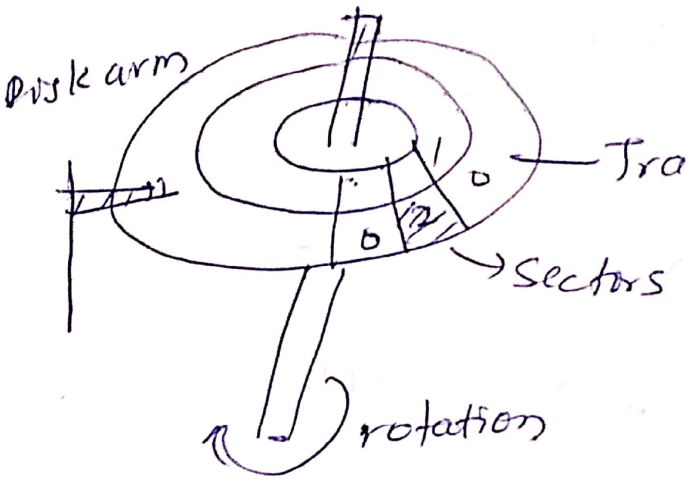
11/11/20

4NF

DBMS :-

Disk structure :-

Each disk platter has a flat, circular shape



Sector = 512 bytes
tracks = 50,000-1
Platter = 1 to 5

Note :- Block
contiguous sectors

R/W in terms of block

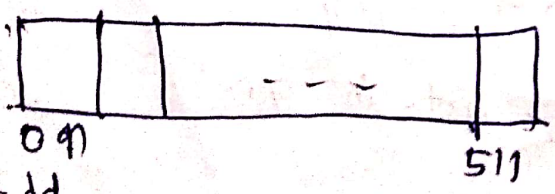
Data storage & Indexing :-

(1) Disk structure
(2) Indexing
(3) B+ tree
(4) Hashing.

=

Block Add = < track No, Sector No >

Block Size = 512 bytes



Student :

SID	SName	Course	Dept	add
1	S1			
2	S2			
3	S3			
4	S4			
5	S5			
6	⋮			
7	⋮			
⋮				
100	S100			

Student :

SID = 10 bytes

SName = 50 bytes

Course = 8 bytes.

Dept = 10 bytes

add = 50 byts

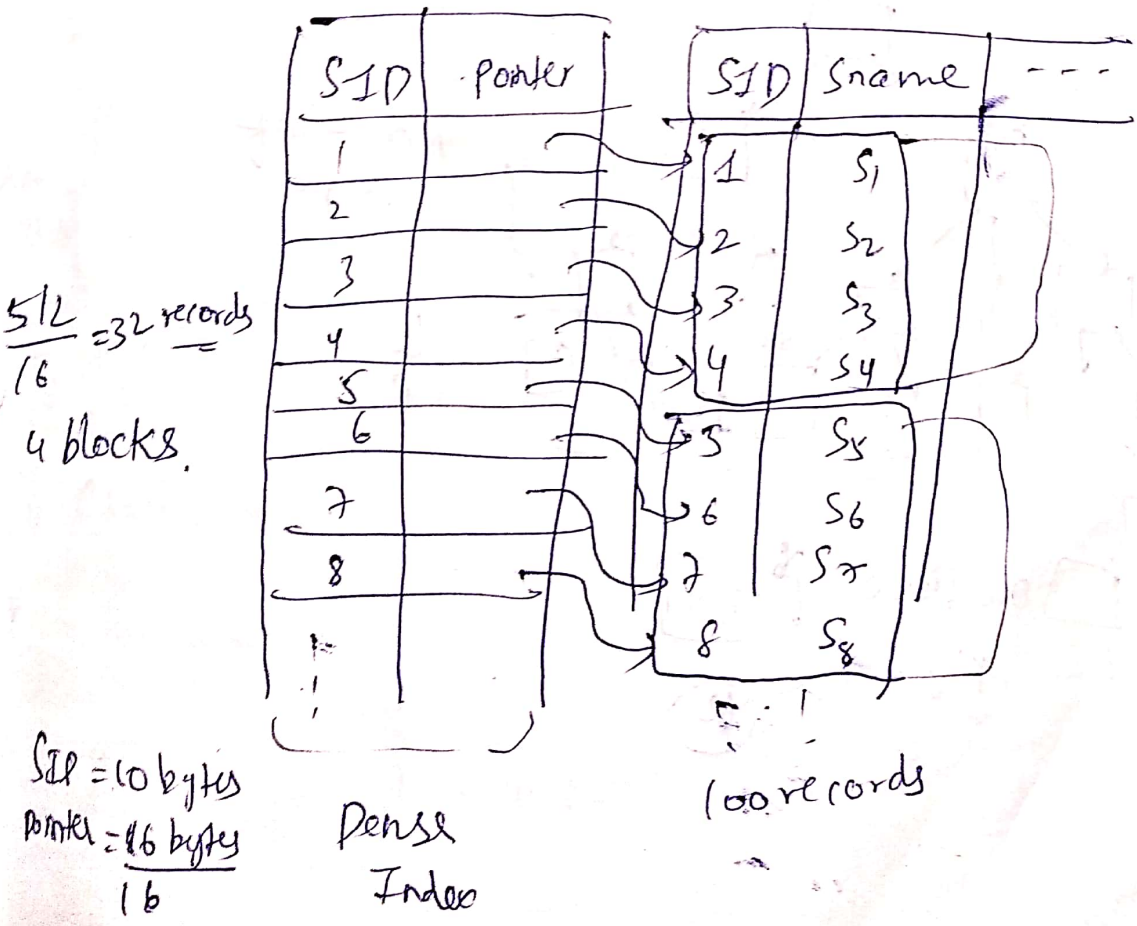
Total = 128 bytes

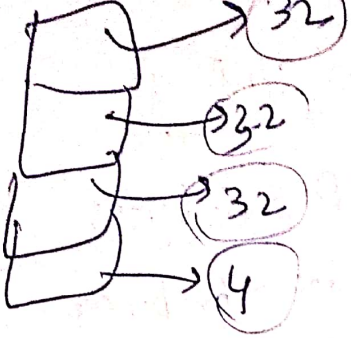
Assume block size - 512 byts

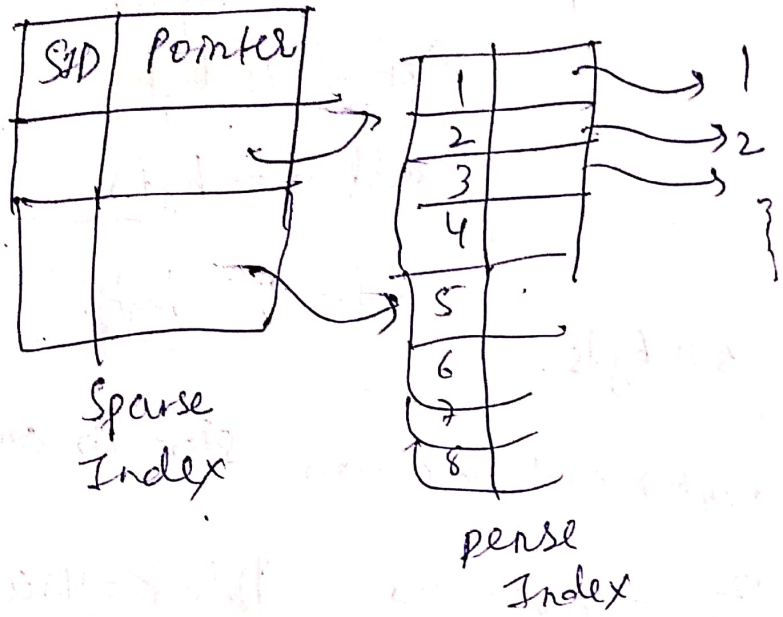
Q) How many student records we can store in one block ?

$\frac{512}{128}$ = 4 records ∴ block = 4 records

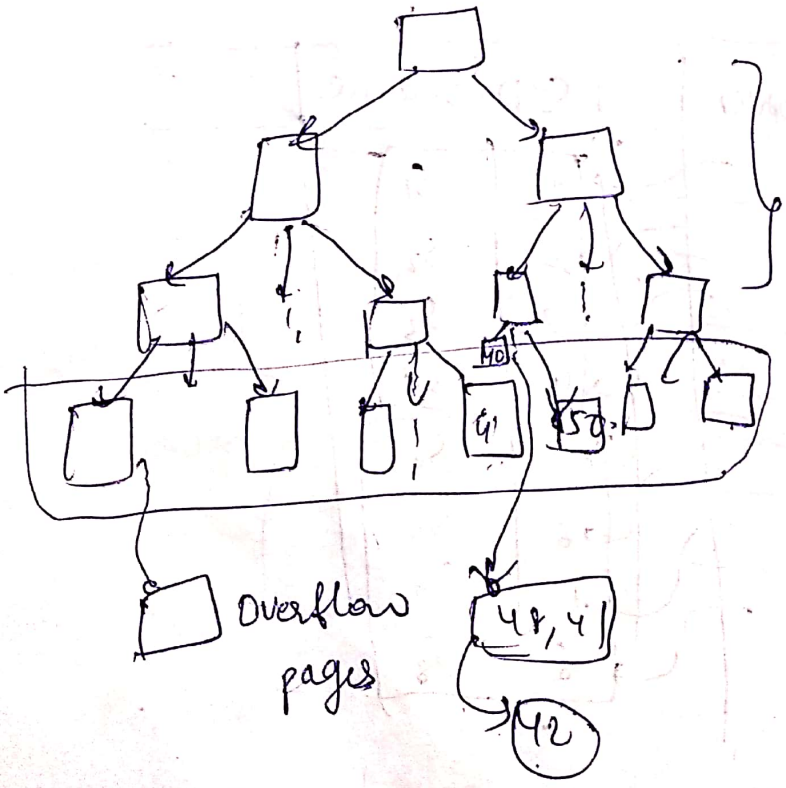
→ We need 25 blocks for 100 records







Index sequential Access model (ISAM)



B+tree :

→ It is a balanced tree

* Internal nodes direct the search.
* Leaf nodes contain the data entry.
* Leaf nodes organized into doubly linked list

Characteristics :

(1) Minimum occupancy is 50% each node, except root

(2) each node contains m entries where $d \leq m \leq 2d$ entries.

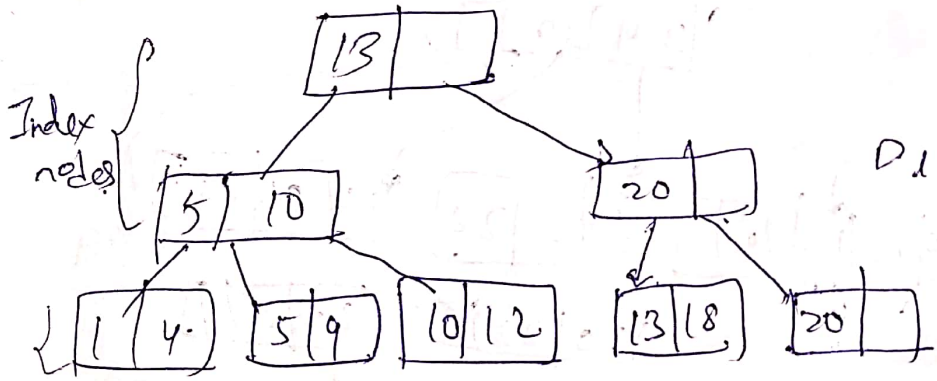
d - order of tree

Ex: order $d = 1$

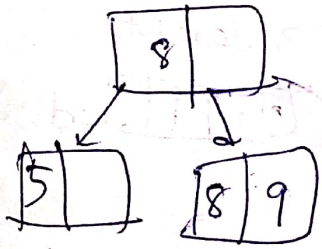
$k_1 < K < k_r$ — Index

$\lceil \frac{m}{2} \rceil$

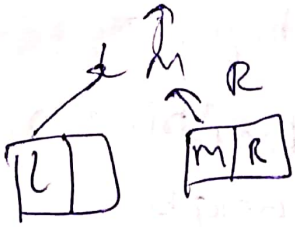
$P_1 \geq K \leq P_r$ - leaf.



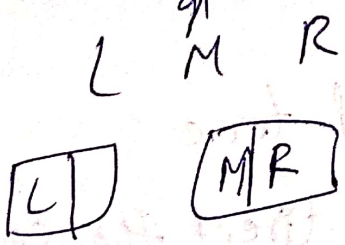
leaf (data field)



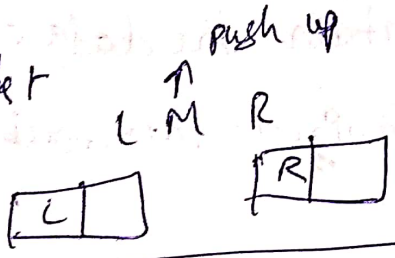
→ leaf.



Leaf node



Index node



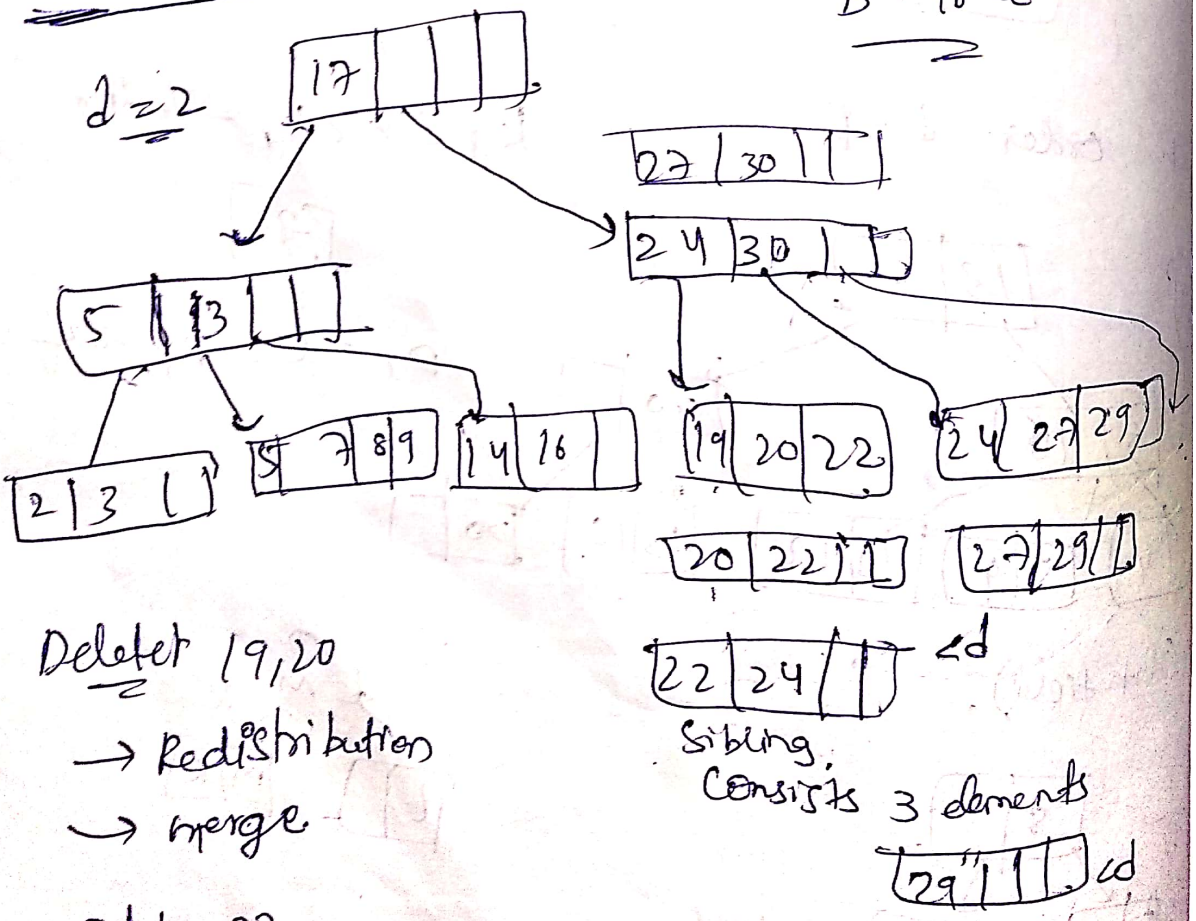
650
799
499

Lab
product

Error
02000 → NO Row found

Create trigger ensure Direction before insert on
tablename FOR EACH ROW .....

B+ tree



Delete 19, 20

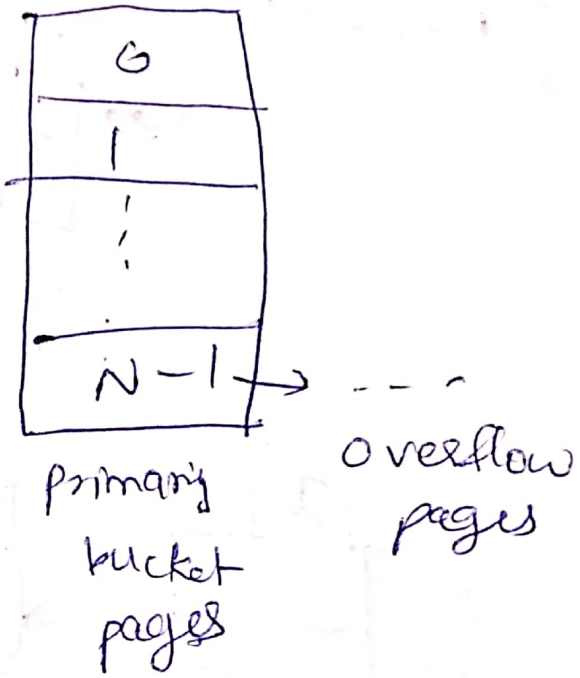
→ Redistribution

→ merge

Delete 27.

→ Static
→ Dynamic

Static Hashing :



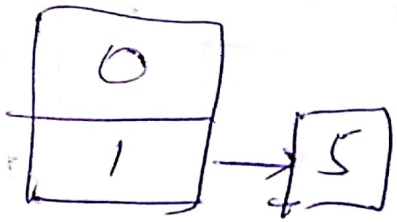
* 0, 1, .. N-1 buckets
* Buckets store data entries

Search: Apply hash, h

$h(r) = r \pmod N$

$h(3) = 3 \bmod 2 = 1$

$h(5) = 5 \bmod 2 = 1$



- Use of directory of pointers to buckets

→ Double the size of no. of buckets

just by doubling the directory and spl the bucket overflowed

Ex: $h(r) = r \bmod 2 = \{0, 1\}$

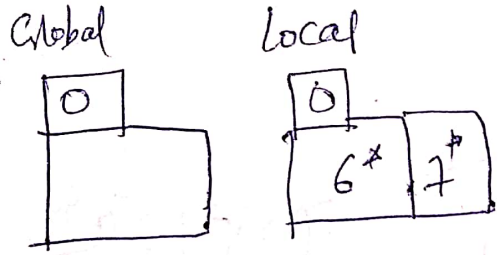
Q) Add 6, 7, 8, 10.

bucket size is 2

d - depth

{ Global
  local

Sol:



6 - 110

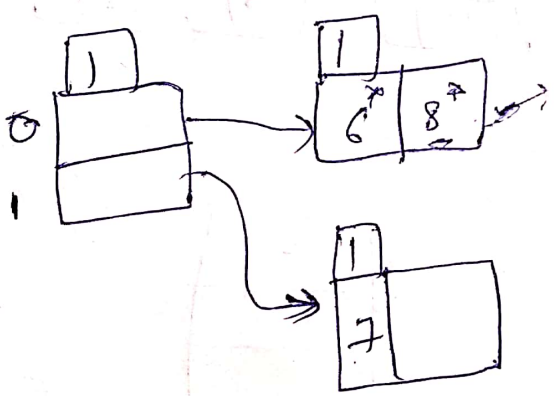
7 - 111

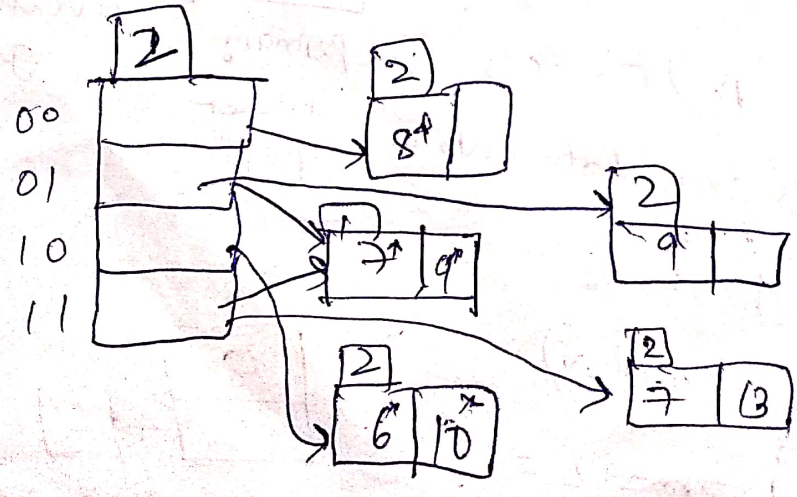
8 - 1000

10 - 1010

9 - 1001

13 - 1101





11/12/20 # Transaction Management

Consistency & Isolation

A $\xrightarrow[50]{\text{trans}}$ B

(-50) $\xrightarrow{40}$ (B) } Inconsistent
)((+40)

Illust: User program's logic

T_1 & T_2 concurrently.

T_1 ; T_2 } should be equivalent
T_2 ; T_1 }

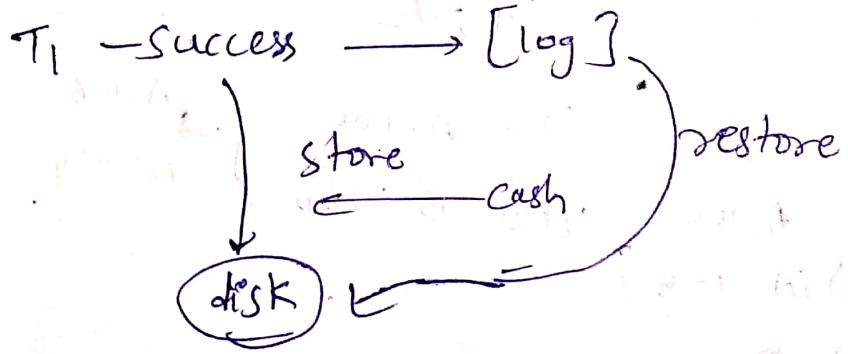
Isolation

Atomicity & Durability

Transaction incomplete
→ Aborted
→ System crash
→ Read unexpected data
∴ Unable to access disk

A $\xrightarrow{\text{trans}}$ B
↓
C

[log] - Used, remembers & restore.



Transactions & Schedules

Transaction - list of actions.

Actions : Read, write of database object.

O - object

$R_T(O)$ - action of T is read object O.

$W_T(O)$ - action of T is write object O.

Transaction must specify : commit, abort.

Schedule (read, write, abort, commit)

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

T_1 & T_2 conflict each other?

200 300

→ WR
→ RW
→ WW

T_1 : transfer Rs. 100 from A to B
T_2 : increment both A & B by 6%

T_1	T_2
A: 200 R(A)	
A: 100 W(A)	
	R(A) A: 100
	W(A) A: 106
	R(B) B: 300
	W(B) B: 318
	commit
B: 318 R(B)	
B: 418 W(B)	
commit	

T_1 ; T_2

100 A-100, → A: 106
400 B+100 B: 424

Unrepeatable reads → (RW conflict)

A – No. of copies available for a book.

Integrity constrain $A \geq 0$

T – places an order, first read A.

then check $A > 0$

then decrement 1

T_1 : R(A) and sees A = 1

T_2 : R(A) and sees A = 1

W(A), decrement by 1, A = 0

commit

T_1 : W(A) and decrement by 1, A = −1.

Overwriting Uncommitted Data (WW conflict)

A & B are two employees, their salaries must be kept equal

T_1 : sets A and B Salaries Rs. 1000

T_2 : sets A and B Salaries Rs. 2000

T_2 : W(A)

T_1 : W(B)

T_2 : W(B) – commit

T_1 : W(A) – commit

A	B
2000	
	1000.
	2000 ←
1000 ←	

Schedule Involving Aborted Transactions

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

T_1 : deduct Rs 100 from A
T_2 : read A & B
add 6%
Commit
T_1 : Aborted

Unrecoverable schedule

Strict Two-phase locking (Strict 2PL)
↳ By using this we can avoid unrecov — schedule

Strict 2PL:

action of T is read on dj
o

lock
- Shared lock $S_T(o)$ – read
- exclusive lock $X_T(o)$ – write, read

Ex:2

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

T_1	T_2
X(A)	
R(A)	
W(A)	
	Request X(A), not allowed T_2 support
X(B)	
R(B)	
W(B)	
commit	
	R(A)
	W(A)
	X(B)
	R(B)

Once committed, the lock can be transferred to another transaction

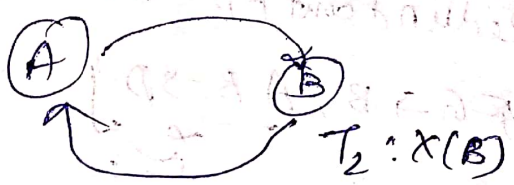
Ex:-

T_1	T_2
S(A)	
R(A)	
	S(A)
	R(A)
	X(B)
	R(B)
	W(B)
	commit
X(C)	
R(C)	
W(C)	
commit	

Serial Interleaving

Deadlock:-

T_1 : X(A)



T_2 : X(B)

T_1 waits until T_2 completion and T_2 waits until T_1 complet

This is a deadlock.

T_1	T_2
X(A)	
R(A)	
W(B)	
	X(B)
	R(B)
	W(B)
	X(A) ← request held in queue
request → X(B)	
held queue	

→ If deadlock happens, we have to terminate it

→ We have to see that deadlock doesn't happen

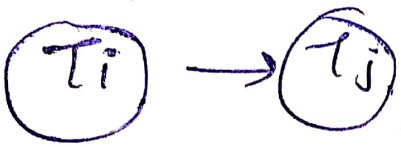
→ We take preventive mechanisms.

20/11/2020

T_1	T_2
X(A)	
R(A)	
W(B)	
	X(B)
	R(B)
	W(B)
request X(B) hold queue	X(A) ← request hold queue

Wait-for-graph :-

The lock manager maintain a structure called a wait-for-graph to detect deadlock cycles.



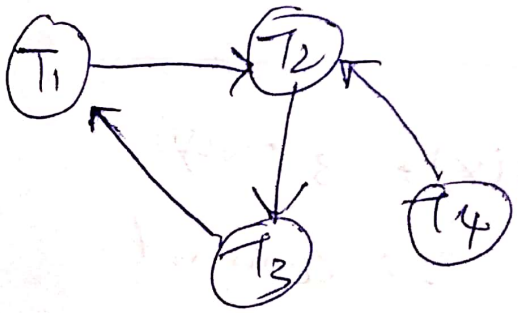
T_i is waiting for T_j to release a lock

(i) Lock manager adds edge to the graph when it queues lock request

(ii) and removes the edge when it grants lock request

Ex:

T_1	T_2	T_3	T_4
S(A)			
R(A)			
	X(B)		
	W(B)		
S(B)			
		S(C)	
		R(C)	
	X(C)		
			X(B)
	~~X(A)~~	X(A)	

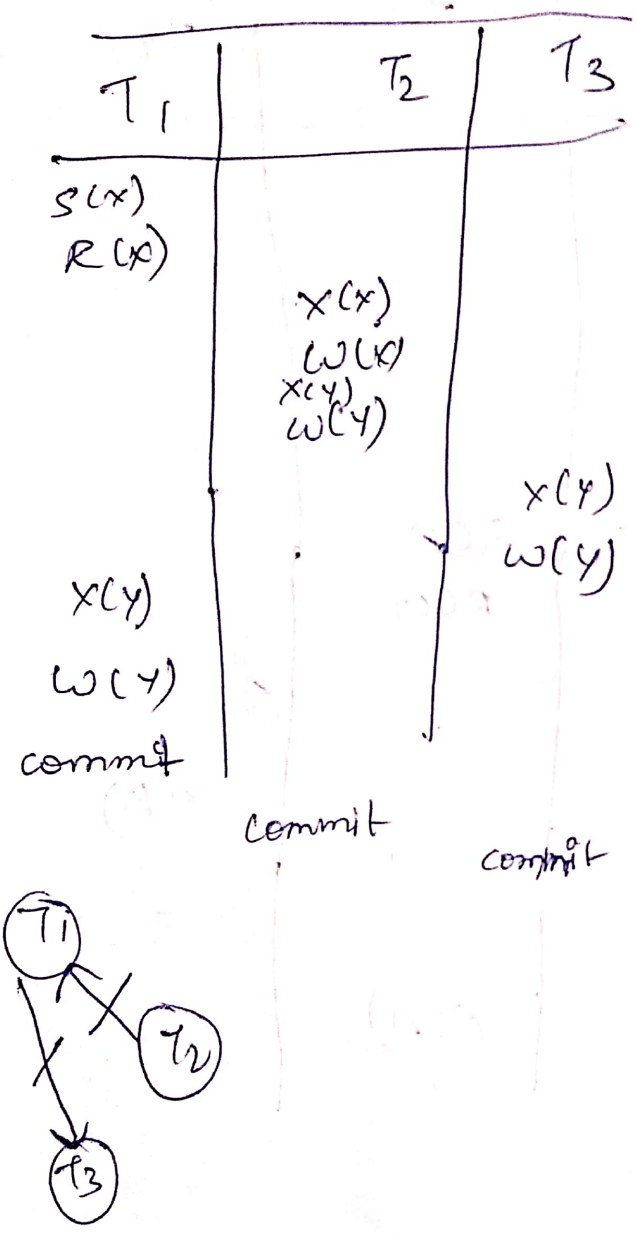


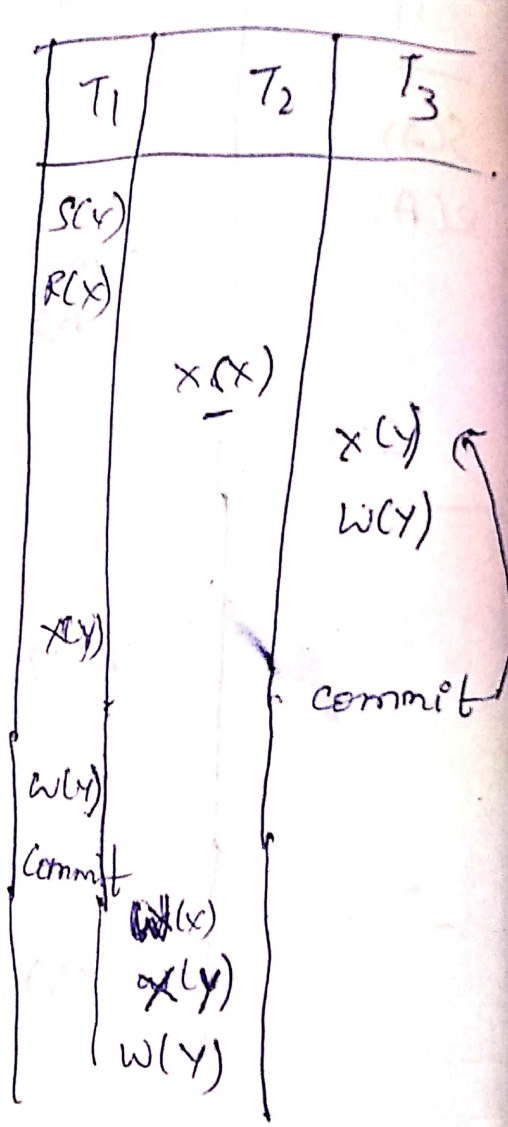
8) S : T_1 : R(x), T_2 : w(x), T_2 : w(y), T_3 : w(y)

T_1 : w(y)

T_1 : commit , T_2 : Commit T_3 : commit

Strict 2PL and wait-for graph.





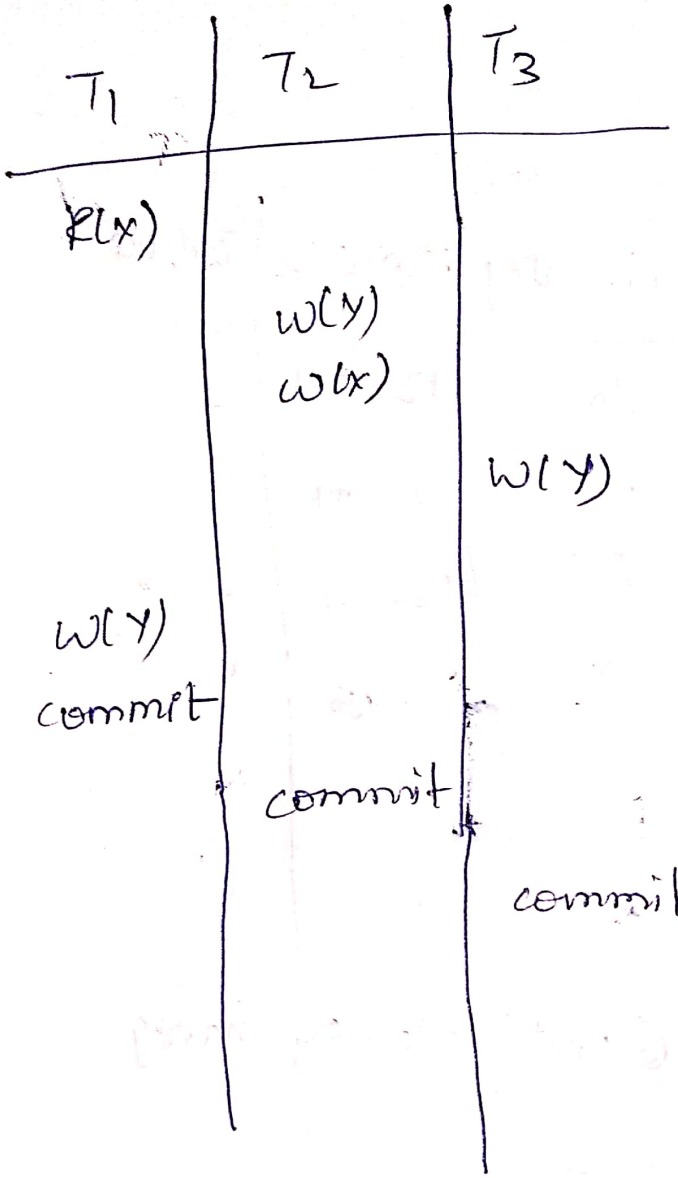
---

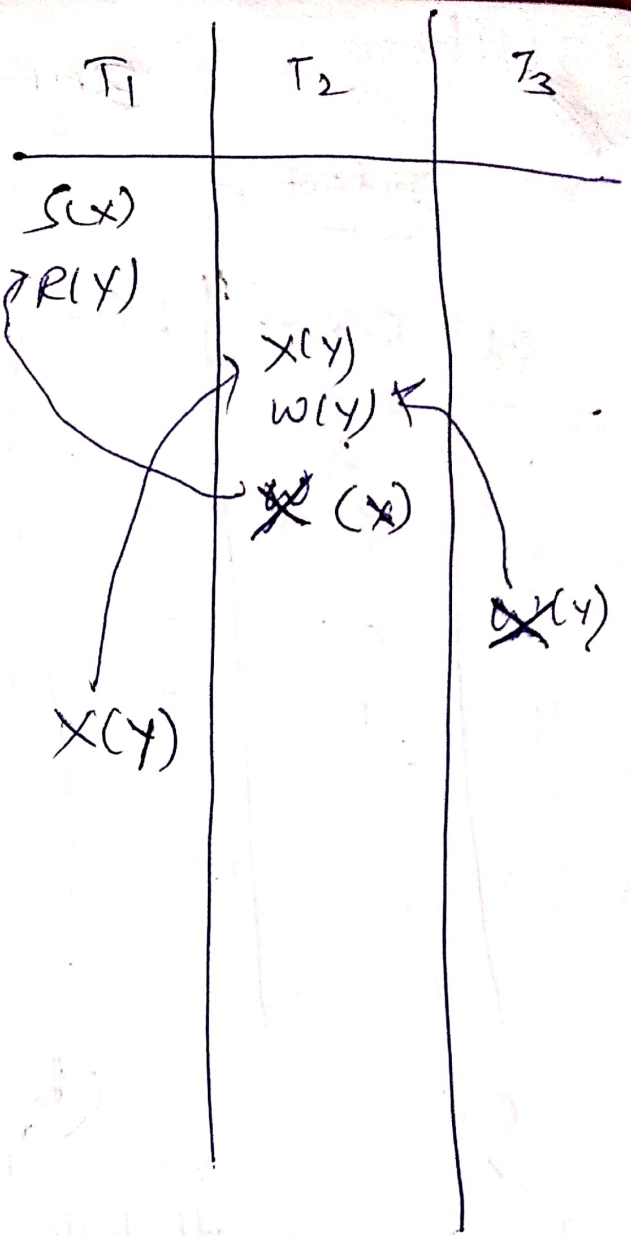
9)

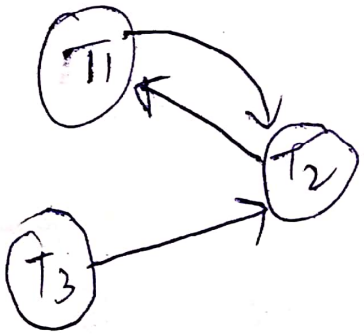
S: T_1 : R(x), T_2 : W(y), T_2 : w(x), T_3 : w(y),

T_1 : w(y), T_1 : commit, T_2 : commit.

T_3 : commit


T1
T2
T3
R(x)
W(y)
W(x)
W(y)
W(y)
commit
commit
commit


T1
T2
T3
S(x)
R(y)
X(y)
W(y)
X(x)
X(y)
X(y)


T1
T2
T3