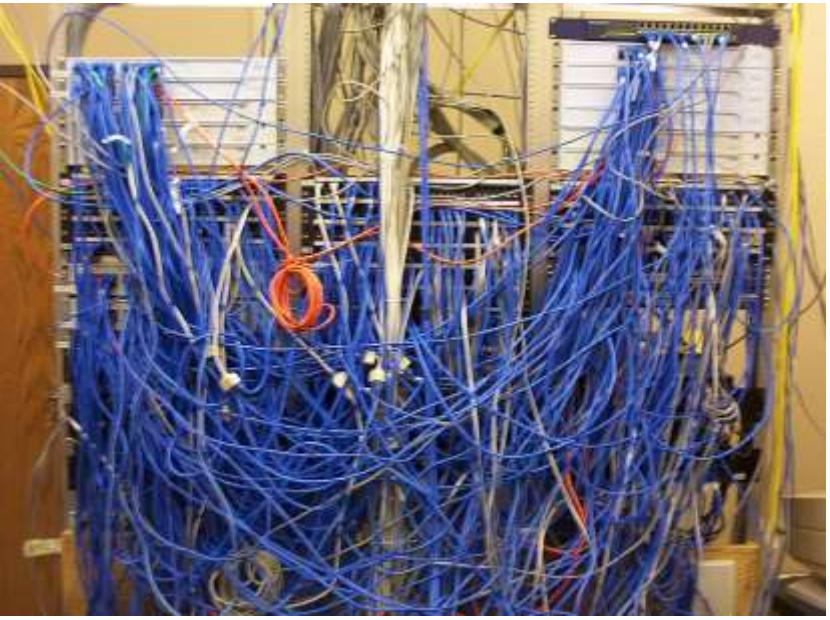
Architectural Patterns

Architectural Design

Topics covered

- Architectural design decisions
- Architectural views
- **Architectural patterns**


GLENNZ



A side note …

- **„parametric design" -** practical application of modern information technology and information science to architectural design

Examples of application types

- 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.

Architectural concepts

- **System**
  - project we are working on (context) - application
  - e.g. a house
- **Subsystem**
  - a part of the system that is not so dependent that it can be a system in its context
  - e.g. kitchen, room
- **Component**
  - A part of the system that has no meaning beyond its context

Architectural concepts - component

- A **component** is also:
  - a software unit replaced and updated independently of the system, an encapsulating set of related functions (and / or data):
    - library - a component connected to the program and called by means of direct calls,
    - service - an out-of-process component with which communication requires a protocol such as Internet services or other RPC form.
  - In this definition, the component can be a **subsystem implementation**.

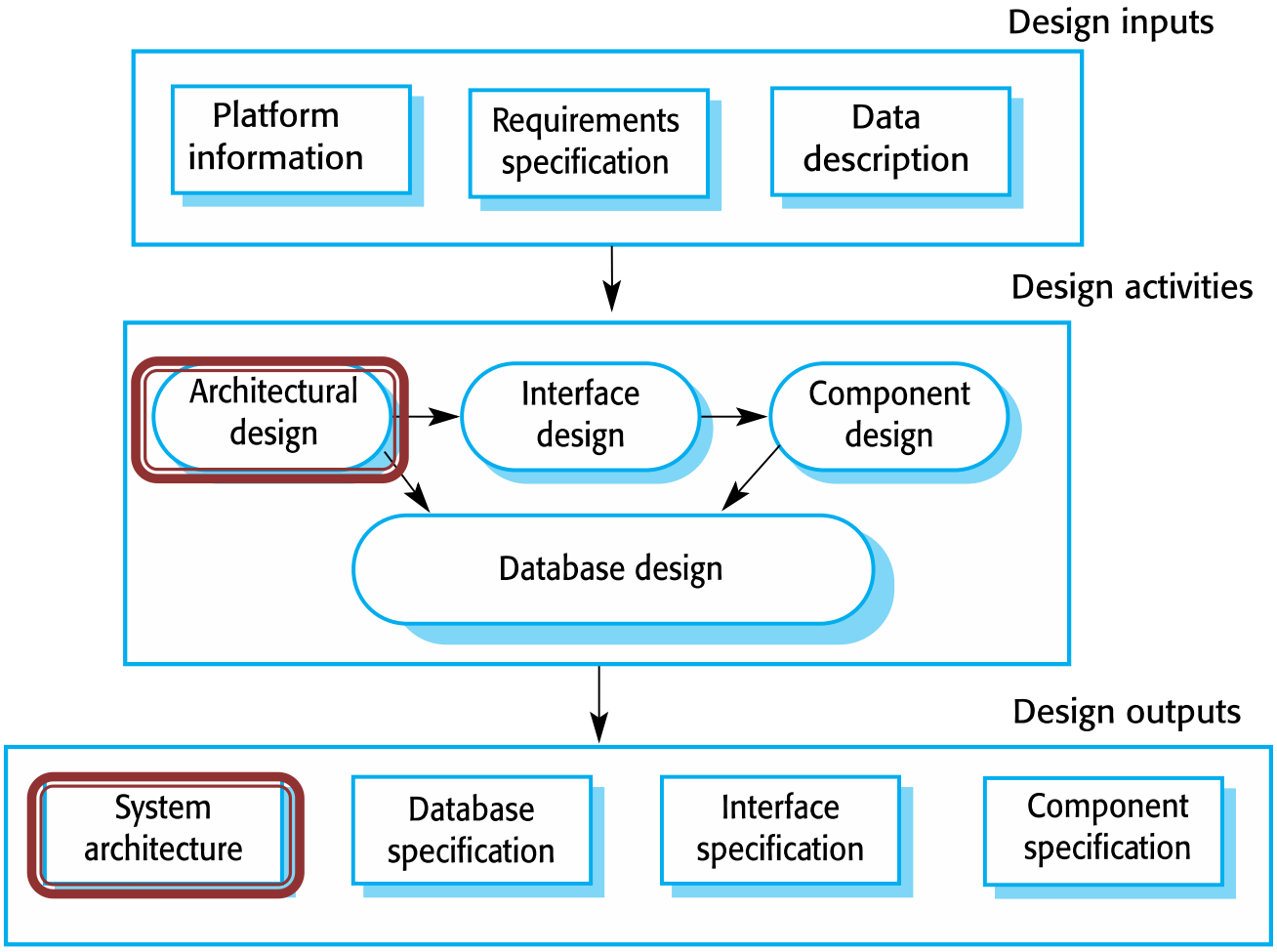
Software architecture

- 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.

Architectural design

- An early stage of the system design process.
- Represents the link between specification and design processes.
- Often carried out in parallel with some specification activities.
- It involves identifying major system components and their communications.

A general model of the design process



Architectural abstraction

- Architecture in the small is concerned with the architecture of individual programs. At this level, we are concerned with the way that an individual program is decomposed into components.
- Architecture in the large is concerned with the architecture of complex enterprise systems that include other systems, programs, and program components. These enterprise systems are distributed over different computers, which may be owned and managed by different companies.

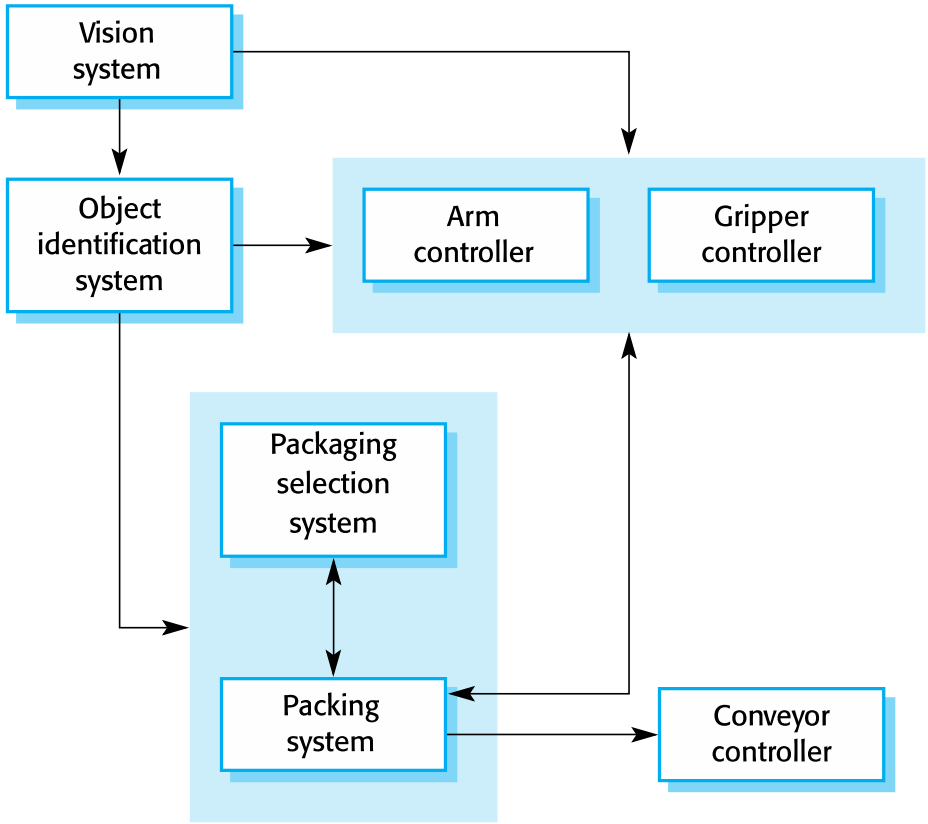
Advantages of explicit architecture

- 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
  - Product-line architectures may be developed.

Architectural representations

- Simple, informal block diagrams showing entities and relationships are the most frequently used method for documenting software architectures.
- But these have been criticized because they lack semantics, do not show the types of relationships between entities nor the visible properties of entities in the architecture.
- Depends on the use of architectural models. The requirements for model semantics depends on how the models are used.

The architecture of a packing robot control system



Box and line diagrams

- Very abstract - they do not show the nature of component relationships nor the externally visible properties of the sub-systems.
- However, useful for communication with stakeholders and for project planning.

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 and these decisions affect the non-functional characteristics of the system.

Architectural design decisions

- Is there a generic application architecture that can be used?
- How will the system be distributed?
- What architectural styles are appropriate?
- What approach will be used to structure the system?
- How will the system be decomposed into modules?
- What control strategy should be used?
- How will the architectural design be evaluated?
- How should the architecture be documented?

Architecture reuse

- Systems in the same domain often have similar architectures that reflect domain concepts.
- Application product lines are built around a core architecture with variants that satisfy particular customer requirements.
- The architecture of a system may be designed around one of more architectural patterns or 'styles'.
  - These capture the essence of an architecture and can be instantiated in different ways.

Architecture and system characteristics

1. Performance
   - Localise critical operations and minimise communications. Use large rather than fine-grain components.
2. Security
   - Use a layered architecture with critical assets in the inner layers.
3. Safety
   - Localise safety-critical features in a small number of sub-systems.
4. Availability
   - Include redundant components and mechanisms for fault tolerance.
5. Maintainability
   - Use fine-grain, replaceable components.

Architectural views

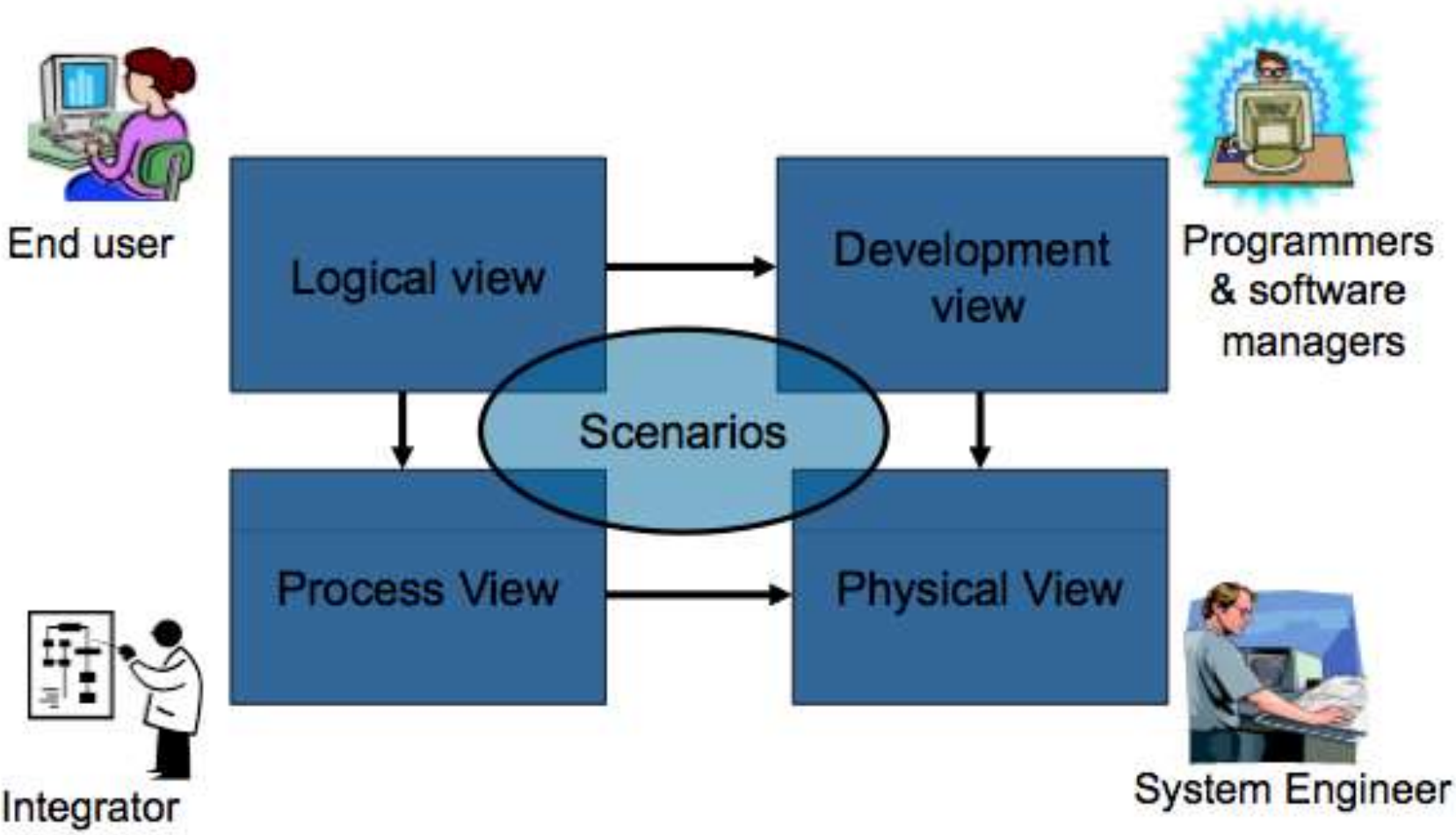
- What views or perspectives are useful when designing and documenting a system's architecture?
- What notations should be used for describing architectural models?
- Each architectural model only shows one view or perspective of the system.
  - It might show how a system is decomposed into modules, how the run-time processes interact or the different ways in which system components are distributed across a network. For both design and documentation, you usually need to present multiple views of the software architecture.

4 + 1 view model of software architecture

1. A logical view, which shows the key abstractions in the system as objects or object classes.
2. A process view, which shows how, at run-time, the system is composed of interacting processes.
3. A development view, which shows how the software is decomposed for development.
4. A physical view, which shows the system hardware and how software components are distributed across the processors in the system.

- Related using use cases or scenarios (+1)

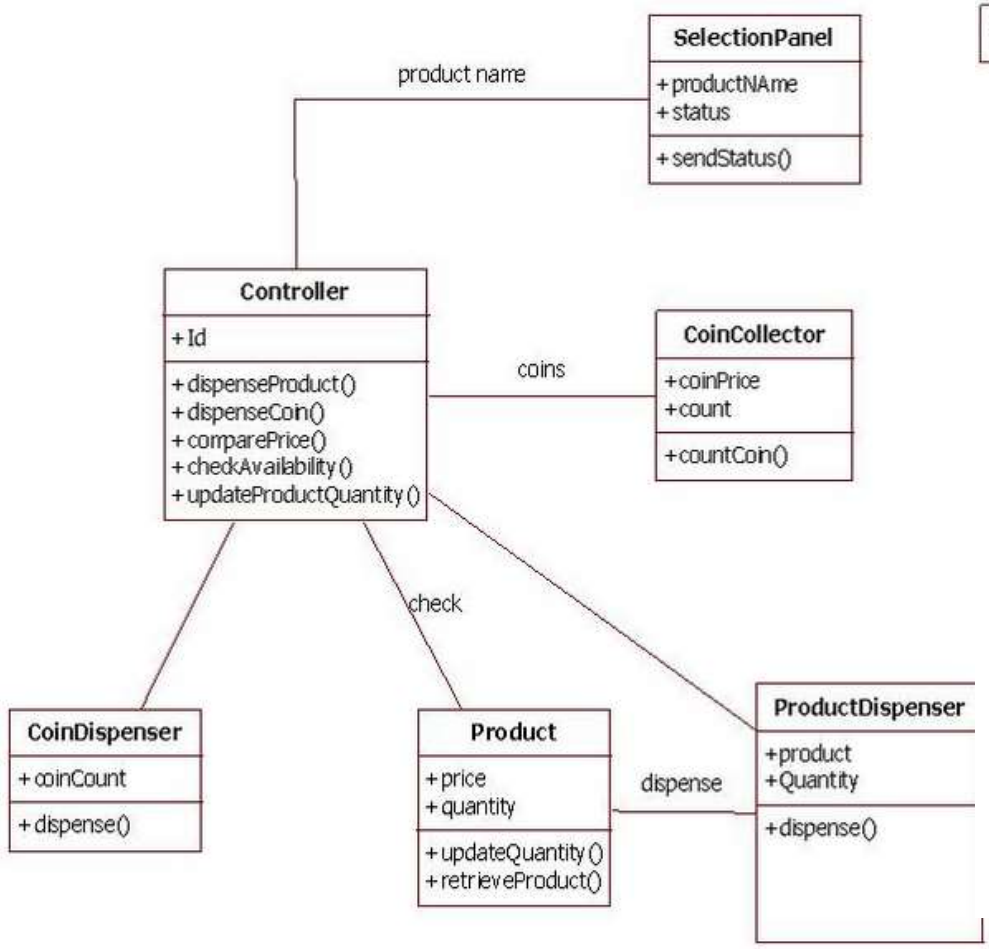
(4+1) view cycle

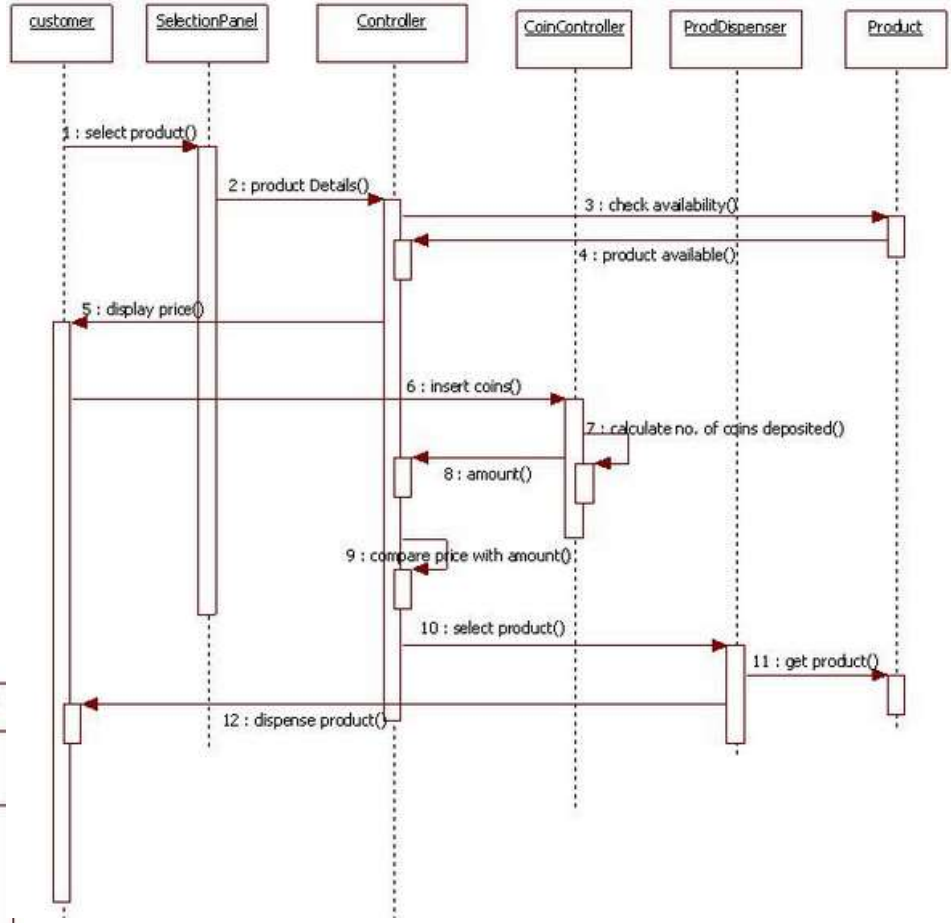


Example of vending machine from http://www.programsformca.com/

Logical view

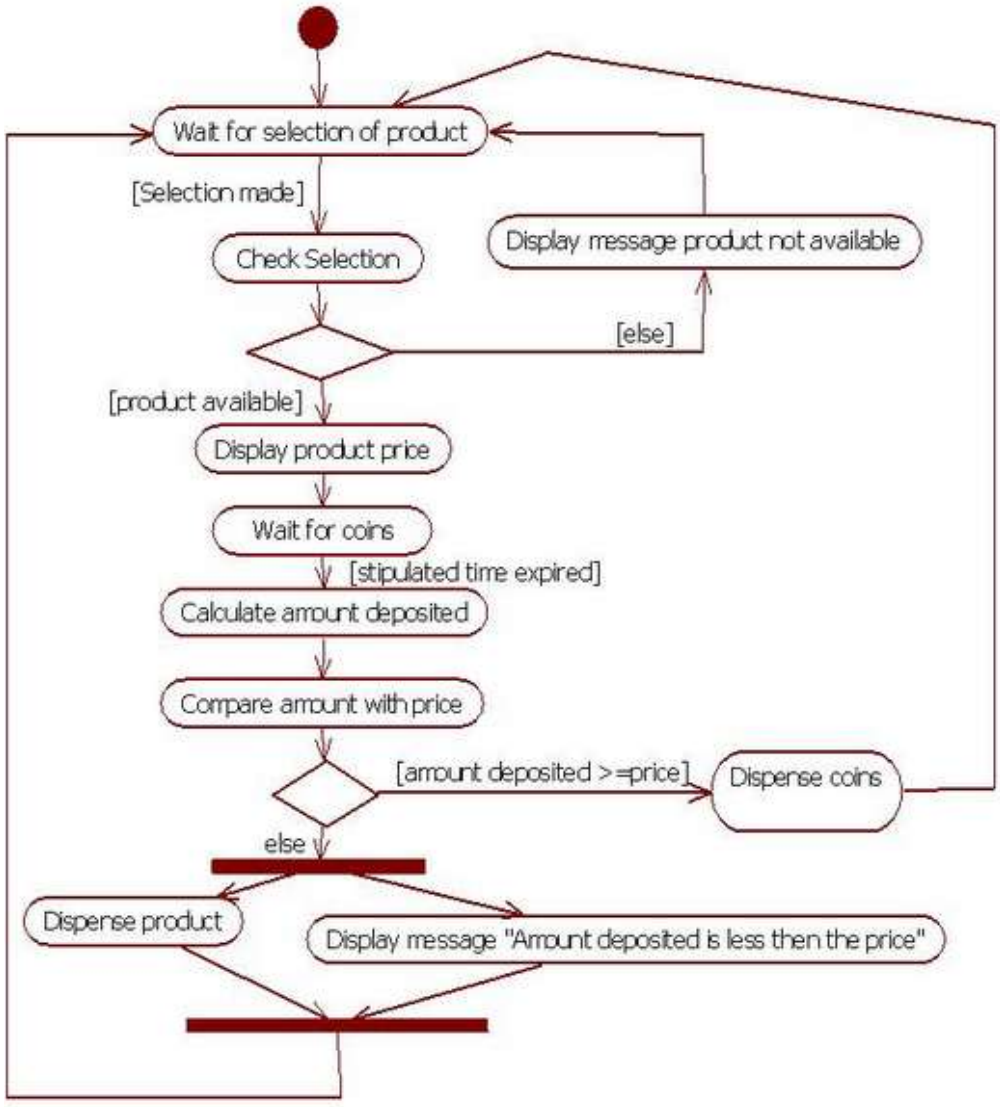
- UML diagram: class diag., sequence diag.





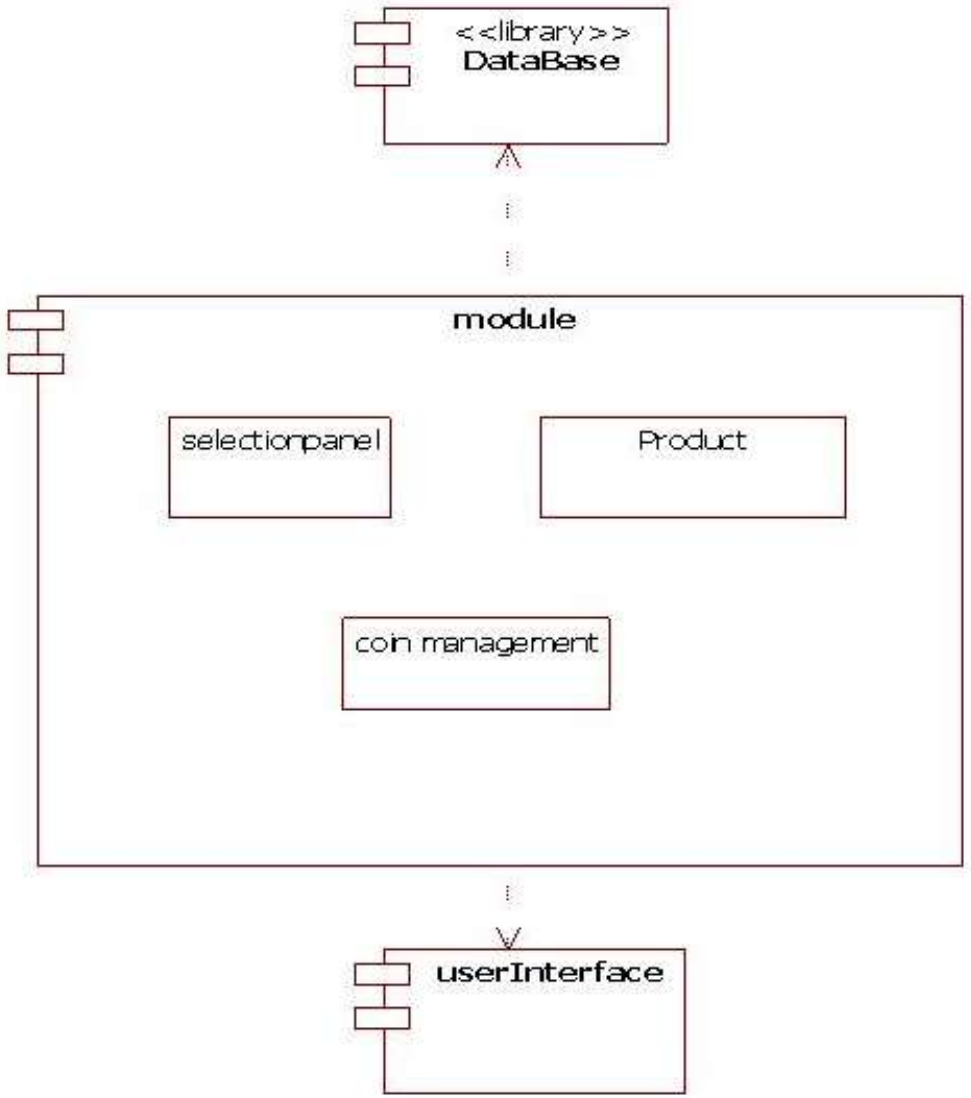
Process view

- UML diagram: activity diagram

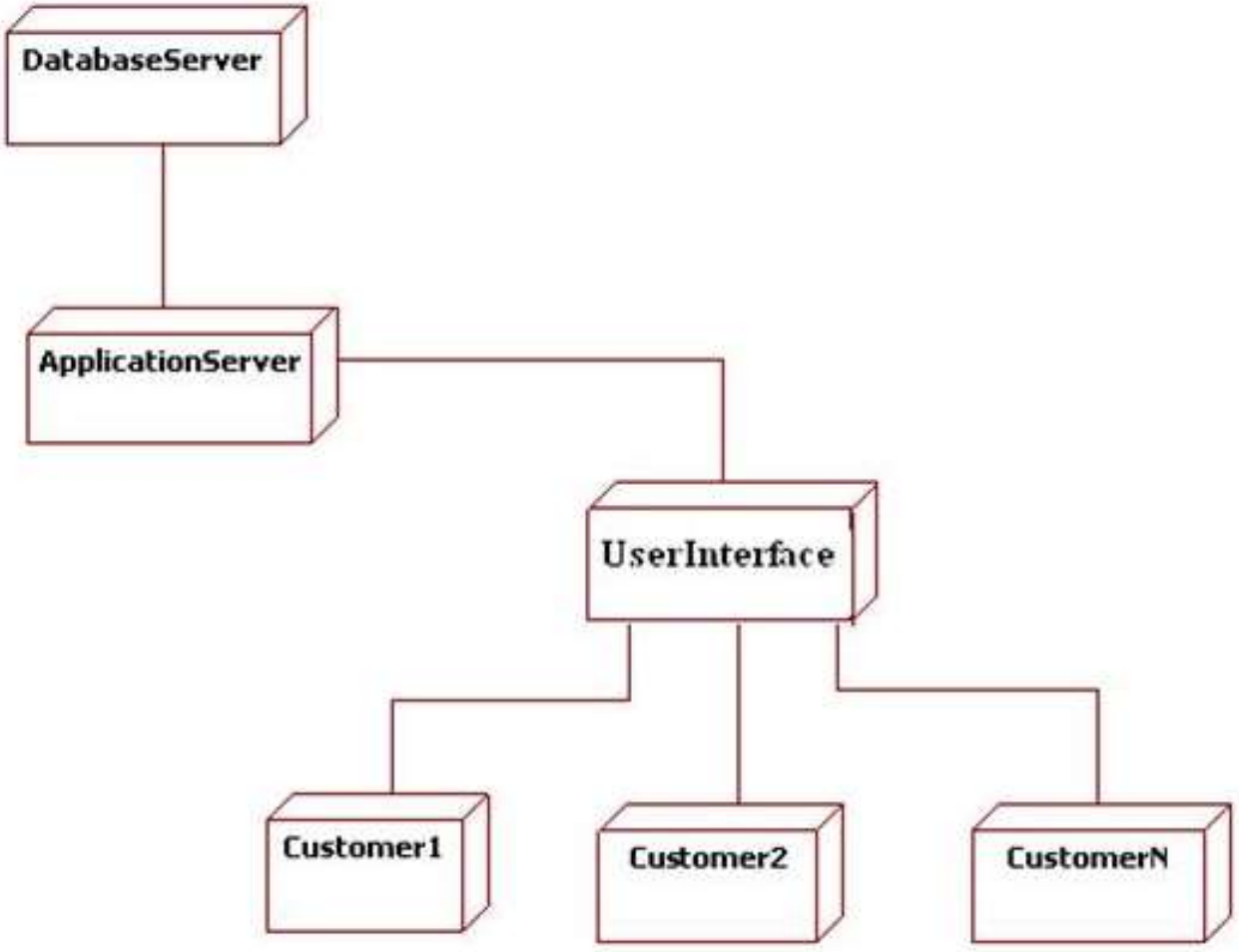


Development view

- UML diagram: Component diag.



Physical view



- UML: Deployment diagram

Architectural patterns

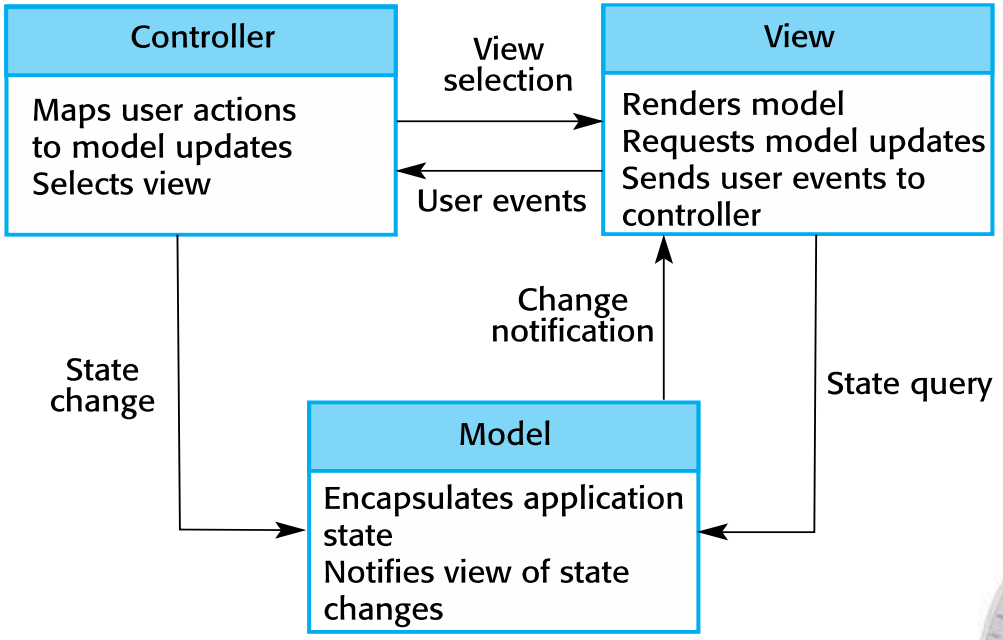
Architectural patterns

- Patterns are a means of representing, sharing and reusing knowledge.
- An architectural pattern is a stylized description of good design practice, which has been tried and tested in different environments.
- Patterns should include information about when they are and when the are not useful.
- Patterns may be represented using tabular and graphical descriptions.

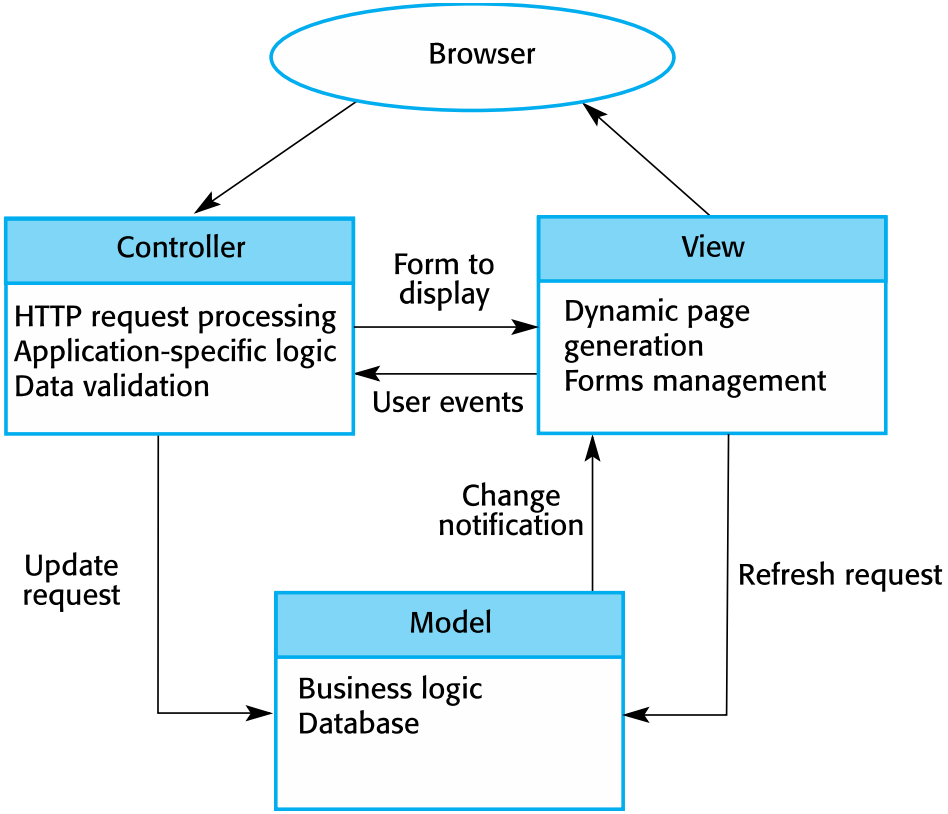
The Model-View-Controller (MVC) pattern

Name	MVC (Model-View-Controller)
Description	Separates presentation and interaction from the system data. The system is structured into three logical components that interact with each other. The Model component manages the system data and associated operations on that data. The View component defines and manages how the data is presented to the user. The Controller component manages user interaction (e.g., key presses, mouse clicks, etc.) and passes these interactions to the View and the Model. (See Next Slide).
Example	Another slide shows the architecture of a web-based application system organized using the MVC pattern.
When used	Used when there are multiple ways to view and interact with data. Also used when the future requirements for interaction and presentation of data are unknown.
Advantages	Allows the data to change independently of its representation and vice versa. Supports presentation of the same data in different ways with changes made in one representation shown in all of them.
Disadvantages	Can involve additional code and code complexity when the data model and interactions are simple.

The organization of the Model-View-Controller



Web application architecture using the MVC pattern



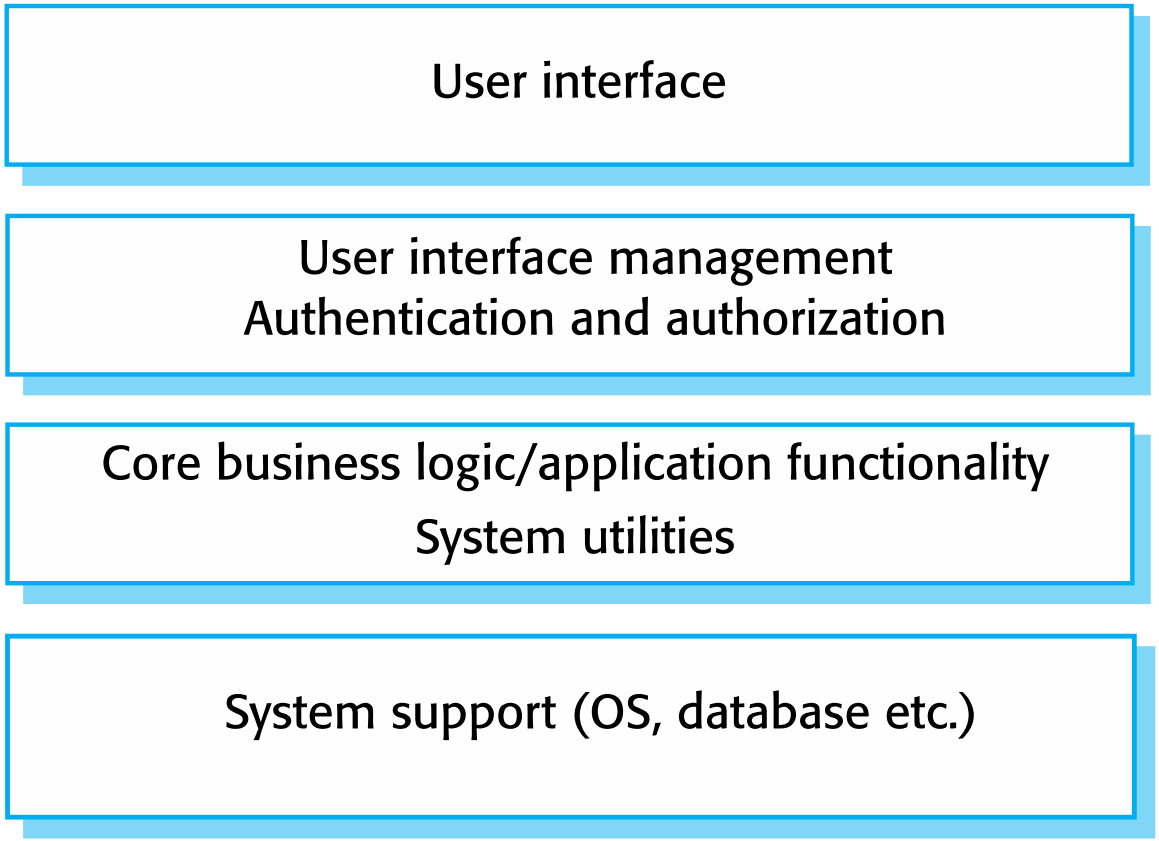
Layered architecture pattern

- Used to model the interfacing of sub-systems.
- Organises the system into a set of layers (or abstract machines) each of which provide a set of services.
- Supports the incremental development of sub-systems in different layers. When a layer interface changes, only the adjacent layer is affected.
- However, often artificial to structure systems in this way.

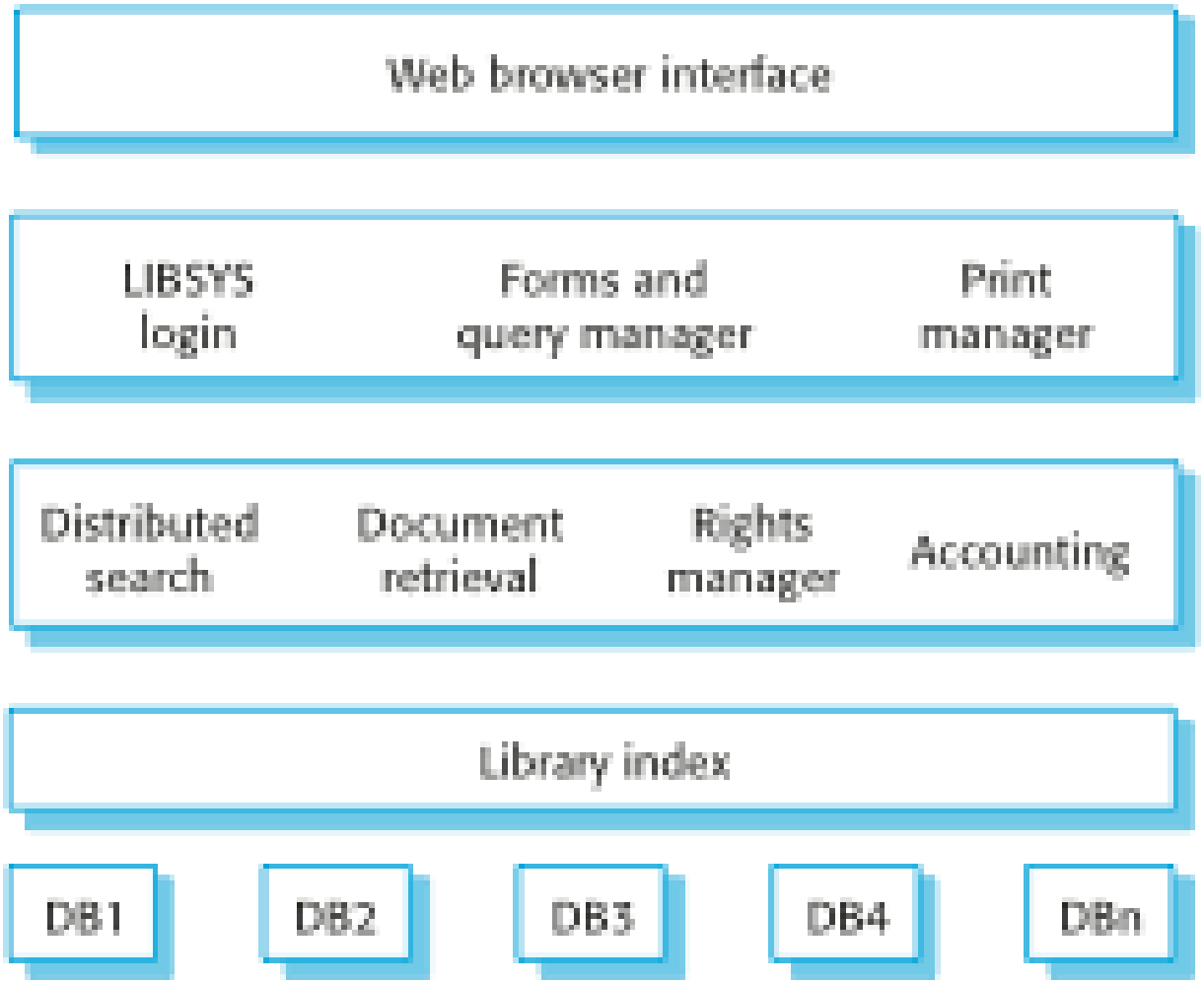
The Layered architecture pattern

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

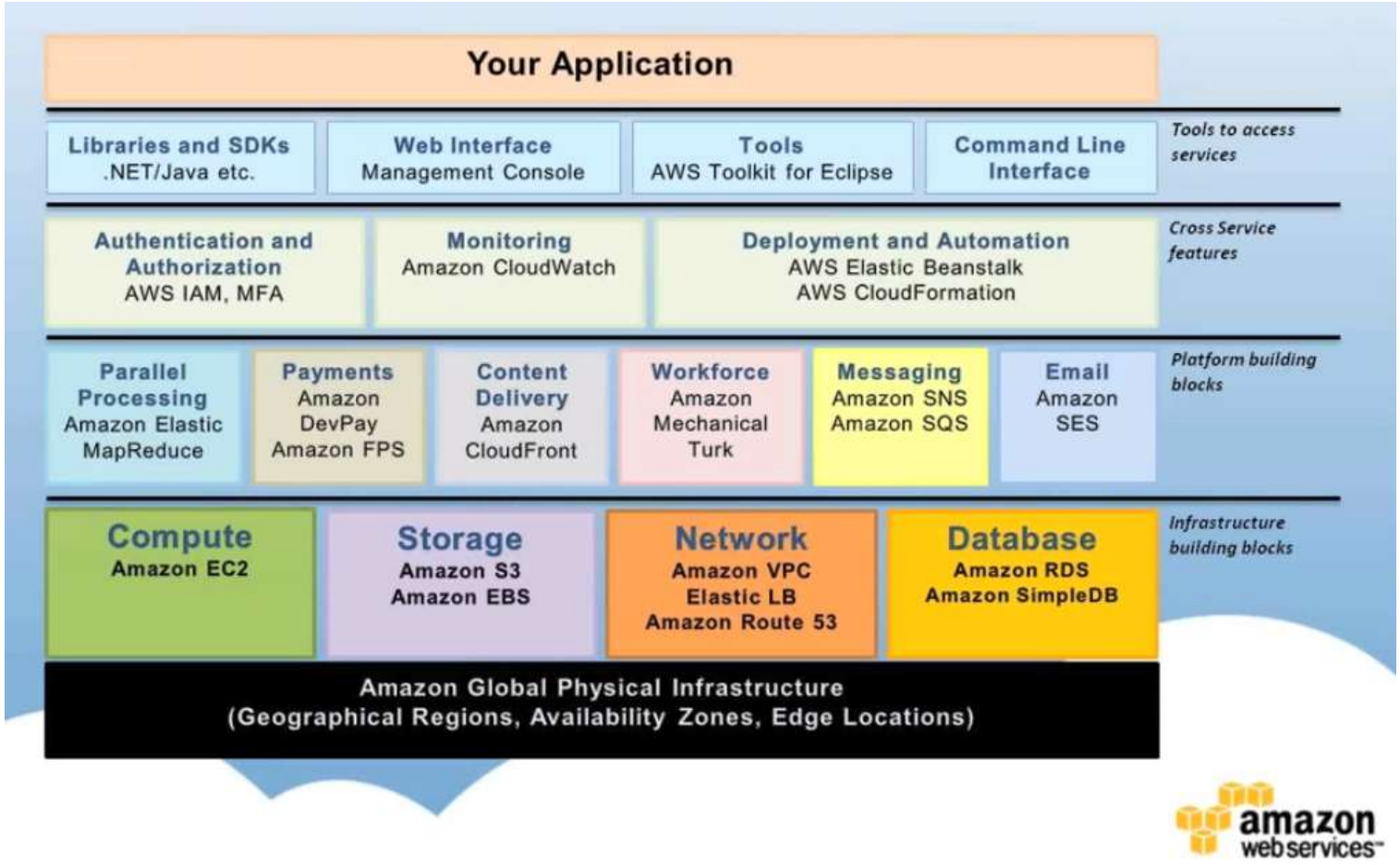
A generic layered architecture



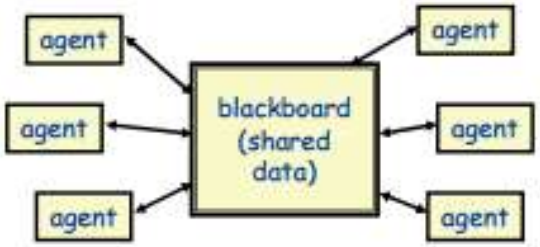
The architecture of the LIBSYS system



„Cloud layers"

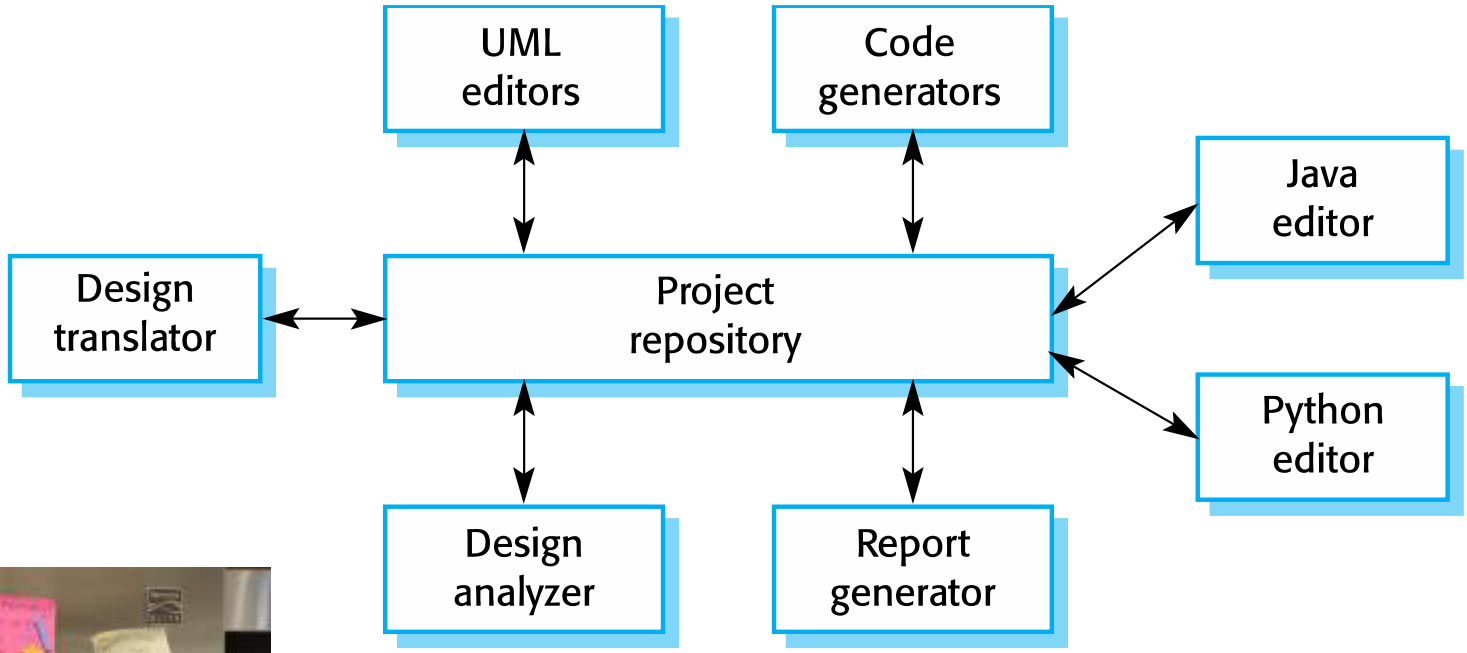


The Repository pattern (Blackboard)



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

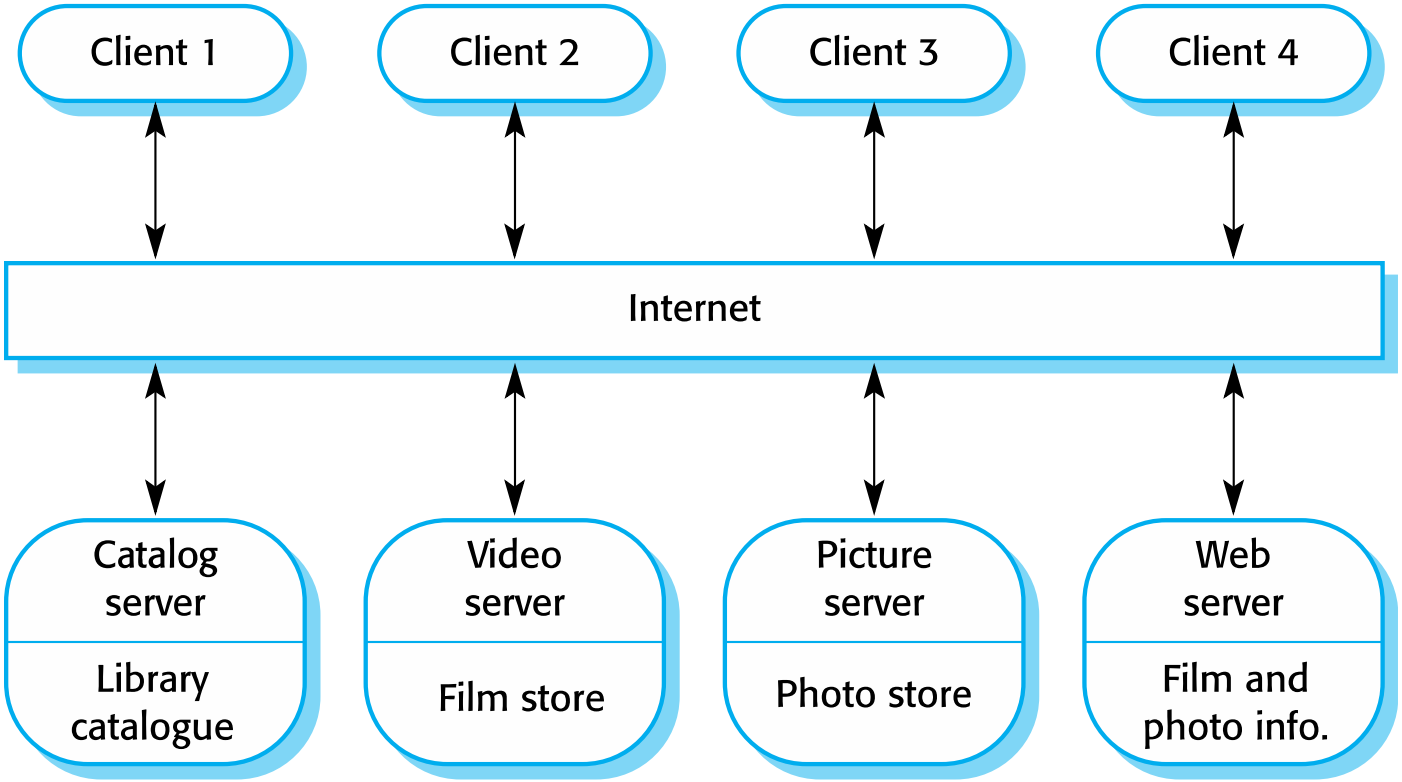
A repository architecture for an IDE



The Client–server pattern

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

A client–server architecture for a film library

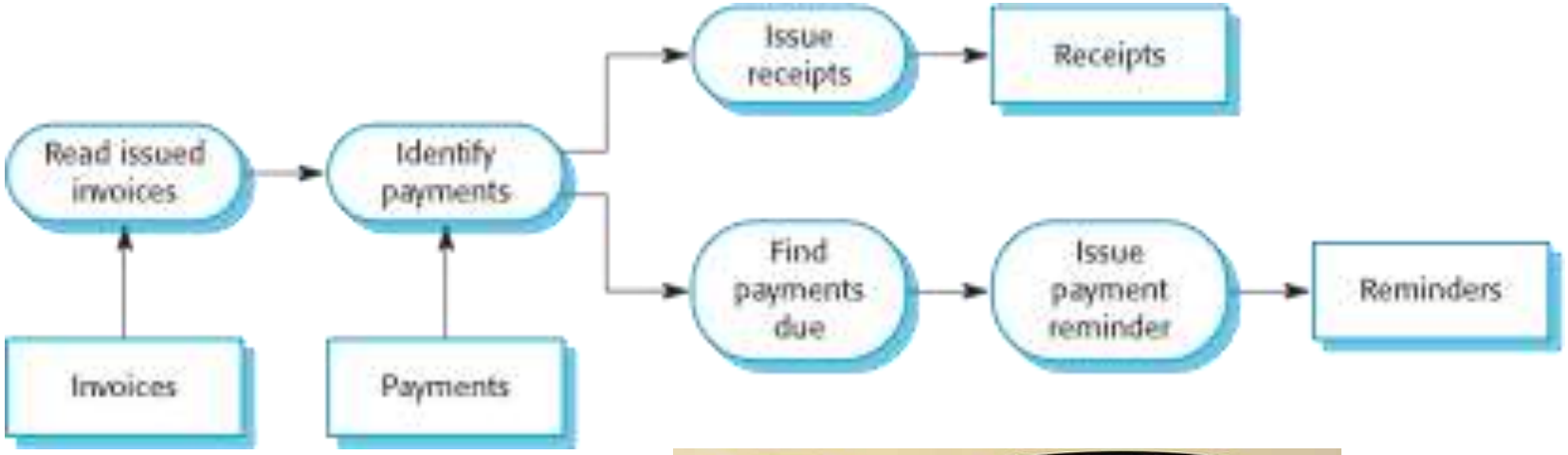


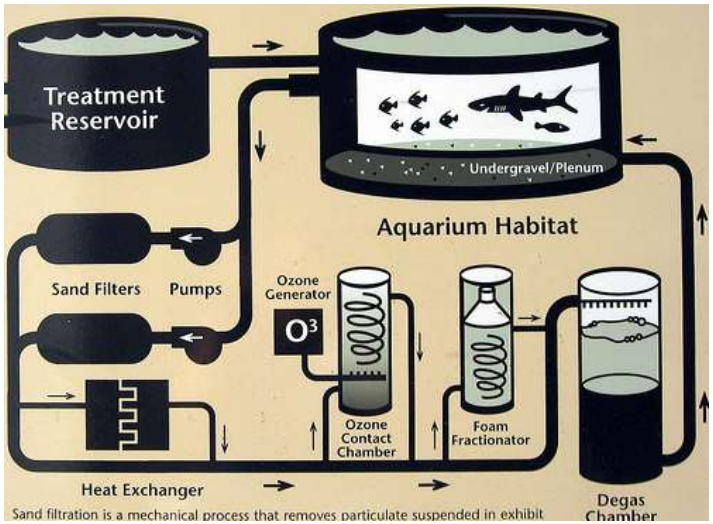


The pipe and filter pattern

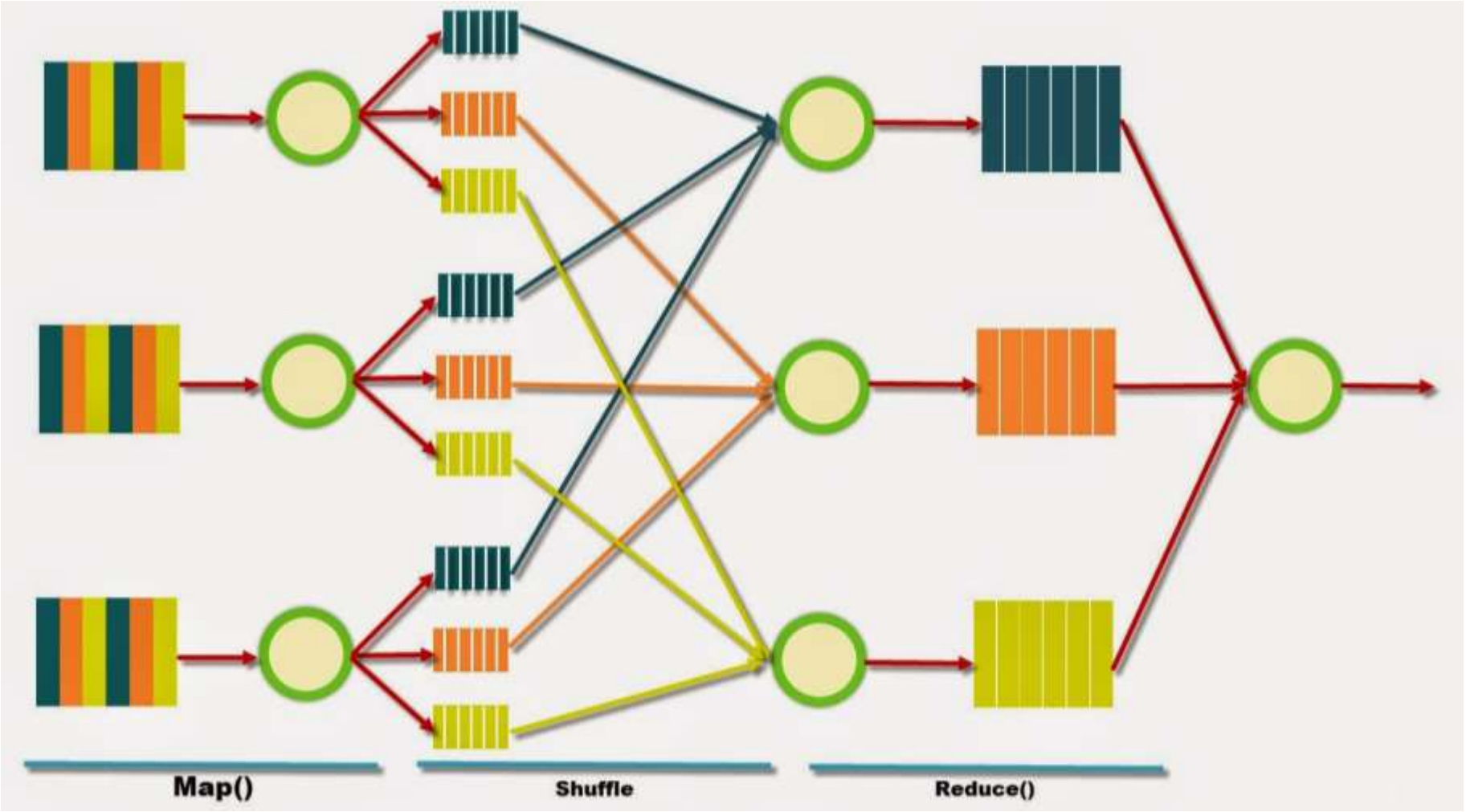
Name	Pipe and filter
Description	The processing of the data in a system is organized so that each processing component (filter) is discrete and carries out one type of data transformation. The data flows (as in a pipe) from one component to another for processing.
Example	Next slide shows an example of a pipe and filter system used for processing invoices.
When used	Commonly used in data processing applications (both batch- and transaction-based) where inputs are processed in separate stages to generate related outputs.
Advantages	Easy to understand and supports transformation reuse. Workflow style matches the structure of many business processes. Evolution by adding transformations is straightforward. Can be implemented as either a sequential or concurrent system.
Disadvantages	The format for data transfer has to be agreed upon between communicating transformations. Each transformation must parse its input and unparse its output to the agreed form. This increases system overhead and may mean that it is impossible to reuse functional transformations that use incompatible data structures.

An example of the pipe and filter architecture





Map-Reduce – distributed processing



Event architecture

- A event-response system model whose operation is based on reactions to time-indeterminate (asynchronous) events.
- The system consists of loosely related components (or services).
- Components can emit events (agents, issuers, publishers) and consume them (consumer, subscriber).

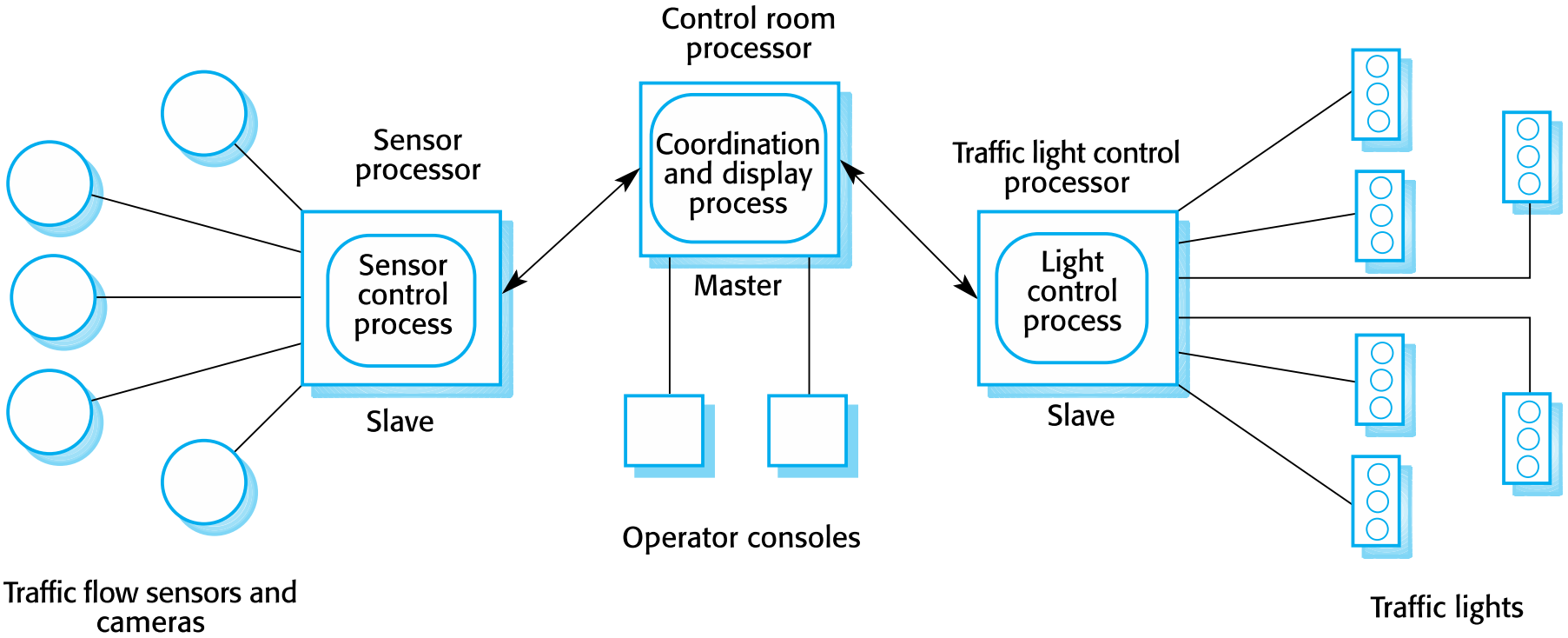
Architectural patterns – distributed systems

- The most popular architectural patterns of distributed applications include:
  - Master-slave architecture,
  - Two-tier client server architecture,
  - Multi-layered client-server architecture,
  - Architecture of distributed components,
  - Service-oriented architecture,
    - Architecture of microservices,
  - Peer-to-peer architecture.

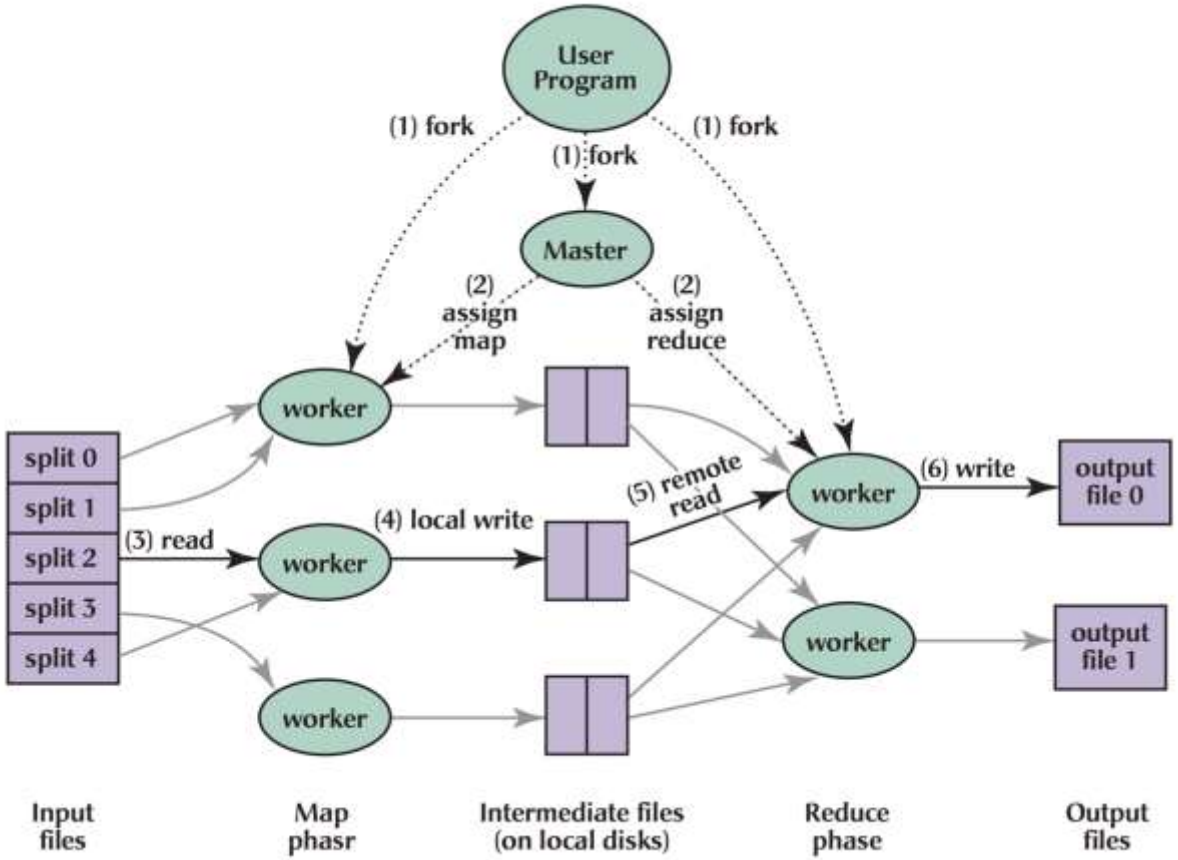
Master-slave architecture

- Applications:
  - Real-time systems, where separate processors are associated with collecting data from the environment, others for processing and yet others for signal management
- The 'master' process is usually responsible for calculating, coordinating, communicating and controlling slave processes.
- 'Slave' processes are dedicated to specific actions (eg collecting data from sensors).

A traffic management system with a master-slave architecture



MapReduce Architecture



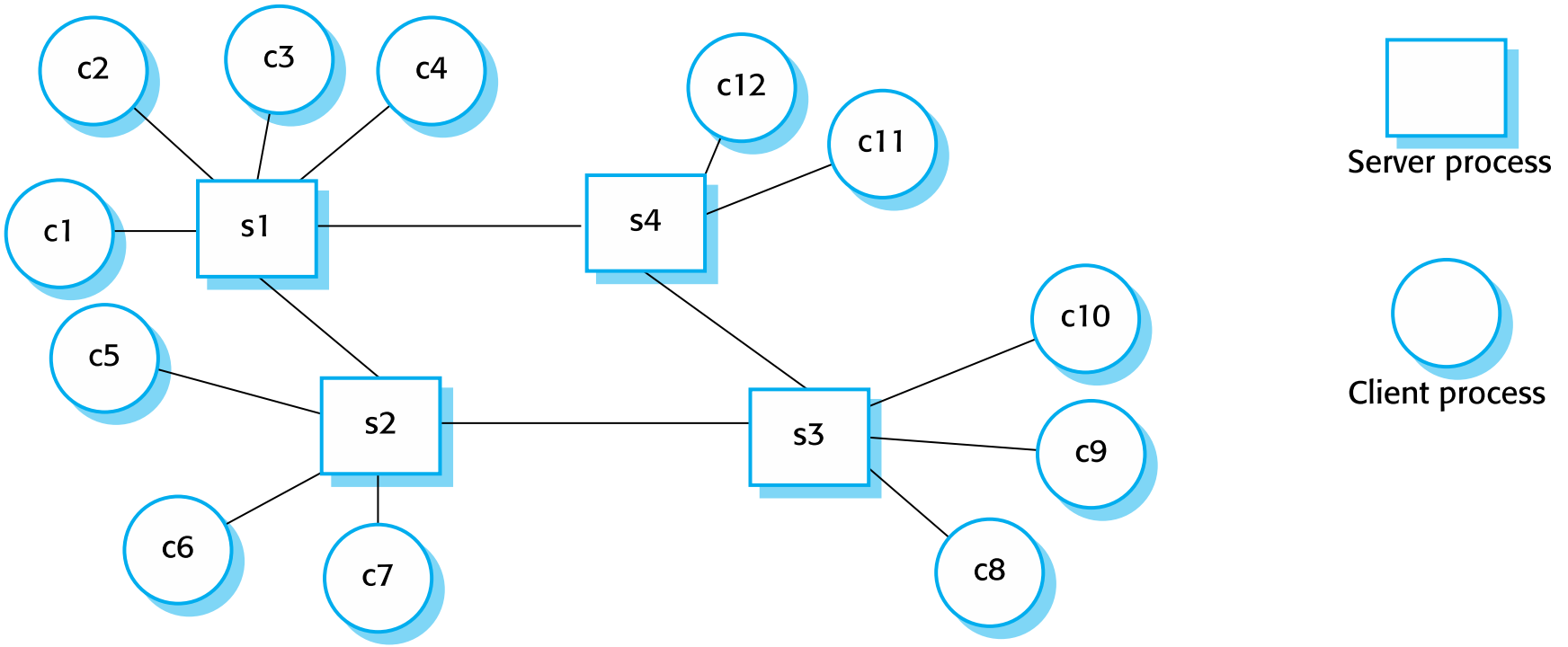
http://homeostasis.scs.carleton.ca/~soma/distos/fall2008/mapreduce2008-cacm.pdf

Tutaj skończyłem 15.V.2018

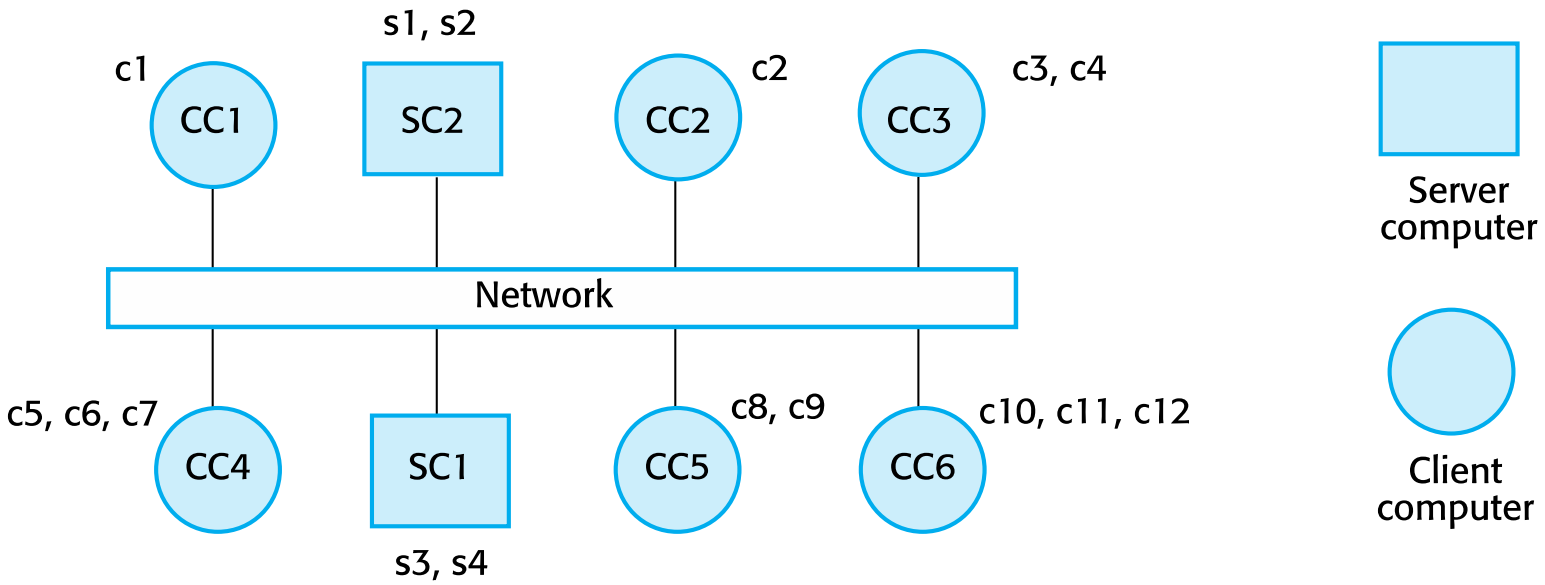
Client-server computing

- Distributed systems that are accessed over the Internet are normally organized as client-server systems.
- In a client-server system, the user interacts with a program running on their local computer (e.g. a web browser or mobile application). This interacts with another program running on a remote computer (e.g. a web server).
- The remote computer provides services, such as access to web pages, which are available to external clients.

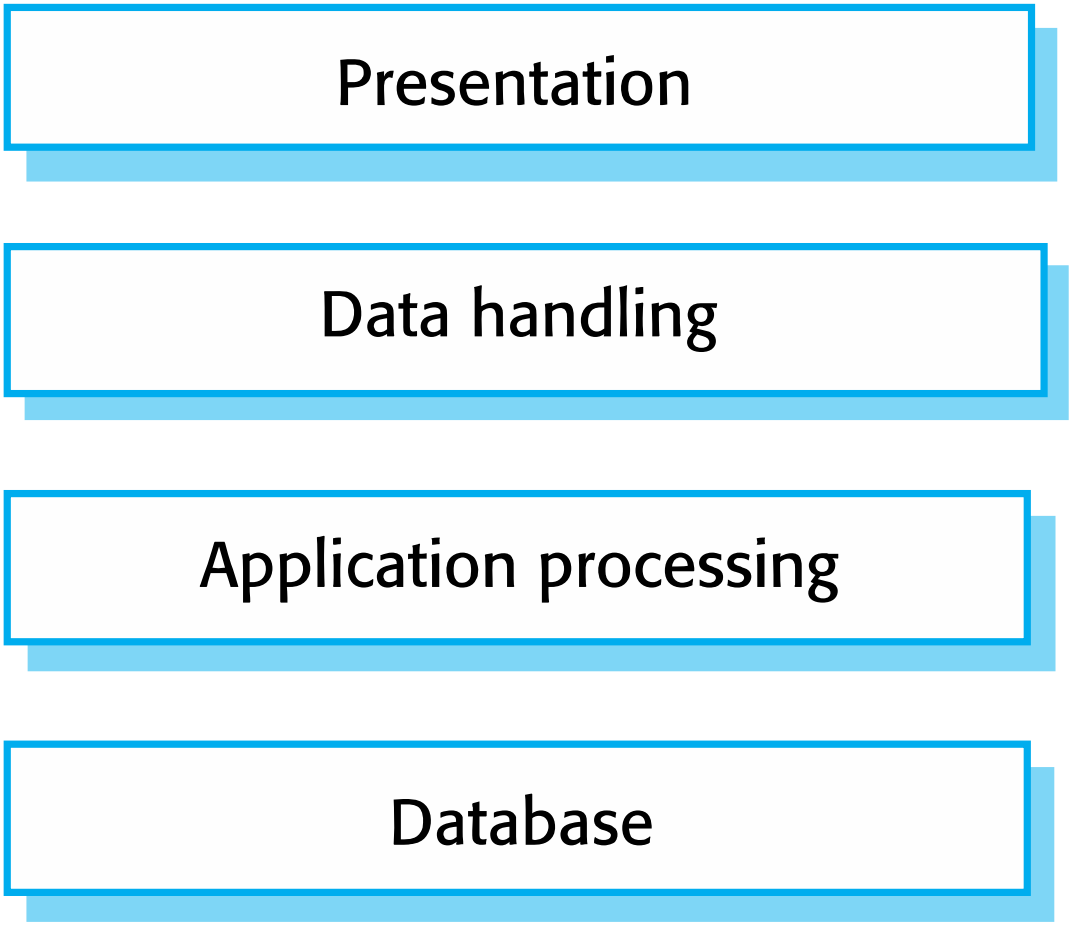
Client–server interaction



Mapping of clients and servers to networked computers



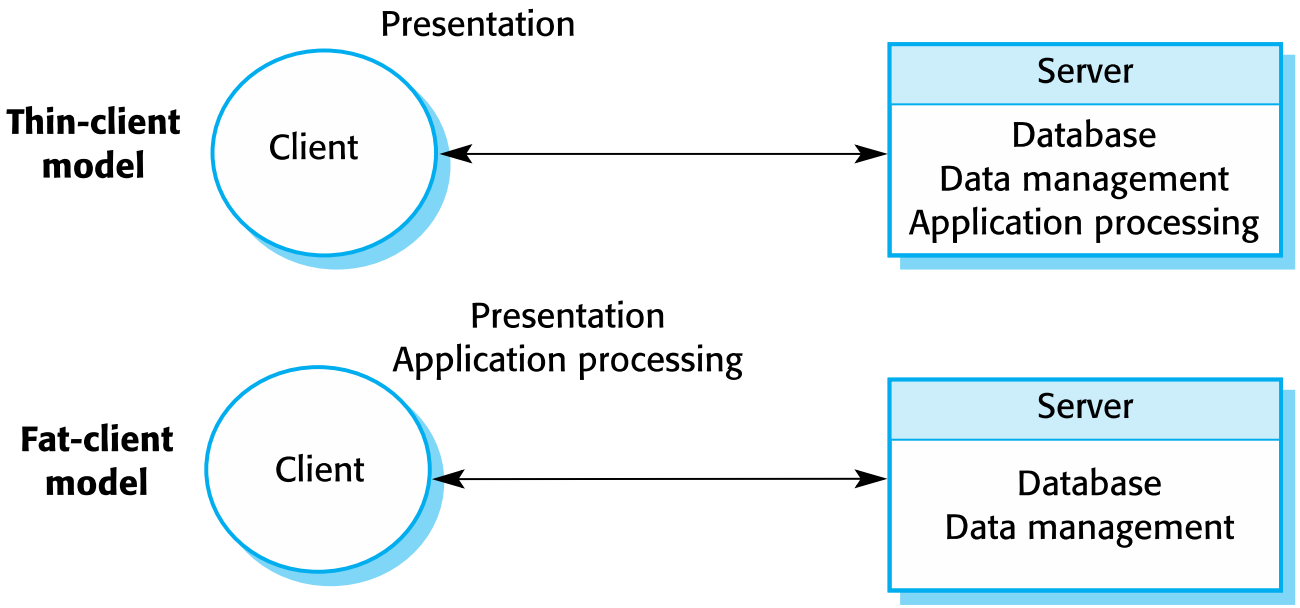
Layered architectural model for client–server applications



Two-tier client-server architectures

- In a two-tier client-server architecture, the system is implemented as a single logical server plus an indefinite number of clients that use that server.
  - Thin-client model, where the presentation layer is implemented on the client and all other layers (data management, application processing and database) are implemented on a server.
  - Fat-client model, where some or all of the application processing is carried out on the client. Data management and database functions are implemented on the server.
- Applications: Simple client-server systems and systems where the need to secure the system requires centralization (e.g., data)

Thin- and fat-client architectural models



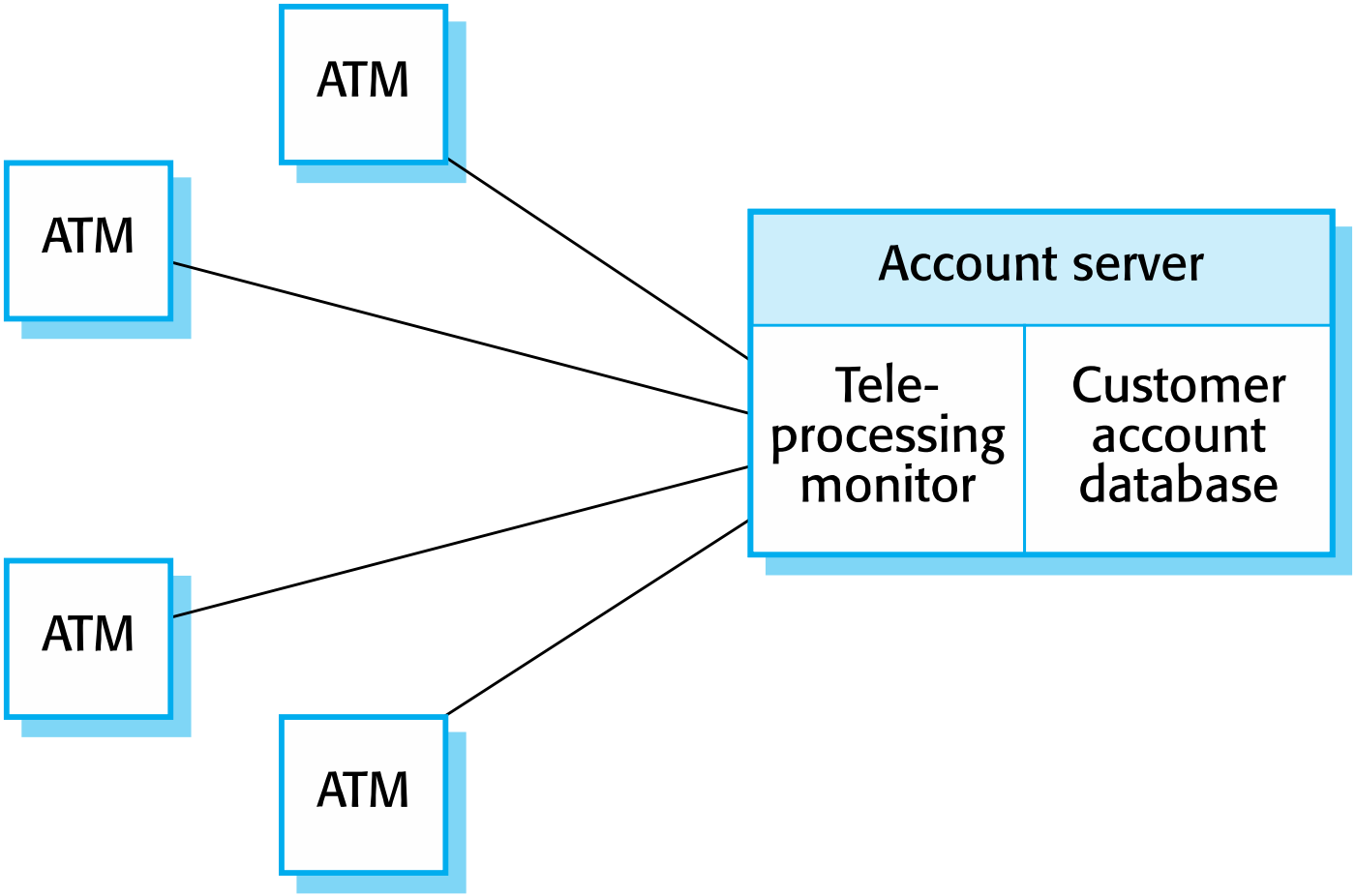
Thin client model

- 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 fat-client architecture for an ATM system

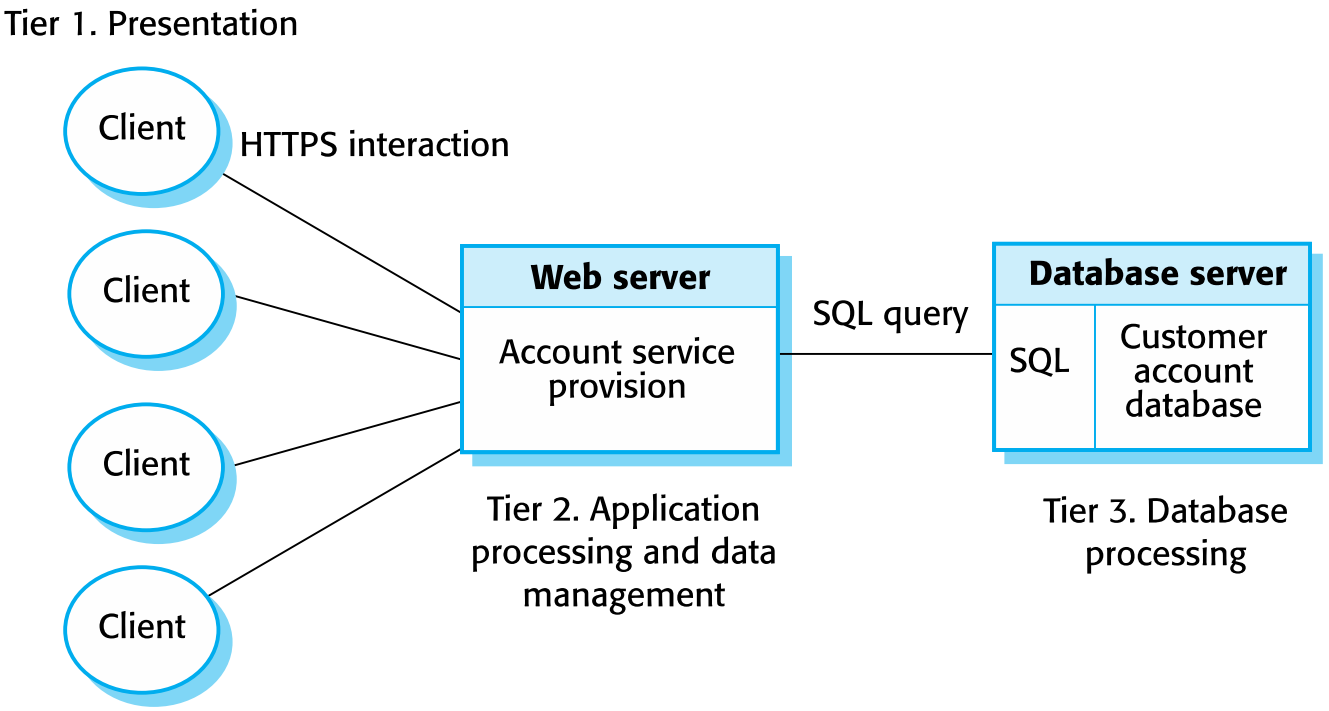


Multi-tier client-server architectures

- In a 'multi-tier client–server' architecture, the different layers of the system, namely presentation, data management, application processing, and database, are separate processes that may execute on different processors.
- This avoids problems with scalability and performance if a thin-client two-tier model is chosen, or problems of system management if a fat-client model is used.

- Applications: A large number of transactions that a server must support.

Three-tier architecture for an Internet banking system

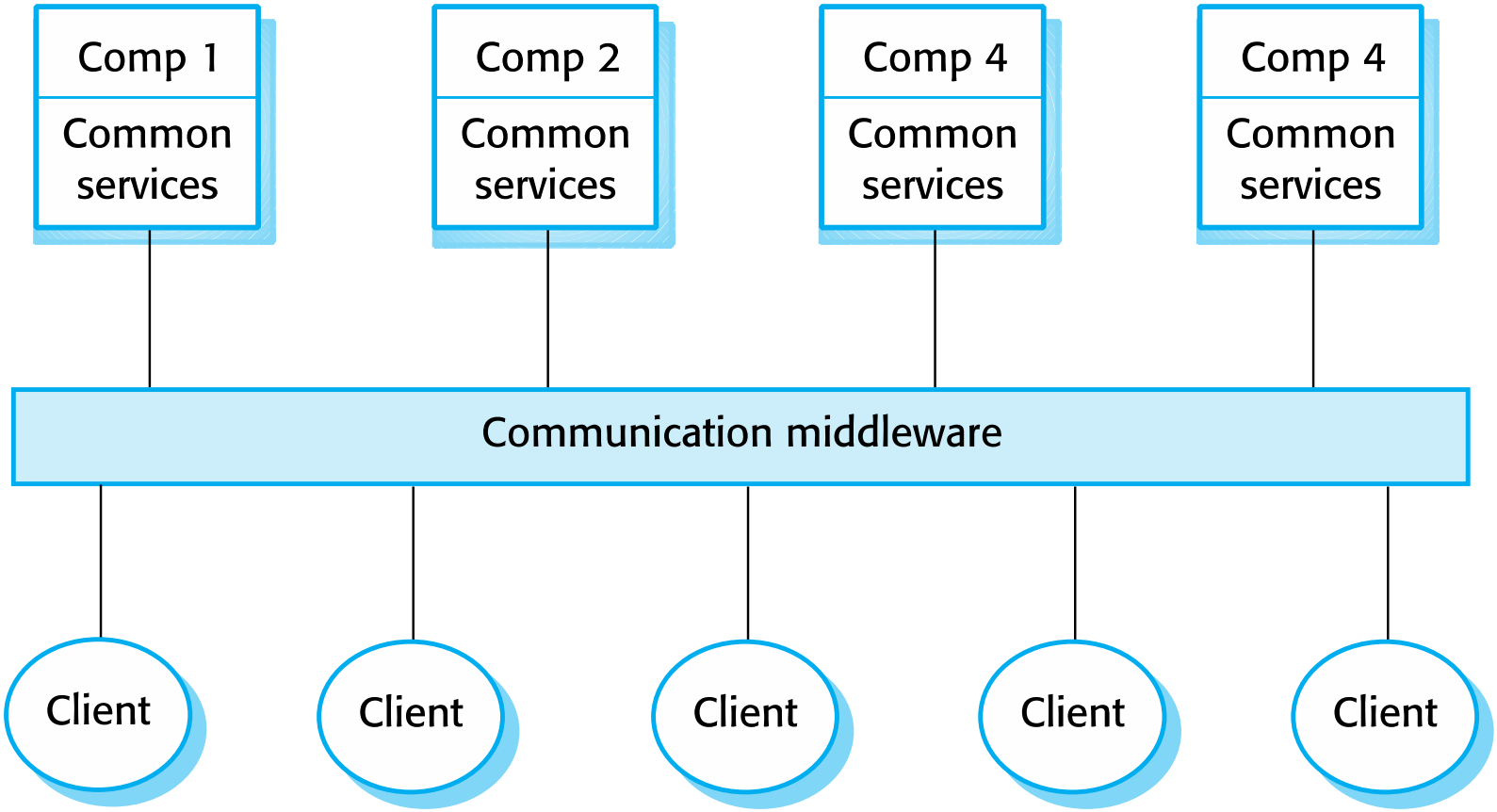


Distributed component architectures

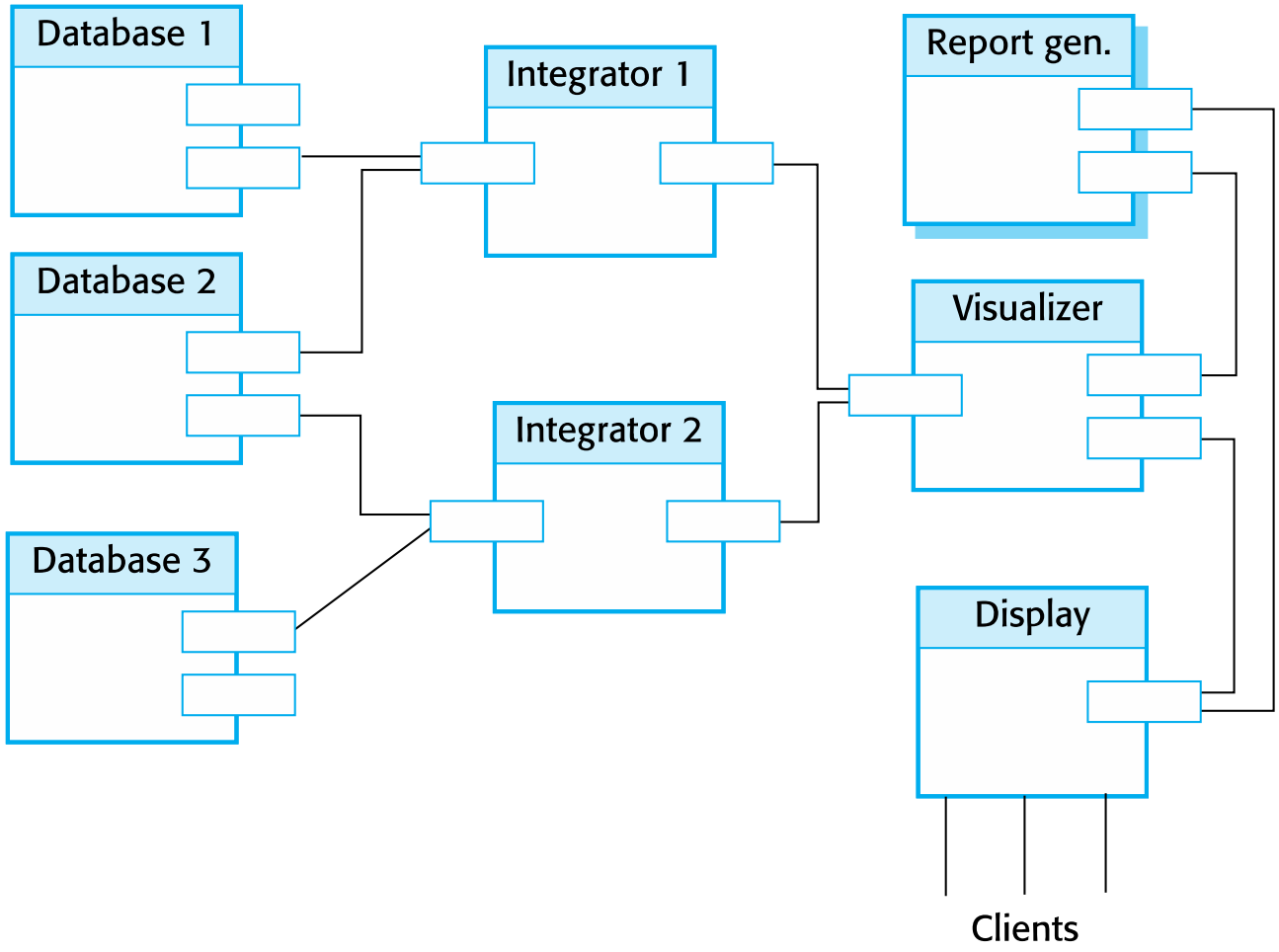
- There is no distinction in a distributed component architecture between clients and servers.
- Each distributable entity is a component that provides services to other components and receives services from other components.
- Component communication is through a middleware system.

- Applications: resources of various systems and databases must be combined. It can also be a model for implementing a multi-tier client server architecture.

A distributed component architecture



A distributed component architecture for a data mining system



Disadvantages of distributed component architecture

- They are more complex to design than client–server systems. Distributed component architectures are difficult for people to visualize and understand.
- Standardized middleware for distributed component systems has never been accepted by the community. Different vendors, such as Microsoft and Sun, have developed different, incompatible middleware.
  - As a result of these problems, service-oriented architectures are replacing distributed component architectures in many situations.

Service Oriented Architectures SOAs

- A means of developing distributed systems where the components are stand-alone services
- Services may execute on different computers from different service providers
- Standard protocols have been developed to support service communication and information exchange

RESTful web services

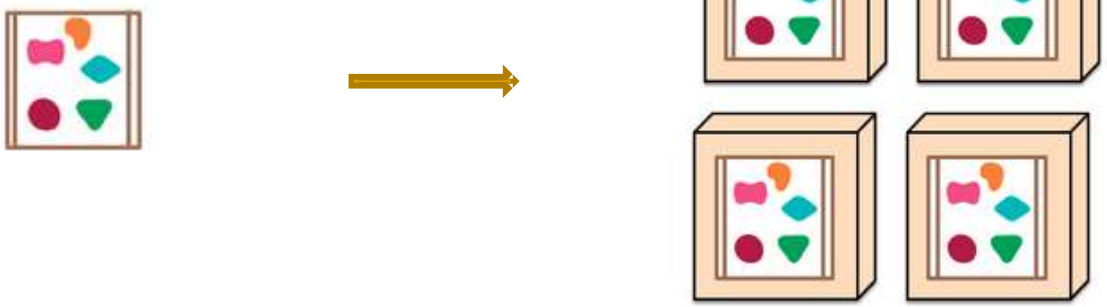
- REST (REpresentational State Transfer) is an architectural style based on transferring representations of resources from a server to a client.
- The pattern is based on the existing Internet infrastructure (does not introduce new protocols), which simplifies the implementation of network services.
- RESTful services involve a lower overhead than so-called 'big web services'

Microservices

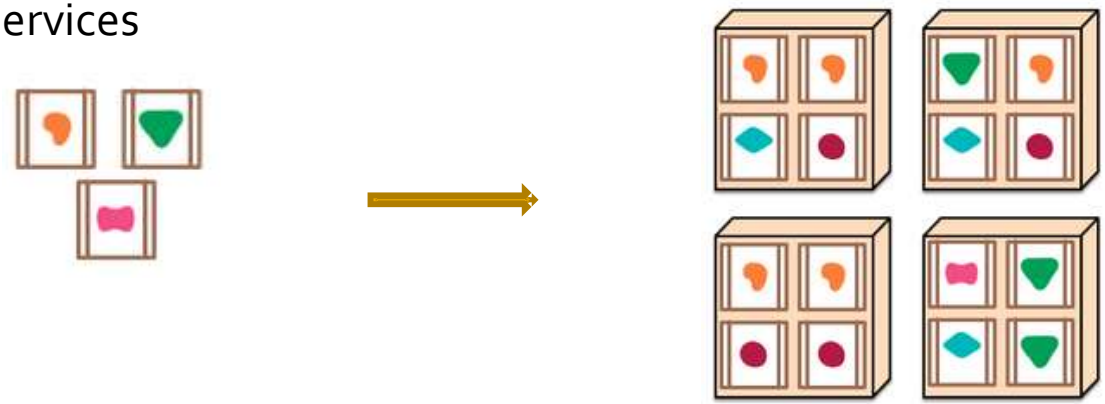
- Building applications with independently implemented services.
  - Launched in separate processes.
  - Communicating with each other using a "light" protocol.
  - Automatically implemented.

Scaling vs microservices

Monolithic application



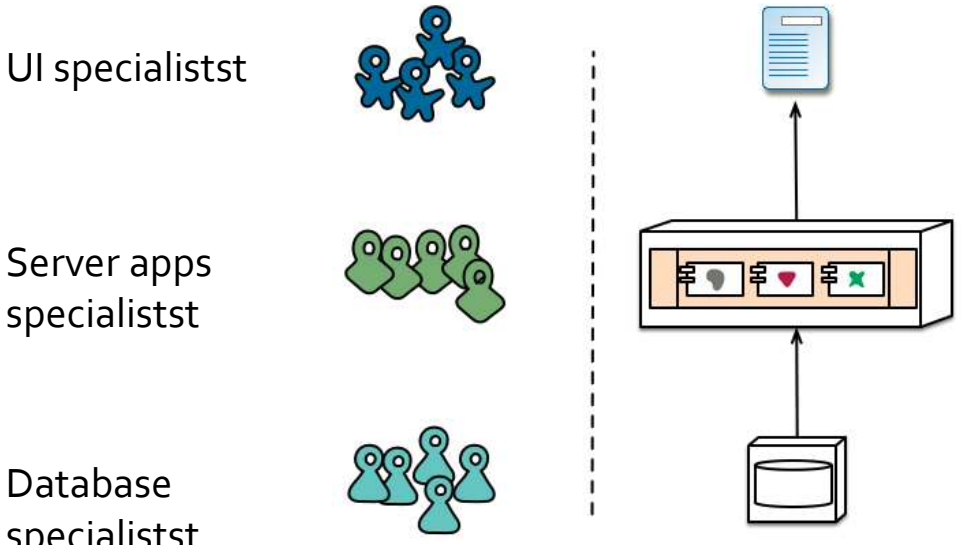
Microservices



Characteristics of microservice architecture

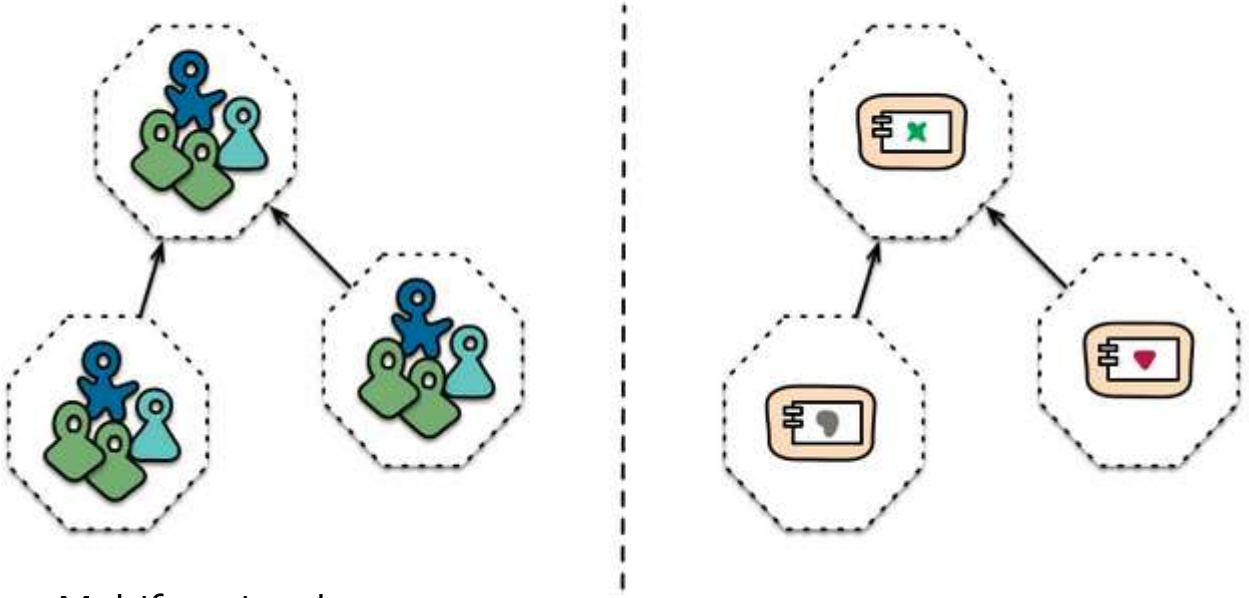
- Organized around a business character
  - Conway's law
    - Every organization that designs systems (broadly understood) strives for a structure that is a copy of the organization's communication structure.

http://www.melconway.com/Home/Committees_Paper.html



martinfowler.com/articles/microservices.html

Microservices and Agile



Multifunctional teams self-organizing around business opportunities

martinfowler.com/articles/microservices.html

Characteristics of microservice architecture

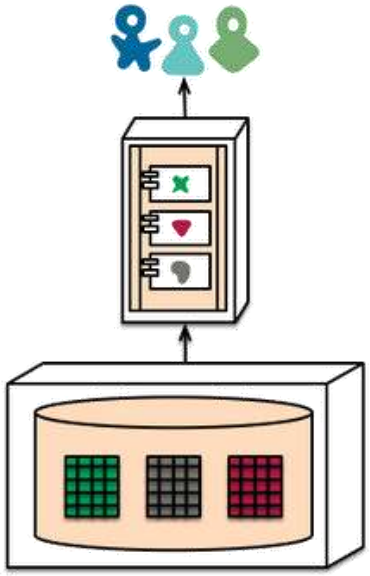
- **Products** not projects
  - Project approach - We provide complete software and "we disband the team"
  - **Product approach -** The team should be the "owner of the product" throughout its life

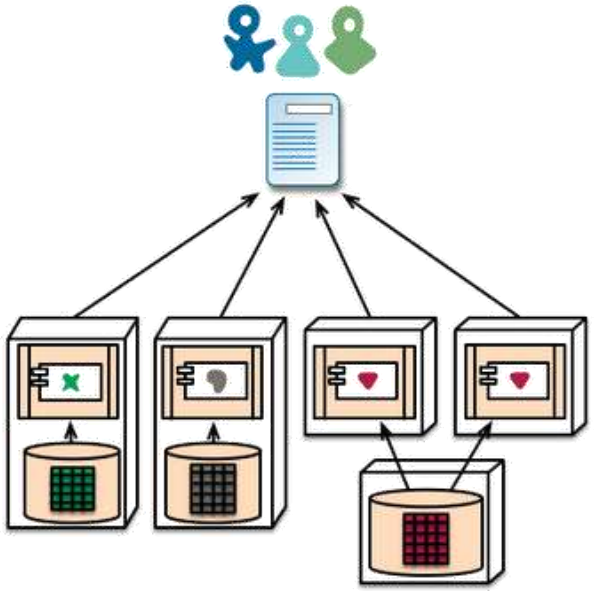
Characteristics of microservice architecture

- Decentralized management
  - Many technology platforms
    - Tools suitable for the problem
    - Developing tools for programmers (open-source)
  - Build it / run it
    - You are responsible for the operation of your system

Characteristics of microservice architecture

- Decentralized data administration
  - Polyglot persistence
    - Non-transaction coordination
    - Tolerating a certain level of inconsistency





martinfowler.com/articles/microservices.html

Characteristics of microservice architecture

- Designed for failures
  - Each service may fail, the system must be prepared for it
    - Quick failure detection (monitoring, real-time log view)
- Evolutionary design
  - Change by adding new services and removing existing ones
    - replaceability
    - restructuring

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.

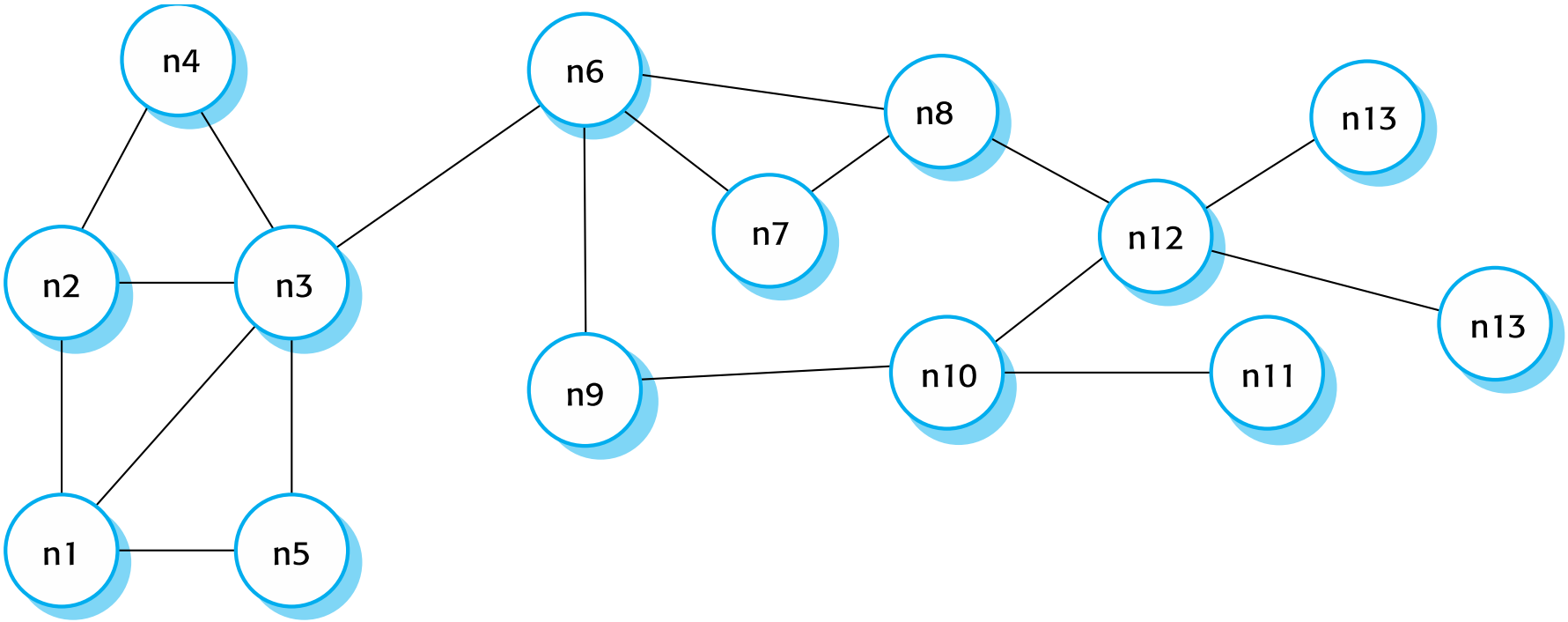
Peer-to-peer systems

- File sharing systems based on the BitTorrent protocol
- Messaging systems such as Jabber
- Payments systems – Bitcoin
- Databases – Freenet is a decentralized database
- Phone systems – Viber
- Computation systems - SETI@home

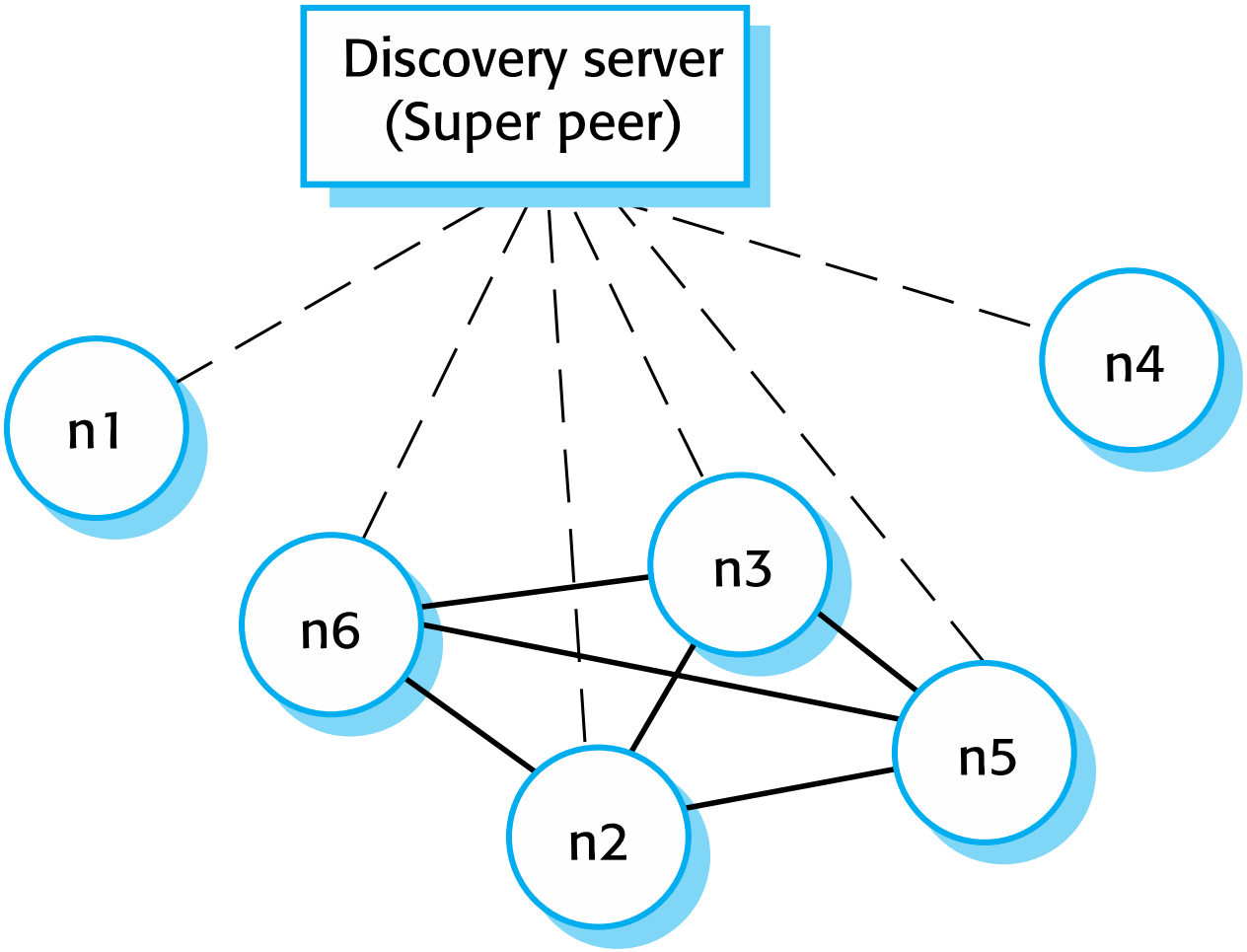
P2P architectural models

- 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.

A decentralized P2P architecture



A semicentralized P2P architecture



Software as a service - SaaS

- Software as a service (SaaS) involves hosting the software remotely and providing access to it over the Internet.
  - Software is deployed on a server (or more commonly a number of servers) and is accessed through a web browser. It is not deployed on a local PC.
  - The software is owned and managed by a software provider, rather than the organizations using the software.
  - Users may pay for the software according to the amount of use they make of it or through an annual or monthly subscription.
- This is one of the so-called cloud computing models.

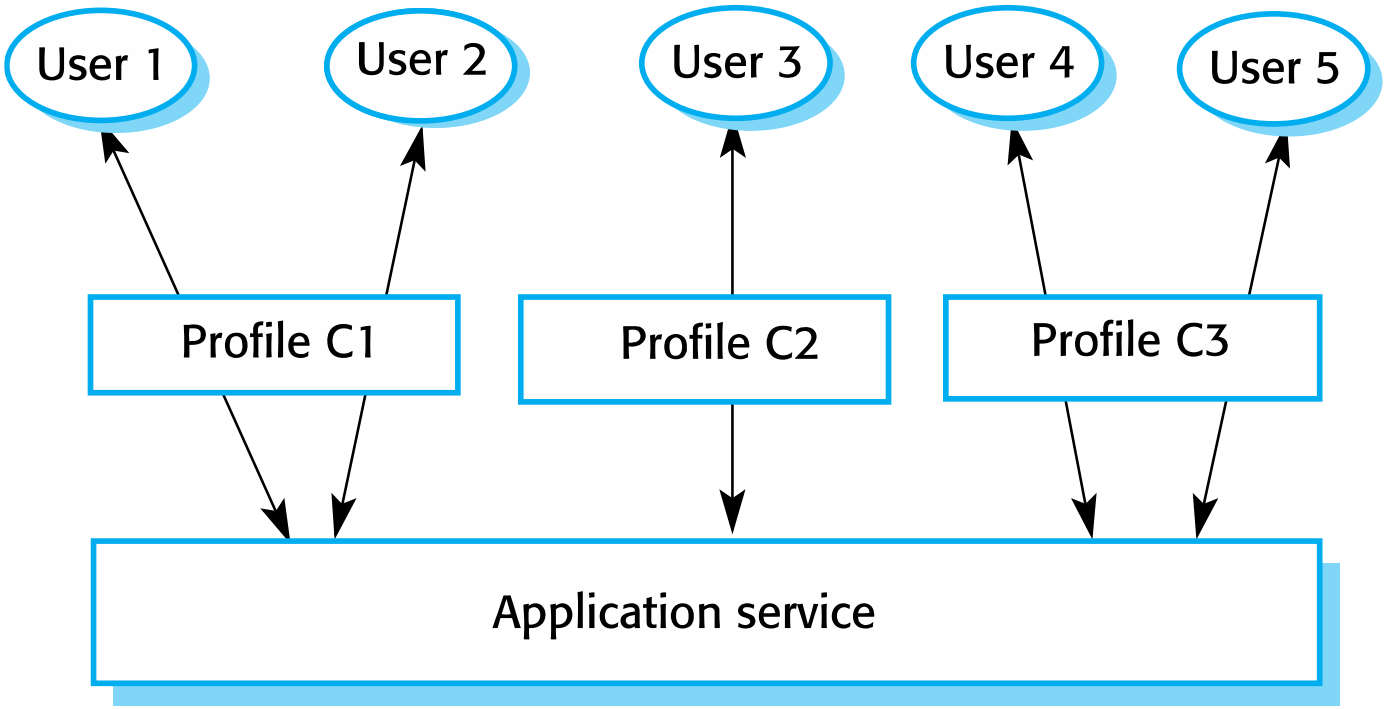
SaaS vs SOA

- Software as a service is a way of providing functionality on a remote server with client access through a web browser. The server maintains the user's data and state during an interaction session. Transactions are usually long transactions e.g. editing a document.

- Service-oriented architecture is an approach to structuring a software system as a set of separate, stateless services. These may be provided by multiple providers and may be distributed. Typically, transactions are short transactions where a service is called, does something then returns a result.

Implementation factors for SaaS (1)

- ***Configurability*** *- How do you configure software for the specific requirements of different organizations?*



Implementation factors for SaaS (2)

- Renting a shared resource pool (called Multi-tenancy):
  - Share the software with the user in such a way that he has the impression that he is working on his own copy of the system.
  - The system architecture enables efficient use of resources.
  - This requires an absolute separation of functionality and system data.

A multitenant database

Tenant	Key	Name	Address
234	C100	XYZ Corp	43, Anystreet, Sometown
234	C110	BigCorp	2, Main St, Motown
435	X234	J. Bowie	56, Mill St, Starville
592	PP37	R. Burns	Alloway, Ayrshire

Implementation factors for SaaS (3)

- ***Scalability*** *- How to design a system so that it can be scaled to an unpredictably large number of users?*
  - Develop applications where each component is implemented as a simple stateless service that may be run on any server.
  - Design the system using asynchronous interaction so that the application does not have to wait for the result of an interaction (such as a read request).
  - Manage resources, such as network and database connections, as a pool so that no single server is likely to run out of resources.
  - Design your database to allow fine-grain locking.

Key points

- A software architecture is a description of how a software system is organized.
- Architectural design decisions include decisions on the type of application, the distribution of the system, the architectural styles to be used.
- Architectures may be documented from several different perspectives or views such as a conceptual view, a logical view, a process view, and a development view.
- Architectural patterns are a means of reusing knowledge about generic system architectures. They describe the architecture, explain when it may be used and describe its advantages and disadvantages.

Key points

- The benefits of distributed systems are that they can be scaled to cope with increasing demand, can continue to provide user services if parts of the system fail, and they enable resources to be shared.
- Issues to be considered in the design of distributed systems include transparency, openness, scalability, security, quality of service and failure management.
- Client–server systems are structured into layers, with the presentation layer implemented on a client computer. Servers provide data management, application and database services.
- Client-server systems may have several tiers, with different layers of the system distributed to different computers.

Key points

- Architectural patterns for distributed systems include master-slave architectures, two-tier and multi-tier client-server architectures, distributed component architectures and peer-to-peer architectures.
- Distributed component systems require middleware to handle component communications and to allow components to be added to and removed from the system.
- Peer-to-peer architectures are decentralized with no distinguished clients and servers. Computations can be distributed over many systems in different organizations.
- Software as a service is a way of deploying applications as thin client- server systems, where the client is a web browser.