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.

- **Example:** Suppose an employee of bank transfers Rs 800 from X's account to Y's account. This small transaction contains several low-level tasks:

X's Account

Open_Account(X)

Old_Balance = X.balance

New_Balance = Old_Balance - 800

X.balance = New_Balance

Close_Account(X)

Y's Account

Open_Account(Y)

Old_Balance = Y.balance

New_Balance = Old_Balance + 800

Y.balance = New_Balance

Close_Account(Y)

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.

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.
- But it may be possible that because of the failure of hardware, software or power, etc. that transaction may fail before finished all the operations in the set.

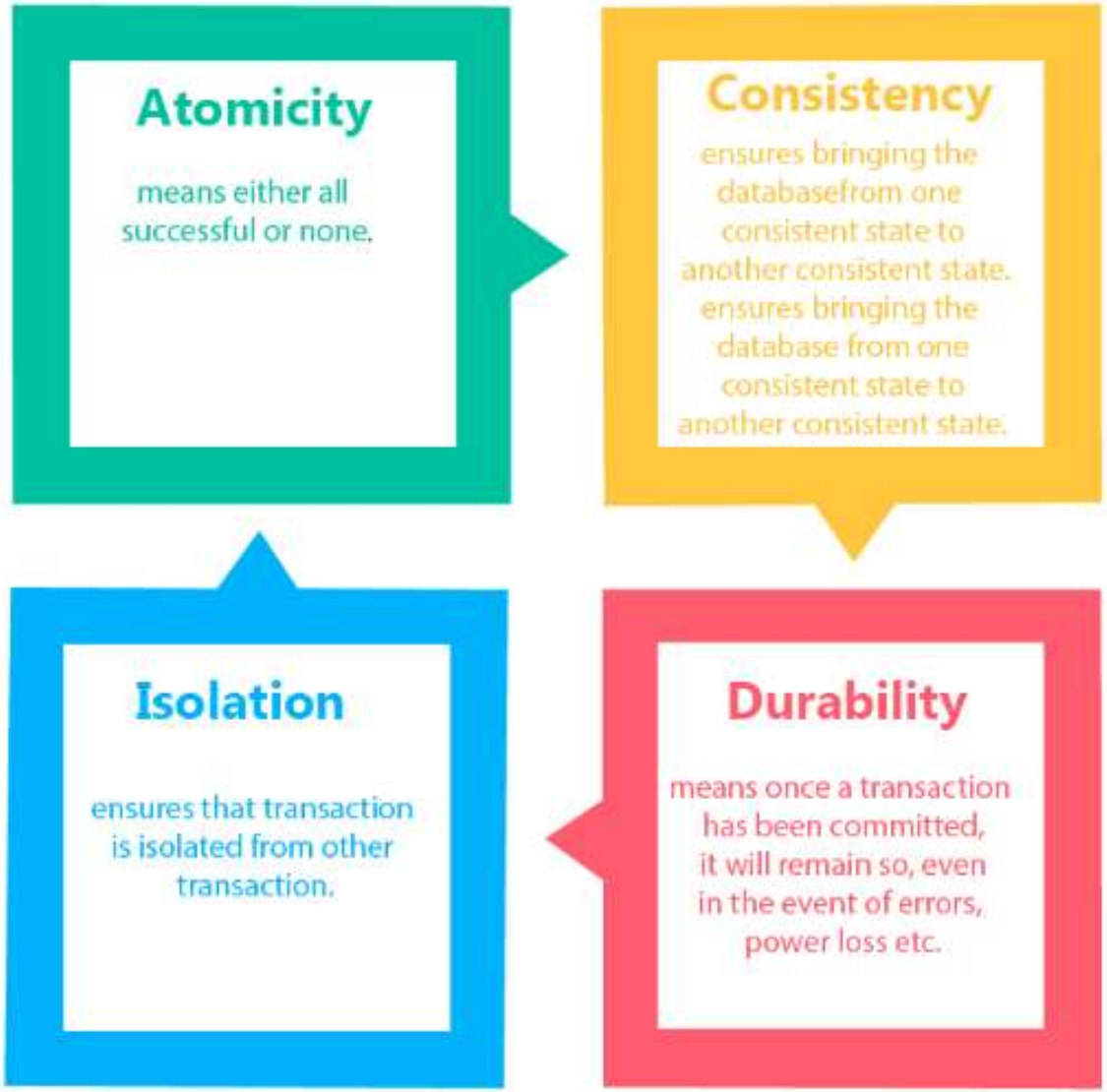
- **For example:** If in the above transaction, the debit transaction fails after executing operation 2 then X's value will remain 4000 in the database which is not acceptable by the bank.
- To solve this problem, we have two important operations:
- **Commit:** It is used to save the work done permanently.
- **Rollback:** It is used to undo the work done.

Transaction property

- The transaction has the four properties. These are used to maintain consistency in a database, before and after the transaction.

Property of Transaction

- Atomicity
- Consistency
- Isolation
- Durability


Atomicity
means either all successful or none.
Consistency
ensures bringing the databasefrom one consistent state to another consistent state. ensures bringing the database from one consistent state to another consistent state.
Isolation
ensures that transaction is isolated from other transaction.
Durability
means once a transaction has been committed, it will remain so, even in the event of errors, power loss etc.

Atomicity

- It states that all operations of the transaction take place at once if not, the transaction is aborted.
- There is no midway, i.e., the transaction cannot occur partially. Each transaction is treated as one unit and either run to completion or is not executed at all.

- Atomicity involves the following two operations:

- **Abort:** If a transaction aborts then all the changes made are not visible.
- **Commit:** If a transaction commits then all the changes made are visible.

Example:

- Let's assume that following transaction T consisting of T1 and T2. A consists of Rs 600 and B consists of Rs 300. Transfer Rs 100 from account A to account B.

T1	T2
Read(A) A:= A-100 Write(A)	Read(B) Y:= Y+100 Write(B)

- After completion of the transaction, A consists of Rs 500 and B consists of Rs 400.
- If the transaction T fails after the completion of transaction T1 but before completion of transaction T2, then the amount will be deducted from A but not added to B.
- This shows the inconsistent database state. In order to ensure correctness of database state, the transaction must be executed in entirety.

Consistency

- The integrity constraints are maintained so that the database is consistent before and after the transaction.
- The execution of a transaction will leave a database in either its prior stable state or a new stable state.
- The consistent property of database states that every transaction sees a consistent database instance.
- The transaction is used to transform the database from one consistent state to another consistent state.

- **For example:** The total amount must be maintained before or after the transaction.

- Total before T occurs = 600+300=900
- Total after T occurs= 500+400=900

- Therefore, the database is consistent. In the case when T1 is completed but T2 fails, then inconsistency will occur.

Isolation

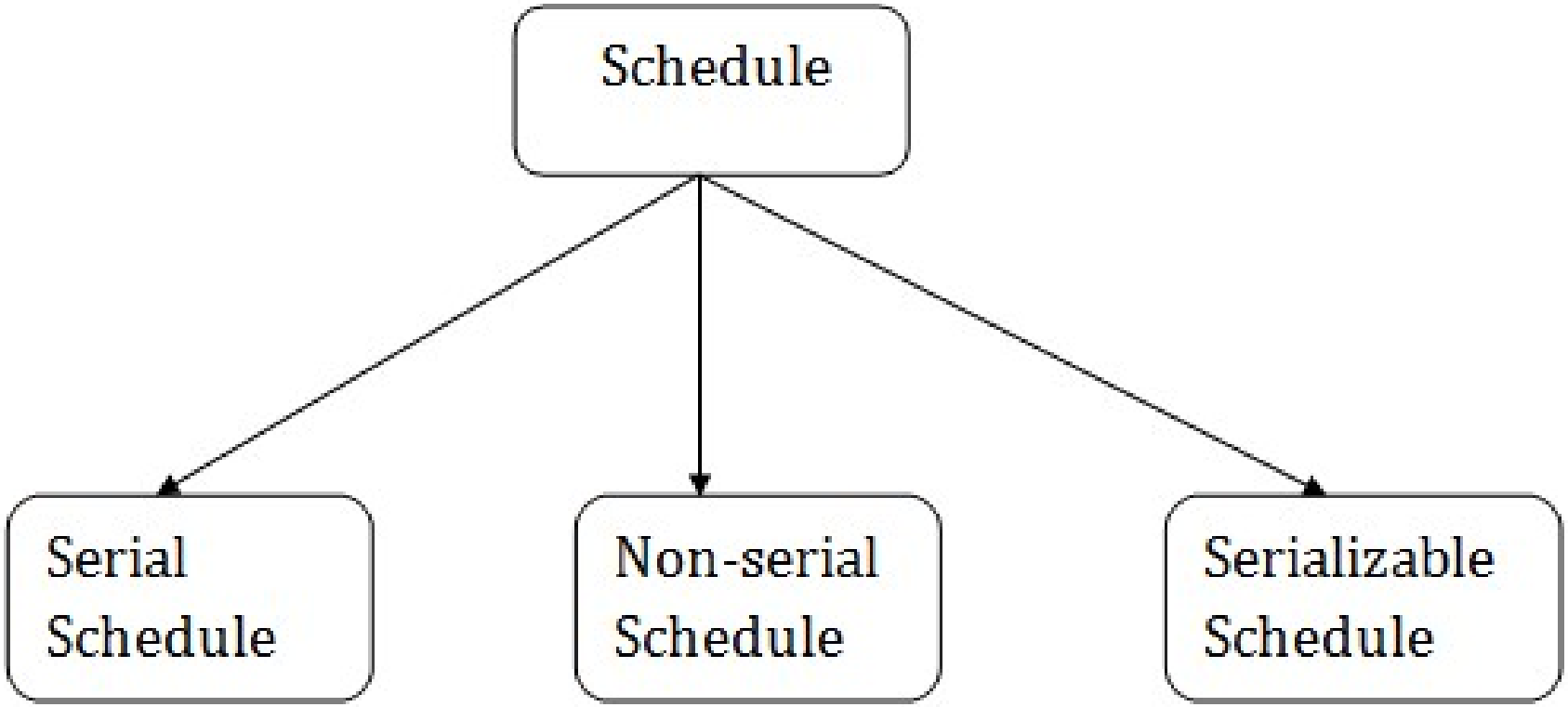
- It shows that the data which is used at the time of execution of a transaction cannot be used by the second transaction until the first one is completed.
- In isolation, if the transaction T1 is being executed and using the data item X, then that data item can't be accessed by any other transaction T2 until the transaction T1 ends.
- The concurrency control subsystem of the DBMS enforced the isolation property.

Durability

- The durability property is used to indicate the performance of the database's consistent state. It states that the transaction made the permanent changes.
- They cannot be lost by the erroneous operation of a faulty transaction or by the system failure. When a transaction is completed, then the database reaches a state known as the consistent state. That consistent state cannot be lost, even in the event of a system's failure.
- The recovery subsystem of the DBMS has the responsibility of Durability property.

Schedule

- A series of operation 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.


Schedule
Serial Schedule
Non-serial Schedule
Serializable Schedule

1. Serial Schedule

- The serial schedule is a type of schedule where one transaction is executed completely before starting another transaction.

- In the serial schedule, when the first transaction completes its cycle, then the next transaction is executed.

- **For example:** Suppose there are two transactions T1 and T2 which have some operations. If it has no interleaving of operations, then there are the following two possible outcomes:
- Execute all the operations of T1 which was followed by all the operations of T2.
- Execute all the operations of T1 which was followed by all the operations of T2.
- In the given (a) figure, Schedule A shows the serial schedule where T1 followed by T2.
- In the given (b) figure, Schedule B shows the serial schedule where T2 followed by T1.

(a)

Time

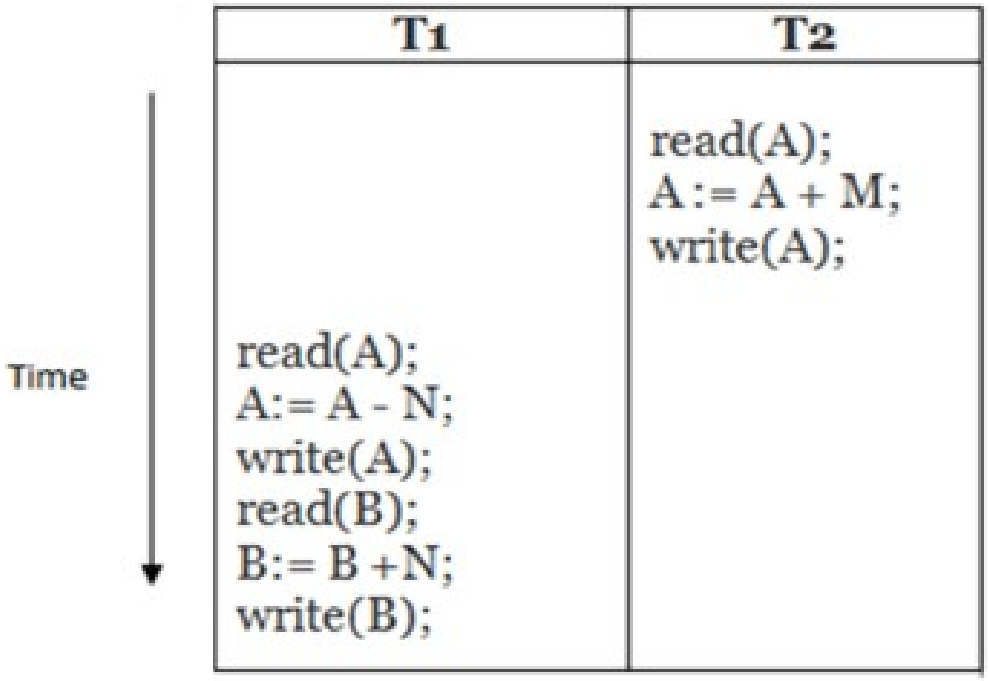
T1	T2

read(A);
A := A – N;
write(A);
read(B);
B := B + N;
write(B);	

	read(A);
	A := A + M;
	write(A);

Schedule A

(b)

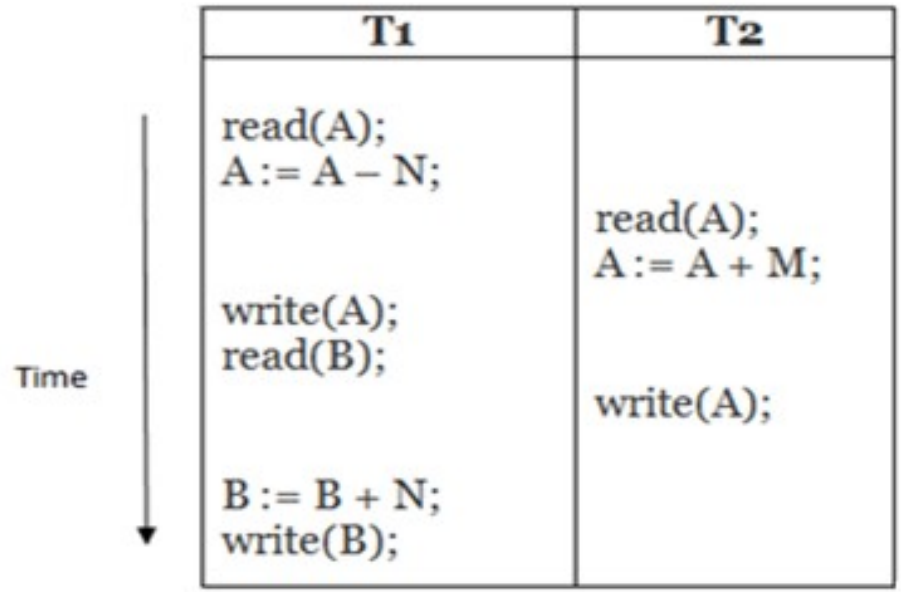

T1
T2
read(A);
A := A + M;
write(A);
Time
read(A);
A:= A - N;
write(A);
read(B);
B:= B +N;
write(B);

Schedule B

2. Non-serial Schedule

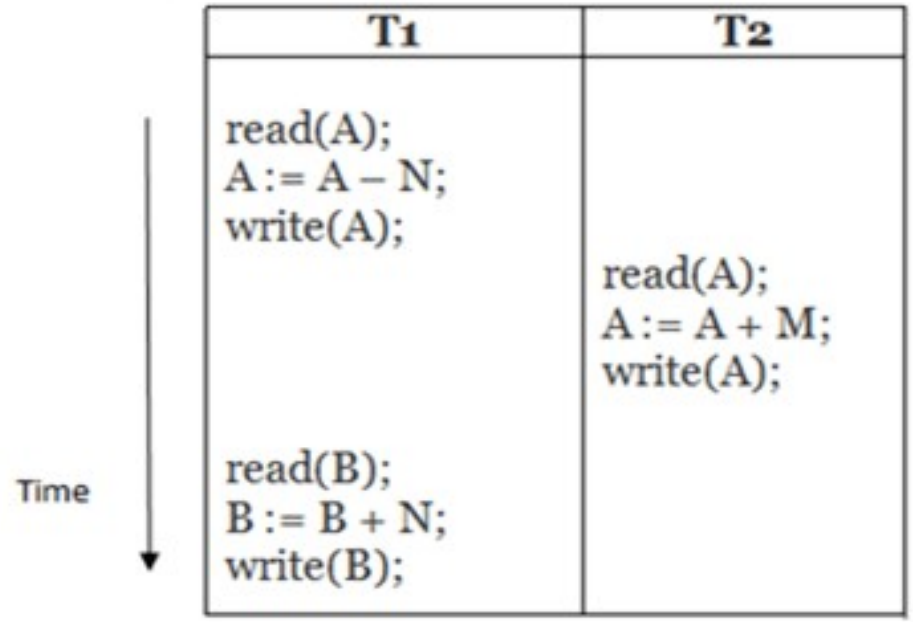
- If interleaving of operations is allowed, then there will be non-serial schedule.
- It contains many possible orders in which the system can execute the individual operations of the transactions.
- In the given figure (c) and (d), Schedule C and Schedule D are the non-serial schedules. It has interleaving of operations.

(c)


T1
T2
read(A);
A := A – N;
write(A);
read(B);
B := B + N;
write(B);
read(A);
A := A + M;
write(A);
Time

Schedule C

(d)


T1
T2
read(A);
A := A – N;
write(A);
read(A);
A := A + M;
write(A);
read(B);
B := B + N;
write(B);
Time

Schedule D

3. Serializable schedule

- The serializability of schedules is used to find non-serial schedules that allow the transaction to execute concurrently without interfering with one another.
- It identifies which schedules are correct when executions of the transaction have interleaving of their operations.
- A non-serial schedule will be serializable if its result is equal to the result of its transactions executed serially.

Serializability

- 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

What is a serializable schedule?

- A serializable schedule always leaves the database in consistent state. A **serial schedule** is always a serializable schedule because in serial schedule, a transaction only starts when the other transaction finished execution. However a non-serial schedule needs to be checked for Serializability.

- A non-serial schedule of n number of transactions is said to be serializable schedule, if it is equivalent to the serial schedule of those n transactions.
- A serial schedule doesn’t allow concurrency, only one transaction executes at a time and the other starts when the already running transaction finished.

Types of Serializability

- There are two types of Serializability.

1. **Conflict Serializability**
2. **View Serializability**

Conflict Serializability

- **Conflict Serializability** is one of the type of Serializability, which can be used to check whether a non-serial schedule is conflict serializable or not.
- 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:

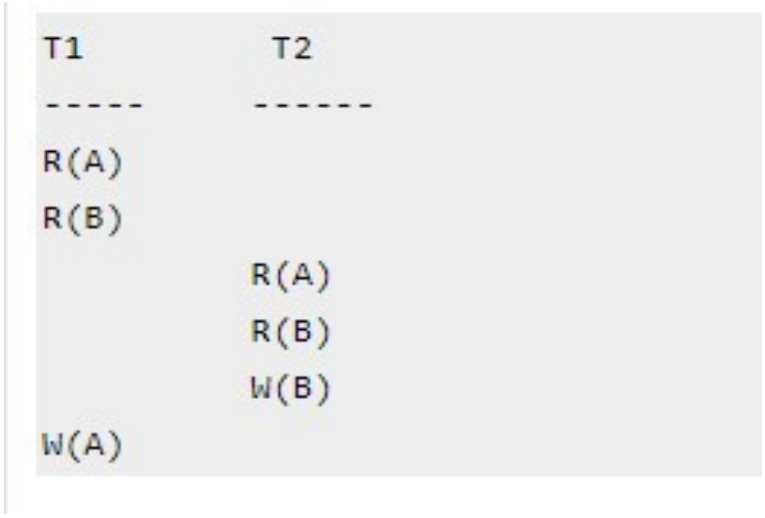
1. Both the operations should belong to different transactions.
2. Both the operations are working on same data item.
3. At least one of the operation is a write operation.

Conflict Equivalent Schedules

- Two schedules are said to be conflict Equivalent if one schedule can be converted into other schedule after swapping non-conflicting operations.

Example of Conflict Serializability

Lets consider this schedule:



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

- To convert this schedule into a serial schedule we must have to swap the R(A) operation of transaction T2 with the W(A) operation of transaction T1.
- However we cannot swap these two operations because they are conflicting operations, thus we can say that this given schedule is **not Conflict Serializable**.

Lets **swap non-conflicting operations**:

After swapping R(A) of T1 and R(A) of T2 we get:

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

After swapping R(A) of T1 and R(B) of T2 we get:

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

After swapping R(A) of T1 and W(B) of T2 we get:

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

We finally got a serial schedule after swapping all the non-conflicting operations so we can say that the given schedule is **Conflict Serializable**.

View Serializability

What is View Serializability?

- View Serializability is a process to find out that a given **schedule** is view serializable or not.
- To check whether a given schedule is view serializable, we need to check whether the given schedule is **View Equivalent** to its serial schedule. Lets take an example to understand what I mean by that.

Given Schedule:

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

Serial Schedule of the above given schedule:

- As we know that in **Serial schedule** a transaction only starts when the current running transaction is finished. So the serial schedule of the above given schedule would look like this:

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

If we can prove that the given schedule is **View Equivalent** to its serial schedule then the given schedule is called **view Serializable**.

Why we need View Serializability?

- We know that a serial schedule never leaves the database in inconsistent state because there are no concurrent transactions execution.
- However a non-serial schedule can leave the database in inconsistent state because there are multiple transactions running concurrently.
- By checking that a given non-serial schedule is view serializable, we make sure that it is a consistent schedule.

- You may be wondering instead of checking that a non-serial schedule is serializable or not, can’t we have serial schedule all the time?
- The answer is no, because concurrent execution of transactions fully utilize the system resources and are considerably faster compared to serial schedules.

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.

Read vs Initial Read:

- You may be confused by the term initial read. Here initial read means the first read operation on a data item, for example, a data item X can be read multiple times in a schedule but the first read operation on X is called the initial read. This will be more clear once we will get to the example in the next section of this same article.

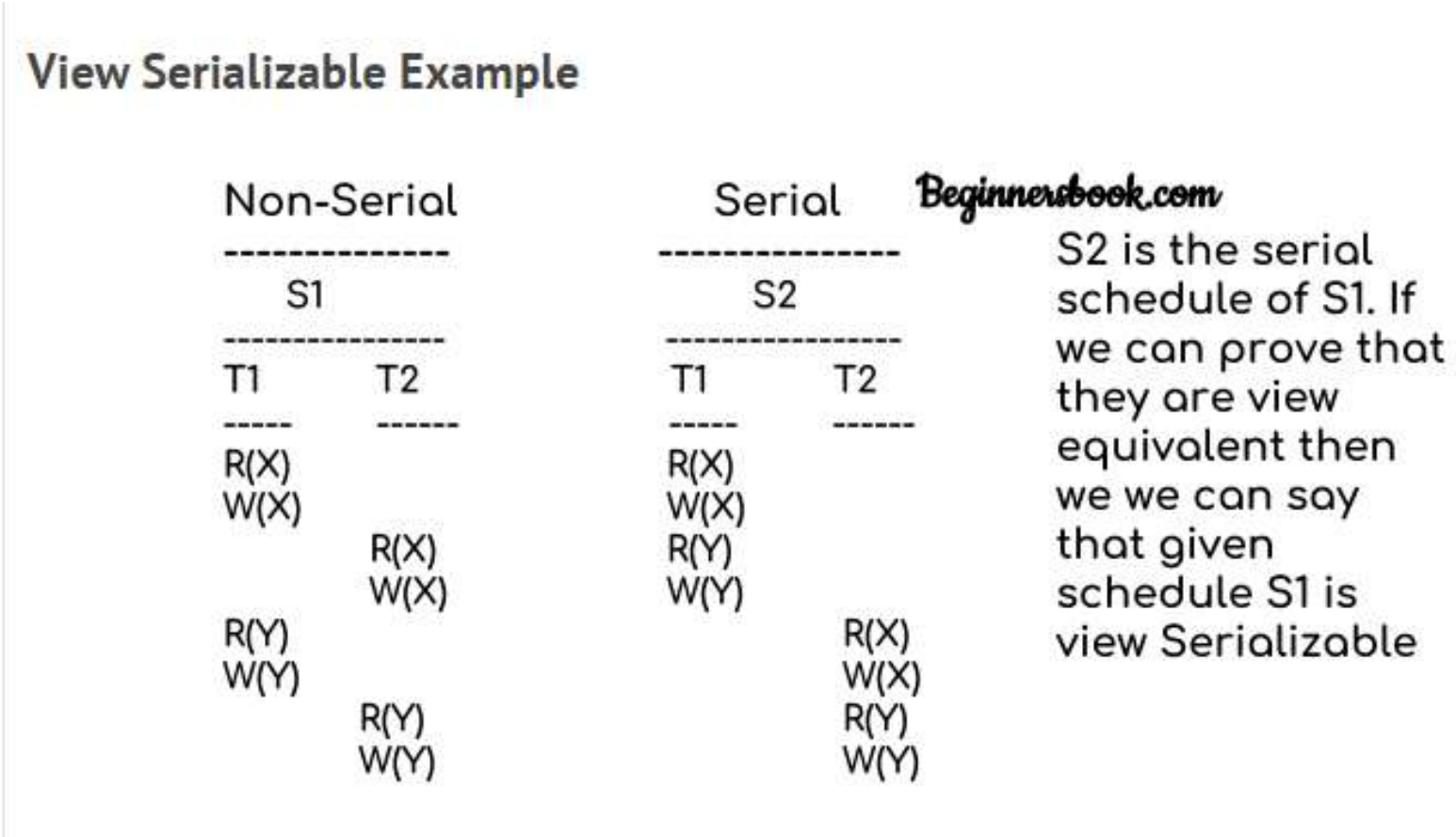
2. **Final Write:** Final write operations on each data item must match in both the schedules. For example, a data item X is last written by Transaction T1 in schedule S1 then in S2, the last write operation on X should be performed by the transaction T1.

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 Serializable

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



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.

Concurrency Control

- Concurrency Control is the management procedure that is required for controlling concurrent execution of the operations that take place on a database.
- But before knowing about concurrency control, we should know about concurrent execution.

Concurrent Execution in DBMS

- In a multi-user system, multiple users can access and use the same database at one time, which is known as the concurrent execution of the database.
- It means that the same database is executed simultaneously on a multi-user system by different users.

- While working on the database transactions, there occurs the requirement of using the database by multiple users for performing different operations, and in that case, concurrent execution of the database is performed.

- The thing is that the simultaneous execution that is performed should be done in an interleaved manner, and no operation should affect the other executing operations, thus maintaining the consistency of the database.
- Thus, on making the concurrent execution of the transaction operations, there occur several challenging problems that need to be solved.

Problems with Concurrent Execution

- In a database transaction, the two main operations are **READ** and **WRITE** operations.
- So, there is a need to manage these two operations in the concurrent execution of the transactions as if these operations are not performed in an interleaved manner, and the data may become inconsistent.
- So, the following problems occur with the Concurrent Execution of the operations:

Problem 1: Lost Update Problems

- The problem occurs *when two different database transactions perform the read/write operations on the same database items in an interleaved manner (i.e., concurrent execution) that makes the values of the items incorrect hence making the database inconsistent*.

For example:

Consider the below diagram where two transactions T_X and T_Y, are performed on the same account A where the balance of account A is $300.

Time	T_x	T_y
t_1	READ (A)	—
t_2	A = A - 50	
t_3	—	READ (A)
t_4	—	A = A + 100
t_5	—	—
t_6	WRITE (A)	—
t_7		WRITE (A)

LOST UPDATE PROBLEM

- At time t1, transaction T_X reads the value of account A, i.e., $300 (only read).
- At time t2, transaction T_X deducts $50 from account A that becomes $250 (only deducted and not updated/write).
- Alternately, at time t3, transaction T_Y reads the value of account A that will be $300 only because T_X didn't update the value yet.
- At time t4, transaction T_Y adds $100 to account A that becomes $400 (only added but not updated/write).

- At time t6, transaction T_X writes the value of account A that will be updated as $250 only, as T_Y didn't update the value yet.
- Similarly, at time t7, transaction T_Y writes the values of account A, so it will write as done at time t4 that will be $400. It means the value written by T_X is lost, i.e., $250 is lost.
- Hence data becomes incorrect, and database sets to inconsistent.

Dirty Read Problems

- The dirty read problem occurs *when one transaction updates an item of the database, and somehow the transaction fails, and before the data gets rollback, the updated database item is accessed by another transaction. There comes the Read-Write Conflict between both transactions.*

For example:

Consider two transactions T_X and T_Y in the below diagram performing read/write operations on account A where the available balance in account A is $300:

Time	T_x	T_y
t_1	READ (A)	—
t_2	A = A + 50	—
t_3	WRITE (A)	—
t_4	—	READ (A)
t_5	SERVER DOWN ROLLBACK	—

DIRTY READ PROBLEM

- At time t1, transaction T_X reads the value of account A, i.e., $300.
- At time t2, transaction T_X adds $50 to account A that becomes $350.
- At time t3, transaction T_X writes the updated value in account A, i.e., $350.
- Then at time t4, transaction T_Y reads account A that will be read as $350.

- Then at time t5, transaction T_X rollbacks due to server problem, and the value changes back to $300 (as initially).
- But the value for account A remains $350 for transaction T_Y as committed, which is the dirty read and therefore known as the Dirty Read Problem.

Unrepeatable Read Problem

- *Also known as Inconsistent Retrievals Problem that occurs when in a transaction, two different values are read for the same database item.*

For example:

Consider two transactions, T_X and T_Y, performing the read/write operations on account A, having an available balance = $300. The diagram is shown below:

Time	T_x	T_y
t_1	READ (A)	—
t_2	—	READ (A)
t_3	—	A = A + 100
t_4	—	WRITE (A)
t_5	READ (A)	—

UNREPEATABLE READ PROBLEM

- At time t1, transaction T_X reads the value from account A, i.e., $300.
- At time t2, transaction T_Y reads the value from account A, i.e., $300.
- At time t3, transaction T_Y updates the value of account A by adding $100 to the available balance, and then it becomes $400.
- At time t4, transaction T_Y writes the updated value, i.e., $400.

- After that, at time t5, transaction T_X reads the available value of account A, and that will be read as $400.
- It means that within the same transaction T_X, it reads two different values of account A, i.e., $ 300 initially, and after updation made by transaction T_Y, it reads $400.
- It is an unrepeatable read and is therefore known as the Unrepeatable read problem.

Concurrency Control

- Concurrency Control is the working concept that is required for controlling and managing the concurrent execution of database operations and thus avoiding the inconsistencies in the database. Thus, for maintaining the concurrency of the database, we have the concurrency control protocols.

Concurrency Control Protocols

- The concurrency control protocols ensure the *atomicity, consistency, isolation, durability* and *serializability* of the concurrent execution of the database transactions.

Therefore, these protocols are categorized as:

- Lock Based Concurrency Control Protocol
- Time Stamp Concurrency Control Protocol
- Validation Based Concurrency Control Protocol

Lock-Based Protocol

- In this type of protocol, any transaction cannot read or write data until it acquires an appropriate lock on it. There are two types of lock:

1. Shared lock:

- It is also known as a Read-only lock. In a shared lock, the data item can only read by the transaction.
- It can be shared between the transactions because when the transaction holds a lock, then it can't update the data on the data item.

2. Exclusive lock:

- In the exclusive lock, the data item can be both reads as well as written by the transaction.
- This lock is exclusive, and in this lock, multiple transactions do not modify the same data simultaneously

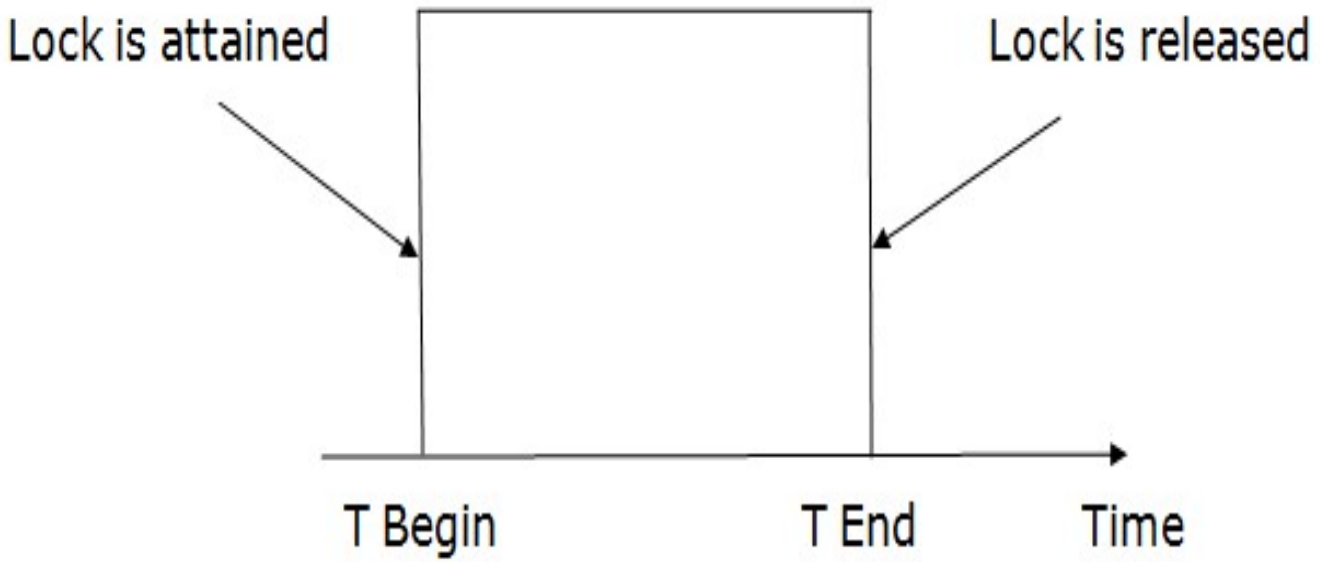
There are four types of lock protocols available:

1. **Simplistic lock protocol**

- It is the simplest way of locking the data while transaction. Simplistic lock-based protocols allow all the transactions to get the lock on the data before insert or delete or update on it. It will unlock the data item after completing the transaction.

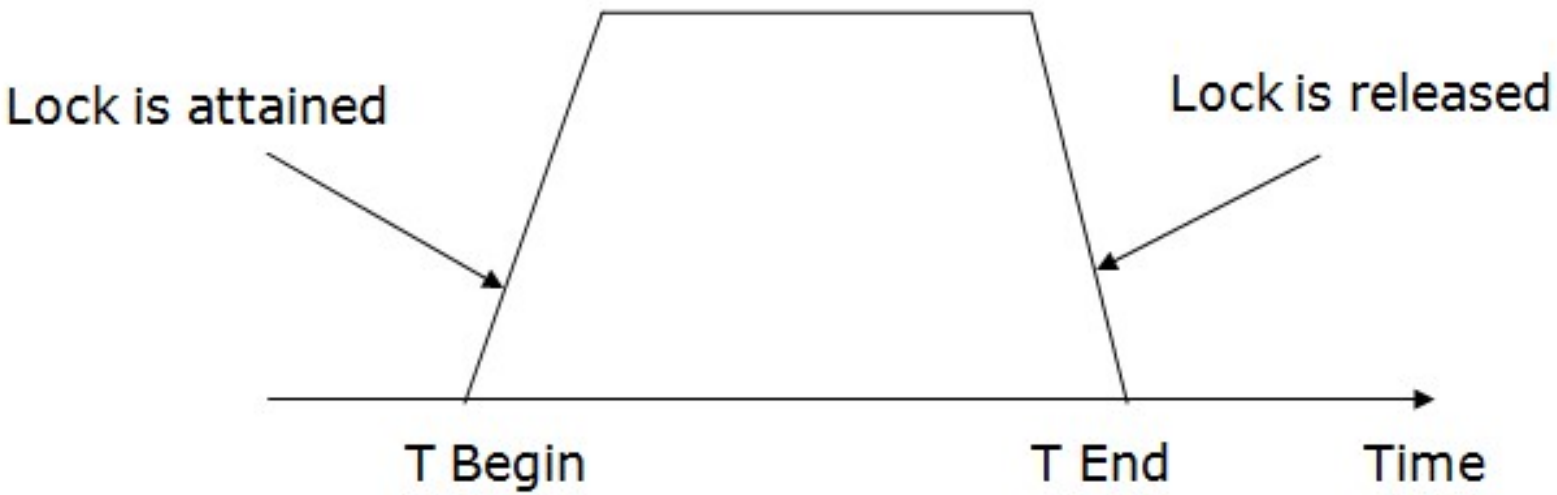
2. Pre-claiming Lock Protocol

- Pre-claiming Lock Protocols evaluate the transaction to list all the data items on which they need locks.
- Before initiating an execution of the transaction, it requests DBMS for all the lock on all those data items.
- If all the locks are granted then this protocol allows the transaction to begin. When the transaction is completed then it releases all the lock.
- If all the locks are not granted then this protocol allows the transaction to rolls back and waits until all the locks are granted.


Lock is attained
Lock is released
T Begin
T End
Time

3. Two-phase locking (2PL)

- The two-phase locking protocol divides the execution phase of the transaction into three parts.
- In the first part, when the execution of the transaction starts, it seeks permission for the lock it requires.
- In the second part, the transaction acquires all the locks. The third phase is started as soon as the transaction releases its first lock.
- In the third phase, the transaction cannot demand any new locks. It only releases the acquired locks.


Lock is attained
Lock is released
T Begin
T End
Time

There are two phases of 2PL:

- **Growing phase:** In the growing phase, a new lock on the data item may be acquired by the transaction, but none can be released.
- **Shrinking phase:** In the shrinking phase, existing lock held by the transaction may be released, but no new locks can be acquired.
- In the below example, if lock conversion is allowed then the following phase can happen:
- Upgrading of lock (from S(a) to X (a)) is allowed in growing phase.
- Downgrading of lock (from X(a) to S(a)) must be done in shrinking phase.

Example:

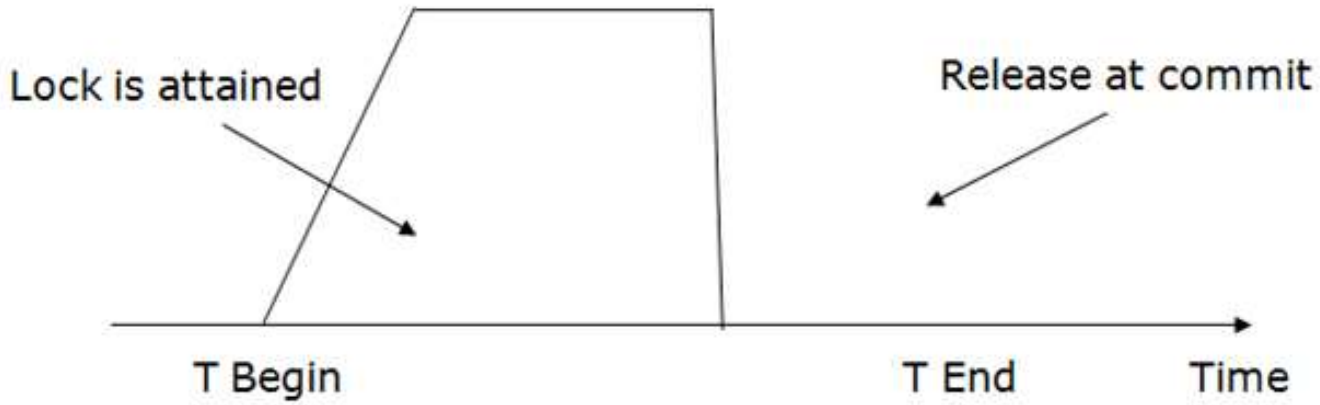
	T1	T2
0	LOCK-S(A)	
1		LOCK-S(A)
2	LOCK-X(B)	
3	——	——
4	UNLOCK(A)	
5		LOCK-X(C)
6	UNLOCK(B)	
7		UNLOCK(A)
8		UNLOCK(C)
9	——	——

The following way shows how unlocking and locking work with 2-PL.

- **Transaction T1:**
- Growing phase: from step 1-3
- Shrinking phase: from step 5-7
- Lock point 3: at
- **Transaction T2:**
- Growing phase: from step 2-6
- Shrinking phase: from step 8-9
- Lock point: at 6

4. Strict Two-phase locking (Strict-2PL)

- The first phase of Strict-2PL is similar to 2PL. In the first phase, after acquiring all the locks, the transaction continues to execute normally.
- The only difference between 2PL and strict 2PL is that Strict-2PL does not release a lock after using it.
- Strict-2PL waits until the whole transaction to commit, and then it releases all the locks at a time.
- Strict-2PL protocol does not have shrinking phase of lock release.



It does not have cascading abort as 2PL does.