Introduction to Distributed Systems

Thoai Nam

High Performance Computing Lab (HPC Lab)

Faculty of Computer Science and Engineering

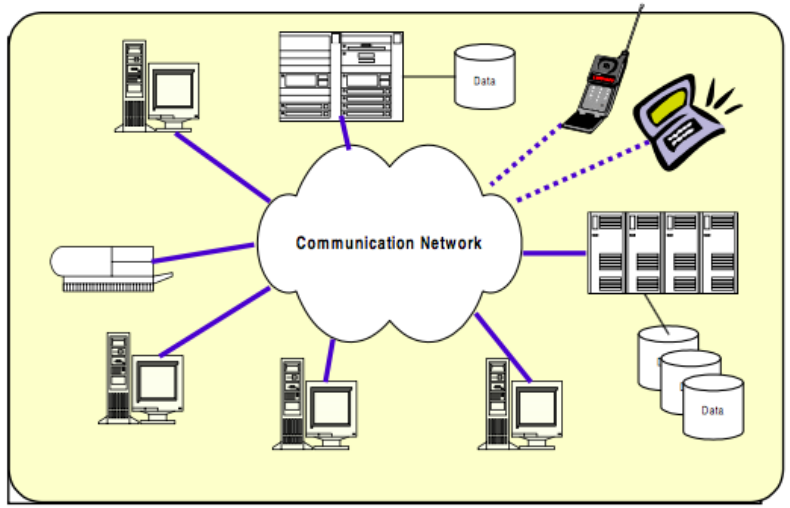
HCMC University of Technology

Applications

Finance and commerce	eCommerce e.g. Amazon and eBay, PayPal, online banking and trading
The information society	Web information and search engines, ebooks, Wikipedia; social networking: Facebook and MySpace.
Creative industries and entertainment	online gaming, music and film in the home, user-generated content, e.g. YouTube, Flickr
Healthcare	health informatics, on online patient records, monitoring patients
Education	e-learning, virtual learning environments; distance learning
Transport and logistics	GPS in route finding systems, map services: Google Maps, Google Earth
Science	The Grid as an enabling technology for collaboration between scientists
Environmental management	sensor technology to monitor earthquakes, floods or tsunamis

A Distributed System is …

Distributed systems are a collection of independent components and machines located on different systems, communicating in order to operate as a single unit.



- Multiple connected CPUs working together
- Components located at networked computers communicate and coordinate their actions only by message passing
- A collection of independent computers that appears to its users as a single coherent system.
- A collection of autonomous computers interconnected by a computer network and equipped with distributed system software to form an integrated computing facility
- Multiple independent machines interconnected through a network to coordinate to achieve a common, consistent service or function
- Known as distributed computing and distributed databases, a distributed system is a collection of independent components located on different machines that share messages with each other in order to achieve common goals
- Today, data is more distributed than ever, and modern applications no longer run in isolation. The vast majority of products and applications rely on distributed systems.

Distributed System examples

- Computer networks
- Telecommunication networks

  Telephone and cellular networks are also examples of distributed networks. Telephone networks have been around for over a century and it started as an early example of a peer-to-peer network. Cellular networks are distributed networks with base stations physically distributed in areas called cells
- Distributed Real-time Systems

  Many industries use real-time systems that are distributed locally and globally. Airlines use flight control systems, Uber use dispatch systems, manufacturing plants use automation control systems, logistics and e-commerce companies use real-time tracking systems
- Parallel & Distributed computing
- Distributed artificial intelligence

  Distributed artificial intelligence is a way to use large scale computing power and parallel processing to learn and process very large data sets using multi-agents
- Distributed database systems

  A distributed database is a database that is located over multiple servers and/or physical locations. The data can either be replicated or duplicated across systems.

  A homogenous distributed database means that each system has the same database management system and data model. They are easier to manage and scale performance by adding new nodes and locations

  Heterogenous distributed databases allow for multiple data models, different database management systems. Gateways are used to translate the data between nodes and usually happen as a result of merging applications and systems.

Disadvantages

- Distribution-aware PLs, OSs and applications
- Security and privacy
- Network connectivity essential
  - Communication may fail (and we might not even know it has failed)
  - Processes may crash (and we might not know)
  - All of this may happen non-deterministically
- Data Integration & Consistency

  being able to synchronize the order of changes to data and states of the application in a distributed system is challenging, especially when there nodes are starting, stopping or failing
- Management Overhead

  more intelligence, monitoring, logging, load balancing functions need to be added for visibility into the operation and failures of the distributed systems

➢ Fault tolerance

we want the system as a whole to continue working, even when some parts are faulty

- "… a system in which the failure of a computer you didn't even know existed can render your own computer unusable." — Leslie Lamport
- This is hard
- Writing a program to run on a single computer is comparatively easy?!

Advantages

- It's inherently distributed
  - E.g. sending a message from your mobile phone to your friend's phone
  - Communication and resource sharing possible
- For better reliability
  - Even if one node fails, the system as a whole keeps functioning
- For better performance
  - Low Latency - having machines that are geographically located closer to users, it will reduce the time it takes to serve users.
- Scalability
  - Unlimited Horizontal Scaling - machines can be added whenever required
  - To solve bigger problems

    e.g. huge amounts of data, can't fit on one machine
- Fault tolerance
- Potential for incremental growth
- Economics – price-performance ratio.

Elements of a Distributed System

- **Resource sharing** - whether it's the hardware, software or data that can be shared
- **Openness** - how open is the software designed to be developed and shared with each other
- **Concurrency** - multiple machines can process the same function at the same time
- **Scalability** - how do the computing and processing capabilities multiply when extended to many machines
- **Fault tolerance** - how easy and quickly can failures in parts of the system be detected and recovered
- **Transparency** - how much access does one node have to locate and communicate with other nodes in the system.

➢ Modern distributed systems have evolved to include autonomous processes that might run on the same physical machine, but interact by exchanging messages with each other.

Challenges

- Heterogeneity
- Openness
- Security
- Scalability
- Failure handling
- Concurrency
- Transparency

Transparency

Transparency	Description
Access	Enables local and remote resources to be accessed using identical operations
Location	Enables resources to be accessed without knowledge of their physical or network location (for example, which building or IP address)
Concurrency	Enables several processes to operate concurrently using shared resources without interference between them
Replication	Enables multiple instances of resources to be used to increase reliability and performance without knowledge of the replicas by users or application programmers
Failure	Enables the concealment of faults, allowing users and application programs to complete their tasks despite the failure of hardware or software components
Mobility	Allows the movement of resources and clients within a system without affecting the operation of users or programs
Performance	Allows the system to be reconfigured to improve performance as loads vary
Scaling	Allows the system and applications to expand in scale without change to the system structure or the application algorithms

Scalability problems

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

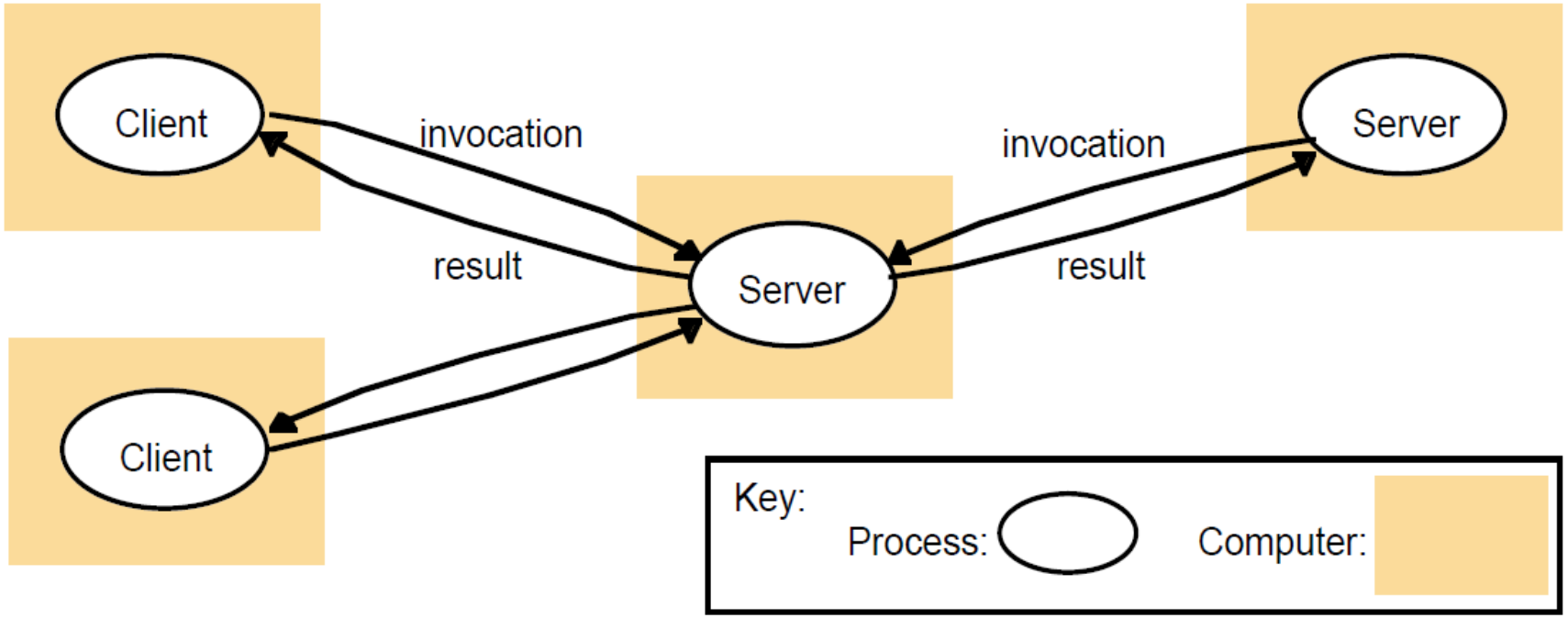
Architecture models

Main types of models

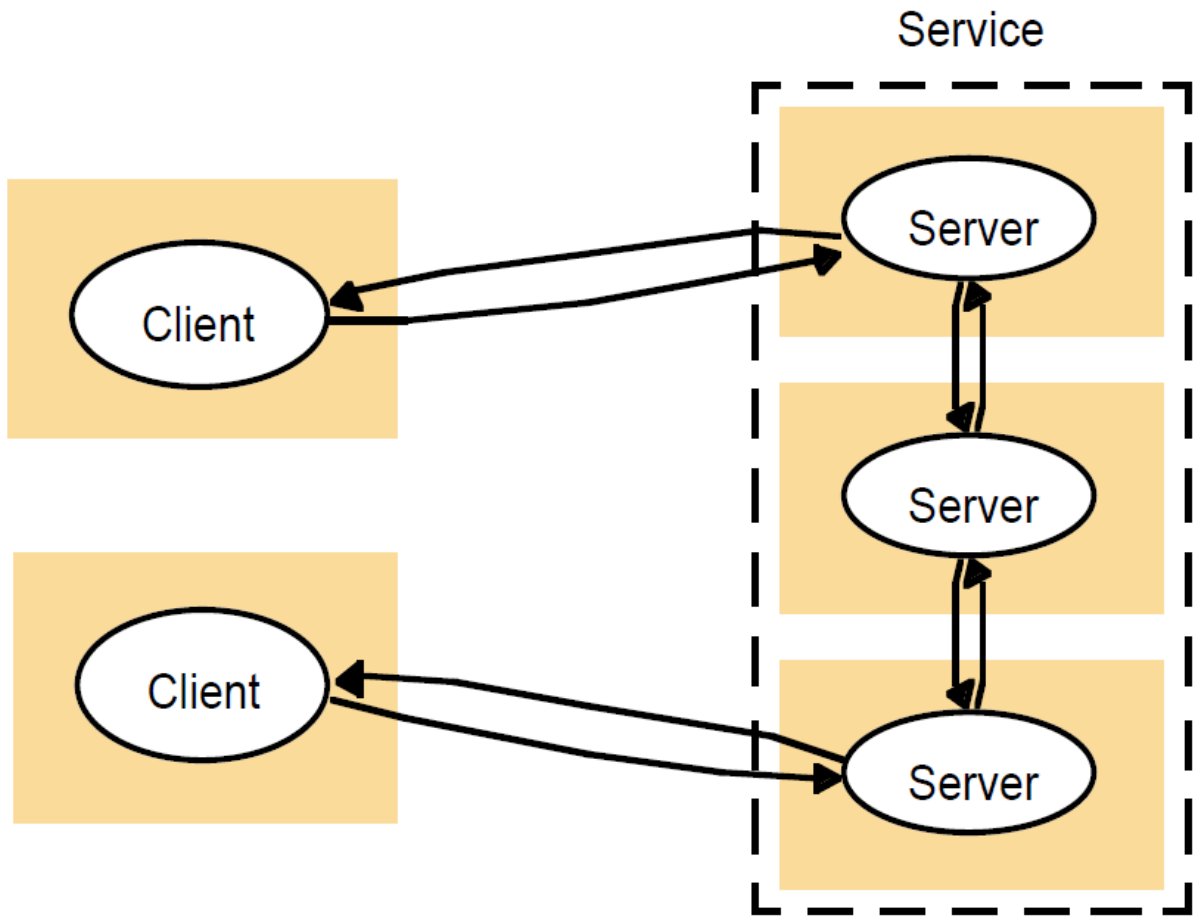
- Client-server
  - first and most commonly used
- Multiple servers
  - to improve performance and reliability
  - e.g. search engines (1000’s of computers)
- Proxy servers
  - to reduce load on network, provide access through firewall
- Peer processes
  - when faster interactive response needed.

Client server

In the early days, distributed systems architecture consisted of a server as a shared resource like a printer, database, or a web server. It had multiple clients (for example, users behind computers) that decide when to use the shared resource, how to use and display it, change data, and send it back to the server. Code repositories like git is a good example where the intelligence is placed on the developers committing the changes to the code.



A service provided by multiple servers



Multi-tier

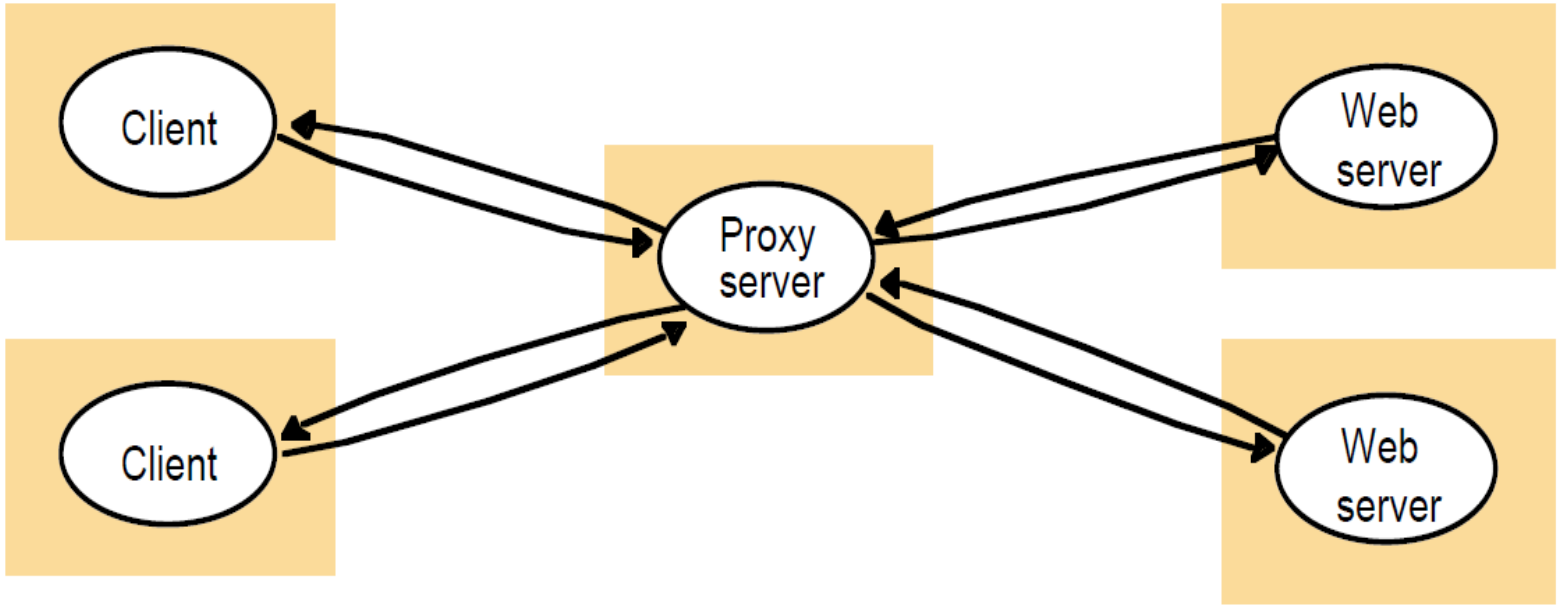
- Three-tier

  In this architecture, the clients no longer need to be intelligent and can rely on a middle tier to do the processing and decision making. Most of the first web applications fall under this category. The middle tier could be called an agent that receives requests from clients, that could be stateless, processes the data and then forwards it on to the servers

- Multi-tier

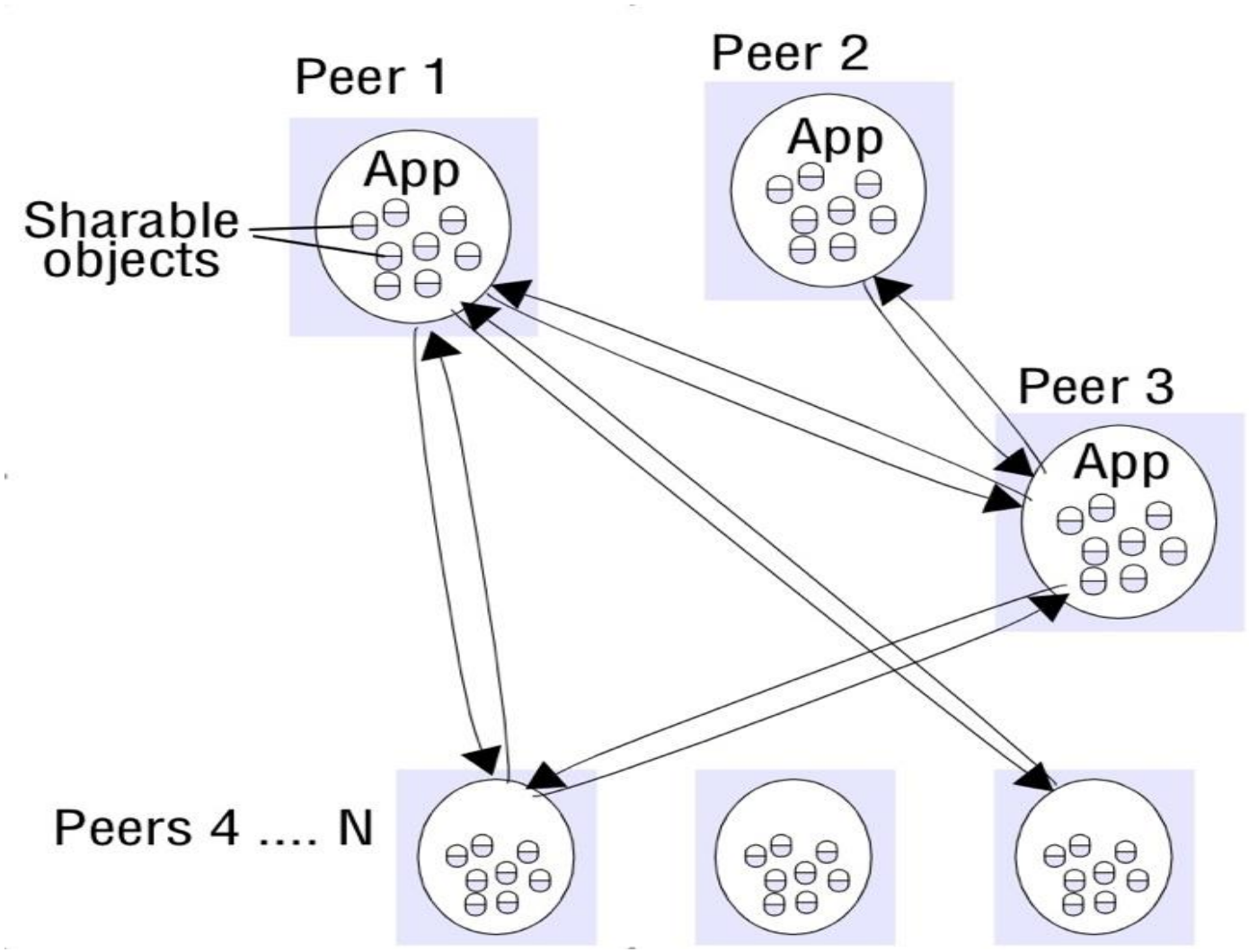
  Enterprise web services first created n-tier or multi-tier systems architectures. This popularized the application servers that contain the business logic and interacts both with the data tiers and presentation tiers.

Proxy servers



Peer-to-peer

There are no centralized or special machine that does the heavy lifting and intelligent work in this architecture. All the decision making and responsibilities are split up amongst the machines involved and each could take on client or server roles. Blockchain is a good example of this.



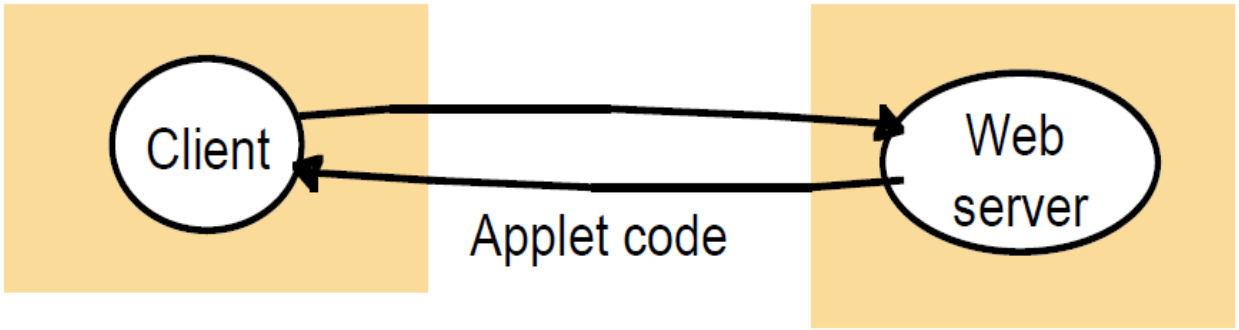
Client server and mobility

- Mobile code
  - downloaded from server, runs on locally
  - e.g. web applets
- Mobile agent (code + data)
  - travels from computer to another
  - collects information, returning to origin

Beware! Security risks

Web applets

a) client request results in the downloading of applet code



b) client interacts with the applet

Design requirements for DSs

Judging how good the architecture is...

- Performance
  - how fast will it respond?
- Quality of Service
  - are video frames and sound synchronized?
- Dependability
  - does it work correctly?

Performance

- Responsiveness
  - fast interactive response delayed by remote requests
  - use of caching, replication
- Throughput
  - dependent on speed of server and data transfer
- Load balancing
  - use of applets, multiple servers

Quality of Service (QoS)

Non-functional properties experienced by users:

- Deadline properties
  - hard deadlines (must be met within T time units)
  - soft deadlines (there is a 90% chance that the video frame will be delivered within T time units)
    - ✧ Multi media traffic, video/sound synchronization
    - ✧ depend on availability of sufficient resources
- Adaptability
  - ability to adapt to changing system configuration

Dependability

- Correctness
  - Dependability
  - correct behavior wrt specification – e.g. use of verification
- Fault-tolerance
  - ability to tolerate/recover from faults – e.g. use of redundancy
- Security
  - ability to withstand malicious attack – e.g. use of encryption, etc.

Distributed Operating Systems

Multiprocessor Operating Systems

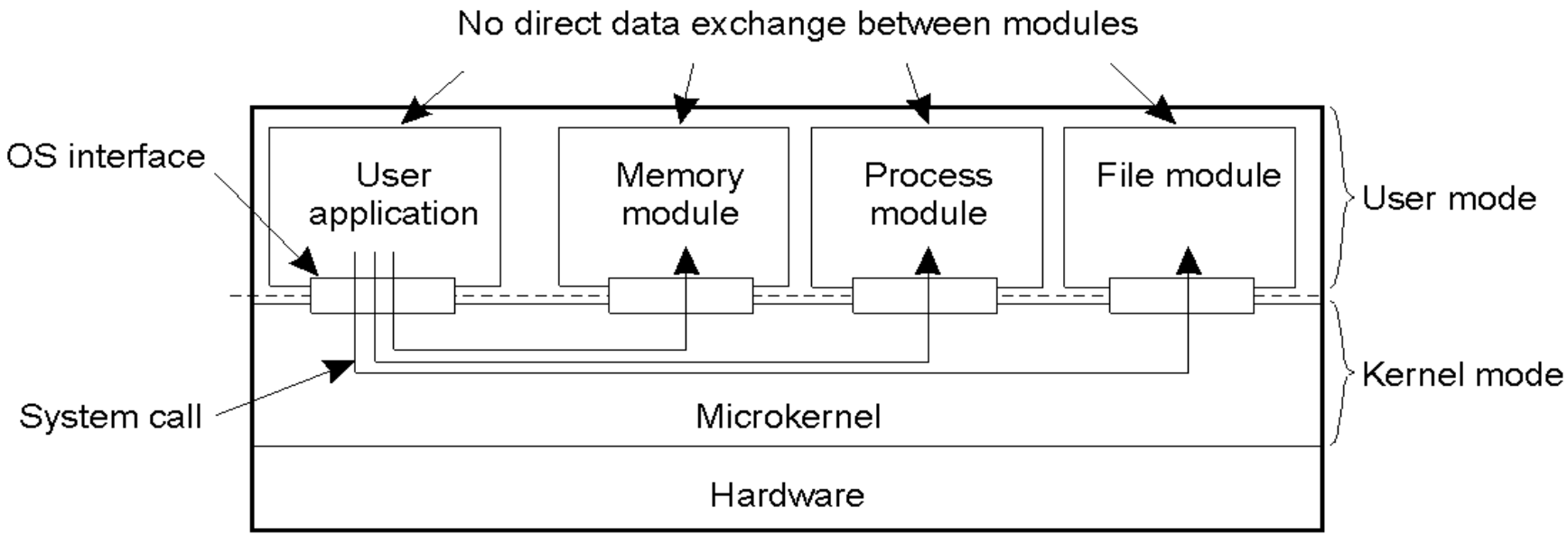
- Like a uniprocessor operating system
- Manages multiple CPUs transparently to the user
- Each processor has its own hardware cache
  - Maintain consistency of cached data

Uniprocessor Operating Systems

- An OS acts as a resource manager or an arbitrator
  - Manages CPU, I/O devices, memory
- OS provides a virtual interface that is easier to use than hardware
- Structure of uniprocessor operating systems
  - Monolithic (e.g., MS-DOS, early UNIX)
    - ✧ One large kernel that handles everything
  - Layered design
    - ✧ Functionality is decomposed into N layers
    - ✧ Each layer uses services of layer N-1 and implements new service(s) for layer N+1

Uniprocessor Operating Systems

- Microkernel architecture
  - Small kernel
  - user-level servers implement additional functionality



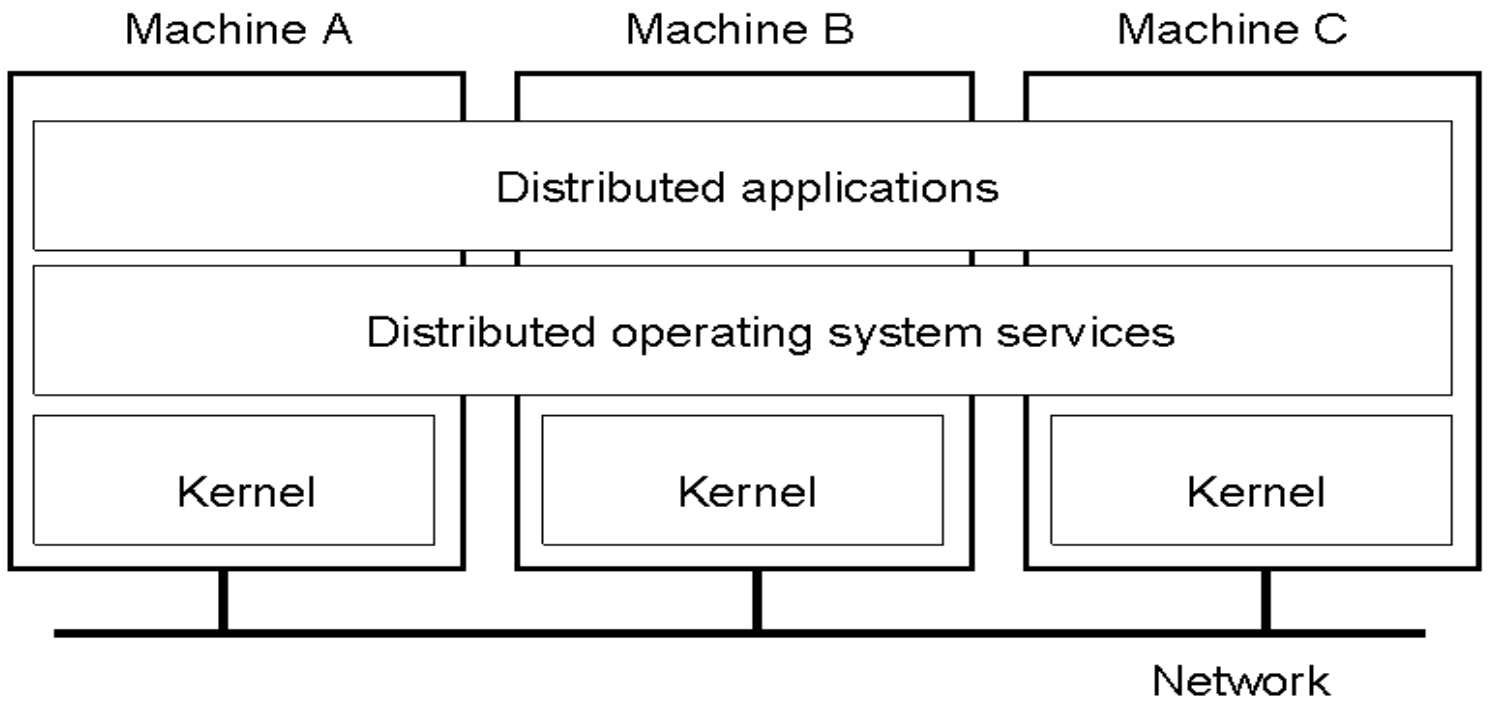
Distributed Operating System

- Manages resources in a distributed system
  - Seamlessly and transparently to the user
- Looks to the user like a centralized OS
  - But operates on multiple independent CPUs
- Provides transparency
  - Location, migration, concurrency, replication,…
- Presents users with a virtual uniprocessor.

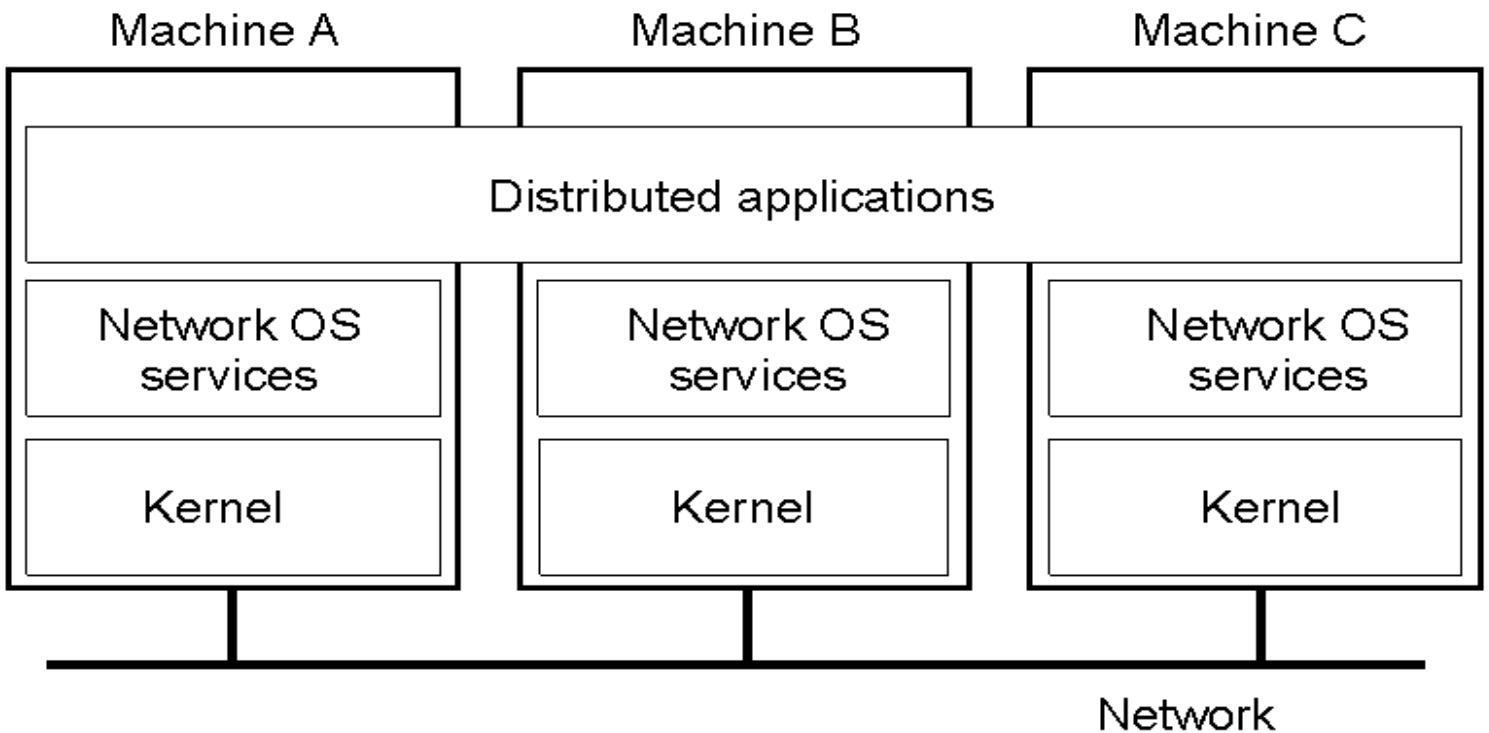
Types of Distributed OSs

System	Description	Main Goal
DOS	Tightly-coupled operating system for multi-processors and homogeneous multi-computers	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

Multicomputer Operating Systems

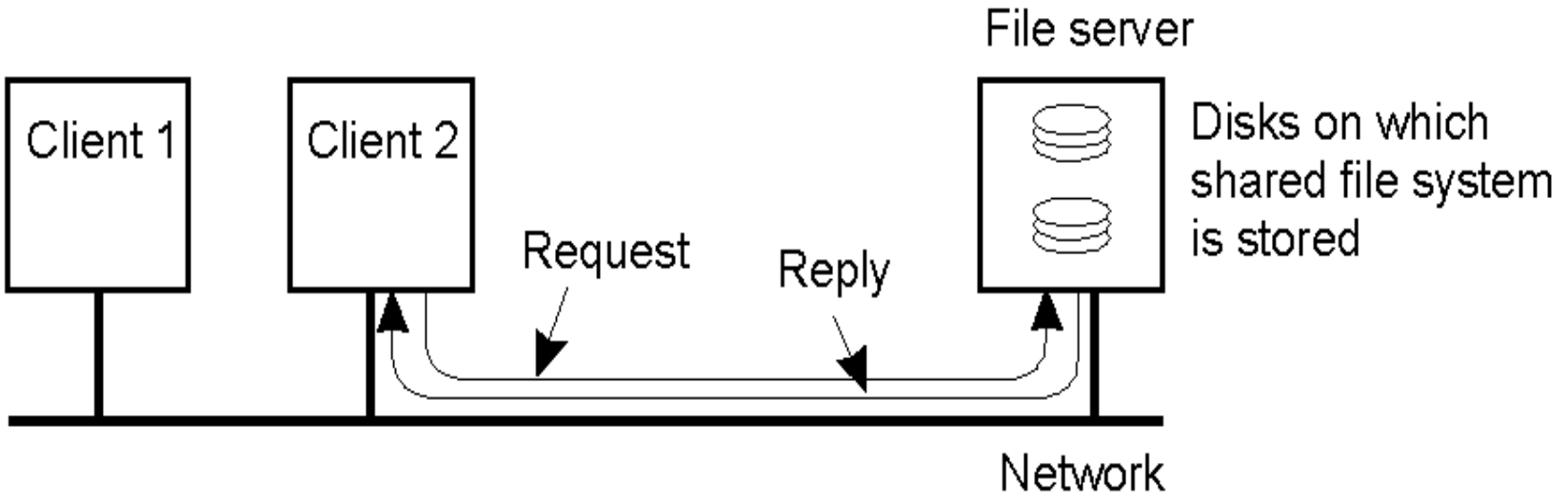


Network Operating System (1)



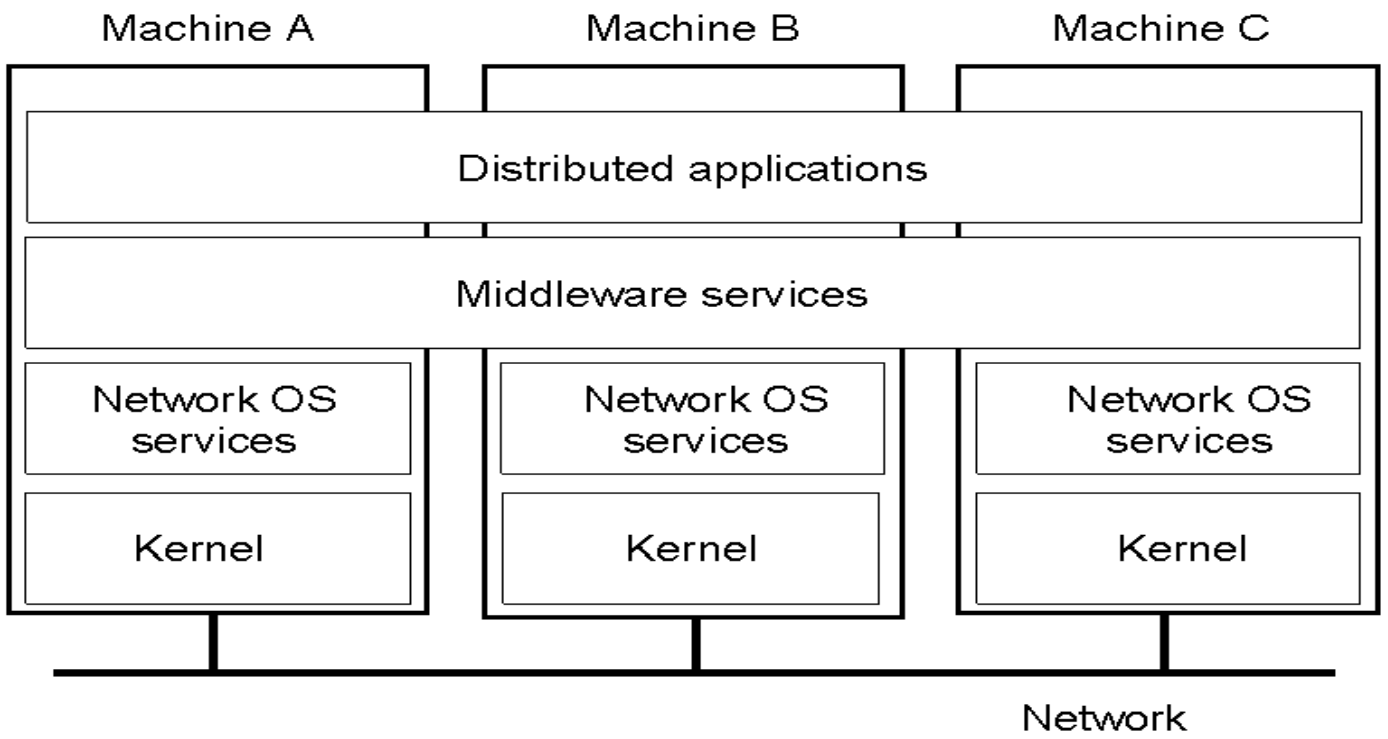
Network Operating System (2)

- Employs
  - Minim
  - Additi



Middleware-based systems

- General structure of a distributed system as middleware



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