Real-Time Systems

Prof. Dr. Rajib Mall

Department of Computer Science and Engineering

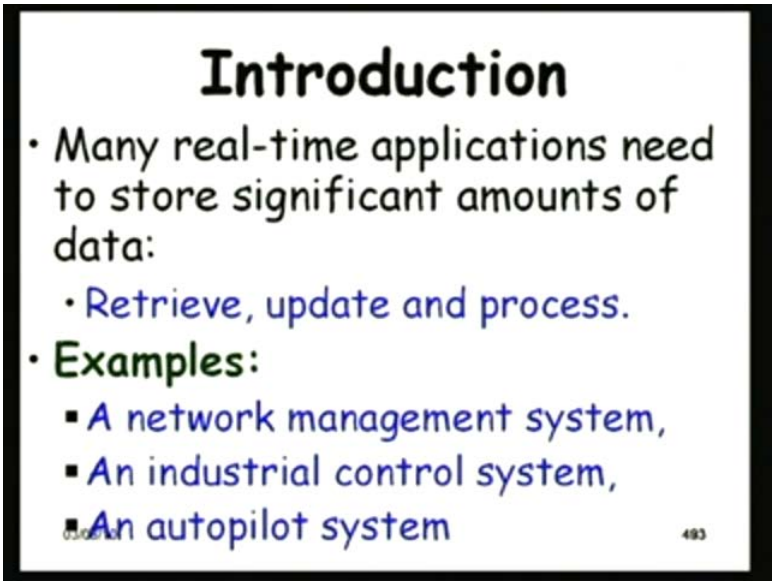
Indian Institute of Technology, Kharagpur

Lecture No. # 40

Real-Time Databases

Good morning. So today, we will start a new topic, which is Real-Time Databases and this is the last topic that we will be discussing.

(Refer Slide Time: 00:33)

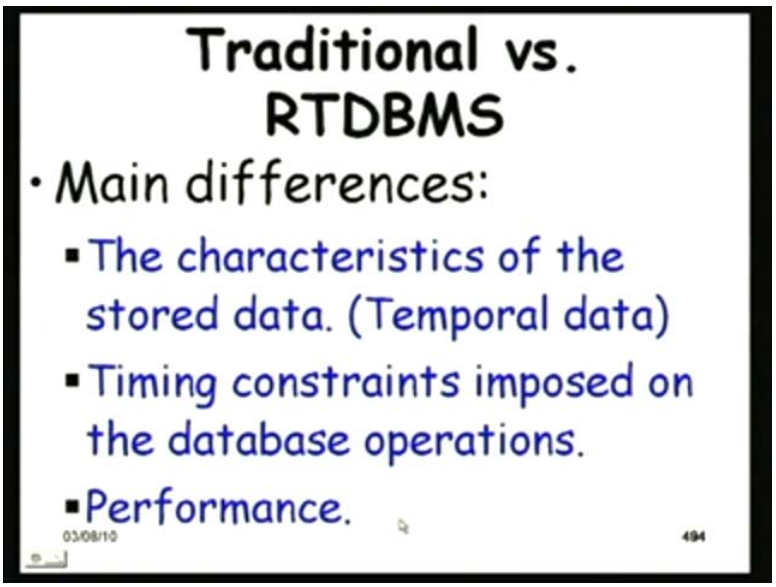


Many real-time applications actually need to store large amounts of data, and they need to process this data, they need to update this data and retrieve data for processing; just to give a few examples: a network management system, it stores data regarding the conditions of the network at any moment. For example, what is the traffic to congestion and so on, at any moment.

An industrial control system monitors various parameters of the industry that is being controlled may be temperature, pressure, may be chemical reaction rate and so on; and based on these parameters which are stored, it is processed and the system is controlled. Similar, is an autopilot system where various sensors collect data such as speed,

acceleration, height etcetera. And these data are stored in a real-time database and this data are processed to find out how the aircraft is to be controlled, how should the acceleration be increased, speed should be maintained and so on.

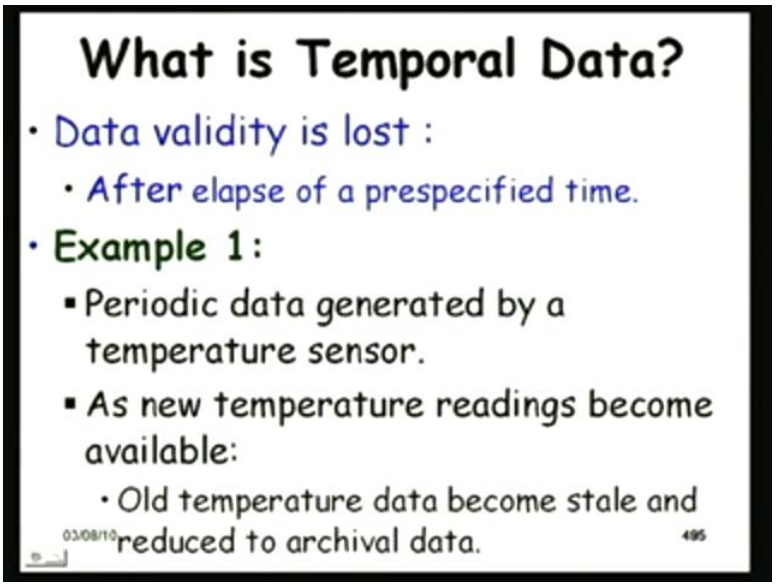
(Refer Slide Time: 02:23)



Now, having known that the real-time databases are required in many applications, many non trivial applications need to store data, and then, process it. Let us try to find out, how are these real-time databases different from the traditional databases. One of the major difference is with respect to the characteristics of the stored data, which we will call as the temporal data, where time is a dimension here, need to store about time, and data validity changes with time, which we did not have in the case of a traditional database; time was inconsequential to the validity of the data.

Another difference is that the transactions on the database operation will have some timing constraints associated with that. In a traditional database, the transactions do not have any such constraints and the performance of the database is crucial for successful operation of the system, and especially, the worst-case performance, which is of prime importance.

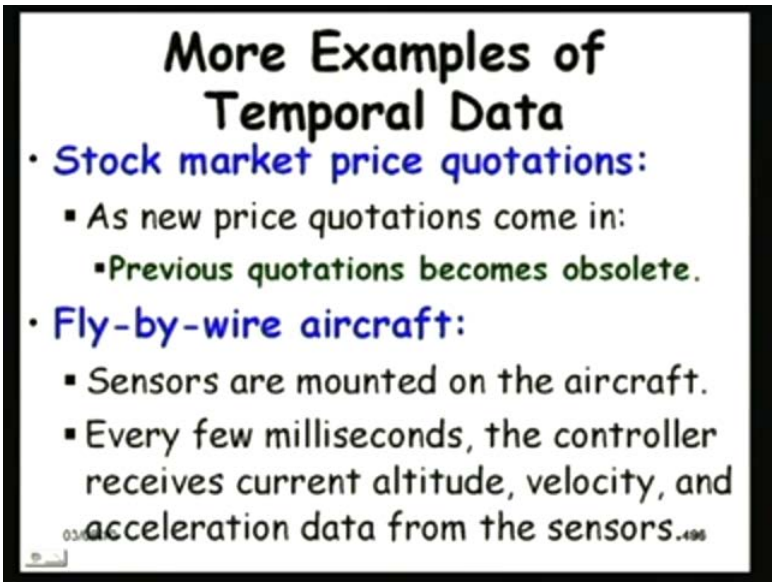
(Refer Slide Time: 03:48)



Now let us see, what is temporal data? Because we said that, the data characteristics are a major difference with respect to the traditional databases. A temporal database is one where the data validity is lost, after elapse of sometime; whereas, in a traditional data, the data continues to be valid for longtime. Now, let us try to understand this example, this definition of how the data validity is lost after elapse of sometime through some examples.

Let us consider periodic data generated by a temperature sensor. So here, as the newer and newer temperature values become available, the old values become archaic and they are not considered as the latest values. So, as new temperature readings available, the old temperature data become stale and they are reduced to only archival value.

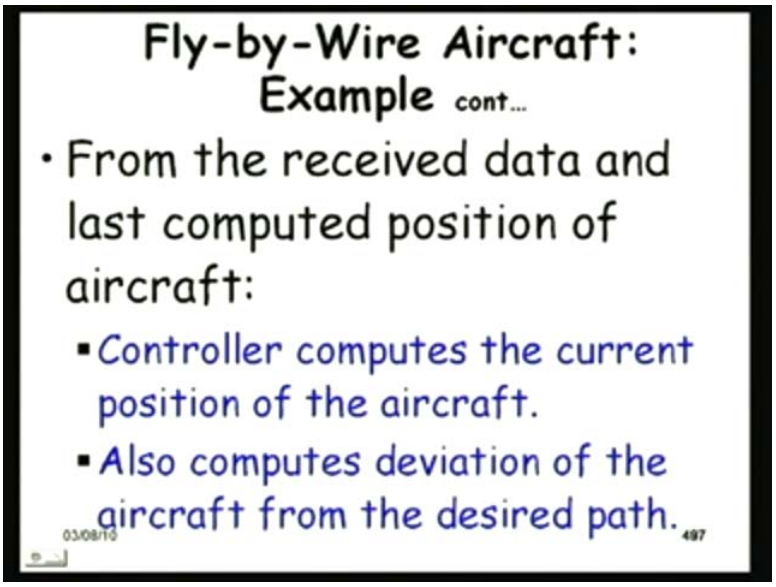
(Refer Slide Time: 05:11)



Some more examples of temporal data are the stock market price quotations and we will also explain this with respect to a fly-by-wire aircraft. Now, let us see the stock market price quotations as new stock prices, stock price quotations become available, the previous quotations become obsolete. So, you cannot basically trade based on the earlier value, you have to… the trading takes place at the newer value, but possibly the older value are still useful; for example, prediction, and validation, and so on.

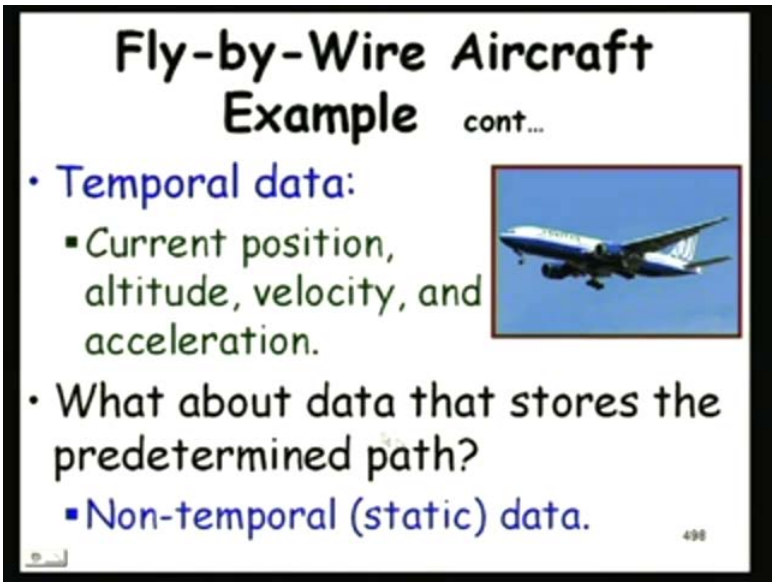
Now, let us look at the fly-by-wire aircraft example; in a fly-by-wire aircraft, the sensors are mounted on the aircraft, various types of sensors; for example, there may be sensor for finding velocity, acceleration and so on; and every few milliseconds, the controller receives data from this different sensor regarding altitude, velocity, acceleration and so on.

(Refer Slide Time: 06:32)



And from the received data and last computed position of the aircraft, the controller computes the current position of the aircraft, and then, by finding out the current position, it finds out how much it has deviated from the desired path, whether it has deviated at all, and if it has deviated, then correction needs to be done, and possibly that may be in terms of controlling some actuators which will do the corrective action.

(Refer Slide Time: 06:32)

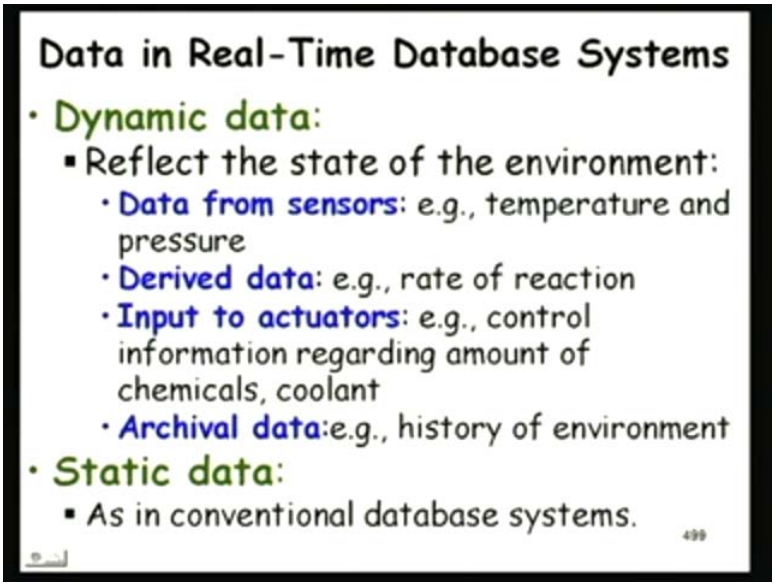


Now, what are the temporal data in the fly-by-wire aircraft example? The value that are read by the sensors, the position, altitude, velocity, acceleration - all are temporal data.

Some of them are primary data and some are, of course, derived data; for example, current position is a derived data; altitude, velocity and acceleration are primary data; but all of them are temporal data. So, this is the one fly-by-wire aircraft, the Boeing 777 which was the first, fly-by-wire aircraft from the Boeing company, which is operational now, a commercial aircraft, where once the autopilot mode is on, it takes over the running of the aircraft; of course, the pilot can override the actions of the automated system.

Now, let us consider another data that is stored in a fly-by-wire aircraft, which is the predetermined path or the required path. Now, what about this data? Will it be a temporal data? Not really; it is a static data; it does not change with time; as the aircraft moves, this data remains constant, data does not change, and therefore, the fly-by-wire aircraft has both temporal data, and the static data or non temporal data, and the computations will take place by using both temporal and the static data.

(Refer Slide Time: 09:26)

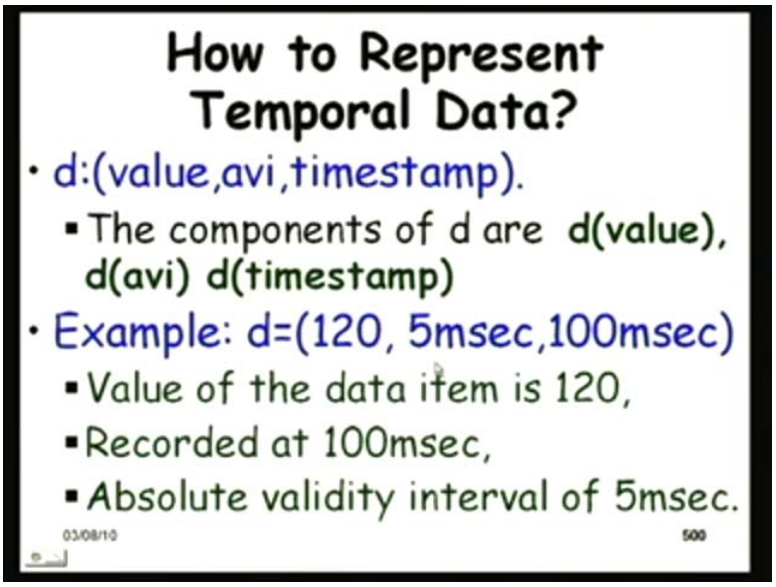


So, now we can think that, the data that is stored in a real-time database system are two types: the dynamic data and the static data. The dynamic data typically reflect the state of the environment. All, the examples that we considered, there were sensors which were continuously getting the monitoring some parameters of the environment and that was reflecting the state of the environment; for example, temperature, pressure and so on.

So, the data from the sensors are the primary data, which can be use to have some data derived from this primary data which will we call as the derived data; for example, rate of chemical reaction or may be the current position and so on. These are examples of derived data, but again these are dynamic data or temporal data. Similar is the value that is computed for input to the actuators; for example, the control information regarding the amount of chemical, coolant etcetera. These are also temporal data, because at different instance of time, the values might change.

But there will be another category of data, which is the archival data, which is the history of the environment. So, as the temporal data becomes absolute, these are reduced to archival data which are how the environment of the system has evolved over time, and then, we have static data which are stored as in a conventional database, these do not change.

(Refer Slide Time: 11:29)

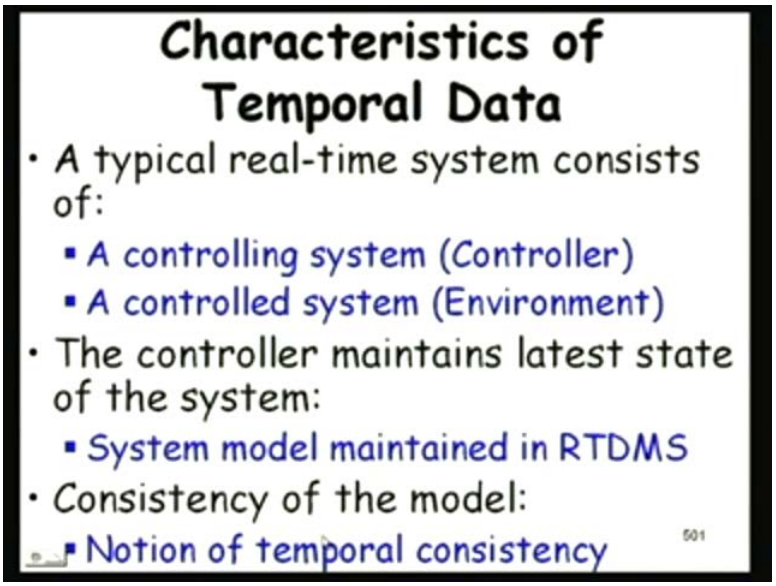


The next question: having known that real-time databases need to deal with temporal data, the question will be - how do I represent a temporal data? We know that this data have element of time associated with that, but then how do you represent the data? The way it is represented is through three components: a data item d is represented through the data value; the absolute validity interval, that is, for a how long the data will remain valid; and then the timestamp on when the data was collected. So, these are the three components, using which temporal data will be represented, and if we want to refer to

any specific component of d, for example, value, we will write d value, d of avi or d timestamp etcetera, this is the notation that is typically followed.

For example, if the temporal data is 120 meters or whatever meters per second or whatever, and then, the avi is 5 millisecond, and the timestamp is 100 millisecond, then the value of the data item is 120 and it the value was recorded at the time 100 millisecond after the system started, and then, the absolute validity interval is 5 millisecond; so, from 105 millisecond, this data will become archival data.

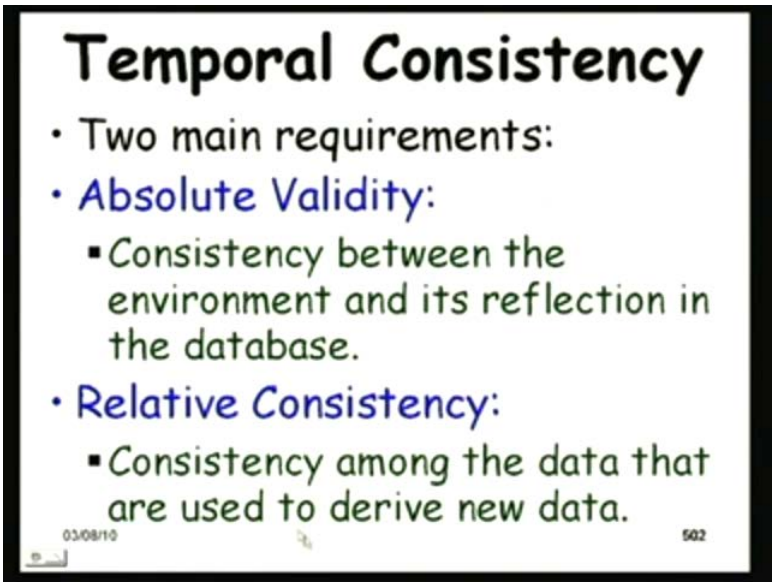
(Refer Slide Time: 13:18)



Now, let us see some characteristics of the temporal data. We have from the beginning been discussing about the structure of a real-time system. It consists of two important components: a controlling system, which is also called as a controller and a controlled system that is the system, which is controlled by the controller, is the environment; and the controller maintains the image of the environment or what is the latest state of the environment in the form of the data collected. So, this we will call as the system model that is maintained in a real-time database management system by the controller, and the controller analyzes this data and caries out some action through the actuators.

Now, what is the consistency model for this data? So, the consistency model, we will define it through the notion of temporal consistency. Now, let us see - what exactly is the temporal consistency, which will define the consistency of temporal data?

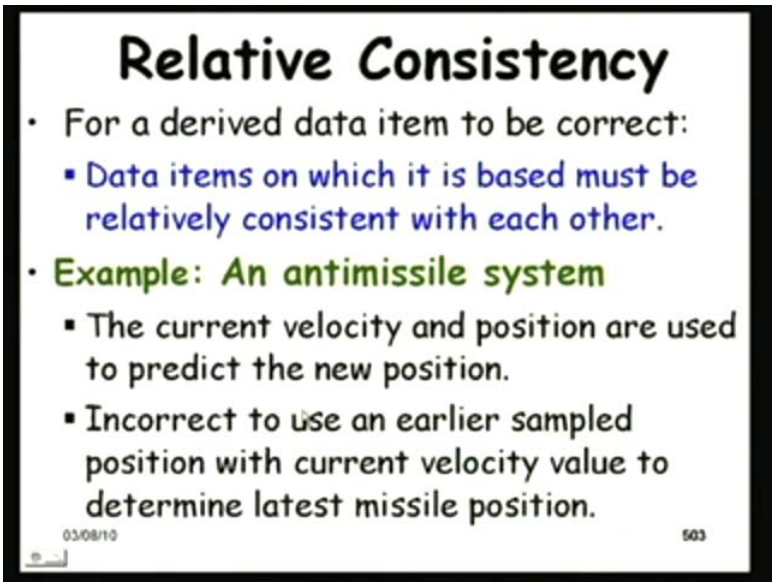
(Refer Slide Time: 15:01)



There are two main components of temporal consistency: one is called as the absolute validity and the other is called as relative validity. So, a data is consistent, if it is both absolutely valid and relatively consistent. Now let us see, what these two terms mean. The absolute validity means, consistency between the environment and its reflection in the database. So, what is the state of the environment at present and how much does this is different from the model that is maintained in the database.

So, as the data becomes, as the environment progresses, the data becomes invalid or archival. So, after the elapse of some time, as the environment evolves, the data becomes more and more deviated from the actual state of the environment, and we will call it as absolutely invalid, and that is the time for which a data remains absolutely valid is given by the avi component of a temporal data. There is another component here, the relative consistency, which is useful when trying to derive new data based on some primary data. So, the primary data that are used to derive a data have to be relatively consistent. Now, we will explain this with respect to an example.

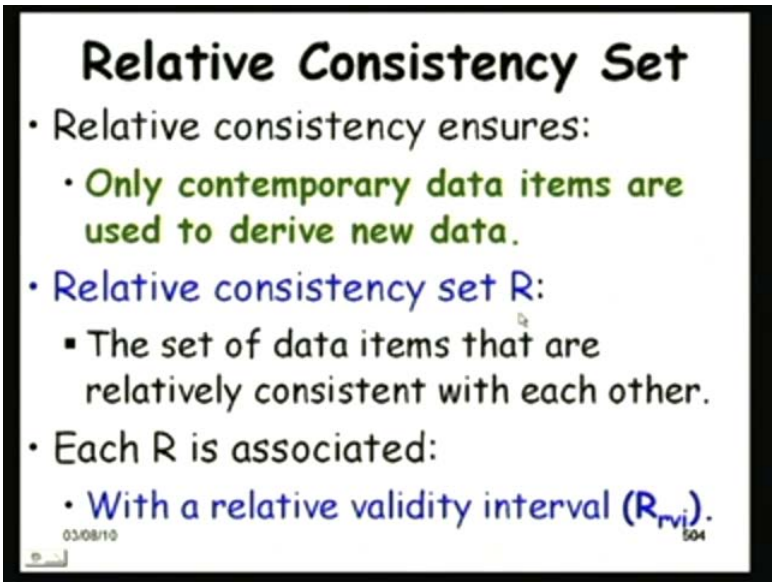
(Refer Slide Time: 16:47)



So, a set of data items are relatively consistent, if they are contemporary data. So, a derived data will be correct, only if the data items on which it is derived are relatively consistent with each other, or in other words, a set of data items would be called as relatively consistent, if they are contemporary data item. Now, let us try to see what we mean this, mean by this through an example.

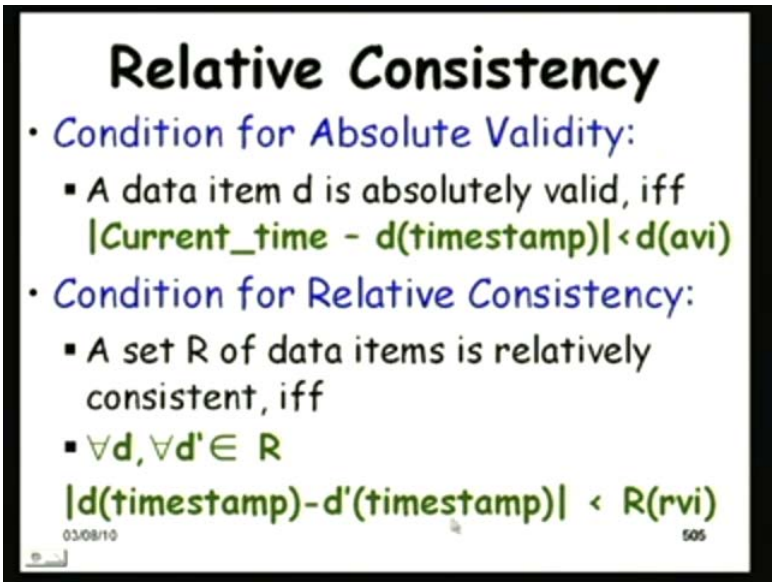
Let as consider an antimissile system. Now, let us say the current velocity and position of the missile is used to predict the new position of the missile. Now, there are two primary parameters: the current velocity and the position that are used to derive the new position. Now, if we incorrectly use an earlier sampled position, and use the latest velocity with a older position, then we are definitely going to get a wrong prediction of the position. So, the position and the velocity have to be contemporary or when the current position, at the current position what is the velocity, we have to consider that. So, that is the notion of a relative consistency.

(Refer Slide Time: 18:39)



The relative consistency requirement ensures that only contemporary data items are used to derive new data. So, this we were just mentioning that unless the data that have been used to derive a new data or tend to the same time period, then we will get a wrong result. So, the relative consistency set R, is the set of all data items that are relatively consistent with each other, or if we use notations to specify this, we will use that; we will we first define what is called as a relative validity interval or R rvi. So, every relative consistency, relatively consistent set will be associated with a R rvi. So, if R is a set of data items, with R we will associate another component which is the rvi or the relative validity interval.

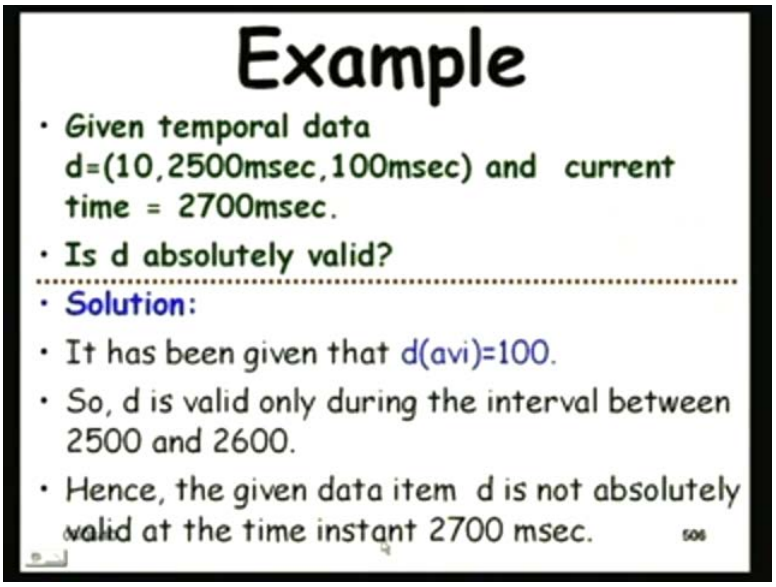
(Refer Slide Time: 19:54)



So, now we can specify this using mathematical notation. So, we can say that for absolute validity, the current time minus the timestamp of the data is less than d avi. So, if d avi is 5, and the timestamp is 100, and the current time is 104, then this is valid, but if the current time is 106 and timestamp is 100, then the data becomes absolutely invalid.

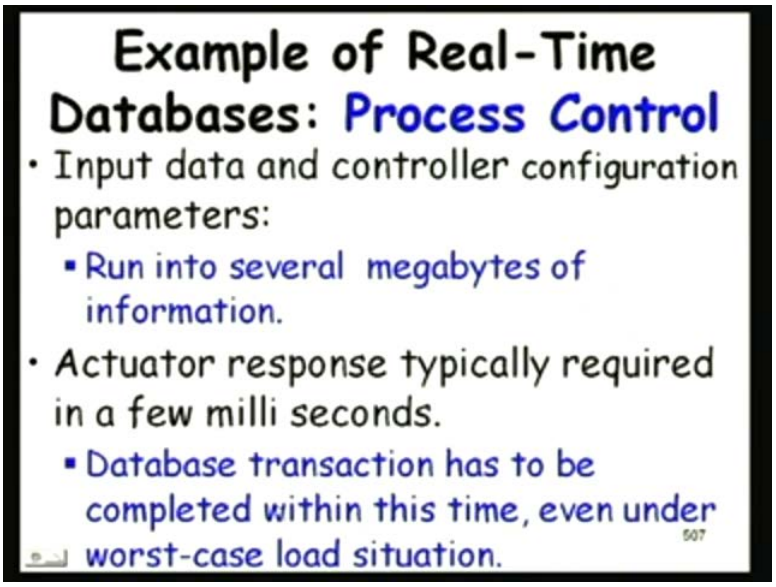
Now, let us see the condition for relative consistency. So, R is a set of data items, for any d, d prime belonging to R, if d timestamp minus d prime timestamp mod is less than R rvi, then we say that, it is relatively consistent. So, if there one is a old data, then the difference between the timestamp of this will be much more than rvi and they will not be relatively consistent.

(Refer Slide Time: 21:15)



Now, let us try to understand this through an example; suppose, we have a temporal data, 10 is the value and 2500 millisecond is the current time and 100 milliseconds is the absolute validity interval, 2500 is the recorded time and the current time is 2700 millisecond; is d absolutely valid? Now, let us try to find that out. So, it is given that d avi that is the absolute validity interval is 100 milliseconds. So, what it really means is that the data items are having value 10 is valid anywhere between 2500 to 2600. Now, as a result d is not absolutely valid at time instant 2700. It would become a archived data and it is absolutely invalid.

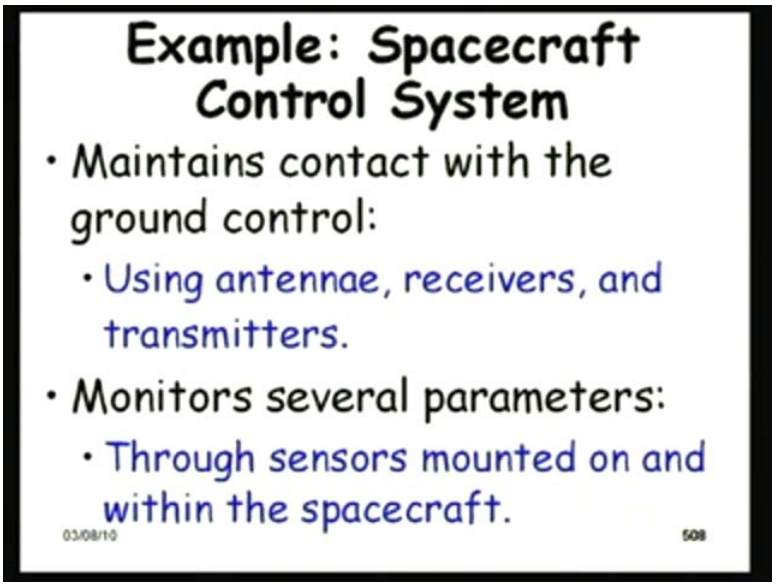
(Refer Slide Time: 22:32)



Now, let us look at some examples from the process control domain about the use of real-time databases. Here, the various data that are maintained, for example, the controller configuration, the sampled value and so on, they run into several megabytes or even sometimes gigabytes of information and based on processing this data, the response required is often in terms of only few milliseconds.

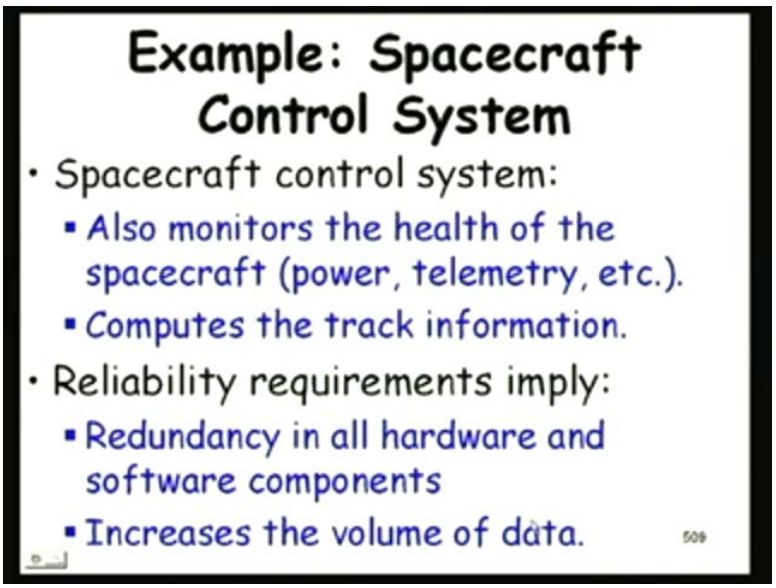
And, unless the database transaction completes within this time, then the system will fail; and not only that it has to complete within few milliseconds, but that must be ensured even under worst-case load situations, even when suddenly there are lot of activity in the database, we cannot afford to have the transaction delayed by the specified time; otherwise, it will lead to a system failure. So, then the question is that the transactions are time constants, how do we ensure that the transactions will complete within the required time.

(Refer Slide Time: 23:58)



Similar is the case with Spacecraft Control Systems, which maintains contact with the ground control using antenna receivers and transmitters, and then, it monitors several parameters through sensors mounted on the spacecraft and also within the spacecraft.

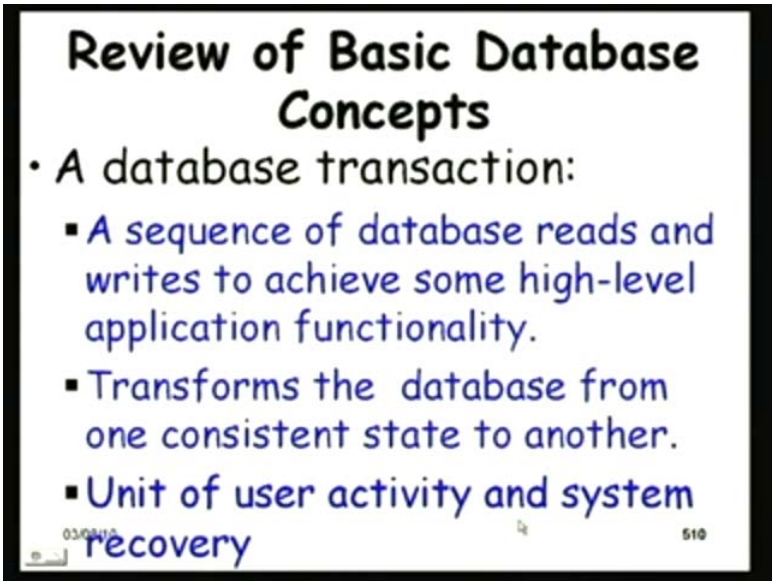
(Refer Slide Time: 23:58)



And the spacecraft control system, based on this data, also monitors the health of the spacecraft like power, telemetry and so on. And it also computes the track information that is - what is the track? It is following, how much is the deviation from the desired track.

And in such a situation, the reliability requirements on the spacecraft required that there has to be some redundancy with respect to the hardware and software components. Now, whenever, there is redundancy, the software and hardware component, there has to be an increase in the volume of data that needs to be handled by the system. So, not only that, this deal with large data by themselves, but also the redundancy requirement increases the volume of the data further, and even when we have such large data, the response time for the transactions has to be ensured.

(Refer Slide Time: 25:27)

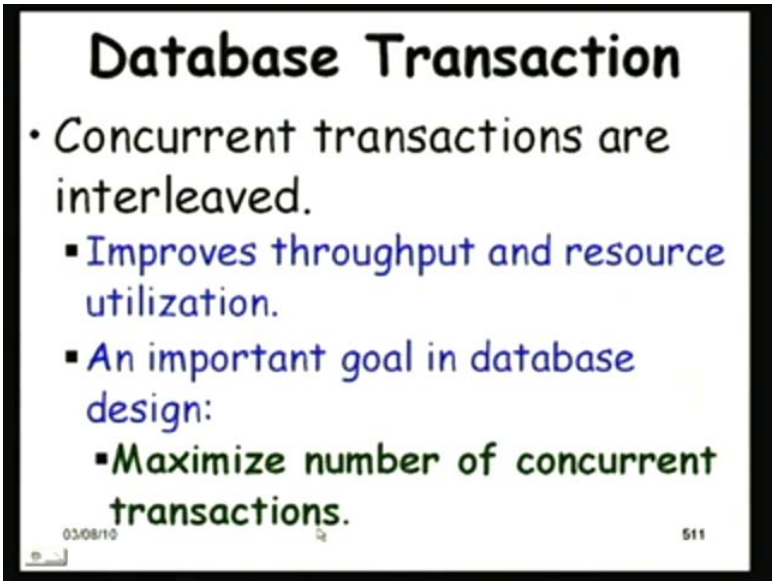


Before, we examine how the response time for the transaction is to be ensured, first let us review a few basic database concepts. A transaction is a very fundamental concept on a database. How do we define a database transaction? A transaction can be defined as a sequence of reads and writes on the database to achieve some high-level application functionality. For example, if you are considering a banking database, which is a traditional database, may be withdrawal of cash request is one application functionality and a spot of this application functionality several reads and writes to the database may be required and this form the database transaction.

Another way to look at a database transaction is to consider it as transforming the database from one consistent state to another. So, when a transaction is incomplete, its half way complete, may be the database state is inconsistent, but when the transaction completes, it leaves the database in a consistent state. So, in other words you can say

that, a transaction transforms the database from one consistent state to another. Another way to look at a transaction is as a unit of user activity and system recovery. So, you just shown this as a user activity in terms of a transaction and also the system recovery takes place in terms of transactions. So, the system rolls back to the last completed transaction.

(Refer Slide Time: 27:32)

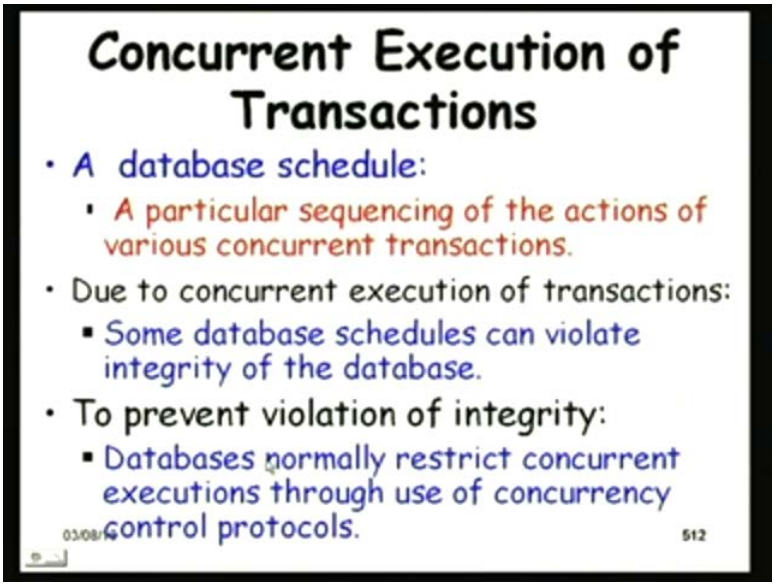


One thing that we must remember that the transactions are not sequential sequentially carried out on a database, but they are concurrent, there can be several transactions which occur on the database concurrently, which are essentially interleaved. So at any instant, any one transaction actually operates on the database, but the activities of the transactions read, write, etcetera are interleaved.So, what this really means is that, the read request for one transaction may be followed by write request of another transaction; write required by the some other transaction, read request with some other transaction and so on. This is required, because it improves the throughput and resource utilization.

A serial database, which does not exist commercially, would be the simplest to implement, where one transaction completes and another transaction starts to operate, but this would be too slow for carrying out any meaningful commercial exercise, and therefore, all commercial databases use concurrent transactions. A database with concurrent transactions improves throughput and also the resource utilization.

Now, given that in any database we have transactions that are active at the same time, an important goal we can define for a database designer is to maximize the number of concurrent transactions; why is that? Why should we maximize the number of concurrent transactions ? The idea is that if we maximize the number of concurrent transactions that the database can support, this will improve the throughput and also the resource utilization.

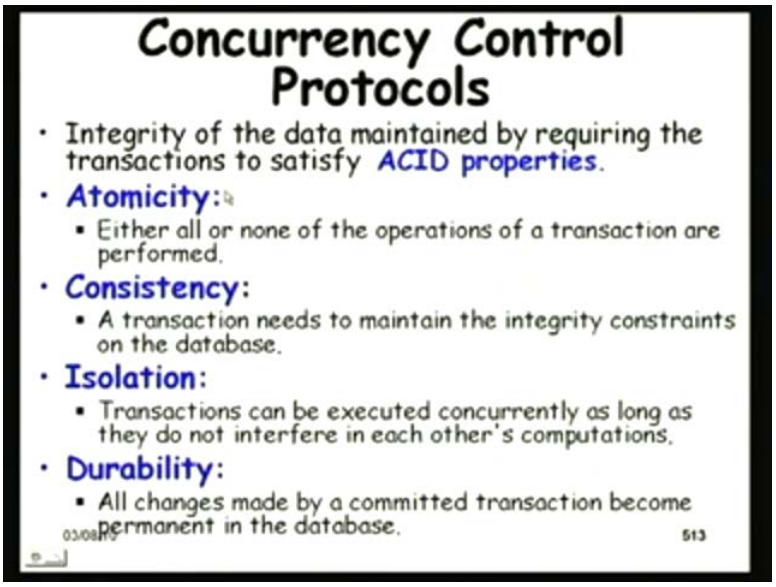
(Refer Slide Time: 29:40)



Now, when we have concurrent execution of transactions, we can define a database schedule as a particular sequencing of the actions of various concurrent transactions; actions are basically read, writes. So, a particular sequencing of the reads reads and writes of various concurrent transactions is called as a database schedule. Now, any arbitrary schedule would lead to problem, because many of the schedules can violate the integrity of the database.

So, these are very basic concepts in database; so, we are not spending time on this; not really explaining how any arbitrary schedule may violate the integrity of the database, and to prevent violation of the integrity, the databases typically restrict the concurrent executions through the use of some concurrency control protocols.

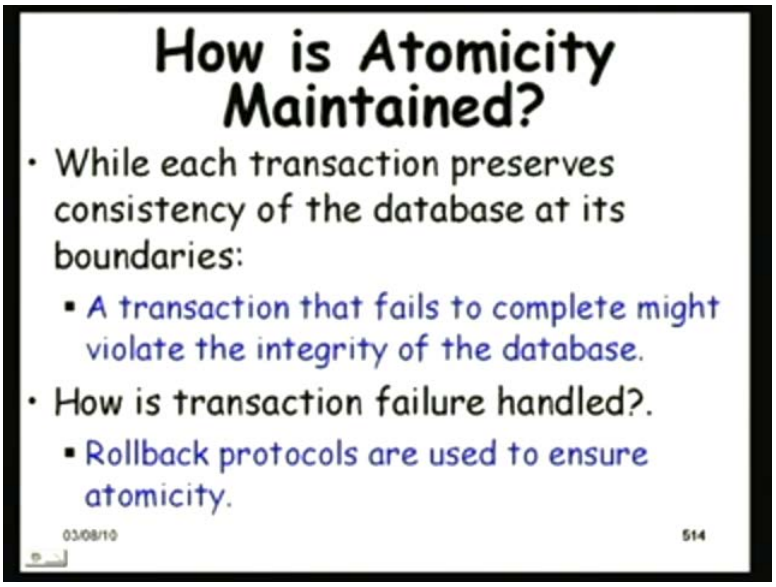
(Refer Slide Time: 30:47)



The concurrency control protocols, actually disallow certain interleaving and allow only certain interleaving which will satisfy, the what is called as the acid properties. The acid properties are atomicity where either all or none of the operations of the transaction are performed. Consistency - a transaction needs to maintain the integrity constraints on the database. Isolation - transaction can be executed concurrently as long as they do not interfere, in each other's computation. Durability - all changes made by a committed transaction become permanent on the database; so after the transaction completes, the effect of the transaction cannot be undo, it is the durability property.

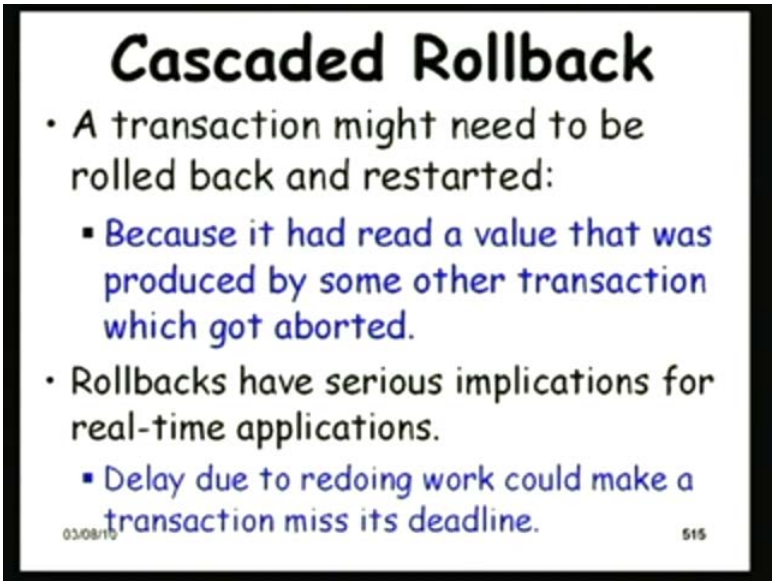
So, these are four essential properties that ensure the consistency of the database and is a very basic in a first level database course these would be discussed. So, we will not spend time on this.

(Refer Slide Time: 31:54)



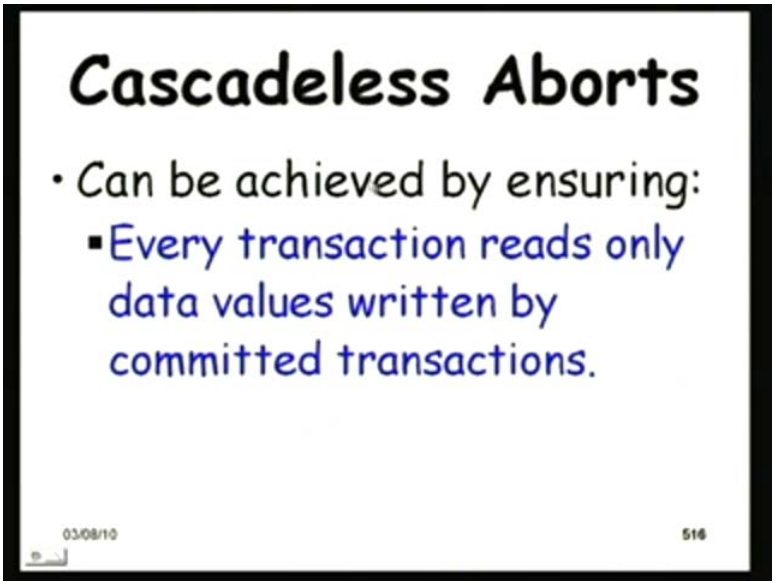
But what if a transaction started and it fails to complete. Then, an incomplete transaction can lead to an inconsistent database. So, whenever there is a transaction failure, there has to be some rollback protocols to ensure that the state of the database is rolled back to the previous consistent state. So, the rollback protocols are used to ensure atomicity of the transactions, and we do not allow a transaction to be partially complete. So, the atomicity ensures that the transactions complete as a whole or there is no effect of the transaction.

(Refer Slide Time: 32:46)



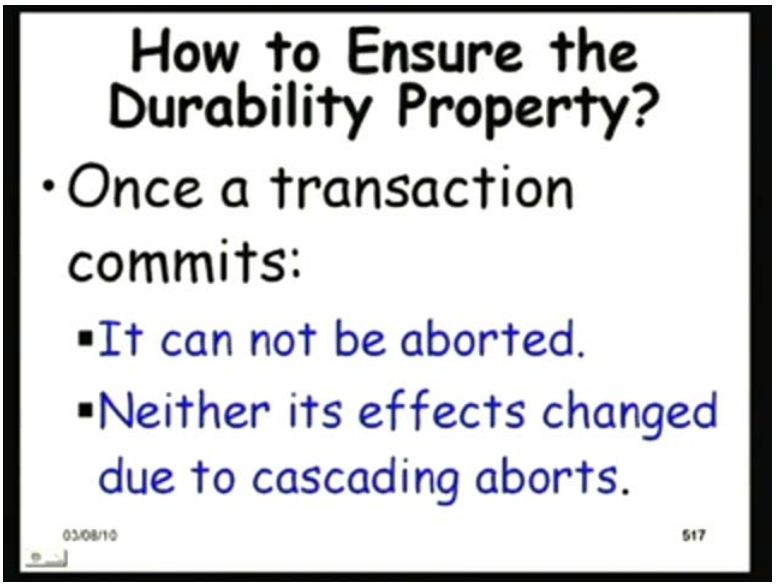
But one problem with the rollback is the cascaded rollback; when we rollback one transaction's effect, then several rollbacks might have to be carried out, because the read and write values produced by this transaction which got aborted might have been used by some other transactions. So, when we have cascaded rollbacks, lot of computations are wasted, and naturally, rollbacks have serious implications for real-time applications, and the delay they may introduce, may require redoing lot of work. The delay may arise due to redoing lot of work that were wasted and this can lead to a transaction missing its deadline.

(Refer Slide Time: 33:41)



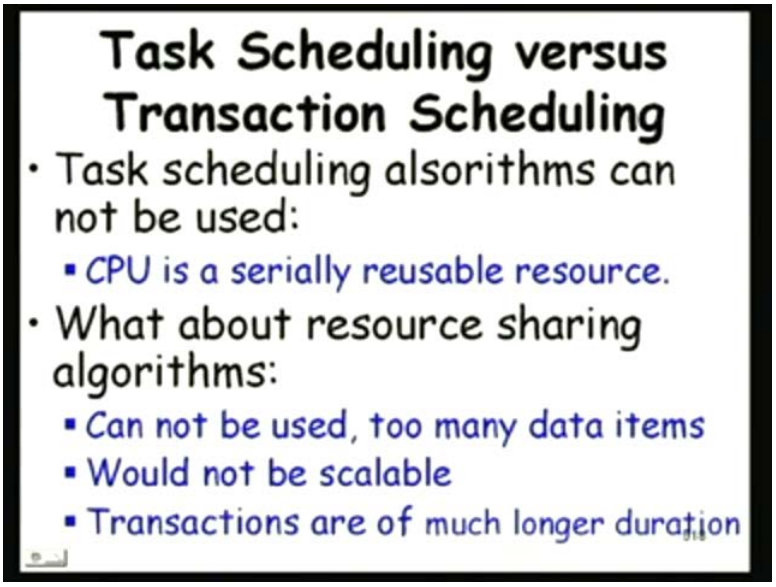
But what about Cascadeless Aborts? This can be achieved by ensuring that every transaction reads only the data values written by the committed transactions, and the partially complete transaction, the results are not shown to the other transaction. Naturally, this will restrict the concurrency or the response time if we do not allow transactions to use partially computed values.

(Refer Slide Time: 34:14)



So, the durability property as we were discussing, is ensured by ensuring that once a transaction commits it cannot be aborted and neither can there be any affect on a complete or a committed transaction due to cascading aborts.

(Refer Slide Time: 34:41)

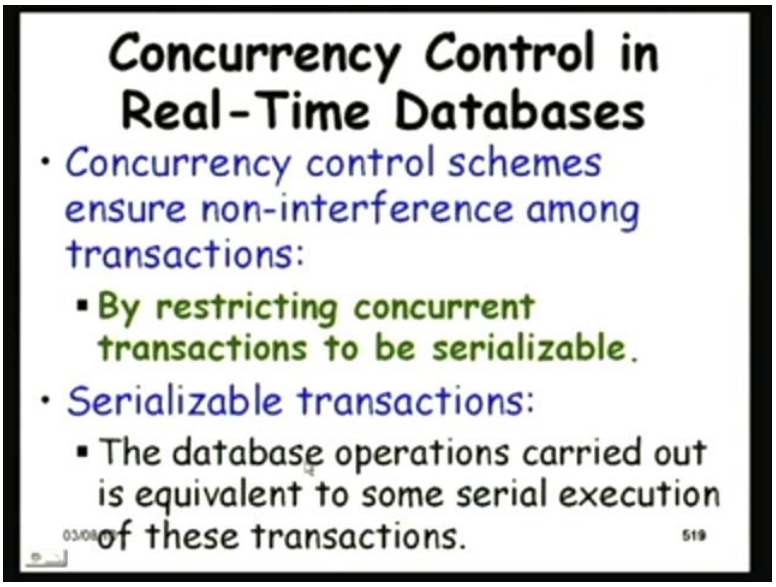


Now, let us try to understand one very fundamental question - how is transaction scheduling different from task scheduling? One thing is that the task scheduling algorithms are very different compared to a transaction scheduling algorithms. One major difference is that the CPU is a serially reusable resource; you can preempt a task

from using the CPU anytime and restart it at a later time without wasting any work. On the other hand, we cannot really stop a transaction anytime and restart later, that will lead to wastage of work.

But what about resource sharing algorithms in the context of task scheduling? Even they cannot be used, because what we are dealing with is too many data items, and the resource sharing algorithm in the context of tasks would become extremely complicated, because we will have to compute ceilings per various resources, and there are millions of them, and it would not be really worthwhile. And not only that, another difference is the transactions are much longer duration compared to tasks. So due to this reason, we cannot use the task scheduling algorithms or resource sharing in the context of task scheduling with any meaningful adaptation. We need entirely different approach to ensure that the transactions meet the deadline.

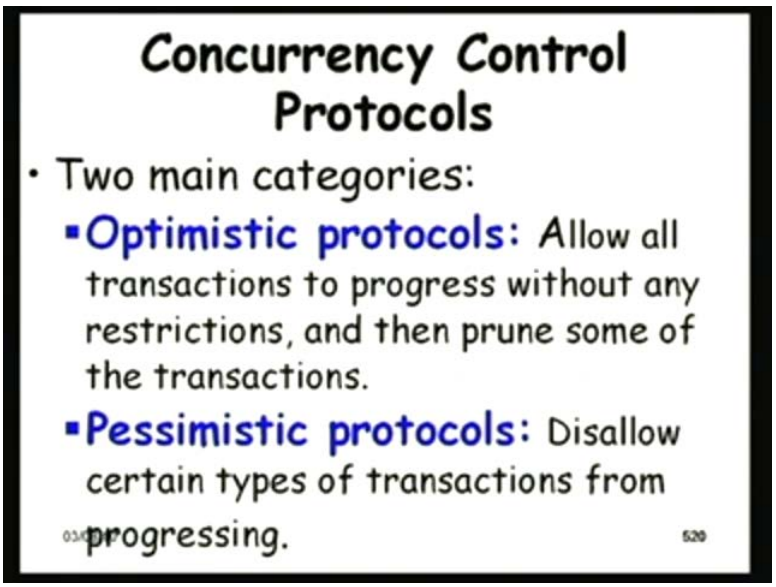
(Refer Slide Time: 36:30)



So, the way it is done is by using appropriate concurrency control schemes, we can ensure the databases can complete in time. So far, we have mentioned that the concurrency control schemes, they actually restricts some transaction interleaving. So, by restricting the transaction interleaving, they allow it to be serializable; that is, they correspond to some serial schedule of operation on the database.

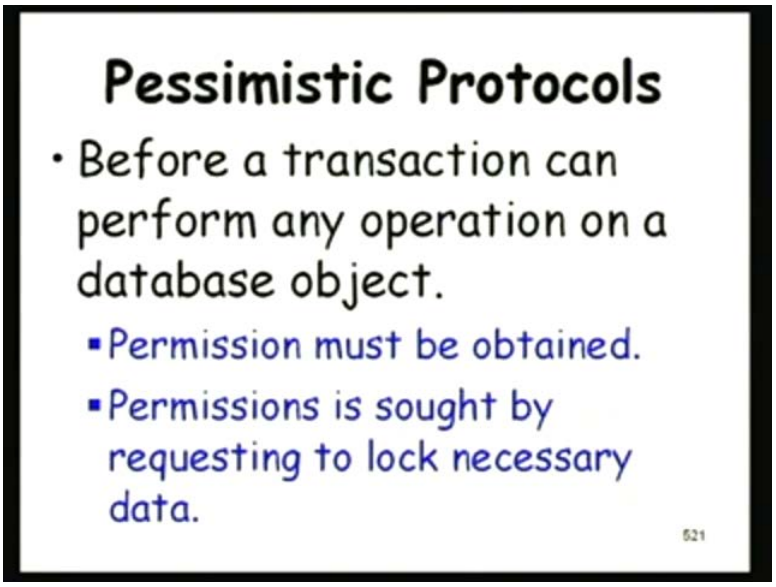
A serializable transaction is such that the database operations carried out is equivalent to some serial execution of the transaction, again a very basic concept from a first level course.

(Refer Slide Time: 37:28)



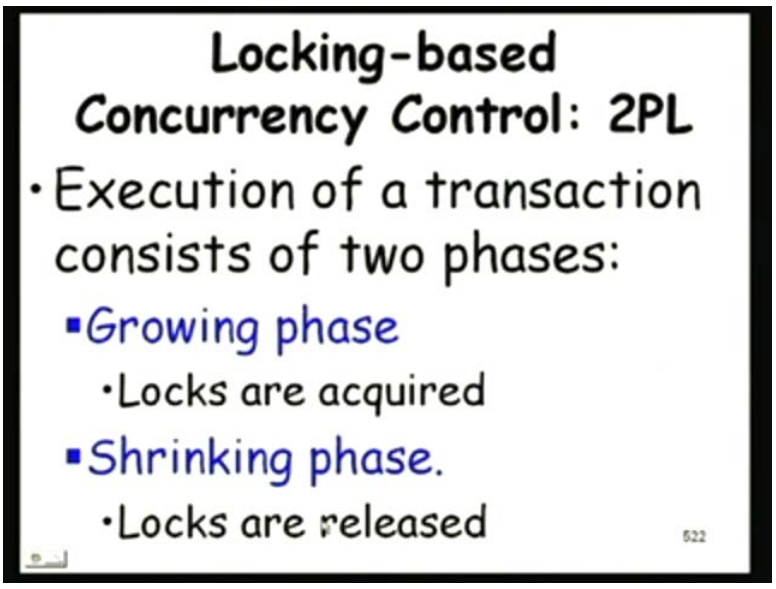
Now, as we said that, this is the primary mechanism we will use to ensure that the real-time database has good performance and the transactions complete in time. Let us see, the kind of protocols that are being used. There are mainly two categories of protocols: the optimistic protocols, which allow transactions to progress without any restrictions, and then, at the time of committing, they prune some of the transactions and abort those; the pessimistic protocols on the other hand, they disallow certain types of transactions from progressing, and those transactions, they are refused some resources, some data and they continue to wait.

(Refer Slide Time: 38:14)



The pessimistic protocols are typically implemented through the use of locks; so here, before a transaction can perform any operation on the database, it seeks locks; it tries to get permissions for locks on the database, and only when the permission is obtained, it can operate on those data.

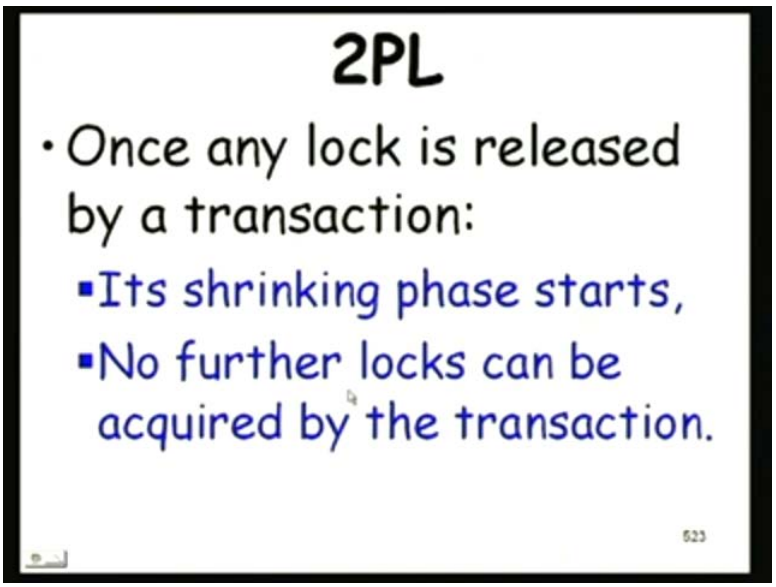
(Refer Slide Time: 38:37)



The most popular pessimistic locking-based concurrency protocol is the 2PL or the two-phase locking; as the name says, two phase locking, there are actually two phases in the execution of a transaction: one is the growing phase where the transaction acquires the

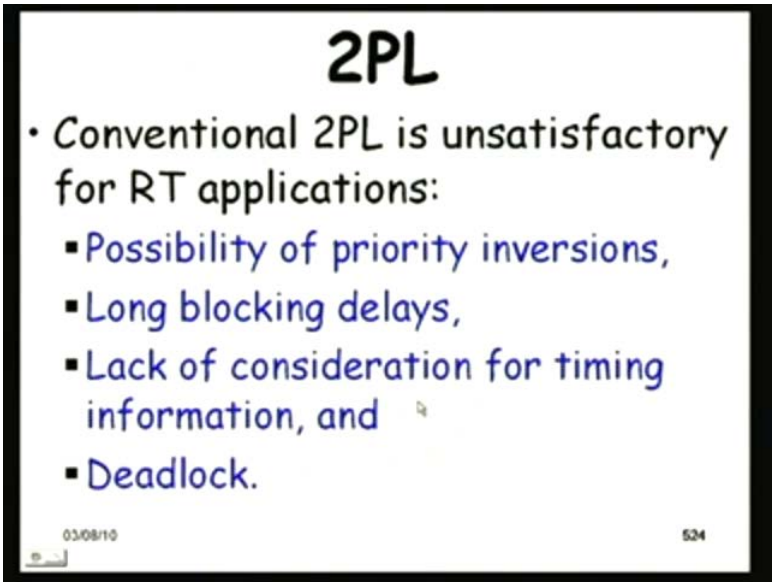
locks and the shrinking phase where the locks are released; and once the shrinking phase starts, no more locks can be acquired, because locks are acquired only in this phase, and once the shrinking phase starts, locks cannot be acquired.

(Refer Slide Time: 39:16)



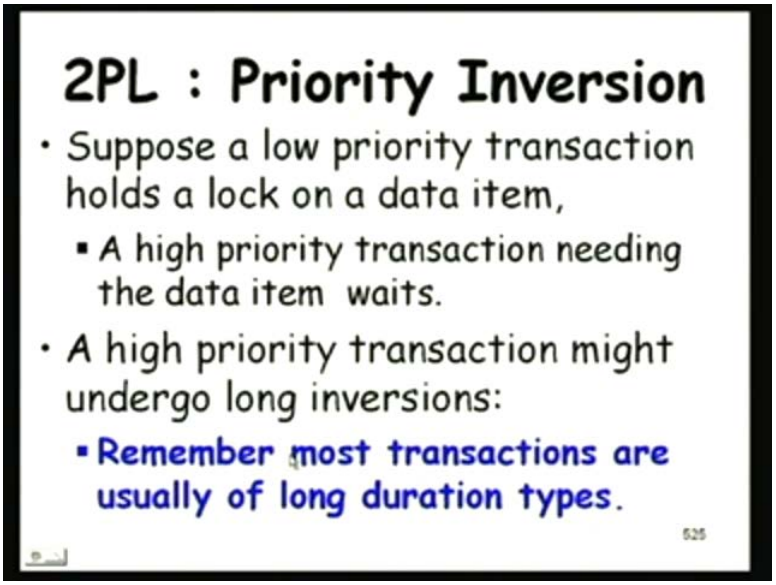
So, once a even a single lock is released, the shrinking phase starts and no further locks can be acquired by the transaction.

(Refer Slide Time: 39:16)



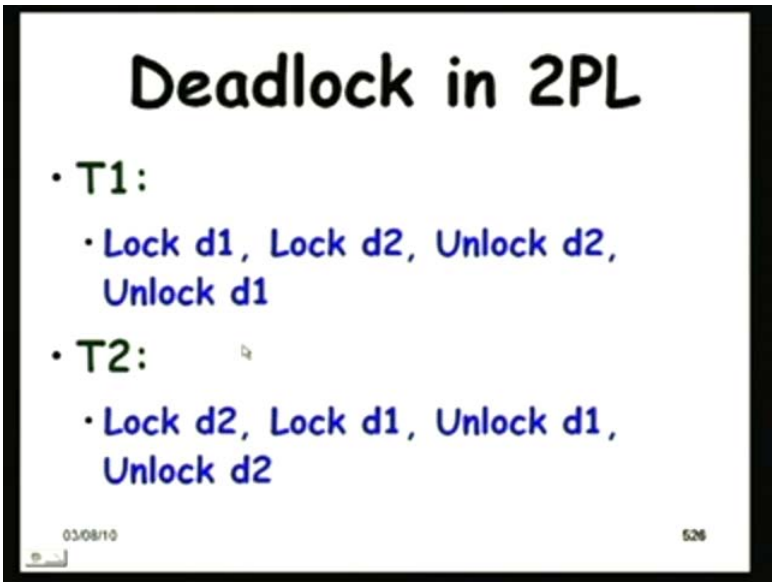
This is a very popular protocol in the traditional databases, but is unsatisfactory for real-time application, because of the possibility of priority inversions, long blocking delays, lack of consideration for timing information and deadlocks. These are some of the problems with 2PL as far as real-time applications are concerned.

(Refer Slide Time: 39:51)



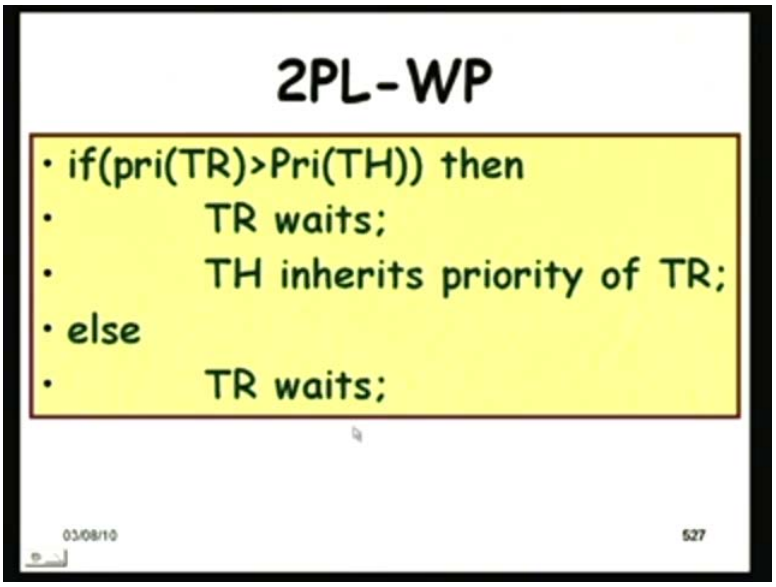
Now, let us see the priority inversion, how does it occur in 2PL. Suppose, a low priority transaction holds a lock on a data item, a higher priority transaction needing that data item will have to wait and the as a result, it will undergo long inversions, because these are long durational databases, most transactions are usually long duration types.

(Refer Slide Time: 40:24)



Deadlocks can arise in very simple to construct examples of deadlock in 2PL. Transaction T1 locks should need to lock d1, then lock d2, and unlock d2 and unlock d1, but t2 need to lock d2, and then, lock d1, then unlock d1 and unlock d2.Suppose, T1 first locks d1 and T2 locks d2. Then, they are deadlocked; neither T1 nor T2 can make progress waiting for each one to release a lock.

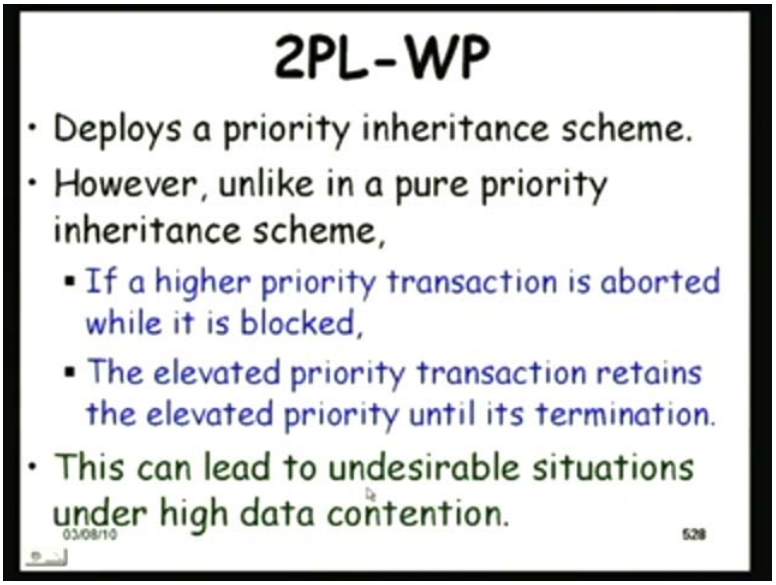
(Refer Slide Time: 41:00)



Now, let us see a mechanism to extend the 2PL for handling real-time transaction. We can define a priority. So, if the priority of the transaction is greater than some threshold

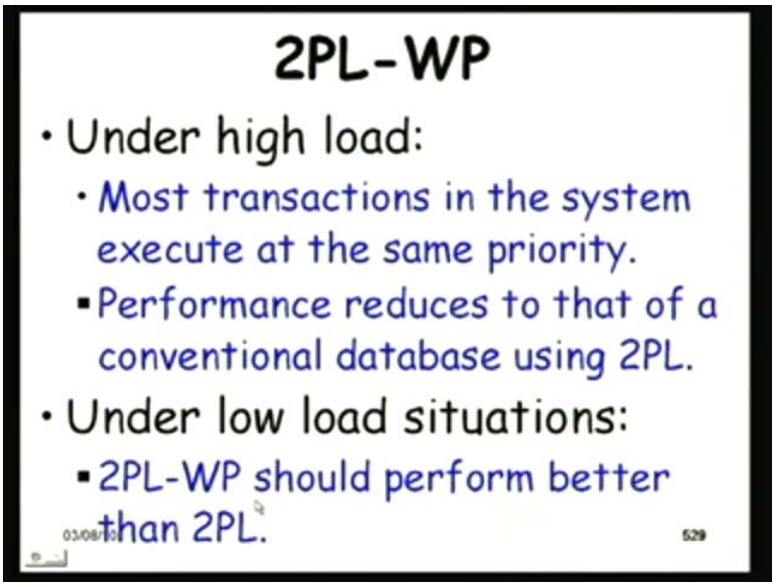
sorry the priority of the requesting transaction is greater than the priority of the holding transaction, then the requesting transaction waits, and the holding transaction inherits the priority of TR, but if the priority of TR is less, then nothing happens, it just continues to wait.

(Refer Slide Time: 41:00)



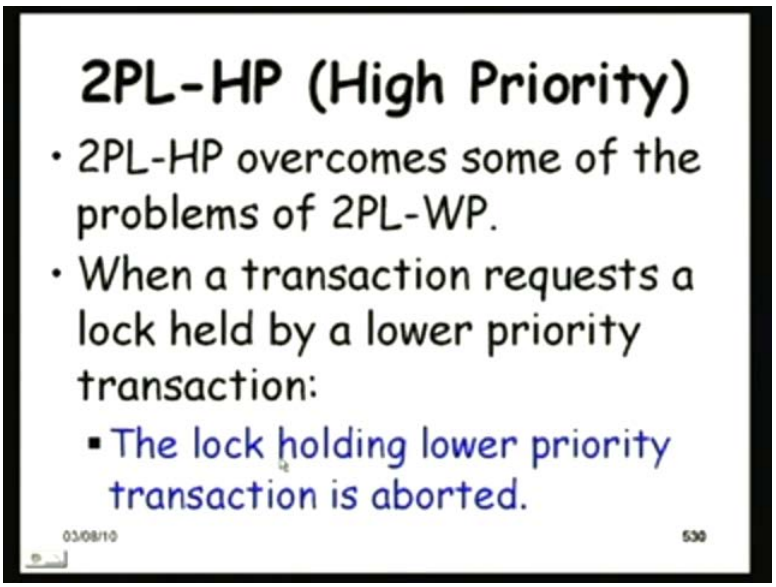
So, as we can see, the 2PL Wait Promote - WP stands for Wait Promote - it deploys a inheritance scheme, but the problem with this is that the elevated priority transaction retains the elevated priority until its termination and it leads to undesirable situations where all transactions operate at very high priorities under high data contention, because each one waits in turn for each other and everybody operates at high priority.

(Refer Slide Time: 41:00)



So, in this case under high load, the performance of 2PL-WP reduces to that of a conventional database using 2PL and the performance really degrades; but under low load situations, the inheritance scheme should allow high priority transactions to complete faster and it should give a better performance.

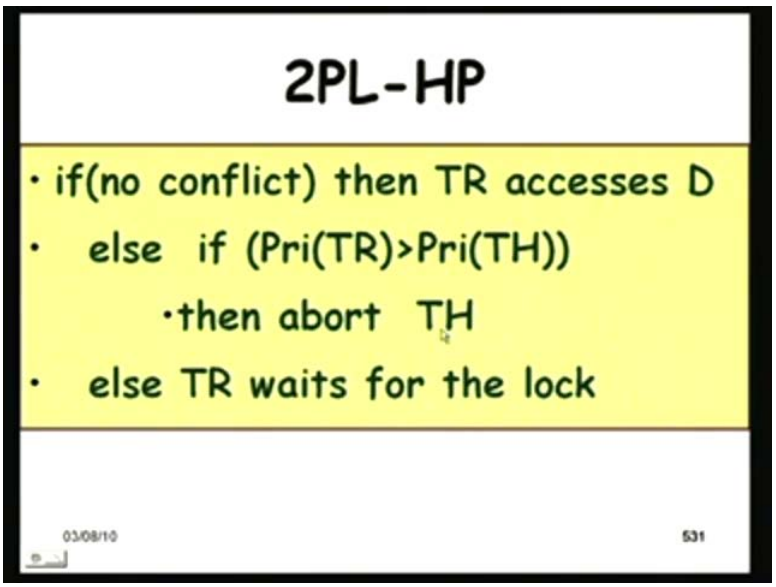
(Refer Slide Time: 42:53)



Now, let us see the 2PL-HP of the high priority which tries to overcome some of the limitations of 2PL-WP; when a transaction requests the lock held by a lower priority transaction, the lock holding transaction lowers, the lock holding lower priority

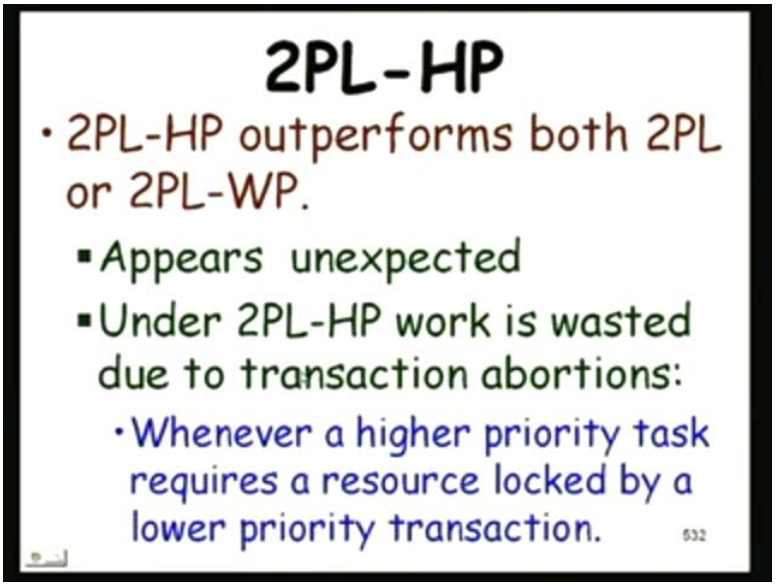
transaction is aborted. So, whenever a higher priority transaction requires a resource being held by a lower priority transaction, the low priority transaction is aborted, so that the high priority transactions get the lock.

(Refer Slide Time: 43:24)



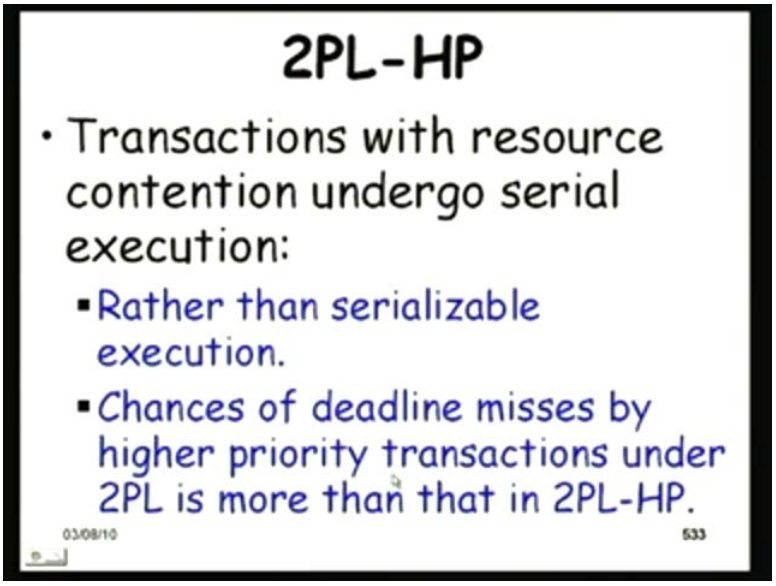
So, here whenever a requesting transaction has higher priority, then the holding transaction is aborted, very simple scheme.

(Refer Slide Time: 43:24)



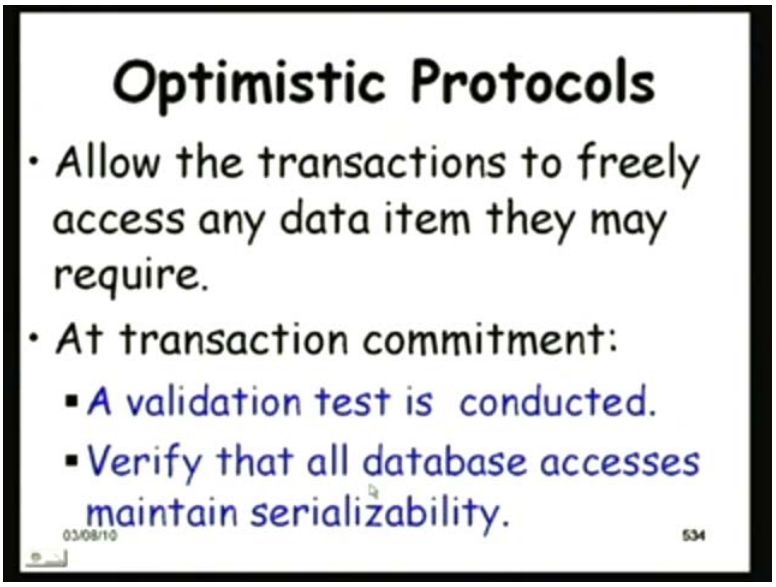
And this outperforms both the 2PL and 2PL-WP. It would appear unexpected, the result, because here, so many transactions are aborted; each time a high priority transaction requests a low resource, the low priority transaction is aborted. So, it appears unexpected that it should perform well, because naturally work is wasted here.

(Refer Slide Time: 43:24)



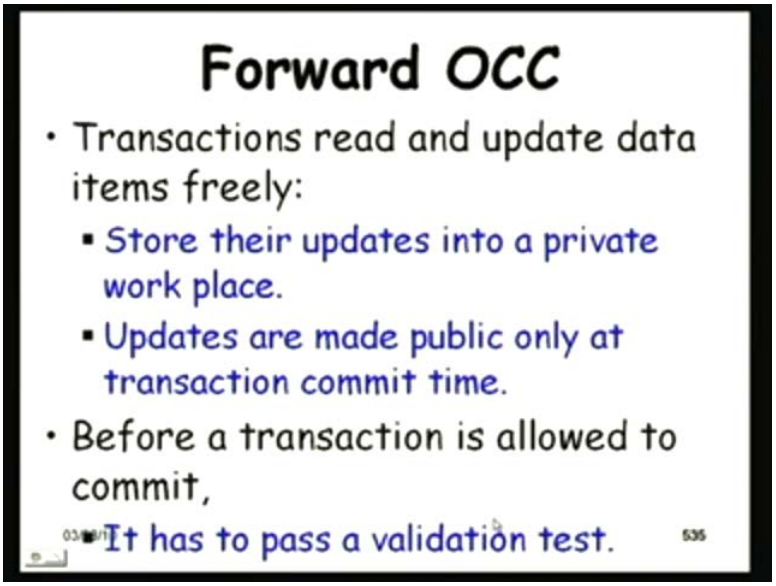
But the transaction with resource constraints, they undergo serial execution rather than serializable execution, and the chances of misses by high priority transaction under 2PL is more than 2PL-HP, and therefore, 2PL-HP performs better.

(Refer Slide Time: 44:33)



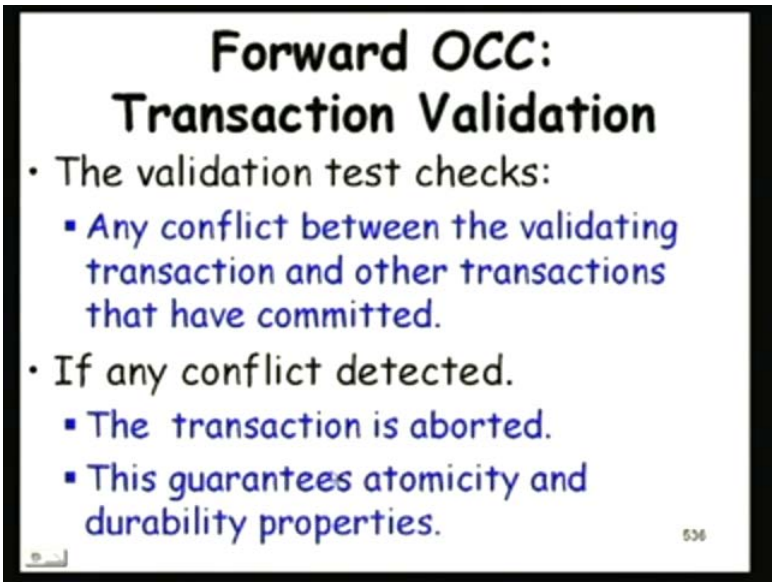
Now, let us look at another category of protocols, which are the optimistic protocol. So, here as you are saying that these protocols, unlike the pessimistic protocols like 2PL, they allow transactions to freely access any data item they might require; no permission, nothing is required; and at the time of transaction commitment, a validation test is conducted to verify that all database accesses maintain serializabilty, and if there is a violation of serializabilty, then some of the transactions are aborted.

(Refer Slide Time: 45:11)



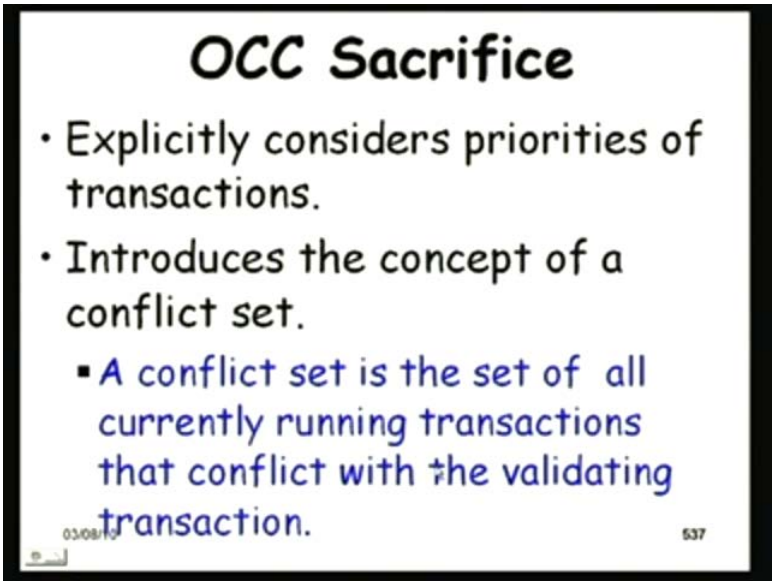
So, one variation is the Forward OCC - Forward Optimistic Concurrency Control protocol algorithm. Here, the transactions read and update data items freely as is with any optimistic protocol; they store their updates in a private workplace, and the updates are made public only at the commitment time, and before a transaction is allowed to commit, a validation test is conducted; now, what if there is a problem in the validation test?

(Refer Slide Time: 45:45)



The validation test checks, the conflict between the validity transaction and the other transactions, which have already committed, because the transactions, which have already committed, nothing can be done due to the durability property. So, if there is a conflict detected with the already committed transaction, then this transaction which was trying to commit would have to be aborted; this would guarantee the atomicity and durability properties.

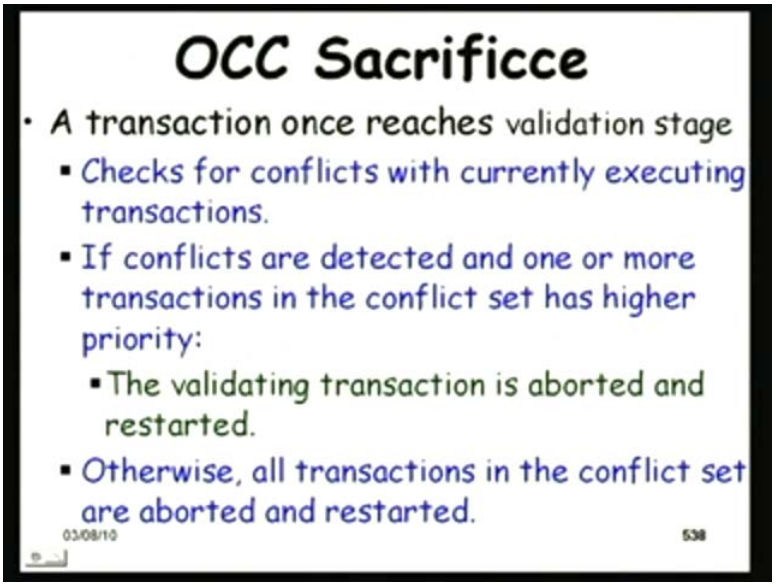
(Refer Slide Time: 46:18)



But the only problem is that in the OCC forward, a transaction which is almost at the end of completion is aborted, and that can lead to not only wastage of lot of work, but also deadline misses for the one that is aborting.

Now, let us consider the OCC sacrifice protocol, which considers the priorities of transactions by introducing concept of a conflict set. A conflict set is the set of all currently running transactions that conflict with the validating transaction. So, each transaction that is running, for which all other transactions that conflict in some operation, they are maintained.

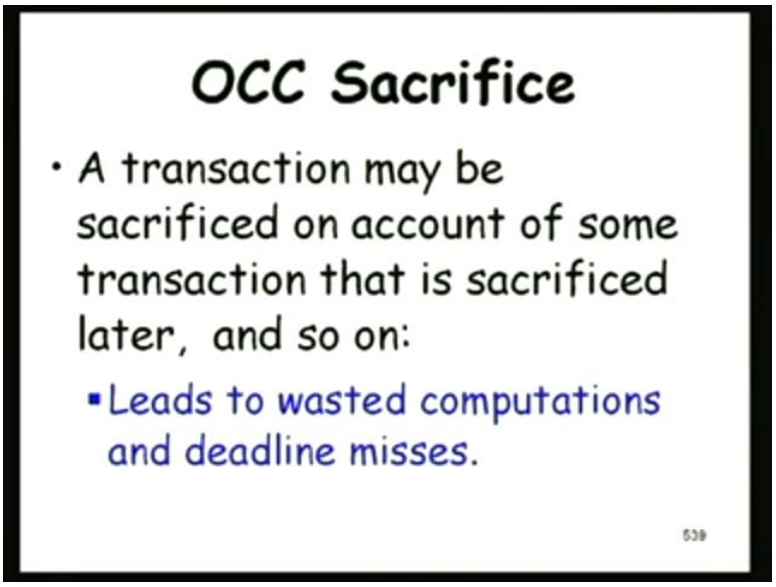
(Refer Slide Time: 46:18)



And once a transaction reaches the validation stage, it checks for conflicts with the currently executing transaction, and if any conflict is detected, and if any one or more transaction has a higher priority than the one that is trying to commit, then the validating transaction is aborted and restarted.

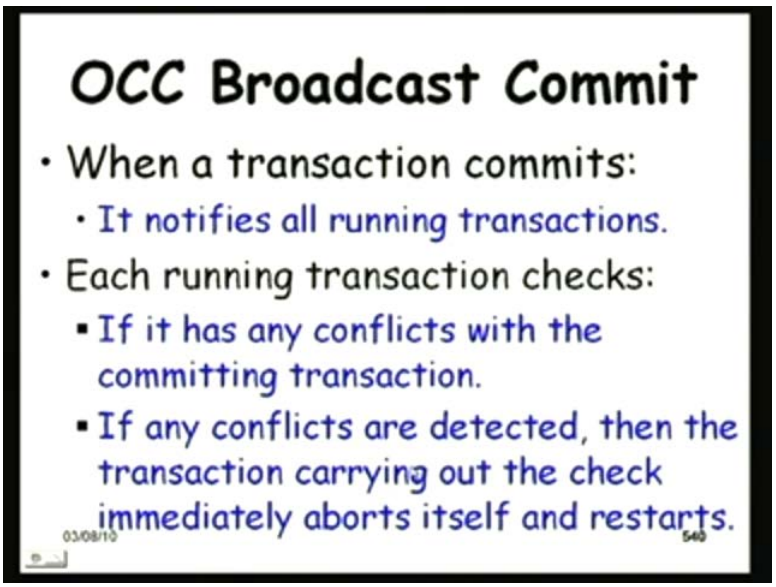
So just see here in the OCC sacrifice, the validation is not with respect to the one which I have already committed, but with respect to those which are currently progressing. And therefore, if the transaction is of higher priority than any of the transaction in the conflict, set, then the transactions in the conflict set are aborted, and they are restarted, and the high priority transaction is allowed to complete; of course, it should work better or give better performance - real-time performance - compared to the OCC forward protocol.

(Refer Slide Time: 46:18)



But a problem here is that a transaction at the end which was sacrificed, which was at the point of commitment was sacrificed on account of some transaction, which will be sacrificed later on and so on. So, this leads to a wasted computations and deadline misses.

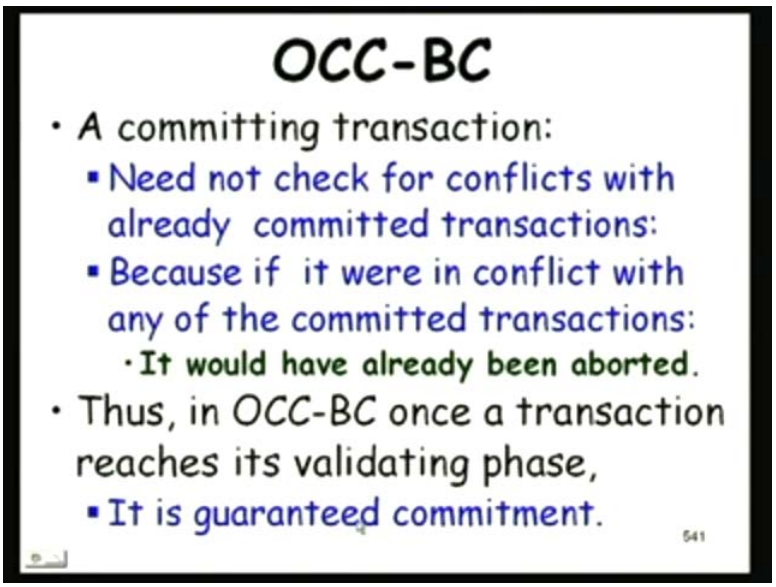
(Refer Slide Time: 48:58)



So, an improved scheme would be the OCC broadcast commit where when a transaction commits, it notifies all running transactions about its intention on committing, and each running transaction would check, if it has any conflict with the committing transaction.

So, if there is a conflict, then the transaction that was running would immediately abort itself and restart. So, just see here, the validating transaction , the one that was trying to commit, is not aborted. So, definitely it is more efficient and it leads to less wasted work.

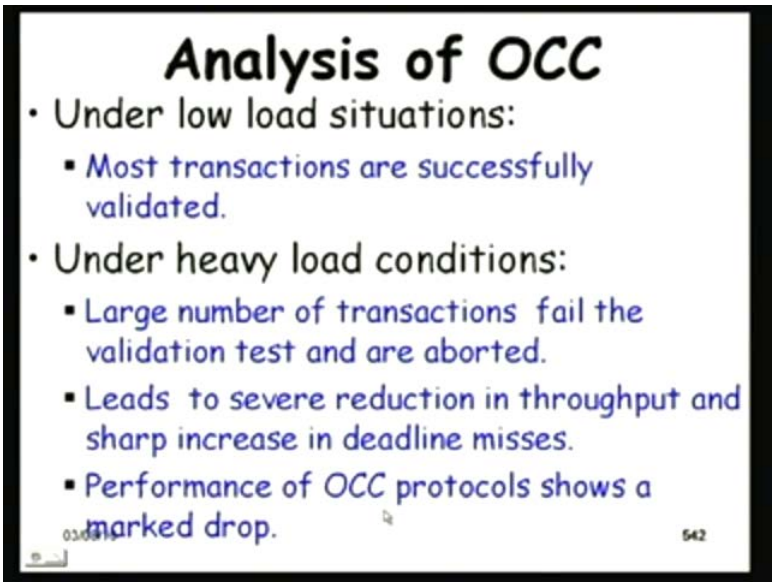
(Refer Slide Time: 49:49)



And also, the transaction that is committing need not check for conflicts with already committed transaction, because if it was in conflict with any of the committed transaction, it would have already been aborted. Since, it has come to the committed transaction stage without being aborted, so, the one which you are already committed must not have any conflict with this.

So, if a transaction was in conflict with any other committed transaction, it would have already been aborted, and since it is nearing commitment, none of the committed transaction will have any conflict with this. And from this reasoning, we can say that in the broadcast commit protocol - the OCC broadcast commit protocol - once a transaction reaches its validating phase, then it would be guaranteed commitment and it would lead to less loss of work.

(Refer Slide Time: 50:59)



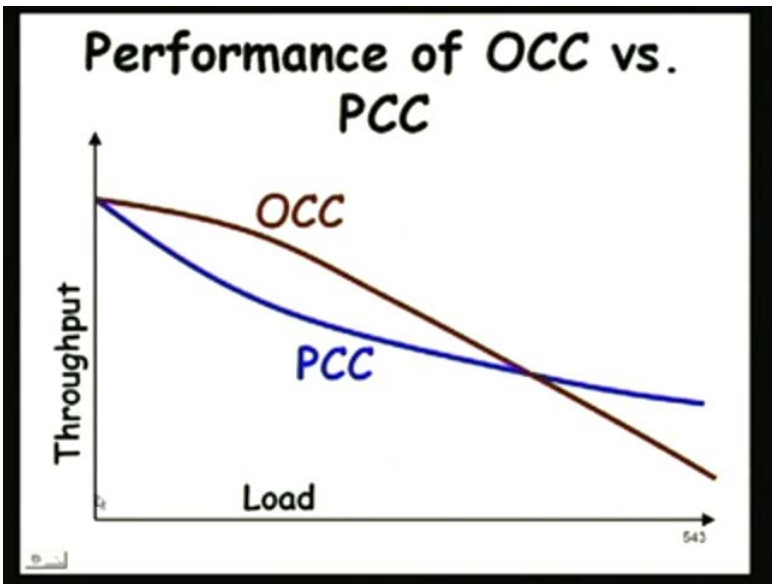
Now, let us try to analyze the performance of OCC. Under low load situations, most transactions are successfully validated, because the conflicts will be very less. So, just see here, we do not take any permission nothing, each transaction proceeds without seeking any permission, and since the conflicts will be very less under low load conditions, all the transactions - almost all transactions - will be successfully validated and they will commit. Naturally here, the overhead will be much less, because no permission is required for any data item, they just operate freely on whatever resources they need, and finally, they all almost all of them complete successfully.

But under heavy load conditions, there are lots of contentions and when a transaction tries to commit, the validation test would detect that there are lot of conflicts, and therefore, it would lead to several aborted transactions, and therefore, the performance of the OCC protocol will come down as the load improves.

On the other hand, in low load situations, the OCC would possibly outperform the pessimistic protocols, but because see here, there is no permission required. They operate freely, and that finally at the commitment, they find that there were very few conflicts and almost all transactions are successfully validated. On the other hand, in the pessimistic protocols even if there are no conflicts, but still locks have to be acquired, before the it the transaction can operate on any data item, and naturally, that is an override.

So under heavy load conditions, many transactions are aborted, because they had operated freely on the data items, and at the time of commitment, the conflicts are detected and leads to lot of aborted transaction; leads to severe reduction in throughput and consequent increase in the deadline misses. So, under heavy load conditions OCC would not perform well and it would show a marked drop.

(Refer Slide Time: 53:52)

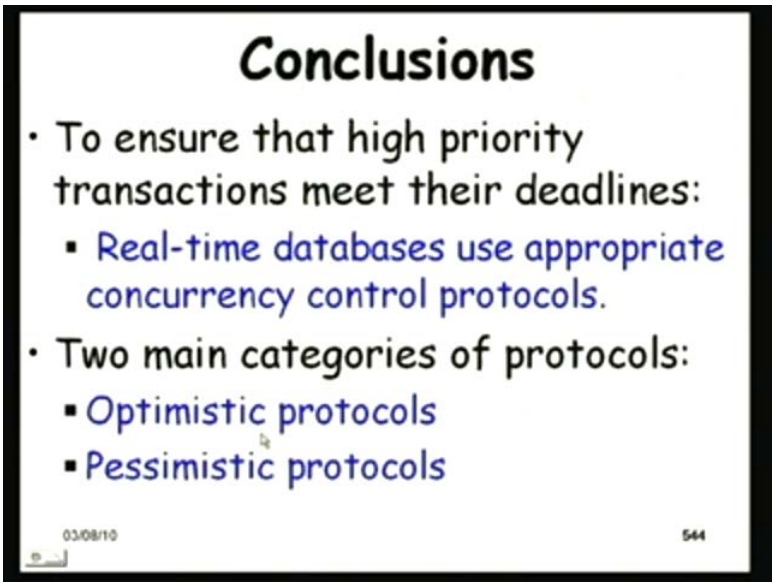


Now, let us see how these two protocols - the optimistic and the pessimistic protocols - would compare with respect to the load. Under light load conditions, the throughput, the chances of deadline misses, etcetera, will be better in OCC. See, this is zero load and there under zero load both are performing similar; zero load means no throughput actually, no transactions are completing, and as the load increases, the throughput of OCC is much better than the pessimistic protocols, because here the transactions operate freely on the data items and hardly there is any conflict.

But as the load increases, the performance of OCC drops, even the PCC drops. The Pessimistic Control Protocol also drops, because as the load increases more and more conflicts are detected and transactions have to wait for acquiring resources. But after sometime, the pessimistic protocol leads to better performance than the OCC, because here too many aborts are taking place, and therefore, under low load situations, an OCC protocol would be preferable. If we design a real-time system where the database should

be always lightly loaded, we can use an OCC protocol, but if there has to be chances of some high load situations, then the pessimistic protocol would have to be used.

(Refer Slide Time: 55:49)



So, now let us conclude on our discussion on real-time databases. So, we had observed that to ensure the high priority transactions to meet their deadlines, concurrency protocols is a important mechanism that is used, and therefore, the concurrency control protocols are a very important parameter in designing real-time databases. And we discussed two main categories of protocols: the optimistic protocols and the pessimistic protocols, and we also discussed the situation in which each one works well. If chances are that the database can become heavily loaded, then pessimistic protocols would have be to be used; whereas, if the database would only the lightly loaded all the time, then the optimistic protocols would perform well.

So, with this, we will complete today's discussion.

Thank you.