COMPUTER SCIENCE

Database Management System

Transaction & Concurrency Control

Lecture_1

Vijay Agarwal sir

TOPICS TO BE COVERED

01 Transaction Concept

02 Serializable Schedule

CHAPTER 1 : FD & Normalization

1. Data base Term
   RDBM Concept
2. FD Concept
   FD types
   - Trivial
   - Non Trivial
   - Semi Non Trivial FD

   Properties of FD
3. Attribute closure

Key Concept

4. Super key
5. Candidate key
6. finding Multiple CK
7. Membership set
8. Equality b/w 2 FD set

(9) Finding # of Super keys & Candidate keys

(10) Minimal Cover

(11) Properties of Decomposition

- Lossless Join
  - Basic Concept
  - Binary Method
  - CHASE TEST
- Dependency Preserving

(12) Closure of FD set

(15) Normal Form

1NF
2NF
3NF
BCNF

→ (1) Concept, (2) Violation case (3) Defination

2NF
3NF
BCNF

Decomposition

Transaction

- Read(A)
- A = A - 1000;
- Write(A)
- Read(B)
- B = B + 1000;
- Write(B)

$A \xrightarrow{1000} B$

Read (Q) : Accessing Data Item (Q)

Write (Q) : Updating the Data Item (Q)

Commit : Indicate Successful Completion of Transaction

OR

Transaction executed Successfully

Transaction Concept

- A transaction is a unit of program execution that accesses and possibly updates various data items.
- E.g. Transaction to transfer Rs 500 from account A to account B:

1. read (A)
2. A : = A – 500
3. write(A)
4. read(B)
5. B: = B + 500
6. write(B)

A

500

B

A C I D

1. Atomicity
2. Consistency
3. Isolation
4. Durablity.

ACID Properties

- A transaction is a unit of program execution that accesses and possibly updates various data items. To preserve the integrity of data the database system must ensure:

  A. Atomicity

  C. Consistency

  I. Isolation

  D. Durability

ACID

Maintain Integrity

A C I D

1. Atomicity → [FULL or None]
2. Consistency
3. Isolation
4. Durablity.

① **Atomicity** : Either Execute all operation of the transaction successfully [Full] or None of them.

[Full or None]

Reason of transaction failure.

- Power failure
- S/w Crash
- H/w Crash
- System Crash
- N/W Error & etc

- Due to Any of these Reason if transaction is Failed before commit than Recovery Management component are there.

- When a transaction is Failed Recovery Management Component ROLLBACK [UNDO ALL MODIFICATION].

- Log's [Transaction Log] : Log contain all the activity (Modification) of the transaction.

② Consistency : Before & After the transaction Database Must be consistent.

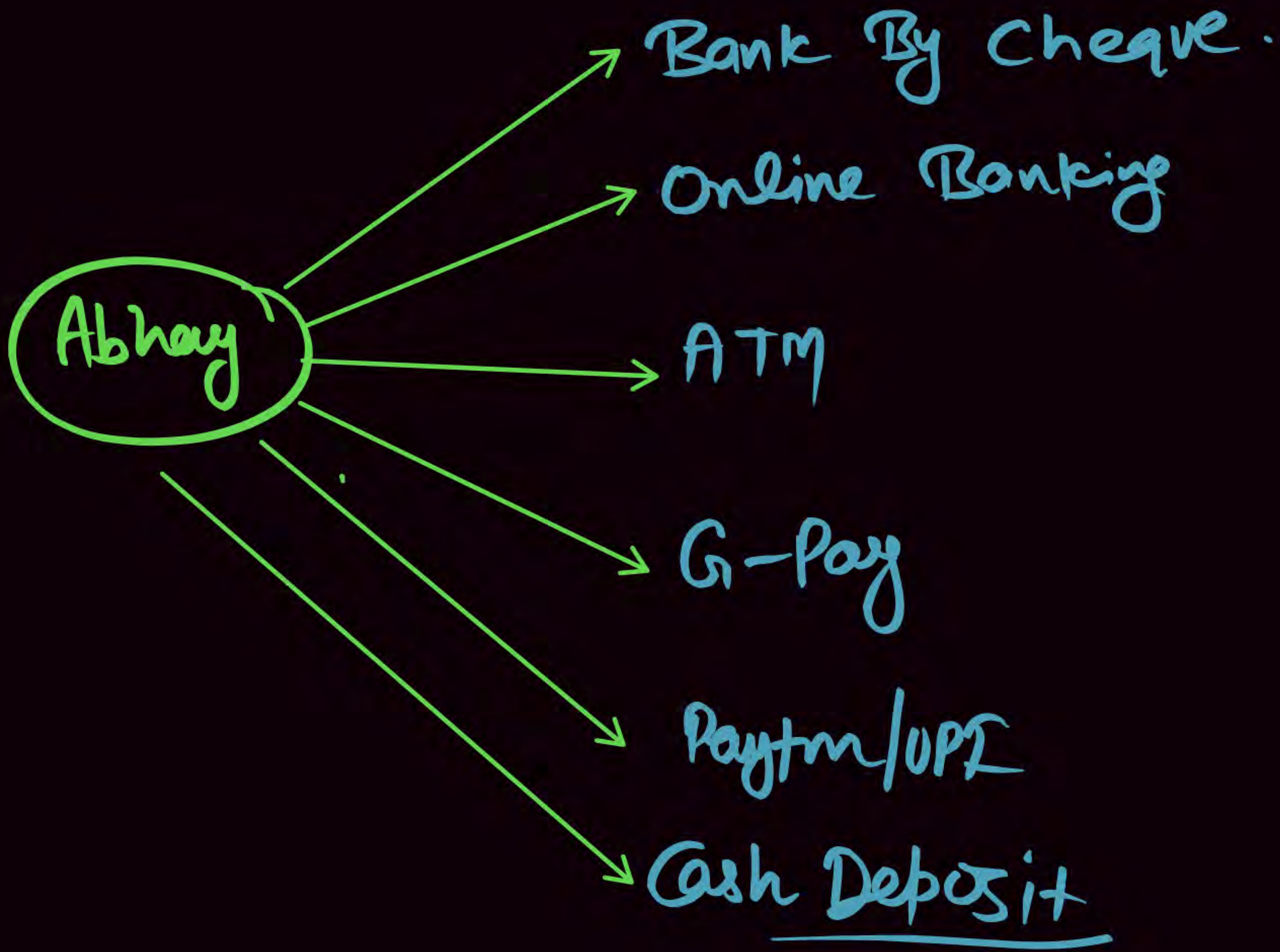
eg

Before	AFTER
A : 4000	A : 3500
B : 2000	B : 2500
6000	6000

'500'

③ Isolation: When Two or More Transaction execute Concurrently then isolation Come into Picture.

Concurrent Execution of Two or More transaction should be equal to Any serial schedule.


Abhay
Bank By Cheque.
Online Banking
ATM
G-Pay
Paytm/UPI
Cash Deposit

④ <u>Durablity</u>: Any change in the Database Must Persist for Long Period of time.

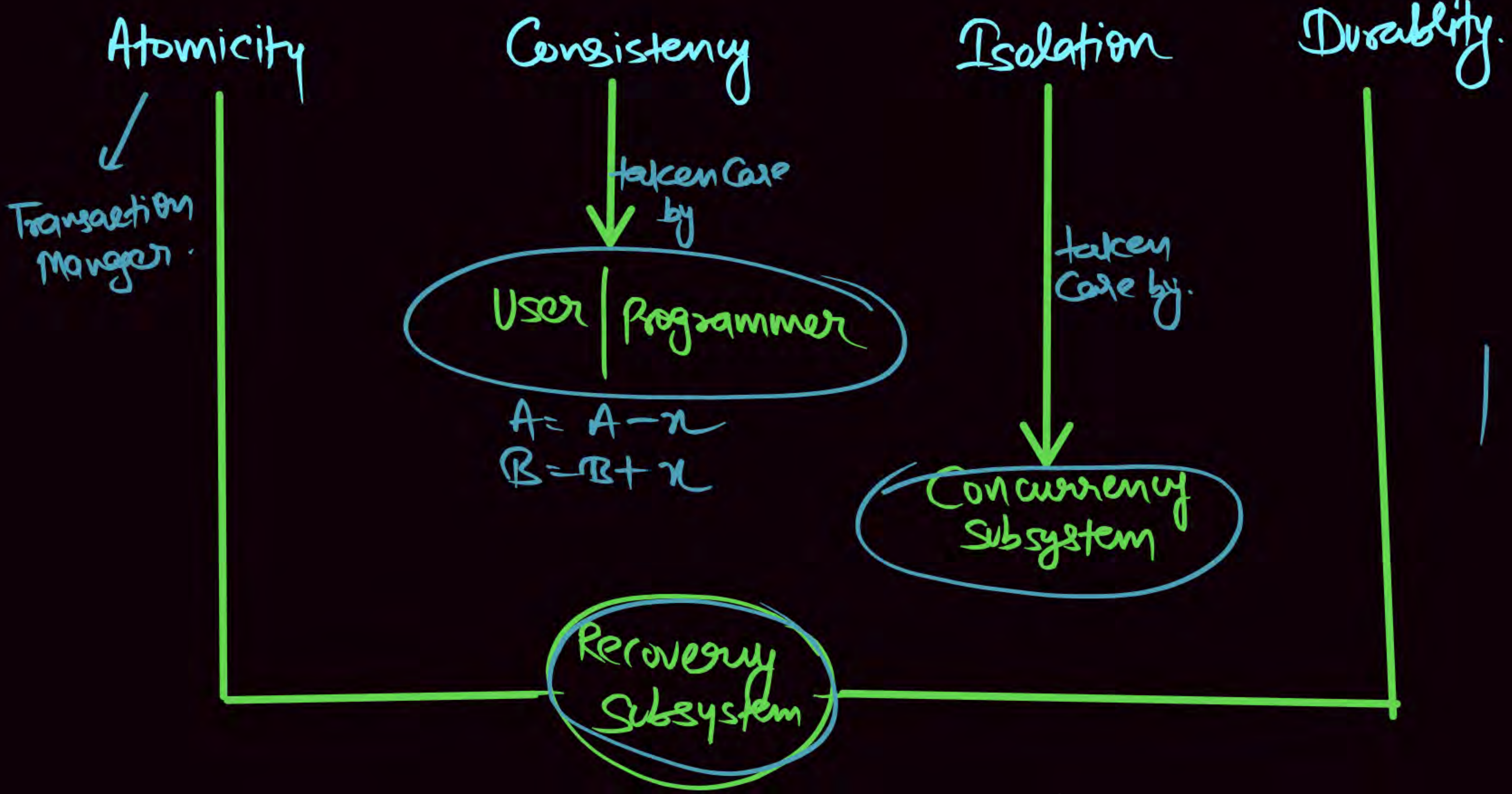
Database Must be able to Recover Under Any Case of failure.

ACID Properties

- **Atomicity:** Either all operations of the transaction are properly reflected in the database or none are.
- **Consistency:** Execution of a transaction in isolation preserves the consistency of the database.
- **Isolation:** Although multiple transactions may execute concurrently, each transaction must be unaware of other concurrently executing transactions. Intermediate transaction results must be hidden from other concurrently executed transactions.

- That is, for every pair of transactions T_i and T_j, it appears to T_i that either T_j, finished exection before T_i started, or T_j started execution after T_i finished.

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


Atomicity
Consistenty
Isolation
Durablity.
Transaction Manager.
taken Care by
User | Programmer
A = A − x
B = B + x
taken Care by.
Concurrency Subsystem
Recovery Subsystem

Transaction State

- (1) Active : the initial state; the transaction stays in this state while it is executing.
- (2) Partially committed : after the final statement has been executed.
- (3) Failed: after the discovery that normal execution can no longer proceed.
- (4) Aborted: after the transaction has been rolled back and the database restored to its state prior to the start of the transaction. Two options after it has been aborted:

- Restart the transcation
  - Can be done only if no internal logical error
- Kill the transaction

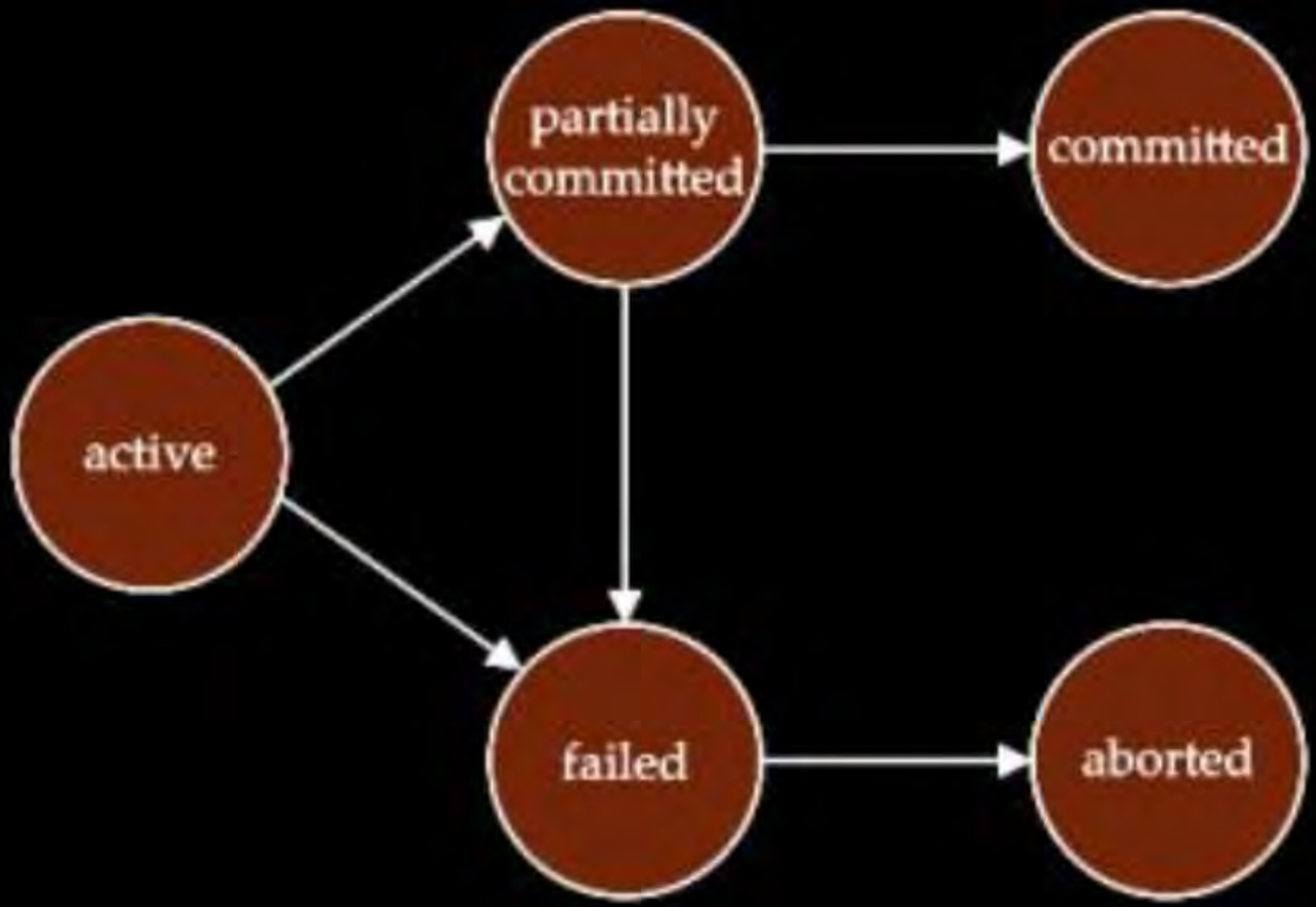
5 **Committed:** After successful completion.

100 99

1 X GATE

Imp X

Transaction State (Cont.)

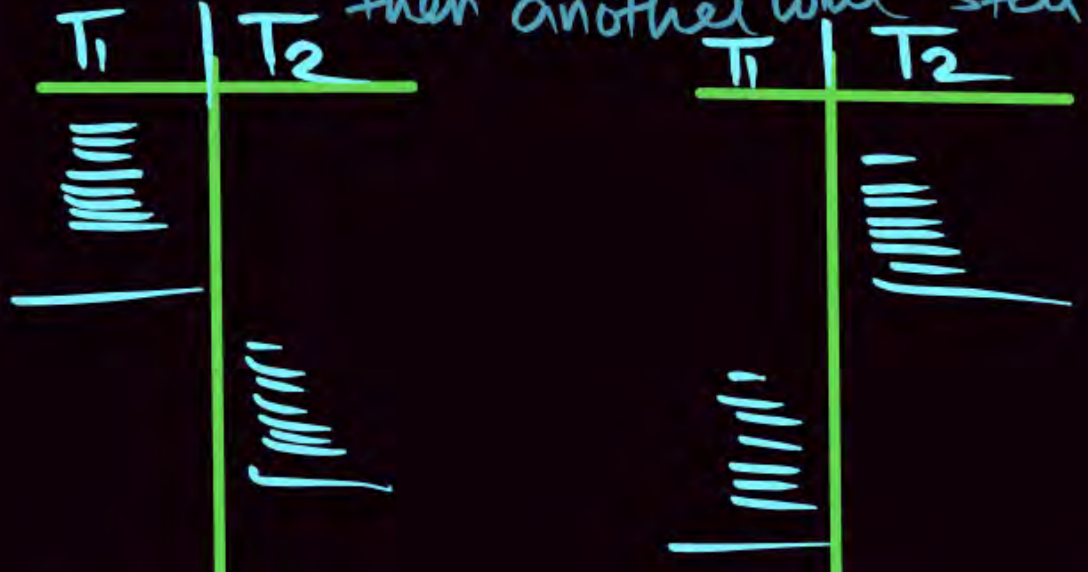


Schedule

[Time order sequence of Two or More transaction]

① Serial Schedule

Execution of one transaction Successfully then another will start.



<T1, T2>
<T1 followed by T2>

<T2, T1>
<T2 followed by T1>

② Non serial Schedule

{Concurrent Execution}

Interleaved execution of Two or More Transaction.

& Many More

eg. If 2 Transaction $\Rightarrow$ Serial Schedule = 2

① $\langle T_1, T_2 \rangle$

(T_1 followed by T_2)

② $\langle T_2, T_1 \rangle$

(T_2 followed by T_1)

eg. If 3 Transaction then $= 3! = 6$ Serial Schedule

$\langle T_1\ T_2\ T_3 \rangle$

$\langle T_1, T_3, T_2 \rangle$

$\langle T_2, T_1\ T_3 \rangle$

$\langle T_2\ T_3\ T_1 \rangle$

$\langle T_3\ T_1\ T_2 \rangle$

$\langle T_3, T_2, T_1 \rangle$

Note If there are n Transaction then $n!$ serial schedule.

Schedules

- **Schedule:** 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.

Commit ✓

Successfully executed.

Abort ✗

Bank
Practical example of Serial & Non Serial Schedule.

2 Transaction then $2! = 2$ Serial Schedule

$\langle T_1, T_2 \rangle$ T_1 followed by T_2

$\langle T_2, T_1 \rangle$ T_2 followed by T_1.

$A \xrightarrow[100Rs]{} B$

T_1

Read(A)

A = A - 100

Write(A)

Read(B)

B = B + 100

Write(B)

T_2 $A \xrightarrow[10\%]{} B$

Read(A)

temp = A * 0.10

A = A - temp

Write(A)

Read(B)

B = B + temp

Write(B)

SERIAL SCHEDULE:

Let T_1 transfer 100 Rs from A to B, and T_2 transfer 10% of the balance from A to B.

Schedule 1

Before: A=2000 +B=3000 = 5000

T_1: A $\xrightarrow{100Rs}$ B; $T_1 \xrightarrow{10\%} T_2$

T_1	T_2
read (A) A=2000	
A: = A – 100 ⇒ 2000-100 =1900	
write (A) → A=1900	
read (B) B=3000	
B: = B + 100 3100	
write (B) → B=3100	
commit	
	read (A) A=1900
	temp := A * 0.1 temp=190 (1900 - 190 = 1710)
	A := A – temp A=1710
	write (A)
	read (B) B=3100
	B := B + temp 3100+190(temp)
	write (B) B=3290
	Commit

After: A=1710 +B: 3290 = 5000 Consistent

$S_1 < T_1\ T_2 >$

Serial schedule in which T_1 is followed by T_2:

Schedule 2

Before: A=2000 +B=3000 = 5000

T_1	T_2
	read (A) A=2000
	temp := A * 0.1 temp=200
	A := A – temp 2000-200
	write (A) A=1800
	read (B) B=3000
	B := B + temp 3000+200
	write (B) B=3200
	Commit
read (A) A=1800	
A: = A – 100 1800-100	
write (A) A=1700	
read (B) B=3200	
B: = B + 100 3200+100	
write (B) B=3300	
commit	

AFTER: A: 1700 +B: 3300 = 5000 Consistent

$S_2 < T_2\ T_1 >$

serial schedule where T_2 is followed by T_1

Now Non Serial Schedule.

NON SERIAL SCHEDULE [Concurrent execution]

A = 2000
+B = 3000
5000

Schedule 3

T_1	T_2
read (A) A=2000	
A: = A – 100	
write (A) A=1900	
	read (A) A=1900
	temp := A * 0.1 temp=190
	A := A – temp 1900-190
	write (A) A=1710
read (B) B=3000	
B: = B + 100	
write (B) B=3100	
commit	
	read (B) B=3100
	B := B + temp 3100+190
	write (B) B=3290
	Commit

A : 1710
+B : 3290
5000
Consistent

C_1

A : 2000
+B : 3000
5000

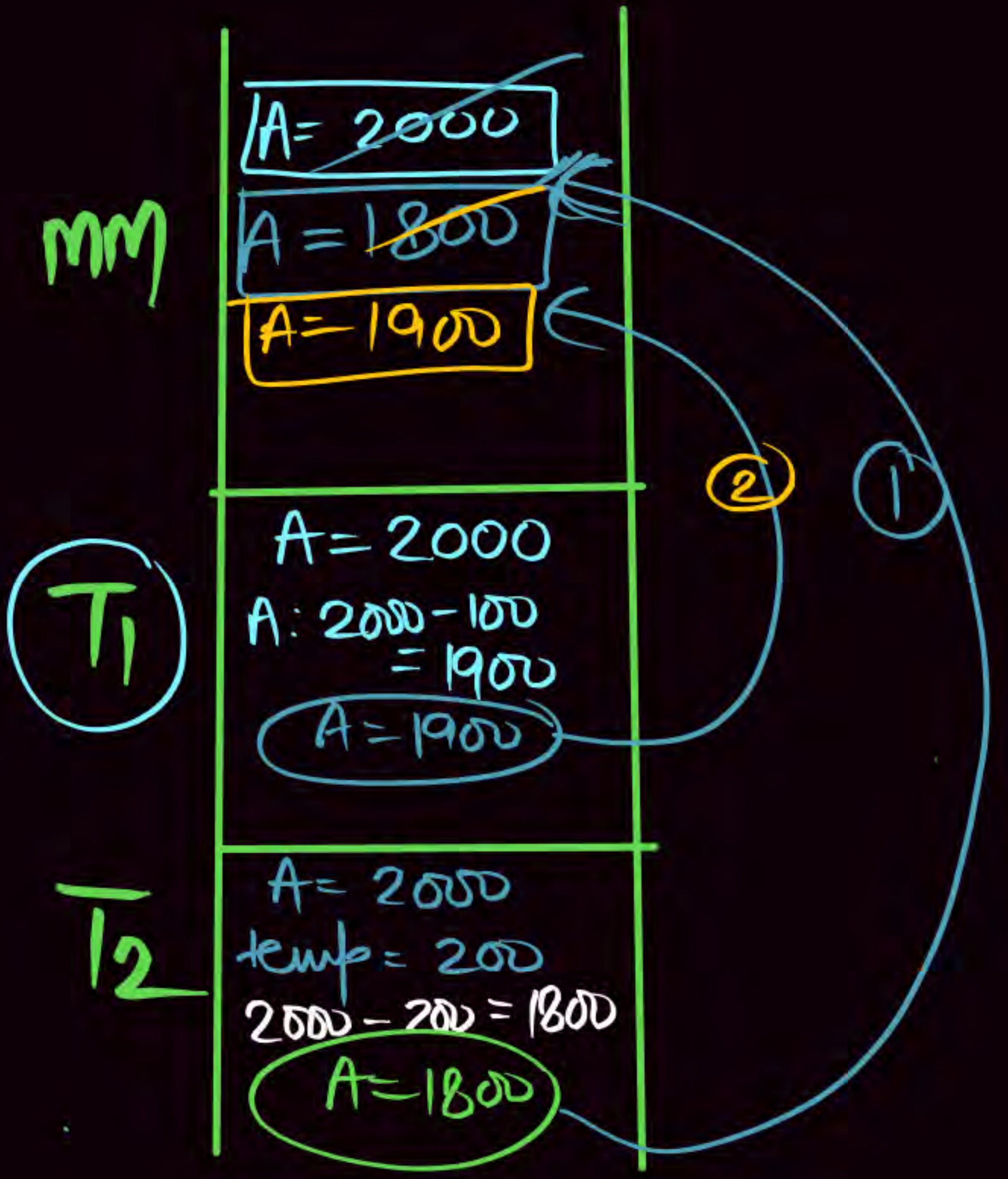
Schedule 4

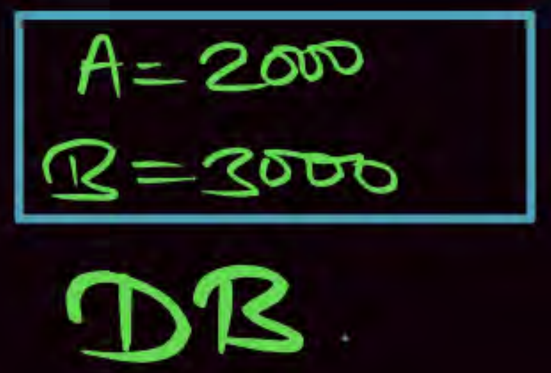
T_1	T_2
A=2000 read (A)	
A: = A – 100 2000-100 =1900	
	read (A) A=2000
	temp := A * 0.1 temp = 200
	A := A – temp
	write (A) A=1800
A=1900 write (A)	
B=3000 read (B)	
B: = B + 100	
B=3100 write (B)	
commit	
	read (B) B=3100
	B := B + temp 3100+200=3300
	write (B) B=3300
	Commit

A : 1900
+B : 3300
5200
InConsistent

C_2

C2:





C2

(a) 5000
(b) 5100
(c) 5200
(d) None of these

Note

Serial Schedule [All Serial Schedule (n!)]
are always Consistent.

Note

Non Serial Schedule [Concurrent Execution]
May or May Not be Consistent.

But we execute Concurrent Execution.

Why Concurrent Execution ? (Non Serial Schedule)

- To Improve CPU Utilization
- Enhanced Throughput
- Fast Response
- Waiting time Reduce.
- Effective Utilization of Multiprocessor
- etc

Serializable Schedule

Serial Schedule

- After Commit of one transaction, begins (Start) another transaction.
- Number of possible serial Schedules with 'n' transactions is "n!"
- The execution sequence of Serial Schedule always generates consistent result.

Example

S : $R_1(A)$ $W_1(A)$ Commit (T_1) $R_2(A)W_2(A)$ commit (T_2).

Advantage

- Serial Schedule always produce correct result (integrity guaranteed)

as no resource sharing.

Disadvantage

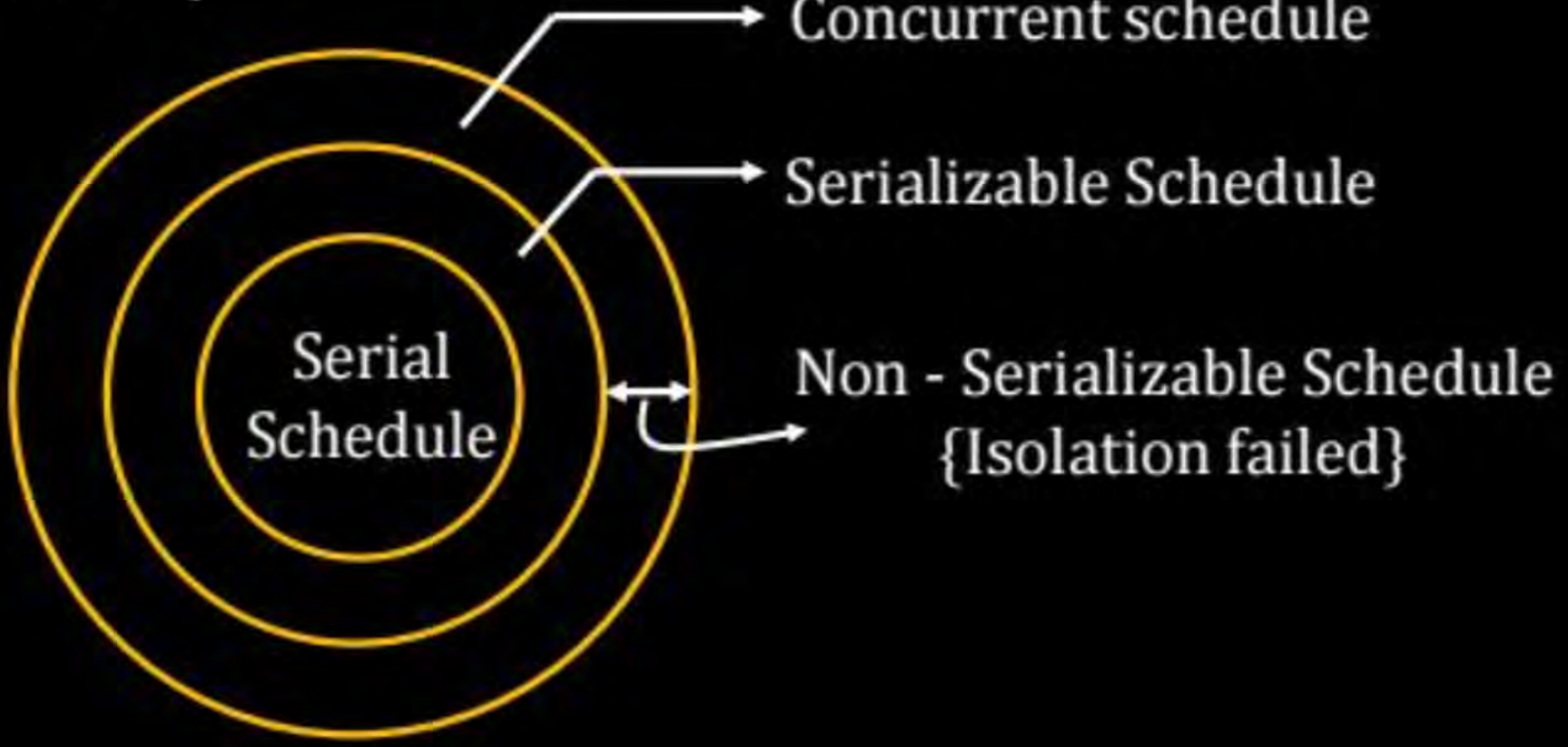
- Less degree of concurrency.
- Through put of system is low.
- It allows transactions to execute one after another.

Serializable Schedule

A Schedule is serializable Schedule if it is equivalent to a Serial Schedule.

(i)Conflict Serializability

(ii)View Serializability



Serializability

- Basic Assumption: Each transaction preserves database consistency.
- Thus, serial execution of a set of transactions preserves database consistency.
- A (possibly concurrent) schedule is serializable if it is equivalent to a serial schedule. Different forms of schedule equivalence give rise to the notions of:
  1. Conflict serializability
  2. view serializability

Conflict Serializability

- If a schedule S can be transformed into a schedule S′ by a series of swaps of non-conflicting instructions, we say that S and S′ are conflict equivalent.
- We say that a schedule S is conflict serializable if it is conflict equivalent to a serial schedule.

Conflict Serializability (Cont.)

- Schedule 3 can be transformed into Schedule 6, a serial schedule where T_2 follows T_1, by series of swaps of non-conflicting instructions. Therefore Schedule 3 is conflict serializable.

Schedule 3

T_1	T_2
read (A)	
Write (A)	
	read (A)
	write (A)
read (B)	
write (B)	
	read (B)
	write (B)

Schedule 6

T_1	T_2
read (A)	
write (A)	
read (B)	
write (B)	
	read (A)
	write (A)
	read (B)
	write (B)

Conflict Serializability (Cont.)

- Example of a schedule that is not conflict serializable:

T_3	T_4
read (Q)	
	write (Q)
write (Q)	

- We are unable to swap instructions in the above schedule to obtain either the serial schedule < T3, T4 >, or the serial schedule < T4, T3 >

Conflict Serializable

A schedule is said to be conflict serializable if it is conflict equivalent to a serial schedule.

Same conflicting operation order in C_1 & S_1

$\therefore$ Its $\{C_1\}$ conflict is conflict serializable.

T_1	T_2
read(A)	
write(A)	
	read(A)
	write(A)
read(B)	
write(B)	
	read(B)
	write(B)

C_L

T_1	T_2
read(A)	
write(A)	
read(B)	
write(B)	
	read(A)
	write(A)
	read(B)
	write(B)

S_L

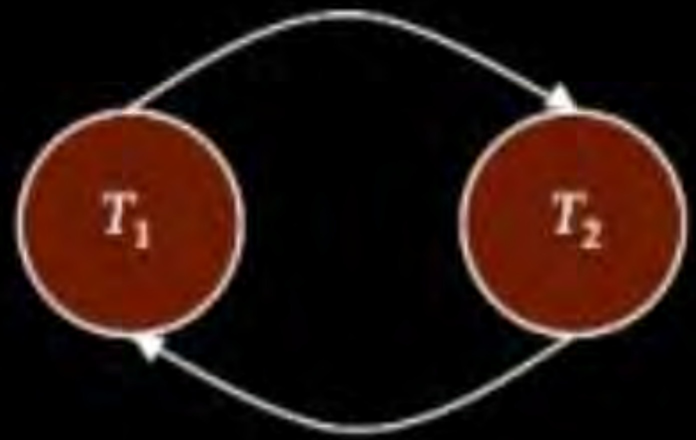
Conflicting Instructions

- Instructions l_i, and l_j of transactions T_i and T_j respectively, conflict if and only if there exists some item Q accessed by both l_i, and l_j, and at least one of these instructions wrote Q.
  1. l_i, = read(Q), l_j = read(Q). l_i and l_j don't conflict.
  2. l_i, = read(Q) l_j = write(Q). They conflict.
  3. l_i, = write(Q) l_j = read(Q). They conflict
  4. l_i= write(Q) l_j = write(Q). They conflict
- Intuitively, a conflict between l_i and l_j forces a (logical) temporal order between them.
  - If l_i, and l_j are consecutive in a schedule and they do not conflict, their results would remain the same even if they had been interchanged in the schedule.

Testing for Serializability

- Testing for conflict serializability.
  - Consider some schedule of a set of transactions $T_1, T_2, \ldots T_n$
  - Precedence graph — a direct graph where the vertices are the transactions (names).
  - We draw an arc from T_i to T_j if the two transaction conflict, and T_i accessed the data item on which the conflict arose earlier.
  - We may label the arc by the item that was accessed.

Example:



A schedule is conflict serializable if and only if its precedence graph is acyclic.

NOTE: CNC [Cycle not conflict serializable]

Q. $S: R_1(A)\ W_1(A)\ R_2(A)\ W_2(A)\ R_1(B)\ W_1(B)\ R_2(B)\ W_2(B)$

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

Q. $R_1(A)\ R_2(A)\ W_2(A)\ W_1(A)\ R_1(B)\ W_1(B)\ R_2(B)\ W_2(B)$

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

Q. $R_1(A)\ W_1(A)\ R_2(B)\ W_2(B)\ R_1(B)\ W_1(B)\ R_2(A)\ W_2(A)$

Important Point 1:

1. If S_1, S_2 Schedule are conflict equal then precedence graph of S_1 and S_2 must be same.
2. If S_1 and S_2 have same precedence graph then S_1 and S_2 may or may not conflict equal.

Q. Consider the following schedules involving two transactions. Which one of the following statements is TRUE?

S_1: $r_1(X); r_1(Y); r_2(X); r_2(Y); w_2(Y); w_1(X)$

S_2: $r_1(X); r_2(X); r_2(Y); W_2(Y); r_1(Y); w_1(X)$

[2007: 2 Marks]

A. Both S_1 and S_2 are conflict serializable

B. S_1 is conflict serializable and S_2 is not conflict serializable

C. S_1 is not conflict serializable and S_2 is conflict serializable

D. Both S_1 and S_2 are not conflict serializable

Q.

Consider the following four schedules due to three transactions (indicated by the subscript) using read and write on a data item x, denoted by r(x) and w(x) respectively. Which one of them is conflict serializable?

[2014(Set-1): 2 Marks]

(A) $r_1(x); r_2(x); w_1(x); r_3(x); w_2(x)$

(B) $r_2(x); r_1(x); w_2(x); r_3(x); w_1(x)$

(C) $r_3(x); r_2(x); r_1(x); w_2(x); w_1(x)$

(D) $r_2(x); w_2(x); r_3(x); r_1(x); w_1(x)$

Q.

Let $r_i(z)$ and $w_i(z)$ denote read and write operations respectively on a data item by a transaction T_i. Consider the following two schedules.

S_1: $r_1(x)\ r_1(y)\ r_2(x)\ r_2(y)\ w_2(y)\ w_1(x)$

S_2: $r_1(x)\ r_2(x)\ r_2(y)\ w_2(y)\ r_1(y)\ w_1(x)$

Which one of the following options is correct? [MCQ: 2021: 2M]

(A) S_1 is conflict serializable, and S_2 is not conflict serializable.

(B) S_1 is not conflict serializable, and S_2 is conflict serializable.

(C) Both S_1 and S_2 are conflict serializable.

(D) Neither S_1 nor S_2 is conflict serializable.

Any Doubt ?

THANK YOU!