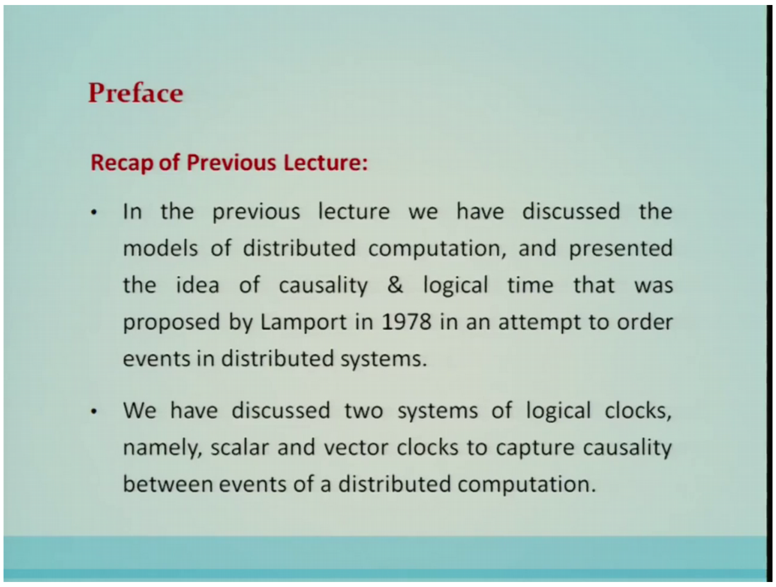
Distributed Systems

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

Lecture - 05
Size of vector clock, Matrix clocks, Virtual time and Physical clock Synchronization

Lecture 5: Size of vector clock, Matrix clock, Virtual time and Physical clock Synchronization. These are the topics of lecture 5. Preface: recap of previous lecture.

(Refer Slide Time: 00:28)



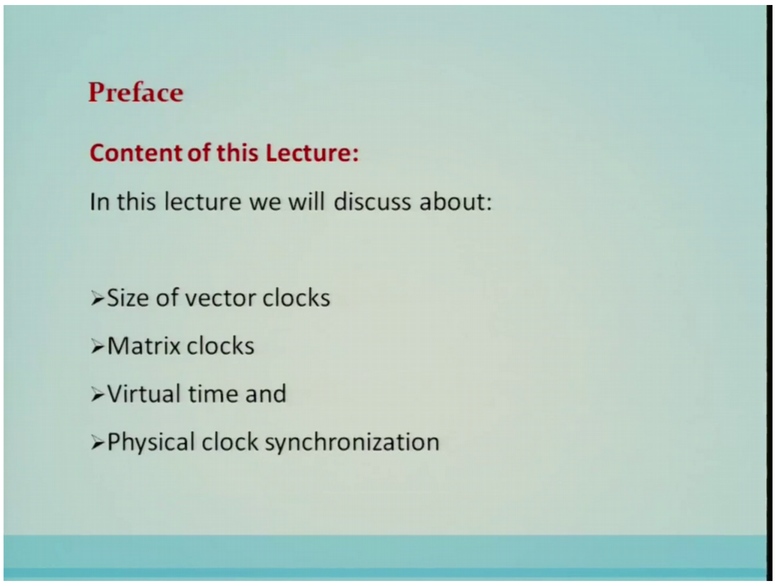
Preface

Recap of Previous Lecture:

- In the previous lecture we have discussed the models of distributed computation, and presented the idea of causality & logical time that was proposed by Lamport in 1978 in an attempt to order events in distributed systems.
- We have discussed two systems of logical clocks, namely, scalar and vector clocks to capture causality between events of a distributed computation.

In previous lecture we have discussed the models of distributed computation and presented the idea of causality and logical time that was proposed by Lamport in 1978, in an attempt to order the events in a distributed system. We have discussed two systems of logical clock namely, scalar and vector clocks to capture causality between events in a distributed system.

(Refer Slide Time: 00:57)



Preface

Content of this Lecture:

In this lecture we will discuss about:

- ➢Size of vector clocks
- ➢Matrix clocks
- ➢Virtual time and
- ➢Physical clock synchronization

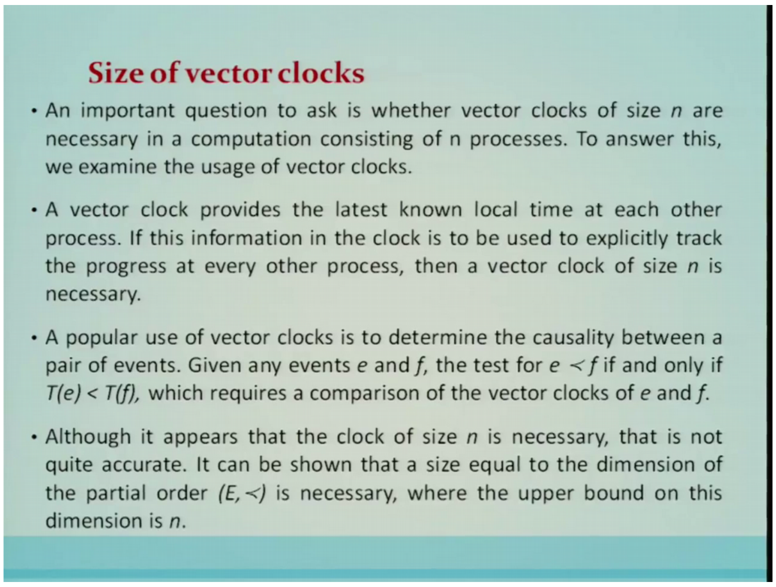
Content of this lecture, in this lecture we will discuss about the size of vector clocks, matrix clock, virtual time and physical clock synchronization.

So, in the last class we have seen the concept of causality which is the fundamental in the design of distributed systems. Usually, causality is tracked using physical time. Now physical time in day to day life we also use it in the form of the loosely synchronized clocks, which we have in the form of wristwatch wall clocks and so on. Our activities are timed using access to these clocks in computer that is in a centralized system there is a single clock and the processes using the issue of the system calls. They can access the computer clock and the events or the processes can get the time out of it, the next process when demands the time it will get the next time and so on.

So, in a centralized system there is no need of clock synchronization, in a distributed system processors have their clocks loosely synchronized all the clocks and as far as synchronization of a clock is concerned becomes a big problems and we are going to see in this particular topic, today all these aspects of causality. How we are going to obtain the causality in a distributed system and physical clocks or a logical clock whichever of them is going to be useful for a design of distributed system.

So, in the last class we have seen the logical clocks in two forms, the first is called scalar clock, the other was the vector clock. Scalar clock provided the clock consistency property and the vector clock provided the strong consistency property. So, the applications which it needs them will be basically going to use them, today we are going to see that the size of the vector clocks; why it is required to be the size of the number of process that is called n. So, that is called size of vector clock, is it required to have an always or we can basically work with less than n still we can have the strong consistency property let us go ahead.

(Refer Slide Time: 03:40)



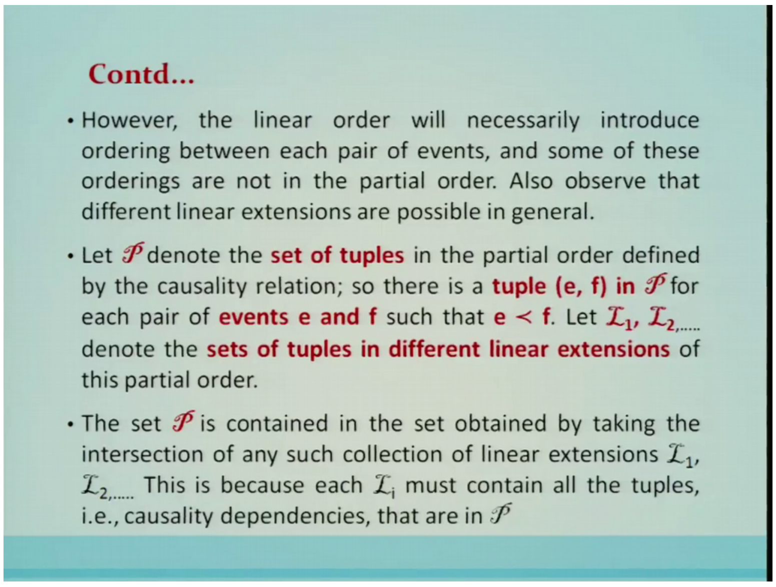
Size of vector clocks

- An important question to ask is whether vector clocks of size *n* are necessary in a computation consisting of n processes. To answer this, we examine the usage of vector clocks.
- A vector clock provides the latest known local time at each other process. If this information in the clock is to be used to explicitly track the progress at every other process, then a vector clock of size *n* is necessary.
- A popular use of vector clocks is to determine the causality between a pair of events. Given any events *e* and *f*, the test for $e \prec f$ if and only if $T(e) < T(f)$, which requires a comparison of the vector clocks of *e* and *f*.
- Although it appears that the clock of size *n* is necessary, that is not quite accurate. It can be shown that a size equal to the dimension of the partial order $(E, \prec)$ is necessary, where the upper bound on this dimension is *n*.

So, size of vector clocks an important question to ask is whether the vector clocks of size n are necessary, in the computation consisting of n processes to answer this question, we have to examine the usage of vector clocks. So, usage of vector clock means the applications where we are going to use them will answer this kind of question. So, the vector clock provides the latest known local time at each other process. So, if this information in the clock is to be used to explicitly track the progress at every other process, then vector clock of size n is necessary.

Now a popular use of vector clock is to determine the causality between the pair of events, given any events e and f the test that e has happened before f if and only if, the timestamp of e is smaller than timestamp of f which requires a comparison of vector clocks of e and f. Although it appears that the size of n is necessary that is not quite accurate, that we are going to see here. It can be shown that the size equal to the dimension of a partial order that is e and that happened before relation is necessary, where the upper bound on this dimension is n not the size of n.

(Refer Slide Time: 05:04)



Contd...

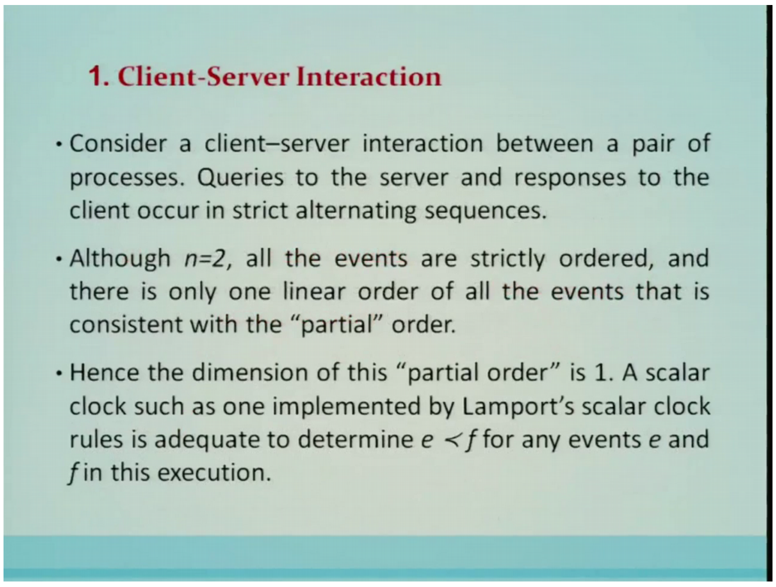
- However, the linear order will necessarily introduce ordering between each pair of events, and some of these orderings are not in the partial order. Also observe that different linear extensions are possible in general.
- Let $\mathscr{P}$ denote the **set of tuples** in the partial order defined by the causality relation; so there is a **tuple (e, f) in** $\mathscr{P}$ for each pair of **events e and f** such that **e** $\prec$ **f**. Let $\mathcal{L}_1$, $\mathcal{L}_2$,..... denote the **sets of tuples in different linear extensions** of this partial order.
- The set $\mathscr{P}$ is contained in the set obtained by taking the intersection of any such collection of linear extensions $\mathcal{L}_1$, $\mathcal{L}_2$,..... This is because each $\mathcal{L}_i$ must contain all the tuples, i.e., causality dependencies, that are in $\mathscr{P}$

So, the definitions in this regard we are going to see to understand this result, on the size of clocks during determining the causality between the pair of events, we firstly to do some definitions linear extension of the partial order, is the linear ordering of e that is consistent with the partial order that, is if two events are ordered in a partial order they are also ordered in the linear order. So, linear extension can be viewed as projecting all the events from different processes on a single time axis, the dimension of a partial order is the minimum number of linear extensions whose intersection gives exactly the partial order.

So, having given this particular definition of a linear order so now we are going to see that linear order, will necessarily introduce the ordering between the pair of events and some of these orderings are not in a partial order, also observe that differently resistances are possible in general and let P denote the set of tuples in a partial order defined by causality relation.

So, there is a tuples e f in P for each pair of events e f. So, that e has happened before f, let linear extensions L 1 L2 and so on; denote the set of tuples in different linear extensions of this partial order. The set capital P is contained in the set obtained by taking the intersections of such collections of linear extensions L1 L 2 and so on. This is because Li contain all the peoples that is the causality dependencies that are in P.

(Refer Slide Time: 06:40)



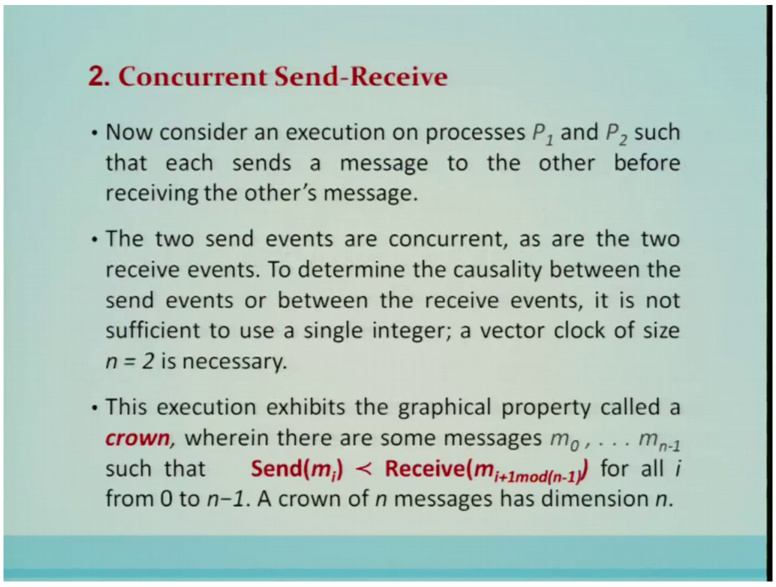
1. Client-Server Interaction

- Consider a client–server interaction between a pair of processes. Queries to the server and responses to the client occur in strict alternating sequences.
- Although $n=2$, all the events are strictly ordered, and there is only one linear order of all the events that is consistent with the "partial" order.
- Hence the dimension of this "partial order" is 1. A scalar clock such as one implemented by Lamport's scalar clock rules is adequate to determine $e \prec f$ for any events e and f in this execution.

Now, let us take one example when there is a client server interaction application, now here the client server interaction happens between a pair of processes the queries to the server and the responses to the client occur in a strict alternating sequences, although n is equal to 2 that is client and server; all the events are is strictly ordered and there is only one linear extension of all events that is consistent with the partial order, hence the dimension of this partial order is 1.

So, the scalar clock such as one implemented by the Lamports scalar, clock rules is id equal to t to determined e has happened before f for any 2 events. So, here only although n is equal 2, but the vector, but only the size of the clock is 1 required to solve this application.

(Refer Slide Time: 07:30)

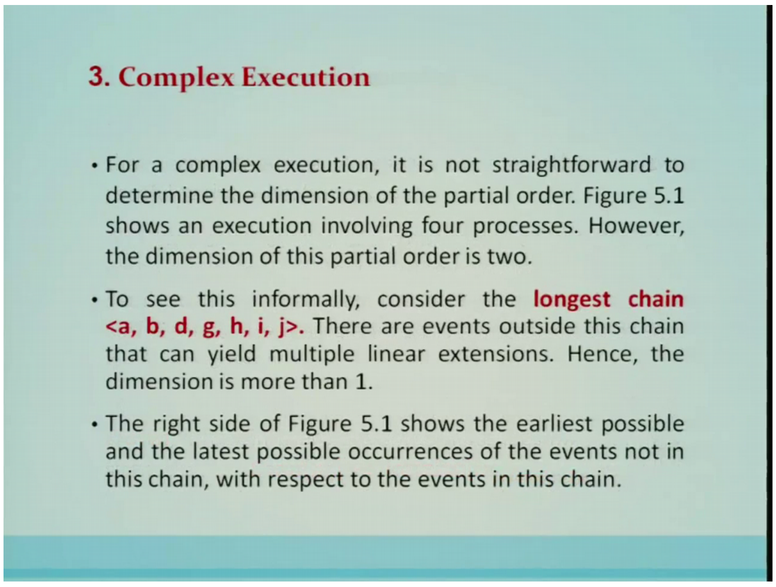


2. Concurrent Send-Receive

- Now consider an execution on processes P_1 and P_2 such that each sends a message to the other before receiving the other's message.
- The two send events are concurrent, as are the two receive events. To determine the causality between the send events or between the receive events, it is not sufficient to use a single integer; a vector clock of size $n = 2$ is necessary.
- This execution exhibits the graphical property called a ***crown***, wherein there are some messages $m_0, \ldots m_{n-1}$ such that **Send(m_i) $\prec$ Receive($m_{i+1 \bmod (n-1)}$)** for all i from 0 to $n-1$. A crown of n messages has dimension n.

Similarly, another application it is called concurrent send and read, send and receive. So, now, consider an execution on processes P 1 and P 2, such that each send a message to the other before receiving the others message. So, the 2 send events are concurrent, as are the 2 receive events. To determine the causality between the send events are between the receive events it is not sufficient to use a single integer a vector clock of size 2 is necessary, this execution exhibits a graphical property called crown. So, a crown of n messages has n dimension.

(Refer Slide Time: 08:15)



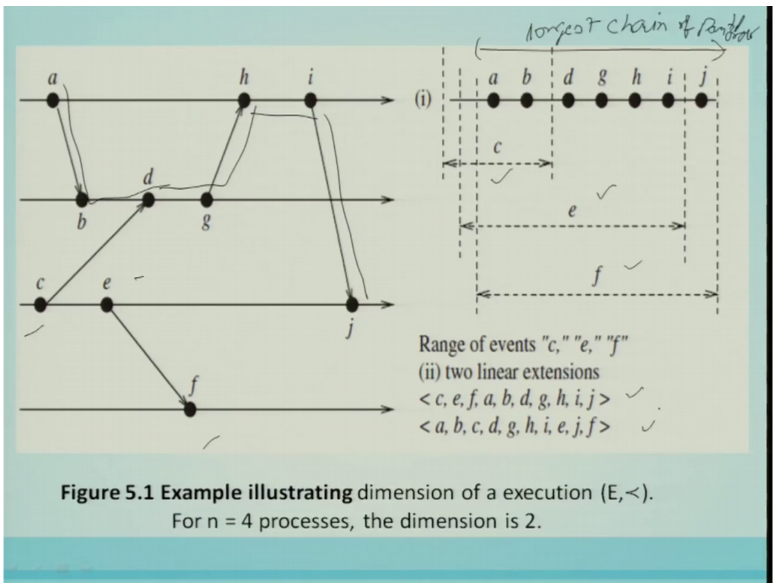
3. Complex Execution

- For a complex execution, it is not straightforward to determine the dimension of the partial order. Figure 5.1 shows an execution involving four processes. However, the dimension of this partial order is two.
- To see this informally, consider the **longest chain <a, b, d, g, h, i, j>**. There are events outside this chain that can yield multiple linear extensions. Hence, the dimension is more than 1.
- The right side of Figure 5.1 shows the earliest possible and the latest possible occurrences of the events not in this chain, with respect to the events in this chain.

So, we are going to see another example of a complex execution. So, in a complex execution to determine the dimension of a partial order is not going to be straight forward. Next figure 5.1 shows an execution of 4 processes; however, the dimension of this partial order is going to be 2. So, that is a vector clock of size 2 is good enough to provide the strong consistency property, among the four process interaction in this particular example.

To see this informally let us consider the longest chain a b d g h i j, there are events outside this chain that can yield multiple linear exchanges and the dimension is more than one, the right side of figure 5.1 we are going to show. Now, that earliest possible and the latest possible occurrences of the events are not in this chain, with respect to the events in this particular chain see this particular example here.

(Refer Slide Time: 09:13)



Range of events "c," "e," "f"

(ii) two linear extensions

< c, e, f, a, b, d, g, h, i, j > ✓

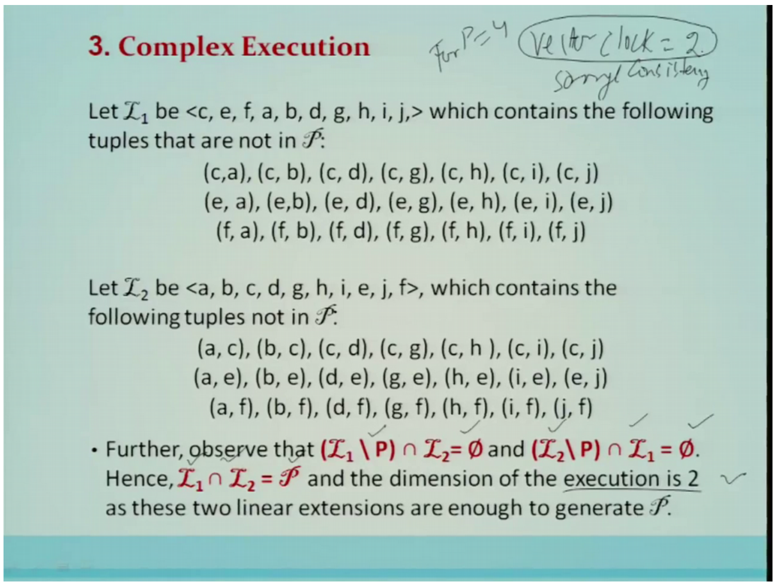
< a, b, c, d, g, h, i, e, j, f > ✓

Figure 5.1 Example illustrating dimension of a execution (E,<).
For n = 4 processes, the dimension is 2.

So, in this example we have seen so the longest linear extension chain that is a b d then g h i and j. So, that is mentioned over here the longest, now out of this particular longest chain. What we can see here these are basically the events which are outside this particular chain that is c e and f. So, c e and f basically outside the chain so, now we are going to form the linear extension. So, 2 linear extensions are possible here in this case so; obviously, the dimension of the partial order is not going to be 1, but more than one.

So, 2 linear extensions if you see is possible with the earliest time and the latest time of c e and f.

(Refer Slide Time: 10:34)



3. Complex Execution

For P=4 $\boxed{\text{Vector Clock} = 2}$ Strong Consistency

Let $\mathcal{L}_1$ be <c, e, f, a, b, d, g, h, i, j,> which contains the following tuples that are not in $\mathcal{P}$:

(c,a), (c, b), (c, d), (c, g), (c, h), (c, i), (c, j)
(e, a), (e,b), (e, d), (e, g), (e, h), (e, i), (e, j)
(f, a), (f, b), (f, d), (f, g), (f, h), (f, i), (f, j)

Let $\mathcal{L}_2$ be <a, b, c, d, g, h, i, e, j, f>, which contains the following tuples not in $\mathcal{P}$:

(a, c), (b, c), (c, d), (c, g), (c, h), (c, i), (c, j)
(a, e), (b, e), (d, e), (g, e), (h, e), (i, e), (e, j)
(a, f), (b, f), (d, f), (g, f), (h, f), (i, f), (j, f)

- Further, observe that $(\mathcal{L}_1 \setminus P) \cap \mathcal{L}_2 = \emptyset$ and $(\mathcal{L}_2 \setminus P) \cap \mathcal{L}_1 = \emptyset$. Hence, $\mathcal{L}_1 \cap \mathcal{L}_2 = \mathcal{P}$ and the dimension of the execution is 2 as these two linear extensions are enough to generate $\mathcal{P}$.

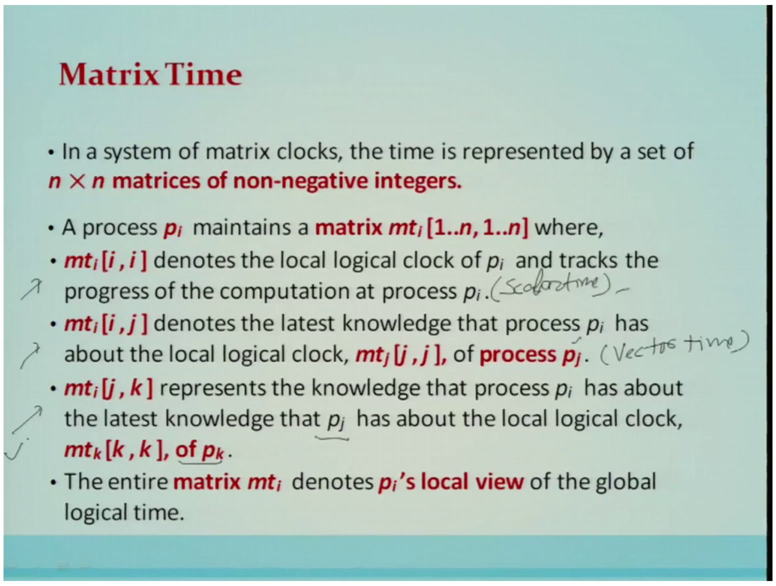
So, there are 2 different linear extensions are possible who are here in this case and these 2 linear extensions if we take intersection, it will give the partial order set that is what is going to be shown here in this particular complete illustrative example.

So, L1 is a linear extension L2 is another linear extension and when L 1 minus partial ordered set P intersection L 2 it becomes empty or a phi, similarly L 2 is a linear extension minus P intersection L 1 is also given giving the empty one. So, hence the intersection of L 1 and L 2 these are the 2 linear extension this will basically generate the complete partial order.

Hence the dimension of this particular execution is 2. So that means, 2 linear extensions are good enough to generate this partial order, hence the vector clock of size 2 is good enough to ensure by strong consistency property and hence basically to solve this application. So, all for the total number of processes is equal to four. So, only the vector clock of size 2 is required here in this particular case.

So, finding out the dimension of the partial order is not going to be easy, computationally is going to be difficult. So, basically posterior legal analysis is required here in this case to identify the size of the vector clock and it can be optimized and different algorithms are going to use this kind of concept in reducing the size of the vector clock.

(Refer Slide Time: 12:15)



Matrix Time

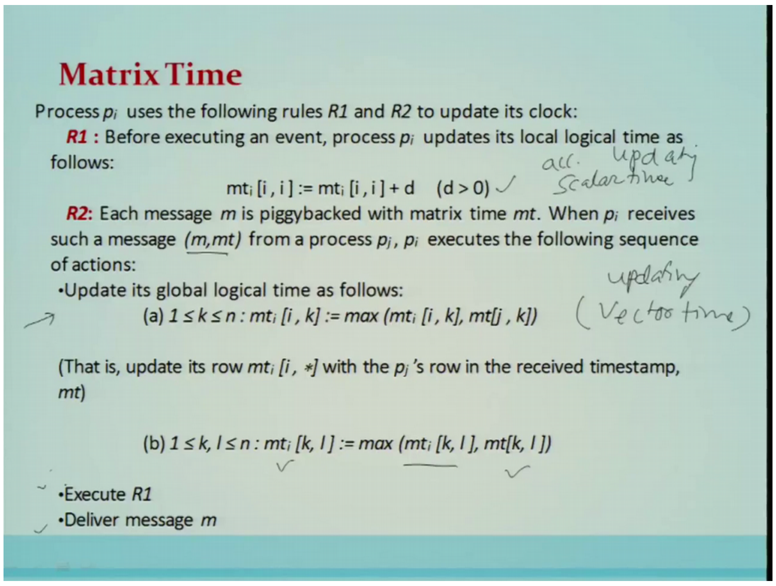
- In a system of matrix clocks, the time is represented by a set of **$n \times n$ matrices of non-negative integers.**
- A process p_i maintains a **matrix $mt_i[1..n, 1..n]$** where,
- $mt_i[i, i]$ denotes the local logical clock of p_i and tracks the progress of the computation at process p_i. (Scalar time)
- $mt_i[i, j]$ denotes the latest knowledge that process p_i has about the local logical clock, $mt_j[j, j]$, of **process p_j**. (Vector time)
- $mt_i[j, k]$ represents the knowledge that process p_i has about the latest knowledge that p_j has about the local logical clock, $mt_k[k, k]$, of p_k.
- The entire **matrix mt_i** denotes **p_i's local view** of the global logical time.

Now, the next clock we are going to see is called the matrix time or a matrix clock. So, in a system of matrix clocks, the time is represented by a set of n by n matrix of non negative integers.

So, let us go and see the details of the matrix time, a process pi maintains a matrix mt i n by n where mt i i comma i denotes the local logical clock of pi and tracks the progress of computation. So, it is going to be working like a scalar time, like we have seen in the scalar clock. The other component that is called mt i i comma j denotes the latest knowledge of process pi, has about local logical clock of process p j. That means, this particular aspect will give you something called vector time, the notion of a vector time is given here in this case.

Now, the third aspect of a matrix time that is mt i j comma k, represents the knowledge that pi has about the latest knowledge that p j process, has about the local logical clock of pk. This global information is stored in the local view of process pi. So, the entire matrix pi denotes the pi local view of the global logical time here in this matrix clock.

(Refer Slide Time: 14:09)



Matrix Time

Process p_i uses the following rules *R1* and *R2* to update its clock:

R1 : Before executing an event, process p_i updates its local logical time as follows:

$$mt_i[i,i] := mt_i[i,i] + d \quad (d > 0)$$

acc. updating Scalar time

R2: Each message *m* is piggybacked with matrix time *mt*. When p_i receives such a message *(m,mt)* from a process p_j, p_i executes the following sequence of actions:

•Update its global logical time as follows:

updating (Vector time)

(a) $1 \le k \le n : mt_i[i,k] := max(mt_i[i,k], mt[j,k])$

(That is, update its row $mt_i[i,*]$ with the p_j's row in the received timestamp, *mt*)

(b) $1 \le k, l \le n : mt_i[k,l] := max(mt_i[k,l], mt[k,l])$

•Execute *R1*

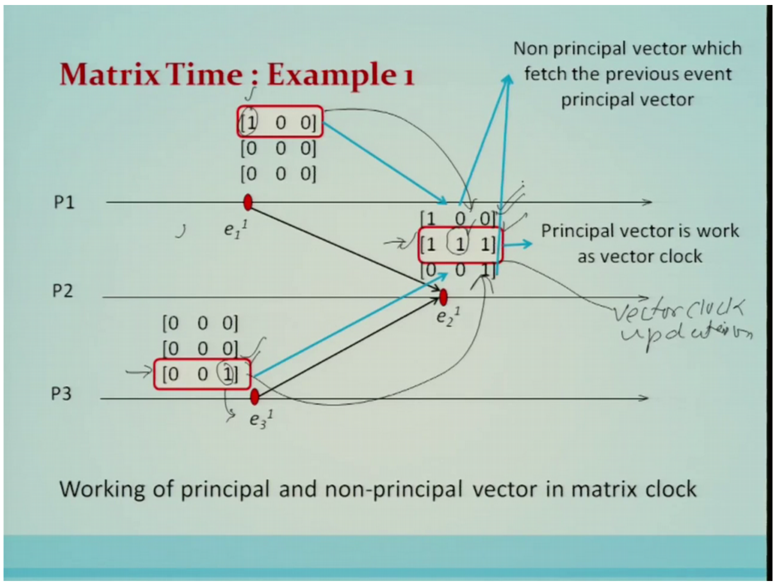
•Deliver message *m*

Now, the 2 rules which matrix clock used to follow to update this clock is R 1 and R 2 we are going to explain it. So, the rule R 1 before executing an event process p i updates its local logical time as follows. So, it will be done according to by scalar clock that we have already seen the rule R 2, each message m is piggybacked with the matrix time mt; when pi receives such a message containing that particular matrix timestamp message from pj, then p i execute the following sequence of actions. So, it will update its global logical time as follows.

So; that means, mt i that is pi process for its i-th, row that is a i-th vector it will update as per the information as per the time, which is as for the matrix which is received by that particular process pj. So, it will be like updating the vector clock, now the remaining portion of the matrix of i will be updated like this.

So, basically for k which is greater than 1; that means, for all other rows and also the complete vector of the basically the rows of other processes are going to be updated as per the information, which is being obtained from the message which has basically sent its the matrix timestamp. After doing this updation according to rule R 2 it will execute R 1 and it will tell you or the message m.

(Refer Slide Time: 16:25)

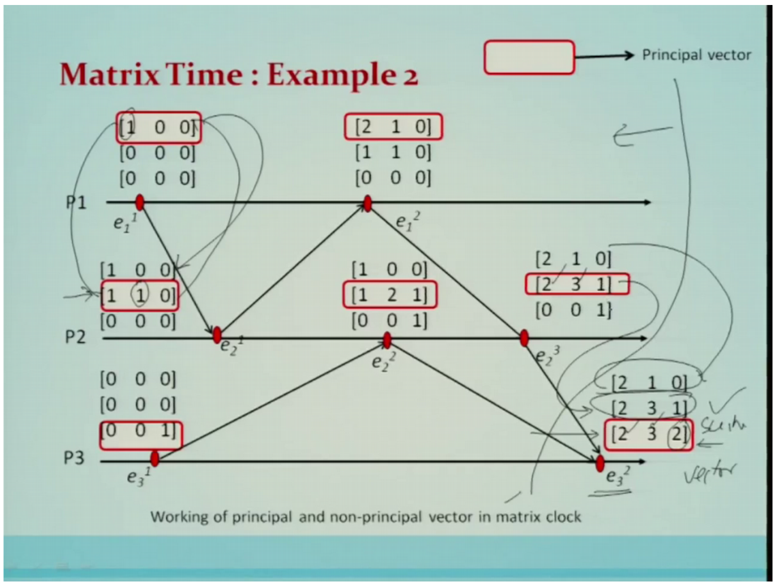


Matrix Time : Example 1
Non principal vector which fetch the previous event principal vector
Principal vector is work as vector clock
vector clock updation
P1
P2
P3
e_1^1
e_2^1
e_3^1
[1 0 0] [0 0 0] [0 0 0]
[1 0 0] [1 1 1] [0 0 1]
[0 0 0] [0 0 0] [0 0 1]
Working of principal and non-principal vector in matrix clock

So, example one you can clearly see that event e 1 is represented here in the matrix and this particular element is representing the local clock or the scalar time of process 1. Now this particular matrix when it sends a message according to the event 1 which is message sent event, similarly event 3 also will basically how this kind of matrix because the third row and the third column will basically indicate the scalar time of process P 3, similarly when they will be received at e 2 of a process P 2. So, now, let us see the details how it will be basically updating the clock.

So, process P 2 first it will basically modify it is a scalar clock. So, that is in the second row and the second column. So, it will be the event number 1, now as far as the time which it has received that will be updated based on the information, based on the local view of the other process for example, process 1 has its clock 1. So, it will be updated over here, similarly process 3 has a clock value 1 so, it will be updated. So, this will be the vector clock kind of updation which we have seen. The remaining part this portion will be copied here and the last portion also will be copied here. So, the P 2 will have the view of the P 3 time, similarly the P 2 will have the view of P 1 time and this particular updation is done according to the matrix clock rule which we have seen.

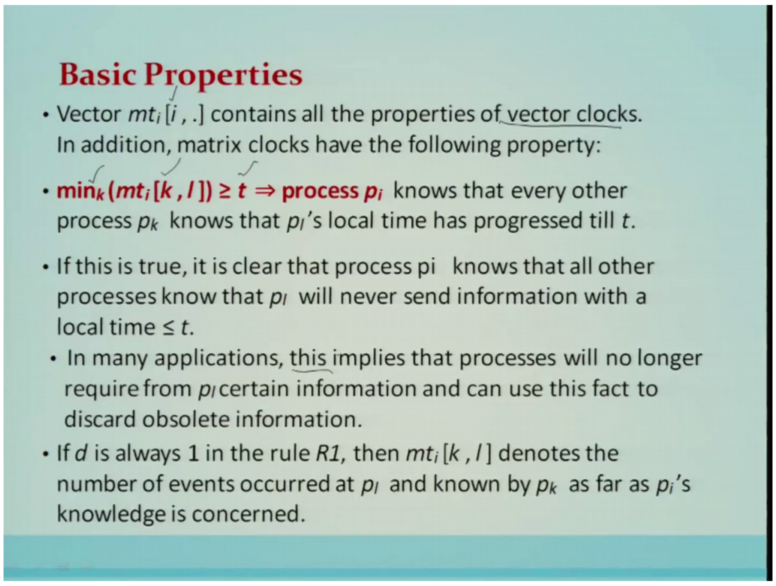
(Refer Slide Time: 18:44)



In another example you can see when this particular message is received to P 2. So, P 2 means its vector at the second row, its particular event that is the scalar event is first updated and then it will update the vector time. So, vector time means the time of one will be updated over here and 0 means it has not seen about any event from P 3 and this will be copied over here as far as the third updation, that is the matrix clock which is the copied the vector of other processes; which P 2 will have the knowledge. Similarly about the other interactions you can see finally, you can see that the event e 3 which has complete view of all the events which has happened. So, let us see its vector.

Now, so e 3 means it is about the process P 3, now its event that is at the second event because e 3 2 if you see it is second event so; that means, it is a scalar time is 2 and its vector will be 2 comma 3, that means it has seen. So, far the message which is coming from 2, so 2 mean it is having 3. So, it will be updated over here similarly it is having the vector 2 so it will be updated. So, the vector clock and scalar clock both are updated, now as far as the vectors of other processes are concerned. So, it will be copied similarly this also will be copied over here. So, this is the matrix time and we have seen the working of a matrix clock, what are the basic properties of a matrix clock that we are going to see now.

(Refer Slide Time: 20:54)



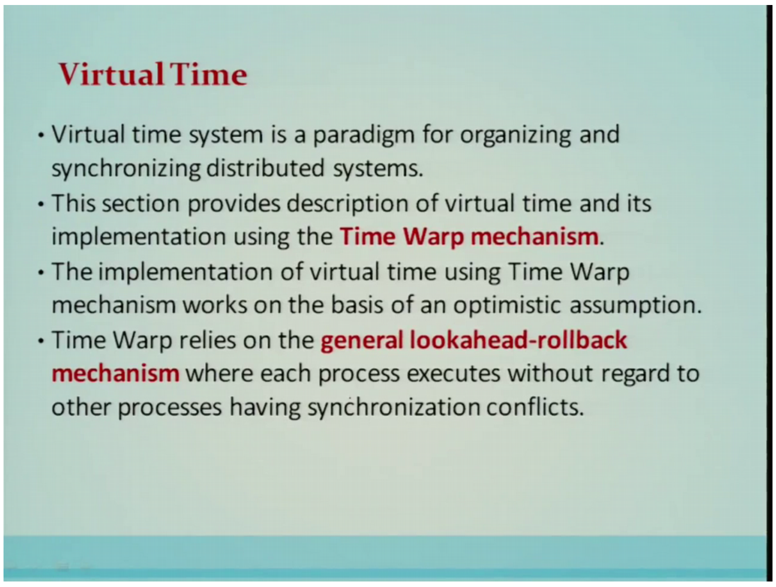
Basic Properties

- Vector $mt_i[i, .]$ contains all the properties of vector clocks. In addition, matrix clocks have the following property:
- **$\min_k(mt_i[k, l]) \geq t \Rightarrow$ process p_i** knows that every other process p_k knows that p_l's local time has progressed till t.
- If this is true, it is clear that process pi knows that all other processes know that p_l will never send information with a local time $\leq t$.
  - In many applications, this implies that processes will no longer require from p_l certain information and can use this fact to discard obsolete information.
- If d is always 1 in the rule $R1$, then $mt_i[k, l]$ denotes the number of events occurred at p_l and known by p_k as far as p_i's knowledge is concerned.

So, the vector mt i that is the i-th row that is nothing but the complete vector, it will have the all the properties of a vector clock, in addition the matrix clock have the following properties; that means, all the processes with which is having the minimum value of k, which has seen which is basically having the information at the p i-th

So, minimum value is basically t so that means, process PI knows that every other process that is process pk knows about P is local time has progressed until t, have having this information there are many applications which will require this information. Which says that the other processes will no longer require the information from P i, which is basically before the time t. Hence, they can discard this absolute information in this particular manner.

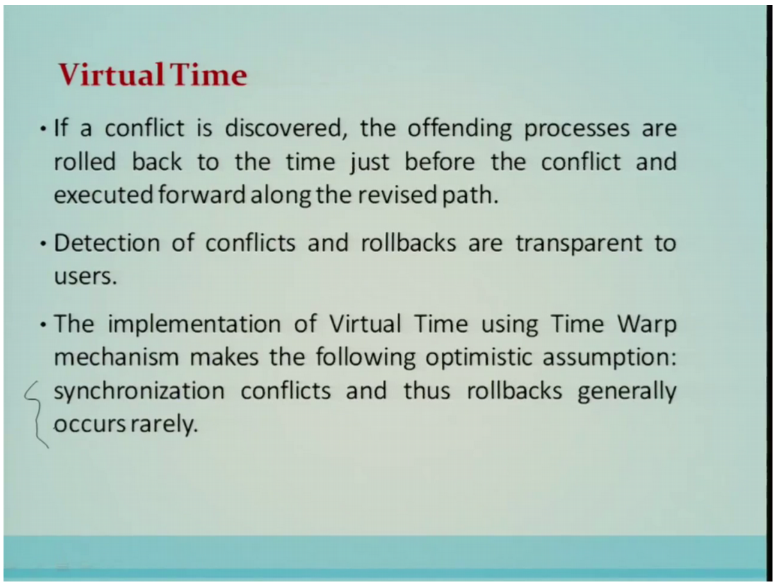
(Refer Slide Time: 22:07)


Virtual Time

- Virtual time system is a paradigm for organizing and synchronizing distributed systems.
- This section provides description of virtual time and its implementation using the **Time Warp mechanism**.
- The implementation of virtual time using Time Warp mechanism works on the basis of an optimistic assumption.
- Time Warp relies on the **general lookahead-rollback mechanism** where each process executes without regard to other processes having synchronization conflicts.

Now, the next time system we are going to consider is called virtual time, now virtual time system is a paradigm for organizing and synchronizing distributed system. So, this particular section we are describing the virtual time and its implementation using time wrap mechanism. The implementation of a virtual time using time wrap mechanism works on the basis of optimistic assumption, what is that optimistic assumption time wrap relies on a general look ahead rollback mechanism, where each process executes without regard to the other processes having synchronization conflict.

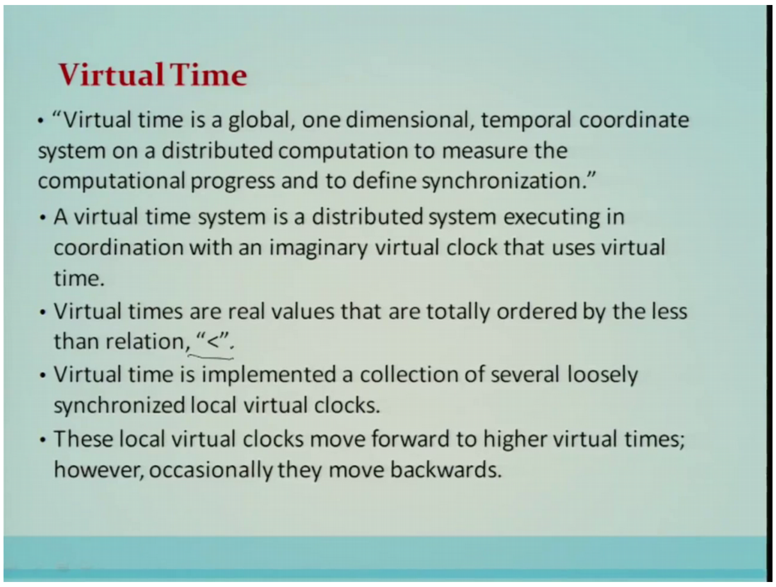
(Refer Slide Time: 22:47)


Virtual Time

- If a conflict is discovered, the offending processes are rolled back to the time just before the conflict and executed forward along the revised path.
- Detection of conflicts and rollbacks are transparent to users.
- The implementation of Virtual Time using Time Warp mechanism makes the following optimistic assumption: synchronization conflicts and thus rollbacks generally occurs rarely.

If a conflict is discovered in the opportunistic in the optimistic assumption, the offending process are rolled back to the time just before the conflict and executed forward along the revised path. Detection of conflicts and roll back are transparent to the user, the implementation of virtual time using time wrap mechanism makes the following optimistic assumption that, synchronization conflict and roll backs generally are occurred really. So, it is not a frequent operation for the rollback or a conflict.

(Refer Slide Time: 23:28)



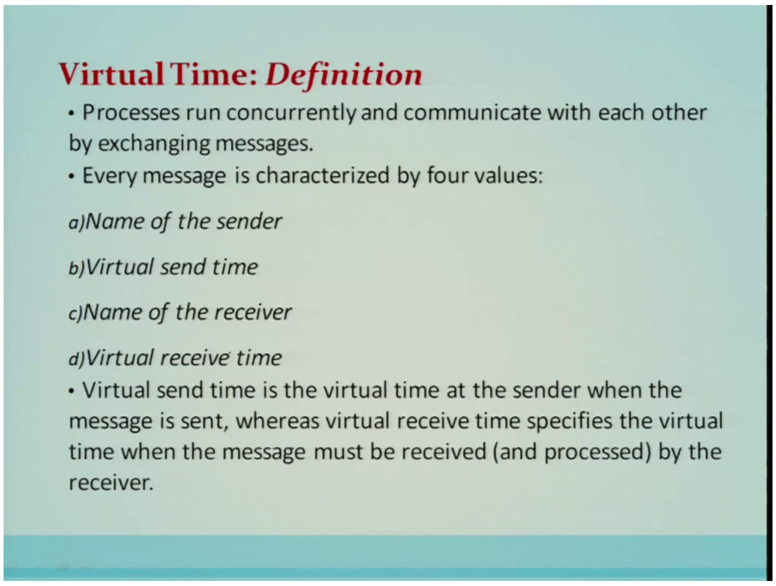
Virtual Time

- "Virtual time is a global, one dimensional, temporal coordinate system on a distributed computation to measure the computational progress and to define synchronization."
- A virtual time system is a distributed system executing in coordination with an imaginary virtual clock that uses virtual time.
- Virtual times are real values that are totally ordered by the less than relation, "<".
- Virtual time is implemented a collection of several loosely synchronized local virtual clocks.
- These local virtual clocks move forward to higher virtual times; however, occasionally they move backwards.

So, virtual time is a global one dimensional temporal coordinates on a distributed computation to measure the computational progress and to define the synchronization. A virtual time system is a distributed system executing the coordination with imaginary virtual clock that uses virtual time. So, virtual times are real values that are totally ordered by the operation by the relation called less than relation.

So, virtual time is implemented a collection of several loosely synchronized local virtual clocks, these local virtual clocks move forward to a higher virtual times. However, occasionally they may move backwards whenever there are conflicts.

(Refer Slide Time: 24:11)



Virtual Time: *Definition*

- Processes run concurrently and communicate with each other by exchanging messages.
- Every message is characterized by four values:

a)Name of the sender

b)Virtual send time

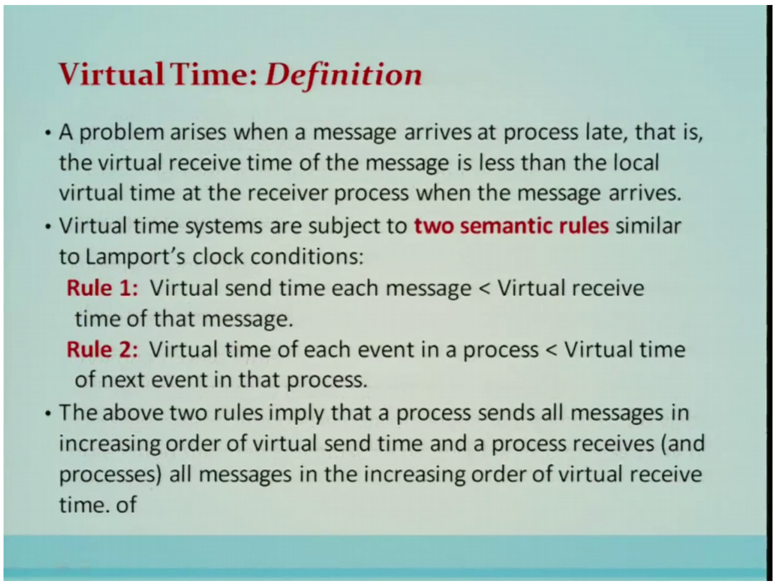
c)Name of the receiver

d)Virtual receive time

- Virtual send time is the virtual time at the sender when the message is sent, whereas virtual receive time specifies the virtual time when the message must be received (and processed) by the receiver.

So, virtual time processes can concurrently communicate with each other by exchanging message. So, every message is characterized by 4 values in virtual time, name the sender is there virtual send time is there, name of receiver and virtual receive time is there. Now virtual send time is the virtual time of the sender when the message is sent, whereas virtual receive time specifies the virtual time when the message must be received by the receiver.

(Refer Slide Time: 24:41)



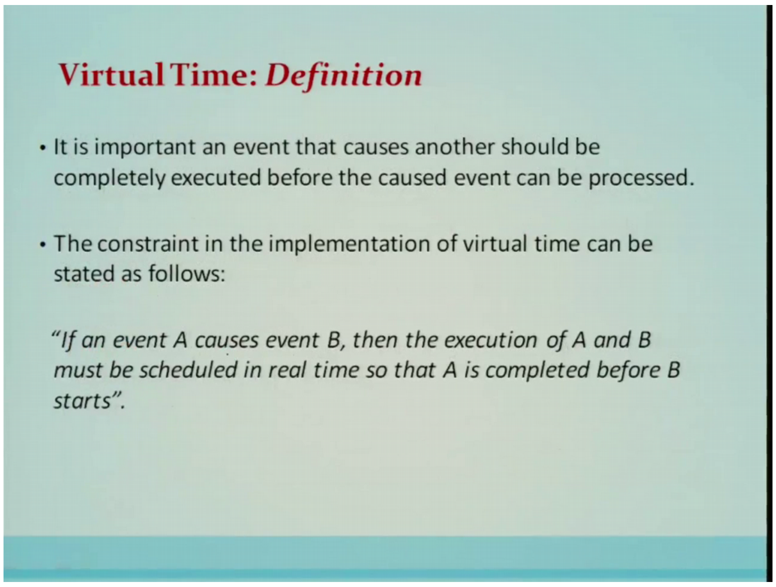
Virtual Time: *Definition*

- A problem arises when a message arrives at process late, that is, the virtual receive time of the message is less than the local virtual time at the receiver process when the message arrives.
- Virtual time systems are subject to **two semantic rules** similar to Lamport's clock conditions:
  Rule 1: Virtual send time each message < Virtual receive time of that message.
  Rule 2: Virtual time of each event in a process < Virtual time of next event in that process.
- The above two rules imply that a process sends all messages in increasing order of virtual send time and a process receives (and processes) all messages in the increasing order of virtual receive time. of

So, the problem arises when the message arrives at a process late, that is the virtual receive time of the message is less than the local virtual time at the receiver process when the message arrives. So, this is going to be handled here in virtual time. So, what your time systems are subject to 2 semantic rules, similar to the Lamports clock conditions rule 1 says that virtual send time each message has will be should be less than the virtual receive time of that particular message, otherwise there will be a problem and problem will be resolved using the rule (Refer Time: 25:19).

Rule 2 says that: virtual time of each event in the in the process, is less than the virtual time of the next event in that particular process. This is going to be solved using the global clock system. So, the above 2 rules imply that the process sends all the message in the increasing order of what virtual send time and the process receives all the message in increasing order of the virtual receive time.

(Refer Slide Time: 25:45)



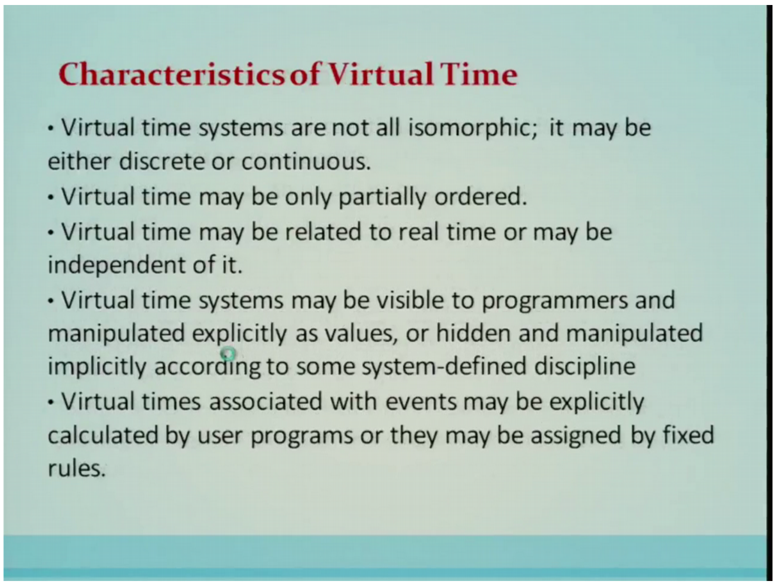
Virtual Time: *Definition*

- It is important an event that causes another should be completely executed before the caused event can be processed.
- The constraint in the implementation of virtual time can be stated as follows:

  "If an event A causes event B, then the execution of A and B must be scheduled in real time so that A is completed before B starts".

Now, it is important and event that causes another should be completely executed before the cost event can be processed, this is already indicated include 1 and rule 2.

(Refer Slide Time: 25:59)

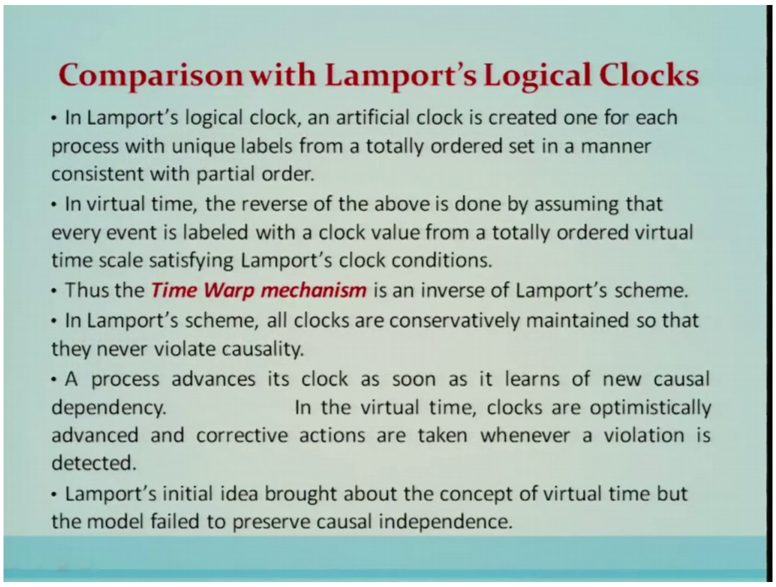


Characteristics of Virtual Time

- Virtual time systems are not all isomorphic; it may be either discrete or continuous.
- Virtual time may be only partially ordered.
- Virtual time may be related to real time or may be independent of it.
- Virtual time systems may be visible to programmers and manipulated explicitly as values, or hidden and manipulated implicitly according to some system-defined discipline
- Virtual times associated with events may be explicitly calculated by user programs or they may be assigned by fixed rules.

So, we are now going to see the characteristics of a virtual time, virtual time systems are not all isomorphic; it may be either discrete or continuous virtual time may be only partially ordered. Virtual time may be related to the real time or may be independent of it, virtual time system may be visible to the programmers and manipulated explicitly as the values of hidden and manipulated explicitly according to some system defined discipline, virtual times associated with the events may be explicitly calculated by the user program or may be assigned by fixed rules.

(Refer Slide Time: 26:35)



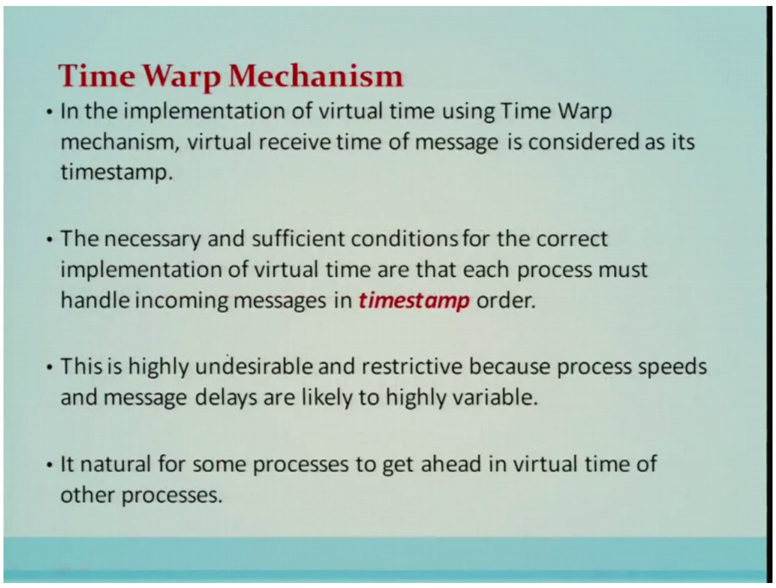
Comparison with Lamport's Logical Clocks

- In Lamport's logical clock, an artificial clock is created one for each process with unique labels from a totally ordered set in a manner consistent with partial order.
- In virtual time, the reverse of the above is done by assuming that every event is labeled with a clock value from a totally ordered virtual time scale satisfying Lamport's clock conditions.
- Thus the ***Time Warp mechanism*** is an inverse of Lamport's scheme.
- In Lamport's scheme, all clocks are conservatively maintained so that they never violate causality.
- A process advances its clock as soon as it learns of new causal dependency. In the virtual time, clocks are optimistically advanced and corrective actions are taken whenever a violation is detected.
- Lamport's initial idea brought about the concept of virtual time but the model failed to preserve causal independence.

So, now we are going to see the comparison with Lamports logical clock, how it is how the virtual time is basically the different. So, in the Lamports logical clock and artificial clock is created one for each process with the unique levels; from totally ordered set in a manner consistent with a partial order that we have seen. In virtual time the reverse of the above is done by assuming that, every event is labeled with a clock value from a totally ordered virtual time scale satisfying the Lamports condition.

Thus, the time warp mechanism is an inverse of Lamports scheme; in Lamports scheme all the clocks are conservatively maintained. So, that they never violate causality the process advances its clock as soon as it learns a new causal dependency in the virtual time, clocks are optimistically advanced and corrective actions are taken whenever violations are detected.

(Refer Slide Time: 27:29)

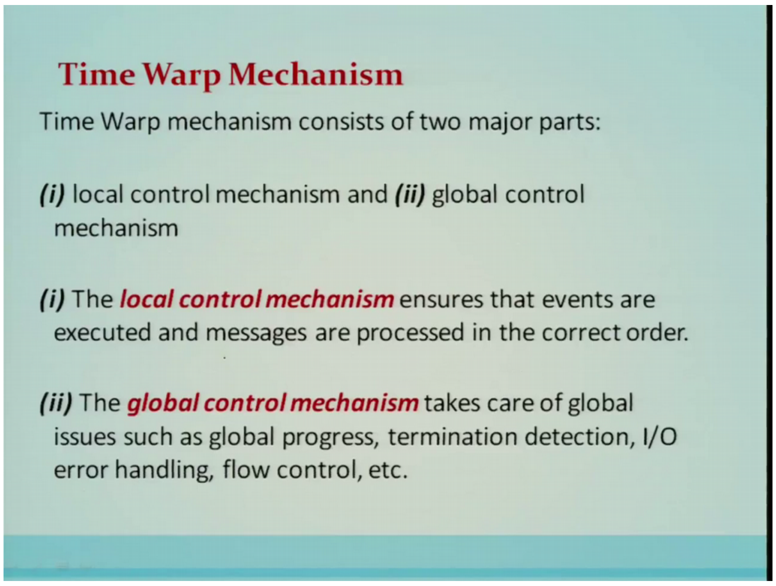


Time Warp Mechanism

- In the implementation of virtual time using Time Warp mechanism, virtual receive time of message is considered as its timestamp.
- The necessary and sufficient conditions for the correct implementation of virtual time are that each process must handle incoming messages in ***timestamp*** order.
- This is highly undesirable and restrictive because process speeds and message delays are likely to highly variable.
- It natural for some processes to get ahead in virtual time of other processes.

So time warp mechanism, so in the implementation of virtual time using time warp mechanism, virtual receive time of a message is considered as a timestamp the necessary and sufficient condition for collect implementation of virtual time are that each process must handle incoming message in the time stamp order.

(Refer Slide Time: 27:53)



Time Warp Mechanism

Time Warp mechanism consists of two major parts:

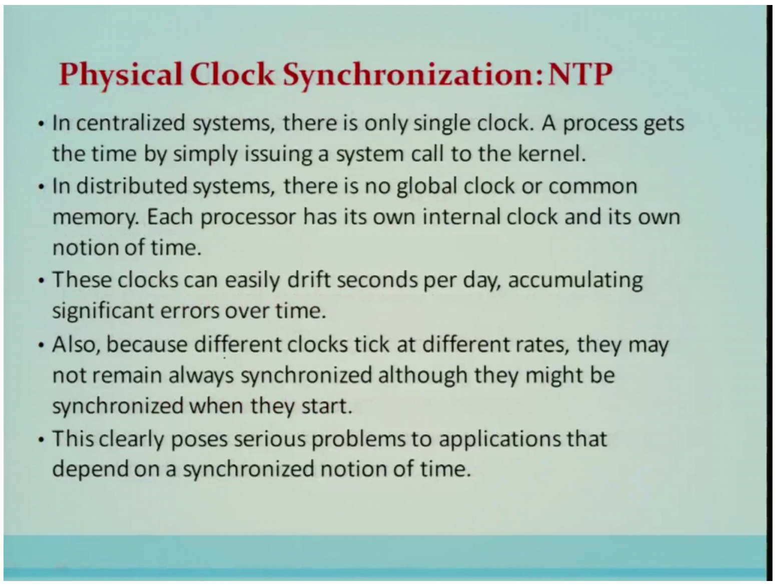
(i) local control mechanism and ***(ii)*** global control mechanism

(i) The ***local control mechanism*** ensures that events are executed and messages are processed in the correct order.

(ii) The ***global control mechanism*** takes care of global issues such as global progress, termination detection, I/O error handling, flow control, etc.

Now, that time warp mechanism consists of 2 major parts, the first is called local control mechanism, the second part is called global control mechanism. Local control mechanism ensures that the events are executed and the messages are processed in the correct order, the global control mechanism takes care of global issues such as global progress, termination, detection I O handling flow control etc; that we have seen in rule 1 and rule 2 and they are implemented in a time warp mechanism.

(Refer Slide Time: 28:27)



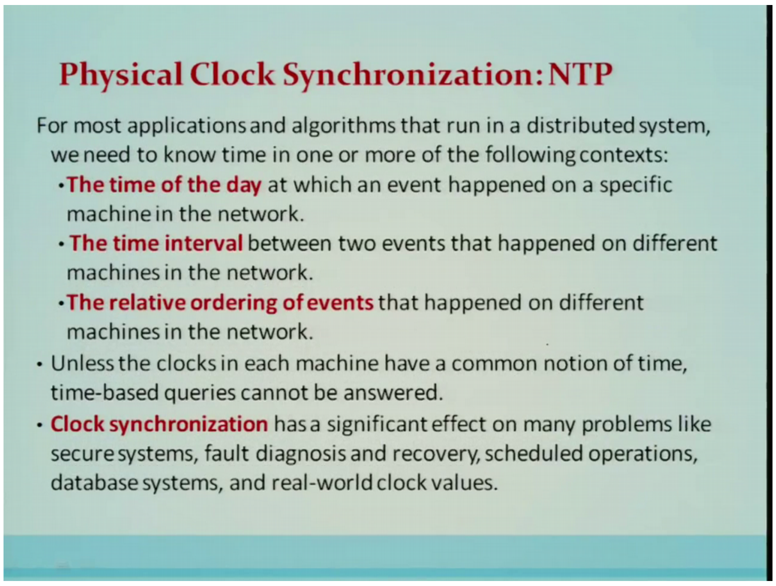
Physical Clock Synchronization: NTP

- In centralized systems, there is only single clock. A process gets the time by simply issuing a system call to the kernel.
- In distributed systems, there is no global clock or common memory. Each processor has its own internal clock and its own notion of time.
- These clocks can easily drift seconds per day, accumulating significant errors over time.
- Also, because different clocks tick at different rates, they may not remain always synchronized although they might be synchronized when they start.
- This clearly poses serious problems to applications that depend on a synchronized notion of time.

Now, the next part of discussion we are going to see that, if different processors are having their physical clocks are going to be used in the distributed system then how are we going to synchronize them and for that we are going to see a protocol which is called a network time protocol, which does the physical clock synchronization. So, let us begin this kind of discussion, in centralized system there is only one single clock the process gets the time by simply using the system call to the kernel.

So, the next process if it gets the time it will get always the higher time because it is getting from the same clock. So, there is no problem of clock synchronization in centralized system, how are you in a distributed system there is no global clock or a common memory, each processor has its own internal clock and its own notion of a time these clocks, can easily drift seconds per day accumulating a significant error over time also because different clocks tick at different rates; they may not remain always synchronize although they may might be synchronized, when they start this clearly poses serious problem to the application that depends upon the synchronized notion of time.

(Refer Slide Time: 29:55)



Physical Clock Synchronization: NTP

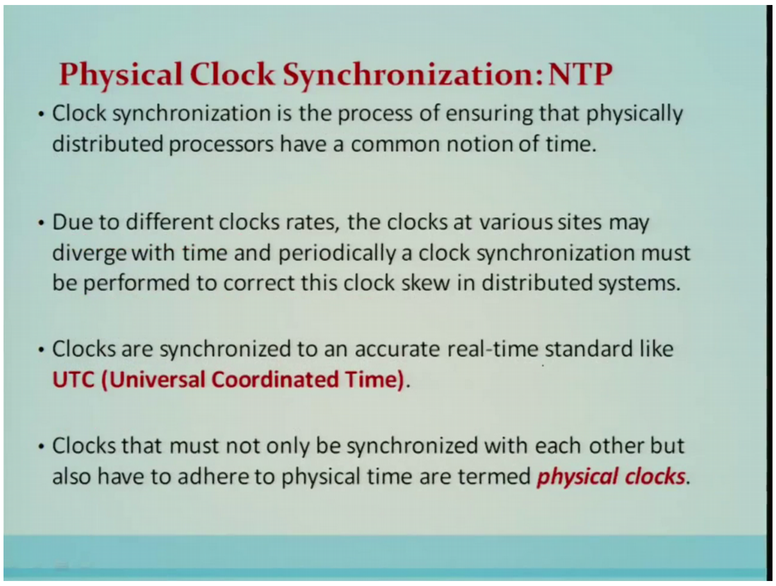
For most applications and algorithms that run in a distributed system, we need to know time in one or more of the following contexts:

- **The time of the day** at which an event happened on a specific machine in the network.
- **The time interval** between two events that happened on different machines in the network.
- **The relative ordering of events** that happened on different machines in the network.

- Unless the clocks in each machine have a common notion of time, time-based queries cannot be answered.
- **Clock synchronization** has a significant effect on many problems like secure systems, fault diagnosis and recovery, scheduled operations, database systems, and real-world clock values.

So, the most practical application algorithm that runs in a distributed system, we need to know the time in one or more of the following context. Now unless the clocks in each machine have a common notion of time based queries cannot be answered. So, clock synchronization is has a significant effect on many problems like secure system fault, diagnosis recovery and so on.

(Refer Slide Time: 30:19)



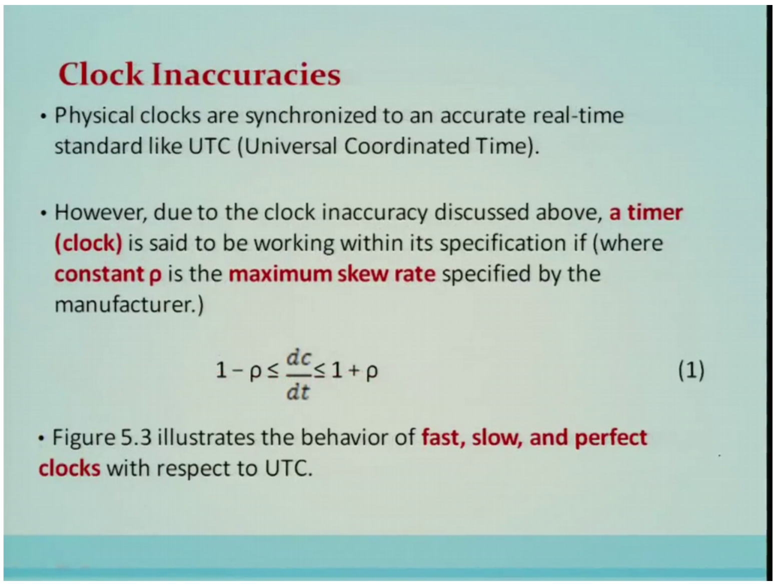
Physical Clock Synchronization: NTP

- Clock synchronization is the process of ensuring that physically distributed processors have a common notion of time.
- Due to different clocks rates, the clocks at various sites may diverge with time and periodically a clock synchronization must be performed to correct this clock skew in distributed systems.
- Clocks are synchronized to an accurate real-time standard like **UTC (Universal Coordinated Time)**.
- Clocks that must not only be synchronized with each other but also have to adhere to physical time are termed ***physical clocks***.

So, clock synchronization is the process of ensuring that physically distributed processors, how common notion of time due to different clock rates. The clocks at various sites may diverge with time and periodically clock synchronization must be performed to correct this clock skew in distributed system, in our day to day life our wrist watch or our wall clock also loses the synchronization and we do the synchronization with a universal coordinated time that is UTC.

So, clocks are synchronized to an accurate real time standard, like universal coordinated time we are not only going to be synchronized, but they have to be synchronized with the with a global clock or a physical time and that is done through UTC universal coordinated time. So, clocks that must not be synchronized with each other, but also have to adhere to the physical time or termed as physical clock. So, physical clock means not only the synchronized set of clocks, but also they have to be coordinated or synchronized with the physical time. So, they should be giving the same set of physical time.

(Refer Slide Time: 31:35)



Clock Inaccuracies

- Physical clocks are synchronized to an accurate real-time standard like UTC (Universal Coordinated Time).
- However, due to the clock inaccuracy discussed above, **a timer (clock)** is said to be working within its specification if (where **constant ρ** is the **maximum skew rate** specified by the manufacturer.)

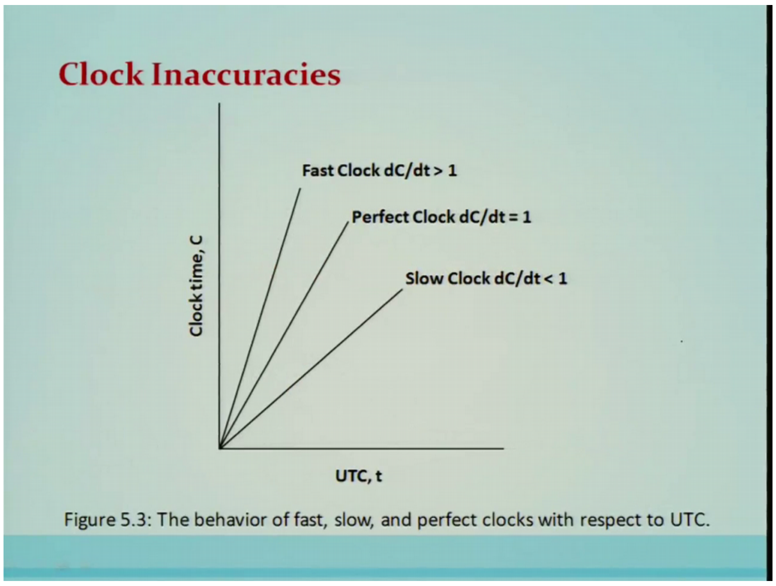
$$1 - \rho \leq \frac{dc}{dt} \leq 1 + \rho \qquad (1)$$

- Figure 5.3 illustrates the behavior of **fast, slow, and perfect clocks** with respect to UTC.

Now, we are going to see into more detail of the clock synchronization and finally, we are going to basically see how these concepts are going to be used in network time protocol. So, clock inaccuracies physical clocks are synchronized to an accurate real time standard like universal coordinated time it is also called UTC.

However, due to the clock in accuracy is discussed above a timer clock is said to be working within a specification, where constant rho is maximum is skew rate specified by the manufacturer; so that means, the clock rate must be bounded by 1 plus rho and lower bounded by 1 minus rho. So, if it is working in this particular equation that is equation number 1, then we have to see that clock is working with the within the specifications.

(Refer Slide Time: 32:34)

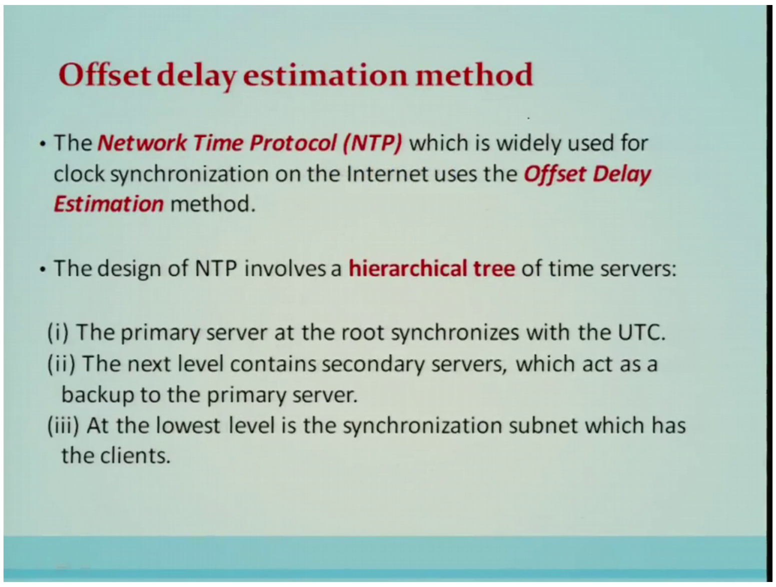


Clock Inaccuracies

Figure 5.3: The behavior of fast, slow, and perfect clocks with respect to UTC.

Otherwise, we can see in these situations, if that particular clock rate that is dC by dt is equal to 1 then it is perfect clock, if it is slower than dC by dt is less than 1 and a fast clock dC by dt is greater than 1. So, the behaviors of fast slow and perfect clock are shown here in this particular diagram of a clock inconsistency.

(Refer Slide Time: 32:55)



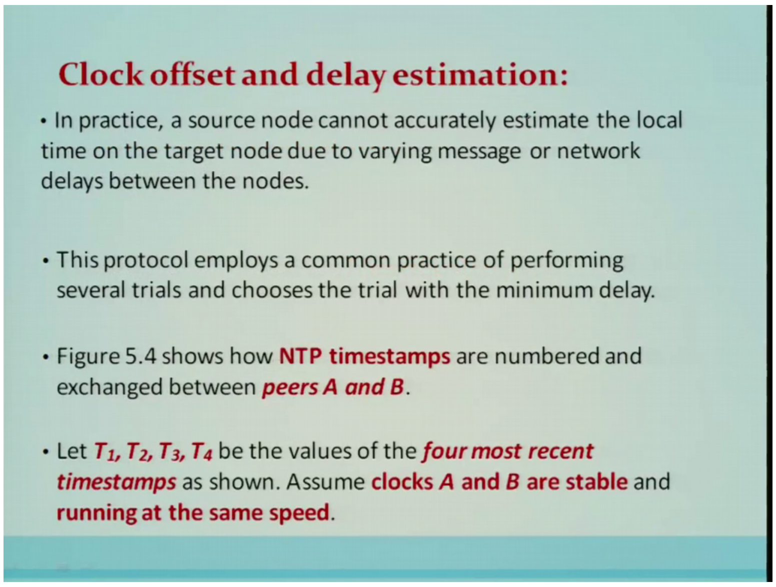
Offset delay estimation method

- The ***Network Time Protocol (NTP)*** which is widely used for clock synchronization on the Internet uses the ***Offset Delay Estimation*** method.
- The design of NTP involves a **hierarchical tree** of time servers:

(i) The primary server at the root synchronizes with the UTC.
(ii) The next level contains secondary servers, which act as a backup to the primary server.
(iii) At the lowest level is the synchronization subnet which has the clients.

Offset delay estimation methods are used in a network time protocol we are going to see what these methods are. So, the network time protocol which is widely used for clock synchronization on the internet, uses offset delay estimation method the design of network time protocol involves, hierarchy that is hierarchical tree of times servers. That means, universal coordinated time will be synchronized in the form of a hierarchical tree. That is the primary server the root synchronizes with UTC the next level contains the secondary servers, which acts as a backup to the primary at the lowest level is the synchronization subset which has the clients.

(Refer Slide Time: 33:43)



Clock offset and delay estimation:

- In practice, a source node cannot accurately estimate the local time on the target node due to varying message or network delays between the nodes.
- This protocol employs a common practice of performing several trials and chooses the trial with the minimum delay.
- Figure 5.4 shows how **NTP timestamps** are numbered and exchanged between ***peers A and B***.
- Let T_1, T_2, T_3, T_4 be the values of the ***four most recent timestamps*** as shown. Assume **clocks *A* and *B* are stable** and **running at the same speed**.

Clock offset and delay estimation, in practice source node cannot accurately estimate the local time on the target node due to the varying messages or the network delays between the nodes. So, that this protocol implies a common practice of performing several trials and choose the trial with a minimum delay. Figure 5.4 we are going to show you the network time protocol timestamps, which are numbered and exchanged between the peers A and B. Let T 1 T 2 and up to T 4 be the values of four most recent timestamps as shown in the next figure. Assume clock A and B are stable and running at the same speed.

(Refer Slide Time: 34:29)

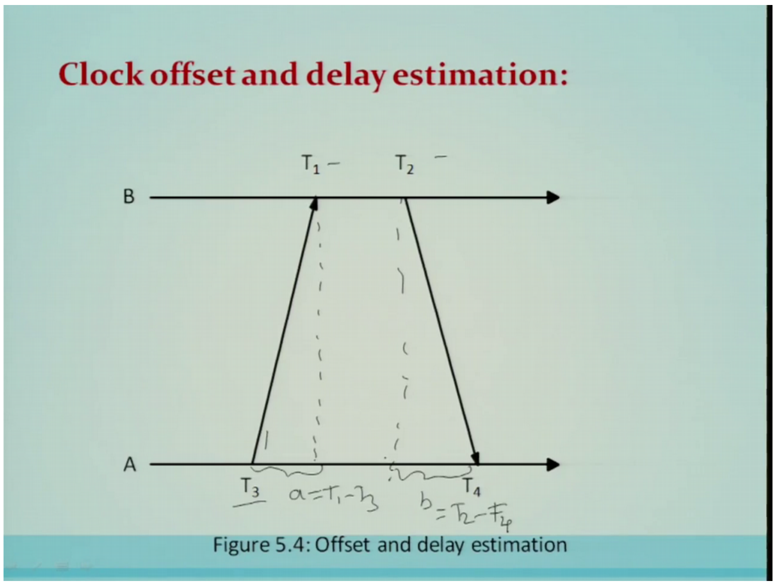
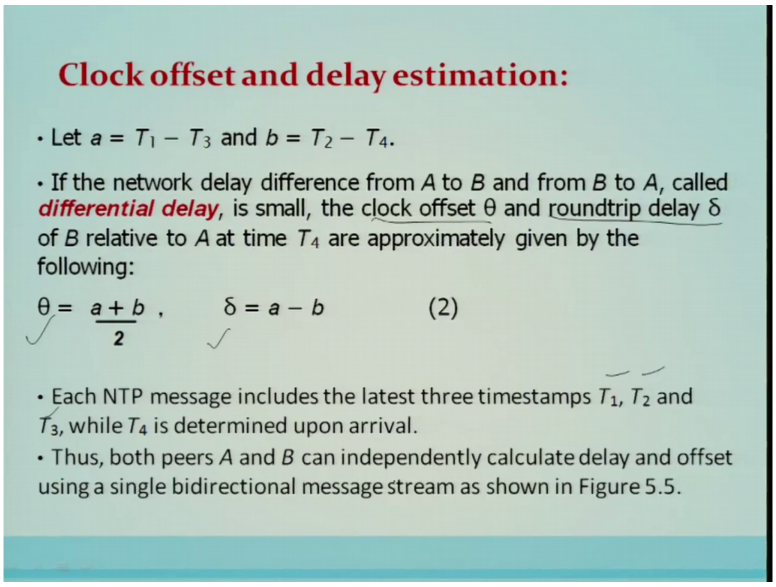

Clock offset and delay estimation:

Figure 5.4: Offset and delay estimation

So, these are the four times we have now shown and we are going to use it for offset and delay estimation.

Now, a is equal to T 1 minus T 3 so this particular portion will become a, that is T 1 minus T 3 and here this portion will become b that is T 2 minus T 4 and this is going to calculate the offset and delay.

(Refer Slide Time: 35:12)


Clock offset and delay estimation:

- Let $a = T_1 - T_3$ and $b = T_2 - T_4$.
- If the network delay difference from *A* to *B* and from *B* to *A*, called ***differential delay***, is small, the clock offset θ and roundtrip delay δ of *B* relative to *A* at time T_4 are approximately given by the following:

$$\theta = \frac{a+b}{2}, \qquad \delta = a - b \qquad (2)$$

- Each NTP message includes the latest three timestamps T_1, T_2 and T_3, while T_4 is determined upon arrival.
- Thus, both peers *A* and *B* can independently calculate delay and offset using a single bidirectional message stream as shown in Figure 5.5.

So, the clock offset which is shown as a theta is nothing but a plus b by 2 and the round trip delay that is little delta is nothing but a minus v. So, each network time protocol message includes the latest 3 timestamp that is T 1 T 2 and T 3, while T 4 is determined upon arrival, does both peers a and b can independently calculate delay and offset using a single bidirectional message stream shown here in this figure 5.5.

(Refer Slide Time: 35:41)

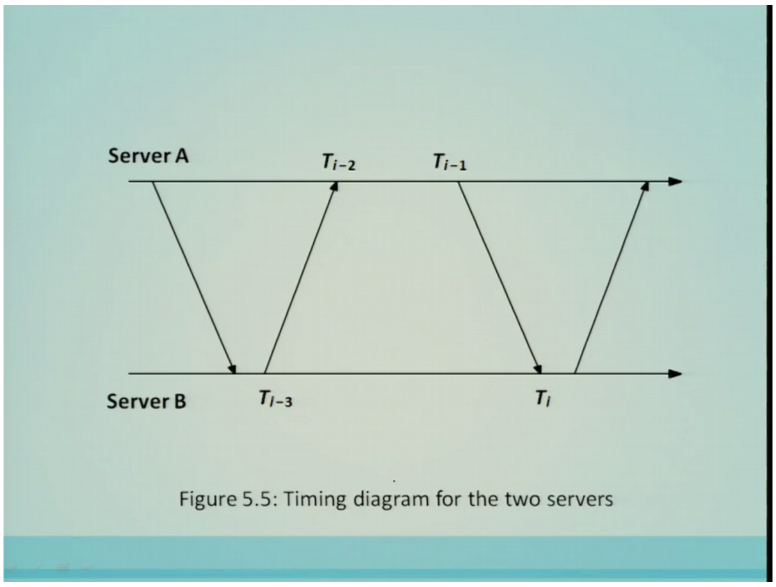
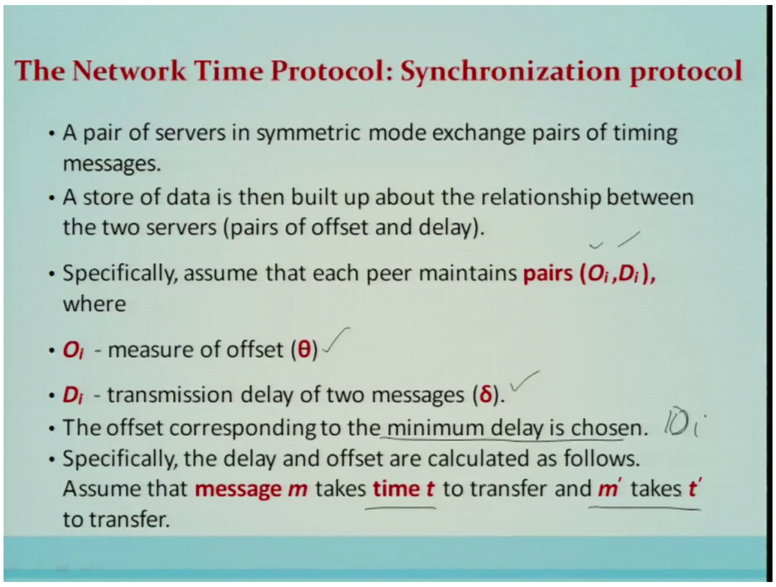


Figure 5.5: Timing diagram for the two servers

(Refer Slide Time: 35:49)



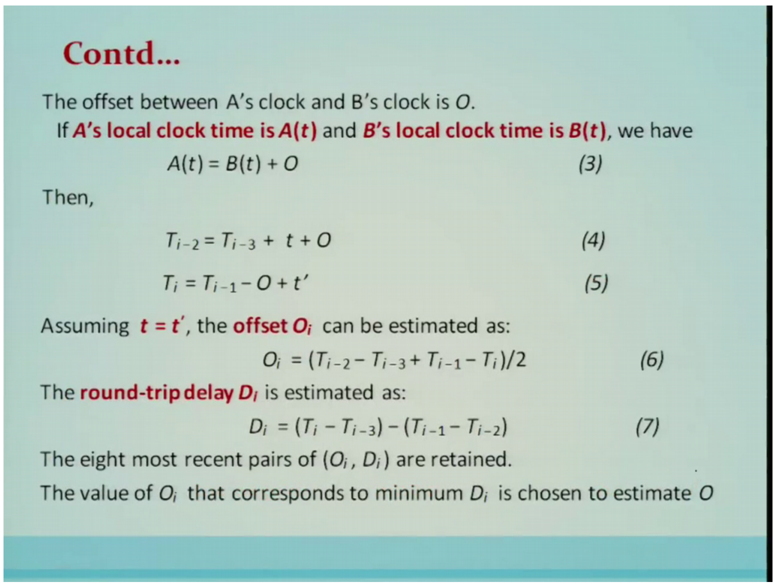
The Network Time Protocol: Synchronization protocol

- A pair of servers in symmetric mode exchange pairs of timing messages.
- A store of data is then built up about the relationship between the two servers (pairs of offset and delay).
- Specifically, assume that each peer maintains **pairs (O_i,D_i),** where
- O_i - measure of offset (θ)
- D_i - transmission delay of two messages (δ).
- The offset corresponding to the minimum delay is chosen. D_i
- Specifically, the delay and offset are calculated as follows. Assume that **message m** takes **time t** to transfer and **m'** takes **t'** to transfer.

Now, with this particular figure 5.5 network time protocol, that is called time synchronization protocol we are going to see. So, a pair of servers in a symmetric mode exchange the pair of the timing messages, store of data is then built up about the relationship between the 2 servers. Specifically assume that the each pair maintains appear that is O i and D I offset and the delay. So, O i is a measure of theta that is the offset and Di is the transmission delay of the 2 messages that is little delta, the offset corresponding to the minimum delay is chosen and that will be basically the O i minimum delay is basically the Di.

So, specifically the delay and offset are calculated as follows, assume that the message m takes time t and message m prime takes t prime to transfer the message.

(Refer Slide Time: 36:49)



Contd...

The offset between A's clock and B's clock is O.

If **A's local clock time is $A(t)$** and **B's local clock time is $B(t)$**, we have

$$A(t) = B(t) + O \quad (3)$$

Then,

$$T_{i-2} = T_{i-3} + t + O \quad (4)$$

$$T_i = T_{i-1} - O + t' \quad (5)$$

Assuming $t = t'$, the **offset O_i** can be estimated as:

$$O_i = (T_{i-2} - T_{i-3} + T_{i-1} - T_i)/2 \quad (6)$$

The **round-trip delay D_i** is estimated as:

$$D_i = (T_i - T_{i-3}) - (T_{i-1} - T_{i-2}) \quad (7)$$

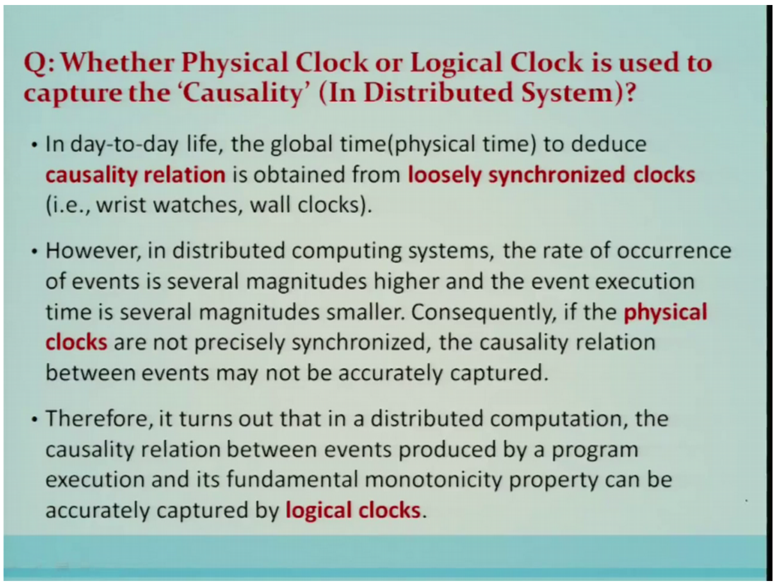
The eight most recent pairs of (O_i, D_i) are retained.

The value of O_i that corresponds to minimum D_i is chosen to estimate O

So, the now the offset between A and B, A's clock and B's clock is let us see O. So, A's local clock time is A t and B's local clock time is B t we have at is equal to Bt plus o, then another equation question number 4 says that T i minus 2 is equal to T i minus 3 plus T plus O, with reference to this particular figure we are going to estimate.

So, same thing that we have seen in the in the previous slides that is being calculated, now assuming T is equal to T prime and offset O i can be estimated as O i is equal to T i minus 2 minus T i minus 3 plus T i minus 1 minus T i divided by 2, similarly round trip D I is estimated as T i minus T i minus 3 minus T i minus 1 and T i minus 2 now having calculated the equation number 6 and 7. So, the 8 most recent these pairs are retained, means they are calculate they are basically estimated calculated. So, the values of the value of O i that corresponds to the minimum Di is chosen estimate of O. So, that we have already seen.

(Refer Slide Time: 38:31)



Q: Whether Physical Clock or Logical Clock is used to capture the 'Causality' (In Distributed System)?

- In day-to-day life, the global time(physical time) to deduce **causality relation** is obtained from **loosely synchronized clocks** (i.e., wrist watches, wall clocks).
- However, in distributed computing systems, the rate of occurrence of events is several magnitudes higher and the event execution time is several magnitudes smaller. Consequently, if the **physical clocks** are not precisely synchronized, the causality relation between events may not be accurately captured.
- Therefore, it turns out that in a distributed computation, the causality relation between events produced by a program execution and its fundamental monotonicity property can be accurately captured by **logical clocks**.

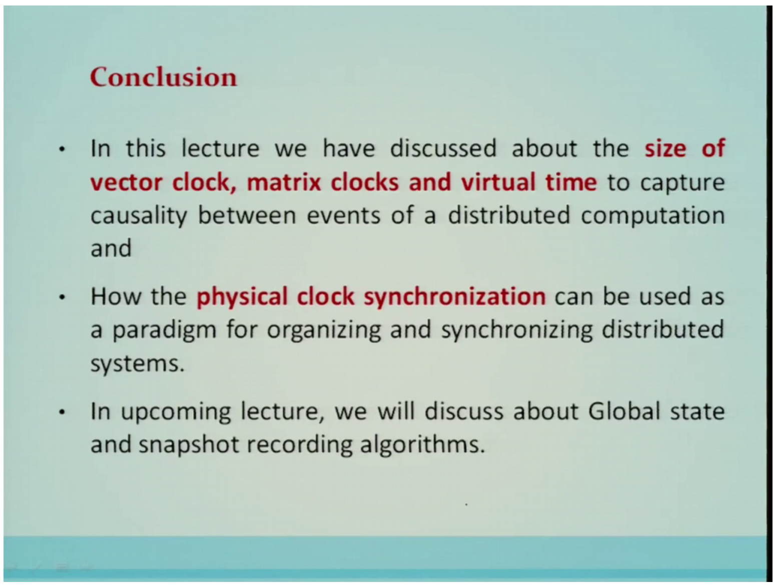
Now the question arises we have seen the physical clock synchronization, we have seen several logical clocks now the question is whether the physical clock or a logical clock is used to capture the causality in distributed system, which of these methods is good in a distributed system? Now in a day to day life the global time that a physical time to reduce the causality relation is obtained from a loosely synchronized clocks, like wristwatch and wall clocks. However, in distributed computing system the rate of occurrence of the events is several magnitude higher and the events execution is several magnitude smaller.

So, consequently if the physical clocks are not precisely synchronized, the causality relation between the events may not be accurately captured. So, this is the most important notion, therefore it turns out that in distributed configuration causality relation between the events produced by the by the program execution and its fundamental monotonicity property can be accurately captured by the logical clocks.

So, if the clocks physical clocks are used and physical clocks using network time protocol, basically they can synchronize up to 10 of milliseconds, but if the number of events which are several magnitude higher and the event execution time is several magnitude is smaller, basically it is not basically fit in to that accuracies than physical clocks even the network time protocol will not be useful for the distributed system applications. So, we have to see what kind of application we are going to run, whether the physical clock can be used with the network time protocol whether can it solve the problem or not. But definitely in all cases the logical clocks which basically capture the fundamental monotonicity property can be captured and can be useful in a distributed system.

Obviously, the logical clock in all situations it will work perfectly fine for distributed application and that is why we have seen so much of different kind of the logical clocks, in this particular part of the discussion.

(Refer Slide Time: 40:41)



Conclusion

- In this lecture we have discussed about the **size of vector clock, matrix clocks and virtual time** to capture causality between events of a distributed computation and
- How the **physical clock synchronization** can be used as a paradigm for organizing and synchronizing distributed systems.
- In upcoming lecture, we will discuss about Global state and snapshot recording algorithms.

So, conclusions in this lecture we have discussed about the size of vector clock, the matrix clock and the virtual time to capture the causality between the events in a distributed system. Then we have seen how the physical clock synchronization can be used as a paradigm for organizing and synchronizing the distributed system.

In upcoming lecture we will discuss about global state and snapshot recording algorithms.

Thank you.