

1. It is the only method that we know
2. It is our favorite method for all situations
3. We understand intuitively that the method is effective in the present circumstances.

Normally we rely on first two reasons.

Requirements Elicitation

1. Interviews

Both parties have a common goal

--- open ended
--- structured

} Interview

Success of the project

Selection of stakeholder

1. Entry level personnel
2. Middle level stakeholder
3. Managers
4. Users of the software (Most important)

Requirements Elicitation

Types of questions.

- Any problems with existing system
- Any Calculation errors
- Possible reasons for malfunctioning
- No. of Student Enrolled

Requirements Elicitation

5. Possible benefits
6. Satisfied with current policies
7. How are you maintaining the records of previous students?
8. Any requirement of data from other system
9. Any specific problems
10. Any additional functionality
11. Most important goal of the proposed development

At the end, we may have wide variety of expectations from the proposed software.

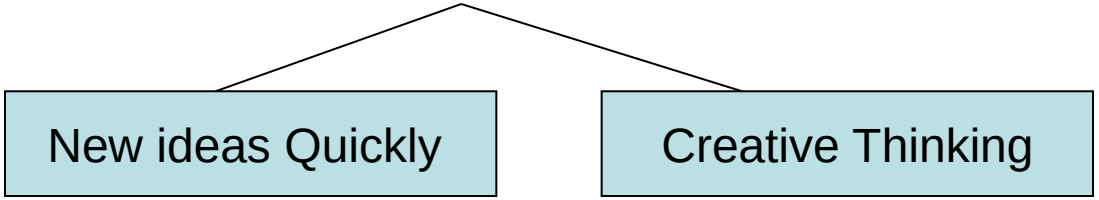
Requirements Elicitation

2. Brainstorming Sessions

It is a group technique

↓

group discussions



Prepare long list of requirements

↓

Categorized
Prioritized
Pruned

*Idea is to generate views ,not to vet them.

Groups

1. Users 2. Middle Level managers 3. Total Stakeholders

Requirements Elicitation

A Facilitator may handle group bias, conflicts carefully.

-- Facilitator may follow a published agenda

-- Every idea will be documented in a way that everyone can see it.

--A detailed report is prepared.

3. Facilitated Application specification Techniques (FAST)

-- Similar to brainstorming sessions.

-- Team oriented approach

-- Creation of joint team of customers and developers.

Requirements Elicitation

Guidelines

1. Arrange a meeting at a neutral site.
2. Establish rules for participation.
3. Informal agenda to encourage free flow of ideas.
4. Appoint a facilitator.
5. Prepare definition mechanism board, worksheets, wall stickier.
6. Participants should not criticize or debate.

FAST session Preparations

Each attendee is asked to make a list of objects that are:

Requirements Elicitation

1. Part of environment that surrounds the system.
2. Produced by the system.
3. Used by the system.

A. List of constraints
B. Functions
C. Performance criteria

Activities of FAST session

1. Every participant presents his/her list
2. Combine list for each topic
3. Discussion
4. Consensus list
5. Sub teams for mini specifications
6. Presentations of mini-specifications
7. Validation criteria
8. A sub team to draft specifications

Requirements Elicitation

4. Quality Function Deployment

-- Incorporate voice of the customer

Technical requirements.

Documented

Prime concern is customer satisfaction

What is important for customer?

-- Normal requirements

-- Expected requirements

-- Exciting requirements

Requirements Elicitation

Steps

1. Identify stakeholders
2. List out requirements
3. Degree of importance to each requirement.

Requirements Elicitation

5 Points : V. Important

4 Points : Important

3 Points : Not Important but nice to have

2 Points : Not important

1 Points : Unrealistic, required further exploration

Requirement Engineer may categorize like:

(i) It is possible to achieve

(ii) It should be deferred & Why

(iii) It is impossible and should be dropped from consideration

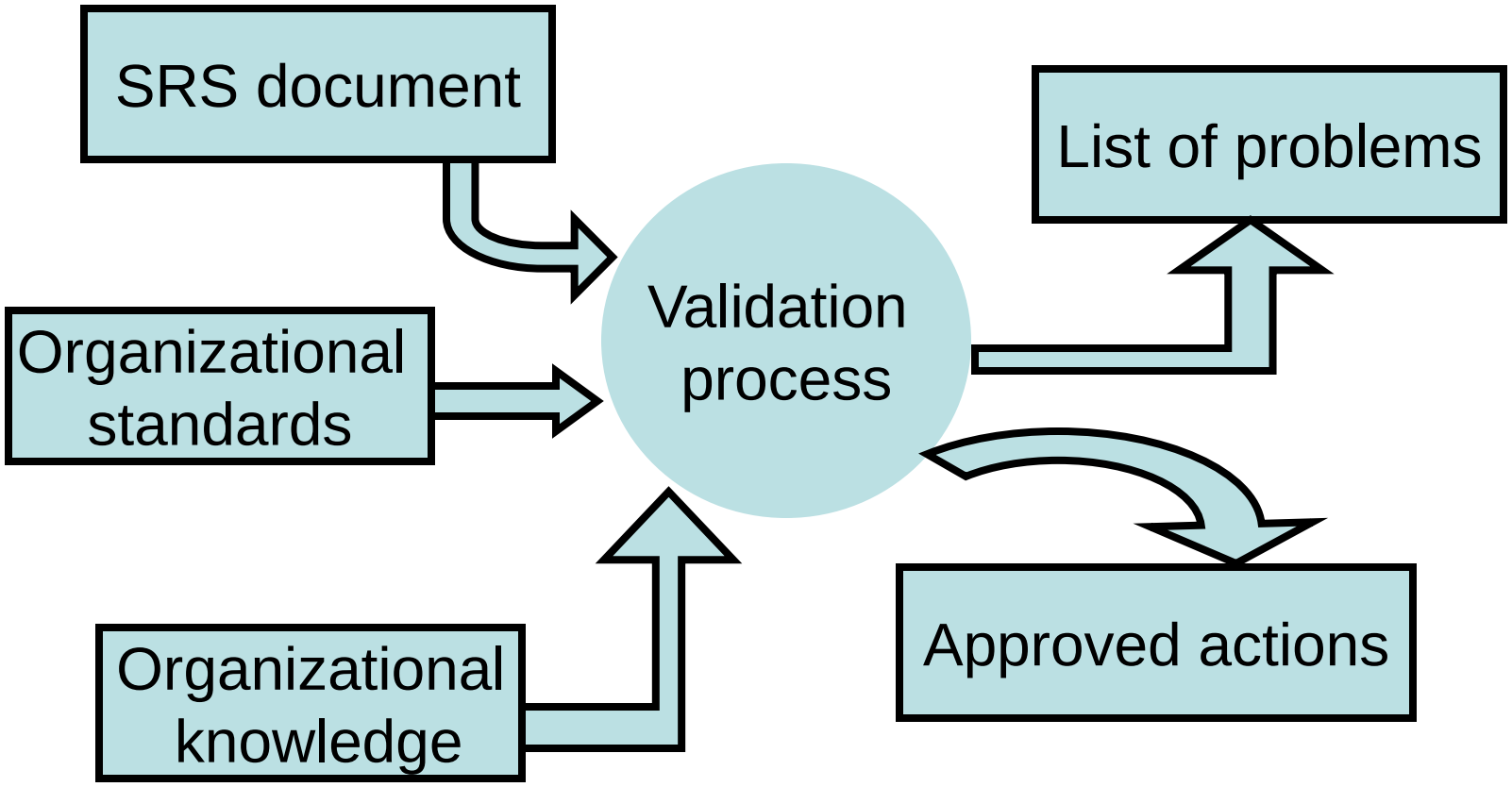
First Category requirements will be implemented as per priority assigned with every requirement.

Requirements Validation

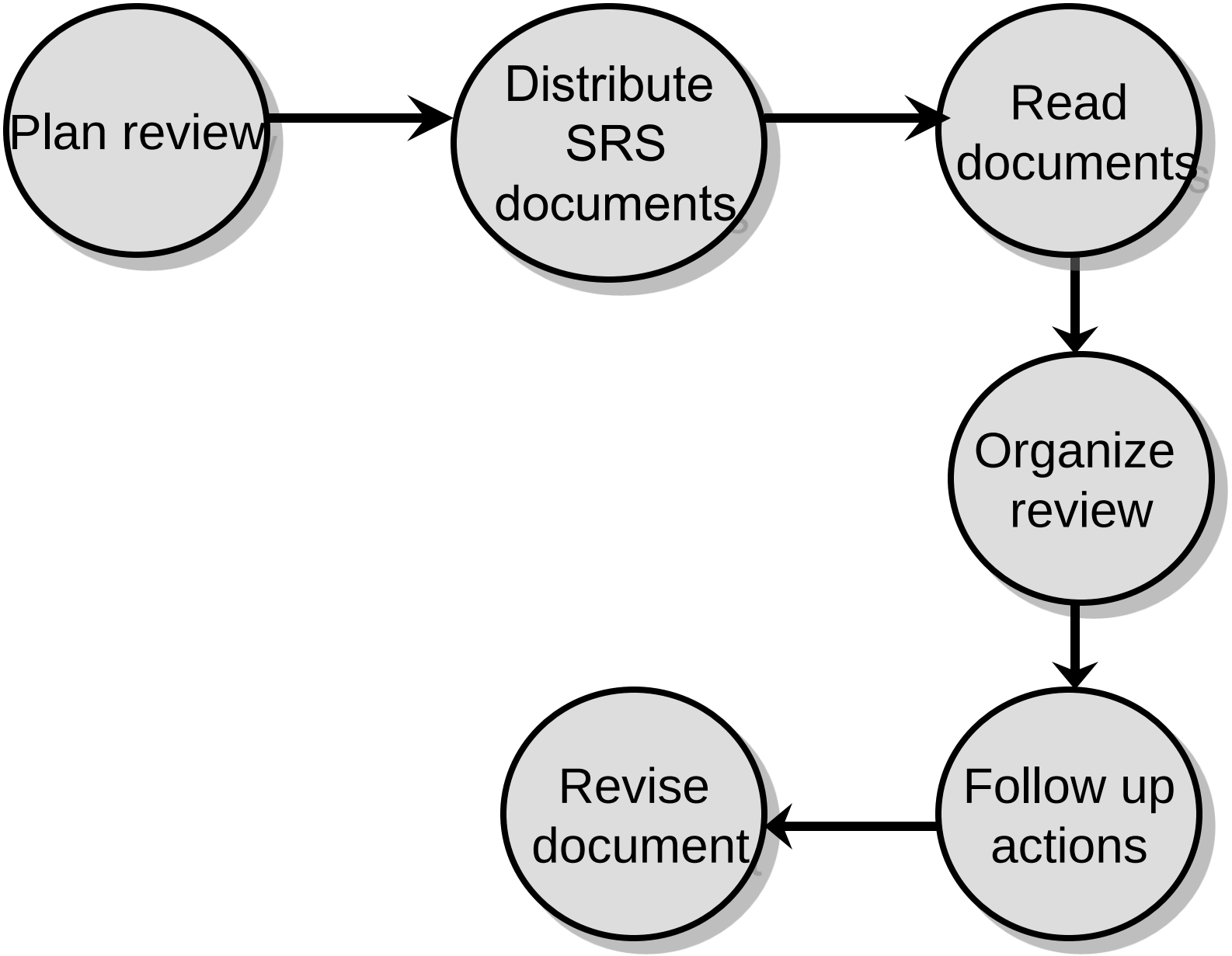
Check the document for:

- ✓ Completeness & consistency
- ✓ Conformance to standards
- ✓ Requirements conflicts
- ✓ Technical errors
- ✓ Ambiguous requirements

Requirements Validation



Requirements Review Process



Requirements Validation

Problem actions

- Requirements clarification
- Missing information
  - find this information from stakeholders
- Requirements conflicts
  - Stakeholders must negotiate to resolve this conflict
- Unrealistic requirements
  - Stakeholders must be consulted
- Security issues
  - Review the system in accordance to security standards

Review Checklists

- ✓ Redundancy
- ✓ Completeness
- ✓ Ambiguity
- ✓ Consistency
- ✓ Organization
- ✓ Conformance
- ✓ Traceability

Prototyping

Validation prototype should be reasonably complete & efficient & should be used as the required system.

Requirements Management

- Process of understanding and controlling changes to system requirements.

ENDURING & VOLATILE REQUIREMENTS

o Enduring requirements: They are core requirements & are related to main activity of the organization.

  Example: issue/return of a book, cataloging etc.

o Volatile requirements: likely to change during software development lifer cycle or after delivery of the product

Requirements Management Planning

- Very critical.
- Important for the success of any project.

Requirements Change Management

- Allocating adequate resources
- Analysis of requirements
- Documenting requirements
- Requirements traceability
- Establishing team communication
- Establishment of baseline

System models

Lecture – 10

Objectives

- To explain why the context of a system should be modelled as part of the RE process
- To describe behavioural modelling, data modelling and object modelling
- To introduce some of the notations used in the Unified Modeling Language (UML)
- To show how CASE workbenches support system modelling

Topics covered

- Context models
- Behavioural models
- Data models
- Object models
- CASE workbenches

System modelling

- System modelling helps the analyst to understand the functionality of the system and models are used to communicate with customers.
- Different models present the system from different perspectives
  - External perspective showing the system's context or environment;
  - Behavioural perspective showing the behaviour of the system;
  - Structural perspective showing the system or data architecture.

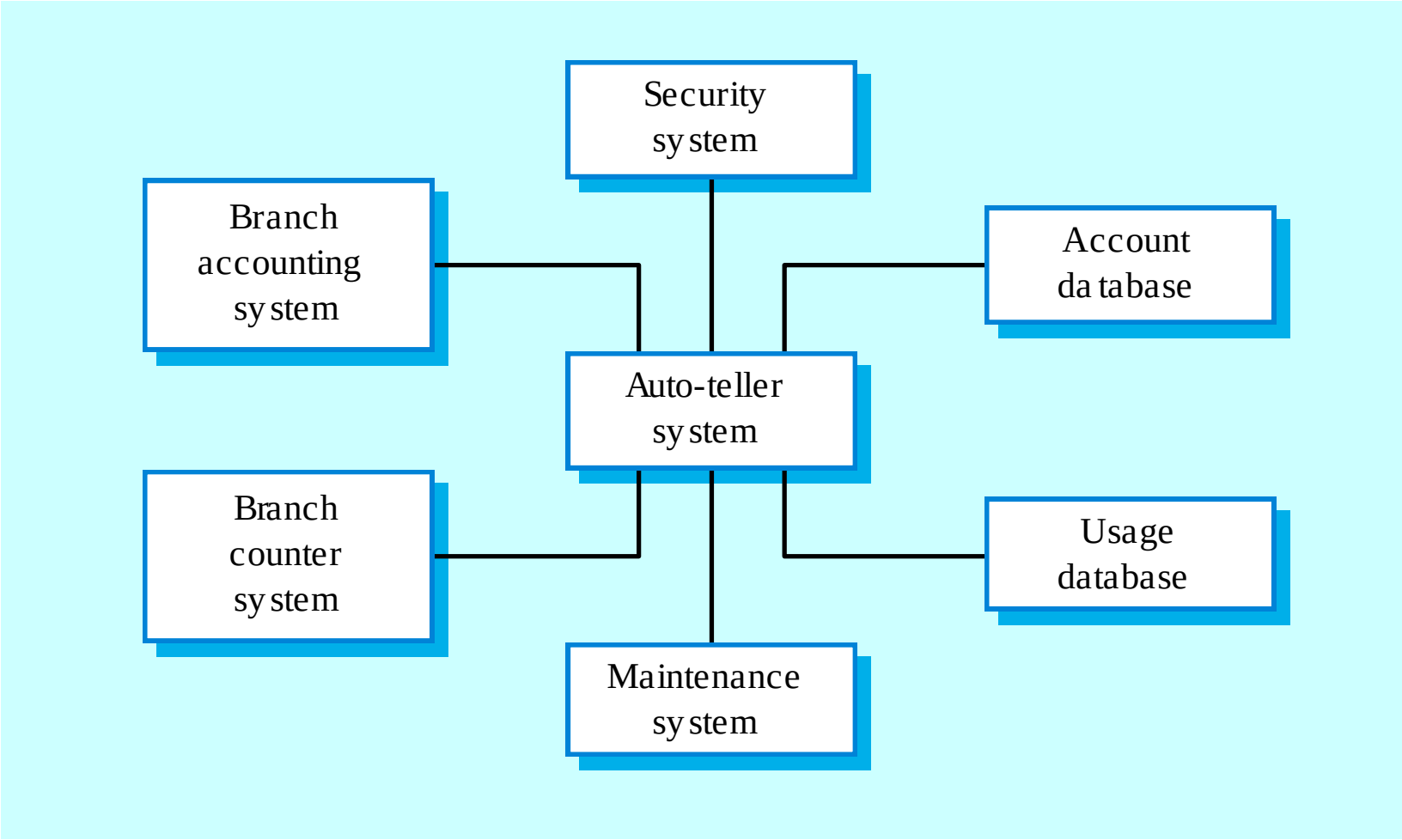
Model types

- Data processing model showing how the data is processed at different stages.
- Composition model showing how entities are composed of other entities.
- Architectural model showing principal sub-systems.
- Classification model showing how entities have common characteristics.
- Stimulus/response model showing the system’s reaction to events.

Context models

- Context models are used to illustrate the operational context of a system - they show what lies outside the system boundaries.
- Social and organisational concerns may affect the decision on where to position system boundaries.
- Architectural models show the system and its relationship with other systems.

The context of an ATM system



Process models

- Process models show the overall process and the processes that are supported by the system.
- Data flow models may be used to show the processes and the flow of information from one process to another.

Equipment procurement process

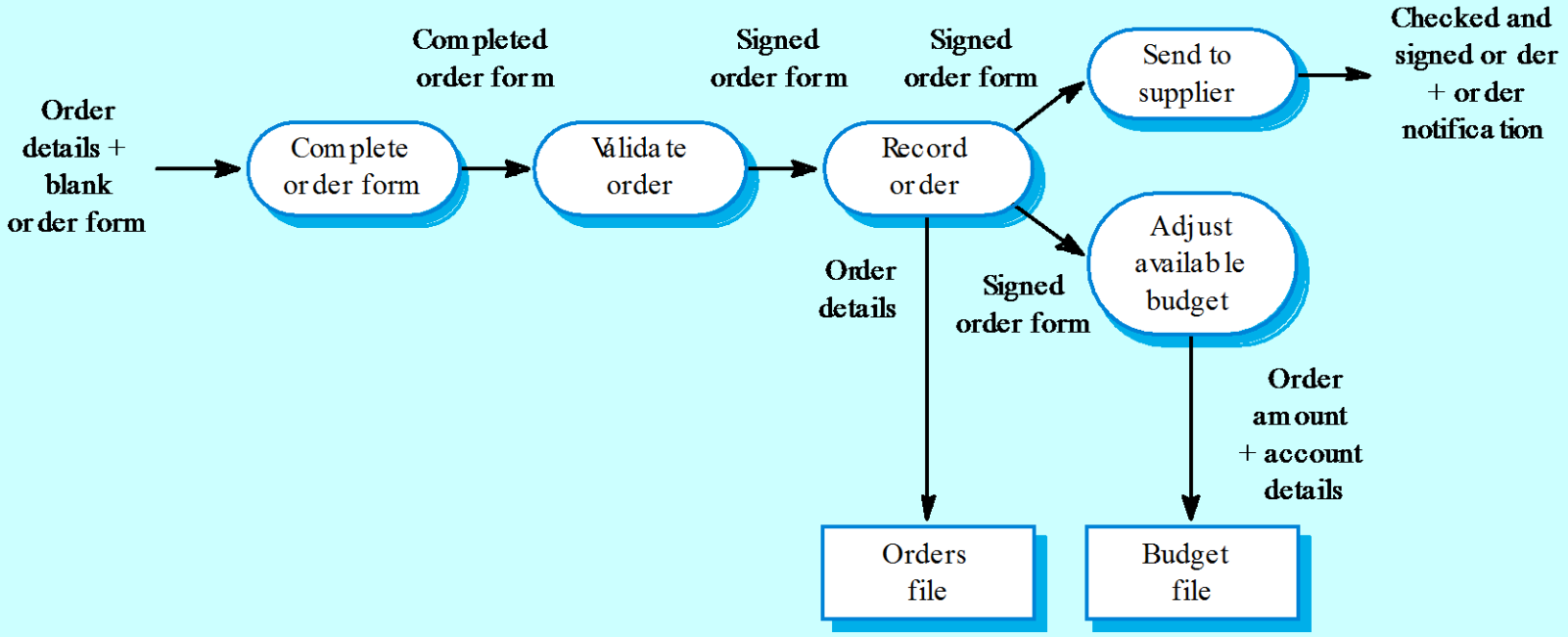
Behavioural models

- Behavioural models are used to describe the overall behaviour of a system.
- Two types of behavioural model are:
  - Data processing models that show how data is processed as it moves through the system;
  - State machine models that show the systems response to events.
- These models show different perspectives so both of them are required to describe the system's behaviour.

Data-processing models

- Data flow diagrams (DFDs) may be used to model the system's data processing.
- These show the processing steps as data flows through a system.
- DFDs are an intrinsic part of many analysis methods.
- Simple and intuitive notation that customers can understand.
- Show end-to-end processing of data.

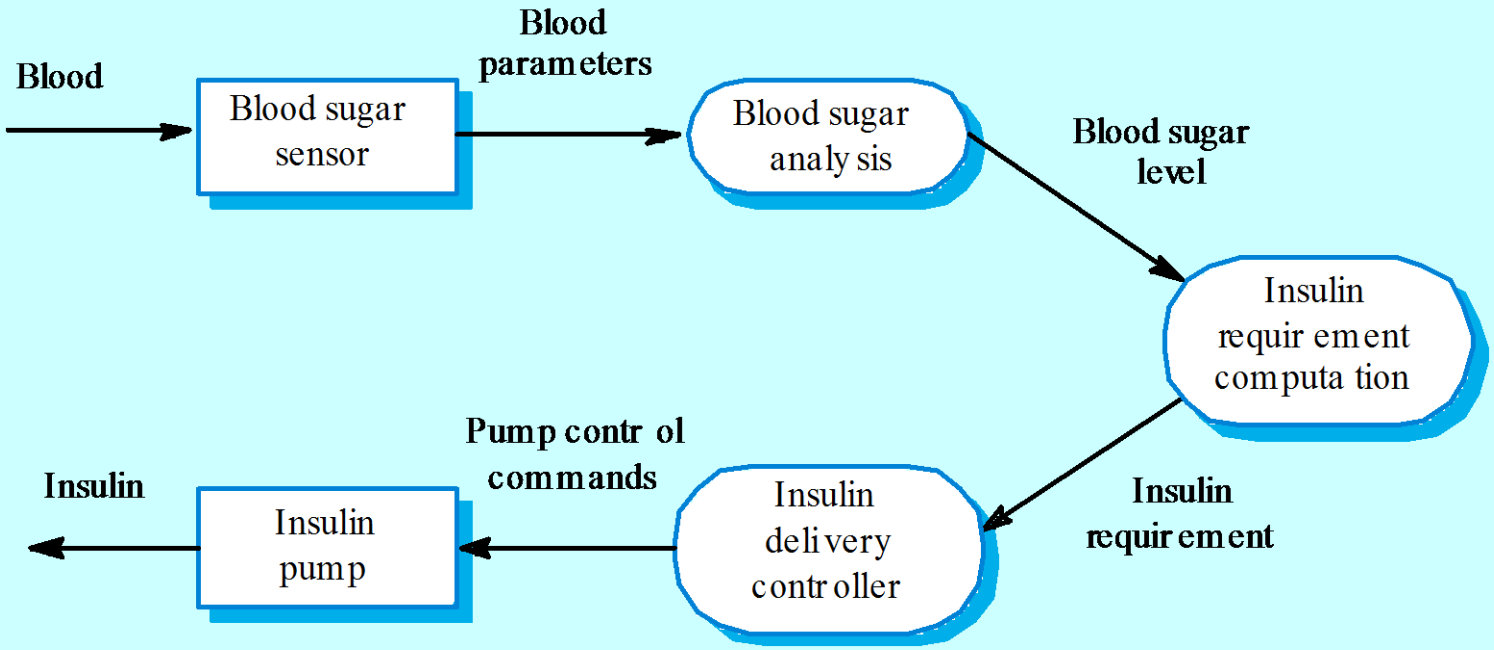
Order processing DFD



Data flow diagrams

- DFDs model the system from a functional perspective.
- Tracking and documenting how the data associated with a process is helpful to develop an overall understanding of the system.
- Data flow diagrams may also be used in showing the data exchange between a system and other systems in its environment.

Insulin pump DFD



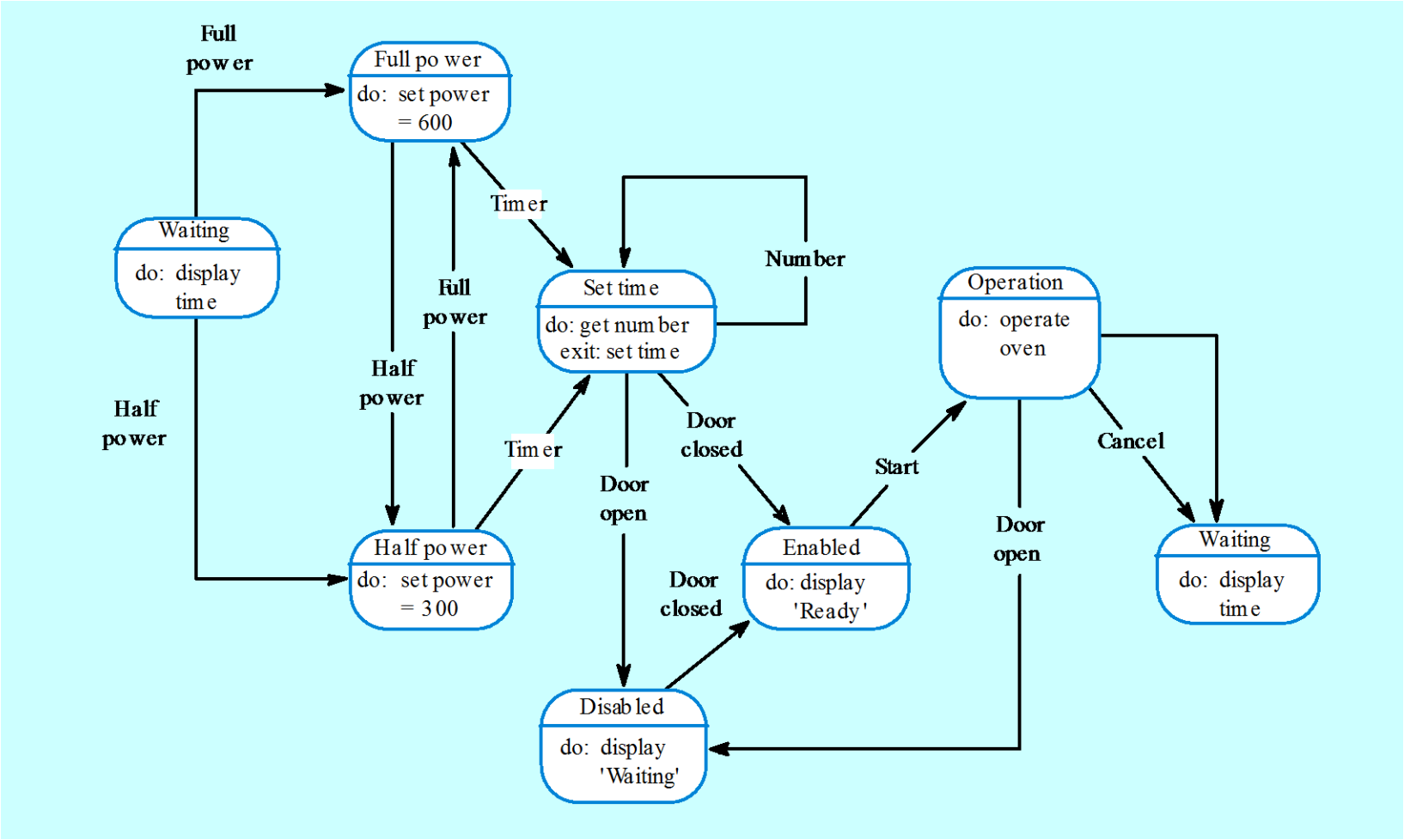
State machine models

- These model the behaviour of the system in response to external and internal events.
- They show the system’s responses to stimuli so are often used for modelling real-time systems.
- State machine models show system states as nodes and events as arcs between these nodes. When an event occurs, the system moves from one state to another.
- Statecharts are an integral part of the UML and are used to represent state machine models.

Statecharts

- Allow the decomposition of a model into sub-models (see following slide).
- A brief description of the actions is included following the ‘do’ in each state.
- Can be complemented by tables describing the states and the stimuli.

Microwave oven model



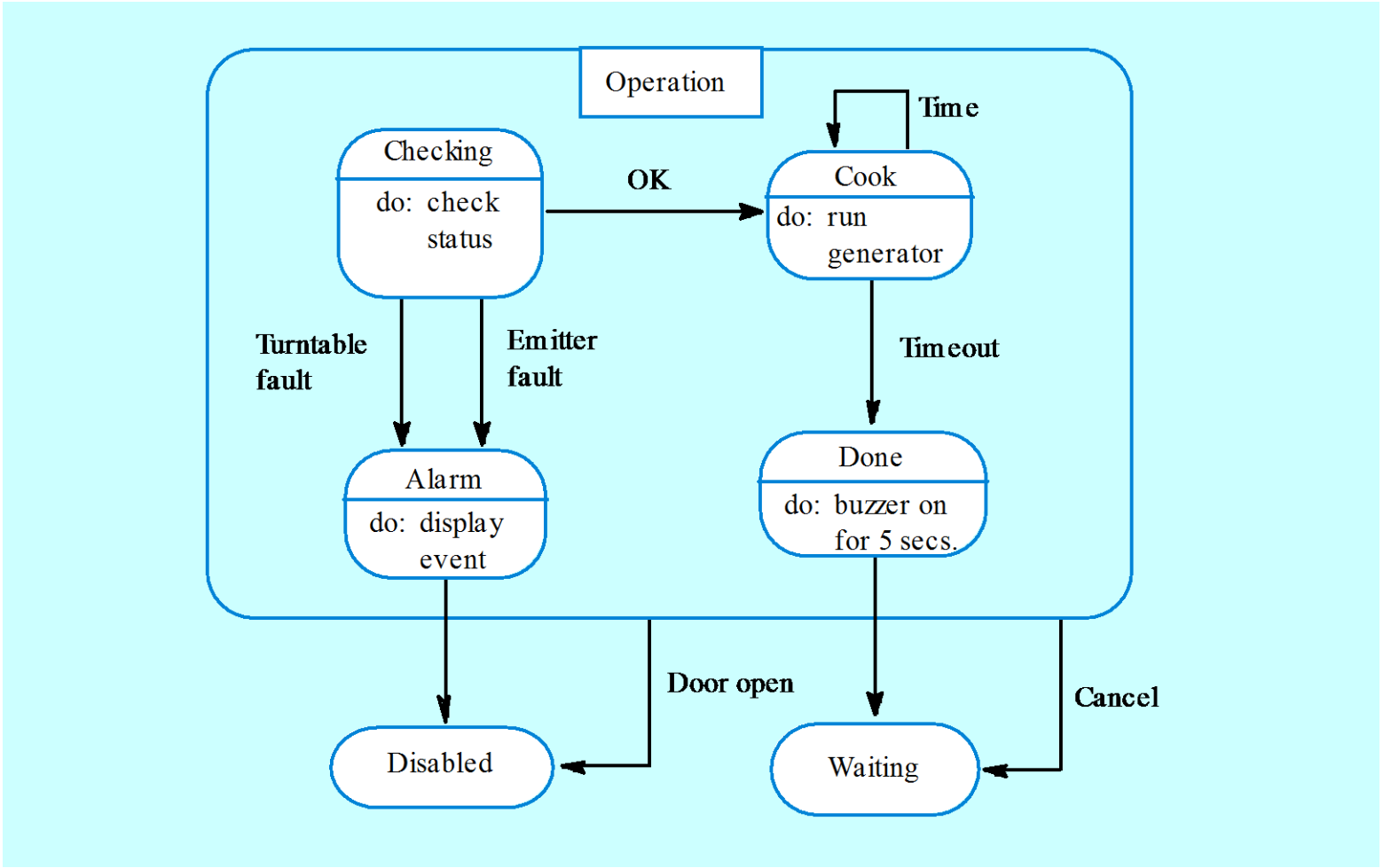
Microwave oven state description

State	Description
Waiting	The oven is waiting for input. The display shows the current time.
Half power	The oven power is set to 300 watts. The display shows ÔHalf powerÕ
Full power	The oven power is set to 600 watts. The display shows ÔFull powerÕ
Set time	The cooking time is set to the userÕs input value. The display shows the cooking time selected and is updated as the time is set.
Disabled	Oven operation is disabled for safety. Interior oven light is on. Display shows ÔNot readyÕ
Enabled	Oven operation is enabled. Interior oven light is off. Display shows ÔReady to cookÕ
Operation	Oven in operation. Interior oven light is on. Display shows the timer countdown. On completion of cooking, the buzzer is sounded for 5 seconds. Oven light is on. Display shows ÔCooking completeÕ while buzzer is sounding.

Microwave oven stimuli

Stimulus	Description
Half power	The user has pressed the half power button
Full power	The user has pressed the full power button
Timer	The user has pressed one of the timer buttons
Number	The user has pressed a numeric key
Door open	The oven door switch is not closed
Door closed	The oven door switch is closed
Start	The user has pressed the start button
Cancel	The user has pressed the cancel button

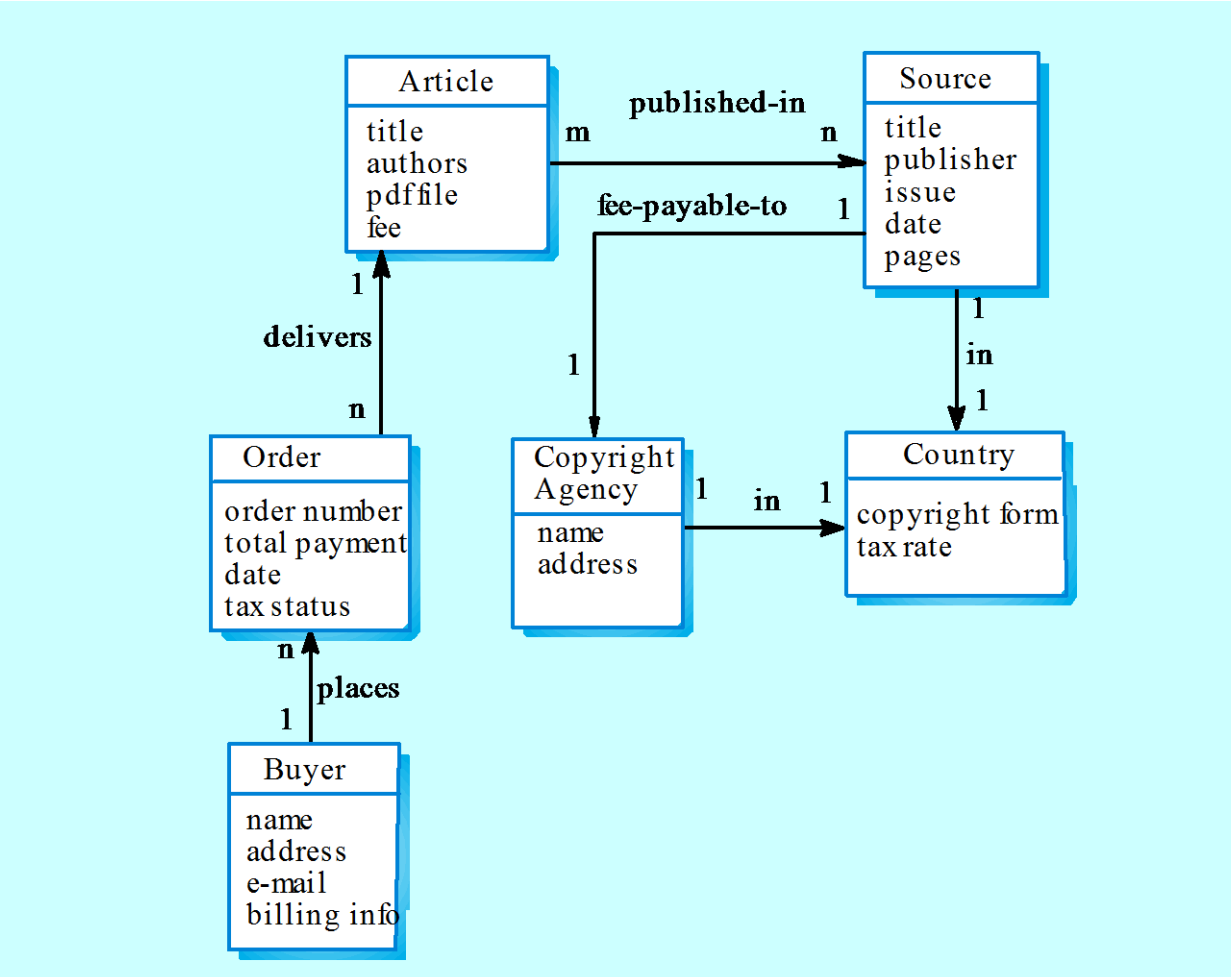
Microwave oven operation



Semantic data models

- Used to describe the logical structure of data processed by the system.
- An entity-relation-attribute model sets out the entities in the system, the relationships between these entities and the entity attributes
- Widely used in database design. Can readily be implemented using relational databases.
- No specific notation provided in the UML but objects and associations can be used.

Library semantic model



Data dictionaries

- Data dictionaries are lists of all of the names used in the system models. Descriptions of the entities, relationships and attributes are also included.
- Advantages
  - Support name management and avoid duplication;
  - Store of organisational knowledge linking analysis, design and implementation;
- Many CASE workbenches support data dictionaries.

Data dictionary entries

Name	Description	Type	Date
Article	Details of the published article that may be ordered by people using LIBSYS.	Entity	30.12.2002
authors	The names of the authors of the article who may be due a share of the fee.	Attribute	30.12.2002
Buyer	The person or organisation that orders a copy of the article.	Entity	30.12.2002
fee-payable-to	A 1:1 relationship between Article and the Copyright Agency who should be paid the copyright fee.	Relation	29.12.2002
Address (Buyer)	The address of the buyer. This is used to any paper billing information that is required.	Attribute	31.12.2002

Lecture – 12

Object models

- Object models describe the system in terms of object classes and their associations.
- An object class is an abstraction over a set of objects with common attributes and the services (operations) provided by each object.
- Various object models may be produced
  - Inheritance models;
  - Aggregation models;
  - Interaction models.

Object models

- Natural ways of reflecting the real-world entities manipulated by the system
- More abstract entities are more difficult to model using this approach
- Object class identification is recognised as a difficult process requiring a deep understanding of the application domain
- Object classes reflecting domain entities are reusable across systems

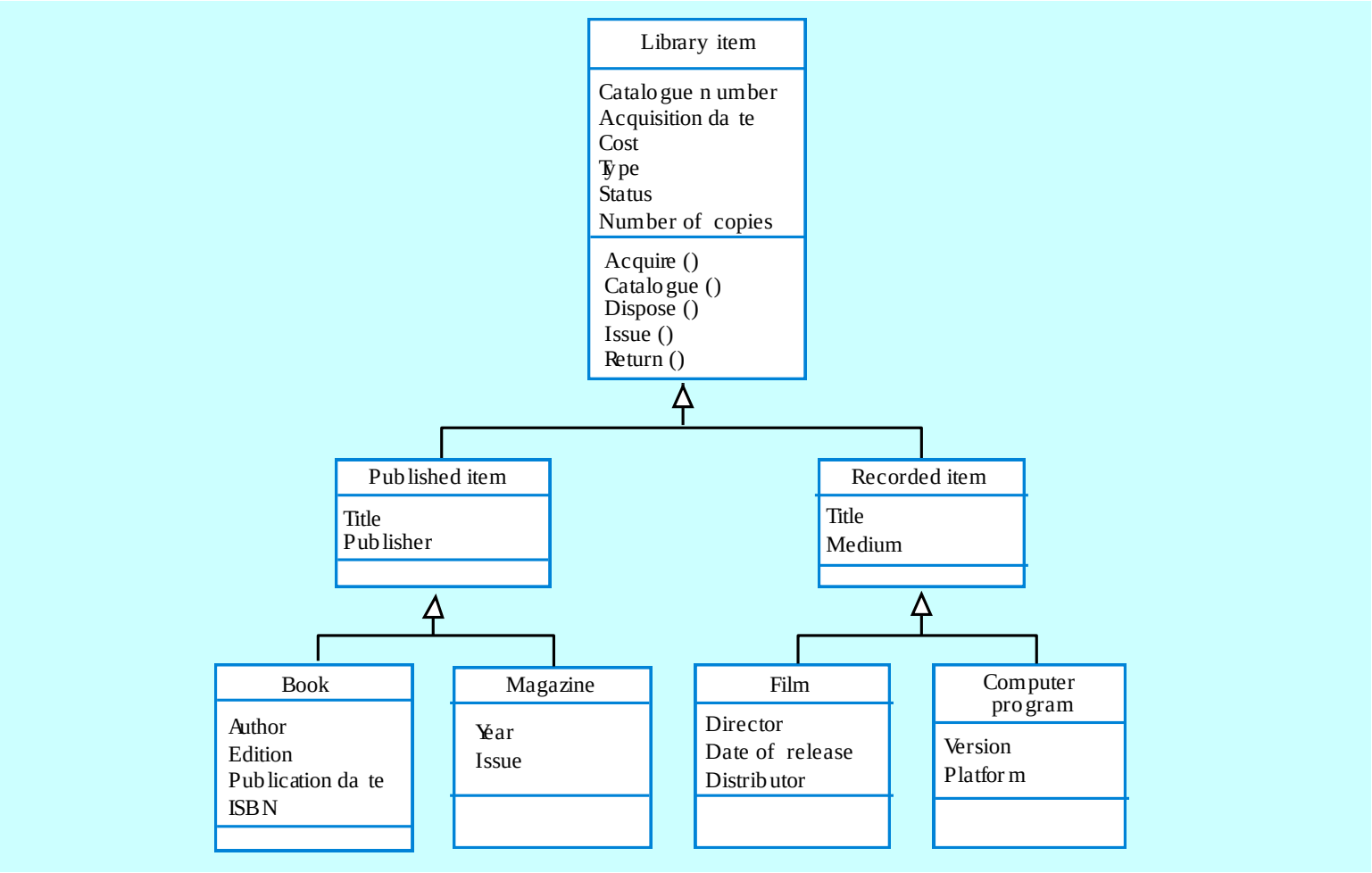
Inheritance models

- Organise the domain object classes into a hierarchy.
- Classes at the top of the hierarchy reflect the common features of all classes.
- Object classes inherit their attributes and services from one or more super-classes. these may then be specialised as necessary.
- Class hierarchy design can be a difficult process if duplication in different branches is to be avoided.

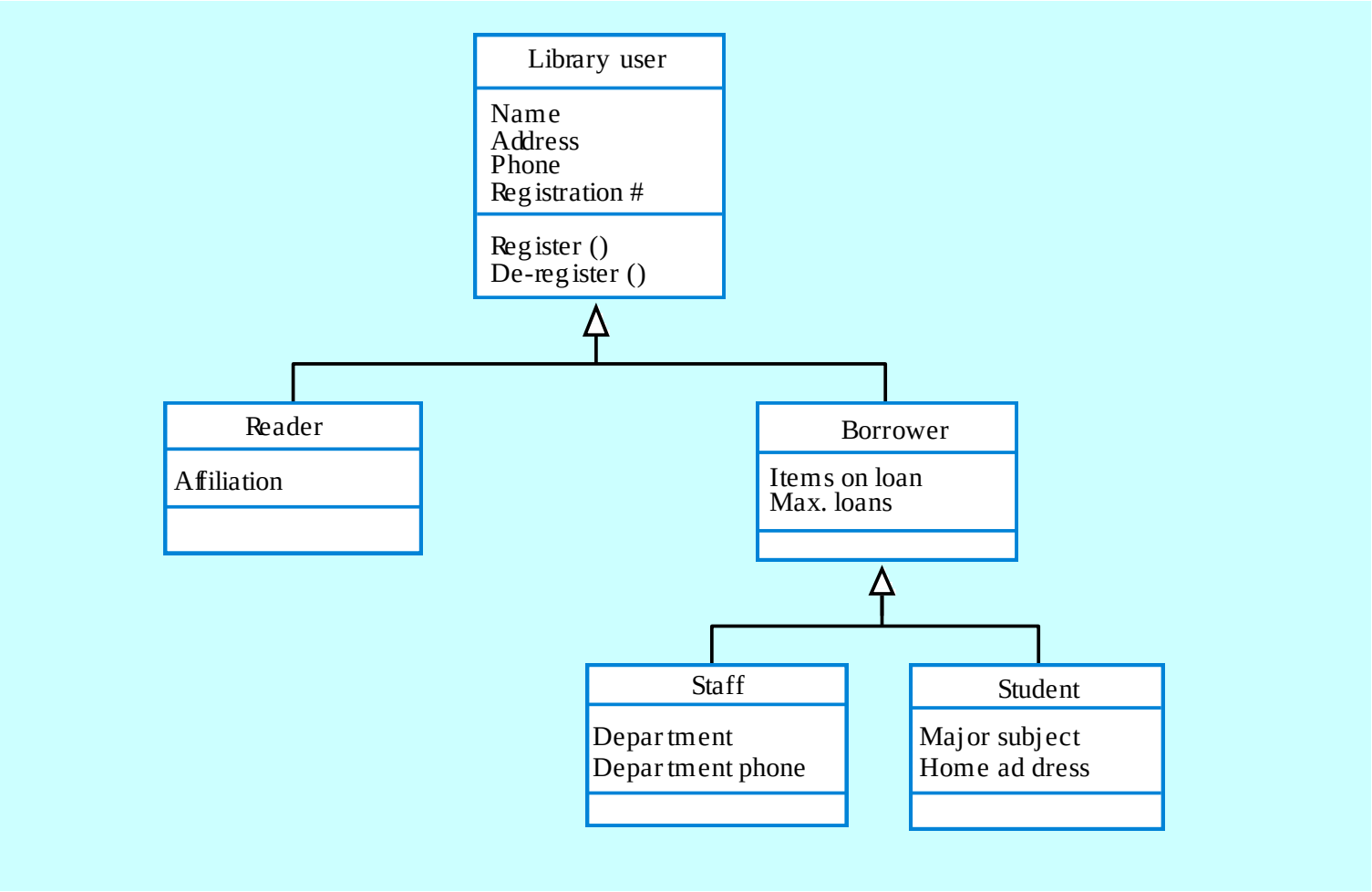
Object models and the UML

- The UML is a standard representation devised by the developers of widely used object-oriented analysis and design methods.
- It has become an effective standard for object-oriented modelling.
- Notation
  - Object classes are rectangles with the name at the top, attributes in the middle section and operations in the bottom section;
  - Relationships between object classes (known as associations) are shown as lines linking objects;
  - Inheritance is referred to as generalisation and is shown ‘upwards’ rather than ‘downwards’ in a hierarchy.

Library class hierarchy



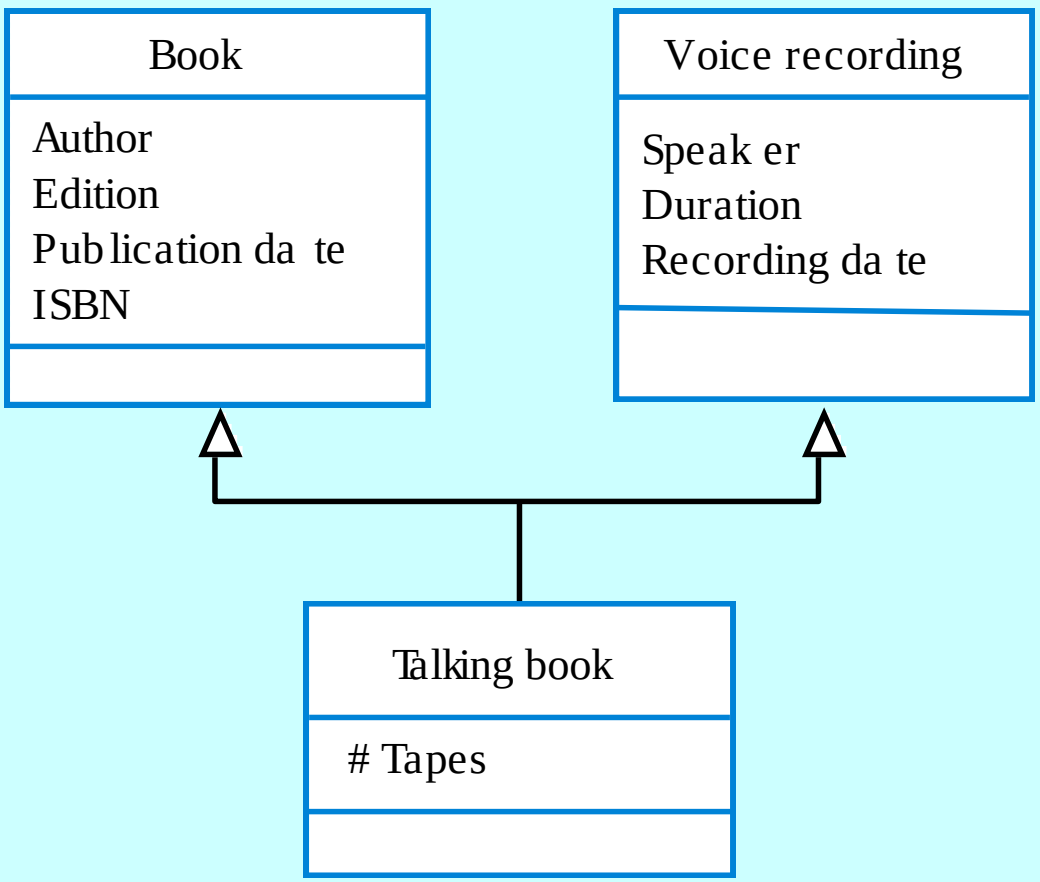
User class hierarchy



Multiple inheritance

- Rather than inheriting the attributes and services from a single parent class, a system which supports multiple inheritance allows object classes to inherit from several super-classes.
- This can lead to semantic conflicts where attributes/services with the same name in different super-classes have different semantics.
- Multiple inheritance makes class hierarchy reorganisation more complex.

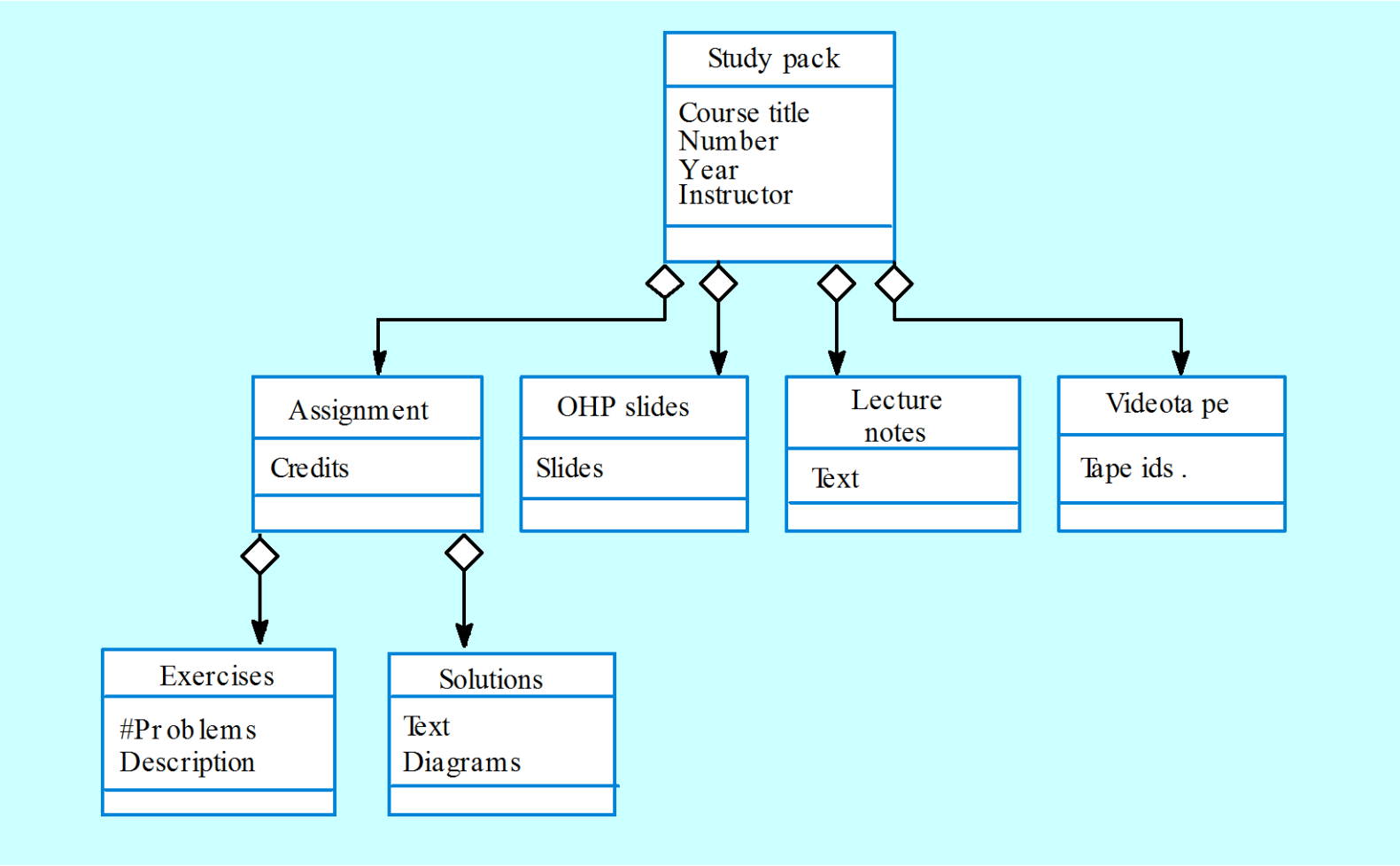
Multiple inheritance



Object aggregation

- An aggregation model shows how classes that are collections are composed of other classes.
- Aggregation models are similar to the part-of relationship in semantic data models.

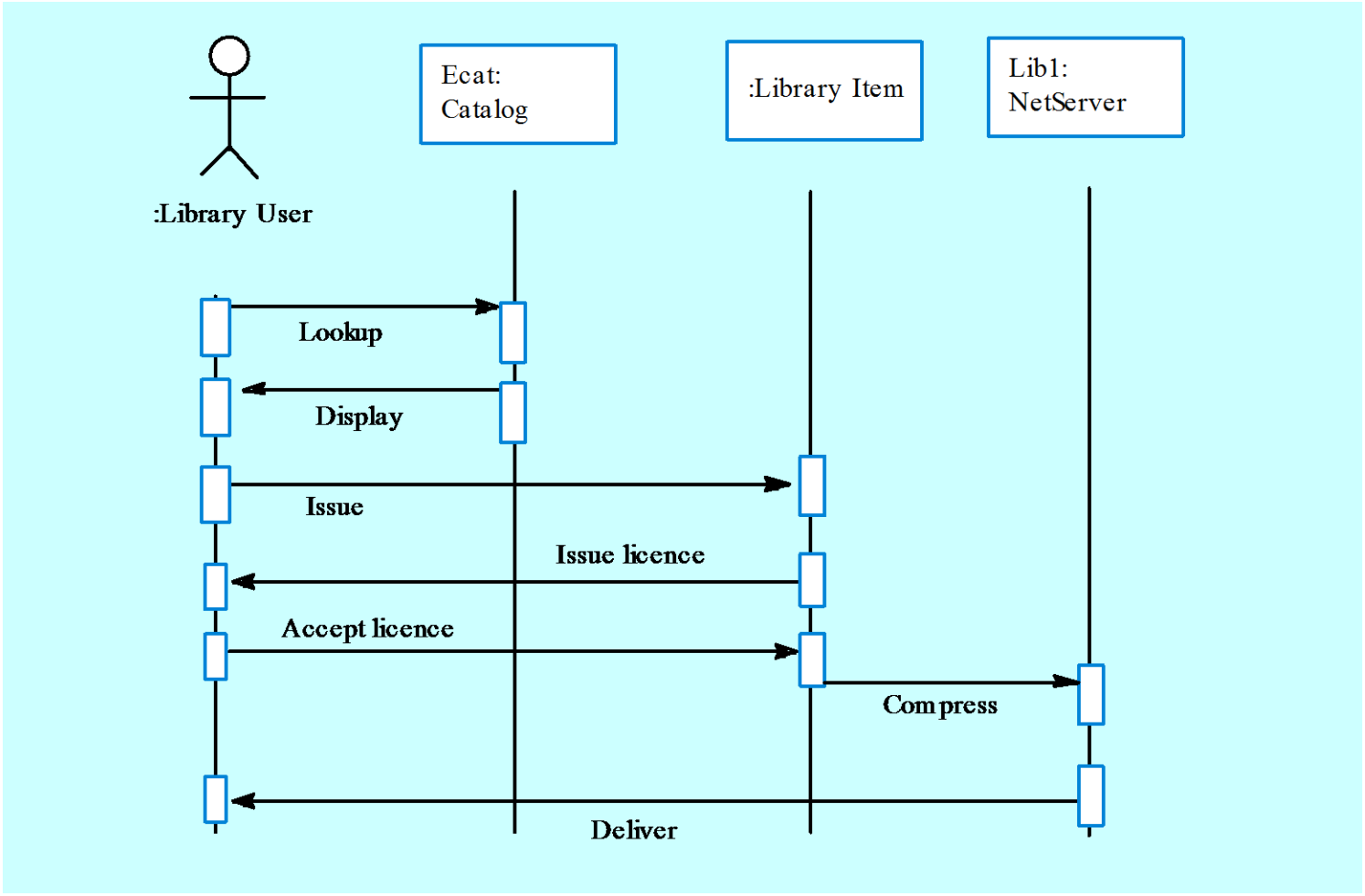
Object aggregation



Object behaviour modelling

- A behavioural model shows the interactions between objects to produce some particular system behaviour that is specified as a use-case.
- Sequence diagrams (or collaboration diagrams) in the UML are used to model interaction between objects.

Issue of electronic items



Structured methods

- Structured methods incorporate system modelling as an inherent part of the method.
- Methods define a set of models, a process for deriving these models and rules and guidelines that should apply to the models.
- CASE tools support system modelling as part of a structured method.

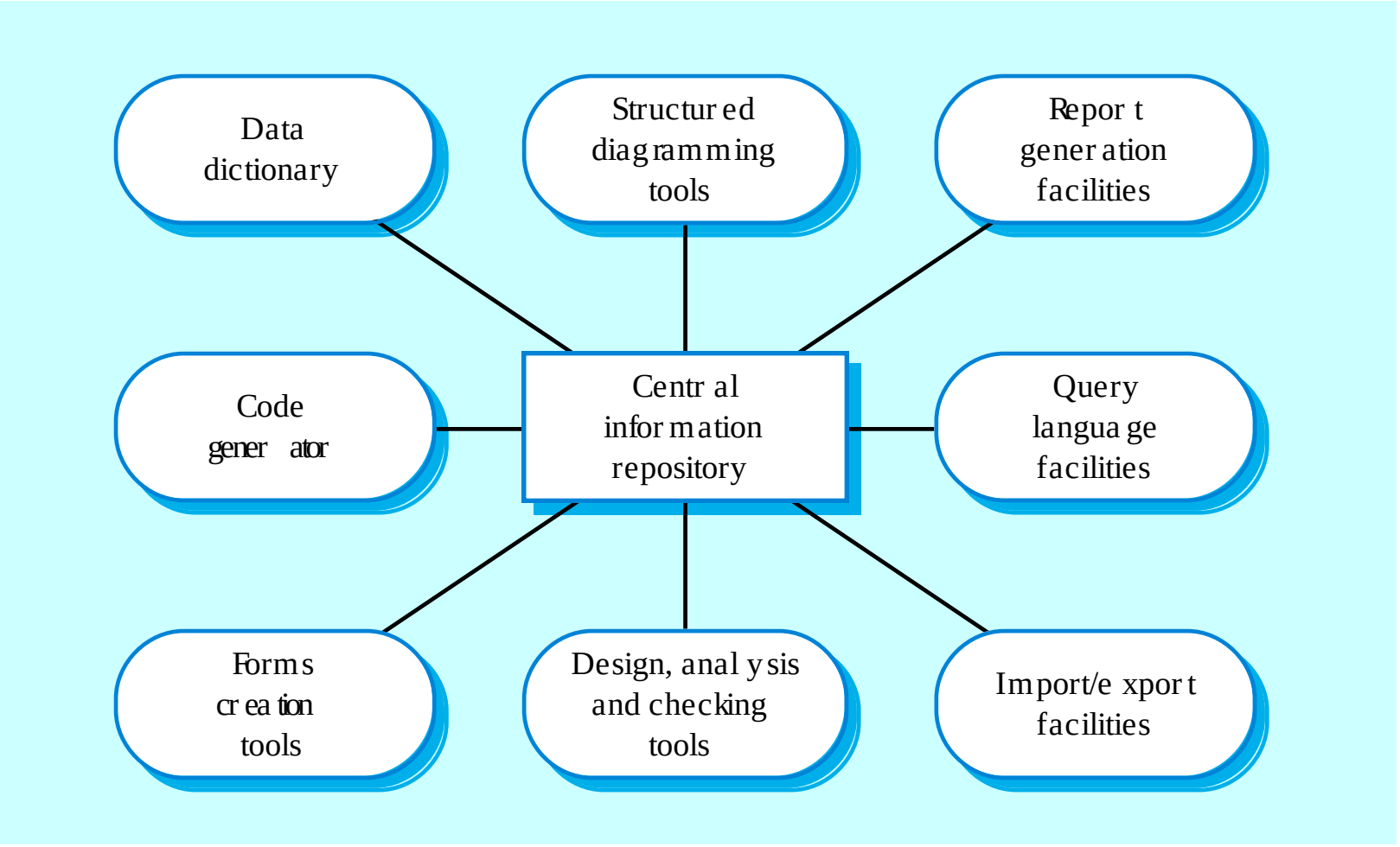
Method weaknesses

- They do not model non-functional system requirements.
- They do not usually include information about whether a method is appropriate for a given problem.
- The may produce too much documentation.
- The system models are sometimes too detailed and difficult for users to understand.

CASE workbenches

- A coherent set of tools that is designed to support related software process activities such as analysis, design or testing.
- Analysis and design workbenches support system modelling during both requirements engineering and system design.
- These workbenches may support a specific design method or may provide support for a creating several different types of system model.

An analysis and design workbench



Analysis workbench components

- Diagram editors
- Model analysis and checking tools
- Repository and associated query language
- Data dictionary
- Report definition and generation tools
- Forms definition tools
- Import/export translators
- Code generation tools

Key points

- A model is an abstract system view. Complementary types of model provide different system information.
- Context models show the position of a system in its environment with other systems and processes.
- Data flow models may be used to model the data processing in a system.
- State machine models model the system’s behaviour in response to internal or external events

Key points

- Semantic data models describe the logical structure of data which is imported to or exported by the systems.
- Object models describe logical system entities, their classification and aggregation.
- Sequence models show the interactions between actors and the system objects that they use.
- Structured methods provide a framework for developing system models.

Critical Systems Specification

Lecture – 13

Objectives

- To explain how dependability requirements may be identified by analysing the risks faced by critical systems
- To explain how safety requirements are generated from the system risk analysis
- To explain the derivation of security requirements
- To describe metrics used for reliability specification

Topics covered

- Risk-driven specification
- Safety specification
- Security specification
- Software reliability specification

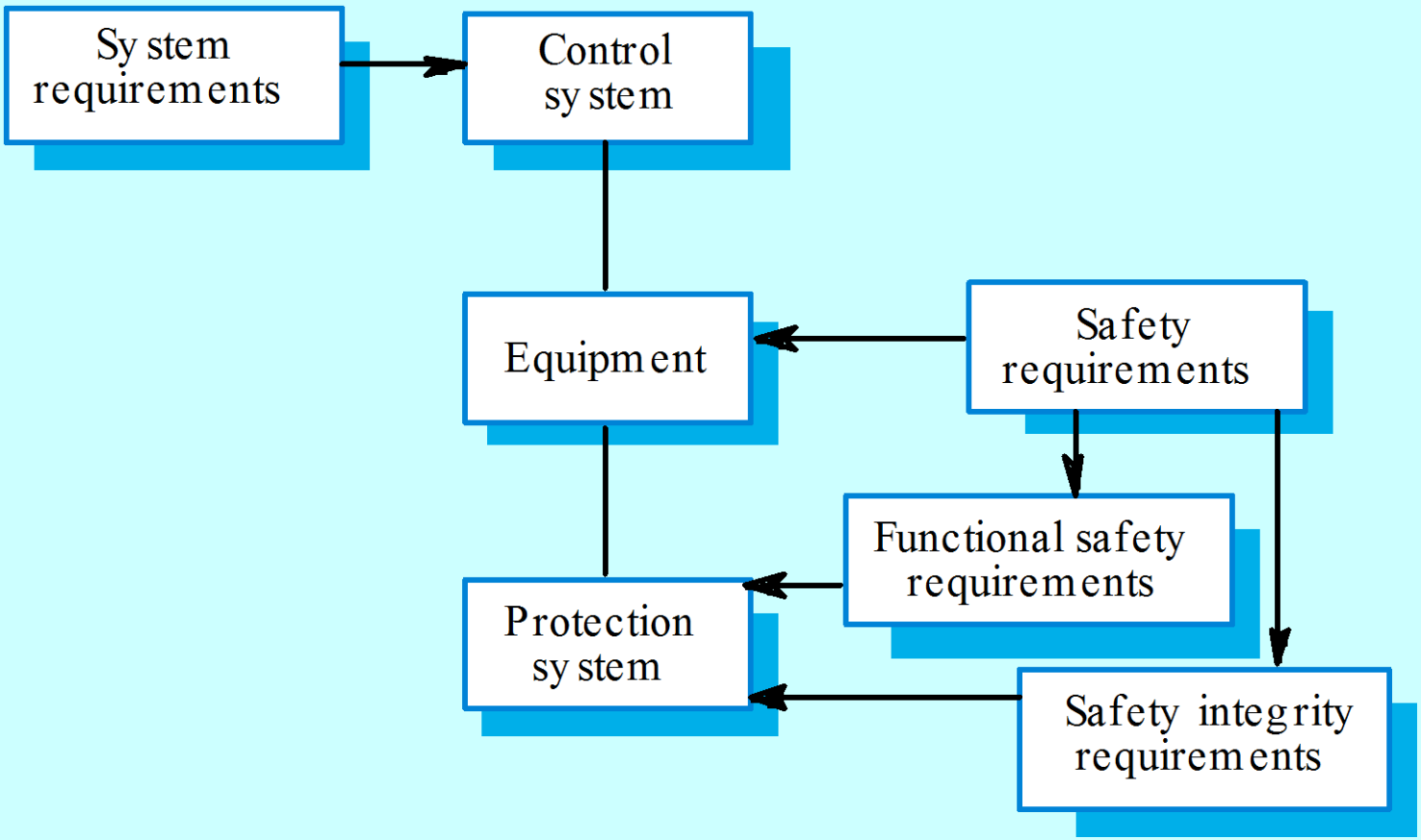
Dependability requirements

- Functional requirements to define error checking and recovery facilities and protection against system failures.
- Non-functional requirements defining the required reliability and availability of the system.
- Excluding requirements that define states and conditions that must not arise.

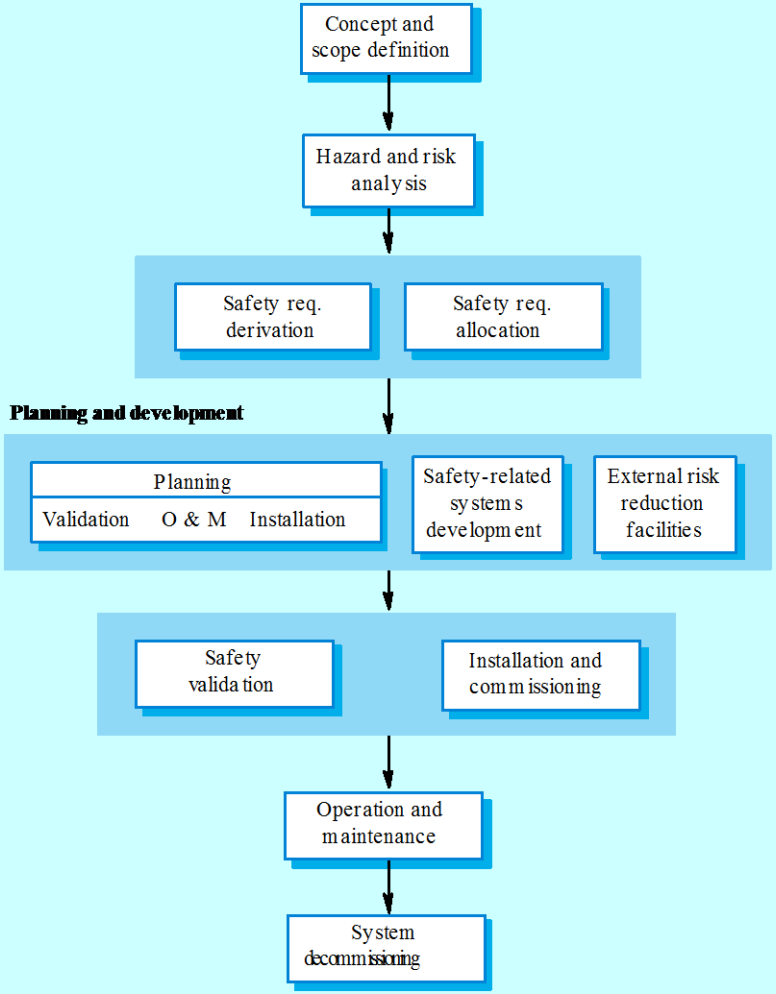
IEC 61508

- An international standard for safety management that was specifically designed for protection systems - it is not applicable to all safety-critical systems.
- Incorporates a model of the safety life cycle and covers all aspects of safety management from scope definition to system decommissioning.

Control system safety requirements



The safety life-cycle



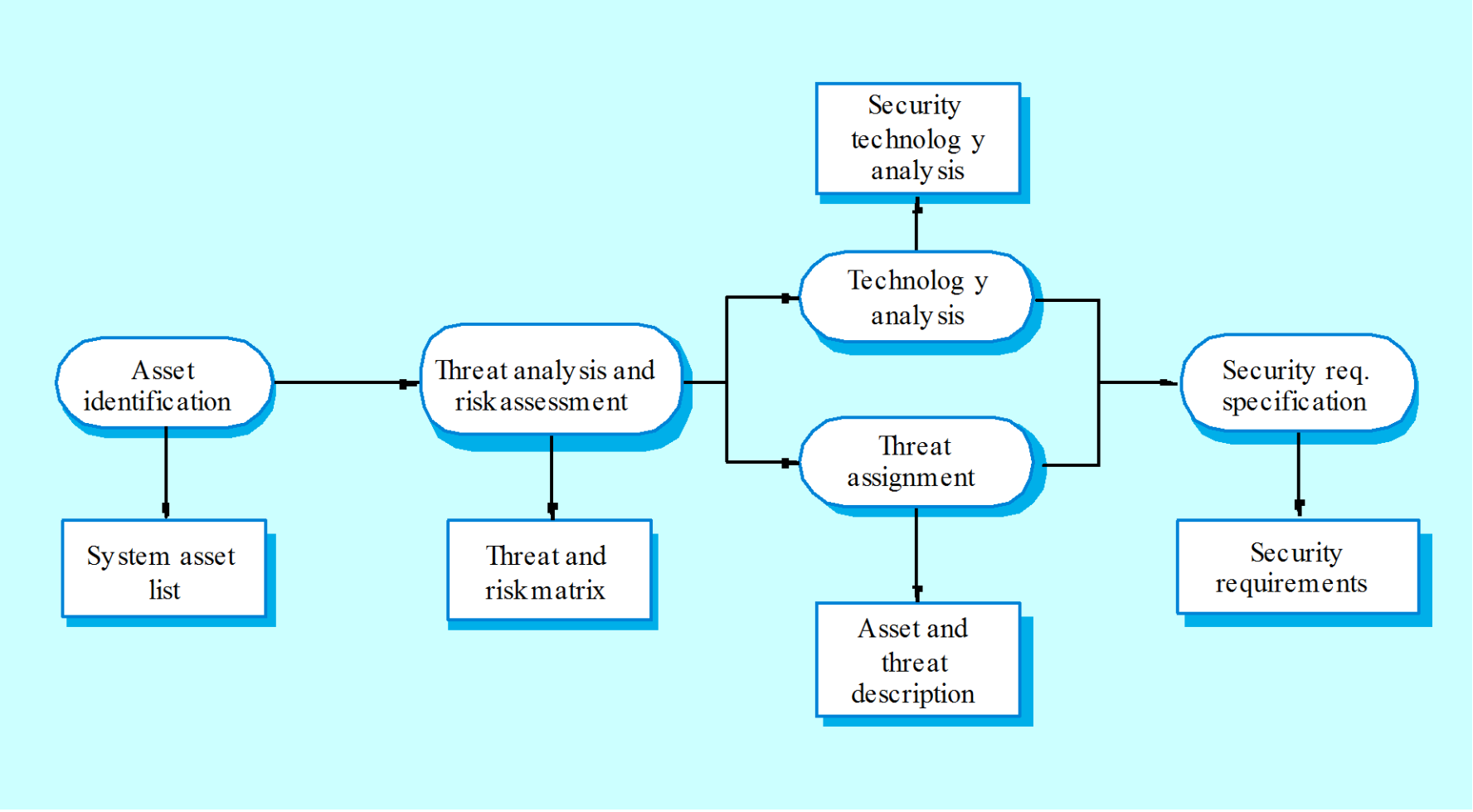
Safety requirements

- Functional safety requirements
  - These define the safety functions of the protection system i.e. the define how the system should provide protection.
- Safety integrity requirements
  - These define the reliability and availability of the protection system. They are based on expected usage and are classified using a safety integrity level from 1 to 4.

Security specification

- Has some similarities to safety specification
  - Not possible to specify security requirements quantitatively;
  - The requirements are often ‘shall not’ rather than ‘shall’ requirements.
- Differences
  - No well-defined notion of a security life cycle for security management; No standards;
  - Generic threats rather than system specific hazards;
  - Mature security technology (encryption, etc.). However, there are problems in transferring this into general use;
  - The dominance of a single supplier (Microsoft) means that huge numbers of systems may be affected by security failure.

The security specification process



Stages in security specification

- Asset identification and evaluation
  - The assets (data and programs) and their required degree of protection are identified. The degree of required protection depends on the asset value so that a password file (say) is more valuable than a set of public web pages.
- Threat analysis and risk assessment
  - Possible security threats are identified and the risks associated with each of these threats is estimated.
- Threat assignment
  - Identified threats are related to the assets so that, for each identified asset, there is a list of associated threats.

Stages in security specification

- Technology analysis
  - Available security technologies and their applicability against the identified threats are assessed.
- Security requirements specification
  - The security requirements are specified. Where appropriate, these will explicitly identified the security technologies that may be used to protect against different threats to the system.

Types of security requirement

- Identification requirements.
- Authentication requirements.
- Authorisation requirements.
- Immunity requirements.
- Integrity requirements.
- Intrusion detection requirements.
- Non-repudiation requirements.
- Privacy requirements.
- Security auditing requirements.
- System maintenance security requirements.

LIBSYS security requirements

SEC1: All system users shall be identified using their library card number and personal password.

SEC2: Users privileges shall be assigned according to the class of user (student, staff, library staff).

SEC3: Before execution of any command, LIBSYS shall check that the user has sufficient privileges to access and execute that command.

SEC4: When a user orders a document, the order request shall be logged. The log data maintained shall include the time of order, the user's identification and the articles ordered.

SEC5: All system data shall be backed up once per day and backups stored off-site in a secure storage area.

SEC6: Users shall not be permitted to have more than 1 simultaneous login to LIBSYS.

System reliability specification

- Hardware reliability
  - What is the probability of a hardware component failing and how long does it take to repair that component?
- Software reliability
  - How likely is it that a software component will produce an incorrect output. Software failures are different from hardware failures in that software does not wear out. It can continue in operation even after an incorrect result has been produced.
- Operator reliability
  - How likely is it that the operator of a system will make an error?

Functional reliability requirements

- A predefined range for all values that are input by the operator shall be defined and the system shall check that all operator inputs fall within this predefined range.
- The system shall check all disks for bad blocks when it is initialised.
- The system must use N-version programming to implement the braking control system.
- The system must be implemented in a safe subset of Ada and checked using static analysis.

Non-functional reliability specification

- The required level of system reliability required should be expressed quantitatively.
- Reliability is a dynamic system attribute- reliability specifications related to the source code are meaningless.
  - No more than N faults/1000 lines;
  - This is only useful for a post-delivery process analysis where you are trying to assess how good your development techniques are.
- An appropriate reliability metric should be chosen to specify the overall system reliability.

Reliability metrics

- Reliability metrics are units of measurement of system reliability.
- System reliability is measured by counting the number of operational failures and, where appropriate, relating these to the demands made on the system and the time that the system has been operational.
- A long-term measurement programme is required to assess the reliability of critical systems.

Reliability metrics

Metric	Explanation
POFOD Probability of failure on demand	The likelihood that the system will fail when a service request is made. A POFOD of 0.001 means that 1 out of a thousand service requests may result in failure.
ROCOF Rate of failure occurrence	The frequency of occurrence with which unexpected behaviour is likely to occur. A ROCOF of 2/100 means that 2 failures are likely to occur in each 100 operational time units. This metric is sometimes called the failure intensity.
MTTF Mean time to failure	The average time between observed system failures. An MTTF of 500 means that 1 failure can be expected every 500 time units.
AVAIL Availability	The probability that the system is available for use at a given time. Availability of 0.998 means that in every 1000 time units, the system is likely to be available for 998 of these.

Probability of failure on demand

- This is the probability that the system will fail when a service request is made. Useful when demands for service are intermittent and relatively infrequent.
- Appropriate for protection systems where services are demanded occasionally and where there are serious consequence if the service is not delivered.
- Relevant for many safety-critical systems with exception management components
  - Emergency shutdown system in a chemical plant.

Rate of fault occurrence (ROCOF)

- Reflects the rate of occurrence of failure in the system.
- ROCOF of 0.002 means 2 failures are likely in each 1000 operational time units e.g. 2 failures per 1000 hours of operation.
- Relevant for operating systems, transaction processing systems where the system has to process a large number of similar requests that are relatively frequent
  - Credit card processing system, airline booking system.

Mean time to failure

- Measure of the time between observed failures of the system. Is the reciprocal of ROCOF for stable systems.
- MTTF of 500 means that the mean time between failures is 500 time units.
- Relevant for systems with long transactions i.e. where system processing takes a long time. MTTF should be longer than transaction length
  - Computer-aided design systems where a designer will work on a design for several hours, word processor systems.

Availability

- Measure of the fraction of the time that the system is available for use.
- Takes repair and restart time into account
- Availability of 0.998 means software is available for 998 out of 1000 time units.
- Relevant for non-stop, continuously running systems
  - telephone switching systems, railway signalling systems.

Non-functional requirements spec.

- Reliability measurements do NOT take the consequences of failure into account.
- Transient faults may have no real consequences but other faults may cause data loss or corruption and loss of system service.
- May be necessary to identify different failure classes and use different metrics for each of these. The reliability specification must be structured.

Failure consequences

- When specifying reliability, it is not just the number of system failures that matter but the consequences of these failures.
- Failures that have serious consequences are clearly more damaging than those where repair and recovery is straightforward.
- In some cases, therefore, different reliability specifications for different types of failure may be defined.

Failure classification

Failure class	Description
Transient	Occurs only with certain inputs
Permanent	Occurs with all inputs
Recoverable	System can recover without operator intervention
Unrecoverable	Operator intervention needed to recover from failure
Non-corrupting	Failure does not corrupt system state or data
Corrupting	Failure corrupts system state or data

Steps to a reliability specification

- For each sub-system, analyse the consequences of possible system failures.
- From the system failure analysis, partition failures into appropriate classes.
- For each failure class identified, set out the reliability using an appropriate metric. Different metrics may be used for different reliability requirements.
- Identify functional reliability requirements to reduce the chances of critical failures.

Bank auto-teller system

- Each machine in a network is used 300 times a day
- Bank has 1000 machines
- Lifetime of software release is 2 years
- Each machine handles about 200, 000 transactions
- About 300, 000 database transactions in total per day

Reliability specification for an ATM

Failure class	Example	Reliability metric
Permanent, non-corrupting.	The system fails to operate with any card that is input. Software must be restarted to correct failure.	ROCOF 1 occurrence/1000 days
Transient, non-corrupting	The magnetic stripe data cannot be read on an undamaged card that is input.	ROCOF 1 in 1000 transactions
Transient, corrupting	A pattern of transactions across the network causes database corruption.	Unquantifiable! Should never happen in the lifetime of the system

Specification validation

- It is impossible to empirically validate very high reliability specifications.
- No database corruptions means POFOD of less than 1 in 200 million.
- If a transaction takes 1 second, then simulating one day's transactions takes 3.5 days.
- It would take longer than the system's lifetime to test it for reliability.

Distributed Systems Architectures

Lecture – 14

Objectives

- To explain the advantages and disadvantages of different distributed systems architectures
- To discuss client-server and distributed object architectures
- To describe object request brokers and the principles underlying the CORBA standards
- To introduce peer-to-peer and service-oriented architectures as new models of distributed computing.

Topics covered

- Multiprocessor architectures
- Client-server architectures
- Distributed object architectures
- Inter-organisational computing

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 therefore very important for enterprise computing systems.

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
  - Sharing of hardware and software resources.
- Openness
  - Use of equipment and software from different vendors.
- Concurrency
  - Concurrent processing to enhance performance.
- Scalability
  - Increased throughput by adding new resources.
- Fault tolerance
  - The ability to continue in operation after a fault has occurred.

Distributed system disadvantages

- Complexity
  - Typically, distributed systems are more complex than centralised systems.
- Security
  - More susceptible to external attack.
- Manageability
  - More effort required for system management.
- Unpredictability
  - Unpredictable responses depending on the system organisation and network load.

Distributed systems architectures

- 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.

Assignment

- Distributed System architecture

Research

- ***Distributed Systems* - IEEE Computer Society**

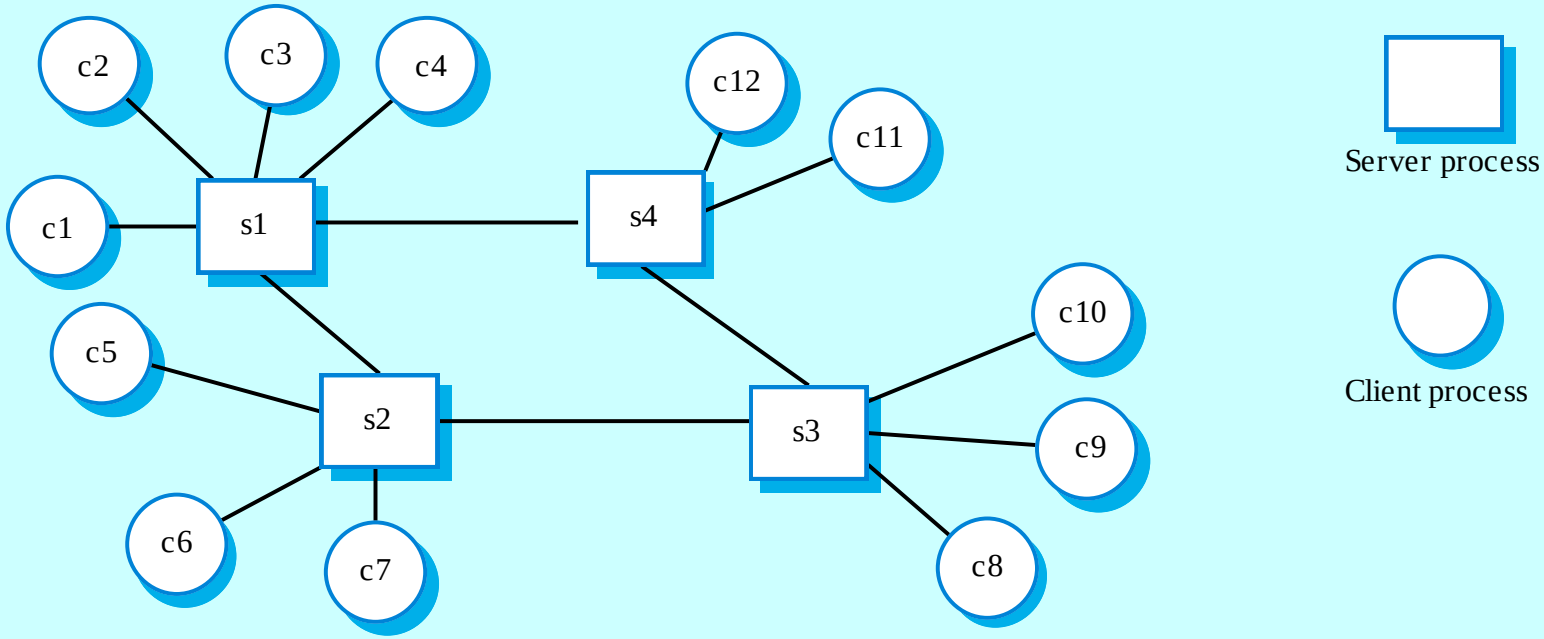
Distributed Systems Architectures

Lecture – 15

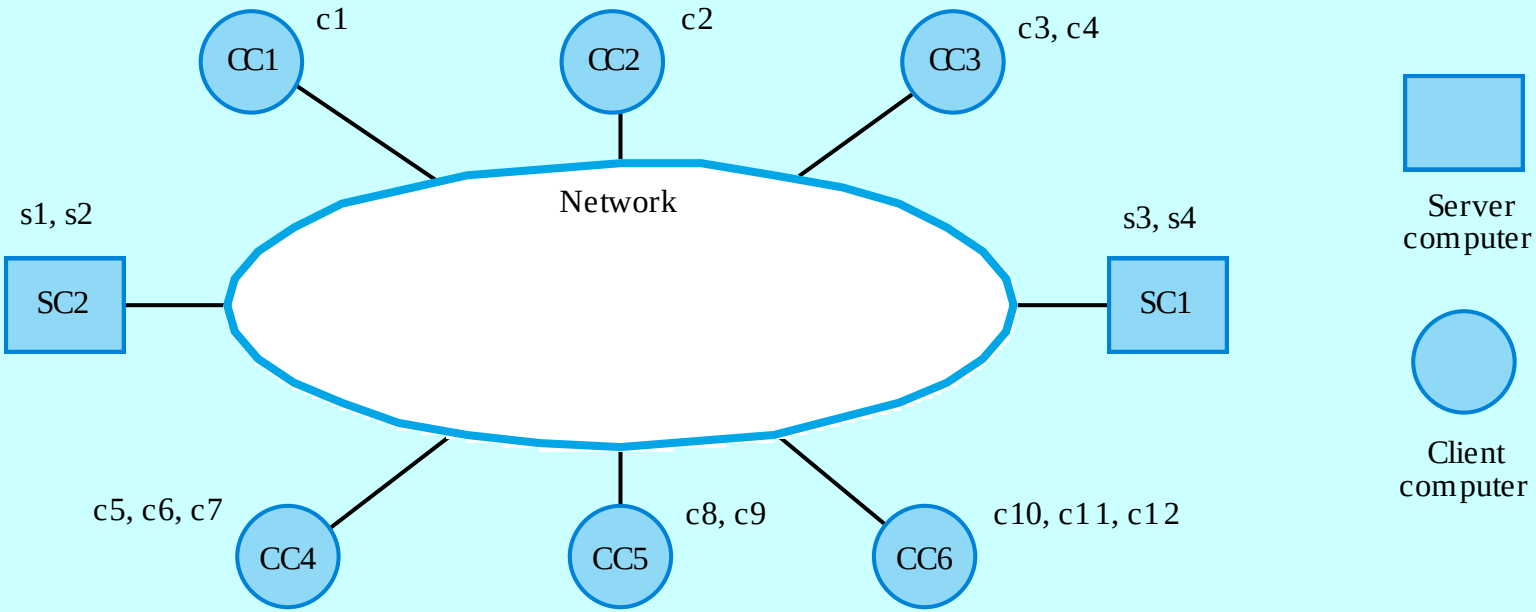
Client-server architectures

- 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



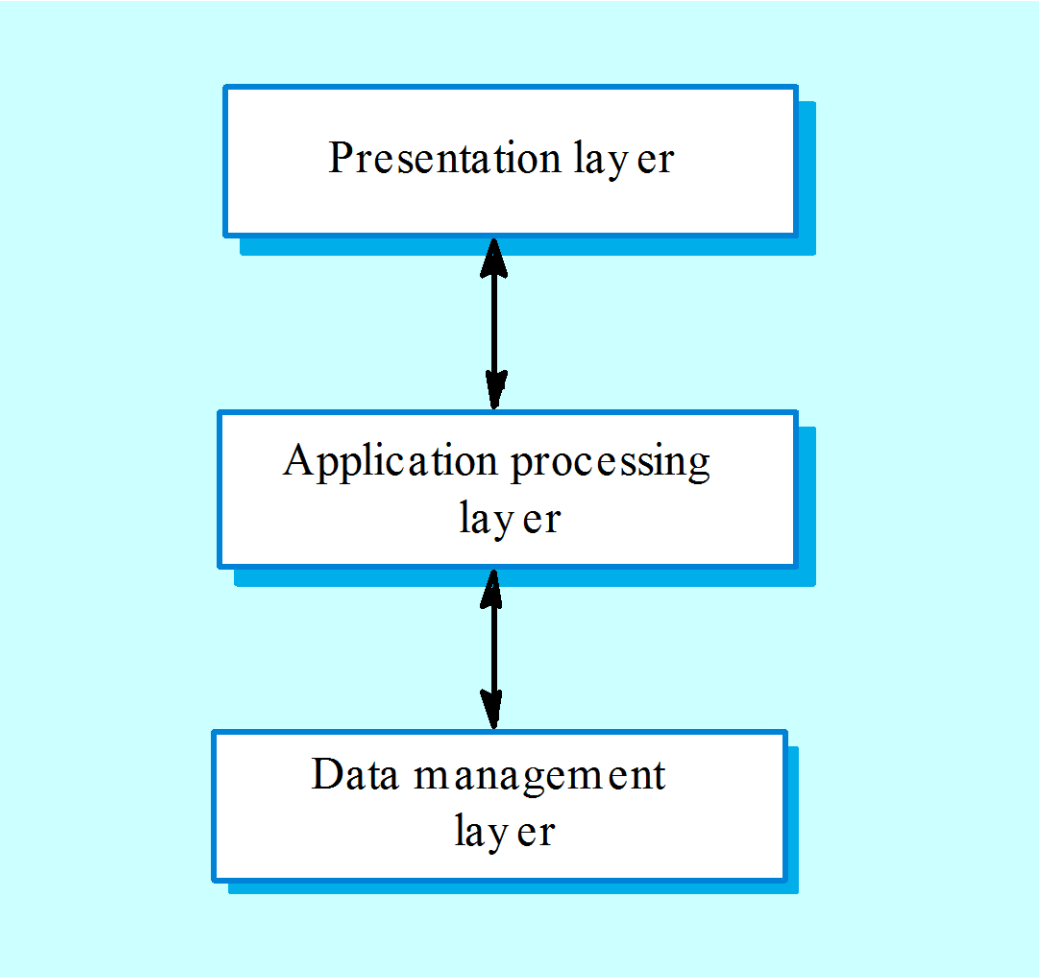
Computers in a C/S network



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.

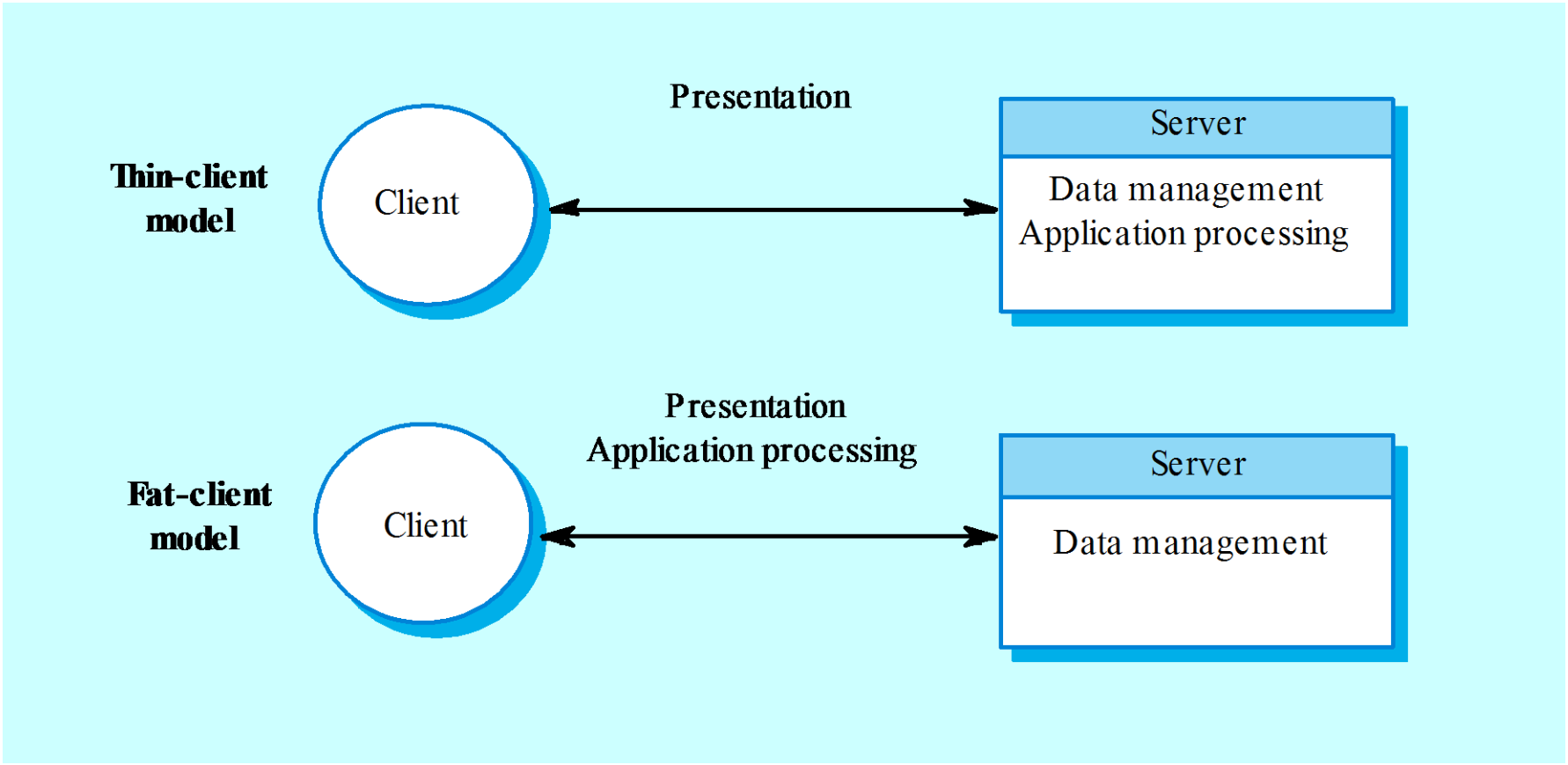
Application layers



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



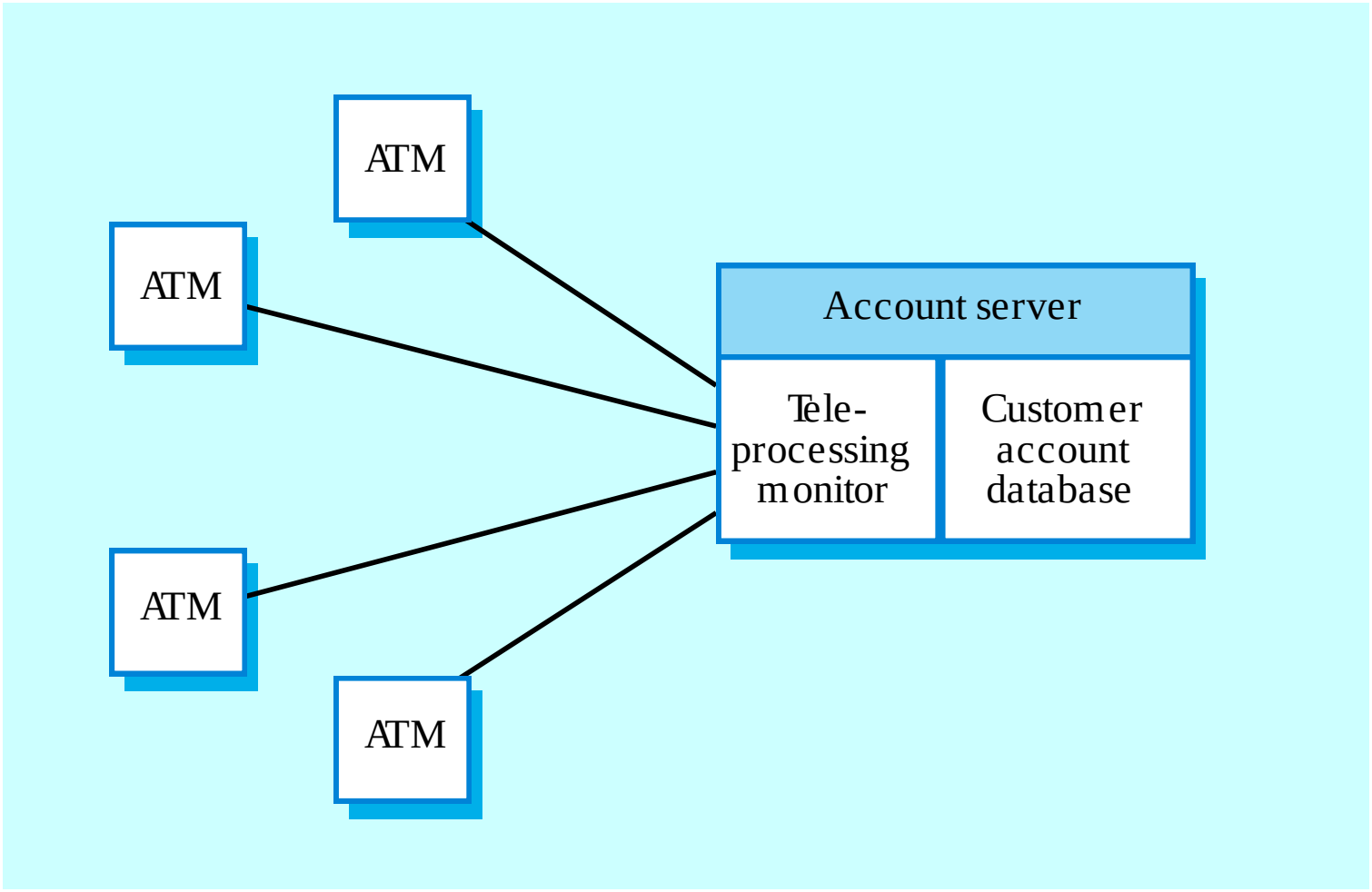
Thin client model

- 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.

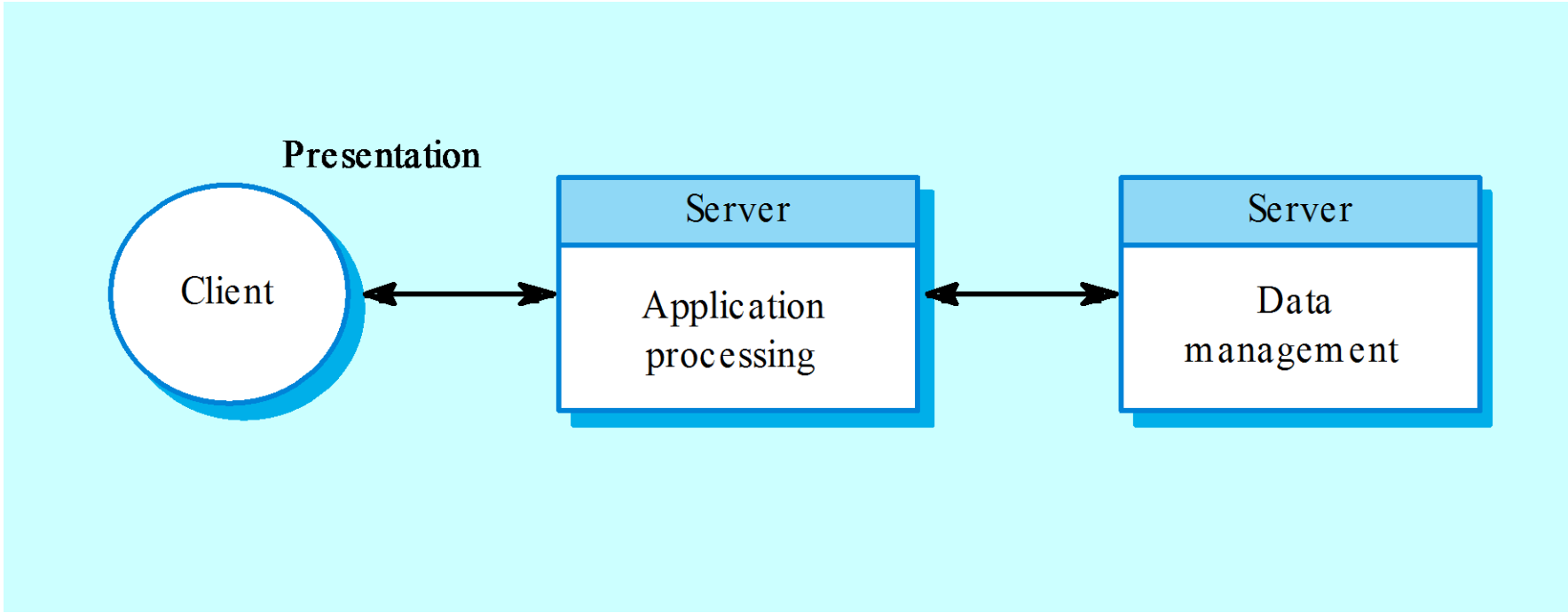
A client-server ATM system



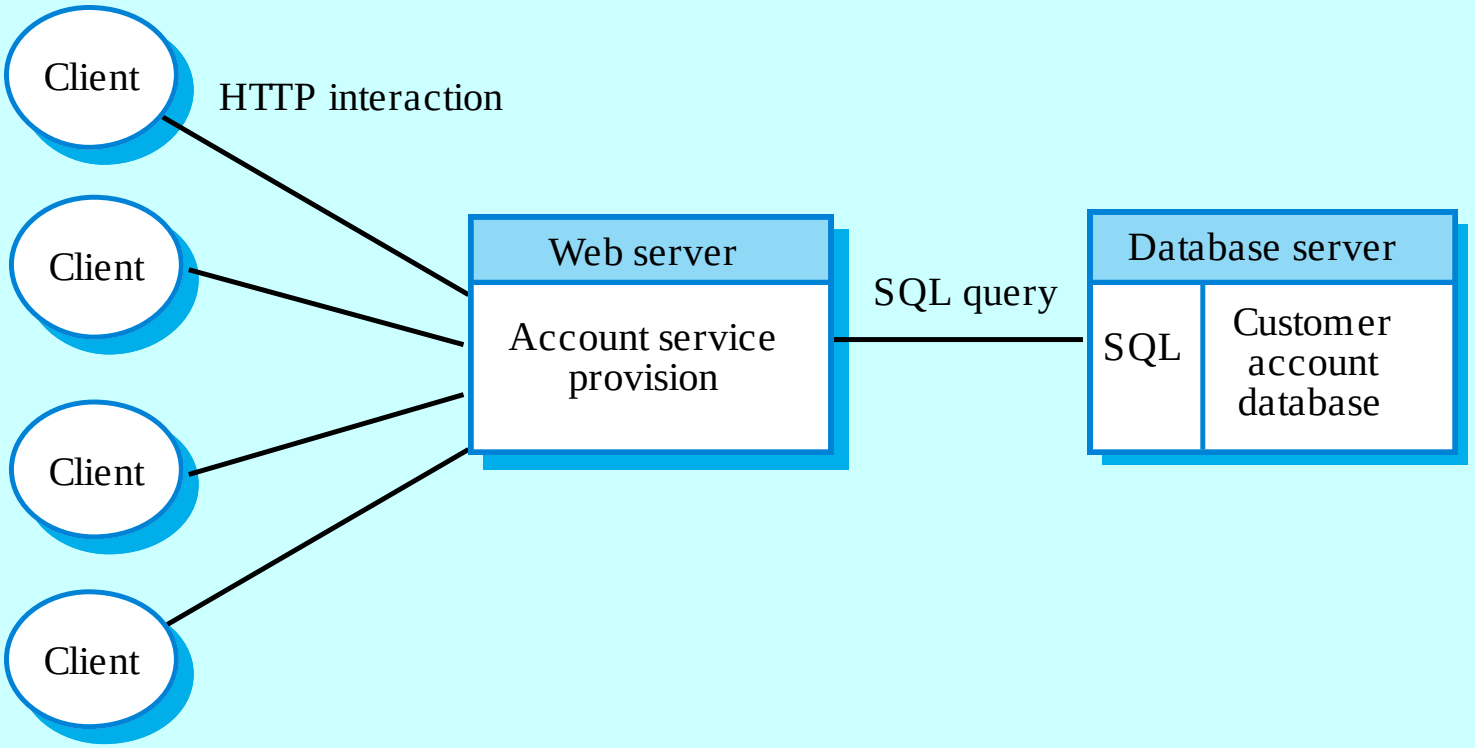
Three-tier architectures

- 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



Use of C/S architectures

Architecture	Applications
Two-tier C/S architecture with thin clients	Legacy system applications where separating application processing and data management is impractical. Computationally-intensive applications such as compilers with little or no data management. Data-intensive applications (browsing and querying) with little or no application processing.
Two-tier C/S architecture with fat clients	Applications where application processing is provided by off-the-shelf software (e.g. Microsoft Excel) on the client. Applications where computationally-intensive processing of data (e.g. data visualisation) is required. Applications with relatively stable end-user functionality used in an environment with well-established system management.
Three-tier or multi-tier C/S architecture	Large scale applications with hundreds or thousands of clients Applications where both the data and the application are volatile. Applications where data from multiple sources are integrated.

Assignment

- Explain Computer System Architecture

Research

- **21st Century *Computer Architecture* - Computing *Research***

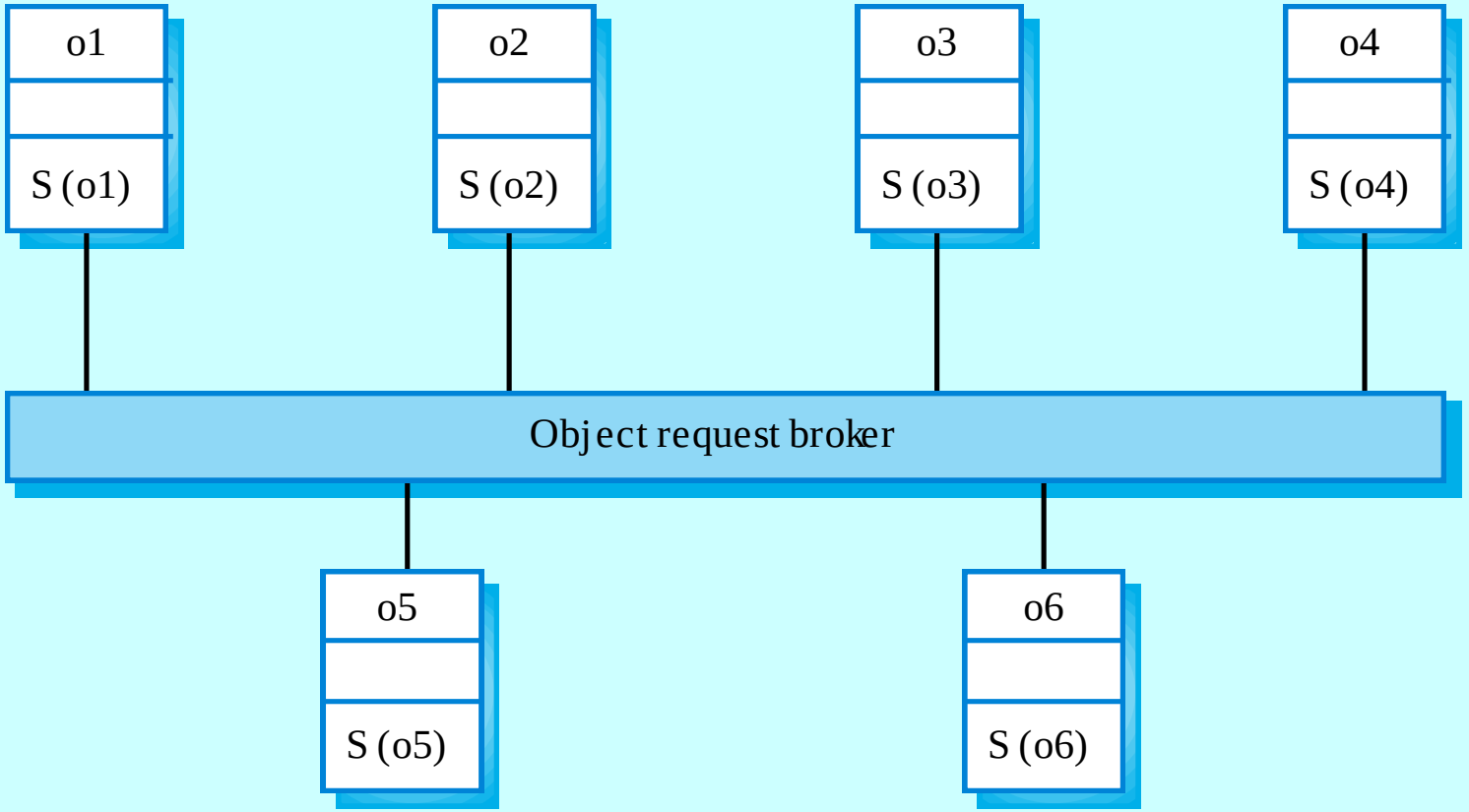
Distributed Systems Architectures

Lecture – 16

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.
- However, distributed object architectures are more complex to design than C/S systems.

Distributed object architecture



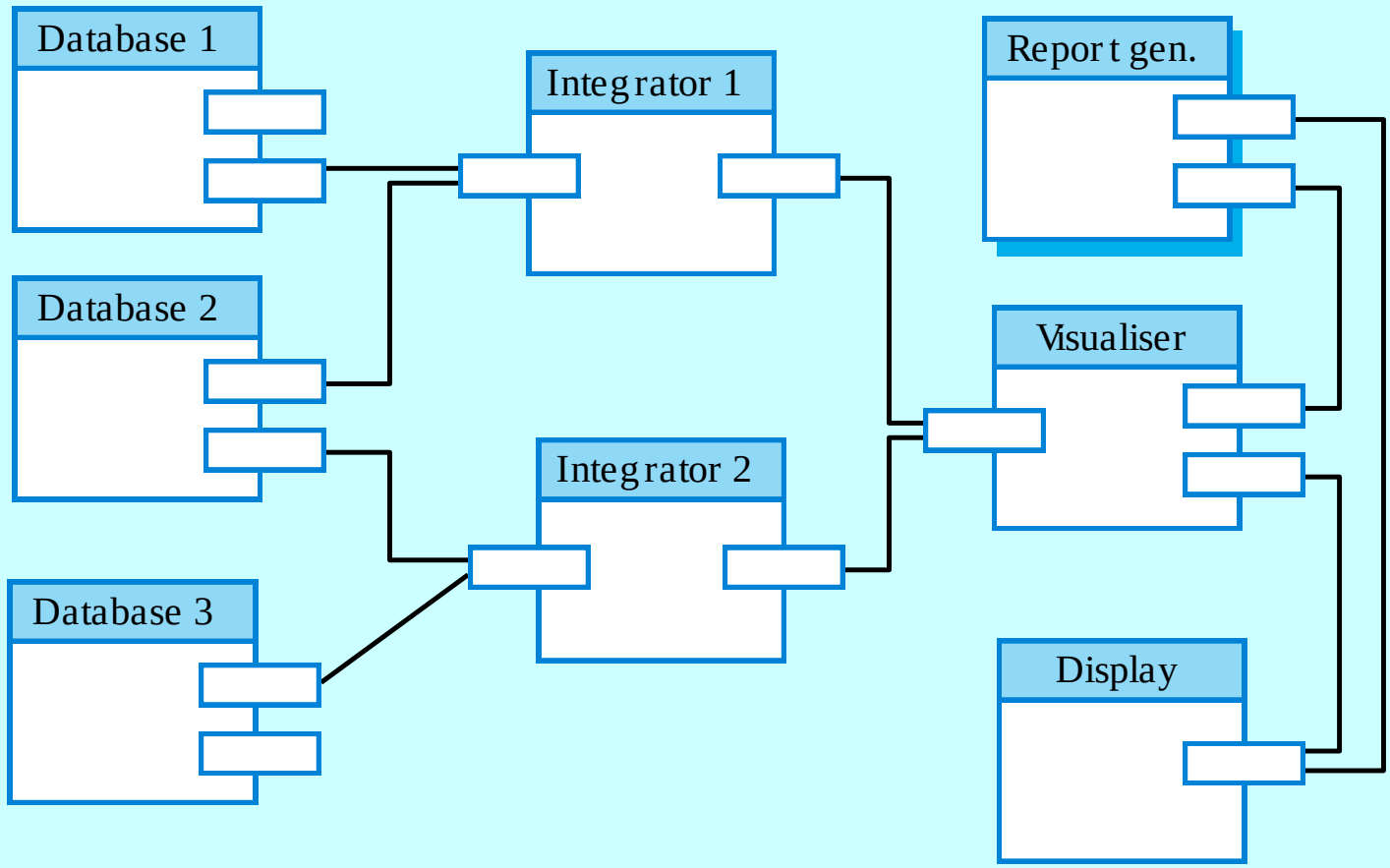
Advantages of distributed object architecture

- 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 common communication framework.

A data mining system



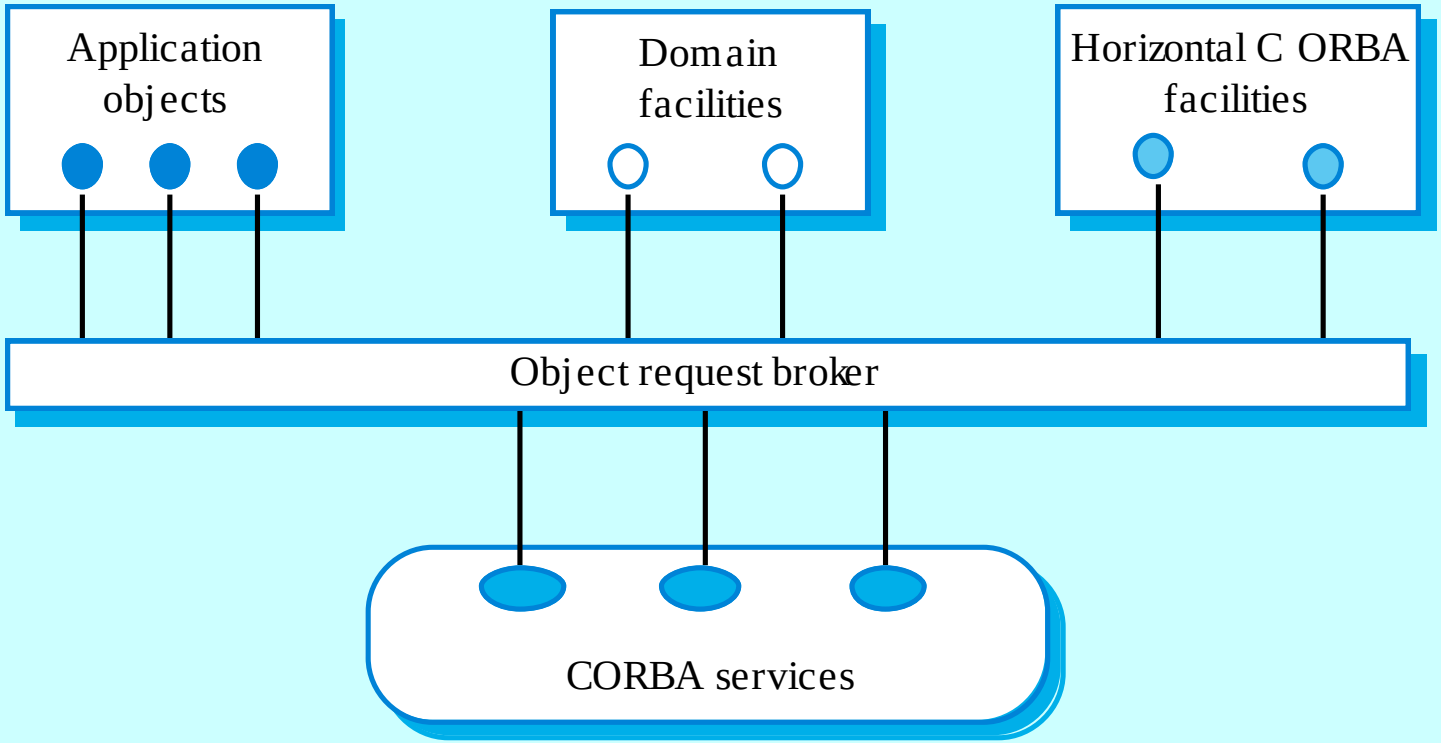
Data mining system

- The logical model of the system is not one of service provision where there are distinguished data management services.
- It allows the number of databases that are accessed to be increased without disrupting the system.
- It allows new types of relationship to be mined by adding new integrator objects.

CORBA

- CORBA is an international standard for an Object Request Broker - middleware to manage communications between distributed objects.
- Middleware for distributed computing is required at 2 levels:
  - At the logical communication level, the middleware allows objects on different computers to exchange data and control information;
  - At the component level, the middleware provides a basis for developing compatible components. CORBA component standards have been defined.

CORBA application structure



Application structure

- Application objects.
- Standard objects, defined by the OMG, for a specific domain e.g. insurance.
- Fundamental CORBA services such as directories and security management.
- Horizontal (i.e. cutting across applications) facilities such as user interface facilities.

CORBA standards

- An object model for application objects
  - A CORBA object is an encapsulation of state with a well-defined, language-neutral interface defined in an IDL (interface definition language).
- An object request broker that manages requests for object services.
- A set of general object services of use to many distributed applications.
- A set of common components built on top of these services.

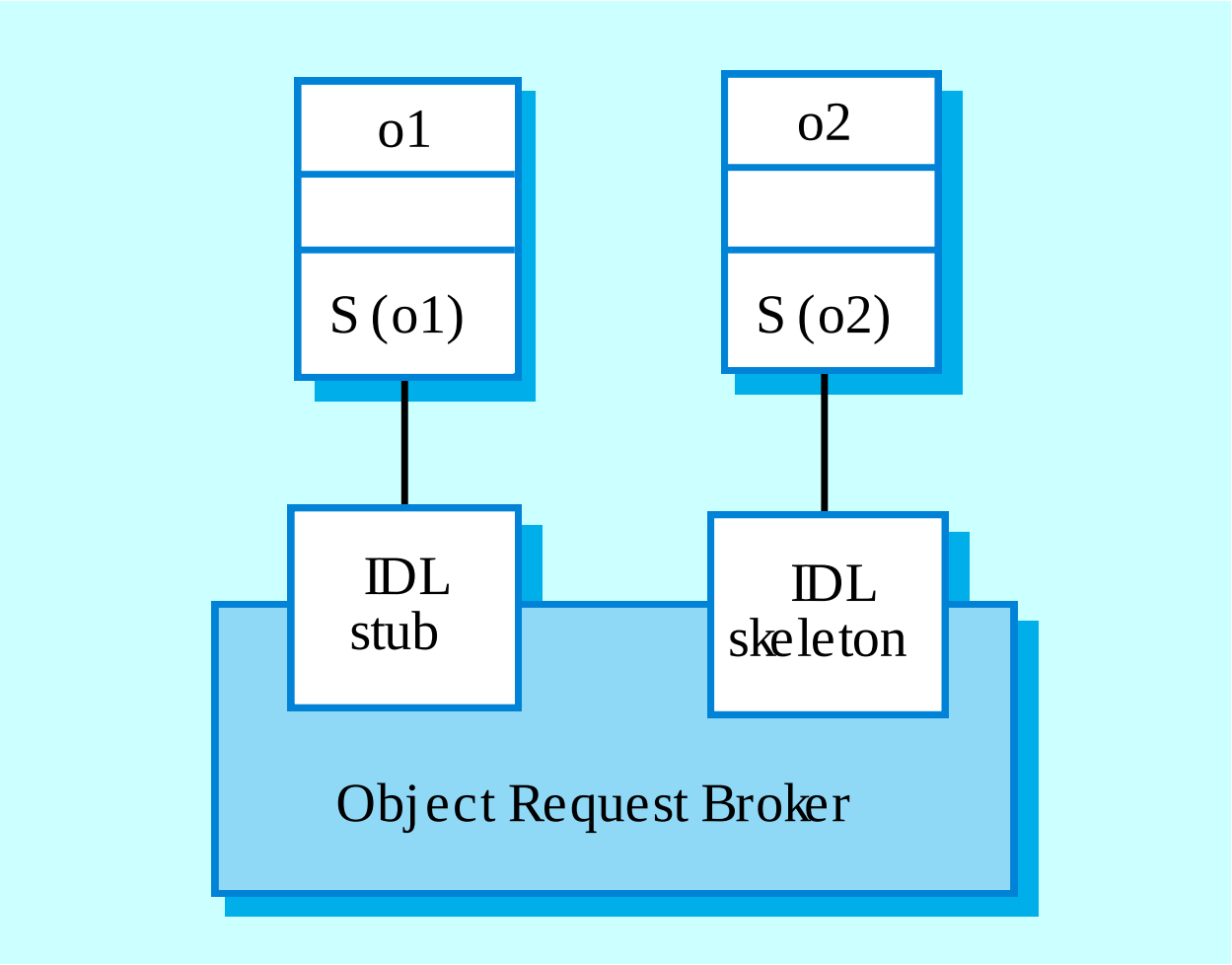
CORBA objects

- CORBA objects are comparable, in principle, to objects in C++ and Java.
- They MUST have a separate interface definition that is expressed using a common language (IDL) similar to C++.
- There is a mapping from this IDL to programming languages (C++, Java, etc.).
- Therefore, objects written in different languages can communicate with each other.

Object request broker (ORB)

- The ORB handles object communications. It knows of all objects in the system and their interfaces.
- Using an ORB, the calling object binds an IDL stub that defines the interface of the called object.
- Calling this stub results in calls to the ORB which then calls the required object through a published IDL skeleton that links the interface to the service implementation.

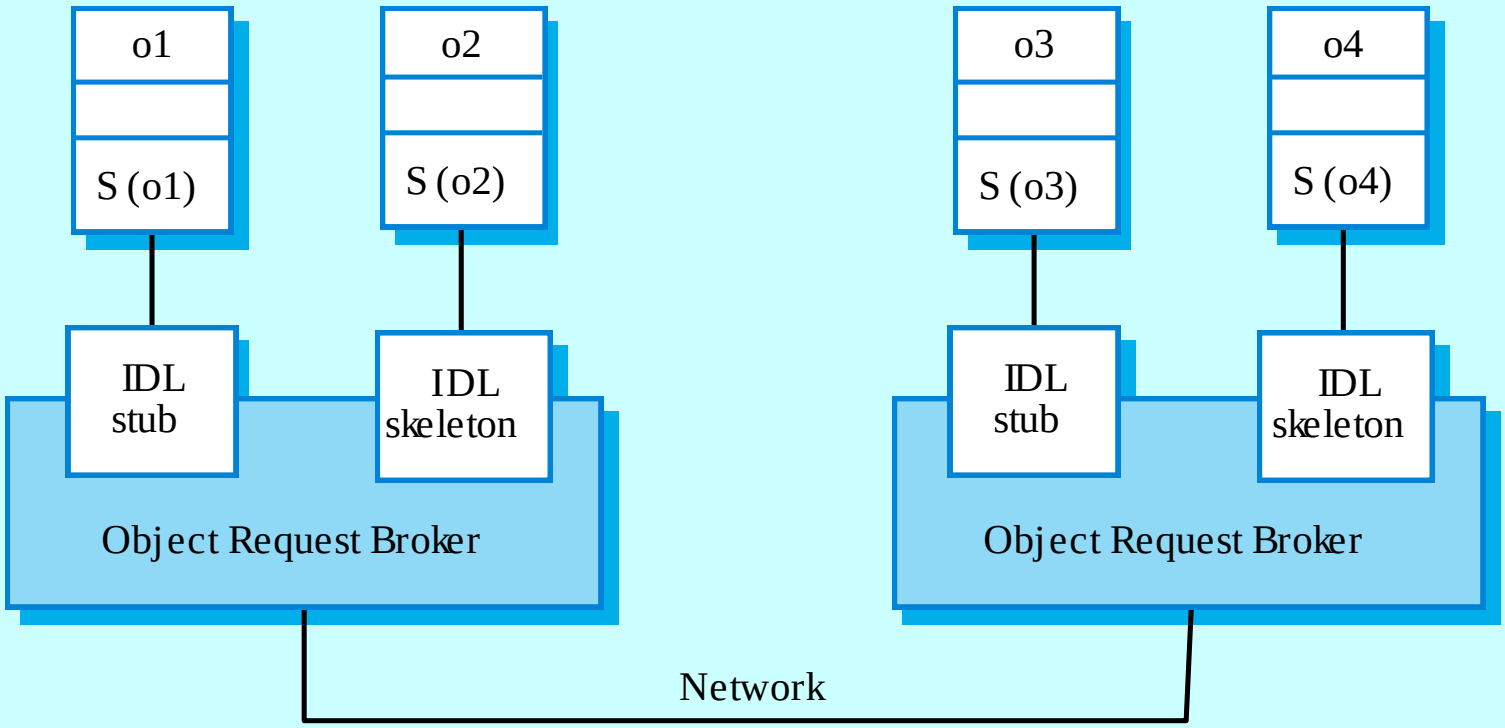
ORB-based object communications



Inter-ORB communications

- ORBs are not usually separate programs but are a set of objects in a library that are linked with an application when it is developed.
- ORBs handle communications between objects executing on the sane machine.
- Several ORBS may be available and each computer in a distributed system will have its own ORB.
- Inter-ORB communications are used for distributed object calls.

Inter-ORB communications



CORBA services

- Naming and trading services
  - These allow objects to discover and refer to other objects on the network.
- Notification services
  - These allow objects to notify other objects that an event has occurred.
- Transaction services
  - These support atomic transactions and rollback on failure.

Inter-organisational computing

- For security and inter-operability reasons, most distributed computing has been implemented at the enterprise level.
- Local standards, management and operational processes apply.
- Newer models of distributed computing have been designed to support inter-organisational computing where different nodes are located in different organisations.

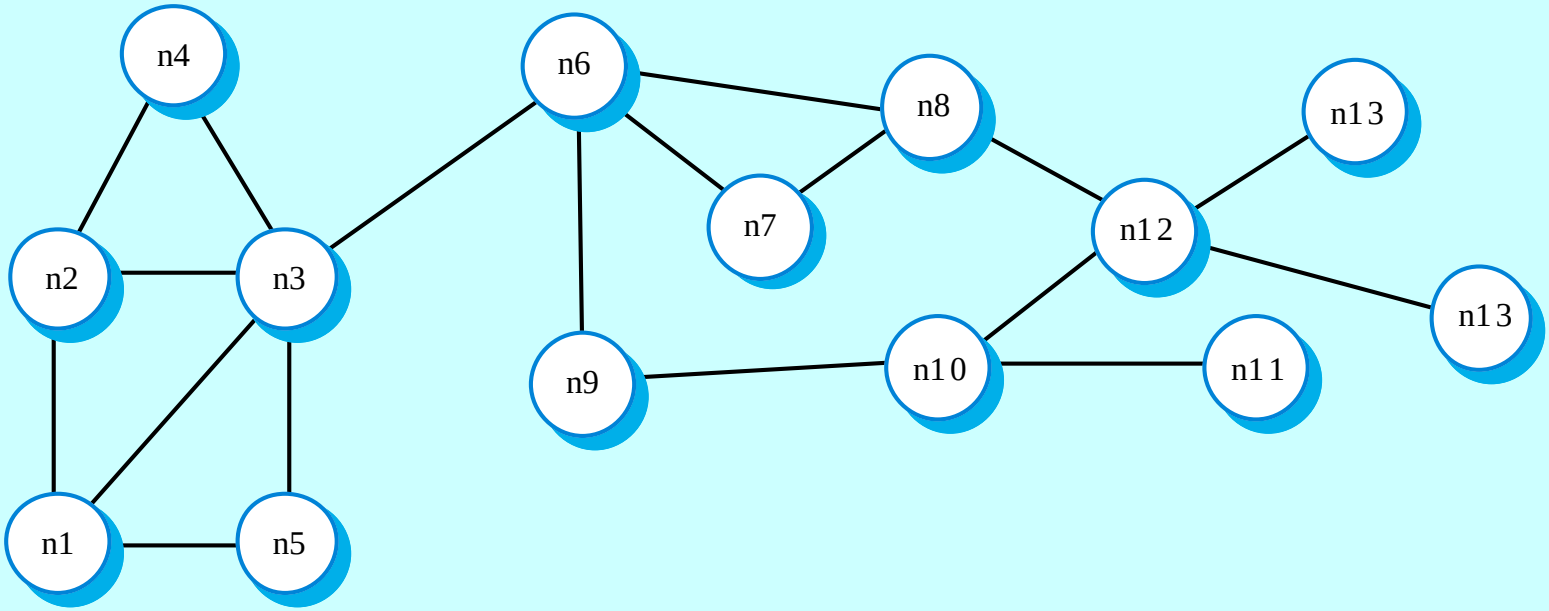
Peer-to-peer architectures

- Peer to peer (p2p) systems are decentralised systems where computations may be carried out by any node in the network.
- The overall system is designed to take advantage of the computational power and storage of a large number of networked computers.
- Most p2p systems have been personal systems but there is increasing business use of this technology.

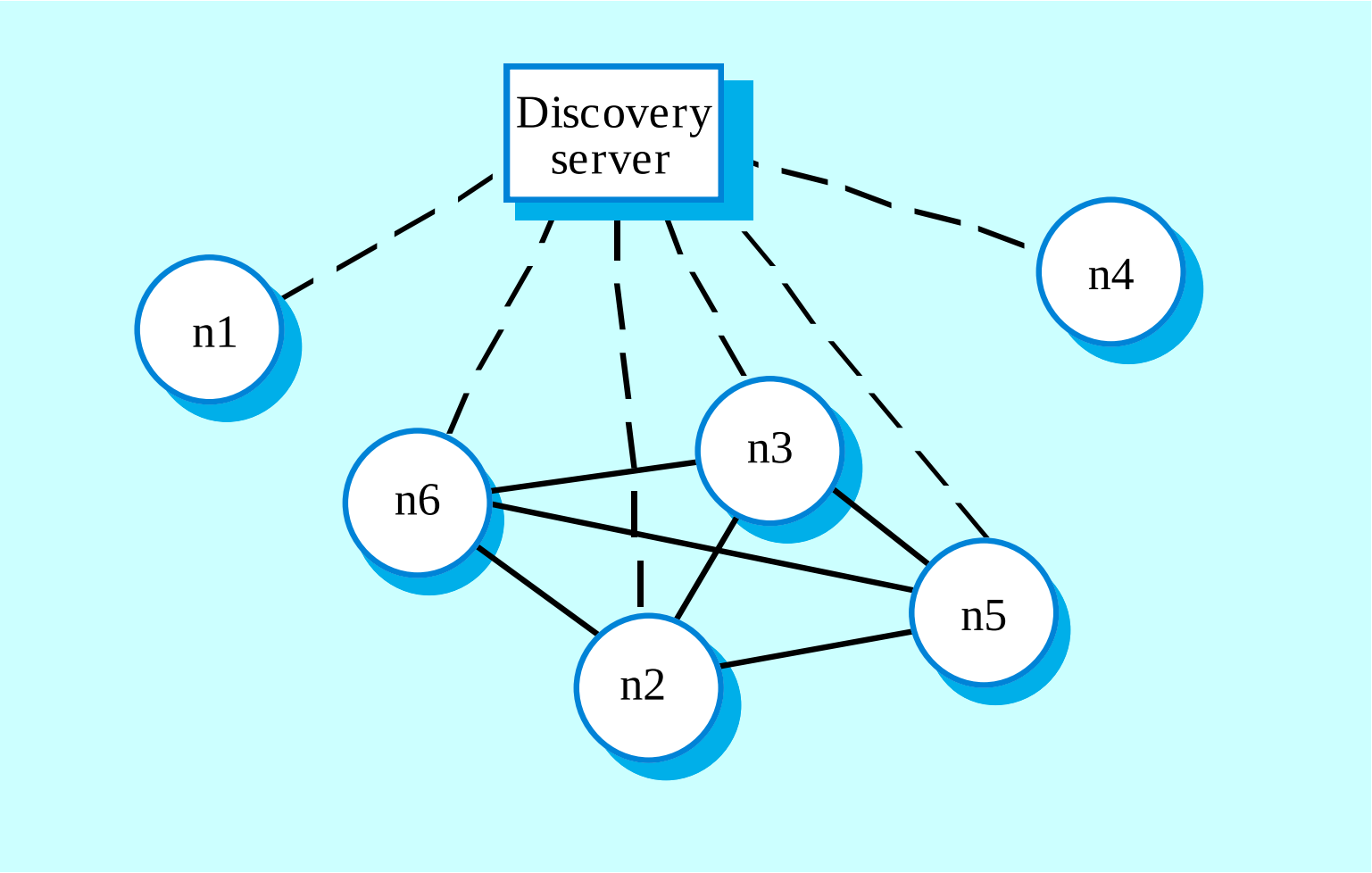
P2p architectural models

- The logical network architecture
  - Decentralised architectures;
  - Semi-centralised architectures.
- Application architecture
  - The generic organisation of components making up a p2p application.
- Focus here on network architectures.

Decentralised p2p architecture



Semi-centralised p2p architecture



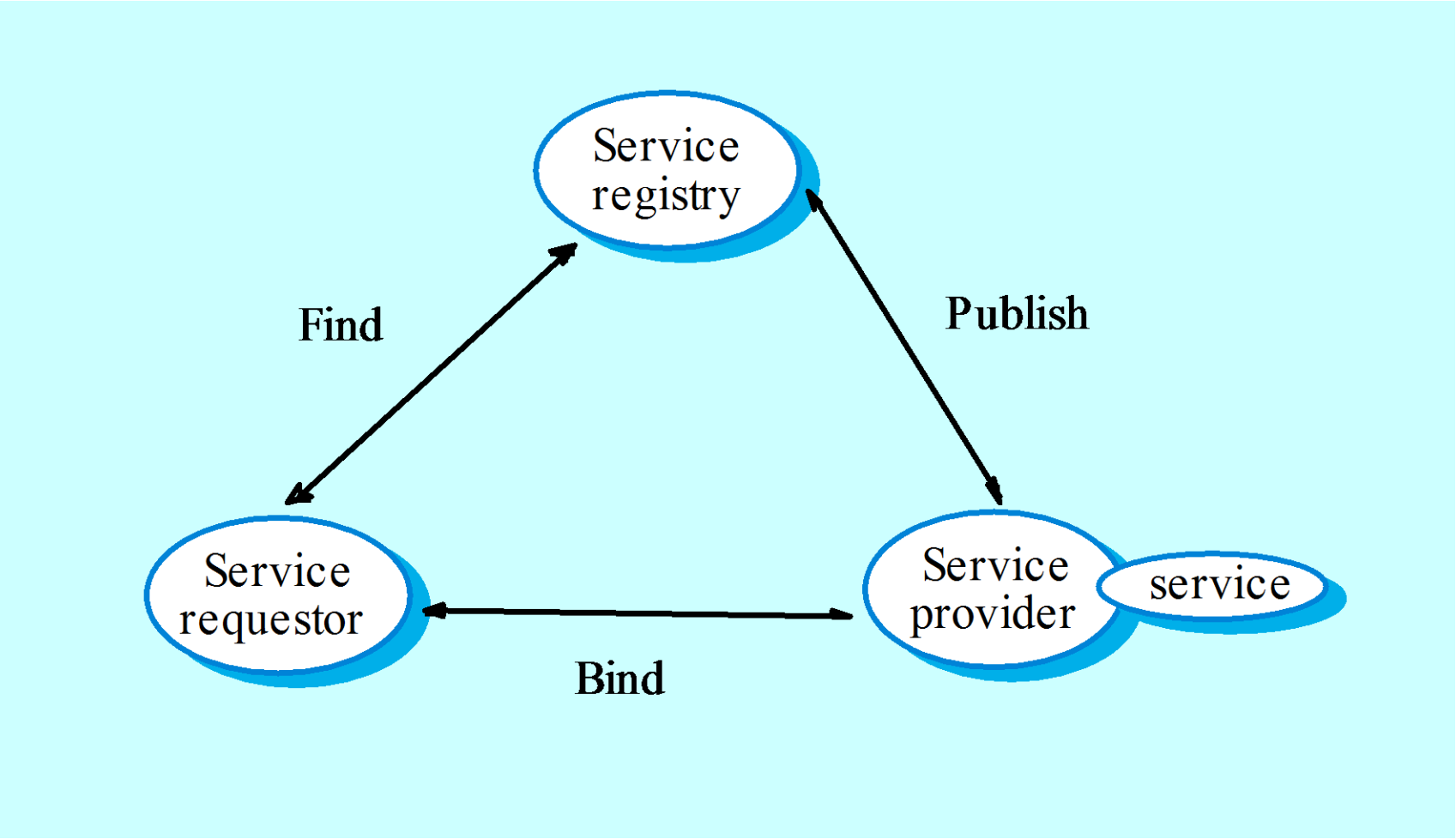
Service-oriented architectures

- Based around the notion of externally provided services (web services).
- A web service is a standard approach to making a reusable component available and accessible across the web
  - A tax filing service could provide support for users to fill in their tax forms and submit these to the tax authorities.

A generic service

- *An act or performance offered by one party to another. Although the process may be tied to a physical product, the performance is essentially intangible and does not normally result in ownership of any of the factors of production.*
- Service provision is therefore independent of the application using the service.

Web services



Services and distributed objects

- Provider independence.
- Public advertising of service availability.
- Potentially, run-time service binding.
- Opportunistic construction of new services through composition.
- Pay for use of services.
- Smaller, more compact applications.
- Reactive and adaptive applications.

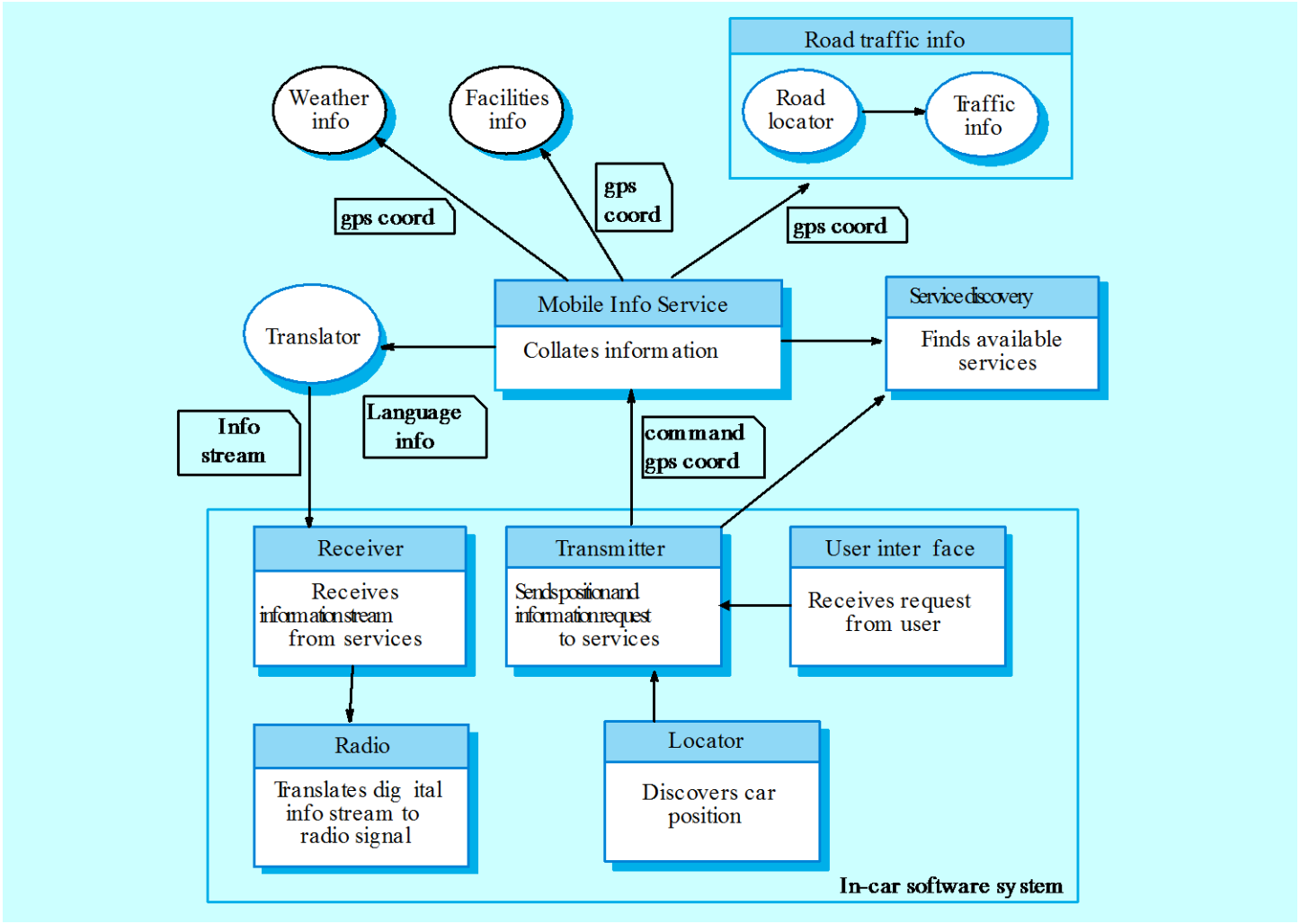
Services standards

- Services are based on agreed, XML-based standards so can be provided on any platform and written in any programming language.
- Key standards
  - SOAP - Simple Object Access Protocol;
  - WSDL - Web Services Description Language;
  - UDDI - Universal Description, Discovery and Integration.

Services scenario

- An in-car information system provides drivers with information on weather, road traffic conditions, local information etc. This is linked to car radio so that information is delivered as a signal on a specific radio channel.
- The car is equipped with GPS receiver to discover its position and, based on that position, the system accesses a range of information services. Information may be delivered in the driver’s specified language.

Automotive system



Assignment

- Explain Distributed System Architecture

Research

- **An *Architecture* for A Scalable Wide Area *Distributed System***