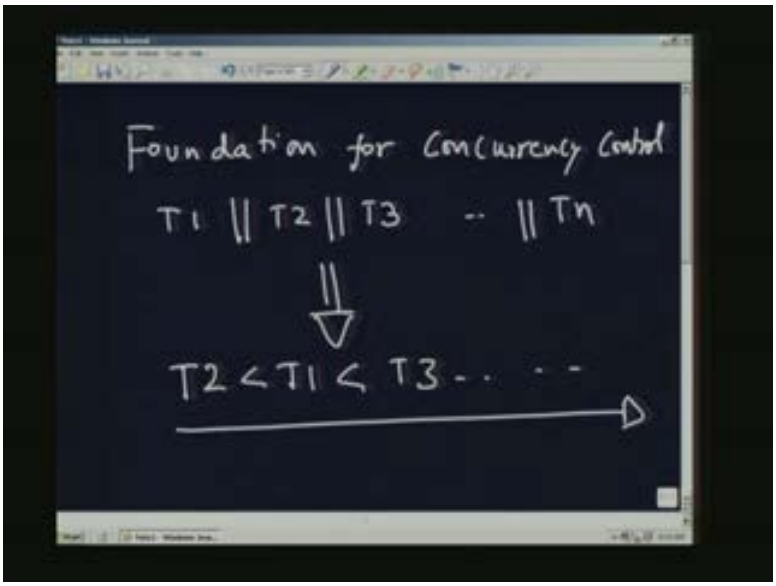
Database Management System
Prof. D. Janakiram
Department of Computer Science & Engineering
Indian Institute of Technology, Madras
Lecture No. # 20

Concurrency Control – Part -1

Foundations for concurrency control in particular what we did was we looked at the problem of serializability when transactions are executing concurrently. For example there are a set of transactions and all these transactions are executing concurrently. We were looking at how this can be interpreted as a serial execution of the transactions. For example some order in which they could be seen to be executed is what we looked at a theory for correct execution of the transactions.

[Refer Slide Time: 02.21]



We also looked at two kinds of a serializabilty, the conflict serializability where the transactions which are executing on the same data item when they conflict, how they can be reduced to serializable transactions. That is what we meant by conflict serializability. We also looked at other forms of serializability like view serializability. What we are going to in this todays class is we are going to look at specific concurrency control protocols. And to begin with what we are going to do is we are going to look at what are the different types of algorithms that will allow concurrent execution of transactions to be serializable that means which produce correct execution of the transactions in a database.

So what we are going to do now is we are going to look at what are the algorithms that we use for achieving serializability and these algorithms are called concurrency control algorithms.

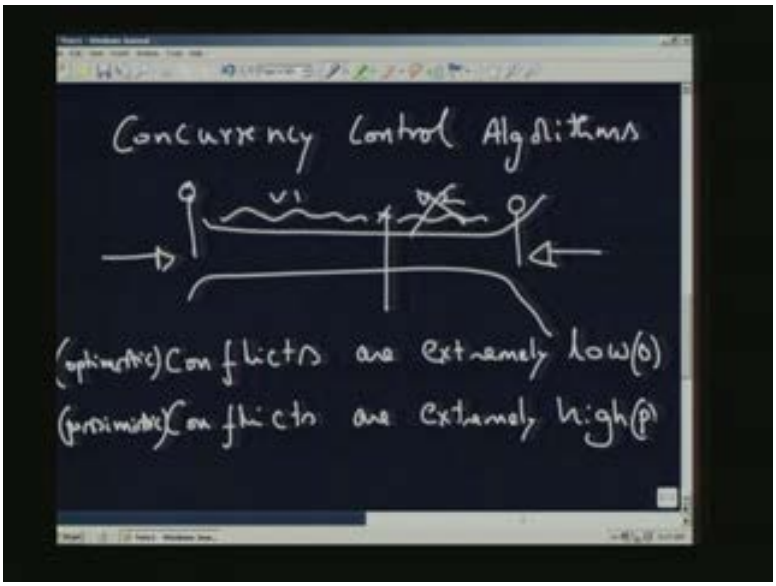
Now there are two broad classes of these concurrency control algorithms based on how they actually view the system. Some view the system in an optimistic fashion and some view the system in a pessimistic fashion. What we mean by optimistic and pessimistic, I just explain with a simple example. Now if you basically look at a narrow pass bridge where vehicles are crossing this bridge and this is relatively a very very low traffic bridge. We don't really look at, the vehicles will hit each other when they are passing through this narrow bridge which in effect means that there are actually vehicles coming from opposite ends.

So you actually assume that relatively, the conflicts are very low which actually means that a scenario where the conflicts are extremely low is one situation. There could be another situation where the conflicts could be extremely high which means that this is a very high traffic bridge and hence you assume the possibility of two vehicles finding themselves passing through this narrow bridge at the same time is likely to be high. In either case what we have to see is the two scenarios are different system configuration. So what we will do is we will broadly classify conflicts being extremely low and conflicts being extremely high.

Now if you typically look at the way we would see when the conflict are extremely high is we basically will put some kind of a traffic signal here and we will ensure only one of the vehicles is into the narrow pass bridge. That means we in effect will ensure that only one of them is going to be inside. If we don't put the traffic lights her, very often what will happen is two vehicles will get into the bridge and realize they are in conflict and one of them has to back track.

For example if we assume that one vehicle V_1 has passed up to this point and V_2 has passed up to this point and you find there is a conflict here. One of them has to backtrack and this basically involves lot of hard work and hence when the conflicts are very high, it is probably not advisable to actually allow the vehicles to get in without any kind of a control. The situation where the conflicts are low is seem to optimistic that means you actually believing that they conflict are less which is an optimistic approach or you view the system in an optimistic way, the second one where the conflicts are very high, you look at it has pessimistic. So this is broadly classified in the two scenarios optimistic and pessimistic scenarios.

[Refer Slide Time: 06.47]

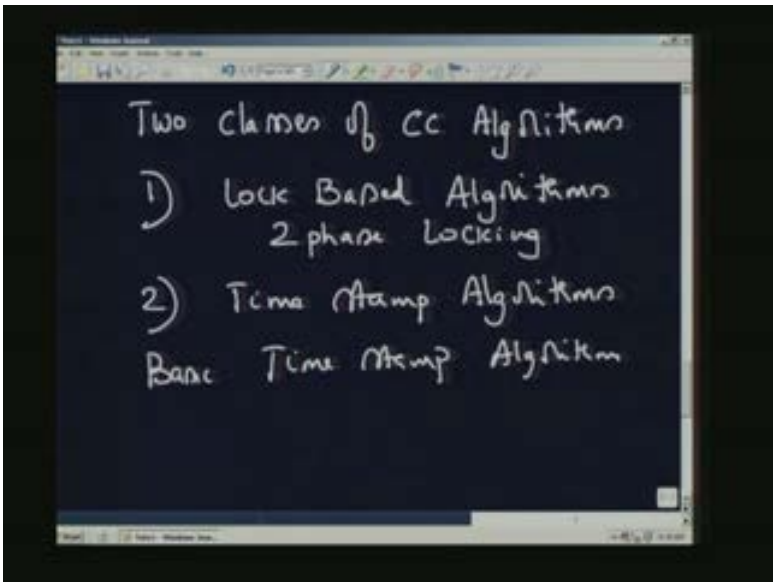


Now what will do is we will see algorithms that operate on the database system assuming that the system is an optimistic scenario. And those algorithm which actually operate assuming that they are in the pessimistic scenario. For example if you know chennai for example in the mount road, if you assume optimistically no vehicles are going to hit each other and remove the traffic lights, everything is going to be chaos there. But on the other hand if you know IIT Madras at gagendra circle, you are not going to install a traffic light because we don't have that much traffic, it is going to be cumbersome.

So in some sense if the conflicts are rare and if you apply the other algorithm, it is going to be overkill. So typically we have to understand, what is the scenario in which the system is in whether it is in optimistic scenario or in the pessimistic scenario and based on that you should be applying this algorithms. Now broadly what we are going to look is these two classes of algorithms which are actually categorized, two classes of cc algorithms but it is possible that there are more algorithms than these two classes.

Now, the one class which we are going to look at is the lock based algorithms which all assume pessimistic scenario which assume that the transactions are going to conflict with each other often. The other basically a time stamp based algorithms. The time stamp based algorithms assume that the system is in the optimistic fashion and hence conflicts are there. What we are going to look at is one lock based algorithm, the most popular algorithm called the two phase locking algorithm and we are going to look at a basic time stamping algorithm for the time stamp basic time for the time stamp based algorithms, we look at the basic time stamp based algorithm.

[Refer Slide Time: 09.16]



Both these classes will tell us that one belongs to the pessimistic class and other belongs to the optimistic class. Now what I am going to do in the rest of the lecture is look at the lock based algorithm called the two phase algorithm in detail and explain what are the properties of this two phase algorithm and we will study this algorithm in detail and this is the most popular algorithm implemented in practice as well. So we are going to spend some time looking at two phase locking algorithm used for concurrency control in database systems.

As the name suggests, this actually is a locking based algorithm. The lock is very important here because what we are trying to do here is when transactions conflict on data items; we are in essential going to allow these transactions to lock the data items. This is equivalent to saying that once the transactions is locked the data item, it is not accessible for other transactions and in effect that protects the data item from being wrongly manipulated by other transactions. So locking is a concept that is used in this particular case.

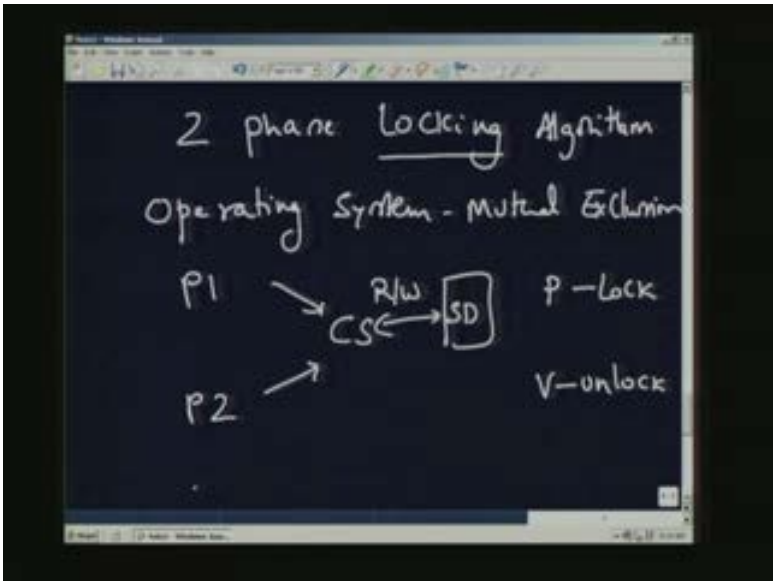
Now to explain the concepts of locking in more detail, we have a problem which we are also familiar in operating systems called the mutual exclusion problem. What actually the mutual exclusion problem will try to do is it actually tells that if there is one process P_1 which is trying to be in a critical section. The critical section is nothing but as piece of code that is trying to address a share data structure between P_1 and P_2.

Let us say there is a shared data structure here between P_1 and P_2 and whenever P_1 is in the critical section, it means that it is trying to manipulate the share data structure here. Now only one of them can be allowed to be in the critical section for manipulating. This is basically the read write operations that they might perform on the shared data structure.

So what in a sense the mutual exclusion problem does is it allows only one of the processes to be in the critical section. If one process is in the critical section, it excludes the other process from being in the critical section. This is what we understand in operating systems as a mutual exclusion problem.

Now this is achieved in the operating system by using two operators called the P and the V operators. P is essentially a lock operator. So when you actually apply a p operation, the process actually uses the P to lock the critical section and uses v to unlock the critical section. This is like I know two people cannot be in the room simultaneously then what the use is they use a lock and once somebody acquires this lock, gets into the room unless he actually unlocks it and comes out of it, the other person cannot enter into the room. That is what actually prevented by using this P and V operators in the operating system context.

[Refer Slide Time: 13.00]

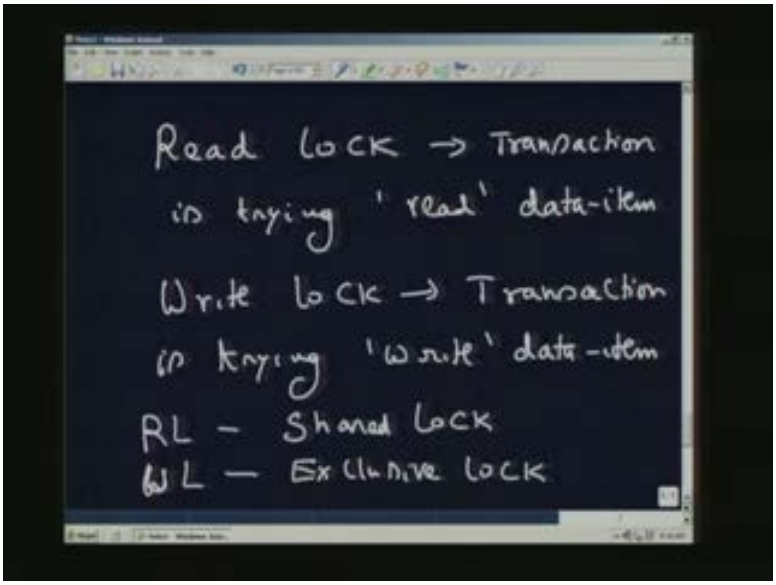


Now in the case of databases, it is slightly different than being just mutually exclusion problem because what we are actually doing in the database context is we are actually assuming that there are different types of locks, there could be a read lock. Whereas in the case of a mutual exclusion problem, we have only one kind a of lock but here there are different types of locks that we introduce. Read lock actually says that transaction is trying to read the data item, is trying to read. This is very important that means it is not going to right on the data item is only going to read the data item.

And if you say there is a write lock, the write lock means that it is trying to write, the transaction wants to write onto the data item. Transaction is trying to write on the data item. It is also you can say read lock, I will abbreviate here afterwards read lock with an RL. An RL shows that the transaction only wants to read the data item and this is also called a shared lock because it's possible that more number of transactions can start reading the item at the same time.

It is shared, that means there won't be any violation if more than one transaction tries to read the same data item. But at any given point of time, only one transaction can write on the data item. So when we abbreviate write lock as WL, this also is called exclusive lock that means this lock will be granted only to one transaction whereas the shared lock can be given to multiple transactions at the same time.

[Refer Slide Time: 15.05]



By actually distinguishing the kind of locks, we will in sense that try to increase the concurrency because whenever a transaction acquires a read lock, it is possible for other transactions also to acquire this lock but whereas if one transaction is given the write lock, the other transaction cannot be given an extra write lock on the same data item unless the work was finished by the first transaction.

So in other words as opposed to looking at the operating system context where we have only one lock, we actually distinguish the semantics of the lock here whether it is read lock or whether it is write lock, based on the database systems tries to optimize the concurrency that is possible when transactions are simultaneously executing, to also understand the granularity of the lock. This is also important issue when you actually look at locking algorithms. The granularity actually means that how the size of the data items that are locked by the database by the transaction.

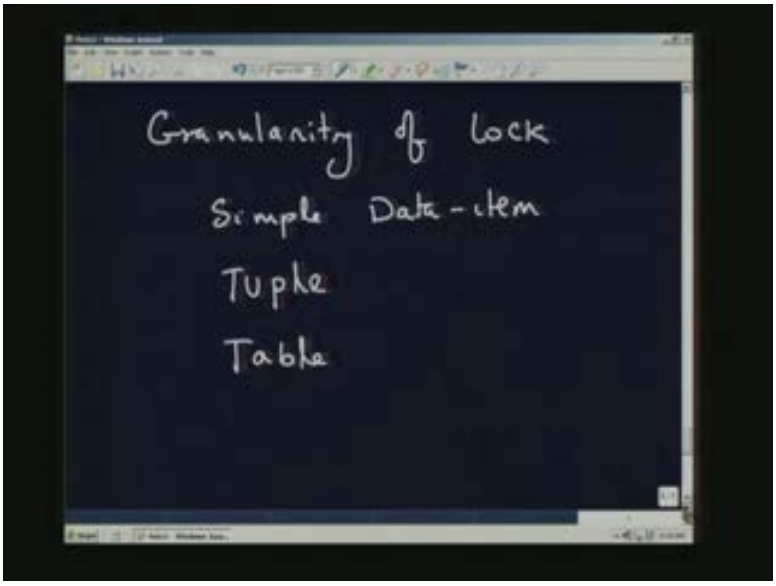
Now it is possible that the transaction only locks a simple data item which actually means that in a database table one single data item, for example in student in the case of a student record, it is possible that we are locking only a we are locking only a student name or we are locking only the students cgpa. We are not locking any of the other information of the student record. The other one could be the entire tuple, tuple means all the things relating to the students, one particular student getting locked that is called a tuple.

This is equivalent to also a row in a database. The entire row is being locked. The other one is the table which actually means that all student records are getting locked. That means that the entire table is getting locked, this is very important to understand the granularity. As you reduce granularity, the transactions only locks that small item but if you start increasing the granularity it starts locking a large number of items. So consequently as you increase a granularity of the lock, you correspondingly reduce the concurrency that is available in the system for transactions to execute.

One simple example is for example if you provide for the entire IIT, access at the in gate where there is there is going to be a lock. There is going to be only one person entering into the entire into the campus at any given point of time and after he goes out the lock is given to the next person, it drastically reduces the number of people who can get into the campus at any given point of time. But on the other hand if the access is controlled at a room level where when they enter into the IIT, when they reach the particular room you want only one person to be entering into the room at aty given point of time and that is the point where the room is locked.

A particular room is locked then the amount of concurrency that is available in the system is extremely high but it depends on where the concurrency is to be provided but we suddenly should lock as small as a granularity item as possible, keeping the consistent criteria into account.

[Refer Slide Time: 19.01]



Now given the understanding of the lock, let us go on to see how a simple two phase locking protocol will be working in the database context. As the term two phase locking explains, there are two phases in these particular algorithm. The first phase is what we call as a growing phase of the transaction in which the transaction tries to acquire all the locks and there is a second phase which is called a shrinking phase where the transaction tries to release the locks.

So if you basically look at the execution in terms of time, let us say this is the time access. Now the transaction starts executing at some point T_1 and what it actually does is as it starts executing, let us say it starts to actually reach execution points where it needs some data items. For example let us take a simple transaction where it is accessing bank card database as shown in the earlier examples.

I will basically try to read the balance in a bank account. Now the first thing that will happen in this case is I try to acquire a read lock on the balance data item which could mean that I might actually lock the entire account of a particular person and that shows that I in effect reached a point where I have locked, I have asked for a lock. Now it is up to the database manager to see that if nobody else is actually has a lock on this at this point of time or if the other transactions own only the read blocks on this data item, I can be granted the lock request.

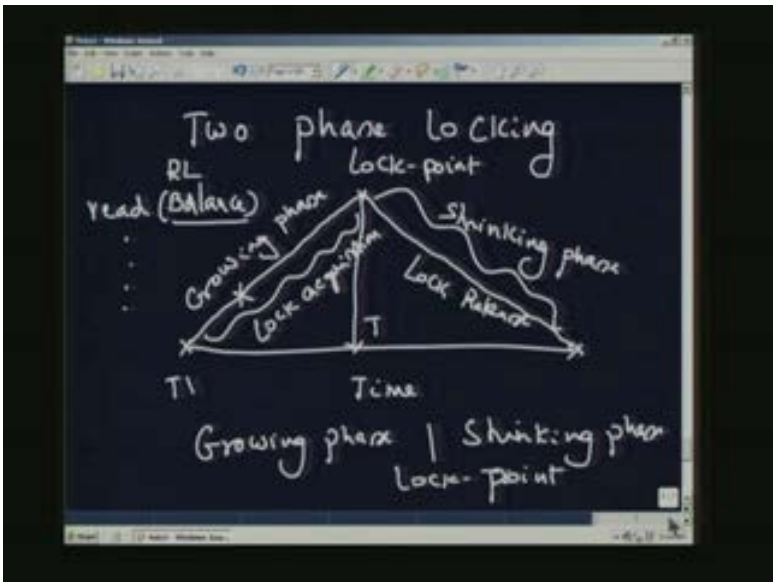
Now as I proceed like this, I might keep asking for a locks on other items as well. So this is basically the growing phase where the transaction is actually asking for locks. This is the first phase of the transaction execution which is called the growing phase. The growing phase, the transaction is trying to get all the locks it needs to do the work that it has to do. For example imagine that you have to move some items form one room to another item, another room then you first step before moving the items from one room to the other room or manipulating the items in a particular room, you try to acquire locks on all the rooms that is the first phase of the growing which is called the growing phase.

After you have got all the locks, you do the operation that you have supposed to do and after that you basically start releasing the locks. This is basically called the shrinking phase. In the shrinking phase, the transaction is actually releasing the locks. This is lock acquisition and this is lock release. In this side it is basically releasing the locks. Now the important condition of two phase locking is once a transaction is released one lock, it cannot ask for any more locks. This is a very important condition and a subtle condition that enforces serializability.

We are going to examine this little more deeply little later but right now what we understand is the condition between these two phases says once the transaction has reached this point which is called the lock point. This is called the lock point. When the lock point has been reached let us say at time T, the transaction is not going to ask for any more locks and it only releases the locks. This we will put it, so the first phase is the growing phase and the second phase is the shrinking phase. Now in terms of the two phases, the condition is you are not going to this lock point, you are not going to ask for any more locks after you have reached actually the lock point.

This is a very important condition that is enforced. Now let us examine how exactly the two phase protocol works for a transaction and understand deeply what are the consequences of applying a two phase locking protocol.

[Refer Slide Time: 23.58]

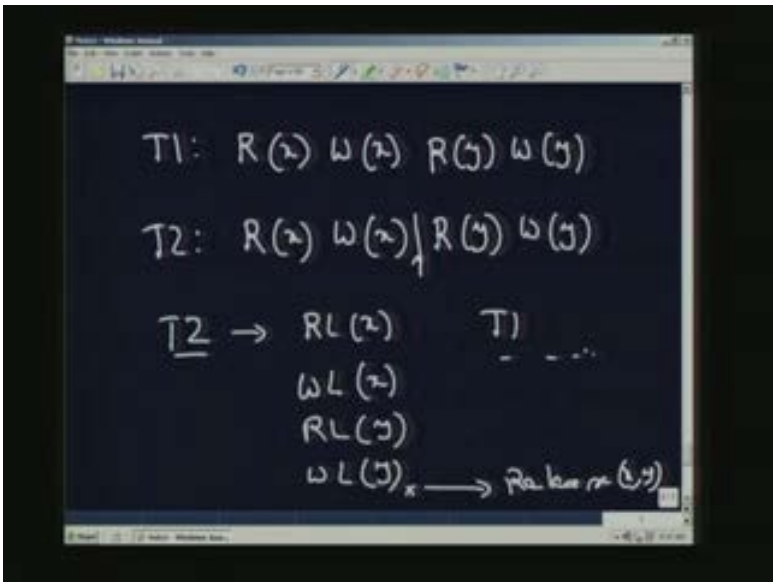


Now I will take a simple transaction and show what exactly would have happened, if I had actually applied a two phase locking algorithm. I will take a simple case of a transaction 1 which is actually reading an item R_x and actually writing an item data item x and then again it is basically trying to acquire or read an item y and then write an item y. There is basically another transaction T_2 which is also doing exactly the same work of reading a data item x, writing a data item x, reading a data item y and writing a data item y.

Now if you basically look at how two phase locking protocol would have worked in this particular case, when both these transactions at some point of time come into the system. Now let us assume that actually T_2 has actually come into the system at some point of time, the first thing it will start doing is as its starts, it requests for a read lock on x, read lock on x.

Now when the read lock on x is granted, it is going to ask for an up gradation of this read lock to a write lock. Now it still cannot release this lock on x, though it knows that at this point of time it is actually finished working with x. Now it is going to work with y but still it cannot release the lock on x because if it has released the lock on x, it cannot ask for the lock on y. This is the condition for two phase locking. If you have released one lock, you cannot ask for any more locks. So the transaction T_2 will hold the lock x and ask for now read lock on y and it will upgrade it to the write lock on y and it this point of time, it will release all the locks.

[Refer Slide Time: 26.05]



Now only when it releases the locks on x and y, can the transaction T_1 start executing which actually means that the locking will ensure if T_2 has acquired a lock on x, it effectively prevents T_1 from acquiring the same lock on x and hence when there are actually two conflicting transactions on a data item, they will be serialized based on who has acquired the lock first. This is how exactly two phase locking protocol will work. Now the meaning of acquiring a lock is it subsequently prevents any other transactions from getting the lock on the data item as long as this transaction is using it.

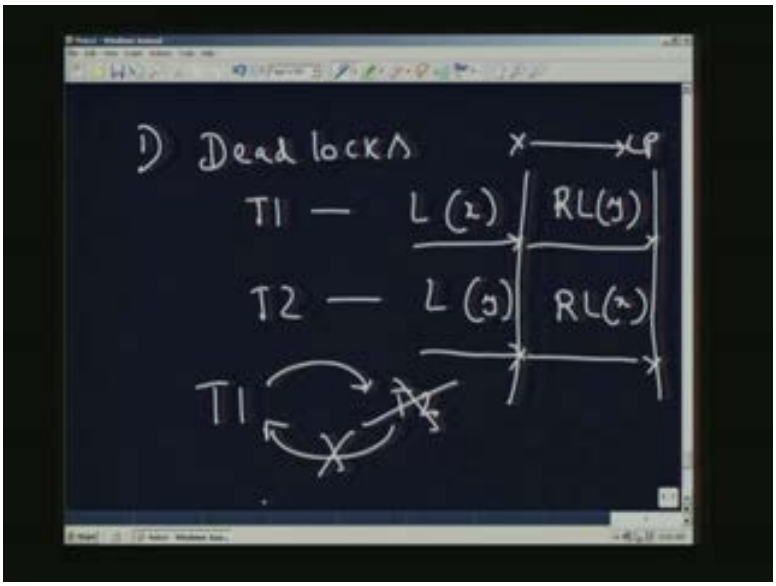
This is typically what is achieved as part of the execution, what you are achieving here is what is called conflict serializability. Here in fact making sure that the transactions execute when they are conflicting on data items in a serializable order. I am going to show a small proof to show that a two phase locking in effect produces serializable order of transactions. Before I do that, I am going to look at some more properties of two phase locking in terms what it can be doing. Since the first thing that is going to happen here is since transaction has to wait, if the transaction has been locked by some other transaction. If a data item is locked by some other transaction, it is possible that transactions could be waiting for each other.

Now this could result in what we call as the problem of dead locks. This is one of the problems of two phase locking. What we mean by dead lock is let us say T_1 has actually acquired a lock on, T_1 has acquired a lock on data item x and T_2 has actually acquired now a lock on data item y. It is possible that now requesting a lock on y and this is requesting a lock on x. Both cannot proceed any further because they have reached this point but they are not going to release this x or y till they reach the point of lock point where this is the lock point. So both will, they are at this point right now and there is no way both these transactions can reach their point but unless they reach the lock point there is no way they are going to release these locks.

So it is possible in which case T_1 is actually waiting for T_2 to release a lock and T_2 is actually waiting for T_1 to release a lock and both will keep on waiting for each other because they are in a continuous loop here. This in effect means that there is a dead lock because neither of them will be able to proceed any further and they will be waiting for each other in this particular context. This is what we mean by dead lock. This is one of the problems of using a pessimistic kind of an algorithm because the algorithms which make the which make the system wait for each other, will transactions to wait for each other can result in this dead locks.

The dead locks will basically bring bring the system performance and the throughput of the system drastically which means that the system time, the system throughput, the number of transaction that are executed by the system can drastically get affected when there is a dead lock condition. Problems of deadlocks are they need to be detected when they happen and the system has to recover back from this deadlock. In this particular case either of this transactions have to be aborted when a deadlock occurs and make sure they release their locks so that the other transaction can proceed. For example to break the dead lock, one of the transactions has to be aborted to make sure that T_1 can proceed. So deadlocks needs detection and subsequently resolving this deadlock require that you abort one of the transactions that is involved in the deadlocks so that the system can proceed further.

[Refer Slide Time: 30.06]



This is one of the consequences of two phase locking algorithm. Two phase locking algorithm also doesn't produce optimal schedules and we are going to look at the problem of what is an optimal schedule at a later point of time because the reason for this is it basically ensures, two phase locking ensures that a transaction always proceeds after it gets acquires locks to the finish.

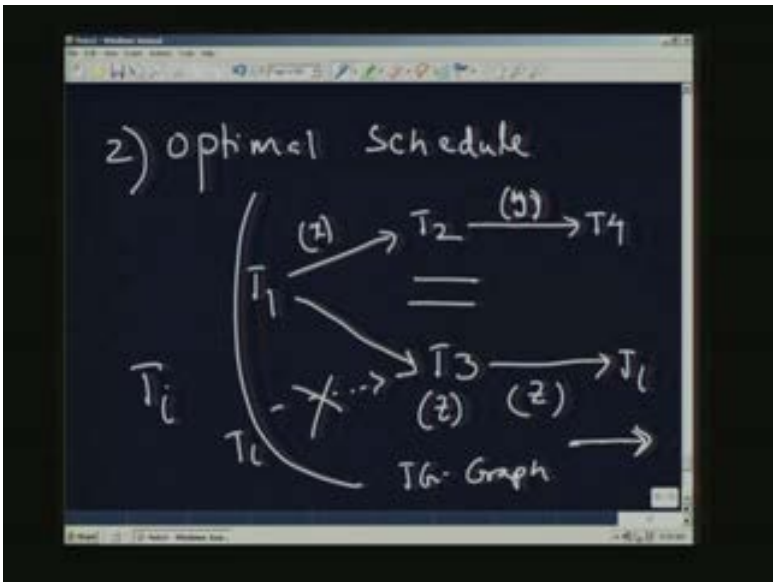
It never grants a lock and later actually aborts the transaction at a later point of time because there is a conflict. Whereas in the case of optimistic algorithms it is the other way. They let the algorithm, they let the transactions proceed to execute up till some point of time and resolve the deadlocks by resolve the conflicts at a later point of time by looking at what they have operated upon and seeing at a later point of time, if the operate on a data items in a conflicting fashion, they will get aborted at a later point of time.

So in that sense optimal schedules may not be possible in the case of two phase locking. We are also going to see a small example and show how time stamping produces optimal schedules compared to two phase locking algorithm. I explain this concept in the slightly different way by taking what we are actually done in the earlier class of looking at a serialization graph. For example if you look at as the transactions T_1 comes into the system, in effect you actually creating this graph which shows how the transactions have executed in the system one before the other. For example in this case let us say there are three transactions and then after that basically one more transaction has been executing.

Now we are basically, what I am doing here is I am saying that T_1 executed before T_2, T_2 executed before T_4. This is required only when there is a conflict. For example they conflict on a data item x, I need to know how exactly they have executed one before the other, otherwise it doesn't make. In this particular case T_2 and T_3 doesn't have any conflict directly and hence there is no need for me to actually put a arc here because they are not operating on common data items.

Now let us say there is an incoming transaction T_i into the system at this point of time. Now what the two phase locking in effect does is if there is a lock, for example let us say there is a z item on which T_3 is actually locked in. Now the only way T_i can come into the system is only after T_3, it can never be. For example if it is conflicting on this z data item with respect to any of the existing transactions, the only way you can allow T i to execute is by being after T_3. There is no way this transaction which arrived now after T_3 has locked to come before. This is prevented which means that the transaction graph, the TG graph here can grow only in the forward direction that is the only way the two phase locking allows the graph to grow.

[Refer Slide Time: 34.00]

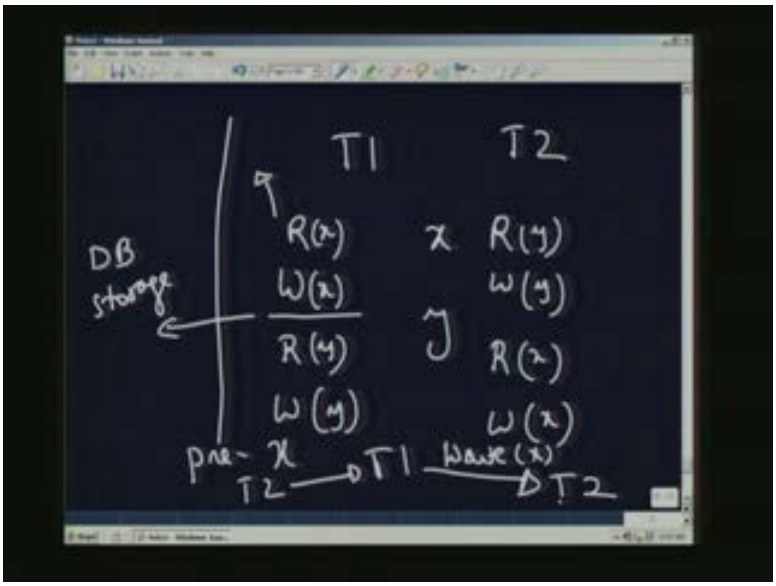


It doesn't let the graph to grow in any other direction but let us understand this problem of this graph growing in other directions as well. This is very interesting for various reasons because that allows a better schedules optimal schedules to be created when you are actually executing the transactions. Now to explain this, I will take a simple example of two transactions T_1 and T_2 executing on two data items x and y.

Now it is possible that I actually allow the read and the write things to proceed in a slightly different way and ensure that as the transactions are operating on this, either they read a pre copy or a later version which is modified by T_1. Now as we saw earlier there is a read x and write x that is happening on x and there is a read y and write y happening on the... Now let us say T_2 exactly does the other way of reading y, writing y and then reading x and then writing x. Now it's possible that when the transaction is actually modified the value of x, you can allow still this transaction to read a pre value which actually means that it is still has not written the value of its modification on to the database. This is the database storage.

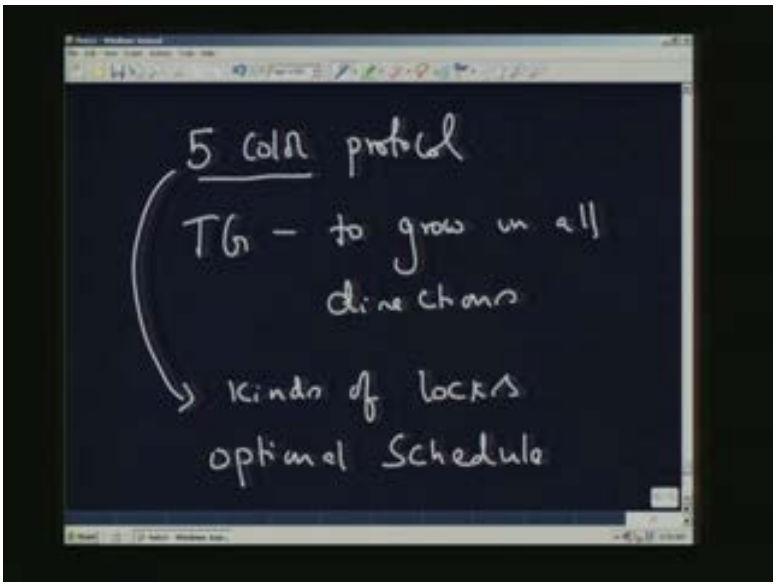
Since it has not yet modified the value on the database storage, it is possible to allow the T_2 to read a value before modification which means that there are three possibilities depending on what you allow T_2 to do after T_1 has actually started executing. If you allow the before value that is pre x to be read then the transaction T_2 is actually coming before T_1 because it is actually reading a value that is actually modified or that is not modified by T_1. If you allow T_2 to read a modified value then in affect it is coming after that is this is this is write x, this is after written, T_1 has actually written the value on x you allow T to read the value.

[Refer Slide Time: 36.30]



Now two phase locking will not allow the transactions to read pre x which in effect means that it will prevent this from happening. More advanced algorithms allow the transactions to be placed anywhere in the transaction graph, even they can come before a particular transaction or after the transaction graph. In effect there is algorithm called the 5 color protocol which allows transaction graph to grow in all directions. These are actually 5 different, the 5 colors here denote 5 different kinds of locks. 5 kinds of locks that a transaction can acquire besides just acquiring a read lock, a write lock, the other kinds of lock with a transaction can be granted when it tries to acquire a read or write data items which were before or after. Depending upon the pre and post whether the transactions data items have been modified before or after, it is possible for the transaction to acquire the locks. And this allows the transaction graph to grow in all directions and this produces sort of, it produces more optimal schedules than what we see in the case of two phase locking.

[Refer Slide Time: 38.12]

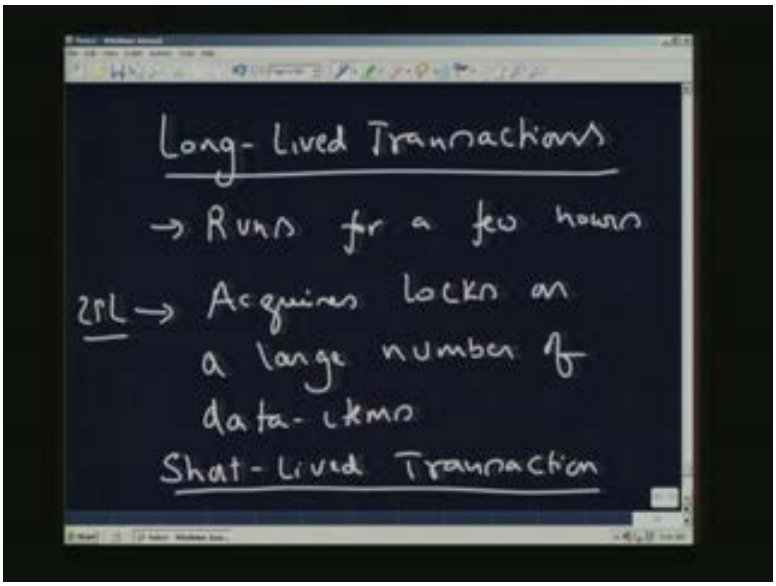


The other interesting problem that you see if you actually look at two phase locking is presence of long live transactions which actually means that the transactions are executing for a longer time. For example, the example for a long live transaction is a transaction that is trying to compute annual interest for every account in the bank. What this does is every year ending time, the savings is taken and the rate, interest need to be paid for each account is calculated. Now what this long live transaction, it runs for a very long time.

Normal transactions run only for a few milliseconds, this runs for a few hours and then it tries to acquire locks on almost all the data items, acquires locks on a large number of data items. Now, what this means is this is property one and this is property two, large number of data items. This in effect introduces lot of problems because this blocks a large number of short transactions, what we actually have is in case short live transactions. Now in the presence of long live transactions, short transactions will have problems of execution because they don't be able to run.

Their response time is going to get drastically effected, when there are long live transactions, if you apply a two phase locking algorithm because the long live transaction will in effect lock all the data items and will not release the locks on other data items till it finishes because that is one of the conditions of the two phase locking.

[Refer Slide Time: 40.09]



So if you apply a two phase locking, for long live transactions which are also there along with short live transactions, the performance of the short live transactions will drastically get effected, they won't be able to execute. The response time is going to be really bad when you actually apply two phase locking. This is another very important issue when you actually look at two phase locking algorithm.

There are lot of protocols which modify the two phase locking. There is not strictly two phase locking, they modify the two phase locking condition to actually allow long live transactions to execute along with short live transactions and there is a whole large number of protocols available for making long live transactions execute along with short live transactions.

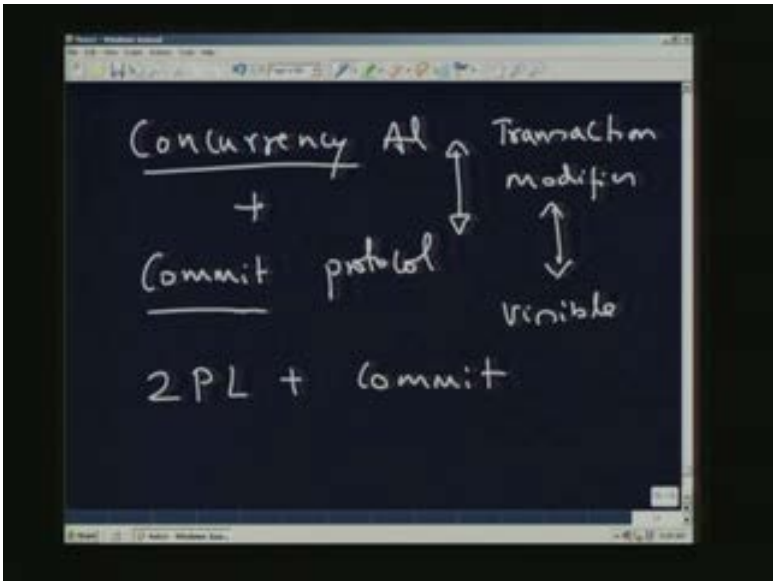
We will not go into details of that but then there is i will I am going to give references at the end of it pointing out a papers which actually give this algorithms, a host of this algorithms. Now what I am going to now touch up on is the other aspect of how a concurrency control algorithm has to be integrated with a commit protocol. Now what we understand by commit protocol and concurrency control algorithm has to be made little more clear. Now as you can see here, it is only after the transactions commits. If you remember the way we actually explained earlier, there is the transaction begins execution by actually saying begin transaction.

When it actually comes to the end of the transaction, you are actually that is the time when you are writing all the values that the transaction is modified back onto the database. It is still that point the values modified by the transaction are not actually written onto the database. That is what we mean by the commit protocol. There is inter relationship between concurrency and commit, in the sense that as the transaction modifies the values, this is this the place where transaction is actually modifying the values, transaction modifies.

And this is the point only after commit point it is actually visible, the modifications are visible only after this point. So there is going to be some kind of an interaction between concurrency protocols and the commit protocols and what we need to understand is how exactly the concurrency control algorithms get integrated with commit protocols because both together can only provide correctness. And if you basically see that the transaction has it modified it is visible then it is going to create difficulties in terms of the other transactions reading the values which are not yet committed by the transaction on the database.

So both concurrency and commit protocols have to work together. And we are going to explain in the next few minutes, how exactly the commit protocols works along with concurrency control algorithms. What I am going to look at it is how the 2 PL algorithm gets integrated with a commit protocol.

[Refer Slide Time: 43.42]



We are going to look at several ways, this can be done with two phase. I will take this simple graph that we have actually taken earlier to see how the commit protocol gets integrated with this. This is a familiar thing that we have actually, now the T_1 is the start of the execution of the transaction. This is basically the end of the transaction and this is the lock point of the transaction. Now what we are actually looking at here is, it is possible once the transaction is reached the commit point, it is possible as it is releasing the locks they are immediately made available for some other transaction.

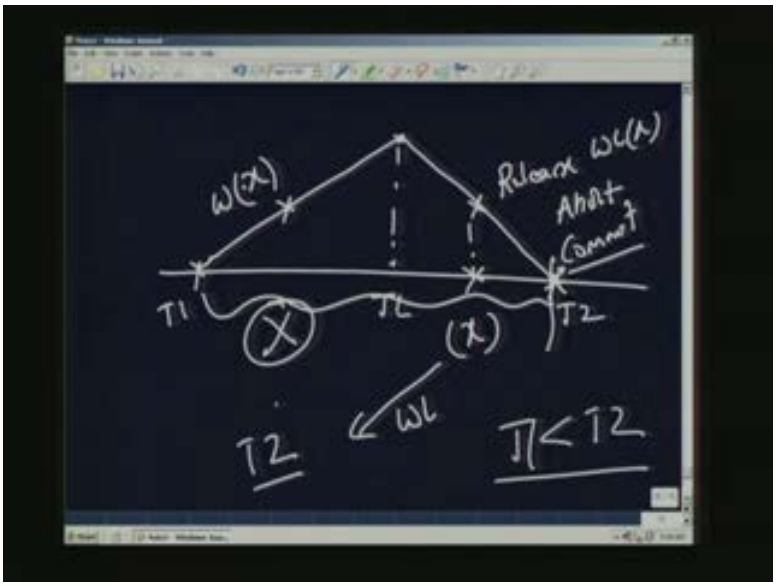
I will take only a simple case of data items x on which the transaction has a write lock. Now let us say it actually finished and releases the lock release write lock on x which actually means that it is possible that this x is available for some other transaction T_2 now to actually start working.

Now when T_2 actually is trying to now acquire lock on this same data item x, remember that this transaction has till not reached the commit because the commit point is here. The transaction is actually committing itself here not before this. So it actually means that we are allowing the transactions to be executed transactions to release locks before they actually reached the commit points.

This is where the interface between the locking protocols, that is the concurrency control protocols and the commit protocols come into picture because the commit protocols start operating at this point whereas the before that we have applied the concurrency control protocol. Now if this protocol releases the lock then the data item is visible for other transactions because they effectively can acquire the lock. But the value that they are going to read is not the value that actually is produced by this transaction because it is still not committed.

But if it is reading the value produced by, but our understanding is since it is released the lock any transaction T_2 acquiring the lock is after and hence there is a relationship between these two that T_1 executed before T_2 and hence this is to be preserved because the value T_2 should read is now the value modified by T_1. Now what is going to happen in this case is if the lock has been released by a transaction before it is actually committed, it's possible at a later point of time at the commit stage, the commit protocol issues an abort which actually means that this transaction T_1 effectively has actually aborted.

[Refer Slide Time: 46.33]



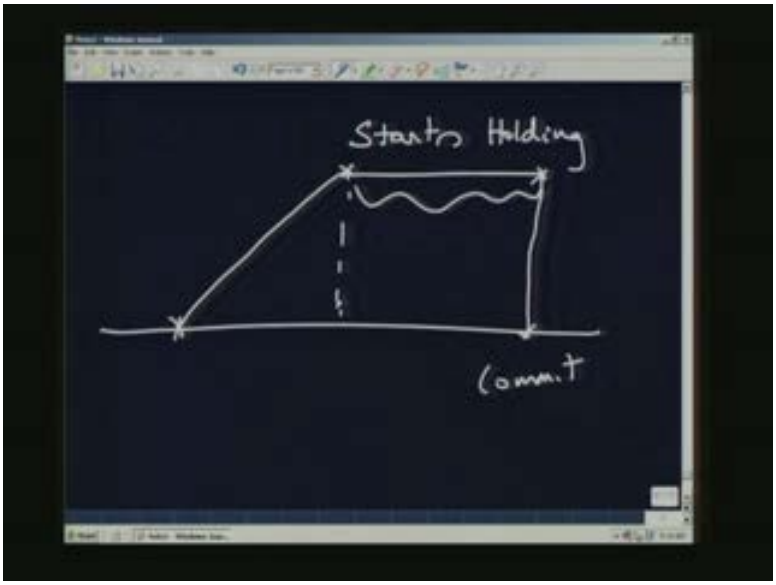
Now this requires that T_2 is also actually aborted. This is what we mean by if you if you allow the transactions to release the locks before you basically result in cascading aborts. T_1 actually modified a value on x, T_2 has actually read this modified value but now T_1 actually aborted for various reasons. Now T_2 should also abort and this is what really will

happen if you apply the transactions to release the locks before they are actually committed.

Now we are modification to the two phase locking algorithms taking this into the account to avoid cascading aborts will require that the transactions actually start acquiring the locks. This is actually the lock point but actually none of them effectively will release their locks till they reach the commit point which actually means that the key point holding on to the locks till they reach the commit point and all the locks are released only after the commit actually happens, which actually means that the transactions commit that means they write their values whatever values they have actually got, they will write these values back on to the system and then they allow or they release all their logs.

The logs are not released before committing, this basically shows that the transaction effectively starts holding, starts holding the logs till it actually reach reaches the commit point. This ensures that the concurrency control and the commit protocols work correctly by actually integrating the concurrency control protocols with the commit protocols. Now what we are going to look at it is all the protocols will require some kind of a modifications when we req when we look at how they integrate with the concurrency control protocols.

[Refer Slide Time: 48.25]



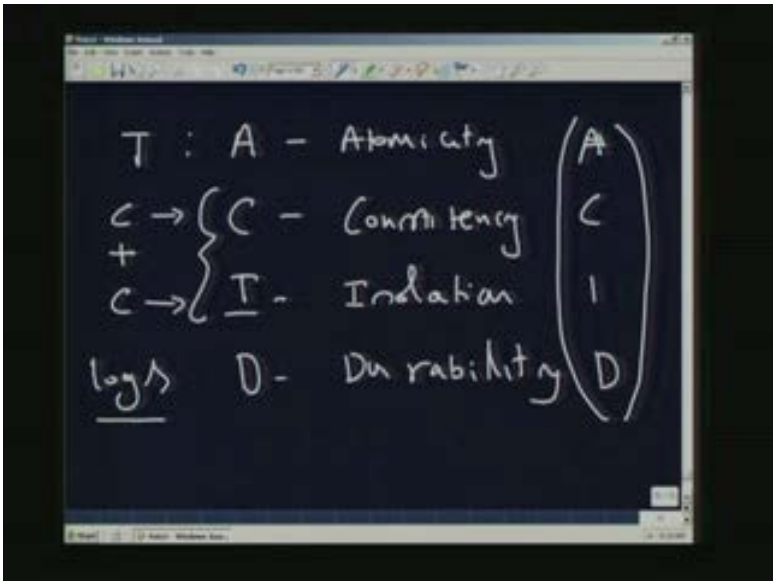
I will show you typically the three properties of, if you if you remember for a transaction, we are basically looking at three properties of atomicity which actually ensures that all or none of the actions of the transactions are written. Then we are basically looking at concurrency control. Typically this is consistency when they basically operating together, they basically produce consistent results then we are actually looking at the property of isolation.

Isolation means that one transaction results are not visible for other transaction till the transaction has committed then we are talking about durability. This durability is the transaction values are permanently written on the database. So this together what constitutes what we actually called as the acid properties of the transactions.

Now they together have to hold for every transaction. What we are looking at here is basically the concurrency control aspects. Now they have to get integrated with the isolation properties, this is where the commit protocols are coming into to picture. This is where the integration has to take place between concurrency control protocols and the commit protocols. When we discuss the time stamping algorithms also which operate in a slightly different way of actually ensuring that properties of serializability are actually enforced at the end of the transaction execution not before.

We have to see how that actually integrates with the commits protocols. It is going to be interesting to see how time stamping protocols ensure commit protocols integrate together in a proper way. When we discuss the time stamping algorithms, we are going to look at how commit and concurrency control protocols integrate with each other in that particular context. Typically the atomicity properties and durability properties are achieved by what we say as recoverability properties which are actually ensured using the logs. Typically the logs are maintained to make sure at any point of time, the transaction can redo or undo its actions and that is achieved using the logs.

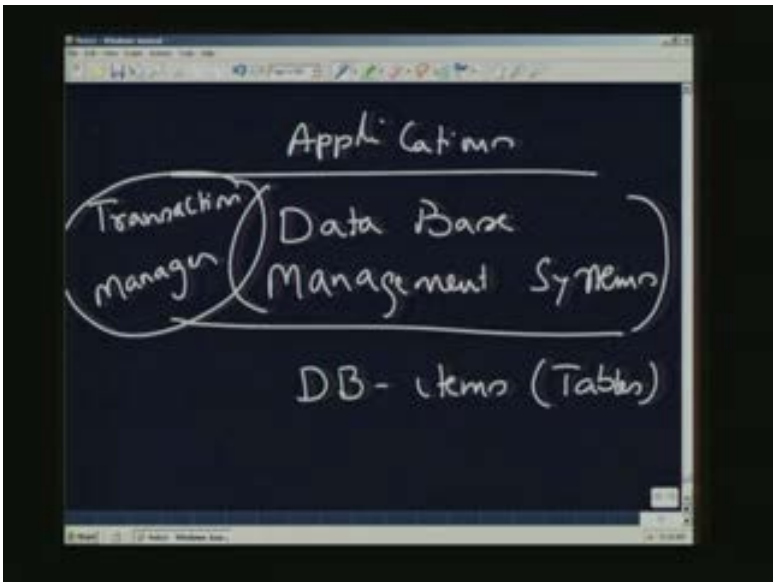
[Refer Slide Time: 51.24]



Logs plus the concurrency and commit protocols together ensure that the transaction acid properties are realized. And to just give you complete picture, what we are basically looking at the lowest tier is the database items. These are nothing but the tables that are stored in the database. Now at the other end, this basically the applications which are trying to modify the database items.

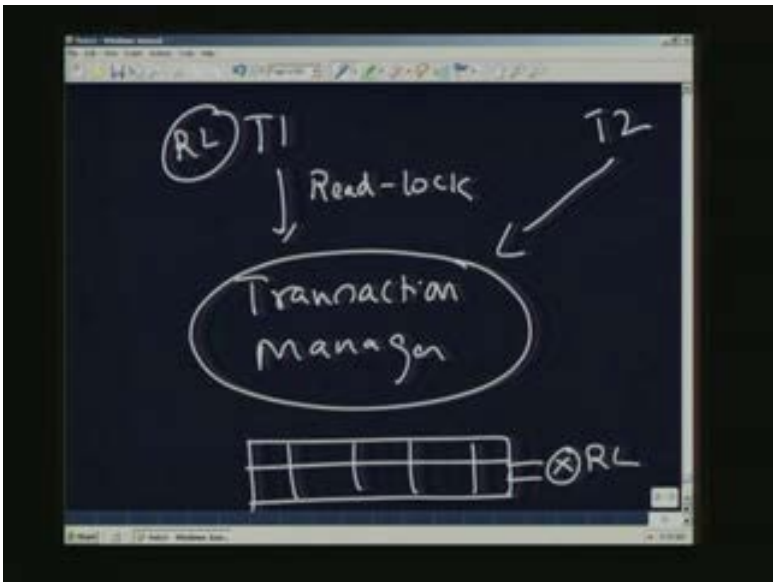
Now all the properties algorithms that we are talking about now come in the tier which is which sits between the applications and the database tables and this is what we mean by the database management system. Now in this case the dbms has various other things, this has to ensure among other things. A part of the subsystem has to deal with the transactions and that is basically what we mean by the transaction manager. Now the transaction manager is part of the dbms and this transaction manager is the one which actually ensures that the as the applications are executing, they serializablity condition is actually enforced using the transaction manager. Now to give an idea of what exactly happens when a transaction T_1 starts executing, when it is actually putting a lock request, let us say it is actually requesting a read lock, this is actually given to the transaction manager.

[Refer Slide Time: 52.44]



It is up to the transaction manager now to grant this request or disallow this request at this point of time. So, all the transactions in effect will make the request to the transaction manager. The transaction manager when it locks the data items for example there is a table here the student record table, now typically it allows a particular tuple to be locked then this lock will be granted based on the request that is basically requested by the transactions.

[Refer Slide Time: 53.43]



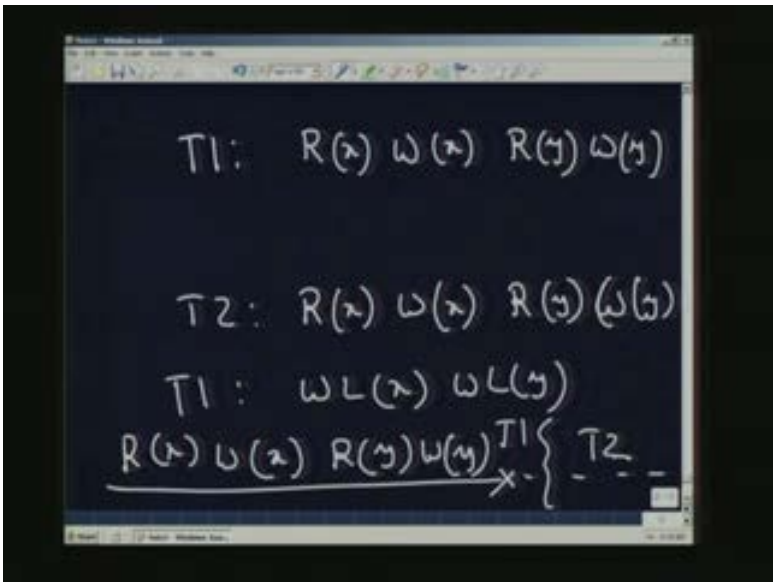
So all the lock requests are coordinated by the transaction manager and it knows which transaction holds what locks at the given point of time and ensures that the lock request are properly coordinated among the various transactions that are executing. It also ensures that along with the, for example if there is a commit protocol that needs to be operated before the lock is released, the transaction manager ensures that the transaction locks are not released till the commit point of the transaction is reached.

This is how exactly the concurrency control protocols work in the case of two phase locking. What we are going to do in the next few minutes is sum up and lead to the next set of algorithms which are typically time stamp based algorithms. Now what I am going to show you in this particular case is a simple execution of a transaction and show how two phase locking may not be a best way of executing the transactions and how one can think of in effect producing more optimal way of executing the transactions. This is the very simple example, I will with this example I will lead lead to the next set of protocols that I will be talking in the next lecture which are called the time stamp based protocols.

In effect looking at the execution of two transaction in T_1 and T_2, if you typically look at a read x, a write x and a read y and a write y by transaction x and T_1 and T_2 let us say just repeats the same kind of execution. You can in effect see that, it's possible for T_1 and T_2 to execute in different possible directions. Now one way the two phase locking ensures that T_1 executes after T_2 is both locks of T_1.

Let us say the write lock on x and write lock on y will be granted for T_1 which in affect prevents T_2 from start executing till T_1 has actually finished which means that the schedule that will be possible in this particular case is read x write x of one, read y write y of the other this is the point where the transaction would have committed T_1 and all the execution of T_2 proceeds after this point.

[Refer Slide Time: 56.31]



But it is possible for, if you carefully notice what is possible here is this is not the best possible execution of the transactions. It is possible for you to say once T_1 has actually finished working on data item one, data item x, it is possible at the point of time for T_2 to start executing on the data item x because T_1 no longer needs that data item on x but the whole set of problems will come if you let this happen.

First thing as we discussed, we will be sacrificing on the isolation property because you are letting transaction T_2 read the values before T_1 has actually committed. So this will sacrifice isolation property of the transitions. So how exactly if you want actually optimize the execution of the transactions, probably two phase locking is not the best possible way of executing but two phase locking is by far the best way of or a simple way of executing the transactions. I think I will stop here and going to come back in the next lecture.