**Database Management System**
**Prof. D. Janakiram**
**Department of Computer Science & Engineering**
**Indian Institute of Technology, Madras**
**Lecture No. 22**

# Concurrency Control – Part -3

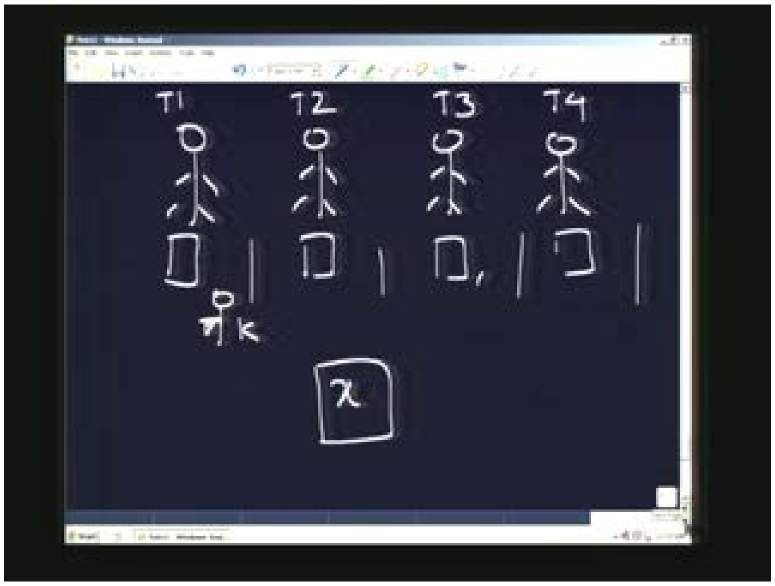
We will continue our discussion on Concurrency Control algorithms in this class. Last few lectures just to recap what we have doing. We looked at a class of algorithms try classify them into two broad classes: pessimistic algorithm and optimistic algorithm. In the last class in particular, we have seen in depth where exactly this difference between optimistic and pessimistic kind of algorithm come. Class before we have seen how 2 PL works in the context of pessimistic algorithm and also we mention that. This class we are going to looked at one optimistic concurrency control algorithm before we actually look at we also going to look at what we understand as time stamp based algorithms as supposed to locking we will look at time stamp based concurrency algorithms.

Now at alltime stamp based algorithms are optimistic algorithms that you have to understand time stamping algorithms essentially used the notion of time for serialiazation that is what they do. Time stamp algorithms need not necessarily be optimistic algorithms but one can formulate optimistic algorithm and really what we are meaning optimistic algorithms? We were saying that the validity of the operations done by the transactions is checked at the end of the execution of transaction that’s what we actually mean by the optimistic algorithm. Now it is not necessary that all time stamp based algorithm check for the consistency of the results consistency of operation and by the transaction at the end of the execution.

They may do something in between they do not need to necessarily do it only at the end of the execution of the transaction it is possible to do it. So what we will do in this class is we will try to understand in a simple way, which we call as basic time stamp algorithm try to understand how time stamp based algorithms work in the context of concurrency control. What we are going to do is? Look at know how you apply time stamp based algorithms how the recovery properties commit properties and recovery properties are integrated in to the time stamp based concurrency algorithm.

If you remember when earlier lecture on two phase locking we discussed how locking algorithm are integrated with a commit protocol. So we are going to do the same exercise of looking at time stamp based concurrency algorithm and also look at how time stamp based algorithms will integrate with commit based algorithms? So that’s what we are going to do in lecture. Now we will proceed to look at basic time stamp based algorithm. So what we will do is we will explain the time stamp based algorithms.

(Refer Slide Time: 5:10)



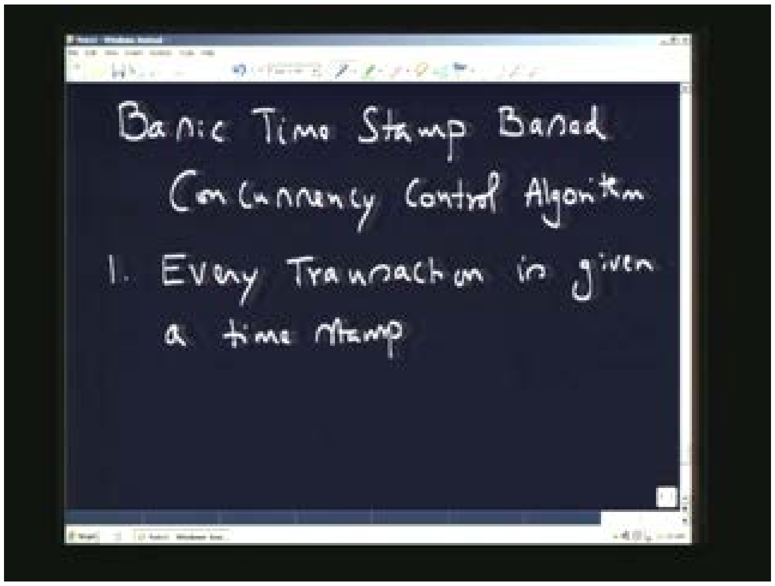
By taking a simple case of a basic graph of basic algorithm which is called the basic time stamp based algorithm, concurrency control algorithm.

(Refer Slide Time: 5:40)

Basic Time Stamp Based Concurrency Control Algorithm

The algorithm has several basic things that's done like any other concurrency control algorithm. The first thing the done is typically in this particular case every transaction is given a timestamp.

(Refer Slide Time: 6:18)



Basic Time Stamp Based
Concurrency Control Algorithm
1. Every Transaction is given
a time stamp

This is if you typically look at you know very good example in real life of getting a timestamp and then executing what you want to do, the darshan of lord Venkataeswara know Tirumalai Tirupathi devasathanam as this thing called they give the time band you know. When you actually want to have darshan of the lord they say that you have to obtain a time band. You know what exactly the time band means? They give you the times sticker and says that if you actually go there know before the time or above just the time, you will be able to see the lord. So basically in other words, the darshan of the lord is to be serialized because you can see one after other, the queue one has to move to see the lord. So basically a serializable will be a problem there.

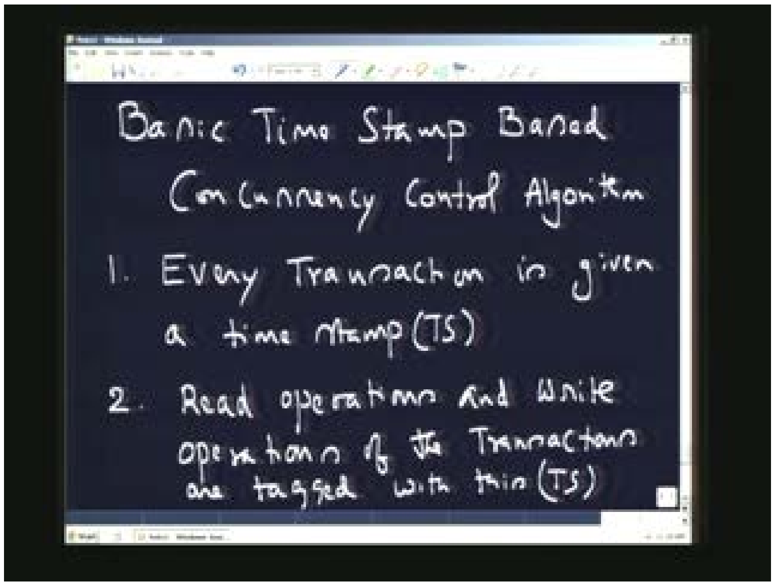
Now one of the ways we solve this in real life by saying that you need this band you know this band is generated. What is the band means there? Essentially getting a time stamp time band and says that all the people what is the time band have to proceed the order of time that they have got that means they will be viewing the lord in terms of the time band. In a similar way, if you look at how the transactions are going to execute, each transactions when it enters the system, it get the similar time band that is what called time band. How exactly this time stamp is given to the transaction? Later we will look at the step but the idea of giving the time stamp to the transaction is that essentially time stamp serializes the order in which transactions are executed. All transactions are executed as in the increasing order of the time.

If a transaction has got a time stamp, its order in terms of execution is fixed because all those transactions which got time stamp before this transaction will commit before this transaction. All those which got a which got a time stamp higher than this can proceed and are going to commit after this transaction is committed. So in that sense in serialize order time the notion of the time here serializes the transaction execution. This is a very important concept. This notion of time and I am using the notion of time is a very important concept. As we go further down, we are going to look at distributed transactions were a single transaction is split into sub transactions and execute on multiple sides. This is like not now using single time stamp. For example; for each person could be carrying is own wrist watch. So you get into additional problems of how do I actually take this different times and now put them all into one zone. For example; now London will be in one zone, Chennai will be in one time zone.

So if you are actually generating transactions in different time zone, you get into problems. Now all of us using the single clock, automatically everything will be serializable but if they use multiple time zones then you get additional problem of making sure that the events across multiple time zone are again serialized. We are going to look at the notion, when we go on look at the transaction and see how this notion of this time becomes critical when you want to address the concurrency control problems in the case of distributed databases. So to understand at you know very simple level, what really happening is when transactions are coming in into the system giving them a time stamp and you are excepting the transactions are executed with respect to the time and essentially produces the serializability. To look at the basic steps in little more detail.

As I was trying to do here every transaction start with will be given a time stamp. This timestamp which will be used subsequently for checking whether the transactions are executing in a consistent fashion or not. Now all read and write operations of the transactions are tagged with the time stamp. Read operations and write operations the transactions at tagged with these time stamp, write operations of the transactions are tagged with these keys will call these time stamp as TS. So all the read and write operations are actually tagged with this time stamp that's the second thing.

(Refer Slide Time: 12:06)


Basic Time Stamp Based Concurrency Control Algorithm

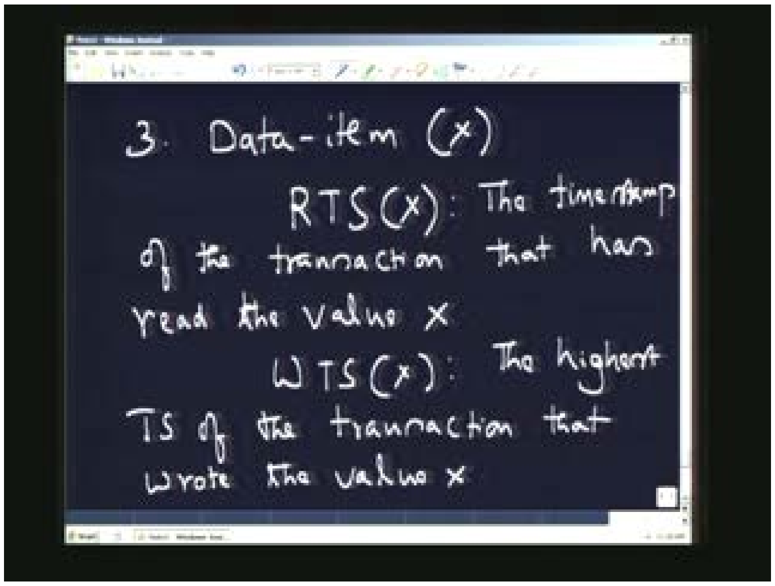
1. Every Transaction is given a time stamp (TS)
2. Read operations and Write operations of the Transactions are tagged with this (TS)

So when the transaction issues are read or write operations, then let us say transaction TI has got a time stamp of ten twenty just understand or at this point of time is eleven thirty. We will say the transaction got a time stamp of eleven thirty. Now subsequently whatever the read or write operations that are being performed by transaction will all get the same number that's one time stamp we are going to use. For example, in a later point of time eleven thirty five or eleven forty, you are doing something but all belong to same transaction. They are not giving the time stamp this is very important.

All the read and write operations of the transaction are tagged with the same time stamp which is given to the start of the execution of the transaction. Now what do we do after this point of time? We also have to actually know for every data item in the data base okay. For every data item X, we need actually what is the read time stamp of the what is the read time stamp of X and read time stamp of X shows the time stamp of the okay the time stamp of the transaction okay that has read the value of read the read the value of X. What is this mean?

This means let us say this is the data item X and there is the transaction which read this data item and time stamp. Let us say eleven fifty and say that is the highest transaction. There could be transaction time stamp lower than this which also read this data item. But we are going to consider the highest time stamp of the transactions that has read the value of x will be the RTS of X. Similarly the write time stamp of x is the time stamp, the highest time stamp. If you want to actually qualify it the highest time stamp or TS of the transaction that wrote the value of X the transaction that wrote the value of X okay. So for every data item, you are going to do have two time stamps corresponding to the transaction highest time stamp value of the transaction that has read the v value and

(Refer Slide Time: 15:25)

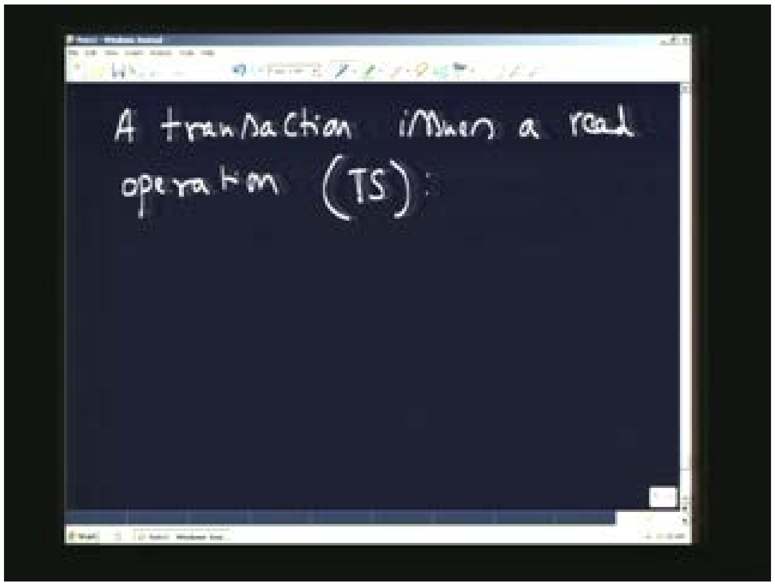

3. Data-item (x)

RTS(x): The timestamp of the transaction that has read the value x

WTS(x): The highest TS of the transaction that wrote the value x

also the value and also the highest time stamp of the transaction which wrote the value. Why do I need this tool? because this essentially tells with respect to this data item. What are the earlier transactions that I have actually done whether the read or write operation are performed by the transaction on this particular data item. Now we will use this time stamp to see the validity criterion when a new transaction comes tries to do some manipulation on this data item. We are going to look at it and say which are all those values which can be allowed by the transaction manager to proceed and which needs to be aborted. Now this is the preliminary thing in terms of how exactly the algorithm maintains the data. Now in terms of actual execution, what I does is, whenever a transaction okay; now consider the case when a transaction issue. A transaction issues the read operation the read operation. Now this read will be tagged with the time stamp of the transaction. So basically, essentially this is TS of the read operation okay, time stamp of the read operation. Now these TS has to be compared with the data item time stamps. Now let us say,

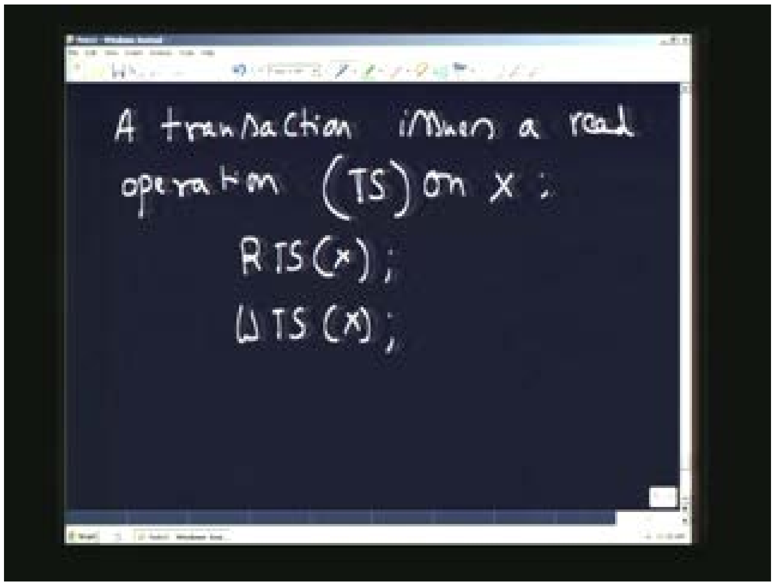
(Refer Slide Time: 17:14)



A transaction issues a read operation (TS):

The read time stamp of X and write time stamp of read operation we will say on X a transcation with time stamp TS issues a read operations on X. Now the RTS and now WTS the value of X are the following. Now what is the condition? Now this read operation can be allowed to proceed. Now the read operation can be allowed to go further as long as the condition that the time stamp of X that the current transaction is greater than the write time stamp of X. Please remember the reads can be shared.

(Refer Slide Time: 18:00)



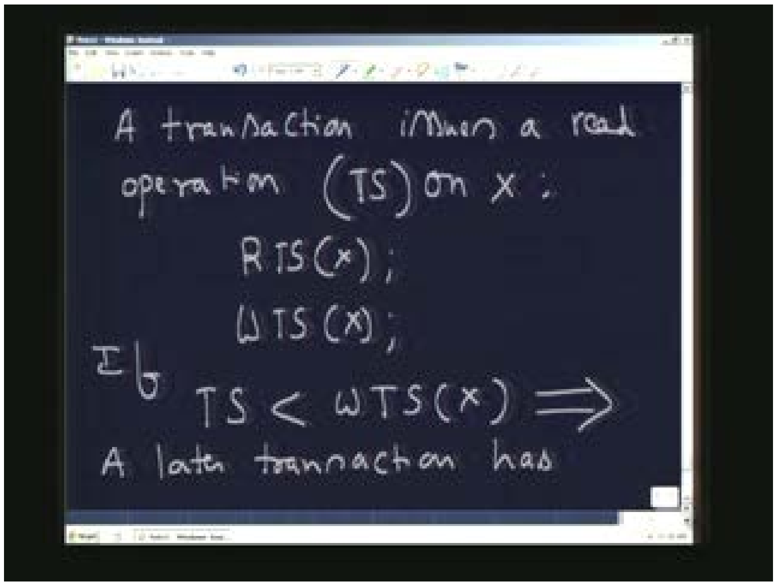
A transaction issues a read operation (TS) on X :

RTS(X) ;

WTS(X) ;

After I write a value any number of people can read that value, but if you assume that only if I am actually reading a value after somebody actually written a value on X, which means that the my time stamp value is lesser than what actually has been produced. Then I need to be little careful and avoid such operations. Understand the condition where TS is less than WTS of X, If TS time stamp of the transaction is less than time stamp of this current transaction is less than right time stamp of X. What does it actually implies that means a later transaction actually return the value. A later transaction than me has written the value.

(Refer Slide Time: 19:08)



A transaction issues a read operation (TS) on X :

RTS(X) ;

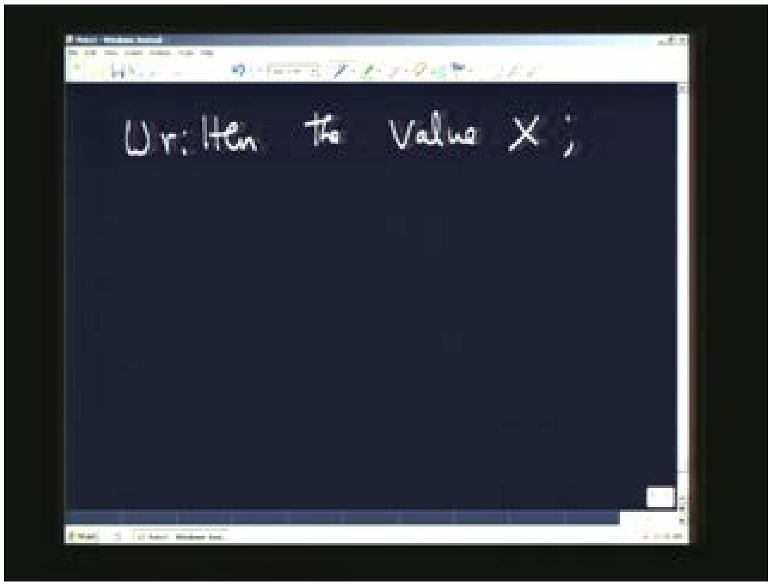
WTS(X) ;

If $TS < WTS(X) \Longrightarrow$

A later transaction has

A later transaction has written the value of x. Now in terms of time stamp order if you want to execute the transactions strictly in the time stamp order, this should not have happen.

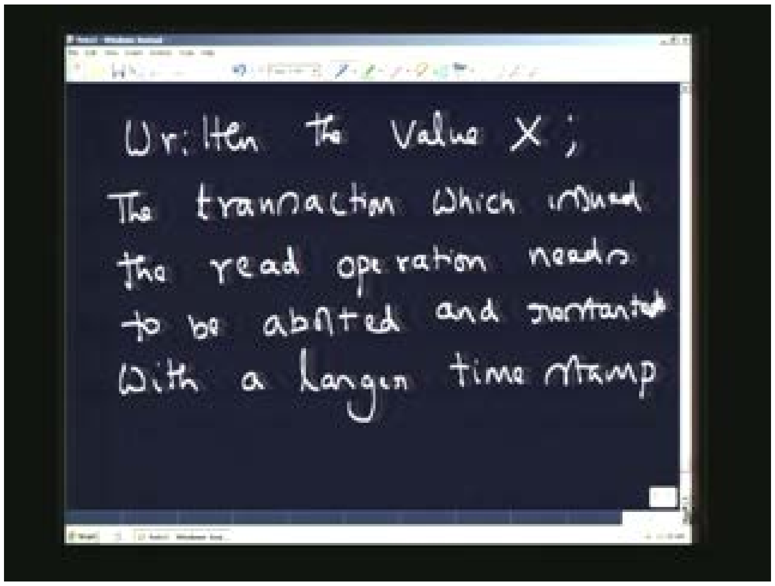
(Refer Slide Time: 19:20)



Written the value X ;

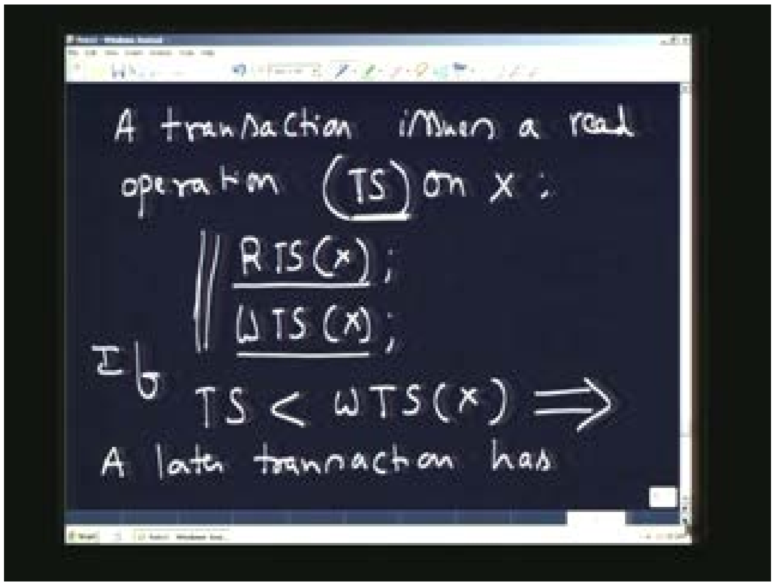
Because now I am actually reading a value produced by somebody who is coming later than me. This is like saying that know if you look at senior junior relationship you know. Now if you essentially says that I am actually I am passing the batch which supposed to passed in 2004 says that its actually passing after the 2003 batch there is a violation, because 2003 batch has already gone, which means that the value return by x is by transaction later than me or older than its me. Now I cannot come back and say to read its value okay. So unless I had actually finished the later transaction would not be able to come and do whatever is trying to do. So in that sense, now this is the case transaction T is to be the issuing transaction has to be aborted the transaction which is actually issuing. If this is the case the transaction issuing the transaction which issues which is sure which should read the operation needs to be aborted okay and it is restarted okay and restarted with the higher order time stamp restart we are going to add higher values time restarted with a larger time stamp okay. Now understand what I am saying just going back.

(Refer Slide Time: 21:30)


Written the value X ;
The transaction which issued the read operation needs to be aborted and restarted with a larger time stamp

A transaction issues the read operation TS on X. Now the transaction time stamp is TS here as shown here and the current values in the database as far as RTS and WTS concerned are shown here. RTS shows the transaction larger time stamp of transaction that has read the value of x. WTS shows the time stamp largest time stamp of transaction which is return a value on x. Now my time stamp is less than WTS. I do not need to compare this RTS, RTS value is higher since reads can be shareable. They need not be exclusive I am still not violating any consistency thing only write I have to read something earlier return by transaction. This is earlier than me rather than a transaction later than me. This essentially shows that a later transaction has written the value of x.

(Refer Slide Time: 22:39)


A transaction issues a read operation (TS) on x ;

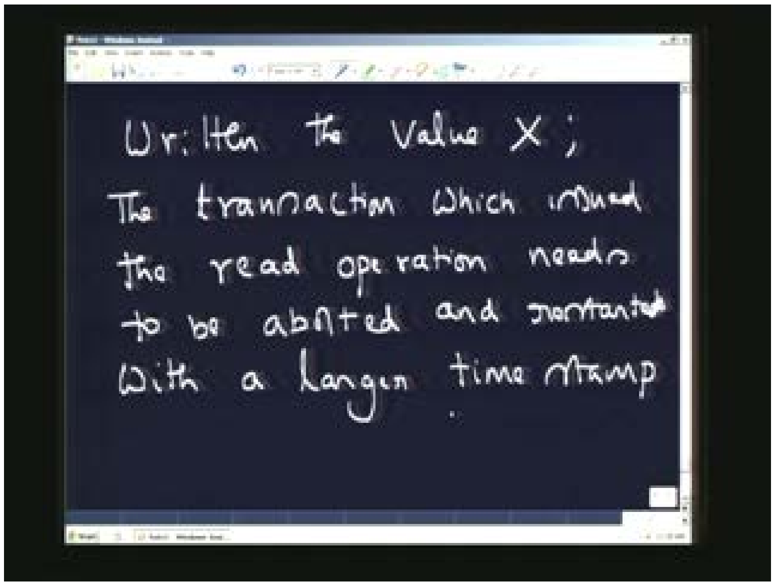
|| RTS(x) ;
|| WTS(x) ;

If $TS < WTS(x) \implies$

A later transaction has

The transaction which is issued the read operation needs to aborted in such case and restarted with the larger time stamp and when you restarted a chance of distractions succeeding his higher because now it will take this and proceed further and chance of the transaction succeeding later is very high.

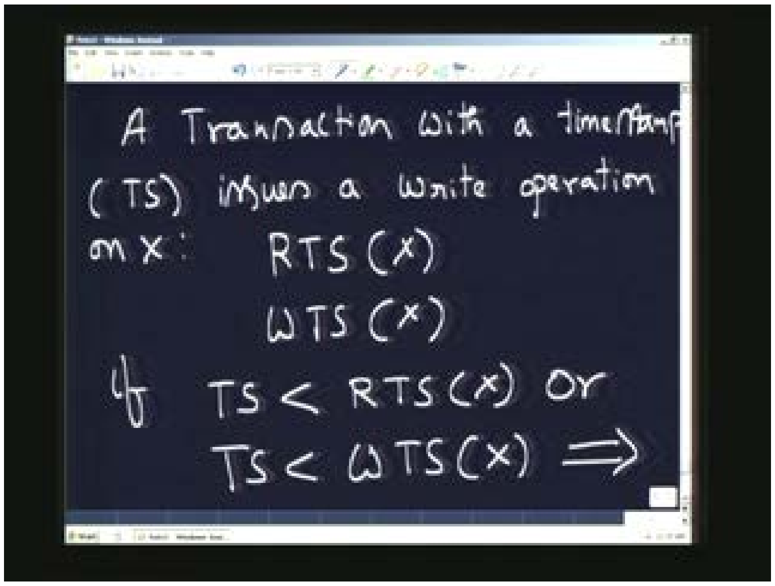
(Refer Slide Time: 22:55)



Written the value X;
The transaction which issued the read operation needs to be aborted and restarted with a larger time stamp.

This is what we will be done as far as the read operation is concerned. Now if you look at the write operation, a transaction let us say with the write operation, transaction with the time stamp, with the time stamp okay TS issues a issues write operation okay. Now if issue a write operation you have to check now. Let us say there are again similar case of RTS is the current value, WTS is the current value of the data item X. Write operation will say on X okay, in which case the RTS and WTS are the current value. Now you need to check if either of these conditions are true, TS is less than the RTS RTS thats the lead time stamp or TS less than the write time stamp of x then what does this implies? This implies that I am writing a value into X which was read by the later transaction or return by the later transaction. In both cases, what I am saying is, now I am trying to manipulate the value of data item but somebody comes later than me actually read as value return this value.

(Refer Slide Time: 24:55)



A Transaction with a timestamp (TS) issues a write operation on X: RTS (X)

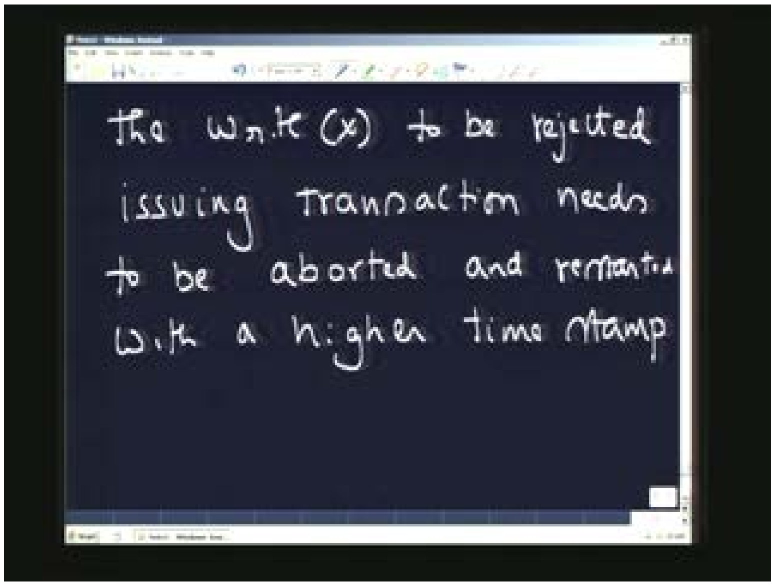
WTS (X)

If TS < RTS (X) or

TS < WTS (X) ⇒

In either of the case, this results in an inconsistent situation because who is somebody coming later than me as actually read. I am actually modifying something would have read some stale value okay. Similarly I have read value that value will be last come then and start modifying it. So in that sense both these conditions the write has to be rejected okay the write operation has to be rejected write on X has to be rejected. Now what are the reject means? Same as the earlier case that, the issuing transaction has to be okay. The issuing transaction needs to be aborted and restarted with a higher stamp. Transaction needs to be aborted and restarted with a higher time stamp. Now this essentially explains their basic steps of the time stamping algorithm okay.

(Refer Slide Time: 26:17)



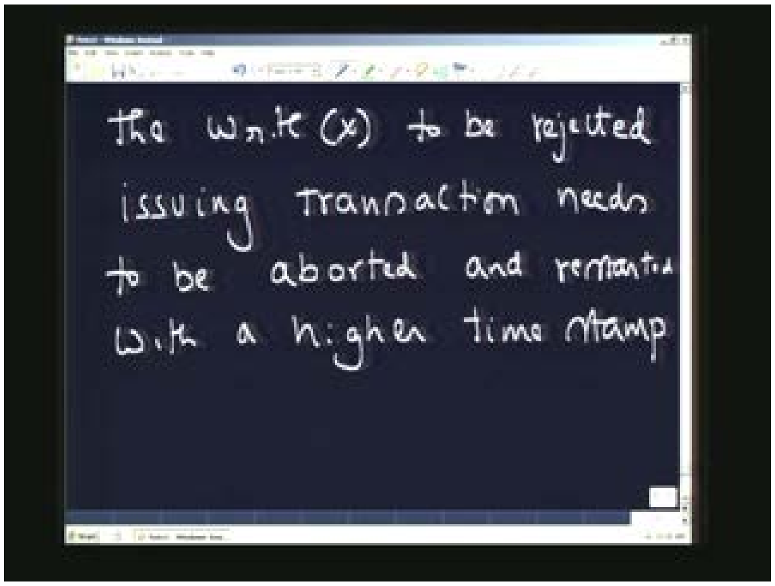
The write (X) to be rejected

issuing Transaction needs to be aborted and restarted with a higher time stamp

Now what we are trying to do in this particular case is, we are trying to say the algorithm tries to provide a basis for know checking. We give transaction a time stamp no we did not still discussed. How the transaction will be given a time stamp know several ways it which can be done. One way is actually you can use a counter. For example; first transaction will be given a counter value of one. The second transaction will be given higher number than this. Since all I am interested is logically showing that the transactions comes one after the other, a counter is good enough for me. Only thing is the counter eventually might become infinitely large means that means the counter value will become so large that at the end of it, somewhere I have to reach at the counter.

So at some point of time when there is more brief period of time like you know. This is period when everybody goes for lunch know like that when the when there are no no transaction for brief period know in the system you can just forget the entire earlier history. You can just set the counter zero and then start doing it again from one that point of time okay that's one way of actually giving time stamp to the transaction. This is basically a logical time stamp because all that doing saying is transaction t 2 which got a counter value of two is higher than a transaction which got a counter value of one. So in terms of actually looking at t 1 and t 2 all that that you have decide in terms of time order is the counter value okay, then are various other ways of actually giving the time stamps for the transaction. The other way is actually to use a physical clock okay, physical clock that gives the values. Physical clock that gives okay this is like saying

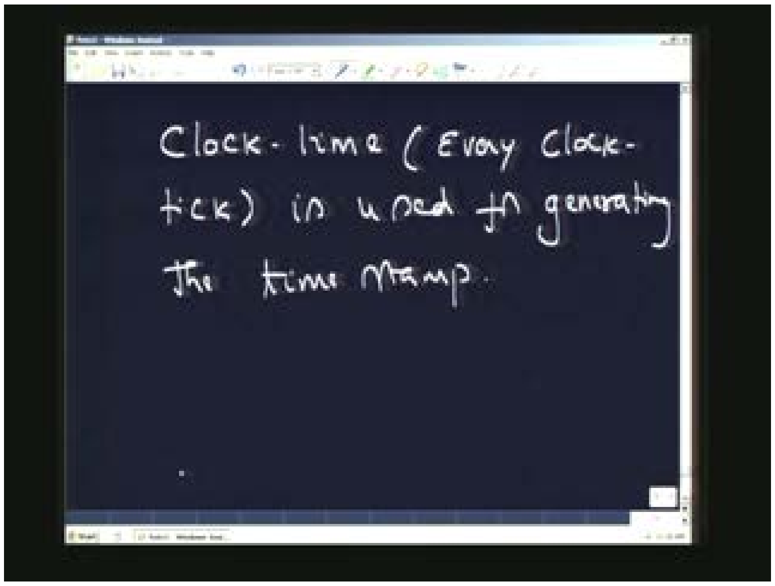
(Refer Slide Time: 28:37)



the Write (x) to be rejected
issuing Transaction needs
to be aborted and restarted
with a higher time stamp

I will use the clock time of machine okay the clock time and then I will basically say that every clock take I generate okay, every clock take is used. Clock take is used for generating the time stamp. Now there are interesting, this is important because every clock take can generate only one time stamp.

(Refer Slide Time: 29:22)

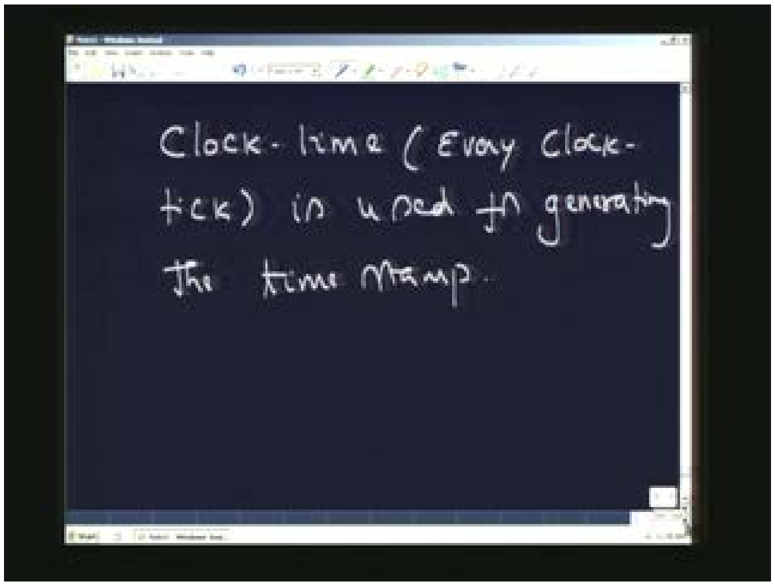

Clock-time (Every clock-tick) is used for generating the time stamp.

So, which means that the number of transactions that can come limited now, because they are limited by the number of clock takes because the time stamp will be generated only one time stamp can be generated in a clock take because two transactions need to get two different time stamp. They cannot get the same time stamp. If they get it then you will have problems of validating them at the end seeing how they will basically know you need to order them. So, you would not be able to order them if both of them get the same time stamp. Then interesting algorithm for generation of time stamps, in fact there is a very interesting algorithm which does not generate time stamps with respect to the starting point.

He generates with respect to the time when the transaction is excepted to finish. This is an extremely interesting way of looking at the problem in an entirely different perspective. There is like saying when somebody entering in to IIT, it is at that point I give a tick to him or I give a band to him which means that is excepted now to follow everywhere else with that band. So the band was given with respect to when actually came in to the system. The other thing is when is excepted to leave the IIT gate. I actually put a band there, saying that this is the time you were excepted to leave the system.

If somebody close up at the gate know after his time over you basically know abort that particular thing. This is like saying that is excepted to leave the system within that particular time I come to the gate at ten o clock. Now one way of saying that is I get the ten o clock at the time stamp the other way interest when we saying is I am supposed to leave this system by twelve o clock. So I give twelve o clock as a time stamp which means that ending time is given that means all the transaction if the transaction dint finish within that particular time period. It will be aborted. So possible for know for the algorithms to generate this time stamp in extremely interesting ways that used for um some algorithms to actually fixed a band. A band of time during which the transaction is excepted to finish the execution. So one can intelligently use generation of time to see how the transaction execution can be controlled. Now has to just come back to the point of how the algorithm works and then see for further issues with respect to the algorithm and how this algorithm is different from locking kind of an algorithm. What we do in this particular algorithm is?

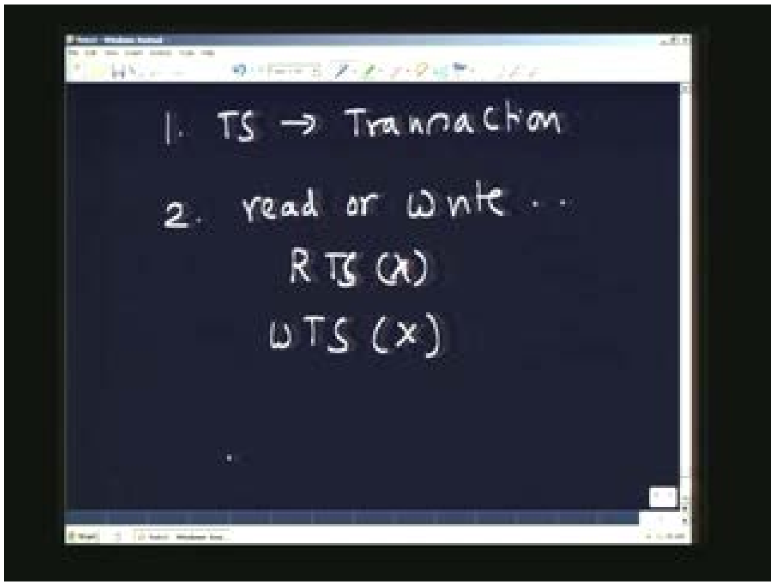
(Refer Slide Time: 32:20)



Clock-time (Every clock-tick) is used for generating the time stamp.

We actually giving a time stamp TS to the transaction. Every transaction is getting a time stamp and then the read and write operations of the transactions read or write checked again the RTS or the WTS. One of the things I have actually forgotten to mention is when you accept a read operation or the write operation, immediately you need to actually update the time stamp of the data item.

(Refer Slide Time: 32:54)



1. TS → Tranaction

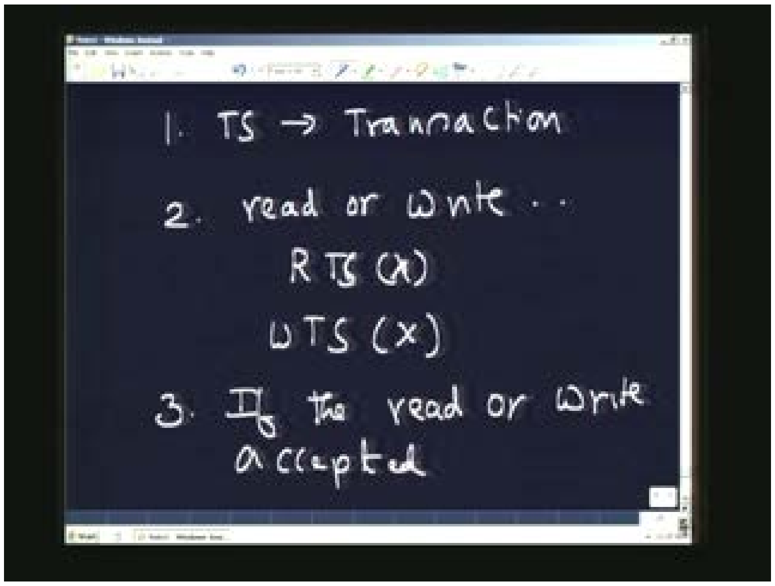
2. read or write . .

RTS (x)

WTS (x)

For example: I showed only in the earlier case how the RTS and WTS will be used to abort a transaction. For example; if those conditions are not met you are going to abort the transaction, but if you say that you gone to accept the reads and writes on the data item. For example; let us say. Now these read or write on TS accepted. If the operation is accepted, if the operation is accepted if the read or write accepted the abort conditions we actually saw earliest. So I will just not mention them here. If the read or write accepted then the RTX corresponding RTX.

(Refer Slide Time: 33:52)



1. TS → Tranaction

2. read or write . .

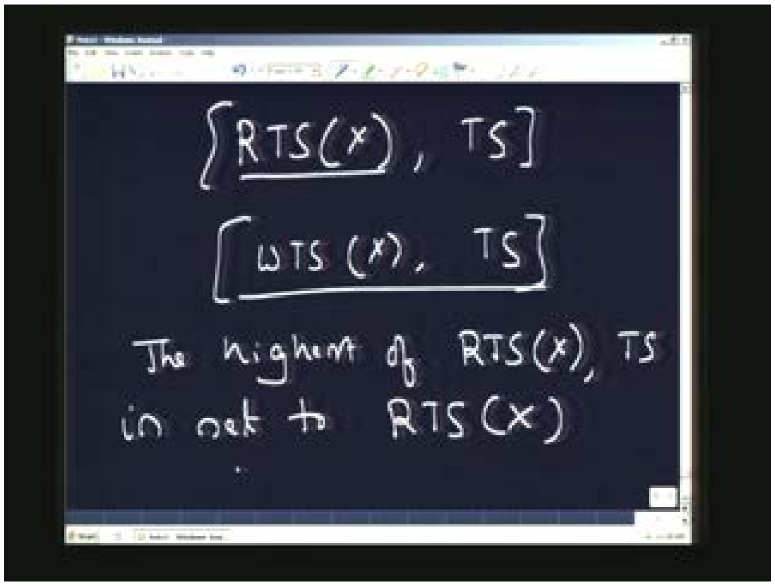
RTS (x)

WTS (x)

3. If the read or write accepted

to be updated RTS of X okay to TS okay highest of this is what should be made equal to know. Now similarly we are going to look at the write time stamp of X and then the TS okay. The highest value is set to the highest of RTX, TS is set to RTS of X similarly for the WTX the highest of WTS X and TS when a write operation is accepted the corresponding value is basically updated. What this means is? The RTS and WTS always reflect the time stamp the highest time stamp of a transaction which either read or written on the data item X.

(Refer Slide Time: 34:52)

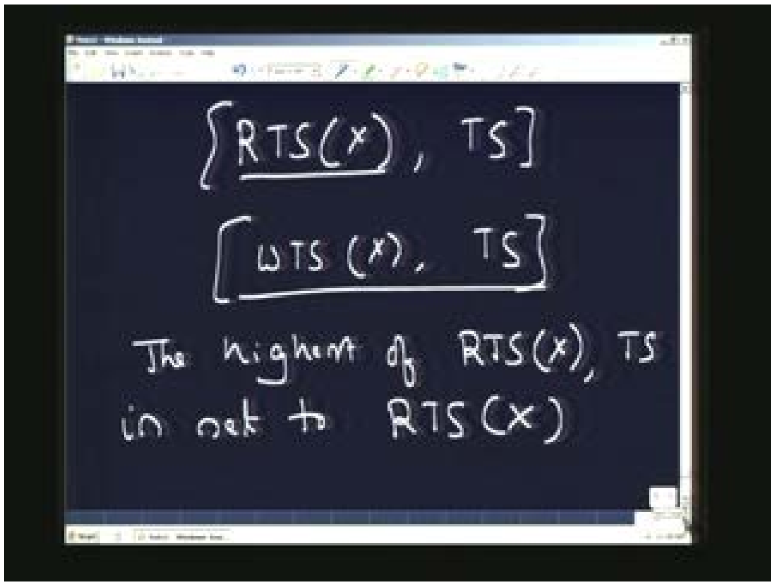

$[\underline{RTS(x)}, TS]$

$[\underline{WTS(x), TS}]$

The highest of RTS(x), TS is set to RTS(x)

That’s what actually happens by making these counters. Typically RTS and WTS are counters, data counters. They stored the value of the transactions that have highest transaction value that either read or written on this data items. Now let us go little more deeper and look at how exactly the basic time stamping algorithm is different from the two PL kind of an algorithm. I will take a simple example and show you where exactly the difference is going to come when you apply a basic time stamping algorithm verses the locking algorithm. If you remember write, we have actually taken a simple case of

(Refer Slide Time: 35:50)



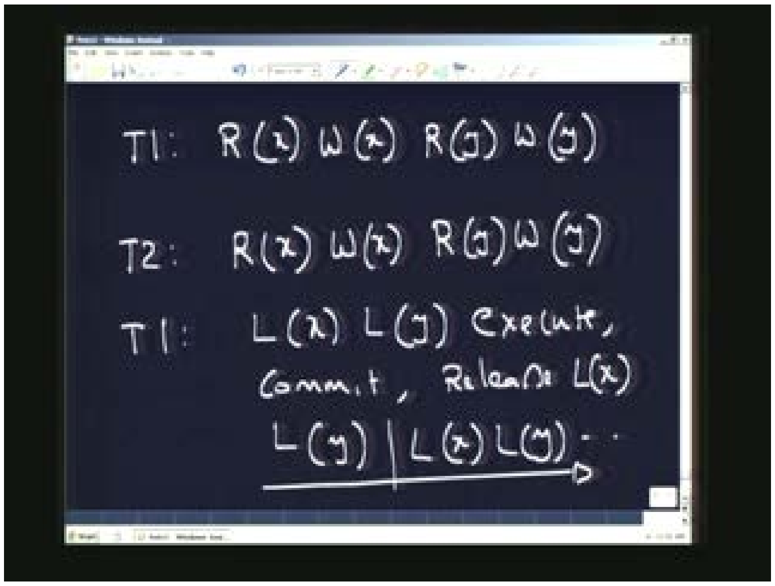
[RTS(x), TS]

[WTS (x), TS]

The highest of RTS(x), TS is set to RTS(x)

a transaction T 1 reading the value of x writing the value of x and then subsequently doing a read of y and then a write of y. Here basically x and y are the data items. Now I actually read the value of x, write the value of x then read the value of y and write the value of y. Now let us say there is another transaction T 2 which exactly does the same thing as done by T 1. Now what would have happen? If actually I applied a two phase locking kind of an algorithm. Now T 1 needs to acquire a lock on x and lock on y and after that it does whatever, then execute then commit. After that release x and y lock on x and lock on y. This is done at the end of the execution of the commitment. So you going to release the lock on x and lock on y after the execution that means only at this point of time T 2 can acquire lock on x, lock on y and then execute and this is only way T 1 and T 2 can proceed. So unless T 1 completely finishes releases its lock T 2 cannot produce.

(Refer Slide Time: 37:58)



T1: R(x) W(x) R(y) W(y)

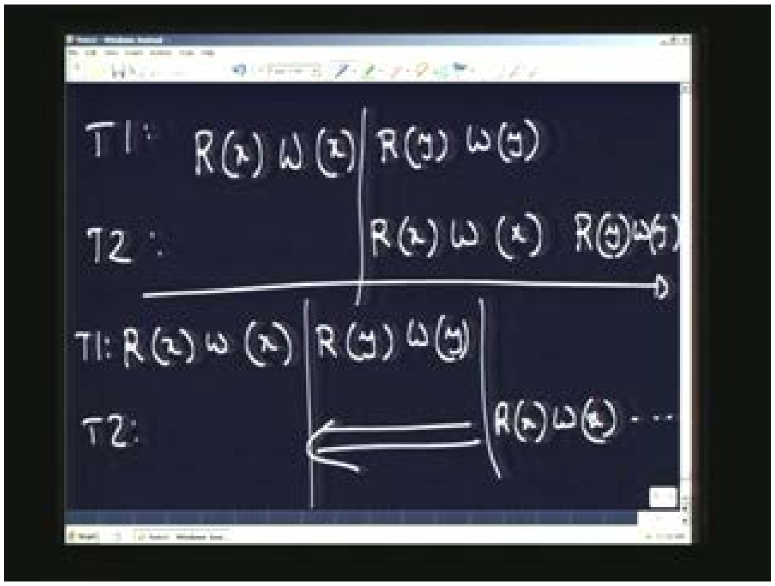
T2: R(x) W(x) R(y) W(y)

T1: L(x) L(y) execute,
Commit, Release L(x)
L(y) | L(x) L(y) . . .

This is very important thing that happen if I applied the locking criteria right but if you carefully observe after this initial point of  T 1 trying to do Rx and Wx, it is that point you can see there is no more use of x for transaction T 1 which means that it is possible for T 2 to start executing from this point not at the later point of time. Now there will be a read y write y here and then there will be read y write y. This is an optimal execution of this scenario, but this wont be possible apply the locking strategy, because locking would have required to two phase locking would have required you actually lock both these data item which means that is only at this point of time the lock on x will be released that means this will be shifted up to this point and you basically start executing the second transaction T 2 from this point.

So if you mark this is T 1 and T 2 you can see the overlap a significantly come down because you are not able to execute now. This is the zone I could have shifted the execution of Rx and Wx to this point but this wont be possible if I apply two phase locking and let us see if it is possible for me to actually to do this if I apply time stamping based algorithm.

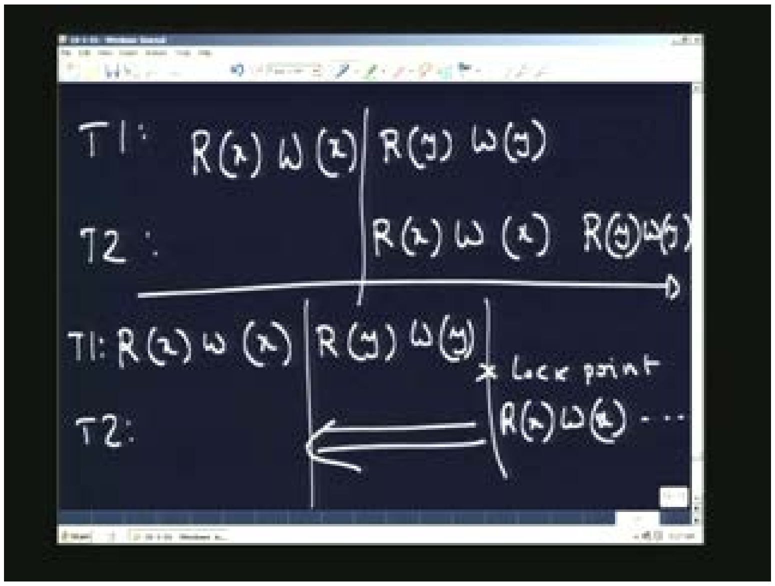
(Refer Slide Time: 39:38)



Now what would have happen? Is T 1 would have got a time stamp of TS 1 and T 2 would have got a time stamp of TS 2. Now all that condition that we have is TS 1 is less than TS 2 which means that we have a case where transaction T 1 has been able to get time stamp which is lower than transaction T 2. Now when they start executing at the end of at the end of execution of the first transaction T 1 it will write the value of time stamp on data items x. It is at this point of time the transaction T 2 will try attempting accessing data item x. Since the time stamp of TS 2 is greater than TS 1 as can be see here it is possible for T 2 to access data item x at this point of time.

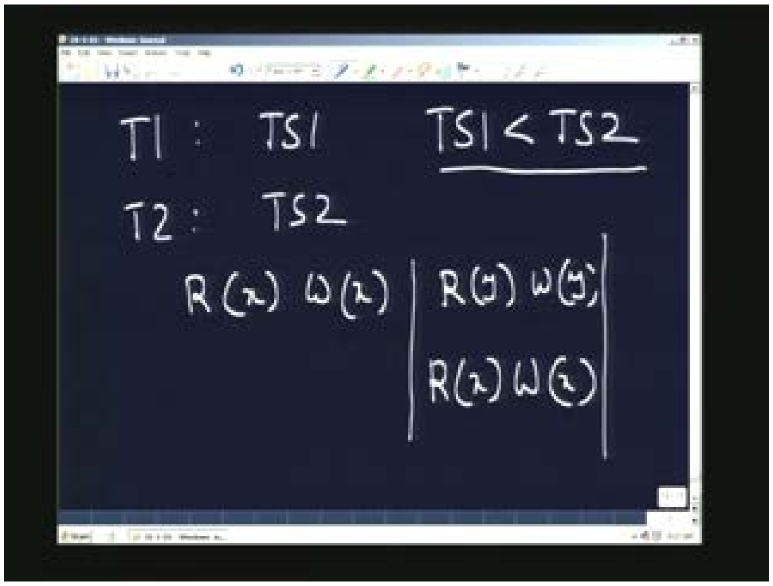
So what really will happen after this point is, you basically will have transaction T 2 accessing the data item x and then writing. Similarly here at this point will have y being accessed by transaction T 1, now the T 2 can start accessing y after this point of time. Now what can be seen here is that the overlap that we were talking earlier. To recollect what  we have been talking earlier is that its possible for the transaction T 2 to overlap with T 1 when transaction T 1 is finished with accessing x and now tries to manipulate y. Now this effectively prevented in two phase locking because T 2 can access lock only after it is released by T 1 and T 1 will not release lock on x till it actually reaches this point is basically the lock point for the transactions. So unless it reaches the lock point its not going to release the lock on x and hence T 2 will not be able to start executing till this point of time.

(Refer Slide Time: 42:27)



But whereas effectively, it can start executing from this point onwards that is what we actually looking at when we look at the time stamping algorithm all that matters here is the TS 1 is less than TS 2 and that is the order in which it will be allowed. Both x and y will be allowed to be accessed by the transactions manager.

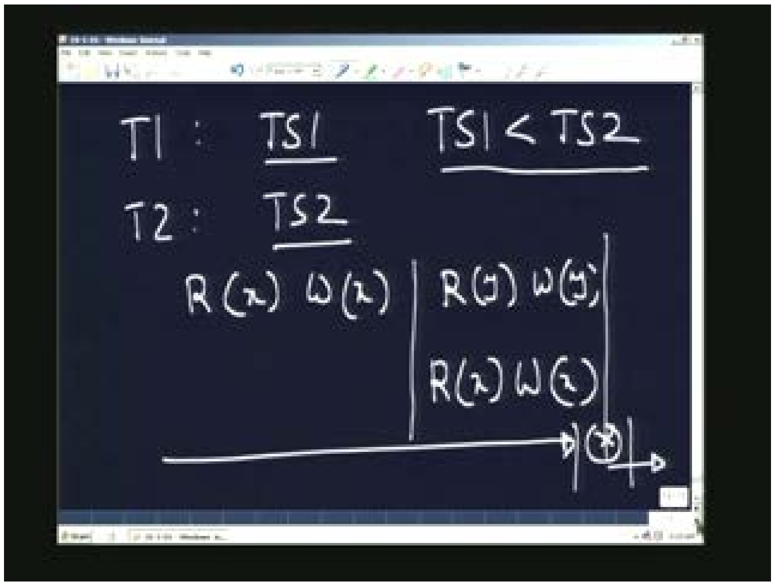
(Refer Slide Time: 42:45)



So in other words, both T 1 and T 2 will be executing in the time stamp order and this permits. In some cases more concurrency then what we seen in the two phase locking. but remember this is not straightly a optimistic kind of an execution because we still looking at how the transaction should execute by looking at the transaction Time stamps which were given at the beginning of the execution of the transaction not at the end right. So it is not fully optimistic in that sense. In a fully optimistic scenario, this would have been done by the transaction at the end of the execution.

For example: T 1 would have written all its value. T 2 would have written all its value and I will be checking which one should be sort of committing at the end of the execution that means both will execute to their finish and then I will actually use a validation point here and say which one of them will pass the validation and make that transaction commit whereas here, I am using the time stamp and using the time stamp to order these transactions in the beginning itself. I know the T 1 has the time stamp TS 1 and T 2 has got the time stamp TS 2 when it started executing and now if TS 1 is less than TS 2 that is order in which T 1 and T 2 will commit. If it is other way around, then the commitment is going to be T 2 before T 1.

(Refer Slide Time: 44:32)

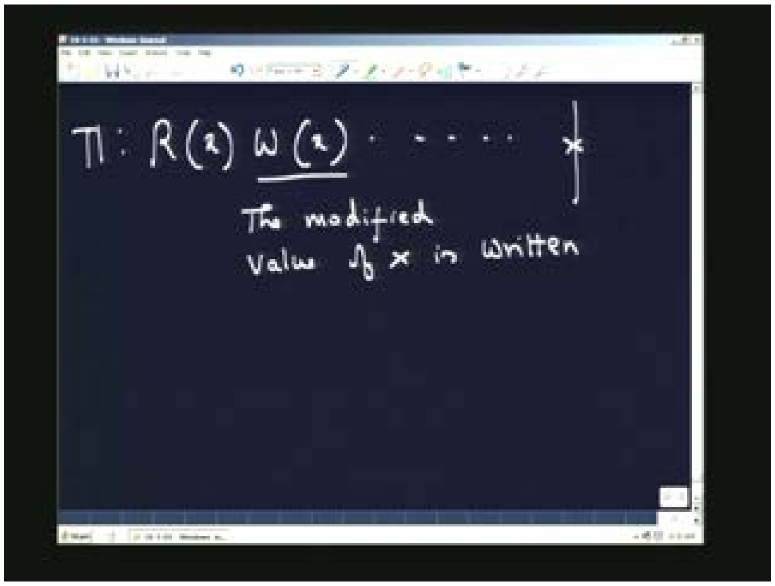


Now this explains what we see as a basic time stamping algorithm? How time stamps are used for concurrency control? Now one of the things that we still did not understand here is how this get integrated how the concurrency control gets integrated with the commit protocols. If you remember, we did this exercise even for two phase locking, when we integrating the two phase locking with commit protocol and that is the reason why we actually modify the two phase locking saying that the locks will not be released by transactions till the transaction commits because if it releases earlier the other transaction can look at the value and this will create cascading abort and other problems.

Now similar thing happens in the case of even time stamping algorithms. We need to see how the time stamping algorithms get integrated with the commit protocols. Now what will do is, we look at a simple mechanism by which the time stamping algorithms get integrated with the commit protocols. A simple exercise here will be to just look at not just how the transactions writes its value to explain the problem. For example, you can you can see here that there is a write x, there is read x followed by write x in the case of transactions x transactions one. Now if you take the transactions has returned actually the values at this point of time that is, it is actually continuing to execute the other things before it actually reaches the commit point.

Now if you understand the right x here the modified value of x is written this stage. The modified value of x is written is strictly not correct because the transaction is not still reached the commit point here. Now let us say at this stage of commit for some reason, this has to be rolled back which means that whatever the value that has actually been written here still needs to be undone which means that any body who is actually reading this modified.

(Refer Slide Time: 47:00)

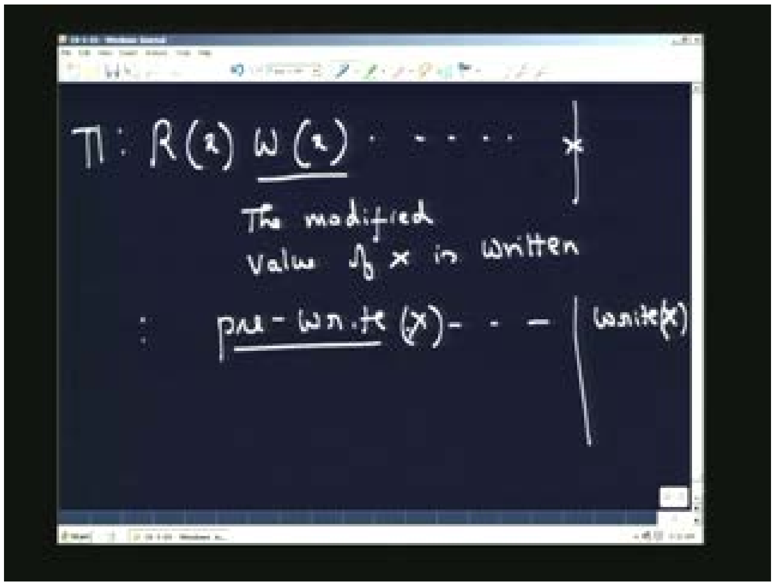


$T1: R(x)\ \underline{W(x)} \cdot \cdot \cdot \cdot \cdot \ \times$

The modified value of x in written

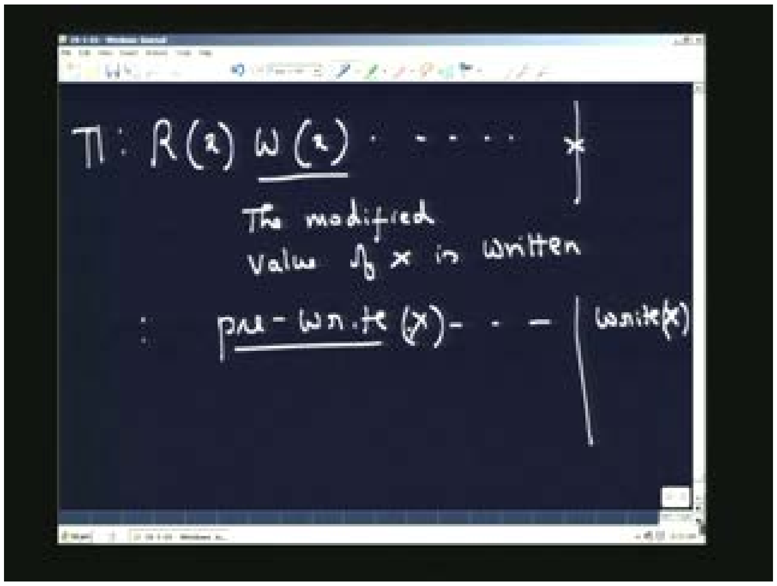
Transaction coming after this would be reading the value that is written by this T 1 and that potentially creates a problem in terms of how the transactions depend on each other. In affect we will be relaxing the concept of isolating one transactions affects on the other and that's what causes this difficulty of relaxing this. What we do in this case is? We will actually replaced this write in what we call as a pre write, that means every transactions to start with will issue not a write but a pre write that means this write instruction that we write seeing here will be a pre write and after exactly it wants to commit and it reaches this last point here. This is the point, it should use a write transaction that means the pre write on x and this is a write on x, the pre writes are not exactly written on to the database, not written on to the permanent storage but they are buffered.

(Refer Slide Time: 48:05)


T1 : R(x) W(x) . . . . .
The modified value of x is written
pre-write (x) - - - | write(x)

Pre writes are buffered and we will actually validate this pre writes and  make sure that the pre writes once accepted are not rejected at a later point of time from consistency point of you and when a write is actually issued by the transaction, updating its pre write its never rejected its always accepted, but the pre writes the transactions can still not commit a pre write which means that the commit stage it does not issue a pre write it doesn’t issue a write its possible that the transactions pre writes will all be rolled back which means that there is not going to be any affect on this pre writes on the actual database. What we will see is now a modification taking pre write in to account, how the pre writes will try to solve the problem for integrating commit protocols, with concurrency protocols.

(Refer Slide Time: 49:07)



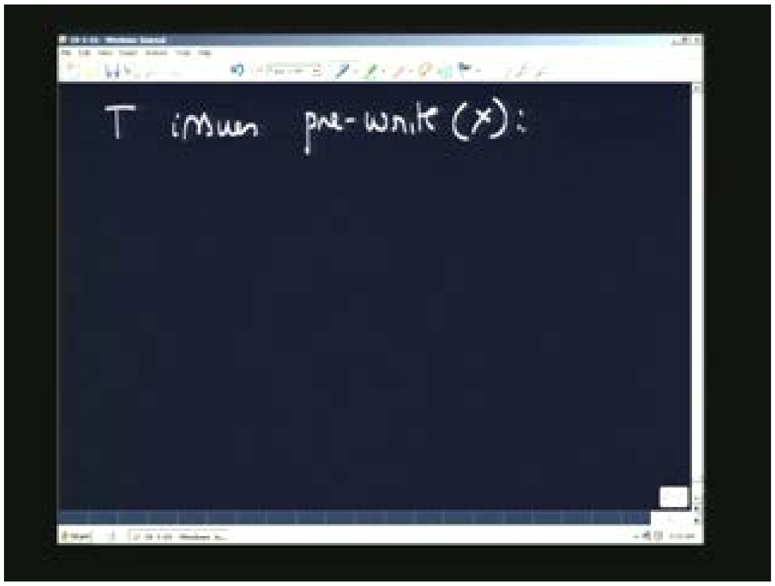
T1 : R(x) W(x) · · · · ·

The modified value of x is written

: pre-write (x) - - - | write(x)

Now if you remember we are actually two checks, when a transactions issues. Now let us say T issues a pre write. Now this pre write has to be checked for pre write on a data item x.

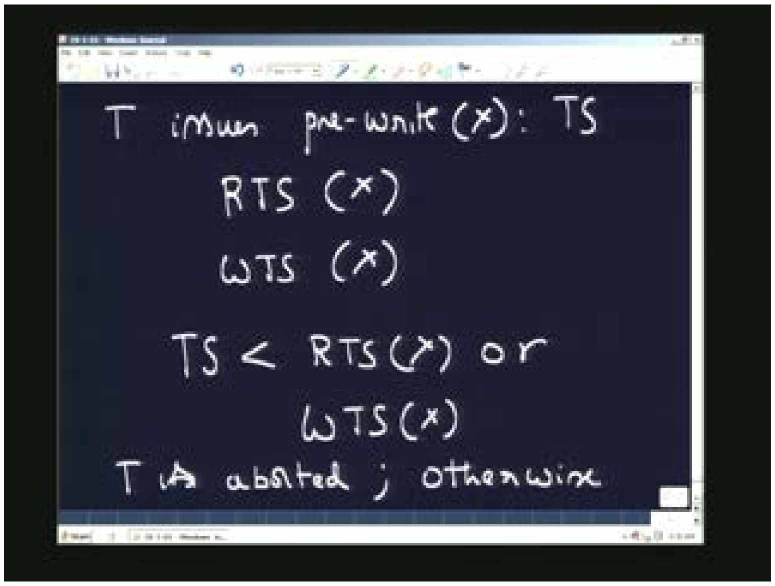
(Refer Slide Time: 49:24)



T issues pre-write (x):

Now this has to be checked for on the database. Items on the database times stamp now this will be checked again is the read time stamp of data item x and also, write time stamp of data item x. Now if the pre write is pre write time stamp of the transactions is less than either RTS or WTS of x that means it is actually less than the write time stamp or the read time stamp, then T is aborted.

(Refer Slide Time: 50:24)



T issues pre-write (x): TS

RTS (x)

WTS (x)

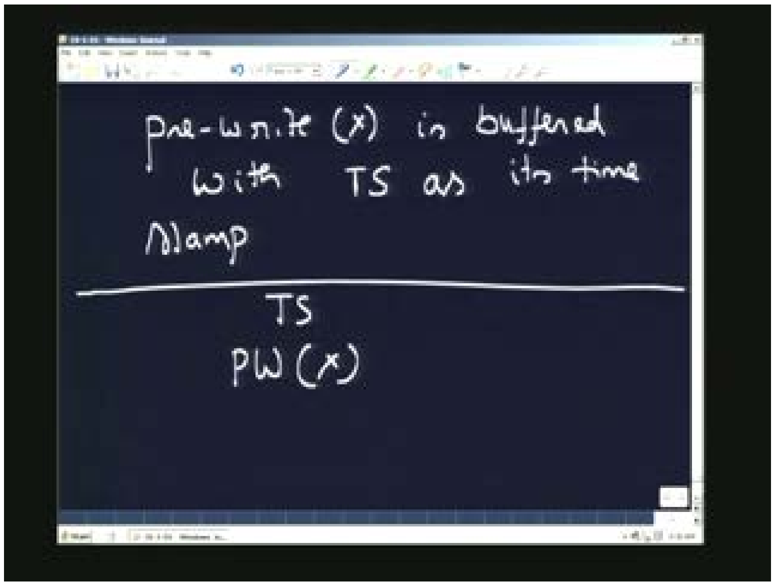
TS < RTS(x) or

WTS(x)

T is aborted ; otherwise

Otherwise we pre write is buffered. Pre write is actually not written but what is done here is pre write x is buffered with its corresponding time stamp buffered with TS. TS has its time stamp okay as its time stamp okay. Now what is this mean? This means essentially the following, all the time stamps that we are talking about here are, with respect to this buffered item here x. For example; if x is actually pre write, pre write on x is buffered with time stamp TS. Now any subsequence reads that we issue here needs to be checked again this pre write.

(Refer Slide Time: 51:24)



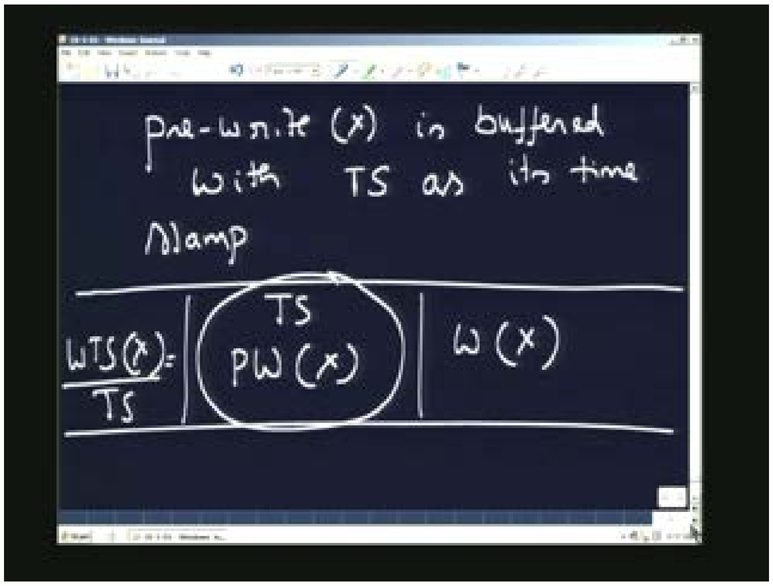
pre-write (x) is buffered with TS as its time stamp

TS

PW (x)

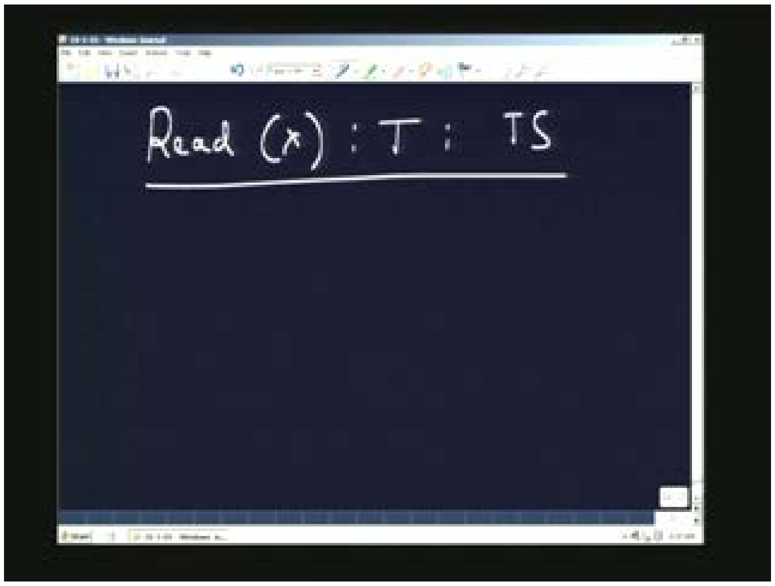
Now at a later point, this pre write becomes a write on x, then it is at this point of time the actual WTS of x is updated to the corresponding TS okay that means this will stay like this for a while in the buffer when this actually comes write comes on x it is at this point of time it will be updated to a write time stamp of x. Now as these pre writes are buffered as the reads comes in to the system reads of a transaction come in to the system, we still needs to check those reads again as any of the pre writes that are already buffered in the system.

(Refer Slide Time: 52:15)



Now let us understand how the reads. Needs to be modified in this particular case; now if you typically look at read of x issued by a transactions T with a time stamp TS is how we will read this. Now this read x time stamp is TS. Now if you remember earlier, this will be checked the write time stamp of x and if the write stamp of x is less than the TS will allow the read to proceed because any number of reads can be done on the data item. Does not really matter, as long as the read time stamp is higher than the write time stamp.

(Refer Slide Time: 52:57)

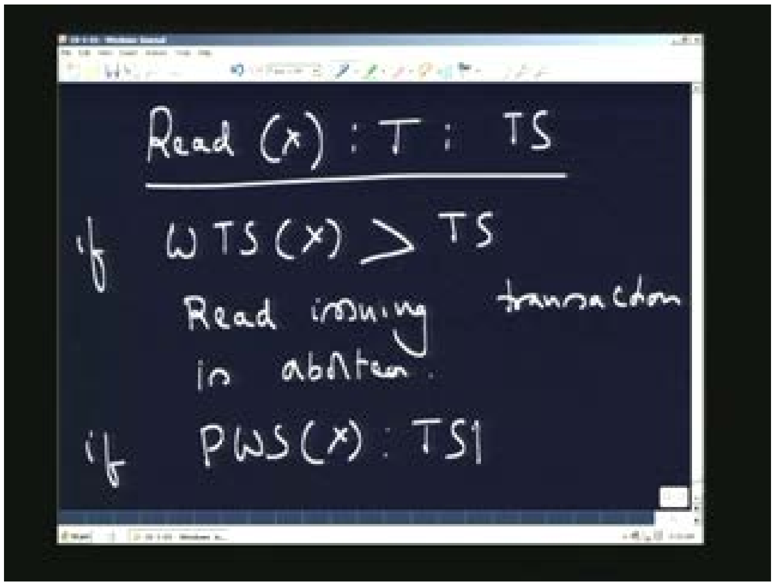


Read (x) : T : TS

because reads can all be done concurrently where as the writes have to be exclusive. Now in that sense it does not really required to be checked again the read time stamp what you need to do is you need to check only the time stamp of x in this particular case again as the write time stamp of the x. Now if you only do write time stamp of x is less than TS. I think I will put other way around which will make things, if write time stamp of x is greater than TS then basically this read has to be rejected. For obvious reasons, because we are actually read after some other transactions which came after you will be actually produces the write value the read issuing transaction is aborted transaction is aborted.

This is same as condition of the earlier one but other case read is allowed in the in this case the read is still not allowed when you actually have a pre write buffered okay. Now what is this pre write buffered means the pre writes buffered means if i have typically there is a pre write time stamp on x with a TS. Now I have to check this time stamp. Let us say it is TSP to just indicate that now this TSP is actually less than the TS.

(Refer Slide Time: 54:45)



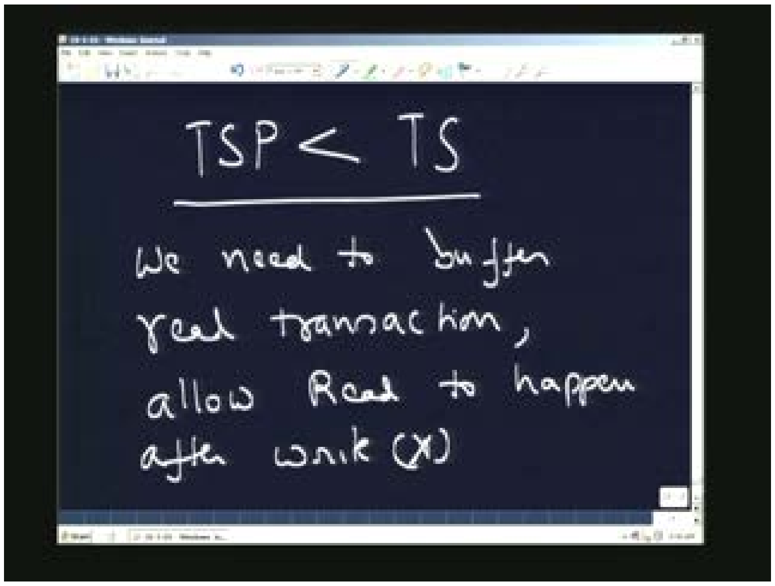
Read (x) : T : TS

if WTS(x) > TS
Read issuing transaction
in abotten.

if PWS(x) : TS1

The TS is the time stamp that I am actually trying to read what does this indicate? This condition indicates that there is a pre write buffered and that please remember that pre write will never going to be rejected when the actual writes comes in. So I actually need to buffered in such a case we need to buffered we need to buffered the read transaction. Read transaction that means you actually postponed read transaction and allow read only after the write has been committed. Allow read to happen after the write happen after the write actually comes write on x comes. This ensures that this typically will ensure that the read of the transaction happens after the writes affects.

(Refer slide time 56.02)

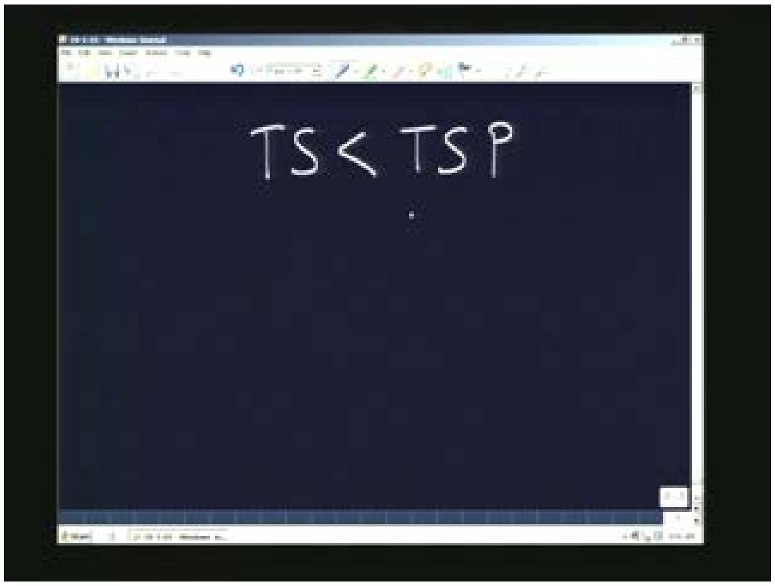


TSP < TS

We need to buffer read transaction, allow Read to happen after write (X)

Now the other conditions were the read on the transaction TS is less than the TSP that's going to be the other condition, then you can allow the read to happen because this in this particular case there is no pre buffered writes on the transaction and hence this TS can be allowed to proceed.

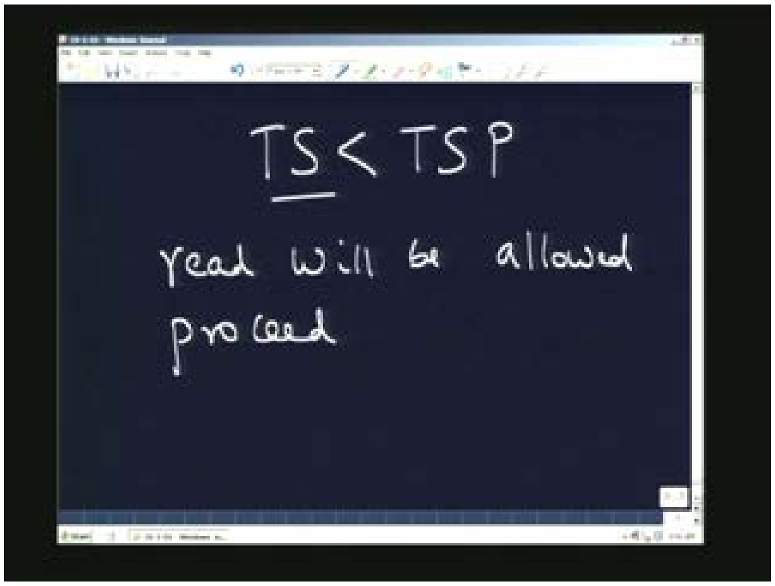
(Refer Slide Time: 56:14)



TS < TSP

read will be allowed to proceed will be allowed to proceed. Now what this shows is? It requires just a minor modification in terms of how we handle when we want to integrate both commit and the time stamping protocols.

(Refer Slide Time: 56:38)



$\underline{TS} < TSP$

read will be allowed proceed

All that we have to do additionally do is, we need to actually make sure that the writes are handled properly in this case by ensuring that they do not actually write on to the database to start with and produce the pre write. What we will do in the next lecture is, we will look at host of other protocols which are most optimistic than the time stamp based protocols in the next lecture.