Lecture 2
CMPU4021 Distributed Systems

Architectural Models for Distributed Systems

Based on Chapter 2: System models, Coulouris, Dollimore and Kindberg, Distributed Systems: Concepts and Design

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.
    - CS model can be modified by:
      - Partition/replication of data at cooperating servers
      - Caching data by proxy servers and clients
      - Use of mobile code (code migration) and mobile agents
      - Open systems (ease of adding/removing mobile devices)

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

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

Introduction

- No concept of global clock in DS, and local clocks are not synchronized
- Message passing is only means of communication.
- Distributed systems suffer
  - delays, failures (failure/delay due to bad links or congestion)
  - security attacks (secure channels or access/capability lists)

Introduction

- The lecture focuses on

  – Variants of architectural models – layered, C-S, aspects of DS for being open systems

  – Fundamental models - abstracting models to reveal
  - design issues,
  - properties,
  - difficulties,
  - correctness,
  - reliability,
  - security, and variations in performance requirements,
  - issues of heterogeneity (h'ware, s'ware, network types, clocks, data types/formats, integrity, QoS)

Architectural models

- *Architecture* – structure which defines the placement of system components, to guarantee reliability, manageability, adaptability, and cost-effectiveness – a consistent frame of reference

- An architectural model of a distributed system first simplifies and abstracts the functions of the individual components of a distributed system and then it considers:
  - the placement of the components across a network of computers – seeking to define useful patterns for the distribution of data and workload;
  - The interrelationships between the components – that is, their functional roles and the patterns of communication between them.

Architectural Models (I)

- An initial simplifications is achieved by classifying processes as *server process*, *client process* and *peer processes* (processes that cooperate and communicate in a symmetrical manner)

- Variants of the C-S Model:

  – Allowing code/data migration from one process to another allows task delegation, reducing delays and communication.

  - Dynamic adding/removing of mobile devices and discovery of added/deleted resources – supporting ubiquitous and mobile DS.

Generations of distributed systems

Distributed systems:	*Early*	*Internet-scale*	*Contemporary*
Scale	Small	Large	Ultra-large
Heterogeneity	Limited (typically relatively homogenous configurations)	Significant in terms of platforms, languages and middleware	Added dimensions introduced including radically different styles of architecture
Openness	Not a priority	Significant priority with range of standards introduced	Major research challenge with existing standards not yet able to embrace complex systems
Quality of service	In its infancy	Significant priority with range of services introduced	Major research challenge with existing services not yet able to embrace complex systems

Communicating entities and communication paradigms

Communicating entities (what is communicating)		*Communication paradigms (how they communicate)*		
System-oriented entities	*Problem-oriented entities*	*Interprocess communication*	*Remote invocation*	*Indirect communication*
Nodes	Objects	Message passing	Request-reply	Group communication
Processes	Components	Sockets	RPC	Publish-subscribe
	Web services	Multicast	RMI	Message queues
				Tuple spaces
				DSM

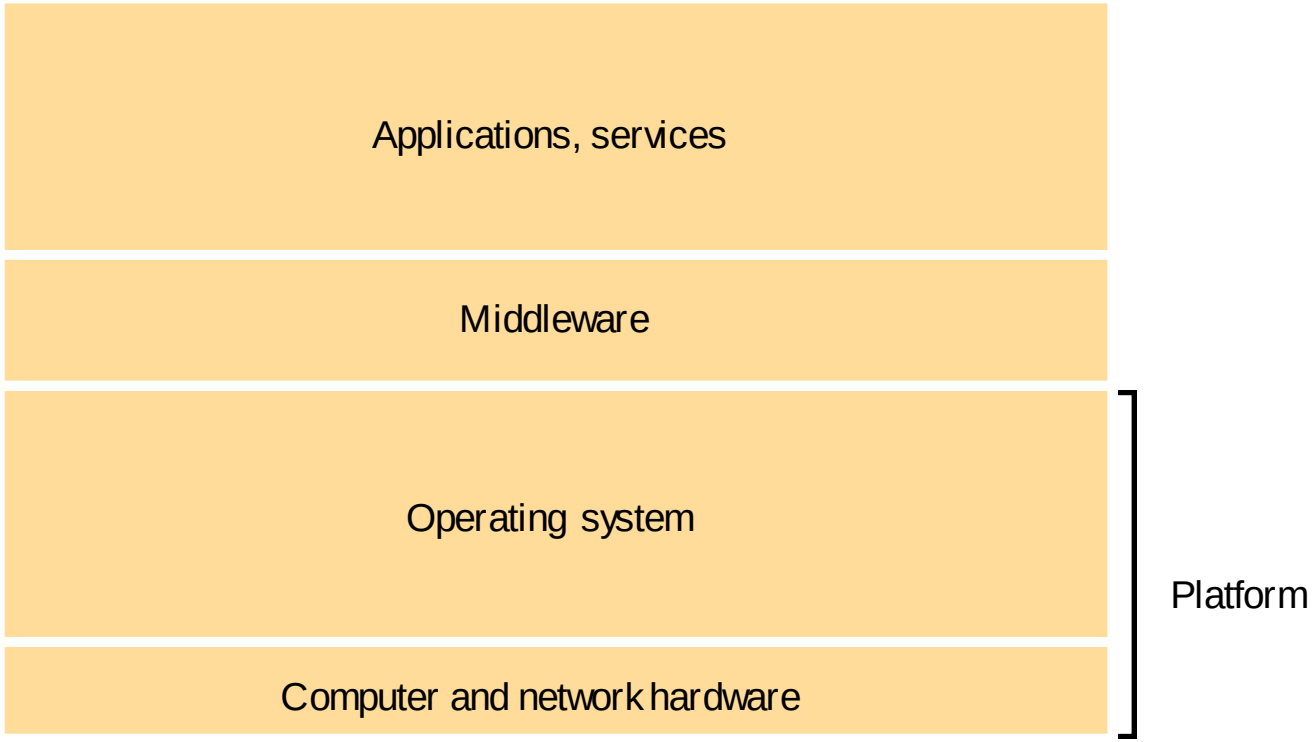
Software Architecture

– Layers of software modules in the same or different computers, and the manner of interactions among modules and services each provides and accessibility

– Each layer/module offers a kind of services. E.g., the Internet has Network Time Protocol, used by server processes solely responsibility for reading and sending ‘LocalHostTime’ info to clients on request

- This process and service-oriented view can be expressed in terms of *service layers*.

Software and hardware service layers in distributed systems



Platform

- **Platform**
  - The lowest-level hardware and software layers, which provide services to the upper Middleware and Application Layers.

- The design and implementation of the platform is independent of the design and implementation of the upper layers
  - the ‘glue’ or interface between the Platform and ‘upper’ layers is realized through systems programming of Application Programming Interfaces (APIs) or Bundles.

Middleware

- Layer of software, which masks heterogeneity in DS, and facilitates API programming by application programmers
- It is represented by processes and objects on ‘service providing’ computers for communication and resource (files/data, code, device) sharing.
- Provides building blocks (classes, objects, interfaces, library modules) for constructing DS components
- It abstracts device-level communication requirements via, e.g., Remote Method Invocation (RMI), Remote Procedure Calling (RPC), request/send, notify, selective or group communication, broadcast, protection/security, data replication/integrity

Middleware (I)

- *Examples*: Java RMI, Common Object Request Broker Architecture (CORBA), Distributed Component Object Model (DCOM), ISO Reference Model for Open Distributed Processing (RM-ODP).

- Infrastructural services:
  - facilities/libraries for naming, security, transactions, persistency, notification

- Domain-specific services:
  - facilities/libraries for communication and local (hidden) services

Middleware – limitations

- Many distributed applications rely entirely on the services provided by the available middleware to support their needs for communication and data sharing.

  E.g., systems constrained by use of only RMI for communication and data sharing – consider a C-S model for request/reply of names and addresses in a database.

- Abstraction and full ‘independence’ of application layer is not achieved, yet!

  (E.g., consider a C-S-based mail service provider where a server must add its own error detection/recovery mechanism for retransmission if a link breaks on large transmissions. Or vice-versa, is possible)

- Arguably, error detection/recovery needs to be at several levels, not only at middleware

Tiered architecture

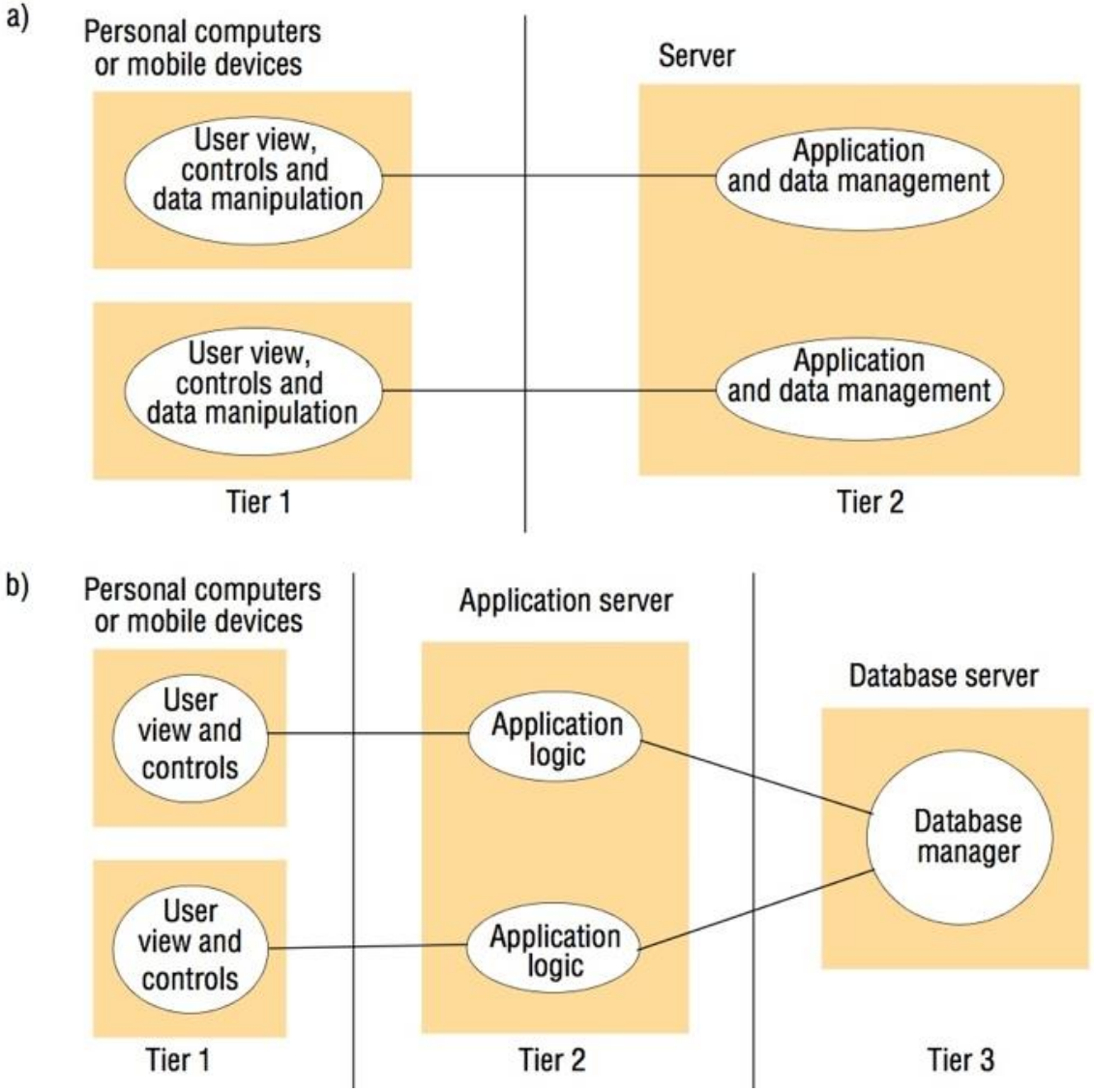
- Tiered architectures are complementary to layering.
- Layering deals with the vertical organization of services into layers of abstraction
- Tiering is a technique to organize functionality of a given layer and place this functionality into
  - appropriate servers and, as a secondary consideration, 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 encyclopedia, adopts a multi-tier architecture to deal with the high volume of web requests
  - up to 60,000 page requests per second.

System architectures

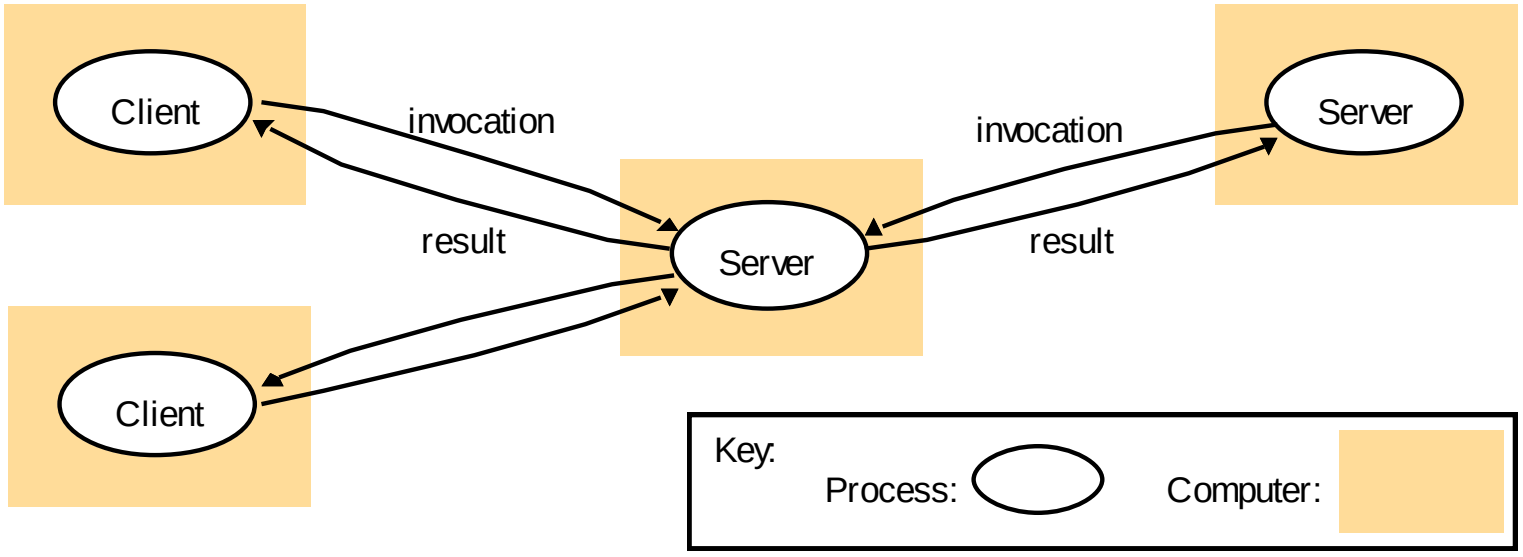
- Architectural design has a major impact on performance, reliability, and security. In a distributed system, processes with well-defined responsibilities interact with each other to perform a *useful* activity.

- 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 code model
  6. Mobile agents model
  7. Network computer model
  8. Thin client model
  9. Mobile devices and spontaneous networking model

The Client-Server (C-S) model

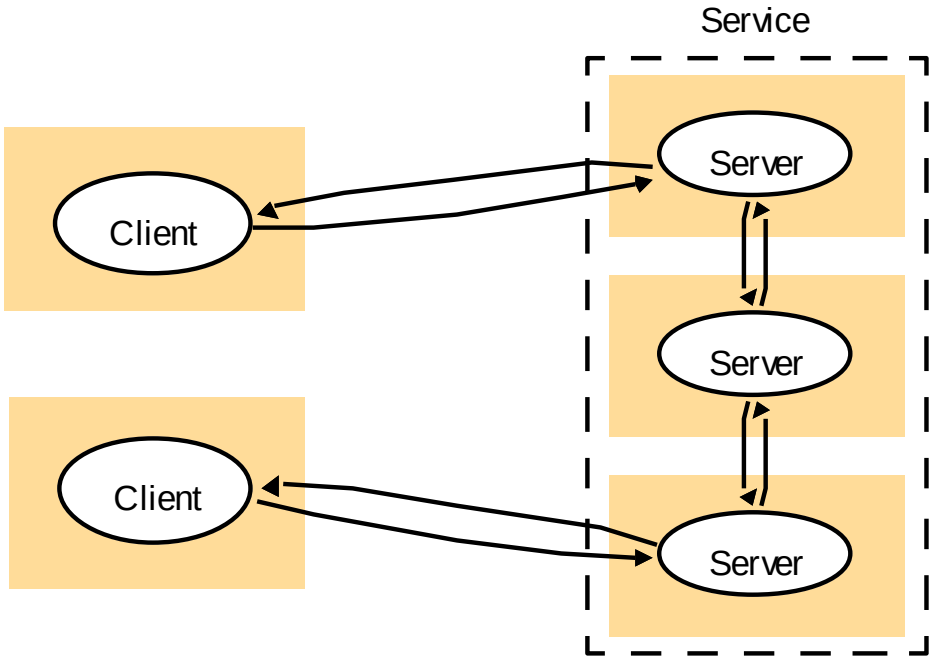
- Widely used, servers/clients on different computers provide services to clients/servers on different computers via request/reply messaging.
- Servers could also become clients in some services, and vice-versa, e.g., for web servers and web pages retrievals, DNS resolution, search engine-servers and web ‘crawlers’, which are all independent, concurrent and asynchronous (synchronous?) processes.



The Multi-Server model

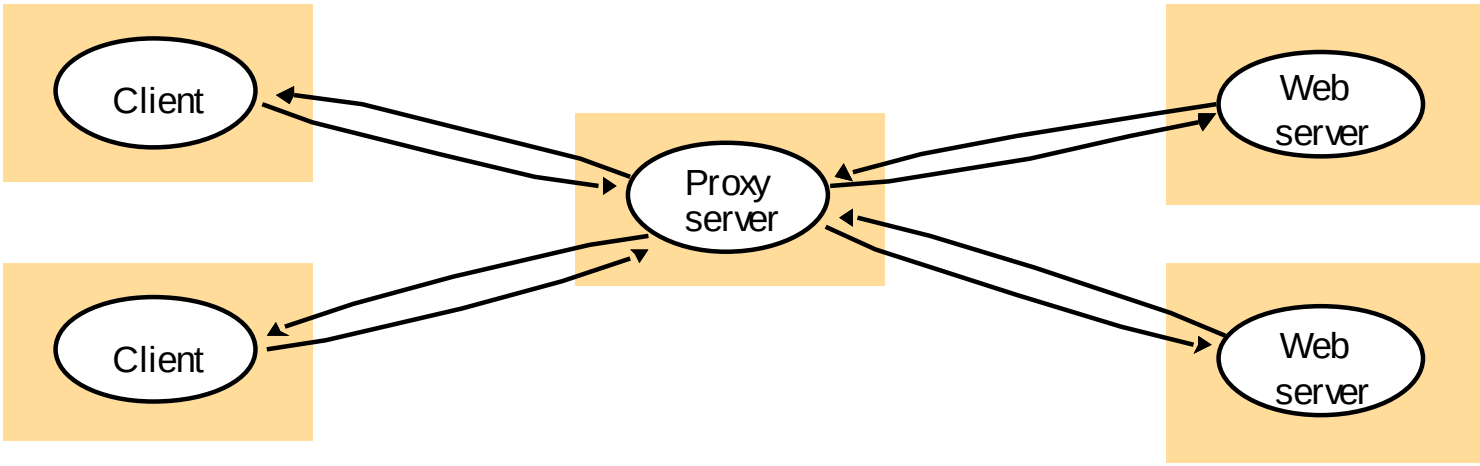
- A DS with multiple, interacting servers responding to parts of a given request in a cooperative manner.

- 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.



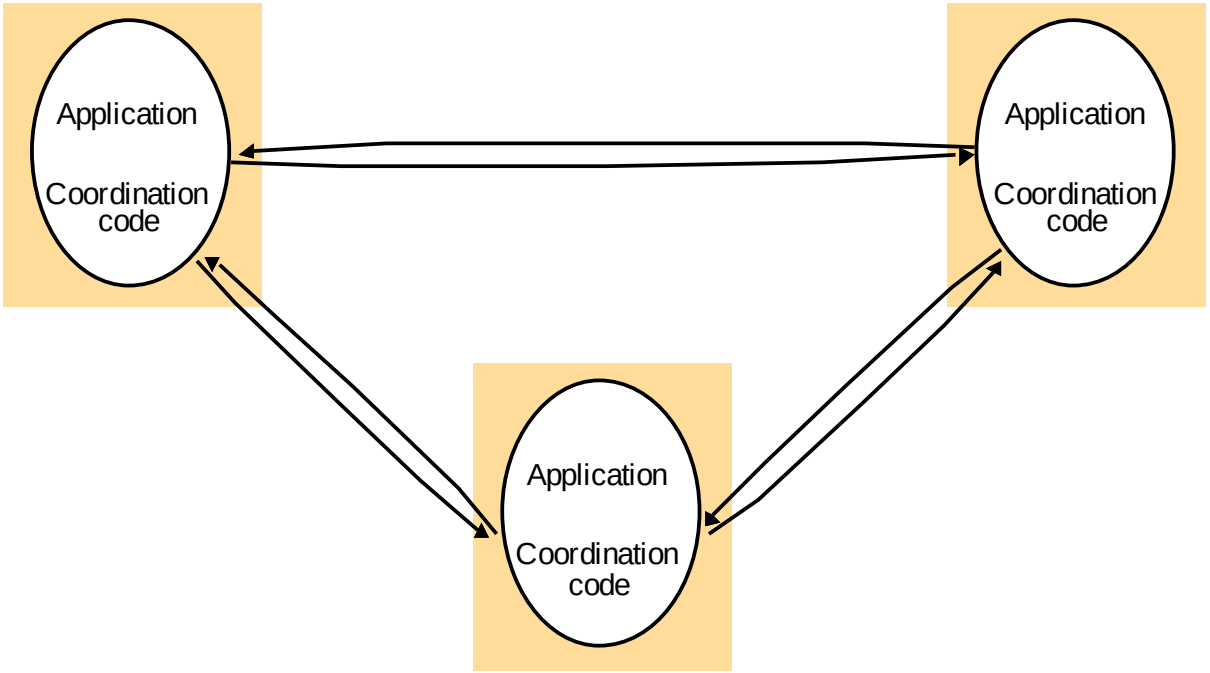
The Proxy Servers and Caches model

- 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

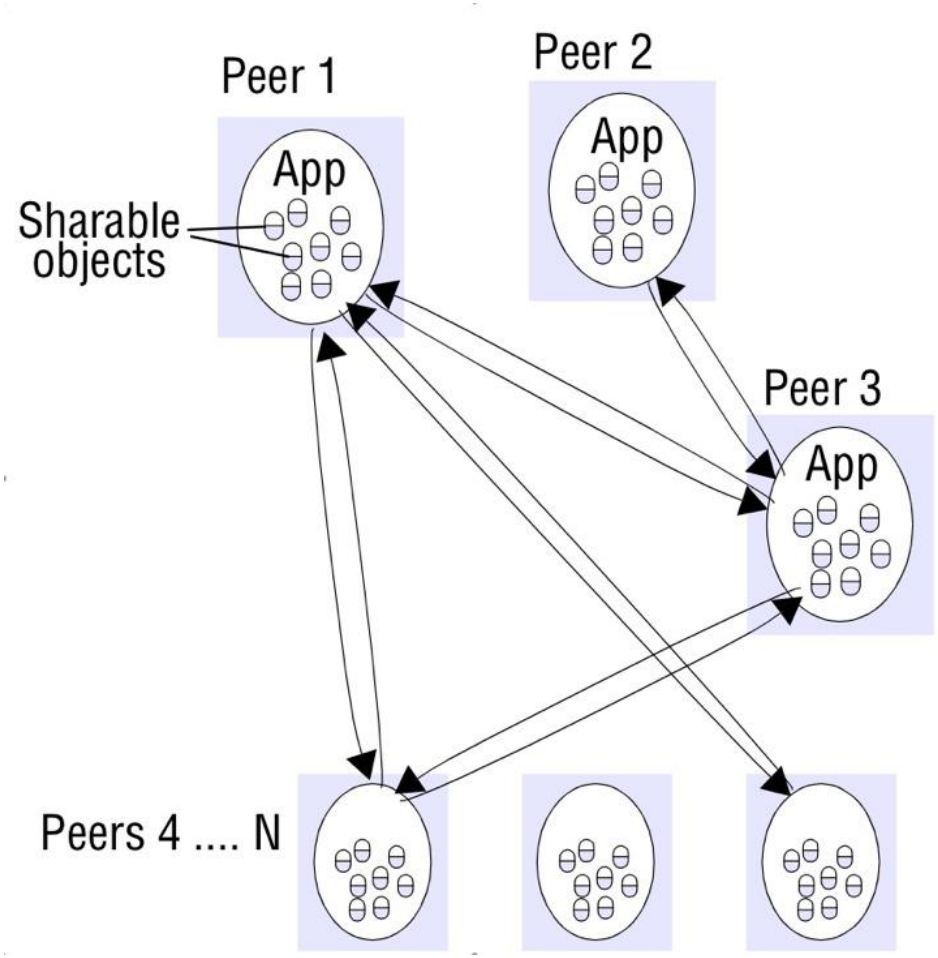
- Without any distinction between servers and clients.
- All processes interact and cooperate in servicing requests.
- Processes are able to maintain consistency and needed synchronization of actions; and pattern of communication depends on the application.
- E.g., consider a ‘whiteboard’ application 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 characteristics

- Their design ensures that each user contributes resources to the system.
- They may differ in the resources that they contribute, but all the nodes in a peer-to-peer system have the same functional capabilities and responsibilities.
- Their correct operation does not depend on the existence of any centrally administered systems.
- They can be designed to offer a limited degree of anonymity to the providers and users of resources.
- A key issue for their efficient operation
  - the choice of an algorithm for the placement of data across many hosts and subsequent access to it in a manner that balances the workload and ensures availability without adding undue overheads.

Peer-to-peer architecture



Peer-to-peer: issues to handle

- Connectivity
  - how to find and connect other P2P nodes that are running in the network
    - unlike traditional servers, they don't have a fixed, known IP address
- Instability
  - nodes may always be joining and leaving the network
- Message routing
  - how messages should be routed to get from one node to another
    - the two nodes may not directly know about each other
- Searching
  - how to find desired information from the nodes connected to the network
- Security - extra issues including
  - nodes being able to trust other nodes,
  - preventing malicious nodes from doing corrupting the P2P network or the individual nodes,
  - being able to send and receive data anonymously, etc.

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

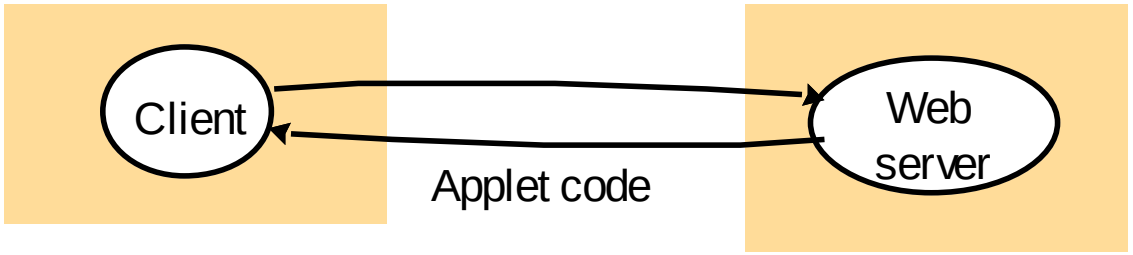
- The *third generation* - middleware layers for the application-independent management of distributed resources on a global scale.
  - Designed to place resources (data objects, files) on a set of computers that are widely distributed throughout the Internet and to route messages to them on behalf of 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 code model

- A variant of the C-S model
- Code migration/mobility allows DS objects to be moved to a client (or server) for execution/processing in response to a client request, e.g., migration of an applet to a local browser – avoiding delays and comm overhead.
  - E.g., dynamic/periodic auto-migration of server-resident code/data to a client .
- The *push model* – one in which the server initiates the interaction by sending update data to clients' applets to a) refresh, say, a stocks web page, b) perform buy/sell condition-checks on clients side, and c) automatically notify the server to buy/sell – all done while the user-application is off or onto other things.
  - Caution: security threat due to potential 'Trojan Horse' problem in migrated code.

Mobile code model example: Web applets

a) client request results in the downloading of applet code



b) client interacts with the applet

Mobile Code

- Advantages:
  - Doing computation close to the user
    - as in Applets example
  - Enhancing browser
    - e.g. to allow server initiated communication
  - Cases where objects are sent to a process and the code is required to make them usable
    - e.g. as in RMI
- Disadvantage:
  - Security threat due to potential ‘Trojan Horse’ problem in migrated code

Mobile agents model

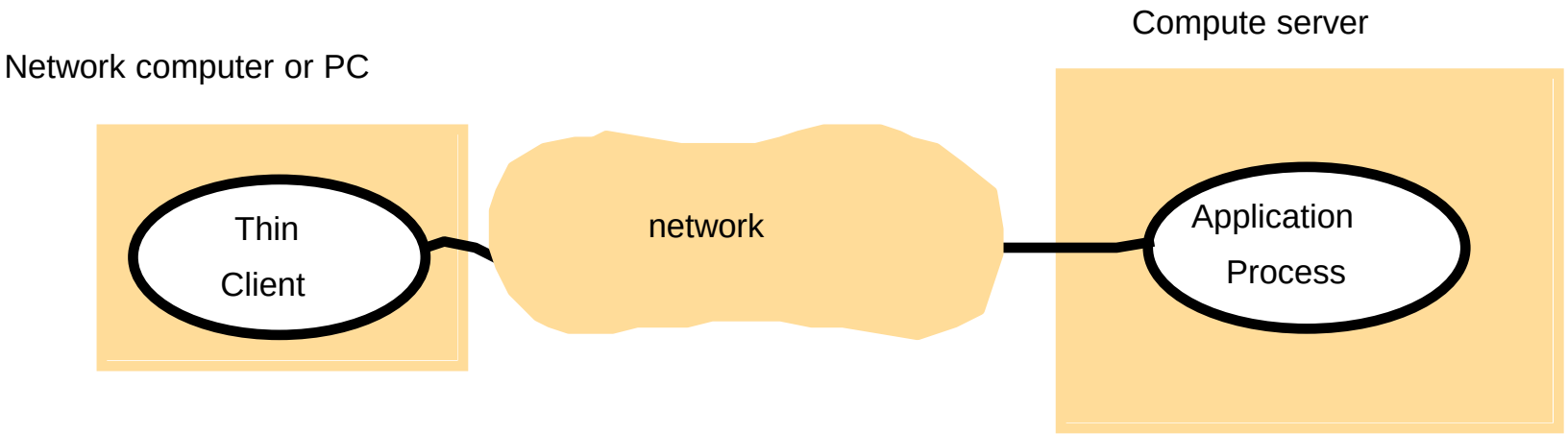
- A variant of the C-S model, where both code and associated data are migrated to a number of computers to carry out specified functions/tasks, and eventually returning results.
- It tends to minimize delays due to communication (vis-à-vis static clients making multiple requests to servers).
  - E.g., 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.

Network computer model

- A variant of the C-S model, where each client computer downloads the OS and application software/code from a server.
- Applications are then run locally and files/OS are managed by server; this way, a client-user can migrate from computer to computer and still access the server, and all updates are done at the server side.
- Local disks are used primarily as cache storage.
- Has low management and hardware costs per computer

Thin Client model

- A variant of the C-S/Network computer model, where each client computer (instead of downloading the OS and application software/code from a server), supports a layer of software which invokes a remote *compute server* for computational services.
- 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.



Thin Client model

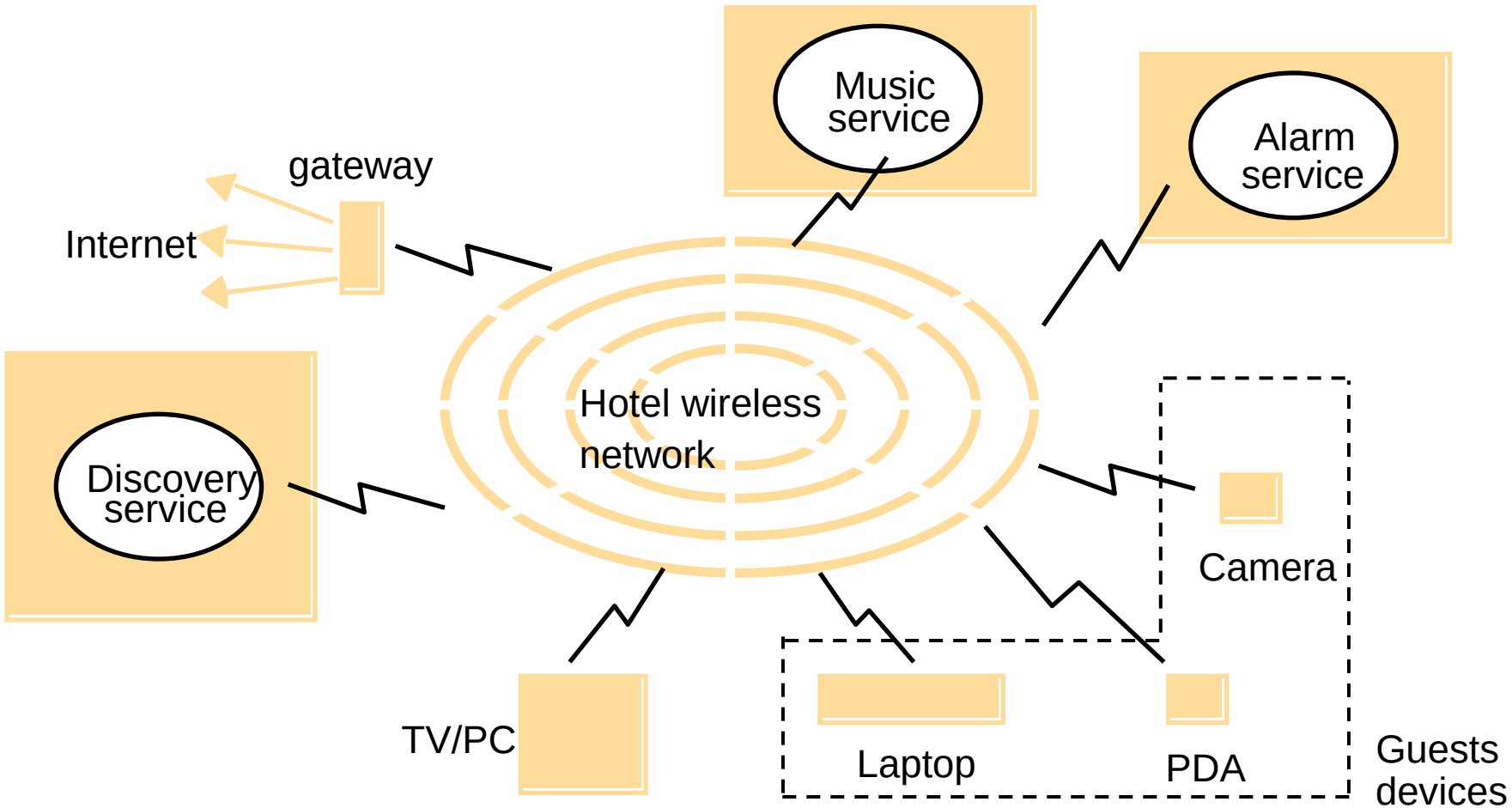
- Drawbacks:
  - delays experienced by users can be increased to unacceptable levels by the need to transfer images and vector information between the thin client and the application process,
    - due to both network and operating system latencies.

- Thin client concept has led to the emergence of virtual network computing (VNC)
  - providing remote access to graphical user interfaces
  - a VNC client (or viewer) interacts with a VNC server through a VNC protocol.
  - Examples: RealVNC, Apple Remote Desktop, TightVNC, Aqua Connect

Mobile devices and Spontaneous networking model

- A form of distributed computing that integrates mobile devices
  - small portable devices: laptops, PDA, mobile phones, digital cameras, wearable computers smart watches)
  - and non-mobile embedded microcomputers – washing machines, set-top boxes, home appliances, automobiles, controllers, sensors
  - by connecting both mobile and non-mobile devices to networks to provide services to user and other devices from both local and global points.

Spontaneous networking in a hotel



Mobile devices and Spontaneous networking model (I)

Key Features:

- Easy connection to a local network
- Easy integration with local services.

Issues:

- Limited connectivity:
  - Users are not always connected as they move around. E.g., they may be intermittently disconnected from a wireless network as they move on a train through tunnels. Issue: how to support the user so that they can work while disconnected.
- Security and privacy:
  - A facility that enables users to access their home intranet while on the move may expose data that is supposed to remain behind the intranet firewall, or it may open up the intranet to attacks from outside.

Mobile devices and Spontaneous networking model (II)

- *Discovery service*:
  - purpose is to accept and store details of services that are available on the network and to respond to queries about them.

- Discovery service offers two interfaces:
  - A *registration service* – accepts registration requests from servers and records the details that they contain.
  - A *lookup service* accepts queries concerning available services and searches its database for registered services that match the queries.

DESIGN REQUIREMENTS FOR DISTRIBUTED ARCHITECTURES

Design requirements: *Quality of service* (QoS) (1)

- The main non-functional properties of systems that affect the quality of the service:
  - Reliability
  - Security
  - Performance
  - Adaptability – to meet changing system configuration and resource availability.

Design requirements for distributed architectures (2)

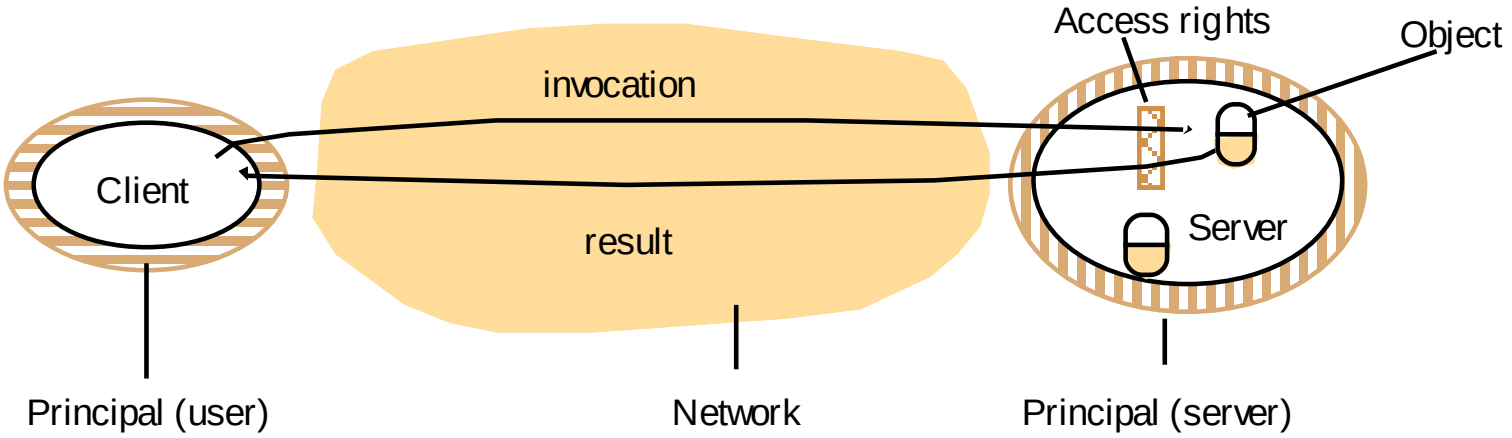
- Use of caching and replication
  - Much of challenges (due to performance constraints) have been mitigated by use of caching and replication.
  - Techniques for cache updates and cache coherence based on cache coherent protocols (e.g., web-caching protocol, a part of the HTTP cache coherent protocol).
  - *Web-caching protocol*:

    A browser or proxy can validate a cached response by checking with the original web server to see whether it is still up to date. The *age* of a response is the sum of the time the response has been cached and the server time.

Security model

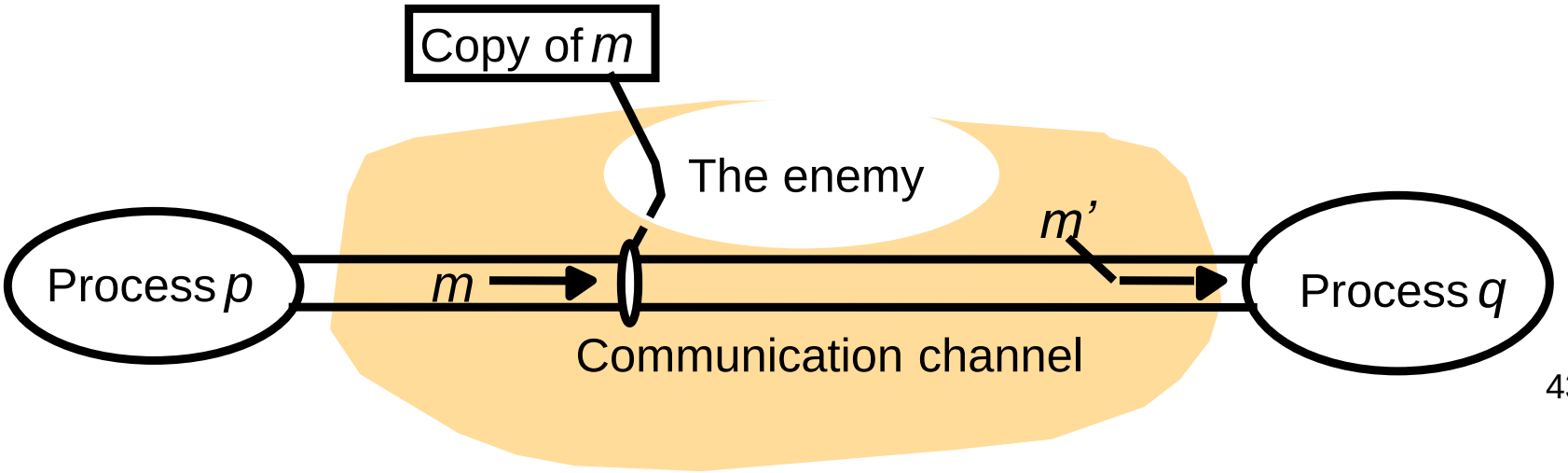
- Security model:
  - The security of a distributed system can be achieved by securing processes and comm channels by protecting objects they encapsulate against unauthorized access.

- Protecting objects
  - Uses *access rights* (allowable ops on given object by a given client or *principal*). A principal may be a user or a process. The server is responsible for verifying the identity of the principal behind each invocation and checking that they have sufficient access rights to perform the requested operation on the particular object invoked.



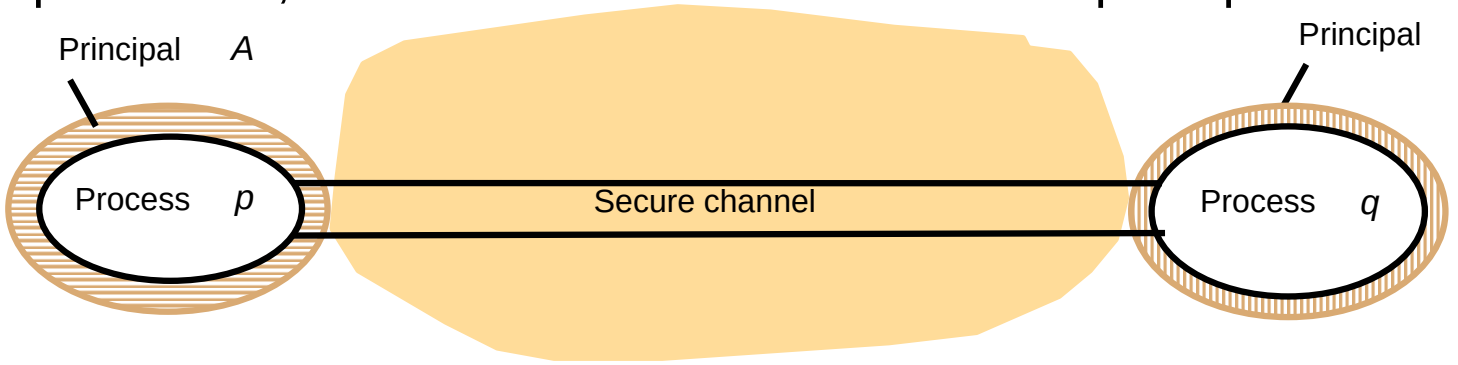
Security model: Securing processes and their interactions

- Enemy model: - eavesdropping via message interception, repeated trials of network access. E.g.,
  - Threats to processes: inserting and forwarding incorrect IP addresses in message to servers or clients to confuse or disguise the enemy-sender.
  - Threats to communication channels: copying, altering or inserting incorrect data into message streams to deceive or replicate unauthorized transactions (e.g. banking)
- These threats can be defeated by the use of *secure channels*
  - *which are based on cryptography and authentication*



Security model: Defeating security threats

- *Cryptography and shared secrets*
  - Cryptography – the science of keeping messages secure
  - Encryption – the process of scrambling a message in such a way as to hide its contents
- *Authentication* – proving the identities supplied by their senders.
- Secure channels
  - Encryption and authentication are used to build secure channels as a service layer on top of existing comms services. A secure channel is a communication channel connecting a pair of processes, each of which acts on behalf of a principal.



Secure Channels

- Each of the processes knows reliably the identity of the principal on whose behalf the other process is executing.
  - This enables the server to protect its objects correctly and allows the client to be sure that is receiving results from a bona fide server.
- Ensure the privacy and integrity (protection against tampering) of the data transmitted across it.

- Each message includes a physical or logical time stamp to prevent messages from being replayed or reordered.
  - Examples: Virtual private networks (VPNs) and the Secure Sockets Layer (SSL) protocol.

Security model

- Other possible threats from an enemy
  - *Denial of service*
    - excessive and pointless invocations on services or message transmissions in a network, resulting in overloading of physical resources (network bandwidth, server processing capacity).
  - *Mobile code*
    - a Trojan horse role, purporting to fulfil an innocent purpose, but in fact including code that accesses or modifies resources that are legitimately available to the host process, but not to the originator of the code.

The uses of security models

- The security model we discussed provides the basis for the analysis and design of secure systems in which these costs are kept to a minimum.

- This analysis involves the construction of a *threat model*
  - listing all the forms of attack to which the system is exposed; and
  - an evaluation of the risks and consequences of each.

Summary

- Most distributed systems are arranged according to one of a variety of architectural models.
- The client-server model is prevalent – the Web and other Internet services such as ftp, news and mail , DNS are based on C-S model.
- Services such as DNS that have large numbers of users and manage a vast amount of information are based on multiple servers and use data partition and replication to enhance availability and fault tolerance.
- Caching by clients and proxy servers is widely used to enhance the performance of service.