Distributed Computing

Semester - 7

BTech - 2022-23

Weekly load hrs	Lectures	Laboratory	Credits
Hrs/w	3hr/w	2hr/w	3 + 1

Syllabus to be shared with students

Distributed Computing

Course Contents (All 5 modules):

- Introduction to Distributed Computing,Distributed Systems
- Communication in Distributed Systems
- Synchronization in distributed systems
- Distributed file systems
- Introduction to cloud computing

Syllabus

Module 1

- Introduction to Distributed Systems
- Definition, Characterization of distributed systems - Trends in distributed systems, Focus on resource sharing, Challenges.
- Types of distributed systems.
- A Model of Distributed Computations
- System Models - Introduction, Physical models , Architectural models , Fundamental models

Distributed Computing - Where is it placed in subjects studied

The hierarchal approach:-

Applications related subjects [eg. full stack]

S/w Engineering, OOMD

DBMS

OS
Networking,
Data Communication, NIS

DC

Hardware, MIT, DEL

Distributed Computing

Arriving here :----

There are majorly 2 kinds of systems

- Tightly coupled (memory sharing)

---- parallel processing systems (Distributed Computing)

- Loosely coupled (independent memory)

---- Distributed Computing systems/Distributed Sytems/Distributed Computing

The TERMS

- Distributed Systems
- Distributed Computing Systems
- Distributed Computing
- Distributed Operating systems

Distributed computing is a field of computer science that studies distributed systems.

Distributed Computing

Distributed system: a collection of independent computers that are connected with an interconnection network.

Distributed computing system : A distributed computer system consists of multiple software components that are on multiple computers, but it runs as a single system. The computers that are in a distributed system can be physically close together and connected by a local network, or they can be geographically distant and connected by a wide area network.

Distributed Computing systems are many times known simply as Distributed Systems

Distributed computing : a method of computer processing in which different parts of a computer program are run on two or more computers that are communicating with each other over a network.

Distributed Operating System : Software for Distributed Systems

Distributed Computing

- Distributed computing refers to the use of distributed systems to complete computing tasks.

- It applies the principles of distributed systems to execute programs on different computer systems connected in a distributed system

Distributed Computing

- Distributed computing and distributed systems share the same basic properties of scalability, fault tolerance, resource sharing, and transparency.

- Distributed computing and distributed systems share the same benefits; namely, they're reliable, cheaper than centralized systems, and have larger processing capabilities.
- However, there's a slight difference in the challenges faced in distributed computing. For example, it's challenging to partition computational tasks into different parts that can be executed on different computers systems in **parallel. [HPC/parallel programming]**

- Additionally, we need to coordinate tasks in distributed computing.

Distributed Computing

Distributed Computing systems software.

- Distributed Operating System(DOS)
- Networking Operating System(NOS)

Distributed Operating System

- What is a Distributed Operating System ?
- How is it different ?
- Why Distributed Operating Systems ?
- Problems with Distributed Operating Systems
- Distributed Operating System Models

What is a Distributed Operating System ?

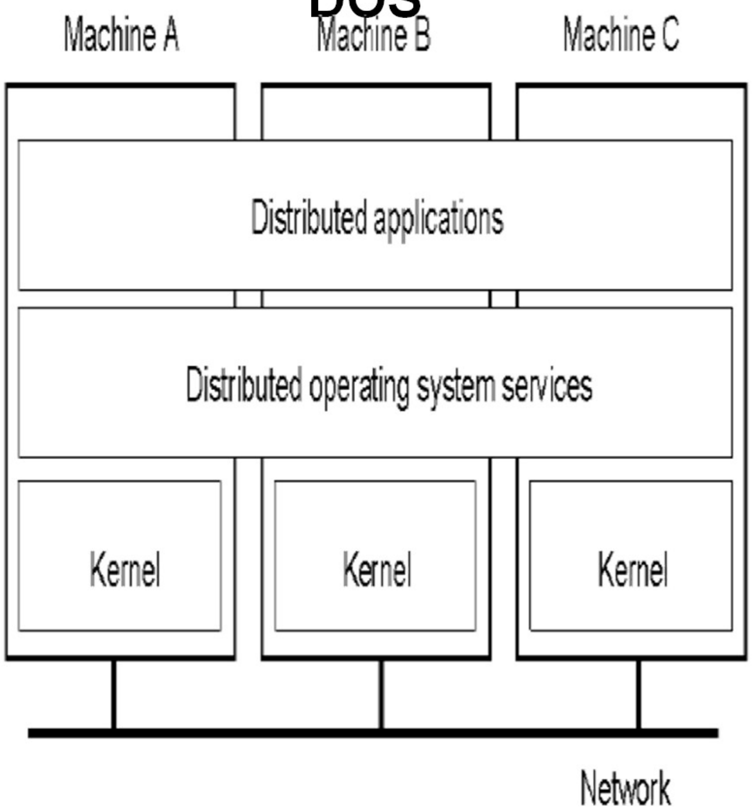
- A Distributed Operating System is the one that runs on multiple autonomous CPUs which provides its users an illusion of an ordinary
- Centralized Operating System that runs on a Virtual Uni-processor.
- Distributed Operating Systems provide resource transparency to the user

How is it different ?

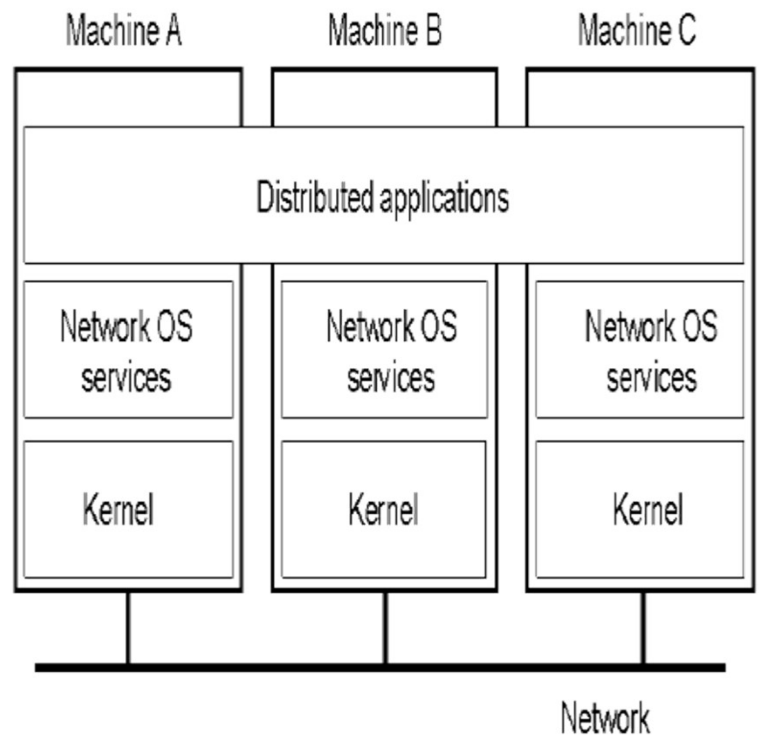
- The Distributed Operating System is unique and resides on different CPUs.
- User processes can run on any of the CPUs as allocated by the Distributed Operating System.
- Data can be resident on any machine that is the part of the Distributed System.
- All multi-machine systems are not Distributed Systems.
- It is the software not the hardware that determines whether a system is distributed or not.

Distributed OS vs. Network OS.

DOS



NOS



Distributed OS

- User is not aware of the multiple CPUs.
- Each machine runs a part of the Distributed Operating System.
- The system is fault-tolerant.

Network OS

- User is aware of the existence of multiple CPUs.

- Each machine has its own private Operating System.

- The system is not fault-tolerant.

Why Distributed Operating Systems ?

- Price/Performance advantage (Availability of cheap and powerful Microprocessors).
- Incremental growth.
- Reliability and Availability.
- Simplicity of Software (Theoretically).
- Provides Transparency.
- Creates another level of abstraction (e.g. Process creation).

Problems with Distributed Operating System

- Communication Protocol Overhead.
- Lack of Simplicity.
- High requirement of the degree of fault tolerance.
- Lack of global state information (e.g. No global Process Tables).
- Atomic Transactions.
- Process and Data Migration (e.g. During Load Balancing and Paging respectively).

Back to distributed Computing

Back to distributed computing……

Definition of a Distributed System

A distributed system consists of a collection of autonomous computers linked by a computer network and equipped with a distributed software enables computers to coordinate their activities and to share resources of the systems.

Definition of Distributed Computing

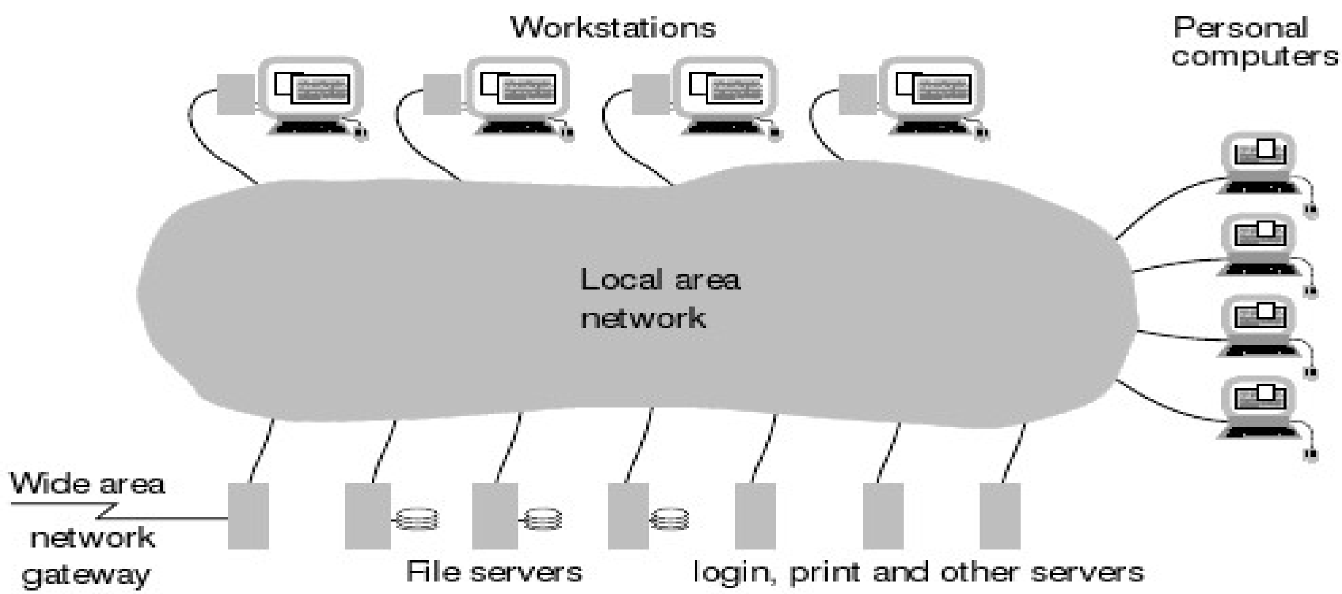
- Distributed computing is a computing concept that, in its most general sense, refers to multiple computer systems working on a single problem.

- In distributed computing, a single problem is divided into many parts, and each part is solved by different computers.

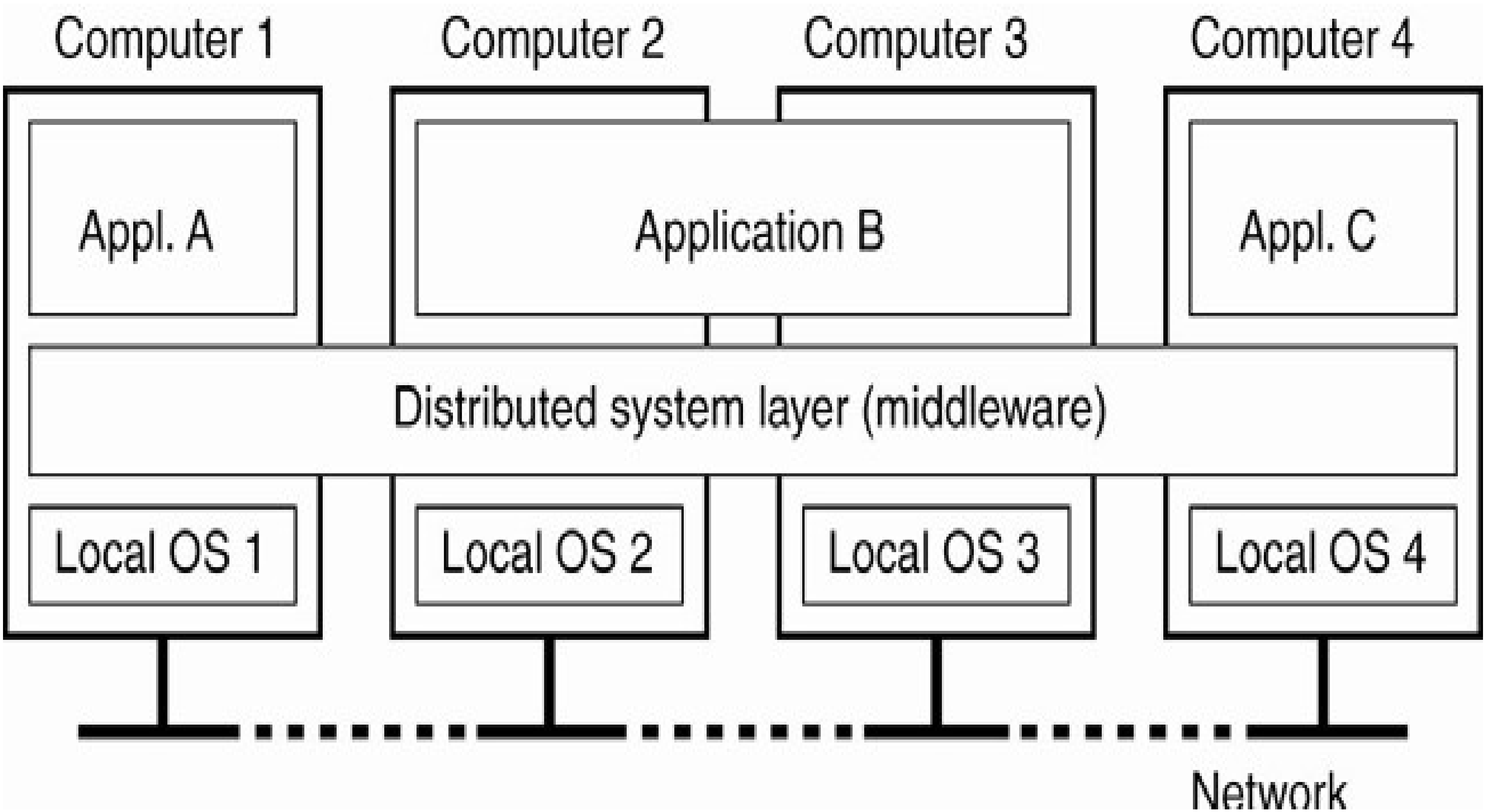
- As long as the computers are networked, they can communicate with each other to solve the problem. If done properly, the computers perform like a single entity.

A Distributed System

Figure 1.1 A simple distributed system.



A Distributed System



Characteristics of a DS/DC

- Resource sharing
- Heterogeneity
- Openness
- Concurrency
- Scalability
- Fault tolerance
- Transparency

(access, location,replication,failure, migration)

Transparency Issues

Transparency	Description
Access	Hide differences in data representation and how a resource is accessed
Location	Hide where a resource is located
Migration	Hide that a resource may move to another location
Relocation	Hide that a resource may be moved to another location while in use
Replication	Hide that a resource is replicated
Concurrency	Hide that a resource may be shared by several competitive users
Failure	Hide the failure and recovery of a resource

Transparency

- ***Access transparency*** enables local and remote resources to be accessed using identical operations.
- ***Location transparency*** enables resources to be accessed without knowledge of their physical or network location (for example, which building or IP address).
- ***Concurrency transparency*** enables several processes to operate concurrently using shared resources without interference between them.

- ***Replication transparency*** *enables multiple instances of resources to be used to* increase reliability and performance without knowledge of the replicas by users or application programmers.
- ***Failure transparency*** *enables the concealment of faults, allowing users and* application programs to complete their tasks despite the failure of hardware or software components.
- ***Mobility transparency*** *allows the movement of resources and clients within a system* without affecting the operation of users or programs.
- ***Performance transparency*** *allows the system to be reconfigured to improve* performance as loads vary.
- ***Scaling transparency*** *allows the system and applications to expand in scale without* change to the system structure or the application algorithms

Concurrent Computing / Parallel Computing/Distributed computing

- The terms "concurrent computing", "parallel computing", and "distributed computing" have much overlap, and no clear distinction exists between them.

- The same system may be characterized both as "parallel" and "distributed"; the processors in a typical distributed system run concurrently in parallel.

- Parallel computing may be seen as a particular tightly coupled form of distributed computing. Distributed computing may be seen as a loosely coupled form of parallel computing.

Examples of Distributed Systems/Distributed Computing Systems

- The Internet

- The Intranet

- Mobile devices

(telecommunication networks,network applications,real-time process control,parallel computation,peer-to-peer)

Goals of a Distributed System

- Performance
- Reliability
- Scalability
- Consistency
- Security

Challengesof a Distributed System

- Heterogeneity
- Openness
- Security
- Scalability
- Transparency
- Concurrency
- Data distribution
- *task distribution [HPC]*

Trends in Distributed Computing

- The emergence of pervasive networking technology.
- The emergence of ubiquitous computing coupled with the desire to support user mobility in distributed systems.
- The increasing demand for multimedia services.
- The view of distributed systems as a utility.

Systems models

Architectural Models (describing components ,and their relationship)

- Client server models
- Peer process models

Fundamental Models

- Interaction Model
- Failure Model
- Security Model

(formal description of properties common in all architectural models)

Architectural elements

- What are the entities
- How do they communicate, or, more specifically, what *communication paradigm* is used?
- What (potentially changing) roles and responsibilities do they have in the overall architecture?
- How are they mapped on to the physical distributed infrastructure (what is their *placement)?*

Communicating entities

- From a system perspective, entities are typically *processes*.
  - In some primitive environments, such as sensor networks, the entities are *nodes*.
  - In most distributed system environments, processes are supplemented **by *threads,*** that are the endpoints of communication

- *Objects:* Objects have been introduced to enable and encourage the use of object oriented approaches in distributed systems

- *Components:*

- *Web services:*

- **Communication paradigms**
  - Inter-process communication - low-level support for communication between processes in distributed systems by message passing.
  - Remote invocation - the most common communication paradigm in distributed systems.

- *Request-reply protocols - M*essages from client to server and then from server back to client, containing encoding operation and holding associated arguments.

- *Remote procedure calls* - The underlying RPC system then hides important aspects of distribution, including the encoding and decoding of parameters

- *Remote method invocation*

Architectural models

- Simplification by distinction between client, server and peer processes
  - Assessment of workload (responsibilities)
  - Failure impact analysis

- More dynamic systems can be built as variations on the client-server model
  - Moving code (from one process to another process)
    - Ex:: Client can download code from server & run it locally.
    - Code & objects are moved to reduce access delays & minimize comm. Traffic.

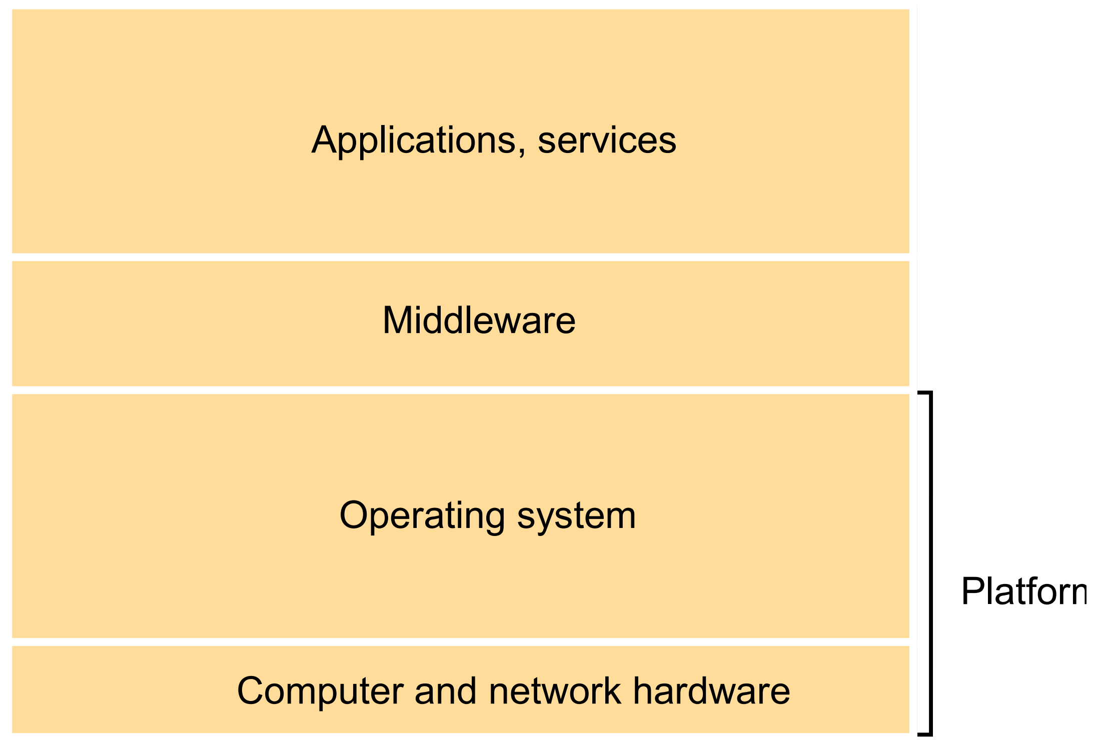
  - Adding/removing nodes or components
    - Discovery and advertisement of services

    - (Some DS enable comp & other mobile devices to be added or removed seamlessly, & allowed them to discover the available services & to offer their services to others.)

Service Layers

- Software architecture
  - Structuring software as layers or modules
  - Services offered and requested by processes on same or different computers (More recently)
- **Service layers::**

  A distributed service can be provided by one or more interacting server processes
  - Ex:: Network Time Protocol



Software and Hardware service layers in DS

Service Layers

- Middleware::
  - Masks heterogeneity & provide convenient prog. Model to application programmers.
  - Represented by processes or objects in a set of computers that interact with each other to implement communication & resource sharing,
  - To Implement communication and resource sharing
    - RMI
    - Group communication
  - Supplies application programming interface
  - **Provides**
    - Building blocks for software components

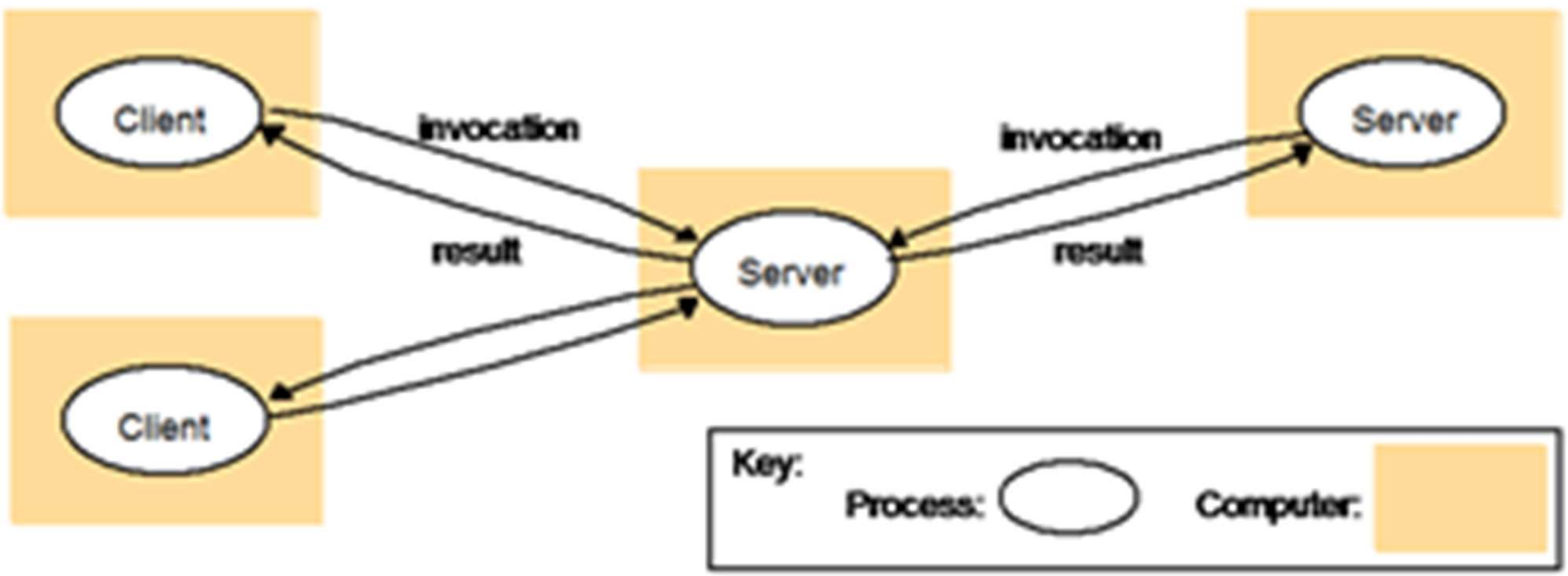
Service Layers: Middleware

- **Object oriented Middleware products & standards are::**
  - Examples:
    - Sun RPC
      - Simple, remote procedure calling package
    - CORBA(Common Object Request Broker Architecture)
      - CORBA is useful because it enables separate pieces of software written in different languages and running on different computers to work with each other like a single application or set of services.
      - SERVICES:--Namming,Security,event notification,persistant storage, transactions.
    - Microsoft DCOM(Distributed Component Object Model)
      - is a proprietary Microsoft technology for communication among software components distributed across networked computers.
    - Java RMI

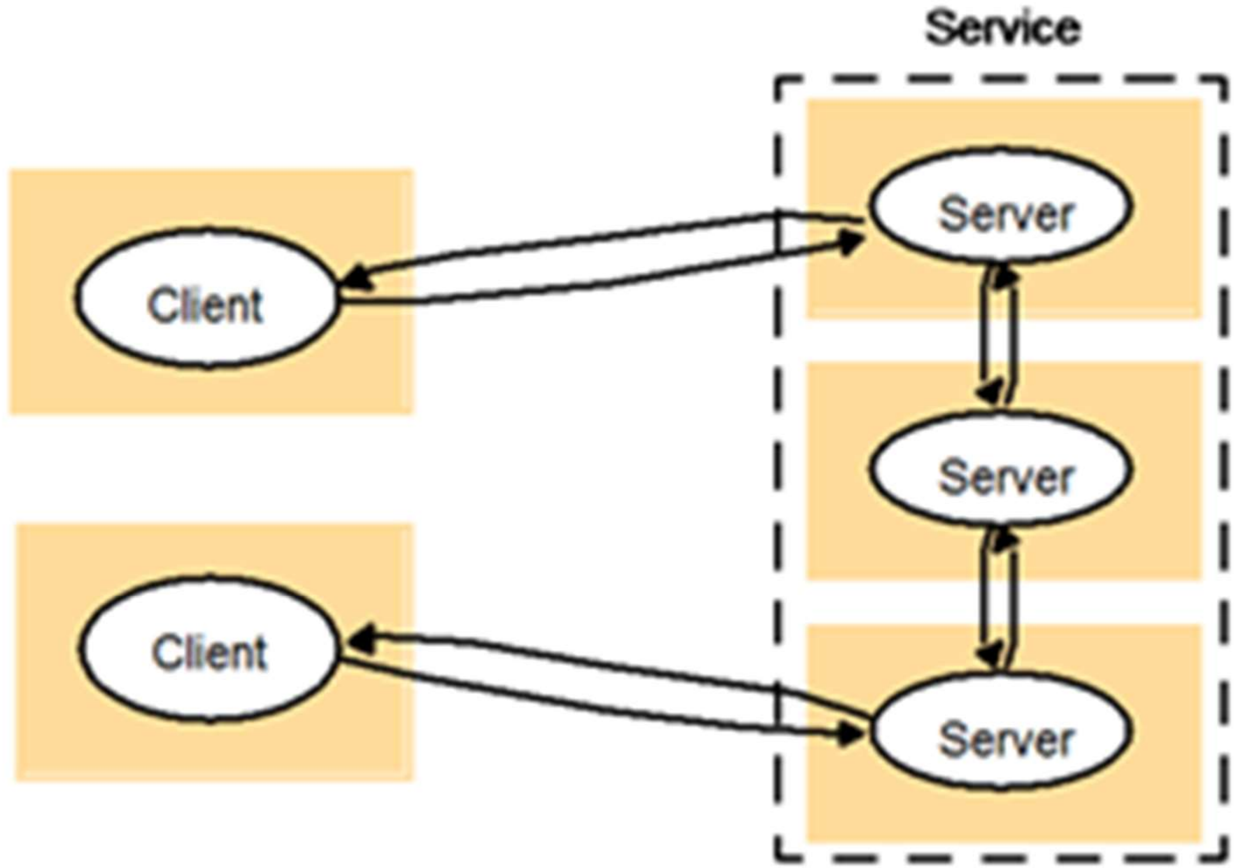
Service Layers: Middleware

- Middleware services
  - Naming
  - Security
  - Transactions
  - Persistent storage
  - Event notification
- Limitations
  - Many distributed applications rely entirely on the services provided by the available middleware to support their needs for communication & data sharing
    - TCP reliable for packets (basic error correction and detection)
    - Mail transfer service increase fault tolerance (maintain record of progress)
    - Some functions can only be correctly implemented with the help of application level information
      - **Correct behaviour in distributed programs depends upon checks, error-correction mechanisms & security mechanism at various levels**

Basic architectural models – client-server

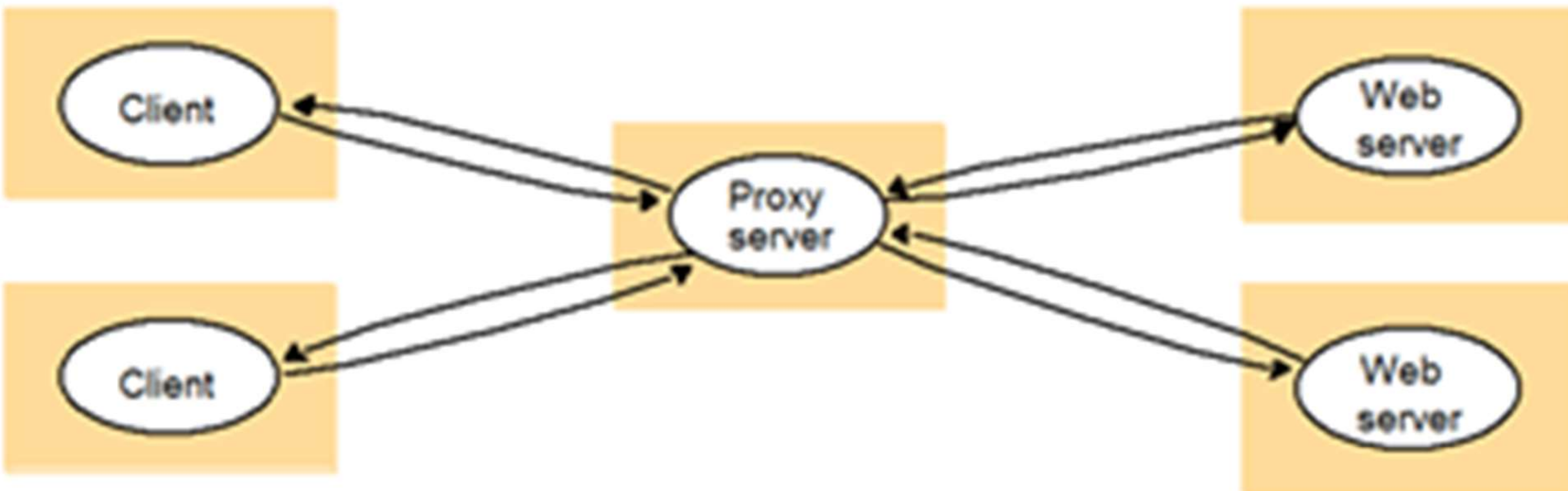


Basic architectural models – multiple servers



Partition set of objects or replicate set of objects

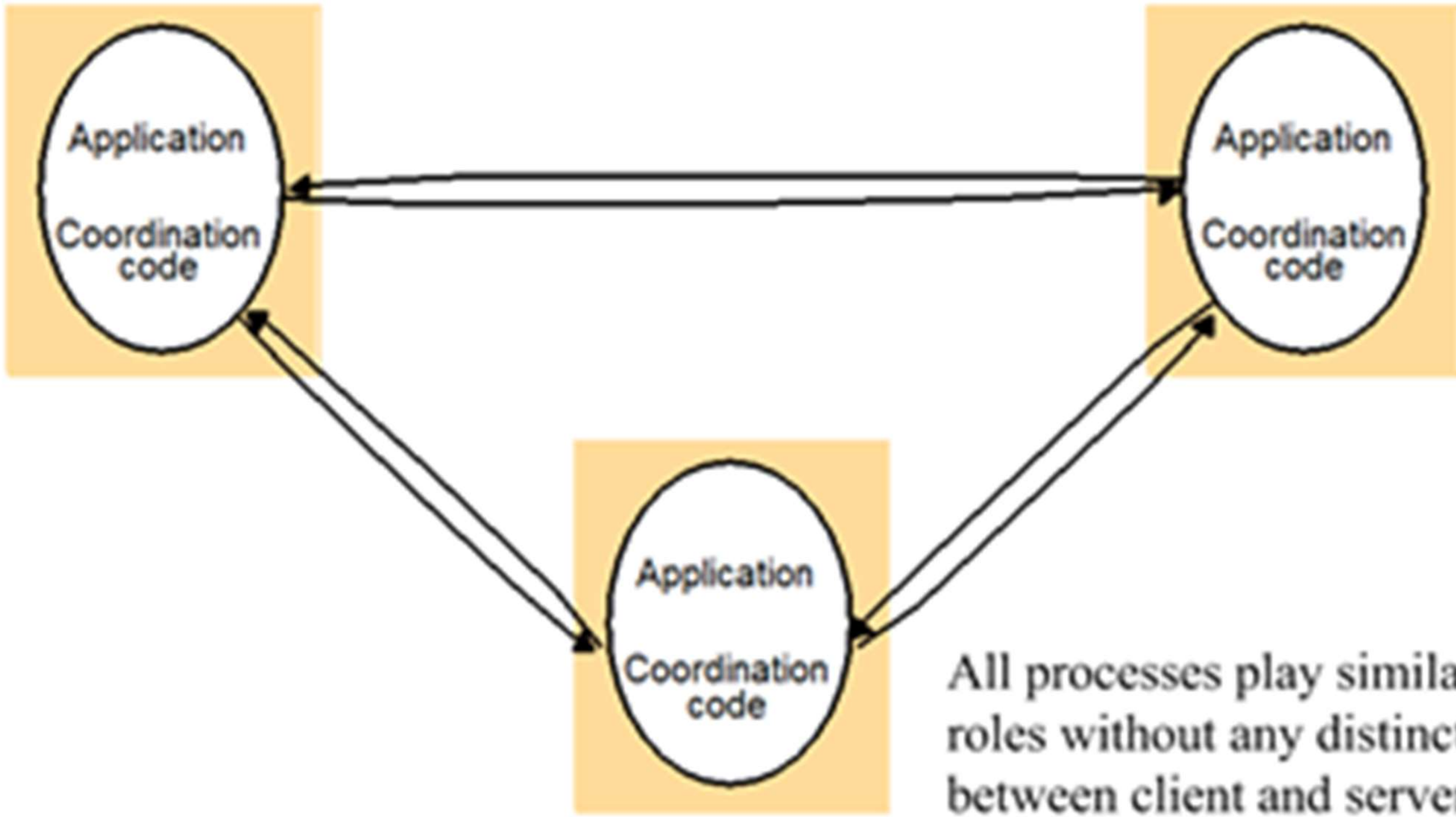
Basic architectural models – proxy server and caching



Cache: Store of recently used data objects closer than the objects themselves

Proxy servers increase performance and availability

Basic architectural models – peer processes



All processes play similar roles without any distinctio[n] between client and servers.

Variations on client server model

- Use of mobile code and mobile agents

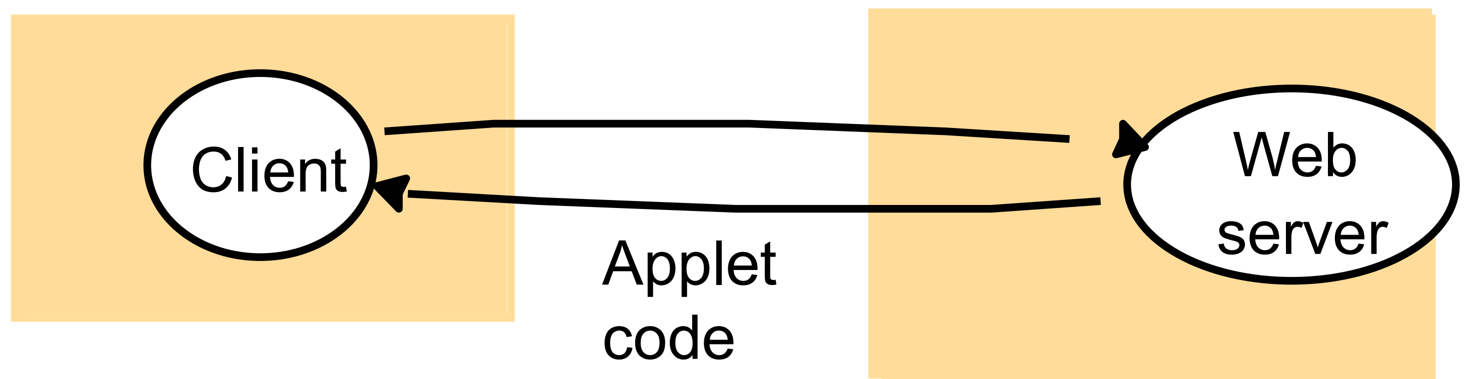
- To be able to add and remove mobile devices in a convenient manner.

Mobile code & Mobile agents

- **Mobile code**
- eg applets
- Only the code can be accessed
- **Mobile agents**
- It’s a running program that includes

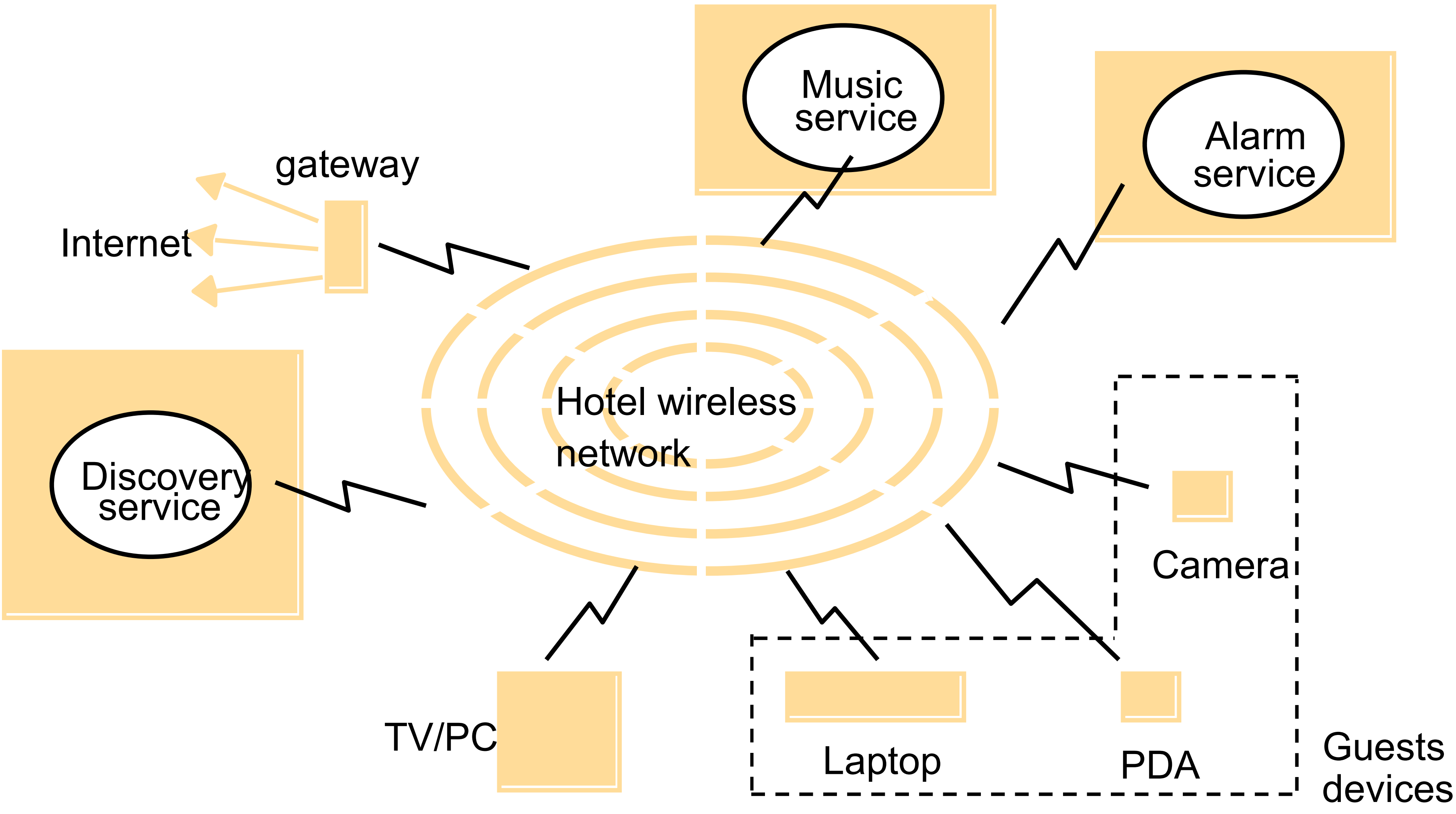
  Both code + data

Web applets



b) client interacts with the applet

Mobile Devices and Spontaneous networking



Mobile Devices and Spontaneous networking

- Key features
  - Easy connection to local network
    - Transparently reconfigured devices
  - Easy integration with local services
    - Autonomous service discoveries
- Issues
  - Internet addressing, and routing assumption
    - computers are at fixed locations
  - Limited connectivity
  - Security and privacy
- Discovery services
  - Registration service
  - Lookup service

Fundamental Models

- Interaction model
- Failure model
- Security model

Interaction model

- Performance of communication channels
- Computer clocks and timing events
- Two variants of the interaction model
- Event ordering

Performance of communication channels

- Communication performance is often a limiting characteristic.
- The delay between the sending of a message by one process and its receipt by another is referred to as latency.
- Bandwidth
- Jitter is the variation in the time taken to deliver a series of messages.

Computer Clock and Timing event

- It is impossible to maintain a single global notion of time.

- There are several approaches to correcting the times on computer clocks. (from GPS)

Two variants of the interaction model

- Synchronous distributed system
  - The time to execute each step f a process has known lower and uper 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.
- Asynchronous distributed system
  - No bound on process executiong speeds
  - No bound on message transmisson delays
  - No bound on clock drift rates.

Event ordering

- In many cases, we are interested in knowing whether an event (sending or receiving a message) at one process occurred before, after or concurrently with another event at another process.

Failure Model

- Omission failures
- Arbitrary failures
- Failure detection
- Impossibility of reaching agreement in the presence of failure
- Masking failure
- Reliability of one to one communication

- Arbitrary failures
  - a process may set wrong values in its data items, or it may return a wrong value in response to an invocation.
- Timing failures •
  - Timing failures are applicable in synchronous distributed systems where time limits are set on process execution time, message delivery time and clock drift rate.
- Masking failure
  - Each component in a distributed system is generally constructed from a collection of other components. It is possible to construct reliable services from components that exhibit failures. For example, multiple servers that hold replicas of data can continue to provide a service when one of them crashes

- Reliability of one to one communication
  - *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.

Byzantine or malicious failure

Byzantine or malicious failure, with authentication.

In this model,a process may exhibit any arbitrary behavior. However, if a faulty process claims to have received a specific message from a correct process, then that

claim can be verified using authentication, based on unforgeable signatures.

Byzantine or malicious failure.

In this model, a process may exhibit any arbitrary behavior and no authentication techniques are applicable to verify any claims made.

Security Model

- Protecting objects
- Securing processes and their interactions
- The enemy
- Defeating security threats
- Other possible threats from the enemy