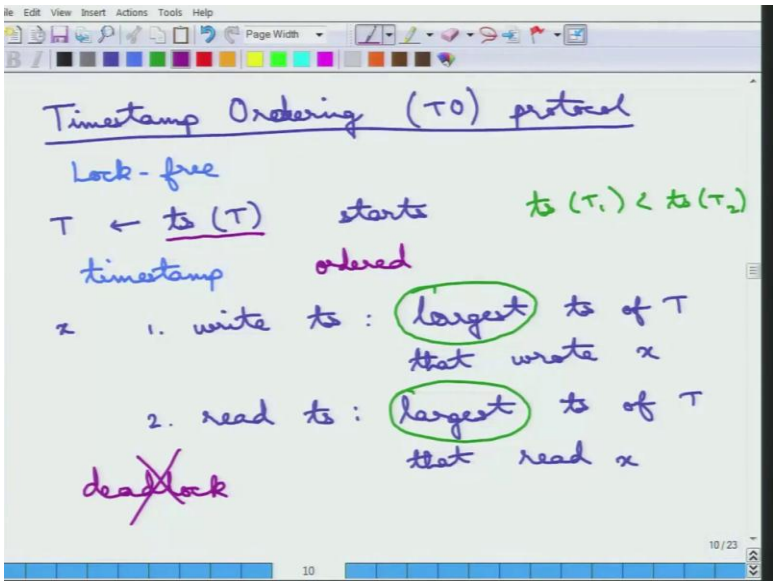
Fundamentals of Database Systems
Prof. Arnab Bhattacharya
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

Lecture – 40
Concurrency Control: Timestamp Ordering Protocol

Next we will move on to another Concurrency Control Protocol, which is the Timestamp based and very importantly this does not use locks, it is the lock free protocol.

(Refer Slide Time: 00:19)



So, this protocol is called Timestamp Ordering or the TO protocol. So, timestamp ordering protocol, this is very importantly this is lock free, so it does not use any lock on the data item. So, first of all what is the timestamp. So, at each transaction is assigned a particular timestamp when it starts, so when the transaction starts it is assigned a timestamp, so this is called the t s of T is essentially the timestamp of T.

So, a timestamp is a logical counter, it is a logical time, it can be the actual wall clock time or any logical counter; such that, this is ordered, this is… So, the timestamps are ordered and it is any logical clock. So, this is an ordered logical clock, so very simply every time tick may be given and in which other timestamp want, then two and so on and so forth. And it can be anything, as long as it is ordered and ordered meaning, so between two timestamps it can be always determine whether the timestamp of the first

timestamp is lesser than the second timestamp or equal, that can be always done, so that is why it is ordered.

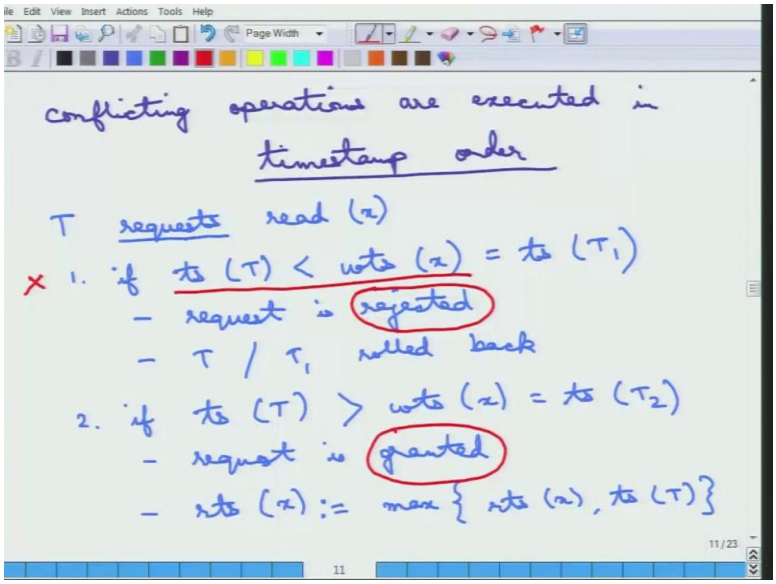
So, that is the timestamp for the transaction, so this when it starts. So, when the transaction starts; that is given a timestamp; that is the timestamp of the transaction. Now, for every data item x, so there are two timestamps that are maintained. The first one is called the write timestamp, which is the largest timestamp of any transaction that wrote x. So, that wrote x successfully, but essentially, so what does it meant, that x is the data item and suppose transaction 1 wrote it and transaction 2 wrote it both, so it is the largest timestamp.

So, in the transaction if transaction 1 starts before transaction 2, then the timestamp of transaction 1 will be less than the timestamp of transaction 2; that is because it is ordered, etcetera. And if both of them suppose as written to then x, then this will get the largest timestamp, so this is important of this is the largest timestamp of any transaction that are successfully written to x. And similar to the write timestamp there is of course, a read timestamp, which is the same definition, it is the largest timestamp of a transaction T, that successfully read x. So, once more this is the largest timestamp of something that has wrote to x and this thing.

So, the timestamp ordering protocol uses only this timestamp. So, it only uses the timestamp and we will see that, but essentially it can be shown that type, so the protocols that use timestamps can never deadlock, because it… So, there is always a strict ordering between the transaction that has started, so transaction 1 if it starts before transaction 2, it must be having a lower timestamp than the ordering 2.

And, so it can never deadlock, because there is an implicit priority order of the transaction based on the timestamp. So, timestamp ordering protocol or for that may be any protocol that you can think of that uses timestamps will not deadlock. So, this is the very important property of that.

(Refer Slide Time: 04:03)



So, coming to the actual protocol, what it does is that it the basic idea of that is that, the conflicting operations, the basic philosophy of the timestamp ordering things is that, the conflicting operations are executed in the timestamp order. So, if there are two operations i 1 and i 2 and then, i 1 has a lower timestamp then it is executed in that particular order, are executed in timestamp order. So, this is the important part of it, this is the of course the intuition or the philosophy of what the timestamp ordering protocol is and we will go to the details of it next.

So, suppose this is the ((Refer Time: 04:49)), so a transaction requests a read. Note that this is a request, because it may or may not successfully end doing it, correct. So, it is just a request a read on x, now couple of things may happen is that the following things may happen is that, if transaction, the timestamp of transaction T is less than the write timestamp of x. Then, what is the essentially the meaning is that, the write timestamp of x essentially says that, there is a transaction. What is the write transaction ((Refer Time: 05:28)) if it is a timestamp of a transaction and that is written to x and that transaction is greater.

So, the timestamp of that transaction is greater than the timestamp of this transaction T; that means, that write has essentially should not have happened before this read. So, essentially this read is late and there is somebody written, some other transaction which was supposed to write later has already written and only now the read request for this

transaction is coming. So, this is not correct, because this is now going to read, if it now reads it is going to read, what has been written by the timestamp, the wts of x.

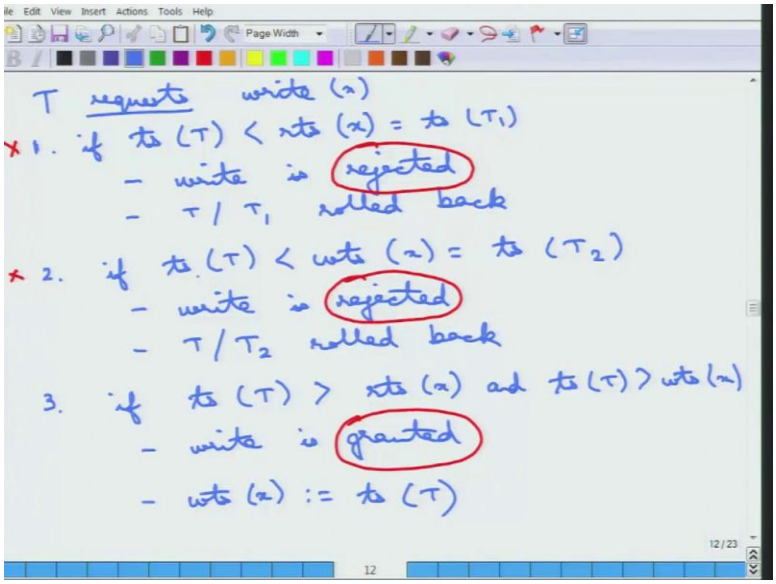
So, just to elaborate this further, so suppose the write timestamp is essentially the timestamp of some other transaction T 1 that has written. So, essentially transaction of the timestamp of this T is less than the timestamp of the transaction 1 that is written, so it should has read the value before T 1 got a chance to write it. So, it has not done it, so this is wrong, so this operation this is not going to be allowed.

So, what it does is that, this request is rejected and, so this request is rejected and this T or T 1, whichever one of them must be rolled back, because they have not done read in the order that they were supposed to do. So, T the read of T should have happen before the write of T 1, but they have not done, either this happen, so this is not correct. So, essentially this is the whole point of this happens, then this is not correct, so then it cannot allow, so this request is rejected, this is very, very important, so this request does not go through.

On the other hand, if the timestamp of transaction T is greater than the write timestamp of x, which is let us say that some other T 2, then there is no problem. Because, T 2 was supposed to write to this x before t s has read, because the timestamp of T 2 is lesser than the timestamp of T and that is what is being done, so this is fine. So, this is simply request is granted, request is granted meaning the T is allowed to read x, which also means that the read timestamp of x may be need to be updated.

So, the read timestamp of x is then updated to the… So, it is the largest read timestamp, so it is either the, whatever it is already there or the timestamp of T. So, essentially the timestamp of T is greater than the read timestamp, then it is updated otherwise it remains the old thing, so that is the thing. So, this is, very importantly this is granted. So, that is the read request; that is what happens when the read is, when a transaction T request the read.

(Refer Slide Time: 08:38)



The next thing is when a transaction T requests are write, so this is again just a request, request are write of x. The first thing is that, if the transaction if the timestamp of T is less than the read timestamp of x. Suppose this is equal to transform timestamp of T 1, this means that again the same problem as in the earlier case. So, the read of timestamp T 1, T 1 has already read it, now T 1 should have read it before the T has the chance to write, because T is earlier, the timestamp of T is lesser than the timestamp of T 1.

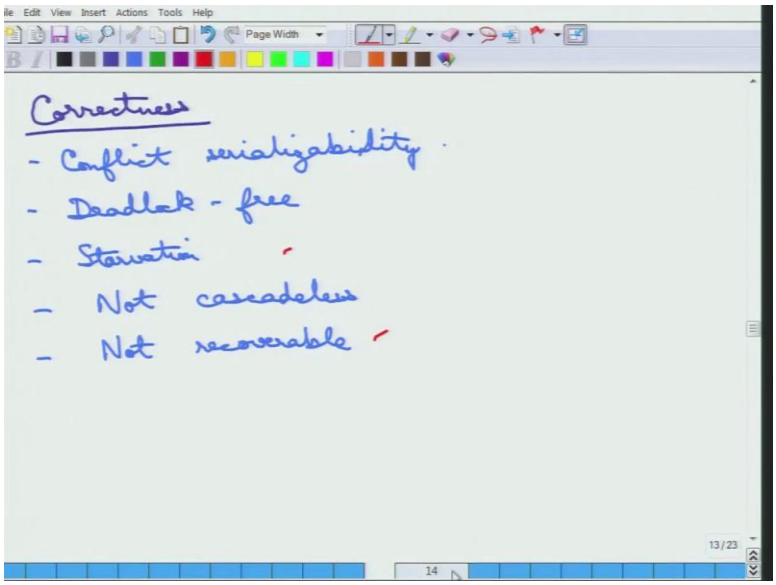
So, T should have written before the read of x by T 1 has done, so again this is wrong, this should not be allow, so this write request is rejected, this write is rejected and the same thing is that T or T 1, one of them is rolled back that is the same idea, because there is a read write conflict and it has not progressed in the correct orders, so this is rejected, so this does not go throw. The second one is that if the timestamp T is less than the write timestamp of x, so let us say this is the timestamp for T 2.

Again the problem is T should have written before T 2, because the timestamp of T is lesser than the timestamp of T 2. So, the write to x for T should have happen before the write of x by T 2, so the same thing, so this write is again rejected. If these are write, write conflict, the previous one was read write conflict, so this is again rejected and then, the same thing is that T or T 2 is rolled back. So, this is rejected and this operation, this has been go through.

And finally, transaction T is greater than the read timestamp of x and transaction and the timestamp is greater than the write timestamp of x. So, which means that there is no problem, all the read that should have earlier has gone earlier and all the write that has gone earlier that has gone earlier, so this is simply allowed, so write is granted. And similar to the previous case, the write timestamp when now need to be updated, so the write timestamp of x is needs to be updated, but now you see that the write timestamp is just needs to be updated to t, because there cannot be any other timestamp which is greater than T.

So, simply this can be taken as the write timestamp of T, so that is the two things. So, these are the two important parts, if that when T request read and write, then what happens essentially. So, very simply if a transaction comes after and then, it is in the correct order, then it is granted otherwise it is rejected.

(Refer Slide Time: 12:09)



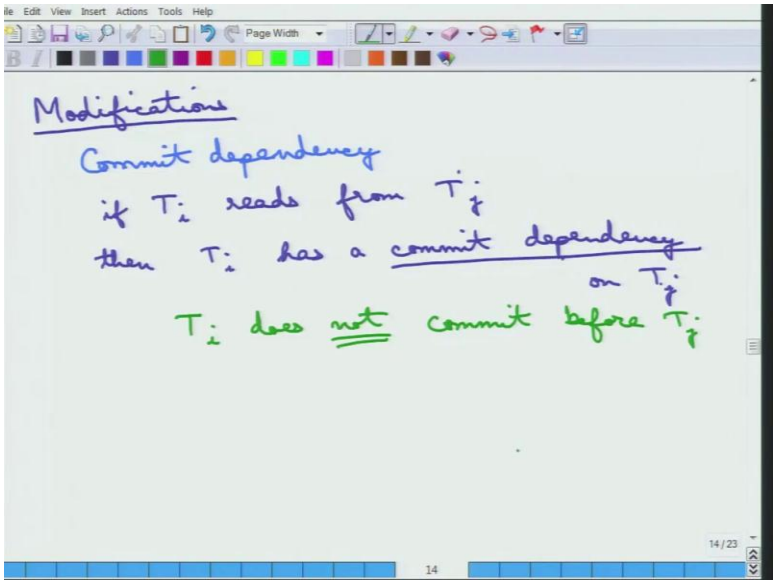
Now, all this is fine, we now need to argue a little more formally about the correctness. Why is this going to be correct? So, the correctness of the timestamp ordering protocol can be argued. So, this first of all, this guarantees conflict serializability, so this is guarantees conflict serializability; that is not probably very hard to see, because it does not allow any conflicting operations to go through that. Hence, the whole idea of the protocol, why it rejects this thing is, because it was violating those conflicts, so that is the thing.

So, essentially the idea is that the conflicting operations are executed in correct timestamp order, if it is not, then if it appears, so if it is requested in a non, if it appears in a non timestamp ordering what it should be, then it is rejected, that is called first thing. The other thing is, when I say there cannot be any deadlock, because the transaction that started earlier always gets the priority, so all these things the transaction with a smaller timestamp is always going to get the priority.

So, dead lock free, so this is called dead lock free, so but there may be starvation and what it happen is the same situation is that. The timestamp there may be other transactions with lower timestamp they keep on coming and the particular transaction is just that keeps on waiting independent. It is always rejected, rejected, rejected and so on and so forth. It may cause starvation and it may not be… So, it is not cascadeless, which means that, so essentially it means that cascading roll backs may happen, because the particular transaction aborts, the other transaction that is dependent on it may also abort, so it not cascadeless.

Let us in fact it is very interestingly it is not even recoverable, because it has allowed the particular a timestamp, it has allowed a particular transaction to go ahead read or write, because the other transaction has not yet come and that transaction may be just have committed or aborted, so it may have just committed and not aborted, so then it is non recoverable. Because, later we find that it should not done then, but by the time it has already committed. So, this is very important, but this is not recoverable. So, although it is deadlock free and it guarantees conflict serializability, it still has these problems of starvation and non recoverability; of course, if it is not recover it is not cascading as well.

(Refer Slide Time: 14:38)

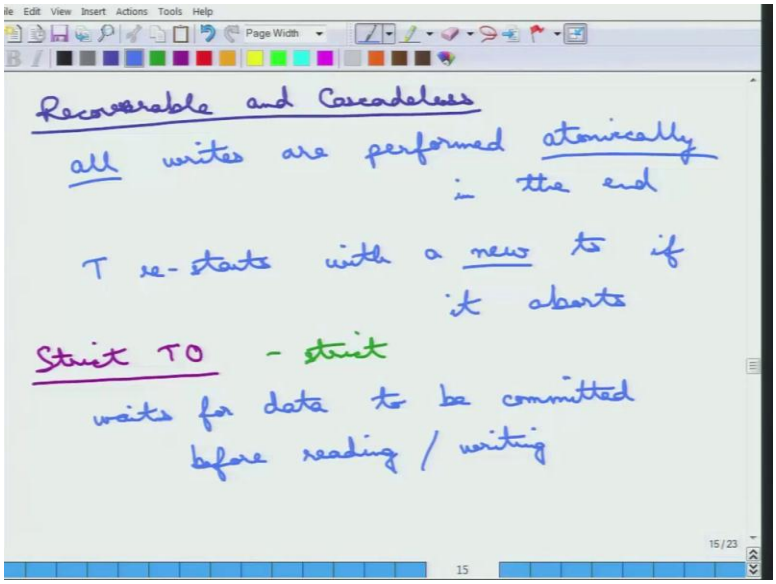


So, there are sudden modification of course, there are certain modification has done to the basic timestamp ordering protocol first of all you can use something called commit dependency there is the concept of commit dependency. So, this is the essentially when sure it is that the transactions are recovery what is the commit dependency is following if T i reads from T j then T i has the commit dependency on T j which is essentially saying that.

So, the dependency of T j; that means, that T i has reads what is the meaning of that T i has produce the value of T j has to read then T i has the commit dependency on T j which meaning that T i cannot commit unless the T j has to commit. So, unless T j commit depends on whether T j commit or not. So, that is called commit dependency. So, essentially ensure that ensure T i does not commit before T j does it does not commit because it is dependent on T j.

So, unless T j commit it does not commit that is the ensure that it is recoverable transaction otherwise what is recoverable in tid then the T i commit then T j aborts T i has already committed T j has wrong commit because T j has value that should not to be happened. So, that is what wasting time correct.

(Refer Slide Time: 16:30)



Now, couple of more thing. So, these things recoverable and cascade less, so this is to making this recoverable and cascade less. So, there can be certain modification done to this thing making this recoverable and cascade less. What can be done is that all writes are performed in the writes are performed in the end atomicity are performed atomically. In the end again, what is the atomically in the end means all the writes go through or none of write are go through, so write it is done in the end.

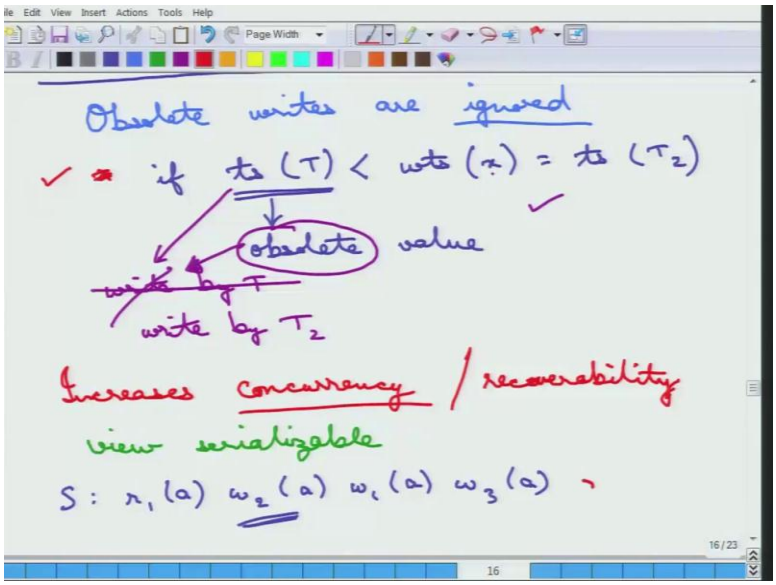
So, essentially we saying that all the writes will be done unless all the writes will be done in the end. So, first of all read in the sense then the writes is done. So, it can be that if the transaction aborts then what happens if the transaction aborts that just restarts then the new timestamps. So, that is higher timestamp and then the commit dependency etc the commit the timestamp because it is now starts with higher timestamp all the transaction that has started earlier we will get the priority about this and because about this and we can try to finish it and so on, so forth.

So, that is one way of doing the other way of doing of that let me do that write it down transaction starts with a new timestamp if it aborts. So, it is new timestamp if it aborts this is the two important things the other way of making this is using locks. You can see that is going to against the philosophy of why timestamp ordering protocol is needed in the timestamp because it needed in the lock free.

So, to use locks to make it recoverable and cascade less essentially having just locking kind of protocol is not a very elegant solution good solution anyway and then there is one version of it called strict timestamp ordering protocol which use this essentially try to make the protocol strict version of the its Waite for data to be committed. So, this essentially waits for data to be committed before it is read or write. So, essentially read or write any uncommitted data.

So, only reads committed. So, that is essentially that meme king the strict schedule. So, it will only read or write a particular data for which the transaction is read or write for which is committed; that means, that is correct then only it read and write. So, that is all serial.

(Refer Slide Time: 19:31)



So, that is. So, this is about the timestamp ordering protocol there is one extra thing that can be done which is called the Thomas's writing rule. So, Thomas's writing rule is to make it little bit more concurrent. So, it is to allow some more schedules which will be not allowed by the timestamp protocol. So, remember what will done for conflict serializability, serializability we did not take care of the blind rights.

So, blind rights, so the same things of absolute rights here the absolute rights is not blind rule absolute rights are ignored. So, we says the absolute rights are simply ignored. So, what does the absolute rights means remember what happen in the right cases is that if the transaction that wants to write timestamp is lesser than the read timestamp of this

transaction is T 2 in the basic timestamp ordering protocol it was rejected because this is because the writes are coming out in this case the Thomas's write rule then the Thomas's write rule which is allowed.

Because, the idea is that T is trying to write the absolute value why it is an absolute value the reason it trying to write an absolute value. Why this is called an absolute value because, T suppose T would have actually written the value of x now the another transaction two which would come after that write by T will be overwritten by the write by T 1.

So, this was absolute anyway it is absolute it is not a problem is that when the absolute is there, which simply ignored, which simply ignore the absolute right. There was a simply… ignore the loss of the absolute right anyway there is an absolute right when the protocol just its goes to then essentially means is that, what is that seed conditions of the write rule. Two of them does not pass through the one pass the middle condition may also to be passed because absolute write.

So, then the absolute write is ignored. So, this increases the concurrency. So, this increases the concurrency of this because it allows some more writes to go through it allows some more transactions to go through successfully complete and commit etc it increase the concurrency and it also increases the recoverability now what is happen that the transaction will not simply aborts it can continue. So, it may be increasing the recoverability, so as the note on this.

So, the allows the sudden view serializable things can allows certain view serializable things it otherwise allowed by the basic things because you can see absolutely this is the new rule the absolute rule is absolutely difference between the conflict and the view serializable things, so here is the small example that will halite this. So, suppose this is r one a w 2 a w 1 a and w 3 a now you see that w 2 is the absolute write and this will allow to the Thomas's right rule, but not otherwise this protocol is allowed. So, that is the end of the timestamp ordering protocol.