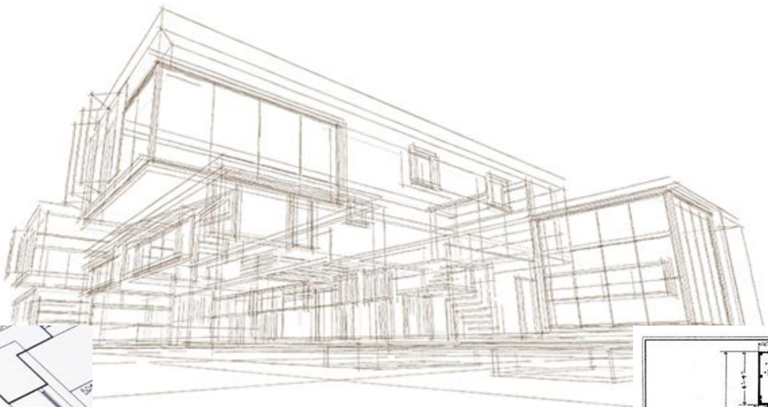
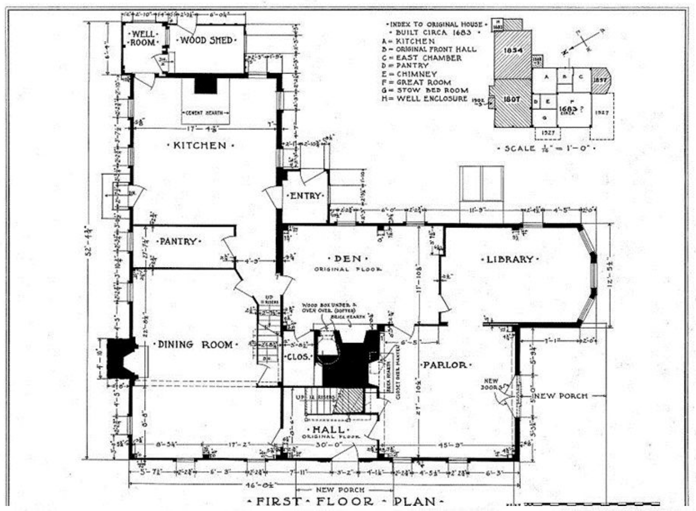
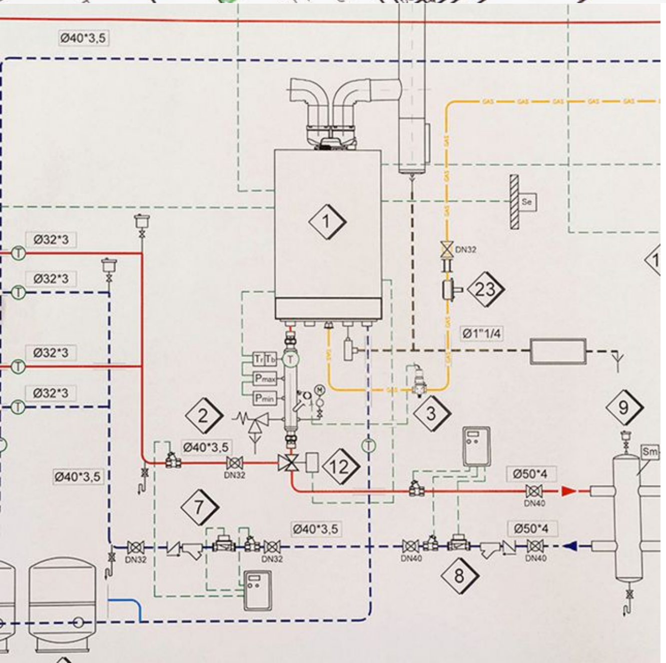
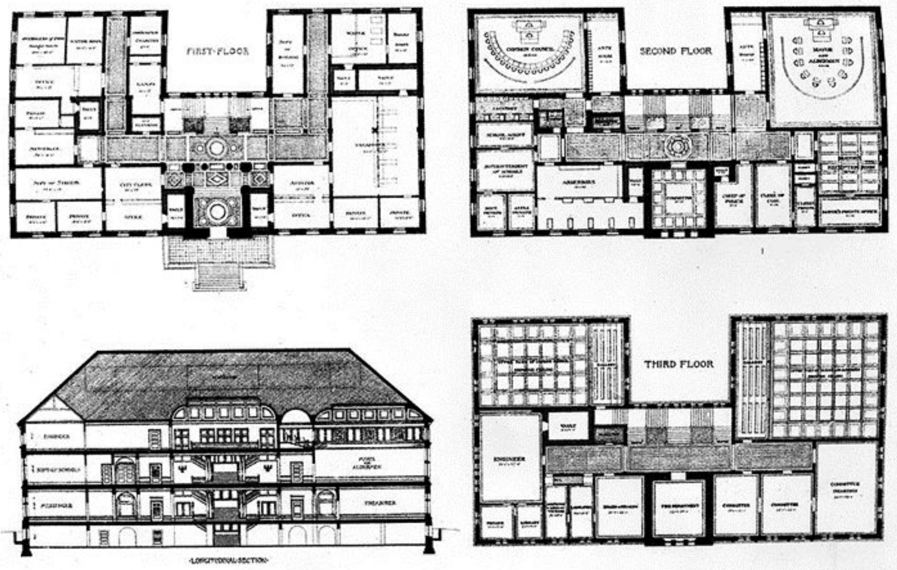
DISTRIBUTED COMPUTING
MODULE 2



- Architectural
- Structural
- Electrical
- Plumbing & Sanitary
- Finishing Drawing







PHYSICAL MODELS

- **Definition**:

  A physical model is a representation of the underlying hardware elements of a distributed system that abstracts away from specific details of the computer and networking technologies employed.

Baseline physical model:

- hardware or software components located at networked computers

- **communicate** and coordinate their actions only **by passing messages**

- This leads to a **minimal** physical model of a distributed system

EARLY DISTRIBUTED SYSTEMS:

- Emerged in the late 1970s and early 1980s
- in response to the emergence of Ethernet
- Between 10 and 100 nodes interconnected by a **local area network**
- **limited Internet connectivity**
- Supported a small range of services such as
  - **shared local printers** and **file servers**
  - **email**
  - file transfer across the Internet.
- Individual systems were largely homogeneous
- Openness not a primary concern.
- Providing quality of service was still very much in its infancy
  (focal point for much of the research)

INTERNET-SCALE DISTRIBUTED SYSTEMS:

- Larger-scale distributed systems started to emerge in the 1990s
- in response to the dramatic growth of the Internet.
  (Google search engine was first launched in 1996)
- The underlying physical infrastructure(physical model) :
  - an extensible set of nodes
  - interconnected by a *network of networks (the Internet)*
- incorporate large numbers of nodes
- provide DS services for global organizations and across organizational boundaries

- Significant level of heterogeneity in terms of:
  - networks
  - computer architecture
  - operating systems
  - languages employed and the
  - development teams involved

- Increasing emphasis on
  - **open standards**
  - **associated middleware** technologies

    eg. such as CORBA, web services

- Additional services employed to provide **end-to-end quality of service** properties

CONTEMPORARY DISTRIBUTED SYSTEMS:

Previous systems: nodes were typically desktop computers

- Static : remaining in one physical location
- Discrete: not embedded within other physical entities
- Autonomous: to a large extent independent of other computers
  - in terms of their physical infrastructure

Significant developments in physical models:

- **Mobile computing**
  - nodes such as laptops or smart phones may move from location to location in a DS
  - Hence, need for added capabilities such as
    - service discovery
    - support for spontaneous interoperation

- **Ubiquitous computing**
  - computers are **embedded** in everyday objects
  - in the surrounding environment

    eg. in washing machines or in smart homes more generally

- **Cloud computing and, in particular, cluster architectures**
  - **Led** to a move from **autonomous nodes**
  - **pools of nodes** that together provide a given service

    eg. a search service as offered by Google

- Thus **distributed systems** physical architecture:
  - significant increase in the level of **heterogeneity**
    - ***tiniest embedded devices*** *utilized in ubiquitous computing through to* ***complex computational elements*** *found in Grid computing*
  - Deploy an increasingly **varied set of networking** technologies
  - Offer a **wide variety of applications** and services
- Such systems potentially involve up to hundreds of thousands of nodes

<u>DISTRIBUTED SYSTEMS OF SYSTEMS</u>

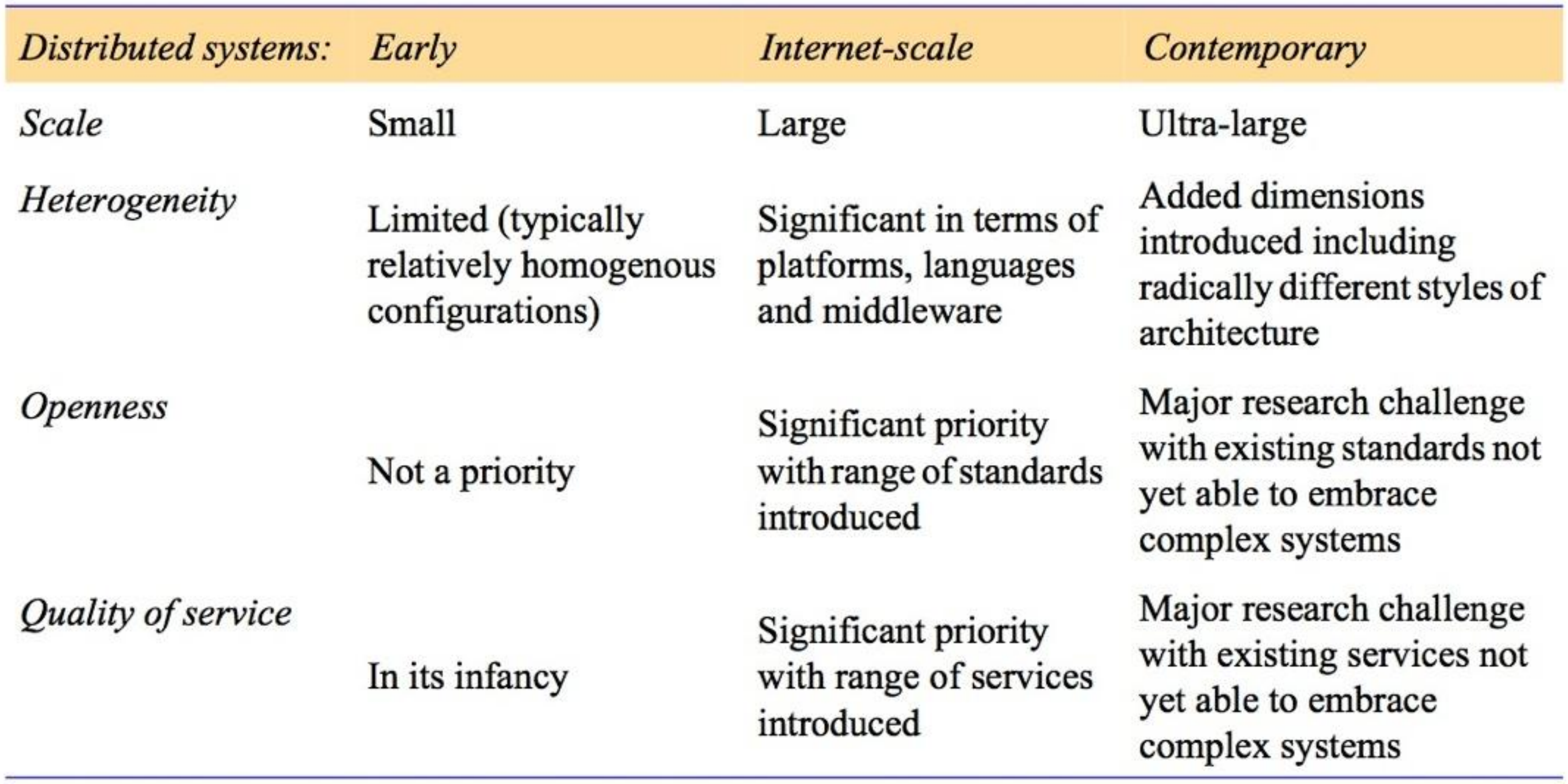
- Ultra-large-scale (ULS) distributed systems –
  - the complexity of modern distributed systems
  - physical architectures referred to as systems of systems (mirroring the view of the Internet as a network of networks).
- A system of systems –
  - a complex system consisting of a series of **subsystems**
  - systems in their own right
  - that **come together** to perform a **particular task** or tasks

Eg. Consider an environmental management system for flood prediction.

In such a scenario, there will be

- **sensor networks** deployed to monitor various environmental parameters relating to rivers, flood plains, tidal effects and so on
- systems that are responsible for **predicting the likelihood of floods** by running (often complex) simulations on cluster computers
- systems to **maintain and analyze historical data**
- systems to **provide early warning**

Generations of distributed systems:



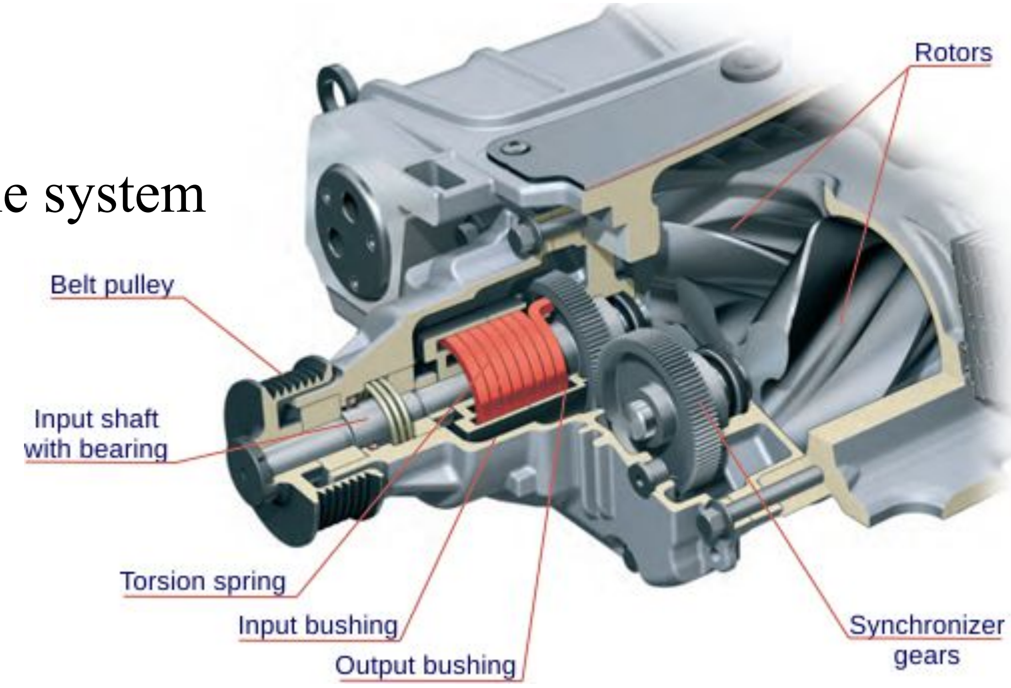
Distributed systems:	*Early*	*Internet-scale*	*Contemporary*
Scale	Small	Large	Ultra-large
Heterogeneity	Limited (typically relatively homogenous configurations)	Significant in terms of platforms, languages and middleware	Added dimensions introduced including radically different styles of architecture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

ARCHITECTURAL MODELS

- The architecture of a system: its structure in terms of separately specified components and their interrelationships.

- The overall goal is to ensure that the structure will meet present and likely future demands on it.

- Major concerns are to make the system
  - reliable
  - manageable
  - adaptable
  - cost-effective



SUPERCHARGER DRIVE WITH TORSIONAL COUPLING

Architectural Models

- Architectural Elements
  - Communicating Entities
  - Communication Paradigms
  - Roles and Responsibilities
  - Placement
- Architectural Patterns
  - Layering
  - Tiered Architecture
  - Thin clients
- Middleware Solutions

Architectural Elements

A. What are the entities?
B. How do they communicate?
C. What roles do they play?
D. How are they mapped in the physical architecture?

COMMUNICATING ENTITIES

1. SYSTEM PERSPECTIVE-
    1. Processes.

        (Except sensor networks and distributed systems with threads)
2. PROGRAMMING PERSPECTIVE-
    1. **Object-**
        - interacting objects representing natural units of decomposition for the given problem domain
        - accessed via interfaces,
        - an associated interface definition language (or IDL) providing a specification of the methods defined on an object

2. **Components-**

Components resemble objects

- they offer **problem-oriented abstractions** for building distributed systems and are also
- accessed through interfaces.

The key difference is that components specify

- not only their (provided) interfaces but **also the assumptions they make**
- **other components/interfaces that must be present for a component to fulfil its function**

i.e. **making all dependencies explicit**

providing a more complete contract for system construction

- encourages third-party development of components and promotes purer composition approach

- additional support for key areas such as deployment and support for server-side programming

3. **Web Services-**

- Approach based on **encapsulation** of behaviour and access through interfaces (like objects and components)
- In contrast, however, web services are intrinsically **integrated into the World Wide Web**, <u>using web standards to represent and discover services.</u>

  i.e. They are partially defined by the web-based technologies they adopt.

STYLE OF USE OF THE TECHNOLOGY

- Objects and components are often used within an organization to develop tightly coupled applications.
- Web services are generally viewed as **complete services** in their own right that can be **combined to achieve value-added services** (often crossing organizational boundaries and hence achieving business to business integration).

COMMUNICATION PARADIGMS

1. Inter-process Communication

Low level support for communication between processes

- Message passing primitives
- Direct access to API offered by IP(socket programming)
- Support for multicast communication

2. Remote Invocation

Based on two way exchange between communicating entities

1. <u>Request - reply protocols</u>(client server computing)
    1. Request- encoding of operation to be processed with arguments
    2. Results of operation

    Primitive method, used in embedded systems

2. <u>Remote Procedure Calls (RPC)</u>
    1. Call procedures in processes on remote computers
    2. RPC s/m hides aspects of distribution including
        1. Encoding and decoding of parameters and results
        2. Passing of messages
        3. Preserving required semantics
    3. Access and location transparency

    (Client server computing proving set of operations through service interface)

3. <u>Remote Method Invocation(RMI)</u>
   i. Resembles RPC but in terms of distributed objects
   ii. Calling object can invoke a method in a remote object
   iii. Underlying details hidden from user
   iv. Go further by supporting object identity and ability to pass object identifiers
   v. Integration into object oriented languages

Here senders and receivers know each other and must exist at the same time

4. <u>Indirect Communication</u>

Communication in indirect through a third entity

Hence strong degree of decoupling between sender and receiver

- **i. Space uncoupling:** Senders do not need to know receivers
- **ii. Time uncoupling:** Senders and receivers need not exist at the same time

Techniques

1. Group Communication:

   Delivery of messages to a set of recipients (one to many communication)

   Group identifiers, group membership

2. Publish Subscribe System:

   Information - dissemination

   Producers(or publishers) distribute info items of interest (events) to similarly large group of consumers (or subscribers)

   Distributed event based system

   Financial trading

3. Message Queues:

   Point to point service

   Queues indirection between producer and consumer

4. Tuple Spaces:

Arbitrary items of structured data called tuples

Tuples can be placed by processes in persistent tuple space

Other processes can read or remove tuples by specifying interest

5. Distributed Shared Memory

Abstraction for sharing data like it was local memory read and write

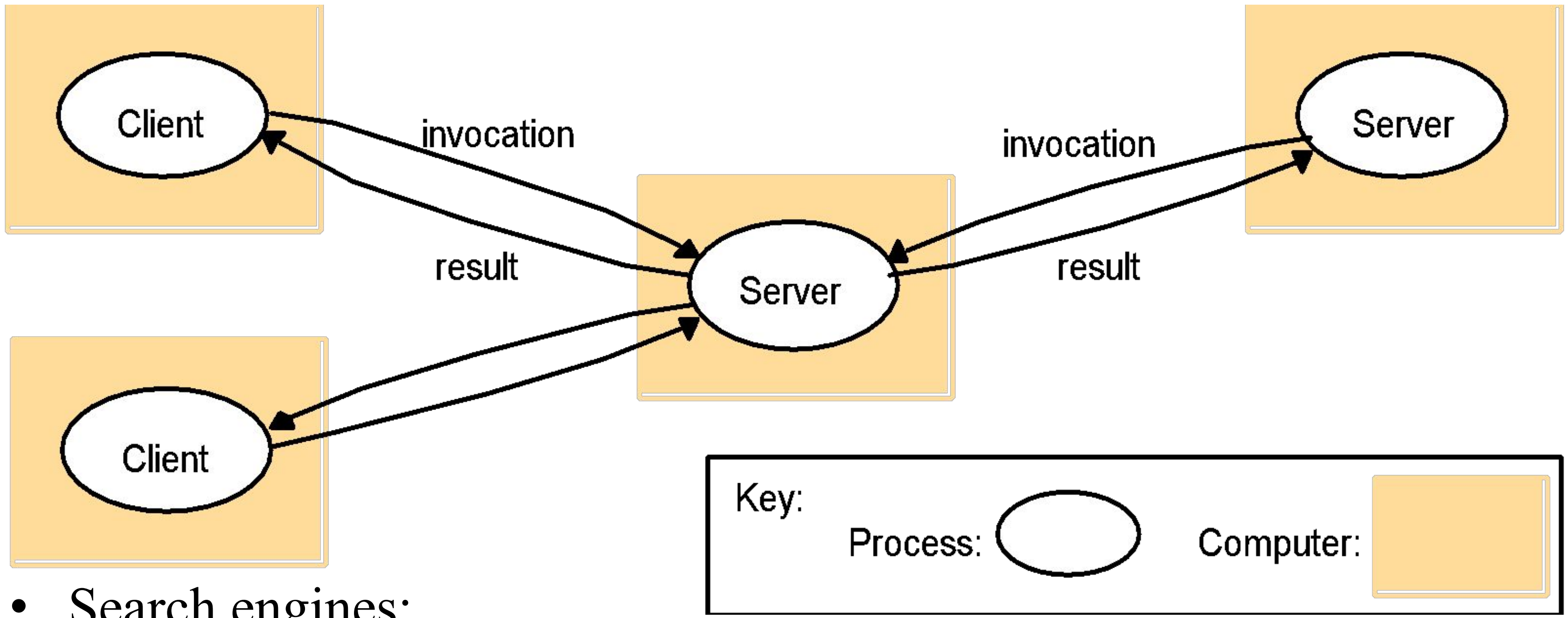
High level of distribution transparency

Ensure data is provided in time, synchronisation and consistency of data

ROLES AND RESPONSIBILITIES

Processes take on roles to perform a useful activity

1. Client Server model:

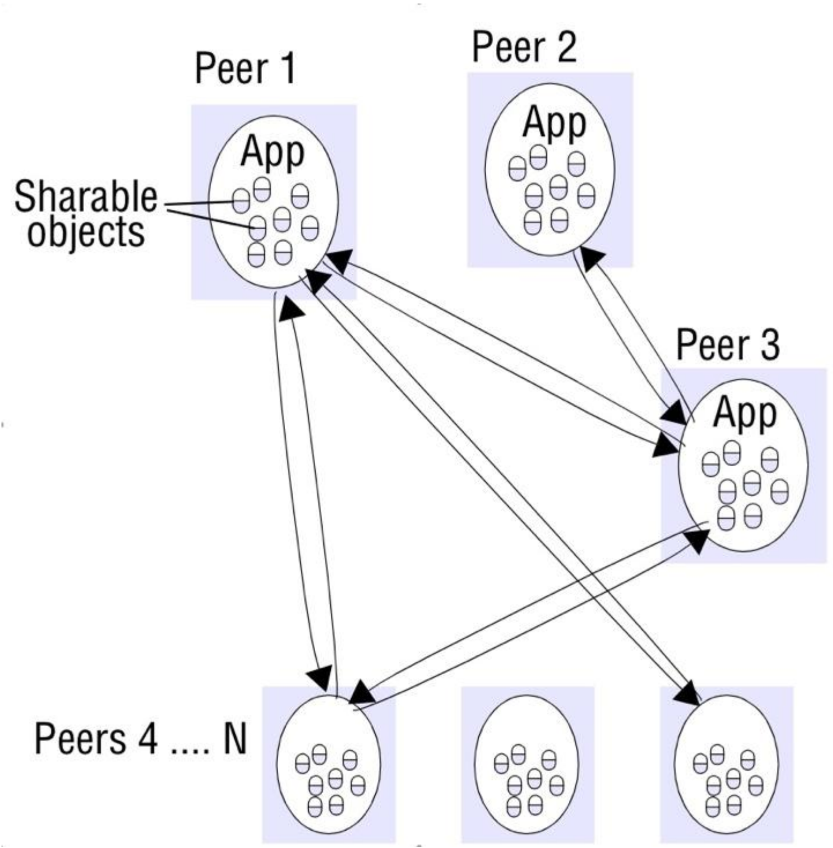


- Search engines:

  responds to queries from browser clients making web pages available

  Runs web crawlers that serve as clients to various other servers
- Web servers are clients of DNS

2. Peer – to –Peer Model:



- All participants run the same program and offer same set of interfaces.
- Distribute shared resources to share computing and communication loads
- Better scalability than client server
- Service grows with the number of users
- Eg. Napster, bit Torrent, Spotify, Tradepal

PLACEMENT

Must be determined with strong application knowledge

Strategies:

1. Mapping of services to multiple servers
2. Caching
3. Mobile code
4. Mobile agents

Mapping of Services to multiple Servers

1. Partition set of objects on which the service is based and distribute objects between themselves.

   Eg. Web servers

2. Servers may maintain replicated copies of these objects on several hosts.

   Eg. Password file in servers

Caching

- A store of recently used data objects closer to a client.
- New objects added to the cache store when received, replacement
- Caches may be co located with each client or may be in proxy server
- Web browsers maintain cache

Mobile Code

- Some websites use functionalities not found in common browsers and require downloading additional code.
- Keeping up-to-date using push operations (server initiates)
- Mobile code is potential security threat to local computer

Mobile Agents

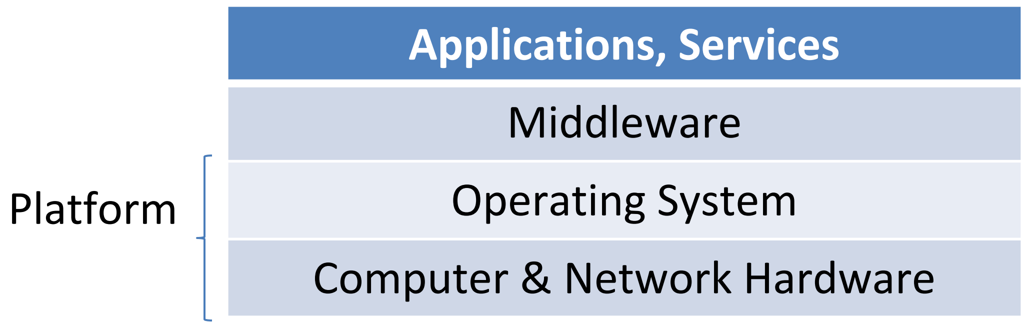
- A running program(code and data) that travels from one computer to another
- Carry out task in someone's behalf and eventually return result
- Eg. Collecting info, compare prices of product from different vendors, install and maintain software.
- Potential Security threats to the resources in computers they visit.
- Identity of the user in secure manner, resource allowed

Architectural Models

- Architectural Elements
  - Communicating Entities
  - Communication Paradigms
  - Roles and Responsibilities
  - Placement
- **Architectural Patterns**
  - Layering
  - Tiered Architecture
  - Thin clients
- Middleware Solutions

LAYERING

- A complex system is partitioned into a number of layers
- Each layer makes use of services offered by the layer below
- Abstraction:
  - higher layers unaware of implementation details
  - unaware of any other layers beneath them
- DS: a vertical organization of services into service layers
- **Platform –**
  - lowest-level hardware and software layers
  - These low-level layers provide services to the layers above them
- **Middleware –**
  - a layer of software whose purpose is to mask heterogeneity
  - to provide a convenient programming model to application programmers



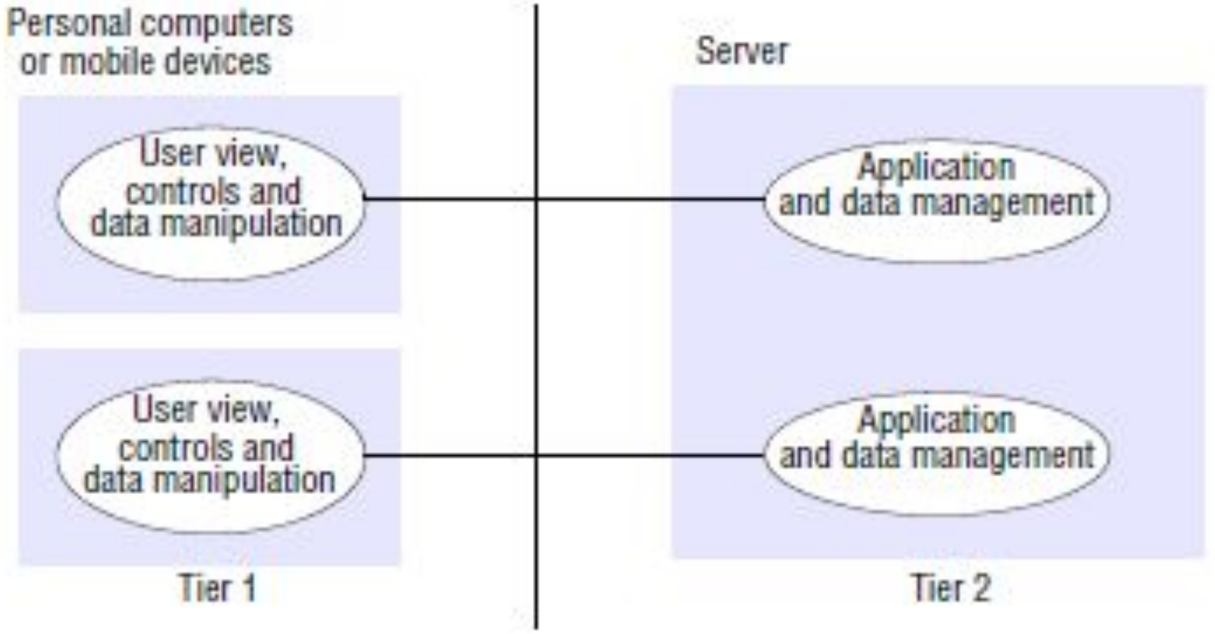
SOFTWARE AND HARDWARE SERVICE LAYERS IN DISTRIBUTED SYSTEMS

TIERED ARCHITECTURE

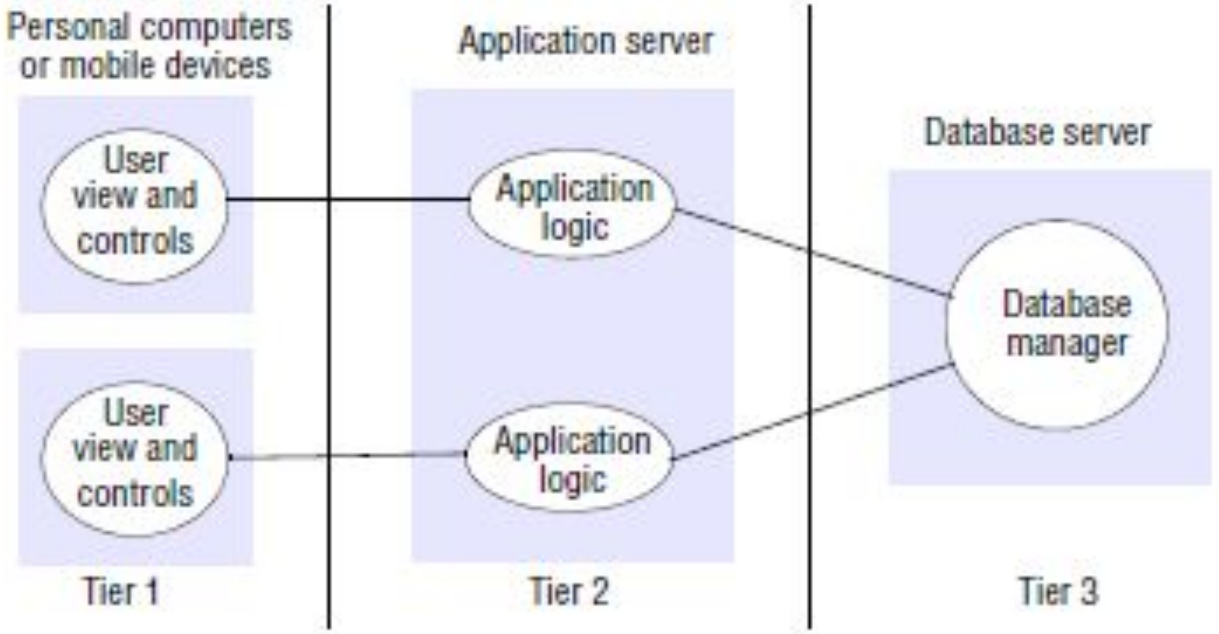
- organize functionality of a given layer and place this functionality into
  - appropriate servers (services)
  - physical nodes (applications)
- Functional decomposition of a given application:
  - Presentation Logic
  - Application Logic
  - Data Logic
- Two tier solution:
  - Application logic is split
  - Adv: Low latency
  - Disadv: Application logic spilt across process boundary thus restriction on which parts of the logic can be directly invoked from which other part.
- Three tier solution:
  - one-to-one mapping from logical elements to physical servers
  - Can enhance maintainability of the software
  - Well defined role for each tier
  - Support for thin clients
  - Added complexity of managing additional servers & added network traffic

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)



- This approach generalizes to **n-tiered** (or multi-tier) solutions:
  - where a given application domain is partitioned into n logical elements, each mapped to a given server element.

  eg. Wikipedia- adopts a multi-tier architecture to deal with the high volume of web requests (up to 60,000 page requests per second).

- The role of AJAX:
  - AJAX (Asynchronous Javascript And XML) is an extension to the standard client-server style of interaction used in the World Wide Web.
  - Meets the need for fine-grained communication between a **Javascript front-end program** running in a web browser and a **server-based back-end program** holding data describing the state of the application.
  - In the standard web style of interaction:
    - A browser sends an **HTTP request to a server** for a page, image or other resource with a given URL.
    - The server replies by **sending an entire pag**e that is either read from a file on the server or generated by a program
    - Browser presents it according to the relevant display method

Constraints:

- Once an **HTTP request for a new web page**, the user is **unable to interact with the page** until the new HTML content is received and presented by the browser.

  This time interval is indeterminate, because it is subject to network and server delays.

- In order to **update even a small part of the current page** with additional data from the server, **an entire new page must be requested** and displayed.
  - Delayed response to the user
  - additional processing at both the client and the server
  - redundant network traffic.

- The contents of a page displayed at a client cannot be updated in response to changes in the application data held at the server.

SOLUTION:

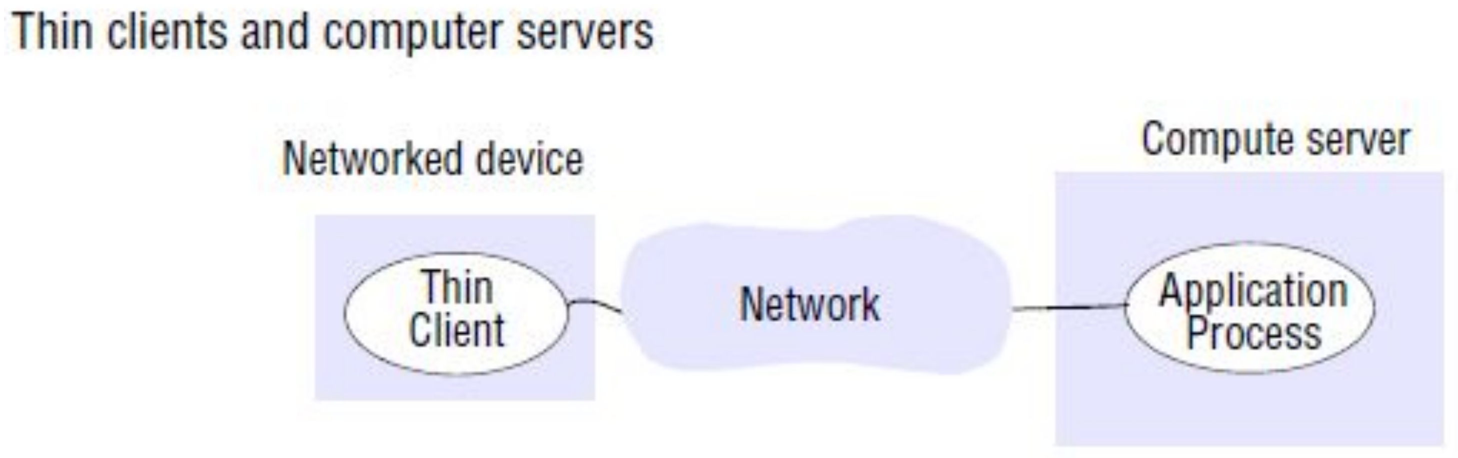
1. **Javascript**, a cross-platform and cross-browser programming language that is downloaded and executed in the browser, constituted a first step towards the removal of those constraints.
   - It is a general-purpose language enabling both user interface and application logic to be programmed and executed in the context of a browser window.

2. **AJAX** is the second innovative step that was needed to enable major interactive web applications to be developed and deployed.
   - It enables Javascript front-end programs to request new data directly from server programs. (*XmlHttpRequest object)*
   - Any data items can be requested and the current page updated selectively to show the new values.
   - Indeed, the front end can react to the new data in any way that is useful for the application.

AJAX example: soccer score updates

new Ajax.Request('scores.php?game=Arsenal:Liverpool',
{onSuccess: updateScore});
function updateScore(request) {
.....
(*request contains the state of the Ajax request including the returned result.*
The result is parsed to obtain some text giving the score, which is used
to update the relevant portion of the current page.)
.....
}

THIN CLIENTS



- **Moving complexity away from the end-user**

- Enabling **access to sophisticated networked services** with few assumptions or demands on the client device
  - Eg. cloud services

- **Simple local devices** -enhanced with a plethora of networked services and capabilities

- **Drawback:** delays experienced by users are increased in **highly interactive graphical activities** such as CAD and image processing

VIRTUAL NETWORK COMPUTING (VNC):

- Providing remote access to graphical user interfaces
- VNC client (or viewer) interacts with a VNC server through a VNC protocol
- The protocol operates at a primitive level in terms of graphics support, based on frame buffers and featuring one operation:
  - the placement of a rectangle of pixel data at a given position on the screen
- Low level approach ensures the protocol will work with any operating system or application
- Represents a significant step forward in mobile computing
- Users are able to access their computer facilities from anywhere on a wide range of devices

Other Architectural Patterns:

1. Proxy pattern:
   - a proxy created in the local address space to represent the remote object
   - Proxy offers exactly the same interface as the remote object
   - the programmer makes calls on this proxy object
   - does not need to be aware of the distributed nature of the interaction

2. Brokerage Pattern:
   - Supporting interoperability in potentially complex distributed infrastructures.
   - In particular, this pattern consists of the trio
     - service provider
     - service requester
     - service broker

3. Reflection:
   - Introspection (the dynamic discovery of properties of the system)
   - intercession (the ability to dynamically modify structure or behaviour)

MIDDLEWARE SOLUTIONS ASSIGNMENT

- Select any one middleware of your choice. Create a Google spreadsheet that will be shared by the members of the team.
- Each person in the group has to contribute some new facts about the middleware in not less than 50 words.
- So those who are to add the facts first check the spreadsheet to see what the others have added.
- Each person’s contribution should be in a different colour. Mention the colour corresponding to the person at the bottom of the sheet.
- Anyone can start. Whoever watches the video first can select the middleware software, create the sheet and make their entry.
- Last date of submission is 24/10/2020.
- Submit the link to the spreadsheet once it is created.

Group 1: Roll numbers 1-10
Group 2: Roll numbers 11-20
Group 3: Roll numbers 21-30
Group 4: Roll numbers 41-10
Group 5: Roll numbers 41-50
Group 6: Roll numbers 51-61

FUNDAMENTAL MODELS

All the models of systems share some fundamental properties:

- **processes** that **communicate** with one another by sending messages over a computer network

- Share the **design requirements** of achieving the
  - **performance**
  - **reliability**
  - **ensuring the security** of the resources

- Fundamental model should contain only the **essential ingredients** in order to **understand and reason** about some aspects of a system's behaviour.

The purpose of such a model is:

- To **make explicit** all the **relevant assumptions** about the systems
- To **make generalizations** concerning what is **possible or impossible**
- The generalizations may take the form of **general-purpose algorithms** or **desirable properties** that are **guaranteed**.
- The guarantees are dependent on
  logical analysis
  mathematical proof
- Decide whether a design will work if implemented it in a particular system
  (By knowing what our designs do, and do not, depend upon)
- We can hope to prove system properties using mathematical techniques
- Help clarify our understanding of our systems

The aspects of distributed systems in fundamental model help discuss and reason:

- *Interaction:*
  - communication takes place with **delays** that are often of **considerable duration**
  - the **accuracy** is limited by :
    - **these delays**
    - **difficulty of maintaining** the **same notion of time** across all the computers

- *Failure:*
  - **The correct operation** of a distributed system is **threatened** whenever a **fault** occurs in any of the **computers** or in the **network**
  - This model defines and classifies the faults.

- *Security:*
  - The **modular nature** of distributed systems and their **openness exposes them to attack** by both external and internal agents.
  - The security model **defines and classifies** the forms of such attacks
  - provides a basis for
    - the analysis of threats to a system
    - the design of systems that are able to resist threats

1. **Interaction Model:**

Distributed system: many processes interacting in complex ways.

Eg. Multiple server processes cooperate (DNS), peer processes(voice conference)

- Algorithm: **distributed algorithm** for distributed systems –
  - steps to be taken by each process including transmission of messages between them
- Message transfer – info transfer and coordination
- Difficult to predict: Process rate and timing of message transmission
- Difficult to describe all the states of distributed algorithm (failures possible)
- Process:
  - Process performs activity
  - Process has its own state (private)
  - State consists of data it can access and update

Performance of Communication Channels:

- Variety of communication channels (streams, messages)

- Latency: delay b/w msg transmission and beginning of receipt
  - Time taken for first string of transmitted bit tot reach destination (eg satellite)
  - Network delay when heavily loaded (eg ethernet)
  - Time taken by OS communication services at both ends

- Bandwidth: shared by number of communication channels

- Jitter : Variation in time to deliver series of mgs (multimedia data)

Computer Clocks an Timing Events:

- Each computer has internal clock
- Timestamps for processes in different computers
- Computer clocks drift from perfect time at different rate in different computers.
- Clock Drift Rate
- Corrective approaches:
  - Get time readings from GPS for every computer:
    - Not practical and expensive
  - Computer with GPS can send timing msgs to others in the network(msg delays)

Variants of Interaction Model:

Hard to set limits on time taken for process execution, msg delivery, clock drift.

1. **Synchronous DS**:

These systems have the following bounds:

- The time to execute each step of a process has known lower and upper bounds.
- Each message transmitted over a channel is received within a known bounded time.
- Each process has a local clock whose drift rate from real time has a known bound.

- Suggest likely upper and lower bounds
- Difficult to arrive at realistic values and provide guarantees of those values.
- Design may not be reliable unless guarantees possible
- Partial solution- Timeouts to detect failures
- Possible with known requirements and guaranteed environments

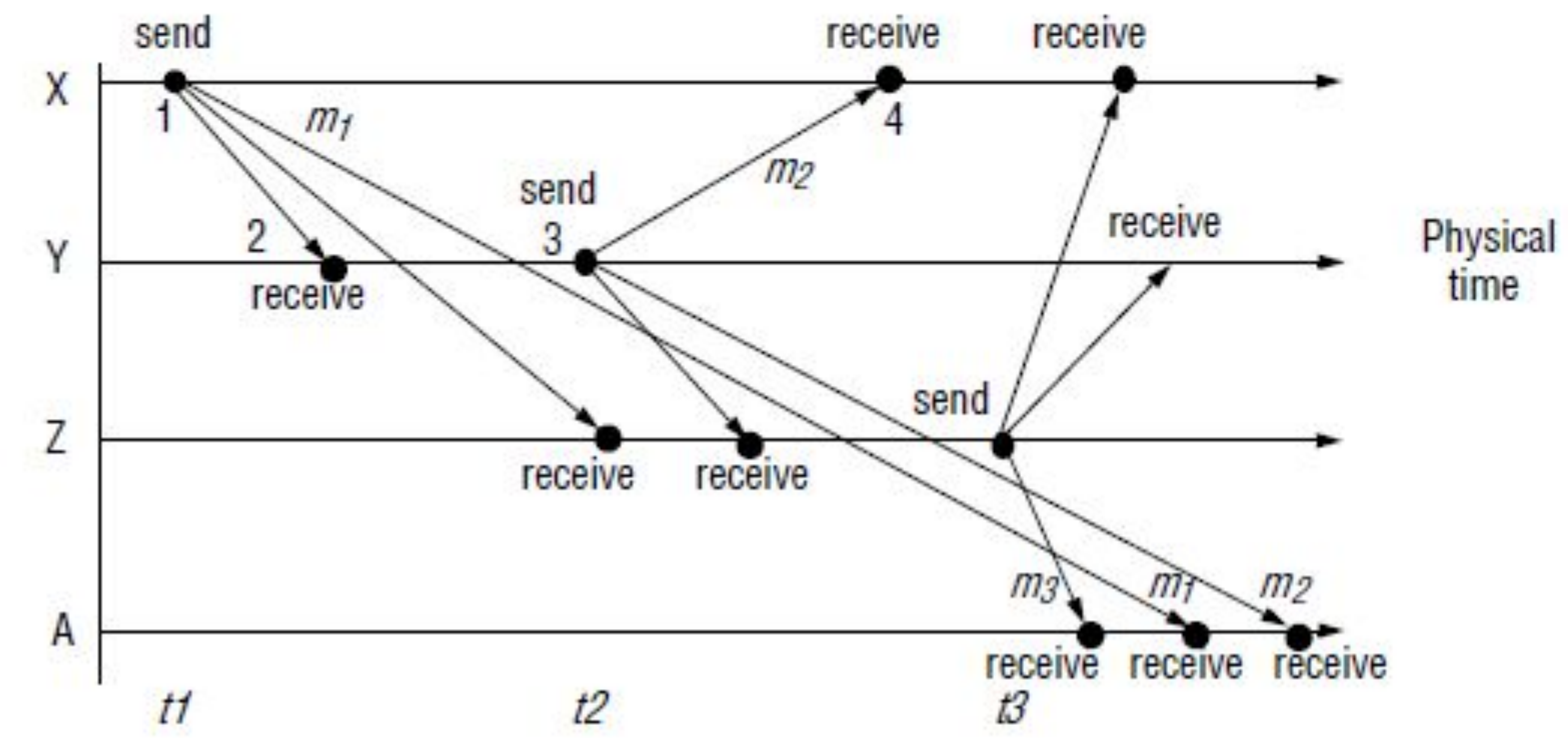
2. **Asynchronous DS:**

Many DS are useful without being synchronous(internet)

- No bounds on execution speed, transmission delay, clock drift rates.
- No assumptions about time intervals in any execution
  eg. Sometimes emails can take days to arrive.
- But they work reasonably well.
  - Eg. Web cannot always provide a particular response within a reasonable time limit
  - browsers have been designed to allow users to do other things while they are waiting.
- Any solution that is valid for an asynchronous distributed system is also valid for a synchronous one.
- Actual distributed systems are often asynchronous because of need for:
  - processes to share the processors
  - communication channels to share the network

Event Ordering:

Real-time ordering of events



2. Failure Model

- Both processes and communication channel may fail – depart from what is considered to be correct or desirable behaviour.

- This model defines the ways in which failure may occur to provide an understanding of the effects of failure.

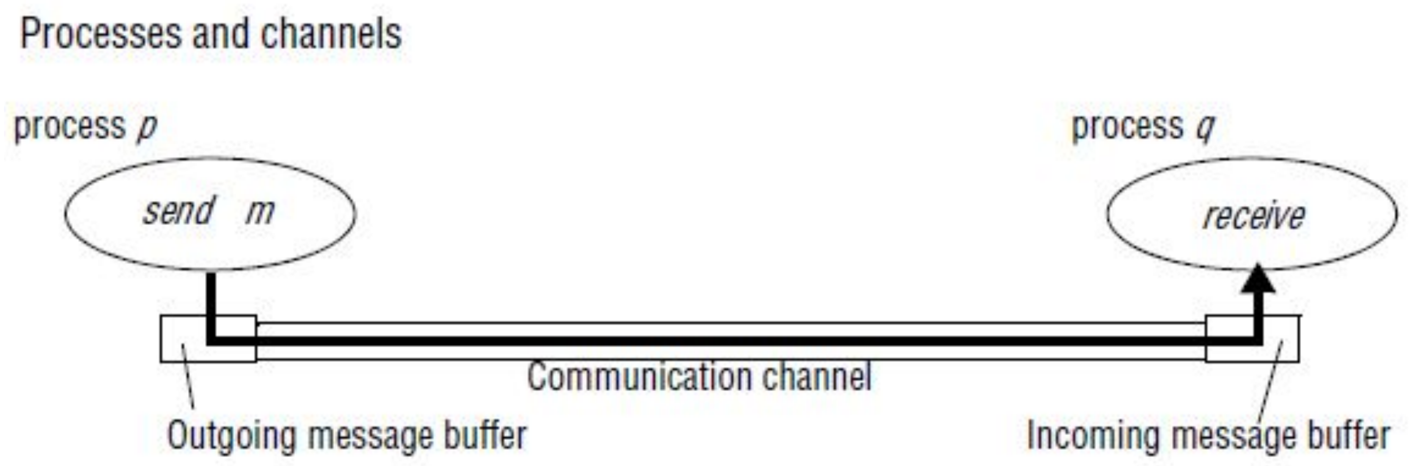
- **Omission Failures**

  When a process or communication channel fails to perform actions that it is supposed to do.

PROCESS OMISSION

- Process crashes: process has halted and will not execute further any of its steps ever.
- Services that can survive crashes are simplified under the assumption that the services that they depend on crash cleanly.
- Detect crash by checking for response upon invoking messages
- But timeouts in asynchronous system can also mean the process is slow or message has not arrived.
- Fail stop – when a process can detect with certainty that another process has crashed

COMMUNICATION OMISSION



- Both buffers provided by OS
- **Failure** when msg is not transferred from p's buffer to q's buffer.
- **Dropping of msgs**: due to
  - Lack of buffer space at the receiver end or intermediate gateway
  - Network transmission error
  - Types:
    1. Send omission failure
    2. Receive omission failure
    3. Channel omission failure

ARBITRARY FAILURES (Byzantine failure)

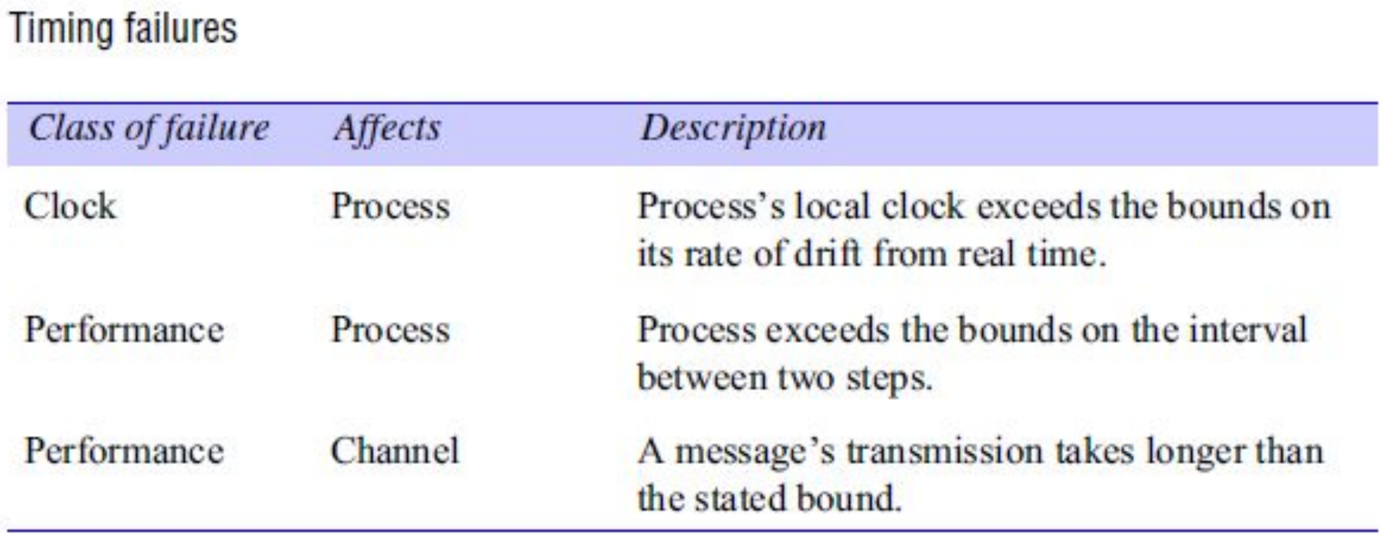
- Any type of failure
- Process sets or return wrong values
- Omit intended processing steps or take unintended processing steps
- Difficult to detect failure by invoking messages
- Arbitrary failures in communication channels by data corruption, delivering non existent msgs or delivering msgs more than once
- Arbitrary failures in communication channels are rare.

Omission and arbitrary failures

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

Timing Failures

- Applicable in synchronous distributed systems where time limits are set



Timing failures

Class of failure	*Affects*	*Description*
Clock	Process	Process's local clock exceeds the bounds on its rate of drift from real time.
Performance	Process	Process exceeds the bounds on the interval between two steps.
Performance	Channel	A message's transmission takes longer than the stated bound.

- Asynchronous systems- overloaded servers respond slowly but it cannot be called timing failure since no guarantee was offered

- Real time OS: timing guarantees are there, more complex system

- Timing relevant for multimedia computers

- **Masking Failures**
  - Each component made of a collection of component
  - Possible to construct reliable services from components that exhibit failure
  - Eg. Using multiple servers
  - Knowledge of failure characteristics help design systems that mask failure
  - Service mask

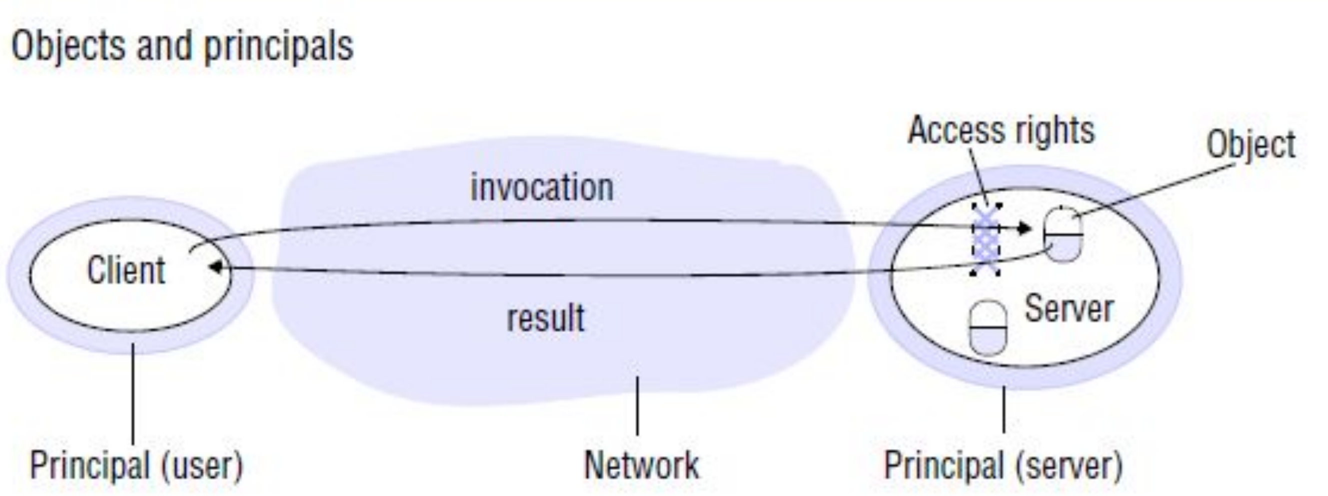
Reliability of one to one communication

- The term 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 messages are delivered twice.
- The threats to integrity come from two independent sources:
  - **Any protocol that retransmits messages but does not reject a message that arrives** twice. Protocols can attach sequence numbers to messages so as to detect those that are delivered twice.
  - **Malicious users that may inject spurious messages, replay old messages or tamper** with messages. Security measures can be taken to maintain the integrity property in the face of such attacks.

3. Security Model

Security of a distributed system can be achieved by securing the processes and the channels used for their interactions and by protecting the objects that they encapsulate against unauthorized access.

PROTECTING OBJECTS



- Include users in the model as the beneficiaries of access rights
- Associate with each invocation and each result the authority on which it is issued. Such an authority is called a principal. A principal may be a user or a process.

SECURING PROCESSES AND THEIR INTERACTION

- The messages are exposed to attack because the network and the communication service that they use are open, to enable any pair of processes to interact.
- Financial transactions, confidential or classified information or any other information whose secrecy or integrity is crucial. Integrity is threatened by security violations as well as communication failures.

Model For The Analysis Of Security Threats

The enemy

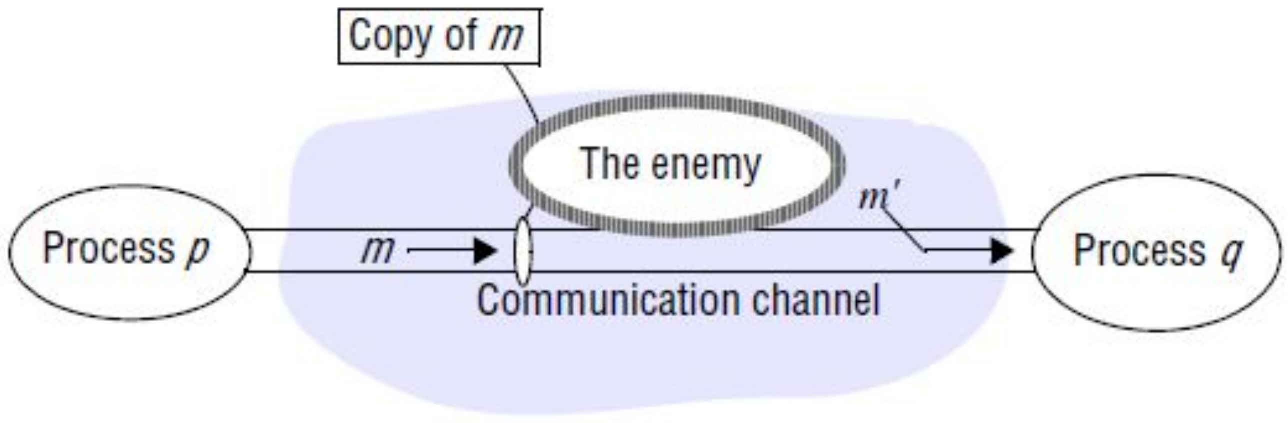
- To model security threats, we postulate an enemy (sometimes also known as the adversary) that is capable of sending any message to any process and reading or copying any message sent between a pair of processes.

Such attacks can be made simply by

- using a computer connected to a network
  - to run a program that reads network messages addressed to other computers on the network,
  - or a program that generates messages that make false requests to services, purporting to come from authorized users.

- The attack may come from a computer that is legitimately connected to the network or from one that is connected in an unauthorized manner.

The enemy



Threats to processes:

A process that is designed to handle incoming requests may receive a message from any other process in the distributed system, and it cannot necessarily determine the identity of the sender.

1. Servers:
   - Cannot determine if the request is from a legitimate user.
   - Identity can be forged.
   - For example, a mail server would not know whether the user behind an invocation that requests a mail item from a particular mailbox is allowed to do so or whether it was a request from an enemy.

2. Clients
   - Cannot determine if the result obtained is from a legitimate server. (Mail server spoofing)

Threats to communication channels:

- An enemy can copy, alter or inject messages as they travel across the network and its intervening gateways.

- A threat to the privacy and integrity of information

- Threat to the integrity of the system.

  For example, a result message containing a user's mail item might be revealed to another user or it might be altered

- Another form of attack is the attempt to save copies of messages and to replay them at a later time, making it possible to reuse the same message over and over again.

  For example, someone could benefit by resending an invocation message requesting a transfer of a sum of money from one bank account to another.

Defeating security threats

- Cryptography and shared secrets
  - A pair of processes **share a secret**
  - **Cryptography** is the science of keeping messages secure, and encryption is the process of scrambling a message in such a way so as to hide its contents(Encryption algorithms)

- **Authentication:** Include in a message an encrypted portion that contains enough of the contents of the message to guarantee its authenticity

- **Secure channels**: Encryption and authentication are used to build secure channels as a service layer on top of existing communication services.
  - Communication channel connecting a pair of processes, each of which acts on behalf of a principal

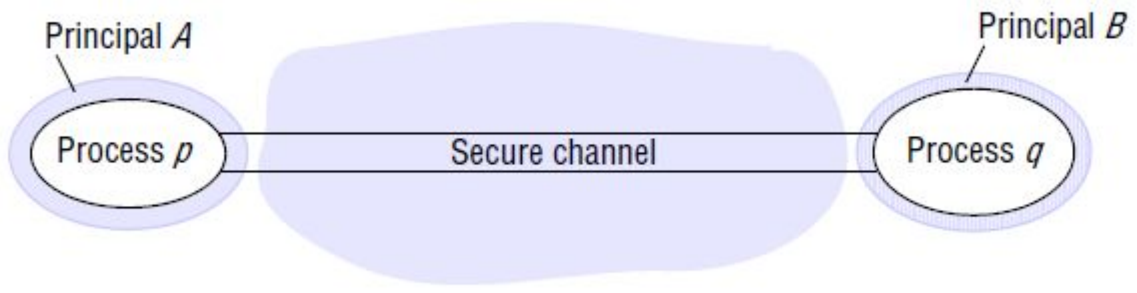
A secure channel has the following properties:

1. Each of the processes knows reliably the identity of the principal on whose behalf the other process is executing.

2. A secure channel ensures the privacy and integrity (protection against tampering) of the data transmitted across it.

3. Each message includes a physical or logical timestamp to prevent messages from being replayed or reordered.

Secure channels



Other possible threats from an enemy

1. Denial of service
2. Mobile code

The Uses Of Security Models

- Analysis and design of secure systems in which these costs are kept to a minimum

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

- The effectiveness and the cost of the security techniques needed can then be balanced against the threats.

Thank you!