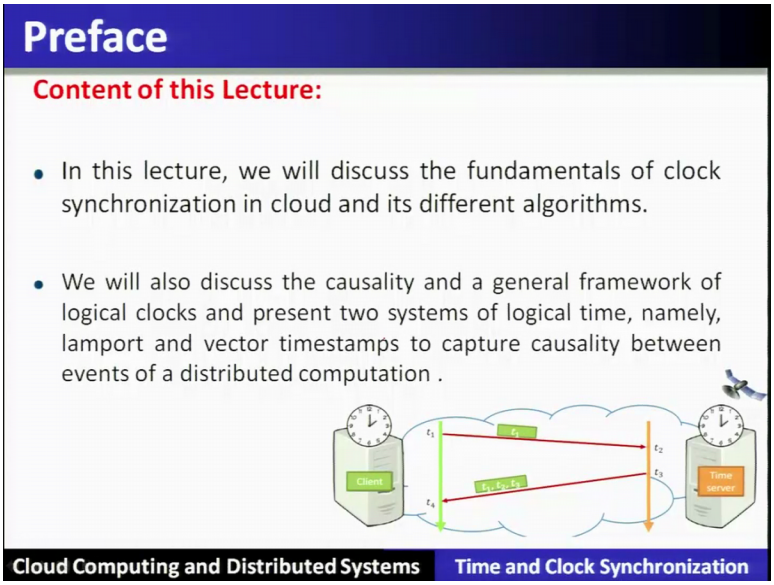
Cloud Computing and Distributed Systems
Dr. Rajiv Misra
Department of Computer Science and Engineering
Indian Institute of Technology, Patna

Lecture – 10
Time and Clock Synchronization

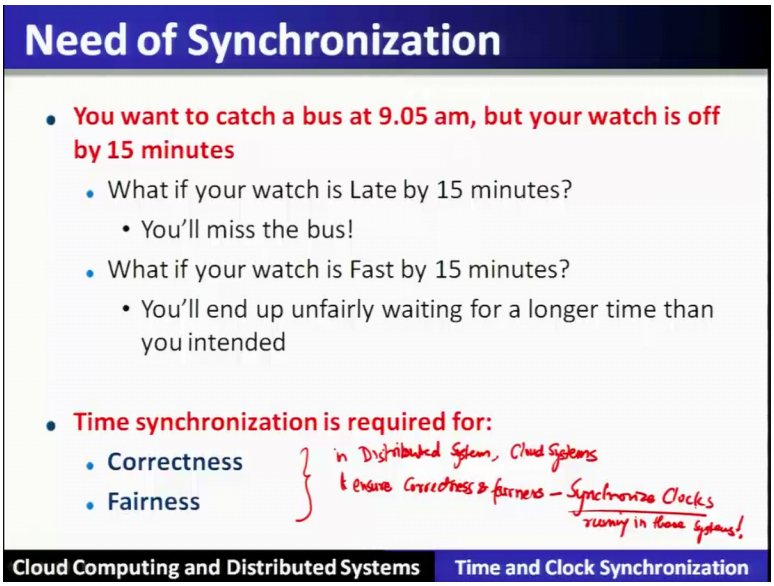
Time and Clock Synchronization in cloud data centers.

(Refer Slide Time: 00:19)



Preface; content of this lecture. We will discuss the fundamentals of clock synchronization, in cloud and also it is different algorithms. Before that we will discuss the importance of time in the distributed system and the cloud systems. We will also cover causality a general framework to avoid the physical clock synchronizations using logical clocks. And we will cover the different logical clock such as Lamport clock, vector clocks to capture the causality between the events in the distributed system and the cloud system.

(Refer Slide Time: 01:21)



So, let us see the importance of time. So, time is one of the most crucial in the distributed system, and as also in the cloud system. But the support for the time and the clock synchronization is nontrivial, it is difficult. In this particular discussion, we will first understand why the timing is required to be maintained in the distributed systems and in particular the cloud computing systems.

Let us see that you want to catch a bus at 9. 05 am, but your watch is off by 15 minutes. In case your watch is running late by 15 minutes and the bus will depart at 9 o clock. So, definitely you are going to miss the bus. On the other hand, if your watch is running fast by 15 minutes, then you will reach earlier and unfairly waiting for a longer time.

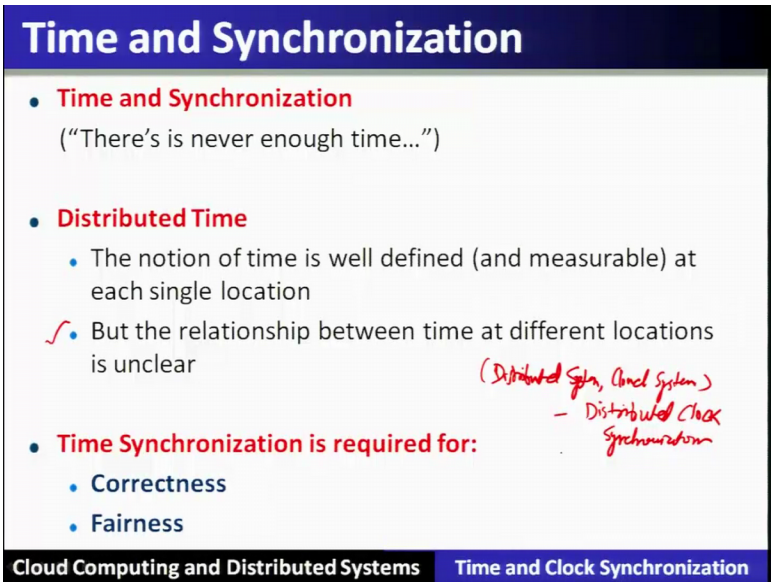
So, this brings up the synchronization of the clock. And we have also seen that if the clock is running late, then you are going to miss a bus; that means, to run the applications and that to ensure the correctness, the clocks need to be synchronized. Hence the correctness is required using the time synchronization of the clocks in the distributed systems. Similarly, if your watch is fast by 15 minutes, then you will reach earlier and you will be unfairly waiting for a longer time. This leads to a fairness that is the compromise with the fairness.

So, time synchronization here also requires that to ensure the fairness in distributed system and a cloud computing system. So, therefore, to ensure the correctness and

fairness in the distributed systems and the cloud systems, to ensure correctness and fairness, there is a need to synchronize the clocks running in these systems.

Now, the question is we will see how this particular synchronization is to be done in such a system.

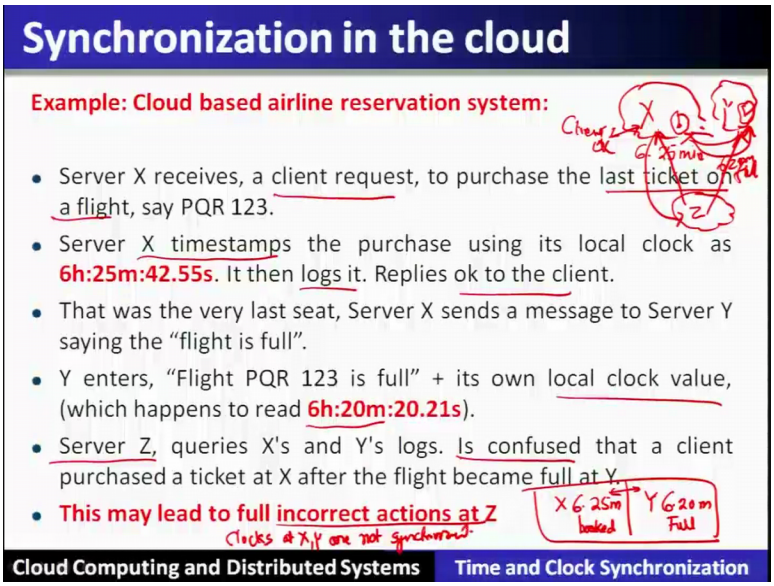
(Refer Slide Time: 05:17)



Now, synchronization of a clock in a single system is a quite trivial, but if there is a distributed system and a cloud computing system where the time is distributed, then the synchronization of the clocks to have a consistent distributed timing at all different location is nontrivial is very, very difficult. So, we are going to see the second case here in our model, which is nothing but a distributed system and the cloud computing system, to ensure a distributed clock synchronization.

So, as we have already seen that time synchronization in this environment that is in the distributed model ensures the correctness and fairness in the different services and applications.

(Refer Slide Time: 06:42)



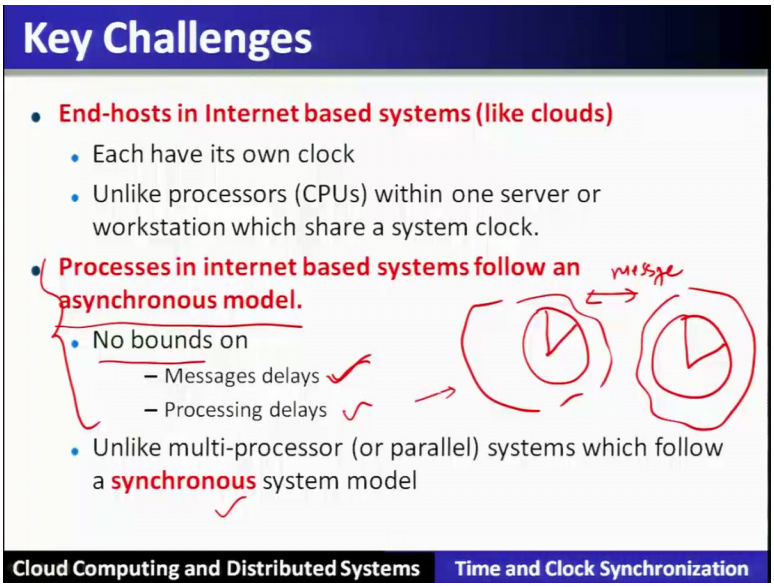
For example, let us say that we are running an airline reservation system using cloud. And we will see the scenario where the synchronization of the clocks are required. So, assume that there is a server x on the cloud and there is another server y and a client request for a reservation. So, the client will send a request to purchase the ticket, which is the last ticket on a particular flight which is maintained at the server x. Server x timestamp the purchase as 6 hours and 25 minutes. That was the last ticket then after the purchase of the ticket and that timestamp at which time this particular ticket was sold out to the client, it will be logged and it will send the reply ok, back.

Since that was the last seat, server x responds to the client and when the next request now comes from the through the server y about booking an another ticket for the same flight which is already full. So, the server x will send the message that it is full to the server y. Now y will enter that the flight is full and will store this information with it is own local time. So, his clock will give another time that is 6 hour 20 minutes. Now another server z queries to x and to y regarding this particular flight seats, but when it gets this information that server x has booked the ticket, the last ticket at 6 hours 25 minutes and y at the time 6 hour 20 minutes the flight was full.

Then in that case the server z will be confused, why because the earlier time that is 6. 20 the server y is showing that the all the seats are filled whereas, later time that is 6. 25 the server x could book a ticket that is the last ticket. Why this problem is there because both

these server x and y their clocks are running different timings. Hence their clocks are not synchronized. So, this particular problem arises because the clocks at x and y or not synchronized.

(Refer Slide Time: 10:47)



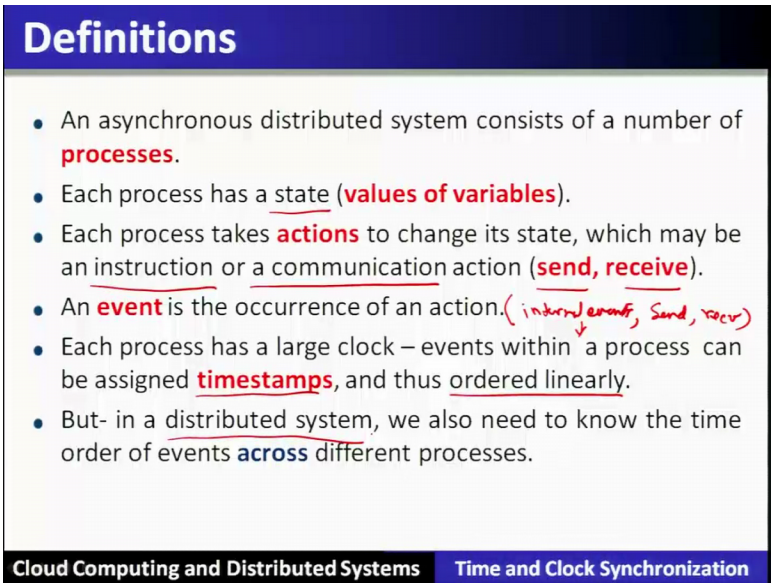
Now, what are the key challenges in clock synchronization problem in distributed systems and the systems like cloud?

Now, here the hosts which are the part of a cloud system and which are internet based connection with each other. Each host in this scenario has it is own local clock and unlike the CPUs within one server or a workstation, which share a common clock here there is an internet based connectivity. So, they have the network and through the network they can communicate they do not have any shared global clock. So, that becomes a problem. So, the processes in the internet based system follow an asynchronous distributed system model. Asynchronous model we have seen in the earlier lecture that they are messages when they come communicate.

For example, these 2 clocks are running in 2 different cloud systems. They can only communicate through the messages. And these messages the delays are not predictable or is no these delays are not bounded similarly the processing delays are also are not bounded in such a system. This asynchronous model is different than the synchronous model where they were in such a system there is a common shared memory where the messages delays are bounded. So, in this scenario the clock synchronization that is in the

cloud system, the clock synchronization across different hosts is nontrivial and we will see that in the asynchronous model how the clock synchronization will create a problem and then you will see the technologies or techniques to solve it.

(Refer Slide Time: 13:18)

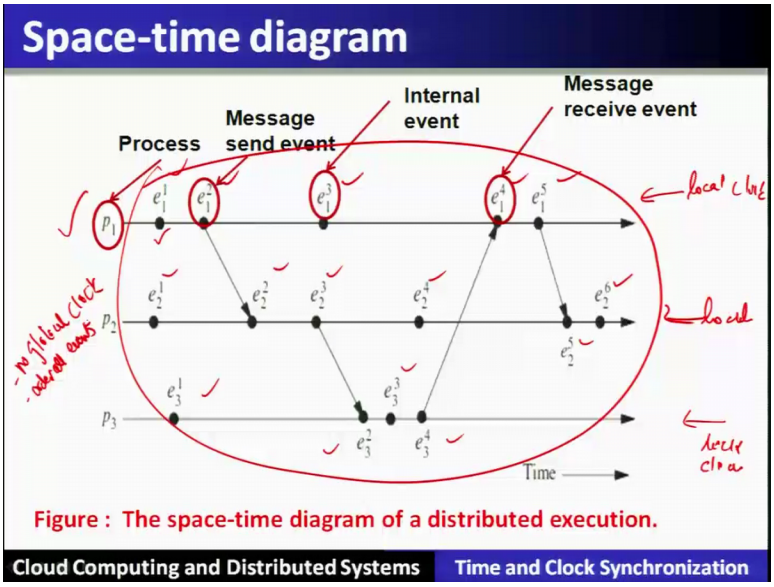


So, let us see the definitions of this asynchronous distributed system model. So, let us consider that the distributed system which is an asynchronous model consists of number of processes and each processes has it is defined state. So, state of a process comprises of it is data, it is stack, it is heap, it is program counter and different registers. So, it is all collapsed and we call it as the state of a process. Now these process takes different actions to change it is states which maybe a instruction or a communication messages so; that means, there are 3 different ways by states can change the first is when the instruction is executed, the second one when a communication takes place that is in the form of a send or a receive of a message

So, intern we define there are 3 different events in the state transitions of such a model, they are the internal events in the form of instructions; then send of a message is a event and a receive of a message is an event. So, each process has a large clock and the events within a particular process which is called a internal events can be assigned with a timestamp of it is local clock and all the events which is called internal events ordered linearly, but in a distributed systems where there are a large collection of such host which

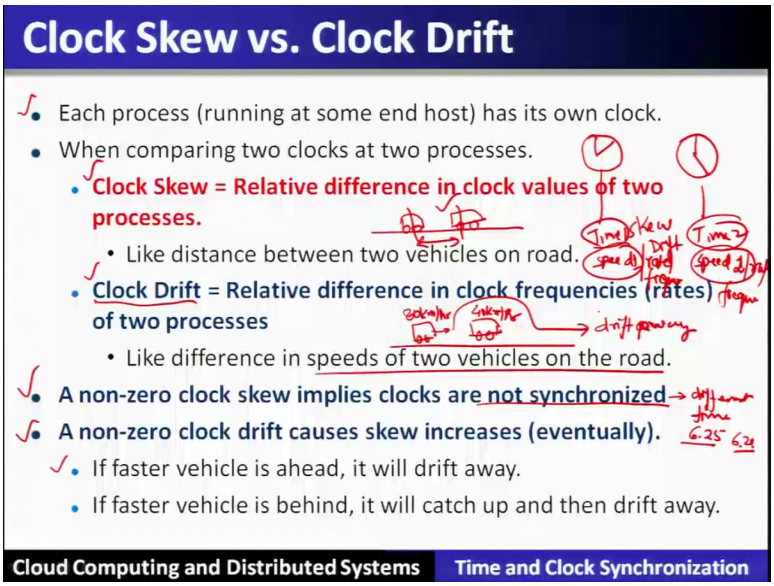
runs their own clocks. So, we need to know the order we need to know the time to order the events across the different processes

(Refer Slide Time: 15:35)



Let us see the example using this particular diagram that a particular process one, it is events 1 2 3 4 5; they are being time through our local clock. Similarly this event P 2 also has it is own clock and can order the events 1 2 3 4 and 5, similarly P 3 also has the local clock which can time this events and using that time all these events can be linearly ordered that is shown here in this space time diagram; however, if we take all the events and want to order them, then there is no global clock which can order all the events there is no global clock which can even which can order all the events in the distributed system.

(Refer Slide Time: 16:58)



So, to do this to see how this, in the absence of a global clock how the events are to be ordered in a distributed system; we will see through a systematic theory.

So, let us see some more definitions in this respect. First we will see the 2 definitions one is the called clock skew the other is called clock drift. Now as you know that each process is running it is own local clock. So, when you compare the clocks of 2 different processes. So, there are 2 things which will immediately be; one is the current timing of both the clocks. Now if these times are not same they are different then it is called a clock skew. So, clock skew is nothing but the difference in the timings across 2 different clocks running in different processes, just like on the road 2 vehicles are running. So, that distance between these 2 vehicles is called the skew similarly the clocks are running with a different timings. So, the difference in the timings is called a clock skew.

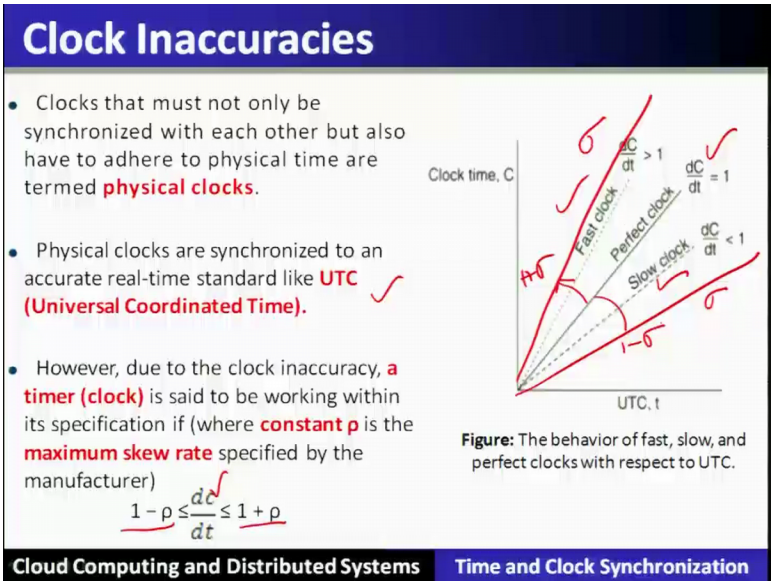
Now, second thing second definition is about the clock drift. Now these clocks runs with a different speed. Speed of a clock 1 speed of a clock 2. Sometimes it is also called as a rate and some people also called as a frequency by which these clocks are running. If they have the same speed or a rate or the frequency, then there is no clock drift, but if the frequencies or the speed is different, then it is called a clock drift. So, that relative difference in the clock frequencies of the rate or the rates of the 2 processes is called a clock drift. For example, 2 vehicles which are running on the road with a different speed;

let us say it is 80 kilometer per hour, and the other one is quite slow that is 40 kilometer per hour.

So, their speeds are different. So, although the 80 kilometer per hour that is a fast running vehicle is behind the slow running vehicle, but after sometime it will cross and then will run away then; that means, drift apart drift away. So, the speed is also important. So, the timing the difference in the timing is called a skew and the difference in the speed is called the drift. So, in this particular clocks or multiple clocks, where more than 2 clocks are there, then these 2 factors are going to play important role in the synchronization.

So, therefore, a non 0 clock skew implies that the clocks are not synchronized; that means, they are giving different timings. Just like in the previous example we have seen one clock was giving the time 6:25, the other clock was giving the time at 6:20. So, they are running at different timings. So, non-zero clock skew implies that the clocks are not synchronized. A non-zero clock drift causes this skew; that means, the difference in their timing to increase eventually. For example, if a faster vehicle ahead, then it will drift away. And if the faster vehicle is behind, then it will catch up and then it will drift away.

(Refer Slide Time: 21:58)

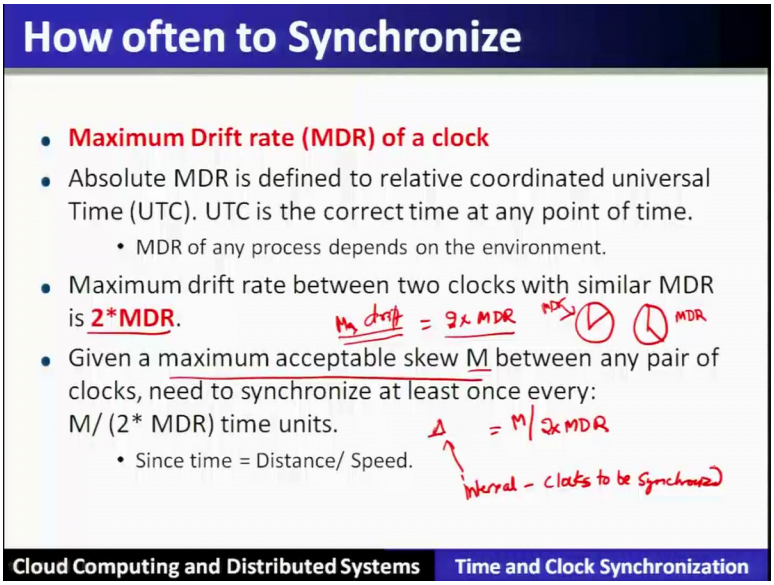


So, having understood the clock skew and clock drift, let us see the clock inaccuracies. So, as far as the physical clocks are concerned that we use;so, not only they have to synchronize with each other, but also has to adhere that physical time, that is the physical

time a common time or accurate time. That is called physical clocks. So, how to get that particular correct time so, that you can synchronize your physical clocks? So, physical clocks are synchronized to the accurate real time, with a standard like UTC universal coordinated time which is supposed to maintain the correct time at all point of time; however, due to the clock inaccuracies a timer of the clock is said to be working within the specifications. So, let us see what are the specifications.

Now, if a constant rho is the maximum skew rate, specified by the manufacturer. It has to be bounded within 1 plus rho and 1 minus rho. So, let us see that dC by dt, if it is 1, then it is the perfect clock; that means, it is synchronized with the standard time. Now if dC by dt is less than 1, then it is a slow clock, and if it is fast, then it is a fast clock. And if it is bounded by these values rho; that means this is 1 plus rho and this is 1 minus rho. Then this is acceptable as far as manufacturer is concerned, but beyond that it need to designated as clock inaccuracies.

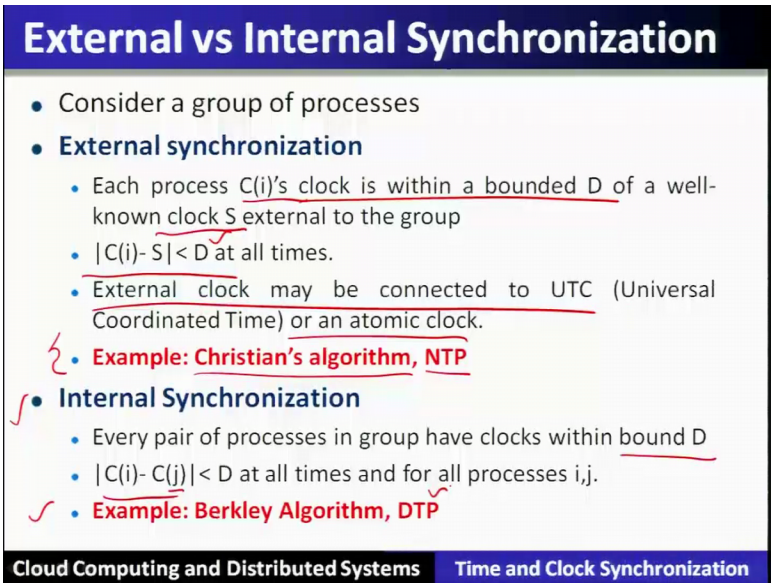
(Refer Slide Time: 24:04)



Now, the question is how often you need to synchronize the physical clocks that depends upon a parameter which is called a maximum drip rate MDR of a clock. The absolute MDR is defined to be the relative to the coordinated universal time UTC. So, MDR of any process depends on the environment. So, the maximum drift rate between the 2 clocks with the similar MDR is equal to 2 times MDR. Because if we consider 2 clocks, maybe one clock is running behind with the rate MDR and the other clock is running

ahead of MDR, then total drift rate here in this case will be 2 times MDR. That is the maximum drift rate. Now given the maximum acceptable clock skew that is m, as you know this is the time, between any pair of the clocks. So, therefore, the amount that time duration when they are required to be synchronized will be this m divided by the script that is 2 times MDR. So, this is the interval. When the clock needs to synchronize; after every such interval, the clocks needs to be synchronized.

(Refer Slide Time: 26:26)

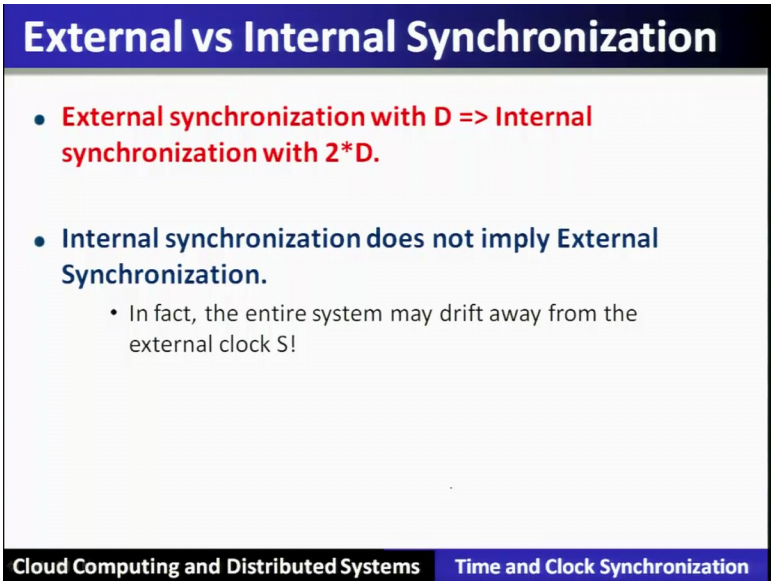


So, having seen this kind of equation, now you will consider two different types of synchronizations. Now if we consider a group of processes in our model of a distributed system, and the cloud system there are 2 kinds of synchronization possible. One is called external synchronization; that means, if the processes used to synchronize their clocks with an external clock; that means, not in a group that is external clock which maintains the correct time then it is called external synchronization.

For example, if a process C i's clock is within a bound D of a well-known clock S external clock S then C i minus S; that means, in the absolute form that clock drift is to be bounded by D at all points of time; that means, it has to be synchronized, that is C i's clock has to be synchronized whenever there is a violation of this condition. This external clock S is nothing but, it is always synchronized at any point of time using universal timing which is also called as a universal coordinated time UTC or the timing which is basically maintained as per the atomic clock.

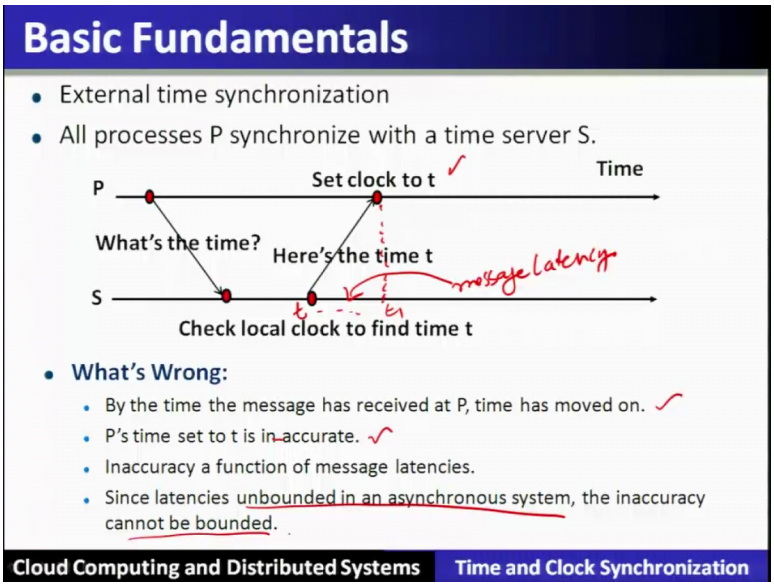
There are two algorithm for this external synchronization, we will cover in this discussion. The first one is called Christian's algorithm the second one is called network time protocol NTP. On the other hand, the group of processes will synchronize their clocks within bound D without the external clock. So; that means, between any 2 process i and j if their clock values is having the difference or skew, then it has to be bounded by D at all points of time. This synchronization can be achieved using the algorithm which is called the Berkley's algorithm. And we will also see the algorithm which is called a data centre time protocol DTP.

(Refer Slide Time: 29:24)



Now, when a small note, to see here is that external synchronization with the bound D will basically implies that, it is the internal synchronization with a 2 times D where as if it is external synchronization that does not implies anything with the external synchronization. Because of the fact that entire system may drift away from the external clock; basic fundamentals of external time synchronization.

(Refer Slide Time: 30:02)

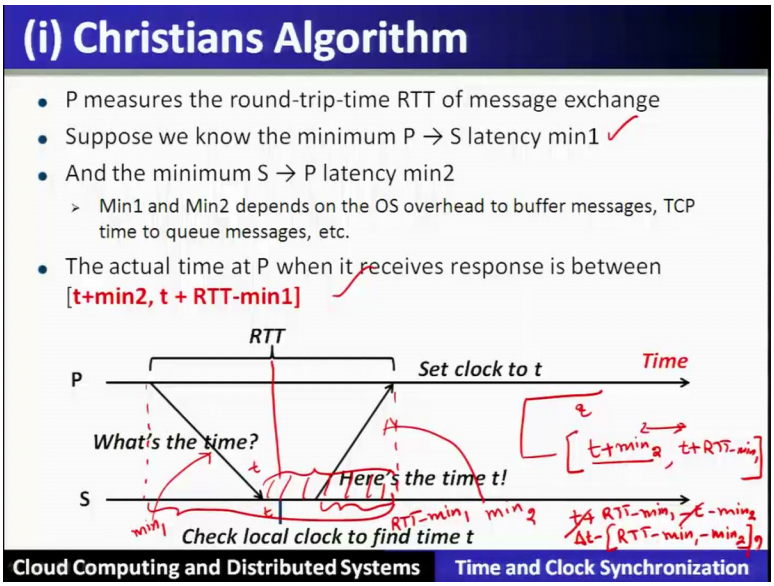


So, let us see through this example. So, all the processes let us say that P will synchronize using a standard time server S which maintains the accurate time. So, such process P sends a message to S what is the time. So, S will check the local time to find out the time let us say t, put in the message and send the message back to the process P here is the time t. So, when the message received by P, it will set it is clock as time t.

So, in this particular manner what is the problem, what is the wrong? So, the problem is that, by the time the message is received at P, the time has moved on; take this particular example. So, this is the time let us say at this instant of S the time is t, but when the message is send to t, this particular time, when this is set as time t the time of the server S will be now t 1; that means, it is a different time, but here the time which is said by P will be the old time hence by the time the message is received at P the time has moved on. So, P S time if it is set to t will become inaccurate.

Now, the inaccuracy is a function of message latency. So, inaccuracy is this particular function of a message latency; that means, if you know this latency and is inform to P. So, P can set this particular value. So, that it can also be accurate or it can reduce the level of inaccuracy. Since latencies are unbounded in an asynchronous system the inaccuracies also cannot be bounded. Hence this particular simple method of synchronizing the clock through a time server S using an asynchronous system to using message passing will have an inaccurate time set.

(Refer Slide Time: 33:09)



Let us see how different algorithms resolves this particular inaccuracy. Christian's algorithm, P measures the round trip time which is called a RTT for the message exchange. So, suppose, we know the minimum latency from sending the message from P to S, let us say this is minimum latency. And the minimum latency if it is travelling from S to P is called as min 2. So, min 1 and min 2, these latencies highly depends upon the operating system overheads to buffer the messages then TCP time involved to queue the messages etcetera.

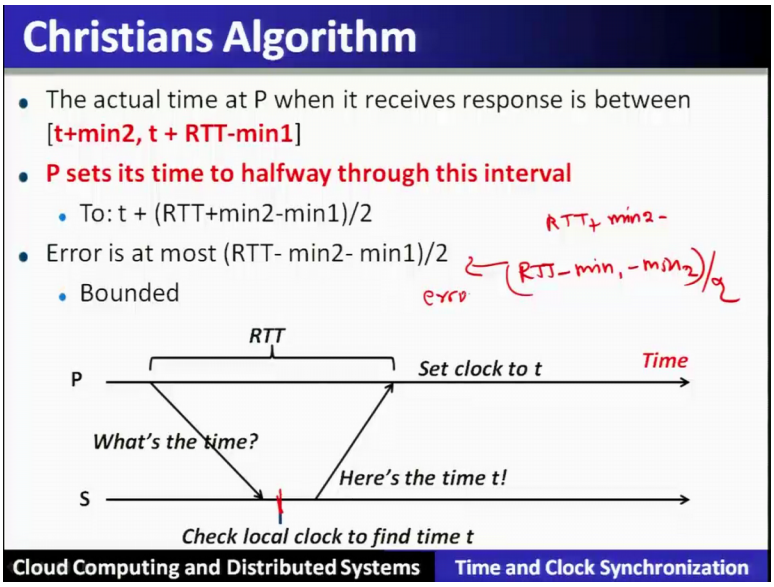
So, the actual time at P, when it is received the response is between t plus min 2 and t plus RTT minus min 1. So, let us see the what is the minimum response. So, minimum response is. So, the actual time at P when it receives the response, the minimum of that bound is t plus because, here this is the time t which is stored and send. So, this particular latency will be added. This is the minimum of latency which is involved and what is the maximum of these values?

So, maximum of these values will be the time t plus this is the round trip time RTT. That is this particular entire time is RTT. So, out of RTT let us say this particular time which is called min 2. If we abstract out of min 1 RTT minus min 1 will be this much of the maximum time. So, t plus RTT minus min 1 will be this particular time and this will be added here. So, that will be t plus RTT minus min 1. So, this is the bound; that means,

the inaccuracies. So, the actual time at P when it receives the response false, the minimum of this particular time t plus min 2.

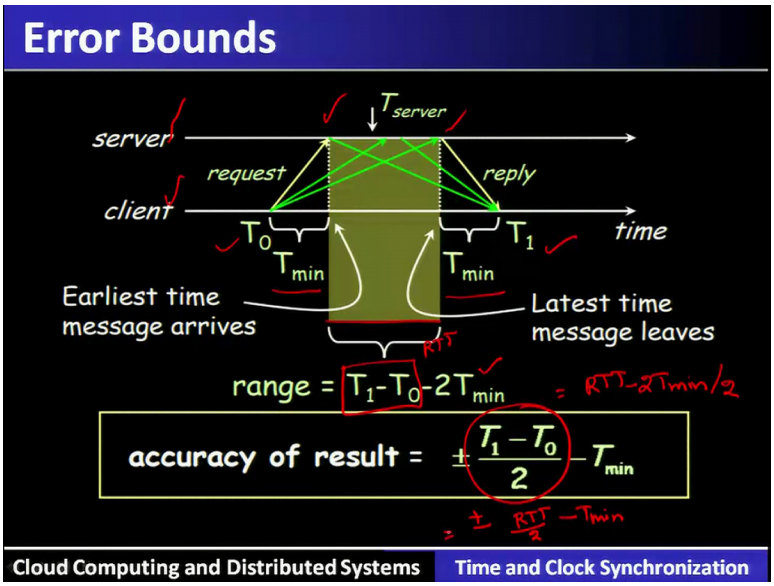
And the maximum that is upper bounded by this particular time t plus RTT minus min 1. So, P will set it is time to the halfway through this particular interval; that means, t plus RTT. So, what is that interval? If we recall that interval will be t, plus RTT minus min 1, minus t minus min 2. That is RTT minus min 1 minus min 2. So, this is the interval and half way of this will be divided by 2. So, let us see.

(Refer Slide Time: 37:28)



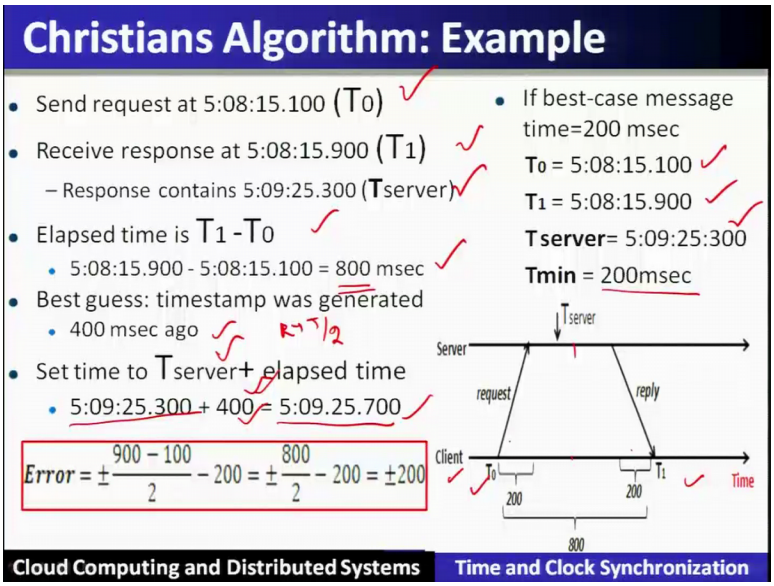
So, half way will be that RTT plus min 2. So, the error will be at most RTT minus min 1 minus min 2 divided by 2 that is this halfway. So, this will be the error and it is bounded by this corresponding value.

(Refer Slide Time: 38:00)



So, again let us see through this particular illustration that this is a server, which maintains the time. And this is the client which want to synchronize it is clocks. Assuming that minimum 1 minimum 2 is equal to minimum. Both are equal in that case without loss of generality; for the sake of simplicity let us understand this. So, the earliest time when the client message will be received is that this start from t 0, plus t min. Similarly, the server will response and the reply will be received the latest time will be when the server sends plus t min. So, that particular range will be that t 1 minus t 0 is the round trip time. And 2 times that minimum value because both are same. It is same as the equation which we have seen. And if we divide by 2 this becomes as this is RTT minus this is t min in. So, this is the accuracy of the result of Cristian's algorithm.

(Refer Slide Time: 39:39)



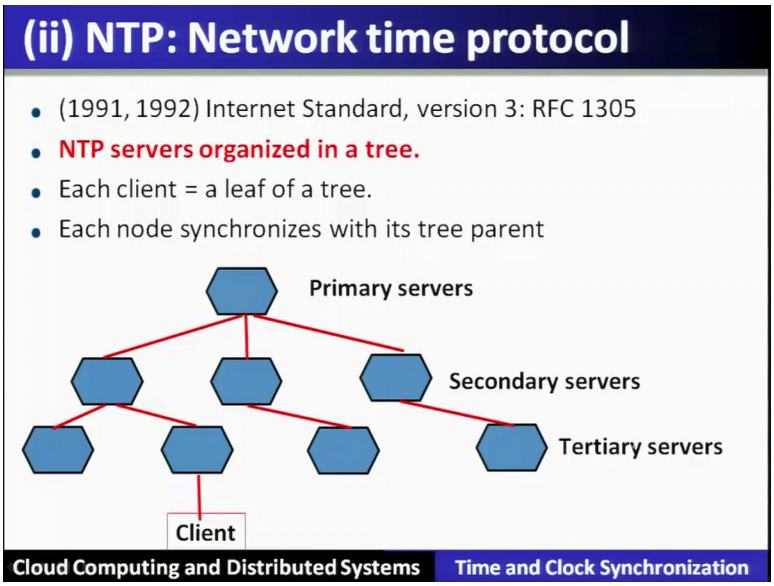
So, here as far as error bounds are concerned, we know that in the synchronization of the clocks we always increase the clock value, but never decrease it. So, if you ever found to be the requirement to decrease the value then you do not do it leave it as it is.

So, it is allowed to increase or decrease the speed of the clock then. So, if the error is too high, it will take multiple readings and average them. Let us understand through another example. Here the server sends this is let us say the client will send at t 0 the request, t 0 is 5:08. It will receive the response at 5:08:15:900. The response contains the timing of the t server. So, it is t 0 and t 1 and t server time 5:09:2 5:300. Total elapsed time is in that particular message that is a difference of timing t 1 minus t 0 that comes out to be 800 milliseconds.

So, if you guess as per the Christian's time so; that means, RTT divided by 2; that means, the guess says that around 400 milliseconds, the message was being originated by the t server. So, the time which will be set by the client will be the time which is sent by the t server, plus this particular average elapsed time that is 400. So, the server time is 5:09:2 5:300. This is the t server plus this offset value is added, comes out to be this much time. So, the clock will be set the client will set the clock value as this one.

Here we can see that t min value is 200 milliseconds why because it is a 400. So, it is 200 and 200 becomes 400 we are in this particular case.

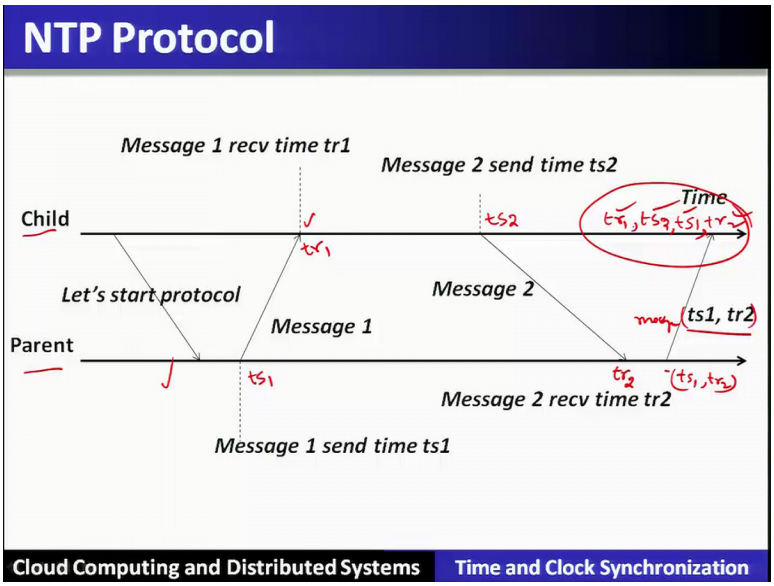
(Refer Slide Time: 42:22)



Now, we will see another time synchronization protocol which is called a network time protocol, NTP, which is a 1991 and 92 it becomes an internet standard, RFC 1305. Here it is assume that the network time servers, they are organized in a form of a tree at each client will be at the leaf of a tree. This particular tree hierarchy of the servers are such that every child will synchronize with it is parent server.

So, on the top let us see you have a primary server which is assumed to be synchronized it is time. And secondary server will synchronize their time with the primary server and tertiary server will synchronize their time with the secondary server and client, will synchronize their time from tertiary server and so on. Let us see the details of network time protocol.

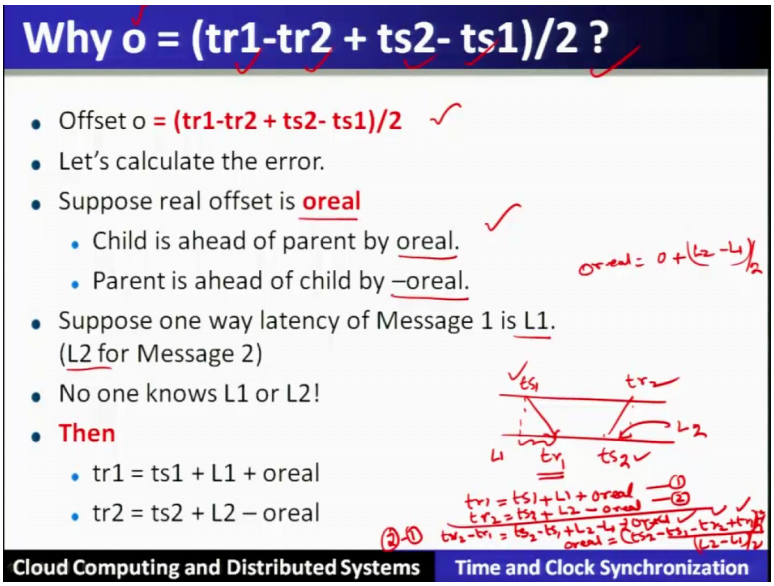
(Refer Slide Time: 43:34)



Here let us see that so, every child will now synchronize it is time with the parent. So, child will send a message. Let us start the protocol. And at time ts 1 the parent will send a message 1 and this particular message 1 will be received by the child at time tr 1. Then child will send another message 2 at the time ts 2 and this particular message will be received by the parent or the server is at tr 2.

Then this particular parent will pack these 2 messages tr 1 and tr 2 here and send in a message, which will be received by the child. So, child will now have it is own timing tr 1 ts 2. And these 2 messages which is received from the message that is tr 2. So, tr 1, tr 2, tr 1 ts 2 all 4 values it has now received. Let us see how this network time protocol uses all 4 values to synchronize the clocks.

(Refer Slide Time: 45:04)

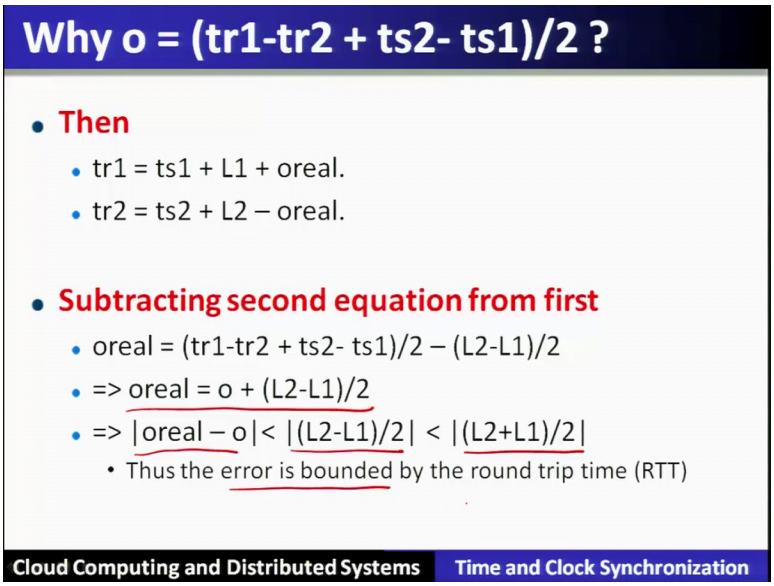


So, using these 4 values it will calculate the offset. So, using this formula tr 1 minus tr 2 plus ts 2 minus tr 1 divided by 2. Let us see how this offset value is calculated.

So, for that let us calculate the total error. Let us assume the real offset is oreal. And we also assume that the child is ahead of the parent by this value oreal. And the parent is behind the child by oreal, which is written as minus oreal; that means, parent is a head of child by minus oreal. Suppose one-way latency of a message 1 is l 1 and the message 2 is l 2 then let us calculate the tr1. So, here let us calculate tr 1. Tr1 is equal to tr 1 plus let us say this particular latency is l 1 plus, the child is ahead of the parent by oreal. Similarly, we can calculate tr2. Tr2 is equal to ts2 plus this latency is l 2, minus oreal; that means, the parent is ahead of a child by minus oreal.

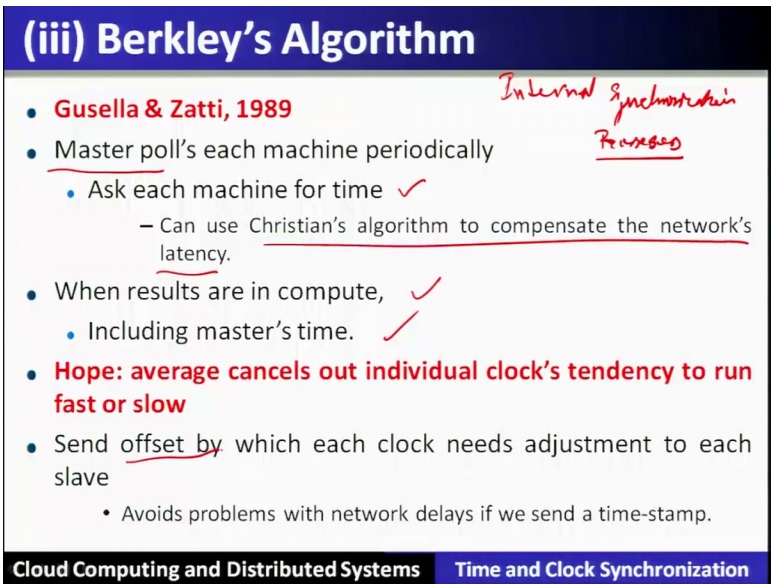
So, this is equation number 1 this is equation number 2. Now we subtract 2 from 1, we will get tr2 minus tr1 is equal to ts2 minus tr1 plus l 2 minus l 1 2 times minus oreal. So, it comes out to be oreal is equal to this will go on the other side. And ts2 minus tr1 then minus tr2 plus tr1. This is one factor plus l 2 minus l 1. And this will be divided by 2. So, this particular value if we rearrange it comes out to be tr1 minus tr2. Then plus ts 2 minus tr1 divided by 2, that we have already obtained this value plus l 2 minus l 1 divided by 2. Now this is an offset. So, oreal is equal to offset plus l 2 minus l 1 divided by 2.

(Refer Slide Time: 48:44)



So, here, we have obtained up to this stage. Oreal is equal to offset o plus l 2 minus l 1 divided by 2. So, if offset we will take on the left side. So, oreal minus offset absolute value will be less than l 2 minus l 1, divided by 2. And this values is also bounded by l 2 plus l 1 divided by 2. So, this is the error bound. So, this particular error is bounded by the round trip time that is RTT.

(Refer Slide Time: 49:24)

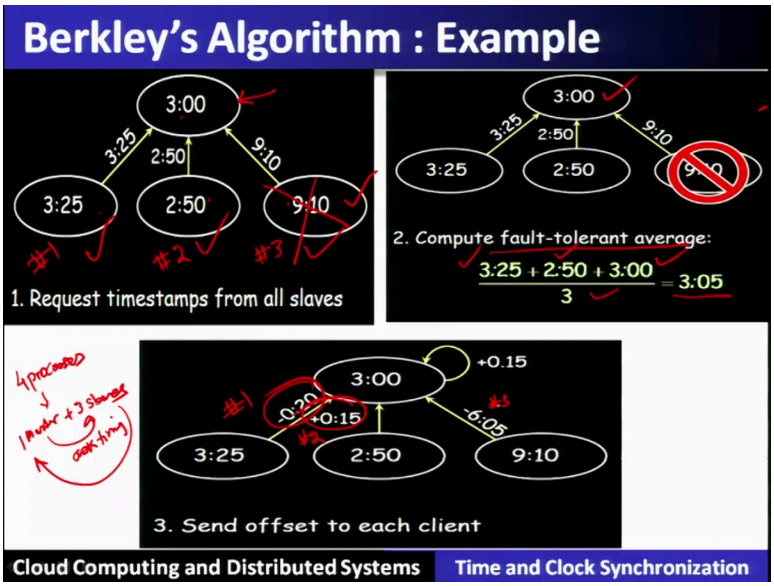


Now, we will see another algorithm that is called Berkley's algorithm. And this is the algorithm for internal synchronization where as the previous algorithm which we have

seen was an external synchronization. Here out of the group of processes, one of these processes will be elected as the master using any leader election algorithm.

And this master will poll each other machines periodically. That is it will ask each machine for it is time. Now as far as the message latency is concerned which basically will be obtained through the previous algorithm, which is the Christian's algorithm. Now when these results are computed, then using the master's time it will be computed as the offsets. Why because the average cancels out the individual clocks tendency to run fast or slow. So, only the offset will be calculated by which is clock needs to be adjusted to each slave.

(Refer Slide Time: 50:56)



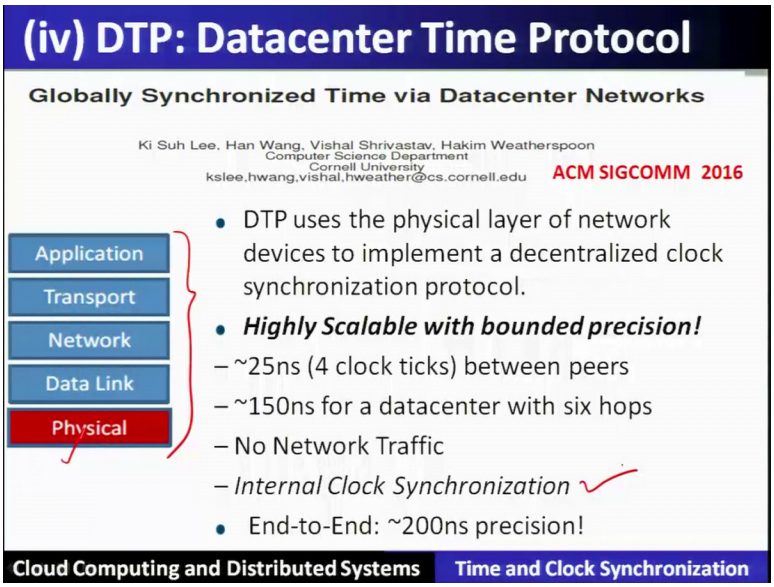
Let us take this particular example, to understand this let us see 1, 2, 3, 4. In the system there are 4 processes. Out of them one is the master plus 3 slaves. So, the master will ask their timings, master will ask the timings of the slaves. So, slaves will send the timing to the master in this particular way.

So, for example, slave one is having time 3:25, slave 2 is having time 2:50, slave 3 is having time 9:10. When all these values will reach to the server is having time 3 o clock. So, server can find out the skew of these particular timings. Now 3:25 and 2:50 according to server's time is good enough; that means, around that around his timing they are running their clocks maybe with some skew. As far as 9:10 is outlier. So, it will not consider in averaging this particular time like here. Now then it will calculate the

average; that means, it will perform the averaging using his value and this average comes out to be 3 o 5. So, with this 3 o 5 this particular time requires this particular offset for server for the slave number one, this outside for the slave number 2 that is plus 15.

And this offset will be calculated for the slave number 3 and this will send to the each client. And hence the Berkley algorithm will synchronize will do the internal synchronization.

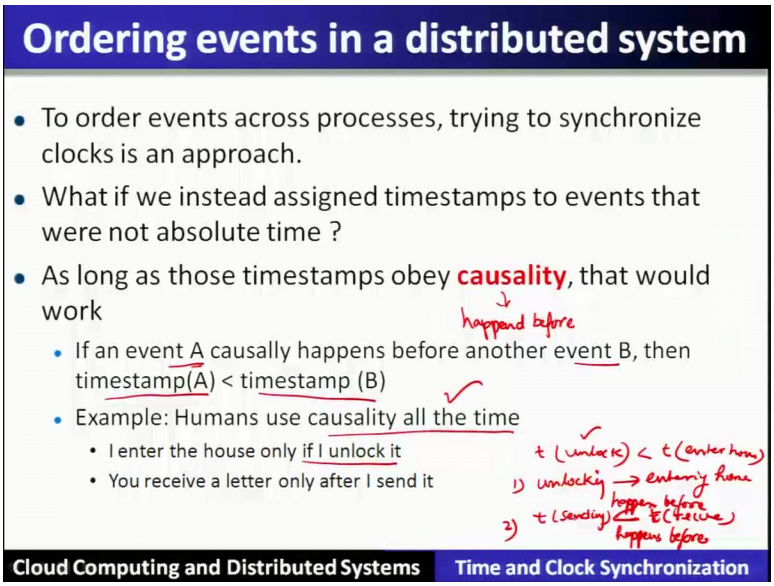
(Refer Slide Time: 53:18)



Now, we will see another time protocol which is also the internal synchronization. That is called data centre time protocol. So, this particular protocol is published in a paper that is called globally synchronized time via datacenter network, which is published in ACM SIGCOMM 2016. Here it considers the layers of the network also into an account for calculating different parameters. So, DTP uses the physical layer of the network.

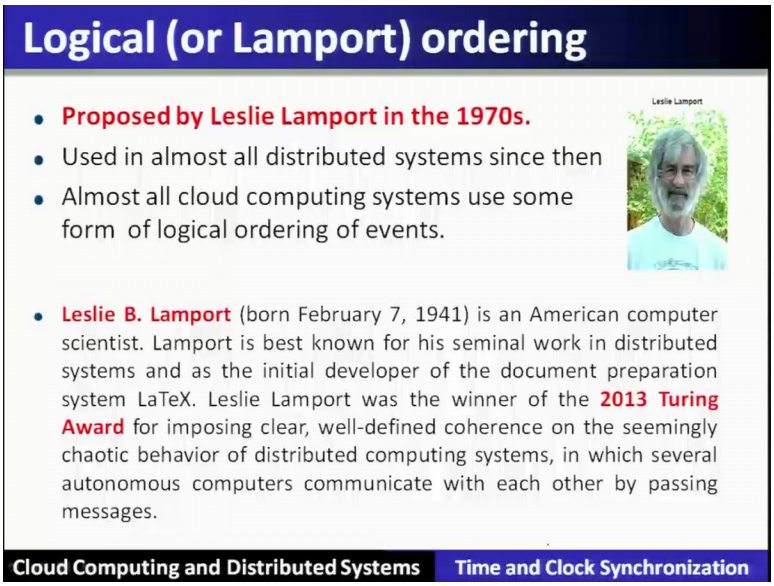
So, the for example, humans use causality at all points of time. Take the example that I enter the house only if I unlock. So, it that means, unlocking of an event the time of that should always be less than the time when you enter the house. So; that means, unlocking happens before the entering into the house.

(Refer Slide Time: 58:39)



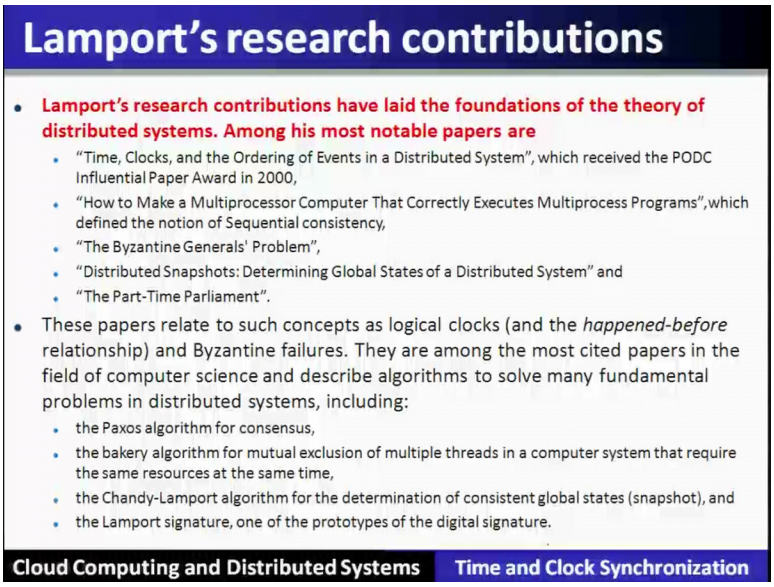
The second example is that you receive a letter only after I send it. So, the sending of a letter happens before the receive of a letter so; that means,. So, this particular way humans are basically using the physical clock and now they are basically ordering the event, for example, here unlocking first and then entering the house or sending letter first and then receiving it later. So, without using physical clock, if you can obey this causality, then that concept can be used without even.

(Refer Slide Time: 60:07)



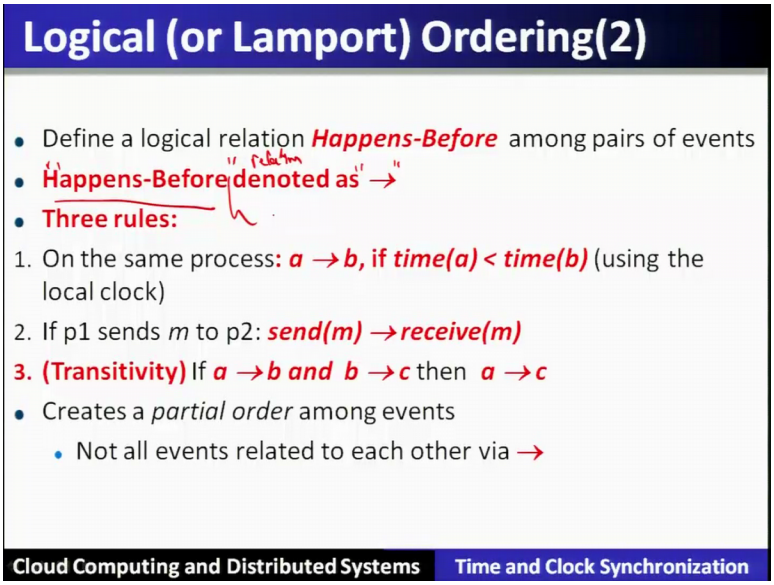
The synchronisation of physical clock; this concept was called as logical ordering given by the or it is also called as a lamport ordering, which was proposed by Leslie lamport in 1970. Almost all distributed system and the cloud system since then are applying this concept.

(Refer Slide Time: 60:29)



So, these are the details of the lamport's contribution. And for which he has got the Turing award that is time clocks and the ordering of events in a distributed systems.

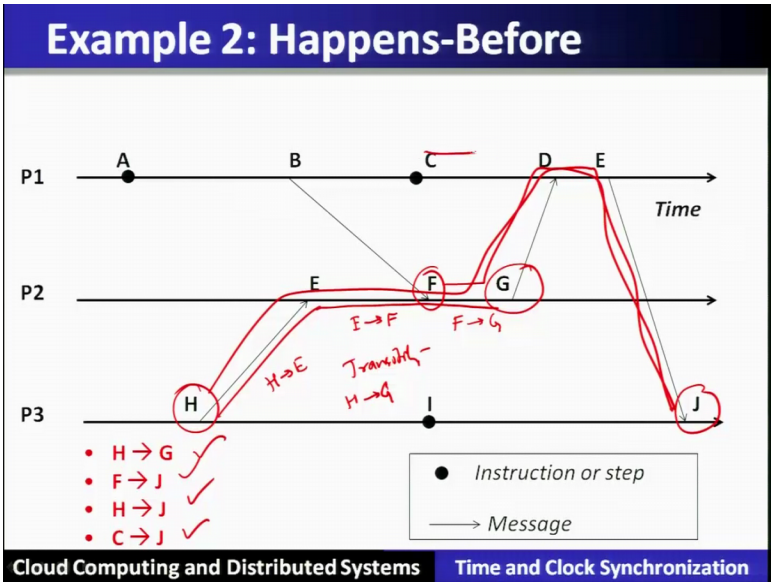
(Refer Slide Time: 60:45)



So, let us see the logical ordering or a lamport ordering, which defines the logical relation which is called happened before relation between the pair of events and is denoted by an arrow. So, this happens before relation is denoted by an arrow. This happens before relation follows 3 different rules. The first rule says that on the same process a has happened before b if the time of a is less than time b, using the logical clock.

The second rule says that if p sends a message m to p, then send off message m has happened before the receive of a message m. Third rule is the transitivity which says that if a has happened before b and b happened before c then a is happened before c. These 3 rules will create a partial order among all the events in the distributed system. Not all events are related to each other that is why it is a partial order via happened before relation.

(Refer Slide Time: 61:59)



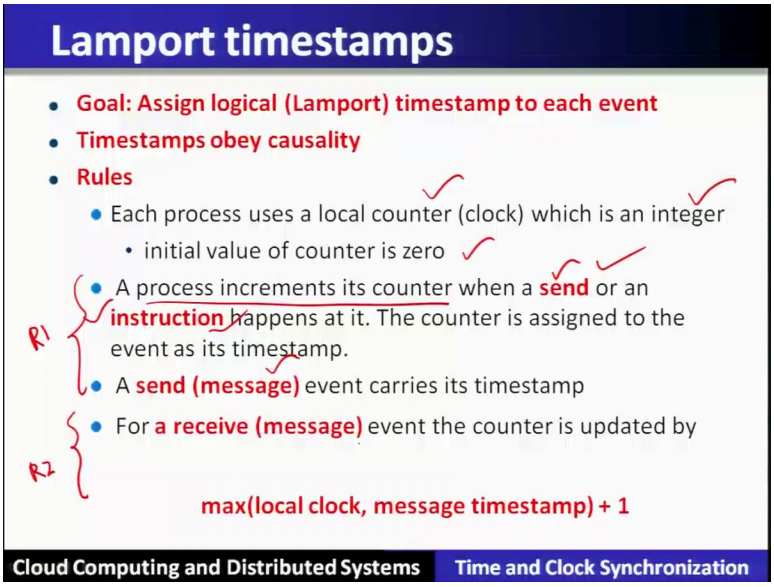
Let us see through an running example. Here we have P 1, P 2, P 3. P 1 is having the events A B C D E, where A is internal event, B is a sendoff a message. F is a receive of a message at the process P 2. Similarly, event j is the received of a message from P 1 at P 3. Let us see that A is happened before B. So, A has happened before B, this we can get through the timing of the internal clock, why? Because A and B they are all they are internal event. Hence the time of A is less than time of B. So, it follows A is happened before B relation.

The second one is b has happened before F. So, b is a send off an event. So, the time of the sending off an event which is maintained by P 1 is less than the time when the message is received as far as the rule 2 is concerned. Hence, B is also happened before f, similarly A is happened before F. So, by transitivity property; that means, A is happened before B and B is happened before F therefore, A is happened before F through the transitive property. So, there is a causal path, this is also called a causal path. So, there exist causal path, it will connect all the events using this happened before relation.

Similarly, let us say that H has happened before G. Here this is the H has happened before G. So, there must exist a causal path which connects H to G; that means, H is a send off event and e is a receive of a message. So, H has happened before E Similarly E and F they are internal events of P 2. So, E has happened before F and F and G their internal events of process P 2, F has happened before G. So, using transitivity property H has happened before G. Similarly, F has happened before J. So, F F has happened before J. So, there must exist a causal path.

Similarly, H has happened before J. So, H has happened before j there exist a causal path. C has happened before J. So, from C there exist a causal path which connects C and J.

(Refer Slide Time: 65:15)



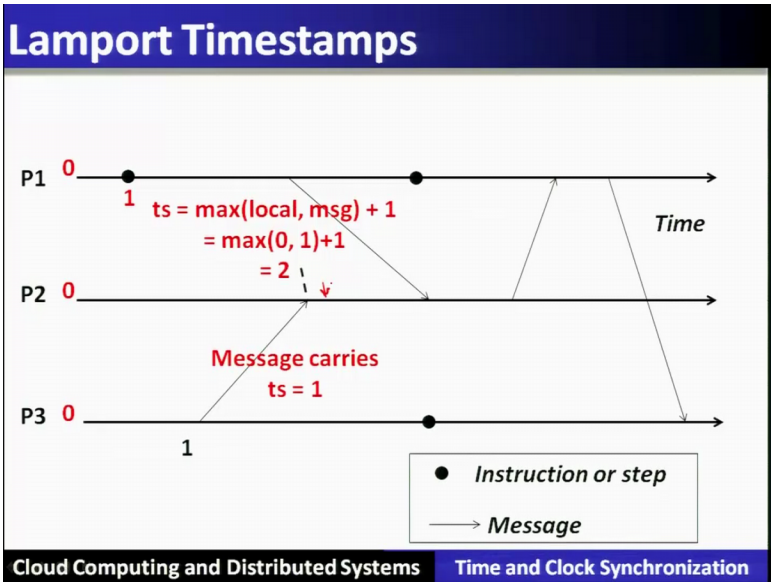
And hence this that particular lamport clock is able to capture the causal relation. Now, the implementation of these timestamps to each event; so that let us, assign a logical

timestamp without a physical clock which will obey the causality. Let us see the implementation using a logical clock which follows the 2 different rules the first rule says that each process uses a local counter which is called a logical clock, which is nothing but an integer which is in a slice 0. A process increments it is counter when it executes the send off a message or the internal execution of an event happens within that particular process.

The counter is assigned to the event as it is timestamp. So, the process increments it is counter whether it is a internal event or it is a send off a message event. And this particular timestamp will be carried with the send off a message. So, this is the rule number one. The rule number 2 says that when a message is received, receive of a message event will update the counter in this form. First of all, the timestamp which is carried in the message and it is local clock, the maximum of these 2 values will be taken up and it will be incremented, that is rule number 2.

So, this will provide the lamports timestamp.

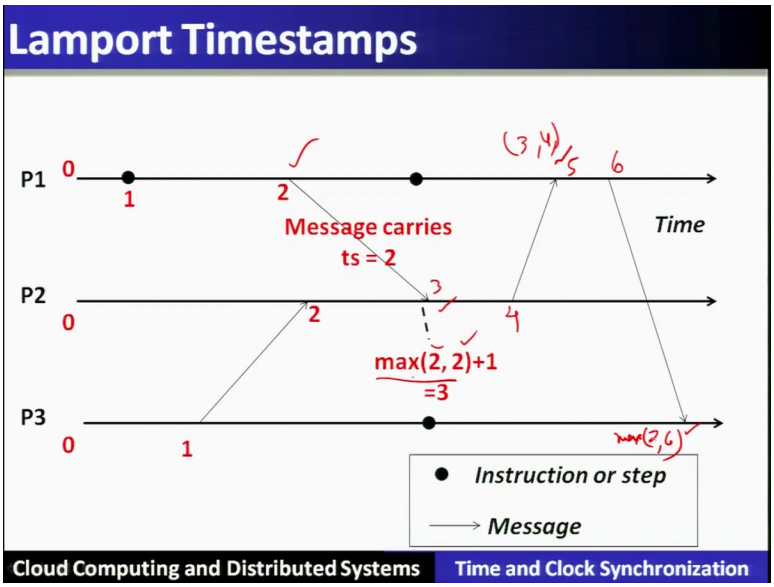
(Refer Slide Time: 67:09)



Let us see through an example, here all the counters are in a slice to 0 as we have seen in the implementation. Then process P 1 will have it is local clock will be in a slice to one or will be incremented from 0 to 1 because it is a internal event. Similarly, as far as process 3 is concerned, the send off a message hence it is clock will be incremented from 0 to 1, and this will be contained in the message and will be send.

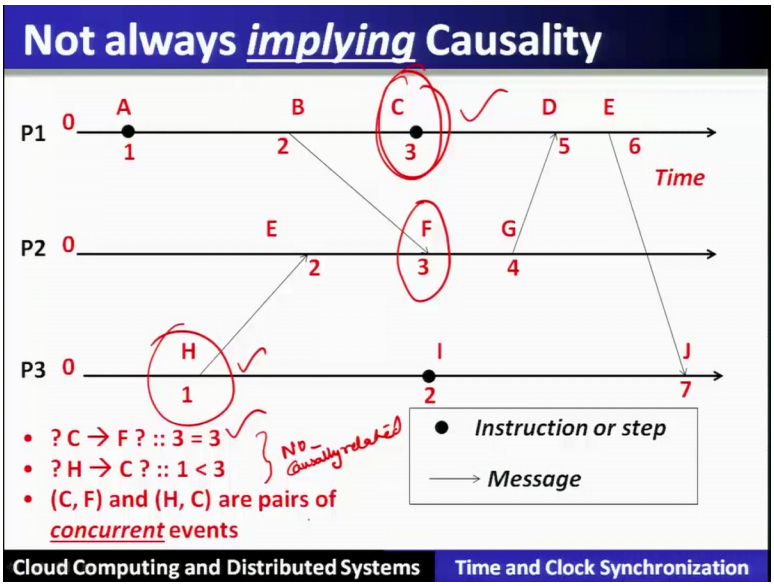
Similarly, when the message will be received, then we will follow the rule R 2 which says that the message which is carrying 1 and the local clock is 0 the maximum of 0 and 1 will become 1. And when it will be incremented the timestamp here will be equal to 2.

(Refer Slide Time: 68:16)



Now, when this particular message will be received, here we can take the maximum of 2 and 2 that become 2 and plus 1 that become 3. This particular event 3 plus 1 it will become 4. And 4 when it will reach over here, 4 and this is 3 the maximum is 4 plus 1 that becomes 5 and 5 plus 1 this will become 6. And when you will it will reach over here; so, the maximum of 6 and 1 2 2 that becomes 6 plus 1; that is 7.

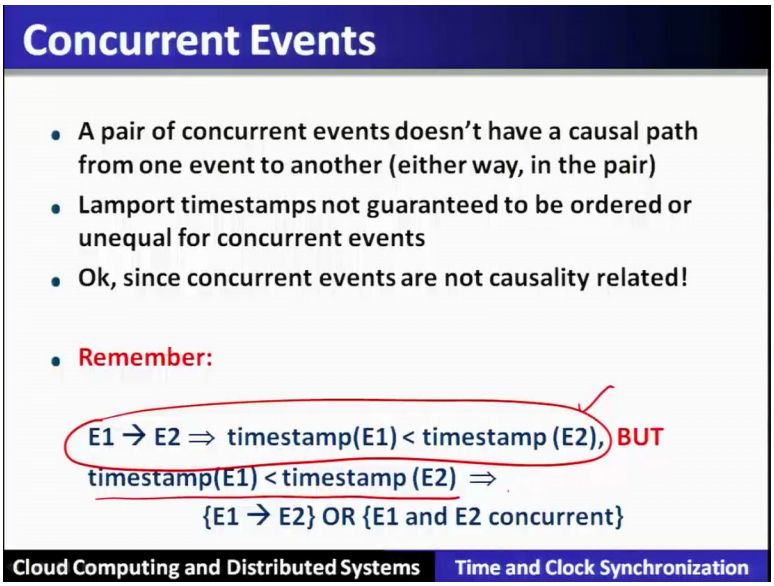
(Refer Slide Time: 69:05)



So, all these we have already explained. And they are obeying the causal properties for example, A has happened before B. So, the timestamp also is less in case of A. B has happened before F. So, the time is A B is less than F. A is happened before F. So, the timestamp A is less than F. Now we have seen the example of the events which are casually related they are following the lamports logical clock. But not all events are causally related; that means, not always implies the causality among all the event that is why it follows the partial order relation.

Let us see through the example C and F although they are timestamp is 3. So, do we infer that C has happened before F? Similarly, H and C H and C; so, the time of H is 1 and the time of C is 3. So, can be infer that H has happened before C. No, we cannot, why because they are not causally related. Hence they do not follow this happened before relation. Hence they are called as concurrent events. So, C F and H C they are the pairs which are not obeying the causality relation. Hence they are called concurrent events here in this case.

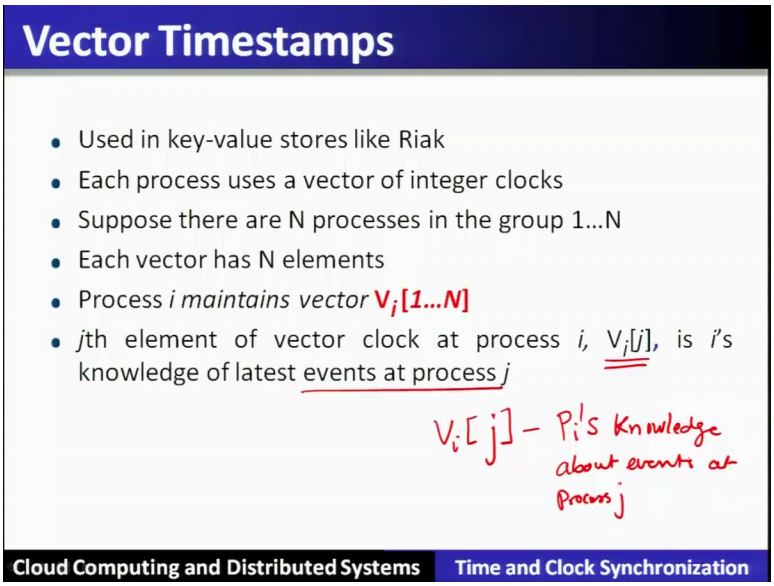
(Refer Slide Time: 70:55)



So, a pair of concurrent even does not have the causal path from one event to another. So, lamport timestamp not guaranteed to be ordered or unequal for concurrent events. This is since the concurrent events are not causally related. But what we can see here as I take away that, if E 1 has happened before E 2, this implies that the timestamp of E 1 is less than the timestamp of E 2. This is what is known here in our logical clock given by the lamport.

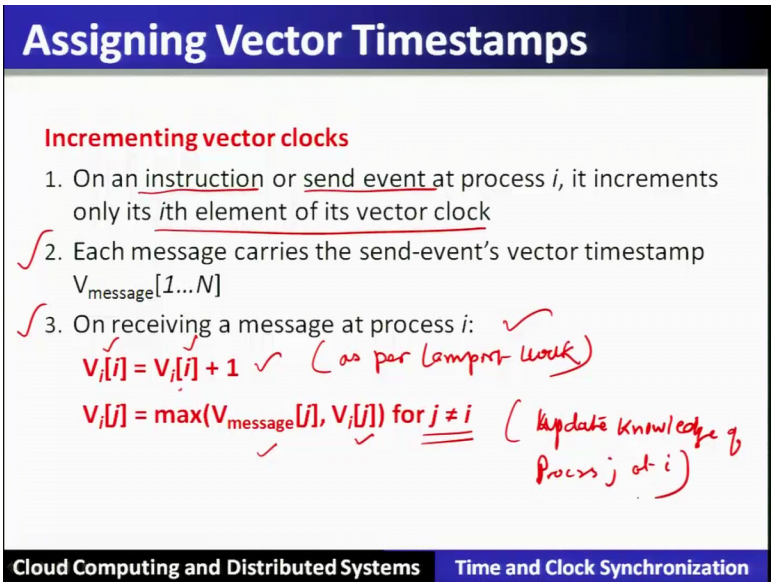
But if we see the timestamp, only and can be inform whether the events are happened before or they are concurrent. That we cannot just by looking a timestamp we cannot infer whether the events are happened before or; that means, causally related or they are not causally related they are concurrent event. That is the drawback of the lamports clock.

(Refer Slide Time: 71:57)



So, the next clock is called a vector timestamp or a vector clocks; which is used in the key value stores like Riak. Each process uses the vector of integer clocks. Suppose there are n processes in a group one to n and each vector has elements. So, process I will maintain the vector i for each process one to N. Now jth element of the vector at process i is denoted by; this is which indicates that the process P i's knowledge about the events at process j.

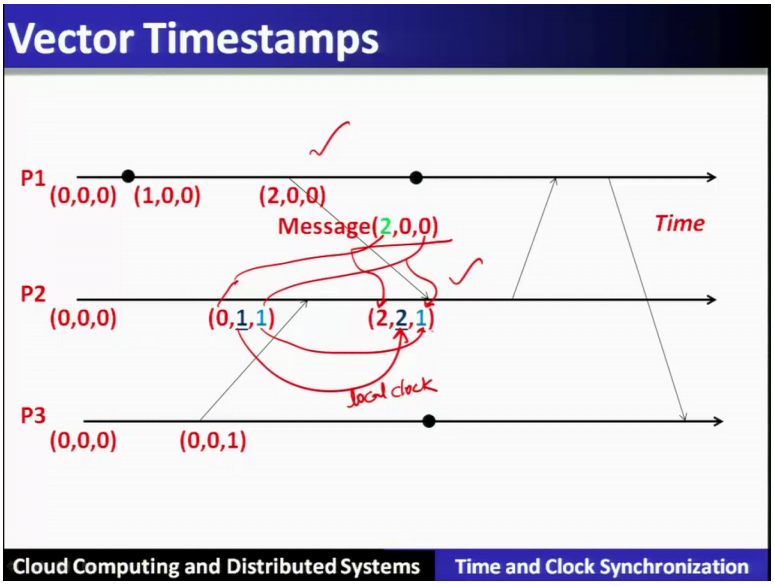
(Refer Slide Time: 73:09)



So, this particular notation we are going to use further in this understanding of vector clocks. Assigning the vector timestamps; that is, on an instruction execution or send off an event at process i increments only ''th element of it is vector clock. And this particular timestamp the entire vector will be timestamp. And so, the message carries the send events vector timestamp within it. When the message will be received at a process i, then process I will increment it is vector. As we have seen this is as per the as per the lamports clock. And then it will take the maximum of the vectors of j and it is knowledge about other j excluding it is own vector; that means, the message the vector of the process j will be used to update it is own clock. So, this is a new step that is about update the knowledge of process j at i.

So, whenever there is a message exchange the clock values will be updated based on the information which is being sent the knowledge of other processes. So, let us see through an example. Here all the counters are in a slice to 0.

(Refer Slide Time: 75:17)



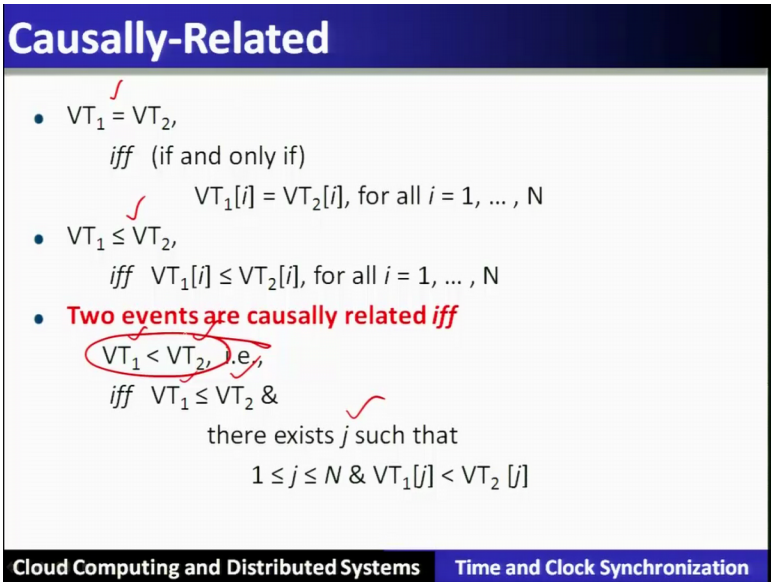
Now, with the event at P 1, only his clock value will be incremented, similarly at for P 3 only this value will be incremented. And whenever there is a message exchange, then these other values will also be updated.

For example, the receive of a message will increment it is local variable, but the message contains the information about the P 3's knowledge. So, this will be updated. So, here it will be updated for the other value. So, better timestamp will not only update it is

knowledge, but whenever a message exchange it will update it is vector based on the based on the information of the other processes. So, this is another example that if when P 2 will send this message with the timestamp 2,0,0. And it will be packed as a message when it will receive at P 2 then P 2 will increment, it is local clock as per the lamports logical clock. Whereas, 0 and 2 they will update to the maximum of 2. So, 1 and 0 will be updated over here. So, this is not updated, same value will be because it has more knowledge than the pone about P 3.

Now, we will see that using vector clock we can perform some operations on the vectors. And this will be used to compare the vectors just by looking the vector we can infer whether the whether 2 events are causally related are there or not.

(Refer Slide Time: 77:14)

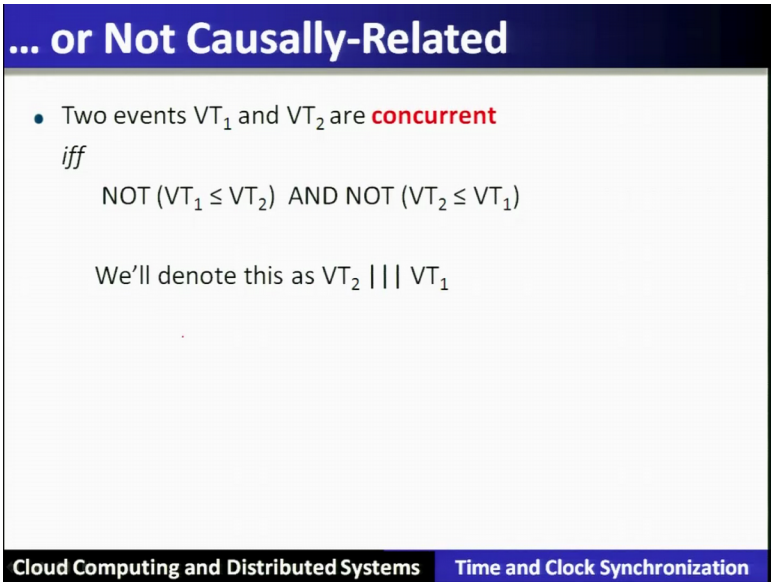


So, that means, if the vector VT 1 will less than vector VT, VT 2. This will implies that there exist 2 vectors which are VT 1 is less than or equal to VT 2 and also there exist a particular value which is strictly less than, then basically this particular condition holds. And that shows that 2 events are causally related.

So, 2 events are concurrent if neither VT 1 is less than or equal to VT 2 and nor VT 2 is less than or equal to hence they are causally related. Vector timestamp obeys the causality relation. This also has been identifying the concurrent events. Let us take an example. So, C and F so, vector C is 3 0 0 F is 2 1 1 and 2 1 1. Now this particular value is higher than this one where as the other elements they are less than this values. So, it is

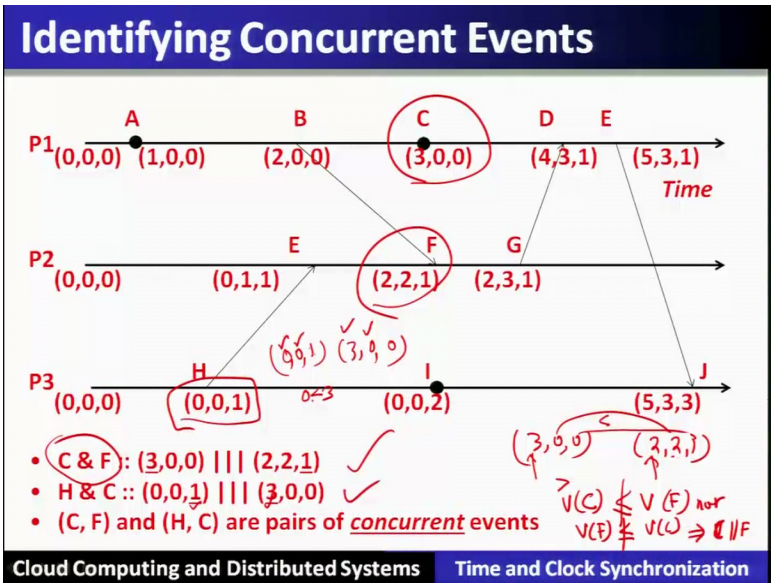
not less than; that means, there is no neither it is neither C is vector of C neither it is less than or equal to vector of C. F nor vector of F is less than or equal to vector of C.

(Refer Slide Time: 78:01)



Hence they are C and F they are concurrent events. Similar H and C, you can see that this one is less than 3 H and C. So, 1 0 is less than 3 that is fine.

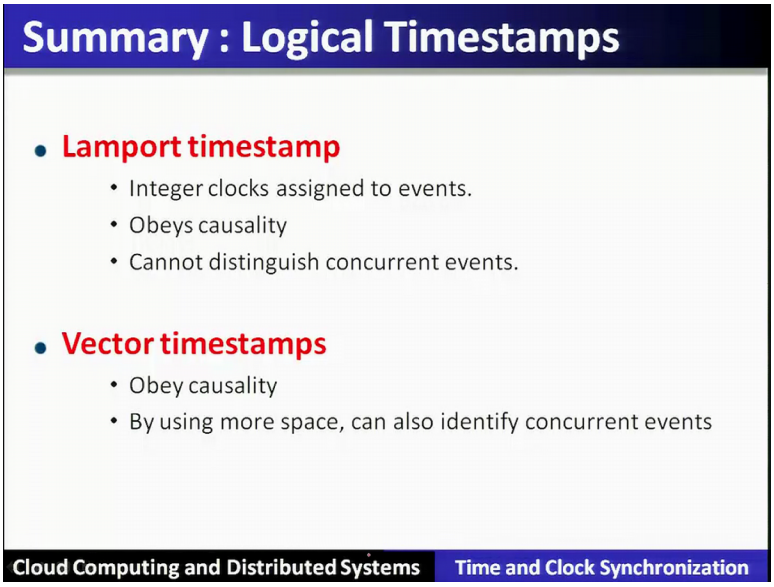
(Refer Slide Time: 78:14)



This also is fine, but one is greater than 0. Therefore, the same principle it is neither less than or equal to; so, that means it is not following obeying the causality relation. Hence they are concurrent events. Summary, we will see that lamport timestamp uses the

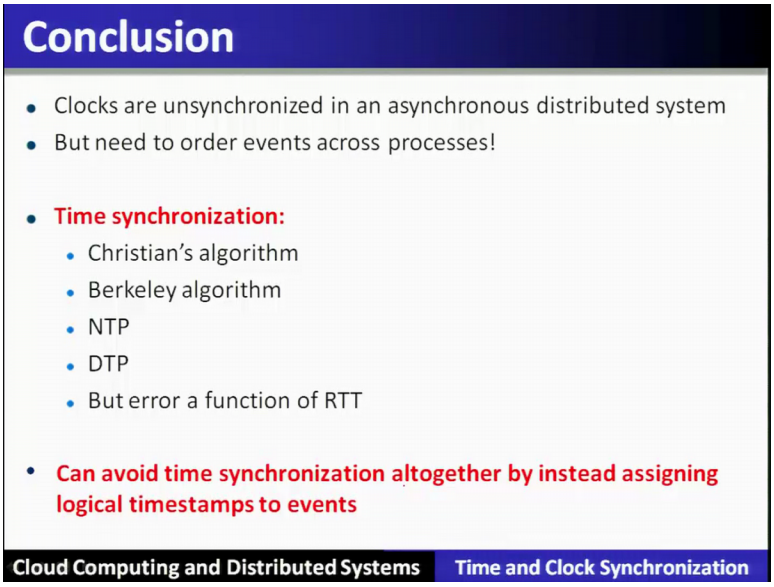
integer clocks to assign the event obeys the causality, but cannot distinguish the concurrent event.

(Refer Slide Time: 80:28)



Vector timestamps obeys the causality by using more space and also can identify the concurrent event more space in the sense. Here the vector is required to be sent in the message is an additional overhead of using the space.

(Refer Slide Time: 80:52)



So, conclusion clocks are unsynchronized or the synchronization of clocks in a asynchronous distributed system using physical clock synchronization method which we

have seen has or having the in accuracies. So, we have seen the methods for doing the clock synchronization in the distributed system. Let us say that they are Christian's algorithm Berkeley algorithm network time protocol DTP, but all these particular synchronization algorithms, they have the error which is the function of round trip time. So, we have seen also the method by avoiding the time synchronization and also ordering the events using the method such as lamport's clock vectors clock.

Thank you.