

Distributed software engineering

Architectural design for software that
executes on more than one processor

Modelli di Progettazione e Pattern Architeturali


STRUTTURAZIONE DELL'ARCHITETTURA

- 1.1 REPOSITORY
- 1.2 CLIENT-SERVER
- 1.3 MACCHINE ASTRATTE

CONTROLLO

- 2.1 CENTRALIZZATO
 - 2.1.1 CALL/RETURN
 - 2.1.2 MANAGER MODEL
- 2.2 EVENT DRIVEN
 - 2.2.1 BROADCAST
 - 2.2.2 INTERRUPT-DRIVEN

SCOMPOSIZIONE MODULARE

- 3.1 OBJEC-ORIENTED ( Baruzzo)
- 3.2 FUNZIONALE (DATA FLOW, PIPELINE, STRUCTURAL)

ARCHITETTURE SPECIFICHE DEL DOMINIO

- 4.1 GENERICHE [MVC]
- 4.2 REFERENCE [ISO/OSI]

ARCHITETTURE APPLICATIVE GENERICHE

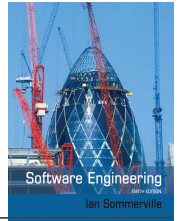
- 1. TRANSACTION PROCESSING [information systems, e-commerce]
- 2. LANGUAGE PROCESSING [compiler, interpreter, ...]

SISTEMI DISTRIBUITI

- 1. SISTEMI MULTIPROCESSORE
- 2. CLIENT-SERVER
 - 2.1 TWO TIER
 - 2.1.1 THIN CLIENT
 - 2.1.2 FAT CLIENT
 - 2.2 MULTI-TIER, THREE-TIER

Distributed systems

Distributed systems (vs. centralized)



❓ Virtually all large computer-based systems are now distributed systems.

“... a collection of independent computers that appears to the user as a single coherent system.”

❓ 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.

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

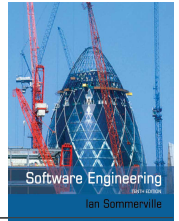
? The ability to continue in operation after a fault has occurred.

Distributed systems issues



- ❑ Distributed systems are **more complex** than systems that run on a single processor.
- ❑ Complexity arises because different parts of the system are **independently** managed as is the network.
- ❑ There is **no single authority** in charge of the system so top-down control is impossible.

Distributed system problems/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.

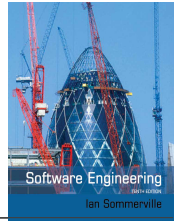
Design issues: Several decisions

- ❑ **Resource identification. Communication.**
- ❑ **Transparency.** To what extent should the distributed system appear to the user as a single system?
- ❑ **Openness.** Should a system be designed using standard protocols that support interoperability?
- ❑ **Scalability.** How can the system be constructed so that it is scaleable?
- ❑ **Security.** How can usable security policies be defined and implemented?
- ❑ **Failure management.** How can system failures be detected, contained and repaired? ❑ **Resilience**
- ❑ **Quality of services - QoS** (performance, affidabilità, disponibilità, ...). How should the quality of service be specified.



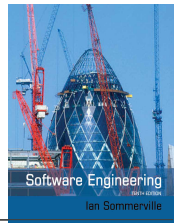
- ❑ **Software architecture**

Middleware



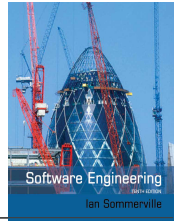
↳ ETEROGENEITA' tra le varie componenti

- [?] The components in a distributed system may be implemented in different programming languages and may execute on completely different types of processor and operating systems. Models of data, information representation and protocols for communication may all be different.
- [?] **Middleware** is software that can manage these diverse parts, and ensure that they can communicate and exchange data.

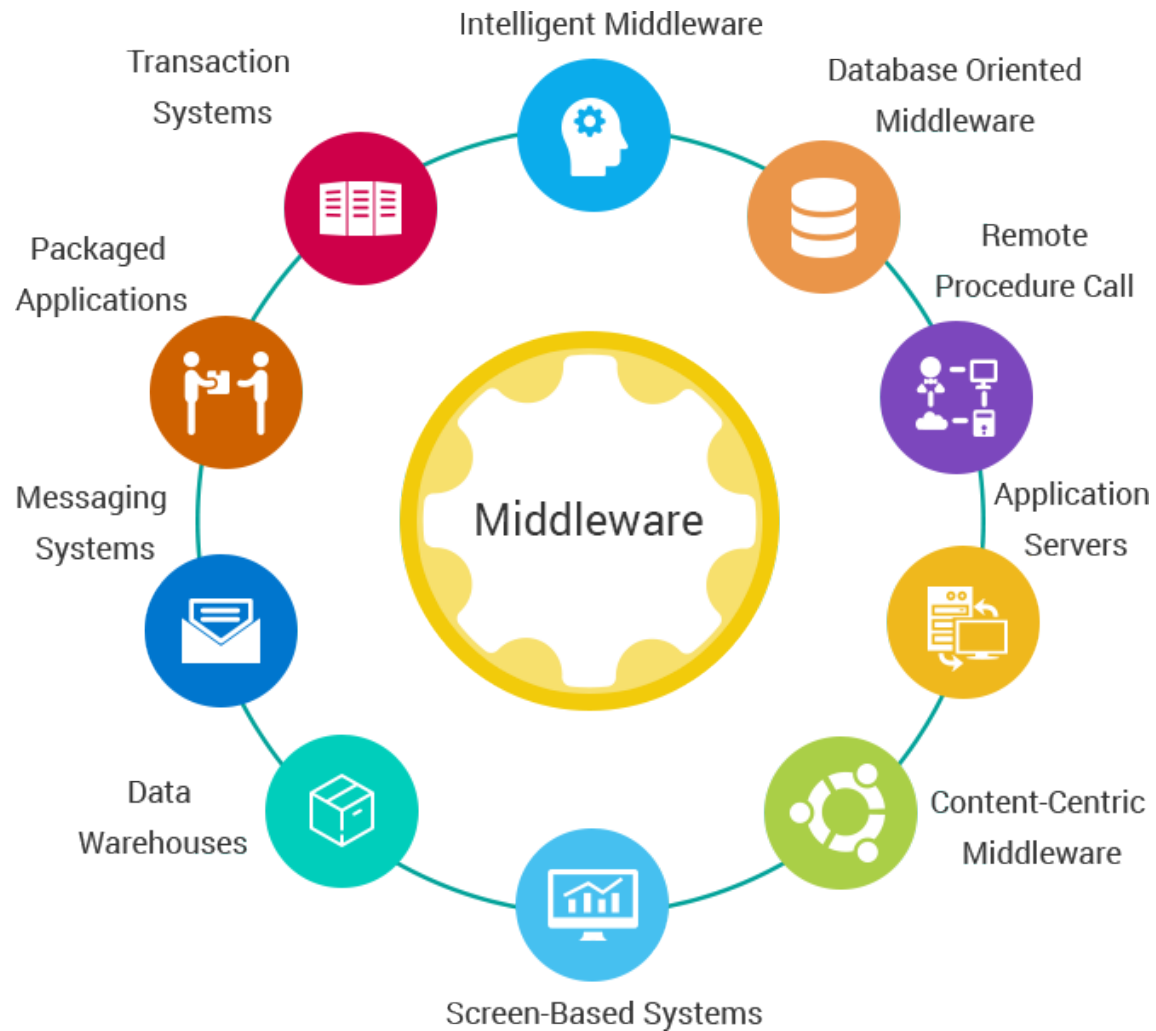


Middleware

Middleware

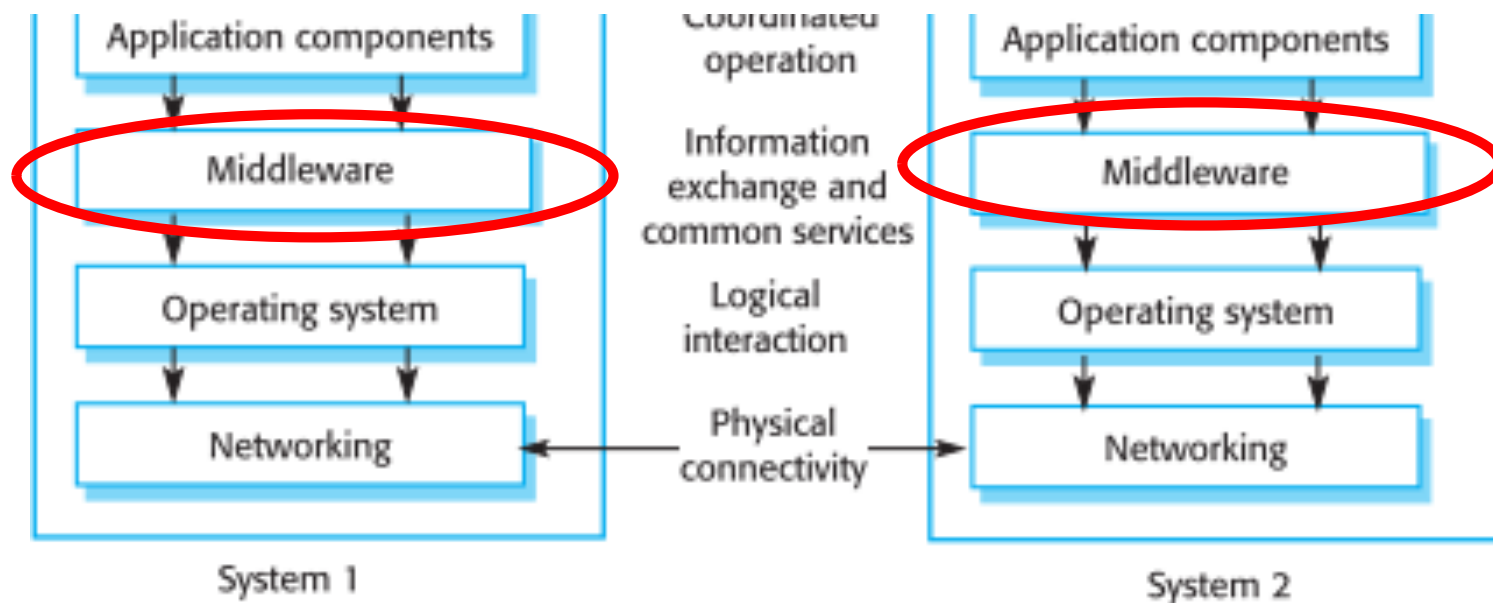


- ❓ Software that manages and **supports the different components** of a distributed system, which sits in the *middle* of the system
- ❓ Middleware **is usually off-the-shelf** rather than specially written software
- ❓ Examples
 - ❓ Transaction processing monitors, Data converters, Communication controllers, interaction support and coordination, provision of common services.
- ❓ Il middleware mappa l'architettura logica nelle risorse fisiche e gestisce le comunicazioni e agisce da intermediatore
- ❓ Es. di sistemi/tool middleware: *CORBA*, *IBM WebSphere*, *Oracle Service Bus*, *SOAP*, *REST*, *JSON*, (*Architetture orientate ai servizi*), *Apache ...*, *TOMCAT*, *MS BizTalk Server*, ...



Middleware in a distributed system

→ rende astratto a chi sviluppa sw applicativo di ciò che c'è sotto, il programmatore non si deve preoccupare



Middleware support



[?] Interaction support, where the middleware coordinates interactions between different components in the system

[?] The middleware provides location transparency in that it isn't necessary for components to know the physical locations of other components.

→ ti dà anche una libreria grafica che puoi usare

[?] The provision of common services, where the middleware provides reusable implementations of services that may be required by several components in the distributed system.

[?] By using these common services, components can easily inter-operate and provide user services in a consistent way.

Modelli di Progettazione e Pattern Architeturali


STRUTTURAZIONE DELL'ARCHITETTURA

- 1.1 REPOSITORY
- 1.2 CLIENT-SERVER
- 1.3 MACCHINE ASTRATTE

CONTROLLO

- 2.1 CENTRALIZZATO
 - 2.1.1 CALL/RETURN
 - 2.1.2 MANAGER MODEL
- 2.2 EVENT DRIVEN
 - 2.2.1 BROADCAST
 - 2.2.2 INTERRUPT-DRIVEN

SCOMPOSIZIONE MODULARE

- 3.1 OBJEC-ORIENTED ( Baruzzo)
- 3.2 FUNZIONALE (DATA FLOW, PIPELINE, STRUCTURAL)

ARCHITETTURE SPECIFICHE DEL DOMINIO

- 4.1 GENERICHE [MVC]
- 4.2 REFERENCE [ISO/OSI]

ARCHITETTURE APPLICATIVE GENERICHE

- 1. TRANSACTION PROCESSING [information systems, e-commerce]
- 2. LANGUAGE PROCESSING [compiler, interpreter, ...]

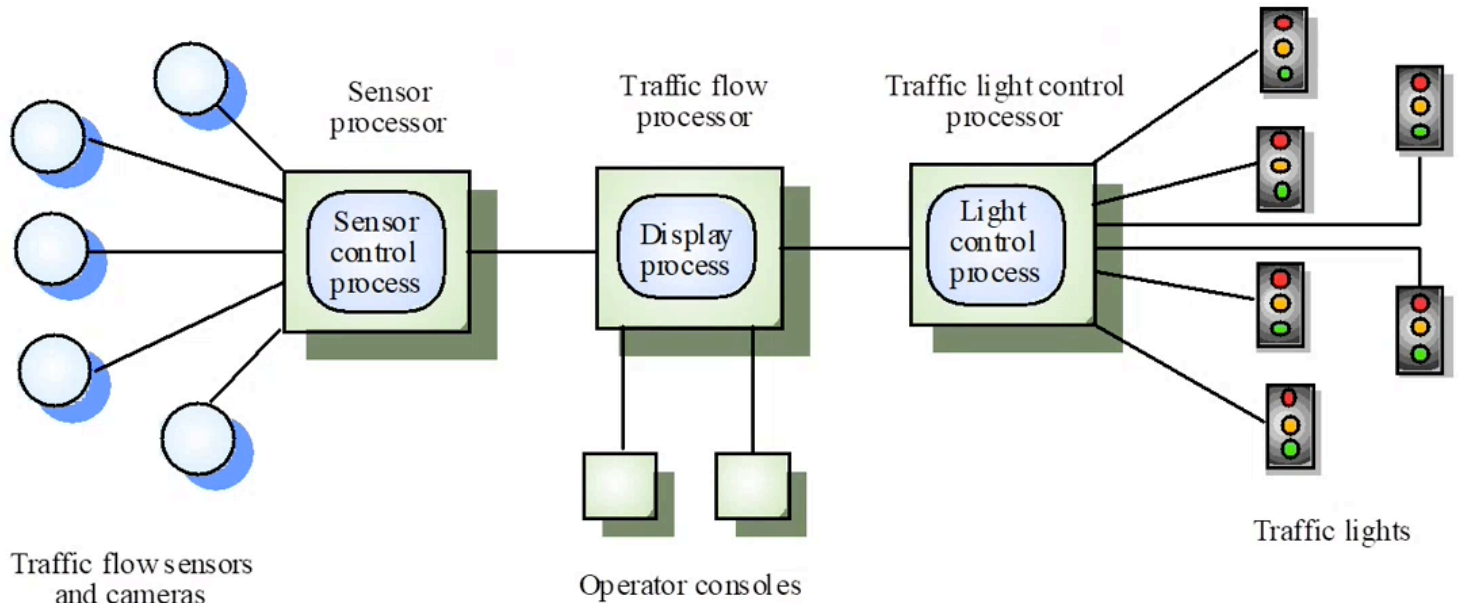
SISTEMI DISTRIBUITI

- 1. SISTEMI MULTIPROCESSORE
- 2. CLIENT-SERVER
 - 2.1 TWO TIER
 - 2.1.1 THIN CLIENT
 - 2.1.2 FAT CLIENT
 - 2.2 MULTI-TIER, THREE-TIER

1. 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 processors may be pre-ordered or may be under the control of a dispatcher or organized in an architecture following the master slave pattern

A multiprocessor traffic control system

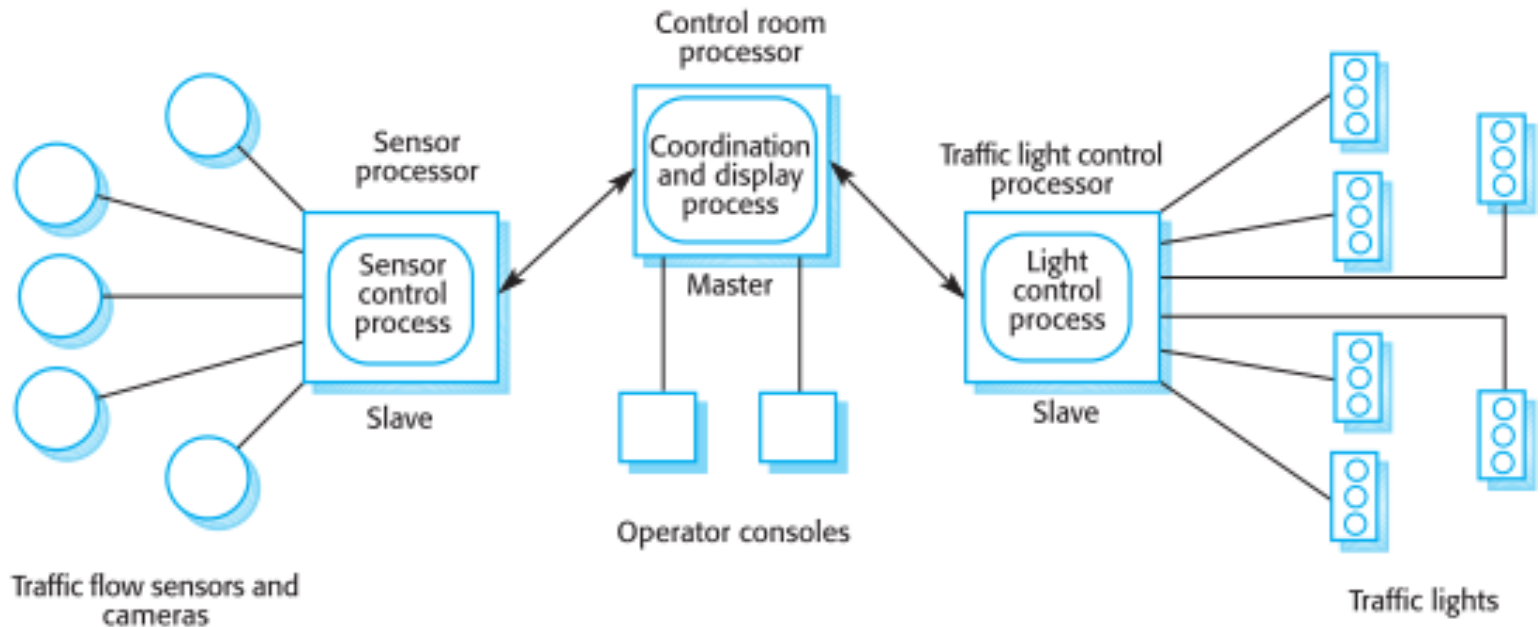


→ una macchina gestisce tutte le altre

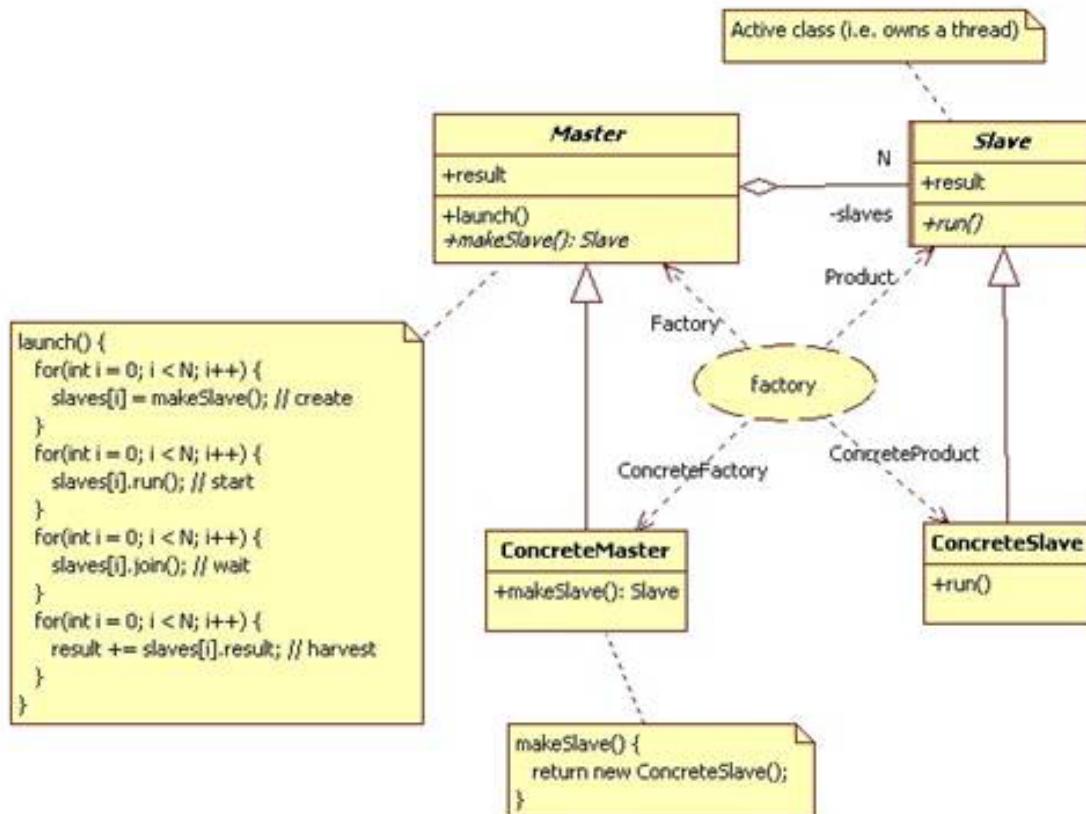
Master-slave pattern for architectures

- ❓ Master-slave architectures are commonly used in real-time systems where there may be separate processors associated with data acquisition from the system's environment, data processing and computation and actuator management.
- ❓ The 'master' process is usually responsible for computation, coordination and communications and it controls the 'slave' processes.
- ❓ 'Slave' processes are dedicated to specific actions, such as the acquisition of data from an array of sensors or actuation

A traffic management system with a **master-slave** architecture



Pattern Master-Slave



Distributed systems architectures



? Client-server architectures

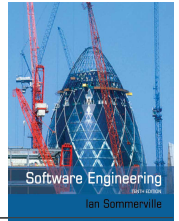
- ? Distributed services which are called on by clients. Servers that provide services are treated differently from clients that use services

? Distributed object architectures

- ? No distinction between clients and servers. Any object on the system may provide and use services from other objects
MORE general

Client-server computing

2. 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.

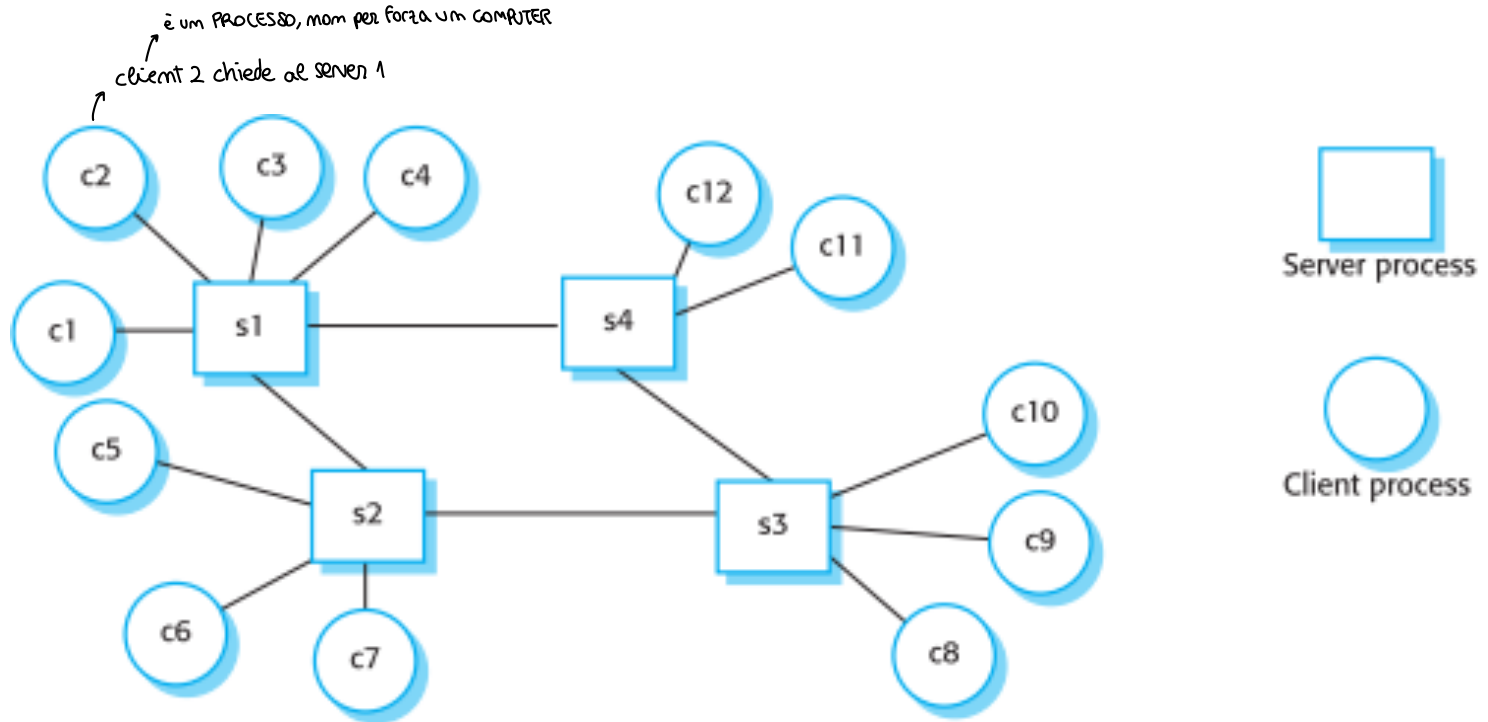
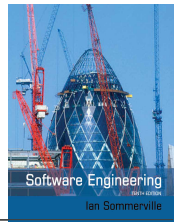
Client and server:

Logical processes vs. **Physical** Organization

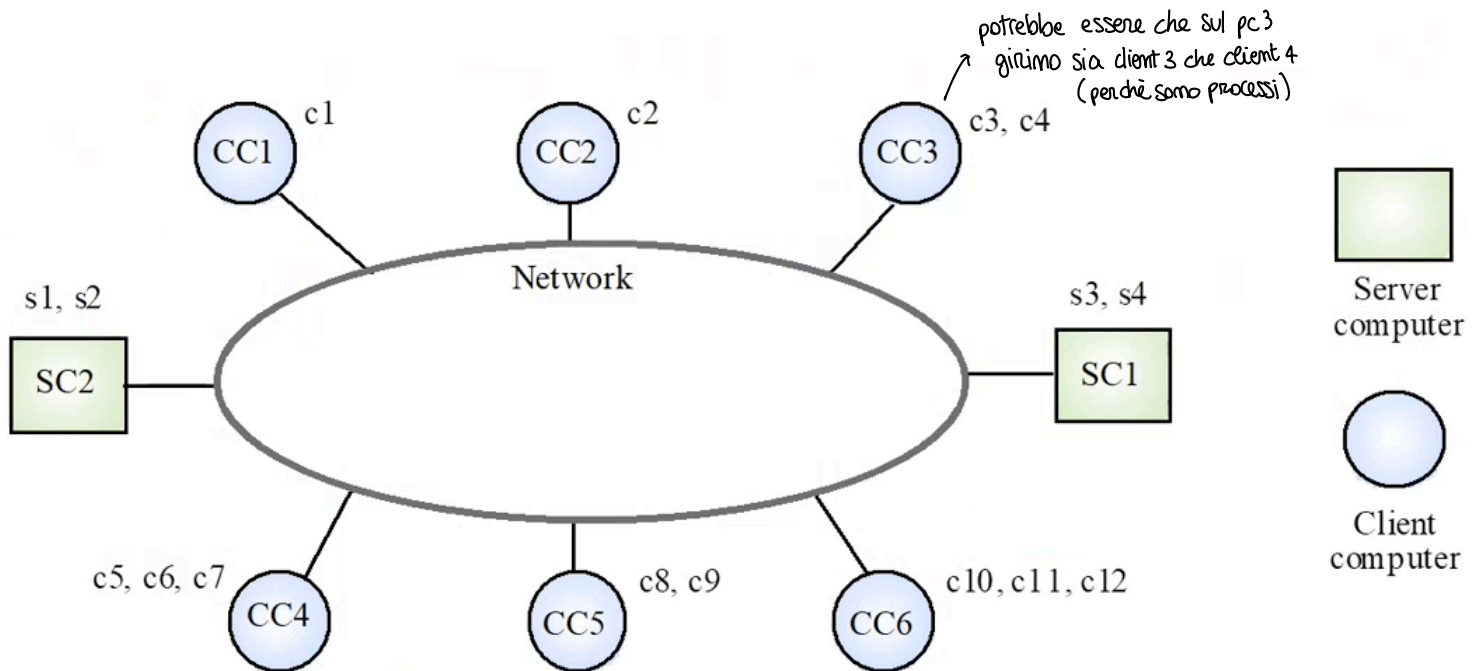


- ❑ The application is modelled as a set of services that are provided by servers and a set of clients that use these services
- ❑ **Clients know of servers but servers need not know of clients**
- ❑ Clients and servers are **logical** processes
- ❑ **The mapping of processors (hw) to processes is not necessarily 1 : 1**

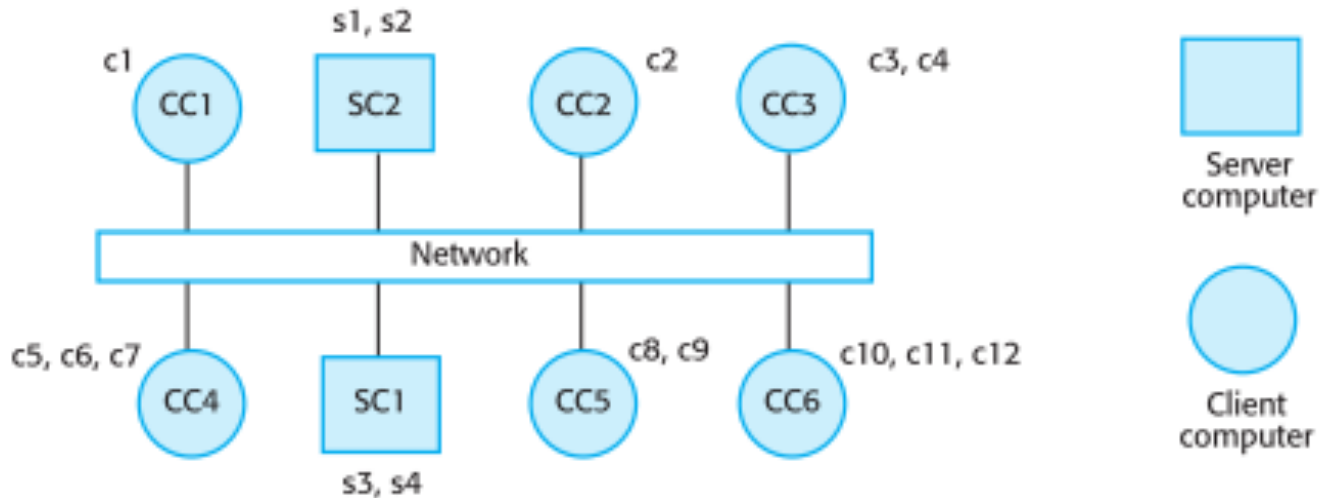
Client-server interaction (**logical**)



Computers in a C/S network (physical)



Mapping of clients and servers to networked computers



Architectural patterns for distributed client-server systems

Typical architectural Approach: **Layered architectural** model for client–server applications



Presentation

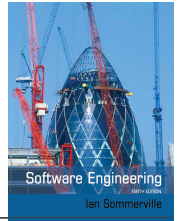
Data handling

Application processing

Database

- Distinction of the different layers
- Various distribution over processors (/servers)

Layers in a client/server system



? *Presentation*

- ? concerned with **presenting information** to the user and **managing all user interaction**.

? *Data handling*

- ? **manages** the **data** that is passed **to** and **from** the client.
Implement checks on the data, generate web pages, etc.

? *Application processing layer*

- ? concerned with **implementing the logic of the application** and so **providing the required functionality** to end users.

? *Database/Data Management layer*

- ? **Stores** data and provides **transaction management** services, etc.

Architectural patterns



- ❓ Widely used ways of organizing the architecture of a distributed system:
 - ❓ *Master-slave architecture*, which is used in real-time systems in which guaranteed interaction response times are required.
 - ❓ **Two-tier client-server architecture**, which is used for simple client-server systems, and where the system is centralized for security reasons.
 - ❓ **Multi-tier client-server architecture**, which is used when there is a high volume of transactions to be processed by the server.
 - ❓ *Distributed component architecture*, which is used when resources from different systems and databases need to be combined, or as an implementation model for multi-tier client-server systems.
 - ❓ *Peer-to-peer architecture*, which is used when clients exchange locally stored information and the role of the server is to introduce clients to each other

2.1 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.

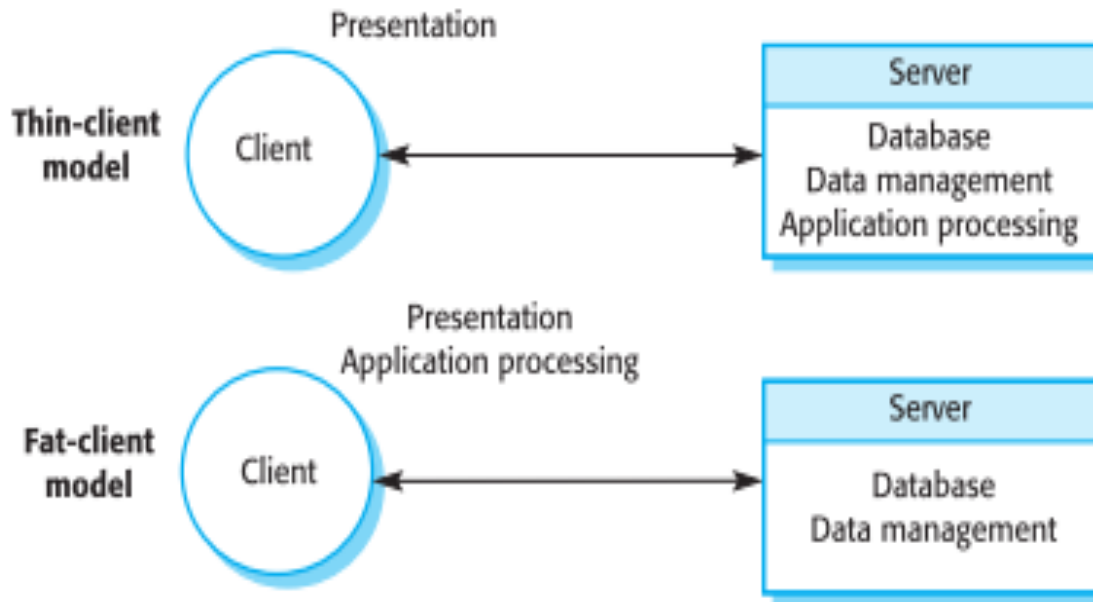
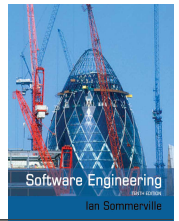
→ vi è implementato (nel THIN CLIENT) solo lo strato presentazione

[?] **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.

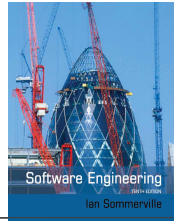
[?] **Fat**-client model, where some or all of the application processing is carried out on the client. Data management and database functions are implemented on the server.

↳ non ha Application processing il SRV

Thin- and fat-client architectural models



2.1.1 Thin client model



? Used when **legacy systems** are migrated to client server architectures.

→ l'interfaccia è esterna sul client

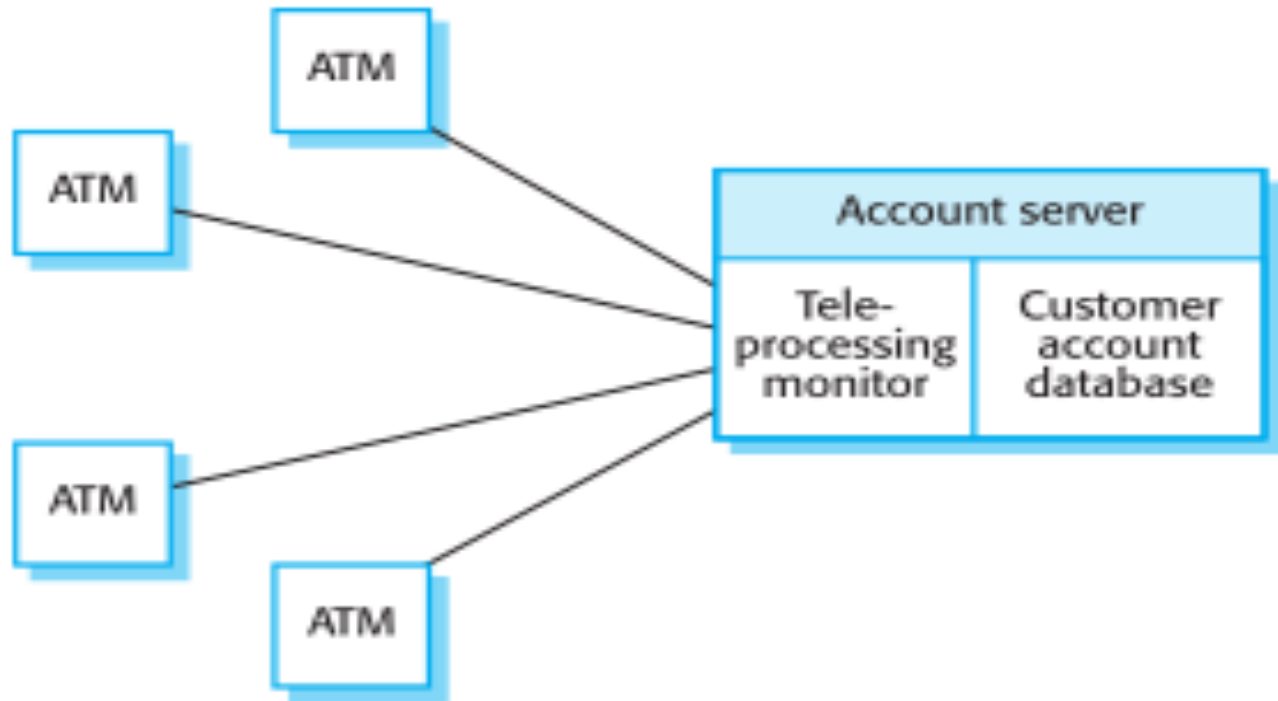
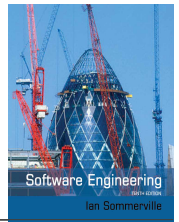
? The legacy system acts as a server in its own right with a graphical interface implemented on a client.

? A major **disadvantage** is that it places a **heavy processing** load on both the **server** and the **network**.

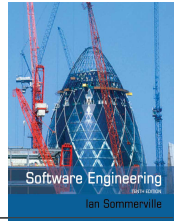
2.1.2 Fat client model

- ❑ **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 powerful client**
- ❑ **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

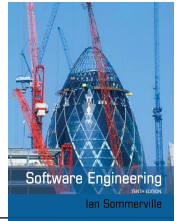


Evolution of thin and fat clients



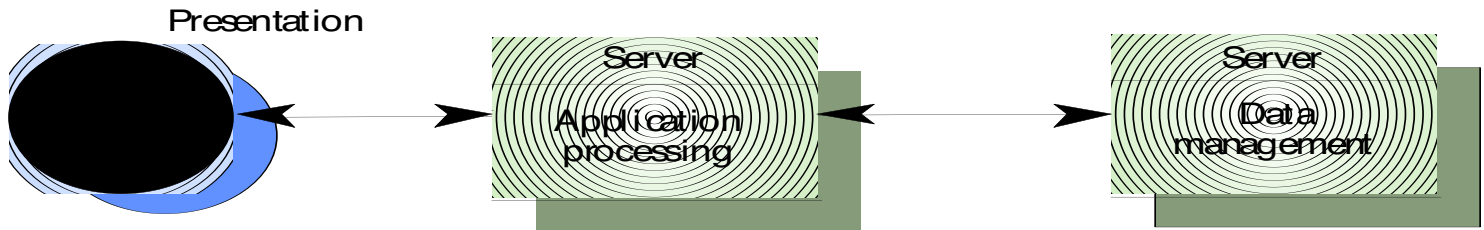
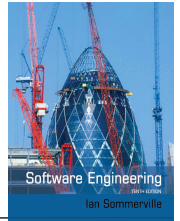
- ❑ **Distinction** between thin and fat client architectures has become **blurred** and has **changed over time**
- ❑ The old solution of **Java-Applet** has been often substituted by **Javascript**, which allows local processing in a browser so 'fat-client' functionality available without software installation
- ❑ **Mobile** Apps carry out some local processing or archiving to minimize demands on network
- ❑ **Auto-update** of apps reduces management problems
- ❑ There are now **very few thin-client** applications with all processing carried out on remote server.

2.2 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.
- ❑ A **more scalable** architecture - as demands increase, extra servers can be added

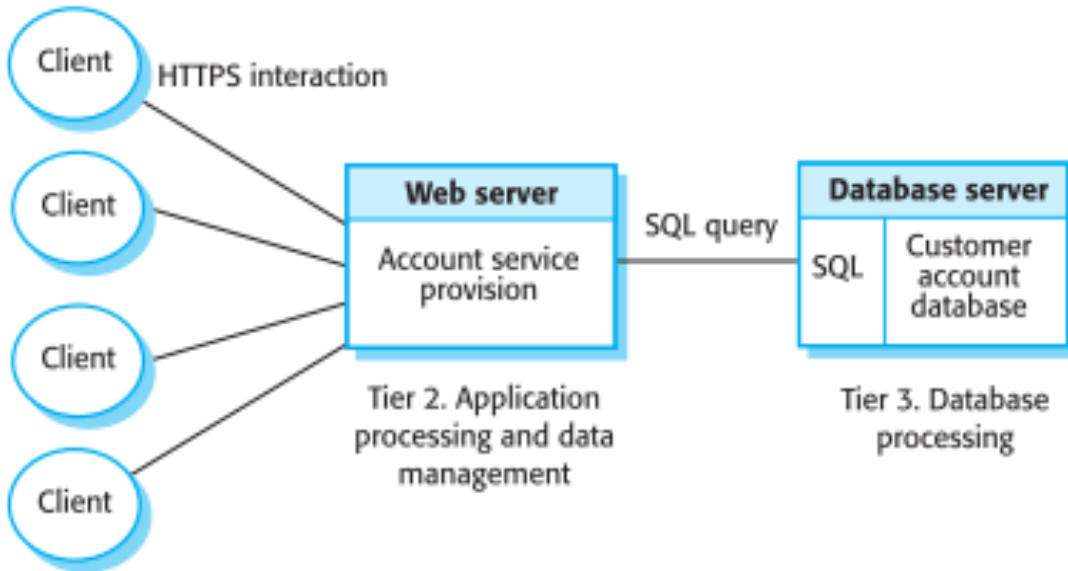
A 3-tier C/S architecture



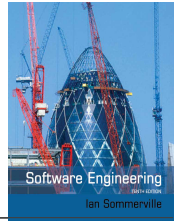
Three-tier architecture for an Internet banking system



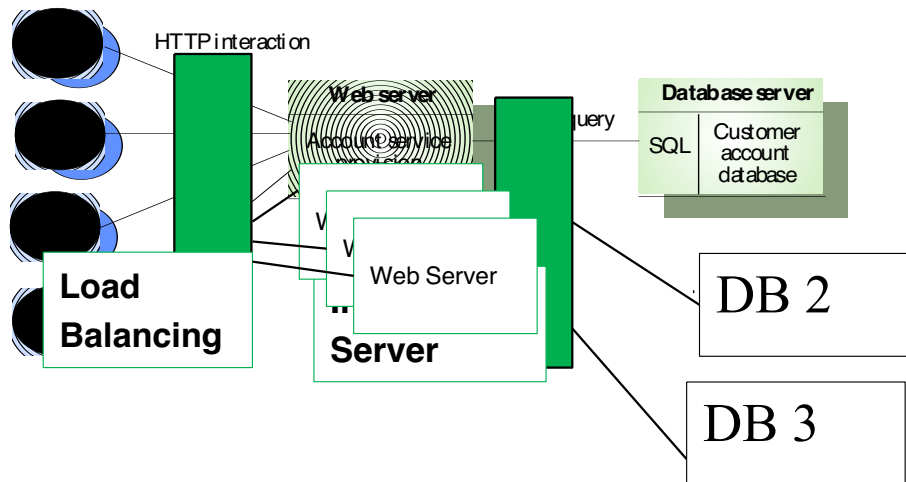
Tier 1. Presentation



Multi-tier Architectures



- ? More than three levels, there are other servers
- ? Example: access to different DB servers, which are interfaced through an integration server
- ? Similarly, load balancing server

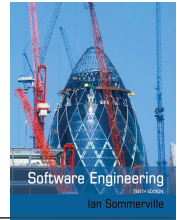


Use of client–server architectural patterns



Architecture	Applications
Two-tier client–server architecture with thin clients	<p>Legacy system applications that are used when separating application processing and data management is impractical. Clients may access these as services, as discussed in Section 18.4.</p> <p>Computationally intensive applications such as compilers with little or no data management.</p> <p>Data-intensive applications (browsing and querying) with nonintensive application processing. Browsing the Web is the most common example of a situation where this architecture is used.</p>

Use of client–server architectural patterns



Architecture	Applications
Two-tier client-server architecture with fat clients	<p>Applications where application processing is provided by off-the-shelf software (e.g., Microsoft Excel) on the client.</p> <p>Applications where computationally intensive processing of data (e.g., data visualization) is required.</p> <p>Mobile applications where internet connectivity cannot be guaranteed. Some local processing using cached information from the database is therefore possible.</p>
Multi-tier client–server architecture	<p>Large-scale applications with hundreds or thousands of clients.</p> <p>Applications where both the data and the application are volatile.</p> <p>Applications where data from multiple sources are integrated.</p>

STRUTTURAZIONE DELL'ARCHITETTURA

- 1.1 REPOSITORY
- 1.2 CLIENT-SERVER
- 1.3 MACCHINE ASTRATTE

CONTROLLO

- 2.1 CENTRALIZZATO
 - 2.1.1 CALL/RETURN
 - 2.1.2 MANAGER MODEL
- 2.2 EVENT DRIVEN
 - 2.2.1 BROADCAST
 - 2.2.2 INTERRUPT-DRIVEN

SCOMPOSIZIONE MODULARE

- 3.1 OBJEC-ORIENTED
- 3.2 FUNZIONALE (DATA FLOW, PIPELINE, STRUCTURAL)

ARCHITETTURE SPECIFICHE DEL DOMINIO

- 4.1 GENERICHE [MVC]
- 4.2 REFERENCE [ISO/OSI]

ARCHITETTURE APPLICATIVE GENERICHE

- 1. TRANSACTION PROCESSING [information systems, e-commerce]
- 2. LANGUAGE PROCESSING [compiler, interpreter, ...]

SISTEMI DISTRIBUITI

- 1. SISTEMI MULTIPROCESSORE
- 2. CLIENT-SERVER
 - 2.1 TWO TIER
 - 2.1.1 THIN CLIENT
 - 2.1.2 FAT CLIENT
 - 2.2 MULTI-TIER, THREE-TIER

Modelli di Progettazione

Set 8 - Cosa ricordare: concetti, motivazioni, conseguenze, relazioni fra concetti, ecc.

- ❓ Cos'è un sistema distribuito (S.D.). Caratteristiche/Obiettivi, vantaggi, criticità. Aspetti progettuali da considerare. 2 tipologie: sistemi client-server e sistemi ad oggetti distribuiti. Middleware.
- ❓ Sistemi Multiprocessore. Pattern master-slave. Architettura logica e architettura fisica. Tipologie di client-server, architettura a 4 strati di riferimento, caratteristiche, vantaggi e limitazioni: two-tier, thin e fat client, distinzione ed evoluzione; three-tier e multi-tier.