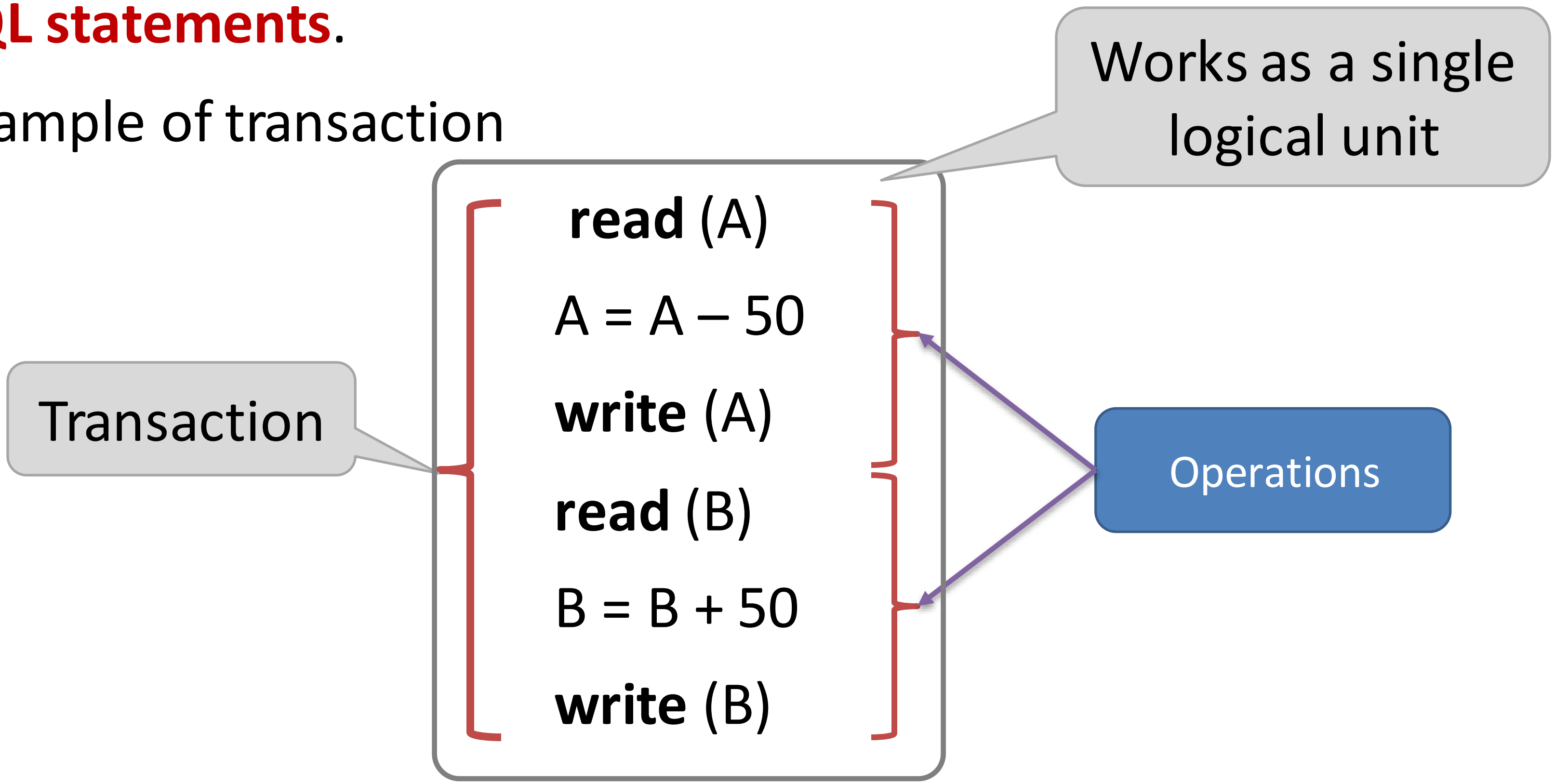
Transaction Management

BY: PROF. PRACHI SHAH

What is transaction?

- A transaction is a **sequence of operations performed as a single logical unit of work**.
- A transaction is a **logical unit of work that contains one or more SQL statements**.
- Example of transaction



ACID properties of transaction

- Atomicity **(Either transaction execute 0% or 100%)**
- Consistency **(database must remain in a consistent state after any transaction)**
- Isolation **(Multiple transaction may execute concurrently, but each transaction should feel like its executing in isolation)**
- Durability **(Once a transaction completed successfully, the changes it has made into the database should be permanent)**

ACID properties of transaction

- **Atomicity**
  - This property states that a **transaction must be treated as an atomic unit**, that is, **either all of its operations are executed or none**.
  - **Either transaction execute 0% or 100%**.
  - For example, consider a transaction to transfer Rs. 50 from account A to account B.
  - In this transaction, if Rs. 50 is deducted from account A then it must be added to account B.

0%

read (A)

A = A – 50

write (A)

FAIL

read (B)

B = B + 50

write (B)

100%

ACID properties of transaction

- **Consistency**
  - The **database must remain in a consistent state after any transaction**.
  - If the database was in a consistent state before the execution of a transaction, it must remain consistent after the execution of the transaction as well.
  - In our example, total of A and B must remain same before and after the execution of transaction.

A=500, B=500

A+B=1000

read (A)

A = A – 50

write (A)

read (B)

B = B + 50

write (B)

A=450, B=550

A+B=1000

ACID properties of transaction

- **Isolation**
  - **Changes occurring in a particular transaction will not be visible to any other transaction until it has been committed**.
  - **Intermediate transaction results must be hidden from other concurrently executed transactions**.
  - In our example once our transaction starts from first step (step 1) its result should not be access by any other transaction until last step (step 6) is completed.

read (A)

A = A – 50

write (A)

read (B)

B = B + 50

write (B)

ACID properties of transaction

- **Durability**
  - After a transaction completes successfully, **the changes it has made to the database persist (permanent)**, even if there are system failures.
  - Once our transaction completed up to last step (step 6) its result must be stored permanently. It should not be removed if system fails.

A=500, B=500

read (A)

A = A – 50

write (A)

read (B)

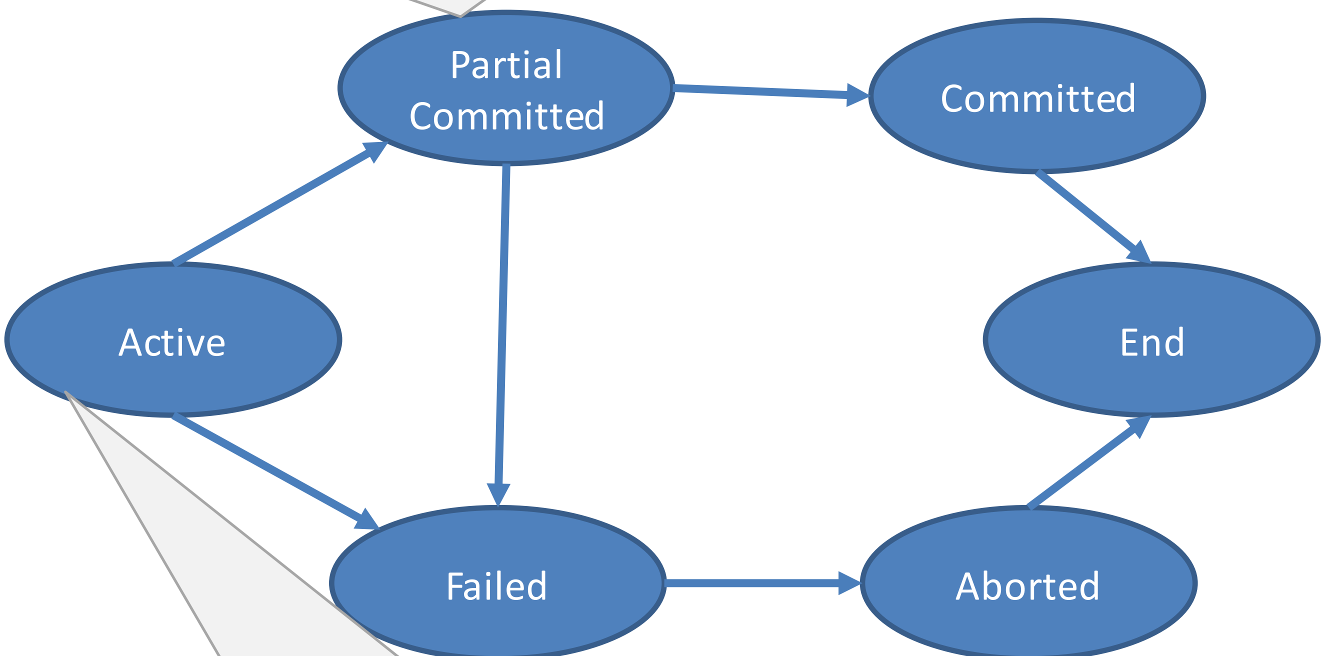
B = B + 50

write (B)

A=450, B=550

Transaction State Diagram \ State Transition Diagram

- When a transaction executes its final operation, it is said to be in a partially committed state.



- This is the initial state.
- The transaction stays in this state while it is executing.

read (A)

A = A – 50

write (A)

read (B)

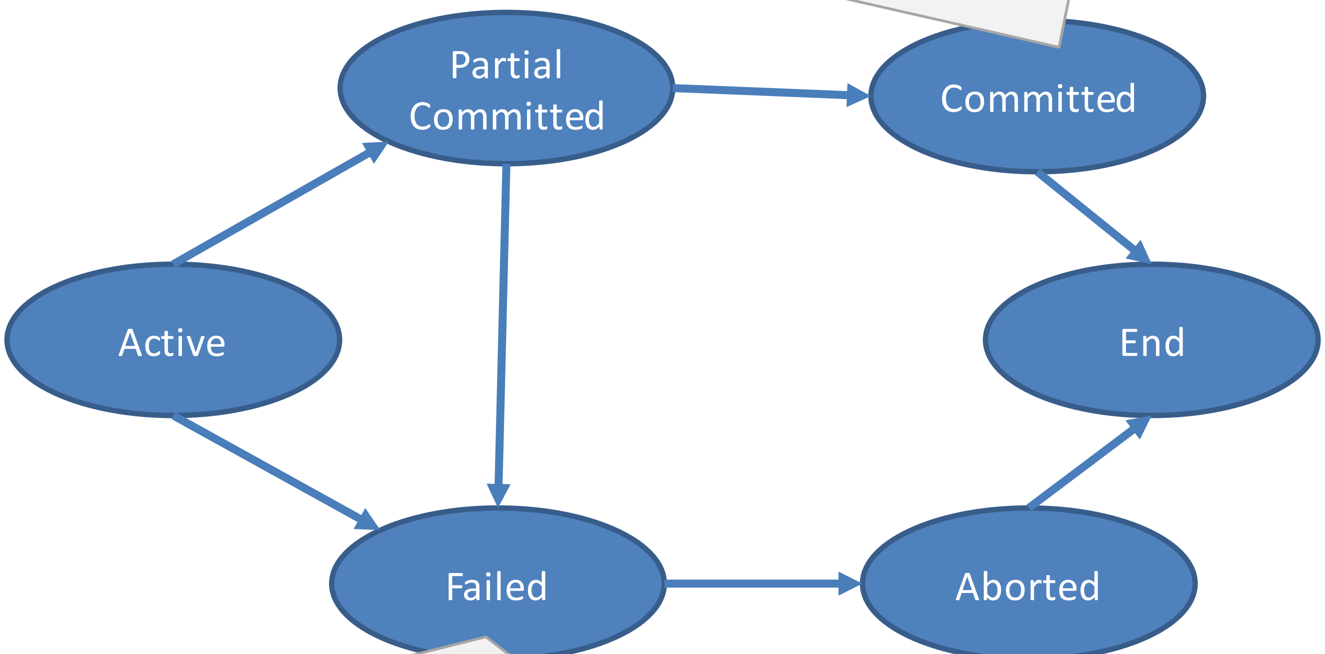
B = B + 50

write (B)

Commit

Transcription State Diagram \ State Transition Diagram

- The transaction enters in this state after successful completion of the transaction.
- We cannot abort or rollback a committed transaction.



- Discover that normal execution can no longer proceed.
- Once a transaction cannot be completed, any changes that it made must be undone rolling it back.

read (A)

A = A – 50

write (A)

read (B)

B = B + 50

write (B)

Commit

Transaction State Diagram \ State Transition Diagram

- Active
  - This is the **initial state**.
  - The transaction **stays in this state while it is executing**.
- Partial Committed
  - When a transaction **executes its final operation/ instruction**, it is said to be in a partially committed state.
- Failed
  - Discover that **normal execution can no longer proceed**.
  - Once a **transaction cannot be completed**, any **changes that it made must be undone rolling it back**.

Transaction State Diagram \ State Transition Diagram

- Committed
  - The transaction enters in this state after **successful completion of the transaction (after committing transaction).**
  - We **cannot abort or rollback a committed transaction**.
- Aborted
  - The **state after the transaction has been rolled back** and the **database has been restored to its state prior to the start of the transaction**.

What is schedule?

- A schedule is a **process of grouping the transactions** into one and **executing them in a predefined order**.
- A schedule is the **chronological (sequential) order** in which **instructions are executed** in a system.
- A schedule is required in a database because **when some transactions execute in parallel, they may affect the result of the transaction**.
- Means if one transaction is updating the values which the other transaction is accessing, then the order of these two transactions will change the result of another transaction.
- Hence a schedule is created to execute the transactions.

Example of schedule

T1	T2	A=B=1000
Read (A)		Read (1000)
A = A - 50		A = 1000 - 50
Write (A)		Write (950)
Read (B)		Read (1000)
B = B + 50		B = 1000 + 50
Write (B)		Write (1050)
Commit		Commit
	Read (A)	Read (950)
	temp = A * 0.1	temp = 950 * 0.1
	A = A - temp	A = 950 - 95
	Write (A)	Write (855)
	Read (B)	Read (1050)
	B = B + temp	B = 1050 + 95
	Write (B)	Write (1145)
	Commit	Commit

Example of schedule

T1	T2	A=B=1000
Read (A)		Read (1000)
Temp = A * 0.1		Temp = 1000 * 0.1
A = A - temp		A = 1000 - 100
Write (A)		Write (900)
Read (B)		Read (1000)
B = B + temp		B = 1000 + 100
Write (B)		Write (1100)
Commit		Commit
	Read (A)	Read (900)
	A = A - 50	A = 900 - 50
	Write (A)	Write (850)
	Read (B)	Read (1100)
	B = B + 50	B = 1100 + 50
	Write (B)	Write (1150)
	Commit	Commit

Serial schedule

- A serial schedule is one in which **no transaction starts until a running transaction has ended**.
- Transactions are executed one after the other.
- This type of schedule is called a serial schedule, as transactions are executed in a serial manner.

Example of serial schedule

T1	T2
Read (A)	
Temp = A * 0.1	
A = A - temp	
Write (A)	
Read (B)	
B = B + temp	
Write (B)	
Commit	
	Read (A)
	A = A - 50
	Write (A)
	Read (B)
	B = B + 50
	Write (B)
	Commit

A=B=1000
Read (1000)
Temp = 1000 * 0.1
A = 1000 - 100
Write (900)
Read (1000)
B = 1000 + 100
Write (1100)
Commit
Read (900)
A = 900 - 50
Write (850)
Read (1100)
B = 1100 + 50
Write (1150)
Commit

Example of serial schedule

T1	T2
	Read (A)
	A = A - 50
	Write (A)
	Read (B)
	B = B + 50
	Write (B)
	Commit
Read (A)	
Temp = A * 0.1	
A = A - temp	
Write (A)	
Read (B)	
B = B + temp	
Write (B)	
Commit	

A=B=1000
Read (1000)
A = 1000 - 50
Write (950)
Read (1000)
B = 1000 + 50
Write (1050)
Commit
Read (950)
Temp = 950 * 0.1
A = 950 – temp (95)
Write (855)
Read (1050)
B = 1050 + temp (95)
Write (1145)
Commit

Concurrent / Interleaved schedule

- Schedule that **interleave the execution of different transactions**.
- Means **second transaction is started before the first one could end** and **execution can switch between the transactions back and forth**.

Example of interleaved schedule

T1	T2
	Read (A)
	Temp = A * 0.1
	A = A - temp
Read (A)	Write (A)
A = A - 50	
Write (A)	
	Read (B)
	B = B + temp
	Write (B)
Read (B)	Commit
B = B + 50	
Write (B)	
Commit	

Example of interleaved schedule

T1	T2
Read (A)	
A = A - 50	
	Read (A)
	Temp = A * 0.1
	A = A - temp
	Write (A)
Write (A)	
Read (B)	
B = B + 50	
Write (B)	
Commit	Read (B)
	B = B + temp
	Write (B)
	Commit

A=B=1000
Read (1000)
A = 1000 – 50
Read (1000)
Temp = 1000 * 0.1
A = 1000 – temp (100)
Write (900)
Write (950)
Read (1000)
B = 1000 + 50
Write (1050)
Commit
Read (1050)
B = 1050 + temp (100)
Write (1150)
Commit

Equivalent schedule

- If two schedules **produce the same result after execution**, they are said to be equivalent schedule.
- They may yield the same result for some value and different results for another set of values.
- That's why this equivalence is not generally considered significant.

Equivalent schedule

T1	T2
Read (A)	
A = A - 50	
Write (A)	
Commit	Read (A)
	Temp = A * 0.1
	A = A - temp
	Write (A)
	Commit
Read (B)	
B = B + 50	
Write (B)	
Commit	Read (B)
	B = B + temp
	Write (B)
	Commit

Both schedules are equivalent

In both schedules the sum "A + B" is preserved.

T1	T2
Read (A)	
A = A - 50	
Write (A)	
Read (B)	
B = B + 50	
Write (B)	
Commit	
	Read (A)
	Temp = A * 0.1
	A = A - temp
	Write (A)
	Read (B)
	B = B + temp
	Write (B)
	Commit

Serializability

- Serializability is a concept which will help us to identify valid concurrent schedules.
- A schedule is serializable **if it is equivalent to a serial schedule**.
- In serial schedules, **only one transaction is allowed to execute at a time** i.e. no concurrency is allowed.
- Whereas in serializable schedules, **multiple transactions can execute simultaneously** i.e. concurrency is allowed.
- Types (forms) of serializability
    1. Conflict serializability
    2. View serializability

Conflicting instructions

- Let I_i and I_j be two instructions of transactions T_i and T_j respectively.
  1. I_i = read(Q), I_j = read(Q)

     I_i and I_j **don't conflict**
  2. I_i = read(Q), I_j = write(Q)

     I_i and I_j **conflict**
  3. I_i = write(Q), I_j = read(Q)

     I_i and I_j **conflict**
  4. I_i = write(Q), I_j = write(Q)

     I_i and I_j **conflict**

T1	T2
read (Q)	
	read (Q)

T1	T2
	read (Q)
read (Q)	

T1	T2
read (Q)	
	write(Q)

T1	T2
	write(Q)
read (Q)	

T1	T2
write(Q)	
	read (Q)

T1	T2
	read (Q)
write(Q)	

T1	T2
write(Q)	
	write(Q)

T1	T2
	write(Q)
write(Q)	

Conflicting instructions

- Instructions **I_i and I_j conflict** if and only if there exists some item **Q accessed by both Ii and Ij**, and **at least one of these instructions write Q**.
- If **both the transactions access different data item** then they are **not conflict**.

Conflict serializability

- If a given **schedule can be converted into a serial schedule by swapping its non-conflicting operations**, then it is called as a conflict serializable schedule.

Conflict serializability (example)

T1	T2
Read (A)	
A = A - 50	
Write (A)	
	Read (A)
	Temp = A * 0.1
	A = A - temp
	Write (A)
Read (B)	
B = B + 50	
Write (B)	
Commit	
	Read (B)
	B = B + temp
	Write (B)
	Commit

T1	T2
Read (A)	
A = A - 50	
Write (A)	
Read (B)	
B = B + 50	
Write (B)	
Commit	
	Read (A)
	Temp = A * 0.1
	A = A - temp
	Write (A)
	Read (B)
	B = B + temp
	Write (B)
	Commit

Conflict serializability

- Example of a schedule that is not conflict serializable:

T1	T2
Read (A)	
	Write (A)
Write (A)	

- We are **unable to swap instructions** in the above schedule to obtain either the serial schedule < T1, T2 >, or the serial schedule < T2, T1 >.

Key points Serializability

- *Every conflict serializable is also a view serializable schedule (NOT vice versa)*

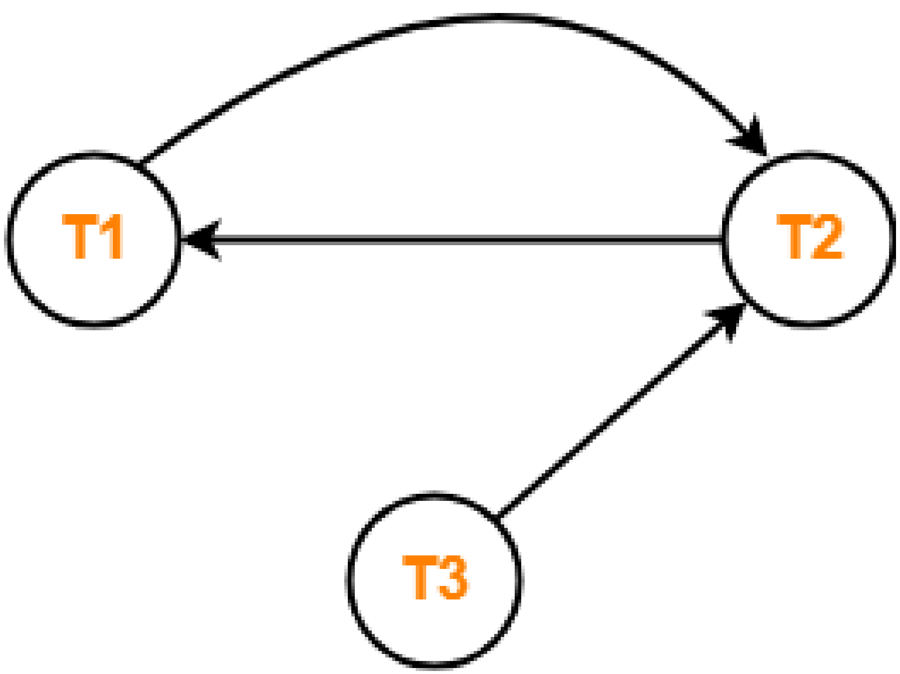
- **Testing of Conflict serializability : Using precedence graph.**

1. Create number of nodes = number of transactions
2. Check for conflict pairs (R-W, W-R, W-W on same data item) and draw the edges between them.
3. Check for LOOP/CYCLE in graph. If loop is there, NOT CONFLICT SERIALIZABLE.

Example 1: Conflict Serializability

Check whether the given schedule S is conflict serializable or not-

S : $R_1(A)$, $R_2(A)$, $R_1(B)$, $R_2(B)$, $R_3(B)$, $W_1(A)$, $W_2(B)$



T1	T2	T3
$R_1(A)$		
	$R_2(A)$	
$R_1(B)$		
	$R_2(B)$	
		$R_3(B)$
$W_1(A)$		
	$W_2(B)$	

S is not conflict serializable.

Example 2: Conflict Serializability

- Consider a schedule S with two transactions T_1 and T_2 as follows;

  S: $R_1(x);W_2(x);R_1(x);W_1(y)$;commit1;commit2;

- Is the schedule S conflict serializable?

Instruction	T_1	T_2
1	$R(x)$	
2		$W(x)$
3	$R(x)$	
4	$W(y)$	
5	*Commit*	
6		*Commit*

Example 2: Conflict Serializability

- Consider a schedule S with two transactions T_1 and T_2 as follows;

S: $R_1(x);W_2(x);R_1(x);W_1(y);$commit1;commit2;

- Is the schedule S conflict serializable?

Schedule S is not conflict serializable schedule.

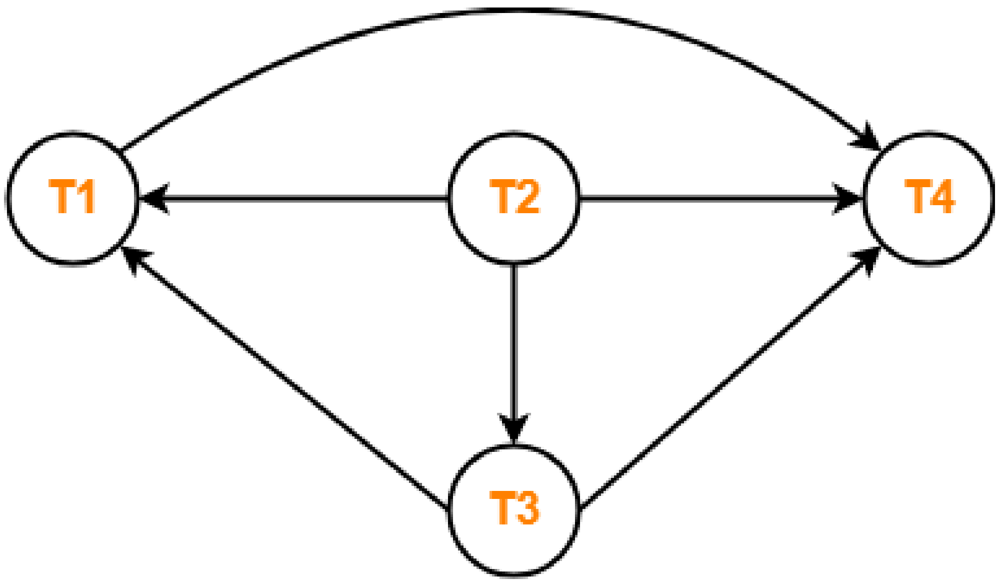
Example 3: Conflict Serializability

Check whether the given schedule S is conflict serializable or not-

T1	T2	T3	T4
	R(X)		
		W(X)	
		Commit	
W(X)			
Commit			
	W(Y)		
	R(Z)		
	Commit		
			R(X)
			R(Y)
			Commit

Example 3: Conflict Serializability

Check whether the given schedule S is conflict serializable or not-



S is conflict serializable.
Serial Order:

T2-T3-T1-T4

View serializability

- Let S1 and S2 be two schedules with the same set of transactions. S1 and S2 are **view equivalent if the following three conditions** are satisfied, for each data item Q

  1. Initial Read
  2. Updated Read
  3. Final Write

Initial Read

- If in **schedule S1, transaction T_i reads the initial value of Q**, then in **schedule S2 also transaction T_i must read the initial value of Q**.

S1

T1	T2
Read (A)	
	Write (A)

S3

T1	T2
	Read (A)
Write (A)	

S2

T1	T2
	Write (A)
Read (A)	

- Above two schedules S1 and S3 are not view equivalent because initial read operation in S1 is done by T1 and in S3 it is done by T2.
- Above two schedules S1 and S2 are view equivalent because initial read operation in S1 is done by T1 and in S2 it is also done by T1.

Updated Read

- If in **schedule S1 transaction T_i executes read(Q), and that value was produced by transaction T_j (if any)**, then in **schedule S2 also transaction T_i must read the value of Q that was produced by transaction T_j**.

S1

T1	T2	T3
Write (A)		
	Write (A)	
		Read (A)

S3

T1	T2	T3
	Write (A)	
Write (A)		
		Read (A)

- Above two schedules are not view serializable because,
- in S1, T3 is reading A that is updated by T2 and
- in S3, T3 is reading A which is updated by T1.

Updated Read

- If in **schedule S1 transaction T_i executes read(Q), and that value was produced by transaction T_j** (if any), then in **schedule S2 also transaction T_i must read the value of Q that was produced by the same write(Q) operation of transaction T_j**.

S1

T1	T2	T3
	Read (A)	
Write (A)		
	Write (A)	
		Read (A)

S2

T1	T2	T3
	Read (A)	
	Write (A)	
		Read (A)
Write (A)		

- Above two schedules are view equal because,
- in S1, T3 is reading A that is updated by T2 and
- in S3 also, T3 is reading A which is updated by T2.

Final Write

- **If T_i performs the final write on the data value in S1, then T_i also performs the final write on the data value in S2**.

S1

T1	T2	T3
Write (A)		
	Read (A)	
		Write (A)

S2

T1	T2	T3
	Read (A)	
Write (A)		
		Write (A)

- Above two schedules is view equal because final write operation
- in S1 is done by T3 and
- in S2 also the final write operation is also done by T3.

View Serializability

T1	T2	T3
$R_1(A)$		
	$W_2(A)$	
$W_1(A)$		
		$W_3(A)$

Possible serial schedules?

T1,T2,T3

T1,T3,T2

T2,T1,T3

T2,T3,T1

T3,T1,T2

T3,T2,T1

You can compare it with any serial schedule, **and if all three conditions are true, It is view serializable.**

1. Initial read?
2. Updated read?
3. Final write?

T1	T2	T3
$R_1(A)$		
$W_1(A)$		
	$W_2(A)$	
		$W_3(A)$

Key points Serializability

- ***Every conflict serializable is also a view serializable schedule (NOT vice versa)***

Example 4: View Serializability

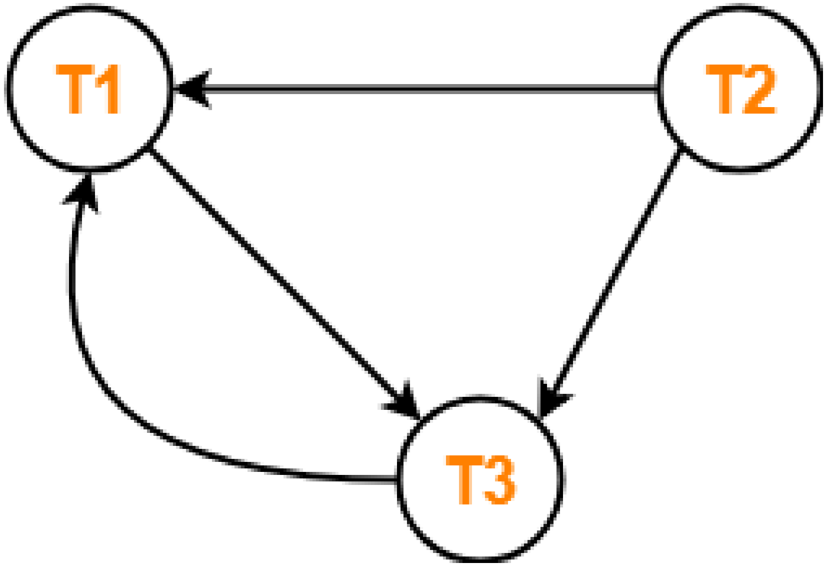
Check whether the given schedule S is view serializable or not-

T1	T2	T3
R (A)		
	R (A)	
		W (A)
W (A)		

Is it conflict serializable?

Example 4: View Serializability

Check whether the given schedule S is view serializable or not-



Is it conflict serializable?
NO

Is it View serializable?

Example 4: View Serializability

Check whether the given schedule S is view serializable or not-

Is it View serializable?
NO

Example 5: View Serializability

Check whether the given schedule S is view serializable or not. If yes, then give the serial schedule.

S : $R_1(A)$, $W_2(A)$, $R_3(A)$, $W_1(A)$, $W_3(A)$

Recoverable Schedule

- **Recoverable schedule**:
- A schedule S is recoverable if
  1. Transaction T_i have read(x), that has been previously written by some other transaction T_j.
  2. Transaction T_j must commit before T_i commits.

- Suppose transaction T_j fails/aborts, then we have to rollback all operations done by T_j, *and it is also necessary to abort T_i (which is dependent on T_j).*
- Order of Commit/Abort Matters.

if T_9 commits after read(A)

T_8	T_9
read(A)	
write(A)	
	read(A)
read(B)	

Recoverable Schedule

- S_a is not recoverable:
  - S_a: $r_1(X)$; $w_1(X)$; $r_2(X)$; $r_1(Y)$; $w_2(X)$; c_2; $w_1(Y)$; c_1;

- Consider the following schedules:
  - S_c: $r_1(X)$; $w_1(X)$; $r_2(X)$; $r_1(Y)$; $w_2(X)$; c_2; a_1;
  - S_d: $r_1(X)$; $w_1(X)$; $r_2(X)$; $r_1(Y)$; $w_2(X)$; $w_1(Y)$; c_1; c_2;
  - S_e: $r_1(X)$; $w_1(X)$; $r_2(X)$; $r_1(Y)$; $w_2(X)$; $w_1(Y)$; a_1; a_2;

- S_c is not recoverable because:
- S_d is recoverable because:
- S_e is recoverable because:

Cascade less Schedule

- Cascading Rollbacks / aborts

T_{10}	T_{11}	T_{12}
read(A)		
read(B)		
write(A)		
	read(A)	
	write(A)	
		read(A)

- Cascade less schedule : **Eliminates Cascading rollbacks**:

1. T_i read(x), Previously written by T_j

2. T_j must commit before T_i reads.

- **Every cascade less schedule is also recoverable.**

Cascade less Schedule

T1	T2
R(X)	
W(X)	
	R(Y)
COMMIT;	
	R(X)

CASCADE LESS?

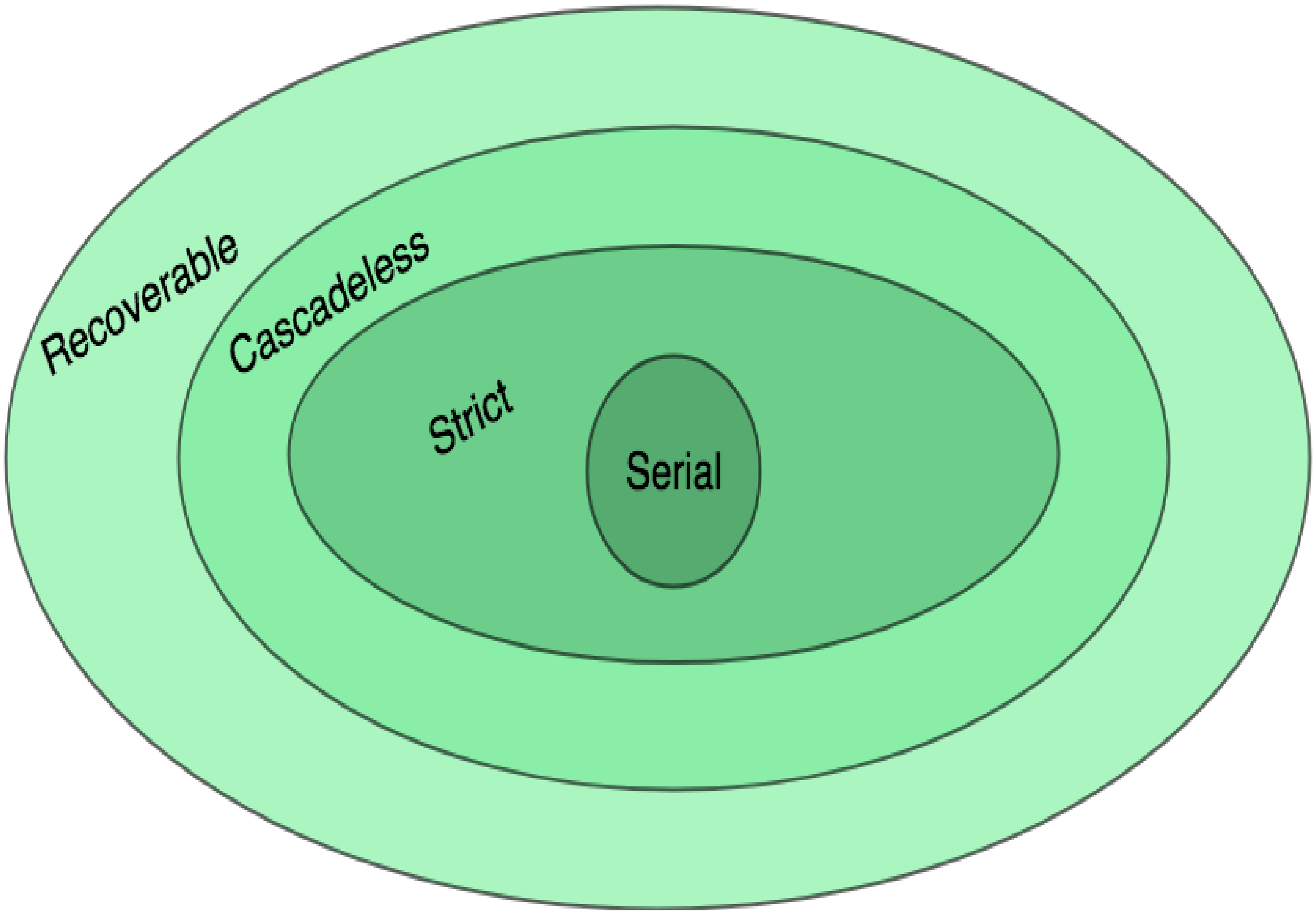
RECOVERABLE?

Strict Schedule

- **Strict schedule**:
- A schedule in which a transaction can neither read or write an item X until the last transaction that wrote X has committed.

- S_a is not strict:
  - S_a: w1(X, 5); w2(X , 8); a1
- Suppose the value of X was originally 9.
- If T1 aborts, as in S_a, the recovery system will restore the value of X to 9, even though it has already been changed to 8 by T2, thus leading to incorrect results.
- Although S_a is cascade-less, it is not strict
  - It permits T2 to write X even though T1 that last wrote X had not yet committed (or aborted).

Schedules



GATE Example Schedule

- Consider the following database schedule with two transactions, T1 and T2.

  S = r2(X); r1(X); r2(Y); w1(X); r1(Y); w2(X); a1; a2;

- S is recoverable?
- S is cascade less?
- S is strict?

GATE Example Schedule

T1	T2
	R(X)
R(X)	
	R(Y)
W(X)	
R(Y)	
	W(X)
a1	
	a2

- Given schedule is recoverable.
- Given schedule is cascade less.
- Given schedule is NOT strict.

The End..!!

Thank You….