→ Transaction & concurrency Control

Just like Process of O.S.

Transaction is set of logically related op^n to perform unit of work.

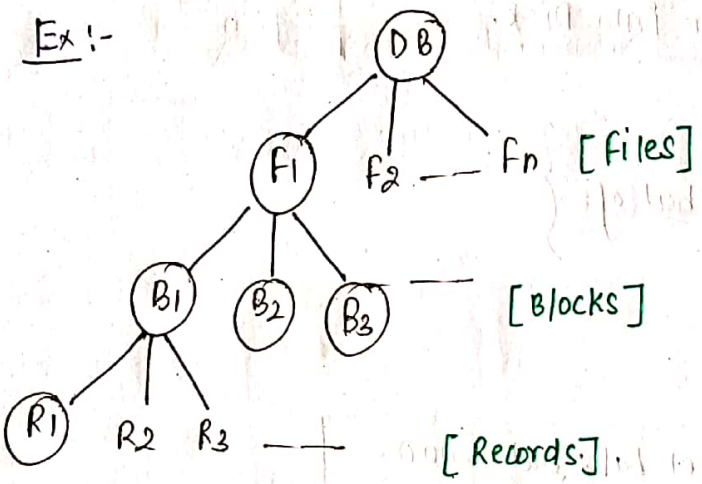
O.S → Process

DBMS → Transaction.

Data Item :- [Shared Resource].

It is a DB element which required to access by many transactions.

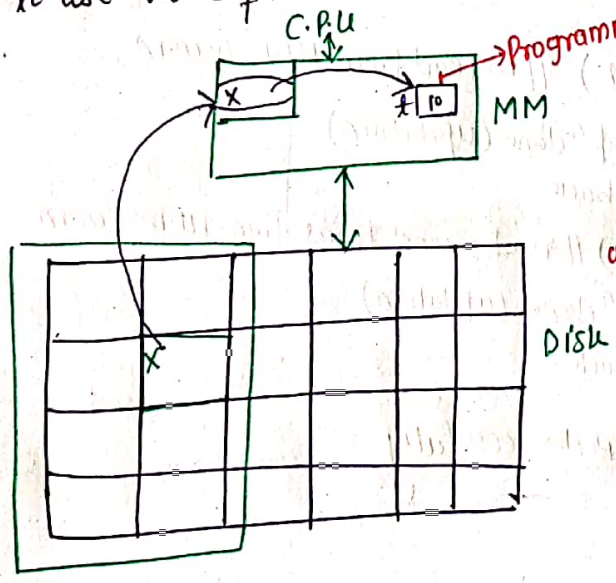
Ex :-



Main op^n of Transaction :-

Read (X) ; X : data item

Access data item X from DB file (Disk) to MM (Programmed Variable) to use value of X in transaction logic.



Read (X) :-

atomic

1) Find block from DB file which consist data item X.
2) Transfer Block from Disk to MM
3) Find Address of X in MM Block & & copy Val of X in programmed Variable.

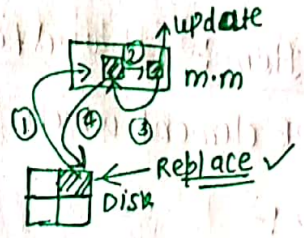
Write(x):-

Updation of data item x in DB file (Disk)

Write(x)

Atomic

1). Find block from DB file (Disk) which consist data Item x.
2) Transfer block to m.m.
3) Find Address of x in m.m && update x.
4) Replace updated block into Disk.



Ex

Account (Cid, bal)

Transfer 5000 from bal of Cid:101 to Cid:102

```
begin trans
    Update Account set bal=bal-5000
    where cid=101;
    Update Account set bal=bal+5000
    wher cid=102;
end trans;
```

⇩

```
begin Trans Ti
    Read(bal of cid:101)  // read from HDD to m.m
    bal=bal-5000;  // opn done (updation)
    write(bal)  // write back
    Read(bal of cid:102)  // read second Trans from HDD to m.m
    bal=bal+5000  // opn done (updation)
    write(bal)  // write back.
    Commit  // Trans executed sucessfully
end Trans Ti
```

• Account (Cid bal)

Set bal of Cid 101, Cid 102 into 5000

```
begin
    update Account set bal=5000 where cid=101;
    update Account set bal=5000 where cid=102;
end
```

Read & write op^{n} of Transaction are atomic in Nature.

```
begin Trans T2
    write(bal: 5000, cid=101)
    write(bal: 5000, cid=102)
    Commit
end Trans
```

writting without reading → K/a blind write.

Blind write :-

```
Trans(T3)
    R(A)
    R(B)
    W(A)
    W(C)  } → Blind write → write without read
    W(D)  }
    Commit
```

ACID Properties :-

To preserve Integrity (correctness) transactions must satisfy ACID Rules.

DBMS:
- A - Atomicity } Maintained by Recovery mgmt component of DBMS s/w.
- D - Durability }
- I - Isolation } maintained by Concurrency control component DBMS s/w.

User:
- C - consistency } maintained by user-DBA / DB developer / DB user.

RMC } Basic → Recovery manager component ✓

CCC } **

concurrency control component ✓

Atomicity :-

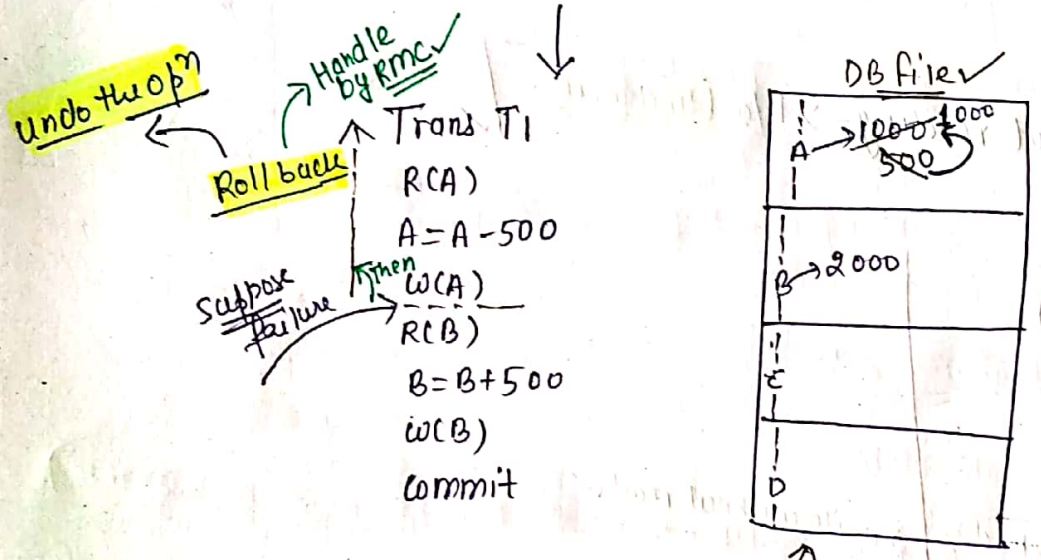
Execute all opⁿ of Transaction including commit // Commit ✓

(or)

Execute None opⁿ of Transaction // Roll back ✓

Before transaction Termination

Ex :- Trans (T_1) : Transfer 500 from A to B.



Atomicity Violates ← due to failure → Incorrect data ✓

↳ {do Rollback} ← done by Recovery manager.

Recovery management Component :-

It handle Rollback transaction if Transaction failed any where before commit.

Rollback (Abort)

Undo modifications of DBFile which are done due to failure transactions.

Transaction log :-

* → Record all the activity ✓

File maintained by Recovery manager component in Disk to record all activities of transaction.

(Trans log)

1) T_1 begins
2) T1 R, A, 1000
3) T_1 W, A, 1000, 500
4) T_2 R, B, 2000
5) T_2 W, B, 2000, 2500
6) T_1 Rollback;

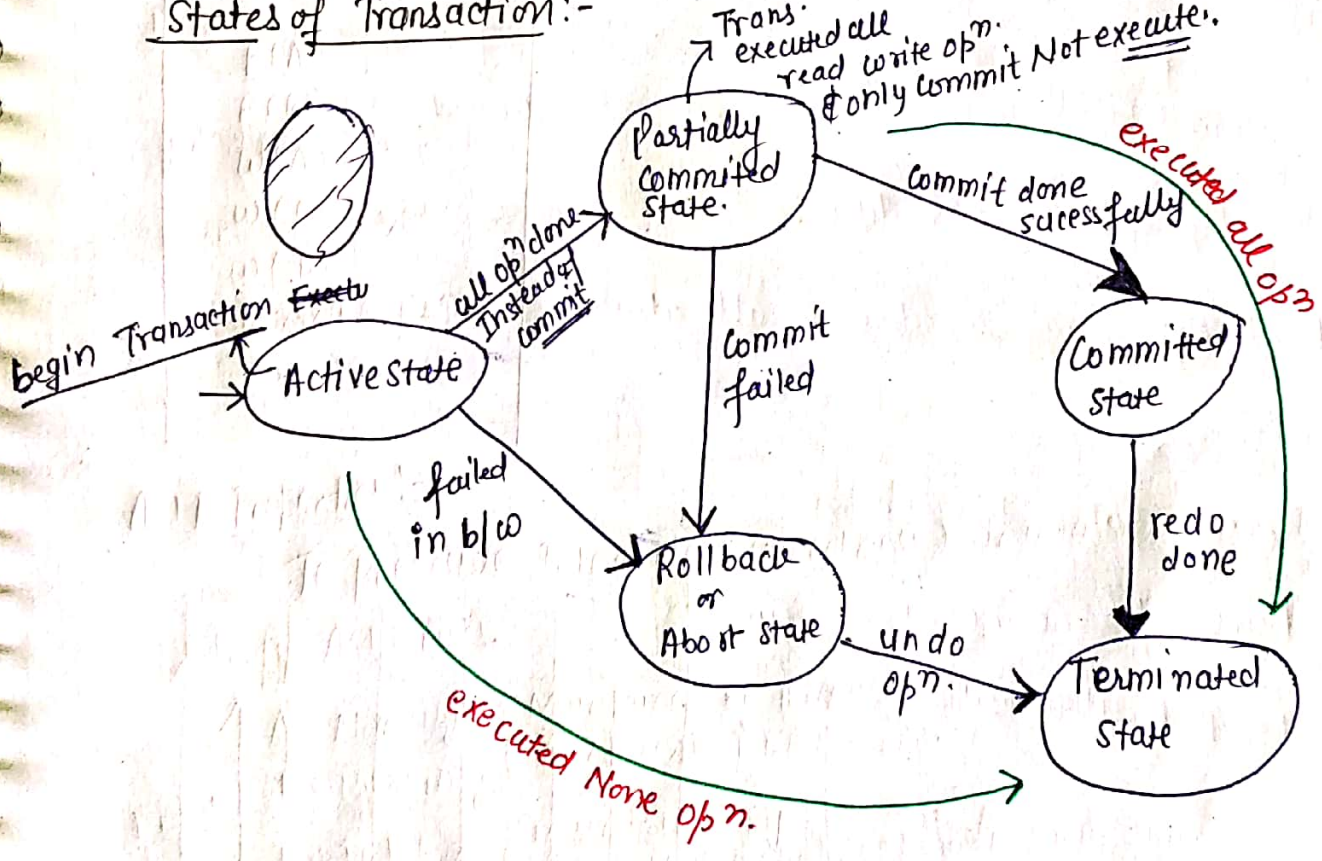
→ It is the Transaction log & It Record all the activity.

Dirty bit :- Block which is updated by uncommitted transaction.
→ transaction commit hone ke baad dirty bit ka stamp hata do.

Redo opⁿ :- All modifications of DB file because of Transaction Commit. ↓

Undo opⁿ :- All modifications of DB file because of transaction Roll back. ↑

States of Transaction :-

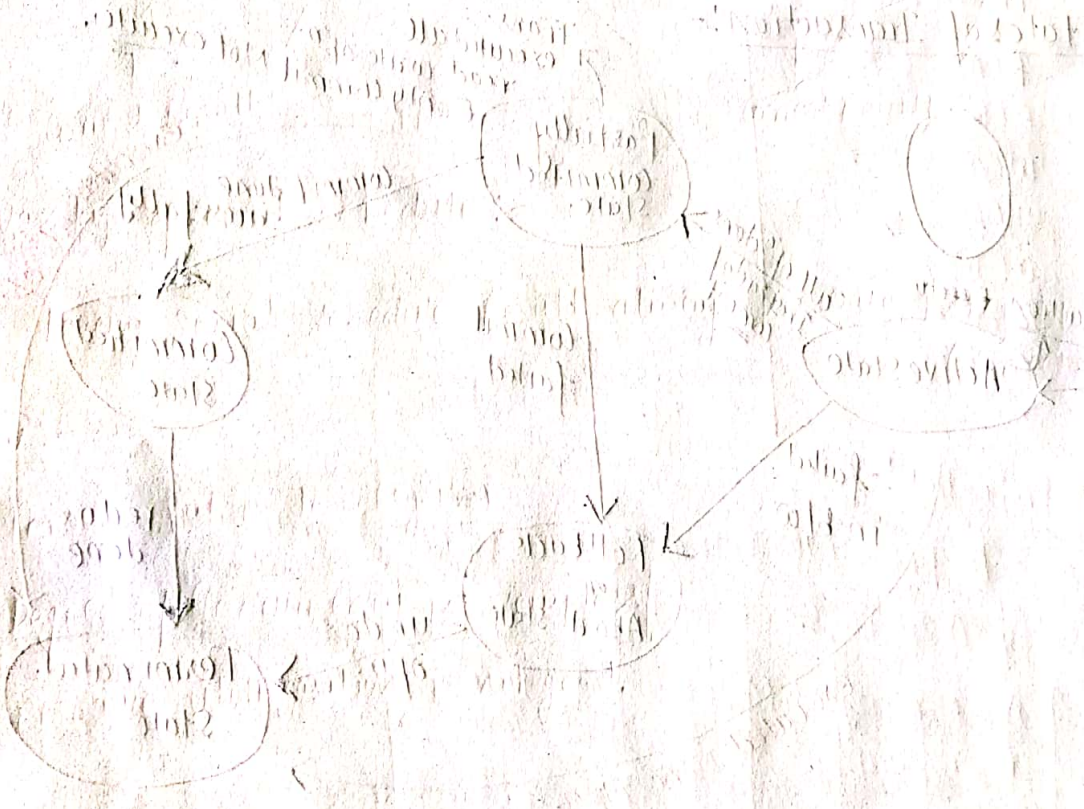


Active state :-

Transaction begins Executions.

Partially Committed state

Transaction Executed all R/w op^n & & only commit not done.



:- Lecture :- 21 :-

Durability :-

Transaction must be able to recover under any case of failure.

Transaction failure due to :-

- → Power failure
- → S/w Crash :- OS overloaded or DBMS overloaded.
- → Concurrency control protocol may kills transaction
- → H/w Crash :- (Disk Crash) ↔ RAID design of Disk must required (Redundant Array of Independent disks)

Can Recover easily. Bcoz Transaction log in Disk in the

Must be recover so, it will be durable

For recovery we Need :-

Transaction state + Transaction log

But when HDD Crashed Then

stored in HDD ✓

Consistency :- [maintained by user].

User requested operations to perform transaction must be logically correct. { If one transactions are from x to y then consistency says that sum of accounts x & y should remain constant at any point of time known as consistency }

Isolation :-

⇨ Concurrent execution of two or more transactions result must be equal to result of any serial execution of transaction.

⇨ Schedule :- Time order execution sequence of two / more transaction.

S:	T_1	T_2	T_3
1		R(A)	
2	R(A)		
3			R(B)
4		W(A)	
5			W(B)
6	W(A)		

⇒ $S: r_2(A) r_1(A) r_3(B) w_2(A) w_3(B) w_1(A)$

Serial schedule :-

Transaction must execute one after other.

T_1: Transfer 500 from A to B. $[r_1(A)\ w_1(A)\ r_1(B)\ w_1(B)\ C_1]$

T_2: display (A+B) $[r_2(A)\ r_2(B)\ C_2]$

(S1)

T_1	T_2
$r_1(A)$	
$w_1(A)$	
$r_1(B)$	
$w_1(B)$	
commit	
	$r_2(A)$
	$r_2(B)$
	Commit

$(T_1 \rightarrow T_2)$ serial

↓

Correct Result

(S2)

T_1	T_2
	$r_2(A)$
	$r_2(B)$
	Commit
$r_1(A)$	
$w_1(A)$	
$r_1(B)$	
$w_1(B)$	
Commit	

$(T_2 \rightarrow T_1)$ serial

↓

Correct Result

Advantage of serial schedule :-

① Every serial schedule Result always preserve Integrity (Correctness)

Disadv :-

① Through-Put is less (system)

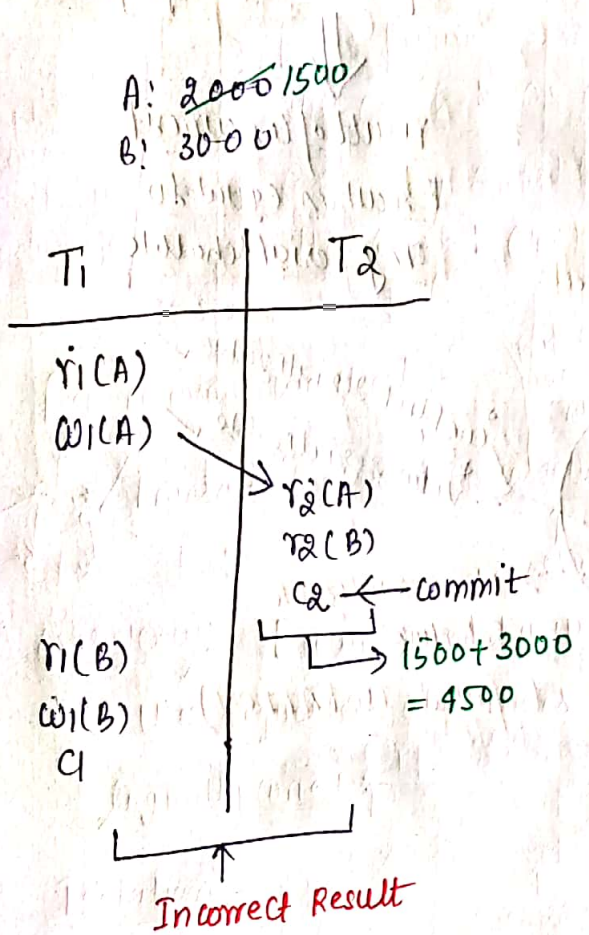
② Degree of Concurrency is very less

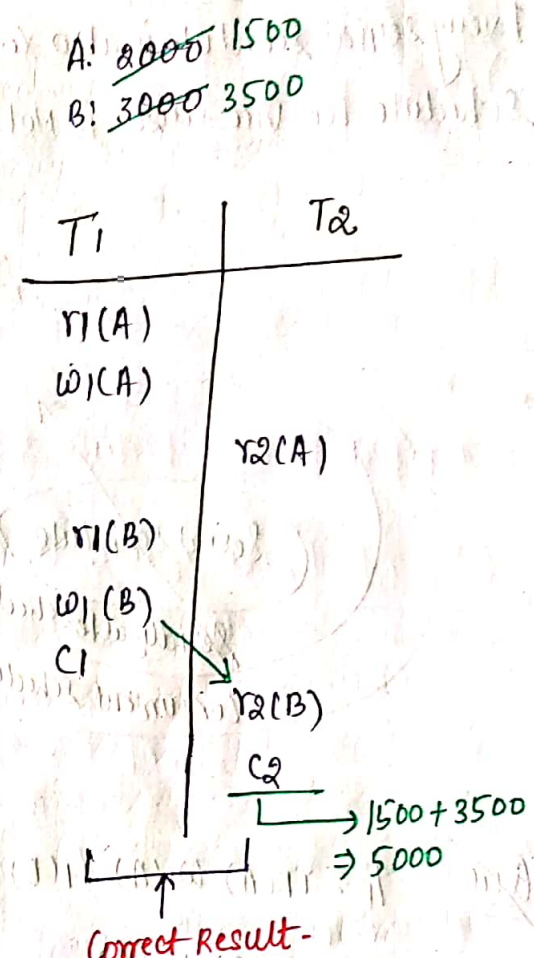
③ Resource utilization is not good.

To Increase degree of concurrency

→ Use Concurrent schedule :-

Simultaneous / Interleaved / concurrent execution of two or more transaction.



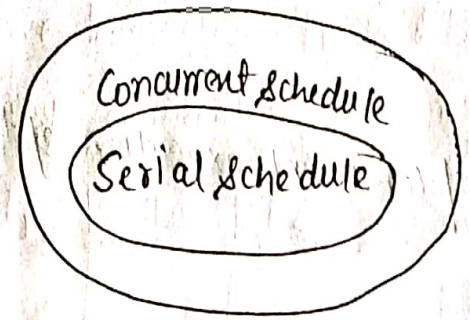


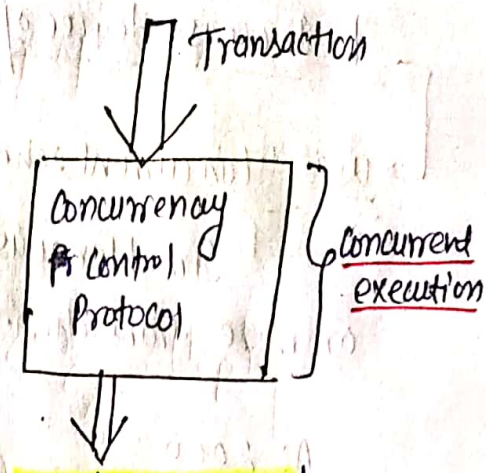
So, Concurrent Result may be correct or may be Incorrect

But when we found Concurrent Result is correct it means that Concurrent programming some where follow serial schedule.

Isolation (Serializable schedule)

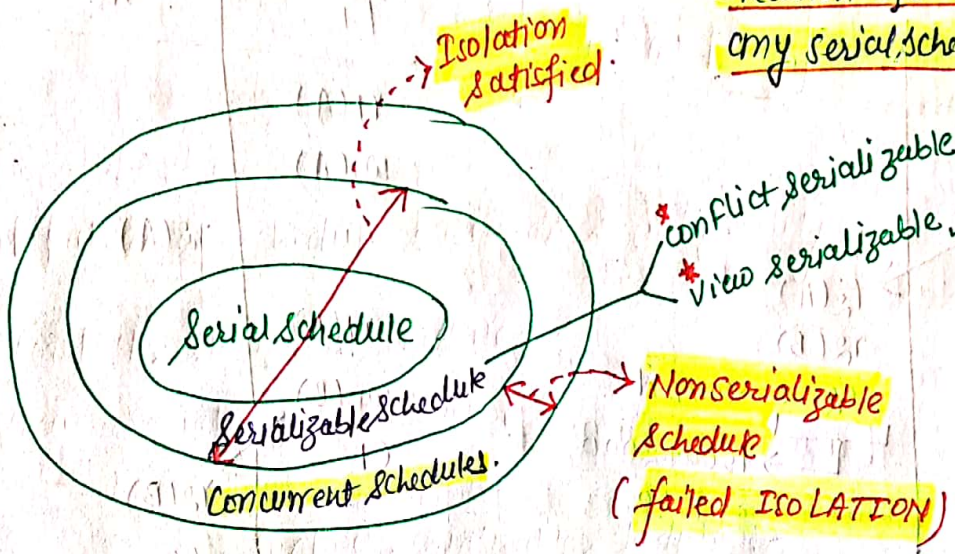
Concurrent schedule Result must be equal to result of any serial schedule.





* Every serial schedule is also Concurrent Schedule but Vice versa is Not true.

result of Concurrent Execution equal to any serial schedule

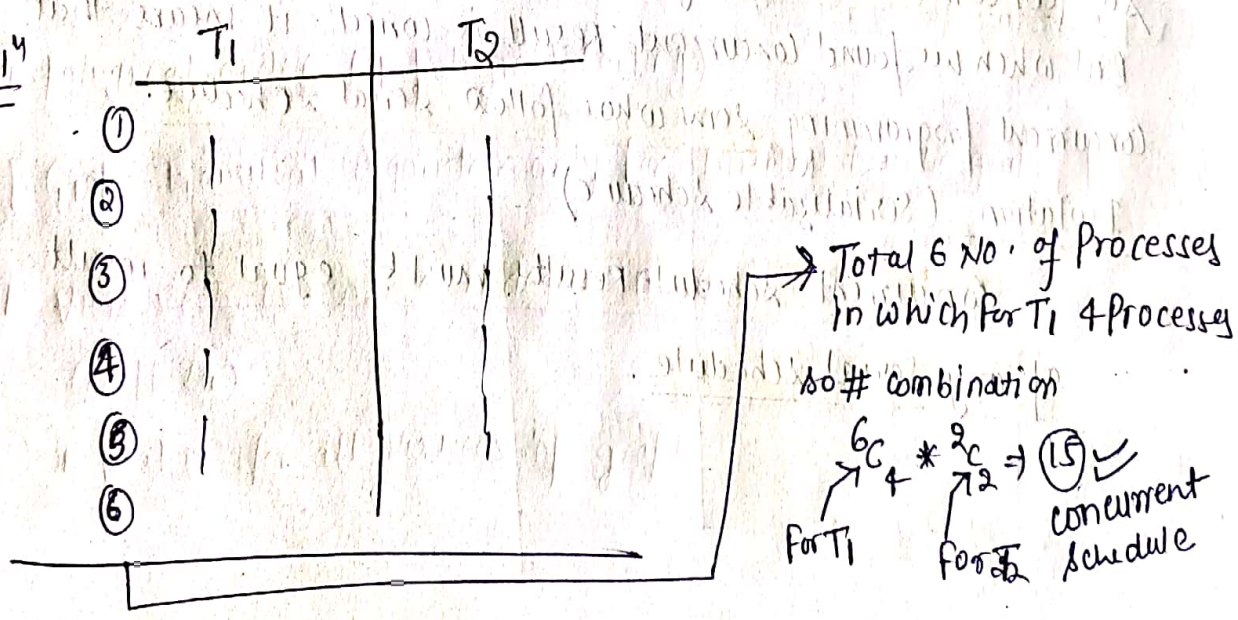


Q. T_1 : $r_1(A)$ $w_1(A)$ $r_1(B)$ $w_1(B)$

T_2 : $r_2(A)$ $r_2(B)$

How many concurrent schedules possible?

Sol^n



No. of serial schedule

If their are n No. of Processes.

$T_1, T_2, T_3, T_4 \text{ ---- } T_n$

Then How many serial schedules are Possible.

Sol^n for three process

$T_1\ T_2\ T_3$

$T_1\ T_2\ T_3$
$T_1\ T_3\ T_2$
$T_2\ T_1\ T_3$
$T_2\ T_3\ T_1$
$T_3\ T_1\ T_2$
$T_3\ T_2\ T_1$

6 serial schedules are Possible ✓

So, **For n Processes → n! serial schedules are Possible**

No. of concurrent schedule :-

T_1 & T_2 are Transaction with n & m op^n each.

How many concurrent schedule possible?

So:

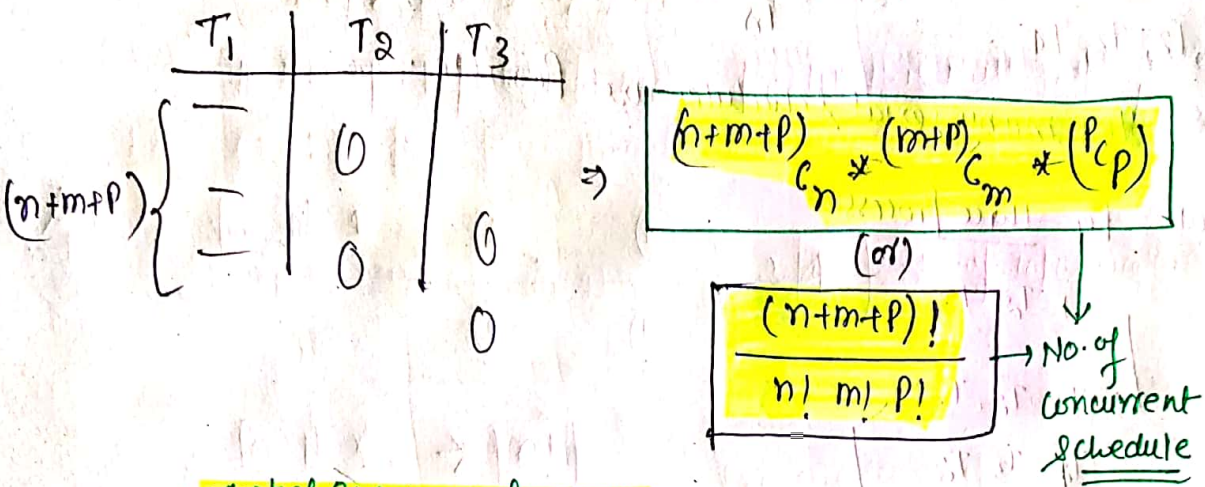
	T_1	T_2
(n+m)	—	o
	—	o
	—	

$\Rightarrow$ $^{(n+m)}C_n * {}^{m}C_m$ → # concurrent schedules

$\Downarrow$

$\frac{(n+m)!}{n!\ m!}$ → concurrent schedule.

⇒ For Three transaction T_1, T_2 & T_3 · with n, m, p op^n each Howmany Concurrent schedule are possible?



*** ↗ Most of Que comes from Here

Classification of schedule based on Serializability :-

1. ~~Conflict~~ Serializable schedule ✓
2. ~~View~~ Serializable schedule ✓

÷ Conflict Serializable schedule :-

SubTopics
1. Topological order
2. Conflict Pairs
3. Precedence graph
4. conflict & equal schedule.

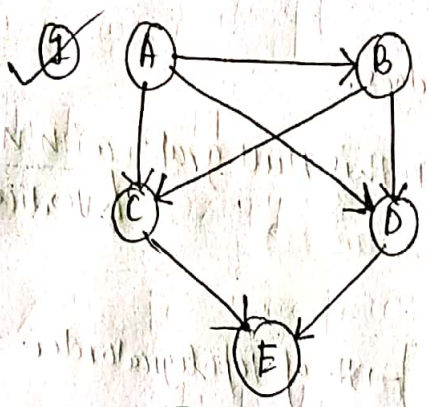
(1) Topological order (Topological sorting)

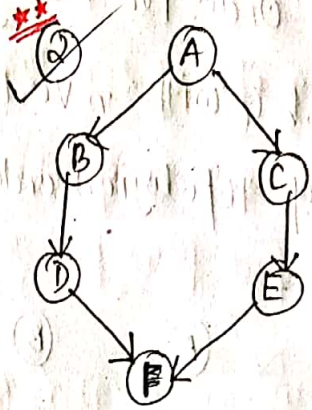
[Graph Traversal Algorithm only for Directed Acyclic graph]

(1) Visit Vector (V) whose Indegree "0" and delete V from Graph (G)

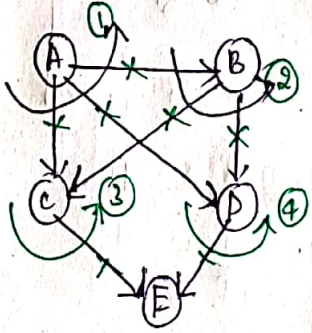
(2) Repeat (1) for all Vertices of graph

Q. Find Topological order of.



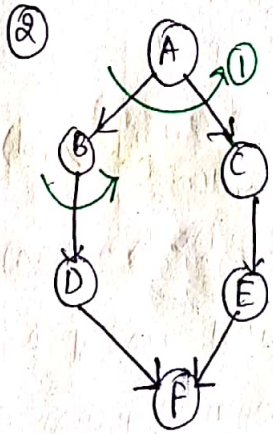


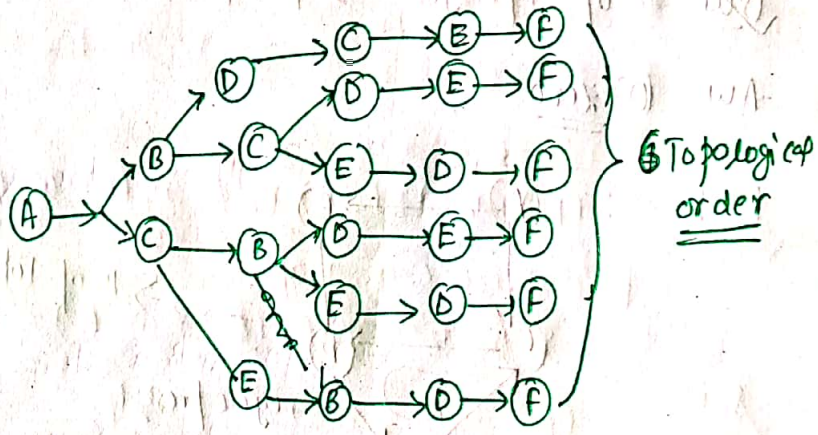
Soln (1)



A → B → C → D → E
(or)
A → B → D → C → E
→ Topological order.

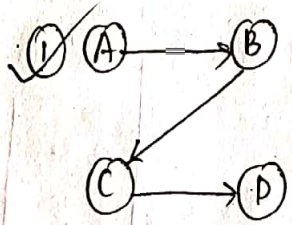
(2)

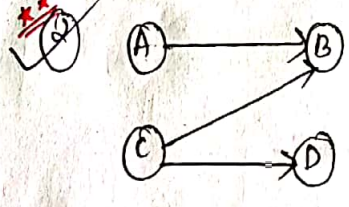




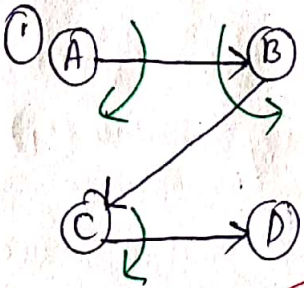
6 Topological order

Q. Find No. of To Pological orders.

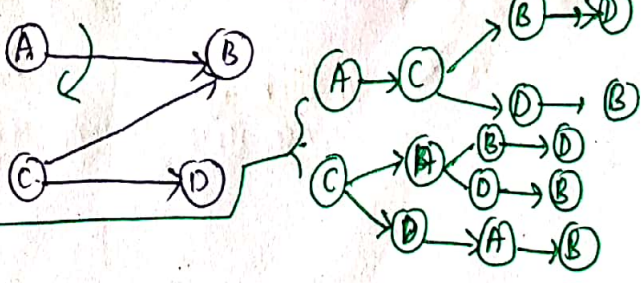




Soln (1)



⇒ A → B → C → D → 2 Topological order



5 Topological orders

Que Topological Order for Null graph.

Graph G with n vertices, 0 edges.

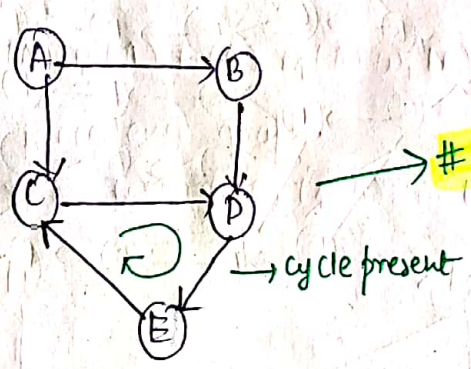
No. of Topological order for like this Null graph with n vertices.

Ans for N=4 (A), (B), (C), (D) → 4! → # of Topological order

for N=n → n!

Que # of Topological graph for cyclic graph.

Ans (0) zero



→ # of Topological order ≥ 0 ⇒ (zero) ✓

Lecture -"22"

→ Conflict Pair :-

Pair of operations from schedule (S) such that :—

***
1] Atleast one write opⁿ
& 2] Over same data item
& 3] from different Transaction.

S : ---- $r_i(x)$ ---- $w_j(x)$ ----
S : ---- $w_i(x)$ ---- $r_j(x)$ ----
S : ---- $w_i(x)$ ---- $w_j(x)$ ----
} Conflict Pair ✓

***
S : ---- $r_i(x)$ ---- $r_j(x)$ ----
S : ---- $r_i(x)$ / $w_i(x)$ ---- $r_j(y)$ / $w_j(y)$ ----
} Non conflict pair ✓

S : T1 | T2

$r_1(A)$
$r_1(B)$
$w_1(B)$ ---- Conflict Pair ✓ ---- $w_2(A)$

$r_1(A)$ ---- $w_2(A)$ } → Conflict Pair
$r_1(B)$ ---- $w_2(A)$ }
$w_1(B)$ ---- $w_2(A)$ } → Non-conflict Pair

***
• Operations from same Transaction are Neither conflict pair nor Non-conflict pair.

Ex: $r_1(A)$ $r_1(B)$
$r_1(A)$ $w_1(B)$
$r_1(B)$ $w_1(B)$
} Neither conflict Nor Non conflict

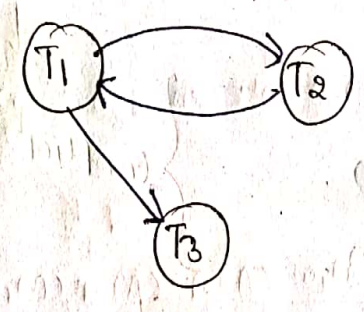
Precedence Graph:

Precedence graph of schedule (S).

Vertices(V): Transactions of schedule (S)

Edges (E): Conflict pairs precedences of schedule (S)

Ex: S: $r_1(A)$ $w_2(A)$ $w_1(A)$ $r_1(B)$ $w_1(B)$ $w_3(B)$



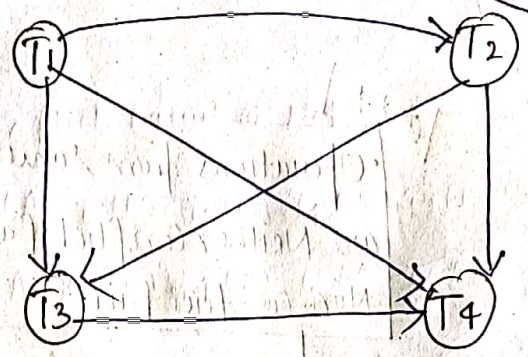
} Precedence graph.

Ex: S: $r_1(A)$ $r_2(A)$ $r_3(A)$ $r_4(A)$ $w_1(B)$ $w_2(B)$ $w_3(B)$ $w_4(B)$

Find precedence graph.

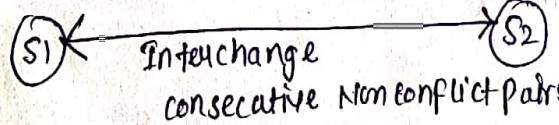
~~$r_1(A)$ $r_2(A)$ $r_3(A)$ $r_4(A)$~~ $w_1(B)$ $w_2(B)$ $w_3(B)$ $w_4(B)$

Soln



Conflict Equal Schedule:-

S_1, S_2 scheduled are conflict equal iff scheduled (S_2) derived by Interchanging of consecutive non-conflict pairs of schedule (S_1)



Interchange consecutive Non conflict pairs. => {Then S1, S2 are Conflict equal.}.

Ex ① Test given $S_1 S_2$ conflict equal.

S_1: $r_1(A)\ r_2(A)\ r_3(A)\ w_1(B)\ w_2(B)\ w_3(B)$

S_2: $r_1(A)\ w_1(B)\ r_3(A)\ r_2(A)\ w_2(B)\ w_3(B)$

Soln

S_1: $r_1(A)\ r_2(A)\ r_3(A)\ w_1(B)\ w_2(B)\ w_3(B)$ (② ①)

$r_1(A)\ r_2(A)\ r_3(A)\ w_1(B)\ w_2(B)\ w_3(B)$ (③)

$r_1(A)\ w_1(B)\ r_3(A)\ r_2(A)\ w_2(B)\ w_3(B)$

→ after 3 consecutive Non conflict pair swapping we get S_2

S_2: $r_1(A)\ w_1(B)\ r_3(A)\ r_2(A)\ w_2(B)\ w_3(B)$

⇓

S_1 & S_2 are conflict equal

*** Ex ②

S_1: $r_1(A)\ w_1(A)\ r_2(A)\ r_2(B)\ r_1(B)\ w_1(B)$

S_2: $r_1(A)\ w_1(A)\ r_1(B)\ w_1(B)\ r_2(A)\ r_2(B)$

Test given S_1, S_2 for conflict equal.

Soln

S_1: $r_1(A)\ w_1(A)\ r_2(A)\ r_2(B)\ r_1(B)\ w_1(B)$ (① ②)

$r_1(A)\ w_1(A)\ r_1(B)\ r_2(A)\ r_2(B)\ w_1(B)$ → Conflict Pair, So, Not allow To swap

S_2: $r_1(A)\ w_1(A)\ r_1(B)\ w_1(B)\ r_2(A)\ r_2(B)$

⇓

S_1, S_2 are not Conflict equal.

*** Conclusion:- (Method 2 to check conflict equal)

S_1 ← Interchange Consecutive Non-Conflict Pairs. → S_2 (S_1, S_2 conflict equal)

S_1, S_2 Conflict equal iff —

*** ① each transaction of S_1 must be exactly same transaction in S_2.

It will be my method

Ex

$r_i(A)$, $r_i(B)$, $w_i(B)$ ≡ $r_i(A)$, $r_i(B)$, $w_i(B)$

$S_1 \to T_1 \equiv S_2 \to T_1$

$S_1 \to T_2 \equiv S_2 \to T_2$

→ same order

② Every conflict pair precedence of S_1 must be exactly same precedence in S_2.

Ex ①

S_1: $r_1(A)\ w_1(A)\ r_2(A)\ r_2(B)\ r_1(B)\ w_1(B)$ → T_1: $r_1(A)\ w_1(A)\ r_2(B)\ w_1(B)$ ✓

T_2: $r_2(A)\ r_2(B)$ ✓

Not equal

S_2: $r_1(A)\ w_1(A)\ r_1(B)\ w_1(B)\ r_2(A)\ r_2(B)$

↓

So, S_1 & S_2 are Not conflict equal ✓

Ex ②

S_1: $r_1(A)\ r_2(A)\ r_3(A)\ w_1(B)\ w_2(B)\ w_3(B)$

S_2: $r_1(A)\ w_1(B)\ r_3(A)\ r_2(A)\ w_2(B)\ w_3(B)$

Sol^n S_1: $r_1(A)\ r_2(A)\ r_3(A)\ w_1(B)\ w_2(B)\ w_3(B)$

$T_1 = r_1(A)\ w_1(B)$ ✓

$T_2 = r_2(A)\ w_2(B)$ ✓

$T_3 = r_3(A)\ w_3(B)$ ✓

S_2: $r_1(A)\ w_1(B)\ r_3(A)\ r_2(A)\ w_2(B)\ w_3(B)$

↳ S_1 & S_2 are conflict Equal.

Que Test for conflict equal or Not.

S_1: $r_1(A)\ r_2(A)\ r_3(A)\ r_4(A)\ w_1(B)\ w_2(B)\ w_3(B)\ w_4(B)$

S_2: $r_1(A)\ w_1(B)\ r_4(A)\ r_2(A)\ r_3(A)\ w_2(B)\ w_3(B)\ w_4(B)$

Sol^n S_1: $r_1(A)\ r_2(A)\ r_3(A)\ r_4(A)\ w_1(B)\ w_2(B)\ w_3(B)\ w_4(B)$

T_1: $r_1(A)\ w_1(B)$ ✓

T_2: $r_2(A)\ w_2(B)$ ✓

T_3: $r_3(A)\ w_3(B)$ ✓

T_4: $r_4(A)\ w_4(B)$ ✓

S_2: $r_1(A)\ w_1(B)\ r_4(A)\ r_2(A)\ r_3(A)\ w_2(B)\ w_3(B)\ w_4(B)$

↓

$(S_1 = S_2)$ → So, S_1 & S_2 are conflict equal. ✓

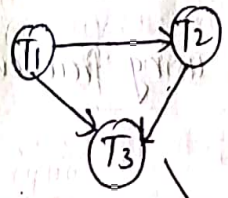
⇨ if S_1, S_2 conflict equal then S_1, S_2 Precedence graph must be same.

⇨ if S_1, S_2 schedules precedence graph not equal Then S_1, S_2 schedules not conflict equal.

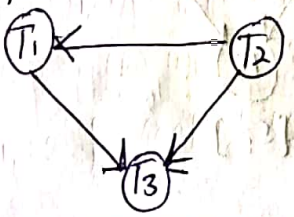
Ex S_1: $r_1(A)\ w_1(B)\ w_2(B)\ w_3(B)$

S_2: $r_1(A)\ w_2(B)\ w_1(B)\ w_3(B)$

→ Precedence graph



→ Precedence graph:



S_1 & S_2 are not conflict equal

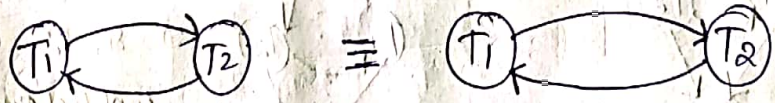
Imp *** But:- if S_1, S_2 schedule with same precedence graph Then Schedule may/ may not conflict equal. (when cycle present in graph)

Ex S_1:

T_1	T_2
$r_1(A)$	
	$w_2(A)$
$w_1(A)$	
	$r_2(A)$

S_2:

T_1	T_2
$r_1(A)$	
	$w_2(A)$
	$r_2(A)$
$w_1(A)$	



Hence, we can use precedence graph to check that schedule are not equal

↓

for schedule equality we can use general method.

If S_1, S_2 schedules with —

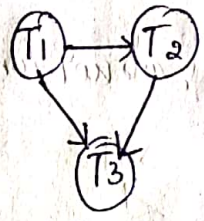
a] same precedence graph
&&
b] Acyclic precedence graph.

→ Conflict equality Test using Precedence graph ✓

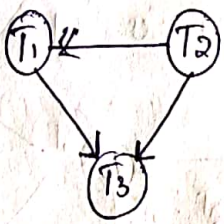
Thus $S_1 S_2$ are conflict equal ⇔ Acyclic + same precedence graph

Conflict equality Test using Precedence graph

Case-I



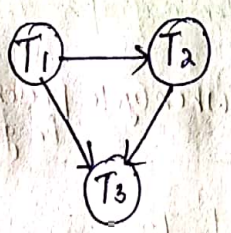
[S1]



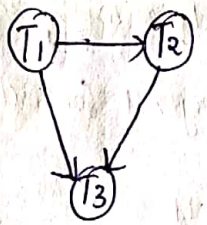
[S2]

⇒ S_1, S_2 not conflict equal

Case-II



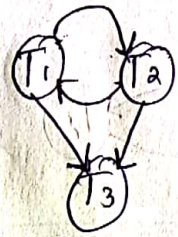
[S1]



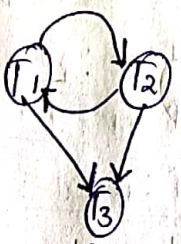
[S2]

⇒ S_1, S_2 conflict equal

Case-III ✓



[S1]



[S2]

⇒ S_1, S_2 may/maynot conflict equal.

Conflict serializable schedule:-

Schedule (S) is conflict serializable iff (some) serial schedule('s) must be conflict equal to schedule (S)

{ Given schedule [S] } ⟷ (conflict equal) { serial schedule(S') }

⇑

conflict serializable.

[Precedence Graph of S must be Acyclic] ⟷ (Conflict Equal) [Serial schedule Precedence graph Acyclic]

↳ && [Topological order ⟷ (conflict equal) serial Schedules.]

** So, schedule (S) is conflict serializable schedule.

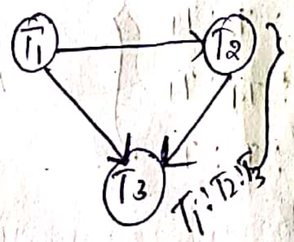
iff - " Precedence Graph of Schedule (S) must be Acyclic".

&&

Conflict equal serial schedules are Topological order of Schedule (S) precedence graph.

Ex ⇒ S: $r_2(A), r_1(B), w_2(A), w_3(A), w_2(B), w_1(C), w_3(C)$

Sol^n



} S is conflict serializable ✓

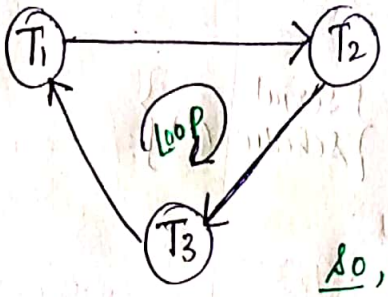
& conflict Equal serial order is

↳ [$T_1 : T_2 : T_3$]

** [Serial schedule precedence graph is always Acyclic]

Que S: $r_1(A)\ w_2(A)\ w_2(B)\ r_3(B)\ w_3(C)\ r_1(C)$ Test schedule (S) conflict serializable or not ??

Sol^n :-



→ Not a conflict serializable ✓

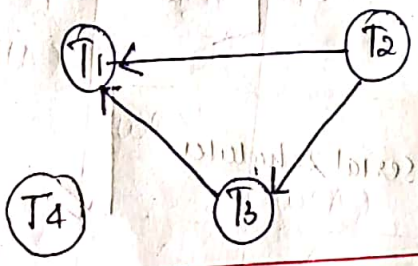
Bcoz of cycle is found (loop)

So, [No. serial schedule conflict equal to schedule (S)]

Que S: $r_1(A)\ r_2(A)\ r_3(A)\ r_4(A)\ w_2(B)\ w_3(B)\ w_1(B)$

How many serial schedules are conflict equal to schedule (S)?

Sol^n



$T_2 : T_3 : T_1$ → Total 4 Topological orders.

T_4 → It can be any where

{# of serial schedules conflict equal to schedule (S)} = {# of Topological orders of schedules) precedence Graph}

- $T_4 : T_2 : T_3 : T_1$
- $T_2 : T_4 : T_3 : T_1$
- $T_2 : T_3 : T_4 : T_1$
- $T_2 : T_3 : T_1 : T_4$

÷ lecture-23 :-

View Serializable Schedule :- ✓

Schedule (s) is **View serializable iff some serial schedule(s) must be view equal to schedule (s)** //

[Given Schedule (s)] ⟺ (View equal) {Some serial Schedule} ✓

View equality Testing :-

S_1, S_2 schedule is a view equal.

iff ① Initial Read :- **Initial Read Same for all schedule S_i.**

S_1 — { T_i → R(x) } S_2 — { T_i → R(x) }

Ex

S_1	T_1	T_2
	(IR) R(A)	
W(A)		
	R(A)	

S_2	T_1	T_2
	(IR) R(A)	
		R(A)
W(A)		

Not View equal ✓

Every IR of S_1 must be also Initial Read in S_2.

Not Initial Read Bcoz it is firstly updated by T_1 then it read

② Updated Reads :-

* **Every updated read of S_1, S_2 must be same.**

Ex

S_1 :

T_i	T_j
W(x)	
	R(x)

S_2 :

T_i	T_j
W(x)	
	R(x)

Not View equal ≠

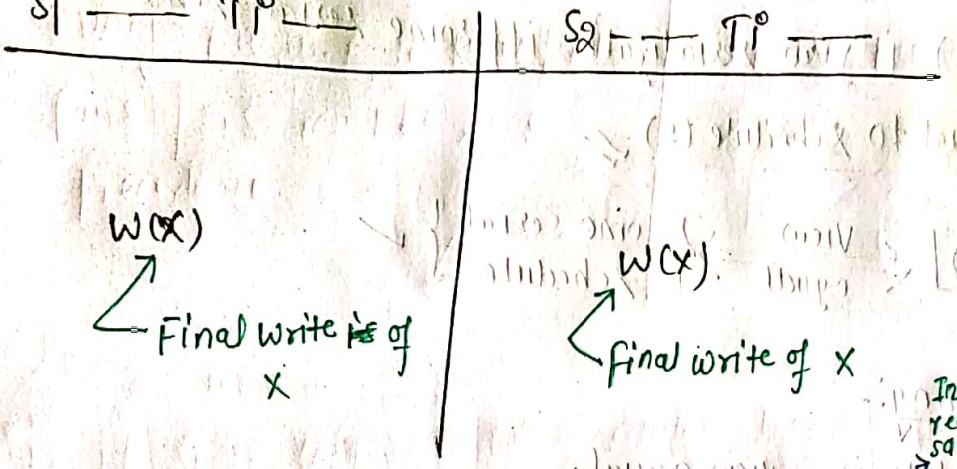
(S1) :

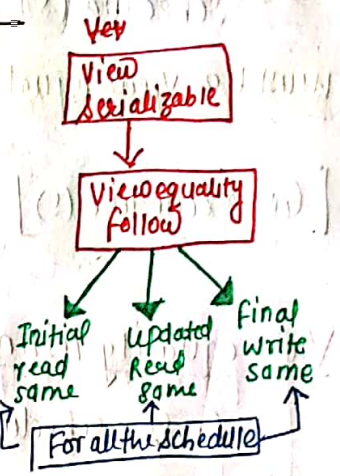
T_1	T_2	T_3
W(A)		
	W(A)	
		R(A)

(S2) :

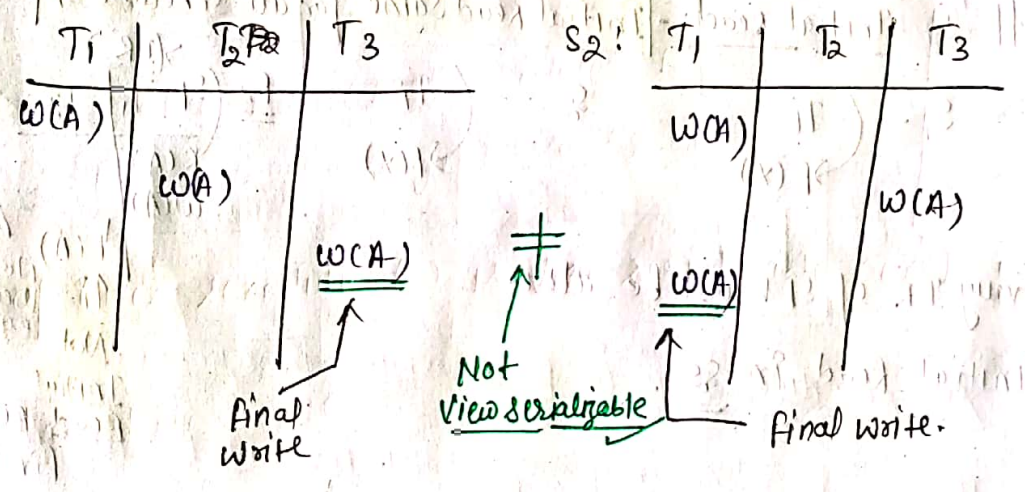
T_1	T_2	T_3
W(A)		
		R(A)
	W(A)	

③ Final write :- Final of S_1 & S_2 must be same.





Ex :-

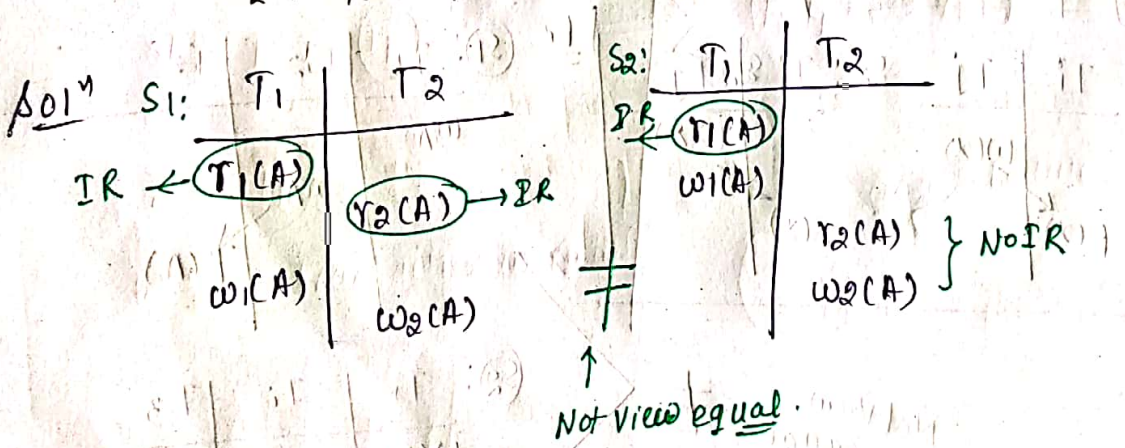


Q Test S_1, S_2 is ~~View~~ schedule ~~or not~~ View equal or Not?

① S_1 : $r_1(A)\ r_2(A)\ w_1(A)\ w_2(A)$

S_2 : $r_1(A)\ w_1(A)\ r_2(A)\ w_2(A)$

Sol^n



Que ② S_1 : $r_1(A)$ $r_2(A)$ $r_3(A)$ $w_1(B)$ $w_2(B)$ $w_3(B)$

S_2 : $r_2(A)$ $r_1(A)$ $r_3(A)$ $w_2(B)$ $w_1(B)$ $w_3(B)$.

Test for S_1 & S_2 are Viewequal or Not.

Sol^n

S_1:

T_1	T_2	T_3
$r_1(A)$		
	$r_2(A)$	
		$r_3(A)$
$w_1(B)$		
	$w_2(B)$	
		$w_3(B)$

S_2:

T_1	T_2	T_3
	$r_2(A)$	
$r_1(A)$		
		$r_3(A)$
	$w_2(B)$	
$w_1(B)$		
		$w_3(B)$

IR ⇒ ✓

Updatedread ⇒ ✓

finalwrite ⇒ ✓

} So, it is View Equal.

But this schedule is not conflict equal.

Que ③

S_1 :

T_1	T_2	T_3
$w_1(A)$		
	$w_2(A)$	
		$w_3(A)$

S_2 :

T_1	T_2	T_3
	$w_2(A)$	
$w_1(A)$		
		$w_3(A)$

S_1, S_2 not conflict equal

But ✓

S_1, S_2 are view equal

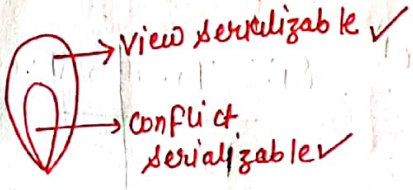
Conclusion :-

Conflict equal

S_1 S_2 conflict equal every R-W, W-R, W-W conflict pairs of S_1 S_2 must be same precedence.

View equal :-

S_1, S_2 View equal iff every IR, UR, FW of S_1, S_2 must be same.



So :-

1) If S_1, S_2 Conflict Equal schedules Then — S_1, S_2 View equal schedule also.

2) If S_1, S_2 not conflict equal schedules then — S_1, S_2 may or may not View equal.

Testing of Given schedule (S) is View serializable or not :-

(1) Checking about Final write :-

V.V.Imp

Data Item

Given schedule	View equal serial
X : T_i (T_j) → Final write	$T_i \rightarrow T_j$
Y : T_i T_j (T_k)	(T_i T_j) → T_k
Z : (T_i)	— any schedule (Koi schedule Aaaye Koi Fenk Nhi Pdega)
W : No transaction writes	

V.V.Imp

2). Checking about Initial read

Given Schedule			find view equal serial (S')
data item	Initial Read	Writes	
X	T_i°	T_j°	$\Rightarrow T_i^\circ \rightarrow T_j$
Y	T_i T_j	$T_J T_K$	$\Rightarrow T_i^\circ \rightarrow (T_J T_K)$ $T_J \rightarrow T_K$
Z	T_i°	No write of Z.	

V.V.Imp

3) Updated Read :-

Given Schedule	View equal schedule
$\Rightarrow W_i^\circ(X) \longrightarrow R_J(X)$ No other transaction writes data item X.	$\Rightarrow T_i^\circ \rightarrow T_J$
$\Rightarrow W_i^\circ(Y) \longrightarrow R_J(Y)$ Some other Transaction T_K also write Y	$\Rightarrow T_K \quad T_i^\circ \xrightarrow{(T_K)} T_J \quad T_K$ T_K° must before T_J & T_K° should not in between $T_i^\circ T_J$.

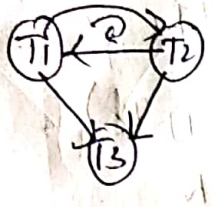
★★★

Que (1) S1: S: $r_2(A)\ w_1(A)\ w_2(A)\ w_3(A)$

(I) Test Conflict serializable.

(II) Test View Serializable.

Sol^n (I)



→ Not conflict serializable.

(II)

Given schedule	View equal serial
Final write ⇒ A: $T_1\ T_2\ T_3$	$(T_1\ T_2) \rightarrow T_3$
Initial read	$T_2 \rightarrow T_1\ T_3$

data Item	IR	W
A	T_2	$T_1\ T_2\ T_3$

$(T_2 : T_1 : T_3)$

Serial schedule view equal to (S)

S is view serializable schedule

Que S: $r_2(A)\ r_2(B)\ w_2(A)\ w_1(A)\ r_3(A)\ w_1(B)\ w_2(B)\ w_3(B)$

test schedule S View Serializable or NOT.

• Not conflict serializable. Then check.

Sol^n

given schedule	View equal serial
A: T_2 T_1	$T_2 \rightarrow T_1$
Final write B: $(T_1\ T_2\ T_3)$	$(T_1 ; T_2) \rightarrow T_3$
Initial Read	

data	IR	W
A	T_2	
B	T_2	

P.T.O

Soln S is Not conflict serializable.

For View serializable :-

Given schedule	View equal schedule
F.W A: T_2 (T_1) B: T_1 T_2 (T_3)	$\Rightarrow$ $T_2 \rightarrow T_1$ $\Rightarrow$ $(T_1 T_2) \rightarrow T_3$
R.R data | I.R | write A | T_2 | T_1 T_2 B | T_2 | T_1 T_2 T_3	$T_2 \rightarrow T_1$ $T_2 \rightarrow (T_1 T_3)$
U.R $W_1(A) \rightarrow R_3(A)$ Other Transaction writes (A) : T_2	$T_1 \rightarrow T_3$ T_1 must before T_3, & T_2 should not b/w T_1, T_3.

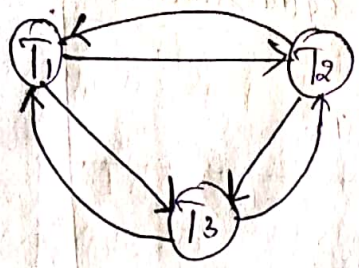
S' $(T_2 \rightarrow T_1 \rightarrow T_3)$ serial schedule

View ~~serializable~~ equal to schedule(S)

S : view serial schedule.

** Que S: $r_1(A)$ $r_2(A)$ $r_3(A)$ $w_1(A)$ $w_2(A)$ $w_3(A)$

Sol^n



→ Not conflict serial

" for View serial .

given data(schedule)	View equal serial
~~for~~	
[F.W]: A :- T1 T2 (T3)	(T1, T2) → T3 → T1 → T3
[I R]: data | I R | W A | T1, T2, T3 | T1, T2 T3	T1 → (T2 T3) T2 → (T1 T3) T3 → (T1 T2) ↳ T3 → T1

No serial schedule View equal to schedule (S) so,

S is Not View serializable schedule .

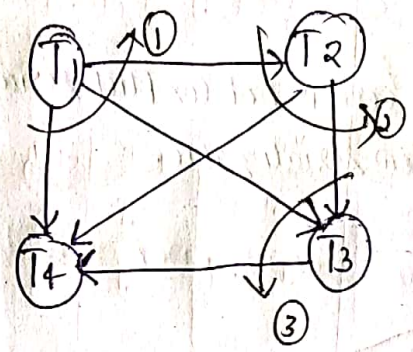
Que consider given schedule (s)

S: $r_1(A)\ r_2(A)\ r_3(A)\ r_4(A)\ w_1(B)\ w_2(B)\ w_3(B)\ w_4(B)\ w_1(C)\ w_3(C)\ r_4(C)$

(1) How many conflict equal serial schedule for s.

(2) How many view ~~seri~~ the equal serial schedule for s.

Soln (1)



$T_1 \to T_2 \to T_3 \to T_4$ → Ans (1)

↓

only one Topological order found

↓

only one serial schedule conflict equal to schedule "s".

(2)

Given schedule	~~Se~~ View equal serial schedule (s')
(1) (F.W) A: X ← No final write. B: $T_1 T_2 T_3 T_4$ C: $T_1 T_3$	$(T_1 T_2 T_3) \to (T_4)$ $(T_1) \to (T_3)$
(2) (I R) Data | I R | W A | $T_1 T_2 T_3 T_4$ | — B | — | — C | — | —	→ —
(3) U.R ① $w_3(C) \to R_4(C)$ other write(C) → T1	$T_3 \to T_4$

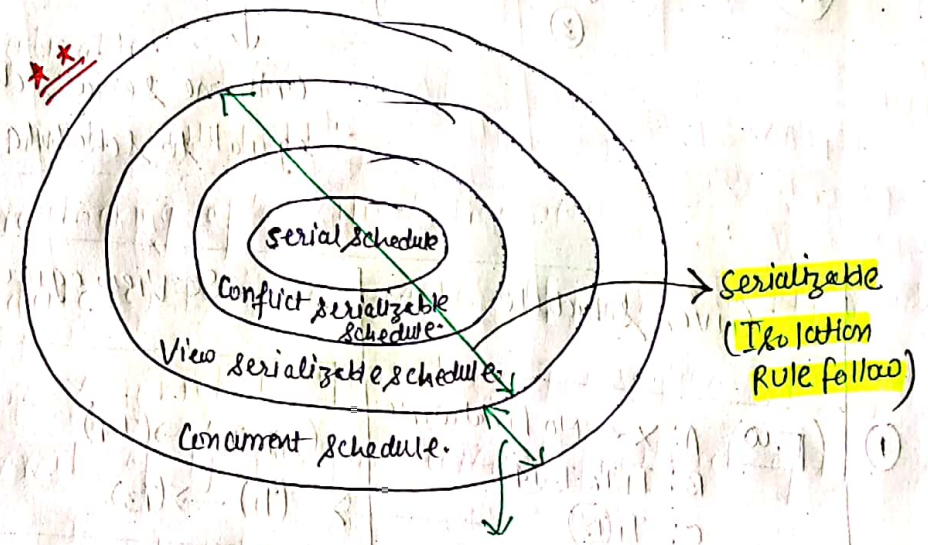
3 Serial order Ans (3)

$T_1\ T_3\ T_4$ (with T_2 below) → $T_2 T_1 T_3 T_4$ ✓, $T_1 T_2 T_3 T_4$ ✓, $T_1 T_3 T_2 T_4$ ✓

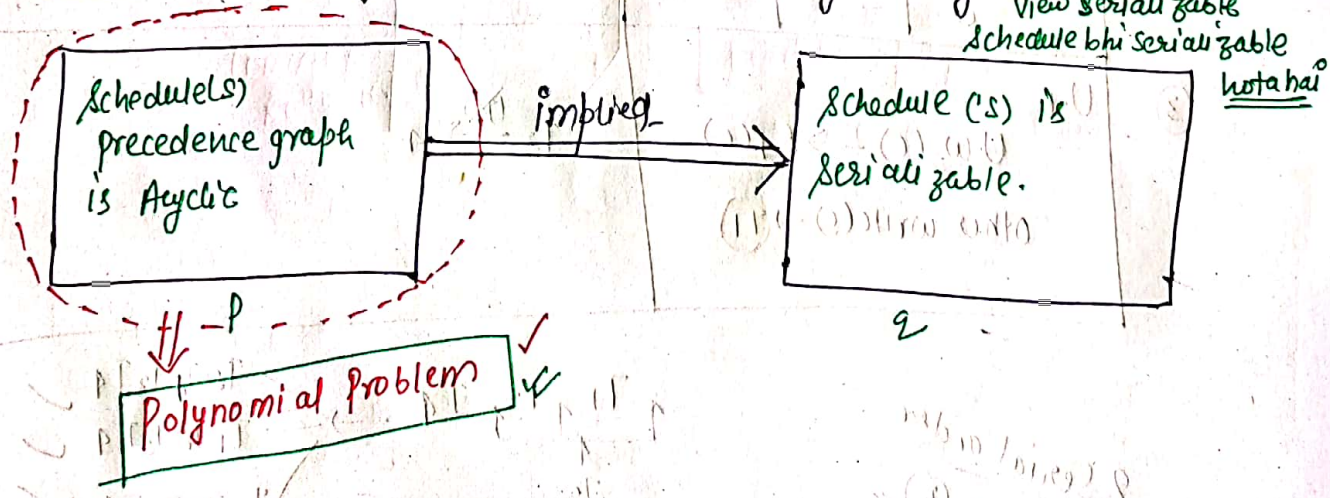
- Conflict equal serial order is one among the View equal serial order → It says conflict serializable is more strict than View serializable.

✓ If schedule(s) conflict serializable then schedule(s) is also View serializable.

✓ If Schedule (s) ~~may/may not view seri~~ is not conflict serializable Then schedule (s) may/may not view serializable.



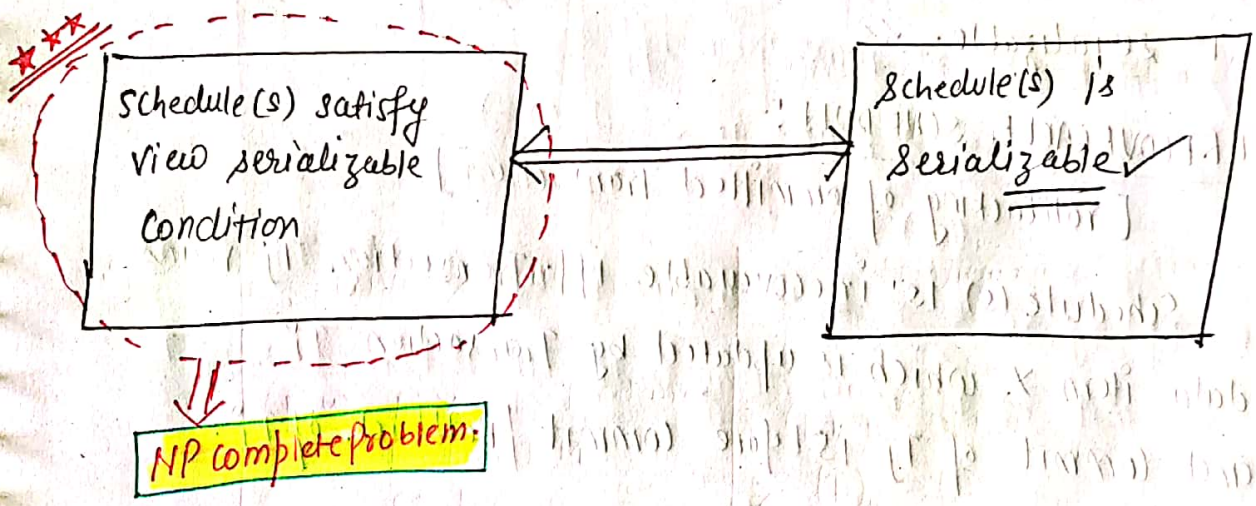
⇒ Conflict serializable schedule Testing is only sufficient but Not Necessary condition for serializability Testing. → Kyun ki only View serializable Schedule bhi serializable hota hai



{ only sufficient but Not Necessary } →
⟶ (Implied) (or) if then
↳ (P → Q)

{ sufficient & & Necessary } ↔
↔ P ↔ Q.
if & only if

View serializable schedule Testing conditions are both sufficient & Necessary cond^n for serializability Testing



- Testing of Conflict-serializable is P-problem
  [Polynomial TC]
  ↓
  TC = $O(n^2)$ ✓
- Testing of View serializable is NPC-problem
  [exponential TC]
  ↓
  TC = $O(2^n)$ ✓

Lecture-24:-

Classification of Schedule based on Recoverability:-

Concurrent Execution may leads

- Irrecoverable Problem
- Cascading Roll Back Problem
- Last update problem

→ Also causes Inconsistency can occurs even schedule is serializable.

IRRECOVERABLE SCHEDULE:-

[rollbacking of committed Transaction].

Schedule (S) is irrecoverable, iff Transaction T_j reads data item X which is updated by Transaction T_i and commit of T_j is before commit / Rollback of T_i.

(S)

T_i	T_j
W(x)	
	R(x)
	commit
Commit / Rollback	

→ Irrecoverable Schedule.

K/a dirty read

[Reading of T_j for same Variable without committing after writting on T_i of Transaction]

Ex :- 'A' balance of customer

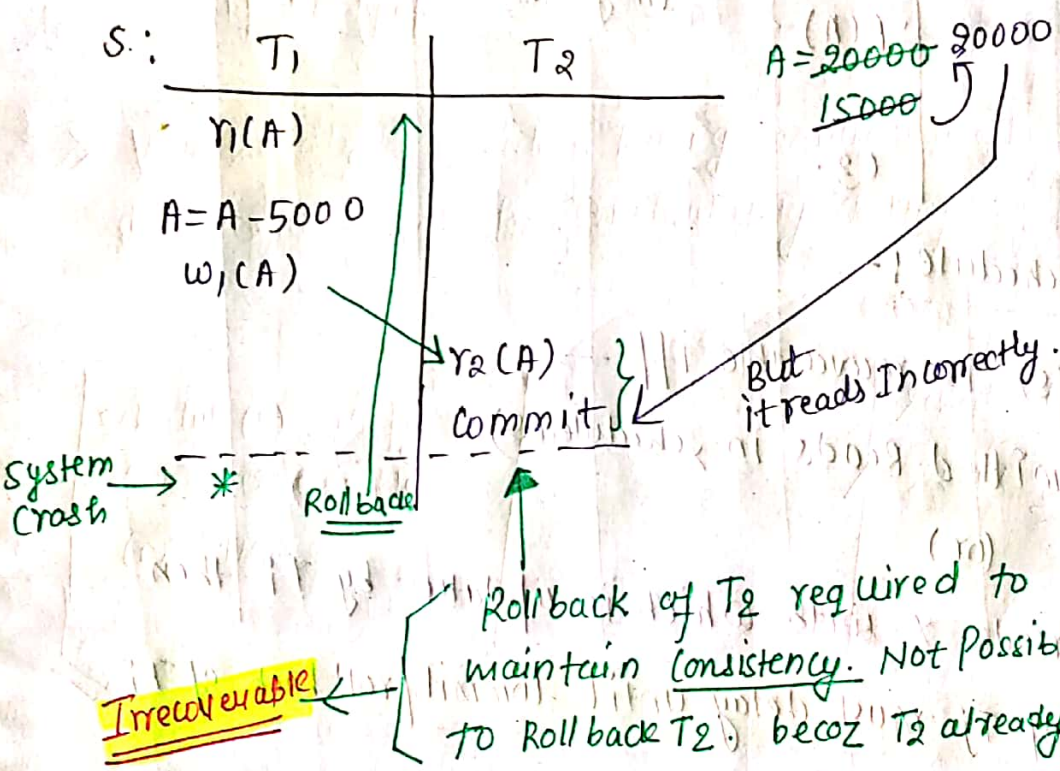
T_1 : Withdraw 5000 from "A"

$[r_1(A), A = A - 5000, w_1(A), C_1]$

T_2 : Check Bal of "A"

$[r_2(A), C_2]$

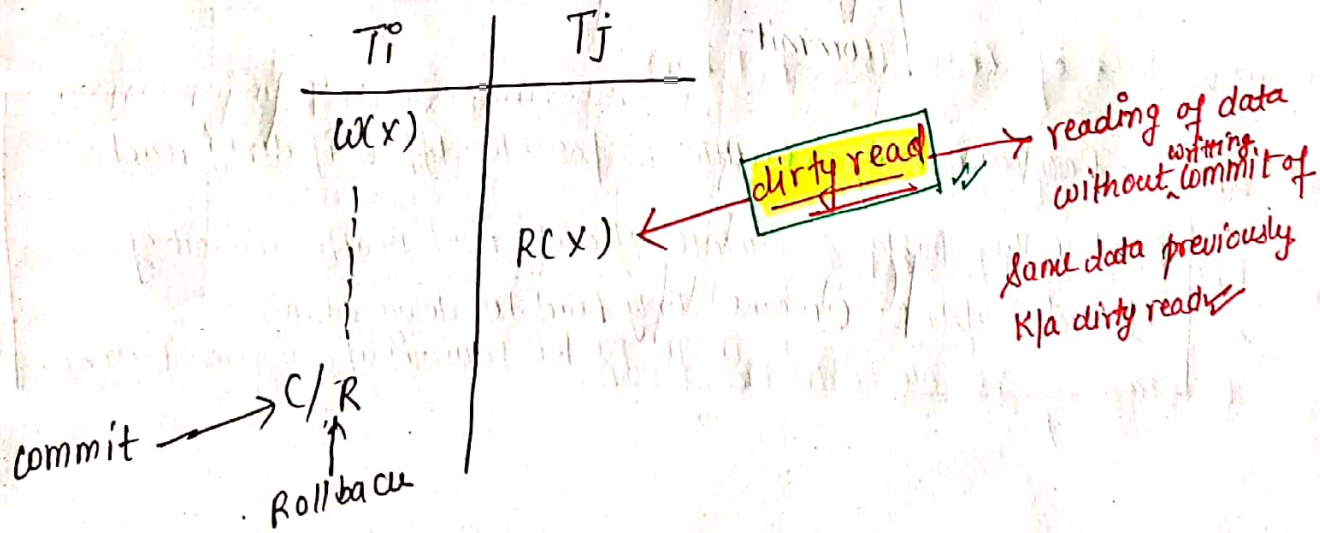
S :



Irrecoverable ← Rollback of T_2 required to maintain Consistency. Not Possible to Rollback T_2. becoz T_2 already committed

÷ Uncommitted Read :- (Dirty read)

Transaction T_j reads X which is updated by uncommitted Transaction T_i.



S:

T_1	T_2	T_3
R(A)		
	W(B)	
	C2	
		R(B)
W(A)		
		R(A) ← dirty read
C1		
		C3.

Recoverable Schedule :-

Schedule (S) is recoverable iff —

① No uncommitted Reads in schedule (S)

(or)

② If T_j reads X ~~with~~ which is updated by T_i then commit of T_j must delay until commit/roll back of T_i

S:

T_i	T_j
W(X)	
	R(X) ← dirty read
Commit/Rollback	!
	!
	commit

- Recoverable of schedule is possible ~~iff~~ if dirty read exist in the Table & commit of dirty read is after commit of Previous update. (means dirty Read ka delay karo)
  Tb Tk Jb Tk ki Previous update commit Na ho

Que Find which one is Recoverable & which is not Recoverable.

1] S:

T_1	T_2
W(A)	
	R(A) ← dirty Read But can rollback
Power failure → *	
Commit	

⇓ Recoverable ✓

2] S:

T_1	T_2
R(A)	
	R(A)
W(A)	
	W(A)
	C2
C1	

↓ No dirty Read. Recoverable ✓

3]

T_1	T_2
R(A)	
W(A)	
	R(A) → dirty Read But can rollback
	R(B)
R(B)	
W(B)	
C1	
	C2

⇓ Recoverable

4]

T_1	T_2
W(A)	
	W(A)
	W(B)
W(B)	
	C2
C1	

⇓ No dirty Read ✓

⇓ Recoverable ✓

5]

T_1	T_2	T_3
R(A)		
	R(B)	
	W(A)	
W(A)		
		R(C)
R(C)		
C1		
		R(A)
	W(B)	
	C2	
		R(B)
		C3

↓ No dirty Read → Recoverable ✓

6]

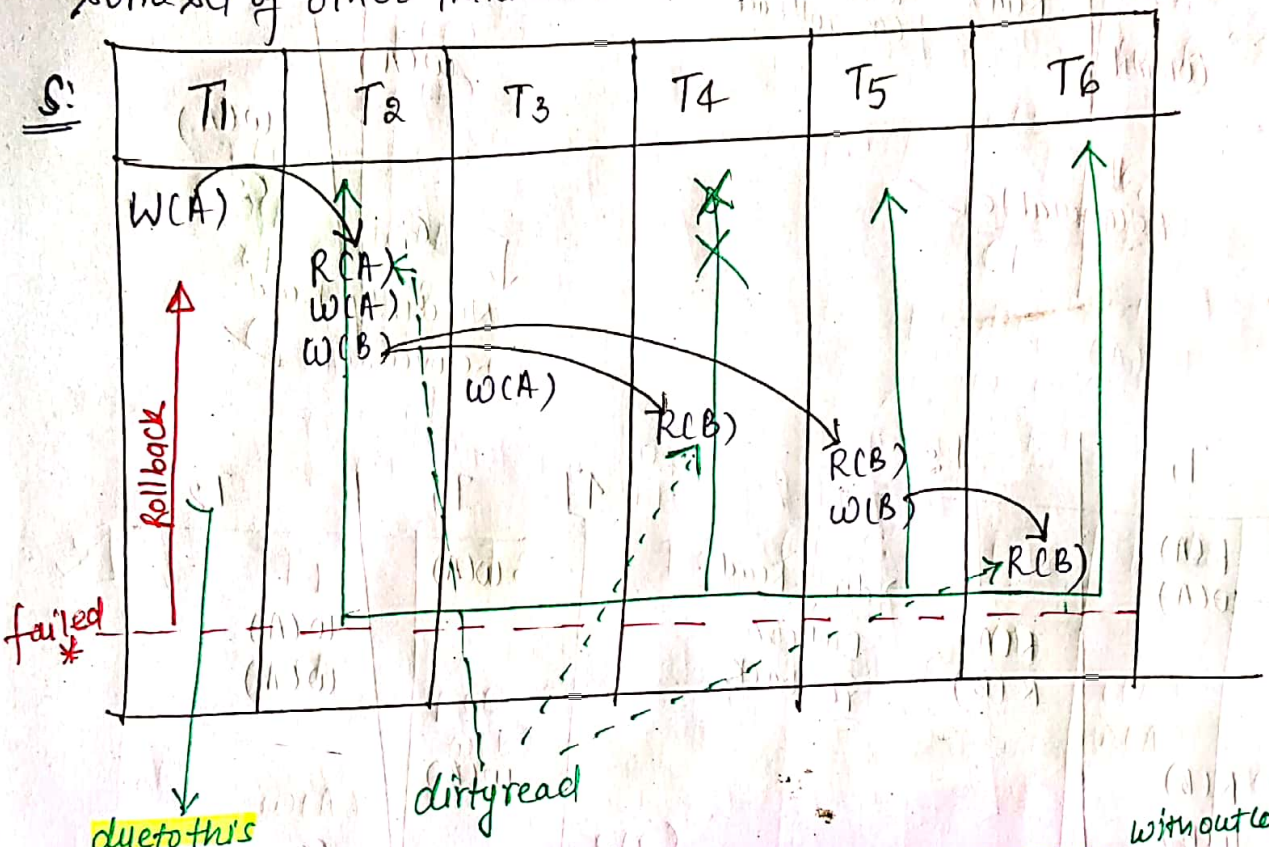
T_1	T_2	T_3
r(A)		
	r(A)	
	w(A)	
		r(A) → dirty read
	w(B)	
R(B) ← dirty read		
		C3
	C2	
C1		

⇐ Not recoverable

Cascading Rollback Problem :-

Bcoz of failure of some Transaction forced to roll back some set of other Transaction in schedule (S)

S:



due to this roll back all dirty read will also roll back (due to connected to each others) without commit

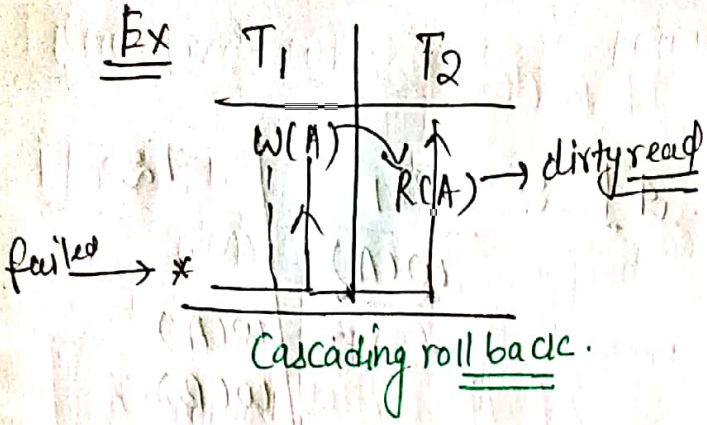
This is k/a cascading Rollback.

because of T_1 failure forced to RB (rollback) T_2, T_4, T_5 & T_6 cascading rollback

Dis advantage of Cascading Rollback :-

1. Wastage of C.P.U Time
2. Wastage of I/O Cost
3. Wastage of resource utilizasation.

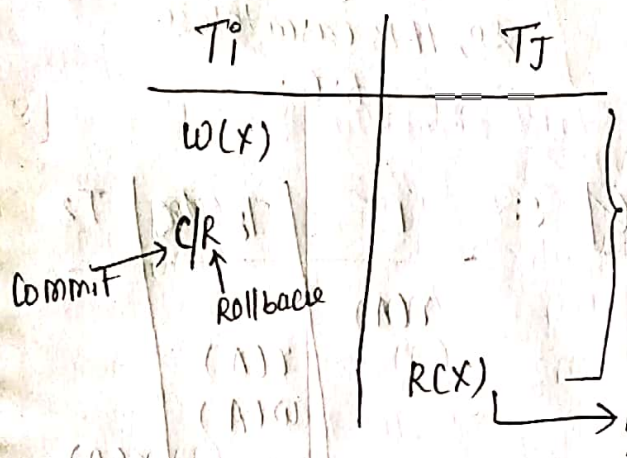
Ex



Cascading roll back.

Cascading Roll back Schedule:- (Avoid Cascade Aborts):-

Dirty read Not allowed.

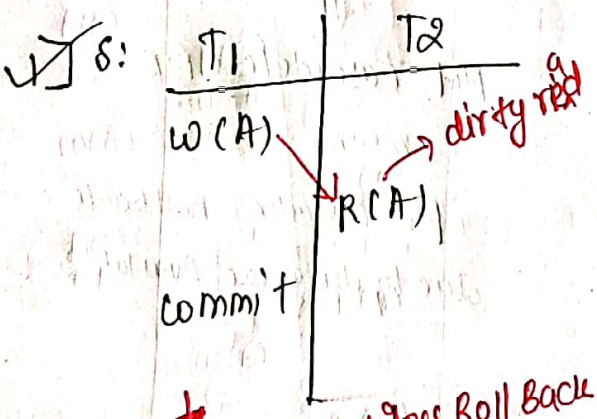


If Ti writes X & & Tj required to Read X which is updated by Ti then—

Read X of Tj must delay untill commit (or) Roll back of Ti

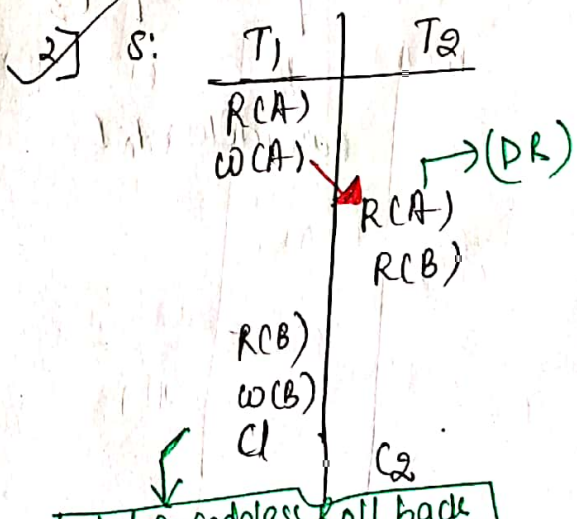
Questions Find which one is Cascadeless Roll back & which one Not.

1) S:



Not Cascadeless Roll Back

2) S:



Not Cascadeless Roll back

3] S:

T_1	T_2
R(A)	
~~W(A)~~	
	R(A)
	~~R(B)~~
~~R(B)~~	
W(A)	
~~C~~	
	W(A)
	C2
C1	

⇓

Cascadeless R·B

4] S:

T_1	T_2
W(A)	
	W(A)
	W(B)
W(B)	
	C2
C1	

No, dirtyread
so, it's cascadeless R·B.

5] S:

T_1	T_2	T_3
R(A)		
	R(B)	
	W(B)	
W(A)		
		R(C)
R(C)		
C1		
	W(B)	
	C2	
		R(A)
		R(B)
		C3

⇓

Cascadeless R·B

6] S:

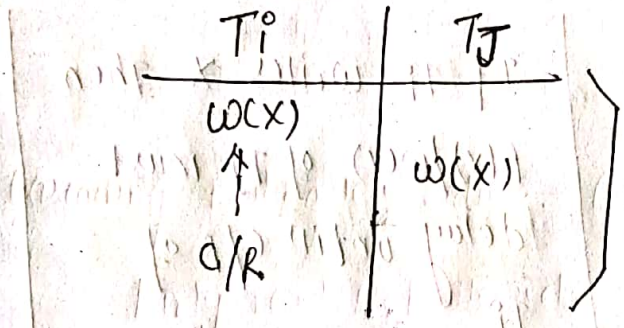
T_1	T_2 ~~T2~~	T_3
r(A)		
	r(A)	
	W(A)	
		r(A)
	W(B)	
R(B)		
		C3
	C2	
C1		

⇓

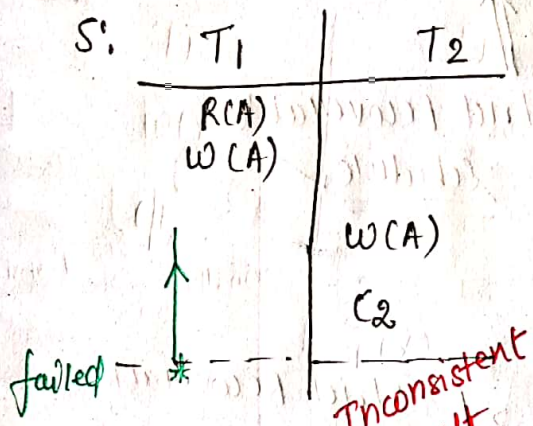
Not cascadeless R·B.
means
it is cascading Rollback due to dirtyread Available ✓

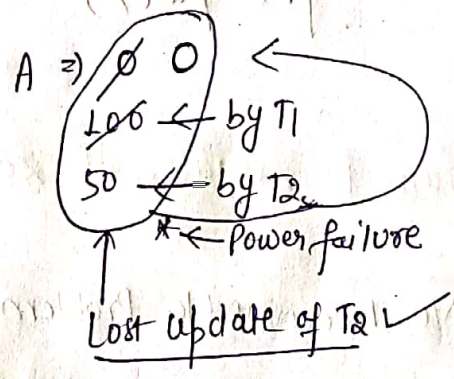
Lost update problem :-

Transaction T_J writes x which is already written by Uncommitted Transaction T_i.



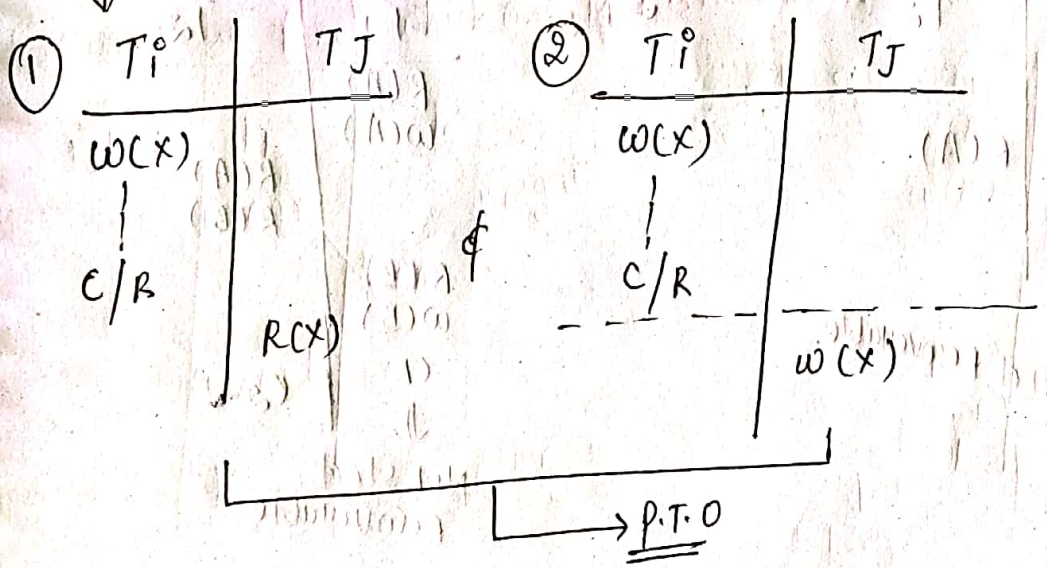
S:





Strict Recoverable schedule :-

Cascadeless Roll Back (& &) No lost update problem.



P.T.O

from previous Point ① & ② It is clear that —
For strict Recoverable Schedule.

T_i	T_j
W(X)	
|	
C/R	
	R(X)/W(X)

If T_i writes X Then R(X)/W(X) of T_j must delay until C/R of T_i

Strict Recoverable Schedule ✓

① No Irrecoverable Problem
② No Cascading Rollback
③ No Lost update problem.

→ For strict Recoverable Schedule

Que :- Find which one is strict Recoverable & which one Not

1] S:

T_1	T_2
W(A)	
	R(A)
Commit	

⇓ Not strict Recoverable.

2] S:

T_1	T_2
R(A)	
W(A)	
	R(A)
	R(B)
R(B)	
W(B)	
C	
	C_2

⇓ Not strict Recoverable

3] S:

T_1	T_2
R(A)	
	R(A)
W(A)	
	W
	W(A)
	C2
C1	

⇓ Notstrict recoverable

4] S:

T_1	T_2
W(A)	
	W(A)
	W(B)
W(B)	
	C2
C1	

⇓ Notstrict Recoverable

5] S:

T_1	T_2	T_3
R(A)		
	R(B)	
	W(A)	
W(A)		
		R(C)
R(C)		
C1		
		R(A)
	W(B)	
	C2	
		R(B)
		C3

⇓ Not strict Recoverable

6] S:

T_1	T_2	T_3
r(A)		
	r(A)	
	W(A)	
		r(A)
	W(B)	
r(B)		
		C3
	C2	
C1		

⇓ Not strict Recoverable

Example of strict Recoverable:-

Sol:-

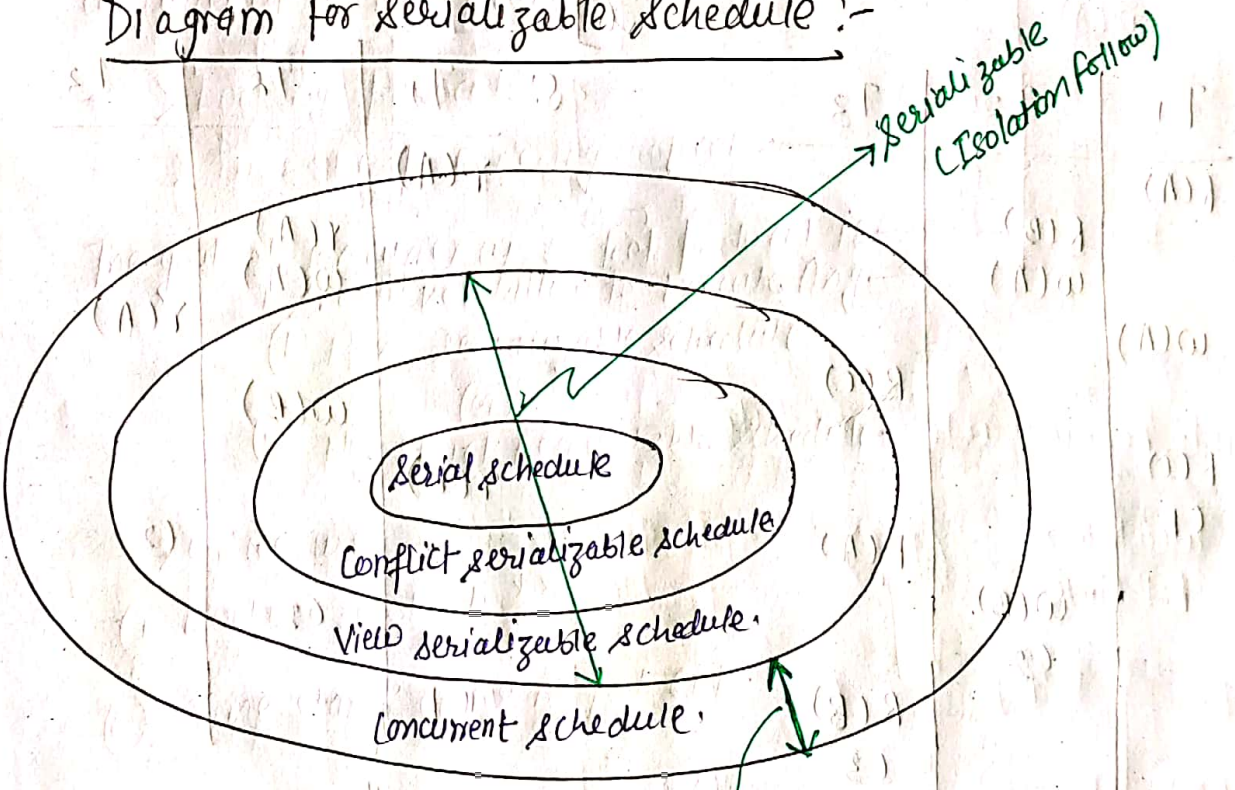
T_1	T_2
R(A)	
	R(A)
W(A)	
C1	
	W(A)
	C2

→ Strict Recoverable

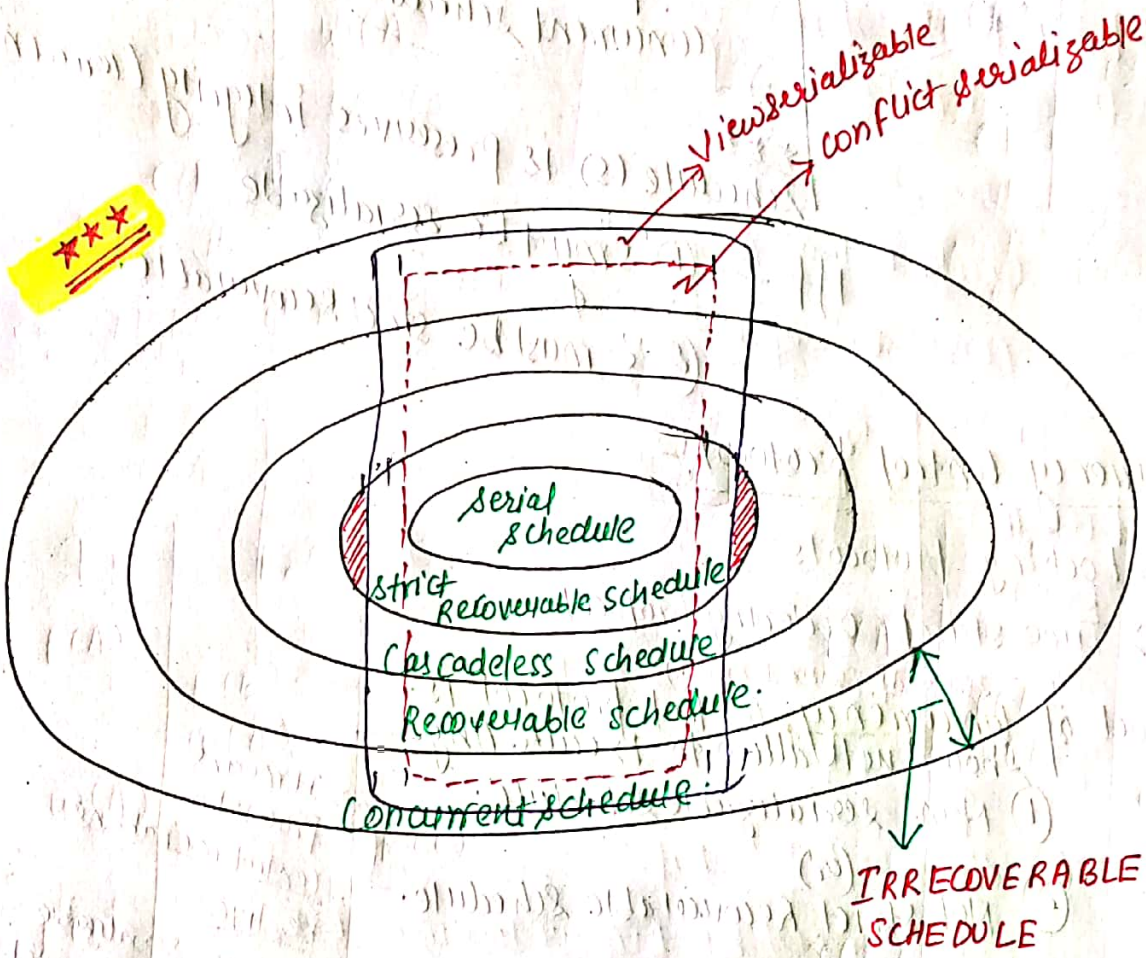
W(A) ----------→ No Lost update problem.

Hence

Diagram for serializable schedule :-



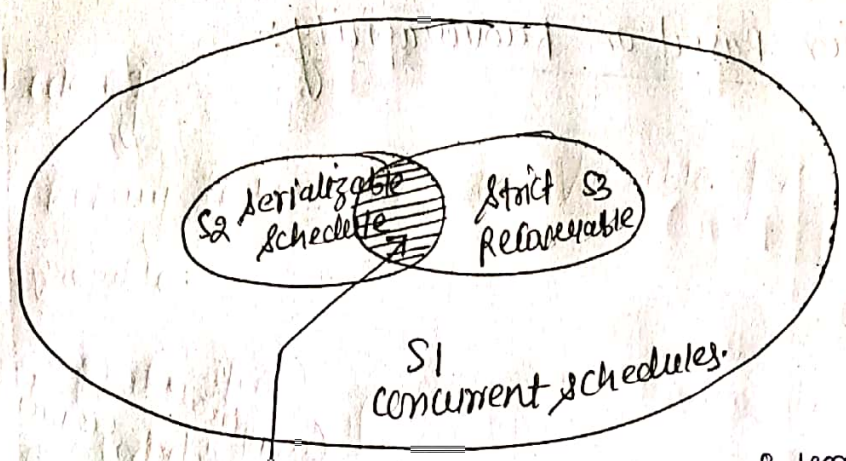
Classification Based on Recoverability :-



- Not Every strict Recoverable schedules are conflict serializable/ view serializable.

&

- Not Every serializable schedule are
  - Recoverable
  - Cascadeless schedule
  - strict Recoverable schedule.



***
Schedule (S) is preserves integrity (correctness)
iff — ① "S" must be serializable
&
② 'S' must be strict Recoverable.

Concurrency Control protocol :-

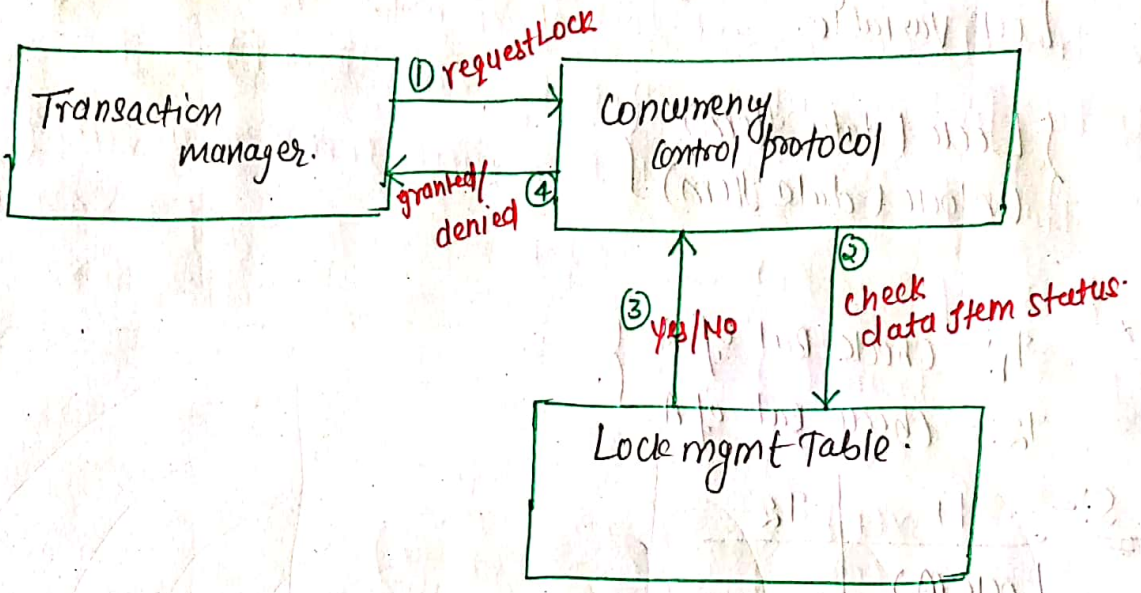
→ Locking protocols.
→ Time stamp ordering protocol

The goal of concurrency control protocol (समरूपता, संगमन (mating)) is —
should not allow to Execute Any —
① Non-serializable schedule.
(or)
② Not strict Recoverable Schedule.
} → It means Process ko control me Rakhne ke liye
① Always serializable schedule process execute kro
② strict Recoverable use kro

So, we use —
① Locking Protocols **
② Time stamp ordering protocols.

Locking Protocols:-

Lock:- It is a variable which is used to identify status of data ~~item~~ item



Ex Trans(T)

Lock(A) ⟸ granted by concurrency control protocol (CCP)

R(A)

W(A)

Lock(B) ⟸ if say it is denied by CCP

| wait ⟵ Then wait

Lock(B) ⟸ granted by CCP

W(B)

Unlock(A)

Unlock(B)

Binary Locking :-

In Binary locking only one Variable provide for Lock Variable.

{ Lock (data Item)
Unlock (data Item) }

Ex T_1 :- Check Bal of A }.
T_2 :- Check Bal of A

S :-

T_1	T_2
Lock (A)	
R(A)	
|	
	Lock(A) ⟸ denied
	R(A)

"S" is Correctly scheduled But
"S" may not allowed by Binary locking Bcoz —
Binary locking Unable to differenciate Read only data & Read write data.

(dis-adv of Binary Locking)

⇒ Binary locking system having → Less degree of Concurrency

Then S/X Locking system come in Existance. called multi mode Locking.

S ← only for Read locking
X ← for Read & write Both locking

T_1	T_2
S(A)	
R(A)	
	S(A) ⟸ grant
	R(A)

Adv of S/X locking is More degree of Concurrency.

Multimode Locking

Shared Lock(S) : Read only lock.
Exclusive lock(X) : Read-write lock.

Trans(T)

S(A) ← grant
Read(A)
X(B) ← grant
Read(B)
write(B)
Unlock(A)
Unlock(B)

T_1	T_2
S(A)	
R(A)	
	X(A) ← denied

:- Lock compatable Table :-

data item	S	X
S	✓	X
X	X	X

Req. by Tj →

Hold by Ti

** Two phase locking Protocol (2PL) { Growing, shrinking } ✓

[Guarantees Serializability] ✓

⇒ Trans(T) can request lock of any data item in any mode until first unlock op^n done by Transaction (T_i)

⇒ Trans(T) not allowed to request locks in unlocking phase of Trans(T)

{ Phle Poora locking & then unlocking }

L → | UL →

Ex

Trans(T)

S(A)
X(B)
S(C)
X(D) *
U(B) *
U(D)
U(A)
U(C)

→ Locking phase (Growing phase)

→ Lock Point of Trans(T)

→ Unlocking phase (shrinking phase)

first unlock ↑

Ex:- $(T_1 \to T_2)$ – Conflict Pair can execute by 2PL only if Lock Point of T_1 before lock Point of T_2

T_1	T_2
S(A)	
⋮	
R(A)	
* U(A)	
	X(A)
	W(A)
	⋮
	U(A) * → Lock Point T_2

Lock Point T_1 ←

** If $T_1 \to T_2$ Conflict Pair executed by 2PL Then T_1 lock Point before T_2 lock Point.

***

⇒ If schedule (S) is not conflict serializable schedule (cyclic precedence graph) then schedule (S) not allowed to execute by 2 PL.

S: $r_1(A)\ w_2(A)\ w_2(B)\ r_1(B)$

→ Not conflict serializable schedule.

So, This is Not allow for 2PL Bcoz

If 2PL locking occure then schedule must be serializable
- conflict ✓
- & viewseriali zable ✓

ex

S:	T_1	T_2
	S(A)	
	$r_1(A)$	
	S(B)	
	U(A)	
		X(A)
		$w_2(A)$
		X(B)
		$w_2(B)$
	$r_1(B)$	
	U(B)	

Here S(B) is locked ✓ → Bcoz → X(B) denied.

Locks occure Example

T_1: Transfer 500 from A→B	T_2 display A+B	T_3 set A,B,C as 10000
T_1	S: T_2	T_3
X(A)	S(A)	X(A)
R(A)	R(A)	W(A)
W(A)	S(B)	X(B)
X(B) — Grow → Locking cond^n	— Grow → Locking cond^n	W(B)
		X(C) — Grow → Locking cond^n
U(A)	U(A)	U(A)
R(B)	R(B)	U(B)
W(B)	U(B) — shrink → unlocking cond^n	W(C)
U(B) — shrink → unlocking cond^n	commit	U(C) — shrink → unlocking cond^n
commit	(2PL)	commit
(2PL)		(2PL)

All previous $T_1, T_2, \& T_3$ Transaction are ——

Concurrent execution of T_1, T_2, T_3 & guaranteed serializable ✓.

Que 2PL meant for

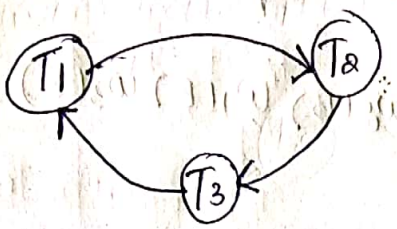
(A) Atomicity.
(B) Consistency
(C) Isolation
(D) durability.

Ans (C) Isolation ✓

:- Lecture -25 :-

→ If schedule (S) is not conflict serializable schedule (cycle will be in Precedence Graph) Then Not allowed to execute by 2PL.

Ex $r_1(A)\ w_2(A)\ w_2(B)\ r_3(B)\ r_3(C)\ w_1(C)$



→ Not conflict serializable schedule :

So ∴ 2PL Not allowed

Que Check for 2PL

T_1	T_2	T_3
$r_1(A)$		
	$w_2(A)$	
	$w_2(B)$	
		$r_3(B)$
		$r_3(C)$
$w_1(C)$		

*** Not conflict serializable (or) Not view serializable.

↓

NOT 2PL used ✓

Sol^n

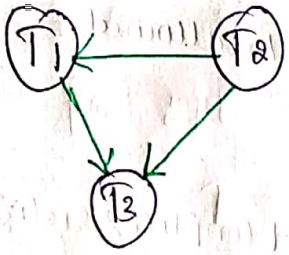
T_1	T_2	T_3
S(A)		
$r_1(A)$		
X(C) U(A)	X(A) X(B)	
	$w_2(A)$	
	$w_2(B)$	
	U(A) U(B)	
		S(B)
		$r_3(B)$ ← S(C) denied ✓
		$r_3(C)$
$w_1(C)$		

Not allowed to execute by 2PL

- If schedule (S) allowed to execute by 2PL.
Then schedule (S) also in conflict serializable.
& equal serial schedule is based on lock point order.

[LockPoint order = Topological order]

Q: S: $r_1(A)$ $r_3(A)$ $r_2(A)$ $w_2(B)$ $w_1(B)$ $w_3(B)$



→ Conflict serializable schedule.
& serial schedule ⇒ T2 : T1 : T3.
(Topological order.)

for checking 2PL:-

***

S:

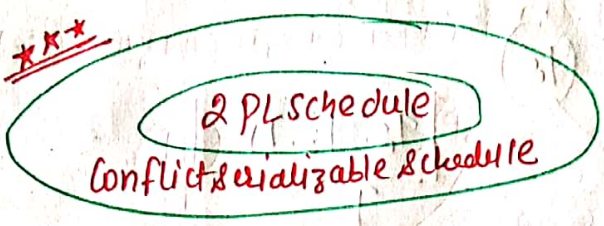
T_1	T_2	T_3
S(A)		
$r_1(A)$		
		S(A)
		$r_3(A)$
	S(A)	
	$r_2(A)$	
	X(B)	
	$w_2(B)$	
X(B)	U(B) U(A) → First lock Point	
$w_1(B)$		
U(B)		
		X(B)
		$w_3(B)$ → Third lock Point
		U(B)
		U(A)

Second lock Point

T2 → T1 → T3 → Lock Point order

- Every schedule which allowed by 2PL is also conflict serializable schedule but Not every conflict serializable schedule is allowed to execute 2PL

***



Que S: $W_2(A)\ W_3(A)\ W_1(A)\ W_2(B)\ W_3(B)\ W_1(B)$

Which is True?

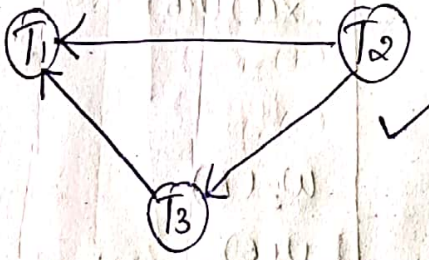
a) S is CSS && allowed by 2PL.

b) S is CSS && not allowed by 2PL

c) S is Not CSS && Not allowed by 2PL

d) S is Not CSS & allowed by 2PL.

Soln



CSS Allowed ✓

for 2PL

T_1	T_2	T_3
	X(A)	
	$W_2(A)$	
	X(B) U(A)	
—	—	X(A) ~~X(A)~~
		$W_3(A)$
		X(B) U(A)
$W_1(A)$		
	$W_2(B)$	
		$W_3(B)$
$W_1(B)$		

Not in 2PL

denied ✓

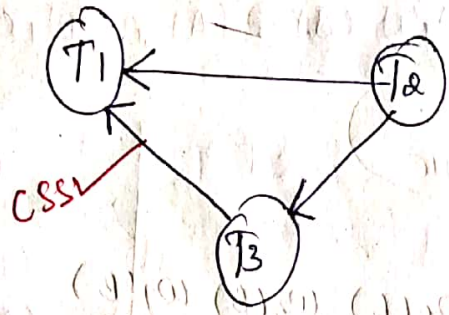
Ans (b)

CSS ✓
2PL ✗

Que S: $w_2(A)\ w_3(A)\ w_2(B)\ w_1(A)\ w_3(B)\ w_1(B)$

Check for CSS & 2PL

Sol^n



CSS ✓

$T_2 \to T_3 \to T_1$

↑ Topological order

For 2PL

T_1	T_2	T_3
	X(A)	
	$w_2(A)$	
	X(B) U(A)	X(A)
		$w_3(A)$
	U(B) $w_2(B)$	X(B) U(A)
X(A)		
$w_1(A)$		
		$w_3(B)$
X(B) U(A)		U(B)
$w_1(B)$		
U(B)		

2PL ✓

$T_2 \to T_3 \to T_1$

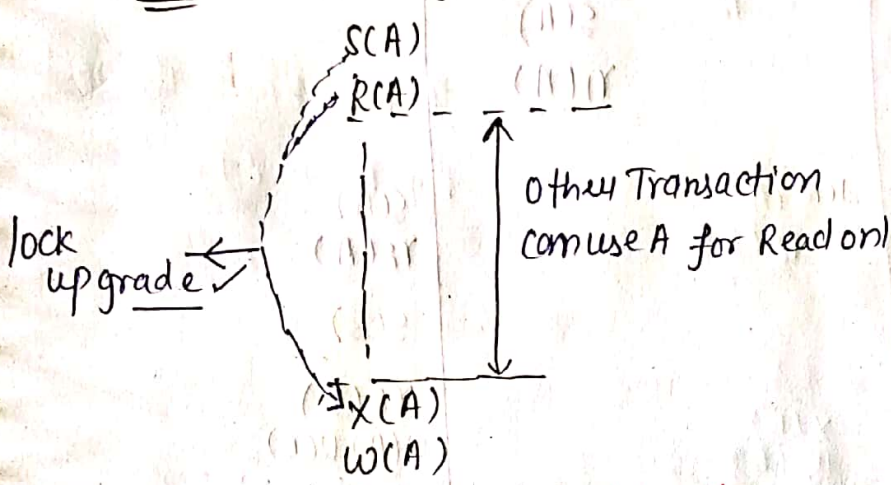
So CSS ✓
2PL ✓

:- Lock upgrading in 2PL :-

If Trans(T) required R(A) and W(A). Trans should Lock "A" in shared lock of "A" & perform R(A) before W(A) should upgrade Lock to exclusive mode by requesting X(A)

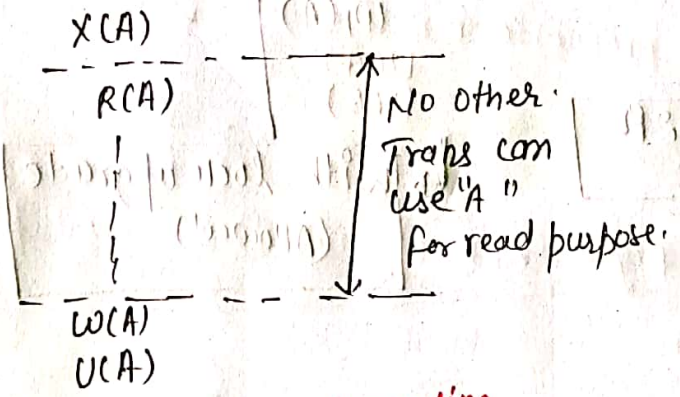
[Lock upgrading should be in Locking phase].

Ex :- Trans (T)



2PL with Lock upgrading.

Ex : Trans (T)



Without Lock upgrading

Adv of Lock up-grade 2PL is more degree of Concurrency

→ 2PL Condition by default consider Lock upgrading

Que S: $r_1(A)\ r_2(A)\ r_2(B)\ w_1(B)\ w_1(A)$

Soln

T_1	T_2
X(A)	
$r_1(A)$	
	S(A) ← denied
	$r_2(A)$
	$r_2(B)$
$w_1(B)$	
$w_1(A)$	

Without lock upgrade-2PL

T_1	T_2
S(A)	
$r_1(A)$	
	S(A)
	$r_2(A)$
	S(B)
	$r_2(B)$
	U(A), U(B)
X(B)	
$w_1(B)$	
X(A)	
$w_1(A)$	
U(A)	
U(B)	

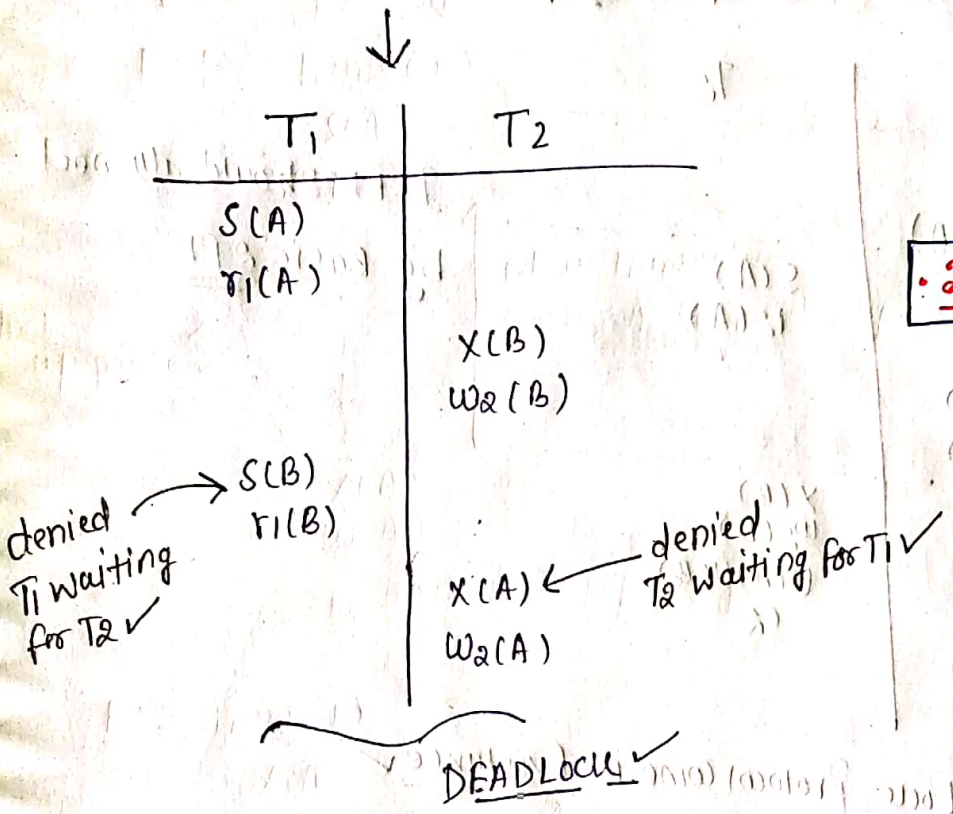
With lock upgrade (Allowed)

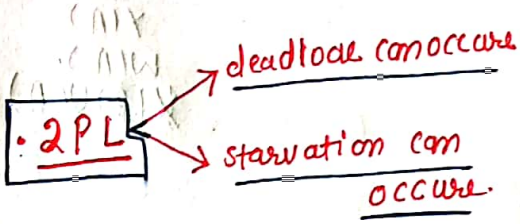
Limitation of 2PL:-

1] 2PL restriction may form **deadlock**.

Ex T_1 : $r_1(A)\ r_1(B)$ [S(A) S(B) U(A) U(B)] ✓

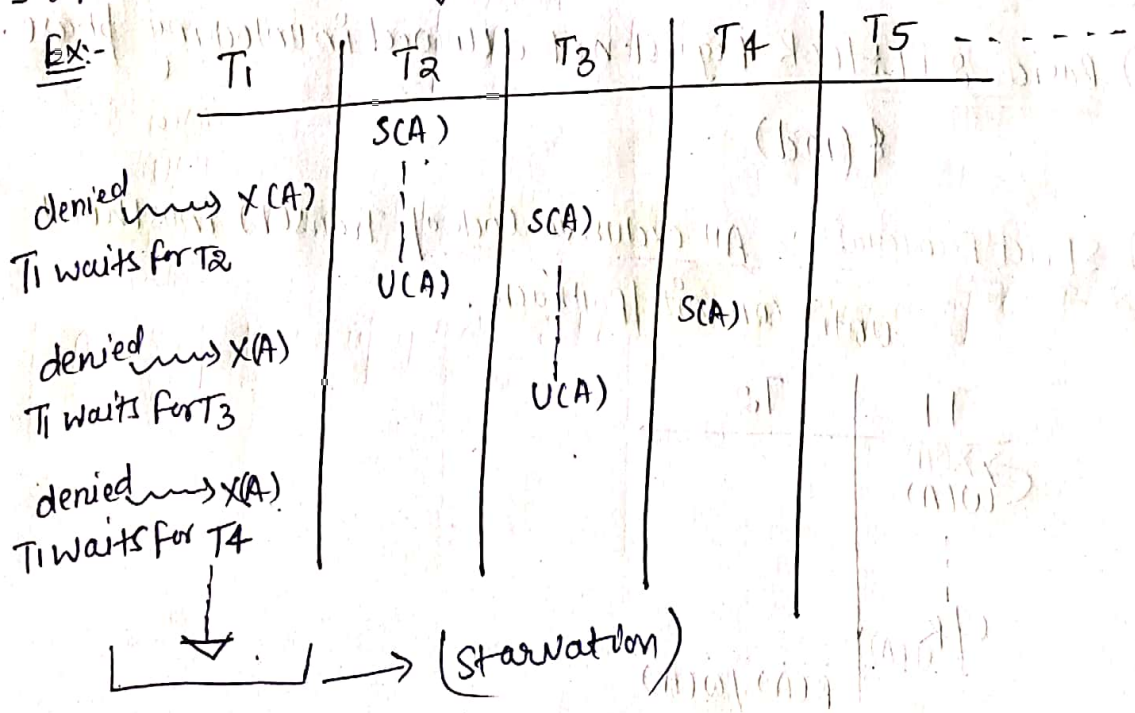
T_2 : $w_2(B)\ w_2(A)$ [X(B) X(A) U(B) U(A)] ✓





2] 2PL restriction may leads **starvation**.

Ex:-



3] 2PL restriction not sufficient for avoiding —

(a) IRREC Problem
(b) Cascading Rollback Problem
(c) Lost update problem

S: $W_1(A)\ r_2(A)\ W_1(B)\ W_2(B)\ C_2\ C_1$

S:

T_1	T_2
X(A)	
W(A)	
X(B) U(A)	
	S(A)
	r2(A)
w1(B)	
U(B)	
	X(B)
	w2(B)
	U(A) U(B)
	C2
C1	

IRREC should allowed by Basic 2PL

Then strict 2PL Locke Protocol come in Existance ✓

Strict 2PL :- Basic 2PL + strict recoverable //

① Basic 2PL :- lock Request not allowed in unlocking phase.

$\$(and)$

② Strict Recoverable:- All exclusive lock of Trans(T) must hold until commit / Rollback.

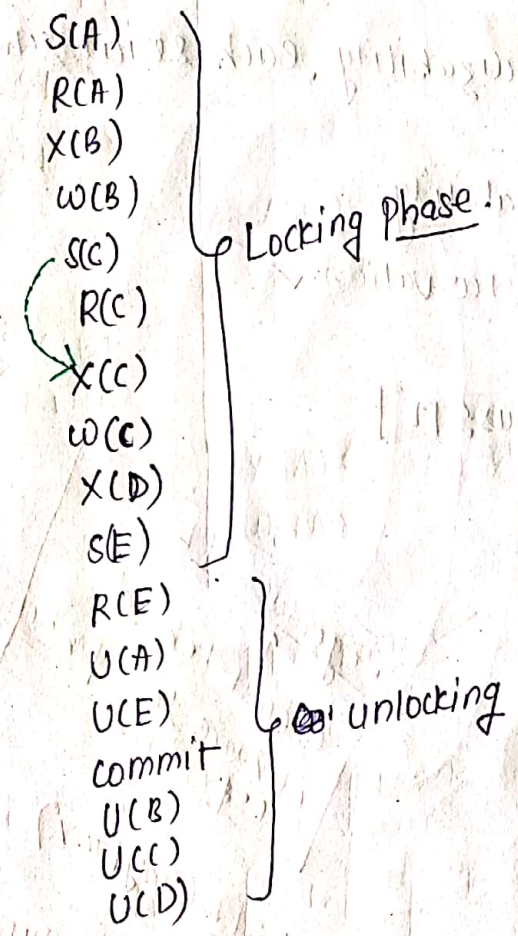
T_1	T_2
X(A)	
W(A)	
⋮	
C / $R_U(A)$	
	$R(A)$ / $W(A)$

Strict 2PL :-

Guranteed For

1. Conflict serializable [Equal serial based on lock points order].
2. Guranteed strict Recoverable
3. may possible deadlock && starvation.

Trans(T) [S/X + strict 2PL]



Regerous - 2PL :-

① Basic 2PL : Lock request not allowed in unlocking phase.

&&

② All locks of Trans(T) {shared / exclusive lock} must hold until Commit / Rollback.

Regerous - 2 PL protocol :-

① Guranteed for Conflict Serializability, each serial schedule is order of commit

② Guranteed for strict Recoverability.

③ Not free from dead lock & starvation.

Trans(T) [S/X + Regerous 2PL]

S(A)
R(A)
X(A)
W(A)
S(B)
R(B)
X(C)
W(C)
S(D)
R(D)
commit ← Lock Point
U(A)
U(B)
U(C)
U(D)

Locking phase

Unlocking phase

Regerous - 2PL
↑
commit kebaad
hi sare lock open krp.
(S&X) both

Strict 2PL (only for X)
↓
Regerous 2PL (only For S&X)
↓
Conservative 2PL (

Conservative 2PL protocol:-

{deadlock free}.

- Transaction should check availability of all data Item, if available then Lock occure, otherwise wait till all data item are Not available -ie

Transaction must check availability of all data items.

If all data items available Then —

Transaction should lock all data items before R/W opn.

Otherwise:-

Transaction should wait && Re-check.

Disadvantage:-

(1) less degree of Concurrency
(2) Starvation can occure
(3) Not strict Recoverable { It can be done & strict Recoverable }.

Ex Trans

```
S(A)
X(B)
S(C)
X(D)
R(A)
U(A)
W(B)
U(B)
R(C)
U(C)
W(D)
U(D)
Commit
```

Wasting of Resource.