Database Management System
Prof. D. Janakiram
Department of Computer Science & Engineering
Indian Institute of Technology, Madras
Lecture No. # 21

Concurrency Control – Part -2

We are continuing discussion on execution of transactions in database systems. We in fact looked at various things like how the execution of the transactions to be controlled in the database systems to produce consistency results. In fact transaction execution is one of the most important things in database system because it affects the performance of the system. If you typically notice during the results time when cbse results or state board results are announced, now lot of people try to access the results through the net. The results is stored more or else on a on a database system. What you will realize is a large number of people simultaneously trying to access these results at the same time that is very important, at the same time because everybody wants to know his result or watch result at the same time as somebody else because everybody is curious to know about the results.

Now this is the time when there are simultaneous hits on the database system, the large number of people were trying to access these systems at the same time. Now normally we notice that it is at this point of time the system fails, no that's the time when it cannot respond to so many requests at the same time which is equivalent to saying that the throughput of the system, the number of simultaneous users it can cater to is going to come down drastically. That's the time the system will be challenged in terms of how many number of transactions it can process in a given point of time that is per second how many transactions you are able to execute on your system.

Another very interesting example is our railway information system. If you have noticed in summer there is a lot of people trying to book the railway tickets online or go to the counters and book their tickets. You can notice that the number of users trying to book the tickets in the last you know couple of days, this being summer seems to be peeking at 30000 per day which is you know, the system should be able to handle that many number of peoples simultaneously coming and try to book their railway reservation simultaneously. So database systems, one of the very critical things in database systems is how many number of transactions the database is able to execute without losing consistency.

Now it's very important that if one user has booked his ticket, somebody else simultaneously accessing the ticket overwrites all the details and the same seat is allotted to two people at the same time, the same birth is given to two people. Then the system is in an inconsistent state. When one user is trying to book and he gets the birth, it should be guaranteed that nobody else is going to get the same birth that's were basically we were trying to understand how the database actually solves the problem of trying to give as

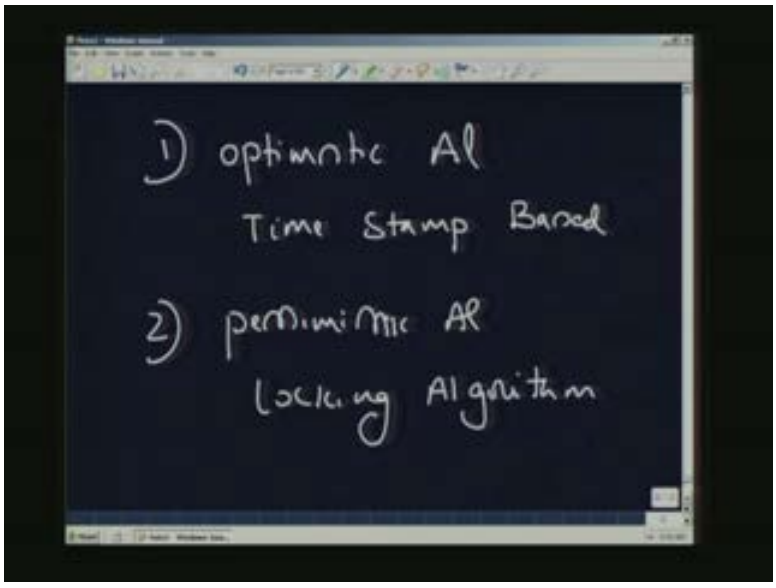
much as throughput as possible and at the same time maintaining the consistency of system state. This is the most important requirement of database systems.

What we did in the last class was we were looking at two phase locking, a very populated protocol that is employed by database systems to ensure that when simultaneous transactions are executing they log the data items. For example in this case, if you what we are discussing few minutes back on the railway reservation case the birth will be for particular train if it is locked by one transaction, it will not be allowed to be locked by another transaction till this transaction finishes operating on that particular item. That is how exactly the lock coordination is done by the transaction manager to ensure that more transactions can be executed but at the same time consistencies ensured.

Unfortunately what we have observed in the two locking phase is beside it is prone to be dead locks because when you locked its possible two different transactions could have locked themselves in a different way and that could have resulted in a deadlock scenario. And it is also possible that since the locks are released only after the commit point, it is possible that more throughput cannot be achieved by using two phase locking that is were basically we see that two phase locking needs lot of you know, those kind of algorithms need change to ensure a better throughput. Two phase locking suddenly doesn't gives that much throughput and so if you actually want to improve the transaction throughput, possibly you should explore other algorithms as well.

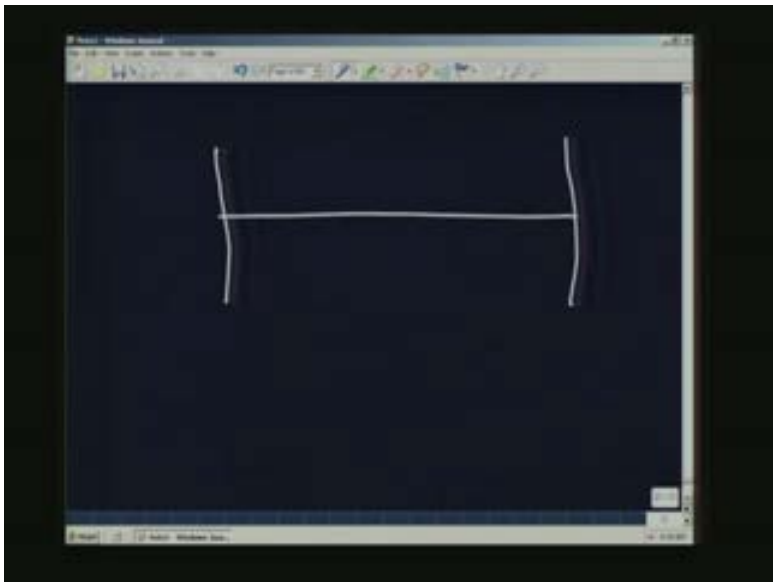
Now one of the things that we did in the last class was to actually classify the algorithms into optimistic algorithms and pessimistic algorithm and we showed that the optimistic class of algorithms, we basically use what is called the time stamp based algorithms whereas in the pessimistic we have basically the conflicts are more, we said we basically using locking algorithms. This is what we actually did in the last class and we said, we will continue to look optimistic based algorithms in this class.

(Refer Slide Time: 7.11)



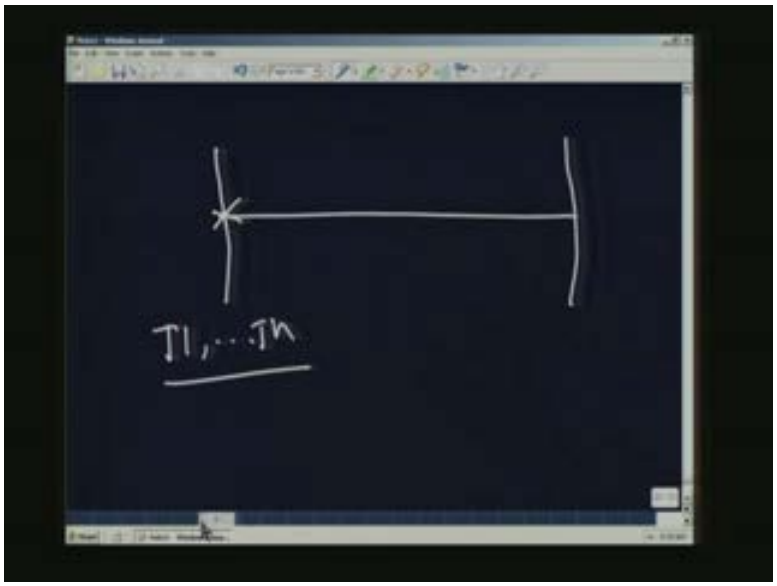
And we are going to look at typically the time based algorithms, they are very popular algorithms in terms of producing a better throughput. I am going to compare both optimistic and pessimistic algorithms in a little more detailed fashion for a few minutes now and then get down to look at time stamp based algorithms in much more detail. But before we go in to time based algorithms, let us understand essentially the difference between pessimistic based algorithms and optimistic based algorithms.

(Refer Slide Time: 8.28)



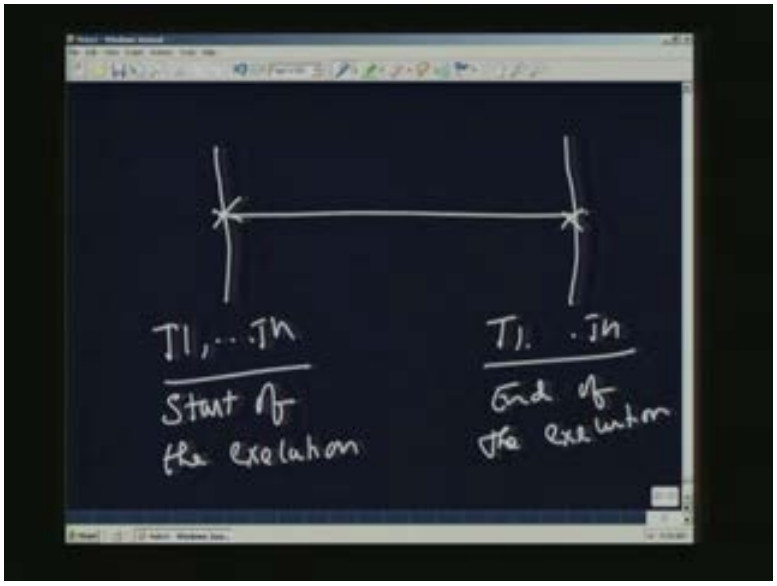
Now really what happens in the case of pessimistic and optimistic based algorithms case is they are in two different spectrums in terms of what actually they do for checking the consistency of the database transactions. Notice that when actually we admit the transactions into the databases if you admit only those transactions that are going to be consistent. Then you have actually doing the admission control for consistency in the beginning of the transaction execution.

(Refer Slide Time: 8.49)



Now it's possible for you to let the transaction execute till the last point and start doing for the consistency check for the transactions at the end of the execution. So it's possible for you to actually do the consistency check at the start of execution versus the end of the execution. Now if you basically do the check at the end of execution, you typically waste the execution time of the transactions. For example it's possible that a transaction is actually executed up to it finish and it is at the point of time, it is told that the execution is not consistent with respect to other transaction execution and that transaction is aborted. Then it is like saying that you did lot of work but some body says at the end of the work, whatever you have done is not consistent, so please come back again and redo whatever you have done.

(Refer Slide Time: 9.11)

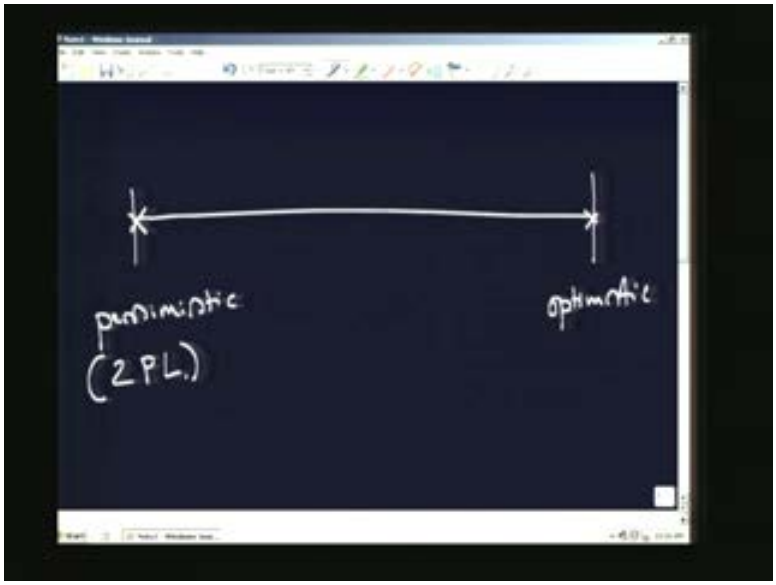


So that is one way of ensuring that you know the consistencies is applied at the end of the execution. On the other hand everything you do from the beginning, a teacher is by your side and looking over your shoulder and looking at every step what your doing and if you actually took a value that is not actually a correct value, he actually stops you at that point and says, wait till you have the correct value and then you are executing then basically that is what we actually see it as a very conservative way of executing the transactions.

The other way of execution is let people proceed and let them submit to you at the end of the execution, whatever they have done and now check whether what they have done is correct or not when two people are simultaneously working, they assume wrong values tell them that your thing is wrong because I have already accepted somebody who has actually done it before. Now you go back and redo assuming that this is the correct value. So it depends on were exactly this consistency criteria is actually applied. But if you assume lots of times, they are not going to be conflicts. The people are going to operate on different set of data items, it's possible for you to actually assume that the execution can be allowed to proceed independently and you check for the consistencies at the end of execution that is one approach.

The other approach is to actually look at these two ends of spectrum, seeing that one end of spectrum is what we are marking as the optimistic end of the spectrum and the other end is what we have marked as pessimistic end of the spectrum. Now we have a number of protocols which lie in between these spectrums, this full spectrum of things that we have. What we have actually seen is 2 PL in grade detail and this falls under the pessimistic side of the things.

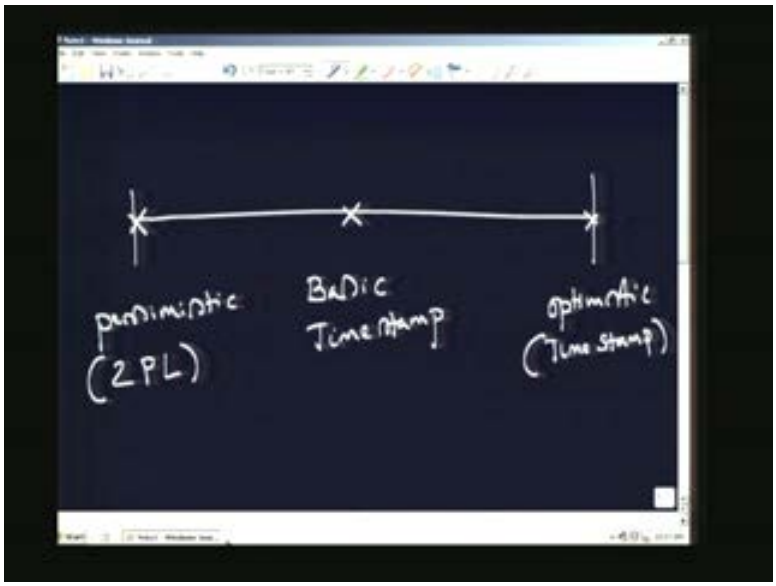
(Refer Slide Time: 11.45)



We have a variety of time stamping protocols; some of them can be seem to fall fully optimistic time stamping, fully optimistic protocols. There are protocols which are time stamp based algorithms which are not completely optimistic, they fall in between the

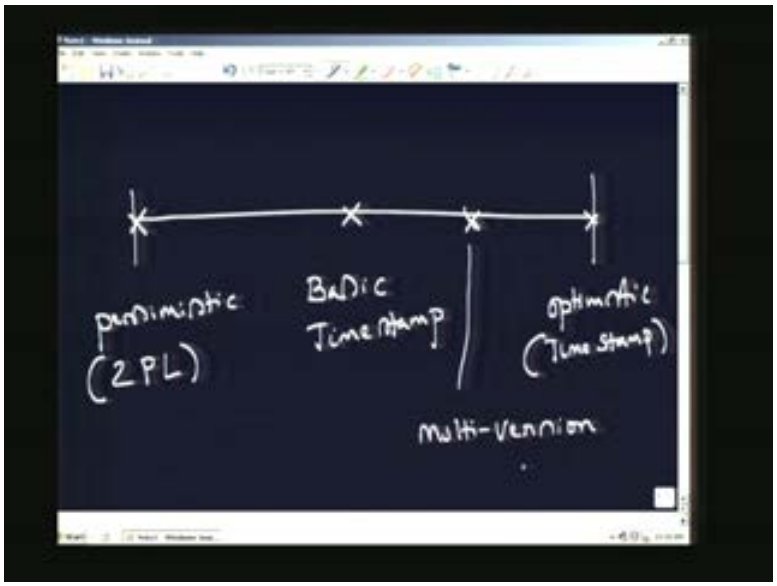
pessimistic and the optimistic protocol. What we are going to do is we are going to look at protocol, called the basic time stamping protocol and then we will see how this protocol is different from the 2 PL locking kind of the algorithms that we have seen earlier.

(Refer Slide Time: 13.05)



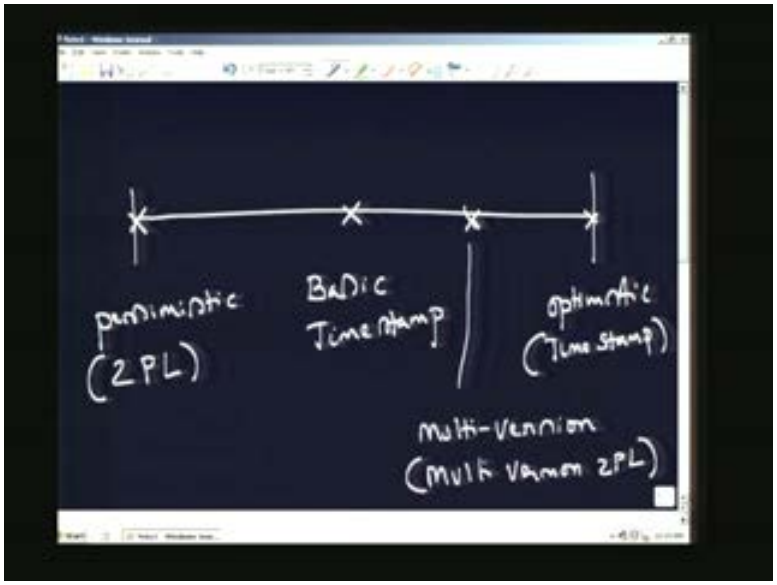
What we will do when we study the basic time stamping algorithm is to understand how time stamp based protocols work in the concurrency control. Then we will modify this basic time stamping protocol to produce what we see as a fully optimistic time version of the time stamping protocol. Also other protocols more notably the multi version protocol, it's possible to map it somewhere here the multi version protocols. What they basically do is they produce multi versions of the data item when reads and writes are going on which means that they keep multiple copies of the data item being manipulated in the database and consequently you have what are called the multi version multi version protocols, concurrency control protocols.

(Refer Slide Time: 14.06)



One modification of the 2 PL, 2 phase locking protocol for multi version is the other thing that we are going to look at which is the multi version 2 PL is another algorithm that we are going to look at as part of the multi version protocols.

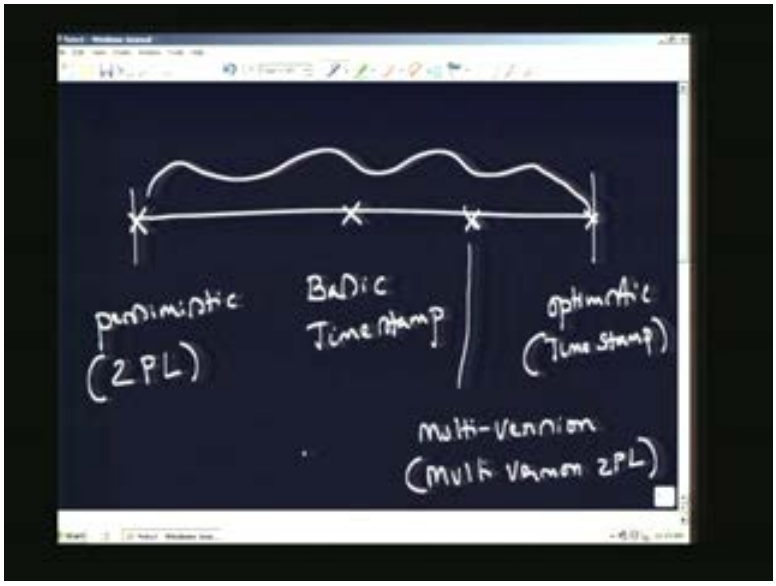
(Refer Slide Time: 14.46)



We will look at basic version multi protocol and we are also going to look at two version locking protocol which is called the modified 2 PL for multi version protocols. What this actually shows is there is whole gamet of algorithms as shown in this picture. If you basically look at this spectrum is quite wide in terms of number of protocols that can be put in between the pessimistic and optimistic kind of algorithms. It depends to a large

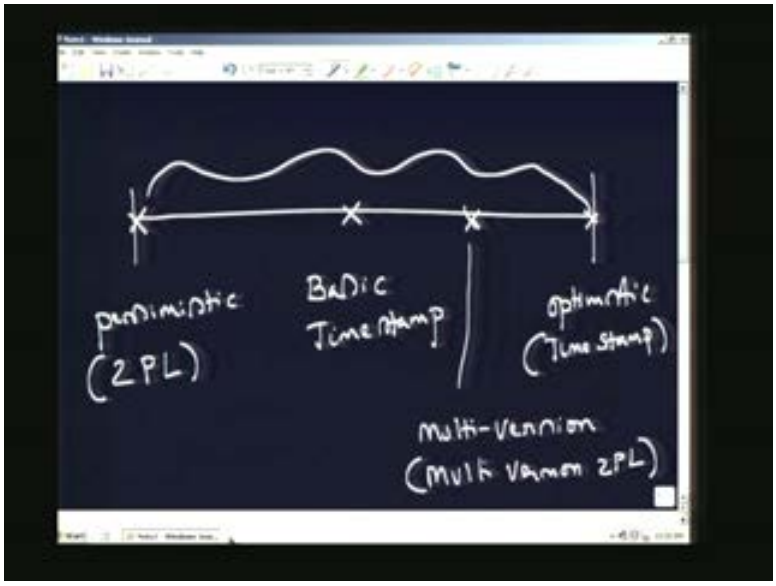
extend to what side the algorithm should be applied, depends to a large extent on the system configuration.

(Refer Slide Time: 15.29)



For example if you assume that they are going to be too many conflicts, the data items are going to be having many conflicts for particular data items then it is not worth actually, putting optimistic kind of protocols. On the other hand where there are not likelihood of lot of conflicts and you start mapping it to the optimistic side pessimistic side then you are likely to have the throughput drastically comes down and it's not worth actually mapping or putting the pessimistic kind of algorithms into your database systems.

(Refer Slide Time: 16.20)

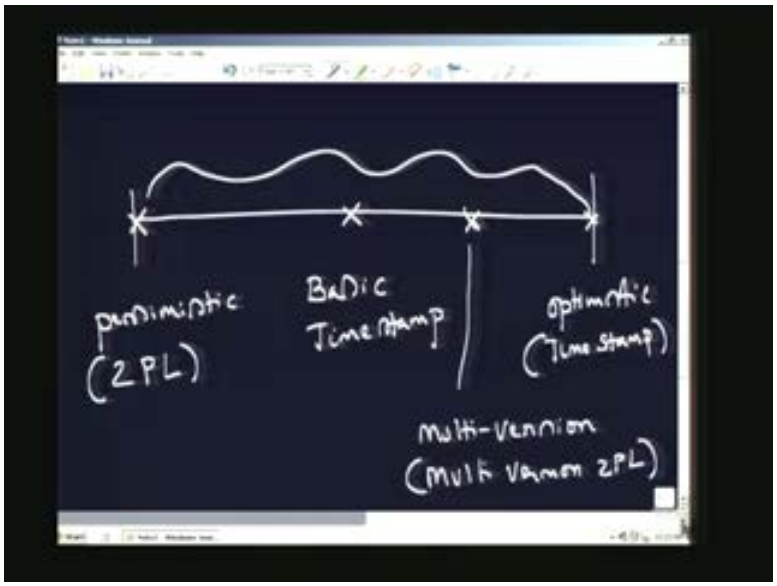


A good example here again is to look at what we see as, typically when a new movie is released, a large number of conflicts you know large number of people try to book for the same movie you know on a particular theatre. And also you going to see that there is a choice for a particular you know set of seats because there are more preferred seats in the theatre compared to other kinds of seats in the theatre. If you have visited the theatre many times you know which is a convenient place to sit and view your movie. In effect mean says a large number of people, when the new movie is released we try to book the tickets and many of them will start asking if you ask the preferences, will start asking for those set of seats that's basically high conflict data items.

If you basically look at the data items, you viewed the data items you can see a large number of people trying to access or trying to modify or manipulate a small set of data items and this is what mean by conflicts being very high for the small percentage of data items. Now this suddenly requires some kind of concurrency control. Now in this particular case if you actually apply a pessimistic algorithm which ensures from the beginning that the transactions operate in a consistent way works well because ultimately a single seat can be booked by only one customer.

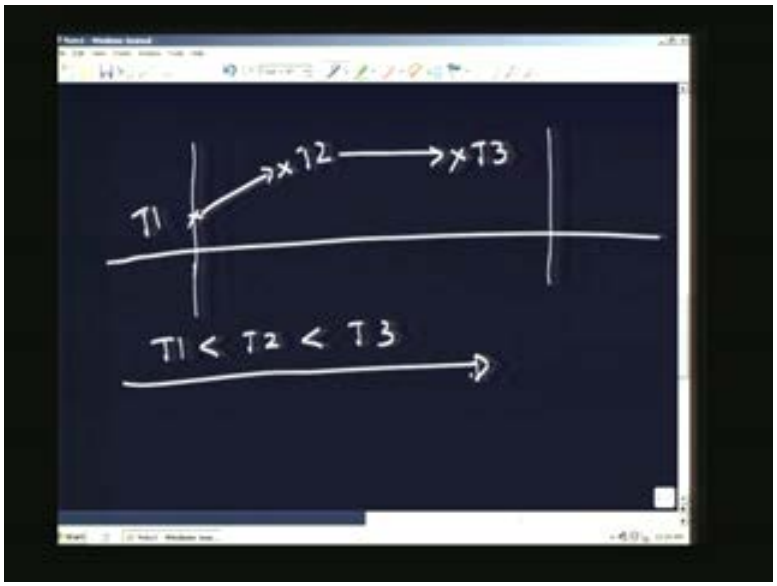
You cannot have multiple people trying to book for the same seat whereas 10 people competed for the 1 ticket and ultimately 9 have to be aborted even if they have all gone ahead and then did whatever manipulations they have to do. But at the end, the database is going to say only one of them is going to get the ticket which means that 9 of them abort after proceeding taking all the information they abort at the end, whereas the pessimistic concurrency control would have actually aborted all the other, would not have allowed the 9 to proceed in the beginning which means that the transaction would have been blocked from actually trying to manipulate the data item once it is actually booked. That's essentially the difference between optimistic and pessimistic kind of an algorithms. What we are going to look at is look at a little deeper in this sense and understand what exactly is the way, this consistency is enforced by the pessimistic and optimistic algorithms as far as the transactions are concerned.

(Refer Slide Time: 19.24)



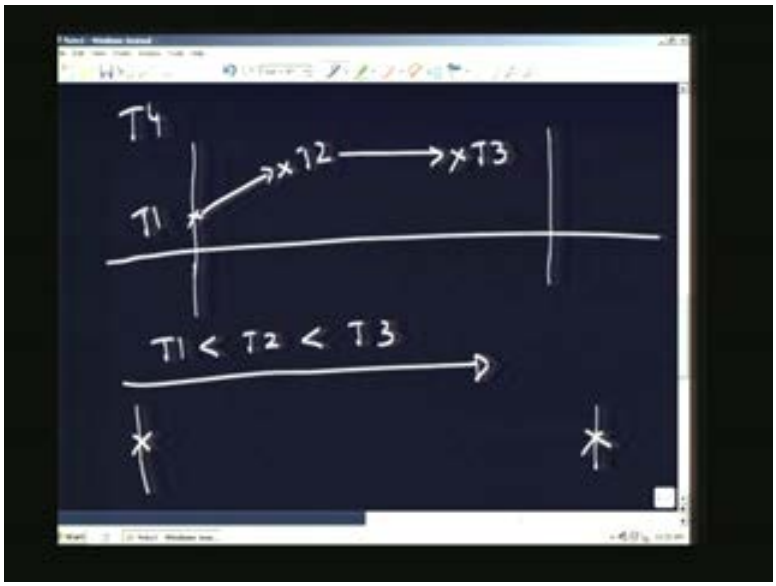
Now what we will do is we will actually take a sample graph and start showing how these graph actually has grown, if you typically looked at the two cases of optimistic and pessimistic concurrency control. The idea of actually looking at this graph is to understand in a more deeper way how the consistency checks at different points will really help you in terms of resolving the conflicts. As you see in the diagram there is typically at this point of the time, there are set of transactions which are executing in the time sense that T_1 which is actually before T_2 and now which is basically before T_3 an order in which the transactions are trying to execute one after the other.

(Refer Slide Time: 20.32)



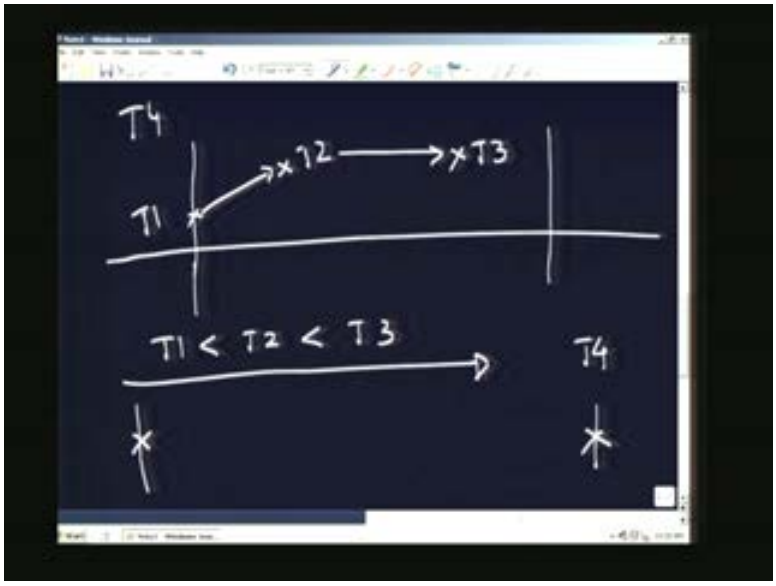
If you keep a new transaction an incoming transaction T_4, now there are two points which we are talking in the graph. One is the entry point and other is the end point. Now T_4, if you allow T_4 to execute without really looking at, what it is trying to do will be consistent or in consistent.

(Refer Slide Time: 21.00)



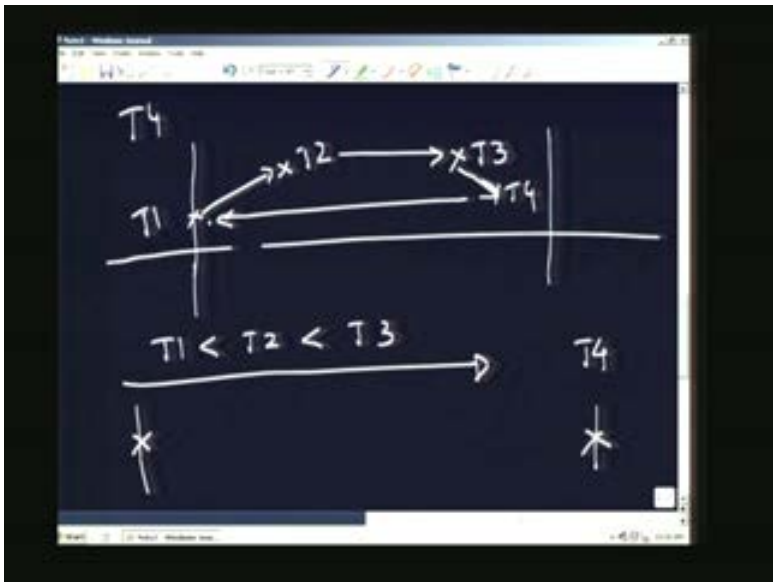
What would have happened is T_4 would have proceeded to execute and when it comes at the end of the execution, then you try figuring out whether whatever the transaction is trying to do makes sense or makes consistency, whether it falls under the consistency criterion whether it satisfies the consistency criterion.

(Refer Slide Time: 21.30)



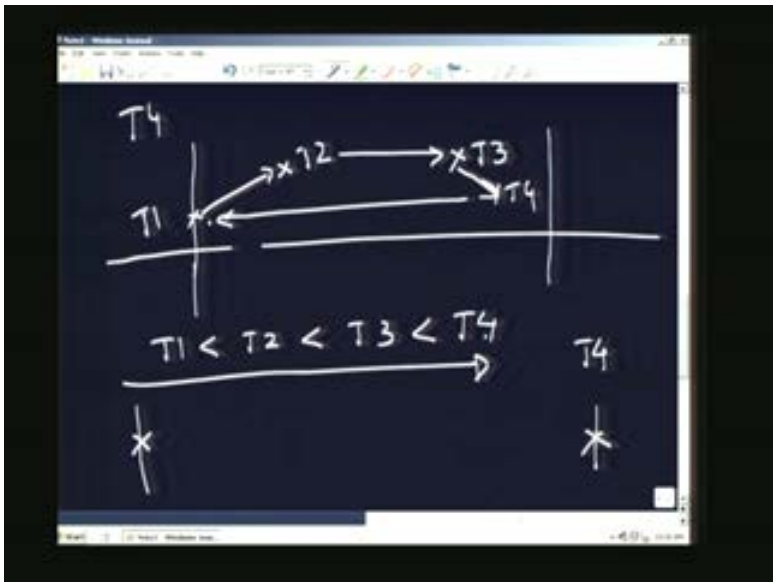
Now all that it means is if this has to be satisfied, if there is lightly to be an arc you know something which shows that the T_4 at the end of the thing has to actually, T_4 at the end of the thing will have a precedence relationship which are shown here carefully follow the diagram to understand what we actually trying to discuss here.

(Refer Slide Time: 21.51)



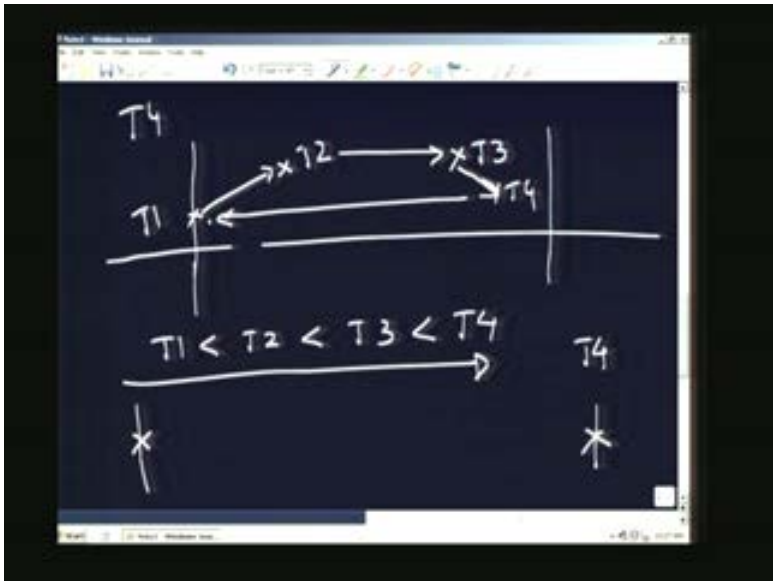
There is a T_4 that entered after I have this graph. Now where do I replace this T_4? If I am actually saying that T_4 comes after T_3 this is fine because this is really doesn't disturb the order.

(Refer Slide Time: 22.29)



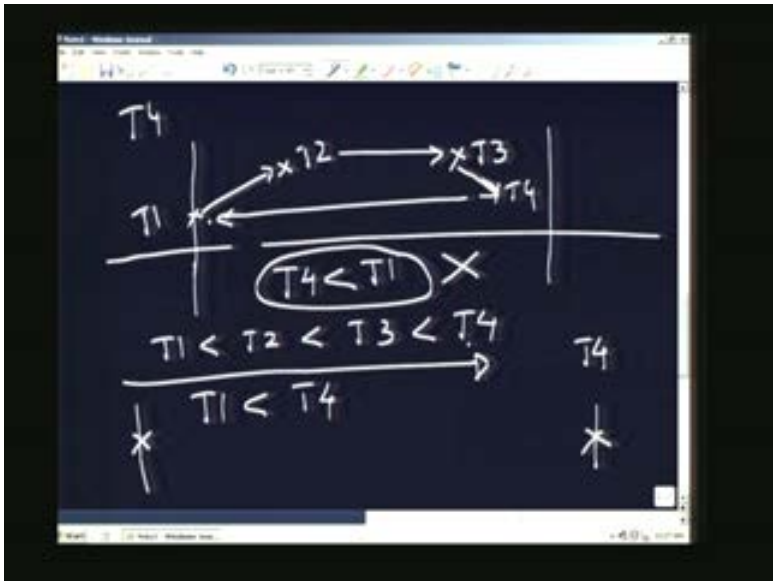
Because T_1 comes before T_2, T_2 comes before T_3, T_3 comes before T_4. So there is specific order in which things are happening for you.

(Refer Slide Time: 22.49)



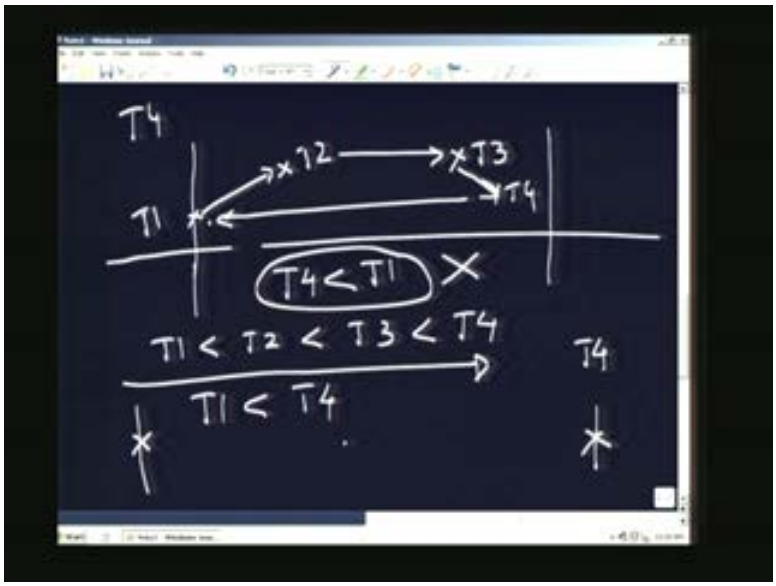
And this is correct order because one after the other the relationship is maintained but on some other conflicting data items if T_4 has to come before T_1 which means that now I want to actually force the relationship on the graph something like this, you wont be able to insert this relationship in to the sequence of actions that you're doing and this actually violates the consistency which means that it wont be possible for me to no longer say this because from this the relation that I get is T_1 is before T_4 by the transitive relationship.

(Refer Slide Time: 23.05)



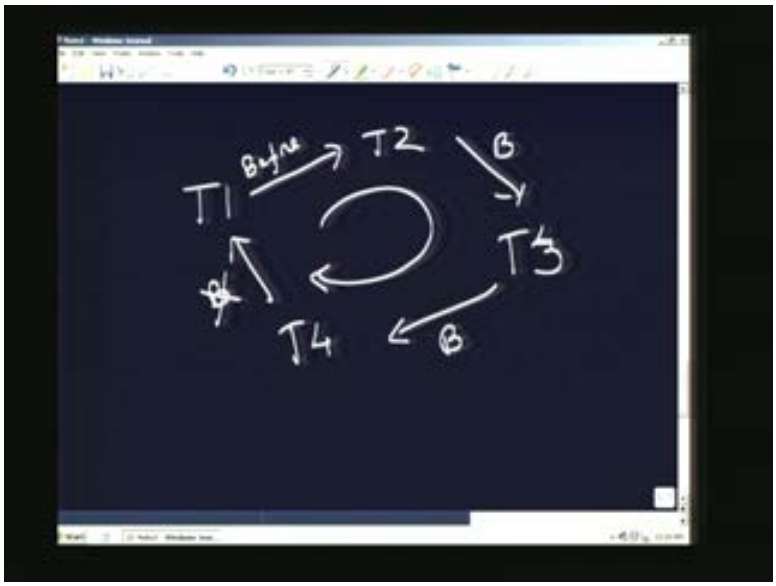
This is like that I ate breakfast then I ate lunch then I actually ate my evening snacks. Now I can't say suddenly my dinner comes after evening snacks but actually my breakfast which is the first event that I have actually performed occurs after my dinner. That is what exactly is happening here which is a violative because by this relationship, the breakfast should be coming before the dinner event but whereas I am saying that my dinner event comes before my breakfast event and this violates the consistency criterion because one after the other as you have actually able to see here, one after the other the relation is correct as long as the future relations doesn't violate whatever order I have been able to come with.

(Refer Slide Time: 24.23)



If you really understand what really we are trying to say in this graph is the following. I have a set of transactions and when actually I write this relationship that I have here, all that is being talked about now is if there is a cycle in this graph, it is violative of the consistency criterion because a cycle, you can't break the cycle one after other. What essentially this is showing is this is before relationship. Now what this is saying that T_1 is before T_2, T_2 is before T_3, T_3 is before T_4. Now suddenly I am saying T_4 is before T_1 and this actually is what introduces the inconsistency in to the execution of the transactions as one after the other.

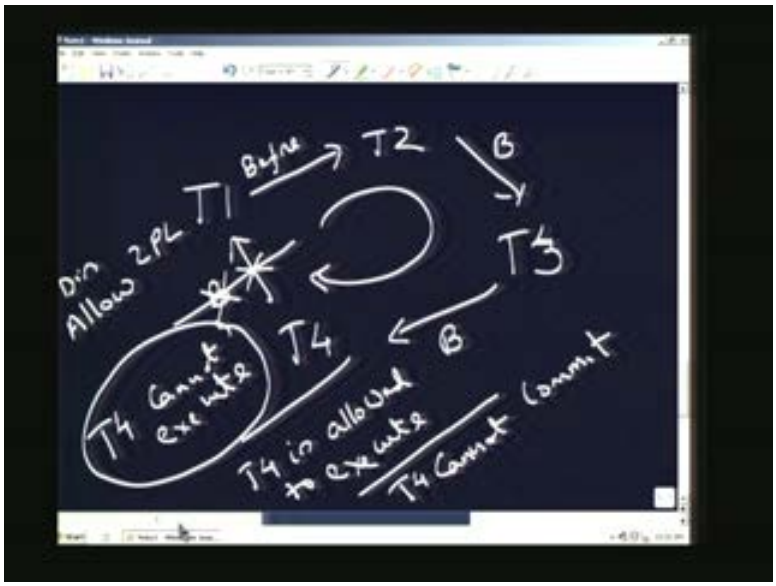
(Refer Slide Time: 24.45)



Essentially this cycle in the transition graph is to be avoided, if you want to produce consistent execution of the transactions. Now understand what we will try to understand here is how does a pessimistic kind of algorithms will really try to solve this problem versus how optimistic concurrency control algorithms tries to solve this problem. I will try use this same transaction graph to illustrate this point that an incoming transaction, we first map in to the transaction graph by the 2 PL which means that it will never allow a transaction to be executed unless it's position in the transaction graph is fixed by the algorithm which means that there is no way T_4 could have executed in the 2 PL case, once it starts executing this condition would have checked.

If this condition is not possible, what 2 PL does is it essentially makes T_4 execution impossible. T_4 would have not executed, T_4 cannot execute which means that I prevent T_4 from execution from the beginning. This is what we call as basically not allowing disallow, disallow the transaction from the beginning and this is what 2 PL would have done. Now what the optimistic concurrency control algorithms will do is T_4 is allowed to execute, allow to execute but then it would have failed when it wants to commit. T_4 cannot commit because at before commitment we basically check whether what T_4 is done consistent or not which is the two spectrum that we are talking. This spectrum is where you don't allow in the beginning itself, in the other case you allow the T_4 to execute and stop it from committing after it has executed, it still cannot commit because its violating the restriction. Now let us go deeper and understand this because this is basis for further discussion when we go on to timestamp algorithm also.

(Refer Slide Time: 26.46)

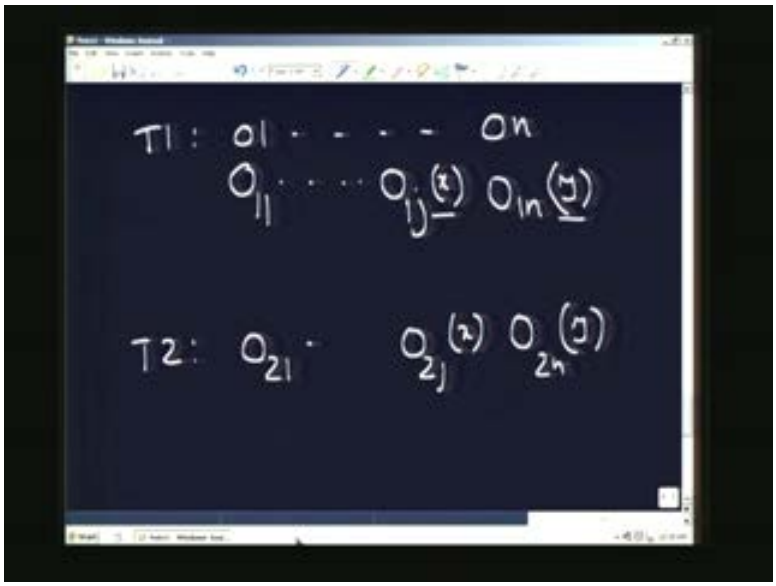


What we will try to understand is how the 2 PL will disallow the transactions from executing from beginning itself, the consistency check will be done at the beginning itself. And as I do this I will also try to informally prove this 2 PL actually produces serializable schedules. Now let us look at a set of conflicting operations ultimately a transaction transaction boils down to a set of operations performed by their transaction.

For example if you typically look a transaction T_1, there is a set of operations this transaction performs and that can be O_1 to O_n. To get the correct subscript what I am going to do is I will actually make this order, this operation with the subscript which is the transaction number in this particular case and under that I will typically look at the further subscript which is the operation number. What this means is an operation i of j means that this is the ith transcation and this is the jth operation. Since this is the first transaction that I am talking about. What I am going to do is I will try to make this just the 1 of j and this becomes operation 1 of n. And we will also to complete the notation what I will basically indicate also is the data item on which this operation being performed.

For example you can indicate here this actually accesses or does something on x or y or whatever it is. Now the meaning of this is operation of first transaction, this is the jth operation, this is the manipulation this is the xth data item. This is the operation n of transaction 1 manipulating data item y. Now what we will basically look at it is when this is executing in the context of a 2 PL, we need to acquire a lock on this data item before you actually proceed on x or y because that is how 2 PL actually ensures that you are not operating unless you acquire a lock. Now in this particular case, we are typically looking at two transactions that are executing simultaneously to understand how they could be manipulating. Now let us say these are the sample transactions of 1 and 2, transactions T_1 and T_2.

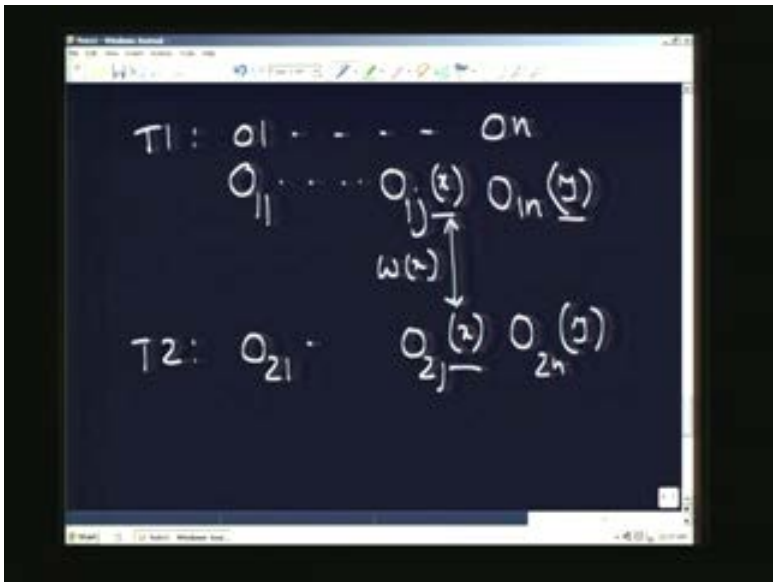
(Refer Slide Time: 28.54)



Now we will say that an operation will be conflicting if it is actually operating on the same data item. What does this actually mean? If in a case of banking transactions if I go and withdraw cash from my account and somebody else also simultaneously withdrawing cash from his account, not my account. We both are essentially operating on our own individual accounts. There is no conflict in this particular case because he is operating on his account and I am operating on my account. The minute both of us go to the bank and try to withdraw the amount from the same account then we will be conflicting.

Again if we both are looking at the balance amount that is available in the account, we still not conflicting because both of us simultaneously can view, what is the current balance in the account. Without really withdrawing if you are just looking at the balance, still you are not producing any inconsistent results. So a simple read, though it is on the same data item it is still not conflicting. If either of us are withdrawing or both of us are withdrawing which means that if one of the operation is a right operation, if both are operating on the same data item in this particular case you can see both of these are actually operating on the same data item x and if one of them is a right operation then only there will be a conflict because if I am withdrawing and he is looking at the current balance that is available in the account. Now it depends on whether I read it after I withdraw it or before, it starts now becoming a conflict operation if one of it is actually right.

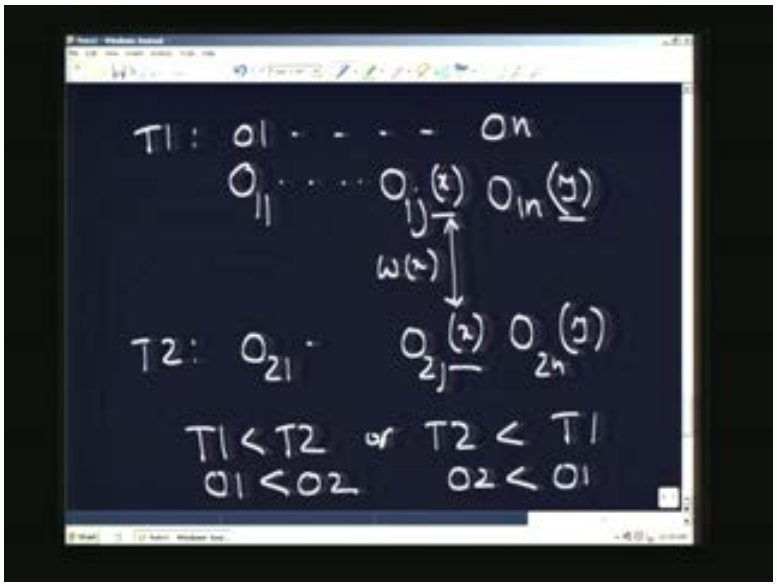
(Refer Slide Time: 33.00)



So this is what we mean by two operations being conflicting. Now only when operations are conflicting, we need to worry about the order in which these operations have executed. If the operations are not conflicting, for example if two of us are withdrawing the money from two different accounts, it doesn't really matter because we are just operating on two different accounts but the minute actually we started operating on the same account, we need to know what exactly happen with respect to that account, who has first seen that account, how much has been withdraw by a person x, how was it actually added later by somebody else all these details one after other needs to be there. The after relation is important, otherwise there is going to be inconsistencies in the final result that you see as far the account is concerned.

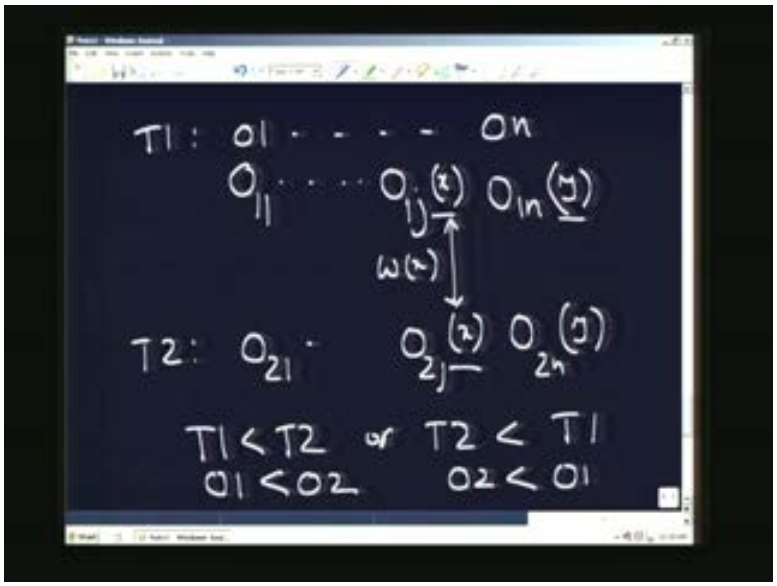
So when we actually look at T_1 and T_2, what we are interested in is if there is conflict between T_1 and T_2 then only we are interested in finding out the relationship between T_1 before T_2 or T_2 before T_1. And this relation essentially boils down to looking at some operation of 1 and some operation of 2 and saying how this two conflicting operations have actually been executed with respect to each other. This is what we actually, at the end of it interested to see T_1 before or T_1 after.

(Refer Slide Time: 34.32)



For example if one of it is a right operation then we say that it's conflicting as we just explained. Now look at operation O_{1j} and operation $_{2j}$ of T_1 and T_2. Since they are conflicting now, if they are not conflicting we can't write this order because they can execute in any order they actually wish, without really producing inconsistent results.

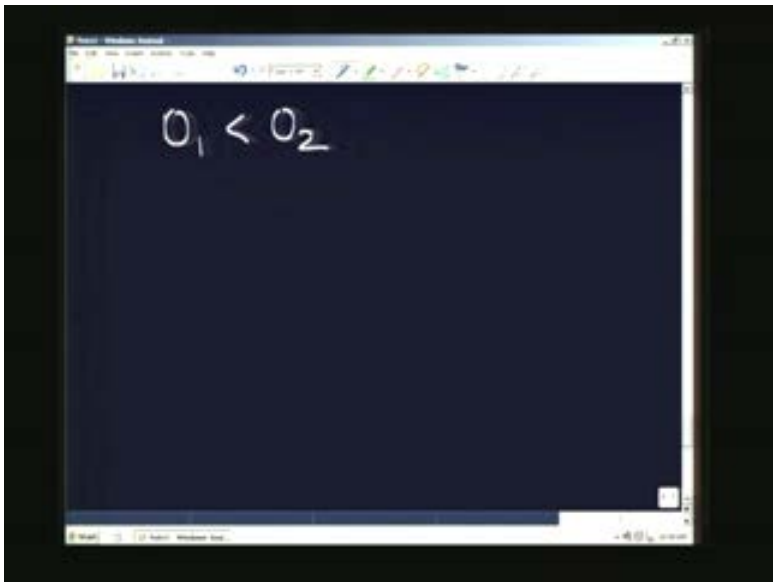
(Refer Slide Time: 35.30)



If they are conflicting, we have to understand which operation executed before the other. Based on that we are going to say that the transaction T_1 is coming before T_2 or T_2 is coming before T_1.

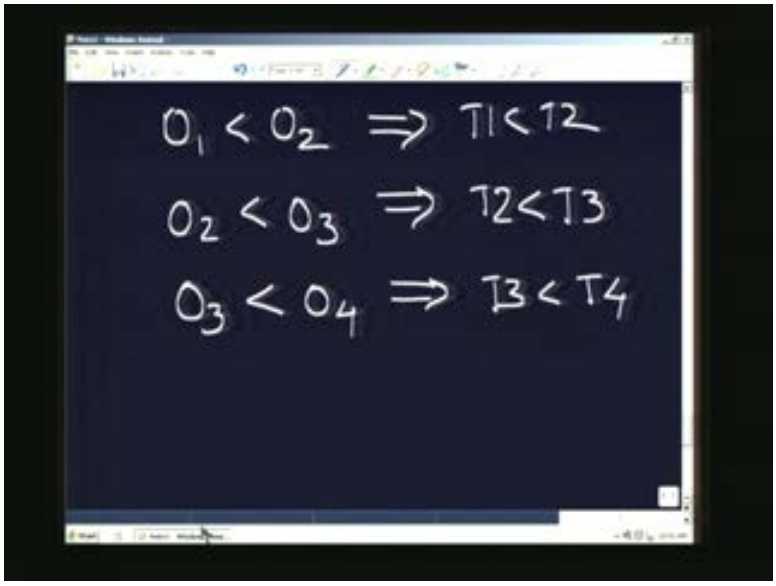
Now what as an end of the series of execution, if you typically look at now I basically will take only the one transaction suffix, I will drop the actual operation suffix, I will say two conflicting operations O_1 and O_2 belong to T_1 and T_2 are executed in this particular order which means that there is an order of T_1 before and T_2.

(Refer Slide Time: 36.00)



Now imagine I have a conflicting O_2 versus O_3 for the third transactions which means that T_2 executed before T_2 executed before T_3. Now if you look at another, this is what we have actually looked at, O_4 this means that T_3 is executed before T_4. Now look at the discussion that we actually had a while ago on the transaction graph. This is what exactly happened in our transaction graph.

(Refer Slide Time: 36.32)



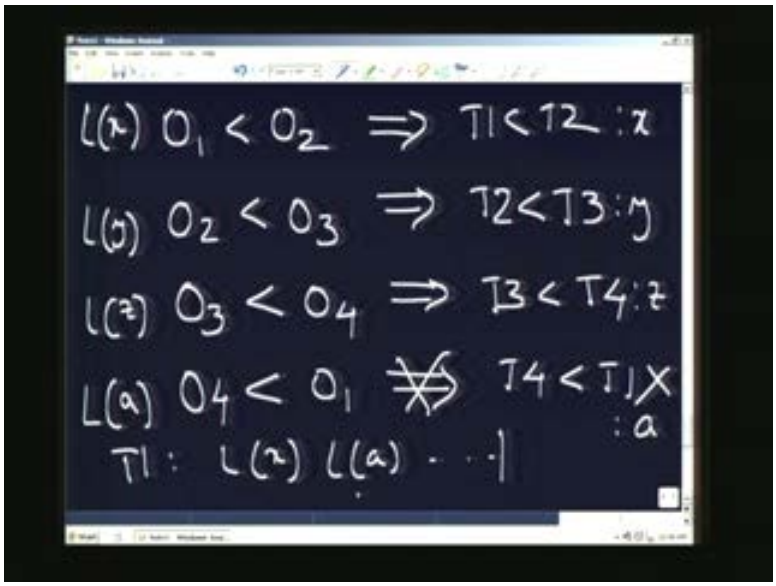
We are just proceeding one after the other but when we actually came to T_4, what is really happening was we are trying to say O_4 also has a conflicting operation with O_1 but these were executed in this particular way which actually means that this is the order that would have happened and which is what actually produces the inconsistent results. Now if you basically look at 2 PL, should this execution would have happened if I consider 2 PL. Now O_1 would have actually locked the data item, let us say this is the conflicting operation is defined here on some data item x that I will indicate it here which means that there is a lock on data item x which was obtained by O_1. Now, let say there is a conflicting operation between O_2 and O_3 on data item y which means that in this particular case, the lock was obtained by O_2 before O_3.

Now let us say this is a conflicting operation Z here and on which actually we have got a lock on jet for O_3 because if the lock was not obtained this sequence is not possible because the transaction will never execute in the case of 2 PL unless the lock was granted. So T_1 would not have been able to execute unless it has obtained a lock on x because between O_1 and O_2 conflicting, T_1 has got the lock before T_2 that's the reason why this relationship is possible, otherwise this relationship is impossible. You can't have the relation as shown in this particular equation here. Now let us say there is between T_4 and T_1, let us say there is an item, data item a this is doesn't fit into regular sequence that's why I am using different xyz this is a sequence, a is out of the sequence. Now between O_4 and O_1, there is a conflicting operation being performed on a right. Now O_4 is saying that it locked, it has got the lock on a before O_1.

Now for a minute think if this is possible in 2 PL. What O_1, what transaction T_1 would have done is it actually requires a lock on x, a lock on a, right before its start executed. Now what this set of equations I have here says is I have got a lock on x, now O_2 would not have got, transaction t would not have got unless I released this lock to it which means that I should have released a lock on x before getting the lock on a because O_2 saying that it has actually got a lock on y before O_3.

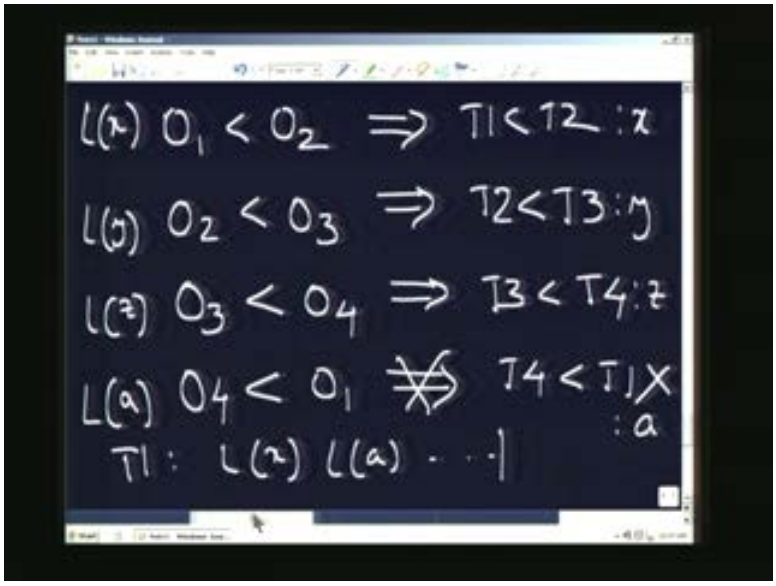
Similarly O_3 is saying I have got a lock on Z before O_4 but O_4 is saying that now I have got lock on a before O_1. If you carefully understand this unless this lock is released by T_1, T_2 would not have got that lock, unless T_2 has got that lock it would not have actually proceeded to get the other lock on y. So actually if you use this, equation we are writing here is inconsistent because 2 PL prevents this by saying if I have a lock I won't release that lock till I get all the other locks.

(Refer Slide Time: 37.18)



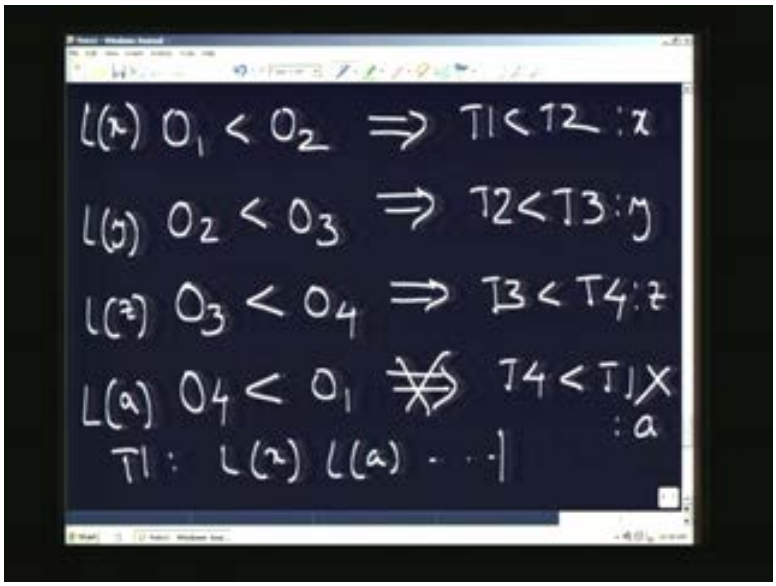
Remember the condition that was imposed by 2 PL which says that unless all the locks are obtained, you are not going to release the previous lock that is what we meant by lock point right. All the locks will be obtained that 2 PL finishes, the transaction finishes the execution then it basically releases the locks. If you release one lock you are not supposed to ask any more locks. With that condition if I have released lock x, I would not have asked for lock on a which means that transaction T_2 would not have been able to execute unless I finished all the executions. Since T_2 would not have got a lock till I finish, there is no way T_4 can say it has actually has come before me and obtained lock a lock on a if this is the way it's supposed to execute.

(Refer Slide Time: 41.55)



This is intuitively what happens with 2 PL. And that is essentially the reason why this important condition is put in 2 PL saying that if you actually release one lock, you are not excepted to ask for any more locks. If that is not followed, you would have ended up actually having this problem. The cycle in the transaction graph would have happened if I allow this condition that x could be released but still you can ask for a. That is what actually is prevented in 2 PL saying that once you have a lock on x, you have to ask for lock a before you release any of the locks that you have required earlier because if you release one lock you cannot ask for any more locks. This essentially prevents the cycle in the transaction graph. This explanation makes you understand how 2 PL checks for the consistency at the entrance, when the transaction is start to executing how 2 PL ensures the consistency requirement.

(Refer Slide Time: 42.42)



Once the transaction starts executing and its start getting the locks, there is no way you can say the transaction is inconsistent. It will never get in to an inconsistent state of execution. What really happens in this particular context is the transaction can get blocked, when its get blocked this results in a scenario of a transaction either permanently waiting if there is a deadlock kind of a scenario or waiting for sufficiently long to acquire a lock but the transaction will never start execution unless it is in an inconsistent state.

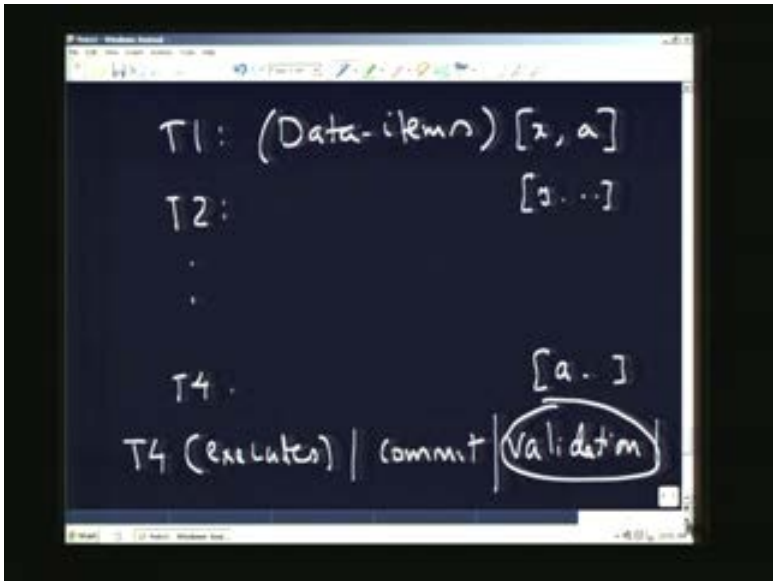
There are certain issues which relate to what is the overhead of this kind of the algorithm which we will discuss towards the end time stamping based algorithms. The other issue is when you actually move to the other end of the execution and model check for the consistency, how exactly that is going to be done. Now we will look at typically the scenario there and see what are the possibilities of that being done and how that will be that will be done in case of optimistic scenario.

Now what really happens in the case of and optimistic algorithm is we basically take a transaction data items on which it is basically operated. This set of data items which are

actually manipulated, in this particular case it is going to be a set, so this set going to be x and a. in the earlier case this is the set which transaction T_1 has actually manipulated. T_2 manipulated a set of data items relating to y and probably some other set. If you take T_4, this is the set which we get here a and probably some other set. Now when T_4 wants to commit, T_4 can just execute whatever it wants to do, T_4 executes.

Now it says I want to commit that is basically when it is trying to write the values of a back on the database. This is at the commit point, what your going to do is it check for the values of a and say whether whatever previous things done by the other transactions are consistent with respect to this. If this is basically what we say as the validation check. the validation check is done at the n and if T_4 passes the validation then I actually allow T_4 to commit. What happens in the case of a purely optimistic kind of scenario? Every transaction is allowed to proceed, what really happens in the case is the transactions takes values, does it in a local copy of the data item and manipulates whatever it needs to do on the local copy and when it actually finally wants to write you perform this validation at the end of it. I will give only an intuitive explanation at this stage to allow you to understand what's happening here but more detailed discussion is going to follow when we actually take up the time stamping algorithms which are the optimistic case of the time stamping algorithm.

(Refer Slide Time: 44.58)



We will go in depth, see how exactly this works. To give to an intuitive explanation of the optimistic scenario, what is done in this case is let us say 4 people are trying to do simultaneously the something. Each one will be given a local copy. This is like 4 people, 4 people in this case as you know you can see the transaction T_1, T_2, T_3 and T_4. Now all the 4 transactions here will be allowed to proceed by taking a local copy from the database. Now 2 PL would have said I am going to order all of you and then you have to get in with the with respect to the tickets I have given you. T_1 is number 1, T_2 is number 2, T_3 is number 3 and T_4 is number 4. So that's the way they are going to execute.

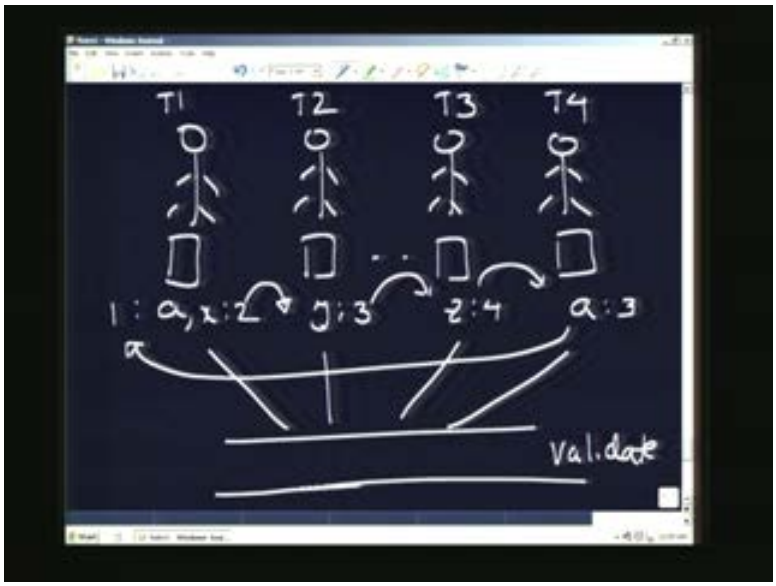
(Refer Slide Time: 47.38)

Now I basically will not bother to give them a ticket when they actually arrived my place. What I will say is you actually try doing whatever you want to do that means each one will get a local copy of the data item. In this particular case as you can see T_3 T_1 and T_4 will get a copy of a. In this case it's going to be a and x, this is going to be y, this is going to be z and other values. Now when you actually finish doing something, for example let us say all the values are produced by the each one of them, at end of it submit to me. When they finish execution, they submit whatever they have done to me.

Now I will look at what is called validate. This validate phase is going to look at, if this problem of a being done in an inconsistent way by T_1 and T_4 will be detected at this stage. For example this will show me that there is something which T_1 done did on a which is inconsistent with T_4. How can that happen? For example what we will say here is T_1 has actually manipulated the value of a with some particular let us say 1 is the manipulated value.

T_1 actually manipulate the value of x to 2 which actually has been used by T_2 and T_2 has actually manipulated the value and produced the value 3 which is actually used by… And now T_3 manipulated this value to 4 which is used by… Now this is actually manipulated this is to 3 and it's actually to be used by T_1. This is where exactly this problem of, I have actually should not have used this and manipulated because I am coming after but with respect to a, T_1 actually has taken T_4 value and manipulated it to 1. Please understand this little more carefully. I actually did something and produced a value for x equals to 2 and that value is used by T_2. This is like I did something in my paper and passed it on to the next person T_2. T_2 use that value and produced a value for y and this y value is passed to T_3 and we used that value to produce a value for Z and that is given to T_4 but T_4 actually given a slip to T_1 before this for a which actually means that there is a cycle here. Who is now took whose result, it is difficult to say because T_4 seems to be a_1, a_1 seems to be affects to t_2 like this, if you basically look at it.

(Refer Slide Time: 48.24)

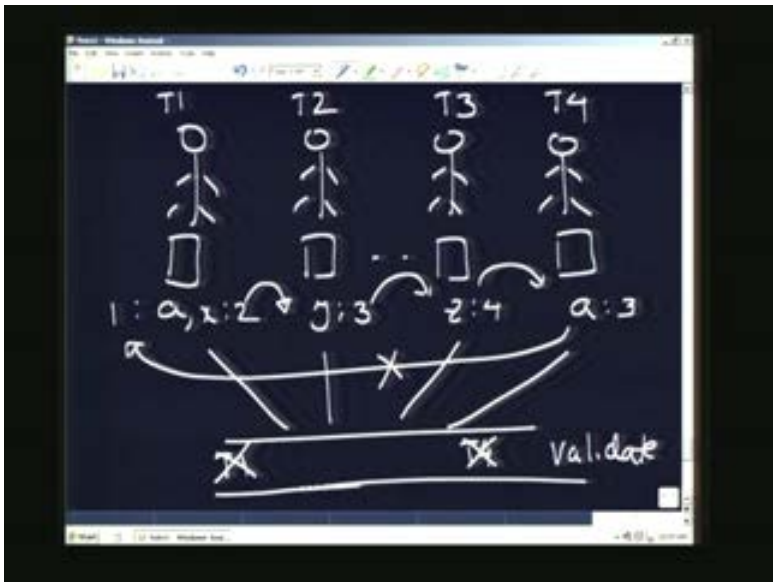


This is what actually produces the inconsistency and this will be what will be detected when they submit their values to me. In this case of 2 PL they will not be allowed to proceed to execute and produce this values but in the case of optimistic algorithms, they will be allowed to proceed, they will be allowed to do this manipulation but when they give the values to me, when I look at them I can see that this is violating the property of consistency which is essentially the cycle in this particular graph.

What the validate phase does in the case of optimistic algorithms is essentially check this dependency at the end of execution and say that now T_4 should be aborted, T_4 has actually produced a value so, I basically abort all those transactions which has actually produced inconsistent results. And I will say that T_1 and T_4 should restart the execution and again they will take the new values from the database and start executing. This is what will happen in the optimistic scenario. Now the problem here is as you can see in the case of a pure pessimistic kind of an algorithm, T_4 would not have wasted its time computing something because I have used some value and I tried computing so I actually sat down and worked out everything but at the end of it I submitted my value and I was told you are inconsistent because you used inputs which are not correct. So you go back and redo what you have done.

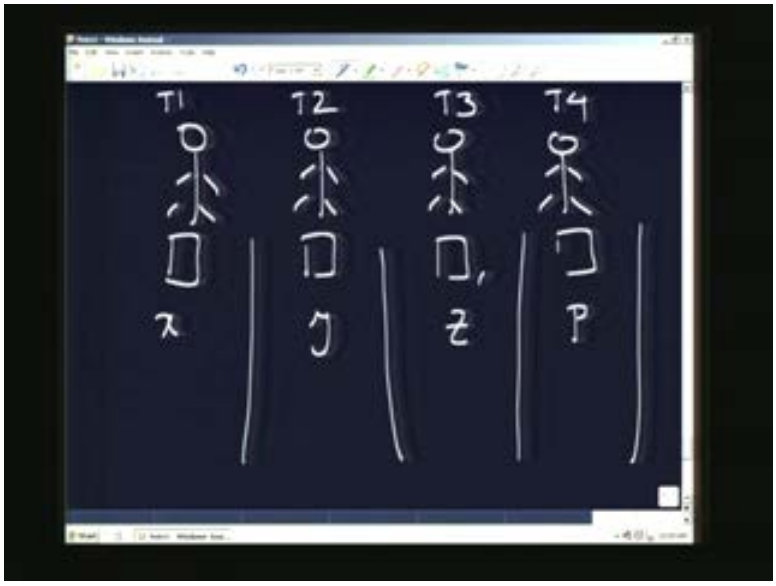
So this is basically a loss of execution time. You have actually unnecessarily executed T_4 and found out that at the end of the day T_4 didn't use the correct value, so it has to abort and it has to restart where this would never have happened at all in the case of 2 PL pessimistic kind of algorithm because they would have been asked to read all these values in the correct way before they start executing the transaction.

(Refer Slide Time: 50.55)



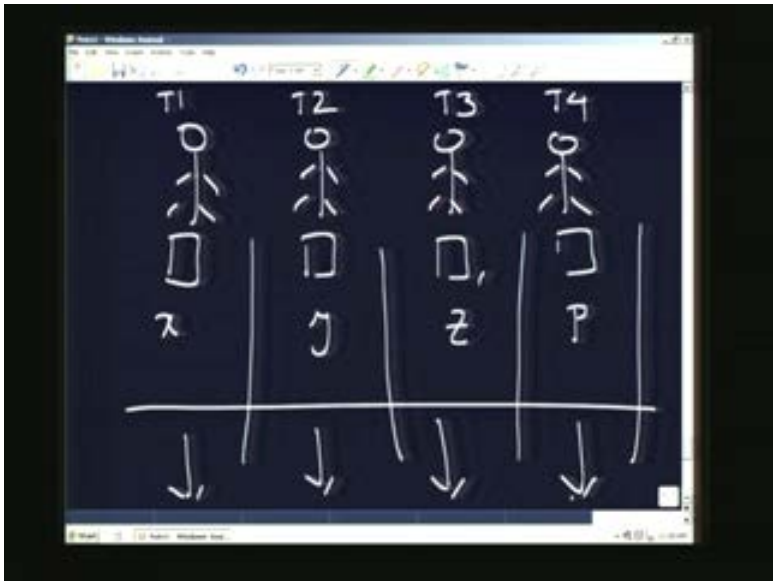
So in that sense the pure pessimistic kind of algorithm would not have allowed this scenario where the aborts are required. Whereas in the case of an optimistic let things proceed but then at the end of the thing you check whether what you have done is correct or not, both have as you can see here both have the plus and the minus. If you except T_1 T_2 T_3 T_4 in this particular case, this diagram you can see they generally operate. For example the scenario what we actually assume here is that a large number of transactions generally operate on different data items not on the same data items. That means this is going to be x, this is going to be y, this is going to be z and this is going to be p and there is completely disjoint sets.

(Refer Slide Time: 54.11)



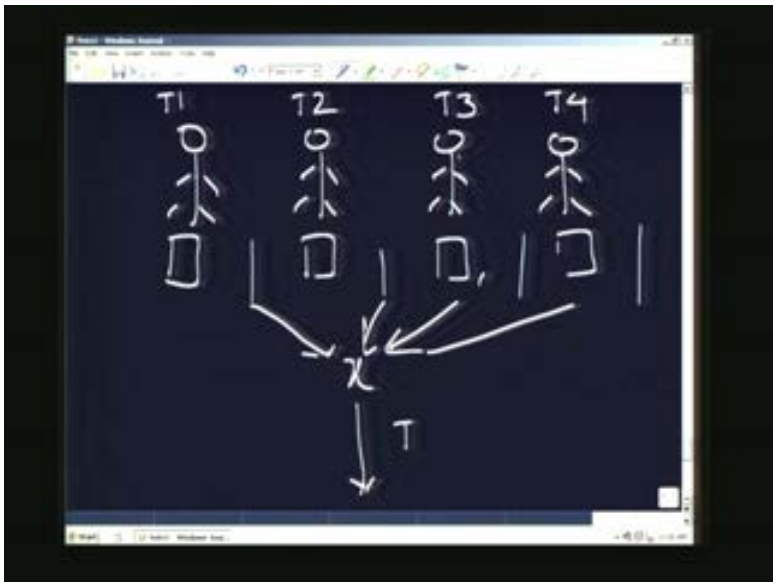
Then you can see that at the end of it they can all go very simply you know without really locking anything because the locking overhead is reduced here. At the end of validation phase they all proceed very smoothly because they are not actually conflicting. This is one case where all of them are using different doors, different keys so all of them pass the validation phase. This is what you will see if you apply a optimistic scenario.

(Refer Slide Time: 55.00)



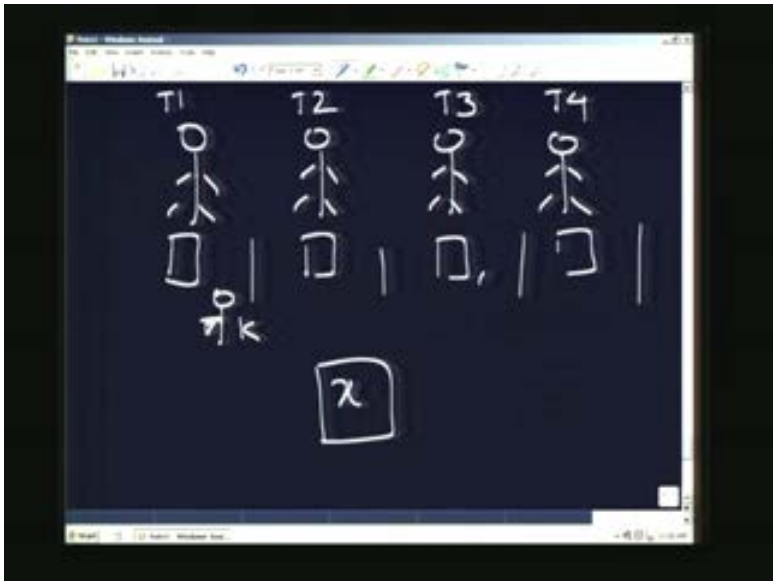
Try to apply a purely pessimistic scenario in this particular case to see what would have happened. Let us say most of the time, all of these transactions operate only on one data item which actually means that it is possible for only one of them to be able to proceed if they actually conflictive which means that all of them will tend to do the same thing data item but only one of them will be able to go, whereas in the other case all of them will be able to go which in effect means that you have to go back and then start reworking out for T_2 T_3 T_4. There is going to be larger number of aborts. If you apply, where there are large number of conflicts, if you apply for pessimistic algorithm there it is going to be a large number of aborts of the transaction and this results in wasted time. In this particular case you can see only one of them have the chance of proceeding because all of them are conflicting. Now if you actually applied a pessimistic kind of scenario, you know that one of them can succeed in getting a lock which means that you would have serialized them one after the other with respect to this conflicting data item and they all coming here and trying to rush through this door would not have happened.

(Refer Slide Time: 55.17)



They will get the key one after the other. This is the case of actually, they are actually trying to hit each other. Now if you basically apply the locking, there is going to be an organized lock on this data item x which is something like this lock is going to be, this is the key for the for the data item, this key is going to be used, passed to one after the other and you can see they all can use this data item one after the other and there is going to be not going to be a conflict after they start executing. This is the other case of applying the pessimistic scenario.

(Refer Slide Time: 56.20)



This lecture what we did is we essentially looked in depth, the difference between optimistic kind of algorithm and pessimistic kind of algorithm. Both have their place in terms of in terms of applications where it should be, were there are contentions are very low, there is no point to apply pessimistic algorithms where there are large number of conflicts there is no point to apply an optimistic algorithm. So, both actually have their place. In fact what we are going to see in the next few lectures is whole gamete of algorithms that fall in-between which also produce excellent results when they are applied to transaction processing system. In the next lecture we are going to take basic time stamping scheme and look at in detail.