Distributed Computing(CSC801)

BE (Computer Engineering)
Sem-VIII A.Y. 2023-24

Lokmanya Tilak Jankalyan Shikshan Sanstha's
Lokmanya Tilak College of Engineering
Sector 4, Vikas Nagar, Koparkhairane, Navi Mumbai - 400 709
University of Mumbai, Mumbai

Course objectives & Outcomes

Course objectives:

1. To provide students with contemporary knowledge in distributed systems
2. To equip students with skills to analyze and design distributed applications.
3. To provide master skills to measure the performance of distributed synchronization algorithms

Course objectives & Outcomes

Course outcomes: On successful completion of course learner will be able to:

- Demonstrate knowledge of the basic elements and concepts related to distributed system technologies;
- Illustrate the middleware technologies that support distributed applications such as RPC, RMI and Object based middleware.
- Analyze the various techniques used for clock synchronization and mutual exclusion and deadlock.
- Demonstrate the concepts of Resource and Process management and synchronization algorithms
- Demonstrate the concepts of Consistency, Replication Management and fault Tolerance.
- Apply the knowledge of Distributed File systems in building large-scale distributed applications.

Syllabus & Books

Syllabus :

Module 1: Introduction to Distributed Systems

Module 2: Communication

Module 3: Synchronization

Module 4: Resource and Process Management

Module 5: Consistency, Replication and Fault Tolerance

Module 6: Distributed File Systems

Books :

- Andrew S. Tanenbaum and Maarten Van Steen, Distributed Systems: Principles and Paradigms, 2nd edition, Pearson Education.
- Pradeep K.Sinha, "Distributed Operating System-Concepts and design", PHI.

Evaluation Scheme

- Internal Assessment (20 marks)

  Avg of Test 1 and Test 2 (Each test for 20 Marks)

- End Semesters Examination (80 marks)

- Distributed Computing Lab

  TW : 25 Marks and Oral : 25 Marks

Module 1: Introduction to Distributed Systems

1.1 Characterization of Distributed Systems: Issues, Goals, and Types of distributed systems, Distributed System Models, Hardware Concepts, Software Concept.

1.2 Middleware: Models of Middleware, Services offered by middleware, Client Server model.

Introduction

Definition :

A distributed system is a collection of autonomous computing elements that appears to its users as a single **coherent system**.

Examples : Internet, Intranet and Emerging Tech. Network

Characteristic features :

- Autonomous computing elements (independent computers) , also referred to as nodes, be they hardware devices or software processes.
- Single coherent system: users or applications perceive a single system => nodes need to collaborate

Coherent system

Essence :

The collection of nodes as a whole operates the same, no matter where, when and how interaction between a user and the system takes place.

Examples :

- An end user cannot tell where a computation is taking place
- Where data is exactly stored should be irrelevant to an application
- If or not data has been replicated is completely hidden

Collection of autonomous nodes

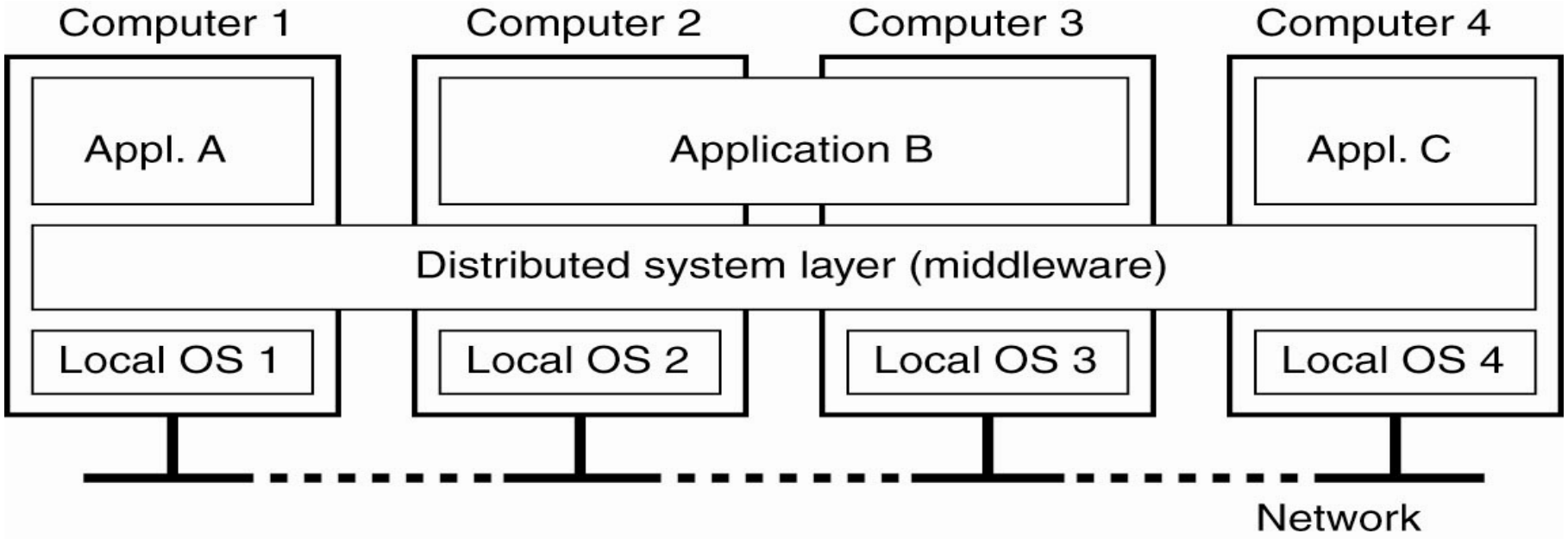
Independent behavior

- Each node is autonomous and will thus have its own notion of time: there is no global clock. Leads to fundamental synchronization and coordination problems.

Collection of nodes

- How to manage group membership?
- How to know that you are indeed communicating with an authorized (non) member?

Definition of a Distributed System (2)



- A distributed system organized as middleware. The middleware layer extends over multiple machines, and offers each application the same interface.
- Distributed system is used components and functions that need not be implemented by applications separately.

Design Goal of Distributed System

- Support sharing of resources
- Distribution transparency
- Openness
- Scalability

Sharing of resources

Typical resources

Printers, computers, storage facilities, data, files…

Why sharing?

Economics Collaboration, Information Exchange (groupware)

Examples

- Cloud-based shared storage and files
- Peer-to-peer assisted multimedia streaming
- Shared mail services (think of outsourced mail systems) Shared Web hosting (think of content distribution networks)

Distribution transparency

Transparent distributed system : Looks to its users as if it were only a single computer system

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
Persistence	Hide whether a (software) resource is in memory or on disk

Openness

- Interoperability: to what extend can work together
- Portability: to what extend an application developed for A can be executed on B that implements the same interface with A
- A system organized as a collection of relatively small and easily replaceable or adaptable components
- Provide definitions of interfaces to internal parts of the system as well

Separate Policy from Mechanism

A distributed system provides only mechanisms. Policies specified by applications and users

Example policies:

- What level of consistency do we require for client-cached data?
- Which operations do we allow downloaded code to perform?
- Which QoS requirements do we adjust in the face of varying bandwidth?
- What level of secrecy do we require for communication?

Policies versus mechanisms

Implementing openness: policies

- What level of consistency do we require for client-cached data?
- Which operations do we allow downloaded code to perform?
- Which QoS requirements do we adjust in the face of varying bandwidth?
- What level of secrecy do we require for communication?

Implementing openness: mechanisms

- Allow (dynamic) setting of caching policies
- Support different levels of trust for mobile code
- Provide adjustable QoS parameters per data stream
- Offer different encryption algorithms

Scalability

Root causes for scalability problems with centralized solutions ….

- The computational capacity, limited by the CPUs
- The storage capacity, including the transfer rate between CPUs and disks
- The network between the user and the centralized service

Examples of scalability limitations

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

Scalability

Observation :

Many developers of modern distributed systems easily use the adjective "scalable" without making clear why their system actually scales.

At least three components :

- Number of users and/or processes (size scalability)
- Maximum distance between nodes (geographical scalability)
- Number of administrative domains (administrative scalability)

Observation : Most systems account only, to a certain extent, for size scalability.

Often a solution: multiple powerful servers operating independently in parallel.

Today, the challenge still lies in **geographical and administrative scalability.**

Techniques for scaling

Three techniques:

- Hiding communication latencies
- Distribution
- Replication

Pitfalls when Developing Distributed Systems

False assumptions made by first time developer:

- The network is reliable.
- The network is secure.
- The network is homogeneous.
- The topology does not change.
- Latency is zero.
- Bandwidth is infinite.
- Transport cost is zero.
- There is one administrator

Types of Distributed Systems

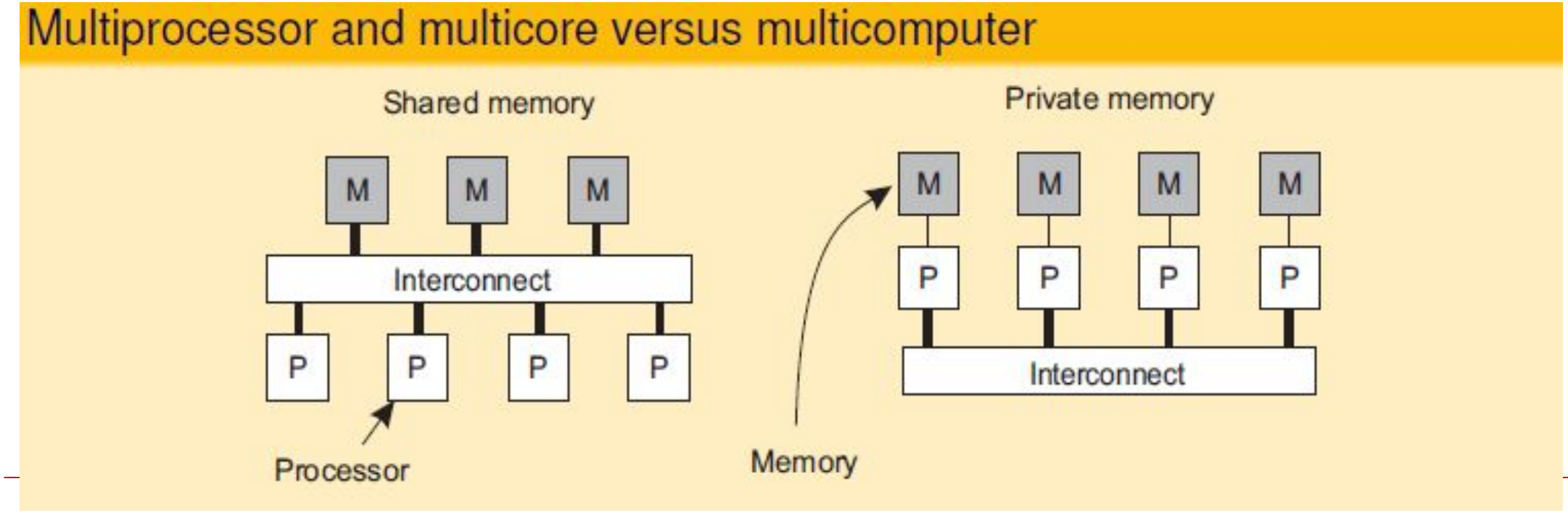
Three types of distributed systems

1. High performance distributed computing systems
2. Distributed information systems
3. Distributed systems for pervasive computing

1. High performance distributed computing systems

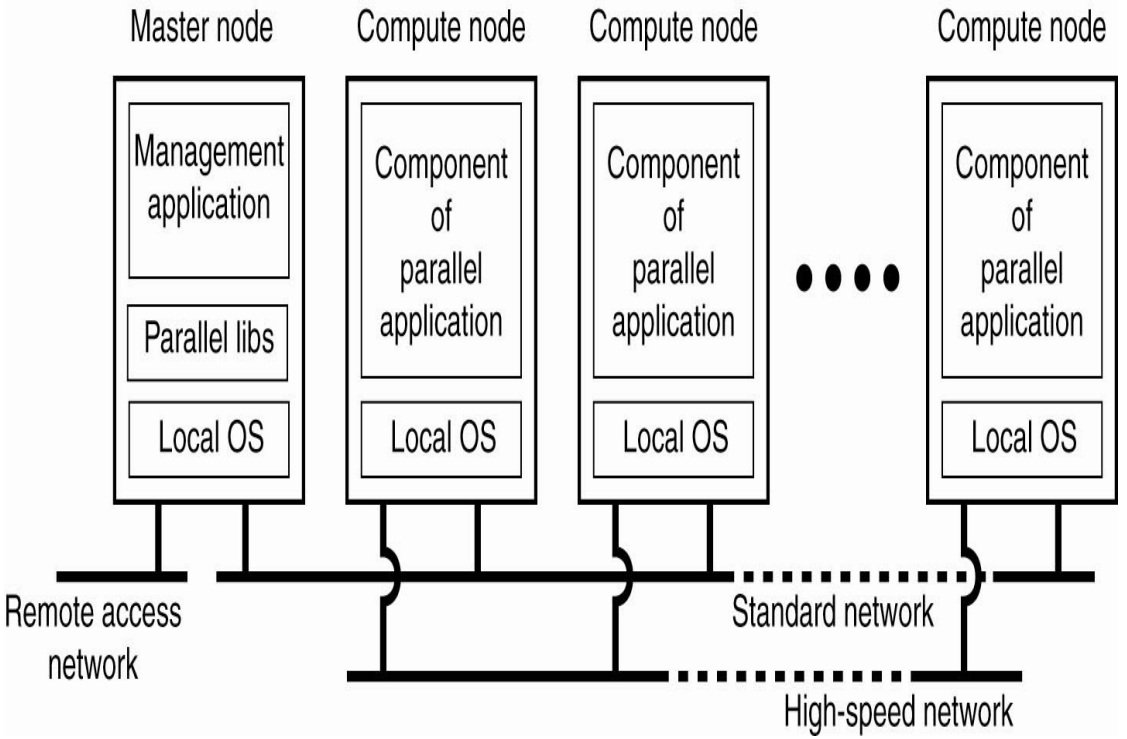
Observation

- High-performance distributed computing started with **parallel computing**
- Multiprocessor and multicore versus multicomputer



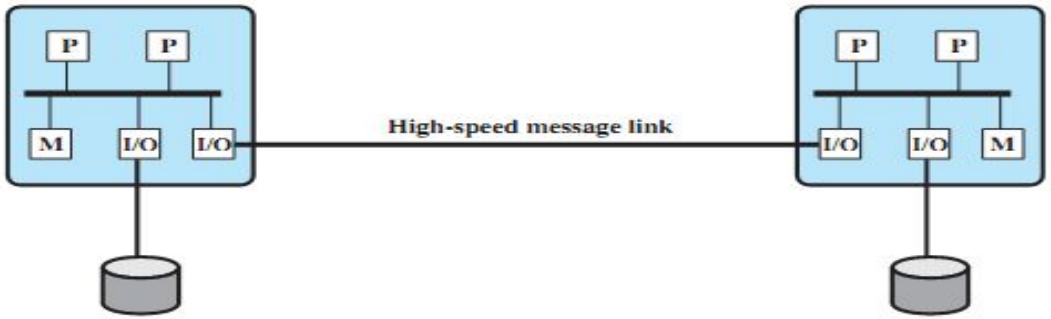
Cluster Computing Systems

- Collection of similar workstations/PCs, closely connected by means of a high-speed LAN:
  - Each node runs the same OS.
  - Homogeneous environment
  - Can serve as a supercomputer
  - Excellent for parallel programming
- Examples: Linux-based Beowulf clusters, MOSIX (from Hebrew University).

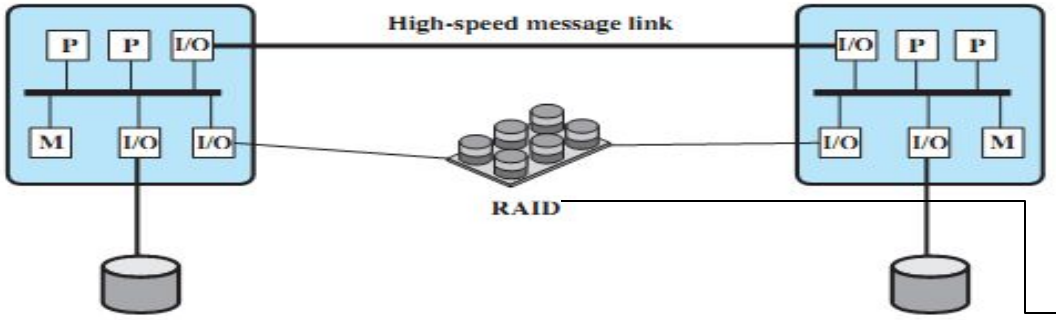


Clustered Systems Architecture

Cluster Computing Systems



(a) Standby server with no shared disk

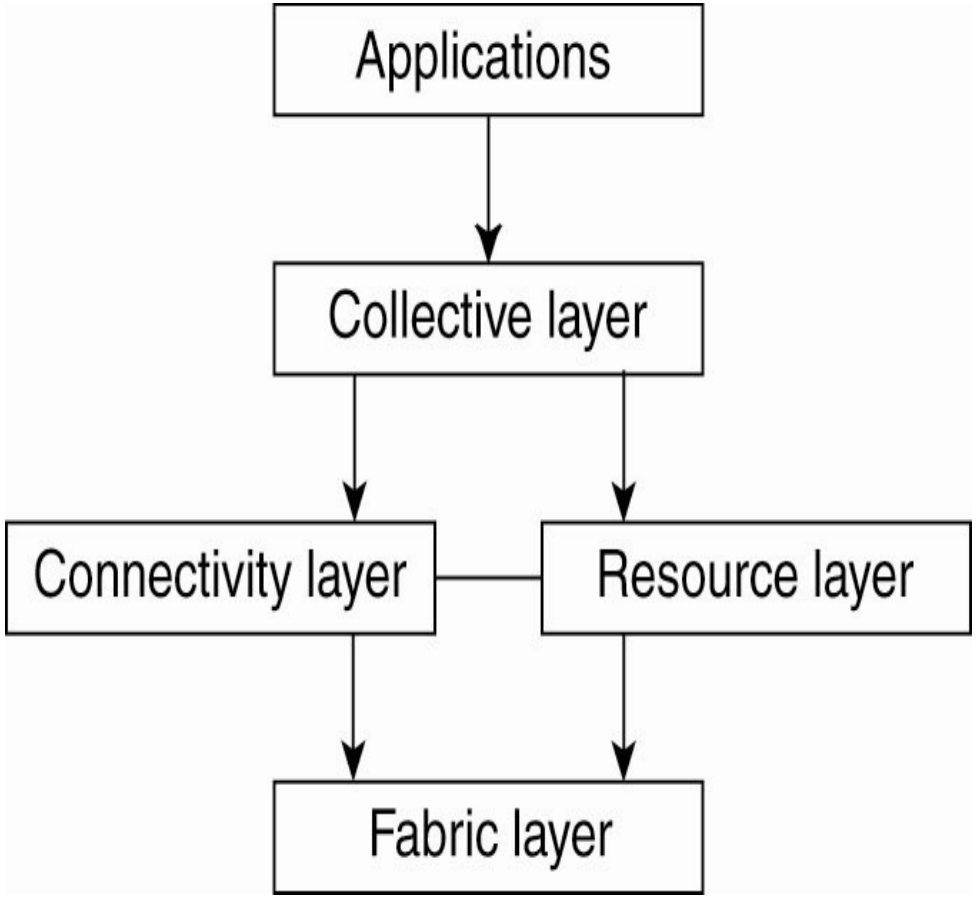


Clustered Configuration

RAID ("Redundant Array of Independent Disks") is a data storage virtualizationis a data storage virtualization technology that combines multiple physical disk driveis a data storage virtualization technology that combines multiple physical disk drive components into one or more logical units

Grid Computing Systems

- Collection of computer resources, usually owned by multiple parties and in multiple locations, connected together such that users can share access to their combined power:
  - Can easily span a wide-area network
  - Heterogeneous environment
  - Crosses administrative/geographic boundaries
  - Supports Virtual Organizations (VOs)
- Examples: EGEE - Enabling Grids for E-SciencE (Europe), Open Science Grid (USA).



Architecture for Grid Computing Systems

2. Distributed Information Systems

Two forms of distributed systems

a. Transaction Processing Systems (TPS)

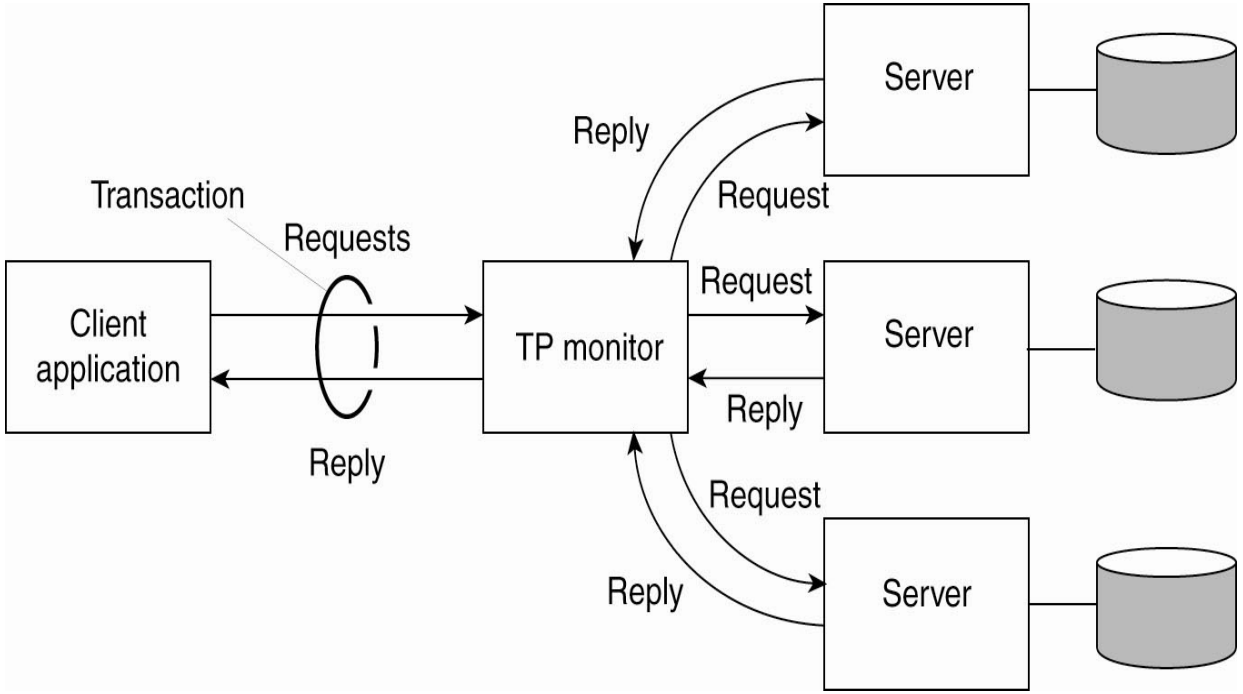
b. Enterprise Application Integration (EAI)

2. Distributed Information Systems

a. Transaction Processing Systems (TPS)

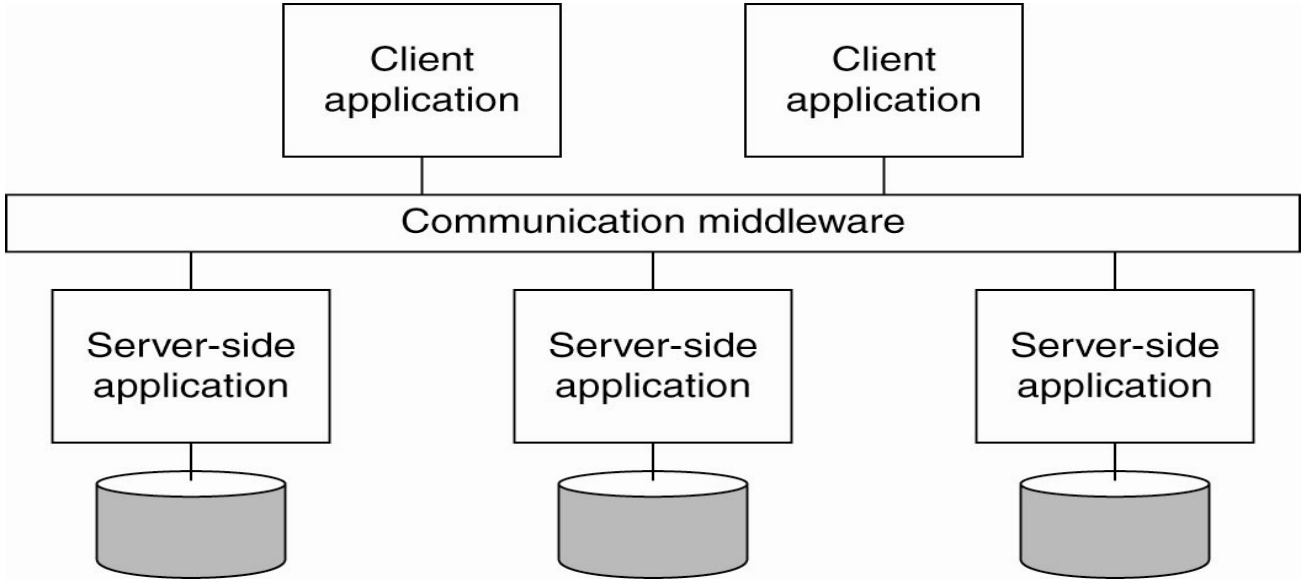
Characteristic properties of transactions:

- Atomic: To the outside world, the transaction happens indivisibly.
- Consistent: The transaction does not violate system invariants.
- Isolated: Concurrent transactions do not interfere with each other.
- Durable: Once a transaction commits, the changes are permanent.



2. Distributed Information Systems

b. Enterprise Application Integration



Middleware offers ..

- Remote Procedure Call (RPC): Requests are sent through local procedure call, packaged as message, processed, responded through message, and result returned as return from call.
- Message Oriented Middleware (MOM): Messages are sent to logical contact point (published), and forwarded to subscribed application.

3. Distributed systems for pervasive computing

Observation :

Emerging next-generation of distributed systems in which nodes are small, mobile, and often embedded in a larger system, characterized by the fact that the system naturally blends into the user's environment.

Three (overlapping) subtypes :

- Ubiquitous computing systems: pervasive and continuously present, i.e. there is a continuous interaction between system and user.
- Mobile computing systems: pervasive, but emphasis is on the fact that devices are inherently mobile.
- Sensor (and actuator) networks: pervasive, with emphasis on the actual (collaborative) sensing and actuation of the environment.

Ubiquitous systems

Core elements :

- Distribution: Devices are networked, distributed, and accessible in a transparent manner
- Interaction: Interaction between users and devices is highly modest
- Context awareness: The system is aware of a user's context in order to optimize interaction
- Autonomy : Devices operate autonomously without human intervention and are thus highly self-managed
- Intelligence :The system as a whole can handle a wide range of dynamic actions and interactions

Distributed System Models /Architecture

Important Models (styles of architecture) for distributed systems

- Layered architectures
- Object-based architectures
- Event-based architectures
- Data-centered architectures

Distributed System Models

A Model/ style is formulated in terms of -

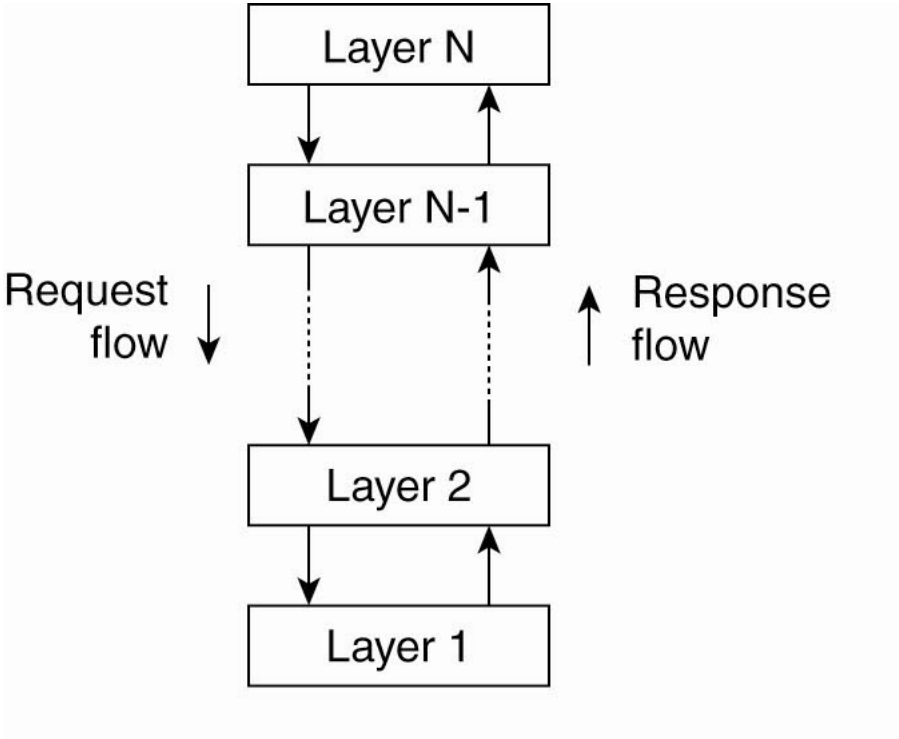
- (replaceable) components with well-defined interfaces
- the way that components are connected to each other
- the data exchanged between components
- how these components and connectors are jointly configured into a system.

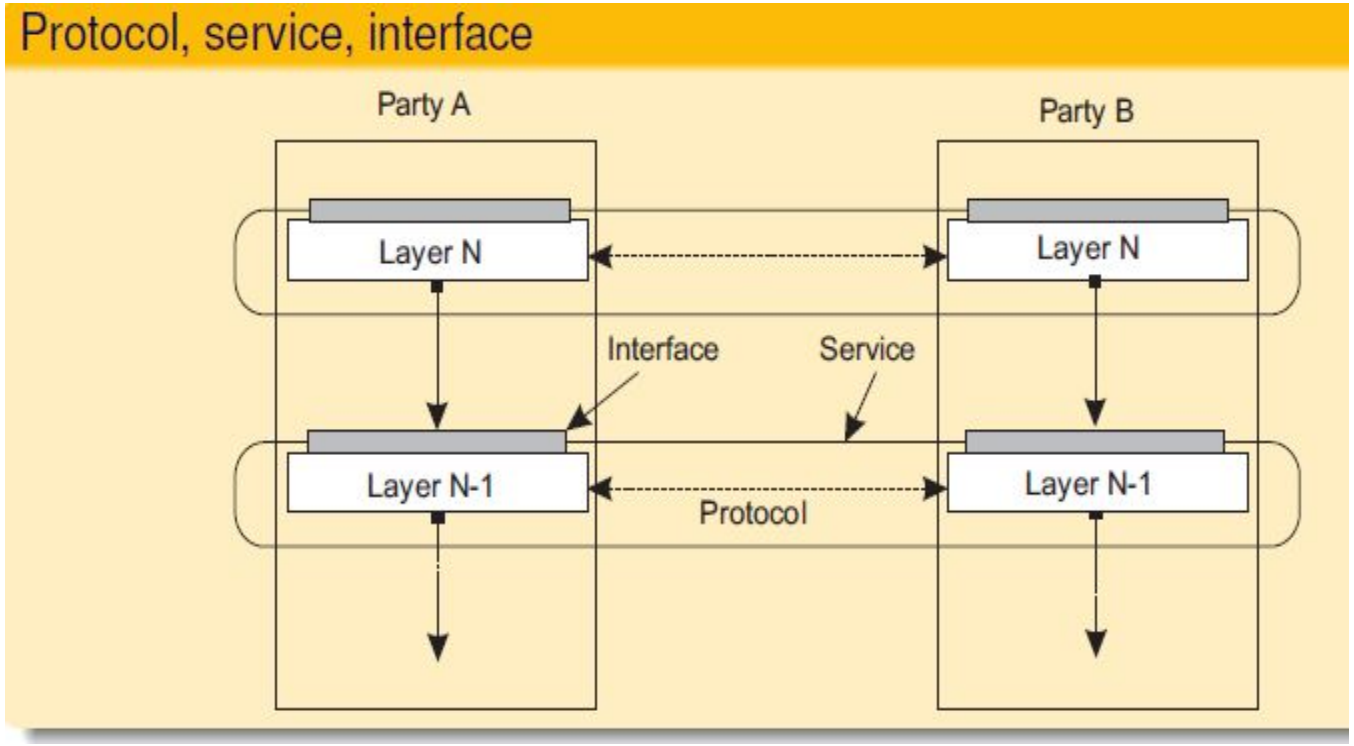
Connector :

A mechanism that mediates communication, coordination, or cooperation among components. Example: facilities for (remote) procedure call, messaging, or streaming.

Distributed System Models

Layered architectures



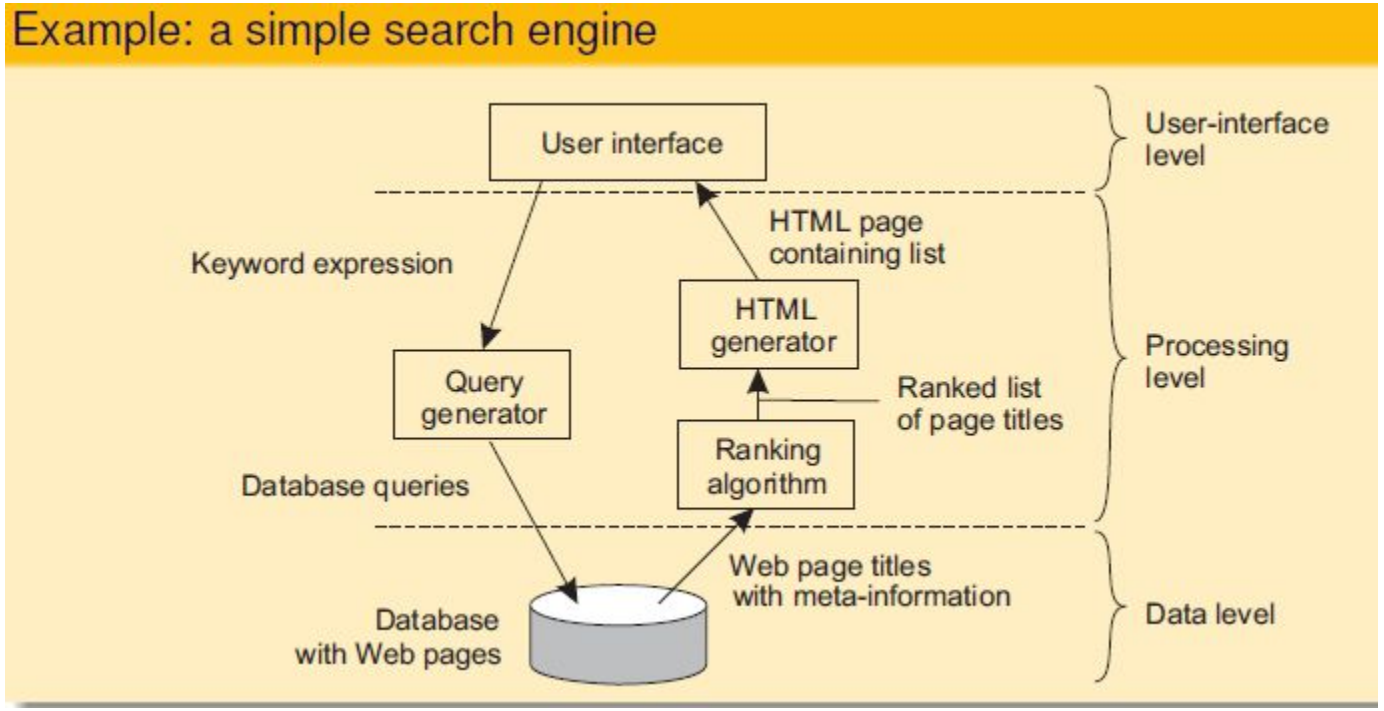


Distributed System Models

Layered architectures

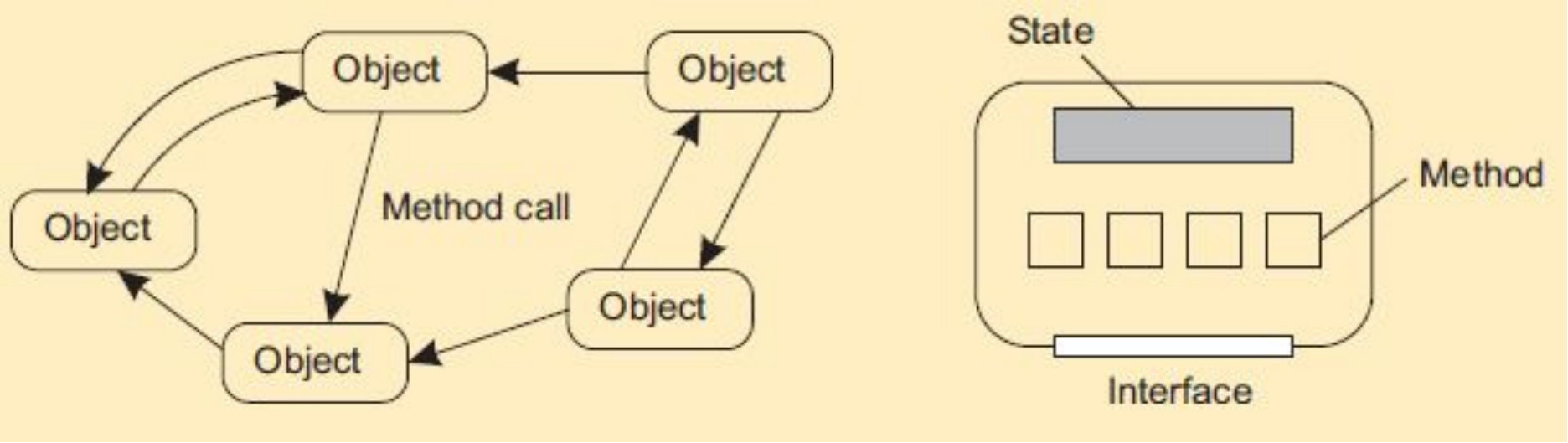
Traditional 3 layered architecture

- Application Layer
- Processing Layer
- Data Layer



Distributed System Models

Object-based architectures: Components are objects, connected to each other through procedure calls. Objects may be placed on different machines; calls can thus execute across a network.



Encapsulation: Objects are said to encapsulate data and offer methods on that data without revealing the internal implementation.

Distributed System Models

Example Object-based architectures (Amzaon's Storage) :

Objects (i.e., files) are placed into **buckets** (i.e., directories). Buckets cannot be placed into buckets. Operations on ObjectName in bucket BucketName require the following identifier:
http://BucketName.s3.amazonaws.com/ObjectName

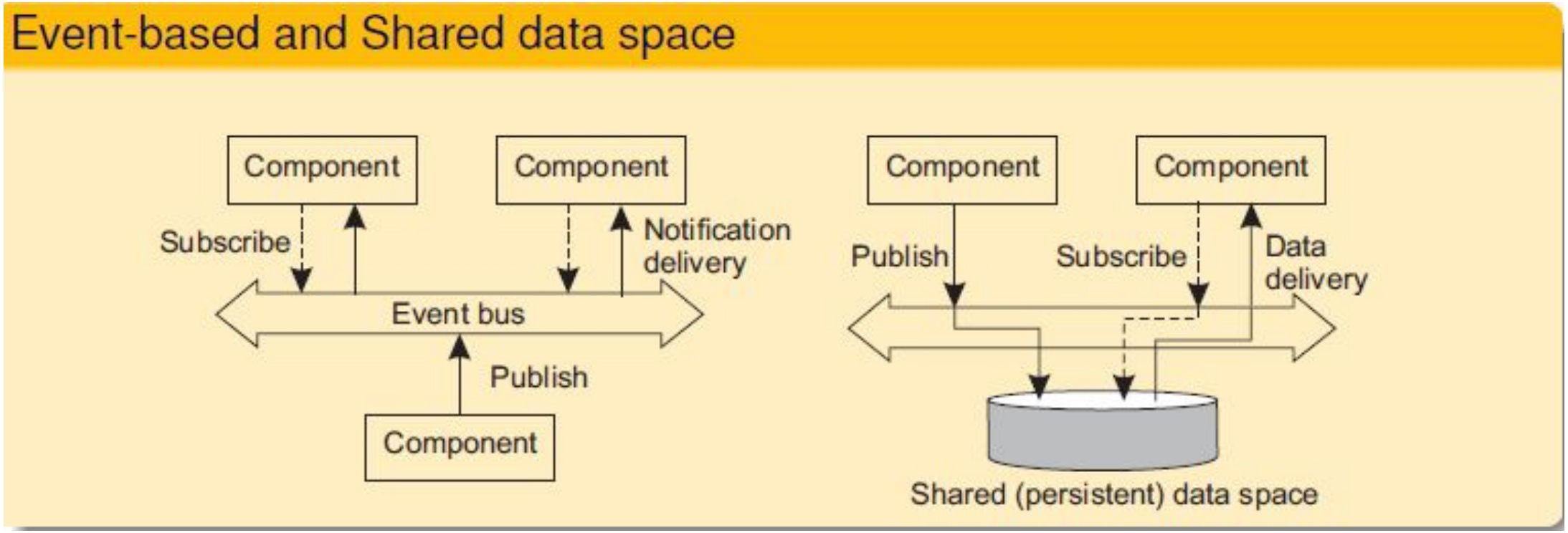
Typical operations

All operations are carried out by sending HTTP requests:

- Create a bucket/object: PUT, along with the URI
- Listing objects: GET on a bucket name
- Reading an object: GET on a full URI

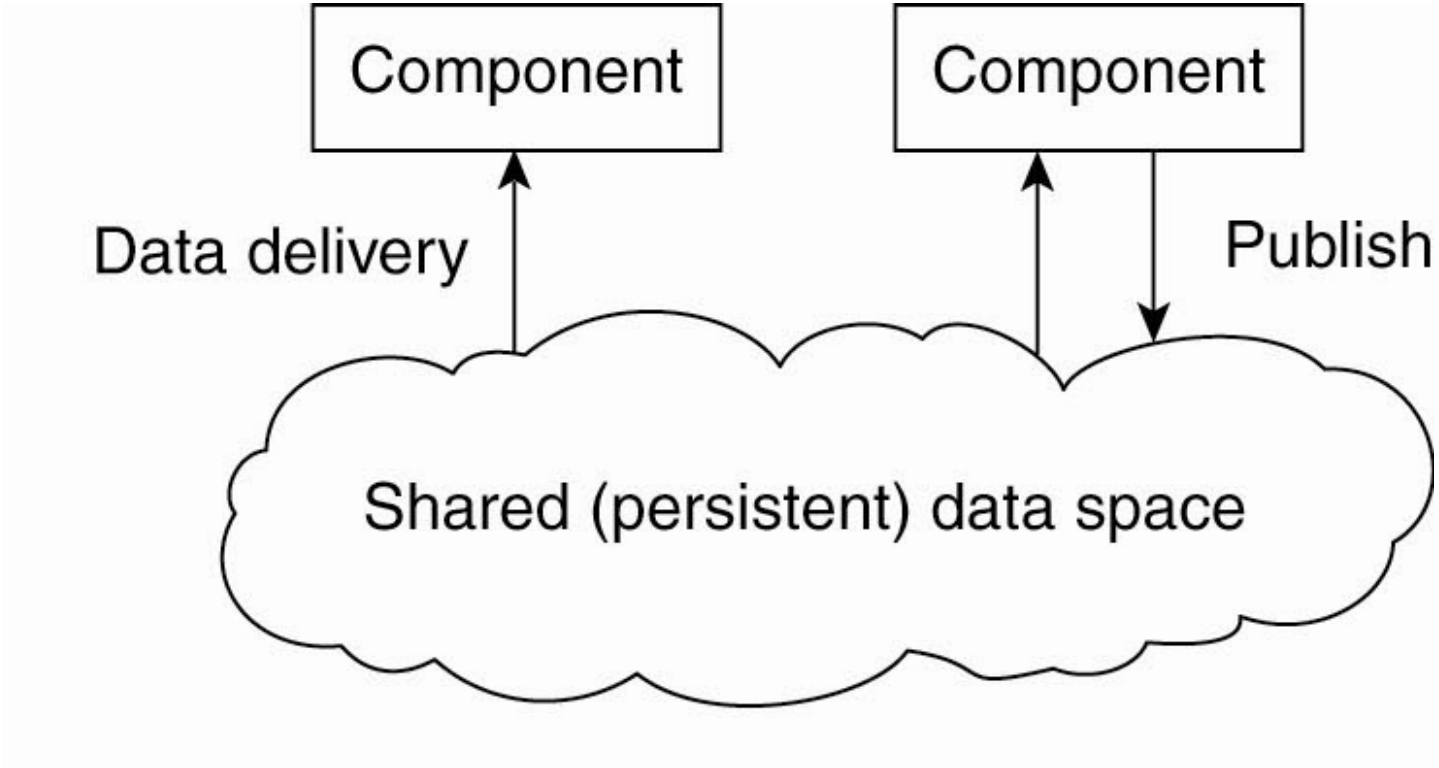
Distributed System Models

Event-based architectures



Distributed System Models

Data-centered architectures



Module 1: Introduction to Distributed Systems

Content….

1.1 Characterization of Distributed Systems: Issues, Goals, and Types of distributed systems, Distributed System Models, **Hardware Concepts, Software Concept.**

1.2 Middleware: Models of Middleware, Services offered by middleware, Client Server model.

Hardware Concept

Classification of Multiple CPU Computer Systems

Into two groups:

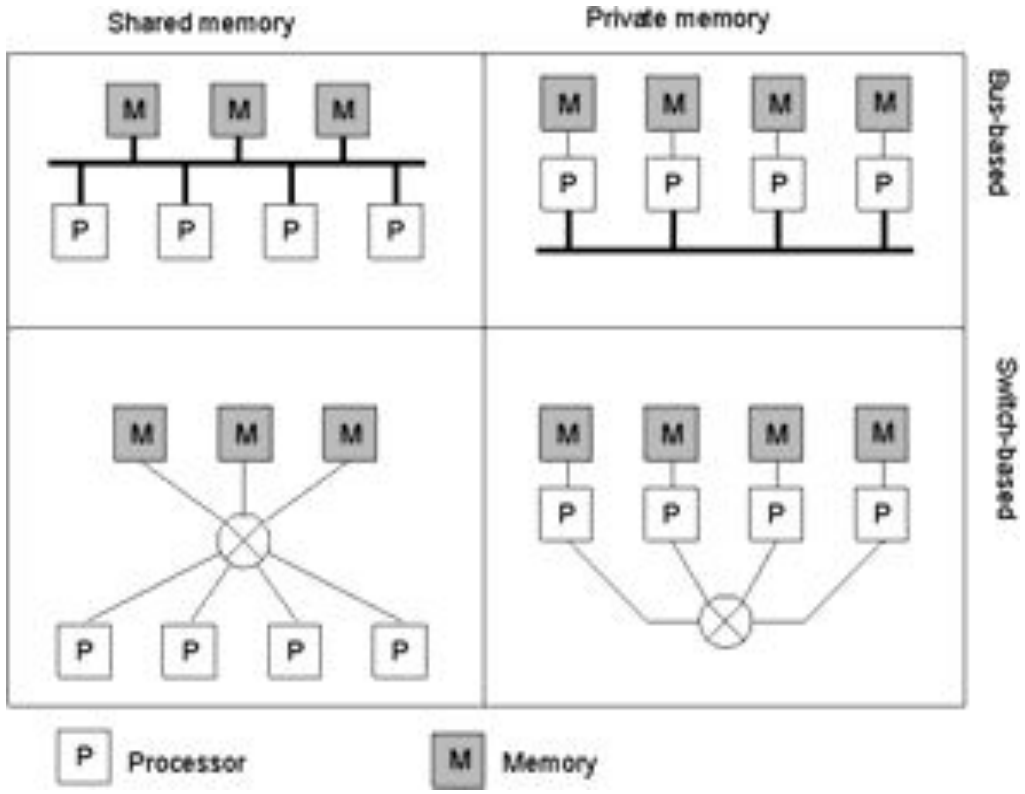
Multiprocessors (shared memory): there is single physical address shared by all CPUs

Multicomputers: each machine has its own private memory. Either Homogeneous or Heterogeneous

Further divided based on the architecture of the interconnection network:

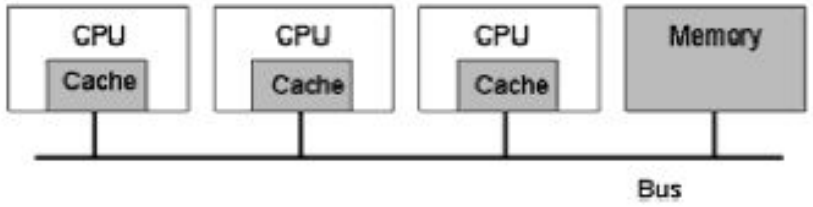
Bus: a single network that connects all machines

Switch



Hardware Concept

Multiprocessors

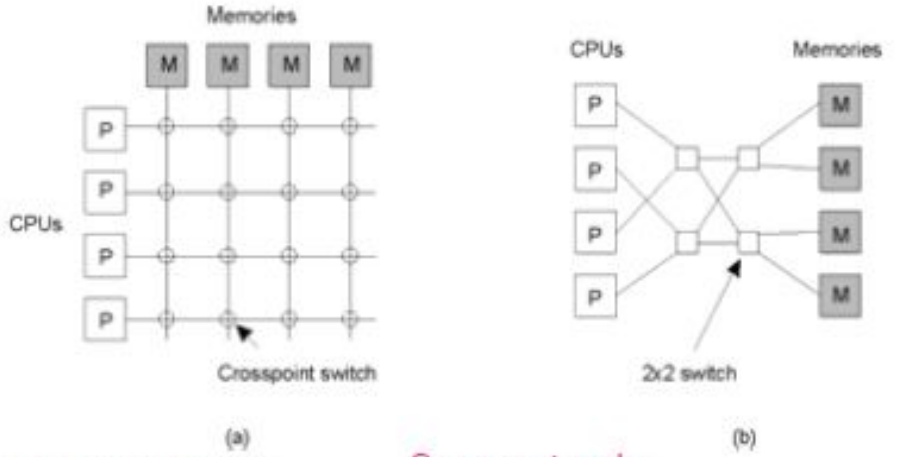


Overload the bus ⇒ cache memory
High hit rate drops the amount of bus traffic
But incoherency

Scalability

Different method to connect the memory with the CPU ⇒ divide the memory in modules

Multiprocessors



n^2 crossbar switches

Omega network

Problem: many switches between the CPU and the memory

NUMA (NonUniform Memory Access) machine: some memory is associated with each CPU

Hardware Concept

Homogeneous Multicomputer Systems

CPU-to-CPU communication

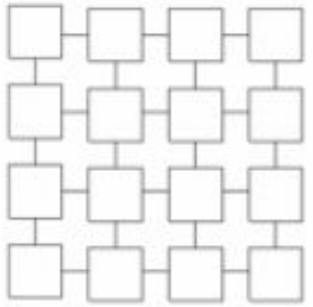
aka System Area Networks (SANs)

Bus-based connected through a multi-access network such as Fast Ethernet, problem?

Switch-based: routed instead of broadcast

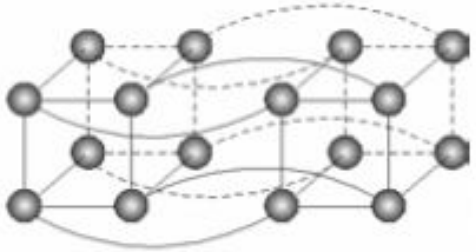
Different topologies

Homogeneous Multicomputer Systems



(a)

Grid



(b)

Hypercube (n-dimensional cube)

4-dimensional

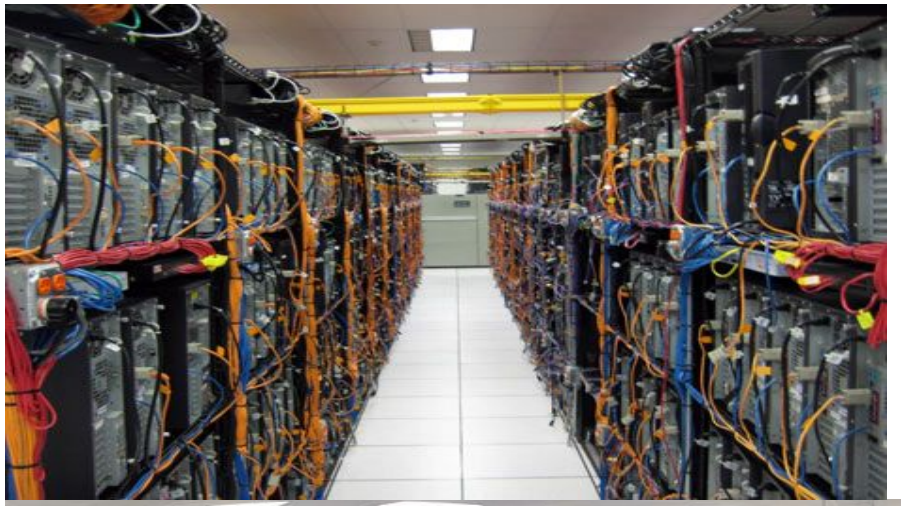
Massively Parallel Processors (MPPs)

Clusters of Workstations (COWs)

Hardware Concept

Heterogeneous Multicomputer Systems :

- **Heterogeneous machines -**
  - High performance parallel systems (multiprocessors and multicomputer)
  - High-end PCS and workstations (servers)
  - Simple network computers
  - Mobile computers (laptops, palmtops, Multimedia workstations) and interconnection networks
  - Local-area gigabit networks
  - Wireless connections
  - Long-haul high-latency connections
  - Wide-area switched megabit connections Scale
- Lack of global view
- transparency is harder



Software Concept

Software Concepts

- Much like an OS (resource managers, hides underlying hardware)
- Tightly-coupled (maintain a global view) – loosely coupled
  - DOS (Distributed Operating System)
  - NOS (Network Operating System)
  - Middleware

System	Description	Main Goal
DOS	Tightly-coupled operating system for multi-processors and homogeneous multicomputers	Hide and manage hardware resources
NOS	Loosely-coupled operating system for heterogeneous multicomputers (LAN and WAN)	Offer local services to remote clients
Middleware	Additional layer at top of NOS implementing general-purpose services	Provide distribution transparency

Distributed Operating Systems

Two types of Distributed OS: Multiprocessor OS and Multicomputer OS

Multi-processor OS

- Shared memory
- Functionality similar to traditional OSs but handle multiple CPUs
- Aim at supporting high performance through multiple CPUs, make their number transparent to the application
- Similar to multitasking uniprocessor OS:
  - All communication done by manipulating data at shared memory locations.
  - Protection is done through synchronization primitives

Distributed Operating Systems

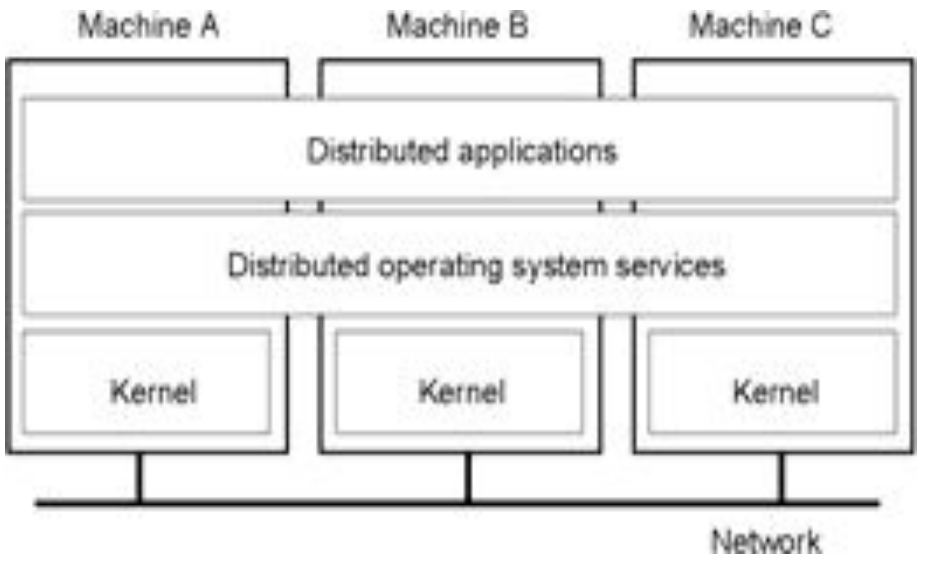
Multicomputer OS

- Harder than multiprocessor OS: Because memory is not shared
- Emphasis shifts to processor communication by **message passing**
- OSs on each computer knows about the other computers
- OS on different machines generally the same
- Services are generally (transparently) distributed across computers

Distributed Operating Systems

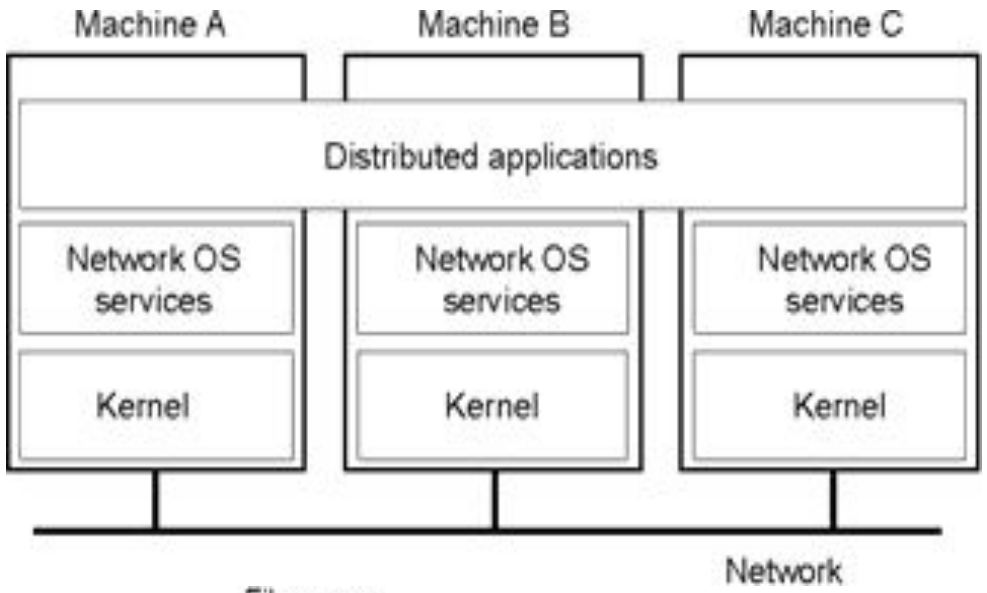
Multicomputer Operating Systems

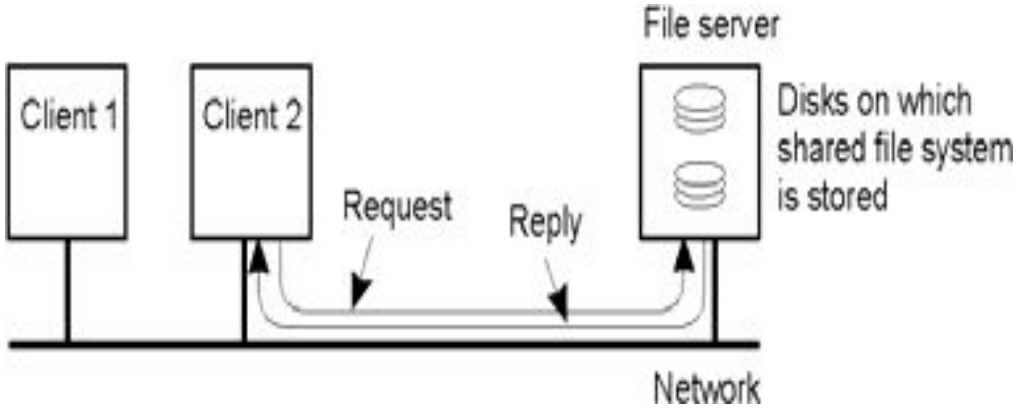
- Each node has each own kernel: modules for managing local resources (memory, local CPU, local disk, etc) + handling interprocess communication (sending and receiving messages to and from other nodes)
- Common layer of software: implements the OS as a virtual machine supporting parallel and concurrent execution of tasks.
- Facilities: assigning a task to a processor, providing transparent storage, general interprocess communication



Network Operating System

- Do not assume that the underlying hardware is homogeneous and that it should be managed as if it were a single system
- Provide facilities to allow users to make use of services provided on a **specific** machine
- rlogin machine
- rcp machine1:file1 machine2:file2





Middleware

DOS: transparency

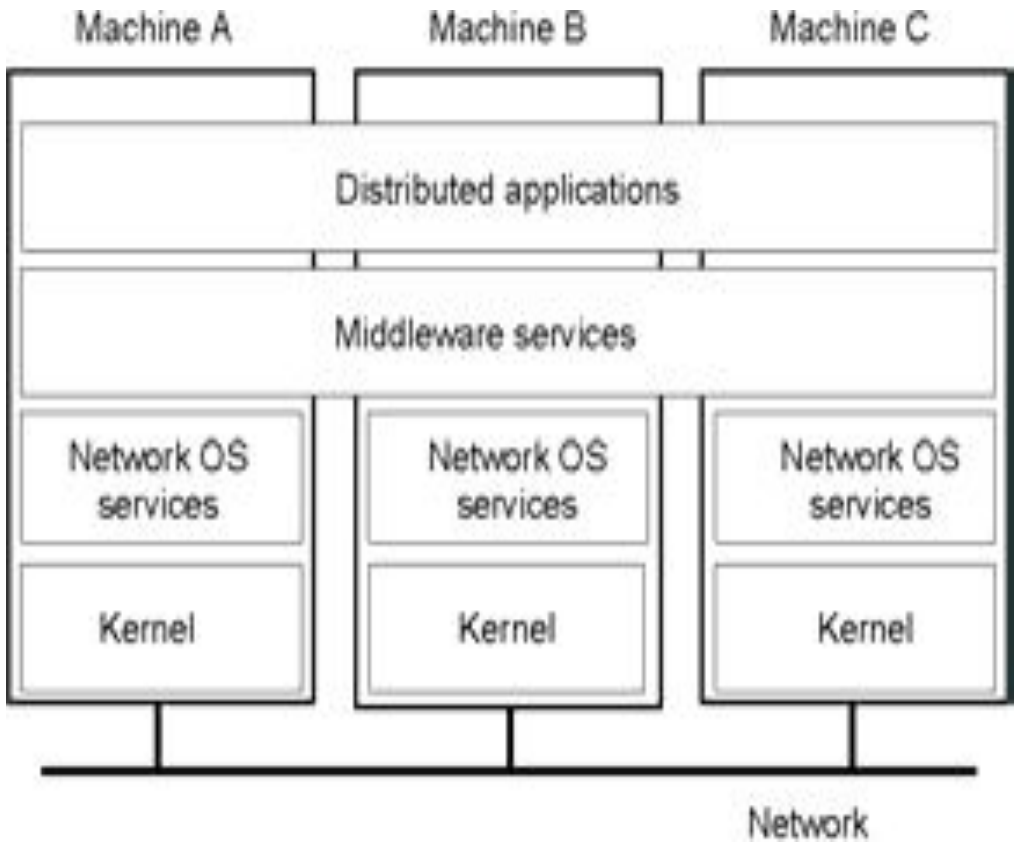
NOS: scalability & openness

Middleware:

- add a layer on top of a NOS for transparency
- provide interface to integrate heterogeneous systems
- manage interaction application <-> server
- roles of middleware
  - as a programming abstraction hide complexity to the programmer
  - as an infrastructure basis of programming abstraction extensions

Middleware

- Middleware itself does not manage an individual mode
- OS on each computer need not know about the other computers
- OS on different computers need not be the same
- Services are generally (transparently) distributed across computers



Middleware Services

1. **Communication facilities (**offer high-level communication facilities to hide low-level message passing)
   - Procedure calls across networks
   - Remote-object method invocation
   - Message-queuing systems
   - Advanced communication streams
   - Event notification service

Middleware Services

2. Information system services (help manage data)

- Large scale system-wide naming services
- Advanced directory services (search engines)
- Location services for tracking mobile objects
- Persistent storage facilities
- Data caching and replication

Middleware Services

3. **Control services** (giving applications control over when, where and how they access data)
    - Code migration
    - Distributed transaction processing

4. **Security services**
    - Authentication and authorization services
    - Simple encryption services
    - Auditing service

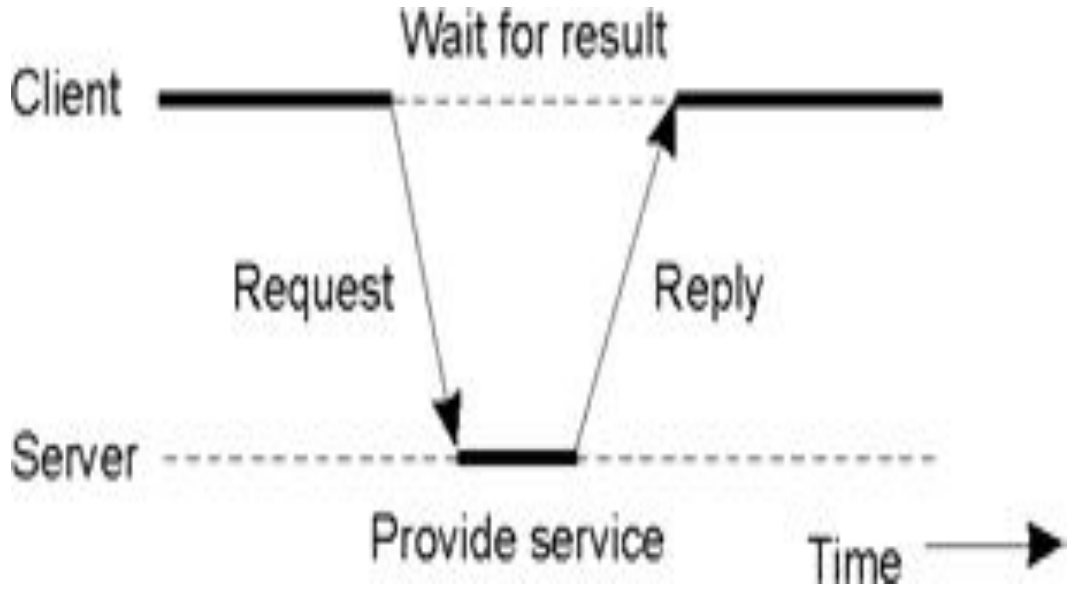
Comparison between Systems

Item	Distributed OS		Network OS	Middleware- based OS
	Multiprocessing	Multicomputer		
Degree of transparency	Very High	High	Low	High
Same OS on all nodes	Yes	Yes	No	No
Number of copies of OS	1	N	N	N
Basis for communication	Shared memory	Messages	Files	Model specific
Resource management	Global, central	Global, distributed	Per node	Per node
Scalability	No	Moderately	Yes	Varies
Openness	Closed	Closed	Open	Open

The Client-Server Model

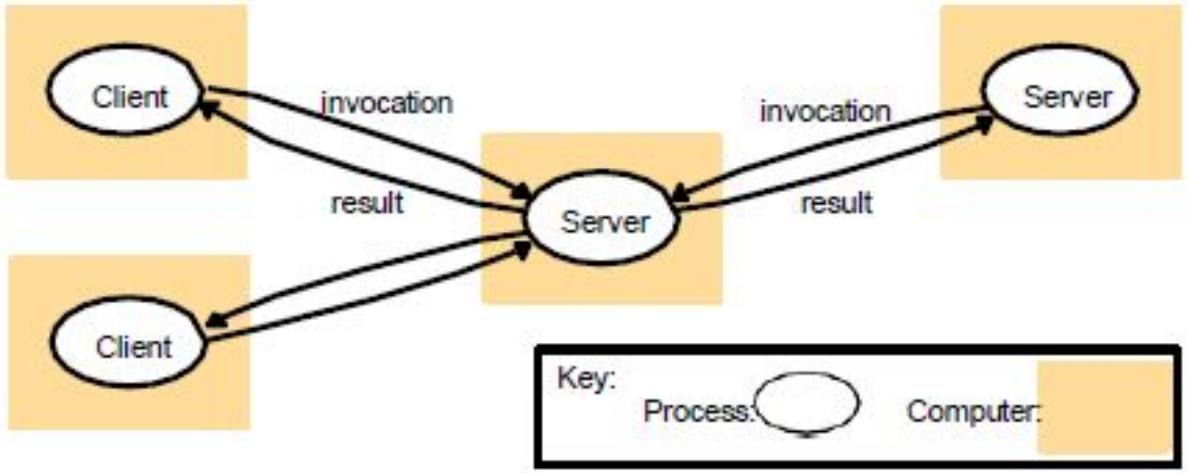
Processes are divided into

- **Servers:** implementing a specific service
- **Clients:** requesting a service from a server by sending it a request and subsequent waiting for the server's reply.
- Distributed across different machines
- Follow a **request-reply**

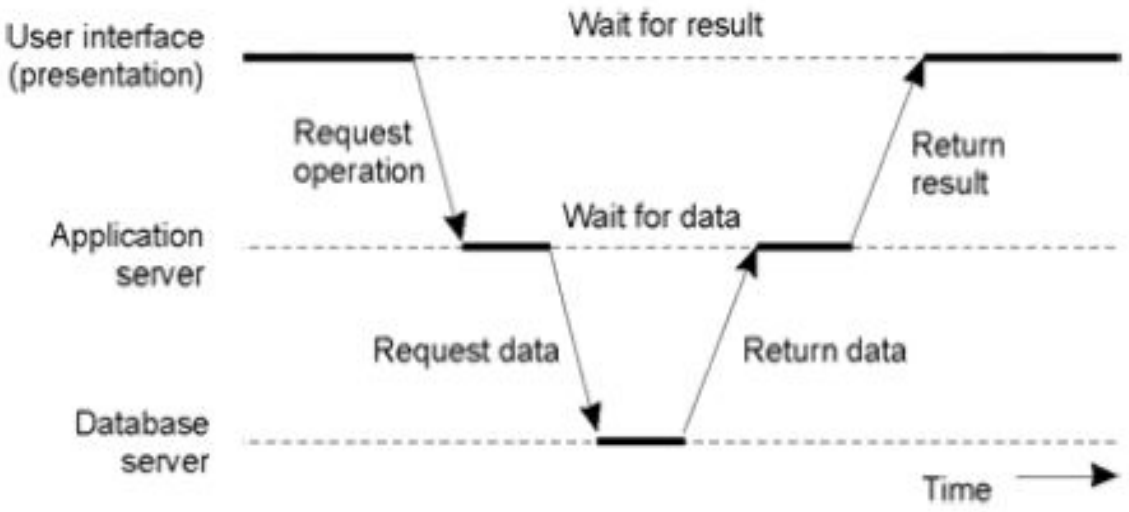


The Client-Server Multitiered Architecture

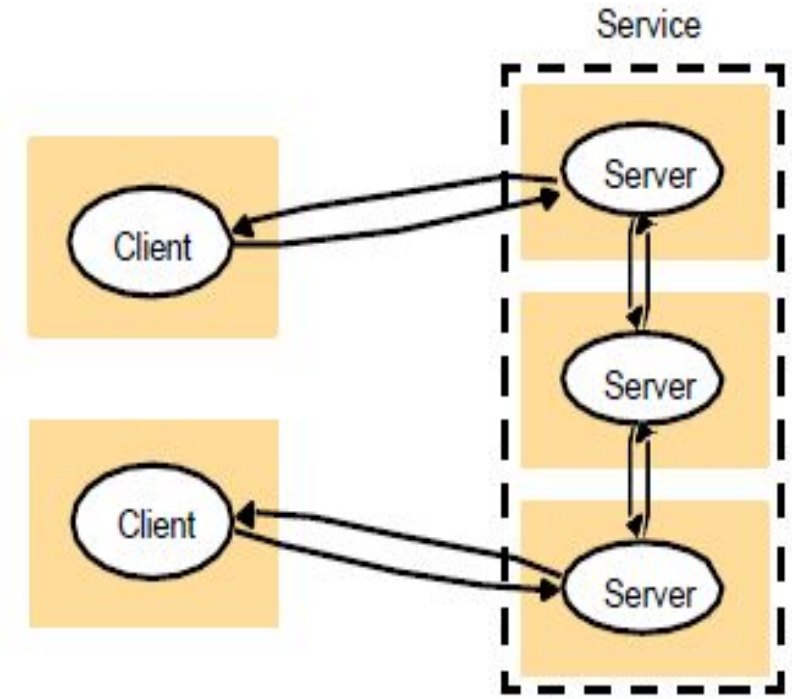
An example of a server acting as a client.



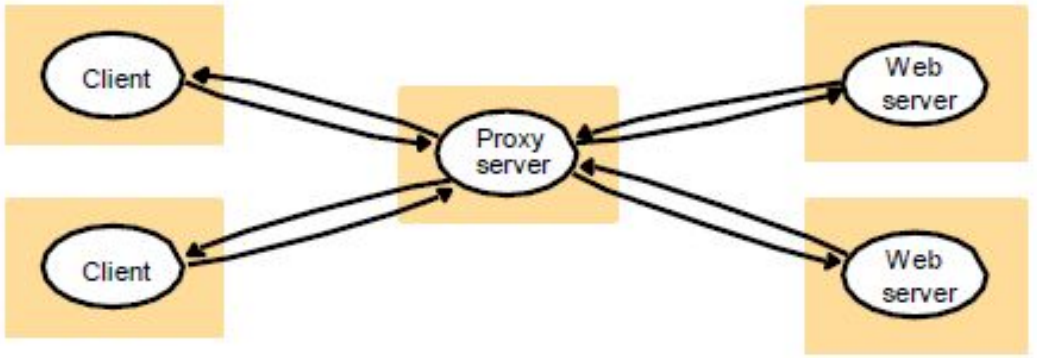
An example of a server acting as a client.



The Client-Server Multitiered Architecture

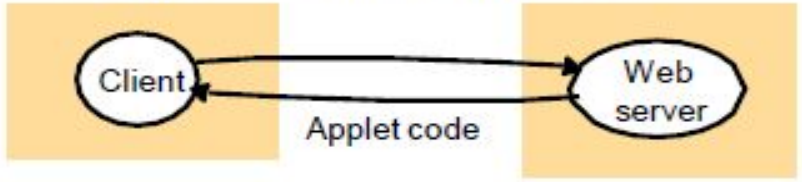


A) Collaborating Server



B) Proxy Server

client request results in the downloading of applet code



client interacts with the applet

References

- Andrew S. Tanenbaum and Maarten Van Steen, “Distributed Systems: Principles and Paradigms”, 2nd edition, Pearson Education.

- George Coulouris, Jean Dollimore, Tim Kindberg,"Distributed Systems: Concepts and Design", 4th Edition, Pearson Education, 2005.

Thank you !!!