Distributed Systems Architectures

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.

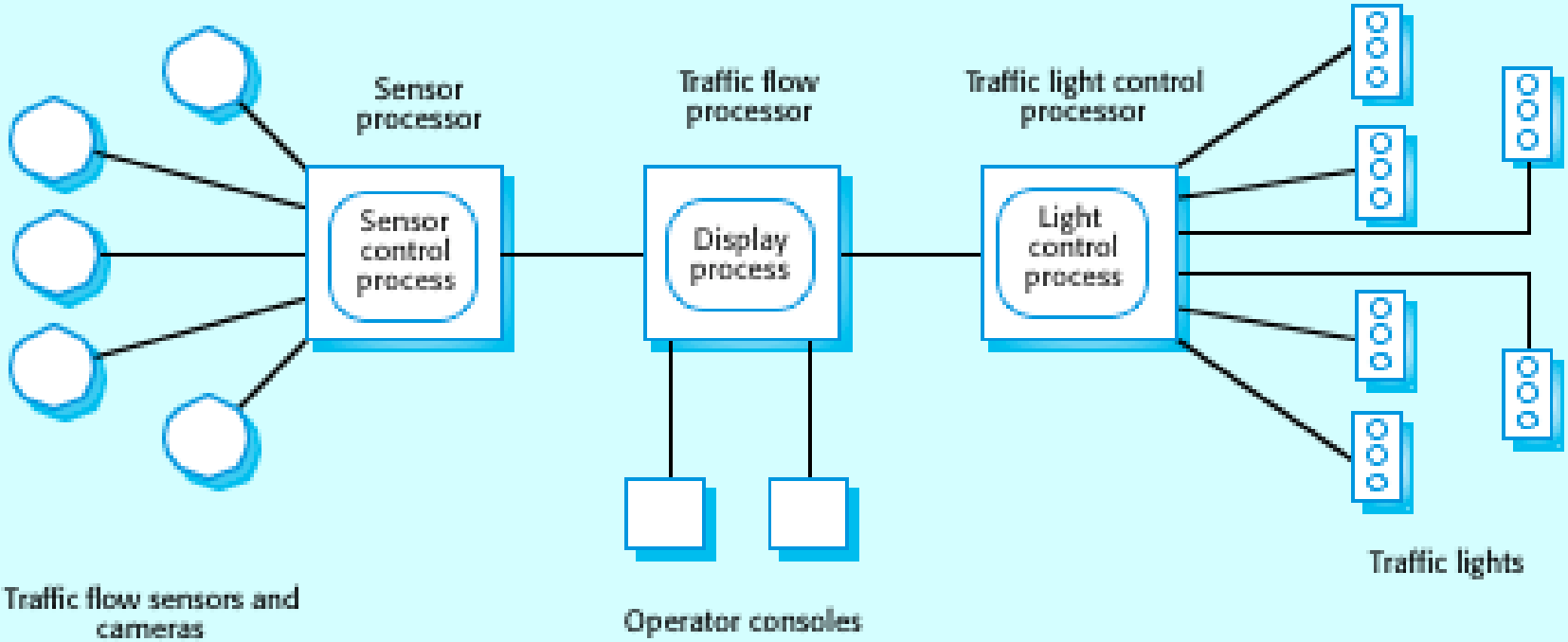
Middleware

- 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.

Multiprocessor architectures

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

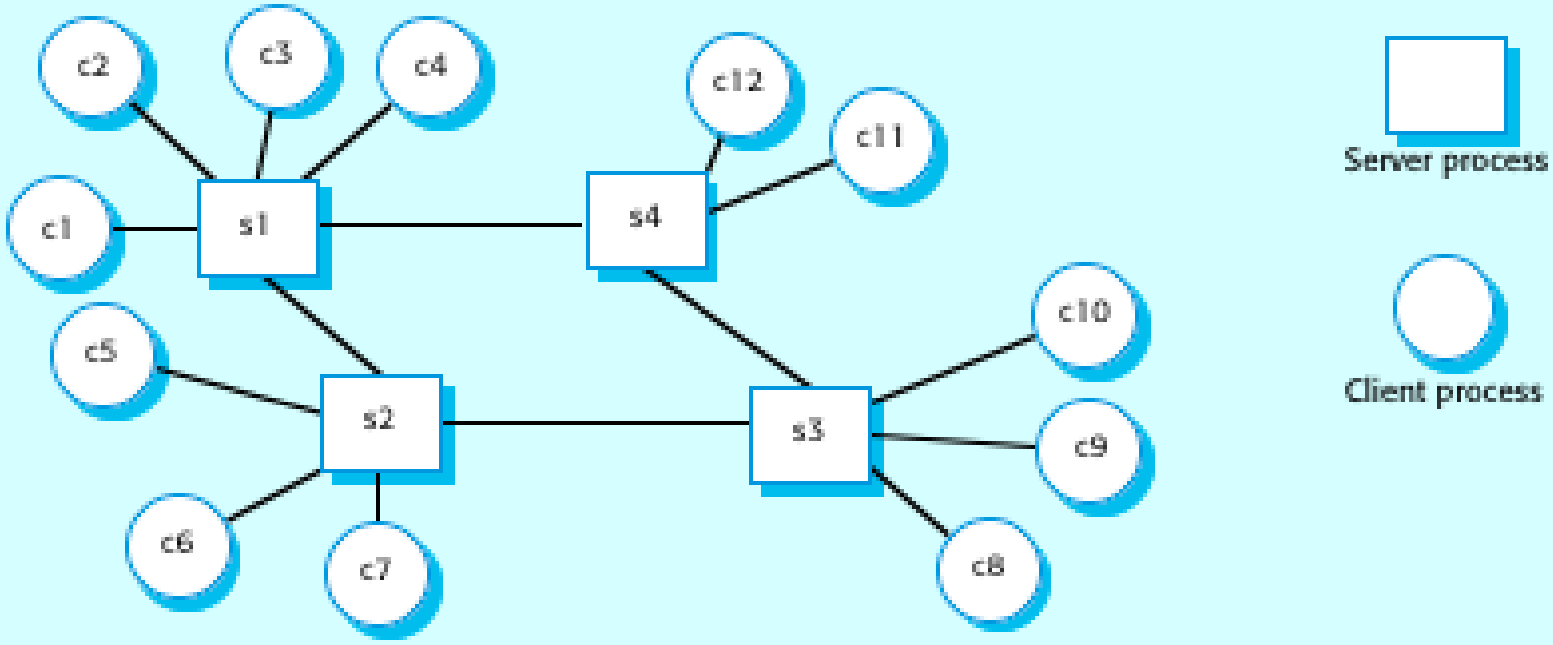
A multiprocessor traffic control system



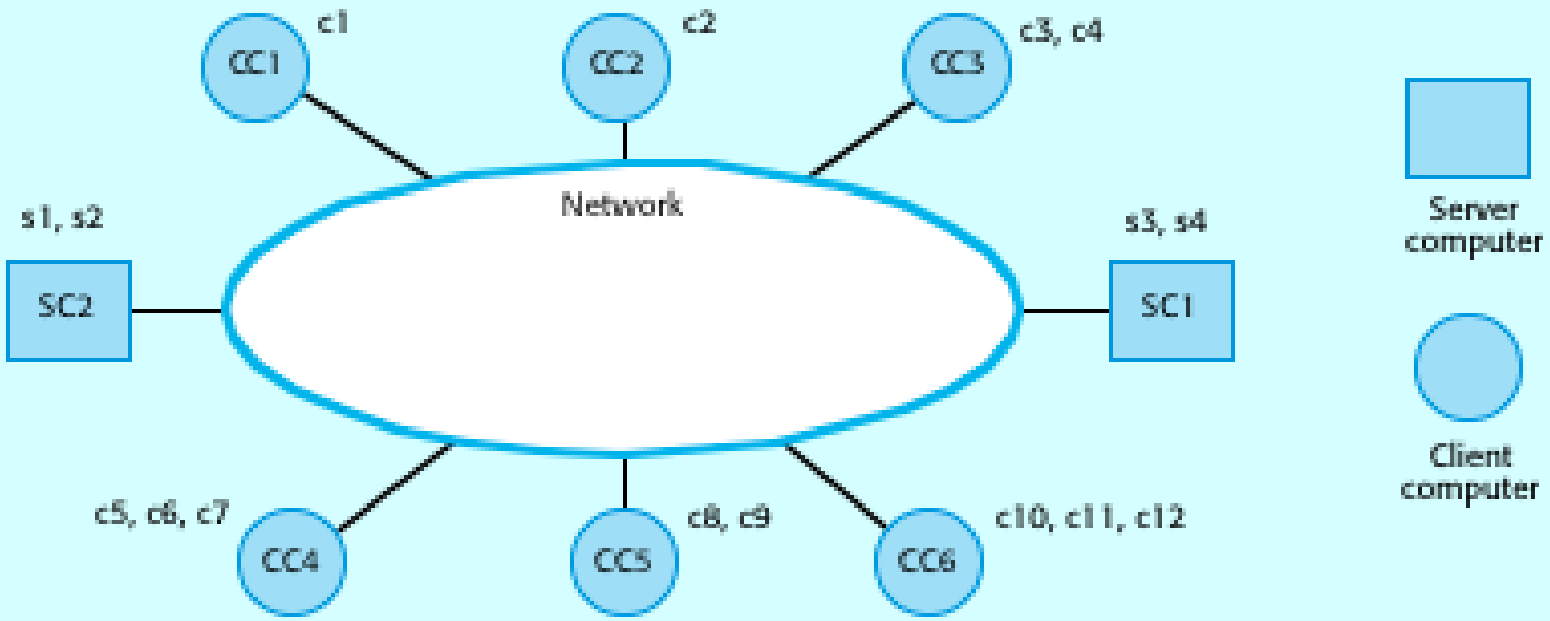
Client-server architectures

- 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



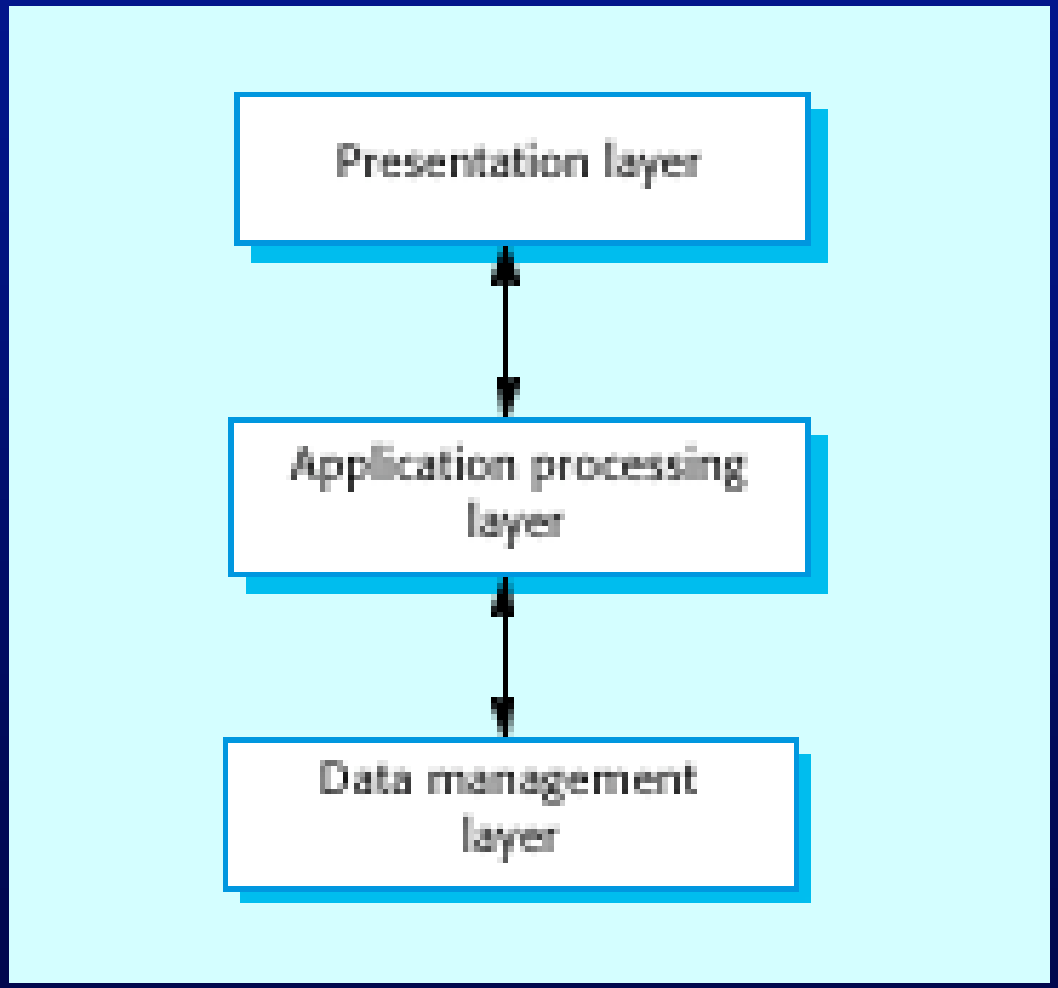
Computers in a C/S network



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.

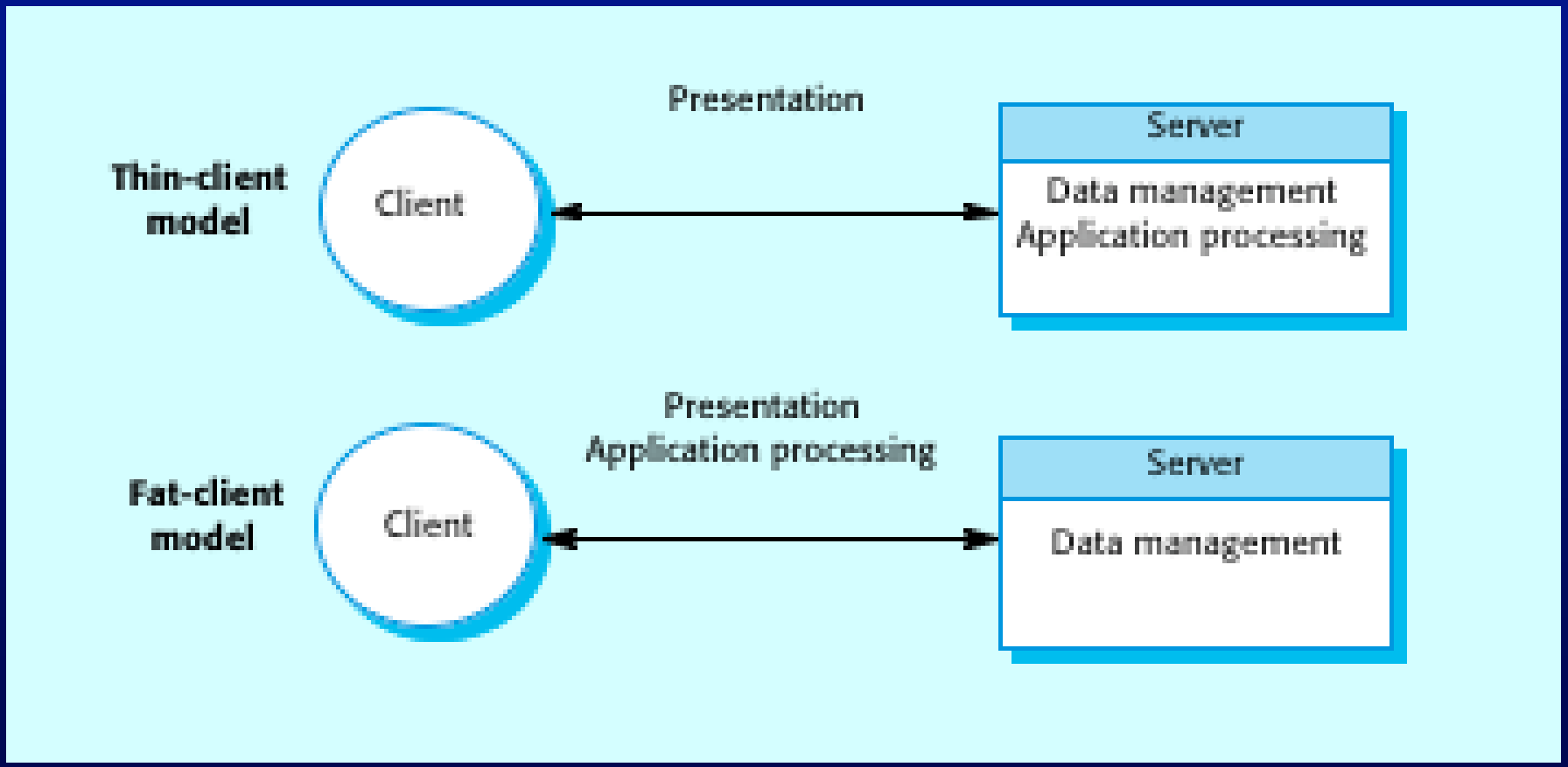
Application layers



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



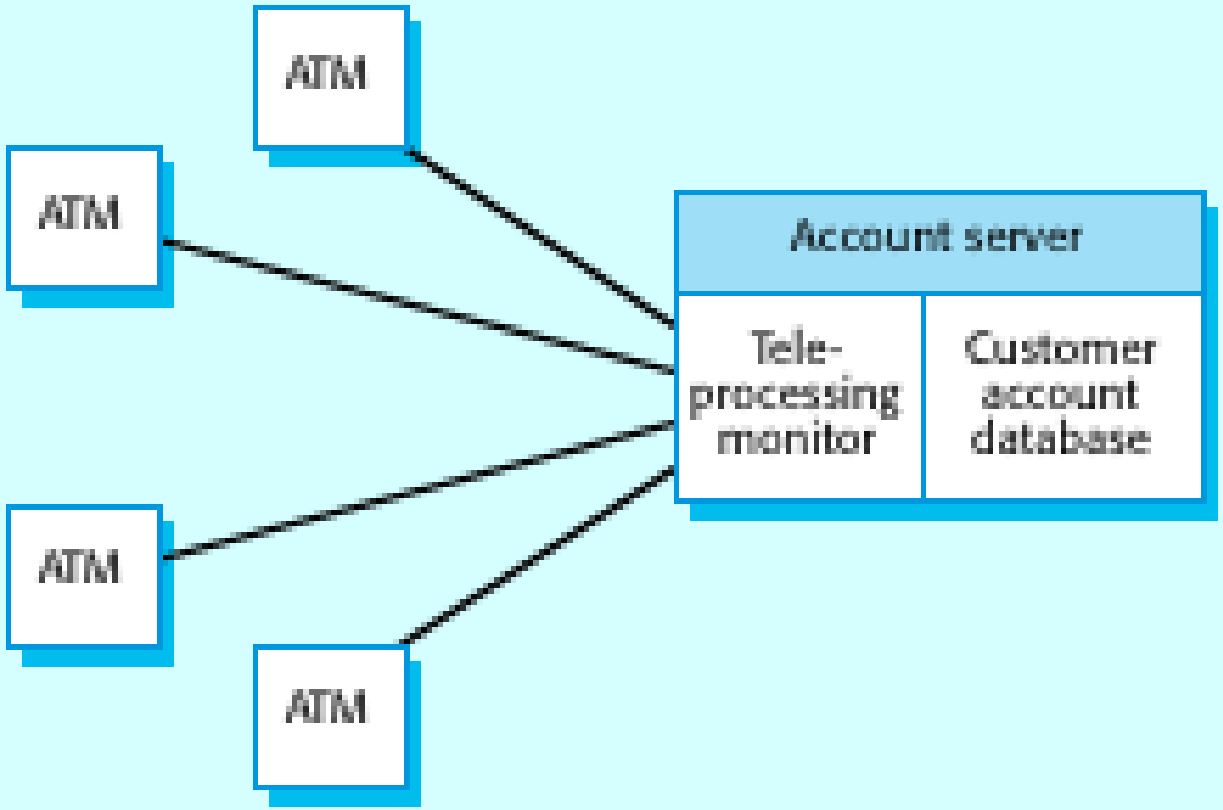
Thin client model

- 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.

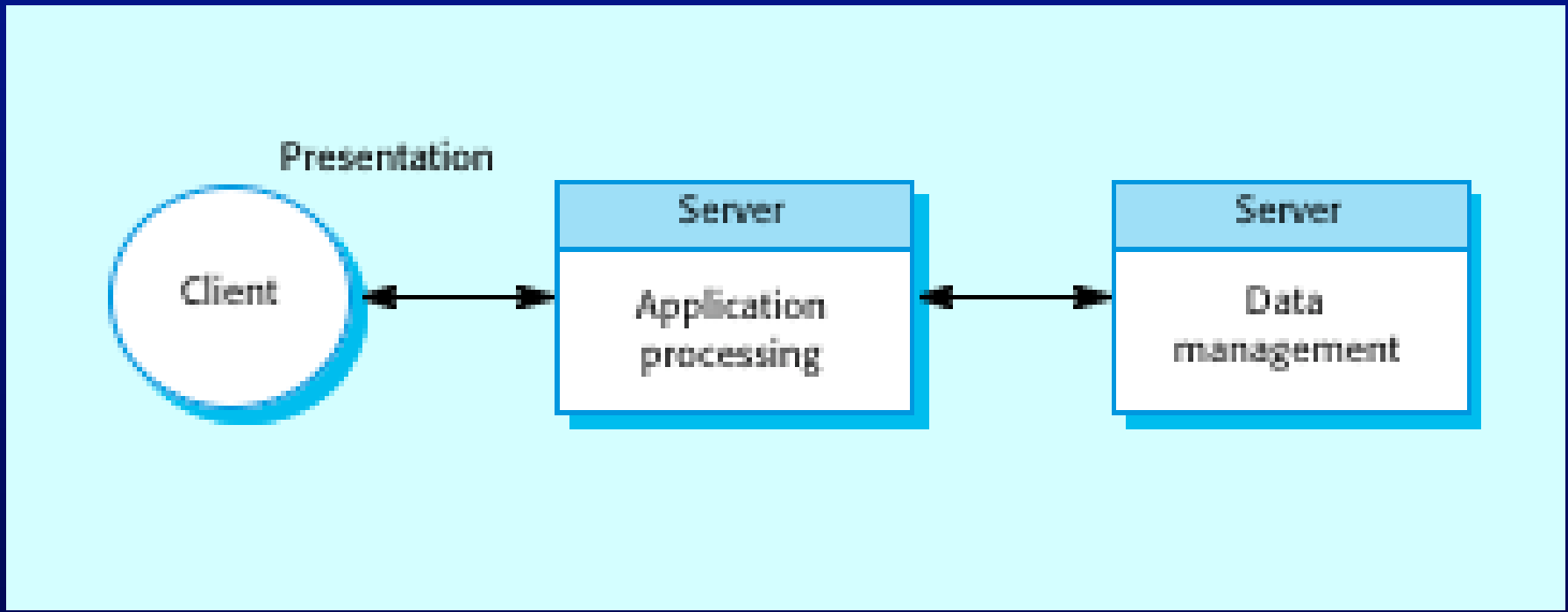
A client-server ATM system



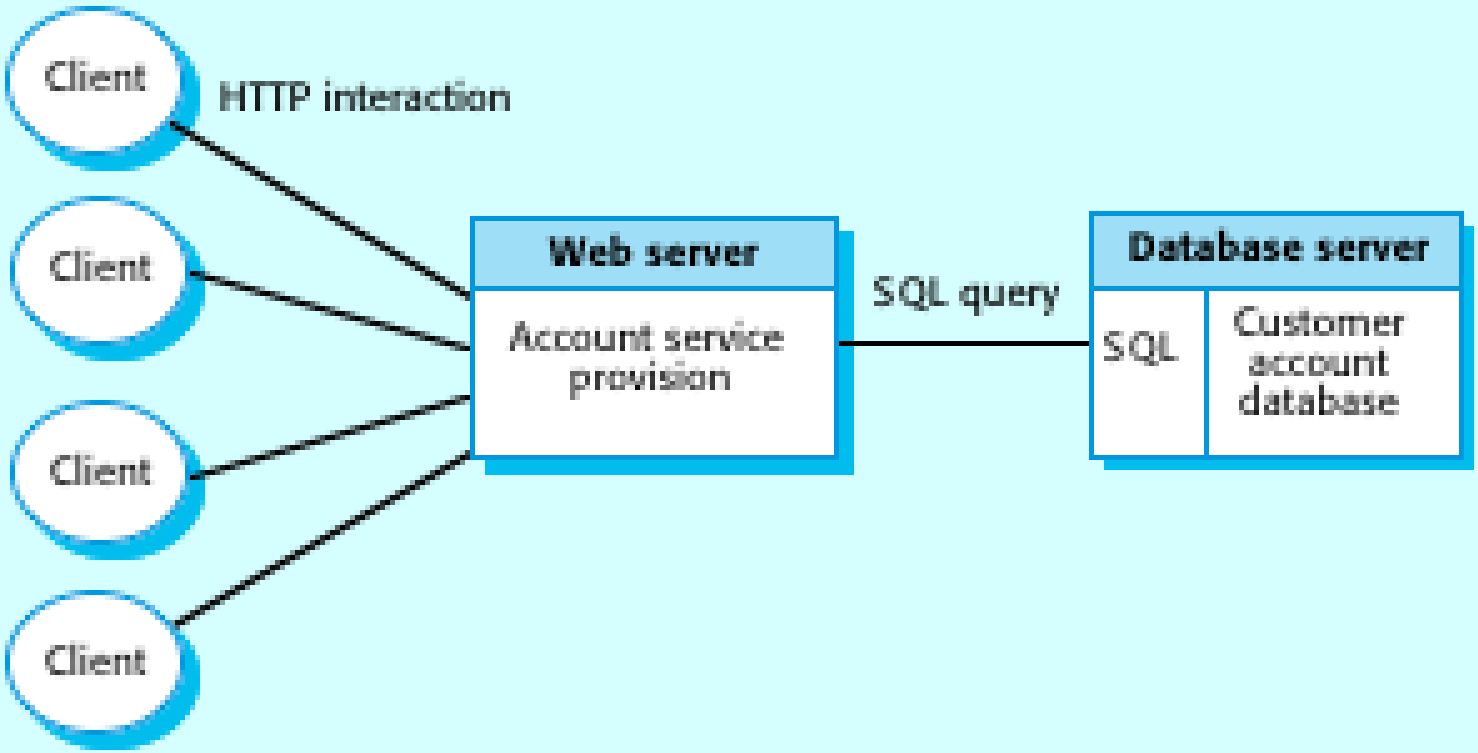
Three-tier architectures

- 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



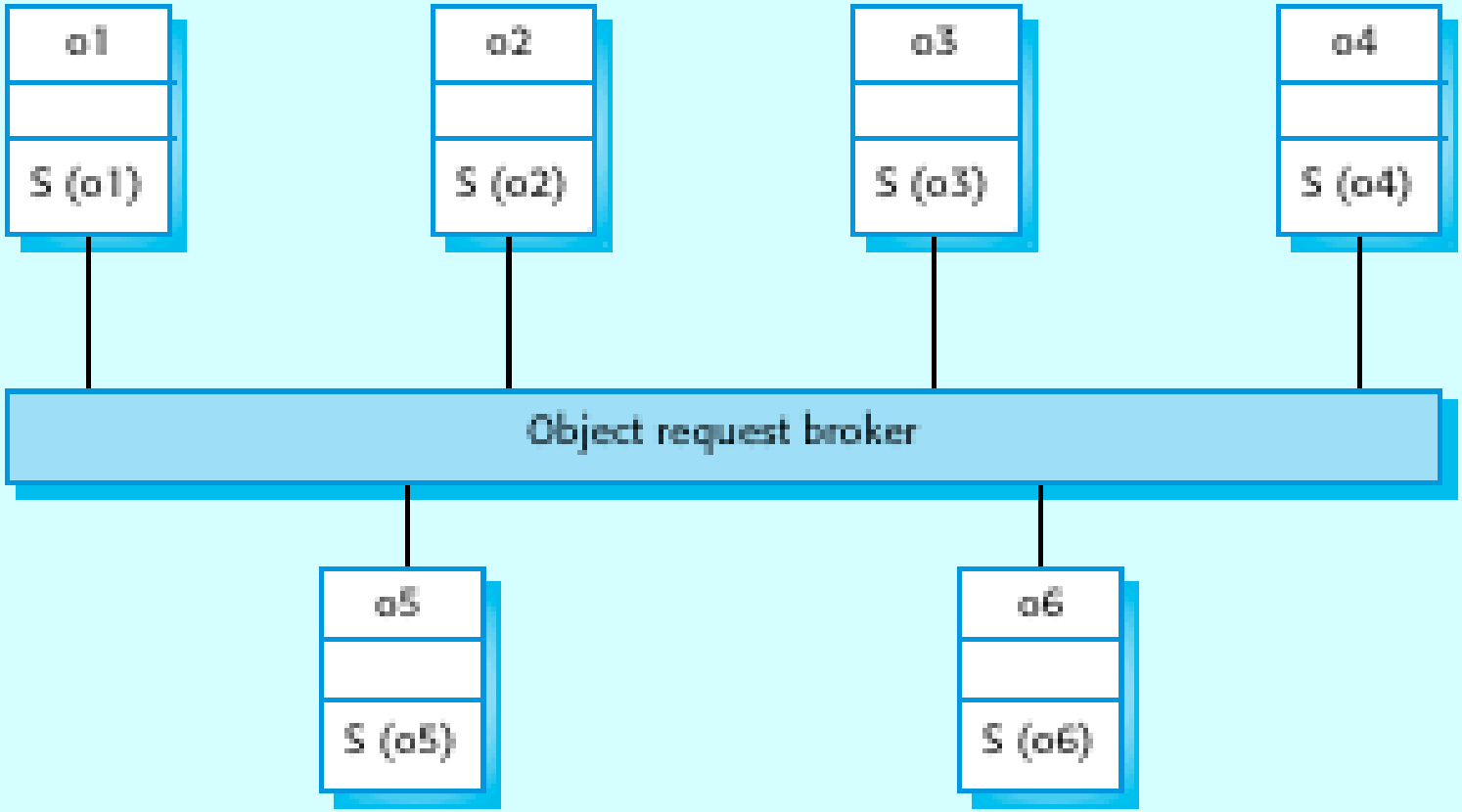
Use of C/S architectures

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.

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



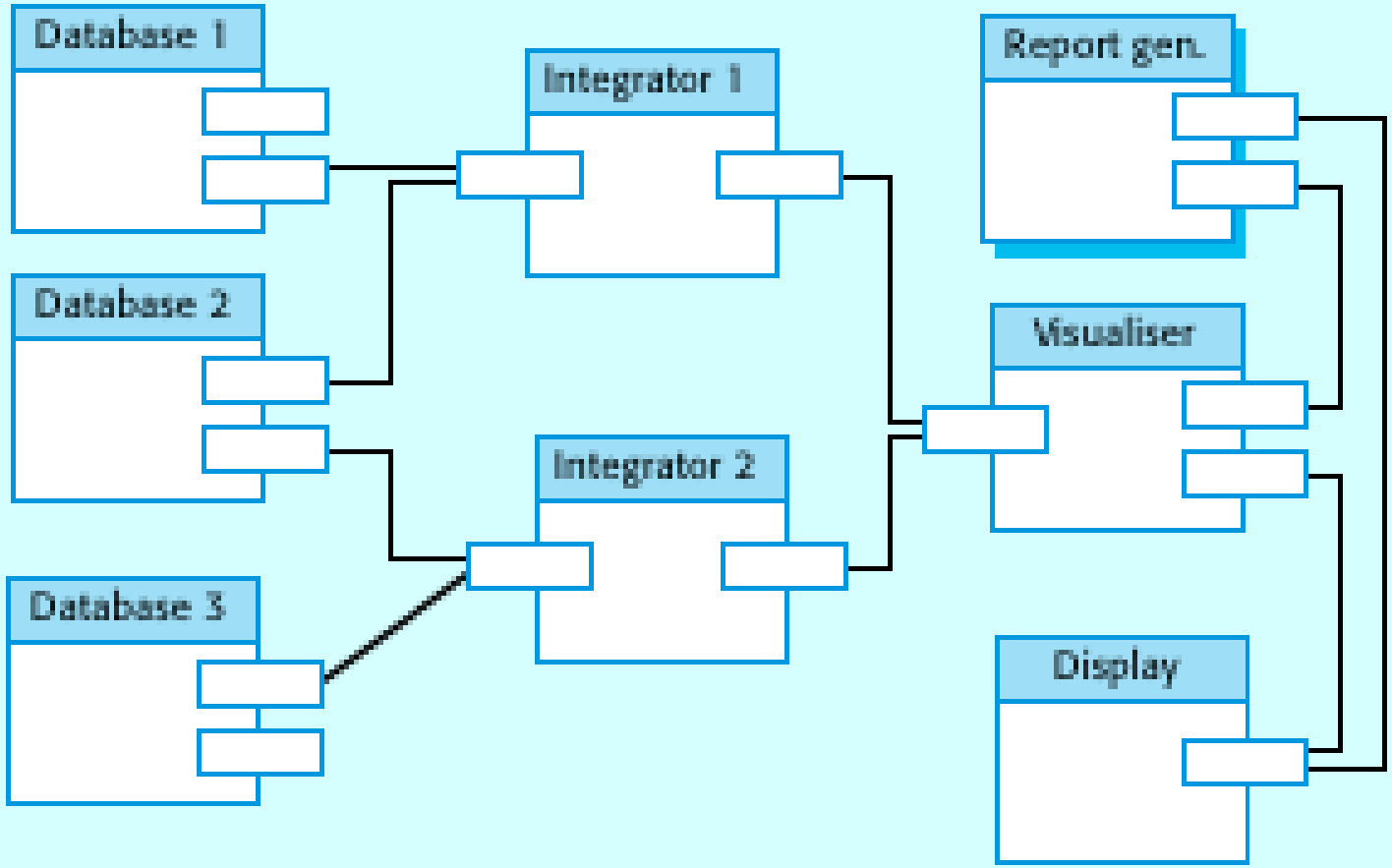
Advantages of distributed object architecture

- 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



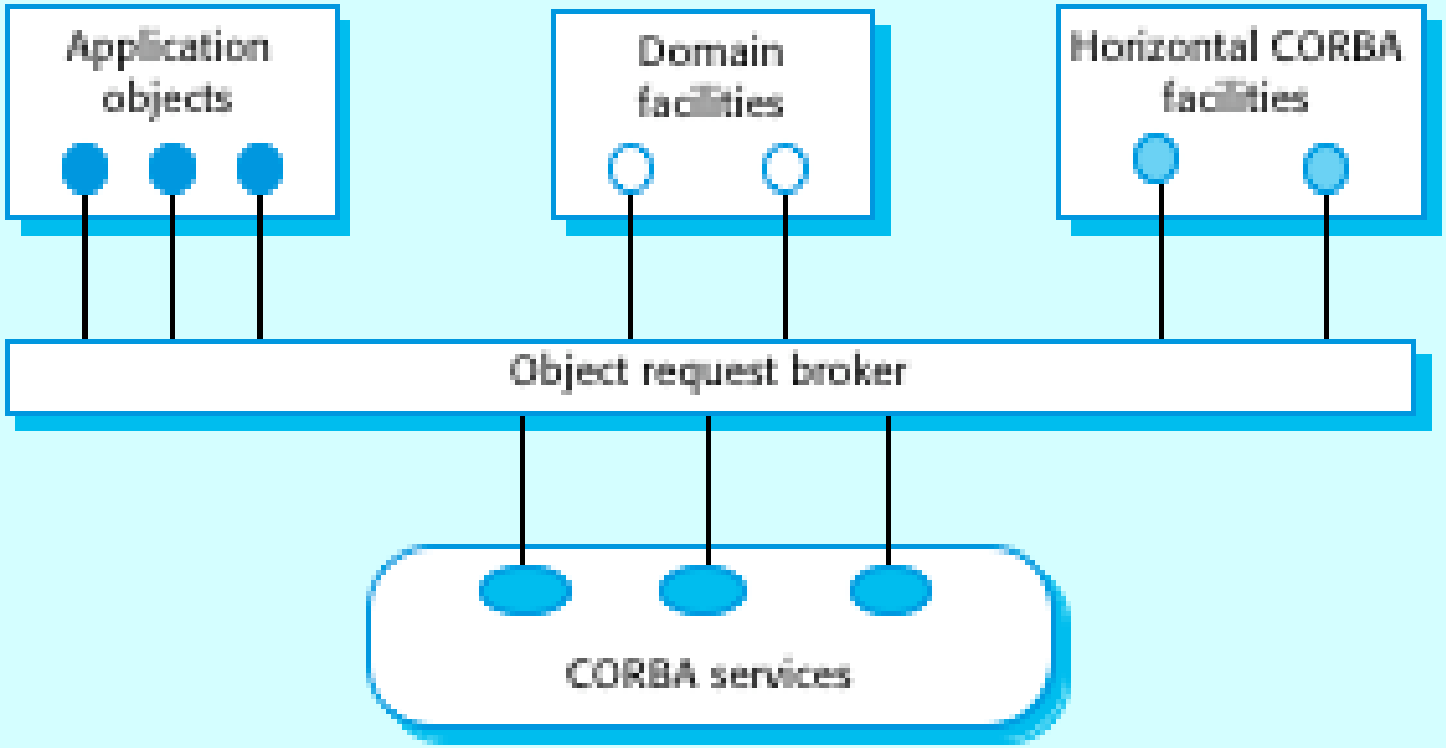
Data mining system

- 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.

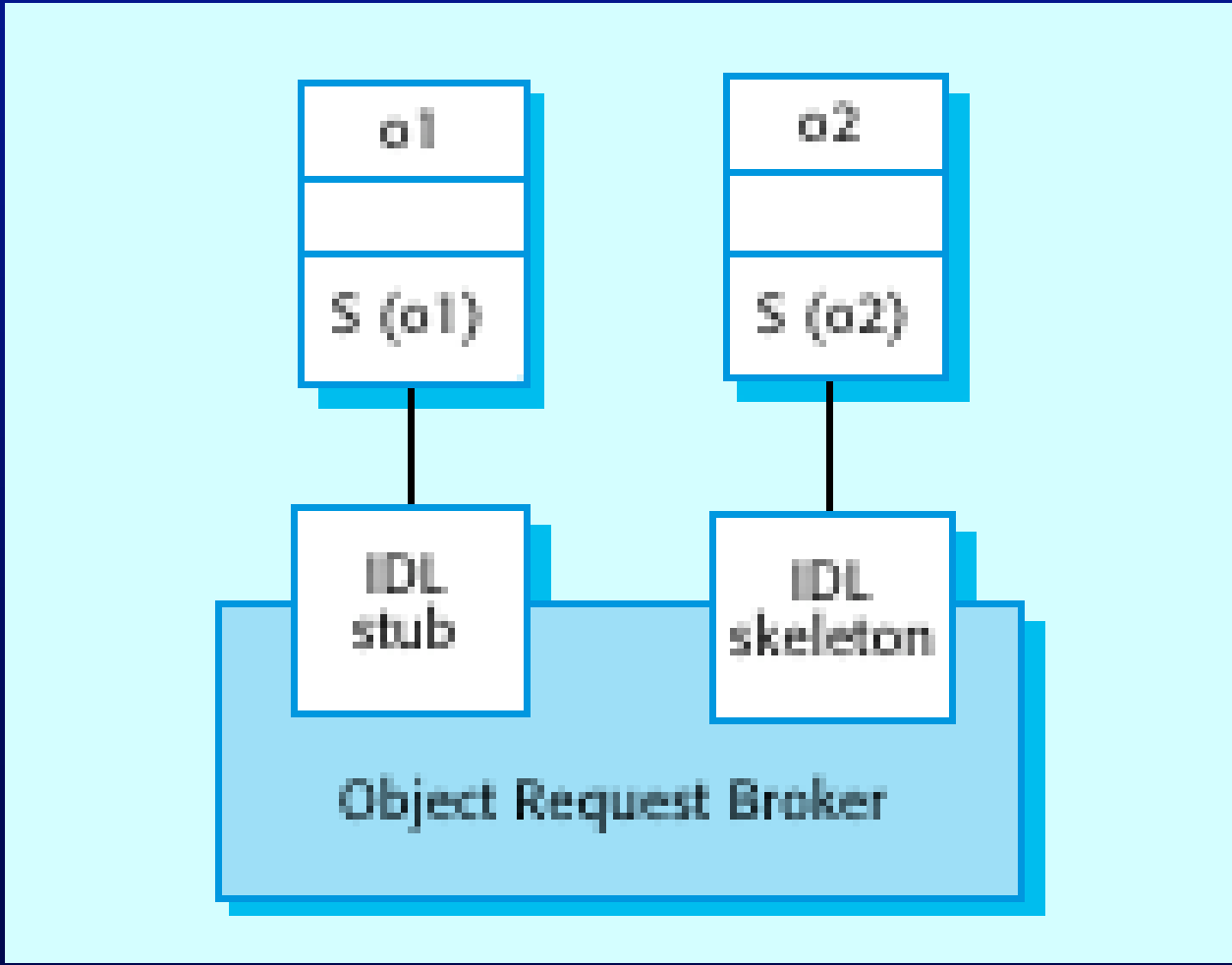
CORBA objects

- 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.

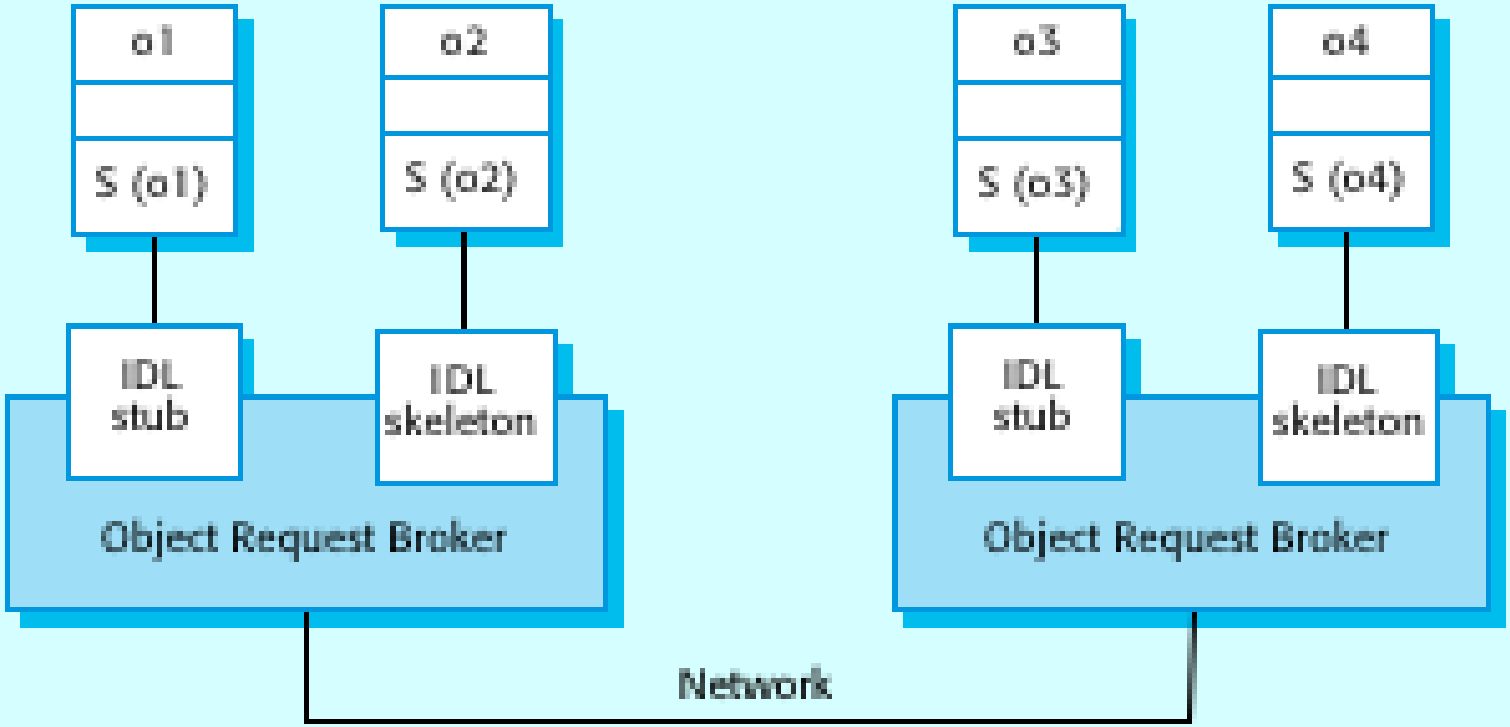
ORB-based object communications



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.

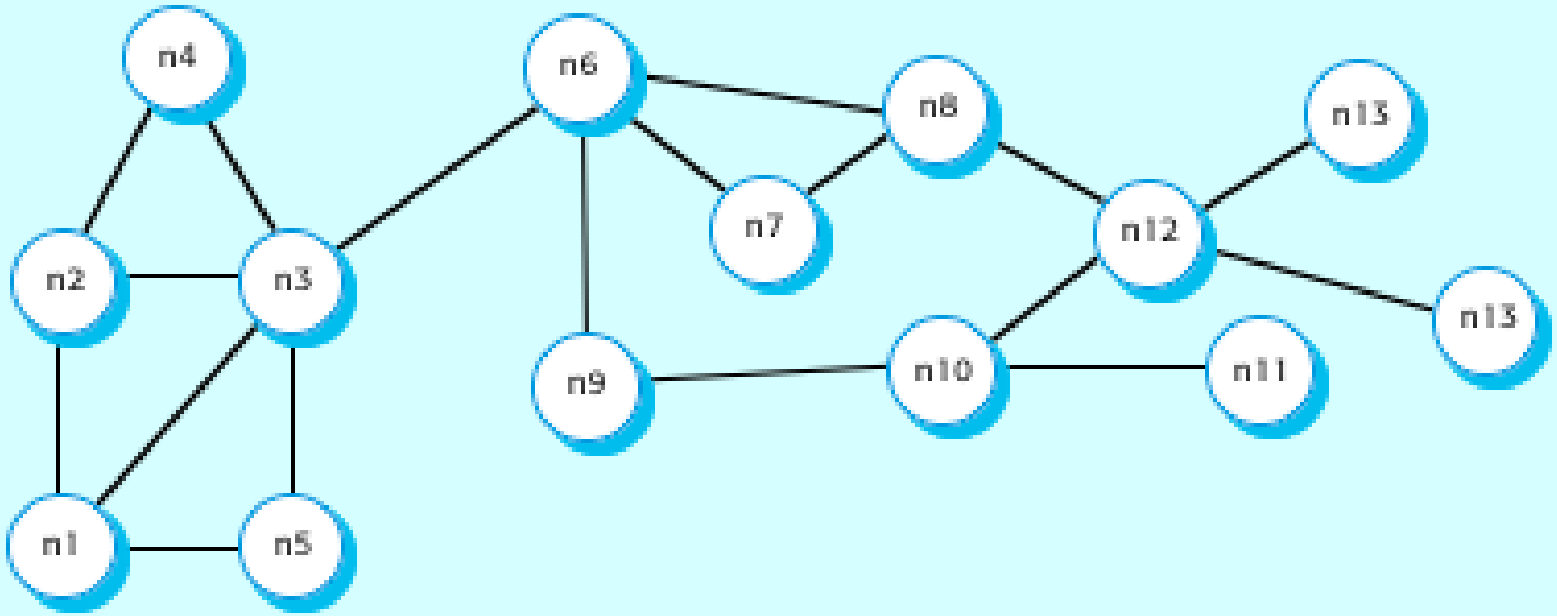
Peer-to-peer architectures

- 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.

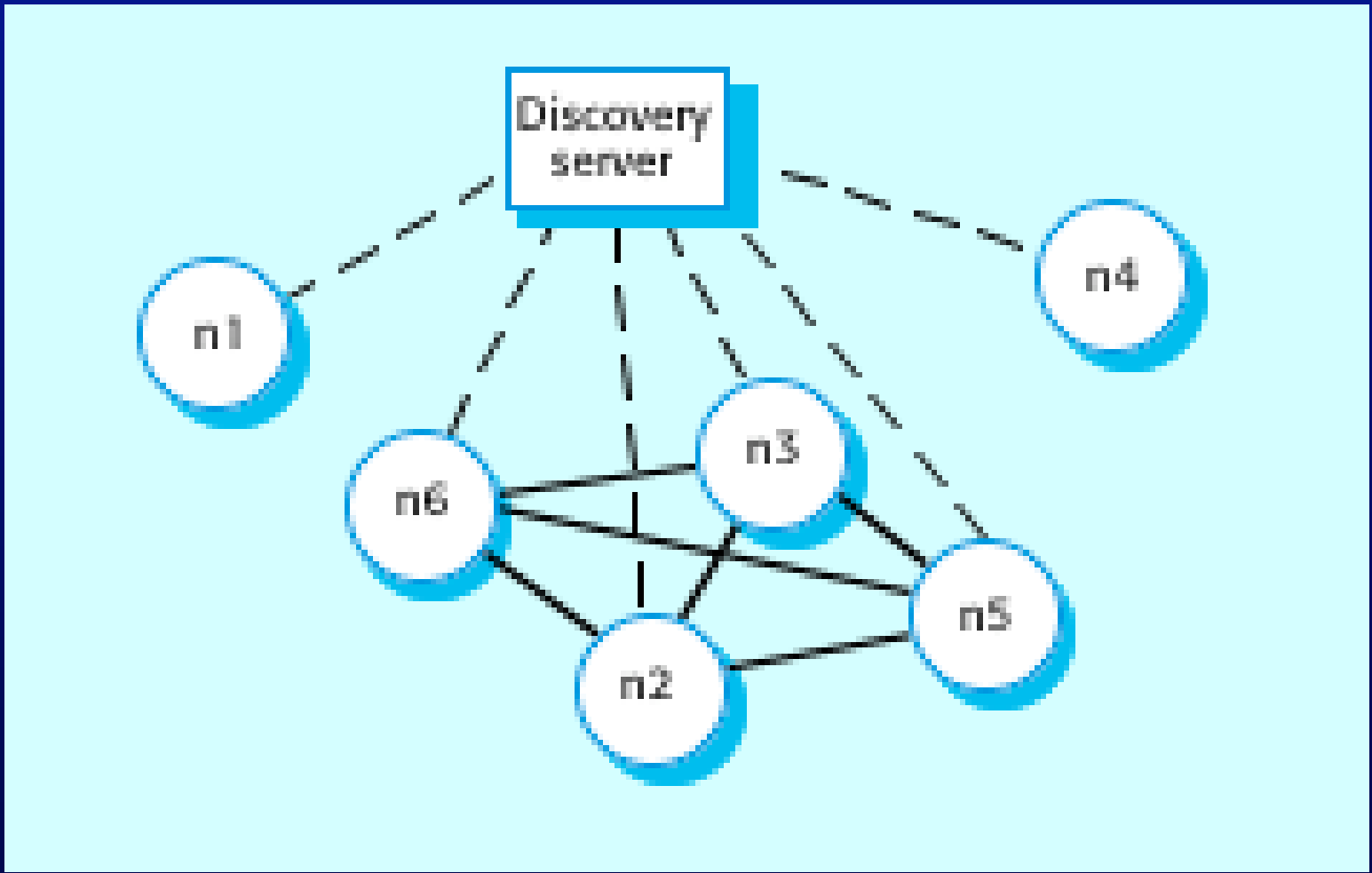
P2p architectural models

- 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



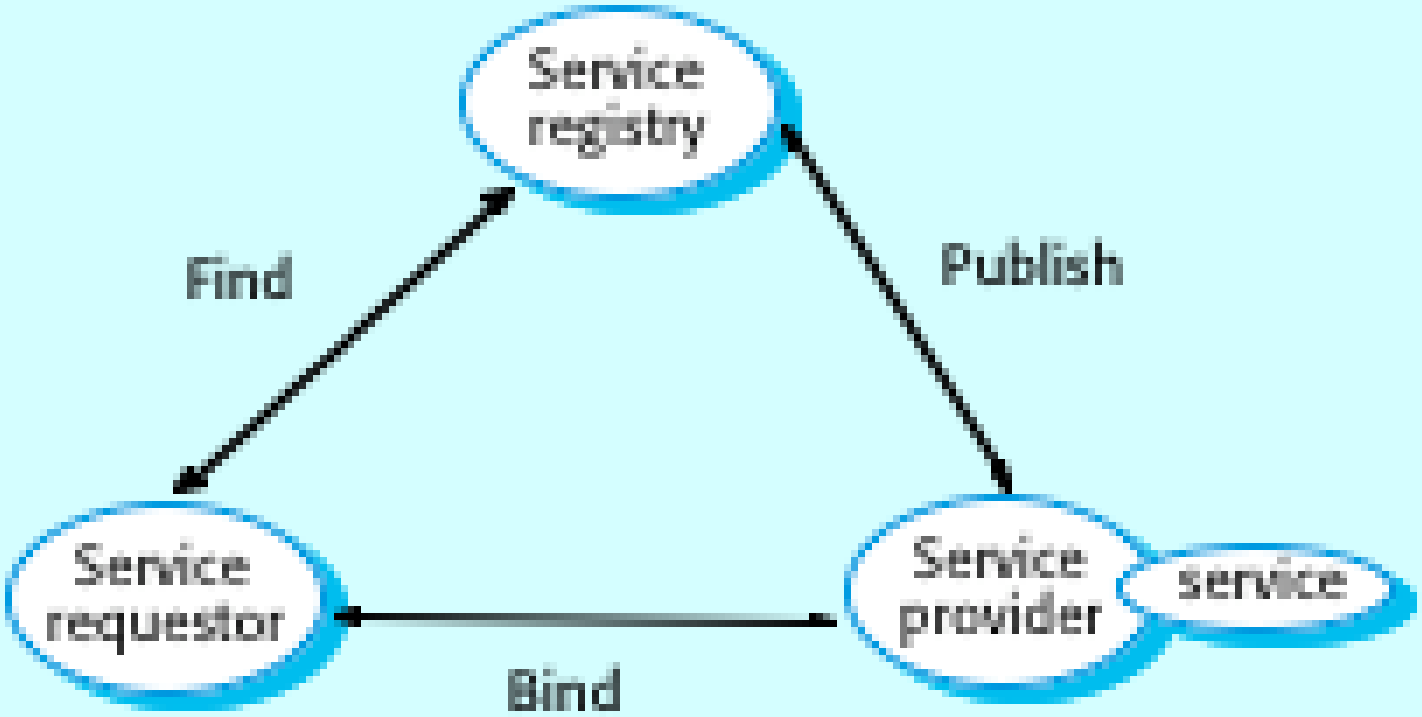
Service-oriented architectures

- 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.

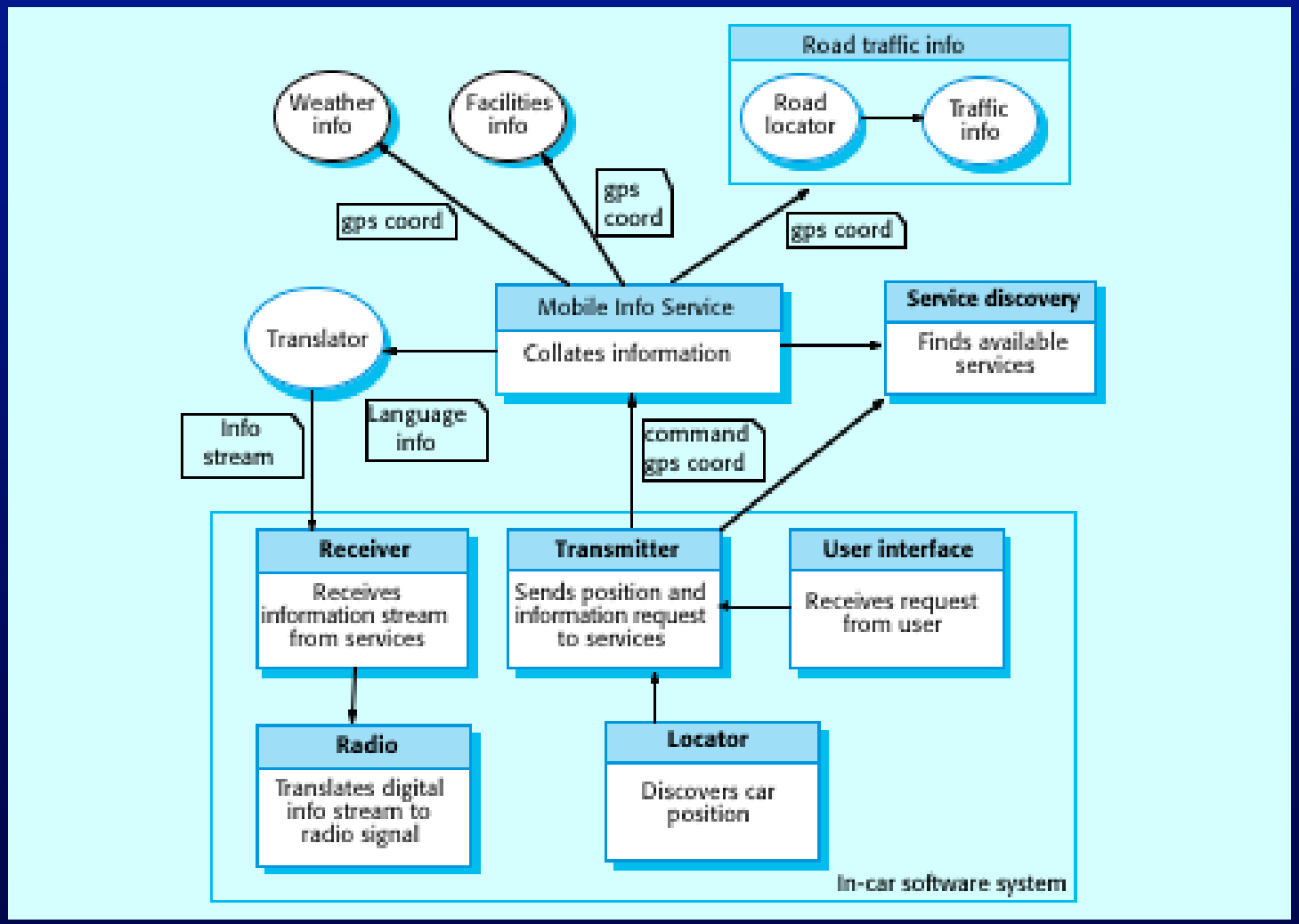
Services standards

- 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



Key points

- Distributed systems support resource sharing, openness, concurrency, scalability, fault tolerance and transparency.
- Client-server architectures involve services being delivered by servers to programs operating on clients.
- User interface software always runs on the client and data management on the server. Application functionality may be on the client or the server.
- In a distributed object architecture, there is no distinction between clients and servers.

Key points

- Distributed object systems require middleware to handle object communications and to add and remove system objects.
- The CORBA standards are a set of middleware standards that support distributed object architectures.
- Peer to peer architectures are decentralised architectures where there is no distinction between clients and servers.
- Service-oriented systems are created by linking software services provided by different service suppliers.