authentication → insert card → account choose → amt. of money give

Transaction: set of operations used to perform a logical unit of work.

→ money transaction

↳ Represent change in db

Read → access, write → modify / change

User to DB

in dB ↑ Read, write will be in RAM] higher speed

ALU will do operation

→ after commit, go to harddisk / dB

ACID Properties:

Atomicity - if fail in even $(n-1)^{th}$ ~~oper~~ step, rollback to the start.

either all or none

restart, not resume

Consistency — Before start and after completion

sum → same

Before sum (A+B) = 2k+3k = 5k

after → 1000 + 4000 = 5k

T — A = 2k, B = 3k

R(A) 2000

A = A - 1000

W(A) 1000

R(B) = 3000

B = B + 1k

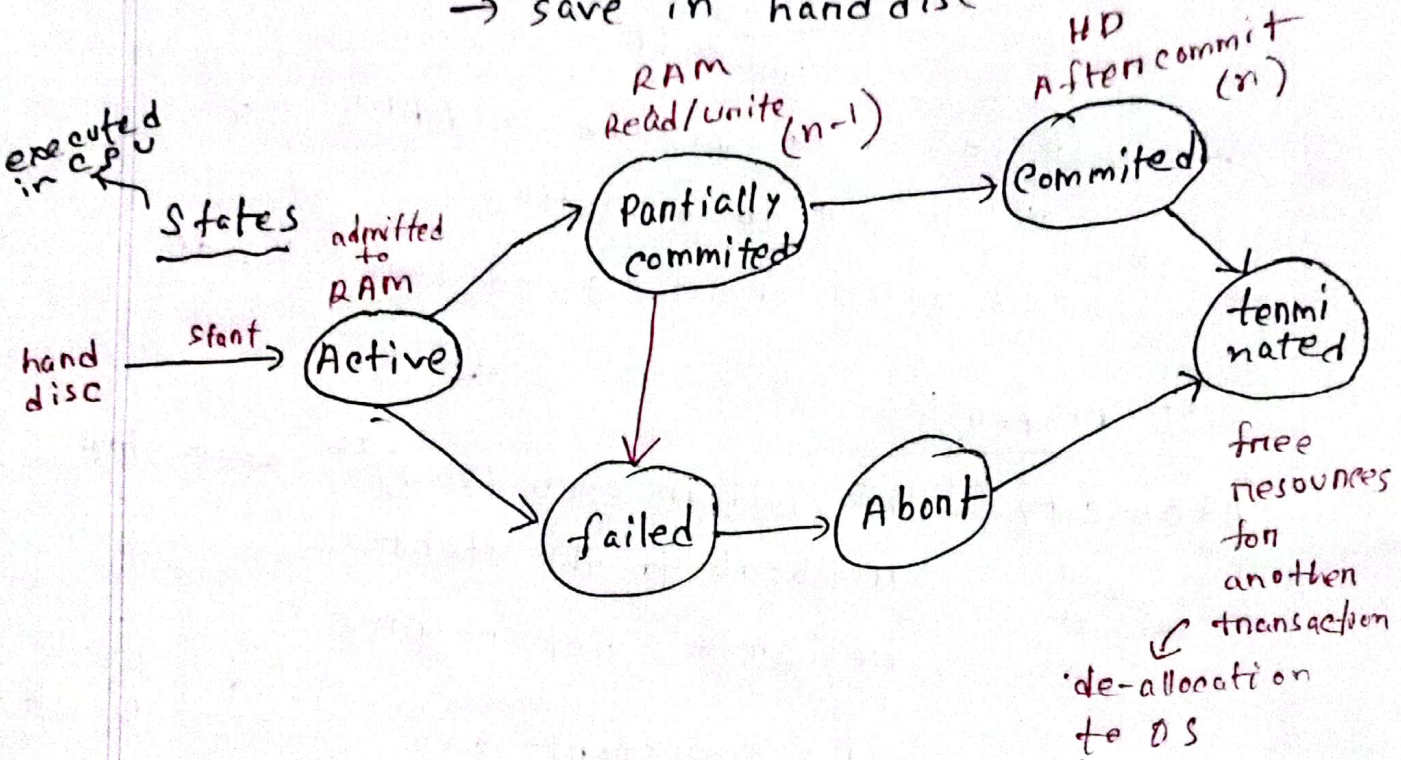
W(B) = 4000

Isolation: if panallel schedule can be converted to senial schedule, conceptually

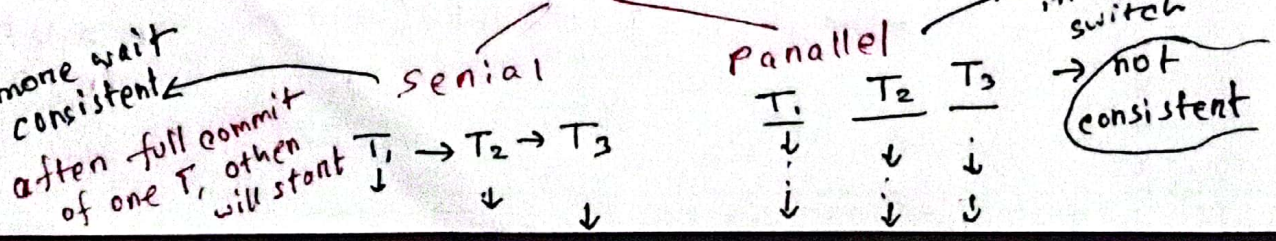
senial $\xrightarrow{\text{always}}$ consistent

Durability: permanent change

→ save in hard disc



Schedule: chronological execution sequence of multiple transactions.



less in serial

Parallel : pros more throughput

↓ no. of t exe. / time

↗ parallel

Concurrency problems (5)

1) Dirty Read / Uncommited read on RAW

↓ Read After WRITE

T_1	T_2
100 R(A)	
50 W(A)	R(A) 50 // Dirty read
⋮	W(A) 40
	commit
Fail	

2) In connect summary :

T_1	T_2
A R,W	Avg (A,B) → inconnect ans.
⋮	
B R,W	

3) Lost update :

multiple users update same thing

→ one update before other ~~update~~ commit

4) phantom read :

T_1	T_2
10 R(x)	R(x) 10
X Del(x)	R(x) X

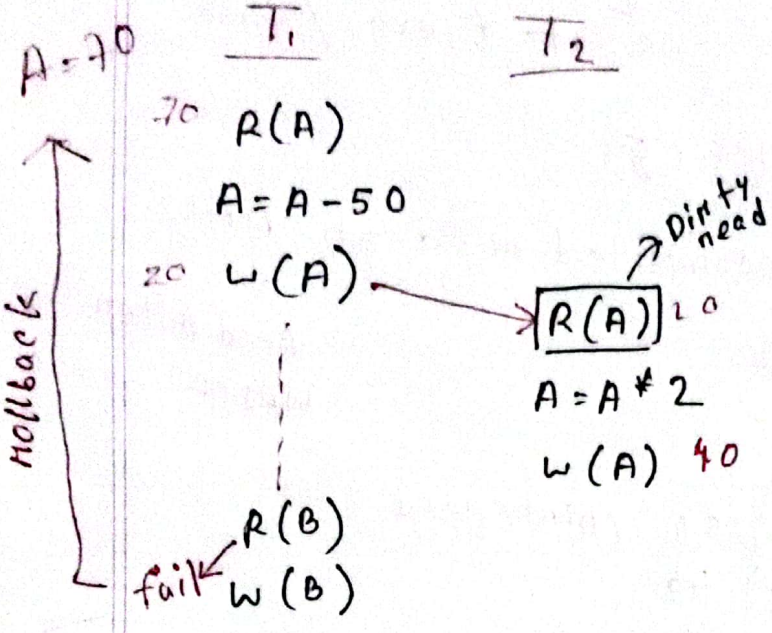
5) Unrepeatable read :

T_1	T_2
10 R(x)	R(x) 10
100 W(x)	R(x) 100

↓ Unrepeatable read

W → R

Write - Read conflict (Dirty Read)

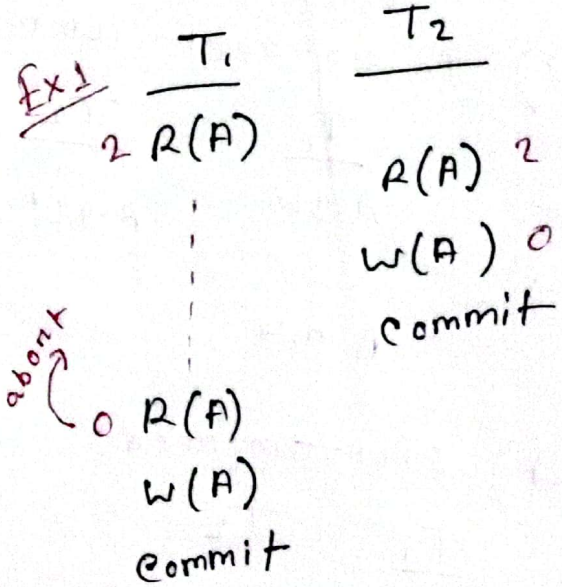


Read - Write conflict (Unrepeatable read)

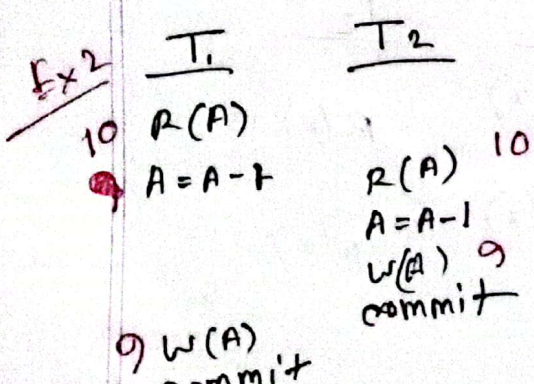
Same data

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

Ex1



Ex2



A = ৬ ৪ ৭ 10

Unrecoverable schedule: value can't be recovered

T_1	T_2
10 R(A)	
A = A - 5	
5 W(A)	
	R(A) 5
	A = A - 2
	W(A) 3
	commit
R(B)	change done by T_2 not recovered
fail	

Rollback

Cascading schedule:

T_1	T_2	T_3	T_4
100 R(A)			
50 W(A)	R(A) 50	R(A) 50	R(A) 50
:			
Fail			

T_1 → roll back so,

~~বাকি~~ 3 T → roll back / abort

→ CPU performance degrade

→ commit এর আগেই বাকি 3 টা T_1 এর read করতে পারে.

Cascadeless:

T_1	T_2	T_3
R(A)		
W(A)	✗ R(A)	✱ R(A)

R(A) not done in T_2 and T_3 untill T_1 done all commit or roll back

once commit → then read

→ Problem: WAW

T_1	T_2
R(A) 100	R(A) 100
W(A) 90	W(A) 80
:	
fail	

roll

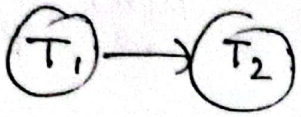
DB, A = 100

T_2 data lost

Cascadeless says, read not happen after write untill commit

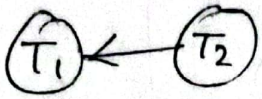
Serializability:

T_1	T_2
R A	
W A	
	R A
	W A



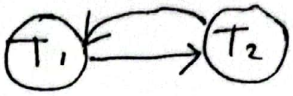
Serial

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



Serial

T_1	T_2
R A	
	R A
	W A
W A	



↳ to serialize make $T_1 \rightarrow T_2$ or $T_2 \rightarrow T_1$ if possible, it's serializable

2 types:

1) conflict seriazability
2) view "

Conflict Equivalent:

non conflict ← R(A) – R(A)

conflict:
- R(A) W(A)
- W(A) R(A)
- W(A) W(A)

non conflict:
- R(B) R(A)
- W(B) R(A)
- R(B) W(A)
- W(A) W(B)

S

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

S2

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

Step: adjacent non-conflict pair check

→ if non-conflict, swap them

S

T_1	T_2
RA	
WA	
	RA
	WA
RB	

swap

→

T_1	T_2
RA	
WA	
	RA
RB	
	WA

swap

→

T_1	T_2
RA	
WA	
RB	
	RA
	WA

same as S2

∴ S and S2 conflict equiv.

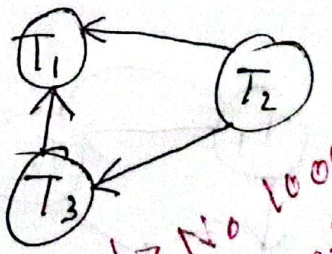
→ so serializable

* Conflict serializability:

– check conflict pairs in other transactions and draw edge

– precedence graph

T_1	T_2	T_3
~~RX~~		~~RY~~
		~~RX~~
	~~RY~~	
	~~RZ~~	WY
	~~WZ~~	
RZ		
WX		
WZ		



↳ No loop / cycle

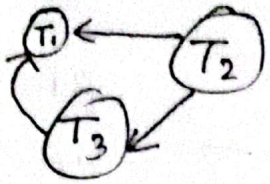
→ conflict serializable

→ serializable

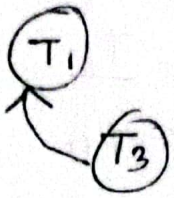
→ consistent

To make it [serializable.]

→ find the node with indegree 0.
→ remove the node and its edges
(again)



T_2



T_3

T_1

∴ serial sequence : $T_2 \rightarrow T_3 \rightarrow T_1$

Why view serializability is used?

→

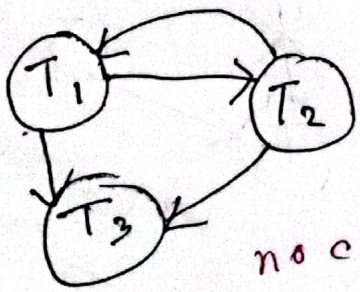
T_1	T_2	T_3
100 ~~RA~~	-20	
-20	~~WA~~ 60	
80 ~~WA~~		-20
		WA 40

→

View:

T_1	T_2	T_3
RA		
-20		
80 WA	-20	
	WA	-20
	60	WA
		40

Both final ans - same
→ view equivalent
→ serializable



no cycle but loop has

→ non-conflict serializable
↳ can't know if serializable or not