Revision 1

DT228/4 Distributed Systems

Introduction

- A distributed system is one in which component systems are networked for communication and coordination through *message passing* only (allowing transparency of independent failure and lack of global clocks)

- 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)

Characteristics

- Concurrency - Collaborative and cooperative problem-solving (interdependencies)
- Independent failures of components - Autonomous but interdependent - requires coordination, graceful degradation
- Lack of global clocks - Local (component) clocks and relativity of time when distributed
- Heterogeneity - Hardware/software (programs, data) variations in component systems
- 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)

Distributed Systems 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)
- Middleware – a software layer that provides a programming abstraction as well as masking the heterogeneity of the underlying networks, hardware, operating systems and programming languages (e.g. CORBA, Java RMI)

  All middleware deals with the differences in operating systems and hardware.

- Mobile code – the code that can be sent from one computer to another and run at the destination (e.g. Java applets). Machine code suitable for running on one type of computer hardware is not suitable for running on another.

  Virtual machines approach – provides a way of making code executable on any hardware: the compiler for a particular language generates code for a virtual machine instead of a particular hardware order code.

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
  - OMG (CORBA)

- Everything conforms to a standard!
- *Open systems* are characterized by the fact that their key interfaces are published.
- *Open distributed systems* are based on the provision of a uniform communication mechanism and published interfaces for access to shared resources. They can be constructed from heterogeneous hardware and software, possibly from different vendors (with careful testing).

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).

- Denial of Service
- Viruses/Worms
- Password Crackers
- Packet sniffers
- Trojan horses
- Spoofing
- Mobile Code

Threats not defeated by secure channels or other cryptographic techniques

- Denial of service attacks
  - Deliberately excessive use of resources to the extent that they are not available to legitimate users
    - E.g. the Internet 'IP spoofing' attack, February 2000
- Trojan horses and other viruses
  - Viruses can only enter computers when program code is imported.
  - But users often require new programs, for example:
    - New software installation
    - Mobile code downloaded dynamically by existing software (e.g. Java applets)
    - Accidental execution of programs transmitted surreptitiously
  - Defences: code authentication (signed code), code validation (type checking, proof), sandboxing.

DS Challenges - Scalability

- A system is scalable if it will remain effective if there is a significant increase in the number of resources and the number of users
- Control cost of physical resources
- Control performance loss
- Prevent software resources running out.
  - IP Addresses (initially 32 bits). New version 128-bit
- 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 - Failure Handling I

- Tolerating failures
  - Web pages (informing users about failure)
- Recovery

  - permanent data rolled back’
- Redundancy (use of redundant components)
  - Duplication in routes, hardware,
  - DNS – every name table replicated in at least two different servers,
  - Databases – replicated in several servers

- Availability – measure of the proportion of time a system is available for use.
- DS provide a high degree of availability regarding hardware faults.

DS Challenges - Concurrency

- Resources can be shared by clients in a distributed system, therefore 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
- For each object' that represents a shared resource, its operations must be synchronised in such a way that its data remains consistent

DS Challenges – Transparency I

The concealment from the user and application programmer of the separation of components in a distributed system

- *Access transparency*: enables local and remote resources to be accessed using identical operations.

  - A GUI with folders, which is the same whether the files are local or remote.

- *Location transparency*: enables resources to be accessed without knowledge of their location.

  - Web resource names or URLs – because the part of the URL that identifies a web server domain name refers to a computer name in a domain, rather than to an Internet address.

- *Concurrency transparency*: enables several processes to operate concurrently using shared resources without interference between them.

DS Challenges – Transparency II

- *Replication transparency*: enables multiple instances of resources to be used to increase reliability and performance without knowledge of the replicas by users or application programmers.
- *Failure transparency*: enables the concealment of faults, allowing users and application programs to complete their tasks despite the failure of hardware or software components.

  - Email is eventually delivered, even when servers or communication links fail.

DS Challenges – Transparency III

- *Mobility transparency*: allows the movement of resources and clients within a system without affecting the operation of users or programs.

  - Mobile phone users (client – resource)

- *Performance transparency*: allows the system to be reconfigured to improve performance as loads vary.

- *Scaling transparency*: allows the system and applications to expand in scale without change to the system structure or the application algorithms.

System architectures

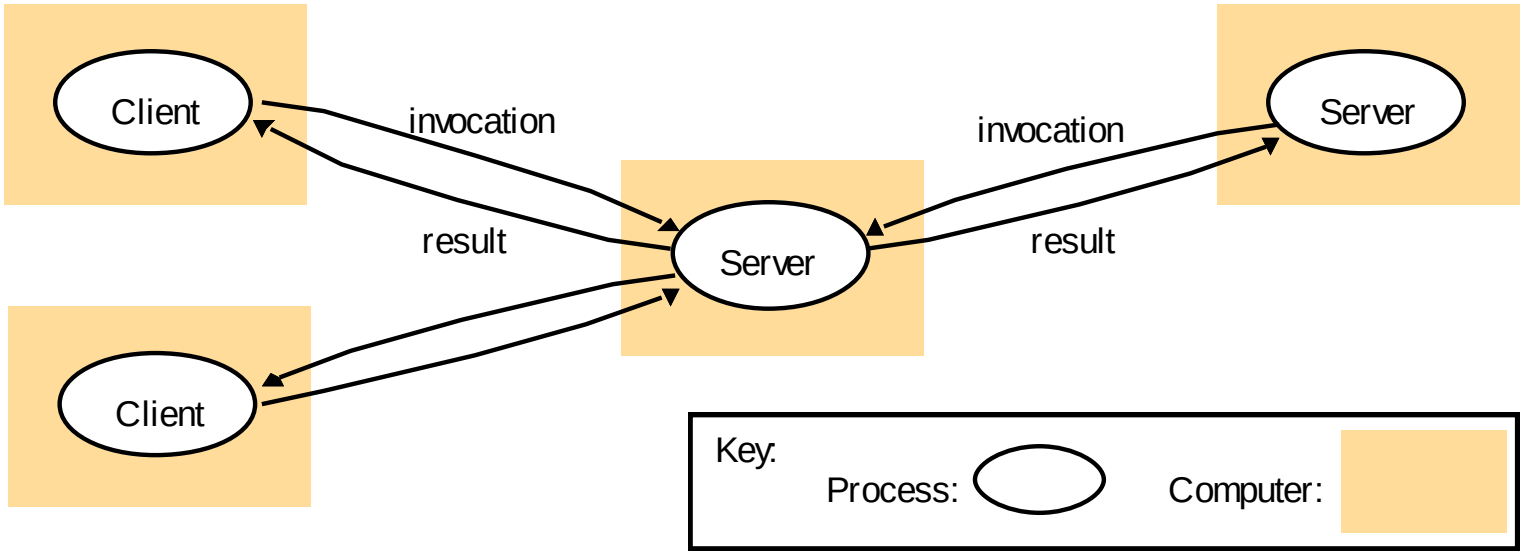
- Architectural design has a major impact on performance, reliability, and security. In a distributed system, processes with well-defined responsibilities interact with each other to perform a *useful* activity.

- 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

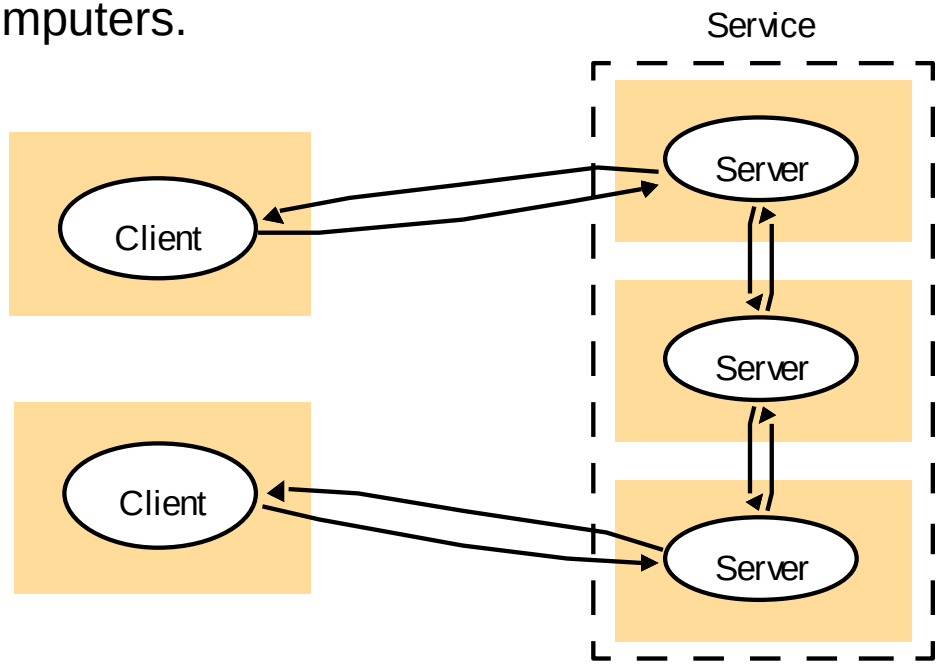
The Client-Server (C-S) model

- 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.



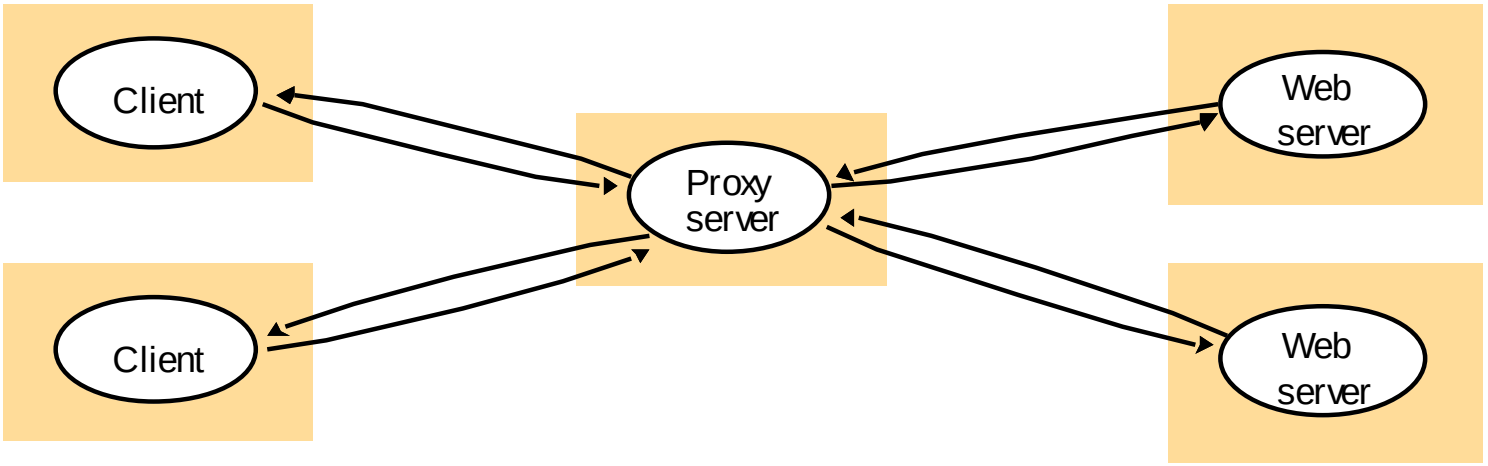
The Multi-Server model

- 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 Sun 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.



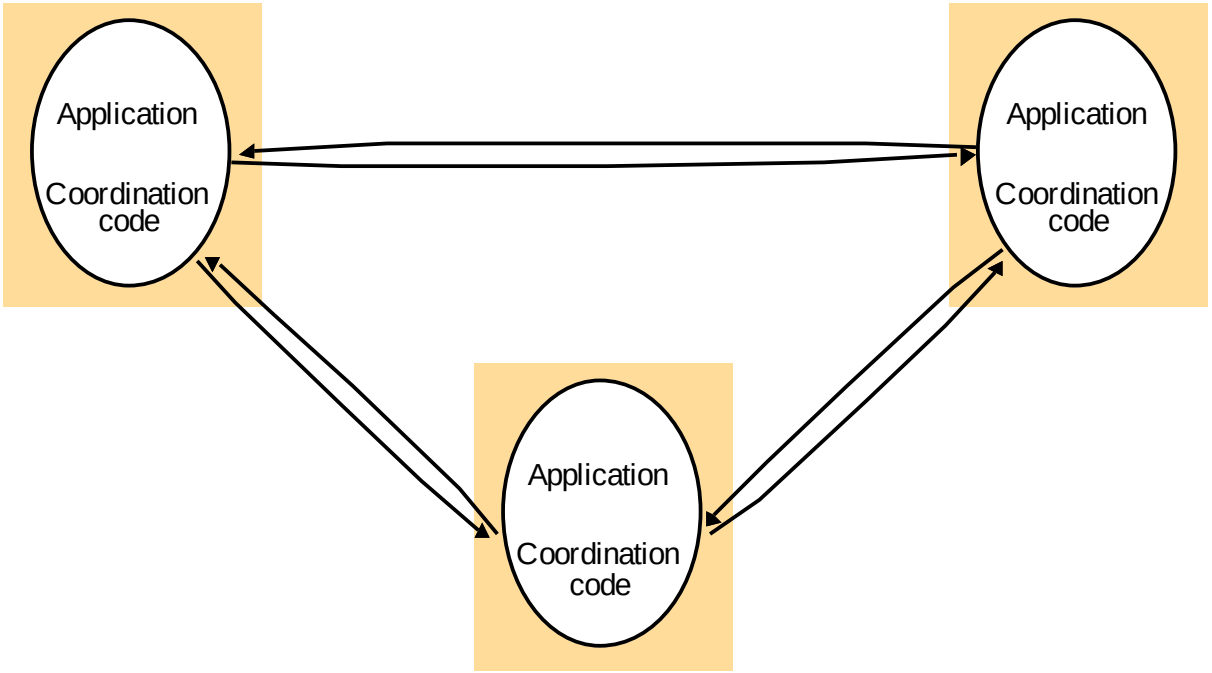
The Proxy Servers and Caches model

- 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 versus client-server

- In a peer to peer architecture, each participating computer is equally privileged. In client-server model servers on different computers provide services to clients on different computers via request/reply messaging.
- Peer to peer networks are typically ad-hoc, with participants joining and leaving as required. Each participant provides a resource, which may include storage capacity, files for download or processor time. Resources are offered directly to peers without the need for intermediate server. In C-S model server(s) is always present.

Peer-to-peer versus client-server

- Peer to peer networks are commonly used in file sharing services.
- In client-server architectures, it may be that one or more of the machines in the network assume server responsibilities, offering services to clients that may make requests. Clients do not share resources. Though servers may be dedicated machines, this term may also be used to describe the services running (as background processes) on any standard machine.

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 agents model

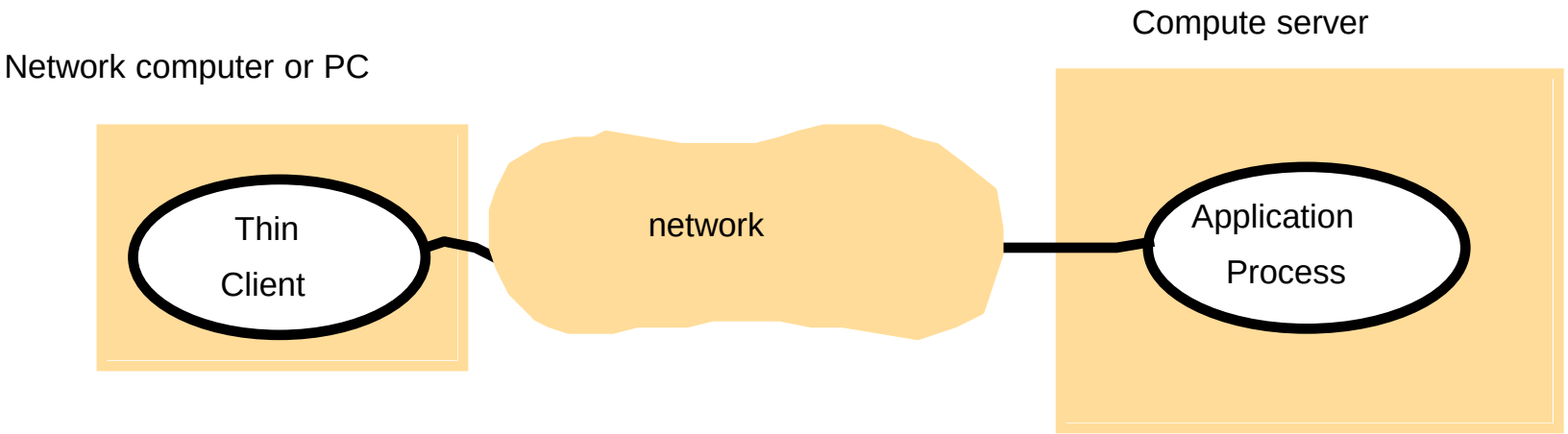
- A variant of the C-S model, where both code and associated data are migrated to a number of computers to carry out specified functions/tasks, and eventually returning results.
- It tends to minimize delays due to communication (vis-à-vis static clients making multiple requests to servers).
- E.g., 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.
- E.g., using Unix X-11 windows system – server process - with a boundary between client and server set at graphical ops/primitives level for servicing window-based objects



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.

Multicasting

- Most transmissions are point-to-point, but several involve one-to-many (either one-to-all – broadcast or selective broadcast – multicast)

- Multicasting technique allows single transmission to multiple destination (simultaneously) by using special addressing scheme

IP Multicast

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

Multicasting (I)

- Most multicast data is audio or video or both (relatively large and robust against data loss).
- Multicast data is sent via UDP (unreliable – but can be as much as three times faster than data sent via connection-oriented TCP).
- A multicast group is specified by a class D IP address and by a standard UDP port number.
- Class D IP addresses are in the range 224.0.0.0 to 239.255.255.255, inclusive.

Multicast Group

- A set of Internet hosts that share a multicast address.
- 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).

External data representation and marshalling

- One of the following methods can be used to enable any two computers to exchange binary data values:
  - The values are converted to an agreed external format before transmission and converted to the local form on receipt;
    - if the two computers are known to be the same type, the conversion to external format can be omitted.
  - The values are transmitted in the sender's format, together with an indication of the format used, and the recipient converts the values if necessary.

- An agreed standard for the representation of data structures and primitive values is called an *external data representation*.

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 and marshalling

Approaches:

- CORBA's common data representation
  - which is concerned with an external representation for the structured and primitive types that can be passed as the arguments and results of remote method invocations in CORBA. It can be used by a variety of programming languages

- Java's object serialization
  - which is concerned with the flattening and external 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)
  - which 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 – but it is now also used to represent the data sent in messages exchanged by clients and servers in web services.

External data representation and marshalling

- Two other techniques for external data representation:
  - Google uses an approach called *protocol buffers* to capture representations of both stored and transmitted data
  - JSON (JavaScript Object Notation) is an approach to external data representation [www.json.org].
- Protocol buffers and JSON represent a step towards more lightweight approaches to data representation (when compared, for example, to XML).

Network Address Translation (NAT)

- When a host on the internal network sends a packet the router saves the source IP address and port on slot in address translation table
- Router replaces source address in packet with routers IP address and source port with virtual port number, indexing translation table slot containing sending computer's address information
- Packet is sent
- When packet is received back, router uses destination port number to access translation table slot, replaces the address and port, and forwards the packet

MobileIP

- All users assumed to have home location that never changes
- Simple access to services does not require a mobile computer to retain a single address, and it may acquire a new IP address at each site: that is the purpose of the Dynamic Host Configuration Protocol (DHCP) – enables a newly connected computer to acquire a temporary address and the address of local resources.

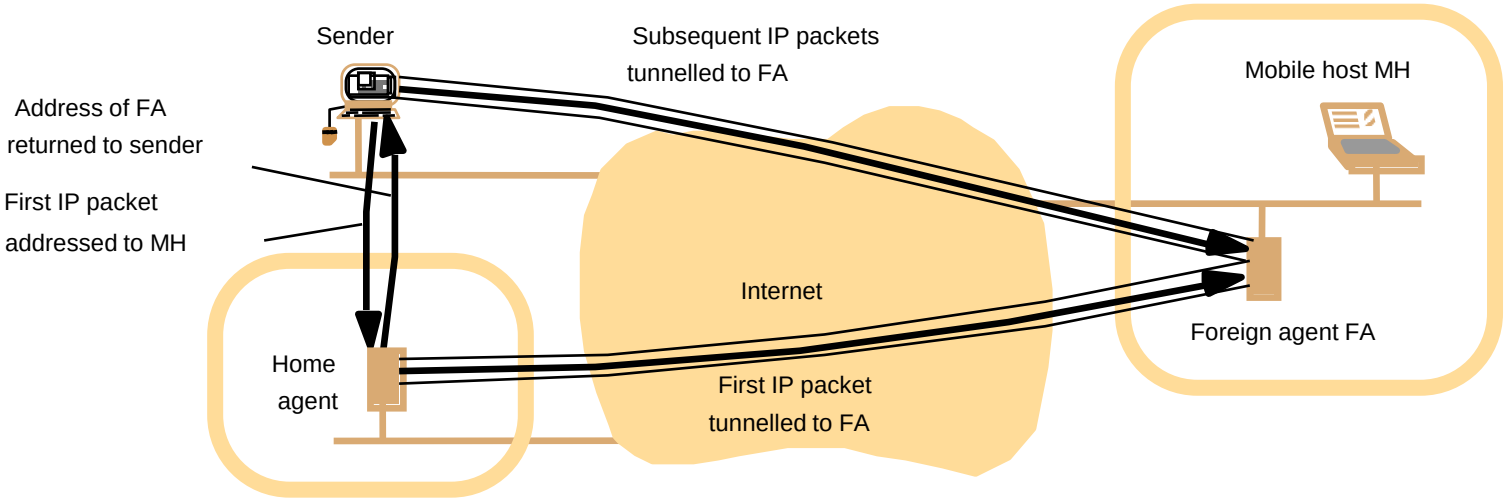
- World is divided into small geographical units, called areas
- Typically an area is a LAN or a WAN
- Each area has at least one *foreign agent* (FA)
- Keeps track of all visiting hosts
- Each area has at least one *home agent* (HA)
- Keeps track of all local hosts who are currently visiting elsewhere

MobileIP: Registration

- New hosts must register with foreign agent
- Periodically each foreign agent broadcasts a packet announcing its existence and its address
- Mobile host must wait for one of these messages, or can alternatively broadcast onto network asking for location of foreign agent
- Mobile host registers with the foreign agent giving its home address, data link layer address and security information
- The FA allocates a *care-of address*' to it – a new, temporary IP address on the local subnet.
  - The FA then contacts the HA giving it the mobile host's home IP address and the care-of' address that has been allocated to it.

MobileIP - Routing

- When a packet is sent to a mobile user:
  - It is routed to the home LAN
  - Intercepted by the home agent
  - Home agent looks up the mobile users new, temporary location and finds address of foreign agent handling the mobile user
- *Home agent* then
  - Encapsulates the packet in the payload of the outer packet and sends it to the foreign agent (Tunnelling)
- After this is received by the *foreign agent* it is decapsulated and sent on to the mobile user as a data link frame.
- HA tells the sender to henceforth send packets to the mobile host be encapsulating them in payload of packets explicitly addressed to the foreign agent.



Design requirements for distributed architectures

- Motivating factors and relevance for distributed objects and processes:
  - Need to share resources – data/code
  - Availability of cheaper microprocessors
  - Existence and advances of communication infrastructure, network protocols, concurrency control techniques

Design requirements for distributed architectures (I)

- Performance
  - – *Responsiveness*: timeliness of response to requests, hampered by delays in computing, communication, layers of middleware, bandwidth.
  - – *Throughput*: determined by server/client speeds and data transfer rates.
  - – *Load balancing*: concurrent but without undue competition for resources – equal distribution of tasks and resources with possible inter-process data/code migration.

Design requirements for distributed architectures (II)

- *Quality of service* (QoS)

  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 (III)

- 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.

Design requirements for distributed architectures (IV)

- Dependability: correctness, security and fault tolerance.
- Dependability issues
  - As related to correctness, security, and fault tolerance (maturing fields)
  - Fault tolerance: issues of fail-safe, graceful degradation, partial availability due to redundancy (of both hw and sw), dynamic reconfigurability
  - Architectural redundancy: multiple computers/storage, replicated data/programs/processes, alternative communication paths/switches, e.g., air traffic control systems, with multiple retransmission with Ack protocols
  - Security: models and protocols that ensure protection of sensitive data/programs/processes from attack/abuse, e.g., securing patient records on a server

Java Multithreading

Thread basics

- A **thread** is a single sequential flow of execution that runs through a program.
- Unlike a process, a thread does not have a separate allocation of memory, but shares memory with other threads created by the same application.
- You can have more than one thread running at the same time inside a single program, which means it shares memory with other threads created by the same application.
- *Multithreading*: using one or more extra threads in order to ‘offload’ processing tasks onto them. These threads need to be programmed explicitly.

Thread States

A thread can be in any of the following states:

- NEW
  - A thread that has not yet started is in this state.
- RUNNABLE
  - A thread executing in the Java virtual machine is in this state.
- BLOCKED
  - A thread that is blocked waiting for a monitor lock is in this state.
- WAITING
  - A thread that is waiting indefinitely for another thread to perform a particular action is in this state.
- TIMED_WAITING
  - A thread that is waiting for another thread to perform an action for up to a specified waiting time is in this state.
- TERMINATED
  - A thread that has exited is in this state.

Thread Priorities

- Each thread is assigned a priority, ranging from MIN_PRIORITY (equals 1) to MAX_PRIORITY (which is 10). A thread inherits its priority from the thread that spawned it.
- The scheduling algorithm always lets the highest priority runnable thread run. If at any time a thread with a higher priority than all other runnable threads becomes runnable, the runtime system schedules it for execution.
- The new higher priority thread is said to preempt the other threads.
- A lower priority thread can only run when all higher priority threads are non-runnable.

Thread synchronization

- Many threads - need to synchronize their activities.
- Need to prevent concurrent access to data structures in the program that are shared among the threads.
- Java provides mechanisms for synchronization and mutual exclusion (allowing only one thread to run through critical code sections in the program).

Monitors

- A monitor is associated with a specific data item and functions as a lock on that data.
- When a thread holds the monitor for some data item, other threads are locked out and cannot inspect or modify the data.
- A thread can acquire the monitor if no other thread currently owns it, and it can release it at will.
- A thread can re-acquire the monitor if it already owns it.

- A locking is achieved by using the Java keyword `synchronized`

Java Object Serialisation

- In Java RMI, both objects and primitive data values may be passed as arguments and results of method invocations
- Serialisation in Java refers to flattening
- All referenced objects are serialised together
- It suffices to state that a class implements *Serializable* to allow its instances to be serialised when required
- The *Serializable* interface has not methods and is provided in the *java.io* package

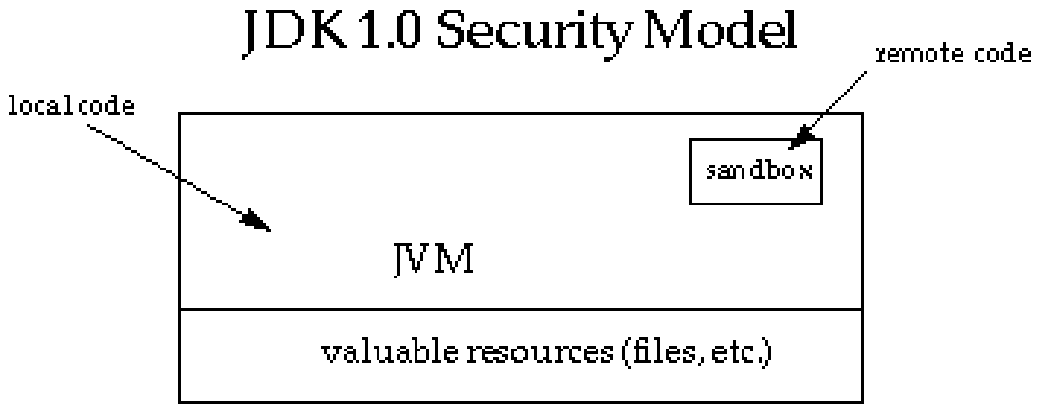
Serialization

- **Serialization**: objects of any class that implements the *java.io.Serializable* interface may be transferred to and from disc files as whole objects, with no need for decomposition of those objects.

- The *Serializable* interface is a marker to tell Java that objects of this class may be transferred on an object stream to and from files.
- Implementation of the *Serializable* interface involves no implementation of methods. The programmer only has to ensure that the class to be used includes the declaration *implements Serializable* in its header line.

Java 2 Platform Security

- Original Sandbox Model
  - Code is executed in the Java Virtual Machine (JVM).
    - JVM simulates execution of Java Byte Code.
  - Sandbox model allows code to run in a very restricted environment.
  - But, local code has full access to valuable system resources.



Sandbox?

- The default sandbox is made of three interrelated parts:
  - The *Verifier* - helps ensure type safety.
  - The *Class Loader* - loads and unloads classes dynamically from the Java runtime environment.
  - The *Security Manager* - acts as a security gatekeeper guarding potentially dangerous functionality.

Evolving the Sandbox Model: Java 2 Platform Security Model

- Easily configurable security policy.
  - Allows application builders and users to configure security policies without having to program
- Easily extensible access control structure.
  - The new architecture allows typed permissions (each representing an access to a system resource) and automatic handling of all permissions (including yet-to-be-defined permissions) of the correct type.
  - No new method in the SecurityManager class needs to be created in most cases.

Secure Code & File Exchange

- The basic idea in the use of digital signatures is as follows.
  - You "sign" the document or code using one of your *private keys*, which you can generate by using `keytool` or security API methods. That is, you generate a digital signature for the document or code, using the `jarsigner` tool or API methods.
  - You send to the other person, the "receiver," the document or code and the signature.
  - You also supply the receiver with the public key corresponding to the private key used to generate the signature, if the receiver doesn't already have it.
  - The receiver uses the *public key* to verify the authenticity of the signature and the integrity of the document/code.
  - A receiver needs to ensure that the public key itself is authentic before reliably using it to check the signature's authenticity. Therefore it is more typical to supply a *certificate* containing the public key rather than just the public key itself.

The Secure Socket Layer (SSL)

- SSL is a widely-used system component that supports secure and authenticated communication.
- Key distribution and secure channels for internet commerce
  - Hybrid protocol; depends on public-key cryptography
  - Originally developed by Netscape Corporation (1994)
  - Extended and adopted as an Internet standard with the name *Transport Level Security (TLS)*
  - Provides the security in all web servers and browsers and in secure versions of Telnet, FTP and other network applications;
  - Used to secure HTTP interactions for use in Internet e-commerce and other security sensitive applications.

SSL/TLS

- Design requirements
  - Secure communication without prior negotiation or help from 3rd parties
  - Free choice of crypto algorithms by client and server
  - Communication in each direction can be authenticated, encrypted or both
- Main features:
  - Negotiable encryption and authentication algorithms:
    - algorithms negotiated during the initial handshake.
  - Bootstrapped secure communication:
    - Unencrypted communication is used for the initial exchanges, then public key-key cryptography and finally switching to secret-key cryptography once a shared key has been established.
- Protocol prefix *https*: in URLs initiates the establishment of an TLS secure channel between a browser and a web server.

SSL Layers

- Two layers:
  - *SSL Record Protocol layer* - which implements a secure channel, encrypting and authenticating messages transmitted through any connection-oriented protocol;
  - *handshake layer* – containing the SSL handshake protocol and two other related protocols that establish and maintain an SSL session (that is, a secure channel).