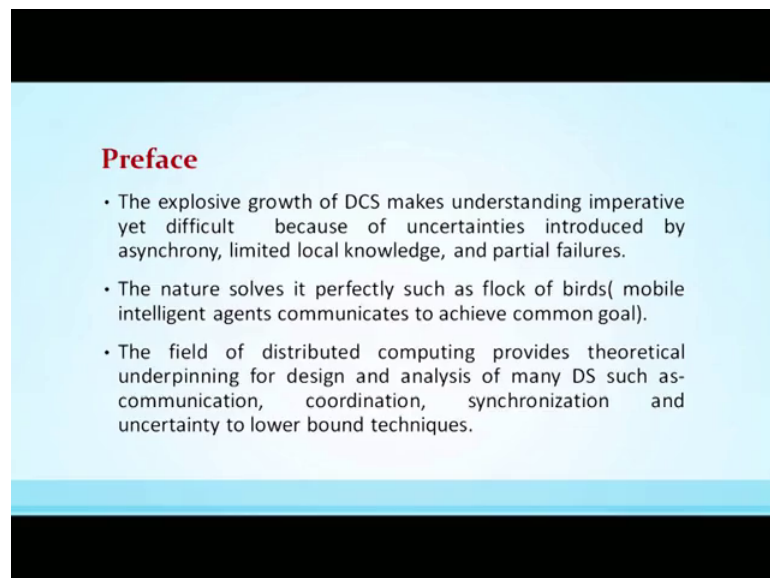


**Distributed Systems**  
**Dr. Rajiv Misra**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Patna**

**Lecture - 01**  
**Introduction to Distributed Systems**

So this is the first lecture Introduction to the Distributed Systems.

(Refer Slide Time: 00:21)



**Preface**

- The explosive growth of DCS makes understanding imperative yet difficult because of uncertainties introduced by asynchrony, limited local knowledge, and partial failures.
- The nature solves it perfectly such as flock of birds( mobile intelligent agents communicates to achieve common goal).
- The field of distributed computing provides theoretical underpinning for design and analysis of many DS such as- communication, coordination, synchronization and uncertainty to lower bound techniques.

In this particular distributed system lecture introduction, we are going to discuss about the different requirements of a distributed computing systems and we are going to discuss, what are the different topics we are going to cover the textbooks and so on so forth.

So, before that let us begin with the preface. So, the explosive growth of distributed computing systems makes understanding imperative yet difficult because of uncertainties introduced by the asynchrony, limited local knowledge, and partial failures. The nature solves it perfectly, such as flock of birds where these birds are the while agents they communicates with each other to achieve a common goal.

However, in the field of distributed computing providing all these intricacies, that is asynchrony, limited local knowledge, partial failures. To understand this course will provide a theoretical underpinning for design and analysis of many distributed systems,

such as and the concepts such as communication, coordination, synchronization and uncertainty to the lower bounds techniques. These together will be discussed in the part of the course. And this particular course will be quite useful as far as the different applications are concerned.

The course structure of a distributed system goes like this.

(Refer Slide Time: 02:00)

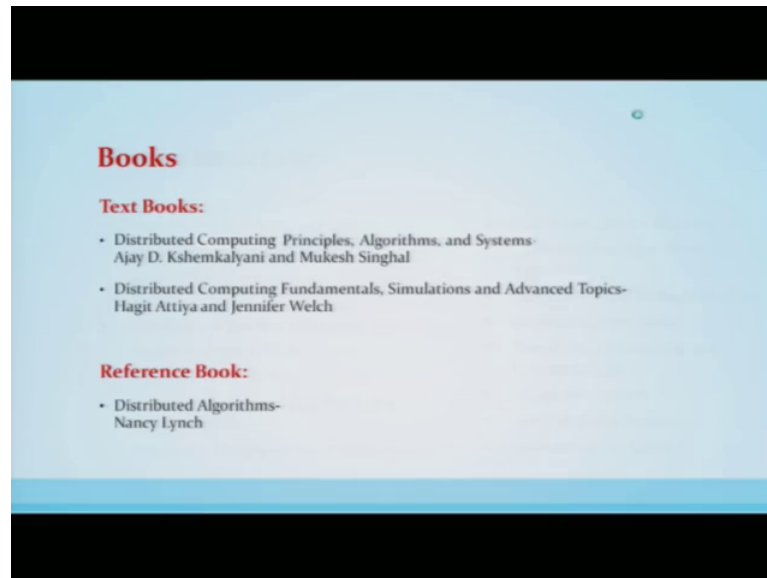


So, this particular course if you see is divided into two different parts. The first part is the perspective from the systems perspective distributed systems. The second part is called distributed systems from algorithms perspective. So, the algorithms which will run on this model of a distributed systems is required to be understood here. So, model also is required to be understood. And then the algorithms: how to design these algorithms, how to analyze it and different intricacies of this algorithm design in this particular problem setting.

So, the main topics which we are going to focus on from algorithms perspective means how to build the spanning trees using flooding algorithms, then the leader election algorithm. These are basically most of the important algorithmic design techniques they are the basic building blocks of the distributed systems. From systems perspective we are going to cover up the global state recording, mutual exclusion, consensus, shared memory, check pointing rollback, distributed hash table. And the case studies of a

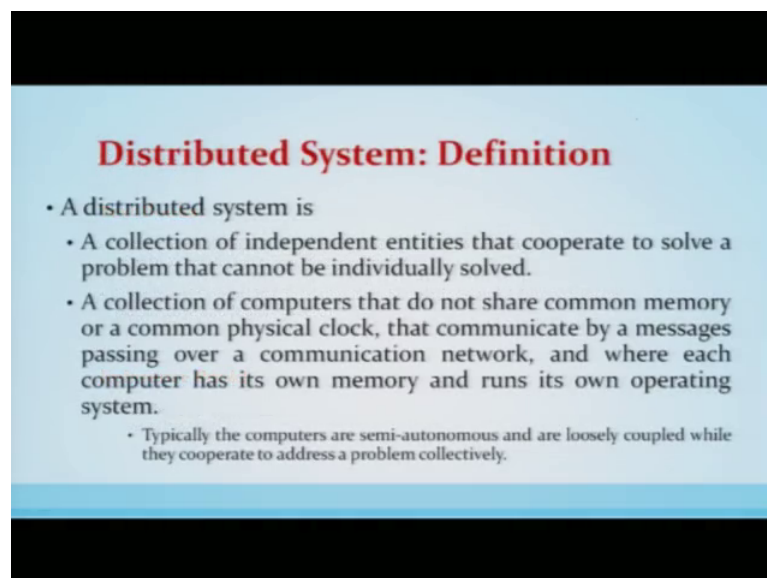
distributed systems which we will cover here in this part of the course structure is peer to peer Google file system HDFS and introduction to the spark.

(Refer Slide Time: 03:19)



With this particular course we will use two textbooks. The first we will deal about the systems perspective, the first one mentioned here as the authors Kshemkalyani and Singhal. The other textbooks will deal with the algorithms perspective and that is by Jennifer Welch. We have the reference book also that is distributed algorithms by Nancy Lynch.

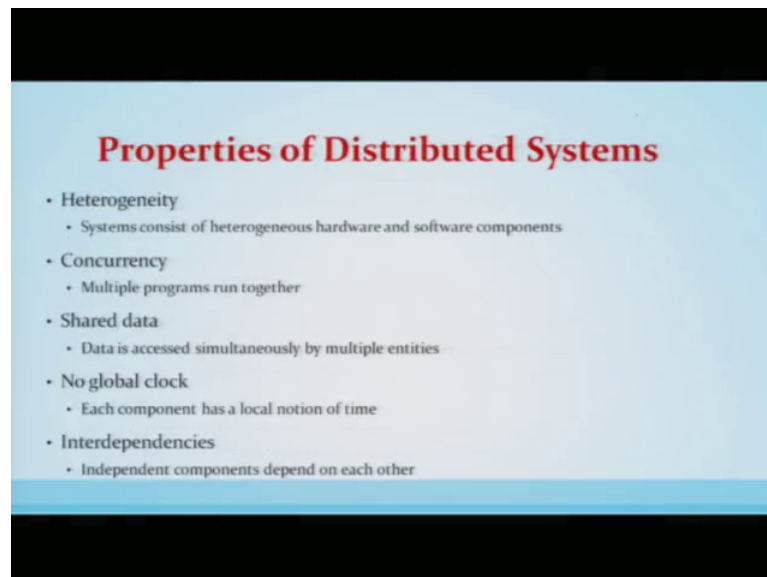
(Refer Slide Time: 03:47)



Let us begin with the definition of a distributed system. Distributed system is a collection of independent entities that cooperate to solve a problem that cannot be solved individually. So, basically it is nothing but a collection of computers. Thus, this particular collection do not share a common memory or do not have a common physical clock, and the only way they can communicate is through the message passing and for that they require a communication network.

The computers used here in distributed systems are semi-autonomous and they are loosely coupled while they cooperate to address the problem collectively.

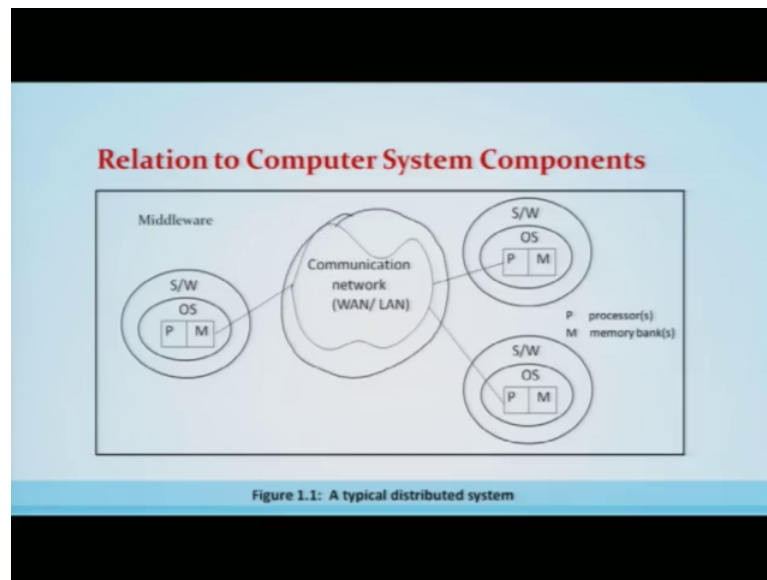
(Refer Slide Time: 04:33)



So, before we understand in more detail about the distributor system, let us have some properties of distributed system to keep in a mind at this point of time. So, heterogeneity is one of the properties, because here the system comprises of different computers autonomous computers and they may be heterogeneous having heterogeneous hardware and software components.

The concurrency is another property of a distributed system, shared data is also another property of a distributed system. So, no global clock is also one of the important properties and inter dependencies are there they depend interdependent components depend on each other.

(Refer Slide Time: 05:17)



Now, to understand the distributed system from the system perspective, let us see this particular figure or diagram. In this particular diagram, you can see the computers are autonomous computers you are presented as processor memory with the operating system and basically the communication protocol stack.

And, this particular these different computers they can communicate through the network that is the communication network. Now, as far as the software is concerned which will build a distributed system this particular software is called basically the middleware and the part of these middleware is basically the software which runs on each computer they are called software's; they are written as the software components.

So, basically this particular distributed system software will basically use the existing computers their operating system and underlying computer network and they run the part of the middleware and together this will form a distributed system. So, middleware will bind the distributed system.

(Refer Slide Time: 06:25)

### Relation to Computer System Components

- A DS connects autonomous processors by communication network which cooperate to run application network.
- The software components that run on each of the computers use the local operating system and network protocol stack for functioning.
- The distributed software is also termed as middleware.
- A distributed execution is the execution of processes across the distributed system to collaboratively achieve a common goal. An execution is also sometimes termed a computation or a run.

So, again further explain the distributed system connects autonomous processors by communication network. And the software component that run on each of the computers use the local operating system and network protocol stack. The distributed software is termed as middleware.

The distributed execution is the execution of the processes across the distributed system to collectively achieve a common goal. The execution is also sometimes termed as the computation or error in a distributed system.

(Refer Slide Time: 06:57)

### Layered Architecture

- The middleware(layered architecture) is the distributed software that drives the distributed system, while providing transparency of heterogeneity at the platform level.

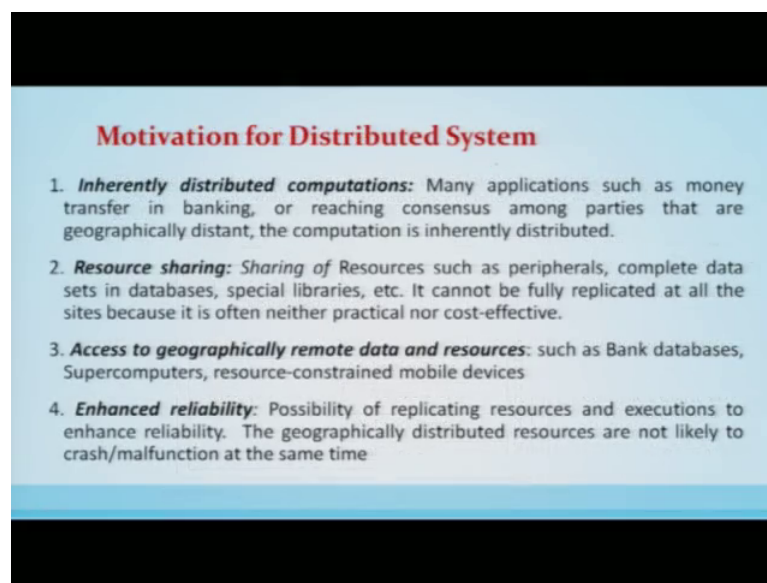
**Several standards as :**  
OMG (Object Management Group )  
CORBA ( Common Object Request Broker Architecture)  
RPC (Remote Procedure Call)  
DCOM (Distributed Component Object Model)  
RMI (Remote Method Invocation)  
MPI (Message-Passing Interface)

Figure 1.2: Layered Architecture of Distributed System Application

Furthermore; the distributed software which is also called a middleware is designed in a layered architecture to simplify the complexity of the distributed software. And this particular middleware or the distributed software that drives the distributed system, it also provides the concurrency of heterogeneity at the platform level.

So, in the diagram you can see the distributed application mentioned over here. So, this particular distributed application will use the distributed software which is a middleware; middleware runs on the operating system of each collection of computers and also it will use the underlying network protocol stack for the communication. And there are several standards also evolved over a time for this particular middleware application; middleware for the distributed software development. That is OMG, CORBA, RPC, DCOM, RMI, MPI, and so on.

(Refer Slide Time: 08:02)

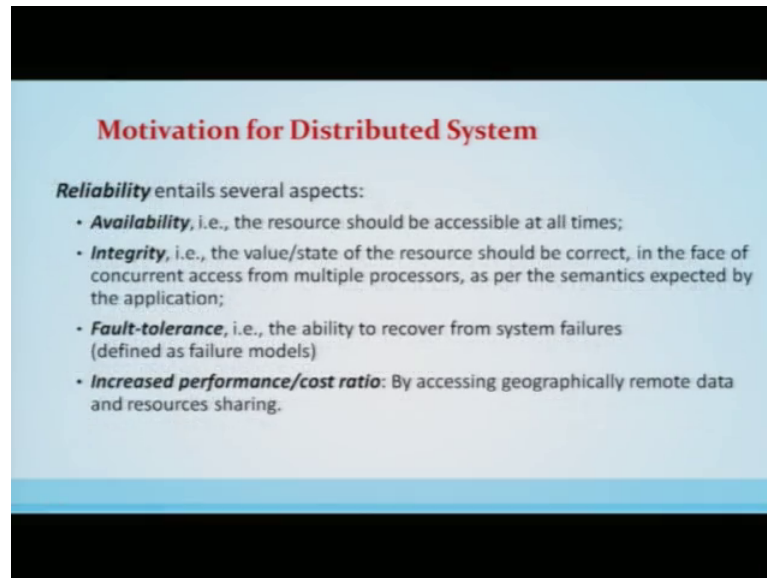


So, that was the overview of the distributed system from a system perspective. Now, we are going to touch upon the motivation of the distributed system. So, inherently distributed computation that is many applications such as money transfer in the banking, or reaching a consensus among the parties that are geographically distant, the computation is inherently distributed. So, for that the model that is distributed system is required for that computation that is the applications which are inherently distributed.

Then, next is the motivation called resource sharing the sharing of the resources such as peripherals, and a complete data set and so on and so forth; is a basically the motivation

behind this building of distributor system. Another motivation is to access the geographically remote data and resources, such as bank database, supercomputer and so on. Reliability: enhanced reliability possibility of replicating the resources and execution to enhance the reliability. Geographically distributed resources are not likely to crash at the same time. That is the motivation of building the distributed system for that.

(Refer Slide Time: 09:19)



**Motivation for Distributed System**

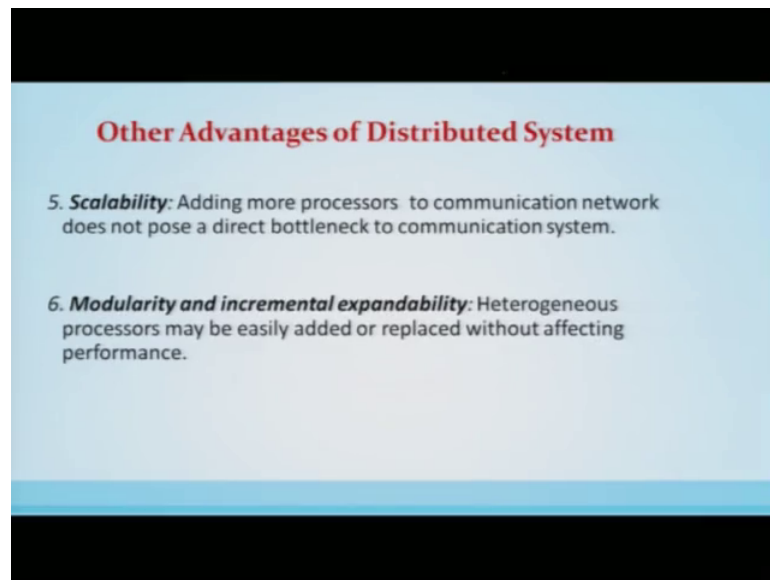
**Reliability** entails several aspects:

- **Availability**, i.e., the resource should be accessible at all times;
- **Integrity**, i.e., the value/state of the resource should be correct, in the face of concurrent access from multiple processors, as per the semantics expected by the application;
- **Fault-tolerance**, i.e., the ability to recover from system failures (defined as failure models)
- **Increased performance/cost ratio**: By accessing geographically remote data and resources sharing.

So, reliability entails several aspects in that case. So, these are basically the availability the resources should be accessible at all the times. Integrity the value or oblique the state of the resource should be correct, in the face of concurrent access, and fault-tolerance the ability to recover from system failures. Increased performance oblique cost ratio by accessing geographically remote data and resource sharing. So, these are basically the reliability will entail these aspects.

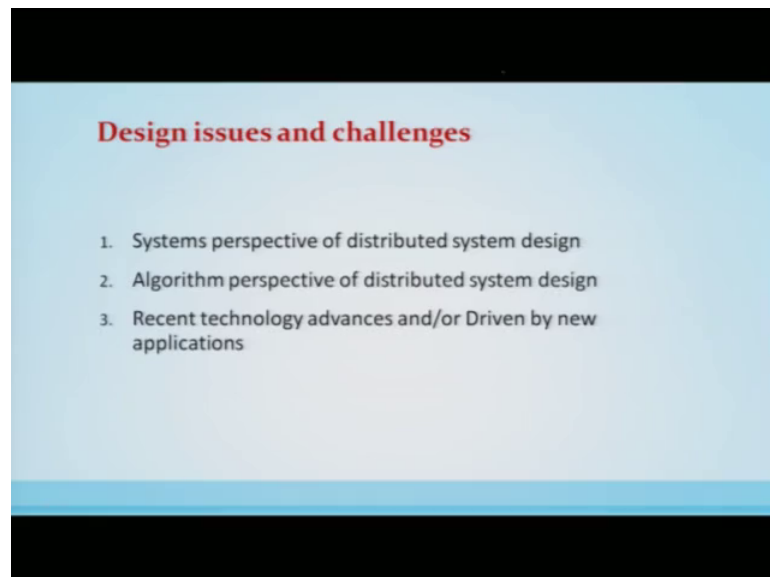


(Refer Slide Time: 10:02)



So, basically other advantage of distributed system is scalability, adding more processor to the communication network does not pose a bottleneck to the communication network. Then, the next advantage is modularity and incremental expandability. So, here the process or heterogeneous processor can be added without any bottleneck problems.

(Refer Slide Time: 10:18)

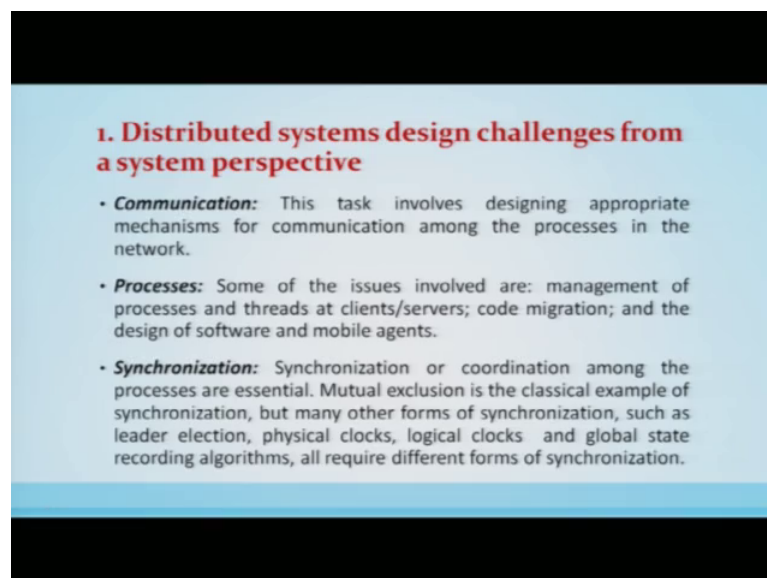


Now, we are going to discuss the design issues and challenges in the distributed system design.

So, from system perspective of a distributed system design, we are going to see; what are the intricacies and we have to understand the theoretical basis for this design. Another thing is algorithmic perspective of it is distributed system design. The next design issues and challenges is based on the recent technology advances and also driven by the new applications and which will basically be the motivation or be the design issues. And also becoming a challenges in evolving the distributed systems.

So firstly, we are going to look upon the design challenges from system perspective of distributed systems.

(Refer Slide Time: 11:14)



**1. Distributed systems design challenges from a system perspective**

- **Communication:** This task involves designing appropriate mechanisms for communication among the processes in the network.
- **Processes:** Some of the issues involved are: management of processes and threads at clients/servers; code migration; and the design of software and mobile agents.
- **Synchronization:** Synchronization or coordination among the processes are essential. Mutual exclusion is the classical example of synchronization, but many other forms of synchronization, such as leader election, physical clocks, logical clocks and global state recording algorithms, all require different forms of synchronization.

Here, the components which are involved here in the systems perspective are the communications that is the communication network where the processors can basically communicate with each other through which processors can communicate. Processes some of the issues involved are: the management of the processes and the threads at the client server; code migration, design of software mobile agents. Synchronization is the most important part.

Synchronization or the coordination among the processes are essential. Mutual exclusion is an example of synchronization, but many other forms of synchronization, such as leader election, physical clocks, logical clocks global state recording algorithms, all require different form of synchronization that we are going to cover up in this part of the course in more details.

(Refer Slide Time: 11:59)

**1. Distributed systems challenges from a system perspective**

- **Fault tolerance:** Requires maintaining correctness in spite of failures of links, nodes, and processes.
- Process resilience, reliable communication, distributed commit, check-pointing and recovery, agreement and consensus, failure detection, and self-stabilization are some of the mechanisms to provide fault-tolerance.

Now, another system level challenge is the fault tolerance.

So, this fault tolerance it requires maintaining correctness in spite of the failures of a links, nodes and processes. So, this particular fault tolerance is basically achieved using the process resilience, reliable communication, distributed commit, check pointing and recovery, agreement and consensus, failure detection, self-stabilization these are some of the techniques which we are going to cover up, when we discuss the design from systems perspective.

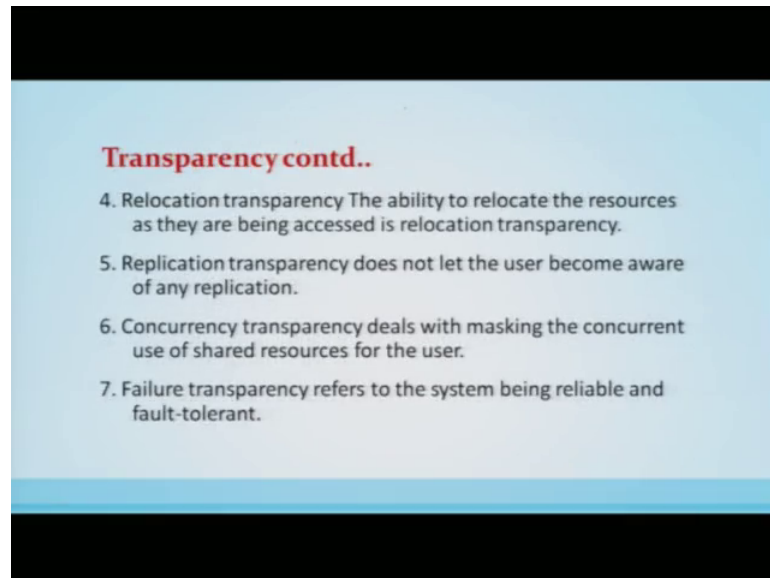
(Refer Slide Time: 12:38)

**1. Distributed systems challenges from a system perspective**

- **Transparency:** Hiding the implementation policies from the user can be classified as:
  1. Access transparency hides differences in data representation on different systems and provides uniform operations to access system resources.
  2. Location transparency makes the locations of resources transparent to the users.
  3. Migration transparency allows relocating resources without changing names.

Another system perspective design angle or aspect is transparency. So, transparency is to hide the implementation policies from the user and this can be a different kind of transparencies: the first one is called access transparency. When it hides the difference says in the data representation on different systems and location transparency when it makes the transparency of the location of the resources.

(Refer Slide Time: 13:13)



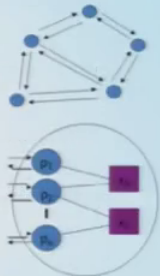
And migration transparency allows the relocating resources without changing the name. Relocation transparency the ability to relocate the resources as they are being accessed is relocation transparency. Replication transmit does not let the user become aware of any replication.

Concurrency transmits deal with masking the concurrent use of shared resources for the user. Failure transparency refers to the system being reliable and fault-tolerant. It is not known to the user at this at any point of time.

(Refer Slide Time: 13:38)

## Distributed Algorithms

- In Distributed Systems, different complexity measures are of interest such as: time, space but now considered communication cost (no. of messages, size, no. of shared variables) and the number of faulty vs. non-faulty components.
- Because of complications faced by DS leads to increase scope of "negative results", lower bounds and impossibility results.



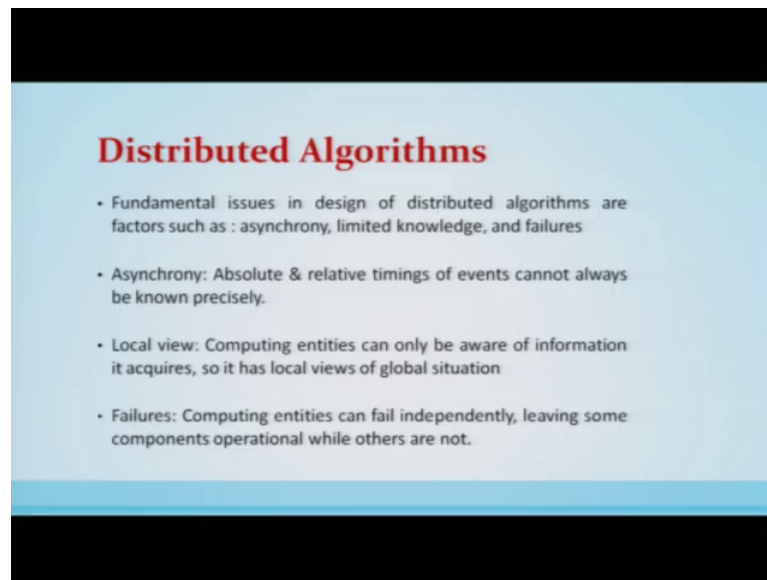
Now, that was the distributed system from system perspective. Now, we are going to touch upon another important component of this particular distributed system distributed computing system that is called distributed algorithms.

So, the algorithms are to be evolved. So, we are going to cover up the fundamental algorithms which will be the building basic building blocks of developing the distributed applications. So, in distributed applications, distributed systems, different complexity measures are of interest such as: the time and space. They were used in the classical or the sequential algorithms as well, but now communication is also evolved. So, communication cost is one of the complexity measure.

So, communication cost includes the number of messages, size of the message and number of share variables. And also another component which will be used in the complexity is called basically the number of faulty versus non-faulty components. Now because of the complications faced by distributed system they lead to the increase the scope of negative results, lower bounds and impossibility results.

So, all these things will be covered up in a form of a in the distributed algorithm design and thus we will discuss more these particular distributed algorithms in the details. So, the fundamental issues in the design of distributed algorithms are the following three factors.

(Refer Slide Time: 15:22)



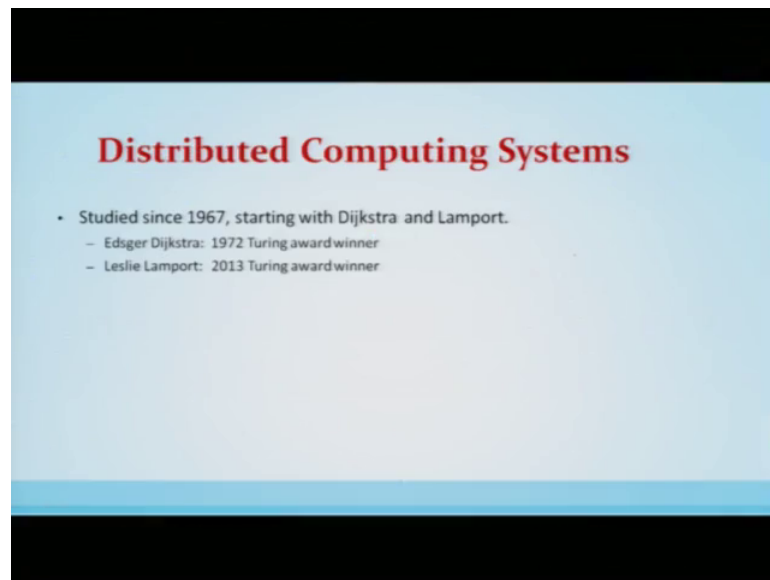
## Distributed Algorithms

- Fundamental issues in design of distributed algorithms are factors such as : asynchrony, limited knowledge, and failures
- Asynchrony: Absolute & relative timings of events cannot always be known precisely.
- Local view: Computing entities can only be aware of information it acquires, so it has local views of global situation
- Failures: Computing entities can fail independently, leaving some components operational while others are not.

The asynchrony: so asynchrony, limited knowledge and failures. There are three different important fundamental design issues in the algorithm distribute algorithm. Asynchrony is basically absolute and relative timing of the events cannot be known precisely.

So, in this particular setting how the algorithms are to be evolved develop. Local view that is the computing entities can only be aware of the information it acquires, so it has only the local view of a global situation. Third one is the failures. So, the computing entities can fail independently, leaving some components operational while others are not. So, these three different factors they add the more complications in design of the distribute algorithm and becomes a challenging to evolve the distribute algorithm in these particular problem setting, that is asynchrony that is we are not knowing the events when it they are going to occur. Local view we are not knowing the complete picture of the global situation yet we have to come up with an algorithm. Failures means the components which are basically involved in that distributed system they can fail independently, and basically this expected that the applications should basically keep on running in spite of failures.

(Refer Slide Time: 16:55)

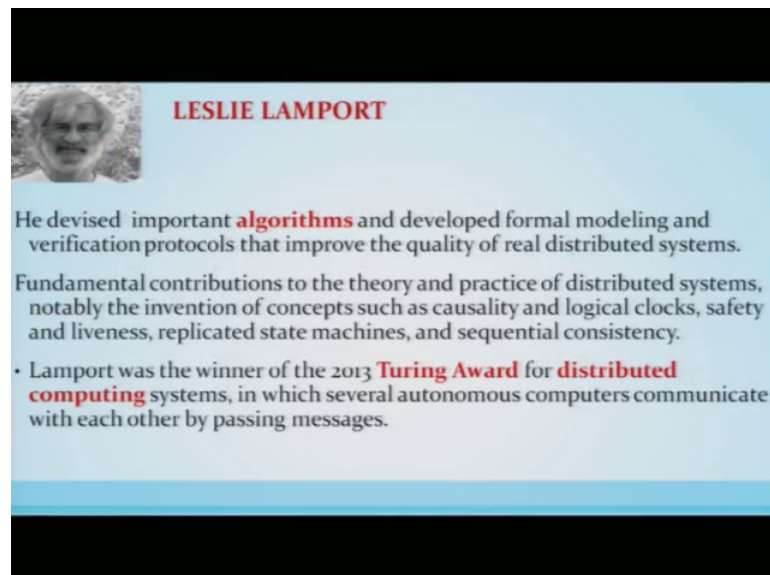



**Distributed Computing Systems**

- Studied since 1967, starting with Dijkstra and Lamport.
  - Edsger Dijkstra: 1972 Turing award winner
  - Leslie Lamport: 2013 Turing award winner

So, distributed computing systems are studied since 1967, starting with Dijkstra and Lamport. Dijkstra in 1972 got Turing award for the works on the distributed algorithms and distributed systems. Leslie Lamport very recently has got the Turing award for his work on basically the distributed algorithms and systems.

(Refer Slide Time: 17:13)



 **LESLIE LAMPORT**

He devised important **algorithms** and developed formal modeling and verification protocols that improve the quality of real distributed systems.

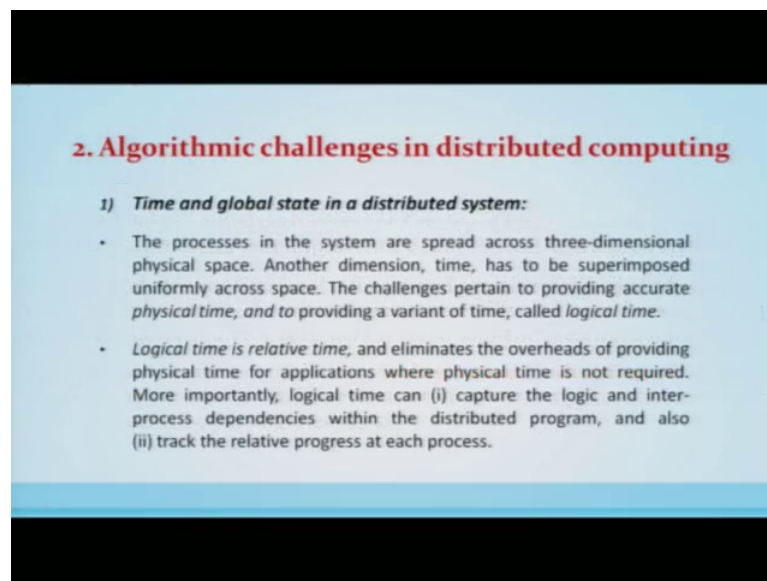
Fundamental contributions to the theory and practice of distributed systems, notably the invention of concepts such as causality and logical clocks, safety and liveness, replicated state machines, and sequential consistency.

- Lamport was the winner of the 2013 **Turing Award** for **distributed computing** systems, in which several autonomous computers communicate with each other by passing messages.

Special mention to the Leslie Lamport, because most of the works whatever he has done we are going to cover up as far as distributed systems fundamentals are concerned.

So, Leslie Lamport devised important algorithms develop formal model verification protocols to improve the quality of real distributed systems. Fundamental contribution to the theory and practice, notably the inventions of the concepts such as causality look logical clock, safety and liveness, replicated state machines, sequential consistency or some of them. So, Lamport was the winner of 2013 that is Turing award for distributed computing.

(Refer Slide Time: 17:56)



**2. Algorithmic challenges in distributed computing**

**1) Time and global state in a distributed system:**

- The processes in the system are spread across three-dimensional physical space. Another dimension, time, has to be superimposed uniformly across space. The challenges pertain to providing accurate *physical time*, and to providing a variant of time, called *logical time*.
- *Logical time is relative time*, and eliminates the overheads of providing physical time for applications where physical time is not required. More importantly, logical time can (i) capture the logic and inter-process dependencies within the distributed program, and also (ii) track the relative progress at each process.

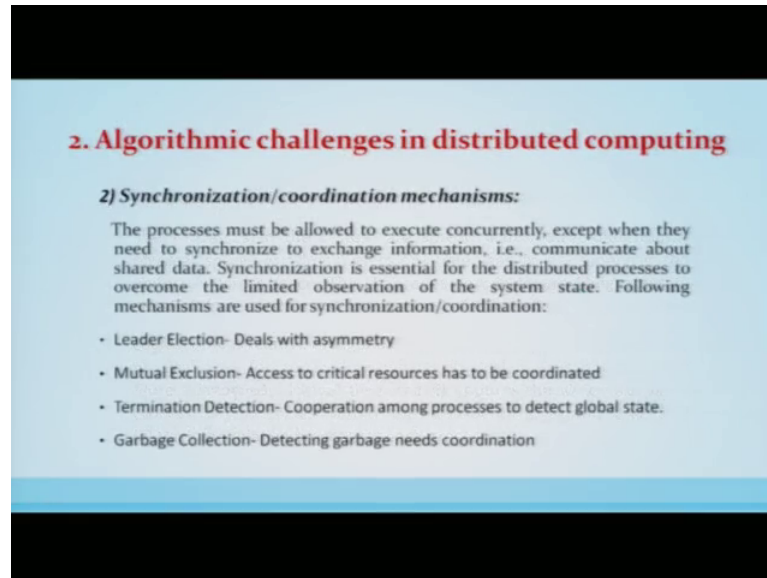
Now, algorithmic challenges in the distributed system, we are going to touch upon that is previously we have seen the design challenges from system perspective. Now we have to see the algorithmic challenges in the developing the distributed system designing the distributed systems. So, time and globally state in a distributed systems. So, first of all this is the important challenge let us see what this is the processes in the system are spread across three-dimensional physical space. Another dimension, is the time, has to be superimposed uniformly across a space.

The challenges pertain to providing accurate physical time, because there is no common clock and to provide a variant of a time, that is called a logical time. So, logical time is the relative time and eliminates the overhead of providing the physical time for the different applications. And basically the logical time basically can capture the logic and the inter-process dependencies within the distributed program, and also track the relative progress at each process.



So, instead of physical having a common physical clock are basically implementation of a common physical clock here we are going to see the how the logical clock and solve without having the physical clock these particular problems.

(Refer Slide Time: 19:19)



**2. Algorithmic challenges in distributed computing**

**2) Synchronization/coordination mechanisms:**

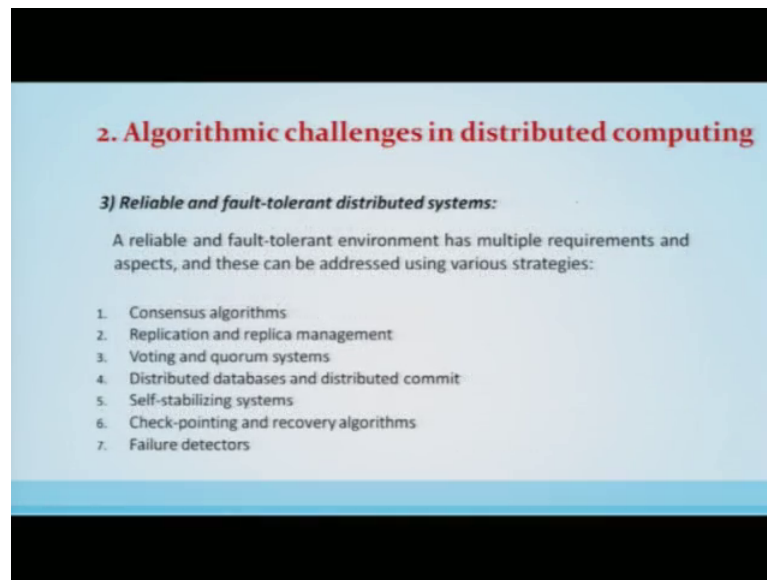
The processes must be allowed to execute concurrently, except when they need to synchronize to exchange information, i.e., communicate about shared data. Synchronization is essential for the distributed processes to overcome the limited observation of the system state. Following mechanisms are used for synchronization/coordination:

- Leader Election- Deals with asymmetry
- Mutual Exclusion- Access to critical resources has to be coordinated
- Termination Detection- Cooperation among processes to detect global state.
- Garbage Collection- Detecting garbage needs coordination

The other problem other algorithmic challenge is the synchronization coordination mechanisms. So, the processes must be allowed to execute concurrently, except when they need to synchronize to exchange the information, that is, communicate about the shared data; so synchronization essential for the distributed processes to overcome the limited observation of the system state. The following mechanisms are used for the synchronization and the coordination. First of all leader election: deals with the asymmetry of a process. And then mutual exclusion: access to the critical resources has to be coordinated through that is done through mutual exclusion.

Then termination detection, that is, cooperation among the processes they will basically able to detect the required state the required global state that is called termination state and that is called termination detection in a distributed system. The next important; that means, thing is called garbage collection detecting the garbage requires another the coordination. So, these are basically the synchronization and coordination mechanisms which will basically be the used up in designing the distributed applications.

(Refer Slide Time: 20:46)



Another thing is another important notion is the reliable and the fault-tolerant distributed system. So, reliable and fault-tolerant environment has multiple requirement aspect, and these can be address by the various strategies which we are going to cover up in this part of the course.

The first one is called consensus algorithm, second is the replication and the replica management, voting and quorum systems, distributed databases and distributed commit, self-stabilization system and check-pointing and recovery algorithm, and failure detectors. So, these together; these strategies will be able to provide the fault-tolerance, that is, if the component like nodes, links are failing yet how the distributed application can basically work on without any disruptions and that is the main requirement of the distributed motivation of a distributed system to have a reliable in a distributed system.

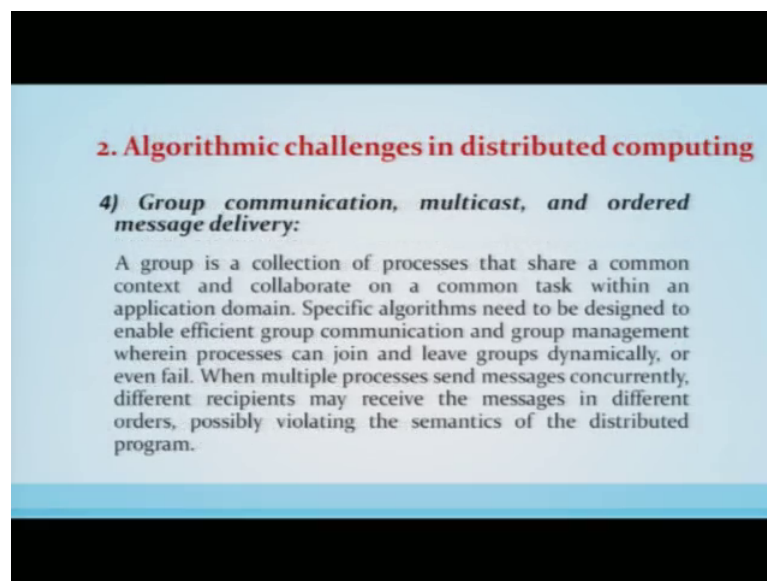
So, basically if the processors or the algorithms which are required are consensus algorithm because in the failures. So, the remaining non-faulty processes they have to basically come up with their consensus on the values and the applications are continuing to run. Another thing is called replica and replica management, because they replicas of that data is available, then the application can run without the problem of failures.

Then voting and a quorum is also a important criteria for example, when the; when some of the systems are when some of the important system is failed, then basically the the among the remaining they have to evolve through the voting and quorum mechanism to

basically run those applications and distributed databases and distributed commit is also basically one of the important applications where they have to decide among discussion with each other they have among the synchronization to see whether the commit which is taking place has to be done or has to be aborted by taking the decisions.

Then, self-stabilization system means if the components are filling then how the system evolves and how much time it takes to stabilize itself. So, that is self-stabilization systems that we are going to also cover up briefly in this part of the course. Check-pointing and recovery system are very important as far as the fault-tolerant is concerned fault-tolerant means if there is a failure how basically the operations which are done has to be basically with the minimal loss has to basically resume their operations. So, check-pointing and rollback recovery algorithms we are going to cover up in this part of the course. Then failure detectors are also very important if how do detect that that the nodes or the links are basically not working or a or a field.

(Refer Slide Time: 23:48)



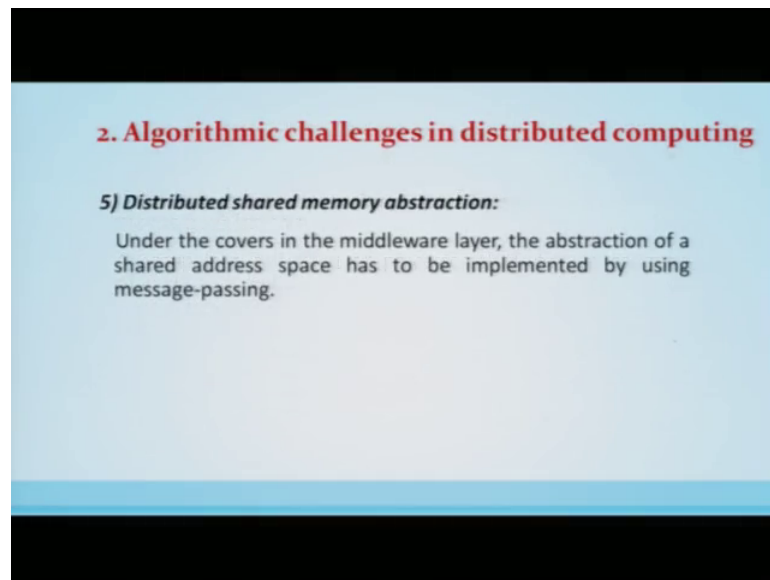
**2. Algorithmic challenges in distributed computing**

**4) Group communication, multicast, and ordered message delivery:**

A group is a collection of processes that share a common context and collaborate on a common task within an application domain. Specific algorithms need to be designed to enable efficient group communication and group management wherein processes can join and leave groups dynamically, or even fail. When multiple processes send messages concurrently, different recipients may receive the messages in different orders, possibly violating the semantics of the distributed program.

Another important algorithmic challenge is basically forming the group communication, multicast, and ordered message delivery. So, there are some applications where the group communication is required so, basically this paradigm is also useful for them to develop the application.

(Refer Slide Time: 24:03)



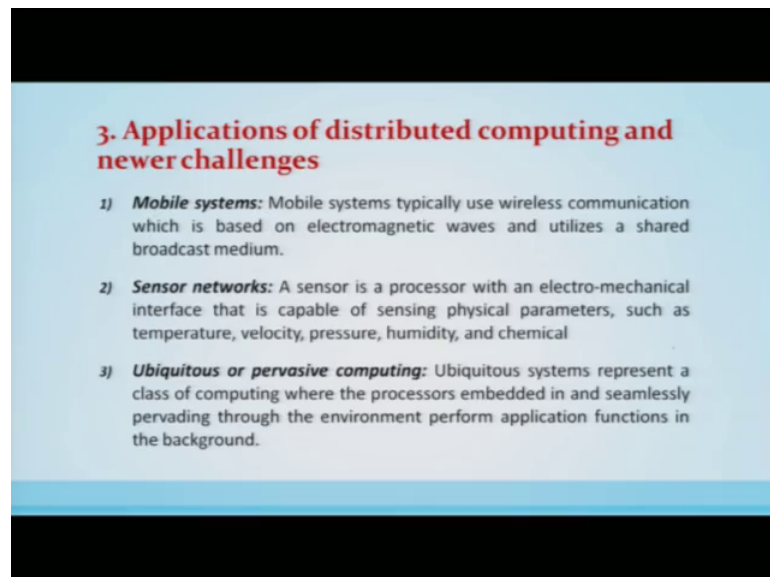
**2. Algorithmic challenges in distributed computing**

**5) Distributed shared memory abstraction:**

Under the covers in the middleware layer, the abstraction of a shared address space has to be implemented by using message-passing.

Distributed shared memory abstraction; now the middleware is the software is the distributed system software will basically use this particular this one abstraction called distributed shared memory. Although, it is not having a common memory, but distributed shared memory is realizable using the message passing systems that we are going to see.

(Refer Slide Time: 24:26)



**3. Applications of distributed computing and newer challenges**

- 1) Mobile systems:** Mobile systems typically use wireless communication which is based on electromagnetic waves and utilizes a shared broadcast medium.
- 2) Sensor networks:** A sensor is a processor with an electro-mechanical interface that is capable of sensing physical parameters, such as temperature, velocity, pressure, humidity, and chemical
- 3) Ubiquitous or pervasive computing:** Ubiquitous systems represent a class of computing where the processors embedded in and seamlessly pervading through the environment perform application functions in the background.

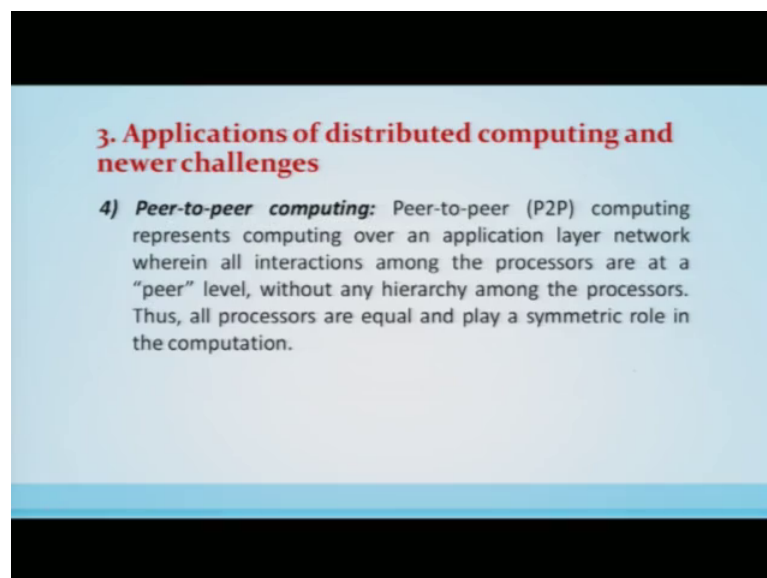
Now, the applications of a distributed computing systems and basically the newer challenges. Mobile systems: Mobile systems typically use the wireless communication which is based on electromagnetic waves and utilizes a shared broadcast medium. So,

mobile system is one of the application of a distributed computing application we are we are different elements are involved and to come up with this particular service and that is called a mobile service and that is an application of a distributed computing. Sensor network is another application.

Sensor is a processor with equip with an electro-mechanical interface and that is capable of sensing physical parameters, such as temperature, velocity, pressure, humidity, and chemical. So, this particular kind of nodes called a sensor node, if basically deployed to basically monitor the cyber physical will make a cyber physical system to monitor any physical activity or event for example, to monitor the whether it is having a volcanic eruption or any other situation.

So, it is having a lot of use in cyber physical system and it is a kind of large scale distributed system that we that is basically using the principles of a distributed systems. Another application here is called ubiquitous or a pervasive computing: Ubiquitous systems represent the class of computing where the processors embedded in seamlessly providing through the environment perform the applications of functions in the background. Examples are the smart environment; smart environment or smart building, smart cities all are examples of a ubiquitous and pervasive computing.

(Refer Slide Time: 26:18)

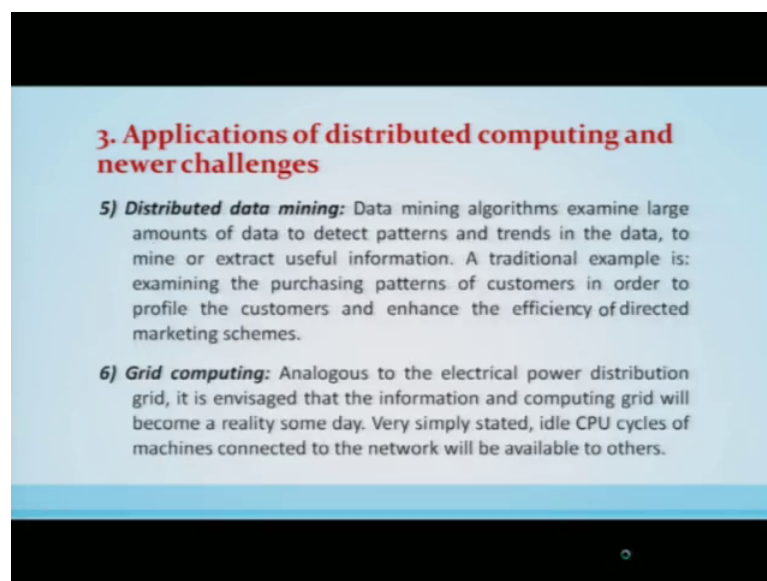


Another application is basically called Peer-to-Peer computing: Peer-to-Peer represents the computing over the application network we are all interactions among the processors

are at the peer levels, without any hierarchy among the processors. Thus, all processors are equal and play a symmetrical role in the computation. So, peer-to-peer network systems are used in providing the resources and services. The peer-to-peer networking is now basically a very challenging as far as developing distributed applications are concerned. Technically also it is quite difficult and, but the simpler model like client server paradigm which is not purely a peer-to-peer it is not purely distributed, this is used by the different applications because industry feels comfortable with client server model.

Peer-to-peer computing models for different applications are evolving a over a period of time the recent application which is added is called bit coin and bit coin is based on peer to peer distributed computing design. We are going to touch upon later on in this part of the course.

(Refer Slide Time: 27:39)

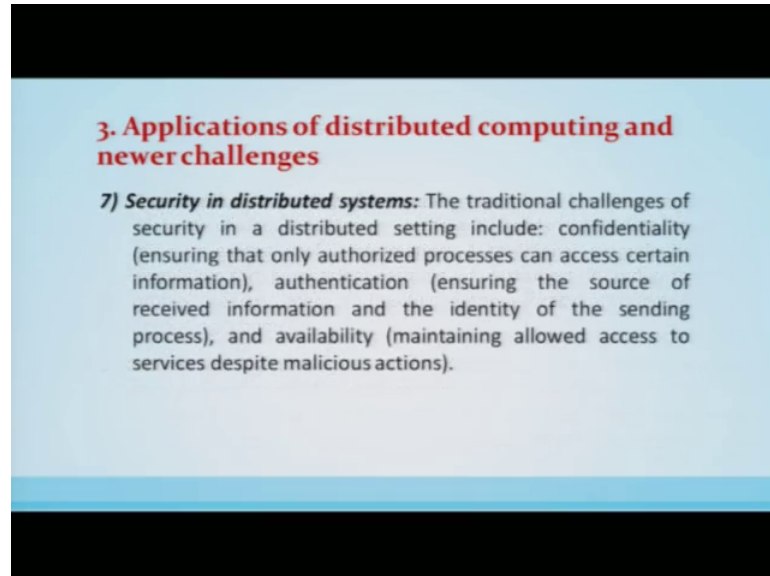


Distributed data mining is another application of a distributed computing. So, distributed data mining algorithms examine large amount of data to detect the patterns and trends in the data, to mine or exact useful information. The traditional example is: examining the purchasing patterns of the customer in order to profile the customers and enhance the efficiency of the directed marketing schemes.

Another application of a distributed computing is fine in the grid computing: Analogous to the electrical power distribution grid, it is envisaged that information and computing

grid will become a reality someday. Very simply stated, idle CPU times of the machines connected to the network will be available to the others.

(Refer Slide Time: 28:33)



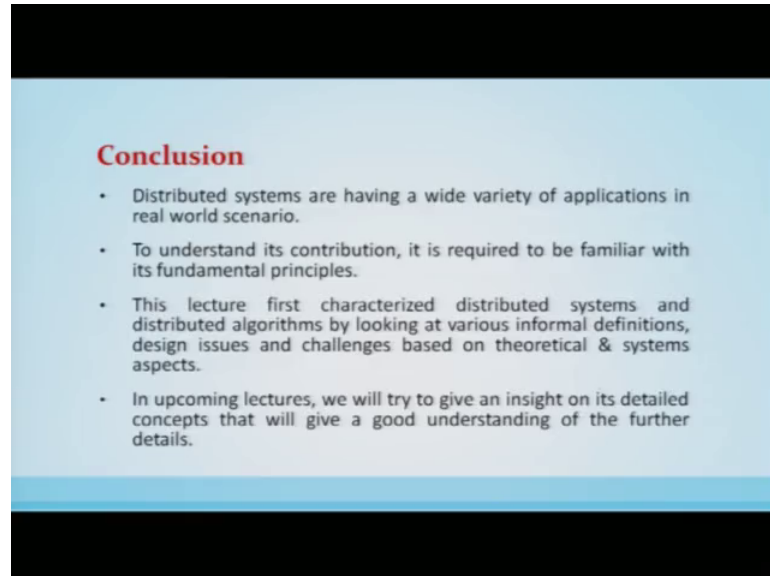
Another application is the security in the distributed. The traditional challenges of the security in a distributed setting include: confidentiality; that means, only authorize person can access, authentication means ensure that the source received the information and the identity of the sending process are basically the genuine and availability is the maintaining allowed access of service despite malicious actions.

So, security in a distributed system is used in the payment systems, the online purchasing and also the recent bit coin, that is, the digital money how the digital money is realized lot of security algorithms are involved in a distributed setting. So, we are going to cover up this aspect also and basically this is important, because in the financial market this is most of this is one of the most important factor in designing such applications in the distributed systems.

So, in the Nutshell, what we have seen here is the distributed systems as having a wide variety of application real world scenarios and some of them we have covered up in this part of the introduction. And this will be the basis and of understanding or pinpointing or underpinning the intricacies the theoretical and intricacies to understand the reasoning how the things are being designed and evolved and also we can verify the it is

correctness, that it is correctly working and also to understand its contribution it is required to be familiar with the fundamental principles.

(Refer Slide Time: 30:17)



So, I told you about this. So, this lecture first categorizes the distributed systems and the distributed algorithms by looking at various informal definitions. The design issues and the challenges based on theoretical and the systems aspects.

In the upcoming lecture, we will try to give an insight on the detailed concept that will give a good understanding of the further details.

Thank you.