

Jai Shree Ram

INDEX

NAME: Anurag Singh UE: DBMS STD: SEC. ROLL NO.

S. No.	Date	Title	Page No.	Teacher's Sign / Remarks
82.		Conflict Serializability; precedence Graph	1-4	
83.		View " " "	4-6.	
84.		Concurrency Control protocols	7-8.	
85.		Drawbacks in Shared / Exclusive	9-11	
86.		phase locking (2PL) protocol	12-15.	
87.		Drawbacks in 2PL protocol	15-17.	
88.		Strict 2PL, Rigorous 2PL & Conservative 2PL	17-21.	Schedu
89.		Basic Timestamp Ordering Protocol	21-24	
90.		Solve Ques" on " " " " "	24-25.	
91.		INDEXING	26-28.	
92.		Ques" on I/O Cost in Indexing	29-31.	
93.		Numerical on " " " " "	31-35.	
94.		Types of INDEXES	35	
95.		Primary INDEX	36-37.	
96.		CLUSTERED Index	37-38.	
97.		Secondary Index (multilevel Indexing)	39-42.	
98.		Intro to B-tree & B+ Tree structure	42-45.	
99.		Insertion in B-Tree	45-47.	
100.		How to find order of B-Tree	48-49.	
101.		SLB B-Tree & B+ Tree	49-53.	
102.		Ques" on order of B+ Tree	53-54.	
103.		Immediate Database Modification	55-57.	
104.		O" on DBMS Basic Concepts & Data Modelling	57-59.	
105.		Imp Ques" on Advance DBMS	60-63.	
106.		Deferred Database Modification	63-66.	
107.		LIKE Command in SQL	66-67.	
108.		Basic PL-SQL programming with Execution	68-69.	

S. No.	Date	Title	Page No.	Teacher's Sign / Remarks
109.		PL-SQL (while, for loop).	69 - 70.	
110.		single & multiRow func's in SQL	70 - 71.	
111.		character func's in SQL	72 - 73.	
112.		Views in Database.	74 - 77.	
*		<u>SQL cheatsheet</u>	78 - 81.	
113.				
114.				
115.				
116.				
117.				
118.				
119.				
120.				
121.				
122.				
123.				
124.				
125.				
126.				
127.				
128.				
129.				
130.				
131.				
132.				
133.				
134.				
135.				
136.				
137.				
138.				
139.				
140.				
141.				
142.				
143.				
144.				
145.				
146.				
147.				
148.				
149.				
150.				
151.				
152.				
153.				
154.				
155.				
156.				
157.				
158.				
159.				
160.				
161.				
162.				
163.				
164.				
165.				
166.				
167.				
168.				
169.				
170.				
171.				
172.				
173.				
174.				
175.				
176.				
177.				
178.				
179.				
180.				
181.				
182.				
183.				
184.				
185.				
186.				
187.				
188.				
189.				
190.				
191.				
192.				
193.				
194.				
195.				
196.				
197.				
198.				
199.				
200.				
201.				
202.				
203.				
204.				
205.				
206.				
207.				
208.				
209.				
210.				
211.				
212.				
213.				
214.				
215.				
216.				
217.				
218.				
219.				
220.				
221.				
222.				
223.				
224.				
225.				
226.				
227.				
228.				
229.				
230.				
231.				
232.				
233.				
234.				
235.				
236.				
237.				
238.				
239.				
240.				
241.				
242.				
243.				
244.				
245.				
246.				
247.				
248.				
249.				
250.				
251.				
252.				
253.				
254.				
255.				
256.				
257.				
258.				
259.				
260.				
261.				
262.				
263.				
264.				
265.				
266.				
267.				
268.				
269.				
270.				
271.				
272.				
273.				
274.				
275.				
276.				
277.				
278.				
279.				
280.				
281.				
282.				
283.				
284.				
285.				
286.				
287.				
288.				
289.				
290.				
291.				
292.				
293.				
294.				
295.				
296.				
297.				
298.				
299.				
300.				
301.				
302.				
303.				
304.				
305.				
306.				
307.				
308.				
309.				
310.				
311.				
312.				
313.				
314.				
315.				
316.				
317.				
318.				
319.				
320.				
321.				
322.				
323.				
324.				
325.				
326.				
327.				
328.				
329.				
330.				
331.				
332.				
333.				
334.				
335.				
336.				
337.				
338.				
339.				
340.				
341.				
342.				
343.				
344.				
345.				
346.				
347.				
348.				
349.				
350.				
351.				
352.				
353.				
354.				
355.				
356.				
357.				
358.				
359.				
360.				
361.				
362.				
363.				
364.				
365.				
366.				
367.				
368.				
369.				
370.				
371.				
372.				
373.				
374.				
375.				
376.				
377.				
378.				
379.				
380.				
381.				
382.				
383.				
384.				
385.				
386.				
387.				
388.				
389.				
390.				
391.				
392.				
393.				
394.				
395.				
396.				
397.				
398.				
399.				
400.				
401.				
402.				
403.				
404.				
405.				
406.				
407.				
408.				
409.				
410.				
411.				
412.				
413.				
414.				
415.				
416.				
417.				
418.				
419.				
420.				
421.				
422.				
423.				
424.				
425.				
426.				
427.				
428.				
429.				
430.				
431.				
432.				
433.				
434.				
435.				
436.				
437.				
438.				
439.				
440.				
441.				
442.				
443.				
444.				
445.				
446.				
447.				
448.				
449.				
450.				
451.				
452.				
453.				
454.				
455.				
456.				
457.				
458.				
459.				
460.				
461.				
462.				
463.				
464.				
465.				
466.				
467.				
468.				
469.				
470.				
471.				
472.				
473.				
474.				
475.				
476.				
477.				
478.				
479.				
480.				
481.				
482.				
483.				
484.				
485.				
486.				
487.				
488.				
489.				
490.				
491.				
492.				
493.				
494.				

82.

Conflict Serializability, Precedence

Graph:

T ₁	T ₂	T ₃
R(x)		
	R(y)	
	R(x)	
	R(y)	
	R(z)	
		w(y)
w(z)		
w(x)		
w(z)		

↓ — Timeline.

- First, we have to make precedence graph → (mean, just graph with edges & vertices).

Vertex → No. of Transaction (T₁, T₂ & T₃)

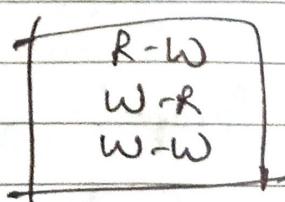
Now

& Edges

Check the conflict pairs in other transactions
(and draw edges).

Like & T₁ → (check in T₂ & T₃)

* Conflict pairs!



of same variable.
lets (x).

* Start!

Now, start with first open". i.e,

T₁ in T₁ → R(x)

2. check the conflict pair of $R(x)$
in other Transac's i.e. $(T_2 \Delta T_3)$.

→ Conflict pair of $R(x) \rightarrow w(x)$

&

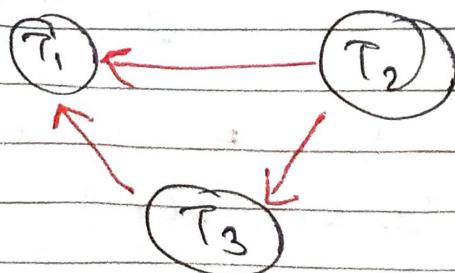
now, similarly,

conflict pair of $w(x) \rightarrow R(x), w(x)$.

Now, do check for every opera's in
all transac's & after checking list
those opera's. & If you find any
conflict pair, then draw edge acc. to
vertex (Transac').

T_1	T_2	T_3	C.P. we find →
$R(x)$		$R(y)$	$R(x) - w(x) (T_3 \rightarrow T_1)$
		$R(x)$	$R(y) - w(y) (T_2 \rightarrow T_3)$
	$R(y)$		$R(z) - w(z) (T_2 \rightarrow T_1)$
	$R(z)$		$w(z) - R(z) (T_2 \rightarrow T_1)$
		$w(y)$	$w(z) - w(z) (T_2 \rightarrow T_1)$
$R(z)$			
$w(x)$			
$w(z)$			

Precedence Graph!?



* Now, come on Graph! ~
Check that if there is any loop/cycle
in the graph.

How to check cycle: → ~~जब तक वह नहीं गति होती~~
Node जब रहता है, तब उसे लूप में कहा दिया जाता है।
उसका फल यह है। If, Yes → Loop/Cycle exists.

(It is not must that the cycle must covers all the nodes. May be possible, it comes just after visiting one node), → also a cycle.

(* In our graph, there is no loop/cycle.)

* No loop/cycle → Conflict Serializable Schedule
If loop/cycle exists → Not C.S.S.

+ Conflict Serializable → Serializable → Consistent.
(Serial Schedule)

* Now, how to make Serializable? →

$T_1 \rightarrow T_2 \rightarrow T_3$	$\left[\begin{array}{c} 6 \\ \text{Cases} \end{array} \right]$
$T_1 \rightarrow T_3 \rightarrow T_2$	
$T_2 \rightarrow T_1 \rightarrow T_3$	
$\cancel{T_2 \rightarrow T_3 \rightarrow T_1}$	
$T_3 \rightarrow T_1 \rightarrow T_2$	
$T_3 \rightarrow T_2 \rightarrow T_1$	

6 → Now, which case?
Cases

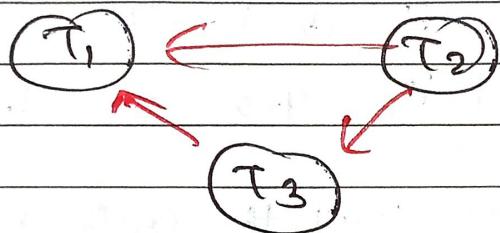
Again, see the graph
→ check

$\boxed{\text{Indegree} = 0}$

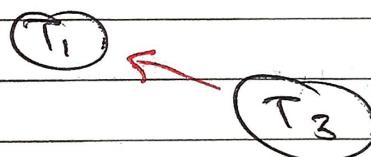


Indegree = 0, means a vertex with '0' indegree.

Mean, $\sqrt{1}, \sqrt{2}$ चीजे ग्र लेज (arrow) नाही आणि रुदा दै.



T_2 has, Indegree = 0,
Now, remove T_2 .



$T_2 \rightarrow T_3 \rightarrow T_1$ ~~is~~

T_1	T_2	T_3
	o	
o		o

— x — . — x —

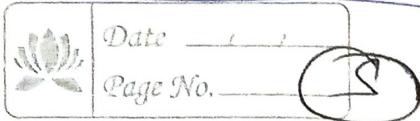
(83.)

View Serializability : →

Q:- Check whether schedule is conflict serializable or not?

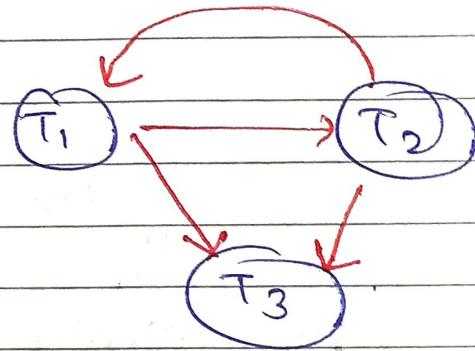
→ First make precedence graph.

Loop \rightarrow No Answerable, then, view Serializable Answers.



S

T ₁	T ₂	T ₃
R(A)		
	w(A)	
w(A)		w(A)



→ Here, we get a loop.

So,

It is Non Conflict Serializable.

↳ Here, we can't tell that it Serializable or not.

Now,

To check that, we use View Serializable y.

Now, we arrange this Table,

T ₁	T ₂	T ₃
R(A)		
w(A)	w(A)	
w(A)		w(A)

We change
the begin

T ₁	T ₂	T ₃
R(A)		
	w(A)	
w(A)		w(A)

Now this be a serial schedule, $T_1 \rightarrow T_2 \rightarrow T_3$.

But,

We have to check whether they match each other or not.

With A = 100.



T_1	T_2	T_3
100 R(A)	$A = A - 40$	
$W(A) - 60$		
$W(A)$ $A = A - 40$ (80)		$W(A)$ $A - 20$ (0)

T_1	T_2	T_3
100 R(A)		
60 $W(A)$		
W(A)	W(A)	20 <u>W(A)</u> 0

Now, apply this same here.

Here, finally
A value - 0

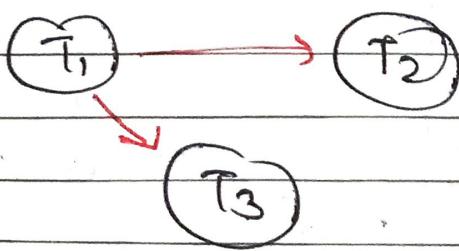
Here also
finally A is '0'

Hence Both are Equivalent.

They are View Equivalent (\equiv).

Note:- b/c, finally output is given by A of T_3 . So, if adjust the pos' of A of T_1 & T_2 . So, there is no problem.

Now,



(By Refining
of only
 T_1)

Hence, $T_1 \rightarrow T_2 \rightarrow T_3$ (By Table).

Note:- Conflict Serializable tell that it is not Serializable (no ans.). But, View Serializable Checks it & tell that it is Serializable.

84.

(parallel schedule)

(C.C.P.)

Concurrency Control Protocols:

(Shared - Exclusive Locking Protocol)

- C.C.P. is that how to make them Serializable, or how to make them recoverable. Schedules are concurrent, but how we can make them serializable & recoverable, this comes under Concurrency Control Protocol (C.C.P.).

→ We achieve this, By using Locking Protocols.

'Shared - Exclusive locking': →

We use 2 locks here,

- Shared lock(s) → if: trans. locked data item in shared mode, then allowed to Read only.

[U → means unlock].

T₁

S(A)

R(A)

U(A)

→ Shared lock.

→ only Read

→ unlock.

- Exclusive lock(x) → if trans. locked data item in exclusive mode then allowed to Read & write both.

T₂

X(A)

R(A)

W(A)

U(A)

→ Exclusive lock.

→ unlock.

Compatibility

Table : -

		Request		S	S	X
		S	X	S	X	X
Grant	S	Yes	No	↑	↑	↑
	X	No	No	↓	↓	↓

Expln :-

S has only R(A)

L X has both R(A), W(A)

so

$\Rightarrow S - S$ (No conflict) b/w R(A)-R(A).

$\Rightarrow S - X$ (Conflict).

R - R

R(A)	R(A)
	W(A)

$\Rightarrow X - S$ (Conflict)

R(A)	R(A)
W(A)	

$\Rightarrow X - X$

R(A)	R(A)
W(A)	W(A)

→ Conflict.

* Problems in S/X Locking : -

- 1.) May not sufficient to produce only Serializable Schedule.
- 2.) May not free from Ir-recoverability.
- 3.) May not free from dead lock.
- 4.) May not free from starvation.



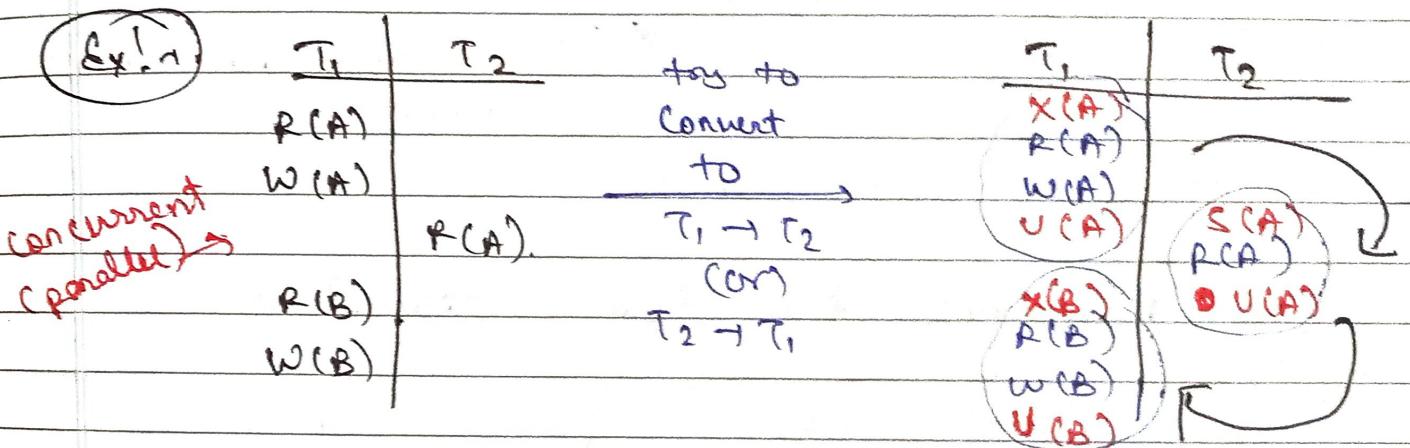
Q1

Q5.

Drawbacks in Shared | Exclusive : S/x locking protocol

→ Locking gives us Serializable Schedule
→ Consistent

i) May not sufficient to produce only Serializable Schedule.
→ It get Hold, Wait [



Note:- Until, we do unlock, U(A) in T₁, we don't be able to put Shared lock S(A) on T₂. b/c by Compatibility Table.

$(X-S) \rightarrow NO$, so first we unlock X(A).

&

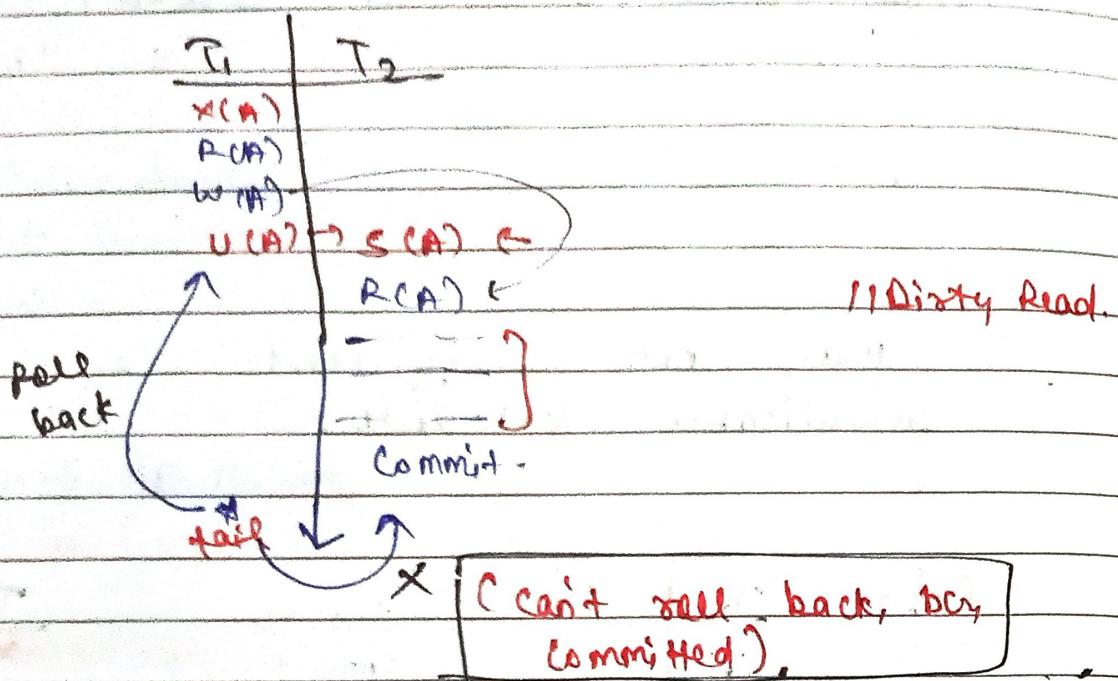
then, we S(A) on T₂ data.

→ Here we don't get Serializable Schedule even after applying S/x locking. As u can see in Table.



2.) May not free from non-recoverability.

Ex:-



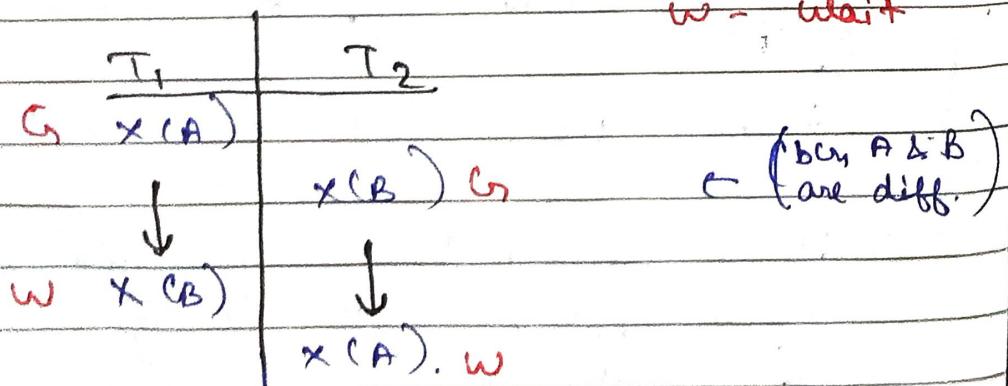
" It can't recover here .

3.) May not free from deadlock.

→ Deadlock! → When 2 person wait for resources, & they both are waiting in an infinite loop, so, when we wait infinitely, it is called Deadlock State.

C - Grant
W - Wait

Ex:-



2) Here, both are in waiting bcz, They are not unlock after use. So,

- T_1 also waits that when T_2 unlocks, he can use on $X(B)$.

So, Both are waiting in an Infinite loop

- 4.) may not free from starvation.

In deadlock, waiting upto infinite time

But,

In starvation, waiting not upto the infinite time.

- Shared in SFR Shared & Tight by compatibility Table. Without unlock.

<u>T₁</u>	<u>T₂</u>	<u>T₃</u>	<u>T₄</u>	
	S(A)			
w X(A)				
Waiting Time.	{ ?	U(A)	S(A) { G	S(A). { G
			U(A)	U(A).

So, here, T_1 waits for $x(A)$ until T_2 unlocks.

80. here starvation also. (बेर शारद द्वारा
Exclusive, तो नि उल्लं ग्रन्त नहीं होता।

86.

phase locking (2PL) protocol in
Transac" Concurrency Control :

→ 2PL (2 phase locking) is just the extension of simple shared/exclusive locking.

We just do modifications in them.

#

2-Phase Locking (2PL) :

→ Growing phase : → locks are acquired
↳ no locks are released.

→ Shrinking phase : → locks are released
↳ no locks are acquired.

TS/XS

T₁
X(A)
S(B)
R(A)
W(A)
R(B)
S(A)
R(C)
S(D)
R(D)

} Growing phase

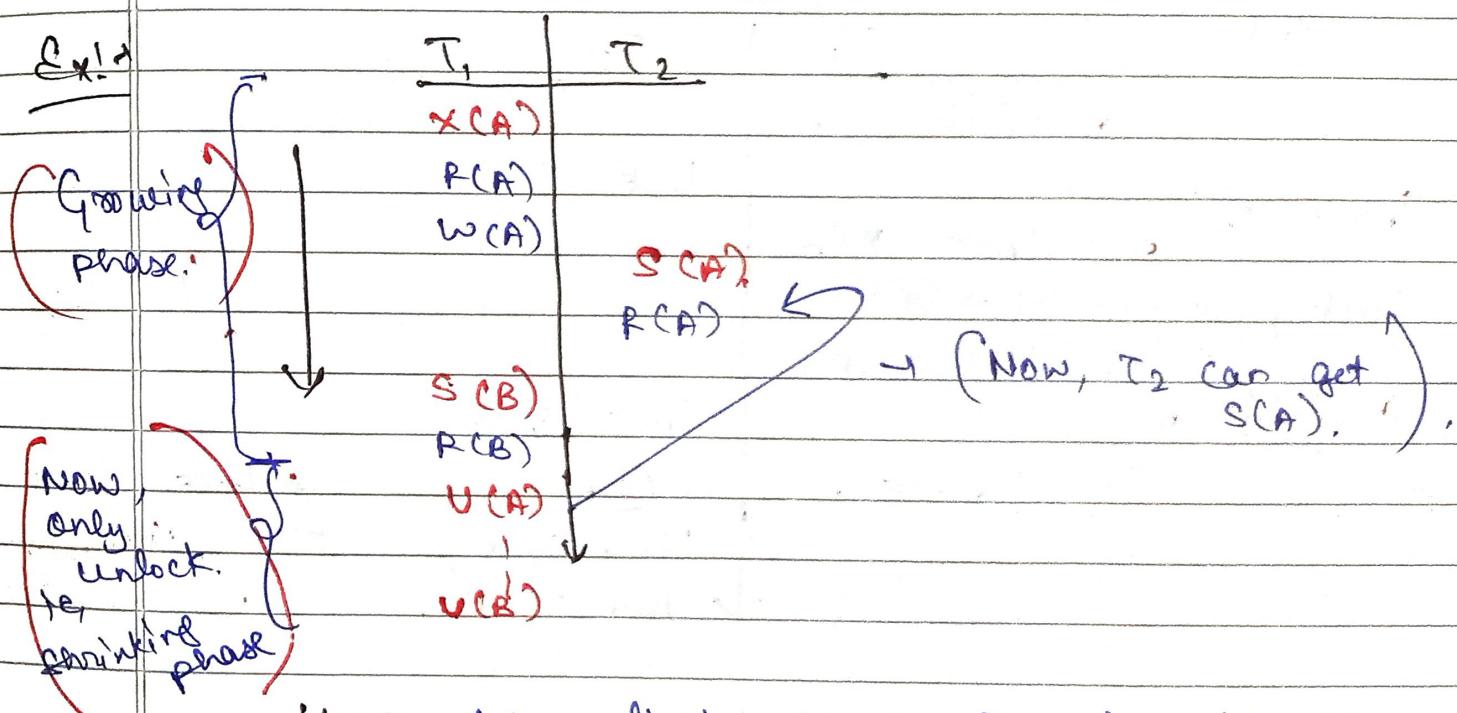
U(A). ← Shrinking. } } Shrinking phase

Note! When Growing phase starts, then only apply locks.

[and],

When we first unlock, i.e., shrinking phase starts, then we only unlock. (and not able to apply any lock).

- ⇒ We achieve Serializability by this, that we don't achieve in simple Shared/Exclusive protocol.
So, we made (2-PL) protocol for this.



Hence, we first starts T_1 , & if T_2 comes in b/w then we don't entertain him. First, we complete T_1 & then goes to T_2 .

$$T_1 \rightarrow T_2$$

Serializability Achieved.

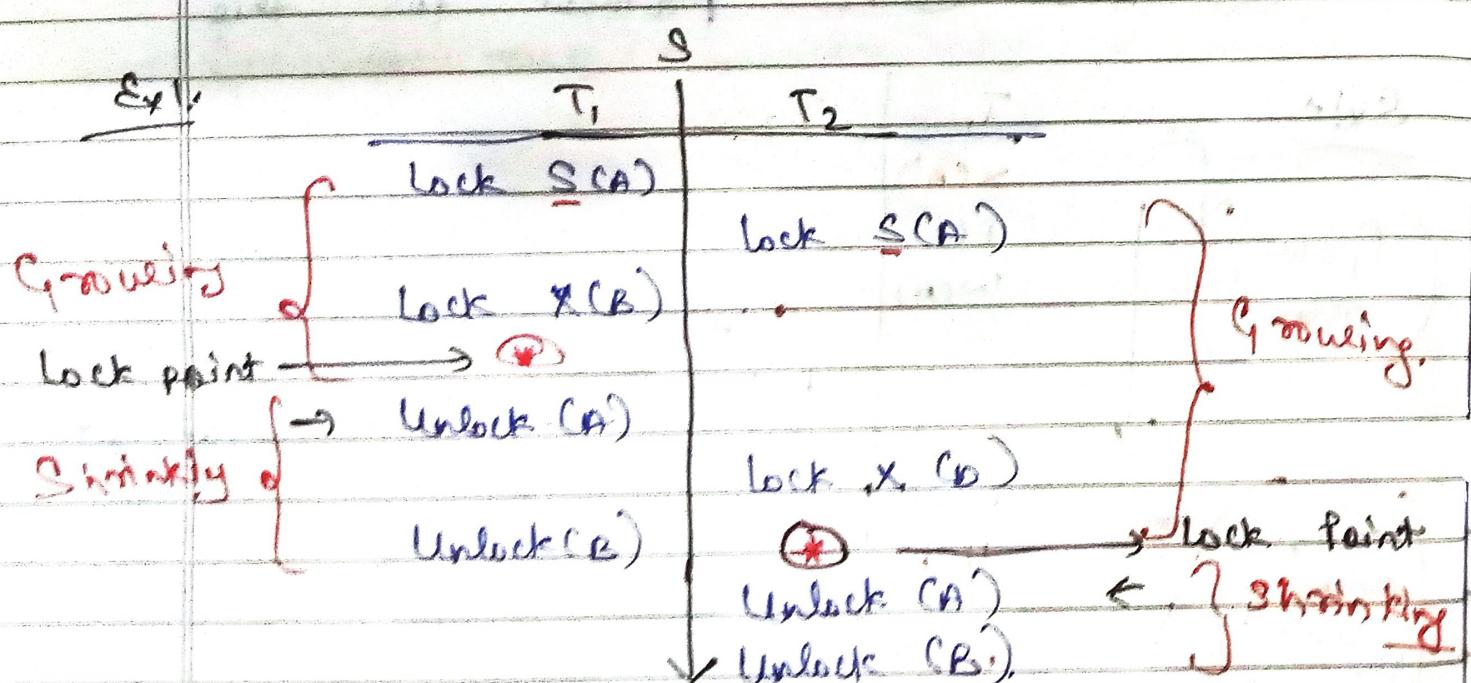
Consistency automatically achieved.



Note: If the transaction which follow (2-PL), then it is always serializable.

Ex: 1) If two get Shared at same Shared at time it will ~~not~~ Means,

While T_1 is in growing phase by using S(A) [Shared Lock], then at same time T_2 also starts its growing phase by using S(A). See by Compatibility Table.



Now,

* Serializability Schedule, How formed?

i.e.

$$T_1 \rightarrow T_2 \text{ (or)} \\ T_2 \rightarrow T_1$$

So,

This is done by lock point.

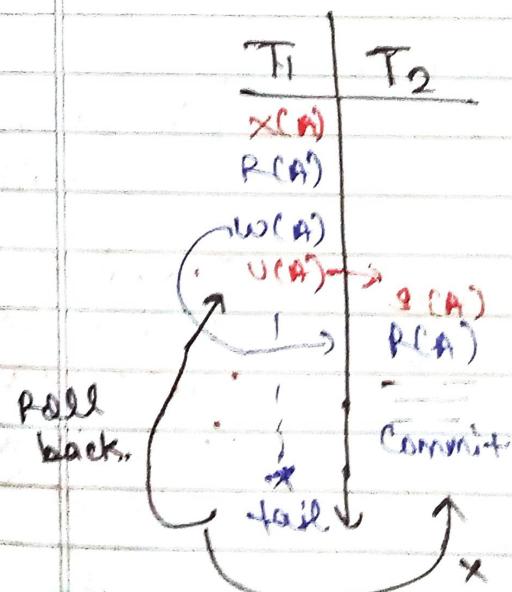
Commit in STG Roll back. ~~exit~~ on the right.

Date _____
Page No. 15

- ⇒ Lock Point: → where trans. is taking the last lock. (or)
where trans. is unlock first time.
- ⇒ Forward Lock Point ~~exit ST MIDL~~ | at trans.
~~exit ST MIDL~~,
i.e., $T_1 \rightarrow T_2$

(87) Drawbacks in (2-PL) protocol : →

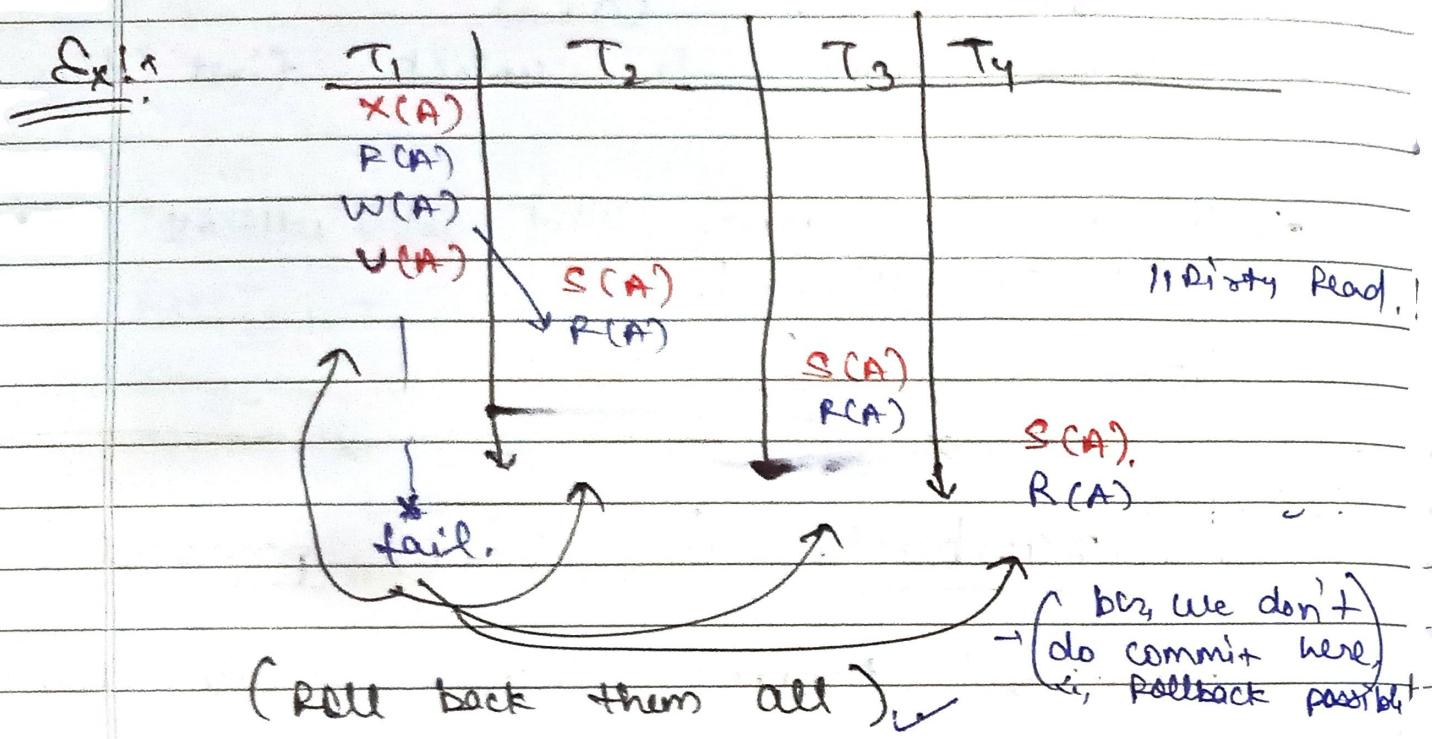
- Advantage: Always ensures Serializability.
- ⇒ Drawbacks: →
 - (1.) May not free from Sr-recoverability.



Hence,

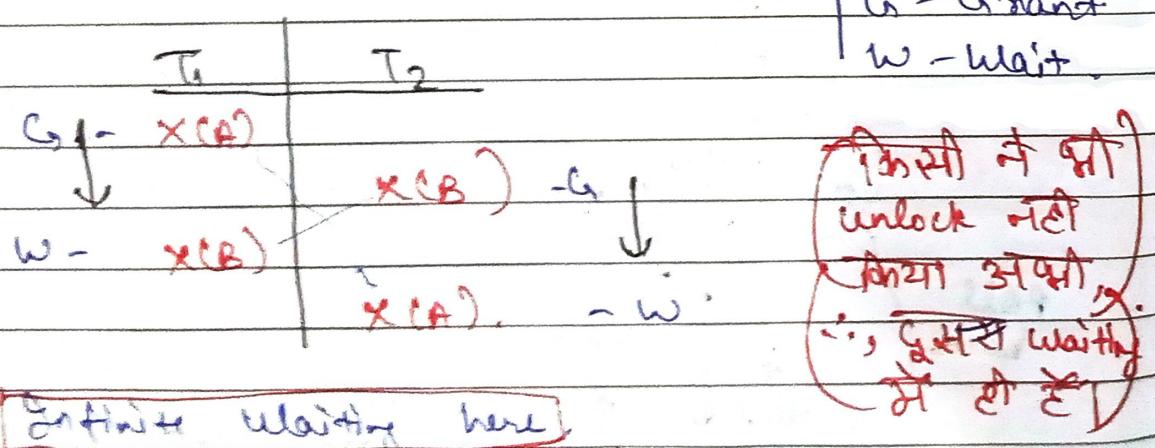
It is Sr. of Sr-recoverable Schedule.
We can't recover it (Roll back it).
(normal).

(2.). Not free from Cascading rollback \rightarrow .



\rightarrow Bad performance. \rightarrow Cascading Rollback is possible here.

(3.) Not free from Deadlocks.



(T_1 & T_2 both waiting here to complete their transac's.).

(4.) Not free from starvation.

(Wait for limited time).

in 2PL

Here, the problem of serializability

T ₁	T ₂	T ₃	T ₄
	S(A)		
w — x(A),	!	s(A)	
	u(A)	!	s(A)
		u(A)	!
			u(A).

Wait

88. Strict 2PL, Rigorous 2PL & Conservative 2PL Schedule \rightarrow

These are the Extension (Enhancement) of 2PL \rightarrow basic.

Strict 2PL \rightarrow

It should satisfy the basic 2PL and all exclusive locks should hold until commit/abort.

Rigorous 2PL \rightarrow It should satisfy the basic 2PL and all shared, exclusive locks should hold until commit/abort.

T ₁	T ₂	T ₃
x(A)		
p(A)		
w(A)		
	u(A)	s(A)
		r(A)
unlock	fail	
roll back		

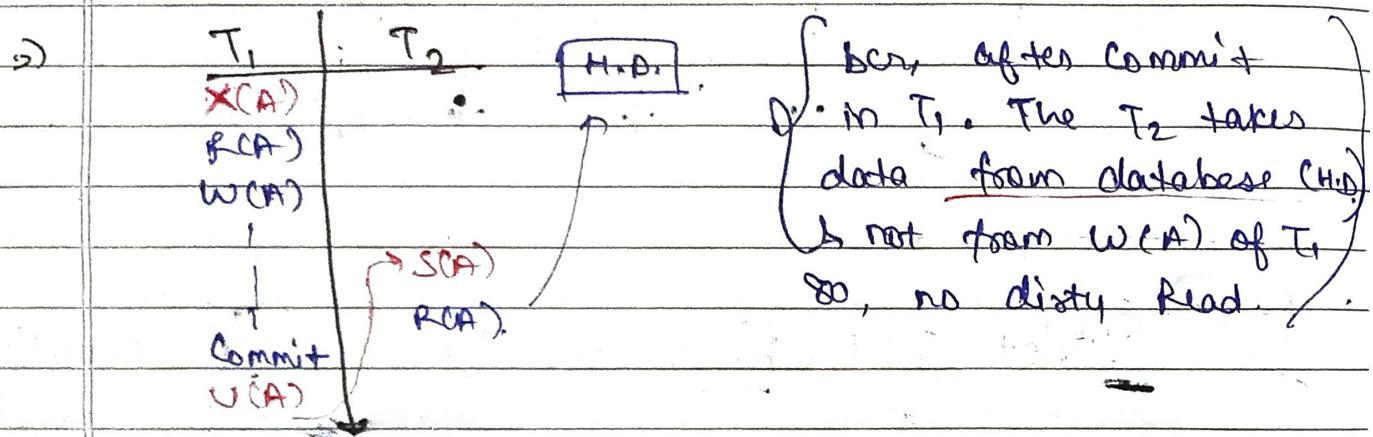
i.e. (Cascading roll back)

\rightarrow (also have to roll back T₂ & T₃)

So, To Stop this \rightarrow

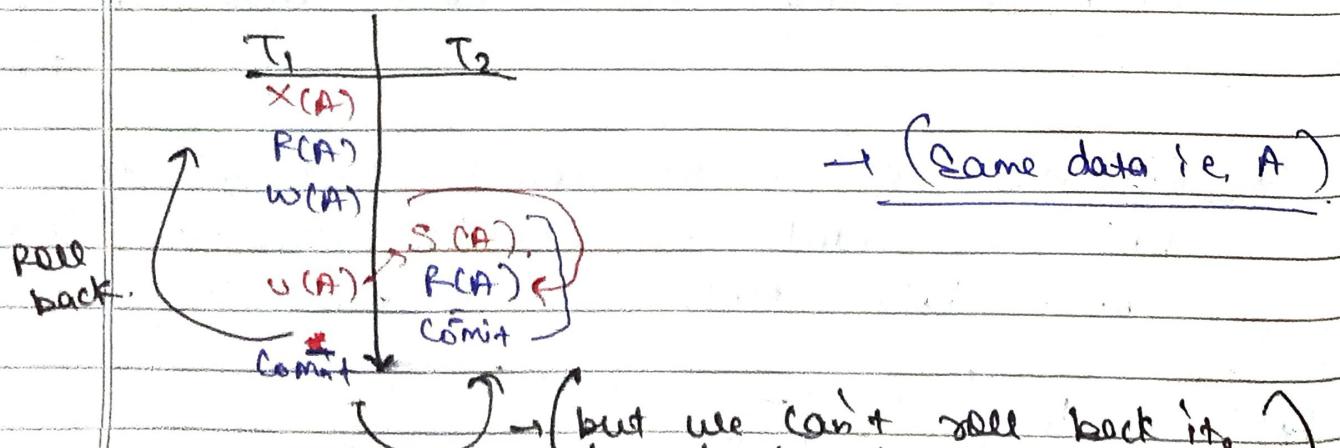
\rightarrow We have to unlock Exclusive lock after Commit.

then, this problem of Cascading Semantics.



Hence, Here, Cascading Rollback is Removed,
 Δ It will always produce Cascadeless.

\Rightarrow Now, Δ Ir-recoverability! \rightarrow



\therefore , Ir-recoverability.

\rightarrow Hence, Δ Right way unlock it with $commit$.

We unlock it after commit is done.