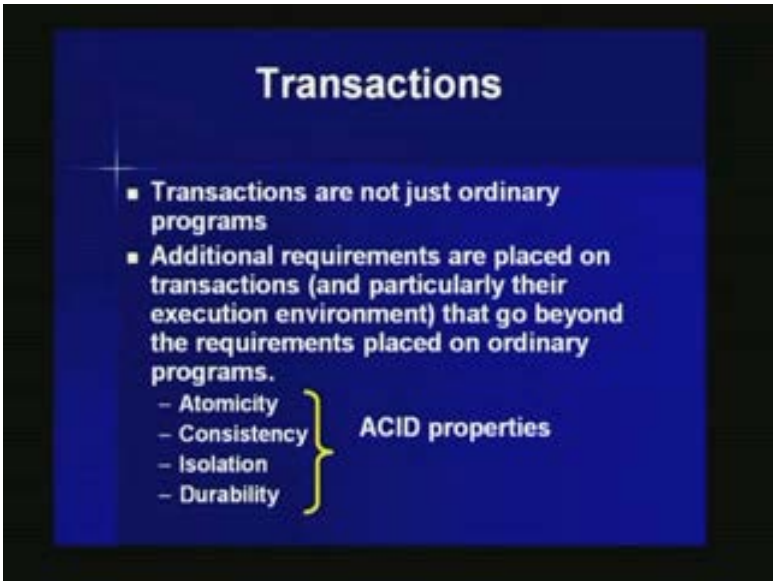
Database Management System
Prof. D. Janakiram
Department of Computer Science & Engineering
Indian Institute of Technology, Madras
Lecture No. # 19

Foundation for Concurrency Control

In the last class we have been looking at the basics of transaction processing systems. What we have done is to look at the basic properties of transactions when they execute in the database system. We looked at the essential properties of transactions in database systems and we have been looking at the four properties that are essential for programs, transactions atomicity, consistency, isolation and durability.

[Refer Slide Time: 1.56]



Atomicity property is required to ensure that all the instructions in the transaction is either executed or none of them are executed. So the property of atomicity ensures that all the instructions either get executed or none of them will be executed. The property of consistency ensures that when more than one transaction operates on the database, the consistent state of the database is maintained. That is they don't really malign the values written on to the database.

The property of isolation provides that the effects in one transaction are not visible to other transaction till it has committed all its values to the transaction. The property of durability ensures that the transaction is returned its value on the database, it will be stored permanently on the database. That's the property of durability. We have been looking at the four properties and trying to understand more on how the consistency property is realized in database systems.

What we are going to do in today's class is to look at the foundations for the consistency management which is also known as the concurrency control mechanisms in database systems. What we are going to look at it is the basic concepts of transactions from the view of concurrency control and this is what we call as foundations of a concurrency control in databases. What I am going to do in today's class is show the basic properties of, basic foundations for concurrency control in databases by first introducing the notion of a schedule. What is a schedule and what are the different things that we can understand by the concept of a schedule.

The second thing that we are going to look at as part of the schedule is we look at how exactly we can produce what are called serializable schedules. This is basically the basic notion of ensuring consistency in databases. So we are going to use these two things, one we are going to look at what is a schedule, what is a meaning of a schedule and how exactly serializable schedules can be produced. After understanding these two things what we are going to do is in the next class, we are going to see some protocols that are their in the database systems that ensure that this schedules produced by the database are serializable schedules. So we look at the protocols in the next class but in today's class, we will understand what we mean by a schedule and what we mean by a serializable schedule.

[Refer Slide Time: 05.28]

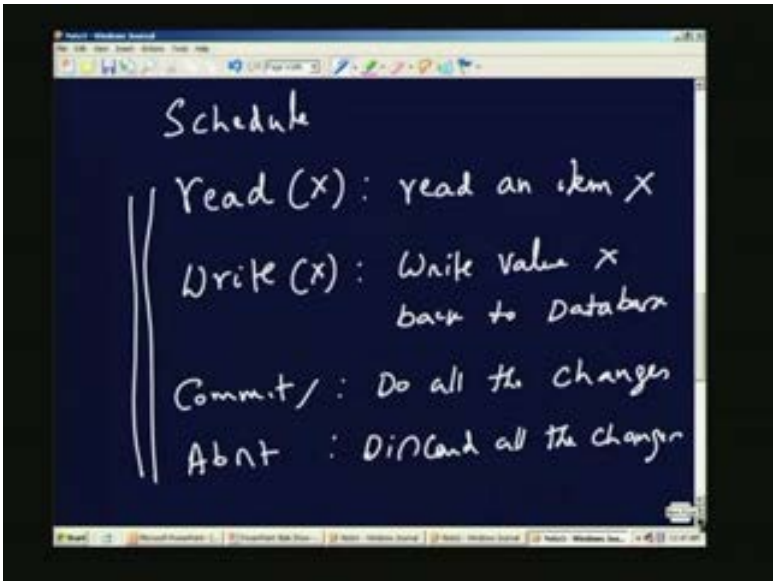
Now let us go and understand the meaning of what we mean by scheduling. Now the meaning of a schedule in database transaction system is, it is essentially a set of operations performed by the transactions on the database. We will consider a very simple schedule let us say S_a and then here what will do is we will write a series of operations performed by the database, by the transactions on the database. We are going to understand first the notion of what are these different operation before we write the actual schedule. What I am going to do is I will say essentially the database operations could be either a read operation which means that the read an item from the database. Read x

shows that this is an operation trying to read a data item, read an item x from the database.

What this means is if this item is not in the main memory, this item will be fetched from the disc and will be transferred into the main memory of the database. And after that this will be used as a program variable by the program to do some operations that means the value is available, now it can perform some computation using the variable. The other operation that the database transaction can do is what we see as write x. write x essentially means that write value of x back to database which means that the updated value of the transaction has to be now returned onto the database. We will also have two extra operations which are performed either a commit or an abort at the end of the transaction execution.

Commit means we are essentially committing all the things that you have done. This is at the end of the execution of the operations. A transaction can issue what is called a commit command, commit actually means do all the changes. And then abort means essentially discard all the changes. So typically the transaction decides at the end of the execution whether to store the values back on to the database or discard them. These are essentially the operations that we are interested when we are dealing with a database. A schedule essentially consists of read operations, write operations, a commit and a abort command.

[Refer Slide Time: 08.44]



Now what will do is we will further define the schedule much more clearly by taking a set of transactions that are operating on the database simultaneously. For example take the case where we have the reservation system where passengers are trying to book their ticket simultaneously. They could be one passenger who is trying to travel to delhi by rajdhani express and he will try to book his ticket for that particular train.

There could be another passenger simultaneously trying to book for the same train for going from chennai to delhi which actually means again both of them, both these operations simultaneously operating on the database. Now both of them should get different seat numbers when this operates simultaneously. If both of them get the same seats, we actually leaving the database in the inconsistent state because two passengers cannot get the same berth to travel from chennai to delhi. So that is what we mean by looking at, i know when transactions are operating simultaneously on a database, how exactly we can produce consistent results on the database.

Now in this particular case what we can see is there could be the first transaction T_1 which reads the current reservation information from the database. We will just abbreviate as read as an r symbol so and we will give an prefix r one here to indicate this is being done by transaction one. This is looking at the rajdhani express availability which is given by r one x. now after actually checking this, it will try to write the value back onto the database saying that it wants the reservation for this.

In some cases the passenger finds that he doesn't have the money, at this point of time he can discard the changes which means that the transaction can get into an abort state after reading the availability but nor booking the final one. It is possible for various reasons. The passenger didn't have a lower berth as he desires, so in all this cases the transaction after started could abort the operation of writing this values back on the system which means that essentially the passenger is not interested in booking the berth for himself after reading the values. Now let us say this basically transaction commits at the end of it which means that you have got the berth and your writing this value back. This is what all the operations that could be performed by the transaction T_1.

Now if you take basically another transaction T_2, it could also be reading the value of this rajdhani express availability for a berth and then writing the value back and then later committing. Now if you basically look at, these are all the operations of the second transaction. It's possible that these two operations of the transactions can be interleaved in any order when they are executed on the database. What does that mean? It means that it's possible that, I could execute r_1 x of T_1 then r_2 x of T_2 then w_1 of x and w_2 of x then say I commit c_1 and c_2. This could be one possible execution sequence because these can be interleaved in whatever fashion that is possible.

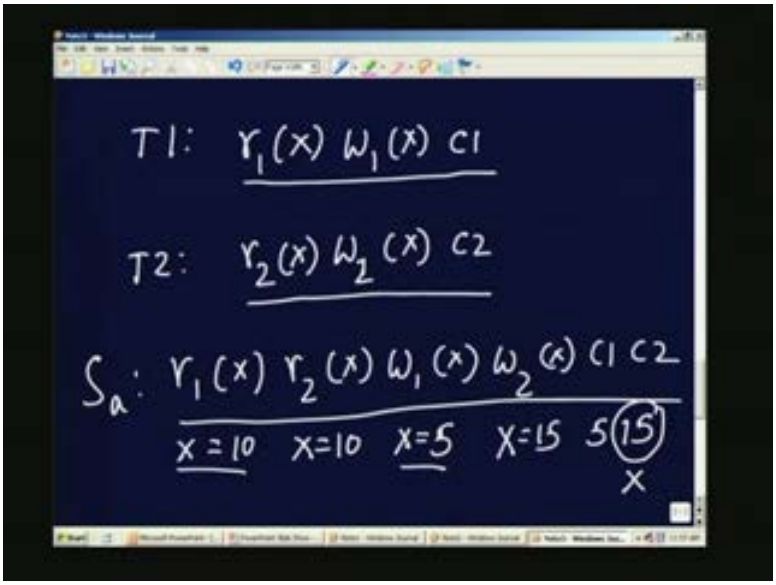
This is basically what we mean by a schedule S_a or some schedule. a schedule is nothing but a sequence of operations that we are performing on the database from transaction T one to T n some n transactions. Now the transactions keep continuously executing on the database which means that as the transactions are coming, we are executing the read and write operations relating to this transactions and either committing or aborting the transactions at the end of what they perform and this process continues. Now as it's happening, we need to ensure that whatever operations are performed by the database is actually leaves the database in a consistent fashion.

For example we can look at the schedule that we have just produced and see what would have happen if the sequence of instructions are executed as shown in a schedule S, S_a in

this particular case. Now as you can see here the value of x, let us say at this point of time is basically 10 at the start of the execution so read x read one of x would have resulted in reading the value of x as 10, read two x will also result in reading the value of x as 10 then a write would actually resulted whatever computations that was done and writing the value.

Let us say actually after this computation, I actually write a value of x equals to 5, I subtract 5 form 10 and let us say the w to actually adds 5 to 10 which means that it results in 15 and after that c_1 commits which means the value of x will be written as 5, when c_2 commits value of x will be written as 15. As you can see here, if the transactions have been executed one after the other, the end result would have been different x, the end value of x would have been different from what was actually produced here.

[Refer Slide Time: 14.38]



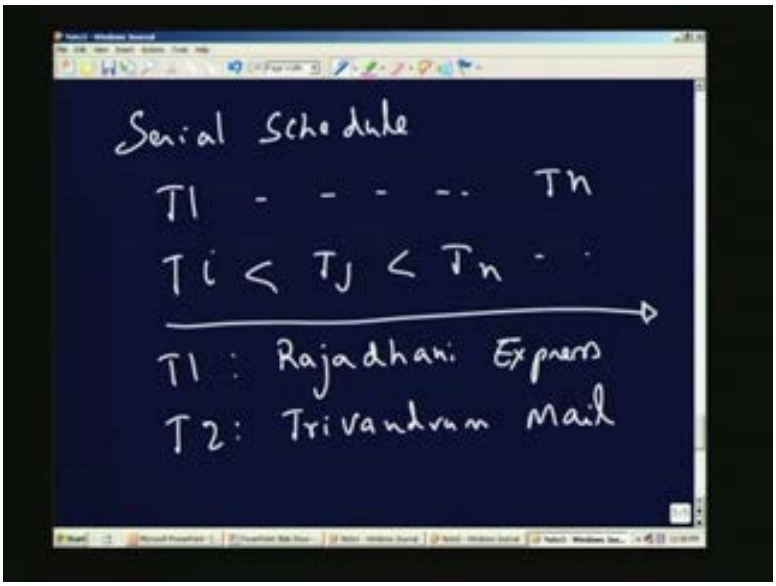
If you carefully notice the effect of the first transaction are last on the database if because the second transaction also read the same value of x, not produced by the first transaction and hence this will result in an inconsistent operation of writing the values onto the database. We will characterize this consistency more carefully by looking at schedule and then trying to characterize the schedules in terms of how they basically, whether they are consistent schedules or whether they produce what we see as consistent results on the database.

Now a simple case is where the transaction T_1 is written assuming that this is the only program that is operating on the database. Let us assume that T_2 is also written assuming that this is the only transaction that is executing on the database. This requires that T_1 is consistent as long as it is executed from start to finish. Consistent from start that is start to end, all the instructions are executed without any other transaction seeing the values used by T_1.

Similarly the same thing is true with actually T_2 which actually means that it will also assume that the start to end is executed as far as T_2 is concerned without being interpreted. What this means is either you execute T_1 completely before T_2 or you execute T_2 before T_1. This is a very important notion here of saying that I have what is called a serial schedule. A serial schedule is one where the transactions are executed in such a way that new transaction is executed only after finishing the earlier transaction. So in this particular case, if you say a serial schedule all the transactions should be executed one after the other.

For example if I have n transactions, there should be a mechanism by which I categorize T i less than T j less than T n like this which actually produces a serial schedule. The only problem with the serial schedule is this is very limiting because it's possible that these transactions can be executed concurrently, simultaneously still actually producing correct results. for example, let us assume that T_1 is booking for rajdhani express and then let us say T_2 is trying to book the reservation for let us say Trivandrum mail. Now there is no conflict between these two which actually means that even when these two execute concurrently, there is not going to be any problem in terms of the end results because they are not conflicting with each other.

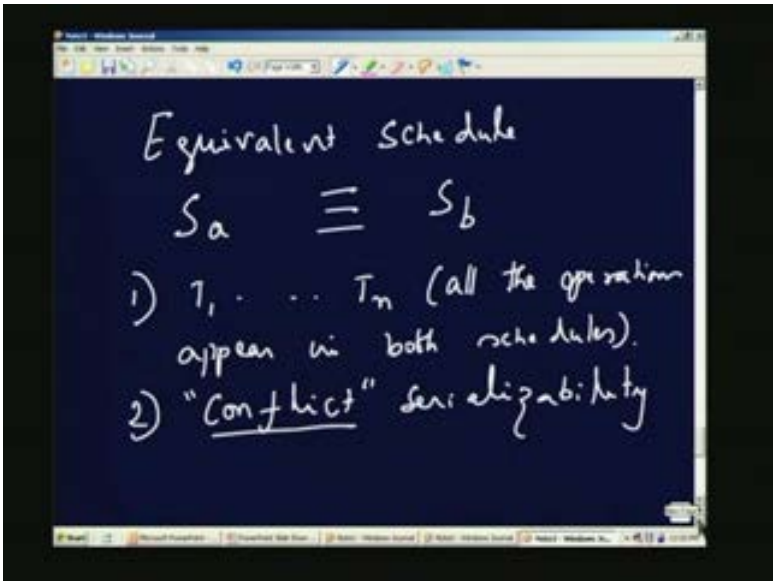
[Refer Slide Time: 18.20]



So by unduly restraining that I know the transactions should be executing one after the other would only affect the database performance. We can, when they are not conflicting suddenly we can execute them in a parallel way and get better performance from the database rather than enforcing a serial order. This is the first important concept of trying to look at a serial schedule. Now what we will try to do is how exactly one can think of a serial schedule and produce a serial equivalence schedule. Not exactly serial schedule but equivalent schedule to a serial schedule. Now what we do is for this, we will define the notion of equivalent schedules. What this means is two schedules can be seen to be equivalent under certain conditions.

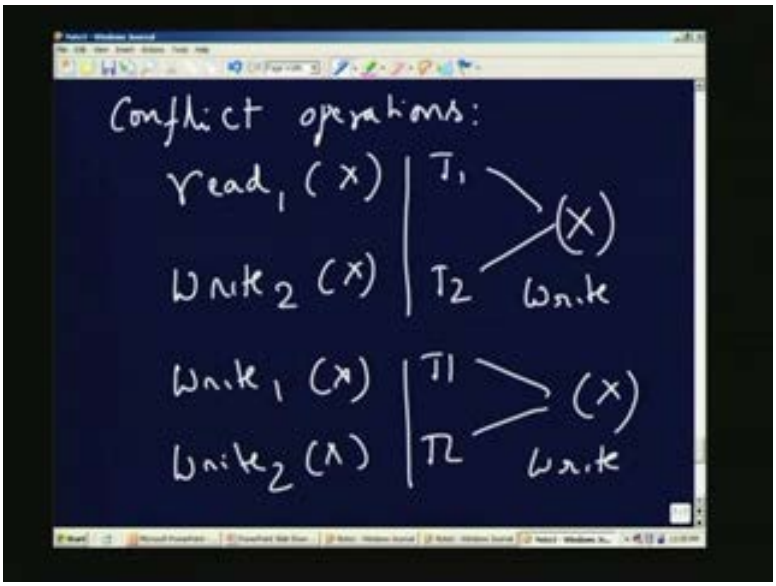
For example, let us take a schedule S_a and a schedule S_b and define what we mean by an equivalent schedule. Two schedules are equivalent if basically all the operations which appear in S_a also appear in S_b. For example for transactions T one to T n, all the operations will define all the operations appear in both schedules. Now after this point to define equivalent schedule, we need the property of saying what kind of equivalence is this between the two schedules. one is to say as i have actually shown in the last slide that when they are actually not conflicting, it doesn't really matter how the operations actually appear in the schedule S a and S b. For this what we define is what we call as conflict serial ability which actually means that only when transactions are conflicting with each other, those operations alone need to taken care, other operations need not be, no they can be executed in any possible order. What this actually means that you need to focus between the two schedules S a and S b on what we see is the conflicting operations and ensure this two conflicting operations are done in the same order in S a and S b.

[Refer Slide Time: 21.22]



Now for this, I will define what it means to say two operations in transactions conflict. conflicting operations are the following. Now one of the operations of the transaction is basically a read operation. Let us say read one of x and the other operation is essentially a write operation. This is conflicting because the transaction T_1 and T_2 are operating on the same data item X and one of the operations is a write in which case we say these two operations are conflicting with each other. There are also other probability where the first operation is a write and the second operation is also a write which essentially means that T_1 and T_2 again will be conflicting with each other with respect to this write operation.

[Refer Slide Time: 22.39]

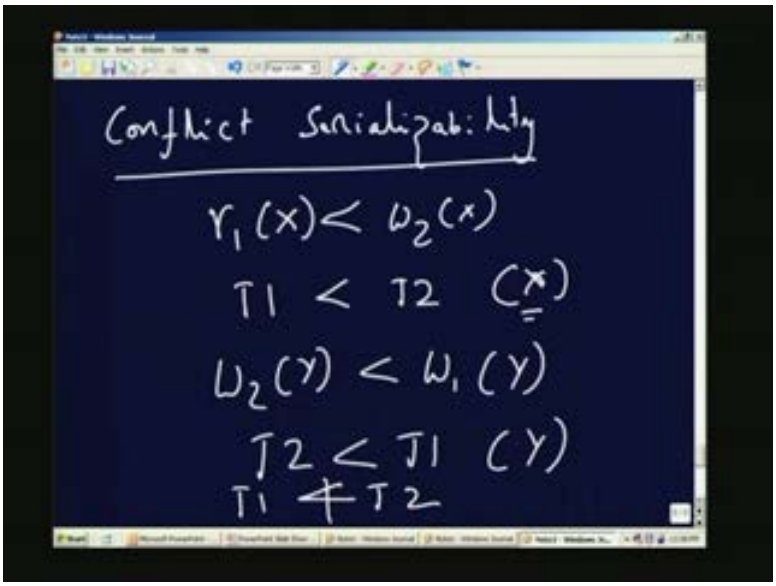


So from this we can infer that, if one of the operations is a write and two transactions are operating on the same data item in this case X and one of the operations is a write then we say that these operations are conflicting. Now all the executions that a transaction does need to worry about how these conflicting operations are executed in a schedule. Now to give this notion what we say is all that we will be worried about is conflict serializability. That means you don't need to really concern yourself about serializing all the operations but you have to actually do what is called the conflict serializability. That means when operations are conflicting, you have to do what is called the conflicts serializability.

Now in this particular case, let us say the two operations r one w as shown in the last slide which means that r one x and w two x, if they have performed this operations one after the other, it essentially means that T one has executed before T two as for data item x is concerned. Now it could be the other way round also, depends on how this is done in the schedule, how exactly this conflicting operation is performed. But suddenly if r one x is performed before w two x, this is the order as far as data item x is concerned.

Now if the transactions are also conflicting on another data item, let us say y and on y database has actually performed operations such that w two y occurred before. Let us say w one y on the database item y if you actually want to order this transactions. Then it is going to be T_2 before T_1 as far as y is concerned. Now if these two operation occur in this order in a schedule it essentially means that there is no fixed order as far as T_1 and T_2 is concerned because as far as x is concerned T_1 is before T_2. As far as y is concerned it is the other way around T_2 is before T_1 which exactly means that I no longer can infer from this T_1 occurred before T_2 or T_2 occurred before T_1 which essentially means that on the conflicting operations, there is no way to actually serialize the transactions by saying T_1 before T_2 or T_2 before T_1 in which case such an execution is not obeying conflict serializability because the conflicting operations are not serializable in the schedule.

[Refer Slide Time: 25.35]



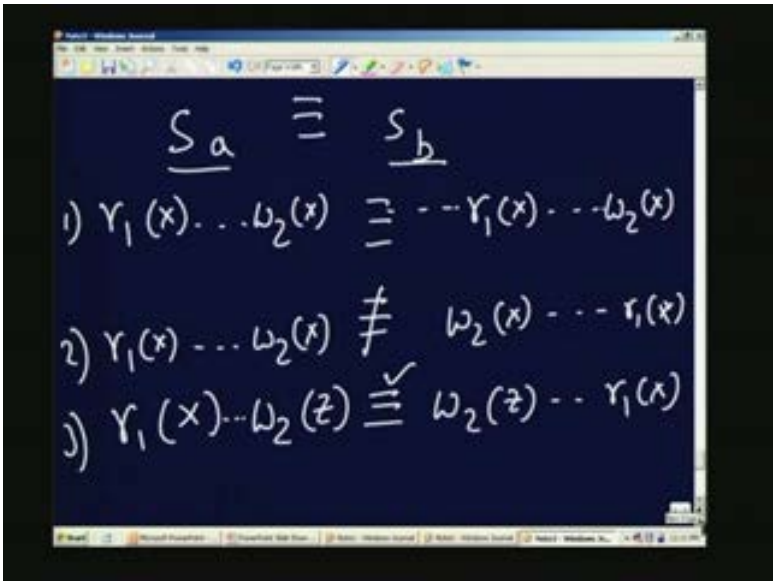
Now to actually give the notion how exactly this can be further looked at. two schedules S a and S b can be seen to be equivalent, if the conflicting operations appear in the same order between these two schedules which means let us say I have a schedule where there is a set of operation that are performed in this particular fashion on schedule a. If they actually are performed in the same order in S b, there could be some other operations interleaved but then as long as the final order that I see between these two conflicting orders is the same then I say these two are equivalent schedules. Now this won't be equivalent if the order in which they appear here is different from each other.

For example if w x in the other schedule comes before r one x then they are not equivalent schedules. If the operations are not conflicting, it doesn't really matter in what order they are appearing. For example let us say there is a data item here in this schedule x on which actually transaction one reads this here and then there is another data item which the transaction two is actually writing which is z in this particular case. Now these are not conflicting operations. now even if you change the order of this operations, it still doesn't matter because they are not conflicting operations which is equivalent to saying that even if you now transfer w_2 to before to r_1 x, it's still is okay for me because these operations are not conflicting and hence we don't really care to actually worry about the order of non-conflicting operations.

Since in the first one, r one x we just recap what we are trying to do here. Two schedules S a and S b are equivalent as long as the conflicting operations appear in the same order between the two schedules. in that sense the two schedules, one and two as shown here three cases where S a is you know some random order where r one x appears before w_2 x, r_1 x indicates that this is a read of transaction one on x. This is read write of transaction two on x since these two are conflicting, the way in which the schedule a there appearing is the read x is before the write x. If it appears in the same order in S b also then we say they are equivalence schedules that all conflicting operations appear in the same order in

the two schedules. We say they are conflict equivalent conflict it terms of equivalence, they are in terms of conflicting operations they are equivalent schedules whereas you can see they are not conflicting, it doesn't really matter in what order they appear.

[Refer Slide Time: 29.22]

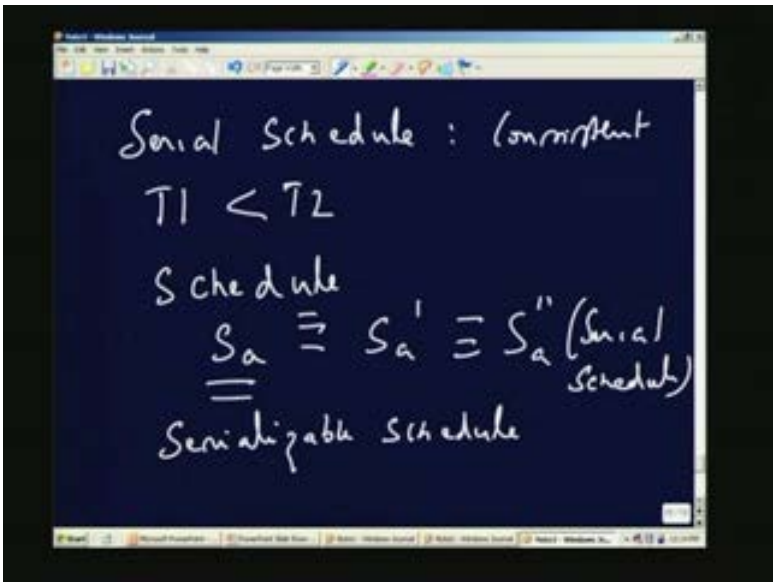


Now to introduce the notion of how we use this conflict equivalence in actually deciphering whether schedules are schedules can be use to decipher whether they produce consistent results. We say a serial schedule is always a consistent schedule. This is the benchmark to say that I produce serial schedule then it is consistent. The simple reason here is T_1, all operations of T_1 executed before all operations of T_2. I see this as consistent execution because I am able to execute all operations of T_1 before T_2 and hence the database is consistent.

What it is doing is consistent. Now when I have a schedule which is not equivalent to a serial schedule, I will try to change the operations which appear in the schedule S a. I transform now this to S a dash but this is an equivalent schedule. As long as the conflicting operations are same between S a and S a dash, this is still an equivalent schedule. This can further be transformed to S a double dash and finally this S a double dash becomes a serial schedule which actually means that I have been able to transform a schedule S a into a serial schedule.

Now we use this notion to say that S a is a serializable schedule, not serial schedule but it is a serializable schedule. In terms of the conflicting operations, the way i see executed this conflicting operations is same as S a double dash a and hence S a is basically a serializable schedule. Now what we are interested in is producing the serializable schedule because serializable schedules can be reduced by swapping the non-conflicting operations in whatever way you want into a serial schedule.

[Refer Slide Time: 31.36]



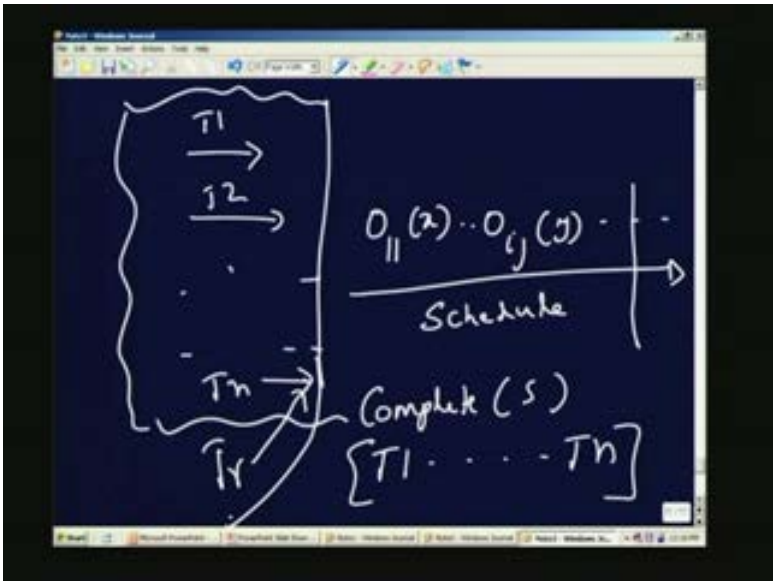
This is in effect conflict serializable schedules, conflict serializable schedules. what we are, as long as a schedule is conflict serializable and at the end of the execution you are able to show that this schedule is equivalent to a serial schedule, S a is a consistent schedule or the operation of S a is consistent and that is what we are interested as far as serializability is concerned. This is a very important notion of actually being able to serialize the transactions produce serializable schedules.

We will try to look at now is how exactly the notion of serializability will be used by database transactions to produce consistent schedules. Now for that what I will show is to start with, as the database is operating, transactions will be coming in at any given point of time into the database. Imagine for example, this is a railway reservation system which means that the passengers keep coming and keep reserving the tickets at any given point of time which means that there is this set of schedules, this set of transactions T_1 T_2 T n which keep generated at different points of time. Now as they keep arriving into the database, some operations of T_1, T_2, T_n will be executed here which actually means that to just produce this we will say O_1 x is the operation of transaction one, $O_{i\,j}$ is a general operation on a data item y on the database. And this is how this operations are executed as far as the database is concerned. This is what we actually mean by a schedule.

Now since the database will be operating continuously, these transactions keep coming regularly into the database. It is not possible at any given point of time to actually close the transaction, close the schedule and say I have actually looking at a particular schedule. What this requires is at a given point of time if you want to analyze, you need to put a break point and say I will actually take what is called a projection of this for the complete schedule which means that all those transactions which have actually committed or aborted as far as the schedule is concerned whose operations are all performed that will be included in this complete schedule.

Let us say up to T some r, I have been able to now do the all the transaction execution then I will say I will execute from T_1. Probably I will do a slightly change here and i say T r to just make sure that we get the thing right, T r comes later which actually means that up to T_1 to T n these are completed transactions. That means the complete projection of schedule S will include up to T_1 to T n which means that my schedule S which is on a partial schedule of all the transactions coming in which includes the T r here. This T r actually goes into this schedule here and then this from this actually, I am actually projecting from this set here to complete S.

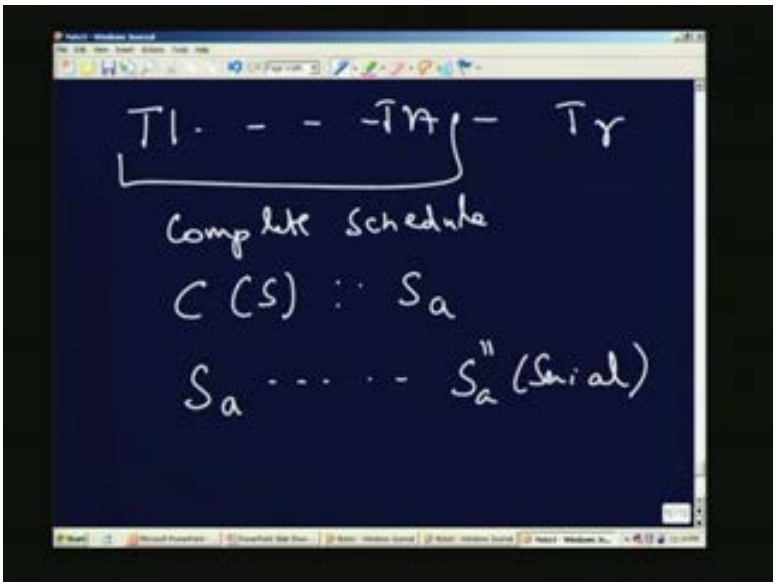
[Refer Slide Time: 35.20]



Now which actually means that T_1 at a given point of time the T_1 to T r are the total set of transactions that I can consider but T_1 to T n is a completed set. And I am actually looking at complete schedule which means that all the operations of these transactions have been included in the complete schedule. Now when you actually take the C of S as the projection then we want to apply the notion of whether this schedule produced is a correct schedule or not. This is when we are going to apply the equivalence this schedule let us say is called S a. Now I apply on S a and then see whether this S a is reducible to some serial schedule.

A simple check of seeing whether a serial schedule is being produced or not, what we can see is a simple algorithm which constructs a graph showing how the transactions are executed in your system.

[Refer Slide Time: 36.33]

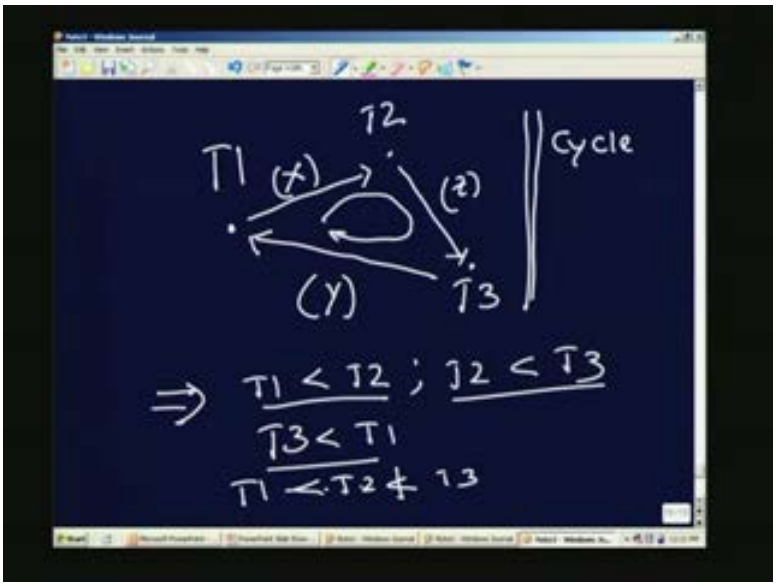


Assume that actually T_1 came into the transaction system, now you try to actually see T_1 put it as a point in the graph. Now let us say another transaction actually T_2 comes into the system. Now assume that T_1 and T_2 actually conflict on a data item x and then this conflicting operations on data item x are executed in such a way that T_1 appears the operation of T_1 appears T_2. Then provide arc showing that there is a precedence relationship between T_1 and T_2 showing that T_1 comes before T_2 as far as this operation is concerned.

Now we assume that there is another transaction which came T_3 and this is conflicting on let us say an item z and this is the operation as far as this is conflict is concerned. Let us assume now between T_3 and T_1 there is a relationship in terms of conflict on Y and if in this case T_3 is executed before T_1 we have essentially a cycle in this graph which means that the conflicting operations actually in terms of the graph of forming a cycle.

What is the meaning of this cycle? Assume now T_1 is less than T_2 as far as first arc is concerned T_2 is less than T_3 as far as second arc is concerned. As far as the third is concerned T_3 before T_1 which shows that it is not possible form this to actually say any particular order in which these three transactions have executed. The last one will be wrong because by a transitive relationship T_1 should have finished before T_3.

[Refer Slide Time: 39.03]



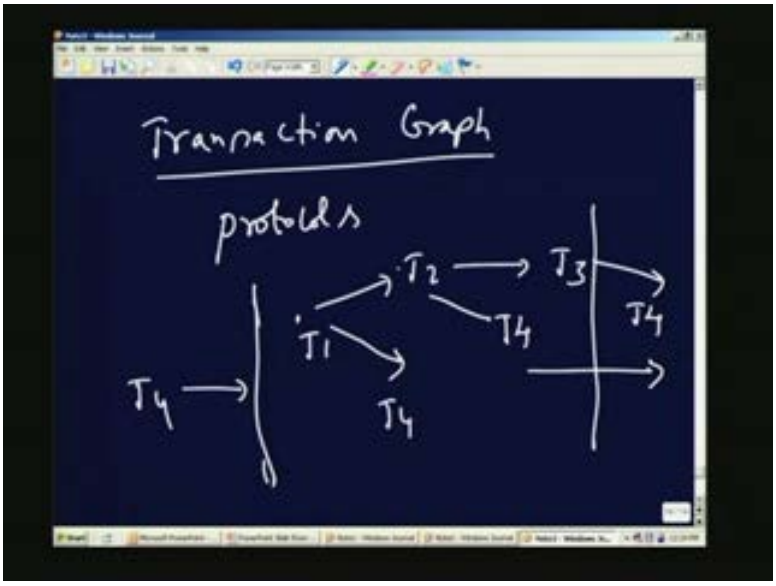
And this is what exactly is done to show whether a schedule is a serial schedule or not. There should be any cycle if there is a cycle it shows that it is a non serializable schedule. The presence of the cycle in the transaction graph is shown to produce non serializable schedule because a cycle prevents you from coming with a order in which the transactions are put in a particular order of one being finished before the other and hence this will not produce a serial schedule. I think this is very important as far as the concept is concerned because we use the transaction graph to understand whether a protocol is basically produces serial schedule or not.

As a later lecture towards the next lecture, what we are going to build is several protocols for actually building the or executing the transactions. Essentially these protocols will try to construct the transaction graph in such a way that an incoming transaction is put in the correct order as far as the sequence is concerned. All that you want to do is you don't want a cycle in the graph, you are not trying to produce a cycle in the graph and the protocol has to ensure that there is no cycle in this particular graph and that's what exactly we can understand. For example imagine that there is currently a current transaction graph looks something like this. Let us say I have three transactions active in my database and this is the current sequence as far as the conflicting operations are concerned.

Let us say a T_4 now comes into the database at this point of time. I have several options of where I can put this to avoid a cycle from coming in into this particular graph. It is possible for me where the protocol always allow the transaction graph to grow only in the forward direction which actually means that it is possible for me to keep this T_4 here that is one possibility or I can put T_4 here or I can put T_4 here which actually means that the graph goes only in the forward direction and when it goes in the forward direction it prevents any cycle from occurring because you are not going to but a backward arc. As

long as you don't put a backward arc, you let the transaction graph only in forward direction.

[Refer Slide Time: 41.38]



It is possible for you to avoid a cycle in the graph. The other possibility is it is possible for the protocol to decide to put it even before which actually means that if it can be made to read the value, let us say there is a write x here and I let this transaction read the value of x before this is modified then it is possible for T_4 to be put before T_1 even when it is coming after the graph. In that sense it is possible that T_4 before T_1 also doesn't produce a cycle and hence this is also a correct schedule. So it depends on how exactly the transaction graph can be allowed grow by these protocols.

Essentially the concept is a graph is constructed and a cycle is prevented from happening in the graph. So I think we understood now the basics concept of how exactly the serializable schedules is used by the transaction system. Now what I am going to show in the next few minutes is to look at other kinds of consistency requirements. An interesting thing to understand as far as consistency is concerned is to look at basically T_1 executing before T_2 is a correct thing to happen. But this need not be the case for example if you look at debit and a credit transaction it is possible that any number of debits and credits can be interleaved as long as the debit occurs as one unit and credit occurs as one unit.

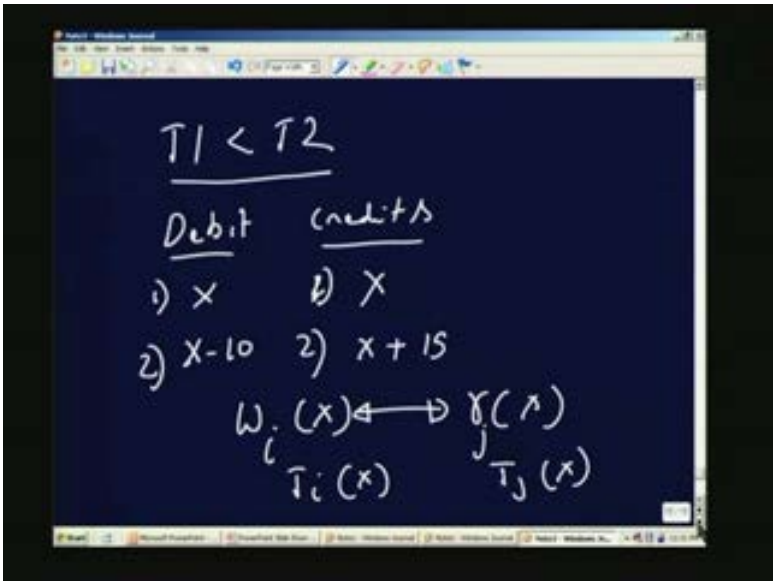
Now this is very interesting because we can start looking at what is a operational semantics and try looking at whether the way the transaction execute is consistent or not. To give a more deeper treatment of this we will take a simple example and then see what exactly we mean by this understanding the semantics and seeing whether the execution is right or wrong as far as the database transaction execution is concerned.

For example imagine I am actually having account and I do a debit on my transaction which actually means that this is basically withdrawal and I actually add some numbers

after that means I credit into my account which is equivalent to saying that I read the value of x here and then I basically add some number here. As long as the read of x is consistent with respect to the write, it does produce consistent results.

What this means is there is basically a write that is happening on x before the read is happening on x. Now this write can be by any transaction, let us say this is by the transaction i and this is by another transaction j. This is the relationship between the two transaction in terms of, I am actually reading the value let us say the j is reading the value produced by i that means T i has actually produced the value of x which is being read by T j.

[Refer Slide Time: 45.38]

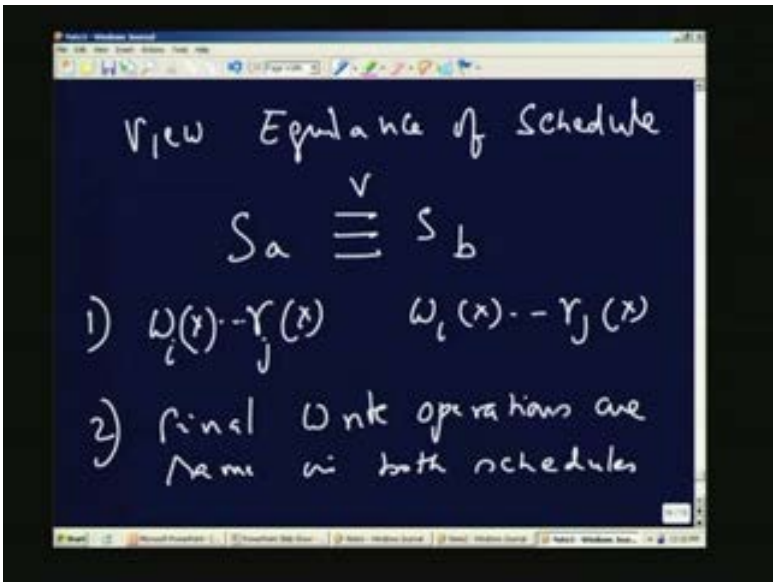


Now as long as this relationship is maintained between transactions in terms of how exactly they read the values of the previous transactions and this is actually maintained between the two schedules, we say that two schedules are equivalent in terms of views. This is called the as appose to actual conflict serializability, this is called the view equivalence of schedules.

Now what this means is two schedules S a and S b are view equivalent as appose to conflict operations equivalent schedules, they are view equivalent schedules if the way actually read operations and the write operations are related is they actually read between the two schedules, the operations are actually the same in terms of the way it has been produced and read. Now if this order is changed between the two schedules then it is basically not, they are not view equivalent.

The final writes between the two schedules also have to be, this is first requirement. The second requirement is the final write operations are same between the two schedules. The same in both schedules these are the two conditions that need to be satisfied for two schedules to be view equivalent.

[Refer Slide Time: 47.06]



It is possible that view equivalence can also be seen as producing consistent results. For example if you look at the typical case of what we considered as debit and credit transactions you know occurring simultaneously. It is possible to see that view equivalence is will produce correct results as appose to the conflict serializability.

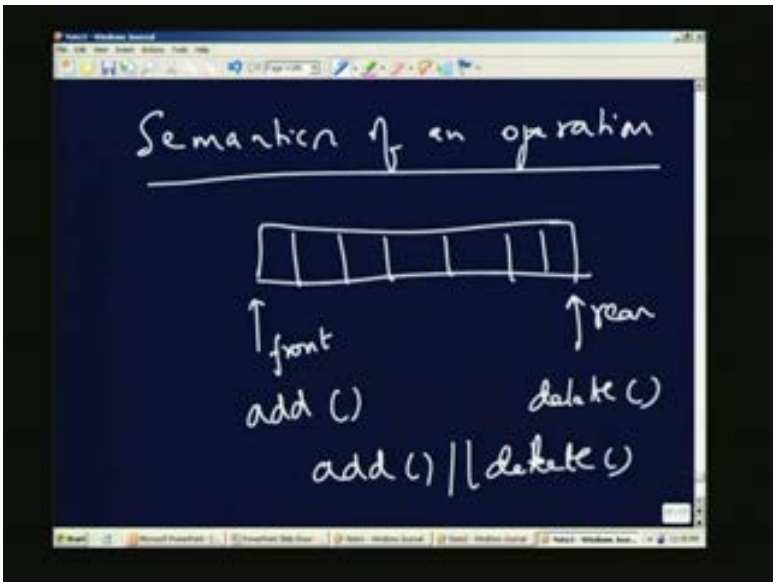
Now, this is interesting because we will start realizing that it is possible to enforce correctness by understanding what is happening with the transaction semantics. For example; it is possible to look at semantics of operations, finally we can look at semantics of an operation and then see whether a particular execution of this operation can be correct.

What I am going to do is I will take a very simple example to show how semantics can be applied for understanding the consistency criterion. It is possible to say that I have I will take as slightly different example here to show what is semantics of an operation. We can take a simple queue as shown in this particular figure. Now, the queue will have what we say as a front pointer and a rear pointer and it will basically have two methods which can be executed which is basically, an add and a delete.

Now, if you can carefully look at how exactly the queue can be left in a consistent condition when adds and deletes are happening simultaneously. Now, you can see that basically an add will will happen at the rear end and a delete will happen at the front end.

Now, adds and deletes can suddenly be concurrent assuming that the queue is not full, the queue is not empty; under those conditions, adds and deletes can occur concurrently because add is actually trying to manipulate the rear pointer, delete is actually manipulating the front pointer.

[Refer Slide Time: 49.28]

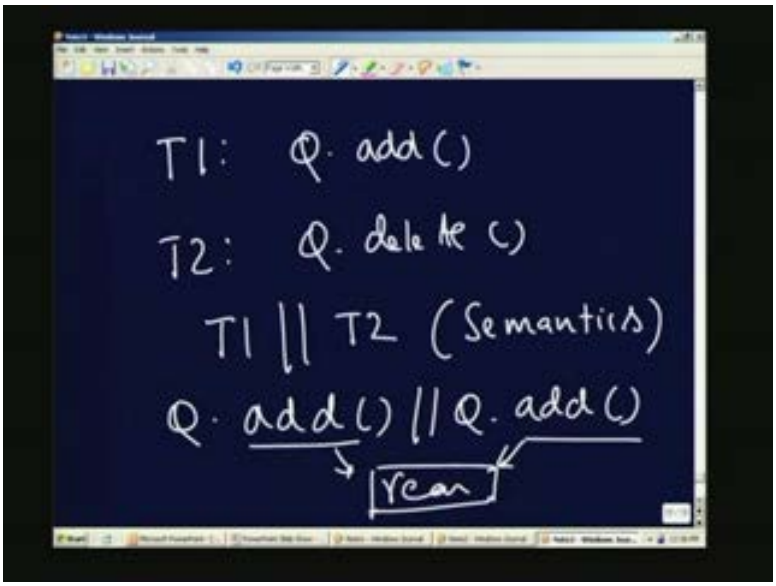


This is very important to look at little more deeper. For example; let us say, there is T1 here and you are actually saying Q dot add. Now, there is a T2 which is actually saying, Q dot delete. Now, we know that from the semantics of add and delete, T1 and T2 can happen concurrently and still produce correct results.

This is basically semantics, knowing the semantics of the operations; I am able to say that these two produce consistent results. Now, if you basically further say that two adds can also happen simultaneously and I have a mechanism for producing, know two adds working simultaneously, it is possible because all that you need is lock the rear pointer and if you allow the rear pointer to be obtained by each add separately, then you need to lock only the rear pointer and ensure that this two adds at the add level can be concurrent. But at the rear level, they will be blocking each other.

That means the access to rear will be may consistent; but at the add level, they can be still working parallely or concurrently.

[Refer Slide Time: 50.54]



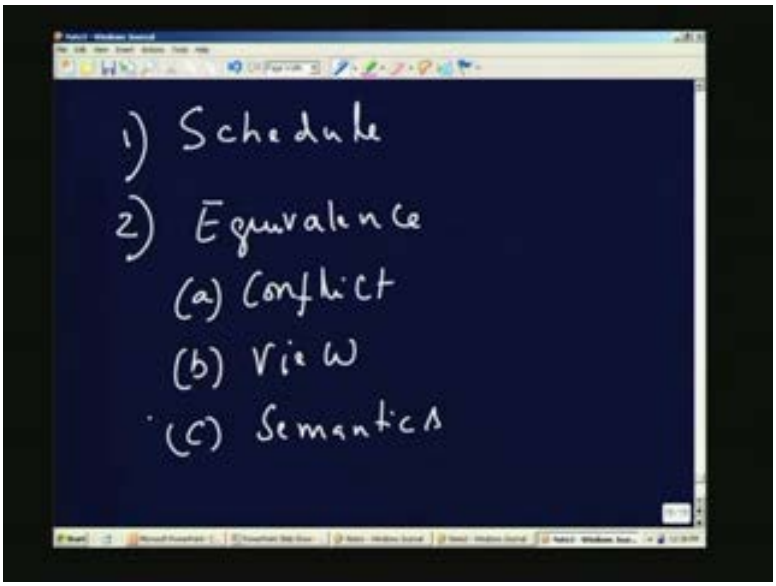
And, this is basically understanding the operation of the transaction and applying what we call as semantic consistency. Since you know what is semantic actually means that, you know the meaning of the operation and apply the meaning of the operation to decide whether something is consistent or not. And, that is very interesting because it is possible to apply a much greater level of consistency criterions by understanding the meaning of the operations.

To just recap what we have done in this particular class and then give you some indicators of how exactly the to go on further reading in this particular subject, I typically covered the idea of what is basically a schedule in this particular class and what I have also done in this particular case is I have actually produced equivalence schedules and this equivalence schedules are from different aspects.

Two schedules are shown to be equivalent from a conflict operation point of view by saying that if the conflicting operations are executed between the two schedules in a particular way, the same order is maintained between the schedules; we call that as conflict equivalence. We also showed view equivalence which actually means that the writes produced by one schedule. The writes and the reads, the way they occur on operations are same between the two schedules. We call that as a view equivalent schedule.

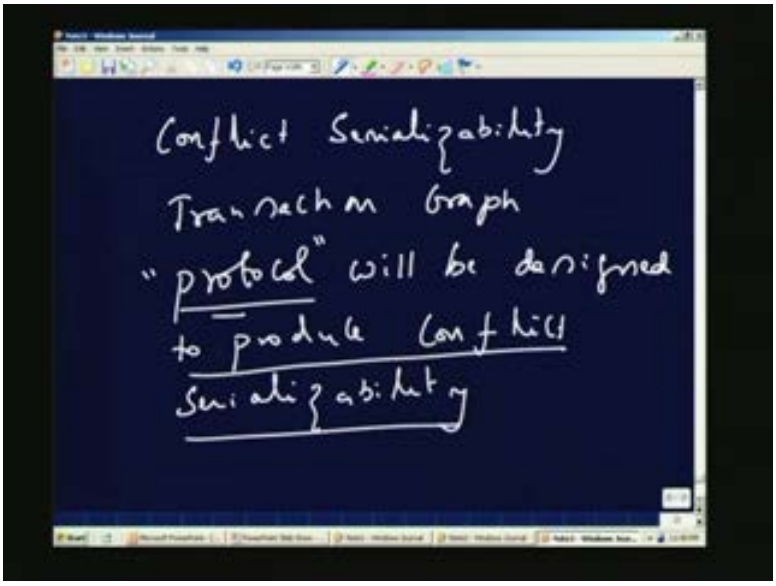
Finally, we also showed what is called semantics and based on semantics, how the schedules can be seemed to be equivalent. You can do the commutative operations as long as they are parallel, whatever order they appear still the schedule is right as long as you commute the commutative operations are performed in any order is still will be producing consistent schedules and so we actually shown How exactly we look at equivalence.

[Refer Slide Time: 53.15]



And, what we have further shown in this particular class is typically, how simple case of conflict serializability can be achieved by constructing a transaction graph. A transaction graph is constructed by producing before and after relationships on the various transactions and that is how actually the conflict serializability is achieved by constructing the transaction graph.

Finally, we have actually shown, how exactly the protocols, various protocols will be used, will be designed to produce the serializable schedule. The criterion for this is will be designed to produce conflict serializability. What I am going to do is in the next class, I am going to discuss a series of protocols which actually produce conflict serializability.
[Refer Slide Time: 54.43]

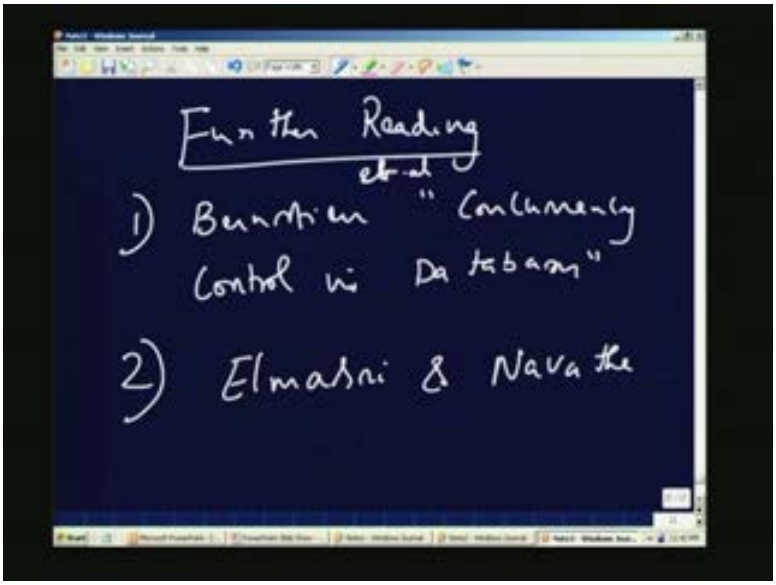


We are going to look at a set of protocols. We start with a most popular protocol of two phase locking and show how two phase locking will produce conflict serializability and also go on to show other kinds of protocols that exploit the property of the constructing the transaction graph without any cycle. That is the essential property is here, there should not any cycle as far as the transaction graph is concerned and the protocols exploit this property of trying to construct the cycle and we essentially can divide the protocols as being optimistic or pessimistic on how actually they construct the transaction graph.

We are going to take this in the next class of looking at the protocols and seeing how different protocols can be constructed for producing serializable schedules. As a thing of further reading on this, you can typically look at there is a book by Burnstien on actually concurrency control in databases. This is an excellent Burnstien and others. Basically a book on concurrency control and you can have a look at this the book as a further reference.

I have also used basic the foundation thing was used by the book on Fundamentals of Database Systems by Elmasri and Navathe, Elmasri and Sham Navathe.

[Refer Slide Time: 56.39]



I have used the chapter from this book while doing this particular; foundations on concurrency control. What, I am going to also do as part of next lecture is while while doing the protocols at the end of the next lecture, I am going to introduce a few problems and try solving them at the end of the next lecture. We will stop here for this lecture.