What is transactions ?

3.1. Concept of transaction, ACID properties, States of transaction
3.2. Concurrency control, Problems in concurrency controls
3.3. Scheduling of transactions, Serializability and testing of serilaizibity

Definition :

Collections of operations that form a single logical unit of work are called transactions

For Ex :

A transfer of funds from one account to another account

Transaction Concept :

A transaction is a unit of program execution that accesses and possibly updates various data items

Programs written in a high level data manipulation language or programming language
(SQL ,C++ or Java or VB.Net)

Properties of Transactions :

To ensure integrity of the data, we require that the database system maintain the following properties

A C I D

A **Atomicity :** Either all operations of the transaction are reflected property in the database or none are

C **consistency :** Execution of a transaction in isolation means with no other transaction executing concurrently preserves the consistency of the database.

I **Isolation :** Even though multiple transactions any execute concurrently, the system guarantees that, t1 and t2, t1 start first then first finish then t2 start or t2 start first then first finish then t1 start

D **Durability :** After a transactions completes successfully, the changes it has to the database persist, even if there are system failures.

Transactions two main operations :

1. **read(x)**
2. **write(x)**

read(x) : which Transfer the data item x from the database to local buffer.

Write(x) :

which Transfer the data item x from local buffer the to database

For Ex: T_i be a transaction that transfer $50 from account A to account b :

T_i :

```
read(A);
A:=A-50;
write(A);
read(B);
B:=B+50;
write(B);
```

Consistency : The consistency requirement here is that the sum of A and B unchanged

Atomicity :

complete transactions :

Before:

Account A : $1000

Account B : $2000

sum of account balance before and after $3000

After :

Account A : $950

Account B : $2050

sum of account balance before and after $3000

consistency in data

Failure Problem :

After deduction $50 from account A power failure problem then

Account A : $950

Account B : $2000

sum of account balance

before ----$3000

after-------$2950

Inconsistency data

Atomicity : All transactions are reflected in the database or none.

Durability :

1. The updates carried out by transactions have been written to disk before the transaction completes

2. Information about the updates carried out by the transaction and written to disk is sufficient to enable the database to reconstruct the update when the database system is restarted after the failure.

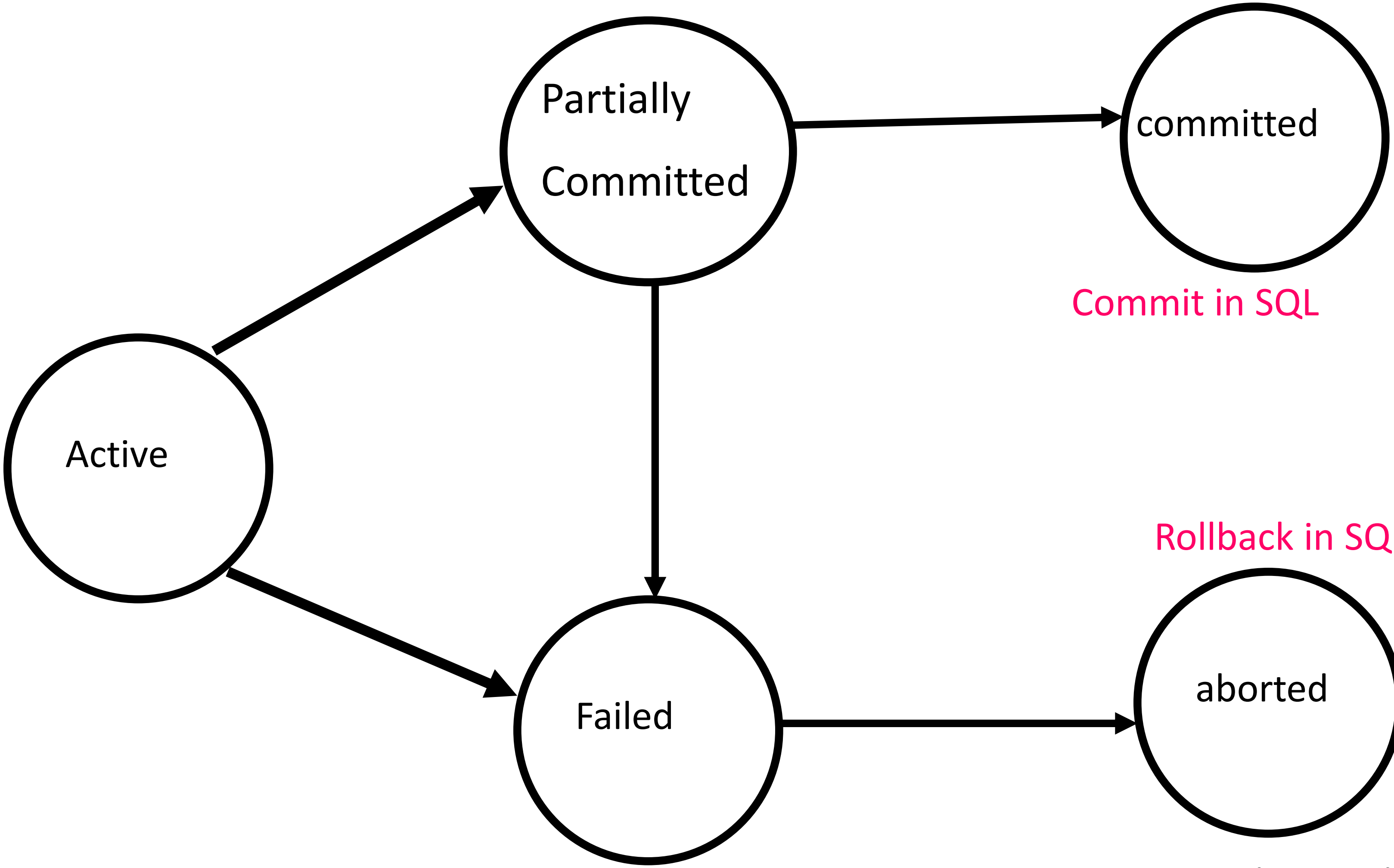
(Recovery management component.)

Isolation : Concurrently allows to executes more than one transactions

Transaction State :

Atomicity – complete transactions

Updates new effects on database



None in atomicity

No effects on database

Active : The initial state : it is executing state

Partially committed : This state comes after the final statement has been executed.

Aborted : After the transaction has been rolled back and the database has been restored to its state prior to the start of the transactions

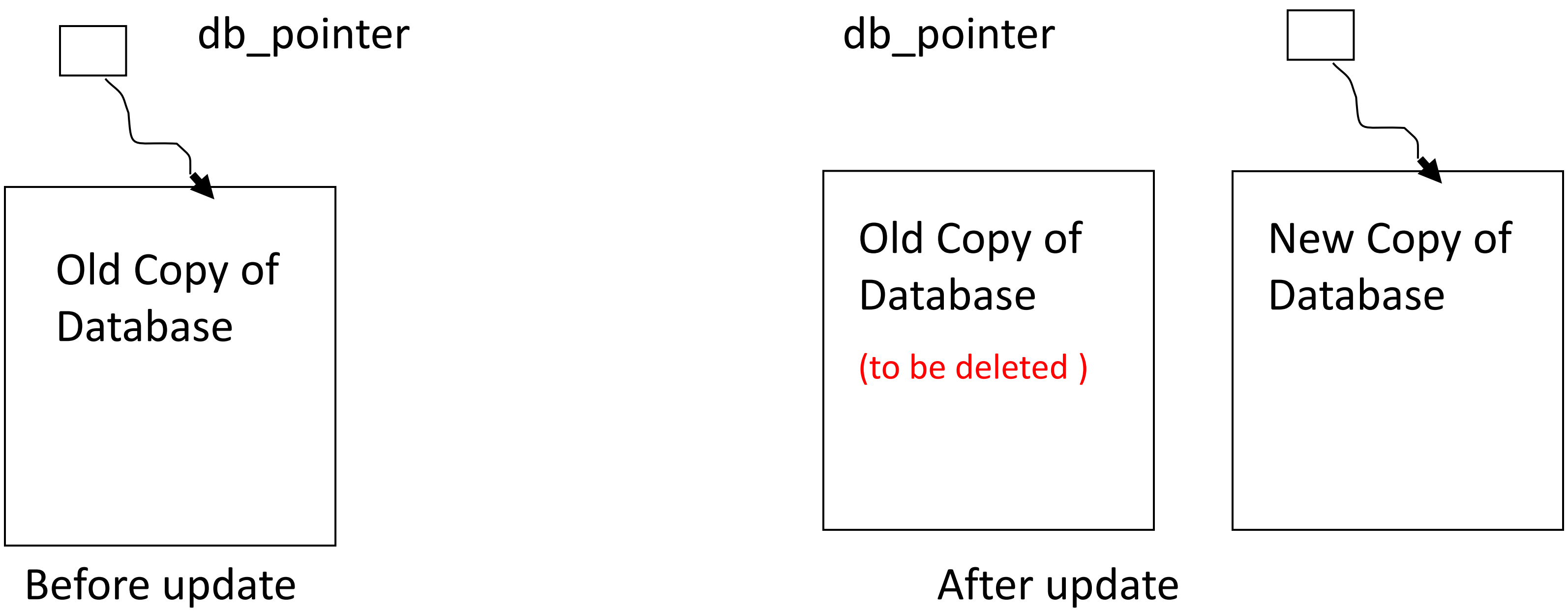
1] Restart : restart transaction only after aborted transaction .
(consider new transaction)

2] Kill : Kill the transactions
1 . internal logical error, new program required to write .
2. bad input or data not found

Committed : This state comes after the successfully completion.

Implementations of Atomicity & Durability :

(A C I D)



Shadow – Copy technique for atomicity & durability

<u>Concurrent Executions : (</u>ACID)

1. Transactions Serially executes
2. Transactions Concurrent executes

<u>Advantages of Concurrent executions :</u>

a) Improve throughput and resource utilization

I/O activity

CPU activity

b) Reducing waiting time

Concurrency – control schemas :

T1 transactions transfer $50 from account A to account B

```
T1 :        read(A);
            A:=A-50;
            write(A);
            Read(B);
            B:=B+50;
            write(B);
```

T2 transactions transfer 10% from account A to account B

```
T2 :    read(A);
        temp:=A*0.1;
        A:=A-temp;
        write(A);
        Read(B);
        B:=B+temp;
        write(B);
```

Suppose A balance : 1000$

B balance : 2000$

These two transactions executes one after by one

(T1 followed by T2)

T1	T2
read(A);	
A:=A-50;	
write(A);	
read(B);	
B:=B+50;	
write(B);	
	read(A);
	temp:=A*0.1;
	A:=A-temp;
	write(A);
	read(B);
	B:=B+temp;
	write(B);

After transactions complete

A balance : 855$

B balance : 2145$

Schedule 1

Serial Schedule

Suppose A balance : 1000$

B balance : 2000$

These two transactions executes one after by one (T2 followed by T1)

T1	T2

	read(A);
	temp:=A*0.1;
	A:=A-temp;
	write(A);
	read(B);
	B:=B + temp;
	write(B);

read(A);
A:=A-50;
write(A);
read(B);
B:=B+50;
write(B);	

After transactions complete

A balance : 850$

B balance : 2150$

Schedule 2

Serial Schedule

Suppose A balance : 1000$

B balance : 2000$

These two transactions executes concurrently

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);	

	read(B);
	B:=B + temp;
	write(B);

After transactions complete

A balance : 855 $

B balance : 2145$

Schedule 3

Concurrent Schedule

Sum after execution is indeed preserved

Suppose A balance : 1000$

B balance : 2000$

These two transactions executes concurrently

T1	T2
read(A); A:=A-50;	

	read(A);
	temp:=A*0.1;
	A:=A-temp;
	write(A);
	read(B);

write(A);
read(B);
B:=B+50;
write(B);	

	B:=B + temp; write(B);

After transactions complete

A balance : 950 $

B balance : 2100$

Schedule 4

Concurrent Schedule

Sum after execution is not preserved

It is the job of the database system to ensure that any schedule that gets executed will leave the database in a consistent state done with the help of

Concurrency control component of dbms

Serializability :

DBMS done this task (concurrency control) by using serializability.

Related to dbms only read(Q) and write(Q) only two operations important. For ex.

These two transactions executes concurrently

T1	T2
read(A); write(A);	
	read(A); write(A);
read(B); write(B);	
	read(B); write(B);

Schedule 3

Concurrent Schedule (only read and write operations consider)

Serializability :

Two types of serializability

1] Conflict Serializability :
2] View Serializability :

Conflict Serializability :

Let us consider a schedule S in which there are two consecutive instructions I_i and I_j, of transactions T_i and T_j respectively (I not equal to J). If Ii and Ij, refer different data items, then we can swap Ii and Ij, without affecting the results of any instructions in the schedule.

Let us consider a schedule S in which there are two consecutive instructions Ii and Ij, of transactions Ti and Tj respectively (I not equal to J). If Ii and Ij, refer same data items, then the order of two steps may matter as follows .

1] I_i =read(Q) and I_j =read(Q)

Order does not matter because both read same value of Q.

Here order does not matter.

2] I_i =read(Q) and I_j =write(Q)

Order does matter because I_i read before updation value of Q. If order change the I_i read after updated values of Q by I_j .

Here order does matter.

3] I $_i$=write(Q) and I_j =read(Q)

Order does matter like above condition.

Here order does matter.

4] I i=write(Q) and Ij =write(Q)

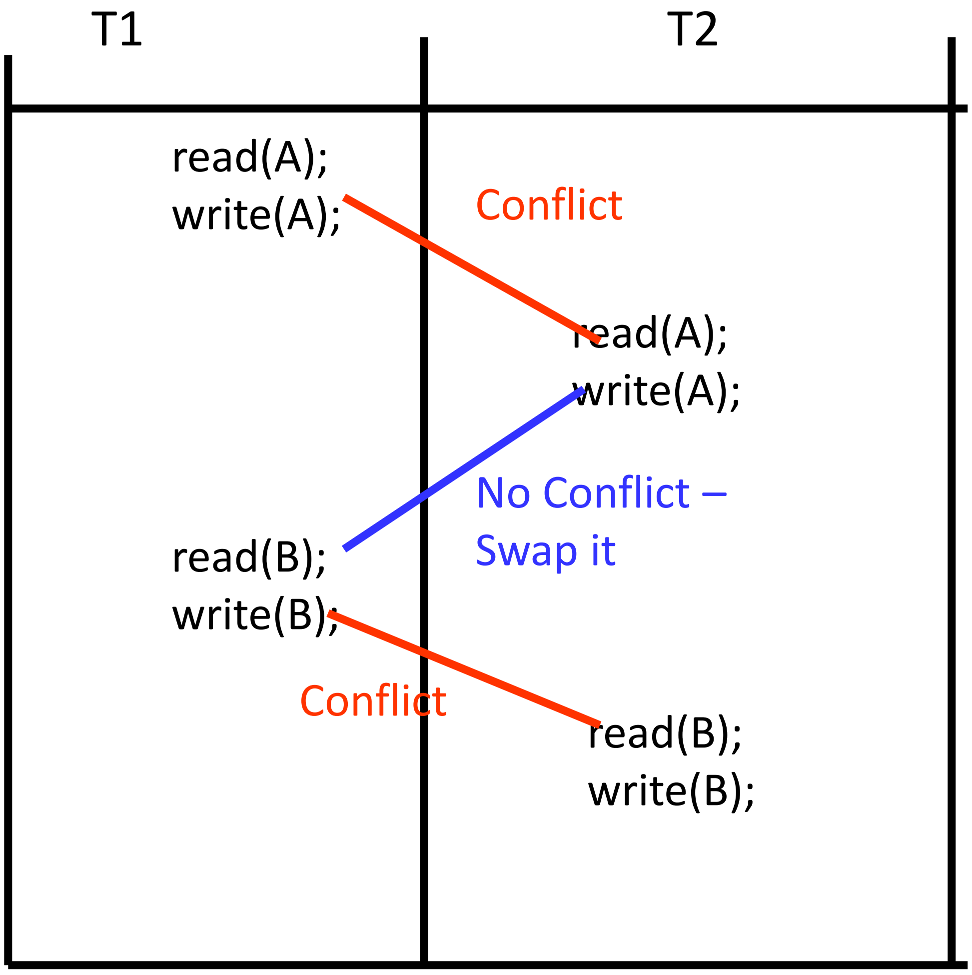
Order does not matter because both are write oprations and not affect on transactions.

Here order does not matter.

Conflict :

We say that I_i and I_j in conflict if they are operations by different transactions on the same data item, and at least one of these instructions is a write oprations.

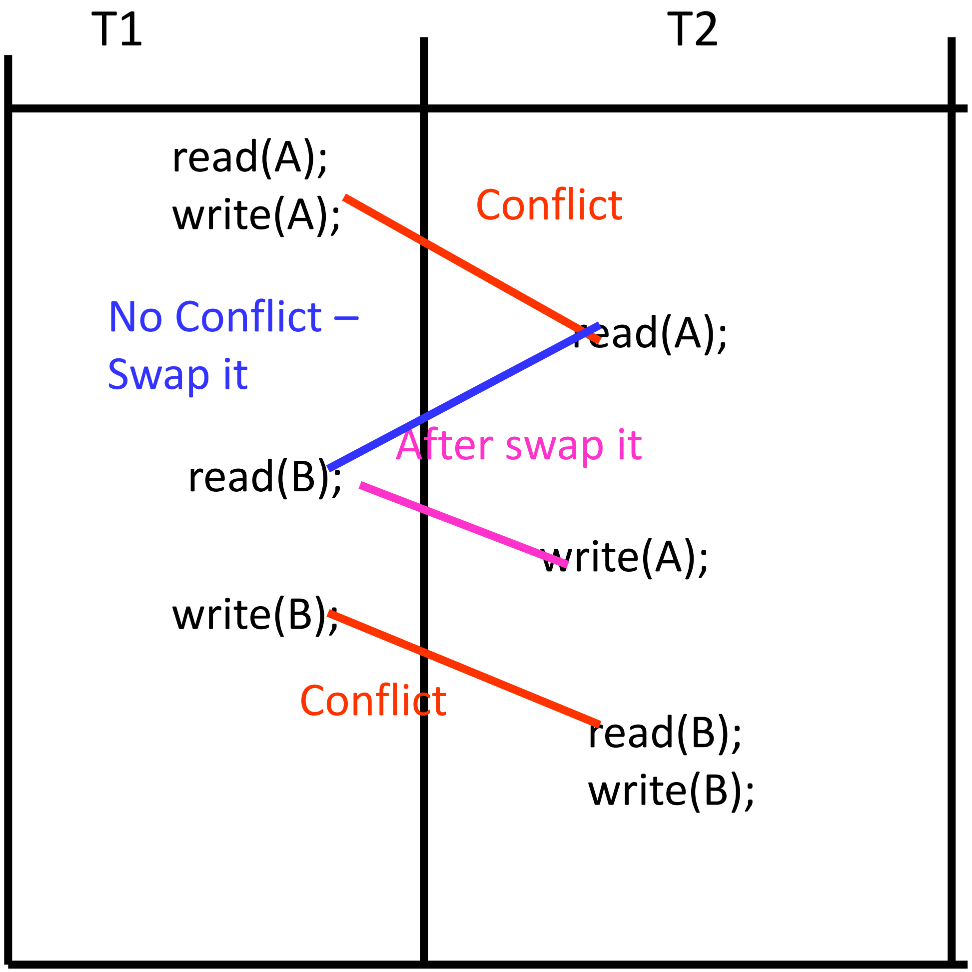
These two transactions executes concurrently



Schedule 3

Concurrent Schedule (only read and write operations consider)

These two transactions executes concurrently



Schedule 3

Concurrent Schedule (only read and write operations consider)

These two transactions executes concurrently

T1	T2
read(A);	
write(A);	
read(B);	
	read(A);
	write(A);
write(B);	
	read(B);
	write(B);

After swap it

No Conflict – Swap it

Conflict

Schedule 3

Concurrent Schedule (only read and write operations consider)

These two transactions executes concurrently

T1	T2
read(A);	
write(A);	
read(B);	
	read(A);
write(B);	
	write(A);
	read(B);
	write(B);

No Conflict – Swap it

After swap it

Schedule 3

Concurrent Schedule (only read and write operations consider)

These two transactions executes concurrently

T1	T2
read(A);	
write(A);	
read(B);	
write(B);	After swap it
	read(A);
	write(A);
	read(B);
	write(B);

Schedule 6

Concurrent Schedule (But after swap it is like serial schedule)

Schedule 6 – serial schedule that is equivalent to schedule 3 – Concurrent schedule

Conflict equivalent : If a schedule S can be transformed into a schedule S’ by a series of swaps of non – conflicting instructions, then we say that S and S’ are Conflict equivalent

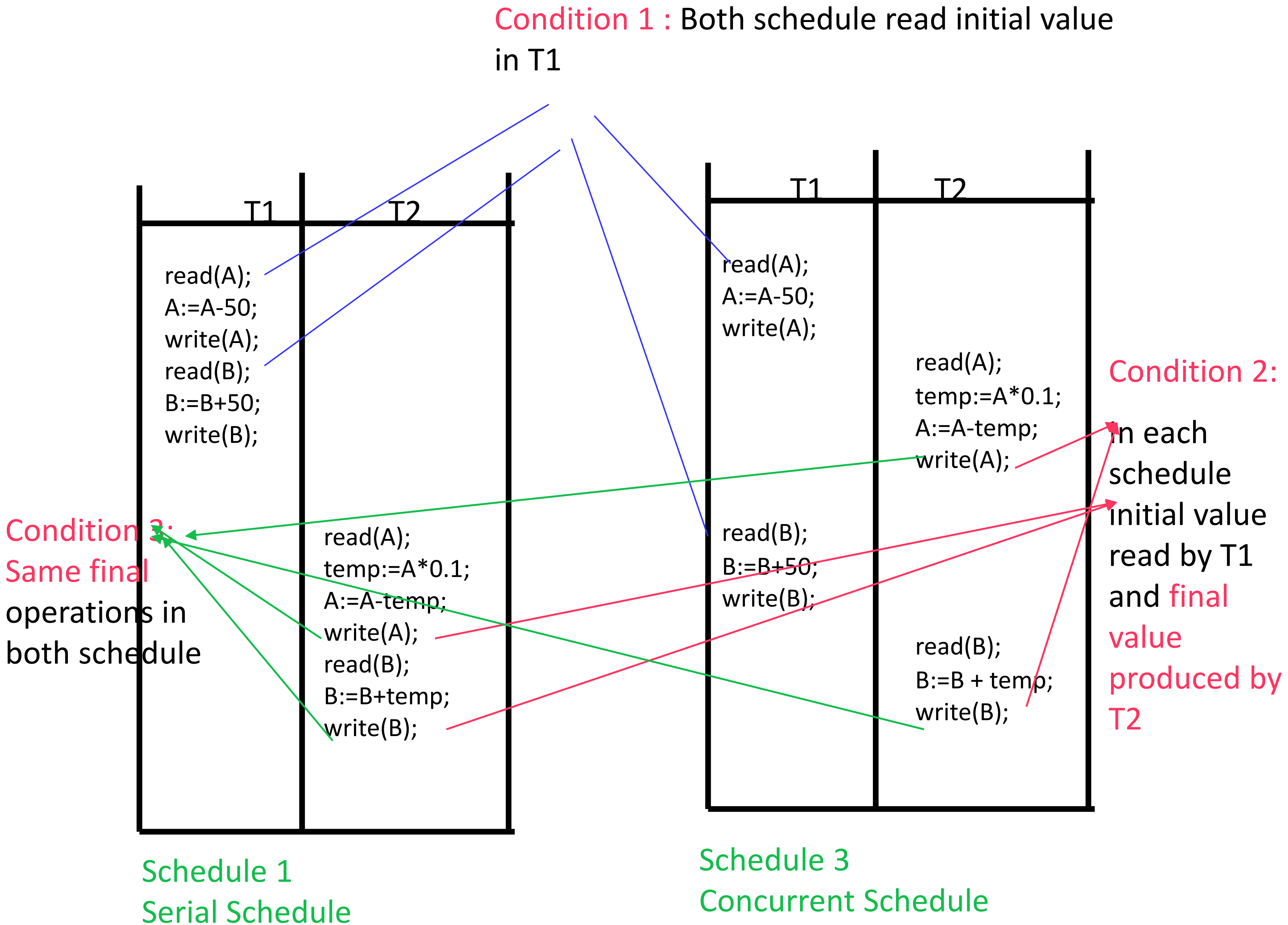
For ex. Schdeule 3 – concurrent schedule is equivalent to schedule 6 which is serial schedule then this type of conflict is called as conflict serializabilty.

View Serializablity : The schedule S and S’ are said to view equivalent if three conditions are met.

Condition 1 : For each data item Q, if transaction Ti read the initial value of Q in schedule S , then transaction Tj must in schedule S’ also read the initial value of Q.

Condition 2 : For each data item Q, if transaction Ti executes read(Q) is schedule S and if that values was produced by write(Q) operations executed by transactions Tj, then the read(Q) operation of transactions Ti must in schedule S1, also read the value of Q that was produced by the same write(Q) operations of transaction Tj.

Condition 3: For each data item Q, if transaction (if any) that performs the final write(Q) operations in schedule S,must perofrm the final write(Q) operation in schedule S'.


Condition 1 : Both schedule read initial value in T1
T1
T2
read(A);
A:=A-50;
write(A);
read(B);
B:=B+50;
write(B);
read(A);
temp:=A*0.1;
A:=A-temp;
write(A);
read(B);
B:=B+temp;
write(B);
Condition 3:
Same final operations in both schedule
Schedule 1
Serial Schedule
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);
read(B);
B:=B + temp;
write(B);
Condition 2:
In each schedule initial value read by T1 and final value produced by T2
Schedule 3
Concurrent Schedule

Schedule 1 is view equivalent to schedule 3 because the values of account A and B read by transactions T2 were produced by T1 in both schedule 1 & 3. Schedule 1 is serial schedule and schedule 3 is concurrent schedule then it is called as view serializable equivalent.

T1	T2
	read(A);
	temp:=A*0.1;
	A:=A-temp;
	write(A);
	read(B);
	B:=B + temp;
	write(B);
read(A);	
A:=A-50;	
write(A);	
read(B);	
B:=B+50;	
write(B);	

Schedule 2

Serial Schedule

T1	T2
read(A);	
A:=A-50;	
write(A);	
read(B);	
B:=B+50;	
write(B);	
	read(A);
	temp:=A*0.1;
	A:=A-temp;
	write(A);
	read(B);
	B:=B+temp;
	write(B);

Schedule 1

Serial Schedule

These two schedules are not......view equivalent .

Recoverability :

If a transactions T1 fails , for whatever reason, we need to undo the effect of this transaction to ensure the atomicity property of the transaction.

In a system that allows concurrent execution, it is necessary also to ensure that any transaction T2 that is dependent on T1 is also aborted.

Two way we concentrate on the schedules how to recover after transaction failure.

Non recoverable Schedule : This type schedule not allow in DBMS

T8	T9
read(A); write(A);	
	read(A);
read(B);	

commit

Fail before it commits.

Schedule 10 (non recoverable schedule) Here T9 already committed and cannot be aborted. That’s why it is impossible to recover correctly from the failure of T8.

DBMS require always recoverable schedule.

A recoverable schedule is one where for each pair of transactions Ti and Tj such that Tj reads a data item previously written by Ti, the commit operation of Ti appears before the Tj .

Cascadeless Schedules :

T10	T11	T12
read(A);		
read(B);		
write(A);		
	read(A);	
	write(A);	
		read(A);

Updated value of A by T10

Updated value of A by T11

Schedule 10 (recoverable but cascade schedule) A single transaction failure leads to a series of transaction rollbacks is called cascading rollback.

Cascading Schedules requires Cascading rollback which is undesirable, requires lots amount of work to recover that why , It is desirable to restrict the schedules to those where cascadeing rollback cannot occurs. Such schedules are called cascadeless schedules.

A Cascadeless Schedules is one where for each pair of transactions Ti and Tj such that Tj read data item previously written by Ti , the commit operation of Ti appears before the read operation of Tj. It is easy to verify that every cascadeless schedule is also recoverable.