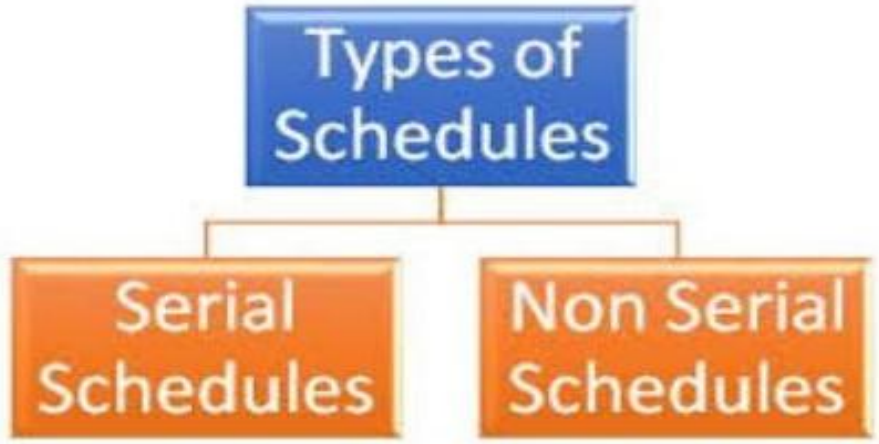
Schedules in DBMS

Dr. Hrudaya Kumar Tripathy,
School of Computer Engineering, KIIT

Schedules

- A series of operations from one transaction to another transaction is known as **schedule**. It is used to preserve the order of the operation in each of the individual transaction.

- The order in which the operations of multiple transactions appear for execution is called as a schedule.



Serial Schedules

In serial schedules, all the transactions execute serially one after the other.

When one transaction executes, no other transaction is allowed to execute.

Transaction T1	Transaction T2
R (A)	
W (A)	
R (B)	
W (B)	
Commit	
	R (A)
	W (B)
	Commit

In this schedule,

- There are two transactions T1 and T2 executing serially one after the other.
- Transaction T1 executes first.
- After T1 completes its execution, transaction T2 executes.
- So, this schedule is an example of a **Serial Schedule**.

Non-Serial Schedules

In non-serial schedules, multiple transactions execute concurrently. Operations of all the transactions are inter leaved or mixed with each other.

Transaction T1	Transaction T2
R (A)	
W (A)	
	R (A)
R (B)	
W (B)	
Commit	
	R (B)
	Commit

In this schedule,

- There are two transactions T1 and T2 executing concurrently.
- The operations of T1 and T2 are interleaved.
- So, this schedule is an example of a **Non-Serial Schedule**.

Finding Number Of Schedules-

Consider there are n number of transactions T1, T2, T3 …. , Tn with N1, N2, N3 …. , Nn number of operations respectively.

Total Number of Schedules-

Total number of possible schedules (serial + non-serial) is given by-

$$\frac{(N1 + N2 + N3 + ... + Nn)!}{N1! \times N2! \times N3! \times .... \times Nn!}$$

Total Number of Serial Schedules-

Total number of serial schedules

= Number of different ways of arranging n transactions

= n!

Total Number of Non-Serial Schedules-

Total number of non-serial schedules

= Total number of schedules – Total number of serial schedules

Problem-

Consider there are three transactions with 2, 3, 4 operations respectively, find-

1. How many total number of schedules are possible?
2. How many total number of serial schedules are possible?
3. How many total number of non-serial schedules are possible?

Total Number of Schedules-

Using the above formula, we have-

$$\text{Total number of schedules} = \frac{(2+3+4)!}{2! \times 3! \times 4!}$$

$$= 1260$$

Total Number of Serial Schedules-

Total number of serial schedules

= Number of different ways of arranging 3 transactions

= 3! = 6

Total Number of Non-Serial Schedules-

Total number of non-serial schedules

= Total number of schedules – Total number of serial schedules

= 1260 – 6 = 1254

Serializability in DBMS-

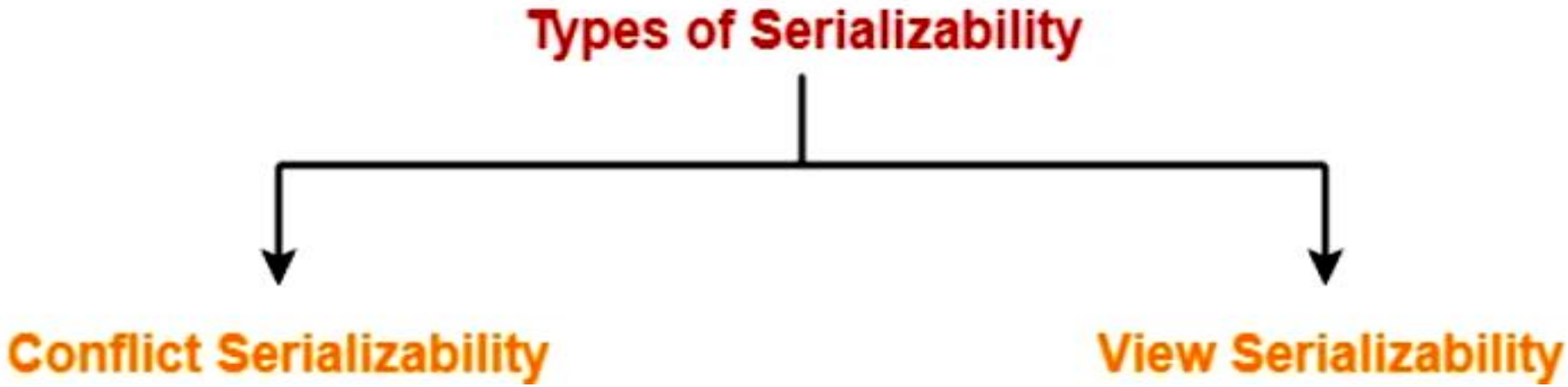
- Some non-serial schedules may lead to inconsistency of the database.
- Serializability is a concept that helps to identify which non-serial schedules are correct and will maintain the consistency of the database.

Serializable Schedules:

If a given non-serial schedule of ‘n’ transactions is equivalent to some serial schedule of ‘n’ transactions, then it is called as a serializable schedule.

Serial Schedules Vs Serializable Schedules-

Serial Schedules	Serializable Schedules
No concurrency is allowed. Thus, all the transactions necessarily execute serially one after the other.	Concurrency is allowed. Thus, multiple transactions can execute concurrently.
Serial schedules lead to less resource utilization and CPU throughput.	Serializable schedules improve both resource utilization and CPU throughput.
Serial Schedules are less efficient as compared to serializable schedules. (due to above reason)	Serializable Schedules are always better than serial schedules. (due to above reason)


Types of Serializability
Conflict Serializability
View Serializability

Conflict Serializability

Conflict Serializability-

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

Conflicting Operations-

Two operations are called as **conflicting operations** if all the following conditions hold true for them-

- Both the operations belong to different transactions
- Both the operations are on the same data item
- At least one of the two operations is a write operation

Example-

Consider the following schedule-

Transaction T1	Transaction T2
R1 (A)	
W1 (A)	
	R2 (A)
R1 (B)	

In this schedule,

- W1 (A) and R2 (A) are called as conflicting operations.
- This is because all the above conditions hold true for them.

Checking Whether a Schedule is Conflict Serializable Or Not-

Step-01:

Find and list all the conflicting operations.

Step-02:

Start creating a precedence graph by drawing one node for each transaction.

Step-03:

- Draw an edge for each conflict pair such that if X_i (V) and Y_j (V) forms a conflict pair then draw an edge from T_i to T_j.
- This ensures that T_i gets executed before T_j.

Step-04:

- Check if there is any cycle formed in the graph.
- If there is no cycle found, then the schedule is conflict serializable otherwise not.

Problem-01:

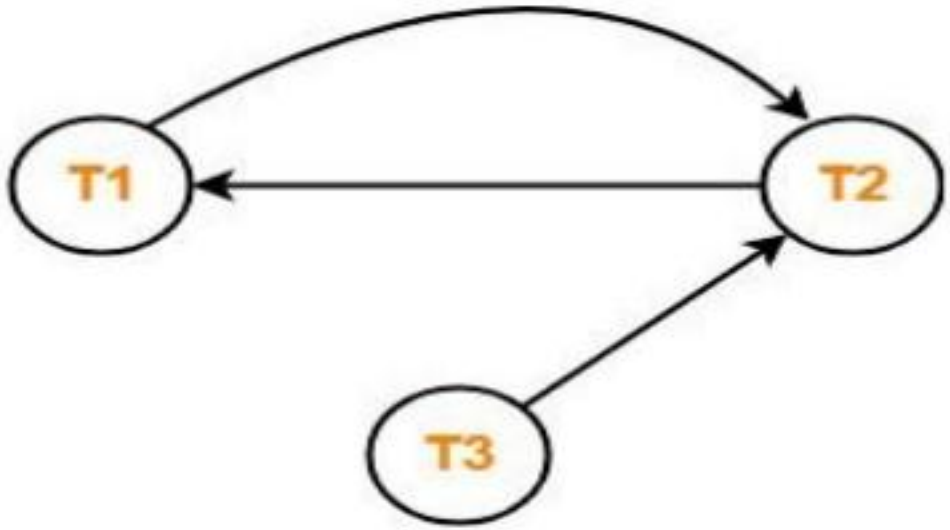
Check whether the given schedule S is conflict serializable or not-

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

Step-01:

List all the conflicting operations and determine the dependency between the transactions-

- $R_2(A)$, $W_1(A)$ $(T_2 \rightarrow T_1)$
- $R_1(B)$, $W_2(B)$ $(T_1 \rightarrow T_2)$
- $R_3(B)$, $W_2(B)$ $(T_3 \rightarrow T_2)$



Step-02:

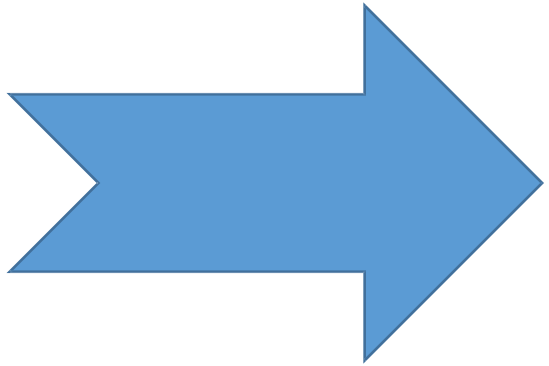
Draw the precedence graph-

- Clearly, there exists a cycle in the precedence graph.
- Therefore, the given schedule S is not conflict serializable.

Problem-02:

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

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



Checking Whether S is Conflict Serializable Or Not-

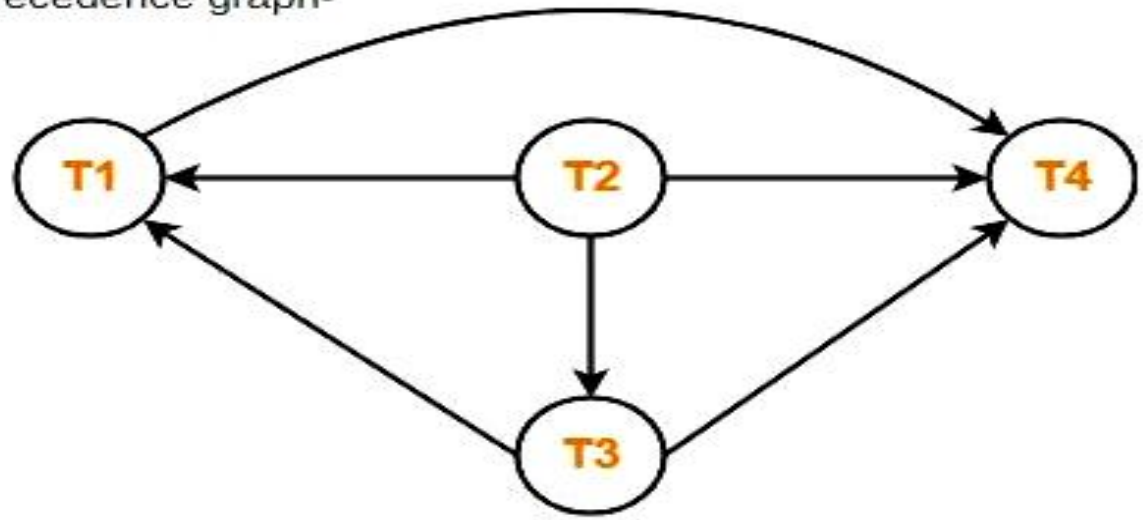
Step-01:

List all the conflicting operations and determine the dependency between the transactions-

- $R_2(X)$, $W_3(X)$ $(T_2 \rightarrow T_3)$
- $R_2(X)$, $W_1(X)$ $(T_2 \rightarrow T_1)$
- $W_3(X)$, $W_1(X)$ $(T_3 \rightarrow T_1)$
- $W_3(X)$, $R_4(X)$ $(T_3 \rightarrow T_4)$
- $W_1(X)$, $R_4(X)$ $(T_1 \rightarrow T_4)$
- $W_2(Y)$, $R_4(Y)$ $(T_2 \rightarrow T_4)$

Step-02:

Draw the precedence graph-



- Clearly, there exists no cycle in the precedence graph.
- Therefore, the given schedule S is conflict serializable.

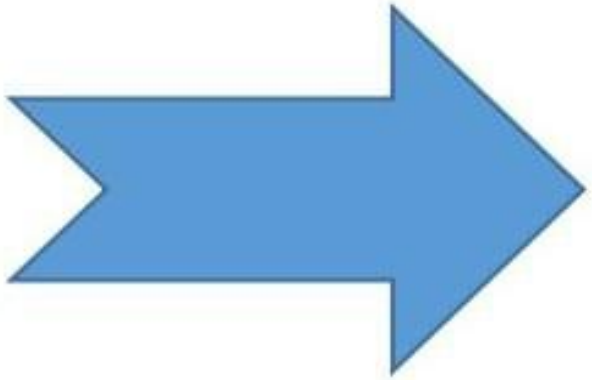
Checking Whether S is Recoverable Or Not-

- Conflict serializable schedules are always recoverable.
- Therefore, the given schedule S is recoverable.

Problem-03:

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

T1	T2	T3	T4
			R(A)
	R(A)		
		R(A)	
W(B)			
	W(A)		
		R(B)	
	W(B)		



Checking Whether S is Conflict Serializable Or Not-

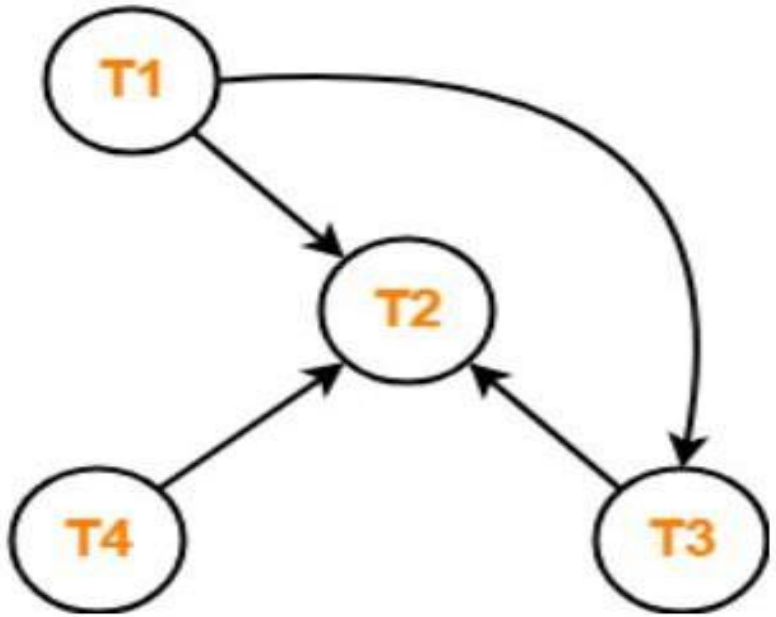
Step-01:

List all the conflicting operations and determine the dependency between the transactions-

- $R_4(A)$, $W_2(A)$ $(T_4 \rightarrow T_2)$
- $R_3(A)$, $W_2(A)$ $(T_3 \rightarrow T_2)$
- $W_1(B)$, $R_3(B)$ $(T_1 \rightarrow T_3)$
- $W_1(B)$, $W_2(B)$ $(T_1 \rightarrow T_2)$
- $R_3(B)$, $W_2(B)$ $(T_3 \rightarrow T_2)$

Step-02:

Draw the precedence graph-



- Clearly, there exists no cycle in the precedence graph.
- Therefore, the given schedule S is conflict serializable.

View Serializability

View Serializability-

If a given schedule is found to be view equivalent to some serial schedule, then it is called as a view serializable schedule.

Consider two schedules S1 and S2 each consisting of two transactions T1 and T2. Schedules S1 and S2 are called view equivalent if the following three conditions hold true for them-

Condition-01:

For each data item X, if transaction T_i reads X from the database initially in schedule S1, then in schedule S2 also, T_i must perform the initial read of X from the database.

Condition-02:

If transaction T_i reads a data item that has been updated by the transaction T_j in schedule S1, then in schedule S2 also, transaction T_i must read the same data item that has been updated by the transaction T_j.

Condition-03:

For each data item X, if X has been updated at last by transaction T_i in schedule S1, then in schedule S2 also, X must be updated at last by transaction T_i.

Checking Whether a Schedule is View Serializable Or Not-

Method-01:

Check whether the given schedule is conflict serializable or not.

- If the given schedule is conflict serializable, then it is surely view serializable. Stop and report your answer.
- If the given schedule is not conflict serializable, then it may or may not be view serializable. Go and check using other methods.

Method-02:

Check if there exists any blind write operation.

(Writing without reading is called as a blind write).

- If there does not exist any blind write, then the schedule is surely not view serializable. Stop and report your answer.
- If there exists any blind write, then the schedule may or may not be view serializable. Go and check using other methods.

Method-03:

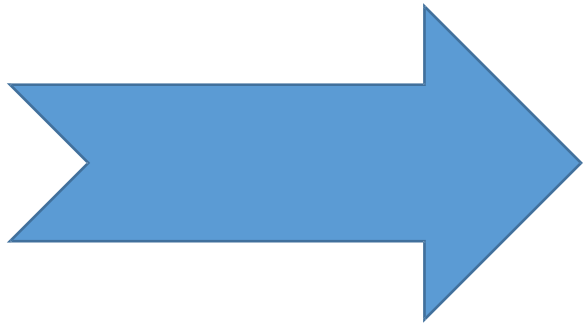
In this method, try finding a view equivalent serial schedule.

- By using the above three conditions, write all the dependencies.
- Then, draw a graph using those dependencies.
- If there exists no cycle in the graph, then the schedule is view serializable otherwise not.

Problem-01:

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

T1	T2	T3	T4
R (A)			
	R (A)		
		R (A)	
			R (A)
W (B)			
	W (B)		
		W (B)	
			W (B)



- We know, if a schedule is conflict serializable, then it is surely view serializable.
- So, let us check whether the given schedule is conflict serializable or not.

Checking Whether S is Conflict Serializable Or Not-

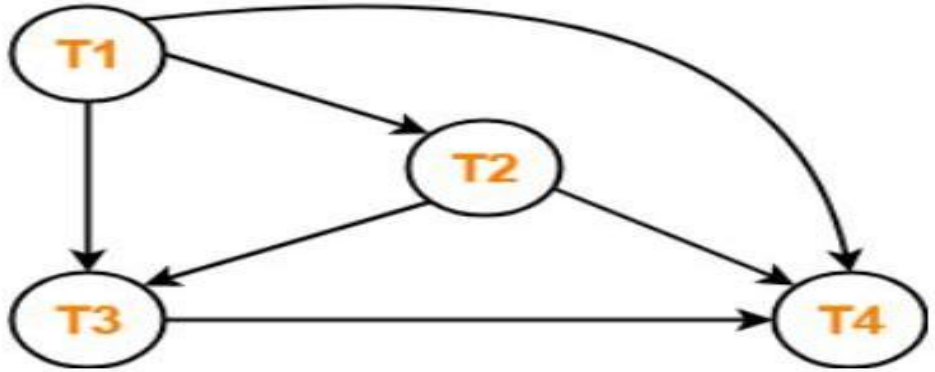
Step-01:

List all the conflicting operations and determine the dependency between the transactions-

- $W_1(B)$, $W_2(B)$ $(T_1 \rightarrow T_2)$
- $W_1(B)$, $W_3(B)$ $(T_1 \rightarrow T_3)$
- $W_1(B)$, $W_4(B)$ $(T_1 \rightarrow T_4)$
- $W_2(B)$, $W_3(B)$ $(T_2 \rightarrow T_3)$
- $W_2(B)$, $W_4(B)$ $(T_2 \rightarrow T_4)$
- $W_3(B)$, $W_4(B)$ $(T_3 \rightarrow T_4)$

Step-02:

Draw the precedence graph-

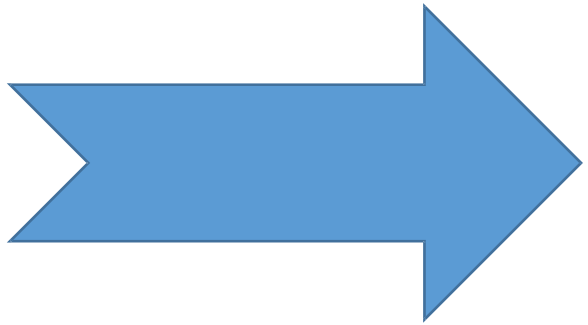


- Clearly, there exists no cycle in the precedence graph.
- Therefore, the given schedule S is conflict serializable.
- Thus, we conclude that the given schedule is also view serializable.

Problem-02:

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

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



- We know, if a schedule is conflict serializable, then it is surely view serializable.
- So, let us check whether the given schedule is conflict serializable or not.

Checking Whether S is Conflict Serializable Or Not-

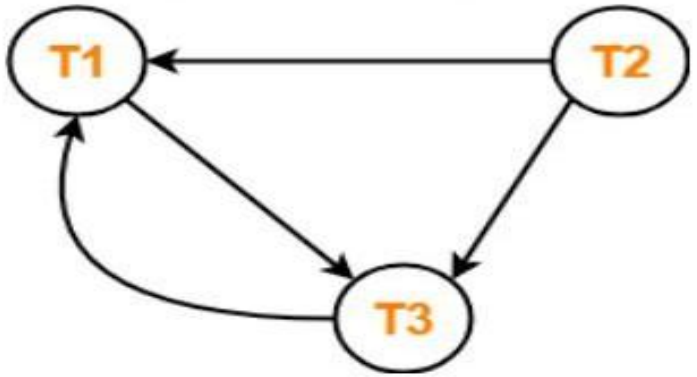
- $R_1(A)$, $W_3(A)$ $(T_1 \rightarrow T_3)$
- $R_2(A)$, $W_3(A)$ $(T_2 \rightarrow T_3)$
- $R_2(A)$, $W_1(A)$ $(T_2 \rightarrow T_1)$
- $W_3(A)$, $W_1(A)$ $(T_3 \rightarrow T_1)$

Step-01:

List all the conflicting operations and determine the dependency between the transactions-

Step-02:

Draw the precedence graph-



- Clearly, there exists a cycle in the precedence graph.
- Therefore, the given schedule S is not conflict serializable.

Now,

- Since, the given schedule S is not conflict serializable, so, it may or may not be view serializable.
- To check whether S is view serializable or not, let us use another method.
- Let us check for blind writes.

Checking for Blind Writes-

- There exists a blind write W_3 (A) in the given schedule S.
- Therefore, the given schedule S may or may not be view serializable.

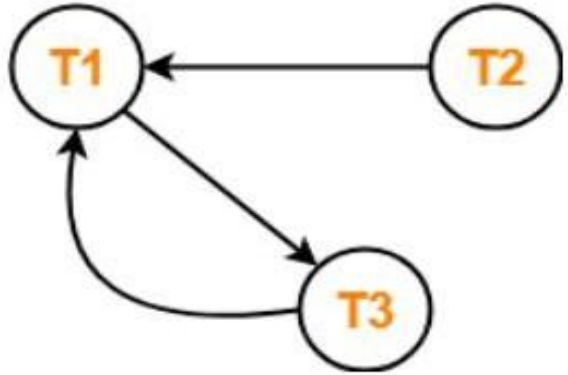
Now,

- To check whether S is view serializable or not, let us use another method.
- Let us derive the dependencies and then draw a dependency graph.

Drawing a Dependency Graph-

- T1 firstly reads A and T3 firstly updates A.
- So, T1 must execute before T3.
- Thus, we get the dependency **T1 → T3**.
- Final updation on A is made by the transaction T1.
- So, T1 must execute after all other transactions.
- Thus, we get the dependency **(T2, T3) → T1**.
- There exists no write-read sequence.

Now, let us draw a dependency graph using these dependencies-

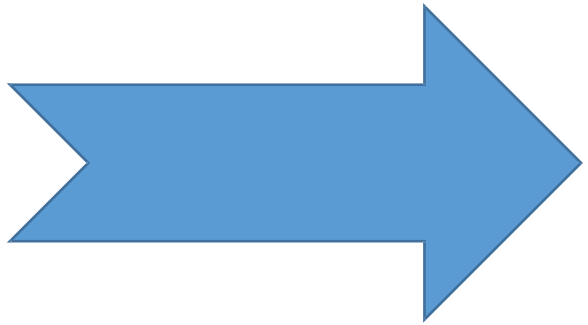


- Clearly, there exists a cycle in the dependency graph.
- Thus, we conclude that the given schedule S is not view serializable.

Problem-03:

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

T1	T2
R (A)	
A = A + 10	
	R (A)
	A = A + 10
W (A)	
	W (A)
R (B)	
B = B + 20	
	R (B)
	B = B x 1.1
W (B)	
	W (B)



- We know, if a schedule is conflict serializable, then it is surely view serializable.
- So, let us check whether the given schedule is conflict serializable or not.

Checking Whether S is Conflict Serializable Or Not-

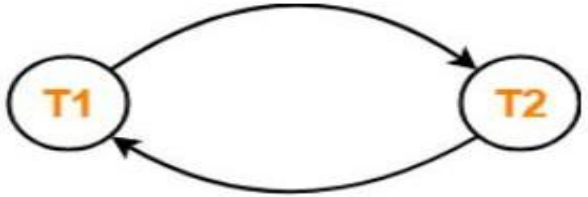
Step-01:

List all the conflicting operations and determine the dependency between the transactions-

- $R_1(A)$, $W_2(A)$ $(T_1 \rightarrow T_2)$
- $R_2(A)$, $W_1(A)$ $(T_2 \rightarrow T_1)$
- $W_1(A)$, $W_2(A)$ $(T_1 \rightarrow T_2)$
- $R_1(B)$, $W_2(B)$ $(T_1 \rightarrow T_2)$
- $R_2(B)$, $W_1(B)$ $(T_2 \rightarrow T_1)$

Step-02:

Draw the precedence graph-



- Clearly, there exists a cycle in the precedence graph.
- Therefore, the given schedule S is not conflict serializable.

Now,

- Since, the given schedule S is not conflict serializable, so, it may or may not be view serializable.
- To check whether S is view serializable or not, let us use another method.
- Let us check for blind writes.

Checking for Blind Writes-

- There exists no blind write in the given schedule S.
- Therefore, it is surely not view serializable.

Problem-04:

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

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

- We know, if a schedule is conflict serializable, then it is surely view serializable.
- So, let us check whether the given schedule is conflict serializable or not.

Checking Whether S is Conflict Serializable Or Not-

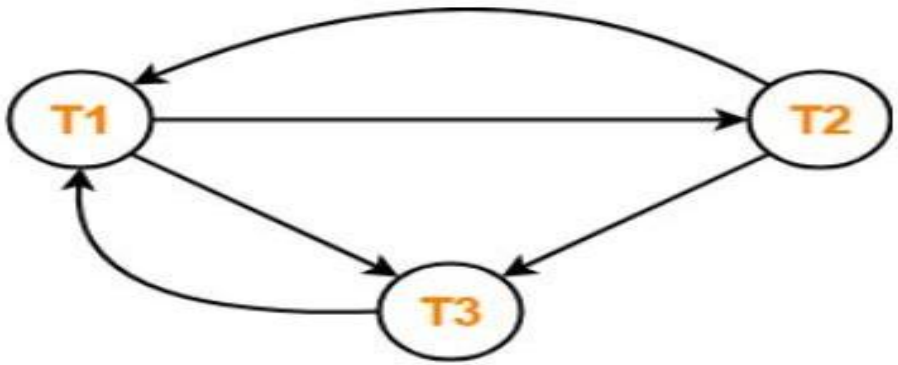
Step-01:

List all the conflicting operations and determine the dependency between the transactions-

- $R_1(A) , W_2(A)$ $(T_1 \rightarrow T_2)$
- $R_1(A) , W_3(A)$ $(T_1 \rightarrow T_3)$
- $W_2(A) , R_3(A)$ $(T_2 \rightarrow T_3)$
- $W_2(A) , W_1(A)$ $(T_2 \rightarrow T_1)$
- $W_2(A) , W_3(A)$ $(T_2 \rightarrow T_3)$
- $R_3(A) , W_1(A)$ $(T_3 \rightarrow T_1)$
- $W_1(A) , W_3(A)$ $(T_1 \rightarrow T_3)$

Step-02:

Draw the precedence graph-



- Clearly, there exists a cycle in the precedence graph.
- Therefore, the given schedule S is not conflict serializable.

Now,

- Since, the given schedule S is not conflict serializable, so, it may or may not be view serializable.
- To check whether S is view serializable or not, let us use another method.
- Let us check for blind writes.

Checking for Blind Writes-

- There exists a blind write W_2 (A) in the given schedule S.
- Therefore, the given schedule S may or may not be view serializable.

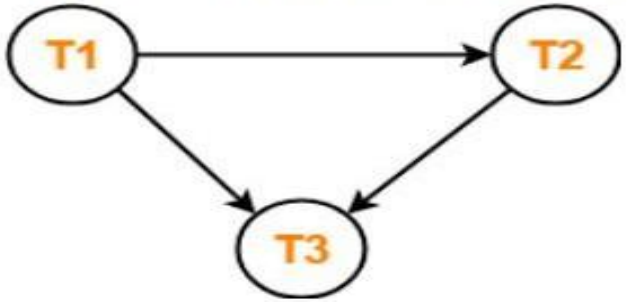
Now,

- To check whether S is view serializable or not, let us use another method.
- Let us derive the dependencies and then draw a dependency graph.

Drawing a Dependency Graph-

- T1 firstly reads A and T2 firstly updates A.
- So, T1 must execute before T2.
- Thus, we get the dependency **T1 → T2**.
- Final updation on A is made by the transaction T3.
- So, T3 must execute after all other transactions.
- Thus, we get the dependency **(T1, T2) → T3**.
- From write-read sequence, we get the dependency **T2 → T3**

Now, let us draw a dependency graph using these dependencies-

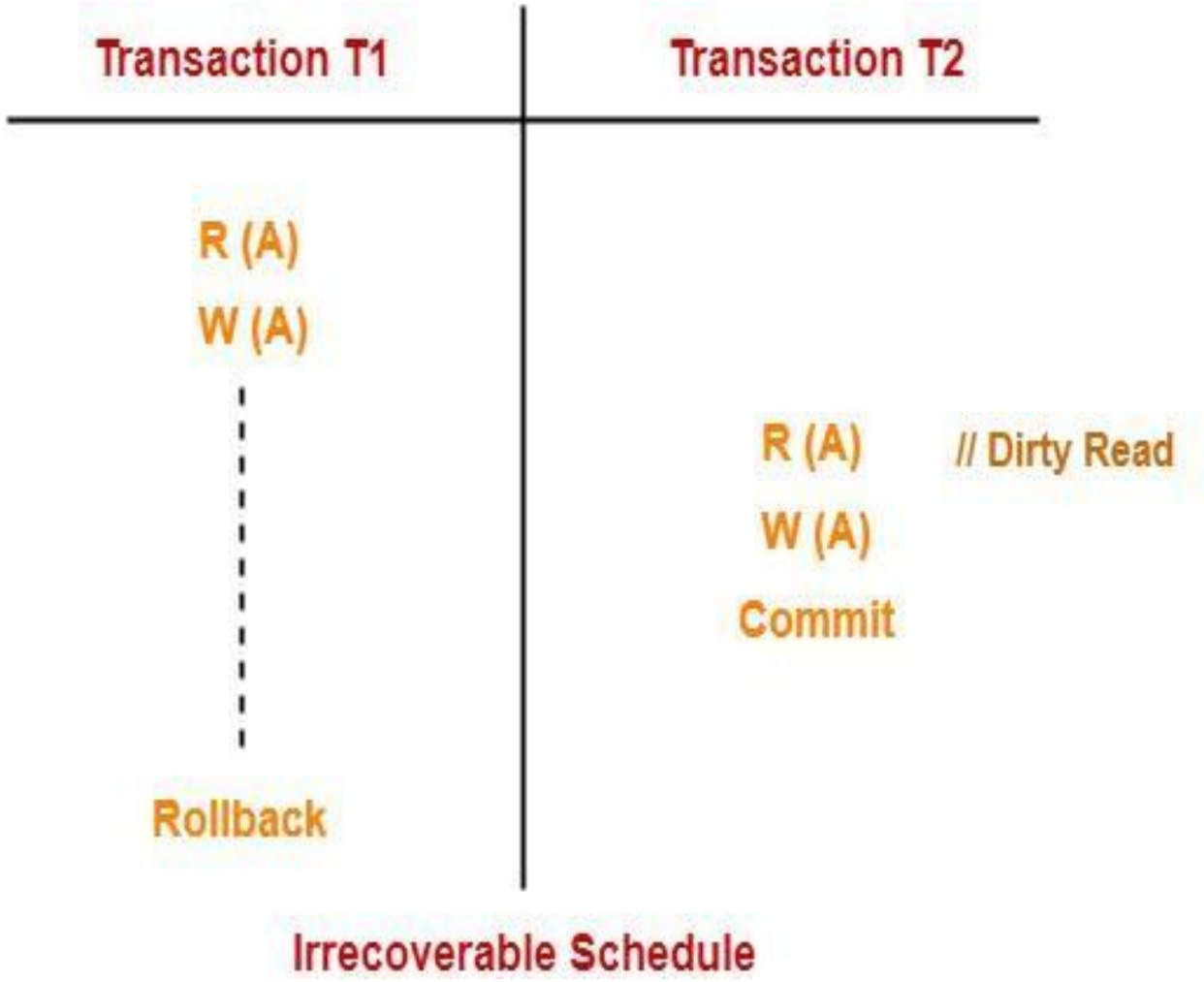


- Clearly, there exists no cycle in the dependency graph.
- Therefore, the given schedule S is view serializable.
- The serialization order T1 → T2 → T3.

Conflict Serializability	View Serializability
If a schedule can be transformed into a serial schedule by performing non-conflicting operations, then it is said to have conflict serializability.	If a schedule is view equal to a serial schedule i.e. no conflicting transactions, then it is said to have view serializability.
To check Conflict Serializability, check if actions are conflicting or not. Actions are said to be conflicting when in different transactions, write operations are involved and transaction values of the same object are changed.	To check for view serializablity, we need to first check if schedule is view equivalent to serial schedule.
Every conflict serializable is view serializable.	But, reverse is not true.
Conflict Serializability is easy to achieve.	View Serializablity is a NP-Hard Problem and hence, hard to achieve.
Conflict serializability is a subset of view serializability.	View serializability is a superset of conflict serializability.
In most of concurrency control schemes, conflict serializability is used.	View Serializablity is rarely used.

Recoverable
vs
Irrecoverable Schedules

If in a schedule, a transaction performs a dirty read operation from an uncommitted transaction & commits before the transaction from which it has read the value then such a schedule is known as an Irrecoverable Schedule.

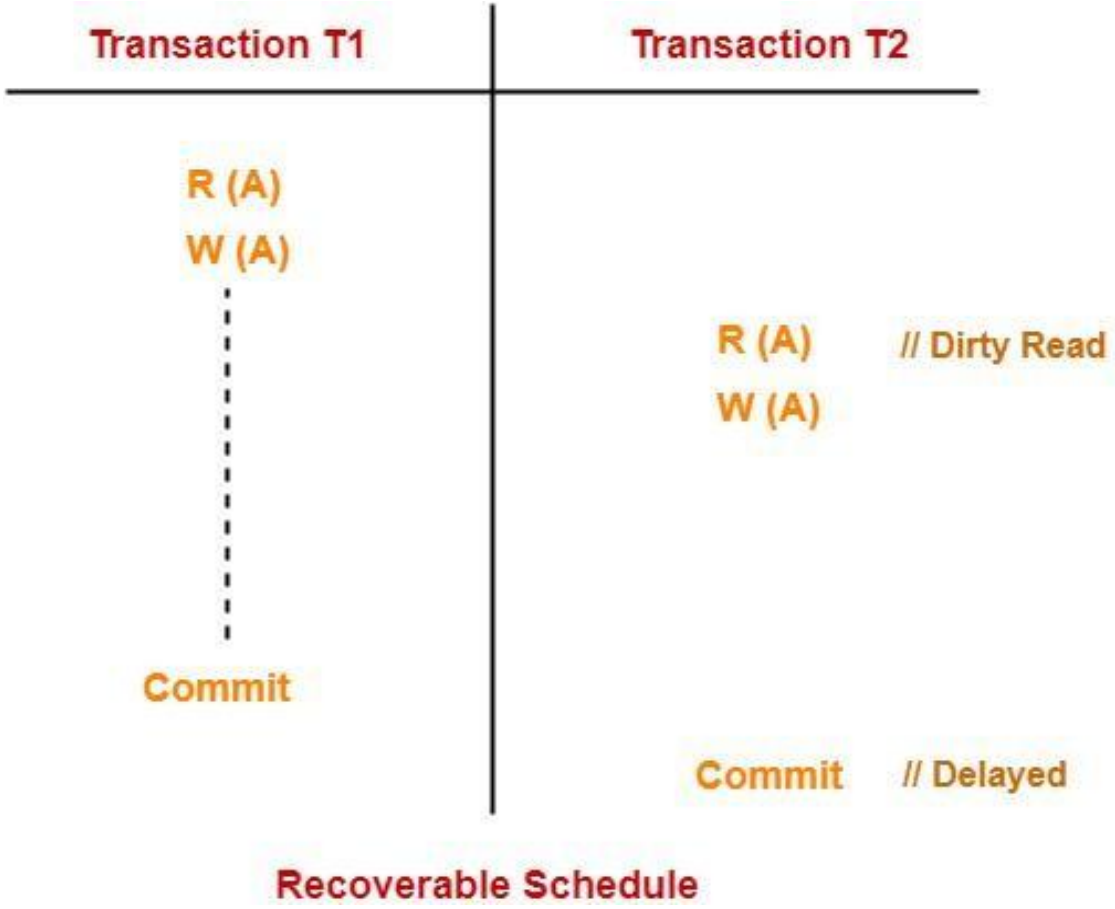


Irrecoverable Schedule

- T2 performs a dirty read operation.
- T2 commits before T1.
- T1 fails later and roll backs.
- The value that T2 read now stands to be incorrect.
- T2 can not recover since it has already committed.

If in a schedule, a transaction performs a dirty read operation from an uncommitted transaction & its commit operation is delayed till the uncommitted transaction either commits or roll backs then such a schedule is known as a Recoverable Schedule.

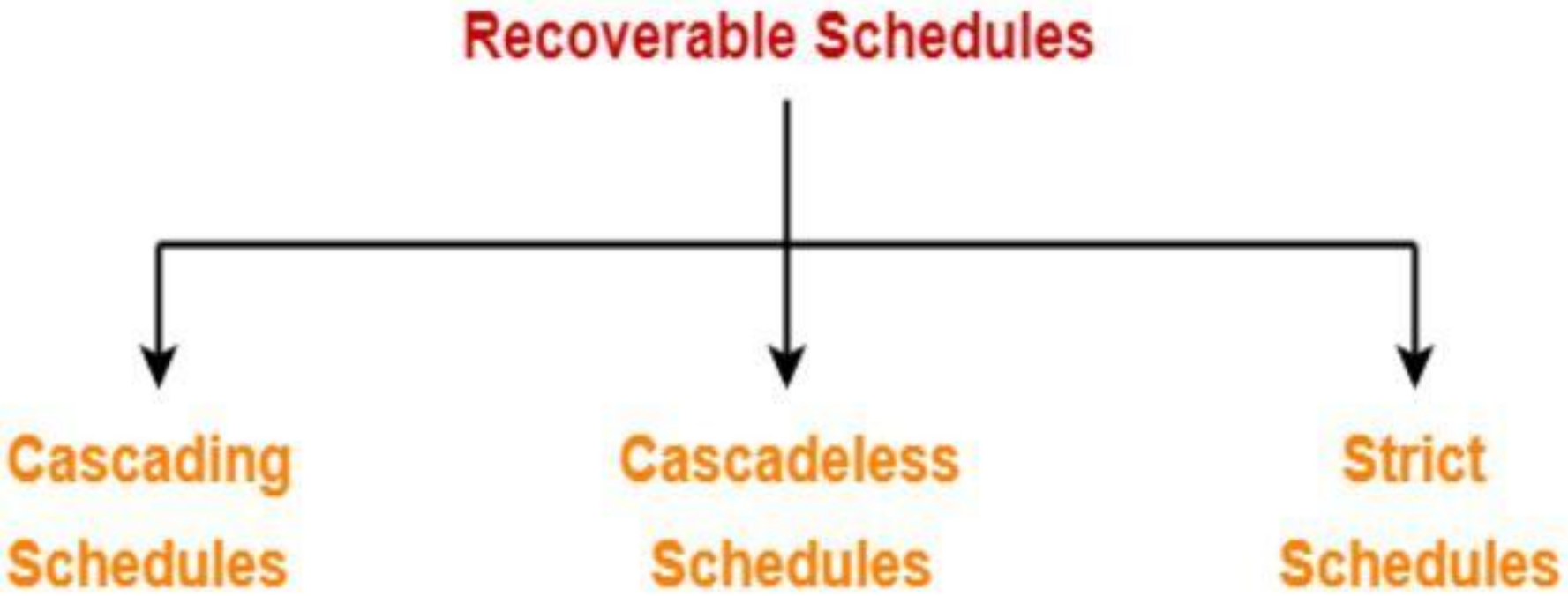
Here, The commit operation of the transaction that performs the dirty read is delayed. This ensures that it still has a chance to recover if the uncommitted transaction fails later.



Recoverable Schedule

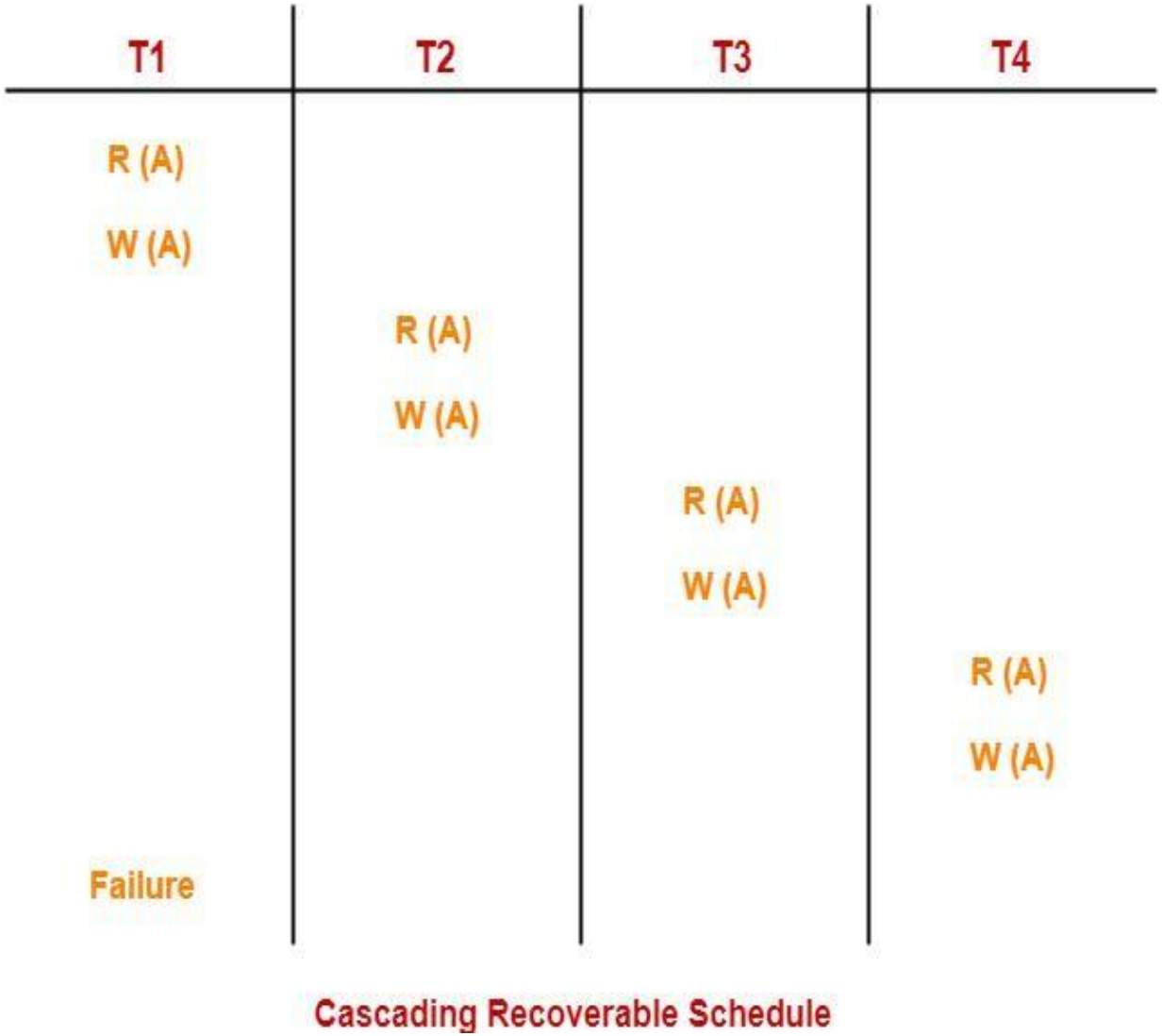
- T2 performs a dirty read operation.
- The commit operation of T2 is delayed till T1 commits or roll backs.
- T1 commits later.
- T2 is now allowed to commit.
- In case, T1 would have failed, T2 has a chance to recover by rolling back.

All conflict serializable schedules are recoverable.


Recoverable Schedules
Cascading Schedules
Cascadeless Schedules
Strict Schedules

Cascading Schedule:

If in a schedule, failure of one transaction causes several other dependent transactions to rollback or abort, then such a schedule is called as a Cascading Schedule or Cascading Rollback or Cascading Abort. It simply leads to the wastage of CPU time.



Cascading Recoverable Schedule

Here,
Transaction T2 depends on transaction T1.
Transaction T3 depends on transaction T2.
Transaction T4 depends on transaction T3.

In this schedule,
The failure of transaction T1 causes the transaction T2 to rollback.
The rollback of transaction T2 causes the transaction T3 to rollback.
The rollback of transaction T3 causes the transaction T4 to rollback.
Such a rollback is called as a **Cascading Rollback.**

If the transactions T2, T3 and T4 would have committed before the failure of transaction T1, then the schedule would have been irrecoverable.

Cascadeless Schedule:

If in a schedule, a transaction is not allowed to read a data item until the last transaction that has written it is committed or aborted, then such a schedule is called as a Cascadeless Schedule.

In other words, Cascadeless schedule allows only committed read operations. Hence, it avoids cascading roll back and thus saves CPU time.

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

Cascadeless Schedule

Cascadeless schedule allows only committed read operations. However, it allows uncommitted write operations.

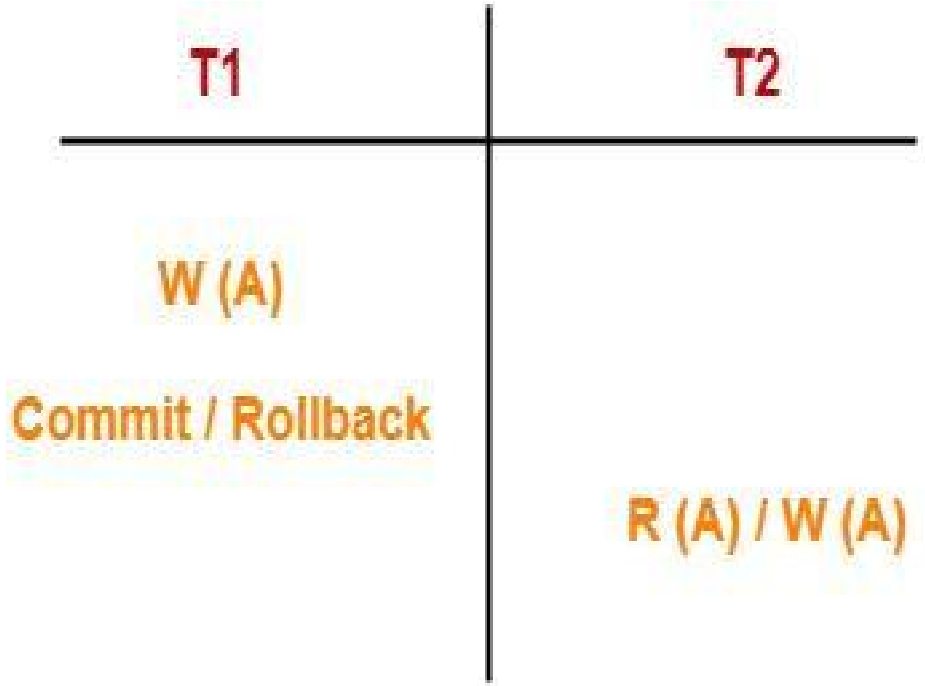
T1	T2
R (A)	
W (A)	
	W (A) // Uncommitted Write
Commit	

Cascadeless Schedule

Strict Schedule:

If in a schedule, a transaction is neither allowed to read nor write a data item until the last transaction that has written it is committed or aborted, then such a schedule is called as a Strict Schedule.

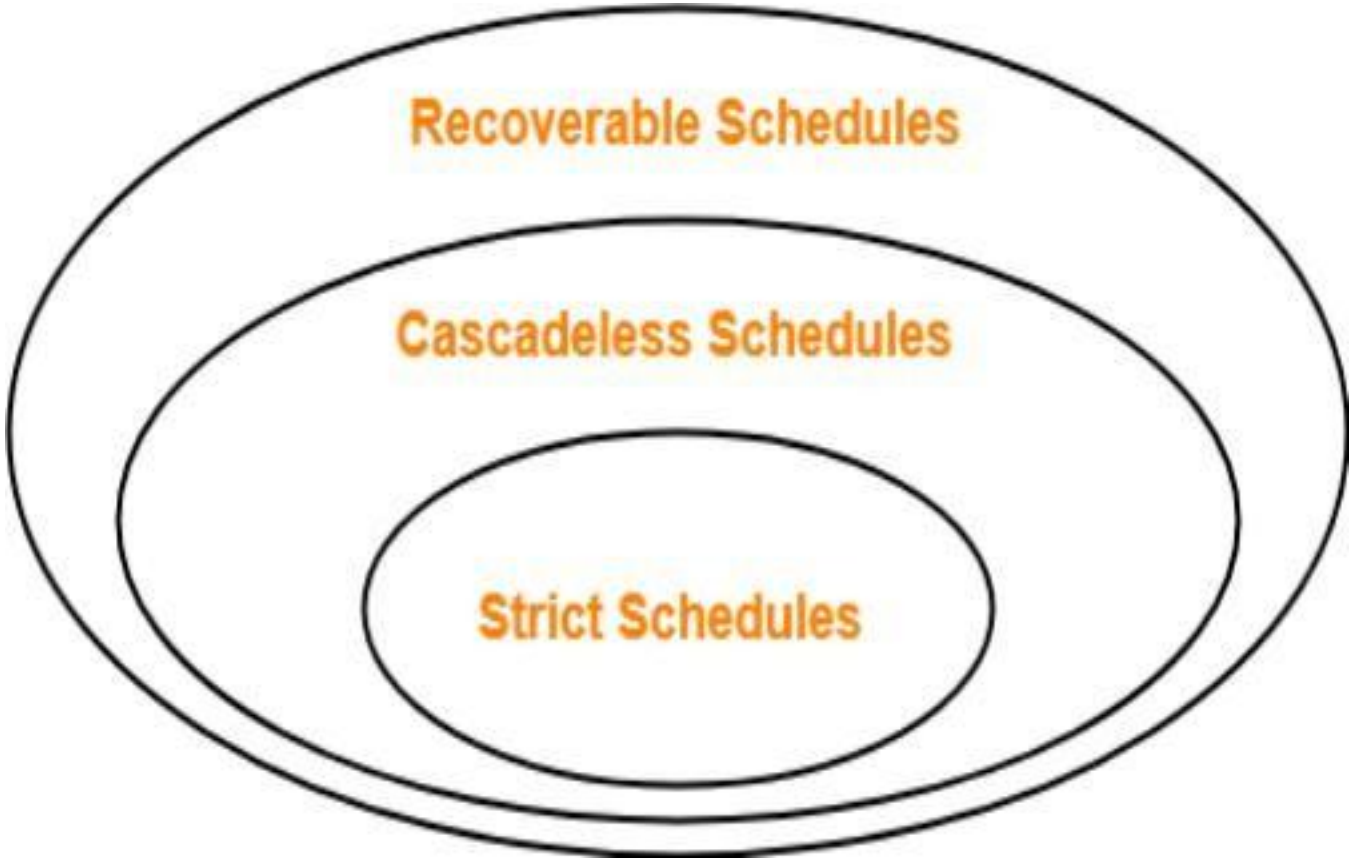
In other words, Strict schedule allows only committed read and write operations. Clearly, strict schedule implements more restrictions than cascadeless schedule.



Strict Schedule

Remember-

- Strict schedules are more strict than cascadeless schedules.
- All strict schedules are cascadeless schedules.
- All cascadeless schedules are not strict schedules.




The End