DISTRIBUTED SYSTEMS
UNIT-1

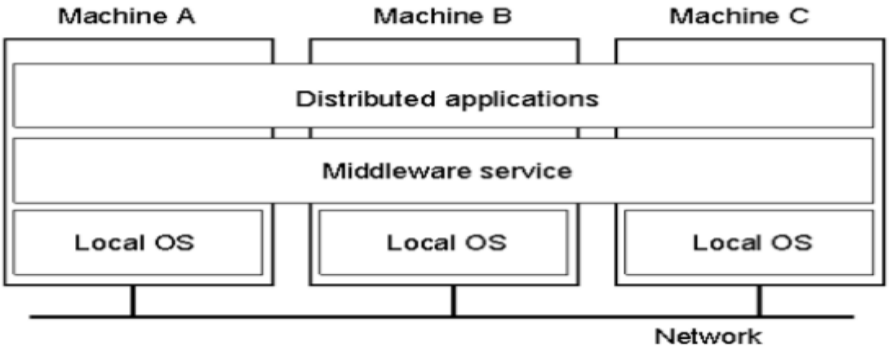
Introduction: Definition of Distributed Systems, Goals: Connecting Users and Resources, Transparency, Openness, Scalability, Hardware Concepts: Multiprocessors, Homogeneous Multicomputer systems, Heterogeneous Multicomputer systems, Software Concepts: Distributed Operating Systems, Network Operating Systems, Middleware, The client-server model: Clients and Servers, Application Layering, Client-Server Architectures.

Definition of Distributed Systems

It is a collection of independent **computers/nodes/components** that appear to the users of the system as a single coherent system. **Users (people or programs)** think they are dealing with a single system. It is a model in which computers on network communicate and coordinate their actions by passing messages. The components interact with each other in order to achieve a common goal.

A computer program that runs in a distributed system is called a distributed program, and distributed programming is the process of writing such programs. There are many alternatives for the message passing mechanism, including pure HTTP, RPC-like connectors and message queues.

Organization of a Distributed System



A distributed system organized as middleware.
Note that the middleware layer extends over multiple machines.

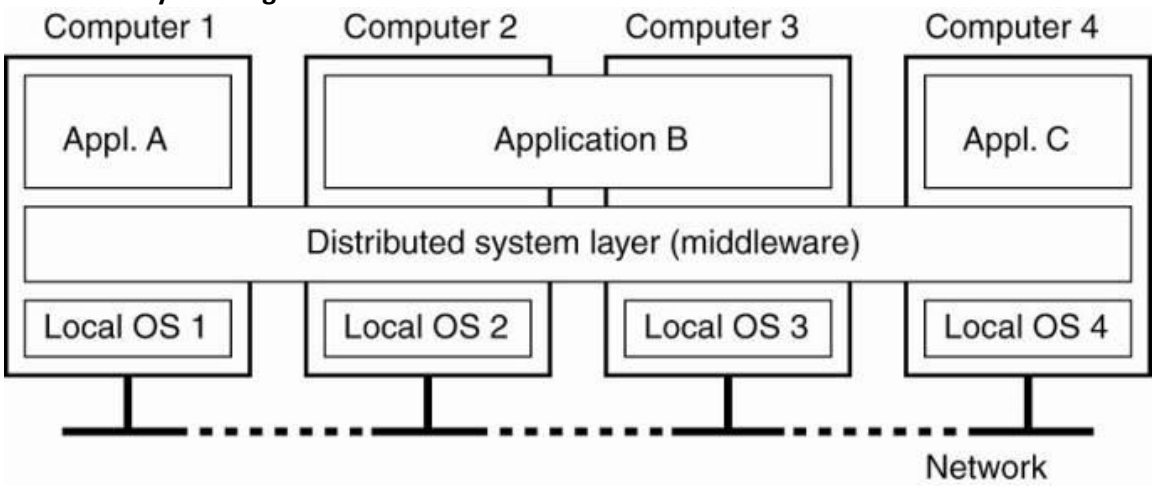
Characteristics of distributed systems

1. differences between various computers and the ways in which they communicate are mostly hidden from users
2. users and applications can interact with a distributed system in a consistent and uniform way, regardless of where and when interaction takes place.
3. distributed systems should also be relatively easy to expand or scale.
4. a distributed system will normally be continuously available

Note

Layers of software support heterogeneous computers and networks while offering a single-system view - sometimes called middleware.

Distributed system organized as middleware



Four networked computers and three applications

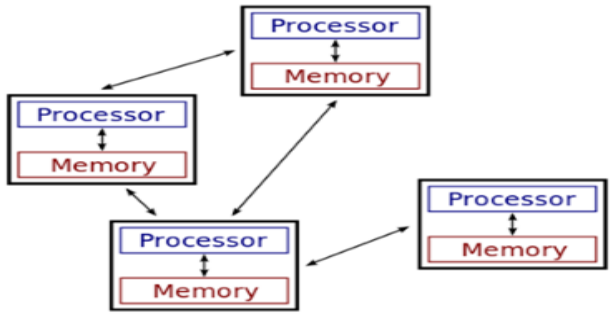
1. Application B is distributed across computers 2 and 3.
2. Each application is offered the same interface.
3. Distributed system provides the means for components of a single distributed application to communicate with each other, but also to let different applications communicate.
4. It also hides the differences in hardware and operating systems from each application.

Examples

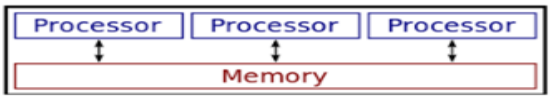
1. The World Wide Web – information, resource sharing
2. Clusters, Network of workstations
3. Distributed manufacturing system (e.g., automated assembly line)
4. Network of branch office computers - Information system to handle automatic processing of orders
5. Network of embedded systems
6. New Cell processor (PlayStation 3)

Advantages	Disadvantages
Economics, Speed, Inherent distribution, Reliability, Incremental growth	Software, Network, More components to fail Security

In parallel computing, all processors may have access to a shared memory to exchange information between processors. In distributed computing, each processor has its own private memory (distributed memory). Information is exchanged by passing messages between the processors.



Distributed System



Parallel System

Goals

1. Connecting Users and Resources.
2. Transparency.
3. Openness.
4. Scalability.

Connecting Users and Resources (Making Resources Accessible)

1. Distributed system Make it easy for the users (and applications) to access remote resources
2. Distributed system to share them in a controlled and efficient way.

Resources - anything: printers, computers, storage facilities, data, files, Web pages, and networks, etc.

Accessibility Issues

1. Security.
2. Unwanted communication.

Transparency

Goal - hide the fact that its processes and resources are physically distributed across multiple computers systems should be transparent

Different forms of transparency in a distributed system (ISO, 1995).

Transparency	Description
Access	Hide differences in data representation and how a resource is accessed
Location	Hide where a resource is located
Migration	Hide that a resource may move to another location
Relocation	Hide that a resource may be moved to another location while in use
Replication	Hide that a resource is replicated
Concurrency	Hide that a resource may be shared by several competitive users
Failure	Hide the failure and recovery of a resource

Degree of Transparency Issues

Timing

e.g. requesting an electronic newspaper to appear in your mailbox before 7 A.M. local time, as usual, while you are currently at the other end of the world living in a different time zone.

Synchronization

e.g. a wide-area distributed system that connects a process in San Francisco to a process in Amsterdam limited by laws of physics - a message sent from one process to the other takes about 35 milliseconds.

· It takes several hundreds of milliseconds using a computer network.
· Signal transmission is not only limited by the speed of light, but also by limited processing capacities of the intermediate switches.

Performance
e.g. many Internet applications repeatedly try to contact a server before finally giving up. Consequently, attempting to mask a transient server failure before trying another one may slow down the system as a whole.

Consistency
e.g. need to guarantee that several replicas, located on different continents, need to be consistent all the time - a single update operation may now even take seconds to complete, something that cannot be hidden from users.

Context Awareness
e.g. notion of location and context awareness is becoming increasingly important, it may be best to actually expose distribution rather than trying to hide it. - consider an office worker who wants to print a file from her notebook computer. It is better to send the print job to a busy nearby printer, rather than to an idle one at corporate headquarters in a different country.

Limits of Possibility
Recognizing that full distribution transparency is simply impossible, we should ask ourselves whether it is even wise to pretend that we can achieve it.

Openness
Goal: offer services according to standard rules that describe the syntax and semantics of those services.
e.g.
1. Computer networks - standard rules govern the format, contents, and meaning of messages sent and received.
2. Distributed systems - services are specified through interfaces, which are often described in an

Interface Definition Language (IDL)

- Interface definitions written in an IDL nearly always capture only the syntax of services

3. Specify names of the available functions with types of parameters; return values, possible exceptions that can be raised, etc.
4. Allows an arbitrary process that needs a certain interface to talk to another process that provides that interface
5. Allows two independent parties to build completely different implementations of those interfaces, leading to two separate distributed systems that operate in exactly the same way.

Properties of specifications
Complete - everything that is necessary to make an implementation has been specified.

Neutral
Specifications do not prescribe what an implementation should look like Lead to:

Interoperability - characterizes the extent by which two implementations of systems or components from different manufacturers can co-exist and work together by merely relying on each other's services as specified by a common standard.

Portability characterizes to what extent an application developed for a distributed system A can be executed, without modification, on a different distributed system B that implements the same interfaces as A.

Goals: an open distributed system should also be extensible. i.e.
1. be easy to configure the system out of different components (possibly from different developers).
2. be easy to add new components or replace existing ones without affecting those components that stay in place.

Scalability
Scalability of a system is measured with respect to:
1. Size - can easily add more users and resources to the system.
2. Geographic extent - a geographically scalable system is one in which the users and resources may lie far apart.
3. Administrative scalability - can be easy to manage even if it spans many independent administrative organizations.

Scalability Limitations of Size

Concept	Example
Centralized services	A single server for all users
Centralized data	A single on-line telephone book
Centralized algorithms	Doing routing based on complete information

Geographical scalability Limitations
Synchronization
e.g. currently hard to scale existing distributed systems designed for local-area networks is that they are based on synchronous communication.
1. A client requesting service blocks until a reply is sent back.
2. Works fine in LANs where communication between two machines is generally at worst a few hundred microseconds.
3. In a wide-area system, inter process communication may be hundreds of milliseconds, three orders of magnitude slower.

Unreliability of communication
1. Communication in wide-area networks is inherently unreliable and point-to-point.
2. local-area networks provide reliable communication based on broadcasting, making it much easier to develop distributed systems. For example, consider the problem of locating a service.
a. e.g. in a local-area system, a process can broadcast a message to every machine, asking if it is running the service it needs.
b. Only those machines that have that service respond, each providing its network address in the reply message.

c. Such a location scheme is unthinkable in a wide-area system: just imagine what would happen if we tried to locate a service this way in the Internet.

Administrative scalability

1. How to scale a distributed system across multiple, independent administrative domains.

a. Major problem - conflicting policies with respect to resource usage (and payment), management, and security.

Scaling Techniques

Three techniques for scaling:

1. Hiding communication latencies. 2. Distribution. 3. Replication.

Hiding communication latencies - important to achieving geographical scalability.

1. Try to avoid waiting for responses to remote service requests.

•e.g, when a service has been requested at a remote machine, an alternative to waiting for a reply from the server is to do other useful work at the requester's side.

•construct the requesting application in such a way that it uses only asynchronous communication.]

2. Reduce the overall communication

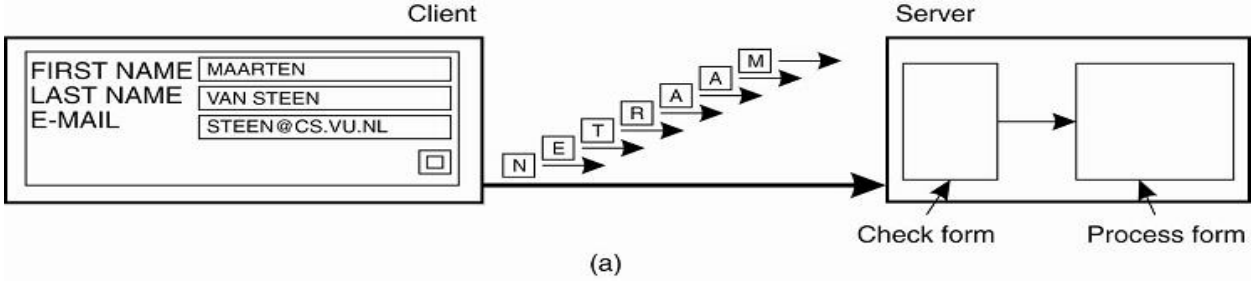
•e.g. in interactive applications when a user sends a request he will generally have nothing better to do than to wait for the answer.

•move part of the computation that is normally done at the server to the client process requesting the service.

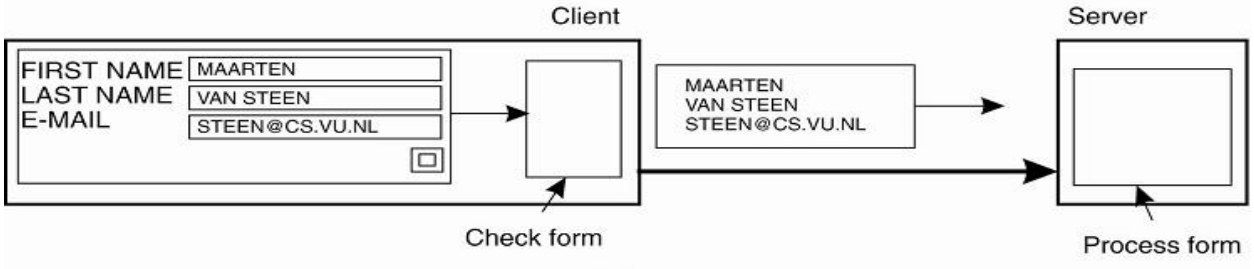
○ typical case - accessing databases using forms.

▪ship the code for filling in the form, and possibly checking the entries, to the client, and have the client return a completed form - approach of shipping code is now widely supported by the Web in the form of Java applets and JavaScript.

The difference between letting (a) a server or (b) a client check forms as they are being filled.



(a)

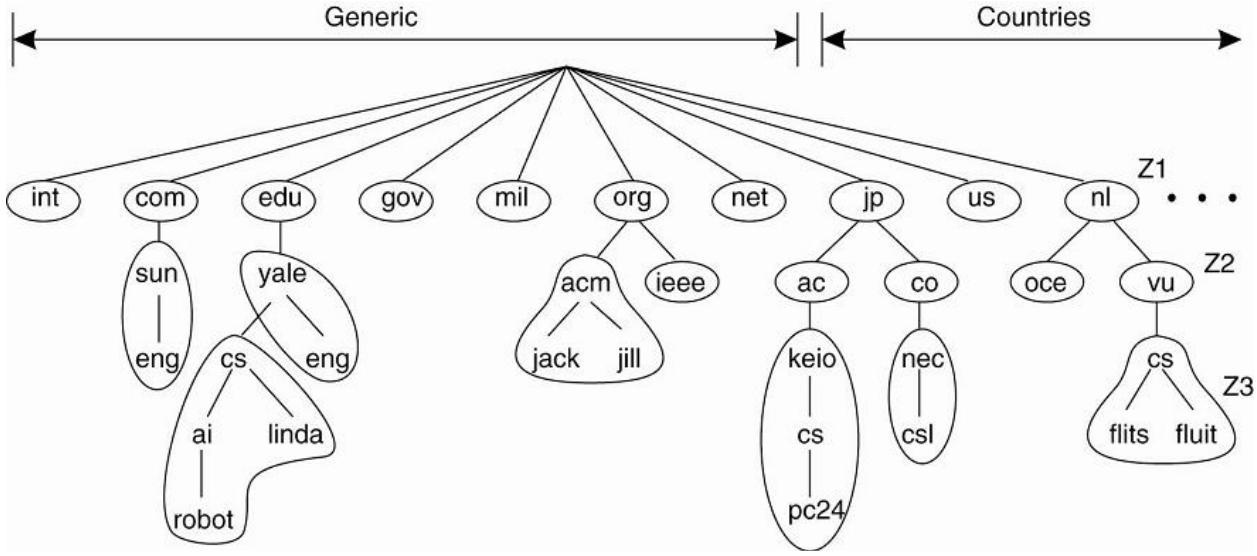


(b)

Distribution

1. Splitting a component into smaller parts and spreading those parts across the system.
· e.g. Internet Domain Name System (DNS).
2. The DNS name space is hierarchically organized into a tree of domains, which are divided into nonoverlapping zones
3. Names in each zone are handled by a single name server.
4. Resolving a name means returning the network address of the associated host.
e.g. the name nl.vu.cs.flits.
5. To resolve this name - first passed to the server of zone which returns the address of the server for zone Z2, to which the rest of name, vu.cs.flits, can be handed. The server for Z2 will return the address of the server for zone Z3, which is capable of handling the last part of the name and will return the address of the associated host.

An example of dividing the DNS name space into zones.



·DNS is distributed across several machines, thus avoiding that a single server has to deal with all requests for name resolution.
·Performance degradation Problems
Solution: replicate components across a distributed system

Replication

1. Increases availability
2. Helps balance the load between components leading to better performance.
3. e.g. in geographically widely-dispersed systems - a copy nearby can hide much of the communication latency problems.

Caching - special form of replication

1. Caching results in making a copy of a resource, generally in the proximity of the client accessing that resource.
2. Caching is a decision made by the client of a resource, and not by the owner of a resource.
3. Caching happens on demand whereas replication is often planned in advance.

Issues of caching and replication - multiple copies of a resource -> modifying one copy makes that copy different from the others -> leads to consistency problems.

Weak consistency

e.g. a cached Web document of which the validity has not been checked for the last few minutes.

Strong consistency

e.g. electronic stock exchanges and auctions.

Problem

1. An update must be immediately propagated to all other copies.
2. If two updates happen concurrently, it is often also required that each copy is updated in the same order.
3. Generally requires some global synchronization mechanism – hard to implement in a scalable way (i.e. speed of light

Pitfalls

False assumptions that everyone makes when developing a distributed application for the first time (by Peter Deutsch):

1. The network is reliable.	2. The network is secure.	3. The network is homogeneous.
4. The topology don't change.	5. Latency is zero.	6. Bandwidth is infinite.
7. Transport cost is zero.	8. There is one administrator.	

Hardware Concepts

1. Multiprocessors.
2. Homogeneous Multicomputer systems.
3. Heterogeneous Multicomputer systems.

Hardware Concepts

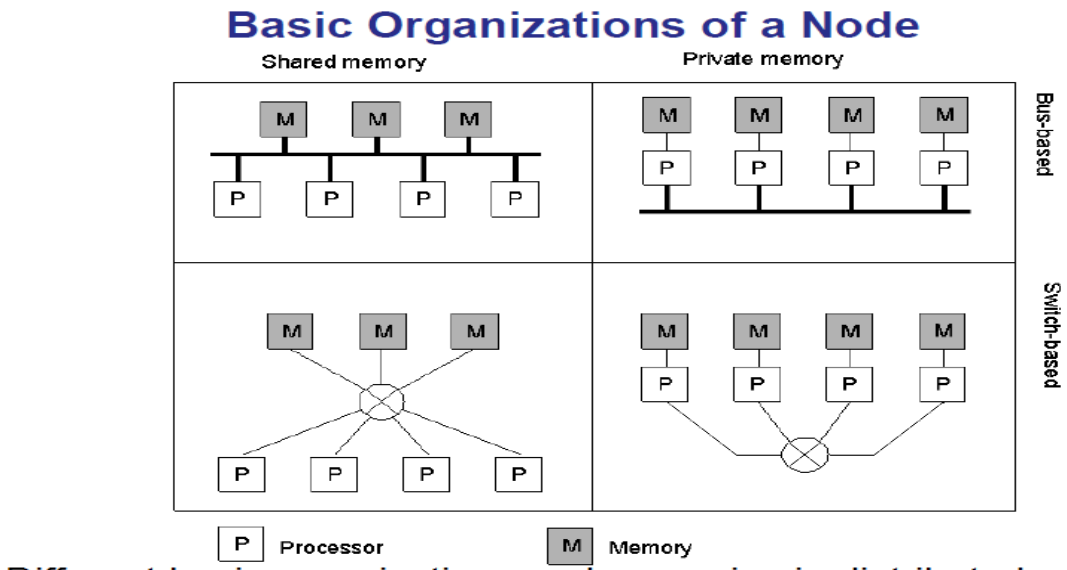
1. Characteristics which affect the behavior of software systems
2. The platform ....

a. the individual nodes (”computer”, ”processor”)
b. communication between two nodes
c. organization of the system (network of nodes)

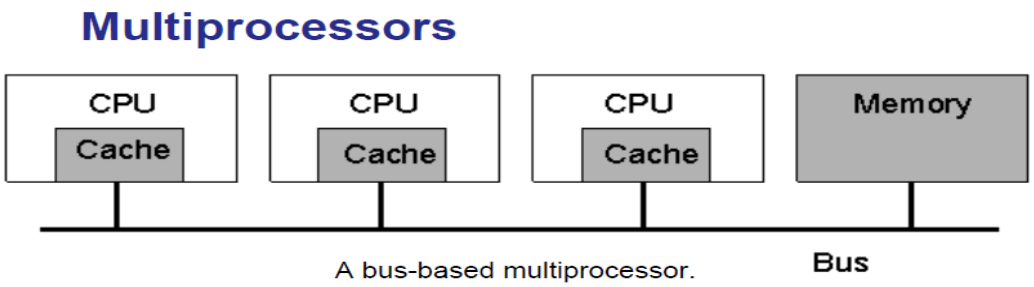
3. ... and its characteristics

a. capacity of nodes
b. capacity (throughput, delay) of communication links
c. reliability of communication (and of the nodes)

4. Which ways to distribute an application are feasible



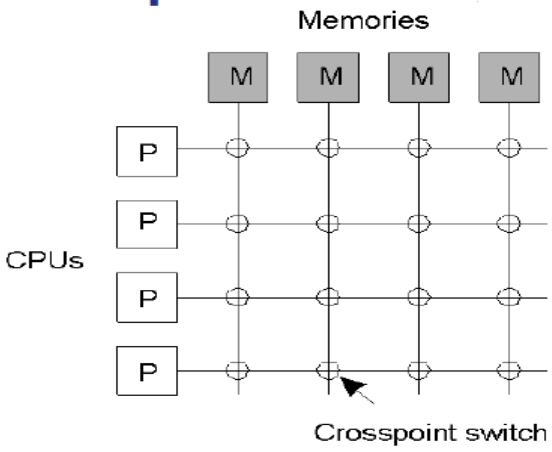
Different basic organizations and memories in distributed computer systems

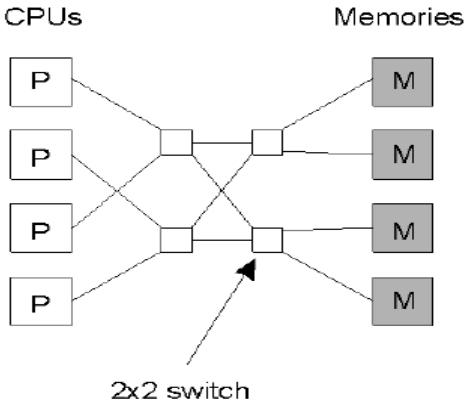


A bus-based multiprocessor

Cache memory, hit rate, coherence, write-through cache, snoopy cache

Multiprocessors

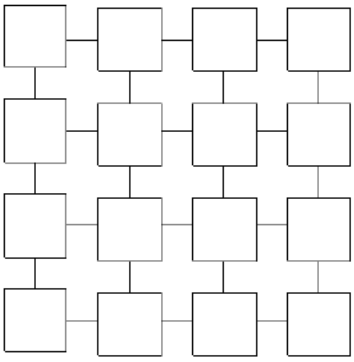




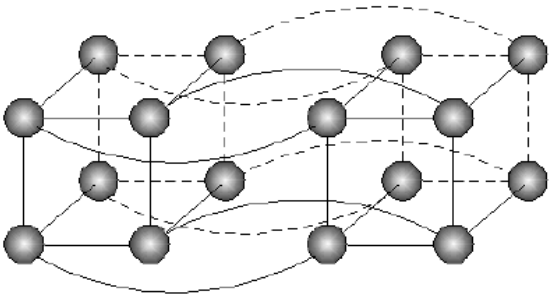
a) A crossbar switch

b) An omega switching network

Homogeneous Multicomputer Systems



a) Grid



b) Hypercube

A new design aspect: locality at the network level

General Multicomputer Systems

Loosely connected systems	Application architectures
a. Nodes: autonomous b. communication: slow and vulnerable c. cooperation at ”service level”	a. multiprocessor systems: parallel computation b. multicomputer systems: distributed systems c. (how are parallel, concurrent, and distributed systems different?)

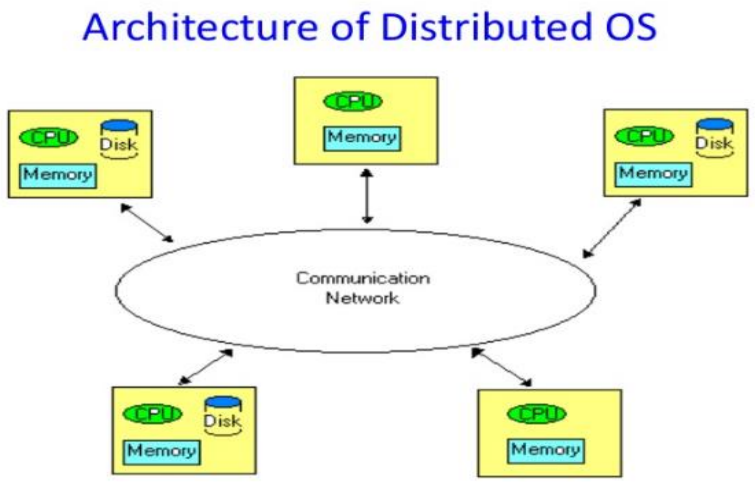
Software Concepts

System	Description	Main Goal

Distributed Operating Systems	Tightly-coupled operating system for multiprocessors and homogeneous multi computers	Hide and manage hardware resources
Network Operating Systems	Loosely-coupled operating system for heterogeneous multi computers (LAN & WAN)	Offer local services to remote clients
Middleware	Additional layer atop of NOS implementing general-purpose services	Provide distribution transparency

Distributed Operating Systems

It is software over a collection of independent, networked, communicating, and physically separate computational nodes. Each individual node holds a specific software subset of the global aggregate operating system.

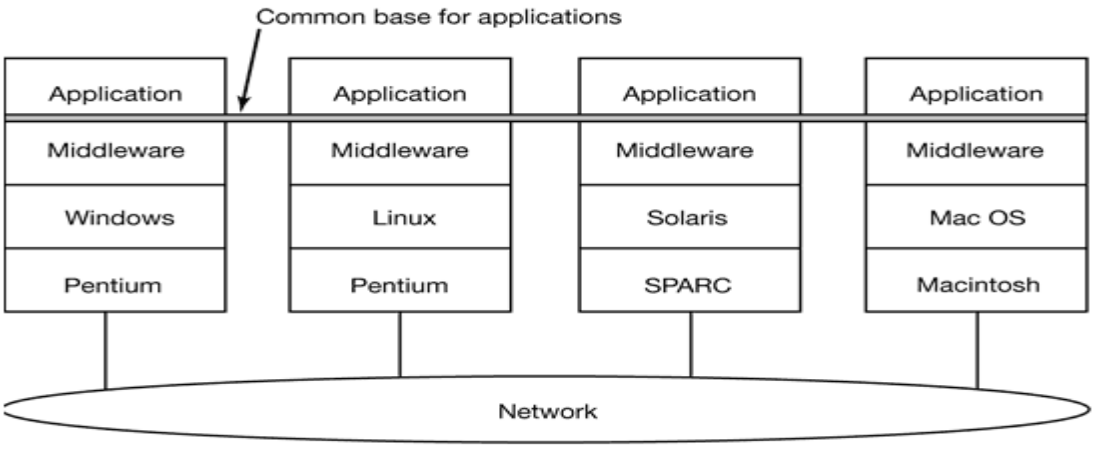


Network Operating Systems

An operating system oriented to computer networking, to allow shared file and printer access among multiple computers in a network, to enable the sharing of data, users, groups, security, applications, and other networking functions. Typically over a local area network (LAN), or private network.

Middleware

It includes web servers, application servers, messaging and similar tools that support application development and delivery. Middleware sits "in the middle" between application software that may be working on different operating systems. The distinction between operating system and middleware functionality is, to some extent, arbitrary. While core kernel functionality can only be provided by the operating system itself, some functionality previously provided by separately sold middleware is now integrated in operating systems.



The client-server model

1. Clients and Servers. 2. Application Layering. 3. Client-Server Architectures.

Client-server model

The client–server model is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients.

Application Layering

It is a layer in the Open Systems Interconnection (OSI) seven-layer model and in the TCP/IP protocol suite. It consists of protocols that focus on process-to-process communication across an IP network and provides a firm communication interface and end-user services.

Open Systems Interconnection Layer Model

Layered Protocols

High level 7→
Create message, 6 → string of bits
Establish Comm. 5→
Create packets 4→
Network routing 3→
Add header/footer tag + checksum 2→
Transmit bits via 1→ comm. medium (e.g. Copper, Fiber, wireless)

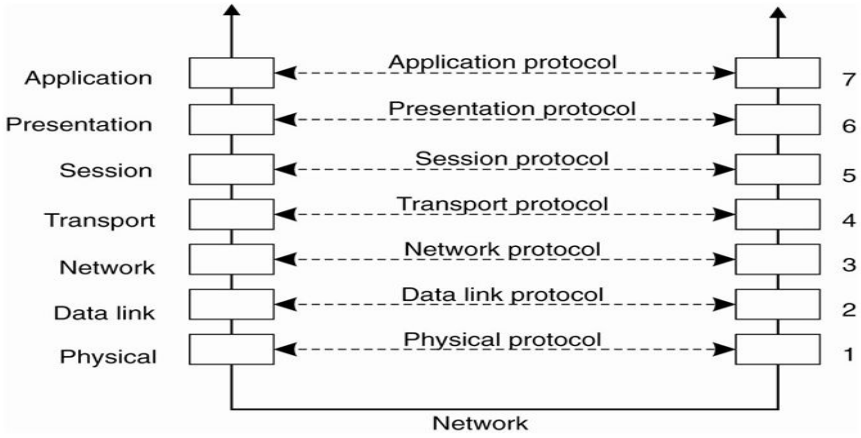
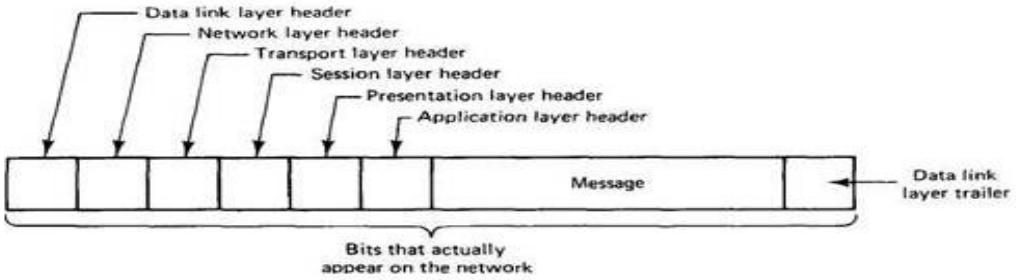


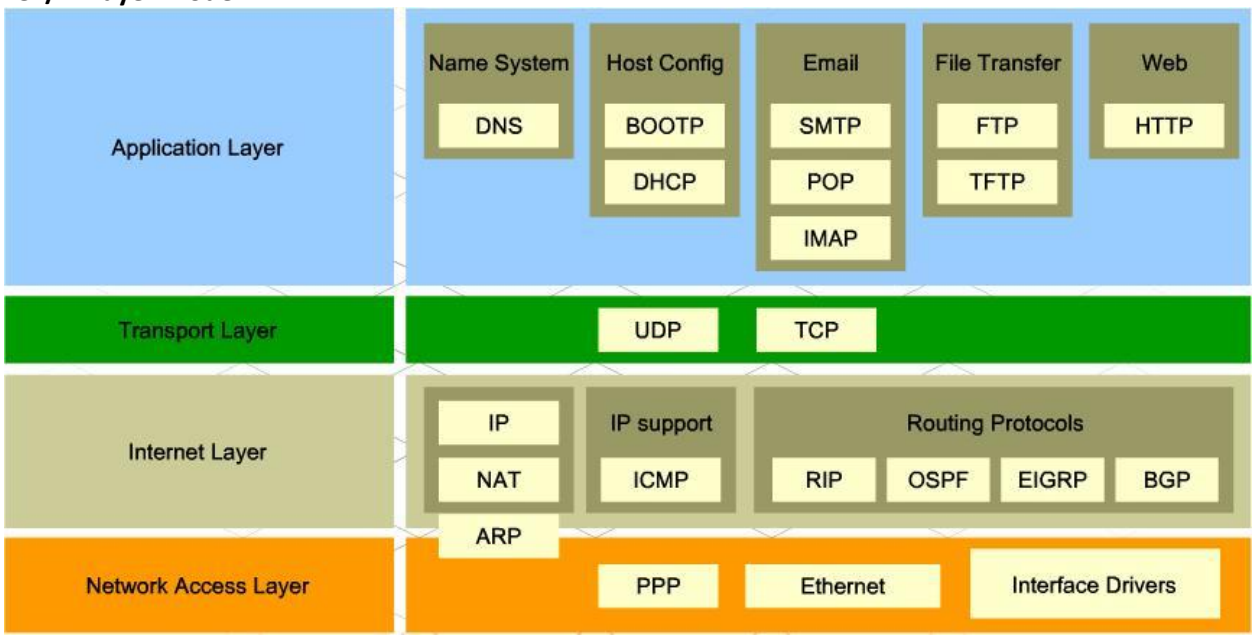
Figure 4-1. Layers, interfaces, and protocols in the OSI model.

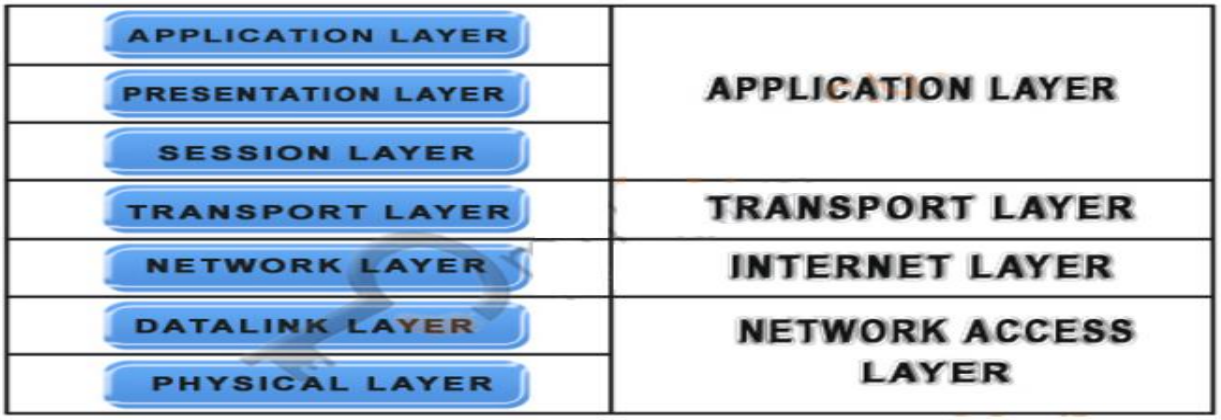
OSI Model

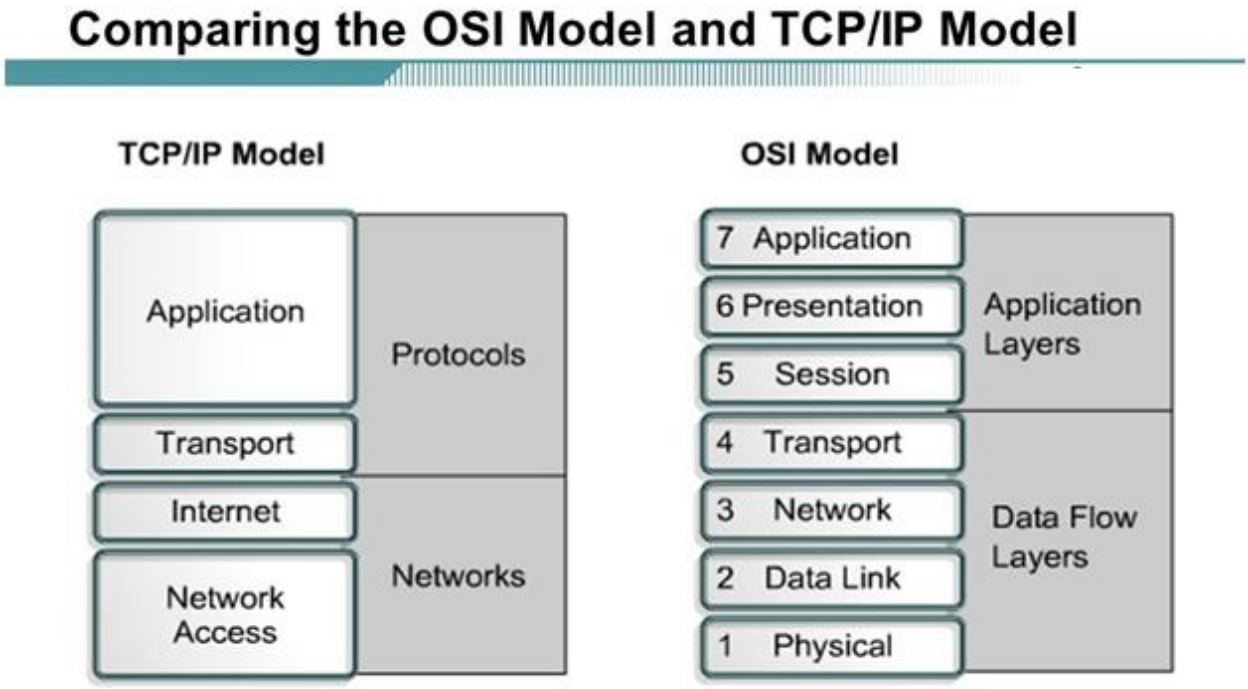
Layer	Function	Examples
Application (Layer 7)	User interface	Telnet, HTTP
Presentation (Layer 6)	Handles encryption & changes to syntax	ASCII/EBCDIC, JPEG/MP3
Session (Layer 5)	Manages multiple applications and maintains synchronisation points	Operating systems, scheduling
Transport (Layer 4)	Provides reliable or best-effort delivery and (optional) error and flow control	TCP, UDP
Network (Layer 3)	Provides logical end-to-end addressing used by routers and hosts	IP
Data Link (Layer 2)	Creates frames from data bits, uses MAC addresses to access endpoints, and provides error detection but no correction	802.3, 802.2, HDLC, Frame Relay
Physical (Layer 1)	Specifies voltage, wire speed, and cable pin-outs	EIA/TIA, V.35



TCP/IP layer Model





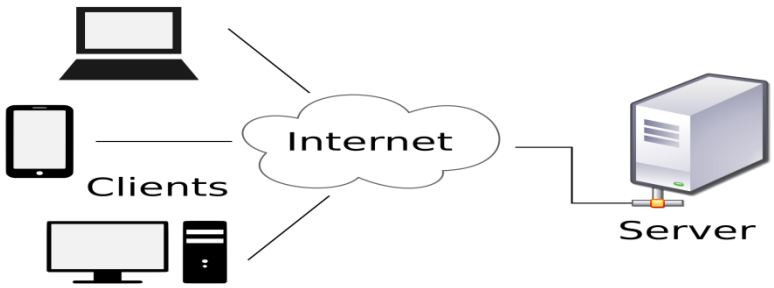


Client-Server Architectures

Various hardware and software architectures are used for distributed computing. At a lower level, it is necessary to interconnect multiple CPUs with some sort of network, regardless of whether that network is printed onto a circuit board or made up of loosely coupled devices and cables. At a higher level, it is necessary to interconnect processes running on those CPUs with some sort of communication system. Distributed programming typically falls into one of several basic architectures: client–server, three-tier, n-tier, or peer-to-peer; or categories: loose coupling, or tight coupling.

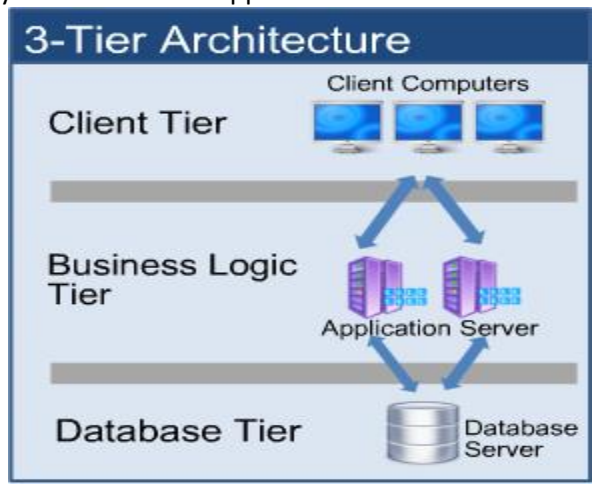
Client–server

Architectures where smart clients contact the server for data then format and display it to the users. Input at the client is committed back to the server when it represents a permanent change.



Three-tier

Architectures that move the client intelligence to middle tier so that stateless clients can be used. This simplifies application deployment. Most web applications are three-tier.



n-tier

Architectures that refer typically to web applications which further forward their requests to other enterprise services. This type of application is the one most responsible for the success of application servers.

Peer-to-peer

Architectures where there is no special machines that provide a service or manage the network resources. Instead all responsibilities are uniformly divided among all machines, known as peers. Peers can serve both as clients and as servers.