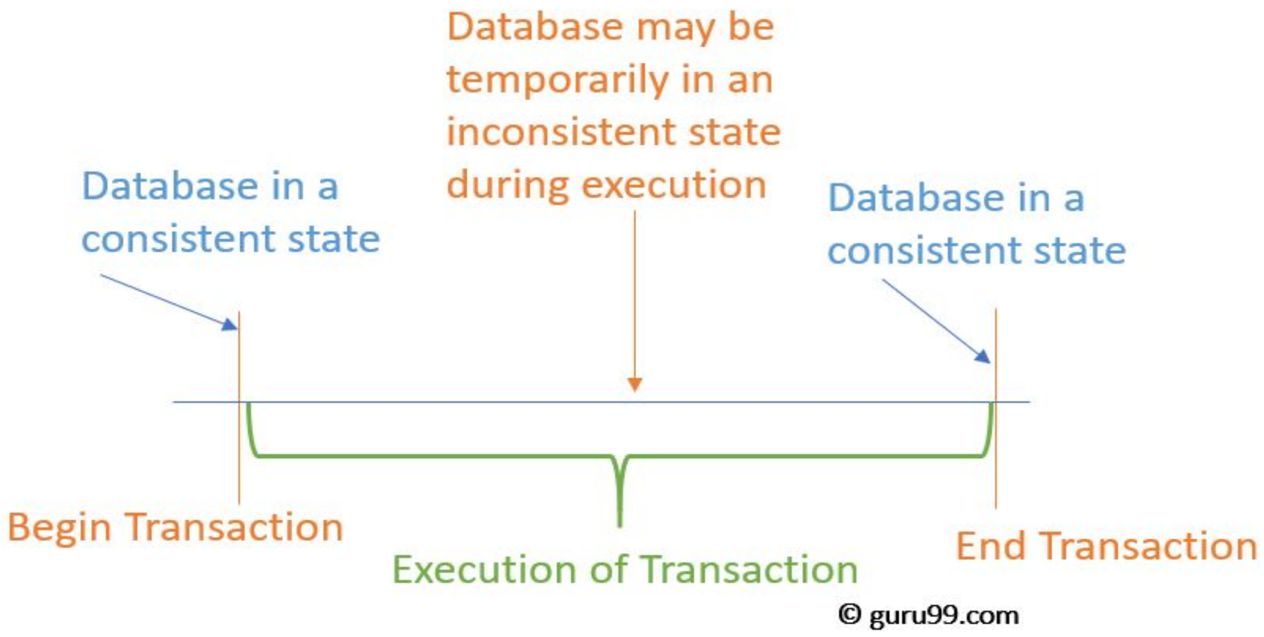
Transaction processing concepts

Transaction

- The transaction is a set of logically related operation. It contains a group of tasks.
- A transaction is an action or series of actions. It is performed by a single user to perform operations for accessing the contents of the database.
- During the transaction the database is inconsistent. Only once the database is committed the state is changed from one consistent state to another.


Database may be temporarily in an inconsistent state during execution
Database in a consistent state
Database in a consistent state
Begin Transaction
Execution of Transaction
End Transaction
© guru99.com

Let's take an example of a simple transaction. Suppose a bank employee transfers Rs 500 from A's account to B's account. This very simple and small transaction involves several low-level tasks.

A's Account

```
Open_Account(A)
Old_Balance = A.balance
New_Balance = Old_Balance - 500
A.balance = New_Balance
Close_Account(A)
```

B's Account

```
Open_Account(B)
Old_Balance = B.balance
 New_Balance = Old_Balance + 500
B.balance = New_Balance
Close_Account(B)
```

Operations of Transaction:

Following are the main operations of transaction:

Read(X): Read operation is used to read the value of X from the database and stores it in a buffer in main memory.

Write(X): Write operation is used to write the value back to the database from the buffer.

Commit: It is used to save the work done permanently.

Rollback: It is used to undo the work done.

Let's take an example to debit transaction from an account which consists of following operations:

1. R(X);
2. X = X - 500;
3. W(X);

Let's assume the value of X before starting of the transaction is 4000.

- The first operation reads X's value from database and stores it in a buffer.
- The second operation will decrease the value of X by 500. So buffer will contain 3500.
- The third operation will write the buffer's value to the database. So X's final value will be 3500.

ACID Properties

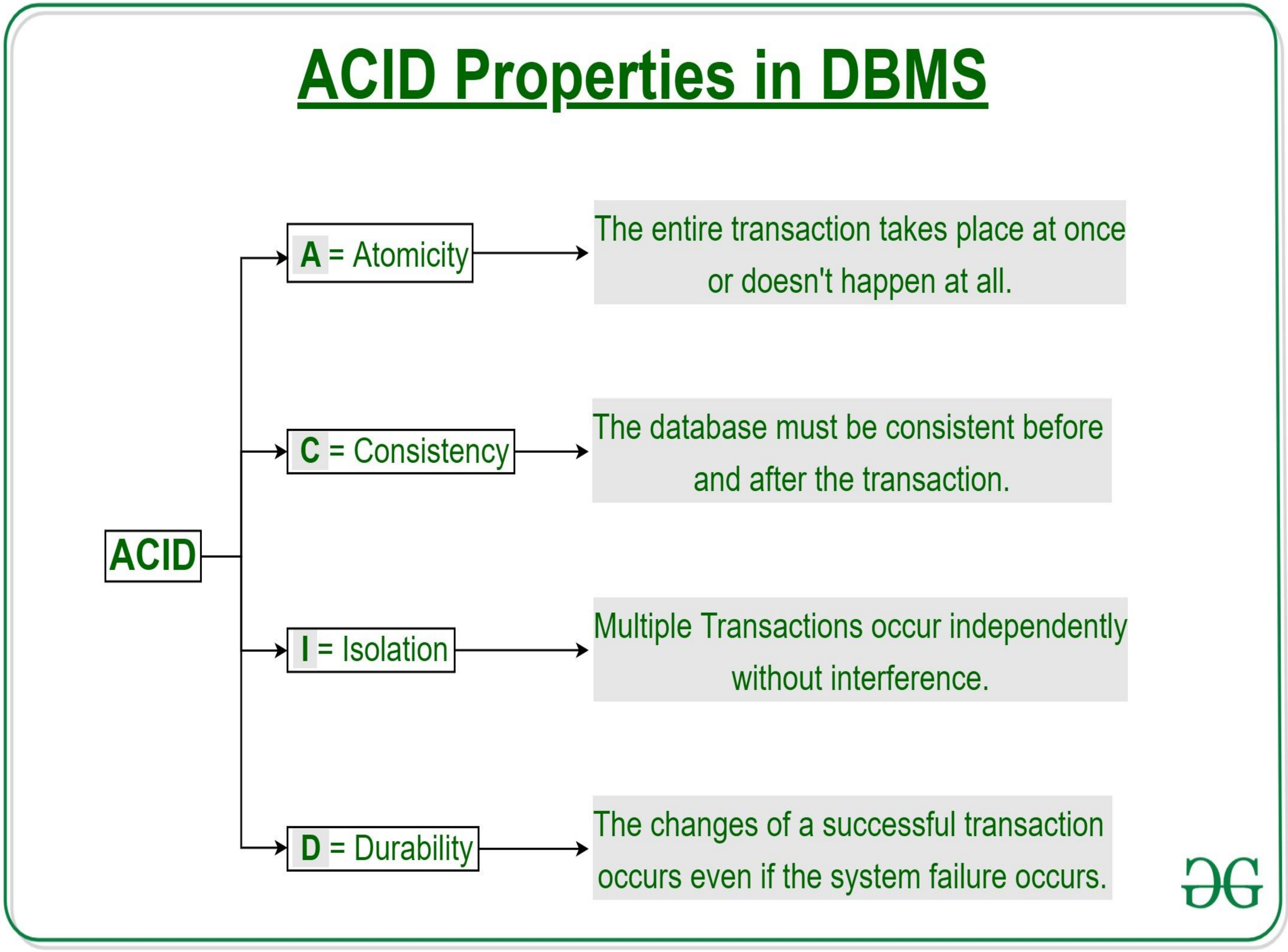
ACID Properties are used for maintaining the integrity of database during transaction processing. ACID in DBMS stands for **A**tomicity, **C**onsistency, **I**solation, and **D**urability.

Atomicity: A transaction is a single unit of operation. You either execute it entirely or do not execute it at all. There cannot be partial execution.

Consistency: Once the transaction is executed, it should move from one consistent state to another.

Isolation: A transaction should be executed as if it is the only one in the system. There should not be any interference from the other concurrent transactions that are simultaneously running.

Durability: · After successful completion of a transaction, the changes in the database should persist. Even in the case of system failures.



ACID Properties in DBMS

A = Atomicity	The entire transaction takes place at once or doesn't happen at all.
C = Consistency	The database must be consistent before and after the transaction.
I = Isolation	Multiple Transactions occur independently without interference.
D = Durability	The changes of a successful transaction occurs even if the system failure occurs.

ACID Property in DBMS with example:

Below is an example of ACID property in DBMS:

Transaction 1: Begin X=X+50, Y = Y-50 END

Transaction 2: Begin X=1.1*X, Y=1.1*Y END

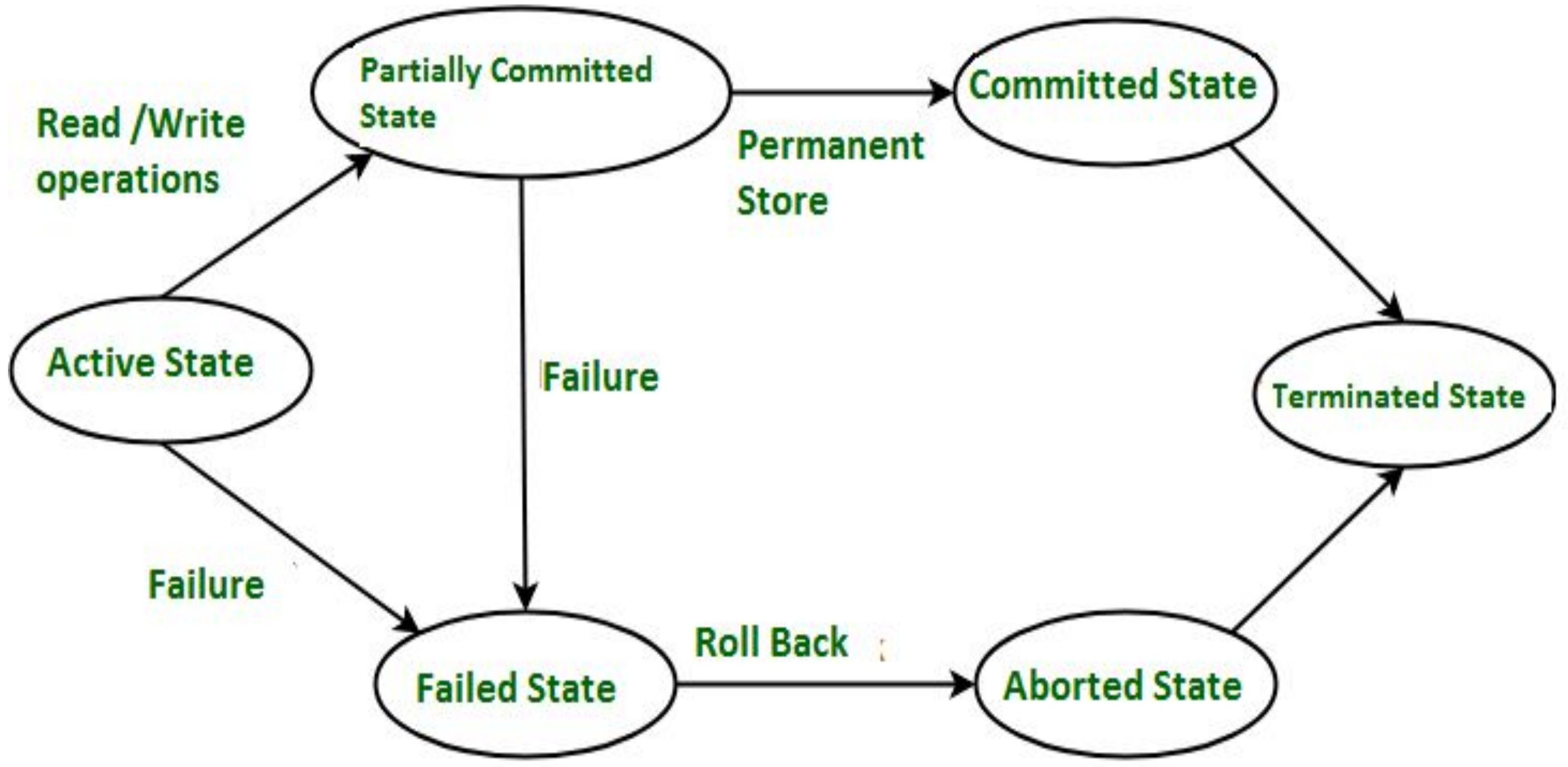
Transaction 1 is transferring $50 from account X to account Y.

Transaction 2 is crediting each account with a 10% interest payment.

If both transactions are submitted together, there is no guarantee that the Transaction 1 will execute before Transaction 2 or vice versa. Irrespective of the order, the result must be as if the transactions take place serially one after the other.

Transaction States in DBMS

States through which a transaction goes during its lifetime.



Transaction States in DBMS

Active State

A transaction enters into an active state when the execution process begins. During this state read or write operations can be performed.

Partially Committed

A transaction goes into the partially committed state after the end of a transaction.

Committed State

When the transaction is committed to state, it has already completed its execution successfully. Moreover, all of its changes are recorded to the database permanently.

Failed State

A transaction considers failed when any one of the checks fails or if the transaction is aborted while it is in the active state.

Aborted state

- If any of the checks fail and the transaction has reached a failed state then the database recovery system will make sure that the database is in its previous consistent state. If not then it will abort or roll back the transaction to bring the database into a consistent state.
- If the transaction fails in the middle of the transaction then before executing the transaction, all the executed transactions are rolled back to its consistent state.
- After aborting the transaction, the database recovery module will select one of the two operations:
    1. Re-start the transaction
    2. Kill the transaction

Terminated State

State of transaction reaches terminated state when certain transactions which are leaving the system can't be restarted.

Concurrent Execution of Transaction

In the transaction process, a system usually allows executing more than one transaction simultaneously. This process is called a concurrent execution. We use the technique of **interleaving** execution of two transactions. For example interleaving is shown below.

T1	T2

Read(A);
Write(A);	

	Read(B);
	Write(B);

Read(C);
Write(C);	

Thus, firstly ,T1 executes ,then T2 executes and then T1 again. This is interleaving This way multiple transactions are allowed to run concurrently in the system.

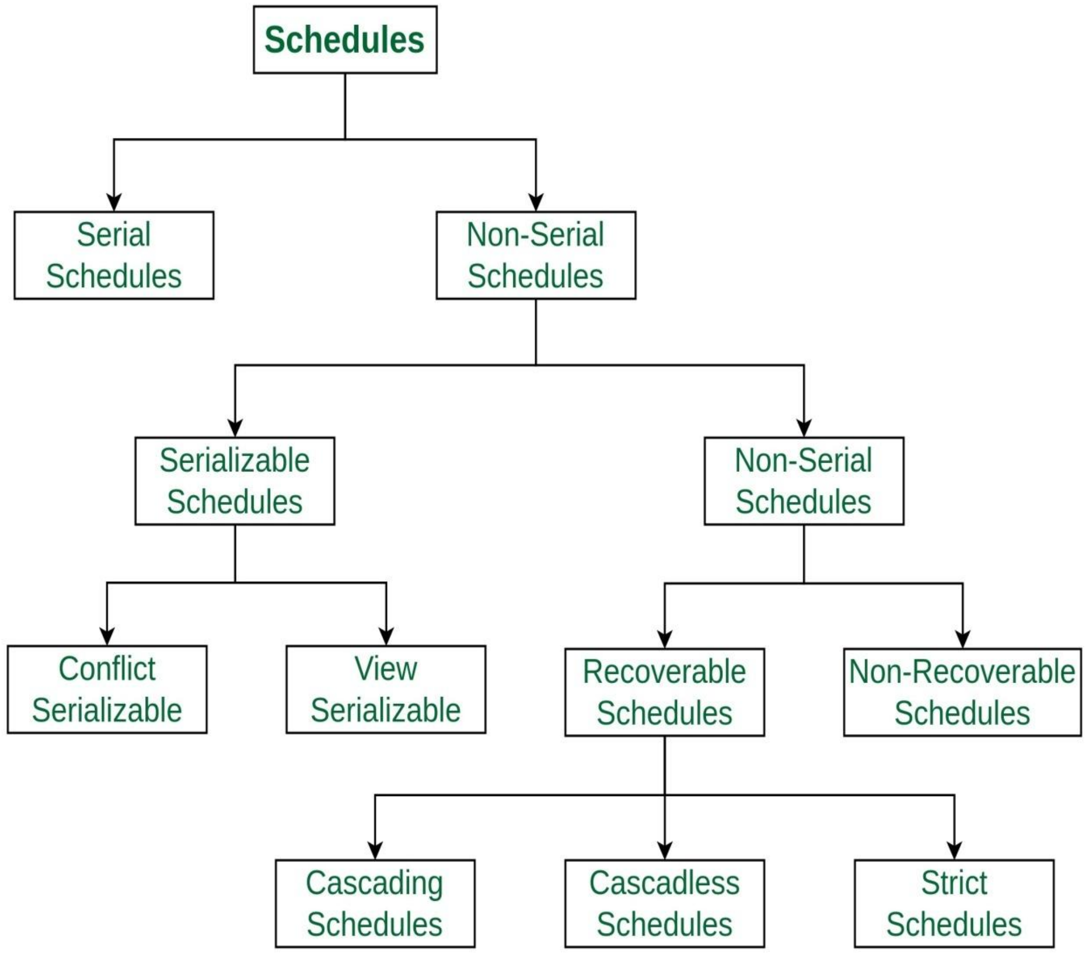
Advantages of concurrent execution of a transaction

- Decrease waiting time or turnaround time.
- Improve response time
- Increased throughput or resource utilization.

Schedules of transactions

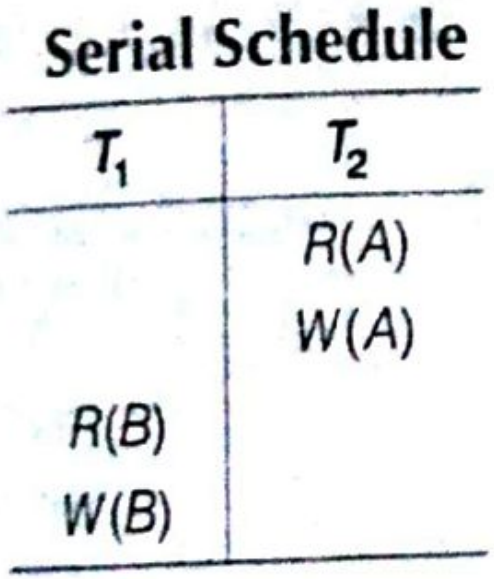
- a sequences of instructions that specify the chronological order in which instructions of concurrent transactions are executed
  - a schedule for a set of transactions must consist of all instructions of those transactions
  - must preserve the order in which the instructions appear in each individual transaction.
- A transaction that successfully completes its execution will have a
- commit instructions as the last statement
  - by default transaction assumed to execute commit instruction as its last step
- A transaction that fails to successfully complete its execution will have an abort instruction as the last statement

Types of schedules in DBMS



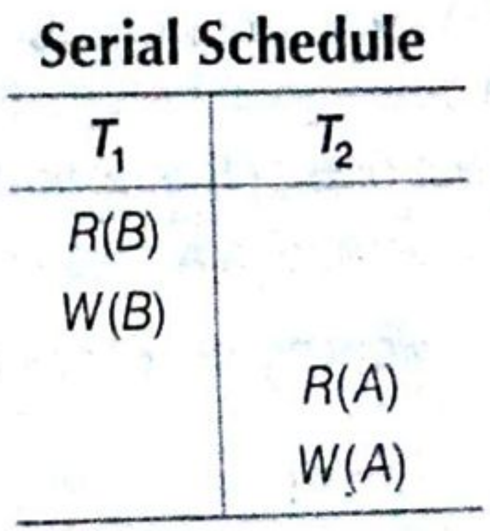
Serial schedules:

- All the transactions execute serially one after the other.
- When one transaction executes, no other transaction is allowed to execute.



Serial Schedule

T_1	T_2
	R(A)
	W(A)
R(B)	
W(B)	



Serial Schedule

T_1	T_2
R(B)	
W(B)	
	R(A)
	W(A)

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 (B)	
	R (A)
R (B)	
W (B)	
Commit	
	R (B)
	Commit

Read Write Conflict in DBMS

This problem occurs when two or more read operations of the same transaction read different values of the same variable.

T1	T2
R(A) A=2 . . . . R(A) A=0 W(A) commit	R(A) A=2 A-2 W(A) A=0 commit

R(A)	R(A)
W(A)	R(A)
R(A)	W(A)
W(A)	W(A)

Recoverable schedule:

If each transaction commits only after all transactions from which it has read has committed

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

Here T2 is performing read operation on A after the T1 has made changes in A using W(A) so T2 can only commit after the commit operation of T1.

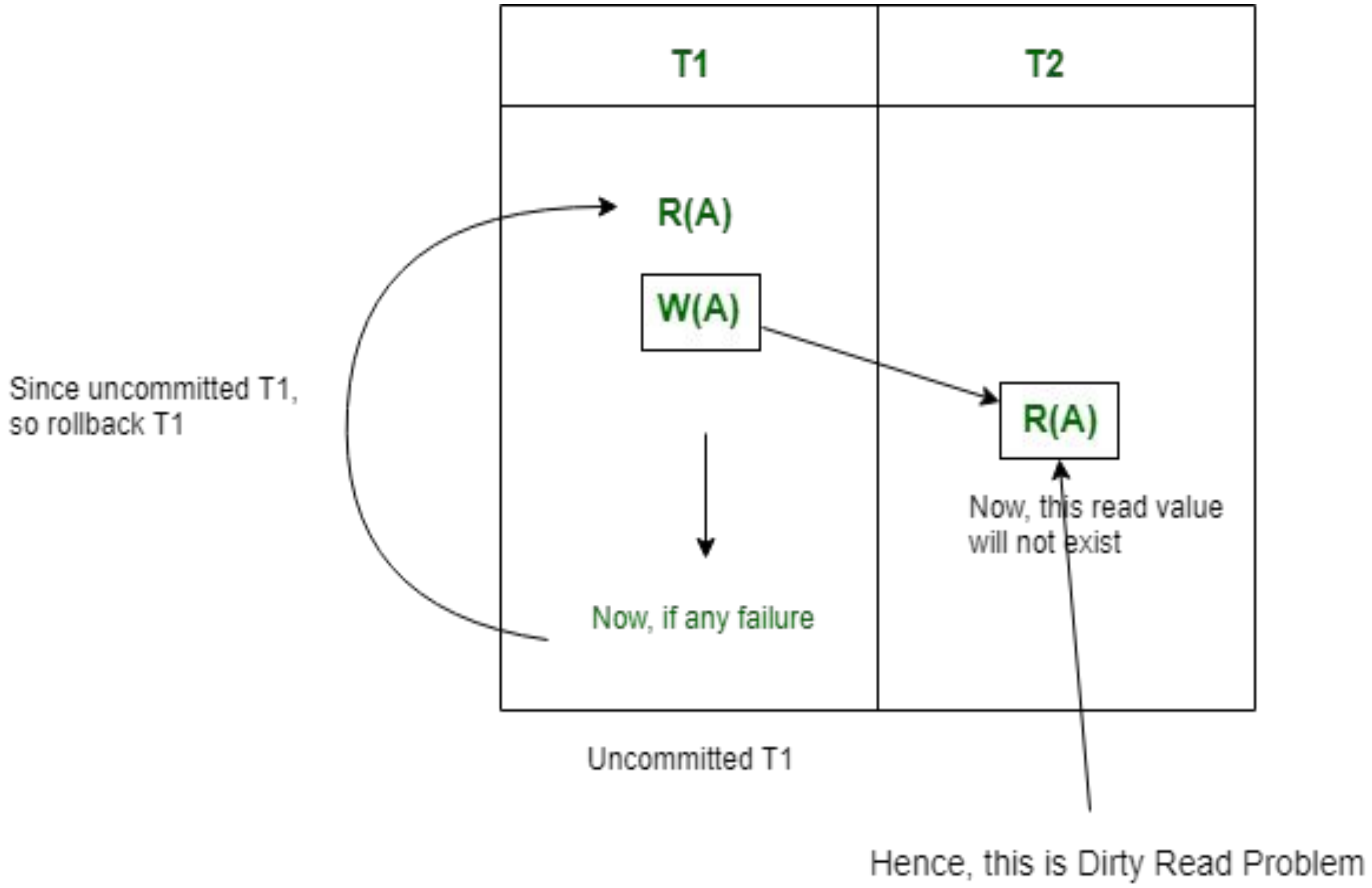
Irrecoverable schedule:

T1	T1's buffer space	T2	T2's buffer space	Database
				A = 6500
Read(A);	A = 6500			A = 6500
A = A - 500;	A = 6000			A = 6500
Write(A);	A = 6000			A = 6000
		Read(A);	A = 6000	A = 6000
		A =A + 1000;	A = 7000	A = 6000
		Write(A);	A = 7000	A = 7000
		Commit;		
Failure Point				
Commit;				

The above table shows a schedule which has two transactions. T1 reads and writes the value of A and that value is read and written by T2. T2 commits but later on, T1 fails. Due to the failure, we have to rollback T1. T2 should also be rollback because it reads the value written by T1, but T2 can't be rollback because it already committed. So this type of schedule is known as irrecoverable schedule

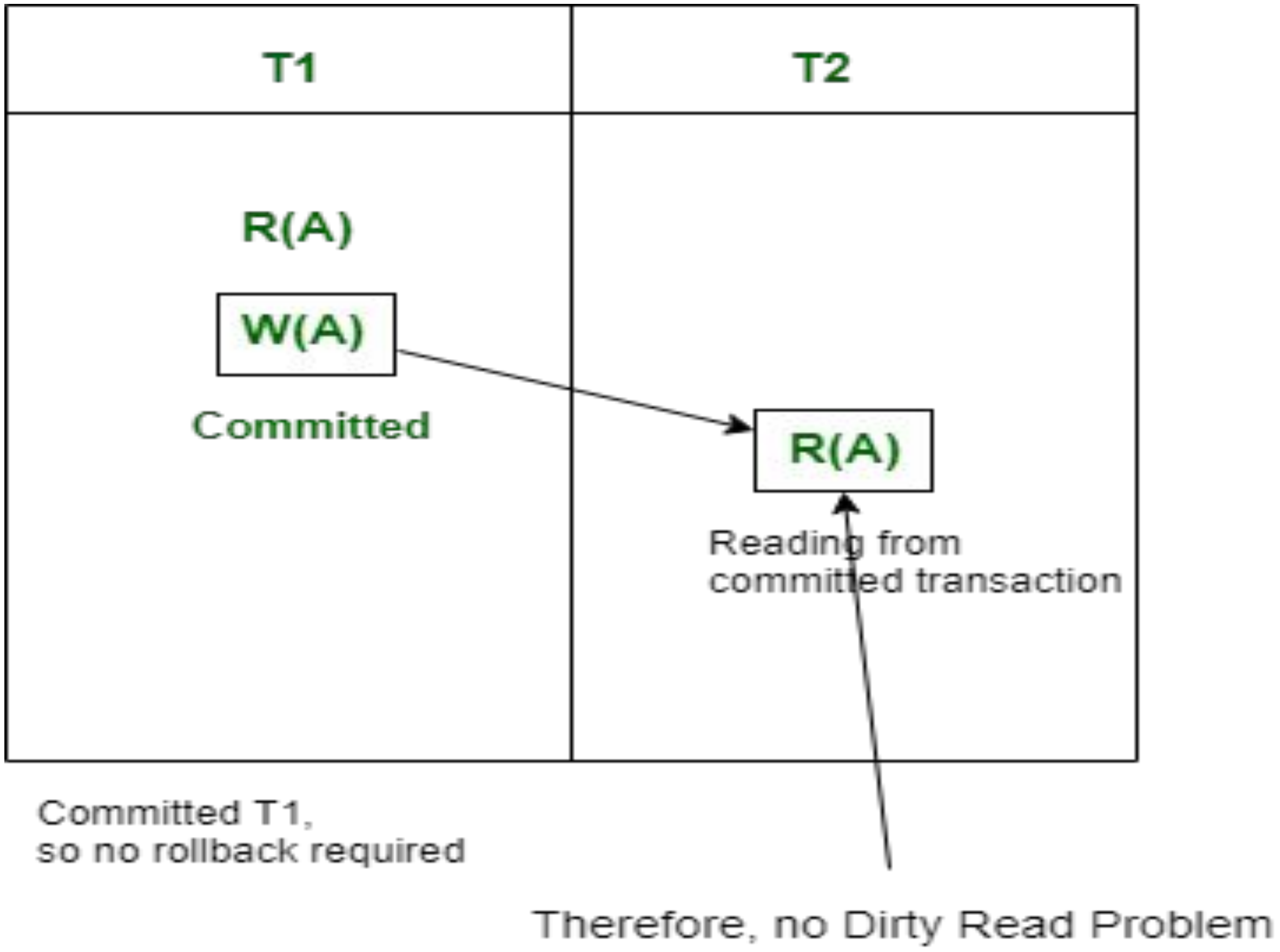
Dirty Reads

When a transaction is allowed to read a row that has been modified by an another transaction which is not committed yet that time Dirty Reads occurred. It is mainly occurred because of multiple transaction at a time which is not committed.



T1
T2
R(A)
W(A)
R(A)
Since uncommitted T1, so rollback T1
Now, this read value will not exist
Now, if any failure
Uncommitted T1
Hence, this is Dirty Read Problem

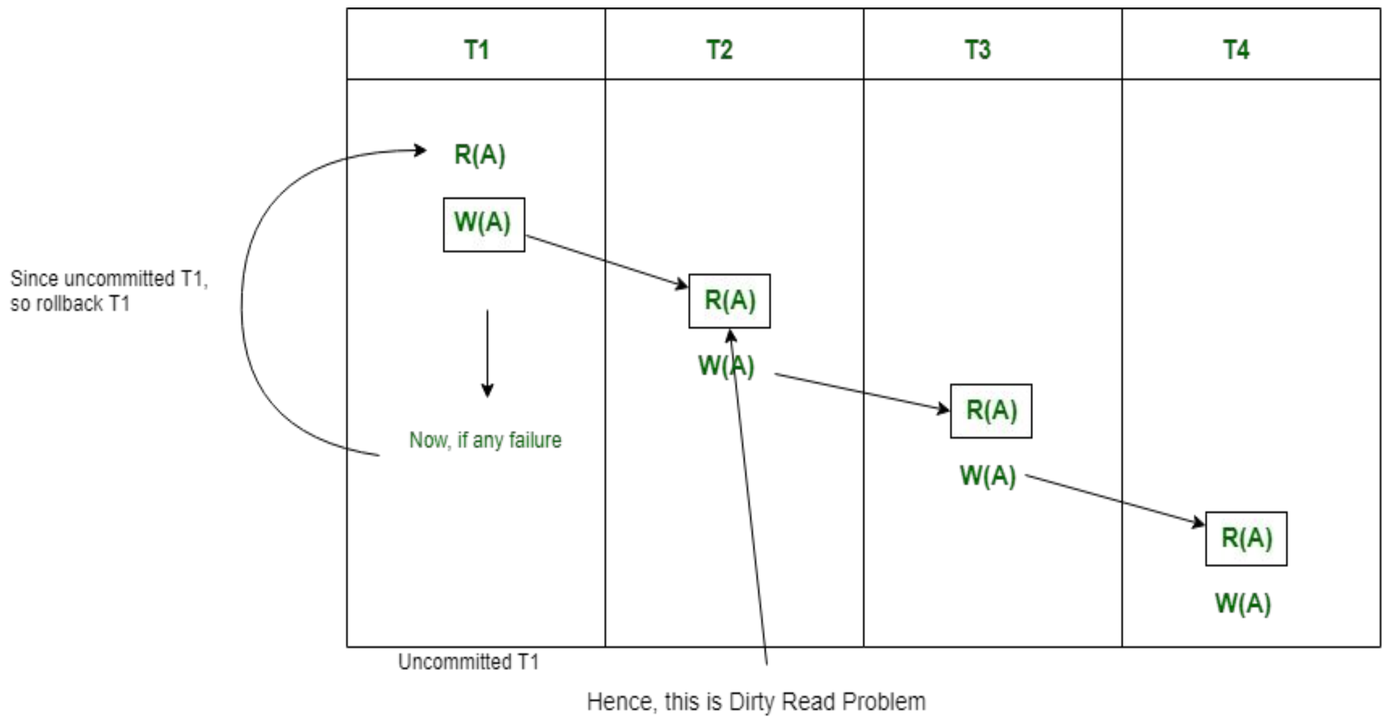
There is no Dirty Read problem, if a transaction is reading from another committed transaction. So, no rollback required.


T1
T2
R(A)
W(A)
Committed
R(A)
Reading from committed transaction
Committed T1, so no rollback required
Therefore, no Dirty Read Problem

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 Rollback or Cascading Abort or Cascading Schedule. It simply leads to the wastage of CPU time.

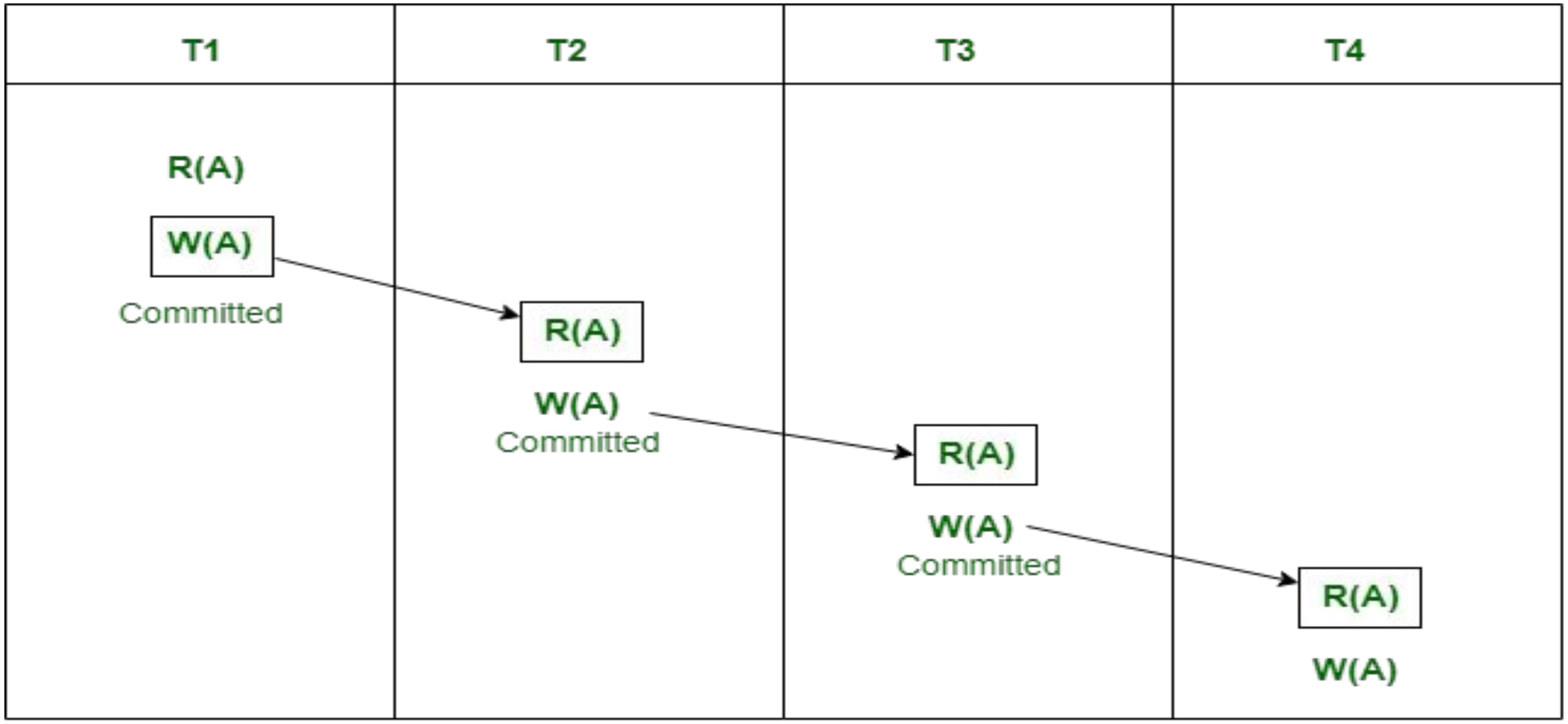
These Cascading Rollbacks occur because of **Dirty Read problems**.



For example, transaction T1 writes uncommitted x that is read by Transaction T2. Transaction T2 writes uncommitted x that is read by Transaction T3. Suppose at this point T1 fails. T1 must be rolled back, since T2 is dependent on T1, T2 must be rolled back, and since T3 is dependent on T2, T3 must be rolled back.
Because of T1 rollback, all T2, T3, and T4 should also be rollback (Cascading dirty read problem).
This phenomenon, in which a single transaction failure leads to a series of transaction rollbacks is called **Cascading rollback**.

Cascadeless Schedule:

In Cascadeless Schedule, if a transaction is going to perform read operation on a value, it has to wait until the transaction who is performing write on that value commits. That means there must not be **Dirty Read**.



No Dirty Read problem

Serializability :

- Serializability is the process of search for a concurrent schedule whose output is equal to a serial schedule where transactions are executed one after the other.
- Depending on the type of schedules, there are two types of serializability:
    - Conflict
    - View

Serial schedule

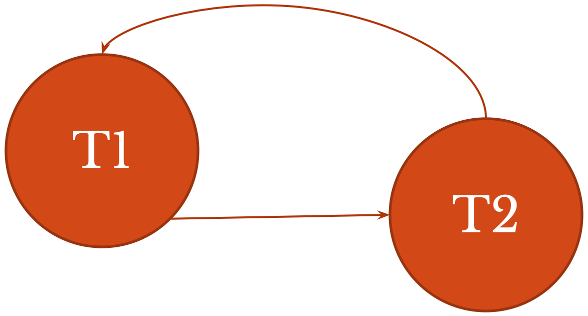
T1	T2
R(A)	
W(A)	
	R(A)
	W(A)

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

$$\begin{tikzcd} T1 \arrow[r] & T2 \end{tikzcd}$$

$$\begin{tikzcd} T1 & T2 \arrow[l] \end{tikzcd}$$

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



When loop is created that is not serial schedule.
So we have to analyze that the serial schedule can be created or not from the non serial schedule and if can be created that should be equivalent to non serial schedule.

Conflict Equivalent: Two schedules are said to be conflict equivalent when one can be transformed to another by swapping non-conflicting pair.

R(A)	R(A)	Non conflict pair
R(A)	W(A)	Conflict pair
W(A)	R(A)	
W(A)	W(A)	
R(B)	R(A)	Non conflict pair
W(B)	R(A)	
R(B)	W(A)	
W(B)	W(A)	

s

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

s_1

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

Is schedule s and s_1 is conflict equivalent ?

s

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

Step 1

T1	T2
R(A)	
W(A)	
	R(A)
	W(A) ↓
↑ R(B)	

Step 2

T1	T2
R(A)	
W(A)	
	R(A) ↓
↑ R(B)	
	W(A)

s_1

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

Swap adjacent non conflict pair

$s \equiv s_1$
Schedule s is conflict equivalent to s_1

Conflict Serializability

A schedule is called conflict serializable if we can convert it into a serial schedule after swapping its non-conflicting operations.

Conflicting operations

Two operations are said to be in conflict, if they satisfy all the following three conditions:

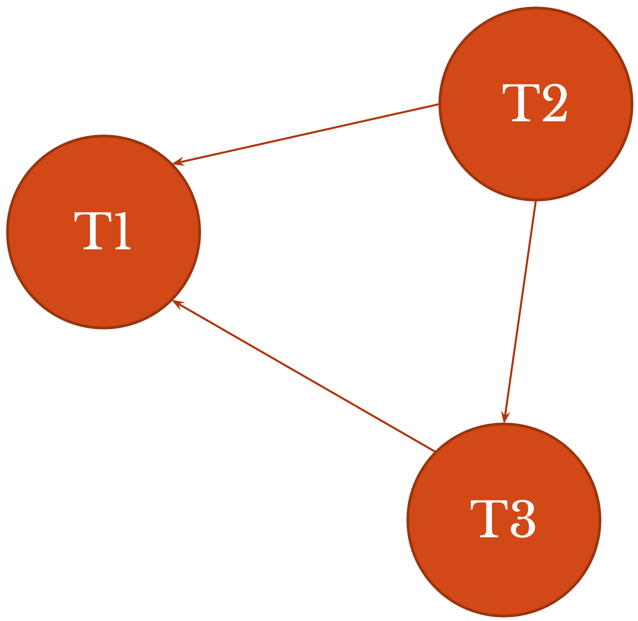
1. Both the operations should belong to different transactions.
2. Both the operations are working on same data item.
3. At least one of the operation is a write operation.

Testing Conflict Serializability

Check conflict pair in other transaction and draw edges

If no loop/no cycle then Conflict

T1	T2	T3
R(x)		
		R(y)
		R(x)
	R(y)	
	R(z)	
		W(y)
	W(z)	
R(z)		
W(x)		
W(z)		



There is no loop/no cycle then this schedule is Conflict Serializable, can be convert to equivalent serial schedule.

Check indegree=0
T2-T3-T1

View Equivalent

Lets learn how to check whether the two schedules are view equivalent.

Two schedules T1 and T2 are said to be view equivalent, if they satisfy all the following conditions:

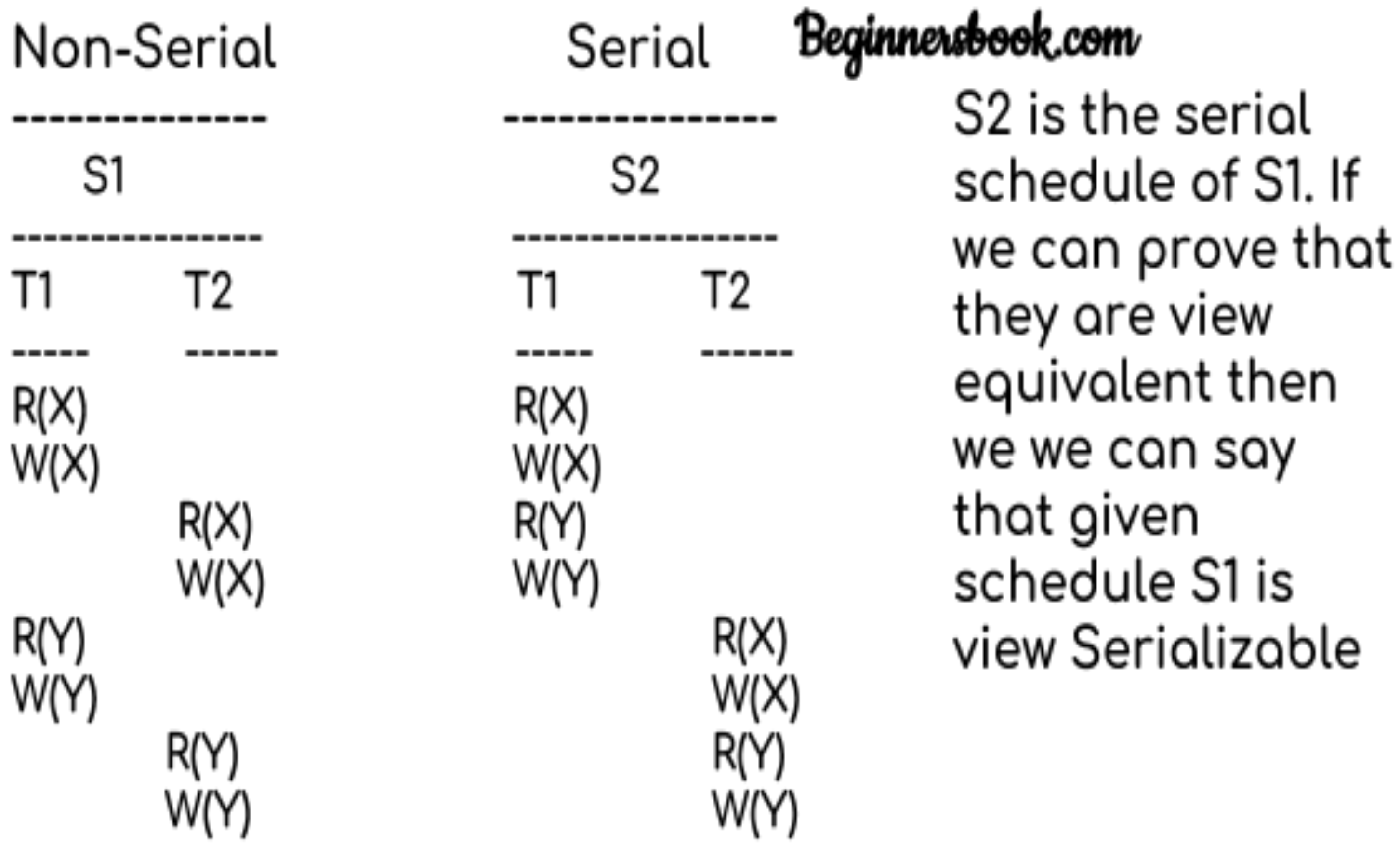
1. **Initial Read:** Initial read of each data item in transactions must match in both schedules. For example, if transaction T1 reads a data item X before transaction T2 in schedule S1 then in schedule S2, T1 should read X before T2.

2. **Final Write:** Final write operations on each data item must match in both the schedules. For example, a data item X is last written by Transaction T1 in schedule S1 then in S2, the last write operation on X should be performed by the transaction T1.

3. **Update Read:** If in schedule S1, the transaction T1 is reading a data item updated by T2 then in schedule S2, T1 should read the value after the write operation of T2 on same data item. For example, In schedule S1, T1 performs a read operation on X after the write operation on X by T2 then in S2, T1 should read the X after T2 performs write on X.

View Serializability

If a schedule is view equivalent to its serial schedule then the given schedule is said to be View Serializable. Lets take an example.



Non-Serial

S1

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

Serial

S2

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

Beginnersbook.com

S2 is the serial schedule of S1. If we can prove that they are view equivalent then we we can say that given schedule S1 is view Serializable

Lets check the three conditions of view serializability:

Initial Read

In schedule S1, transaction T1 first reads the data item X. In S2 also transaction T1 first reads the data item X.

Lets check for Y. In schedule S1, transaction T1 first reads the data item Y. In S2 also the first read operation on Y is performed by T1.

We checked for both data items X & Y and the **initial read** condition is satisfied in S1 & S2.

Final Write

In schedule S1, the final write operation on X is done by transaction T2. In S2 also transaction T2 performs the final write on X.

Lets check for Y. In schedule S1, the final write operation on Y is done by transaction T2. In schedule S2, final write on Y is done by T2.

We checked for both data items X & Y and the **final write** condition is satisfied in S1 & S2.

Update Read

In S1, transaction T2 reads the value of X, written by T1. In S2, the same transaction T2 reads the X after it is written by T1.

In S1, transaction T2 reads the value of Y, written by T1. In S2, the same transaction T2 reads the value of Y after it is updated by T1.

The update read condition is also satisfied for both the schedules.

Result: Since all the three conditions that checks whether the two schedules are view equivalent are satisfied in this example, which means S1 and S2 are view equivalent. Also, as we know that the schedule S2 is the serial schedule of S1, thus we can say that the schedule S1 is view serializable schedule.

Thank You