COMPUTER SCIENCE

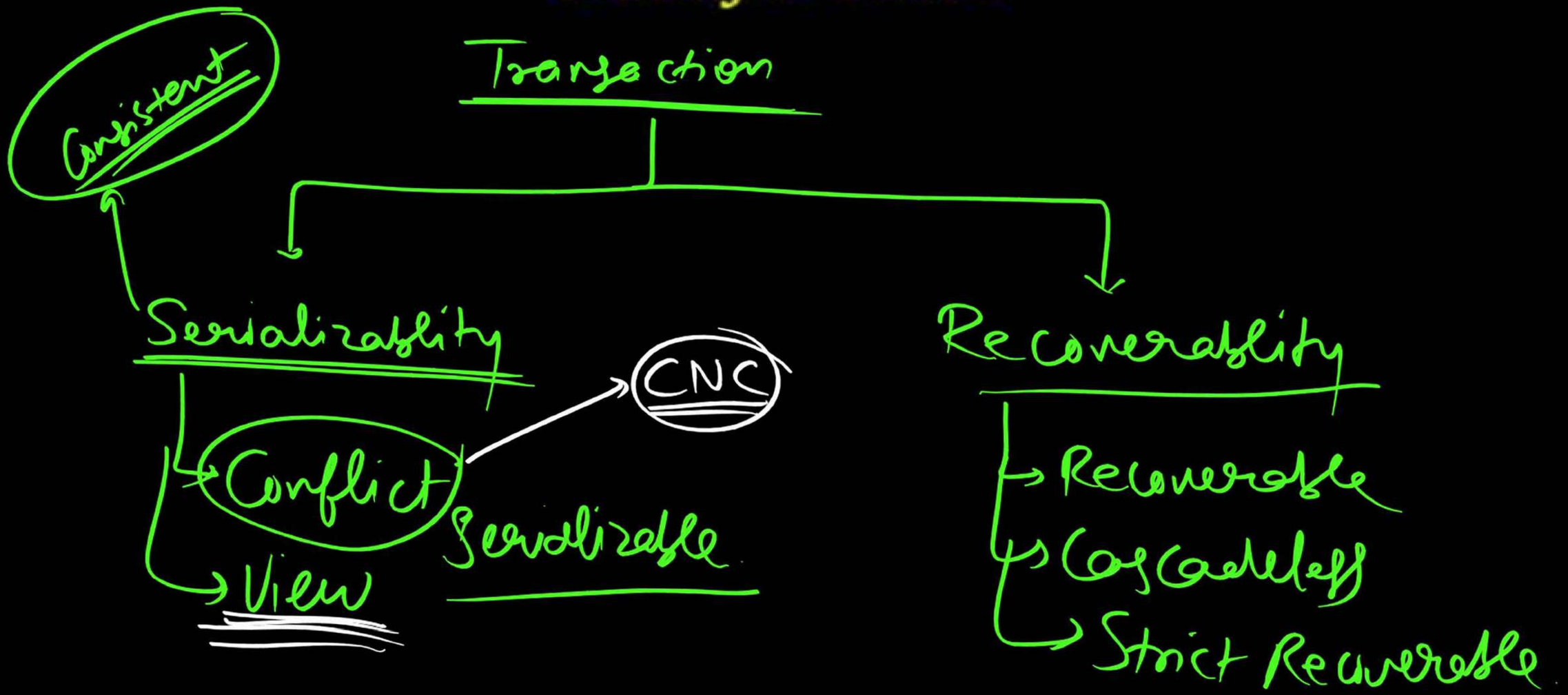
Database Management System

Query Language

Lecture_05

Vijay Agarwal sir

Today's GOAL


Consistent
Transaction
Serializability
CNC
Conflict
Serializable
View
Recoverability
Recoverable
Cascadeless
Strict Recoverable

Implementation of Concurrency Control

Lock Based Protocol

- SHARED [S]
- Exclusive [X]

SHARED Lock [S]

⇓

'Only Read'

Syntax:

```
Lock-S(A)
Read(A)
Unlock-S(A)
```

Exclusive Lock [X]

⇓

Write | Write Read | Read Write

Syntax:

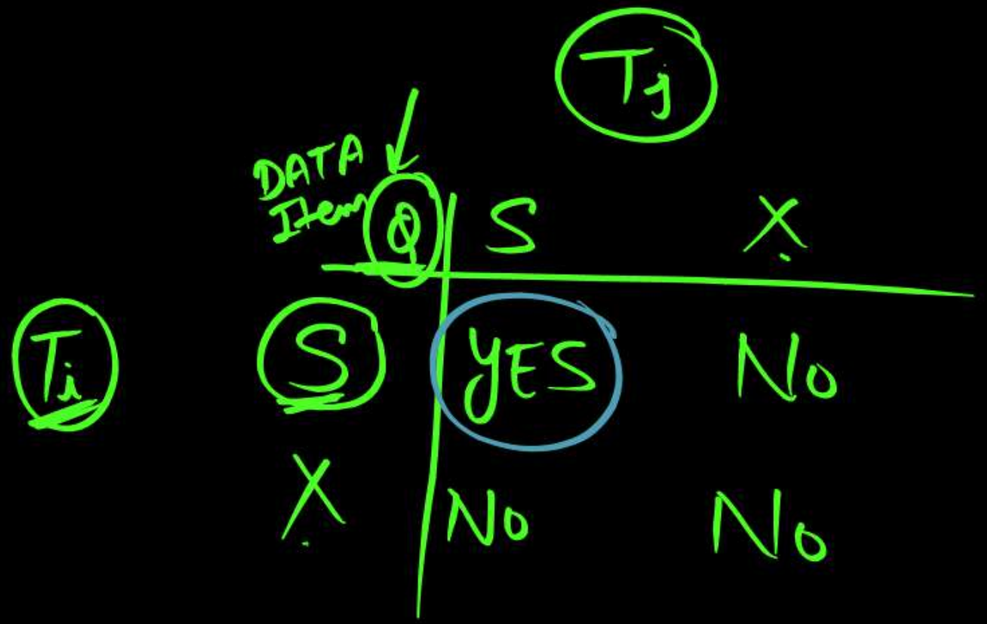
```
Lock-X(A)
Write(A) | R(A) W(A) | W(A) R(A)
Unlock-X(A)
```

Lock-Based Protocols

- A lock is a mechanism to control concurrent access to a data item
- Data items can be locked in two modes:

  1. exclusive (X) mode. Data item can be both reads as well as written. X-lock is requested using **lock-X** instruction.
  2. Shared (S) mode. Data item can only be read. S-lock is requested using **lock-S** instruction.

- Lock requests are made to concurrency-control manager. Transaction can proceed only after request is granted.


Tj
DATA Item
Q
S
X
Ti
S
YES
No
X
No
No

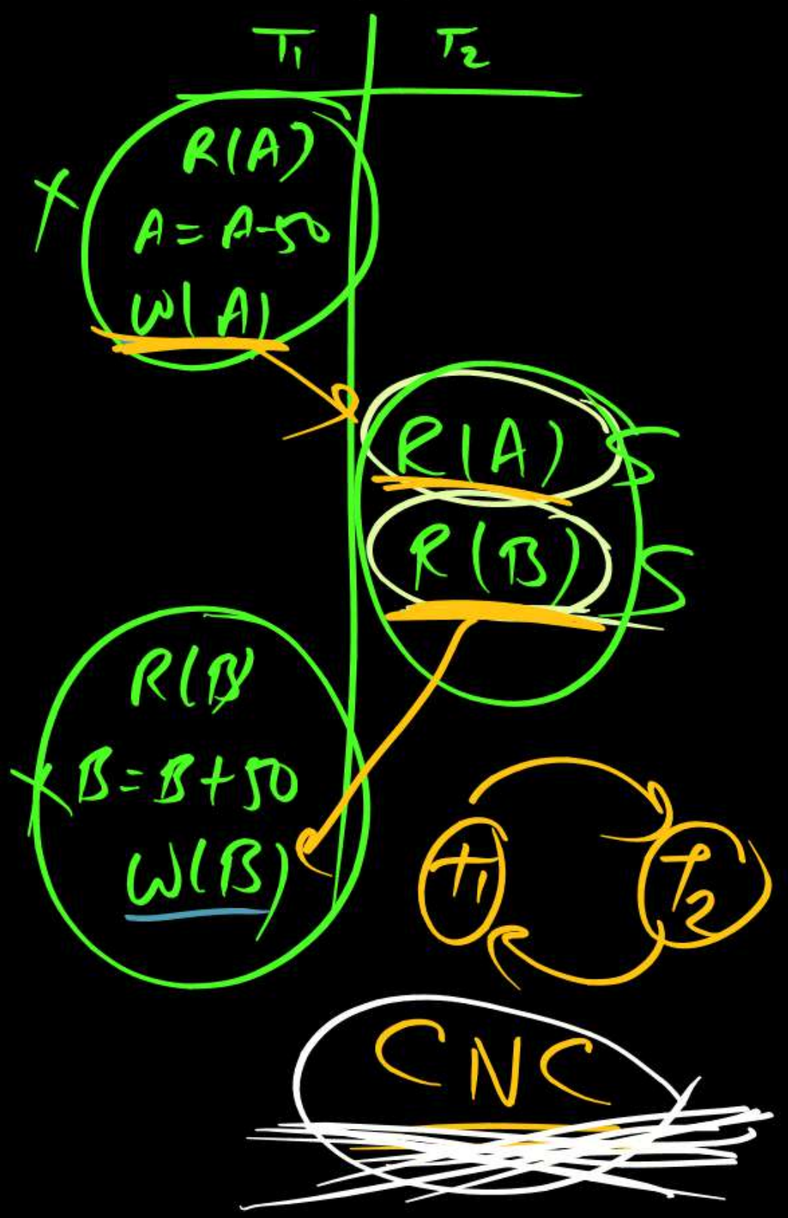
Lock-Based Protocols (Cont.)

- Lock-compatibility matrix

	S	X
S	true	false
X	false	false

- A transaction may be garneted a lock on an item if the requested lock is compatible with lock already held on the item by other transactions
- Any number of transactions can hold shared locks on an item
- But if any transaction holds an exclusive on the item no other transaction may hold any loc on the item.

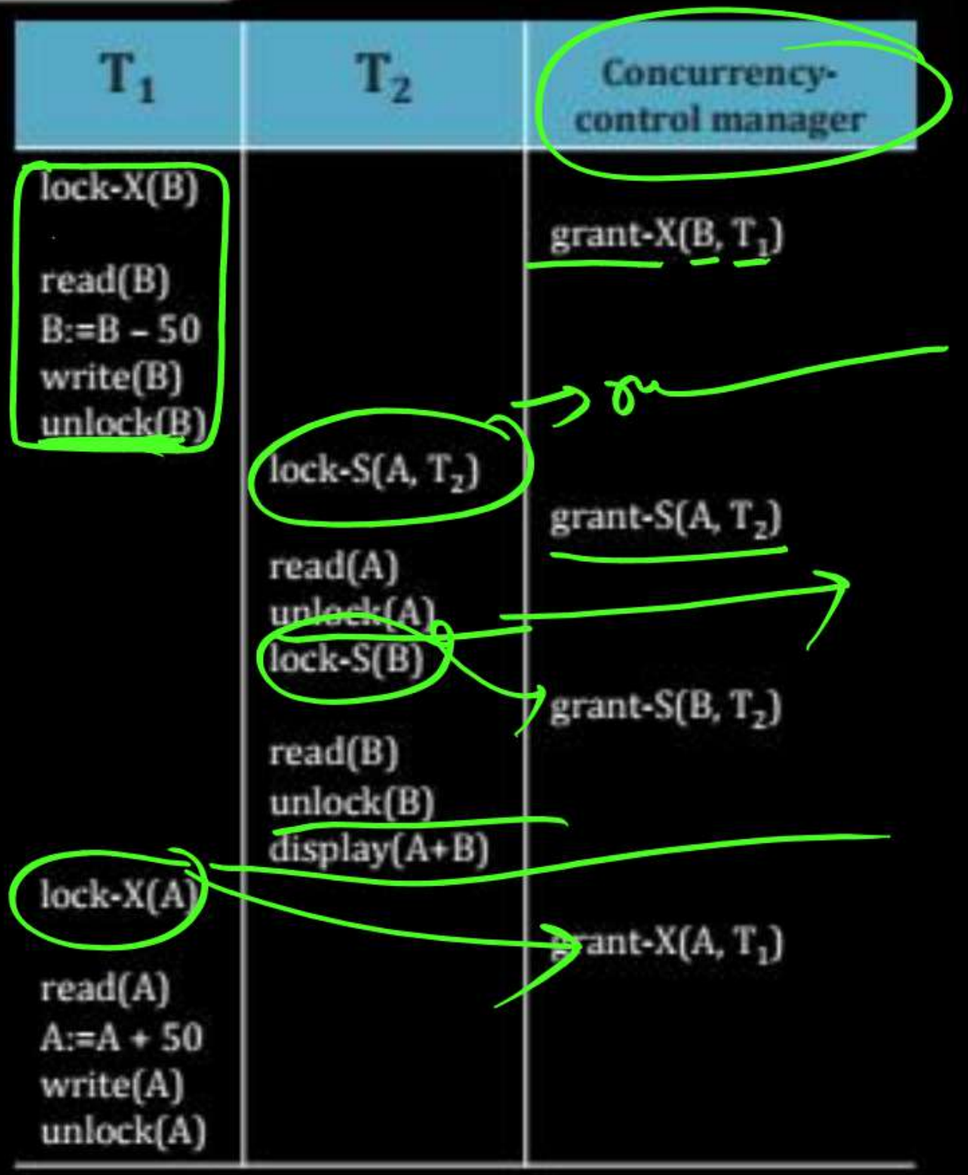
LOCK



T_1	T_2	
Lock-X(A)		Grant-X(A, T_1)
R(A)		
A=A-50		
W(A)		
Unlock-X(A)		Release/Revoke-X(A, T_1)
	Lock-S(A)	grant-S(A, T_2)
	R(A)	
	Unlock-S(A)	release-S(A, T_2)
	Lock-S(B)	grant-S(B, T_2)
	R(B)	
	Unlock(B)	release-S(B, T_2)
Lock-X(B)		grant-X(B, T_1)
R(B)		
B+50		
Write(B)		revoke-X(B, T_1)
Unlock-X(B)		

Schedule with Lock Grants

- A locking protocol is a set of rules followed by all transactions while requesting and releasing locks.
- Locking protocols enforce serializability by restricting the set of possible schedules.



2PL [2 Phase Locking Protocol]

Each Transaction Issue Lock & Unlock Request in Two Phases.

1. Growing Phase [Acquire the lock but must NOT Release Any lock]
2. Shrinking Phase [Release the lock but must Not ask for Any New lock]

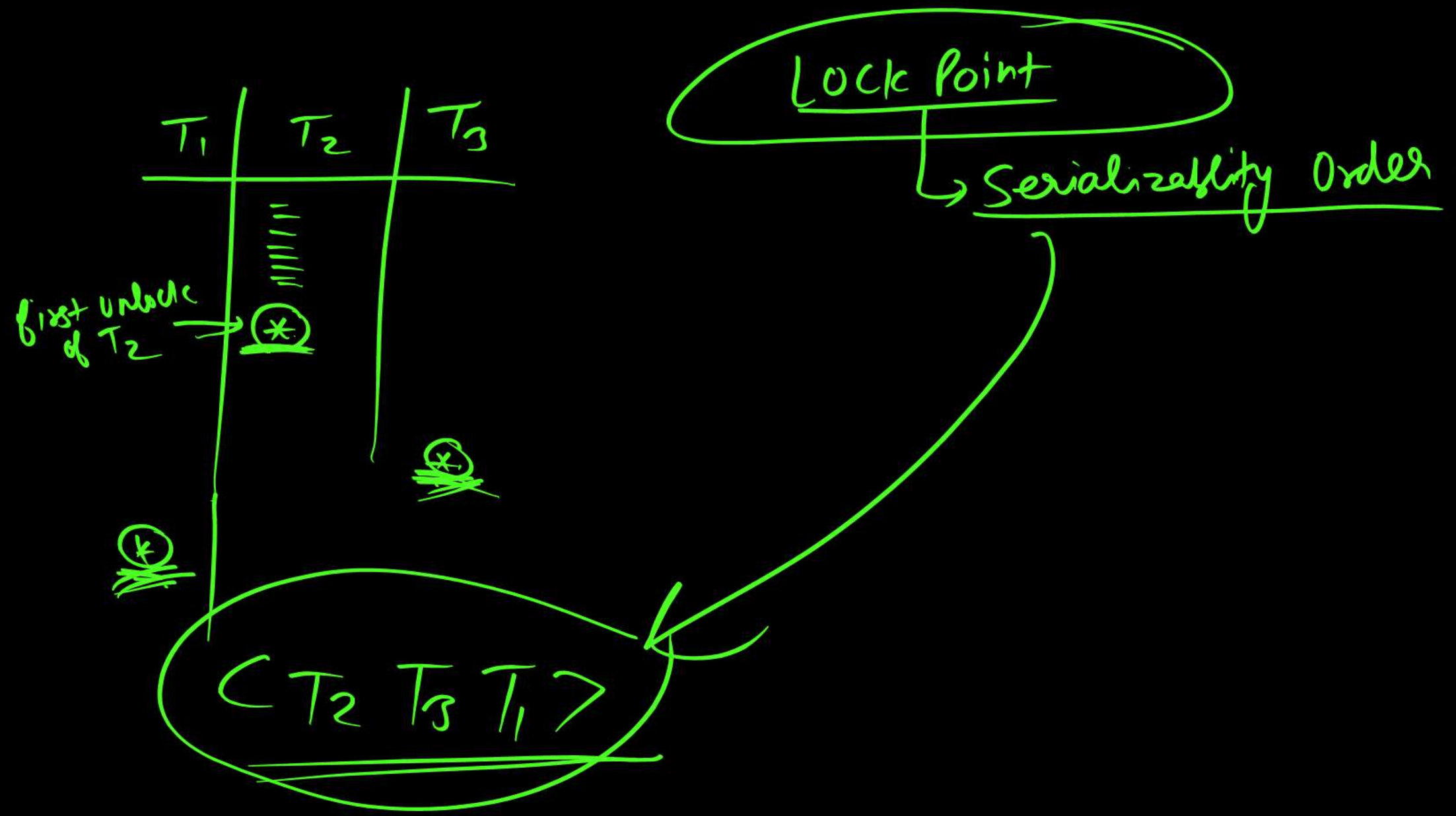
Note

Each Transaction finish first its Growing Phase then Shrinking Phase.

Lock Point : from where Shrinking Phase start.

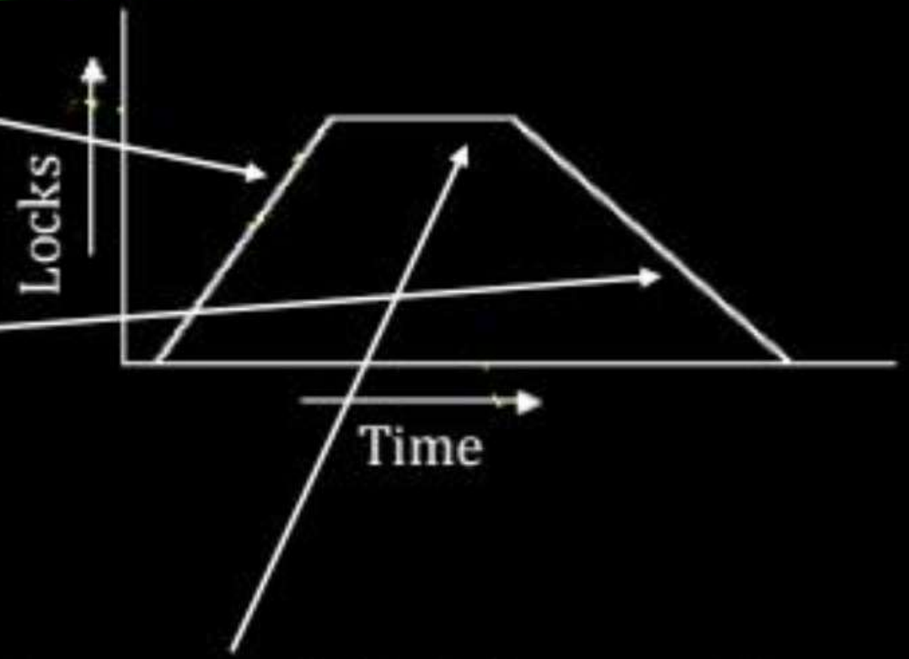
OR

Position of last lock OR First Unlock operation.

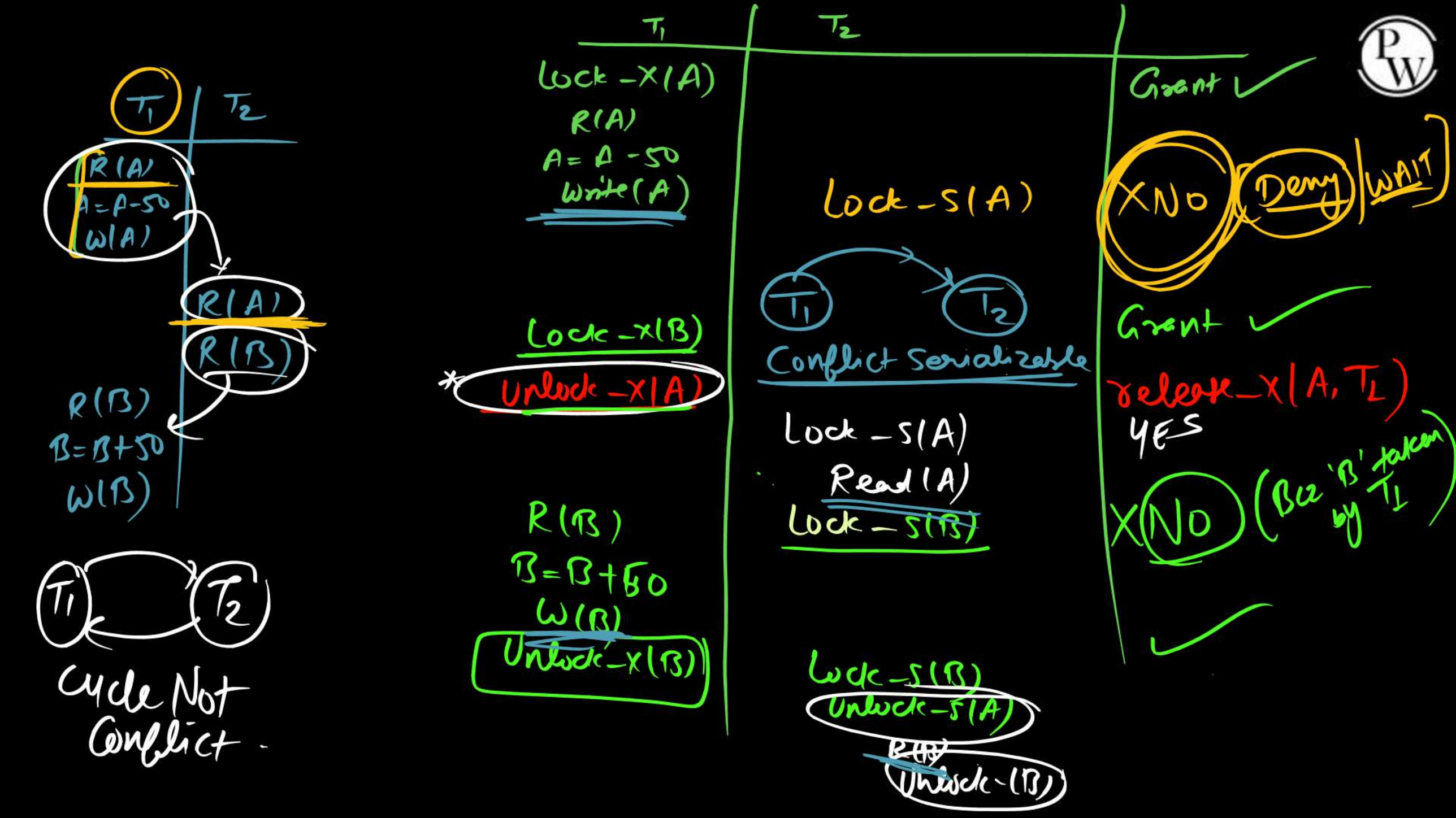

Lock Point
Serializability Order
T_1
T_2
T_3
first unlock of T_2
$\langle T_2\ T_3\ T_1 \rangle$

The Two-Phase Locking Protocol

- A protocol which ensures conflict-serializable schedules.
- Phase 1: Growing Phase
  - Transaction may obtain locks
  - Transaction may not release lock
- Phase 2: Shrinking Phase
  - Transaction may release locks
  - Transaction may not obtain locks



- The protocol assures serializability. It can be proved that transactions can be serialized in the order of their lock points (i.e., the point where a transaction acquired its final lock).

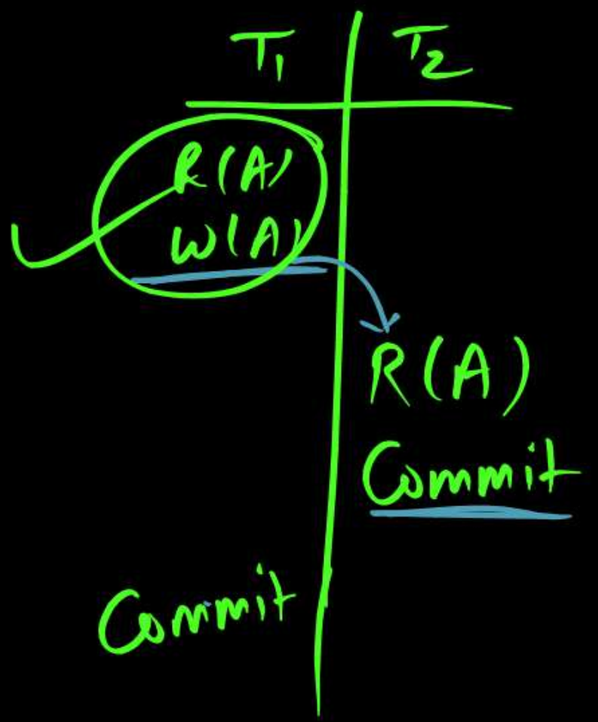

T1
T2
R(A)
A=A-50
W(A)
R(A)
R(B)
R(B)
B=B+50
W(B)
T1
T2
Cycle Not Conflict.
T1
T2
Lock-X(A)
R(A)
A=A-50
Write(A)
Lock-X(B)
Unlock-X(A)
R(B)
B=B+50
W(B)
Unlock-X(B)
Lock-S(A)
T1
T2
Conflict Serializable
Lock-S(A)
Read(A)
Lock-S(B)
Lock-S(B)
Unlock-S(A)
R(B)
Unlock-(B)
Grant
X No
Deny/WAIT
Grant
release-X(A, T1)
YES
X No (Bz 'B' taken by T1)

Note

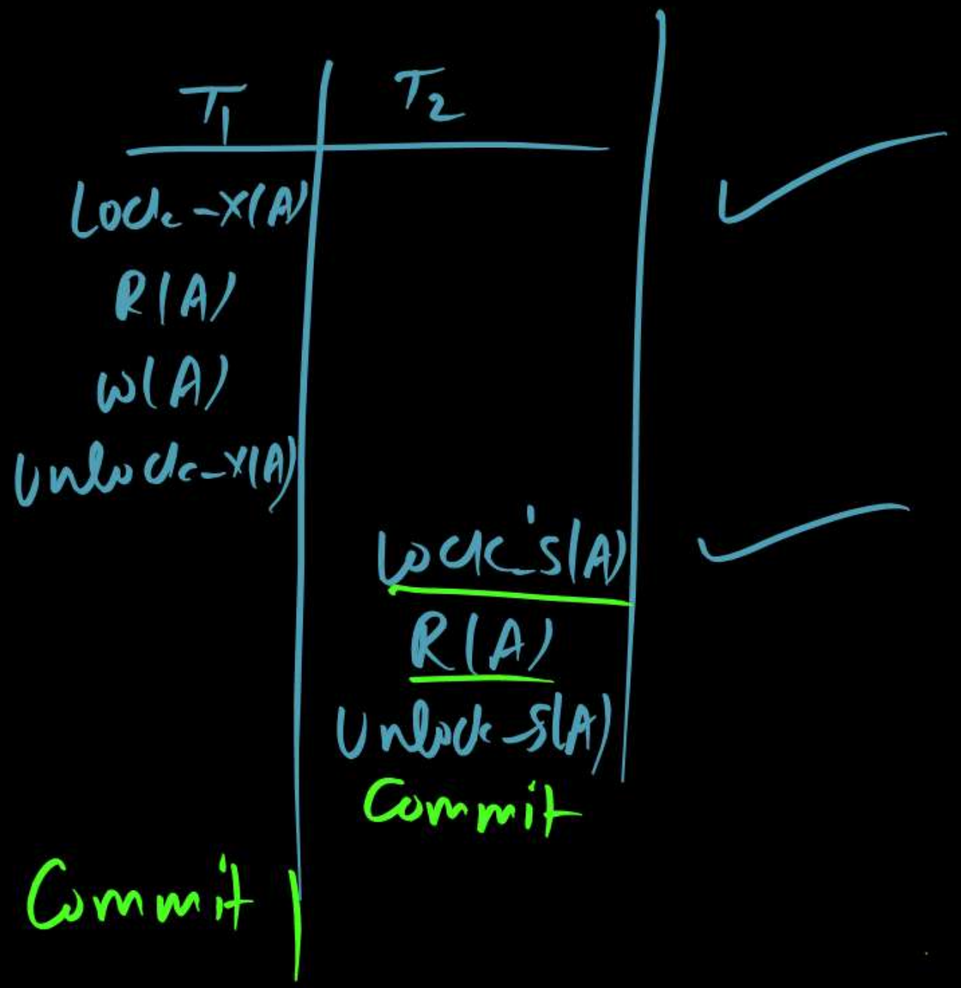
If a schedule followed by 2PL (Applied 2PL successfully) then its Conflict Serializable Schedule.

Note

2PL ensure Serializability.

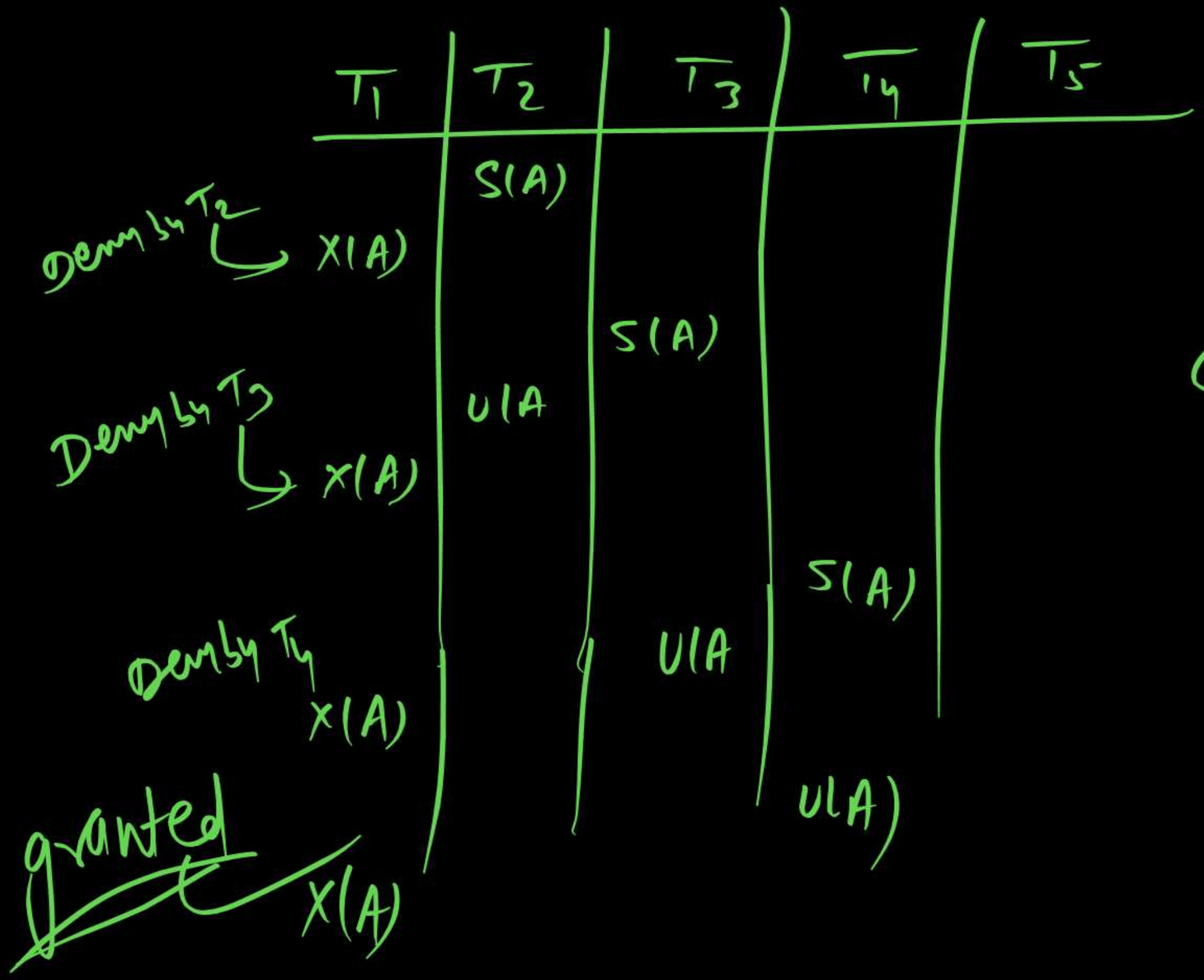


Irrecoverable

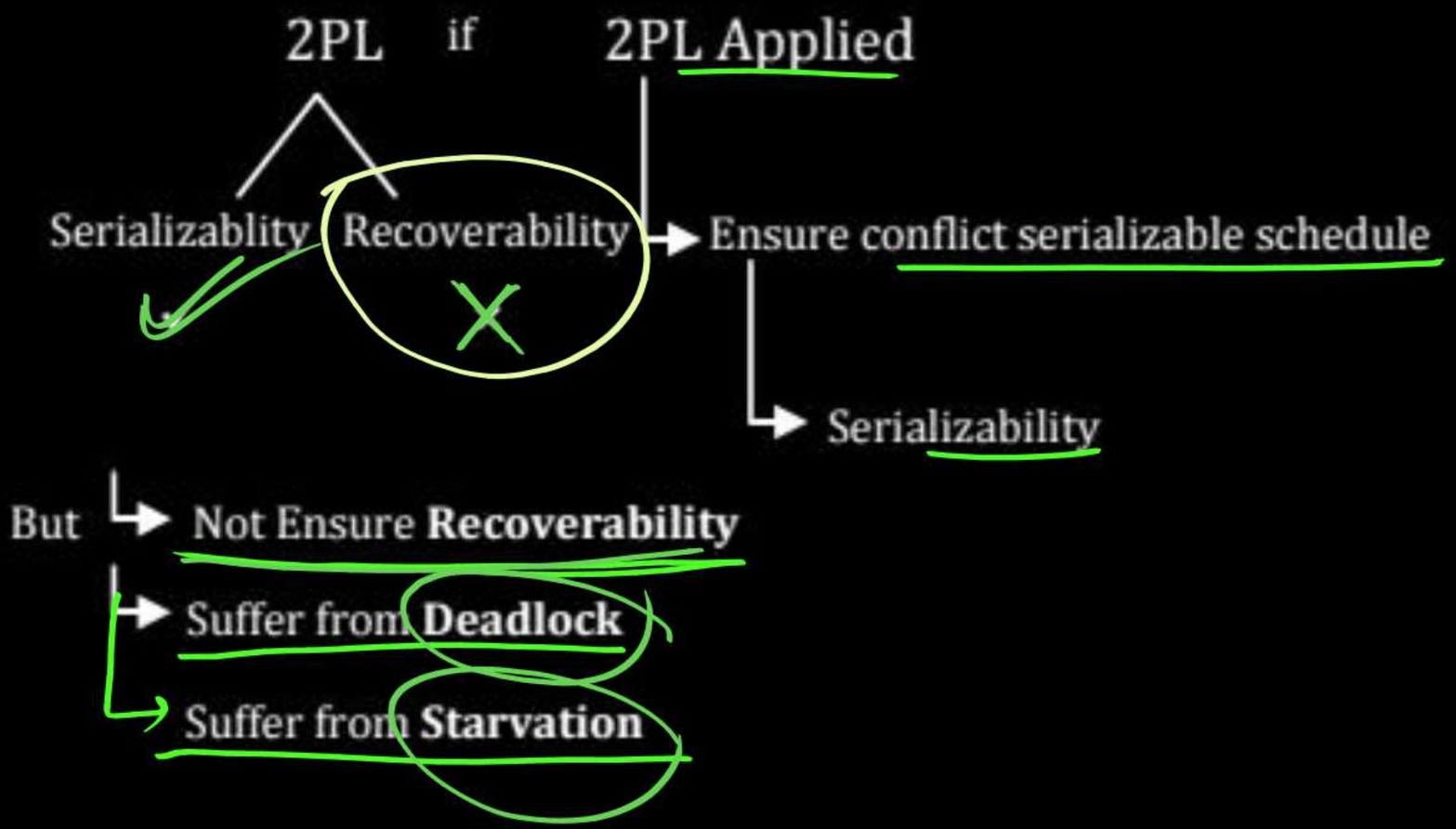


It's Irrecoverable but followed by 2PL.


T1
T2
W(A)
R(B)
W(B)
R(A)
T1
T2
X(A)
W(A)
S(B)
R(B)
Deny by T2
X(B)
S(A)
Deny by T1
Deadlock


T1
T2
T3
T4
T5
S(A)
Denied by T2
X(A)
S(A)
Denied by T3
U(A
X(A)
S(A)
Denied by T4
X(A)
U(A
U(A)
granted
X(A)

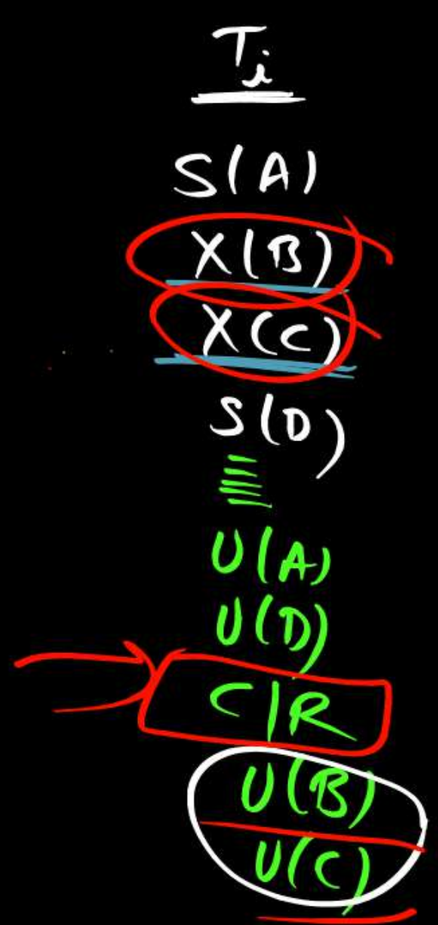
Starvation

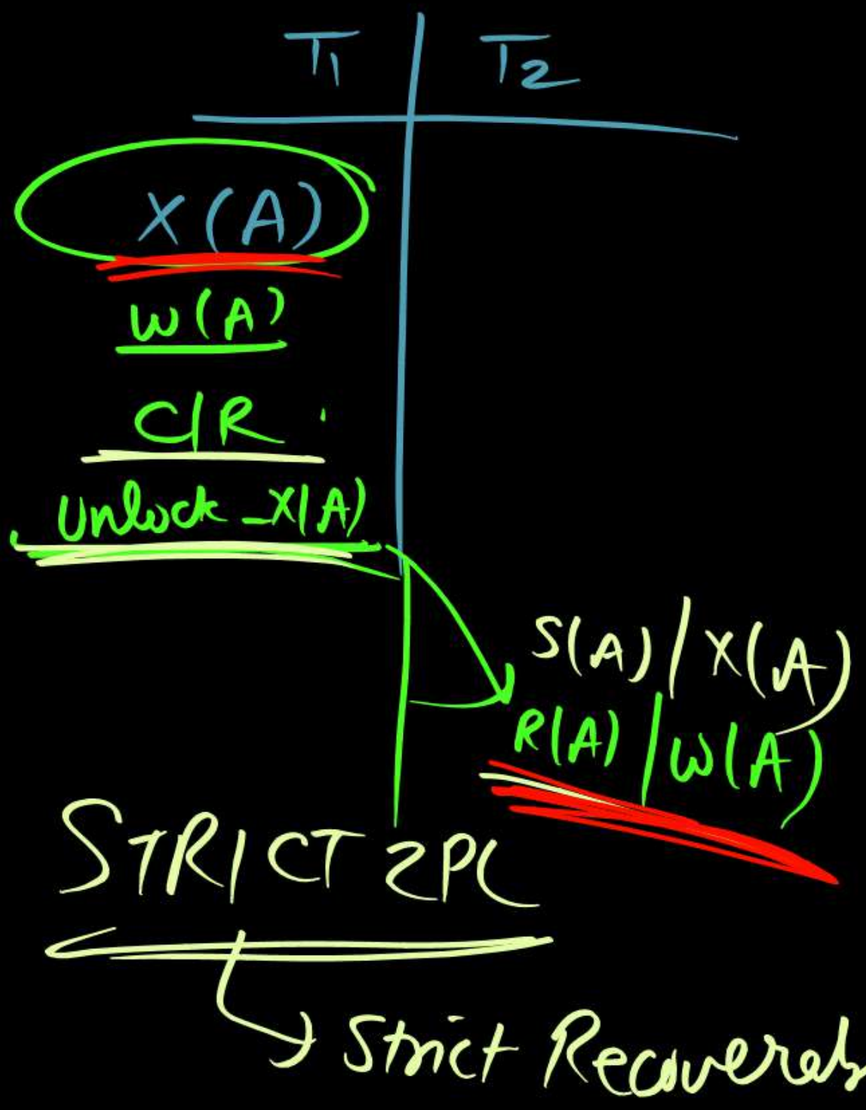

2PL if 2PL Applied
Serializablity
Recoverability
X
Ensure conflict serializable schedule
Serializability
But
Not Ensure Recoverability
Suffer from Deadlock
Suffer from Starvation

STRICT 2PL : 2PL + All Exclusive Lock Must be Hold Untill Commit/Rollback of the Transaction

OR

2PL + All X lock Release After Commit/Rollback of the Transaction



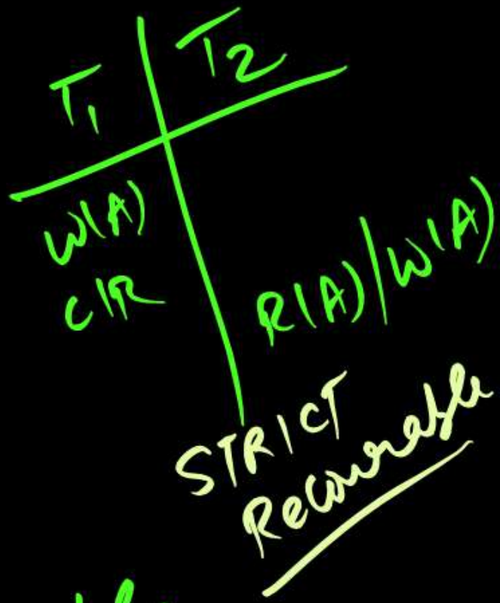


STRICT 2PL

↳ Recoverable

↳ Cascadeless

↳ Strict Recoverable



(S/X)

Rigorious 2PL :- 2PL + All Lock Hold Untill Commit/Rollback

(or)

2PL + Release All lock after C/R

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

- Deadlock
- Starvation

C/R

U(A)
U(B)
U(C)
U(D)

STRICT 2PL

Rigorious 2PL

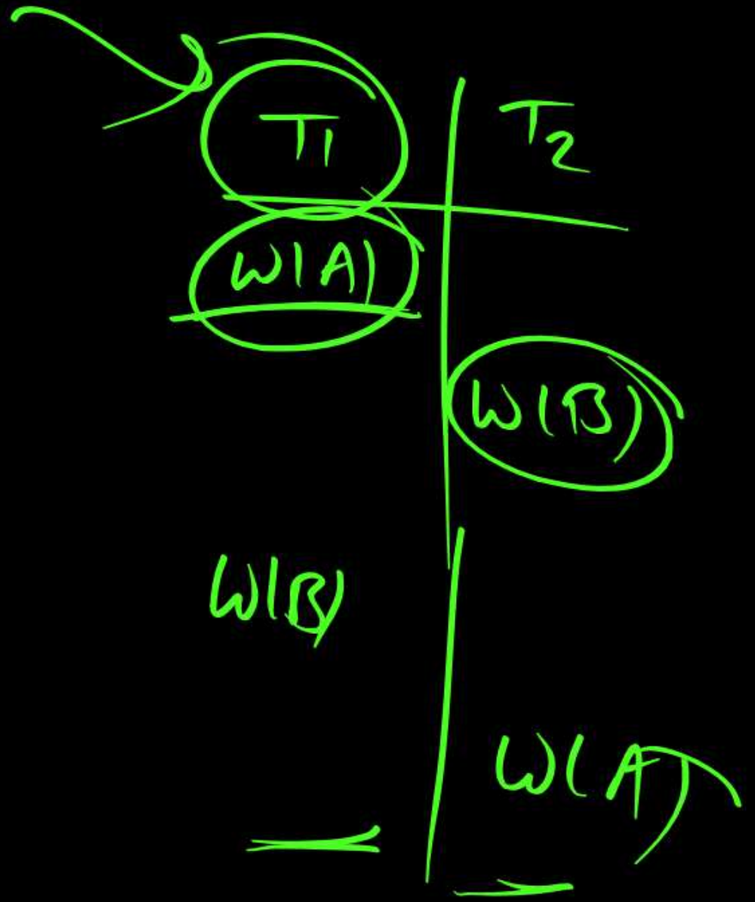
→ Suffer Deadlock & Starvation.

The Two-Phase Locking Protocol (Cont.)

- Two-phase locking does not ensure freedom from deadlocks
- Extensions to basic two-phase locking needed to ensure recoverability of freedom from cascading roll-back
  - Strict two-phase locking: a transaction must hold all its exclusive locks till it commits/aborts.
    - Ensures recoverability and avoids casecading roll-backs
  - Rigorous two-phase locking: a transaction must hold all locks till commit/abort.
    - Transactions can be serialized in the order in which they commit.
- Most databases implement rigorous two-phase locking, but refer to if as simply two-phase locking

Conservative 2PL : Acquire all the lock Whenever transaction Start &

Release all lock After Commit.



↳ No Deadlock

Only Starvation

1	2	3	4	5
Lock-S(A)	Lock-S(A)	Lock-S(A)	Lock-S(A)	Lock-S(A)
R(A)	R(A)	R(A)	Lock-X(B)	R(A)
Lock-X(B)	Lock-X(B)	Lock-X(B)	R(A)	Unlock(A)
R(B)	Unlock(A)	R(B)	R(B)	Lock-X(B)
Unlock(A)	R(B)	W(B)	W(B)	R(B)
W(B)	W(B)	Commit	Commit	W(B)
Unlock(B)	Commit	Unlock(A)	Unlock(A)	Unlock(B)
Commit	Unlock(B)	Unlock(B)	Unlock(B)	Commit

Basic 2PL

Strict 2PL

R(A)

R(B)

W(B)

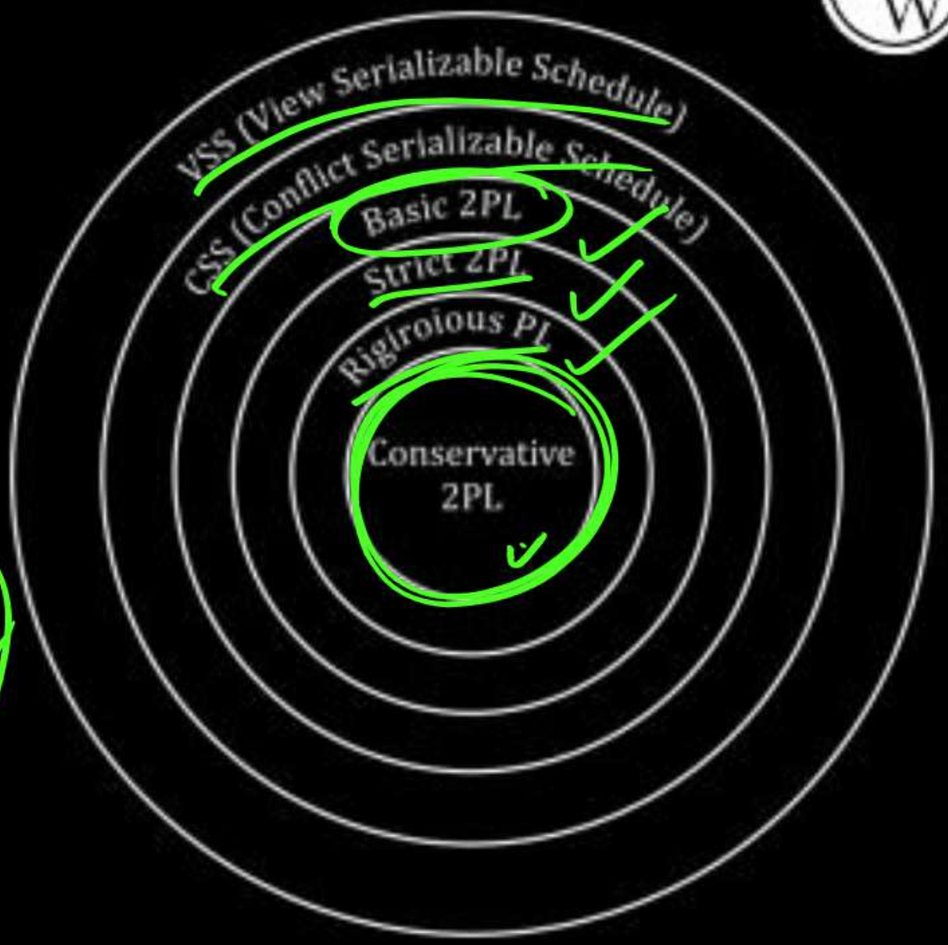
Rigorious 2PL

+ Strict

+ Basic 2PL

Conservative 2PL

Not 2PL



TIMESTAMP BASED CONCURRENCY CONTROL

Time Stamp

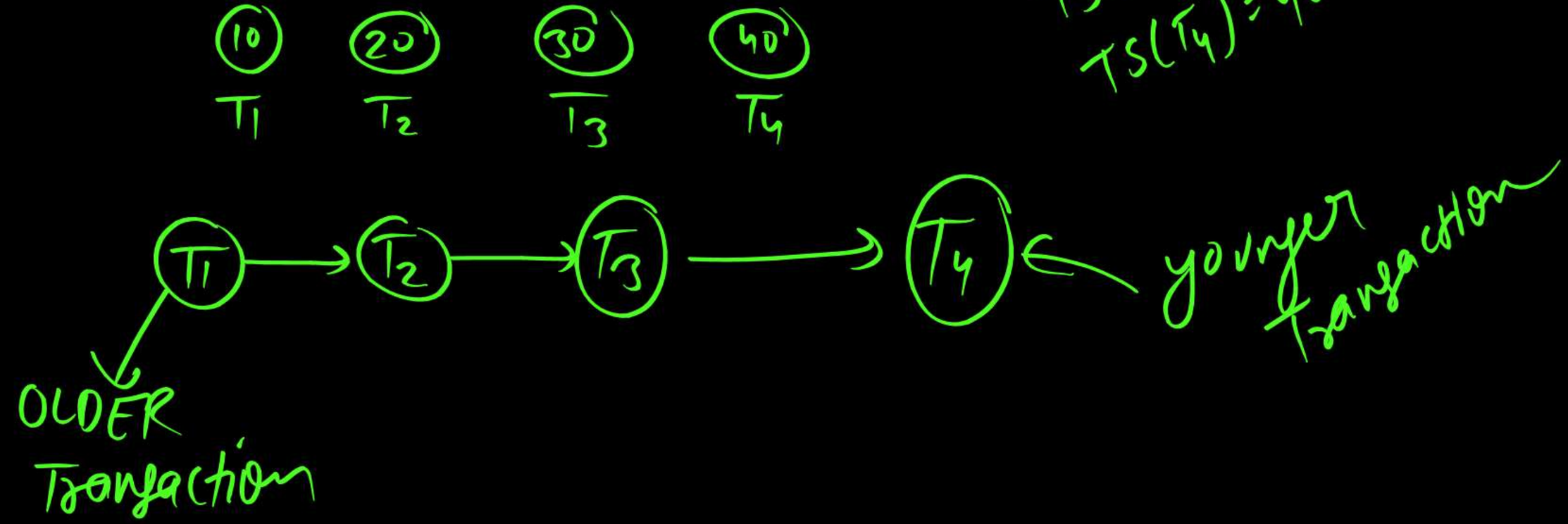
↳ Unique Value

$TS(T_1) = 10$

$TS(T_2) = 20$

$TS(T_3) = 30$

$TS(T_4) = 40$

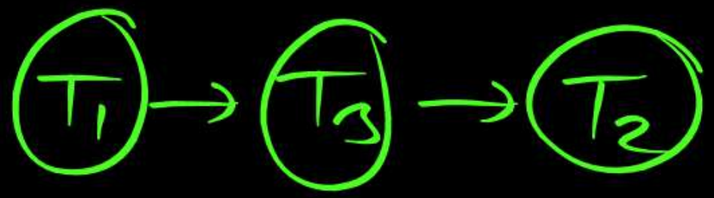


Timestamp-Based Protocols

TS(T_i)

- Each transaction T_i is issued a timestamp TS(T_i) when it enters the system.
  - Each transaction has a unique timestamp
  - Newer transaction have timestamp strictly greater than earlier ones
  - Timestamp could be based on a logical counter
    - Real time may not be unique
    - Can use (wall-clock time, logical counter) to ensure
- Timestamp-based protocols manage concurrent execution such that **time-stamp order = serializability order**
- Several alternative protocols based on timestamps

10	30	20
T_1	T_2	T_3

Serializability Order : 

2 Type of Time Stamp

(1) Read Time Stamp(A) ⇒ RTS(A)

(2) Write Time Stamp(A) ⇒ WTS(A)

RTS(A) : max Time Stamp [R(A)] Successfully

10	20	30
T_1	T_2	T_3
R(A)		
	R(A)	
		R(A)

RTS(A) = 10

RTS(A) = 20 [max(10,20)]

RTS(A) = 30 [max(20,30)]

10	20	30
T_1	T_2	T_3
W(A)		
	W(A)	
		W(A)

WTS(A) =

Timestamp-Based Protocols

The **timestamp ordering (TSQ) protocol**

- Maintains for each data Q two timestamp values:
  - **W-timestamp**(Q) is the largest time-stamp of any transaction that executed write (Q) successfully.
  - **R-timestamp** (Q) is the largest time-stamp of any transaction that executed **read** (Q) successfully.
- Imposes rules on read and write operations to ensure that
  - any conflicting operations are executed in timestamp order
  - out of order operations cause transaction rollback

Time Stamp Protocol

I. Transaction Time Stamp. $TS(T_i)$ (Fixed)

II. Time Stamp of Data Item (Variable)

$RTS(Q)$
$WTS(Q)$

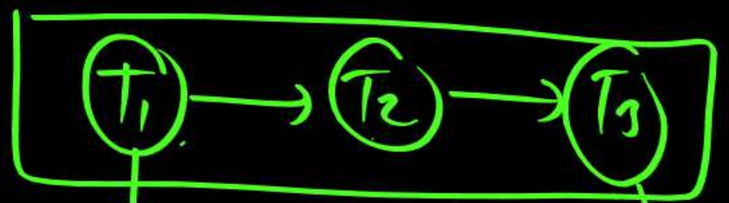
max Trans' stamp that perform operation successfully

10 T_1 $\quad$ 20 T_2 $\quad$ 30 T_3

$TS(T_1) = 10$

$TS(T_2) = 20$

$TS(T_3) = 30$



OLDER Transaction

Younger (Newer) Transaction

R-W
W-R
W-W

Note: Transaction Order Same — All Conflict operation order

R-W
W-R
W-W

(I) T_i : Read(Q)

$TS(T_i) < WTS(Q)$: Reject, & Rollback

(II) T_i : write(Q)

$TS(T_i) < RTS(Q)$

$TS(T_i) < WTS(Q)$

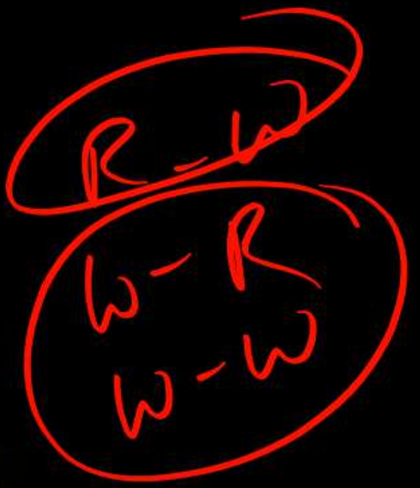
Write operation & T_i Rollback
Reject

I: T_i – Read(Q) (Transaction T_i Issue R(Q) Operation)

(i) If TS (T_i) < WTS (Q): Read operation Reject & T_i Rollback.

(ii) If TS (T_i) ≥ WTS(Q): Read operation is allowed

and Set Read – TS(Q) = max[RTS(Q), TS (T_i)]



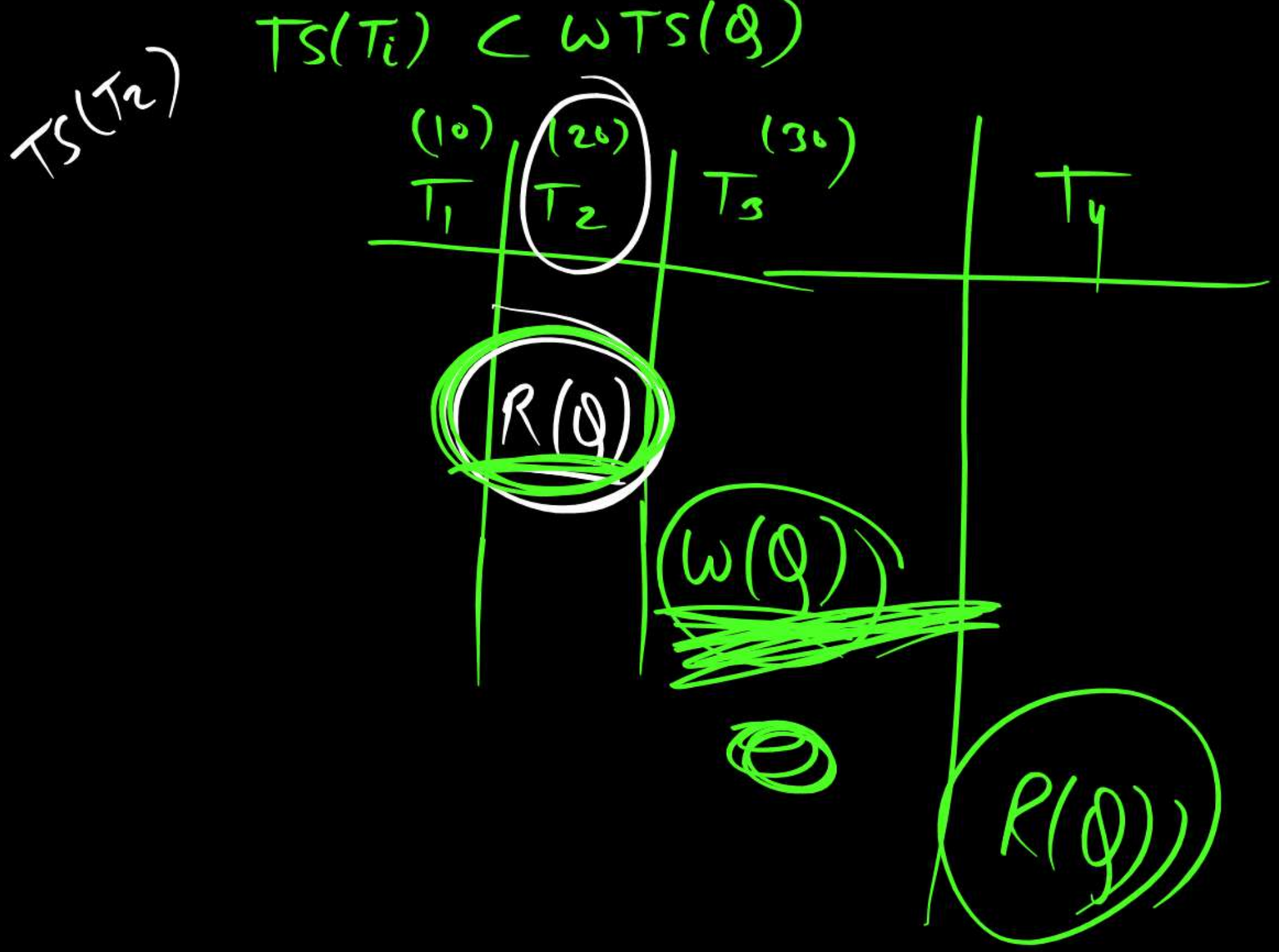
II: T_i – Write(Q) (Transaction T_i Issue Write(Q) Operation)

(i) If TS (T_i) < RTS (Q): Write operation Reject & T_i Rollback.

(ii) If TS (T_i) < WTS(Q): Write operation Reject & T_i Rollback.

(iii) Otherwise execute write (Q) operation

Set Read WTS(Q) = TS (T_i)


TS(Ti) < WTS(Q)
TS(T2)
(10)
(20)
(30)
T1
T2
T3
T4
R(Q)
W(Q)
R(Q)

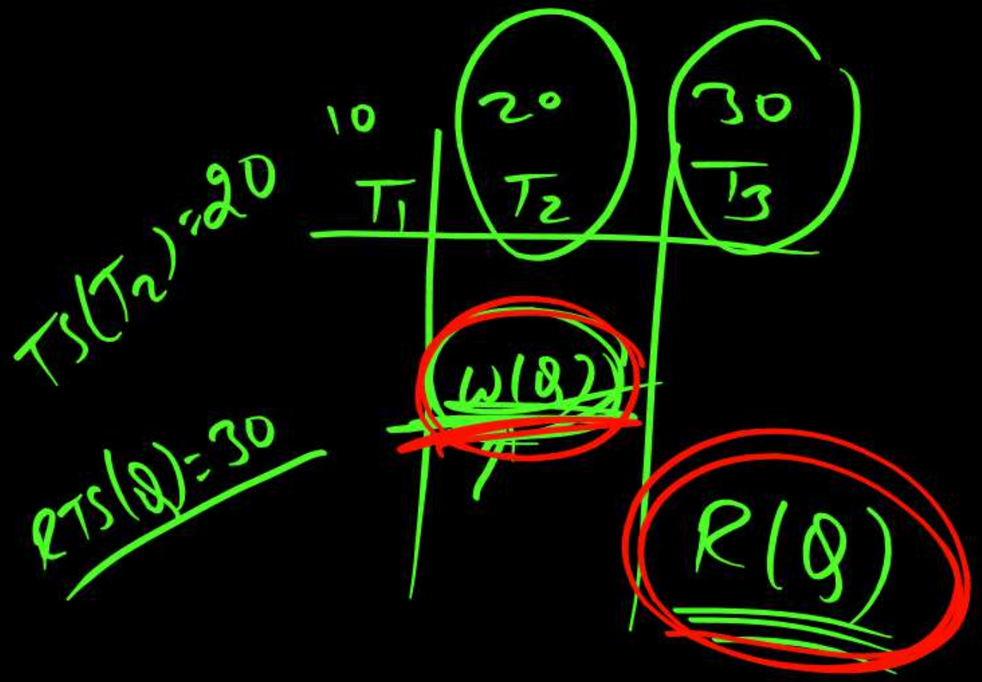
Timestamp-Based Protocols (Cont.)

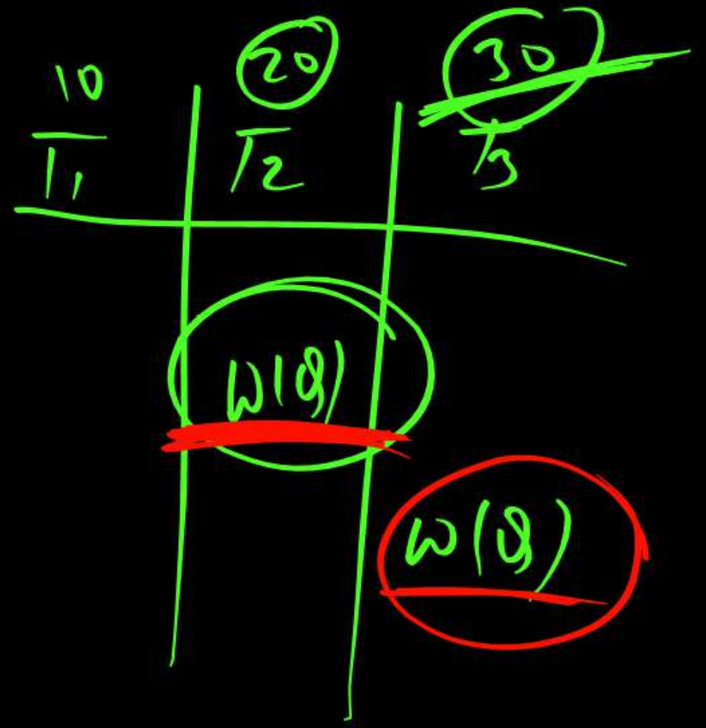
- Suppose a transaction T_i issues a **read** (Q)
  1. If $TS(T_i) \leq$ **W**-timestamp (Q), then T_i needs to read a value of Q that was already overwritten.
     - Hence, the **read** operation is rejected, and T_i is rolled back.
  2. If $TS(T_i) \geq$ **W**-timestamp (Q), then the **read** operation is executed, and R-timestamp(Q) is set to.

     max(R-timestamp (Q), TS (T_i)).

Timestamp-Based Protocols (Cont.)

- Suppose a transaction T_i issues **write**(Q)
  1. If $TS(T_i)$ < **R**-timestamp(Q), then the value of Q that T_i is producing was needed previously, and the system assumed that the value would never be produced.
     - Hence, the **write** operation is rejected, and T_i is rolled back.
  2. If $TS(T_i)$ < **W**-timestamp (Q), then T_i is attempting to write an obsolete value of Q.
     - Hence, this **write** operation is rejected, and T_i is rolled back.
  3. Otherwise, the **write** operation is executed, and W-timestamp(Q) is set to $TS(T_i)$.

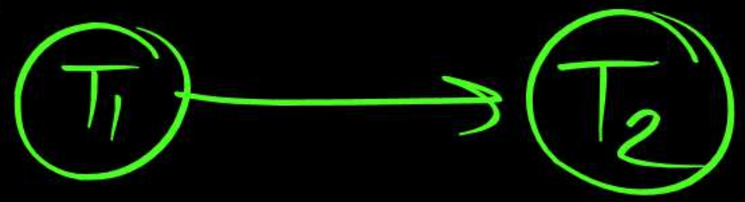

TS(T2)=20
RTS(Q)=30
10
T1
20
T2
30
T3
W(Q)
R(Q)


10
T1
20
T2
30
T3
W(Q)
W(Q)

If Transaction order

All Conflict operation order then its allowed by TSP.

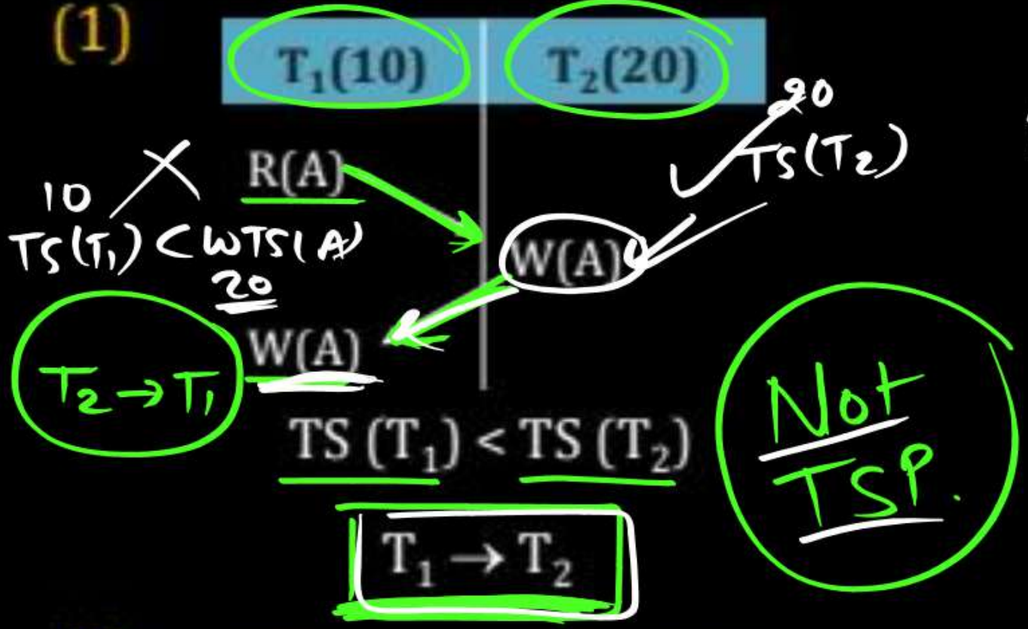
eg



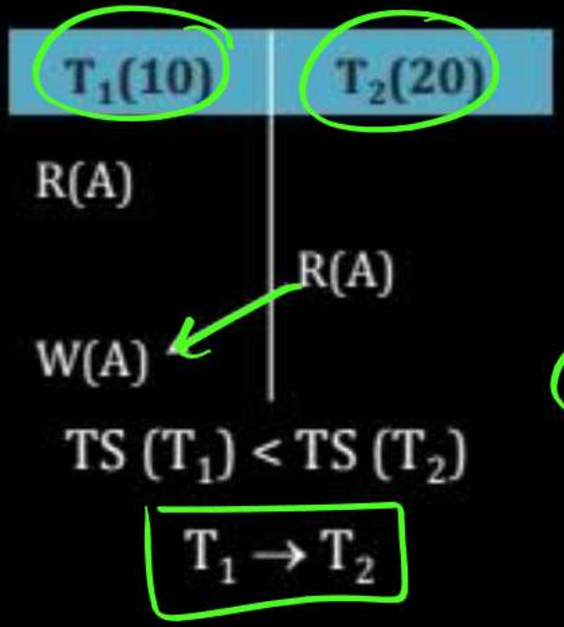
(T_1 followed by T_2)

eg All Conflict operation order must be $T_1 \rightarrow T_2$

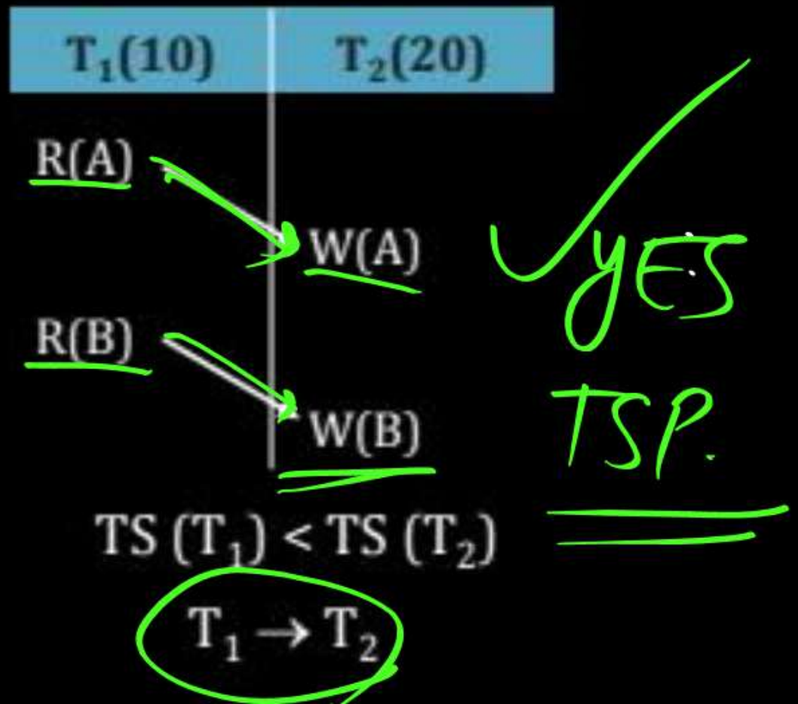
(1)


T1(10)
T2(20)
R(A)
W(A)
W(A)
10
TS(T1) < WTS(A)
20
20
TS(T2)
10
RTS(A)
T2 → T1
TS (T1) < TS (T2)
T1 → T2
Not TSP.

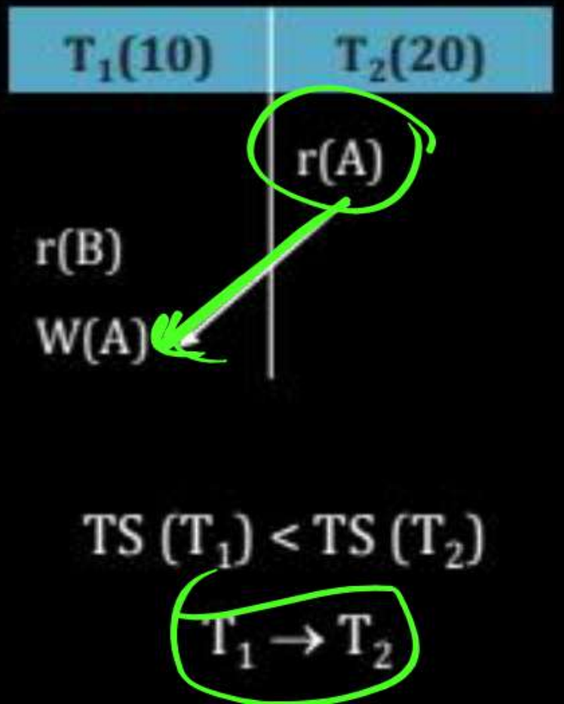
(2)


T1(10)
T2(20)
R(A)
R(A)
W(A)
TS (T1) < TS (T2)
T1 → T2
R(A) - W(A)
T2 → T1
Not TSP.

(3)


T1(10)
T2(20)
R(A)
W(A)
R(B)
W(B)
YES
TSP.
TS (T1) < TS (T2)
T1 → T2

(4)


T1(10)
T2(20)
r(A)
r(B)
W(A)
Not
TSP.
TS (T1) < TS (T2)
T1 → T2

Thomas Write Rule (View Serializability)

T_i : Read(Q)

$TS(T_i) < WTS(Q)$; Reject & Rollback

Otherwise Read operation allowed.

T_i : Write(Q)

① $TS(T_i) < RTS(Q)$; Reject & Rollback

② $TS(T_i) < WTS(Q)$; Ignored & No Rollback

Thomas' Write Rule

- Modified version of the timestamp-ordering protocol in which obsolete **write** operations may be ignored under certain circumstances.
- When T_i attempts to write data item Q, if $TS(T_i) < W\text{-timestamp}(Q)$, then T_i is attempting to write an obsolete value of {Q}.
  - Rather than rolling back T_i as the timestamp ordering protocol would have done, this {**write**} operation can be ignored.
- Otherwise this protocol is the same as the timestamp ordering protocol.
- Thomas' Write Rule allows greater potential concurrency.
  - Allows some view-serializable schedules that are not conflict-serializable.

Thomas Write Rule (View Serializability)

1. TS (T_i) < RTS(Q) : Rollback
2. TS(T_i) < WTS(Q) : Write operation is Ignored and No Roll back

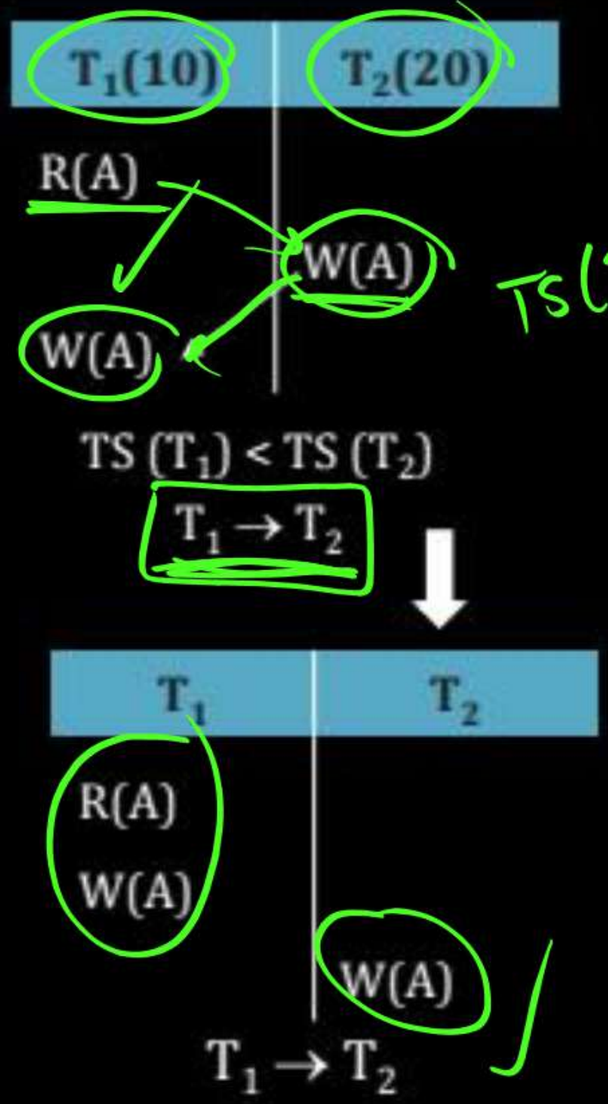
Same as TSP

Time Stamp Protocol: Ensure serializability deadlock free but starvation possible

Deadlock Prevention Algorithm

(1) Wait-Die (2) Wound-wait

Older Younger



TS(T2) < WTS(A) Ignored & No Rollback

Query Language

Procedural Query Language

↳ WHAT to Retrieve

HOW

Relational Algebra:

Non Procedural Query Language

- SQL
- TRC

Relational Algebra

- Procedural language
- Six basic operators
  - Select :(σ)
  - Project :(π)
  - Union:(∪)
  - Set difference: –
  - Cartesian product: x
  - Rename : ρ
- The operators take one or two relations as inputs and produce a new relation as a result.

Derived operators

1. JOIN (⋈)
2. Division [÷]
3. Intersection (∩)

$R \cap S = R - (R - S)$

$R = [1, 2, 3, 4, 5]$

$S = [3, 4, 5, 6, 7]$

$R \cap S = [3, 4, 5]$

$R - (R - S)$

$R - S = [1, 2]$

$R - (R - S) \Rightarrow [1, 2, 3, 4, 5] - (1, 2) = [3, 4, 5]$

Note

Basic Idea of Query Language

Executed → Tuple by Tuple (Row by Row)

Reserves (R_1)

Sid	Bid	day
22	101	10/10/96
58	103	11/12/96

Sailors(S_1)

Sid	Sname	Rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

Sailors(S_2)

Sid	Sname	Rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

Selection (σ)

Syntax

$\sigma_{\text{Condition}}(\text{Relation})$

It Select tuples Based on specified Condition.

eg

$\sigma_{\text{CGPA} > 8}(\text{STUDENT})$

Projection (π)

Syntax

$\pi_{\text{Attribute}}(\text{Relation})$

(or)

Attribute List

It Select Attribute / Attribute List.

$\pi_{\text{sid}}(\text{Student})$

$\pi_{\text{sid, Contact}}(\text{STUDENT})$

Selection (σ)

Q. $\sigma_{\text{rating} > 8}(S_2)$

Sailors (R_2)

Sid	Sname	Rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

Output:

Sid	Sname	Rating	age
28	yuppy	9	35.0
58	rusty	10	35.0

Q. $\pi_{sname,\ rating}\ (\sigma_{rating\ >\ 8}\ (S_2))$

Output:

Sname	Rating
yuppy	9
rusty	10

Projection (π)

⇓

'Distinct'

Projection(π)

Q. $\pi_{age}(S_2)$

Sailors (S_2)

Sid	Sname	Rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

Output:

age
35.0
55.5

Q. $\pi_{\text{sname, rating}}(S_2)$

Output:

Sname	Rating
yuppy	9
lubber	8
guppy	5
rusty	10

Set operation

'# Attribute'

Union

1. Arity Same
2. Range Similar

R

1	2	3

X

S

1	2

RollNo	Name

X

Branch	Contact

Union Operation

- Notation: $r \cup s$
- Defined as :

$$r \cup s = \{t | t \in r \text{ or } t \in s\}$$

- For $r \cup s$ to be valid.
  1. r, s must have the same arity (same number of attributes)
  2. The attribute domains must be compatible (example: 2nd column or r deals with the same type of values as does the 2nd column of s)

Example:

To find all courses taught in the Fall 2009 semester, or in the Spring 2010 semester, or in both.

$$\pi_{course\_id}(\sigma_{semester = \text{“Fall”} \wedge year = 2009}(section)) \cup \pi_{course\_id}(\sigma_{semester = \text{“Spring”} \wedge year = 2010}(section))$$

Set Difference Operation

- Notation: r – s
- Defined as :

$$r - s = \{t \mid t \in r \text{ and } t \notin s\}$$

- Set differences must be taken between compatible relations.
  - r and s must have the same arity
  - attribute domains of r and s must be compatible

Example:

Example: to find all courses taught in the Fall 2009 semester, but not in the Spring 2010 semester

$$\pi_{course\_id}(\sigma_{semester = \text{“Fall”} \wedge year = 2009}(section))$$

$$\pi_{course\_id}(\sigma_{semester = \text{“Spring”} \wedge year = 2010}(section))$$

Assume Relation R & Relation S consist M & N Tuple Respectively

R ⇒ m Tuple

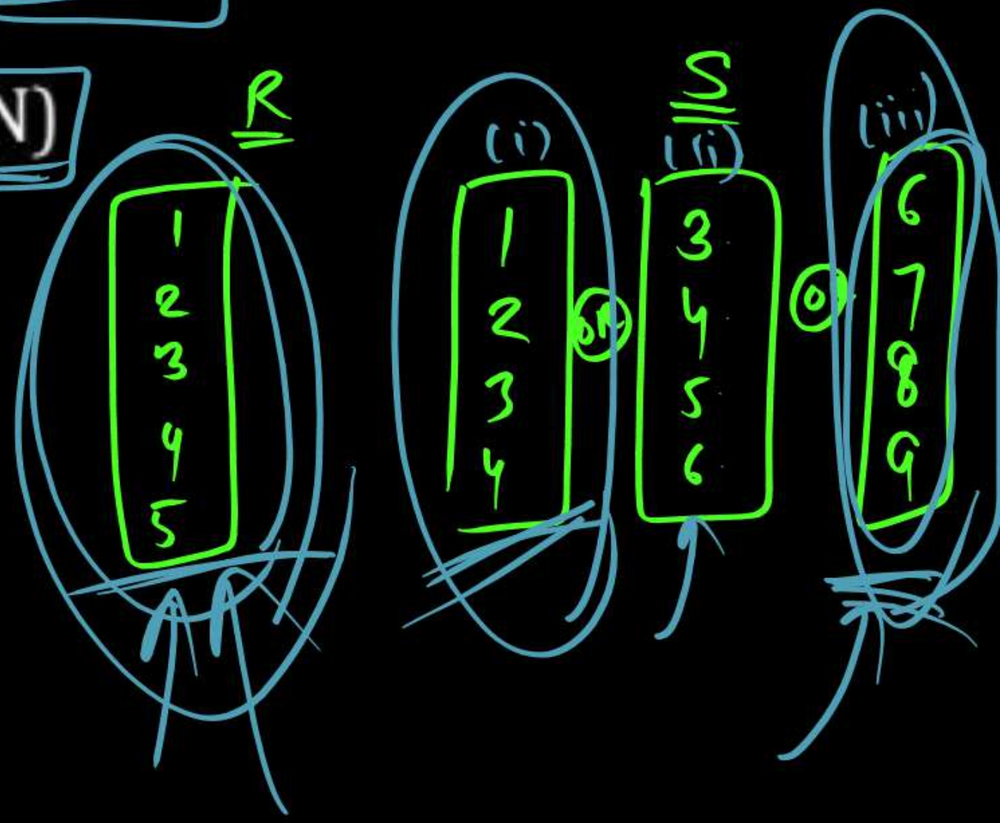
S ⇒ N Tuple

(1) Range of tuples in R ∪ S = max(M, N) to M + N

(2) Range of tuples in R ∩ S = ϕ to min (M, N)

(3) Range of tuples in R – S = ϕ to M

(4) Range of tuples in S – R = ϕ to N



Example:

28 yuppy
44 Guppy

Sailors (S_1)

Sid	Sname	Rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

Sailors (S_2)

Sid	Sname	Rating	age
28	Yuppy	9	35.0
31	Lubber	8	55.5
44	Guppy	5	35.0
58	rusty	10	35.0

5

2

$S_1 \cup S_2$

$S_1 \cap S_2$

$S_1 - S_2 \Rightarrow$ (1) → 22

$S_2 - S_1 \Rightarrow$ (2) → 28, 44

(1) Union

Sid	Sname	Rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0
44	guppy	5	35.0
28	yuppy	9	35.0

$S1 \cup S2$

(2) Set Difference

Sid	Sname	Rating	age
22	dustin	7	45.0

S1 – S2

(3) Intersection

Sid	Sname	Rating	age
31	lubber	8	55.5
58	rusty	10	35.0

$S1 \cap S2$

CROSS Product (R X S)

R	S
n_1 Tuple	n_2 Tuple
C_1 Attribute	C_2 Attribute

$R \times S$ = $n_1 \times n_2$ Tuple
$C_1 + C_2$ Attribute

Cross – Product

Reserves (R_1)

Sid	bid	day
22	101	10/10/96
58	103	11/12/96

Sailors (S_1)

Sid	Sname	Rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

R_1 : 2 Tuple
3 Attribute

S_1 : 3 Tuple
4 Attribute

$R_1 \times S_1 = 2 \times 3 = 6$ Tuple

$3 + 4 = 7$ Attribute

$R_1 \times S_1$

(Sid)	Sname	Rating	age	(Sid)	bid	day
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96

Selection (σ)

Projection (π)

① Arity
② Power
- Union
- Intersection
- Set Difference

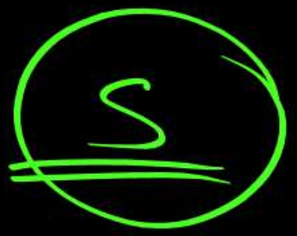
Arity

CROSS Product

$R \times S = n_1 \times n_2$ Tuple

$C_1 + C_2$ Attribute

CROSS Product ?



JOIN

R.x < S.y

Join Operations

(1) Conditional Join ($\bowtie_c$)

(2) Equi join

(3) Natural join

(4) Left outer join

(5) Right outer join

(6) Full outer join

JOIN ($\bowtie_c$) $(R \bowtie_c S)$

① Condition Join

Step 1 CROSS Product of R & S. $(R \times S) \Rightarrow n_1 \times n_2$ Tuple $c_1 + c_2$ Attribute

Step 2 Select the Tuple which satisfy given Condition

Step 3 Projection of Attribute.

$$R \bowtie_c S = \Pi_{\text{All Attribute}}\left[\sigma_{\text{Given Condition}}(R \times S)\right]$$

step ③ — $\Pi_{\text{All Attribute}}$

step ② — $\sigma_{\text{Given Condition}}$

step ① — $(R \times S)$

Conditional Join

$$R \bowtie_c S = \sigma_C (R \times S)$$

$R_1 \times S_1$

R_1 Sid (Sid)	Sname	Rating	age	S_1 Sid (Sid)	bid	day
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96

Conditional Join

$S1 \bowtie_{S1.sid < R1.sid} R1$

output

(Sid)	Sname	Rating	age	(Sid)	bid	day
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	58	103	11/12/96

$S1 \bowtie_{S1.sid < R1.sid} R1$

- Result schema same as that of cross-product.
- Fewer tuples than cross – product, might be able to compute more efficiently.
- Sometimes called a theta –join.

Equi – Join

A special case of condition join where the correction c contains only equalities.

$R_1.Sid = S_1.Sid$

$R_1 \times S_1$

R_1 (Sid)	Sname	Rating	age	S_1 (Sid)	bid	day
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96

Equi Join ⇒ Equality Condition

Equi – Join

$S1 \bowtie_{sid} R1$

Sid	Sname	Rating	age	bid	day
22	dustin	7	45.0	101	10/10/96
58	rusty	10	35.0	103	11/12/96

- Result schema similar to cross – product, but only one copy of fields for which equality is specified.
- Natural join: Equijoin on all common fields.

$R(A\ B\ C)$ $\quad$ $S(B\ C\ D)$

$R \bowtie S =$

$$\Pi_{\text{Distinct Attribute}}\left[\sigma_{\text{equality cond. on all common Attribute}}(R \times S)\right]$$

$R \bowtie S =$

$$\underline{\Pi_{ABCD}}\left[\sigma_{\substack{R.B=S.B \wedge \\ R.C=S.C}}(R \times S)\right]$$

Natural Join ($\bowtie$) $(R \bowtie S)$

Step ① Cross Product of R & S $(R \times S)$

Step ② Select the tuple which satisfy equality condition On All Common Attribute.

Step ③ Projection of Distinct Attribute

JOIN

1. Conditional Join
2. Equi Join
3. Natural Join.
4. Left outer Join
5. Right outer Join
6. Full Outer Join.

THANK YOU!