Models of Distributed Systems
The aims of this lecture are to introduce:
- Architectural models;
- Fundamental models.

For further information and background, read Pages 29-64

Architecture Models

- **Distributed Systems are foremost highly complex software systems**
  Examples:
  - **Nortel Networks DMS100 switch** - it is the biggest seller of a line of Digital Multiplex System (DMS) telephone exchange switches manufactured by Nortel Networks: 2530 million lines of code, 3000 software developers, > 20 years life cycle to date.
  - **Motorola**: 20% of engineers produce hardware, 80% produce software

- **An architecture model** – defines the way in which the components of systems interact with one another and the way in which they are mapped onto an underlying network of computers.
  - It simplifies and abstracts the functions of the individual components of a distributed system (e.g. client/server/peer processes)
  - It considers the placement of components across a network of computers as well as the interrelationships between the components.
• **Architectural paradigms pertinent to distributed systems**
  - Layers and client/server

**Layers - Basic Idea**

- Breaking up the complexity of systems by designing them through **layers** and **services**
  - **layer**: group of closely related and highly coherent functionalities
  - **service**: functionality provided to a superior layer

**Examples of layered architectures**

- **operating systems**
  (kernel, other services)

- **compute network**
  (OSI layers, Cisco network layers)

---

**Network Architecture**

**Example - Open Systems Interconnection Basic Reference Model (OSI-BRM)**

PDU: Protocol Data Unit
**Software and hardware service layers in DS**

- **Platform**: Hardware and operating system
  - Windows NT / Pentium processor
  - Solaris / SPARC processor
  - Solaris/ x86
  - MacOS/ PowerPC
  - Linux/ x86

**Middleware** – achieve:

- transparency of heterogeneity at platform level
- communication and resource sharing

**Examples:**

- **OMG – CORBA** (Common Object Request Broker Architecture):
  - OMG’s modeling standards, including the Unified Modeling Language (UML) and Model Driven Architecture (MDA), enable powerful visual design, execution and maintenance of software and other processes, including IT Systems Modeling and Business Process Management.
  - OMG’s middleware standards and profiles are based on the Common Object Request Broker Architecture (CORBA) and support a wide variety of industries.

- **Microsoft – DCOM** (Microsoft Distributed Component Object Model):
  - supports communication among objects on different computers—on a LAN, a WAN, or even the Internet.
Middleware (Cont.)

Examples (Cont.):

- **Sun-Java RMI** (Java Remote Method Invocation)
  - enables the programmer to create distributed Java technology-based to Java technology-based applications, in which the methods of remote Java objects can be invoked from other Java virtual machines, possibly on different hosts.

- **ISO - RM-ODP** (Reference Model of Open Distributed Processing)
  - provides the concepts and rules of distributed processing to ensure openness between interacting distributed application components.
  - **Openness** is a combination of characteristics: i.e. scalability, accessibility, heterogeneity, autonomy and distribution.

System Architectures - Client/Server Model

**Basic C/S Model**

- **Client**: Process wishing to access data, use resources or perform operations on a different computer.

- **Server**: Process managing data and all other shared resources amongst servers and clients, allows clients access to resource and performs computation.

- **Interaction**: invocation/result message pairs.

**Example** - http server: client (browser) requests page, server delivers page
**Variants of C/S model**
- Service provided by multiple servers

**Examples:** many commercial web services are implemented through different physical servers (e.g., amazon.com)

**Motivation:** performance reliability (e.g., cnn.com, download servers, etc.)

- Servers maintain either replicated or distributed database

**Variants of C/S model (Cont.)**

- **Proxy servers** act as an intermediary for requests from clients seeking resources from other servers.
- The proxy server evaluates the request according to its filtering rules. For example, it may filter traffic by **IP address** or **Protocol**.
Proxy servers (Cont.)

• Most proxies are a web proxy, allowing access to content on the World Wide Web. A proxy server has a large variety of potential purposes, including:
  • To keep machines behind it anonymous (mainly for security).
  • To speed up access to resources (using caching). Web proxies are commonly used to cache web pages from a web server.
  • To apply access policy to network services or content, e.g. to block undesired sites.
  • To log/audit usage, i.e. to provide company employee Internet usage reporting.
  • To scan transmitted content for malware before delivery.
  • To scan outbound content, e.g., for data leak protection.
  • To circumvent regional restrictions.

Variants of C/S model (Cont.)

- Caching
  • Proxy server maintains cache store of recently requested resources
  • Frequently used in search-engines:
### Further Variants of C/S Model

**Mobile Agents**
- Executing programs (code + data), migrating amongst processes, in different computers, carrying out an autonomous task, usually on behalf of some other process.

**Example** - They might be used to install and maintain software on the computers within the organization or compare prices from a number of vendors by visiting the site of each vendor and performing a series of database operations.

**Advantages**: Flexibility, savings in communications cost.

**Thin Clients**
- Executing windows-based user interface on local computer while application executes on computer server.

**Example** - The X Window System (or X11) is a software and network protocol that provides a graphical user interface (GUI) for networked computers. It also manages the interactive input devices such as mouse, keyboard, etc.

---

**Further Variants of C/S Model**

**Mobile Code** - Code that is sent to a client process to carry out a specific task.

**Examples**:
- **Applets** – Programs run locally, thus improving interactive response
- **Active Messages** (containing communications protocol code)

**a)** Client request results in the downloading of applet code

**b)** Client interacts with the applet
Mobile Devices and Spontaneous Networking

- for mobile computing code
  - personal digital assistants (PDAs)
  - how to connect to Internet
    - wireless LANs/MANs - wireless Personal Area Networks

Benefits

- no need for wire-line connection
- easy access to locally available services

Challenges - support for convenient connection and integration:

- **IP address** - Internet assumes device has IP address in fixed sub-network. Possible solution - dynamic allocation of IP addresses
- **Limited connectivity of devices** - unavailable when in tunnels, airplanes, etc.
- **Privacy** - ubiquity of location information
- **Security** - access to device and access rights in a dynamic, heterogeneous environment? For example, in a hotel, the hotel and guests are vulnerable to security attacks from guests and hotel employees
Mobile Devices and Spontaneous Networking (Cont.)

- **Discovery services** – accept and store details of services that are available on the network and respond to queries from clients about them. They offer two interfaces:
  - **registration service** - accept registration requests from servers, stores properties in database of currently available services
  - **lookup service** - match requested services with available servers. The result returned includes sufficient details to enable clients to select between several similar services

**Interfaces and Objects**

- **Impact of C/S architecture on software architecture**
  - Each **server process** is seen as a **single entity** with a fixed interface defining the functions that can be invoked in it
  - In O-O languages such as C++ and Java, distributed processes can be constructed in a more O-O manner
  - Objects can be encapsulated in server processes, and references to them are passed to other processes so that their **methods** can be accessed by remote invocation (via CORBA and Java RMI)
  - Encapsulation promotes information hiding - Data is accessed only by the **class methods**
  - **Encapsulation** protects data from outside objects and protects outside objects from the data
**Example of protecting objects**

- Convert from miles to kilometers

---

**Design Requirements for Distributed Architectures**

- **Performance issues** arising from the limited processing and communication capabilities of computers and networks can be categorized by the following:
  - **Responsiveness** – affected by the load and performance of the server and the network as well as by delays in all the software components involved (client and server operating systems, communication and middleware services)
    - Good response times can be achieved by systems with few software layers, and small quantities of data transferred between the client and server.
  - **Throughput** – the rate at which computational work is done. This is affected by processing speeds at clients and servers and by data transfer rates.
  - **Balancing computational loads** – enable applications and service processes to proceed concurrently without competing for the same resources and to exploit the available computational resources. For example, using the facility of the DNS domain name look up service and running applets on the client computers.
Design Requirements for Distributed Architectures

- **Quality of Service (QoS)**—affected by *non-functional properties* of a system. It applies to operating systems as well as to networks.
  - **Performance** - response times, throughput, etc.
  - **Reliability** and **security** are critical in the design of most computer systems (more on these issues later – fundamental models)
  - **Adaptability** - ability to meet changing system configuration
  - **Time-limits** - ability to meet time deadlines. For example, retrieving a film from a video server and presenting it on the user’s screen – the successive frames of video need to be displayed within some specified time limits for satisfactory results.

This implies a requirement for the system to provide guaranteed computing and communication resources that are sufficient to enable applications to complete each task on time.

**Example** – to browse the web, may have quiet good performance but when they are heavily loaded their performance deteriorates significantly – *in no way can they be said to provide QoS.*

Design Requirements for Distributed Architectures

- **Web Caching** – As mentioned earlier, both web browsers and proxy servers *cache* responses to client requests to web servers.
  - a browser or proxy can validate a cached response by checking with the original web server (using cache consistency protocol) to see whether it is still up to date. If not, the server returns a fresh response, which is cached instead of the stale response
  - browsers and proxies store the **expiry time** and **server time** together with the cached response. This enables a browser or proxy receiving future requests to calculate the age of a response and decides whether the cached response is likely to be stale or not.

- **Dependability issues** – correctness, security and **fault tolerance**
  - **fault tolerance**: system is expected to continue to function correctly in the presence of faults in hardware, software and networks
    - **reliability** is achieved through redundancy and in communication protocols it is achieved by retransmitting until an acknowledgment message has been received.
**Fundamental Interaction Models**

**Distributed System** - Multiple processes connected by communication channels
- All models of DS share the design requirements given in previous slides are primarily concerned with:
  (i) performance and reliability characteristics of process and networks
  (ii) the security of the resources in the system.

- Interaction model
  - **Latency**: delay between sending and receipt of a message
    - network access time (e.g., Ethernet retransmission delay)
    - time for first bit to travel from sender’s network interface to receiver’s network interface
    - processing time within the sending and receiving processes
  - **Throughput**: number of units (e.g., packets) delivered per time unit
  - **Bandwidth**: amount of information (e.g., bits) transmitted per time unit
  - **Delay jitter**: variation in delay between different messages of the same type (e.g., video frames in ATM networks)

- Interaction model (Cont.)
  - **Computer clocks and timing events**
    - each computer in DS has its own internal clock – current time for local processes
    - computer clocks drift from perfect time at rates differ from one another
    - An approach to correcting the times on computer clocks:
      - Using radio receivers to get time readings from GPS with an accuracy of about 1 µs
  - **Variants of the interaction model** – In a DS, it is hard to set time limits on the time taken for process execution, message delivery or clock. Two opposing extreme positions provide a pair of simple models:
    - Strong assumption of time - synchronous
    - No assumption about time - asynchronous
**Interaction model (Cont.)**

- **Synchronous distributed system**
  - time to execute each step of computation within a process has known lower and upper bounds
  - message transmitted over a channel is received within a known bounded time
  - each process has a clock whose drift rate from real time is bounded by a known value

- **Asynchronous distributed system** – has no bound on:
  - process execution times,
  - message delivery times, and
  - clock drift rate

- **Notes**
  - synchronous distributed systems are easier to handle, but determining realistic bounds can be hard or impossible in DS
  - asynchronous systems are more abstract and general - a distributed algorithm executing on one system is likely to also work on another one

---

**Events ordering** – The execution of a system can be described in terms of events and their ordering despite the lack of accurate clock

**Events ordering rules**:
1. if \( e_1 \) and \( e_2 \) happen in the same process, and \( e_2 \) happens after \( e_1 \), then \( e_1 \rightarrow e_2 \)
2. if \( e_1 \) is the sending of a message \( m \) and \( e_2 \) is the receiving of the same message \( m \), then \( e_1 \rightarrow e_2 \)
3. hence, \( \rightarrow \) describes a partial ordering relation on the set of events in the distributed system

**Example** – consider the following set of exchanges between a group of e-mail users: X, Y, Z and A on a mailing list:

1. User X sends a message with the subject *Meeting*.
2. Users Y and Z reply by sending a message with the subject *Re: Meeting*
Fundamental Interaction Models

- **Real-time ordering of events** - X’s message was sent first, Y reads it and replies to all; Z reads both X’s message and Y’s reply and then sends another reply, which references both X’s and Y’s messages.

Due to the independent delays in message delivery, the messages may be delivered as shown above. Some users may view the messages in the wrong order, for example user A might see the following order:

1. \( m_3 \) (Re: Meeting) from Z
2. \( m_1 \) (Meeting) from X
3. \( m_2 \) (Re: Meeting) from Y

Fundamental Interaction Models

- **Failure model** – defines the ways in which failure may occur in order to provide an understanding of the effects of failures (*omission, arbitrary, and timing failures*).

  - **Omission Failures** – refer to cases when a process or communication channel fails to perform actions that is supposed to do.
    - **process omission failures**: process crashes (halted and will not execute any further steps of its program ever)
      - detection with timeouts
      - crash is called **fail-stop** if other processes can detect with certainty that process has crashed
    - **communication omission failures**: dropping of messages. possible causes: network transmission error and receiver incoming message buffer overflow.
### Failure model (Cont.)

- **Arbitrary (or Byzantine) failures** – describe the worst possible failure semantics, in which any type of error may occur.
  
  - **process**: omit intended processing steps or carry out unwanted ones – it cannot be detected by seeing whether the process responds to invocations!
  
  - **communication channel**: e.g., non-delivery, corruption or duplication – it can be recognized by software and thus faulty message can be rejected. For example, checksums are used to detect corrupted messages, and message numbers can be used to detect non-existent and duplicated messages.

### Class of Failure

<table>
<thead>
<tr>
<th>Class of failure</th>
<th>Affects</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fail-stop</td>
<td>Process</td>
<td>Process halts and remains halted. Other processes may detect this state.</td>
</tr>
<tr>
<td>Crash</td>
<td>Process</td>
<td>Process halts and remains halted. Other processes may not be able to detect this state.</td>
</tr>
<tr>
<td>Omission</td>
<td>Channel</td>
<td>A message inserted in an outgoing message buffer never arrives at the other end’s incoming message buffer.</td>
</tr>
<tr>
<td>Send-omission</td>
<td>Process</td>
<td>A process completes a send, but the message is not put in its outgoing message buffer.</td>
</tr>
<tr>
<td>Receive-omission</td>
<td>Process</td>
<td>A process is put in a process’s incoming message buffer, but that process does not receive it.</td>
</tr>
<tr>
<td>Arbitrary (Byzantine)</td>
<td>Process</td>
<td>Process/channel exhibits arbitrary behaviour: it may send/transmit arbitrary messages at arbitrary times, commit omissions; a process may stop or take an incorrect step.</td>
</tr>
</tbody>
</table>

### Timing failures – applicable in synchronous DS where time limits are set on

- process execution,
- message delivery time, and
- clock drift rate

<table>
<thead>
<tr>
<th>Class of Failure</th>
<th>Affects</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock</td>
<td>Process</td>
<td>Process’s local clock exceeds the bounds on its rate of drift from real time.</td>
</tr>
<tr>
<td>Performance</td>
<td>Process</td>
<td>Process exceeds the bounds on the interval between two steps.</td>
</tr>
<tr>
<td>Performance</td>
<td>Channel</td>
<td>A message’s transmission takes longer than the stated bound.</td>
</tr>
</tbody>
</table>

**Note**: Any one of the above timing failures may result in responses being unavailable to clients within the specified time interval.

- In asynchronous DS, an overloaded server may respond too slowly, but we cannot say that it is a timing failure since no guarantee has been offered.
Failure model (Cont.)

- **Masking failures** – Possibility of constructing reliable services from components that exhibit failures.
  
  **Example**: Multiple servers that hold replicas of data can continue to provide a service when one of them crashes.

- **A service masks a failure**, either by hiding it all together or by converting it more acceptable type of failure.
  
  **Example**: Checksums are used to mask corrupted messages – effectively converting an arbitrary failure into an omission failure.
  
  - Omission failures can be hidden by using a protocol that retransmits messages that do not arrive at their destination.

- Although basic communication channels can exhibit the omission failures, it is possible to build a communication service that masks some of these failures – reliable communication is defined in terms of:
  
  - **validity** – any message in the outgoing message buffer is eventually delivered to the incoming message buffer.
  
  - **integrity** – the message received is identical to one sent, and no message are delivered twice.

Security model – security of DS can be achieved by securing the process and the channels used for their interactions and by protecting the objects that they encapsulate against unauthorized access.

Protecting access to objects

- **access rights** – specifies who is allowed to read or write its state
- **in client/server systems**: involves authentication of clients

![Diagram](image-url)
- **Security model (Cont.)**
  - The enemy is capable of sending any message to any process and reading or copying any message between a pair of processes

- **Threats** from a potential enemy can be classified as:
  - threats to processes,
  - threats to communicational channels,
  - denial of service, and
  - mobile codes

---

- **Security model (Cont.)**
  - **Threats to processes** - A process that is designed to handle incoming requests may receive a message from any other process in a DS.

  **Problems:**
  - unauthenticated requests/replies is possible; e.g., “man in the middle”
  - lack of reliable knowledge of the source of a message is a threat to correct functioning of both servers and clients:

    **Servers** – an enemy might generate an invocation with a false identity. For example, a mail server would not know whether the user behind an invocation that requests a mail item is allowed to do so or whether it was a request from an enemy.

    **Clients** – when a client receives the result of an invocation from a server, it cannot necessarily tell whether the source of the result message is from the intended server or from an enemy, perhaps “spoofing” the mail server.
• **Security model (Cont.)**

- **Threats to communication channels:** enemy may copy, alter or inject messages as they travel across network. Such attacks present a threat to the privacy and integrity of information as it travels over the network and to the integrity of the system.

  **Example** – someone can benefit by resending an invocation message requesting a transfer of a sum of money from one bank account to another.

- **Denial of service:** the enemy overloading physical resources such as bandwidth and processing capacity by generating debilitating network or server load so that network services become de facto unavailable.
  - e.g., “pings” to selected web sites

- **Mobile code**
  - requires privileges to execute on target machine
  - code may be malicious (e.g., mail worms)

- **Some techniques of defeating security threats**
  - **Cryptography and shared secrets** – Modern cryptography is based on encryption algorithms that use secret keys that are difficult to guess.
  - **Authentication** – The use of shared secrets and encryption provides the basis for authentication of messages – proving the identities supplied by their senders. This is achieved by including in a message an encrypted portion that guarantee its authenticity.
  - **Secure channels** – Encryption and authentication are used to build secure channels as a service layer on top of existing communication services. Properties of secure channels are:
    - Each knows reliably the identity of the principle whose behalf the other process is executing
    - Secure channels ensures that privacy and integrity of the data transmitted across it
    - Each message includes a physical or logical time stamp to prevent message from being replayed or reordered.