R B R

Account(Cid , bal)

Set bal of Cid 101 , Cid 102 to 5000

In SQL:

```
begin
    update Account set
    bal = 5000 where Cid = 101
    update Account set
    bal = 5000 where Cid = 102
end.
```

low level execution:

begin transaction T_1

write (bal: 5000 where Cid = 101) → w(Sal)

write (bal: 5000 where Cid = 102) → w(Sal)

commit

end transaction

Read (Sal) → r(sal)

~~Sal = S + 100~~

write (Sal) → w(sal)

Eg of a transaction:

Trans (T_1)

- R(A) ✓
- R(B) ✓
- W(A) ✓
- W(C) — Blind writes. Because C and D values are not read. They are directly written.
- W(D) — Blind writes. Because C and D values are not read. They are directly written.
- Commit

ACID properties: To preserve integrity (correctness) of transactions

- ✓ A – Atomicity
- ✓ D – Durability

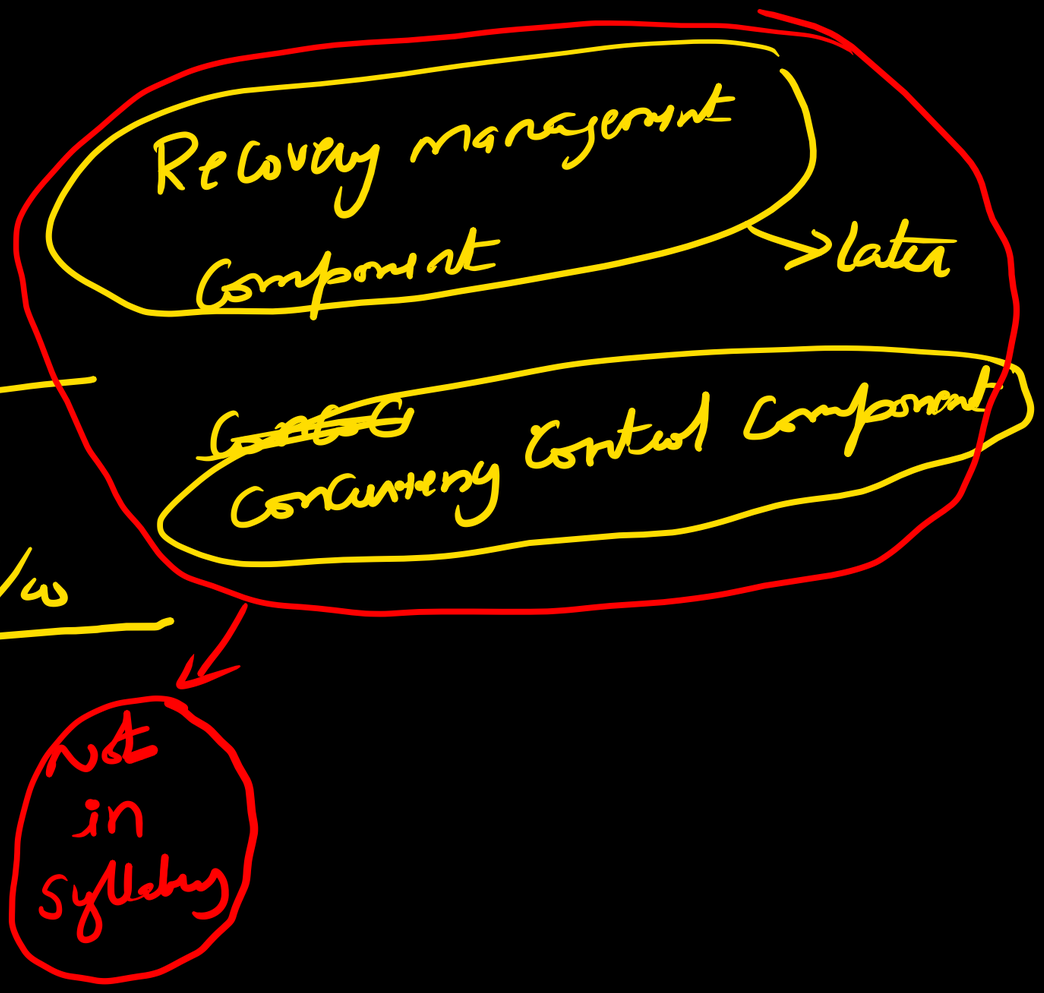
} maintained by Recovery management Component of DBMS S/w

- ✓ I – Isolation

} maintained by Concurrency Control Component of DBMS S/w

- ✓ C – Consistency

} maintained by user
DBA / DB dev / DB user.



Atomicity: Execute all operations of transaction including commit
(or)
execute none of the operations of transaction

All or none.

1
2
3
4

Ex: There is a transaction to transfer 500 from acc A to acc B.

Trans (T_1)
R(A)
A = A - 500
w(A)
R(B)
B = B + 500
w(B)
Commit

failure → 500 is deducted from Acc A. But 500 is not added to acc B

failure leads to incorrect data. To preserve integrity we will used atomicity and do a roll back.

Recovery manager is responsible for succesful roll back of a faild transaction.

Recovery management Component:

It rolls back transaction if transaction faild anywhere before commit.

Roll back (& abort): undo modifications of DB file which are done by faild transaction

Q) How do recovery manager perform rollback?

A) Transaction log

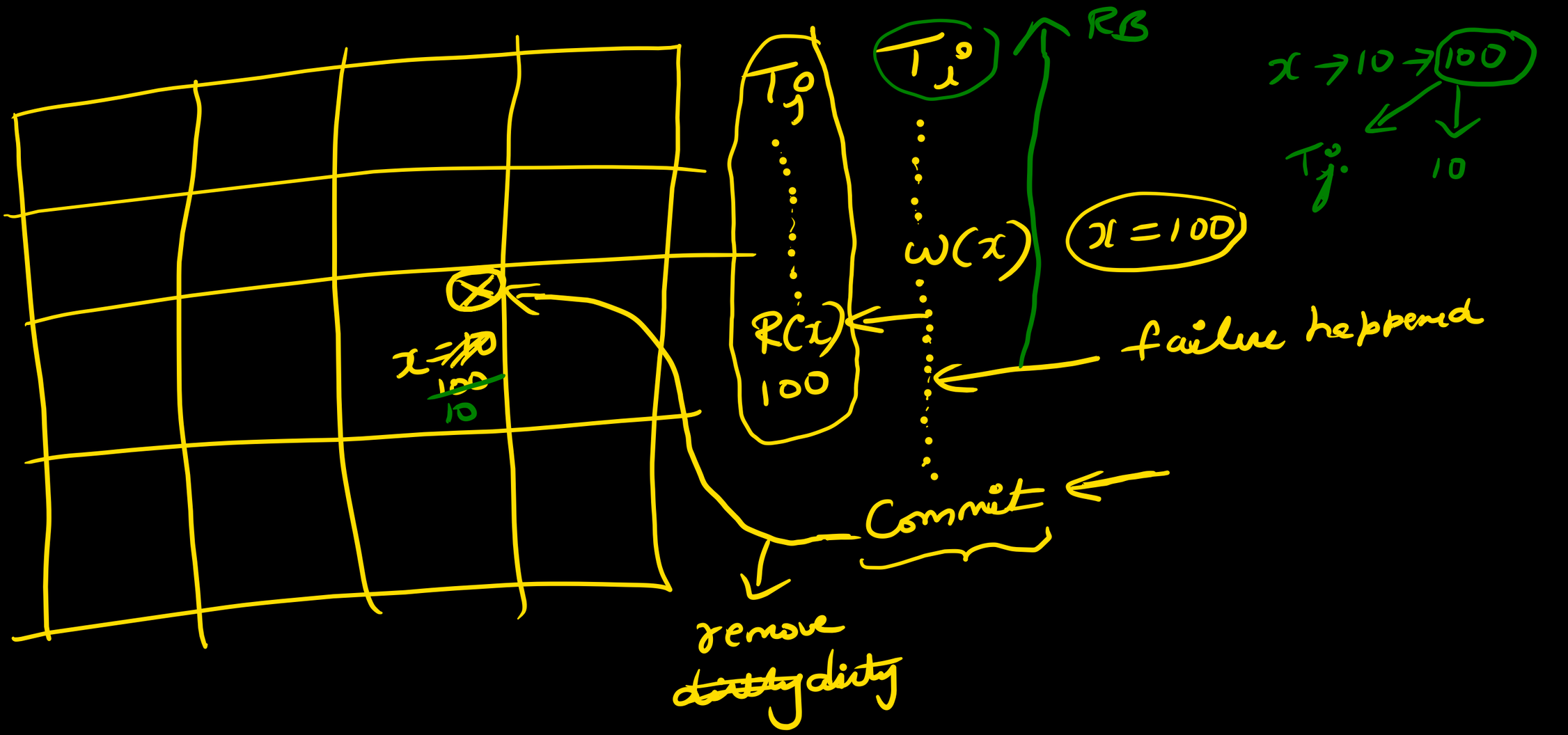
Transaction log:

Transaction log

1.	T_1 begin
8.	T_1, R, A, 100
3	T_2 begin
4	T_2, W, B 1000

} every action of any transaction is logged into transaction log

Transaction log is a file maintained by recovery manager in the Disk to record all actions of transactions. For roll back, Recovery management component will see transaction log and do it.


RB
T_j
T_i
$x \rightarrow 10 \rightarrow 100$
T_j
10
$x = 100$
W(x)
R(x)
100
failure happened
Commit
remove
dirty

A Block that is updated by a uncommitted trans is a dirty block.

Why is it dirty? Because there is a chance of roll back and some other transaction might read the uncommitted data.

(we should not let any transaction access dirty data)

T_1

R(w)

~~write~~ R(w)

R(x)

w(x)

Commit ← once commit is reached there is no roll back

✓ Redo operations:

All modifications that will be done by because of commit. ✓

(Ex: removing of dirty flag of bit, flushing of data)

↓

CPU writes.

undo operation: ✓

All modification of OS file

Lecog of transation undo

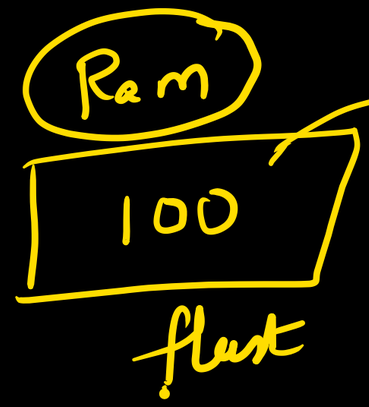
Ex:

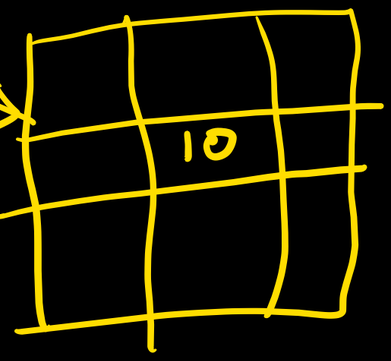
(T1)

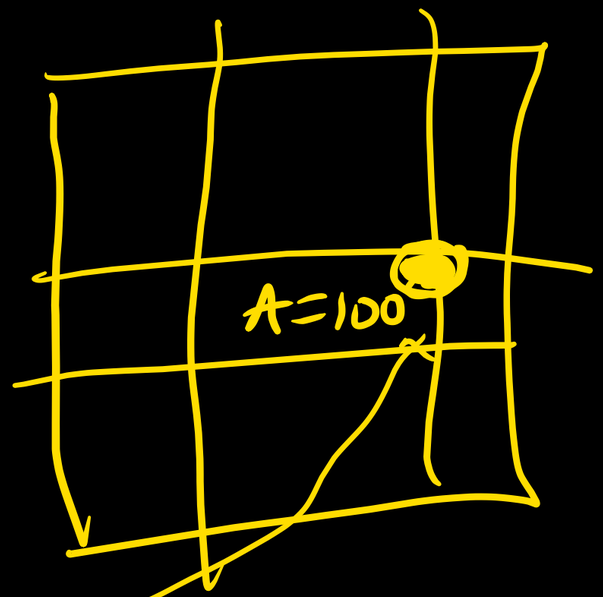
R(A) → 10

W(A) A=100

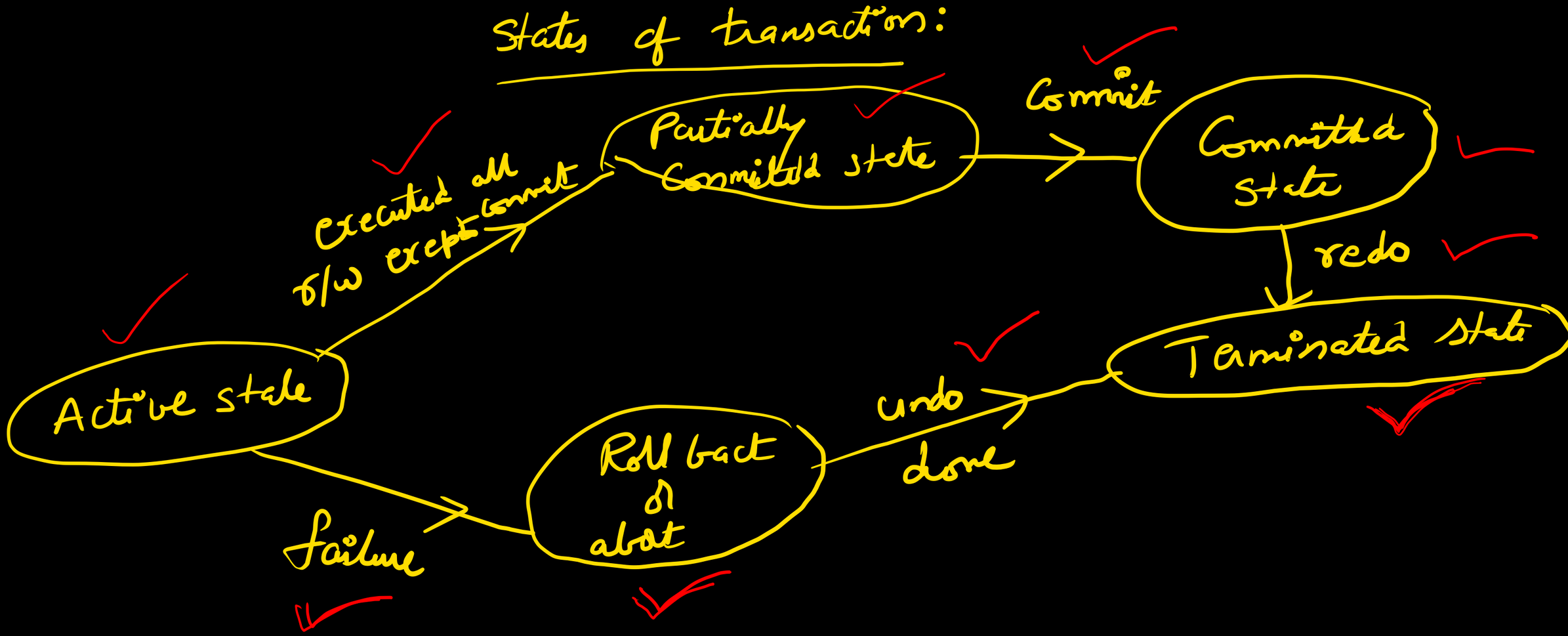
Commit







States of transaction:



Atomicity ends here

Durability :

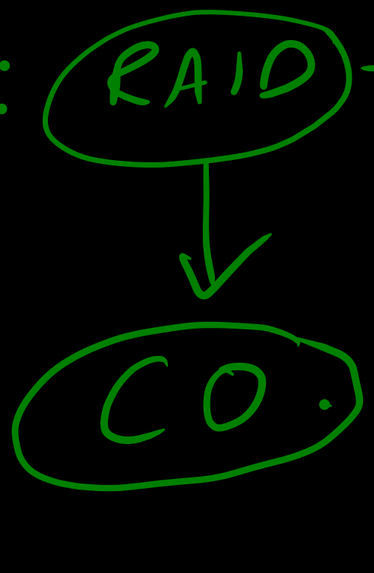
Transaction should recover under any case of failure.

Transaction may fail because of following.

- Power failure → not serious
- S/w Crash
  - OS overload
  - DBMS overload
- Concurrency control protocol may kill transaction → not serious
- H/w Crash (Disk Crash) → serious problem.

If disk fails → then transaction log will be lost.

Sol: RAID → Redundant array of independent disks.

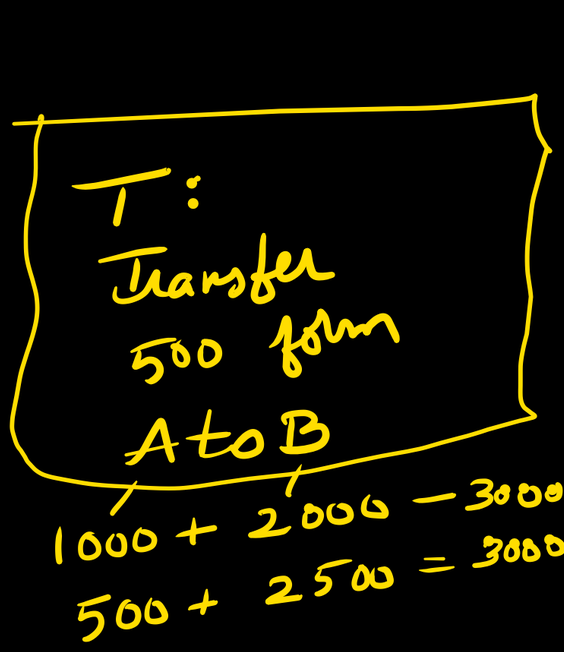


Durability done

Consistency:

This is user's responsibility.

User should check if the operations of a transaction is logically correct.



A+B before execution of Transaction

same

~~A=B~~ A+B after execution of transaction

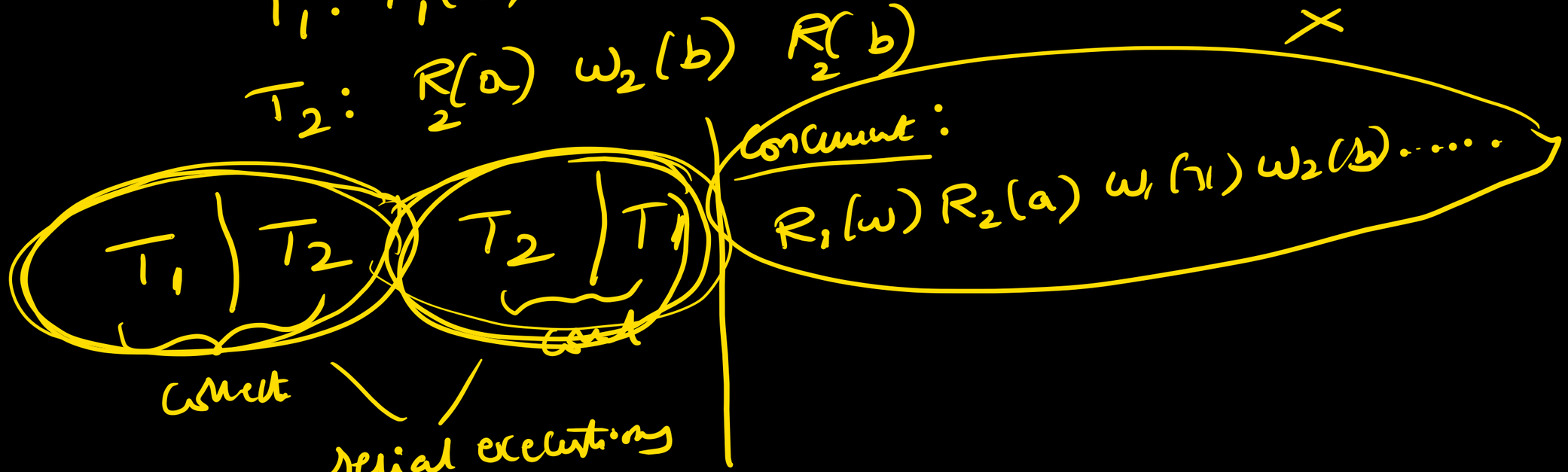
Consistency is over

Isolation: **** important ft gate

Concurrent execution of two or more transactions must be equal to result of any serial execution of transactions

T_1: $R_1(w)$ $W_1(x)$ $R_1(y)$ $W_1(y)$

T_2: $R_2(a)$ $W_2(b)$ $R_2(b)$



Schedule: Time order execution sequence of two or more transactions.

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

✓ ✓ S: $R_2(A), R_1(A), R_3(B), W_2(A), W_3(B), W_1(A)$

(2) → 2nd transaction

Trans: group of actions which are trying to achieve some task

Schedule is group of ~~interleaved~~ transactions

S :

T_1	T_2	T_3
	$R_2(A)$	
$R_1(A)$		$R_3(B)$
	$W_2(A)$	
$W_1(A)$		$W_3(A)$

✓

For a schedule, there can be any no of transactions

Serial schedule:

Transactions should execute one after other without interleaving

T_1 : Transfer 500 from A to B

$[r_1(A)\ \ w_1(A)\ \ r_1(B)\ \ w_1(B)\ \ c_1]$

T_2 : Display A + B

$[r_2(A)\ \ r_2(B)\ \ c_2]$

T_1 : Transfer 500 from A to B

$[r_1(A)\ \ w_1(A)\ \ r_1(B)\ \ w_1(B)\ \ C_1]$

T_2 : Display A + B

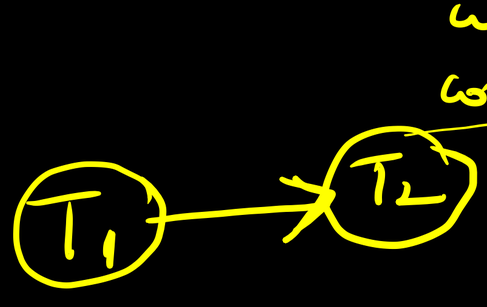
$[r_2(A)\ \ r_2(B)\ \ C_2]$

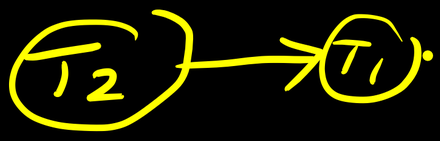
T_1	T_2
$r_1(A)$	
$w_1(A)$	
$\lambda_1(B)$	
$w_1(B)$	
commit	
	$r_2(A)$
	$r_2(B)$
	C_2

$T_1\ T_2$

$T_2\ T_1$

} Serial schedules.





T_1	T_2
	$\lambda_2(A)$
	$\lambda_2(B)$
	C_2
$r_1(A)$	
$w_1(A)$	
$r_1(B)$	
$w_1(B)$	
C_1	

T_1	T_2
$r_1(A)$	
$w_1(A)$	
$r_1(B)$	
$w_1(B)$	
commit	
	$r_2(A)$
	$r_2(B)$
	C_2

$T_1 T_2$

S_1

T_1	T_2
	$r_2(A)$
	$r_2(B)$
	C_2
$r_1(A)$	
$w_1(A)$	
$r_1(B)$	
$w_1(B)$	
C_1	

S_2

$S_1 \cong S_4$

$S_3 \neq S_1$ or S_2.

S_3 S_4

Concurrent Schedules.

S_4 is equivalent to S_1

$S_1 \cong S_4$

$\therefore$ S_4 is equivalent to a serial schedule S_1.

$\therefore$ S_4 is called serializable

T_1	T_2
$r_1(A)$	
$w_1(A)$	
	$r_2(A)$
$r_1(B)$	
$w_1(B)$	
C_1	
	$r_2(B)$
	C_2

S_4

$T_1\ T_2\ T_3$

$T_1\ T_3\ T_2$

$T_2\ T_1\ T_3$

$T_2\ T_3\ T_1$

$T_3\ T_1\ T_2$

$T_3\ T_2\ T_1$

$\&(3!)$.

If there are 10 transactions, then how many serial schedules will be possible

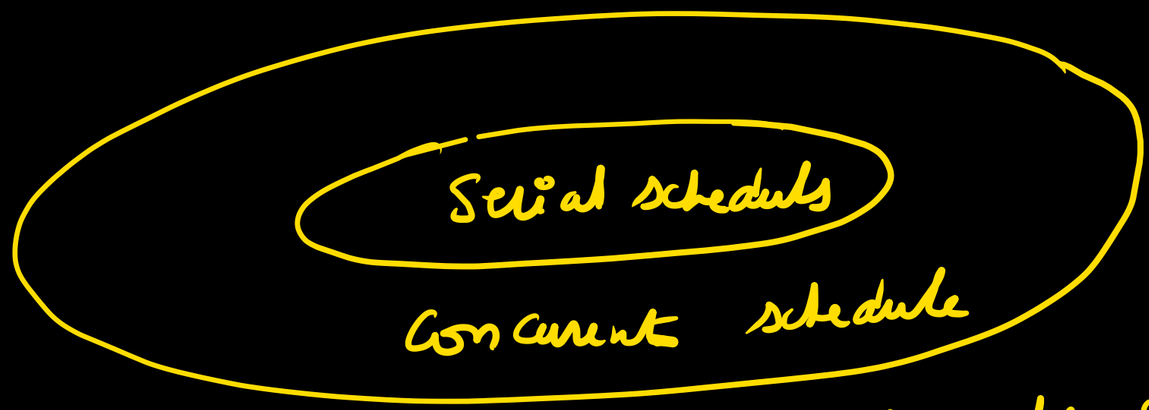
$(10!)$

adv: Data items in a serial schedule are accessed by one transaction at a time, then is no problem of inconsistency.

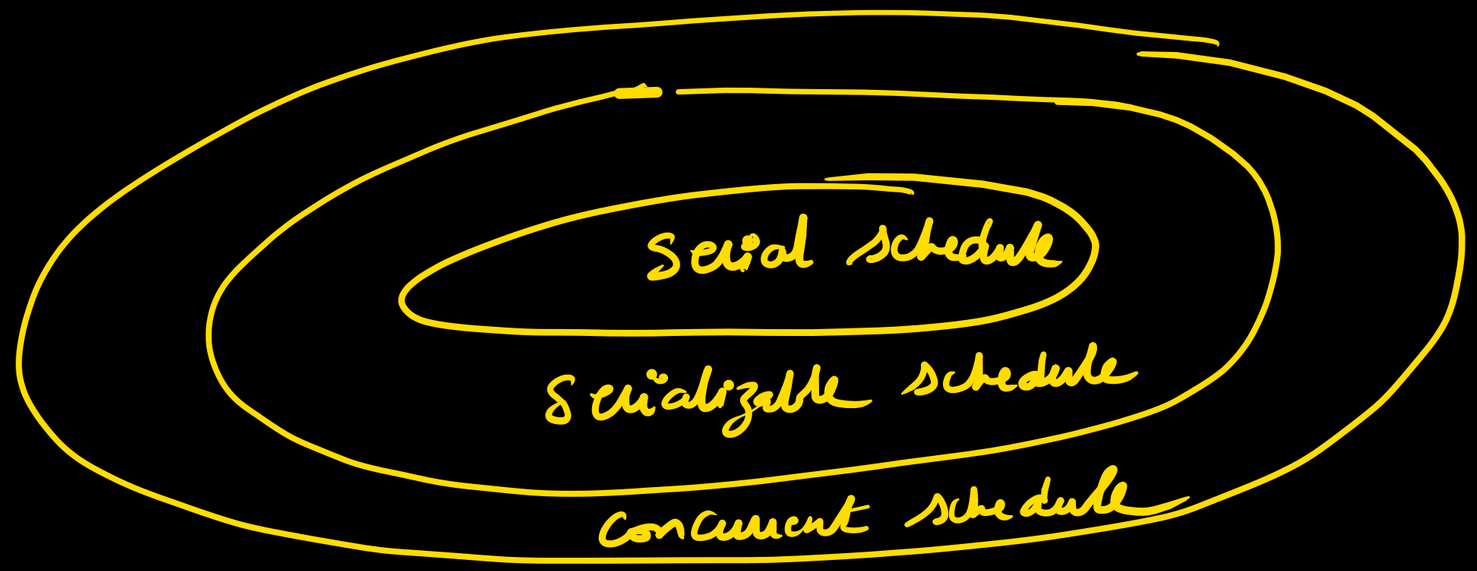
adv of serial schedule

dis: Concurrency doesn't happen, less throughput, no sharing of resources.

Isolation: Concurrent schedule result must be equal to result of any serial schedule.
These kind of schedules are also called serializable schedules.



Every serial schedule can also be called Concurrent Schedules
But ~~all serial schedules~~ there are schedules which
are not ~~so~~ serial.


Serial schedule
Serializable schedule
Concurrent schedule

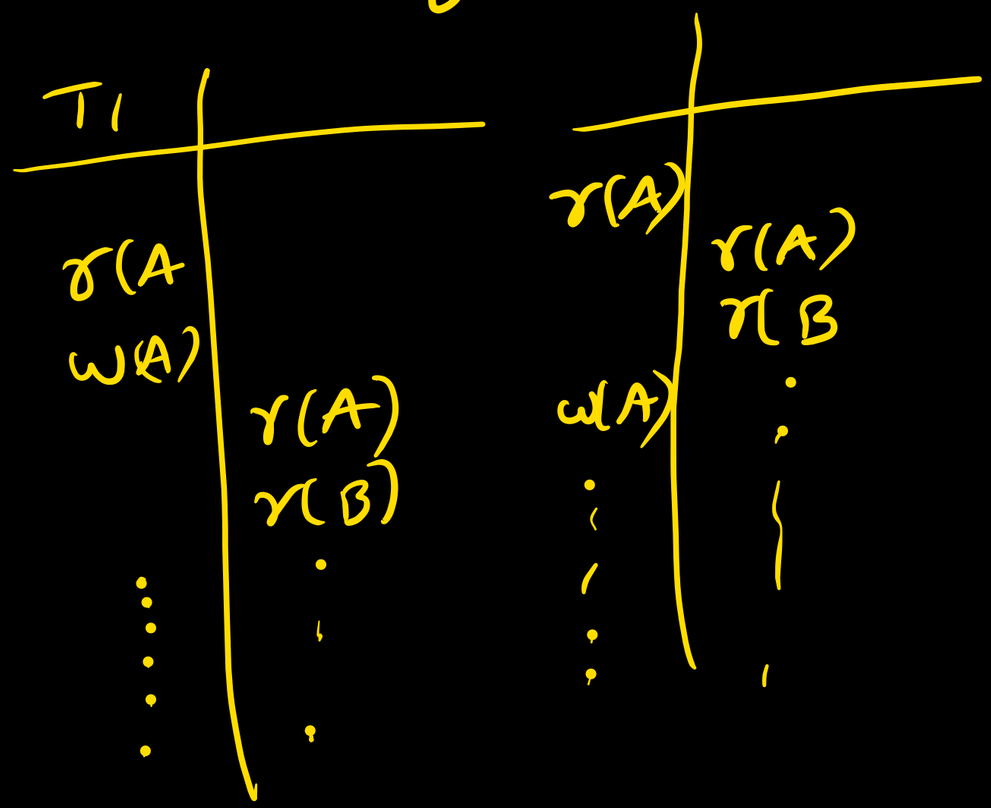
Concurrency control protocol :-

It allows only the concurrent schedules which are equivalent to some serial schedule i.e serializable schedule.

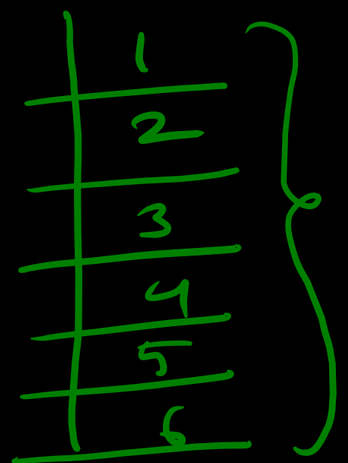
T_1: $r_1(A)\ w_1(A)\ r_1(B)\ w_1(B)$ → no rearrangement

T_2: $r_2(A)\ r_2(B)$ → no rearrangement

How many concurrent schedules are possible



$$\frac{6!}{4!\,2!}$$



64

T_2: $r_2(A)\ r_2(B)$ ⇒ $r_2(B)\ r_2(A)$

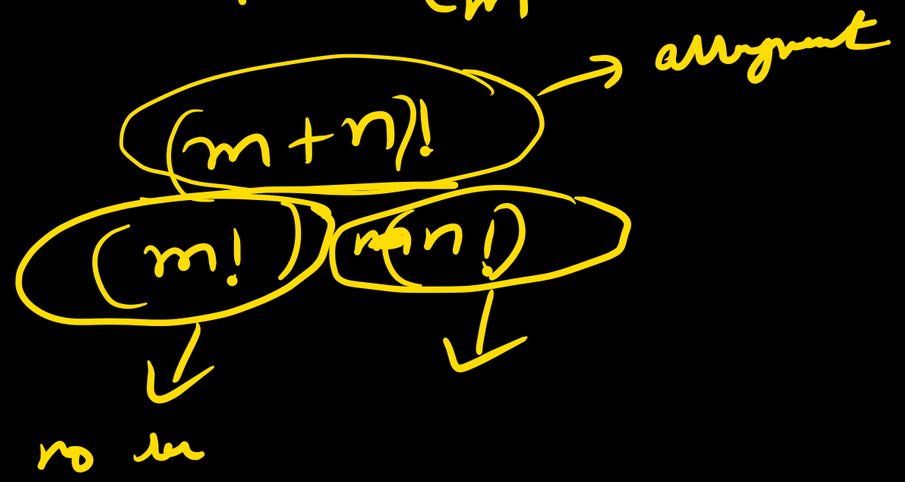
${}^6C_4 \rightarrow T_1 \quad {}^2C_2 \rightarrow T_2$

$${}^6C_4 \times {}^2C_2 = {}^6C_4.$$

→ T_1, T_2 trans with m & n operations each. How many concurrent schedules are possible?

$$^{m+n}C_m * {}^{n}C_n$$

$$^{m+n}C_m$$



T_1, T_2, T_3 are transactions with n, m, p operations each.

How many concurrent schedules are possible?

$$\frac{(n+m+p)!}{n!\ m!\ p!}$$

We have completed ACID properties.