Database Management System
Prof. D. Janakiram
Department of Computer Science & Engineering
Indian Institute of Technology, Madras
Lecture No. 23

Concurrency Control – Part -4

In the last lecture, we have been looking at time stamp based concurrency control techniques. What we have been looking at in more particular is, how the basic time stamp mechanism works in terms of ordering the transactions, in terms of its time stamps. We also looked at how the basic time stamp protocol can be integrated with commit protocol. Towards the end of lecture, we saw how the basic time stamping techniques can be integrated with the commit protocol. We just recap what we have been doing there.

We introduced instead of write, a pre write and then ask the transaction to issue a pre write instead of write to start with, and pre write is going to be buffered and any read transactions incoming read transactions will be checked against this pre write to see if they need to be buffered or they can be they can be satisfied, the read transactions can be satisfied. So what we have done in the last lecture? We also mentioned that in the last lecture, basic time stamp mechanism is not truly optimistic concurrency control algorithm because we are not actually looking at looking at validation. At the end of transaction execution, what we are still doing is we are ordering the transaction as they enter in to the system rather than looking at the end of the execution whether what they have done is truly satisfies the consistency requirements.

What we are going to do in today's lecturer is, to further continue looking at these models of concurrency control and see a slightly different kind of algorithms which include the truly optimistic version of a time stamping algorithm. To start with, what i will do is, i will look at a time stamping algorithm, time stamping based protocol that is truly optimistic in the sense that the validation will be done at the end of the execution of the transaction and we are going to look at in depth that particular protocol and also we will look at another different approach to concurrency control which is actually the multi version concurrency control algorithms.

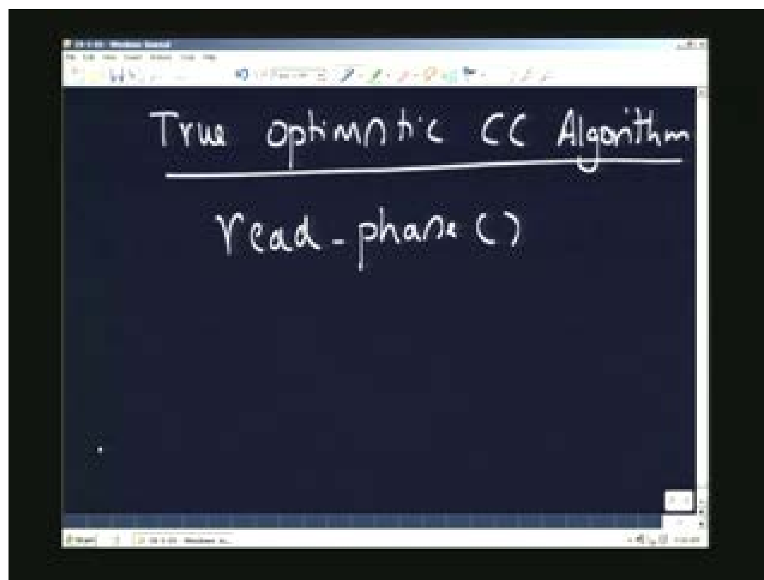
The multi version solves the problems of concurrency control by producing a new version of the data item, each time you write the value which means that the old value is still preserved when you are actually producing a new value further data item and that's solves the concurrency control in a completely different way. For example; actually each time you try to do something since preserving a old value a large extent a problem of concurrency control is elevated by maintaining a multiple copies of the data item, but a consequent problem is that you will end up paying a huge overhead for this space. A completely degenerated case is where you not only maintain the new version but also the time at which this was done.

This typically comes down what to see as a temporal database because we actually record when a value is changed not only the changed value, the new version of the changed value but also the time when it actually happened and that becomes what we call as temporal database. Concepts of temporal database are suddenly beyond the scope of this particular series of lecturers, but I encourage you to read the material on your own to understand. After i finish this I will encourage you to read a little bit more on your own on temporal database which constitutes a very important and interesting aspect of databases by itself. As we go further down, what we are going to do is, we will start our discussion by first looking at a truly optimistic time stamp based protocol to start with and then proceed on multi version protocols.

I am also going to look at multi version based concurrency control algorithm and figures them, version of the multi version two version protocols in the context of two phase locking and we were going to look at in depth. In the next class what i am going to do is, i am going to review some of these things that I have done in last lecturers with set of questions and then giving more explanation of what was actually done, by looking at a series of questions review questions that we can attempt on the last seven lectures. In the seven lectures, we are going to have review questions. In the next class, we can be prepared on the things we have done so far. So that, you will be able to look at the review questions more carefully at the end of the seventh lecture.

Now we will start with the optimistic protocol. Now a truly optimistic protocol as we were taking about we have the approach to looking at the problem of concurrency control, the CC algorithm will have the approach of actually doing the checking at the end of the execution of the transactions. Now every transaction in this case, can be thought of as having several faces. The first phase can be the read phase which means that the transaction typically reads whatever is needed by it.

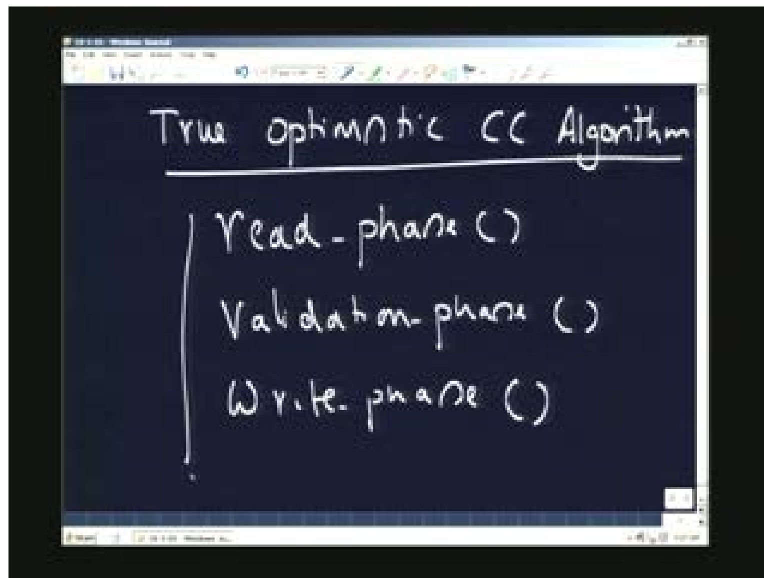
(Refer Slide Time: 7:56)



All the data items that are actually required by it and then it also manipulates them, but only one thing is does not actually write them back on to them database till such point actually the transaction gets validation. So you have an extra phase here which is called the validation phase. Now the validation phase make sure that different transactions if they are conflicting with each other enter in to the validation phase and they get validated and subsequently enter after validation phase in to their write phase, which actually means that we have these three phases; a read phase in which the transaction reads all the values and then you have the subsequent validation phase after the transaction finishes all its required things, it gets in to the validation phase.

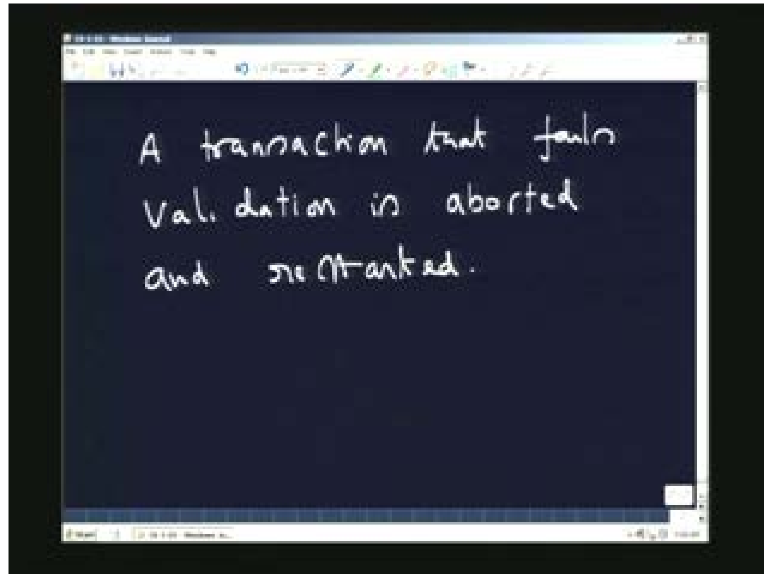
In the validation phase, any conflicts are detected and make sure that if the transaction passes the validation phase, it is in a consistent fashion whatever is trying to do and after it enters the write phase it is allowed to write the values of the transactions on the database. Now if the transaction does not pass through the validation phase, it is actually the transaction does not pass through the validation phase is actually aborted.

(Refer Slide Time: 9:20)



The transaction that fails validation, validation is aborted and restarted. Now it can be seen in this particular case.

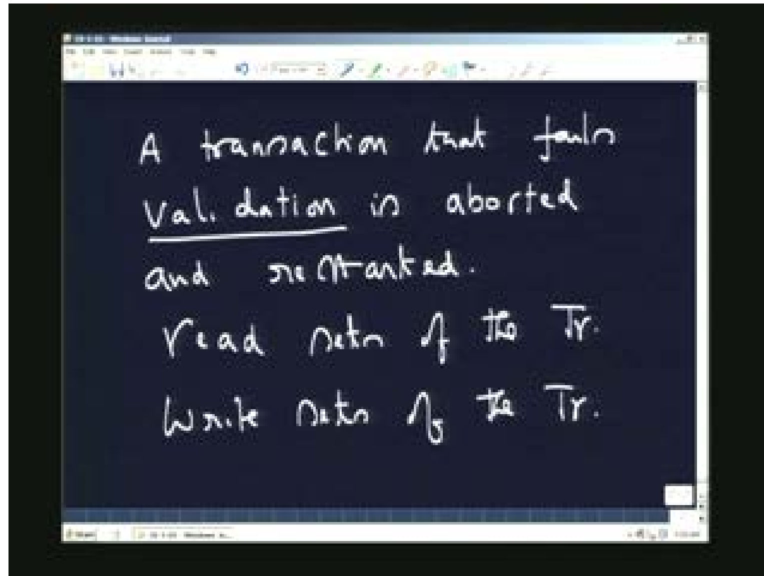
(Refer Slide Time: 9:54)



What is happening really is the transactions just enter the systems, they read in the read phase they take all the required data items. They do whatever they need to do and they come in to the validation phase. When they come in to the validation phase, the system checks for any possible conflicts between the various transactions and once the transactions passes the validation phase, you enter in to the right phase into the right value. Now the important requirement here is validation.

How does the transactions gets validated when they are conflicting with each other? To ensure this, what we will do is we will essentially look at the read sets and the write sets of the transactions. Read sets of the transaction will agree with Tr and the write sets of the transaction. Now this is the important requirement because at the end of the day, at the end of the transaction execution, what we essentially do is, we look at the read sets

(Refer Slide Time: 11:09)

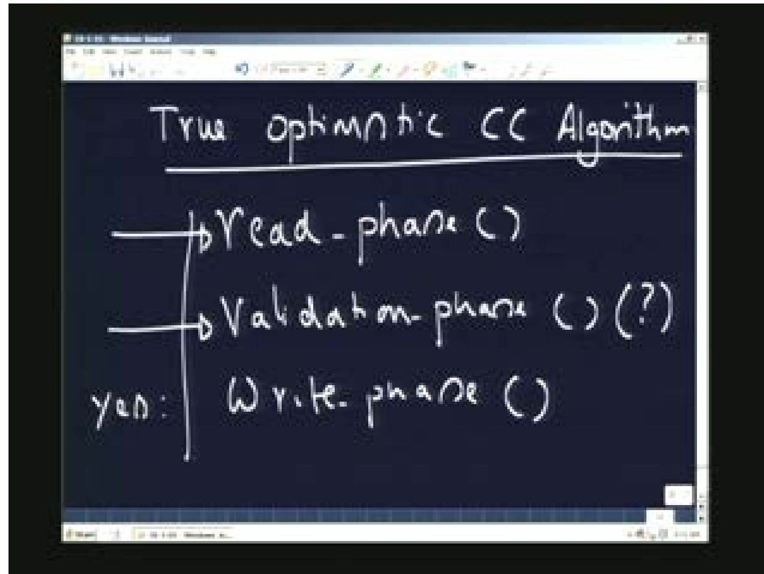


and the write sets and decide whether there is any conflict and based on that we are actually going to decide whether the transactions validate against the each other or there is a possible inconsistency produced by the value transaction and that is how exactly, the transactions are allowed to commit or mean to abort come back and do the redo whatever they done in earlier. This is truly optimistic, because what we are doing in this particular case as you can see here is, we are actually allowing the transactions to go through this read phase irrespective of whether they validate or they do not validate only when they come to the validation phase, we perform this check.

Once they actually validation is performed and it passes through the validation phase and it actually enter in to the write phase depending on the answer to this yes or no. If it is yes basically the transaction enters the write phase and tries to write the value on the database. So this is how exactly as you can see the validation, since it is performed in the end of the execution. This is a truly optimistic concurrency control to this how exactly the system proceeds to execute. On the other hand, as we saw earlier the pessimistic case the validation is done at the beginning of the transaction execution, not at the end of the transaction execution.

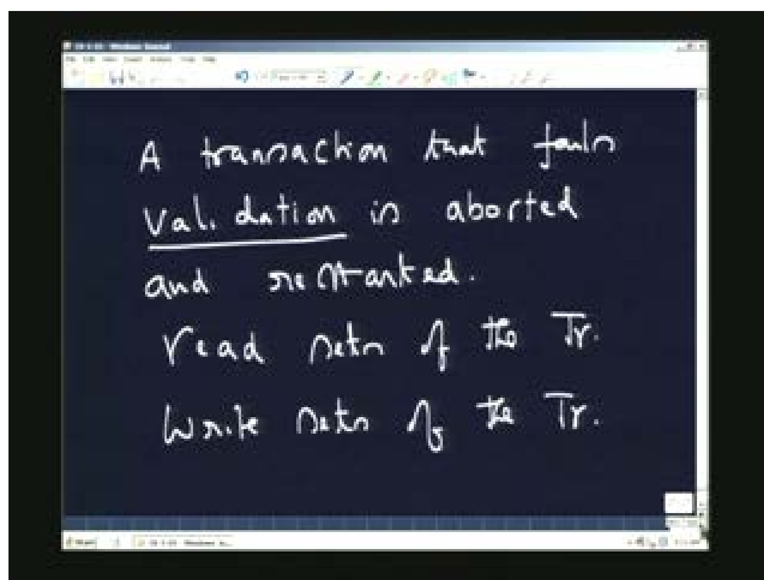
For example: Even before the transaction is allowed to read it needs to acquire to corresponding the logs. Only then, it will be allowed to proceeding to other phases which means that it will be blocked at the beginning. It will never be asked to abort after it is actually got the logs for other than actually deadlock or something else happens in the transaction aborted for those reasons.

(Refer Slide Time: 13:03)



Otherwise the transactions once it get the locks, will proceed to execute finish its results write the values and then only it releases the logs. When in this particular case, the transaction starts executing to start with and it get validated at the end of its execution to see what it has done is correct or not and it is aborted if it has produced wrong results and it is made to redo the things again. So we have actually a completely two different approaches to solving the same problem. Now let us understand what exactly happens, if you apply the true optimistic concurrency control. Now one of the things that we will be doing here is we need to record though we have looked at three phases.

(Refer Slide Time: 13:54)

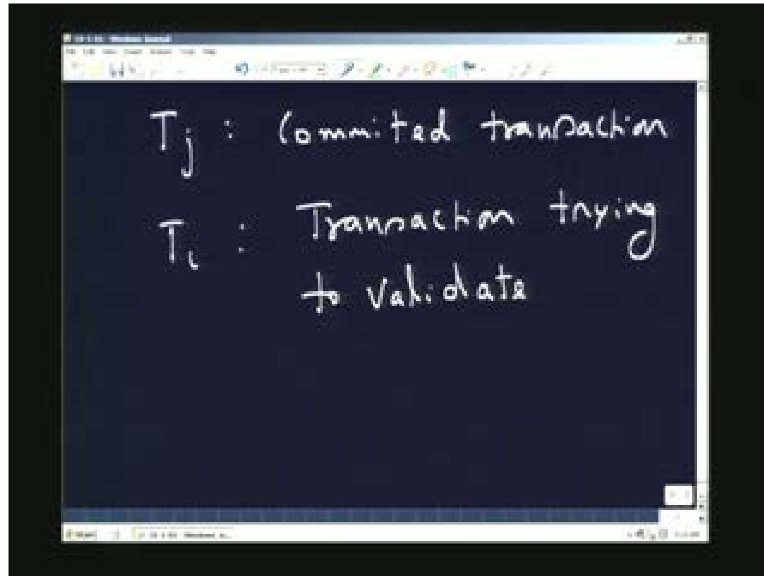


We will be needing lot more information to see whether the transactions validate or not or they do not validate. To explain this what we will what i will do is i will give intuitively give you what happens when we see algorithm and then go into details of the algorithm physically. Now as you can see if there is no read if read sets of the transaction do not conflict with each other in which case, it does not really matter how they actually went about doing their activities. If there is a write set conflict, write conflict or read write conflict, then I need to worry about how to order these transactions. Now what I look at is. For example: you imagine, there is somebody in front of me who has actually done something database. This is what we called as the committed set.

Now this committed set has gone ahead something on the database before actually I came try to do something. Now I need to worry about, what is the thing this committed set has done and see. If I actually conflict them respect to the committed set which is equivalent to say that this is Tj which has got committed. Now you have to realize that I have to maintain information about the current committed transactions, what all the data sets? What is that they have done on the database? Now i will take a look at the committed sets of the database and see with which are those committed sets? I am actually conflicting to see if I can validate again as well. Now if there is conflict between the committed set and my set.

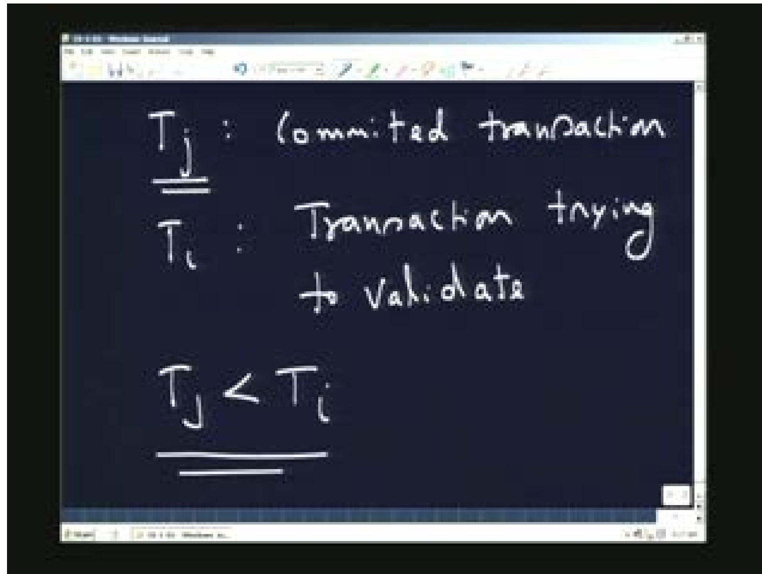
I am the current transaction which is trying to validate. Now the committed set has to see whether there is an overlap between the read and write sets and there is no overlap between the read sets or write sets depending on that you can decide now what exactly has happened before or after and based on that you can say whether i validated against them or not. A simple case is, if there is a conflict in the sets you have to make sure that a strict ordering is actually ensured between the two transactions that conflicting against each other and you progressively relaxing this requirement of the phases one before or one after depending on how much conflict is really existing between these two transactions. Now we will start explaining now with this background what exactly is done to see where these conflicts are coming, how we can look at validating the transactions. Now to give this explanation we will take simple example will start explaining how the algorithm works. As I just explain what I am going to do is I will see I will maintain. For example; Tj is the committed transaction. We will take this as a committed transaction. Now Ti is the transaction trying to validate, transaction trying to validate now okay.

(Refer Slide time: 17:53)



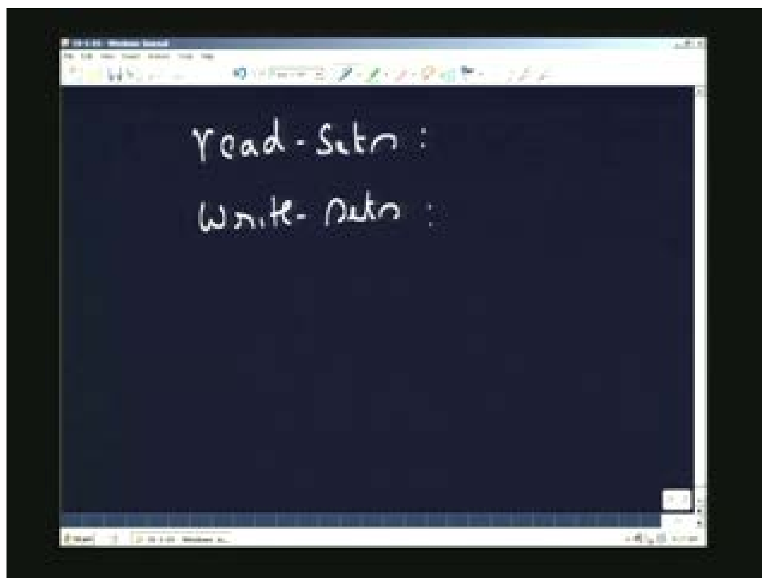
Now what I am going to look at is how exactly the Tj and Ti sets u know read and write sets of Ti and Tj really conflict with each other and based on that we have to see whether Tj should come before or Ti should come. In this case it is already clear that Tj has come before, so the order is fixed. Since the order is fixed what i am going to do now is, since this order got fixed because Ti Tj is already committed. Ti's all operations should be enforcing this requirement that it comes after Tj. Now to what level Ti should be coming after depends on the conflict. For example: if there is non-conflict, it does not really matter how Ti actually work with respect to read and write phases, but as you start seeing that is more and more conflicts in the system. More and more conflicts in the system, you have to worry about how exactly this ordering is done.

(Refer Slide Time: 18:27)



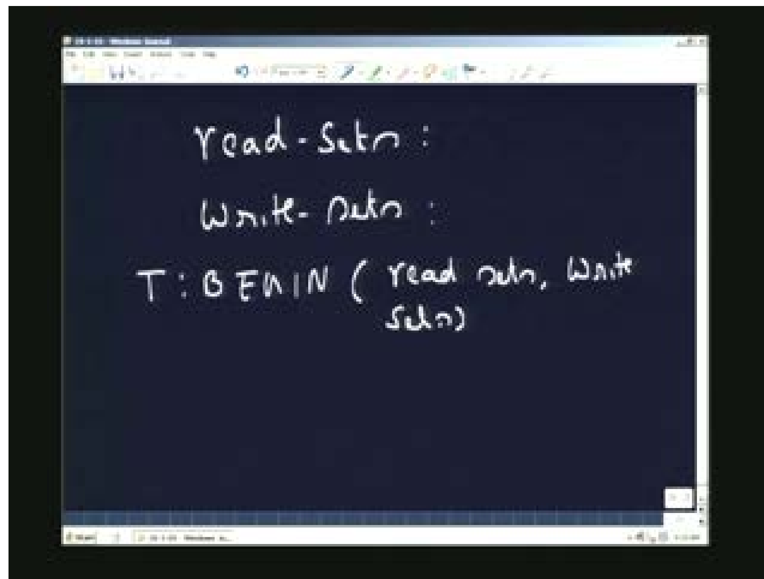
Let us see now, how exactly this how can be further, there is read sets and there are the write sets. The read and the write sets of each of these transactions have to be now looked at, this is the first thing. Now the requirement that I need to know read sets and write sets of the transaction is the strict one.

(Refer Slide Time: 19:30)



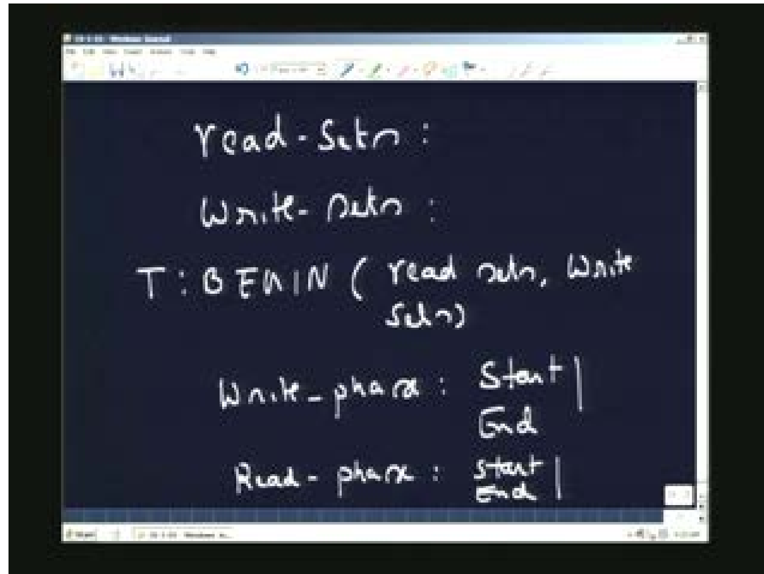
In fact, this often causes problem because this is the restriction that requires that we know a priory what are the read sets and write sets of the transactions. Some level of pre-processing will be needed for us to be generating the read sets and write sets. This is the first thing that happened in the optimistic concurrency control. Some algorithms other than optimistic algorithms also require that I know the read sets and write sets of the transaction is a strict one, that will be the assumption that means for example, if i write my t begin and at this stage i actually gave what are the read sets and write sets of the transaction. Now one way is, the user gives the read sets and write sets that were the transaction writer grammar gives the reads and write sets.

(Refer Slide Time: 20:31)



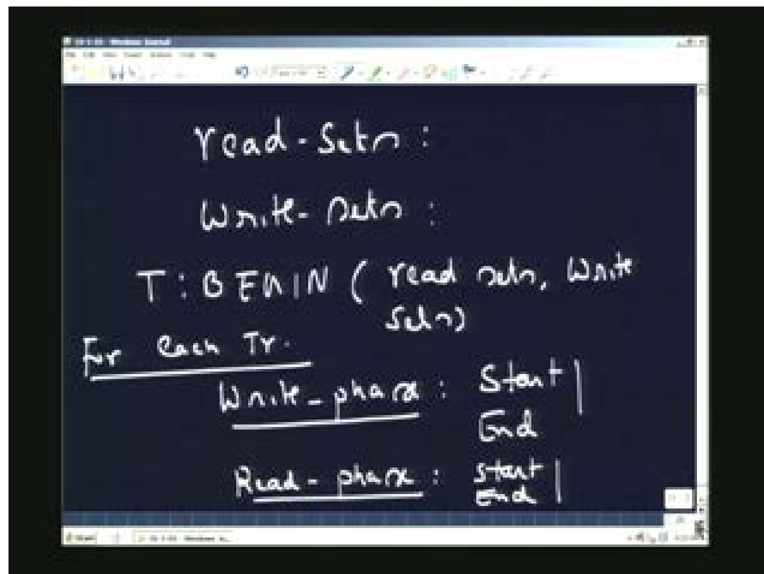
The other one is the compiler can look at and actually generate these reads sets and write sets. If the compiler is generating the read sets and write sets, suddenly it is going to be superset of the reads and writes that actually might get executed when the transaction is executed. I will not be knowing which are all the data item that I will be needing I do a particular transaction. So I parse it actually a start it time, I might indicate a superset of the set of data items i might actually access, when I go to the, when I actually execute my transactions. That is what see in terms of the reads sets and the write sets. The other thing that we need to maintain as part of this is what we see as the write phase. When it has actually starts and end of the write phase. Start and end of the write phase. Similarly in terms of read phase, we have to see the start and the end. These are the times which we need to maintain each for these phases because based on this we are going to now say

(Refer Slide Time: 21:58)



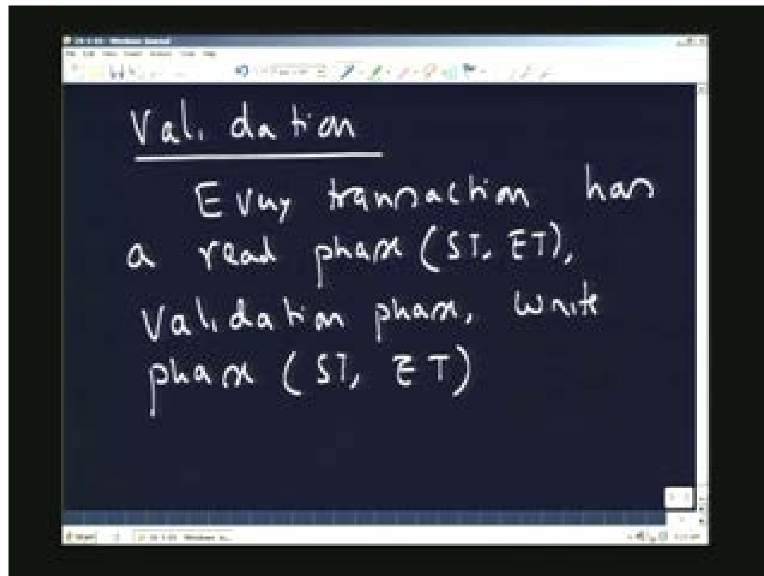
whether the validation of Tj coming before Ti is true or not and that is the reason why we need to maintain for each transaction, the write phase for each transaction we need to do this. Each Tr will be waited as transaction, we need to look at the write phase and read phase starting time and the ending time for both the read and write phase. Given this, now let us look at what really we will we need to do in terms of validating these transactions.

(Refer Slide Time: 22:16)



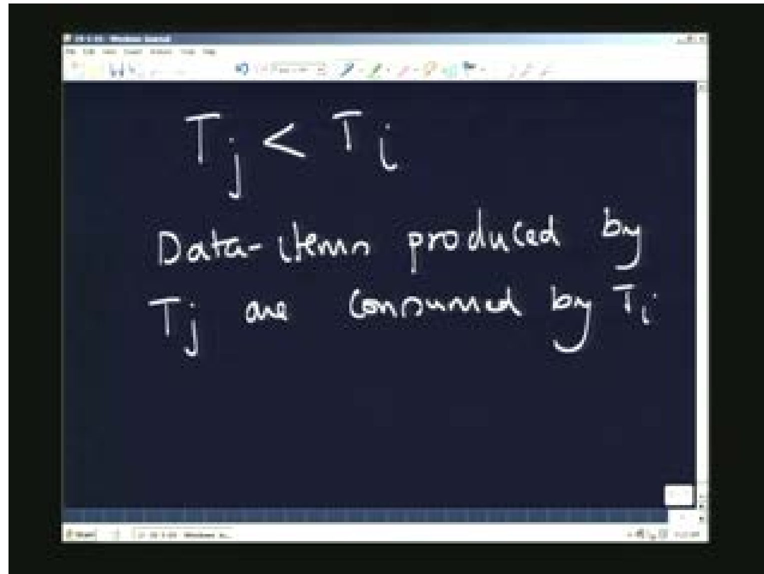
Now every transaction, as it is explained will have three phases. Every transaction will get into three phases here. Every transaction has a read phase. Now a read phase is marked by a start of the read phase, end of the read phase, there is a start time and the end time for the read phase. Then it enters the validation phase and then we have the write phase. Again we have the starting time and the ending time for each of the transactions. Now this is maintained as part of the execution.

(Refer Slide Time: 23:20)



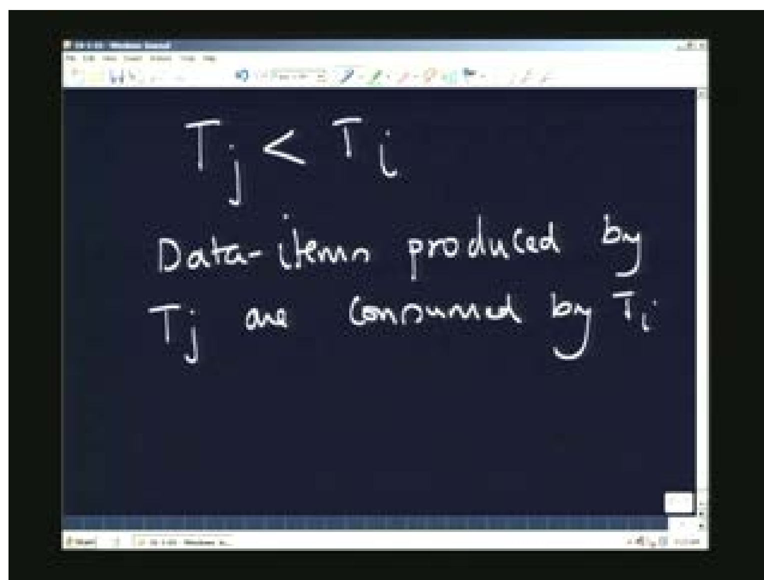
So for every transaction when the read has actually started, when the write has actually started when the read has actually ended, when the write has ended. Now if you understand now the criterion that Tj should be coming before Ti. Now, if there is a true conflict between Tj and Ti which means that there is basically some data item which are produced by Tj have to be read by Ti, which means that data items produced. What we mean by data items produced is, the value of this is return by Tj produced by Tj or read by or consumed by Ti consumed by Ti. Now it is very clear that unless the write phase of j is finished, the read phase of i should not have started.

(Refer Slide Time: 24:14)



If it starts early, then it means that this equation. The equation we are looking at that Tj should have should have lesser you know in terms of time order should be coming before Ti would not be valid because Ti would have read values not produced by Tj but by some other transactions. This is very important to understand. So, for this to be validated conflict data item between Ti and Tj. In terms of Tj write some values Ti reads some values which equivalent to conflict.

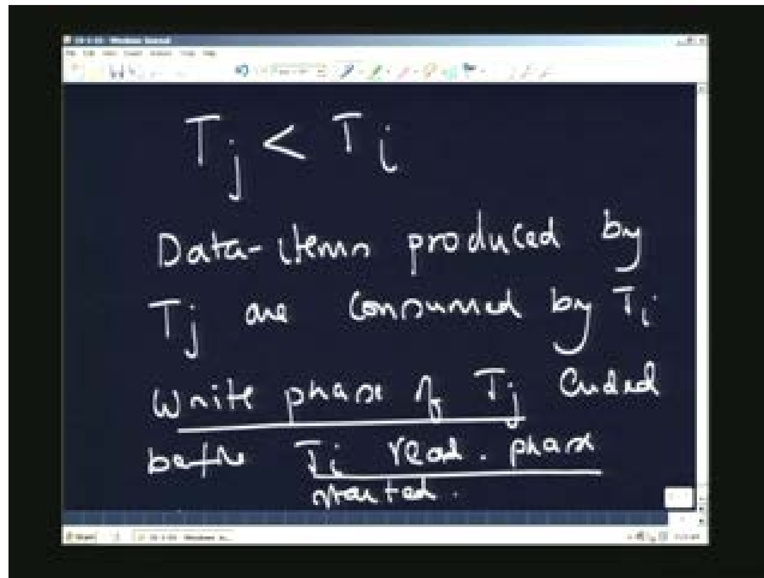
(Refer Slide Time: 24:52)



There are common items between the write item of Tj and the read set of Ti. In which case, we need to ensure that this criteria that is enforced is write phase of write phase of

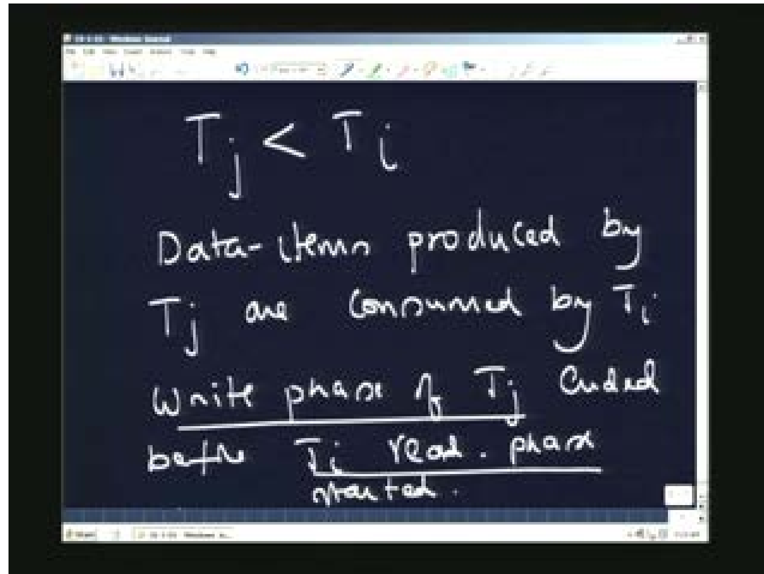
write phase Ti of Tj ended before Ti read phase started, before Ti read phase started. What I am trying to explain here the meaning of this is, Ti read phase would have read the values, Tj write phase would have written the values which means that the write phase of Tj ending before read phase of Ti ensures that Ti read the correct value produced by Tj.

(Refer Slide Time: 25:45)



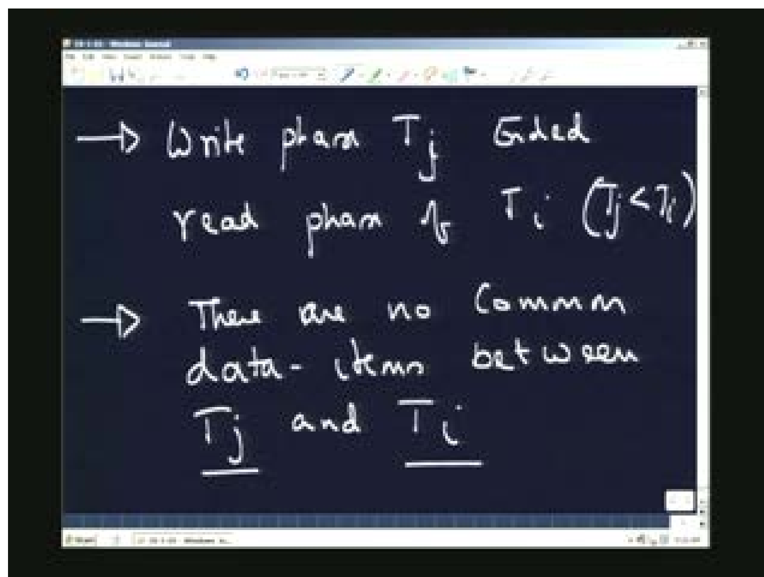
Since Tj is already committed. This ensures that Ti is valid in the case where Tj is committed and Ti is read a value before Tj is committed that means the write phase not ended which means that Tj still not return the value but Ti start reading those values which means that conflict has not been resolved properly. This will not be valid in which case, this condition that Tj is coming before Ti cannot be valid if this is not true. To what we will ensure is first condition, what we will ensure is that as I said,

(Refer Slide Time: 25:57)



The write phase of this is the true Tj ended. This ensures that before the read phase of Ti. This ensures that Tj is strictly before Ti because Tj has finished all its work, then Ti is coming. The other condition where there are common data items between the two write sets. Progressively we can relax at the end of the thing we probably can say that there are no common data items. There are no common data items before data items between Tj and Ti. What does this actually mean? This means 'Tj' is working on a separate set of data item.

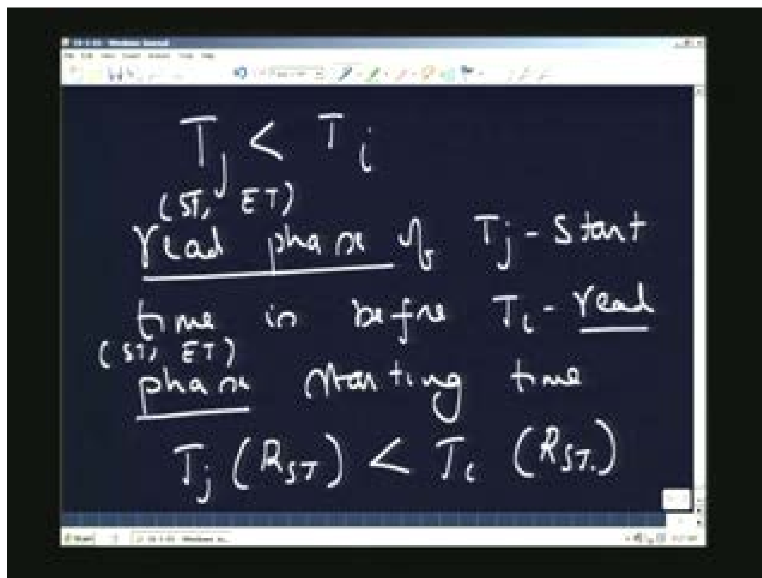
(Refer Slide Time: 27:41)



Ti is working on two separate sets of data items. In which case, for the condition Tj comes before Tj will be notionally correct all that i need to do is their read phase of Tj read phase of Tj know start, start read phase of Tj the starting time is before the Ti staring time. Ti read phase staring time read phase starting time okay. What does this actually mean? I will try to slightly rewrite it for you so that, you understand the meaning of this. Read phase always has two times. Remember that read phase has a start time and an end time. Now that all I compare to that is start time of Tj and then i need to actually compare the Tj's read phase.

Again this will again compare have start time and the end time. Now all the I have to do is Tj read start time. This is what i am looking at is less than Tis read start time read phase start time, because since they do not conflict, it does not really matter, how they actually got executed but Tj to come before Ti all should it done is, it read the data items from the database before Ti. Take this condition satisfies, then it find with me that Tj is committed coming after that but all my reads happened after Tj has actually finished its read phase. Now this is what is exactly done in terms of you just recap what we have done in this algorithm. All that we are doing is, we are ensuring that every transaction goes to three phases.

(Refer Slide Time: 29:29)



The read phase, validation phase and a write phase and in the validation phase, we are actually looking for this transactions conflicts with any previously committed transactions and the validation phase ensures that whatever has been done has validates again and regerates reads and writes gets validate again the committed transactions and that ensures that whatever once the validation phase is crossed the transaction, it can safely go and commit. All other future transactions that come now can validate as transaction right. Now the most interesting fact here is, for example: imagine there are two vehicles coming in to the IIT. If you know, how the IIT Chennai is organized. There

is only one IN gate to which all four wheelers can get in to the campus. Now let us say all of them are raising know to go in to some program in our open air theatre or student activities center. Here there will be a limited parking lot. There only limited space in the parking lot. Now where do we enforce know in terms of who will actually win, in terms of putting his car in the parking lot.

One thing is to say that, I know my parking lot will take only two hundred cars, then I say that when I enter to the IIT IN gate, I start giving the numbers one two three four five like up to two hundred say that, beyond two hundred cars will not enter which means that the cars will be turned back the minute the two hundred and one car actually tries to enter in to the IN gate. That means it is actually at the IN gate level, I am actually enforcing the concurrency control. I do not even let cars inside my system. Once I know I cannot handle them but then I have the counter there which makes sure this criterion is ensured before it enters in to the system.

The other one is I do not enforce this rule. There will let cars go in in to my through IN gate but when they actually reach the parking lot, it is at that point i actually see which of them can get in to my parking lot. Now the one which probably came later at the IN gate raised passed the car which was in between which means that though at the in gate level, it is not really first car enter in to that, but by the time it reach the parking lot, it is faster than the other car. Then I let that car to get it which means that the transaction is raised with each other for committing. Now when they raised with each other, they committed at the end of it that can be an altogether different transactions that are committing. If the car is entering into parking lot are not necessarily the cars that came in the same order at the IN gate.

If you remember a basic time stamping ordering would have been would have given a time stamp for each one of them and allowed them to enter in to the parking lot as the time stamp. So it is not truly optimistic in that sense, a locking would have also worked in a similar way except in a slightly different way where it have actually ensure that for each one of them specific lot somebody would have got a token and that token would have been used for enter into the parking lot.

So in some sense, truly optimistic algorithm will allow. It is possible for example: a car which entered but tries not to get in the parking lot or park for some reason it get struck, then there is no point actually trying to reserve the lot for you at the end of the day because you let these who ever comes in you let them get them in and then assume that going to be less than 200 cars at any given point of time using a strict ordering at the IN gate does not really makes sense because you are given at point of time assuming that the conflicts are very rare. As they enter in there, at the end of the day all the cars enter in your gate automatically committed for finding a parking lot, but when you start finding there is going to be large number of cars that will be coming in allowing them inside the system does not make the sense because many of them have to return back.

This is what would have been done in the case of an optimistic algorithm. Optimistic algorithm would have actually allowed all these cars in to parking lot. So let us say there

are only two hundred cars can be parked in the parking lot but then if you actually allow a large number of cars to get in there, many of them go back spending all the fuel of coming up to that point and going back whereas you know that you can only take only two hundred cars inside.

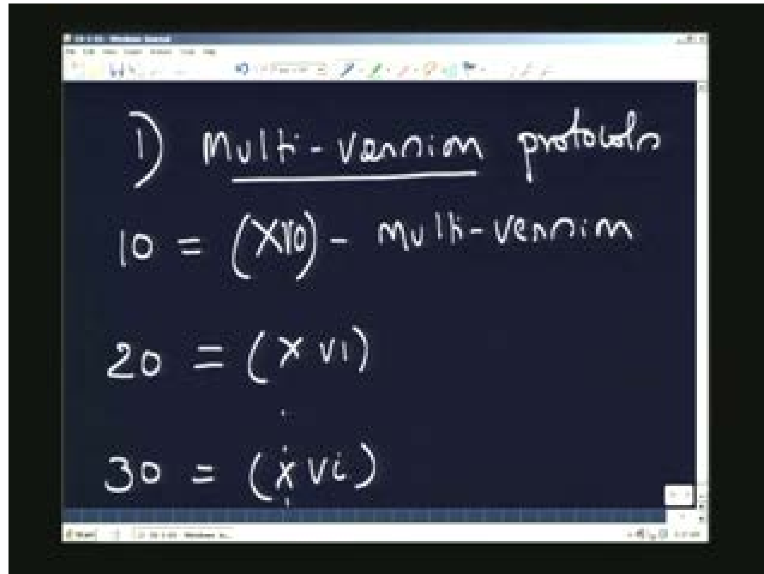
So, in the case where there is high conflict, high contention it is good to actually apply pessimistic approaches like two phase locking because they ensure that you do not really have lot of aborted transactions but whereas when you assume that there are going to be ten or fifteen cars are going anyway come inside putting a large restriction on them ask them to take token all that does not really is worth, that kind of approach does not really worth because at the end of the day all these find their value true in to the parking lot in which case optimistic concurrency control is quiet good. To just sum up, we see is optimistic concurrency control allows the transactions to proceed but there will be wastage when they actually abort, because they have to redo all the work they have done. Pessimistic approaches block the transactions and allow them to proceed only when the road is clear, when the road ahead is clear.

So it make sure that the transactions proceed and they never abort. When start proceeding, they never abort. So both have placed in terms of where there applicable and where there can be used. Now this sort of things comes up our basic concurrency control algorithm discussion where put up in broad perspective the class of algorithms belonging to pessimistic broad class of algorithms belonging to optimistic algorithms. What I am going to do in the next fifteen twenty minutes is to see a completely different set of algorithms.

They actually work in a different approach and different way to just give perspective on is this two classes but a whole set of classes that lie in between that's what we are going to do in the next few minutes we have. To start with, what I am going to do is, i am going to look at a class of algorithms that are termed as multi version protocol. Now these multi version protocols are different because they tend to actually not you know overwrite the value but what they do is, for example if the meaning of the multi version, if you understand here is, every data item X will have multiple versions. Now obviously for example: the X value is ten here. Now when you rewrite this X value at later point of time what you actually get is twenty, then this is going to be a new version of X. It is called X version 1 assuming that this is X version gnome.

So you typically tend to produce what we see is several versions. Each time you actually change a value. For example: now return it the ith time. Now this is going to be i^{th} value that you are producing. Now in the traditional case what we do is, we actually overwrite this value.

(Refer Slide Time: 38:26)

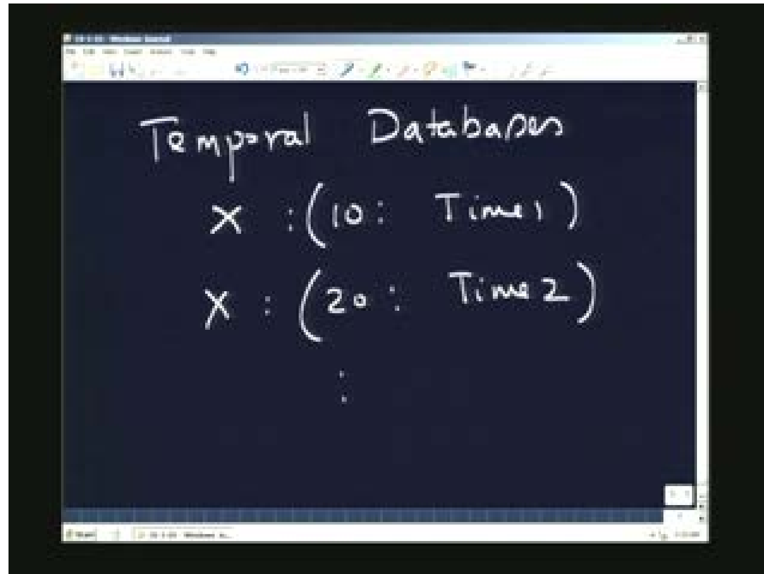


Since we overwrite the value, the old value is completely lost that is where basically the problem of what i have done is correct or not, is multiple people are simultaneously writing this value is possibly what I have written what written by that is where we have problem concurrency control coming in to picture. Now the problem becomes completely different, if you say I maintain multiple versions. For example: you assume in the case of banking database where when i actually, simultaneously two people are withdrawing from the same account or depositing in the same account. You have to ensure that one finishes, one finishes writing the value then other comes and does its part of it. Otherwise, one of it writes going to be loss, but if you actually maintain multiple versions the problem is entirely different.

For example: I take one value produce a new version is take another value produces a new version is that all i need to worry is that versions produces a consistent. There new versions being produced and these versions are consistent the problems becomes severe the older value is always with respect to that older value only producing the new value and if somebody takes the new value and again produces the other value, it becomes a new version of the old value. Now a completely degenerated case of this is, what we are talking in the beginning of the lecture which is called the time based or database or temporal databases. Now temporal databases are also sometimes called historical databases. They preserve the history of what actually is being done.

For example: X value at ten at time t okay. So this X value twenty at time a different time. So you basically record the time as an event as a couple. The time becomes one of the elements of the database values. So value plus the time records at what value is this time and the value. This is very important.

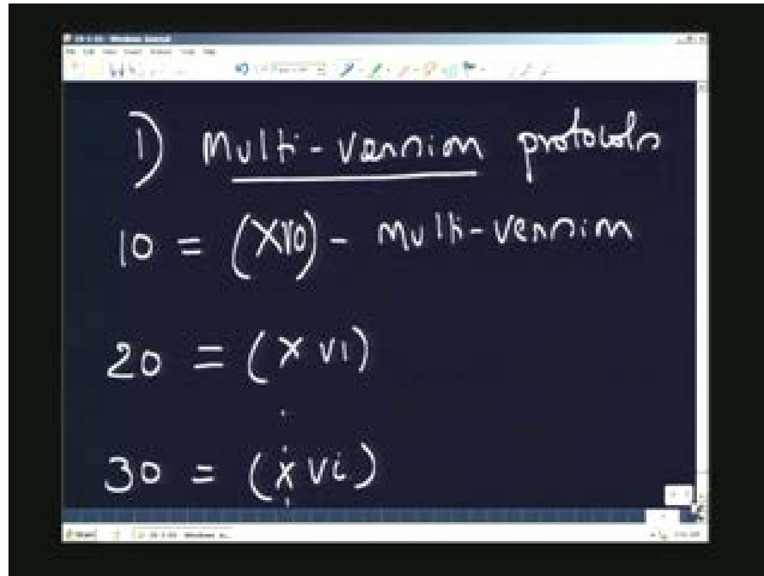
(Refer Slide Time: 40:51)



If you look at some kind of applications like the stock prices for example: The stock price at this point of time will be different from the same stock price, stock value. For example: if you take a particular company like TCS, its stock value is going to be at a particular point of time, its stock value is something, but at different point of time its stock value is something else. So the database if you just give the stock value. It is not really useful because you also need to know the current time at which this was actually done and that is what we mean by purely temporal database the temporal here means component the time component value is also stored.

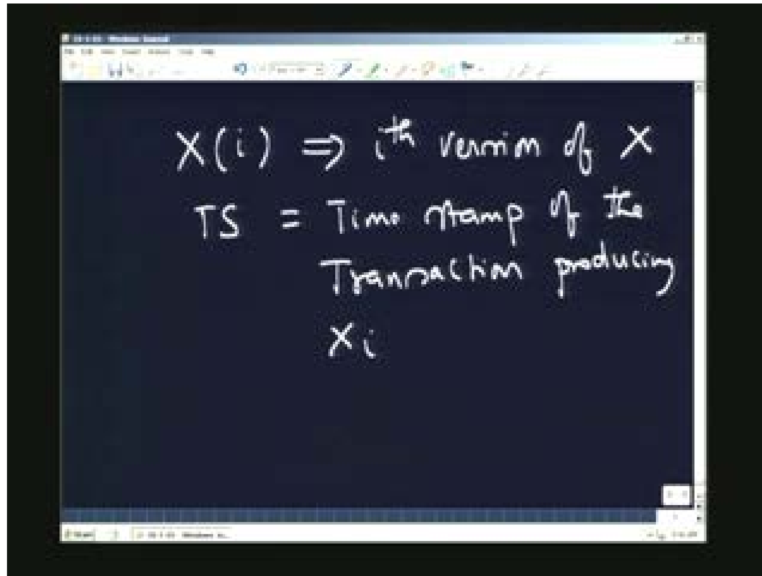
If you start recording the entire history that is completely a degenerative case of multi version database. In the case of multi version database, a multi version concurrency control you maintain certain number of versions. Not all the versions that can be two versions which means that you keep only the current value and the previous value. You are not keeping value that are beyond that time and that becomes a two version database. It is possible that you maintain n number of versions which means that all the values previous values of the data item is always stored. Now what we will see is, set of simple multi version protocols, concurrency control protocols.

(Refer Slide Time: 42:26)



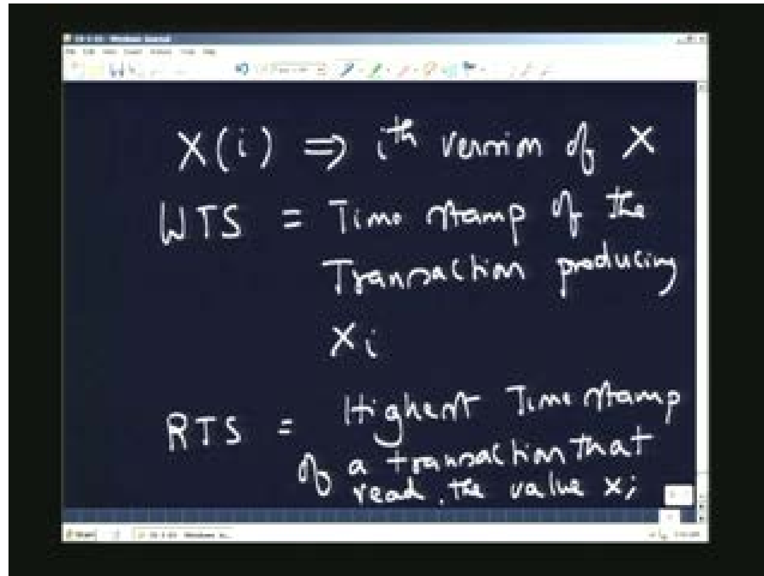
To see how exactly the multi version protocol works and I will also see that as a specific case, a two version two phase locking protocol which have been dealing earlier. Extension of this two version protocol to the case of a two phase locking protocol. Now let me explain how a general multi version protocol will be working and later explain how it extended for the two version two phase locking what we will do is, for every item data Xi we basically going to do record. This is the meaning of this is the ith version of x i^{th} version of data item X. Now i^{th} version would have been produced by some transactions. Now what I will record here is which is the transaction time stamp? This is the time stamp of the transaction that is actually produced value time stamp of the transaction that has actually return. This transaction producing this value, producing xi.

(Refer Slide Time: 43:53)



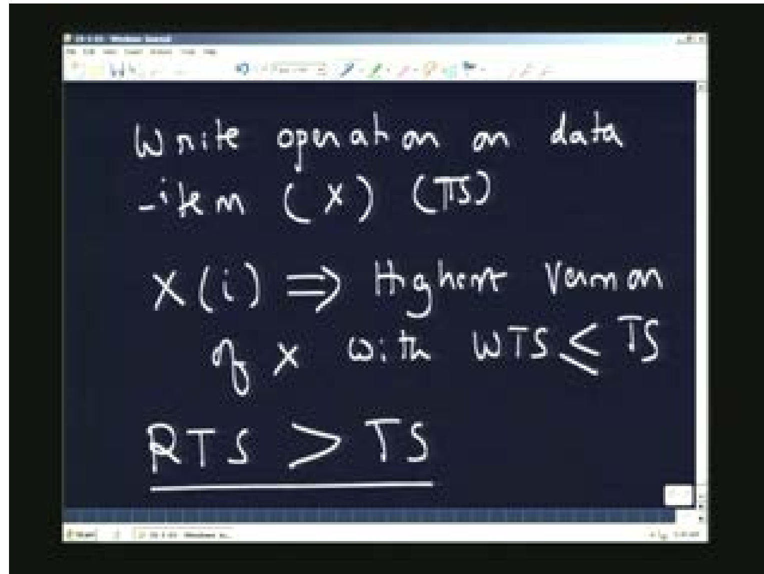
Now, if some other transactions actually try taking the value of X and producing this then it would have become Xi plus one. Each time you write a value remember that you producing a new value new version of the value which means that it becomes if somebody wants to write on X Xi it becomes Xi plus one is no longer Xi. Now the other time stamp that i will be interested in since this is the write time stamp, the other time stamp I will be interested in is a read time stamp. Now it is remembered, this has to be the highest because they could be remembered several transactions leading Xi. Among them, i am interested in the highest time stamp of a transaction that read the value of this highest time stamp of a transaction that read the value of X. I am making it very simple for you. You can read at the end of it, i will give you some reference which you can read more of this. So there are two things we are doing as shown in the earlier case, the write time stamp which gives the time stamp of the transaction that produced. The read time stamp, highest time stamp that read the value of X.

(Refer Slide Time: 45:16)



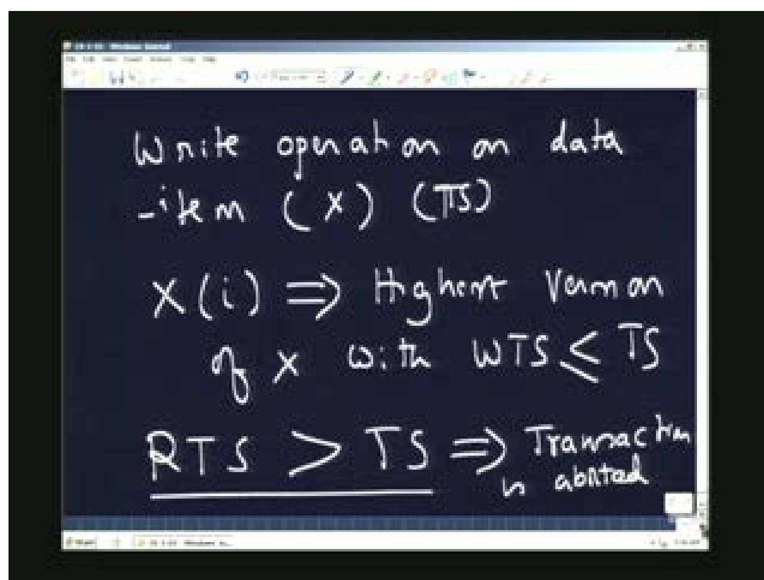
Now what we are going to do now is, for example: You look at the write operation on a data item at the end of it, the transaction issues and now as it issues, a write operation on a data item X. Now as it issues, the write operations on X, I have to find out the X i^{th} version which is basically the highest version produced highest version of X with time stamp, with write time stamp less than this transaction time stamp or equal to almost that means, i look at the latest value of xi which is the current value which I should be taking for Xi to be now operating upon. Now for this particular value, i should see the write time stamp. Please remember, write time stamp is the highest time stamp of Xi read by another transaction. For example; there is already a transaction that read the Xi value. Now i am interested in finding out who read this value of Xi.

(Refer Slide Time: 46:53)



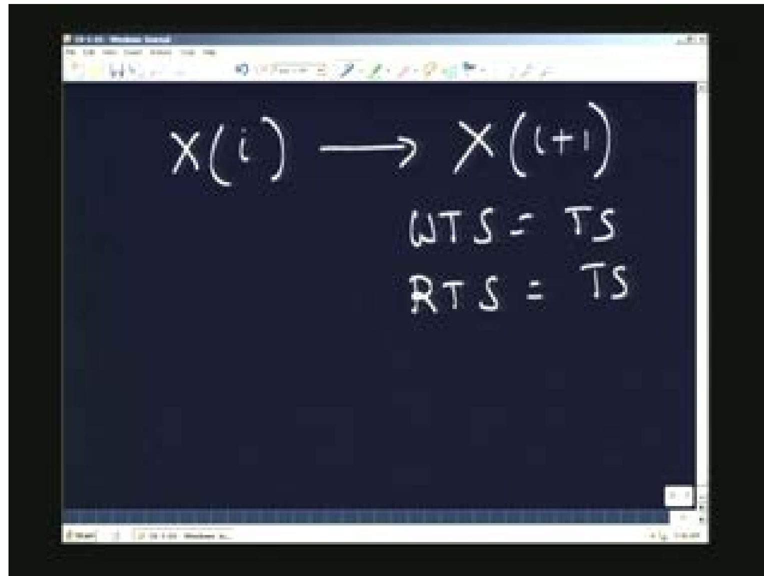
Now if this transaction time stamp which has read is greater than TS, is current transaction time stamp, it means potentially what I will be doing we will be wrong in this particular case, because I am trying to produce a new version of this particular Xi. Xi plus one, but this is already read by somebody who is coming later than me which means that I will be violating, I should not be producing a value which already read by somebody who should be coming later than me show potentially in this particular case, transaction should be aborted. Transaction is aborted and restarted.

(Refer Slide Time: 47:29)



If this condition is not true, what i am going to do is, if the condition is not true that nobody has read this value. If the condition is not true, what i am going to do is, if the condition is not true that nobody has else read the value, Xi is now taken and then a new version of this value will be produced which is going to be X (i plus 1) for which the read write time stamp will be said to TS and the read time stamp will also be set to TS that is the current transaction which has actually produced. This is how exactly a new version of the data item will be produced, if the earlier condition is satisfied.

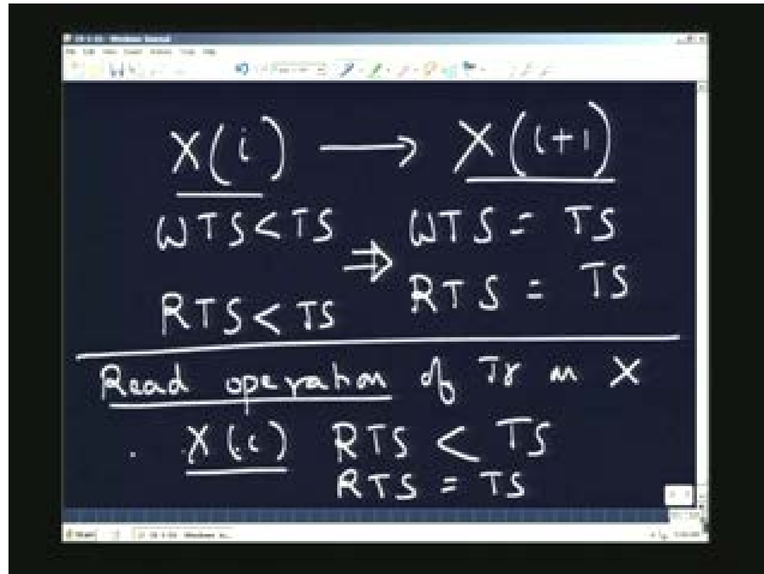
(Refer Slide Time: 48:07)



What in effect is saying is, if i essentially look at when i am trying to write the latest value which I should be using to produce this value that I have here. What i am going to look at is, the write time stamp is suddenly less than the TS, then I look at the read time stamp of this and make sure that the read time stamp is no more than the Xi time stamp which i am setting and if this is correct, then i will basically proceed and produce the new version. The condition is here this is less than TS, is also less than TS in which case actually we produce the new version of the version of the value.

Now read operations are quite simple compared to this. Read will essentially what it will do is, it will look at read operation of the transaction on X, it will actually first look at all the sizes and make sure that Xi satisfying the RTS, read time stamp highest RTS is less than the TS, that means this is the latest value of Xi which is less than the TS which this time stamp can read, which this transaction can read. After this, the TS value will be set to the time stamp of the current read operation that means the Xi, the highest Xi will be taken always you read the latest value of Xi and when you reach that latest value of Xi, then make sure that you are actually setting the read time stamp of this to the current time stamp which means now the RTS the highest read time stamp of Xi will be equivalent to the current transaction which read the operation. That is how the read operations will be performed on the transaction. This explains, how the basic multi version protocols execute.

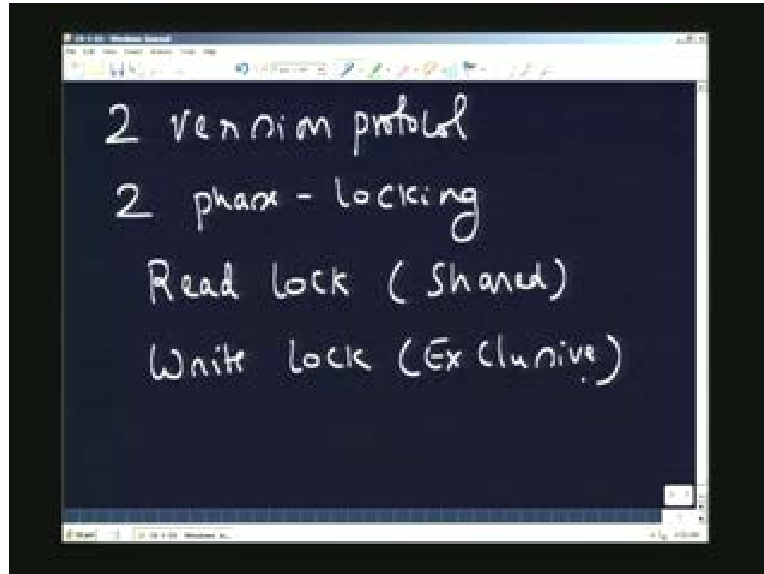
(Refer Slide Time: 50:15)



In the case of multi version protocol, I did not keep any specific limit on the number of versions that are produced by the multi-version protocol. What this means is, the 'I' can be any number. Now often the problem will be that, i will be ending in highest cost in this particular case because i have to store a large number of data items because not just the current value, but the previous values all have to be stored in the database which means that the database storage overhead is going to be somewhat higher in this particular case and that is one of the tricky issues of the multi version protocol.

That is where i actually paying a larger cost a more specific is actually the two version protocol which means that you keep only two versions of the data items. Two version protocol and we will look at these two version protocol extension to two phase locking. If you remember in the case of two phase locking, we actually use two locks: one is the read lock and other is the write lock. Read lock is basically a shared lock and write lock is the basically an exclusive lock.

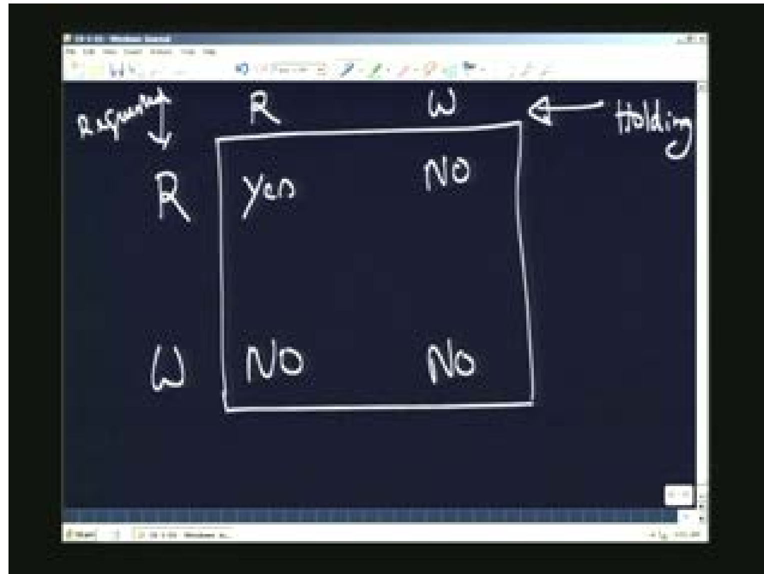
(Refer Slide Time: 51:57)



Now what we have done is in the case of two phase locking every data item is either locked in the read lock or the write lock more and in the presence of a read lock another read lock can be allowed. There is a write lock another write lock another read lock will not be allowed. If you look at typically the matrix of what will be allowed, in what case you end up actually having, let us say this read write lock which is the current transaction holding, then you have a requested ones which are read or write. These are the requested and these are the holding, currently holding.

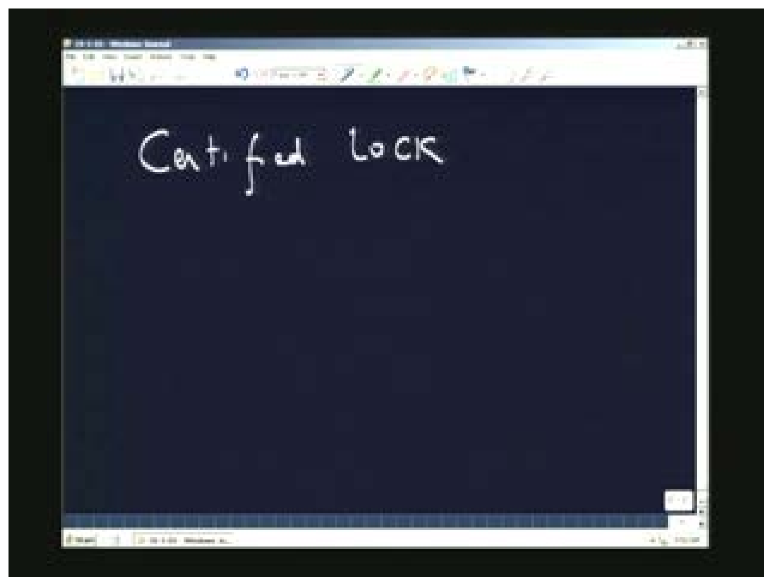
Now it is possible at the end of the day, if it is basically a read lock in the presence of another read lock is yes. If there is a read lock on a data item X, you can also grant another read lock. If there is a read lock, this is going to be ‘No’. If there is a write lock, read will be disallowed. If there is a write, another write will be disallowed which means a data item on which a read clock is currently there, another transaction cannot ask for write lock, but it can ask for read lock since read is shared.

(Refer Slide Time: 53:10)



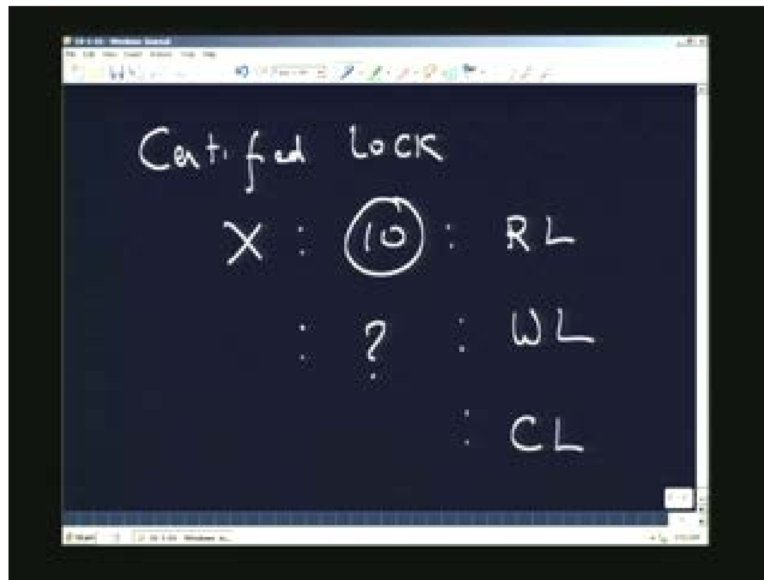
If the transaction is holding write lock on a particular data item, no longer you can ask for on particular data item. This is how two phase locking prevents on a particular data item things being pretend in an inconsistent way. Now, what we will do in the multi version extension of the protocol is we will introduce a new lock called the certified lock. Now what the certified lock will do is? It will allow in the phase of write also reads which means that you can read a previous value of the data item when somebody still holds the write lock, because it is still not return a value on the database.

(Refer Slide Time: 53:59)



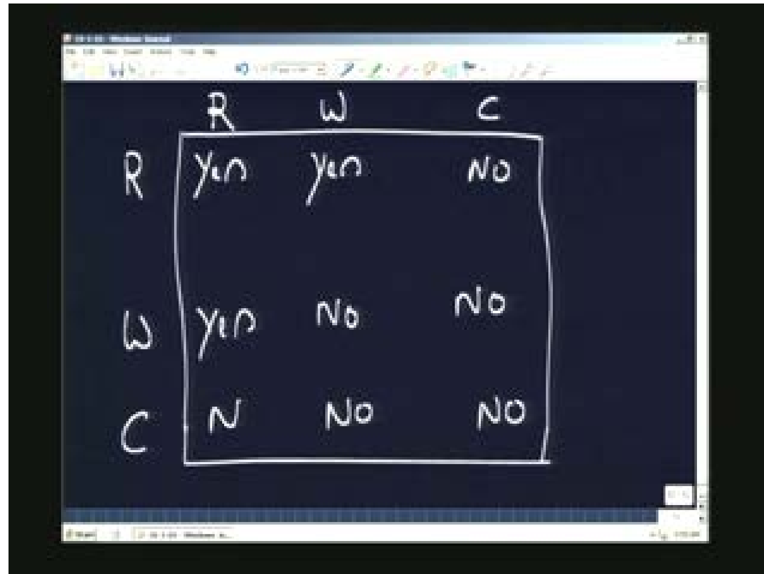
For example: Remember X read lock data value item value of, this is the read lock that i have got and another transaction has actually got a write lock on this, but it still not written the value of this. This is what we will do here is, we will say this is the question mark here which means that it still not return the value. Now it is possible for me to actually allow this write lock. In the presence of write lock, a read lock has to be taken, but then when actually this transaction wants to write, it will upgrade this to certified lock. Now when a certified lock comes, it is going to write the value and both write and read value are going to be disallowed in the presence of a certified lock. Now this becomes a two value in an extension of two version multi version protocol because there are two versions. Old value which will be allowed to read in the presence of write lock, but when actually the value is return that is the time, it will be updated in the earlier it will become somebody can use.

(Refer Slide Time: 55:06)



So, this is basically two version extension of a multi version protocol that we see. Now to give better explanation what really we gain by doing this will again produce the matrix which we produce earlier. By saying that, there are three locks. Now read write and certified lock and similarly will have a read, write and certified lock. Now, in the presence of a read lock you can still grant a read lock. In the presence of write lock, you can still grant a write lock, yes but then certifies it is no. Now in the case of a somebody holding a read lock, you still will be able to grant a write lock here, because it can be on previous value, write value is basically no and all other cases it is going to be no, because in the presence of certified lock you can get any other locks.

(Refer Slide Time: 56:12)



Now the advantage of this is, when somebody has actually got a write lock, please understand intuitively what is the meaning of what we are trying to do here? what we are trying to do here is, when somebody is trying to hold a write lock, it is only intentional lock. I am intending to write. I am actually not writing. For example: if the transaction executes for a sufficiently long time, he need not actually block others from even reading. The others can read the previous value commit and go as long as i have not trying to commit. He commits again my old value. So it comes before me finishes everything. I do not need to like somebody trying to write in the railway reservation case. Takes the form, tries to fill the form, but while he is trying to fill, there is no point trying to block everybody else from trying to commit or trying to come before him and trying to finish his transaction that when he actually writes it and gives it, that is the time it is going to be blocked, not before that. This allows a higher level concurrency, but at the cost of actually an additional lock that i will be holding, when i actually reach that particular point.

This is an extension because i am only trying to have two values of a particular data item and allow with respect to this two values, who can come before me or after me that's what exactly is achieved when you use the two phase locking extension to the two version protocol and that is how it actually gives more concurrency as compared to earlier two phase locking. This is the interesting extension to two phase locking using the concept of multi version protocol. What we are going to do in the next class is take a few examples for all the lectures i have done and do review questions on the topic we have done so far.

Thank you.