Database Management System
Prof. D. Janakiram
Department of Computer Science & Engineering
Indian Institute of Technology, Madras
Lecture No. 24

Distributed Transaction Models

We have been considering transactions in the data executing on a single system. Implicitly this is assumption that we been making do we did not explicitly state that underline assumption of the model transaction model is that the transactions are executing on the node and the data is also fully resident on same node. What we mean by node is the node is a single computer system on which there is an operating system image and on which the database is running and now the transaction which are part of applications are all running on the same system, that's the assumption which we are making when we are actually explaining all the transactions models.

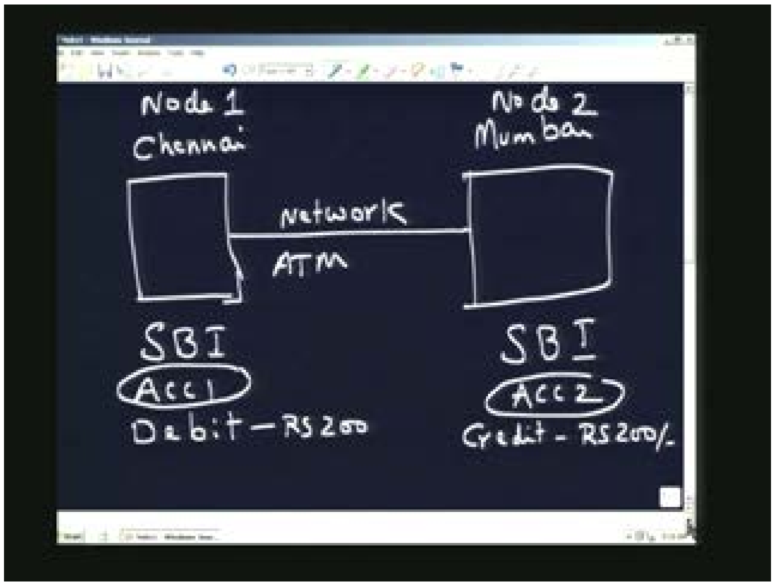
Now consider an example where for example, railway reservation system where sitting in Chennai and your trying to make booking for train leaving out of Delhi which could mean that you are actually trying to access the database that is resident physically on a computer system in Delhi sitting in Chennai. Now, it could mean several things. It could mean just that you are actually trying to access the database to a physical connectivity which could be a telephone line, a lease line, a some kind of connectivity between your interface which could be just display interface to the system in Delhi which means that you are still physically working on the computer system that is located in Delhi that's could be one model. But, this will not be one model which could be present when you work with systems which are dispersed and which are connected by wide area network, local area network or this is what we called as distributed system.

When computer systems are connected by a network they could logically present a single node kind of abstraction for the end user. This is what we mean by the distributed system. Now when the data is distributed and the transactions can be executed on different nodes of the distributed system, we called the scenario as distributed transactions that means we are no longer assuming that, the transactions execute on a single system computer system but they could be executed on different nodes of a distributed system. To explain this scenario what I will do is, I will take a very example of a banking system and explain how the scenario look like.

Let us say that we have a bank in Chennai SBI branch. Now I have also a branch in let us say Mumbai again SBI branch could be a different bank also and it is possible that i have an account one here and have an account two here and i am actually doing a fund transfer from account one to account two, which means that i am going to do a debate here from account one by let us say two hundred rupees. Now it is possible for me to credit the following thing on to the other thing. Now this is data for account one is resident on node 1 which is basically node okay and the account to which is the other account data that is resident on node two which is in Mumbai. Now this means that physically the data is distributed in two locations one in Chennai branch and the other in Mumbai branch.

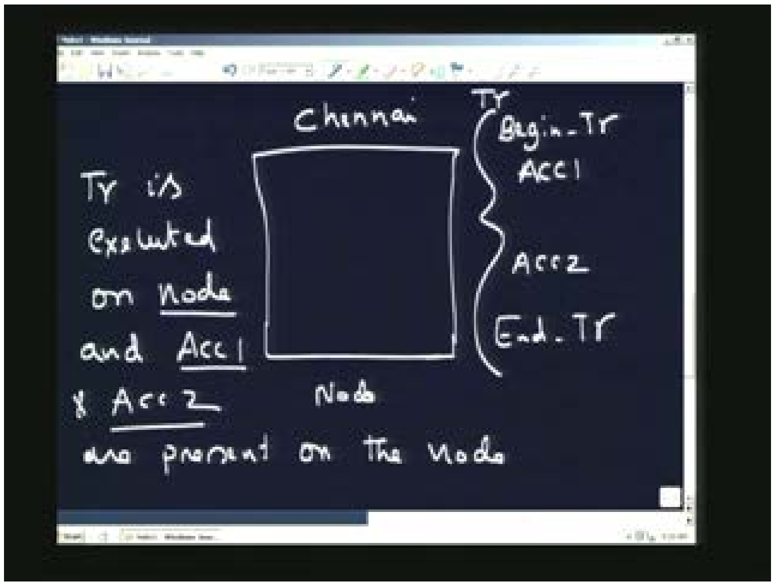
Now if you assume that these two are connected by some kind of a network does not really matter what is the network but you can you can assume that physically these two nodes are connected by an underlined network. This network could be an atm network okay or could it be a fiber optic network or it could be variety of even a satellite network is present between these two nodes. So we are actually assuming that some kind of connectivity exist between these two nodes which affectively means that the information from one node to another node can be accessed or the data can be from one node can be accessed. The thing in another node otherwise vice versa it is possible.

(Refer Slide Time: 7:15)



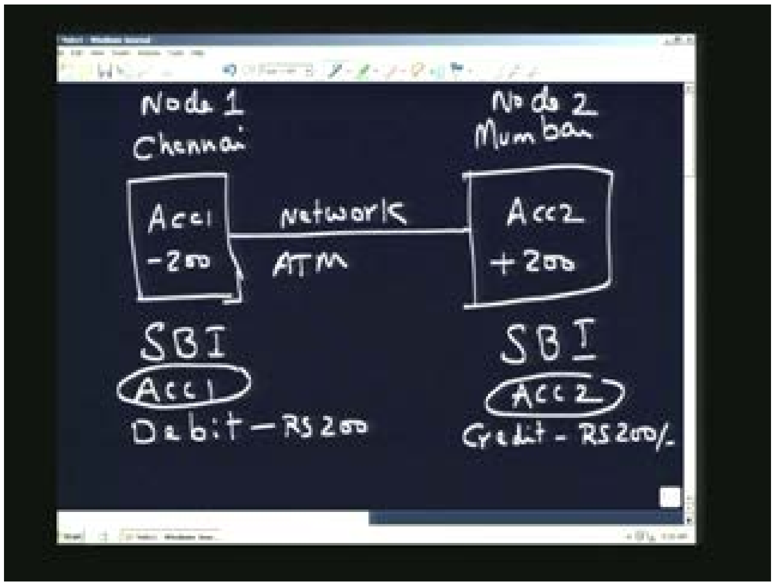
Now supposed to these if you are actually seeing a single case where both account one and account two present on a single node. Both are present in Chennai. This is what the case we actually look at account one and account two and all that we had begin transaction end transaction as part of the transaction here and all this will be executed in the same branch of one system, that means physically this entire transaction is executed on a single node. Now all the things that we so far have been talking belong to this scenario where both account one and account two are present on a single node and the entire transaction. Here TR is executed on a single node. So TR is executed on node which is the single node here and account one and account two are present on the node. So there is no remote access. This TR does not access any remote information both accounts are present on the same node. This is what ideally we are talking about.

(Refer Slide Time: 9:00)


Chennai
Tr
Begin_Tr
Acc1
Acc2
End_Tr
Tr is executed on Node and Acc1 & Acc2 are present on the Node
Node

When we are talking about one single node of distributed system. As opposed to this, if you consider this particular case, we are talking about account 1 and debiting some other two hundred rupees from this account here and account 2 which is present on different node and adding 200 rupees here. Now imagine what could happen if these are executed on two different nodes. Now it is possible in other case, a power has failed in the whole thing is proceeding in the single system case it is possible that power could fail or other things could happen when the transaction is executing.

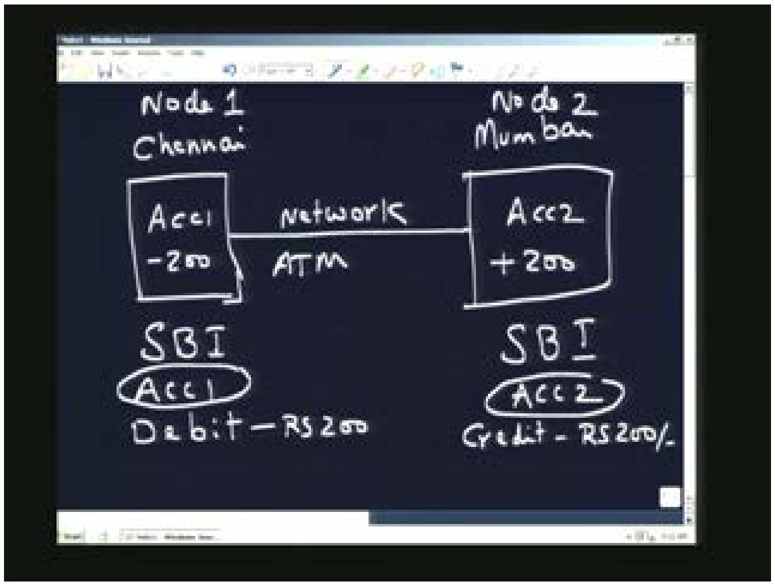
(Refer Slide Time: 9:25)



This is what we consider the acid properties of the transaction. Now if the transaction debiting and crediting from one account to another account you should ensure that the properties are completely preserved as acid properties are completely preserved as far as the transaction is concerned. Now in a single node now when the power fails the log records could be used to make sure that the transaction will always be an in consistent state. For example: for part of a transaction executed that is credit part is executed the debit part is not executed.

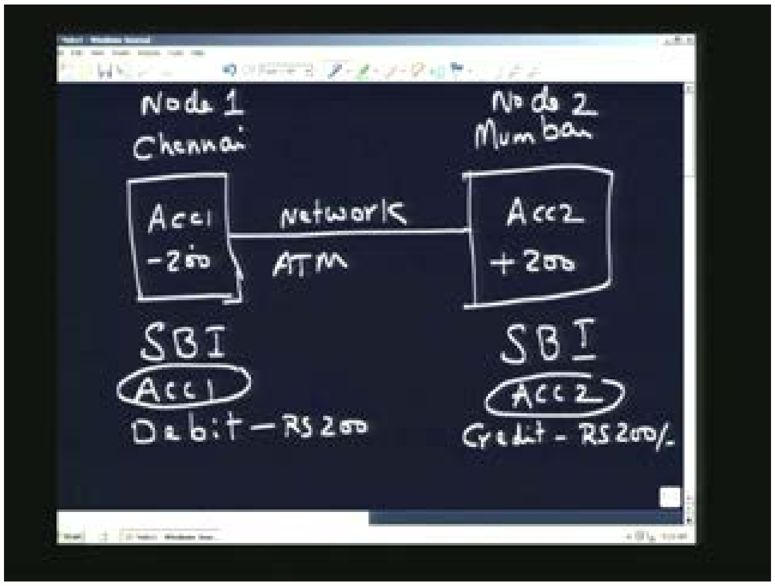
This can be figure out from the log logs of the transaction and either you can do undo or redo the transaction depending upon the state of the current transaction logs record that that is preserved. Now the scenario becomes quite complicated if you essentially look at distributed system. Now you imagine account 1 debiting is happening in one node account 2 crediting is happening in another node. Now it is possible that, part of the transaction is got executed part of the transaction is not executed for various other reasons. Firstly, to know that they both got executed itself is an important issue which is like two friends separated in two different locations try to ascertain whether the other friend exactly the other than excepting to do.

(Refer Slide Time: 11:07)



It requires a few phone calls and also making sure if the phone rings, the person is present. So many other issues get involved when people are physically separated and they have to coordinate and do some activity, but on the other hand, if they are present in the same node, in the same room, it becomes lot more easier when they have to coordinate. The minute they are physically separated when phone rings, somebody does not pick up does it mean they are busy with something that is why he did not pick up the phone. You have to make assumptions really relating to behavior to other node with respect to your node and also if you put a telegram it could be a different mode of communication compared to ringing using a telephone and trying to reach him. So what is the mode of communication between these two systems between node 1 and node 2? What is the kind of communication primitives that are present between these two systems that also makes an important requirement when you study this model.

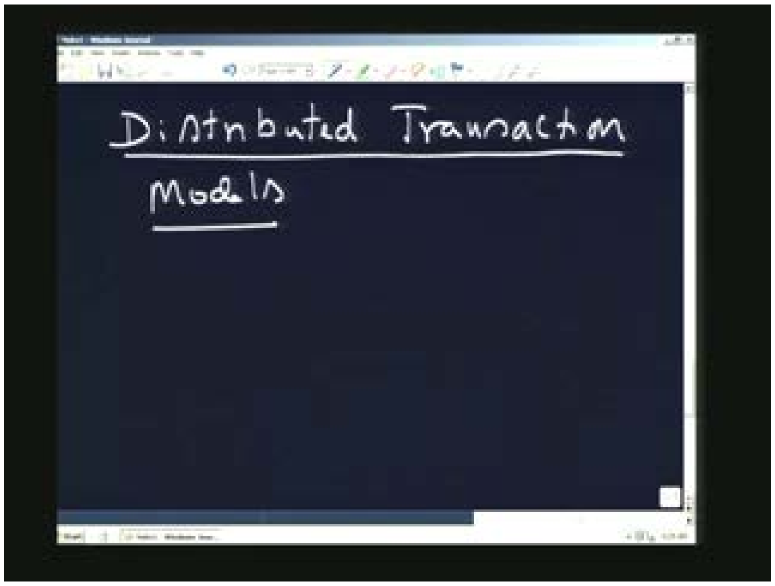
(Refer Slide Time: 12:10)



What you are trying to say is, if you assume the transactions execute on a multiple nodes of distributed system, the whole approach to concurrency control and commit protocols takes an extra dimension because you have to consider here the possibilities of distributed system. What can happen in distributed system? Node can fail, network can partition, network can fail with results in the system getting partition. For example: there is a temporary network failure between node one and node two, the node two may not be reachable from node one. So all these are becoming important.

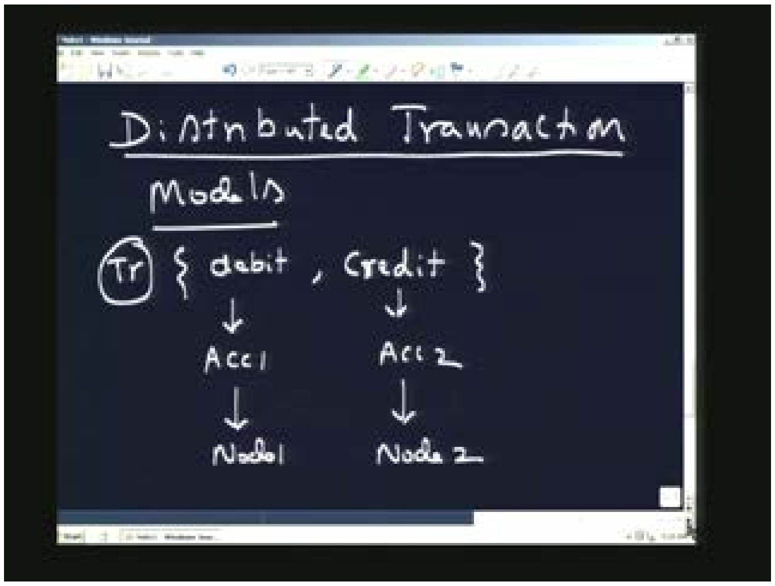
Now the transactions take these in to account and they have to ensure the acid properties when these things happen that's the most important thing. So we will actually consider this model as a distributed transaction model and we will study some concepts relating to distributed transaction model starting from this lecture.

(Refer Slide Time: 13:46)



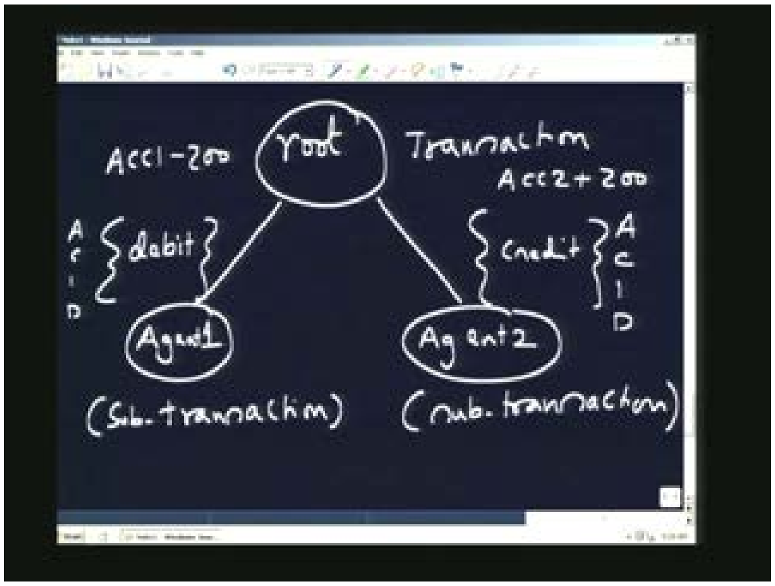
Now I will explain at very high level, the distributed transaction model and get start more details of some of more transaction models to explain the essential difference between centralized system database system and distributed systems with respect to the transaction models. We will take a very simple example and illustrate with respect to the simple example how exactly we can build a distributed transaction model on top of distributed database systems. Now continuing the discussion we had on credit debit transaction, what we will see essentially in the case of distributed transaction is a transaction TR which we have seen there will have now, two portions which is the debit portion of the transaction and the credit portion of the transaction. Now, debit portion of the transaction is executing on a different account 1 which is on a node 1 which is Chennai node. This is resident on a different node and this resident on node 2. Now we call this in credit and debit two sub transactions of this root transactions TR. Now TR becomes the root transaction.

(Refer Slide Time: 15:10)



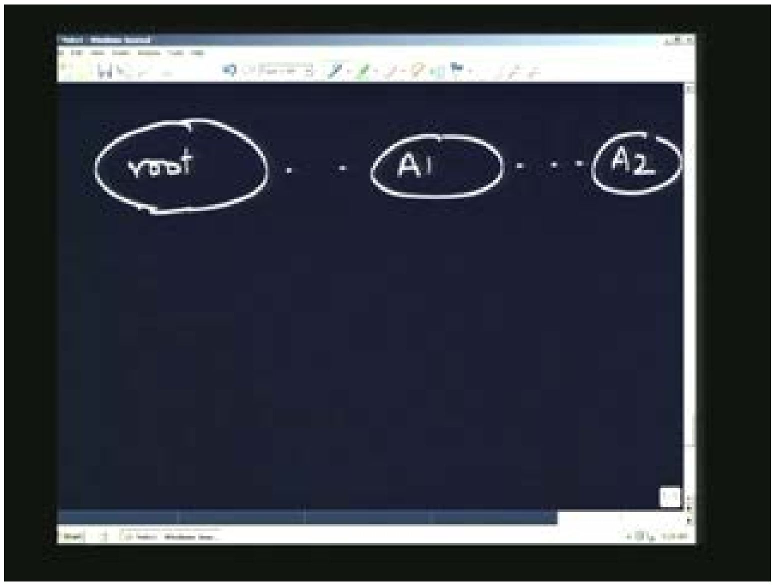
So there is a root transaction which is the complete transaction. Now what this root transaction will do is it will actually spawn two agents which are nothing but two sub transactions agent 1 and agent 2. This agent 1 is actually sub transaction of the main transaction agent 2 is another sub transaction 2. Now a single transaction is split into two sub transactions. So that, these two sub transactions are executed on two different nodes of the system. For example: the root is going to be on one node, agent 1 is going to be in another node. Agent 2 is going to be in another node. Now agent 1 is going to execute the debit portion of the transaction and agent 2 will execute only the credit portion of the transaction that means, it will access account one here and make sure in the account two hundred rupees is debited and here it will access account two and make sure two hundred is added here. Now while doing this, all the acid properties have to be preserved as part of this which means that atomicity, concurrency, isolation, durability all the acid properties independently have to be preserved as part of this particular exercise.

(Refer Slide Time: 16:58)



You do not want if power goes off in the middle of transaction you do not want to be unsure what really happen to your account on node one. It should be the preserver the property that two hundred rupees detected by this transaction atomically you know when accessing this account, other transactions are not allowed to access this account. When it is modifying, somebody not to allow the results before it finish its computation. Now after finish it executing it is completely preserved on the database. All these in an atomicity concurrency, isolation, durability properties. Now to achieve this issue at the highest level, this is the model what we have is the agent. The root agent spawning other agents okay agent one and agent one on different nodes. These agent 1 and 2 are responsible for executing the transactions on the local node and together they form a complete transaction.

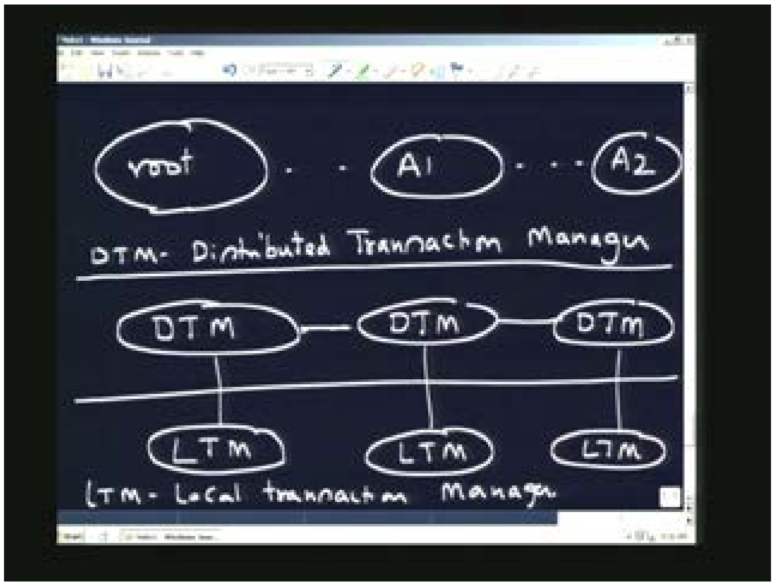
(Refer Slide Time 18:19)



For example: additionally we have to worry about if A 2 alone is done, A 1 is not done, then again we have the inconsistency because A 1 is debit, A 1 is debit A 2 is credit. So you do not want the debit being done, not credit or vice versa or both have to be done simultaneously. When this is done, this also done this is also not done this is also done that's what we mean by atomicity either the whole thing is executed or none of its executed. Now to ensure that this kind of agreement is reached between the various sub portions or sub transactions that are executing on different nodes. We need what is called an exercise a module that i actually does this which make sure that transaction properties which includes the commit property that all of them either commit or none of them commit.

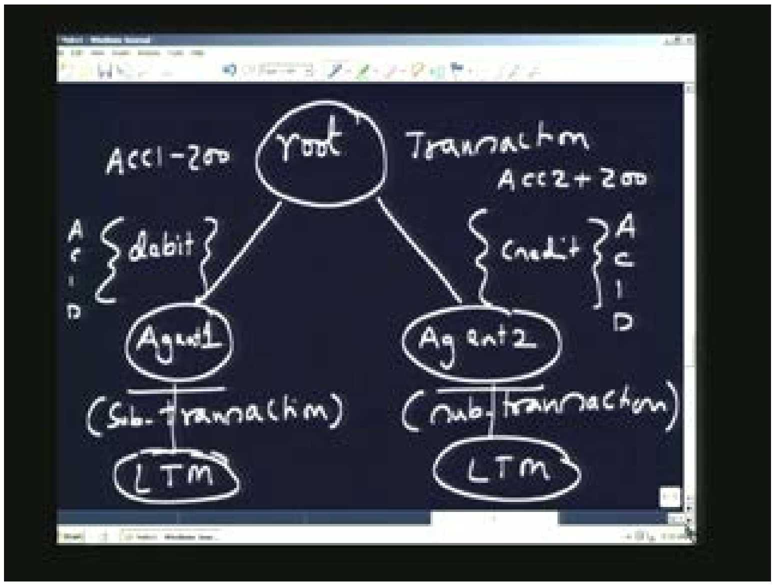
The transaction the transaction end all these to preserve these we will have an all nodes manager running which is called the distributed transaction manager. This is the responsibility of the distributed transaction manager. This DTM stands for distributed transaction manager distributed transaction manager. Now that DTM on every node ensures that the transaction properties are obtained on each one of these nodes when they are participating in the execution of the transaction. Now each local execution which needs to again maintain transaction properties will come under what we will see in LTM, LTM is nothing but the local transaction manager and if you see, LTM stands for local transaction manager.

(Refer Slide Time: 20:24)


root
A1
A2
DTM- Distributed Transaction Manager
DTM
DTM
DTM
LTM
LTM
LTM
LTM - Local transaction Manager

If you see exactly what we have said in the earlier slide, the credit and debit here which are executed by agent 1 and agent 2 needs to be preserved by the transaction property and this is obtained by the acid by the preserved acid property for debit transaction and credit credit transaction. This is ensured for agent 1 and agent 2 by the LTM. LTM preservers the logs and all the related things done in the centralized database context. The LTM make sure al the things are executed by the agent or as per the acid properties and that is what is achieved by actually having the LTM.

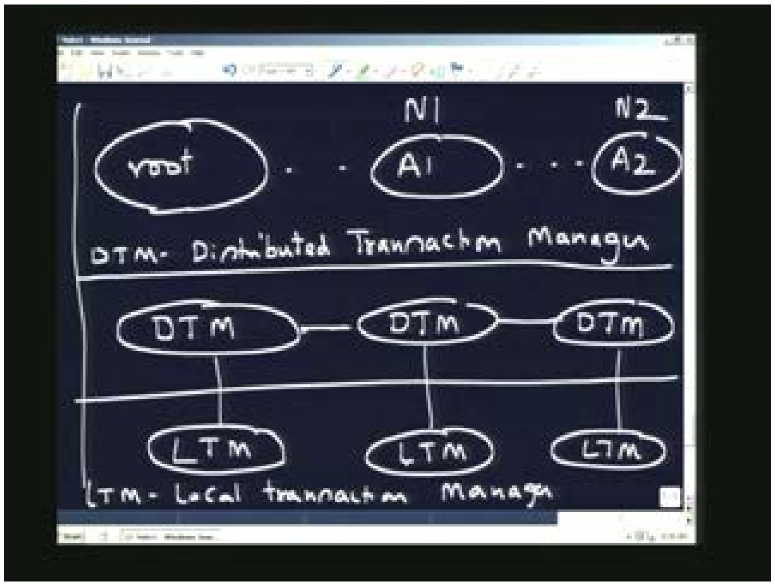
(Refer Slide Time: 21:41)



On each of the nodes to explain this is the model, this is the distributed transaction model. What you see at the highest level to just recap what we have been telling along the complete transaction. Now become the root transaction as part of the root transaction when you will have the sub transaction which needs to be started on different nodes of the distributed system. Now here agent1 gets started node 1 and agent 2 started on node 2, agent 1 indicates a sub transaction the root transaction. Agent 2 indicates the other two sub transaction of the root transaction.

Now for staring this agents and also making sure that the coordinate among themselves when they are committing, corresponding properties are achieved by the distributed transaction manager which is called the DTM. Now the DTM is responsible for preserving the transaction properties across the various nodes. Now, when the agent starts on a local machine whatever it gets executed in the local node, it preserve by this local transaction manager that means the local transaction manager is responsible for ensuring whatever is executed in the local node obeys the transactional properties and this is the complete model of the distributed transaction model.

(Refer Slide Time: 23:17)

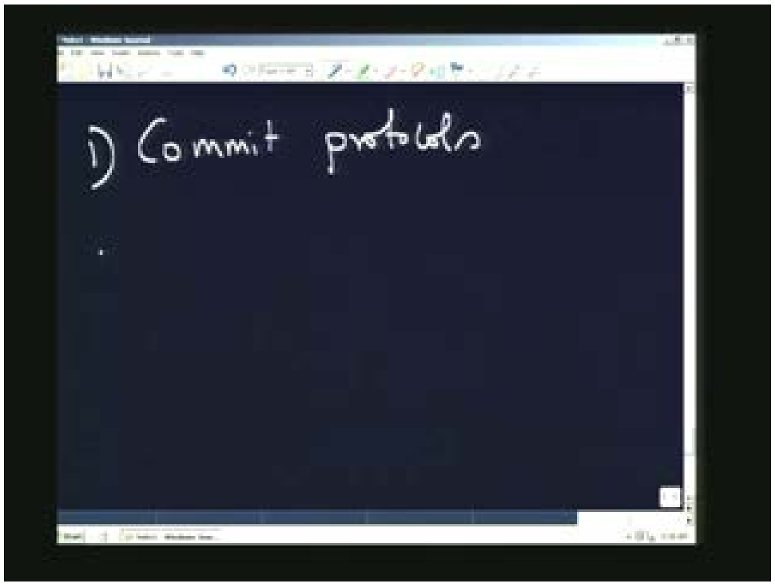

N1
N2
root
A1
A2
DTM - Distributed Transaction Manager
DTM
DTM
DTM
LTM
LTM
LTM
LTM - Local transaction Manager

What we are going to look at is, we are going to look at with respect to this model, how the concurrency control and model the commit protocols are going to be executed in this context and suddenly this becomes little more complicated than what we have studied in this centralized scenario. For example: commit in the centralized scenario is just writing the logs and making sure all the values modified by the transactions are return back on to the database and then if your using two phase locking, then make sure that the locks are released so that other transaction can make use of now the data items modified by the current transaction. So commit becomes a simple exercise of just writing the logs and writing the modified values back on to the database.

Now if you consider the distributed scenario where there are multiple agents running on the multiple nodes of the distributed system, the commit no longer becomes that simple. Now you need to worry about whether node 1 and node 2. For example: some reason node 2 wants to commit each part of its transaction and then you need to know node1 should roll back. You need some kind of agreement to be reached among the various nodes before they actually commit the values on to the database system. So this is basically called the commit protocol, commit protocol is the most specialized protocol than a general agreement problem.

Now all that you have to agree in the case of commit protocols is either to commit or abort. There is no other agreement that you try to reach in the case of commit protocol because one could mean i want to agree to commit my part of the transaction. Zero could mean that i could abort a part of the transaction. So all the nodes at the end of this transaction that agents involved in distributed transaction have to agree for one or zero depending on one actually means that all together to commit zero means they do not want to commit. So we typically we going to look at commit protocols and study various commit protocols for achieving consciences. This is the first part of the distributed transaction model, what kind of commit protocols exist and how do they work with respect to various scenario like node failure, network failure, partitions.

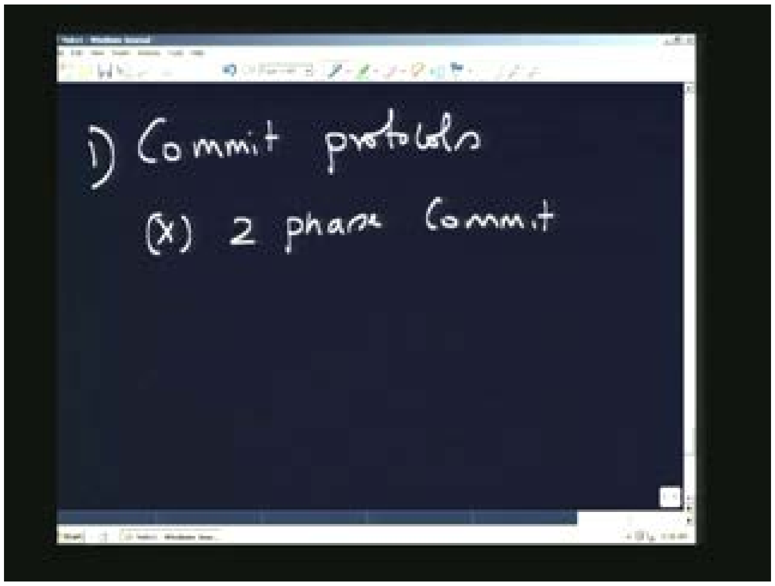
(Refer Slide Time: 26:15)



1) Commit protocols

So we are going to look at various scenarios what could happen in the distributed system with respect to this how these commit protocols really work that's going to be the first part of the lecture, going to focus on the commit protocols. We are going to essentially look at two kinds of commit protocols. One is the two phase commit protocol which has the name suggest does it two phases that means the first phase which participations are prepared, in the second phase actual commitment take place and that is what we mean by the two phase commit protocol and this two phase commit protocol naturally integrates

(Refer Slide Time: 26:59)

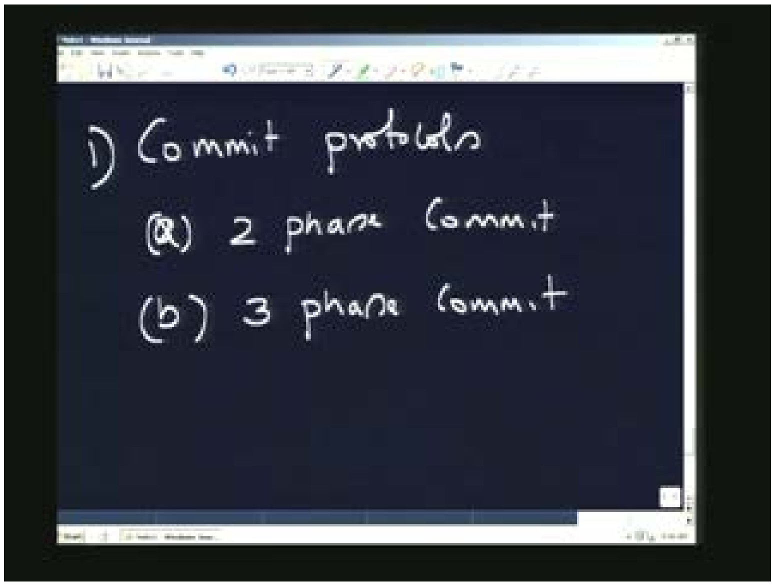


1) Commit protocols

(x) 2 phase Commit

with the two phase locking protocol that we have studied earlier in the case of a centralized database system. So we will study the two phase commit protocol and study extension of the two phase commit protocol in to what we see as the three phase commit protocol. Now the idea of this three phase commit protocol is when sudden things happen in the two phase protocol which we are going to see the protocol will get lock that means the protocol any way to proceed further till the recovery take place in other words certain kind of failures occur, the two phase commit actually locks the protocol blocks the system whereas actually modifying it in three phase commit protocol, we will able to avoid the wait for.

(Refer Slide Time: 28:24)



1) Commit protocols

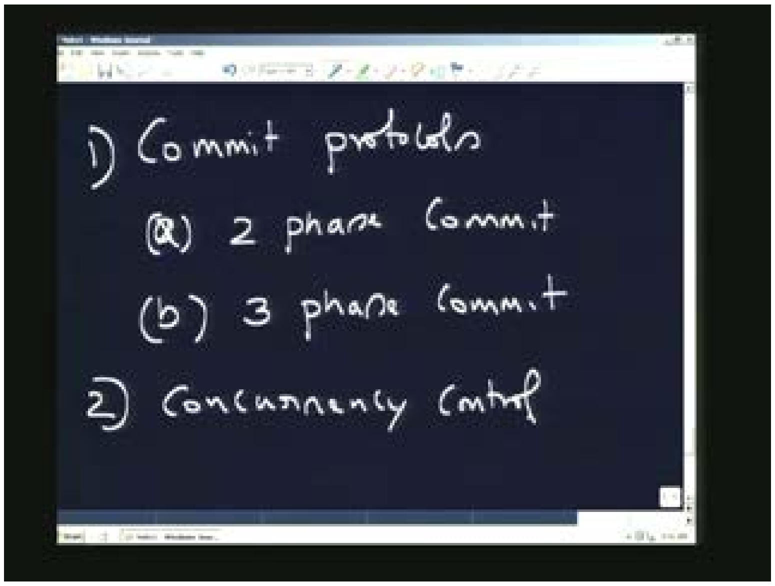
(a) 2 phase Commit

(b) 3 phase Commit

It actually becomes the wait free protocol in some sense, when some kind of failure occurs wait free protocol are extremely important because when failures occurs if the system is not blocked, because of the failure it is a good property of the system, because it allows the system to actually go forward and not blocked by the failure. So wait free protocols are very interesting and important in distributed systems. Now from two phase to three phase commit protocol, what we see is, you can avoid certain kind of failures and make the system resilient for this kind of failures.

The system becomes more robust under three phase commit protocol as opposed to two phase commit protocol. We are going to study the two phase commit protocol in detail and then going to study the three phase commit protocol in the context of distributed transactions to start with. In this particular class, we are going to focus on two phase commit protocol and study the two phase commit protocol in detail and see what under circumstances two phase commit protocol works correctly. Now after looking at the commit protocols, they are going to look at concurrency control protocol in the context of distributed transactions,

(Refer Slide Time: 29:38)



1) Commit protocols

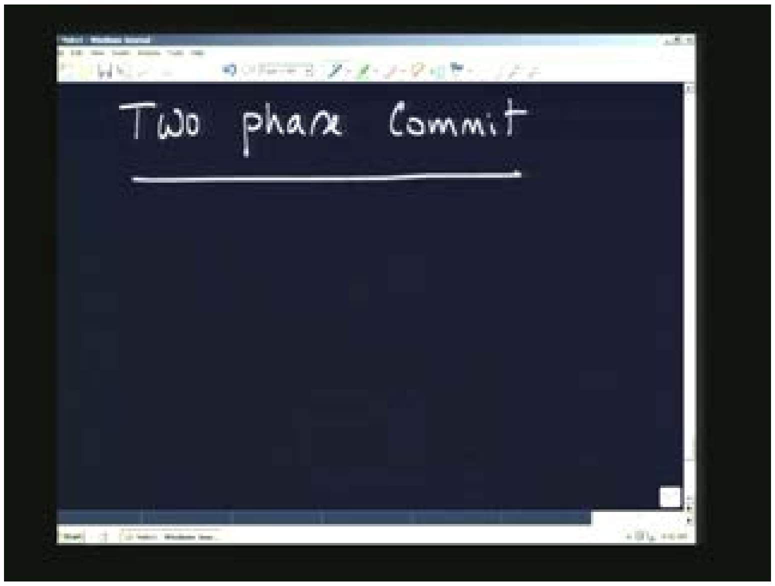
(a) 2 phase Commit

(b) 3 phase Commit

2) Concurrency Control

is going to be later part of the talk we will focus on the concurrency control mechanism as applied to the distributed transaction systems. Now the rest of the lecture from this point will concentrate on the two phase commit protocol and we will try to explain in detail the two phase commit protocol. Now let us look what kind of scenario will be required when you actually looking at the two phase commit protocol. Now to explain what really commit protocol means?

(Refer Slide Time: 30:14)

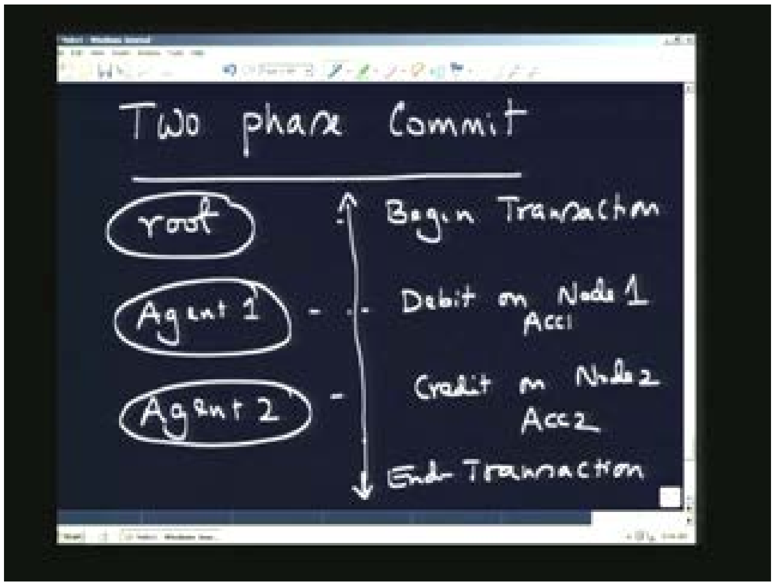


Two phase Commit

We have actually taken a root transaction which actually started executing on this is basically the begin transaction. This is the begin transaction. Now I have basically the agent 1 which is the debit portion of the transaction. This is debit on node 1 debit on node 1. This is basically the sub transactions agent 1 agent 2 is the credit on node 2. This is particularly operating on account 1 and this is operating on account 2. Assume that some point of time, this is the end transaction. Now when the actual agent 1 and agent 2 are started on different nodes of the distributed system and they start executing on the two different nodes at some point of time when i reach the end of the transaction. I am going to look at this part and at this point of time i have to decide whether i will be committing my transaction.

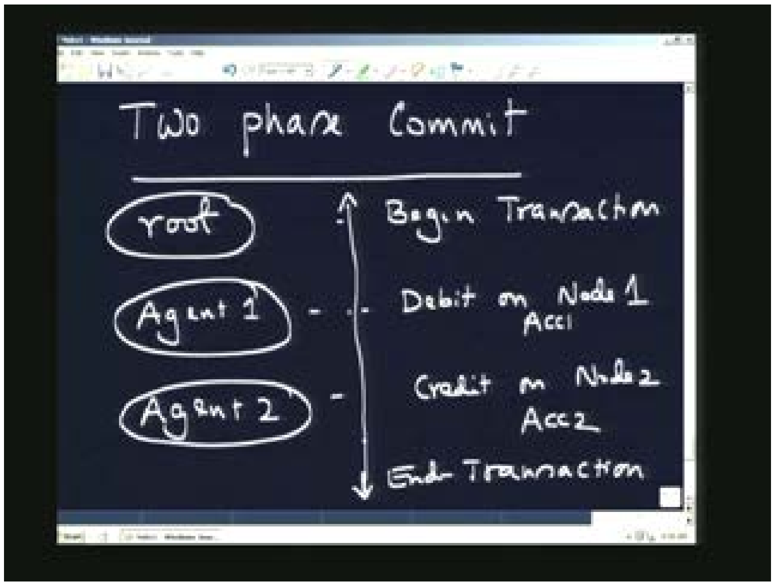
I have reached this point now and when i reach this point, i need to understand now whether agent 1 and agent 2 are willing to now go with writing or doing the debit and credit completely. For various reasons its possible that agent 1 or agent 2 may not be willing to complete part of the transaction. One reason could be account 1 is not present on node 1 or the account has been closed for various reasons which means that account 1 is not present on node 1 which means the debit cannot produce logically further in which case the agent 1 whatever agent 2 wants to do agent 1 will say i am going to abort my part of the transactions because i cant complete my part of the deal.

(Refer Slide Time: 32:33)



I cannot finish debit transactions on my node, because account 1 has been closed. The other case is account 1 have sufficient funds. Let us say it has no balance and you try to withdraw two hundred rupees from the account, when there is no balance in the account. Again agent is to say look i am not going to commit my part of the transaction because there is no way I can commit this transactions. So in all these cases both nodes have to agree both agent 1 and agent 2 have to agree to commit their transaction for this whole thing to go through.

(Refer Slide Time: 33:13)



Two phase Commit

root

Agent 1

Agent 2

Begin Transaction

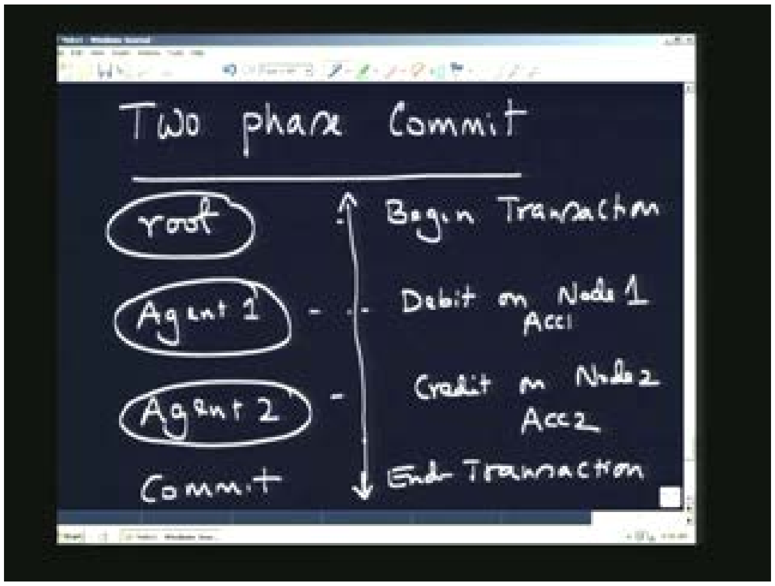
Debit on Node 1 Acc1

Credit on Node 2 Acc2

End Transaction

Otherwise, there is no point to trying through have one part of the transaction executed and the other part not been executed. Now to ascertain this, this commit protocol gets started ideally at the end transaction. For example: when you reach the end of the transaction, you have to actually now start this commit protocol to ascertain whether this whole transaction can be committed or not which means that at this point of time, all the agents involved in the transaction have to participate in this commit protocol to ensure whether they are committing the transaction or not committing the transaction that is the part of the commit protocol.

(Refer Slide Time: 33:58)



Two phase Commit

root

Agent 1

Agent 2

Commit

Begin Transaction

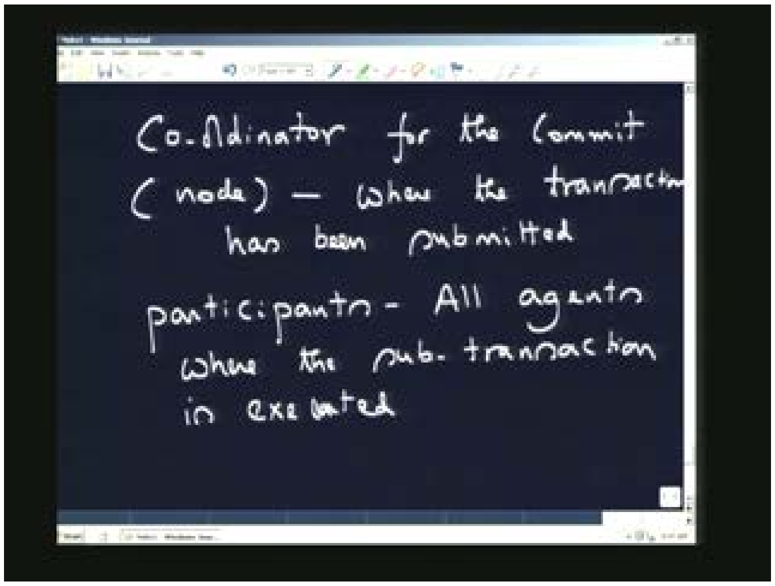
Debit on Node 1 Acc1

Credit on Node 2 Acc2

End Transaction

Now let me explain this by taking what we actually call as a coordinator of the transaction, which means that we will first fix a coordinator for the commit protocol which means that there is one single node, there is a node in the distributed system that is going to act as a coordinator for the commit protocol and ideally this is the one where the transaction has been submitted which will try to coordinate transaction has been that's what we mean by coordinator. Now all the agents where this transaction is being executed we call them actually participants. They are all participating in the commit protocol. So they are called participants. So all agents where the sub transaction is executed is called the participants where call the sub transactions is executed is called the participants.

(Refer Slide Time: 35:37)

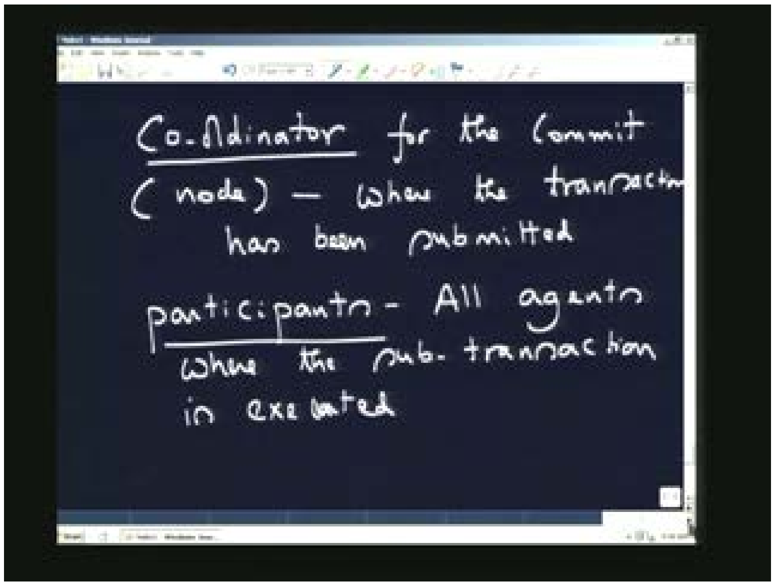


Co-ordinator for the Commit
(node) — where the transactio
has been submitted

participants - All agents
where the sub-transaction
is executed

So we have actually two things in the two phase commit. One the first coordinator and number of participants. Now the responsibility of the coordinator is to initiate the commit protocol and make sure some response from all the participants and based on to that the coordinator take a decision again inform all the participants and properly terminate the commit protocol at the end of the execution. So this is what we will call as a coordinator. Each participant has a responsibility for its own local transaction it should replying on behalf of the local agent it should ensure that certain things are done by the local agent before the participant reply back to the coordinator with a reply. Now we are going look at full detail what are step that we will be done by the coordinator and what are the steps done by the coordinator participant in the case of two phase commit protocol.

(Refer Slide Time: 36:37)

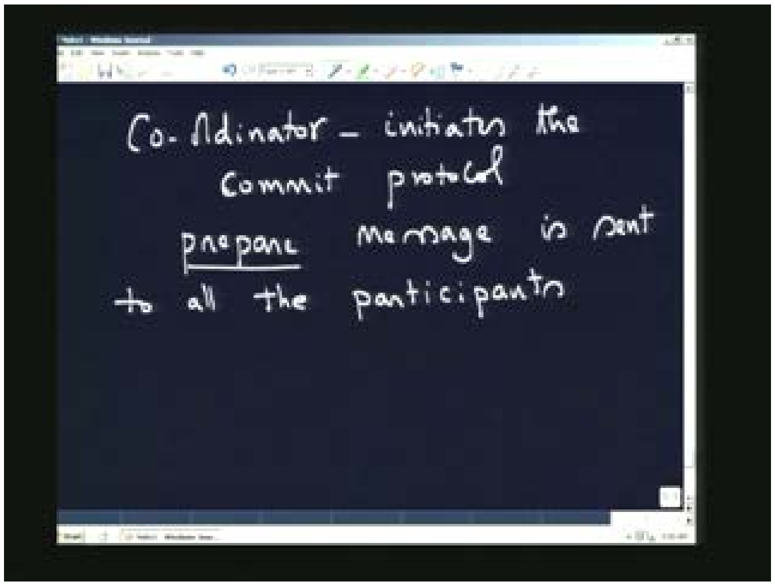


Co-ordinator for the Commit (node) – where the transaction has been submitted

participants – All agents where the sub-transaction is executed

Now what we do in the case of two phase commit protocol is the coordinator initiates the commit protocol. At the end of the transaction he initiates the commit protocol. By what we mean by imitating is he can send certain messages two various participants and make sure he get some response from this participant when he is ensuring when he is actually executing this commit protocol. Now this is the responsibility of the coordinator, to start the commit protocol at the end of the transaction. This he does by sending what he is called a prepare message to the entire prepared message. This message is a special message which is called the prepared message. He prepares the participants for this prepare message is sent to all the participants.

(Refer Slide Time: 38:09)

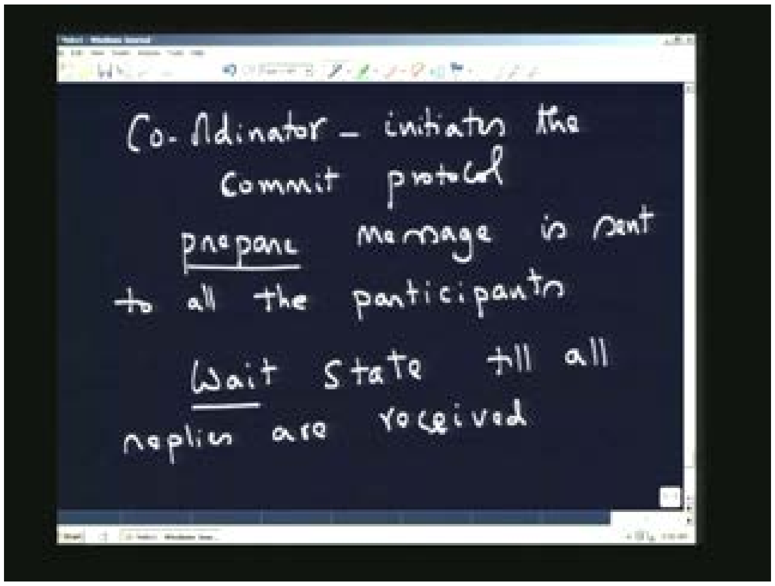


Co-ordinator – initiates the commit protocol

prepare message is sent to all the participants

Now after sending the prepare message, the coordinator enters what we call as a the wait state because still now decided on the final outcome of this particular weight state till all replies are received or some of this were basically the problem come. If there is a problem in terms of some participants not responding because there is a failure either of the network.

(Refer Slide Time: 38:42)



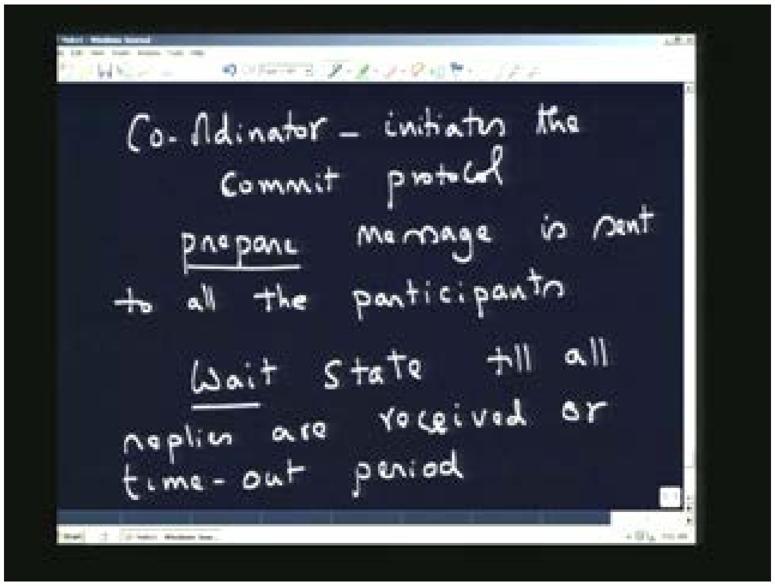
Co-ordinator – initiates the commit protocol

prepare message is sent to all the participants

Wait state till all replies are received

or the node then they are not going to respond if they do not respond what is going to happen in the coordinator. So the coordinator basically puts a or time out period actually starts a time out period immediately after sending the prepare message. Now he will wait for the time out period and if replies do not come within this time out period he is going to decide based on what replies he got because he has not got the reply.

(Refer Slide Time: 39:18)



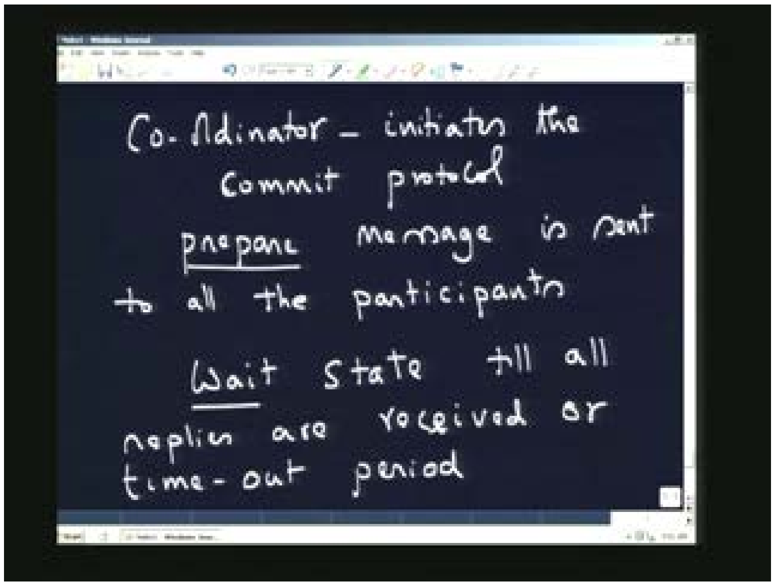
Co-ordinator – initiates the commit protocol

prepare message is sent to all the participants

Wait state till all replies are received or time-out period

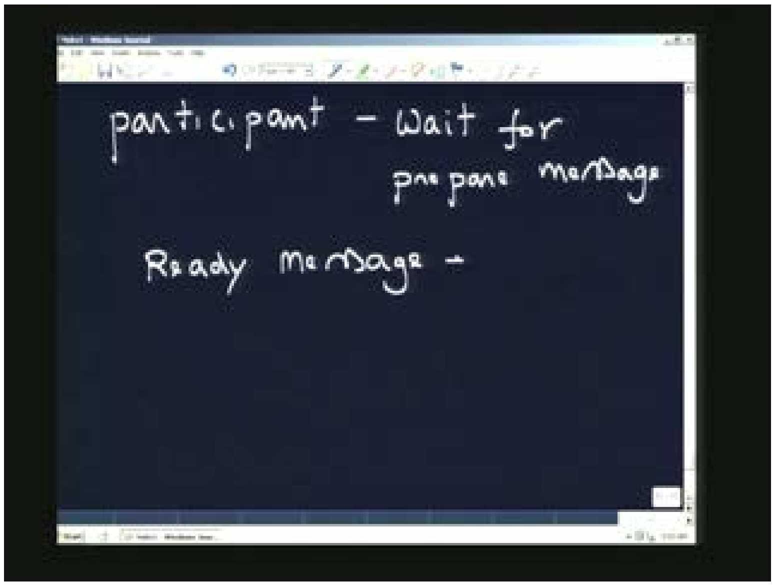
He will ensure that you know based on this, he will make a decision which actually make sure that even in that case we can still make a decision but probably he might make a decision on the negative side saying that i will abort which is the safer decision to make because when the participant comes at lateral point of time, it is always possible to ask the participants to abort rather than to commit. So you will wait for a time out period and after the time out period, he is going to make a decision relating to whether he should be committing or abort, always when time out has reached and replies are not come is going to make a decision relating to abort of the transactions rather than the commit of the transaction.

(Refer Slide Time: 40:16)


Co-ordinator - initiates the commit protocol
prepare message is sent to all the participants
Wait state till all replies are received or time-out period

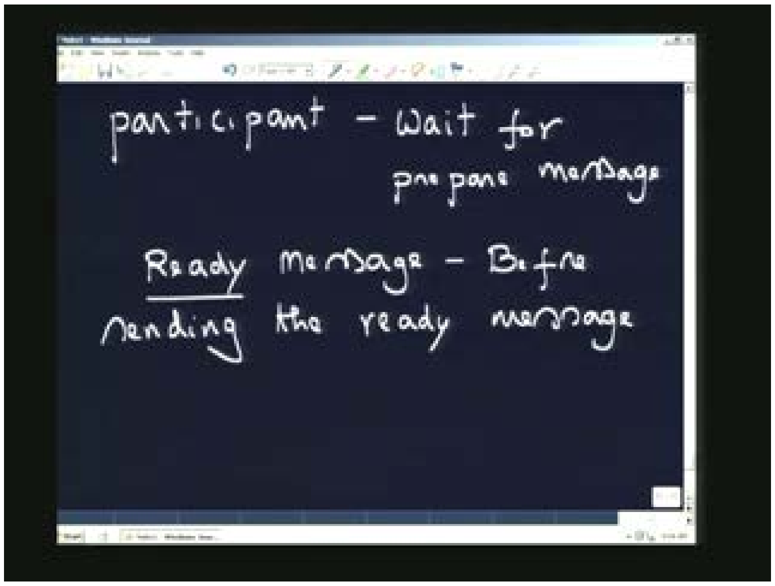
Now as far as the participant is concerned, participant actually decides now waits for the commit protocol that means to start with is in an undecided state wait for prepare message from the coordinator. Now when you get a prepared message from the coordinator. Now we have to apply the two things which participant can do, participant can actually reply with a ready message.

(Refer Slide Time: 41: 04)


participant - Wait for prepare message
Ready message -

What ready message means is that the participant is willing to commit his part of a transaction that's what actually means by the ready message. The ready message indicates that the participant is willing to commit his transaction that means he has executed successfully his part of the transaction and he is willing to actually commit his sub transaction that's what it mean by the ready message. Now, before sending the ready message, if he is willing and before sending the ready message, he needs to actually ensure that before sending the ready message what should be he doing? He should be ensuring that all the logs are return because if he does not write the locks and he replies with the ready message.

(Refer Slide Time: 41:55)

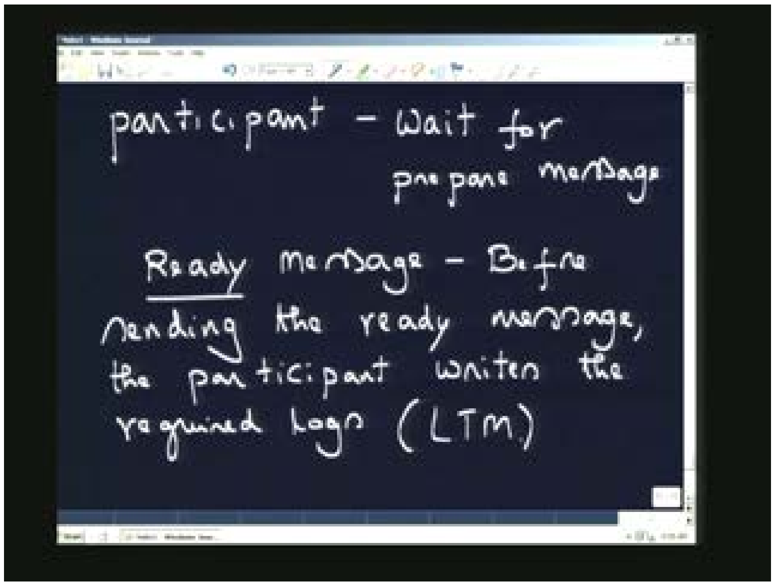


participant - Wait for prepare message

Ready message - Before sending the ready message

The coordinator assumes that whatever may happen later, the participant is willing to go with the commitment of the transaction. Tomorrow you cannot come back and say look i am not going to commit. Once you have replied with ready message, you are committing your self to commit this transaction. So it is important for you write all the logs and also if you are using locking protocol, you are not going to release the locks till you know the decision of the coordinator because if the coordinator now says please come in, you need to write all the values on to the database and then release your locks. So before sending the ready message, the participant should write all the required locks. Participant writes the required locks.

(Refer Slide Time: 43:16)

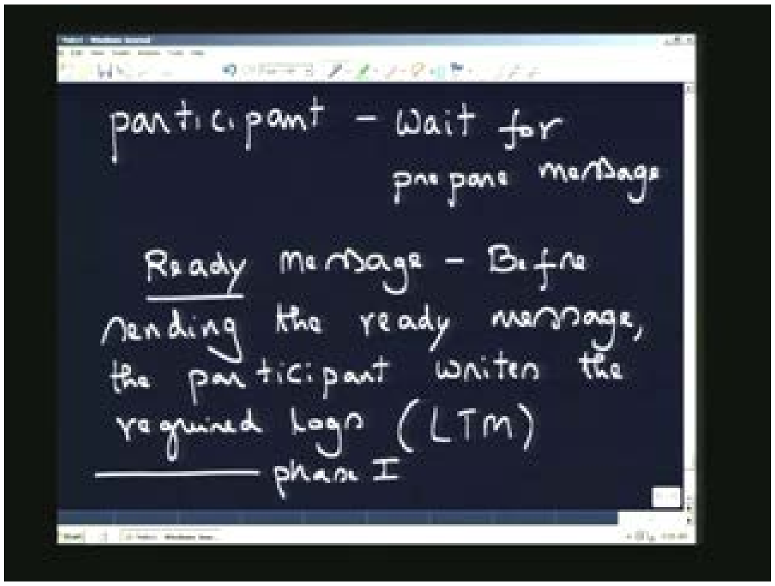


participant – Wait for prepare message

Ready message – Before sending the ready message, the participant writes the required logs (LTM)

Now, if you carefully observe all this writing the locks preserving all the different properties of the transactions is achieved by the local transaction manager. LTM ensures that all this local transactions properties are maintained using the LTM. Now once the participant it's a ready message the coordinator can now decide if he receives all the ready messages he can now decide, what needs to be done. Up to this point, we call this is phase 1 that means phase 1 is preparing the participants for commit.

(Refer Slide Time: 43:41)



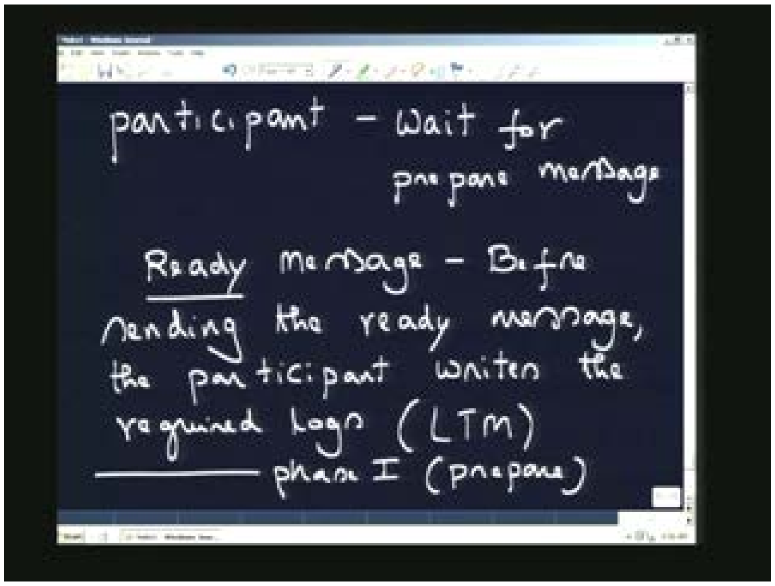
participant – Wait for prepare message

Ready message – Before sending the ready message, the participant writes the required logs (LTM)

phase I

You are preparing this is actually prepared phase. Actually, you are preparing the participants for committing only in the next phase, phase 2 you actually commit.

(Refer Slide Time: 43:52)



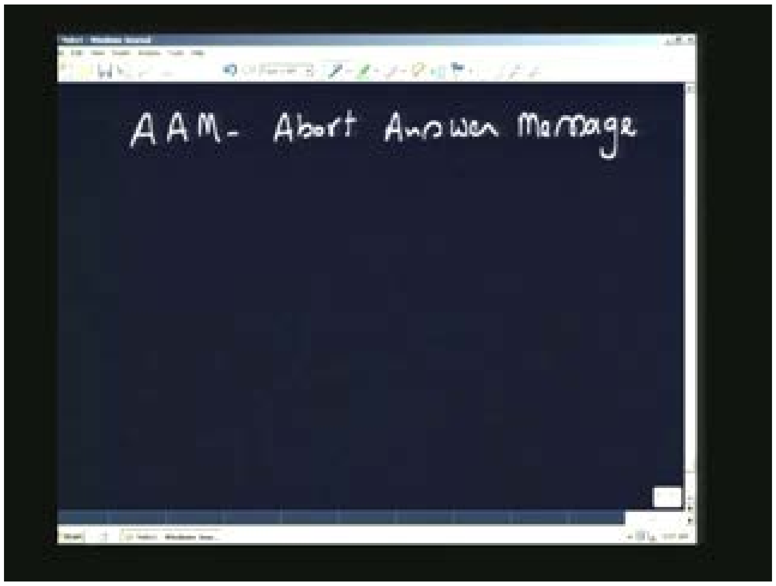
participant – Wait for prepare message

Ready message – Before sending the ready message, the participant writes the required logs (LTM)

——— phase I (prepare)

So there are two separate phases, thats why called two phase commit protocol. In the first phase, you are preparing all the participants for committing. In the second phase you are actually committing the transaction. Now it is possible in phase 1, you reply back for some reason the participant is not willing to commit because as we explained earlier, there was no sufficient fund. All these could be reasons while the participant may not to commit his transactions. In all these cases, what the participant does is, he actually sends an abort answer message which means that he sense saying that i am aborting. His answer for prepare is abort answer message.

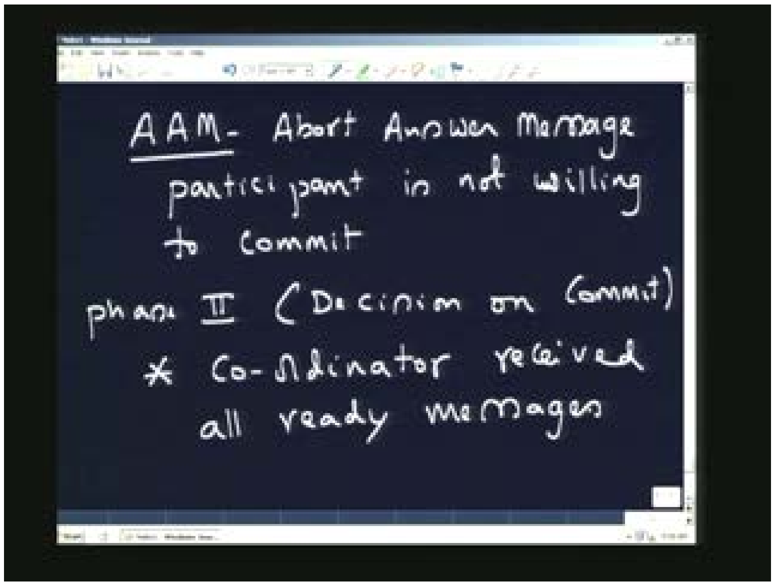
(Refer Slide Time: 44:46)



AAM- Abort Answer Message

When a participant is actually giving an abort message, participant is not willing to commit his part as the transaction, participant is not willing to commit not willing to commit when coordinator receives this abort answer message, he will be forced to take only in the second phase we are going to take that commit decision. So phase 2 once the coordinator enters phase 2, he is actually making the decision on commit. He is deciding now on decision commit he is made. Now what are the different decisions? It is possible in the time out period coordinator receive all the coordinator received all ready messages from all the participants all the ready messages from all the participants.

(Refer Slide Time: 46:01)



AAM- Abort Answer Message

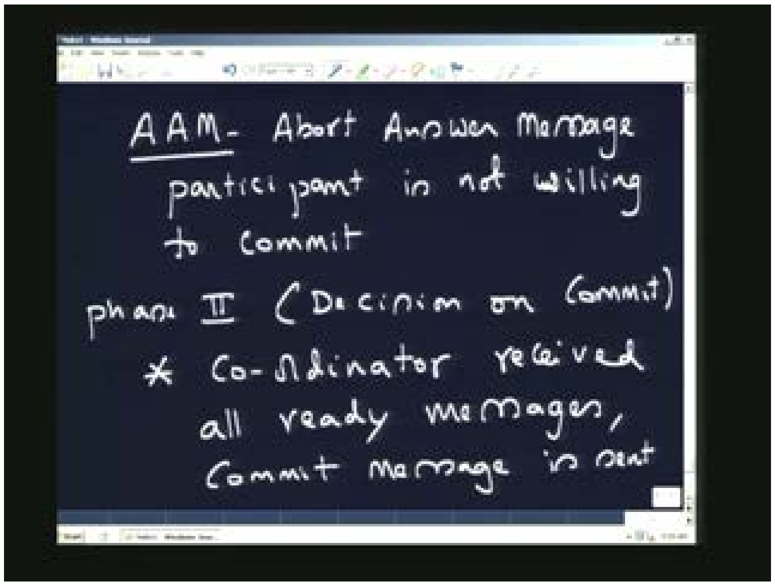
participant is not willing to commit

phase II (Decision on Commit)

* Co-ordinator received all ready messages

Then now, the coordinator in this particular case, we will take a decision to commit the transaction. That is obvious because all the participants in the transaction willing to all the agent transaction are willing to commit their part of the transaction. Hence it is logical for the coordinator to take the decision to commit in the entire transaction. But before actually we taking the commit decision okay, before sending the commit message the agent has coordinator has to write decision on a stable storage. So that whatever may happen later, the coordinator can recover back see what is the decision that was made by him, with respect to this particular transaction that is the logical thing which he does.

(Refer Slide Time: 47:10)



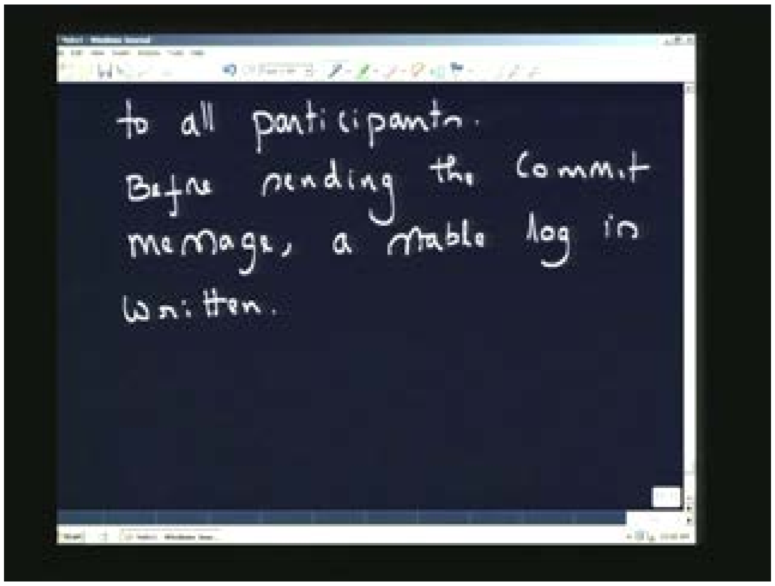
AAM - Abort Answer Message
participant is not willing
to commit

phase II (Decision on Commit)

* Co-ordinator received
all ready messages,
Commit message is sent

So, we except in this particular case the coordinator receives all ready message commit decision is taken commit message is sent to commit message is sent to all participants. Before sending the commit message, before sending these are the steps that the coordinator has to follow before sending the commit message, he needs to actually write the commit log. The commit message he needs to write a stable log is written a stable log. This is important because whatever may happen to the coordinator, he should be able to now tell the rest of the participants at later point of time, what was the decision that was taken on this particular transaction.

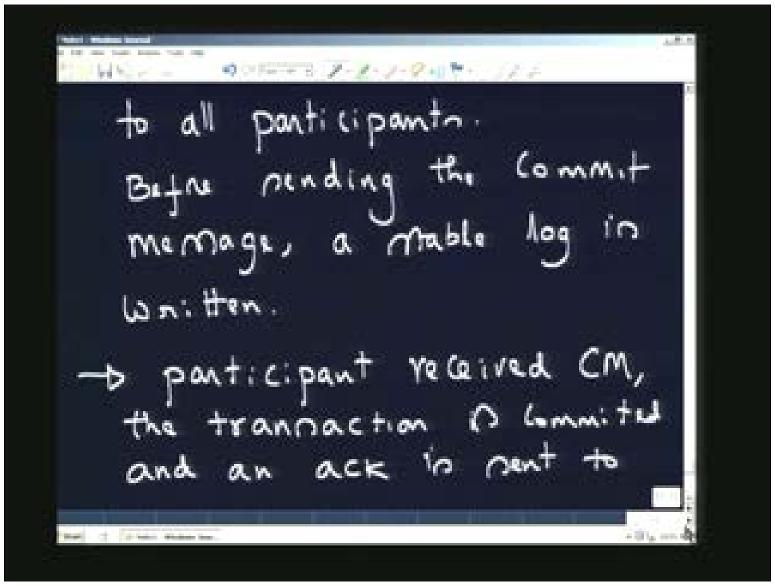
(Refer Slide Time: 48:03)



to all participants.
Before sending the commit message, a stable log is written.

When the participant receives now, at the second phase, phase 2 when the participant receives participant receives the commit message receives CM message commit message, the transaction is committed and an acknowledgement is sent to the transaction is committed and the ack is sent acknowledgement is sent an ack is sent is sent to the coordinator

(Refer Slide Time: 48:50)

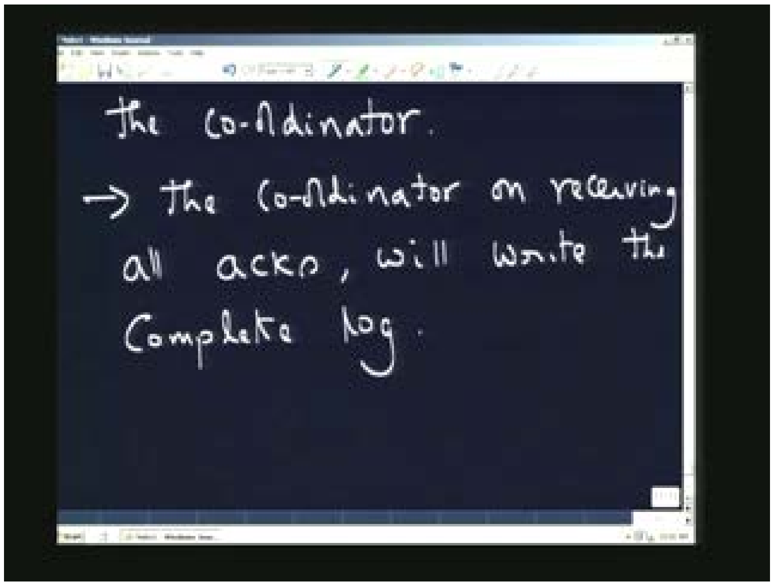


to all participants.
Before sending the commit message, a stable log is written.

→ participant received CM, the transaction is committed and an ack is sent to

Now the coordinator, when he receives in the second phase all the acknowledgement the coordinator on receiving the coordinator on receiving on receiving all ack that is acknowledgement will write complete log will write the complete log. Now at this point of this time the transaction is completed and it can be closed, successfully by the coordinator. Now there could be various cases in between.

(Refer Slide Time: 49:41)

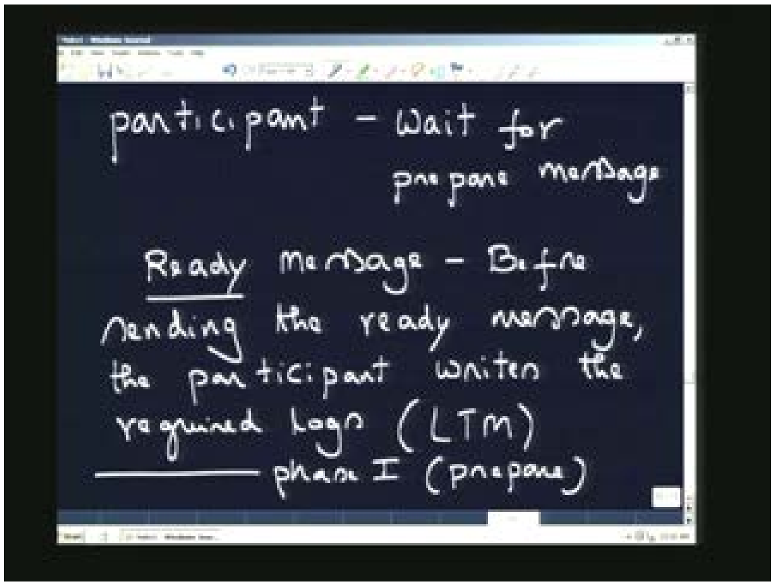


the co-ordinator.

→ the co-ordinator on receiving all ackn, will write the complete log.

This was talking about only a successful completion on the transaction by the coordinator as it is proceeding from one phase to another phase. To recap what we have done phase 1, this is actually in the phase 1 coordinator imitates the commit protocol and here he actually sends the prepare message is sent to the all participants. This is in the phase 1 first action of the coordinator and enters in to the wait state for a time out period for replies from all the participants. Now the participant who is waiting for the prepared message can actually respond with the ready message. Before sending the ready message, participants make sure he write all the stable logs and enters into a now a wait state of the decision of the coordinator.

(Refer Slide Time: 50:28)

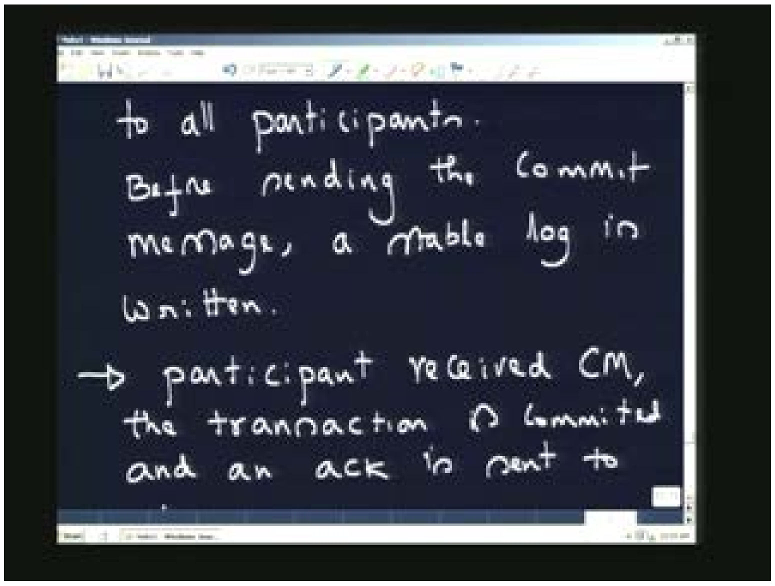


participant – wait for
prepare message

Ready message – Before
sending the ready message,
the participant writes the
required logs (LTM)
———— phase I (prepare)

Now the coordinator after this point enter phase two the phase two is entered by the coordinator when he receives all the replies from the no all the ready messages from the participant or a time out period as a write. Now in either case it takes a decision if all the messages are received ready messages are received he commits the transactions writes the commit log and sends the commit messages to all the participants. Now when the participant in the second phase and receive the commit message, they actually write all the transactions are committed and an acknowledgement is sent back to the coordinator.

(Refer Slide Time: 51:25)



to all participants.
Before sending the commit
message, a stable log is
written.

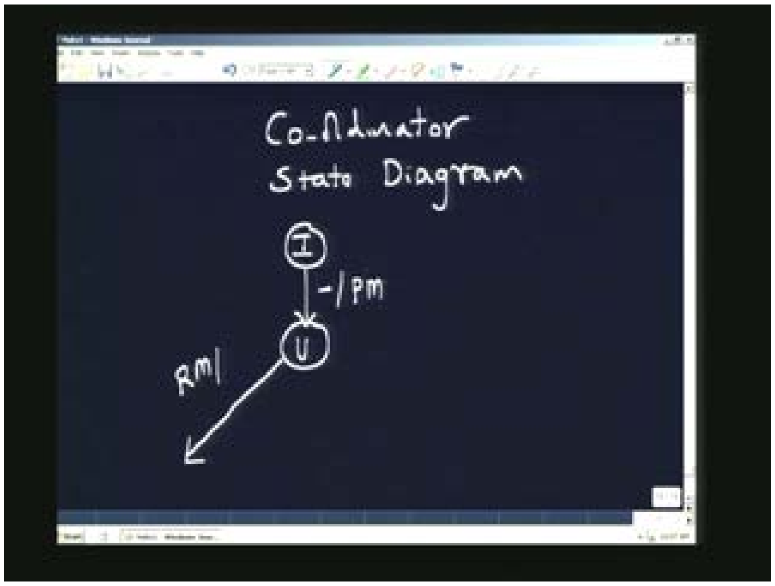
→ participant received CM,
the transaction is committed
and an ack is sent to

So that now he can write the complete log and close the transaction. Now it is possible that in the first phase. We did not discuss the case were its possible for the participant to reply with an abort message. Now when the participant replies with an abort message or coordinator waits for an time out period, in either of its cases the coordinator is going to take a abort decision and this will be communicated to all the participants and when receive the abort decision, the participants are abort the transaction. This is what actually happens with the two phase commit.

Now before we going to in details of looking at what are the different things can happen what are the different failures scenario that can be presented in this case and how this two phase protocol is resilient that kind of failure. We will try first looking at what was the state transition diagram for the coordinator and the participant, that explains this whole scenario in lot more detail and after that we are going to look in more details, how the two phase commit protocol is resilient to certain kind of failures when the commit protocols is executed.

Now, to just explain this, what we are going to take is, this is actually the coordinator state transition diagram. Coordinator state transition diagram. I am writing the state diagram of the coordinator to start with. To start with, the coordinator is based on the initial state and from the initial state what the coordinator does is he actually sends prepare message to all the participants. So this actually we will have some level of undecided state. This is the state where the decision was not taken by the enters its wait state which is the undecided state. Now this is basically the prepare message is sent by the coordinator and he is waiting for the decisions. Now its possible that actually he waits from this state possible for the coordinator from this undecided state that he receives the all the ready messages this is what we actually seen. Now when he gets all the ready messages, he is going to take when the input to the coordinator is ready messages.

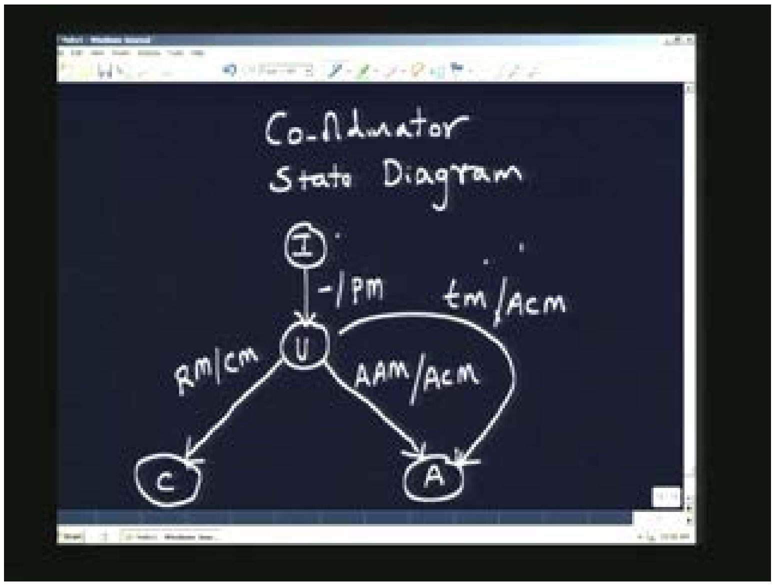
(Refer Slide Time: 54:27)



All the ready messages have been sent to the coordinator, then you are going to get the commit command commit message from the coordinator and the state in which actually the coordinator reaches the state the commit state. Now it is possible that the coordinator actually got abort answer message from the participants when he got an actually abort answer message and he is going to taken an abort command message that means is going to send an abort command to all the participants and that is typically when an abort decision is taken by the coordinator. Now it is possible that, when from the initial state the coordinator can reach the abort decision if there are no sufficient replies, that is the time out period when a time out period is reached.

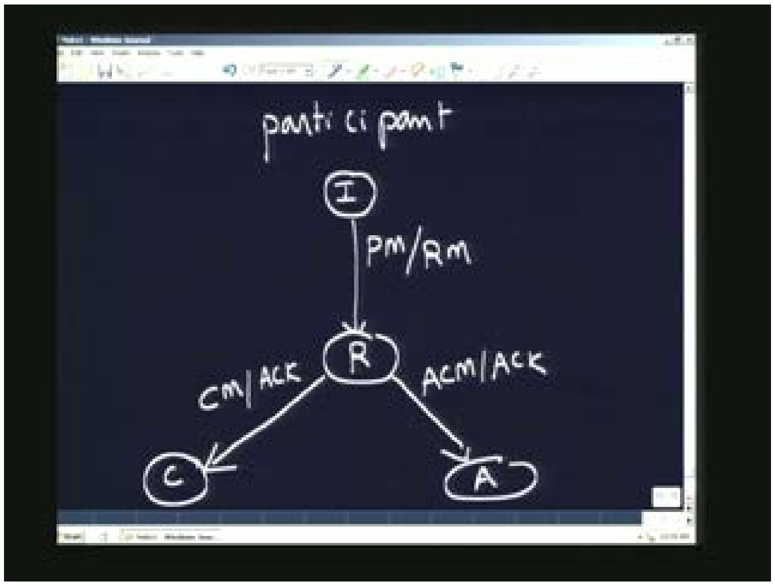
When the time out period is reached its possible for the actually by sending the prepare message and gets to the undecided state. So strictly speaking, this arrow should come from the undecided state here which means that the this state from which this arrow will come, there is a time out period and after the time out period, the coordinator has to take an abort command message that he waits for the time out period and after the time out period he actually takes the decision to abort the message.

(Refer Slide Time: 56:09)



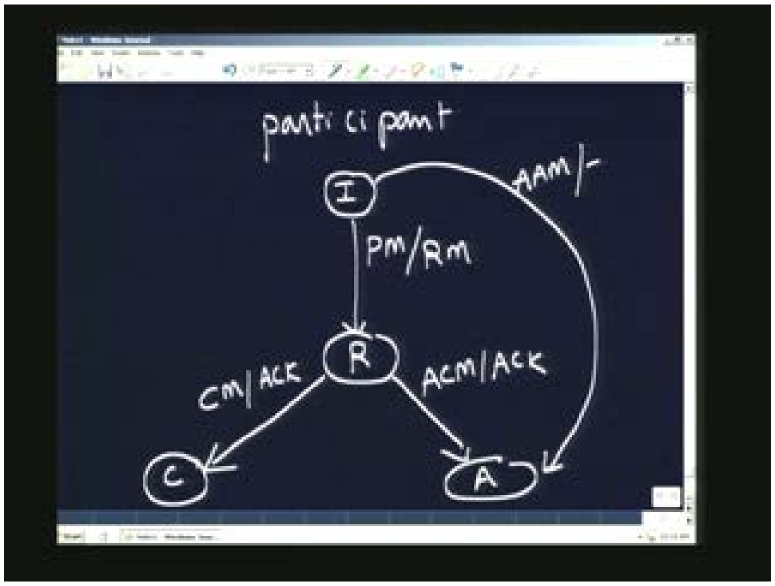
Now if you look at the participants' state transition diagram, it looks something like this. To start with actually the participant is in initial state. Now the participants receives the prepare message and the answers with the ready message then he basically enters the ready state here. Now from the ready state, it is possible for the participant to reach when he reach when he receives the command message, he could be receiving the he could be sending an acknowledgement message and could be reaching the commit state. If he actually receives an abort message, abort command message, again he will acknowledge, but he will reach the abort state.

(Refer Slide Time: 57:09)



It is possible for the participant from the initial stage itself. By actually responding with an abort answer message he could be reaching this state. He actually sends an abort answer message, with which he actually reaches the abort message.

(Refer Slide Time: 57:27)



This sort of summarizes what we are doing now in two phase commit protocol. What we are going to do in the next class is, take these two transition diagram and then explain how the two phase commit protocol is resilient for failures.