CMPU4021 Distributed Systems

Revisions - 1

What is a Distributed System (DS)?

- Multiple computers working together on one task
- Computers are connected by a network, and exchange information
- Key motivation for DS is
  - Sharing resources, e.g.
    - data storage,
    - printers, files, databases,
    - programs/applications,
    - multimedia services: camera video/image, frames, audio/phone data

DS Characteristics

- Concurrency
  - Collaborative and cooperative problem-solving (interdependencies)
- Independent failures of components
  - Autonomous but interdependent - requires coordination, graceful degradation
- Lack of global clocks
  - Each node is autonomous and will thus have its own notion of time: there is no global clock.
  - Leads to fundamental synchronization and coordination problems.
- Heterogeneity
  - Hardware/software (programs, data) variations in component systems

DS Characteristics I

- Openness
  - Modularity, architecture and standards allow extensibility
- Security
  - Protection (internal and external) against malicious use or attack – integrity
- Scalability
  - Accommodation of increased users and resource demands over time
- Transparency
  - Hide separation of components (hidden by middleware)

DS Characteristics

- Concurrency
  - Collaborative and cooperative problem-solving (interdependencies)
- Independent failures of components
  - Autonomous but interdependent - requires coordination, graceful degradation
- Lack of global clocks
  - Each node is autonomous and will thus have its own notion of time: there is no global clock.
  - Leads to fundamental synchronization and coordination problems.
- Heterogeneity
  - Hardware/software (programs, data) variations in component systems

DS Characteristics I

- Openness
  - Modularity, architecture and standards allow extensibility
- Security
  - Protection (internal and external) against malicious use or attack – integrity
- Scalability
  - Accommodation of increased users and resource demands over time
- Transparency
  - Hide separation of components (hidden by middleware)

Examples of DS - The Internet

- Interconnected computers/intranets, which communicate over backbone/medium via messages using the IP (Internet protocol)
  - A *backbone*
    - a network link with a high transmission capacity, employing satellite connections, fibre optic cables and other high-bandwidth circuits.

- Medium:

  - wired: copper circuits, fibre optic

  - wireless: satellite

DS Challenges

- Fundamental issue
  - Different nodes have different knowledge.
  - One node does not know the status of other nodes in the network

- If each node knew exactly the status at all other nodes in the network, computing would be easy:
  - Impossible, theoretically and practically

Theoretical issue: Knowledge cannot be perfectly up to date

- Information transmission speed
  - hardware and software limitations of the nodes & network
- New things can happen while information is traveling from node A to node B
- B can never be perfectly up to date about the status of A

Practical Challenges

- Communication is costly
  - It is not practical to transmit everything from A to B all the time
- There are many nodes
  - Transmitting updates to all nodes and receiving updates from all nodes are even more impractical

DS Challenges – Heterogeneity I

- Heterogeneity (variety and difference) applies to:
  - Networks
    - differences are masked by the fact that all of the computers use the Internet protocols to communicate.
  - Hardware
    - data types, such as integers, may be represented in different ways on different sorts of hardware (byte ordering: big-endian, little-endian)
  - Operating systems
    - do not provide the same application API to the Internet protocols.
  - Programming languages
    - used different representations for characters and data structures, such as arrays and records.
  - Developers
    - representation of primitive data items and data structures needs to be agreed upon (standards)

DS Challenges - Openness

- The characteristic that determines whether the system can be extended and re-implemented in various ways.
- Can it be extended with new content/services without disruption to the underlying system?
- Key interfaces/standards are published
- Standards
  - Request For Comments (RFC)s
  - IETF
  - W3C
- Everything conforms to a standard!

DS Challenges - Security

- Three main issues
  - Confidentially
    - protection against disclosure to unauthorized individuals
  - Integrity
    - protection against alteration or corruption
  - Availability
    - protection against interference with the means to access the resources

DS Challenges – Scalability

- Avoid performance bottle neck
  - Algorithms should be decentralised to avoid having performance bottlenecks.

    Predecessor of the Domain Name System (DNS) kept the name table in a single master file that could be downloaded to any computers that needed it - fine with a few hundred computers.

    The DNS removed the bottleneck by partitioning the name table between servers located throughout the Internet and administered locally.

- Scalability techniques:

  - Replicated data, caching, multiple servers etc.

DS Challenges - Failure Handling

- Failures in distributed systems are mostly partial failures which can make failure handling more difficult

- Detecting failures
  - Checksums

- Masking failures
  - retransmission
  - duplicate files

DS Challenges - Concurrency

- Resources can be shared by clients in a distributed system
  - several clients may access a shared resource at the same time
- Not acceptable that each request be processed in turn
  - must be able to process requests concurrently
- Data consistency
  - For each ‘object’ that represents a shared resource, its operations must be synchronised in such a way that its data remains consistent

Transparencies

- Hide distribution from users and from software

- The concealment from the user and application programmer of the separation of components in a distributed system:
  - Access transparency
  - Location transparency
  - Concurrency transparency
  - Replication transparency
  - Failure transparency
  - Mobility transparency
  - Performance transparency
  - Scaling transparency

Interprocess communication

- To communicate
  - one process sends a message (a sequence of bytes) to a destination; and
  - another process at the destination receives the message.
- This activity involves the communication of data from the sending process to the receiving process
  - may involve the synchronization of the two processes.

Multicast communication

- The basic idea is to disseminate information from **one** sender to **multiple** receivers.
- An operation that sends a single message from one process to each of the members of a group of processes
  - usually in such a way that the membership of the group is transparent to the sender.
- Can span multiple physical networks
- There is a range of possibilities in the desired behaviour of a multicast.
- The simplest multicast protocol provides no guarantees about message delivery or ordering.

Multicast

- One-to-many or many-to-many distribution
- In computer networking
  - Group communication where information is addressed to a group of destination computers simultaneously
- Group communication, either
  - *application layer multicast*
  - *network assisted multicast*
    - makes it possible for the source to efficiently send to the group in a single transmission
- Network assisted multicast
  - May be implemented at the Internet layer using IP multicast

IP Multicast

- Built on top the Internet Protocol
- An implementation of multicast communication
- IP packets are addressed to computers
  - ports belong to the TCP and UDP levels
- IP multicast allows the sender to transmit a single IP packet to a set of computers that form a multicast group.

IP Multicast

- At the application programming level
  - IP multicast is available only via UDP.
  - An application program performs multicasts by sending UDP datagrams with multicast addresses and ordinary port numbers.
  - It can join a multicast group by making its socket join the group, enabling it to receive messages to the group

- At the IP level
  - A computer belongs to a multicast group when one or more of its processes has sockets that belong to that group.
  - When a multicast message arrives at a computer, copies are forwarded to all of the local sockets that have joined the specified multicast address and are bound to the specified port number.

Multicast Group

- A set of Internet hosts that share a multicast address.
- Dynamic membership
  - Machine can join or leave at any time
- Any data sent to the multicast address is relayed to all the members of the group. Membership in a multicast group is open; hosts can enter or leave the group at any time.
- Groups can be either permanent or transient:
  - Permanent groups have assigned addresses that remain constant, whether or not there are any members in the group; typically given names
  - Most multicast groups are transient – exist only as long as they have members (in the 255 -- 238 range).

IP multicasting

- No restriction on number of hosts in a group
- Machine does not need to be a member to send messages
- Efficient: Packets are replicated only when necessary

IP Multicast Use

- Initially
  - Internet radio, NASA shuttle missions, collaborative gaming
- IPTV
  - Cable TV networks are moving to IP delivery
  - Multicast allows one stream of data to be sent to multiple subscribers using a single address

Concurrency

- Concurrency in distributed system**s**
  - Concurrent requests to its resources
  - Each resource must be designed to be safe in a concurrent environment

- *Computer program*
    - *Collection of instructions*
      - *process is the actual execution of those instructions*

- Concurrent programming
  - Systems can do more than one thing at a time
  - E.g. streaming audio application must simultaneously read the digital audio off the network, decompress it, manage playback, and update its display.
  - The word processor should always be ready to respond to keyboard and mouse events, no matter how busy it is reformatting text or updating the display.
  - Software that can do such things is known as *concurrent* software.

- In concurrent programming, there are two basic units of execution:
  - *Processes* and *threads*

Threads

- Sometimes called *lightweight processes*
- A **thread** is a single sequential flow of execution that runs through a program.
- Threads exist within a process
  - every process has at least one
- Unlike a process, a thread does not have a separate allocation of memory, but shares memory with other threads created by the same application.
- Thread context switching can be done entirely independent of the operating system.

Threads vs Processes

Processes

- Composed of 1 or more threads
- Have their own address space
- Communicate via message passing

Threads

- Share common memory and resources
- Can communicate via shared memory

Thread usage in distributed systems

- Provide a way of allowing blocking system calls without blocking the entire process in which the thread is running
  - Allows maintaining multiple logical connections at the same time.

Threads usage: multiple servers

Example

- Web servers replicated across multiple machines, where each server provides exactly the same set of Web documents.
- When a request for a Web page comes in, the request is forwarded to one of the servers
- When using a multithreaded client, connections may be set up to different server replicas
  - allowing data to be transferred in parallel, effectively establishing that the entire Web document is fully displayed in a much shorter time than with a nonreplicated server.
- This approach is possible only if the client can handle parallel streams of incoming data
  - Threads are ideal for this purpose.

Marshalling and Unmarshalling

- The information stored in running programs is represented as data structures
  - e.g., by sets of interconnected objects
- The information in message consists of sequences of bytes.
- Irrespective of the form of communication used, the data structures must be
  - Flattened, converted to a sequence of bytes before transmission
  - Rebuilt on arrival
- *External data representation*
  - An agreed standard for the representation of data structures and primitive values

Marshalling and Unmarshalling

- *Marshalling* is the process of taking a collection of data items and assembling them into a form suitable for transmission in a message.
  - Marshalling consists of the translation of structured data items and primitive values into an external data representation.

- *Unmarshalling* is the process of disassembling them on arrival to produce an equivalent collection of data items at the destination.
  - Unmarshalling consists of the generation of primitive values from their external data representation and the rebuilding of the data structures.

External data representation: Approaches

- Java's object serialization
  - which is concerned with the flattening and data representation of any single object or tree of objects that may need to be transmitted in a message or stored on a disk. It is for use only by Java.

- XML (Extensible Markup Language)
  - Defines a textual format for representing structured data.
  - It was originally intended for documents containing textual self-describing structured data
    - For example documents accessible on the Web
  - Now also used to represent the data sent in messages exchanged by clients and servers in web services.

External data representation: other techniques

- Protocol buffers
  - Google uses an approach called *protocol buffers (aka* protobuf) to capture representations of both stored and transmitted data
  - offers a common serialization format for Google, including the serialization of requests and replies in remote invocation

- JSON (JavaScript Object Notation)
  - an approach to external data representation [www.json.org].

- Protocol buffers and JSON
  - more lightweight approaches to data representation
    - when compared, for example, to XML.

Google Protocol Buffers vs XML

- Simpler format compared to XML, faster in operation
- But, Google infrastructure is a relatively closed system
  - It does not address interoperability across open systems
    - XML does
- XML is significantly richer
  - it generates self-describing messages that contain the data and associated metadata describing the structure of the messages
  - Protocol buffers do not provide this facility directly

Introduction

- Architectural model of a distributed system

  - Concerned with hierarchy/placement of component parts and relationships, e.g., Client-Server (C-S) and Peer-Process models.

  – Define the layer/levels of components, placement, and manner of interactions

  – Well described mapping of components/functions onto underlying physical network architecture (of computers)

Tiered architecture

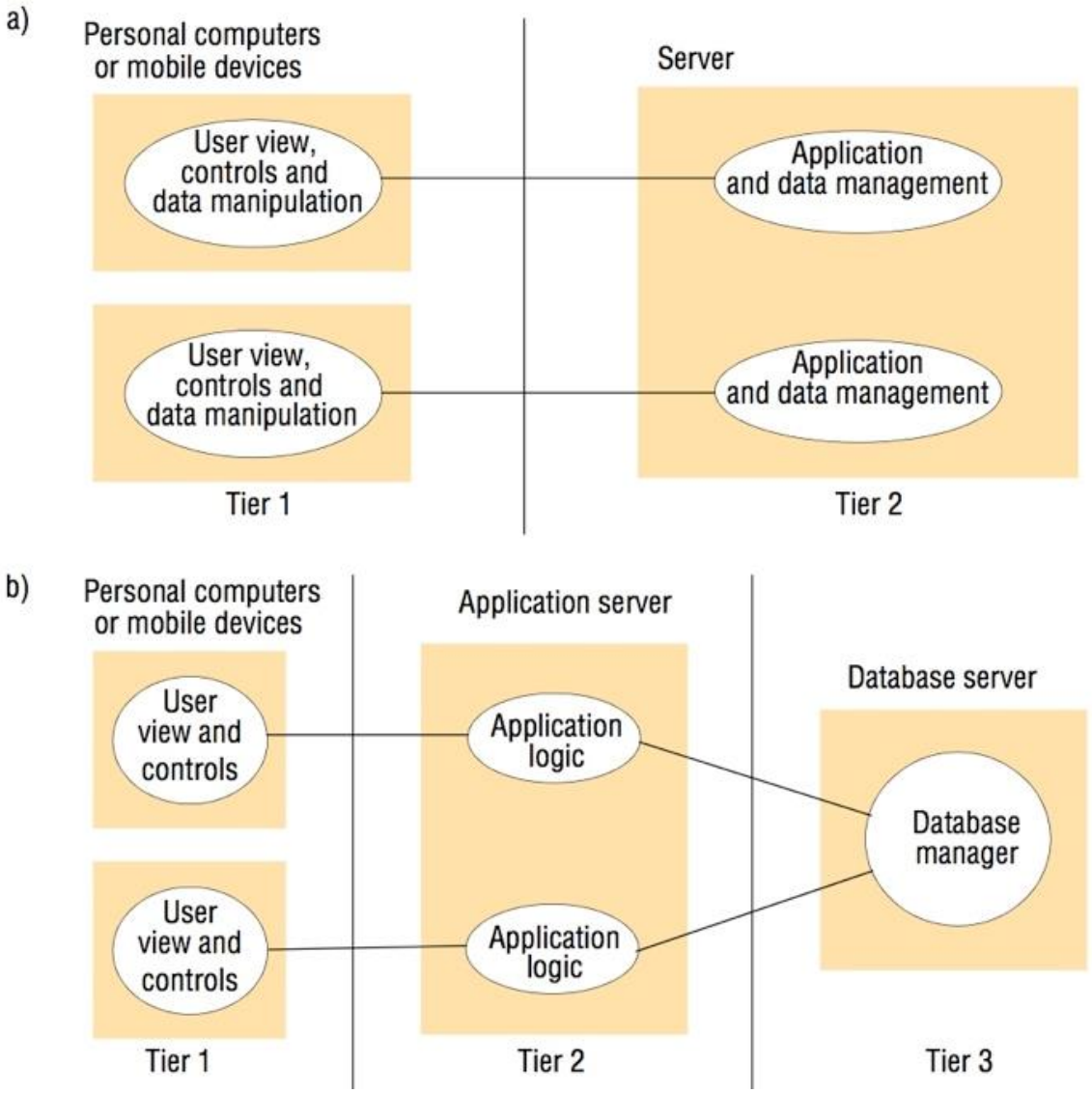
- Tiered architectures are complementary to layering.
- Layering deals with the vertical organization of services into layers of abstraction
- Tiering is a technique to organize functionality of a given layer and place this functionality into
  - appropriate servers and, as a secondary consideration, on to
  - physical nodes.
- This technique is most commonly associated with the organization of applications but it also applies to all layers of a distributed systems architecture.

Tiered architecture

Consider the functional decomposition of a given application, as follows:

- The presentation logic
  - concerned with handling user interaction and updating the view of the application as presented to the user
- The application logic
  - concerned with the detailed application-specific processing associated with the application (also referred to as the business logic, although the concept is not limited only to business applications)
- The data logic
  - concerned with the persistent storage of the application, typically in a database management system

Two-tier and three-tier architectures



Multi-tier architectures

- n-tiered (or multi-tier) solutions
- Application domain is partitioned into n logical elements, each mapped to a given server element.
- Example: Wikipedia, the web-based publicly editable encyclopaedia, adopts a multi-tier architecture to deal with the high volume of web requests
  - up to 60,000 page requests per second.

System architectures

- Architectural design has a major impact on performance, reliability, and security. In a distributed system, processes with well-defined responsibilities interact with each other to perform a *useful* activity.

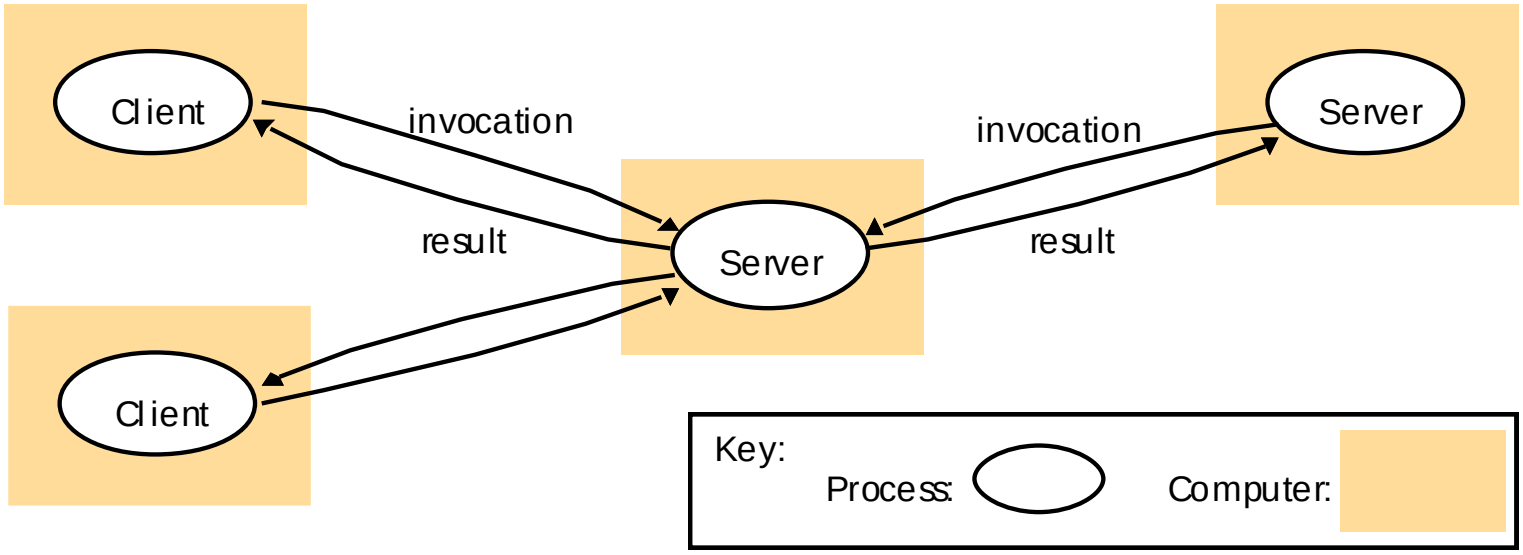
- Main types of architectural models:
  1. The C-S model
  2. The Multi-Server model
  3. The Proxy Servers and Caches model
  4. The Peer Process model

- Variations on the C-S model:
  5. Mobile agents model
  6. Thin client model

The Client-Server (C-S) model

- Widely used, servers/clients on different computers provide services to clients/servers on different computers via request/reply messaging.

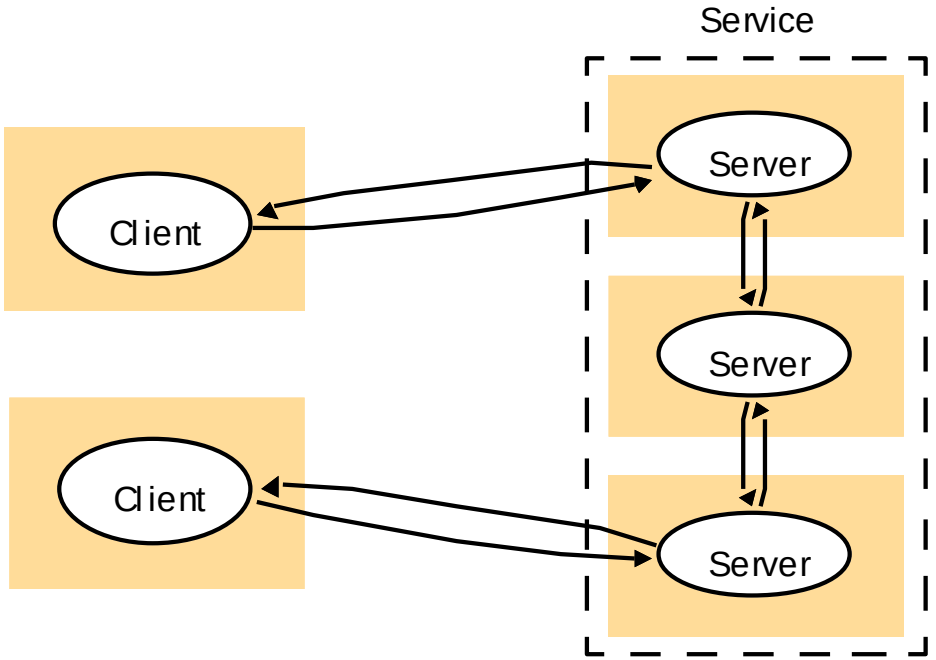
- Servers could also become clients in some services, and vice-versa, e.g., for web servers and web pages retrievals, DNS resolution, search engine-servers and web ‘crawlers’, which are all independent, concurrent and asynchronous (synchronous?) processes.



The Multi-Server model

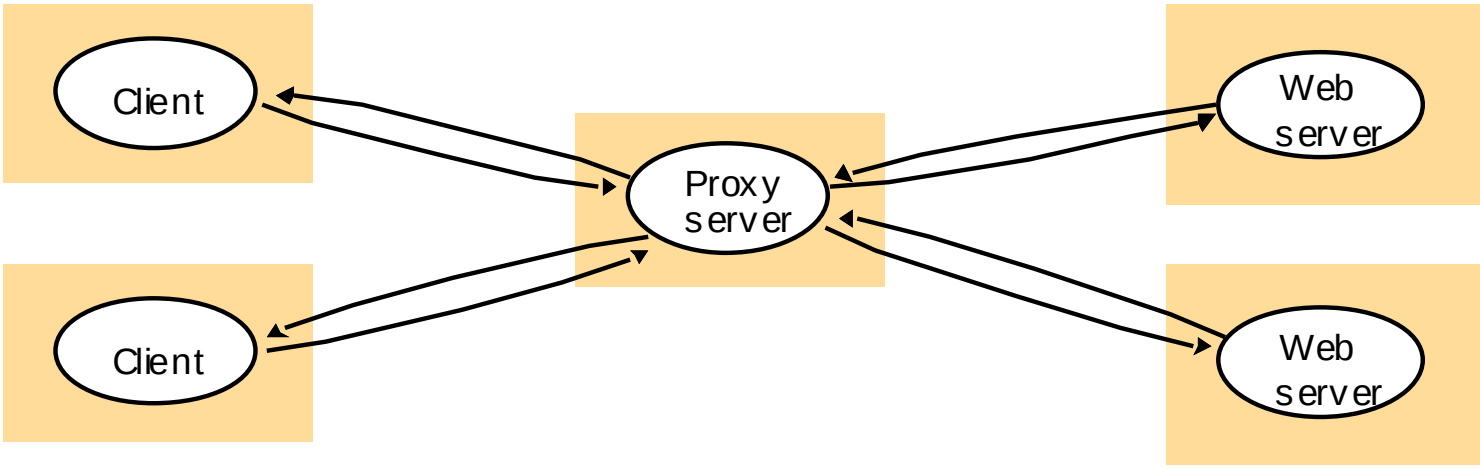
- A DS with multiple, interacting servers responding to parts of a given request in a cooperative manner.

- Service provision is via the partitioning and distributing of object sets, data replication, (or code migration).
  - E.g., A browser request targeting multiple servers depending on location of resource/data OR replication of data at several servers to speed up request/reply turnaround time, and guarantee availability and fault tolerance – consider the Network Information Service (NIS) replication of network login-files for user authorization.
  - Replication is used to increase performance and availability and to improve fault tolerance. It provides multiple consistent copies of data in processes running in different computers.



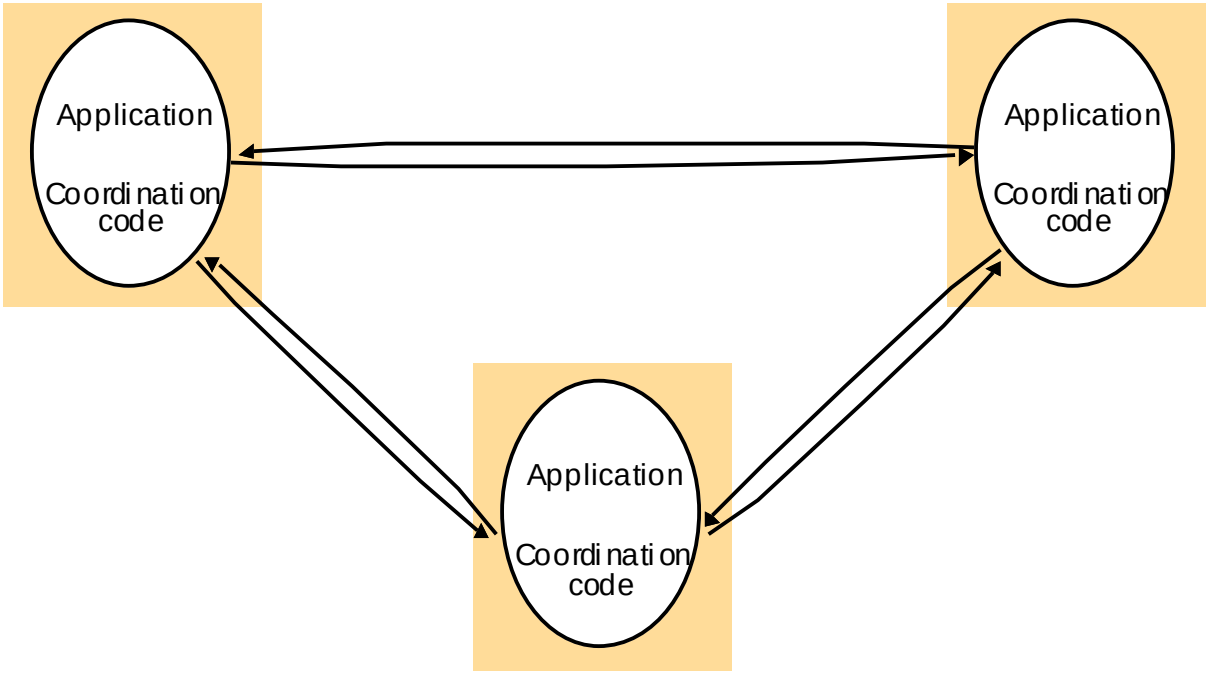
The Proxy Servers and Caches model

- A *cache* is a store of recently used data objects that is closer than the objects themselves.
- Caching frequently used: objects/data/code, which can be collocated at all clients, or located at a single/multiple 'proxy' server(s) and accessed/shared by all clients.
- When requested object/data/code is not in cache is it fetched or, sometimes, updated. E.g., clients caching of recent web pages.
- Web proxy servers provide a shared cache of web resources for the client machines at a site or across several sites. The purpose of proxy servers is to increase availability and performance of the service by reducing the load on the wide-area network and web servers. Proxy servers can take on other roles: e.g., they may be used to access remote web servers through a firewall.



The Peer Process model

- Without any distinction between servers and clients.
- All processes interact and cooperate in servicing requests.
- Processes are able to maintain consistency and needed synchronization of actions; and pattern of communication depends on the application.
- E.g., consider a ‘whiteboard’ application where multiple peer processes interact to modify a shared picture file – interactions and synch done via middleware layer for notification/group comm.



Peer-to-peer systems

Three generations of peer-to-peer system and application development can be identified:

- The *first generation* was launched by the Napster music exchange service
- A *second generation* of files sharing applications offering greater scalability, anonymity and fault tolerance followed:
  - Freenet, Gnutella, Kazaa and BitTorrent

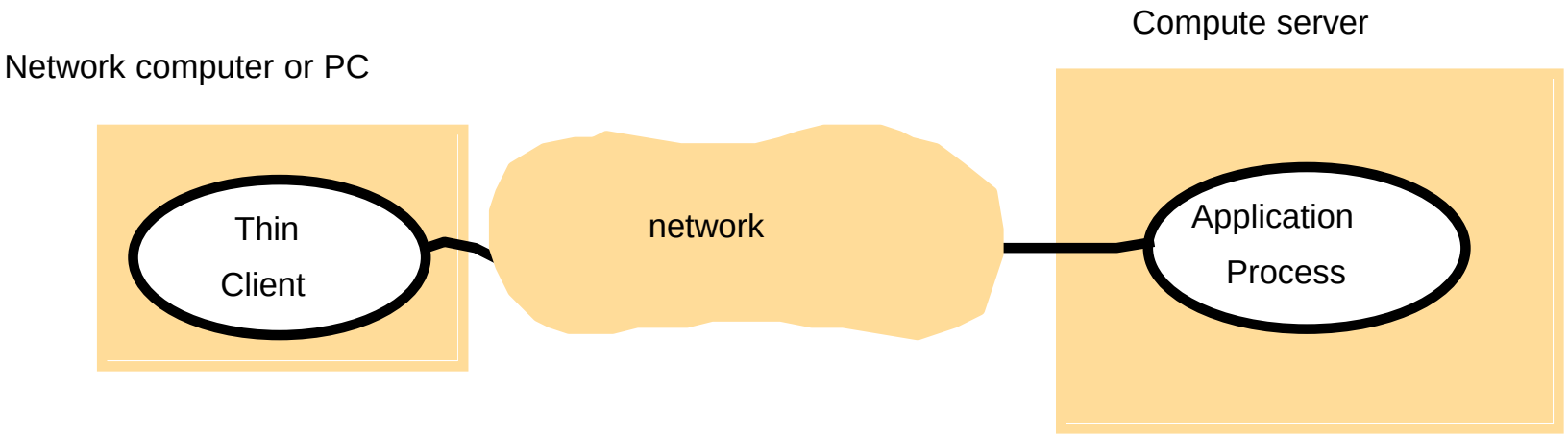
- The *third generation* - middleware layers for the application-independent management of distributed resources on a global scale.
  - Designed to place resources (data objects, files) on a set of computers that are widely distributed throughout the Internet and to route messages to them on behalf of clients,
    - relieving clients of any need to make decisions about placing resources and to hold information about the whereabouts of the resources they require.
    - Examples include: Pastry, Tapestry, CAN, Chord and Kademlia

Mobile agents model

- Both code and associated data are migrated to a number of computers to carry out specified functions/tasks, and eventually returning results
  - A variant of the C-S model
- It tends to minimize delays due to communication (vis-à-vis static clients making multiple requests to servers).
  - E.g., a software installation-agent installing applications on different computers for given hardware-configs; or price compare-agent checking variations in prices for a commodity; or a worm agent that looks for idle CPU cycles in cluster-computing.
- Caution: security threat due to potential ‘Trojan Horse’ problem in migrated code, incomplete exec or ‘hanging’ of agents.
- The mobile agents may not be able to complete their tasks if they are refused access to the information they need.

Thin Client model

- Each client computer supports a layer of software which invokes a remote *compute server* for computational services.
  - A variant of the C-S/Network computer model,
- The compute server will typically be a multiprocessor or cluster computer.
- If the application is interactive and results are due back to client-user, delays and communication can eclipse any advantages.



Resource-based architectures: RESTful architectures

- View a distributed system as a collection of resources, individually managed by components.

- Resources may be added, removed, retrieved, and modified by (remote) applications.

Resource-based architectures

1. Resources are identified through a single naming scheme
2. All services offer the same interface
3. Messages sent to or from a service are fully self-described
4. After executing an operation at a service, that component forgets everything about the caller

Publish-subscribe architectures

- Dependencies between processes as loose as possible
- Separation between processing and coordination
- View a system as a collection of autonomously operating processes.
- **Coordination** encompasses the communication and cooperation between processes.