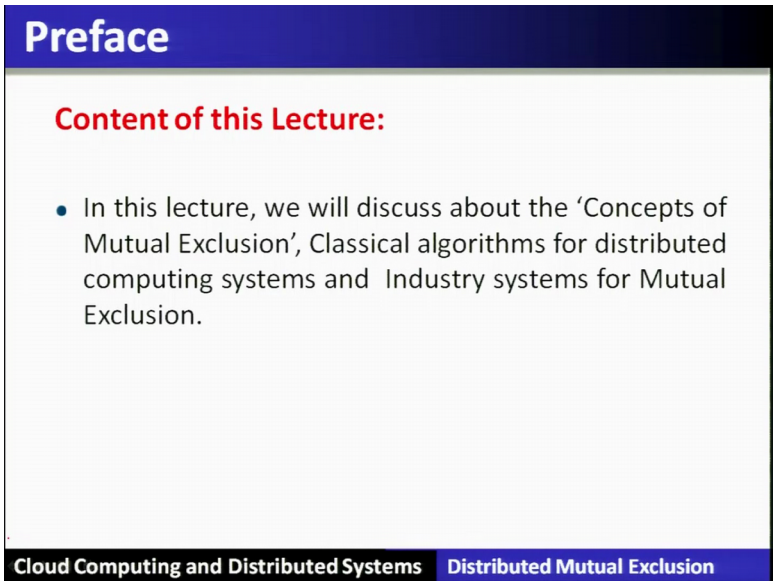
Cloud Computing and Distributed Systems
Dr. Rajiv Mishra
Department of Computer Science and Engineering
Indian Institute of Technology, Patna

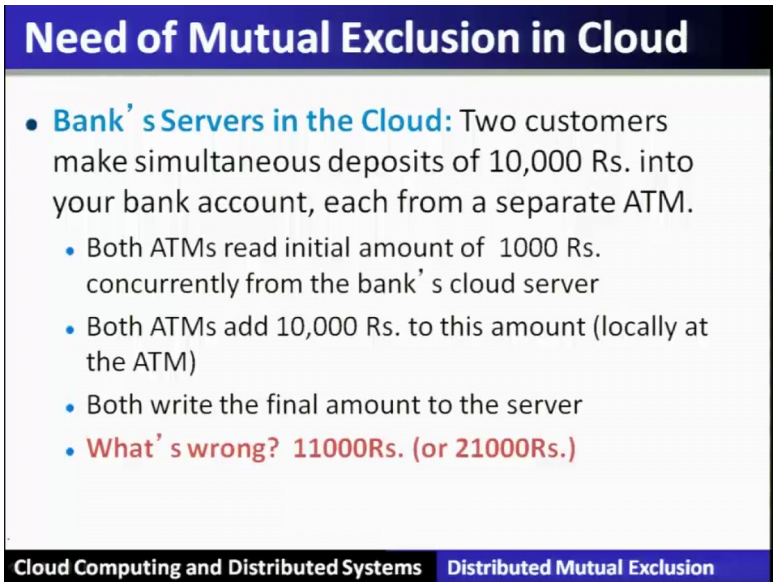
Lecture-12
Distributed Mutual Exclusion

(Refer Slide Time: 00:17)



Distributed Mutual Exclusion; Content of this lecture; in this lecture, we will discuss concepts of mutual exclusion used in the Cloud Computing Systems and also in the classical Distributed Systems and also we will see the industry systems which are using a different notion of mutual exclusion, the need of mutual exclusion in the cloud.

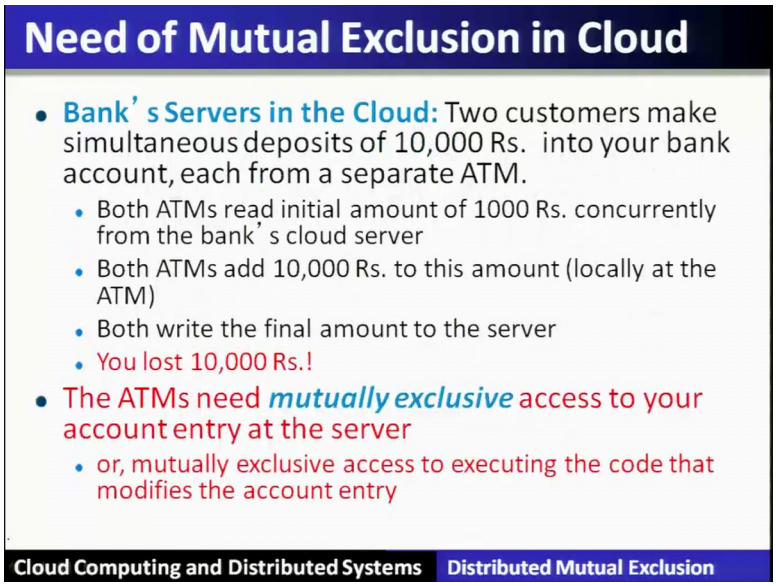
(Refer Slide Time: 00:40)



Let us see through an example that is the banks server in the cloud, let us consider 2 customers who makes simultaneous deposits of 10,000 rupees into your bank account each from the separate ATMs.

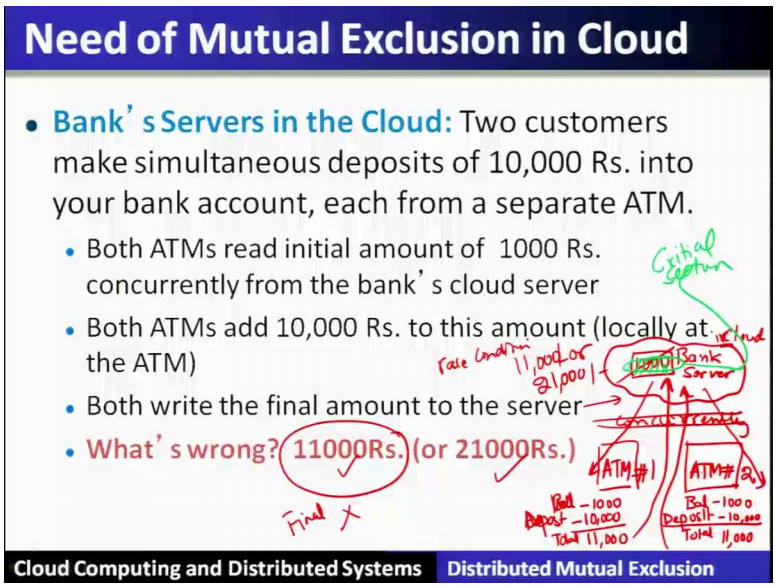
Both ATMs read initial values of amount, let us say 1000 concurrently from the banks cloud server. So, both the customers will now read the value 1000, second step; now both the ATM will add 10,000 to this amount locally at this ATM and then they both write the final amount to the server. Now what will be the final value which will be shown in your account? So, is it 11,000 or it is 21,000. So, what is wrong in this particular method?

(Refer Slide Time: 01:38)



Now, here we can say that if both write the final amount to the server, then one of them will be losing 10,000.

(Refer Slide Time: 01:52)



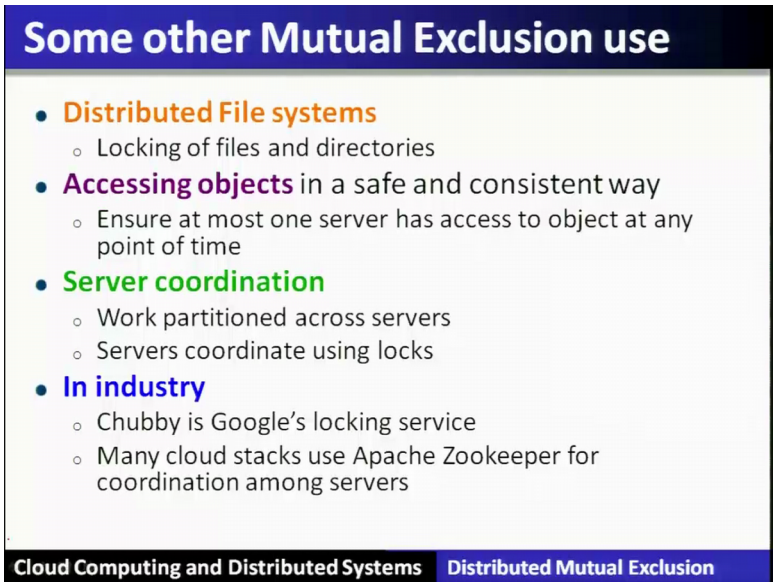
And this particular amount will be written as 11,000; that means, both will overwrite and the same amount will be shown and a loss of 10,000 will be let us see the illustration of this banks servers example. Now let us see that here this is the bank server and this is there in the cloud. Now this is an ATM 1 and this is an ATM number 2, it has initially 1000 in its account.

Now, both of them reads this value and they have the value balance as 1000, both will see the balance as 1000, they want to add or a deposit 10,000 to it, total becomes 11,000. Similarly, here it also want to deposit 10,000. So, total becomes 11,000, both they write these values to the server at concurrently, this is to be done concurrently what will happen over here both will be writing at the same point of time where the value will be 11,000 or 21,000 that is not known.

So, whether this is correct or this is correct. So, if the final value is comes out to be 11,000, then 10,000 will be lost. So, this condition is called a race condition and this can be solved with the help of accessing this particular section which is called a critical section which is nothing, but a code which allows the access to this particular value of your bank account that is the problem in this mode of operation. So, what is the issue what is the problem. So, the ATM need mutual exclusive access to your bank account entry which is located at the cloud so; that means, one at a time. So, one at a time means one customer will read your bank account then add 10,000 to it and then write back.

And once it is written back, then another; the next customer will read it and then again it will add some value and then write it back. So, in this manner, the final amount will be shown as 21,000, but that is possible if mutual exclusion that is the sequential access to your account is being provided from the cloud hence mutually exclusive access to executing the code that modifies or that accesses the bank account entry is going to be a crucial one.

(Refer Slide Time: 06:16)

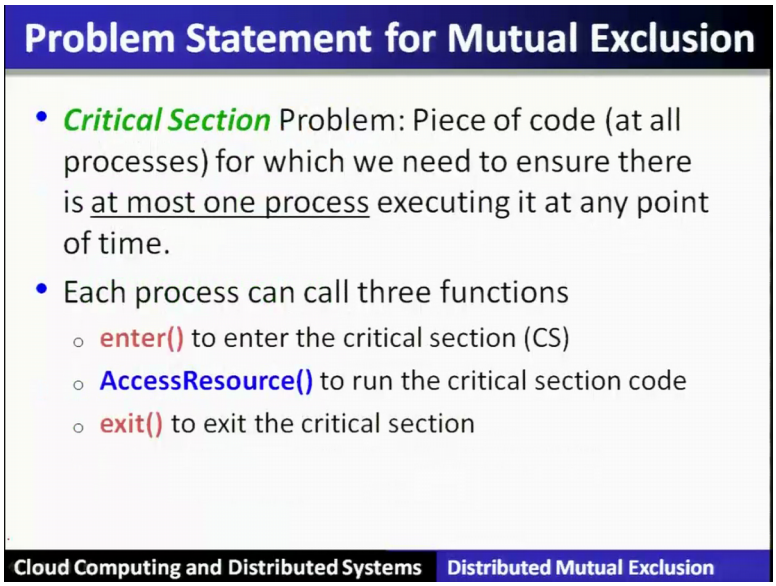


And here comes the role of a mutual exclusion, there are other mutual exclusion example such as distributed file system here, before for supporting the concurrent access on a file and directories they have to be locked.

So, again the concept of mutual exclusion will be implemented here in this scenario of a distributed file system; similarly accessing the object in a safe and consistent manner. So, that for a concurrent access at most one server has access to the object at any point of time is to be ensured through the mutual exclusion similarly for the server coordination. So, if several servers are there, they have to be locked before some operations are to be performed or some work is to be done and that is done through the locks and that is also a way of ensuring the mutual exclusion.

Now, in the industry the system such as chubby is a Google's locking system that is implemented on the cloud, similarly many cloud stacks such as uses the notion of apache zookeeper for coordination among the servers and this also ensures the mutual exclusion implementation from the industry perspective.

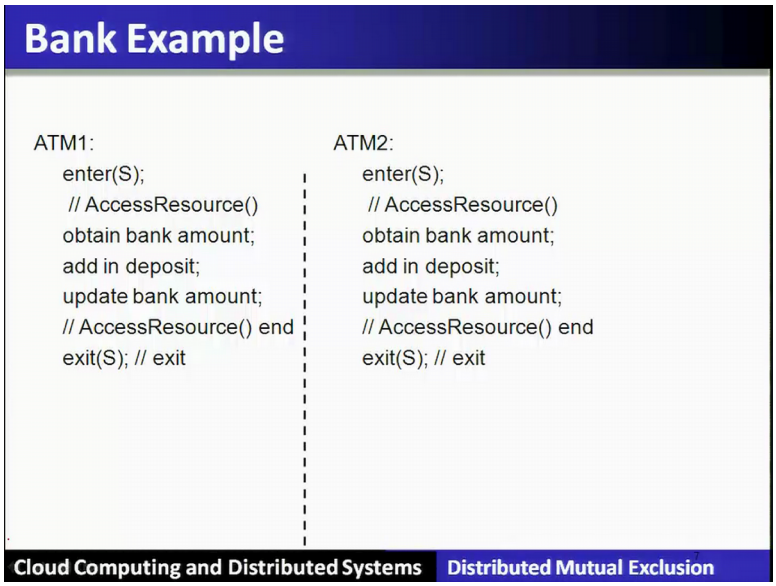
(Refer Slide Time: 07:43)



All these things we are going to see in this further section of this lecture. So, for the mutual exclusion, we have a piece of code which is going to be very crucial to ensure that it has to be accessed at most by one process at a time and this becomes a critical section problem.

To solve this particular problem or to pose this problem there are 3 functions defined for a critical section problem or a enter to enter the critical section then within it, you can access the resources of a critical section using routine called access resource and finally, when the critical section use is over then there is a routine which will exit the critical section.

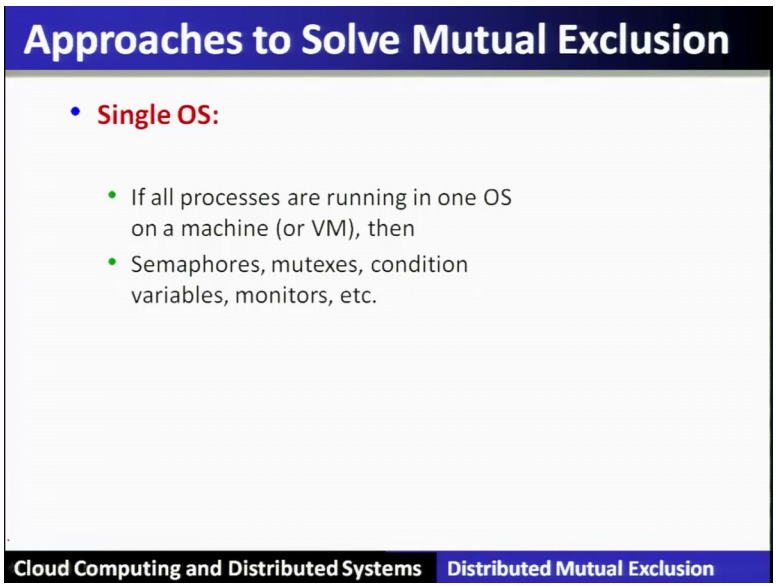
(Refer Slide Time: 08:37)



Let us see in the bank example, now two ATMs which basically are accessing your bank account first has to run this enter with S; S is some function of which will ensure the mutual exclusion, then it will ensure the access of your bank account using access resource.

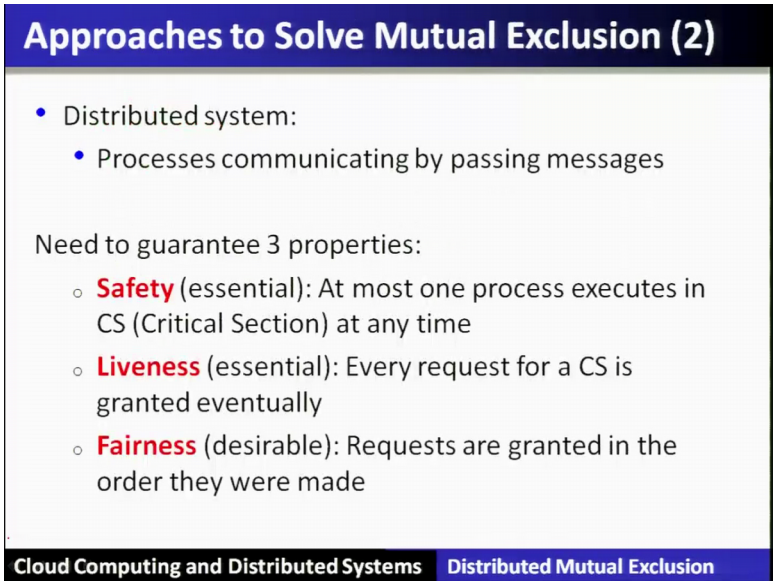
And finally, when the access of resource is finished, then it will end and exit that particular function call again both this ATMs will do this way. So, this access entry and access resource end exit, they will ensures the mutual exclusion one at a time.

(Refer Slide Time: 09:22)



This particular way of ensuring the critical section or a mutual exclusion through the critical section is normally done in a single operating system with the help of constructs like semaphore, mutexes, condition variables, monitors, etcetera.

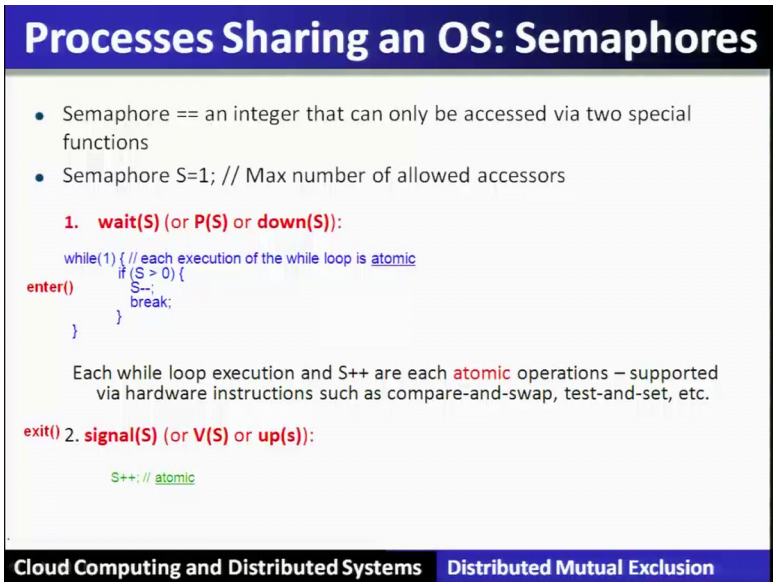
(Refer Slide Time: 09:41)



However in a distributed system such notions are not possible why because the distributed systems are primarily the message passing systems and so as the cloud systems.

So, but let us see in such systems, how we are going to ensure the mutual exclusion. So, such mutual exclusion requires to guarantee 3 different properties they are safety property this is very essential in the sense at most one process executes in the critical section at any time. Second property is called liveness which is also essential, this says that every request for a critical section is granted eventually and third one is called fairness that is also a desirable property which says that the requests are granted in the order in which the requests are made to the system.

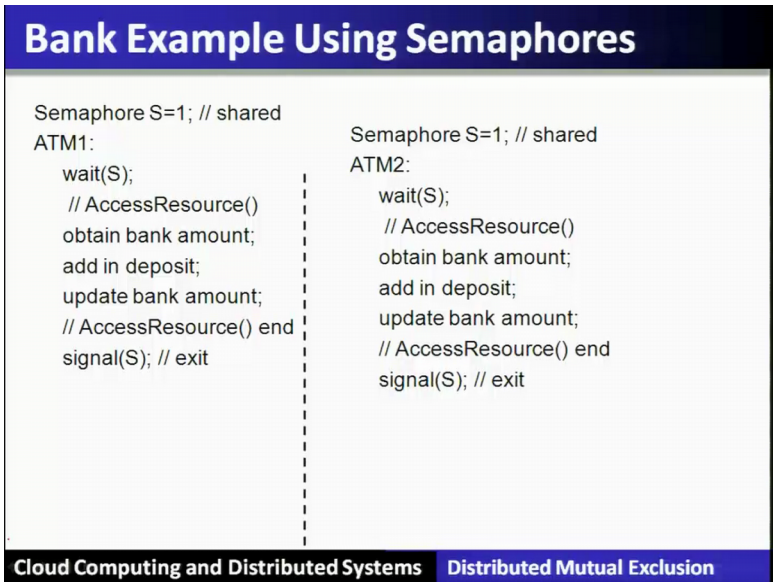
(Refer Slide Time: 10:46)



Now, when there is a single operating system and the processes sharing the operating system using the notion of a semaphore. So, semaphore is a shared variable that in a single operating system scenario is possible. So, semaphore is a integer that can only be accessed via two special functions which are called wait and signal sometimes, they are also called as a P and V. Let us see that only one process is allowed to enter to access the critical section. So, the value of S is equal to 1. So, whenever there is a entry part of the code then wait or a P of S is executed that is nothing, but a while loop, but it is an atomic function which will reduce the value of S.

And this is then going to be atomic operation that is once it is started it has to finish in whole it cannot be interrupted in between, similarly, there is a signal or a V function that is when this is to be executed when the exit part of the code is executed so that the other process can enter into the critical section.

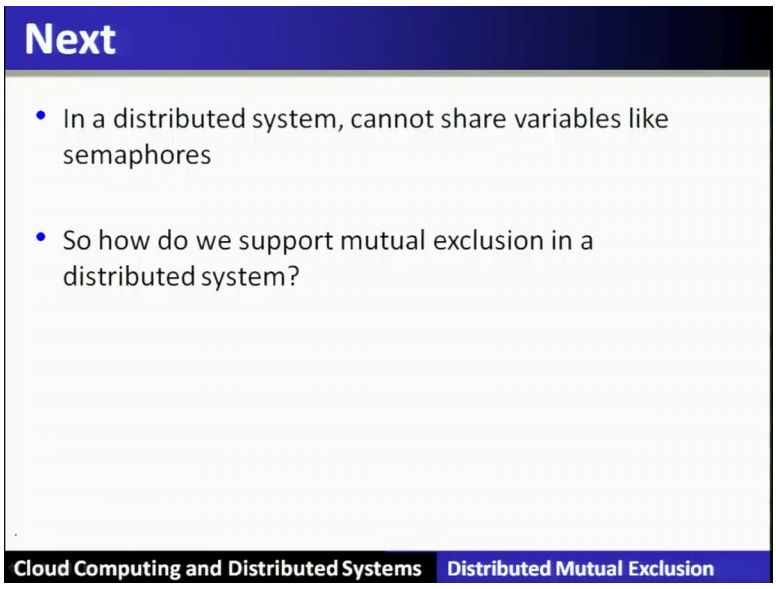
(Refer Slide Time: 12:07)



Using this semaphore, you can see the bank example when semaphore value variable as is initialized to 1. So, whenever there is a wait is executed, then it will basically decrement the value of S by 1 and enter into the critical section.

On the ATM 2, if at that point of time, it want to enter or it want to access the critical section then it will not be allowed why because the value of S is not 1 as per the conditions similarly, these particular signal will set the variable the value of S again to 1; so that the other process can execute into a critical section.

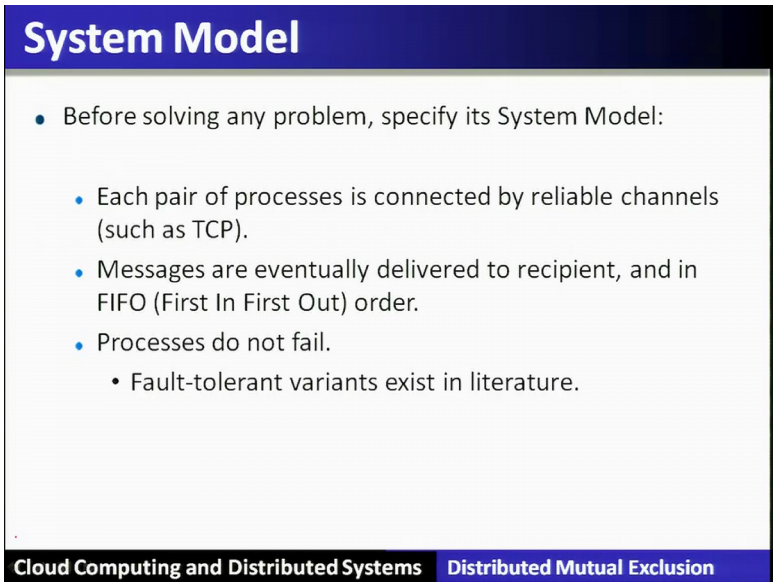
(Refer Slide Time: 13:01)



So far so good that in a single operating system such variables are supported by the hardware, but in distributed systems shared variables are not a pass ability, why because the only way to communicate is via the message passing, there is no shared memory concept in the distributed system.

So, we will see how mutual exclusion is to be supported in a distributed system and that too in a cloud model.

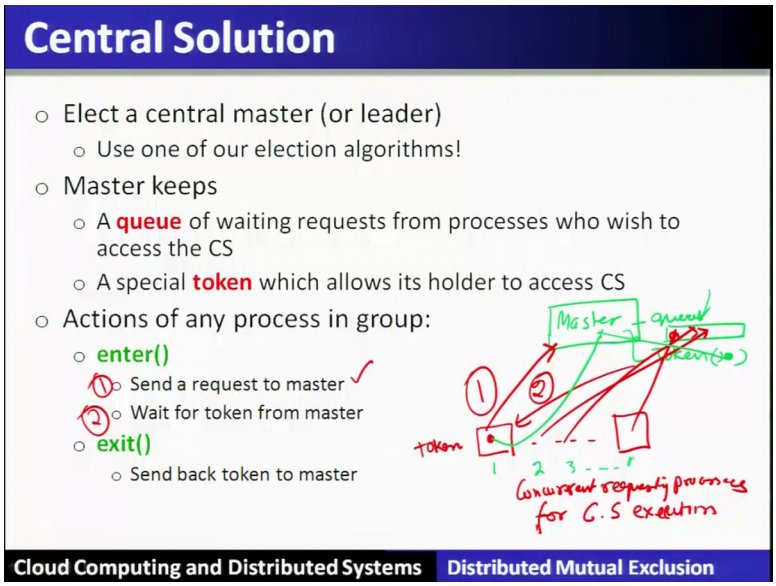
(Refer Slide Time: 13:26)



So, let us assume first the system model and let us see this particular way of implementing the mutual exclusion in this scenario that is the distributed systems. Now we assume that each pair of processes are connected by a reliable communication channel and the messages which are communicating they eventually are delivered to the recipients and also, they follow the principle, further we also assume that the processes do not fail.

So, fault tolerant variants also exist, but for the sake of simplicity, we will understand the concepts by assuming that the processes do not fail initially.

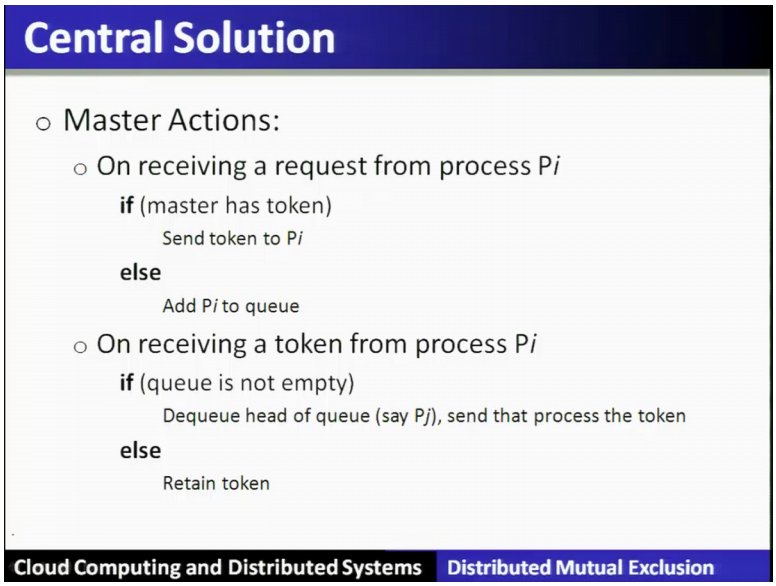
(Refer Slide Time: 14:12)



Let us see first the central solution and then we will go on a distributed model of this particular solution, let us assume there is a single master or a leader which can be elected by any leader election algorithm and that master will keep a queue of all waiting requests from the concurrent accesses which the processes are making to access the critical section now there is ex there exist a special token which allows its holder to access the critical section. So, the actions of any process in a group are primarily, it will execute a enter by enter, it means that it will send a request to the master and then it waits for its response.

From the master, once it gets the token from the master token allows that particular process whosever is having the token can enter into a critical section and once it is use is over, then it will execute the exit which is nothing, but sending the token back to the master let us say this particular string central solutions solves the mutual exclusion problem.

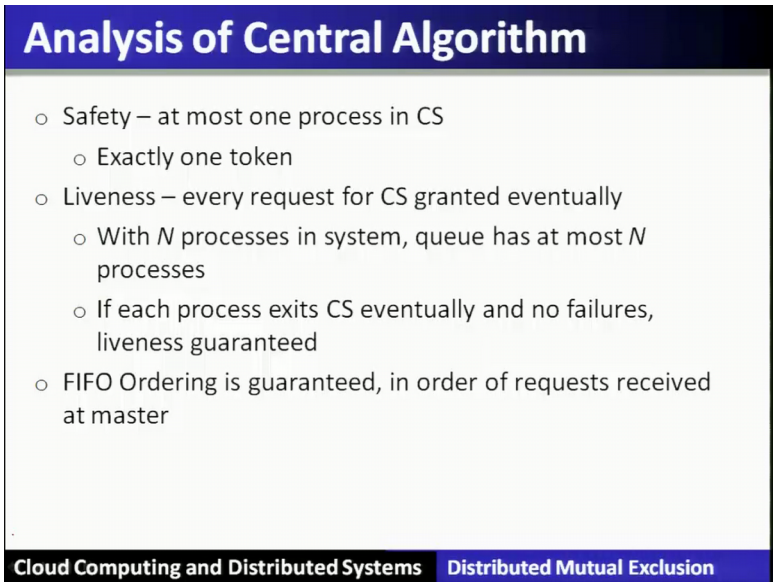
(Refer Slide Time: 15:25)



Here in this particular scenario let us see the illustration of this central solution here we will see that there is a master which maintains a queue and also, it maintains a token. So, if any of the requesting process concurrent to and for the critical section execution, they have to make the request, they will send the request, this is the message number one and wait for the token. So, the token will be given back by another message is message number 2.

Now, after having the token, it can enter into a critical section. So, since there is only one token which is given by the masters other processes they can join in the queue why because the token is already given. Now having done its job the token will be returned back, this token will be returned back to the master and the master will also have the token with it with him, similarly once the token is there then it will grand the token to the next waiting process and so on. So, token will be granted in this order 1, 2, 3 and so on up to let us say N.

(Refer Slide Time: 17:43)

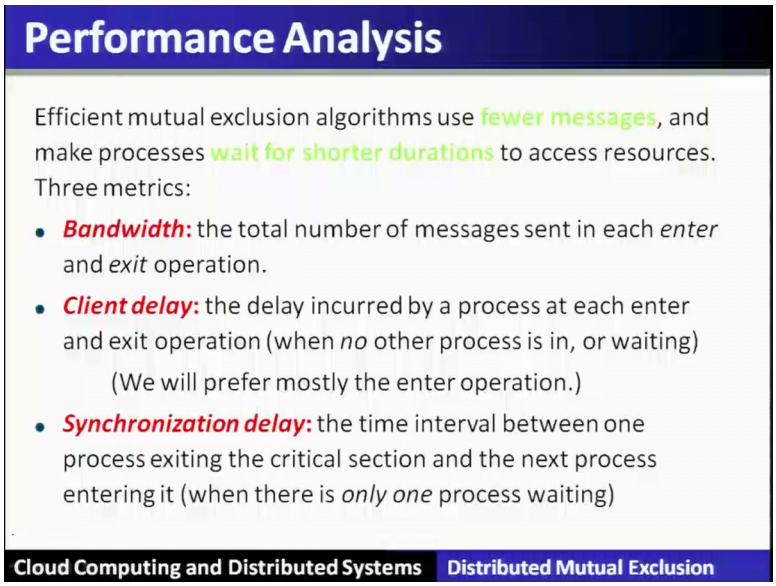


Let us see the analysis of this central algorithm, it ensures safety why because there is exactly one token and which is controlled by the master.

So, at a time only one process is given in this particular token and the mutual exclusion that is the safety property they ensured liveness property; that means, other processes who are concurrently making the request to enter in a critical section will eventually be granted why because as soon as the exit call is executed the token will be returned back of the process which is in the critical section. Since there are no failures so, the token will be given to the next waiting process by the master. So, master ensures that eventually all the process will be given the token in some order and that order is the FIFO order.

So, the master maintains a queue of all the waiting processes and ensures that the token is to be distributed in that particular order hence all 3 properties safety liveness and fairness are ensured here in the central algorithm.

(Refer Slide Time: 19:02)



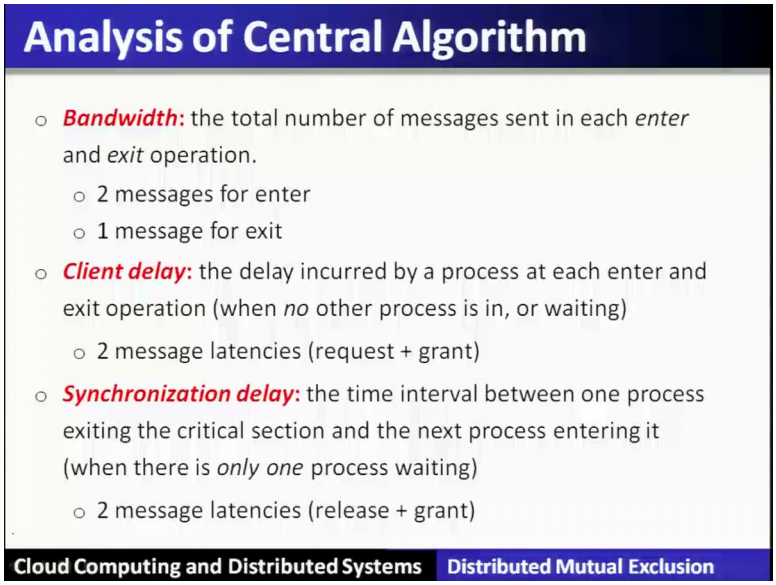
Let us analyze the performance of the central algorithm although it basically ensures the mutual exclusion, but to analyze its performance whether it is efficient or not we will check whether it is going to use the fewer messages and whether the waiting of the for the critical section is also very shorter duration.

So, to ensure these two properties fewer messages and a short delay to access the common resource by the concurrent request, there are 3 different metrics which will ensure these two properties to be analyzed, the first one called the bandwidth that is total number of messages sent in each enter and exit operation. So, that becomes; so, the bandwidth says that if the more number of messages are required for entry and exit then bandwidth is going to be more and this will be against the fewer messages requirement of an efficient mutual exclusion algorithm.

The second metric is called the client delay that is the delay incurred by a process at each entry and exit operations when no other process is on or waiting for the critical section. So, that makes the analysis whether it requires a short delay to access the critical section resources or not. Third one is called synchronization delay that is the time interval between one process exits the critical section and next process enters it this is also important why because. So, after the exit there are some message communication after which the next process will enter. So, what is that particular delay is also going to be

important notion contributing the delays? So, synchronization delay is also going to be important analysis.

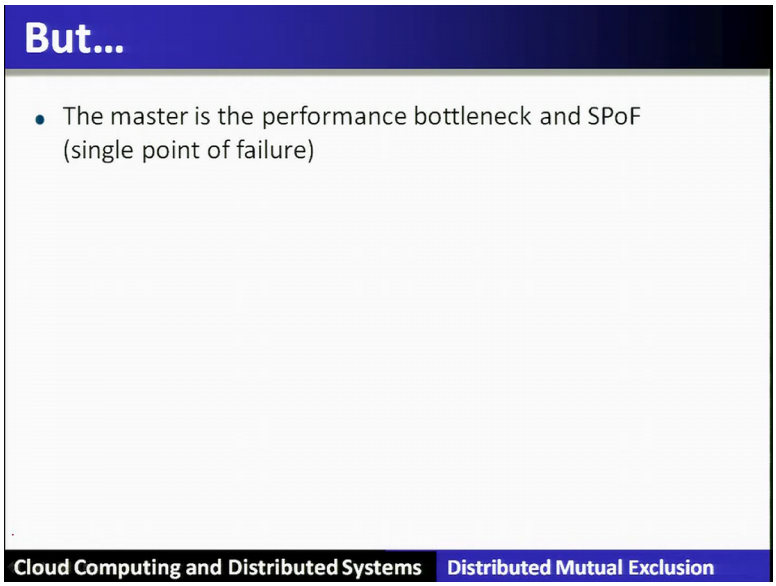
(Refer Slide Time: 21:20)



So, let us analyze based on these 3 metrics the central algorithm as you know the bandwidth, the total number of messages sent in each enter and exit is nothing, but two messages to enter, two messages means first the request is sent to the master and the master has to send back the token. So, 2 messages are required to enter and as far as the exit is concerned the one which is the process which is in the critical section when it exits it has to return back the token back to the master. So, 1 message is required for the master and the bandwidth is 3 messages.

Similarly, the client delay will incur 2 messages this delay is incurred by a process at each enter and exit operation. So, exit operation. So, that will be required 2 message delay when there are no when there are no process is in the critical section or waiting because message request has to go and the master has to grant similarly as far as the synchronization delay is concerned it also requires the 2 message latency why because exit will send the token back to the master and master will then allow the next one to go.

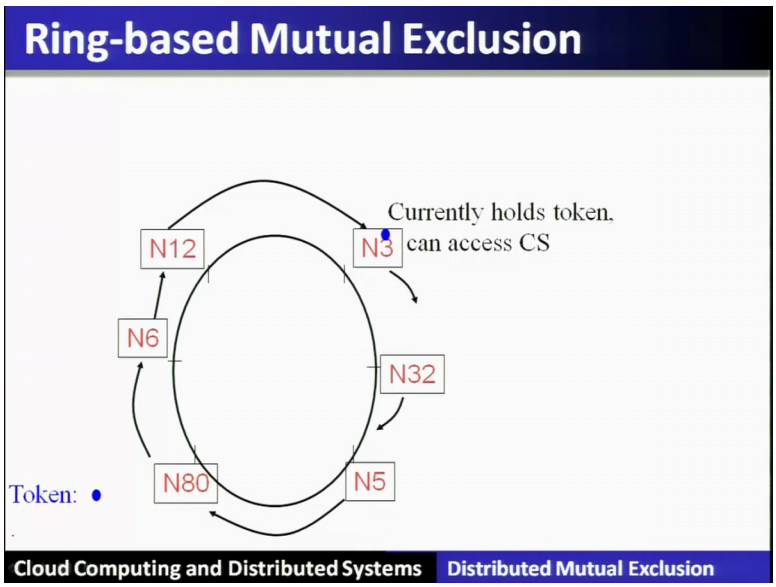
(Refer Slide Time: 22:41)



Using the master, the mutual exclusion problem we have seen going to be solved, but there is a performance bottleneck and also the single point of failure, what happens if the master fails.

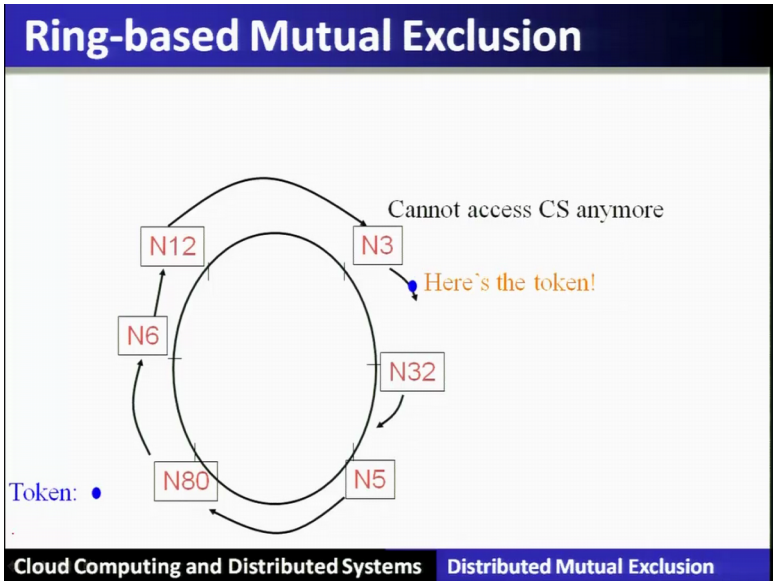
So, let us see some other distributed mutual exclusion algorithm which are going to solve this problem also and also the problem of a mutual exclusion.

(Refer Slide Time: 22:57)



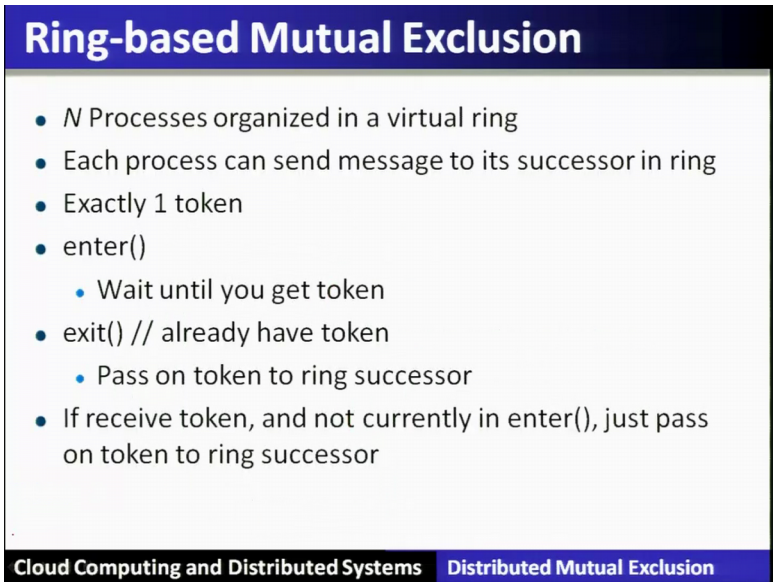
Let us consider a ring based mutual exclusion where logically all the nodes participating, they form a logical ring and there is one token which is circulating among those in a form of this overly ring structure.

(Refer Slide Time: 23:21)



Now, the nodes or the processes which are having the token they can enter into the critical section and the token will be circulated if the, if somebody requires it to enter into a critical section.

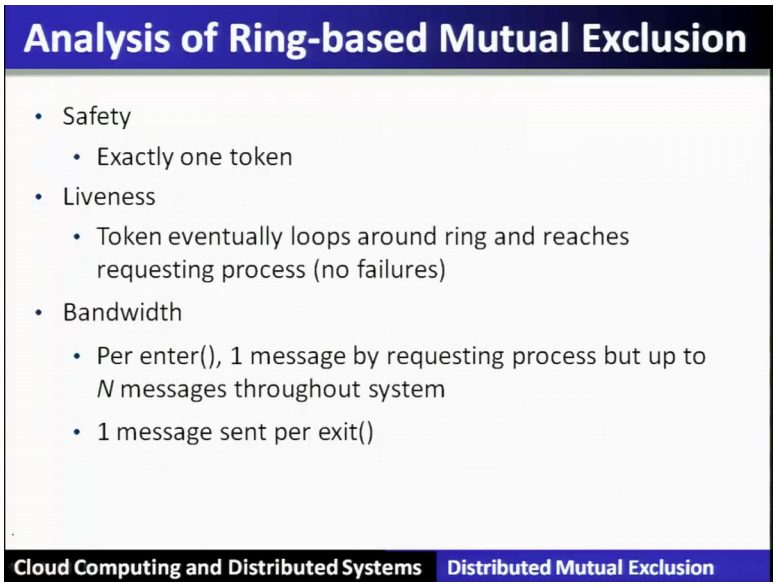
(Refer Slide Time: 23:39)



So, hence in this particular ring based mutual exclusion, let us assume that there are N processes which are organized in a virtual ring and each process can send messages to its successor in the ring exactly there is 1 token and enter function will wait until you get the token and exit is that if you already have a token, then I will pass on to the successor.

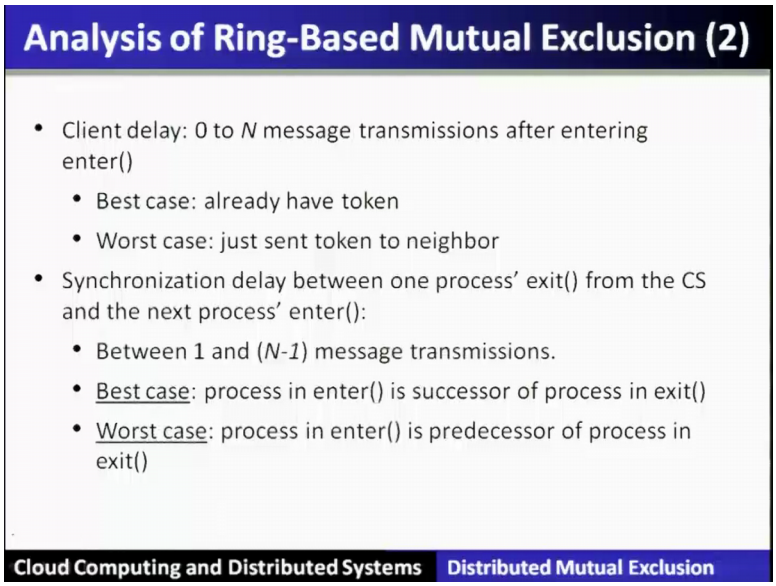
Now, if the receive token are not currently in the enter, then it will just pass on the token to the next successor.

(Refer Slide Time: 24:15)



In this particular algorithm that is a ring based algorithm if let us see the analysis safety is ensured why because there is only one token liveness is also ensured why because the token eventually loops around the ring and reaches to the requesting process by assuming that there are no failures bandwidth, if we analyze per enter 1 message by requesting the process up to the up to N different messages throughout system and 1 message send per each exit.

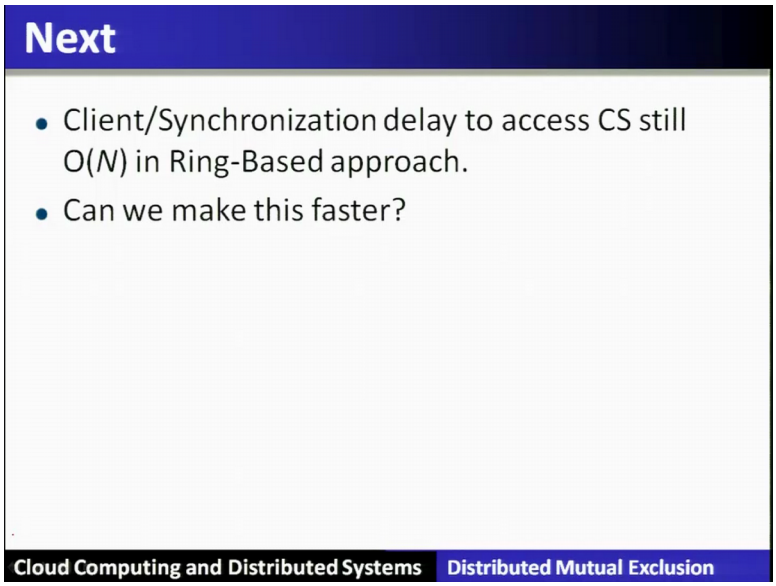
(Refer Slide Time: 24:47)



Client delay is basically ranging from zero to N message transmission of entering after entering the enter. So, best case is already have the token then it will go into the critical section without any delay worst case means it has just send the token to a neighbor, it has to pass through again and minus one and come back to them that is N different message transmissions are required. Similarly synchronization delay if we analyze between the process which has exited from a critical section and next process who want to enter in a who is allowed to enter in a critical section is between from 1 to N minus 1 messages transmission

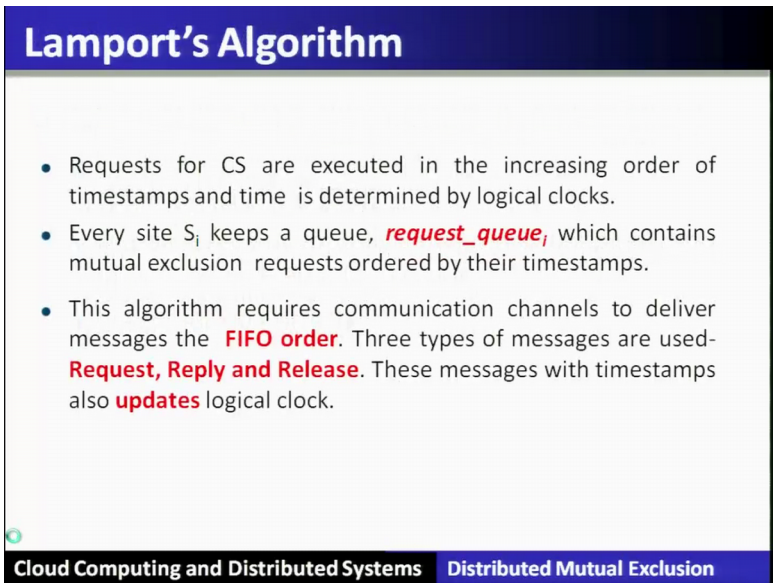
So, the best case when the process in the entry is successor of the process in the exit; so, that is the case only one message transmission is required the worst case basically is with the predecessor. So, it has to basically wait.

(Refer Slide Time: 25:45)



So, here we see that the synchronization delay is quite substantial that is of the order N. So, how we can make it a bit faster?

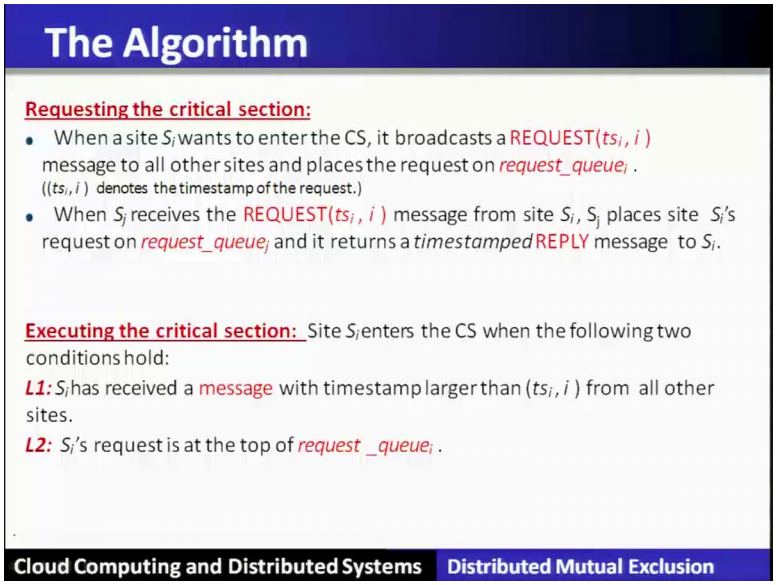
(Refer Slide Time: 25:55)



Now, let us see the algorithm which is given by the Leslie Lamport for the distributed mutual exclusion here the request for the critical section are executed in increasing order of the timestamp and the time is being maintained by a help of the logical clock, which is also called a Lamport's clock.

Now, here every site S i keeps a queue and this is called a request queue i which contains the mutual exclusion request which are ordered by their timestamps, this algorithm requires the communication channel to deliver the messages in FIFO order, there are 3 different type of messages which are used in this algorithm, they are request reply and release. These messages with timestamp also updates the logical clock.

(Refer Slide Time: 26:50)

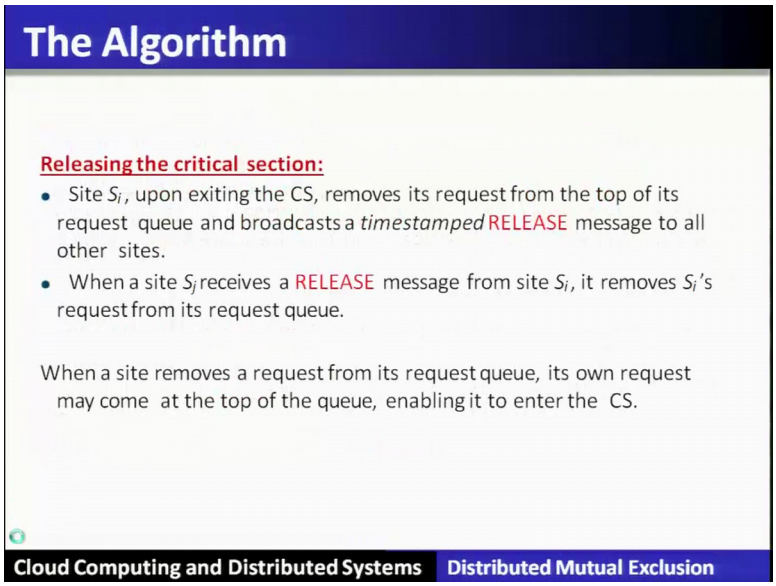


Let us see the algorithm called Lamport's algorithm, the first phase first step is called requesting the critical section. So, when a site S i want to enter in a critical section, it broadcasts a request message with a timestamp and the id of a process i.

This particular message is broadcasts to all other sites and also it places this request in its request queue when S j another process receives this request from S i, then it places it in its request queue and returns a reply with the timestamp to S i back. The second step is the process can execute in a critical section if the following two conditions holds, the first one is called L 1 condition which says that S i has received the message with the timestamp larger than its own timestamp from all other sites; that means, his timestamp is the lowest that is it is the highest priority process which is on the top of the queue.

L 2 property says that sis request is top of the request queue why because all other timestamps are higher than that so; obviously, it will be on the top of the queue.

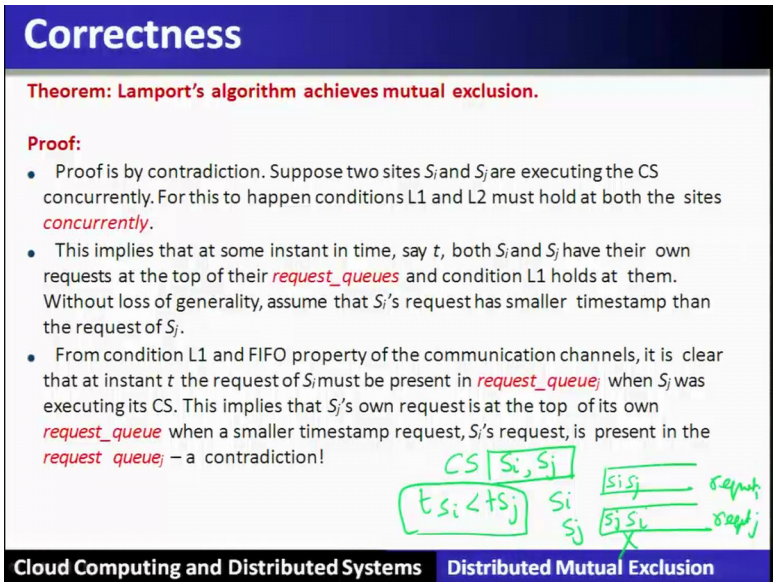
(Refer Slide Time: 28:21)



Third step is called releasing the critical section. So, a site S i upon exiting the critical section removes its request from the top of its request queue and broadcasts a timestamp release message to all of the sites when a site S j receives a release message from site S i, it removes sis request from the from its request queue.

So, when a site removes a request from its request queue its own request may come at the top of the queue enabling it to enter into the critical section meaning in the sense that the release message will allow to change the q; that means, the one which has already exited will be removed from the request queue of all other sites including that site itself allowing the other waiting process the next waiting process can enter into critical section.

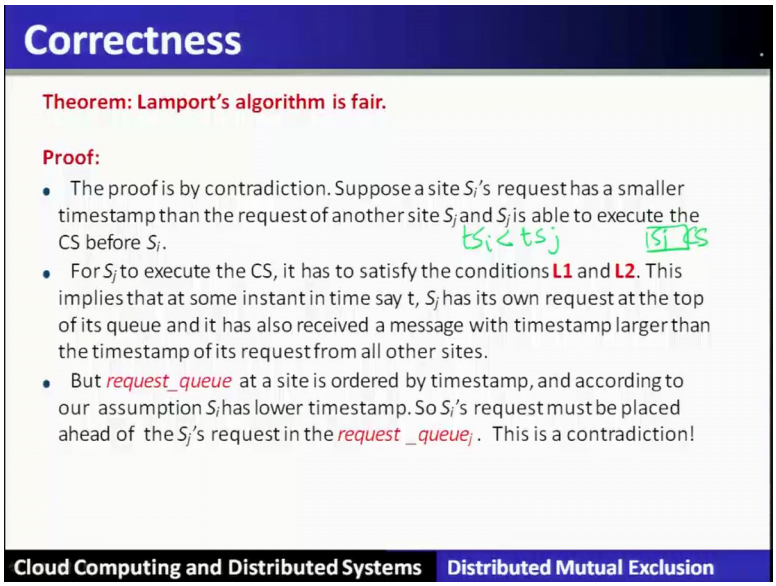
(Refer Slide Time: 29:24)



So, correctness Lamport's algorithm achieves mutual exclusion let us see the proof of this theorem, proof is by contradiction suppose two sites S i and s j are executing into the critical section that is possible when the two conditions L 1 and L 2 must hold for both the sites at the same point of time this implies that at some instant of time t both S i and S j have their own request at their top of the request queues and the condition L 1 holds at them without loss of generality assume that sis request has a smaller timestamp than S j.

Now, if S i's request has the smaller timestamp than S j then from condition L 1 and FIFO property of the channel it is clear that at instant t the request of S i must be present in request queue of i when S j was when S j was executing into its critical section. This implies that sis own request queue is at the top of its own and S j's request is at the top of its own queue. So, which is not possible why because the timestamp of S i is smaller than timestamp of S j. So, it cannot be like this both the request S i and S j they are they are present in their request queues, but they have to be in the same order why because the timestamp of i is less than timestamp of j, hence both of them entering into the critical section is a contradiction.

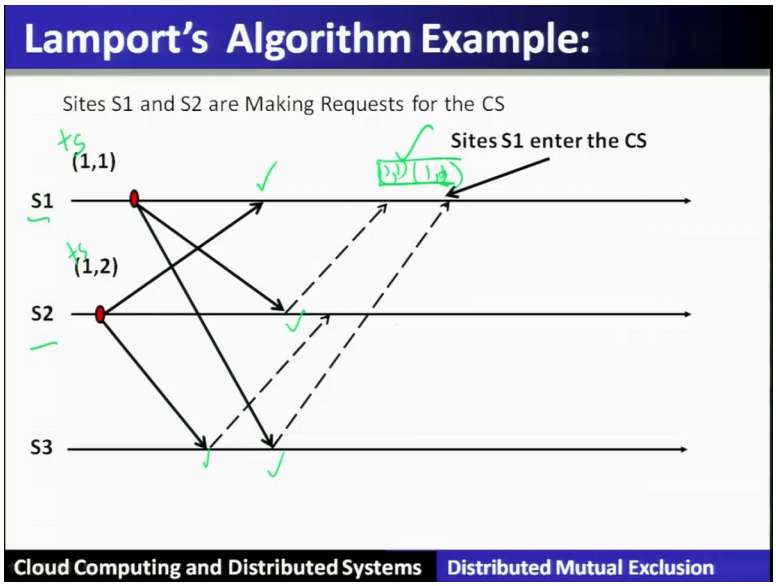
(Refer Slide Time: 32:06)



Now, another correctness criteria which says that the Lamport's algorithm achieves fairness proof of this is also given to be given by the contradiction suppose S i's request has a smaller timestamp then the request of another. Then the request another site S j and S i is able to execute the critical section before S j for S j to execute the critical section for S j to execute the critical section it has to satisfy the condition L 1 and L 2 this implies that at some instant in time say t S j has its own request at the top of its queue and it has also received a message with the timestamp larger than the timestamp of its request from all other sites

But request queue at site is ordered by the timestamp according to our assumption S i has the low timestamp. So, S i's request must be placed ahead of S j S j s request in a request queue which is a contradiction hence, it achieves a fairness in the sense it follows the FIFO property or the properties that the one which is having the lower timestamp has higher priority.

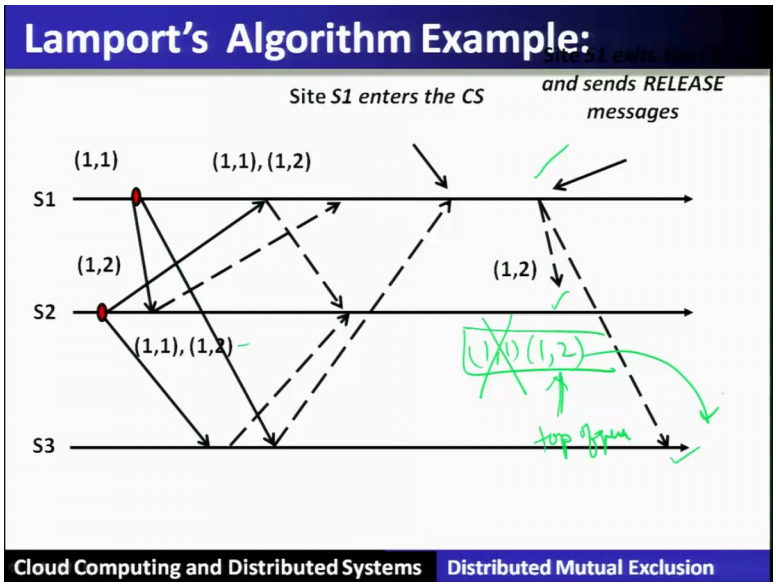
(Refer Slide Time: 33:47)



So, it can be; so, let us see the example which will illustrate this Lamport's algorithm, let us assume that simultaneously two sites S 1 and S 2, they want to enter in a critical section and so, they have requested and they have broadcast their request. So, S 2's request will basically be received at S 1 and S 3 and S 1's request will be received at S 2 and S 3. So, they also send their timestamp this is a timestamp and its number.

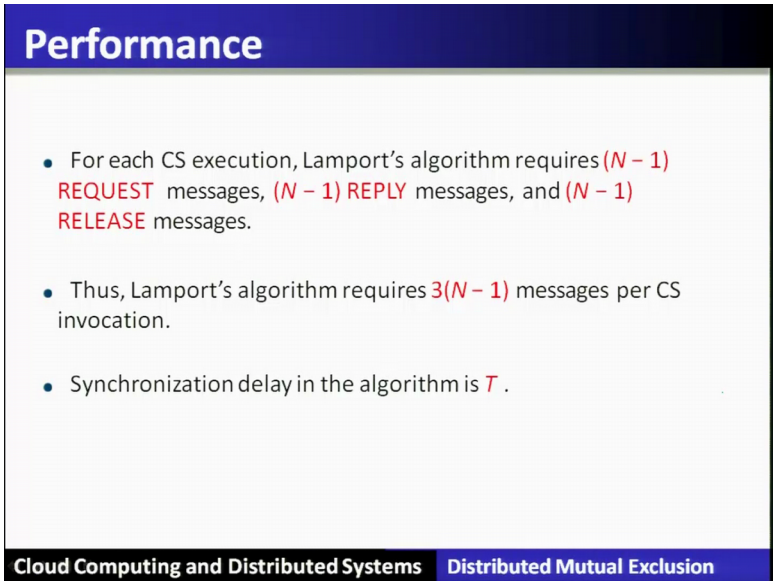
Now, as far as the second step is concerned which says that they have to reply back. So, S 2 will reply to S 1 and S 3 will also reply and when S 1 will receive the all the replies, then it will see that his request 1 comma 1 is at the top of the queue. Similarly, S 2 after receiving all the replies.

(Refer Slide Time: 35:03)



So, what will happen is now S 1 will enter into a critical section. So, after it exists the critical section it will send a release message and this release message will reach to all of them including S 2. So, once S 2 will receive the release message it will remove it from its request queue so that his request will be at the top of the queue, now it will go in a critical section that will be shown in the next slide. So, now S 2 will enter into a critical section.

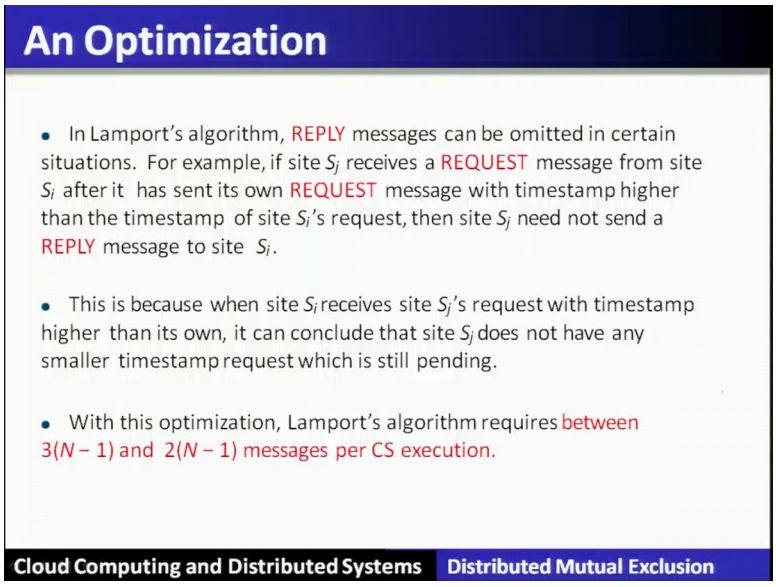
(Refer Slide Time: 35:50)



Let us see the performance of this Lamport's algorithm. So, it requires N minus 1 different messages in the form of request message N minus one different messages it has to send in the form of reply and N minus 1 in the form of release message total bandwidth of this algorithm is 3 N minus 1 for per critical section in invocation.

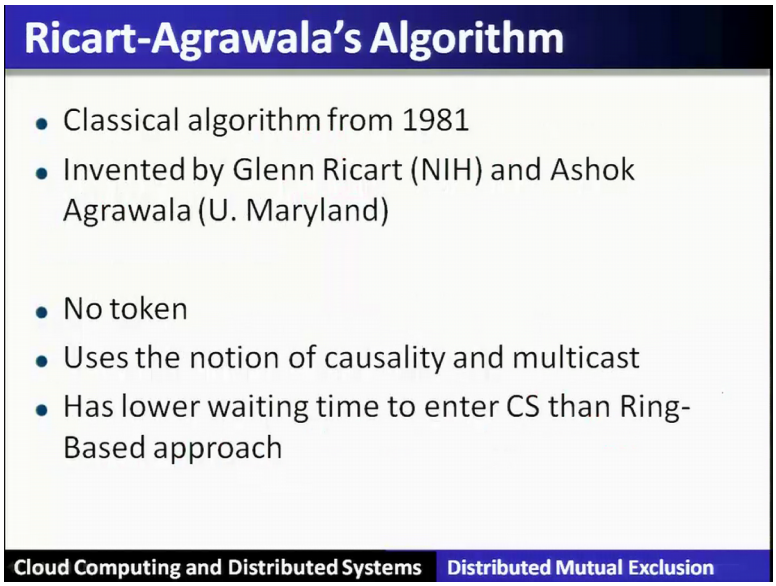
Synchronization delay of this algorithm is quite obvious that is T why because after the release message the one which is at the toper, it can go into critical section. So, it is only T as soon as the release message is received that is nothing, but a T delay.

(Refer Slide Time: 36:40)



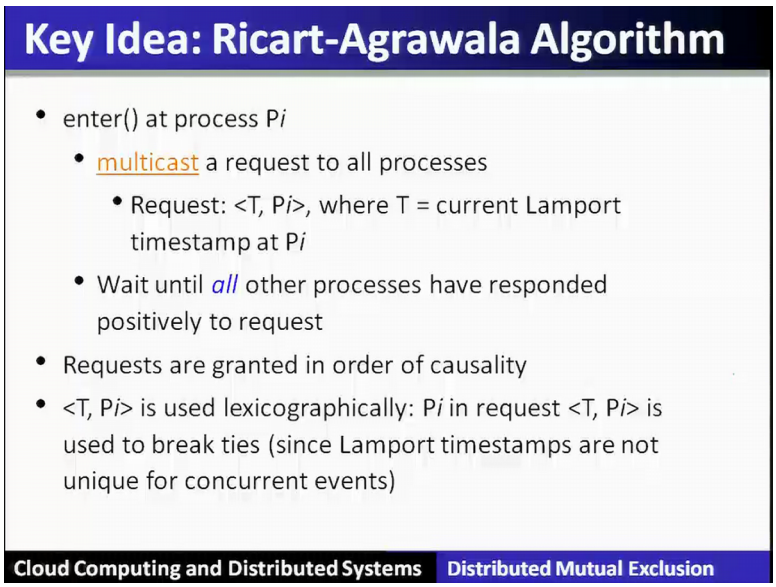
There is a possibility of optimization of this algorithm Lamport so; that means, optimization says that the reply can also play the role of the release as well. So, release message you can be suppressed. So, with this optimization Lamport's algorithm requires between 3 N minus 1 and 2 N minus 1 messages per critical section invocation.

(Refer Slide Time: 37:11)



The next algorithm is an improvement of the Lamport's algorithm which is called Ricart Agrawala's algorithm. So, this is a classical algorithm for distributed mutual exclusion given in 1981 by two famous persons one is Ricart, which was in NIH and the other one is Agrawala which was in Maryland university. Assumes no token it also assumes the notion of causality with the help of a Lamport's logical clock and also uses the multicast.

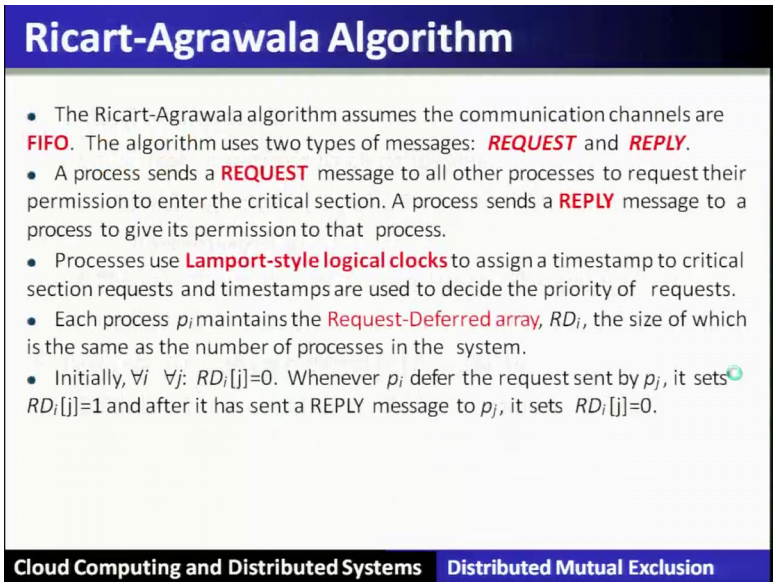
(Refer Slide Time: 37:59)



Now, the key idea of Ricart Agrawala algorithm says that when it runs and it enter to a critical section; that means, it wants to go in a critical section let us say a process P i it

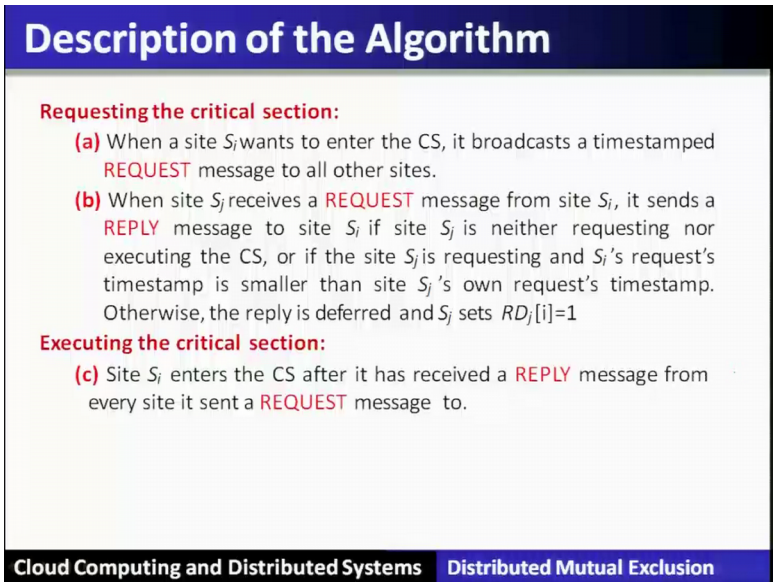
will send a multicast to all other processes in the form of a request and that request is with the timestamp for causality and it will wait for the responses which are positive from all other sites. So, requests are granted in the order of causality so; that means, the one which is having the lowest timestamp is having the highest priority.

(Refer Slide Time: 38:49)



So, Ricart Agrawala algorithm uses request and reply and also it uses the Lamport logical clock and also it uses an array which is called a request deferred array RD.

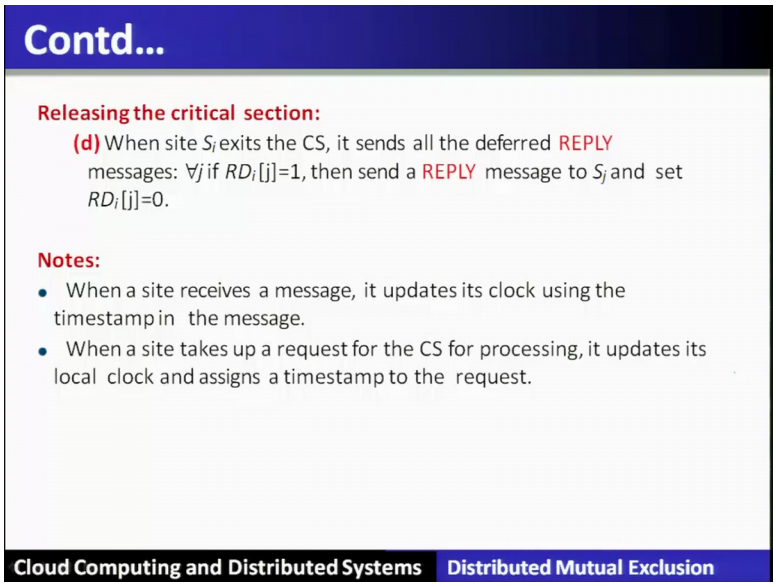
(Refer Slide Time: 39:09)



Let us see the algorithm it has again 3 different steps, the first step is requesting the critical section. So, when a site S i wants to enter a critical section it broadcast timestamp request message to all the sites now, when a site S j receives a request message from site S i it sends a reply message to S i if site S j is neither requesting nor executing critical section or if the site S j is requesting and sis request is request timestamp is smaller than S j's own request timestamp. Then you send a reply, otherwise reply is deferred and S j sets the in its deferred request j setting the i is equal to one why because it has not replied to i.

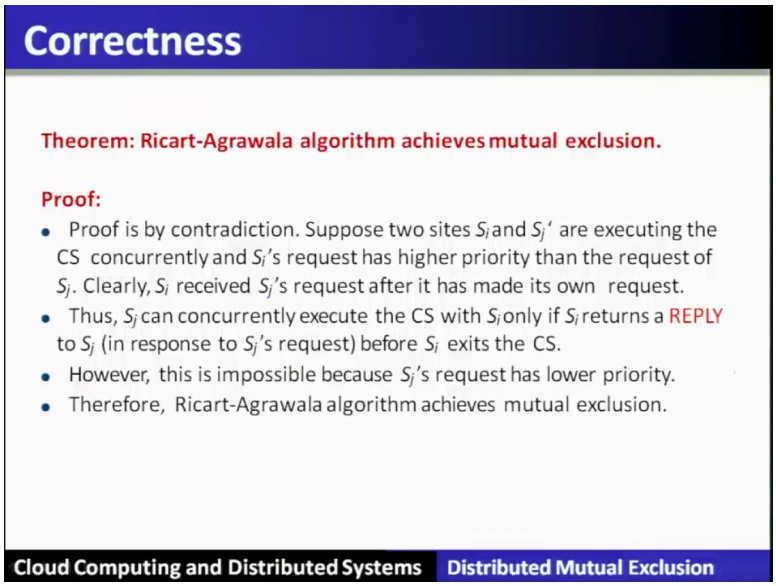
Executing critical section; that means, the site S i enters critical section after it has received the reply from every site it has sent a request message to releasing the critical section when site S i exists the critical section.

(Refer Slide Time: 40:20)



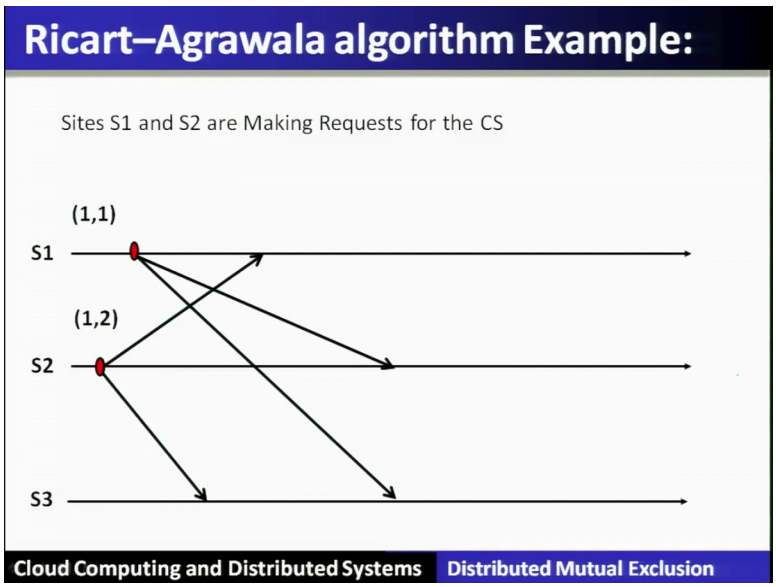
It sends all the deferred replies messages which are maintained here in request deferred array by sending the reply messages to S i and set the request deferred to null. So, by this way after receiving the all the replies the other waiting processes can enter a critical section.

(Refer Slide Time: 40:53)



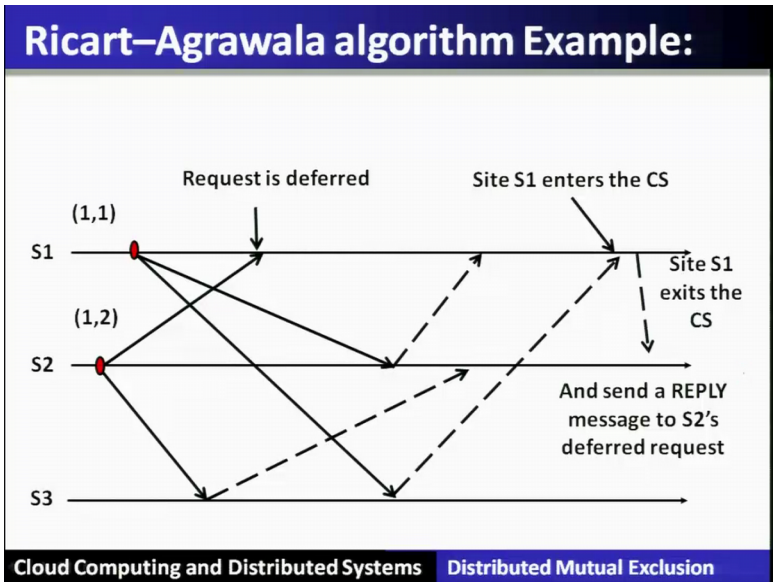
This Ricart Agrawala algorithm achieves mutual exclusion again, we can prove by contradiction which is quite obvious.

(Refer Slide Time: 41:07)



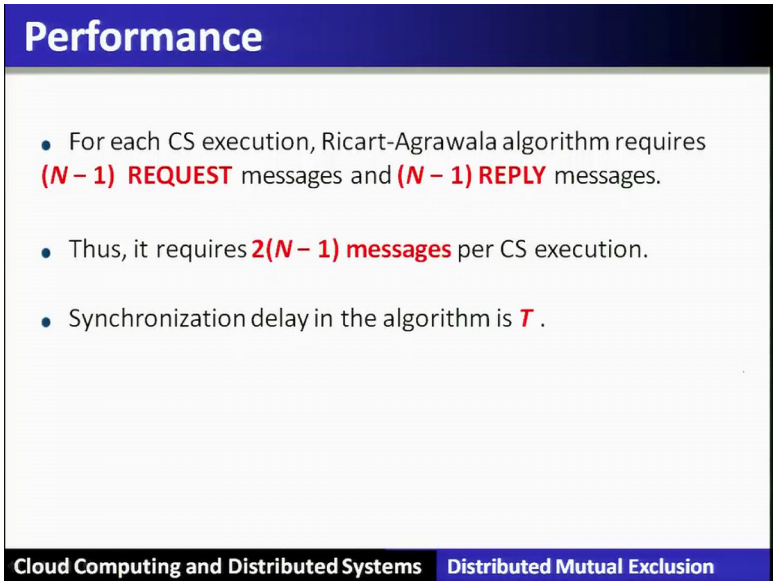
Let us see an example to understand the working of Ricart Agrawala algorithm let us see the 3 different sites out of them S 1 and S 2 concurrently making the request to enter in a critical section with their timestamps shown over here S 1.

(Refer Slide Time: 41:20)



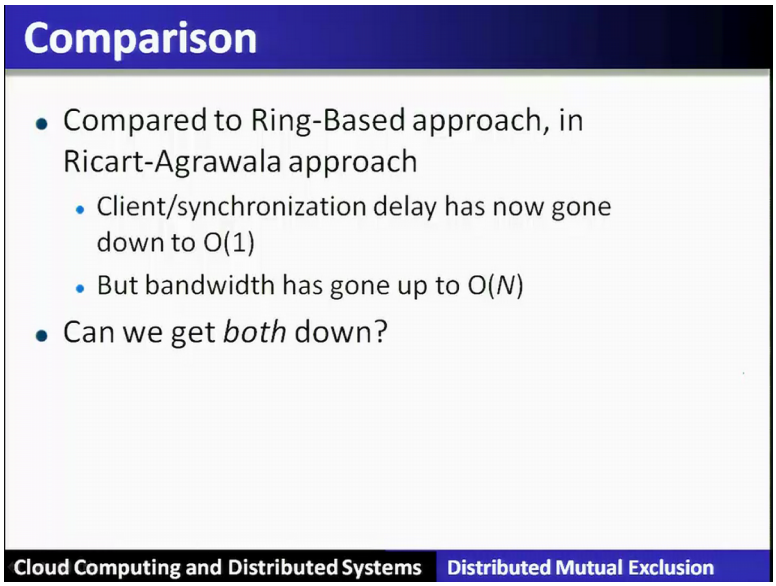
Now, as far as S 2 when it sends a request to S 1 S 1 timestamp, if you compare is lower. So, his request will be deferred. So, it S 1 will not send the reply back to us to. So, S 1 will enter into a critical section S 1 will send a S 1 will enter into a critical section and after exit, it will send the reply back to S 2. So, the S 2 can also enter into critical section.

(Refer Slide Time: 41:54)



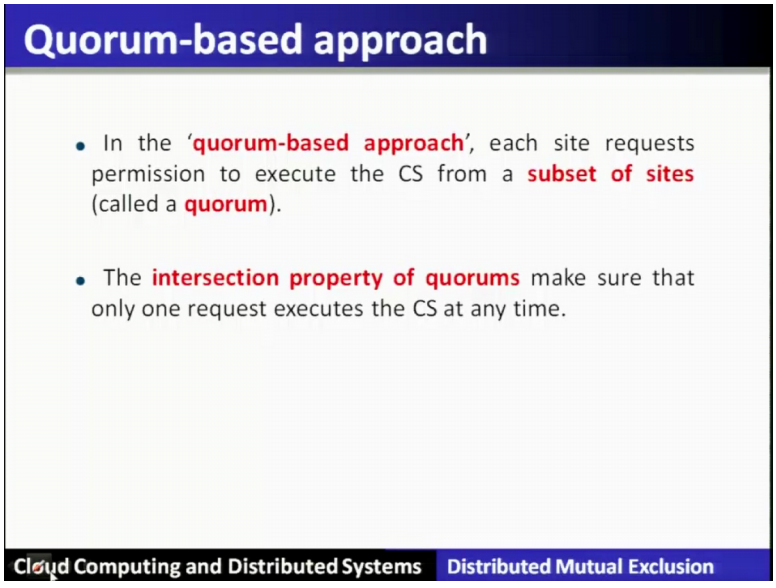
Performance of Ricart Agrawala algorithm is 2 N minus 1 messages per critical section invocation synchronization delay is same as Lamport algorithm that is T.

(Refer Slide Time: 42:12)



Compared to Ricart compared to ring based algorithm, Ricart Agrawala algorithm approach you can see that here the client server delay is you reduce that it is T means of the order one, but the bandwidth is of the order N. So, how the bandwidth can further be reduced quorum based approach.

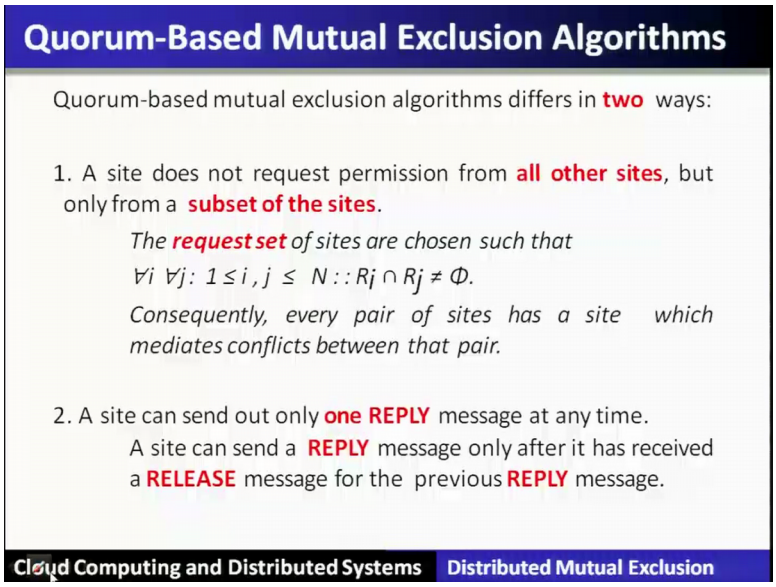
(Refer Slide Time: 42:37)



In the quorum based approach, each site requests permission to execute the critical section from a subset of sites which is called a quorum. This particular quorum or a subset of sites has to satisfy some properties for example, the property is called as

intersection properties of quorums. That make sures that only one request executes in the critical section at any point of time and not all sites are required to take permissions from Quorum based mutual exclusion algorithms differs from the previous previously discussed algorithms in two different ways.

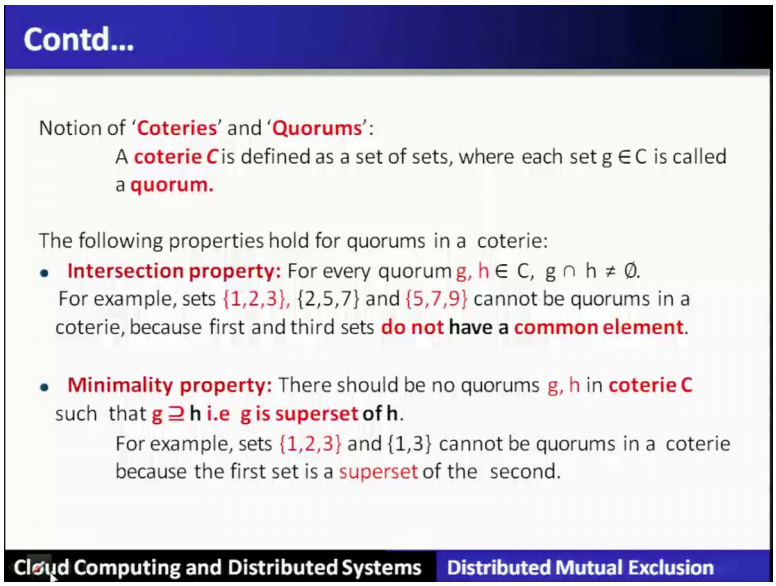
(Refer Slide Time: 43:11)



The first it differs that the site does not request permission from all other sites, but only from a subset of sites which is called a quorum. So, the request set of the sites are chosen to follow the intersection properties that is the request set of let us say a site i and j, if we take the intersection it will be a non null why because there must exist a site between these two request set which mediates the conflict between these two pairs.

The other way this particular quorum based algorithm differs from the previously discussed algorithm is that the site can send out only one reply message at any point of time; that means, a site can send a reply message only after it has received a release message from the previous reply messages. Unlike in the previous discussed algorithm where a particular site has to give has to reply to all other sites, here only one reply is required. So, this is an optimized algorithm compared to the other.

(Refer Slide Time: 44:33)

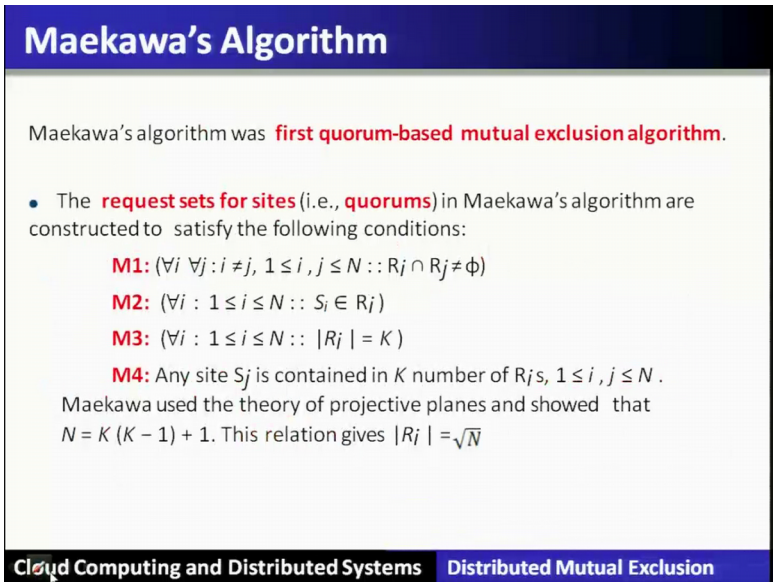


In this algorithm, there is a notion of coteries and the quorums a coterie is defined as a set of sets where each set g which is an element which is in coterie c is called a quorum. The following properties hold for the quorums in a coterie the first one is called intersection property for every quorum let us say g and h which is there in coterie the intersection of these two quorums is a not null.

For example that sets 1 2 3, 2 5 7 and 5 7 9; 3 different sets shown here in this example cannot be the quorums of a coterie why because if you take the intersection of the first set that is 1 2 3 and the intersection of the third one that is 5 7 9, it does not have a not null; that means, it is null so; that means, it does not have a common member hence intersection properties not satisfied therefore, these 3 different sets shown in the example are not the coteries or they are not the quorums.

The second property which quorums in a coterie follows is called minimality property which says that there should be no quorum say for example, g and h in coterie c such that quorum g is a superset of quorum h in a coterie c that is g is a superset of h; that means, it is violating the minimality property, if it is a superset of another quorum. So, for example, the set 1, 2, 3 and 1, 3 they cannot be quorum why because 1, 2, 3 is a superset of 1 3 and therefore, minimality property is violated.

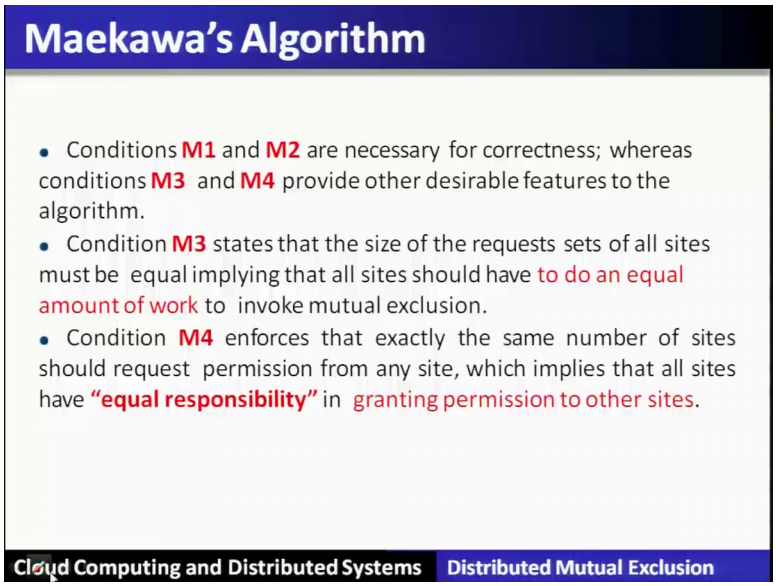
(Refer Slide Time: 46:38)



Using the concept of quorums Maekawa has given the first algorithm which is called Maekawas algorithm for mutual exclusion that is the first quorum based mutual exclusion algorithm. In this algorithm the request sets for the sites which is also nothing, but the quorum are constructed to satisfy the following 4 properties, they are which is designated as M 1 which says that critical section property; that means, for any two request sets if we take the intersection it is not null the another property M 2 which says that any site is in the request set. So, that is all the sites are participating in some or the other request sets.

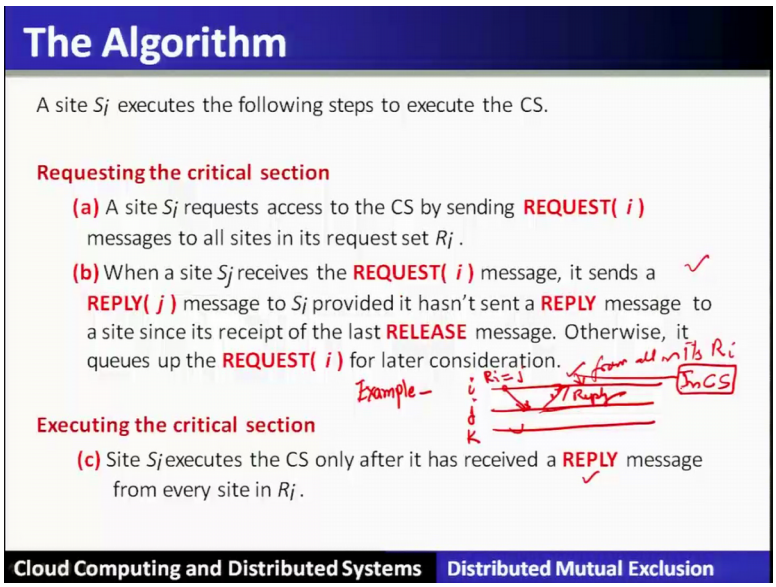
M 3 property says that the cardinality of a request set is k and the value of k is computed as root of N, that is the request set that is the size of quorum is not n, but root N that is the subset of the sites M 4 the fourth property says that any site S j is contained in K number of R js so; that means, the fourth property says that any site is contained in k number of R js; that means, there is a property of a load balancing; that means, every site has to do equal amount of work here in Maekawa's algorithm.

(Refer Slide Time: 48:17)



So, the property M 1 and M 2 are the necessary for the correctness whereas, M 3 and M 4 they provide other desirable properties of the algorithm the property M 3 states that the size of the request set of all the sites must be equal implying that all site should have to do an equal amount of work to invoke the mutual exclusion. Whereas, the condition enforces that exactly the same number of sites should basically request permission from any site which implies that all sites have equal responsibility in granting the permission to the other sites.

(Refer Slide Time: 48:51)



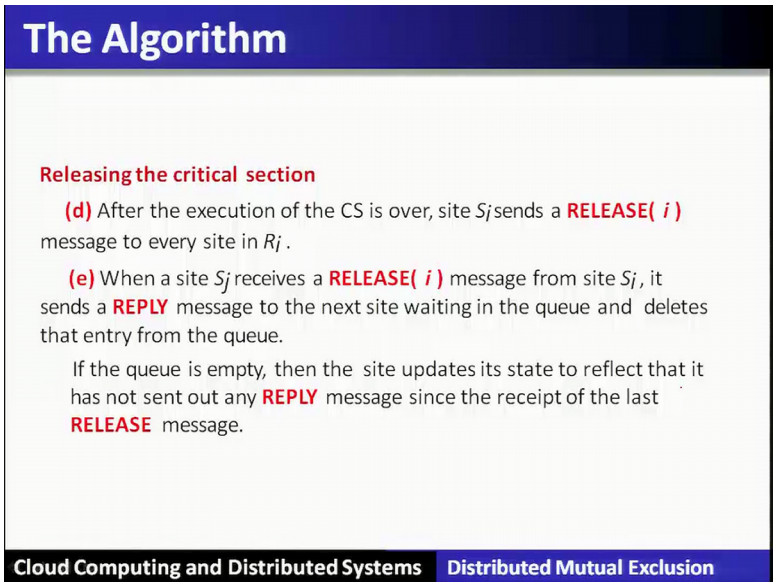
Let us see the algorithm which is given by the Maekawa for mutual exclusion which is called a quorum based mutual exclusion algorithm the site S i executes the following step to execute the critical section the first one is to request for the critical section site S i request to access the critical section by sending a request and request of i and this particular message is sent to all the sites in its request set R i.

Now, when a site S j receives the request i message it sends a reply message to S i provided it has not sent a reply message to a site since its receipt of the last release message. Otherwise it queues up the request i message for later consideration third one is or second one is called executing the critical section that is site S i executes the critical section only after it has received the reply message from every site in R i.

Let us see through an example let us consider the 2 sites or 3 sites i j and k. So, if i is requesting it has to send the request set, let us say the request set of i is having only j. So, it will send the message only to j not to k this is the first step in the second step when the site S j receives the request message then it sends a reply to S j provided it has not sends a reply to anyone since j has not sends a reply so, it will send a reply back. Now if it has send a reply already to some other site then it will not send then it will queue up this particular request needs queue this request will be there why because it has already given the reply.

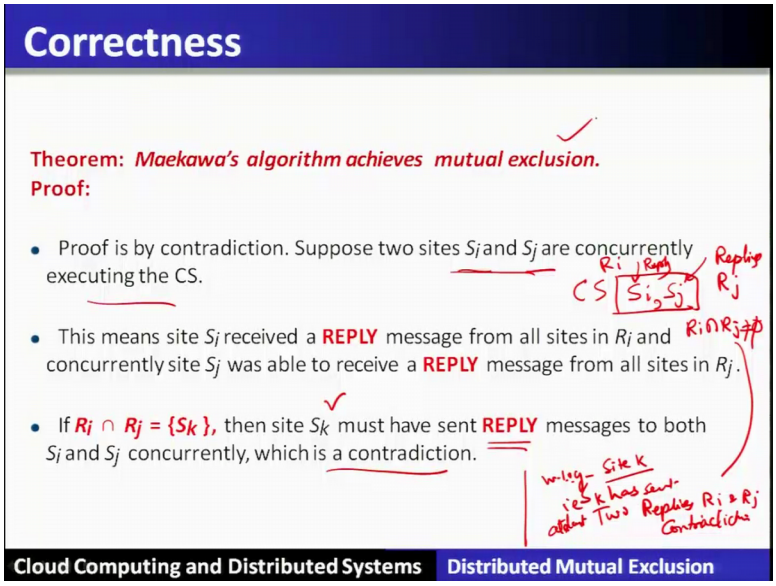
The critical section; that means, when site j replies back and this is the only site in the request set; that means, i has got the replies from all in its request set, then it will be in critical section. So, at this point of time it will be in critical section, it will execute in a critical section.

(Refer Slide Time: 51:55)



Now, let us see the third step, releasing the critical section after executing the critical section by a site i the site i sends the release message to every site in R i now when site S j release message from S i; it sends a reply message back to the next site waiting in the queue and deletes that entry from the queue. If the queue is empty then the site updates its state to reflect that it has not sent out any reply since the receipt of the last release message.

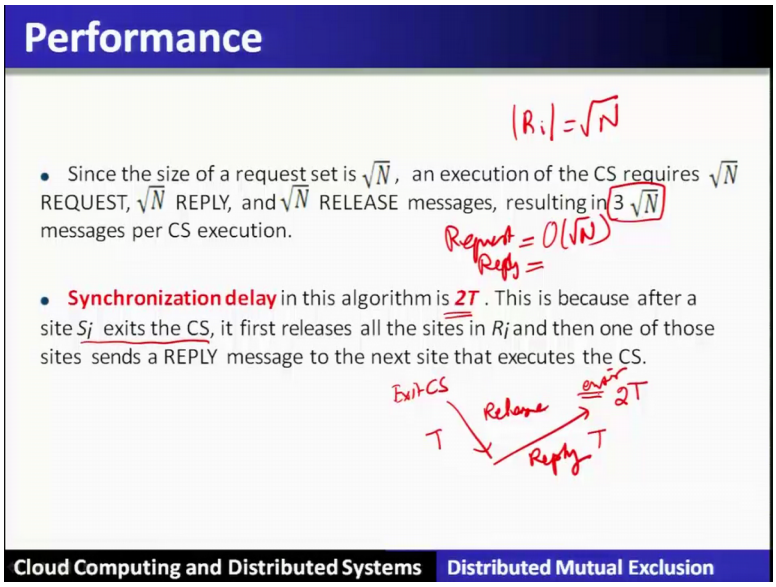
(Refer Slide Time: 52:27)



Maekawa's algorithm achieves mutual exclusion proof is by contradiction suppose two sites S i and S j concurrently executing in into the critical section this means that the S i has received the reply from all the sites in its request set and S j also has received the replies from its request sets R j. Now we know that there is a intersection properties which says that R i intersection R j is not null therefore, there must exist a site without loss of generality.

Let us say that the site is k. So, k has; that means, this means that k has sent two replies at least, that is two R i and R j which is a contradiction why because the second property, that is the second property which says that it can send only one reply every site has to send only one reply which is the contradiction. Hence, the our assumption we say that the two sides are currently executing critical section hence it achieves the mutual exclusion.

(Refer Slide Time: 54:24)

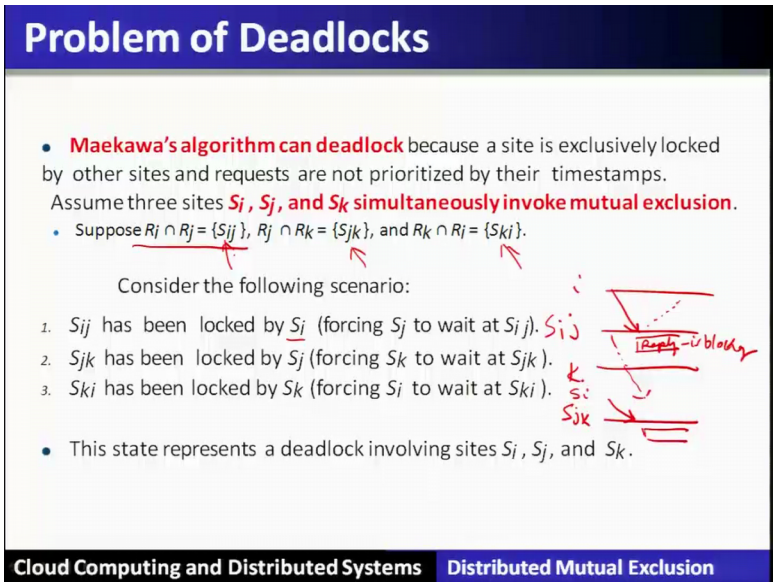


Next is the performance of this algorithm now we see that there are 3 different type of messages request set requesting the second one to get the replies and then finally, when it exist the critical section release, now you know that the size of request set is root N. So, the request messages will be of the order root N similarly reply also will be that much amount and the release also will be so; that means, there are 3 route N messages for critical section invocation is required here in this algorithm. Synchronization delay is 2 T why because after S i exit the critical section then it sends a release to all other sites and

then the site which has queued up the other requesting site it will again send the reply message.

So, release will trigger the reply message, hence this is one t this is another T that is total 2 T; that means, it will exit after it exist the critical section 2 T is the synchronization delay to enter to a critical section by any requesting site, hence the synchronization delay is off to T.

(Refer Slide Time: 56:16)

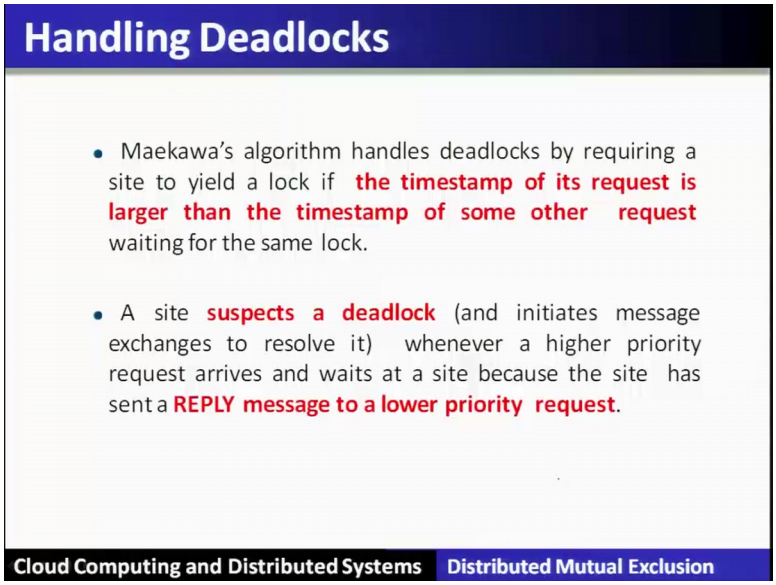


The Maekawa's algorithm suffers from the deadlocks because a site is exclusively locked by the other sites and the requests are also not prioritize either timestamp here you have seen that there is no use of Lamport timestamps to see this plus considered the 3 different sites S i S j and S k simultaneously they want to execute the critical section. So, they invoke the mutual exclusion algorithm and we know that out of these sites S i S j and S k if you take the intersection of their request set let us assume that R i and R j have the common site that S i S i j, S similarly S j and S k will i have a common site as j k similarly for k i it will be S k i.

Now, consider the scenario that S i j has been locked by S i; that means, this will force S j to wait at S i j. So, for example, if this is if S i forcing S j to wait. So, here S j will be waiting at S i j because it has send a request, but it has not send the reply back reply is queued up. Why because it has already send the reply to some other to some other site similarly, S j k has been blocked by S j let us say here there is S j and that is locked.

Similarly S k i has been locked S k so; that means, these 3 different sites S i S j and S k, they are waiting for the replies from their common sites that is S i j S j k and S k i which has already given the replies and waiting for the replies from the other waiting process for this will form a circular wait and therefore, this may lead to a deadlock in this approach.

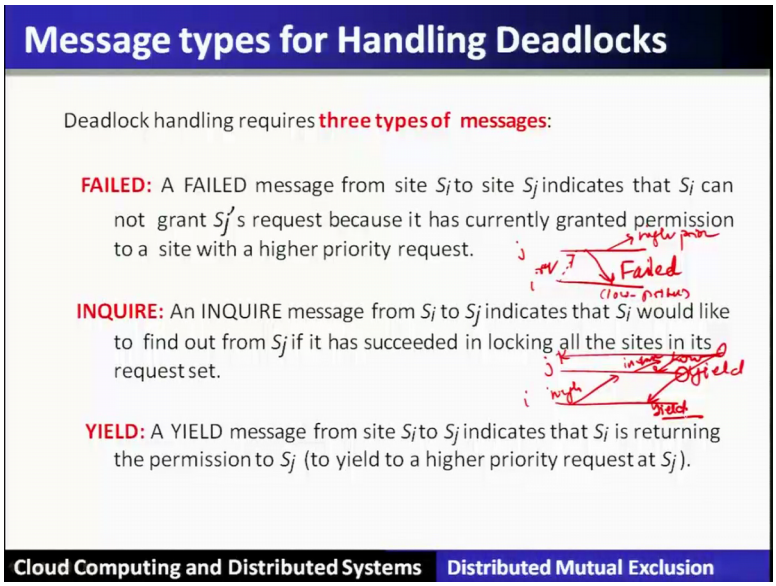
(Refer Slide Time: 59:05)



How to handle the deadlock Maekawa algorithm has also given the provision to handle the deadlock; so, Maekawa hand I will go algorithm handles deadlock by requiring site to yield the lock. If the timestamp of its request is larger than the timestamp of other requests waiting for the same lock [FL]. Handling deadlocks Maekawa's algorithm handles deadlocks by requiring a site to yield the lock if the timestamp of its request is larger than the timestamp of some other request waiting for some lock.

That means, it has given a provision of yielding a lock if higher priority process comes and founds that the replies has already been sent to a low priority process, then the lock will be yield and given back to the higher priority process therefore, this way that a lock will be broken up. Let us see now it does in Maekawas algorithm this concept of yielding.

(Refer Slide Time: 60:38)

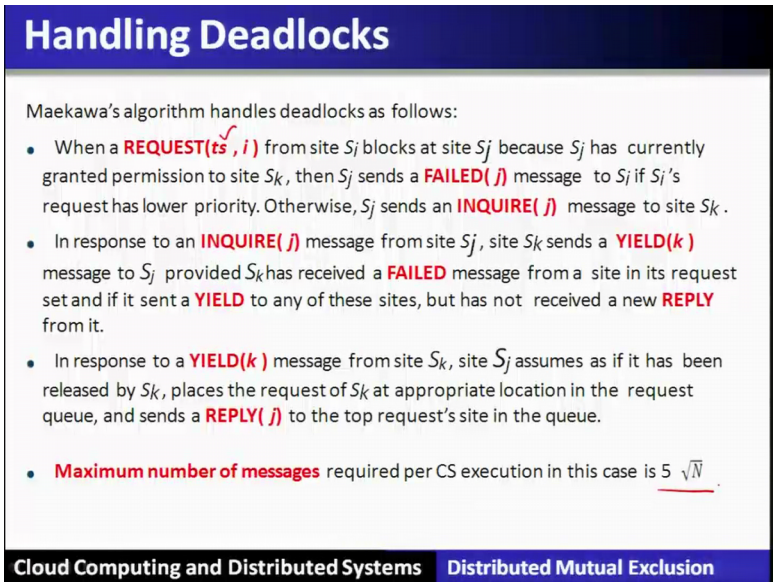


To handle the deadlock in the Maekawa's algorithm there are 3 different type of messages the first one is call failed message failed message from a site i to j indicates that as i cannot grant S j is this request because it has correctly granted the permission to a site with a higher priority request, take this example for example.

For example, this is let us say i and j i is requesting for a lock this will be given a failed message, if j has already given to a high priority process and this i is a low priority process. So, it will fail, the second message is called inquire if this is not the case N if i is a higher priority and j has already given to some low priority, then it will not fail, but it will inquire message an inquire message from S i to S j indicates that I would like to find out from S j if it has succeeded and locking all it sites in its request set. So, here what j will do for example, j will send an inquire message 2 k to find out whether k is successful in getting all the locks in its request set if not then there will be a provision called yield message. So, yield message from site S i to S j will indicate that S i is returning the permission to S j to yield to a high priority request at S j.

So, for example, after inquiring it has identified that it has not k has not succeeded in getting all the locks then what will happen this j will take the action it will yield the lock; that means, it will take the lock back from k and it will give it to higher priority process using a yield. So, in this way you see a low priority process is broken out from the circularly waiting set of sites using this 3 kind of messages failed inquire and yield.

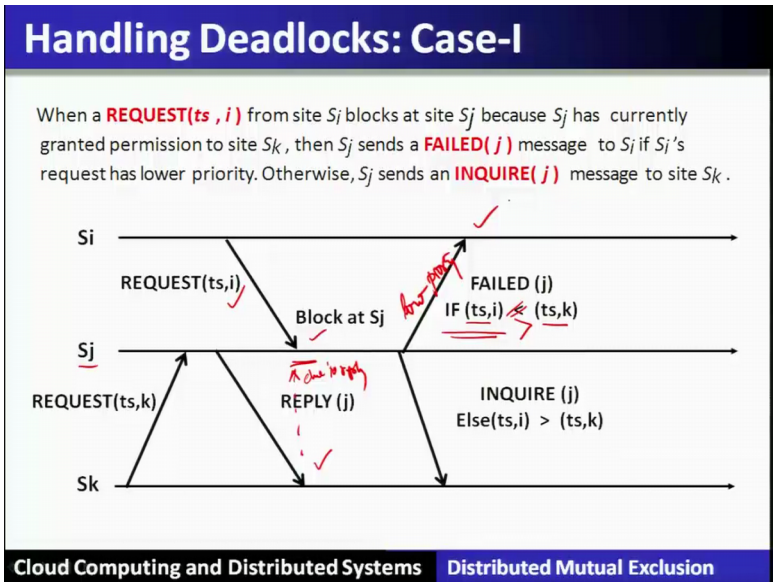
(Refer Slide Time: 63:44)



Let us see how this all happens inside the algorithm. So, now, it also uses the timestamp to determined the high priority processes. So, when i request with the timestamp t i from a site S i blocks at S j because S j has currently granted permission to site S k then S j will send a failed message to S i if S i is request has a lower priority otherwise as j will send it inquire message to site S k.

Now, in response to inquire message from S j k, I will send yield message to site S k provided S k has received message from a site in its request set; that means, S k has not succeeded in getting all the locks acquired from its request set and you send yield to any of these sides. But has not received the new reply from it, in response to the yield message from the site S k S j will assume that it has been released by S k and places the request of S k at the appropriate location in the request queue and send the reply to the top of the request set in the queue in this process the total number of messages required to enter in a critical section is 5 root N.

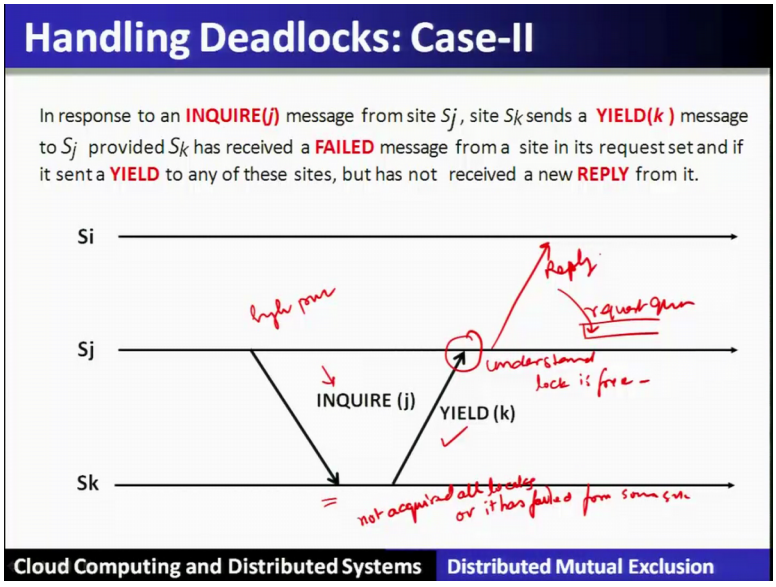
(Refer Slide Time: 65:26)



Let us see through this particular example we have already explained it, i is requesting and which gets block at at S g. Now S j will has already send the replies to k therefore, it gets a block due to this reply now then this particular message which is requesting will basically use the timestamps to compare their priorities.

Now, if the timestamp if the timestamp of i is basically less than the timestamp of k then it will so; that means, if the timestamp of i is higher than the timestamp of k; that means, it is if it is low priority, then it will send a failed message.

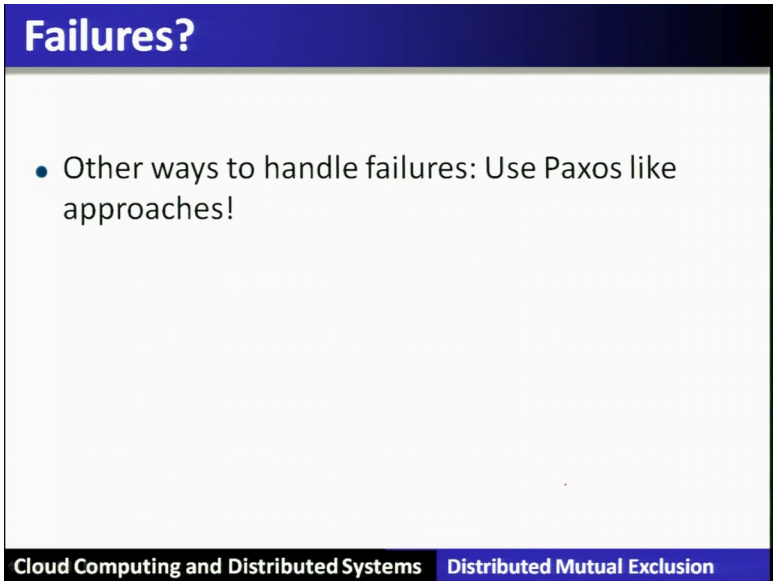
(Refer Slide Time: 64:42)



So, let us see through this example here in this case. So, if it is the lower if it is the high priority, then it will send a inquire message to find out the status of the lock and if it is found that S k has not acquired all the locks or it has failed from some site then in that case it will yield the lock S k will yield the lock.

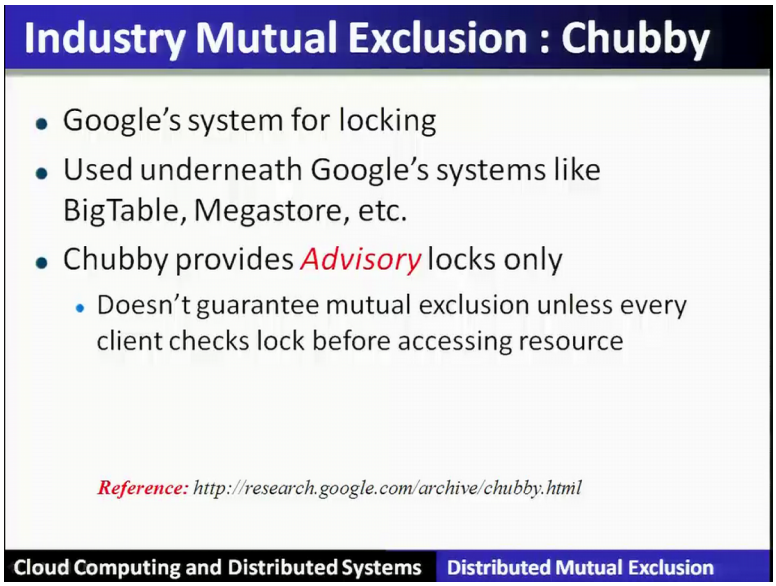
S j will understand that this lock is free it can be given to any other process. So, it will now check its request queue and whatever process is on the top of the queue, it will send the reply to the other higher waiting process. This way this deadlock is handled in this particular scenario this is what I have already explained.

(Refer Slide Time: 68:02)



Now, the failures we have seen the that the previous algorithms have assumed there is no failure. So, we will see some other method how to handle the failures and yet you can be solving the mutual exclusion problem. So, we will see that Paxos like schemes are being used widely in the industry solutions.

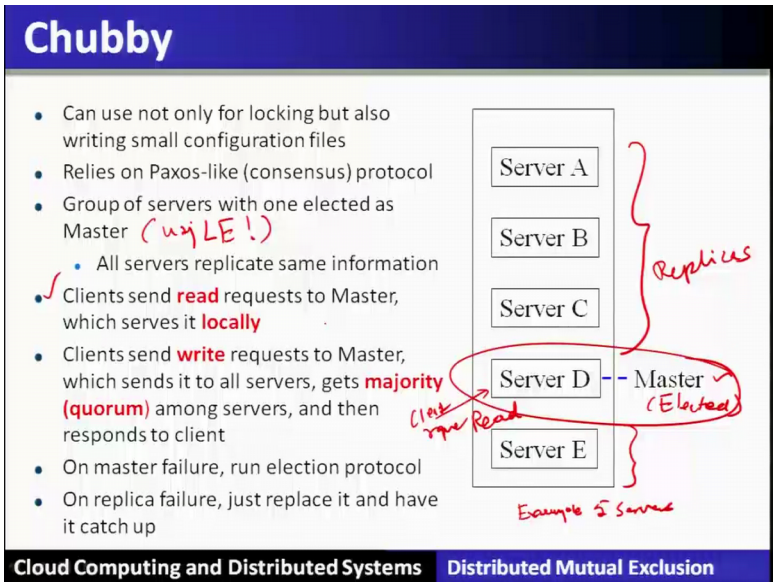
(Refer Slide Time: 68:25)



So, we are going to discuss briefly about the Google's chubby system which is nothing, but a locking system which uses the Paxos like scheme for the failures handling failures and also to solve the problem of distributed mutual exclusion.

This Google's chubby system in its Google system you can in the, its use in the BigTable and megastore like applications now the chubby uses advisory locks; advisory locks means that the client has to invoke this particular lock then only the mutual exclusion is guaranteed if the client does not do that then the mutual exclusion is not taken care of. So, that means, that it does not guarantees the mutual exclusion unless every client locks before accessing the critical resources.

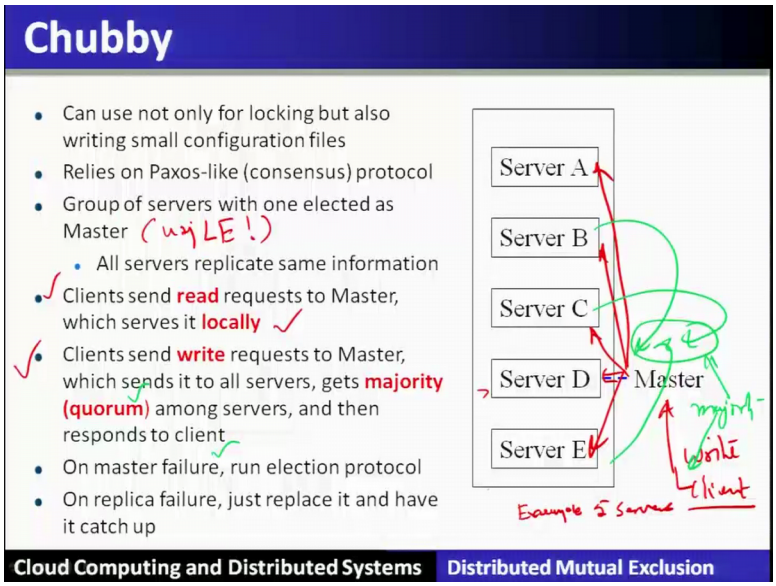
(Refer Slide Time: 69:38)



Let us go and see the details of chubby system. So, chubby is a lock is a distributed lock service and it also basically writing into a small configuration files. So, locking and configuration files are two important services which this Google chubby system provides and which relies on the Paxos like consensus protocol to deal with the failures. In this particular system there is a group of servers out of them one is elected as a master using any leader election algorithm and other servers are the replicas.

In this example this is a group of 5 servers one which is called a master let us say that it is elected among other 5 servers and all other servers which are not elected they are called replica servers; in this particular protocol, the client will send the requests the read request will be sent to the master and master will serve locally. So, the client send the read request to the master which serves it locally directly.

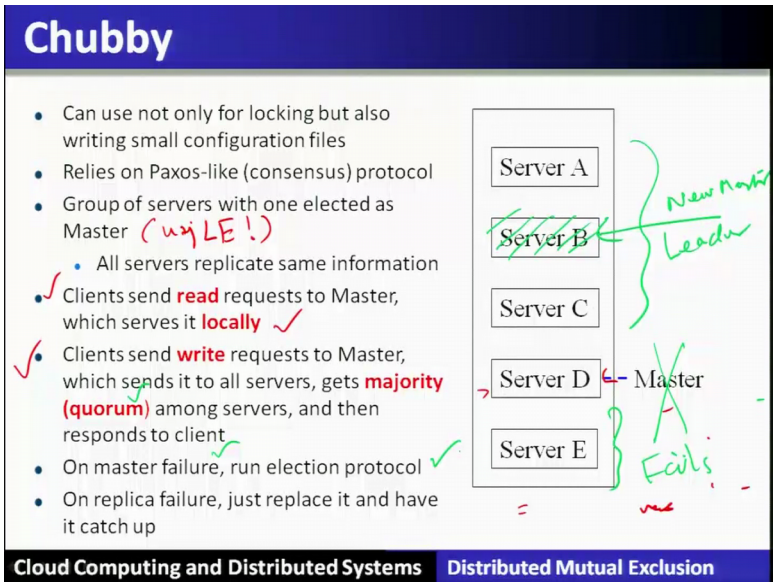
(Refer Slide Time: 71:31)



However when a client sends a write request to the master then master will send it to all the servers it will broadcast and wait for the replies.

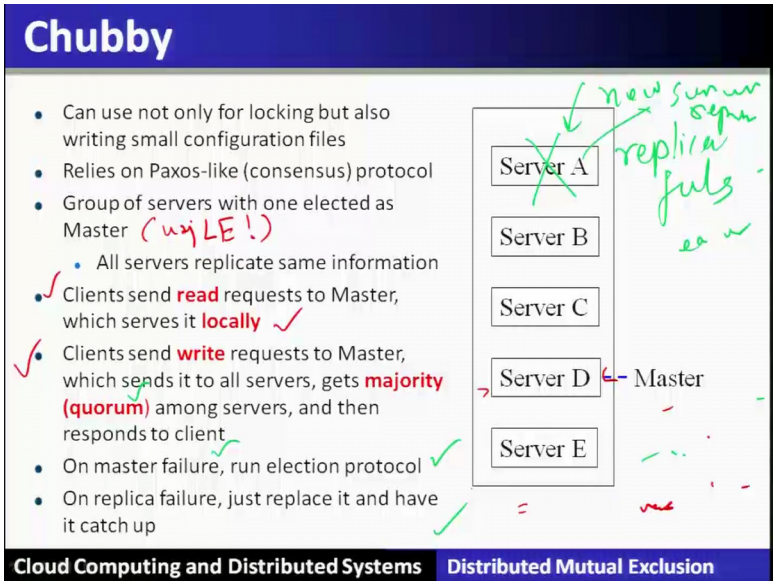
If the majority replies this becomes a majority. So, majority is also decided by the size of the quorum. So, if the majority gets the replies or response. So, the master correspondingly response back with the right operations, now you know that if the majority is taken care in the right operation and as far as any two quorums are concerned they are not null as far as the intersection properties are concerned so; that means, there must be some site which will like which will mediate. So, that all replicas are basically updated with this right operations eventually. As far as the failures are concerned, when the master fails when this particular master fails.

(Refer Slide Time: 73:01)



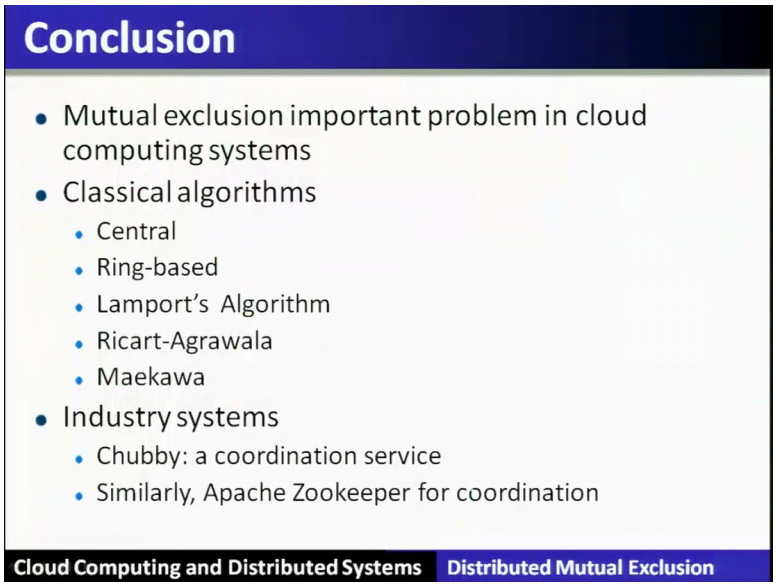
Then again among the remaining the leader election will happen and let us say this will become a new master; now when a replica fails on the other hand when this replica fails.

(Refer Slide Time: 73:20)



Then it will just replace it with some other new server and it will catch up with the other set of operations.

(Refer Slide Time: 73:50)



Conclusion; mutual exclusion is an important problem in a cloud computing system and we have seen several classical mutual exclusion algorithms such as central algorithm ring based algorithm Lamport's algorithm, Ricart Agrawala algorithm and Maekawa's algorithm. We have seen the Maekawa's algorithm is more efficient as far as performance in compared to the Lamport's and Ricart Agrawala algorithms are concerned we have also seen that these algorithms are assuming that there are no failures as far as the industry systems are concerned they assume thats a system fails and yet they are able to resolve the mutual exclusion.

Failures are handled in a Paxos like scheme that we have seen in a chubby system which is basically a coordination based lock system. Similarly we will see that apache zookeeper is also a coordination service that also uses the Paxos like schemes and also solve.