

# Client – Server & Distributed System

---

A Basic Introduction

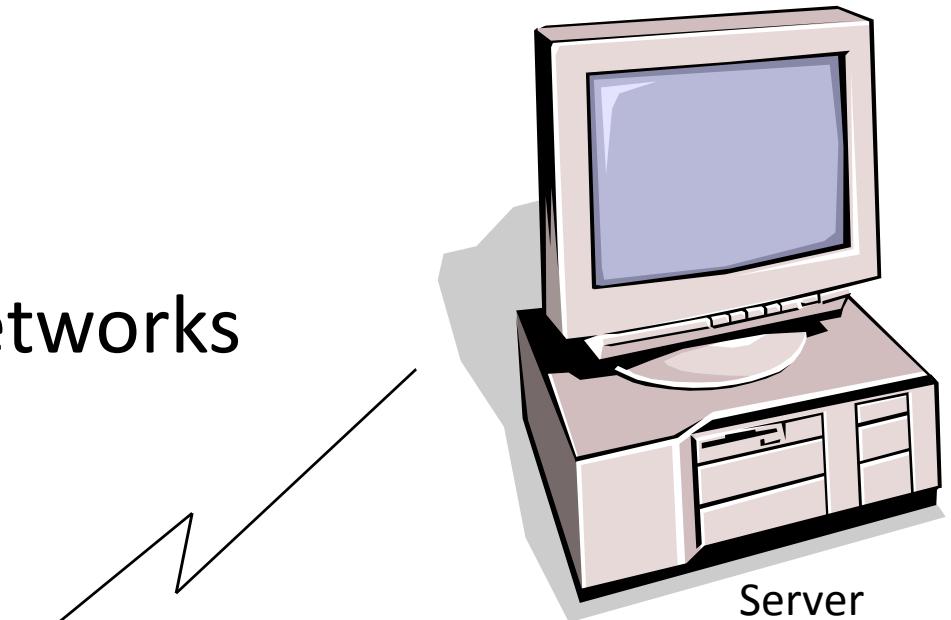
# Client Server Architecture

- A network architecture in which each computer or process on the network is either a *client* or a *server*.

Source: <http://webopedia.lycos.com>

# Components

- Clients
- Servers
- Communication Networks



# Clients

- Applications that run on computers
- Rely on servers for
  - Files
  - Devices
  - Processing power
- Example: E-mail client
  - An application that enables you to send and receive e-mail

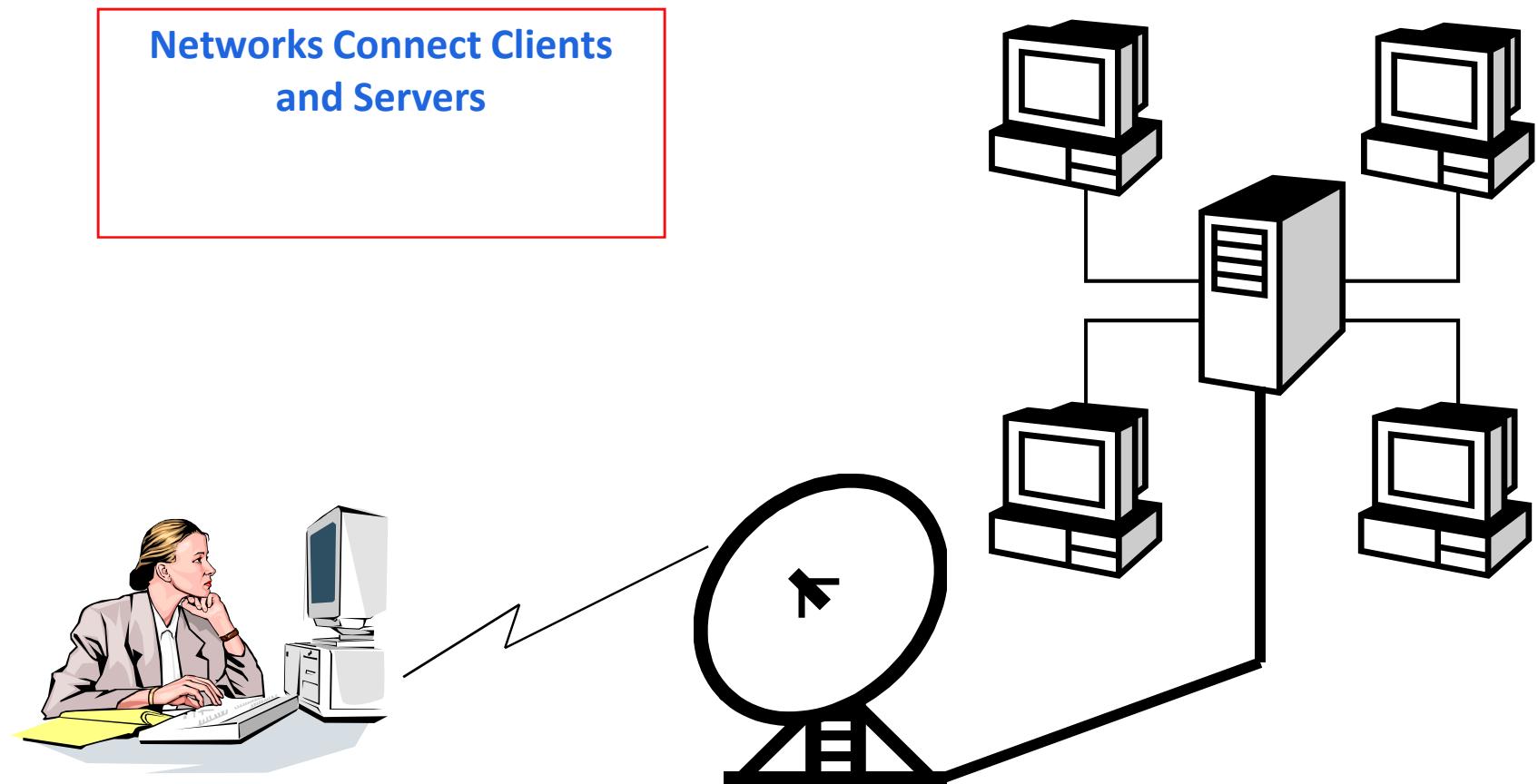
Clients are Applications

# Servers

- Computers or processes that manage network resources
  - Disk drives (file servers)
  - Printers (print servers)
  - Network traffic (network servers)
- Example: Database Server
  - A computer system that processes database queries

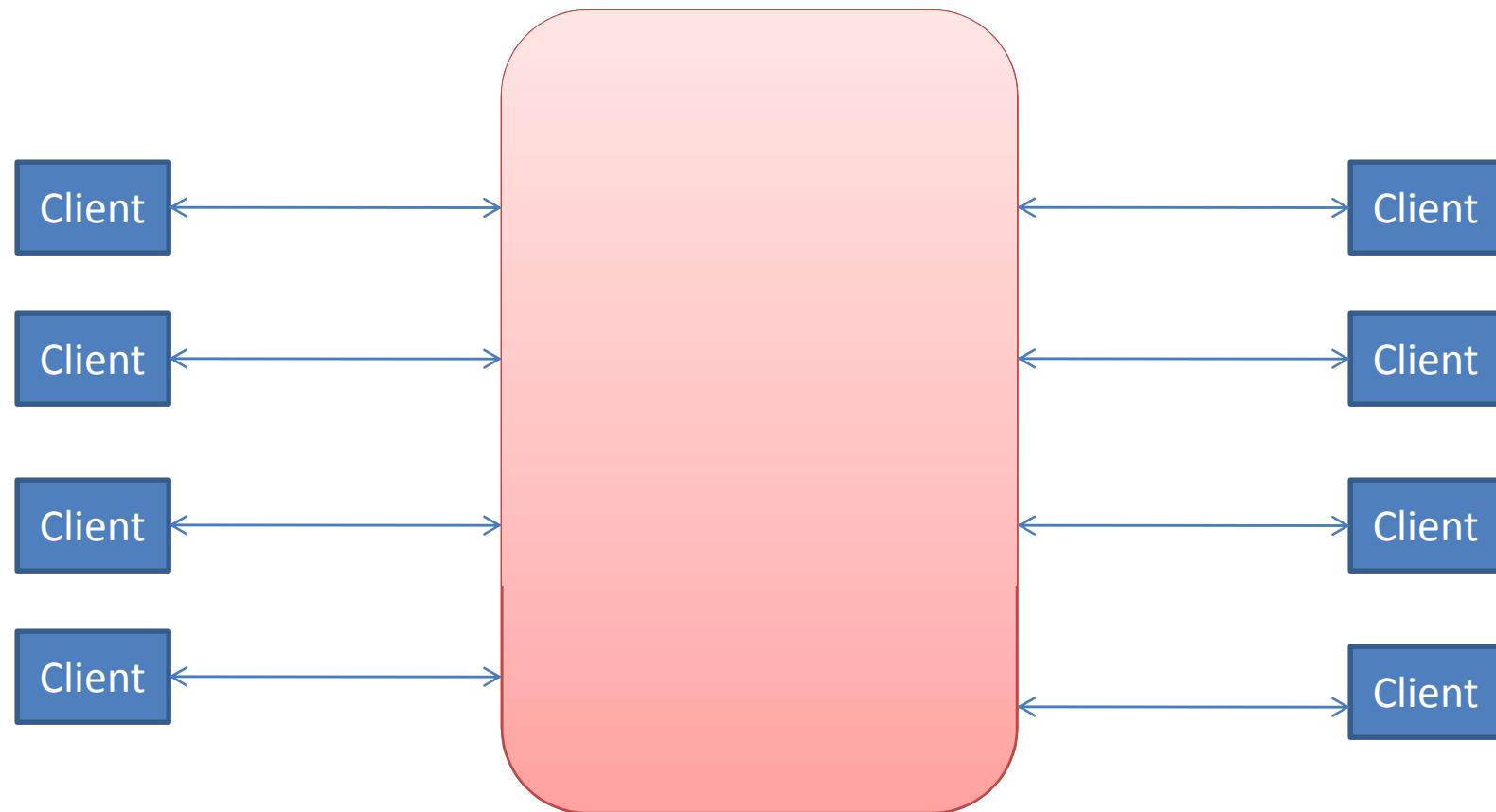
**Servers Manage Resources**

# Communication Networks



# Client–Server Computing

- The computing environment might consists of collection of equally powerful computers having same processor speed and equal amount of memory.
- The equal distribution of resources typically does not provide the best services to users.
- An alternative distribution of resources is to buy at least one machine much more powerful and have the other machines arranged so that users may connect to the more powerful machine when they need.



The client/server design provides users with a means to issue commands which are sent across a network to be received by a server which executes their commands for them. The results are then sent back to the client machine which sent the request in order that the user may see the results.

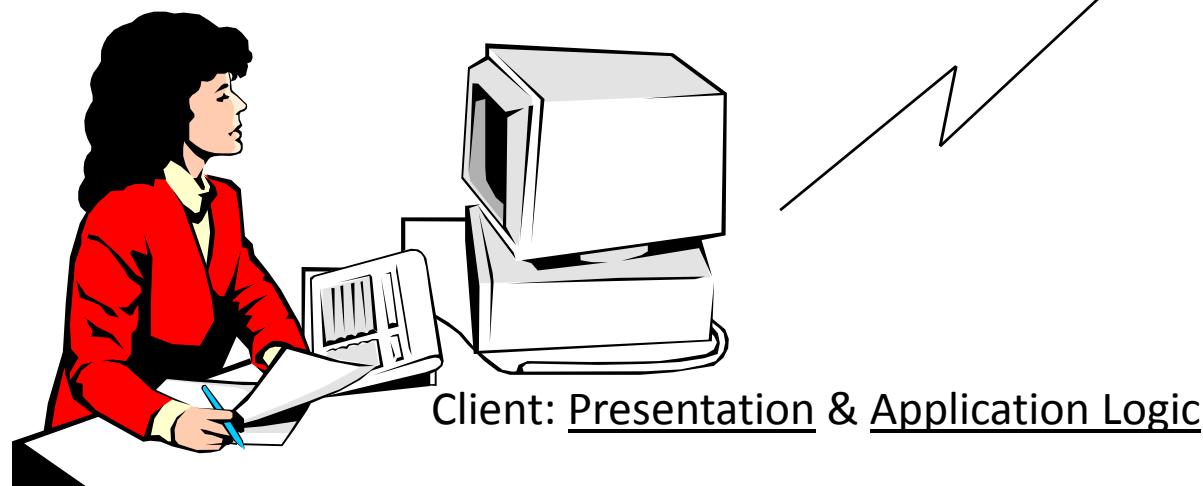
# Client–Server Computing

- Process takes place
  - on the server and
  - on the client
- Servers
  - Store and protect data
  - Process requests from clients
- Clients
  - Make requests
  - Format data on the desktop

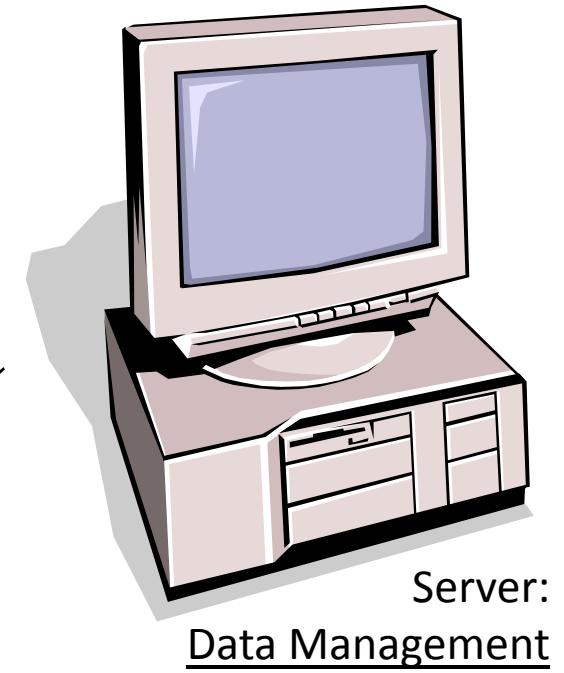
Client-Server Computing Optimizes  
Computing Resources

# Application Functions

- Software application functions are separated into three distinct parts

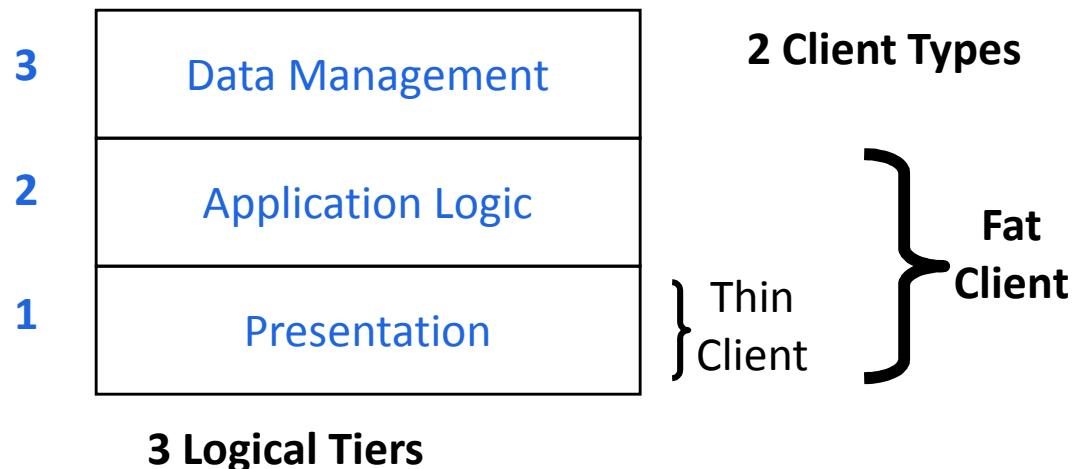


Client: Presentation & Application Logic



Server:  
Data Management

# Application Components



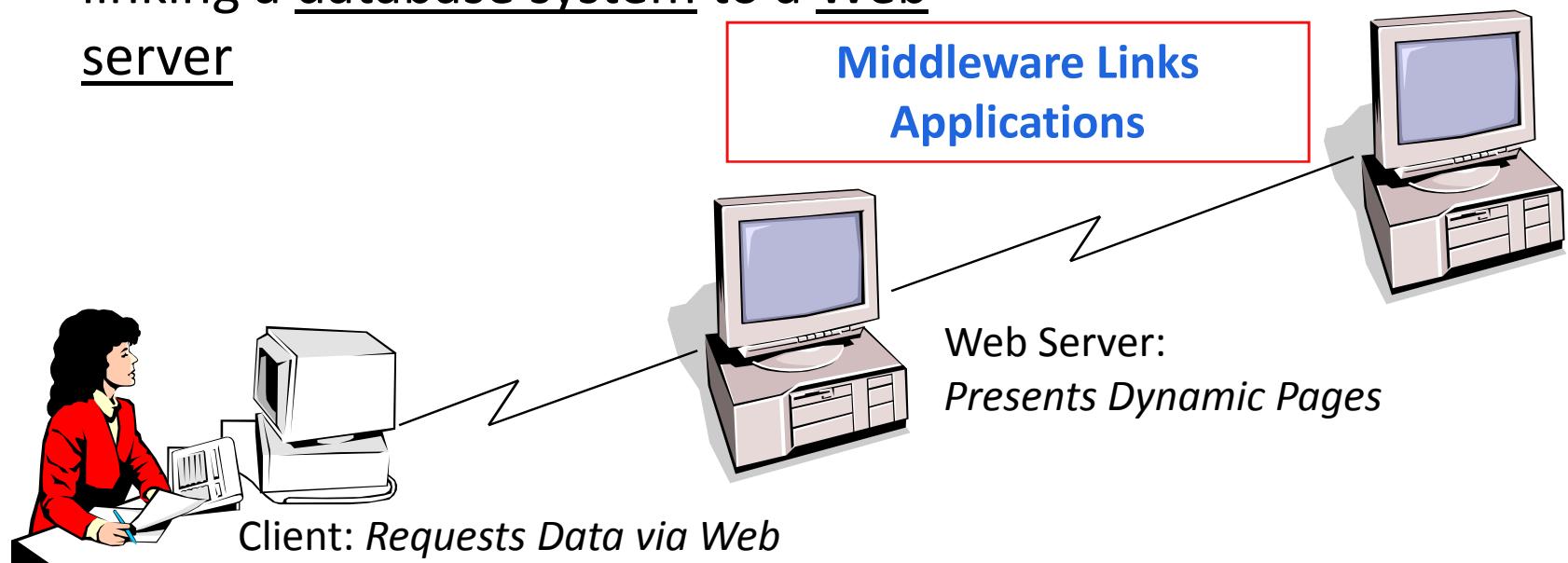
## Database Applications:

Most common use of client-server architectures

# Middleware

- Software that connects two otherwise separate applications
- Example: Middleware product linking a database system to a Web server

Database Server:  
*Manages Data*



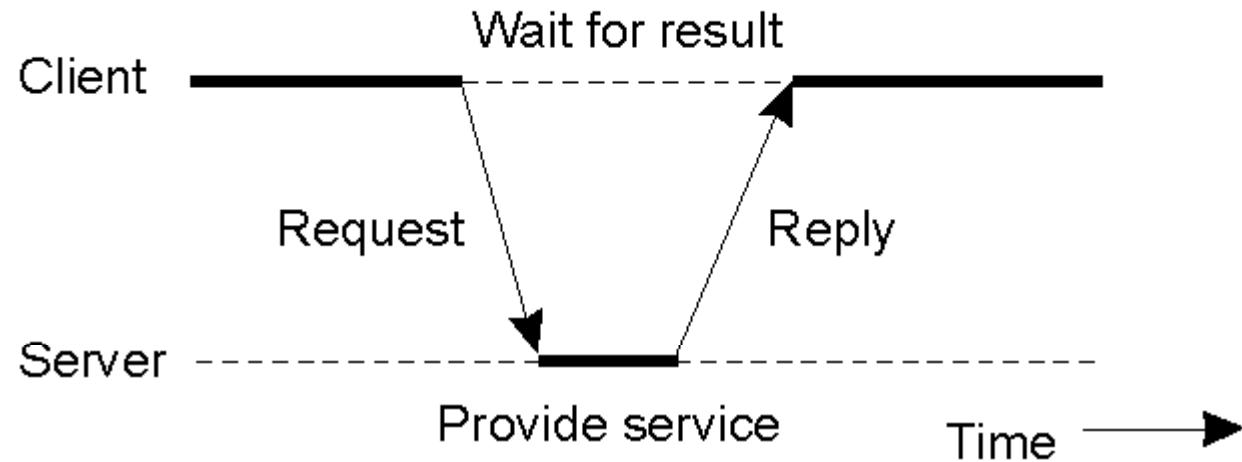
# Types of Servers

From A to Z

- Application Servers
- Audio/Video Servers
- Chat Servers
- Fax Servers
- FTP Servers
- Groupware Servers
- IRC Servers
- List Servers
- Mail Servers
- News Servers
- Proxy Servers
- Telnet Servers
- Web Servers

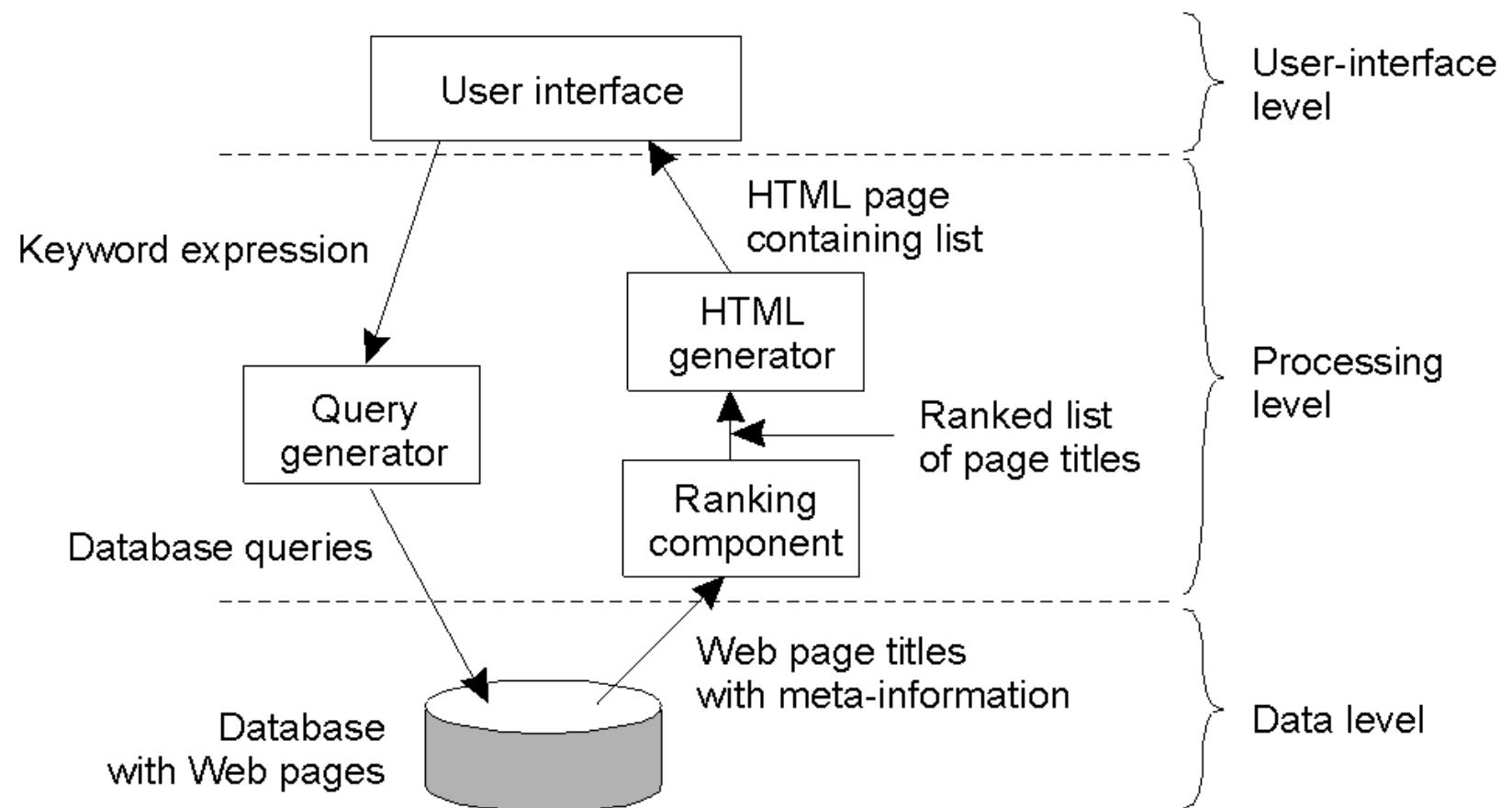
Source: <http://webopedia.lycos.com>

# Client-Server Model

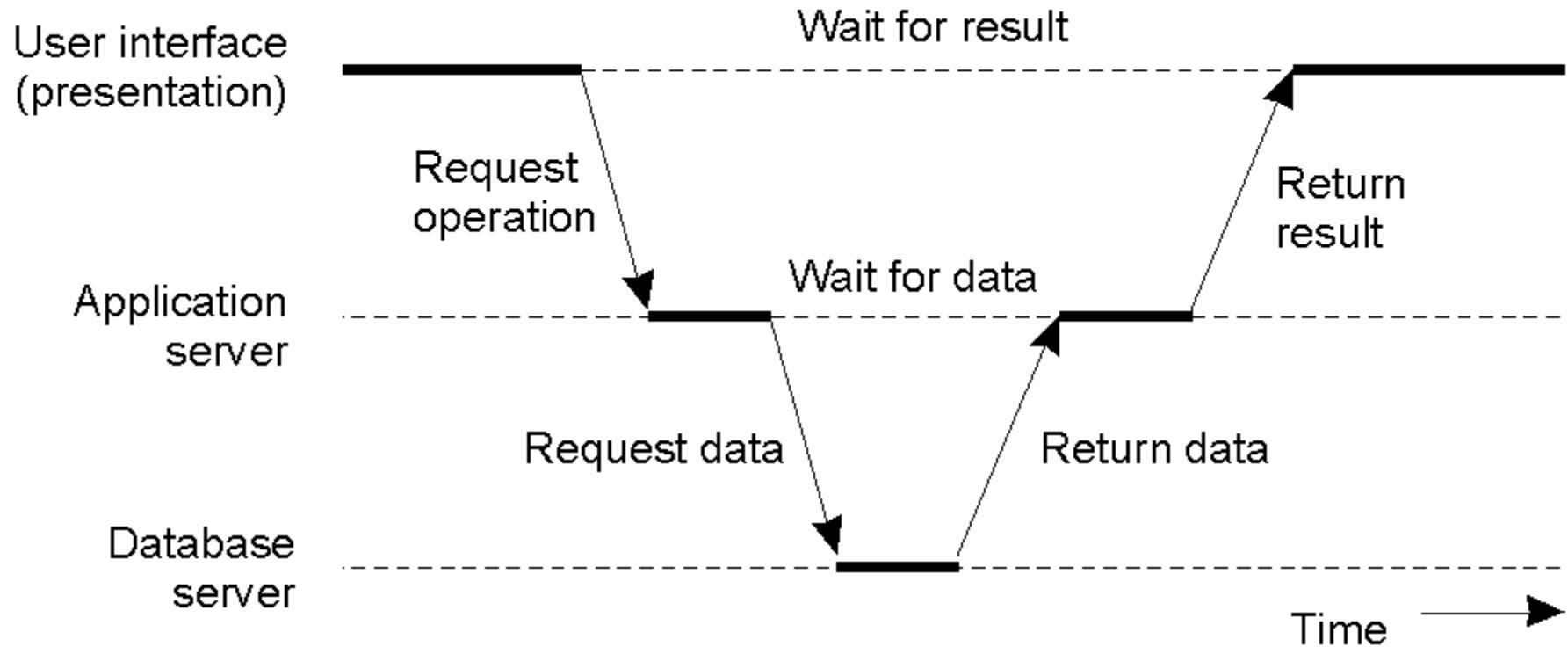


- Use TCP/IP for reliable network connection.
  - This implies the client must establish a connection before sending the first request.

# Internet Search Engine

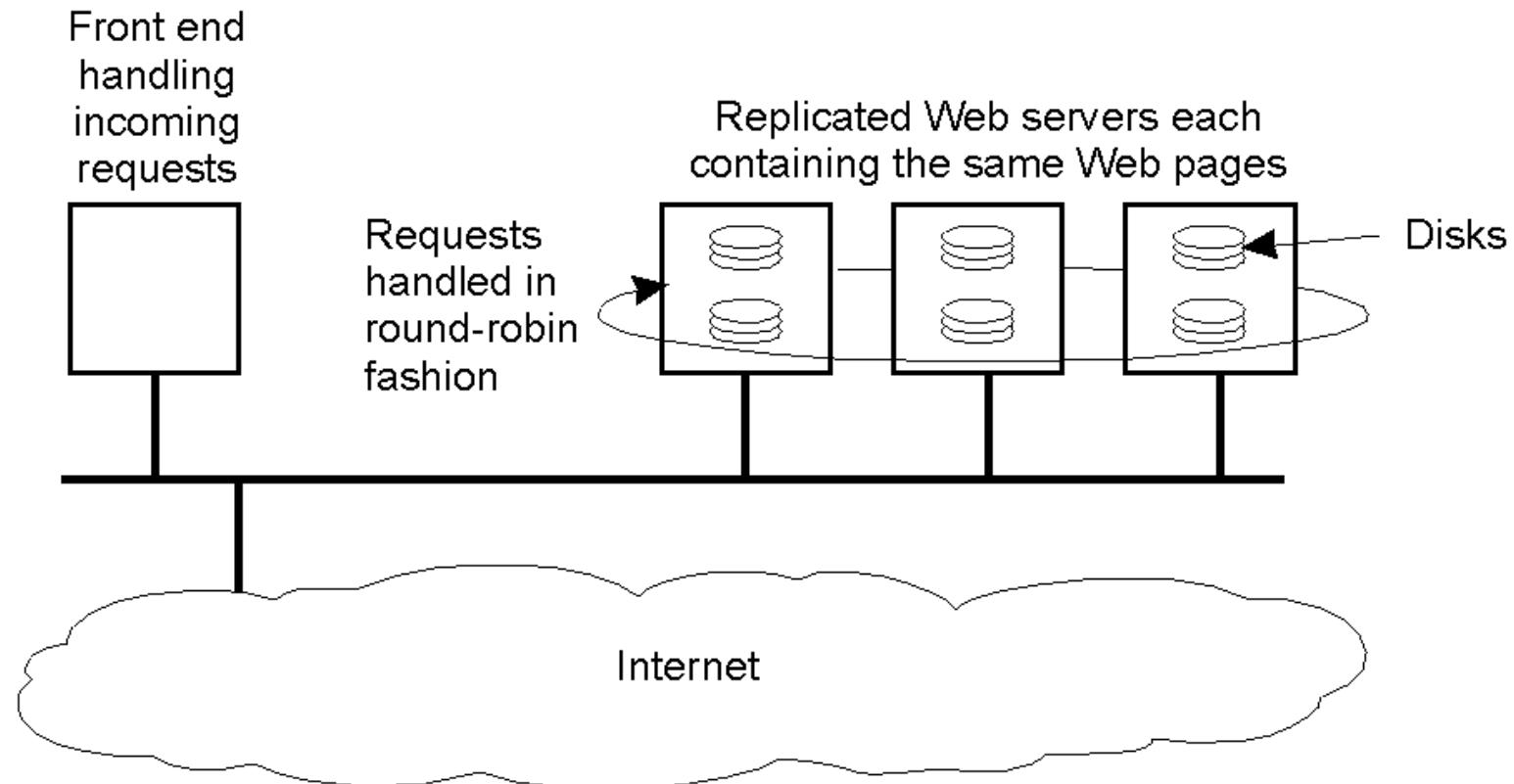


# Multitiered Architectures: 3 tiers



- Server may act as a client
  - Example would be transaction monitor across multiple databases

# Horizontal Distribution



- Distribute servers across nodes
  - E.g., Web server “farm” for load balancing
- Distribute clients in peer-to-peer systems.

# Introduction to Distributed Systems

- Why do we develop distributed systems?
  - availability of powerful yet cheap microprocessors (PCs, workstations), continuing advances in communication technology,
- **What is a distributed system?**
- A distributed system is a collection of independent computers that appear to the users of the system as a single system.
- Examples:
  - Network of workstations
  - Distributed manufacturing system (e.g., automated assembly line)
  - Network of branch office computers

- Definition: a *distributed system* is
  - A collection of independent computers that appears to its users as a single coherent system.
  - one that looks like an ordinary system to its users, but runs on a set of autonomous processing elements (PEs) where each PE has a separate physical memory space and the message transmission delay is not negligible.
  - There is close cooperation among these PEs. The system should support an arbitrary number of processes and dynamic extensions of PEs.

# Definition of Distributed System

- Distributed Systems encounter number of terminologies:
  - distributed, network, parallel, concurrent, and decentralized.
- Parallel means lockstep actions on a data set from a single thread of control.
- Distributed means that the cost or performance of a computation is governed by the communication of data and control.

- A system is centralized if its components are restricted to one site, decentralized if its components are at different sites with no or limited or close coordination
- When a decentralized system has no or limited coordination, it is called networked; otherwise, it is termed distributed indicating a close coordination among components at different sites.

# Advantages of Distributed Systems over Centralized Systems

- **Economics:** a collection of microprocessors offer a better price/performance than mainframes. Low price/performance ratio: cost effective way to increase computing power.
- **Speed:** a distributed system may have more total computing power than a mainframe. Ex. 10,000 CPU chips, each running at 50 MIPS. Not possible to build 500,000 MIPS single processor since it would require 0.002 nsec instruction cycle. Enhanced performance through load distributing.
- **Inherent distribution:** Some applications are inherently distributed. Ex. a supermarket chain.
- **Reliability:** If one machine crashes, the system as a whole can still survive. Higher availability and improved reliability.
- **Incremental growth:** Computing power can be added in small increments. Modular expandability
- **Another deriving force:** the existence of large number of personal computers, the need for people to collaborate and share information.

# Advantages of Distributed Systems over Independent PCs

- **Data sharing:** allow many users to access to a common data base
- **Resource Sharing:** expensive peripherals like color printers
- **Communication:** enhance human-to-human communication, e.g., email, chat
- **Flexibility:** spread the workload over the available machines

# Disadvantages of Distributed Systems

- **Software**: difficult to develop software for distributed systems
- **Network**: saturation, lossy transmissions
- **Security**: easy access also applies to secrete data
- **Distribution of control**
  - Hard to detect faults
  - Administration issues
- **Performance**
- Interconnect & servers must scale

# Goals of D.S.

- Transparency
- Openness
- Reliability
- Performance
- Scalability

## Design Challenges of Distributed Systems

---

- Designers of distributed systems need to take the following challenges into account:
  - Heterogeneity
    - ❖ Heterogeneous components must be able to interoperate.
  - Openness
    - ❖ Interfaces should allow components to be added or replaced.
  - Security
    - ❖ The system should only be used in the way intended.

# Design Challenges of Distributed Systems

---

## ➤ Scalability

- ❖ System should work efficiently with an increasing number of users.
- ❖ System performance should increase with inclusion of additional resources.

## ➤ Failure handling

- ❖ Failure of a component (partial failure) should not result in failure of the whole system.

## ➤ Transparency

- ❖ Distribution should be hidden from the user as much as possible

# Transparency

- How to achieve the single-system image, i.e how to make a collection of computers appear as a single computer.
- Hiding all the distribution from the users as well as the application programs can be achieved at two levels:
  - hide the distribution from users
  - At a lower level, make the system look transparent to programs.

# Forms of Transparency in a Distributed System

| <b>Transparency</b> | <b>Description</b>  |
|---------------------|---|
| 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 or is migrated to newer version         |
| Relocation          | Hide that a resource may be moved to another location while in use                        |
| Replication         | Hide that a resource may be shared by several competitive users                           |
| Concurrency         | Hide that a resource may be shared by several competitive users<br><small>accesse</small> |
| Failure             | Hide the failure and recovery of a resource   |
| Persistence         | Hide whether a (software) resource is in memory or on disk                                |

# Openness

- Make it easier to build and change
- The first step in openness is publishing the documentation of software components and interfaces of the components to make them available to software developers
- **Monolithic Kernel:** systems calls are trapped and executed by the kernel. All system calls are served by the kernel, e.g., UNIX.
- **Microkernel:** provides minimal services
  - IPC
  - some memory management
  - some low-level process management and scheduling
  - low-level i/o (E.g. multiple system interfaces.)

# Reliability

- Distributed system should be more reliable than a single system.
  - **Availability**: fraction of time the system is usable.
  - **Redundancy** improves it.
  - Need to maintain **consistency**
  - Need to be **secure**
  - **Fault tolerance**: need to mask failures, recover from errors.

# Performance

- Performance loss due to communication delays:
  - fine-grain parallelism: high degree of interaction
  - coarse-grain parallelism
  - (*Granularity is the extent to which a system is broken down into small parts, either the system itself or its description or observation*)
- Performance loss due to making the system fault tolerant

# Scalability

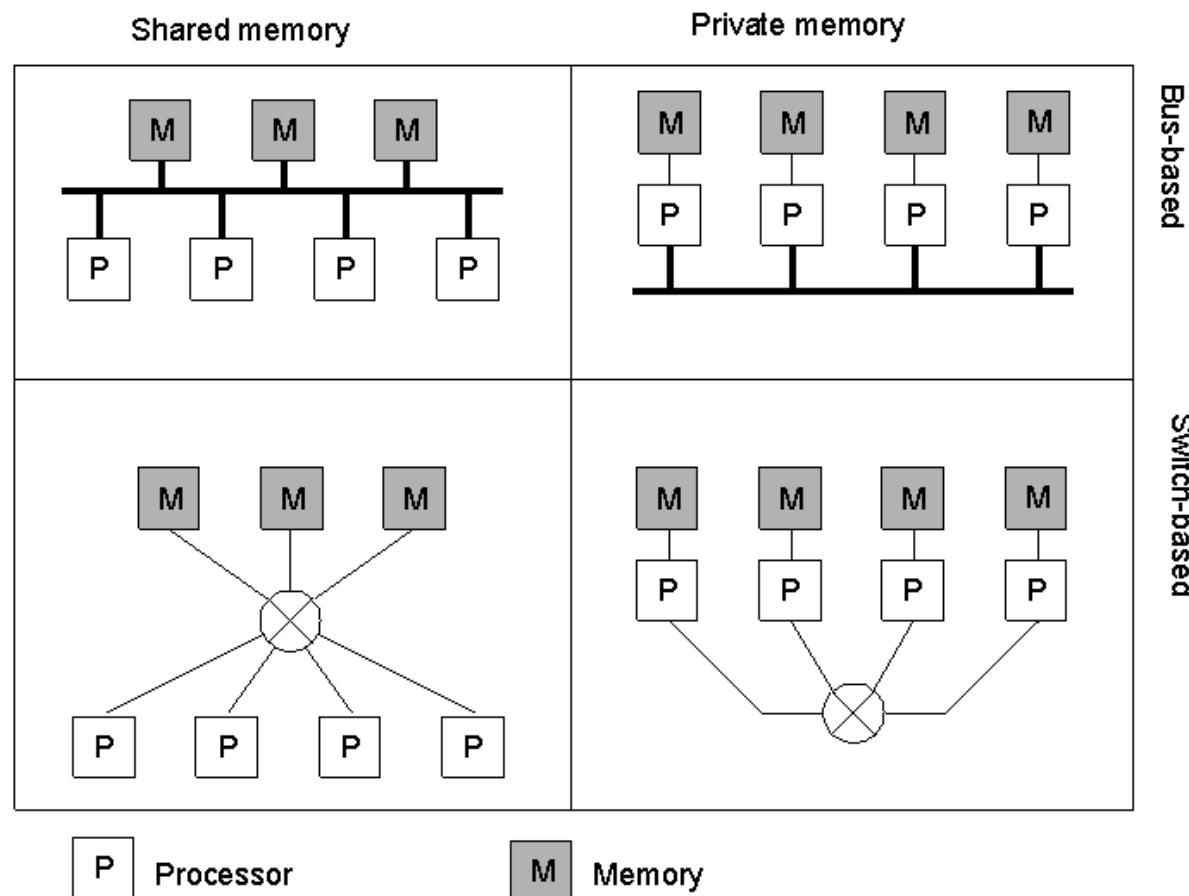
- System should work efficiently with an increasing number of users.
- System performance should increase with inclusion of additional resources
- Techniques that require resources linearly in terms of the size of the system are not **scalable**. (e.g., broadcast based query won't work for large distributed systems.)

# Pitfalls when Developing Distributed Systems

- False assumptions made by first time developer:
  - The network is reliable.
  - The network is secure.
  - The network is homogeneous.
  - The topology does not change.
  - Latency is zero.
  - Bandwidth is infinite.
  - Transport cost is zero.
  - There is one administrator.

# Hardware Concepts

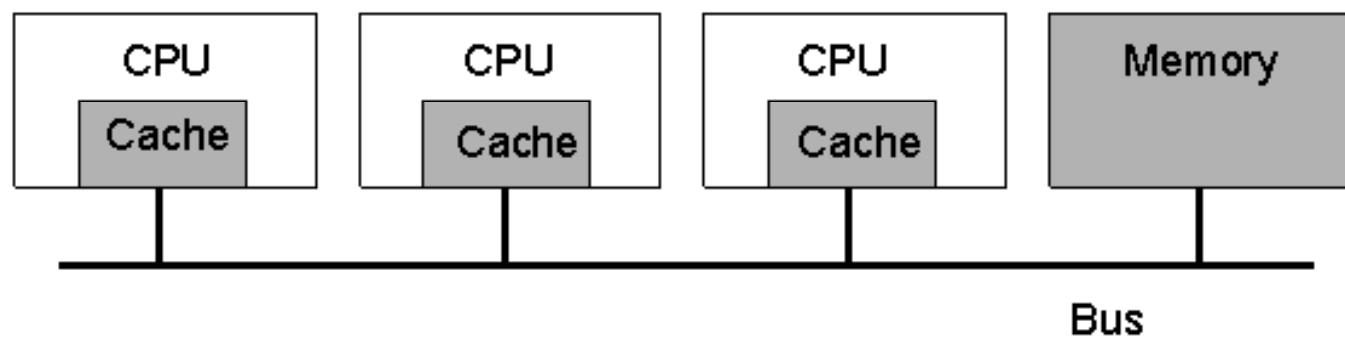
Basic organizations and memories in distributed computer systems



# Hardware Considerations

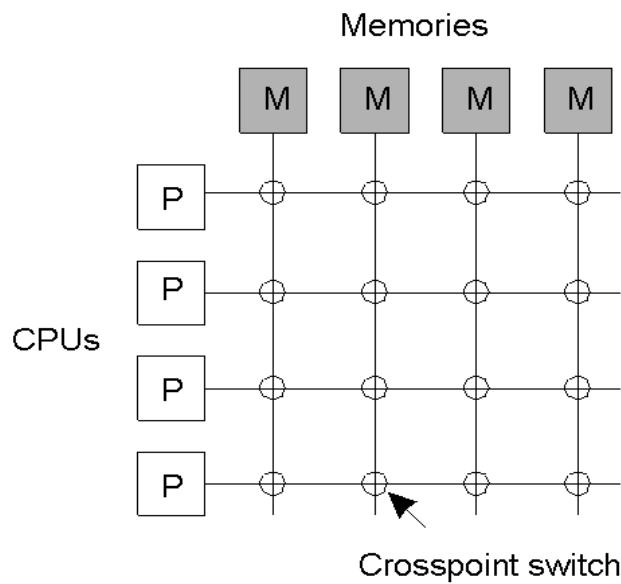
- General Classification:
  - **Multiprocessor** – a single address space among the processors
  - **Multicomputer** – each machine has its own private memory.
- OS can be developed for either type of environment.

# Multiprocessors



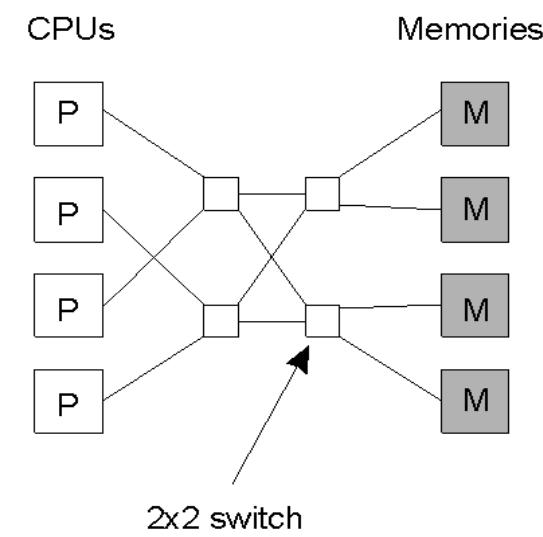
A bus-based multiprocessor

# Multiprocessors



(a)

A crossbar switch



(b)

An omega switching network

# Enslow's Model of DS

- Enslow (Scientist) proposed that distributed systems can be examined using three dimensions of hardware, control, and data.
- Distributed system = distributed **hardware** + distributed **control** + distributed **data**

- a system can be classified as a distributed system if all three categories (hardware, control, and data) reach a certain degree of decentralization.
- Several points in the dimension of hardware organization are as follows:
  - H1. A single CPU with one control unit.
  - H2. A single CPU with multiple ALUs (arithmetic and logic units). There is only one control unit

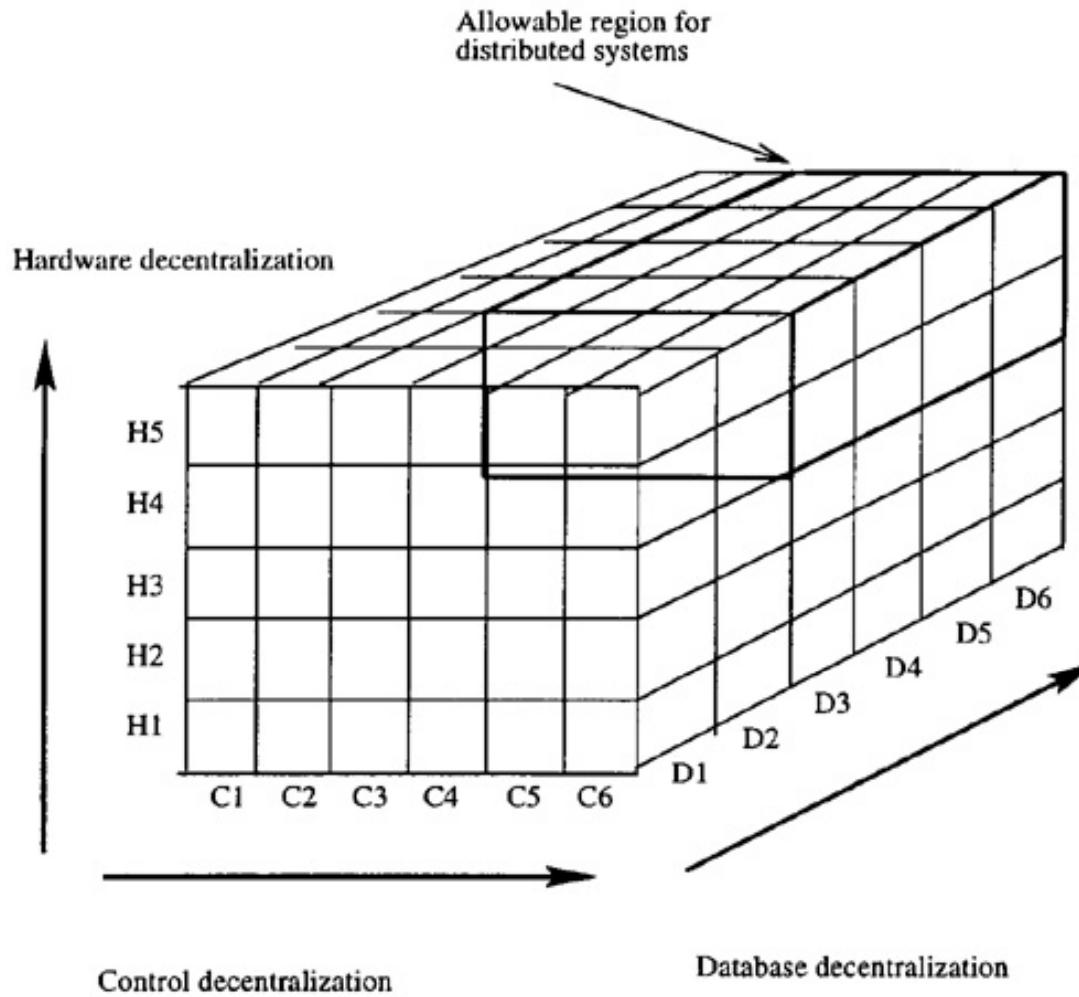
- H3. Separate specialized functional units, such as one CPU with one floating-point coprocessor.
  - H4. Multiprocessors with multiple CPUs but only one single I/O system and one global memory.
  - H5. Multicomputers with multiple CPUs, multiple I/O systems and local memories.
- 
- Similarly, points in the control dimension in order of increasing decentralization are the following:
    - C1. Single fixed control point. Note that physically the system may or may not have multiple CPUs.

- C2. Single dynamic control point. In multiple CPU cases the controller changes from time to time among CPUs.
- C3. A fixed master/slave structure. For example, in a system with one CPU and one coprocessor, the CPU is a fixed master and the coprocessor is a fixed slave.
- C4. A dynamic master/slave structure. The role of master/slave is modifiable by software.
- C5. Multiple homogeneous control points where copies of the same controller are used.
- C6. Multiple heterogeneous control points where different controllers are used

- The database has two components that can be distributed: files and a directory that keeps track of these files
- Distribution can be done in one of two ways, or a combination of both: replication and partition
- A database is partitioned if it is split into sub-databases and then each sub-database is assigned to different sites

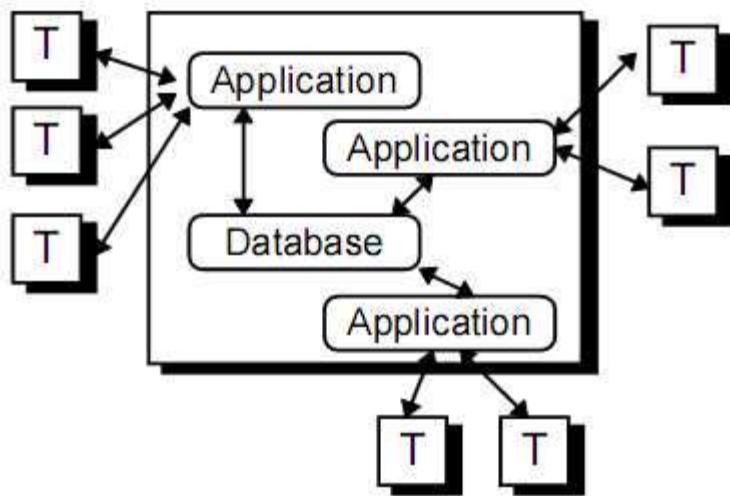
- D1. Centralized databases with a single copy of both files and directory.
  - D2. Distributed files with a single centralized directory and no local directory.
  - D3. Replicated database with a copy of files and a directory at each site.
- 
- D4. Partitioned database with a master that keeps a complete duplicate copy of all files.
  - D5. Partitioned database with a master that keeps only a complete directory.
  - D6. Partitioned database with no master file or directory.

- A system is a distributed one if it has:
  - Multiple processing elements (PEs).
  - Interconnection hardware.
  - Shared states.

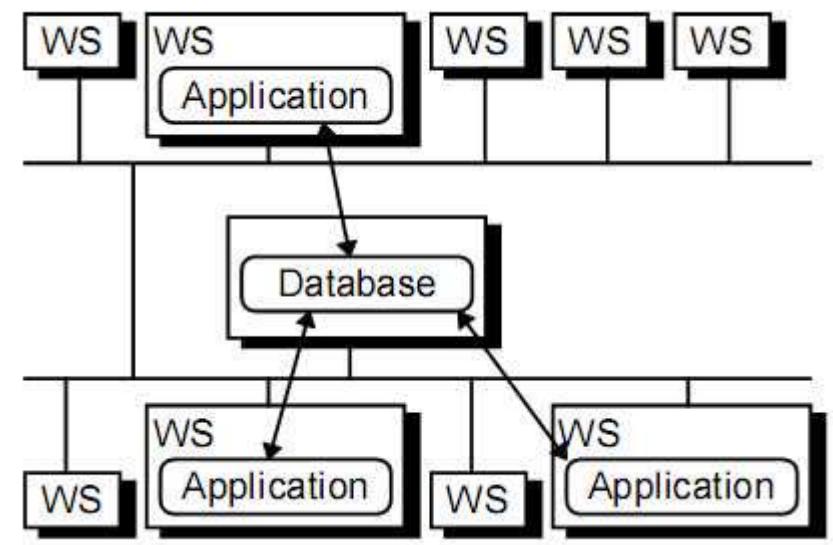


## Enslow's model of distributed systems

*(Some researchers also considered computer networks and parallel computers as part of distributed system)*



## Traditional Applications



## Distributed Applications

# Role of DCE

- Distributing applications requires the creation of a distributed environment in which they can run.
- Many vendors have already solved some of the relevant problems for their proprietary environments.
- OSF DCE provides a vendor-neutral solution
  - It's a platform for building distributed applications
  - It can support a range of commercial applications
  - It builds on work already done by vendors

# Requirement of Distributed Env.

- A supporting protocol for distributed Applications
- Mechanisms to exploit the environment's inherent parallelism.
- A way to locate distributed services, i.e., a directory service
- Security services, A mechanism for synchronizing the internal clocks of distributed systems.

# **Requirement .....**

- Support for simple systems:
  - Personal computers
  - Diskless system
- Optionally, some number of distributed applications such as:
  - A distributed file service
  - A network print service
  - Others

# Distribution Problems

- What approach should be used to distribute Applications?
  - Remote Procedure Call (RPC)
  - Message Passing TCP/IP
- What directory service(s) should be used?
  - A local directory must be fast and flexible
  - A global directory must be standard and widely supported

# Problems ....

- How should security be provided?
  - What services are needed?
  - What mechanisms should be used to provide those services ?
- What protocol should be used to synchronize clocks?
  - There are several possible choices
- How can simple systems be supported?
  - Provide special treatment for PCs and diskless workstations.
  - Alternatively, treat them like any other system in the distributed environment

# Problems...

- What distributed applications should be provided?
  - A distributed file service is essential
  - There are many other possibilities

# DCE Approach

- Distributing applications
  - Use remote procedure call (RPC)
- Allowing parallelism
  - Support a Threads package
- Directory Services
  - Use a Cell Directory Service for local lookups
  - Provide options for a global directory service

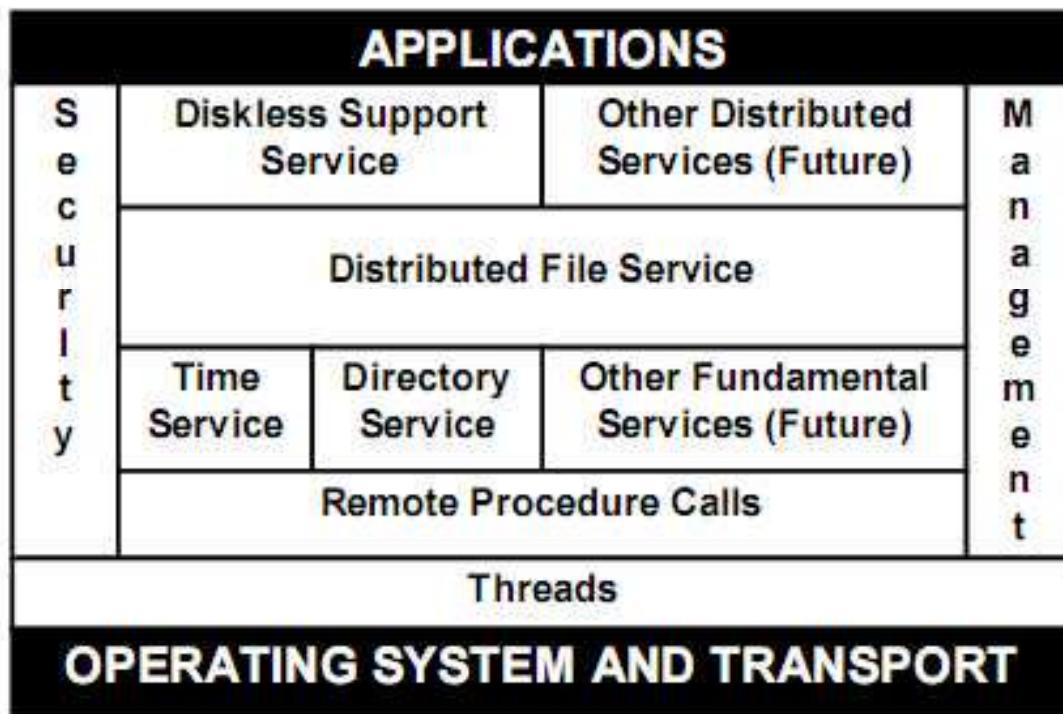
# DCE ...

- Security
  - Provide authentication, authorization, data integrity, and data privacy
- integrity, and data privacy
  - Use a Distributed Time Service (DTS)
  - Allow some integration with the widely used Network Time Protocol (NTP)

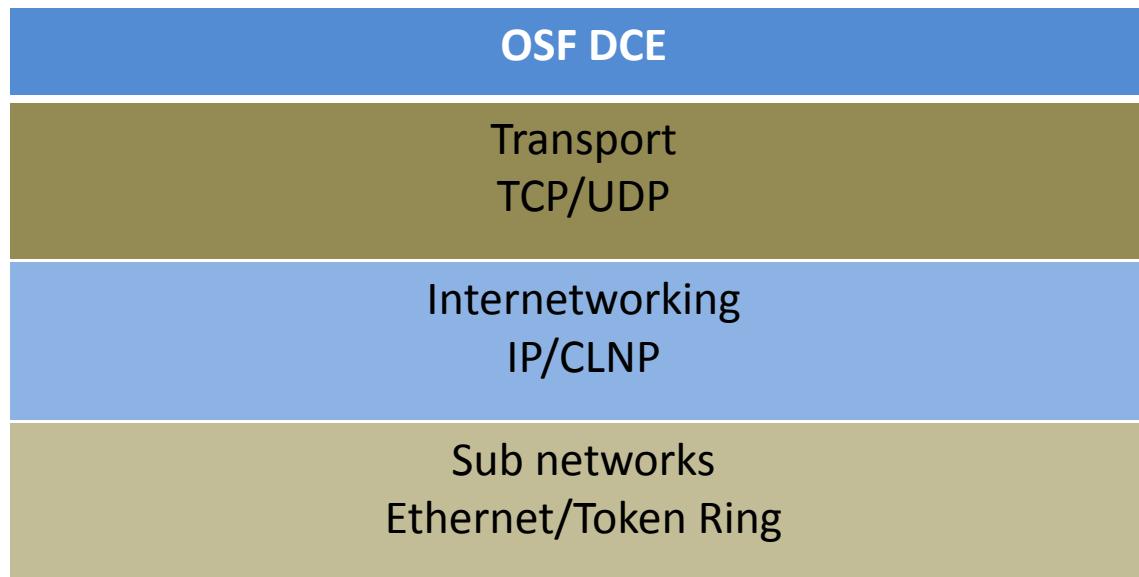
# **DCE ..**

- Simple Systems
  - Provide special services for diskless support
  - Treat PCs like any other system
- Distributed applications
  - Provide a Distributed File Service (DFS)
  - Allow the creation of a common distributed environment to encourage competition among application developers

# OSF DCE: A System View



# DCE & Distributed Computing



Open Source Foundation (OSF) DCE: A Layered View

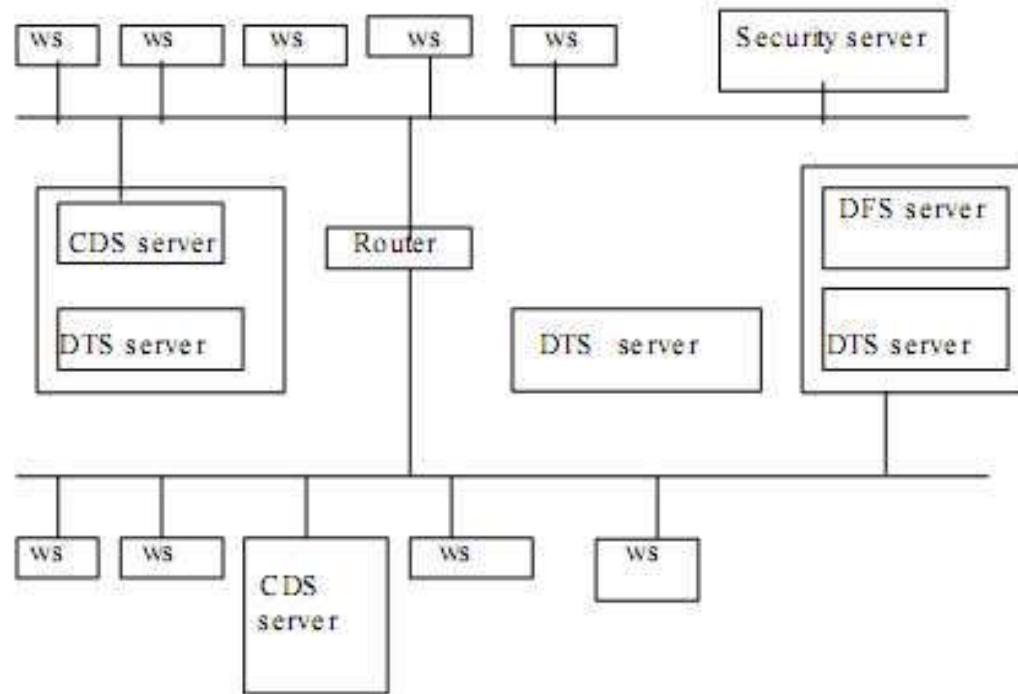
# DCE: Clients & Servers

- DCE relies on the notion of clients and servers
- Clients request services
- Servers provide services
- A single machine may support both the clients and servers
- A single process may act as both a client and a server at different times

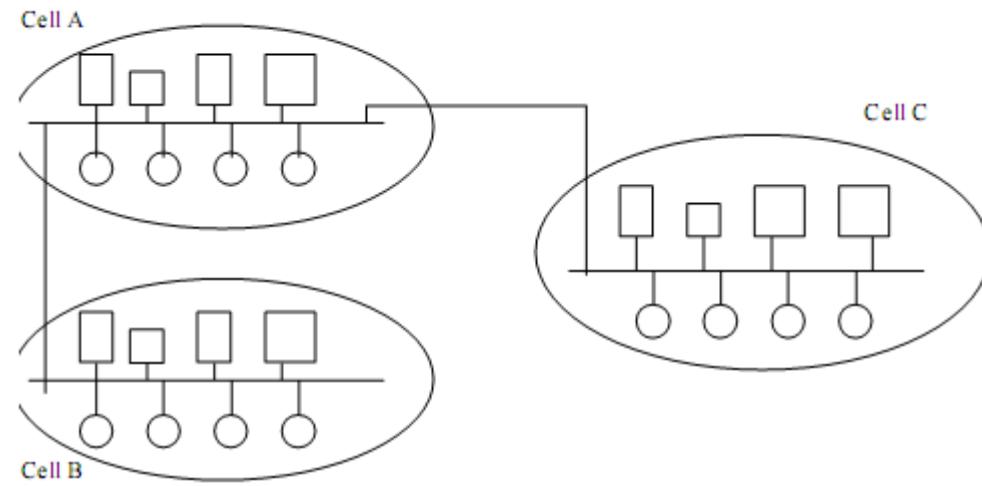
# cell

- Mostly clients perform most of their communication with only a few servers.
- In DCE, clients and servers that communicate mostly with one another are grouped into a cell
- A cell is an administrative unit
- Every machine belongs to one cell
- A cell may consists of two to thousand systems
- DCE optimizes intra-cell communication

# cells



# cells



## MODELS OF DISTRIBUTED SYSTEMS

### 1. Architectural Models

### 2. Interaction Models

### 3. Fault Models



Petru Eles, IDA, LiTH

## Basic Elements

- **Resources** in a distributed system are shared between **users**. They are normally encapsulated within one of the computers and can be accessed from other computers by communication.
- Each resource is managed by a program, the *resource manager*; it offers a communication interface enabling the resource to be accessed by its users.
- Resource managers can be in general modelled as *processes*.  
If the system is designed according to an object-oriented methodology, resources are encapsulated in *objects*.



Petru Eles, IDA, LiTH

## Architectural Models

How are responsibilities distributed between system components and how are these components placed?

- Client-server model
- Peer-to-peer

Variations of the above two:

- Proxy server
- Mobile code
- Mobile agents
- Network computers
- Thin clients
- Mobile devices



Petru Eles, IDA, LiTH

## Client - Server

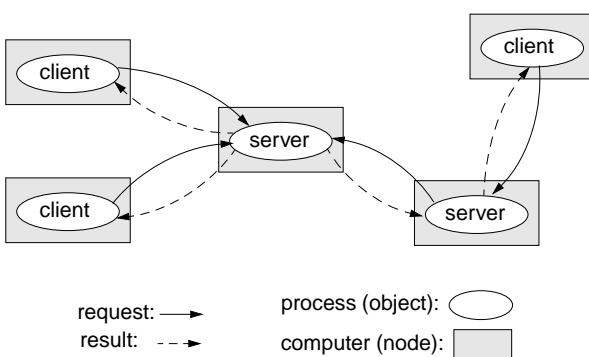
☞ The system is structured as a set of processes, called *servers*, that offer services to the users, called *clients*.

- The client-server model is usually based on a simple request/reply protocol, implemented with *send/receive* primitives or using *remote procedure calls* (RPC) or *remote method invocation* (RMI):
  - the client sends a request (invocation) message to the server asking for some service;
  - the server does the work and returns a result (e.g. the data requested) or an error code if the work could not be performed.



Petru Eles, IDA, LiTH

### Client - Server (cont'd)



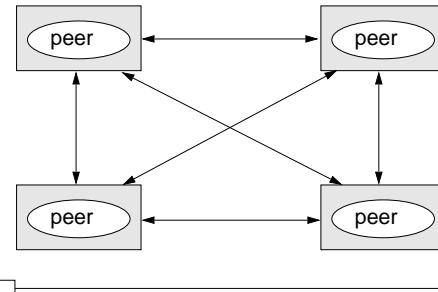
- A server can itself request services from other servers; thus, in this new relation, the server itself acts like a client.



Petru Eles, IDA, LiTH

### Peer-to-Peer

- ☞ All processes (objects) play similar role.
- Processes (objects) interact without particular distinction between clients and servers.
- The pattern of communication depends on the particular application.
- A large number of data objects are shared; any individual computer holds only a small part of the application database.
- Processing and communication loads for access to objects are distributed across many computers and access links.
- This is the most general and flexible model.



Petru Eles, IDA, LiTH

### Peer-to-Peer (cont'd)

- ☞ Some problems with client-server:

- Centralisation of service ⇒ poor scaling
  - Limitations:
    - capacity of server
    - bandwidth of network connecting the server

- ☞ Peer-to-Peer tries to solve some of the above

- It distributes shared resources widely



share computing and communication loads.

- ☞ Problems with peer-to-peer:

- High complexity due to
  - cleverly place individual objects
  - retrieve the objects
  - maintain potentially large number of replicas.



Petru Eles, IDA, LiTH

### Variations of the Basic Models

- ☞ Client-server and peer-to-peer can be considered as basic models.

- Several variations have been proposed, with considering factors such as:

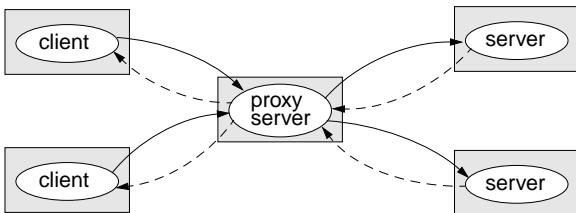
- multiple servers and caches
- mobile code and mobile agents
- low-cost computers at the users' side
- mobile devices



Petru Eles, IDA, LiTH

## Proxy Server

- A proxy server provides copies (replications) of resources which are managed by other servers.



- Proxy servers are typically used as caches for web resources. They maintain a cache of recently visited web pages or other resources. When a request is issued by a client, the proxy server is first checked, if the requested object (information item) is available there.
- Proxy servers can be located at each client, or can be shared by several clients.
- The purpose is to increase performance and availability, by avoiding frequent accesses to remote servers.



Petru Eles, IDA, LiTH

## Mobile Code

- Mobile code: code that is sent from one computer to another and run at the destination.

Advantage: remote invocations are replaced by local ones.

Typical example: Java applets.

Step 1: load applet



Step 2: interact with applet



Petru Eles, IDA, LiTH

## Mobile Agents

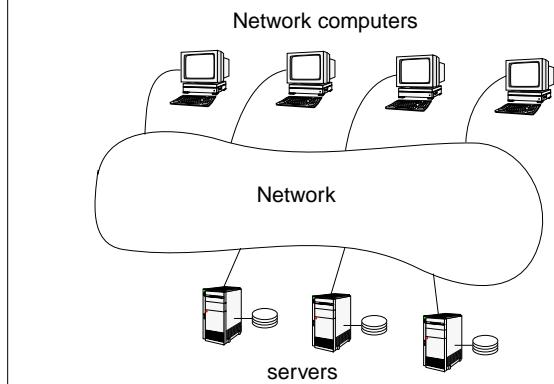
- Mobile agent: a running program that travels from one computer to another carrying out a task on someone's behalf.
- A mobile agent is a complete program, code + data, that can work (relatively) independently.
- The mobile agent can invoke local resources/data.

Typical tasks:

- Collect information
- Install/maintain software on computers
- Compare prices from various vendors by visiting their sites.

Attention: potential security risk (like mobile code)!

## Network Computers



- Network computers do not store locally operating system or application code. All code is loaded from the servers and run locally on the network computer.

Advantages:

- The network computer can be simpler, with limited capacity; it does not need even a local hard disk (if there exists one it is used to cache data or code).
- Users can log in from any computer.
- No user effort for software management/administration.



Petru Eles, IDA, LiTH

## **Thin Clients**

- ☞ The thin client is a further step, beyond the network computer:
  - Thin clients do not download code (operating system or application) from the server to run it locally. *All code is run on the server, in parallel for several clients.*
  - The thin client only runs the user interface!

**Advantages:**

- All those of network computers but the computer at the user side is even simpler (cheaper).

- ☞ Strong servers are needed!



Petru Eles, IDA, LiTH

## **Mobile Devices**

- ☞ Mobile devices are hardware, computing components that move (together with their software) between physical locations.
  - This is opposed to software agents, which are software components that migrate.
  - Both clients and servers can be mobile (clients more frequently).

**Particular problems/issues:**

- Mobility transparency: clients should not be aware if the server moves (e.g., the server keeps its Internet address even if it moves between networks).
- Problems due to variable connectivity and bandwidth.
- The device has to explore its environment:
  - Spontaneous interoperation: associations between devices (e.g. clients and servers) are dynamically created and destroyed.
  - Context awareness: available services are dependent on the physical environment in which the device is situated.



Petru Eles, IDA, LiTH

## **Interaction Models**

How do we handle time? Are there time limits on process execution, message delivery, and clock drifts?

- Synchronous distributed systems
- Asynchronous distributed systems



Petru Eles, IDA, LiTH

## **Synchronous Distributed Systems**

**Main features:**

- Lower and upper bounds on execution time of processes can be set.
- Transmitted messages are received within a known bounded time.
- Drift rates between local clocks have a known bound.

**Important consequences:**

1. In a synchronous distributed system there is a notion of global physical time (with a known relative precision depending on the drift rate).
2. Only synchronous distributed systems have a predictable behaviour in terms of timing. *Only such systems can be used for hard real-time applications.*
3. In a synchronous distributed system it is possible and safe to use timeouts in order to detect failures of a process or communication link.

- ☞ It is difficult and costly to implement synchronous distributed systems.



Petru Eles, IDA, LiTH

## Asynchronous Distributed Systems

- ☞ Many distributed systems (including those on the Internet) are asynchronous.
- No bound on process execution time (nothing can be assumed about speed, load, reliability of computers).
- No bound on message transmission delays (nothing can be assumed about speed, load, reliability of interconnections)
- No bounds on drift rates between local clocks.

Important consequences:

1. In an asynchronous distributed system there is no global physical time. Reasoning can be only in terms of logical time (see lecture on time and state).
2. Asynchronous distributed systems are unpredictable in terms of timing.
3. No timeouts can be used.



Petru Eles, IDA, LiTH

## Asynchronous Distributed Systems (cont'd)

- ☞ Asynchronous systems are widely and successfully used in practice.

In practice timeouts are used with asynchronous systems for failure detection. However, additional measures have to be applied in order to avoid duplicated messages, duplicated execution of operations, etc.



Petru Eles, IDA, LiTH

## Fault Models

- What kind of faults can occur and what are their effects?
- Omission faults
  - Arbitrary faults
  - Timing faults
- ☞ Faults can occur both in processes and communication channels. The reason can be both software and hardware faults.
  - ☞ Fault models are needed in order to build systems with predictable behaviour in case of faults (systems which are fault tolerant).
  - ☞ Of course, such a system will function according to the predictions, only as long as the real faults behave as defined by the "fault model". If not .....
  - ☞ These issues will be discussed in some of the following chapters and in particular in the chapter on "Recovery and Fault Tolerance".



Petru Eles, IDA, LiTH

## Omission Faults

- ☞ A processor or communication channel fails to perform actions it is supposed to do. This means that the particular action is not performed!
  - We do not have an omission fault if:
    - An action is delayed (regardless how long) but finally executed.
    - An action is executed with an erroneous result.
- ☞ With synchronous systems, omission faults can be detected by timeouts.
    - If we are sure that messages arrive, a timeout will indicate that the sending process has crashed. Such a system has a *fail-stop* behaviour.



Petru Eles, IDA, LiTH

## Arbitrary (Byzantine) Faults

- This is the most general and worst possible fault semantics.

Intended processing steps or communications are omitted or/and unintended ones are executed. Results may not come at all or may come but carry wrong values.

## Timing Faults

- Timing faults can occur in synchronous distributed systems, where time limits are set to process execution, communications, and clock drifts.

A timing fault occurs if any of this time limits is exceeded.



## Summary

- Models can be used to provide an abstract and simplified description of certain relevant aspects of distributed systems.
  - Architectural models define the way responsibilities are distributed among components and how they are placed in the system.
- We have studied three architectural models:
- Client-server model
  - Peer-to-peer
  - Several variations of the two

- Interaction models deal with how time is handled throughout the system.

Two interaction models have been introduced:

- Synchronous distributed systems
- Asynchronous distributed systems

- The fault model specifies what kind of faults can occur and what their effects are.

Fault models:

- Omission faults
- Arbitrary faults
- Timing faults

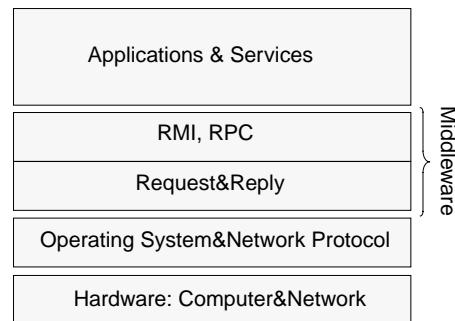


## COMMUNICATION IN DISTRIBUTED SYSTEMS

- Communication System: Layered Implementation**
- Network Protocol**
- Request and Reply Primitives**
- RMI and RPC**
- RMI and RPC Semantics and Failures**
- Group Communication**



## Communication Models and their Layered Implementation



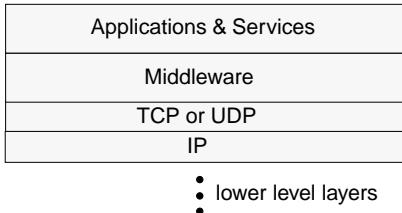
- This chapter concentrates on communication between distributed objects by means of two models: **remote method invocation (RMI)** and **remote procedure call (RPC)**.
- RMI, as well as RPC, are based on *request* and *reply* primitives.
- Request* and *reply* are implemented based on the network protocol (e.g. TCP or UDP in case of the Internet).



## Network Protocol

- Middleware and distributed applications have to be implemented on top of a network protocol. Such a protocol is implemented as several layers.

In case of the Internet:



- TCP (Transport Control Protocol) and UDP (User Datagram Protocol) are both transport protocols implemented on top of the Internet protocol (IP).



Petru Eles, IDA, LiTH

## Network Protocol (cont'd)

☞ TCP is a reliable protocol.

- TCP guarantees the delivery to the receiving process of all data delivered by the sending process, in the same order.
- TCP implements additional mechanisms on top of IP to meet reliability guarantees.
  - Sequencing:  
A sequence number is attached to each transmitted segment (packet). At the receiver side, no segment is delivered until all lower-numbered segments have been delivered.
  - Flow control:  
The sender takes care not to overwhelm the receiver (or intermediate nodes). This is based on periodic acknowledgements received by the sender from the receiver.
  - Retransmission and duplicate handling:  
If a segment is not acknowledged within a specified timeout, the sender retransmits it. Based on the sequence number, the receiver is able to detect and reject duplicates.
  - Buffering:  
Buffering is used to balance the flow between sender and receiver. If the receiving buffer is full, incoming segments are dropped. They will not be acknowledged and the sender will retransmit them.
  - Checksum:  
Each segment carries a checksum. If the received segment doesn't match the checksum, it is dropped (and will be retransmitted)



Petru Eles, IDA, LiTH

## Network Protocol (cont'd)

- ☞ UDP is a protocol that does not guarantee reliable transmission.
- UDP offers no guarantee of delivery.  
According to the IP, packets may be dropped because of congestion or network error. UDP adds no additional reliability mechanism to this.
- UDP provides a means of transmitting messages with minimal additional costs or transmission delays above those due to IP transmission.  
Its use is restricted to applications and services that do not require reliable delivery of messages.
- If reliable delivery is requested with UDP, reliability mechanisms have to be implemented at the application level.



Petru Eles, IDA, LiTH

## Request and Reply Primitives

☞ Communication between processes and objects in a distributed system is performed by message passing.

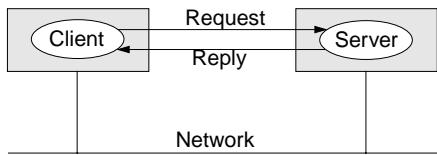
- In a typical scenario (e.g. client-server model) such a communication is through request and reply messages.



Petru Eles, IDA, LiTH

### Request-Reply Communication in a Client-Server Model

The system is structured as a group of processes (objects), called *servers*, that deliver services to *clients*.



The client:

```

send (request) to server_reference;
receive(reply);
  
```

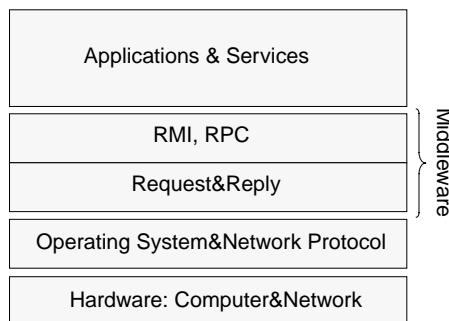
The server:

```

receive(request) from client-reference;
execute requested operation
send (reply) to client_reference;
  
```



### Remote Method Invocation (RMI) and Remote Procedure Call (RPC)



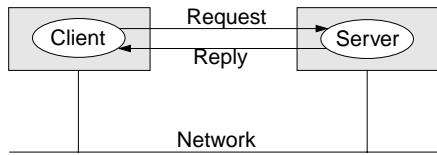
The goal: make, for the programmer, distributed computing look like centralized computing.

The solution:

- Asking for a service is solved by the client issuing a simple *method invocation or procedure call*; because the server can be on a remote machine this is a *remote invocation (call)*.
- *RMI (RPC) is transparent*: the calling object (procedure) is not aware that the called one is executing on a different machine, and vice versa.



### Remote Method Invocation



The client writes:

```

server_id.service(values_to_server, result_arguments);
  
```

The server contains the method:

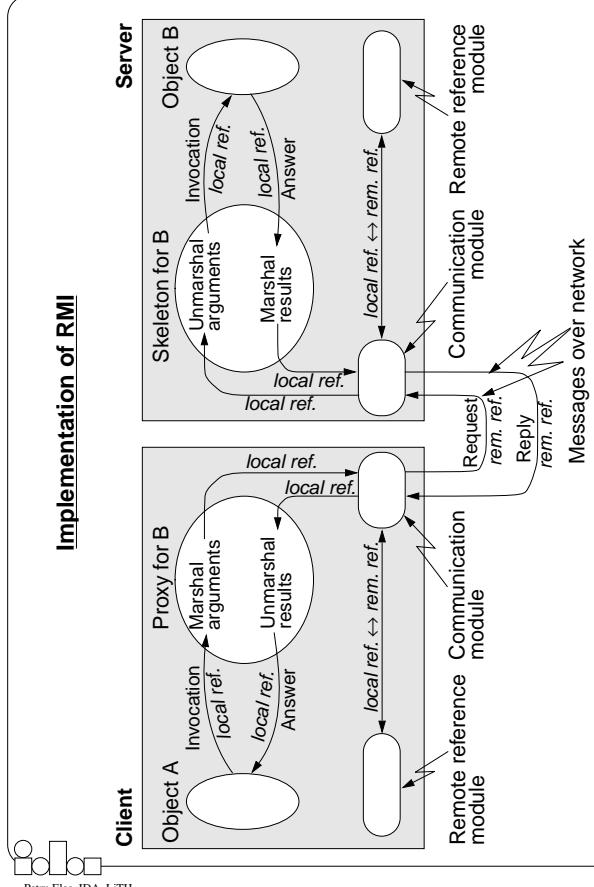
```

public service(in type1 arg_from_client; out type2 arg_to_client)
{ --- };
  
```

- The programmer is unaware of the request and reply messages which are sent over the network during execution of the RMI.



### Implementation of RMI



## Implementation of RMI (cont'd)

### Who are the players?

- Object A asks for a service
- Object B delivers the service

### Who more?

- The proxy for object B

- If an object A holds a remote reference to a (remote) object B, there exists a proxy object for B on the machine which hosts A. The proxy is created when the remote object reference is used for the first time. For each method in B there exists a corresponding method in the proxy.
- The proxy is the local representative of the remote object ⇒ the remote invocation from A to B is initially handled like a local one from A to the proxy for B.
- At invocation, the corresponding proxy method *marshals* the arguments and builds the message to be sent, as a request, to the server. After reception of the reply, the proxy *unmarshals* the received message and sends the results, in an answer, to the invoker.



## Implementation of RMI (cont'd)

### The skeleton for object B

- On the server side, there exists a skeleton object corresponding to a class, if an object of that class can be accessed by RMI. For each method in B there exists a corresponding method in the skeleton.
- The skeleton receives the request message, unmarshals it and invokes the corresponding method in the remote object; it waits for the result and marshals it into the message to be sent with the reply.
- A part of the skeleton is also called *dispatcher*. The dispatcher receives a request from the *communication module*, identifies the invoked method and directs the request to the corresponding method of the skeleton.



## Implementation of RMI (cont'd)

### Communication module

- The communication modules on the client and server are responsible of carrying out the exchange of messages which implement the request/reply protocol needed to execute the remote invocation.
- The particular messages exchanged and the way errors are handled, depends on the RMI semantics which is implemented (see slide 40).

### Remote reference module

- The remote reference module translates between local and remote object references. The correspondence between them is recorded in a *remote object table*.
- Remote object references are initially obtained by a client from a so called *binder* that is part of the global name service (it is not part of the remote reference module). Here servers register their remote objects and clients look up after services.



## Implementation of RMI (cont'd)

### Question 1

What if the two computers use different representation for data (integers, chars, floating point)?

- The most elegant and flexible solution is to have a standard representation used for all values sent through the network; the proxy and skeleton convert to/from this representation during marshalling/unmarshalling.

### Question 2

Who generates the classes for proxy and skeleton?

- In advanced middleware systems (e.g. CORBA) the classes for proxies and skeletons can be generated automatically. Given the specification of the server interface and the standard representations, an interface compiler can generate the classes for proxies and skeletons.



### Implementation of RMI (cont'd)

- ☞ Object A and Object B belong to the application.
- ☞ Remote reference module and communication module belong to the middleware.
- ☞ The proxy for B and the skeleton for B represent the so called *RMI software*. They are situated at the border between middleware and application and usually can be generated automatically with help of available tools that are delivered together with the middleware software.



Petru Eles, IDA, LiTH

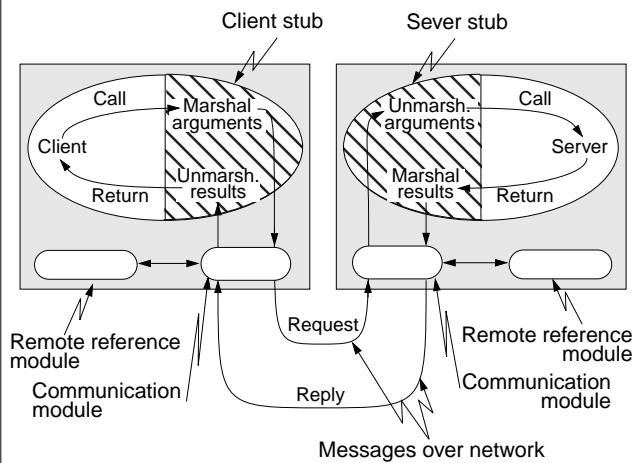
### The History of an RMI

1. The calling sequence in the client object activates the method in the proxy corresponding to the invoked method in B.
2. The method in the proxy packs the arguments into a message (marshalling) and forwards it to the communication module.
3. Based on the remote reference obtained from the remote reference module, the communication module initiates the request/reply protocol over the network.
4. The communication module on the server's machine receives the request. Based on the local reference received from the remote reference module the corresponding method in the skeleton for B is activated.
5. The skeleton method extracts the arguments from the received message (unmarshalling) and activates the corresponding method in the server object B.
6. After receiving the results from B, the method in the skeleton packs them into the message to be sent back (marshalling) and forwards this message to the communication module.
7. The communication module sends the reply, through the network, to the client's machine.
8. The communication module receives the reply and forwards it to the corresponding method in the proxy.
9. The proxy method extracts the results from the received message (unmarshalling) and forwards them to the client.



Petru Eles, IDA, LiTH

### Remote Procedure Call



Petru Eles, IDA, LiTH

### RMI Semantics and Failures

- If everything works OK, RMI behaves exactly like a local invocation. *What if certain failures occur?*

We consider the following classes of failures which have to be handled by an RMI protocol:

1. Lost request message
2. Lost reply message
3. Server crash
4. Client crash

☞ We will consider an *omission failure* model.  
This means:

- Messages are either lost or received correctly.
- Client or server processes either crash or execute correctly. After crash the server can possibly restart with or without loss of memory.



Petru Eles, IDA, LiTH

## Lost Request Messages

- The communication module starts a timer when sending the request; if the timer expires before a reply or acknowledgment comes back, the communication module sends the message again.

Problem: what if the request was not truly lost (but, for example, the server is too slow) and the server receives it more than once?

- We have to avoid that the server executes certain operations more than once.
- Messages have to be identified by an identifier and copies of the same message have to be filtered out:
  - If the duplicate arrives and the server has not yet sent the reply  $\Rightarrow$  simply send the reply.
  - If the duplicate arrives after the reply has been sent  $\Rightarrow$  the reply may have been lost or it didn't arrive in time (see next slide).



## Lost Reply Message

The client can not really distinguish the loss of a request from that of a reply; it simply resends the request because no answer has been received in the right time.

- If the reply really got lost, when the duplicate request arrives at the server it already has executed the operation once!
- In order to resend the reply the server may need to reexecute the operation in order to get the result.

### Danger!

- Some operations can be executed more than once without any problem; they are called *idempotent operations*  $\Rightarrow$  no danger with executing the duplicate request.
- There are operations which cannot be executed repeatedly without changing the effect (e.g. transferring an amount of money between two accounts)  $\Rightarrow$  *history* can be used to avoid re-execution.

History: the history is a structure which stores a record of reply messages that have been transmitted, together with the message identifier and the client which it has been sent to.



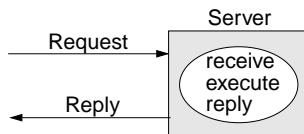
## Conclusion with Lost Messages

- Based on the previous discussion  $\Rightarrow$  correct, exactly once semantics (see slide 46) can be implemented in the case of lost (request or reply) messages. *If all the measures are taken* (duplicate filtering and history):
  - When, finally, a reply arrives at the client, the call has been executed correctly (exactly one time).
  - If no answer arrives at the client (e.g. because of broken line), an operation has been executed at most one time.
- However, the situation is different if we assume that the server can crash.

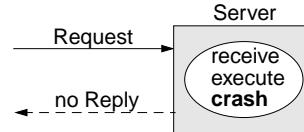


## Server Crash

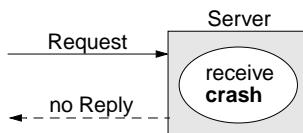
### a) The normal sequence:



### b) The server crashes after executing the operation but before sending the reply (as result of the crash, the server doesn't remember that it has executed the operation):



### c) The server crashes before executing the operation:



## Server Crash (cont'd)

### Big problem!

The client cannot distinguish between cases *b* and *c*. However they are very different and should be handled in a different way!

What to do if the client noticed that the server is down (it didn't answer to a certain large number of repeated requests)?



Petru Eles, IDA, LiTH

## Client Crash

The client sends a request to a server and crashes before the server replies.



The computation which is active in the server becomes an *orphan* - a computation nobody is waiting for.

### Problems:

- wasting of CPU time
- locked resources (files, peripherals, etc.)
- if the client reboots and repeats the RMI, confusion can be created.

The solution is based on identification and killing the orphans.



Petru Eles, IDA, LiTH

## Server Crash (cont'd)

### Alternative 1: at least once semantics

- The client's communication module sends repeated requests and waits until the server reboots or it is rebound to a new machine; when it finally receives a reply, it forwards it to the client.



When the client got an answer, the RMI has been carried out at least one time, but possibly more.

### Alternative 2: at most once semantics

- The client's communication module gives up and immediately reports the failure to the client (e.g. by raising an exception)



- If the client got an answer, the RMI has been executed exactly once.
- If the client got a failure message, the RMI has been carried out at most one time, but possibly not at all.

### Alternative 3: exactly once semantics

- This is what we would like to have (and what we could achieve for lost messages): the RMI has been carried out exactly one time.  
**However this cannot be guaranteed, in general, for the situation of server crashes.**



Petru Eles, IDA, LiTH

## Conclusion with RMI Semantics and Failures

☞ If the problem of errors is ignored, *maybe semantics* is achieved for RMI:

- the client, in general, doesn't know if the remote method has been executed once, several times or not at all.

☞ If server crashes can be excluded, *exactly once semantics* is possible to achieve, by using retries, filtering out duplicates, and using history.

☞ If server crashes with loss of memory (case b on slide 44) are considered, only *at least once* and *at most once* semantics are achievable in the best case.

In practical applications, *servers can survive crashes without loss of memory*. In such cases history can be used and duplicates can be filtered out after restart of the server:

- the client repeats sending requests without being in danger operations to be executed more than one time (this is different from alternative 2 on slide 46):
  - If no answer is received after a certain amount of tries, the client is notified and he knows that the method has been executed at most one time or not at all.
  - If an answer is received it is forwarded to the client who knows that the method has been executed exactly one time.



Petru Eles, IDA, LiTH

### Conclusion with RMI Semantics and Failures (cont'd)

- ☞ RMI semantics is different in different systems. Sometimes several semantics are implemented among which the user is allowed to select.
- ☞ And no hope about achieving exactly once semantics if servers crash ?!

In practice, systems can come close to this goal. Such are transaction-based systems with sophisticated protocols for error recovery.

- ☞ More discussion in chapter on fault tolerance.



### Group Communication

- ☞ The assumption with client-server communication and RMI (RPC) is that two parties are involved: the client and the server.
- ☞ Sometimes, however, communication involves multiple processes, not only two. A solution is to perform separate message passing operations or RMIs to each receiver.
- With *group communication* a message can be sent to multiple receivers in one operation, called *multicast*.

#### Why do we need it?

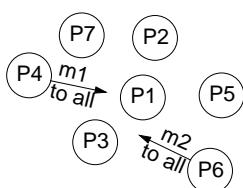
- Special applications: interest-groups, mail-lists, etc.
- Fault tolerance based on replication: a request is sent to *several* servers which all execute the same operation (if one fails, the client still will be served).
- Locating a service or object in a distributed system: the client sends a message to all machines but only the one (or those) which holds the server/object responds .
- Replicated data (for reliability or performance): whenever the data changes, the new value has to be multicast to all processes managing replicas.



### Group Communication (cont'd)

#### Essential features:

- **Atomicity** (all-or-nothing property): when a message is multicast to a group, it will either arrive correctly at all members of the group or at none of them.
- **Ordering**
  - **FIFO ordering**: The messages *from any one client to a particular server* are delivered in the order sent.
  - **Totally-ordered multicast**: when several messages are transmitted to a group the messages reach all the members of the group in the same order.



Either each process receives the messages in the order  $m_1, m_2$  or each receives them in the order  $m_2, m_1$ .

### Summary

- Middleware implements high level communication under the form of Remote Method Invocation (RMI) or Remote Procedure Call (RPC). They are based on request/reply protocols which are implemented using message passing on top of a network protocol (like the Internet).
- Client-server is a very frequently used communication pattern based on a request/reply protocol; it can be implemented using send/receive message passing primitives.
- RMI and RPC are elegant mechanisms to implement client-server systems. Remote access is solved like a local one.
- Basic components to implement RMI are: the proxy object, the skeleton object, the communication module and the remote reference module.
- An essential aspect is RMI semantics in the presence of failures. The goal is to provide *exactly once* semantics. This cannot be achieved, in general, in the presence of server crashes.
- Client-server communication, in particular RMI, involves exactly two parties. With group communication a message can be sent to multiple receivers.
- Essential features of a group communication facility are: atomicity and ordering.



## DISTRIBUTED HETEROGENEOUS APPLICATIONS AND CORBA

### 1. Heterogeneity in Distributed Systems

### 2. Middleware

### 3. Objects in Distributed Systems

### 4. The CORBA Approach

### 5. Components of a CORBA Environment

### 6. CORBA Services



Petru Eles, IDA, LiTH

## Heterogeneity in Distributed Systems

- ☞ Distributed applications are typically heterogeneous:
  - different hardware: mainframes, workstations, PCs, servers, etc.;
  - different software: UNIX, MS Windows, IBM OS/2, Real-time OSs, etc.;
  - unconventional devices: teller machines, telephone switches, robots, manufacturing systems, etc.;
  - diverse networks and protocols: Ethernet, FDDI, ATM, TCP/IP, Novell Netware, etc.

### ☞ Why?

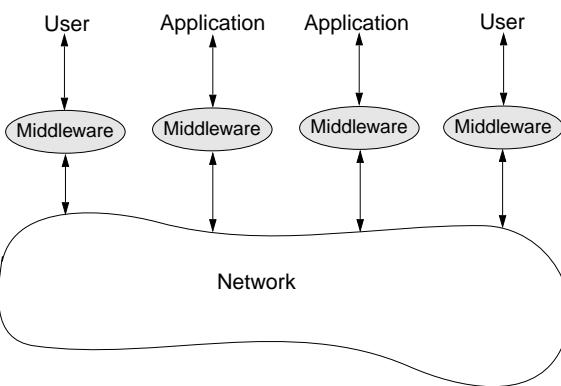
- Different hardware/software solutions are considered to be optimal for different parts of the system.
- Different users which have to interact are deciding for different hardware/software solutions/vendors.
- **Legacy systems.**



Petru Eles, IDA, LiTH

## Middleware

- ☞ A key component of a heterogeneous distributed client-server environment is *middleware*.
- *Middleware* is a set of services that enable applications and end users to interact with each other across a heterogeneous distributed system. Middleware software resides above the network and below the application software.



Petru Eles, IDA, LiTH

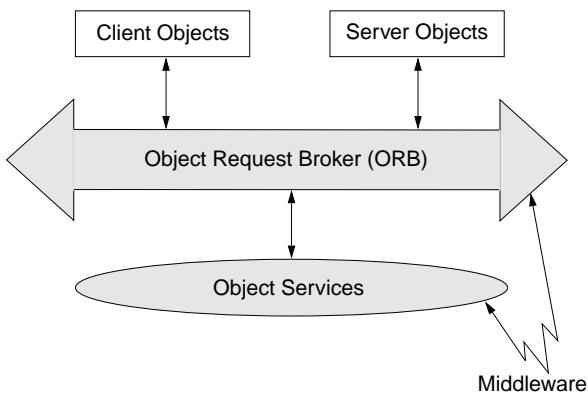
## Middleware (cont'd)

- Middleware should make the network transparent to the applications and end users ⇒ users and applications should be able to perform the same operations across the network that they can perform locally.
- Middleware should hide the details of computing hardware, OS, software components across networks.
- Different kind of software qualifies, to certain extent, as middleware:
  - File-transfer packages (FTP) and email;
  - Web browsers;
  - CORBA

Petru Eles, IDA, LiTH

## Objects in Distributed Systems

- A distributed application can be viewed as a collection of objects (user interfaces, databases, application modules, customers).



Petru Eles, IDA, LiTH

## Objects in Distributed Systems (cont'd)

- Objects are data surrounded by code; each one has its own attributes and methods which define the behavior of the object; objects can be clients, servers, or both.
- Middleware:
  - Object brokers allow objects to find each other in a distributed system and interact with each other over a network; they are the backbone of the distributed object-oriented system.
  - Object services allow to create, name, move, copy, store, delete, restore, and manage objects.
- Modeling in terms of OO concepts does not necessarily imply use of OO programming languages for implementation or the use of OO database managers as part of the system.



Petru Eles, IDA, LiTH

## Objects in Distributed Systems (cont'd)

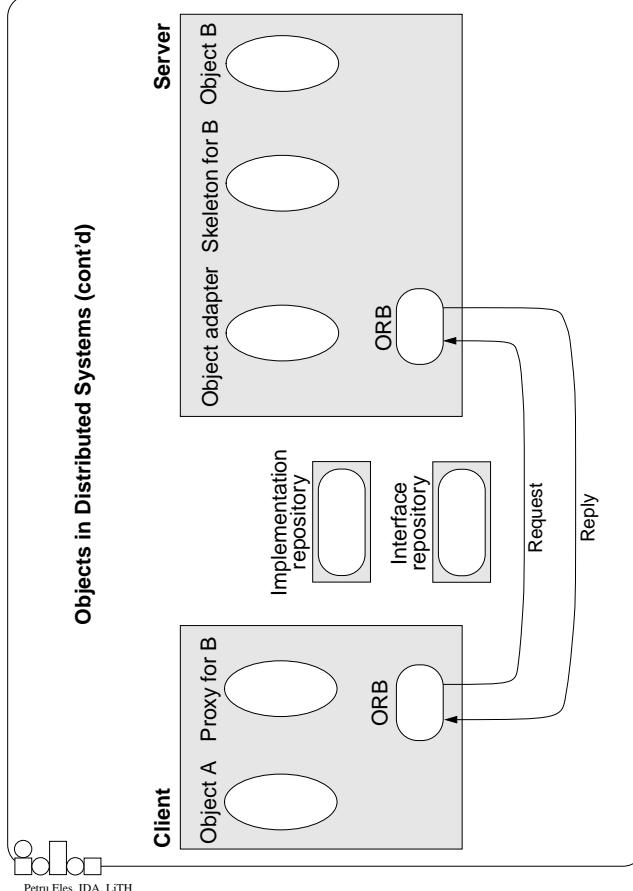
If we relate to the picture in Lecture 2/3, slide 32, which explains RMI, we can recognize some components:

- We have the client and server objects.
- We have the skeleton and proxy, which are on the border between middleware and application.
- The *communication module* and the *remote reference module* are part of the ORB.

Additional components which are part of the middleware:

- Object adapter
- Implementation repository
- Interface repository.

## Objects in Distributed Systems (cont'd)



Petru Eles, IDA, LiTH



Petru Eles, IDA, LiTH

## Interface Definition Language

- ☞ An **interface** specifies the API (Application Programming Interface) that the clients can use to invoke operations on objects:
    - the set of operations
    - the parameters needed to perform the operations.
  - One or more interfaces can be defined for an object. Such, different interfaces can be defined for different classes of users of the same object.
  - Interfaces are defined by using an **interface definition language** (IDL).
- CORBA IDL is an example of such a language.



## Interface Definition Language (cont'd)

- ☞ Middleware products (such as CORBA) provide interface compilers that parse the IDL description of the interface. Such a compiler produces the code which represents:
    - the classes corresponding to the *proxies* (in the language of the client).
    - the classes corresponding to the *skeletons* (in the language of the server).
  - If the client or the server are not in an object oriented language, the compiler generates a client stub (instead of proxy class) respectively server stub (instead of skeleton class).
  - ☞ IDLs are declarative languages; they do not specify any executable code, but only declarations.
  - ☞ IDLs should be implementation language independent ⇒ the interface is defined independent of the language in which the server and its clients are implemented.
- Language mappings* have to be defined which allow to compile the IDL interface and to generate proxies and skeletons in the implementation languages of the clients and of the server respectively.



## CORBA

- ☞ Object Management Group (OMG): a non-profit industry consortium formed in 1989 with the goal to develop, adopt, and promote standards for the development of distributed heterogeneous applications.
- ☞ One of the main achievements of OMG is the specification of a Common Object Request Broker Architecture (CORBA).
- The *CORBA specification details the interfaces and characteristics of the Object Request Broker*; it practically specifies the middleware functions which allow application objects to communicate with one another no matter where they are located, who has designed them and in which language they are implemented.
- OMG only provides a specification; there are several products which, to a certain extent, implement the OMG specification.



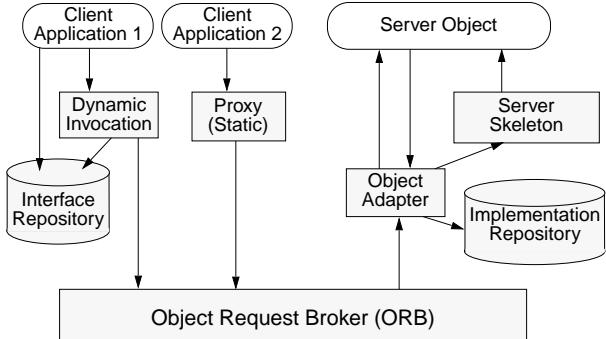
## CORBA (cont'd)

- Key concepts:
  - CORBA specifies the middleware services used by the application objects.
  - An object can be a client, a server or both.
  - Object interaction is through requests: the information associated with a request is
    1. an operation to be performed
    2. a target object
    3. zero or more parameters.
  - CORBA supports *static* as well as *dynamic binding*; dynamic binding between objects uses runtime identification of objects and parameters.
  - The *interface* represents the contract between client and server; an IDL has been defined for CORBA; proxies and skeletons (client and server stubs) are generated as result of IDL compilation.
  - CORBA objects do not know the underlying implementation details; an *object adapter* maps the generic model to a specific implementation.



## CORBA (cont'd)

## Components of a CORBA environment:



Petri Eles, IDA, LiTH

## Interface and Implementation Repository

## Interface Repository

- The interface repository provides a (standard) representation of available object interfaces for all objects in the distributed environment. It corresponds to the server objects' IDL specification.
  - The clients can access the interface repository to learn about the server objects, determine the types of operations which can be invoked and the corresponding parameters. This is used for *dynamic invocation* of objects.

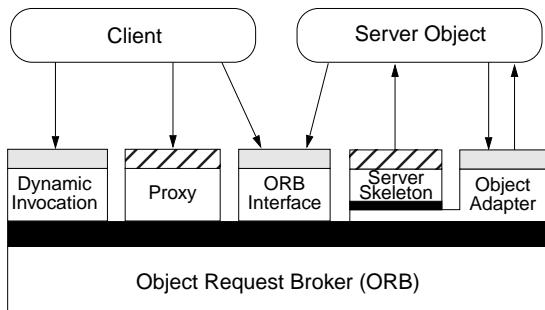
## Implementation Repository

- Implementation details for the objects implementing each interface are stored in the implementation repository:
    - the main information is a mapping from the server object's name to the file name which implements the respective service;
    - there is information concerning the object methods and information needed for method selection.
  - Information stored in the implementation repository can be specific to the operating system running on the respective server object's computer.
  - The representation in the implementation repository can be specific for a certain CORBA implementation.
  - The implementation repository is used by the *object adapter* in order to solve an incoming call and activate the right object method (via a *server skeleton*).

Petri Eles, IDA, LiTH

## The Object Request Broker (ORB)

### ORB and its interfaces:



- ORB implementation dependent interface
- Interface identical for all ORB implementations
- ▨ Proxies and skeletons for each server interface

Distributed Systems

## The Object Request Broker (cont'd)

- The ORB, through its interfaces, provides mechanisms by which objects transparently interact with each other.
  - Issuing of a request from a client can be dynamic or static; it is performed through the *proxies* (*client stubs*) or the *dynamic invocation interface*.
  - Invocation of a specific server method is performed by the server skeleton which gets the request forwarded from the *object adapter*.
  - The *ORB interface* can be accessed also *directly* by clients and object implementations for certain services: e.g. directory services, services connected to naming, manipulation of object references.

Patrik Eles, IDA, LETH

## Static and Dynamic Invocation

- ☞ CORBA allows both *static and dynamic invocation* of objects. The choice is made depending on how much information, concerning the server object, is available at compile time.

### Static Invocation

- Static invocation is based on compile time knowledge of the server's interface specification. This specification is formulated in IDL and is compiled into a *proxy (client stub)*, corresponding to the programming language in which the client is encoded.
- For the client, an object invocation is like a local invocation to a proxy method. The invocation is then automatically forwarded to the object implementation through the ORB, the object adapter and the skeleton.
- Static invocation is efficient at run time, because of the relatively low overhead.



Petru Eles, IDA, LiTH

## Static and Dynamic Invocation (cont'd)

### Dynamic Invocation

- Dynamic invocation allows a client to invoke requests on an object without having compile-time knowledge of the object's interface.
- The object and its interface (methods, parameters, types) are detected at run-time. CORBA provides, through the *dynamic invocation interface*, the mechanisms in order to inspect the *interface repository*, to dynamically construct invocations and provide argument values corresponding to the server's interface specification.
- Once the request has been constructed and arguments placed, its invocation has the same effect as a static invocation.
- The execution overhead of a dynamic invocation is huge.
- From the server's point of view, static and dynamic invocation are identical; the server does not know how it has been invoked.  
The server invocation is always issued through its skeleton, generated at compile time from the IDL specification.



Petru Eles, IDA, LiTH

## The Basic Object Adapter

- ☞ The object adapter (OA) is the primary interface between the server object implementation and the ORB.

Services provided by the OA:

- Object registration: OA provides operations by which certain entities, specified in a given programming language, are registered as *CORBA objects*.
- Object reference generation: OA generates object references to CORBA objects.
- Object upcalls: OA dispatches incoming requests to the corresponding registered objects.
- Server process and object activation: if needed, OA starts up server processes and activates objects as result of incoming invocations.



Petru Eles, IDA, LiTH

## Other CORBA Services

These services, and others, have been specified by the CORBA documents; current products implement only some of them.

### Naming and Trading Services:

- The basic way an object reference is generated is at creation of the object when the reference is returned.
- Object references can be stored together with associated information (e.g. names and properties).
- The *naming service* allows clients to find objects based on names.
- The *trading service* allows clients to find objects based on their properties.

### Transaction Management Service: provides two-phase commit coordination among recoverable components using transactions.

### Concurrency Control Service: provides a lock manager that can obtain and free locks for transactions or threads.

### Security Service: protects components from unauthorized users; it provides authentication, access control lists, confidentiality, etc.

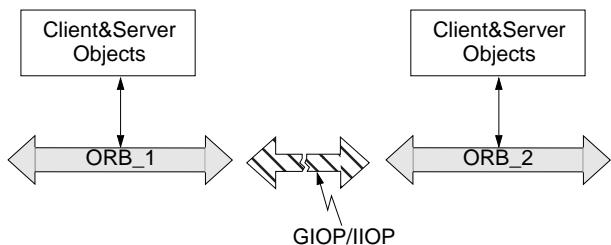
### Time Service: provides interfaces for synchronizing time; provides operations for defining and managing time-triggered events.



Petru Eles, IDA, LiTH

## Inter-ORB Architecture

- ☞ Implementations of ORBs differ from vendor to vendor ⇒ how do we solve interaction between objects which are running on different CORBA implementations?



- General Inter-ORB Protocol (GIOP): GIOP is defined in CORBA 2.0; it specifies a set of message formats and common data representations for interactions between ORBs and is intended to operate over any connection oriented transport protocol.
- Internet Inter-ORB Protocol (IIOP): IIOP is a particularization of GIOP; it specifies how GIOP messages have to be exchanged over a TCP/IP network.

## Summary

- Distributed systems are typically heterogeneous. Middleware is the set of services which enable the components to interact with each other without taking notice of the distributed and heterogeneous character of the environment.
- The API visible for the user of a service is defined in an IDL. The IDL compiler generates proxies and skeletons (client and server stubs). IDLs should be implementation language independent.
- CORBA is the OMG's specification for an Object Request Broker (ORB). Several vendors provide different (partial) implementations consistent with this specification.
- The ORB, through its interfaces, provides mechanisms by which objects transparently interact with each other.
- Objects in CORBA can be invoked statically and dynamically. Static invocation is based on compile time knowledge of the server's interface specification. Dynamic invocation allows a client to invoke requests on an object without having compile-time knowledge of the object's interface.
- The object adapter is the interface between the object implementation and the ORB. It provides services for registration of objects and their activation.
- CORBA 2.0 defines protocols for interaction between ORBs implemented by different vendors.



Petru Eles, IDA, LiTH



Petru Eles, IDA, LiTH

## PEER-TO-PEER SYSTEMS

### 1. Characteristics of Peer-to-Peer Systems

### 2. The Napster File System

### 3. Peer -to-Peer Middleware

## Basic Characteristics

- ☞ Main characteristics of peer-to-peer systems:

- Each user contributes resources to the system.
- All the nodes have the same functional capabilities and responsibilities (although they may differ in the resources they contribute).
- Correct operation does not depend on the existence of any centrally-administered system.

- ☞ Key issues:

- Choice of strategy for
  - the placement of data and their replica across many hosts;
  - the access to data.

Such that

- workload of nodes and communication lines is balanced;
- availability of data is provided.

- ☞ Anonymity of providers and users is offered (at least to a certain degree).



Petru Eles, IDA, LiTH



Petru Eles, IDA, LiTH

### Why Do We Need It?

- ☞ If only particular servers which are centrally managed, can provide services/data, then scalability is limited:
  - server capacity
  - network bandwidth provided to a server
  
- ☞ To avoid the scaling problem
  - Peer-to-peer systems use the data and computing resources available in the personal computers and workstations present on the Internet and other networks.
  - Instead of separately managed servers, services are provided by all these resources together.
  
- ☞ Important!
  - Availability of individual processes/computers in a peer-to-peer system is unpredictable

↓

Services cannot rely on guaranteed access to a host.

  - Availability can be improved by replication on several hosts.



Petru Eles, IDA, LiTH

### The Evolution of Peer-to-Peer Systems

#### First Generation:

Napster (1999)

- ☞ The index is centralised!

#### Second Generation:

Freenet (2000)

Gnutella (2000)

Kazaa (2001)

BitTorrent (2002-2003)

- ☞ Only semi-centralised or completely distributed.
- ☞ Better anonymity, scalability, fault tolerance.

#### Third Generation

Peer-to-Peer Middleware

- ☞ Platforms for application-independent management of distributed resources.
- ☞ Used to implement peer-to-peer applications.



Petru Eles, IDA, LiTH

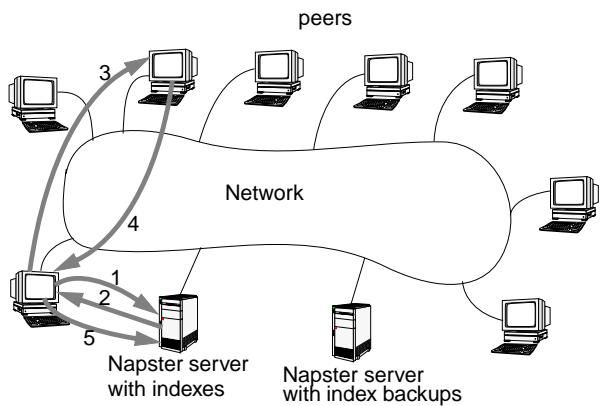
### The Napster File Sharing System

- ☞ Napster provides a globally-scalable information storage and retrieval service for digital music files.
  
- ☞ Napster was the first to demonstrate the feasibility of a peer-to-peer solution on large scale.
  
- ☞ Napster, as an open service, was shut down July 2001, as result of lawsuits on copyright issues.



Petru Eles, IDA, LiTH

### The Napster File Sharing System (cont'd)



- Step 1: File location request;
- Step 2: List of peers offering the files;
- Step 3: File request;
- Step 4: File loading;
- Step 5: Index update (user adds own files to pool of shared resources).



Petru Eles, IDA, LiTH

### The Napster File Sharing System (cont'd)

- ☞ Napster uses a centralised index (with replicas for increased availability).
- ☞ The whole pool of files is distributed over the personal computers of the peers.
- ☞ In order to achieve load balancing:
  - When creating and sending the list of peers offering the file (step 2), Napster takes into account locality (the distance between the requesting client and the potential servers).



Petru Eles, IDA, LiTH

### Problems with Napster

- ☞ Centralised index:
  - Scaling problem (server capacity and network bandwidth).
  - Anonymity of operators is not possible: for example, legal responsibility for copyright issues can be put on operators maintaining the central index.
- ☞ A completely distributed index can both provide better scaling and anonymity.
- ☞ Napster did not provide particular solutions for consistency of replica updates or for guaranteed availability. This was no problem because of the particular application, music files:
  - Music files are immutable (they don't change after being created) ⇒ there is no need to maintain replicas consistent.
  - If a file is unavailable at a certain moment it can be downloaded later.
- ☞ Second generation systems (see slide 26) have tried to solve the above problems by applying various specific, ad hoc solutions.



Petru Eles, IDA, LiTH

### Peer-to-Peer Middleware

- Peer-to-peer middleware systems provide a support for the implementation of distributed services that are located across many hosts in a widely distributed network.
- They provide a programming interface to application programmers for the implementation of peer-to-peer applications.
- No completely mature commercial products yet available.



Petru Eles, IDA, LiTH

### Peer-to-Peer Middleware (cont'd)

- ☞ The main function of peer-to-peer middleware:
  - automatic placement, replication and subsequent location of distributed objects managed by the peer-to-peer systems.
- ☞ Functionality supported:
  - add and remove hosts to/from the system;
  - add and remove resources (objects) to/from the systems;
  - allow clients to locate any individual resource made available and communicate with it.



Petru Eles, IDA, LiTH

## Requirements for Peer-to-Peer Middleware

### Scalability

- Exploit the hardware resources of a very large number of hosts.
- Support applications with millions of objects located on tens/hundreds of thousands of hosts.

### Load balancing: balanced distribution of workload among computers and network links.

- Random placement of resources.
- Use of replicas for heavily-used resources.

### Optimization of interaction: the distance between nodes that interact affects the response time and the load of the network.

- Resources should be placed close to nodes that access them the most.



Petru Eles, IDA, LiTH

## Requirements for Peer-to-Peer Middleware (cont'd)

### Adaptation to dynamic host availability: computers can whenever join the system or leave it.

- Provide a dependable service despite the unpredictable availability of the infrastructure.
- When a host joins the system the resources provided are integrated and the load redistributed.
- When a host leaves the system, the load and resources are redistributed.
- Systematic replication of objects for availability

### Security

- Proper authentication and encryption mechanisms to ensure integrity and privacy.

### Anonymity, deniability, resistance to censorship

- Keep anonymity of holders and recipients of data;
- Plausible denial of responsibility for holding/ supplying data.



Petru Eles, IDA, LiTH

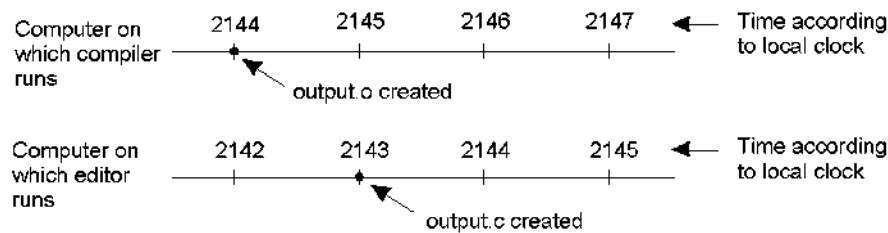
## Summary

- Peer-to-peer systems are a possible solution for the scaling problems with traditional client-server systems.
- Scaling in peer-to-peer systems is solved by exploiting the resources available on the personal computers and workstations available in the network, instead of using dedicated and centrally maintained servers.
- Napster has been the first widely used peer-to-peer system. While the pool of files is completely distributed over the personal computers of the hosts, Napster is still using a centralised index. This has consequences with regard to both scaling and anonymity.
- Peer-to-peer middleware is supporting the implementation of peer-to-peer applications. They provide automatic placement, replication, and location of objects in the peer-to-peer systems.
- Peer-to-peer systems are highly efficient to store and manage very large amounts of objects which are immutable (they don't change after being created) or which only rarely are updated.



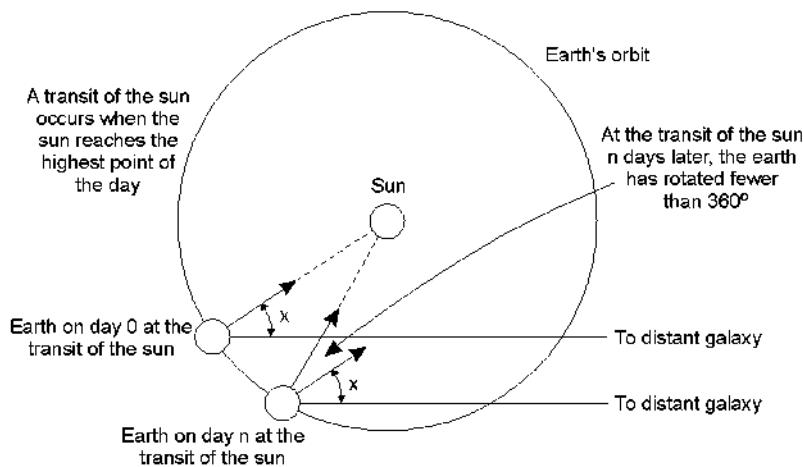
Petru Eles, IDA, LiTH

## Clock Synchronization



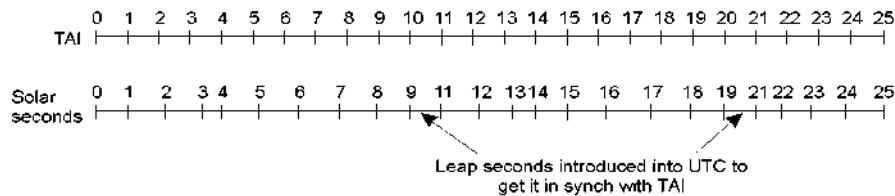
When each machine has its own clock, an event that occurred after another event may nevertheless be assigned an earlier time.

## Physical Clocks (1)



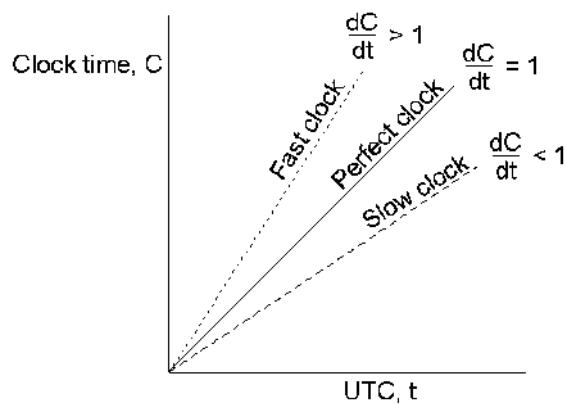
Computation of the mean solar day.

## Physical Clocks (2)



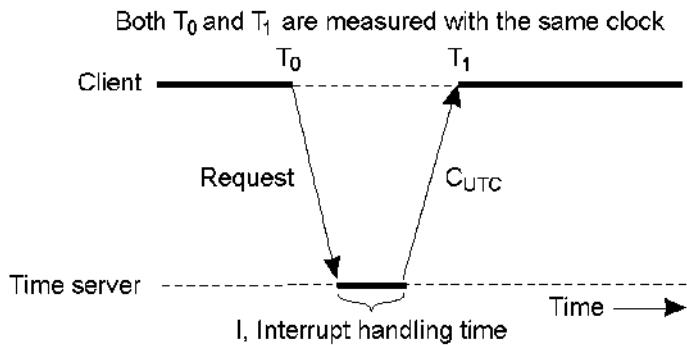
TAI seconds are of constant length, unlike solar seconds. Leap seconds are introduced when necessary to keep in phase with the sun.

## Clock Synchronization Algorithms



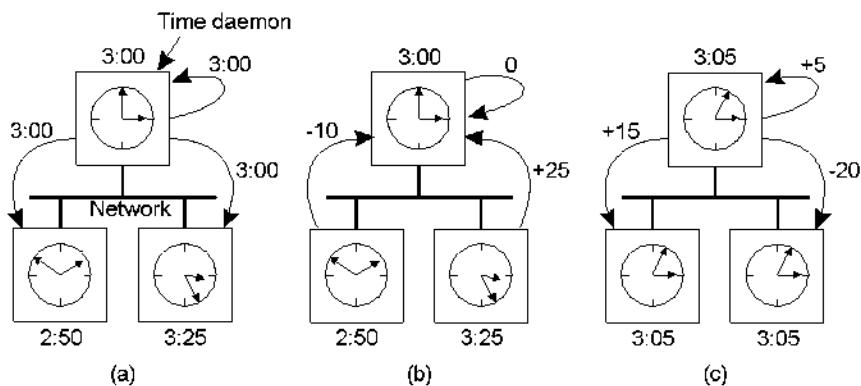
The relation between clock time and UTC when clocks tick at different rates.

## Cristian's Algorithm



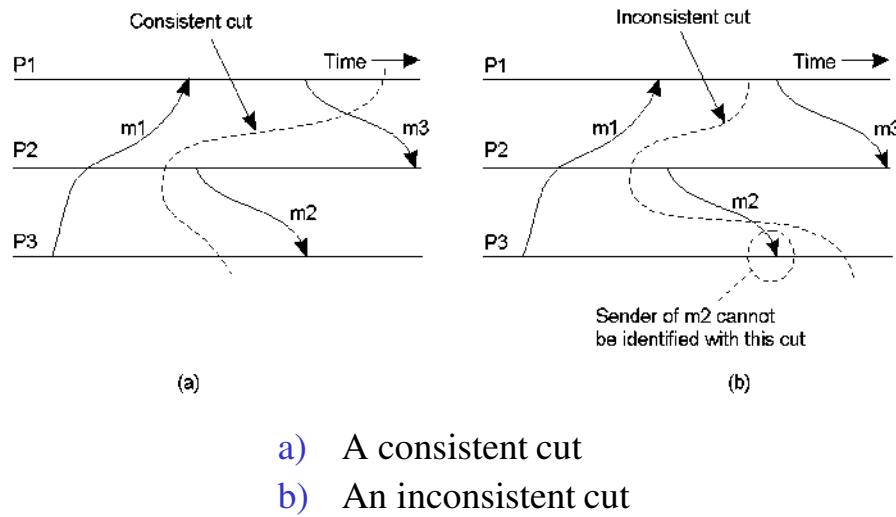
Getting the current time from a time server.

## The Berkeley Algorithm

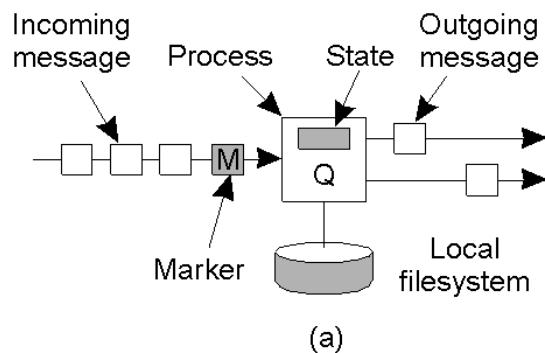


- a) The time daemon asks all the other machines for their clock values
- b) The machines answer
- c) The time daemon tells everyone how to adjust their clock

## Global State (1)

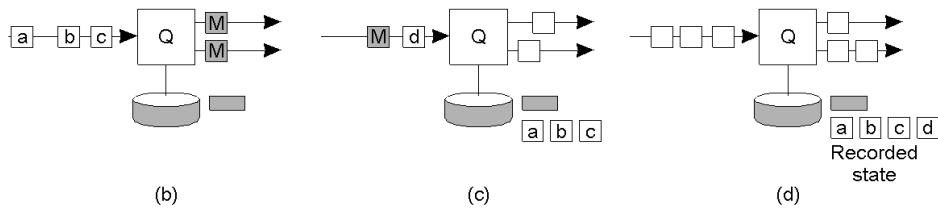


## Global State (2)



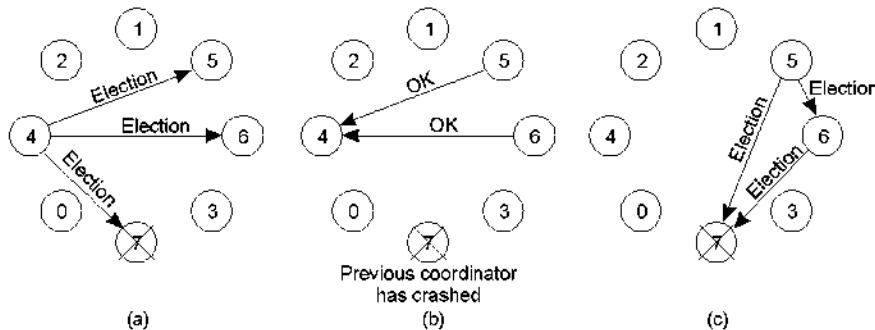
- a) Organization of a process and channels for a distributed snapshot**

## Global State (3)



- b)** Process Q receives a marker for the first time and records its local state
- c)** Q records all incoming message
- d)** Q receives a marker for its incoming channel and finishes recording the state of the incoming channel

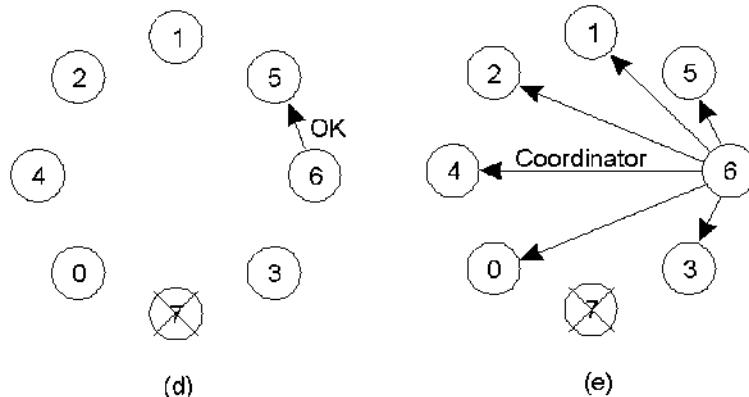
## The Bully Algorithm (1)



The bully election algorithm

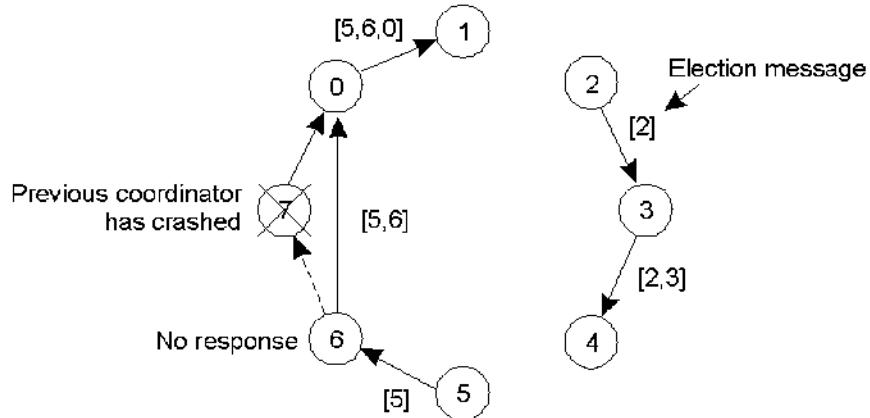
- Process 4 holds an election
- Process 5 and 6 respond, telling 4 to stop
- Now 5 and 6 each hold an election

### Global State (3)



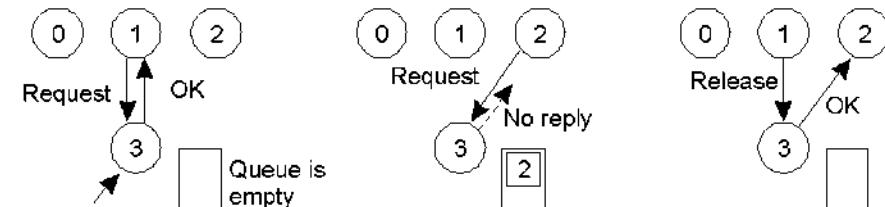
- d) Process 6 tells 5 to stop
- e) Process 6 wins and tells everyone

### A Ring Algorithm



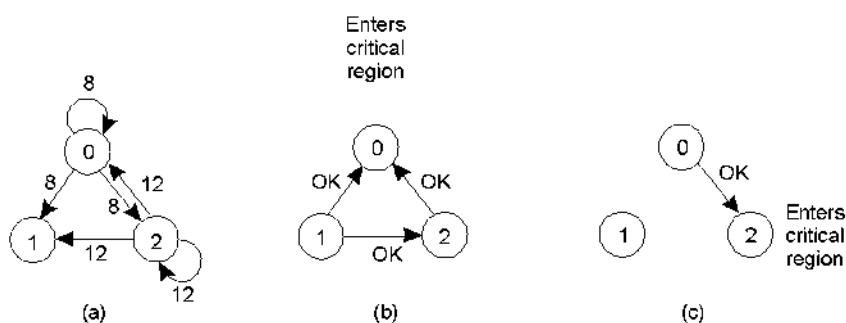
Election algorithm using a ring.

## Mutual Exclusion: A Centralized Algorithm



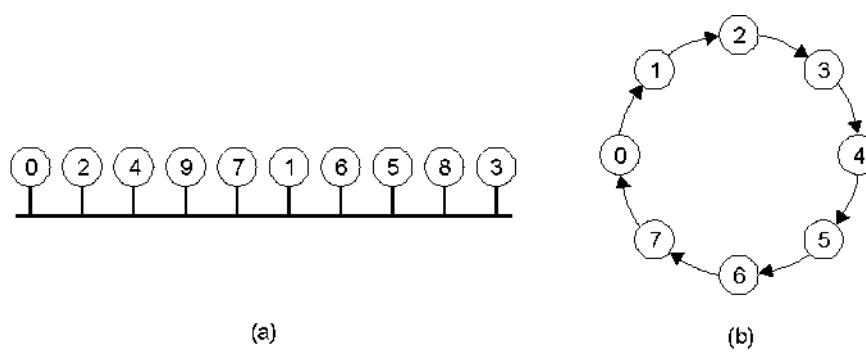
- Process 1 asks the coordinator for permission to enter a critical region. Permission is granted
- Process 2 then asks permission to enter the same critical region. The coordinator does not reply.
- When process 1 exits the critical region, it tells the coordinator, when then replies to 2

## A Distributed Algorithm



- Two processes want to enter the same critical region at the same moment.
- Process 0 has the lowest timestamp, so it wins.
- When process 0 is done, it sends an OK also, so 2 can now enter the critical region.

## A Toke Ring Algorithm



- a) An unordered group of processes on a network.
- b) A logical ring constructed in software.



# *Synchronization in Distributed Systems*

CS-4513 Distributed Systems  
Hugh C. Lauer

Slides include materials from *Modern Operating Systems*, 3<sup>rd</sup> ed., by Tannenbaum,  
*Operating System Concepts*, 7<sup>th</sup> ed., by Silberschatz, Galvin, & Gagne,  
*Distributed Systems: Principles & Paradigms*, 2<sup>nd</sup> ed. By Tanenbaum and Van Steen, and  
*Distributed Systems: Concepts and Design*, 4<sup>th</sup> ed., by Coulouris, et. al.



## *Issue*

- Synchronization within one system is hard enough
  - Semaphores
  - Messages
  - Monitors
  - ...
- Synchronization among processes in a distributed system is much harder



## Reading Assignment

- See Coulouris *et al*
  - Chapter 11, *Time and Global States*
  - Chapter 12, *Coordination and Agreement*
- Note that *Atomic Transactions* are an example of coordination and agreement.



## *Example*

- File locking in NFS
  - Not supported directly within NFS v.3
- Need *lockmanager* service to supplement NFS



# *What about using Time?*

- *make* recompiles if *foo.c* is newer than *foo.o*
- Scenario
  - *make* on machine *A* to build *foo.o*
  - Test on machine *B*; find and fix a bug in *foo.c*
  - Re-run *make* on machine *B*
  - *Nothing happens!*
- Why?



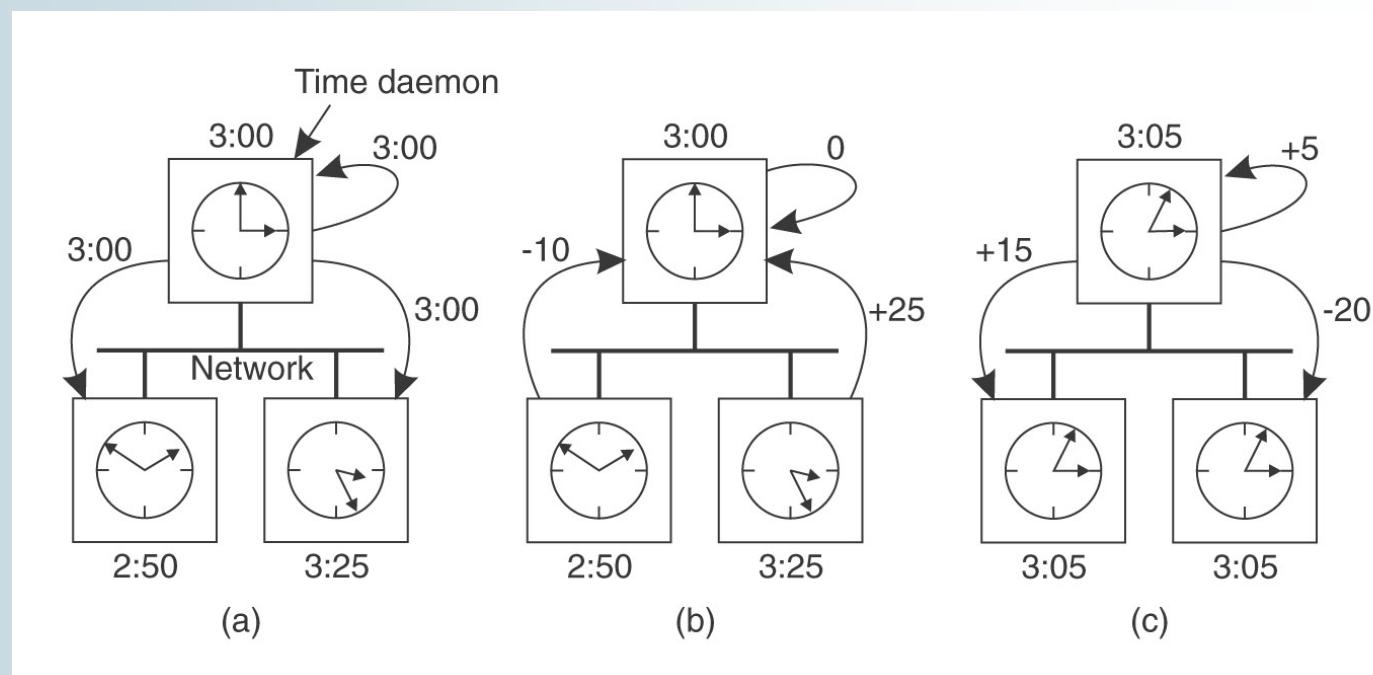
# Problem

- Time *not* a reliable method of synchronization
- Users mess up clocks
  - (and forget to set their time zones!)
- Unpredictable delays in Internet
- Relativistic issues
  - If  $A$  and  $B$  are far apart physically, and
  - two events  $T_A$  and  $T_B$  are very close in time, then
  - which comes first? how do you know?



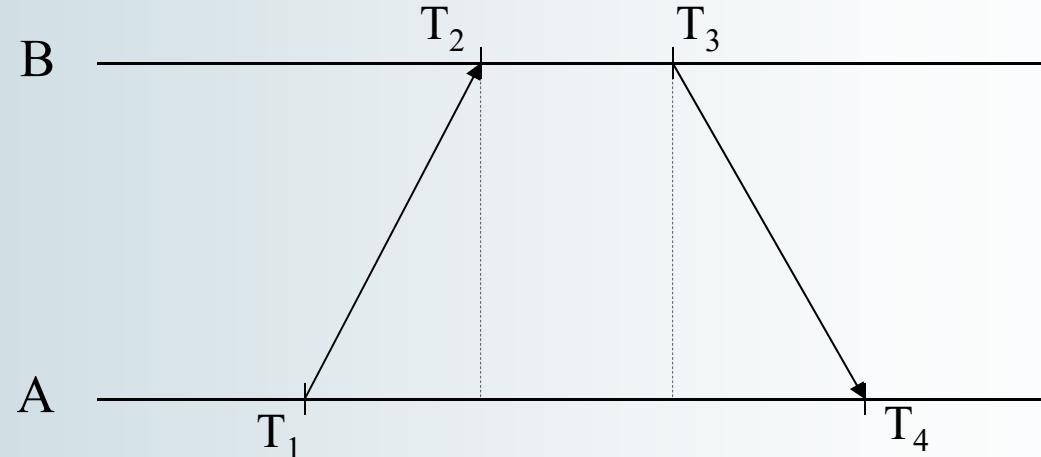
# Berkeley Algorithm

- Berkeley Algorithm
  - Time Daemon polls other systems
  - Computes average time
  - Tells other machines how to adjust their clocks





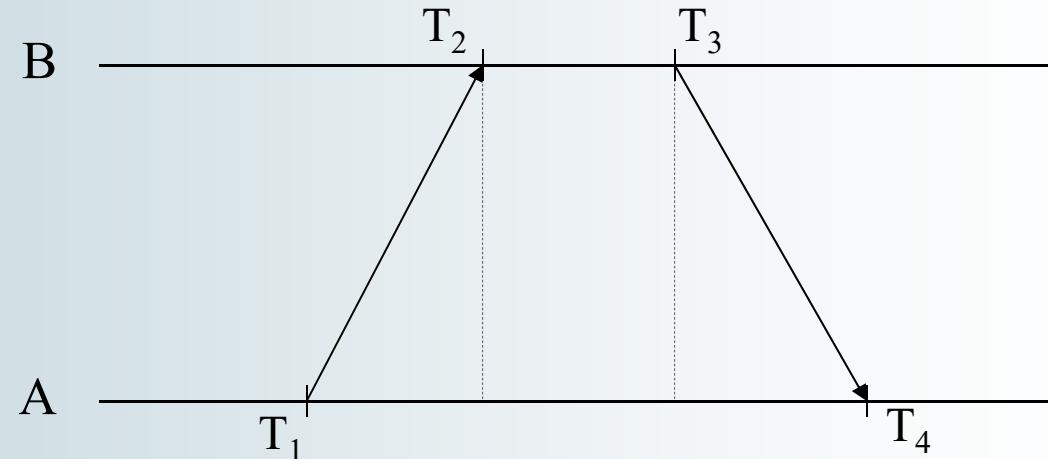
# *NTP (Network Time Protocol)*



- $A$  requests time of  $B$  at its own  $T_1$
- $B$  receives request at its  $T_2$ , records  $T_2$
- $B$  responds at its  $T_3$ , sending values of  $T_2$  and  $T_3$
- $A$  receives response at its  $T_4$
- Question: what is  $\theta = T_B - T_A$ ?



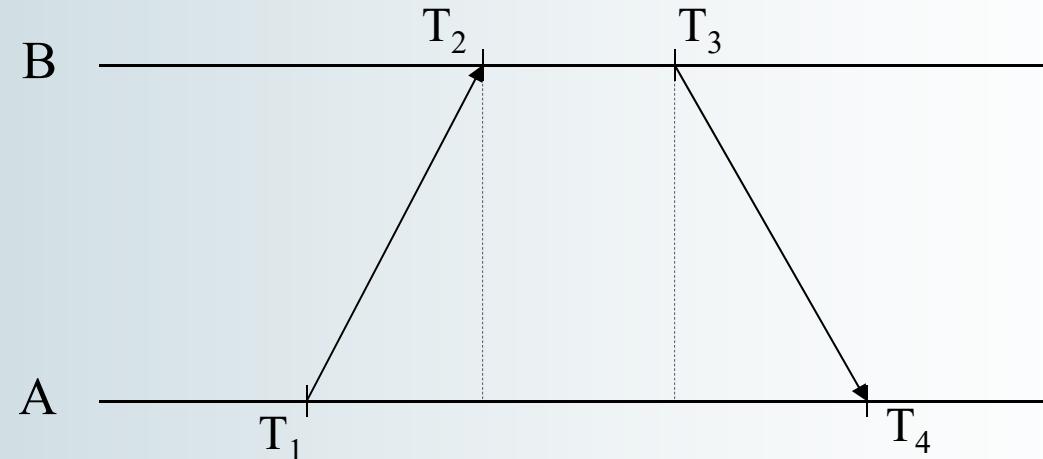
# *NTP (Network Time Protocol)*



- Question: what is  $\theta = T_B - T_A$ ?
- Assume transit time is approximately the same both ways
- Assume that  $B$  is the time server that  $A$  wants to synchronize to



# NTP (*Network Time Protocol*)

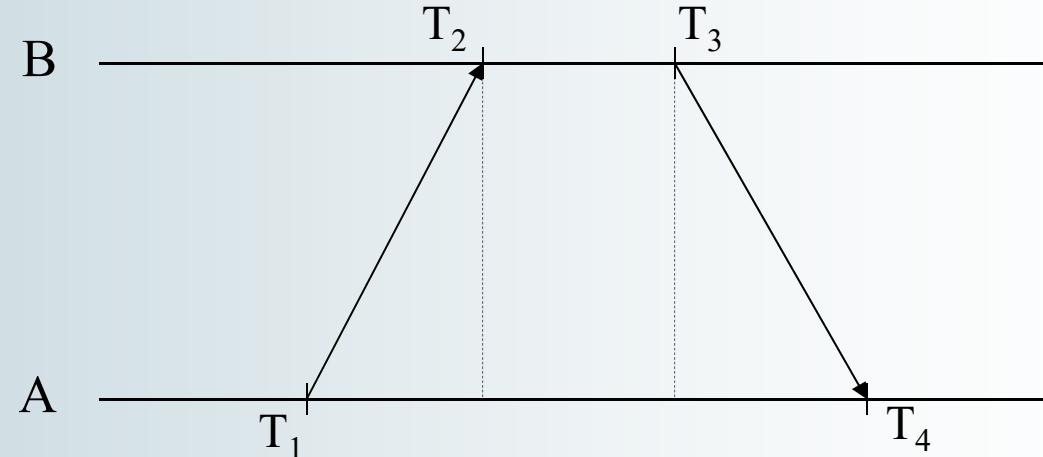


- $A$  knows  $(T_4 - T_1)$  from its own clock
- $B$  reports  $T_3$  and  $T_2$  in response to NTP request
- $A$  computes total transit time of

$$(T_4 - T_1) - (T_3 - T_2)$$



# NTP (*Network Time Protocol*)



- One-way transit time is approximately  $\frac{1}{2}$  total, i.e.,

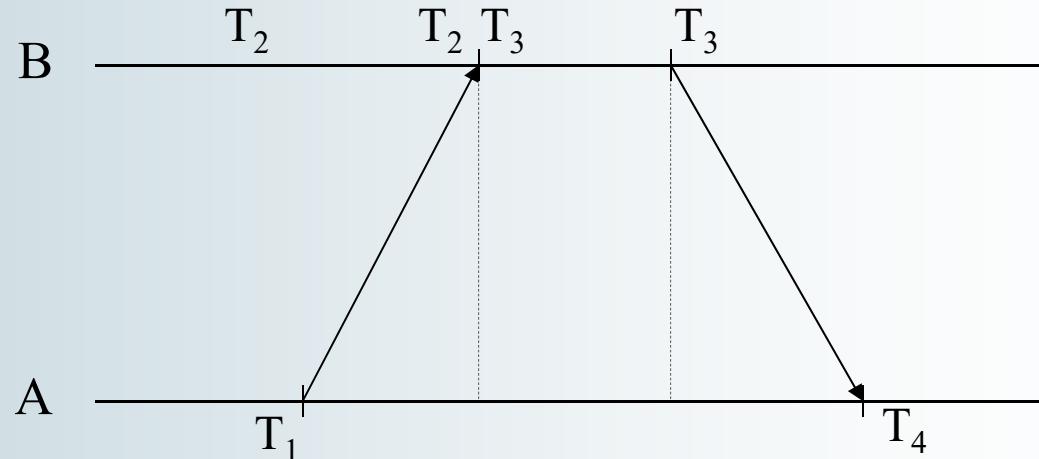
$$\frac{(T_4 - T_1) - (T_3 - T_2)}{2}$$

- $B$ 's clock at  $T_4$  reads approximately

$$T_3 + \frac{(T_4 - T_1) - (T_3 - T_2)}{2} = \frac{(T_4 - T_1) + (T_2 + T_3)}{2}$$



# NTP (Network Time Protocol)



- $B$ 's clock at  $T_4$  reads approximately (from previous slide)  
$$\frac{(T_4 - T_1) + (T_2 + T_3)}{2}$$
- Thus, difference between  $B$  and  $A$  clocks at  $T_4$  is  
$$\frac{(T_4 - T_1) + (T_2 + T_3)}{2} - T_4 = \frac{(T_2 - T_1) + (T_3 - T_4)}{2}$$



## *NTP (continued)*

- Servers organized as *strata*
  - *Stratum 0* server adjusts itself to WWV directly
  - *Stratum 1* adjusts self to *Stratum 0* servers
  - Etc.
- Within a stratum, servers adjust with each other



## *Adjusting the Clock*

- If  $T_A$  is slow, add  $\varepsilon$  to clock rate
  - To speed it up gradually
- If  $T_A$  is fast, subtract  $\varepsilon$  from clock rate
  - To slow it down gradually



## *Problem (again)*

- All of this helps, but not enough!
- Users mess up clocks
  - (and forget to set their time zones!)
- Unpredictable delays in Internet
- Relativistic issues
  - If  $A$  and  $B$  are far apart physically, and
  - two events  $T_A$  and  $T_B$  are very close in time, then
  - which comes first? how do you know?

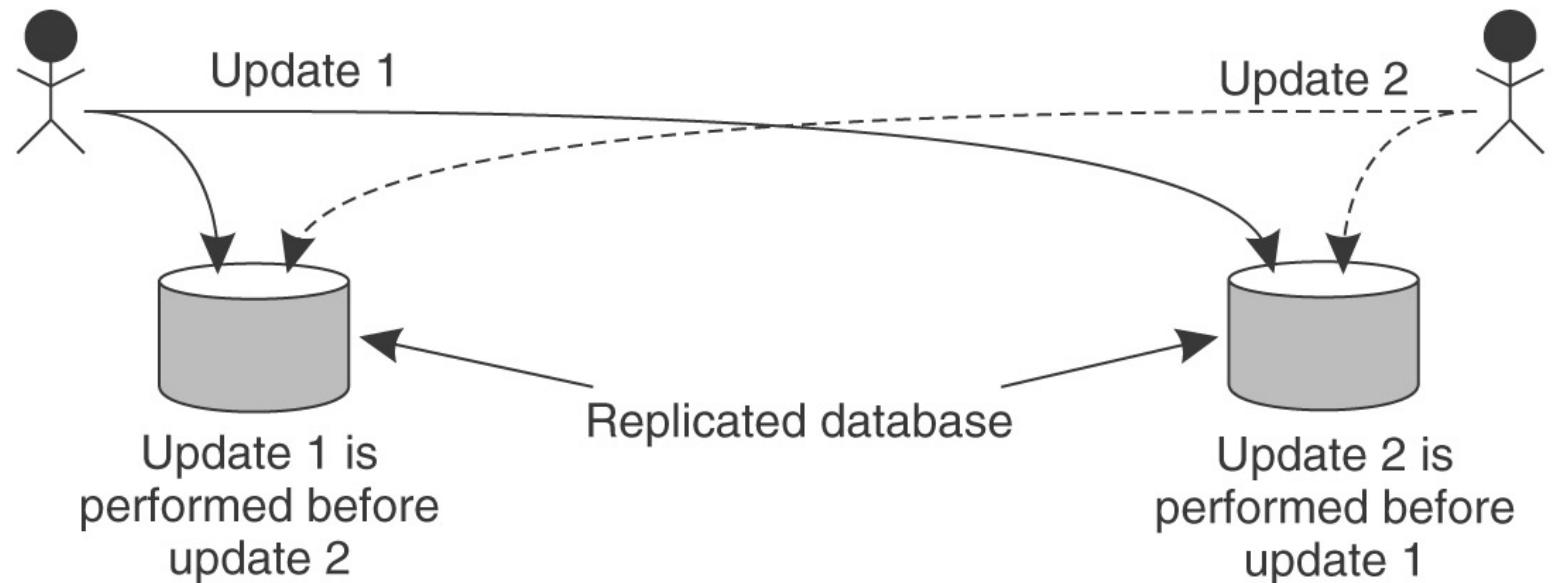


## *Example*

- At midnight PDT, bank posts interest to your account based on current balance.
- At 3:00 AM EDT, you withdraw some cash.
- Does interest get paid on the cash you just withdrew?
- Depends upon which event came first!
- What if transactions made on different replicas?

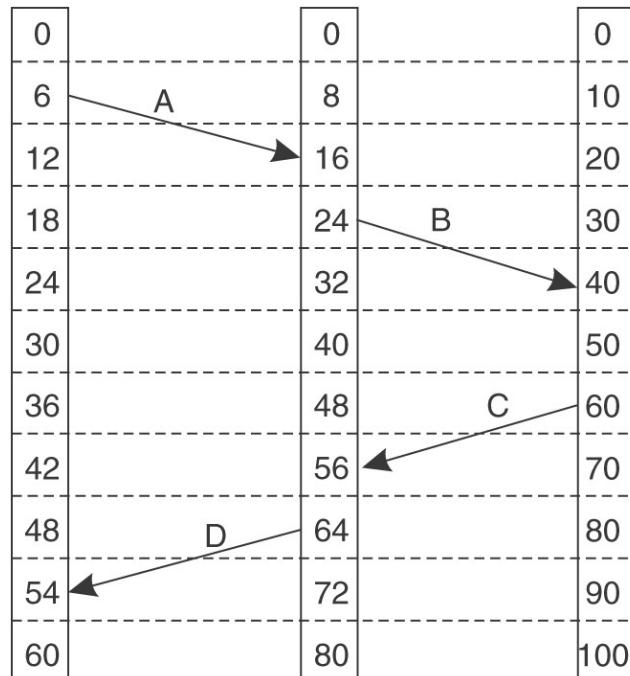


## *Example (continued)*





# *Exaggerated View*



It is impossible to conclude anything about order of events by comparing clocks



## *Solution — Logical Clocks*

- Not “clocks” at all
- Just monotonic counters
  - Lamport’s temporal logic
- Definition:  $a \rightarrow b$  means
  - $a$  occurs before  $b$
  - More specifically, all processes agree that first  $a$  happens, then later  $b$  happens
- E.g.,  $\text{send}(\text{message}) \rightarrow \text{receive}(\text{message})$

For example, if  $b$  is known to be *caused* by something associated with  $a$



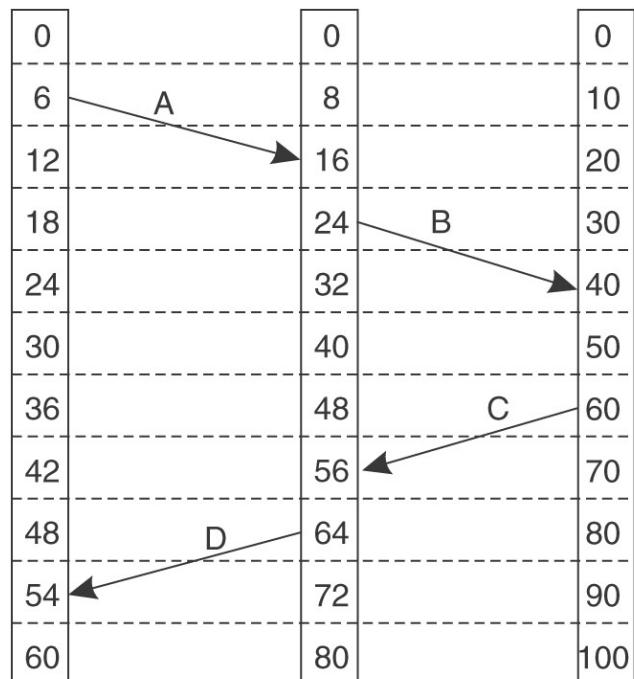
# *Implementation of Logical Clocks*

- Every machine maintains its own logical “clock”  $C$
- Transmit  $C$  with *every* message
- If  $C_{\text{received}} > C_{\text{own}}$ , then adjust  $C_{\text{own}}$  forward to  $C_{\text{received}} + 1$
- Result: Anything that is *known* to follow something else in *time* has larger *logical clock* value.



# Logical Clocks (continued)

Without Logical Clocks

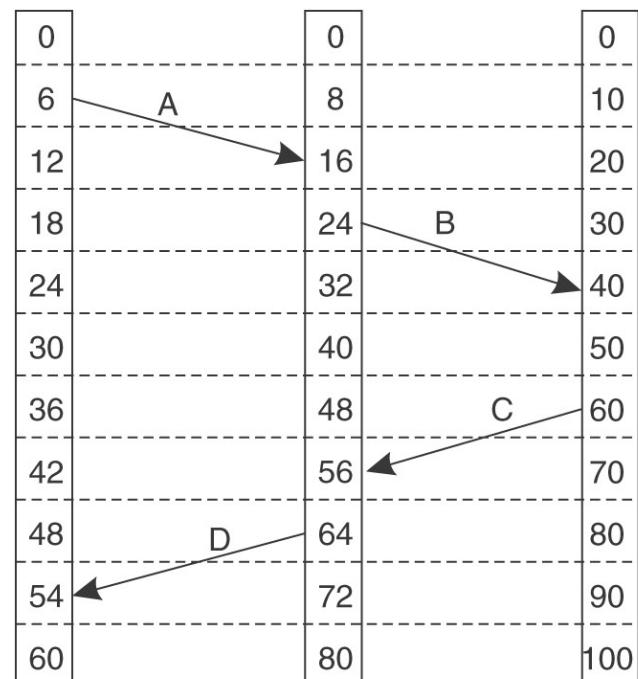


(a)



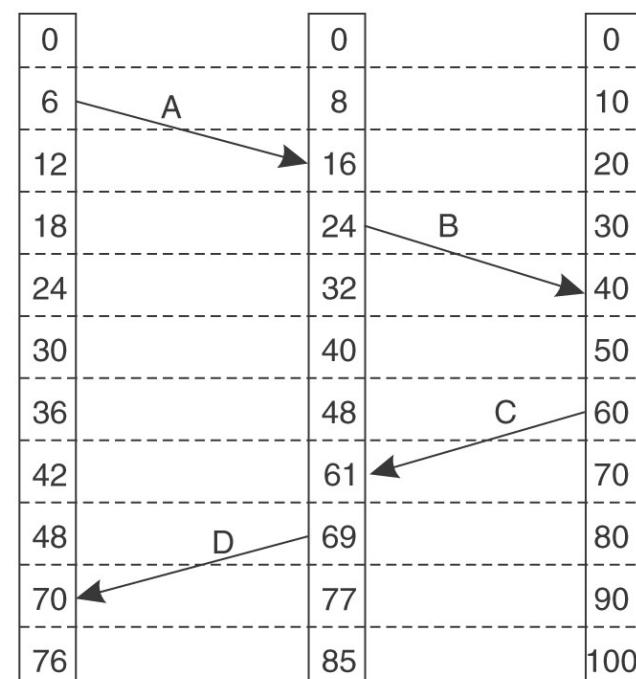
## Logical Clocks (continued)

Without Logical Clocks



(a)

With Logical Clocks



(b)



## Variations

- See Coulouris, *et al*, §11.4
- Note: Grapevine *timestamps* for updating its registries behave somewhat like logical clocks.



*Questions?*

## TIME AND STATE IN DISTRIBUTED SYSTEMS

### 1. Time in Distributed Systems

### 2. Lamport's Logical Clocks

### 3. Vector Clocks

### 4. Causal Ordering of Messages

### 5. Global States and their Consistency

### 6. Cuts of a Distributed Computation

### 7. Recording of a Global State



Petru Eles, IDA, LiTH

## Time in Distributed Systems

- Because each machine in a distributed system has its own clock there is no notion of *global physical time*.
- The  $n$  crystals on the  $n$  computers will run at slightly different rates, causing the clocks gradually to get out of synchronization and give different values.

### Problems:

- Time triggered systems: these are systems in which certain activities are scheduled to occur at predefined moments in time. If such activities are to be coordinated over a distributed system we need a coherent notion of time.

Example: time-triggered real-time systems

- Maintaining the consistency of distributed data is often based on the *time* when a certain modification has been performed.

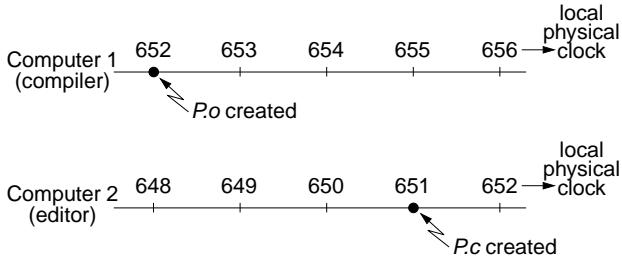
Example: a *make* program.

Petru Eles, IDA, LiTH

## Time in Distributed Systems (cont'd)

### The *make*-program example

- When the programmer has finished changing some source files he starts *make*; *make* examines the times at which all object and source files were last modified and decides which source files have to be (re)compiled.



Although  $P_c$  is modified after  $P_o$  has been generated, because of the clock drift the time assigned to  $P_c$  is smaller.



$P_c$  will not be recompiled for the new version!

## Time in Distributed Systems (cont'd)

### Solutions:

#### Synchronization of physical clocks

- Computer clocks are synchronized with one another to an achievable, known, degree of accuracy  $\Rightarrow$  within the bounds of this accuracy we can coordinate activities on different computers using each computer's local clock.
- Physical clock synchronization is needed for distributed real-time systems.

#### Logical clocks

- In many applications we are not interested in the physical time at which events occur; what is important is the relative order of events! The *make*-program is such an example (slide 3).
- In such situations we don't need synchronized physical clocks. Relative ordering is based on a virtual notion of time - *logical time*.
- Logical time is implemented using *logical clocks*.



Petru Eles, IDA, LiTH



Petru Eles, IDA, LiTH

## Lamport's Logical Clocks

- The order of events occurring at different processes is critical for many distributed applications.  
Example:  $P_o\_created$  and  $P_c\_created$  in slide 3.

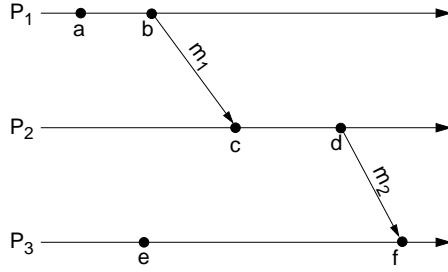
- Ordering can be based on two simple situations:
  - If two events occurred in the same process then they occurred in the order observed following the respective process;
  - Whenever a message is sent between processes, the event of sending the message occurred before the event of receiving it.

- Ordering by Lamport is based on the *happened-before relation* (denoted by  $\rightarrow$ ):
  - $a \rightarrow b$ , if  $a$  and  $b$  are events in the same process and  $a$  occurred before  $b$ ;
  - $a \rightarrow b$ , if  $a$  is the event of sending a message  $m$  in a process, and  $b$  is the event of the same message  $m$  being received by another process;
  - If  $a \rightarrow b$  and  $b \rightarrow c$ , then  $a \rightarrow c$  (the relation is transitive).



## Lamport's Logical Clocks (cont'd)

- If  $a \rightarrow b$ , we say that event  $a$  causally affects event  $b$ .  
The two events are causally related.
- There are events which are not related by the *happened-before* relation.  
If both  $a \rightarrow e$  and  $e \rightarrow a$  are false, then  $a$  and  $e$  are concurrent events; we write  $a \parallel e$ .



$P_1, P_2, P_3$ : processes;  
 $a, b, c, d, e, f$ : events;

$a \rightarrow b, c \rightarrow d, e \rightarrow f, b \rightarrow c, d \rightarrow f$   
 $a \rightarrow c, a \rightarrow d, a \rightarrow f, b \rightarrow d, b \rightarrow f, \dots$   
 $a \parallel e, c \parallel e, \dots$



## Lamport's Logical Clocks (cont'd)

- Using physical clocks, the happened before relation can not be captured. It is possible that  $b \rightarrow c$  and at the same time  $T_b > T_c$  ( $T_b$  is the physical time of  $b$ ).

- Logical clocks can be used in order to capture the *happened-before relation*.
  - A logical clock is a monotonically increasing software counter.
  - There is a logical clock  $C_{P_i}$  at each process  $P_i$  in the system.
  - The value of the logical clock is used to assign *timestamps* to events.  $C_{P_i}(a)$  is the timestamp of event  $a$  in process  $P_i$ .
  - There is no relationship between a logical clock and any physical clock.

To capture the happened-before relation, logical clocks have to be implemented so that

$$\text{if } a \rightarrow b, \text{ then } C(a) < C(b)$$

## Lamport's Logical Clocks (cont'd)

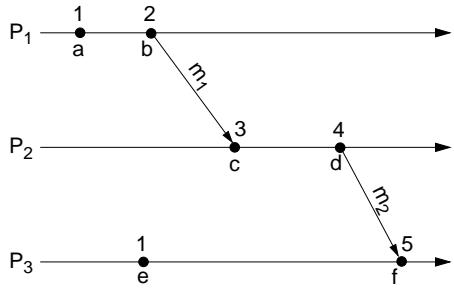
- Implementation of logical clocks is performed using the following rules for updating the clocks and transmitting their values in messages:

[R1]:  $C_{P_i}$  is incremented before each event is issued at process  $P_i$ :  $C_{P_i} := C_{P_i} + 1$ .

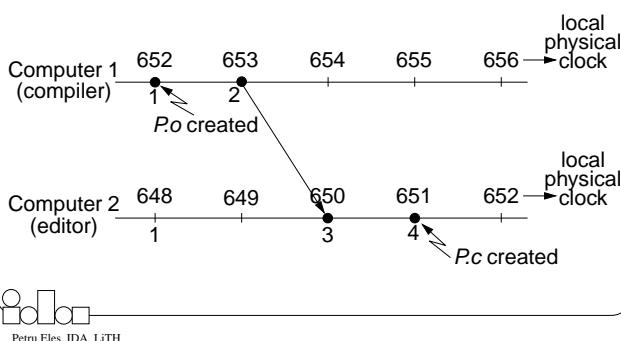
- [R2]:
- When  $a$  is the event of sending a message  $m$  from process  $P_i$ , then the timestamp  $t_m = C_{P_i}(a)$  is included in  $m$  ( $C_{P_i}(a)$  is the logical clock value obtained after applying rule R1).
  - On receiving message  $m$  by process  $P_j$ , its logical clock  $C_{P_j}$  is updated as follows:  
 $C_{P_j} := \max(C_{P_j}, t_m)$ .
  - The new value of  $C_{P_j}$  is used to timestamp the event of receiving message  $m$  by  $P_j$  (applying rule R1).
- If  $a$  and  $b$  are events in the same process and  $a$  occurred before  $b$ , then  $a \rightarrow b$ , and (by R1)  $C(a) < C(b)$ .
  - If  $a$  is the event of sending a message  $m$  in a process, and  $b$  is the event of the same message  $m$  being received by another process, then  $a \rightarrow b$ , and (by R2)  $C(a) < C(b)$ .
  - If  $a \rightarrow b$  and  $b \rightarrow c$ , then  $a \rightarrow c$ , and (by induction)  $C(a) < C(c)$ .



### Lamport's Logical Clocks (cont'd)



- For the *make-program* example we suppose that a process running a compilation notifies, through a message, the process holding the source file about the event  $P.o$  created  $\Rightarrow$  a logical clock can be used to correctly timestamp the files.

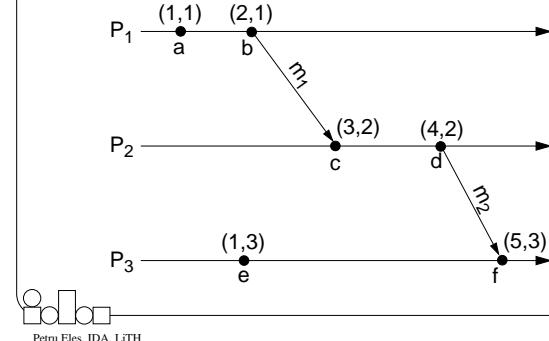


Petru Eles, IDA, LiTH

### Problems with Lamport's Logical Clocks

☞ Lamport's logical clocks impose only a partial order on the set of events; pairs of distinct events generated by *different* processes can have identical timestamp.

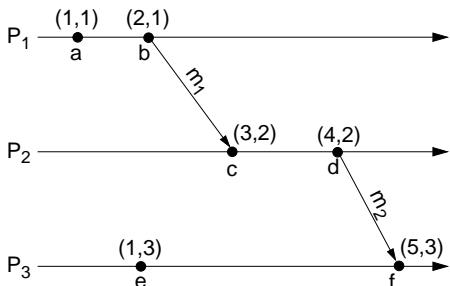
- For certain applications a total ordering is needed; they consider that no two events can occur at the same time.
- In order to enforce total ordering a *global logical timestamp* is introduced:
  - the global logical timestamp of an event  $a$  occurring at process  $P_i$ , with logical timestamp  $C_{P_i}(a)$ , is a pair  $(C_{P_i}(a), i)$ , where  $i$  is an identifier of process  $P_i$
  - we define  
 $(C_{P_i}(a), i) < (C_{P_j}(b), j)$  if and only if  
 $C_{P_i}(a) < C_{P_j}(b)$ , or  $C_{P_i}(a) = C_{P_j}(b)$  and  $i < j$ .



Petru Eles, IDA, LiTH

### Problems with Lamport's Logical Clocks (cont'd)

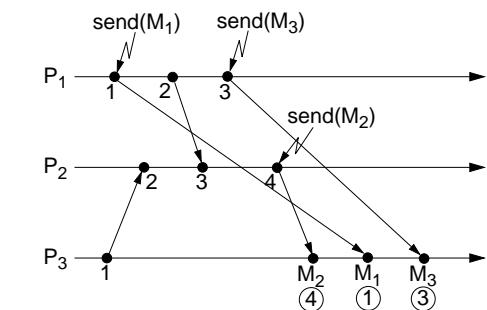
- ☞ Lamport's logical clocks are not powerful enough to perform a causal ordering of events.
- if  $a \rightarrow b$ , then  $C(a) < C(b)$ .  
 However, the reverse is not always true (if the events occurred in different processes):  
 if  $C(a) < C(b)$ , then  $a \rightarrow b$  is **not necessarily** true.  
 (it is only guaranteed that  $b \rightarrow a$  is not true).



$C(e) < C(b)$ , however there is no causal relation from event  $e$  to event  $b$ .

- By just looking at the timestamps of the events, we cannot say whether two events are causally related or not.

### Problems with Lamport's Logical Clocks (cont'd)



- We would like messages to be processed according to their causal order.  
 We would like to use the associated *timestamp* for this purpose.
- Process  $P_3$  receives messages  $M_1$ ,  $M_2$ , and  $M_3$ .  
 $send(M_1) \rightarrow send(M_2)$ ,  $send(M_1) \rightarrow send(M_3)$ ,  
 $send(M_3) \parallel send(M_2)$
- $M_1$  has to be processed before  $M_2$  and  $M_3$ .  
 However  $P_3$  has not to wait for  $M_3$  in order to process it before  $M_2$  (although  $M_3$ 's logical clock timestamp is smaller than  $M_2$ 's).

Petru Eles, IDA, LiTH

## Vector Clocks

- ☞ Vector clocks give the ability to decide whether two events are causally related or not by simply looking at their timestamp.

- Each process  $P_i$  has a clock  $C_{P_i}^V$ , which is an integer vector of length  $n$  ( $n$  is the number of processes).
- The value of  $C_{P_i}^V$  is used to assign timestamps to events in process  $P_i$ .  
 $C_{P_i}^V(a)$  is the timestamp of event  $a$  in process  $P_i$ .
- $C_{P_i}^V[i]$ , the  $i$ th entry of  $C_{P_i}^V$ , corresponds to  $P_i$ 's own logical time.
- $C_{P_i}^V[j]$ ,  $j \neq i$ , is  $P_i$ 's "best guess" of the logical time at  $P_j$ .  
 $C_{P_i}^V[j]$  indicates the (logical) time of occurrence of the last event at  $P_j$  which is in a *happened-before* relation to the current event at  $P_i$ .



Petru Eles, IDA, LiTH

## Vector Clocks (cont'd)

- ☞ Implementation of vector clocks is performed using the following rules for updating the clocks and transmitting their values in messages:

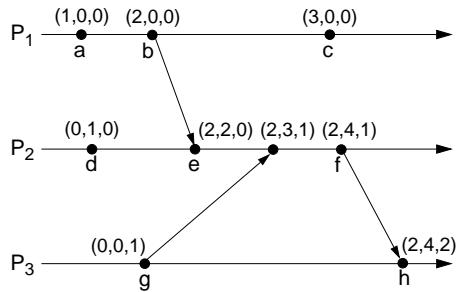
[R1]:  $C_{P_i}^V$  is incremented before each event is issued at process  $P_i$ :  $C_{P_i}^V[i] := C_{P_i}^V[i] + 1$ .

- [R2]:
- When  $a$  is the event of sending a message  $m$  from process  $P_i$ , then the timestamp  $t_m = C_{P_i}^V(a)$  is included in  $m$  ( $C_{P_i}^V(a)$  is the vector clock value obtained after applying rule R1).
  - On receiving message  $m$  by process  $P_j$ , its vector clock  $C_{P_j}^V$  is updated as follows:  
 $\forall k \in \{1, 2, \dots, n\}, C_{P_j}^V[k] := \max(C_{P_j}^V[k], t_m[k])$ .
  - The new value of  $C_{P_j}^V$  is used to timestamp the event of receiving message  $m$  by  $P_j$  (applying rule R1).



Petru Eles, IDA, LiTH

## Vector Clocks (cont'd)



For any two vector timestamps  $u$  and  $v$ , we have:

- $u = v$  if and only if  $\forall i, u[i] = v[i]$
- $u \leq v$  if and only if  $\forall i, u[i] \leq v[i]$
- $u < v$  if and only if  $(u \leq v \wedge u \neq v)$
- $u \parallel v$  if and only if  $\neg(u < v) \wedge \neg(v < u)$

☞ Two events  $a$  and  $b$  are causally related if and only if  $C^V(a) < C^V(b)$  or  $C^V(b) < C^V(a)$ . Otherwise the events are concurrent.

☞ With vector clocks we get the property which we missed for Lamport's logical clocks:

- $a \rightarrow b$  if and only if  $C^V(a) < C^V(b)$ .  
 Thus, by just looking at the timestamps of the events, we can say whether two events are causally related or not.



Petru Eles, IDA, LiTH

## Causal Ordering of Messages Using Vector Clocks

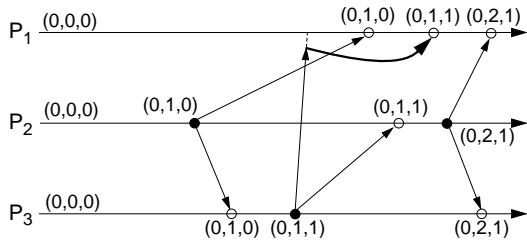
The problem has been formulated on slide 12:

- ☞ We would like messages to be processed according to their causal order.
- If  $Send(M_1) \rightarrow Send(M_2)$ , then every recipient of both messages  $M_1$  and  $M_2$  must receive  $M_1$  before  $M_2$ .



Petru Eles, IDA, LiTH

### Causal Ordering of Messages Using Vector Clocks (cont'd)



- ☞ A message delivery protocol which performs causal ordering based on vector clocks.

- Basic Idea:

- A message is delivered to a process only if the message immediately preceding it (considering the causal ordering) has been already delivered to the process. Otherwise, the message is buffered.
- We assume that processes communicate using broadcast messages.  
(There exist similar protocols for non-broadcast communication too.)



Petru Eles, IDA, LiTH

### Causal Ordering of Messages Using Vector Clocks (cont'd)

- The events which are of interest here are the sending of messages  $\Rightarrow$  vector clocks will be incremented only for message sending.

☞ Implementation of the protocol is based on the following rules:

- [R1]: a) Before broadcasting a message  $m$ , a process  $P_i$  increments the vector clock:  $C_{P_i}[l] := C_{P_i}[l] + 1$ .  
b) The timestamp  $t_m = C_{P_i}^V$  is included in  $m$ .

- [R2]: The receiving side, at process  $P_j$ , delays the delivery of message  $m$  coming from  $P_i$  until both the following conditions are satisfied:

- $C_{P_j}[l] = t_m[l] - 1$
- $\forall k \in \{1, 2, \dots, n\} - \{l\}, C_{P_j}^V[k] \geq t_m[k]$

Delayed messages are queued at each process in a queue that is sorted by their vector timestamp; concurrent messages are ordered by the time of their arrival.

- [R3]: When a message is delivered at process  $P_j$ , its vector clock  $C_{P_j}^V$  is updated according to rule R2b for vector clock implementation (see slide 14).

- ☞  $t_m[l] - 1$  indicates how many messages originating from  $P_i$  precede  $m$ .  
Step R2.1 ensures that process  $P_j$  has received all the messages originating from  $P_i$  that precede  $m$ .  
Step R2.2 ensures that  $P_j$  has received all those messages received by  $P_i$  before sending  $m$ .



Petru Eles, IDA, LiTH

### Global States

- ☞ The problem is how to collect and record a *consistent global state* in a distributed system.

#### Why a problem?

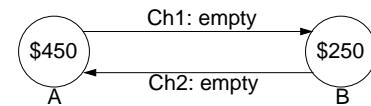
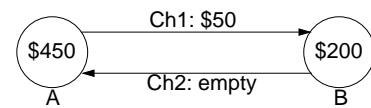
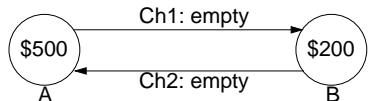
- Because there is no global clock (no coherent notion of time) and no shared memory!



Petru Eles, IDA, LiTH

### Global States (cont'd)

Consider a bank system with two accounts  $A$  and  $B$  at two different sites; we transfer \$50 between  $A$  and  $B$ .



| A   | Ch1   | B   | C : consistent<br>NC: not consistent |
|-----|-------|-----|--------------------------------------|
| 500 | empty | 200 | C                                    |
| 500 | 50    | 200 | NC                                   |
| 450 | 50    | 200 | C                                    |
| 450 | empty | 200 | NC                                   |
| 500 | 50    | 250 | NC                                   |
| 450 | 50    | 250 | NC                                   |
| 450 | empty | 250 | C                                    |
| 500 | empty | 250 | NC                                   |



Petru Eles, IDA, LiTH

### Global States (cont'd)

- ☞ In general, a global state consists of a set of local states and a set of states of the communication channels.
- ☞ The state of the communication channel in a *consistent global state* should be the sequence of messages sent along the channel before the sender's state was recorded, excluding the sequence of messages received along the channel before the receiver's state was recorded.
- ☞ It is difficult to record channel states to ensure the above rule  $\Rightarrow$  *global states are very often recorded without using channel states*.  
This is the case in the definition below.



Petru Eles, IDA, LiTH

### Formal Definition

- $LS_i$  is the local state of process  $P_i$ .  
Beside other information, the local state also includes a record of all messages sent and received by the process.
- We consider the global state  $GS$  of a system, as the collection of the local states of its processes:  
 $GS = \{LS_1, LS_2, \dots, LS_n\}$ .
- A certain global state can be consistent or not!
- $send(m_{ij}^k)$  denotes the event of sending message  $m_{ij}^k$  from  $P_i$  to  $P_j$   
 $rec(m_{ij}^k)$  denotes the event of receiving message  $m_{ij}^k$  by  $P_j$
- $send(m_{ij}^k) \in LS_i$  if and only if the sending event occurred before the local state was recorded;  
 $rec(m_{ij}^k) \in LS_j$  if and only if the receiving event occurred before the local state was recorded.
- $transit(LS_i, LS_j) = \{m_{ij}^k \mid send(m_{ij}^k) \in LS_i \wedge rec(m_{ij}^k) \in LS_j\}$   
 $inconsistent(LS_i, LS_j) = \{m_{ij}^k \mid send(m_{ij}^k) \notin LS_i \wedge rec(m_{ij}^k) \in LS_j\}$



Petru Eles, IDA, LiTH

### Formal Definition (cont'd)

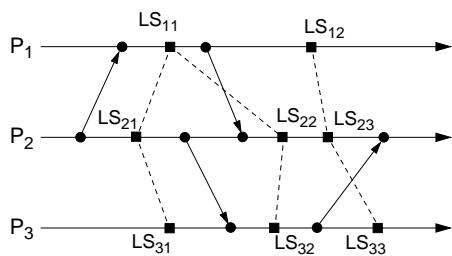
- ☞ A global state  $GS = \{LS_1, LS_2, \dots, LS_n\}$  is *consistent* if and only if:  

$$\forall i, \forall j: 1 \leq i, j \leq n :: inconsistent(LS_i, LS_j) = \emptyset$$
  - In a consistent global state for every received message a corresponding send event is recorded in the global state.
  - In an inconsistent global state, there is at least one message whose receive event is recorded but its send event is not recorded.
- ☞ A global state  $GS = \{LS_1, LS_2, \dots, LS_n\}$  is *transitless* if and only if:  

$$\forall i, \forall j: 1 \leq i, j \leq n :: transit(LS_i, LS_j) = \emptyset$$
  - All messages recorded to be sent are also recorded to be received.
- ☞ A global state is *strongly consistent* if it is consistent and transitless.
  - A strongly consistent state corresponds to a consistent state in which all messages recorded as sent are also recorded as received.

Note: the global state, as defined here, is seen as a collection of the local states, without explicitly capturing the state of the channel.

### Formal Definition (cont'd)



$\{LS_{11}, LS_{22}, LS_{32}\}$  is inconsistent;  
 $\{LS_{12}, LS_{23}, LS_{33}\}$  is consistent;  
 $\{LS_{11}, LS_{21}, LS_{31}\}$  is strongly consistent.

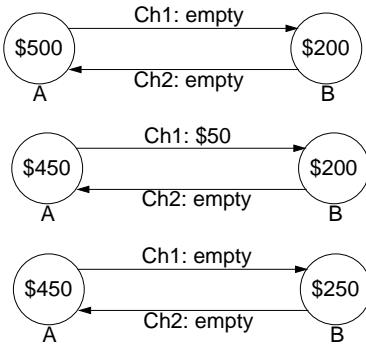


Petru Eles, IDA, LiTH



Petru Eles, IDA, LiTH

### Formal Definition (cont'd)



| A                                   | B                                   | C : consistent<br>NC: not consistent |
|-------------------------------------|-------------------------------------|--------------------------------------|
| 500                                 | 200                                 | {A,B}: strongly C                    |
| 450<br>(mess <sub>1</sub> sent)     | 200                                 | {A,B}: C                             |
| 500<br>(mess <sub>1</sub> received) | 250                                 | {A,B}: NC                            |
| 450<br>(mess <sub>1</sub> sent)     | 250<br>(mess <sub>1</sub> received) | {A,B}: strongly C                    |

- After registering of the receive event(s) a consistent state becomes strongly consistent. It is considered to be a normal (transient) situation.

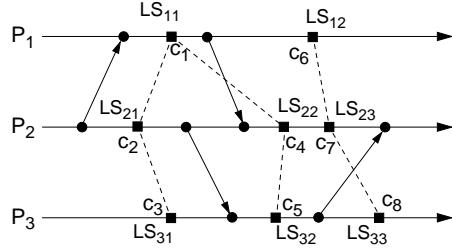


Petru Eles, IDA, LiTH

### Cuts of a Distributed Computation

☞ A **cut** is a graphical representation of a global state. A **consistent cut** is a graphical representation of a consistent global state.

- A cut of a distributed computation is a set  $Ct = \{c_1, c_2, \dots, c_n\}$ , where  $c_i$  is the cut event at process  $P_i$ .
- A cut event is the event of recording a local state of the respective process.

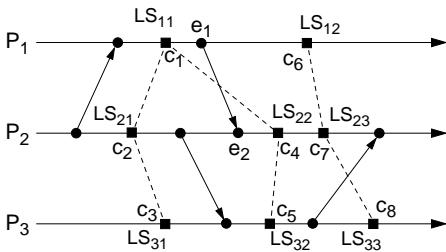


Petru Eles, IDA, LiTH

### Cuts of a Distributed Computation (cont'd)

☞ Let  $e_k$  denote an event at process  $P_k$ . A cut  $Ct = \{c_1, c_2, \dots, c_n\}$  is a consistent cut if and only if  $\forall P_i \forall P_j \exists e_i \exists e_j$  such that  $(e_i \rightarrow e_j) \wedge (e_j \rightarrow c_i) \wedge \neg(e_i \rightarrow c_j)$

- A cut is consistent if every message that was received before a cut event was sent before the cut event at the sender process.



$\{c_1, c_4, c_5\}$  is not consistent:  $(e_1 \rightarrow e_2) \wedge (e_2 \rightarrow c_4) \wedge \neg(e_1 \rightarrow c_5)$



Petru Eles, IDA, LiTH

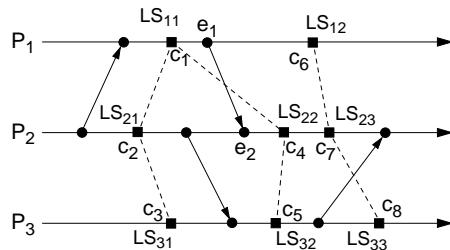
### Cuts of a Distributed Computation (cont'd)

#### Theorem

A cut  $Ct = \{c_1, c_2, \dots, c_n\}$  is a consistent cut if and only if no two cut events are causally related, that is:

$$\forall c_i \forall c_j :: \neg(c_i \rightarrow c_j) \wedge \neg(c_j \rightarrow c_i)$$

- A set of concurrent cut events form a consistent cut.



$\{c_1, c_2, c_3\}$ : strongly consistent (no communication line is crossed)

$\{c_6, c_7, c_8\}$ : consistent (communication line is crossed but no causal relation).

$\{c_1, c_4, c_5\}$ : not consistent;  $c_1 \rightarrow c_4$



Petru Eles, IDA, LiTH

## Global State Recording (Chandy-Lamport Algorithm)

- The algorithm records a collection of local states which give a consistent global state of the system. *In addition it records the state of the channels which is consistent with the collected global state.*
- Such a recorded "view" of the system is called a *snapshot*.
- We assume that processes are connected through one directional channels and message delivery is FIFO.
- We assume that the graph of processes and channels is strongly connected (there exists a path between any two processes).
- The algorithm is based on the use of a special message, *snapshot token*, in order to control the state collection process.



## Global State Recording (cont'd)

Some discussion on how to collect a global state:

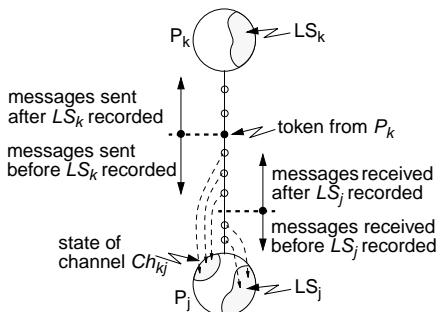
- A process  $P_i$  records its local state  $LS_i$  and later sends a message  $m$  to  $P_j$ ;  $LS_i$  at  $P_j$  has to be recorded before  $P_j$  has received  $m$ .
- The state  $SCh_{ij}$  of the channel  $Ch_{ij}$  consists of all messages that process  $P_i$  sent before recording  $LS_i$  and which have not been received by  $P_j$  when recording  $LS_j$ .
- A snapshot is started at the request of a particular process  $P_i$ ; for example, when it suspects a deadlock because of long delay in accessing a resource;  $P_i$  then records its state  $LS_i$  and, before sending any other message, it sends a token to every  $P_j$  that  $P_i$  communicates with.
- When  $P_j$  receives a token from  $P_i$ , and this is the first time it received a token, it must record its state before it receives the next message from  $P_i$ . After recording its state  $P_j$  sends a token to every process it communicates with, before sending them any other message.



## Global State Recording (cont'd)

What about the channel states?

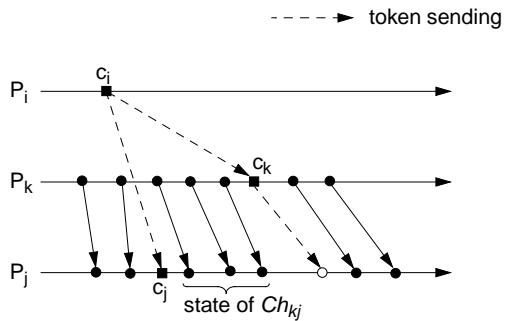
- $P_i$  sends a token to  $P_j$  and this is the first time  $P_j$  received a token  $\Rightarrow P_j$  immediately records its state. All the messages sent by  $P_i$  before sending the token have been received at  $P_j \Rightarrow SCh_{ij} := \emptyset$ .
- $P_i$  receives a token from  $P_k$ , but  $P_j$  already recorded its state.  $M$  is the set of messages that  $P_j$  received from  $P_k$  after  $P_i$  recorded its state and before  $P_j$  received the token from  $P_k \Rightarrow SCh_{kj} := M$ .



- The algorithm terminates when all processes have received tokens on all their input channels.
- The process that initiated the snapshot should be informed; it can collect the global snapshot.

## Global State Recording (cont'd)

Maybe, you prefer this view:



- Don't forget when you look to the picture: we assumed that message passing on a channel connecting two processes is FIFO.



## Global State Recording (cont'd)

### The algorithm

☞ Rule for sender  $P_i$ :

```
/* performed by the initiating process and by any
   other process at the reception of the first token */
[SR1]:  $P_i$  records its state.
[SR2]:  $P_i$  sends a token on each of its outgoing channels.
```

☞ Rule for receiver  $P_j$ :

```
/* executed whenever  $P_j$  receives a token from
   another process  $P_i$  on channel  $Ch_{ij}$  */
[RR1]: if  $P_j$  has not yet recorded its state then
    Record the state of the channel:  $SCh_{ij} := \emptyset$ .
    Follow the "Rule for sender".
else
    Record the state of the channel:  $SCh_{ij} := M$ ,
    where  $M$  is the set of messages that  $P_j$ 
    received from  $P_i$  after  $P_i$  recorded its state
    and before  $P_j$  received the token on  $Ch_{ij}$ .
end if.
```



## Summary

- In a distributed system there is no exact notion of global physical time. Physical clocks can be synchronized to a certain accuracy.
- In many applications not physical time is important but only the relative ordering of certain events. Such an ordering can be achieved using logical clocks.
- Lamport's logical clocks are implemented using a monotonic integer counter at each site. They can be used in order to capture the happened-before relation.
- The main problem with Lamport's clocks is that they are not powerful enough to perform a causal ordering of events.
- Vector clocks give the ability to decide whether two events are causally related or not, by simply looking at their timestamps.



## Summary (cont'd)

- As there doesn't exist a global notion of physical time, it is very difficult to reason about a global state in a distributed system.
- We can consider a global state as a collection of local states and, possibly, a set of states of the communication channels.
- A global state can be consistent or not.
- A cut is a graphical representation of a global state. Using cuts it is easy to elegantly reason about consistency of global states.
- It is possible to record local states and states of the channels, so that together they provide a consistent view of the system. Such a view is called a snapshot.



## DISTRIBUTED MUTUAL EXCLUSION AND ELECTION

- 1. Mutual Exclusion in Distributed Systems**
- 2. Non-Token-Based Algorithms**
- 3. Token-Based Algorithms**
- 4. Distributed Election**
- 5. The Bully and the Ring-Based Algorithms**



Petru Eles, IDA, LiTH

### Mutual Exclusion

- ☞ Mutual exclusion ensures that concurrent processes make a serialized access to shared resources or data.



The well known *critical section* problem!

- ☞ In a distributed system neither shared variables (semaphores) nor a local kernel can be used in order to implement mutual exclusion!  
Thus, mutual exclusion has to be based exclusively on message passing, in the context of unpredictable message delays and no complete knowledge of the state of the system.
- ☞ Sometimes the resource is managed by a server which implements its own *lock* together with the mechanisms needed to synchronize access to the resource ⇒ mutual exclusion and the related synchronization are transparent for the process accessing the resource.  
This is typically the case for database systems with *transaction processing* (see *concurrency control* in Database course!)



Petru Eles, IDA, LiTH

### Mutual Exclusion (cont'd)

- ☞ Often there is no synchronization built in which implicitly protects the resource (files, display windows, peripheral devices, etc.).
- A mechanism has to be implemented at the level of the process requesting for access.
- ☞ Basic requirements for a mutual exclusion mechanism:
  - **safety:** at most one process may execute in the critical section (CS) at a time;
  - **liveness:** a process requesting entry to the CS is eventually granted it (so long as any process executing the CS eventually leaves it).  
Liveness implies freedom of *deadlock* and *starvation*.



- ☞ There are two basic approaches to distributed mutual exclusion:
  1. **Non-token-based:** each process freely and equally competes for the right to use the shared resource; requests are arbitrated by a central control site or by distributed agreement.
  2. **Token-based:** a logical token representing the access right to the shared resource is passed in a regulated fashion among the processes; whoever holds the token is allowed to enter the critical section.



Petru Eles, IDA, LiTH

### Non-Token-Based Mutual Exclusion

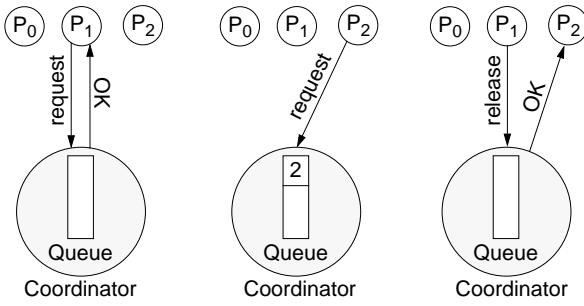
- Central Coordinator Algorithm
- Ricart-Agrawala Algorithm



Petru Eles, IDA, LiTH

### Central Coordinator Algorithm

- ☞ A central coordinator grants permission to enter a CS.



- To enter a CS, a process sends a request message to the coordinator and then waits for a reply (during this waiting period the process can continue with other work).
- The reply from the coordinator gives the right to enter the CS.
- After finishing work in the CS the process notifies the coordinator with a release message.



Petru Eles, IDA, LiTH

### Central Coordinator Algorithm (cont'd)

- ☞ The scheme is simple and easy to implement.
- ☞ The strategy requires only three messages per use of a CS (*request, OK, release*).

#### Problems

- The coordinator can become a performance bottleneck.
- The coordinator is a critical point of failure:
  - If the coordinator crashes, a new coordinator must be created.
  - *The coordinator can be one of the processes competing for access; an election algorithm (see later) has to be run in order to choose one and only one new coordinator.*



Petru Eles, IDA, LiTH

### Ricart-Agrawala Algorithm

- ☞ In a distributed environment it seems more natural to implement mutual exclusion, based upon distributed agreement - not on a central coordinator.
- ☞ It is assumed that all processes keep a (Lamport's) logical clock which is updated according to the rules in Fö. 5, slide 8.
- The algorithm requires a total ordering of requests ⇒ requests are ordered according to their global logical timestamps; if timestamps are equal, process identifiers are compared to order them (see Fö. 5, slide 10).
- ☞ The process that requires entry to a CS multicasts the request message to all other processes competing for the same resource; it is allowed to enter the CS when all processes have replied to this message. The request message consists of the requesting process' timestamp (logical clock) and its identifier.
- ☞ Each process keeps its state with respect to the CS: *released, requested, or held*.



Petru Eles, IDA, LiTH

### Ricart-Agrawala Algorithm (cont'd)

#### The Algorithm

- ☞ Rule for process initialization
  - /\* performed by each process  $P_i$  at initialization \*/
  - [RI1]:  $state_{P_i} := RELEASED$ .
- ☞ Rule for access request to CS
  - /\* performed whenever process  $P_i$  requests an access to the CS \*/
  - [RA1]:  $state_{P_i} := REQUESTED$ .  
 $T_{P_i} :=$  the value of the local logical clock corresponding to this request.
  - [RA2]:  $P_i$  sends a request message to all processes; the message is of the form  $(T_{P_i}, i)$ , where  $i$  is an identifier of  $P_i$ .
  - [RA3]:  $P_i$  waits until it has received replies from all other  $n-1$  processes.
- ☞ Rule for executing the CS
  - /\* performed by  $P_i$  after it received the  $n-1$  replies \*/
  - [RE1]:  $state_{P_i} := HELD$ .  
 $P_i$  enters the CS.



Petru Eles, IDA, LiTH

### Ricart-Agrawala Algorithm (cont'd)

☞ Rule for handling incoming requests

```
/* performed by  $P_i$  whenever it received a request
   ( $T_{P_j}, j$ ) from  $P_j$  */
[RH1]: if  $state_{P_i} = HELD$  or (( $state_{P_i} = REQUESTED$ )
   and ( $T_{P_i}, i < T_{P_j}, j$ )) then
   Queue the request from  $P_j$  without replying.
else
   Reply immediately to  $P_j$ .
end if.
```

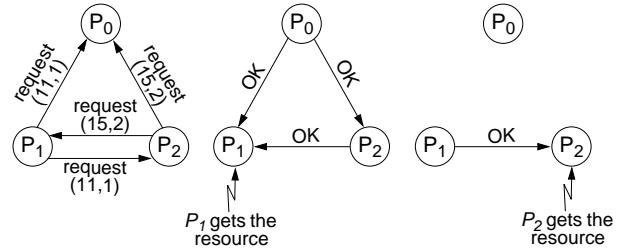
☞ Rule for releasing a CS

```
/* performed by  $P_i$  after it finished work in a CS */
[RR1]:  $state_{P_i} := RELEASED$ .
 $P_i$  replies to all queued requests.
```



### Ricart-Agrawala Algorithm (cont'd)

- A request issued by a process  $P_j$  is blocked by another process  $P_i$  only if  $P_i$  is holding the resource or if it is requesting the resource with a higher priority (this means a smaller timestamp) than  $P_j$ .



### Problems

- The algorithm is expensive in terms of message traffic; it requires  $2(n-1)$  messages for entering a CS:  $(n-1)$  requests and  $(n-1)$  replies.
- The failure of any process involved makes progress impossible if no special recovery measures are taken.



### Token-Based Mutual Exclusion

- Ricart-Agrawala Second Algorithm
- Token Ring Algorithm



### Ricart-Agrawala Second Algorithm

- ☞ A process is allowed to enter the critical section when it got the token. In order to get the token it sends a request to all other processes competing for the same resource. The request message consists of the requesting process' timestamp (logical clock) and its identifier. Initially the token is assigned arbitrarily to one of the processes.

- ☞ When a process  $P_i$  leaves a critical section it passes the token to one of the processes which are waiting for it; this will be the first process  $P_j$ , where  $j$  is searched in order  $[+1, +2, \dots, n, 1, 2, \dots, i-2, i-1]$  for which there is a pending request. If no process is waiting,  $P_i$  retains the token (and is allowed to enter the CS if it needs); it will pass over the token as result of an incoming request.

- ☞ How does  $P_i$  find out if there is a pending request?

Each process  $P_i$  records the timestamp corresponding to the last request it got from process  $P_j$  in  $request_{P_i}[j]$ . In the token itself,  $token[j]$  records the timestamp (logical clock) of  $P_j$ 's last holding of the token. If  $request_{P_i}[j] > token[j]$  then  $P_j$  has a pending request.



### Ricart-Agrawala Second Algorithm (cont'd)

#### The Algorithm

☞ Rule for process initialization

/\* performed at initialization \*/

[RI1]:  $state_{P_i} := \text{NO-TOKEN}$  for all processes  $P_i$ , except one single process  $P_x$  for which  $state_{P_x} := \text{TOKEN-PRESENT}$ .

[RI2]:  $token[k]$  initialized 0 for all elements  $k = 1 \dots n$ .  $request_{P_i}[k]$  initialized 0 for all processes  $P_i$  and all elements  $k = 1 \dots n$ .

☞ Rule for access request and execution of the CS

/\* performed whenever process  $P_i$  requests an access to the CS and when it finally gets it; in particular  $P_i$  can already possess the token \*/

[RA1]: if  $state_{P_i} = \text{NO-TOKEN}$  then

$P_i$  sends a request message to all processes; the message is of the form  $(T_{P_i}, i)$ , where  $T_{P_i} = C_{P_i}$  is the value of the local logical clock, and  $i$  is an identifier of  $P_i$ .

$P_i$  waits until it receives the token.

end if.

$state_{P_i} := \text{TOKEN-HELD}$ .

$P_i$  enters the CS.



Petru Eles, IDA, LiTH

### Ricart-Agrawala Second Algorithm (cont'd)

☞ Rule for handling incoming requests

/\* performed by  $P_j$  whenever it received a request  $(T_{P_j}, j)$  from  $P_i$  \*/

[RH1]:  $request_{P_j}[j] := \max(request_{P_j}[j], T_{P_j})$ .

[RH2]: if  $state_{P_j} = \text{TOKEN-PRESENT}$  then

$P_j$  releases the resource (see rule RR2).  
end if.

☞ Rule for releasing a CS

/\* performed by  $P_i$  after it finished work in a CS or when it holds a token without using it and it got a request \*/

[RR1]:  $state_{P_i} = \text{TOKEN-PRESENT}$ .

[RR2]: for  $k = [i+1, i+2, \dots, n, 1, 2, \dots, i-2, i-1]$  do

if  $request_{P_i}[k] > token[k]$  then

$state_{P_i} := \text{NO-TOKEN}$ .

$token[i] := C_{P_i}$  the value of the local logical clock.

$P_i$  sends the token to  $P_k$

break. /\* leave the for loop \*/

end if.

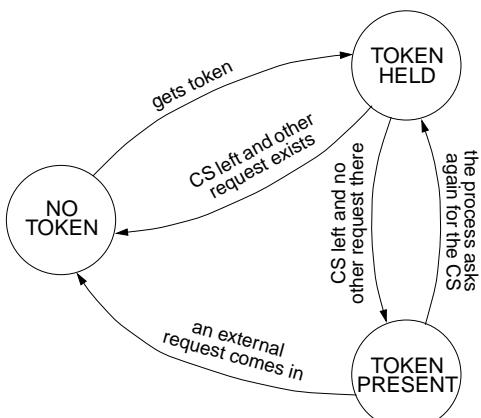
end for.



Petru Eles, IDA, LiTH

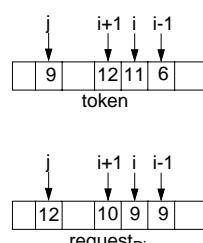
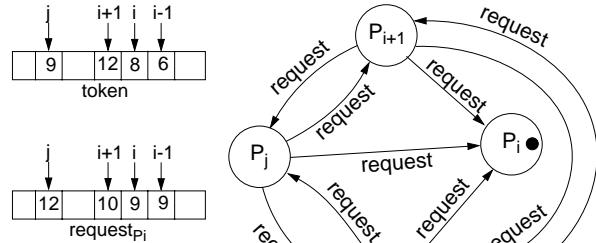
### Ricart-Agrawala Second Algorithm (cont'd)

☞ Each process keeps its state with respect to the token: NO-TOKEN, TOKEN-PRESENT, TOKEN-HOLD.



Petru Eles, IDA, LiTH

### Ricart-Agrawala Second Algorithm (cont'd)



Petru Eles, IDA, LiTH

### Ricart-Agrawala Second Algorithm (cont'd)

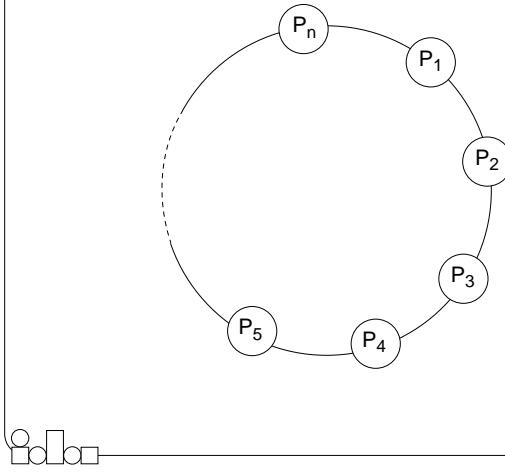
- ☞ The complexity is reduced compared to the (first) Ricart-Agrawala algorithm: it requires  $n$  messages for entering a CS:  $(n-1)$  requests and one reply.
- ☞ The failure of a process, except the one which holds the token, doesn't prevent progress.



Petru Eles, IDA, LiTH

### Token Ring Algorithm

- ☞ A very simple way to solve mutual exclusion  $\Rightarrow$  arrange the  $n$  processes  $P_1, P_2, \dots, P_n$  in a logical ring.
- ☞ The logical ring topology is created by giving each process the address of one other process which is its neighbour in the clockwise direction.
- ☞ The logical ring topology is unrelated to the physical interconnections between the computers.



Petru Eles, IDA, LiTH

### Token Ring Algorithm (cont'd)

#### The algorithm

- The token is initially given to one process.
- The token is passed from one process to its neighbour round the ring.
- When a process requires to enter the CS, it waits until it receives the token from its left neighbour and then it retains it; after it got the token it enters the CS; after it left the CS it passes the token to its neighbour in clockwise direction.
- When a process receives the token but does not require to enter the critical section, it immediately passes the token over along the ring.



Petru Eles, IDA, LiTH

### Token Ring Algorithm (cont'd)

- ☞ It can take from 1 to  $n-1$  messages to obtain a token. Messages are sent around the ring even when no process requires the token  $\Rightarrow$  additional load on the network.



The algorithm works well in heavily loaded situations, when there is a high probability that the process which gets the token wants to enter the CS. It works poorly in lightly loaded cases.

- ☞ If a process fails, no progress can be made until a reconfiguration is applied to extract the process from the ring.
- ☞ If the process holding the token fails, a unique process has to be picked, which will regenerate the token and pass it along the ring; an *election algorithm* (see later) has to be run for this purpose.



Petru Eles, IDA, LiTH

## Election

- Many distributed algorithms require one process to act as a coordinator or, in general, perform some special role.

### Examples with mutual exclusion

- Central coordinator algorithm: at initialisation or whenever the coordinator crashes, a new coordinator has to be elected (see slide 6).
- Token ring algorithm: when the process holding the token fails, a new process has to be elected which generates the new token (see slide 20).



Petru Eles, IDA, LiTH

## Election (cont'd)

- We consider that it doesn't matter which process is elected; what is important is that one and only one process is chosen (we call this process the coordinator) and all processes agree on this decision.

- We assume that each process has a unique number (identifier); in general, election algorithms attempt to locate the process with the highest number, among those which currently are up.

- Election is typically started after a failure occurs. The detection of a failure (e.g. the crash of the current coordinator) is normally based on time-out  $\Rightarrow$  a process that gets no response for a period of time suspects a failure and initiates an election process.

- An election process is typically performed in two phases:

- Select a leader with the highest priority.
- Inform all processes about the winner.



Petru Eles, IDA, LiTH

## The Bully Algorithm

- A process has to know the identifier of all other processes (it doesn't know, however, which one is still up); the process with the highest identifier, among those which are up, is selected.
- Any process could fail during the election procedure.
- When a process  $P_i$  detects a failure and a coordinator has to be elected, it sends an *election message* to all the processes with a higher identifier and then waits for an *answer message*:
  - If no response arrives within a time limit,  $P_i$  becomes the coordinator (all processes with higher identifier are down)  $\Rightarrow$  it broadcasts a *coordinator message* to all processes to let them know.
  - If an *answer message* arrives,  $P_i$  knows that another process has to become the coordinator  $\Rightarrow$  it waits in order to receive the *coordinator message*. If this message fails to arrive within a time limit (which means that a potential coordinator crashed after sending the *answer message*)  $P_i$  resends the *election message*.
- When receiving an *election message* from  $P_i$ , a process  $P_j$  replies with an *answer message* to  $P_i$  and then starts an election procedure itself, unless it has already started one  $\Rightarrow$  it sends an *election message* to all processes with higher identifier.
- Finally all processes get an *answer message*, except the one which becomes the coordinator.



Petru Eles, IDA, LiTH

## The Bully Algorithm (cont'd)

### The Algorithm

- By default, the state of a process is ELECTION-OFF

- Rule for election process initiator

/\* performed by a process  $P_i$ , which triggers the election procedure, or which starts an election after receiving itself an *election message* \*/

[RE1]:  $state_{P_i} := \text{ELECTION-ON}$ .

$P_i$  sends an *election message* to all processes with a higher identifier.

$P_i$  waits for *answer message*.

**if** no *answer message* arrives before time-out **then**

$P_i$  is the coordinator and sends a *coordinator message* to all processes.

**else**

$P_i$  waits for a *coordinator message* to arrive.

**if** no *coordinator message* arrives before time-out **then**

restart election procedure according to RE1

**end if**

**end if**.



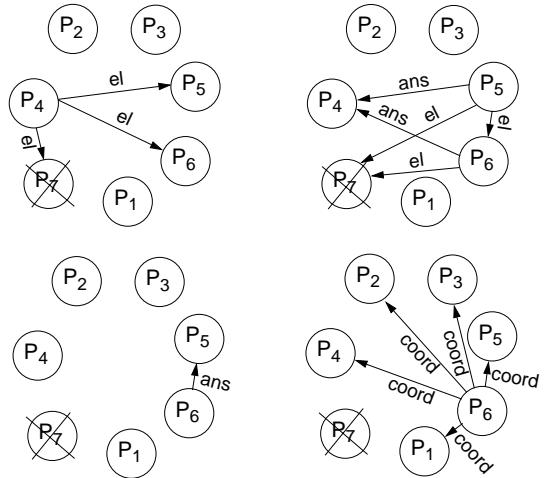
Petru Eles, IDA, LiTH

### The Bully Algorithm (cont'd)

- ☞ Rule for handling an incoming *election message*  
/\* performed by a process  $P_i$  at reception of an *election message* coming from  $P_j$  \*/
- [RH1]:  $P_i$  replies with an *answer message* to  $P_j$ .
- [RH2]: if  $\text{state}_{P_i} := \text{ELECTION-OFF}$  then  
start election procedure according to RE1  
end if



### The Bully Algorithm (cont'd)



- If  $P_6$  crashes before sending the coordinator message,  $P_4$  and  $P_5$  restart the election process.
- ☞ The best case: the process with the second highest identifier notices the coordinator's failure. It can immediately select itself and then send  $n-2$  coordinator messages.
- ☞ The worst case: the process with the lowest identifier initiates the election; it sends  $n-1$  election messages to processes which themselves initiate each one an election  $\Rightarrow O(n^2)$  messages.



### The Ring-Based Algorithm

- ☞ We assume that the processes are arranged in a logical ring; each process knows the address of one other process, which is its neighbour in the clockwise direction.
- ☞ The algorithm elects a single coordinator, which is the process with the highest identifier.
- ☞ Election is started by a process which has noticed that the current coordinator has failed. The process places its identifier in an *election message* that is passed to the following process.
- ☞ When a process receives an *election message* it compares the identifier in the message with its own. If the arrived identifier is greater, it forwards the received *election message* to its neighbour; if the arrived identifier is smaller it substitutes its own identifier in the *election message* before forwarding it.
- ☞ If the received identifier is that of the receiver itself  $\Rightarrow$  this will be the coordinator. The new coordinator sends an *elected message* through the ring.



### The Ring-Based Algorithm (cont'd)

- ☞ An optimization
  - Several elections can be active at the same time. Messages generated by later elections should be killed as soon as possible.
- ↓
- Processes can be in one of two states: *participant* and *non-participant*. Initially, a process is *non-participant*
- The process initiating an election marks itself *participant*.
- A *participant* process, in the case that the identifier in the *election message* is smaller than the own, does not forward any message (it has already forwarded it, or a larger one, as part of another simultaneously ongoing election).
- When forwarding an *election message*, a process marks itself *participant*.
- When sending (forwarding) an *elected message*, a process marks itself *non-participant*.



### The Ring-Based Algorithm (cont'd)

#### The Algorithm

- ☞ By default, the state of a process is NON-PARTICIPANT
- ☞ Rule for election process initiator
 

```
/* performed by a process  $P_i$ , which triggers the election procedure */
```
- [RE1]:  $state_{P_i} := \text{PARTICIPANT}$ .
- [RE2]:  $P_i$  sends an *election message* with  $message.id := i$  to its neighbour.
- ☞ Rule for handling an incoming *election message*

```
/* performed by a process  $P_j$ , which receives an election message */
```
- [RH1]: **if**  $message.id > j$  **then**
  - $P_j$  forwards the received *election message*.
  - $state_{P_j} := \text{PARTICIPANT}$ .
  - elseif**  $message.id < j$  **then**
    - if**  $state_{P_j} = \text{NON-PARTICIPANT}$  **then**
      - $P_j$  forwards an *election message* with  $message.id := j$ .
      - $state_{P_j} := \text{PARTICIPANT}$
    - end if**
  - else**
    - $P_j$  is the coordinator and sends an *elected message* with  $message.id := j$  to its neighbour.
    - $state_{P_j} := \text{NON-PARTICIPANT}$ .
  - end if.**



Petru Eles, IDA, LiTH

### The Ring-Based Algorithm (cont'd)

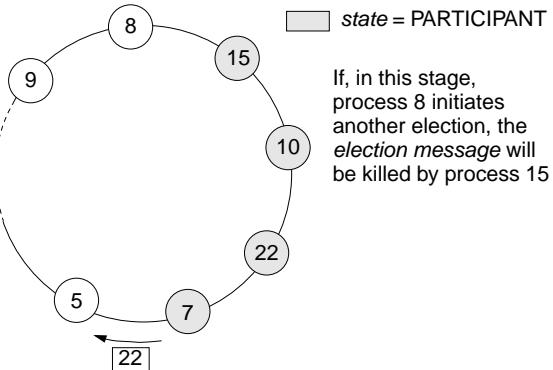
- ☞ Rule for handling an incoming *elected message*

```
/* performed by a process  $P_i$ , which receives an elected message */
```
- [RD1]: **if**  $message.id \neq i$  **then**
  - $P_i$  forwards the received *elected message*.
  - $state_{P_i} := \text{NON-PARTICIPANT}$ .
  - end if.**



Petru Eles, IDA, LiTH

### The Ring-Based Algorithm (cont'd)



- ☞ With one single election started:
  - On average:  $n/2$  (election) messages needed to reach maximal node;  $n$  (election) messages to return to maximal node;  $n$  messages to rotate *elected message*.  
Number of messages:  $2n + n/2$ .
  - Worst case:  $n-1$  messages needed to reach maximal node;  
Number of messages:  $3n - 1$ .
- ☞ The ring algorithm is more efficient on average than the bully algorithm.

### Summary

- In a distributed environment no shared variables (semaphores) and local kernels can be used to enforce mutual exclusion. Mutual exclusion has to be based only on message passing.
- There are two basic approaches to mutual exclusion: non-token-based and token-based.
- The central coordinator algorithm is based on the availability of a coordinator process which handles all the requests and provides exclusive access to the resource. The coordinator is a performance bottleneck and a critical point of failure. However, the number of messages exchanged per use of a CS is small.
- The Ricart-Agrawala algorithm is based on fully distributed agreement for mutual exclusion. A request is multicast to all processes competing for a resource and access is provided when all processes have replied to the request. The algorithm is expensive in terms of message traffic, and failure of any process prevents progress.
- Ricart-Agrawala's second algorithm is token-based. Requests are sent to all processes competing for a resource but a reply is expected only from the process holding the token. The complexity in terms of message traffic is reduced compared to the first algorithm. Failure of a process (except the one holding the token) does not prevent progress.



Petru Eles, IDA, LiTH



Petru Eles, IDA, LiTH

### Summary (cont'd)

- The token-ring algorithm very simply solves mutual exclusion. It is requested that processes are logically arranged in a ring. The token is permanently passed from one process to the other and the process currently holding the token has exclusive right to the resource. The algorithm is efficient in heavily loaded situations.
- For many distributed applications it is needed that one process acts as a coordinator. An election algorithm has to choose one and only one process from a group, to become the coordinator. All group members have to agree on the decision.
- The bully algorithm requires the processes to know the identifier of all other processes; the process with the highest identifier, among those which are up, is selected. Processes are allowed to fail during the election procedure.
- The ring-based algorithm requires processes to be arranged in a logical ring. The process with the highest identifier is selected. On average, the ring-based algorithm is more efficient than the bully algorithm.



# Replication

30-Jun-23

COMP28112 Lecture 15

1

## What is Replication?

- Make multiple copies of a data object and ensure that all copies are identical
- Two Types of access; reads, and writes (updates)
- Reasons, have a backup plan:
  - Handle more work (e.g. web-servers)
  - Keep data safe (fault tolerance)
  - Reduce latencies (CDN's and Caching)
  - Keep data available

30-Jun-23

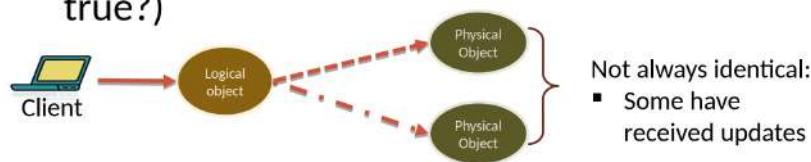
COMP28112 Lecture 15

2

1

# Replication requirements

- ❑ Transparency (illusion of a single copy)
  - Clients must be unaware of replication
- ❑ Consistency
  - Obtain identical results from different copies (is that true?)

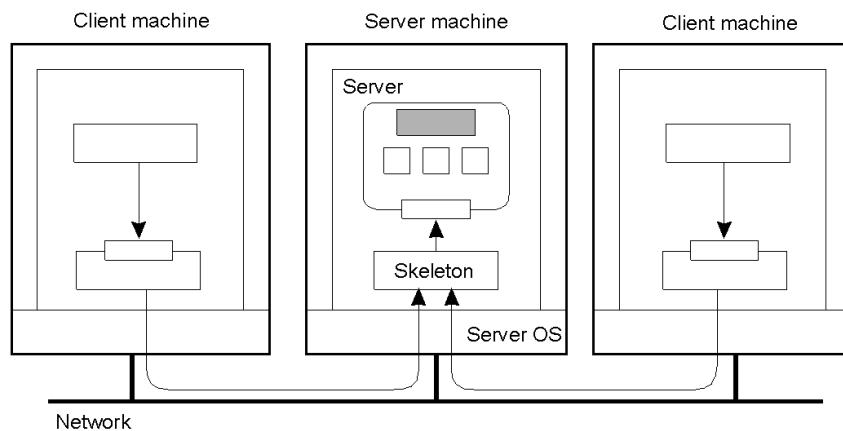


30-Jun-23

COMP28112 Lecture 15

3

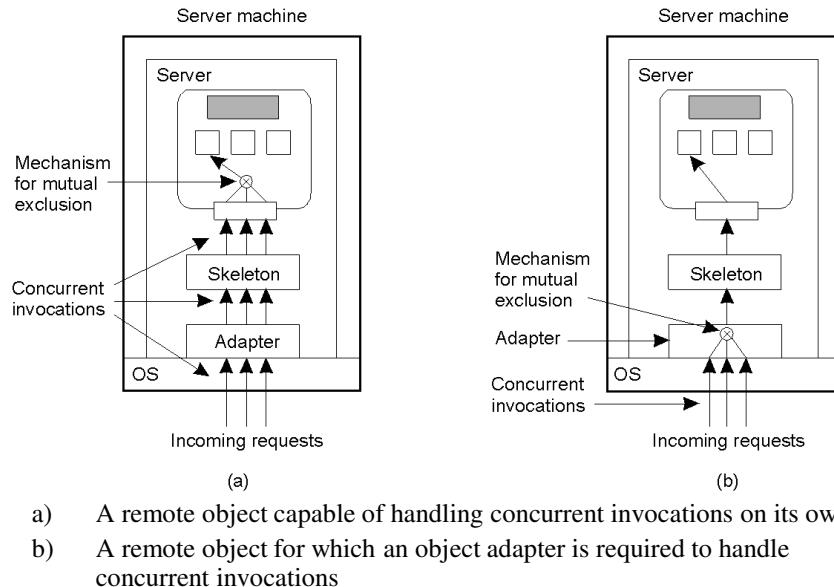
## Object Replication (1)



Organization of a distributed remote object shared by two different clients.

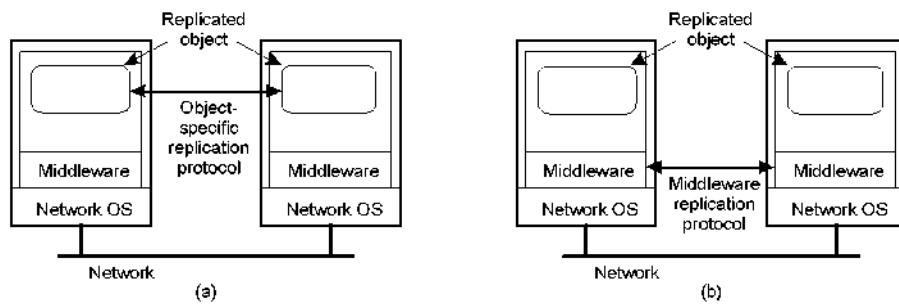
2

## Object Replication (2)



- a) A remote object capable of handling concurrent invocations on its own.
- b) A remote object for which an object adapter is required to handle concurrent invocations

## Object Replication (3)



- a) A distributed system for replication-aware distributed objects.
- b) A distributed system responsible for replica management

## Reasons for Replication (1)

- Reliability

- **Redundancy** is a key technique to increase availability. If one server crashes, then there is a replica that can still be used. Thus, failures are tolerated by the use of redundant components. Examples:

- There should always be at least two different routes between any two routers in the internet.
    - In the Domain Name System, every name table is replicated in at least two different servers.
    - A database may be replicated in several servers to ensure that the data remains accessible after the failure of any single server.

*Trade-off: there is some cost associated with the maintenance of different replicas.*

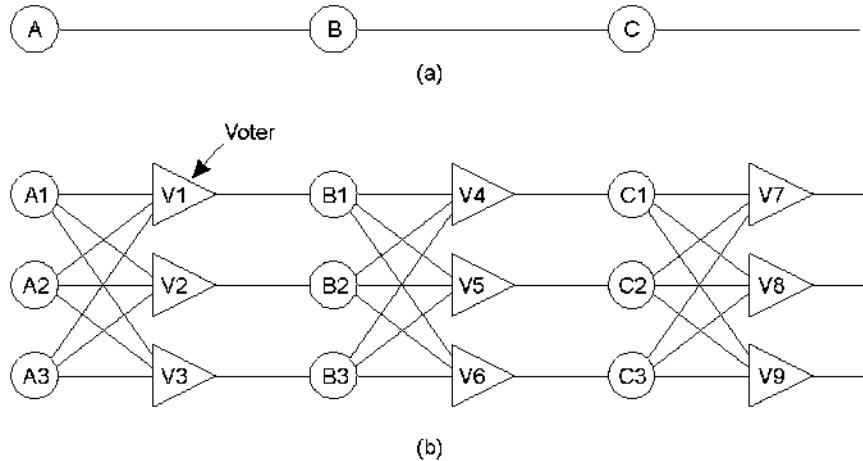
## Redundancy

- Wouldn't you prefer to use an **expensive** airplane with triple-redundancy for all hardware resources [1] as opposed to a **cheap** airplane with a single-resource (no redundancy)?
- **Availability** of a replicated service over a period of time:
  - $1 - \text{Probability}(\text{all replicas have failed})$
  - $\text{Probability}(\text{all replicas have failed}) = \text{Probability}(\text{replica 1 has failed}) \times \text{Probability}(\text{replica 2 has failed}) \times \text{Probability}(\text{replica 3 has failed})$ .
  - Probabilities are between 0 (=no chance/impossible) and 1 (=certainty).
- **Example:** if the probability of a system failing is 0.3 (30%), the probability of two copies of the system failing at the same time is  $0.3 \times 0.3 = 0.09$ , the probability of three copies all failing at the same time is 0.027, the probability of ten copies all failing at the same time is 0.000005905. This means that with ten copies we can have availability of 99.9994095%.
- To compute probability of failure ( $p$ ) of a single node, use mean time between failures ( $f$ ) and mean time to repair a failure( $t$ ):  $p = 1 - t/(f+t)$
- The degree of redundancy has to strike a balance with the additional cost needed to implement it!

[1] "Triple-triple redundant 777 primary flight computer". Proceedings of the 1996 IEEE Aerospace Applications Conference.

# Failure Masking by Redundancy

(see Tanenbaum, fig 8.2, p.327)



## Triple modular redundancy

# Reasons for Replication (2)

- Performance
  - By placing a copy of the data in the proximity of the process using them, the time to access the data decreases. This is also useful to achieve scalability. Examples:
    - A server may need to handle an increasing number of requests (e.g., google.com, ebay.com). Improve performance by replicating the server and subsequently dividing the work.
    - **Caching:** Web browsers may store locally a copy of a previously fetched web page to avoid the latency of fetching resources from the originating server. (cf. with processor architectures and cache memory)

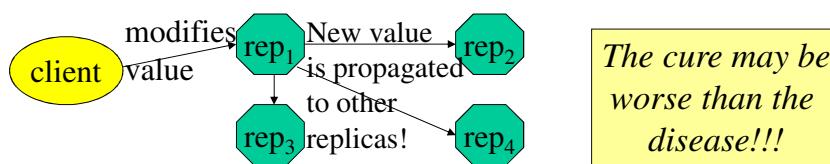
## How many replicas?

(or, how do we trade cost with availability?)

- **Capacity Planning:** the process of determining the necessary capacity to meet a certain level of demand (a concept that extends beyond distributed computing)
- Example problem:
  - Given the cost of a customer waiting in the queue;
  - Given the cost of running a server;
  - Given the expected number of requests;
  - How many servers shall we provide to guarantee a certain response time if the number of requests does not exceed a certain threshold?
- Problems like this may be solved with mathematical techniques. **Queuing theory**, which refers to the mathematical study of queues (see Gross & Harris, Fundamentals of queuing theory) may be useful.

## The price to be paid...

- Besides the cost (in terms of money) to maintain replicas, what else could be against replication?
- Consistency problems:
  - If a copy is modified, this copy becomes different from the rest. Consequently, modifications have to be carried out on all copies to ensure consistency. When and how those modifications need to be carried out determines the price of replication. Example (4 replicas):



## The price to be paid... (cont)

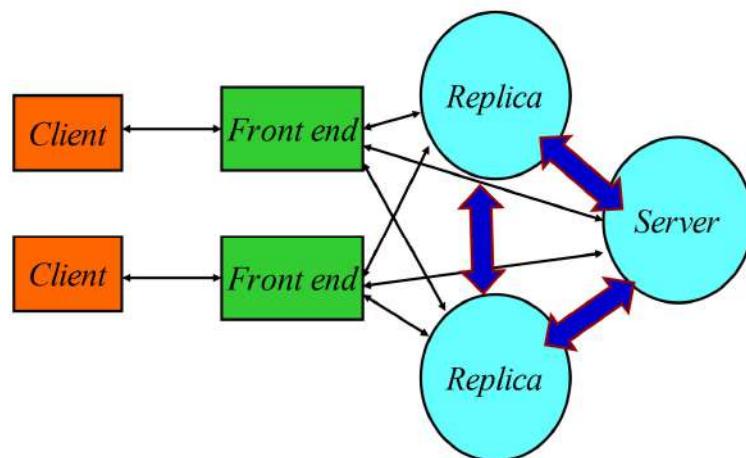
- Keeping multiple copies consistent may itself be subject to serious scalability problems!
- In the example before, when an update occurs it needs to be propagated to all other replicas. No other processes should read the same value from the other replicas before the update happened... However:
  - What if it is unlikely that there will ever be a request to read that same value from other replicas?
  - At the same time with the update, there is a request to read this value from another process – which one came first?
- Global synchronisation takes a lot of time when replicas are spread across a wide area network.
- **Solution:** Loosen the consistency constraints! So, copies may not be always the same everywhere... To what extent consistency can be loosened depends on the access and update patterns of the replicated data as well as on the purpose for which those data are used.

30-Jun-23

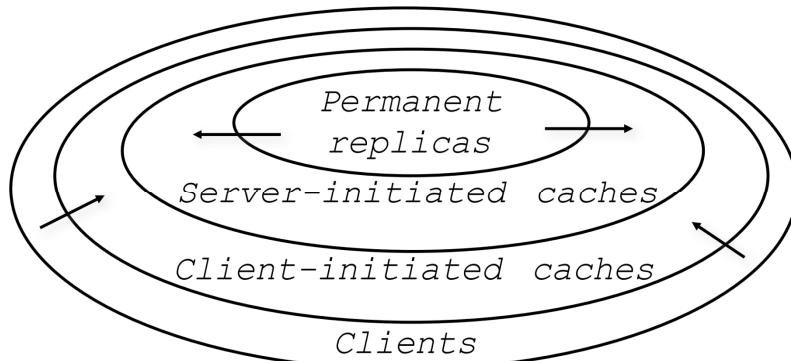
COMP28112 Lecture 15

13

## Replication Architecture



# Placement of Replicas



30-Jun-23

COMP28112 Lecture 15

15

# Placement of Replicas

- Permanent replicas
  - Clusters of servers
  - Geographically dispersed web mirrors (Akamai)
- Server-initiated caches
  - Placement of hosting servers
- Client initiated caches
  - enterprise proxies or web browser caches

30-Jun-23

COMP28112 Lecture 15

16

8

# Update propagation in replicas

- Push based propagation
  - A replica pushes the update to the others
  - May push the new data or parameters of the update operation
- Pull-based propagation
  - A replica requests another replica to send the newest data it has
- Pushing data vs. pushing updates
  - Pushing updates reduces traffic

## Types of ordering adapted to replication

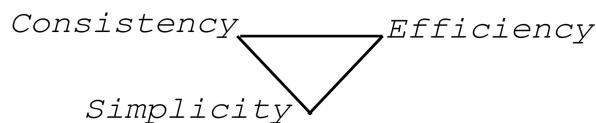
FIFO- if a client issues  $r$  and then  $r'$ , any correct Replica Manager that handles  $r'$  handles  $r$  before it

Causal- if the issuing of  $r$  happened-before issuing  $r'$ , then any correct Replica Manager that handles  $r'$  handles  $r$  before it

Total – if a correct Replica Manager handles  $r$  before  $r'$ , then any correct RM that handles  $r'$  handles  $r$  before it

# Consistency issues

- A contract between the client developer and a provider of the replicated service
  - The provider guarantees that the data will be updated according to some consistency criteria
  - The application developer will need to devise applications with these criteria in mind
- “Ideal consistency”: system behavior is indistinguishable from a non-replicated system
- The consistency-efficiency-simplicity triangle



# Homework

- Study different consistency models used in replication

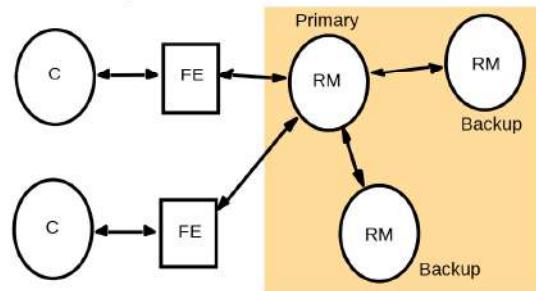
# Passive (primary-backup) replication

- ❑ One *primary* replica manager, many *backup* replicas
  - If primary fails, backups can take its place (election!)

- ❑ Implements linearizability if:

- A failing primary is replaced by a unique backup
- Backups agree on which operations were performed before primary crashed

- ❑ View-synchronous group communication!

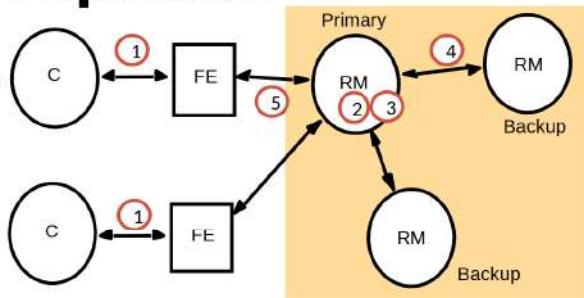


# Passive (primary-backup) replication

- One server plays a special primary role
  - Performs all the updates
  - May propagate them to backup replicas eagerly or lazily
  - Maintains the most updated state
- Backup servers may take off the load of processing client requests but only if stale results are ok
- Implementable without deterministic operations
- Typically easier to implement than active replication
- Less network traffic during the normal operation but longer recovery with possible data loss
- Several sub-schemes (cold backup, warm backup, hot standby)

# Steps of passive replication

1. Request
  - Front end issues request with unique ID
2. Coordination
  - Primary checks if request has been carried out, if so, returns cached response
3. Execution
  - Perform operation, cache results
4. Agreement
  - Primary sends updated state to backups, backups reply with Ack.
5. Response
  - Primary sends result to front end, which forwards to the client



What happens if the primary RM crashes?

- Before agreement
- After agreement

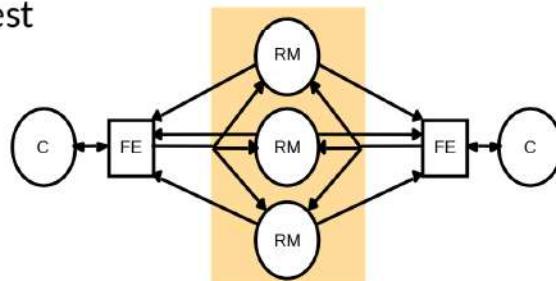
30-Jun-23

COMP28112 Lecture 15

23

# Active replication

- RMs play equivalent roles
- All replica managers carry out all operations
- Front ends multicast one request at a time (FIFO)
- Requests are totally ordered
- Implements sequential consistency
- Tolerate Byzantine failures



30-Jun-23

COMP28112 Lecture 15

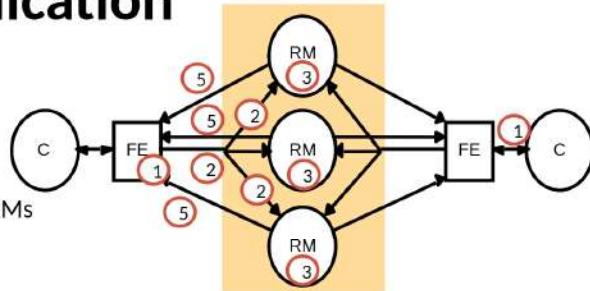
24

12

## Steps of active replication

### 1. Request

- ❑ Front end adds unique identifier to request, multicasts to RMs



### 2. Coordination

- ❑ Totally ordered request delivery to RMs

### 3. Execution

- ❑ Each RM executes request

### 4. Agreement

- ❑ Not needed

### 5. Response

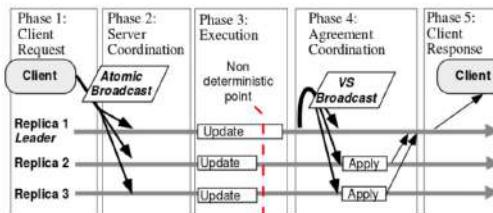
- ❑ All RMs respond to front end, front end interprets response and forwards response to client

## Comparing active and passive replication

- ❑ Both handle crash failures (but differently)
- ❑ Only active can handle arbitrary failures
- ❑ Passive may suffer from large overheads
- ❑ Optimizations?
  - Send “reads” to backups **in passive**  
Lose linearizability property!
  - Send “reads” to specific RM **in active**  
Lose fault tolerance
  - Exploit commutativity of requests to avoid ordering requests **in active**

# Semi Active Replication

- Intermediate solution between Active and Passive replication
- Main difference with active replication
  - each time replicas have to make a non-deterministic decision, a process, called the leader , makes the choice and sends it to the followers

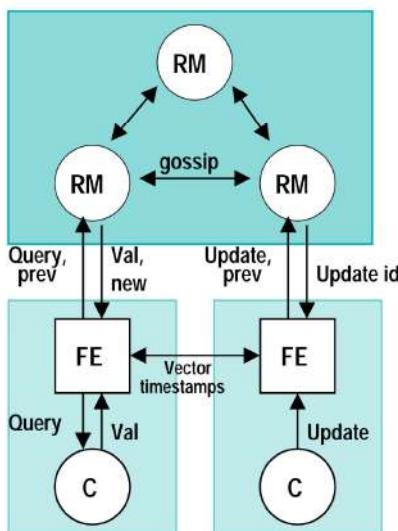


30-Jun-23

COMP28112 Lecture 15

27

## The Gossip Architecture



The gossip service front end handles client operations using an application-specific API and turns them into gossip operations (queries or updates). RM updates are **lazy** in the sense that gossip messages may be exchanged only occasionally. Each front end keeps a **vector timestamp** *prev*, reflecting the latest data values accessed by the client/front end. When clients communicate directly they piggyback their vector timestamps, which are then merged.

30-Jun-23

COMP28112 Lecture 15

28

14

# Gossip Architecture

## Processing requests

1. *Request:* The front end sends the request to a RM. Queries are usually synchronous but updates are asynchronous: the client continues as soon as the request is passed to the front end and the front end propagates the request in background.
2. *Update response:* The RM replies to the front end as soon as it has received an update.
3. *Coordination:* The RM that receives a request does not process it until it can apply the request according to the required ordering constraints. This may involve receiving updates from other replica managers, in gossip messages.
4. *Execution:* The RM executes the request.
5. *Query response:* If the request is a query the RM responds at this point.
6. *Agreement:* The RMs only coordinate via gossip messages.

# Fault Tolerance and Reliability in DS

## Fault Tolerance

- Basic concepts in fault tolerance
- Masking failure by redundancy
- Process resilience

# Motivation

- Single machine systems
  - Failures are all or nothing
    - OS crash, disk failures
- Distributed systems: multiple independent nodes
  - Partial failures are also possible (some nodes fail)
- *Question:* Can we automatically recover from partial failures?
  - Important issue since probability of failure grows with number of independent components (nodes) in the systems
  - $\text{Prob(failure)} = \text{Prob(Any one component fails)} = 1 - \text{P(no failure)}$

# A Perspective

- Computing systems are not very reliable
  - OS crashes frequently (Windows), buggy software, unreliable hardware, software/hardware incompatibilities
  - Until recently: computer users were “tech savvy”
    - Could depend on users to reboot, troubleshoot problems
  - Growing popularity of Internet/World Wide Web
    - “Novice” users
    - Need to build more reliable/dependable systems
  - Example: what is your TV (or car) broke down every day?
    - Users don’t want to “restart” TV or fix it (by opening it up)
- Need to make computing systems more reliable

# Basic Concepts

- Fault – physical defect, imperfection, or flaw that occurs within hardware or software unit.
- Error – manifestation of a fault. Deviation from accuracy or correctness.
- Failure – if error results in the system performing one of its functions incorrectly.

## Basic Concepts (cont'd)

- Need to build *dependable* systems
- Requirements for dependable systems
  - Availability: system should be available for use at any given time
    - 99.999 % availability (five 9s) => very small down times
  - Reliability: system should run continuously without failure
  - Safety: temporary failures should not result in a catastrophic
    - Example: computing systems controlling an airplane, nuclear reactor
  - Maintainability: a failed system should be easy to repair
  - Security: avoidance or tolerance of deliberate attacks to the system

# Basic Concepts (cont'd)

- Fault tolerance: system should provide services despite faults
  - Transient faults
  - Intermittent faults
  - Permanent faults

## Failure Models

| Type of failure   | Description  |
|---|--|
| Crash failure   | A server halts, but is working correctly until it halts  |
| Omission failure<br><i>Receive omission</i><br><i>Send omission</i>         | A server fails to respond to incoming requests<br>A server fails to receive incoming messages<br>A server fails to send messages |
| Timing failure  | A server's response lies outside the specified time interval   |
| Response failure<br><i>Value failure</i><br><i>State transition failure</i> | The server's response is incorrect<br>The value of the response is wrong<br>The server deviates from the correct flow of control |
| Arbitrary failure   | A server may produce arbitrary responses at arbitrary times  |

- Different types of failures.

## Fault types

- Node (hardware) faults
  - Program (software) faults
  - Communication faults
  - Timing faults
- 
- Implies types of redundancy

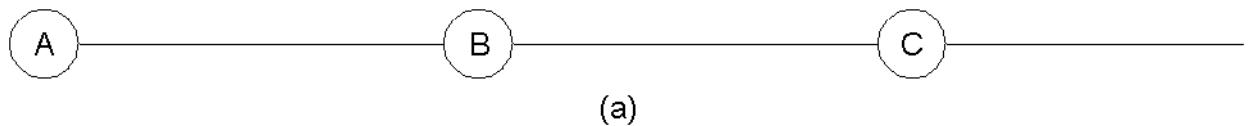
## Types of redundancy

- Hardware redundancy, extra PE, I/O
- Software redundancy, extra versions of modules
- Information redundancy, error detection bits
- Time redundancy, additional time to perform functions of a system

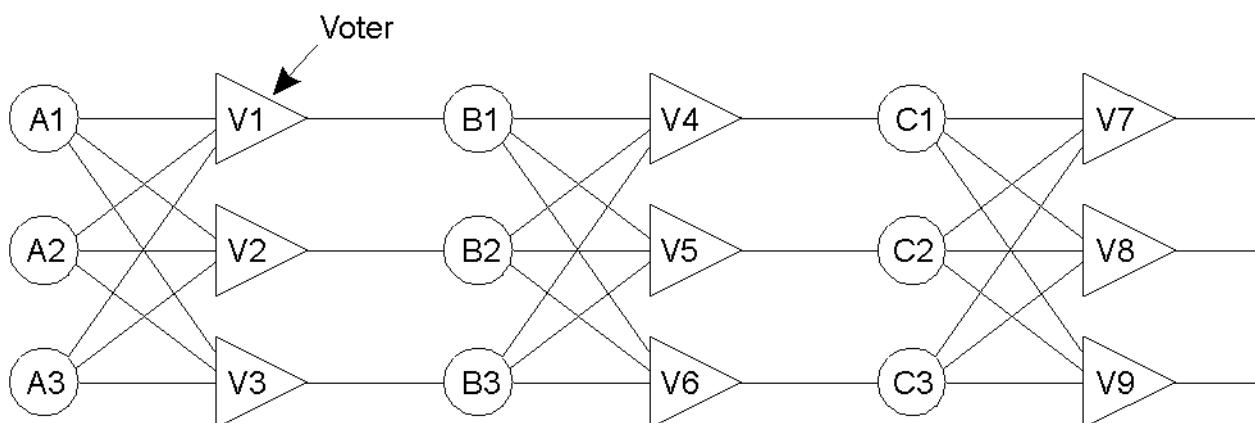
# Fault handling methods

- Active replication – all replication modules and their internal states are closely synchronized.
- Passive replication – only one module is active but other module's internal states are regularly updated by means of checkpoint from active module.
- Semi-active – hybrid of both active and passive replication. Low recovery overhead.

## Failure Masking by Redundancy



(a)



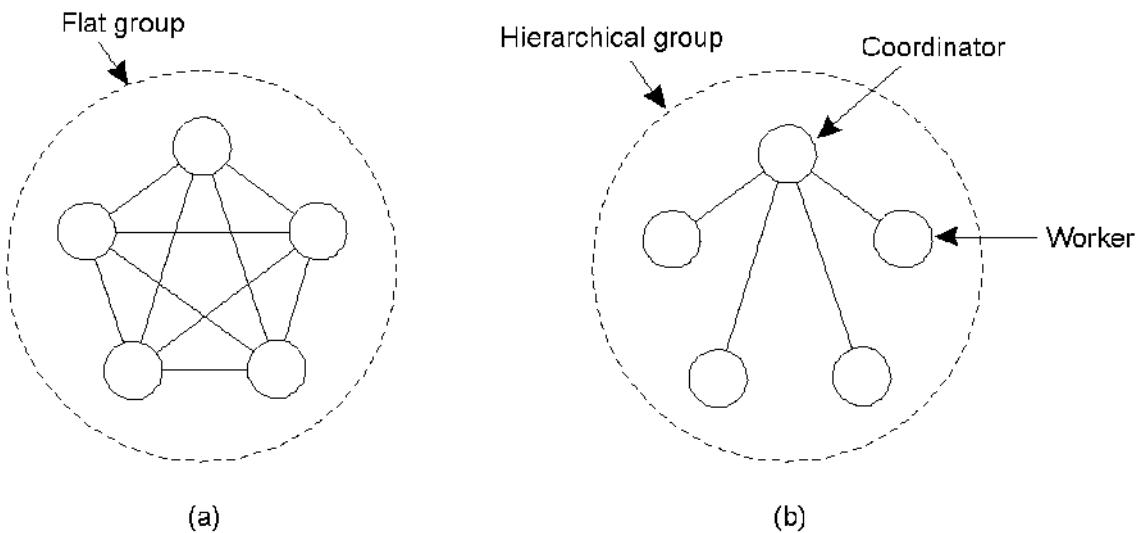
(b)

- Triple modular redundancy.

# Process Resilience

- Handling faulty processes: organize several processes into a group
  - All processes perform same computation
  - All messages are sent to all members of the group
  - Majority need to agree on results of a computation
  - Ideally want multiple, independent implementations of the application (to prevent identical bugs)
- Use *process groups* to organize such processes

## Flat Groups versus Hierarchical Groups



Advantages and disadvantages?

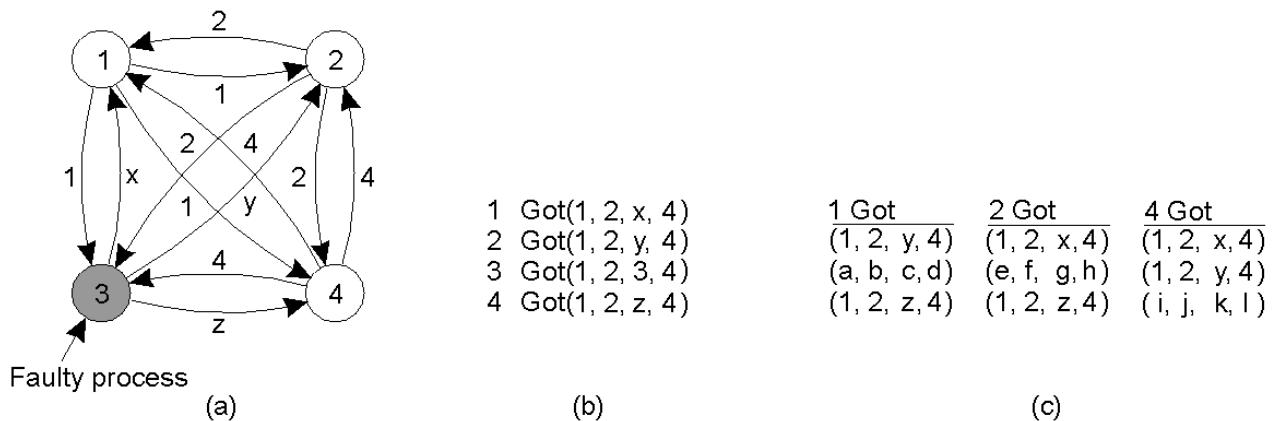
# Agreement in Faulty Systems

- How should processes agree on results of a computation?
- *K-fault tolerant*: system can survive k faults and yet function
- Assume processes fail silently
  - Need  $(k+1)$  redundancy to tolerate k faults
- *Byzantine failures*: processes run even if sick
  - Produce erroneous, random or malicious replies
    - Byzantine failures are most difficult to deal with
  - Need ? Redundancy to handle Byzantine faults

## Byzantine Faults

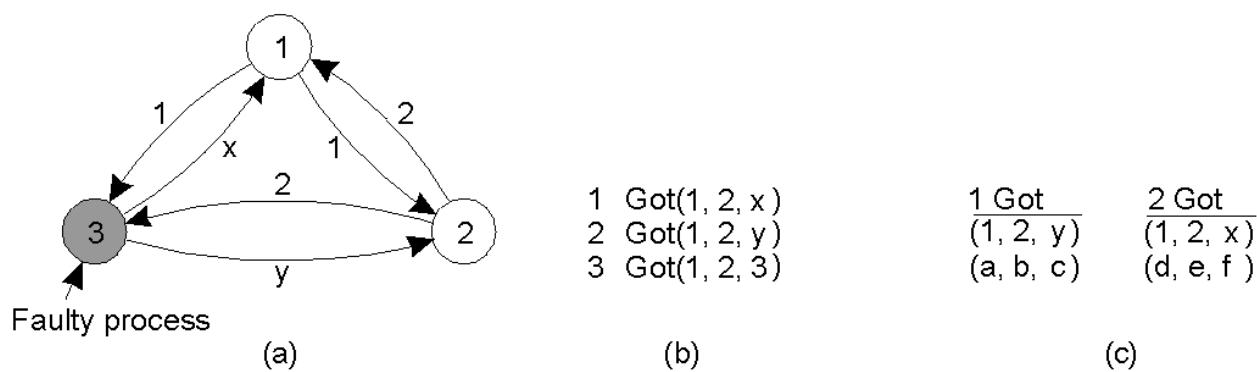
- Simplified scenario: two perfect processes with unreliable channel
  - Need to reach agreement on a 1 bit message
- Two army problem: Two armies waiting to attack
  - Each army coordinates with a messenger
  - Messenger can be captured by the hostile army
  - Can generals reach agreement?
  - Property: Two perfect process can never reach agreement in presence of unreliable channel
- Byzantine generals problem: Can N generals reach agreement with a perfect channel?
  - M generals out of N may be traitors

# Byzantine Generals Problem



- Recursive algorithm by Lamport
- The Byzantine generals problem for 3 loyal generals and 1 traitor.
  - a) The generals announce their troop strengths (in units of 1 kilosoldiers).
  - b) The vectors that each general assembles based on (a)
  - c) The vectors that each general receives in step 3.

## Byzantine Generals Problem Example



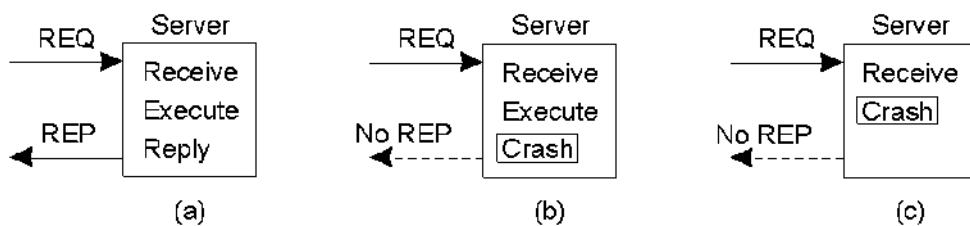
- The same as in previous slide, except now with 2 loyal generals and one traitor.
- Property: With  $m$  faulty processes, agreement is possible only if  $2m+1$  processes function correctly [Lamport 82]
  - Need more than two-thirds processes to function correctly

# More on Fault Tolerance

- Reliable communication
  - One-one communication
  - One-many communication
- Distributed commit
  - Two phase commit
  - Three phase commit
- Failure recovery
  - Checkpointing
  - Message logging

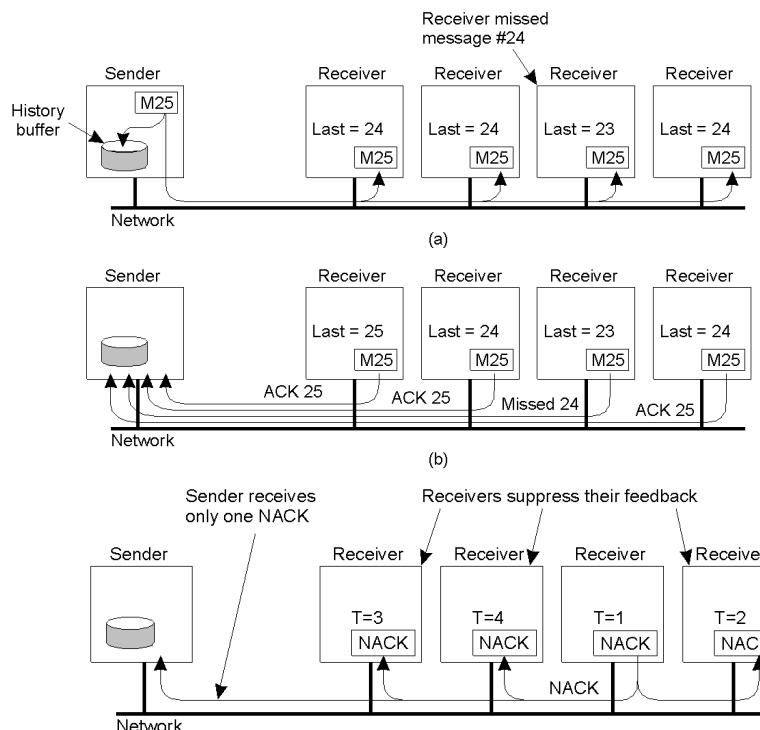
## Reliable One-One Communication

- Issues were discussed in Lecture 3
  - Use reliable transport protocols (TCP) or handle at the application layer
- RPC semantics in the presence of failures
- Possibilities
  - Client unable to locate server
  - Lost request messages
  - Server crashes after receiving request
  - Lost reply messages
  - Client crashes after sending request



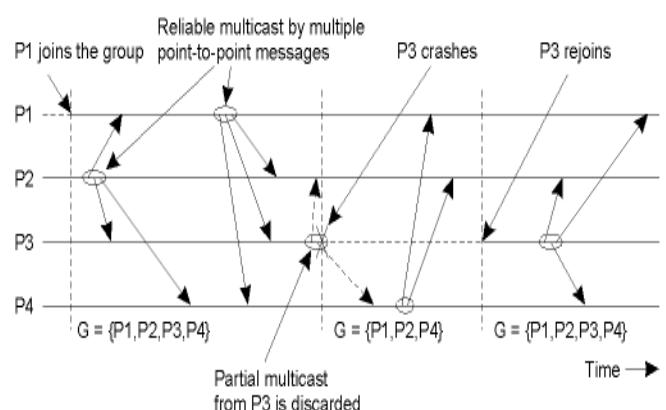
# Reliable One-Many Communication

- Reliable multicast
  - Lost messages => need to retransmit



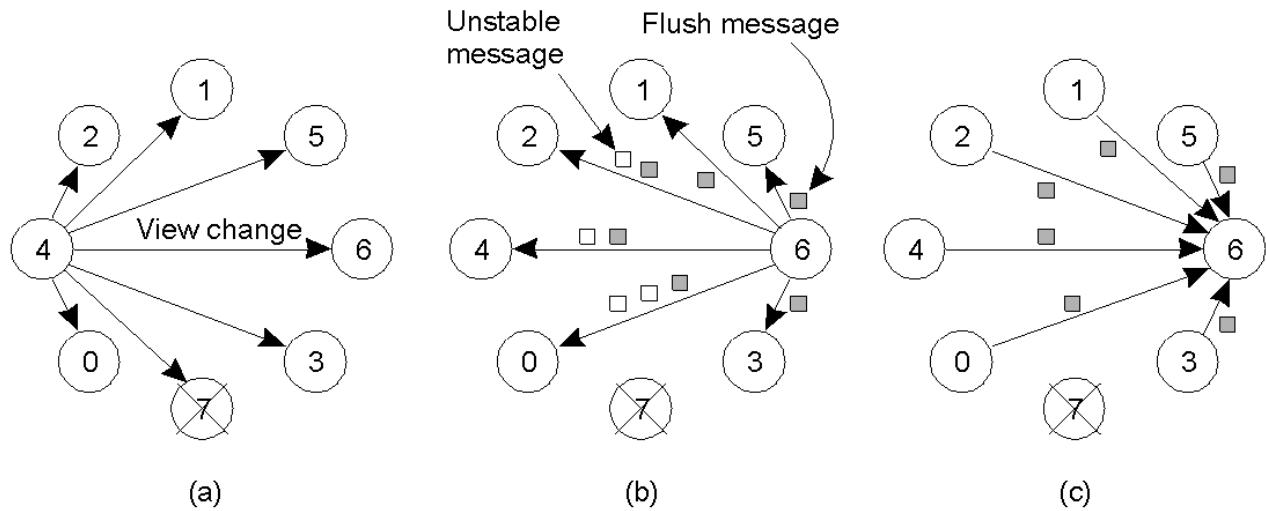
## Atomic Multicast

- Atomic multicast: a guarantee that all process received the message or none at all
  - Replicated database example
- Problem: how to handle process crashes?
- Solution: *group view*
  - Each message is uniquely associated with a group of processes
    - View of the process group when message was sent
    - All processes in the group should have the same view (and agree on it)



Virtually Synchronous Multicast

# Implementing Virtual Synchrony in Isis



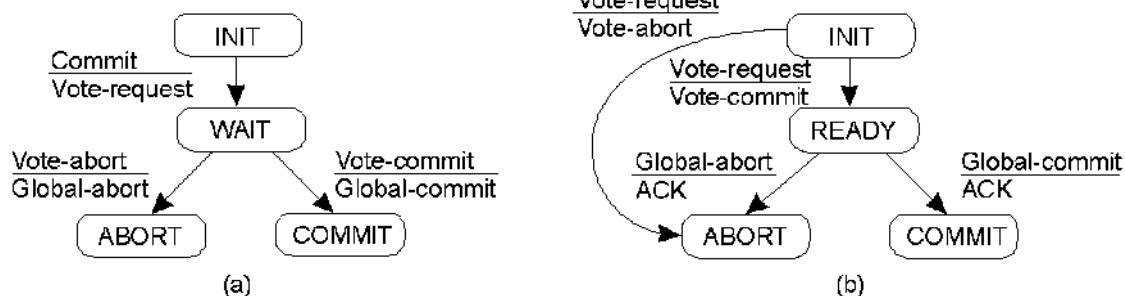
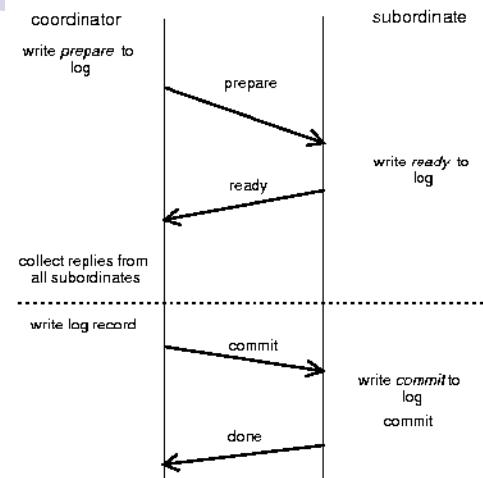
- a) Process 4 notices that process 7 has crashed, sends a view change
- b) Process 6 sends out all its unstable messages, followed by a flush message
- c) Process 6 installs the new view when it has received a flush message from everyone else

## Distributed Commit

- Atomic multicast example of a more general problem
  - All processes in a group perform an operation or not at all
  - Examples:
    - Reliable multicast: Operation = delivery of a message
    - Distributed transaction: Operation = commit transaction
- Problem of distributed commit
  - All or nothing operations in a group of processes
- Possible approaches
  - Two phase commit (2PC) [Gray 1978 ]
  - Three phase commit

# Two Phase Commit

- Coordinator process coordinates the operation
- Involves two phases
  - Voting phase: processes vote on whether to commit
  - Decision phase: actually commit or abort



## Implementing Two-Phase Commit

actions by coordinator:

```
while START _2PC to local log;
multicast VOTE_REQUEST to all participants;
while not all votes have been collected {
    wait for any incoming vote;
    if timeout {
        while GLOBAL_ABORT to local log;
        multicast GLOBAL_ABORT to all participants;
        exit;
    }
    record vote;
}
if all participants sent VOTE_COMMIT and coordinator votes COMMIT{
    write GLOBAL_COMMIT to local log;
    multicast GLOBAL_COMMIT to all participants;
} else {
    write GLOBAL_ABORT to local log;
    multicast GLOBAL_ABORT to all participants;
}
```

- Outline of the steps taken by the coordinator in a two phase commit protocol

# Implementing 2PC

## actions by participant:

```

write INIT to local log;
wait for VOTE_REQUEST from coordinator;
if timeout {
    write VOTE_ABORT to local log;
    exit;
}
if participant votes COMMIT {
    write VOTE_COMMIT to local log;
    send VOTE_COMMIT to coordinator;
    wait for DECISION from coordinator;
    if timeout {
        multicast DECISION_REQUEST to other participants;
        wait until DECISION is received; /* remain blocked */
        write DECISION to local log;
    }
    if DECISION == GLOBAL_COMMIT
        write GLOBAL_COMMIT to local log;
    else if DECISION == GLOBAL_ABORT
        write GLOBAL_ABORT to local log;
} else {
    write VOTE_ABORT to local log;
    send VOTE_ABORT to coordinator;
}

```

## actions for handling decision requests:

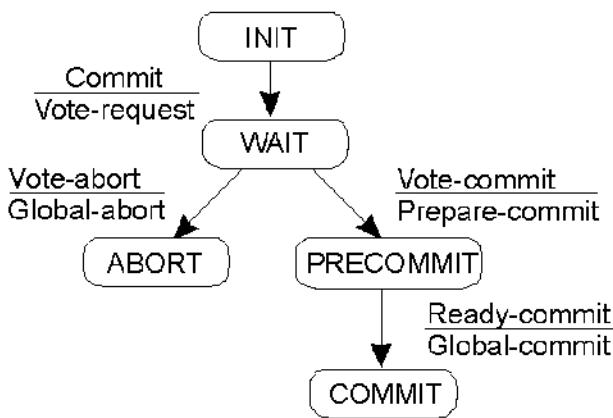
```

/*executed by separate thread */

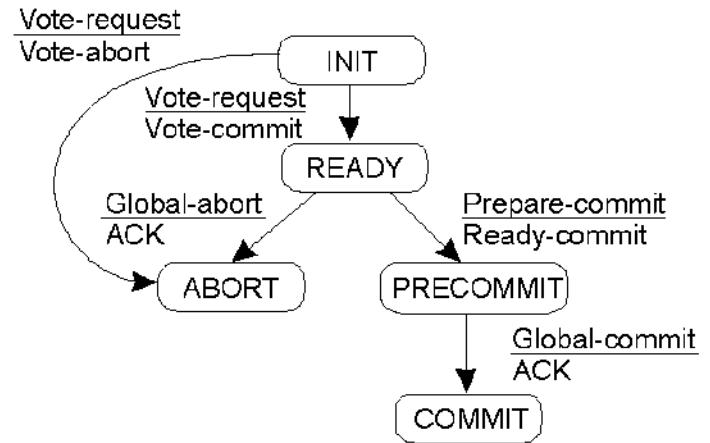
while true {
    wait until any incoming DECISION_REQUEST is
    received; /* remain blocked */
    read most recently recorded STATE from the local log;
    if STATE == GLOBAL_COMMIT
        send GLOBAL_COMMIT to requesting
        participant;
    else if STATE == INIT or STATE ==
        GLOBAL_ABORT
        send GLOBAL_ABORT to requesting participant;
    else
        skip; /* participant remains blocked */
}

```

# Three-Phase Commit



(a)



(b)

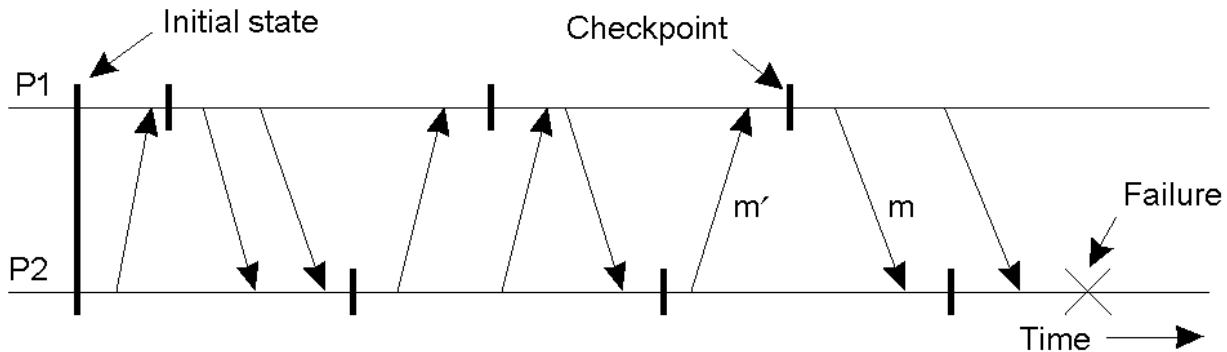
Two phase commit: problem if coordinator crashes  
(processes block)

Three phase commit: variant of 2PC that avoids blocking

# Recovery

- Techniques thus far allow failure handling
- Recovery: operations that must be performed after a failure to recover to a correct state
- Techniques:
  - Checkpointing:
    - Periodically checkpoint state
    - Upon a crash roll back to a previous checkpoint with a *consistent state*

## Independent Checkpointing



- Each process periodically checkpoints independently of other processes
- Upon a failure, work backwards to locate a consistent cut
- Problem: if most recent checkpoints form inconsistent cut, will need to keep rolling back until a consistent cut is found
- Cascading rollbacks can lead to a domino effect.

# Coordinated Checkpointing

- Take a distributed snapshot
- Upon a failure, roll back to the latest snapshot
  - All process restart from the latest snapshot

# Message Logging

- Checkpointing is expensive
  - All processes restart from previous consistent cut
  - Taking a snapshot is expensive
  - Infrequent snapshots => all computations after previous snapshot will need to be redone [wasteful]
- Combine checkpointing (expensive) with message logging (cheap)
  - Take infrequent checkpoints
  - Log all messages between checkpoints to local stable storage
  - To recover: simply replay messages from previous checkpoint
    - Avoids recomputations from previous checkpoint

# CONCURRENCY CONTROL

# CONTENT

---

- ▶ Methods for concurrency control
  - ▶ Locks
  - ▶ Optimistic concurrency control
  - ▶ Timestamp ordering
- ▶ Distributed Transactions
- ▶ Distributed Deadlock

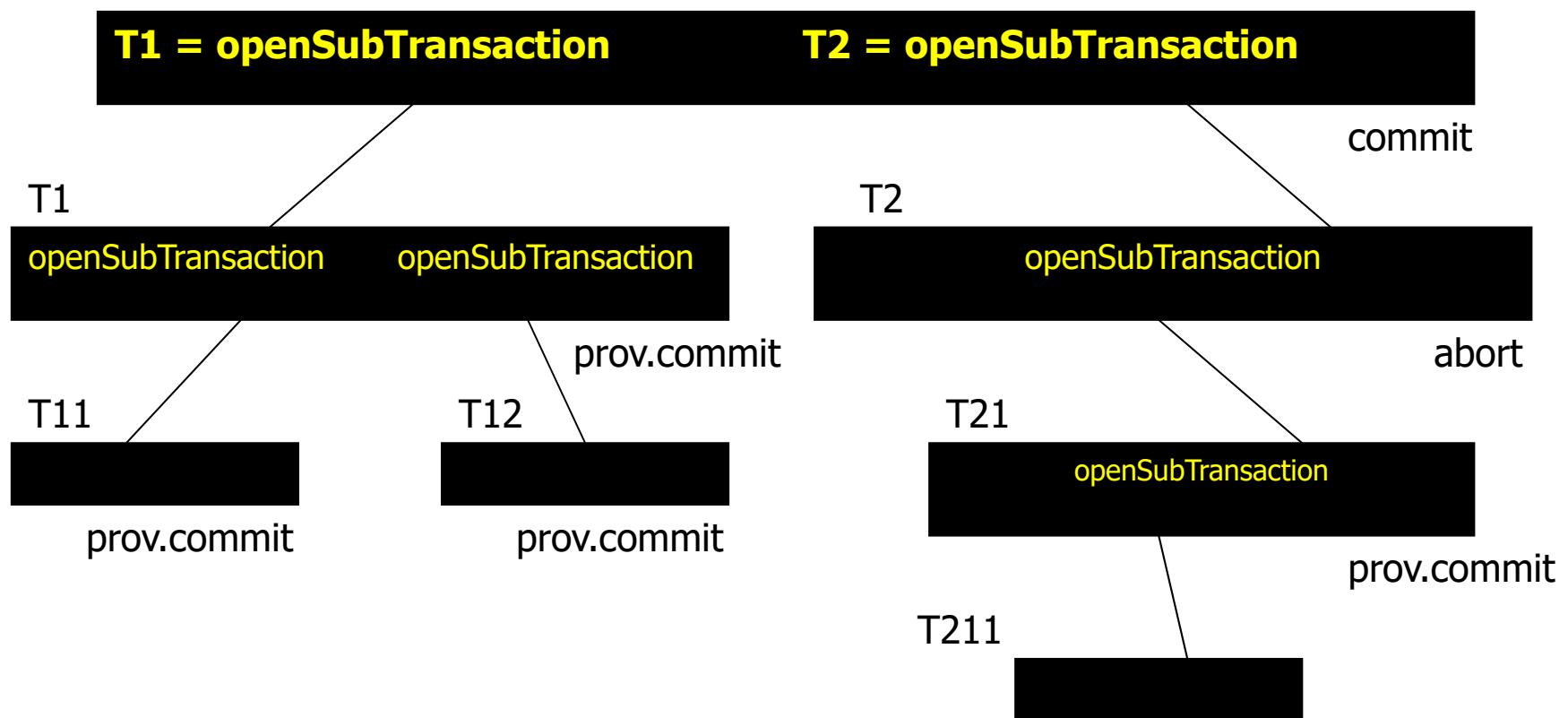
# Transactions (ACID)

---

- **Atomic**: All or nothing. No intermediate states are visible.
- **Consistent**: system invariants preserved.
- **Isolated**: Two transactions do not interfere with each other. They appear as serial executions.
- **Durable**: The commit causes a permanent change.

# Nested transaction

**T: top-level transaction**



# NESTED TRANSACTION

---

- ▶ The outermost transaction in a set of nested transactions is called the top-level transaction.
- ▶ Transactions other than the top-level transactions are called sub-transaction.
- ▶ Any sub-transaction appears atomic to its parent with respect to failures.
- ▶ Sub-transaction at the same level can run concurrently but their access to common objects is serialized.

## NESTED TRANSACTION

---

- ▶ Each sub-transaction can fail independently of its parent and of the other sub-transaction.
- ▶ When a sub-transaction aborts, the parent transaction can sometimes choose an alternative sub-transaction to complete its task.
- ▶ If all the tasks is done on same level, then it is called **flat transaction**.

# TRANSACTION & NESTED TRANSACTION

---

- ▶ Advantages of nested transaction:
  - ▶ Sub-transaction at one level (and descendant) may run concurrently with other sub-transaction: Additional concurrency in a transaction. If sub-transactions run in different servers, they can work parallel.
  - ▶ Subtransactions can commit or abort independently

# Schemes for Concurrency control

- ▶ **Locking**
  - ▶ Server attempts to gain an exclusive ‘lock’ that is about to be used by one of its operations in a transaction.
  - ▶ Can use different lock types (read/write for example)
  - ▶ Two-phase locking
- ▶ **Optimistic concurrency control**
- ▶ **Time-stamp based concurrency control**

# METHOD FOR CONCURRENCY CONTROL

---

## ▶ Lock:

- ▶ Server attempts to lock any object that is about to use by client's transaction.
- ▶ Requests to lock objects are suspended and wait until the objects are unlocked.
- ▶ **Serial equivalence:** transaction is not allowed any new locks after it has release a lock.
  - ▶ Two-phase lock: growing phase (new locks are acquired), shrinking phase (locks are released).
- ▶ **Strict execution:** locks are held until transaction commits/aborts (Strict two-phase locking).
- ▶ **Recoverability:** locks must be held until all the objects it updated have been written to permanent storage.

# METHOD FOR CONCURRENCY CONTROL

---

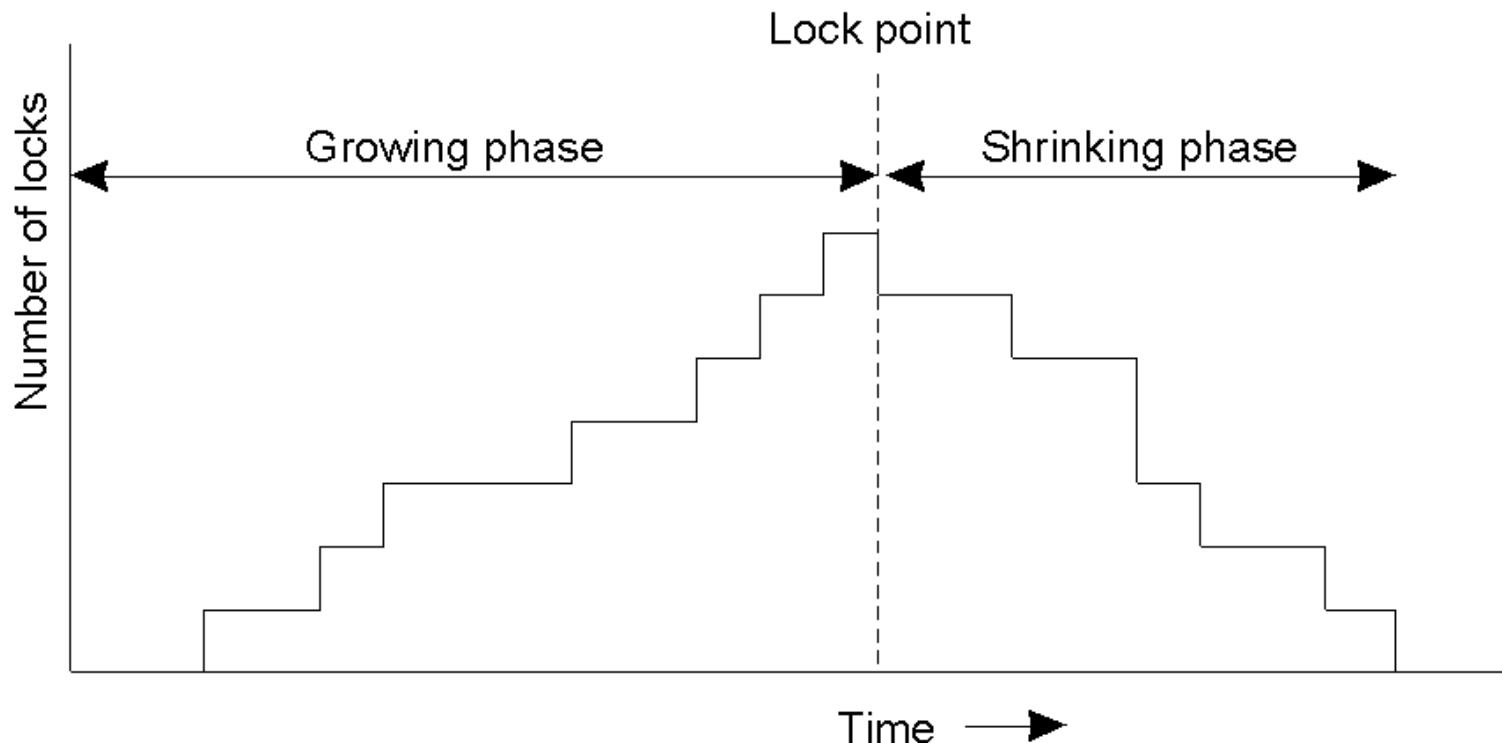
- ▶ **Lock:**
  - ▶ Simple exclusive lock reduces concurrency → locking scheme for multiple transaction reading an object, single transaction writing an object.
  - ▶ Two types of locks used: read locks (shared lock) & write locks.
  - ▶ Operation conflict rules:
    - ▶ Request for a write lock is delayed by the presence of a read lock belonging to another transaction.
    - ▶ Request for either a read/write lock is delayed by the presence of a write lock belonging to another transaction.

## Transaction T and U with exclusive locks.

| TRANSACTION T   | TRANSACTION U   |
|---|---|
| <pre>balance = b.getBalance( ); b.setBalance(balance*1.1); a.withdraw(balance/10);</pre>  | <pre>balance = b.getBalance( ); b.setBalance(balance*1.1); c.withdraw(balance/10);</pre>  |
| <pre>openTransaction balance = b.getBalance( ); lock B b.setBalance(balance*1.1); a.withdraw(balance/10); lock A</pre><br><pre>closeTransaction      unlock A,B</pre> | <pre>openTransaction balance = b.getBalance( ); wait for T'lock on B ... b.setBalance(balance*1.1); lock B c.withdraw(balance/10) ; lock C closeTransaction      unlock B,C</pre> |

- Assumption: balance of ABC are not yet locked when transaction T and U starts.
- When T starts using B, then it is locked for B. subsequently when U starts using B, it is still locked for T and so U waits. When T committed, B is unlocked and C resumes : effective serialization

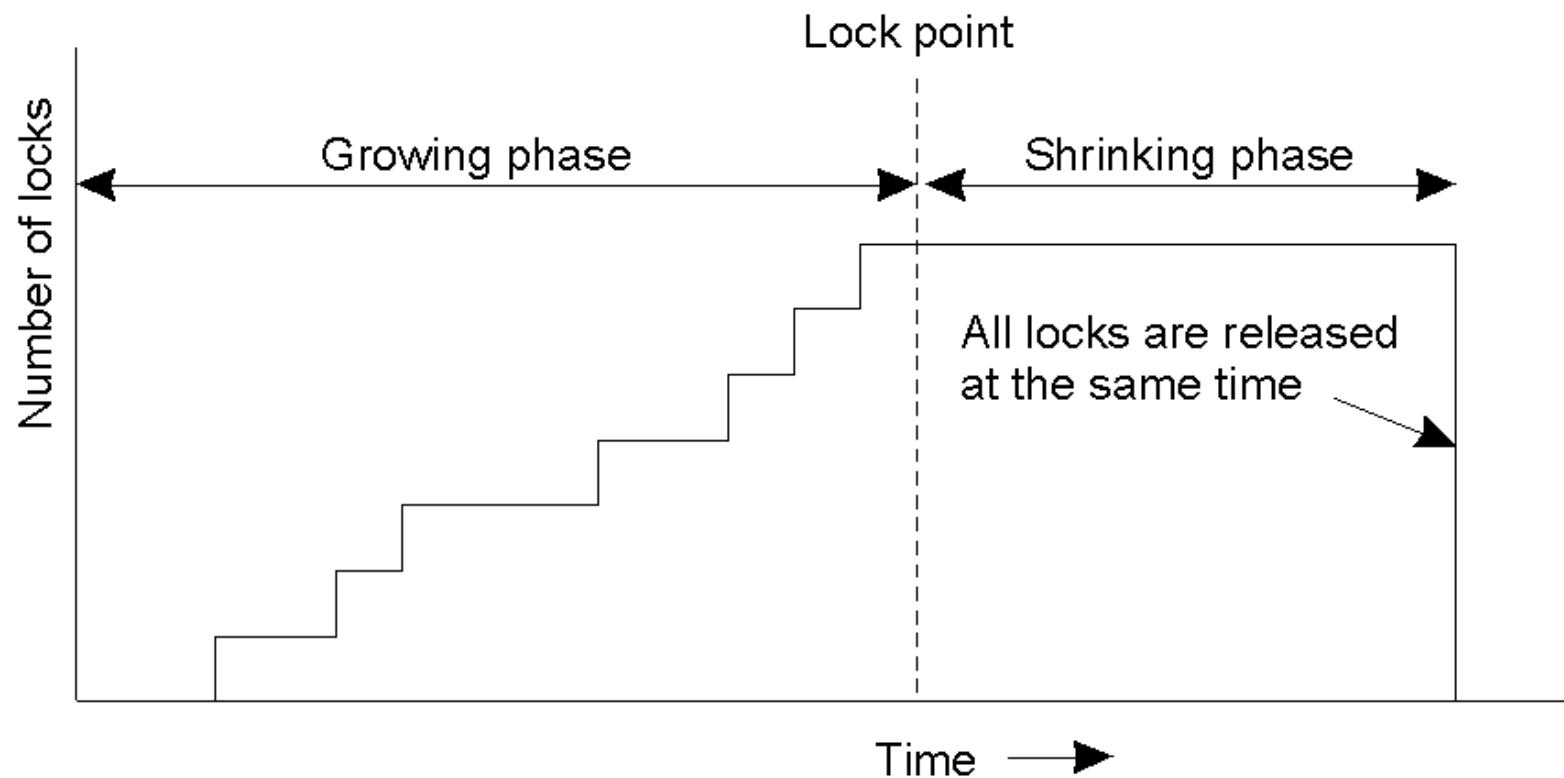
# Two-Phase Locking (1)



In two-phase locking, a transaction is not allowed to acquire any new locks after it has released a lock

# Strict Two-Phase Locking (2)

- Strict two-phase locking.



# Use of locks in strict two-phase locking

1. When an operation accesses an object within a transaction:
  - (a) If the object is not already locked, it is locked and the operation proceeds.
  - (b) If the object has a conflicting lock set by another transaction, the transaction must wait until it is unlocked.
  - (c) If the object has a non-conflicting lock set by another transaction, the lock is shared and the operation proceeds.
  - (d) If the object has already been locked in the same transaction, the lock will be promoted if necessary and the operation proceeds.  
(Where promotion is prevented by a conflicting lock, rule (b) is used.)
2. When a transaction is committed or aborted, the server unlocks all objects it locked for the transaction.

## Lock compatibility

| For one object   | Lock requested |       |
|------------------|----------------|-------|
|                  | Read           | Write |
| Lock already set | none           | OK    |
|                  | read           | OK    |
|                  | write          | Wait  |

# METHOD FOR CONCURRENT CONTROL

---

- ▶ **Locking rule for nested transactions:**
  - ▶ Locks that are acquired by a successful subtransaction is inherited by its parent & ancestors when it completes. Locks held until top-level transaction commits/aborts.
  - ▶ Parent transactions are not allowed to run concurrently with their child transactions.
  - ▶ Subtransactions at the same level are allowed to run concurrently.

# Method for concurrent control

---

- ▶ Deadlock:
  - ▶ Definition: A state in which each member of a group of transactions is waiting for some other member to release a lock.
- ▶ Prevention:
  - ▶ Lock all the objects used by a transaction when it starts → not a good way.
  - ▶ Request locks on objects in a predefined order → premature locking & reduction in concurrency.

# METHOD FOR CONCURRENT CONTROL

---

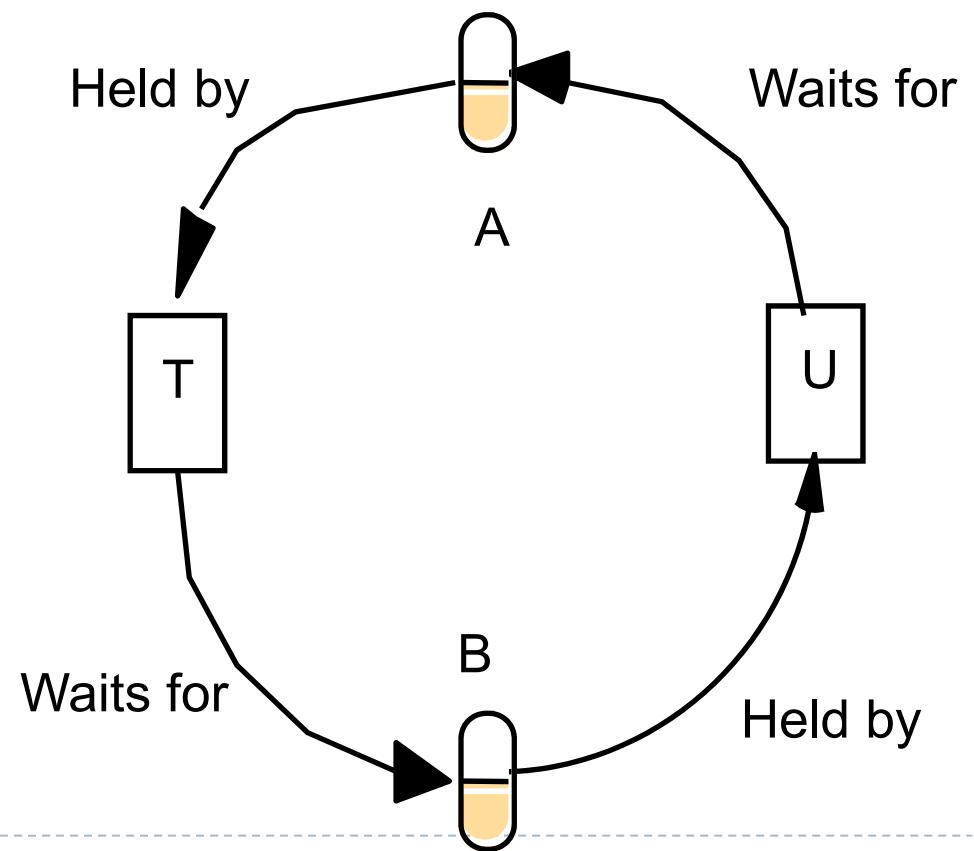
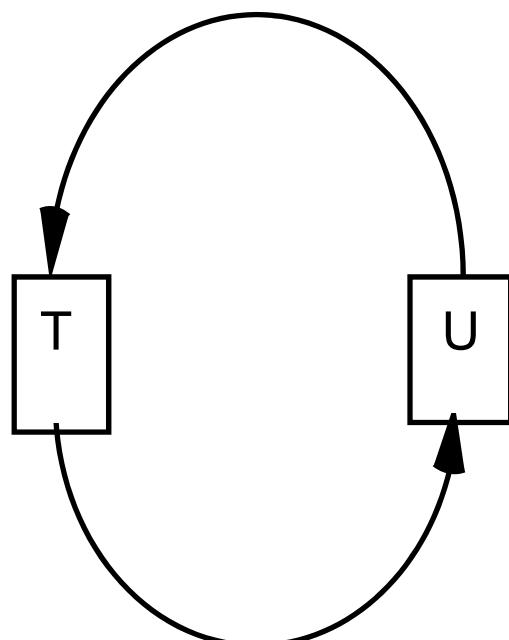
- ▶ **Deadlock:**
  - ▶ Detection: Finding cycle in a wait-for graph → select a transaction for aborting to break the cycle.
    - ▶ Choice of transaction to be aborted is not simple.
  - ▶ Timeouts: each lock is given a limited period in which it is invulnerable.
    - ▶ Transaction is sometimes aborted but actually there is no deadlock.
- ▶ If we use locking to implement concurrency control in transactions, we can get deadlocks (even within a single server)
- ▶ So we need to discuss:
  - ▶ Deadlock detection within a single system
  - ▶ Distributed deadlock

## Deadlock detection

---

- ▶ A deadlock occurs when there is a cycle in the *wait-for* graph of transactions for locks
- ▶ There may be more than one
- ▶ Resolve the deadlock by aborting one of the transactions ....
- ▶ E.g. the youngest, or the one involved in more than one cycle, or can even use “priority” ....

## A cycle in a wait-for graph



# METHOD FOR CONCURRENCY CONTROL

---

- ▶ Drawbacks of locking:
  - ▶ Lock maintenance represents an overhead that is not present in systems that do not support concurrent access to shared data.
  - ▶ Deadlock. Deadlock prevention reduces concurrency. Deadlock detection or timeout not wholly satisfactory for use in interactive programs.
  - ▶ To avoid cascading aborts, locks can't be released until the end of the transaction. This may reduce significantly the potential for concurrency.

# METHOD FOR CONCURRENT CONTROL

---

- ▶ **Optimistic concurrency control:**
  - ▶ Is an alternative optimistic approach to the serialization of transactions that avoids the drawbacks of locking.
  - ▶ Idea: in most applications, the likelihood of two clients transactions accessing the same object is low.
  - ▶ Transactions are allowed to proceed as though there were no possibility of conflict with other transactions until the client completes its task and issues a close-Transaction request.

# METHOD FOR CONCURRENT CONTROL

---

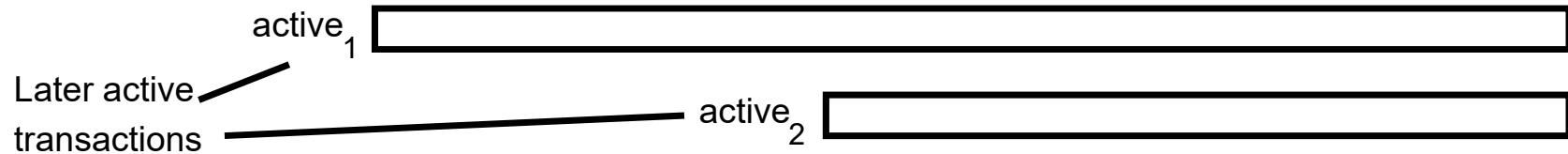
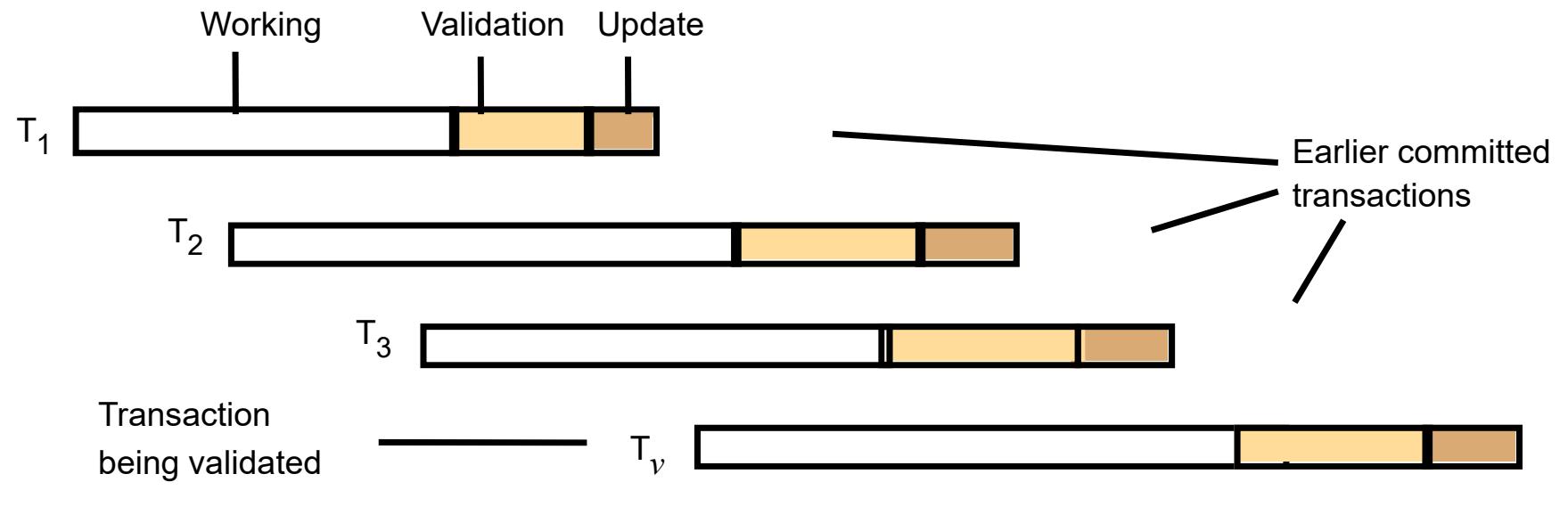
- ▶ **Optimistic concurrency control:**
  - ▶ Each transaction has the following 3 phases:
    - ▶ **Working phase:** each transaction has a tentative version of each of the objects that it updates.
    - ▶ **Validation phase:** Once transaction is done, the transaction is validated to establish whether or not its operations on objects conflict with operations of other transactions on the same object. If not conflict, can commit; else some form of conflict resolution is needed and the transaction may abort.
    - ▶ **Update phase:** changes in tentative versions are made permanent if transaction is validated

# METHOD FOR CONCURRENT CONTROL

## ▶ Optimistic concurrency control:

- ▶ **Validation of transactions:** use the read-write conflict rules to ensure that the scheduling of a transaction is serially equivalent with respect to all other overlapping transactions.
- ▶ **Backward validation:** check the transaction undergoing validation with other preceding overlapping transactions (enter the validation phase before).
  - ▶ Read set of the transaction being validated is compared with the write sets of other transactions that have already committed.
- ▶ **Forward validate:** check the transaction undergoing validation with other later transactions
  - ▶ Write set of the transaction being validated is compared with the read sets of other overlapping active transactions (still in working phase).

# Validation of transactions



# Schemes for Concurrency control

---

- ▶ **Time-stamp based concurrency control**
  - ▶ Each transaction is assigned a unique timestamp at the moment it starts
    - ▶ In distributed transactions, Lamport's timestamps can be used
  - ▶ Every data item has a timestamp
    - ▶ Read timestamp = timestamp of transaction that last read the item
    - ▶ Write timestamp = timestamp of transaction that most recently changed an item

# METHOD FOR CONCURRENT CONTROL

---

- ▶ **Timestamp ordering:**
  - ▶ **Basic timestamp ordering rule:**
    - ▶ A transaction's request to write an object is valid only if that object was last read and written by earlier transactions. A transaction's request to read an object is valid only if that object was last written by an earlier transactions
- ▶ **Timestamp ordering write rule:**

if  $(T_c \geq \text{maximum read timestamp on } D \ \&\& T_c > \text{write timestamp on committed version of } D)$   
    perform write operation on tentative version of  $D$  with  
    write timestamp  $T_c$

else /\**write is too late*\*/  
    abort transaction  $T_c$

# METHOD FOR CONCURRENT CONTROL

---

## ► Timestamp ordering read rule:

If ( $T_c >$  write timestamp on committed version of D)

{ let  $D_{selected}$  be the version of D with the maximum write timestamp  $\leq T_c$

if ( $D_{selected}$  is committed)

perform read operation on the version  $D_{selected}$

else

wait until the transaction that made version  $D_{selected}$  commits or aborts then reapply the read rule

}

Else

abort transaction  $T_c$

# Concurrency Control for Distributed Transactions

---

- ▶ Locking
  - ▶ Distributed deadlocks possible
- ▶ Timestamp ordering
  - ▶ Lamport time stamps
    - ▶ for efficiency it is required that timestamps issued by coordinators be roughly synchronized

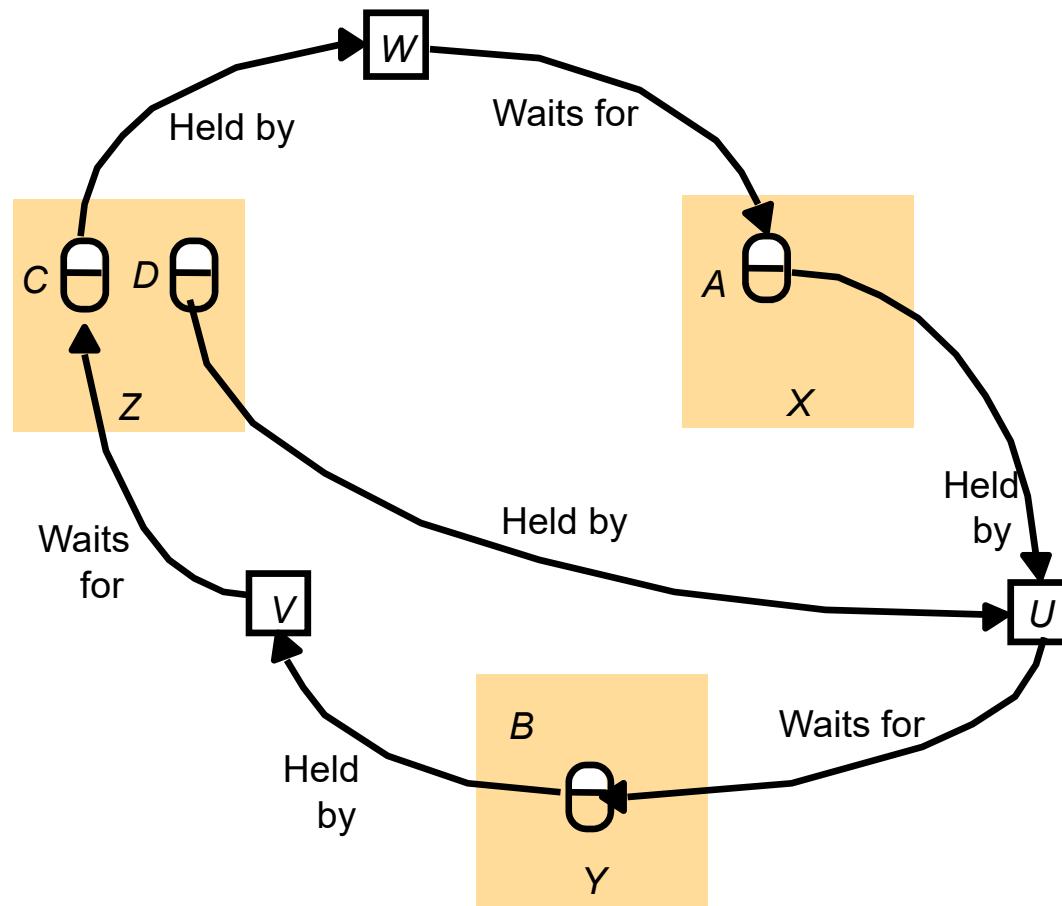
# Distributed Deadlock

---

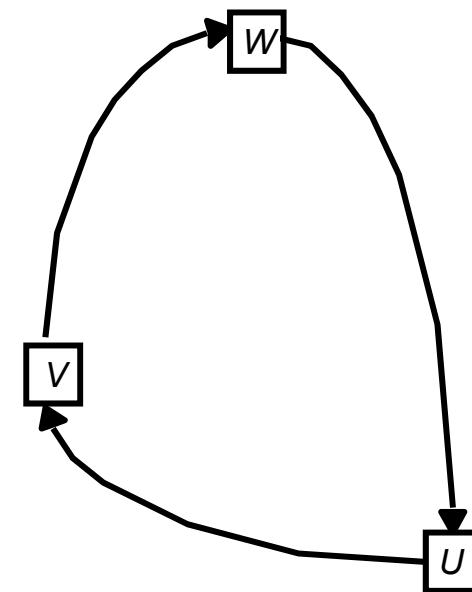
- ▶ Within a single server, allocating and releasing locks can be done so as to maintain a wait-for graph which can be periodically checked.
- ▶ With distributed transactions locks are held in different servers – and the loop in the entire wait-for graph will not be apparent to any one server
- ▶ One solution is to have a coordinator to which each server forwards its wait-for graph
- ▶ But centralised coordination is not ideal in a distributed system

# Distributed deadlock

(a)



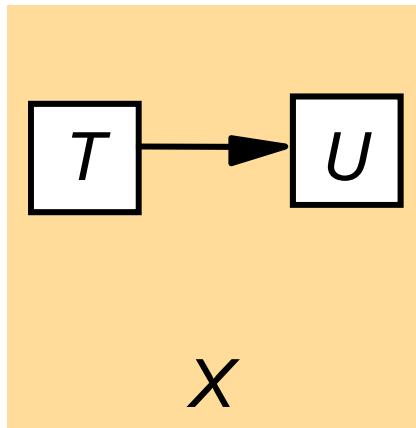
(b)



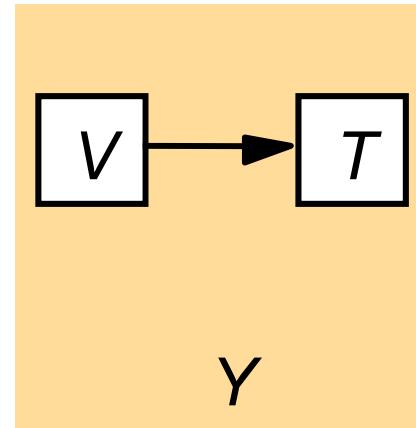
Transactions

# Local and global wait-for graphs

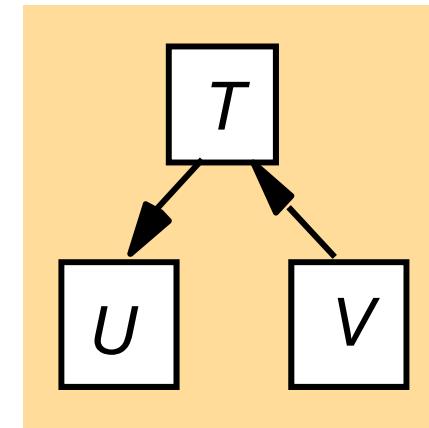
local wait-for graph



local wait-for graph



global deadlock detector



# Atomic Commit Protocols

---

- ▶ The atomicity of a transaction requires that when a distributed transaction comes to an end, either all of its operations are carried out or none of them
- ▶ Two phase commit (2PC)
- ▶ Three Phase Commit (3PC)
- ▶ Recovery....



Covered In CH-9



# The two-phase commit protocol - 1

---

*Phase 1 (voting phase):*

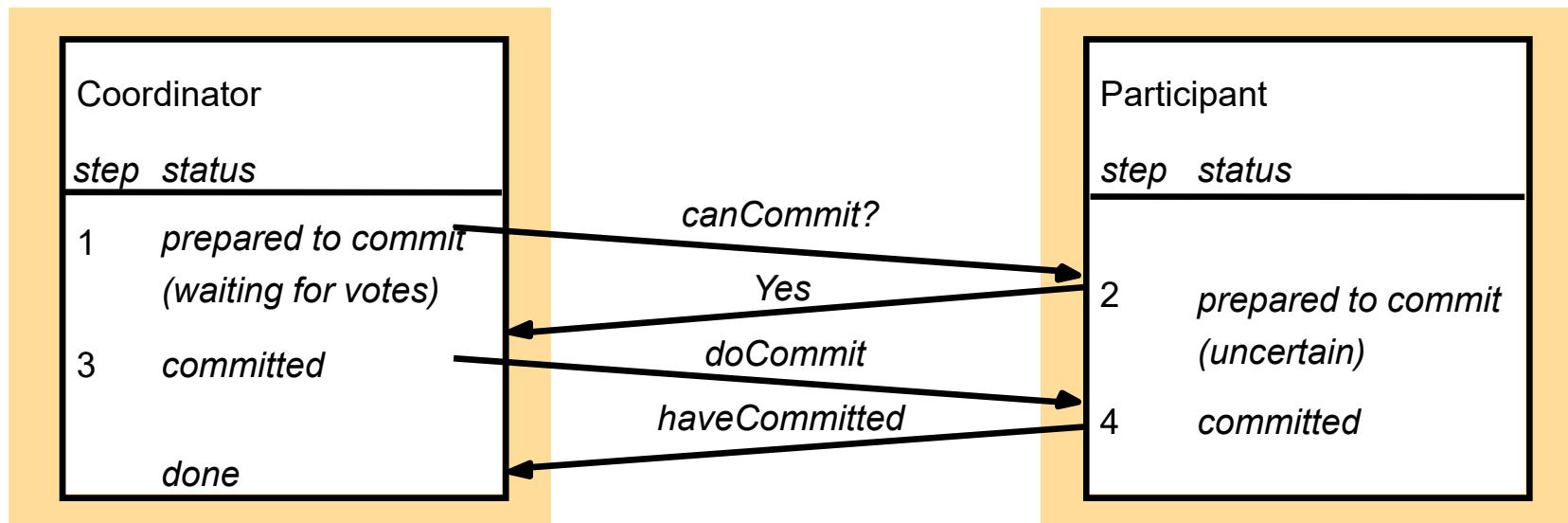
1. The coordinator sends a *canCommit?* (*VOTE\_REQUEST*) request to each of the participants in the transaction.
2. When a participant receives a *canCommit?* request it replies with its vote *Yes* (*VOTE\_COMMIT*) or *No* (*VOTE\_ABORT*) to the coordinator. Before voting *Yes*, it prepares to commit by saving objects in permanent storage. If the vote is *No* the participant aborts immediately.

# The two-phase commit protocol - 2

*Phase 2 (completion according to outcome of vote):*

3. The coordinator collects the votes (including its own).
  - (a) If there are no failures and all the votes are *Yes* the coordinator decides to commit the transaction and sends a *doCommit (GLOBAL\_COMMIT)* request to each of the participants.
  - (b) Otherwise the coordinator decides to abort the transaction and sends *doAbort (GLOBAL\_ABORT)* requests to all participants that voted *Yes*.
4. Participants that voted *Yes* are waiting for a *doCommit* or *doAbort* request from the coordinator. When a participant receives one of these messages it acts accordingly and in the case of commit, makes a *haveCommitted* call as confirmation to the coordinator.

# Communication in two-phase commit protocol



# Operations for two-phase commit protocol

---

*canCommit?(trans)-> Yes / No*

Call from coordinator to participant to ask whether it can commit a transaction.  
Participant replies with its vote.

*doCommit(trans)*

Call from coordinator to participant to tell participant to commit its part of a transaction.

*doAbort(trans)*

Call from coordinator to participant to tell participant to abort its part of a transaction.

*haveCommitted(trans, participant)*

Call from participant to coordinator to confirm that it has committed the transaction.

*getDecision(trans) -> Yes / No*

Call from participant to coordinator to ask for the decision on a transaction after it has voted *Yes* but has still had no reply after some delay. Used to recover from server crash or delayed messages.

# Two-Phase Commit protocol - 3

---

## **actions by coordinator:**

```
while START _2PC to local log;  
multicast VOTE_REQUEST to all participants;  
while not all votes have been collected {  
    wait for any incoming vote;  
    if timeout {  
        write GLOBAL_ABORT to local log;  
        multicast GLOBAL_ABORT to all participants;  
        exit;  
    }  
    record vote;  
}  
if all participants sent VOTE_COMMIT and coordinator votes COMMIT{  
    write GLOBAL_COMMIT to local log;  
    multicast GLOBAL_COMMIT to all participants;  
} else {  
    write GLOBAL_ABORT to local log;  
    multicast GLOBAL_ABORT to all participants;  
}
```

**Outline of the steps taken by the coordinator in a two phase commit protocol**

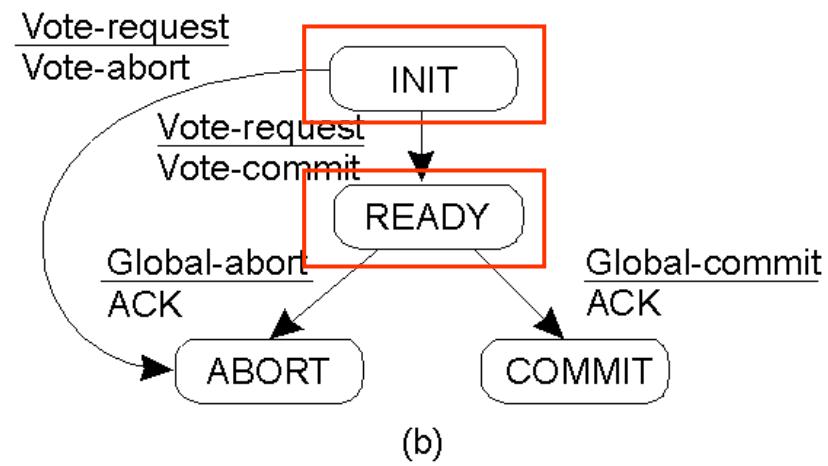
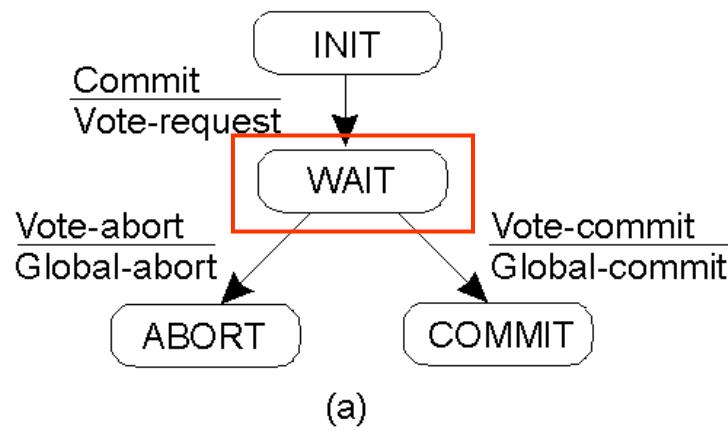
# Two-Phase Commit protocol - 4

Steps taken by participant process in 2PC.

## actions by participant:

```
write INIT to local log;  
wait for VOTE_REQUEST from coordinator;  
if timeout {  
    write VOTE_ABORT to local log;  
    exit;  
}  
if participant votes COMMIT {  
    write VOTE_COMMIT to local log;  
    send VOTE_COMMIT to coordinator;  
    wait for DECISION from coordinator;  
    if timeout {  
        multicast DECISION_REQUEST to other participants;  
        wait until DECISION is received; /* remain blocked */  
        write DECISION to local log;  
    }  
    if DECISION == GLOBAL_COMMIT  
        write GLOBAL_COMMIT to local log;  
    else if DECISION == GLOBAL_ABORT  
        write GLOBAL_ABORT to local log;  
} else {  
    write VOTE_ABORT to local log;  
    send VOTE_ABORT to coordinator;  
}
```

# Two-Phase Commit protocol - 5



- a) The finite state machine for the coordinator in 2PC.
- b) The finite state machine for a participant.

If a failure occurs during a ‘blocking’ state (red boxes), there needs to be a recovery mechanism.

# Two Phase Commit Protocol - 6

## Recovery

- ▶ ‘Wait’ in Coordinator – use a time-out mechanism to detect participant crashes. Send GLOBAL\_ABORT
- ▶ ‘Init’ in Participant – Can also use a time-out and send VOTE\_ABORT
- ▶ ‘Ready’ in Participant P – abort is not an option (since already voted to COMMIT and so coordinator might eventually send GLOBAL\_COMMIT). Can contact another participant Q and choose an action based on its state.

| State of Q | Action by P  |
|------------|--|
| COMMIT     | Transition to COMMIT   |
| ABORT      | Transition to ABORT  |
| INIT       | Both P and Q transition to ABORT<br>(Q sends VOTE_ABORT)   |
| READY      | Contact more participants. If all participants are ‘READY’, must wait for coordinator to recover |



# Two-Phase Commit protocol - 7

---

**actions for handling decision requests:** /\* executed by separate thread \*/

```
while true {
    wait until any incoming DECISION_REQUEST is received; /* remain blocked */
    read most recently recorded STATE from the local log;
    if STATE == GLOBAL_COMMIT
        send GLOBAL_COMMIT to requesting participant;
    else if STATE == INIT or STATE == GLOBAL_ABORT
        send GLOBAL_ABORT to requesting participant;
    else
        skip; /* participant remains blocked */
```

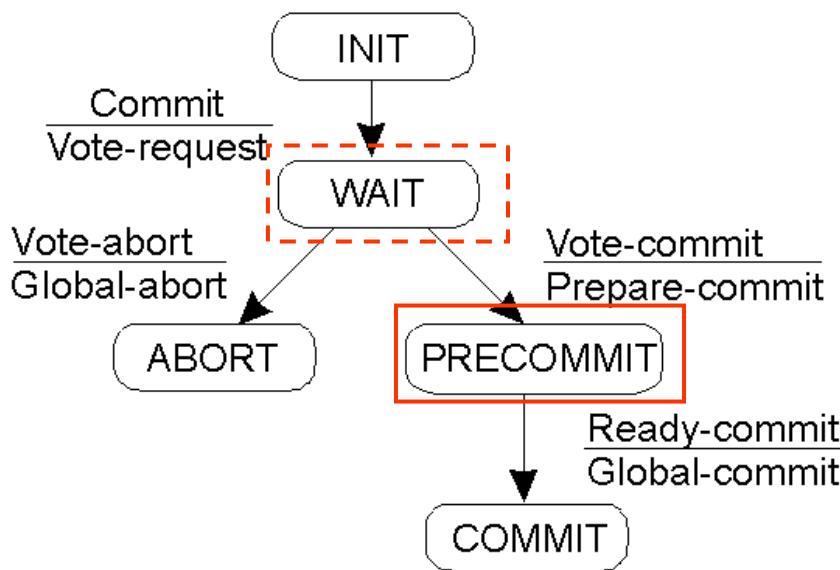
Steps taken for handling incoming decision requests.

# Three Phase Commit protocol - 1

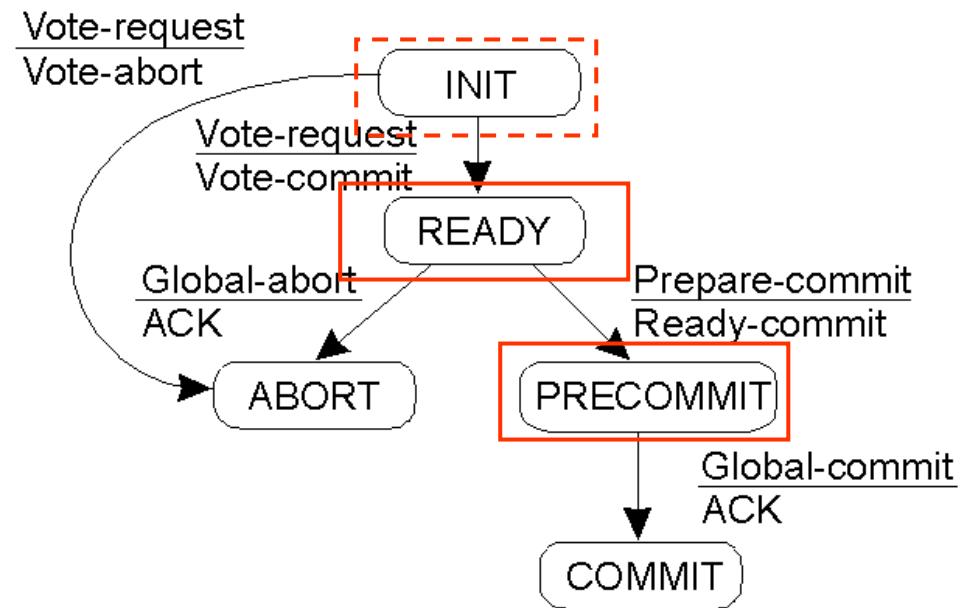
---

- ▶ **Problem with 2PC**
  - ▶ If coordinator crashes, participants cannot reach a decision, stay blocked until coordinator recovers
- ▶ **Three Phase Commit3PC**
  - ▶ There is no single state from which it is possible to make a transition directly to either COMMIT or ABORT states
  - ▶ There is no state in which it is not possible to make a final decision, and from which a transition to COMMIT can be made

# Three-Phase Commit protocol - 2



(a)



(b)

- a) Finite state machine for the coordinator in 3PC
- b) Finite state machine for a participant

# Three Phase Commit Protocol - 3

- ▶ ‘Wait’ in Coordinator – same
- ▶ ‘Init’ in Participant – same
- ▶ ‘PreCommit’ in Coordinator – Some participant has crashed but we know it wanted to commit. GLOBAL\_COMMIT the application knowing that once the participant recovers, it will commit.
- ▶ ‘Ready’ or ‘PreCommit’ in Participant P – (i.e. P has voted to COMMIT)

| State of Q | Action by P  |
|------------|--|
| PRECOMMIT  | Transition to PRECOMMIT. If all participants in PRECOMMIT, can COMMIT the transaction  |
| ABORT      | Transition to ABORT  |
| INIT       | Both P (in READY) and Q transition to ABORT (Q sends VOTE_ABORT)   |
| READY      | Contact more participants. If can contact a majority and they are in ‘Ready’, then ABORT the transaction.<br>If the participants contacted in ‘PreCommit’ it is safe to COMMIT the transaction |

*Note: if any participant is in state PRECOMMIT, it is impossible for any other participant to be in any state other than READY or PRECOMMIT.*

# Cloud Computing

# What is Cloud Computing?

- **Cloud Computing** is a general term used to describe a new class of network based computing that takes place over the Internet,
  - basically a step on from Utility Computing
  - a collection/group of integrated and networked hardware, software and Internet infrastructure (called a platform).
  - Using the Internet for communication and transport provides hardware, software and networking services to clients
- These platforms hide the complexity and details of the underlying infrastructure from users and applications by providing very simple graphical interface or API (Applications Programming Interface).

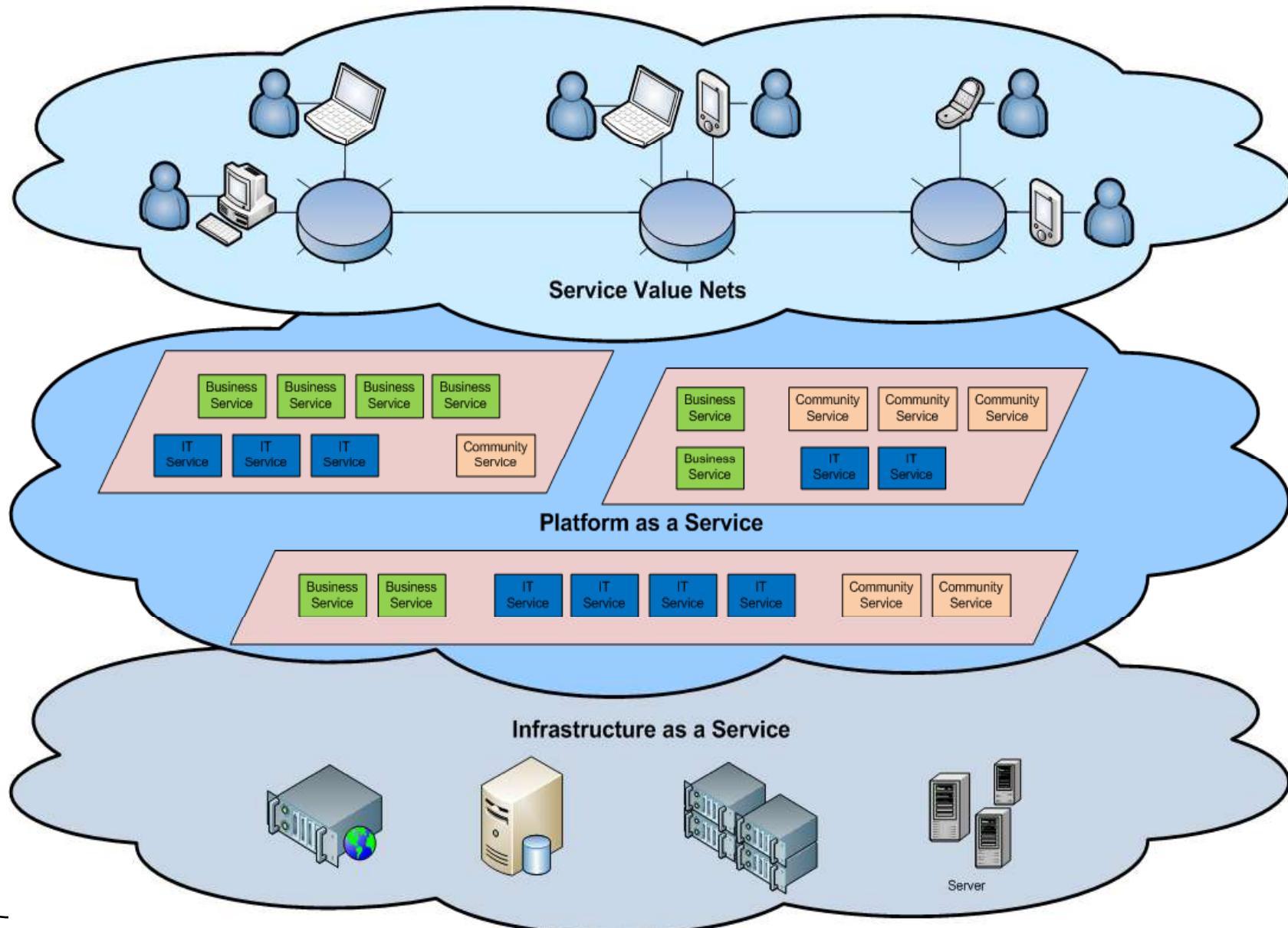
# What is Cloud Computing?

- In addition, the platform provides on demand services, that are always on, anywhere, anytime and any place.
- Pay for use and as needed, elastic
  - scale up and down in capacity and functionalities
- The hardware and software services are available to
  - general public, enterprises, corporations and businesses markets

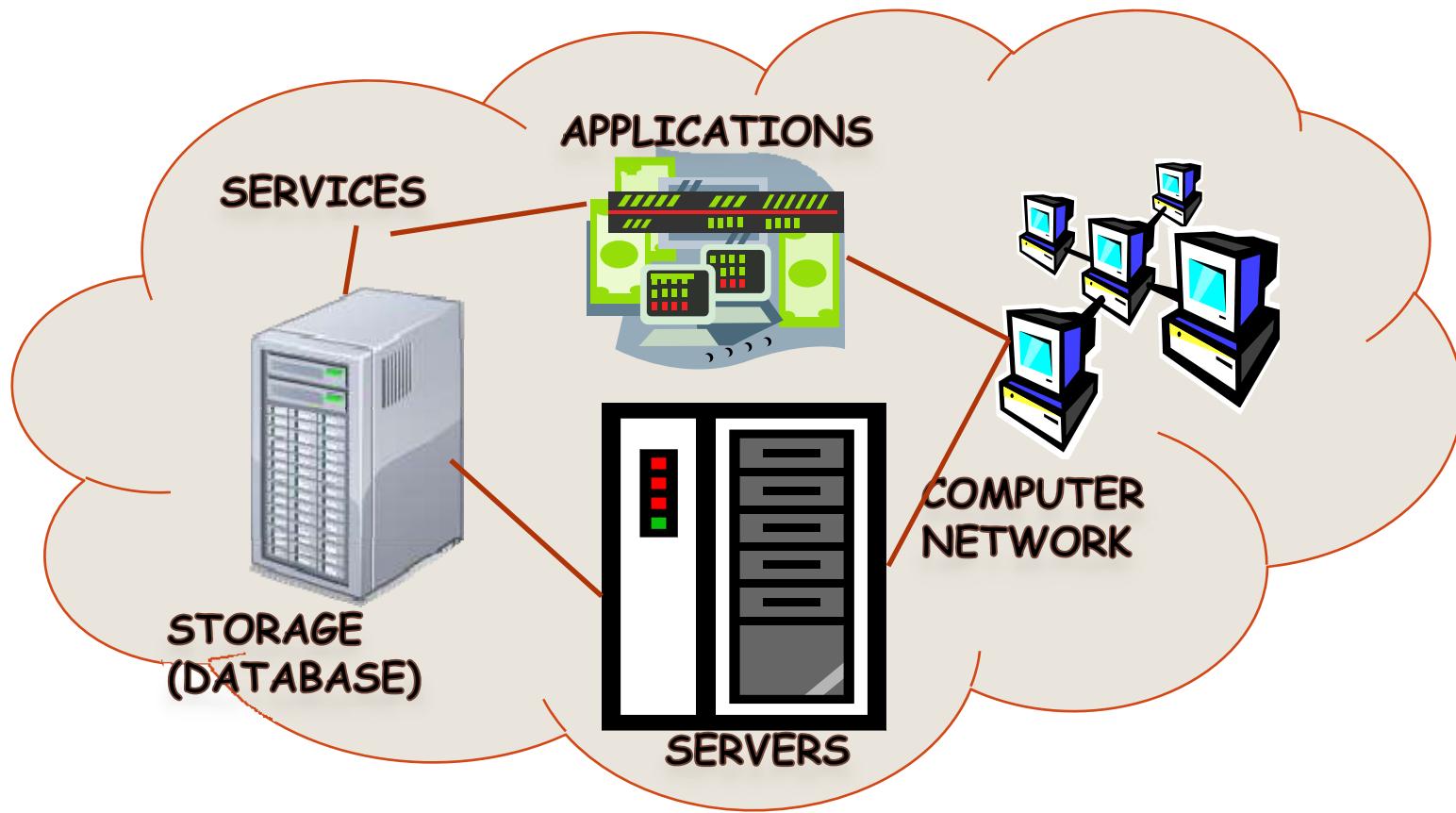
# Cloud Summary

- Cloud computing is an umbrella term used to refer to Internet based development and services
- A number of characteristics define cloud data, applications services and infrastructure:
  - **Remotely hosted:** Services or data are hosted on remote infrastructure.
  - **Ubiquitous:** Services or data are available from anywhere.
  - **Commodified:** The result is a utility computing model similar to traditional that of traditional utilities, like gas and electricity
    - you pay for what you would want!

# Cloud Architecture



# What is Cloud Computing



- Shared pool of configurable computing resources
- On-demand network access
- Provisioned by the Service Provider

# Cloud Computing Characteristics

## Common Characteristics:

**Massive Scale**

**Resilient Computing**

**Homogeneity**

**Geographic Distribution**

**Virtualization**

**Service Orientation**

**Low Cost Software**

**Advanced Security**

## Essential Characteristics:

**On Demand Self-Service**

**Broad Network Access**

**Rapid Elasticity**

**Resource Pooling**

**Measured Service**

# Cloud Service Models

Software as a Service (SaaS)

Platform as a Service (PaaS)

Infrastructure as a Service (IaaS)

SalesForce CRM

LotusLive

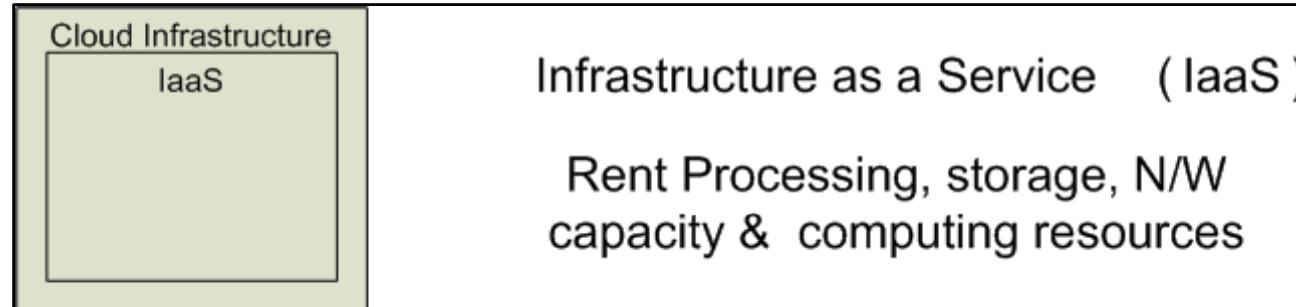
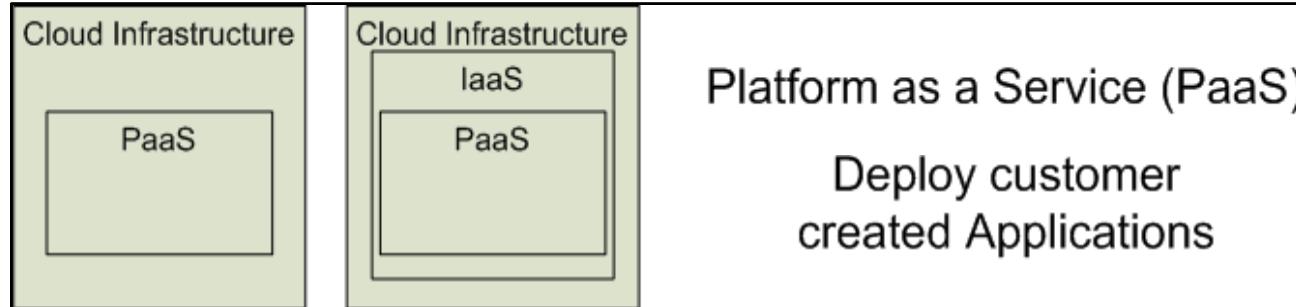
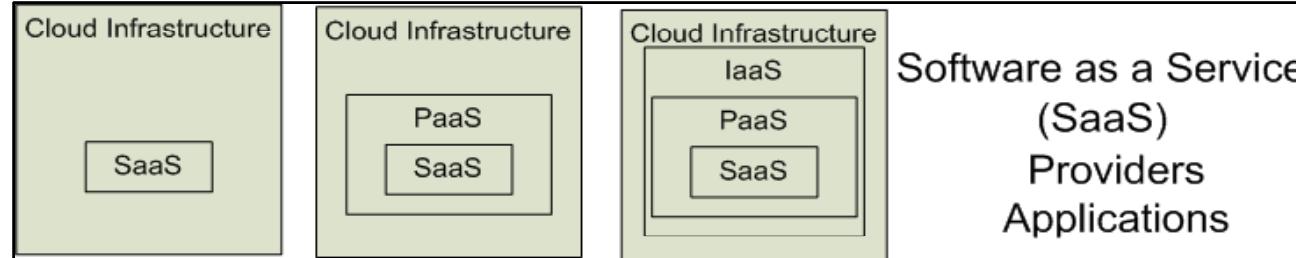


Google App



amazon web services™

rackspace®  
HOSTING



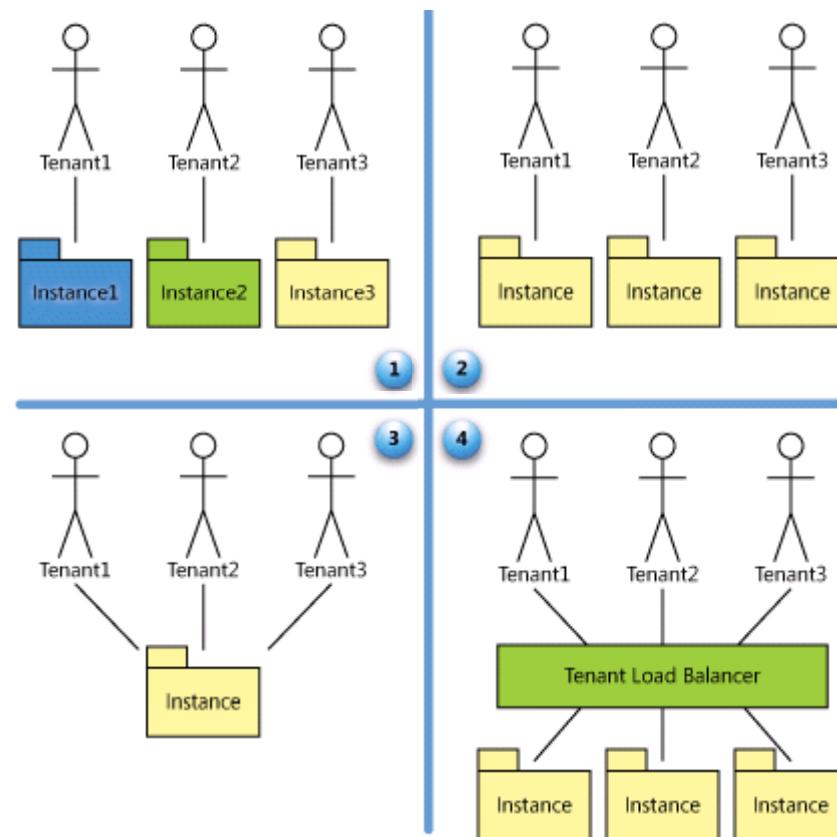
# SaaS Maturity Model

Level 1: Ad-Hoc/Custom –  
One Instance per customer

Level 2: Configurable per  
customer

Level 3: configurable & Multi-  
Tenant-Efficient

Level 4: Scalable, Configurable &  
Multi-Tenant-Efficient



# Different Cloud Computing Layers

|                                       |  |
|---------------------------------------|--|
| <b>Application Service<br/>(SaaS)</b> | MS Live/ExchangeLabs, IBM, Google Apps; Salesforce.com Quicken Online, Zoho, Cisco |
| <b>Application Platform</b>           | Google App Engine, Mosso, Force.com, Engine Yard, Facebook, Heroku, AWS            |
| <b>Server Platform</b>                | 3Tera, EC2, SliceHost, GoGrid, RightScale, Linode                                  |
| <b>Storage Platform</b>               | Amazon S3, Dell, Apple, ...  |

# Cloud Computing Service Layers

| Services               | Description   |
|------------------------|---|
| Application Focused    | <b>Services</b><br>Services - Complete business services such as PayPal, OpenID, OAuth, Google Maps, Alexa  |
|                        | <b>Application</b><br>Application - Cloud based software that eliminates the need for local installation such as Google Apps, Microsoft Online    |
|                        | <b>Development</b><br>Development - Software development platforms used to build custom cloud based applications (PAAS & SAAS) such as SalesForce |
| Infrastructure Focused | <b>Platform</b><br>Platform - Cloud based platforms, typically provided using virtualization, such as Amazon ECC, Sun Grid                        |
|                        | <b>Storage</b><br>Storage - Data storage or cloud based NAS such as CTERA, iDisk, CloudNAS  |
| Hosting                | Hosting - Physical data centers such as those run by IBM, HP, NaviSite, etc.  |

# Basic Cloud Characteristics

- The “**no-need-to-know**” in terms of the underlying details of infrastructure, applications interface with the infrastructure via the APIs.
- The “**flexibility and elasticity**” allows these systems to scale up and down at will
  - utilising the resources of all kinds
    - CPU, storage, server capacity, load balancing, and databases
- The “**pay as much as used and needed**” type of utility computing and the “**always on!, anywhere and any place**” type of network-based computing.

# Basic Cloud Characteristics

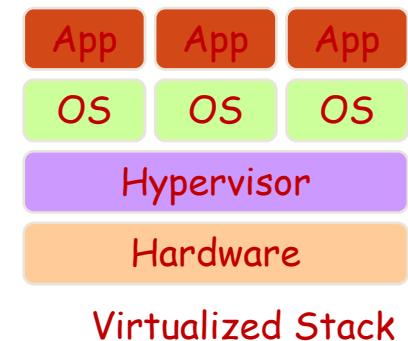
- Cloud are transparent to users and applications, they can be built in multiple ways
  - branded products, proprietary open source, hardware or software, or just off-the-shelf PCs.
- In general, they are built on clusters of PC servers and off-the-shelf components plus Open Source software combined with in-house applications and/or system software.

# Software as a Service (SaaS)

- SaaS is a model of software deployment where an application is hosted as a service provided to customers across the Internet.
- SaaS alleviates the burden of software maintenance/support
  - but users relinquish control over software versions and requirements.
- Terms that are used in this sphere include
  - **Platform as a Service** (PaaS) and
  - **Infrastructure as a Service** (IaaS)

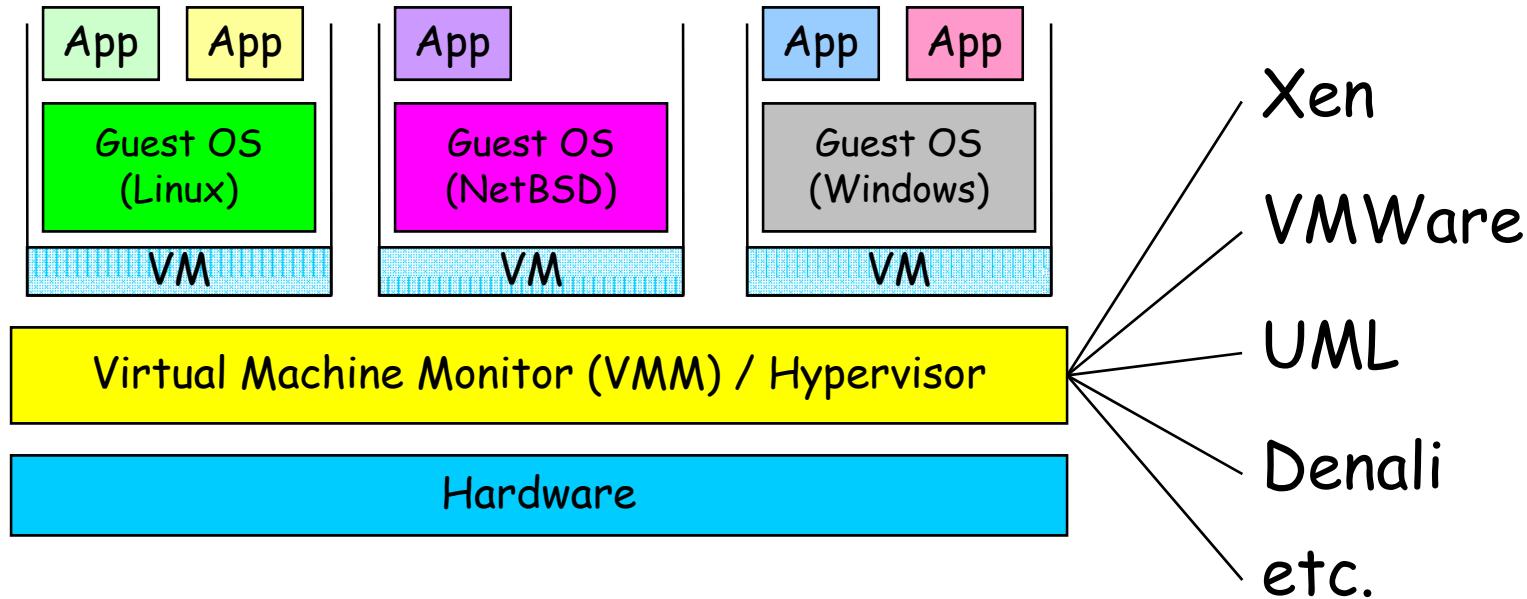
# Virtualization

- Virtual workspaces:
  - An abstraction of an execution environment that can be made dynamically available to authorized clients by using well-defined protocols,
  - Resource quota (e.g. CPU, memory share),
  - Software configuration (e.g. O/S, provided services).
- Implement on Virtual Machines (VMs):
  - Abstraction of a physical host machine,
  - Hypervisor intercepts and emulates instructions from VMs, and allows management of VMs,
  - VMWare, Xen, etc.
- Provide infrastructure API:
  - Plug-ins to hardware/support structures



# Virtual Machines

- VM technology allows multiple virtual machines to run on a single physical machine.



*Performance:* Para-virtualization (e.g. Xen) is very close to raw physical performance!

# Virtualization in General

- Advantages of virtual machines:
  - Run operating systems where the physical hardware is unavailable,
  - Easier to create new machines, backup machines, etc.,
  - Software testing using “clean” installs of operating systems and software,
  - Emulate more machines than are physically available,
  - Timeshare lightly loaded systems on one host,
  - Debug problems (suspend and resume the problem machine),
  - Easy migration of virtual machines (shutdown needed or not).
  - Run legacy systems!

# What is the purpose and benefits?

- Cloud computing enables companies and applications, which are system infrastructure dependent, to be infrastructure-less.
- By using the Cloud infrastructure on “pay as used and on demand”, all of us can save in capital and operational investment!
- Clients can:
  - Put their data on the platform instead of on their own desktop PCs and/or on their own servers.
  - They can put their applications on the cloud and use the servers within the cloud to do processing and data manipulations etc.

# Cloud-Sourcing

- Why is it becoming a Big Deal:
  - Using high-scale/low-cost providers,
  - Any time/place access via web browser,
  - Rapid scalability; incremental cost and load sharing,
  - Can forget need to focus on local IT.
- Concerns:
  - Performance, reliability, and SLAs,
  - Control of data, and service parameters,
  - Application features and choices,
  - Interaction between Cloud providers,
  - No standard API – mix of SOAP and REST!
  - Privacy, security, compliance, trust...

# Some Commercial Cloud Offerings



Amazon Elastic Compute Cloud (Amazon EC2) - Beta



TAP INTO THE  
**POWER OF NETWORK.COM**

3tera

info@3tera.com (040) 305 0050

CAREERS | Sitemap

Cloud Computing

Cloudware - Cloud Computing Without Compromise



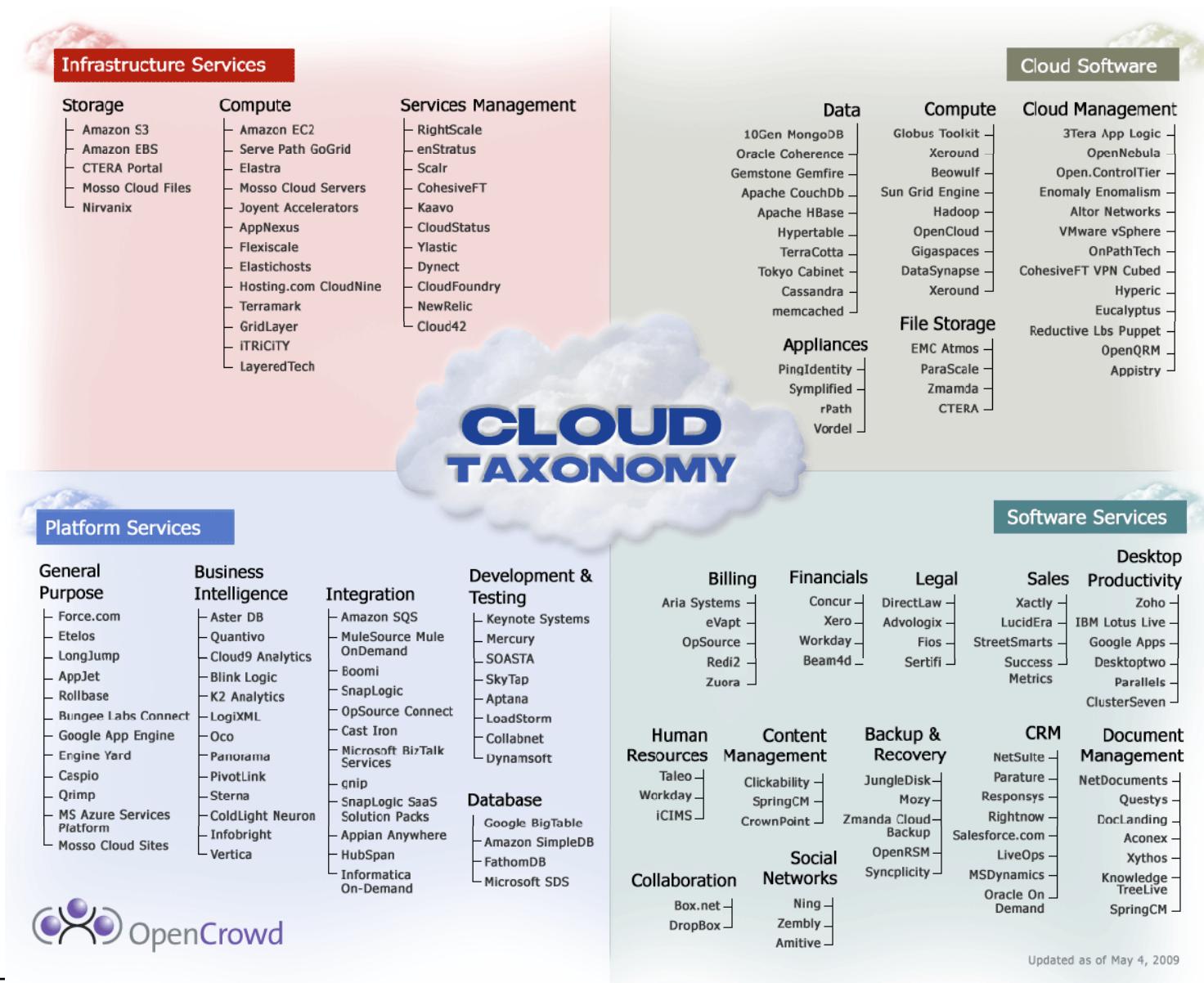
**MOSSO**  
the hosting cloud



**VERIO**

An NTT Communications Company

# Cloud Taxonomy



# Cloud Storage

- Several large Web companies are now exploiting the fact that they have data storage capacity that can be hired out to others.
  - allows data stored remotely to be temporarily cached on desktop computers, mobile phones or other Internet-linked devices.
- Amazon's Elastic Compute Cloud (EC2) and Simple Storage Solution (S3) are well known examples
  - Mechanical Turk

# Amazon Simple Storage Service (S3)

- Unlimited Storage.
- Pay for what you use:
  - \$0.20 per GByte of data transferred,
  - \$0.15 per GByte-Month for storage used,
  - Second Life Update:
    - 1TBytes, 40,000 downloads in 24 hours - \$200,



# Utility Computing – EC2

- Amazon Elastic Compute Cloud (EC2):
  - Elastic, marshal 1 to 100+ PCs via WS,
  - Machine Specs...,
  - Fairly cheap!
- Powered by Xen – a Virtual Machine:
  - Different from Vmware and VPC as uses “para-virtualization” where the guest OS is modified to use special hyper-calls:
  - Hardware contributions by Intel (VT-x/Vanderpool) and AMD (AMD-V).
  - Supports “Live Migration” of a virtual machine between hosts.
- Linux, Windows, OpenSolaris
- Management Console/AP

# EC2 – The Basics

- Load your image onto S3 and register it.
- Boot your image from the Web Service.
- Open up required ports for your image.
- Connect to your image through SSH.
- Execute your application...

# Opportunities and Challenges

- The use of the cloud provides a number of opportunities:
  - It enables services to be used without any understanding of their infrastructure.
  - Cloud computing works using economies of scale:
    - It potentially lowers the outlay expense for start up companies, as they would no longer need to buy their own software or servers.
    - Cost would be by on-demand pricing.
    - Vendors and Service providers claim costs by establishing an ongoing revenue stream.
  - Data and services are stored remotely but accessible from “anywhere”.

# Opportunities and Challenges

- In parallel there has been backlash against cloud computing:
  - Use of cloud computing means dependence on others and that could possibly limit flexibility and innovation:
    - The others are likely become the bigger Internet companies like Google and IBM, who may monopolise the market.
    - Some argue that this use of supercomputers is a return to the time of mainframe computing that the PC was a reaction against.
  - Security could prove to be a big issue:
    - It is still unclear how safe out-sourced data is and when using these services ownership of data is not always clear.
  - There are also issues relating to policy and access:
    - If your data is stored abroad whose policy do you adhere to?
    - What happens if the remote server goes down?
    - How will you then access files?
    - There have been cases of users being locked out of accounts and losing access to data.

# Advantages of Cloud Computing

- Lower computer costs:
  - You do not need a high-powered and high-priced computer to run cloud computing's web-based applications.
  - Since applications run in the cloud, not on the desktop PC, your desktop PC does not need the processing power or hard disk space demanded by traditional desktop software.
  - When you are using web-based applications, your PC can be less expensive, with a smaller hard disk, less memory, more efficient processor...
  - In fact, your PC in this scenario does not even need a CD or DVD drive, as no software programs have to be loaded and no document files need to be saved.

# Advantages of Cloud Computing

- Improved performance:
  - With few large programs hogging your computer's memory, you will see better performance from your PC.
  - Computers in a cloud computing system boot and run faster because they have fewer programs and processes loaded into memory...
- Reduced software costs:
  - Instead of purchasing expensive software applications, you can get most of what you need for free-ish!
    - most cloud computing applications today, such as the Google Docs suite.
  - better than paying for similar commercial software
    - which alone may be justification for switching to cloud applications.

# Advantages of Cloud Computing

- Instant software updates:
  - Another advantage to cloud computing is that you are no longer faced with choosing between obsolete software and high upgrade costs.
  - When the application is web-based, updates happen automatically
    - available the next time you log into the cloud.
  - When you access a web-based application, you get the latest version
    - without needing to pay for or download an upgrade.
- Improved document format compatibility.
  - You do not have to worry about the documents you create on your machine being compatible with other users' applications or OSes
  - There are potentially no format incompatibilities when everyone is sharing documents and applications in the cloud.

# Advantages of Cloud Computing

- Unlimited storage capacity:
  - Cloud computing offers virtually limitless storage.
  - Your computer's current 1 Tbyte hard drive is small compared to the hundreds of Pbytes available in the cloud.
- Increased data reliability:
  - Unlike desktop computing, in which if a hard disk crashes and destroy all your valuable data, a computer crashing in the cloud should not affect the storage of your data.
    - if your personal computer crashes, all your data is still out there in the cloud, still accessible
  - In a world where few individual desktop PC users back up their data on a regular basis, cloud computing is a data-safe computing platform!

# Advantages of Cloud Computing

- Universal document access:
  - That is not a problem with cloud computing, because you do not take your documents with you.
  - Instead, they stay in the cloud, and you can access them whenever you have a computer and an Internet connection
  - Documents are instantly available from wherever you are
- Latest version availability:
  - When you edit a document at home, that edited version is what you see when you access the document at work.
  - The cloud always hosts the latest version of your documents
    - as long as you are connected, you are not in danger of having an outdated version

# Advantages of Cloud Computing

- Easier group collaboration:
  - Sharing documents leads directly to better collaboration.
  - Many users do this as it is an important advantages of cloud computing
    - multiple users can collaborate easily on documents and projects
- Device independence.
  - You are no longer tethered to a single computer or network.
  - Changes to computers, applications and documents follow you through the cloud.
  - Move to a portable device, and your applications and documents are still available.

# Disadvantages of Cloud Computing

- Requires a constant Internet connection:
  - Cloud computing is impossible if you cannot connect to the Internet.
  - Since you use the Internet to connect to both your applications and documents, if you do not have an Internet connection you cannot access anything, even your own documents.
  - A dead Internet connection means no work and in areas where Internet connections are few or inherently unreliable, this could be a deal-breaker.

# Disadvantages of Cloud Computing

- Does not work well with low-speed connections:
  - Similarly, a low-speed Internet connection, such as that found with dial-up services, makes cloud computing painful at best and often impossible.
  - Web-based applications require a lot of bandwidth to download, as do large documents.
- Features might be limited:
  - This situation is bound to change, but today many web-based applications simply are not as full-featured as their desktop-based applications.
    - For example, you can do a lot more with Microsoft PowerPoint than with Google Presentation's web-based offering

# Disadvantages of Cloud Computing

- Can be slow:
  - Even with a fast connection, web-based applications can sometimes be slower than accessing a similar software program on your desktop PC.
  - Everything about the program, from the interface to the current document, has to be sent back and forth from your computer to the computers in the cloud.
  - If the cloud servers happen to be backed up at that moment, or if the Internet is having a slow day, you would not get the instantaneous access you might expect from desktop applications.

# Disadvantages of Cloud Computing

- Stored data might not be secure:
  - With cloud computing, all your data is stored on the cloud.
    - The question is How secure is the cloud?
  - Can unauthorised users gain access to your confidential data?
- Stored data can be lost:
  - Theoretically, data stored in the cloud is safe, replicated across multiple machines.
  - But on the off chance that your data goes missing, you have no physical or local backup.
    - Put simply, relying on the cloud puts you at risk if the cloud lets you down.

# Disadvantages of Cloud Computing

- HPC(High Performance Computing) Systems:
  - Not clear that you can run compute-intensive HPC applications that use MPI/OpenMP!
  - Scheduling is important with this type of application
    - as you want all the VM to be co-located to minimize communication latency!
- General Concerns:
  - Each cloud systems uses different protocols and different APIs
    - may not be possible to run applications between cloud based systems
  - Amazon has created its own DB system (not SQL 92), and workflow system (many popular workflow systems out there)
    - so your normal applications will have to be adapted to execute on these platforms.

# The Future

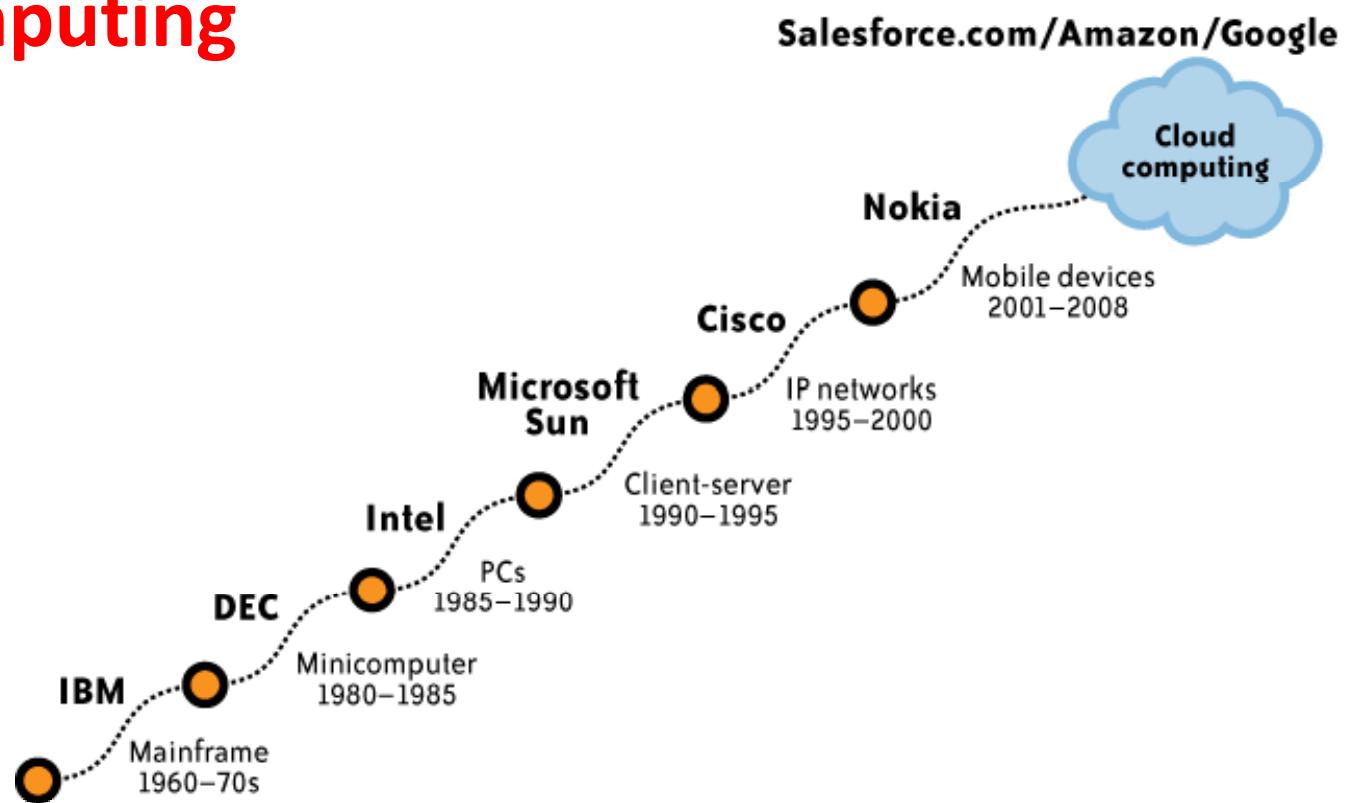
- Many of the activities loosely grouped together under cloud computing have already been happening and centralised computing activity is not a new phenomena
- Grid Computing was the last research-led centralised approach
- However there are concerns that the mainstream adoption of cloud computing could cause many problems for users
- Many new open source systems appearing that you can install and run on your local cluster
  - should be able to run a variety of applications on these systems

# Cloud Computing Architecture

# Definition of Cloud Computing

- Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.
- This cloud model promotes availability and is composed of five essential **characteristics**, three **service models**, and four **deployment models**.

# Evolution of model computing



## **What is Cloud Computing?**

**Common,  
Location-independent,**

**Online**

**Utility that is available on  
Demand**

--- (Chan, 2009)

# Essential Cloud Characteristics

- On-demand self-service
  - Get computing capabilities as needed automatically
- Broad network access
  - Services available over the net using desktop, laptop, PDA, mobile phone

# Essential Cloud Characteristics (Cont.)

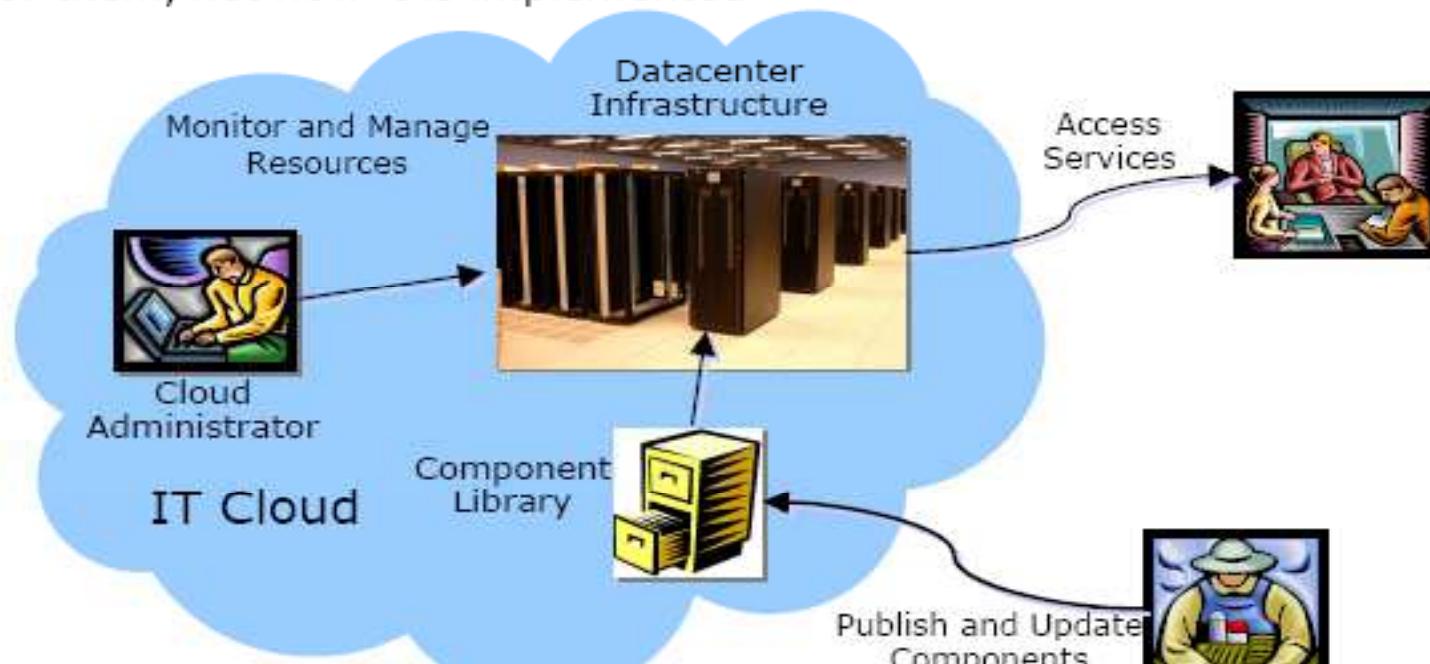
- Resource pooling
  - Location independence
  - Provider resources pooled to serve multiple clients
- Rapid elasticity
  - Ability to quickly scale in/out service
- Measured service
  - Control, optimize services based on metering

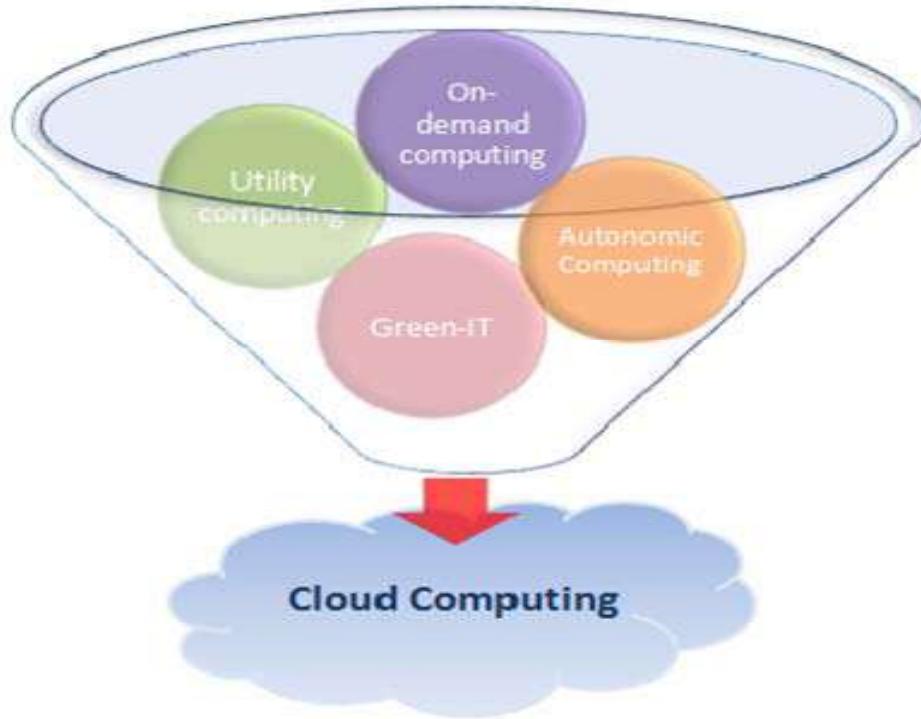
# Common Cloud Characteristics

- Cloud computing often leverages:
  - Massive scale
  - Homogeneity
  - Virtualization
  - Resilient computing
  - Low cost software
  - Geographic distribution
  - Service orientation
  - Advanced security technologies

Cloud Computing is an emerging style of computing in which **applications**, **data**, and **resources** are **provided as services** to users over the Web

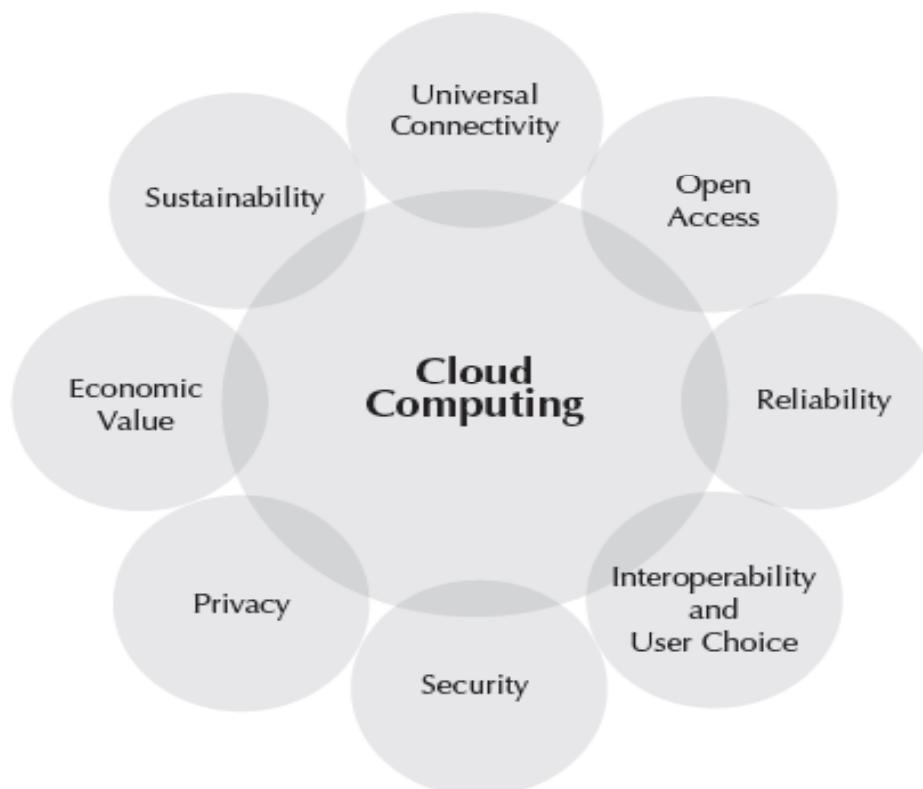
- Services provided may be available globally, always on, low in cost, “on demand”, massively scalable, “pay as you grow”, ...
- Consumers of the services need only care about *what* the service does for them, *not how* it is implemented





Cloud computing is an emerging business model that delivers computing services over the Internet in an elastic self-serviced, self-managed, cost-effective manner with guaranteed Quality of Service(QoS).

## Fundamental Elements of cloud Computing



## Five characteristics of cloud computing

Resource-Pooling

Scalable

On-Demand-Self-Service

uses-Internet-Technologies

Metered-by-Use

Elastic

## Pros and Cons of cloud computing



## **Types of clouds (Cloud Deployment Models)**

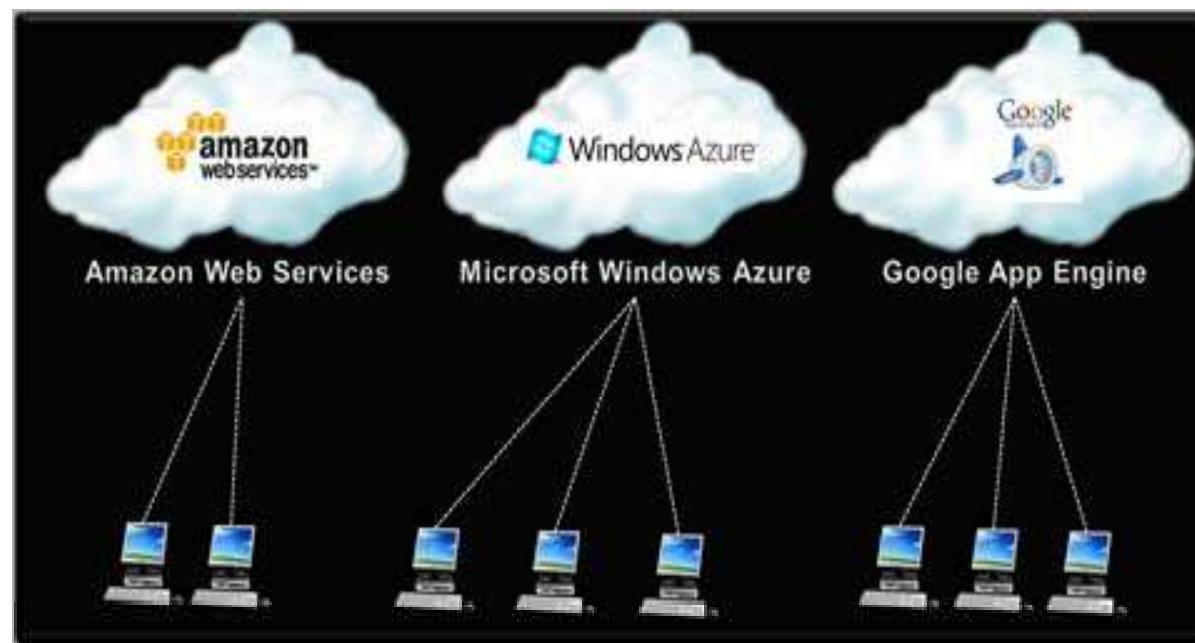
- Public/External cloud
- Hybrid/ Integrated cloud
- Private/Internal cloud
- Community/Vertical Clouds

A **public cloud** (also called External Cloud) is one based on the standard cloud computing model, in which a service provider makes resources, such as applications and storage, available to the general public over the Internet. Public cloud services may be free or offered on a pay-per-usage model.



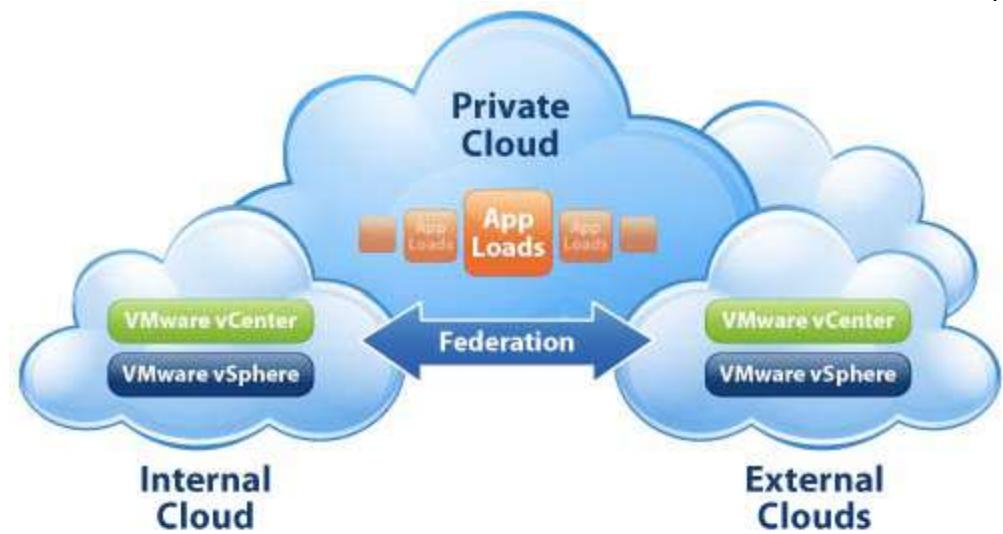
**The main benefits of using a public cloud service are:**

- Easy and inexpensive set-up because hardware, application and bandwidth costs are covered by the provider.
- Scalability to meet needs.
- No wasted resources because you pay for what you use.

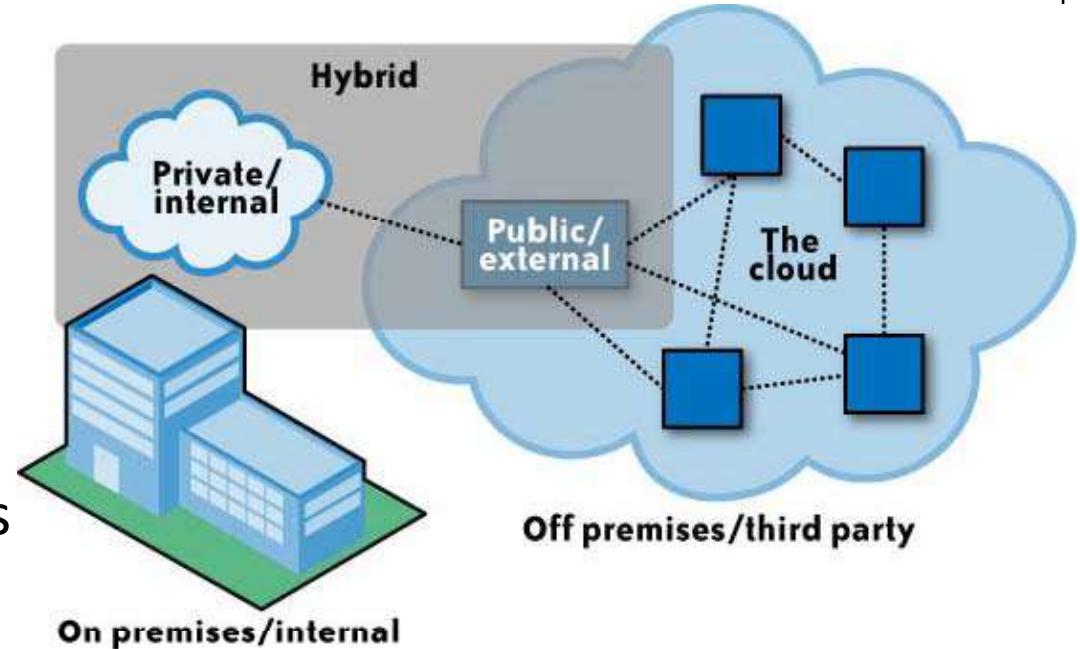


**Private cloud** (also called internal cloud or corporate cloud) is a marketing term for a proprietary computing architecture that provides hosted services to a limited number of people behind a firewall.

→ Advances in virtualization and distributed computing have allowed corporate network and datacenter administrators to effectively become service providers that meet the needs of their "customers" within the corporation.



**A hybrid cloud** is a composition of at least one private cloud and at least one public cloud. A hybrid cloud is typically offered in one of two ways: a vendor has a private cloud and forms a partnership with a public cloud provider, or a public cloud provider forms a partnership with a vendor that provides private cloud platforms



**Community clouds** are a deployment pattern suggested by NIST, where semi-private clouds will be formed to meet the needs of a set of related stakeholders or constituents that have common requirements or interests.

→ Communities of Interest (COI) constructs typical of the federal government may be enabled by community clouds to augment their wiki-centric collaboration processes with cloud enabled capabilities as well.

# Cloud reference model

## **Cloud Service Models :**

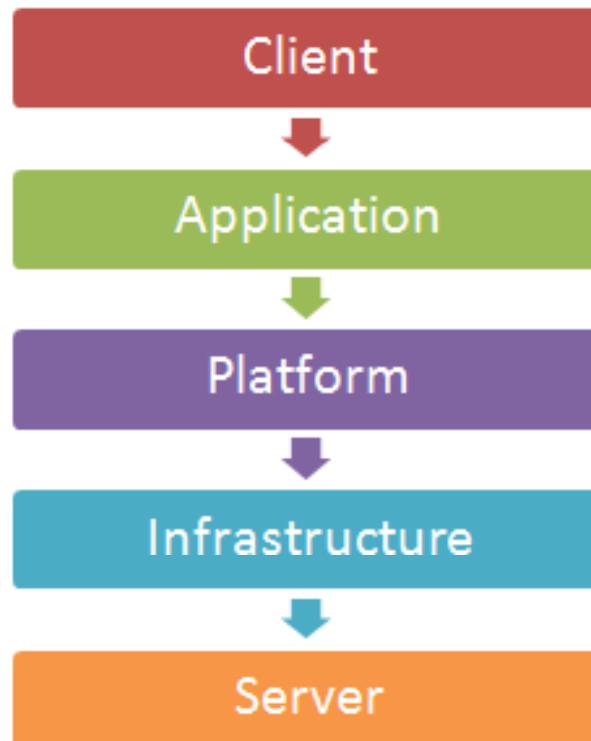
Cloud Computing can be broadly classified into three \*aaS, i.e., three layers of Cloud Stack, also known as **Cloud Service**

**Models or SPI Service Model:**

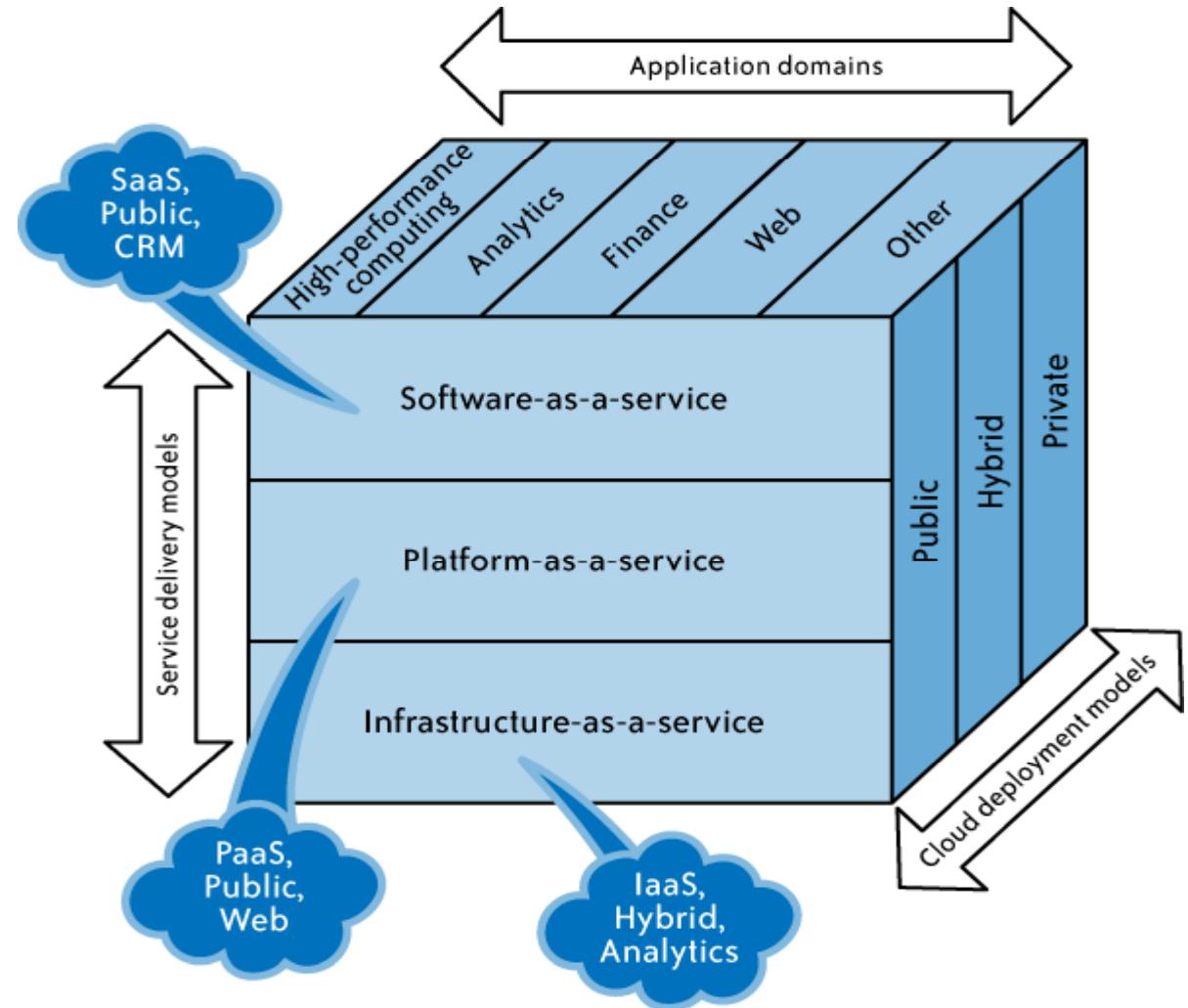
- Infrastructure-as-a-Service (**IaaS**)
- Platform-as-a-Service (**PaaS**)
- Software-as-a-Service (**SaaS**)

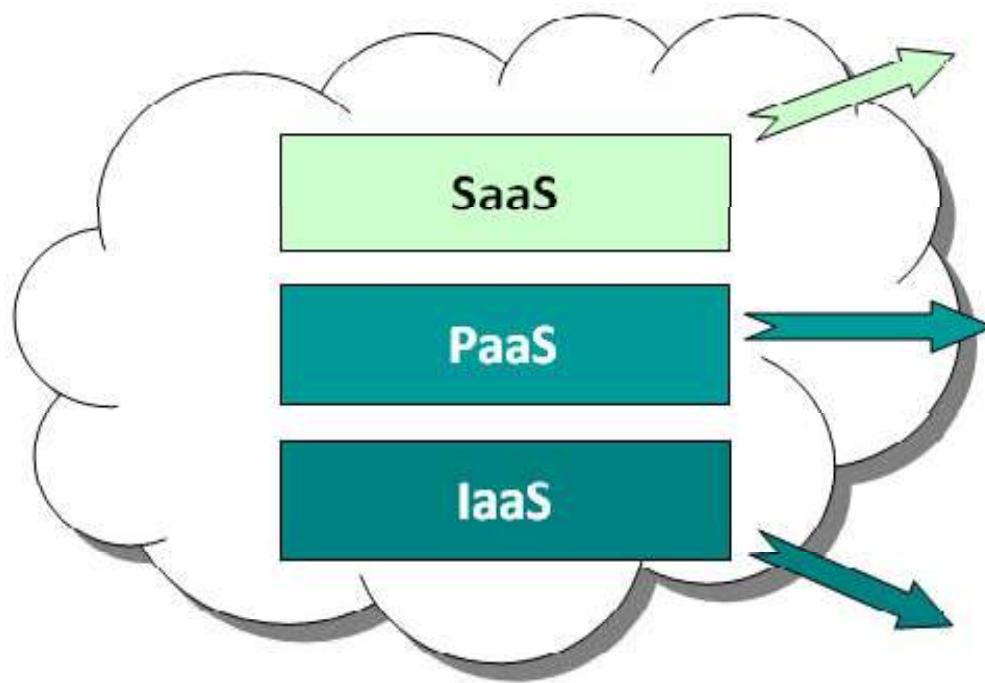
# Layers

- Client
- Application
- Platform
- Infrastructure
- Server



## Diagram for : Relationship between services, uses, and types of clouds





| Who Uses It              | What Services are available  | Why use it?  |
|--------------------------|--|--|
| Business Users           | EMail, Office Automation, CRM, Website Testing, Wiki, Blog, Virtual Desktop ...                      | To complete business tasks   |
| Developers and Deployers | Service and application test, development, integration and deployment                                | Create or deploy applications and services for users                                       |
| System Managers          | Virtual machines, operating systems, message queues, networks, storage, CPU, memory, backup services | Create platforms for service and application test, development, integration and deployment |

|                           | Amazon  | Google   | Salesforce  | Customer Implications   |
|---------------------------|---|--|---|---|
| Software as Service       |   |   |  | <ul style="list-style-type: none"> <li>+ Application logic, platform and infrastructure abstracted</li> <li>+ Significant reduction in effort to deploy, run and manage</li> <li>- Apps can be configured but may not meet highly customized requirements</li> </ul>  |
| Platform as Service       |   |  |  | <ul style="list-style-type: none"> <li>+ Platform &amp; infrastructure abstracted</li> <li>+ Custom apps can be built orders of magnitude more quickly and cheaply</li> <li>- Custom apps still need to be supported and managed</li> </ul>                           |
| Infrastructure as Service |  |  |   | <ul style="list-style-type: none"> <li>+ Physical infrastructure abstracted</li> <li>+ Can be scaled up and down as needed</li> <li>- Needs to be provisioned/managed</li> <li>- Higher levels of stack still need to be managed, maintained and supported</li> </ul> |

SaaS  
Software as a Service

PaaS  
Platform as a Service

IaaS  
Infrastructure as a Service

SaaS  
Software as a Service

SaaS

## Software delivery model

- Increasingly popular with SMEs
- No hardware or software to manage
- Service delivered through a browser

SaaS

# Advantages

- Pay per use
- Instant Scalability
- Security
- Reliability
- APIs (application programming Interface)

SaaS

## Examples

- CRM
- Financial Planning
- Human Resources
- Word processing

## Commercial Services:

- Salesforce.com
- emailcloud

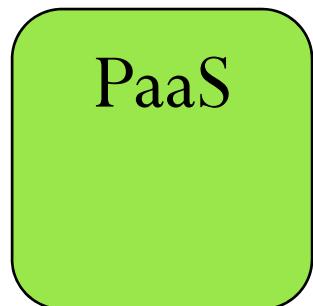
PaaS  
Platform as a Service

# Platform delivery model

PaaS

- Platforms are built upon Infrastructure, which is expensive
- Estimating demand is not a science!
- Platform management is not fun!

# Popular services



- Storage
- Database
- Scalability

# Advantages

PaaS

- Pay per use
- Instant Scalability
- Security
- Reliability
- APIs

PaaS

# Examples

- Google App Engine
- Mosso
- AWS: S3

IaaS

Infrastructure as a Service

# Computer infrastructure delivery model

Access to infrastructure stack:

- Full OS access
- Firewalls
- Routers
- Load balancing

IaaS

# Advantages

- Pay per use
- Instant Scalability
- Security
- Reliability
- APIs

IaaS

# Examples

- Flexiscale
- AWS: EC2

IaaS

SaaS  
Software as a Service

PaaS  
Platform as a Service

IaaS  
Infrastructure as a Service

SaaS

PaaS

IaaS

# Common Factors

- Pay per use
- Instant Scalability
- Security
- Reliability
- APIs

SaaS

PaaS

IaaS

# Advantages

- Lower cost of ownership
- Reduce infrastructure management responsibility
- Allow for unexpected resource loads
- Faster application rollout

SaaS

# Cloud Economics

PaaS

IaaS

- Virtualisation lowers costs by increasing utilisation
- Economies of scale afforded by technology
- Automated update policy

SaaS

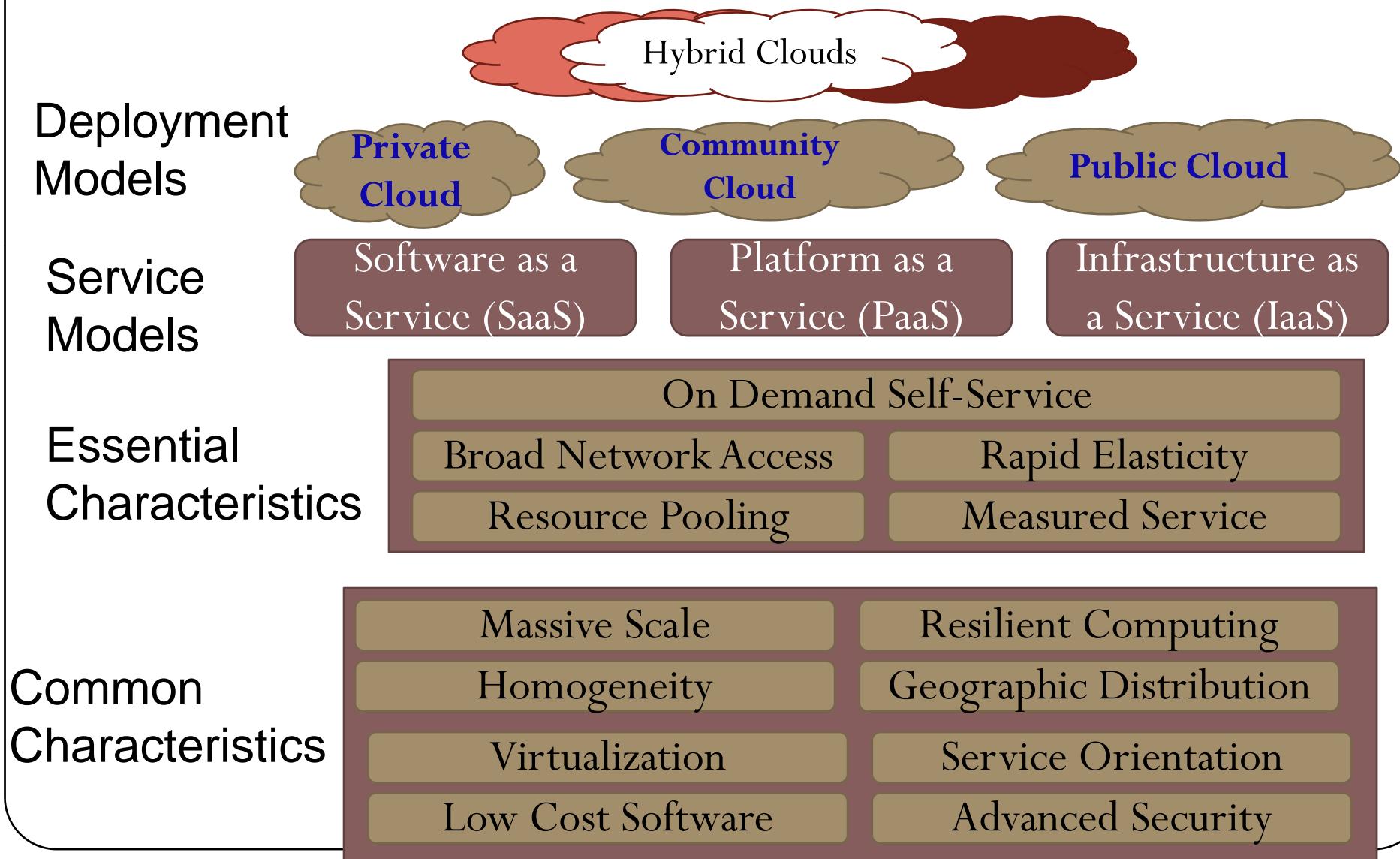
PaaS

IaaS

# Risks

- Security
- Downtime
- Access
- Dependency
- Interoperability

# The NIST (National Institute of Standards and Technology) Cloud Definition Framework



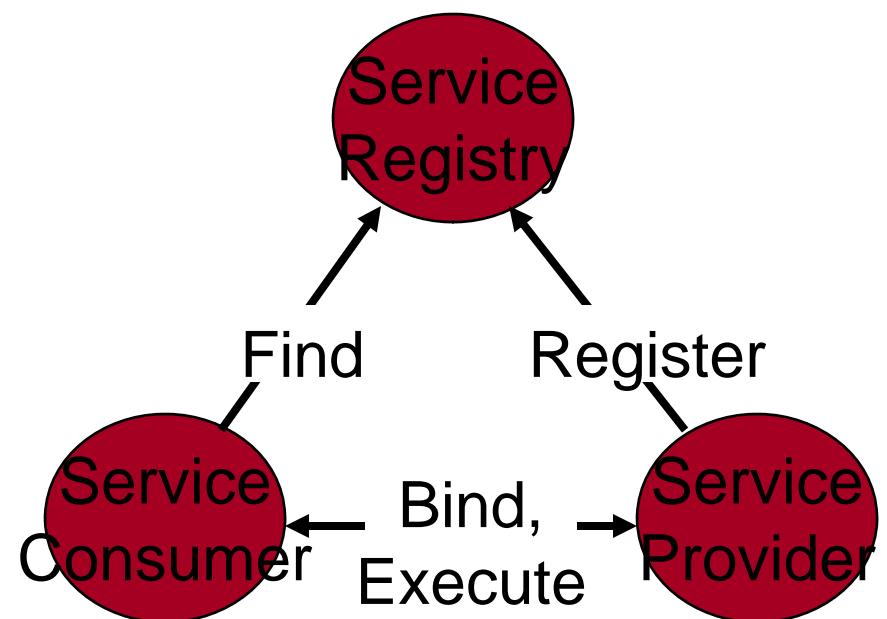
# **SERVICE ORIENTED ARCHITECTURE**

# What is Service Oriented Architecture (SOA)?

- “A service-oriented architecture is essentially a collection of services. These services communicate with each other.
- The communication can involve either simple data passing or it could involve two or more services coordinating some activity.
- Some means of connecting services to each other is needed.”
- “Service-oriented architecture (SOA) provides methods for systems development and integration where systems group functionality around business processes and package these as *interoperable services*.
- An SOA infrastructure allows different applications to exchange data with one another as they participate in business processes.
- SOA separates functions into distinct units, or services, which developers make accessible over a network in order that users can combine and reuse them in the production of business applications “

# What is Service Oriented Architecture (SOA)?

- Is not a computing architecture but a style of programming
- An SOA application is a composition of services
- A “service” is the building block/ unit of an SOA
- Services encapsulate a business process
- Service Providers Register themselves
- Service use involves: Find, Bind, Execute
- Most well-known instance is Web Services



# SOA Actors

- Service Provider

- From a business perspective, this is the owner of the service. From an architectural perspective, this is the platform that provides access to the service.

- Service Registry

- This is an information space of service *descriptions* where service providers publish their services and service requesters find services and obtain binding information for services.
  - Allows service consumers to locate service providers that meet required criteria

- Service Consumer

- From a business perspective, this is the business that requires certain function to be fulfilled. From an architectural perspective, this is the *client* application that is looking for and eventually invoking a service.

# SOA Principles

- Formal contract
- Loose coupling
- Abstraction
- Reusability
- Autonomy
- Statelessness
- Discoverability
- Composability

Thomas Erl, SOA Principles of Service Design, Prentice Hall 2007 ISBN:0132344823

# SOA Principles – Formal contract

- According to SOA Formal contract principle every service needs to have an official, standardized, formal contract.
- A great deal of emphasis is placed on specific aspects of contract design, including:
  - the manner in which services express **functionality (functional description contract)**
  - how data types and data models are defined (**information model**)
  - how policies are asserted and attached. (**non-functional description contract**)
  - how interaction with the service is to be performed (**behavioral contract**)

# SOA Principles – Loose coupling

- SOA is a loosely coupled arrangement of services and service consumers. At design time, loose coupling means that services are designed with no affinity to any particular service consumer. Inside the service, no information is assumed as to the purpose, technical nature or business nature of the service consumer. Thus, a service is fully decoupled from a service consumer.
- However, the service consumer is dependent on the service (that is, it embeds literal references to service interfaces). Thus, SOA is a semi-coupled (or *loosely coupled*) architecture. It differs from an *event-driven architecture*, in which all participating software components are decoupled from others, and also from a *monolithic architecture*, in which all software components are designed to operate only in the initially intended context (that is, logically tightly coupled).
- Design-time loose coupling is essential to SOA because it enables the non-intrusive reuse of service interfaces. However, tools can't guarantee design-time loose coupling. *Poorly designed services, which are logically locked into their service consumers, may render the entire application monolithic — despite the use of SOA-style technologies.*

“Introduction to Service-Oriented Architecture”, Yefim V. Natis, Roy W. Schulte,  
14 April 2003

# SOA Principles – Abstraction

- This principle emphasizes the need to hide as much of the underlying details of a service as possible.
- By using abstraction previously described loosely coupled relationship is directly enabled and preserved
  - There are 4 levels of abstraction in SOA as:
    - *technology abstraction*
    - *functional abstraction*
    - *programming logic abstraction*
    - *quality of service abstraction*

# SOA Principles – Reusability

- The reusability principle suggest to contain and express agnostic logic as services that can be positioned as reusable enterprise resources
- Reusability will:
  - Allow for service logic to be repeatedly leveraged over time so as to achieve a high Return on investment( ROI)
  - Increase business agility on an organizational level
  - Enable the creation of service inventories that can be easily integrated and used in various use-cases

Thomas Erl, SOA Principles of Service Design, Prentice Hall 2007 ISBN:0132344823

# SOA Principles – Autonomy

- SOA Autonomy principle implies that services have control over the solution logic they implement.
- SOA Autonomy/ Service Autonomy can be observed as various levels:
  - ***Runtime autonomy*** – represents the amount of control a service has over its execution environment at runtime
  - ***Design-time autonomy*** – represents the amount of governance control a service owner has over the service design

# SOA Principles – Statelessness

- This means a service must do its best to hold onto state information pertaining to an interaction for as small a duration as possible, e.g., do not retain awareness of a message once it is processed.
- Statelessness in a service means that if the service is enlisted in a flow, than it doesn't retain any state referring to the enclosing flow. From a message perspective, it means that once a service has received and processed a message, it doesn't retain memory of the passage of that message.
- This helps with concurrent access scaling

# Statelessness in SOA and REST

- SOA and REST share the Statelessness principle
- REST provides explicit state transitions
- REST Servers are stateless and messages can be interpreted without examining history.
- Persistent data can be given explicit URIs on the server.
- Messages can refer to persistent data through links to Uniform Resource Identifier(URI)s.

# Statelessness in SOA and REST

- **In SOA**
  - Stateless communication although communication can be stateful as well
    - Received or sent messages can trigger state change
    - Operations requiring sequence of messages
  - Capable to support transactions
    - set of operations with pass or fail results
  - Tighter coupling between components
- **In REST**
  - Stateless communication
    - Document transfer only
    - A party is not aware of its partner current state
  - Party receiving information can decide how to process it
  - HTTP caching possible
  - Looser coupling between components

# SOA Principles – Discoverability

- SOA Discoverability is meant to help one avoid the accidental creation of services that are either redundant or implement logic that is redundant. The discoverability principle can be referred to the ***design of an individual service so that it becomes as discoverable as possible*** – no matter whether the discoverability extension or product actually exists in the surrounding implementation environment.
- Discovery is a central task in SOA. SOA Discoverability is centered on Service Discoverability. Service Discoverability is meant to refer to the technology architecture's ability to provide a mechanism of discovery, for example a ***service directory, service registry or a service search engine***.
- Services be designed as resources that are highly discoverable in some fashion. Each service should be equipped with the metadata that is required to properly communicate its capabilities and meaning.

# SOA Principles – Composability

- Allow us to chain services together to provide new services
- Composition has the advantage that one can put together composite applications at a speed greater than writing one from scratch
- Building new services and application becomes quicker and cheaper

# SOA Properties – Self- Properties

- Most service architectures aim for „self-“ properties to reduce management load by design:
  - Self-Configuration
  - Self-Organization
  - Self-Healing
  - Self-Optimization
  - Self-Protection

# Self-Configuration

- Service architectures comprise of a huge amount of different components (services and hardware). Configuration is a challenging task in such environments.
- The idea of **self-configuration** is the adoption of the self-organization and fully distributed cooperation capabilities known from groups with cooperative social behavior which collaborate to solve a problem. Every member of the group can decide which part of the problem it can solve and which “QoS” it can provide.

# Self-Organization

- A system is **self-organizing** if it automatically, dynamically and autonomously adapts itself to achieve global goals more efficiently under changing conditions.

# Self-Healing

- The task of **self-healing** is to assure that a system meets some defined conditions as far as possible, i.e. to guarantee that all services running in the framework stay available, even in the case of partial outages in the system.

# Self-Optimization

- The **self-configuration** is responsible to find a good distribution of the services in terms of the given resources of the service description. The target of the **self-optimization** is to distribute the services of the application in a way that the considered resources are utilized evenly.
- A typical approach is to find an adequate configuration at the beginning and to optimize the application during runtime.

# Self-Protection

- **Self-protection** techniques cope with intentionally or unintentionally malicious peers or services in a framework. They behave as the “immune system” of a service framework as they are permissive to good-natured services and messages but can detect appearing malicious events.

# SOA Benefits

## **Business Benefits**

- Focus on Business Domain solutions
- Leverage Existing Infrastructure
- Agility

## **Technical Benefits**

- Loose Coupling
- Autonomous Service
- Location Transparency
- Late Binding

# Security Issues in the Cloud

- In theory, minimizing any of the issues would help:
  - Loss of Control
    - Take back control
      - Data and apps may still need to be on the cloud
      - But can they be managed in some way by the consumer?
    - Lack of trust
      - Increase trust (mechanisms)
        - Technology
        - Policy, regulation
        - Contracts (incentives): topic of a future talk
    - Multi-tenancy
      - Private cloud
        - Takes away the reasons to use a cloud in the first place
      - Strong separation

## CSE 486/586 Distributed Systems Paxos --- 1

Steve Ko  
Computer Sciences and Engineering  
University at Buffalo

CSE 486/586, Spring 2014

### Recap

- NFS
  - Caching with write-through policy at close()
  - Stateless server
- One power efficient design: FAWN ([Fast Array of Wimpy Nodes](#))
  - Embedded CPUs & Flash storage
  - Write problem: block erasure first
  - FTL presents a logical structure different from the physical structure. Physically, it's log-structured.

### FTL: Flash Translation Layer

CSE 486/586, Spring 2014

2

### Paxos

- A consensus algorithm
  - Known as one of the most efficient & elegant consensus algorithms
  - If you stay close to the field of distributed systems, you'll hear about this algorithm over and over.
- What? Consensus? What about FLP (the impossibility of consensus)?
  - Obviously, it doesn't solve FLP.
  - It relies on failure detectors to get around it.
- Plan
  - Brief history (with a lot of quotes)
  - The protocol itself
  - How to "discover" the protocol

(FLP: Fischer, Lynch, and Paterson)

CSE 486/586, Spring 2014

3

### Brief History

- Developed by Leslie Lamport (from the Lamport clock)
- *"A fault-tolerant file system called Echo was built at SRC in the late 80s. The builders claimed that it would maintain consistency despite any number of non-Byzantine faults, and would make progress if any majority of the processors were working."*
- *"I decided that what they were trying to do was impossible, and set out to prove it. Instead, I discovered the Paxos algorithm."*
- *"I decided to cast the algorithm in terms of a parliament on an ancient Greek island (Paxos)."*

CSE 486/586, Spring 2014

4

### Brief History

- The paper abstract:
  - *"Recent archaeological discoveries on the island of Paxos reveal that the parliament functioned despite the peripatetic propensity of its part-time legislators. The legislators maintained consistent copies of the parliamentary record, despite their frequent forays from the chamber and the forgetfulness of their messengers. The Paxos parliament's protocol provides a new way of implementing the state-machine approach to the design of distributed systems."*
- *"I gave a few lectures in the persona of an Indiana-Jones-style archaeologist."*
- *"My attempt at inserting some humor into the subject was a dismal failure. People who attended my lecture remembered Indiana Jones, but not the algorithm."*

CSE 486/586, Spring 2014

5

### Brief History

- People thought that Paxos was a joke.
- Lamport finally published the paper 8 years later in 1998 after it was written in 1990.
  - Title: "The Part-Time Parliament"
- People did not understand the paper.
- Lamport gave up and wrote another paper that explains Paxos in simple English.
  - Title: "Paxos Made Simple"
  - Abstract: "The Paxos algorithm, when presented in plain English, is very simple."
- Still, it's not the easiest algorithm to understand.
- So people started to write papers and lecture notes to explain "Paxos Made Simple." (e.g., "Paxos Made Moderately Complex", "Paxos Made Practical", etc.)

CSE 486/586, Spring 2014

6

## Review: Consensus

- How do people agree on something?
  - Q: should Steve give an A to everybody taking CSE 486/586?
  - Input: everyone says either yes/no.
  - Output: an agreement of yes or no.
  - FLP: this is impossible even with one-faulty process and arbitrary delays.
- Many distributed systems problems can cast into a consensus problem
  - Mutual exclusion, leader election, total ordering, etc.
- Paxos
  - How do multiple processes agree on a value?
  - Under failures, network partitions, message delays, etc.

CSE 486/586, Spring 2014

7

## Review: Consensus

- People care about this!
- Real systems implement Paxos
  - Google Chubby
  - MS Bing cluster management
  - Etc.
- Amazon CTO Werner Vogels (in his blog post “Job Openings in My Group”)
  - *“What kind of things am I looking for in you?”*
  - *“You know your distributed systems theory: You know about logical time, snapshots, stability, message ordering, but also acid and multi-level transactions. You have heard about the FLP impossibility argument. You know why failure detectors can solve it (but you do not have to remember which one diamond-w was). You have at least once tried to understand Paxos by reading the original paper.”*

CSE 486/586, Spring 2014

8

## CSE 486/586 Administrivia

- PA3 scores will be posted by tonight.
- Midterm scores will be posted by tonight.
- PA4 released.
  - Tester will be released soon.
  - A small correction will be posted as well.

CSE 486/586, Spring 2014

9

## Paxos Assumptions & Goals

- The network is **asynchronous** with message delays.
- The network can **lose or duplicate** messages, but **cannot corrupt** them.
- Processes can **crash**.
- Processes are **non-Byzantine** (only crash-stop).
- Processes have **permanent storage**.
- Processes can **propose** values.
- The goal: every process agrees on a value out of the proposed values.

CSE 486/586, Spring 2014

10

## Desired Properties

- Safety
  - Only a value that has been proposed can be chosen
  - Only a single value is chosen
  - A process never learns that a value has been chosen unless it has been
- Liveness
  - Some proposed value is eventually chosen
  - If a value is chosen, a process eventually learns it

CSE 486/586, Spring 2014

11

## Roles of a Process

- Three roles
- **Proposers:** processes that propose values
- **Acceptors:** processes that accept (i.e., consider) values
  - “Considering a value”: the value is a candidate for consensus.
  - Majority acceptance → choosing the value
- **Learners:** processes that learn the outcome (i.e., chosen value)
- In reality, a process can be any one, two, or all three.
- Important requirements
  - The protocol should work under process failures and with delayed and lost messages.
  - The consensus is reached via a majority ( $> \frac{1}{2}$ ).

CSE 486/586, Spring 2014

12

## Roles of a Process

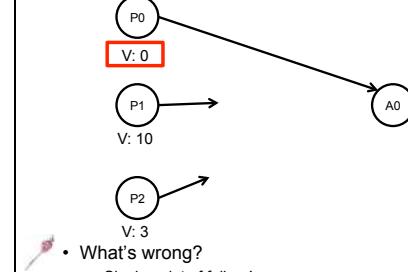
- In reality, a process can be any one, two, or all three.
- Important requirements
  - The protocol should work under process failures and with delayed and lost messages.
  - The consensus is reached via a majority ( $> \frac{1}{2}$ ).
- Example: a replicated state machine
  - All replicas agree on the order of execution for concurrent transactions
  - All replica assume all roles, i.e., they can each propose, accept, and learn.

CSE 486/586, Spring 2014

13

## First Attempt

- Let's just have one acceptor, choose the first one that arrives, & tell the proposers about the outcome.



- What's wrong?

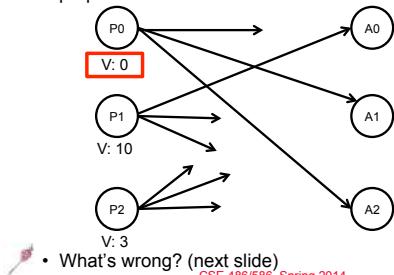
- Single point of failure!

CSE 486/586, Spring 2014

14

## Second Attempt

- Let's have multiple acceptors; each accepts the first one; then all choose the majority and tell the proposers about the outcome.



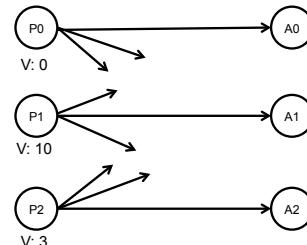
- What's wrong? (next slide)

CSE 486/586, Spring 2014

15

## Second Attempt

- One example, but many other possibilities



CSE 486/586, Spring 2014

16

## Paxos

- Let's have each acceptor accept (i.e., consider *multiple proposals*).
  - An acceptor accepting a proposal doesn't mean it will be chosen. A majority should accept it.
  - "Hope" that one of the multiple accepted proposals will have a vote from a majority (will get back to this later)
- Paxos: how do we select one value when there are multiple acceptors accepting multiple proposals?

CSE 486/586, Spring 2014

17

## Paxos Protocol Overview

- A proposal should have an ID.
  - $(\text{proposal } \#, \text{ value}) == (N, V)$
  - The proposal # strictly increasing and globally unique across all proposers
- Three phases
  - Prepare phase: a proposer learns previously-accepted proposals from the acceptors.
  - Propose phase: a proposer sends out a proposal.
  - Learn phase: learners learn the outcome.

CSE 486/586, Spring 2014

18

## Paxos Protocol Overview

- Rough description of the **proposers**
  - Before a proposer proposes a value, it will ask acceptors if there is any proposed value already.
  - If there is, the proposer will propose the same value, rather than proposing another value.
  - The behavior is **altruistic**: the goal is to reach a consensus, rather than making sure that "my value" is chosen.
- Rough description of the **acceptors**
  - The goal for acceptors is to accept the highest-numbered proposal coming from all proposers.
  - An acceptor tries to accept a value V with the highest proposal number N.
- Rough description of the **learners**
  - All learners are passive and wait for the outcome.

CSE 486/586, Spring 2014

19

## Paxos Phase 1

- A proposer chooses its proposal number N and sends a **prepare request** to acceptors.
  - "Hey, have you accepted any proposal yet?"
- An acceptor needs to reply:
  - **If it accepted anything**, the accepted proposal and its value with **the highest proposal number less than N**
  - A **promise to not accept** any proposal numbered **less than N** any more (to make sure that it doesn't alter the result of the reply).

CSE 486/586, Spring 2014

20

## Paxos Phase 2

- If a proposer receives a reply from a majority, it sends an **accept request** with the proposal (N, V).
  - V: the value from **the highest proposal number N** from the replies (i.e., the accepted proposals returned from acceptors in phase 1)
  - Or, **if no accepted proposal was returned in phase 1**, a new value to propose.
- Upon receiving (N, V), acceptors either:
  - **Accept** it
  - Or, **reject** it if there was another prepare request with N' higher than N, and it replied to it.

CSE 486/586, Spring 2014

21

## Paxos Phase 3

- Learners need to know which value has been chosen.
- Many possibilities
- One way: have each acceptor respond to all learners
  - Might be effective, but expensive
- Another way: elect a "distinguished learner"
  - Acceptors respond with their acceptances to this process
  - This distinguished learner informs other learners.
  - Failure-prone
- Mixing the two: a set of distinguished learners

CSE 486/586, Spring 2014

22

## Problem: Progress (Liveness)

- **There's a race condition for proposals.**
- P0 completes phase 1 with a proposal number N0
- Before P0 starts phase 2, P1 starts and completes phase 1 with a proposal number N1 > N0.
- P0 performs phase 2, acceptors reject.
- Before P1 starts phase 2, P0 restarts and completes phase 1 with a proposal number N2 > N1.
- P1 performs phase 2, acceptors reject.
- ... (this can go on forever)

CSE 486/586, Spring 2014

23

## Providing Liveness

- Solution: **elect a distinguished proposer**
  - I.e., have only one proposer
- If the distinguished proposer can successfully communicate with a majority, the protocol guarantees liveness.
  - I.e., if a process plays all three roles, Paxos can tolerate failures  $f < 1/2 * N$ .
- Still needs to get around FLP for the leader election, e.g., having a failure detector

CSE 486/586, Spring 2014

24

## Summary

- Paxos
  - A consensus algorithm
  - Handles crash-stop failures ( $f < 1/2 * N$ )
- Three phases
  - Phase 1: prepare request/reply
  - Phase 2: accept request/reply
  - Phase 3: learning of the chosen value

CSE 486/586, Spring 2014

25

## Acknowledgements

- These slides contain material developed and copyrighted by Indranil Gupta (UIUC).

CSE 486/586, Spring 2014

26

## Edge Computing: Use Cases and Challenges

Grace A. Lewis

Software Engineering Institute  
Carnegie Mellon University  
Pittsburgh, PA 15213



Copyright 2019 Carnegie Mellon University. All Rights Reserved.

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

The view, opinions, and/or findings contained in this material are those of the author(s) and should not be construed as an official Government position, policy, or decision, unless designated by other documentation.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

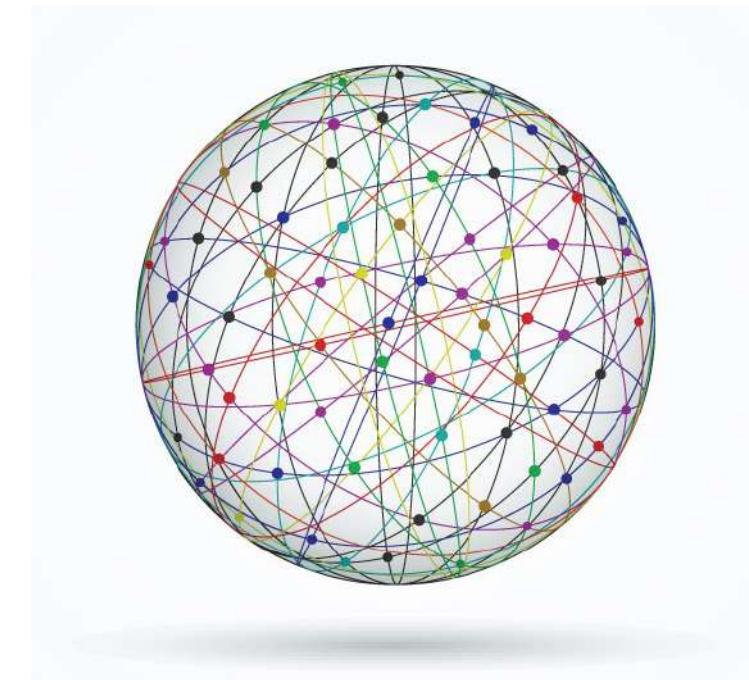
This material was prepared for the exclusive use of SCSS 2019 and may not be used for any other purpose without the written consent of permission@sei.cmu.edu.

DM19-0872

# Edge Computing

Idea is to push applications, data and computing power to the edge of the Internet, in close proximity to mobile devices, sensors, and end users

An early example is Akamai, with servers around the world to distribute web site content from locations close to the user (content delivery networks, or CDNs)



# Edge Computing: Drivers

## Latency

- data processing close to where it originates avoids round-trip time to the cloud

## Bandwidth

- optimization of communication to and from the cloud

## Privacy/security

- sensitive data stays local

## Connectivity

- continued processing (in some cases) despite lack of connectivity to the cloud

## Local dependencies

- data processing close to points of interaction with end users and other system components



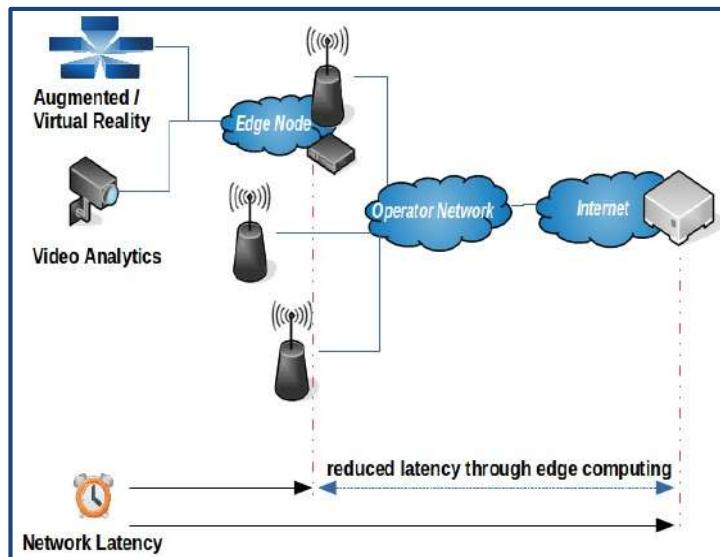
# Edge Computing: The Telco View

Opportunity for providing edge computing devices in existing infrastructure

- e.g., micro data centers at the base of cellular towers

Multiple organizations seeking standardization:  
Multi-Access Edge Computing (MEC), Open Edge Computing (OEC), OpenFog consortium, etc.

Business model is still not clear: Who pays for the service? Consumer? Content Provider?



Edge Computing according to the Open Edge Computing Initiative [1]

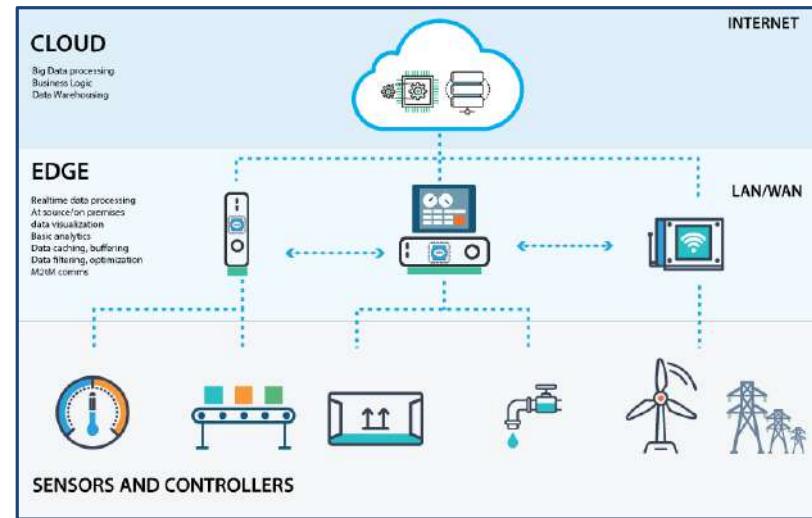
# Edge Computing: The Cloud Provider View

Goal is mainly to provide

- Content Delivery Network (CDN) services
- IoT data processing and aggregation for data in transit to the cloud

Examples

- Azure IoT Edge — deploy business logic to edge devices and monitor from the cloud
- Amazon
  - AWS CloudFront — CDN Service, includes Lambda@Edge
  - AWS Greengrass — connected IoT devices can run AWS Lambda functions and other code on locally-collected data



Industrial IoT: IoT to Edge to Cloud [2]

# Edge Computing: The “Appliance” View

Goal is to provide a “data center in a box” to push cloud computing capabilities to the edge

- Often combined with networking capabilities such as edge gateways and smart routers

Many players in this space, such as Amazon, Cisco, Dell EMC, HPE, etc.

## Disconnected Operations

AWS Snowball Edge — large-scale data transfer service with an embedded computing platform (based on AWS Greengrass plus Lambda functions)



*Snowball Edge Device [3]*

# Opportunities for DoD and Government

Edge Computing via “appliances” can provide computation and data to support a wide variety of missions

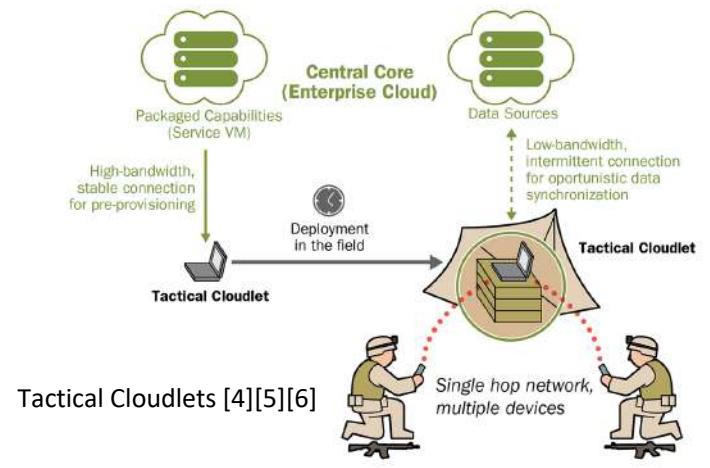
- Military
- Humanitarian
- Public safety
- Public service



# Computation and Data in Disconnected Environments

Providing computation-intensive capabilities and data at the edge when there is no access to the cloud

- Speech recognition
- Face recognition
- Speech translation
- Image recognition
- Image processing
- Air/water quality analysis



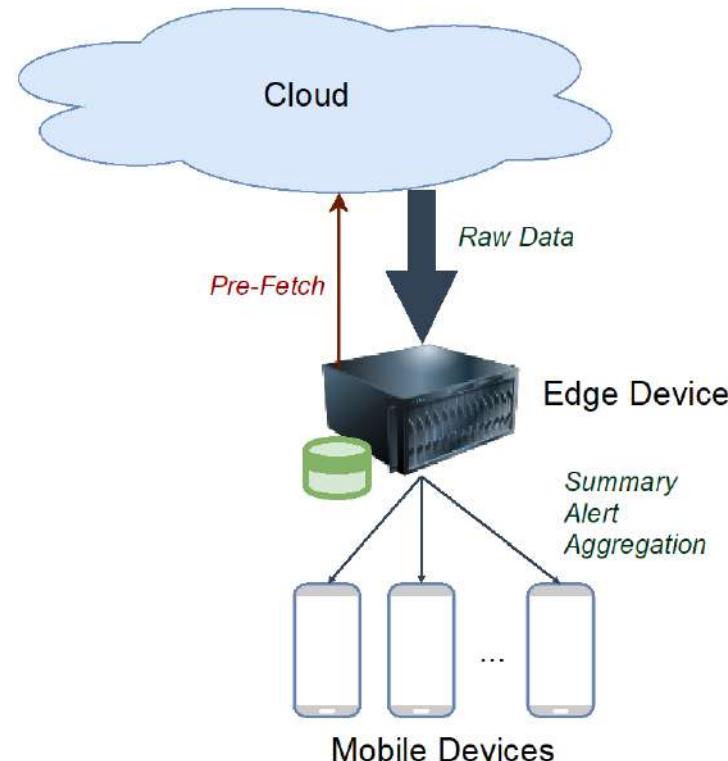
# Data Pre-Processing, Filtering, and Pre-Fetching (Cloud to Edge)

Using edge devices to

- pre-process,
- pre-fetch, or
- filter unnecessary data from streams intended for mobile devices

Goal: Mobile devices receive only the data that they need, when they need it

- reduced bandwidth
- reduced latency
- reduced cognitive load

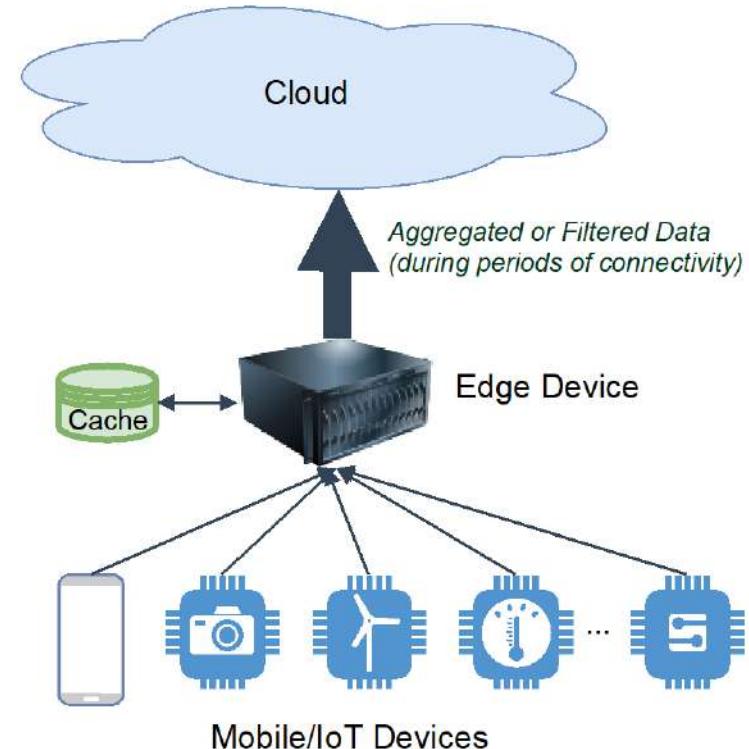


# Data Pre-Processing and Caching (Edge to Cloud)

Using edge devices to

- pre-process, or
- cache

data heading for enterprise repositories



# Field Operations

People that spend time away from their main offices or labs, such as researchers, medics, and sales personnel, can leverage portable surrogates to support their computation and data needs

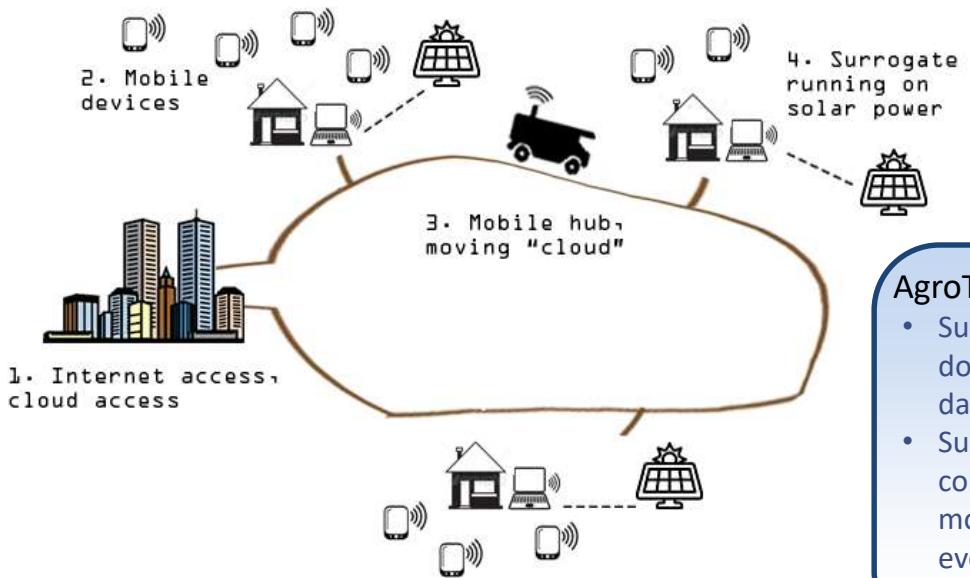


Leverages microfluidic paper-based analytical devices ( $\mu$ PADs)

PowerSense: Image Processing for Dengue Detection [7]

# Resource-Challenged Environments

Less-privileged regions characterized by limited Internet access, limited electricity and network access, and potentially low levels of literacy can leverage surrogates to obtain information to support their communities



## AgroTempus Features [8]

- Surrogates in villages download and cache data from mobile hub
- Surrogates upload field-collected data to the mobile hub which eventually syncs with the cloud



# Challenges

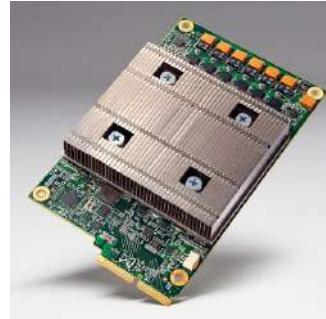
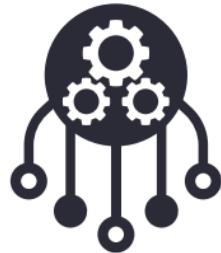
Hardware (especially in the context of Edge AI)

Privacy

Security

Data and computation allocation to edge devices  
(especially at runtime)

Resource discovery



Google Edge TPU BETA [9]



The Joneses (2009)



**Privacy** is the state of being undisturbed or alone; a desire for anonymity, the wish to protect their privacy in the unnoticed or unidentified.

# Summary

Edge Computing is about pushing applications, data and computing power to the edge of the Internet, in close proximity to mobile devices, sensors, and end users

Edge Computing via “appliances” can provide computation and data to support a wide variety of missions

I challenge you to think about use cases for Edge computing beyond IoT

- Military
- Humanitarian
- Public safety
- Public service



# References

- [1] Open Edge Computing Initiative. <http://openedgecomputing.org> (2019)
- [2] Open Automation Software. IIoT Edge Computing vs. Cloud Computing. <https://openautomationsoftware.com/blog/iiot-edge-computing-vs-cloud-computing/> (2019)
- [3] Amazon. AWS Snow Family. <https://aws.amazon.com/snow/?c=17&pt=6> (2019)
- [4] Echeverría, Sebastián, Grace A. Lewis, James Root, and Ben Bradshaw. "Cyber-foraging for improving survivability of mobile systems." In MILCOM 2015-2015 IEEE Military Communications Conference, pp. 1421-1426. IEEE, 2015.
- [5] Echeverría, Sebastián, Dan Klinedinst, Keegan Williams, and Grace A. Lewis. "Establishing trusted identities in disconnected edge environments." In 2016 IEEE/ACM Symposium on Edge Computing (SEC), pp. 51-63. IEEE, 2016.
- [6] Lewis, Grace A., Sebastián Echeverría, Dan Klinedinst, and Keegan Williams. "Secure VM migration in tactical cloudlets." In MILCOM 2017-2017 IEEE Military Communications Conference (MILCOM), pp. 388-393. IEEE, 2017.
- [7] Matthews, Jerrid, et al. "PowerSense: power aware dengue diagnosis on mobile phones." Proceedings of the First ACM Workshop on Mobile Systems, Applications, and Services for Healthcare. ACM, 2011.
- [8] Brion, Reuel. Demonstrator for a Cyber-Foraging System to Support Agricultural Knowledge Exchange in Resource-challenged Environments. Masters Thesis. VU University Amsterdam. 2015.
- [9] Google. Edge TPU <sup>BETA</sup>. <https://cloud.google.com/edge-tpu/> (2019)

# Contact Information

## Grace A. Lewis

Tactical and AI-Enabled Systems (TAS) Initiative  
Software Solutions Division (SSD)

Software Engineering Institute  
4500 Fifth Avenue  
Pittsburgh, PA 15213-2612  
USA

Phone: +1 412-268-5851  
Email: [glewis@sei.cmu.edu](mailto:glewis@sei.cmu.edu)  
WWW: <http://www.sei.cmu.edu/staff/glewis>





# Fog Computing and its Ecosystem

Ramin Elahi, Adjunct Faculty  
UC Santa Cruz Silicon Valley

# SNIA Legal Notice

- 
- The material contained in this tutorial is copyrighted by the SNIA unless otherwise noted.
  - Member companies and individual members may use this material in presentations and literature under the following conditions:
    - Any slide or slides used must be reproduced in their entirety without modification
    - The SNIA must be acknowledged as the source of any material used in the body of any document containing material from these presentations.
  - This presentation is a project of the SNIA Education Committee.
  - Neither the author nor the presenter is an attorney and nothing in this presentation is intended to be, or should be construed as legal advice or an opinion of counsel. If you need legal advice or a legal opinion please contact your attorney.
  - The information presented herein represents the author's personal opinion and current understanding of the relevant issues involved. The author, the presenter, and the SNIA do not assume any responsibility or liability for damages arising out of any reliance on or use of this information.

**NO WARRANTIES, EXPRESS OR IMPLIED. USE AT YOUR OWN RISK.**

# Abstract

## ► Fog Computing and its Ecosystem

In relation to “cloud computing”, it is bringing the computing & services to the edge of the network. Fog provides data, compute, storage, and application services to end-users. Fog Computing is also known as Edge Computing within the industry. The distinguishing Fog characteristics are its proximity to end-users, its dense geographical distribution, and its support for mobility. Services are hosted at the network edge or even end devices such as set-top-boxes or access points. Thus, it can alleviate issues the IoT (Internet of Things) is expected to produce such as reducing service latency, and improving QoS, resulting in superior user-experience. Fog Computing supports emerging Internet of Everything (IoE) applications that demand real-time/predictable latency (industrial automation, transportation, networks of sensors and actuators). Thanks to its wide geographical distribution the Fog paradigm is well positioned for real time big data and real time analytics. Fog supports densely distributed data collection points, hence adding a fourth axis to the often mentioned Big Data dimensions (volume, variety, and velocity).

# Agenda

- The first wave of Cloud computing
- Challenges of Cloud Computing
- Transitioning from Cloud to IoT & IoE to Fog Computing
- General awareness on IoT, IoE & Fog Computing
- Introducing Fog Computing as an Extension of Cloud
- Comparing & Contrast of Fog and Cloud
- What's Next?
- Q&A

# Today's Major Industry Trends

Flash

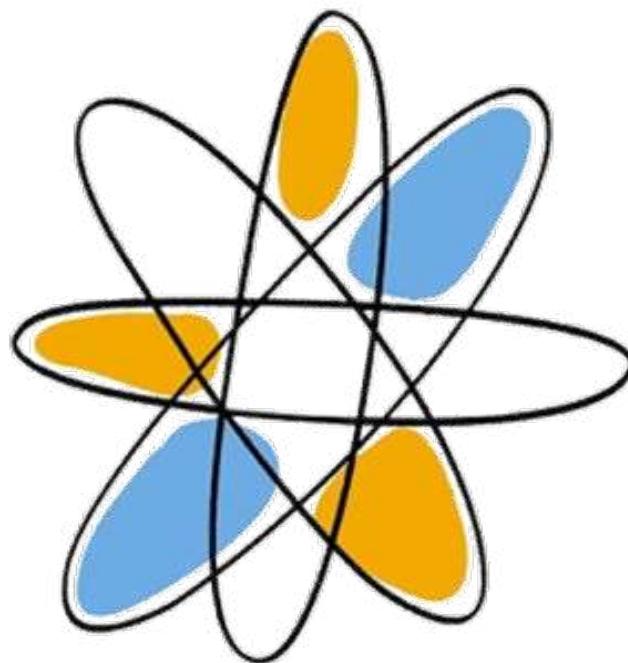
Cloud

Software-defined  
data centers

Data Analytics

Mobility

Big Data

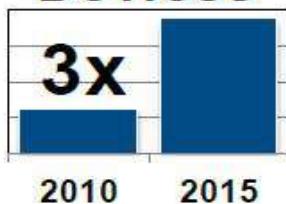


*Industry*

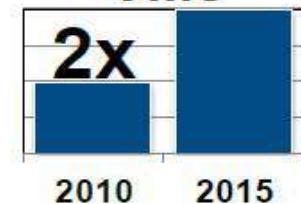
# The Rapid Growth in Data Centers

Datacenters are the cornerstone of business...

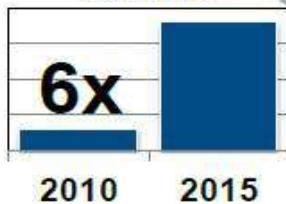
**Devices**



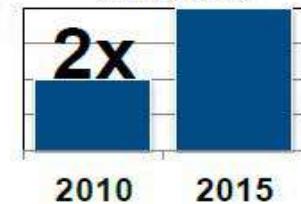
**VMs**



**Data**



**Users**

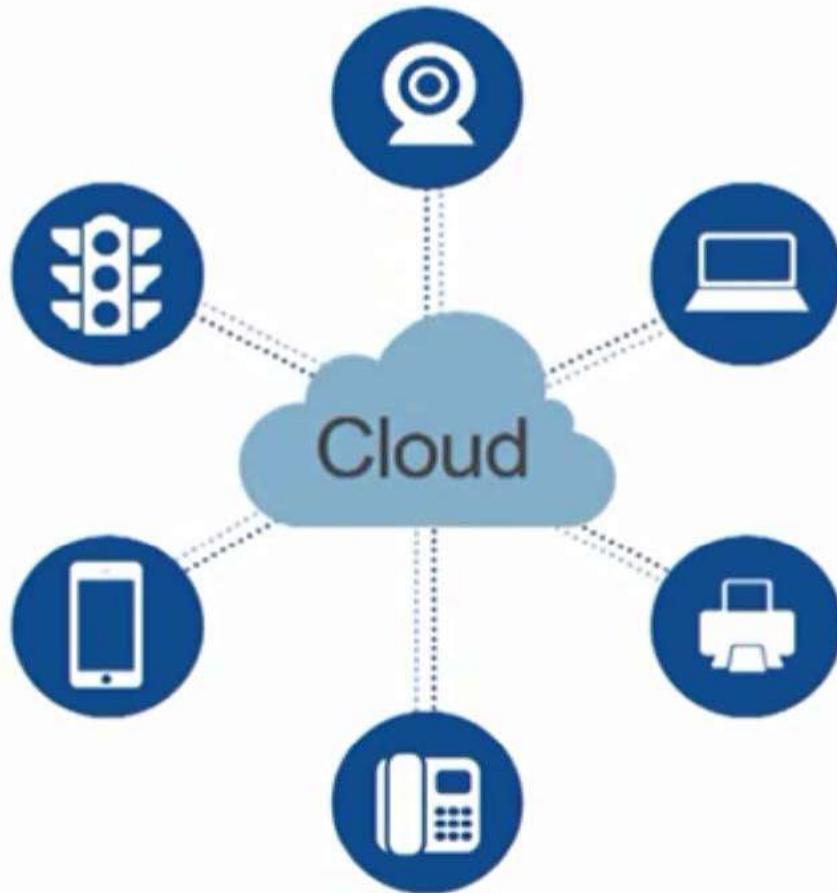


...managing information assets is a key datacenter task

# It's all in the Cloud!



# The First Wave of Cloud



# Today's Cloud and its Characteristics



## Common Characteristics:

**Massive Scale**

**Resilient Computing**

**Homogeneity**

**Geographic Distribution**

**Virtualization**

**Service Orientation**

**Low Cost Software**

**Advanced Security**

## Essential Characteristics:

**On Demand Self-Service**

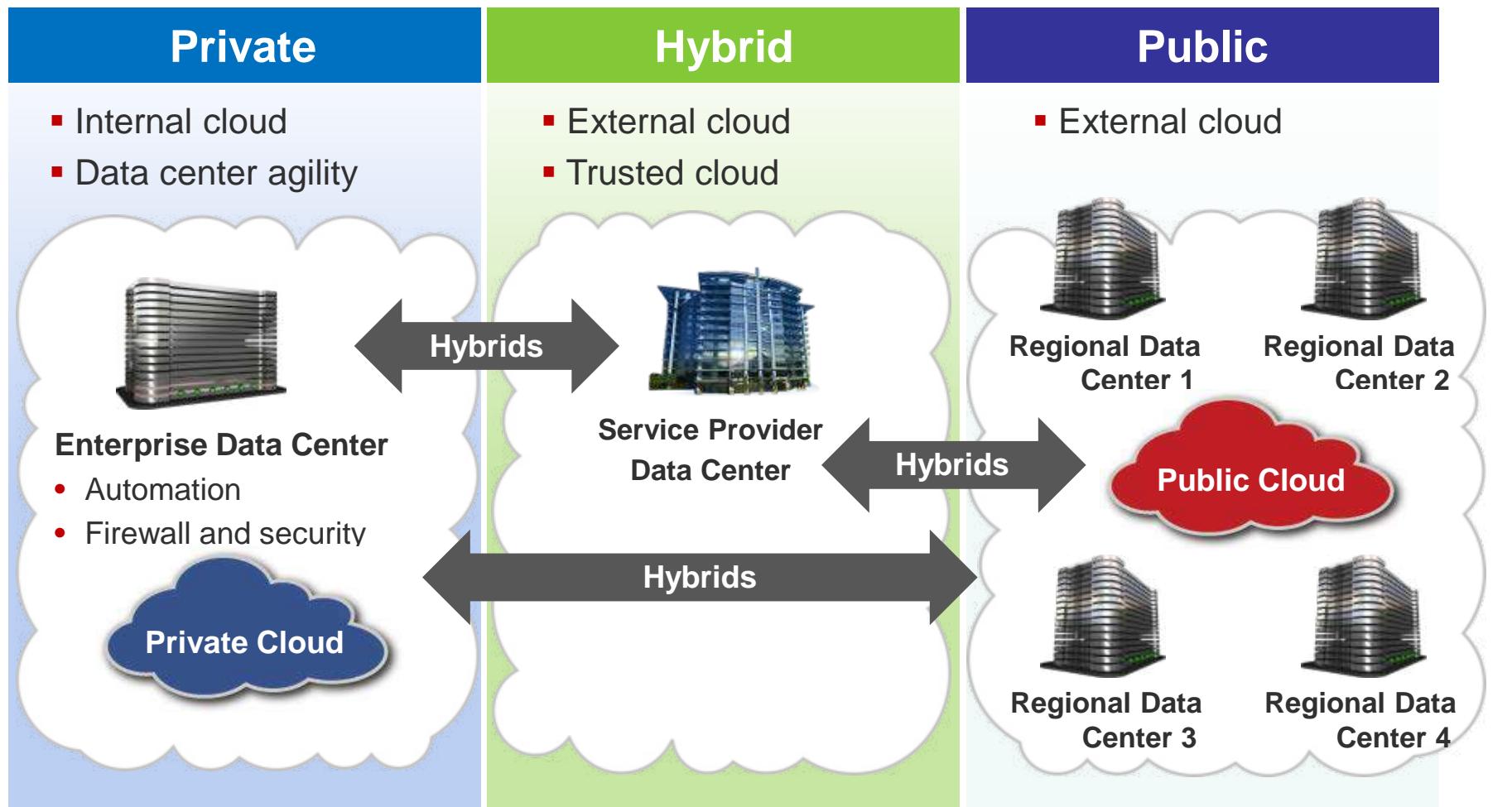
**Broad Network Access**

**Rapid Elasticity**

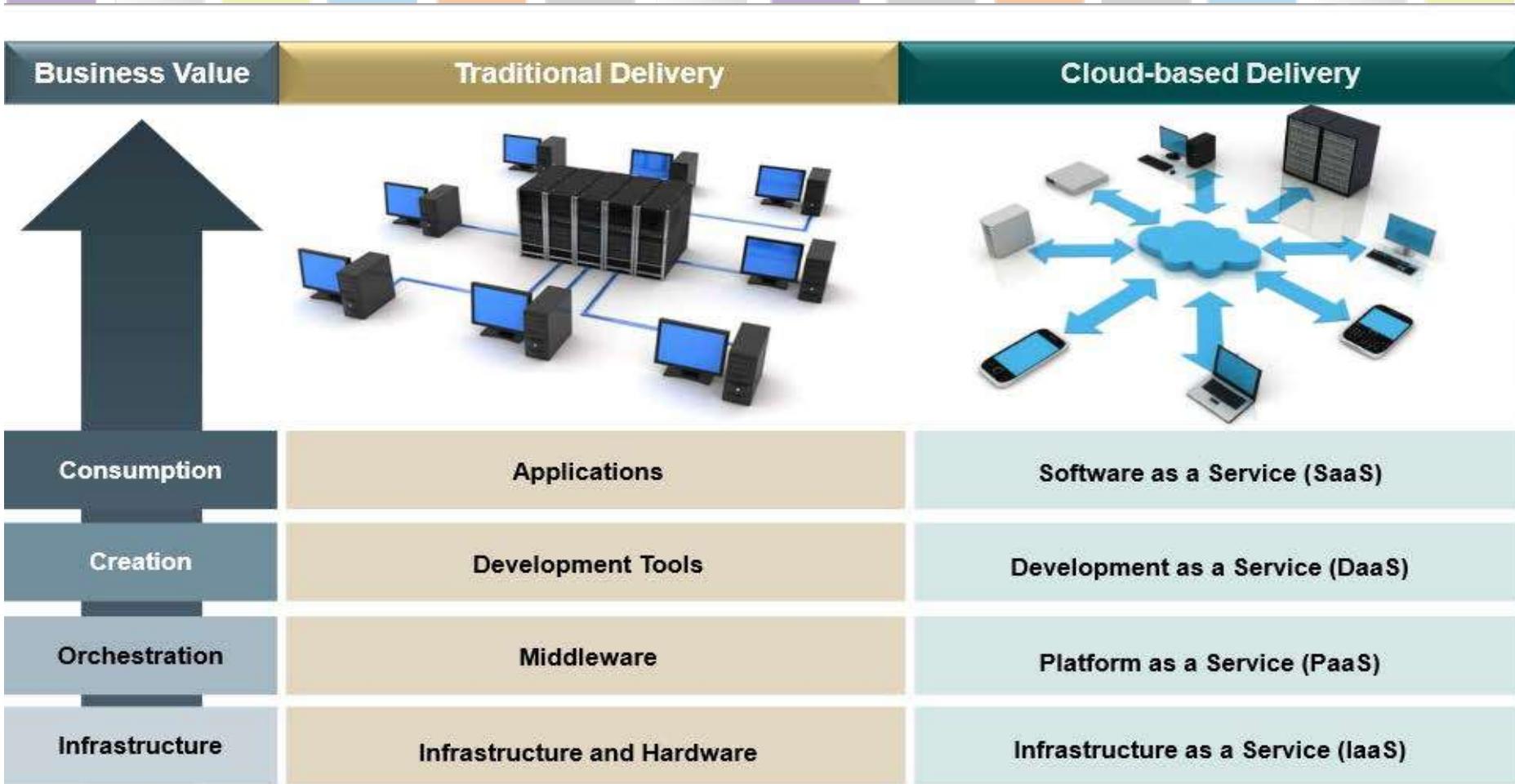
**Resource Pooling**

**Measured Service**

# Types of Cloud

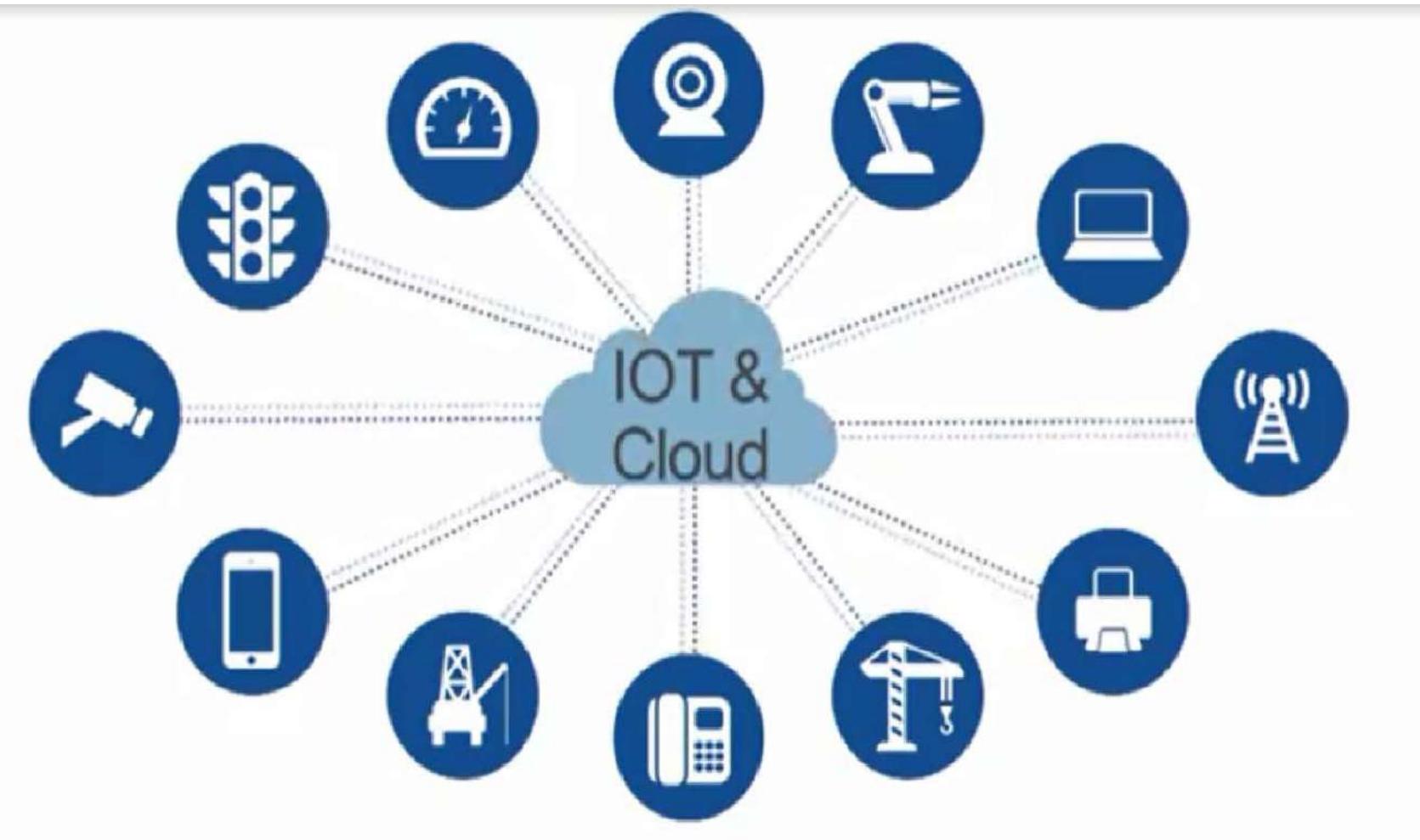


# Traditional IT Delivery Translated to Cloud

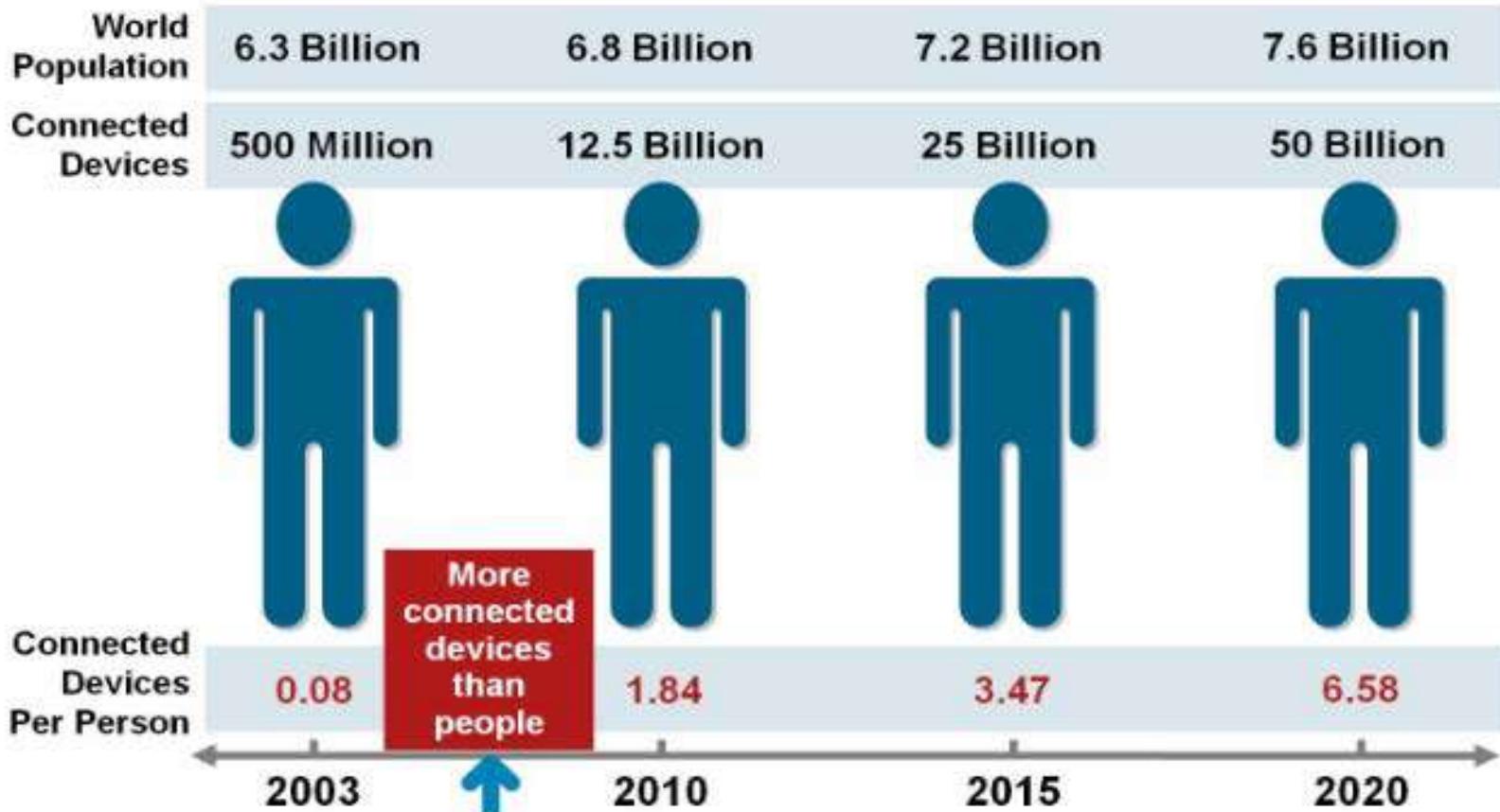


Source: R Wang and Insider Associates; A Software Insider's Point of View Understanding The Many Flavors of Cloud Computing and SaaS, R "Ray" Wang, Phil Waine, Michael Cote, and James Governor; Forrester Report; [Grail Research Analysis](#)

# The Latest Wave of Cloud & IoT Adoption



# The “Birth” of IoT: Circa 2008 & 2009



Source: *The Internet of Things*, by Dave Evans, Cisco IBSG 2011.

# Now comes the IoE!

## Data

Leveraging data into more useful information for decision making



## Things

Physical devices and objects connected to the Internet and each other for intelligent decision making, often called

**Internet of Things (IoT)**



## People

Connecting people in more relevant, valuable ways



## Internet of Everything

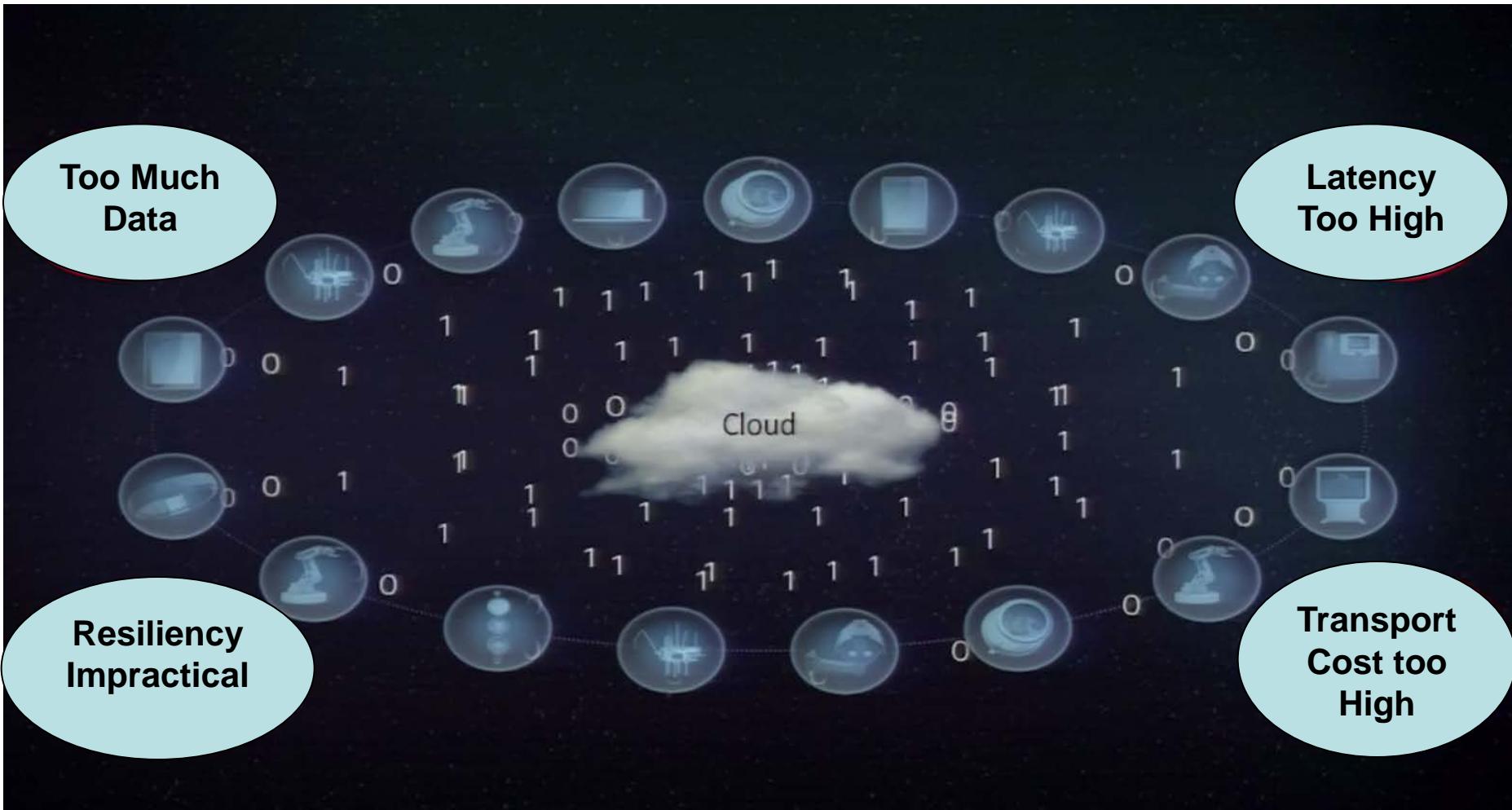


## Process

Delivering the right information to the right person (or machine) at the right time

# The Challenges of Cloud Computing

As more and more nodes are added to the network....



# Digitizing Drives Data & Infrastructure to the Network Edge

## INCREASING DIGITIZATION

2014—3.4 ZB    2019—10.4 ZB

A shift in Storage & Compute Architecture may be in order?



Source: Cisco Global Cloud Index Forecast, 2014-2019, Global IoT Study.

# Introducing Fog Computing

Also Known As Edge Computing Throughout Industry

A paradigm that extends Cloud computing and services to the edge of the network. Similar to Cloud, Fog provides data, compute, storage, and application services to end-users.



# Characteristics of Fog Computing

- A paradigm that extends Cloud computing to the edge of the network
- Low latency & location awareness
- send the right data to the cloud for big data analytics and storage
- Wide-spread geographical distribution
- Strong presence of streaming and real time applications
- Handle an unprecedented volume, variety, and velocity of data
- Heterogeneity of connected objects
- Fog applications to communicate directly with mobile devices
- Predominant role of wireless access

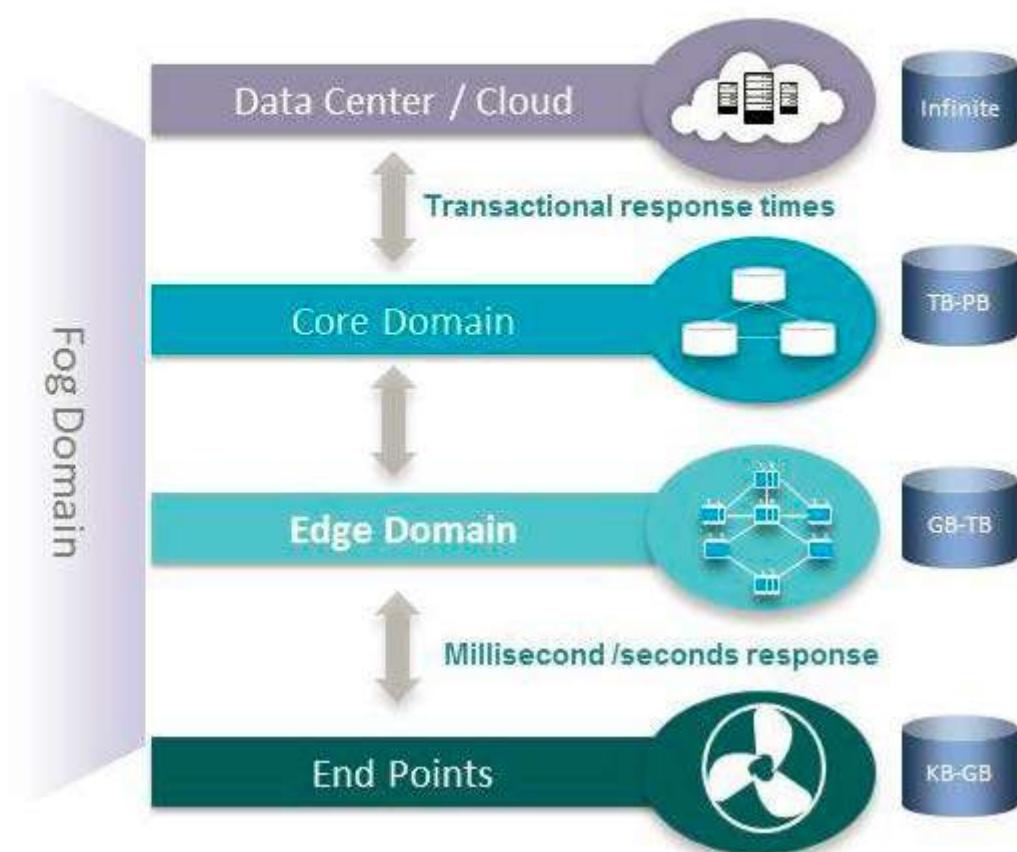
# More Simplified View of Fog Architecture

Fog computing is...

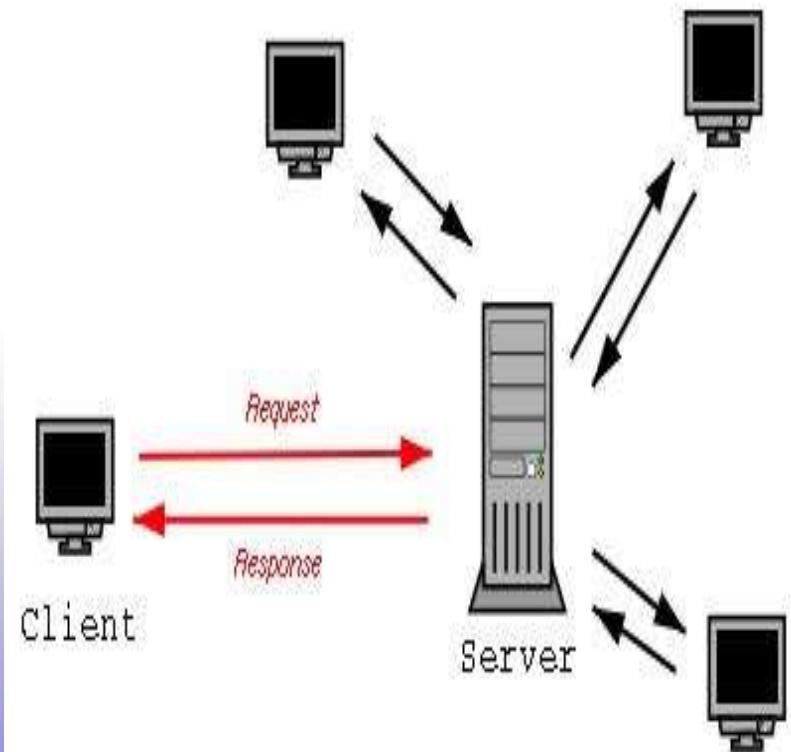
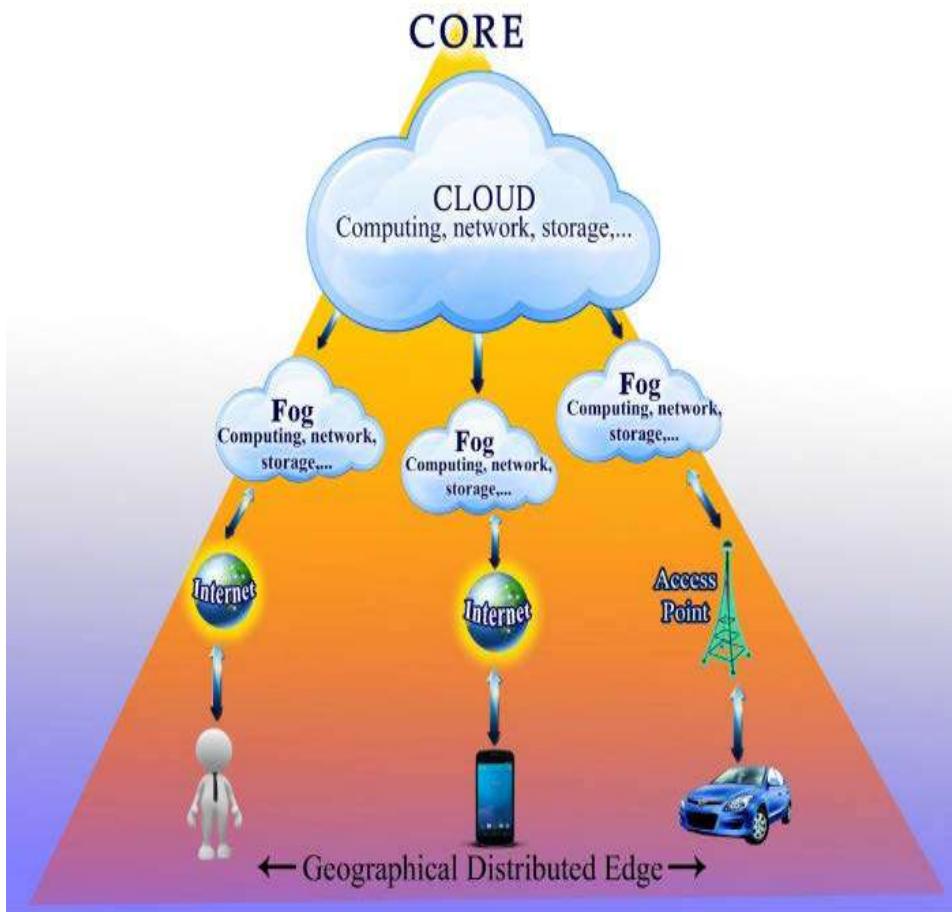
A system-level architecture  
to extend

*Compute  
Network  
Storage*

Capability of Cloud to the  
edge of the IoT network

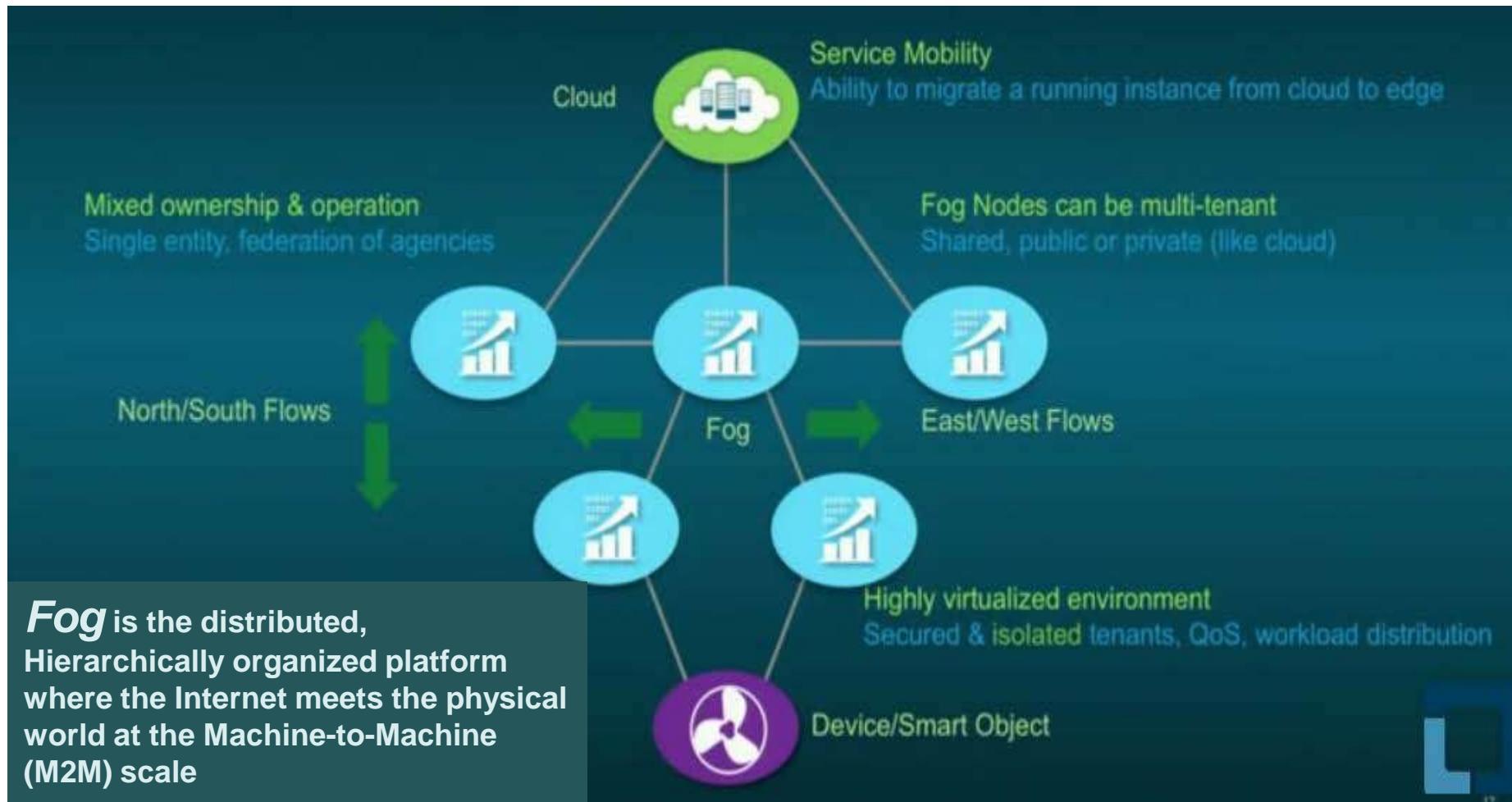


# Fog Computing Vs. Client –Server Model

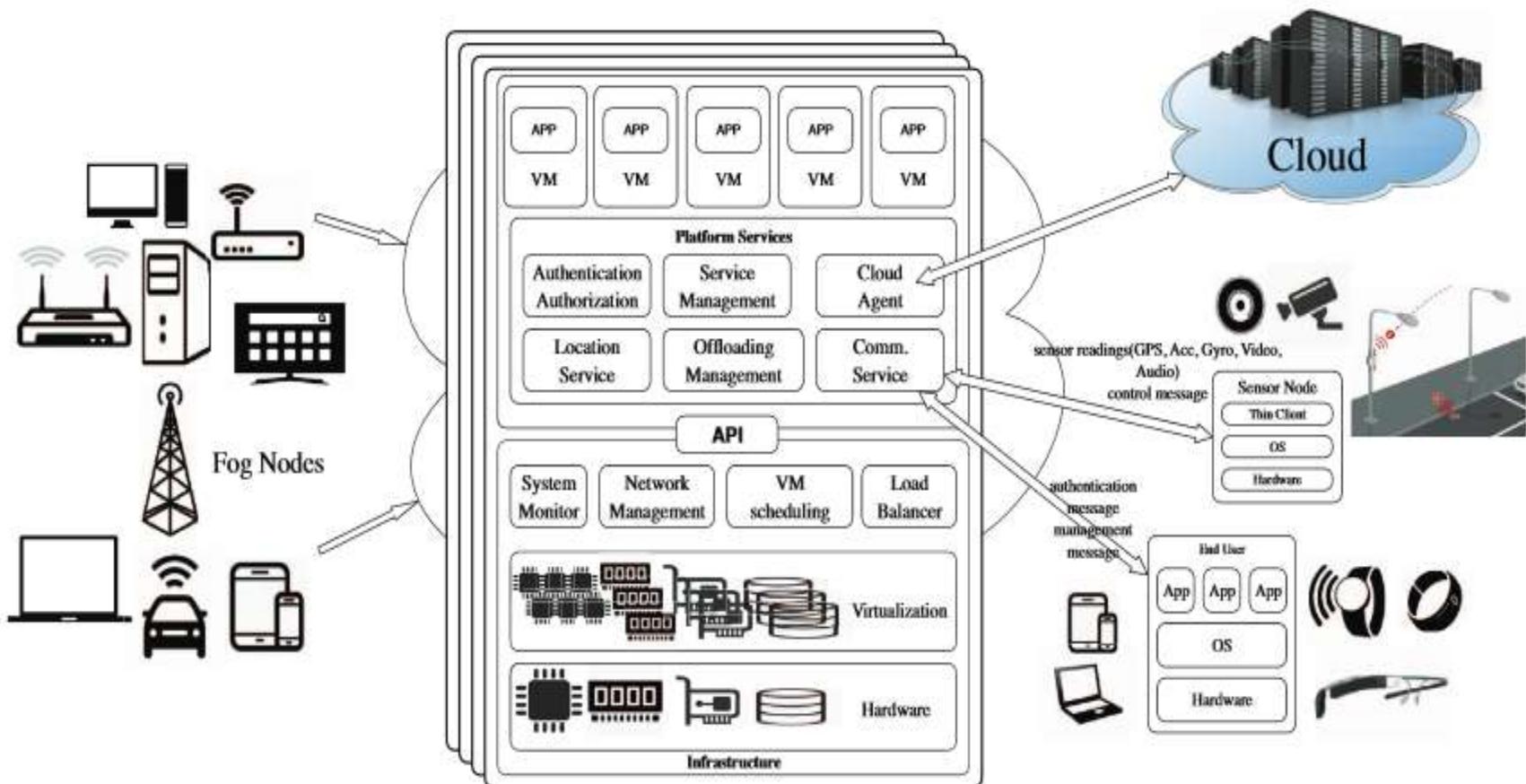


Source: Fog Computing Made Easy with the Help of Citrix & Billboard Manager, Journal of Computer Applications Vol. 121, No 7, Jul 2015

# Hierarchy of Fog Computing Architecture



# The Components for Fog Computing Platform



# Pushing Intelligence Up Toward the Cloud



## Cloud Challenges

- Critical Latency Req.
- Data Rich Mobility
- Geographic Diversity
- Network Bandwidth limit.
- Reliability/Robustness
- Analytics Challenges
- User Data/Geo. Privacy

## How Fog can Help

- + Fewer Network hops
- + Data locality & Local Caches
- + Intelligence localized as appropriate
- + Local processing / less core Net. Load
- + Fast Failover; local resp. in Emergency
- + Analytics & Storage at the Right Tier
- + Fog can Aggregate User Data

# Pushing Intelligence Down Toward the Endpoints



## Intelligent Endpoint Challenges

- Endpoint Physical Constraints
  - Energy/Power
  - Space
  - Environment (temp/, humidity & Vib.)
- Endpoint Functional Constraints
  - Processor throughput
  - Storage capacity
  - Reliability
  - Modularity
- Endpoint Security Constraints

## How Fog Can Help

- + Fog nodes can access more energy
- + Fog Nodes can be physically larger
- + Better cooling systems in many Fog Nodes
- + Terabytes > PB storage cap.
- + Fog capabilities can be more redundant
- + Modules can be added as needed
- + Fog has better physical/network security

# Data “Gravity” – IoT Objects generates 2EB/Day



46 million smart meters in the U.S alone 1.1 billion data points (.5TB) / day



A single consumer packaged good manufacturing machine generates 13B data samples/day



A large offshore field produces 0.75TB data/week



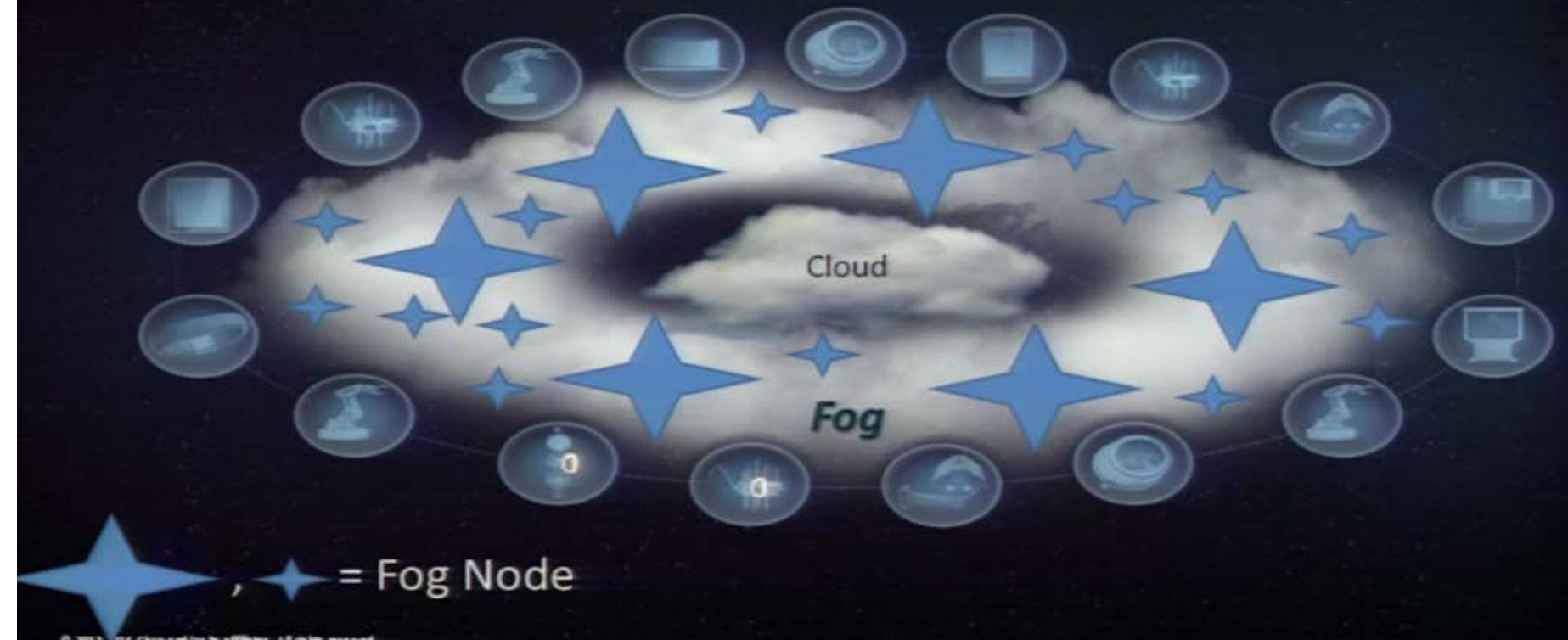
A jet engine produces 20TB flight data/hour

90% of the world's data created in last 2 years

Source: From Cloud to Fog Computing and IoT | LinuxCon + CloudOpen North America 2014

# How Fog Computing can help?

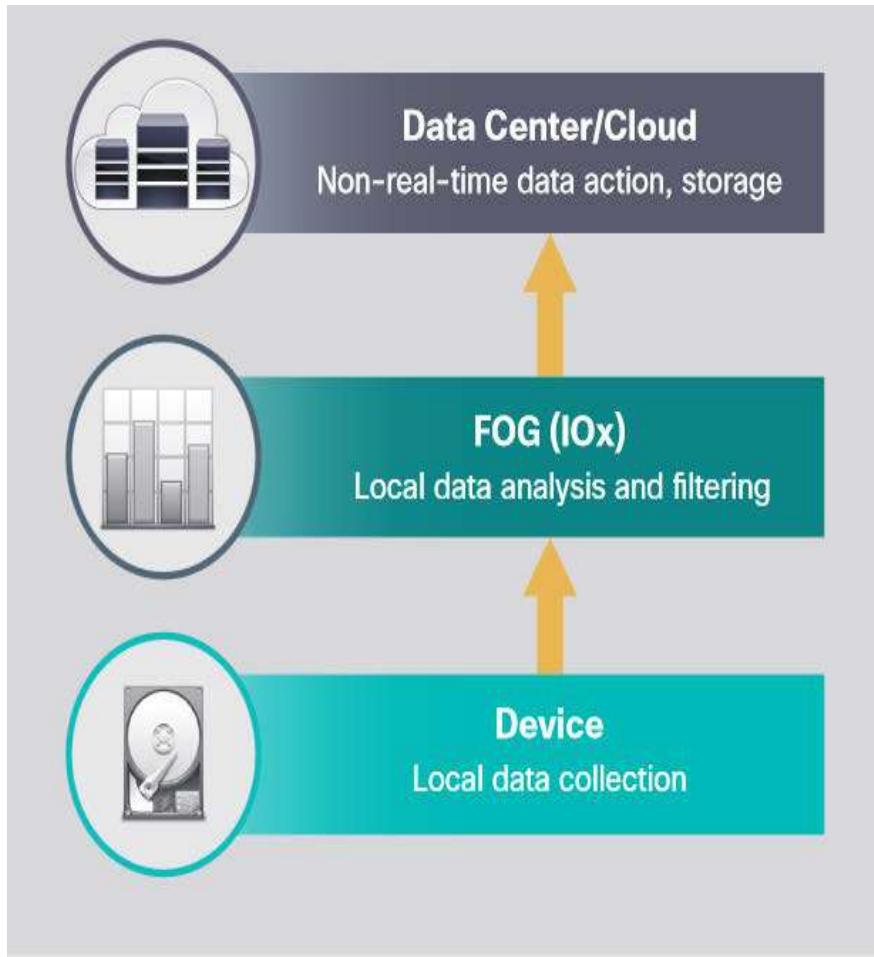
Fog Puts Intelligence Closer to the Data Source  
Fog Nodes Implement Local Processing, Storage, and Networking



© 2013-2014 Cisco and/or its affiliates. All rights reserved.

# IoT with Fog Computing

## At-a-Glance

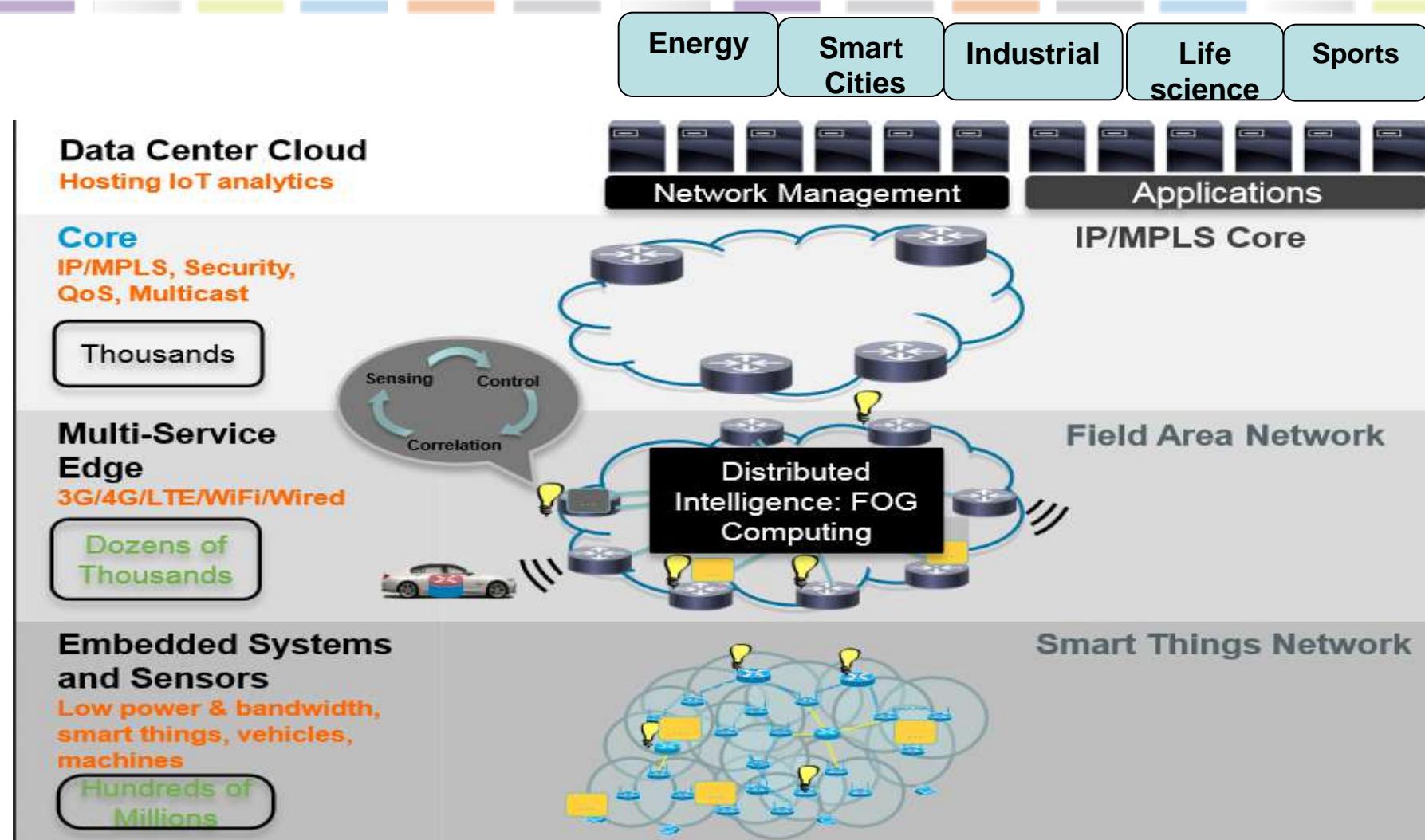


## What You Can Do:

- **Analyze and act on data right at the network edge**
- **Use bandwidth and storage capacity more efficiently sending only relevant information to the cloud**
- **Connect any protocol or device through an open platform**

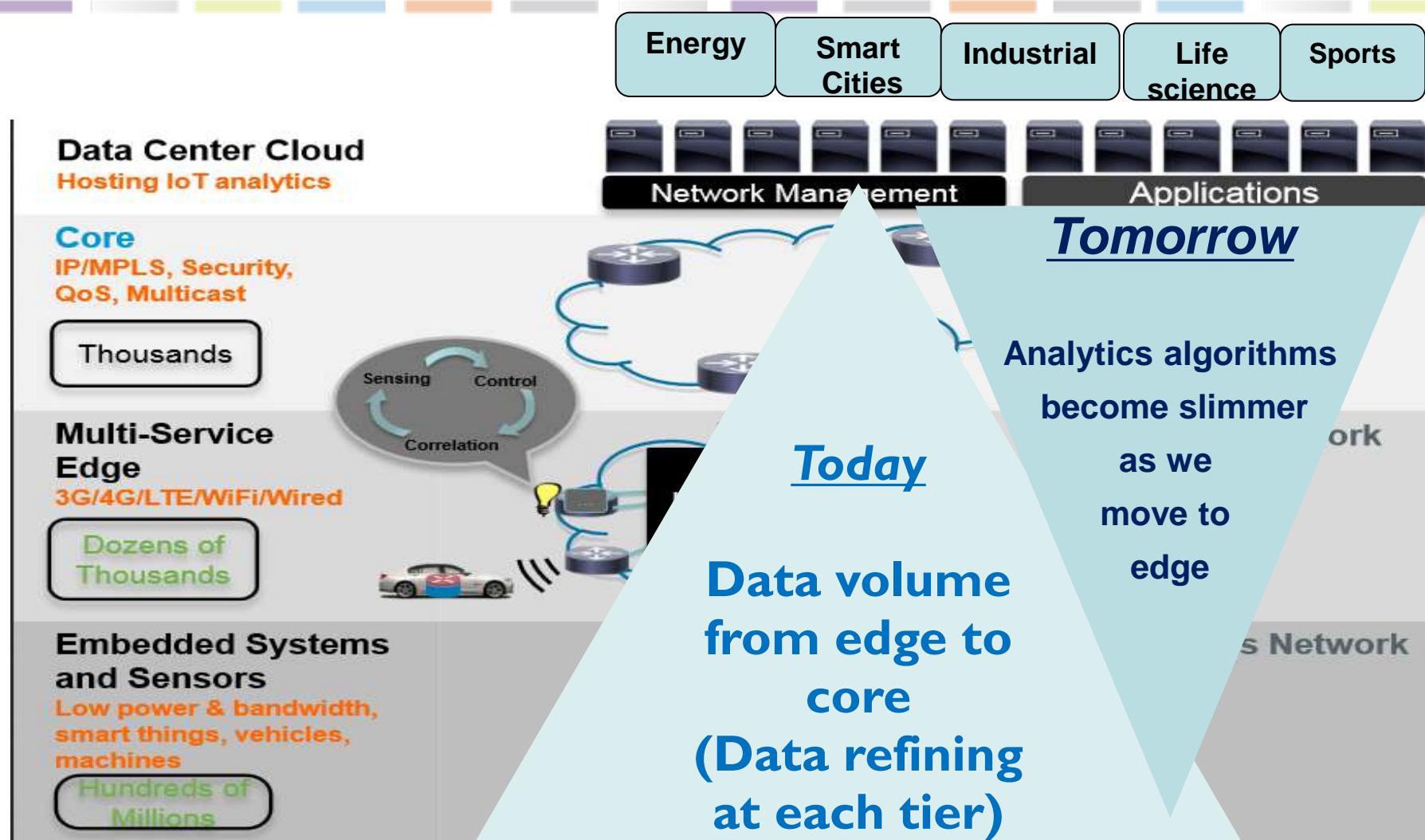
# Emerging Architecture for Data Analytics Processing

1/2



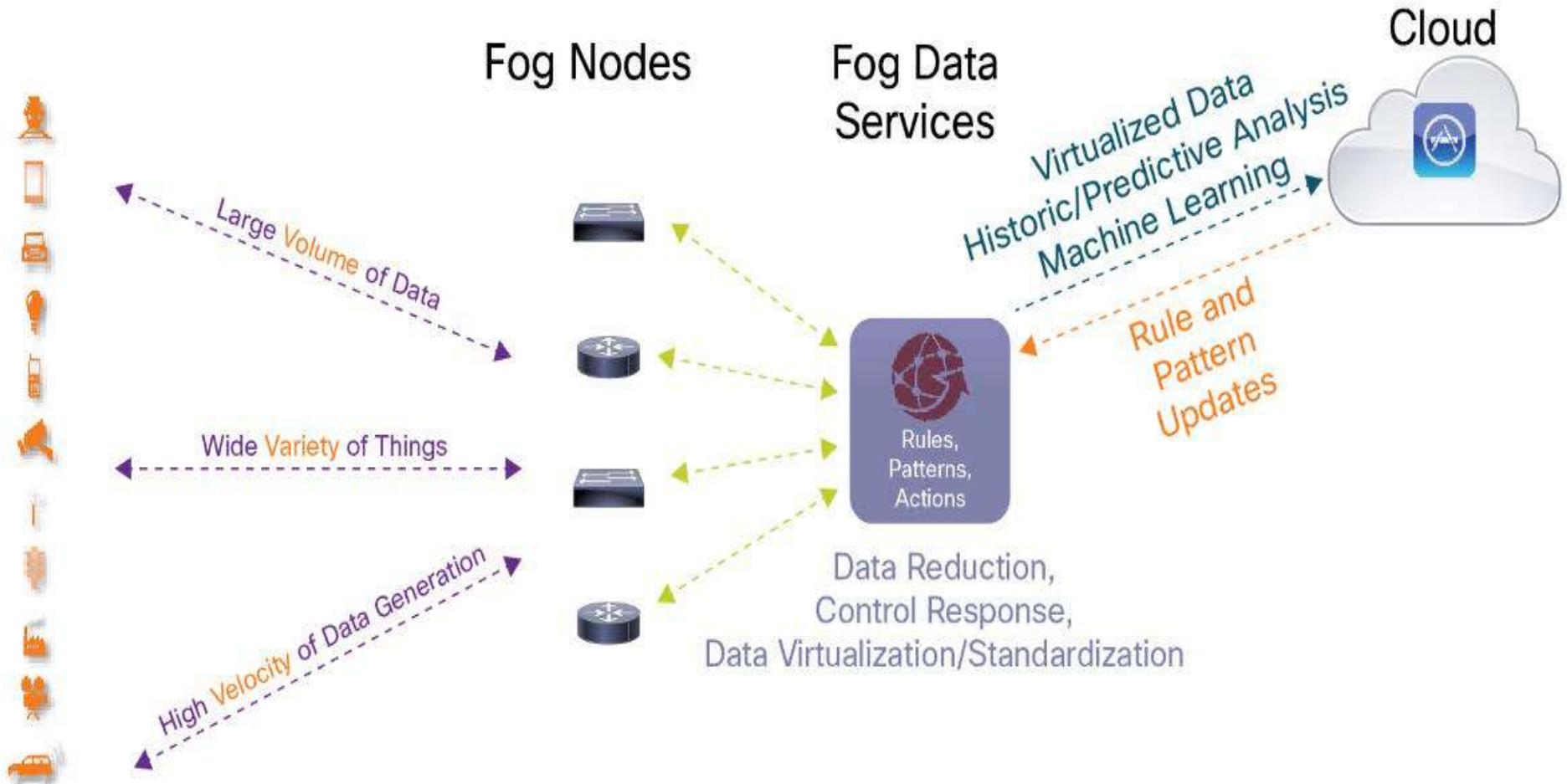
# Emerging Architecture for Data Analytics Processing

2/2



# Movement of Data from Fog to Cloud

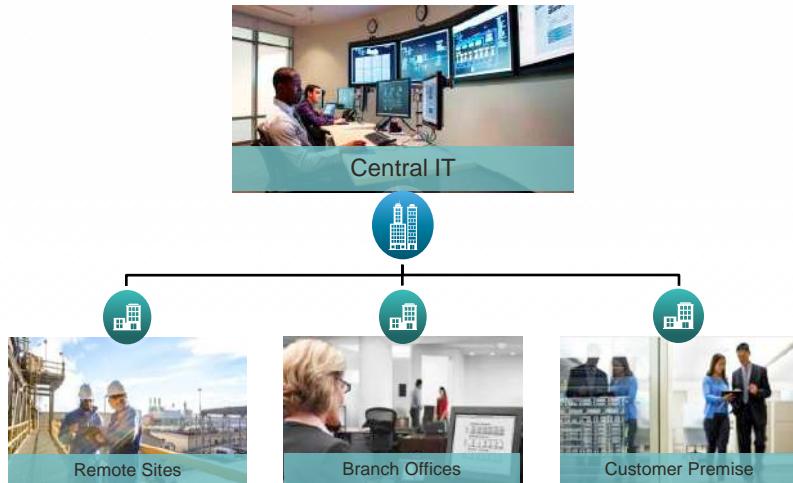
Fog Data Services Coordinate the Movement of Data from Fog to Cloud



# Edge Scaling Computing & Customer Needs



## Computing Near the Source of Demand



## Enabling Small Scale IT



### Customer Needs

- “No Assembly Required” total computing solution
- Simplified systems management
- Easy scalability from 1~15 servers
- Low power / cooling footprint

### Customer Needs

- Computing proximity for IoT / Fog, Remote Site, Branch
- Comprehensive remote management at global scale

IDC estimates that the amount of data analyzed on devices that are physically close to the Internet of Things is approaching 40 percent *IDC Press Release Dec 2014*

# Compare & Contrast

1/2



| Requirements                         | Cloud Computing          | Fog Computing                      |
|--------------------------------------|--------------------------|------------------------------------|
| Latency                              | High                     | Low                                |
| Delay Jitter                         | High                     | Very low                           |
| Location of Servers                  | Within Internet          | At the edge close to Nodes         |
| Distance between the client & server | Multiple hops            | One hop                            |
| Security                             | Varies amongst providers | Can be more defined and customized |
| Attack on Data-in-Flight             | High probability         | Limited with less probability      |
| Location awareness                   | No                       | Yes                                |

Source: Parts taken from Fog Computing, J.HariPriyanka, April 2015, <http://www.slideshare.net/haripriyanka58/fog-computing-47425209>

# Compare & Contrast

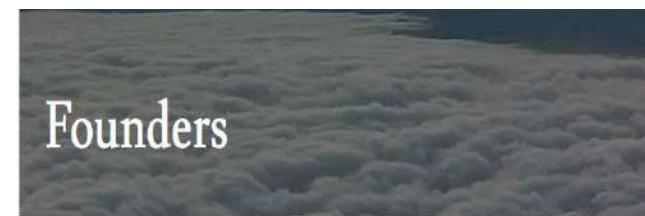
2/2



| Requirements                   | Cloud Computing                                    | Fog Computing |
|--------------------------------|--|---------------|
| Geo. Distribution              | Centralized  | Distributed   |
| No. of Server Noes             | Few  | Very large    |
| Support for Mobility           | Limited  | Supported     |
| Real Time Interactions         | Supported but may be difficult to achieve & Costly | Supported     |
| Type of last mile connectivity | Leased line  | wireless      |

Source: Parts taken from Fog Computing, J.HariPriyanka, April 2015, <http://www.slideshare.net/haripriyanka58/fog-computing-47425209>

# A Major Milestone in Fog Computing



# Open Fog Consortium

OUR MISSION: TO DRIVE INDUSTRY AND ACADEMIC LEADERSHIP IN FOG COMPUTING ARCHITECTURE, TESTBED DEVELOPMENT, AND A VARIETY OF INTEROPERABILITY AND COMPOSABILITY DELIVERABLES THAT SEAMLESSLY LEVERAGE CLOUD AND EDGE ARCHITECTURES TO ENABLE END-TO-END IOT SCENARIOS.

# What's Next for Fog Computing?



- Identify use cases where Fog provides advantages
- Refine our views on Fog architecture
- Define an application architecture that facilitates interoperability & application migration
- Experiment with Fog APIs
- Understand how fog can help out businesses
- Doing store & compute at the edge does not undermine the importance of the center. In fact, the Data Center needs to be a stronger nucleus for expanding computing

# Attribution & Feedback



The SNIA Education Committee thanks the following Individuals for their contributions to this Tutorial.

## Authorship History

Name/Date of Original Author here:

Ramin Elahi/ November 2015

## Additional Contributors

- Chuck Byers, Cisco Systems
- Rethinking Archiving: Exploring the path to improved IT efficiency, Marshall Amaldas & Brad Nisbet, IDC, SNIA Education
- Research at Cisco Fog Computing, Ecosystem, Architecture and Applications
- Cisco Global Cloud Index Forecast, 2014-2019, Global IoT Study
- NetApp ACI Training on FlexPod
- FOG COMPUTING, J.HariPriyanka, April 2015
- R Wang and Insider Associates Forrester Report and Grail Research Analysis
- Fog Computing Made Easy, International Journal of Computer Applications vol 121, No 7, Jul 2015
- IDC Reveals Worldwide Internet of Things Predictions for 2015

*Please send any questions or comments regarding this SNIA Tutorial to [tracktutorials@snia.org](mailto:tracktutorials@snia.org)*

# Outline

---

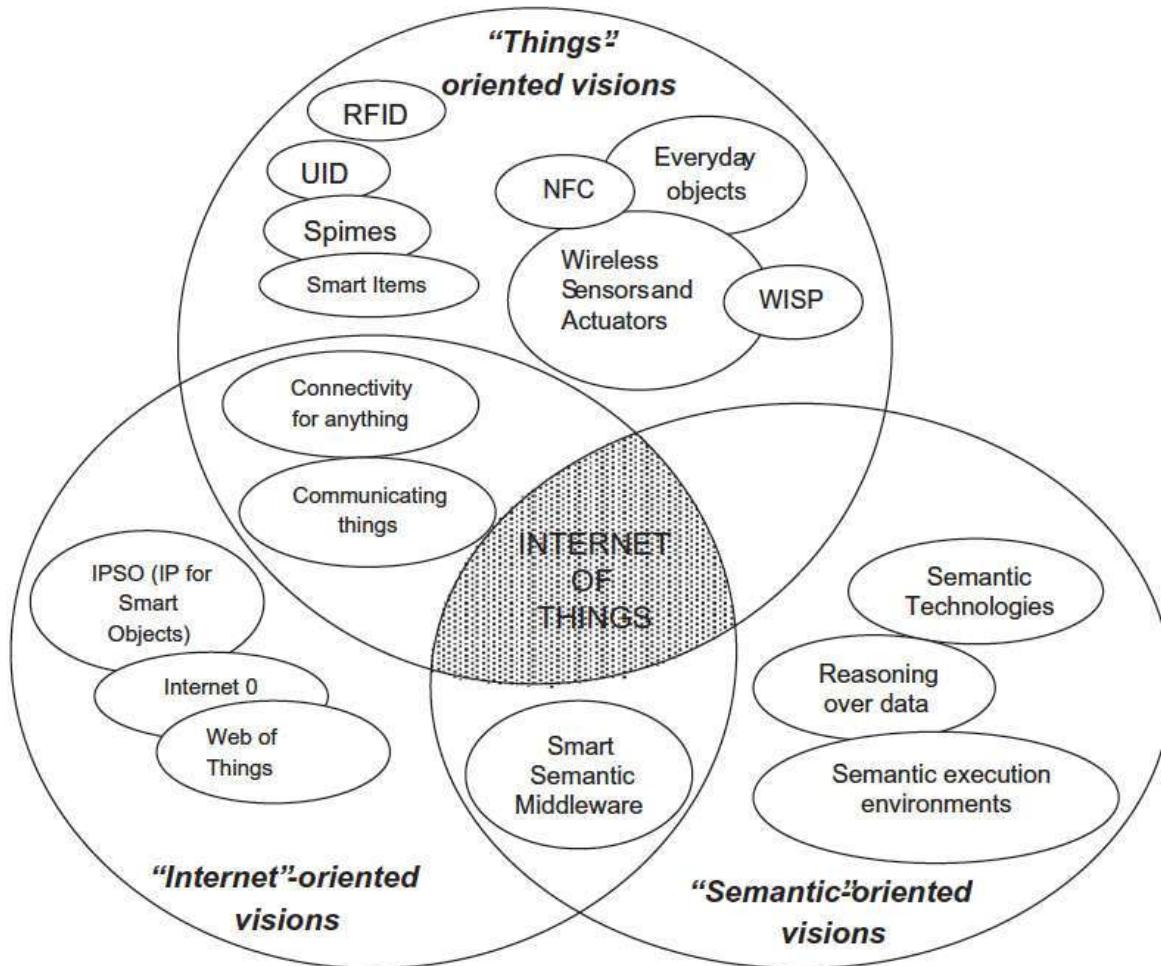
- Introduction to IoT
- Enabling technologies
- Open problems and future challenges
- Applications

# What is IoT?



- A phenomenon which connects a variety of ***things***
  - Everything that has the ability to communicate

# Connection of Multiple Visions



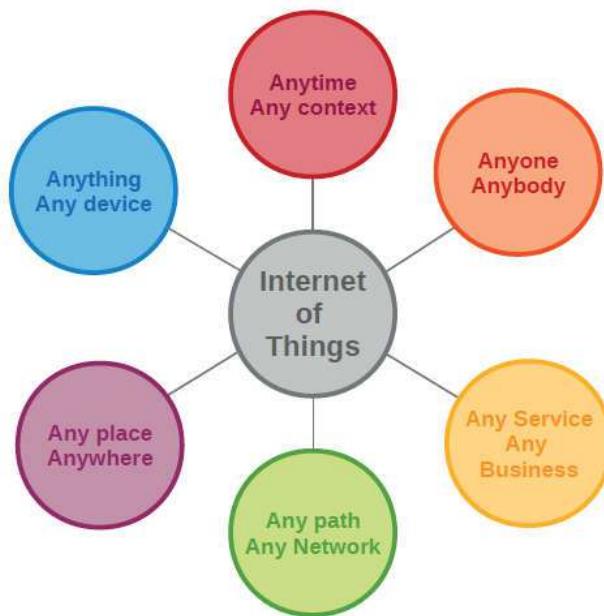
Source: Atzori et al. 2010



# IoT Definitions

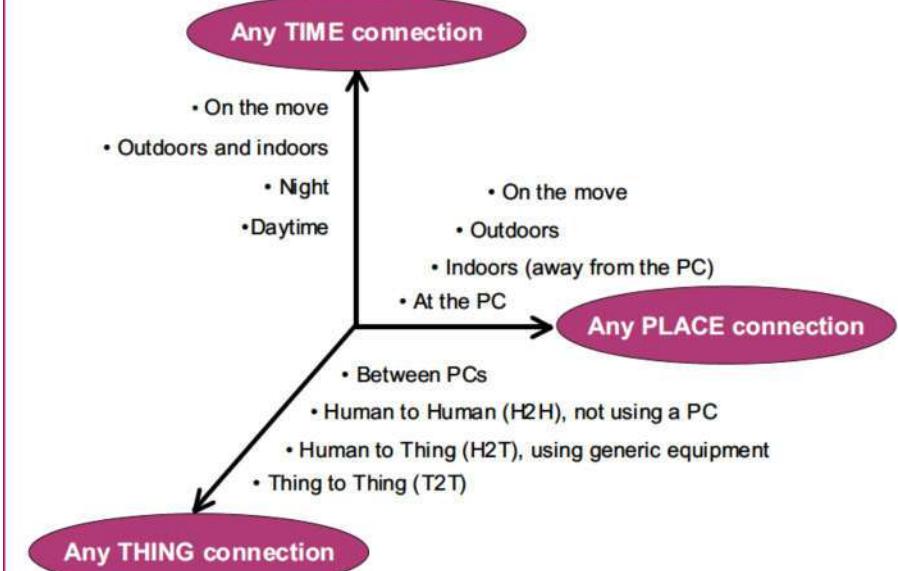
- The Internet of Things, also called The Internet of Objects, refers to a wireless network between objects, usually the network will be wireless and self-configuring, such as household appliances. (**Wikipedia**)
- The term "Internet of Things" has come to describe a number of technologies and research disciplines that enable the Internet to reach out into the real world of physical objects. (**IoT 2008**)
- “Things having identities and virtual personalities operating in smart spaces using intelligent interfaces to connect and communicate within social, environmental, and user contexts”. (**IoT in 2020**)

# Any-X Point of View



Source: Perera et al. 2014

Figure 1 – A new dimension



Source: ITU adapted from Nomura Research Institute

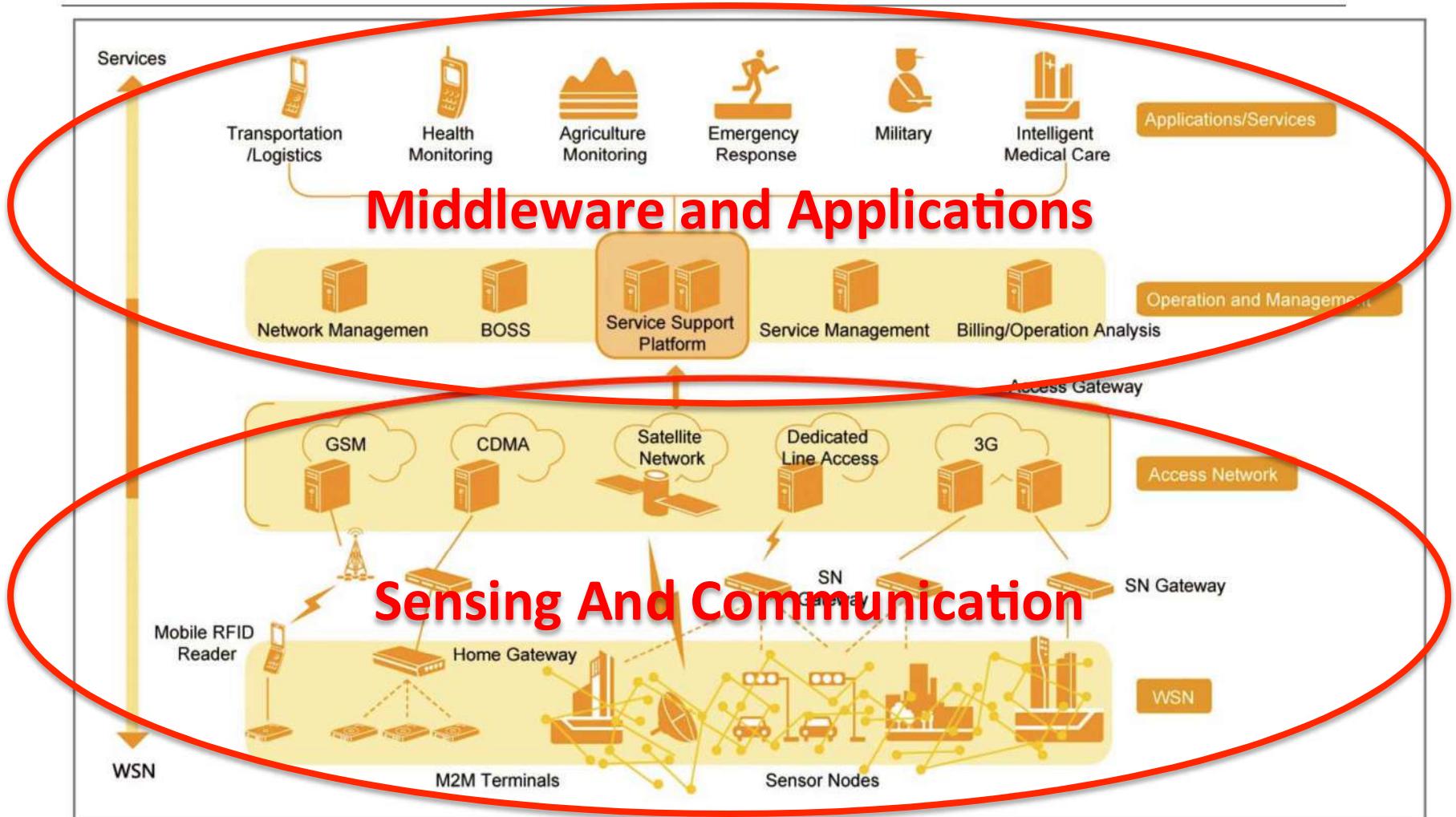
- The Internet of Things allows people and things to be connected Anytime, Anyplace, with Anything and Anyone, ideally using Any path/network and Any service.

# Characteristics of IoT

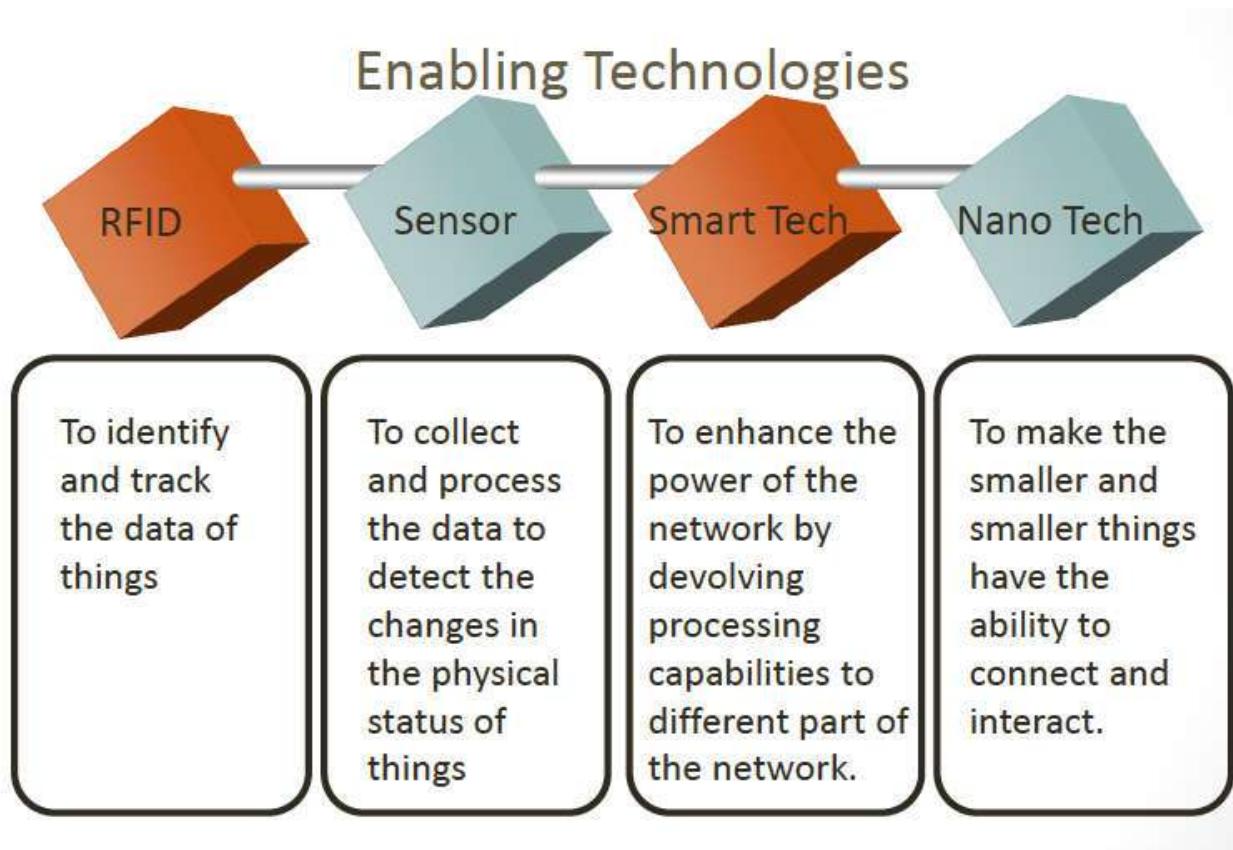
---

1. Intelligence
  - Knowledge extraction from the generated data
2. Architecture
  - A hybrid architecture supporting many others
3. Complex system
  - A diverse set of dynamically changing objects
4. Size considerations
  - Scalability
5. Time considerations
  - Billions of parallel and simultaneous events
6. Space considerations
  - Localization
7. Everything-as-a-service
  - Consuming resources as a service

# IoT Layered Architecture



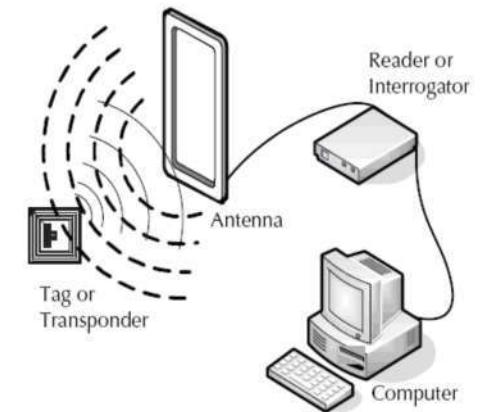
# Networking and Communication



- RFID to smallest enabling technologies, such as chips, etc.

# RFIDs

- The reduction in terms of size, weight, energy consumption, and cost of the radio takes us to a new era
  - This allows us to integrate radios in almost all objects and thus, to add the world “anything” to the above vision which leads to the IoT concept
- Composed of one or more readers and tags
- RFID tag is a small microchip attached to an antenna
- Can be seen as one of the main, smallest components of IoT, that collects data



# Wireless Technologies

---

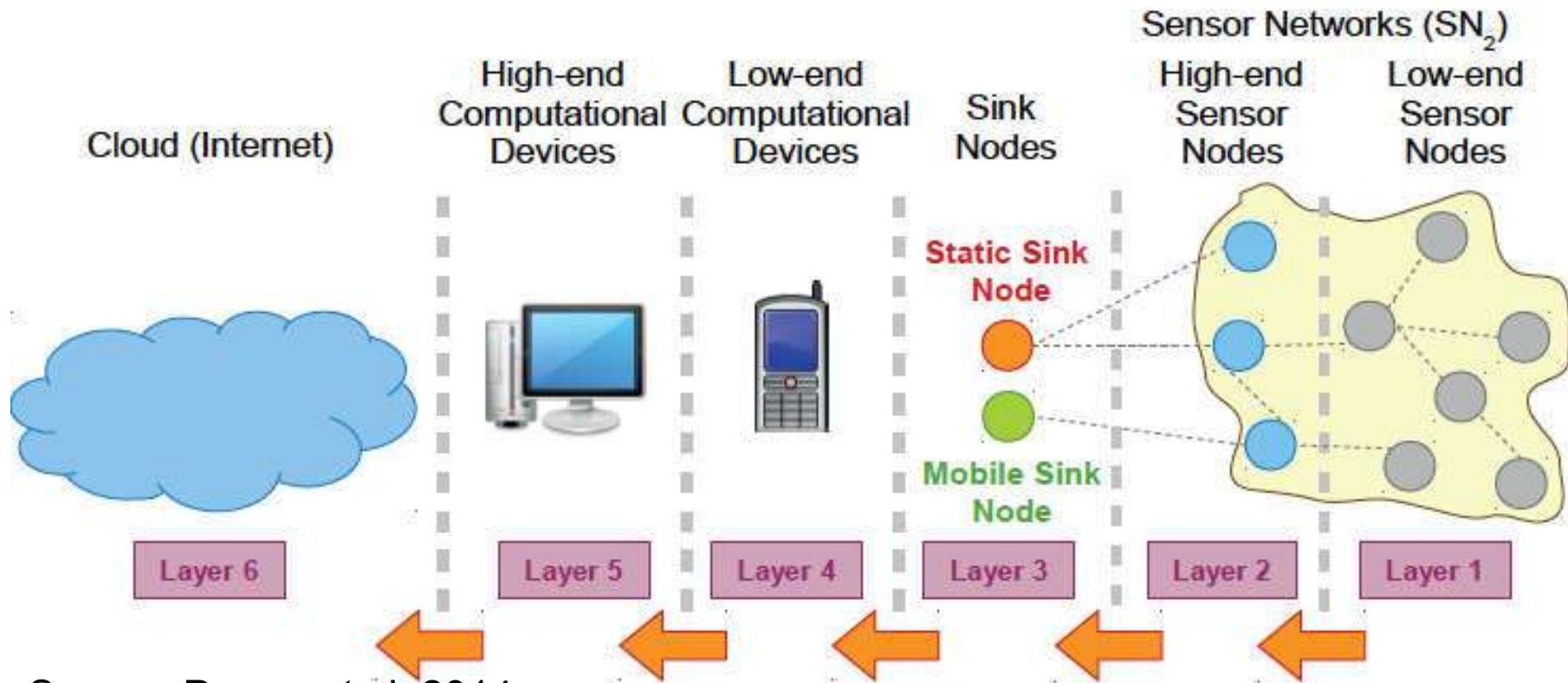
- Telecommunication systems
  - Initial/primary service: mobile voice telephony
  - Large coverage per access point (100s of meters – 10s of kilometers)
  - Low/moderate data rate (10s of kbit/s – 10s of Mbits/s)
  - Examples: GSM, UMTS, LTE
- WLAN
  - Initial service: Wireless Ethernet extension
  - Moderate coverage per access point (10s – 100s meters)
  - Moderate/high data rate (Mbits/s – 100s)
  - Examples: IEEE 802.11(a-g), Wimax

# Wireless Technologies

- 
- Short range:
    - Direct connection between devices – sensor networks
    - Typical low power usage
    - Examples: Bluetooth, Zigbee, Z-wave (house products)
  - Other examples:
    - Satellite systems
      - Global coverage
      - Applications: audio/TV broadcast, positioning, personal communications
    - Broadcast systems
      - Satellite/terrestrial
      - Support for high speed mobiles
    - Fixed wireless access
      - Several technologies including DECT, WLAN, IEEE802.16, etc.

# Sensor Networks (SNs)

- Consist of a certain number (which can be very high) of sensing nodes (generally wireless) communicating in a wireless multi-hop fashion



Source: Perera et al. 2014



# Sensor Networks (SNs)

---

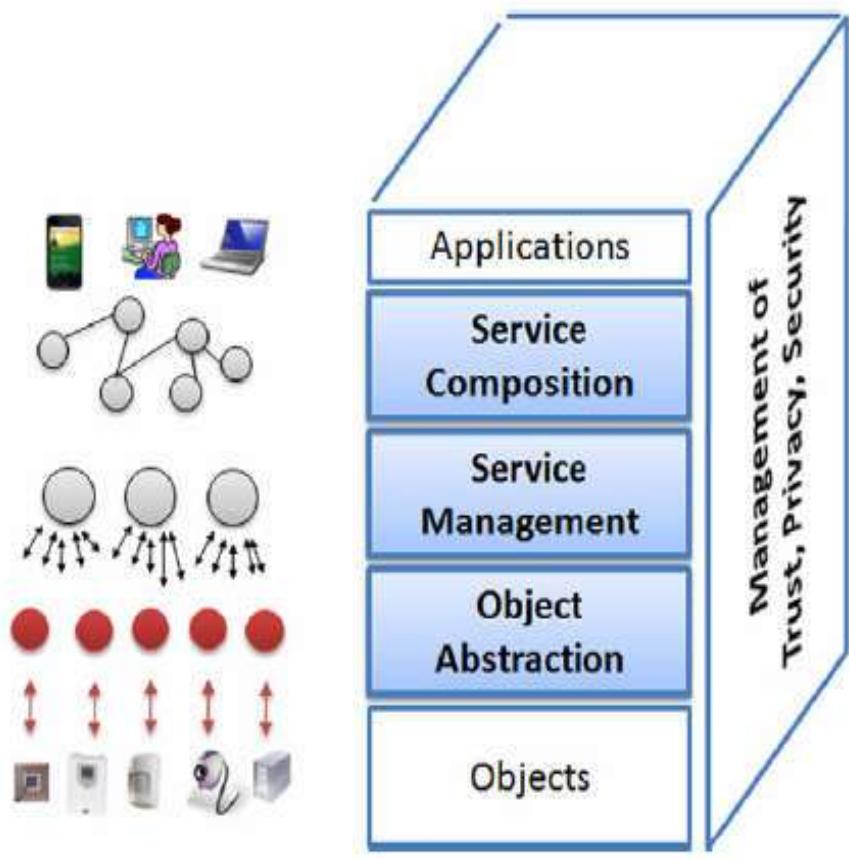
- SNs generally exist without IoT but IoT cannot exist without SNs
- SNs have been designed, developed, and used for specific application purposes
  - Environmental monitoring, agriculture, medical care, event detection etc.
- For IoT purposes, SNs need to have a middleware addressing these issues:
  - Abstraction support, data fusion, resource constraints, dynamic topology, application knowledge, programming paradigm, adaptability, scalability, security, and QoS support

# Middleware

---

- *Middleware is a software layer that stands between the networked operating system and the application and provides well known reusable solutions to frequently encountered problems like heterogeneity, interoperability, security, dependability [Issarny, 2008]*
- IoT requires stable and scalable middleware solutions to process the data coming from the networking layers

# Service Oriented Architecture (SOA) see



- Middleware solutions for IoT usually follow SOA approaches
- Allows SW/HW reuse
  - Doesn't impose specific technology
- A layered system model addressing previous issues
  - Abstraction, common services, composition

Source: Atzori et al. 2010



# Other Middleware Examples

- Fosstrak Project
  - Data dissemination/aggregation/filtering/interpretation
  - Fault and configuration management, lookup and directory service, tag ID management, privacy
- Welbourne et al.
  - Tag an object/create-edit location info/combine events collected by antennas
- e-Sense Project
  - Middleware only collects data in a distributed fashion and transmits to actuators
- UbiSec&Sens Project
  - Focuses on security → secure data collection, data store in memory, etc.

# Open Problems and Challenges

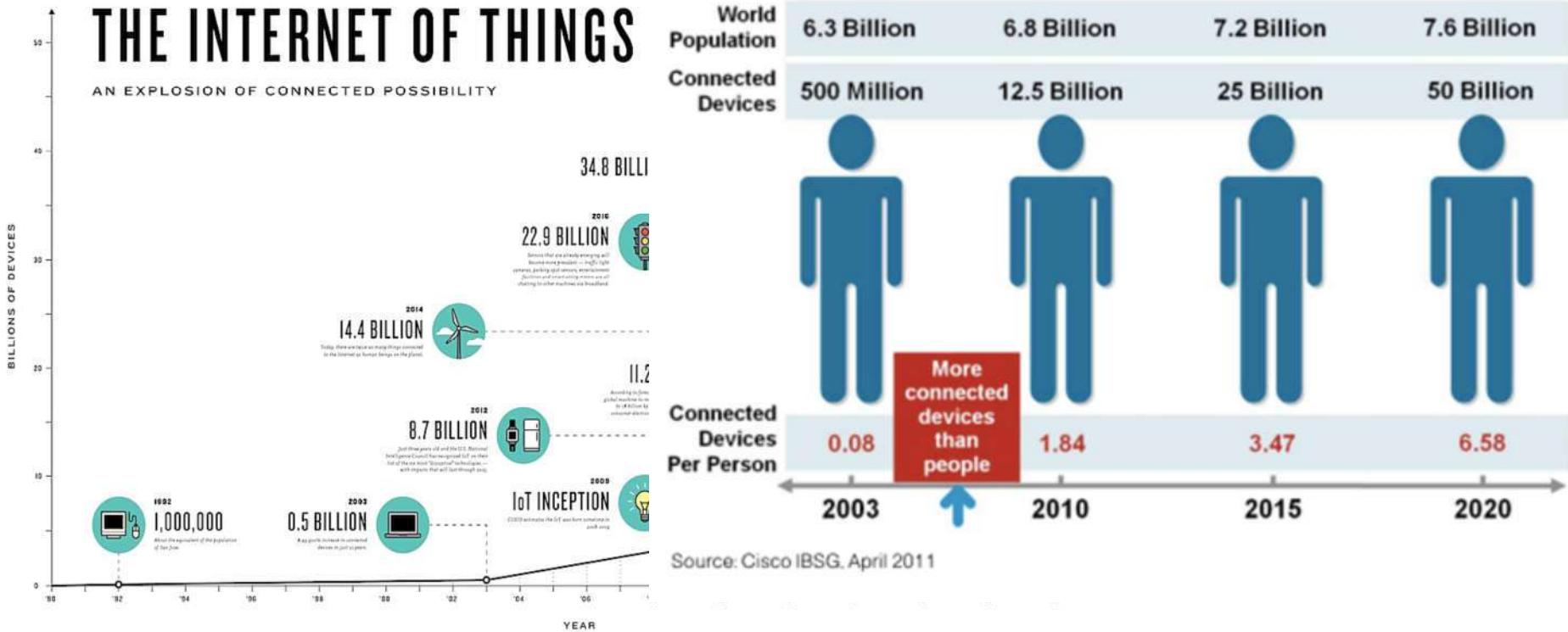
- Lack of standardization
- Scalability
  - Addressing issues
  - Understanding the big data
- Support for mobility
- Address acquisition
- New network traffic patterns to handle
- Security/Privacy issues

# Standardization

- Several standardization efforts but not integrated in a comprehensive framework
- Open Interconnect Consortium: Atmel, Dell, Intel, Samsung and Wind River
- Industrial Internet Consortium: Intel, Cisco, GE, IBM
- AllSeen Alliance: Led by Qualcomm, many others

| Standard  | Objective  | Status   | Comm.<br>range (m) | Data rate<br>(kbps) | Unitary<br>cost (\$) |
|---|--|----------|--------------------|---------------------|----------------------|
| <i>Standardization activities discussed in this section</i> |  |          |                    |                     |                      |
| EPCglobal   | Integration of RFID technology into the electronic product code (EPC) framework, which allows for sharing of information related to products             | Advanced | ~1                 | $\sim 10^2$         | ~0.01                |
| GRIPS   | European Coordinated Action aimed at defining RFID standards supporting the transition from localized RFID applications to the <i>Internet of Things</i> | Ongoing  | ~1                 | $\sim 10^2$         | ~0.01                |
| M2M   | Definition of cost-effective solutions for machine-to-machine (M2M) communications, which should allow the related market to take off                    | Ongoing  | N.S.               | N.S.                | N.S.                 |
| 6LoWPAN   | Integration of low-power IEEE 802.15.4 devices into IPv6 networks  | Ongoing  | 10–100             | $\sim 10^2$         | ~1                   |
| ROLL  | Definition of routing protocols for heterogeneous low-power and lossy networks   | Ongoing  | N.S.               | N.S.                | N.S.                 |
| <i>Other relevant standardization activities</i>            |  |          |                    |                     |                      |
| NFC   | Definition of a set of protocols for low range and bidirectional communications  | Advanced | $\sim 10^{-2}$     | Up to 424           | ~0.1                 |
| Wireless Hart   | Definition of protocols for self-organizing, self-healing and mesh architectures over IEEE 802.15.4 devices  | Advanced | 10–100             | $\sim 10^2$         | ~1                   |
| ZigBee  | Enabling reliable, cost-effective, low-power, wirelessly networked, monitoring and control products  | Advanced | 10–100             | $\sim 10^2$         | ~1                   |

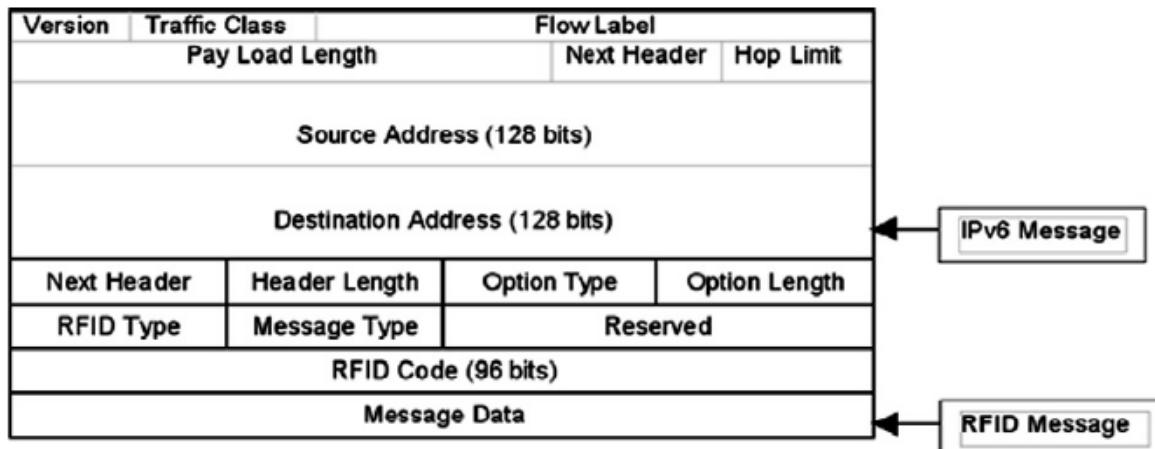
# Scalability



- Number of devices increasing exponentially
  - How can they uniquely be tagged/named?
  - How can the data generated by these devices be managed?

# Addressing Issues

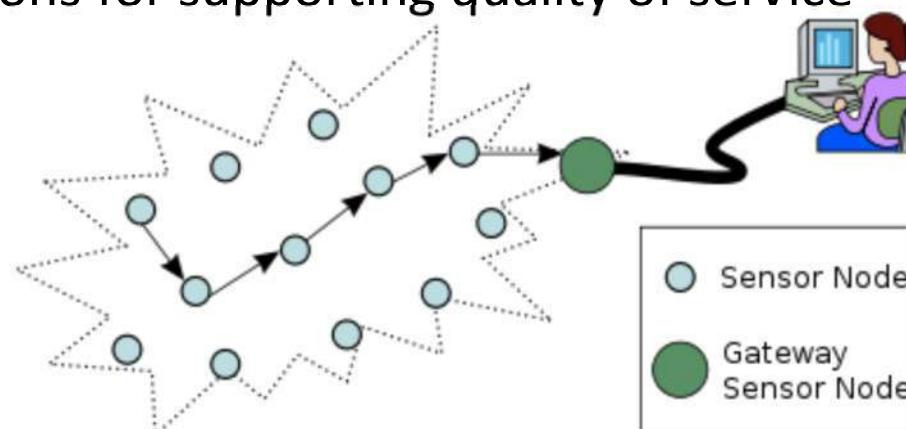
- Incredibly high number of nodes, each of which will produce content that should be retrievable by any authorized user
  - This requires effective addressing policies
  - IPv4 protocol may already reached its limit. Alternatives?
  - IPv6 addressing has been proposed for low-power wireless communication nodes within the 6LoWPAN context
- IPv6 addresses are expressed by means of 128 bits → 10<sup>38</sup> addresses, enough to identify objects worth to be addressed
- RFID tags use 64–96 bit identifiers, as standardized by EPCglobal, solutions to enable the addressing of RFID tags into IPv6 networks



Encapsulation of RFID message into an IPv6 packet.  
 Source: Atzori et al. (2010)

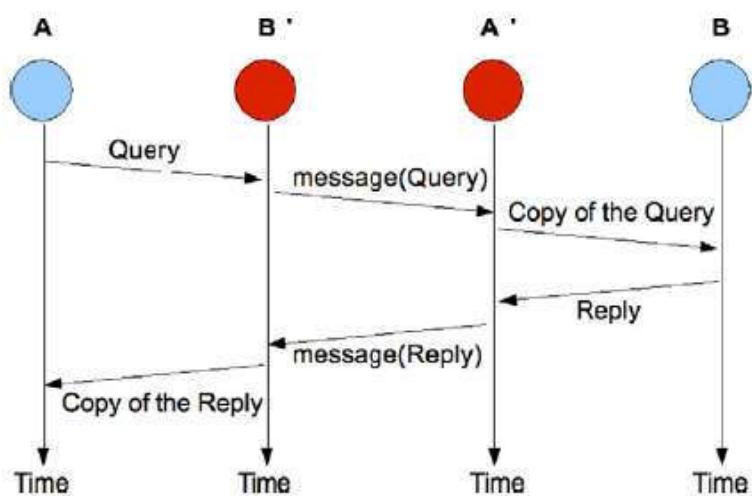
# New Traffic to Handle

- The characteristics of the smart objects traffic in the IoT is still not known
  - Important → basis for the design of the network infrastructures and protocols
- Wireless sensor networks (WSNs) traffic characterization
  - Strongly depend on the application scenario
  - Problems arise when WSNs become part of the overall Internet
  - The Internet will be traversed by a large amount of data generated by sensor networks deployed for heterogeneous purposes → extremely different traffic characteristics
  - Required to devise good solutions for supporting quality of service



# Security

- The components spend most of the time unattended
  - It is easy to physically attack them
- IoT components are characterized by low capabilities in terms of both energy and computing resources
  - They can't implement complex schemes supporting security
- Authentication problem
  - Proxy attack, a.k.a. man in the middle attack problem



- Data integrity
  - Data should not be modified without the system detecting it
  - Attacks on the node
    - Memory protection
  - Attacks over the network
    - Keyed-Hash Message Auth. Code

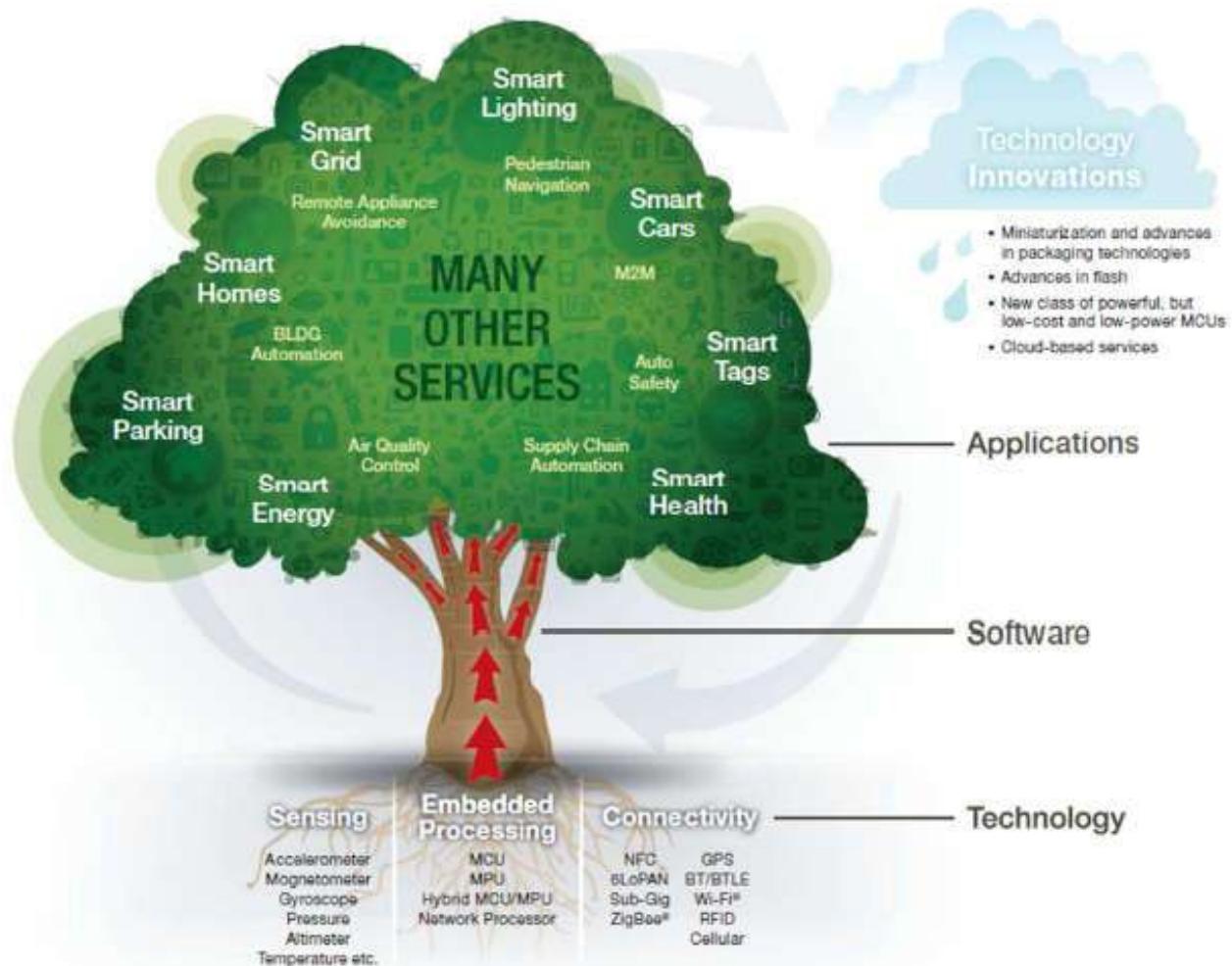
Man in the middle attack Source: Atzori et al. (2010)

# Privacy

- How is it different than traditional privacy?
  - Legislative issues
  - Ethics issues
- Easy for a person to get involved in IoT even if he/she does not know
- Data can be stored indefinitely
- Current solutions are not enough
  - Encryption, pseudo-noise signal, privacy broker

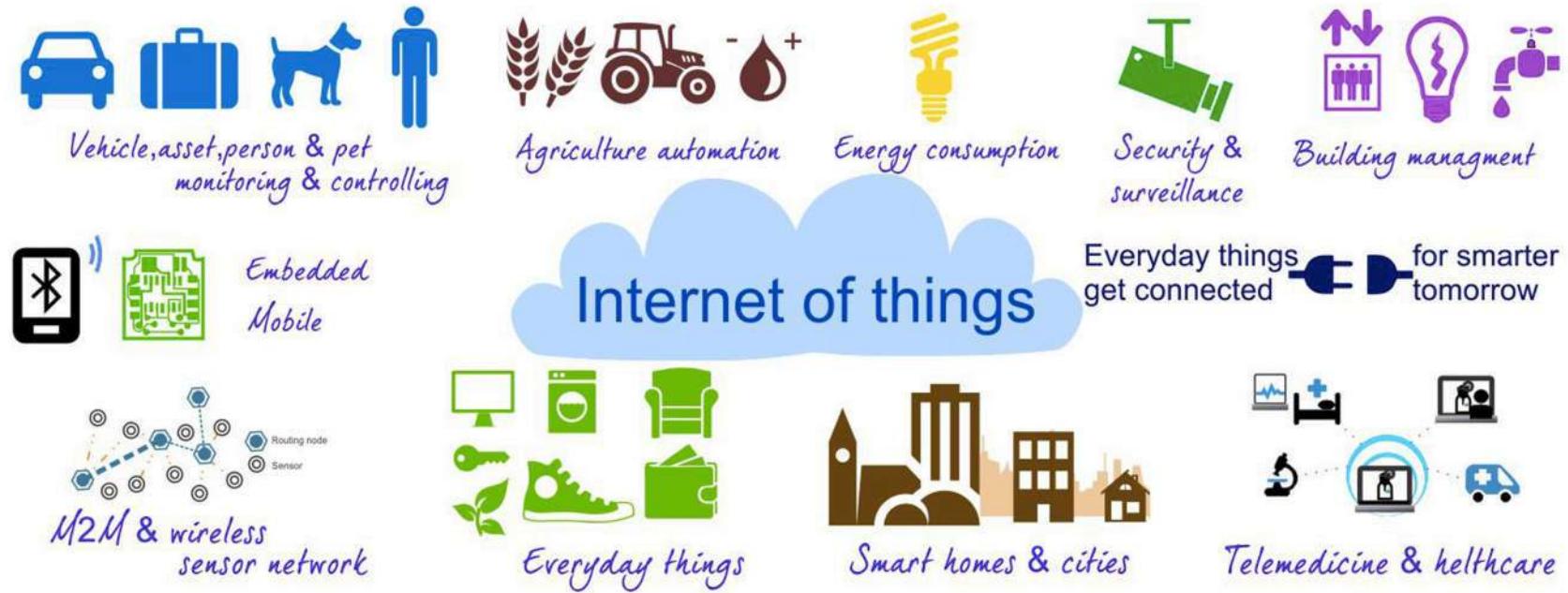


# Again - Overall Picture



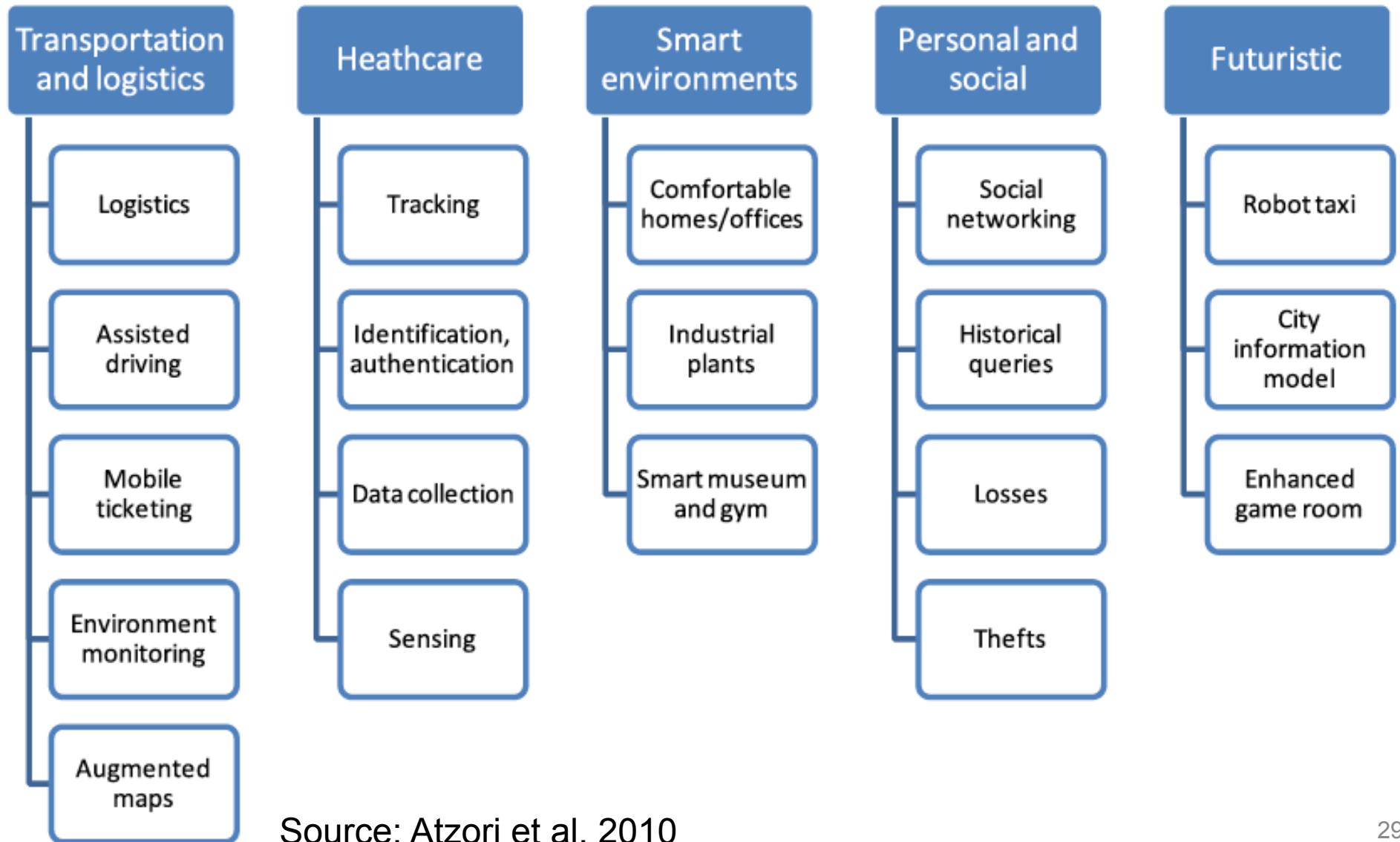
Source: "What the Internet of Things (IoT) Needs to Become a Reality,"  
White Paper, by K. Karimi and G. Atkinson

# Applications



- Several different domains
  - Transportation and logistics
  - Healthcare
  - Smart environment (home, office, etc.)
  - Personal and social domain

# Application Domains and Scenarios



Source: Atzori et al. 2010

# Healthcare Applications

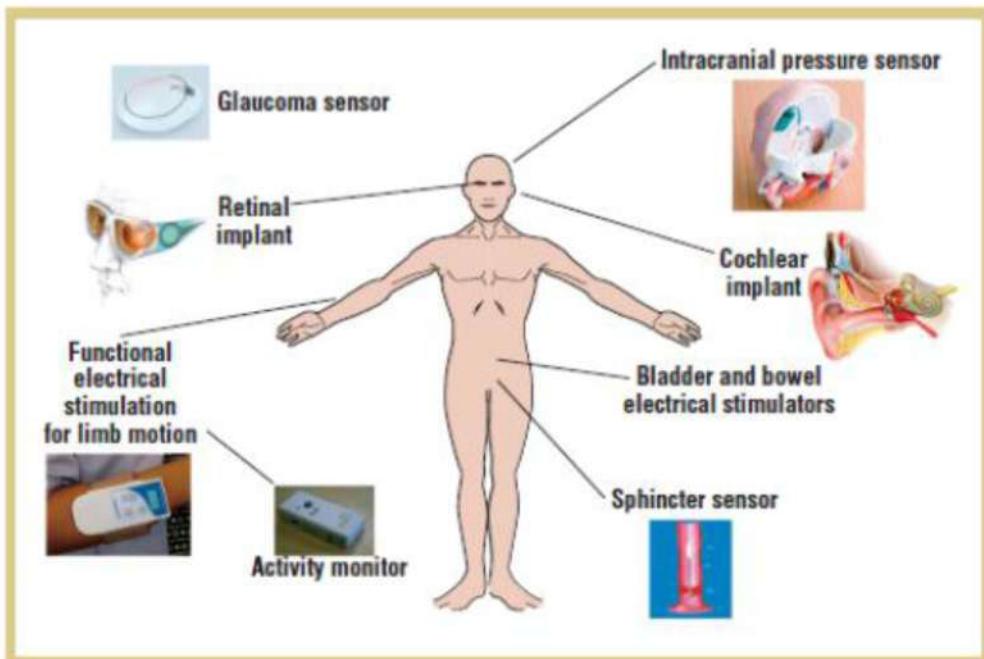


Figure 6. Fully implantable wireless sensor for the intracranial pressure monitoring system.

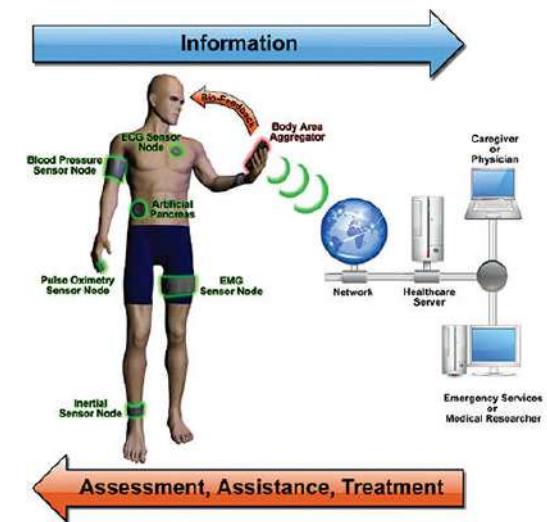
Source: Qian Zhang. Lecture notes. 2013

- Various sensors for various conditions
- Example ICP sensor: Short or long term monitoring of pressure in the brain cavity
- Implanted in the brain cavity and senses the increase of pressure
- Sensor and associated electronics encapsulated in safe and biodegradable material
- External RF reader powers the unit and receives the signal
- Stability over 30 days so far

# Healthcare Applications



- Other applications:
  - National Health Information Network
  - Electronic Patient Record
  - Home monitoring and control
    - Pulse oximeters, blood glucose monitors, infusion pumps, accelerometers
  - Bioinformatics
    - Gene/protein expression
    - Systems biology
    - Disease dynamics



Source: Qian Zhang. Lecture notes. 2013



# Environmental Application: CitiSense

- Air quality monitoring project in UCSD CSE

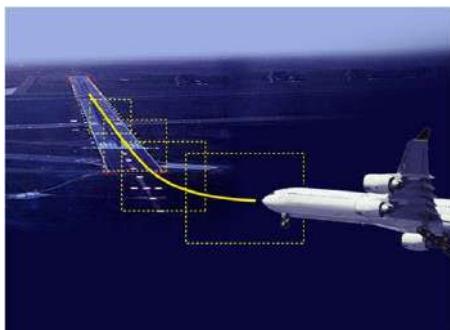


- Environmental application
- Electrochemical **sensors**, **microcontroller** for data collection and transmission to an **Android** app
- **Actuation**: air quality is immediately reported, as well as retransmitted to a backend for larger-scale analysis



# Transportation Applications

- **Vehicle control:** Airplanes, automobiles, autonomous vehicles
  - All kinds of sensors to provide accurate, redundant view of the world
  - Several processors in cars (Engine control, break system, airbag deployment system, windshield wiper, door locks, entertainment system, etc.)
  - Actuation is maintaining control of the vehicle
  - Very tight timing constraints and requirements enforced by the platforms

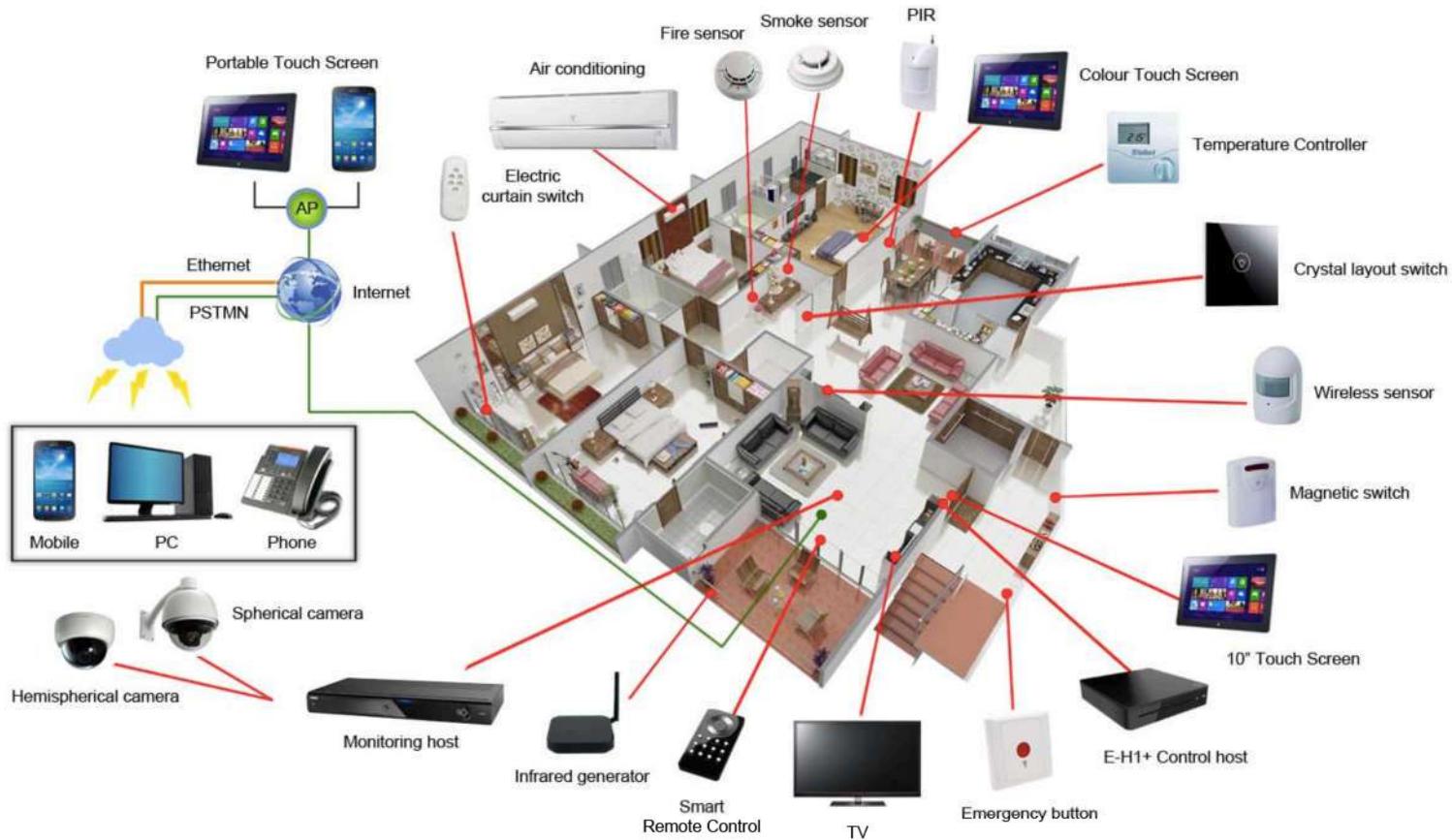




# Example Transportation Scenarios

1. A network of sensors in a vehicle can interact with its surroundings to provide information
  - Local roads, weather and traffic conditions to the car driver
  - Adaptive drive systems to respond accordingly
2. Automatic activation of braking systems or speed control via fuel management systems.
  - Condition and event detection sensors can activate systems to maintain driver and passenger comfort and safety through the use of airbags and seatbelt pre-tensioning
3. Sensors for fatigue and mood monitoring based on driving conditions, driver behavior and facial indicators
  - Ensuring safe driving by activating warning systems or directly controlling the vehicle

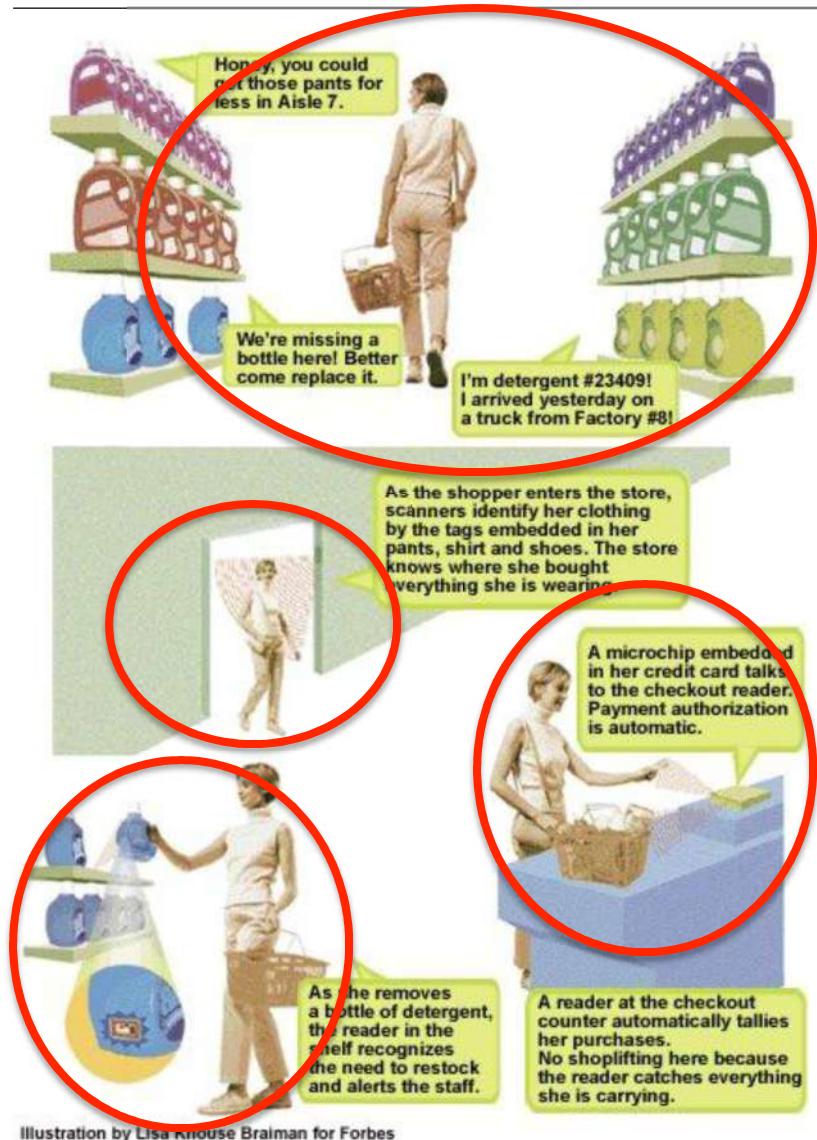
# Smart Home Applications



- Smart meters, heating/cooling, motion/temperature/lighting sensors, smart appliances, security, etc.



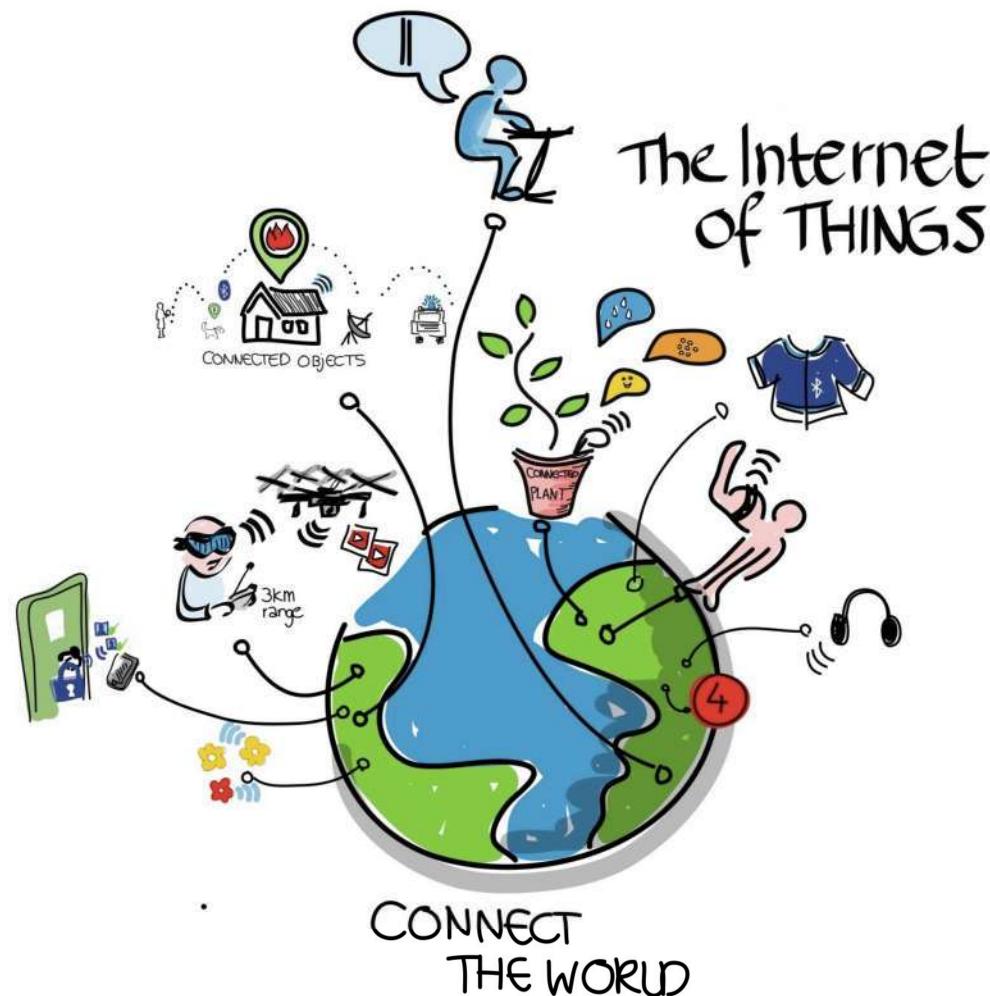
# A Futuristic Application: Shopping see



- When entering the doors, scanners will identify the tags on her clothing.
- When shopping in the market, the goods will introduce themselves.
- When paying for the goods, the microchip of the credit card will communicate with checkout reader.
- When moving the goods, the reader will tell the staff to put a new one.

Source: Qian Zhang. Lecture notes. 2013

# An exciting future!



# Distributed and Edge Computing

# Distributed Systems Security

Sharad K. Ghimire

Department of Electronics and Computer Engineering  
Pulchowk Campus  
Institute of Engineering  
Tribhuvan University

# Distributed Systems Security

Introduction

Overview of cryptography and data privacy

Security issues and techniques in distributed system

Security challenges in edge computing

Introduction to digital forensics

# Contents

Introduction

Overview of cryptography and data privacy

# Security

The word 'secure' Derived from Latin *securus*, meaning freedom from anxiety: *se* (without) + *cura* (care, anxiety)

# Security ?

The state of being free from danger or threat

Security can be defined as being free from danger, or feeling safe

Protection of a person, building, organization, or country against threats such as crime or attacks by foreign countries or terrorist

# Security (Related with IT)

Security is all about protecting valuables

Here the “valuables” are computer related assets instead of money

In these days the protection of money is a subset of computer asset security

Information seems to be the currency of the 21st century

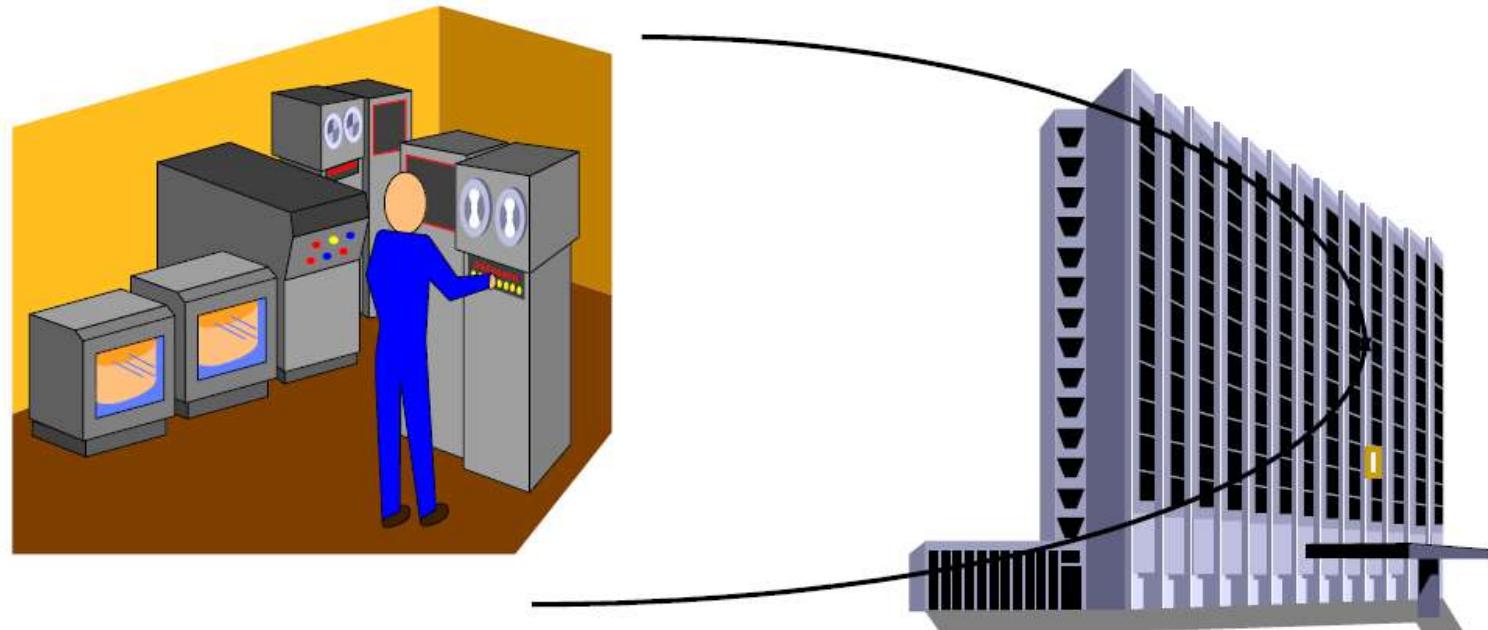
The protection of computer systems and networks from the theft of or damage to their hardware, software, or electronic data, as well as from the disruption or misdirection of the services they provide

# Why Security

**Past:** Computers and computer communication were primarily used by university researchers for sending email and by corporate employees for sharing resources ⇒ security was not the issue

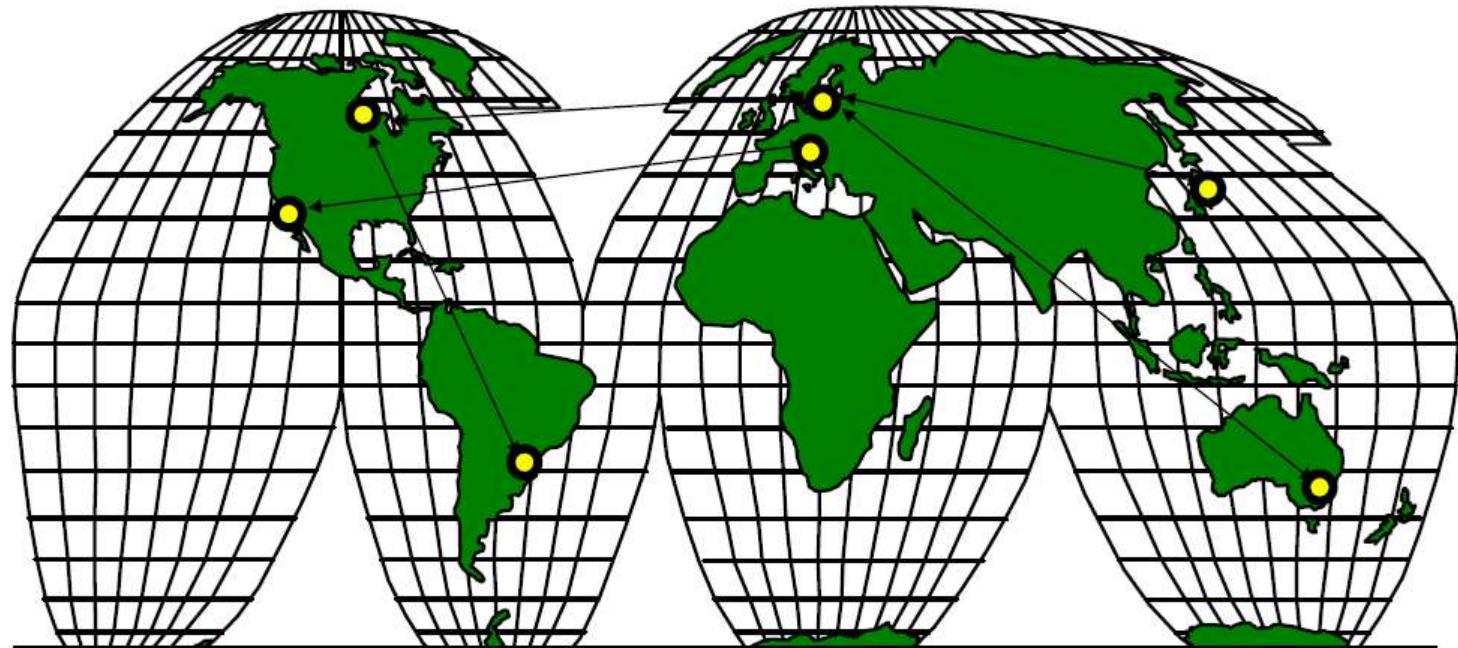
**Now:** The millions of ordinary citizens are using network communications for banking, shopping, using e-payment systems and many more, so weakness after weakness has been found ⇒ security become a great issue

## Past Situation (Single System)



**Physical security and control of access to computers**

# Current Situation



Authentication, message protection, authorization

# Why Security

When credit card payments over the Internet were first considered ⇒ traffic between customer and merchant need to be protected

Danger might be scanning Internet traffic for packets containing credit card numbers is an attack with a low yield

SSL was developed to deal with this problem

Badly protected servers at a merchant site holding a database of customer credit card numbers are a much more rewarding target and such attacks have occurred, either to obtain credit card numbers or to blackmail the merchant

Identity theft

# Security Engineering

A field of engineering that focuses on the security aspects during design of systems that need to be able to deal robustly with possible sources of disruption, ranging from natural disasters to malicious acts

Security engineering is about **building systems to remain dependable in the face of malice, error, or mischance**

We can say that security engineering has a **goal to raise the effort for an attack to a level where the costs exceed the attacker's gains**

# Security Requirements

# Security Requirements

## **Confidentiality –**

- Protect data contents and access

## **Integrity –**

- Protect data accuracy

## **Availability –**

- Ensure timely service

# Threats

## Security Attacks

### Threat to confidentiality

Snooping

Traffic analysis

### Threat to integrity

Modification

Masquerading

Replaying

Repudiation

### Threat to availability

Denial of service

# Threats to confidentiality

## Snooping:

- Unauthorized access to or interception of data

## Traffic Analysis:

- Encipherment of data may make it non-intelligible for the interceptor, but s/he can obtain some other type information by monitoring online traffic, e.g. find the electronic address of sender and receiver and guess the nature of transaction and so on

# Threats to integrity

**Modification:** Modification of information to make it beneficial

**Masquerading:** Masquerading or spoofing, happens when the attacker impersonates somebody else. For example, an attacker might steal the bank card and PIN of a customer and pretend that s/he is that customer

**Replaying:** Attacker obtains a copy of a message sent by a user and later tries to replay it

**Repudiation:** This type of attack is different from others because it is performed by one of the two parties in the communication; either the sender or the receiver. The sender of the message might later deny that she has sent the message

# Threats to availability

## **Denial of Service:**

A very common attack

It may slow down or totally interrupt the service of a system by using various techniques

# Attacks Types

Classifying security attacks (RFC 2828) in terms of passive and active attacks:

**Passive attack:** Attempts to learn or make use of information from the system but does not affect system resources

**Active attack:** Attempts to alter system resources or affect their operation

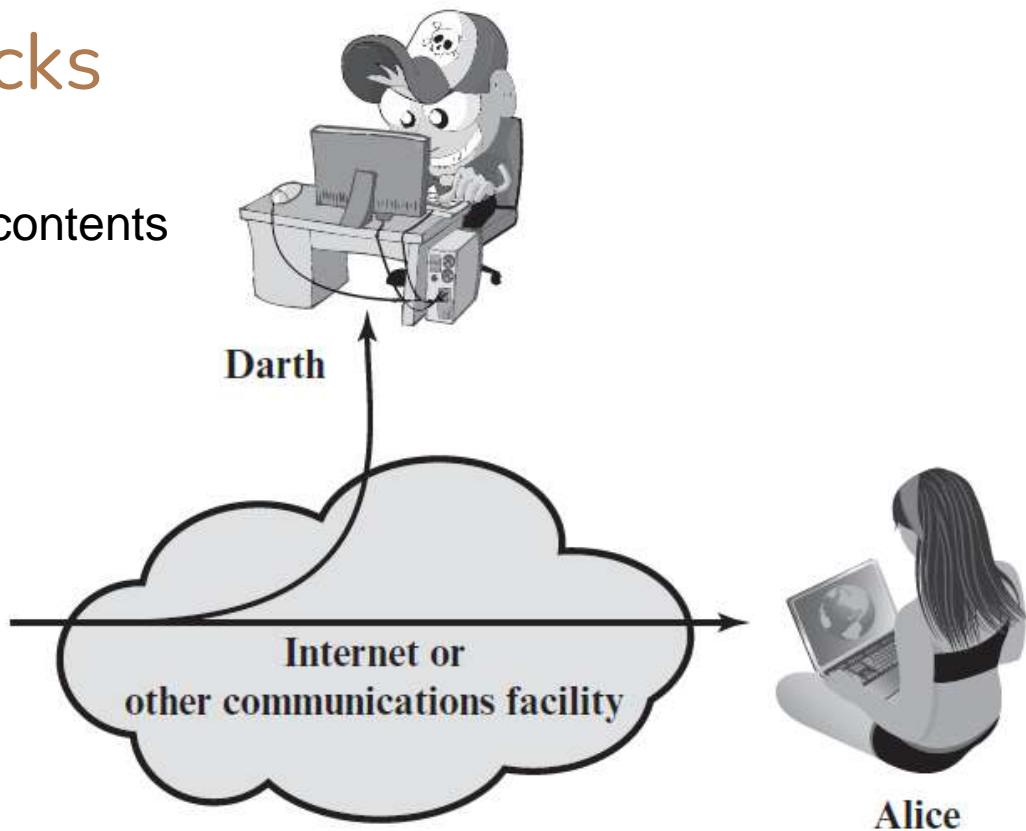
# Passive Attacks

Release of message contents

Traffic analysis



Bob



Alice

# Passive Attacks

Attempt to learn or make use of information from the system but do not affect system resources

To obtain information

- Release of possibly sensitive/confidential message contents
- Traffic analysis which monitors frequency and length of messages to get info on senders

Difficult to detect

But relatively easy to prevent, can be prevented using encryption

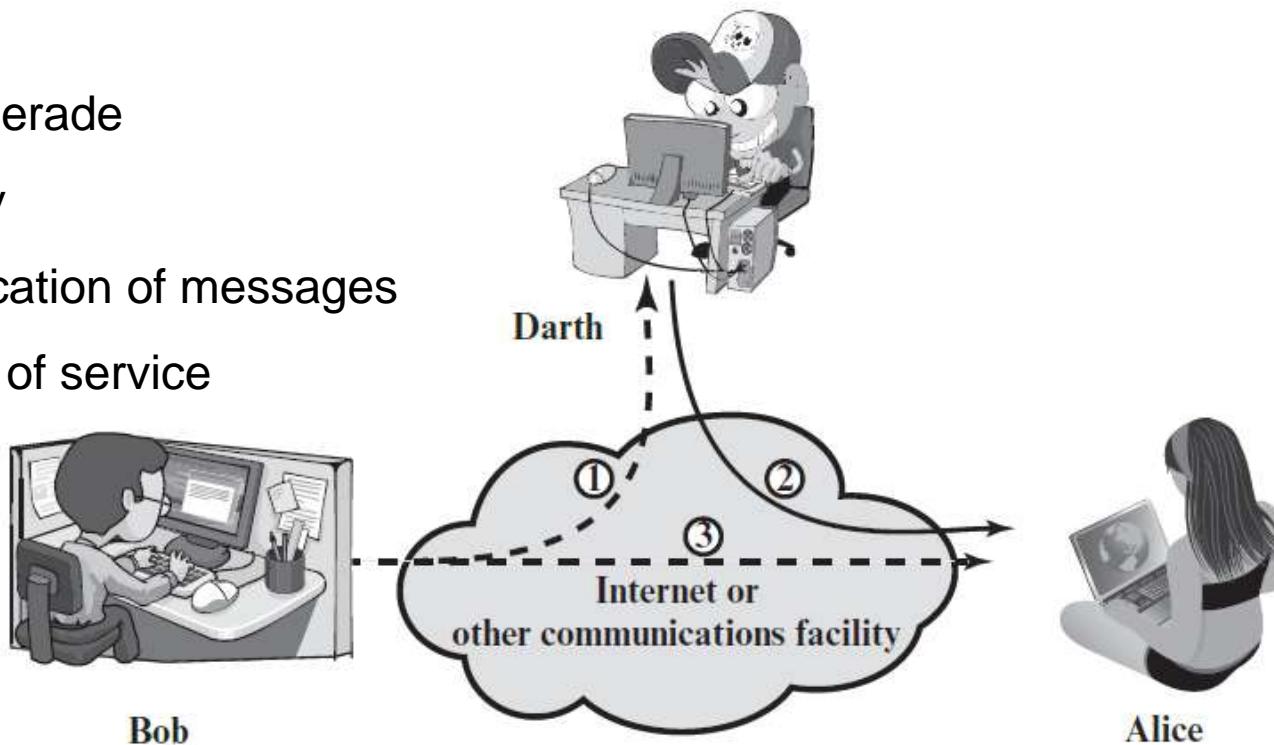
# Active Attacks

Masquerade

Replay

Modification of messages

Denial of service



# Active Attacks

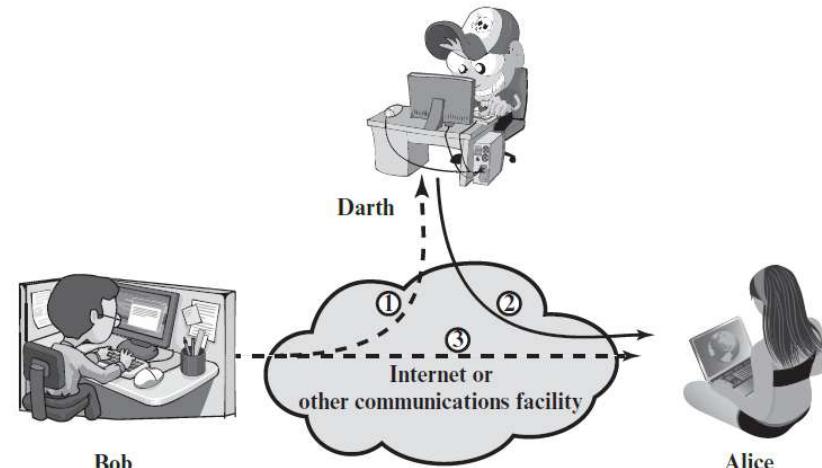
Active attack attempts to alter system resources or affect their operation

- Masquerade
- Replay
- Modification of messages
- Denial of service

Easy to detect

Hard to prevent

Focus on detection and recovery



# Techniques

Some techniques are used for implementation of security goals:

## Cryptography

- Greek origin meaning - “secret writing”
- Transforming message to make them secure and immune to attacks

## Steganography

- Greek origin meaning - “covered writing”
- Concealing the message itself by covering it with something else

# Cryptography

Cryptography, a word with Greek origins means “secrete writing”

From Ancient Greek: translit "hidden, secret"; and graphein, "**to write**", or "**study**" - **practice and study** of techniques for secure communication

# Cryptography

An art of achieving security by encoding messages to make them non-readable i.e. to introduce secrecy

Process of hiding or coding information so that only the intended person or system can read it

Has been used to encode messages for thousands of years

Historically, four groups of people have used and contributed to the art of cryptography: Military, Diplomatic groups, Diarists, and Lovers

## Cryptography (contd...)

It is being used in credit cards, computer passwords, ecommerce and many more in nowadays

It is about constructing and analyzing protocols that prevent third parties or the public from reading the private messages

Importance is being increased day by day

## Cryptography (contd...)

In the early days, cryptography used to be performed by using manual techniques

For example, let the message were the word "**private**" can be converted in to different form to yield **TVMZEXI** so that it cannot be easily understood by anyone

Initially it seems to be difficult to crack but once the algorithm is known, the secrecy is lost

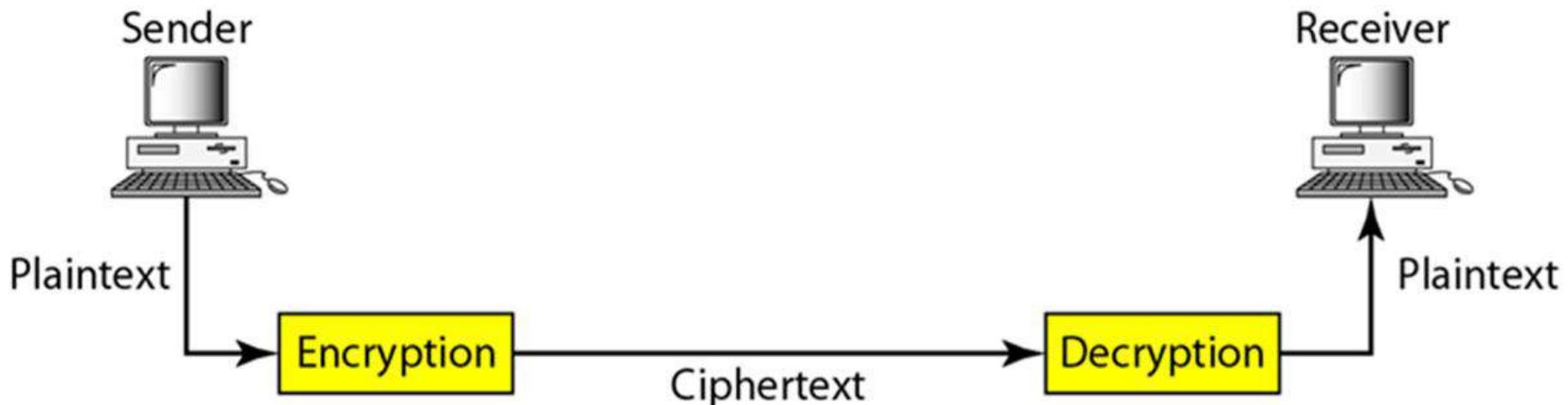
## Cryptography (contd...)

Some cryptographic algorithms are very trivial to understand, replicate and therefore easy to crack

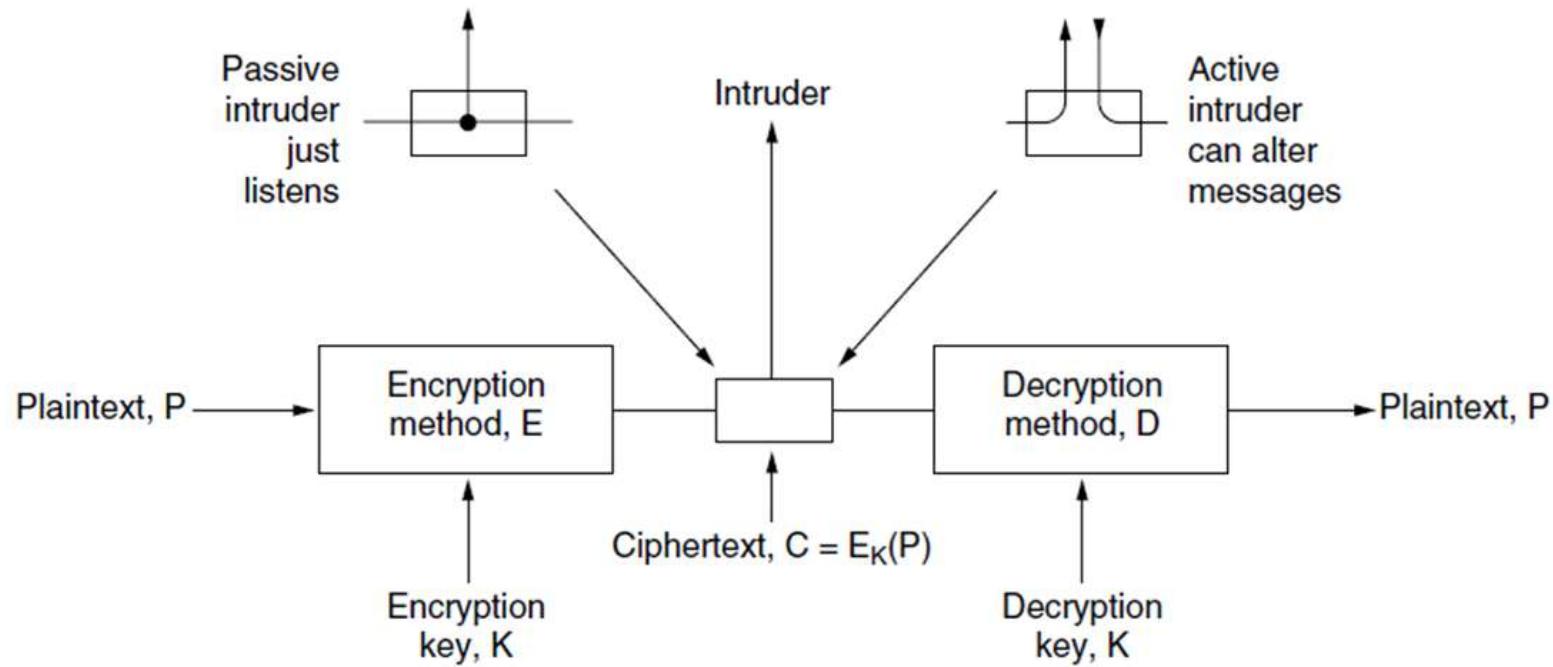
Some other cryptography algorithms are highly complicated and therefore difficult to crack

The rest are somewhere in the middle

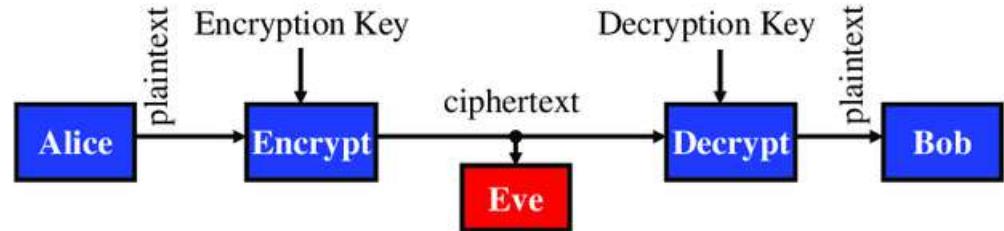
# Cryptography Components



# Encryption Model



# Terminologies



**Plaintext**:- Refers to a message in plain form, i.e. the original message that is readily intelligible (also called cleartext)

**Encryption**:- Process of disguising the message to hide it from unwanted ones

**Ciphertext**:- A disguised message obtained by encryption process to the plaintext

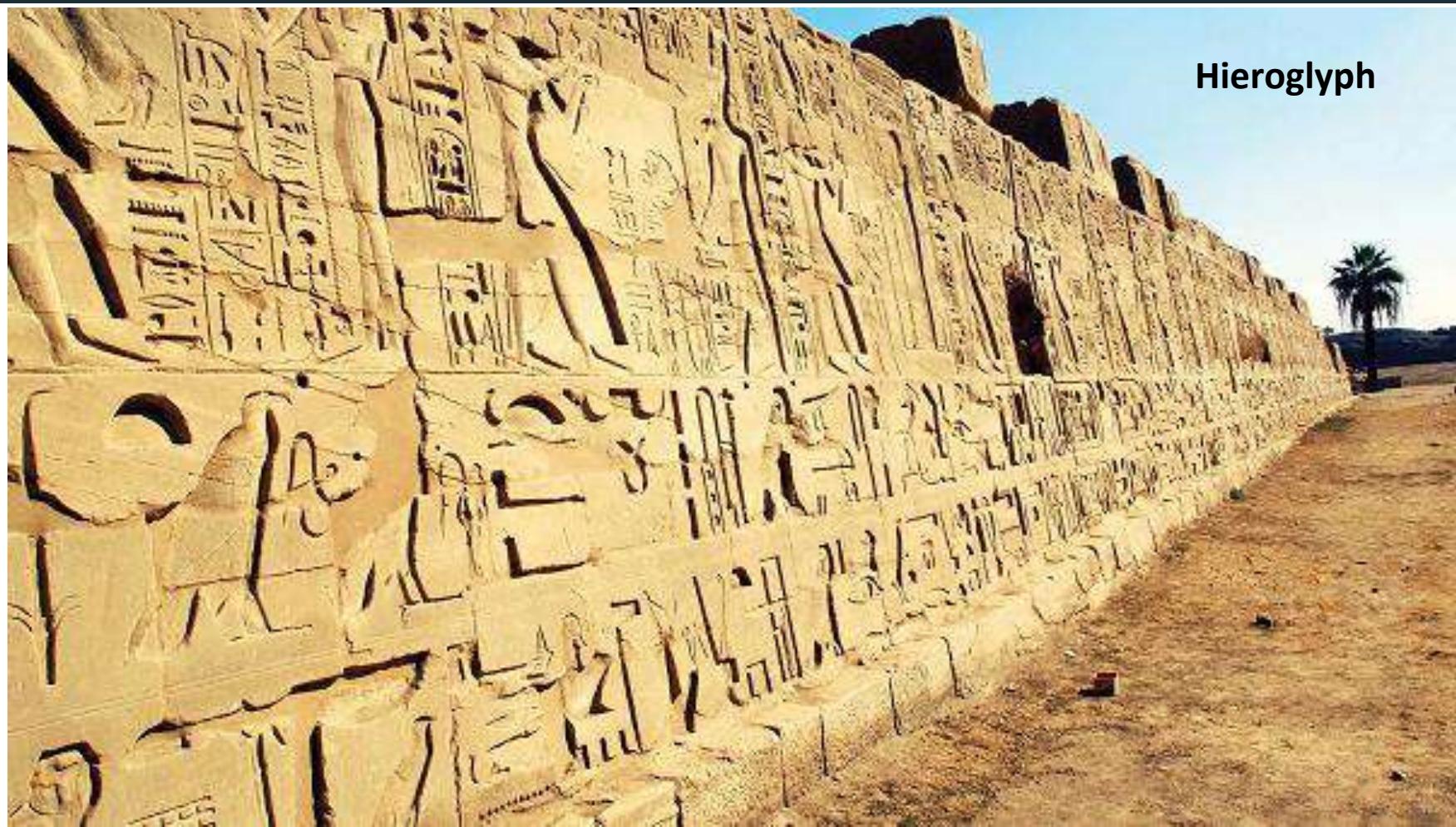
**Decryption**:- The process of extracting the plaintext from ciphertext

**Cipher**:- An algorithm used for performing encryption or decryption

**Key**:- A value (or stream of characters or bits) that is applied in cryptography to get ciphertext from plaintext or plaintext from ciphertext

# History of Cryptography

Hieroglyph



# Caesar Cipher

**Around 100 BC**, Julius Caesar was known to use a form of encryption to convey secret messages to his army generals posted in the war front

This substitution cipher, known as **Caesar cipher**, is perhaps the most mentioned historic cipher in academic literature

In a substitution cipher, each character of the plain text is substituted by another character to form the cipher text

It was a shift by 3 cipher, i.e., each character was shifted by 3 places, so the character 'A' was replaced by 'D', 'B' by 'E', and so on

## History of Cryptography (contd...)

Up to the Second World War, most of the work on cryptography was for military purposes, usually used to hide secret military information

Cryptography attracted commercial attention post-war, with businesses trying to secure their data from competitors

In the early 1970's, IBM realized that their customers were demanding some form of encryption, so they formed a "crypto group" headed by Horst-Feistel ⇒ designed a cipher called **Lucifer**

## History of Cryptography (contd...)

In 1973, the Nation Bureau of Standards (now called NIST) in the US put out a request for proposals for a block cipher which would become a national standard ⇒ Lucifer was eventually accepted and was called **DES or the Data Encryption Standard**

In 1997, and in the following years, DES was broken by an exhaustive search attack (using high processing power of computer)

The main problem with DES was the small size of the encryption key

## History of Cryptography (contd...)

As computing power increased it became easy to brute force all different combinations of the key to obtain a possible plain text

In 1997, NIST again put out a request for proposal for a new block cipher ⇒ which received several submissions

In 2000, it accepted Rijndael, and christened it as **AES** or the **Advanced Encryption Standard**

Now AES is a widely accepted standard used for symmetric encryption

# Kerckhoffs's Principle

Stated by Dutch-born cryptographer Auguste Kerckhoffs in the 19th century, also known as Kerckhoff's law ⇒ a principle in cryptography

According to the principle, a cryptosystem should be secure, even if everything about the system, except the key, is public knowledge

Referred to as "security through obscurity is not the security", i.e. opposite to the idea of "security through obscurity"

This concept is widely embraced by many cryptographers

## Kerckhoffs's Principle (contd...)

Based on the idea that the security of a cryptographic system should not depend on the secrecy of its algorithm, design or implementation

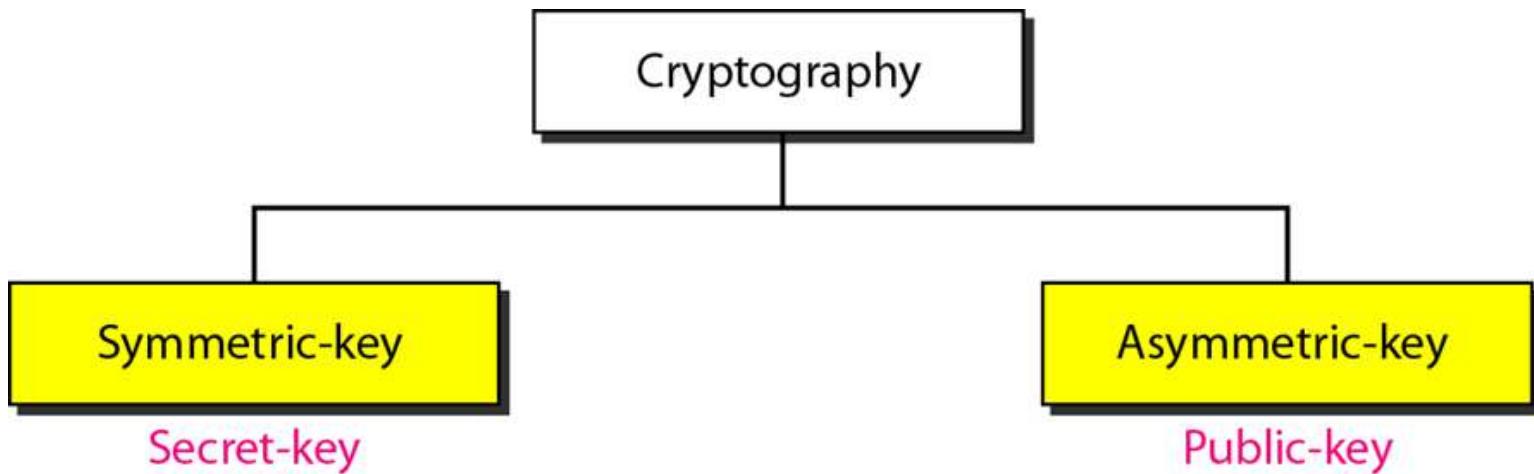
Instead, the security of the system should depend solely on the secrecy of the key that should be kept secret

A security system is secure only if its details can be shared with the world

Allows others to verify the security of the protocol without compromising the security of the system

# Cryptography Categories

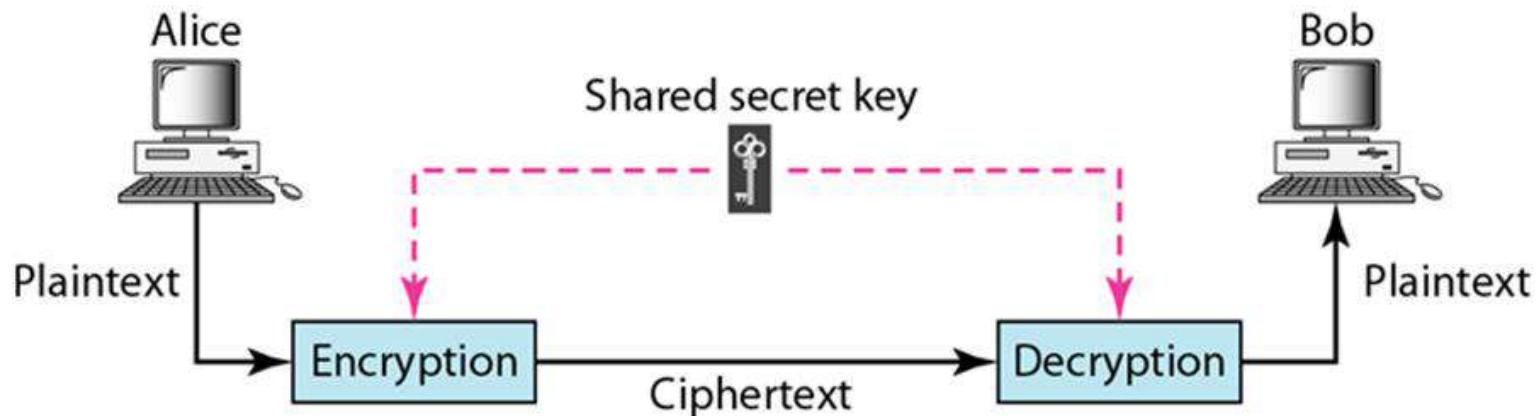
# Categories of Cryptography



# Symmetric-key Cryptography

Same key is used by the sender (for encryption) and the receiver (for decryption)

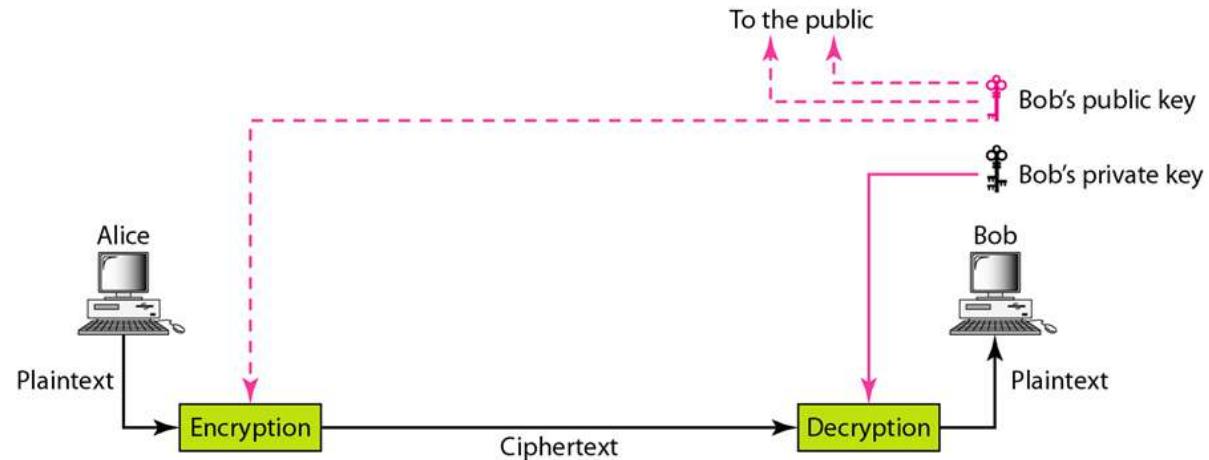
The key need to be shared between sender and receiver

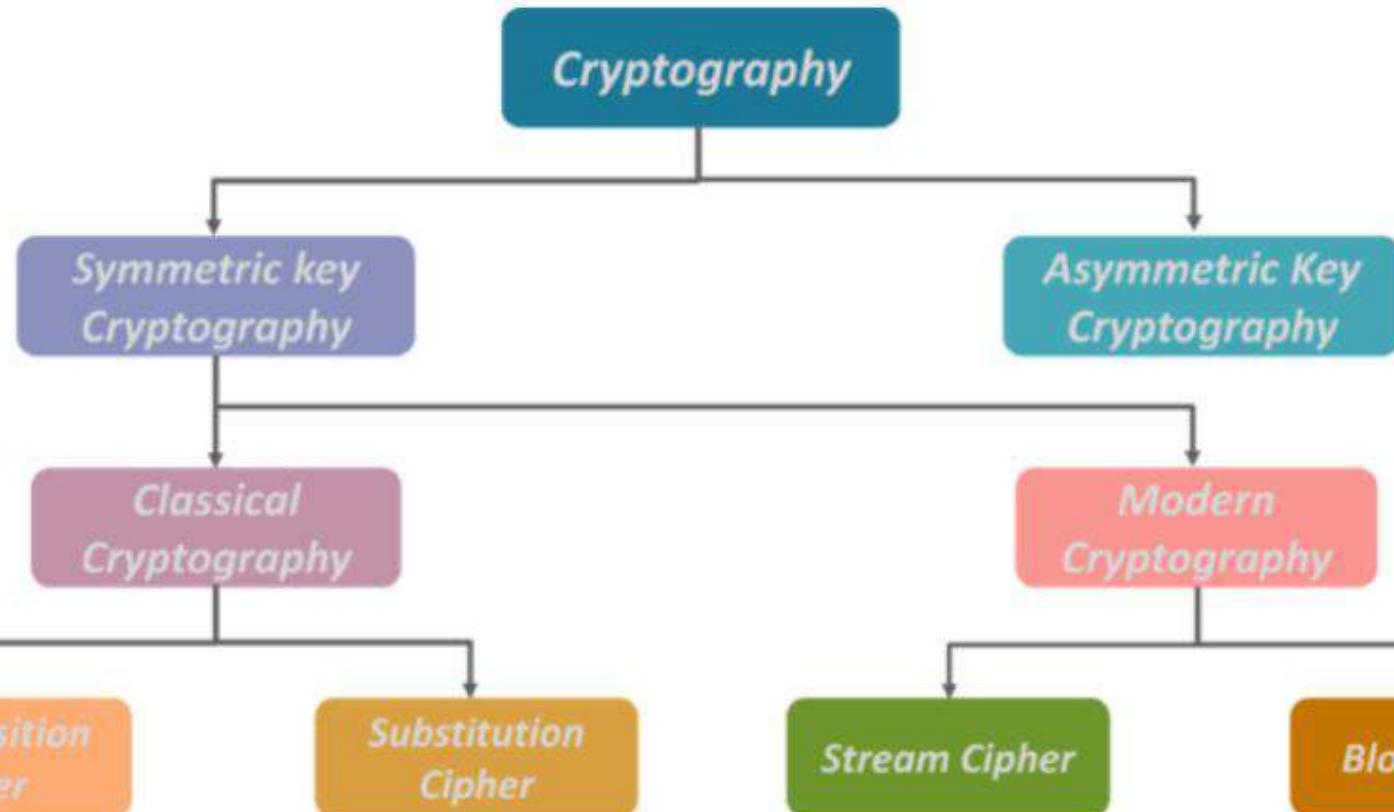


# Asymmetric-key Cryptography

There are two keys: A private key and a public key

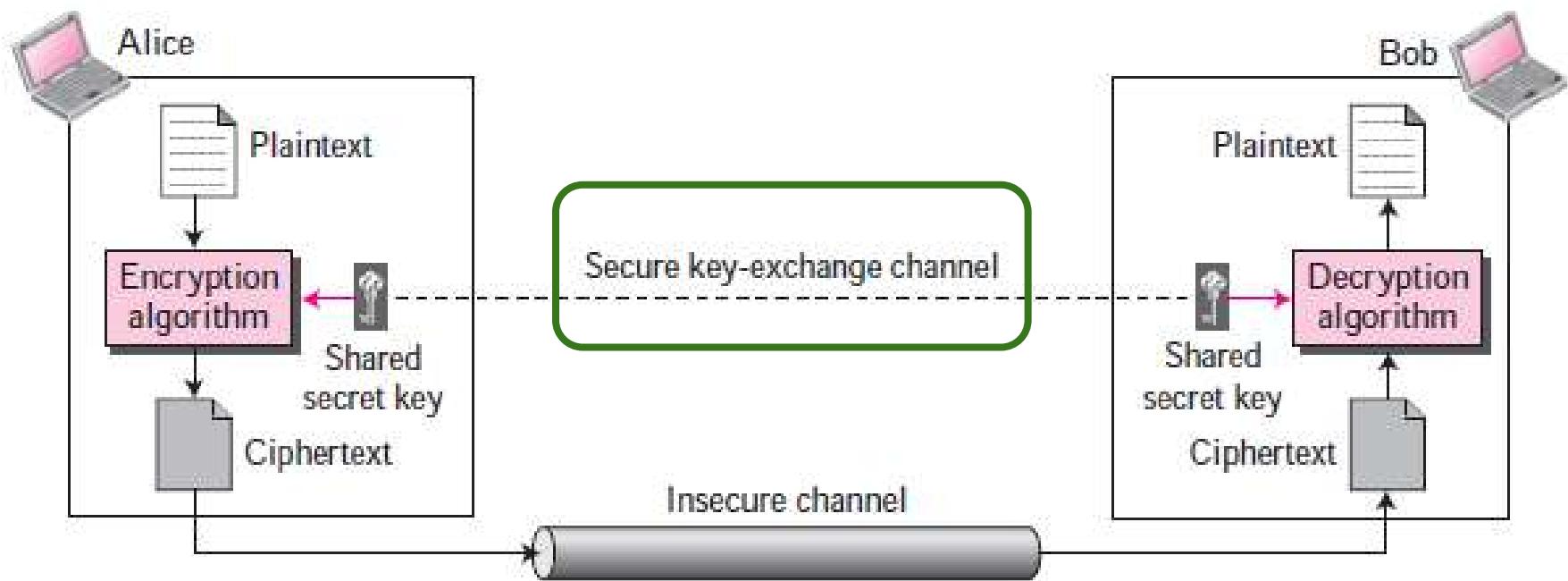
Private key is kept by receiver and Public key is announced to the public





# Symmetric Encryption

# Symmetric Encryption



## Symmetric Encryption (contd...)

Symmetric encryption, also referred to as conventional encryption or single-key encryption, was the only type of encryption in use prior to the development of public key encryption in the 1970s

It remains by far the most widely used from very early days (thousand years ago) to present days as well

## Symmetric Encryption (contd...)

Universal technique for providing confidentiality for the transmitted data

Also referred to as conventional encryption or single-key encryption

Was the only type of encryption in use prior to the introduction of public-key encryption

In many cases, from Julius Caesar to diplomatic, military, and several commercial users, have used symmetric encryption for secret communication

# Symmetric Encryption (contd...)

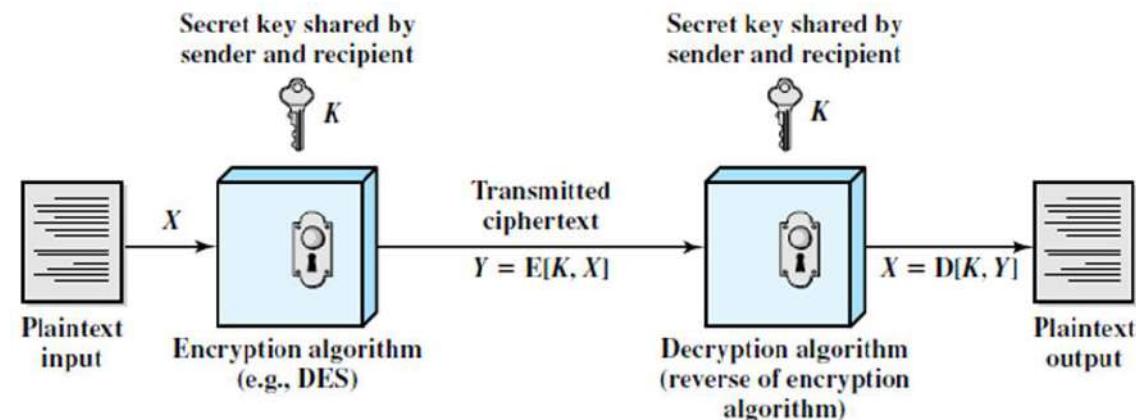
Plaintext

Encryption Algorithm

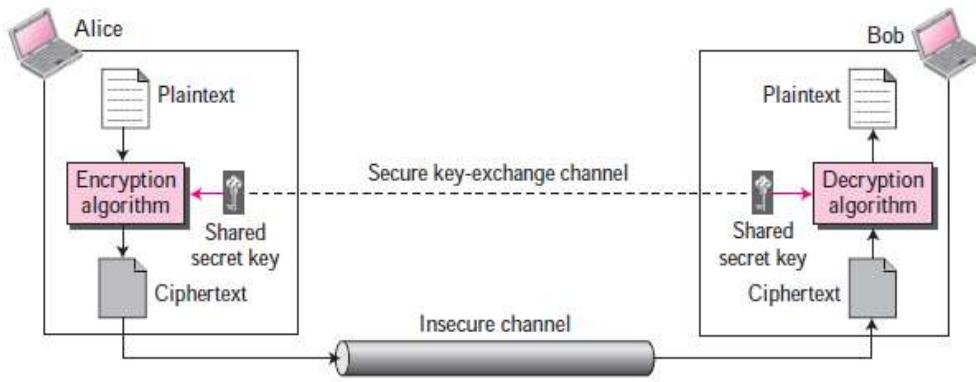
Secret Key

Ciphertext

Decryption Algorithm



# Symmetric Encryption (contd...)



If  $P$  is the plaintext,  $C$  is the ciphertext, and  $K$  is the key, then

- Encryption algorithm  $E_k(x)$  creates the ciphertext from the plaintext
- Decryption algorithm  $D_k(x)$  creates the plaintext from the ciphertext

$E_k(x)$  and  $D_k(x)$  are inverses of each other: they cancel the effect of each other if they are applied one after the other on the same input

Since, we have

- Encryption:  $C = E_k(P)$
  - Decryption:  $P = D_k(C)$
- $$\Rightarrow D_k(E_k(x)) = E_k(D_k(x)) = x$$

# Classical Cryptography (Traditional Ciphers)

## Traditional ciphers

### Substitution ciphers

### Transposition ciphers

Monoalphabetic

Polyalphabetic

A transposition cipher reorders symbols.

A substitution cipher replaces one symbol with another.

# Substitution Ciphers

Any character of plain text from the given fixed set of characters is substituted by some other character from the same set depending on a key

For example with a shift of 1, A would be replaced by B, B would become C, and so on

The simple substitution cipher is a cipher that has been in use for many hundreds of years

The simple substitution cipher offers very little communication security

## Substitution Cipher (contd...)

**Plaintext:** HELLO  
**Ciphertext:** KHOOR

In a substitution cipher, each character of the plain text is substituted by another character to form the cipher text

Let a plaintext “hello” and its corresponding ciphertext using monoalphabetic cipher as “KHOOR”

Similarly, let us see the same plaintext and its corresponding ciphertext using polyalphabetic cipher as “ABNZF”

**Plaintext:** HELLO  
**Ciphertext:** ABNZF

# Monoalphabetic Cipher

A character (or a symbol) in the plaintext is always changed to the same character (or symbol) in the ciphertext regardless of its position in the text, e.g. if a letter A in the plaintext is changed to letter D, every letter A is changed to letter D

The relationship between letters in the plaintext and the ciphertext is one-to-one

The simplest monoalphabetic cipher is the additive cipher (or shift cipher)

# Shift cipher

One of the simplest cipher

For convenient mathematical operations numerical values can be used for each letter

The plaintext, ciphertext, and key are integers in modulo 26

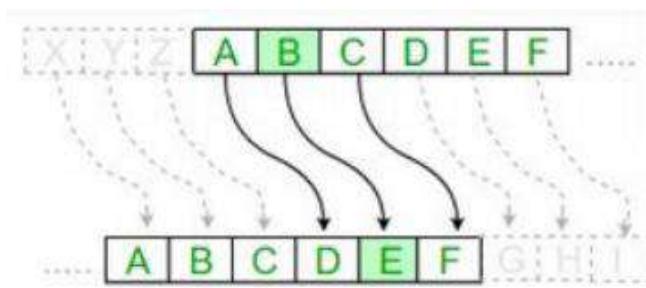
**Julius Caesar** used an additive cipher, with a key of 3 to communicate with his officers  $\Rightarrow$  additive ciphers are sometimes referred to as the **Caesar** cipher

# Caesar Cipher

**Plain Text :** A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

**Cipher Text :** D E F G H I J K L M N O P Q R S T U V W X Y Z A B C

$$C_i = E(P_i) = P_i + 3$$



The message “hello” is enciphered as  $\Rightarrow$  “KHOOR”

# Example - Encryption

Use the additive cipher with key = 15 to encrypt the message “hello”

## Solution

Plaintext: h → 07    Encryption:  $(07 + 15) \text{ mod } 26$     Ciphertext: 22 → W

Plaintext: e → 04    Encryption:  $(04 + 15) \text{ mod } 26$     Ciphertext: 19 → T

Plaintext: l → 11    Encryption:  $(11 + 15) \text{ mod } 26$     Ciphertext: 00 → A

Plaintext: l → 11    Encryption:  $(11 + 15) \text{ mod } 26$     Ciphertext: 00 → A

Plaintext: o → 14    Encryption:  $(14 + 15) \text{ mod } 26$     Ciphertext: 03 → D

The result is “WTAAD”

# Example - Decryption

Use the additive cipher with key = 15 to decrypt the message “WTAAD”

## Solution

Ciphertext: W → 22 Decryption:  $(22 - 15) \text{ mod } 26$  Plaintext: 07 → h

Ciphertext: T → 19 Decryption:  $(19 - 15) \text{ mod } 26$  Plaintext: 04 → e

Ciphertext: A → 00 Decryption:  $(00 - 15) \text{ mod } 26$  Plaintext: 11 → l

Ciphertext: A → 00 Decryption:  $(00 - 15) \text{ mod } 26$  Plaintext: 11 → l

Ciphertext: D → 03 Decryption:  $(03 - 15) \text{ mod } 26$  Plaintext: 14 → o

The result is “hello”

# Caesar Cipher

These kinds of ciphers depend on the secrecy of the system and not on the encryption key

Once the system is known, the encrypted messages can easily be decrypted

Very limited no of possible keys

# Monoalphabetic substitution ciphers

## Example:

|              |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|--------------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Plaintext →  | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z |
| Ciphertext → | N | O | A | T | R | B | E | C | F | U | X | D | Q | G | Y | L | K | H | V | I | J | M | P | Z | S | W |

Let us see the plain text:

**this message is easy to encrypt but not easy to find the key**

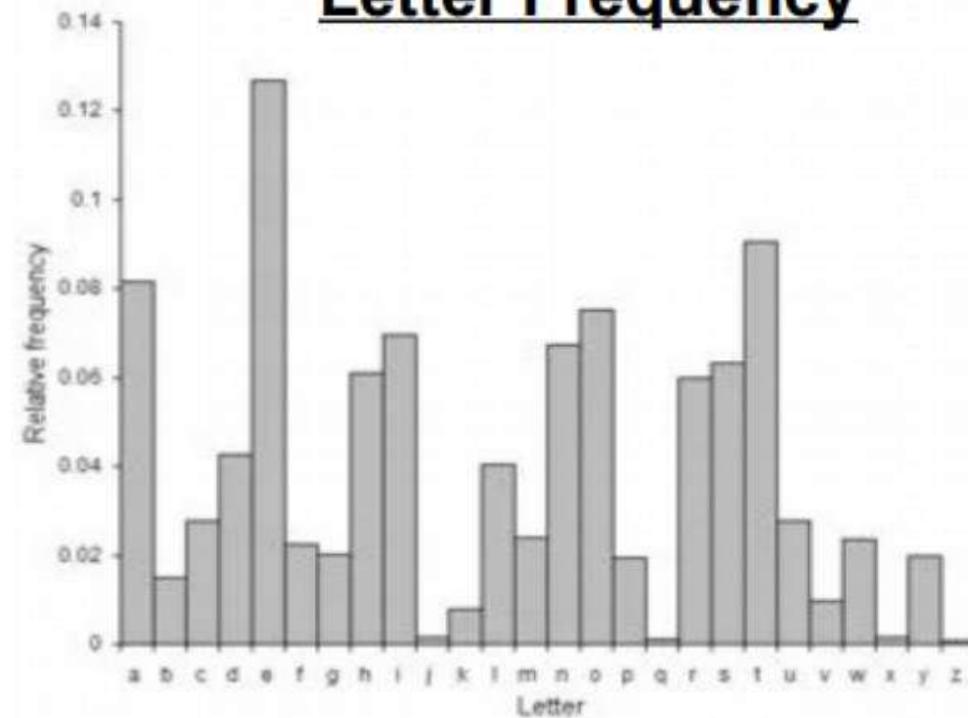
Then the ciphertext becomes:

**ICFVQRVVNEFVRNVSIYRGAGHSLOJICNHTIYBFGTICRXRS**

**Plain Text : A B C D E F G H I J K L M N O P Q R S T U V W X Y Z**

**Cipher Text : K E Y G H I J K L M N O P Q R S T U V W X Y Z A B C**

## Letter Frequency



## Monoalphabetic Substitution

# Polyalphabetic Ciphers

Each occurrence of a single character may have a different substitute

One-to-many relationship between a character in the plaintext to a character in the ciphertext

Can hide the letter frequency of the underlying language

Intruder like Eve cannot use single-letter frequency statistics to break

Ciphertext character is dependent on both the corresponding plaintext character and the position of the plaintext character in the message

# Polyalphabetic Substitutions Example

## Table for Odd Positions

**Plain Text :** A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

**Cipher Text :** A D G J N O S V Y B E H K N Q T W Z C F I L O R U X

## Table for Even Positions

**Plain Text :** A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

**Cipher Text :** N S X C H M R W B G I Q V A F K P U Z E J O T Y D I

Plain Text: ssibl

Cipher Text: CZYSH

# Vigenere cipher

Invented by the French cryptographer Blaise de Vigenère in the 16th century to encrypt and decrypt messages

Easy to understand and implement and has a long history of usage

Uses simple form of polyalphabetic substitution cipher ⇒ uses multiple cipher alphabets to encrypt the plaintext

For many years it was considered as literally “the unbreakable cipher”

Gained a reputation for being very strong because resisted all attempts to break around up to **three centuries**

# Vigenere cipher

A sequence of text is used as a key

The encryption key was repeated multiple times spanning the entire message, and then the cipher text was produced by adding the message character with the key character modulo 26

$k = \boxed{\text{CRYPTOCRYPTOCRYPT}}$

+ mod 26

$m = \text{HAVEANICEDAYTODAY}$   
 $c = \text{KSUUUCLUDTUNWGQCS}$

## Encryption

The plaintext( $P$ ) and key( $K$ ) are added modulo 26

$$E_i = (P_i + K_i) \bmod 26$$

## Decryption

$$D_i = (E_i - K_i) \bmod 26$$



## Plaintext

|   | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| B | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A |
| C | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B |
| D | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C |
| E | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D |
| F | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E |
| G | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F |
| H | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G |
| I | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H |
| J | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I |
| K | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J |
| L | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K |
| M | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L |
| N | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M |
| O | O | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N |
| P | P | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
| Q | Q | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P |
| R | R | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q |
| S | S | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R |
| T | T | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S |
| U | U | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T |
| V | V | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U |
| W | W | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V |
| X | X | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W |
| Y | Y | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X |
| Z | Z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y |

## Vigenere Cipher Table

# Example

Encryption

| plaintext                        | m  | a | t  | h  | i  | s  | f  | u  | n  |
|----------------------------------|----|---|----|----|----|----|----|----|----|
| $m_i$                            | 12 | 0 | 19 | 7  | 8  | 18 | 5  | 20 | 13 |
| keyword                          | j  | i | m  | j  | i  | m  | j  | i  | m  |
| $k_i$                            | 9  | 8 | 12 | 9  | 8  | 12 | 9  | 8  | 12 |
| $c_i \equiv m_i + k_i \pmod{26}$ | 21 | 8 | 5  | 16 | 16 | 4  | 14 | 2  | 25 |

Ciphertext: vifqqeocz

# Example

## Decryption

| ciphertext                       | v  | i  | f  | q  | q  | e  | o  | c  | z  |
|----------------------------------|----|----|----|----|----|----|----|----|----|
| $c_i$                            | 21 | 8  | 5  | 16 | 16 | 4  | 14 | 2  | 25 |
| $d_i$                            | 17 | 18 | 14 | 17 | 18 | 14 | 17 | 18 | 14 |
| $m_i \equiv c_i + d_i \pmod{26}$ | 12 | 0  | 19 | 7  | 8  | 18 | 5  | 20 | 13 |
| plaintext                        | m  | a  | t  | h  | i  | s  | f  | u  | n  |

**Plaintext:** math is fun

# Vigenere cipher

The first cipher brought the idea of introducing encryption keys

Comparing this to Caesar cipher, the secrecy of the message depends on the secrecy of the encryption key, rather than the secrecy of the system ⇒ Kerckhoff's principle

Not so secure

# Transposition Ciphers

# Transposition Cipher

Also known as a permutation cipher

A method that scrambles the positions of characters (transposition) without changing the characters themselves

Reorder units of plaintext (typically characters or groups of characters) according to a regular system to produce a ciphertext which is a permutation of the plaintext

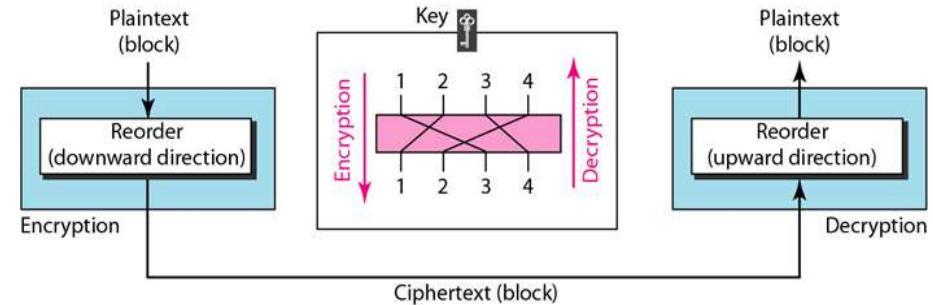
Differ from substitution ciphers, which do not change the position of units of plaintext but instead change the units themselves

# Transposition Cipher

The substitution ciphers are based on the replacing plaintext symbol with a ciphertext symbol

Here the mapping is achieved by performing some sort of permutation on the plaintext letters ⇒ referred to as a transposition cipher

# Transposition Cipher

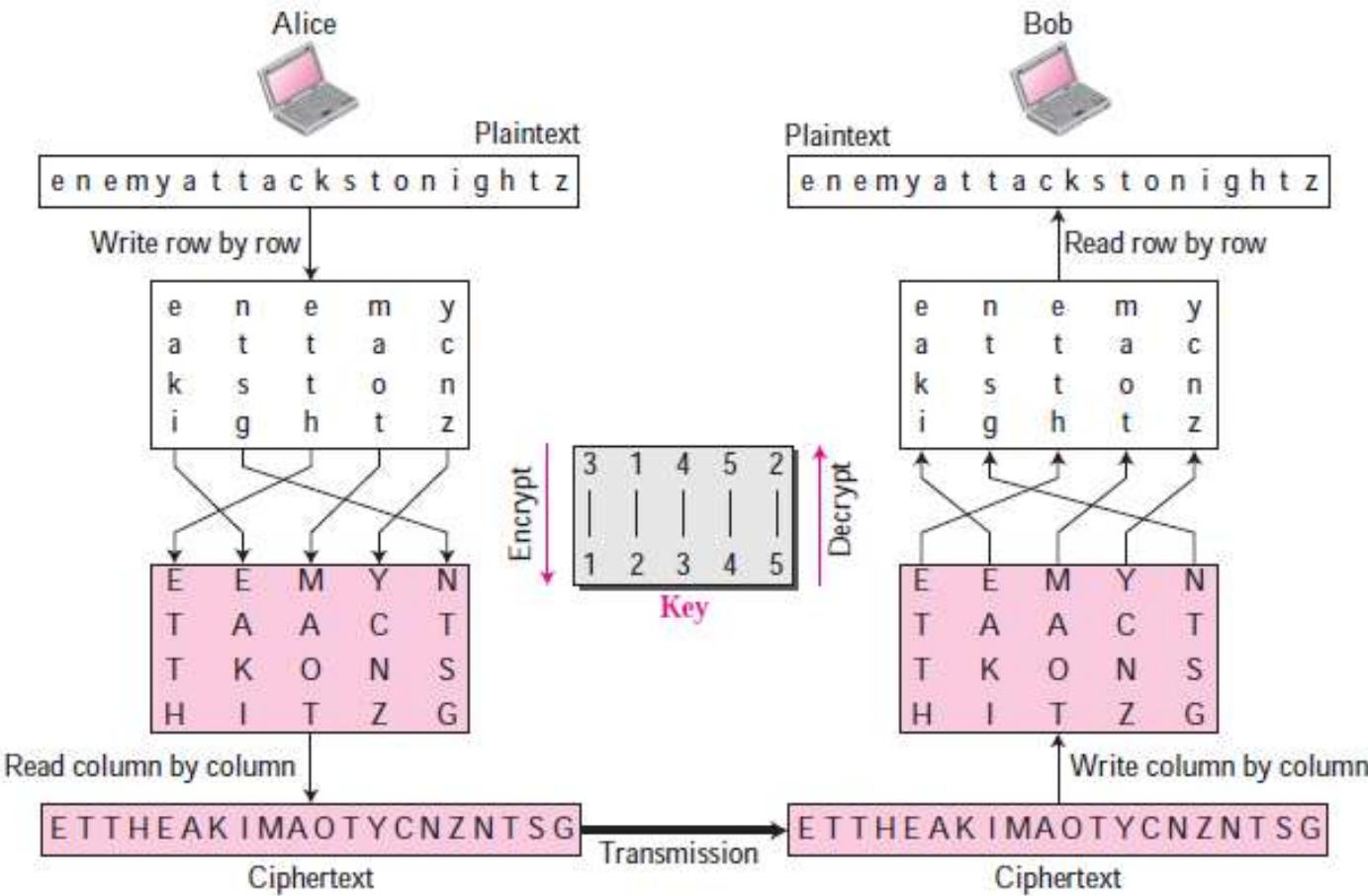


There is no substitution of characters instead their locations changed

It reorders (permutes) symbols in a block of symbols

Key is the mapping between the position of the symbols in plaintext and ciphertext

# Transposition Cipher Example



# Transposition Cipher

A pure transposition cipher is easily recognized because it has the same letter frequencies as the original plaintext

For the type of columnar transposition just shown, cryptanalysis is fairly straightforward and involves laying out the ciphertext in a matrix and playing around with column positions

The transposition cipher can be made significantly more secure by performing more than one stage of transposition

The result become more complex permutation that is not easily reconstructed

# Cryptography Attacks

# Cryptography Attacks

Malicious attempts to compromise the security of cryptographic systems

Aims to exploit vulnerabilities and gain unauthorised access to sensitive information

Attacks cause threats to the confidentiality, integrity, and availability of encrypted data

Attackers employ various strategies to breach cryptographic defences, targeting weaknesses in algorithms, keys, or implementation processes

# Types of cryptanalytic attacks

**Cipher text only** – A copy of cipher text alone is known to the cryptanalyst

**Known plaintext** – The cryptanalyst has a copy of the cipher text and the corresponding plaintext

**Chosen plaintext** – The cryptanalysts can encrypt a large number of suitably chosen plaintexts and try to use the resulting cipher texts to deduce the key

**Chosen cipher text** – The cryptanalyst decrypt several string of symbols, and tries to use the results to deduce the key

# Brute Force

Using a systematic method of trying every possible key until the correct one is found

Involves an exhaustive trial-and-error approach, making it time-consuming but effective if encryption keys are weak or easily guessable

Can target various cryptographic systems, including passwords, encryption keys, and digital signatures

To mitigate the risk of brute force attacks, use of strong and complex encryption keys is essential

# Brute Force

Longer and complex keys exponentially increase the time and computational resources for attackers to succeed

The effectiveness of cryptographic defences relies on the resilience against brute force attempts, emphasising the importance of robust key management practices in the digital security landscape

# Preventing Cryptography Attacks

Cryptographic attacks pose a substantial threat to the security of sensitive information, necessitating robust preventive measures to safeguard against potential breaches

Implementing effective strategies involves a multifaceted approach

# Preventing Cryptography Attacks

Use strong encryption technique and unique keys for encryption

Regularly update the cryptographic algorithms and protocols to ensure they are not obsolete

Secure the keys

Ensure that the cryptographic system is implemented correctly

Regularly test the system for vulnerabilities

# Requirements for Security

Strong encryption algorithm

- Even algorithm is known, unable to decrypt without key
- Even if plaintexts & ciphertexts available, unable to find the key

Sender and receiver must **share secret key securely**

Once key is known, all communication using this key is readable

Now it can be broken within few minutes due to the processing power of current computers

| Key Size (bits)             | Number of Alternative Keys     | Time Required at 1 Decryption/ $\mu$ s                | Time Required at $10^6$ Decryptions/ $\mu$ s |
|-----------------------------|--------------------------------|---|--|
| 32                          | $2^{32} = 4.3 \times 10^9$     | $2^{31} \mu$ s = 35.8 minutes                         | 2.15 milliseconds                            |
| 56                          | $2^{56} = 7.2 \times 10^{16}$  | $2^{55} \mu$ s = 1142 years                           | 10.01 hours                                  |
| 128                         | $2^{128} = 3.4 \times 10^{38}$ | $2^{127} \mu$ s = $5.4 \times 10^{24}$ years          | $5.4 \times 10^{18}$ years                   |
| 168                         | $2^{168} = 3.7 \times 10^{50}$ | $2^{167} \mu$ s = $5.9 \times 10^{36}$ years          | $5.9 \times 10^{30}$ years                   |
| 26 characters (permutation) | $26! = 4 \times 10^{26}$       | $2 \times 10^{26} \mu$ s = $6.4 \times 10^{12}$ years | $6.4 \times 10^6$ years                      |

Table: Average Time Required for Exhaustive Key Search

# Goals of Cryptography

Services provided by cryptography

- ❖ Confidentiality (secrecy)
- ❖ Integrity (anti-tampering)
- ❖ Authentication
- ❖ Non-repudiation

# Distributed and Edge Computing

# Distributed Systems Security

Sharad K. Ghimire

Department of Electronics and Computer Engineering  
Pulchowk Campus  
Institute of Engineering  
Tribhuvan University

# Contents

Overview of cryptography and data privacy

Modern Ciphers

Confusion and Diffusion

Block and Stream Ciphers

Block Ciphers: DES, AES

Block Cipher modes of operation

# Modern Ciphers

# Confusion and Diffusion

Claude Shannon defined these properties that a good cryptosystem should have to hinder statistical analysis

# Confusion

Confusion means that the key does not relate in a simple way to the cipher-text, i.e., **each character of the ciphertext** should depend on **several parts of the key**

Defines making the relationship between the key and the cipher as difficult and as possible

Specifies that the ciphertext provides no clue about the plaintext

# Confusion

The relationship between the data of the cipher text and the value of the encryption has to remain as difficult as applicable

The main goal of confusion is to create it very complex to find the key even if one has most of the plaintext-ciphertext pairs produced with the similar key

Each bit of the Ciphertext should be based on the complete key and in several ways on multiple bits of the key, changing one bit of the key must change the Ciphertext completely

# Diffusion

If we change a character of plaintext, then several characters of the ciphertext should change, similarly, if we change a character of the ciphertext, then several characters of the plaintext should change

Diffusion can define to the property that the repetition in the statistics of the plaintext is “dissipated” in the statistics of the Ciphertext

The output bits should be based on the input bits in a difficult way so that in case one bit of the plaintext is modified, thus the cipher-text should change completely in an unstable or pseudo-random manner

# Diffusion

The statistical mechanism of the plaintext is used up into a high-range data of the ciphertext ⇒ it is achieved by having each plaintext digit influence the value of some ciphertext digits

Similarly each ciphertext digit be influenced by some plaintext digits

The frequency statistics of characters in the plaintext are diffused over several characters in the ciphertext

# Confusion and Diffusion

Confusion and diffusion are both cryptographic approaches

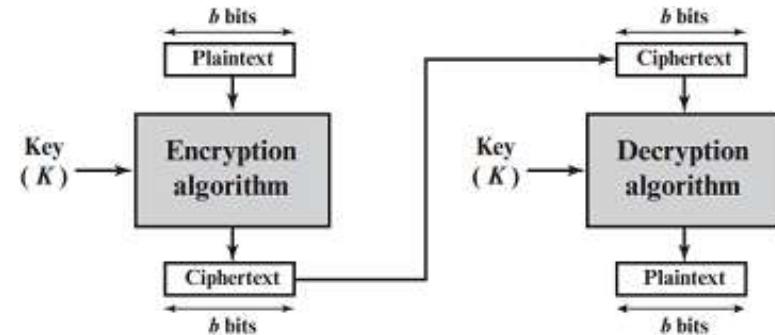
**Confusion**:- The relationship between the ciphertext statistics and the encryption key value ⇒ as difficult as possible

**Diffusion**:- Attempt to disguise the statistical nature of the plaintext by spreading the influence of each individual plaintext digit over a large number of ciphertext digits

Some ciphers can be confusion-only or diffusion-only, but any "reasonable" block cipher uses both confusion and diffusion

# Block & Stream Ciphers

# Block Cipher



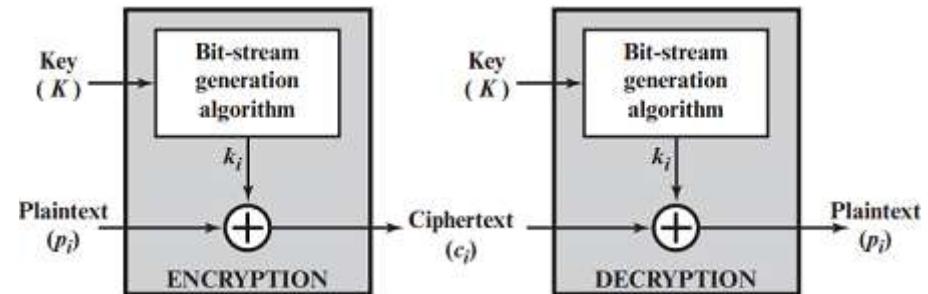
A group of plaintext symbols of size  $m$  ( $m > 1$ ) are encrypted together creating a group of ciphertext of the same size

A single key is used to encrypt the whole block even if the key is made of multiple values

Most modern symmetric encryption algorithms are block ciphers

A block of plaintext is treated as a whole and used to produce a ciphertext block of equal length  $\Rightarrow$  typical block size of 64 or 128 bits

# Stream Cipher



Convert each symbol of plaintext directly into a symbol of cipher-text

Encryption and decryption are done one symbol (such as a character or a bit) at a time  $\Rightarrow$  needs a plaintext stream, a ciphertext stream, and a key stream

Encrypts a digital data stream one bit or one byte at a time

Call the plaintext stream P, the ciphertext stream C, and the key stream K

$$P = P_1 P_2 P_3, \dots$$

$$C = C_1 C_2 C_3, \dots$$

$$K = (k_1, k_2, k_3, \dots)$$

$$C_1 = E_{k1}(P_1)$$

$$C_2 = E_{k2}(P_2)$$

$$C_3 = E_{k3}(P_3) \dots$$

# Block & Stream Cipher

## **Block Cipher:**

A block cipher is a deterministic algorithm operating on fixed length groups of bits called blocks

The size of block is defined by the algorithm, such as 64 bits, 128 bits etc.

## **Stream Cipher:**

Each plaintext message is encrypted one at a time with the corresponding digit of the keystream, to give a ciphertext stream

# Stream Cipher

# Stream Cipher

Encrypts plaintext of one byte or one bit or similar unit at a time

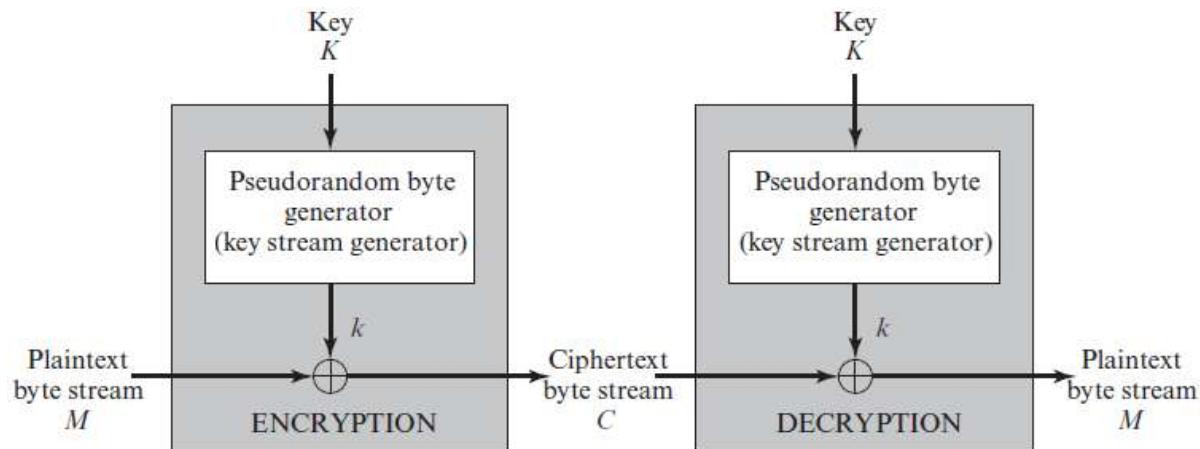
More efficient for real-time processing

Several stream ciphers have been used in different protocols during the last few decades

E.g. RC4

In stream cipher, a key is input to a pseudorandom bit generator that produces a stream of 8-bit numbers that are apparently random

The output of the generator, called a keystream, is combined one byte at a time with the plaintext stream using the bitwise exclusive-OR (XOR) operation



Stream Cipher Diagram

## Example

E.g., let the next byte generated by the generator is 01101100 and the next plaintext byte is 11001100, then the resulting ciphertext byte is

11001100 plaintext

$\oplus$  01101100 key stream

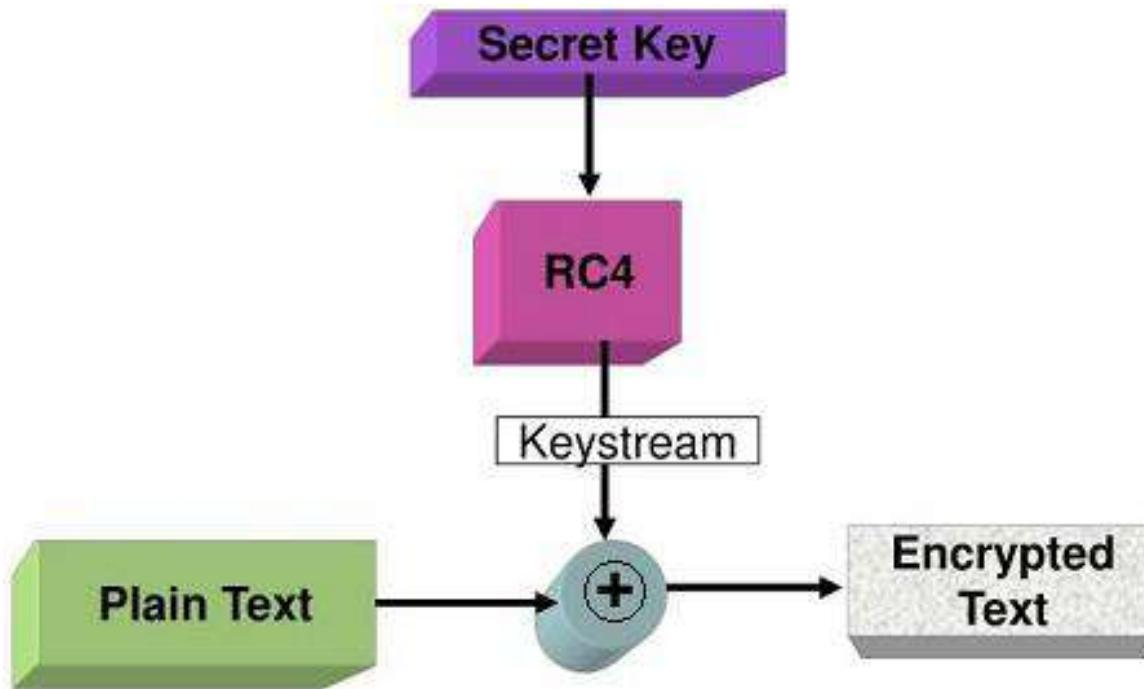
10100000 ciphertext

Decryption requires the use of the same pseudorandom sequence as:

10100000 ciphertext

$\oplus$  01101100 key stream

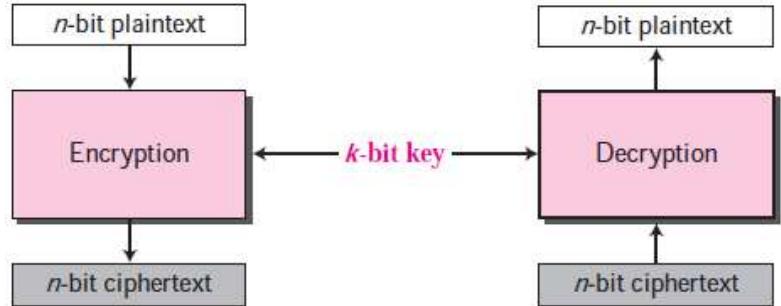
11001100 plaintext



Block Diagram of RC4 Cipher

# Block Ciphers

# Block Ciphers



Most common symmetric algorithms are block ciphers

A block cipher encrypts block of predefined number bits and generates same number of bits as ciphertext

Made up of fixed number of bits in a block (typically 64 bits or 128 bits)

Use a key of fixed sized typically 56 bits, 128 bits, 192 bits or 256 bits long

Block ciphers are frequently used to encrypt large amounts of data into data blocks

# Modern Block Ciphers

Most important symmetric algorithms, using block ciphers:

- Data Encryption Standard (DES) and Triple-DES (3DES)
- Advanced Encryption Standard (AES)

# Data Encryption Standard (DES)

# Data Encryption Standard (DES)

Designed by IBM

Adopted by US government as a standard encryption

Dominant encryption since its introduction in 1977

64 bit plaintext blocks with 64 bit ciphertext

56 bit key

Broken in 1998 by Electronic Frontier Foundation

# DES

Based on the Feistel block cipher, called LUCIFER, developed in 1971 by IBM cryptography researcher Horst Feistel

DES was adopted by the National Bureau of Standards (now called the National Institute of Standards and Technology) as Federal Information Processing Standard

Uses 16 rounds of the Feistel structure, using a different key for each round

Over the years, DES became the dominant symmetric encryption algorithm

Plaintext 64, ciphertext 64-bit and key size of 56-bits

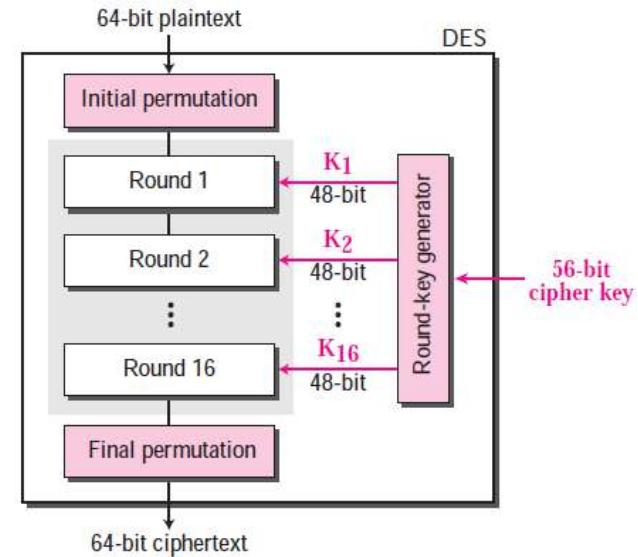
# DES Structure

DES is based on the Feistel Cipher

DES functions

- Round function
- Key schedule
- Additional processing – Initial and final permutation

# DES Structure



**Initial & Final Permutations:** Predefined permutation of 64 bits data

**Rounds:** DES uses 16 rounds, and each round is a Feistel cipher

Each round has a different operations: mixing and swapping

Round-key generator generates different keys for each round from given 56 bit key

# DES Properties

Just only one bit difference in plaintext causes significant differences in the bits of ciphertext

**Avalanche Effect:** A small change in the plaintext (or key) creates a significant change in the ciphertext  $\Rightarrow$  DES has been proved to be with this property

Plaintext: 0000000000000000

Key: 22234512987ABB23

Ciphertext: 4789FD476E82A5F1

Plaintext: 0000000000000001

Key: 22234512987ABB23

Ciphertext: 0A4ED5C15A63FEA3

Each bit of the ciphertext depends on many bits on the plaintext

# DES strength/weakness

The strength of DES lies on following facts:

**The nature of algorithm:** Cryptanalyst can perform cryptanalysis by exploiting the characteristic of DES algorithm but no one has succeeded in finding out the weakness

**The size of key:** 56-bit key is used in encryption, there are  $2^{56}$  possible keys. A brute-force attack on such number of keys is impractical at that time in past, but it is not considered as secure now due to increased processing power of computers ⇒ Key size is the main weakness of DES

# DES

Broken in 1998 by Electronic Frontier Foundation

- Special purpose US\$250,000 machine
- With detailed published description
- Less than three days

Only 56-bit key is considered as not secure

DES is considered as worthless now

# How to Increase the Security?

Modification/Amendment of current algorithm

OR

Switch to a newer suitable algorithm

# Multiple DES

As the major criticism of DES regards its key length ⇒ with available technology and the possibility of parallel processing, a brute-force attack on DES is feasible

In the approach of using multiple (cascaded) instances of DES with multiple keys; does not require an investment in new software and hardware ⇒ one approach is use double DES (**2DES**)

2DES was not found to be effective

# Triple DES

To improve the security of DES, triple DES (3DES) was proposed  
Uses three stages of DES for encryption and decryption

# Triple DES

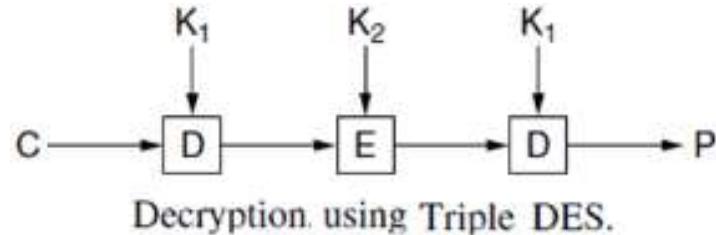
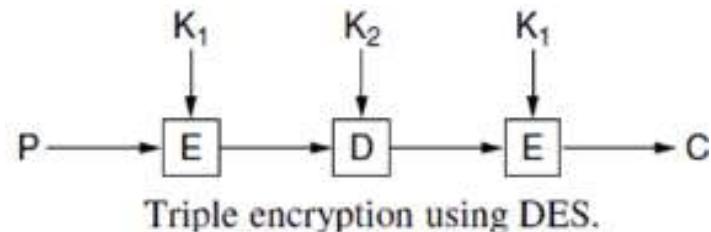
Two versions: using **2 keys or 3 keys**

3 executions of DEA algorithm

Sequence of E - D - E in encryption

Sequence of D - E - D in decryption

Effective key length 112 or 168 bit



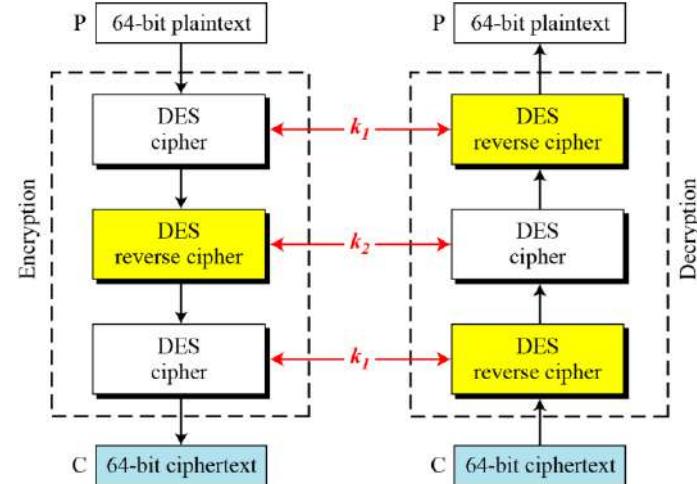
# Triple DES with 2 Keys

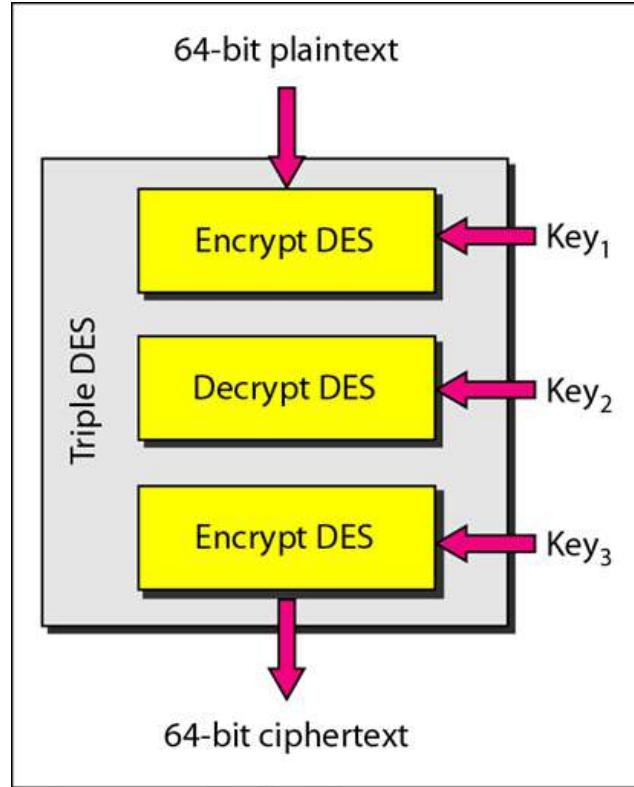
Only two keys:  $k_1$  and  $k_2$

The first and the third stages use  $k_1$ ; the second stage uses  $k_2$

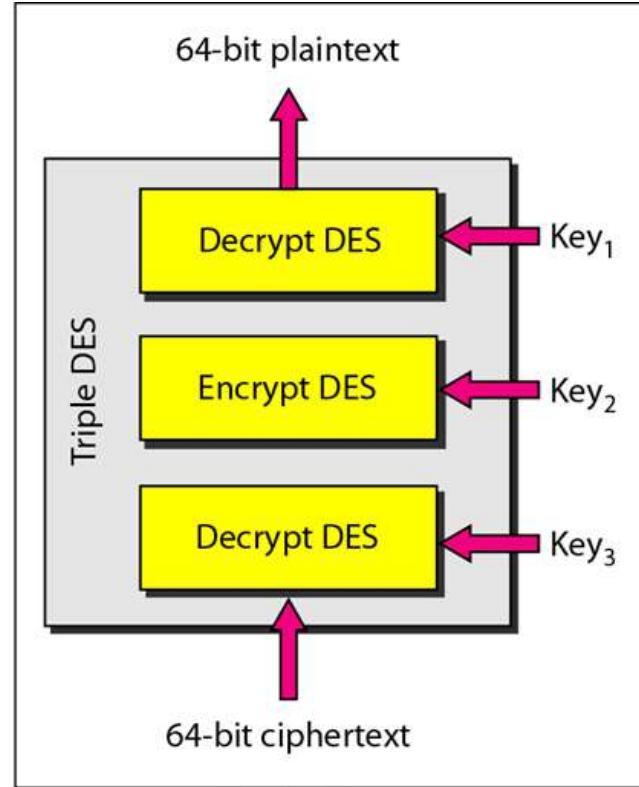
To make it compatible with single DES, the middle stage uses decryption in the encryption site and encryption in the decryption site

A message encrypted with single DES with key  $k$  can be decrypted with triple DES if  $k_1 = k_2 = k \Rightarrow$  much stronger than double DES





a. Encryption Triple DES



b. Decryption Triple DES

## Triple DES with Three Keys

# 3DES Analysis

Breaking 3DES is not practical ⇒ considered as secure

But it is very slow due to three consecutive blocks of DES (typically in software implementation) ⇒ considered as inefficient

Block size (64 bit) is too small

3DES ⇒ Intermediate solution to address the need of high security

⇒ Need of some efficient algorithm

# Advanced Encryption Standard (AES)

# AES

AES is a variant of the Rijndael block cipher developed by two Belgian cryptographers, Joan Daemen and Vincent Rijmen

A symmetric block cipher with the block length of 128 bits

The most common / widely used block cipher in current use

Introduced by NIST in 2001

Security strength equal to or better than 3DES

Significantly improved efficiency

Allows for different key lengths: 128, 192, or 256 bits

Encryption consists of 10 rounds of processing for 128-bit keys, 12 rounds for 192-bit keys, and 14 rounds for 256-bit keys

# Why AES ?

Developed to provide the highest level of security for the most sensitive data

Uses large data block for encryption i.e. 128 bits data

AES combines speed and security properly

AES is Unbreakable by even Brute Force (till now)

On the basis of required security level either one of 128 bits or 192 bits or 256 bits key can be used

# Open Call for AES

On January 2, 1997, NIST announced that they wished to choose a successor to DES ⇒ an initiation of an effort to develop the new algorithm

NIST asked for input from interested parties on how the successor should be chosen

NIST made a formal call for algorithms on September 12, 1997

# Advanced Encryption Standard

NIST (National Institute of Standards and Technology) issued call for proposals for the newer algorithm (AES) in 1997

The requirements:

- The algorithm must be a symmetric block cipher
- The full design must be public
- Key lengths of 128, 192, and 256 bits must be supported
- Both software and hardware implementations must be possible
- The algorithm must be public or licensed on nondiscriminatory terms

# AES Submission

21 algorithms submitted

After the First AES Candidate Conference, NIST announced that 15 out of 21 received algorithms had met the requirements and been selected as the first candidates (August 1998)

Fifteen designs were from several countries: CAST-256, CRYPTON, DEAL, DFC, E2, FROG, HPC, LOKI97, MAGENTA, MARS, RC6, Rijndael, SAFER+, Serpent, and Twofish

# AES Evaluation

The advantages and disadvantages were investigated by cryptographers

They were assessed not only on security, but also on performance in a variety of settings (PCs of various architectures, smart cards, hardware implementations) and on their feasibility in limited environments (smart cards with very limited memory, low gate count implementations, FPGAs)

After the Second AES Candidate Conference, NIST announced that 5 out of 15 candidates: **MARS, RC6, Rijndael, Serpent, and Twofish**

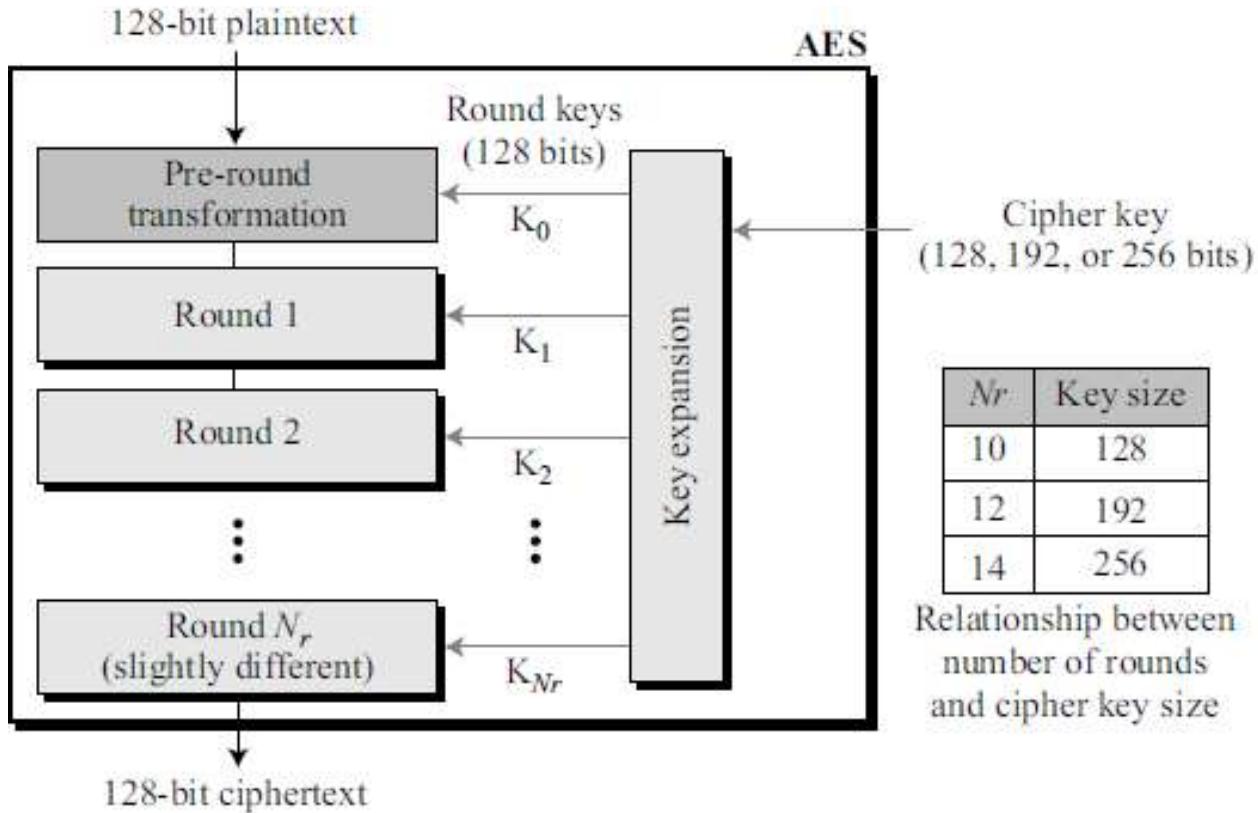
# AES Selection

On October 2, 2000, NIST announced that **Rijndael** had been selected as the proposed AES

In February 2001, NIST announced that a draft of the Federal Information Processing Standard (FIPS) was available for public review and comment

On November 26, 2001, NIST announced that AES was approved as FIPS 197

# AES Structure



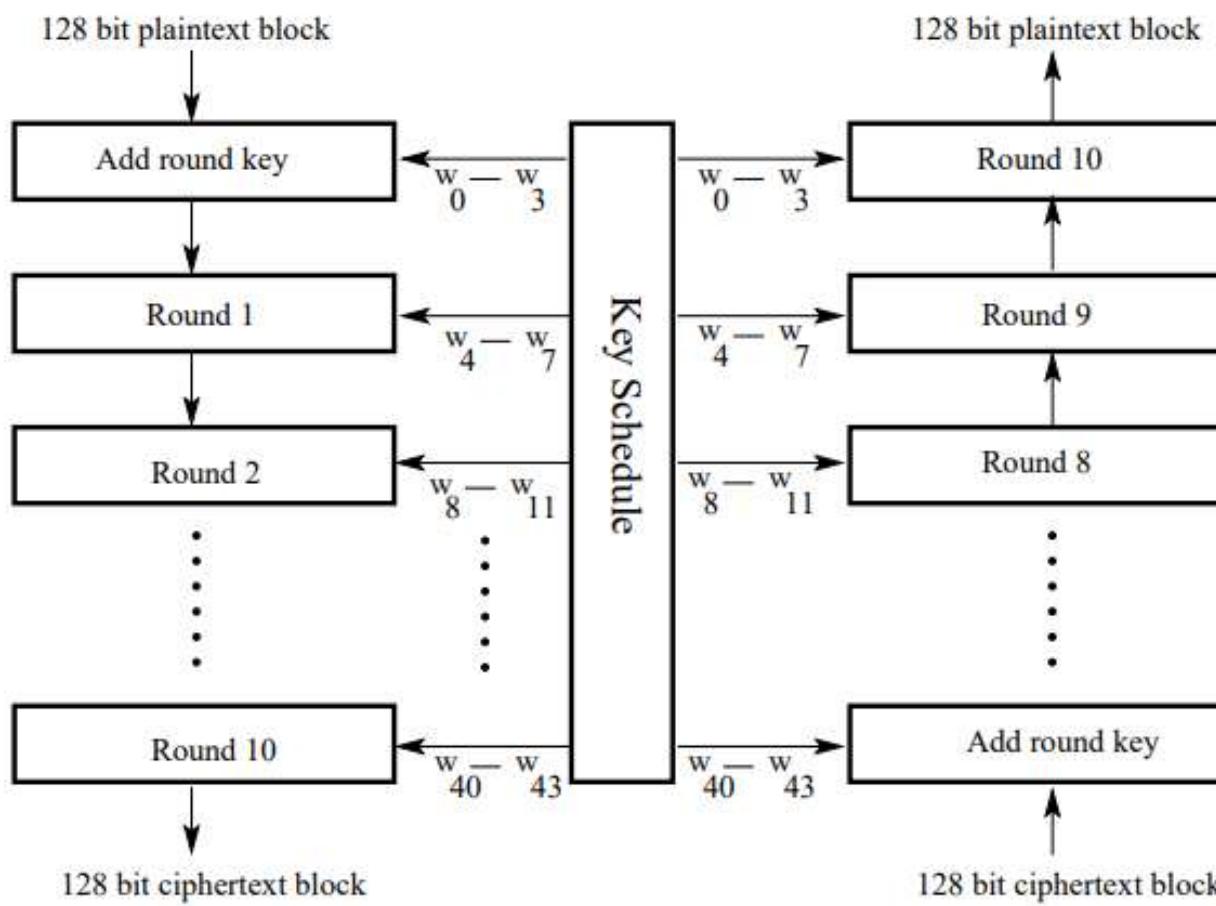
## General design of AES encryption cipher

# AES Structure

AES is a non-Feistel cipher that encrypts and decrypts a data block of 128 bits  
⇒ uses 10, 12, or 14 rounds based on key size that can be 128, 192, or 256 bits

The encryption algorithm (called cipher) and the decryption algorithm (called inverse cipher) is similar, but the round keys are applied in the reverse order

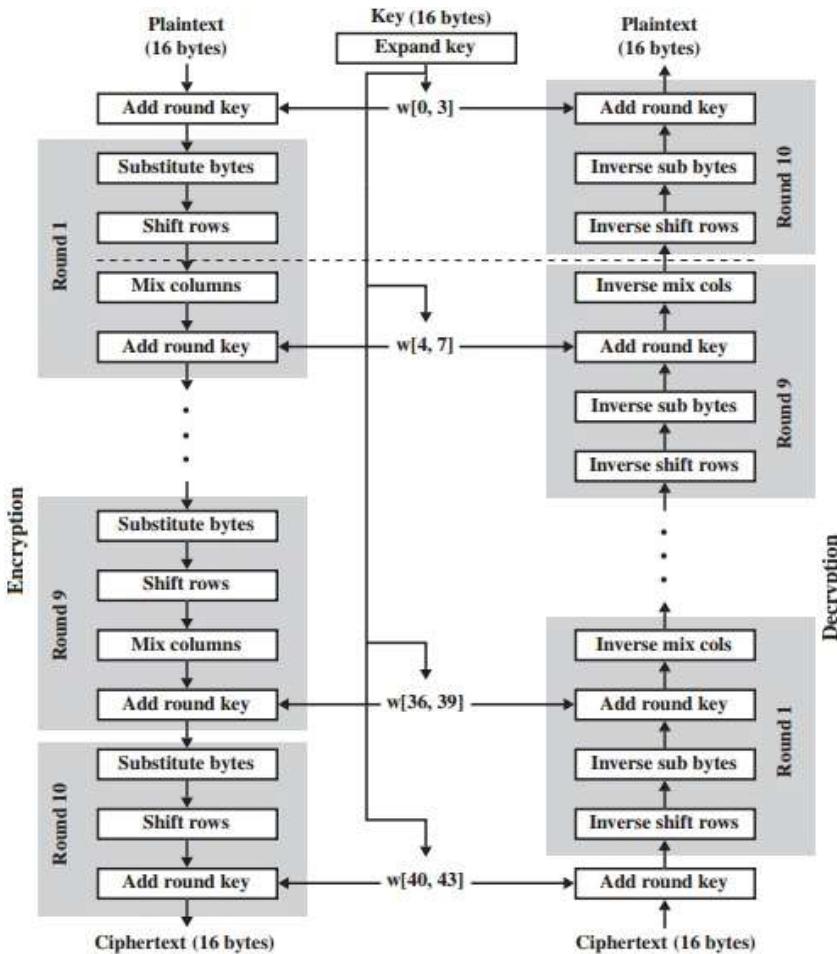
Though the key size can vary i.e., 128, 192 or 256 bits, the round keys, which are created by the key-expansion algorithm are always 128 bits, the same size as the plaintext or ciphertext block



## AES Encryption

## AES Decryption

# AES Encryption and Decryption



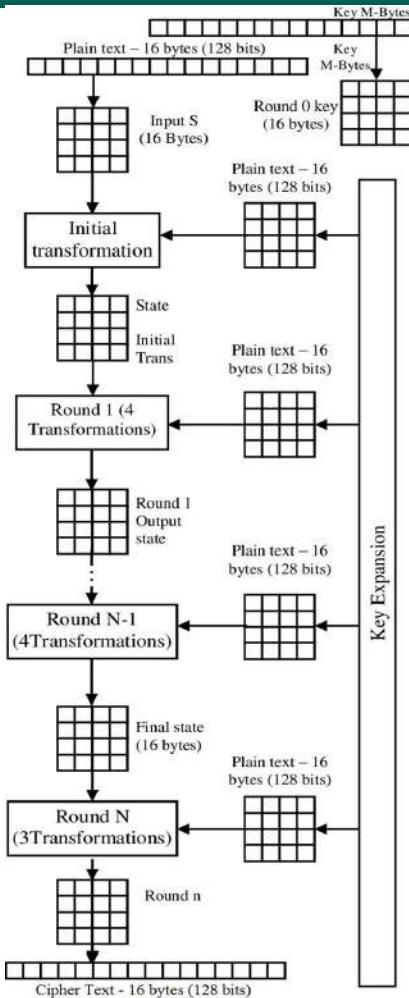
AES has defined three versions, with 10, 12, and 14 rounds.

Each version uses a different cipher key size (128, 192, or 256), but the round keys are always 128 bits.

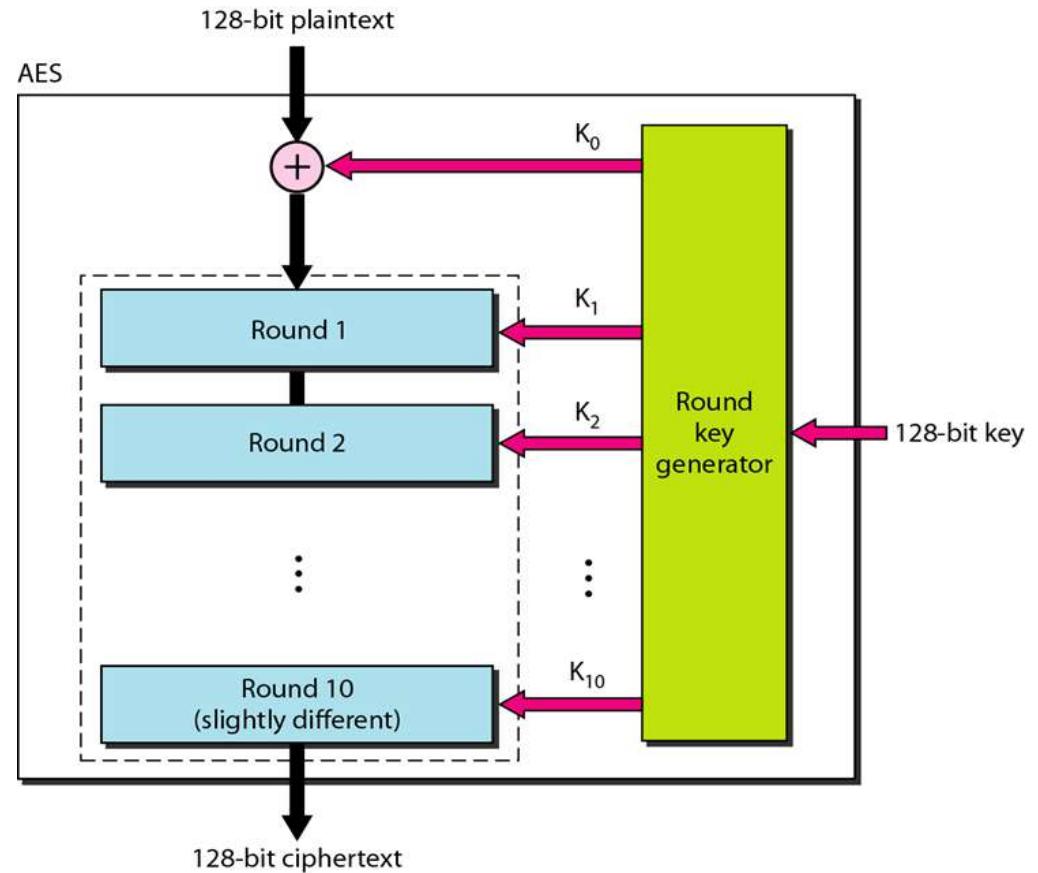
# Transformations

AES uses four types of transformations: substitution, permutation, mixing, and key-adding

- Substitute bytes
- Shift rows  $\Rightarrow$  permutation
- Mix columns
- Add round key



## AES Encryption Process



Structure of AES Cipher with 128 bits key (having 10 rounds only)

# Analysis of AES

Security: AES was designed after DES ⇒ most of the known attacks on DES were already tested on AES; none of them has broken the security of AES so far

AES ⇒ more secure than DES due to the larger-size key (128, 192, 256 bits)

In addition, AES provides two other versions with longer cipher keys

The lack of weak keys is another advantage of AES over DES

### Average Time Required for Exhaustive Key Search

| Key Size (bits)             | Cipher         | Number of Alternative Keys           | Time Required at $10^9$ Decryptions/s                         | Time Required at $10^{13}$ Decryptions/s |
|-----------------------------|----------------|--------------------------------------|---|--|
| 56                          | DES            | $2^{56} \approx 7.2 \times 10^{16}$  | $2^{55} \text{ ns} = 1.125 \text{ years}$                     | 1 hour                                   |
| 128                         | AES            | $2^{128} \approx 3.4 \times 10^{38}$ | $2^{127} \text{ ns} = 5.3 \times 10^{21} \text{ years}$       | $5.3 \times 10^{17} \text{ years}$       |
| 168                         | Triple DES     | $2^{168} \approx 3.7 \times 10^{50}$ | $2^{167} \text{ ns} = 5.8 \times 10^{33} \text{ years}$       | $5.8 \times 10^{29} \text{ years}$       |
| 192                         | AES            | $2^{192} \approx 6.3 \times 10^{57}$ | $2^{191} \text{ ns} = 9.8 \times 10^{40} \text{ years}$       | $9.8 \times 10^{36} \text{ years}$       |
| 256                         | AES            | $2^{256} \approx 1.2 \times 10^{77}$ | $2^{255} \text{ ns} = 1.8 \times 10^{60} \text{ years}$       | $1.8 \times 10^{56} \text{ years}$       |
| 26 characters (permutation) | Monoalphabetic | $26! = 4 \times 10^{26}$             | $2 \times 10^{26} \text{ ns} = 6.3 \times 10^9 \text{ years}$ | $6.3 \times 10^6 \text{ years}$          |

# Ciphering a Long Sequence of Message

# Ciphering a Long Sequence of Message

In real life applications, the text to be enciphered is of variable size and normally much larger than 64 or 128 bits ⇒ **stream cipher** can be used

Block ciphers are strong for encryption ⇒ they are designed to encrypt fixed sized block of message

How to encrypt **a long sequence of message** by using **block ciphers**?

While transmitting a long messages there are some additional issues for secure communication ⇒ different modes of operation using block cipher

# Block cipher modes of operation

## Cipher Modes

- Electronic Code Book Mode (ECB)
- Cipher Block Chaining Mode (CBC)
- Cipher Feedback Mode (CFB)
- Output Feedback Mode (OFB)
- Counter Mode (CTR)

# Electronic Code Book Mode (ECB)

# Electronic Code Book Mode (ECB)

The simplest mode  $\Rightarrow$  sequence of message  $\Rightarrow$  multiple b-bit blocks

Plaintext is handled one block at a time and each block of plaintext is encrypted using the same key

The term codebook is used because, for a given key, there is a unique ciphertext for every b-bit block of plaintext

We can imagine a gigantic codebook in which there is an entry for every possible b-bit plaintext pattern showing its corresponding ciphertext

# Electronic Code Book Mode (ECB)

For a message longer than  $b$  bits, the message is broken into  $b$ -bit blocks, padding the last block if necessary

Decryption is performed one block at a time, using the same key

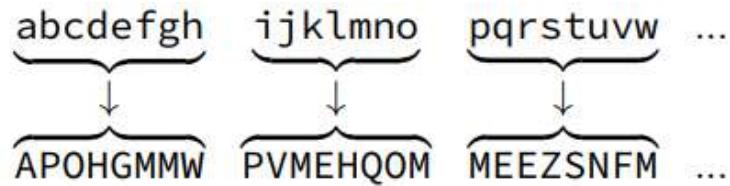
The plaintext (padded as necessary) consists of a sequence of  $b$ -bit blocks,  $P_1, P_2, \dots, P_N$ ; the corresponding sequence of ciphertext blocks is  $C_1, C_2, \dots, C_N$

ECB mode is used as:

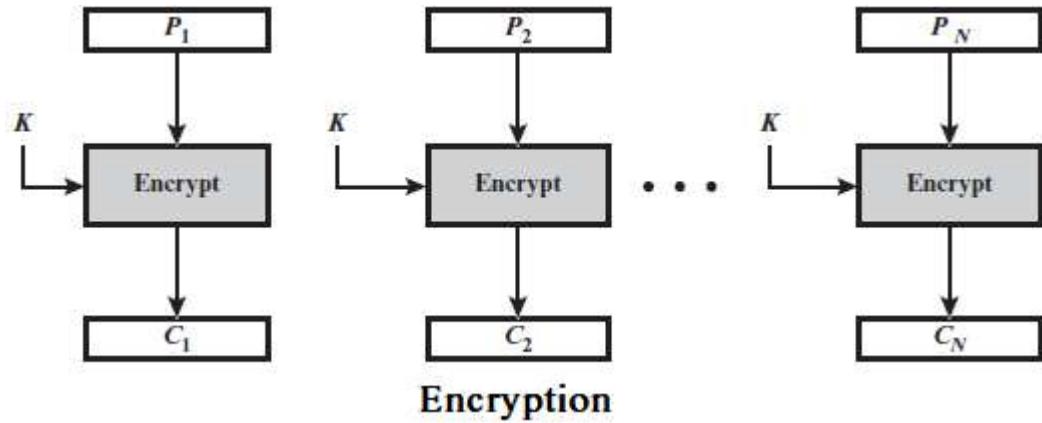
|     |   |   |
|-----|---|---|
| ECB | $C_j = E(K, P_j) \quad j = 1, \dots, N$ | $P_j = D(K, C_j) \quad j = 1, \dots, N$ |
|-----|---|---|

# Electronic Code Book Mode (ECB)

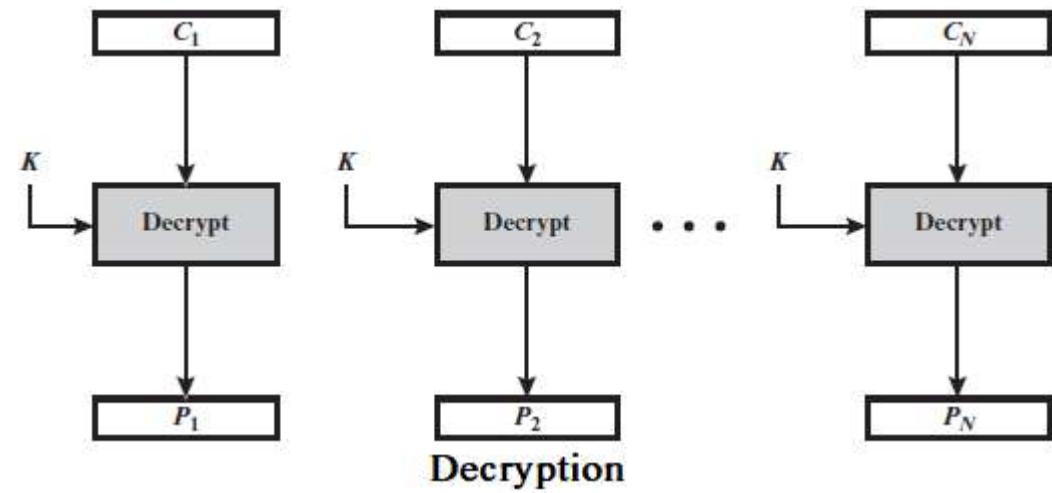
Let us simply divide an incoming stream into different blocks, and encrypt each block



Referred to as the Electronic Code Book method because the encryption process can be represented by a fixed mapping between the input blocks of plaintext and the output blocks of cipher text



Encryption

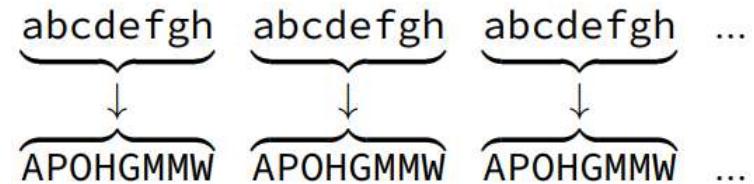
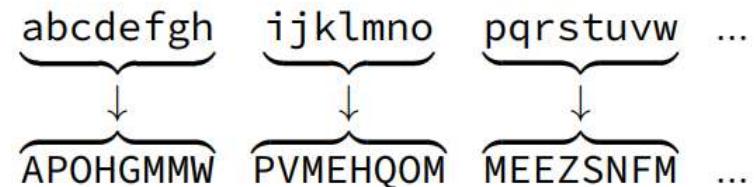


ECB Block Diagram

# Electronic Code Book Mode (ECB) Issues

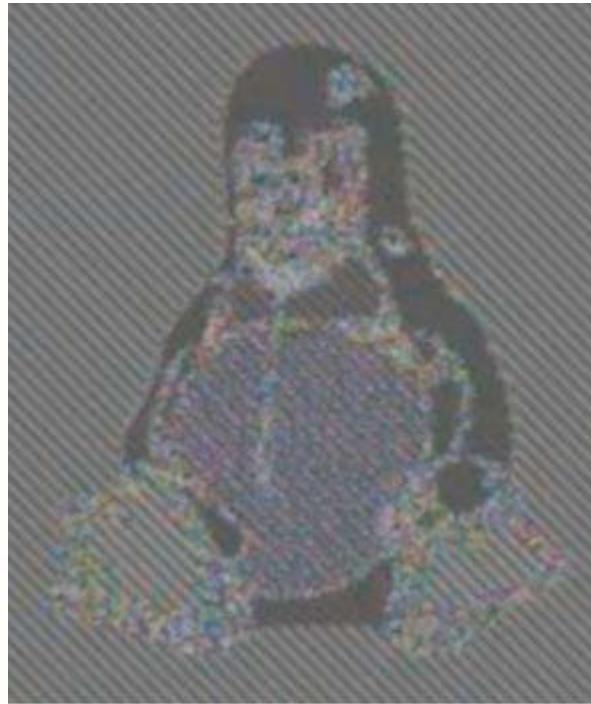
Simply divide an incoming stream into different blocks, and encrypt each block as:

In this mode, identical input blocks will always map to identical output blocks





Original Image



Encrypted using ECB



Encrypted using other modes

**Electronic Code Book Mode (ECB)**

# Advantages & Limitations

Very simple and easy to implement

Can be used for sending a few blocks of data

Message repetitions may show in ciphertext

Some information can be exposed

- With the messages that change very little
- Particularly with data such as graphics

# Cipher Block Chaining (CBC)

# Cipher Block Chaining (CBC)

To overcome the security deficiencies of ECB  $\Rightarrow$  need of a technique in which the same plaintext block, if repeated, produces different ciphertext blocks  $\Rightarrow$  a way to satisfy this requirement is the cipher block chaining (CBC) mode

In CBC, the input to the encryption algorithm is the XOR of the current plaintext block and the preceding ciphertext block; the same key is used for each block  $\Rightarrow$  in effect, we have chained together the processing of the sequence of plaintext blocks

The input to the encryption function for each plaintext will be changed, i.e., same plaintext will be different  $\Rightarrow$  repeating patterns of bits are not exposed

# Cipher Block Chaining (CBC)

Message is broken into blocks

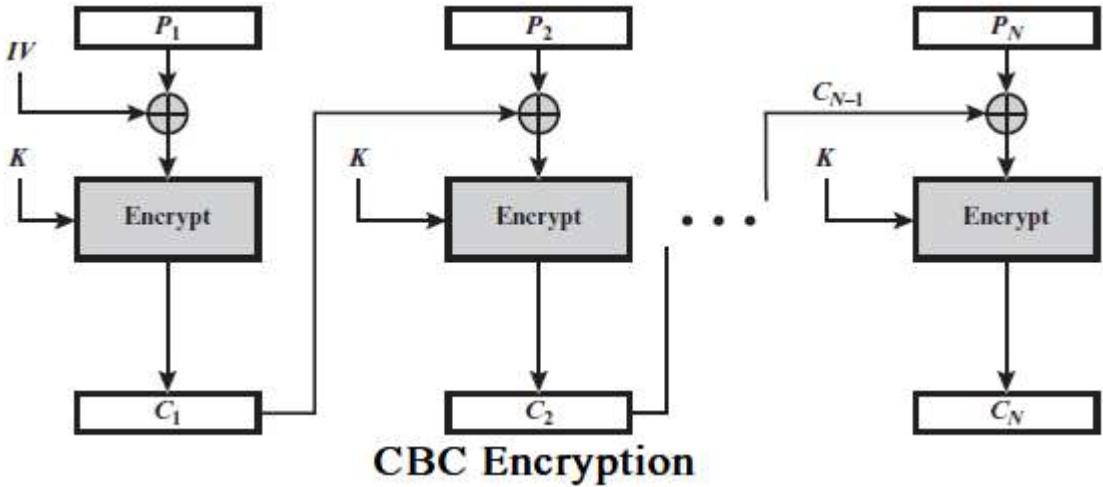
Lined together in encryption operation

Each previous cipher block is chained with current plaintext block, hence name

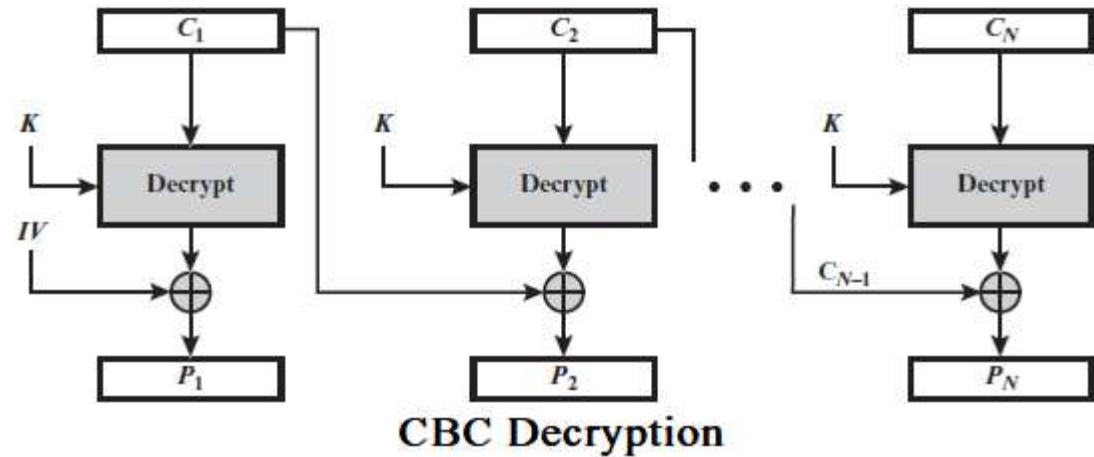
Use initial vector (IV) to start process

- $C_i = DES_{K1} (P_i \text{ XOR } C_{i-1})$
- $C_{-1} = IV$       (Initialization vector)

Uses: Bulk data encryption, authentication



**CBC Encryption**

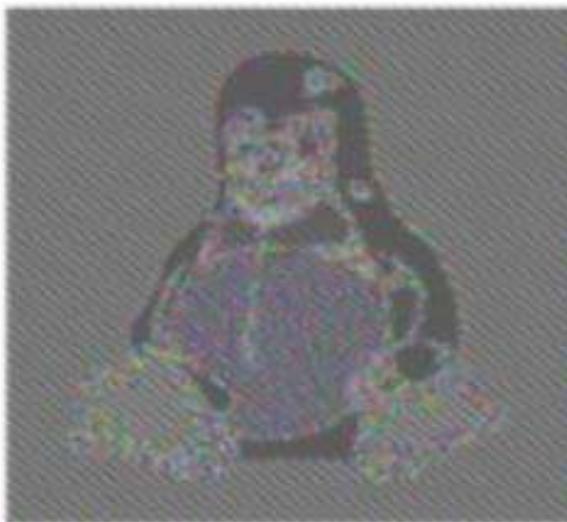


**CBC Block Diagram**

**CBC Decryption**



**Original image**



**Image with AES  
using ECB**



**Image with AES  
using CBC**

# CBC Analysis

ECB: Independent blocks, Same PT  $\Rightarrow$  Same CT

CBC: Dependent blocks, Same PT  $\Rightarrow$  Different CT

CBC uses concept of chaining  $\Rightarrow$  No parallelism is possible  $\Rightarrow$  Slow

Repeating patterns of bits are not exposed in CBC

Key is same for all blocks as in ECB

Need of IV (Initialization Vector), that should be known to sender & receiver

# Cipher Feedback Mode (CFB)

# Cipher FeedBack (CFB)

For AES, DES, or any block cipher, encryption is performed on a block of  $b$  bits, so as in DES,  $b = 64$  and in of AES,  $b = 128$

However, it is possible to convert a block cipher into a stream cipher, using one of the three modes **CFB mode**, **OFB mode**, and **CTR mode**

A stream cipher eliminates the need to pad a message to be an integral number of blocks, and it also can operate in real time

In stream cipher, each character can be encrypted and transmitted immediately using a character-oriented stream cipher

# Cipher FeedBack (CFB)

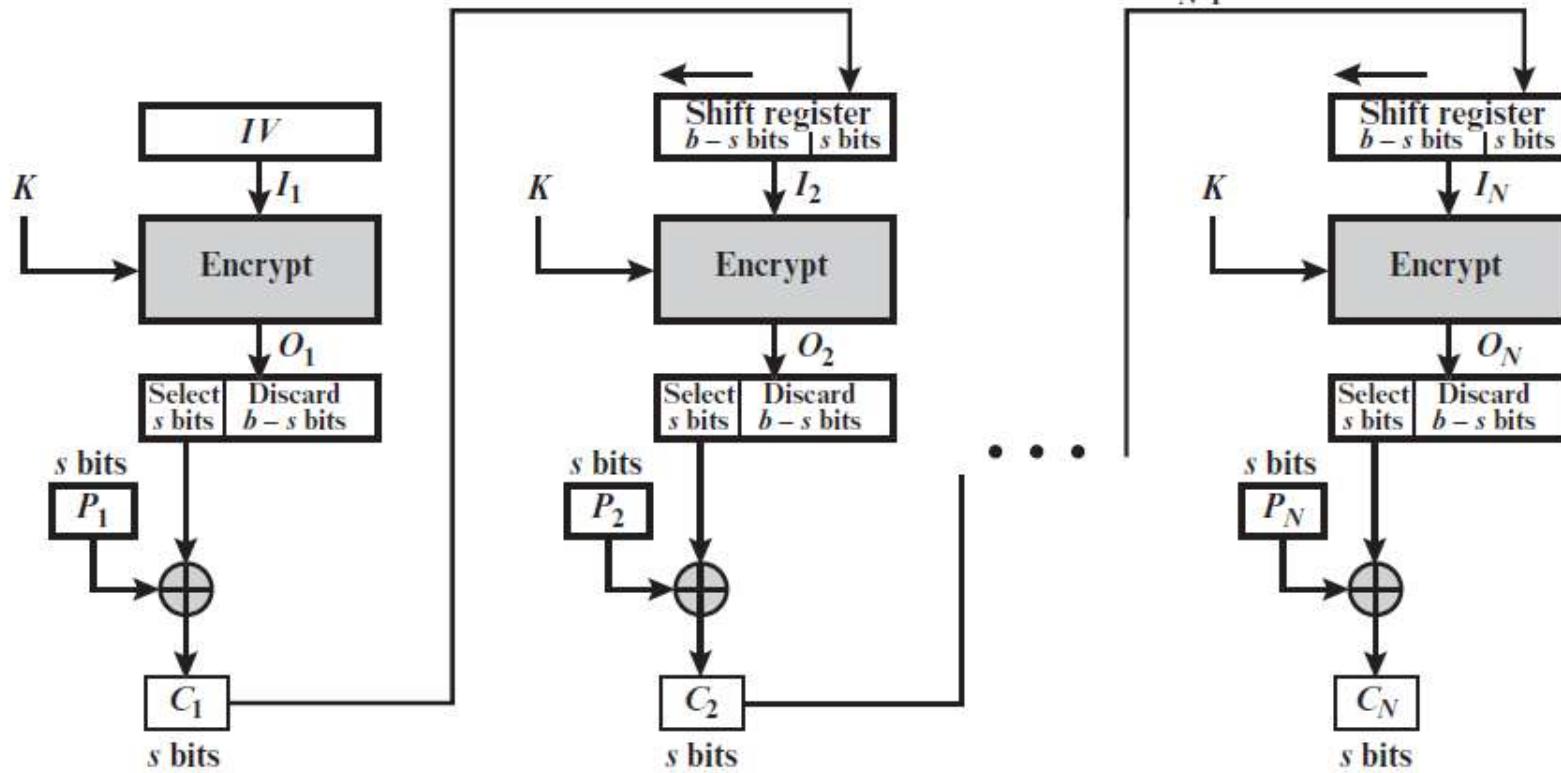
Message is treated as a stream of bits

Added to the output of the block cipher

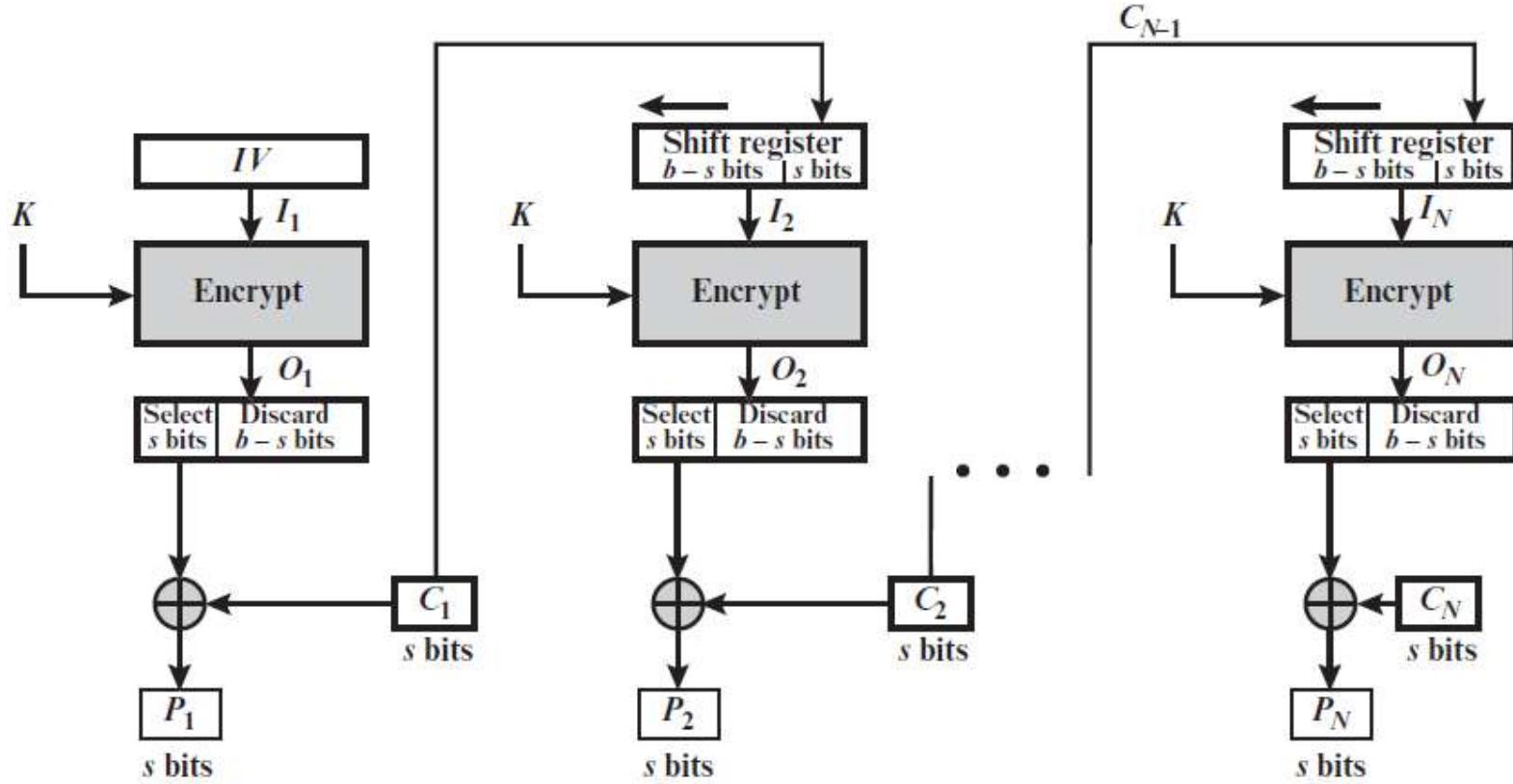
Standard allows any number of bits (1, 8, 64 or 128) to be feedback

- Denoted as CFB-1, CFB-8, CFB-64, CFB-128 etc

Most efficient to use all bits in block (64 or 128)



## CFB Encryption



## CFB Decryption

# CFB

The unit of transmission is  $s$  bits; a common value is  $s = 8$

As with CBC, the units of plaintext are chained together, so that the ciphertext of any plaintext unit is a function of all the preceding plaintext

Rather than blocks of  $b$  bits, the plaintext is divided into segments of  $s$  bits

# Output Feedback Mode (OFB)

# The Output Feedback Mode (OFB)

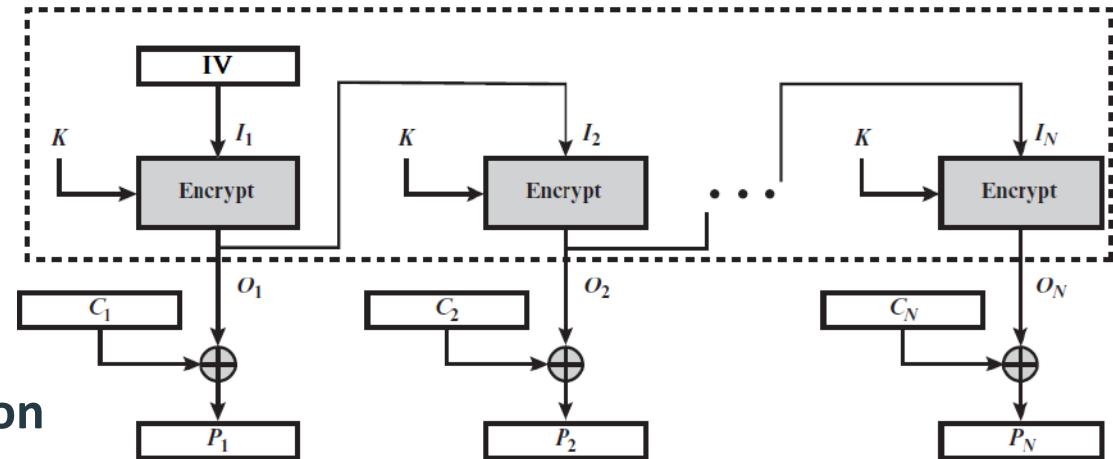
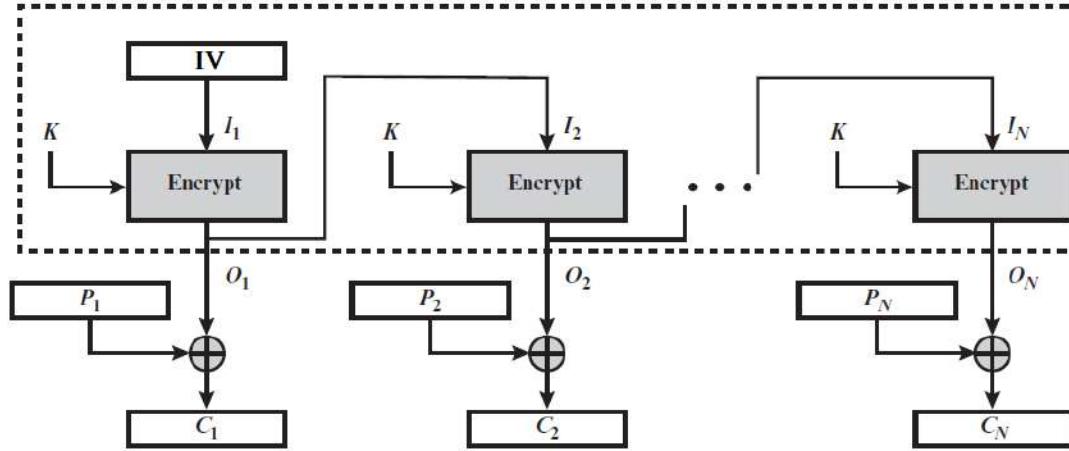
Very similar to the CFB mode

Can also be used as a stream cipher

Some bits of the most significant position are fed back from the output of the block cipher encryption algorithm, rather than actual ciphertext byte

If some bits of the ciphertext get garbled, only those bits of plaintext get garbled; so it is more resistant to transmission bit errors

The message can be of any arbitrary size



## OFB Encryption and Decryption

# OFB Pros & Cons

## Pros:

Bit errors do not propagate, e.g., if a bit error occurs in C1, only the recovered value of P1 is affected; subsequent plaintext units are not corrupted

Same PT with same key  $\Rightarrow$  different CT

PT length can be of random choice

## Cons:

More vulnerable to modification attack, consider that complementing a bit in the ciphertext complements the corresponding bit in the recovered plaintext

# Counter Mode (CTR)

# Counter Mode (CTR)

CTR mode makes block cipher way of working similar to a stream cipher

Although interest in the counter (CTR) mode has increased later on with application to ATM (asynchronous transfer mode) network security and IPsec (IP security), this mode was proposed in 1979

In this mode, subsequent values of an increasing counter are added to a nonce value (the nonce means a number that is unique: number used once) and the results are encrypted as usual

The nonce plays the same role as initialization vectors in the previous modes

# CTR

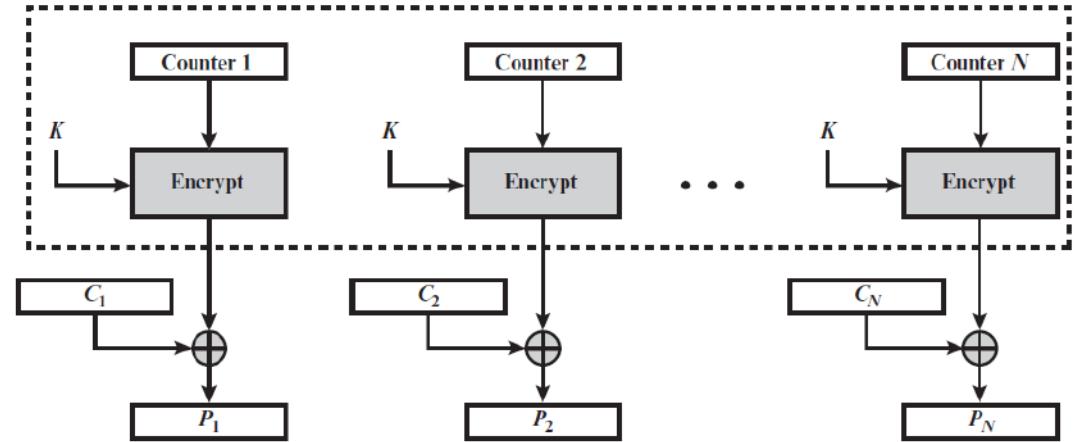
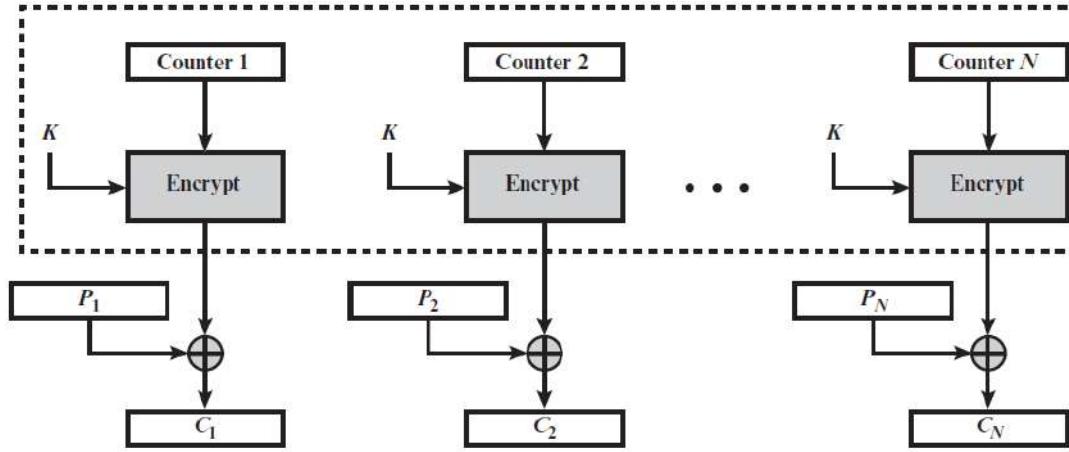
A counter equal to the plaintext block size is used

The counter value must be different for each plaintext block that is encrypted

Typically, the counter is initialized to some value and then incremented by 1 for each subsequent block (modulo  $2^b$ , where b is the block size)

For encryption, the counter is encrypted and then XORed with the plaintext block to produce the ciphertext block; there is no chaining

For decryption, the same sequence of counter values is used, with each encrypted counter XORed with a ciphertext block to recover the corresponding plaintext block



## CTR Encryption and Decryption

# Counter Mode (CTR)

Fast encryption and decryption

Can be implemented on parallel machines because there is not block-to-block feedback

Any block can be decrypted with random access

The security of CTR is similar with the other modes for using block ciphers

# Distributed and Edge Computing

# Distributed Systems Security

Sharad K. Ghimire

Department of Electronics and Computer Engineering  
Pulchowk Campus  
Institute of Engineering  
Tribhuvan University

# Contents

Asymmetric Key Cryptography

Diffie-Hellman Algorithm

RSA Algorithm

Message Integrity

Digital Signature

# Asymmetric Cryptography

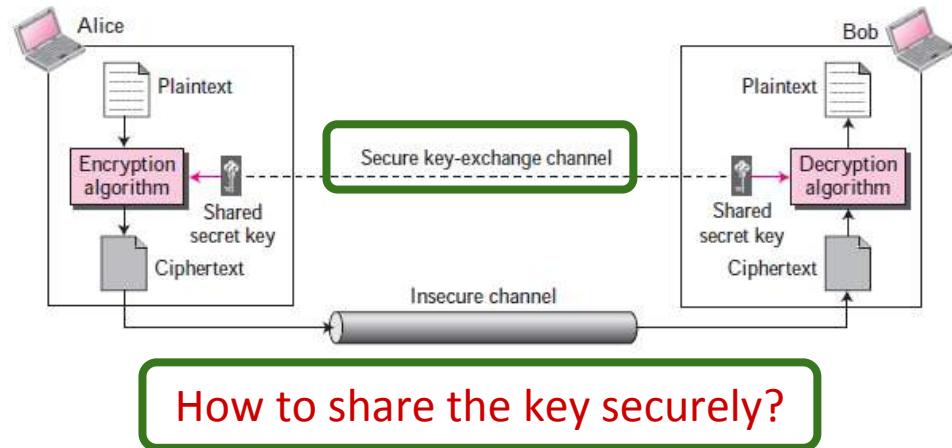
| Key Size (bits)             | Number of Alternative Keys     | Time Required at 1 Decryption/ $\mu$ s                | Time Required at $10^6$ Decryptions/ $\mu$ s |
|-----------------------------|--------------------------------|---|--|
| 32                          | $2^{32} = 4.3 \times 10^9$     | $2^{31} \mu$ s = 35.8 minutes                         | 2.15 milliseconds                            |
| 56                          | $2^{56} = 7.2 \times 10^{16}$  | $2^{55} \mu$ s = 1142 years                           | 10.01 hours                                  |
| 128                         | $2^{128} = 3.4 \times 10^{38}$ | $2^{127} \mu$ s = $5.4 \times 10^{24}$ years          | $5.4 \times 10^{18}$ years                   |
| 168                         | $2^{168} = 3.7 \times 10^{50}$ | $2^{167} \mu$ s = $5.9 \times 10^{36}$ years          | $5.9 \times 10^{30}$ years                   |
| 26 characters (permutation) | $26! = 4 \times 10^{26}$       | $2 \times 10^{26} \mu$ s = $6.4 \times 10^{12}$ years | $6.4 \times 10^6$ years                      |

Table: Average Time Required for Exhaustive Key Search

As AES allows key lengths of 128, 192, and 256 bits, what about time required for exhaustive key search?

Why is symmetric encryption not enough?

# Why Asymmetric?



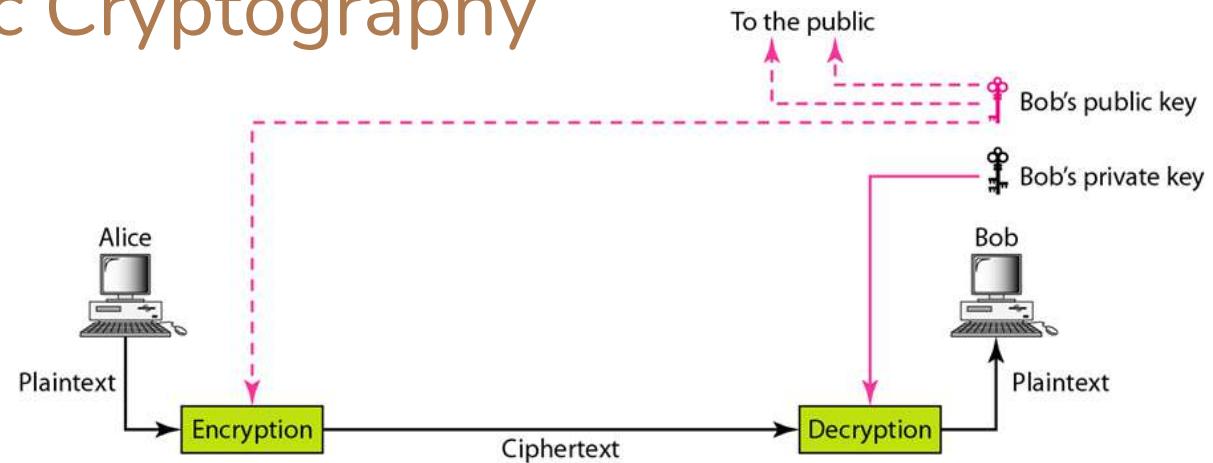
Symmetric encryption is universal technique for providing confidentiality for the transmitted data ⇒ single-key encryption

Symmetric encryption is very strong e.g. AES 256

# Asymmetric Cryptography

Public-Key Cryptography

# Asymmetric Cryptography



There are two keys: A **private key** and a **public key** ⇒ **Public Key Cryptography**

Private key is kept by the receiver and Public key is announced to the public

# Asymmetric Cryptography

In the setting of private-key encryption, two parties agree on a secret key **k** which can be used (by either party) for both encryption and decryption

Public-key encryption is asymmetric in both these respects ⇒ specifically, one party (the receiver) generates a pair of keys called the **public key** and the **private key**

The public key is used by a sender to encrypt a message for the receiver; the receiver then uses the private key to decrypt the resulting ciphertext

The goal is to avoid the need for two parties to meet in advance to agree on key

# Asymmetric Cryptography

Public-key, or asymmetric cryptography is one of the **greatest** revolution in the **history of cryptography**

Virtually all cryptographic systems have been based on the elementary tools of **substitution** and **permutation**

With the availability of computers, even more complex systems were devised, the most prominent of which was the Lucifer at IBM ⇒ Data Encryption Standard (**DES**) ⇒ still based on the basic tools of **substitution** and **permutation**

# Asymmetric Cryptography

Public-key algorithms are based on **mathematical functions** rather than on **substitution** and **permutation**

Public-key cryptography is asymmetric, involving the use of two separate keys, in contrast to symmetric encryption, which uses only one key

The use of two keys has profound consequences in the areas of **confidentiality, key distribution, authentication** as well as **non-repudiation**

# Symmetric & Asymmetric Cryptography

There are some common misconceptions concerning public-key encryption:

The public-key encryption is more secure from cryptanalysis than is symmetric encryption

- The security of any encryption scheme depends on the length of the key and the computational work involved in breaking a cipher
- There is nothing in principle about either symmetric or public-key encryption that makes one superior to another from the point of view of resisting cryptanalysis

# Symmetric & Asymmetric Cryptography

The public-key encryption is a general-purpose technique that has made symmetric encryption obsolete:

- Due to computational overhead of current public-key encryption schemes, there seems no foreseeable likelihood that symmetric encryption will be abandoned
- The use of public-key cryptography is in key management and signature applications is almost universally accepted

# Need of Both (Sym & Asym)

## Either one is not eliminating another

The asymmetric key (public-key) cryptography does not eliminate the need for symmetric-key (secretkey) cryptography

The asymmetric-key cryptography, which uses mathematical functions for encryption and decryption, is much slower than symmetric-key cryptography, so for encipherment of large messages ⇒ symmetric-key cryptography is still needed

# Need of Both (Sym & Asym)

## Either one is not eliminating another

On the other hand, the speed of symmetric-key cryptography does not eliminate the need for asymmetric-key cryptography

Asymmetric-key cryptography is still needed for authentication, digital signatures, and secret-key exchanges ⇒ to be able to use all aspects of security today, we need both symmetric-key and asymmetric-key cryptography ⇒ one complements the other

Both cryptography will exist in parallel and continue to serve the community

# Need of Asymmetric Cryptography

**Confidentiality:** Only Bob can read Alice's message without sharing secret ⇒ used to share key of symmetric encryption for large message

**Authenticity:** Alice can digitally “sign” her message, so Bob knows that only Alice could have sent it ⇒ he also knows Tom couldn't have tampered with the message in transit

**Non-repudiation:** Alice can't deny she sent (or at least saw) the message contents later on

# Public-Key Encryption Operation

# Public Key Encryption - Operation

Public key cryptography  $\Rightarrow$  two keys: a **private key** and a **public key**

The public key is announced to the public, whereas the private key is kept by the receiver because the private key should be kept secret

The sender uses the public key of the receiver for encryption and the receiver uses his private key for the decryption

These algorithms have the important characteristic, i.e., it is computationally infeasible to determine the decryption key given only knowledge of the cryptographic algorithm and the encryption key

# Public Key Encryption - Operation

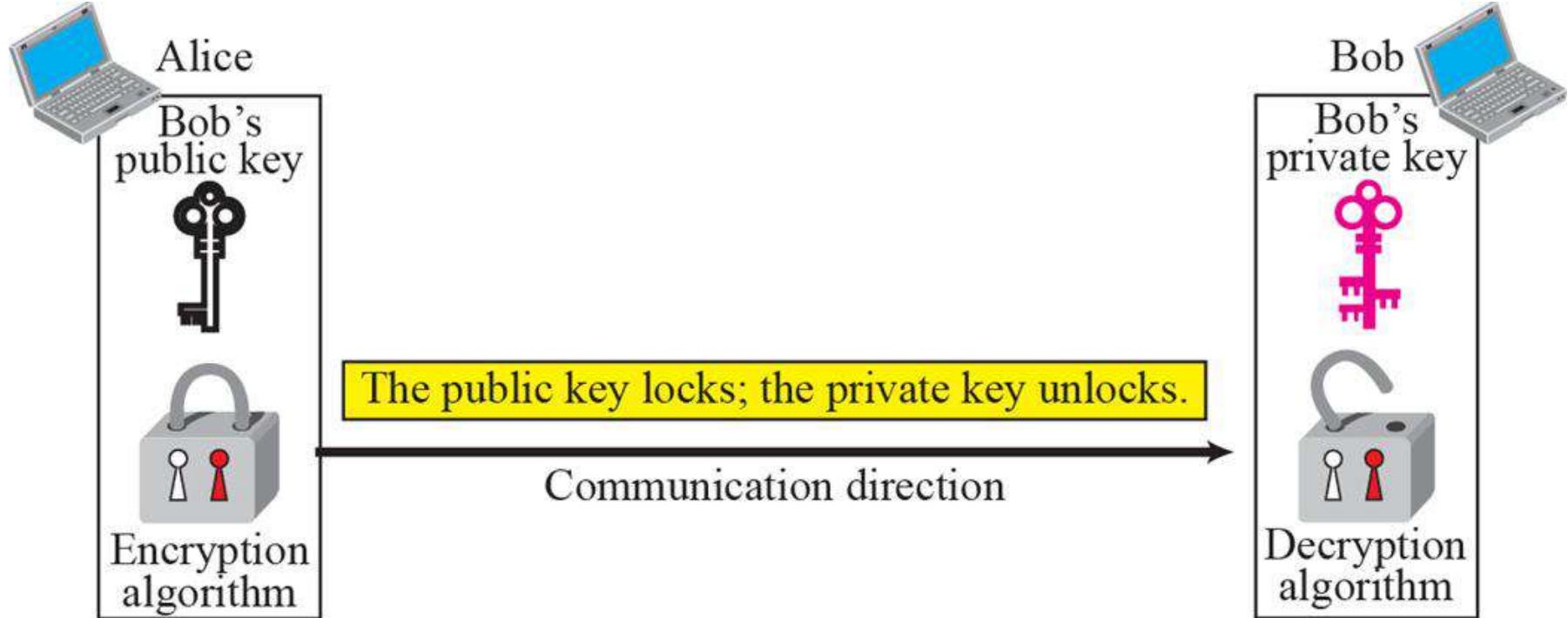
Public key is used for encryption

Private key is used for decryption

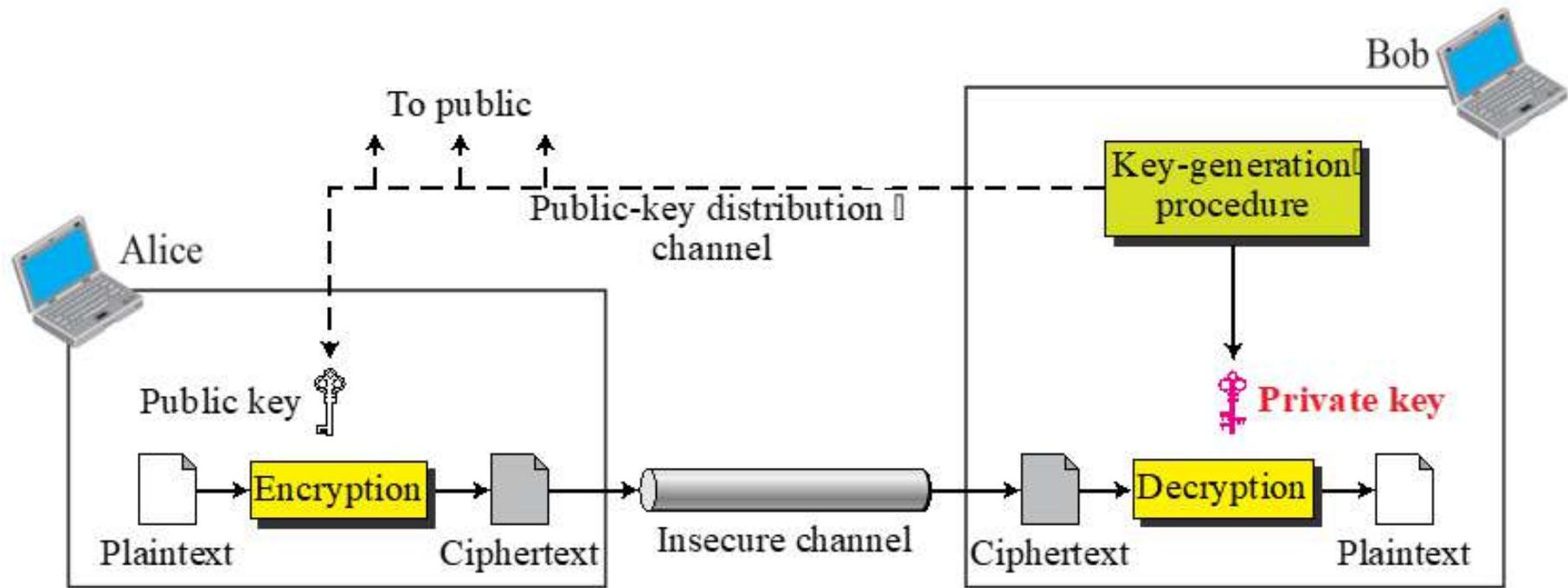
Infeasible to determine decryption key given encryption key and algorithm

Steps:

- User generates pair of keys
- User places one key in public domain
- To send a message to user, encrypt using public key
- User decrypts using private key



*Locking and unlocking in asymmetric-key cryptosystem*



**General idea of asymmetric-key cryptosystem**

# Diffie-Hellman Algorithm

# Diffie-Hellman Algorithm

Originally designed for key exchange

Two parties create a symmetric session key to exchange data without having to remember or store key for further use

No need to meet to agree on the key

Common key exchange can be done through public channel such as Internet

The purpose of the algorithm is to enable two users to securely exchange a key that can then be used for subsequent symmetric encryption of messages

The algorithm itself is limited to the exchange of secret values

## Diffie-Hellman Algorithm - Steps:

Alice chooses a large random number  $x$  and calculates  $R_1 = g^x \bmod p$

Bob chooses another large random number  $y$  and calculates  $R_2 = g^y \bmod p$

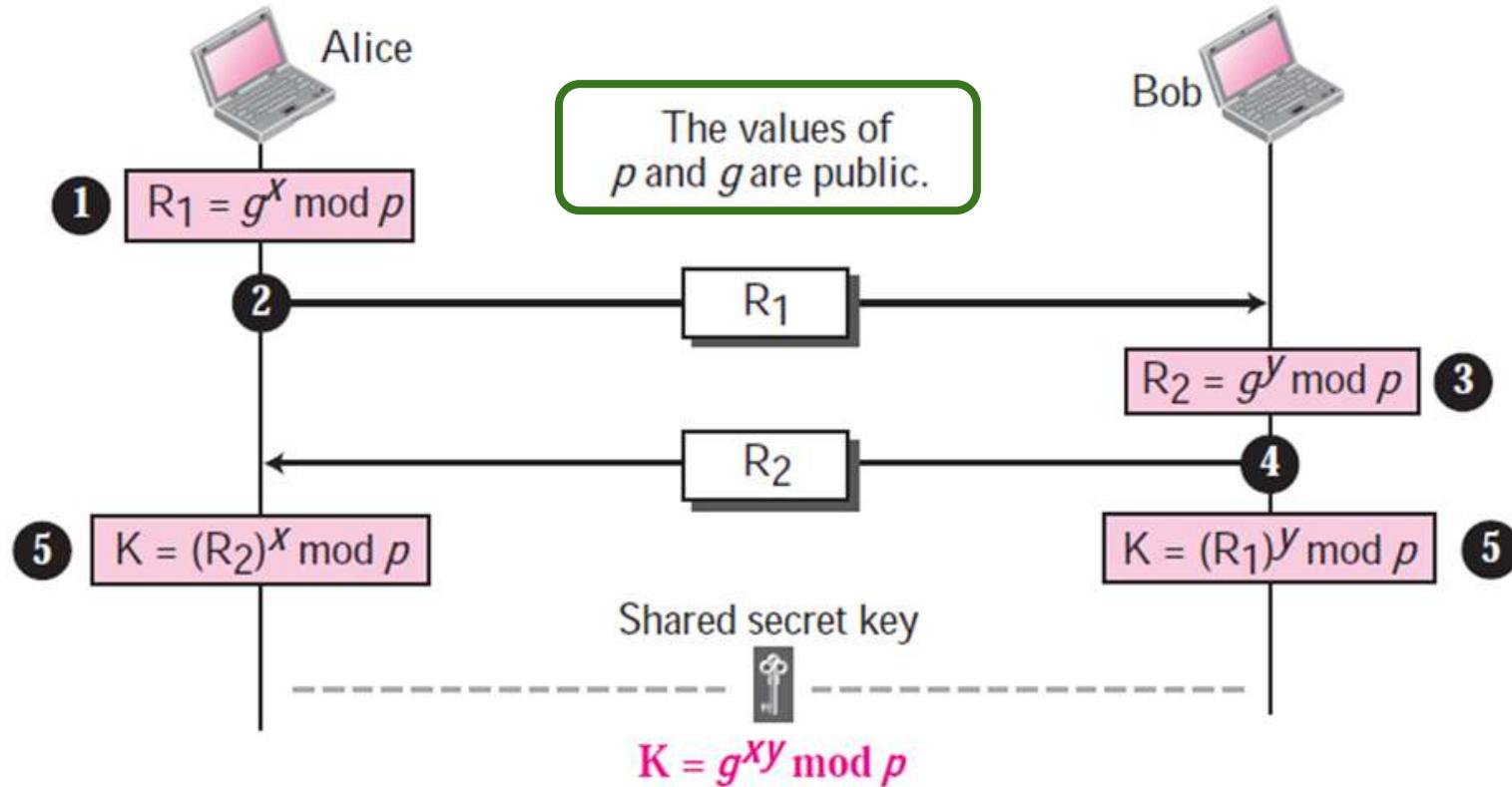
Alice sends  $R_1$  to Bob (but Alice does not send the value of  $x$ )

Bob sends  $R_2$  to Alice (but Bob does not send the value of  $y$ )

Alice calculates  $K = (R_2)^x \bmod p$

Bob also calculates  $K = (R_1)^y \bmod p$

The symmetric key for the session is  $K = g^{xy} \bmod p$



Diffie-Hellman Key Exchange, Shared Key is:  $K = g^{xy} \text{ mod } p$

# Diffie-Hellman Method Example

Assume  $g = 7$  and  $p = 23$ . The steps are as follows:

Alice chooses  $x = 3$  and calculates  $R_1 = 7^3 \bmod 23 = 21$

Bob chooses  $y = 6$  and calculates  $R_2 = 7^6 \bmod 23 = 4$

Alice sends the number 21 to Bob

Bob sends the number 4 to Alice

Alice calculates the symmetric key  $K = 4^3 \bmod 23 = 18$

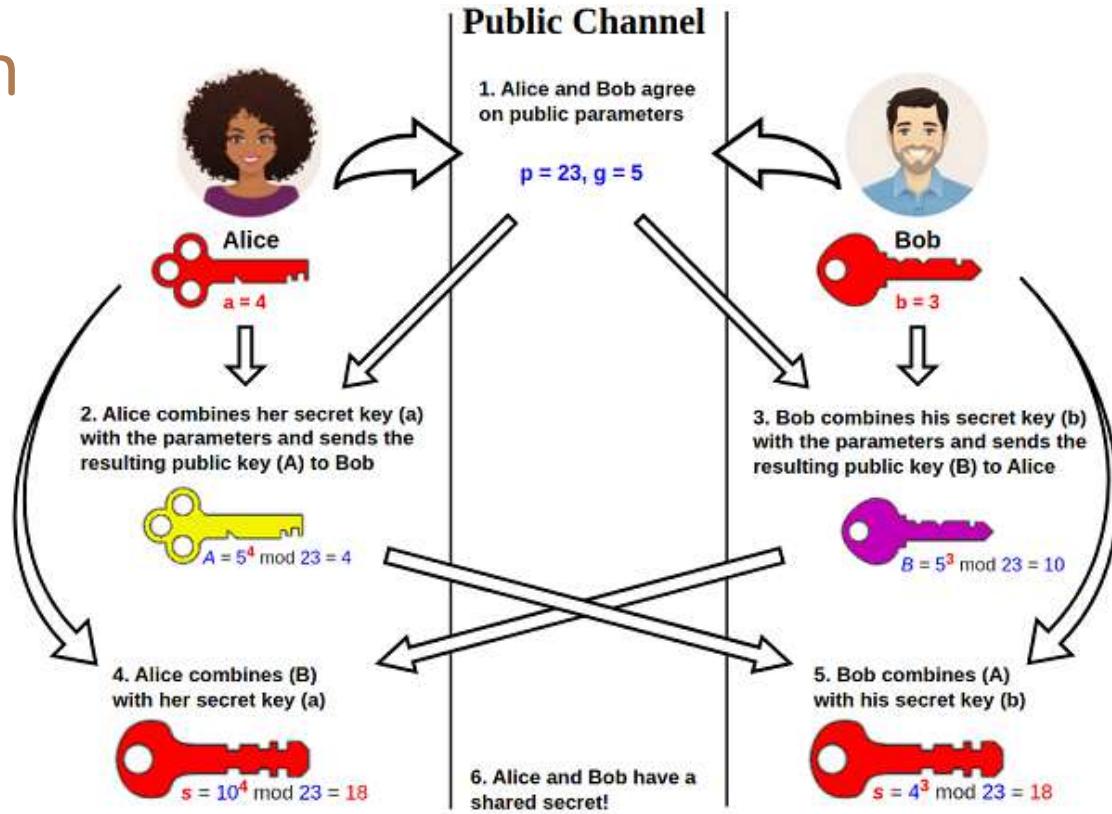
Bob calculates the symmetric key  $K = 21^6 \bmod 23 = 18$

The value of  $K$  is the same for both Alice and Bob

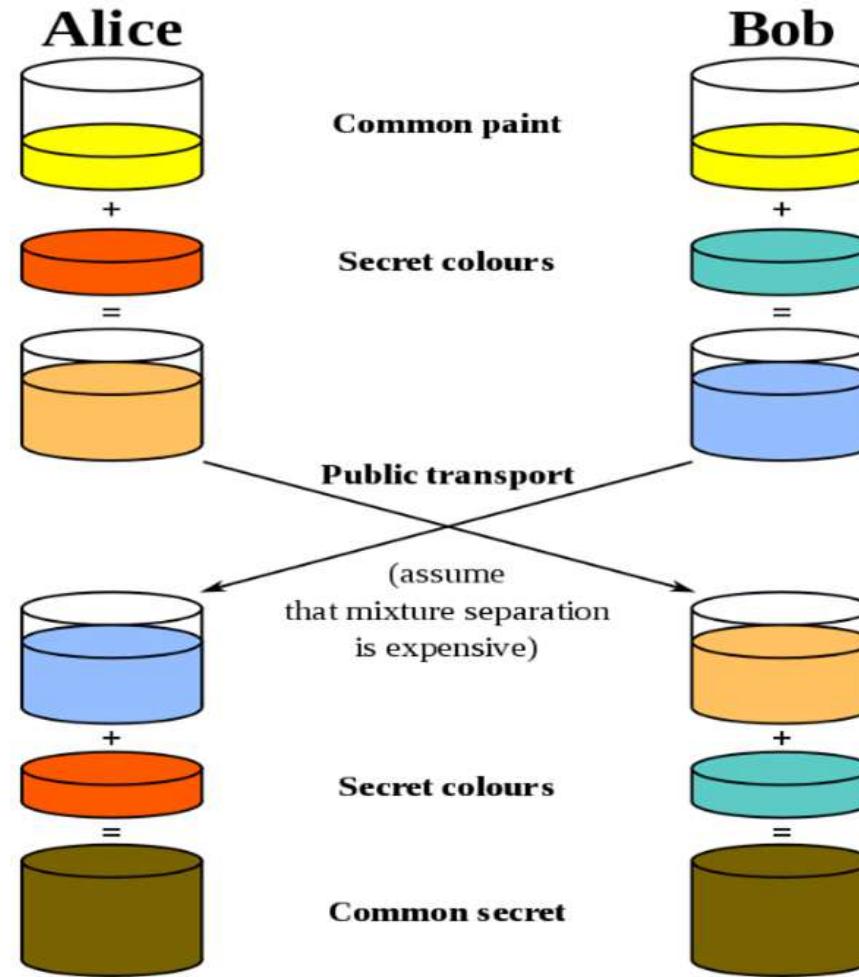
$$g^{xy} \bmod p = 7^{18} \bmod 23 = 18$$

# Diffie-Hellman Key Exchange

Two parties got a common secret key, without passing common secret across the public channel



# Illustration of Diffie-Hellman key exchange



## Realistic Example

Let us create a random integer of 512 bits (the ideal is 1024 bits). The integer  $p$  is a 159-digit number. We also choose  $g$ ,  $x$ , and  $y$  as:

|     |   |
|-----|---|
| $p$ | 764624298563493572182493765955030507476338096726949748923573772860925<br>23566666075542363742330966118003338106194730130950414738700999178043<br>6548785807987581 |
| $g$ | 2   |
| $x$ | 557   |
| $y$ | 273   |

|                      |  |
|----------------------|--|
| <b>R<sub>1</sub></b> | 844920284205665505216172947491035094143433698520012660862863631067673<br>619959280828586700802131859290945140217500319973312945836083821943065<br>966020157955354  |
| <b>R<sub>2</sub></b> | 435262838709200379470747114895581627636389116262115557975123379218566<br>310011435718208390040181876486841753831165342691630263421106721508589<br>6255201288594143 |
| <b>K</b>             | 155638000664522290596225827523270765273218046944423678520320400146406<br>500887936651204257426776608327911017153038674561252213151610976584200<br>1204086433617740 |

**Showing the values of R1, R2, and K**

# Analysis of Diffie-Hellman

The secret key between Alice and Bob is made of three parts:  $g$ ,  $x$ , and  $y$

The first part is public; everyone knows  $1/3$  of the key;  $g$  is a public value

The other two parts must be added by Alice and Bob; each of them adds one part  $\Rightarrow$  Alice adds  $x$  as the second part for Bob; Bob adds  $y$  as the second part for Alice

When Alice receives the  $2/3$  completed key from Bob, she adds the last part, her  $x$ , to complete the key

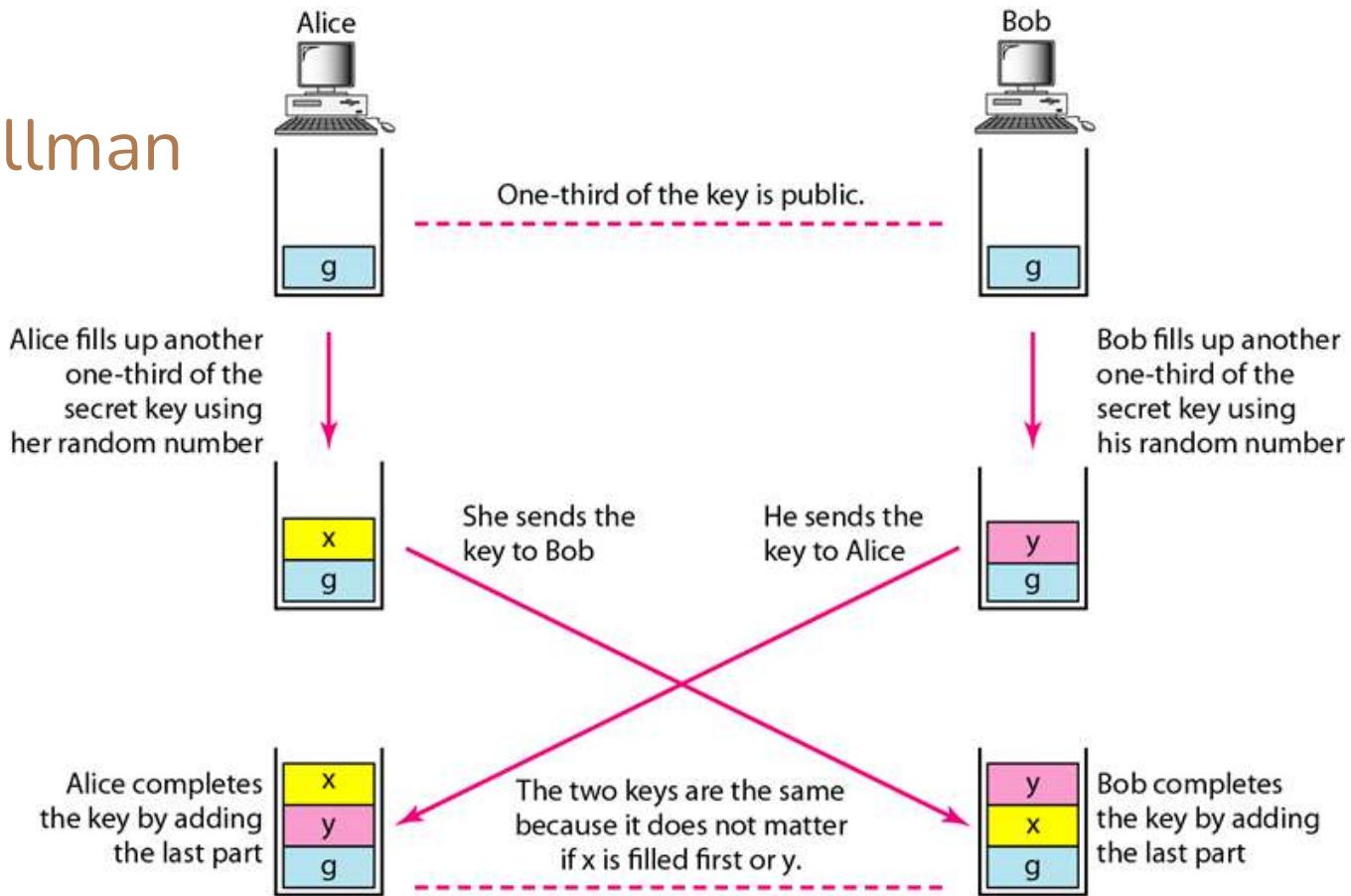
# Analysis of Diffie-Hellman

When Bob receives the 2/3-completed key from Alice, he adds the last part, his  $y$ , to complete the key

Here the key in Alice's hand consists of  $g$ ,  $y$ , and  $x$  and the key in Bob's hand also consists of  $g$ ,  $x$ , and  $y$ , these two keys are the same because  $g^{xy} = g^{yx}$

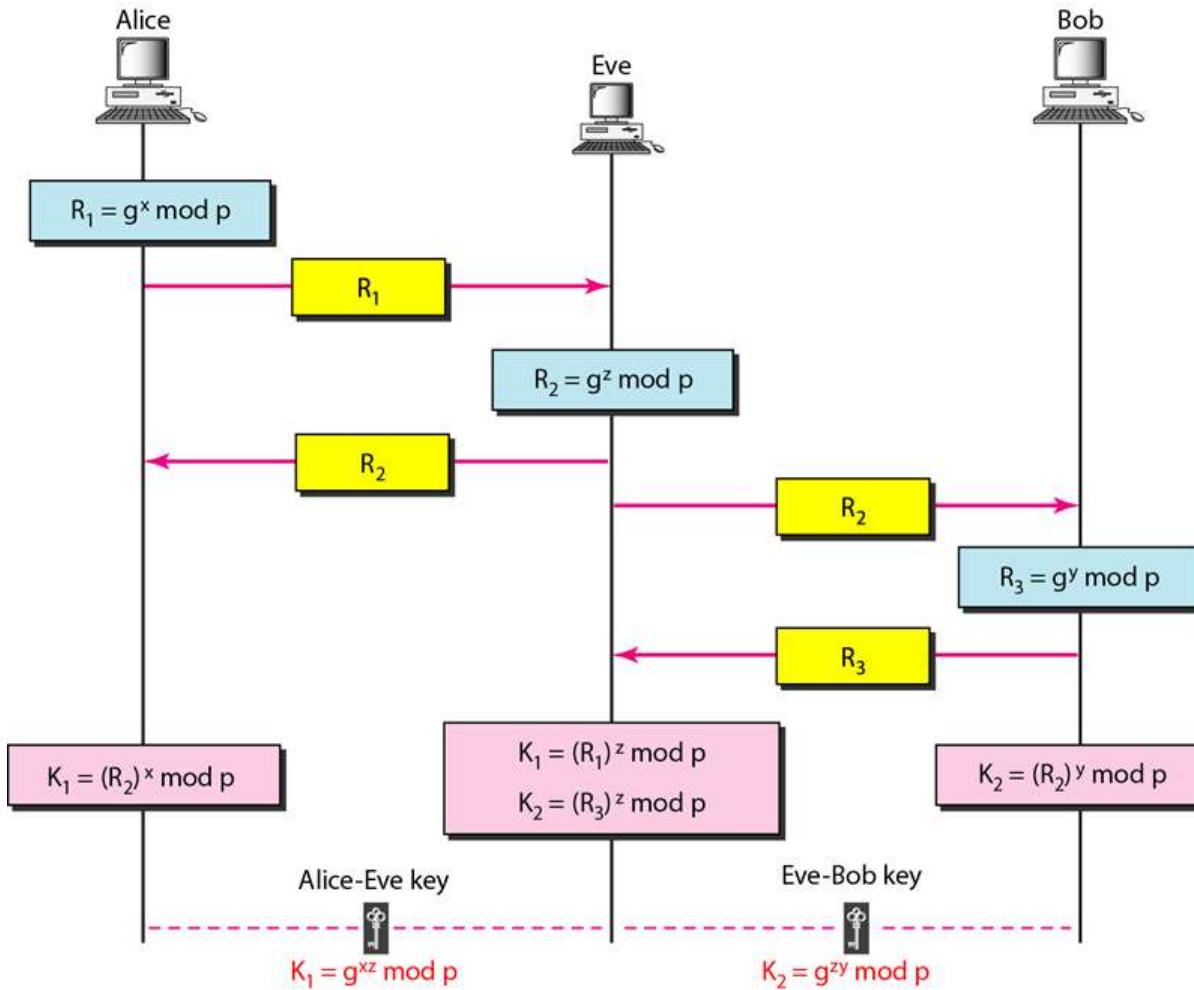
Although the two keys are the same, Alice cannot find the value  $y$  used by Bob because the calculation is done in modulo  $p$ ; Alice receives  $g^y \bmod p$  from Bob, not  $g^y$

# Diffie-Hellman Idea



# Man-in-the-Middle Attack

p and g are public.



# RSA (Rivest-Shamir- Adleman) Algorithm

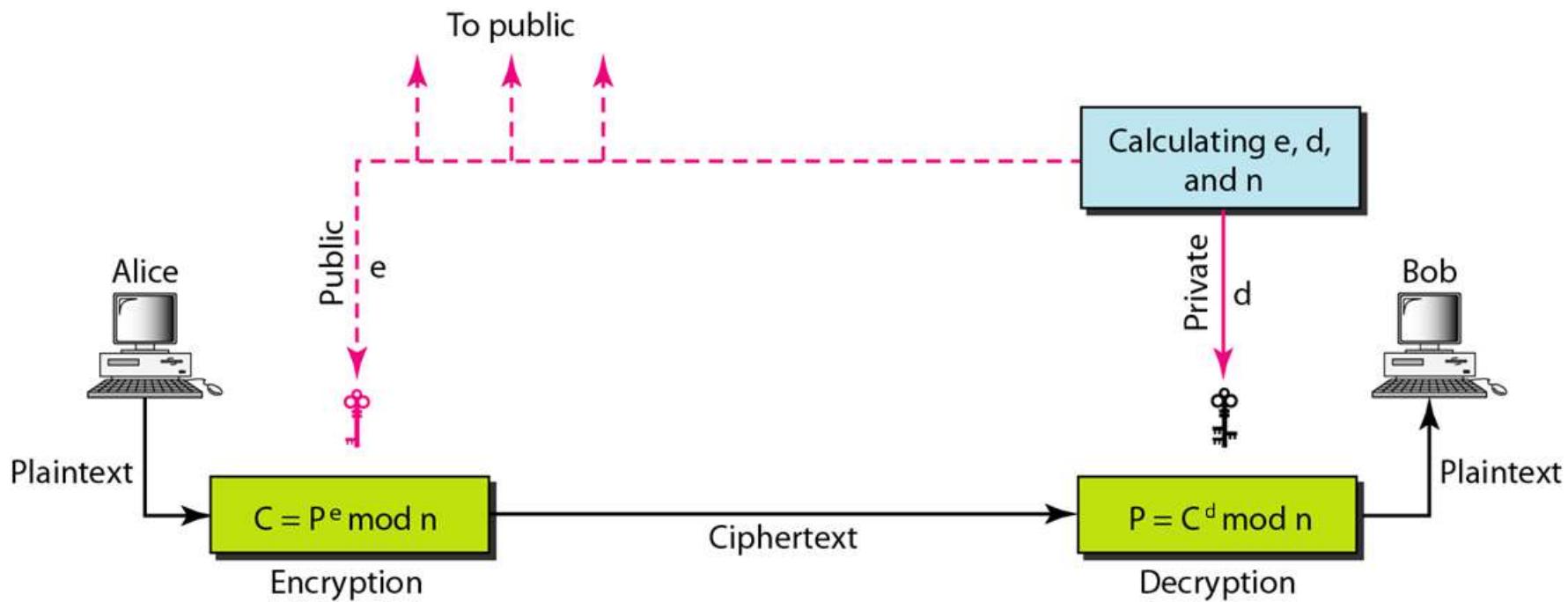
# RSA (Rivest-Shamir-Adleman) Algorithm

The most common public-key algorithm used by modern computers to encrypt and decrypt messages

An asymmetric cryptographic algorithm also known as public key cryptographic algorithm

Named from its inventors **Rivest, Shamir, and Adleman**, who publicly described the algorithm in 1977

# RSA Algorithm



# RSA Algorithm

Public Key: e, n  
Private Key: d, n

Choose two different large random **prime numbers p and q**

Calculate  $n = pq \Rightarrow n$  is the modulus for the public key and the private keys

Calculate the  $\phi(n) = (p-1)(q-1)$

Choose an integer e, such that  $1 < e < \phi(n)$ , such that e and  $\phi(n)$  share no factor other than 1, i.e.  $\gcd(e, \phi(n)) = 1 \Rightarrow e$  is announced as public key exponent

Compute d such that  $de \bmod \phi(n) = 1 \Rightarrow d$  is kept as the private key exponent

In RSA  $(e, n)$  is public; and the integer  $d$  is private

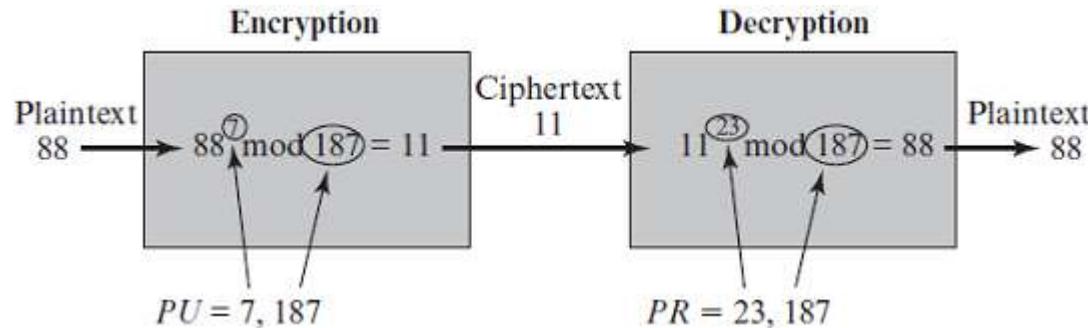
# RSA Encryption and Decryption

## Encryption

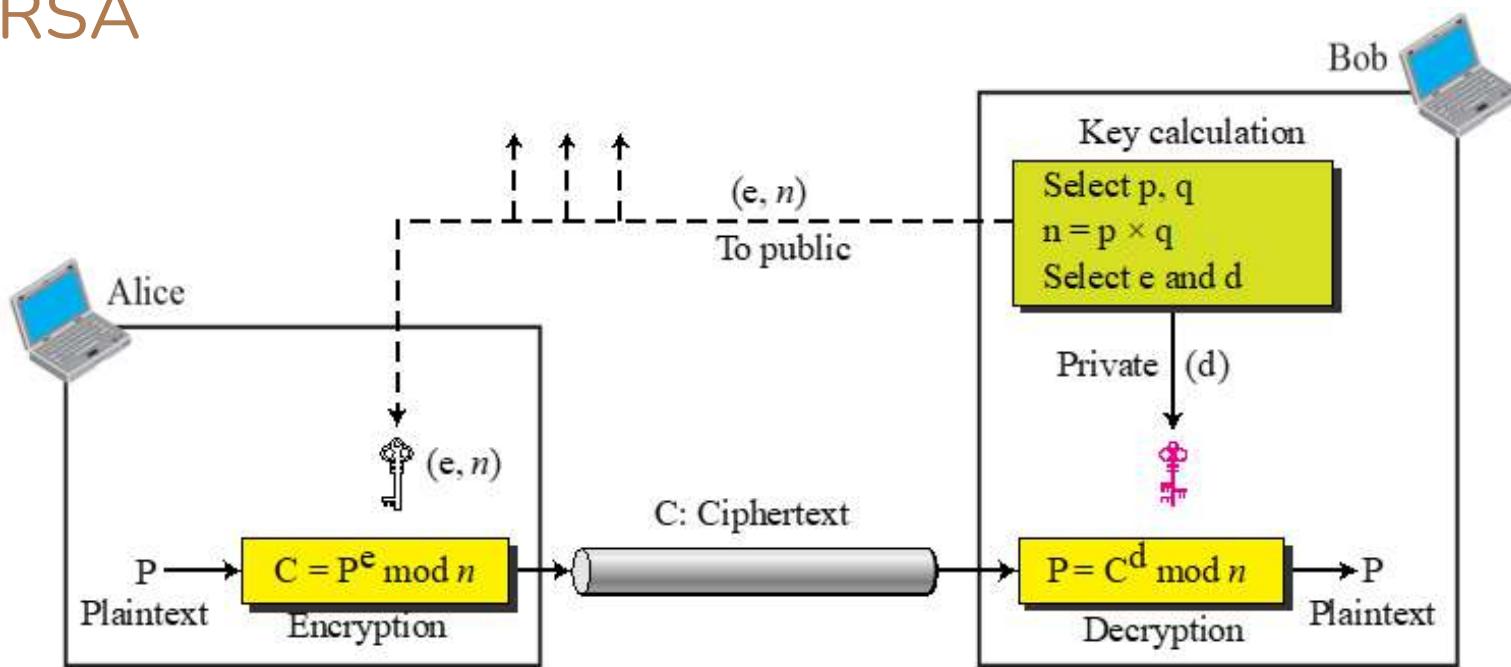
$$C = P^e \bmod n$$

## Decryption

$$P = C^d \bmod n$$



# Encryption, decryption, and key Generation in RSA



# RSA Example Key Generation

Choose two prime numbers, p and q  $\Rightarrow$  let p = 5 and q = 7

Compute n = p \* q  $\Rightarrow$  n = 5 \* 7 = 35

Calculate the  $\phi(n) = (p - 1) * (q - 1) \Rightarrow \phi(35) = (5-1) * (7-1) = 24$

Choose an integer e, such that  $1 < e < \phi(n)$  and  $\gcd(e, \phi(n)) = 1 \Rightarrow$  let e = 5

Compute the modular multiplicative inverse of e, which is  $d = e^{-1} \pmod{\phi(n)}$

$\Rightarrow d = 29$  (since  $5 * 29 = 1 \pmod{24}$ )

Now, the public key (e, n) = (5, 35)

And the private key (d, n) = (29, 35)

# RSA Example: Encryption and Decryption

Let us represent the letters a to z by numbers from 01 to 26

As plaintext message is **j**, the corresponding numerical plaintext is  $P = 10$  and with encryption key  $e = 5$

$$C = P^e \pmod{n}$$

$$C = 10^5 \pmod{35} = \mathbf{22} \text{ (cipher text)}$$

Receiver uses the decryption key  $d = 29$

The ciphertext message  $C = 22$  is decrypted as:

$$M = C^d \pmod{n}$$

$$M = 22^{29} \pmod{35} = 10$$

While decoding  $10 \Rightarrow j$  same with that of plaintext ( $P$ )

# RSA Example Key Generation

Let, two random prime numbers **p = 5** and **q = 11**

Now, **n = 5 × 11 = 55**

Then  **$\phi = (p-1) \times (q-1) = 4 \times 10 = 40$**

Let us choose a prime value **e = 7**

Since, the decryption key d must be the multiplicative inverse of e modulo  $\phi$

**$e \times d \bmod \phi = 1$** , As the value 23 satisfies the requirement, so **d = 23**

# Exercise

Encrypt the message “CRYPTO” using RSA

Also decrypt the encrypted message to recover the plaintext

## RSA Keys

Public:  $e, n \Rightarrow 7, 55$

Private:  $d, n \Rightarrow 23, 55$

Encoding message in numeric values (using 1 to 26 for A to Z):

|   |   |    |
|---|---|----|
| C | ⇒ | 3  |
| R | ⇒ | 18 |
| Y | ⇒ | 25 |
| P | ⇒ | 16 |
| T | ⇒ | 20 |
| O | ⇒ | 15 |

# Encryption

$$C = P^e \pmod{n} \text{ with } e, n \Rightarrow 7, 55$$

$$3^7 \pmod{55} = 42$$

$$18^7 \pmod{55} = 17$$

$$25^7 \pmod{55} = 20$$

$$16^7 \pmod{55} = 36$$

$$20^7 \pmod{55} = 15$$

$$15^7 \pmod{55} = 5$$

$$M = C^d \pmod{n} \text{ with } d, n \Rightarrow 23, 55$$

$$42^{23} \pmod{55} = 3$$

$$17^{23} \pmod{55} = 18$$

$$20^{23} \pmod{55} = 25$$

$$36^{23} \pmod{55} = 16$$

$$15^{23} \pmod{55} = 20$$

$$5^{23} \pmod{55} = 15$$

# More Practical Example

Alice creates a pair of keys by choosing  $p = 397$  and  $q = 401$

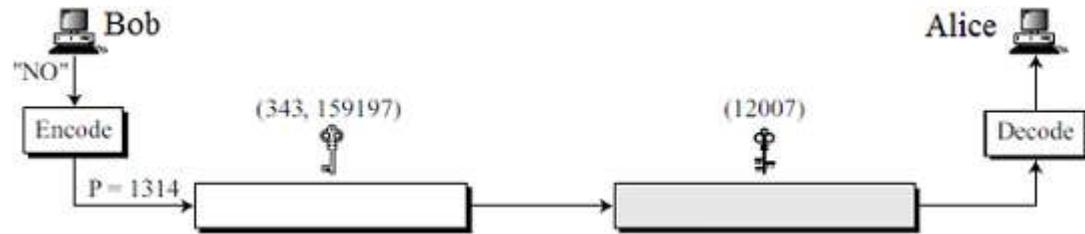
Then  $n = 159,197$  and  $\phi(n) = 396 \cdot 400 = 158,400$

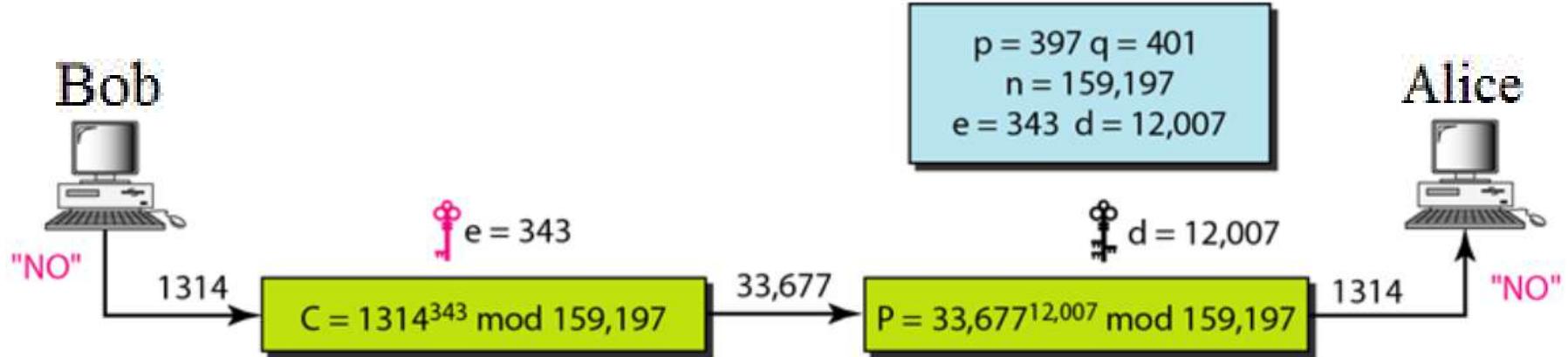
She then chooses  $e = 343$  and  $d = 12,007$

Now Bob wants to send a message “NO” to Alice by knowing  $e$  and  $n$

Bob changes each character to a number (from 00 to 25) with each character coded as two digits

He then concatenates the two coded characters and gets a four-digit number; plaintext as **1314**





Bob then uses e and n to encrypt the message

The ciphertext becomes  $1314^{343} \text{ mod } 159,197 = 33,677$

Alice receives the message 33,677 and uses the decryption key d to decipher it as  $33,677^{12,007} \text{ mod } 159,197 = 1314$

Alice then decodes 1314 as the message "NO"

# A Realistic Example of RSA

# A Realistic Example of RSA

Randomly chose an integer of 512 bits

**p =** 96130345313583504574191581280615427909309845594996215822583150879647940  
45505647063849125716018034750312098666606492420191808780667421096063354  
219926661209

The integer q is a 160-digit number

**q =** 12060191957231446918276794204450896001555925054637033936061798321731482  
14848376465921538945320917522527322683010712069560460251388714552496900  
0359660045617

**n = 11593504173967614968892509864615887523771457375454144775485526137614788  
54083263508172768788159683251684688493006254857641112501624145523391829  
27162507656772727460097082714127730434960500556347274566628060099924037  
10299142447229221577279853172703383938133469268413732762200096667667183  
1831088373420823444370953**

While calculating n, it becomes 309 digits

Similarly  $\phi$  becomes 309 digits as

**$\phi = 11593504173967614968892509864615887523771457375454144775485526137614788$   
54083263508172768788159683251684688493006254857641112501624145523391829  
27162507656751054233608492916752034482627988117554787657013923444405716  
98958172819609822636107546721186461217135910735864061400888517026537727  
7264467341066243857664128**

**e = 35535**

**d = 58008302860037763936093661289677917594669062089650962180422866111380593852  
82235873170628691003002171085904433840217072986908760061153062025249598844  
48047568240966247081485817130463240644077704833134010850947385295645071936  
77406119732655742423721761767462077637164207600337085333288532144708859551  
36670294831**

By choosing  $e = 35,535$  the value of  $d$  is computed

Now, let Alice wants to send the message “THIS IS A TEST” which can be changed to a numeric value by using the 00-26 encoding scheme (26 is the space character)

**P = 1907081826081826002619041819**

**C =** 4753091236462268272063655506105451809423717960704917165232392430544529  
6061319932856661784341835911415119741125200568297979457173603610127821  
8847892741566090480023507190715277185914975188465888632101148354103361  
6578984679683867637337657774656250792805211481418440481418443081277305  
9004692874248559166462108656

**P =** 1907081826081826002619041819

# Asymmetric vs. Symmetric Cryptography

# Key Length for Encryption

112 Bits of 3DES

-----

2048 Bits of RSA

128 Bits of AES-128

-----

3072 Bits of RSA

192 Bits of AES-192

-----

7680 Bits of RSA

# Need of Both

**Asymmetric key** cryptography does not eliminate the need for **symmetric-key** cryptography; Both are essential because one complements the other

Asymmetric-key cryptography, is much slower than symmetric key cryptography

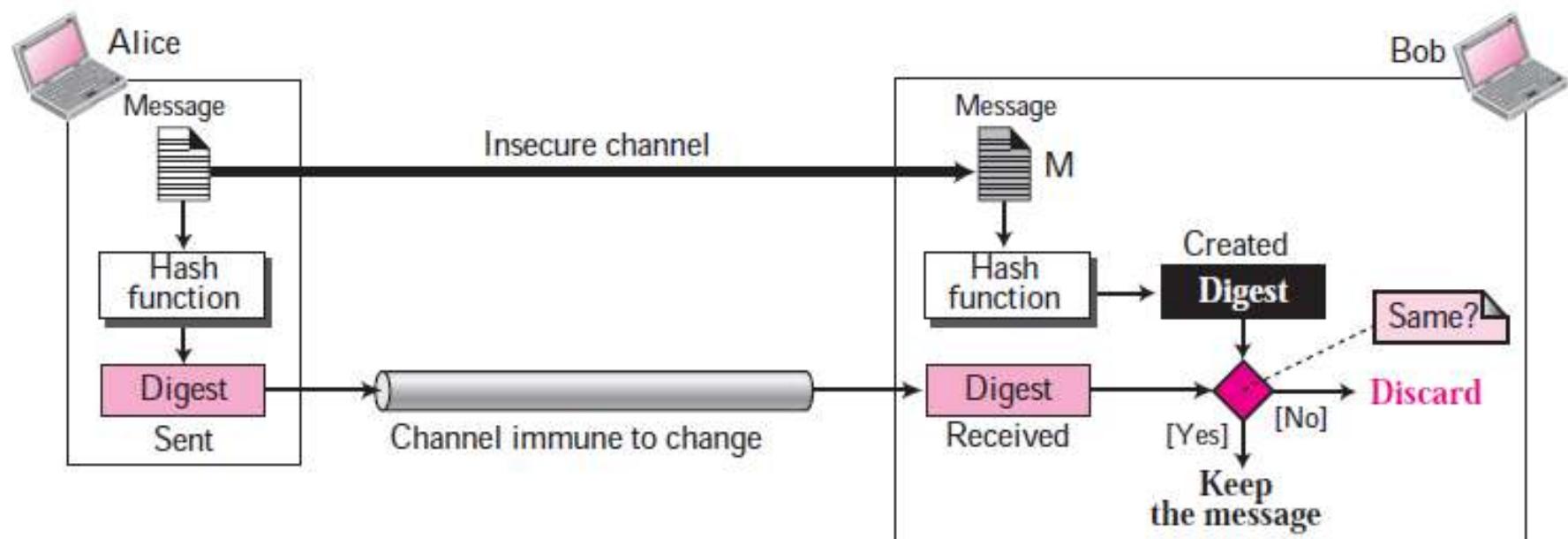
For encipherment of large messages, symmetric-key cryptography is still needed

On the other hand, the speed of symmetric-key cryptography does not eliminate the need for asymmetric-key cryptography

Asymmetric-key cryptography is still needed for authentication, digital signatures, and secret-key exchanges

# Message Integrity

# Message and Message Digest



**The message digest needs to be safe from change.**

# Hash Functions

Take a message of arbitrary length

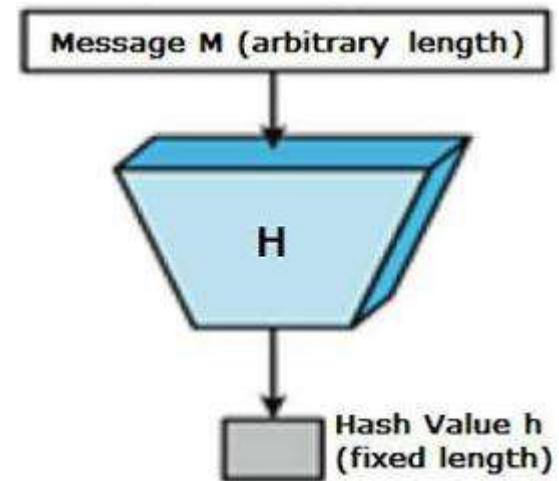
Creates a message digest of fixed length

Message size can be varied but the digest output is of fixed size

Several hash algorithms were designed by Ron Rivest: MD2, MD4, MD5

Secure Hash Algorithm (SHA) is a standard developed by the NIST

Different versions of SHA: SHA0, SHA1, SHA2, SHA3



# Hash Function Properties

Can be applicable for any sized message  $M$

Produces fixed-length output  $h$

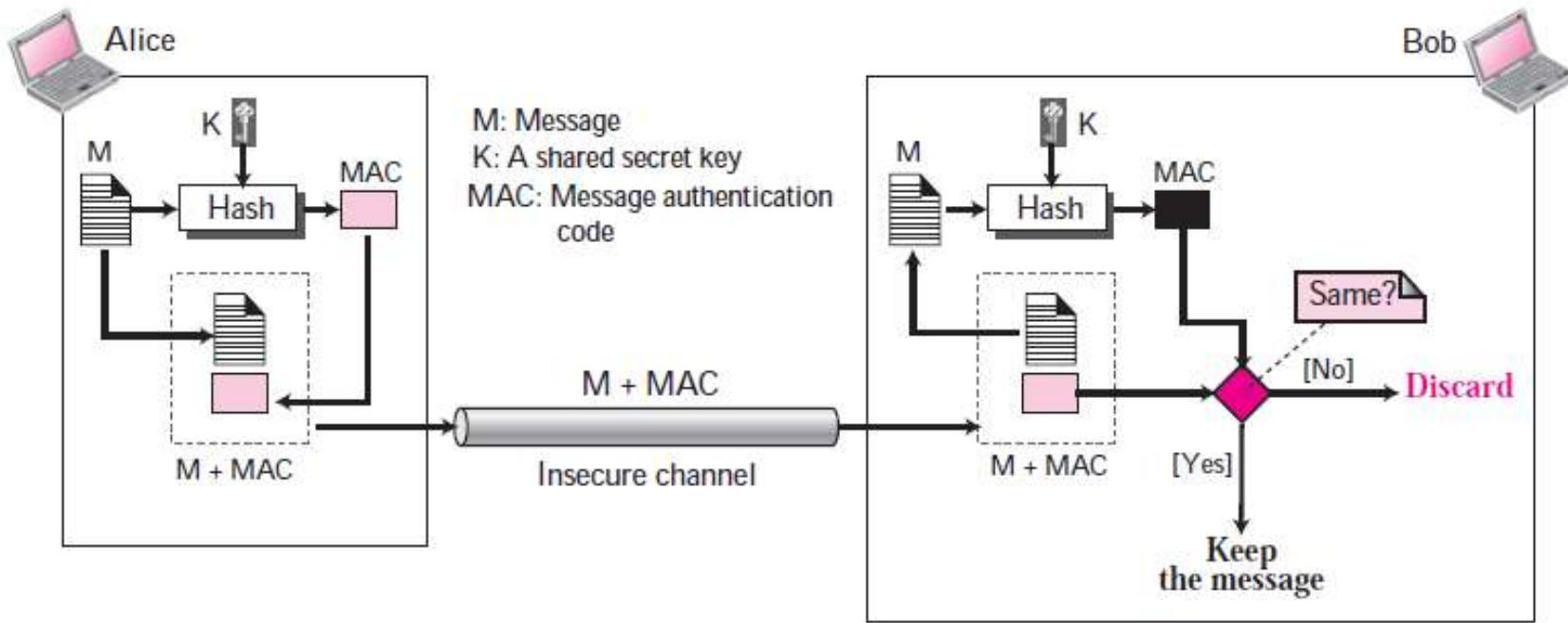
Easy to compute  $h = H(M)$  for any message  $M$

**One-way property**:- Given  $h$ , it is infeasible to find  $x$  s.t.  $H(x) = h$

**Collision resistance**:- Given  $x$ , it is infeasible to find a  $y$  s.t.  $H(y) = H(x)$

Similarly, It is infeasible to find any  $x,y$  s.t.  $H(y) = H(x)$

# Message Authentication Code (MAC)



# Requirements for MAC

Given a message and a MAC, it should be infeasible to find another message with same MAC

MAC should depend equally on all bits of the message

# MAC Properties

A MAC is a cryptographic checksum

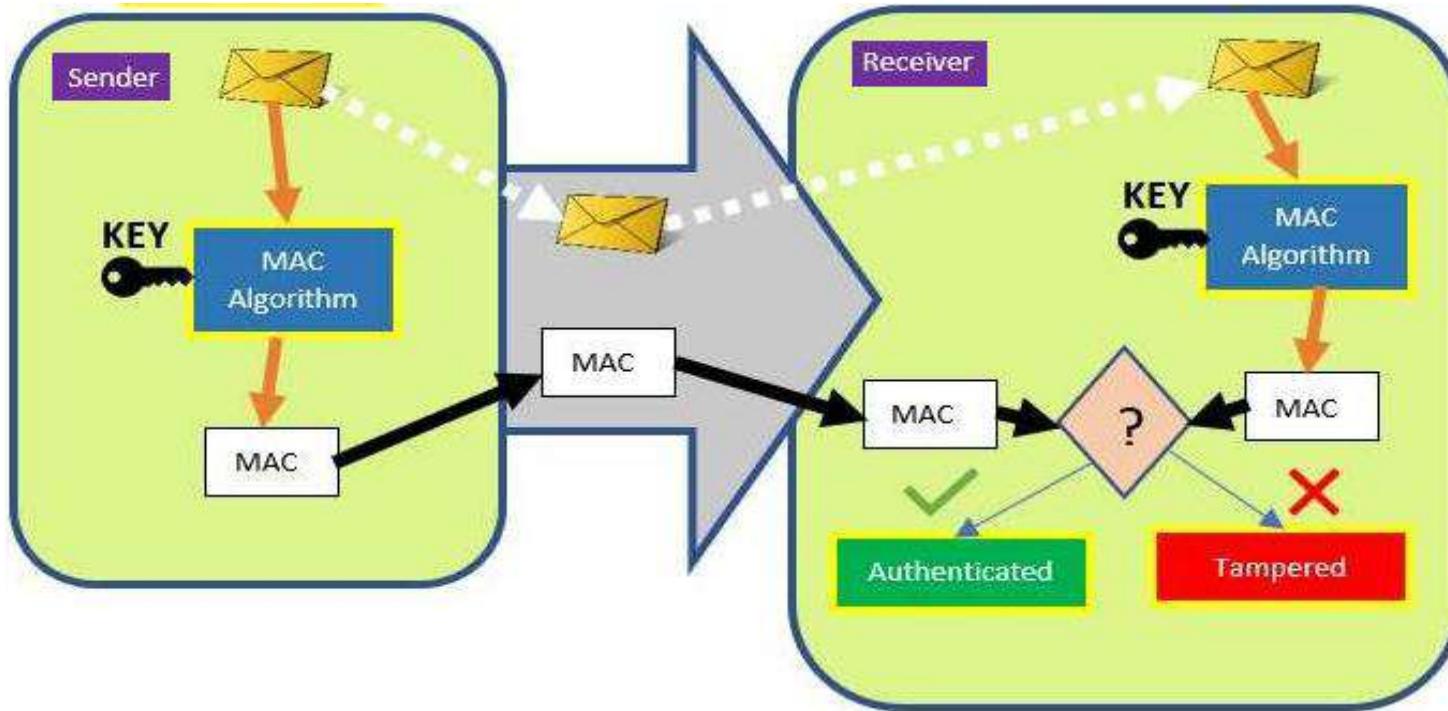
- $\text{MAC} = C_K(M)$

Can accept variable-length message M

Using a secret key K to a fixed-sized authenticator

Potentially many messages may have same MAC but finding these needs to be very difficult

# Message Authentication using MAC



# Digital Signatures

**Message authentication**

**Data Integrity**

**Non-repudiation**

# Digital Signature

Required Conditions:

- Receiver can verify claimed identity of sender
- Sender cannot later repudiate contents of message

# Digital Signature

A standard element of most cryptographic protocol suites, and are commonly used for software distribution, financial transactions and in other cases where it is important to detect forgery or tampering

Often used to implement electronic signatures, which include any electronic data that carries the intent of a signature

Employ asymmetric cryptography

# Public-Key Signatures

Sender encrypts message with private key

Receiver decrypts with sender's public key

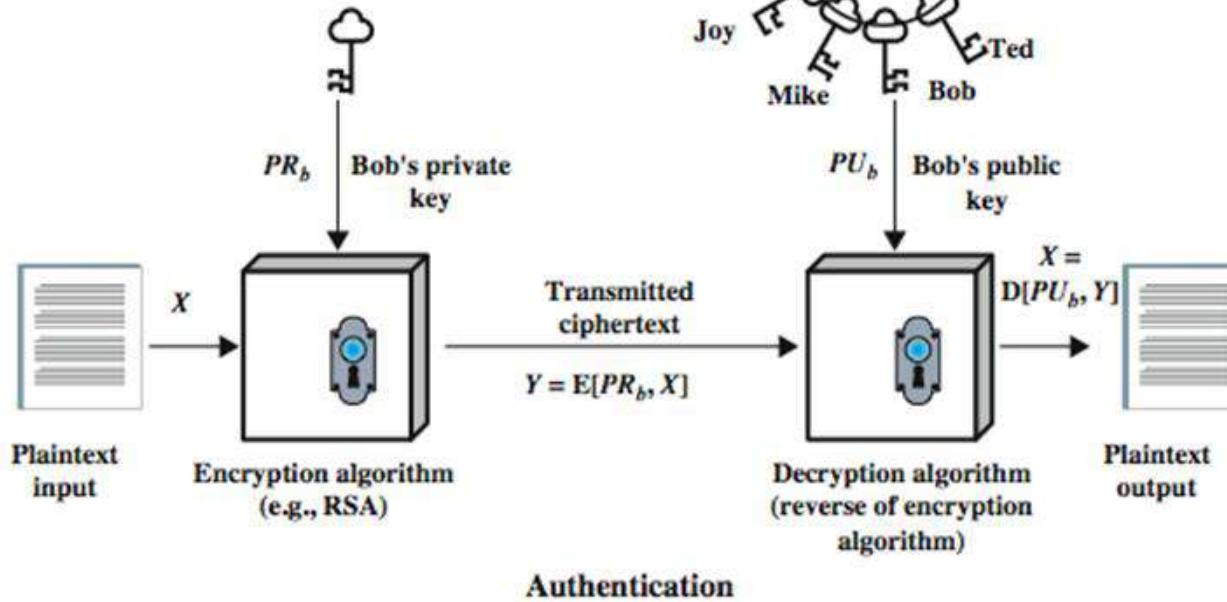
Authenticates sender

Does not give privacy of data

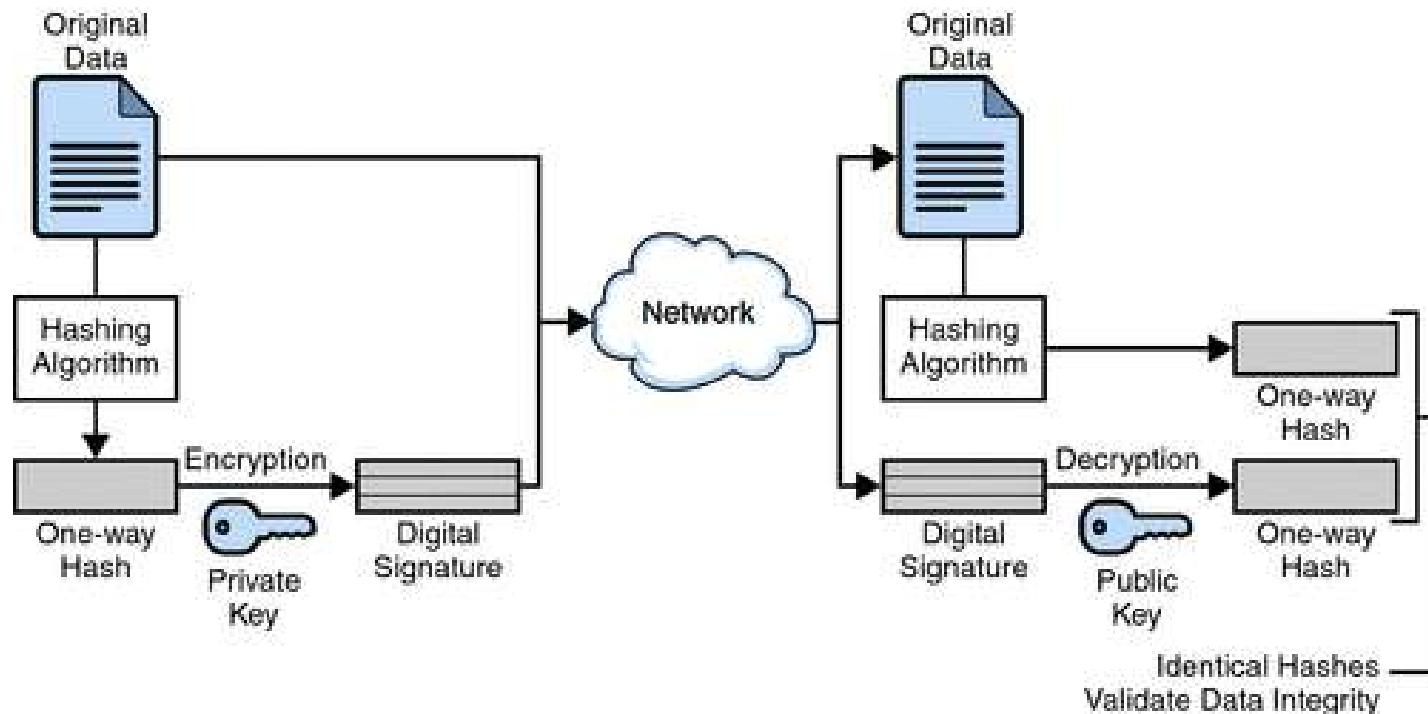
More efficient to sign authenticator

- A secure hash of message
- Send signed hash with message

# Public-Key Signatures



# Digital Signature



# Distributed and Edge Computing

# Distributed Systems Security

# Contents

Security issues and techniques in distributed system

Security challenges in edge computing

Introduction to digital forensics

# Security issues and techniques in distributed system

# Characteristics of Distributed Systems

Multiple entities

Openness

Heterogeneity

Scalability

Concurrency

Transparency

Resource sharing

# Challenges of Distributed System

Synchronization

Fault Tolerance

**Security**

# Distributed Systems Security

# Distributed Systems Security

Securing distributed systems is crucial for ensuring data integrity, confidentiality, and availability across interconnected networks

In the past, security was typically handled on an end-to-end basis, since all the work involved in ensuring safety occurred “within” a single system and was controlled by one or two or few administrators

The distributed systems has created a new ecosystem that brings with it unique challenges to security

As they have multiple nodes working together to achieve a common goal

# Distributed Systems Security

Security in a distributed system poses unique challenges that need to be considered when designing and implementing systems

A compromised computer or network may not be the only location where data is at risk; other systems or segments may also be infected with malicious code

As any threat can occur anywhere, even across distances in networks with few connections between them

# Distributed Systems Security

**Security Threats**

**Security Policy**

**Security Mechanisms**

# Security Threats

Different types of security threats:

- Interception
- Interruption
- Modification
- Fabrication

# Interception

Refers to the situation that an unauthorized party has gained access to a service or data

It also happens when data are illegally copied

# Interruption

Refers to the situation in which services or data become unavailable, unstable, destroyed and so on

Example: Denial of service attacks

# Modification

Involves unauthorized changing of data or tampering with a service so that it no longer adheres to its original specifications

# Fabrication

Refers to the situation in which additional data or activity are generated that would normally not exist

# Security Policy

A description of security requirements, is needed that is known as security policy

A security policy describes precisely which action the entities in a system are allowed to take and which ones are prohibited

Entities include users, services, data, machines, and so on

# Security Mechanisms

**Important security mechanisms:**

- Encryption
- Authentication
- Authorization
- Auditing

# Encryption

Fundamental to computer security and important to protect data

Essential for ensuring confidentiality and integrity of the sensitive data

Need to protect sensitive data from unauthorized access, modification, or theft as well as ensure the privacy of data

Transforms plaintext data into ciphertext using algorithms (e.g., RSA, AES) so that attacker cannot understand

Ensures data confidentiality during transmission and storage

# Encryption

Allows us to check whether data have been modified unauthentically ⇒ supports for integrity checks

Transport Layer Security (TLS) / SSL be used as standard protocol to encrypt data exchanged between clients and servers. Establishes a secure communication channels to prevent eavesdropping and tampering

Key Management: Manages encryption keys used to encrypt and decrypt data, that ensures secure storage, rotation, and distribution of keys to authorized entities

# Authentication

The process of recognizing a user's identity

Used to verify the claimed identity of a user, client, server, host or other entity

The mechanism of associating an incoming request with a set of identifying credentials

The credential often takes the form of a password, which is a secret and known only to the individual and the system

Typically, users are authenticated by means of passwords, but there are many other ways to authenticate clients

# Authentication Mechanisms

**Password-based Authentication:** Users authenticate with a username and password, commonly used approach but vulnerable to password breaches and phishing attacks

**Multi-factor Authentication:** Uses two or more authentication factors (e.g., password + OTP, fingerprint etc.), that enhances security by adding an extra layer of verification

**Biometric Authentication:** Uses unique biological parameters (e.g., fingerprints, facial recognition) for authentication, and it provides strong authentication but may require specialized hardware

**Certificate-based Authentication:** Uses digital certificates to authenticate clients and servers. Certificates are issued by trusted Certificate Authorities (CAs)

# Authorization

After a client has been authenticated, it is necessary to determine whether that client is authorized to perform the action requested

A security mechanism to determine access levels or user/client privileges related to system resources

Process of granting or denying access to a network resource which allows the user access to various resources based on the user's identity

# Authorization Controls

**Role-Based Access Control:** Assigns permissions based on roles (e.g., admin, user, manager). Simplifies access management by grouping users with similar responsibilities

**Attribute-Based Access Control:** Grants access based on attributes (e.g., user properties, resource properties)

**Access Control Lists (ACLs):** Lists of permissions associated with users or groups. Allows direct access to resources to specify who can access them and what actions they can perform

**Policy-Based Access Control:** Uses policies to define access rules based on conditions & attributes, it offers flexibility to adapt access controls dynamically based on changing conditions

# Auditing

Auditing tools are used to trace access contents and ways

Audit logs can be very useful for the analysis of a security breach, and subsequently taking measures against intruders

For this reason, attackers are generally keen not to leave any traces that could eventually lead to exposing their identity

In this sense, logging accesses makes attacking sometimes a riskier business

# Why Security Audit ?

- To verify that the current security strategy is adequate or not
- To check that the security training efforts are working
- To uncover any extraneous hardware and software
- To uncover the flaws introduced by new technology or processes
- To prove the organization is compliant with regulations

# Security Challenges in Edge Computing

# Edge computing

Latency is a problem where nearly instantaneous transfer of information is necessary

Edge computing addresses this problem by:

- Moving the task of initial data processing to connected devices
- Using edge data centers in place of central servers

# Security Issues in Edge Computing

# Data Security and Privacy Requirements

Confidentiality

Integrity

Availability

Authentication and access control

Privacy requirement

# Data Security and Privacy Challenges

Edge computing utilizes different technologies to put the computing in the proximity of data sources

The data security and privacy-preserving have become the basic requirements to protect end users in their business, economics, and daily life

We must admit that security and privacy should be addressed in every layer in designing edge computing systems

# Challenges in edge computing

## Core Infrastructure:

- Privacy leakage
- Data tampering
- Denial of service
- Service manipulation

## Edge Network

- Denial of service
- Man-in-the-middle

## Edge Servers:

- Privacy leakage
- Denial of service
- Service manipulation
- Physical damage

## Mobile Edge Devices

- Injection of information
- Service manipulation

# Data Security and Privacy Mechanisms

- Data confidentiality
- Data integrity
- Secure data computation
- Authentication
- Access control
- Privacy preserving

# Introduction to digital forensics

# Forensic

Forensic Science is the application of scientific methods to establish factual answers to legal problems

Crime reconstruction is the determination of the actions and events surrounding the commission of a crime

An investigation is a systematic examination, typically with the purpose of identifying or verifying facts

A key objective during investigations is to identify key facts related to a crime or incident, and a common methodology used is referred to as 5WH

# What is digital forensics?

Emerging discipline in computer security

Investigation that takes place after an incident has happened

Try to answer questions: Who, what, when, where, why, and how

Determine what the incident was and get back to a working state

Internal investigation

Criminal investigation

Support for “real world” investigations

# Digital Forensics

Conventionally the forensics was limited to the recovery and analysis of biological and chemical evidence during criminal investigations

There is an increased adoption of digital devices and growing incidence of cyber crime

The art of recovering and analysing the contents found on digital devices such as desktops, laptops, tablets, smartphones, etc. for the purpose of investigation whenever an incident is happened

# Digital Forensics

Digital forensics refers to forensic science applied to digital information, whereas a digital investigation refers to investigations in the digital domain

The use of scientifically derived and proven methods toward the preservation, collection, validation, identification, analysis, interpretation, documentation, and presentation of digital evidence derived from digital sources for the purpose of facilitating or furthering the reconstruction of events found to be criminal, or helping to anticipate unauthorized actions shown to be disruptive to planned operations

# Digital Forensics

The practice of collecting, analysing and reporting on digital data in a way that is legally admissible

Can be used in the detection and prevention of crime and in any dispute where evidence is stored digitally

Use of specialized techniques for recovery, authentication and analysis of electronic data when a case involves issues relating to reconstruction of computer usage, examination of residual data, and authentication of data by technical analysis or explanation of technical features of data and computer usage

# Digital Forensic Requirements

Requires specialized expertise that goes beyond normal data collection and preservation techniques available to end-users or system support personnel

Similar to all forms of forensic science, it is comprised of the application of the law to computer science

Deals with the preservation, identification, extraction, and documentation of computer evidence

# Digital Forensic Requirements

Also involves the use of sophisticated technological tools and procedures that must be followed to guarantee the accuracy of the preservation of evidence and the accuracy of results concerning computer evidence processing

Uses specialized techniques for recovery, authentication, and analysis of computer data, typically of data which may have been deleted or destroyed

# Forensic Process

# Forensic Process

Defines a structured investigation of digital evidence from any device capable of storing or processing data in a digital form

Many similarities to physical investigation processes

But the evidence of interest is digital in this case

Traditional forensics processes are challenged by how to gather what evidence to support a hypothesis of a crime or incident

The process needs to ensure that whatever digital evidence is identified, it must be managed properly for it to prove a case

# Forensic Process

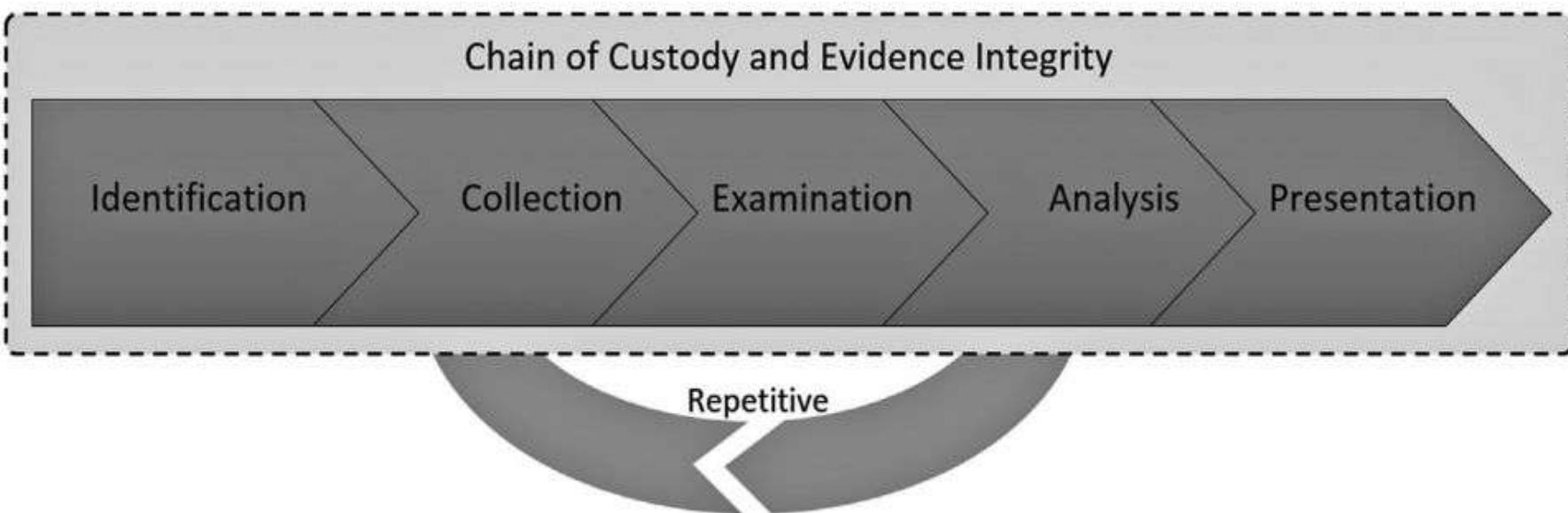
Accused murderers, robbers, and other criminals may have their electronic devices analyzed in an investigation to identify evidence about a crime

Such evidence can provide answers to the 5WH questions of investigations

Evidence may be found in diverse places like in a chat session, communication log, or any other digital traces. In the case of corporate investigations, loss or misuse of corporate confidential information is typically discovered by employees, customers, partners, or incident management analysts using security solutions and tools

The digital forensics process can thus be used in criminal investigations, corporate investigations, or even private investigations

# The Digital Forensics Process



# Identification Phase

Basis for all the following phases or activities during the digital investigation

Task of detecting, recognizing, and determining the incident or crime to investigate

Incidents can be identified based on complaints, alerts, or other indications

Can be used to identify which evidence or objects to look for the investigation

The identification of an incident or a crime leads to the formation of a hypothesis about what might have happened

A digital forensics investigation is a response to an observed incident and the first responder must act accordingly

# Example crime scene with multiple digital devices



# Collection Phase

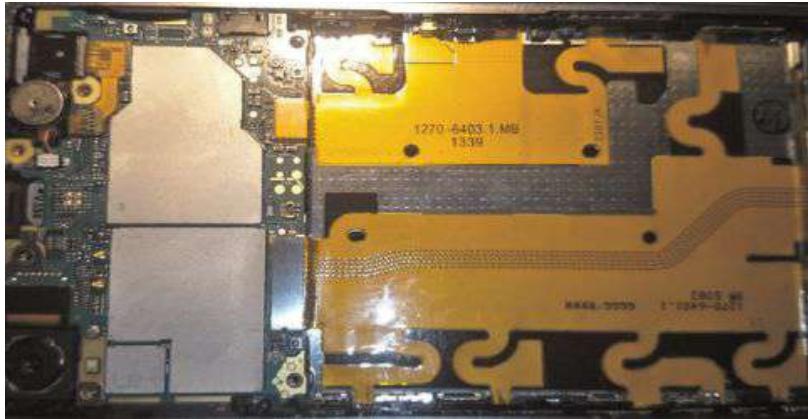
Refers to the acquisition or copying of the data

Collection of data from digital devices to make a digital copy using forensically sound methods and techniques

This is when a forensic investigator gains access to the electronic device(s) containing raw data that has been identified as relevant for the specific case

The collection phase of the digital forensics process is common to most literature and scientific research in digital forensics

# Sources of Digital Evidence



## Examples of order of volatility.

| Type of storage media and data                  | Typical storage lifespan and longevity<br>(dependent on usage) |
|---|--|
| System registers, peripheral memory, and caches | Nanoseconds  |
| RAM   | Ten nanoseconds  |
| Network state                                   | Milliseconds   |
| Running system processes                        | Seconds  |
| Data on disk (cache)                            | Minutes  |
| Cloud storage                                   | Months to years  |
| HDD data storage                                | Years  |
| Floppies and other magnetic tape-based media    | Years to decades   |
| CD-ROMs, DVDs, print-outs,                      | Decades  |
| Read-only memory; flash and SSD data storage    | Decades to centuries   |

# Examination Phase

Preparation and extraction of potential digital evidence from collected data sources

Collected data must be examined and prepared for later analysis as part of the examination phase

it is important to document the actions and handling of the data to support the chain of custody

The examination often requires restructuring, parsing, and preprocessing of raw data to make it understandable for a forensic investigator in the upcoming analysis

To facilitate this phase, an analyst typically uses forensic tools and techniques appropriate for extracting relevant information



# Analysis Phase

In this phase forensic investigators determine the digital objects to be used as digital evidence to support or refute a hypothesis of a crime, incident, or event

The processing of information that addresses the objective of the investigation with the purpose of determining the facts about an event, the significance of the evidence, and the person(s) responsible

Statistical methods manual analysis, techniques for understanding protocols and data formats, linking of multiple data objects (e.g., through the use of data mining), and timelining are some of the techniques that are used for analysis

As for all other investigative phases, the chain of custody is also important for the preservation and traceability of the collected data in the analysis phase

# Presentation Phase

The process by which the examiner shares results from the analysis phase in the form of reports to the interested party or parties

The presentation phase involves the final documentation and presentation of the results of the investigation to a court of law or other applicable audiences, such as a corporation's top management or crisis management team

The presentation is based on objective findings with a sufficient level of certainty, based on the analysis of digital evidence

It is important that the findings are summarized and that all actions performed during the investigation are accounted for and described in a fashion understandable by the audience

*Wish you all the best for your final examination*