**B.E 4/4 IT I SEM (A, B) Internal -I**

Subject : Distributed Systems Subject Code: BIT-406

Academic Year: 2016-2017 Date:

**Course Outcome Mapping Table**

|  |  |  |
| --- | --- | --- |
| **Unit** | **Course Out Come** | **Question** |
| 1 | 1 | 1,2,4a,5a,6a |
| 2 | 2 | 3,4b,6b |
| 3 | 3 | - |
| 4 | 4 | - |
| 5 | 5 | - |

**Part – A**

**Answer the following questions (2\*3=6)**

1. What is a distributed System?

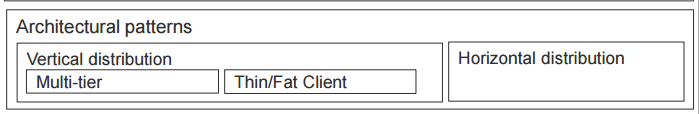
It is a collection of independent **computers/nodes/components** that appear to the users of the system as a single coherent system. **Users (people or programs)** think they are dealing with a single system.

It is a model in which computers on [network](https://en.wikipedia.org/wiki/Computer_network) communicate and coordinate their actions by [passing messages](https://en.wikipedia.org/wiki/Message_passing). The components interact with each other in order to achieve a common goal.

2. What is open distributed system?

An **Open Distributed System** is made up of components that may be obtained from a number of different sources, which together work as a single **distributed system**. In 1988 the International Standards Organization (ISO) began work on preparing standards for **Open Distributed** Processing (ODP).

3. What is vertical RPC?



**Part – B**

**Answer any two questions (2\*7=14)**

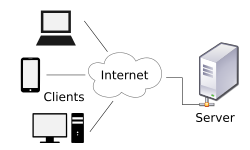
4. a. Explain about Client Server Models?

**The client-server model**

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

**Client-server model**

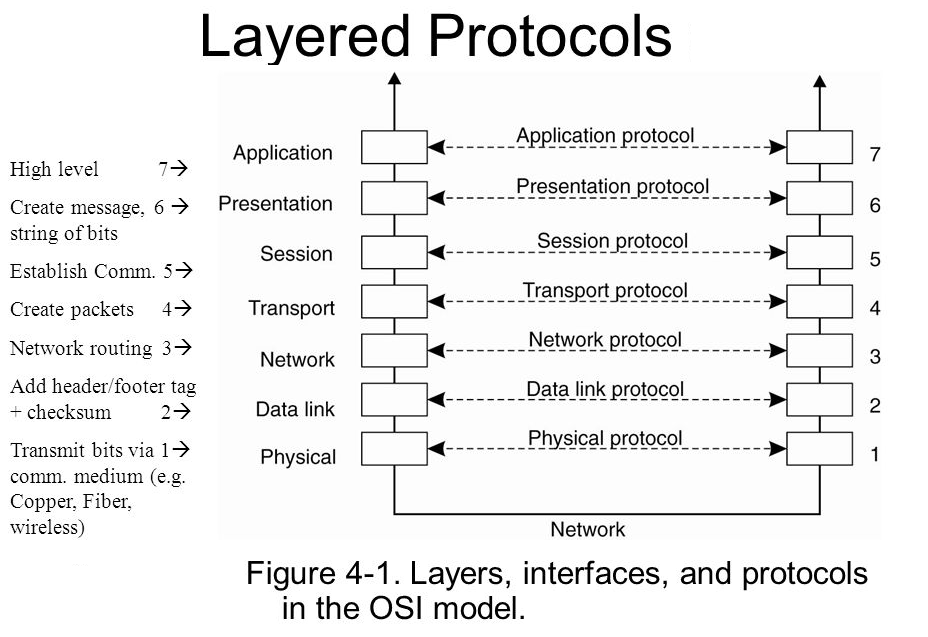
The client–server model is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients.

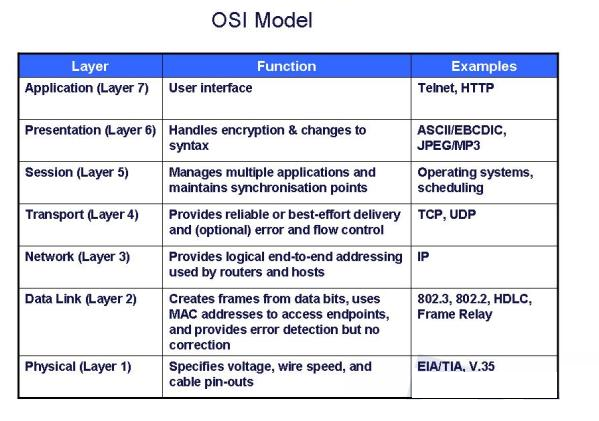


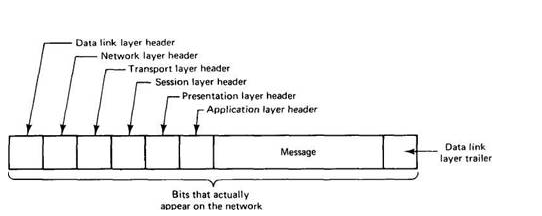
**Application Layering**

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.

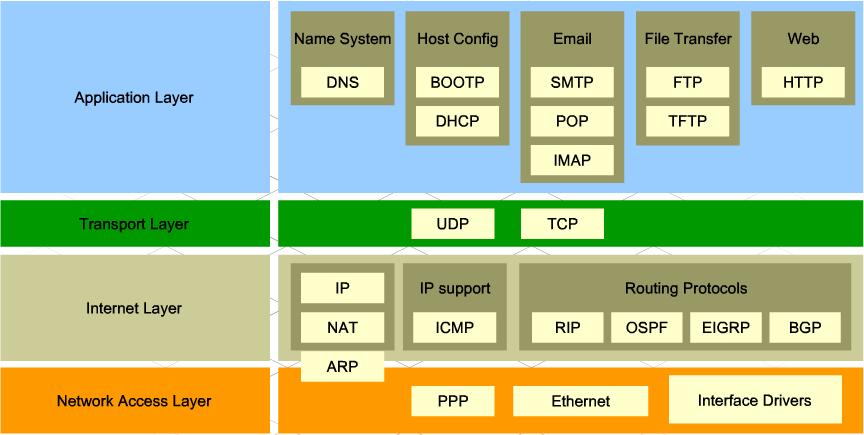
**Open Systems Interconnection Layer Model**

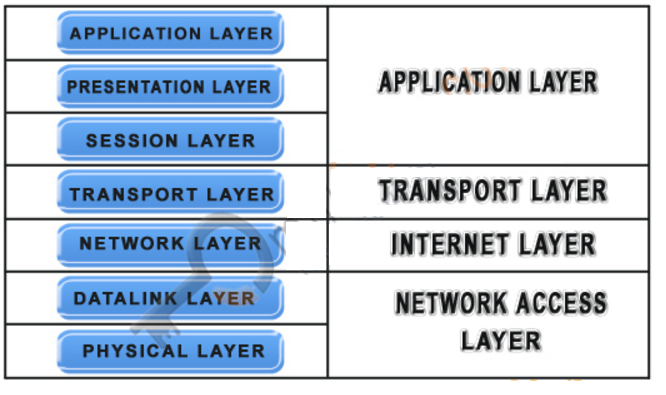


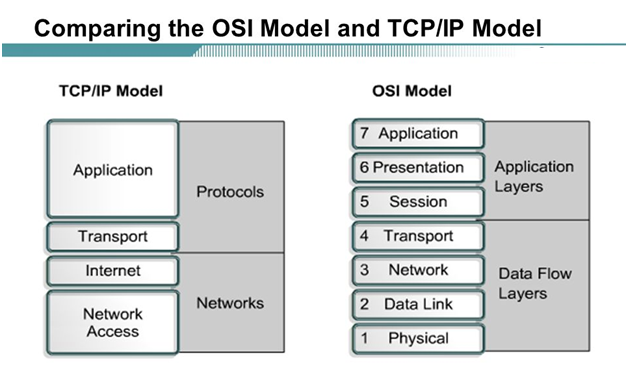




**TCP/IP layer Model**







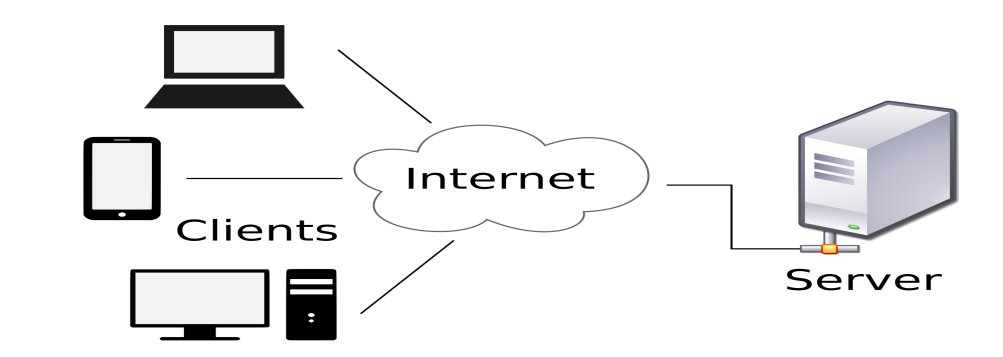
**Client-Server Architectures**

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](https://en.wikipedia.org/wiki/Process_(computing)) running on those CPUs with some sort of [communication system](https://en.wikipedia.org/wiki/Communication_system).

Distributed programming typically falls into one of several basic architectures: [client–server](https://en.wikipedia.org/wiki/Client%E2%80%93server), [three-tier](https://en.wikipedia.org/wiki/Three-tier_(computing)), [n-tier](https://en.wikipedia.org/wiki/Multitier_architecture), or [peer-to-peer](https://en.wikipedia.org/wiki/Peer-to-peer); or categories: [loose coupling](https://en.wikipedia.org/wiki/Loose_coupling), or [tight coupling](https://en.wikipedia.org/wiki/Computer_cluster).

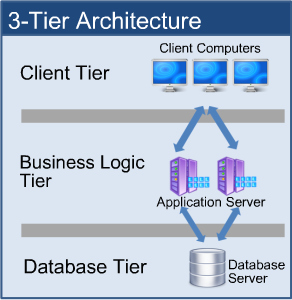
[**Client–server**](https://en.wikipedia.org/wiki/Client%E2%80%93server)

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**](https://en.wikipedia.org/wiki/Three-tier_(computing))

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.

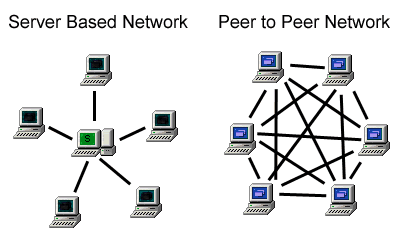


[**n-tier**](https://en.wikipedia.org/wiki/Multitier_architecture)

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](https://en.wikipedia.org/wiki/Application_server).

[**Peer-to-peer**](https://en.wikipedia.org/wiki/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.



b. Explain about Lower and higher level protocols?

**Lower level Protocols (Device to Device).**

The lowest protocol always deals with "low-level", physical interaction of the hardware. Every higher layer adds more features.

**a. IP (Internet Protocol)** (The Address of the Machine)).

**b. TCP (Transmission Control Protocol) (**Proof of Delivery, rules or reassembling partitioned messages)).

c. Implementation in physical layer and data link layer of the stack. Group data bits into frames and adds a pattern called checksums at either end of frame.

5. a. Explain about the goals of a distributed 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 |

<http://csis.pace.edu/~marchese/CS865/Lectures/Chap1/Chapter1a.htm>

**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

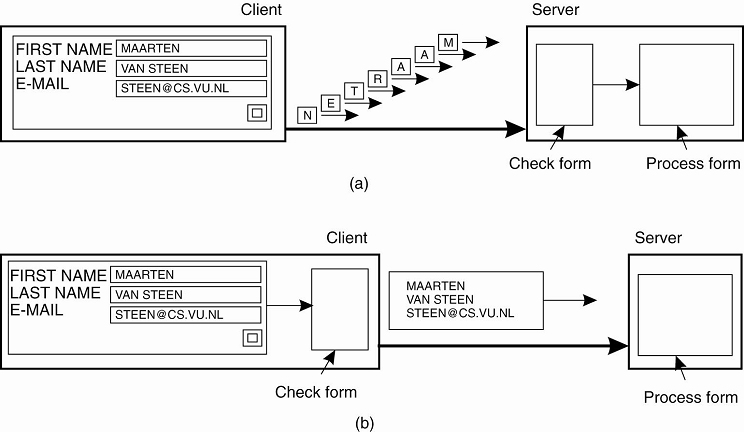
e.g. in interactive applications when a user sends a request he will generally have nothing better to do than to wait for the answer.

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

O 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.



6. a. Explain about hardware concepts.

**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)

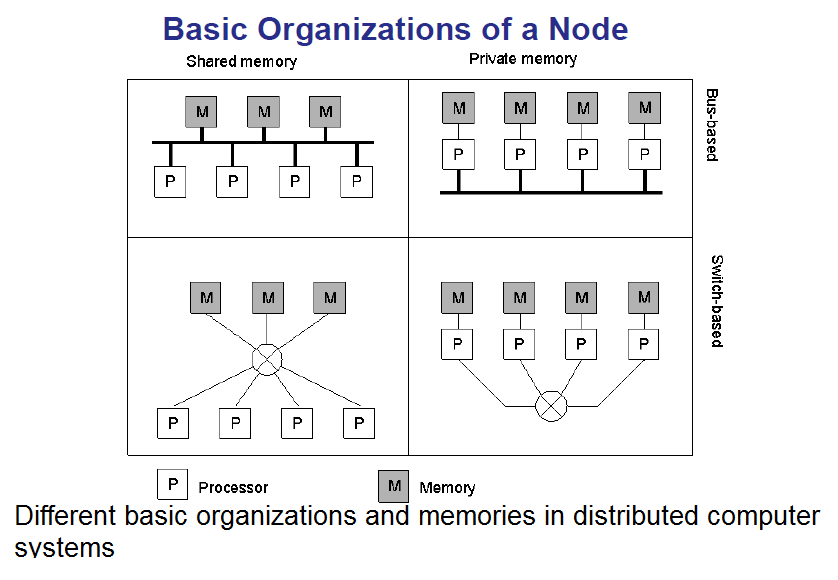
3. ... and its characteristics

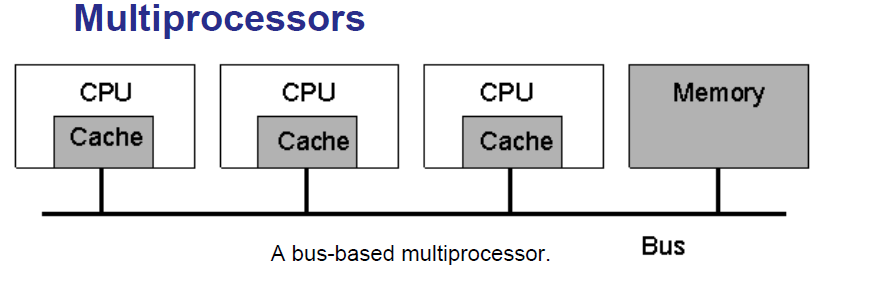
a. capacity of nodes

b. capacity (throughput, delay) of communication links

c. reliability of communication (and of the nodes)

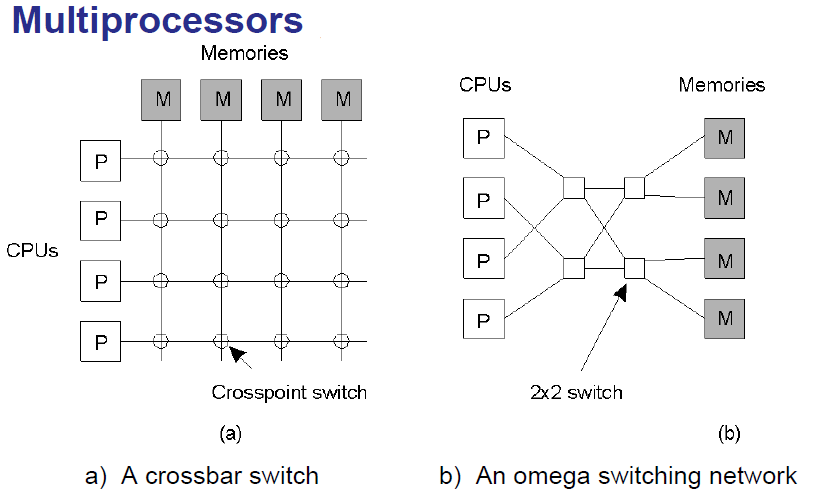
4. Which ways to distribute an application are feasible

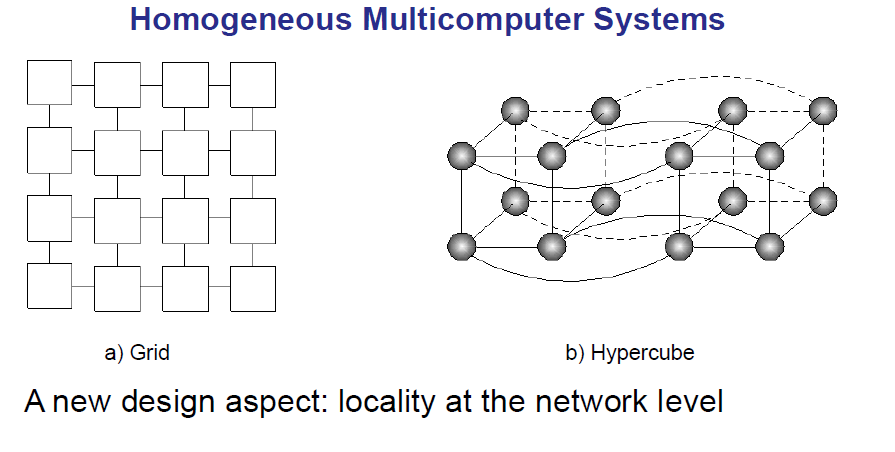




**A bus-based multiprocessor**

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





**General Multicomputer Systems**

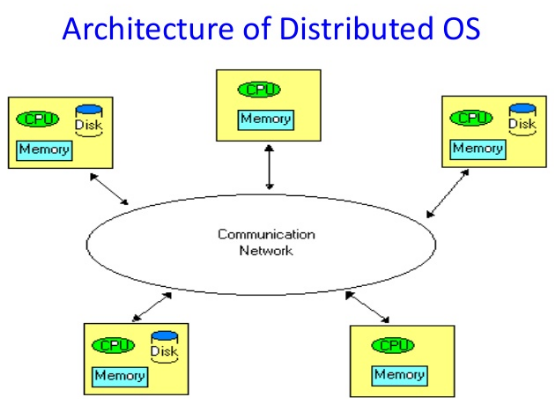
|  |  |
| --- | --- |
| **Loosely connected systems** | **Application architectures** |
| a. Nodes: autonomous  b. communication: slow and vulnerable  c. cooperation at ”service level” | a. multiprocessor systems: parallel computation  b. multicomputer systems: distributed systems  c. (how are parallel,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

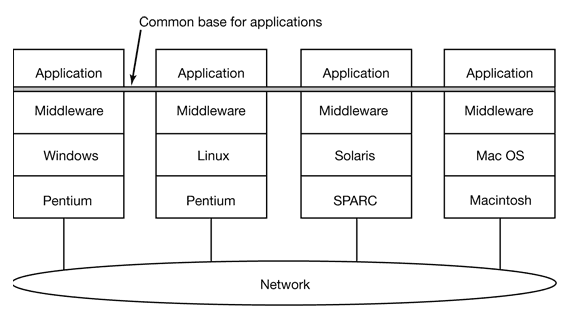


**Network Operating Systems**

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](https://en.wikipedia.org/wiki/Web_server), [application servers](https://en.wikipedia.org/wiki/Application_server), messaging and similar tools that support application development and delivery. Middleware sits "in the middle" between [application software](https://en.wikipedia.org/wiki/Application_software) that may be working on different [operating systems](https://en.wikipedia.org/wiki/Operating_system). 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.

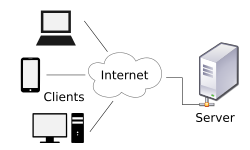


**The client-server model**

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

**Client-server model**

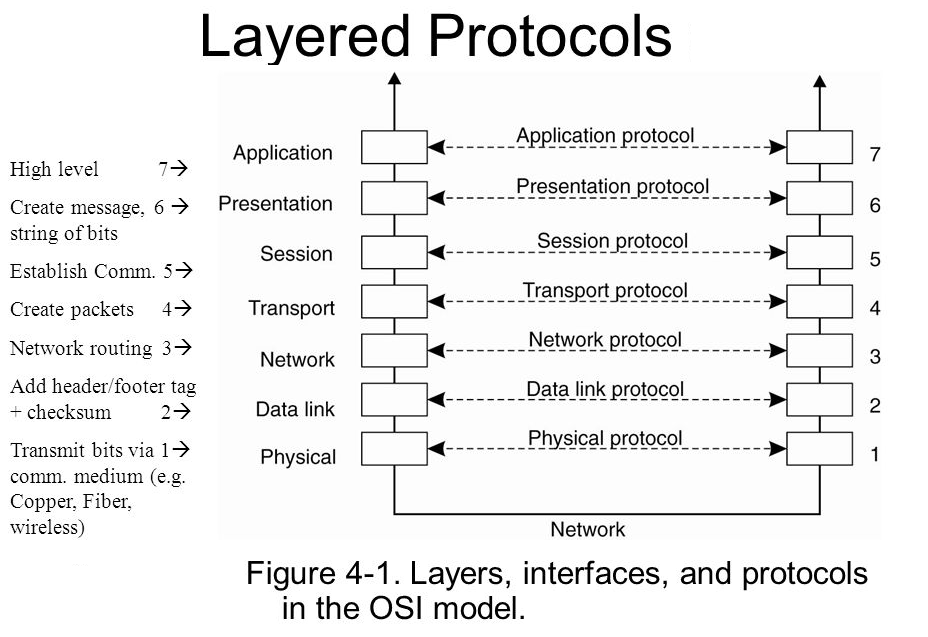
The client–server model is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients.

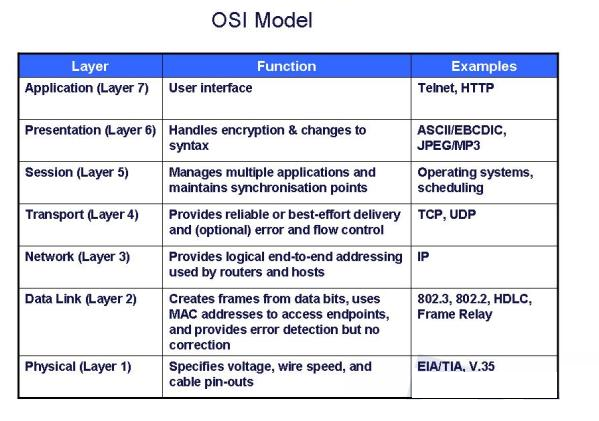


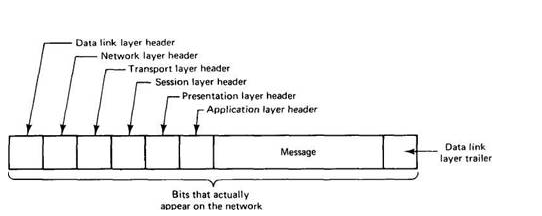
**Application Layering**

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.

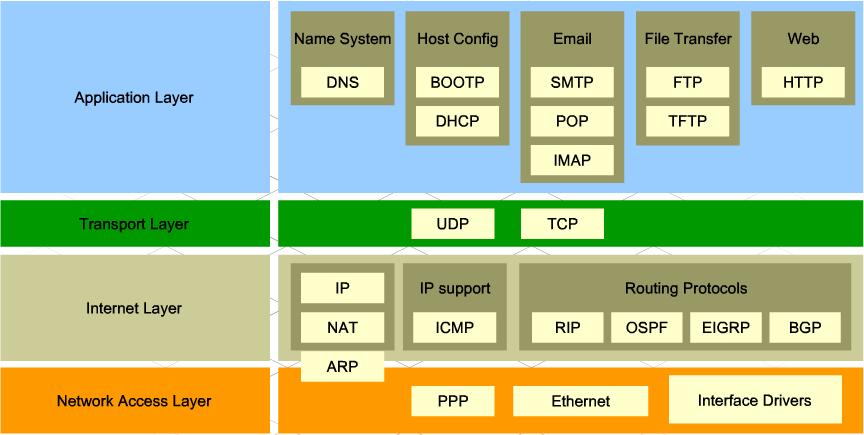
**Open Systems Interconnection Layer Model**

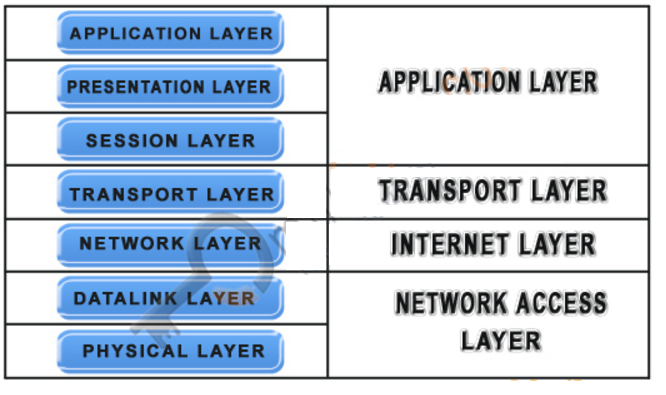


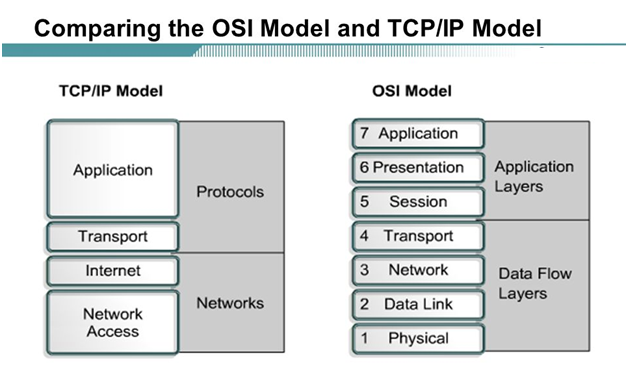




**TCP/IP layer Model**







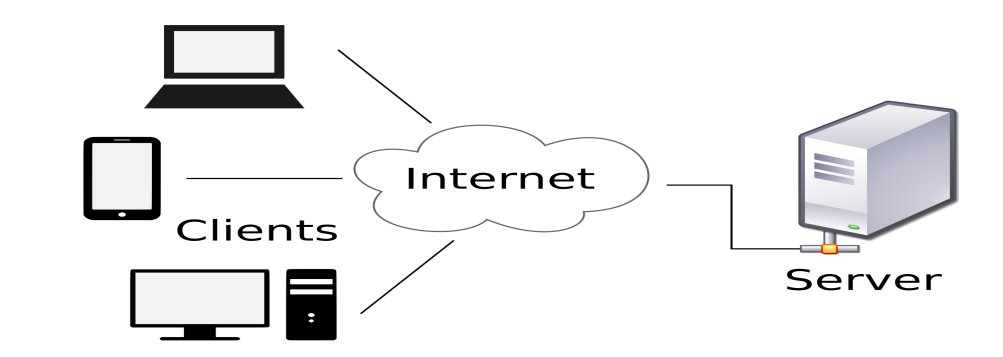
**Client-Server Architectures**

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](https://en.wikipedia.org/wiki/Process_(computing)) running on those CPUs with some sort of [communication system](https://en.wikipedia.org/wiki/Communication_system).

Distributed programming typically falls into one of several basic architectures: [client–server](https://en.wikipedia.org/wiki/Client%E2%80%93server), [three-tier](https://en.wikipedia.org/wiki/Three-tier_(computing)), [n-tier](https://en.wikipedia.org/wiki/Multitier_architecture), or [peer-to-peer](https://en.wikipedia.org/wiki/Peer-to-peer); or categories: [loose coupling](https://en.wikipedia.org/wiki/Loose_coupling), or [tight coupling](https://en.wikipedia.org/wiki/Computer_cluster).

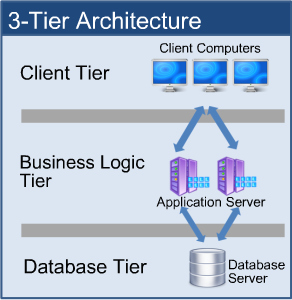
[**Client–server**](https://en.wikipedia.org/wiki/Client%E2%80%93server)

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**](https://en.wikipedia.org/wiki/Three-tier_(computing))

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.

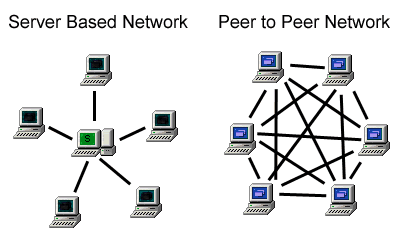


[**n-tier**](https://en.wikipedia.org/wiki/Multitier_architecture)

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](https://en.wikipedia.org/wiki/Application_server).

[**Peer-to-peer**](https://en.wikipedia.org/wiki/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.



b. Explain about Remote Object Invocation.

**Remote Object Invocation**

RMI (**Remote Method Invocation**) is a way that a programmer, using the Java programming language and development environment, can write **object**-oriented programming in which **object**s on different computers can interact in a distributed network.

**Distributed objects** are objects (in the sense of [object-oriented programming](https://en.wikipedia.org/wiki/Object-oriented_programming)) that are distributed across different [address spaces](https://en.wikipedia.org/wiki/Address_space), either in multiple [computers](https://en.wikipedia.org/wiki/Computer) connected via a [network](https://en.wikipedia.org/wiki/Computer_network) or even in different [processes](https://en.wikipedia.org/wiki/Process_(computing)) on the same computer, but which work together by sharing data and invoking methods.

The main method of [distributed object communication](https://en.wikipedia.org/wiki/Distributed_object_communication) is with [remote method invocation](https://en.wikipedia.org/wiki/Remote_method_invocation), generally by message-passing: one object sends a message to another object in a remote machine or process to perform some task. The results are sent back to the calling object.

## Local vs. Distributed Objects

Local and distributed objects differ in many respects. Here are some of them:

1. **Life cycle** : Creation, migration and deletion of distributed objects is different from local objects
2. **Reference** : Remote references to distributed objects are more complex than simple pointers to memory addresses
3. **Request Latency** : A distributed object request is orders of magnitude slower than local method invocation
4. **Object Activation** : Distributed objects may not always be available to serve an object request at any point in time
5. **Parallelism**: Distributed objects may be executed in parallel.
6. **Communication** : There are different communication primitives available for distributed objects requests
7. **Failure**: Distributed objects have far more points of failure than typical local objects.
8. **Security**: Distribution makes them vulnerable to attack.

