Distributed System

Definition

- A collection of independent computers
  - Autonomous machines
- Appear to the users as a single computer
  - Software aspect
  - Difference between computers and way of communication hidden from users
  - Internal organization hidden from users

Example

- Banking System
  - Master computer in each branch office for local accounts, transactions
  - Each computer can communicate with any other computer
  - Transactions without considering the location of account, customer
  - Users consider it as a single system
- Domain Name System (DNS)
  - Distributed lookup table
- Google
- Facebook
- Email Servers
  - SMTP

Advantages

- Economical Considerations
  - Grosch's Law: Computing power of a CPU $\propto$ (Price of a CPU)^2
  - No longer effective
  - Cost effective solution: Equip large number of cheap CPU's together
  - Achieve better price/performance ratio using microprocessors than mainframes
- Speed
  - More total computing power than a mainframe
  - Reducing processing bottlenecks
- Inherently Distributed Applications
  - Supermarket chain with local stores, inventories, sales, decisions
- Reliability and Fault Tolerance
  - Single chip/machine failure will not crash the whole system
  - Ability to continue work in the presence of failures
  - Continuously available
- Incremental Growth
  - Adding more processors to the system to increase the system's performance
- Scalability
- Resource Sharing
  - Users access a common database and share expensive peripherals

Advantages

- Communication
  - Easier human to human communication: Email
- Flexibility
  - Spread the workload over the all available machines
- Interaction between user and resource
  - Uniform
  - Consistent

Disadvantages

- Complex
  - Tough to provide suitable software support
- Networking
  - Message loses: Special software to recover
  - Overloaded
  - Saturation: Either replaced or added
  - Failure
- Security
  - Access to secret, confidential data

Goals

- Connecting Users and Resources
  - Make it easy for users to access remote resources
  - Resources: Computer, Printer, Storage facilities, Data, File
  - Cost effectiveness: Sharing a costly resource (printer) with several users is more cost effective than using different resource for each user
  - Easier to collaborate and exchange information: Exchanging mails, files, documents
- Transparency
  - Hiding the fact that processes and resources are physically distributed across multiple computers
  - What is a transparent computer system?

  Forms of transparency
  - Access: Hiding how a resource is accessed

    Hiding differences in data representation (Little Endian to Big Endian and vice versa)

    Hiding how files are manipulated

    Hiding differences in naming conventions

    Different OS
  - Location: Hiding physical location of resources

    Achieved by assigning logical names to resources
  - Migration: Hiding movement of a resource (data) to another location
  - Relocation: Hiding movement of a resource (data) to another location while it is in use

    Using mobile/laptop while moving from one place to another without getting disconnected

Goals

- Transparency

  Forms of transparency

  - Replication: Hiding the fact that a resource is replicated
    Hiding the fact that several copies of a resource exist
    All replicas should have the same name
    Helps increasing availability, performance: By placing a copy close the place where it is accessed
    What is the relation between replication transparency and location transparency?
  - Concurrency: Hiding concurrent access of resources
    Cooperative resource sharing
    Competitive resource sharing
    Consistency achieved through locking mechanism
  - Parallelism: Hiding parallel activities
  - Failure: Hiding failure of a resource and recovery from failure
  - Persistency: Hiding whether a resource is in volatile memory or disk
  - Scaling: Hiding the fact that the system has been scaled (no change in system structure, algorithm)
  - Performance: Hiding the fact that the system has been reconfigured to improve performance

Goals

- Transparency

  Degree of transparency

  - Transparency effects performance
  - How?

- Openness
  - Should be flexible
  - Easy to configure with different components
  - Easy to add or replace components
  - Easy to extend, re-implement in various ways
  - Interoperability and Portability

- Scalability

  With respect to size: Ease of adding more users and resources

  Problems

  - Centralized Services: Single server for all users

    Many clients and many requests

    Bottleneck problem

    Example: Bank Account, Medical Record, Personal Loan, Result, Mail Server

Goals

- Scalability

  With respect to size: Ease of adding more users and resources

  Problems

  - Centralized Data: Single database

    Example: Online telephone book, DNS (Distributed data)
  - Centralized Algorithms: Single algorithm

    Routing based on complete information

    Overloading the network

  Characteristics of decentralized algorithms

  - Complete information about system state is not available to any machine
  - Machines make decisions based only on local information
  - Failure of one machine does not ruin the algorithm
  - No assumption about global clock

  Geographical scalability: Distant users and resources

  Problems

  - Synchronous Communication
    - Client blocks until a reply is sent back
    - LAN requires a few hundred microseconds for communication between two machines

Goals

- Scalability

  Geographical scalability: Distant users and resources

  Problems

  - Synchronous Communication
    - WAN requires hundreds of milliseconds for such communication (considering IPC)
    - Tough to scale existing LANs with synchronous communication
  - Reliability
    - LANs provide with reliable communication based on broadcasting
    - WANs are inherently unreliable and point to point
    - Tough to scale existing LANs maintaining reliability

  Administrative Scalability: Scale a distributed system across multiple, independent domains

  - Conflict policies to manage resources and handle security issues

Scaling Techniques

- Hiding communication latencies
  - Geographical scalability
  - Try to avoid waiting for responses to remote service requests
  - Applicable to asynchronous communication (batch processing, parallel applications)
  - For synchronous communication: Reduce the overall communication

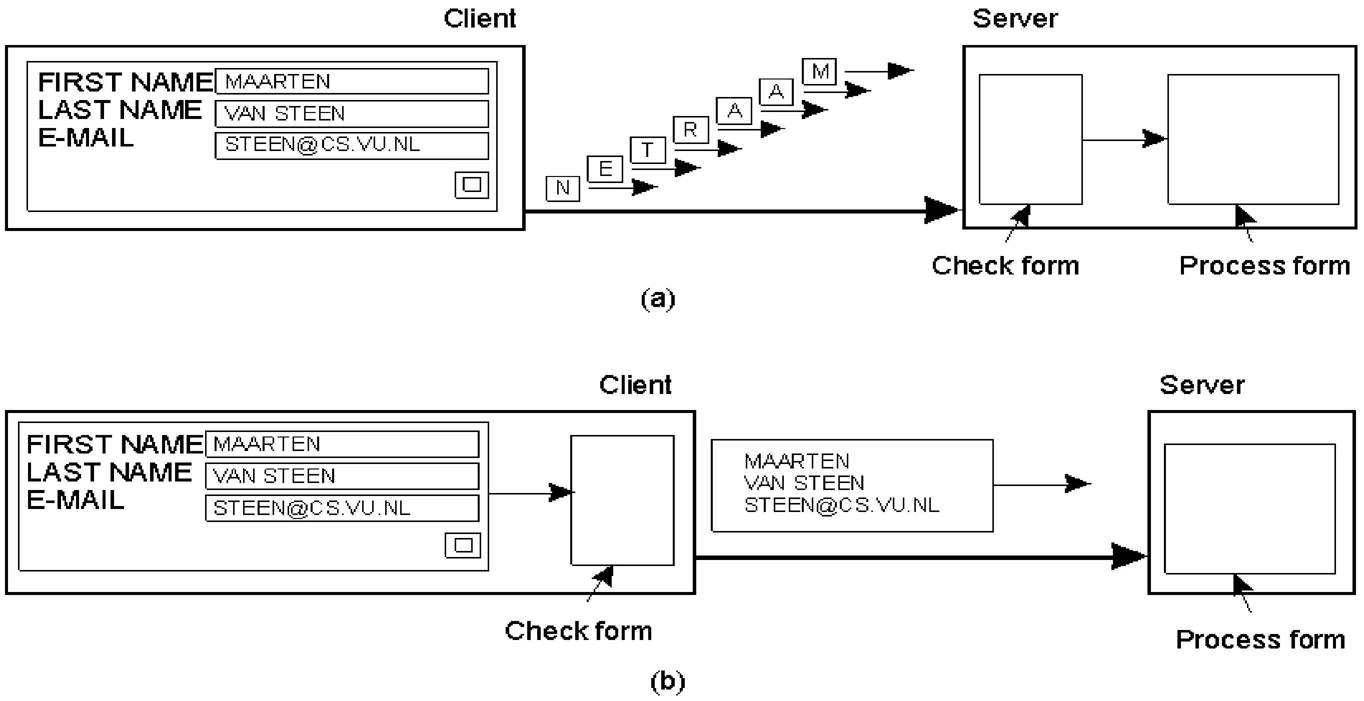


Figure 1: The difference between letting (a) a server check forms or (b) a client check forms as they are being filled

Scaling Techniques

- ❑ Hiding communication latencies
  - Shipping code: Java applets
- ❑ Distribution
  - Taking a component, splitting it, spreading those across the system
  - Example: DNS

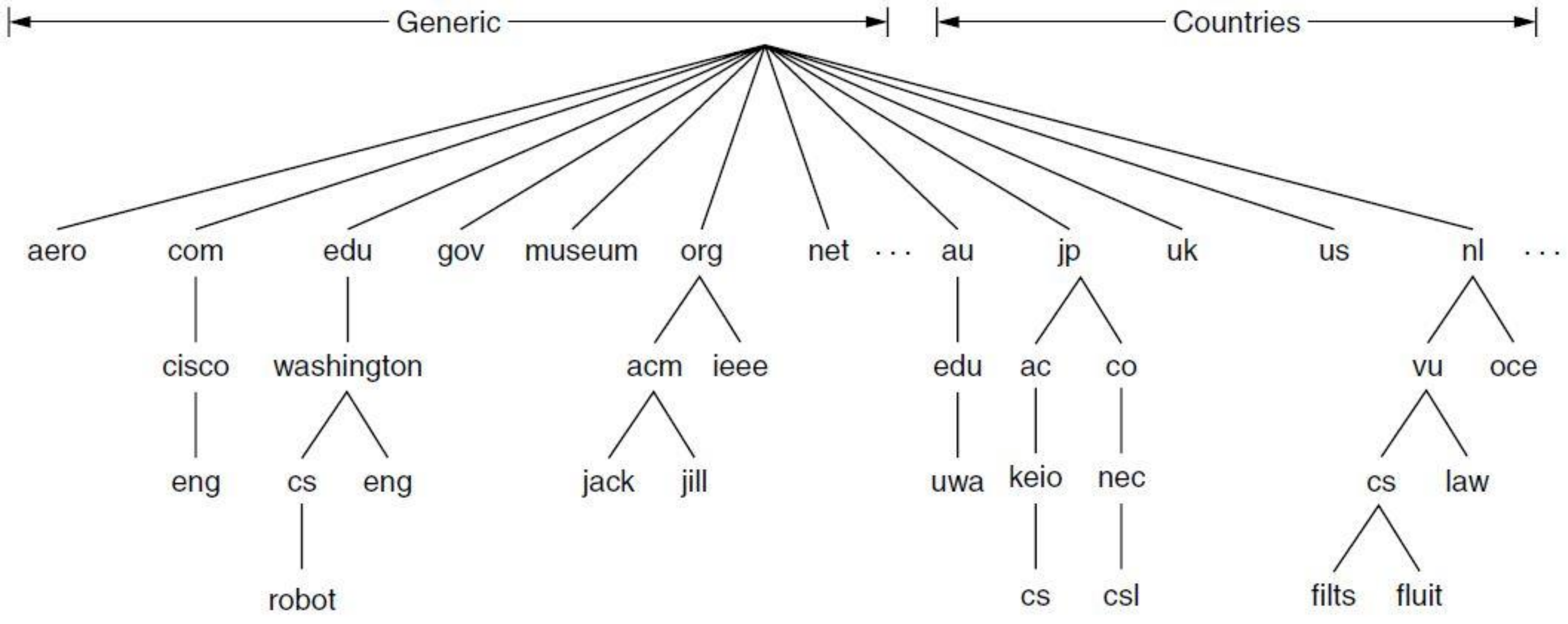


Figure 2: Example of dividing the DNS name space

Scaling Techniques

- ❑ Distribution

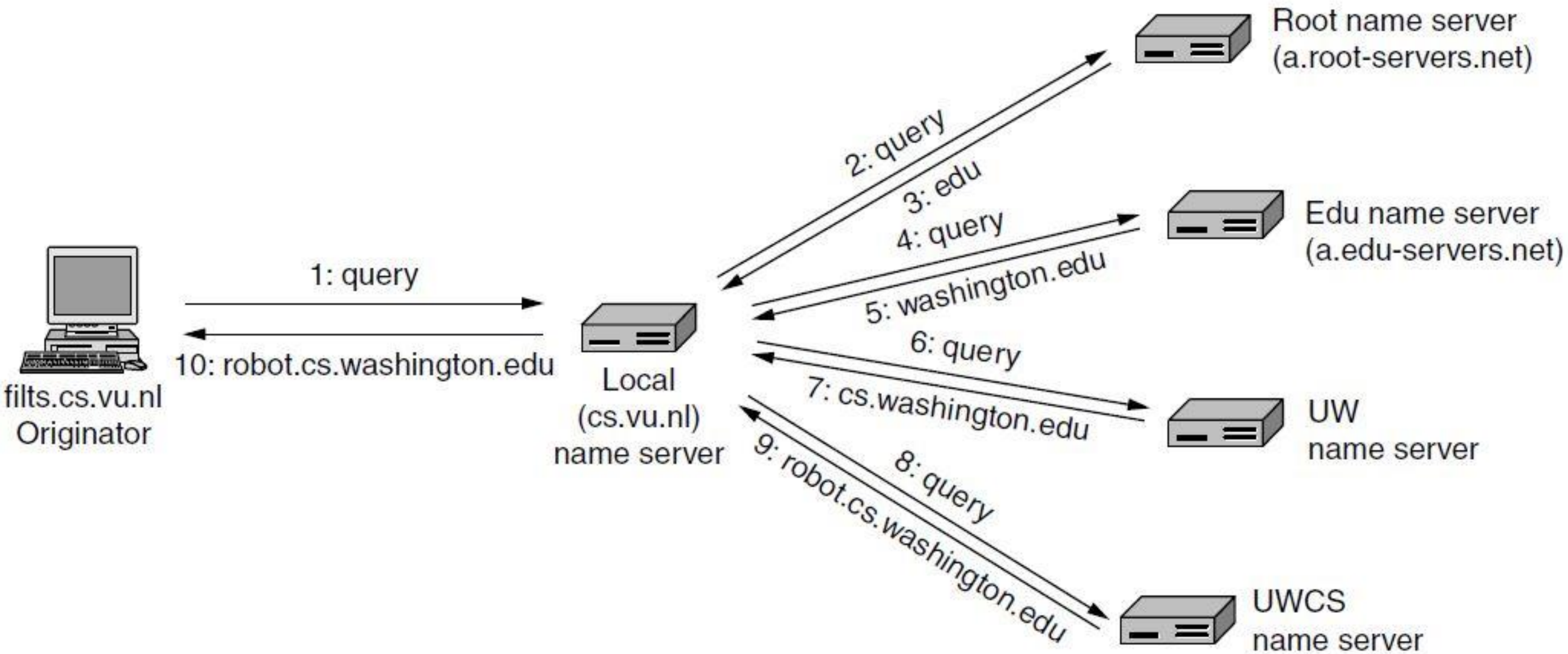
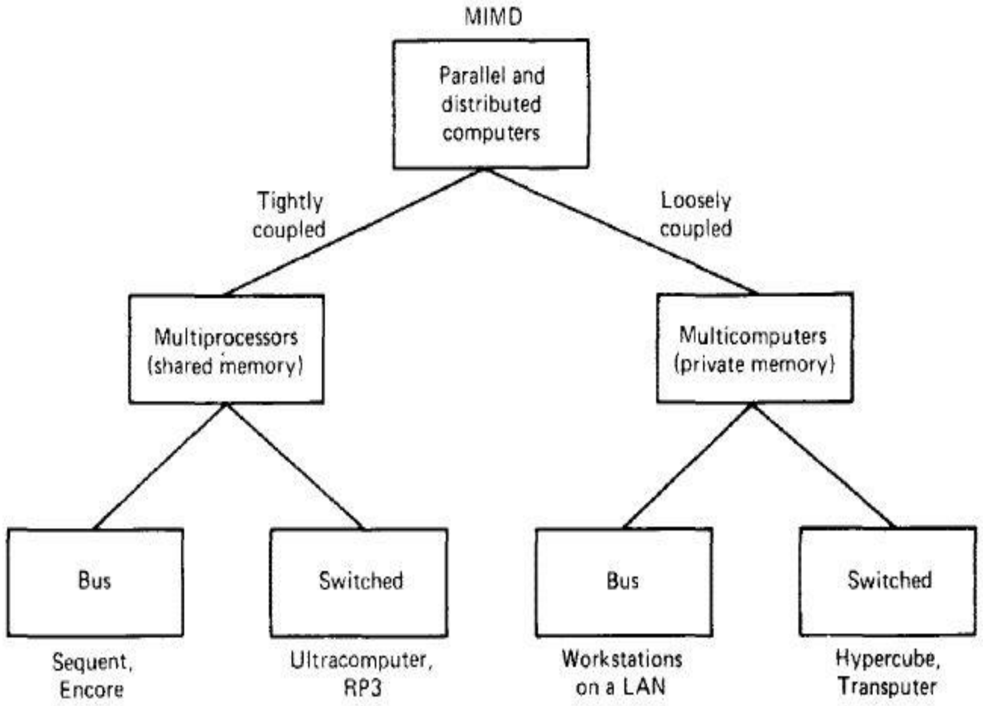


Figure 3: Example of a resolver looking up a name

Scaling Techniques

- Replication/ Caching
  - Replicate components across a distributed system
  - Making a copy of resource
  - Increases availability
  - Helps to balance the load
  - Better performance
  - Having a copy of resource nearby helps hiding communication latency
  - Caching is a special form of replication
  - Caching decision made by the client
  - Problem: Consistency
  - Modifying one copy and leaving rest unmodified

Hardware Concepts



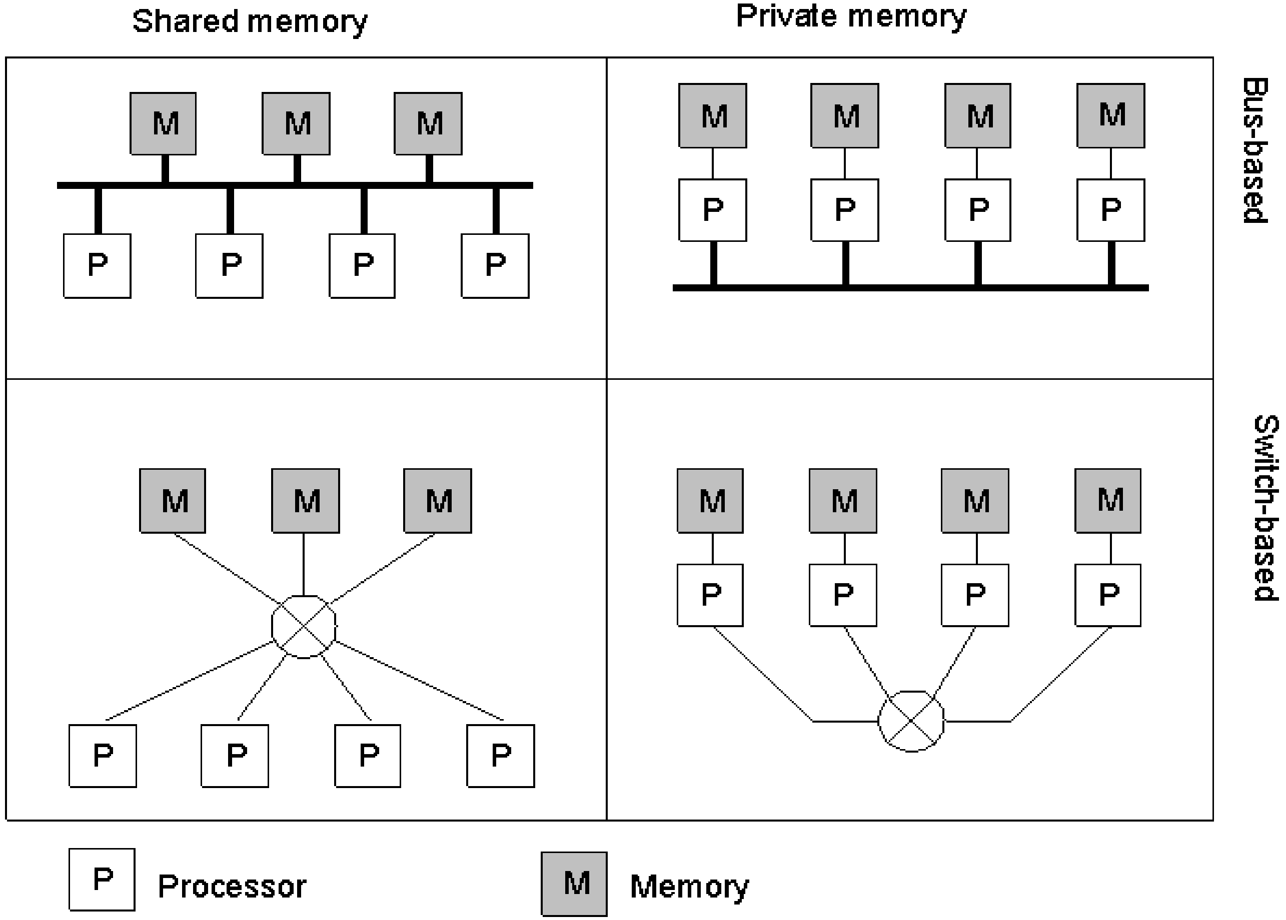


Figure 4.1: Taxonomy of parallel and distributed computer systems

Figure 4.2: Basic organizations of processors and memories in distributed computer systems

Hardware Concepts

- ❑ Multiprocessor: Single physical address space (PC)
- ❑ Multicomputer: N physical address space (PCs connected through a network)
  - Homogeneous
  - Heterogeneous: Connection of different types of independent computers
- ❑ Bus-based: Single backbone/connection medium (Cable television connection)
- ❑ Switch-based: Individual wires with wiring patterns from machine to machine (World wide public telephone system)

Bus-based multiprocessor

- Coherency: CPU-A reads a location of memory just after CPU-B writes to memory (only 2 CPUs)

  With many CPUs trying to read and write simultaneously the bus gets overloaded, performance degrades.

  Solution: Adding cache memory to CPU

  Cache holds most recently accessed words

  While processing a request at first the cache is checked whether it contains the requested word

  If found in cache then no bus request is made

  Hit rate: If a requested word is found in cache

  Does higher cache size guarantee higher hit rate?

Hardware Concepts

Bus-based multiprocessor

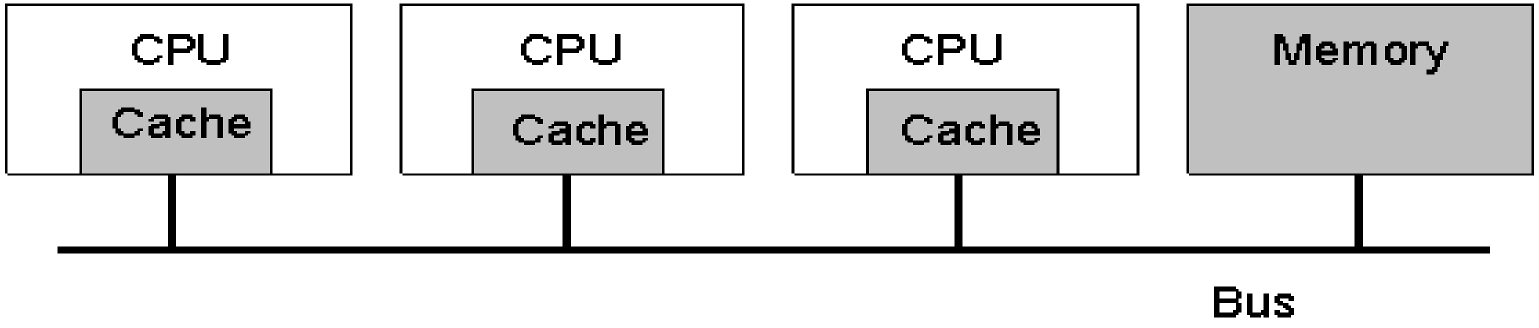


Figure 5: Bus-based multiprocessor

- Coherency: Cache coherency problem
  - CPU-A reads X from memory, gets 10, caches it
  - CPU-B reads X from memory, gets 10, caches it
  - CPU-A writes 20 to X, updates local cache
  - CPU-C reads X
- Problem: Limited scalability

Cache-A	Cache-B	Cache-C	Memory
10	-	-	10
10	10	-	10
20	10	10	10

Hardware Concepts

Switch-based multiprocessor (Crossbar)

- Connection between each CPU and memory
- Crosspoint switch at every intersection
- Crosspoint switch-Opened/Closed
- Crosspoint closed while accessing memory
- Many CPUs can access the memory simultaneously
- Can many CPUs access the same memory simultaneously?
- n CPUs and n memories: n^2 number of switches
- Problem: A lot of switches required when the number of CPU/ Memory is huge

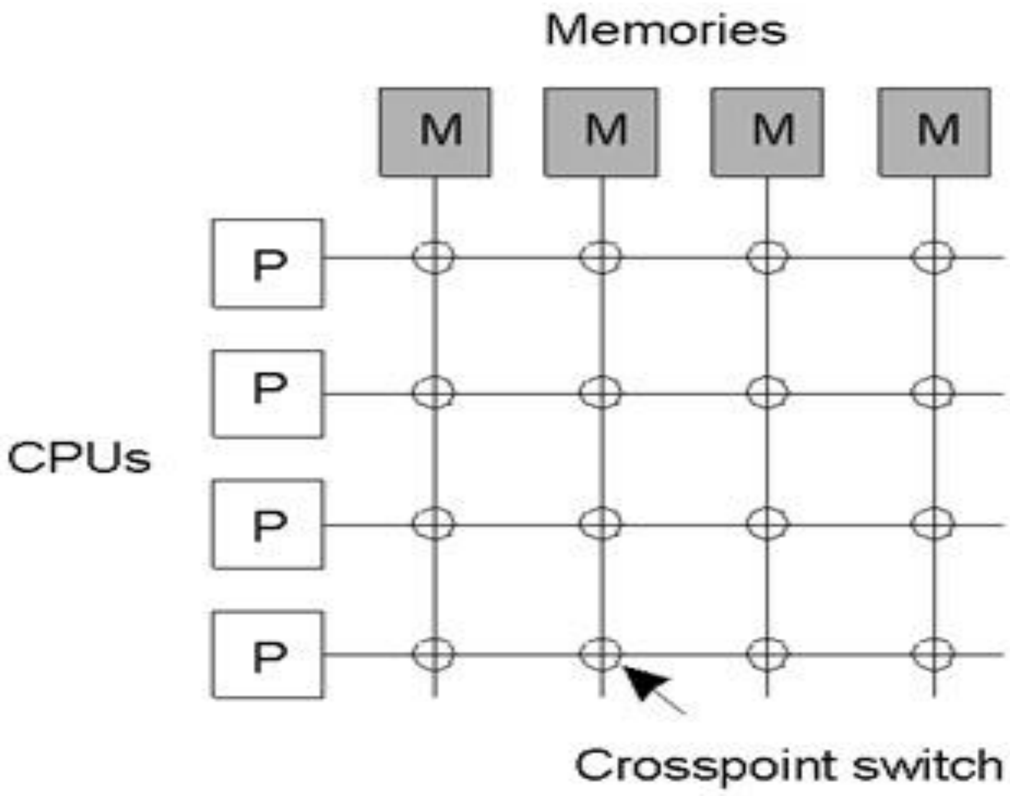


Figure 6: Crossbar switch

Hardware Concepts

Switch-based multiprocessor (Omega)

For n CPUs and n memories

- Switching stage: $\log_2 n$
- Each stage contains: $\frac{n}{2}$ switches
- Total switches: $\frac{n \log_2 n}{2}$
- Which is bigger? n^2 or $\frac{n \log_2 n}{2}$
- Problem: Delay

For n=1024 with read operation

Switching stages: $\log_2 1024 = 10$ (CPU to Memory)

Switching stages: $\log_2 1024 = 10$ (Memory to CPU)

Total switching stages: 20

If instruction execution time is 10 nsec

then it should perform 20 stages in 10 nsec

The switching time must be 0.5 nsec for each switch

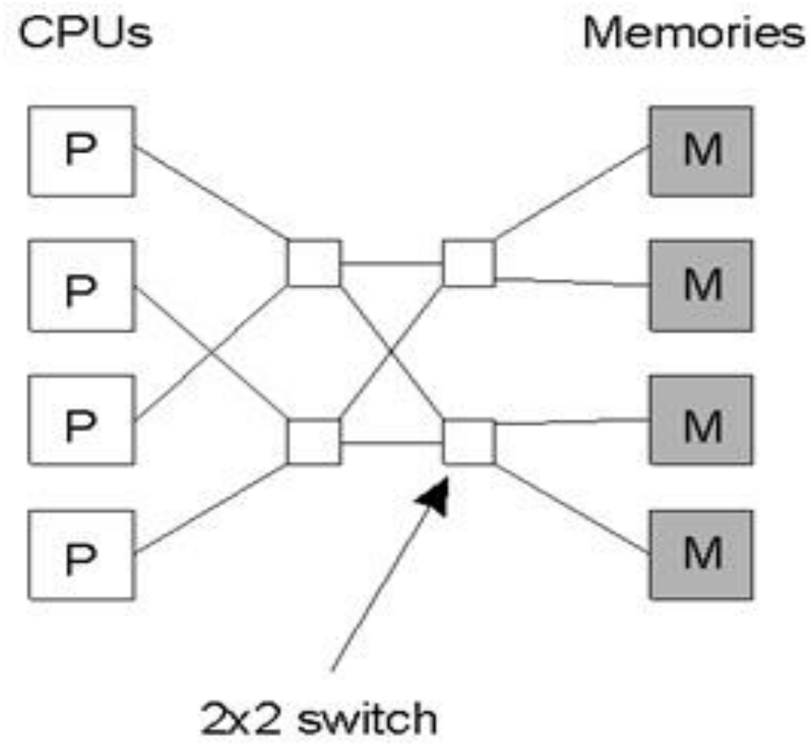


Figure 7: Omega switching network

Hardware Concepts

Switch-based multiprocessor (Omega)

- Solution: NUMA, CC-NUMA

Bus-based multicomputer (Homogeneous)

Switch-based multicomputer (Homogeneous)

Heterogeneous multicomputer system

- Lack of global system view
- An application can not assume that the same performance or services are available everywhere
- Sophisticated software needed to build applications for heterogeneous multicomputers
- Distributed systems provide a software layer which shields applications from what is going on at the hardware level

Software Concepts

- Determines the view of a distributed system
- Operating systems for distributed computers
  - Tightly Coupled Systems
    - OS maintains single global view of resources
    - Manages multiprocessors and multicomputer (homogeneous) systems
    - Hides the complications of managing hardware
    - Distributed Operating System (DOS)
      - Uniprocessor Operating Systems
        - Kernel mode, user mode
        - Tough to adapt, replace OS components
        - Organize the OS into two parts: Module to manage H/W in user mode, microkernel to be executed in kernel mode
      - Multiprocessor Operating Systems
        - Semaphore
        - Mutex
        - Futex
        - Monitor
        - Condition variables

Software Concepts

- Operating systems for distributed computers
  - Tightly Coupled Systems
    - Distributed Operating System (DOS)
      - Multicomputer Operating Systems
        - Communication through message passing
        - Separate module for IPC
        - Common layer of S/W to support parallel and concurrent execution of various tasks
        - S/W implementation of shared memory
        - Masking H/W failures
        - Providing transparent storage

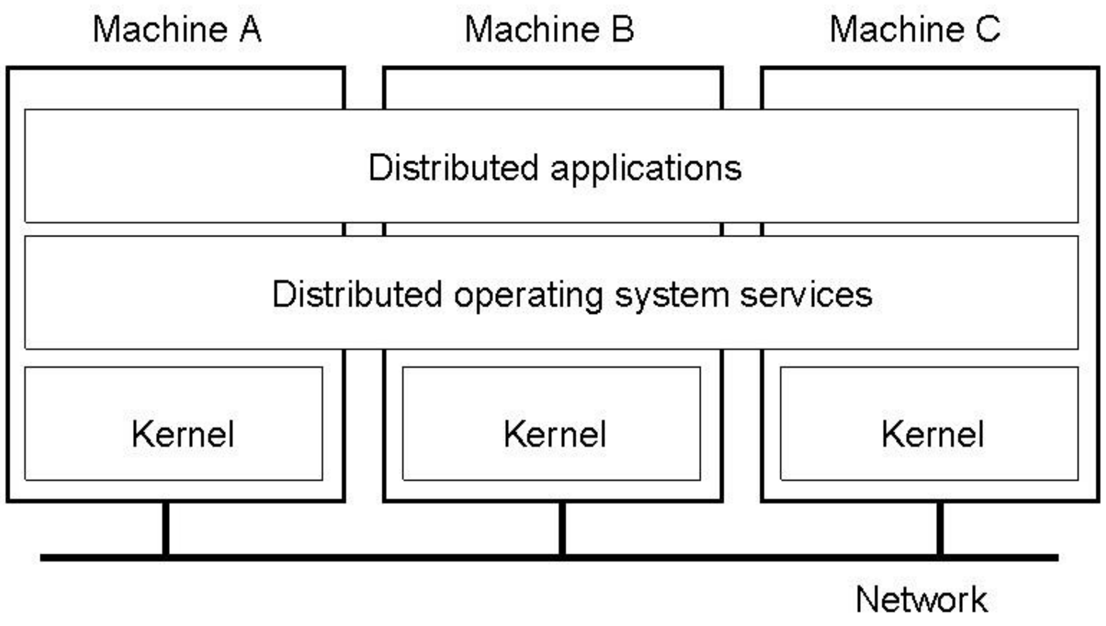


Figure 8: General structure of a multicomputer OS

Software Concepts

- Operating systems for distributed computers
  - Tightly Coupled Systems
    - Distributed Operating System (DOS)
      - › Distributed Shared Memory Systems
        - Programming multiprocessors is easier than programming multicomputers
        - Emulating shared memory on multicomputers
        - Page based DSM
        - Address space divided into pages
        - Pages spread all over the processors
        - Trap occurs when processor references an address not present locally
        - OS fetches the page
        - Remote RAM used as backing store
        - Example
          16 pages and 4 processor, replication (read only)
        - False Sharing: Same page contains data of two independent processes from two different processors
        - How to determine the page size?

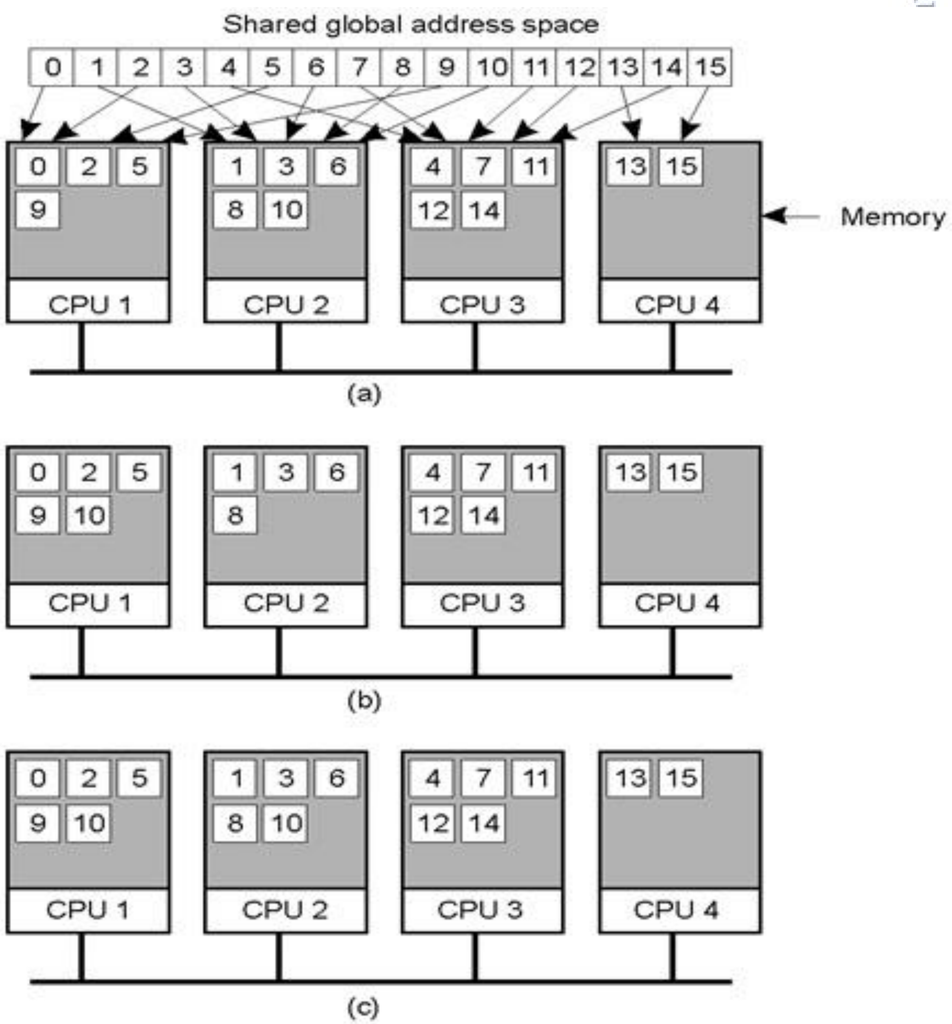


Figure 9: (a) Pages distributed
(b) CPU 1 references page 10
(c) Replication used

Software Concepts

- Operating systems for distributed computers
  - Loosely Coupled Systems
    - Network Operating System (NOS)
    - Manages multicomputer (heterogeneous) systems
    - Each machine runs with own OS
    - OSs help to make local services and resources available to others
    - Allows a user to log into another machine remotely
    - Allows to copy files from one machine to another
    - Easy to add or remove a machine
    - Can not provide single global view

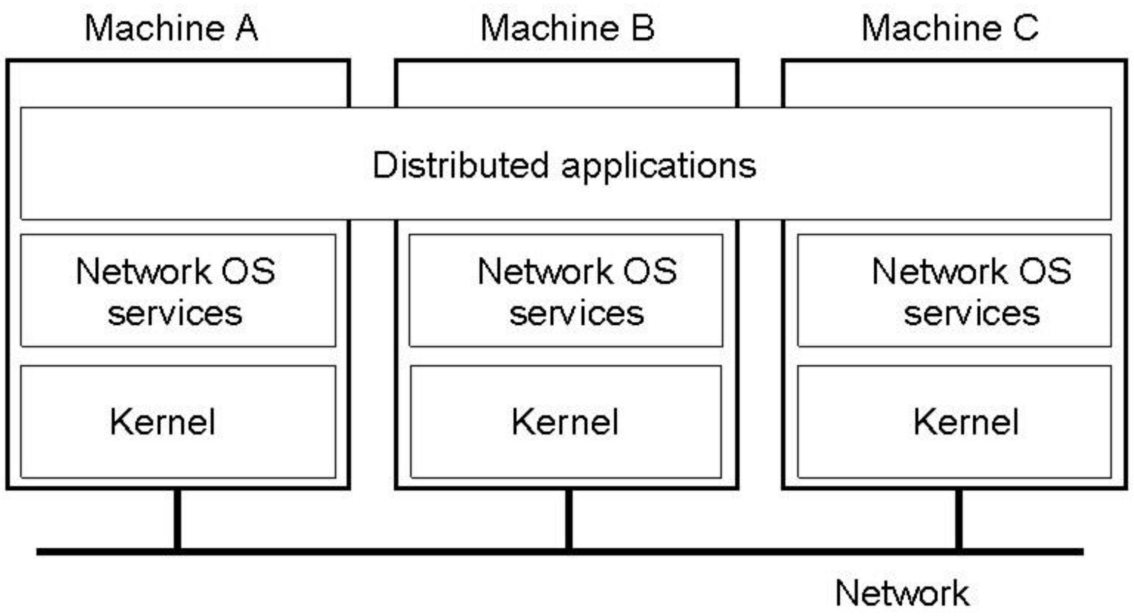


Figure 10: Structure of a NOS

Software Concepts

- Middleware
  - Additional layer of S/W
  - Liaison between application and network OS
  - Extends over multiple machines
  - Provides distribution transparency
  - Hides heterogeneity
  - Transparent access to remote data (distributed file systems, distributed database)
  - Offers high level communication
  - Hides low level message-passing
  - Same protocol and interfaces in open middleware-based distributed system

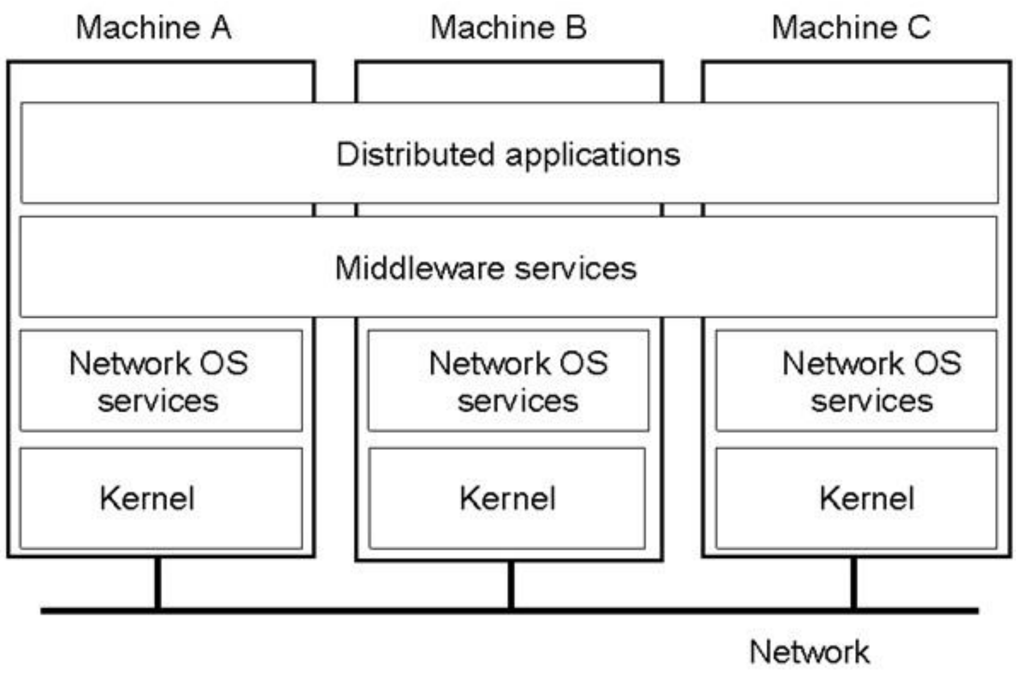


Figure 11: Structure of a distributed system as middleware

Software Concepts

- Comparison between multiprocessor OS, multicomputer OS, NOS, middleware-based distributed system

Item	Distributed Operating System		Network OS	Middleware-based DS
	Multiprocessor	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

Centralized Architectures

Two-tiered architecture

- ❑ Processing divided into two groups
- ❑ Client sends request to server for any service and waits for the server to reply
- ❑ Server processes client's request: Database service
- ❑ Can be implemented using both connection oriented and connectionless protocol
- ❑ Connection oriented protocol: Costly connection set up and release
- ❑ Connectionless protocol: Transmission failures
- ❑ Three different levels
  - User-interface level
    - Programs which allow users to interact with applications
    - Handles interaction with user
  - Processing level
    - Core functionality of an application
  - Data level
    - Operation on database or file system
    - Maintain consistency of data
- ❑ Problem: Server handles everything

  Not properly distributed

  Bottleneck problem

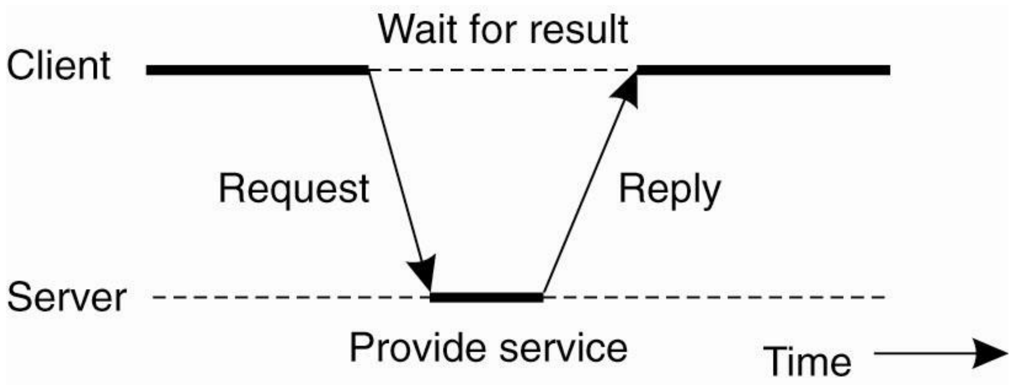


Figure 12: Interaction between a client and a server

Centralized Architectures

Three-tiered architecture

- User interface
  - Data presented to user
  - Input taken from user
  - Interaction with user
- Application Server
  - Interface between user and database
  - Validation, calculation of data
- Database Server
  - Manipulation of the database

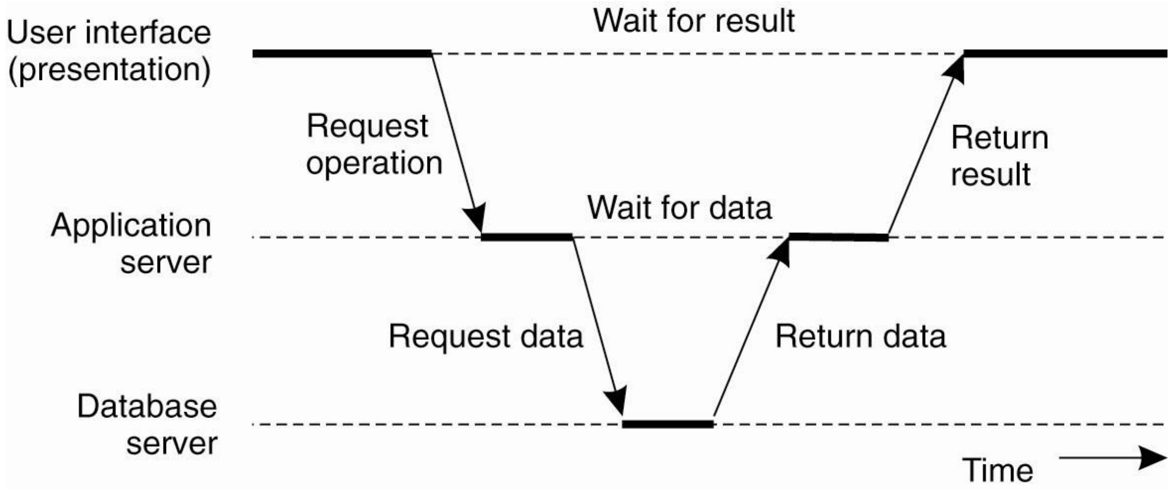


Figure 13: Example of a server acting as client

Decentralized Architectures

- Vertical Distribution
  - Logically different components are placed on different machines
- Horizontal Distribution
  - Client/Server physically split into logically equivalent parts
  - Each part operates on own share of complete dataset
  - Ensures better load balancing

Peer-to-peer

- Centralized
  - Napster
- Distributed
  - Gnutella
- Combination
  - KaZaA

Peer-to-peer

- Centralized
  - Napster
    - Centralized directory server
    - Locating content centralized
    - File transfer decentralized
    - Performance bottleneck
    - Single point of failure

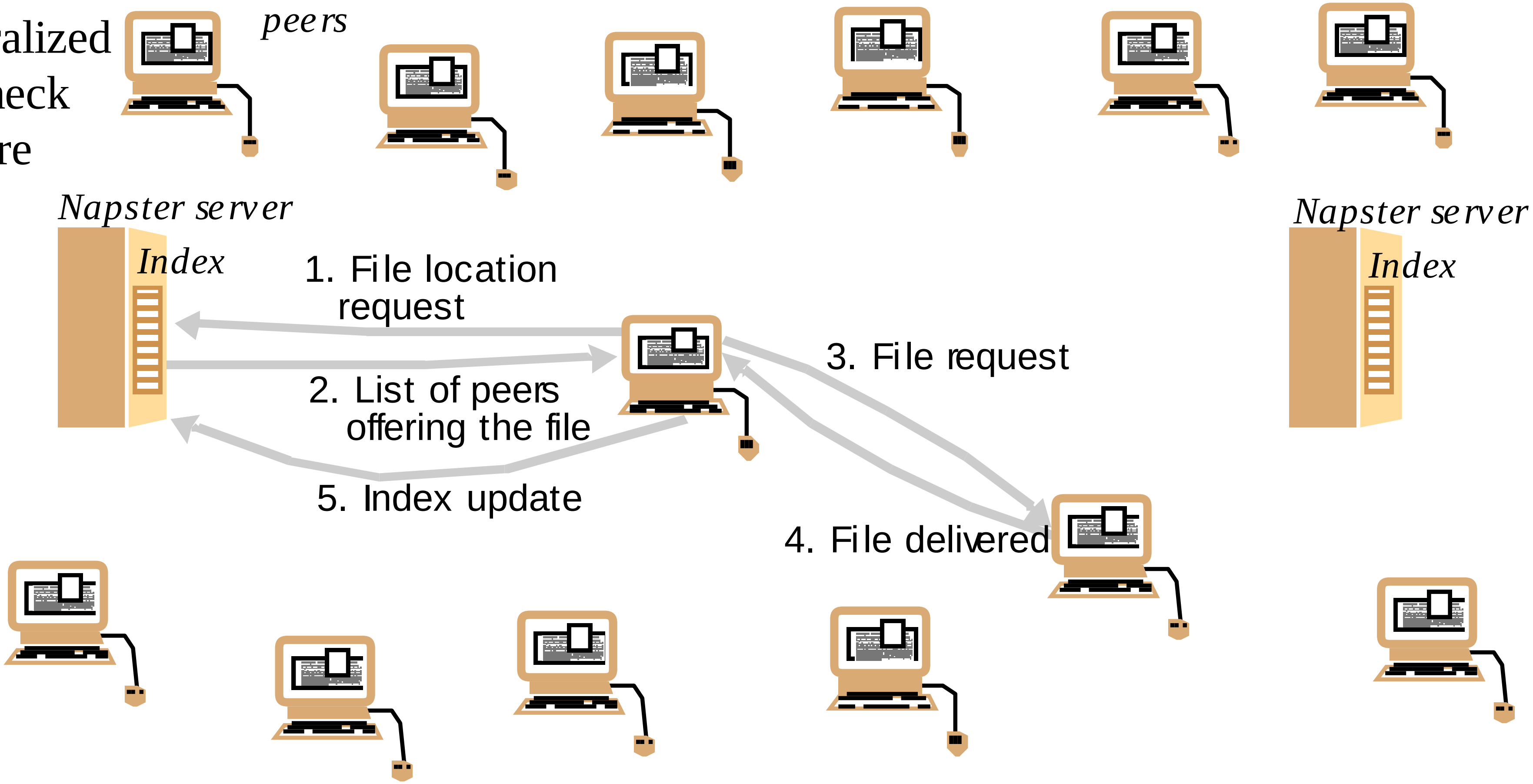


Figure 14: Peer-to-peer file sharing (Napster)

Peer-to-peer

- Distributed
  - Gnutella
    - Query flooding
      - Peer sends file request to all its neighbors
      - If these neighbors do not have the file, they send the request to their neighbors and so on
      - Query hit sent over reverse path
      - Scalability problem
    - No central server
    - Scalability issue

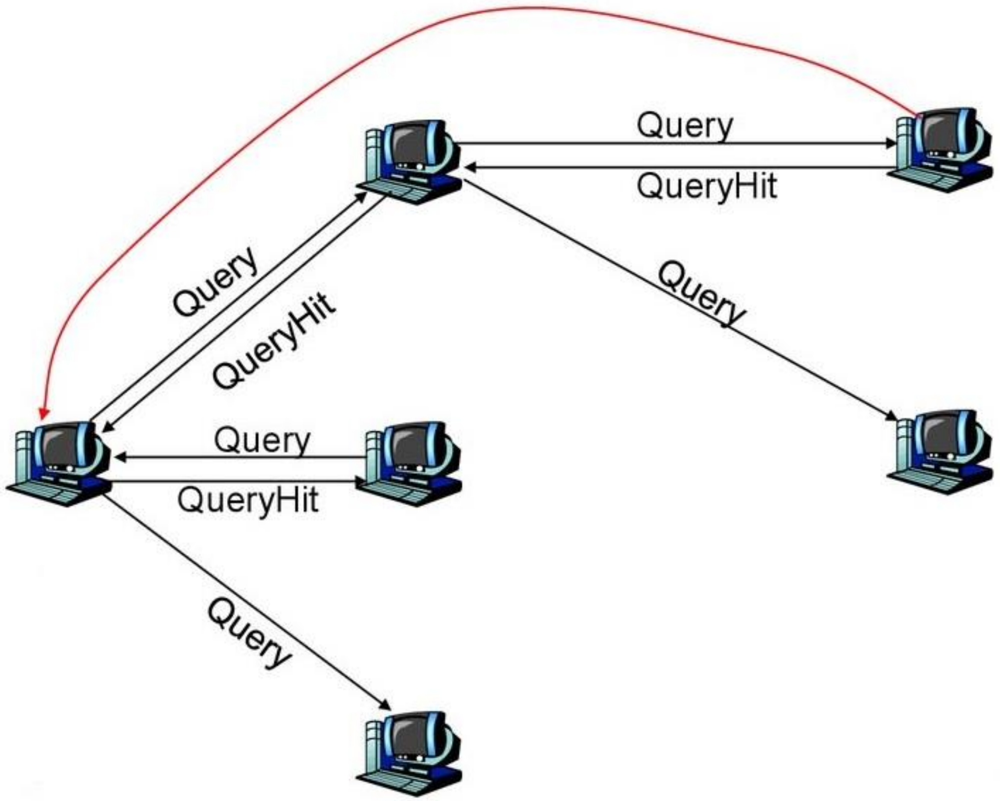


Figure 15: Peer-to-peer file sharing (Gnutella)

Peer-to-peer

- ❑ Combination
    - KaZaA
        - Exploiting heterogeneity
            - All peers are not equal, Some have higher priority
            - Two types of peers: Group leaders, assigned to group leaders (children)
            - Group leaders have the track of all peers assigned to them
            - Group leaders are interconnected
        - Peers send file request to group leaders
        - If requested file is found within the group then group leader responds
        - If not found then the request is forwarded to other group leaders and so on
        - Other group leaders process the request and respond or forward
- ❑ How does BitTorrent and µTorrent work?

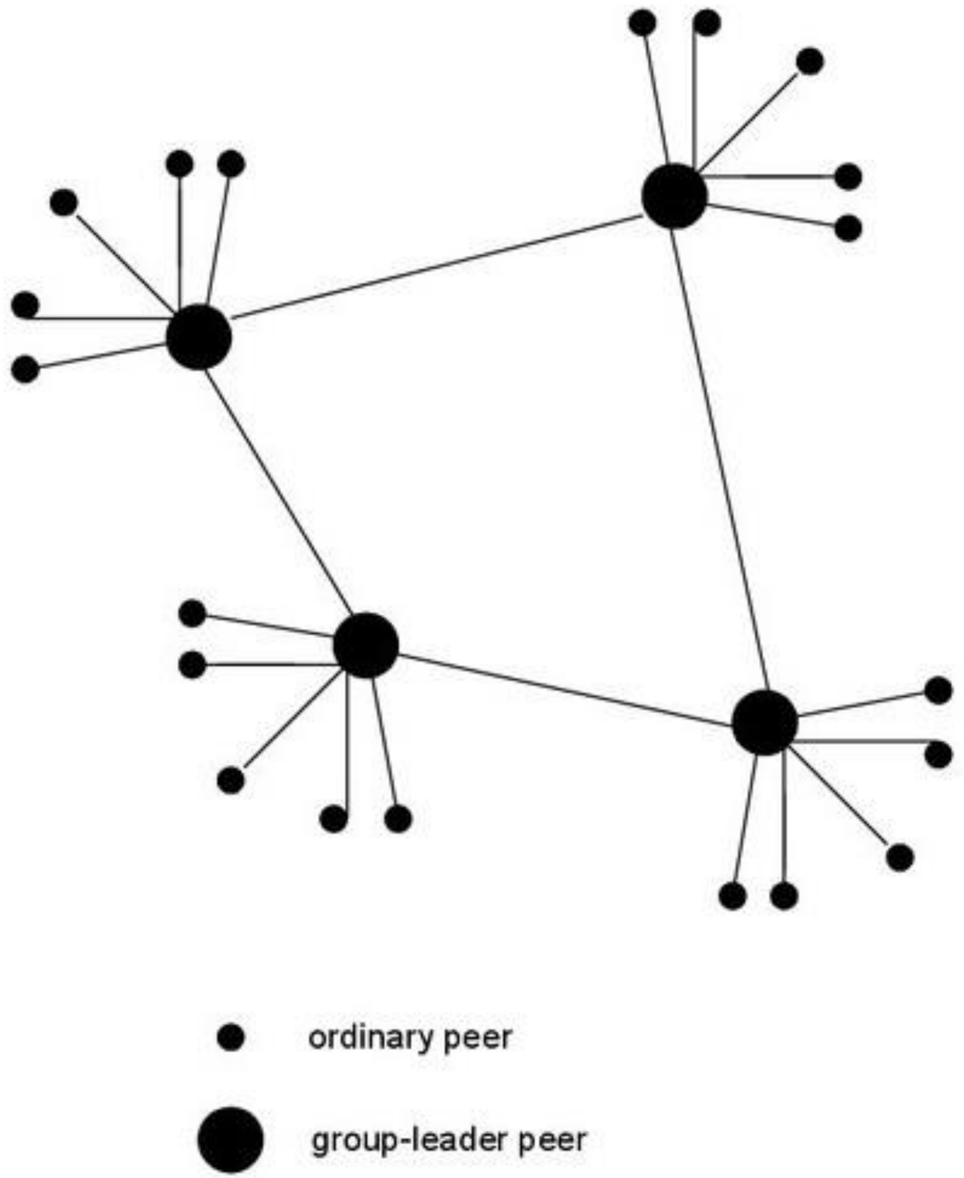


Figure 16: Peer-to-peer file sharing (KaZaA)