COMPUTER SCIENCE

Database Management System

Transaction Concept:
Recoverable Schedule &
Lock Based Protocol(2PL,
Strict 2PL, Conservation 2PL,
Time stamp Protocol

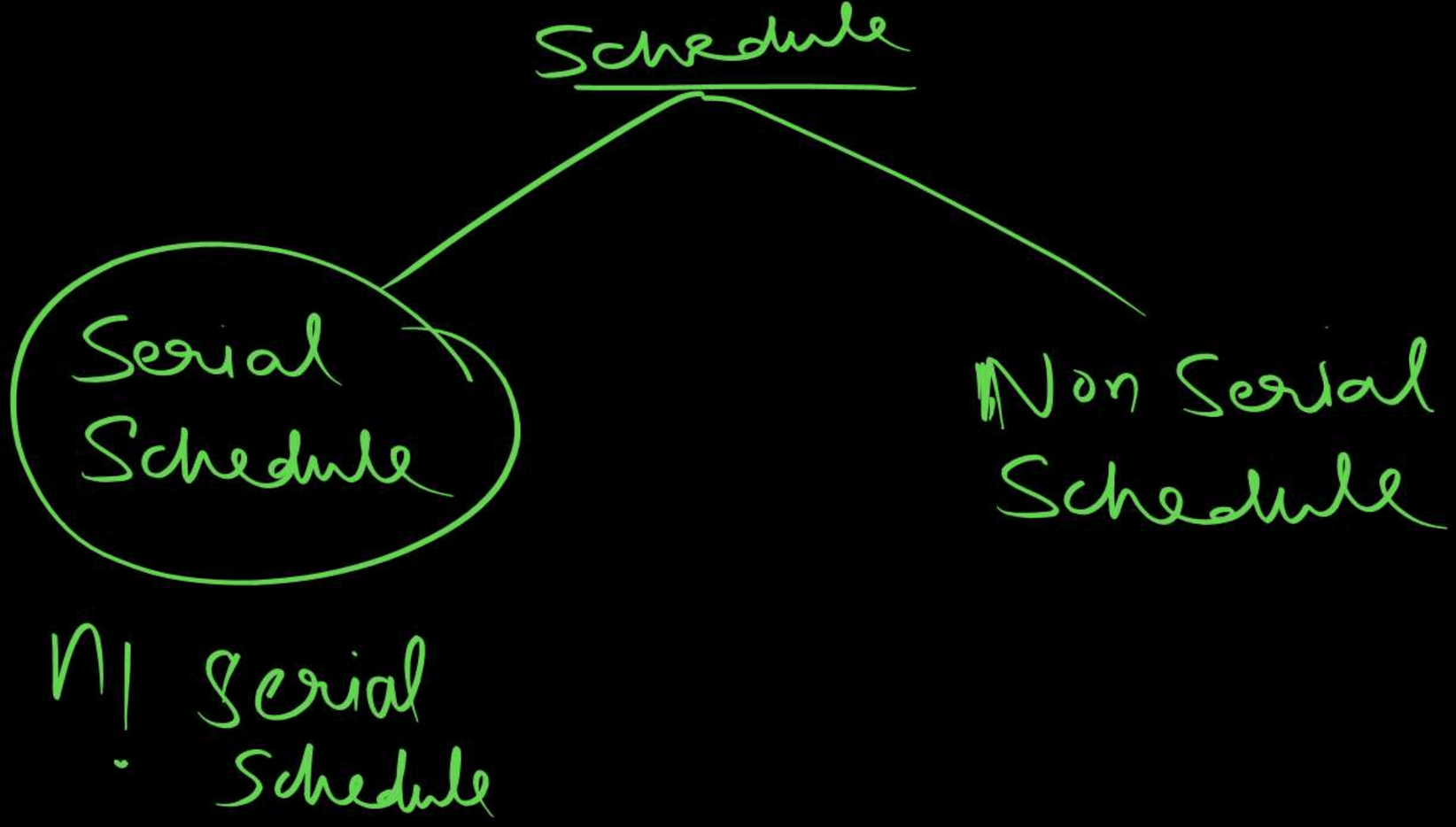
Lecture_04

Vijay Agarwal sir

Today Goal

Transaction Concept

- ACID Property


Schedule
Serial Schedule
Non Serial Schedule
n! Serial Schedule

2 Transaction then 2! Serial schedul

$\langle T_1\ T_2 \rangle$

$\langle T_2\ T_1 \rangle$

3 Transaction then 3! ⇒ 6 Serial schedule

$\langle T_1\ T_2\ T_3 \rangle$

$\langle T_1\ T_3\ T_2 \rangle$

$\langle T_2\ T_1\ T_3 \rangle$

$\langle T_2\ T_3\ T_1 \rangle$

$\langle T_3\ T_1\ T_2 \rangle$

$\langle T_3\ T_2\ T_1 \rangle$

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.

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

Schedule 1

T_1	T_2
read (A)	
A: = A – 100	
write (A)	
read (B)	
B: = B + 100	
write (B)	
commit	
	read (A)
	temp := A * 0.1
	A := A – temp
	write (A)
	read (B)
	B := B + temp
	write (B)
	Commit

Consistent

A: 1710
B: 3290
5000

Serial schedule in which T_1 is followed by T_2:

Schedule 2

T_1	T_2
	read (A)
	temp := A * 0.1
	A := A – temp
	write (A)
	read (B)
	B := B + temp
	write (B)
	Commit
read (A)	
A: = A – 100	
write (A)	
read (B)	
B: = B + 100	
write (B)	
commit	

$S_2 < T_2\ T_1 >$

Consistent

A: 1700
B: 3300
5000

serial schedule where T_2 is followed by T_1

#q transaction

Note Serial Schedule (all Serial Schedule [n!])

are always Consistent

Note Non Serial Schedule (Concurrent execution)

May ⓞⓡ May Not be Consistent

Why Concurrent Execution ?

- To Improve CPU Utilization
- Enhanced Throughput
- Effective Resource Utilization
- etc

Schedule 3

T_1	T_2
read (A)	
A: = A – 100	
write (A)	
	read (A)
	temp := A * 0.1
	A := A – temp
	write (A)
read (B)	
B: = B + 100	
write (B)	
commit	
	read (B)
	B := B + temp
	write (B)
	Commit

C_1

Schedule 4

T_1	T_2
read (A)	
A: = A – 100	
	read (A)
	temp := A * 0.1
	A := A – temp
	write (A)
	read (B)
write (A)	
read (B)	
B: = B + 100	
write (B)	
commit	
	B := B + temp
	write (B)
	Commit

C_2

C_1 ⇒ Equal to Serial Schedule $S_1 < T_1\ T_2 >$

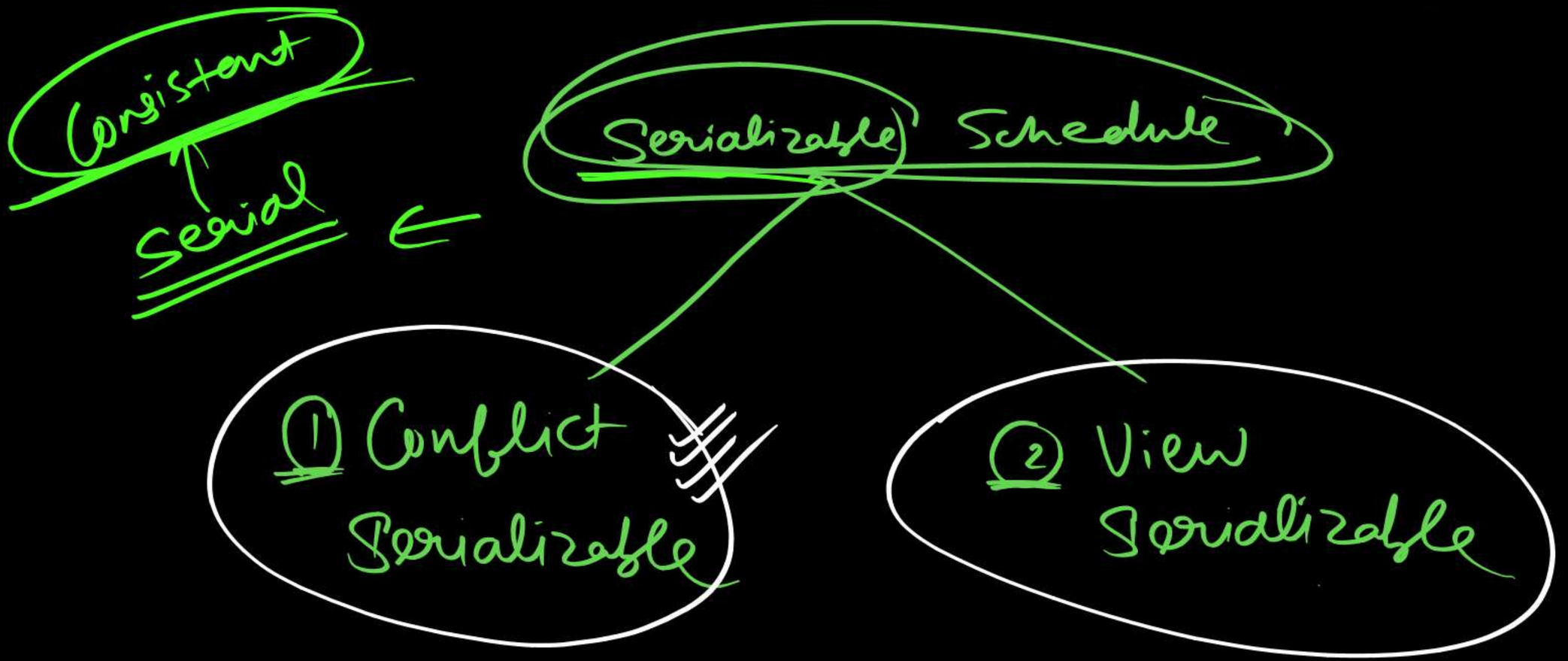
Serializable

⇓

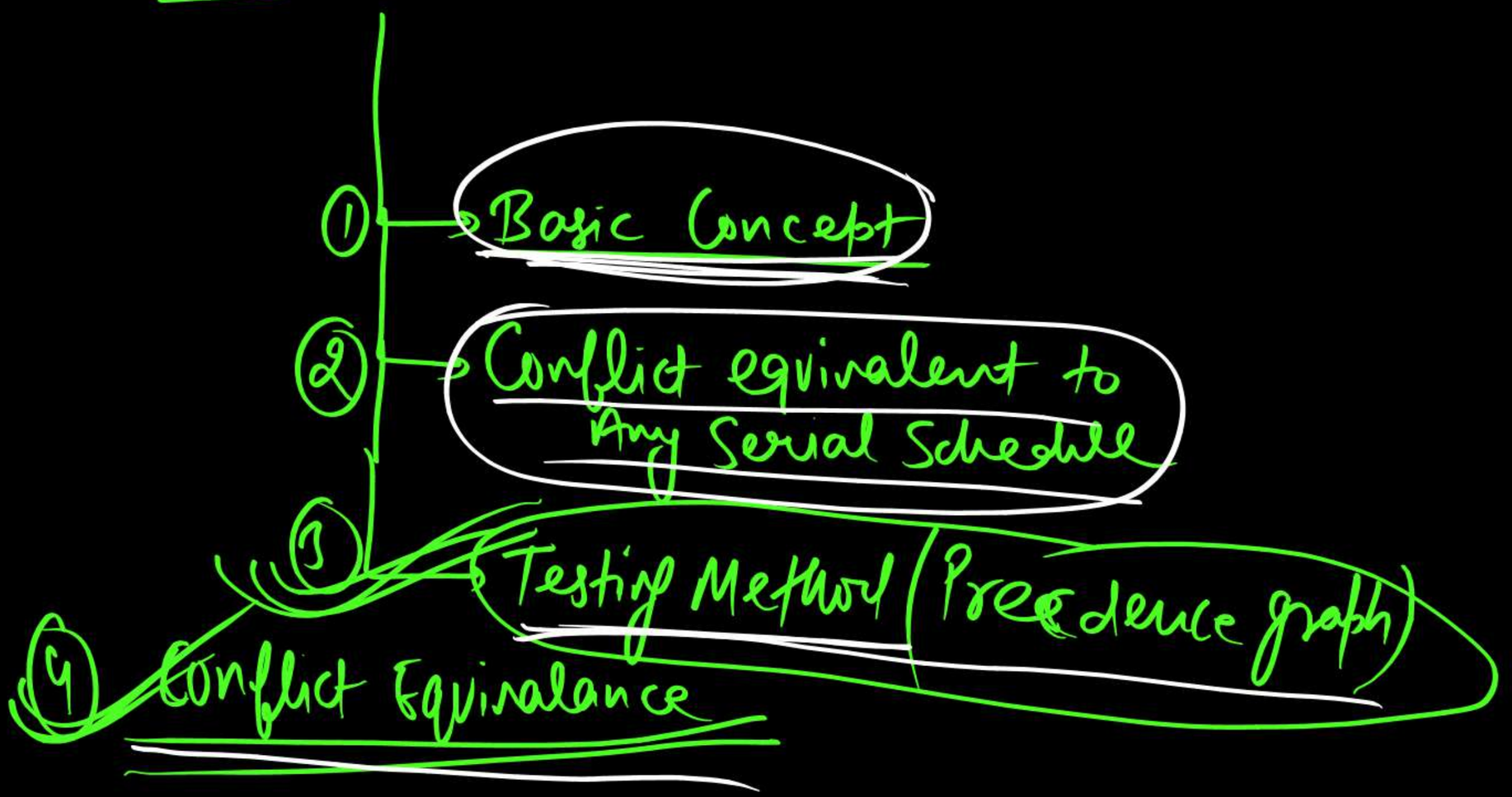
Consistent

Serializable Schedule : If a Concurrent Execution (Non Serial Schedule) has been executed, that could have the same effect on the Database, as a Schedule executed without any Concurrent Execution (Any Serial Schedule). Called Serializable Schedule.

Note: Serializable Schedule always Consistent


Consistent
Serial
Serializable Schedule
1 Conflict Serializable
2 View Serializable

Conflict Serializable



Conflict Instn

Same Data

R(A) – W(A)

W(A) – R(A)

W(A) – W(A)

Non Conflict

R(A) – R(A)

① Different Data Item

Non Conflict Instruction

$(i \neq j)$

Same Data Item	T_i	T_j	
	R(A)	R(A)	Non Conflicting Instruction
Operating on Different Data Item	R(A)	W(B)	

Conflict Instⁿ

T_i	T_j	Same Data Item
R(A)	W(A)	Conflict Instruction
W(A)	R(A)	
W(A)	W(A)	

R - W
W - R
W - W

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

Conflicting Instructions

- Instructions I_i, and I_j of transactions T_i and T_j respectively, conflict if and only if there exists some item Q accessed by both I_i, and I_j, and at least one of these instructions wrote Q.
  1. I_i, = read(Q), I_j = read(Q). I_i and I_j don't conflict.
  2. I_i, = read(Q) I_j = write(Q). They conflict.
  3. I_i, = write(Q) I_j = read(Q). They conflict
  4. I_i= write(Q) I_j = write(Q). They conflict
- Intuitively, a conflict between I_i and I_j forces a (logical) temporal order between them.
  - If I_i, and I_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.

Conflict Serializable

Any Serial Schedule

Concurrent Schedule $\neq$

$\equiv$

Non Serial Schedule

Convert into

Swap of series of Non Conflicting Instruction

⇒ Any serial schedule

Conflict Serializability

Def 1 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.

Def 2 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.

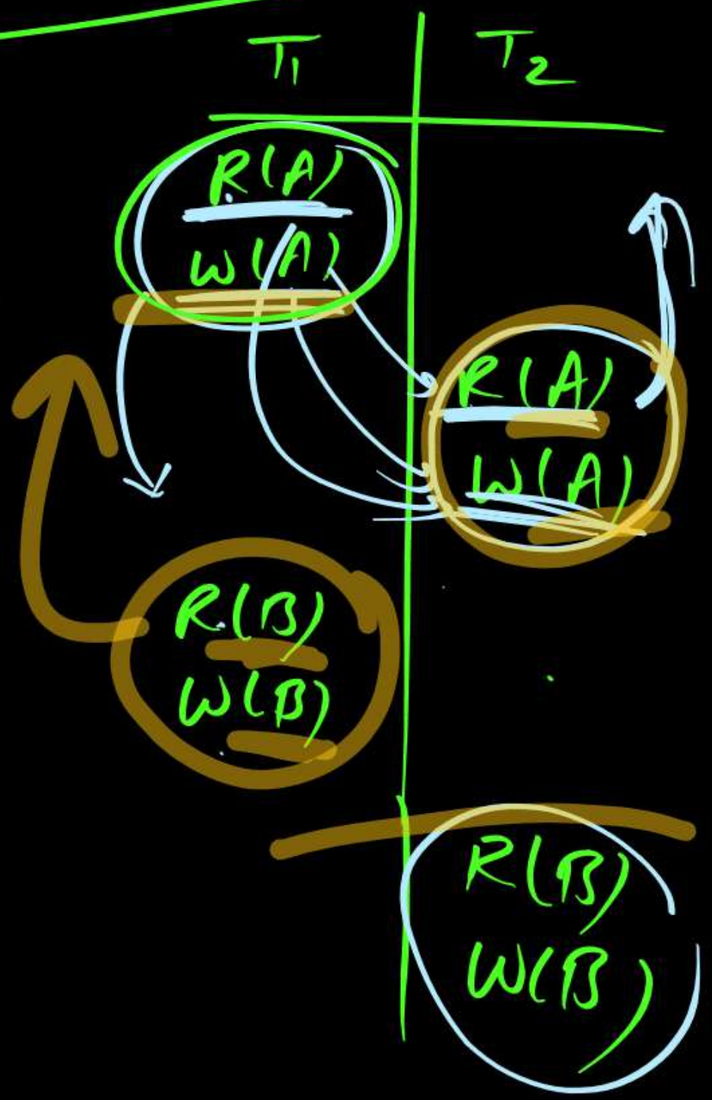
Schedule 3

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 Serializable $S_1 < T_1\ T_2 >$



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

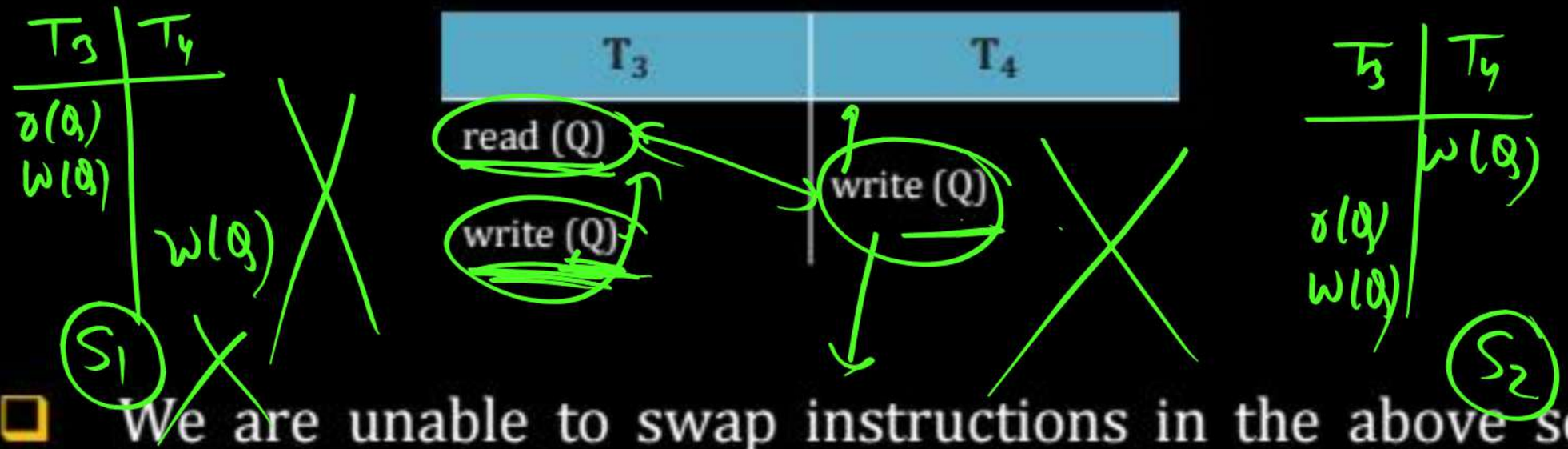
$S_2 < T_2\ T_1 >$

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

$S_1 < T_1\ T_2 >$

Conflict Serializability (Cont.)

- Example of a schedule that is not conflict serializable:



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


T1
T2
R(Q)
W(Q)
W(Q)

Testing for Conflict Serializability

Directed edge

Precedence Graph Method

Directed Graph

$G : (V, E)$

V : Vertex : Set of Transaction

E : Edge (Directed) : $T_i \rightarrow T_j$

T_i

Conflict Instⁿ	
R(Q)	T_j : W(Q)
W(Q)	R(Q)
W(Q)	W(Q)

CNC : [Cycle Not Conflict]

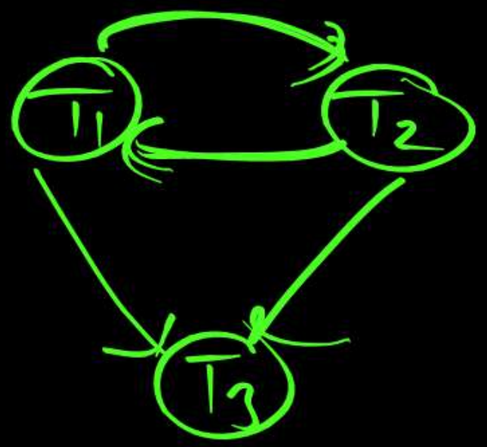
Cycle

Not

Conflict

Note

Precedence graph Contain Any Cycle then Not Conflict Serializable

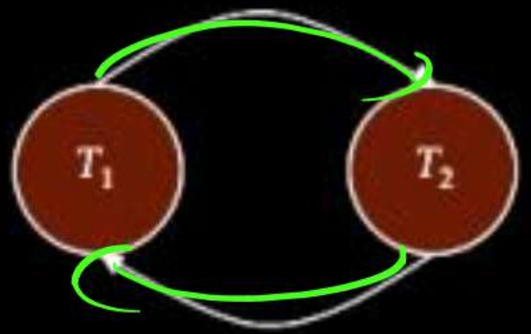


Cycle Not Conflict.

Testing for Serializability

- Testing for conflict serializability.
  - Consider some schedule of a set of transactions T_1, T_2, ...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:



CNC : Cycle Not Conflict.

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

NOTE: CNC [Cycle not conflict serializable]

Q.

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

$(i \neq j)$

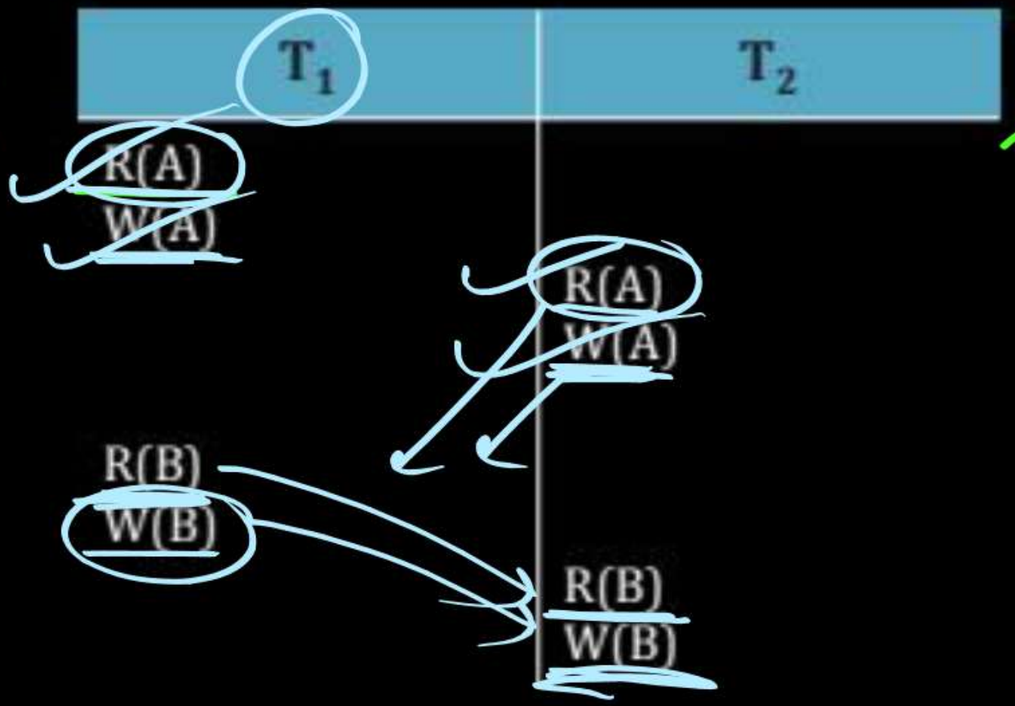
Directed edge

$T_i \rightarrow T_j$

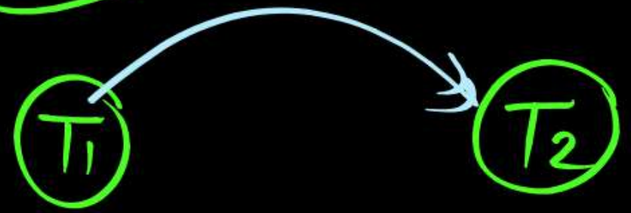
$R(Q) \Rightarrow W(Q)$

$W(Q) \Rightarrow R(Q)$

$W(Q) \Rightarrow W(Q)$



A: 1710
+B: 3290
5000 ≡ S: <T1 T2>



Conflict Serializable

<T1 T2>.

Q.

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

$(i \neq j)$

Directed edge

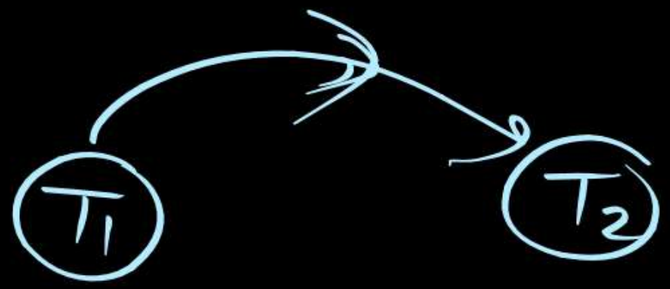
$T_i \rightarrow T_j$

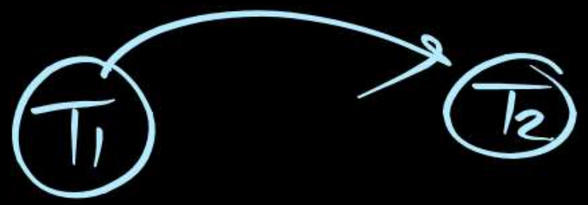
$R(Q) \Rightarrow W(Q)$

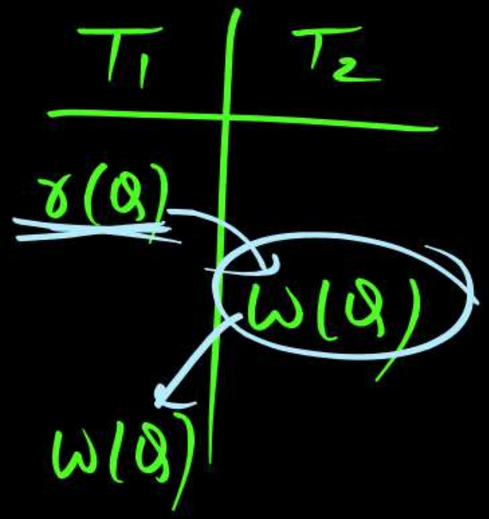
$W(Q) \Rightarrow R(Q)$

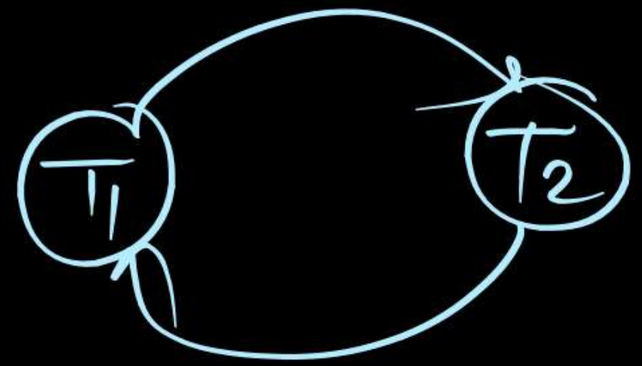
$W(Q) \Rightarrow W(Q)$

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




T1
T2





Cycle Not Conflict.

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)

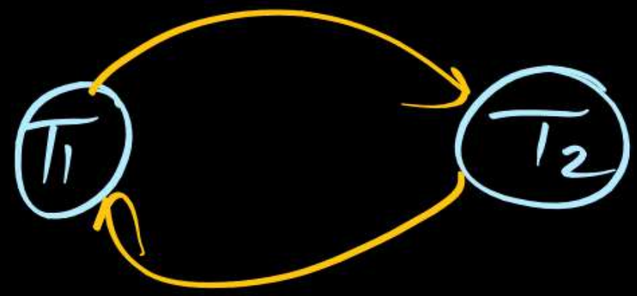
5200

5100

Inconsistent

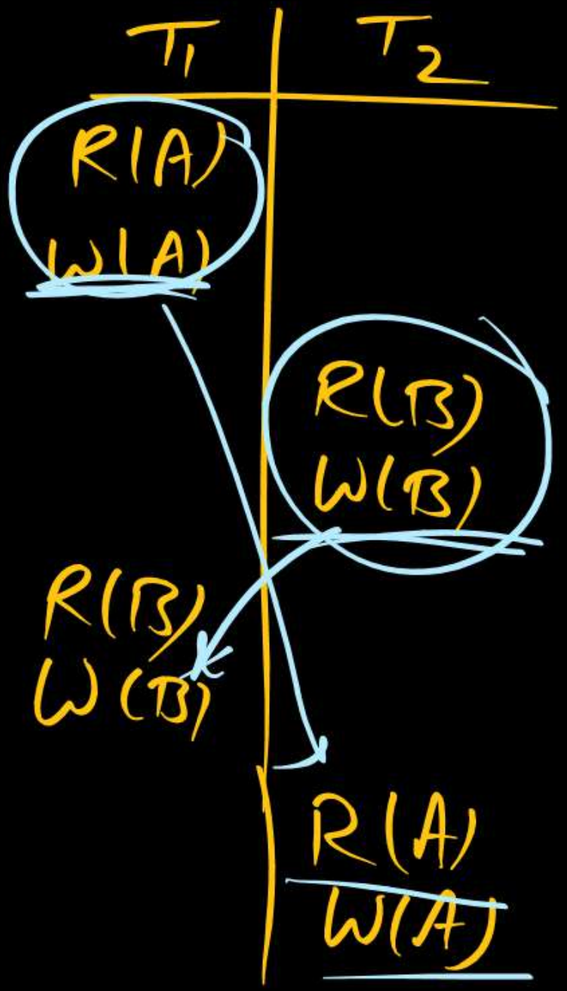
CNC

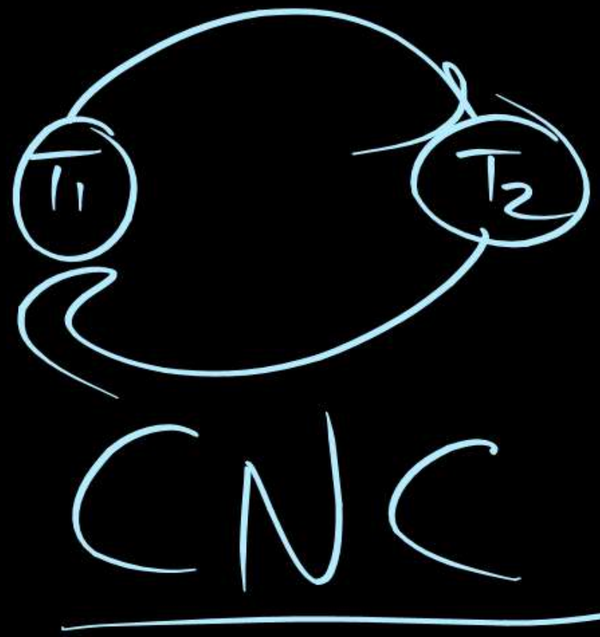
Cycle Not Conflict

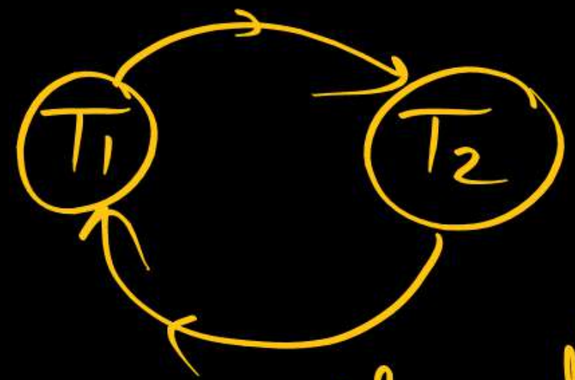


Q.

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







Cycle
Not Conflict

Start from Ti which Indegree = 0 ⇒ No Incoming edge

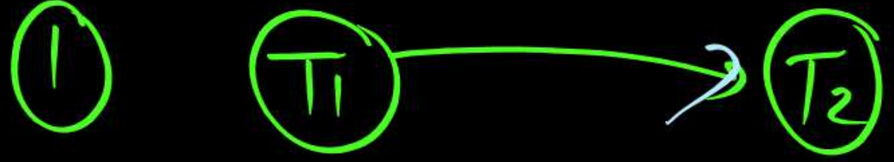
(S)

Serializability Order

(Topological Sorting)

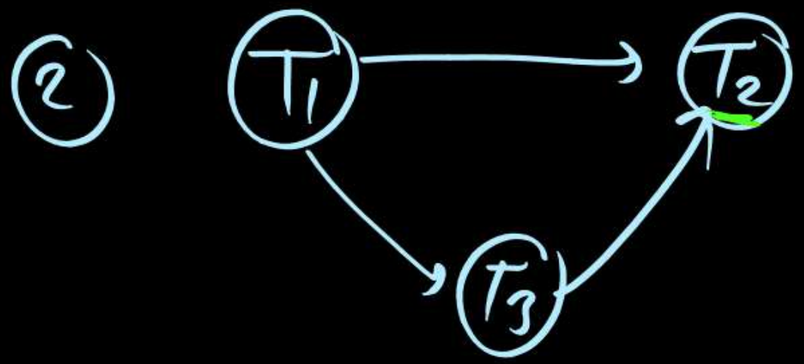
If Conflict Serializable schedule equivalent to Which serial Schedule ab S.

Serializability order

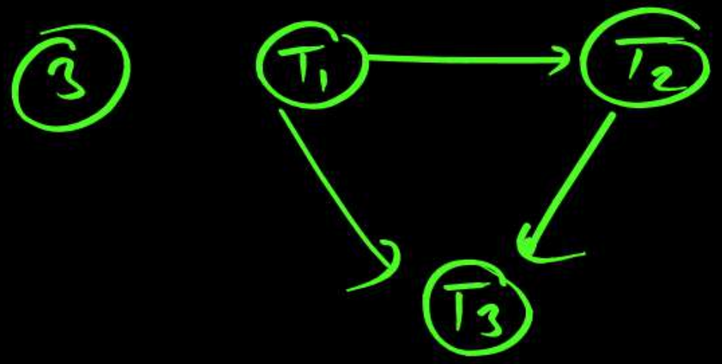
(1) 

⟨T1 T2⟩

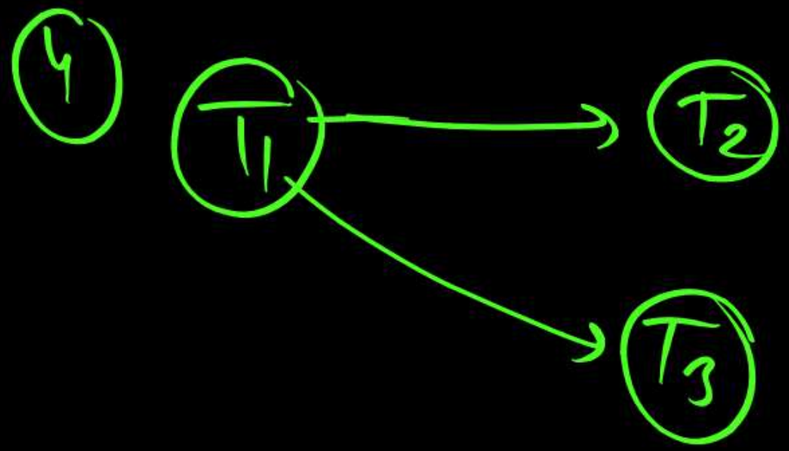
Serial schedule
⟨T1 T2⟩

(2) 

⟨T1 T3 T2⟩.



$\langle T_1\ T_2\ T_3 \rangle$



$\langle T_1\ T_2\ T_3 \rangle$

(or)

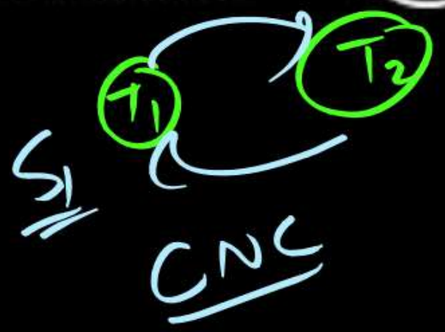
$\langle T_1\ T_3\ T_2 \rangle$.

Q.

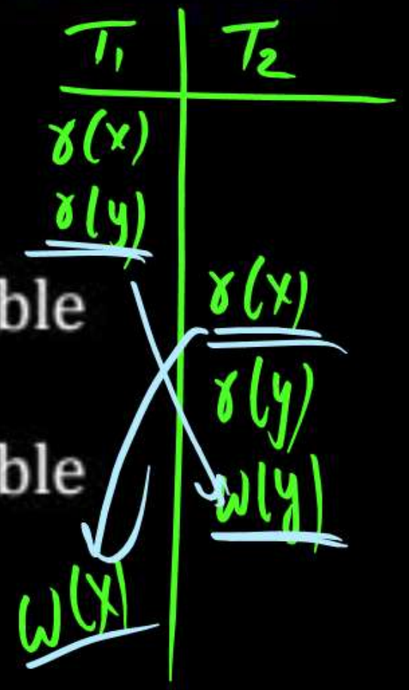
Consider the following schedules involving two transactions. Which one of the following statements is TRUE?

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

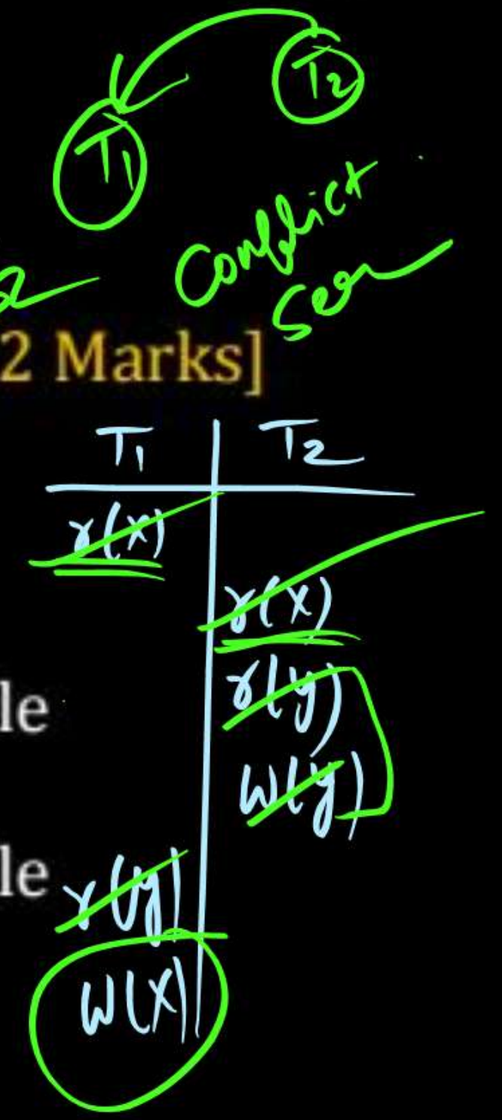
Q.

Consider the following 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 on a data element z by a transaction T_i, respectively. Consider the schedule S with four transactions.

S: $R_4(x), R_2(x), R_3(x), R_1(y), W_1(y), W_2(x), W_3(y), R_4(y)$

Which one of the following serial schedules is conflict equivalent to S?

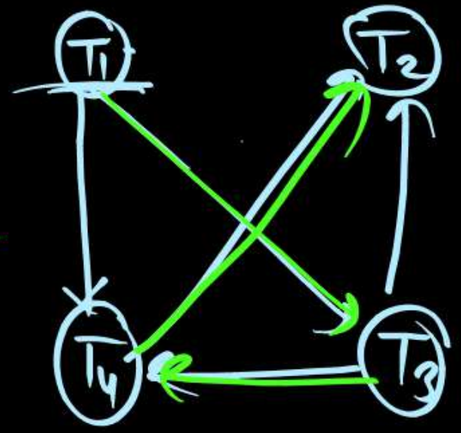
[2022: 2 Marks]

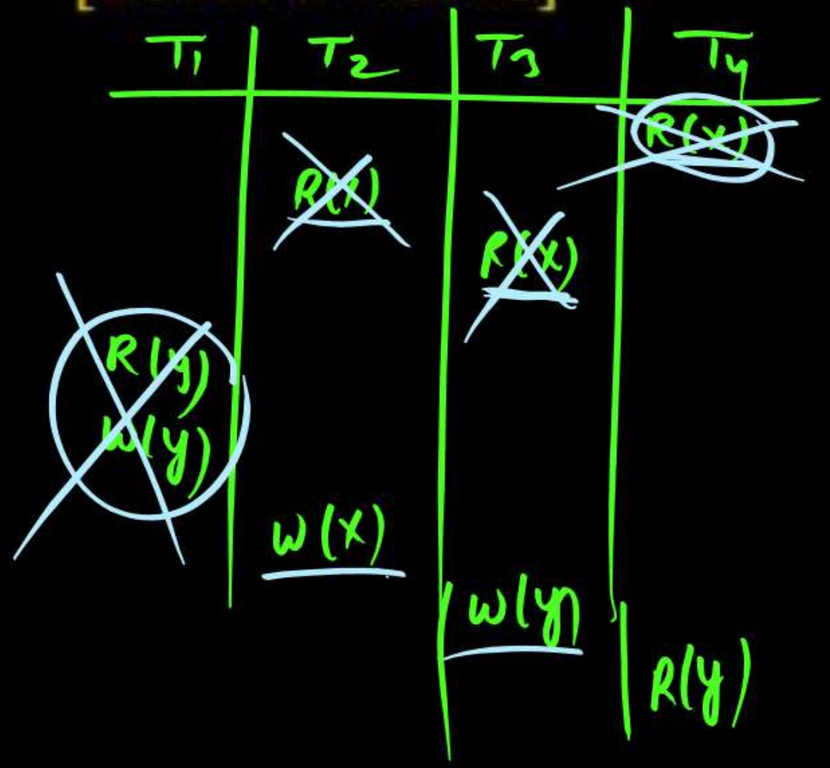
A $T_1 \rightarrow T_3 \rightarrow T_4 \rightarrow T_2$

B $T_1 \rightarrow T_4 \rightarrow T_3 \rightarrow T_2$

C $T_4 \rightarrow T_1 \rightarrow T_3 \rightarrow T_2$

D $T_3 \rightarrow T_1 \rightarrow T_4 \rightarrow T_2$





Q.

Let $R_i(z)$ and $W_i(z)$ denote read and write operations on a data element z by a transaction T_i, respectively. Consider the schedule S with four transactions.

S: $R_4(x), R_2(x), R_3(x), R_1(y), W_1(y), W_2(x), W_3(y), R_4(y)$

Which one of the following serial schedules is conflict equivalent to S?

[2022: 2 Marks]

A $T_1 \rightarrow T_3 \rightarrow T_4 \rightarrow T_2$

B $T_1 \rightarrow T_4 \rightarrow T_3 \rightarrow T_2$

C $T_4 \rightarrow T_1 \rightarrow T_3 \rightarrow T_2$

D $T_3 \rightarrow T_1 \rightarrow T_4 \rightarrow T_2$

x: $T_4 \rightarrow T_2$, $T_3 \rightarrow T_2$

y: $T_1 \rightarrow T_3$, $T_1 \rightarrow T_4$, $T_3 \rightarrow T_4$

$T_1 \rightarrow T_3 \rightarrow T_4 \rightarrow T_2$

Q.

Consider the following transaction involving two bank accounts x and y.

read(x); x: = x – 50; write (x); read (y); y: = y + 50; write (y)

The constraint that the sum of the accounts x and y should remain constant is that of [2015(Set-2): 1 Marks]

A Atomicity

B Consistency

C Isolation

D Durability

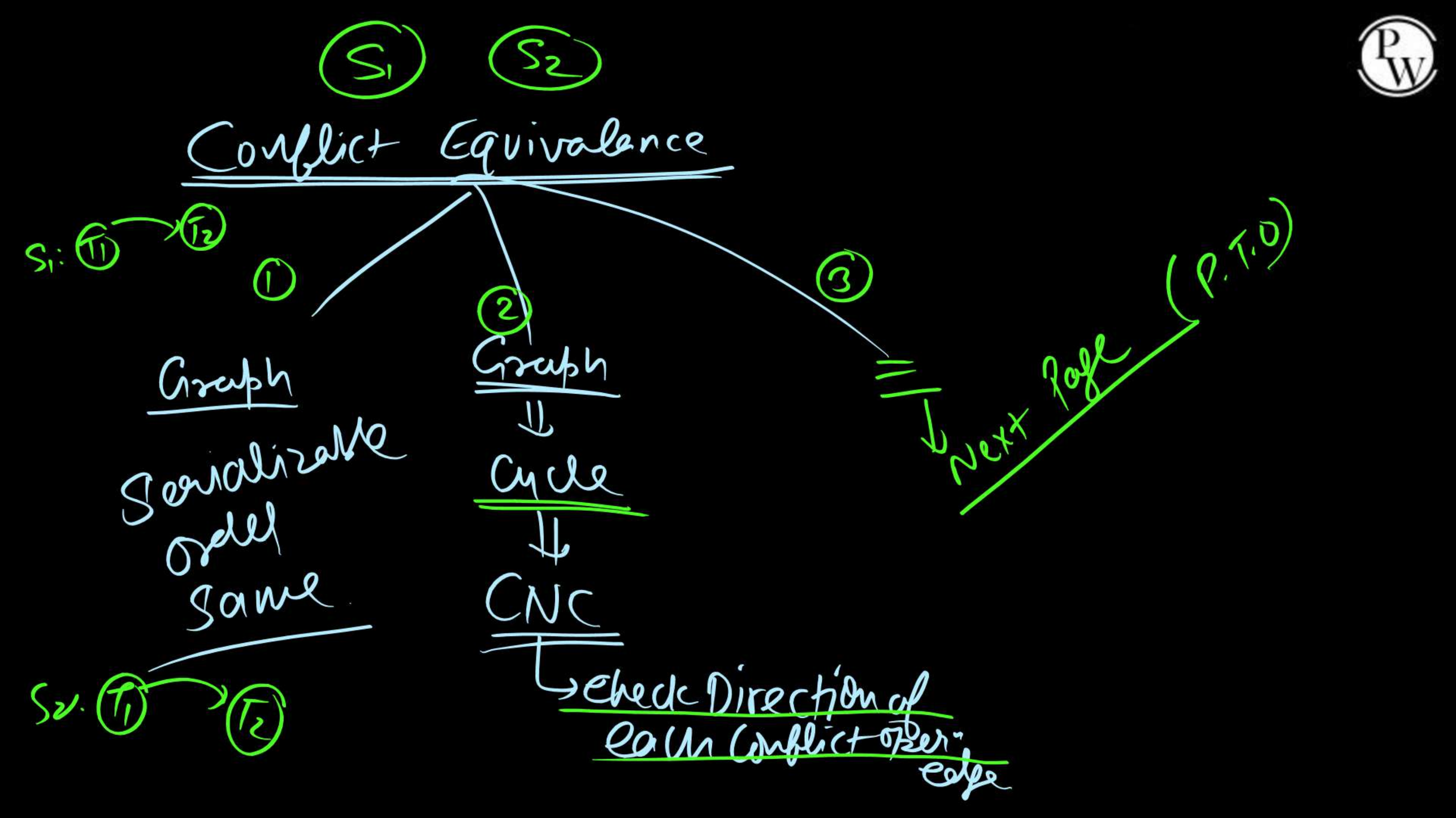
Conflict Serializable ⇒ Core Concept

↳ Swap of Non Conflict Instn ⇒ Any Serial Schedule

Testing Method

Conflict Equivalence

Conflict Serializable


S1
S2
Conflict Equivalance
S1: T1 T2
1
2
3
Graph
Serializable order same
Graph
Cycle
CNC
check Direction of each Conflict operation edge
Next Page (P.T.O)

Conflict Equivalance

S_1 & S_2 Conflict equivalance if all Conflict operation Must be executed in Same Order in S_1 & S_2.

T_i	T_j
R(A)	W(A)
W(A)	R(A)
W(A)	W(A)

} Conflict operation.

Conflict Equivalent Schedule

Two schedule are said to be conflict equivalent, if all conflicting operations in both the schedules must be executed in the same order.

Q.

S_1: $R_1(x)$ $W_1(x)$ $R_2(y)$ $W_2(y)$ $R_1(y)$

S_2: $R_1(x)$ $W_1(x)$ $R_1(y)$ $R_2(y)$ $W_2(y)$

S_1 Not Conflict Equivalent S_2

T_1	T_2
R(x)	
W(x)	
R(y)	
	R(y)
	W(y)

$R_1(y) - W_2(y)$

or

R-W: $T_1 \rightarrow T_2$

S_1

$W_2(y) - R_1(y)$

or

W→R: $T_2 \rightarrow T_1$

T_1	T_2
R(x)	
W(x)	
	R(y)
	W(y)
R(y)	

Q.

S_1: $R_1(A)$ $W_1(A)$ $R_2(A)$ $W_2(A)$ $R_1(B)$ $W_1(B)$

S_2: $R_1(A)$ $W_1(A)$ $R_2(A)$ $R_1(B)$ $W_2(A)$ $W_1(B)$

(S1)

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

S_1

$R_1(A) - W_2(A)$

$W_1(A) - R_2(A)$

$W_1(A) - W_2(A)$

R(A)-W(A) : $T_1 \to T_2$

W(A)-R(A) : $T_1 \to T_2$

W(A)-W(A) : $T_1 \to T_2$

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

$T_1 \to T_2$

$R_1(A) - W_2(A)$

$W_1(A) - R_2(A)$

$W_1(A) - W_2(A)$

Q.

Consider a schedule of transactions T_1 and T_2:

T_1	RA			RC		WD		WB	Commit	
T_2		RB	WB		RD		WC			Commit

Here, RX stands for "Read(X)" and WX stands for "Write(X)". Which one of the following schedules is conflict equivalent to the above schedule?

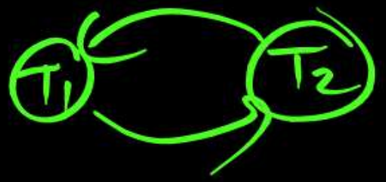
[2020: 2 Marks]

A

T_1					RA	RC	WD	WB	Commit	
T_2	RB	WB	RD	WC						Commit

B

T_1				RA	RC	WD	WB		Commit	
T_2	RB	WB	RD					WC		Commit



C

T_1	RA	RC	WD	WB					Commit	
T_2					RB	WB	RD	WC		Commit

D

T_1	RA	RC	WD	WB					Commit	
T_2					RB	WB	RD	WC		Commit

Q.

Consider a schedule of transactions T_1 and T_2:

T_1	RA			RC		WD		WB	Commit	
T_2		RB	WB		RD		WC			Commit

$R_2(B) - W_1(B)$

$W_2(B) - W_1(B)$

$R_1(C) - W_2(C)$

$R_2(D) - W_1(D)$

Here, RX stands for "Read(X)" and WX stands for "Write(X)". Which one of the following schedules is conflict equivalent to the above schedule?

[2020: 2 Marks]

A

T_1					RA	RC	WD	WB	Commit	
T_2	RB	WB	RD	WC						Commit

B

T_1				RA	RC	WD	WB		Commit	
T_2	RB	WB	RD					WC		Commit

C

T_1	RA	RC	WD	WB					Commit	
T_2					RB	WB	RD	WC		Commit

D

T_1	RA	RC	WD	WB					Commit	
T_2					RB	WB	RD	WC		Commit

$R_2(B) - W_1(B)$

$W_2(B) - W_1(B)$

$R_2(D) - W_1(D)$

$R_1(C) - W_2(C)$

Q.

Consider a schedule of transactions T_1 and T_2:

T_1	RA			RC		WD		WB	Commit	
T_2		RB	WB		RD		WC			Commit

(B)

Here, RX stands for "Read(X)" and WX stands for "Write(X)". Which one of the following schedules is conflict equivalent to the above schedule?

[2020: 2 Marks]

A

T_1					RA	RC	WD	WB	Commit	
T_2	RB	WB	RD	WC						Commit

B

T_1				RA	RC	WD	WB		Commit	
T_2	RB	WB	RD					WC		Commit

C

T_1	RA	RC	WD	WB					Commit	
T_2					RB	WB	RD	WC		Commit

D

T_1	RA	RC	WD	WB					Commit	
T_2					RB	WB	RD	WC		Commit

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_L & S_1

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

$R_1(A) - W_2(A)$
$W_1(A) - R_2(A)$
$W_1(A) - W_2(A)$
$R_1(B) - W_2(B)$
$W_1(B) - R_2(B)$
$W_1(B) - W_2(B)$

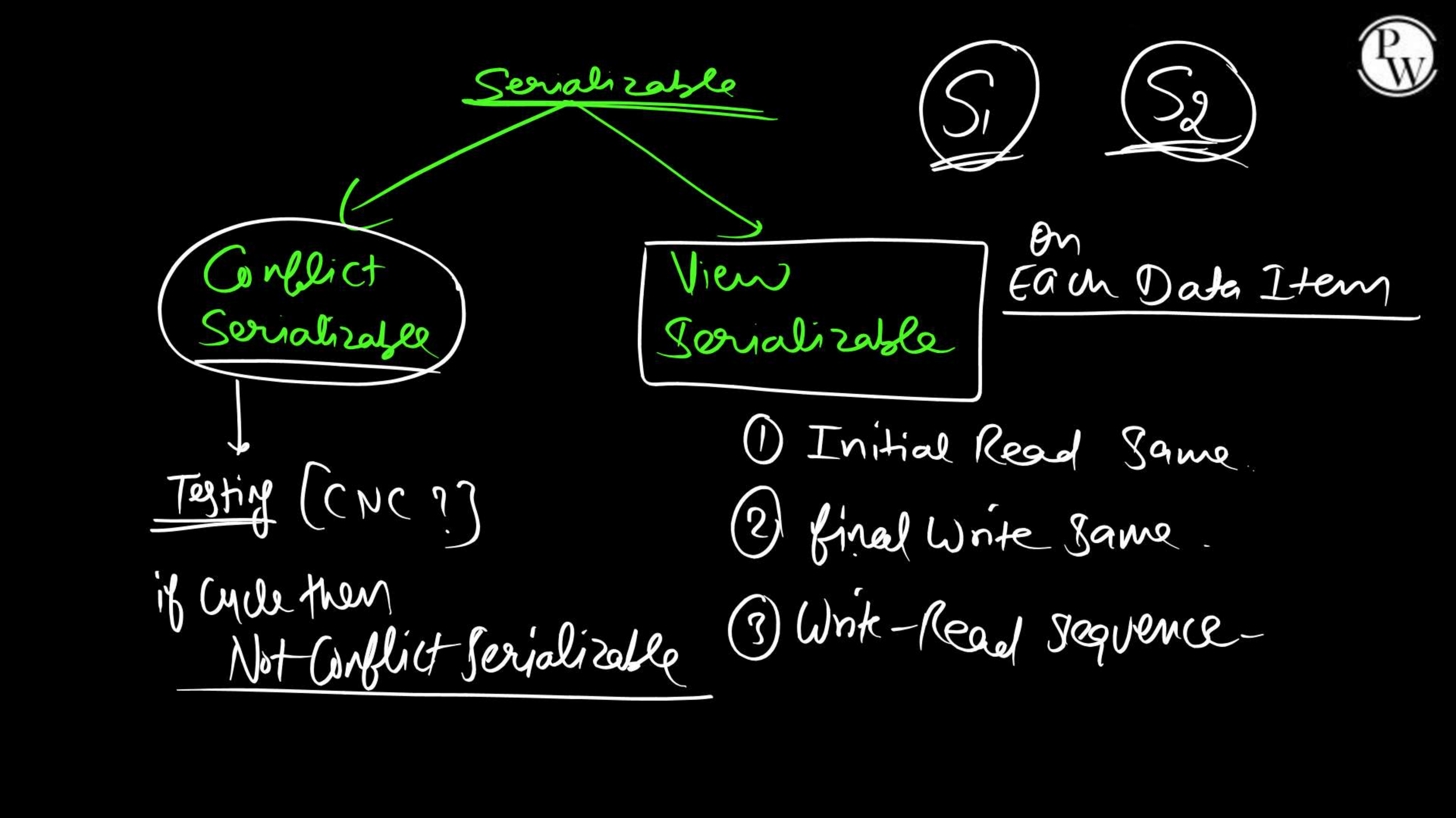
T_1	T_2
read(A)	
write(A)	
	read(A)
	write(A)
read(B)	
write(B)	
	read(B)
	write(B)

C_L

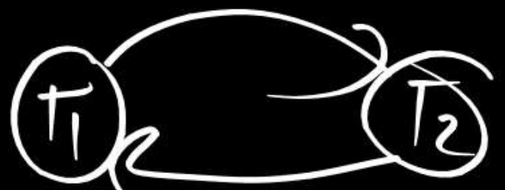
T_1	T_2
read(A)	
write(A)	
read(B)	
write(B)	
	read(A)
	write(A)
	read(B)
	write(B)

S_L

$R_1(A) - W_2(A)$
$W_1(A) - R_2(A)$
$W_1(A) - W_2(A)$
$R_1(B) - W_2(B)$
$W_1(B) - R_2(B)$
$W_1(B) - W_2(B)$


Serializable
S1
S2
Conflict Serializable
View Serializable
On Each Data Item
Testing [CNC ?]
if cycle then Not Conflict Serializable
① Initial Read Same.
② final Write Same.
③ Write-Read sequence-

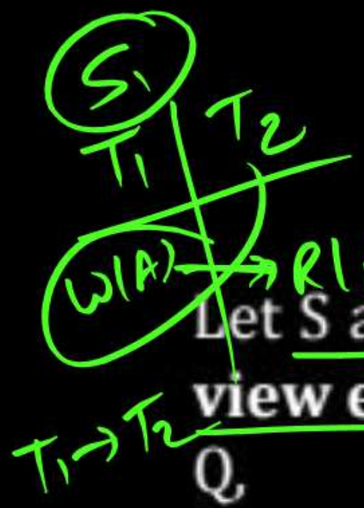
T_1	T_2
R(A)	
	W(A)
R(A)	



Cycle Not Conflict

1. Initial Read
2. final Write
3. Write - Read

View Serializability



Let S and S' be two schedules with the same set of transactions. S and S' are **view equivalent** if the following three conditions are met, for each data item Q,

1. If in schedule S, transaction T_i reads the initial value of Q, then in schedule S' also transaction T_i must read the initial value of Q.
2. If in schedule S transaction T_i executes **read**(Q), and that value was produced by transaction T_j (if any), then in schedule S' also transaction T_i must read the value of Q that was produced by the same **write**(Q) operation of transaction T_j.
3. The transaction (if any) that performs the final **write**(Q) operation in schedule S must also perform the final **write**(Q) operation in schedule S'.

S_1 $\equiv$ VIEW S_2 S'

① Initial Read

② final Write

③ Update Read [Write-Read Sequence]

View Serializability

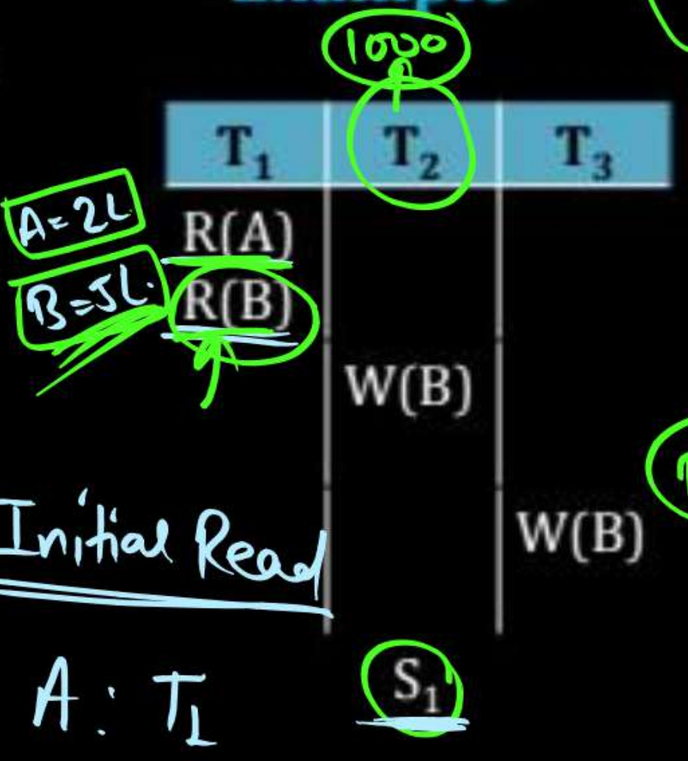
- **View Serializable Schedule:** View equivalent serial schedule.
- **View Equivalent:** S_1 and S_2 said to be view equivalent.

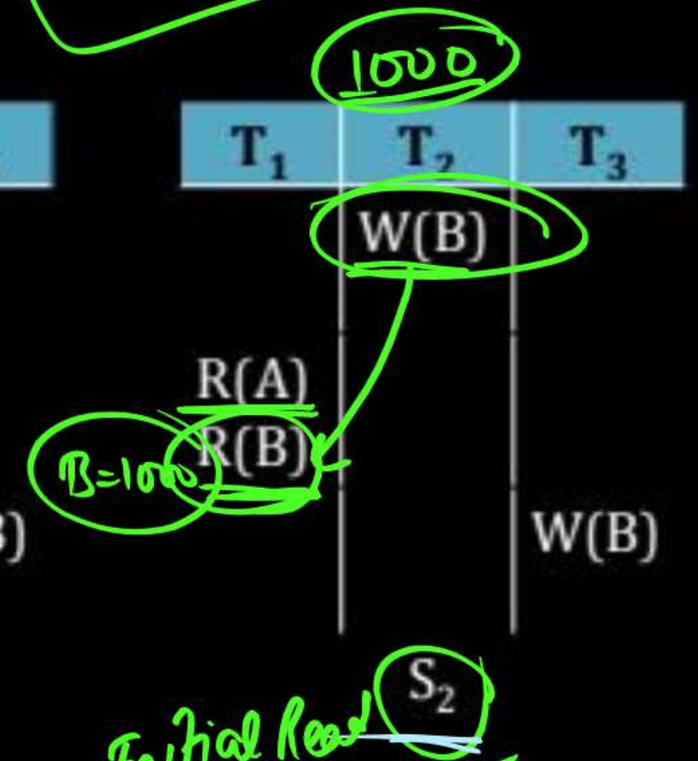
  Only if

  [1] initial reads of S_1 and S_2 should be same.

A & B

Example



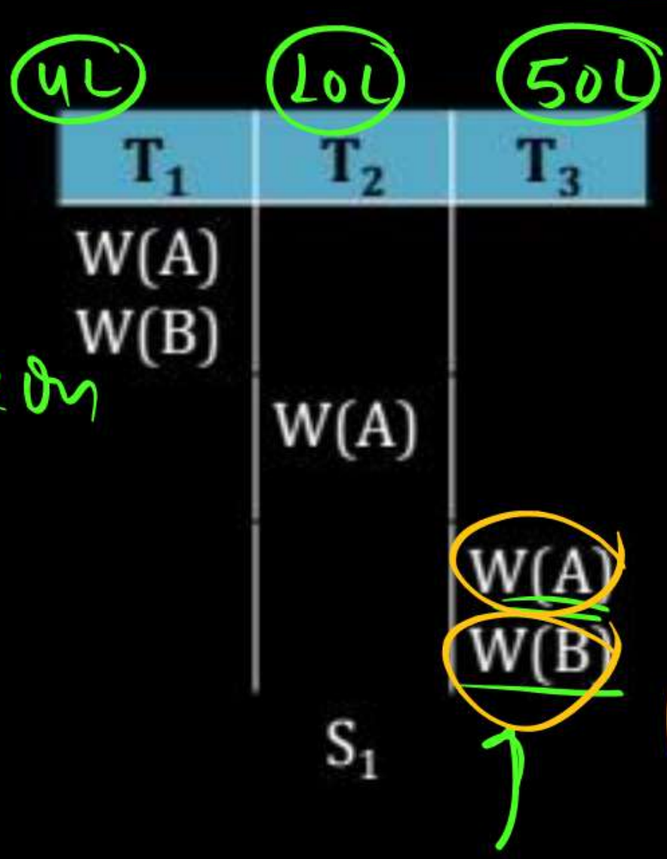


$S_1 \neq S_2$

View Serializability

Q.

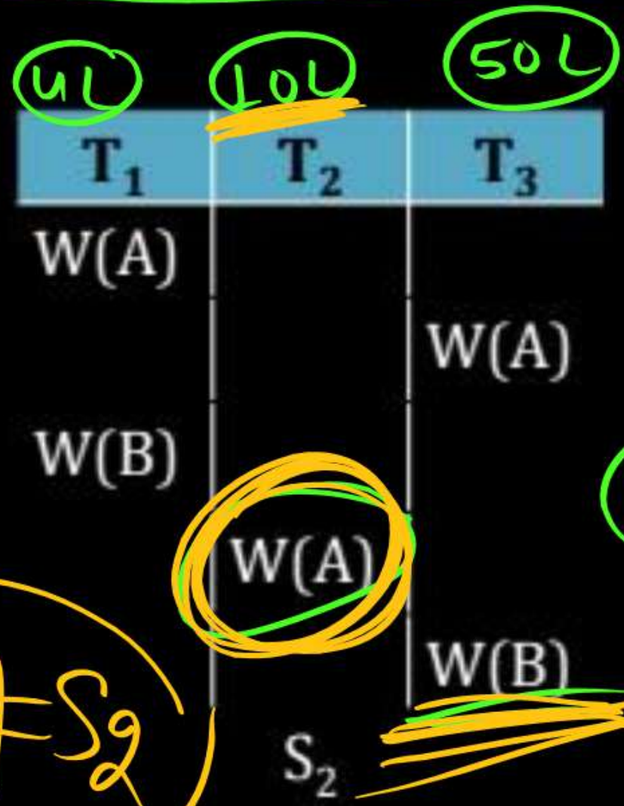
Final updations for every data item should be same in S_1 and S_2



final Write on

A: T3

B: T3



final Write on

A: T2

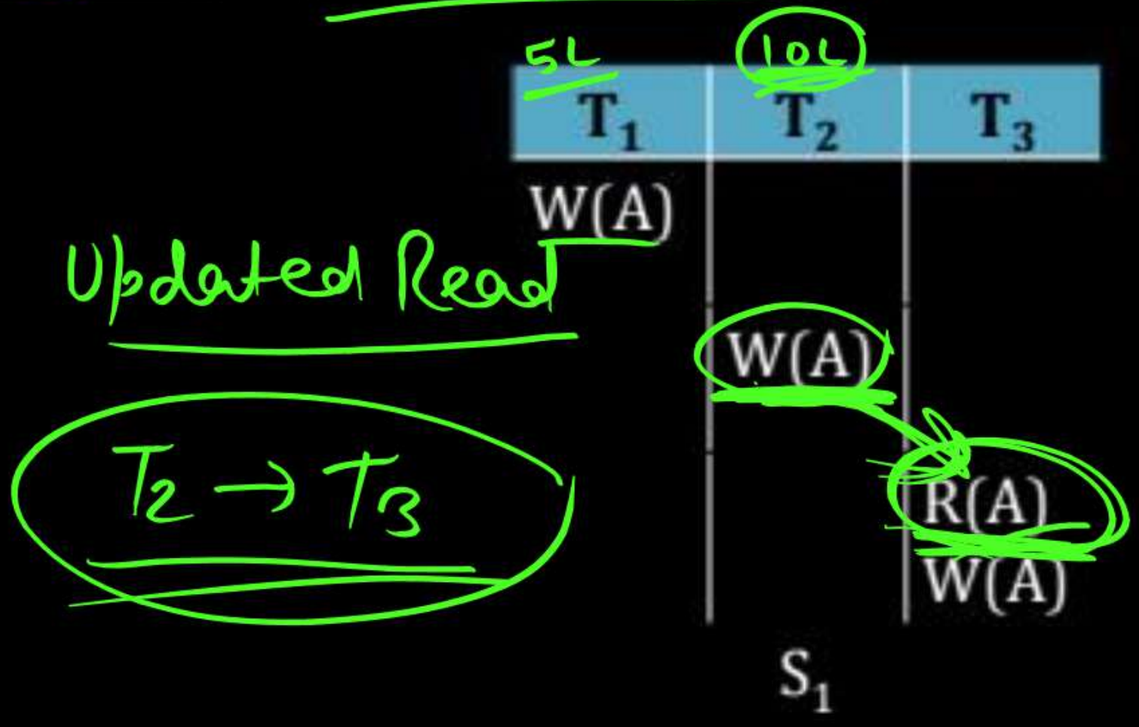
B: T3

$S_1 \neq S_2$

View Serializability

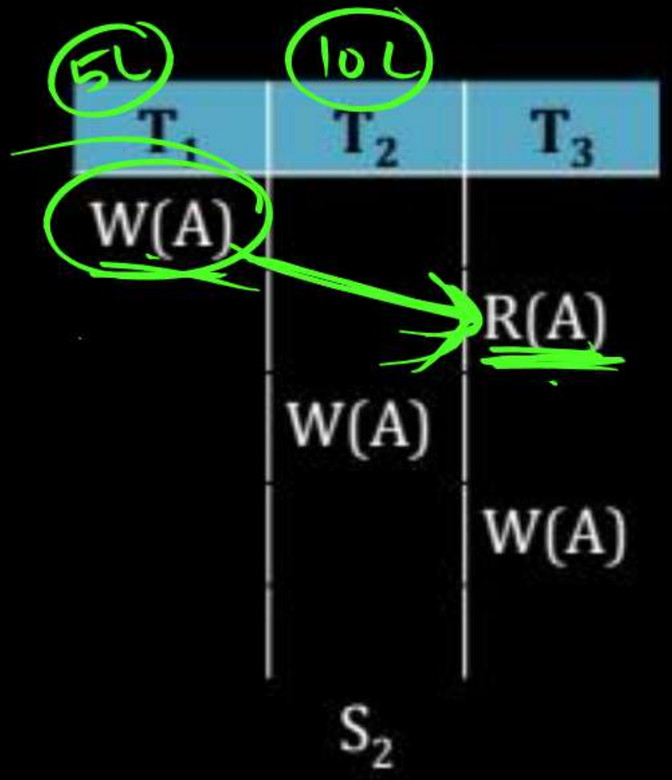
3.

Write-Read sequence should also be equal. (Updated Reads should be same



Updated Read

$T_2 \rightarrow T_3$



Updated Read

$T_1 \rightarrow T_3$

View Serializability (Cont.)

- A schedule S is **view serializable** if it is view equivalent to a serial schedule.
- Every conflict serializable schedule is also view serializable.
- Below is a schedule which is view-serializable but not conflict serializable.

T_{27}	T_{28}	T_{29}
read(Q)		
	write(Q)	
write(Q)		
		write(Q)

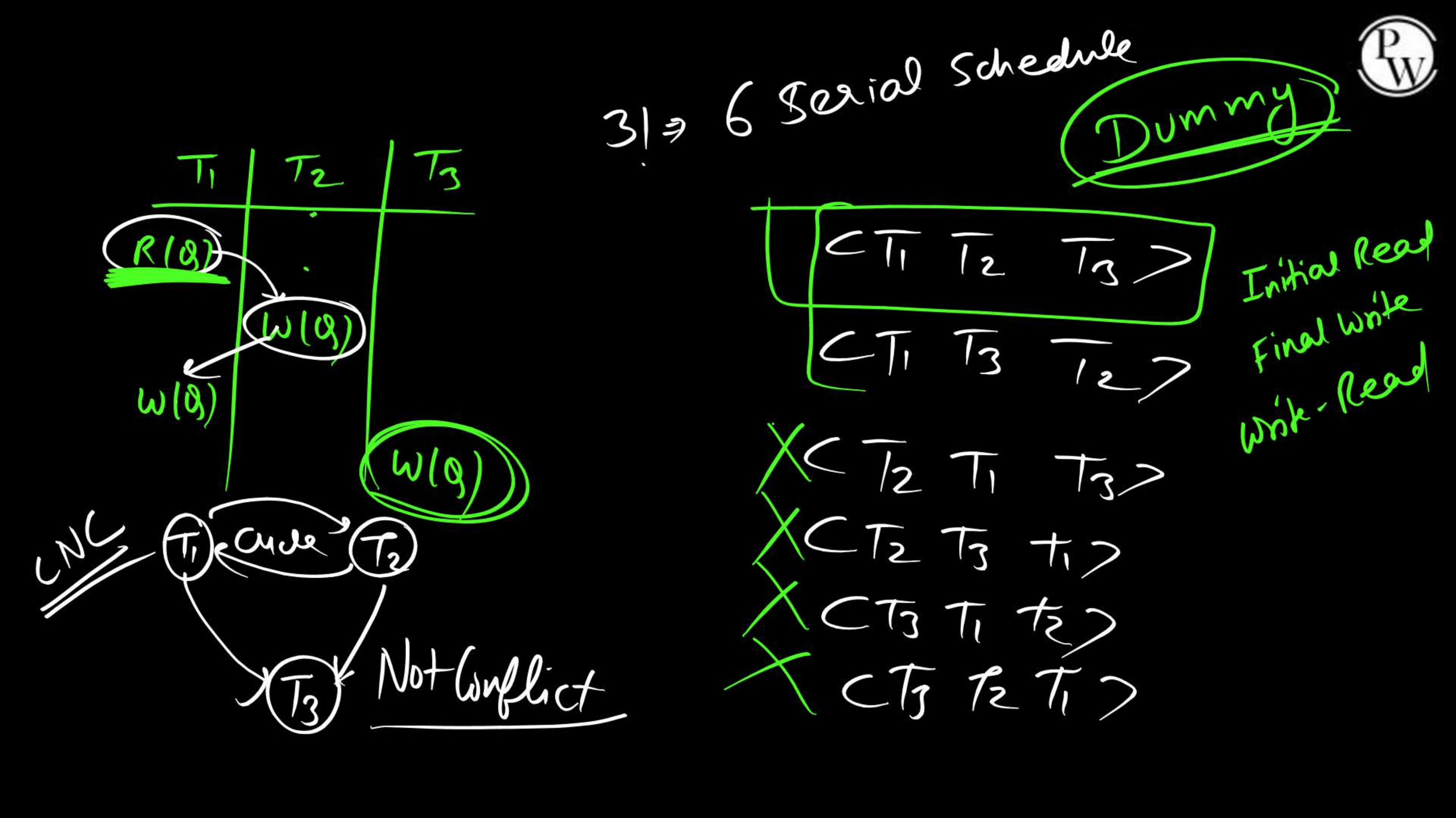
Note:

Every view serializable schedule that is not conflict serializable has **blind writes.**

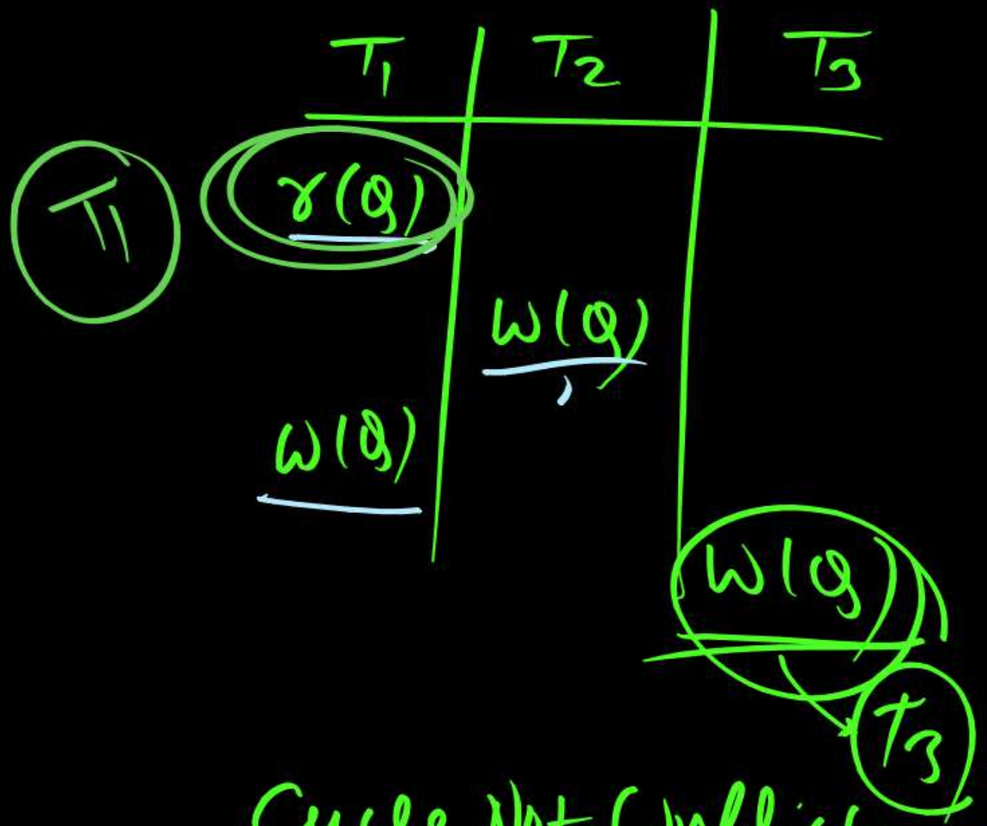
Conflict Serializable

- Testing ⇒ Precedence Graph
  - ⇓
  - CNC

If Conflict ⇒ then Definately its View S.

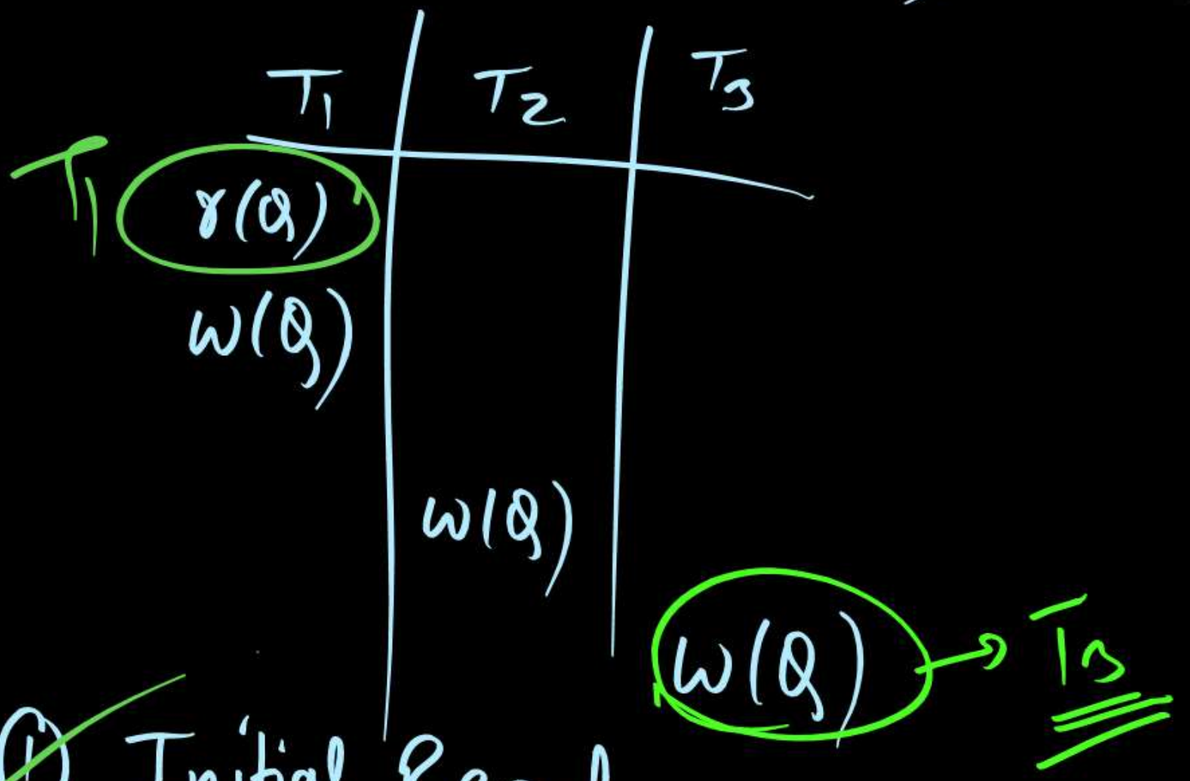

3! ⇒ 6 Serial Schedule
Dummy
T1 T2 T3
R(Q)
W(Q)
W(Q)
W(Q)
CNC
Cycle
T1
T2
T3
Not Conflict
<T1 T2 T3>
<T1 T3 T2>
Initial Read
Final Write
Write - Read
<T2 T1 T3>
<T2 T3 T1>
<T3 T1 T2>
<T3 T2 T1>

S'

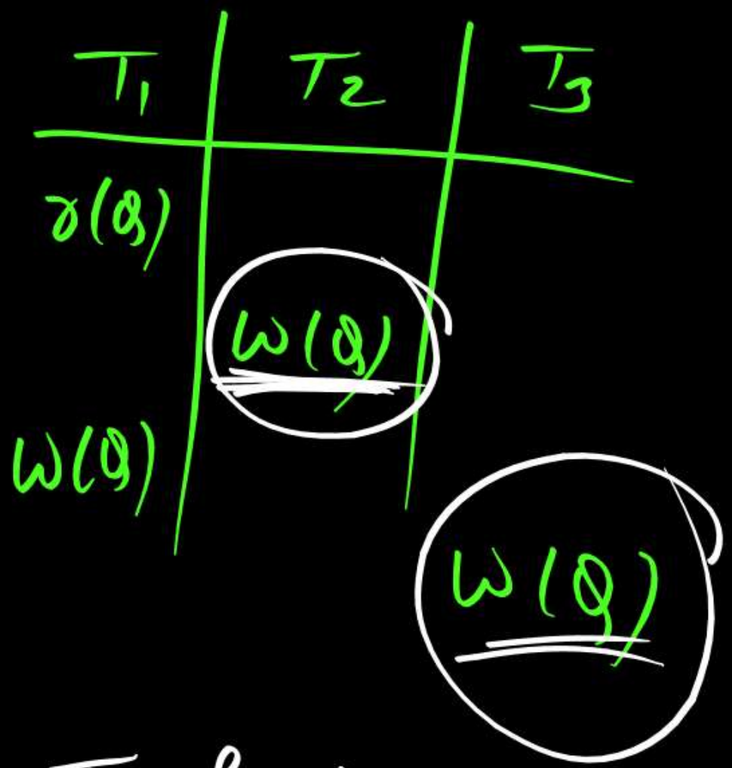


Cycle Not Conflict

S (T1 T2 T3)



1. Initial Read
2. Final Write
3. Write-Read [Updated Read]

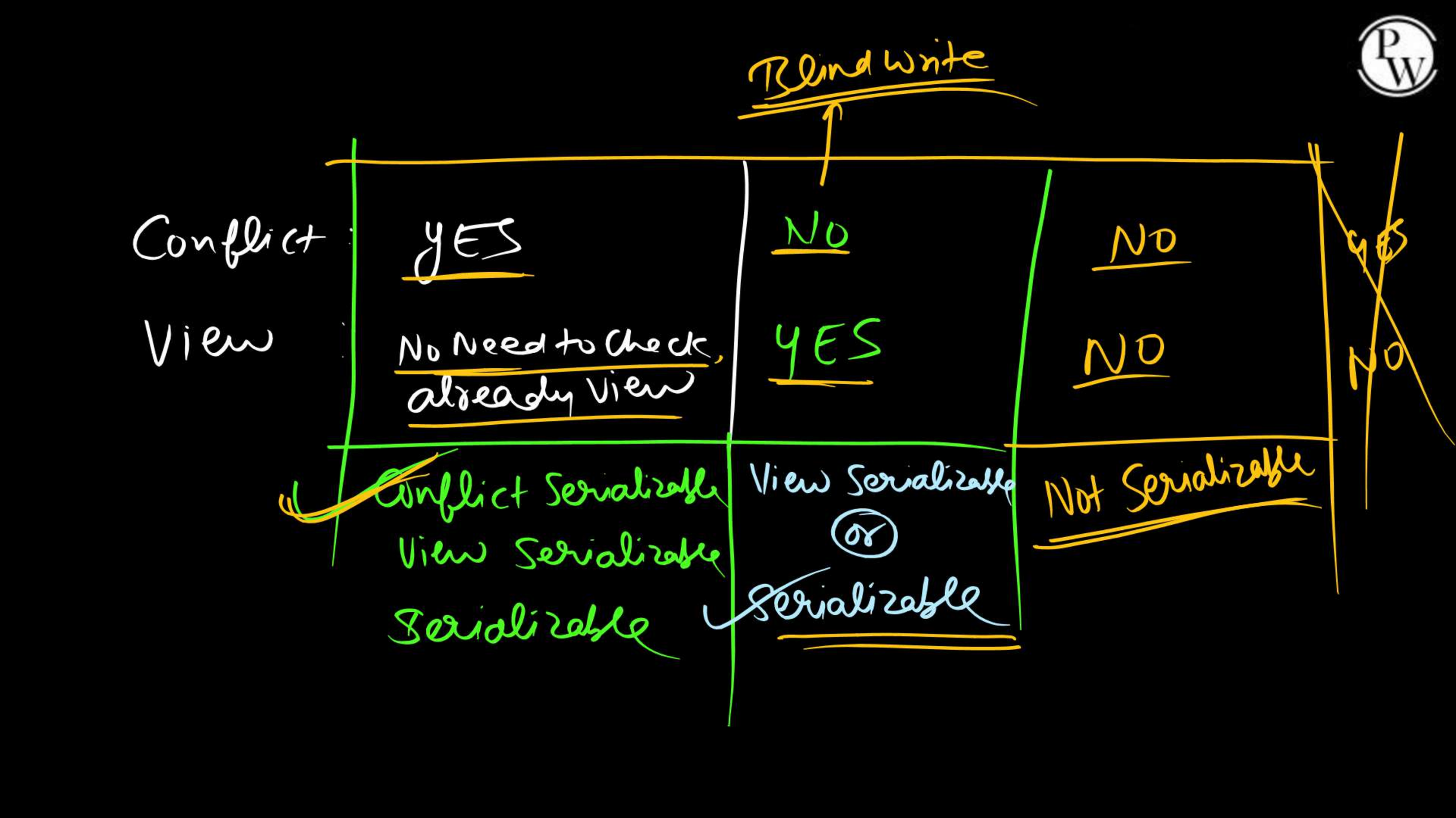


T2 & T3: Blind Write

→ Blind Write
↳ Write without Reading

Not Conflict Serializable
But View Serializable

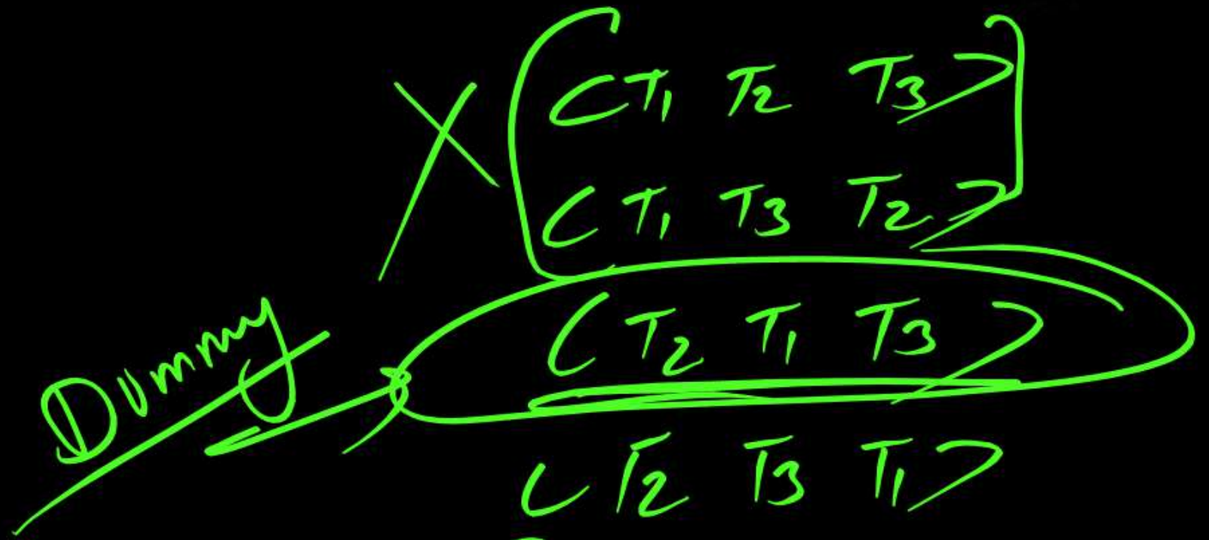
∴ its Serializable Schedule


Blind Write
Conflict
View
YES
NO
NO
No Need to Check, already View
YES
NO
Conflict Serializable
View Serializable
Serializable
View Serializable
(or)
Serializable
Not Serializable
YES
NO

Q.

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

Cycle Not Conflict



$C\langle T_1\ T_2\ T_3\rangle$

$C\langle T_1\ T_3\ T_2\rangle$

$C\langle T_2\ T_1\ T_3\rangle$

$C\langle T_2\ T_3\ T_1\rangle$

$C\langle T_3\ T_1\ T_2\rangle$

$C\langle T_3\ T_2\ T_1\rangle$

Q.

Not Conflict But View Seri Serializable

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

① Initial Read

A : T_2

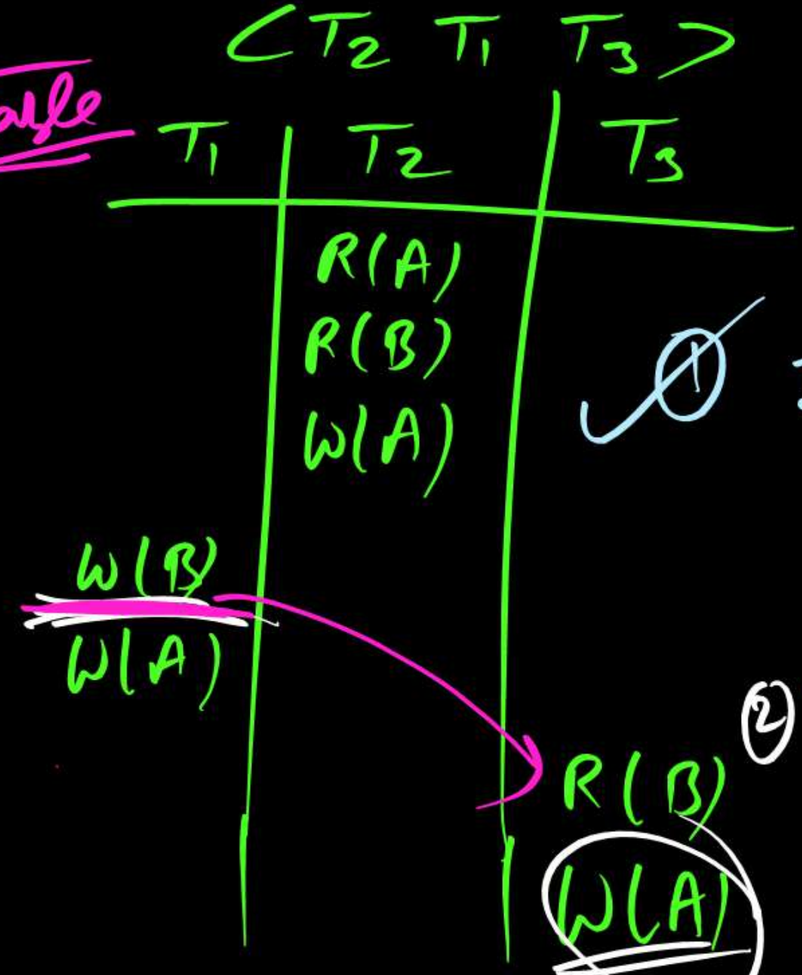
B : T_2

② final Write

A : T_3

B : T_1

③ Update Read $T_1 \rightarrow T_3$



① Initial Read

A : T_2

B : T_2

② final Write

A : T_3

B : T_1

③ Unrelated Read $T_1 \rightarrow T_3$

Q.

Consider the following schedule S of transactions T_1 and T_2:

Which of the following is TRUE about the schedule S? [2004: 2 Marks]

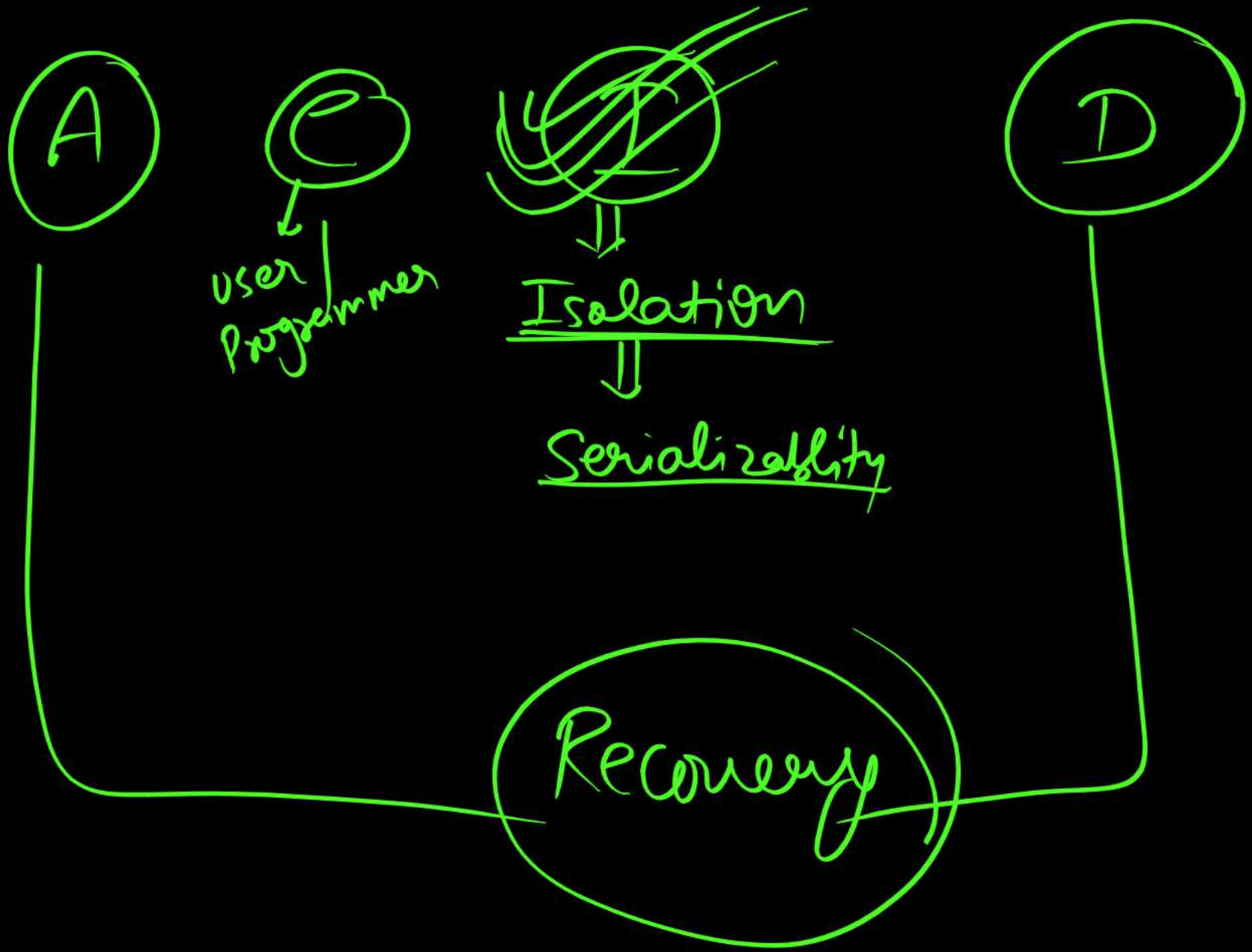
T_1	T_2
Read(A)	
A = A–10	
	Read(A)
	Temp = 0.2 * A
	Write(A)
	Read(B)
Write(A)	
Read(B)	
B = B +10	
Write(B)	
	B = B + Temp
	Write(B)

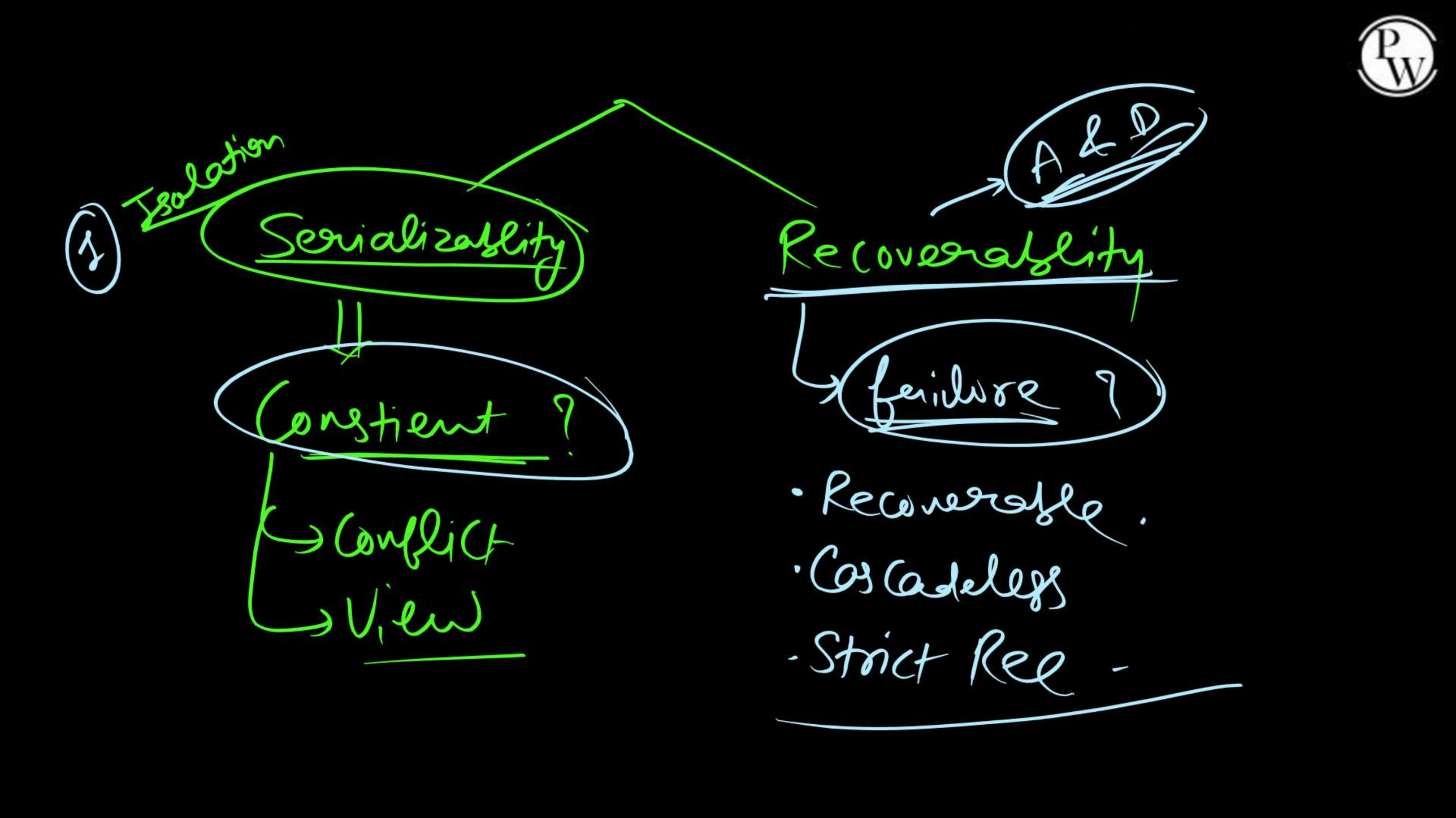
A. S is serializable only as T_1, T_2

B. S is serializable only as T_2, T_1

C. S is serializable both as T_1, T_2 and T_2 T_1

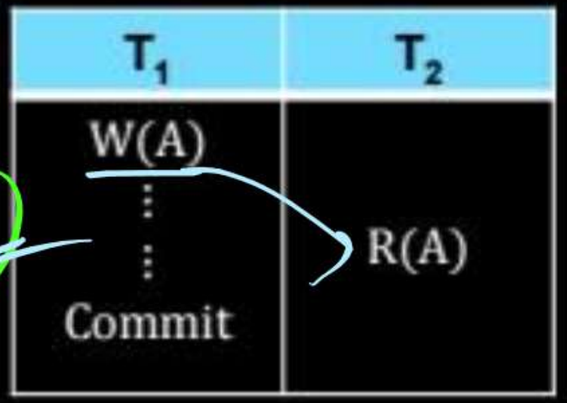
D. S is not serializable either as T_1 or as T_2


A
C
I
D
User Programmer
Isolation
Serializability
Recovery


1
Isolation
Serializability
Consistent ?
Conflict
View
Recoverability
A & D
failure ?
• Recoverable.
• Cascadeless
• Strict Rec -

Dependency

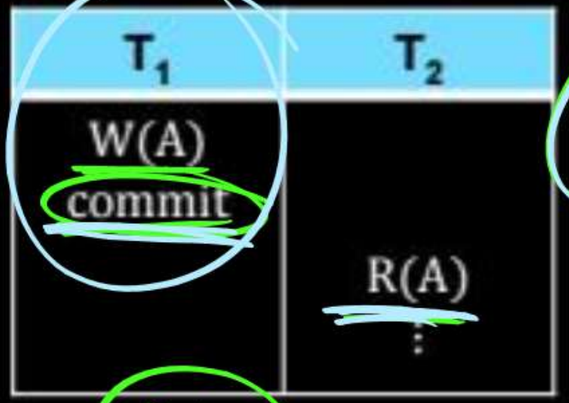
(1)



YES

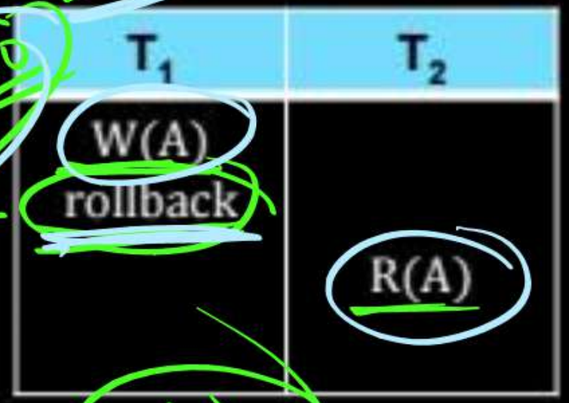
T_2 Depends on T_1

(2)



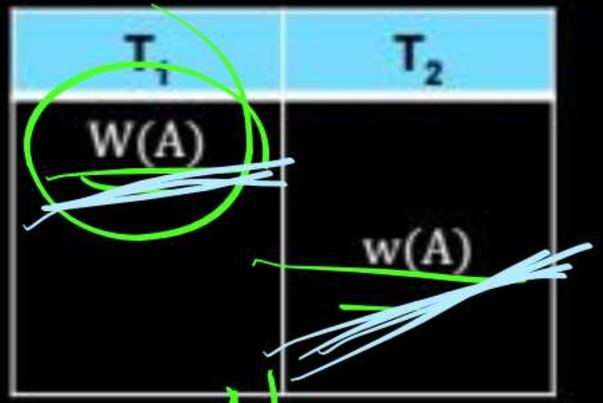
No

(3)



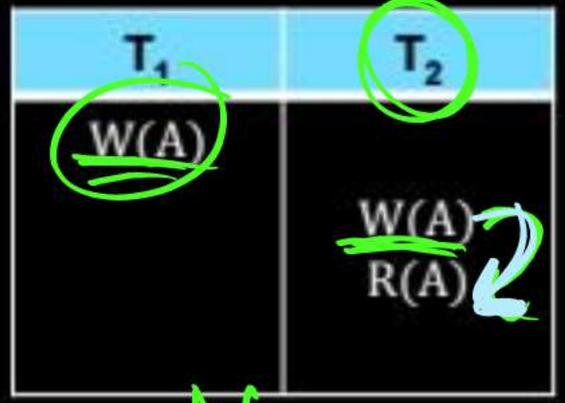
No

(4)



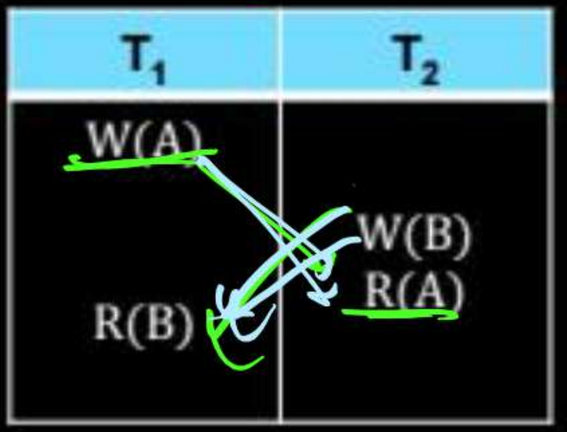
No

(5)



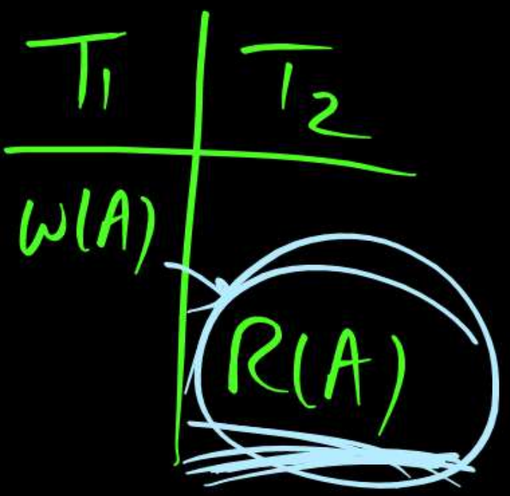
No

(6)

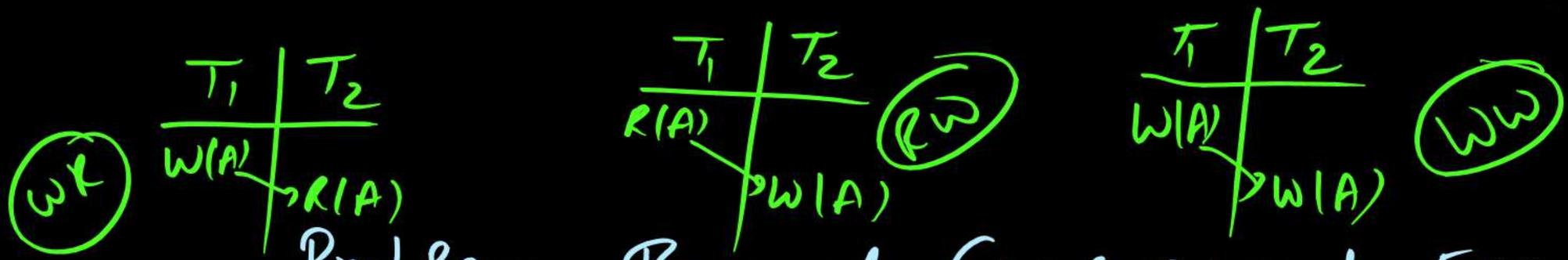


T_2 Dep. T_1
T_1 Dep. T_2

Uncommitted Read / Dirty Read / WR



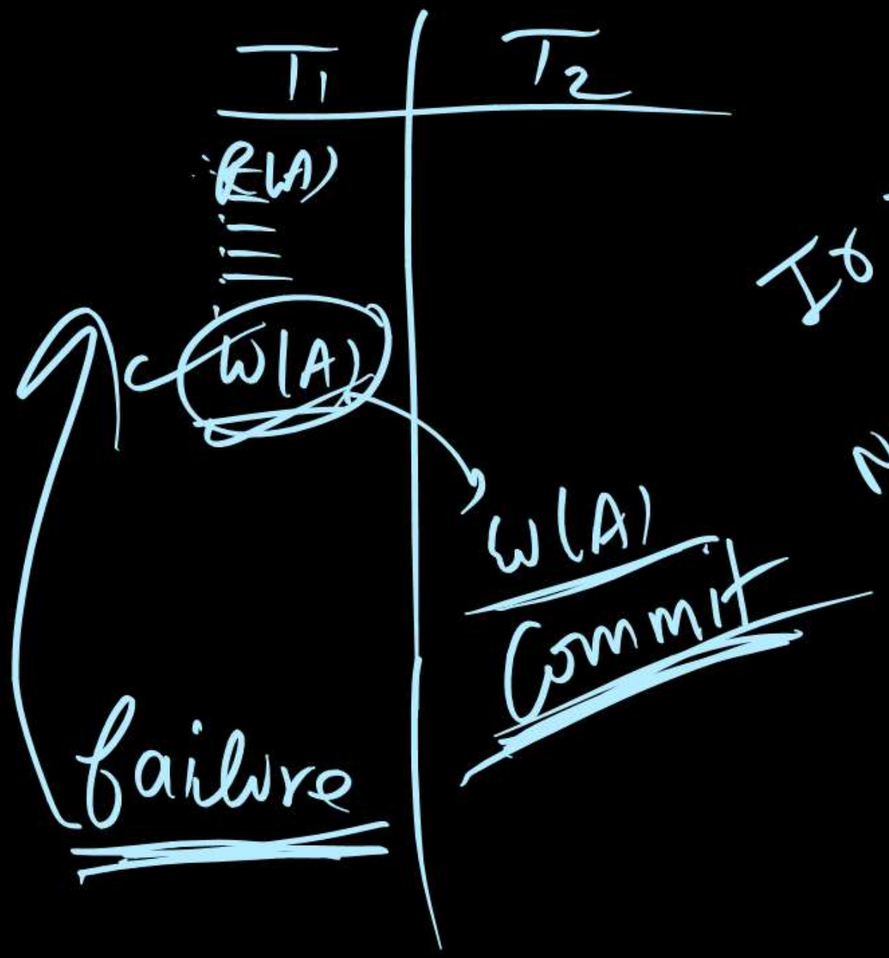
Modify (Updated) by One Transaction & Read by another Transaction



Problem Bcz of Concurrent Execution

Conflict
1. WR (Write - Read) // Dirty Read / Uncommitted Read
2. RW (Read - Write) //
3. WW (Write - Write) / Lost Update Problem
4. Phantom Table Problem

Recoverable Schedule



Is recoverable
or
Non recoverable

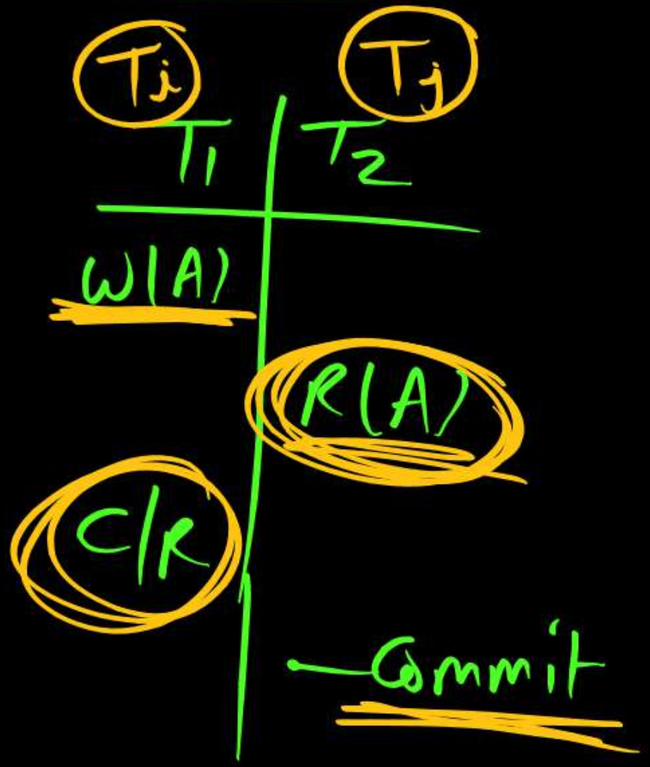
Recoverable Schedule

T_1	T_2
W(A)	
	R(A)
C/R	
	→ Commit

Recoverable

T_1	T_2
R W(A)	
	R(A)
	Commit
*failure	
	Irrecoverable

Recoverable Schedule



If T_2 Depend on T_1, then Commit of T_2 Must be Delayed Untill C/R of T_1.

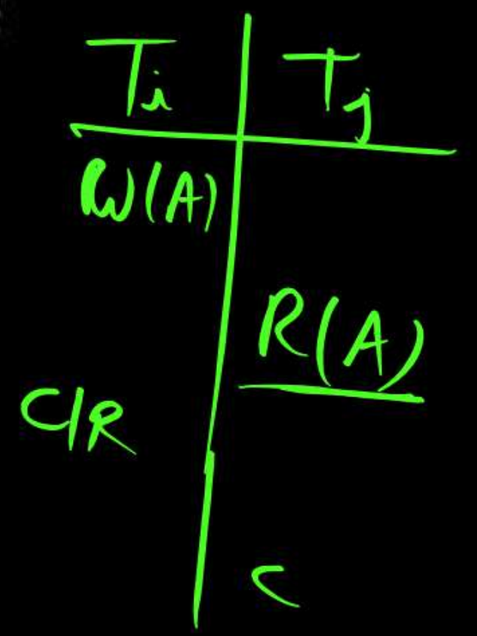
Recoverable Schedules

Need to address the effect of transaction failures on concurrently running transactions.

- Recoverable schedule — if a transaction T_j reads a data item previously written by a transaction T_i, then the commit operation of T_i appears before the commit operation of T_j.
- The following schedule is not recoverable

Irrecoverable

T_8	T_9
read(A)	
write(A)	Read(A)
	commit
read(B)	



- If T_8 should abort, T_9 would have read (and possibly shown to the user) an inconsistent database state. Hence, database must ensure that schedules are recoverable.

Recoverable

T_1	T_2
W(A)	
	R(A)
C/R	
	→ Commit

Examples

(1)

T_1	T_2
w(A)	$R_1(A)$
Rollback	C_2(commit)

Irrecoverable

(2)

T_1	T_2
w(A)	R(A)
C_1	C_2

Non Recoverable

(3)

T_1	T_2
w(A)	
C|R	R(A)

Rec

(4)

T_1	T_2
w(A)	
Commit/ Rollback	w(A) R(A) commit

(5)

T_1	T_2
w(A)	
C|R	R|A rollback

Rec

(6)

T_1	T_2
w(A)	
C|R	R(A)
	C|R

Rec

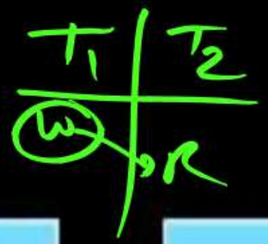
Recoverable Schedule

T_1	T_2
W(A)	
	R(A)
C/R	
	→ Commit

1. WR
2. RW
3. WW
4. Cascading Rollback

NOTE: Recoverable schedule may or may not be free from

- WR problem / uncommitted Read
- RW Problem
- WW Problem



T_1	T_2
R(A)	
	W(A)
	Commit
Commit	

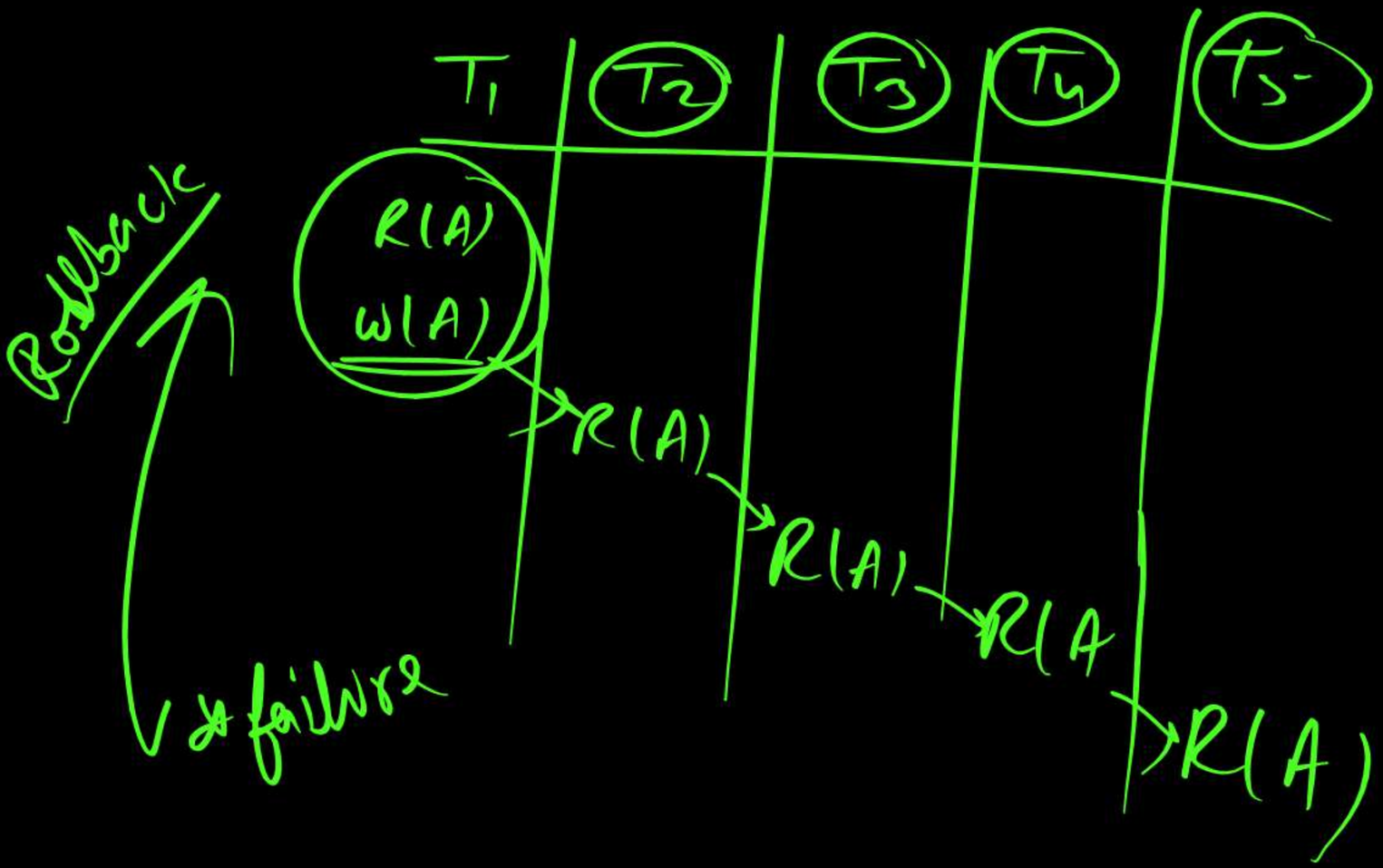
Recoverable
But RW Problem

T_1	T_2
W(A)	
	W(A)
	Commit
Commit	

Recoverable
But WW Problem

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

Recoverable
But WR Problem


T1
T2
T3
T4
T5
R(A)
W(A)
R(A)
R(A)
R(A)
R(A)
Rollback
* failure

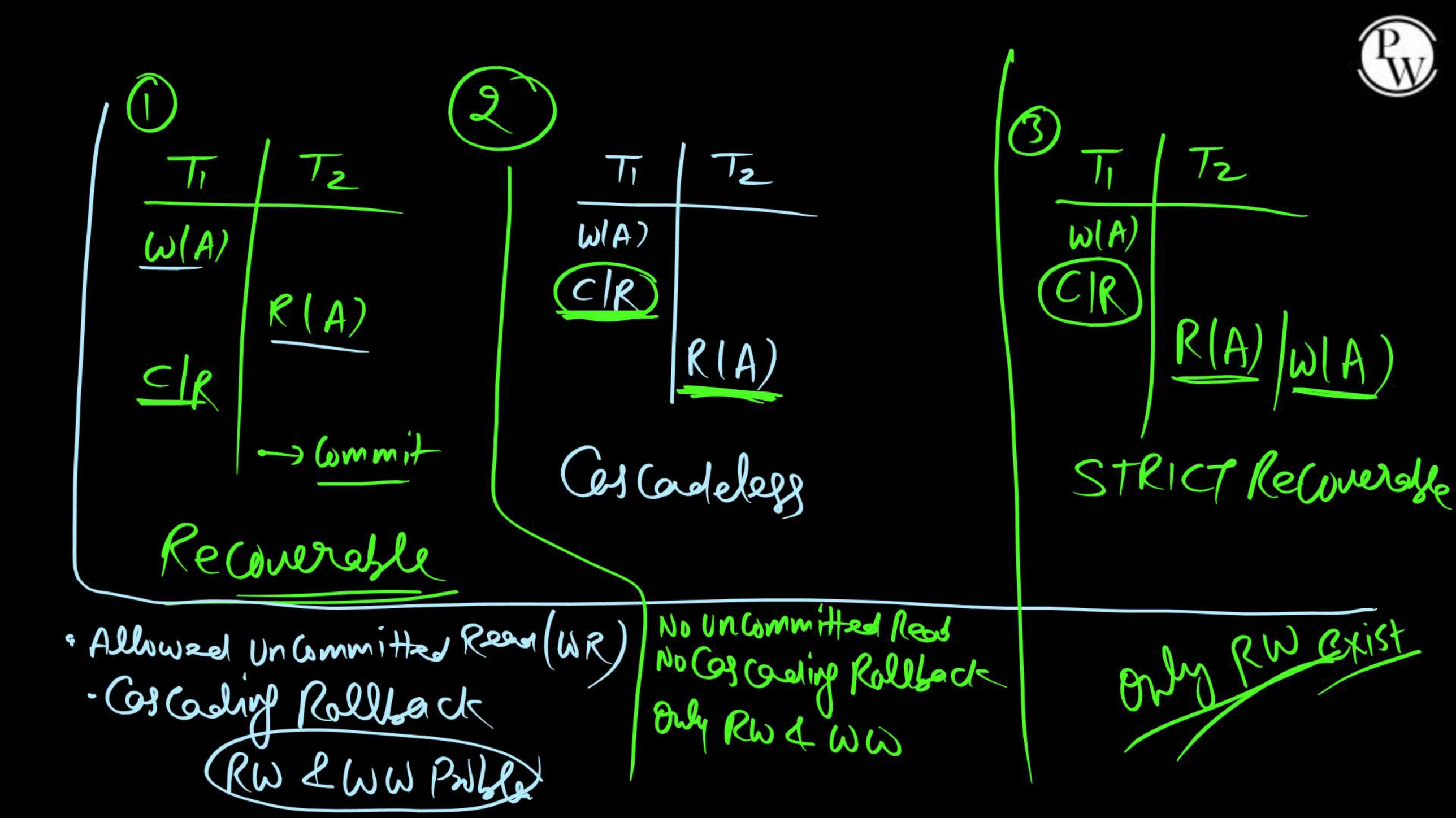
Cascading Rollbacks

- Cascading rollback – a single transaction failure leads to a series of transaction rollbacks. Consider the following schedule where none of the transactions has yet committed (so the schedule is recoverable)

T_{10}	T_{11}	T_{12}
read(A)		
read(B)		
write(A)		
	read(A)	
	write(A)	
		read(A)
abort		

If T_{10} fails, T_{11} and T_{12} must also be rolled back.

- Can lead to the undoing of a significant amount of work

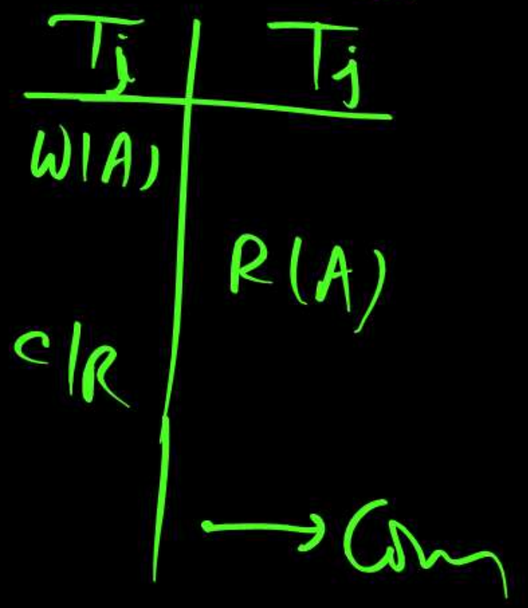

1
T1 | T2
W(A)
R(A)
C/R
→ Commit
Recoverable
2
T1 | T2
W(A)
C/R
R(A)
Cascadeless
3
T1 | T2
W(A)
C/R
R(A)/W(A)
STRICT Recoverable
• Allowed Uncommitted Read (WR)
- Cascading Rollback
RW & WW Possible
No Uncommitted Read
No Cascading Rollback
Only RW & WW
Only RW Exist

Cascadeless Schedules

- Cascadeless schedules — cascading rollbacks cannot occur;
  - For each pair of transactions T_i and T_j such that T_j reads a data item previously written by T_i, the commit operation of T_i appears before the read operation of T_j.
- Every cascadeless schedule is also recoverable

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

Cascadeless Schedule



Strict Recoverable Schedule

T_1	T_2
W(A) C|R	R(A)|w(A)

(1)

T_1	T_2
W(A)	
	R(A)
C|R	
	Commit

Recoverable

- Un Committed Read (WR)
  - RW
  - WW
  - Cascadeless

(2)

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

Cascadeless

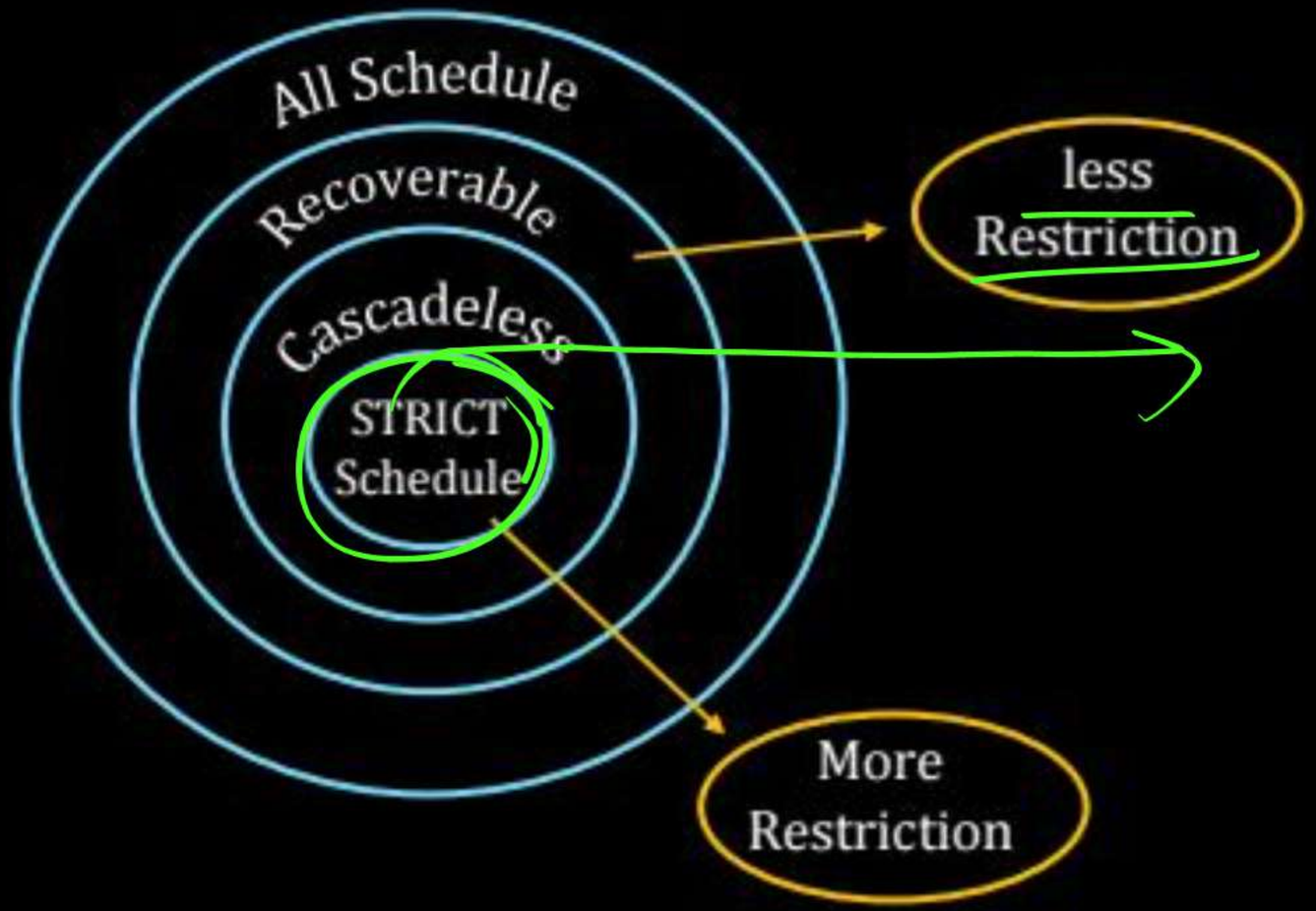
- RW Problem
- WW Problem

(3)

T_1	T_2
W(A)	
C|R	
	R(Q)/W(Q)

Strict schedule

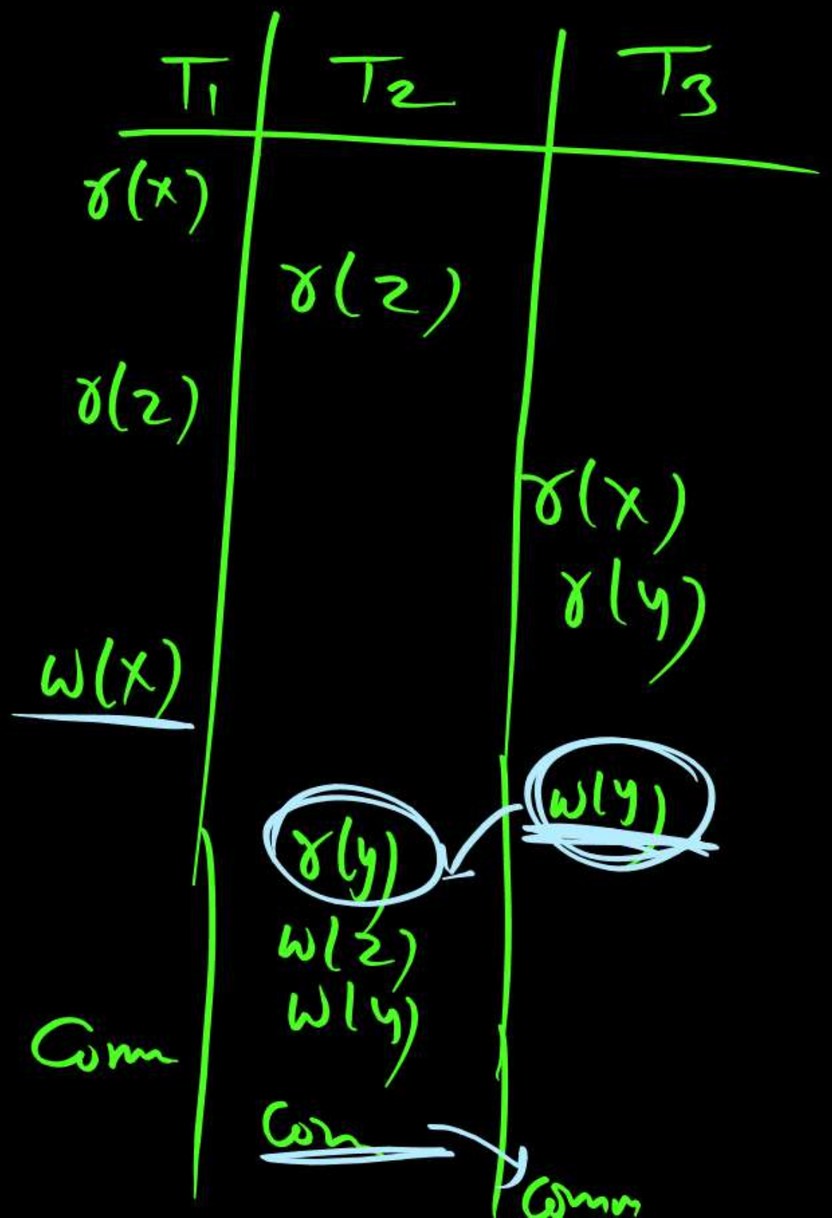
Only RW


All Schedule
Recoverable
Cascadeless
STRICT
Schedule
less
Restriction
More
Restriction

Q. $r_1(x)r_2(z)r_1(z)r_3(x)r_3(y)w_1(x)w_3(y)r_2(y)w_2(z)w_2(y)\ C_1\ C_2\ C_3$

↓

Commit 3



Recoverable

each Data Item
W → R

Irrecoverable

Q. $r_1(x)\ r_2(x) w_1(y) w_2(y) r_2(y)\ C_1\ C_2$.

T_1	T_2
L_1	L_3
L_2	L_4

$$\begin{cases} L_1\ L_2\ L_3\ L_4 \\ L_3\ L_4\ L_1\ L_2 \end{cases}$$

T_1	T_2
0	$\frac{-1}{1}$]
0	

00 11
1100
0101
0110
1010
1001

$L_1\ L_3\ L_2\ L_4$ ⓜ $L_1L_3L_4L_2$

$L_3\ L_1\ L_4\ L_2$ ⓜ $L_3L_1L_2L_4$

Finding Total Number of concurrent Schedule

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

T_1	T_2
L_1 L_2	L_3 L_4

T_1	T_2
0 0	1 1

$L_1L_2L_3L_4$

$L_3L_4L_1L_2$

} Serial

$L_1L_3L_2L_4$ (or) $L_1L_3L_4L_2$

$L_3L_1L_4L_2$ (or) $L_3L_1L_2L_4$

} Non Serial

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

$S_1 <T_1T_2>$

(1)

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

$S_2 <T_2T_1>$

(2)

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

(3)

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

(4)

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

(5)

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

(6)

Concurrent Schedule $\Rightarrow \frac{(n_1+n_2)!}{(n_1)!\,(n_2)!}$

- Serial
- Non Serial

$m!$ (m : # of Transaction)

Total # Concurrent Schedule $= \frac{(n_1+n_2)!}{(n_2)!(n_2)!}$

$T_1 \rightarrow n_1$ operation / 2 operation

$T_2 \rightarrow n_2$ operation / 2 operation

$= \frac{(2+2)!}{(2)!(2)!} = \frac{4\times3\times2}{2\times2} = 6$

Total Concurrent = 6

Total non serial Schedule = Total Concurrent - Serial schedule(m!)

m: # of transaction

$2! = 2$

$= 6 - 2$

Serial = 2

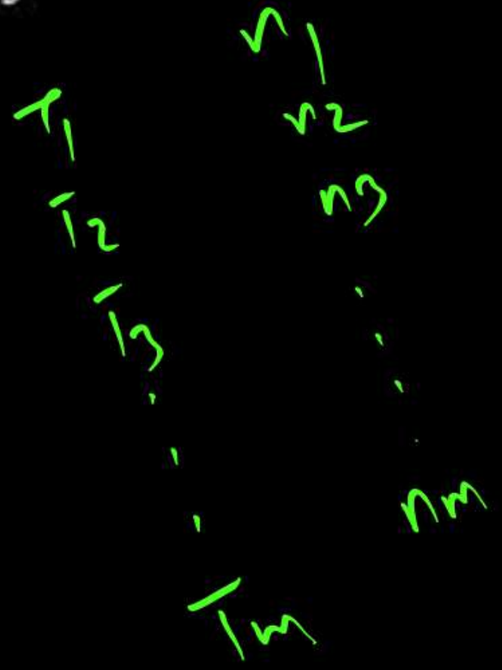
Total non Serial = 4

NOTE:

The Number of Concurrent schedule that can be formed
Over m transaction having n_1 n_2 n_3 ....n_m operation respectively

Total # of Concurrent Schedule	$= \frac{(n_1+n_2+n_3+\cdots n_m)!}{(n_1!)(n_2!)(n_2!)\ldots(n_m!)}$

Total # of Non Serial Schedule	$= \frac{(n_1+n_2+n_3+\cdots n_m)!}{(n_1!)(n_2!)(n_2!)\ldots(n_m!)} - m!$



Q. Find total number of Concurrent & non serial schedule over these transaction T_1 T_2 & T_3 having 2, 3 & 4 operation respectively?

Total # of Concurrent Schedule $= \frac{(2+3+4)!}{(2!)(3!)(4!)} = \frac{9!}{2!3!4!} = \frac{9\times8\times7\times6\times5\times4!}{2\times3\times2\times4!} = 1260$

Non Serial = 1260 - [3!]

Non serial = 1254

3 transaction Serial schedule $= 3! = 3\times2 = 6$

1260 − 6 = 1254

3! = 6

THANK YOU!