Revision 1

CMPU4021 Distributed Systems

Introduction

What is a Distributed System (DS)?

- Multiple computers working together on one task
- Computers are connected by a network, and exchange information
- 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

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 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 – Heterogeneity I

- Middleware
  - 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!

DS Challenges – Openness I

- *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

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

Architectural models

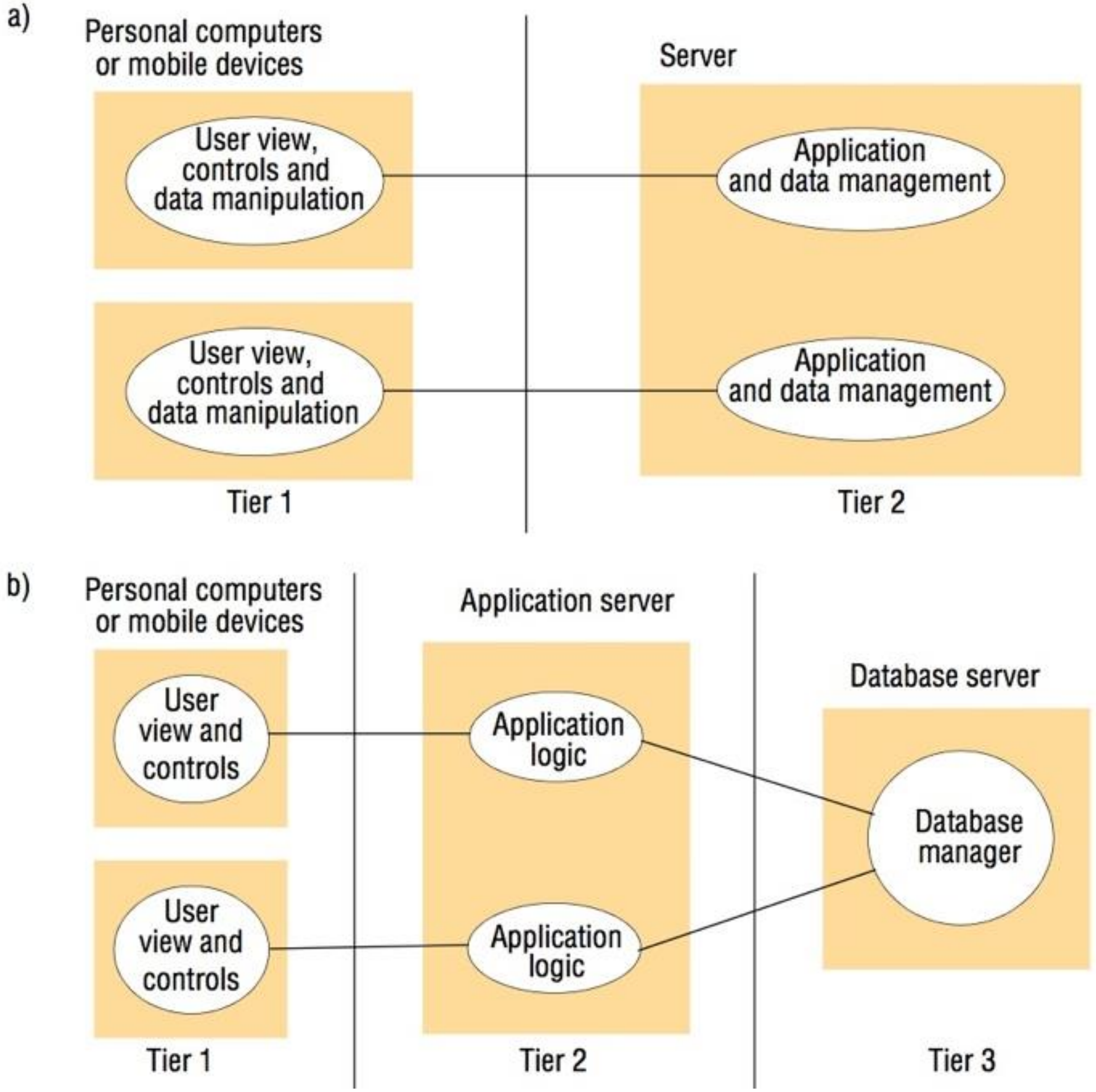
- *Architecture* – structure which defines the placement of system components, to guarantee reliability, manageability, adaptability, and cost-effectiveness – a consistent frame of reference

- 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.

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.

Two-tier and three-tier architectures


a)
Personal computers or mobile devices
Server
User view, controls and data manipulation
Application and data management
User view, controls and data manipulation
Application and data management
Tier 1
Tier 2
b)
Personal computers or mobile devices
Application server
Database server
User view and controls
Application logic
Database manager
User view and controls
Application logic
Tier 1
Tier 2
Tier 3

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

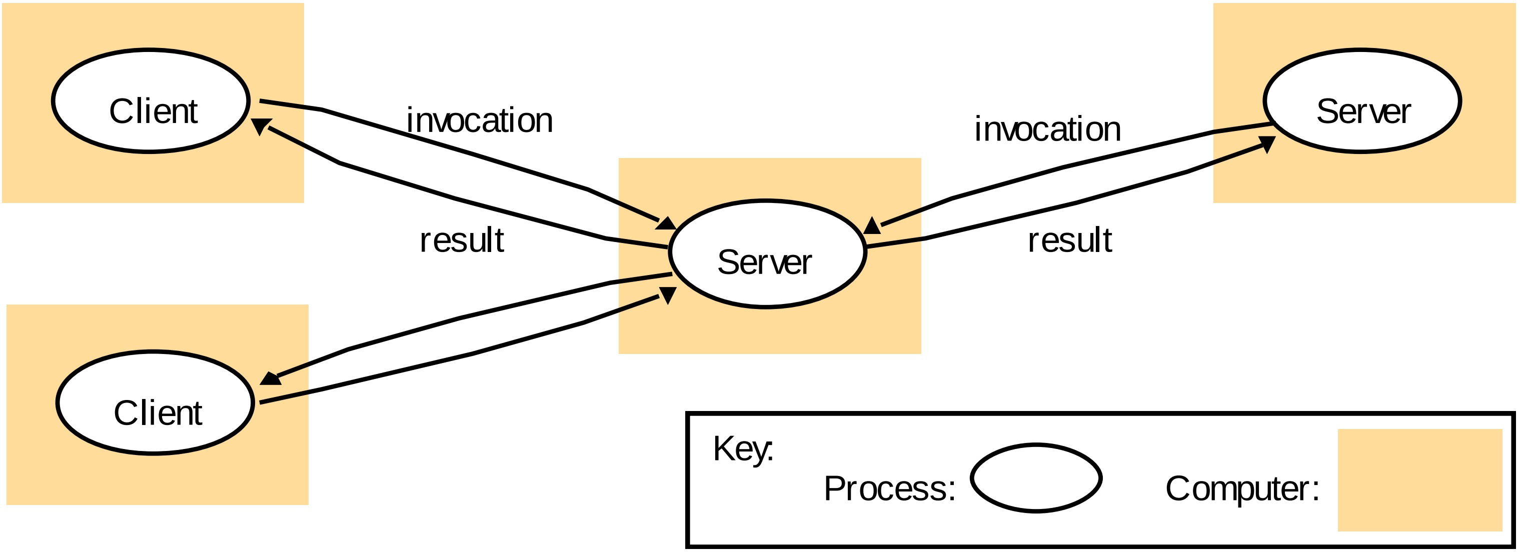
- Architectural design has a major impact on performance, reliability, and security. In a distributed system, processes with well-defined responsibilities interact with each other to perform a *useful* activity.

- 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

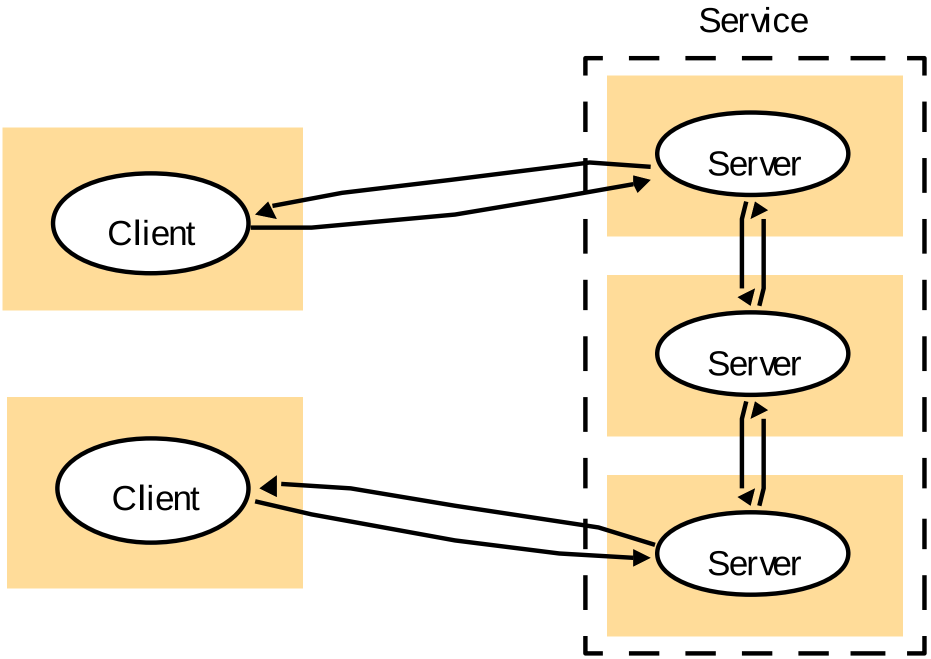
- Widely used, servers/clients on different computers provide services to clients/servers on different computers via request/reply messaging.
- Servers could also become clients in some services, and vice-versa, e.g., for web servers and web pages retrievals, DNS resolution, search engine-servers and 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.

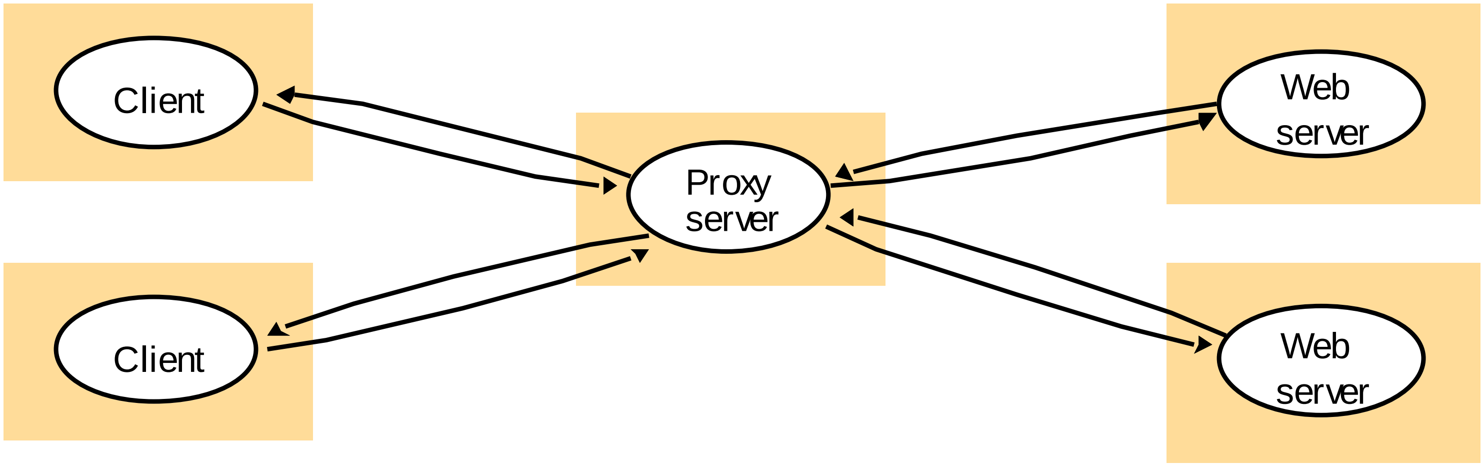


The Proxy Servers and Caches model

- A *cache* is a store of recently used data objects that is closer than the objects themselves.

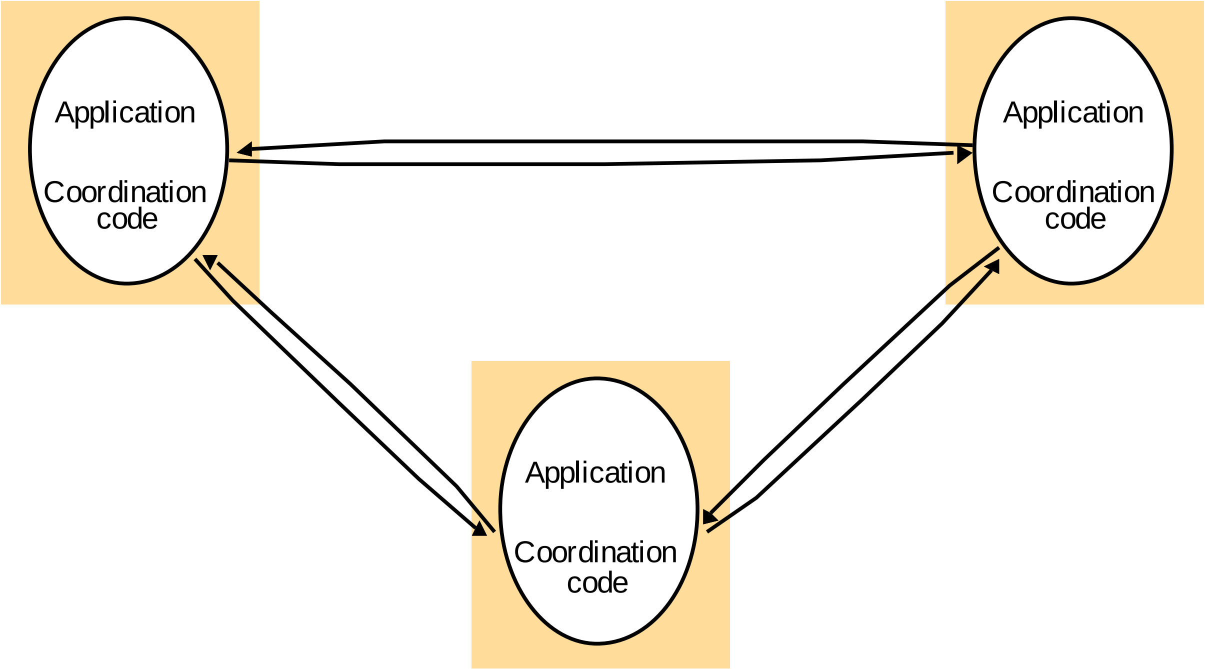
- Caching frequently used: objects/data/code, which can be collocated at all clients, or located at a single/multiple ‘proxy’ server(s) and accessed/shared by all clients.

- 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.

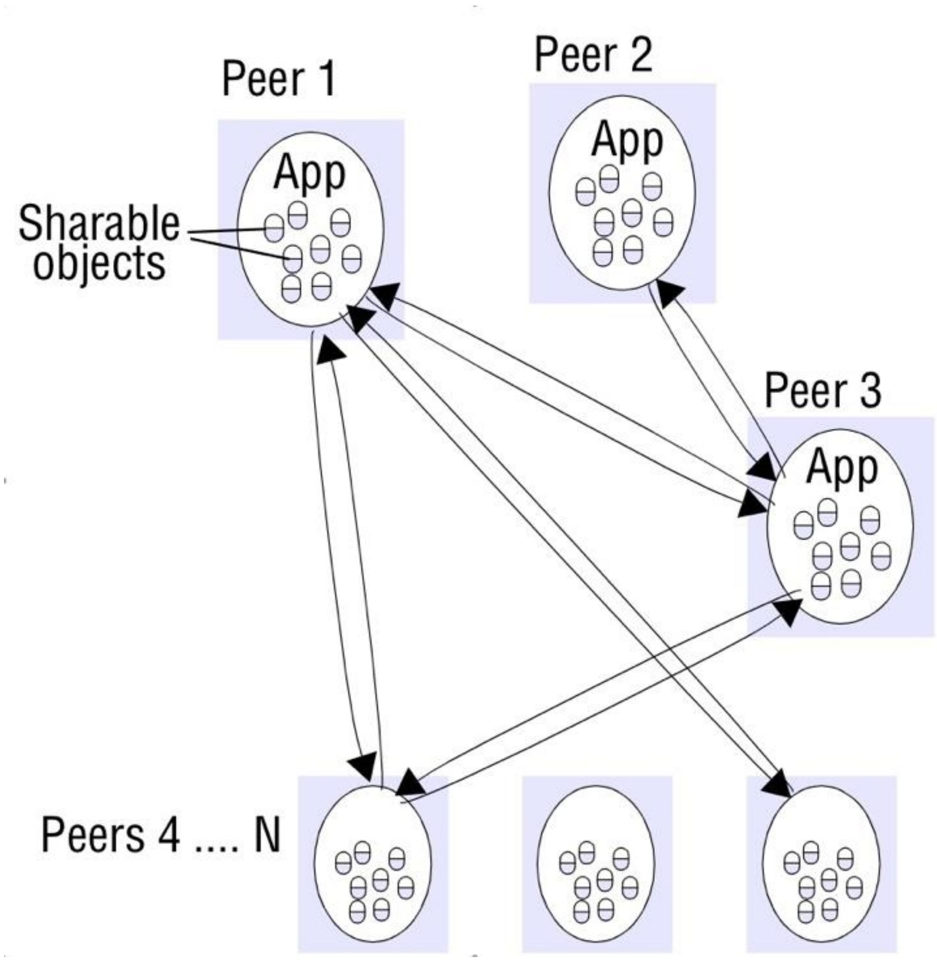
- They may differ in the resources that they contribute, but all the nodes in a peer-to-peer system have the same functional capabilities and responsibilities.

- 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

- 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.

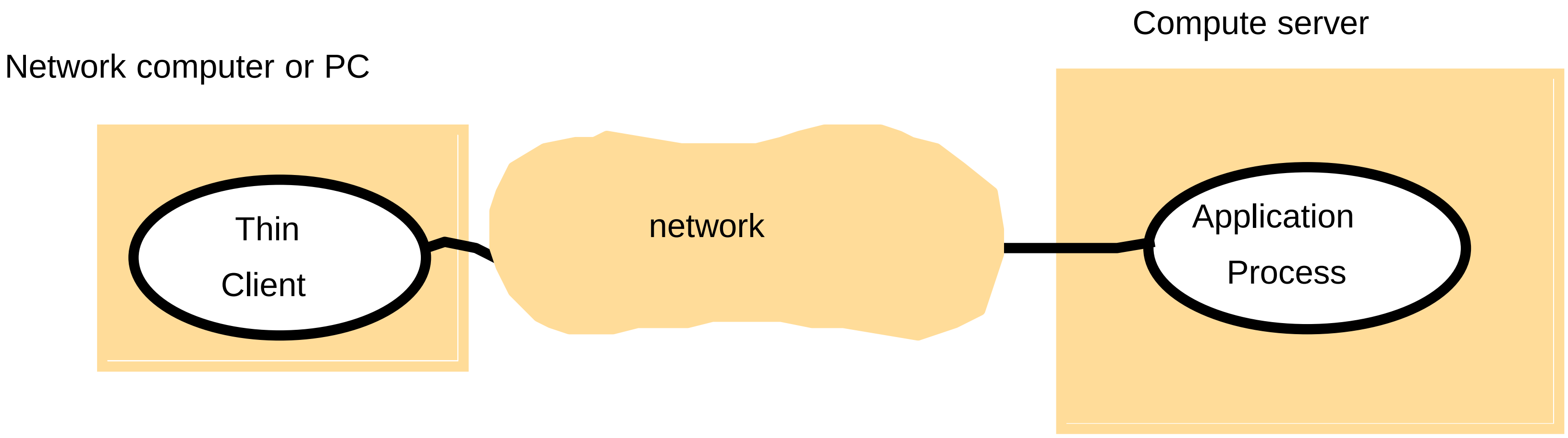
- Applications are then run locally and files/OS are managed by server; this way, a client-user can migrate from computer to computer and still access the server, and all updates are done at the server side.

- 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

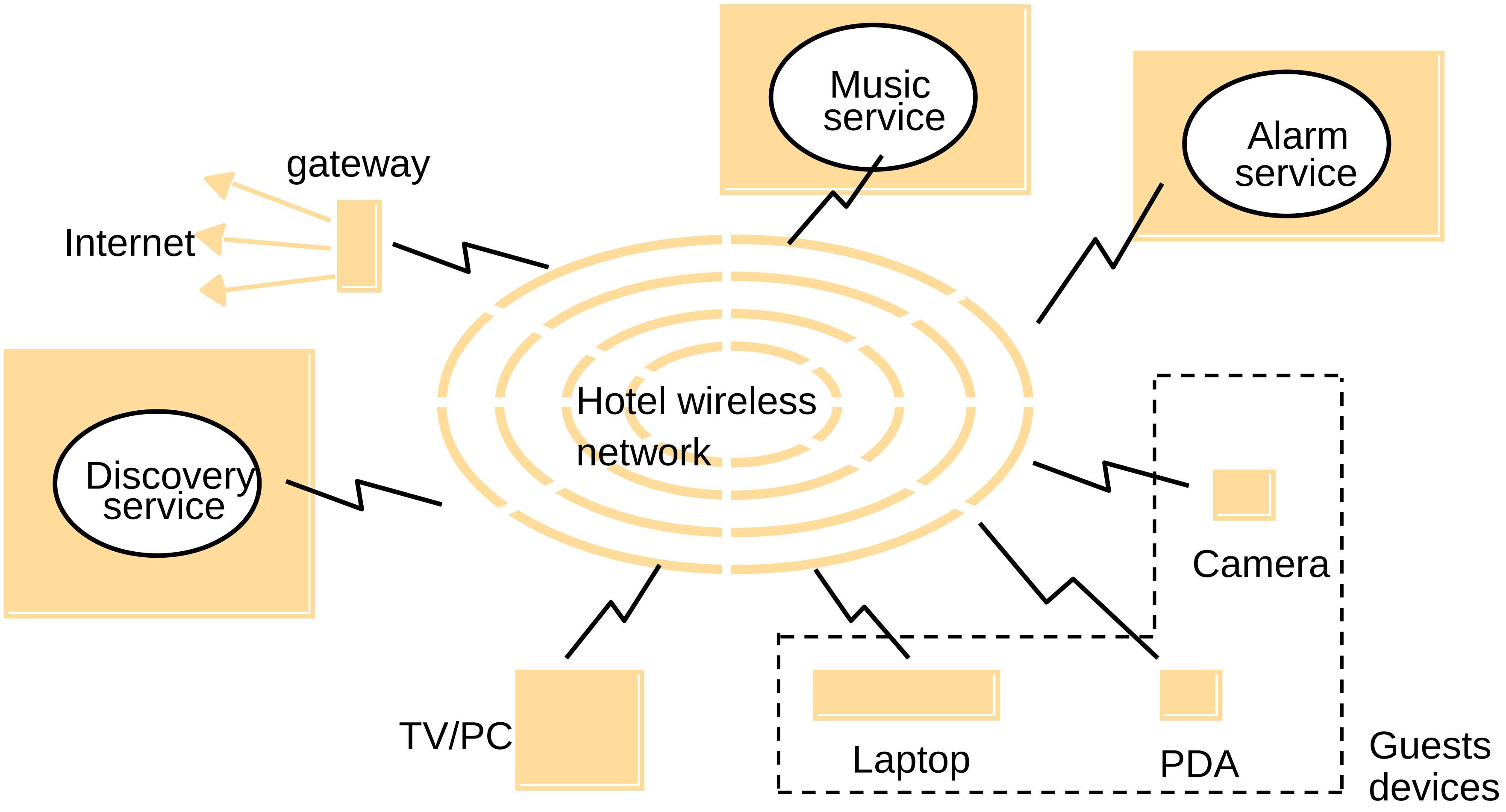
- Drawbacks:
  - delays experienced by users can be increased to unacceptable levels by the need to transfer images and vector information between the thin client and the application process,
    - due to both network and operating system latencies.

- 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.

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.

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 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.

IP Multicast

- Multicast addresses may be permanent or temporary.
- Permanent groups exist even when there are no members – their addresses are assigned by IANA
- The remainder of the multicast addresses are available for use by temporary groups, which must be created before use and cease to exist when all the members have left.
- When a temporary group is created, it requires a free multicast address to avoid accidental participation in an existing group.

Multicasting

- Broader than *unicast (*one sender and one receiver, point-to-point communication) but narrower and more targeted than broadcast communication.

- Sends data from one host to many different hosts, but not to everyone;
  - the data only goes to clients that have expressed an interest by joining a particular multicast group.

- Used for ‘public meetings’ on the Internet
  - a multicast socket sends a copy of the data to a location close to the parties that have declared an interest in the data;
  - the data is duplicated only when it reaches the local network serving the interested clients;
  - the data crosses the Internet only once.)

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

- 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*.

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.

Serializable Objects

- The ability to store and retrieve Java objects is essential to building all but the most transient applications.

- The key to storing and retrieving objects in a serialized form is representing the state of objects sufficient to reconstruct the object(s).

- To serialize an object means to convert its state to a byte stream so that the byte stream can be reverted back into a copy of the object.
- A Java object is serializable if its class or any of its superclasses implements either the `java.io.Serializable` interface or its subinterface, `java.io.Externalizable`. Deserialization is the process of converting the serialized form of an object back into a copy of the object.

- For example:
  - `java.awt.Button` class implements the `Serializable` interface, so you can serialize a `java.awt.Button` object and store that serialized state in a file.
  - Later, you can read back the serialized state and deserialize into a `java.awt.Button` object.

Serializing data to XML

- Pros
  - XML is human readable
  - There are binding libraries for lots of languages.
  - A good choice if you want to share data with other applications/projects.
- Cons
  - Space intensive
  - Encoding/decoding it can impose a huge performance addition on applications.
  - Navigating an XML DOM tree is considerably more complicated than navigating simple fields in a class.

External data representation: other techniques

- Two other techniques for external data representation:
  - Google uses an approach called *protocol buffers (aka* protobuf) 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
  - more lightweight approaches to data representation
    - when compared, for example, to XML).

JSON (JavaScript Object Notation)

- JSON is a lightweight data-interchange format based on a subset of the JavaScript Programming Language Standard ECMA-262

- Derived from JavaScript that is used in web services and other connected applications.

- Browsers can parse JSON into JavaScript objects natively.

- On the server, JSON needs to be parsed and generated using JSON APIs.

Protocol Buffers (*protobuf*)

- A mechanism for serializing structured data.
- Similar to XML
  - smaller, faster, and simpler

- Uses binary format
  - rather than text format of XML and JSON

- Is in fact an IDL (Interface Definition Language)

Protocol Buffers

- Properties:
  - Efficient, binary serialization
  - Support protocol evolution
    - Can add new parameters
    - Order in which parameters are specified is not important
    - Skip non-essential parameters
  - Supports types, which give you compile-time errors
  - Supports quite complex structures

- Usage:
  - It is a binary encoding format that allows you to specify a *schema* for your data
  - Protocol buffers are used for other things, e.g., serializing data to non-relational databases – their backward-compatible feature make them suitable for long-term storage formats

- Google uses them

Concurrency

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

- 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*

- *`java.util.concurrent`* ***packages***

Processes

- A process has a self-contained execution environment

- Has a complete, private set of basic run-time resources

- Each process has its own memory space

- To facilitate communication between processes, most operating systems support *Inter Process Communication* (IPC) resources
  - such as pipes and sockets

- IPC is used not just for communication between processes on the same system, but processes on different systems

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.

Threads

- Threads share the process's resources, including memory and open files.
  - This makes for efficient, but potentially problematic, communication

- Multithreading
  - 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.

- Every application has at least one thread (started with `main()`) — or several, if you count "system" threads that do things like memory management and signal handling.

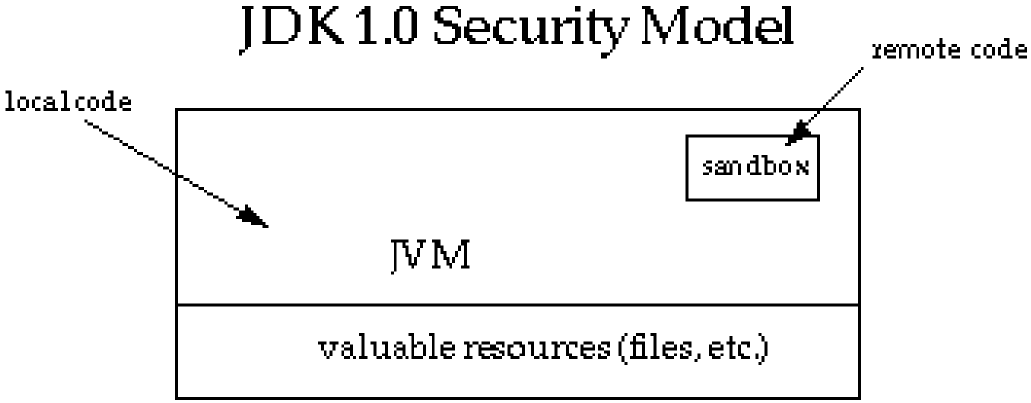
Thread States

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.

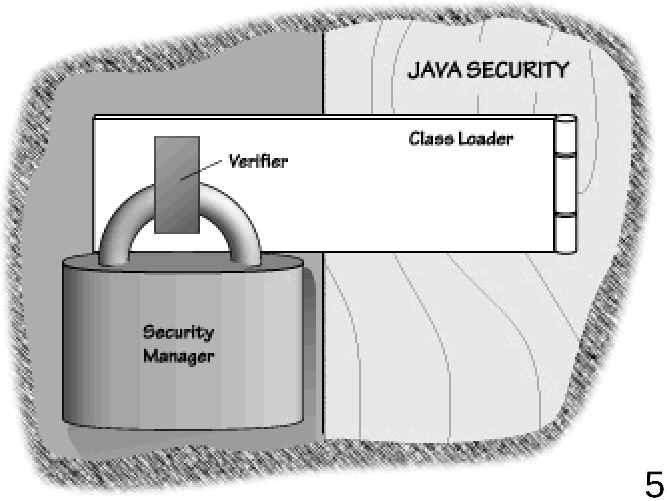
Java 2 Platform Security

- 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.



Java 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.

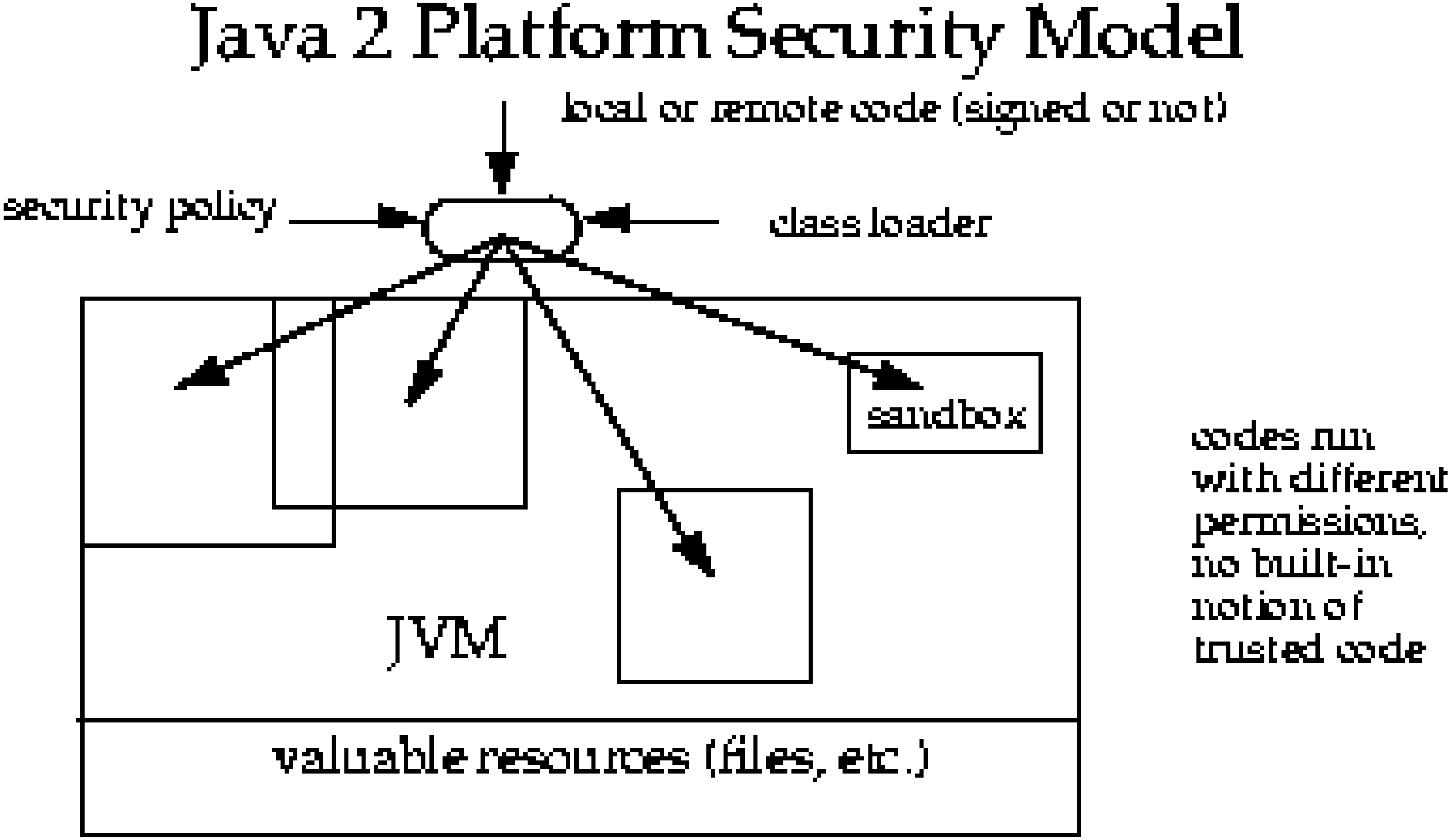
Java 2 Platform Security Model

- Extension of security checks to all Java programs, including applications as well as applets.

- There is no longer a built-in concept that all local code is trusted.

- Local code (e.g., non-system code, application packages installed on the local file system)
  - is subjected to the same security control as applets, although it is possible, if desired, to declare that the policy on local code (or remote code) be the most liberal, thus enabling such code to effectively run as totally trusted.

Java 2 Platform Security

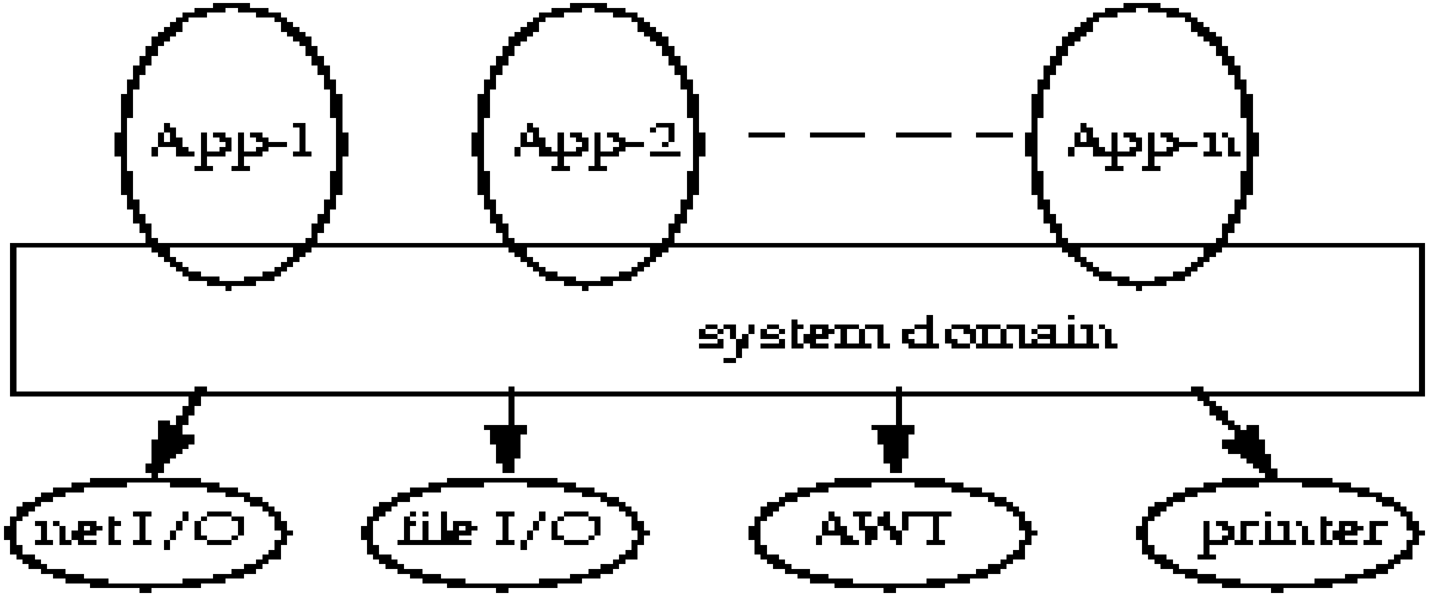

Java 2 Platform Security Model
local or remote code (signed or not)
security policy
class loader
sandbox
codes run with different permissions, no built-in notion of trusted code
JVM
valuable resources (files, etc.)

Java 2 Platform Security: Protection Domains

- Protection Domains
  - Set of objects that are currently directly accessible by a principal.
  - Principal is an entity in the computer system to which permissions are granted.
  - Serves to group and to isolate between units of protection.
  - Protection domains are either system domains or application domains.

Java 2 Platform Security: Protection Domains

- Protection domains generally fall into two distinct categories:
  - system domain
  - application domain.
- It is important that all protected external resources, such as the file system, the networking facility, and the screen and keyboard, be accessible only via system domains.



INDIRECT COMMUNICATION

Indirect Communication

- The essence of indirect communication is to communicate through an intermediary

- No direct coupling between the sender and the one or more receivers.

Indirect Communication Usage

- In distributed systems where change is anticipated
  - in mobile environments where users may rapidly connect to and disconnect from the global network – and must be managed to provide more dependable services.

- For event dissemination in distributed systems where the receivers may be unknown and liable to change
  - E.g. in managing event feeds in financial systems

Indirect communication techniques

1. Group communication

   - in which communication is via a group abstraction with the sender unaware of the identity of the recipients

2. Publish-subscribe systems

   - a family of approaches that all share the common characteristic of disseminating events to multiple recipients through an intermediary

3. Message queue systems

   - messages are directed to the familiar abstraction of a queue with receivers extracting messages from such queues

4. Shared memory–based approaches

   - distributed shared memory and tuple space approaches, which present an abstraction of a global shared memory to programmers.

GROUP COMMUNICATION

Group Communication

- A message is sent to a group and then this message is delivered to all members of the group.

- The sender is not aware of the identities of the receivers.

- Represents an abstraction over multicast communication

- May be implemented over
  - IP multicast; or
  - An equivalent *overlay network* adding significant extra value in terms of
    - managing group membership,
    - detecting failures and
    - providing reliability and ordering guarantees.

Group Communication: Key areas of application

- An important building block for reliable distributed systems
- The reliable dissemination of information to potentially large numbers of clients,
  - E.g. in the financial industry, where institutions require accurate and up-to date access to a wide variety of information sources;
- Support for collaborative applications, where events must be disseminated to multiple users to preserve a common user view
  - E.g. in multiuser games;
- Support for a range of fault-tolerance strategies, including the consistent update of replicated data or the implementation of highly available (replicated) servers;
- Support for system monitoring and management, including for example load balancing strategies.

Group Communication: Programming Model

- The central concept is that of a group with associated group membership
  - processes may join or leave the group.

- Processes can then send a message to this group and have it propagated to all members of the group with certain guarantees in terms of reliability and ordering.

- Thus, group communication implements multicast communication
  - in which a message is sent to all the members of the group by a single operation.

- A process issues only one multicast operation to send a message to each of a group of processes

Group Communication: Other Key Distinctions

- A wide range of group communication services has been developed, and they vary in the assumptions they make:
  - Closed and open groups
  - Overlapping and non-overlapping groups
  - Synchronous and asynchronous systems

Group Communication: Other Key Distinctions

- A wide range of group communication services has been developed, and they vary in the assumptions they make:
  - Closed and open groups
  - Overlapping and non-overlapping groups
  - Synchronous and asynchronous systems

PUBLISH-SUBSCRIBE SYSTEMS

Publish-Subscribe Systems

- Publish-Subscribe Systems
- Also known as
  - as *distributed event-based systems*

- A publish-subscribe system is a system where
  - *publishers* publish structured events to an event service and
  - *subscribers* express interest in particular events through
  - *subscriptions* which can be arbitrary patterns over the structured events.

- For example, a subscriber could express an interest in all events related to a book, such as the availability of a new edition or updates to the related web site.

- The task of the publish subscribe system is to match subscriptions against published events and ensure the correct delivery of *event notifications*.

- A given event will be delivered to potentially many subscribers, and hence publish-subscribe is fundamentally a one-to-many communications paradigm.

Publish-subscribe systems

- Used in a wide variety of application domains, particularly those related to the large-scale dissemination of events.

- Examples:
  - financial information systems;
  - other areas with live feeds of real-time data (including RSS feeds);
  - support for cooperative working, where a number of participants need to be informed of events of shared interest;
  - support for ubiquitous computing, including the management of events emanating from the ubiquitous infrastructure (for example, location events)
  - a broad set of monitoring applications, including network monitoring in the Internet.

- Publish-subscribe is also a key component of Google's infrastructure, including for example the dissemination of events related to advertisements, such as 'ad clicks', to interested parties.

Publish-subscribe systems: The programming model

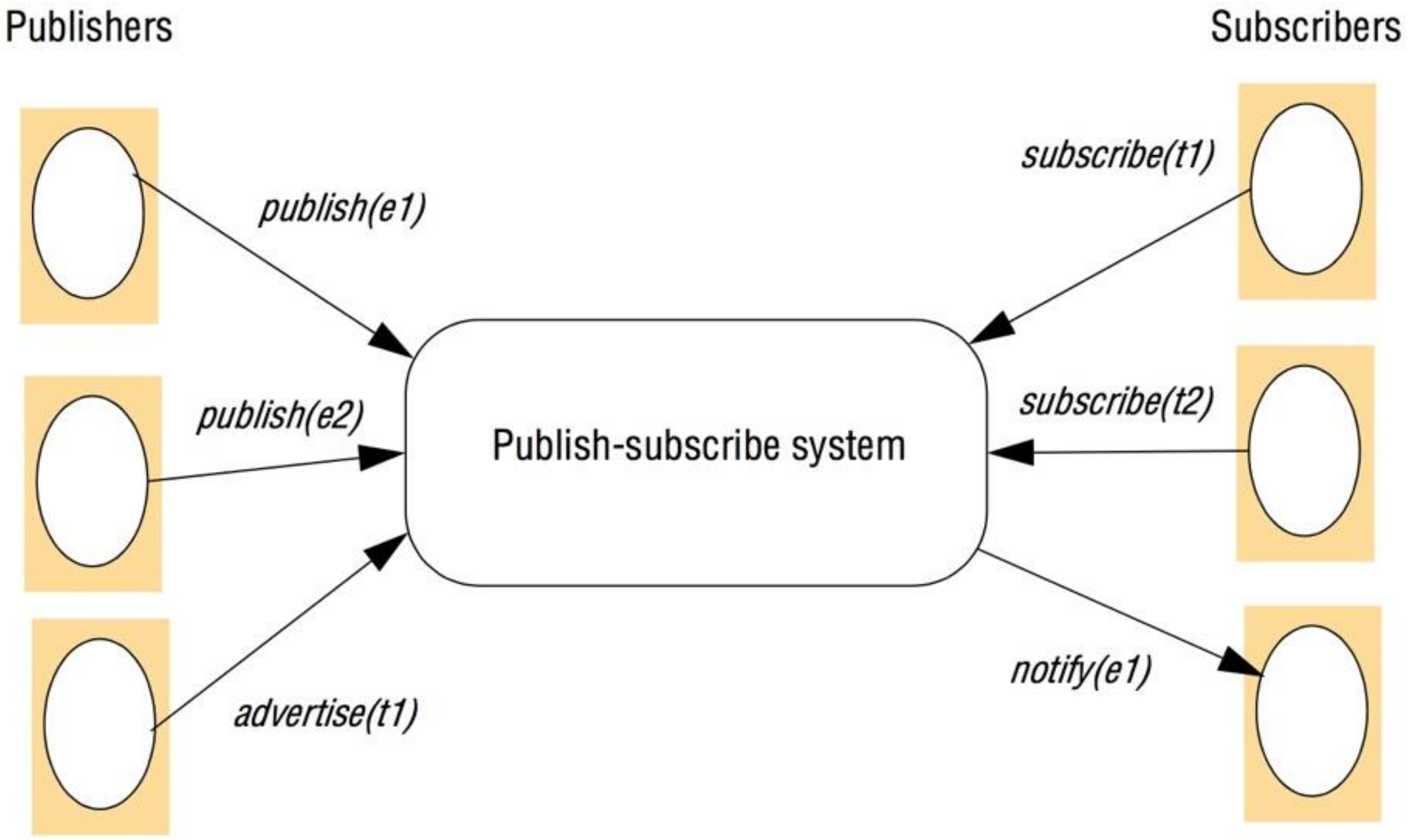
- Publishers disseminate an event *e* through a *publish(e)* operation and subscribers express an interest in a set of events through subscriptions.

- In particular, they achieve this through a *subscribe(f)* operation where *f* refers to a filter
  - that is, a pattern defined over the set of all possible events.
  - The expressiveness of filters (and hence of subscriptions) is determined by the subscription model

- Subscribers can later revoke this interest through a corresponding *unsubscribe(f)* operation.

- When events arrive at a subscriber, the events are delivered using a *notify(e)* operation.

The publish-subscribe paradigm



The participants in distributed event notification

- The main component is an event service that maintains a database of published events and of subscribers’ interests.
- Events at an object of interest are published at the event service.
- Subscribers inform the event service about the types of events they are interested in. When an event occurs at an object of interest, a notification is sent to the subscribers to that type of event.

The roles of the participating objects

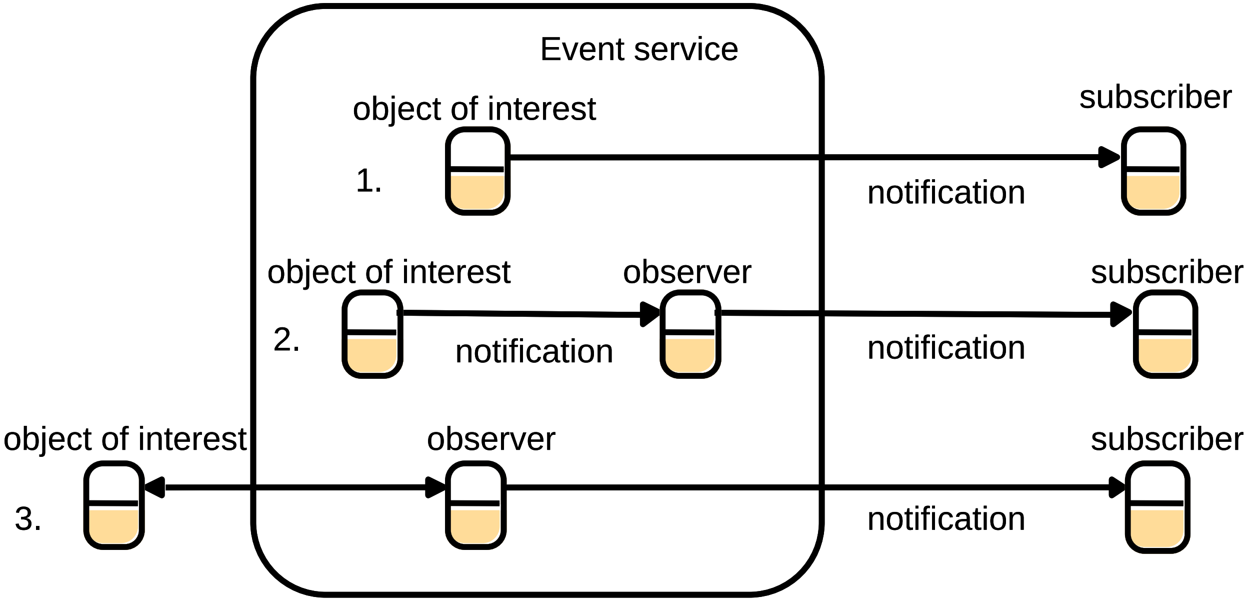
- *The object of interest*:
  - Experiences changes of state, as a result of its operations being invoked.
    - E.g. events such as a person wearing an active badge entering a room, in which case the room is the object of interest and the operation consists of adding information about the new person to its records of who is in the room.
- *Event*
  - occurs at an object of interest as the result of the completion of a method execution.
- *Notification*:
  - An object that contains information about an event; it contains the type of the event and its attributes such as the identity of the object of interest, the method invoked, the time of occurrence or a sequence number.

The roles of the participating objects (cont)

- *Subscriber*:
  - An object that has subscribed to some type of events in another object. It receives notifications about such events.
- *Observer objects*:
  - Purpose – to decouple an object of interest from its subscribers. An object of interest can have many different subscribers with different interests.
- Publisher:
  - An object that declares that it will generate notifications of particular types of event; it may be an object of interest or an observer.

Architecture for distributed event notification

1. An object of interest inside the event service without an observer. It sends notifications directly to the subscribers.
2. An object of interest inside the event service with an observer. The object of interest sends notifications via the observer to the subscribers.
3. An object of interest outside the event service. In this case, an observer queries the object of interest in order to discover when events occur. The observer sends notifications to the subscribers.



Message queue systems

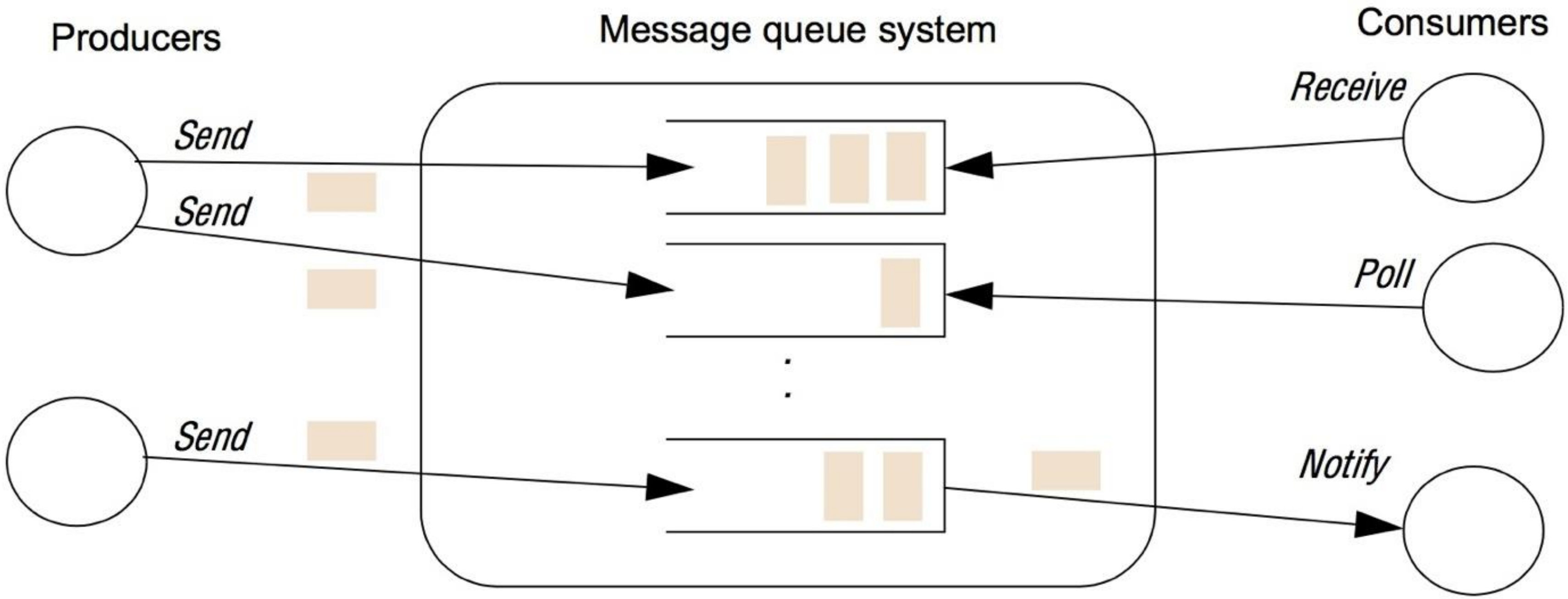
- They are point-to-point
  - the sender places the message into a queue, and it is then removed by a single process. Message queues are also referred to as Message-Oriented Middleware.

- A major class of commercial middleware with key implementations
  - IBM's WebSphere MQ,
  - Microsoft's MSMQ
  - Oracle's Streams Advanced Queuing (AQ).

- The main use of such products is to achieve Enterprise Application Integration (EAI)
  - that is, integration between applications within a given enterprise – a goal that is achieved by the inherent loose coupling of message queues.

- They are also extensively used as the basis for commercial transaction processing systems because of their intrinsic support for transactions.

The message queue paradigm



Shared memory approaches: DSM

- Distributed shared memory (DSM) is an abstraction used for sharing data between computers that do not share physical memory.

- Processes access DSM by reads and updates to what appears to be ordinary memory within their address space.

- However, an underlying runtime system ensures transparently that processes executing at different computers observe the updates made by one another

- It is as though the processes access a single shared memory, but in fact the physical memory is distributed
- Examples:
  - Apollo Domain file system