Introduction

Distributed Systems

Bachelor In Informatics Engineering

Universidad Carlos III de Madrid

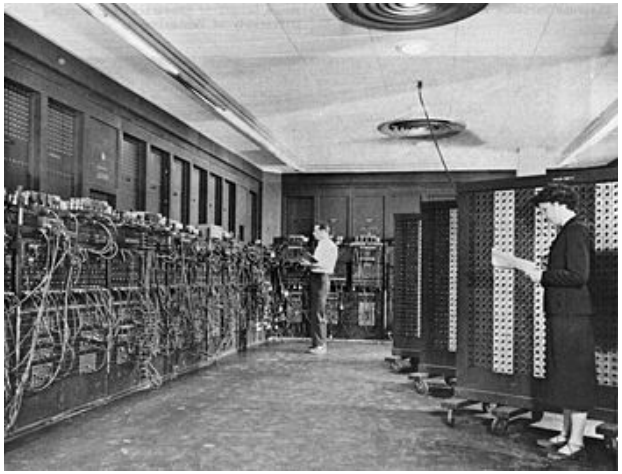
What we'll see in this class

- The evolution of computer science
- What is a distributed system
- Examples of distributed applications
- Strengths and weaknesses of distributed systems
- Distributed vs. parallel systems
- Design challenges
- What is the *middleware*
- Computation paradigms for distributed systems
- Hardware platforms for distributed systems

The evolution of computer science

- The **70s**:
  - Mainframes
    - Shared time systems
    - Centralized resources
    - Simple terminals

  - User interfaces were not very friendly.
  - First computer networks.



The evolution of computer science

- The **80s**:
  - PCs and workstations

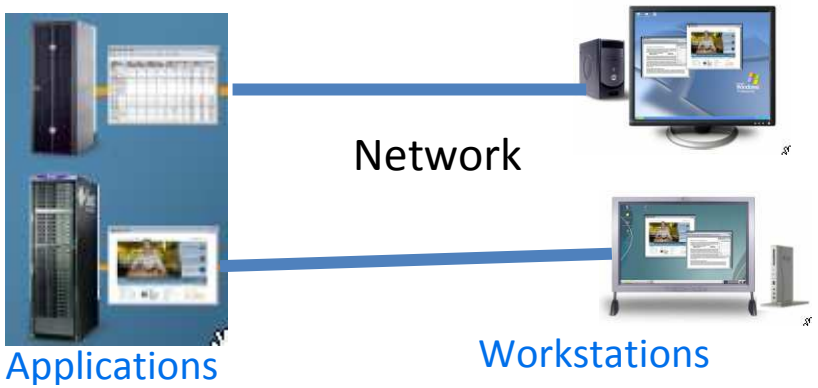
  Applications are mostly complex and executed locally
  - UIs become more friendly
  - Local networks (LAN)
  - First distributed Oss
    - Mach, Sprite, Chorus, ...

The evolution of computer science

- The **90s**:
  - Client/server applications
  - Execution becomes more descentralized
  - The web!
  - Web-based applications take off
    - E-comerce
    - Multimedia
    - Control systems
    - Medical applications
    - Supercomputing using the Internet

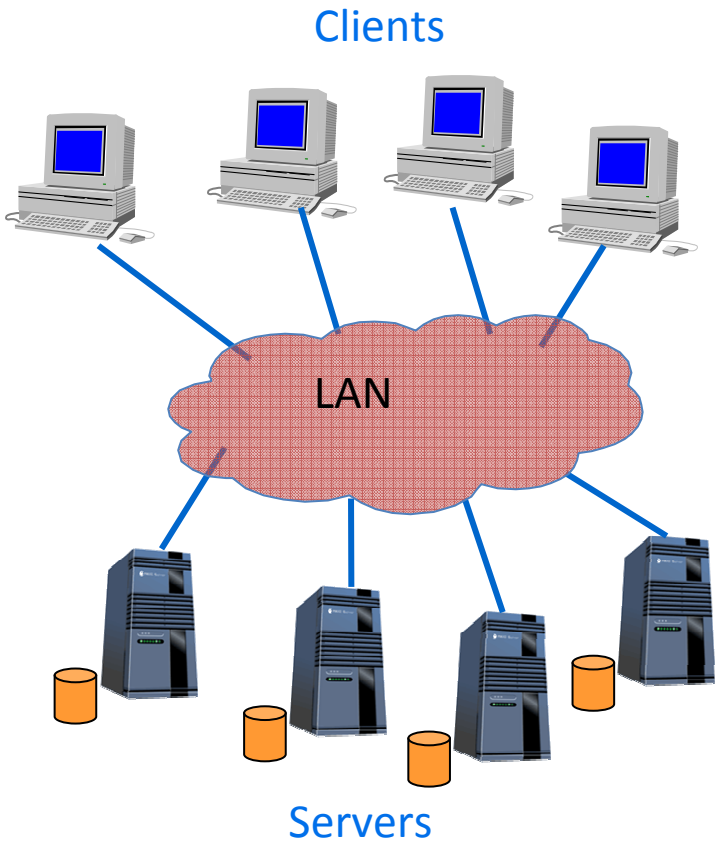
The evolution of computer science

- **2000:**
  - New distributed computing paradigms
    - Grid computing
    - Peer-to-Peer
    - Ubiquitous computing
  - Mobile devices
  - Web-based applications
  - Mostly network-based applications



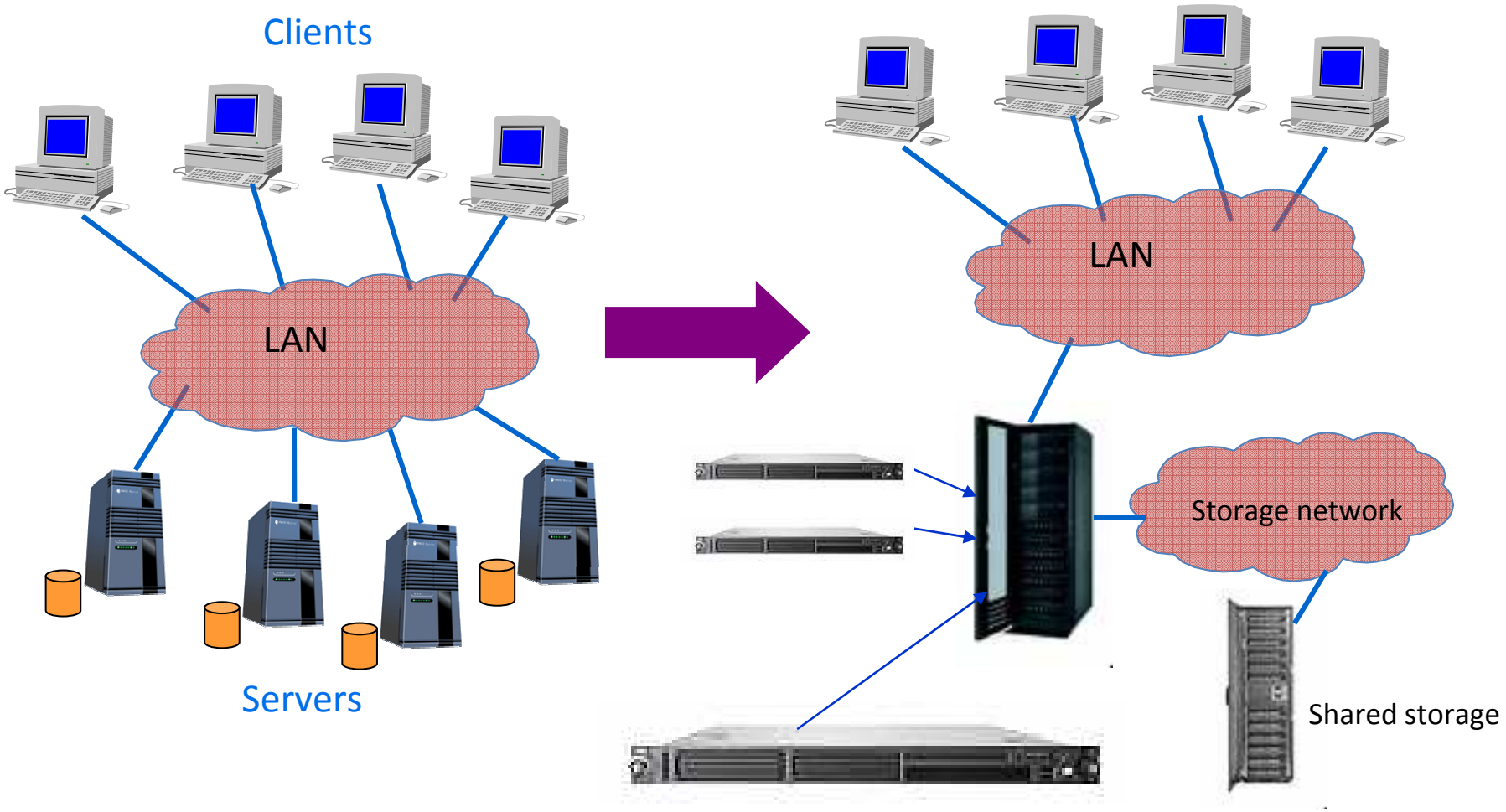
The evolution of computer science

- Traditional infrastructure



The evolution of computer science

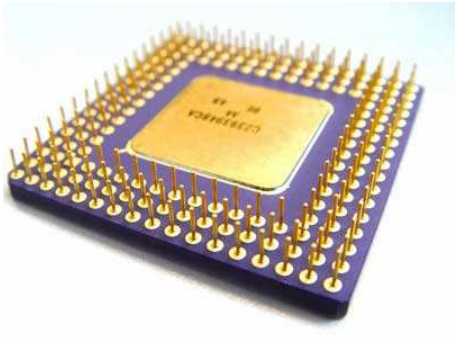
- Traditional infrastructure
- Resource consolidation



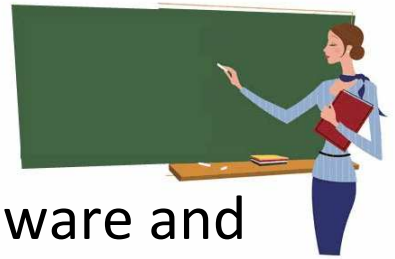
Evolution

- **Bell's Law (1972)**:
  - A new computer technology will come out every 10 years

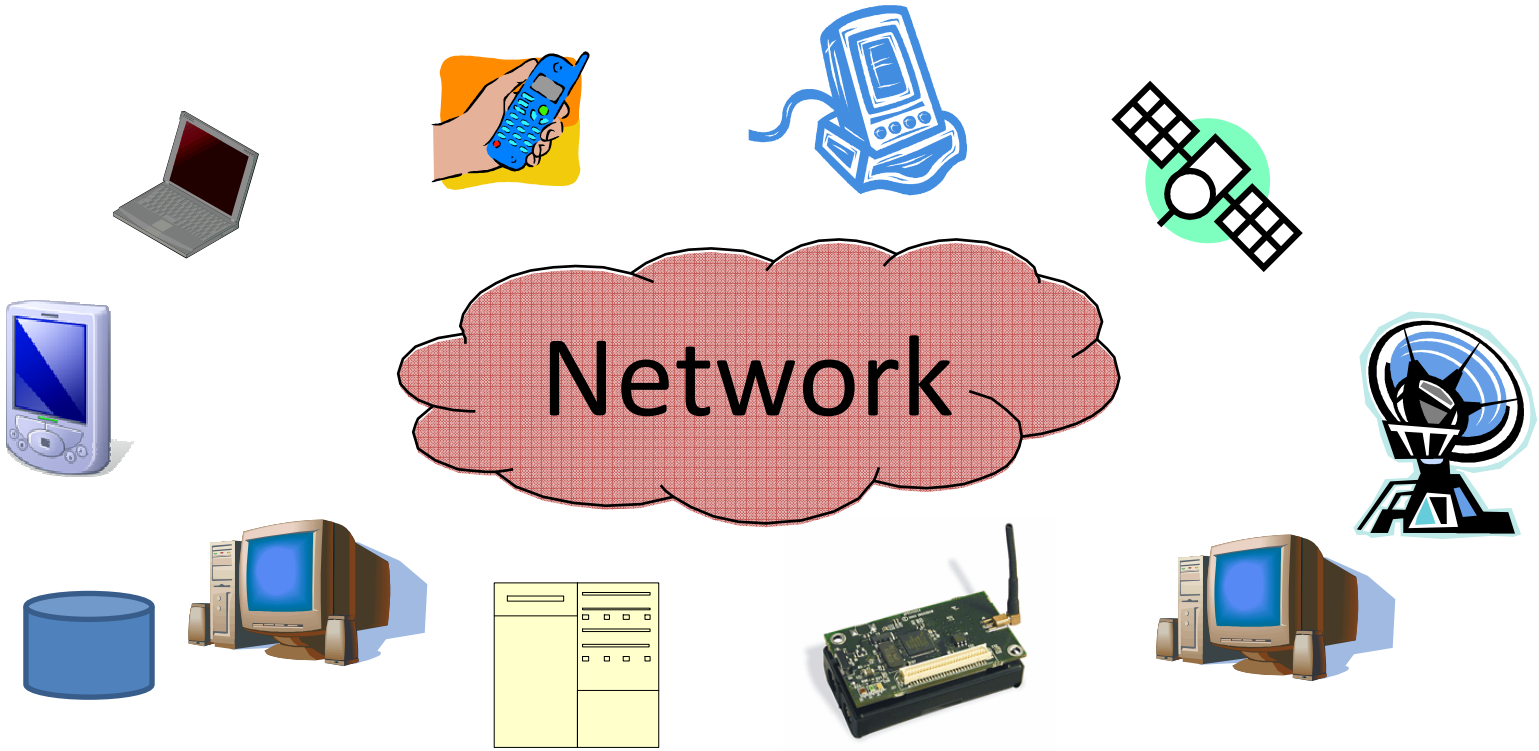
- **Moore's Law (1970)**:
  - The number of transistors per chip will double every 24 months



Distributed Systems



- Consists of physically distributed resources (hardware and software) which are connected via a network and are able to collaborate to accomplish a given task.



Recap

- A process is an executing program
- A computer network is a set of computers connected via an interconnection network
- A distributed system is a set of computers (w/o shared memory nor common clock) which are connected via an interconnection network
  - Distributed application: A set of processes executing on one or more computers and which collaborate and communicate via message passing
- A protocol is a set of rules and instructions which specify how communication is realized in a distributed system via message passing

Another definition of a distributed system

"*A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable.*"

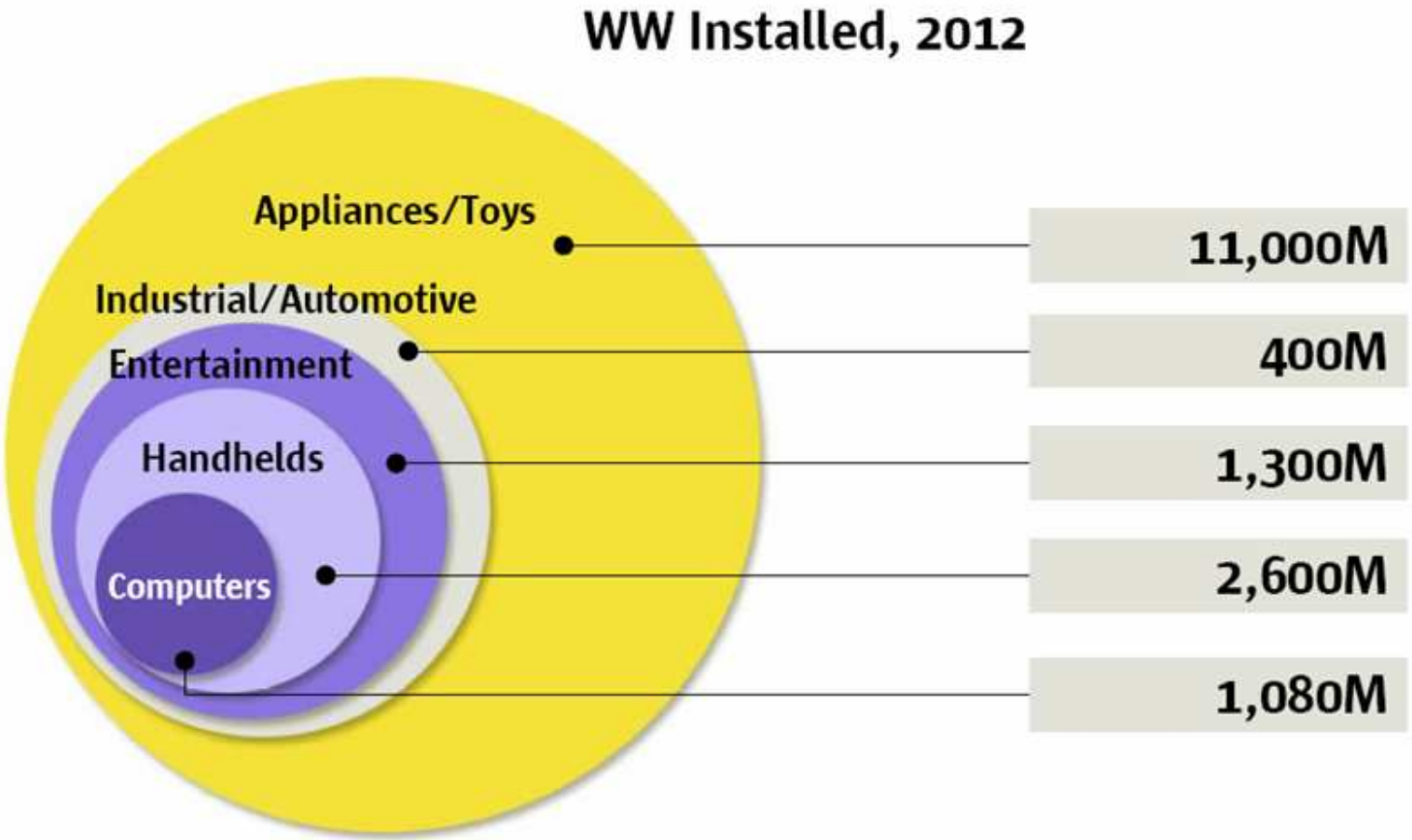
-Leslie Lamport

Examples: The Web

Examples of distributed systems and applications

- Email (IMAP, POP)
- File transfer programs (FTP)
- News services
- World Wide Web (WWW)
- Air traffic control systems
- Bank applications
- E-commerce
- Multimedia applications (videoconferences, video on-demand , etc.)
  - Bandwidth is an order of magnitude bigger than for other apps.
  - They require quality of service (QoS)
- Medical apps. (image transfer)

In the future…



Source: IDC Estimates, 2004

Advantages of distributed systems

- Resource sharing (HW, SW, data)
  - Remote resource access
    - Client-server model
    - Object-based model
- Good cost/performance ratio
- Scalable
- Fault-tolerance
  - Replication
- Concurrency: multiple simultaneous users
- Computing speed/capacity for executing apps. on many machines in parallel

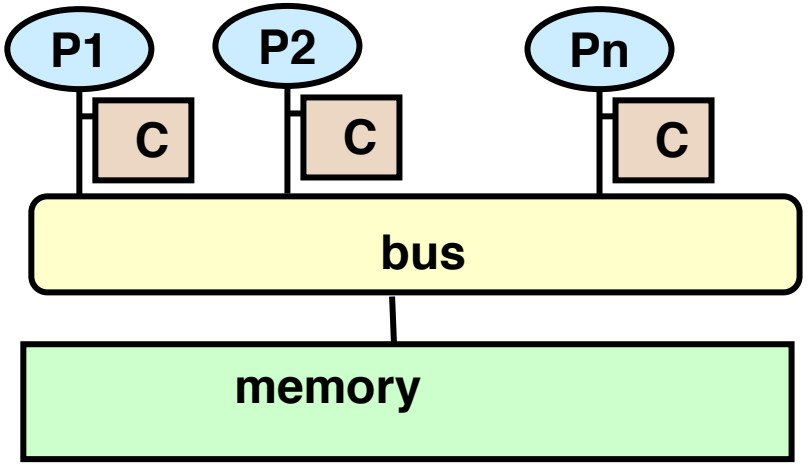
Weaknesses of distributed systems

- Interconnection:
  - Cost
  - Reliability, message loss
  - Saturation
- Security of communication may be a problem
- Software is more complex

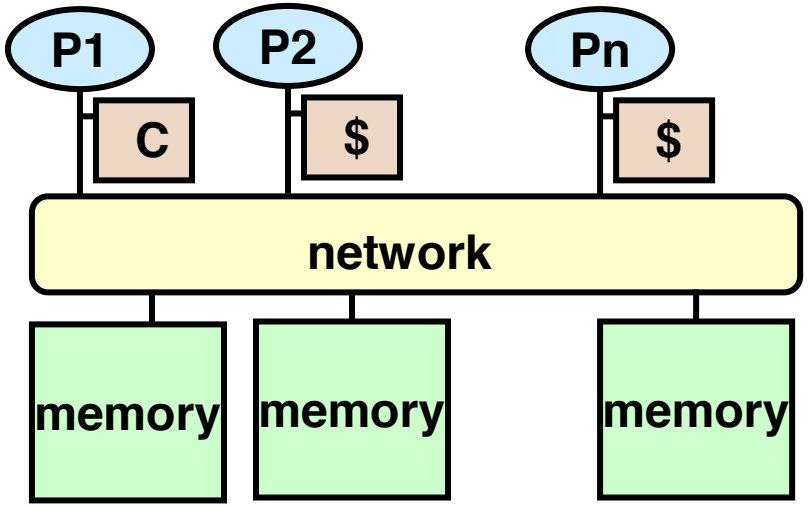
Distributed vs. concurrent systems

- **Distributed systems**
  - Goal: resource sharing, collaboration
  - Computers connected in a network
- **Concurrent systems**
  - Goal:
    - High performance(*speedup*)
    - High productivity
  - Parallel machines (dedicated architectures)
    - Multiprocessors, clusters, supercomputers
  - Networks of workstations working in cluster mode
  - Grid Computing (www.gridcomputing.com)

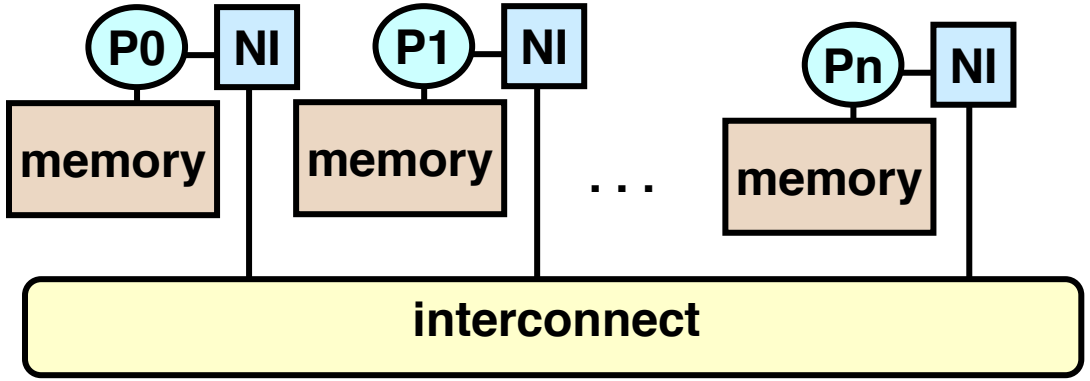
Parallel architectures



Shared memory

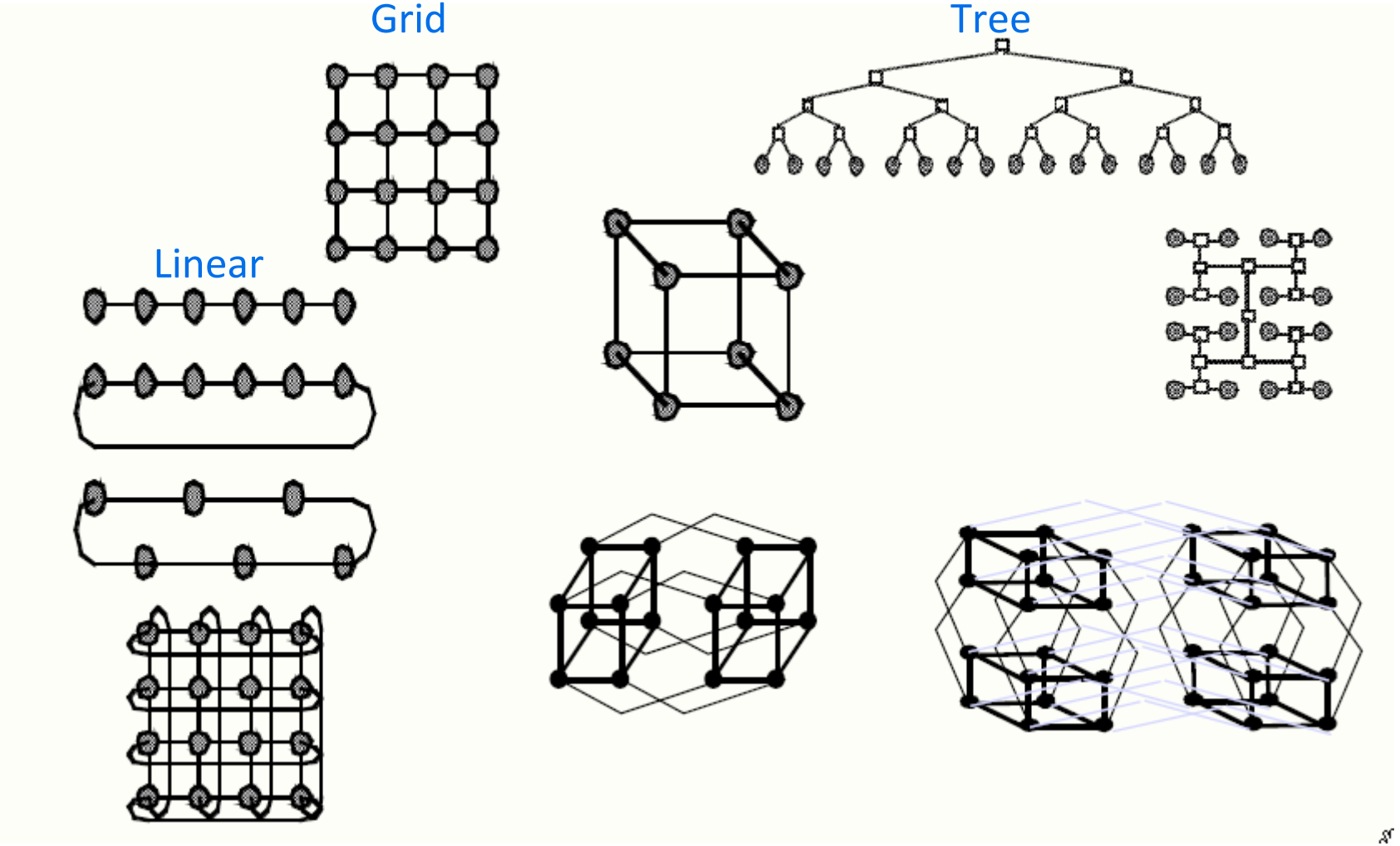


Distributed shared memory



Distributed memory

Network topologies



Distributed systems – design challenges

- Components are heterogenous
- Naming issues
- Communication and syncronization issues
- Performance issues
- Concurrency
- Scalability
- Software structure
- Reliability
- Quality of service (QoS)
- Transparency

Heterogeneity

- Components in a distributed system may vary in terms of:
  - Networks
  - HW
  - OS
  - Programming languages
  - Applications

Solution

- Open systems
  - Public specs and interfaces (e.g. RFCs)
  - Uniform communication mechanisms
  - Can be built on top of heterogenous HW and SW

- Examples:
  - TCP/IP
  - NFS
  - CORBA (www.omg.org)
  - Globus (www.globus.org)
  - Web services

Naming conventions

- Users name objects (e.g. www.uc3m.es)
- Programs name objects via an identifier (e.g. 163.117.131.31)
- Resolving a name means obtaining the identifier starting from the name
- Important: the names must be domain-independent
- Design issues to consider:
  - The namespace (size, structure, hierarchy, ...)
  - The naming system which does the name resolution (e.g. DNS)

Communication and syncronization

- Basic mechanism: message passing
  - Synchronous
  - Asynchronous
- Processes on different machines communicate via send/receive primitives
  - Remote procedure calls
  - Remote object invocation
- Group communication
  - *Multicast*, *broadcast*
  - Useful for teamwork, fault tolerance, replication, ensuring data consistency, etc

What to take into consideration

- For a system with n users to be scalable the number of resources must be proportional to O(n)
- Use distributed algorithms
  - Generally these make use of hierarchical rather than linear structures
- Avoid bottlenecks
  - Descentralized algorithms
- Avoid the SW resources running out
  - E.g.: 32 bits for IP addresses

Software structure

Centralized systems

- The OS:
  - manages the HW resources efficiently
  - offers services to the applications

Software structure

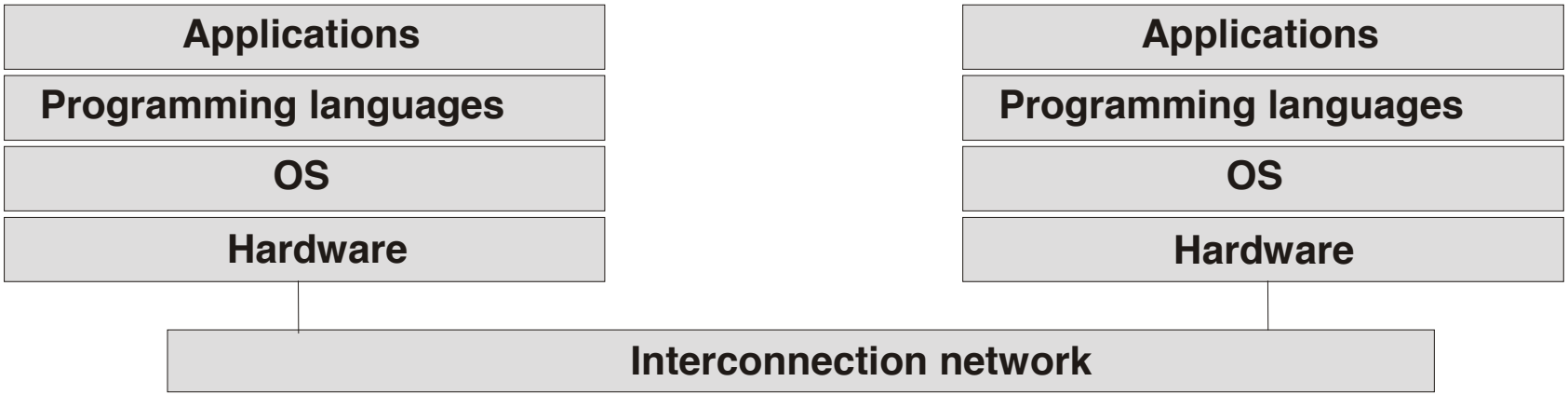
Distributed systems

- Three options:
  - Use machines with separate OSs
  - Use a distributed OS
  - Use a middleware or a distributed environment

- Important to make programming distributed apps easy and transparent

Machines with separate OSs

- The user sees a set of independent machines
  - No transparency
- Access to other machine's resources is explicit
- Hard to use



Reliability

- The probability that a system keeps delivering the desired functionality under given conditions and during a given period of time

- To ensure reliability one must guarantee:
  - Consistency
  - Security
  - Error handling

Consistency

- The problem occurs when various processes access and update data concurrently – aka coherence
  - Update consistency
  - Replication consistency
  - Cache coherence
  - Failure consistency
  - Clock consistency

Failure handling

- In general in distributed systems partial failures may occur
- The goal of a distributed system is availability
  - Measures the percentage of time that a system is available for use
- Improving availability:
    - Tolerating failures
    - Failure detection
    - Failure masking
    - Failure recovery
    - Redundancy

Quality of service (QoS)

- The ability to satisfy the time requirements when transmitting and processing multimedia data streams in real time
- System performance is measured in:
  - Response time
    - Latency
  - Data transfer rate
    - The transfer speed between two computers, usually measured in bps
  - The performance is determined by:
    - Communication network
    - Communication services
    - OS
    - The distributed programming support

Transparency

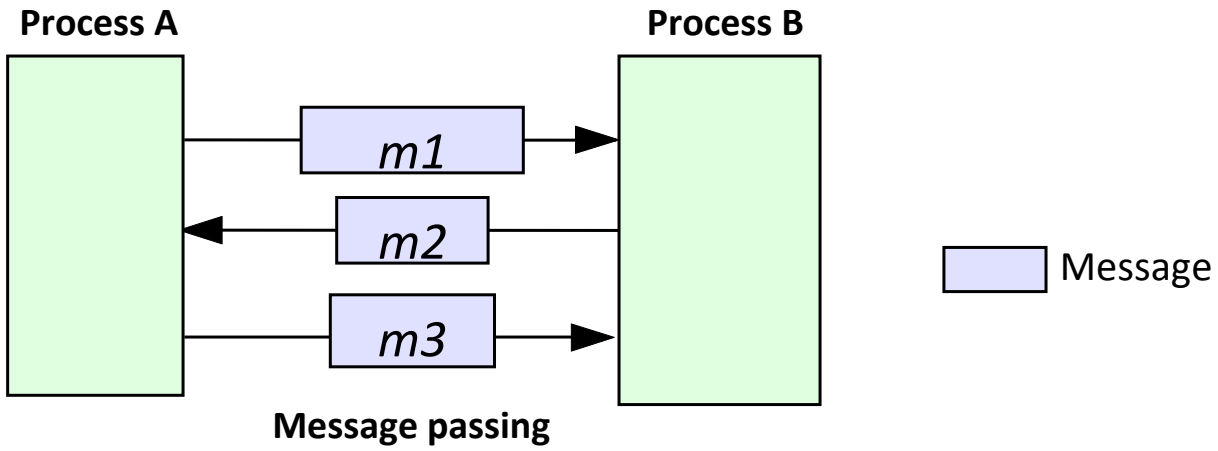
- The concealment from the user of the separation of components in a distributed system
  - **Access T**: local and remote resources are accessed via same operations
  - **Location T**: access without knowledge of their physical/network location
  - **Concurrency T**: processes may operate concurrently on shared resources without interference
  - **Replication T**: multiple instances of resources to increase reliability and performance
  - **Failure T**: concealment of faults
  - **Mobility T**: allows movement of resources and clients without affecting operation of users and programs
  - **Scaling T**: allows system and apps to expand without changes in system structure and algorithms

Distributed computation paradigms

- Message passing
- Client-server
- Remote procedure call
- Peer-to-peer
- Distributed objects
- Mobile agents
- Services
- Collaborative applications *(groupware)*

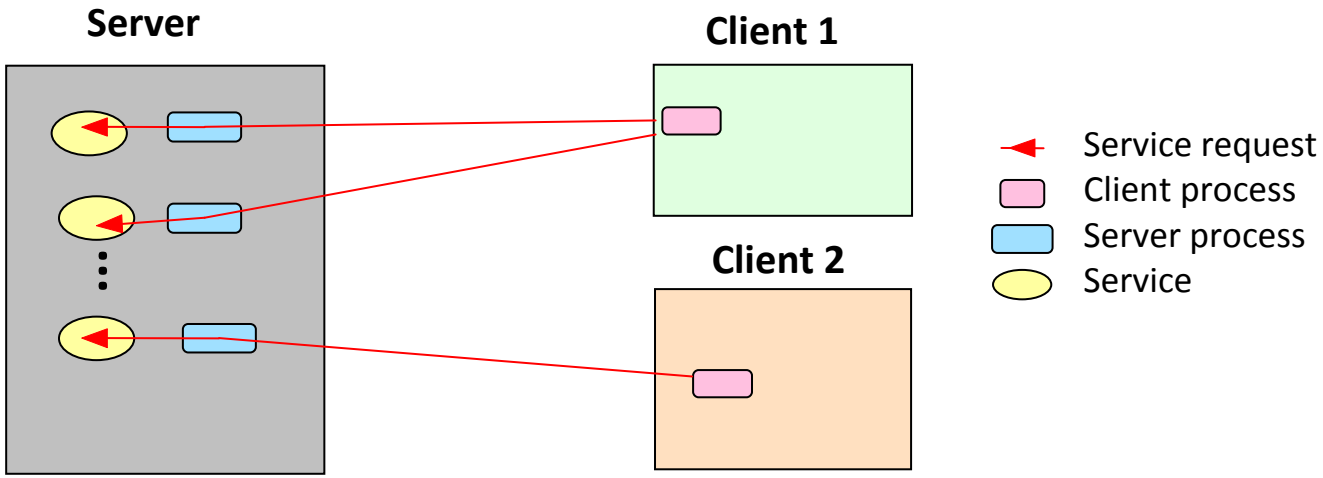
Message passing

- The fundamental paradigm for building distributed apps
- A process sends a request message
- The message is received by the destination process which processes it and replies with another message
- The reply may cause further messages from the sender process



Client-Server

- Asigns different roles - client and server – to the communicating processes
- Server:
  - Offers a service
  - Passive: waits for requests
  - May in turn be clients of other services
- Client:
  - Asks for service
  - Active: sends requests

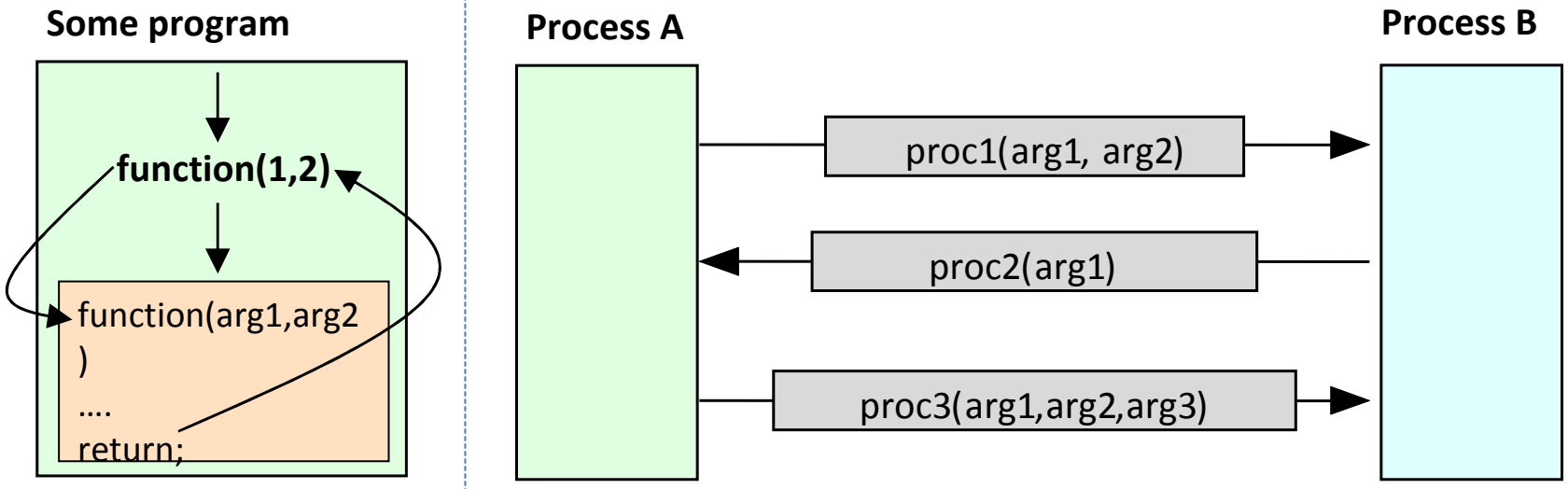


Client-Server

- Efficient abstraction
- Asymmetrical role asignment helps with synchronization
- Implemented via:
  - Sockets
  - Remote procedure calls (RPC)
  - Remote method invocation (RMI, CORBA, …).
- Mostly useful for centralized services
- E.g.: Internet services (HTTP, FTP, DNS, …)

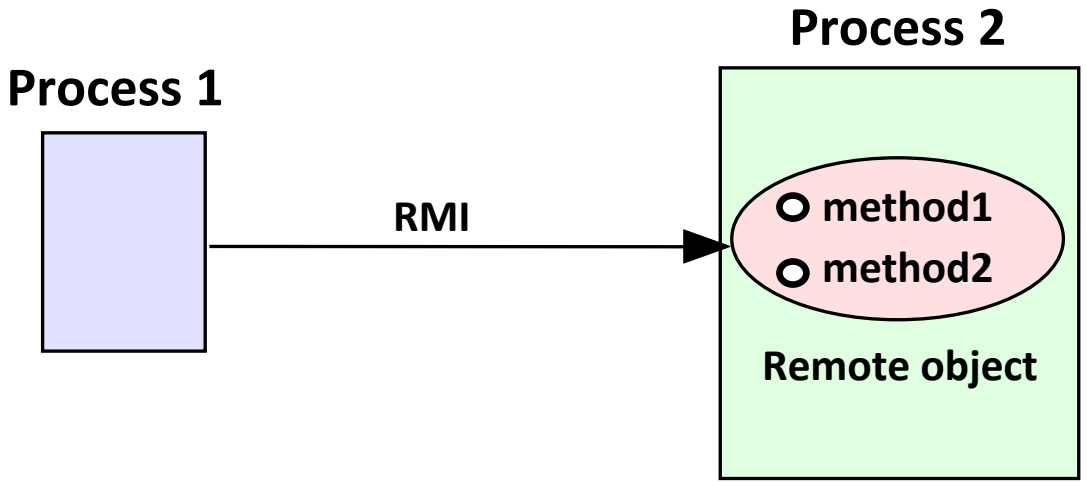
Remote procedure calls

- Idea: make it such that distributed apps are programed the same as non-distributed ones
- Conceptually same as a local procedure call



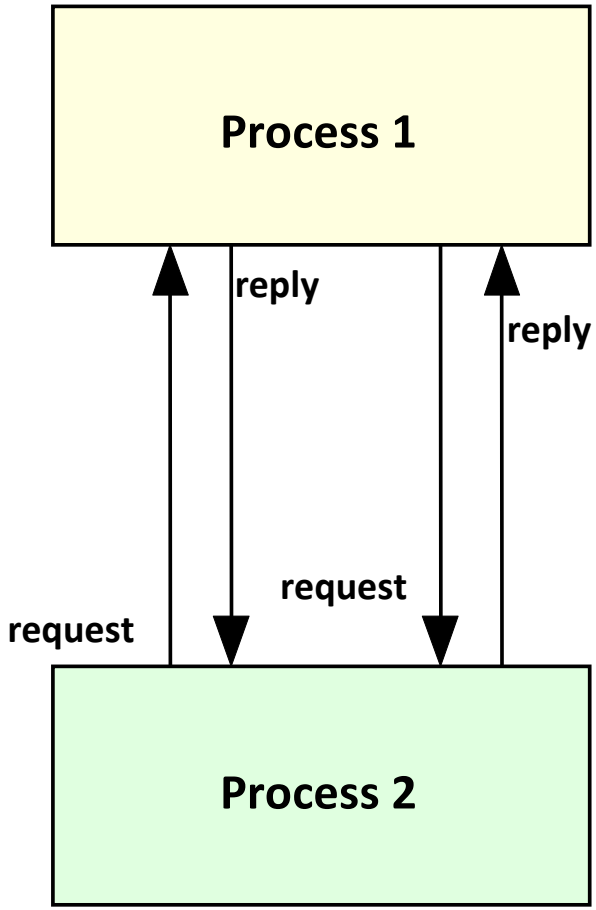
Remote method invocation

- Similar to RPCs – the process invokes a method local to another process
- **E.g.**: CORBA, Java RMI, Microsoft COM, DCOM, Java Beans, .NET Remoting



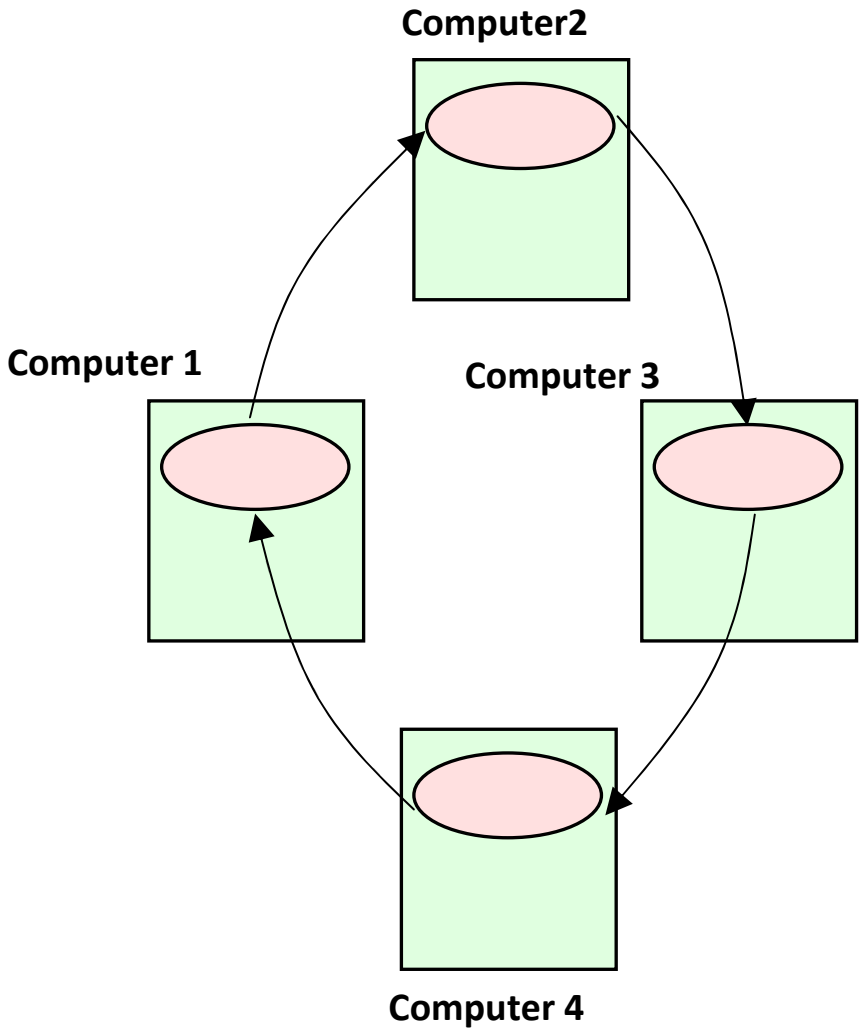
Peer-to-Peer

- The processes involved in communication have the same role:
  - Client **and** server
  - Efficient access to data and other resources that they collectively store and manage
- E.g.:
  - *Napster* → file sharing



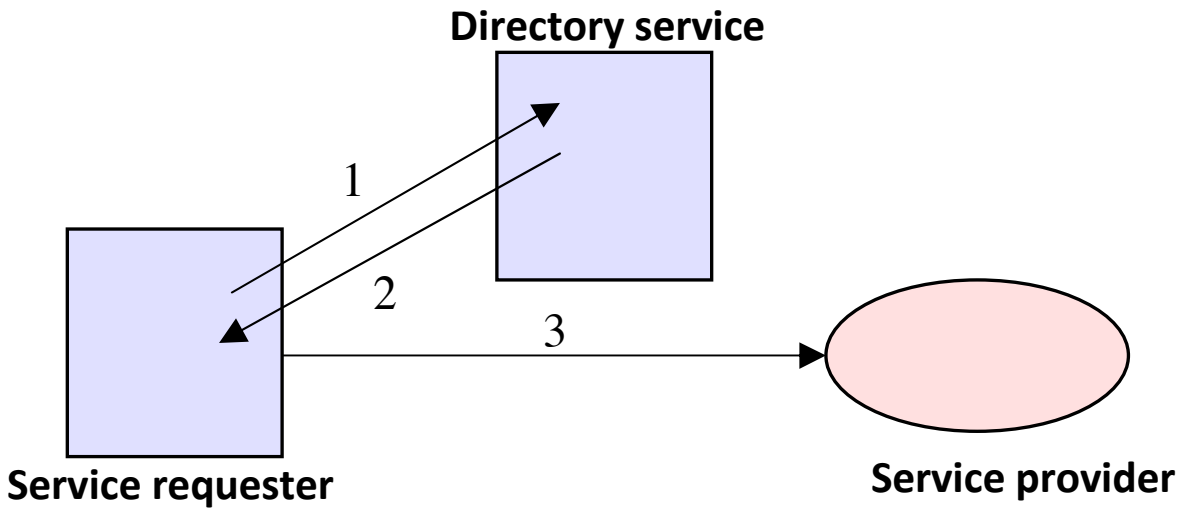
Mobile agents

- A running program that
  - Is launched on an origin computer
  - Travels from one computer to another in a network
  - Carries out a task on someone's behalf
  - Eventually returns with the results
- No message exchange
- Security problem: the executable mobile code may be malicious



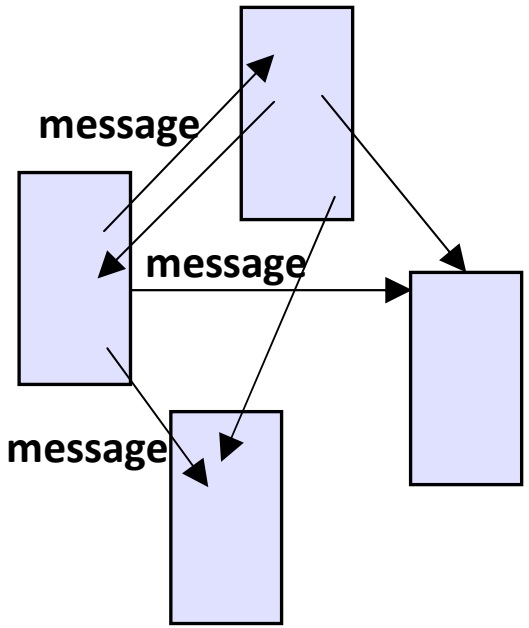
Web services

- A process requests a service via a reference provided by the directory service
- Services must register with the directory service to get their services published
- Location transparency
- E.g.: **SOAP**

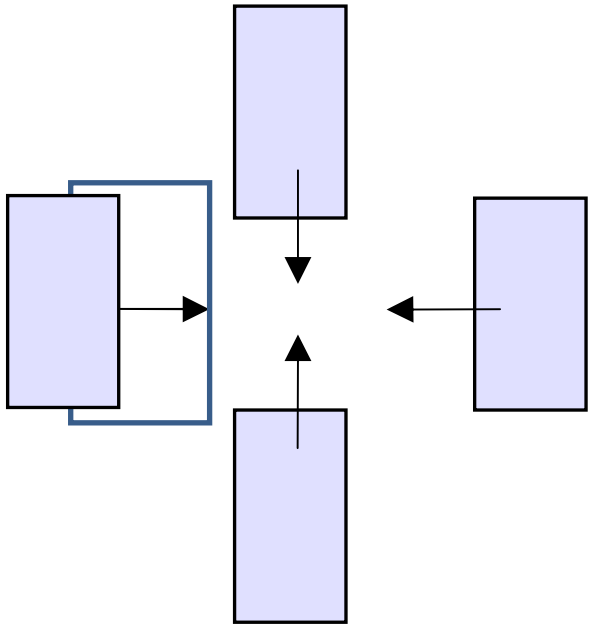


Collaborative applications (*groupware)*

- Several processes participate in a common work session
- Comunication: unicast, multicast, broadcast
- Two types:
  - Message-based: send messages to others in the group
  - Whiteboard-based: use a shared whiteboard to R/W data



Message-based groupware



Whiteboard-based groupware