Operating Systems

Interface between the hardware and the rest: editors, compilers, database systems, application programs, your programs, etc.

- Allows portability, enables easier programming,
- The manager of different resources (memory, CPU, disk, printer, etc) in your system
- Takes responsibility away from the users, tends to improve metrics (throughput, response time, etc)
- 2 types:
  - *monolithic*: all functions are inside a single kernel (central part of the OS)
  - *microkernel-based*: non-basic functions float as servers (there is a small kernel for the basic functionality)

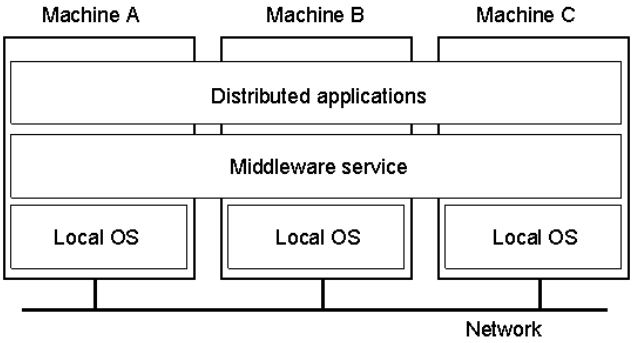
Distributed Systems

A distributed system is:

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

(also, a collection of systems that when one breaks nothing works)

Distributed Systems



A distributed system organized as middleware. What's middleware? Note that the middleware layer extends over multiple machines.

Issues in Distributed Computing

- Why distribute? What's bad about centralized? DISTRIBUTION:
  - allows sharing of data, code, devices, messages, etc
  - is more flexible (can add more resources, scalable)
  - is **cheaper** (several small machines are cheaper than one powerful). That is, the price/performance ratio is smaller than in centralized
  - is usually faster (same as above)
  - can be fault tolerant (if one site fails, not all computations fail)
  - much, MUCH MORE!!!
- More complex? YES, much more (here is the much more)
- More time consuming? (messages need to go back and forth)
- Slower response time? (messages, but can parallelize comps)
- What about reliability, security, cost, network, messages, congestion, load balancing...

Distributed OS Services

- Global *Inter-Process Communication* (IPC) primitives, transparent to the users (currently support to *client-server* computing)
- Global *protection* schemes, so that a validation at a site needs to be validated at another site (Kerberos)
- Global *process management*: usual (destroy, create, etc....) + migration, load distribution, so that the user need not manually logon to a different machine. The OS takes charge, and executes the program requested by the user in a less-loaded, fastresponding machine (compute server, file server, etc).
- Global *process synchronization* (supporting different language paradigms for heterogeneity and openness)
- *Compatibility* among machines (binary, protocol, etc)
- Global *naming* and *file system*

Distributed OSs

- **Transparency** attempts to hide the nature of the system from users.
  - Good, because users usually don't need to know details
  - Degree of transparency is important, too much may be too much
- **Performance** is usually an issue that needs to be studied for a specific system architecture, application, users, etc.
- **Scalability** is important in the long run and general use—some applications, systems, users, etc do not need scalability
- **Distributed algorithms** are also needed, which have the following characteristics:
  - State information should be distributed to all nodes (how? overhead?)
  - Decisions are made based on local information (why?)
  - Fault tolerance (what for?)
  - No global/synchronized clocks (why?)

Transparency in a Distributed 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 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
Parallelism	Hide how many resources are being used
Persistence	Hide whether a (software) resource is in memory or on disk

Degree of Transparency

Observation: Aiming at full distribution transparency may be too much:

- Users may be located in different continents; distribution is apparent and not something you want to hide
- Completely hiding failures of networks and nodes is (theoretically and practically) impossible
  - You cannot distinguish a slow computer from a failing one
  - You can never be sure that a server actually performed an operation before a crash
- Full transparency will cost performance, exposing distribution of the system
  - Keeping Web caches *exactly* up-to-date with the master copy
  - Immediately flushing write operations to disk for fault tolerance

Scalability

- Three dimensions:
  - **Size:** Number of users and/or processes
  - **Geographical:** Maximum distance between nodes
  - **Administrative:** Number of administrative domains
- 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

Techniques for Scaling

- **Distribution:** Partition data and computations across multiple machines:
  - Move computations to clients (e.g., Java applets)
  - Decentralized naming services (e.g., DNS)
  - Decentralized information systems (e.g., WWW)
- **Replication:** Make copies of data available at different machines (e.g., replicated file servers, databases, mirrored websites, etc)
- **Caching:** Allow client processes to access local copies
  - Web caches (e.g., browser/web proxy)
  - File caches (e.g., server or client)

Scalability

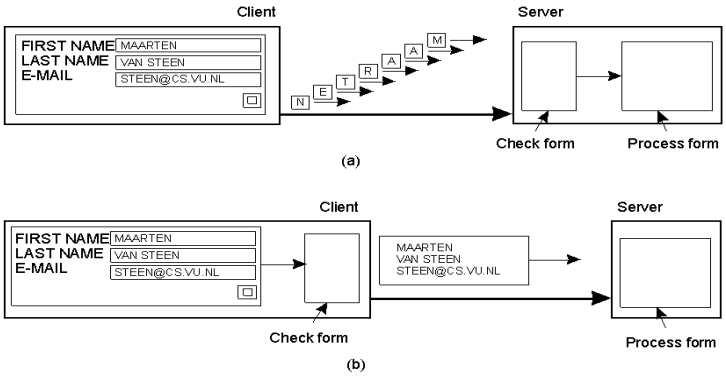
Applying scaling techniques is easy, except for:

- *Having multiple copies (cached or replicated), leads to **inconsistencies**: modifying one copy makes that copy different from the rest.*
- *Always keeping copies consistent and in a general way requires **global synchronization** on each modification. Global synchronization precludes large-scale solutions.*

Observation1: If we can tolerate inconsistencies, we may reduce the need for global synchronization.

Observation2: Tolerating inconsistencies is application dependent.

Scaling Techniques (Example 1)

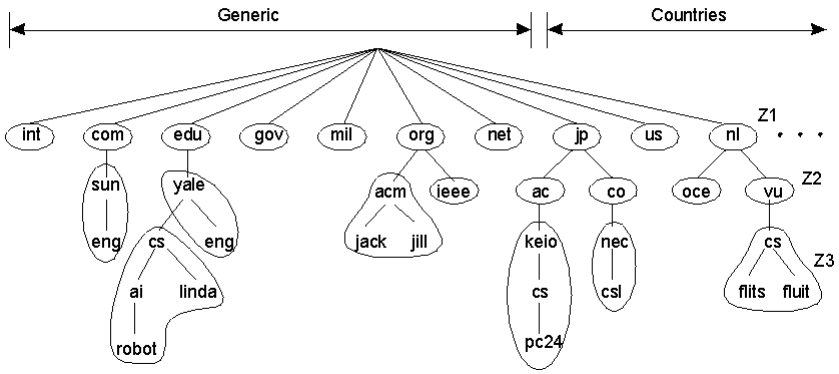


The difference between letting:

a) a server or

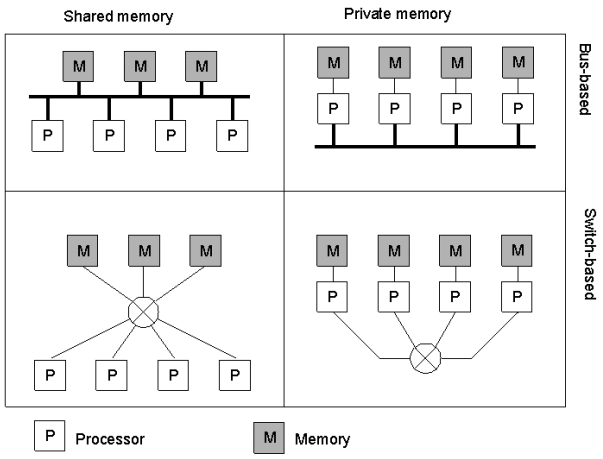
b) a client check forms as they are being filled

Scaling Techniques (Example 2)



An example of dividing the DNS name space into zones.

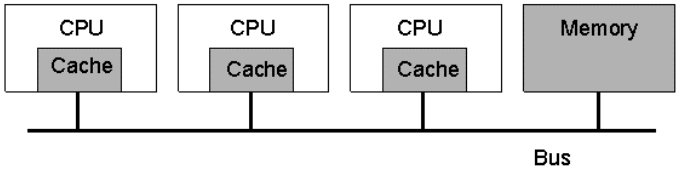
Hardware Concepts



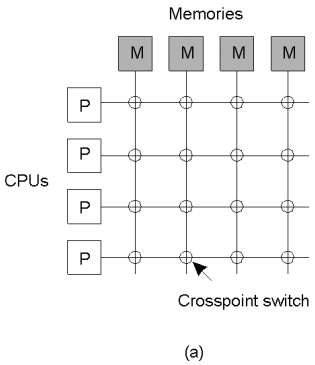
Different basic organizations and memories in distributed computer systems

Multiprocessors (1)

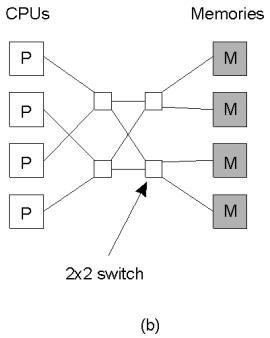
A bus-based multiprocessor.



Multiprocessors (2)

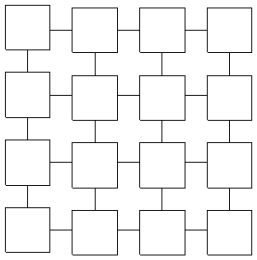


A crossbar switch

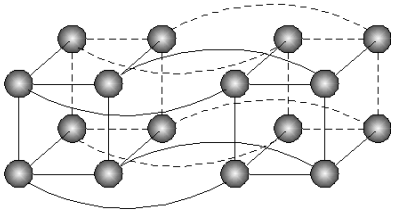


An omega switching network

Homogeneous Multicomputer Systems



(a) Grid



(b) Hypercube

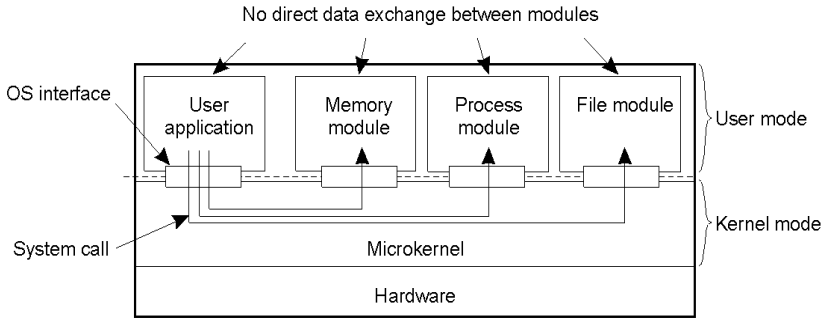
Software Concepts

Distributed systems can be achieved in 3 ways:

- DOS (Distributed Operating Systems)
- NOS (Network Operating Systems)
- 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

Uniprocessor Operating Systems

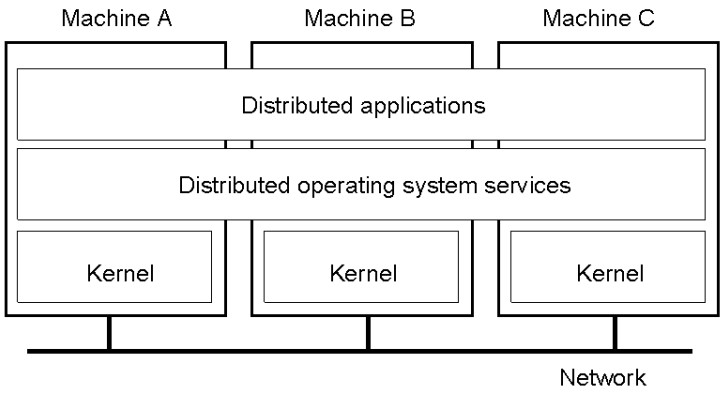


Separating applications from operating system code through a microkernel.

Multiprocessor Operating Systems

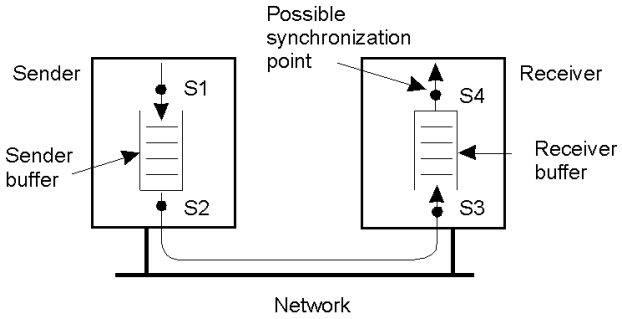
- Mutual exclusion and synchronization:
  - Semaphores?
  - Test-n-set and Swap instructions?
  - Spin locks?
  - Monitors?
  - A combination thereof?
- Implementation issues:
  - Shared memory?
  - Message passing?
  - A combination thereof?

Multicomputer Operating Systems (1)



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

Multicomputer Operating Systems (2)



- No shared memory → message passing
- Typically no broadcasting, thus need software
- Hard(er) to do synchronization
- No centralized decision making
- In practice, then, only very few truly distributed multicomputer OSs exist (Amoeba? Authors? ☺)

Multicomputer Operating Systems (3)

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

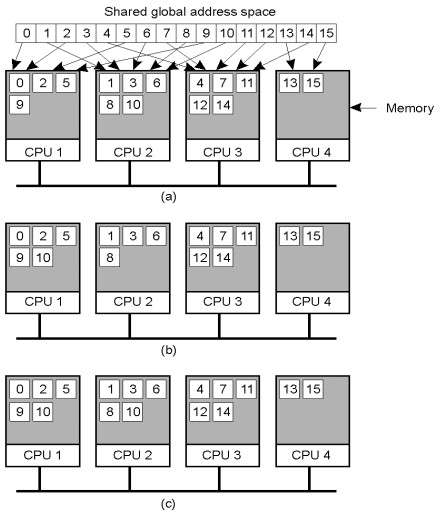
Relation between blocking, buffering, and reliable communications.

Distributed Shared Memory Systems (1)

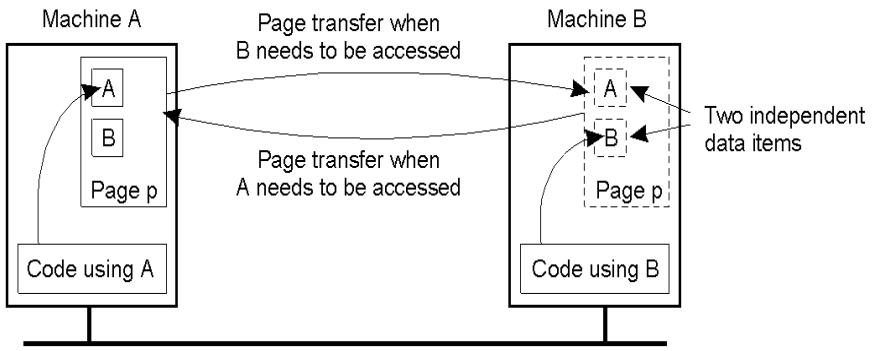
a) Pages of address space distributed among four machines

b) Situation after CPU 1 references page 10

c) Situation if page 10 is read only and replication is used



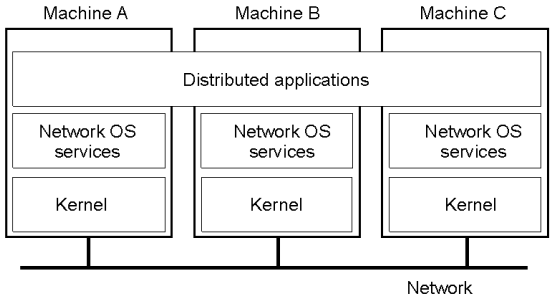
Distributed Shared Memory Systems (2)



False sharing of a page between two independent processes.

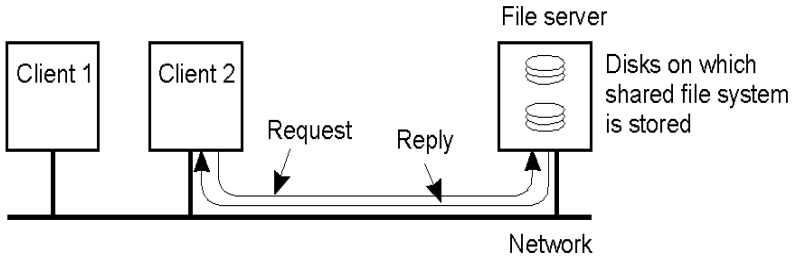
Network Operating System (1)

- Each computer has its own operating system with networking facilities
- Computers work independently (i.e., they may even have different operating systems)
- Services are tied to individual nodes (ftp, telnet, WWW)
- Highly file oriented (basically, processors share *only* files)



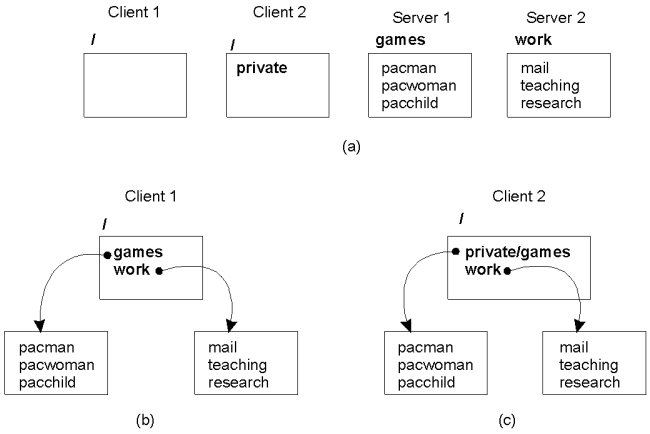
Network Operating System (2)

Two clients and a server in a network operating system.

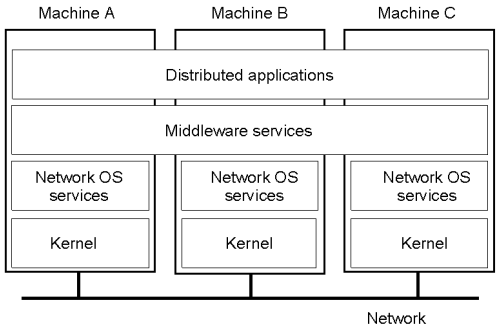


Network Operating System (3)

Different clients may mount the servers in different places.



Middleware



- OS on each computer need not know about the other computers
- OS on different computers need not generally be the same
- Services are generally (transparently) distributed across computers

Middleware

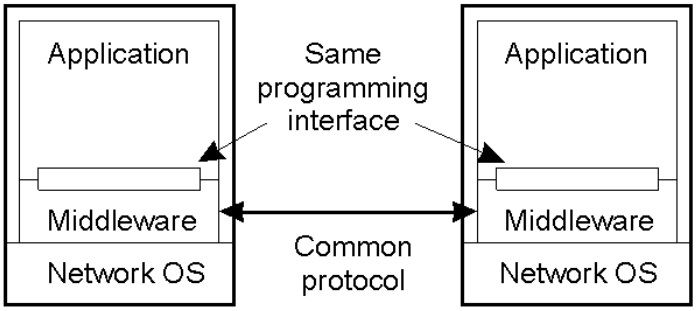
- **Motivation:** Too many networked applications were hard or difficult to integrate:
  - Departments are running different NOSs
  - Integration and interoperability only at level of primitive NOS services
  - Need for federated information systems:
    - Combining different databases, but providing a single view to applications
    - Setting up enterprise-wide Internet services, making use of existing information systems
    - Allow transactions across different databases
    - Allow extensibility for future services (e.g., mobility, teleworking, collaborative applications)
  - Constraint: use the existing operating systems, and treat them as the underlying environment (they provided the basic functionality anyway)

Middleware Services

- Communication Services: replace primitive sockets with
  - Remote Procedure Calls (or Remote Method Invocations)
  - Message passing
  - Communicating streams
- Information Services: data management
  - Large-scale, system-wide naming
  - Advanced directory services
  - Location services
  - Persistent storage
  - Data caching and replication
- Security Services: secure communication *and* processing
  - Authentication and authorization
  - Encryption

Middleware and Openness

Give applications control of when, where and how to access data (e.g., code migration and distributed transaction processing)



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.

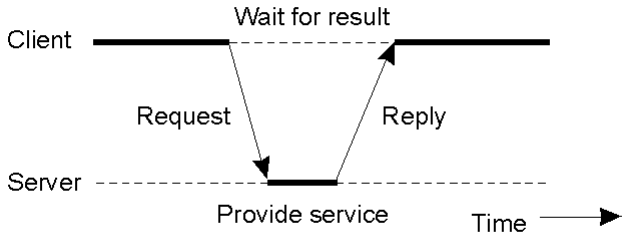
Comparison between Systems

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

A comparison between multiprocessor operating systems, multicomputer operating systems, network operating systems, and middleware based distributed systems.

Clients and Servers

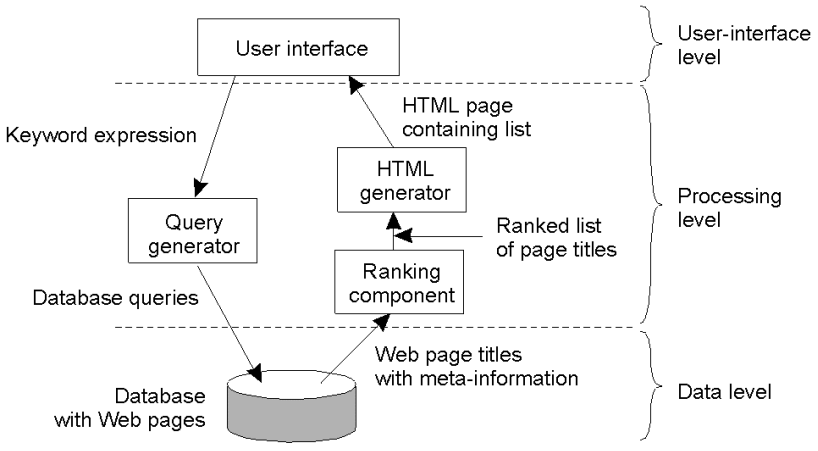
- Servers typically manage shared resource, more intensive use
- Clients are thinner, less resource intensive, provide interface, manage smaller (digital?) components (e.g., barcode readers)
- Clients and servers may be in different machines
- Request/reply model (blocking or not?)



Application Layering

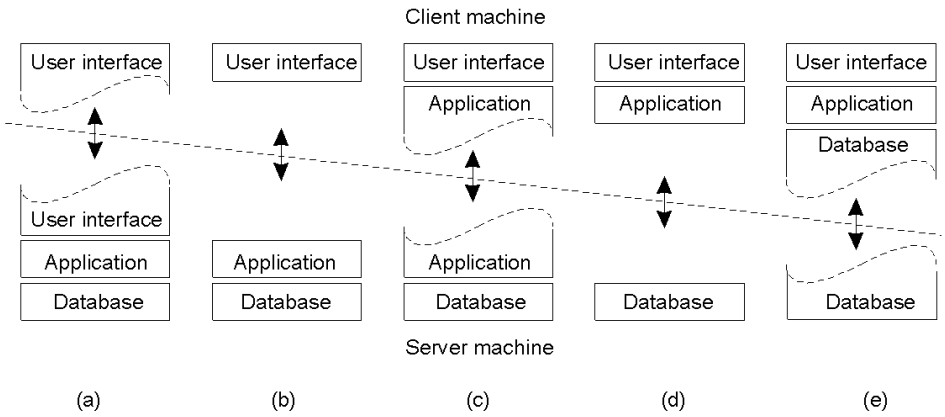
- Traditional three layers:
  - User interface (only interface)
  - Processing layer (no data)
  - Data layer (only data)
- Typical application: database technology
- Many others

Layering

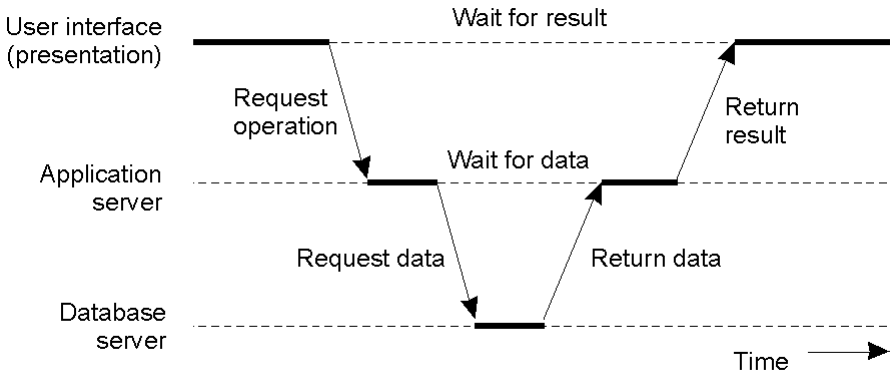


The general organization of an Internet search engine into three different layers

Alternative client-server organizations

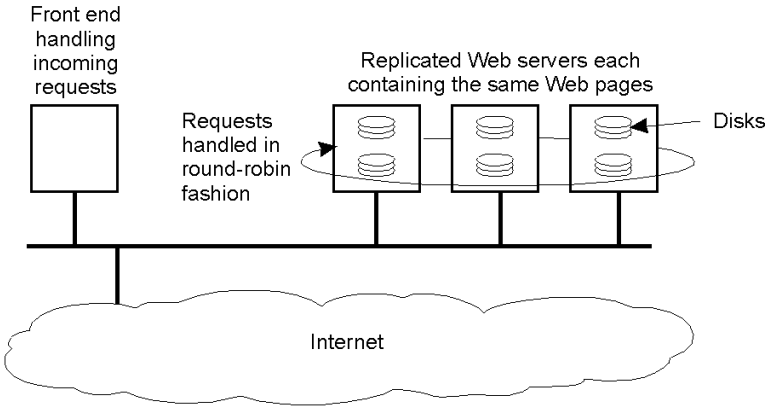


Example Architecture



An example of a server acting as a client.

Modern Architectures



An example of horizontal distribution of a Web service.

Alternative C/S Architectures

- **Cooperating servers:** Service is physically distributed across a collection of servers. E.g.,
  - Replicated file systems
  - Network news systems
  - Naming systems (DNS, X.500, …)
  - Workflow systems
- **Cooperating clients:** distributed application exists by virtue of client collaboration:
  - Teleconferencing where each client owns a (multimedia) workstation
  - Publish/subscribe (push/pull) architectures in which role of client and server is blurred