# Basics



## Topics to be covered

Definitions and Examples
Goals
Models (architectural, fundamental)
Hardware and Software Concepts
The Client-Server Model



## Historical

**Two developments from mid 50s**

- Powerfull microprocessors

  100 million dollars -- 1 instr per sec

  1000 dollars -- 10 million instr per sec

  $10^{12}$ price/performance gain

  Rolls Royce cost 1 dollar -- a billion miles per gallon
  (200-page manual to open the door)

- Local and Wide Area networks (LANs and WANs)



## Definition of a Distributed System

A distributed system is:

a collection of independent computers that appears to its users as a single coherent system

Two aspects:

(1) Independent computers

(2) Single system $\Rightarrow$ middleware



## Definition of a Distributed System

Characteristics

(1) Heterogeneity hidden

(2) Interact with a consistent and uniform way

(3) Continous availability

(4) Scale

Issues

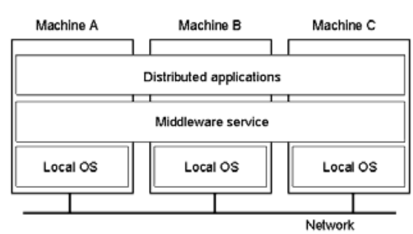
(1) Concurrency

(2) No global clock

(3) Independent failures



## A Distributed System as Middleware



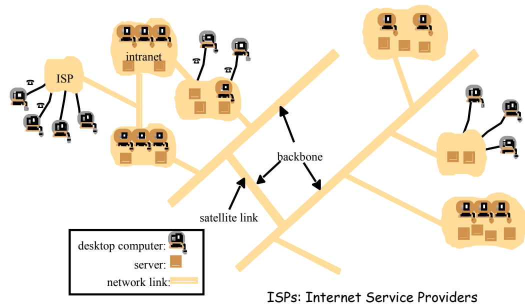

*Note that the middleware layer extends over **multiple** machines.*

## Examples of Distributed Systems

The Internet
Intranets
Mobile and Ubiquitous Computing
The Web
p2p systems (such as Napster)
File systems (SUN, CODA, Adrews)
Storage Systems (Occean)
Object-based Systems (CORBA, DCOM, etc)
Groupware



## A typical portion of the Internet

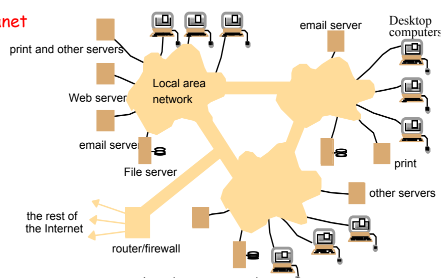



ISPs: Internet Service Providers

Backbone links the Intranets together



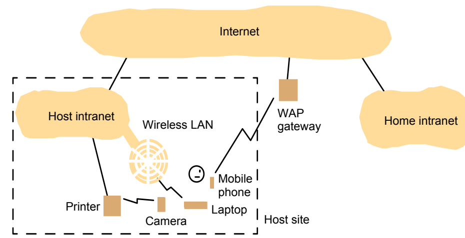
## A Typical Intranet




- A portion of the Internet separately administrated
- Several LANs linked by backbone connections
- Connected to the Internet via a router
- Firewalls protects an intranet by preventing unauthorized messages leaving or entering; implementing by filtered messages



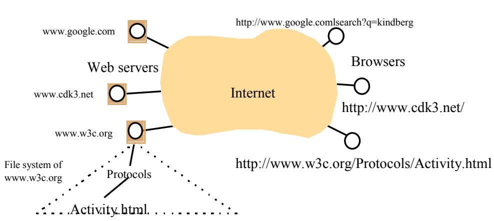
## Portable and handheld devices in a distributed system




- Devices: laptop computers, handheld devices (e.g., PDAs, video cameras), wearable devices, devices embedded in appliances
- Mobile computing, ubiquitous computing, location-aware computing
- In the figure above: 3 different forms of wireless connections: wireless LAN, mobile phone through WAP, infra-red link



## Resource sharing on the Web




- WWW a system for publishing and accessing resources and services across the Internet
- Web browsers act as clients
- Request resources (e.g., web pages) from web servers
- CERN , 1989
- Hypertext structure among documents



## Resource sharing on the Web

- HTML: a language for specifying the content and layout of pages to be displayed by web browsers

- URL: resource locators – goal identify a resource to enable the browser to locate it

scheme:scheme-specific-location

scheme : ftp, http

http://servername[:port][/pathnameofServer][?argument]

Optional number of the port on which the server listens for requests

Optional path name of the server's resource

Set of arguments in the case of a program e.g., search?q=maria

Servername is a DNS name

- HTTP: simple protocol pecifying the ways in which browsers interact with web servers

## Computers in the Internet

| Date | Computers | Web servers |
|---|---|---|
| 1979, Dec. | 188 | 0 |
| 1989, July | 130,000 | 0 |
| 1999, July | 56,218,000 | 5,560,866 |



## Computers vs. Web servers in the Internet

| Date | Computers | Web servers | Percentage |
|---|---|---|---|
| 1993, July | 1,776,000 | 130 | 0.008 |
| 1995, July | 6,642,000 | 23,500 | 0.4 |
| 1997, July | 19,540,000 | 1,203,096 | 6 |
| 1999, July | 56,218,000 | 6,598,697 | 12 |



## Goals

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



## Connecting Users and Resources

Typical resources

Printers, computers, storage facilities, data, files

Why sharing?

Economics

Collaboration, Information Exchange (groupware)

Problems with sharing

Security

Unwanted Communication



## Transparency in a Distributed System

Transparent distributed system:

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



## Transparency in a Distributed System

**access transparency**

Hide differences in data representation and how a resource is accessed

Intel (little endian format)/Sun SPARC (big endian) (order of bytes)

OS with different file name conversions

## Transparency in a Distributed System

**location transparency**

Hide where a resource is located

importance of naming, e.g., URLs

**migration transparency**

Hide that a resource may move to another location

**relocation transparency**

Hide that a resource may move to another location while in use

example, mobile users



## Transparency in a Distributed System

**replication transparency**

Hide that a resource is replicated

subsumes that all replicas have the same name (and thus location transparency)

**concurrency transparency**

Hide that a resource may be shared by several competitive users

leave the resource in a consistent state

more refined mechanism: transactions



## Transparency in a Distributed System

**failure transparency**

Hide the failure and recovery of a resource

*L. Lamport: You know you have one [distributed system] when the crash of a computer you 've never heard of stops you for getting any work done*

Important problem: inability to distinguish between a dead resource and a painfully slow one

**persistent transparency**

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



## Different Forms of Transparency in a Distributed System (summary)

| 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 may be shared by several competitive users |
| 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 |



*(Coulouris et. al.)*

*Access transparency:* enables local and remote resources to be accessed using identical operations. (same)

*Location transparency:* enables resources to be accessed without knowledge of their location. (same) – also migration and relocation

*Mobility transparency:* allows the movement of resources and clients within a system without affecting the operation of users or programs.

*Replication transparency:* enables multiple instances of resources to be used to increase reliability and performance without knowledge of the replicas by users or application programmers.

*Concurrency transparency:* enables several processes to operate concurrently using shared resources without interference between them.

*Failure transparency:* enables the concealment of faults, allowing users and application programs to complete their tasks despite the failure of hardware or software components. – also persistent transparency

**Performance transparency:** allows the system to be reconfigured to improve performance as loads vary.

**Scaling transparency:** allows the system and applications to expand in scale without change to the system structure or the application algorithms.



## Degree of Transparency

Not always desirable

Examples?

Users located in different continents (context-aware)

Not always possible

Examples?

Hiding failures (you can distinguish a slow computer from a failing one/whether an action was performed before a crash)

Trade-off between a high degree of transparency and the performance of a system

Keep web caches exactly up-to-date

Immediately flushing write operations to disk

Retry to access a web page to mask a failure

## Goals

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



## Openness

### Open distributed system

Be able to interact with services from other open systems, irrespectively of the underlying environment

Offers services according to standard rules that describe the **syntax** and the **semantics** of these services

- Rules formalized in protocols
- Services specified through interfaces (described in an Interface Definition Language (IDL) (but only the syntax part)
- Neutral and complete specifications (with regards to a potential implementation)



## 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



## Openness

- 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?



## Scalability

Along three different dimensions:

- size (number of users and/or resources processes)
- geographical (maximum distance between nodes)
- administrative (number of administrative domains)

*The (non) solution: powerful servers*



## Scalability Problems (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 |

Decentralized algorithms

- No complete information about the system state
- Make decision only on local information
- Failure of one machine does not ruin the algorithm
- No assumption of a global clock

## Scalability (geographical)

Geographical scalability:

Synchronous communication
In WAN, unreliable and point-to-point

Related to centralized solutions



## Scalability (administrative)

How to scale a distributed system across multiple, independent administrative domains: conflicting policies with respect to resource usage (and payment), management and security

Expand to a new domain

- Protect itself against malicious attacks from the new domain
- The new domain has to protect itself against malicious attacks from the distributed system



## Scaling Techniques

Three techniques:

- hiding communication latencies
- distribution
- replication



## Scaling Techniques

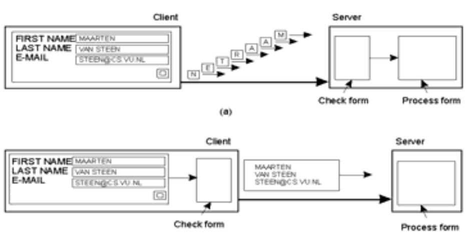
Hiding communication latencies

try to avoid waiting for responses to remote service requests as much as possible

- asynchronous communication (do something else)
- moving part of the computation to the client process
  code shipping in the form of java applets



## Scaling Techniques




The difference between letting:

a) a server or

b) a client check forms as they are being filled



## Scaling Techniques

Distribution

Taking a component, splitting into smaller parts, and spreading these parts across the system

Example:

(1) The World Wide Web

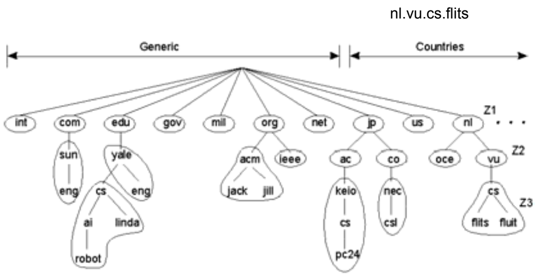
(2) Domain Name Service (DNS)

hierarchically organized into a tree of domains

Each domain divided into non overlapping zones

The names in each domain handled by a single name server

## Scaling Techniques




An example of dividing the DNS name space into zones.



## Scaling Techniques

Replication

- increase availability
- balance the load
- reduce communication latency
- but, consistency problems

Caching (client-driven)



# Models



## System Models

1. Architectural models
2. Fundamental models

An architectural model of a distributed system is concerned with the placement of its parts and the relationships between them

Examples include the client-server model and the p2p model

- determine the distribution of data and computational tasks amongst the physical nodes of the system
- useful when evaluating the performance, reliability, scalability and other properties of distributed systems



## Architectural Models

Software Layers :Structuring of software as layers or modules

| |
|---|
| Applications, Service |
| Middleware |
| Operating Systems |
| Computer and Network Hardware |

Platform

More later …

The end-to-end argument

Some communication activities can be completely and reliably implemented only with the knowledge of applications standing at the nd-points of the communication system



## Architectural Models

System architectures : division of responsibilities between system components and the placement of components on computing nodes in the network

Client-server model and its variations

More later …

## Fundamental Models

Fundamental models are concerned with a more *formal description* of the properties that are *common* in all of the architectural models

Models:

- Interaction model deals with performance and with the difficulty of setting time limits in distributed systems, for example for message delivery

- Failure model gives a precise specification of the faults that can be exhibited by processes and communication channels. Defines reliable communication and correct processes.

- Security model discusses the possible threats to processes and communication channels.



## Interaction Model

- Distributed systems are composed of multiple interacting processes

- Their behavior and state can be described by a distributed algorithm: a definition of the steps to be taken by each process including the transmission of messages between them

- Messages are transmitted between processes to transfer information among them and to coordinate their activity



## Interaction Model

Communication performance characteristics:

Latency: delay between sending a message by one process and its receipt by another

Bandwidth of a computer network: total amount of information that can be transmitted over it in a given time

Jitter: the variation in the time taken to deliver a series of messages

Computer clocks:

Clock drift rate: relative amount that a computer clock differs from a perfect reference clock



## Variants of the Interaction Model

Based on whether they set time limits (lower and upper bounds) on:

- Process execution speeds
- Message transmission delays
- Clock drift rates

Synchronous distributed systems (can set timeouts, can be built)

Asynchronous distributed systems (e.g., Internet, web)

Despite the lack of accurate clocks, the execution of a system can be described in terms of events and their ordering



## Failure Model

Classification of failures:

Omission failures:

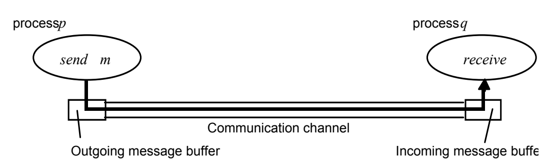
when a process or communication channel fails to perform actions that is supposed to do

Process omission failure: crash

*Fail-stop crash* if other processes can detect certainly that the process has crashed



## Failure Model




Communication omission failures: send-omission (loss of messages between the sending process and the outgoing message buffer), receive-omission (loss of messages between the incoming buffer and ther receiving process), and channel omission (loss of messages between)

## Failure Model

Classification of failures:

Arbitrary or Byzantine failures

Arbitrary failures of processes and channels



## Omission and arbitrary failures

| *Class of failure* | Affects | Description |
|---|---|---|
| Fail-stop | Process | Process halts and remains halted. Other processes may detect this state. |
| Crash | Process | Process halts and remains halted. Other processes may not be able to detect this state. |
| Omission | Channel | A message inserted in an outgoing message buffer never arrives at the other end's incoming message buffer. |
| Send-omission | Process | A process completes a send, but the message is not put in its outgoing message buffer. |
| Receive-omission | Process | A message is put in a process's incoming message buffer, but that process does not receive it. |
| Arbitrary (Byzantine) | Process or channel | Process/channel exhibits arbitrary behaviour: it may send/transmit arbitrary messages at arbitrary times, commit omissions; a process may stop or take an incorrect step. |



## Timing failures

In synchronous systems:

| *Class of Failure* | *Affects* | *Description* |
|---|---|---|
| Clock | Process | Process's local clock exceeds the bounds on its rate of drift from real time. |
| Performance | Process | Process exceeds the bounds on the interval between two steps. |
| Performance | Channel | A message's transmission takes longer than the stated bound. |



## Security Model

Securing the processes and the channel and protecting the objects

Protecting the objects: access rights (specify who is allowed to perform each operation of an object)

Associate with each invocation and each result the authority on which it is issued called a principal



## Security Model

To model security threats, we postulate an enemy (or adversary)

Send any message to any process and reading/copying any message between a pair of processes

1. Threats to processes (cannot identify the identity of the sender: holds for both clients and servers)
2. Threats to communication channels
3. Other possible threats (mobile code, denial of service)



# Hardware Concepts

## 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

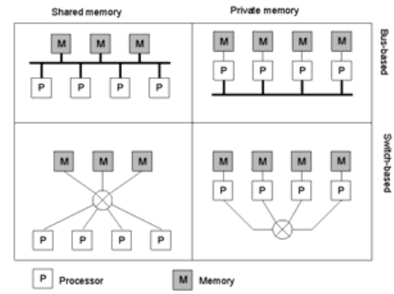
Further divided based on the architecture of the interconnection network:

Bus: a single network that connects all machines

Switch

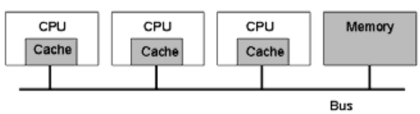


## Hardware Concepts






## Multiprocessors



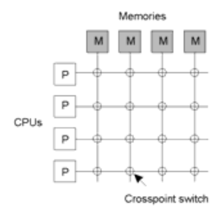

Overload the bus ⇒ cache memory
High hit rate drops the amount of bus traffic
But incoherency

Scalability

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

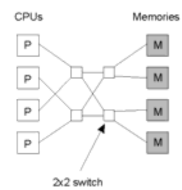


## Multiprocessors




(a)

$n^2$ crossbar switches




(b)

Omega network

Problem: many switches between the CPU and the memory

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



## Homogeneous Multicomputer Systems

CPU-to-CPU communication

aka System Area Networks (SANs)

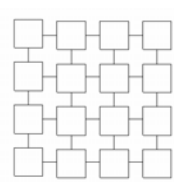
**Bus-based** connected through a multi-access network such as Fast Ethernet, problem?

**Switch-based**: routed instead of broadcast

Different topologies

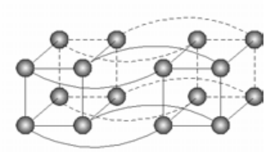


## Homogeneous Multicomputer Systems



(a)

Grid



(b)

Hypercube (n-dimensional cube)

4-dimensional

Massively Parallel Processors (MPPs)

Clusters of Workstations (COWs)

## Heterogeneous Multicomputer Systems

Heterogeneous machines

- High performance parallel systems (multiprocessors and multicomputers
- 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 Concepts



## 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 atop of NOS implementing general-purpose services | Provide distribution transparency |



## Distributed Operating Systems

- Two types: 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:
  - o All communication done by manipulating data at shared memory locations.
  - o Protection is done through synchronization primitives



## Multicomputer Operating Systems

Harder than traditional (multiprocessor) OS: Because memory is not shared

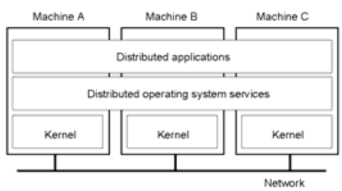
Emphasis shifts to processor communication by message passing

- OSs on each computer knows about the other computers
- OS on different machines generally the same
- Services are generally (transparently) distributed across computers



## Multicomputer Operating Systems

General structure




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

## Multicomputer Operating Systems

Processor communication by message passing

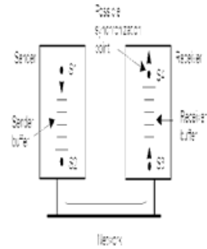
- Often no simple global communication (e.g., broadcast)
- No simple system-wide synchronization mechanisms
- Virtual (distributed) shared memory requires OS to maintain global memory map in software (Distributed Shared Memory (DSM) **vs** Only message passing
- Inherent distributed resource management: no central point where allocation decisions can be made

Practice: only very few truly multicomputer OS exist



## Multicomputer Operating Systems

Semantics of message passing



Buffering of messages at the sender or the receiver

Four possible synchronization points:

**S1** (block the sender when its buffer is full)

**S2** (message has been send)

**S3** (message has arrived at the receiver)

**S4** (message has been delivered to the receiver)



## Multicomputer Operating Systems

Semantics of message passing (continued)

Is the communication reliable?

| Synchronization point | Send buffer | Reliable comm. guaranteed? |
|---|---|---|
| Block sender until buffer not full | Yes | Not necessary |
| Block sender until message sent | No | Not necessary |
| Block sender until message received | No | Necessary |
| Block sender until message delivered | No | Necessary |



## Multicomputer Operating Systems

Emulate shared memory for multicomputers

Distributed Shared Memory Systems (DSMs)

The address space is divided up into pages with the pages being spread over all the processors in the system

When a processor references an address that is not present locally, a trap occurs, and the OS fetches the page



## Network Operating System

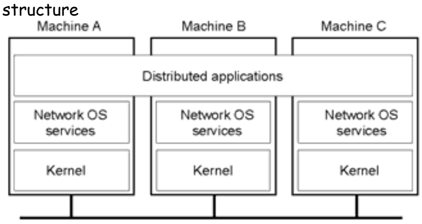
Do not assume that the underlying hardware is homogeneous and that it should be managed as if it were a single system

Provide facilities to allow users to make use of services provided on a **specific** machine

rlogin machine

rcp machine1:file1 machine2:file2

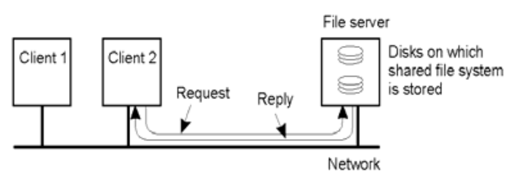
General structure





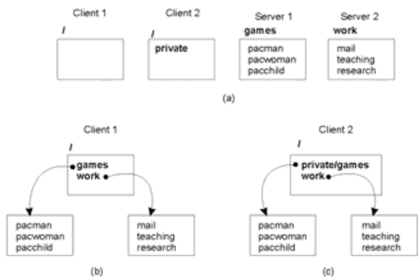

## Network Operating System

Some provide a shared global file system



## Network Operating System

Clients import (or mount) file systems)




Different clients may mount the servers in different places.



## Network Operating Systems

Some characteristics:

• Each computer has its own OS with networking facilities

• Computers work independently (i.e, they may even have different OS)

• Services are to individual nodes (ftp, telnet, www)

• Highly file oriented (basically, processors share *only* files)

• Compared to distributed OSs

  • Lack of transparency (harder to use; need to be managed independently)

  • Easier to add/remove a machine (scalability, openness)



## Middleware

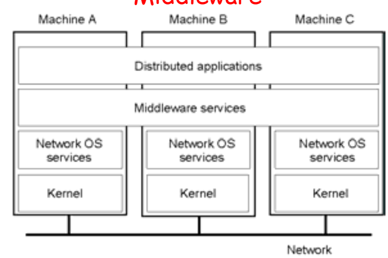
DOS: transparency

NOS: scalability & openness

Middleware: add a layer on top of a NOS for transparency



## 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 Models

Based on some model or paradigm, such as:

• All resources are treated as files (UNIX and Plan 9)

• Distributed file systems

• Remote Procedure Calls (RPCs): allow a process to call a procedure whose implementation is located on a remote machine

• Distributed objects: transparently invoke objects residing on remote machines

• Distributed documents



## Middleware Services

**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

**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

**Control services** (giving applications control over when, where and how they access data)

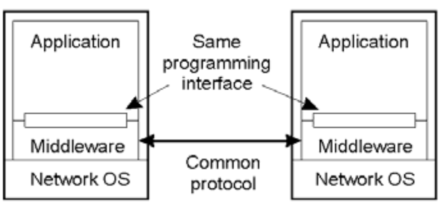
- Code migration
- Distributed transaction processing

**Security services**

- Authentication and authorization services
- Simple encryption services
- Auditing service



## Middleware and Openness




In an open middleware-based distributed system, the protocols used by each middleware layer should be the same, as well as the interfaces they offer to applications.



## Comparison between Systems

| Item | Distributed OS | | Network OS | Middleware-based OS |
|---|---|---|---|---|
| | Multiproc. | Multicomp. | | |
| 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

(and other system architectures)



## Clients and Servers

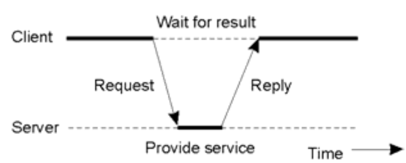
Processes are divided into

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



## Application Layering

Traditional three-layered view

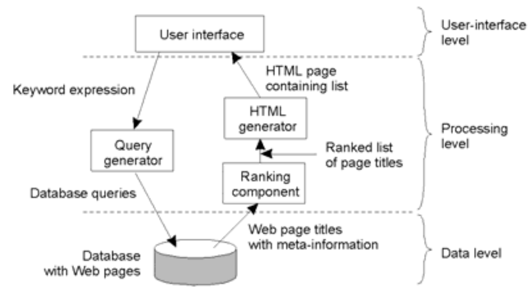
User-interface layer: programs that allow end users to interact with the application; differ in their sophistication

Processing layer: contains the functions of an application

Data layer: contains the data that a client wants to manipulate through the application components (persistency, consistency, data independence)



## Application Layering

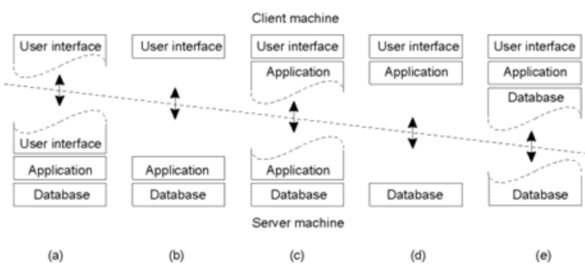



The general organization of an Internet search engine into three different layers



## Multitiered Architectures

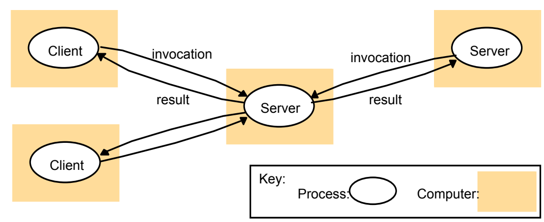
Alternative client-server organizations






## Multitiered Architectures

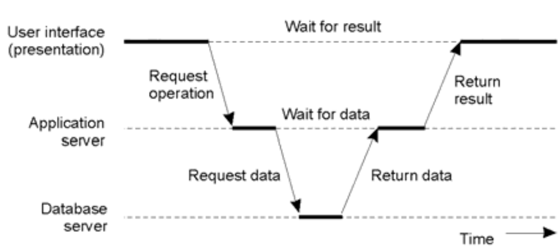
An example of a server acting as a client.






## Multitiered Architectures

An example of a server acting as a client.





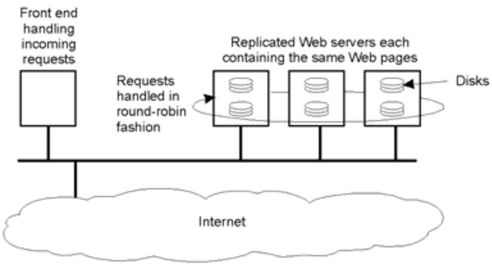

## Alternative Architectures

Vertical distribution: placing logically different components on different machines

Horizontal distribution: a client or server may be physically split up into logically equivalent parts; each operating on its own share of the complete data

# Modern Architectures

An example of horizontal distribution of a Web service.






# Alternative Architectures

Cooperating servers: service is physically distributed across a collection of services:

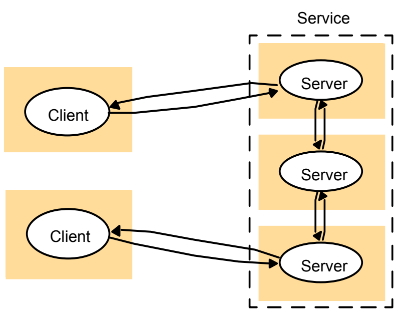
- Traditional multi-tiered architectures
- Replicated files systems
- Network news services
- Large-scale naming systems, etc

Cooperating clients: distributes applications exist by virtue of client collaboration:

- Teleconferencing
- Publish/subscribe

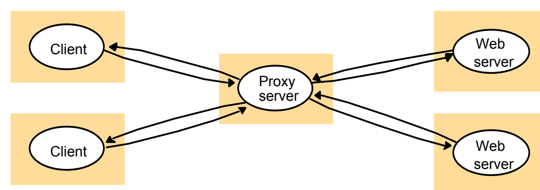


Collaborating servers





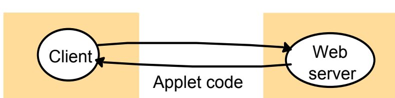

Proxy servers






Web applets

a) client request results in the downloading of applet code

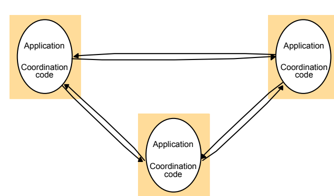



b) client interacts with the applet




# Alternative Architectures

Peer-to-Peer Systems



# Basics