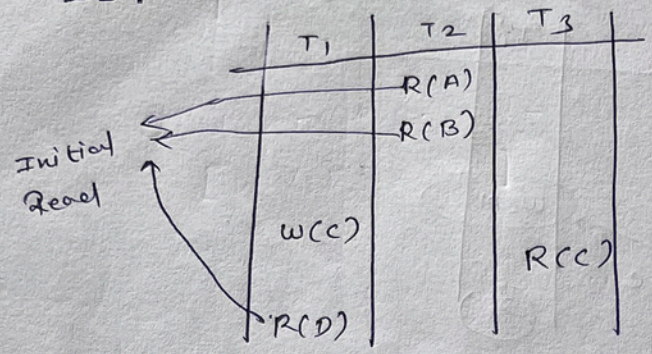
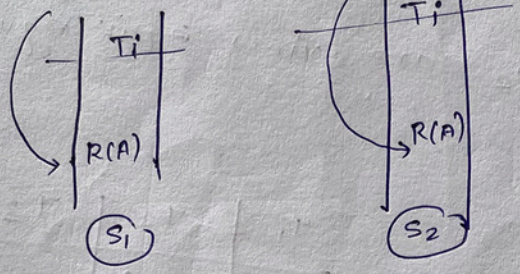
⇒ S_1, S_2 (view) Equal iff ⇒

(1) If some Transaction T_i Reading "A" from initial DB in S_1. Then in S_2 also T_i should read A from initial DB.

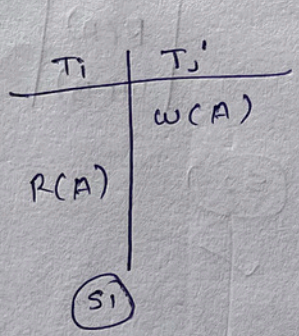
[Our initial read of S_1 & S_2 should be same]

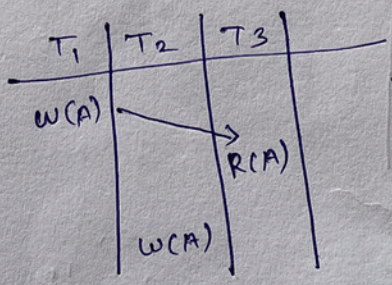


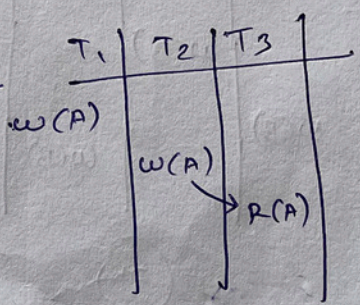
For exam



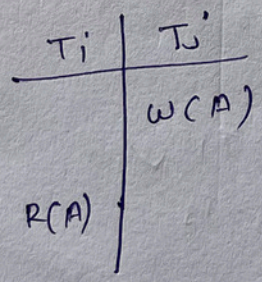
and (2) If Transaction T_i Reads "A" which is updated by T_j in S_i they in S_2 also T_j should read "A" updated by T_j only. [write read sequences should be same in S_1 & S_2]



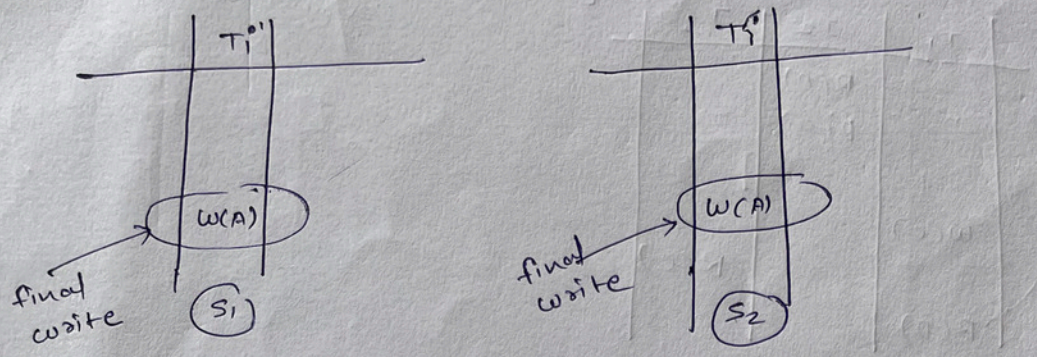




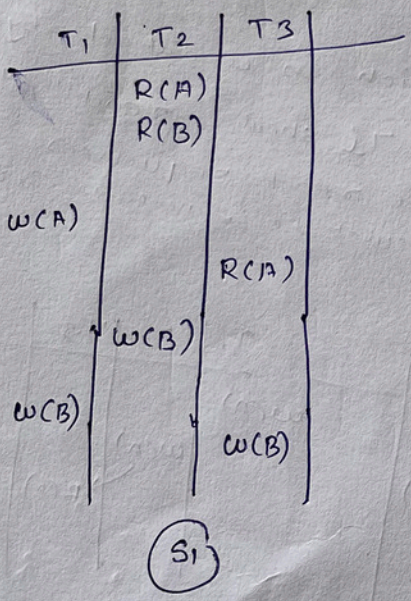
$[S_1 \neq S_2]$

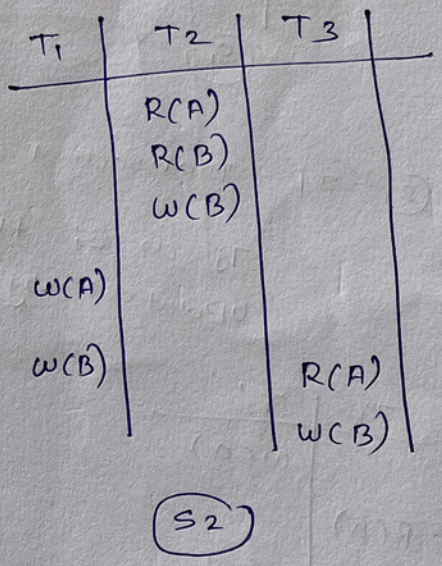


and (3) If Transaction T_i updates "A" finally in S_1 then in S_2 also ("A") should update by T_i finally.



⇒ Test given S_1 and S_2 are view ~~equi~~ equal or not?





	S_1	S_2	
Initial read	A : T_2, B = T_2	A : T_2, B = T_2	✓
W-R sequence	$T_1 \rightarrow T_3$	$T_1 \rightarrow T_3$	✓
last update / final write	A : T_1, B = T_3	A : T_1, B = T_3	✓

So S_1 and S_2 are view equal schedule.

Example:

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

S_1

Initial read:- A : T_3

W - A : $T_2 \rightarrow T_3$

final write :- A : T_2

B : T_3

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

S_2

A : T_3

$T_2 \rightarrow T_3$

A : T_1 ✗

B : T_3 ✓

So S_1 and S_2 are not view equal schedule

* Serializable Schedule :- Schedule (S) is serializable if and only if there exist some serial schedule (S') which is view equal to given schedule.

S [given schedule] (=) S' [Any one serial schedule]

view equal

→ Then S is serializable schedule.

Solve :-

Ex:

T_1	T_2
$R_1(A)$	
	$R_2(A)$
	$W_2(A)$
$W_1(A)$	

(S)

Test schedule (S) is serializable or not?

Solution:

S' : $T_1 \rightarrow T_2$

T_1	T_2
$R_1(A)$	
$W_1(A)$	
	$R_2(A)$
	$W_2(A)$

S is not view equal to S' : $T_1 \rightarrow T_2$

S is not view equal to $T_2 \rightarrow T_1$

S' : $T_2 \rightarrow T_1$

T_1	T_2
	$R_2(A)$
	$W_2(A)$
$R_1(A)$	
$W_1(A)$	

So (S non-serializable schedule).

Example:-

T_1	T_2
$w_1(A)$	
	$w_2(A)$
	$w_2(B)$
$w_1(B)$	

(S)

S is not view equal to S': $T_1 \rightarrow T_2$

S is not view equal to S': $T_2 \rightarrow T_1$

(So it is non-serializable schedule.)

Example:-

T_1	T_2
$R_1(A)$	
	$R_2(A)$
	$R_2(B)$
$w_1(B)$	
$w_1(A)$	

this S is equal to S': $T_2 \rightarrow T_1$

So S is serializable schedule.

⇒ Problem because of Concurrent execution.

- WR problem
- RW problem
- WW problem

- Irrecoverable problem
- Cascading Rollback problem
- Lost Update problem

⇒ MIMP Classification of schedule :-

→ Serializable schedule
- (a) Conflict Serializable
- (b) View Serializable!

⇒ Recoverability schedules
- (a) Recoverable schedule
- (b) Cascadeless RB Recov. schedule
- (c) Strict Recoverable schedule

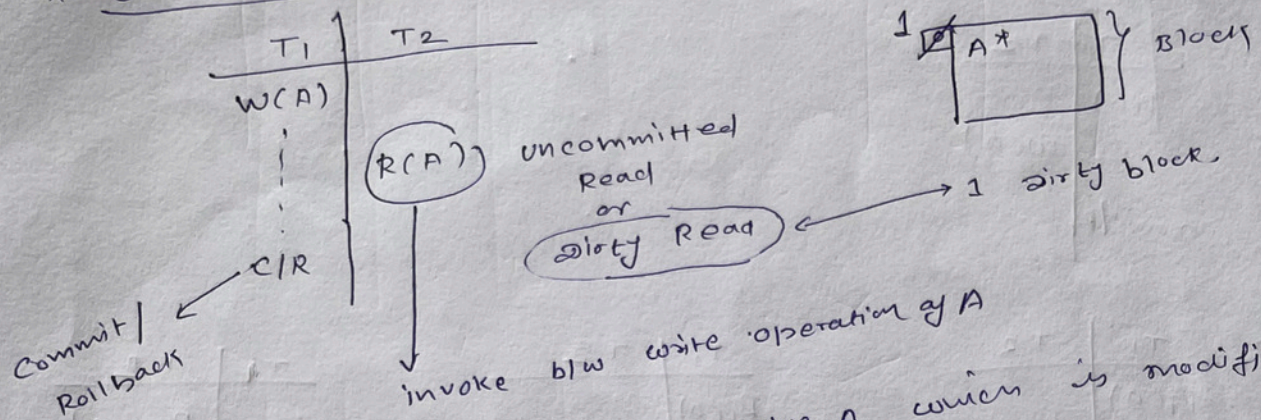
⇒ Protocols for concurrency control :-

* Locking protocol
* Timestamp ordering :-

* Problems because of Concurrent execution ⇒ (5)

→ WR- problem (write - Read) or (Read after write) problem ⇒

* Uncommitted Read ⇒



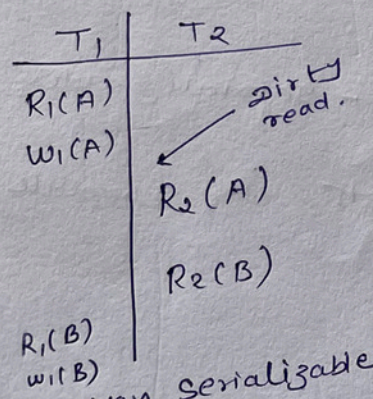
A Transaction T2 reading A which is modified by uncommitted transaction "T1".

Definition (WR - Prob.) ⇒ 1) S is non- serializable (and)

2) Uncommitted Read exist in S

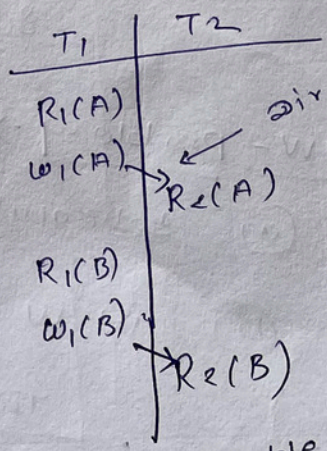
⇒ Test for following schedule:-

Ex:- (1)



S : Non serializable schedule and Dirty Read exist

Ex- (2)



S : Serializable schedule [T1 → T2 serial] but dirty read exist.

→ RW - Problem!-

(a) Schedule S is Non Serializable
and

(b) R-W Conflict exists.

T_1	T_2
R(A)	
	W(A)
commit/R	

Ex -

T_1	T_2
$R_1(A)$	
	$R_2(A)$
	$W_2(A)$
$W_1(A)$	
Commited here	

T_1	T_2
$R_1(A)$	
$W_1(A)$	
	$R_2(A)$
	$W_2(A)$

T_1	T_2
	$R_2(A)$
	$W_2(A)$
$R_1(A)$	
$W_1(A)$	

S: Non serializable schedule

Read - write conflict is here

So RW - problem exist

→ WW - Problem =>

(a) Schedule S is non - serializable!
and

(b) W-W conflict exists in schedule S.

T_1	T_2
$W_1(A)$	
	$W_2(A)$
C/R.	

Ex:1 ①

T_1	T_2
$W_1(A)$	
	$W_2(A)$
	$W_2(B)$
$W_1(B)$	

S: Non-serializable
[WW problem]

Ex:2 ②

T_1	T_2
$W_1(A)$	
	$W_2(A)$
$W_1(B)$	
	$W_2(B)$

✗ [WW problem]
S: serializable. ④

To avoid WR, RW, WW problems concurrency control Component should restrict, Non-serializable schedule.

⇒ IRRECOVERABLE Problems:- [Rollbacking of ~~Commi~~ Comitted Trans.]

A ∅ 10

T_1	T_2
$W_1(A)$	
Possible	$R(A) = 10$
	commit
failed	Roll back Not Possible

[IRRECOVERABLE schedule]

Ex:- A: Joint Account A: ~~₹0000~~ 9000

T_1	T_2
$R(A)$	
$A = A - 1000$	
$W(A)$	
	$R(A) - 9000$
	commit
failed	Not possible [IRRECOVERABLE]

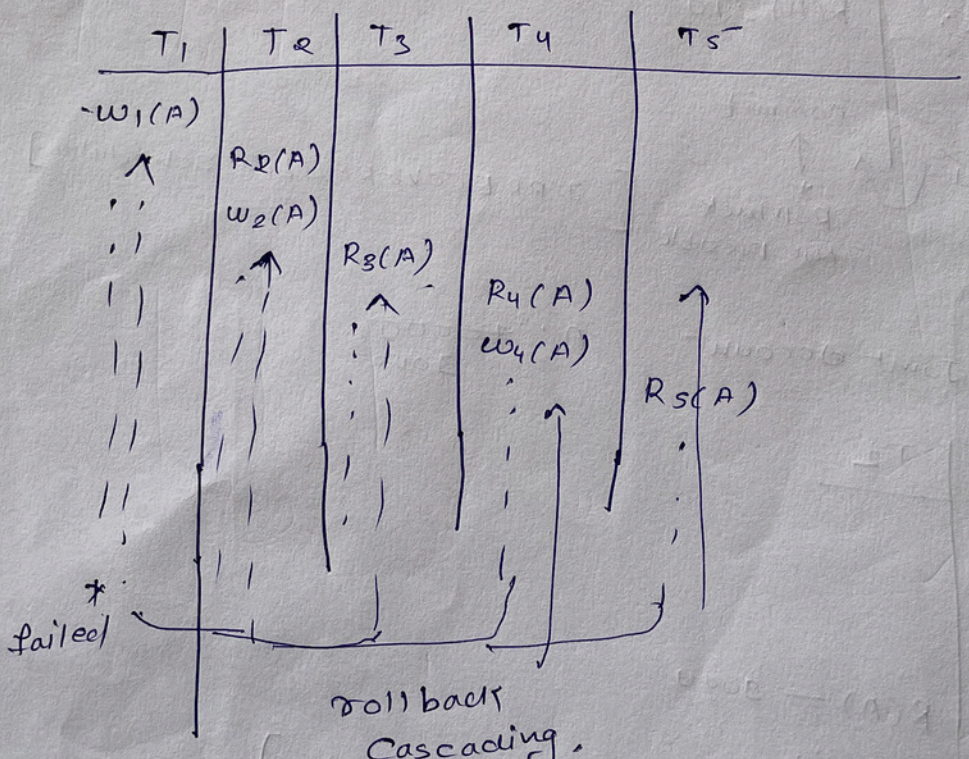
⇒ IRRECOVERABLE Schedule condition:⇒

T_1	T_2
$W_1(A)$	
	$R(A)$
	C_2
C/R	

⇒ If transaction T_2 reads "A" which is modified by uncommitted Transaction T_1 and commit of T_2 before C/R of Trans. T_1.

They this type of schedules known as IRRECOVERABLE schedule (which also causes inconsistency).

⇒ Cascading Rollback problem!- Because of failure of one transaction required to rollback some set of other transaction also, is called Cascading rollback.



→ Cascading rollback happened!-

→ There is no Inconsistency.

Dis advantage!- ① wastage of CPU execution time

② and I/O access cost.

* Lost update problem ⇒ W(A) operation of some transaction is lost.

T_1	T_2
$w_1(A)$	
	$w_2(A)$ = 20 : Lost update !.
failed !	

A = 0 → 10 → 20 (T_1, T_2)

Transig.

T_1 : A_{old} = 0
A_{new} = 10
→ roll back and give A = 0

T_2 : A_{old} = 10, new = 20

[The final 'A' value must be "20" but bcz of T_1 Rollback final A value is "0"] [Inconsistent result]

* To avoid IRRECOVERABLE :-

T_1	T_2
$w_1(A)$	
	$R_2(A)$
C_1/R_1	
	C_2

* To avoid cascading Rollback → (restrict uncommitted reads).

T_1	T_2
$w(A)$	
R/C	
	$R_2(A)$.

* To avoid lost update problem

T_1	T_2
$w_1(A)$	
C/R	
	$w_2(A)$.

* classification based on Recoverability:-

(1) Recoverable schedule ⇒ Schedule S is is recoverable, iff

(a) No uncommitted reads in S,
or.

(b) If Transaction T_2 reads "A" which is modified by T_1 they commit of T_2 should delay unit 'commit / Rollback of T_1.

T_1	T_2
$W_1(A)$	
	$R_2(A)$
C_1 / R_1	
	C_2

Q. which of the following schedule are recoverable?

1)

T_1	T_2
$W_1(A)$	
	$R_2(A)$
	R_2
C_1 / R_1	

⇒ Recoverable.

2)

T_1	T_2
$R_1(A)$	
	$W_2(A)$
	$W_2(B)$
$W_1(B)$	
	C_2
C_1	

No- uncommitted reads so it is recoverable schedule.

3)

T_1	T_2
$R_1(A)$	
	$R_2(A)$
	$W_2(A)$
$W_1(A)$	
	C_2
C_1	

No- uncommitted read

↓

Recoverable schedule

4)

T_1	T_2
$R(A)$	
	$W(A)$
	$W(B)$
$R(B)$	
$W(C)$	
	$W(C)$
	C_2
C_1	

Recoverable schedule

Recoverable → Condition → true from :->
↑ Irrecoverable schedule.

Not true from ⇒
* Cascading RB
* Lost Update
* RW / WR / WW problems.

2) Cascadeless Rollback Recoverable schedules ⇒
[avoid cascading rollback in Recoverable schedule]
[Restrict the ~~the~~ uncommitted reads]

schedule S is cascadeless RB recoverable schedules if and only if:-

Transaction T_1 writes data item "A" some other trans. T_2, R(A) should delay until C/R of T_1.

T_1	T_2
$W_1(A)$	
C/R.	
	$R_2(A)$

Q. Find which of the schedules is Cascadeless RB Recoverable:-

1)

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

Cas. RB Recove
→ No uncomm. read.

2)

T_1	T_2
$W_1(A)$	
	$W_2(A)$
	$W_2(B)$
$W_1(B)$	
	C2
C_1	

Cas. RB Recove.
→ No uncommitted read.

3)

T_1	T_2
$W_1(A)$	
	$W_2(A)$
	$R_2(A)$
	C2

→ This is not a uncommitted Read so it is also Cas. RB Recoverable

→ Cascadeless RB Rec. schedule :-

Free from :-> irrecoverable problem
→ Cascading rollback problem
→ write read problem

Not free from :- Lost update problem is possible
→ RW problem
→ WW problem

(3) strict Recoverable schedule :-

{ Cascadeless RB Rec. } + { No lost update problem }

⇓

T_1	T_2
$w_1(A)$	
c/R	
	$R_2(A)$

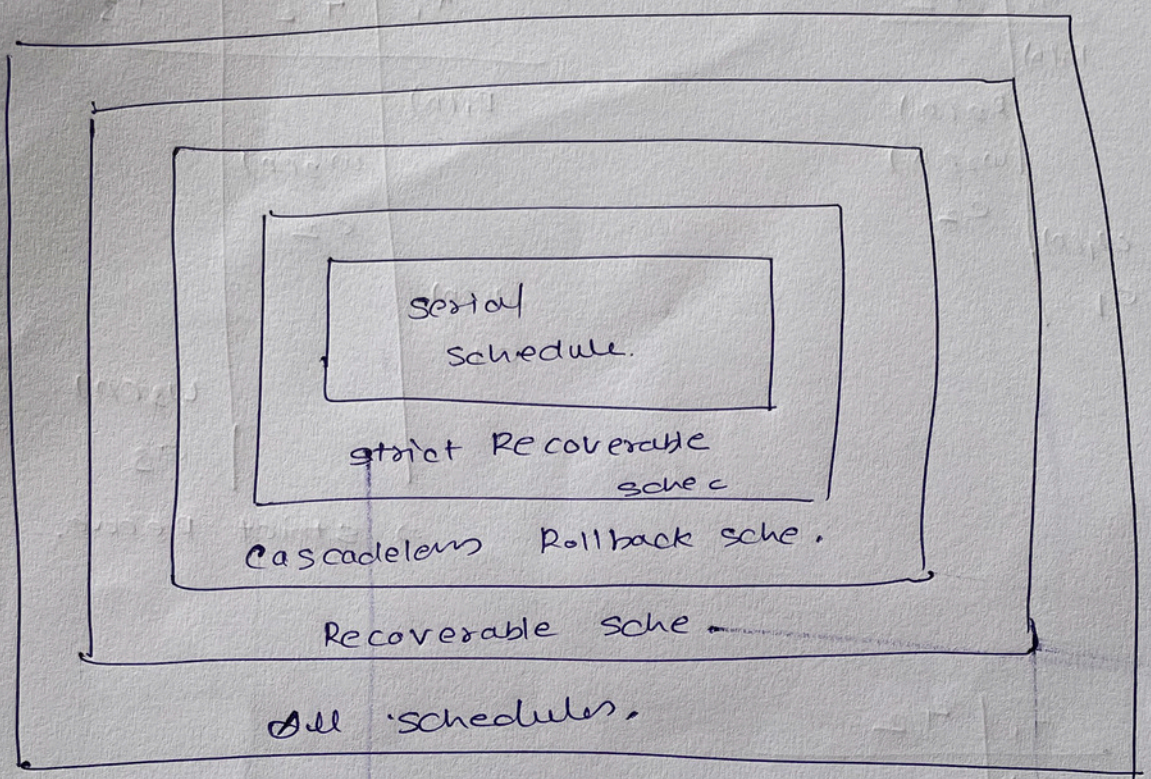
T_1	T_2
$w_1(A)$	
c/R	
	$w_2(A)$

⇓

T_1	T_2
$w_1(A)$	
c/R	
	$R_2(A) / w_2(A)$

⇒ { strict Recoverable schedule }

⇒ If transaction T_1 writes "A" then $R(A) / w(A)$ by Transaction T_2 should delay until c/R of T_1.


Serial
schedule.
strict Recoverable
sche c
Cascadeless Rollback sche.
Recoverable sche
All schedules.

Q. which of the schedules are strict recoverable? -

(1)

T_1	T_2
$R_1(A)$	
	$R_2(A)$
	$W_2(A)$
	C_2
$W_1(A)$	
C_1	

2)

T_1	T_2	T_3
$R_1(A)$		
	$W_2(A)$	
	C_2	
$W_1(A)$		
C_1		
		$W_3(A)$
		C_3

⇒ strict Recove.

(3)

T_1	T_2
$W_1(A)$	
	$W_2(A)$
	$R_2(A)$ ⇒ Not strict Recov.
	C_2
C_1	

→ strict Recoverable schedule —

Free from :- → Irrecoverable
→ Cascading RB
→ WR problem
→ Lost update
→ WW problem

Not free from → RW problem.