Advanced Operating Systems

20MCAT172

(Elective 1)

Module V

Syllabus

Database Systems: Problem of Concurrency Control – Serializability – Basic Synchronization Primitives for Concurrency Control – Lock-Based Algorithms – Time-Stamped Based Algorithms – Optimistic Algorith**ms.**

(Mukesh Singhal and Niranjan G. Shivaratri, "***Advanced Concepts in Operating Systems*** – Distributed, Database, and Multiprocessor Operating Systems", Tata McGraw-Hill, 2001.)

Database Systems

Database Systems

- A database system consists of a set of shared data objects that can be accessed by users.
- A data object can be a page, a file, a segment or a record.
- For the purpose of concurrency control, we will view database as a collection of data objects (d1,d2,.....dm).
- The state of a database is given by the values of its data objects.
- In a database, certain semantic relationships, called **consistency assertions** or **integrity constraints** must hold among its data objects.
- A database is said to be **consistent** if the values of its data objects satisfy all of its consistency assertions.

Transactions

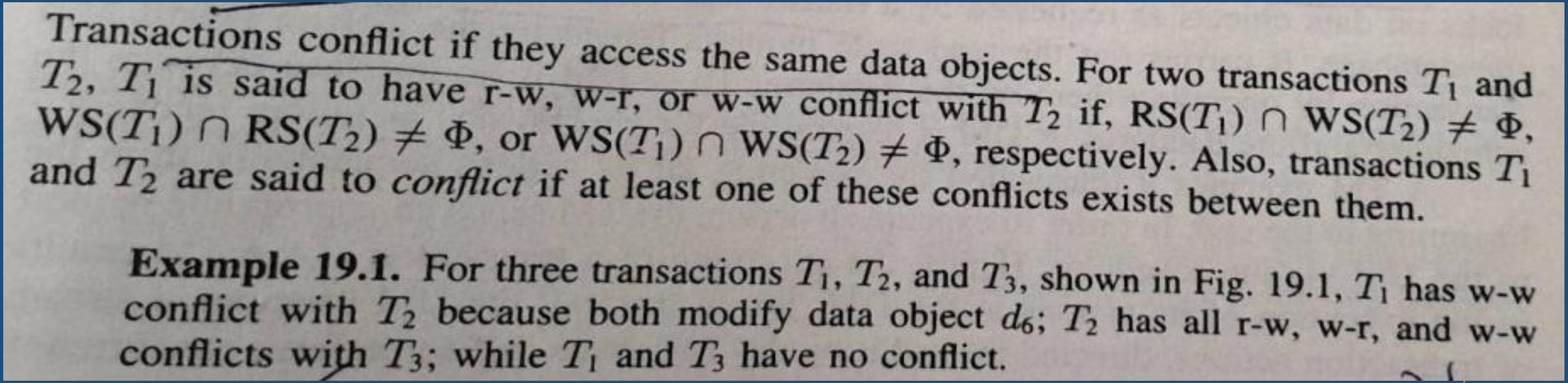
- A user interacts with a database by performing read and write actions on the data objects.
- The actions of a user are normally group together (as a program) to form a single logical unit of interaction, termed a **transaction**.
- A transaction consists of a sequence of read, compute and write statements that refer to the data objects of a database.
- The following are the properties of a transaction:
    - A transaction preserves the consistency of a database.
    - A transaction terminates in finite time.

Transactions

- A transaction that does not modify any data object but just read some of them is referred as a read-only transaction.
- A transaction that modifies at least one date object is known as an update transaction or an update.
- The term transaction is used in a general sense to stand for query or update.
- For a transaction, the set of data objects that are read by it are referred to as it's Readset and the set of data objects that are written by it are referred to as its Writeset.
- Readset of a transaction - RS(T)
- Writeset of a transaction – WS(T)

Transactions

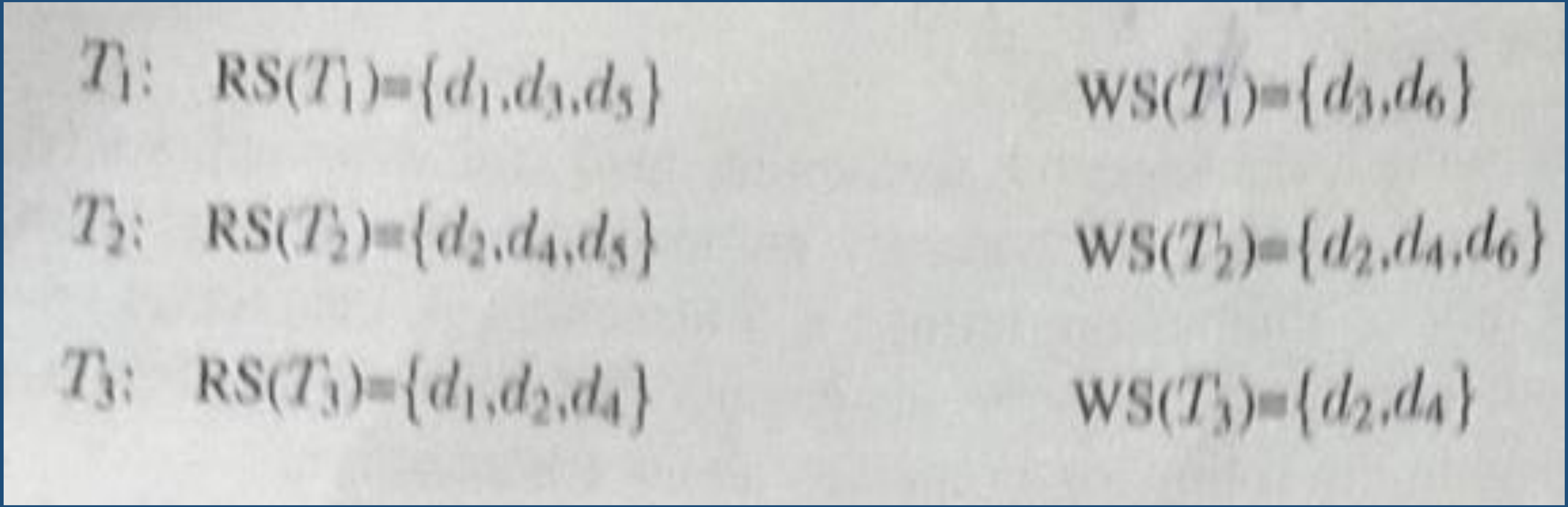
- Conflict in DBMS can be defined as **two or more different transactions accessing the same variable** and atleast one of them is a write operation.
- There are three types of conflict in the database transaction.
  - Write-Read (WR) conflict
  - Read-Write (RW) conflict
  - Write-Write (WW) conflict



Transactions conflict if they access the same data objects. For two transactions T_1 and T_2, T_1 is said to have r-w, w-r, or w-w conflict with T_2 if, $RS(T_1) \cap WS(T_2) \neq \Phi$, $WS(T_1) \cap RS(T_2) \neq \Phi$, or $WS(T_1) \cap WS(T_2) \neq \Phi$, respectively. Also, transactions T_1 and T_2 are said to *conflict* if at least one of these conflicts exists between them.

Example 19.1. For three transactions T_1, T_2, and T_3, shown in Fig. 19.1, T_1 has w-w conflict with T_2 because both modify data object d_6; T_2 has all r-w, w-r, and w-w conflicts with T_3; while T_1 and T_3 have no conflict.

Transactions



T_1: $RS(T_1)=\{d_1,d_3,d_5\}$ $WS(T_1)=\{d_3,d_6\}$

T_2: $RS(T_2)=\{d_2,d_4,d_5\}$ $WS(T_2)=\{d_2,d_4,d_6\}$

T_3: $RS(T_3)=\{d_1,d_2,d_4\}$ $WS(T_3)=\{d_2,d_4\}$

Transaction Processing

- A transaction is executing its action one by one from the beginning to the end.
- A Read Action of a transaction is executed by reading the data object in the workspace of the transaction.
- A Write Action of a transaction modifies a data object in the workspace and eventually writes it to the database.

The Concurrency Control model of Database Systems

- A database system consist of three software modules:
    - A transaction manager (TM)
    - A data Manager (DM)
    - A Scheduler
- A TM supervises the execution of a transaction.
- A TM interacts with the DM to carry out the execution of a transaction.
- TM assign a timestamp to a transaction or issue requests to lock and unlock data objects on behalf of a user.
- TM is an interface between users and the database system.

The Concurrency Control model of Database Systems

- The scheduler is responsible for enforcing concurrency control.
- It grants or releases locks on data objects as requested by a transaction.
- The DM manages the database.
- DM carries out the read-write requests issued by the TM.
- DM is an interface between the scheduler and the database.
- The DM is responsible for failure recovery.

The Concurrency Control model of Database Systems

- A TM executes a transaction by executing all its actions sequentially from the beginning to the end.
- In order to execute an action, the TM sends an appropriate request to the DM via the scheduler.
- DM executes a stream of transaction actions.
- To perform concurrency control, the scheduler modifies the stream of actions directed toward the DM.

The Concurrency Control model of Database Systems

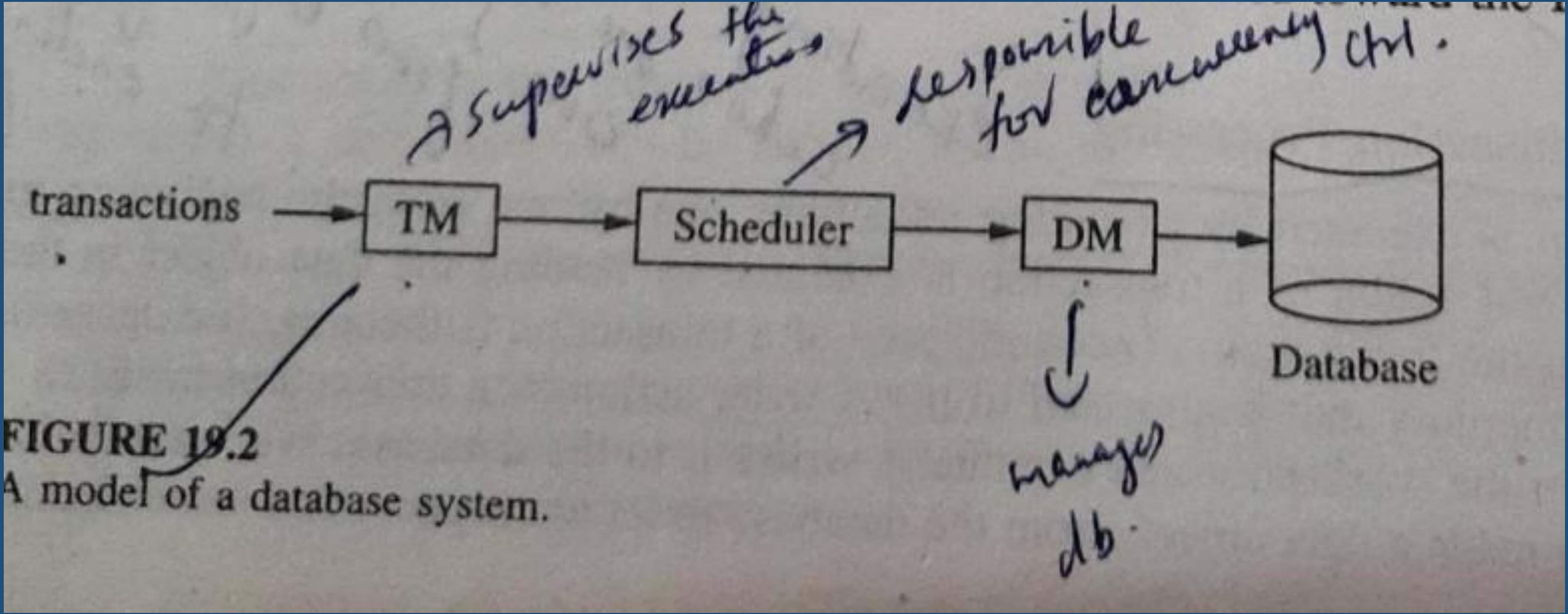


FIGURE 19.2
A model of a database system.

The Problem of Concurrency Control

- In a database system, several transactions are under execution simultaneously.

- Efficiency can be improved by executing transactions concurrently, ie, by executing read and write actions from several transactions in an interleaved manner.

- Since concurrently running transactions may access the same data objects, the following situations may arise:

    1. Inconsistent Retrieval
    2. Inconsistent Update

The Problem of Concurrency Control

1. Inconsistent Retrieval
   - It occurs when a transaction reads some data objects of a database before another transaction has completed with its modification of those data objects.

2. Inconsistent Update
   - It occurs when many transactions read and write onto a common set of data objects of a database, leaving the database in an inconsistent state.

Serializability

- **The** transactions are set of instructions and these instructions perform operations on database.
- When multiple transactions are running concurrently then there needs to be a sequence in which the operations are performed because at a time only one operation can be performed on the database.
- This sequence of operations is known as **Schedule**.
- When multiple transactions are running concurrently then there is a possibility that the database may be left in an inconsistent state.
- Serializability is a concept that helps us to check which Schedules are serializable.
- A serializable schedule is the one that always leaves the database in consistent state.
- Execution of a transaction is modeled by a **log** and the correctness condition is stated in terms of logs.

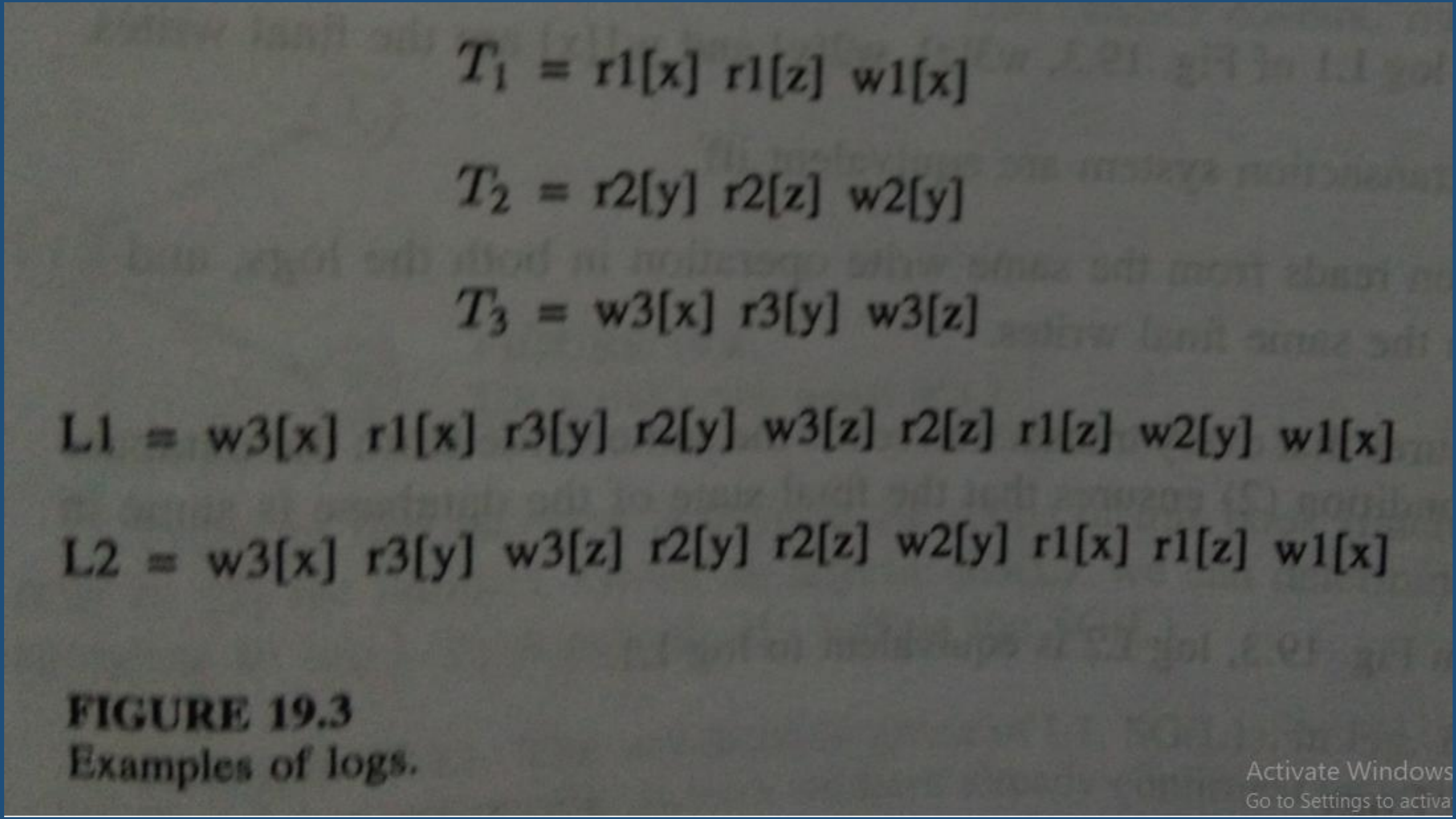
Serializability

LOGS

- The serializability theory models executions of a concurrency control algorithm by a history variable called the log (also called schedule).
- A log captures the chronological order in which read and write actions of transactions are executed under a concurrency control algorithm.
- Let T= $\{T_0, T_1, \ldots T_n\}$ be a transaction system. A log over T models an interleaved execution of $T_0, T_1, \ldots T_n$.

Serializability

LOGS



T_1 = r1[x] r1[z] w1[x]

T_2 = r2[y] r2[z] w2[y]

T_3 = w3[x] r3[y] w3[z]

L1 = w3[x] r1[x] r3[y] r2[y] w3[z] r2[z] r1[z] w2[y] w1[x]

L2 = w3[x] r3[y] w3[z] r2[y] r2[z] w2[y] r1[x] r1[z] w1[x]

FIGURE 19.3
Examples of logs.

Serializability

Serial Logs

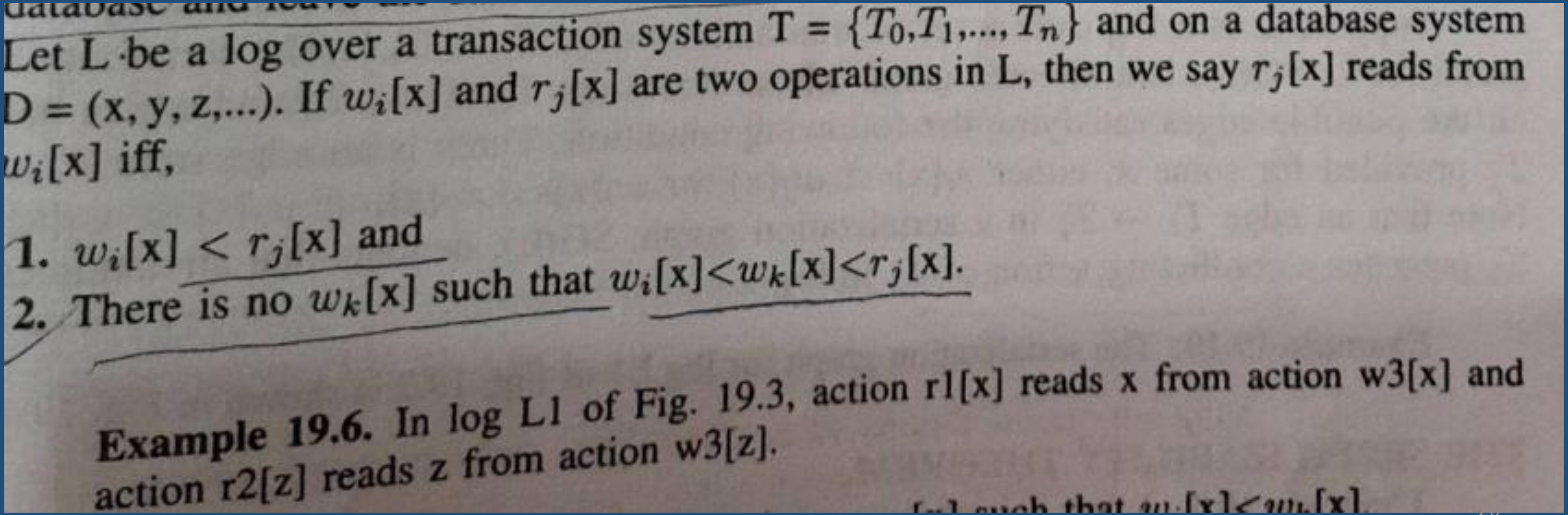
- In a database system, if transactions are executed strictly serially, all the actions of each transaction must complete before any action of the next transaction can start, then the resulting log is termed a serial log.
- A serial log represents an execution of transactions where actions from different transactions are not interleaved.
- For eg. For a set of transactions T_1 , T_2 ,......, T_n , a serial log is of the form T_{i1} T_{i2} T_{in} ,where i1,i2,....,in is a subset of T1.

- EXAMPLE : Log L2 of Fig. 19.3 (Actions from different transactions have not been interleaved)

Serializability

Log Equivalence

- Two logs are equivalent if all the transactions in both the logs see the same state of the database and leave the database in the same state after all the transactions are finished.



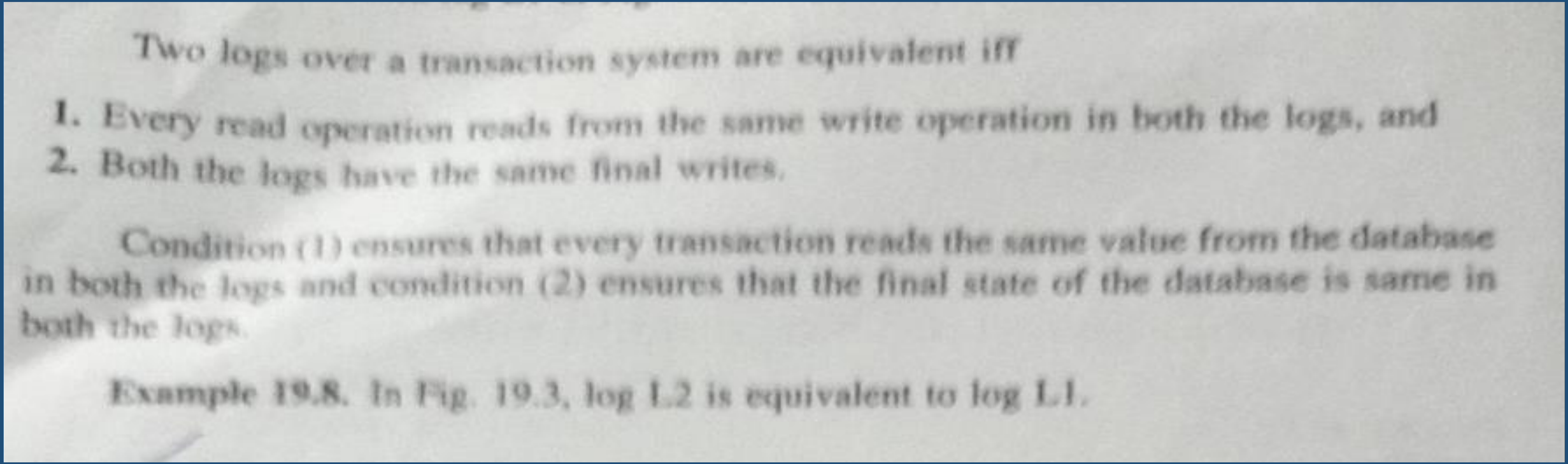
Let L be a log over a transaction system T = $\{T_0, T_1, \ldots, T_n\}$ and on a database system D = (x, y, z,...). If $w_i[x]$ and $r_j[x]$ are two operations in L, then we say $r_j[x]$ reads from $w_i[x]$ iff,

1. $w_i[x] < r_j[x]$ and
2. There is no $w_k[x]$ such that $w_i[x]<w_k[x]<r_j[x]$.

Example 19.6. In log L1 of Fig. 19.3, action r1[x] reads x from action w3[x] and action r2[z] reads z from action w3[z].

Serializability

Log Equivalence



Two logs over a transaction system are equivalent iff

1. Every read operation reads from the same write operation in both the logs, and
2. Both the logs have the same final writes.

Condition (1) ensures that every transaction reads the same value from the database in both the logs and condition (2) ensures that the final state of the database is same in both the logs.

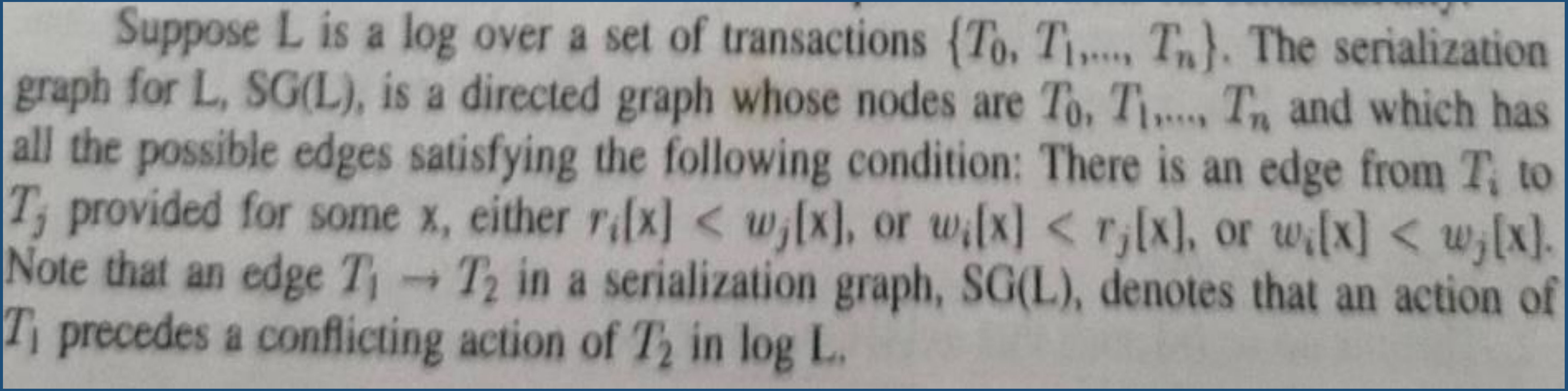
Example 19.8. In Fig. 19.3, log L2 is equivalent to log L1.

Serializability

The SerializabilityTheorem:

“ A log L is serializable iff SG(L) is acyclic. “

- **A serialization graph is constructed from a log.**



Suppose L is a log over a set of transactions $\{T_0, T_1, \ldots, T_n\}$. The serialization graph for L, SG(L), is a directed graph whose nodes are $T_0, T_1, \ldots, T_n$ and which has all the possible edges satisfying the following condition: There is an edge from T_i to T_j provided for some x, either $r_i[x] < w_j[x]$, or $w_i[x] < r_j[x]$, or $w_i[x] < w_j[x]$. Note that an edge $T_1 \rightarrow T_2$ in a serialization graph, SG(L), denotes that an action of T_1 precedes a conflicting action of T_2 in log L.

Basic Synchronization Primitives for Concurrency Control

Basic Synchronization Primitives for Concurrency Control

- **Concurrency** is the execution of the multiple instruction sequences at the same time.
- It happens in the operating system when there are several process threads running in parallel.
- The running process threads always communicate with each other through shared memory or message passing.
- Concurrency results in sharing of resources result in problems like deadlocks and resources starvation.
- It helps in techniques like coordinating execution of processes, memory allocation and execution scheduling for maximizing throughput.

Basic Synchronization Primitives for Concurrency Control

Principles of Concurrency :

- Both interleaved and overlapped processes can be viewed as examples of concurrent processes, they both present the same problems.

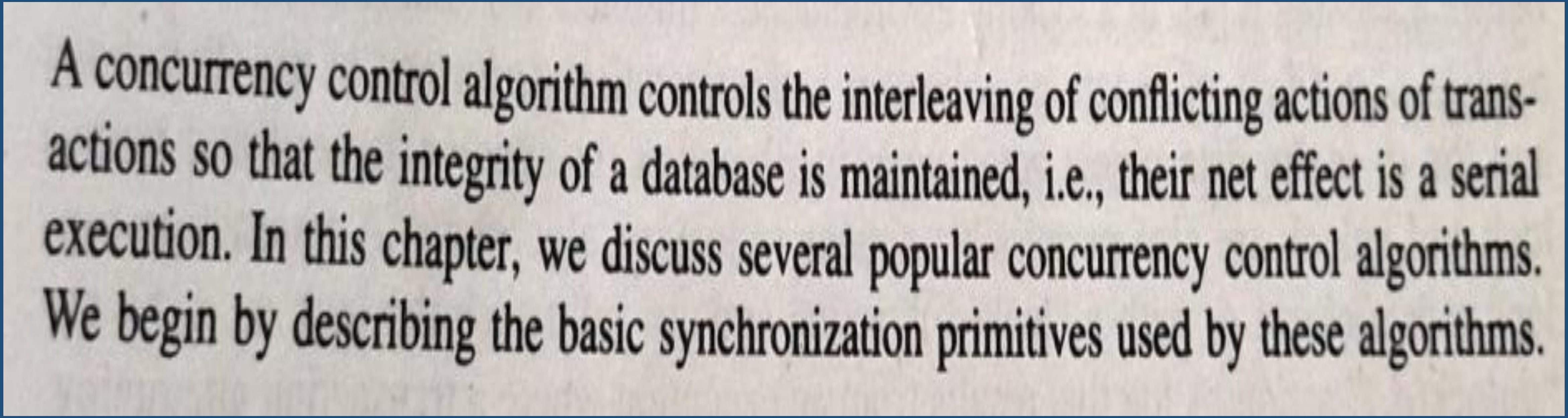
- The relative speed of execution cannot be predicted.
- It depends on the following:
    - The activities of other processes
    - The way operating system handles interrupts
    - The scheduling policies of the operating system

Basic Synchronization Primitives for Concurrency Control

Problems in Concurrency :

- **Sharing global resources**
- **Optimal allocation of resources**
- **Locating programming errors**
- **Locking the channel**

Concurrency Control Algorithms



A concurrency control algorithm controls the interleaving of conflicting actions of transactions so that the integrity of a database is maintained, i.e., their net effect is a serial execution. In this chapter, we discuss several popular concurrency control algorithms. We begin by describing the basic synchronization primitives used by these algorithms.

Basic Synchronization Primitives

- LOCKS
- TIMESTAMPS

Basic Synchronization Primitives

- **LOCKS**
  - Each data object has a lock associated with it.
  - A transaction can request, hold, or release the lock on a data object.
  - When a transaction holds a lock, the transaction is said to have locked the corresponding data object.
  - A transaction can lock a data object in two modes: Exclusive and Shared.
  - If a transaction has locked a data object in exclusive mode, no other transaction can concurrently lock it in any mode.
  - If a transaction has locked a data object in shared mode, other transactions can concurrently lock it but only in shared mode.
  - By locking data objects, a transaction ensures that the locked data objects are inaccessible to other transactions.

Basic Synchronization Primitives

- **TIMESTAMPS**
  - A unique number that is assigned to a transaction or a data object.
  - Timestamps are commonly generated according to Lamport’s Scheme.
  - Timestamps have two properties:
    - Uniqueness
      - They are unique system wide.
    - Monotonicity
      - The value of timestamp increases with time.
  - Timestamps allow us to place a total ordering on the transactions of a distributed database system by simply ordering the transactions by their timestamps.

Lock-Based Algorithms

LOCK BASED ALGORITHMS

- In lock based concurrency control algorithms, a transaction must lock a data object before accessing it.
- In a locking environment, a transaction T is a sequence $\{a_1 (d_1), a_2 (d_2), \ldots., a_n (d_n)\}$ of n actions, where a_i is the operation performed in the ith action and the d_i is the data object acted upon in ith action.
- Lock and Unlock are also permissible actions in locking algorithms.
- A transaction can lock a data object d_i with a ***"lock(d_i)"*** action and can unlock it by an ***"unlock(d_i)"*** action.
- A log that results from an execution where a transaction attempting to lock an already locked data object waits, is referred to as a **legal log**.

LOCK BASED ALGORITHMS

- A transaction is well-formed if it
  - Locks a data object before accessing it.
  - Does not lock a data object more than once, and
  - Unlocks all the locked data objects before it completes.

- To guarantee serializability, additional constraints needed, These constraints are expressed as locking algorithms.

LOCK BASED ALGORITHMS

STATIC LOCKING

- In static locking, a transaction acquires locks on all the data objects it needs before executing any action on the data objects.
- Static locking requires a transaction to pre-declare all the data objects it needs for execution.
- A transaction unlocks all the locked data objects only after it has executed all of its actions.

LOCK BASED ALGORITHMS

STATIC LOCKING

- Static Locking is conceptually very simple.
- It seriously limits concurrency because any two transactions that have a conflict must execute serially.
- Drawbacks
    - Limits the performance of database system.
    - It requires a priori knowledge of the data objects to be accessed by transactions. (Impractical in applications where the next data objects to be locked depends upon the value of another data object.)

LOCK BASED ALGORITHMS

Two-Phase Locking (2PL)

- Dynamic Locking scheme in which a transaction requests a lock on a data object when it needs the data object.
- Database consistency is not guaranteed if a transaction unlocks a locked data object immediately after it is done with it.
- Two-Phase locking imposes a constraint on lock acquisition and the lock release actions of a transaction to guarantee consistency.

LOCK BASED ALGORITHMS

- In two-phase locking, a transaction cannot request a lock on any data object after it has unlocked a data object.
- Thus, a transaction must have acquired locks on all the needed data objects before unlocking a data object.
- A two-phase locking has two phases:
  - A Growing Phase
  - Shrinking Phase

Two-Phase Locking (2PL)

- A two-phase locking has two phases:
  - A Growing Phase
    - During which a transaction requests locks (Without releasing any lock)
  - Shrinking Phase
    - Which starts with the first unlock action, during which a transaction releases locks. (Without requesting any more locks)

- The stage of a transaction when the transaction holds locks on all the needed data objects is referred to as its ***lock point***.

Two-Phase Locking (2PL)

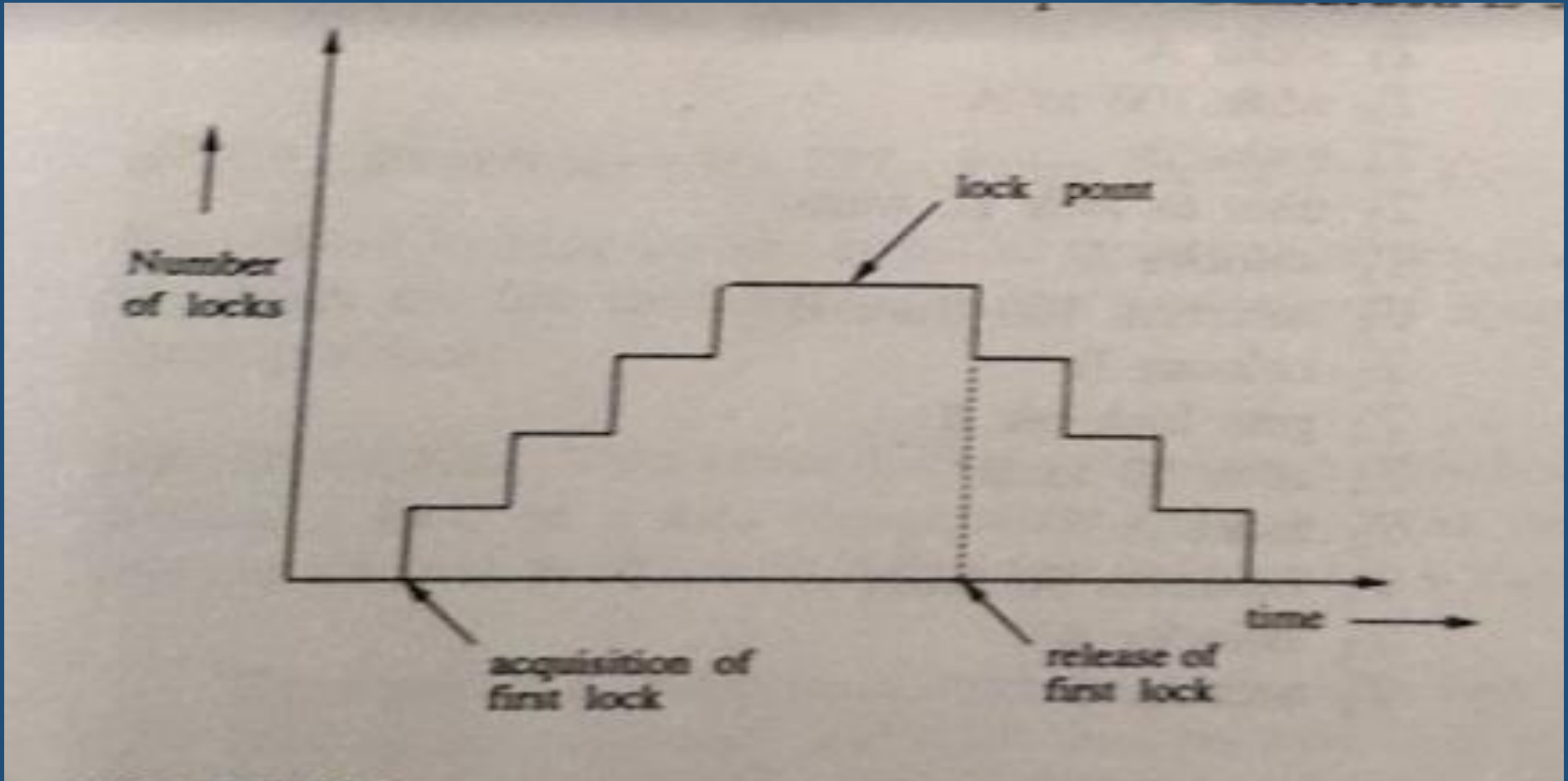


FIGURE 20.1
A schematic diagram of a two-phased transaction.

Problems With 2PL

- Two-Phase locking suffers from the problems of **deadlock** and **cascaded aborts.**
- **Two Phase** locking is prone to deadlocks because a transaction can request a lock on a data object while holding locks on other data objects.
- A set of transactions are deadlocked if they are involved in a circular wait.
- When a transaction is rolled back, all the data objects modified by it are restored to their original states. In this case, all transactions that have read by them must also be restored and so on. This phenomenon is called the cascaded roll-back.
- Two phase locking suffers from the problem of cascaded roll-back because a transaction may be rolled back after it has released the locks on some data objects and other transactions have read those modified data objects.

Timestamp-Based Locking

- When a transaction is submitted, it is assigned a unique timestamp.
- The timestamps of transaction specify a total order on the transactions and can be used to resolve conflicts between transactions.
- When a transaction conflicts with another transaction, the concurrency control algorithm makes a decision based on the result of the comparison of their timestamp.

Timestamp-Based Locking

- The use is to prevent deadlock.
- Conflict Resolution: A conflicit is resolved by taking one of the following actions.
    - Wait-The requesting transaction is made to wait until the conflicting transaction either completes or aborts.
    - Restart-Either the requesting transaction or the transaction it conflicts with is aborted and started afresh.

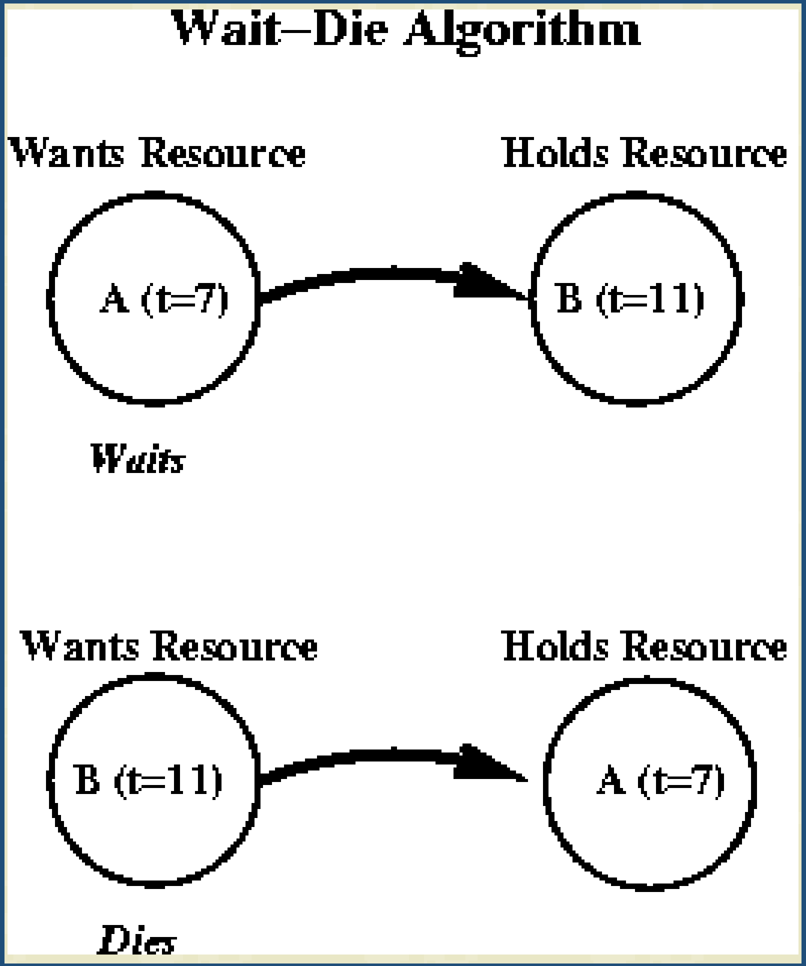
  Restarting is achieved by using one of the following primitives.

  - Die- Requesting transaction aborts and starts afresh.
  - Wound- The transaction in conflicit with the requesting transaction is tagged as wounded and a message "wounded" is sent to all sites that the wounded transaction has visited.
- Wait-Die Algorithm
- Wound-Wait Algorithm

Timestamp-Based Locking

The ***Wait-Die*** algorithm:

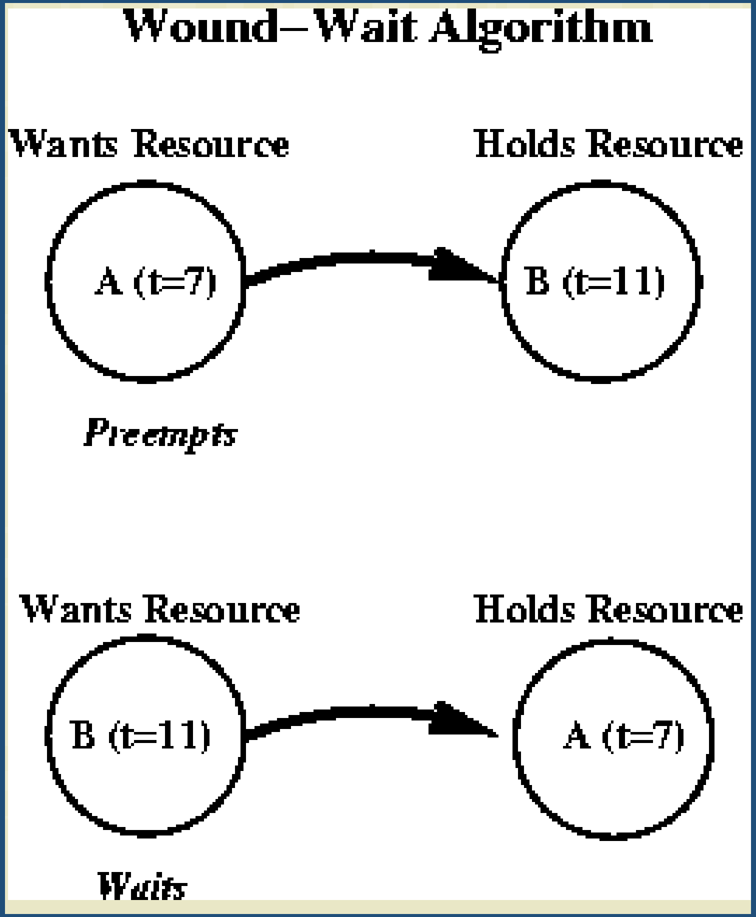
- Allow wait only if waiting process is older.
- Since timestamps increase in any chain of waiting processes, cycles are impossible.



Timestamp-Based Locking

The ***Wound-Wait*** algorithm: Preemptive algorithm.

- Allow wait only if waiting process is younger.
- Here timestamps decrease in any chain of waiting process, so cycles are again impossible.
  It is wiser to give older processes priority.



Timestamp-Based Locking

Comparison Between The Algorithms

The Wound-Wait algorithm preempts the younger process. When the younger process re-requests resource, it has to wait for older process to finish. This is the better of the two algorithms.

- Waiting Time:
  - In the WAIT-DIE algorithm, an older transaction is made to wait for younger ones.
  - In the WOUND-DIE algorithm, an older transaction never waits for younger ones and wound all the younger transactions.

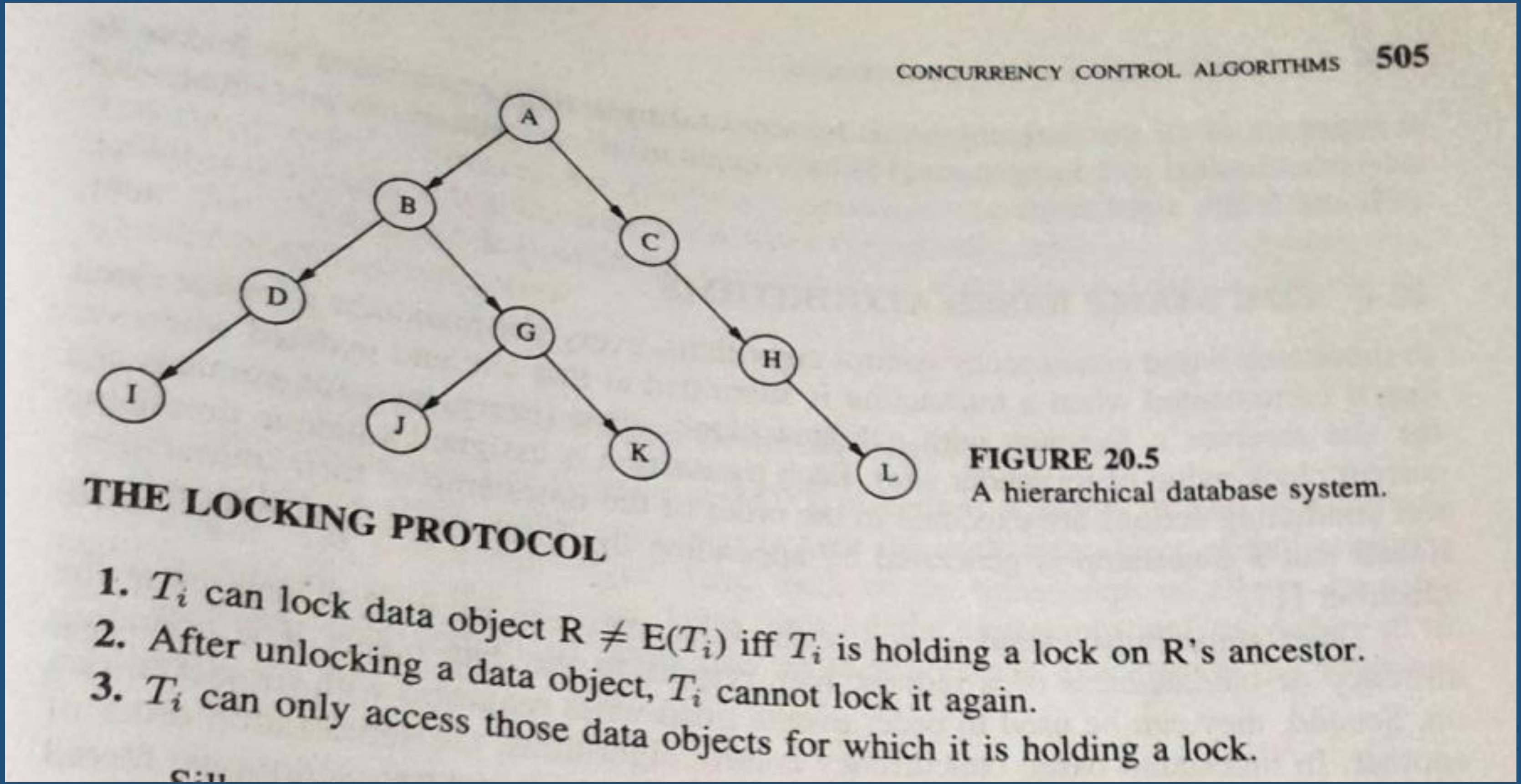
- Number of Restarts:
  - In the WAIT-DIE algorithm, the younger requester dies and is restarted.
  - In the WOUND-DIE algorithm, if the requester is younger, it waits rather than continuously dying and restarting.

Non Two Phase Locking

- When the data objects of a database system are hierarchically organized (hierarchical database systems), a non two phase locking protocol can ensure serializability and freedom from deadlocks.
- In non-two-phase locking, a transaction can request a lock on a data object even after releasing locks on some data objects.
- A data object cannot be locked more than once by the same transaction.
- In order to access a data object, a transaction must first lock it.
- If a transaction attempts to lock a data object that is already locked, the transaction is blocked.
- When a transaction unlocks a data object, one of the transactions waiting for it gets a lock on it and resumes.

Non Two Phase Locking

- When a transaction T_i starts, it selects a data object in the database tree for locking and can be subsequently lock the data objects only in the subtree with root node.



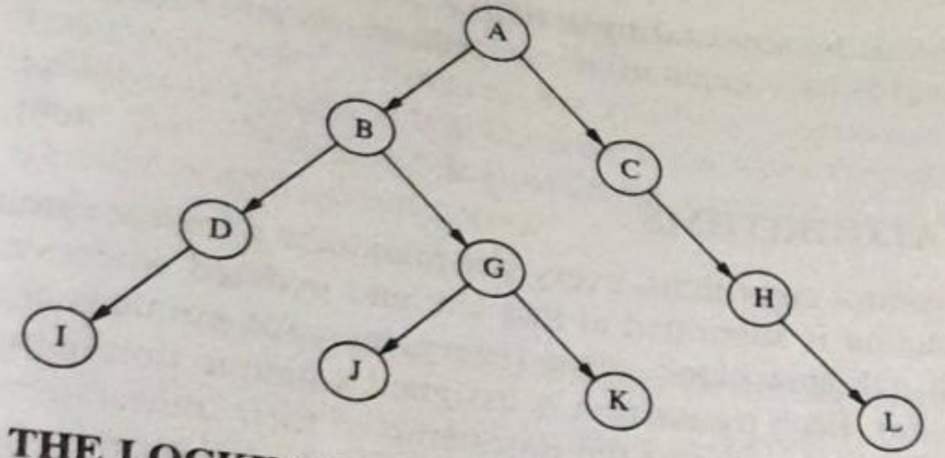


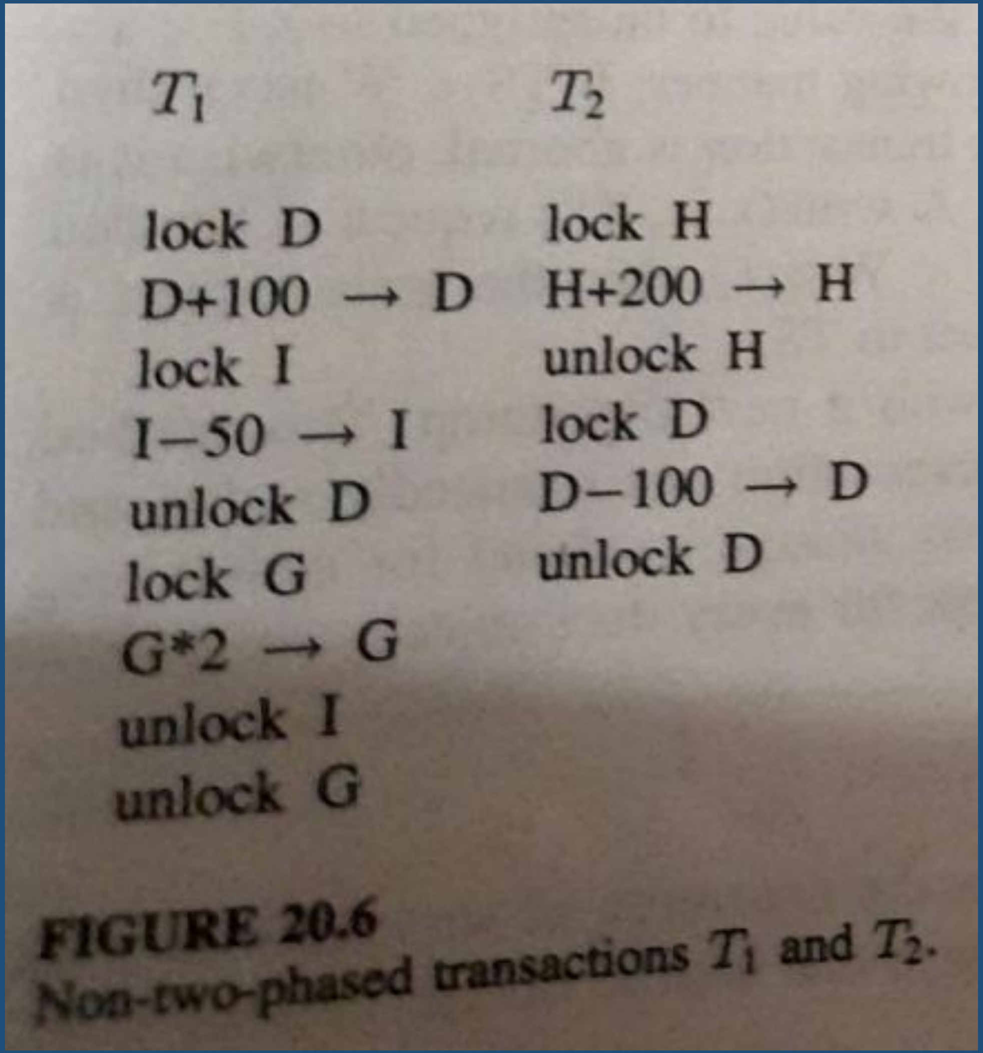
FIGURE 20.5
A hierarchical database system.

THE LOCKING PROTOCOL

1. T_i can lock data object R $\neq$ E(T_i) iff T_i is holding a lock on R's ancestor.
2. After unlocking a data object, T_i cannot lock it again.
3. T_i can only access those data objects for which it is holding a lock.

Non Two Phase Locking

•



T_1	T_2
lock D	lock H
D+100 → D	H+200 → H
lock I	unlock H
I−50 → I	lock D
unlock D	D−100 → D
lock G	unlock D
G*2 → G	
unlock I	
unlock G	

FIGURE 20.6
Non-two-phased transactions T_1 and T_2.

Non Two Phase Locking

- Advantages:
  - It is free from deadlocks.
  - A lock can be released when it is no longer needed.
  - The availability of data objects to other transaction is higher.

Time-Stamped Based Algorithms

Time-Stamped Based Algorithms

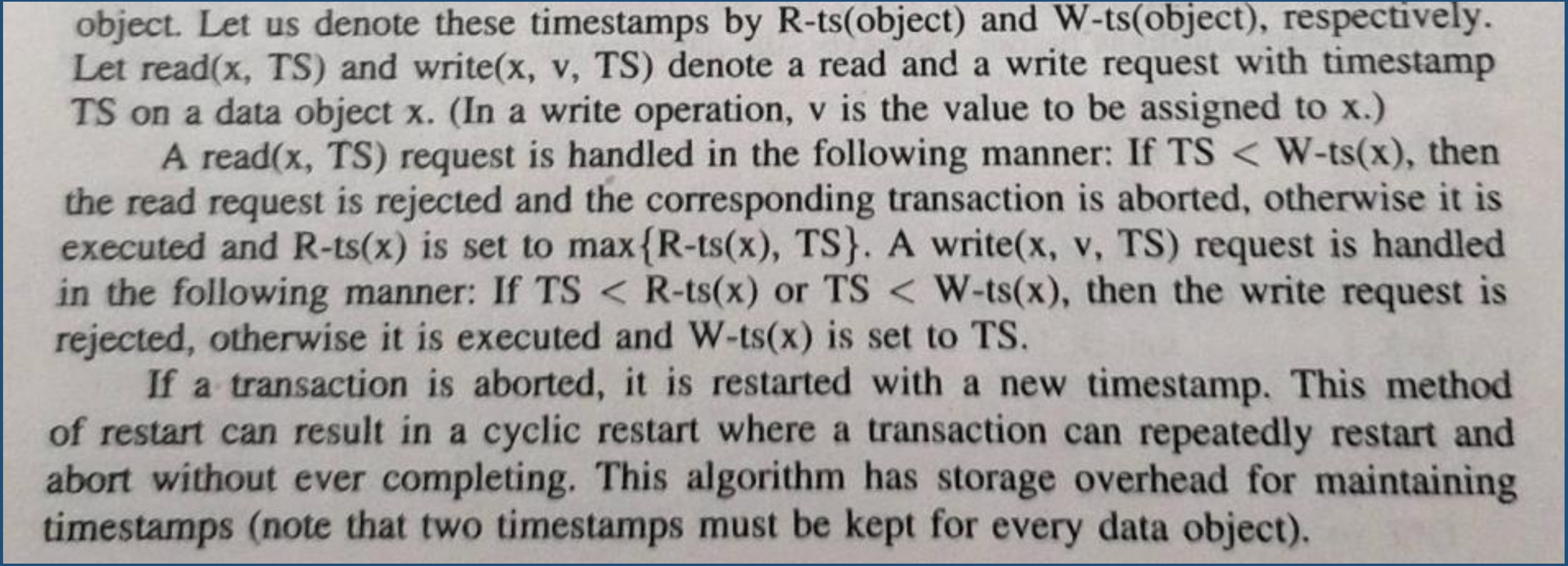
- In timestamp based concurrency control algorithms, every site maintains a logical clock that is incremented when a transition is submitted at that site and updated whenever the site receives a message with a higher clock value.
- Every message contains the current clock value of its sender site.
- Each transaction is assigned a unique timestamp and conflicting actions are executed in the order of the timestamp of their transactions.
- Timestamps can be used in two ways:
  - First, they can be used to determine the outdatedness of a request with respect to the data object it is operating on.
  - Second, they can be used to order events with respect to one another.
- In timestamp based concurrency control algorithms, the serialization order of transactions is selected a priori (decided by their timestamp) and transactions are forced to follow this order.

Time-Stamped Based Algorithms

- Timestamp based concurrency control algorithms are:
    - Basic Timestamp Ordering Algorithms
    - Thomas Write Rule (TWR)
    - Multiversion Timestamp Ordering Algorithm
    - Conservative Timestamp Ordering Algorithm

Basic Timestamp Ordering Algorithms (BTO)

- The scheduler at each DM (Data Manager) keeps track of the largest timestamp of any read and write processed thus far for each data object.



object. Let us denote these timestamps by R-ts(object) and W-ts(object), respectively. Let read(x, TS) and write(x, v, TS) denote a read and a write request with timestamp TS on a data object x. (In a write operation, v is the value to be assigned to x.)

A read(x, TS) request is handled in the following manner: If TS < W-ts(x), then the read request is rejected and the corresponding transaction is aborted, otherwise it is executed and R-ts(x) is set to max{R-ts(x), TS}. A write(x, v, TS) request is handled in the following manner: If TS < R-ts(x) or TS < W-ts(x), then the write request is rejected, otherwise it is executed and W-ts(x) is set to TS.

If a transaction is aborted, it is restarted with a new timestamp. This method of restart can result in a cyclic restart where a transaction can repeatedly restart and abort without ever completing. This algorithm has storage overhead for maintaining timestamps (note that two timestamps must be kept for every data object).

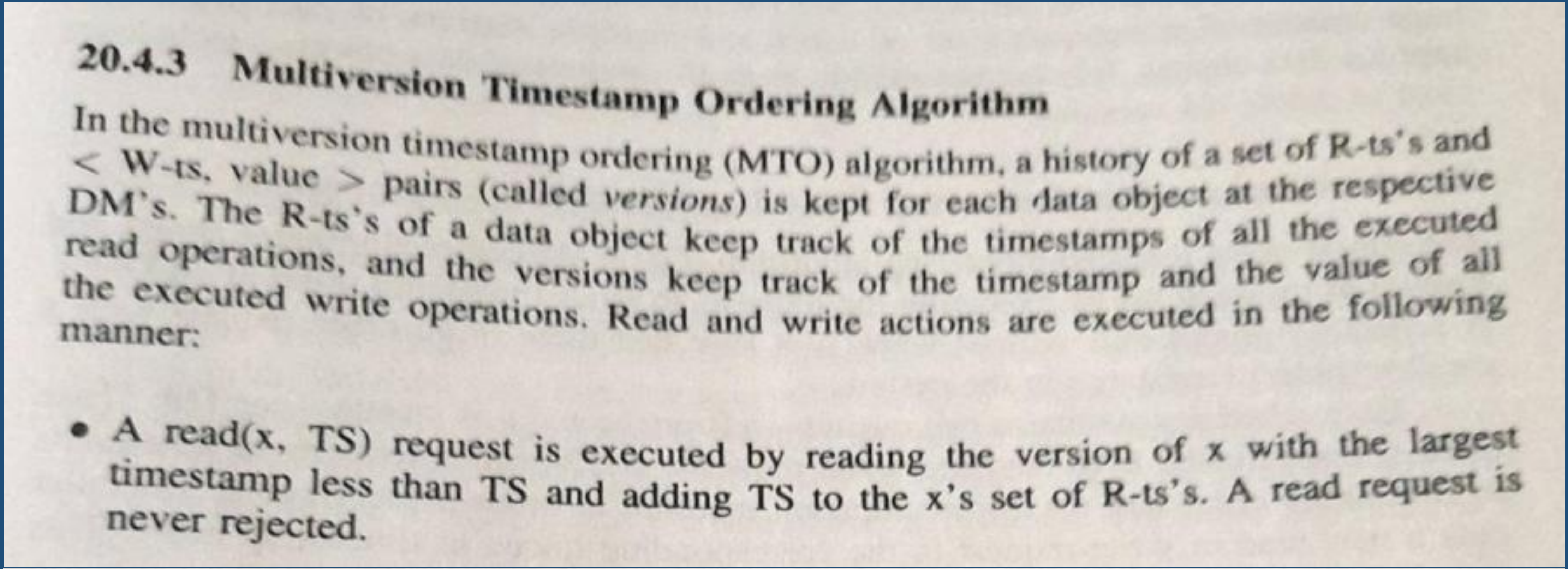
Thomas Write Rule (TWR)

- It states that, **if a more recent transaction has already written the value of an object**, then a less recent transaction does not need perform its own write since it will eventually be overwritten by the more recent one.
- Is suitable only for the execution of write actions.
- For a write(x, v, TS), if TS < W-ts(x), then TWR says that instead of rejecting the write, simply ignore it.
- This is sufficient to enforce synchronization among writes.
- An additional mechanism is needed for synchronization between read and writes because TWR takes care of only write-write synchronization.
- TWR is an improvement over BTO algorithm because it reduces the number of transaction aborts.

Multiversion Timestamp Ordering Algorithm

- A problem with 2PL is that it can lead to deadlocks.
- Multiversion timestamp ordering scheme solves this problem by ordering transactions and aborting transactions that access data out of order.
- It also increases the concurrency in the system by never making an operation block.
- The basic idea in this scheme is to assign transactions timestamps when they are started, which are used to order these transactions.
- If two transactions access data items in an order that is inconsistent with their time stamps, then one of them is aborted.

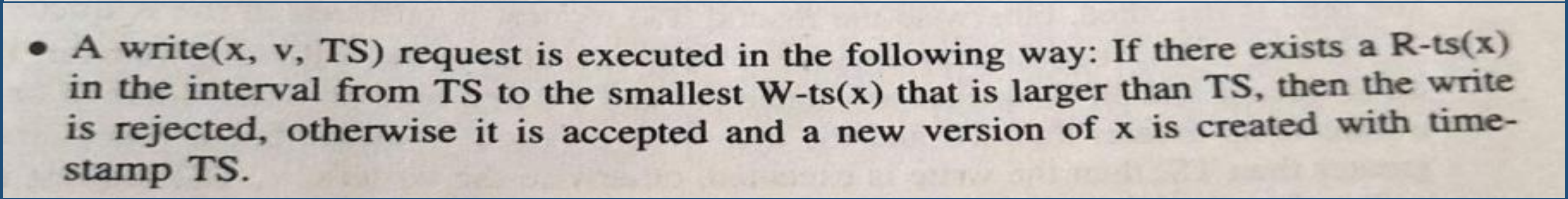
Multiversion Timestamp Ordering Algorithm



20.4.3 Multiversion Timestamp Ordering Algorithm

In the multiversion timestamp ordering (MTO) algorithm, a history of a set of R-ts's and < W-ts, value > pairs (called *versions*) is kept for each data object at the respective DM's. The R-ts's of a data object keep track of the timestamps of all the executed read operations, and the versions keep track of the timestamp and the value of all the executed write operations. Read and write actions are executed in the following manner:

- A read(x, TS) request is executed by reading the version of x with the largest timestamp less than TS and adding TS to the x's set of R-ts's. A read request is never rejected.



- A write(x, v, TS) request is executed in the following way: If there exists a R-ts(x) in the interval from TS to the smallest W-ts(x) that is larger than TS, then the write is rejected, otherwise it is accepted and a new version of x is created with time-stamp TS.

Multiversion Timestamp Ordering Algorithm

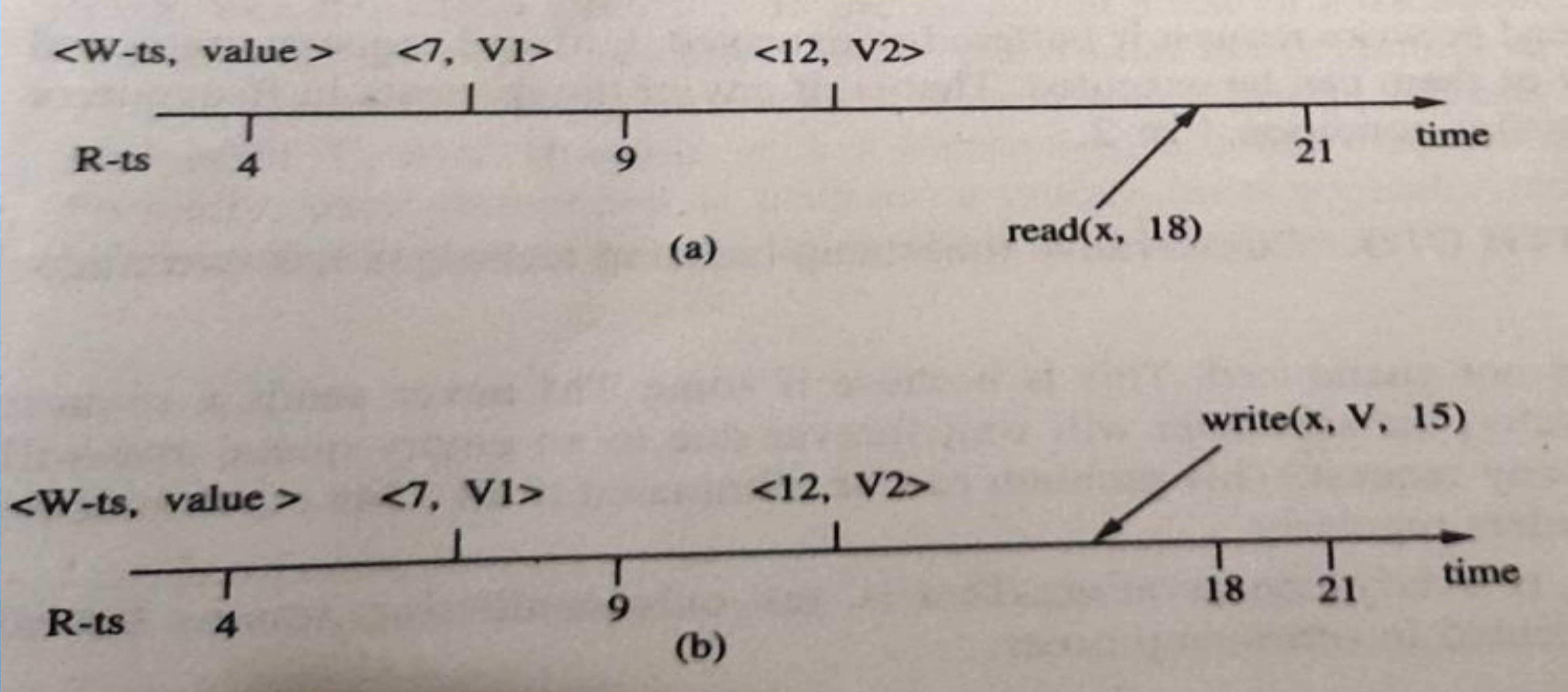
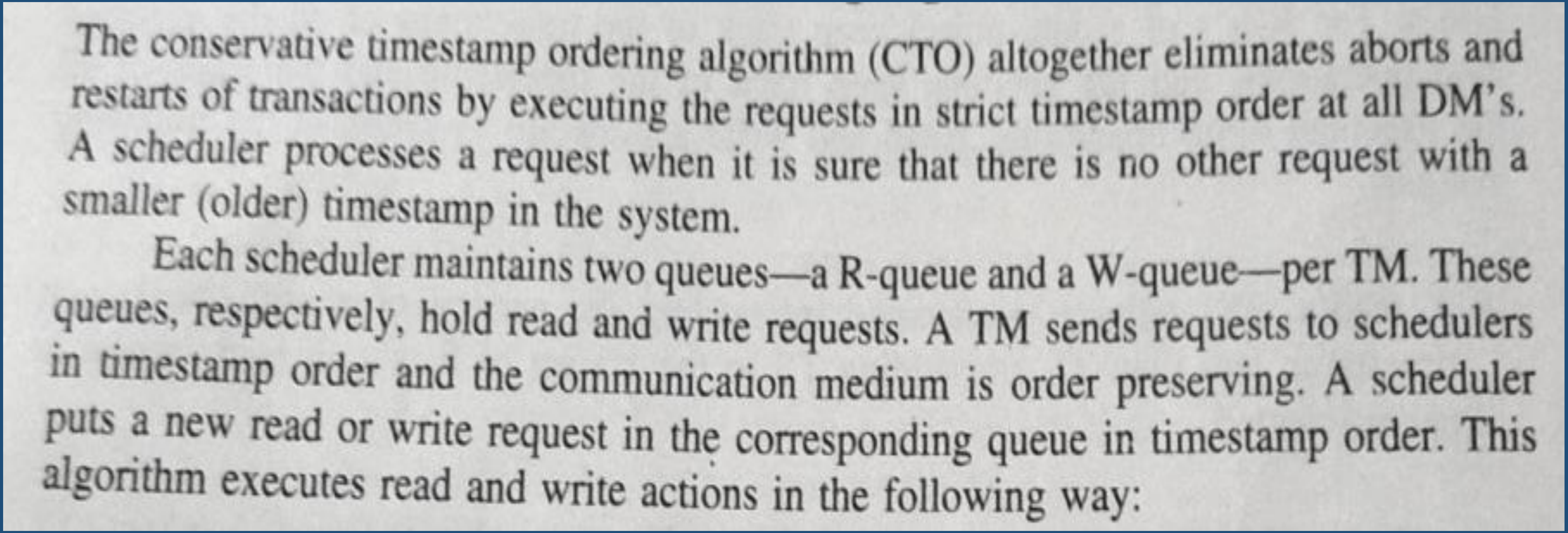


FIGURE 20.7
An example of MTO.

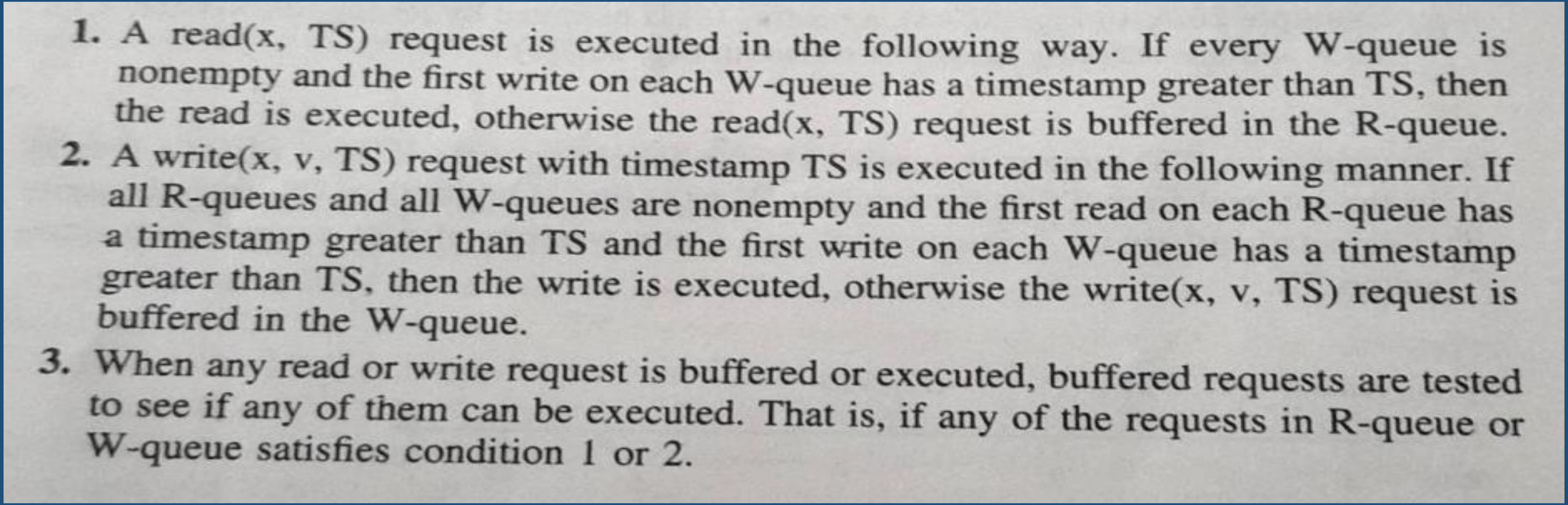
Conservative Timestamp Ordering Algorithm



The conservative timestamp ordering algorithm (CTO) altogether eliminates aborts and restarts of transactions by executing the requests in strict timestamp order at all DM's. A scheduler processes a request when it is sure that there is no other request with a smaller (older) timestamp in the system.

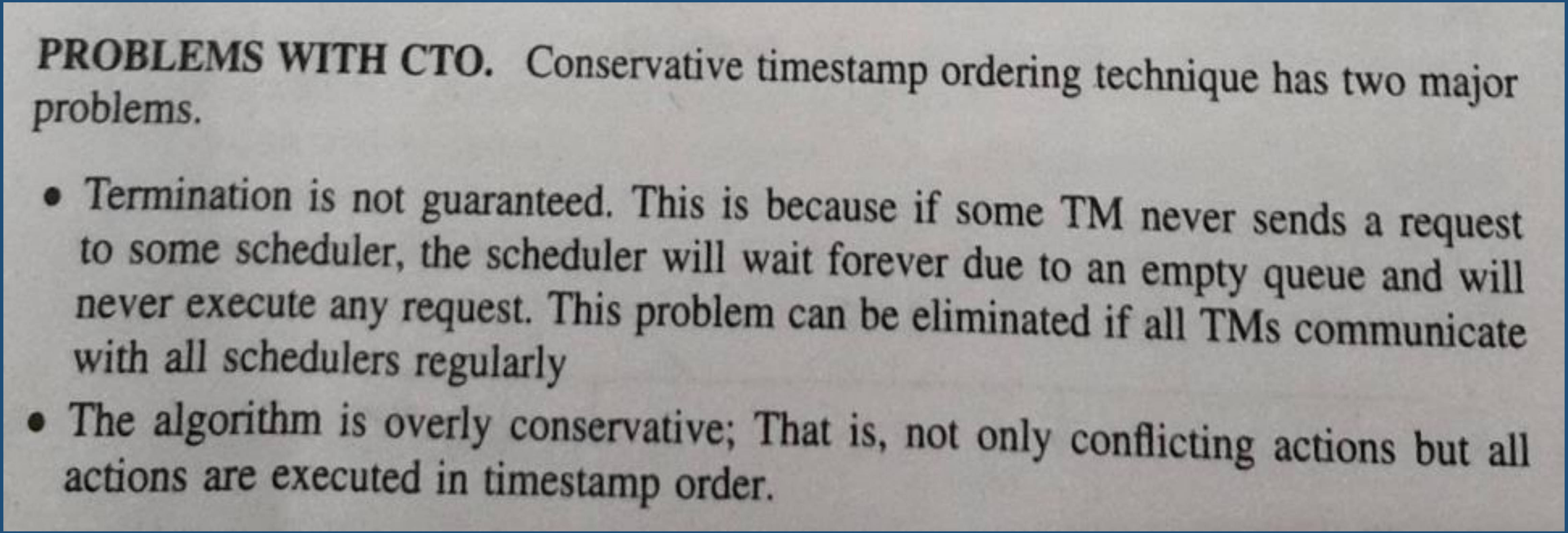
Each scheduler maintains two queues—a R-queue and a W-queue—per TM. These queues, respectively, hold read and write requests. A TM sends requests to schedulers in timestamp order and the communication medium is order preserving. A scheduler puts a new read or write request in the corresponding queue in timestamp order. This algorithm executes read and write actions in the following way:

Conservative Timestamp Ordering Algorithm



1. A read(x, TS) request is executed in the following way. If every W-queue is nonempty and the first write on each W-queue has a timestamp greater than TS, then the read is executed, otherwise the read(x, TS) request is buffered in the R-queue.
2. A write(x, v, TS) request with timestamp TS is executed in the following manner. If all R-queues and all W-queues are nonempty and the first read on each R-queue has a timestamp greater than TS and the first write on each W-queue has a timestamp greater than TS, then the write is executed, otherwise the write(x, v, TS) request is buffered in the W-queue.
3. When any read or write request is buffered or executed, buffered requests are tested to see if any of them can be executed. That is, if any of the requests in R-queue or W-queue satisfies condition 1 or 2.

Conservative Timestamp Ordering Algorithm



PROBLEMS WITH CTO. Conservative timestamp ordering technique has two major problems.

- Termination is not guaranteed. This is because if some TM never sends a request to some scheduler, the scheduler will wait forever due to an empty queue and will never execute any request. This problem can be eliminated if all TMs communicate with all schedulers regularly
- The algorithm is overly conservative; That is, not only conflicting actions but all actions are executed in timestamp order.

Optimistic Algorithms

Optimistic Algorithms

- Optimistic Concurrency Control Algorithms are based on the assumption that conflicts do not occur during execution time.
- No synchonization is performed when a transaction is executed.
- However, a check is performed at the end of the transaction to make sure that no conflicts have occured.
- If there is a conflict, the transaction will be aborted.
- Otherwise, the transaction is commited. Since conflicts do not occur very often, this algoritm is very efficient compared to other locking algorithms.

Optimistic Algorithms

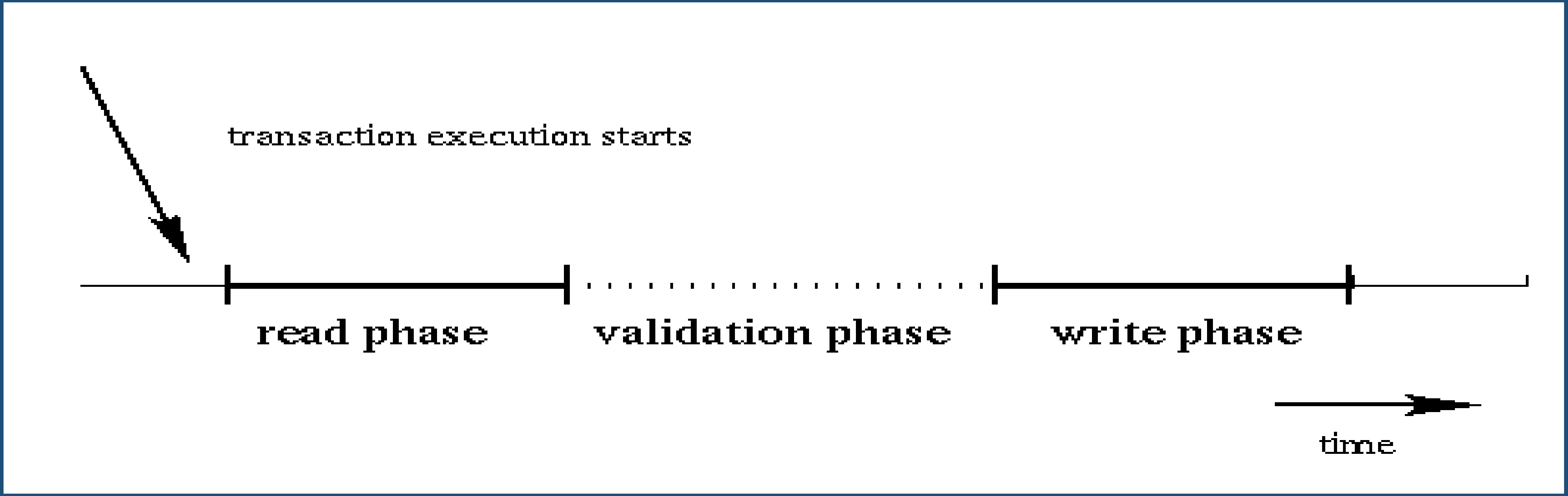
Kung-Robinson Algorithm

- Kung and Robinson were the first to propose an optimistic method for concurrency control.
- The optimistic situation for this algorithm happens when
  - conflicts are unlikely to happen
  - the system consists mainly read-only transactions (such as a query dominant system)
- Basic Idea: No synchonization check is performed during transaction processing time, however, a validation is performed to make sure there is no conflicts occurred. If a conflict is found, the tentative write is discarded and the transaction is restarted.

Optimistic Algorithms

The algorithm

- Divide the execution of transaction into three phases:



Optimistic Algorithms

- **Read phase:** data objects are read, the intended computation of the transaction is done, and writes are made on a **temporary** storage.

- **Validation phase:** check to see if writes made by the transaction violate the consistency of the database. If the check finds out any conflicts, the data in the temporary storage will be discarded. Otherwise, the write phase will write the data into the **database**.

- **Write phase:** If the validation phase passes ok, write will be performed to the database. If the validation phase fails to pass, all temporary written data will be aborted.

Thank You