Transaction Management/ACID in DBMS

Dr. Munesh Singh

Indian Institute of Information Technology
Design and Manufacturing,
Kancheepuram
Chennai-600127

March 25, 2019

Transaction Management

How transcation happens?

- A transaction is a program unit whose execution may change the contents of a database
- **Database before transaction**
- **Database after transaction**
- Transaction is used to represent a logical unit of database processing that must be completed in its entirely to ensure correctness
- Transaction is a collection of **Read**,**Update**,**Write**.

ACID Properties of Transaction

Atomicity

- **Atomicity (ALL or NONE)** Ensure that a transaction will run to completion as an indivisible unit. At the end no change occur to database or database change should be consistent manner.
- Example: Given A=2000, B=3000, initial=5000
  Rs=500/- from A− >B
  T1
  R(A,a)
  a-500
  W(A,a)< −

ACID Properties of Transaction

Atomicity

- **Atomicity (ALL or NONE)** Ensure that a transaction will run to completion as an indivisible unit. At the end no change occur to database or database change should be consistent manner.
- Example: Given A=2000, B=3000, initial=5000
  Rs=500/- from A− >B
  T1
  R(A,a)
  a-500
  W(A,a)< − Power failure A=1500, B=3000, final=4500
  Read(B,b)
  b=b+500
  W(B,b)

ACID Properties of Transaction

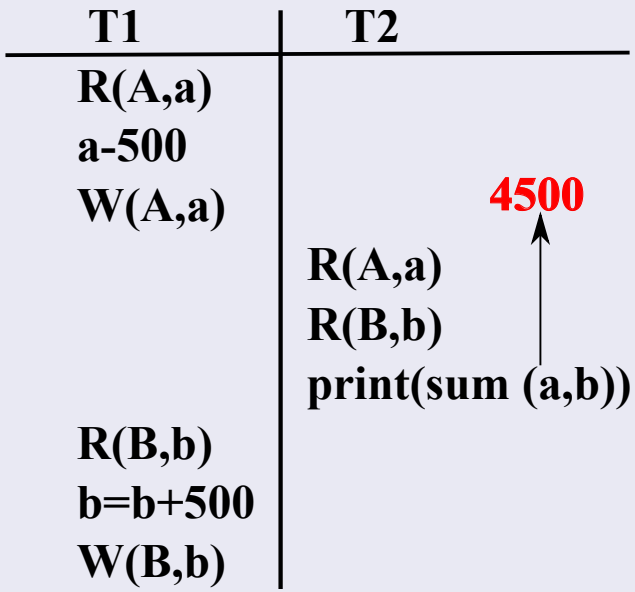
Consistency

- **Consistency: (correctness)** Ensure that if the database was in a consistent state before the start of a transaction, then or termination, the database will also be in a consistent state.
- Example: Sum(A,B) before transaction = Sum(A,B) after transaction

ACID Properties of Transaction

Isolation

- **Isolation** Indicate that the actions performed by a transaction will be isolated or hidden from outside the transaction until it terminates
- Example: Given A=2000, B=3000, initial=5000 Rs=500/- from A− >B

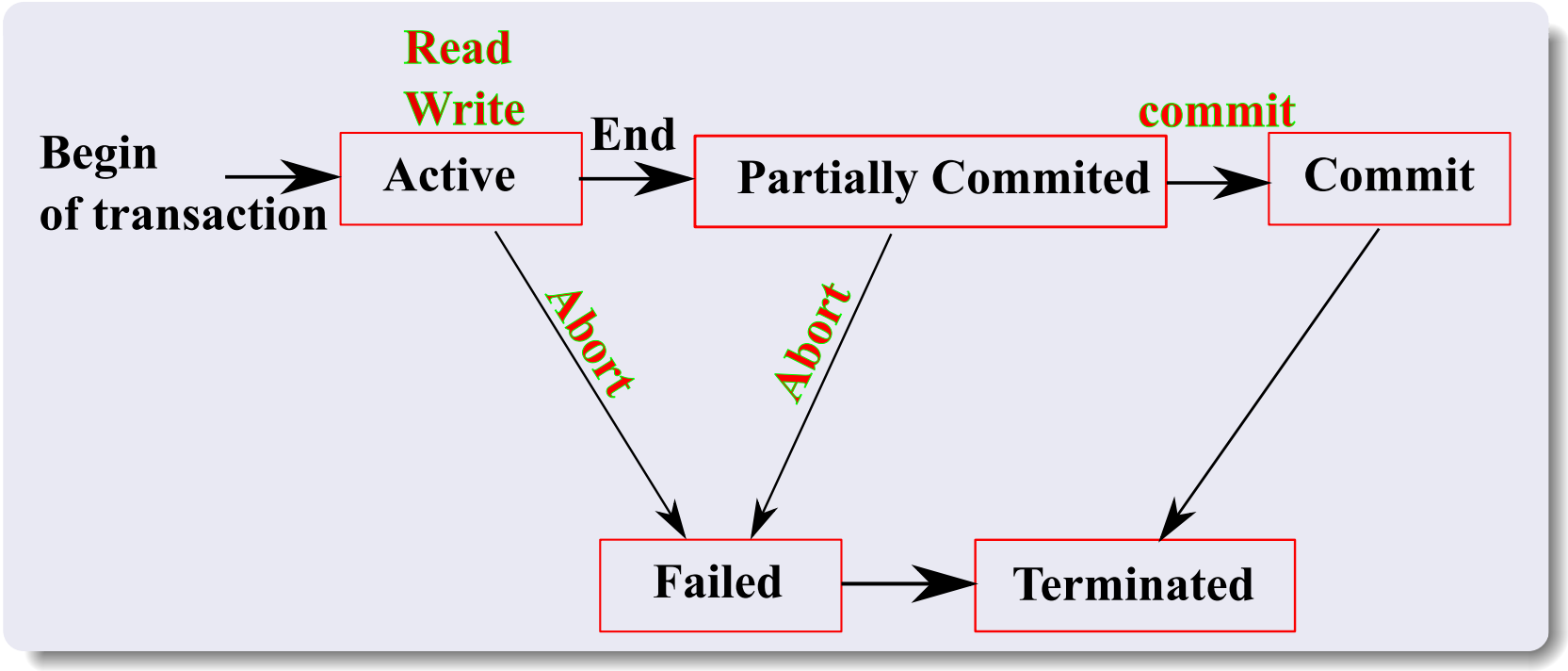


ACID Properties of Transaction

Durability

- **Durability** All updates done by a transaction must become permanent
- Ensure that the commit action of a transaction on its termination will be reflected in the database

State of a Transaction



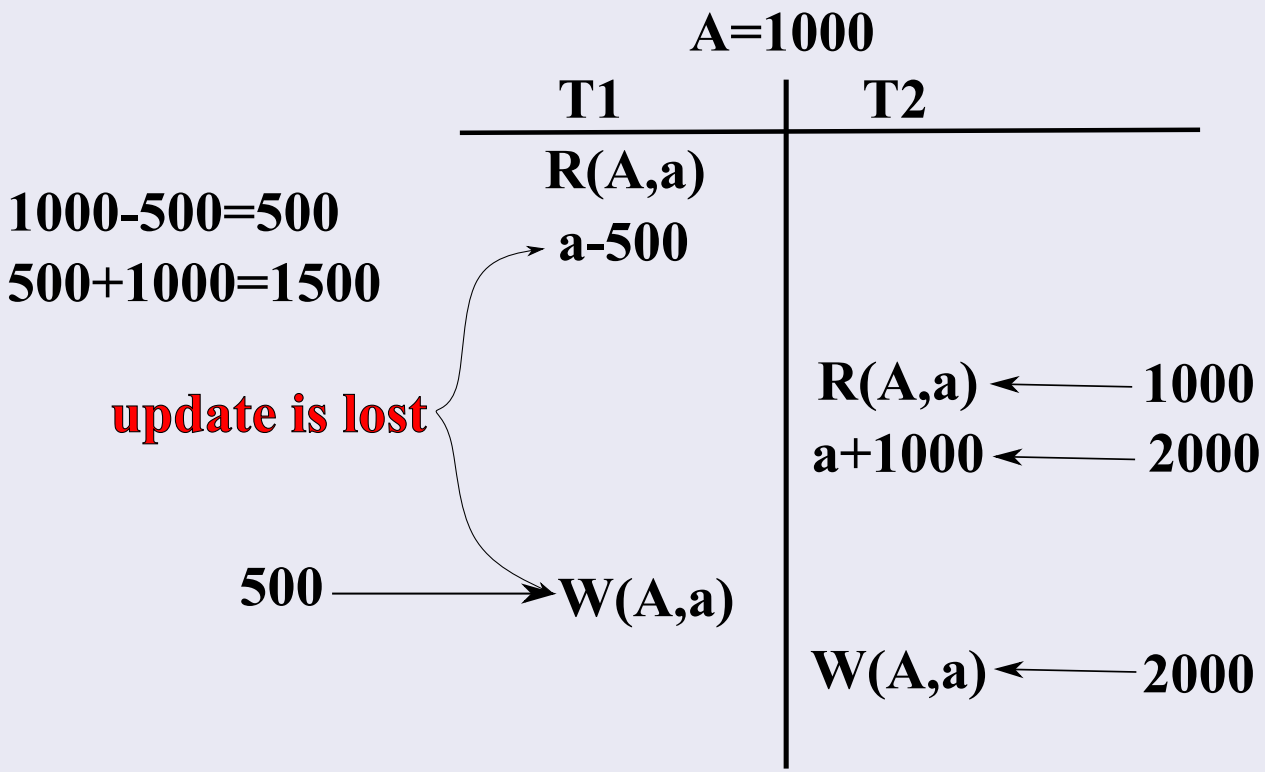
Concurrent Execustion

- It implies interleaving execution of operations of a transaction
- Benefits
  - Helps in reducing waiting time
  - Improve throughput and resource utilization
- **Schedule** It represents the order in which instructions of a transactions are executed

Problem with Concurrent Execution

Lost update Problem (W-W Conflict)

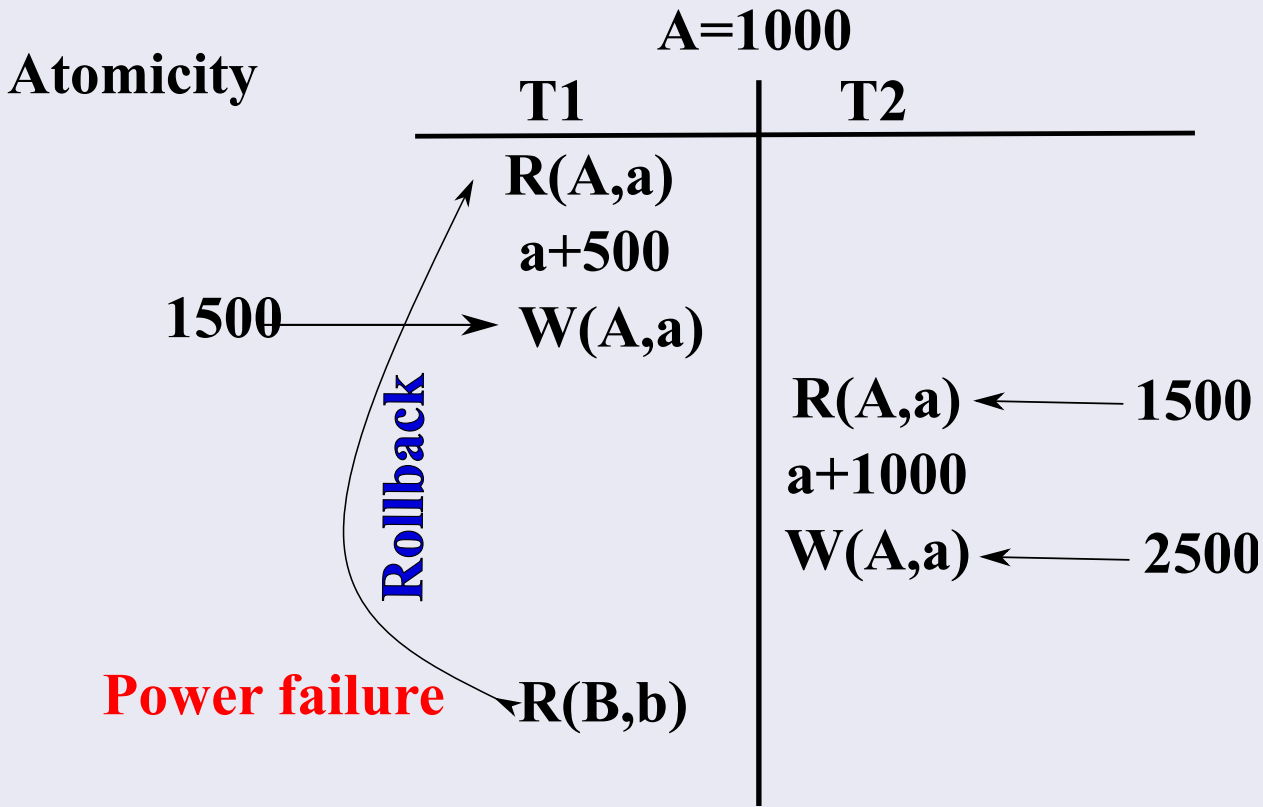
- Occurs when two transactions accesses the same database item have their operations interleaved in a way that makes the value of the database item incorrect.



Problem with Concurrent Execution

Temporary update (dirty read) Problem (W-R Conflict)

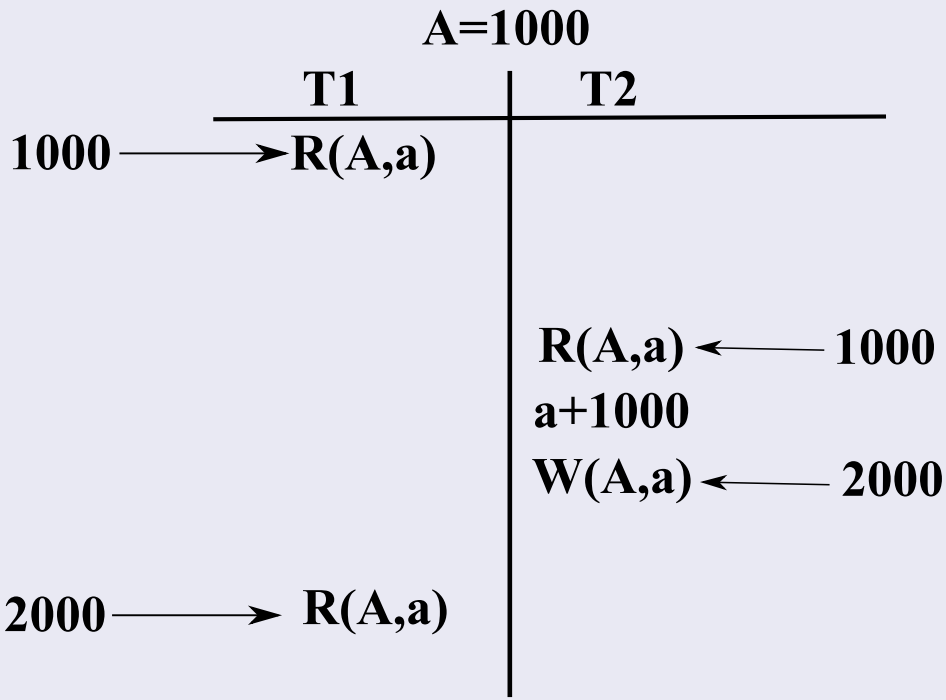
- Occurs when one transaction updates a database item and then the transaction fails, bit its update is read by some other transaction.



Problem with Concurrent Execution

Unrepeatable Read (W-R Conflict)

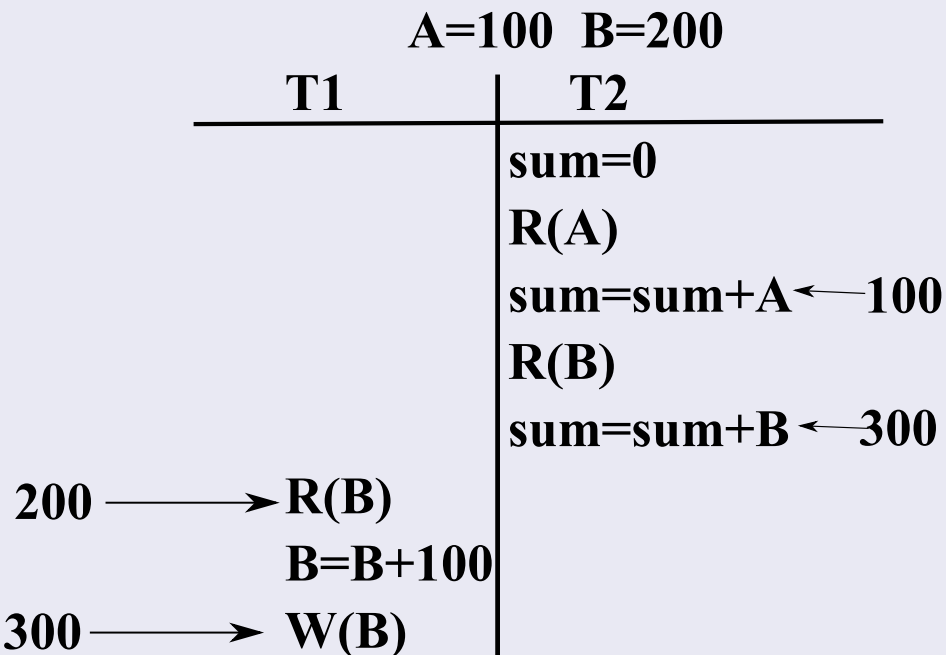
- If a transaction 'Ti' reads an item value twice and the item is changed by another transaction 'Tj' in between the two read operation. Hence 'Ti' receives different values for its two read operation of the same item



Problem with Concurrent Execution

Incorrect Summary Problem:

- If one transaction is calculating an aggregate summary function on a no of records, while other transaction is updating some of these records, the aggregate function may calculate some values before they are updated and other after they are updated results in incorrect summary.



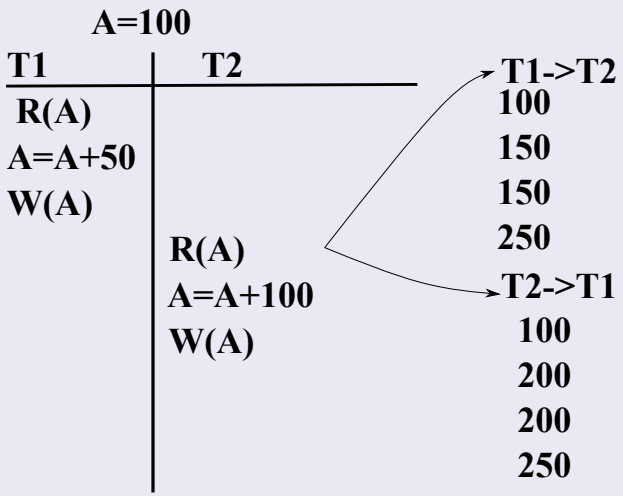
Schedules:

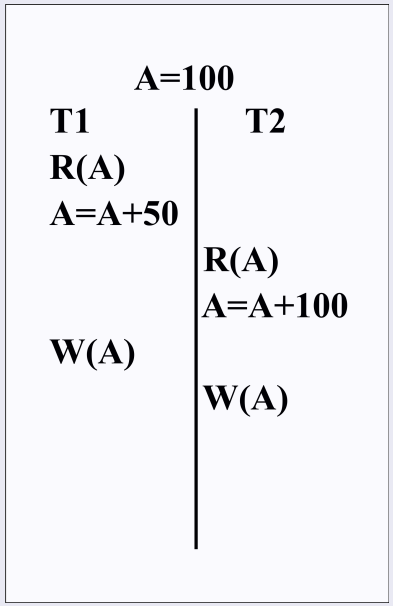
- A schedule 'S' of n transaction 'T1,T2,....Tn' is an ordering of operations of the transactions in chronological order
- When several transactions are executing concurrently then the order of execution of various instructions is known as schedule

Schedules:

Types of schedule

- **Serial Schedule:** Does not interleave the actions of any operations of different transactions.
- Always ensure a consistent state
  $T1 -> T2 -> T3$
  $T1 -> T3 -> T2$

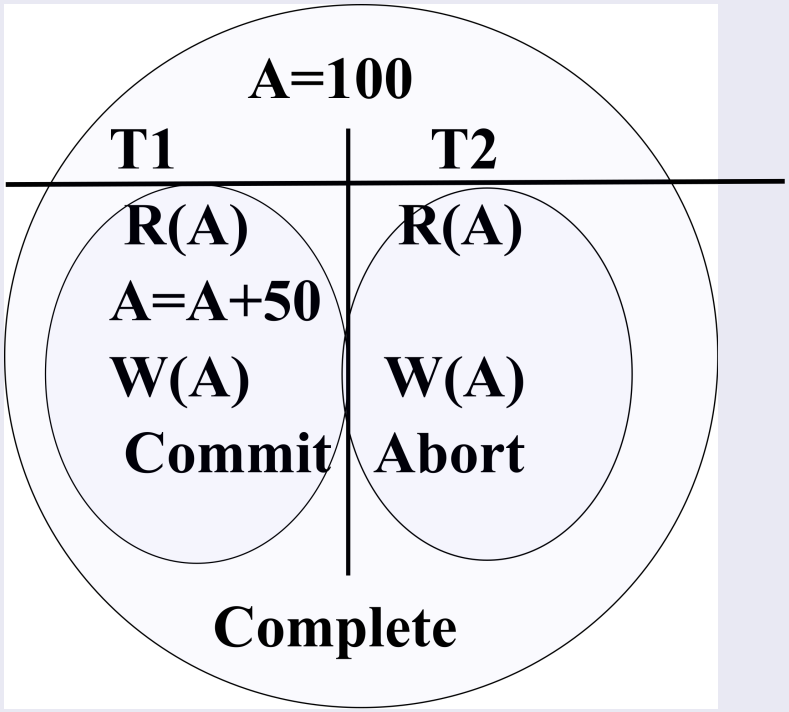




Schedules:

Types of schedule

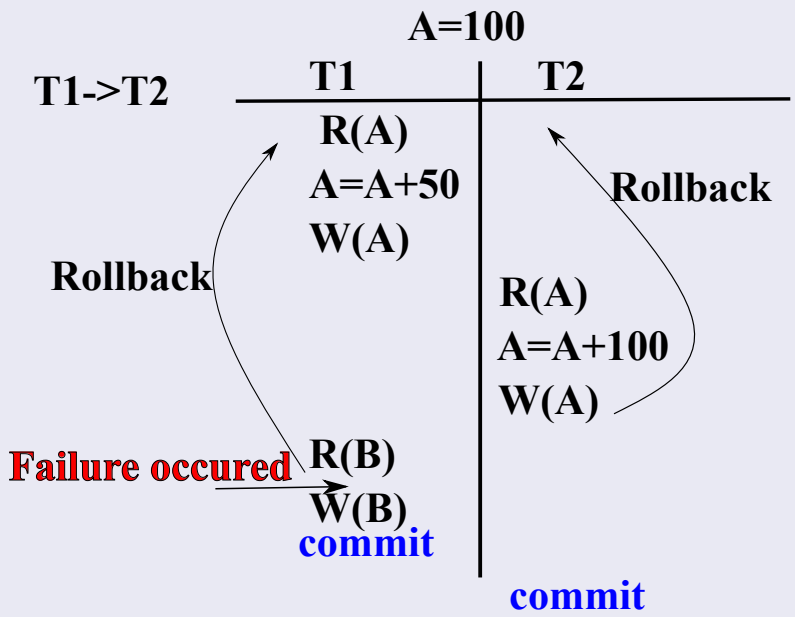
- **Complete Schedule:** If the last operation of each transaction is either abort or commit.



Schedules:

Types of schedule

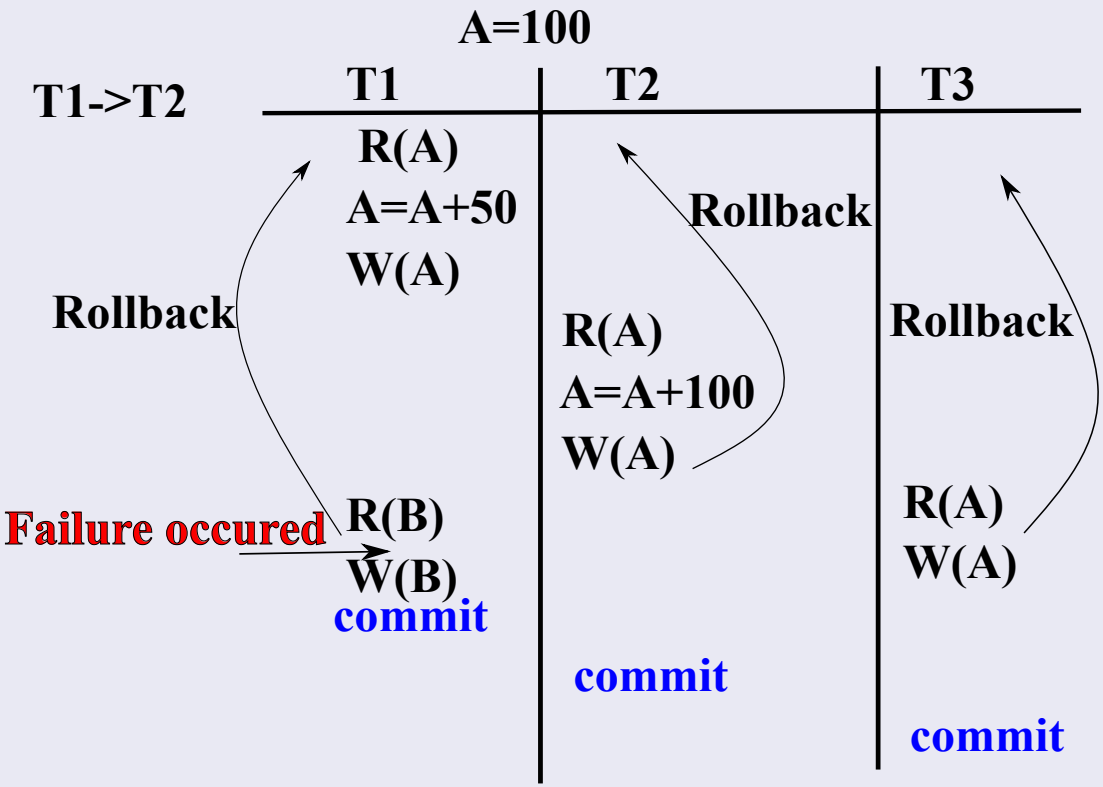
- **Recoverable Schedule:** Is one where for each pair of transactions (Ti,Tj), such that Tj reads a data item that was previously written by Ti, then the commit operation of Ti should appear before commit operation of Tj.



Schedules:

Types of schedule

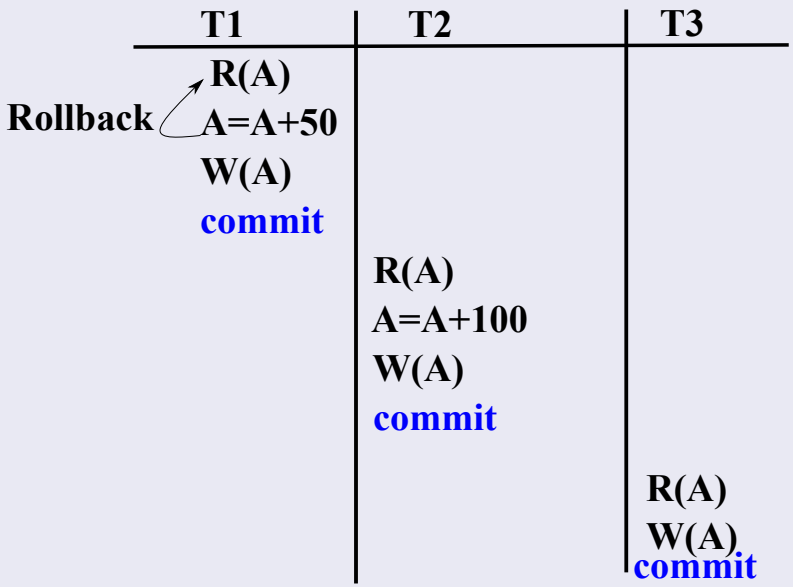
- **Cascading rollback Schedule:**



Schedules:

Types of schedule

- **Cascadingless Schedule:** Is one where for each pair of transaction(Ti,Tj) such that Tj read a data item written by Ti, then the commit operation of Ti should appear the read operation of Tj.



Schedules:

Types of schedule

- **Strict Schedule:** If a value written by a transaction cannot be read or overwritten by another transaction until the transaction is either aborted or committed
- Every strict schedule is both Recoverable and Cascade-less

T1	T2
R(A)	
A=A+50	
W(A)	
commit	
	R(A)
	A=A+100
	W(A)
	commit

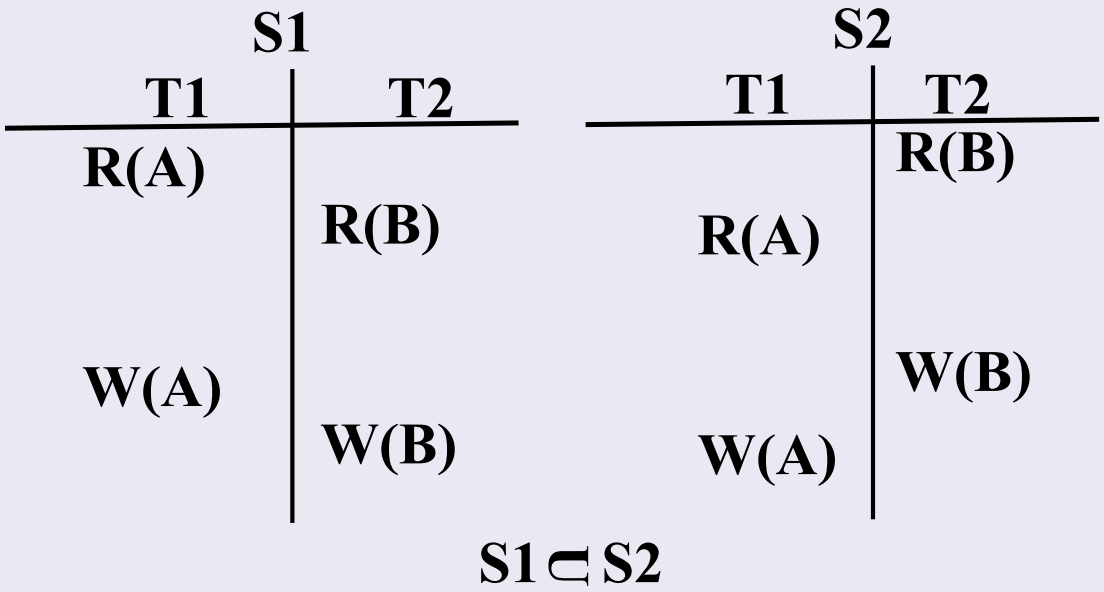
Conflict Transactions

Conflict Operations

- Operations are said to be conflicting if:
  - Belong to different transactions.
  - Access to same database item 'A'
  - At-least one of them is a write operation.(R-W)(W-R)(W-W)
- **Equivalent Schedule** Two schedules 'S1' and 'S2' are said to be equivalent schedule if they produce the same final database state.
- **Result Equivalent Schedule:-** Produce same final database state for same initial value of data.

Conflict Equivalent

- Two schedules are said to be conflict equivalent if all conflicting operations in both the schedule must be executed in the same order
- **Question 1** Check for conflict Equivalent
  - S1: R1(A), R2(B), W1(A),W2(B)
  - S2: R2(B), R1(A), W2(B),W1(A)

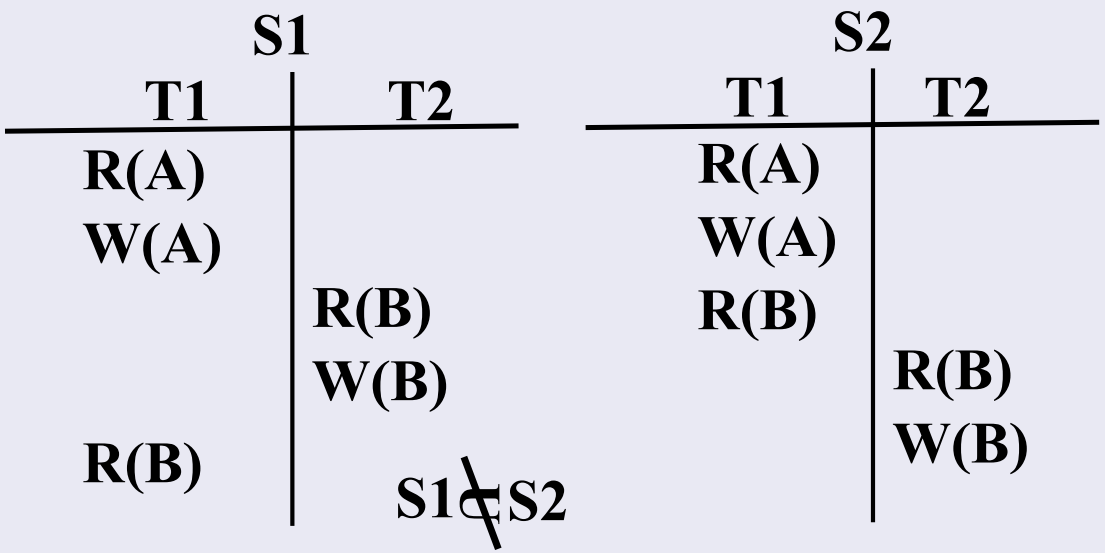


Conflict Equivalent

- **Question 2** Check for conflict Equivalent
  - S1: R1(A),W1(A) R2(B),W2(B),R1(B)
  - S2: R1(A), W1(A), R1(B),R2(B),W2(B)

Conflict Equivalent

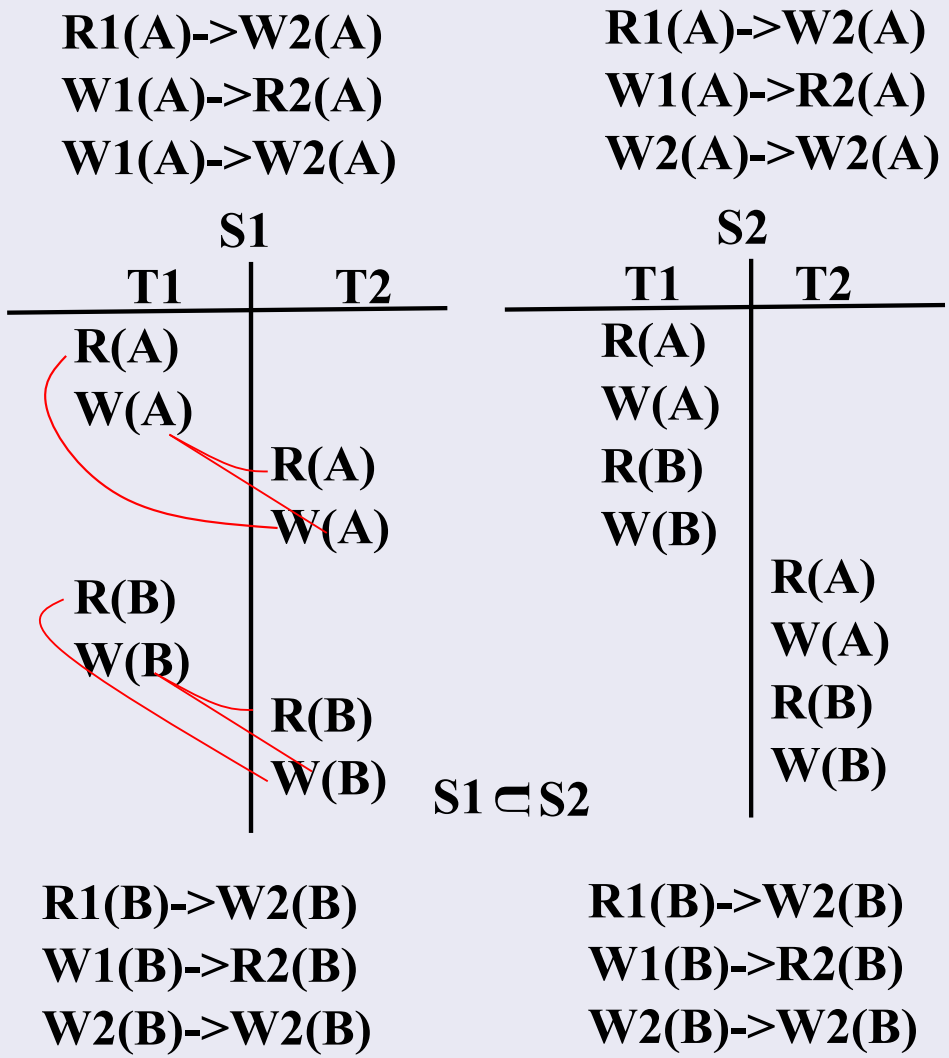
- **Question 2** Check for conflict Equivalent
  - S1: R1(A),W1(A) R2(B),W2(B),R1(B)
  - S2: R1(A), W1(A), R1(B),R2(B),W2(B)



Serializability

- A Schedule 'S' of 'n' transactions is serializable if it is equivalent to some serial schedule of the same 'n' transactions.
- **Conflict Serializable** If it is conflict equivalent to serial schedule

Serializability Example



Serializability Gate Example

- S1: R1(X) R1(Y) R2(X) R2(Y) W2(Y) W1(X)
- S2: R1(X) R2(X) R2(Y) W2(Y) R1(Y) W1(X)
  1. Both S1 and S2 are Conflict Serializable
  2. Only S1 is Conflict Serializable
  3. Only S2 is conflict Serializable
  4. None

Test for Conflict Serializability

Precedence Graph is Used

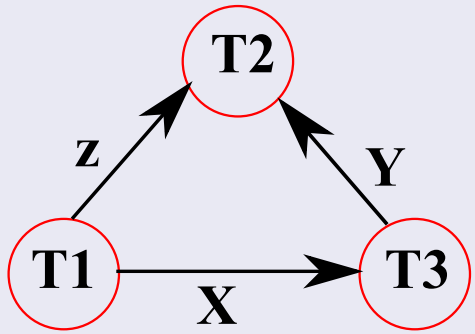
- Let 'S' be a schedule, construct a directed graph known as precedence graph
- Graph consist of a pair of G=(V,E) where
  - V: is a set of vertices
  - E: set of edges
- **Algorithm for creation of graph**
  - Create a node for each transaction
  - A directed edge, Ti− >Tj, if Ti reads a value of an item written by Ti.
  - Directed edge Ti− >Tj, if Tj writes a value into item after it has been read by Ti.
  - Directed edge, Ti− >Tj, if Tj Write after Ti Write.
- A schedule is conflict serializable if an only if precedence graph is acyclic

Test for Conflict Serializability

Conflict Serializability

- **Question** Check for conflict Serializability

T1	T2	T3
R(X)		
	R(Z)	
R(Z)		R(X)
		R(Y)
		W(X)
	R(Y)	
	W(Z)	
	W(Y)	

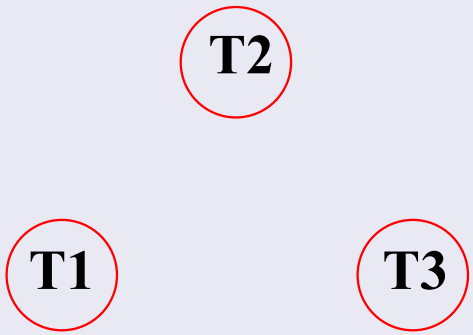


Test for Conflict Serializability

Conflict Serializability

- **Question** Check for conflict Serializability

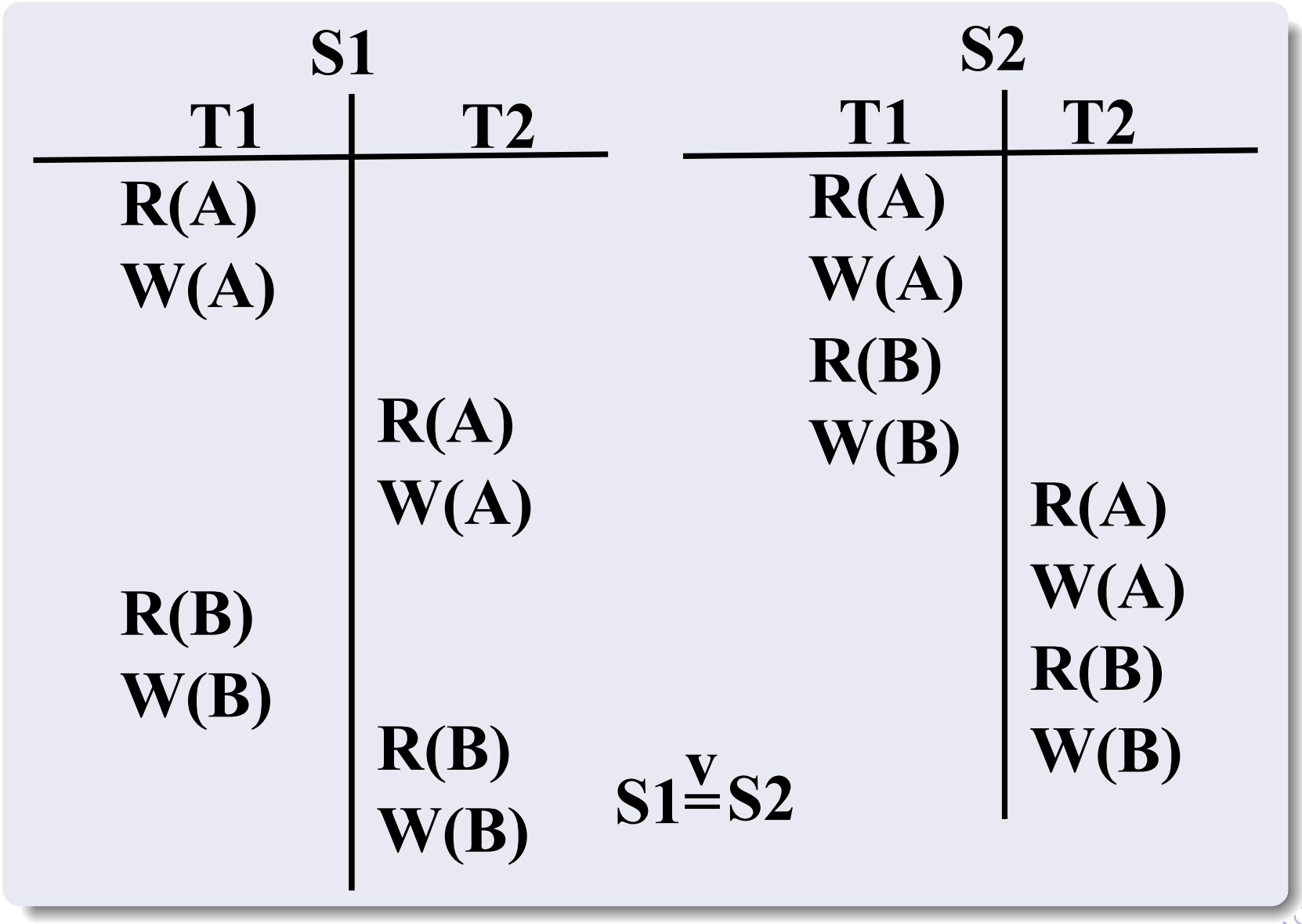
T1	T2	T3
R(X)		
		R(X)
		W(X)
W(x)		
	R(X)	



View Serializability

- Two schedules 'S' and 'S" are view equivalent if the following conditions are met:
  1. For each data item Q, if Ti reads an initial value of in schedule S, then Ti in S' also reads an initial value of Q.
  2. If Ti executes Reads Q in S, and that value was produced by Ti (if any), then Ti must in schedule S' also reads the value of Q that was produced by Tj.
  3. For each data item Q, the transaction that perform the final write(Q) operation in schedule S must also perform the final write (Q) in schedule S'.
- A schedule is view serializable if it is view equivalent to a serial schedule.
- **Note:** Every conflict serializable schedule is also view serializable but not vice-versa,

View Serializability Example



Concurrency Control

- It is the process of managing simultaneous execution of transactions in a shared database, to ensure the serializability of transactions.
- **Purpose of concurrency control**
  1. To enforce isolation
  2. To preserve database consistency
  3. To resolve read-write and write-write conflicts

Concurrency Control Techniques

- **Lock-Based Protocol:** A lock gurantees exclusive use of a data item to a current transaction.
  1. To Access Data item (Lock Acquired)
  2. After Completion of transaction (Release the Lock)
  3. All data items must be accessed in a mutually exclusive manner
- **Types of Locks:**
  1. Shared Lock (Lock-S): It is used for Read data item value
  2. Exclusive Lock (Lock-X) It is used for Both Read and Write

Compatibility B/W Lock Model

	LockS	LockX
LockS	✓	X
LockX	X	X

T1	T2
Lock-X(B)	
Read(B)	
B+50	
Write(B)	
Unlock-X(B)	
	Lock-S(B)
	Read(B)
	Unlock-S(B)

Note: Any Number of Transaction can Hold Shared Lock,but Exclusive Lock only by One tranasaction at a time

Two-Phase Locking Protocol

- Requires both locks and unlocks being done in two phases.
- **Phases**
  - Growing (Expanding) Phase: New Locks on items can be acquired
  - Shrinking phase: Existing locks releases, but no new lock can be acquired
- As soon as final lock as been acquired it goes into Lock Point
- Once the Lock point is acquired, it can goes to shrinking phase

Two-Phase Locking Protocol

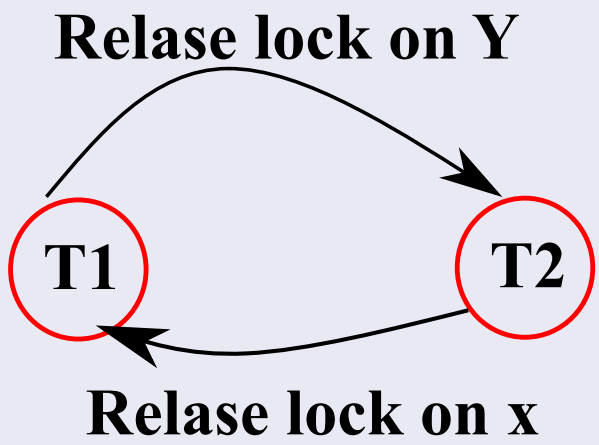
- 2PL protocol enforces Serializability but may reduce concurrency due to the following reasons
  1. Holding Lock UN-necessarily
  2. Locking too early
  3. Penalty to other transaction

Variations of 2PL Locking Protocol

1. **Conservative (static) 2PL**: Acquire all lock before it starts. Release all locks after commit E.g: Avoid cascading rollback, Deadlock free
2. **Strict 2PL**: Exclusive lock can't be released until commit. Helps in cascade-less schedule
3. **Rigrous 2PL** Shared/Exclusive can't be released until commit. Avoid cascading rollback
4. However, the strict 2PL and Rigrous 2PL may lead to Deadlock

Deadlock

- A system is in a deadlock state if there exists a set of transactions such that every transaction in the set is waiting for another transaction in the set

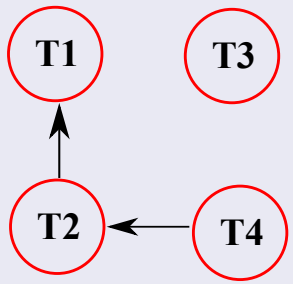


(x,y)	(x,y)
Lock(x)	Lock(y)
x+y	y-x

Deadlock Detection

- Simple way to detect a state of deadlock is to draw wait-for graph
- G(V,E), V= Nodes describing transaction, E=Directed Edge
- Ti− >Tj (directied edge)Ti is waiting for a resource data item held by Tj

Transactions	Data items	Lock Mode
T1	Q	Shared
T2	P Q	Exclusive Exclusive
T3	Q	Shared
T4	P	Exclusive



If cycle not occur,it means deadlock free

Techniques to Control Deadlock

Deadlock Prevention

- This protocol ensure that the system will never enter into a deadlock state
  1. Mutual Exclusion
  2. Hold and Wait
  3. No preemption
  4. Circular wait

Other Techniques in Deadlock Prevention

Use of timestamps

- Assuming that Ti request a data item currently held by Tj
  1. **Wait-Die scheme**
     if Ts(Ti)<Ts(Tj) [Ti is older than Tj]
     Ti is allowed to wait otherwise if Ti is younger than Tj then abort Ti (Ti dies) and restart it later with same timestamps.
  2. **Wound-wait scheme**
     if Ts(Ti)<Ts(Tj)[Ti is older than Tj]
     Then abort Tj(Ti wounds Tj) and restart it with same timestamps. Otherwise Ts(Ti¿Ts(Tj))− > Ti is allowed to wait

Other Techniques in Deadlock Prevention

Use of timestamps

- Assuming that Ti request a data item currently held by Tj
  1. **Time-Out Based Scheme**
     Based on Lock-timeouts, A transaction that has requested a Lock waits for at most a specified amount of time. If the lock is not granted within that time, transaction is said to timeout and it rolls itself back and restarts.

Starvation

- A transaction is starved if it cannot process for am indefinite period of time while other transactions in the system continue normally
- Other reason of starvation: Algorithm dealing with deadlock prevention select some transaction as victim repeatedly, thus causing it abort and never finish excursion
- To prevent starvation:
  1. Modify timestamps
  2. Increase the priority of transaction

Deadlock Recovery

- Selection of victim T1$<->$T2 (abort T1) selection of victim based on minimum cost
  1. Length of transaction(younger))
  2. Data item used by transaction [less no of data items])
  3. Data items that are to be locked[More data items to lock]
  4. How many transaction to be rollback [min of rollback]
- Rollback
  1. Full rollback [starting points]
  2. Partial rollback [saved points, locked point]
- Starvation: [Take care while selecting a victim so that a victim does not get starved]

Deadlock Recovery

Shadow Paging

- Requires fewer disk access than do-log methods.
- Maintain two page Tables during the life cycle of transaction
- When transaction start both page tables are identical
- Shadow page table is never changed over duration of transaction
- Current page table may change during Write operation
- All input and output operations use the current page table to lock database on disk
- Store shadow page table in Nonvolatile storage
- **When transaction commit system write current page table to nonvolatile storage. The current page table then becomes new shadowed P.T**

Deadlock Recovery

Shadow Paging

- Advantages
  1. Log-records overhead is removed
  2. Faster Recovery
- Drawback of shadow paging
  1. Commit overhead
  2. Data fragmentation
  3. Garbage collection

Log Based Recovery

- Log is the most commonly used structure for recording the database modification
- Update log has following fields:-
  1. Transaction identifier
  2. Data item identifier
  3. Old value (prior to write)
  4. New value (after write)
- Example of Log record
  1. $\{T1\ start\}$
  2. $\{T1, x2, 10, 15\}$
  3. $\{T1\ commit\}$
  4. $\{T1\ Abort\}$

Log Based Recovery

Write Ahead Log Strategy

- Log is written before any update is made to the database
- Transaction is not allowed to modify the physical data base until the undo portion of log is written to stable storage
- **Operations in Recovery Procedure**
  1. UNDO (Ti)$->$ Restore the old value
  2. REDO(Ti)$->$ Updates by new values

Approaches in Transaction Recovery Procedure:

Deffered Database Modification

- Transaction do not immediately update the physical database
- Only transaction log is updated
- Database is physically updates only after the transaction reaches its commit point

Immediate Database Modification

- Database is immediately updated by the transaction operations during the execution of transaction even before it reaches commit point
- In case of transaction abort before it reaches to commit point, a rollback or UNDO operation need to be done to restore the database to its earlier consistent state