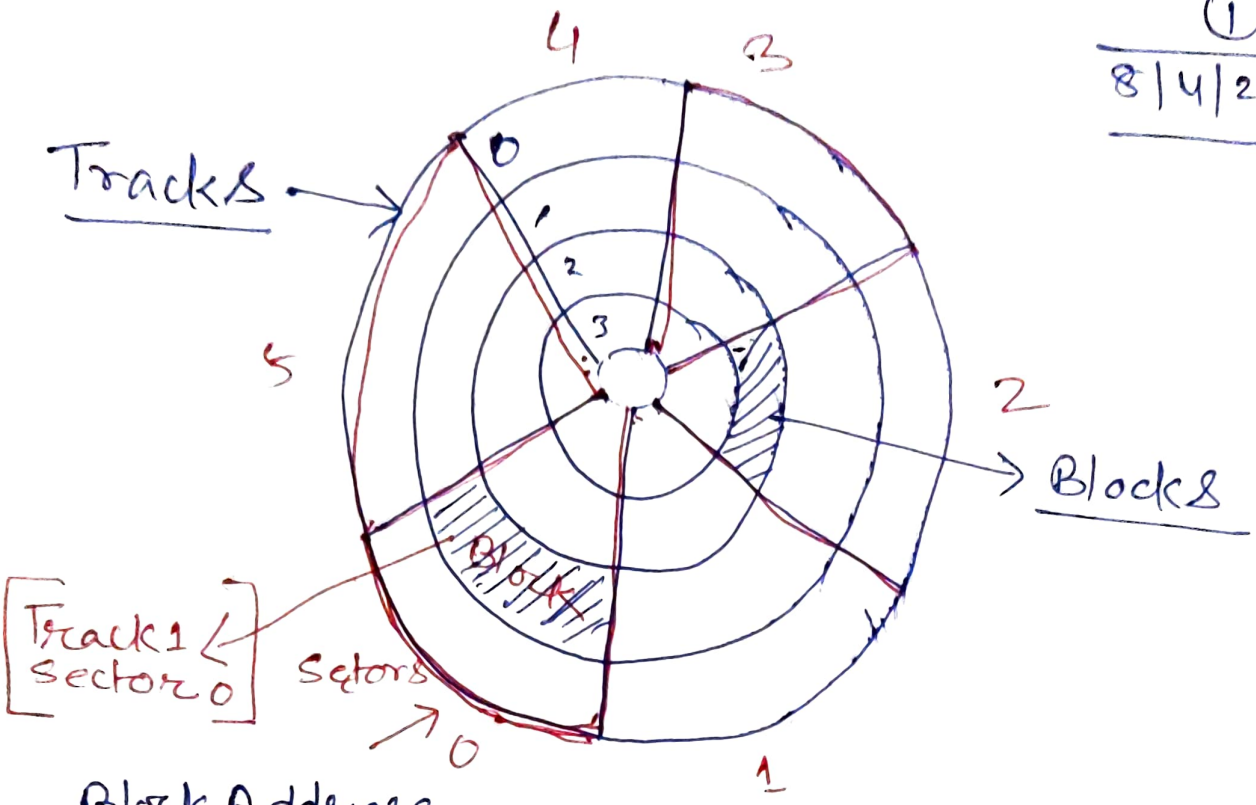
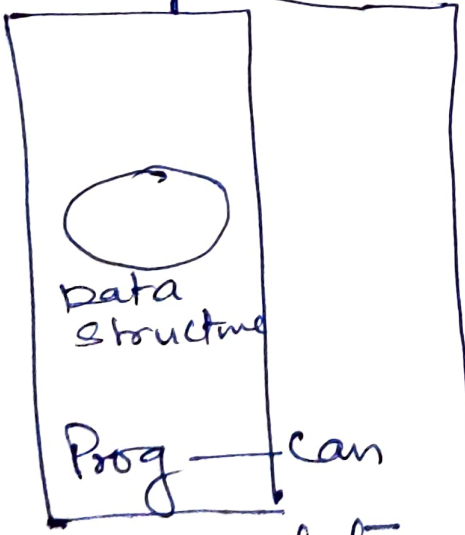


Block Address = (Track no., sector no.)

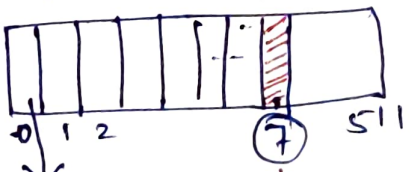
Suppose we take block size as 512 bytes.

(RAM) main memory



access data from disk through RAM.

Block



Offset

① Track 1
② sector 0
③ offset 7 →

Processing of data can't be done directly on disk. data will be accessed in RAM and processed (ops) and sent back to disk.

8/4/21

Organizing data in RAM through program is done through data structure and in disk through dbms

Emp

empid – 10 bytes
name – 50 "
dept – 10 "
section – 8 "
address – 50 "

Size = 128 bytes

	empid	name	dept
128	1.	RAM	ESE
	2.	SHYAM	CSE
	3.	AMAN	MAE
	4.	MANAV	ECE
	5.	ANU	MAE
	:	:	:
	100	:	:

Block size = 512 bytes

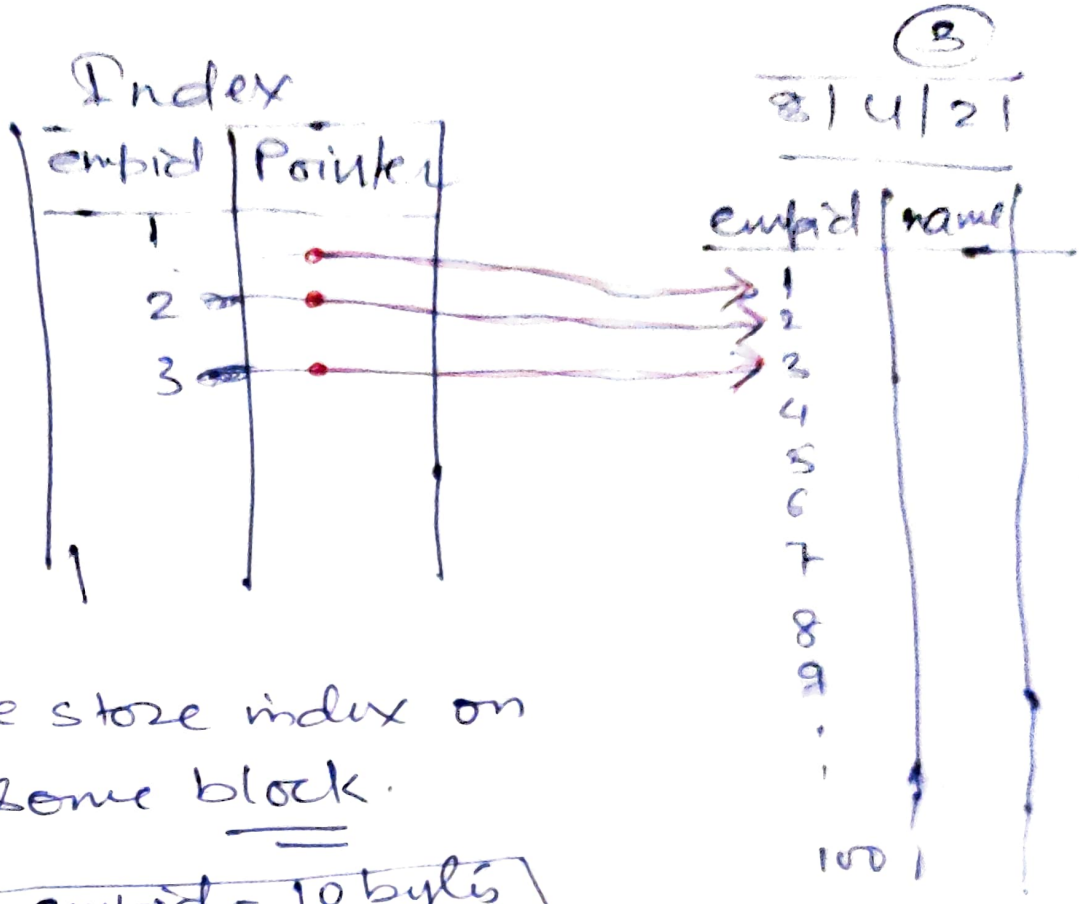
1 record size = 128 bytes.

No. of records per block = $\frac{512}{128}$

= (4) records

So for 100 records we need $\frac{100}{4}$ = 25 blocks in disk to store this table.

If query to search a record e.g.
select emp with empid = 5
this query needs to access 25 blocks as it contains 100 records.



we store index on some block.

empid - 10 bytes
Pointer - 6 bytes

Total = 16 bytes.

In one Block = $\frac{512}{16} = 32$ (Pointers or index only)

$\frac{\text{records}}{32} = \frac{100}{32} = 3.3$ (approx 4)

So roughly we need 4 blocks to store index for 100 records

this to access a record we need to access index block (4) and then database block (only one)

total = 5 block only

Benefit of indexing.

increasing performance

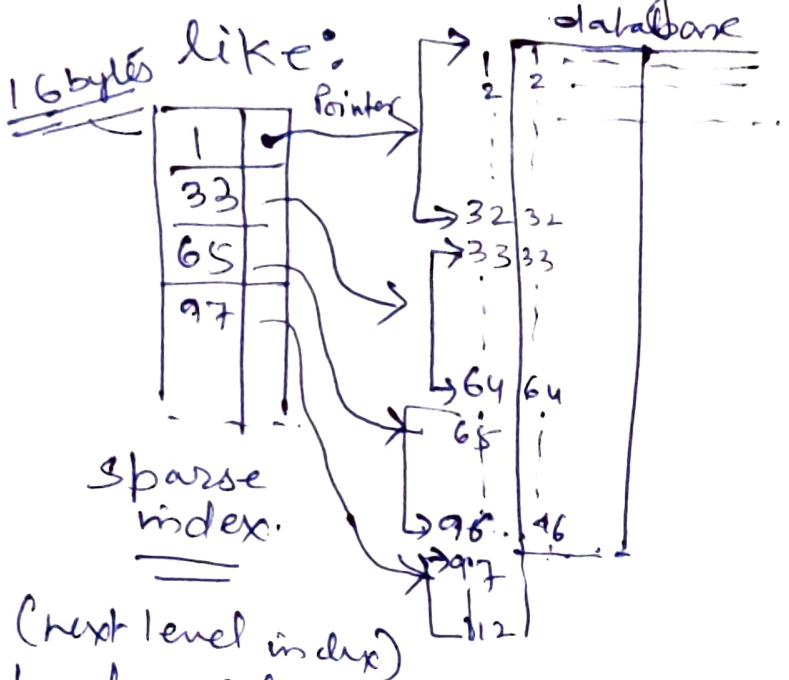
Multilevel Indexing

9/4/21

No. of Record grows from 100 to 1000.

Now we need 250 blocks to store records and 40 blocks to store indexes.

In one block we can have 32 pointers or indexes so we create higher index like:



higher index.	index	Records
1.2 ≈ 2 blocks	40 blocks	1000

32 | 1000 | 31.2

≈ 32

32 × 16 = 512 ⇒ 1 block for next-higher index.

9/4/21

So Now using next-level index we one block higher index Fizly block of index one block of database

---

if indexing is not there then we need to traverse 250 blocks to search a record.

---

if higher level index is not there then 32+1 = 33 blocks to search a record

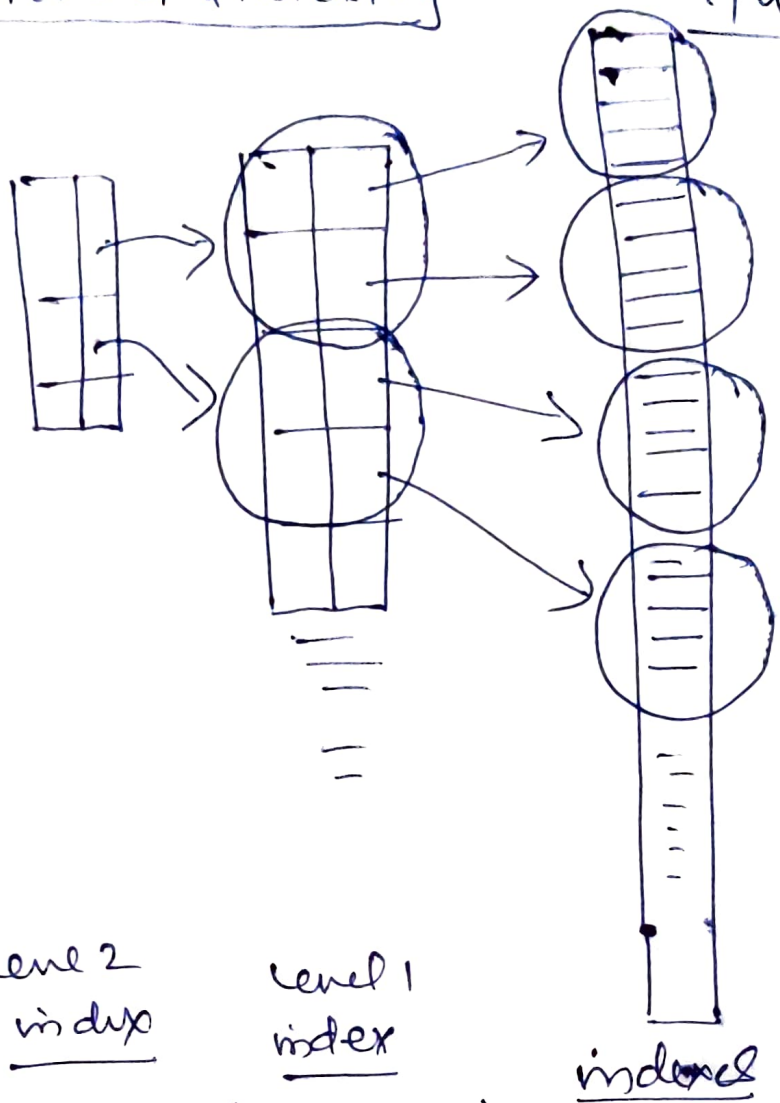
---

But next level higher index will reduce no. of blocks to search for a particular record to 1+1+1 = 3 blocks only

---

Multilevel Indexing

9/4/21

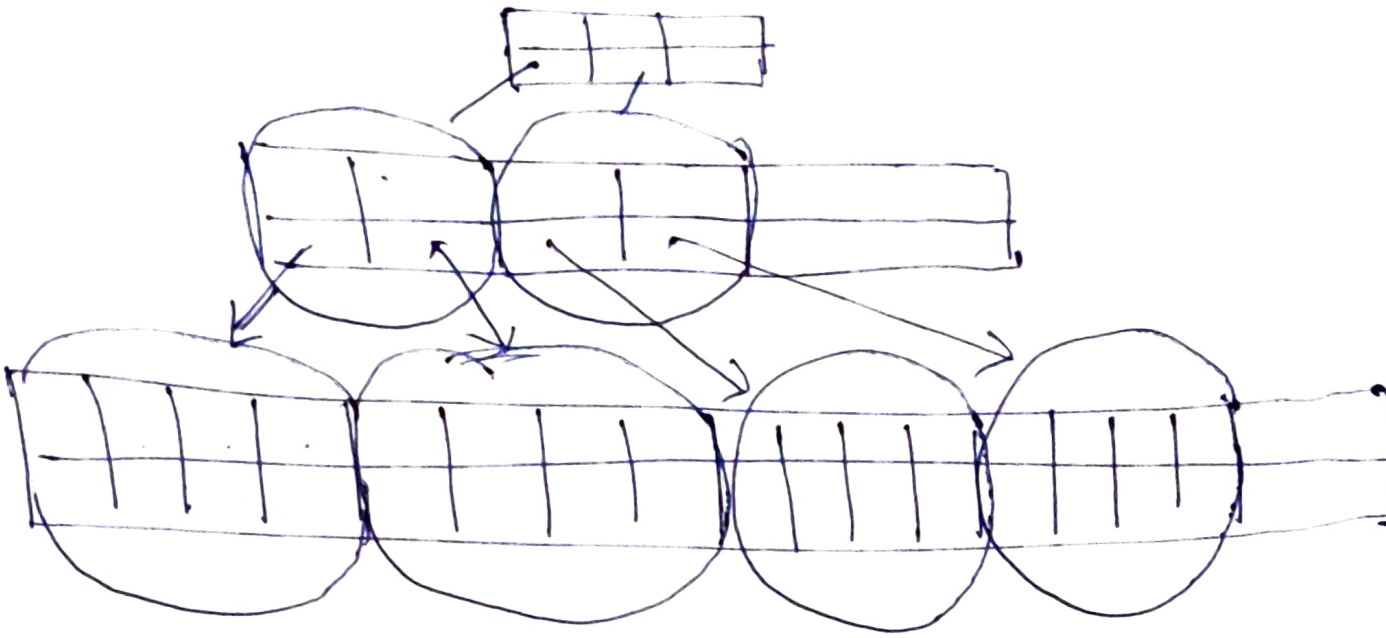


As no. of records growing, size of index also grows so we need to increase the level of indexing to reduce the no. of blocks to access to search a record.

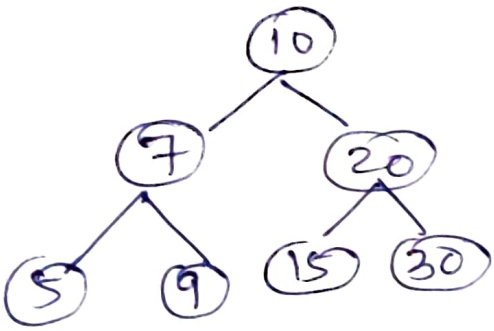
we need self managed indexes so that as no. of records increases in database indexing is created and of records deleted, indexing also gets reduced.

9/4/21

Turn the previous diagram clockwise we get-

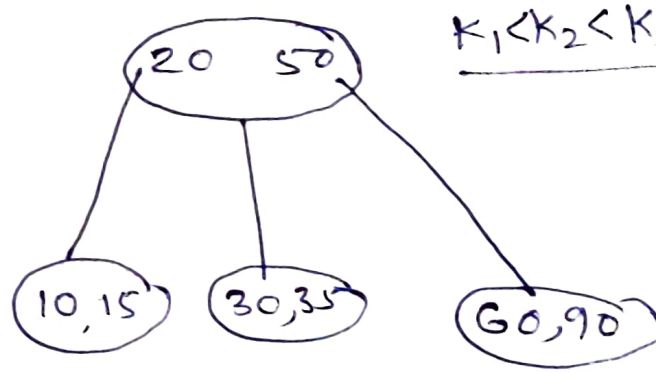


Self Managed Multilevel Indexing



1 key
2 child

$K_1 < K_2 < K_3$



2 key
3 children
3 way-search tree

M-way search Tree

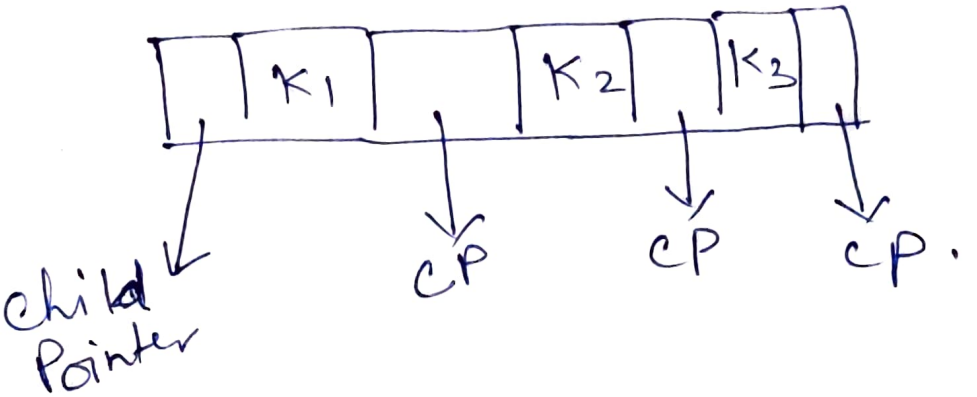
each node can have at most m children, (m-1) keys.

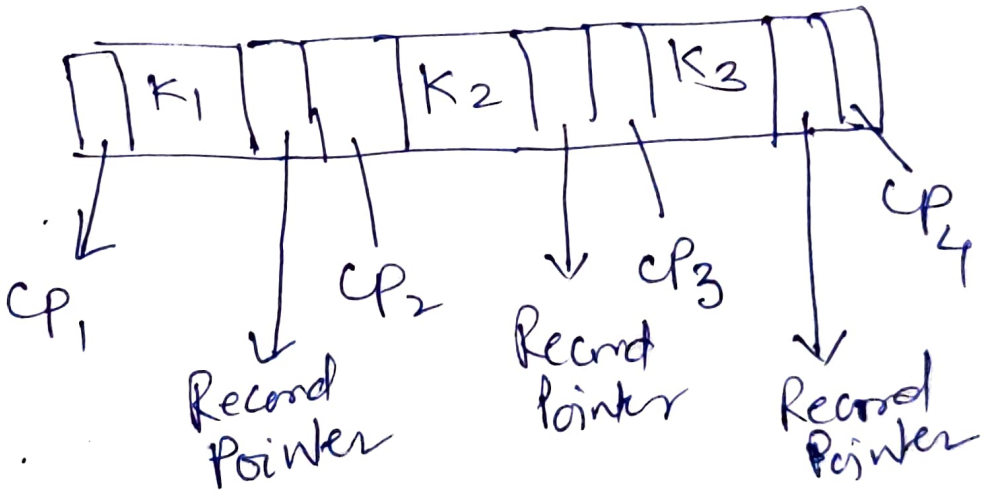
M-way Search Tree

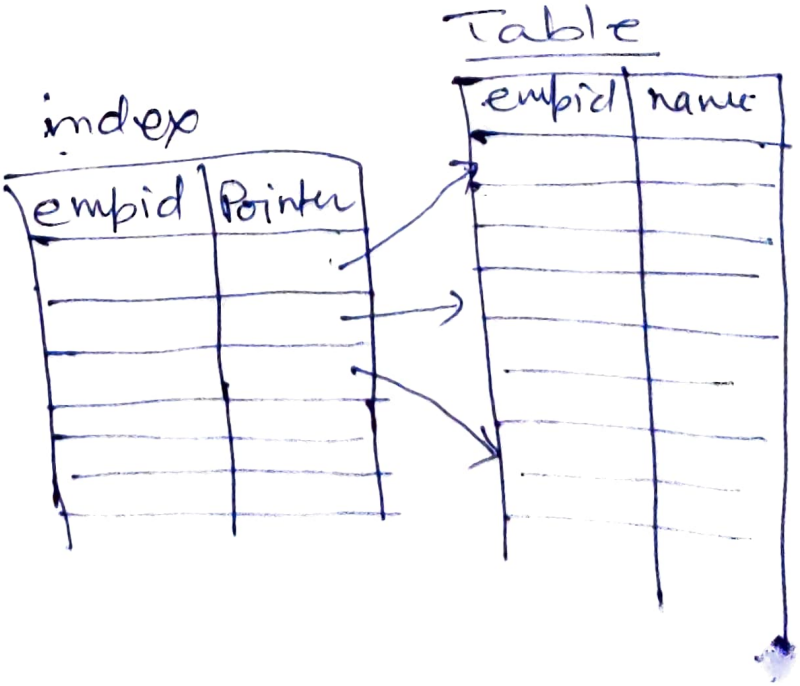
9/4/21

m-way ST

4-way





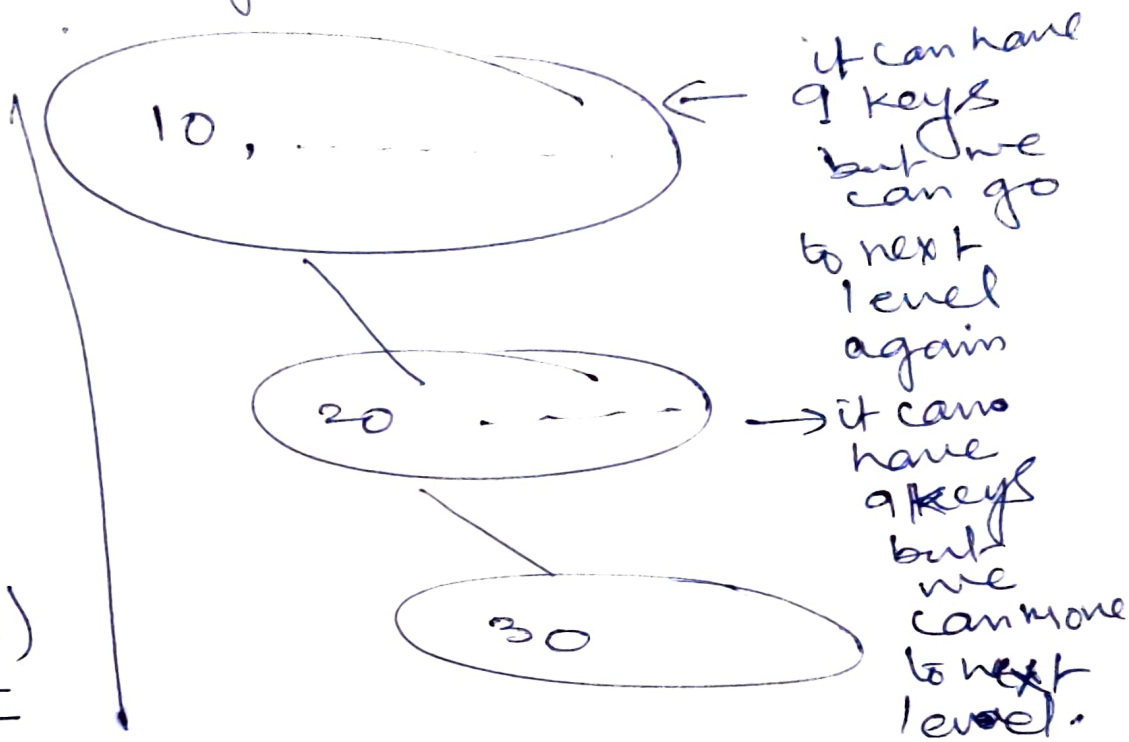


M-way Search Tree

9/4/21

10-way search tree

keys – 10, 20, 30 ..... n



To search a key it can take O(n) time because no strict rule in m-way searchtree. So B-Tree came with some specific rules.

B-Tree

Rule 1. Every node we must fill at least half:

i.e. $\lceil m/2 \rceil$ ceil function.

children must be there.

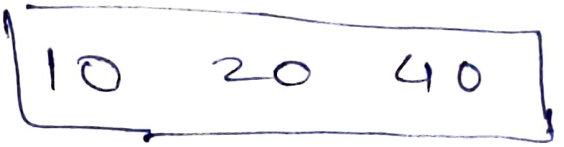
2. Root can have min 2 child
3. All leaf at same level.
4. Creation process is bottom up

B-Tree

m = 4 → keys = 3 only

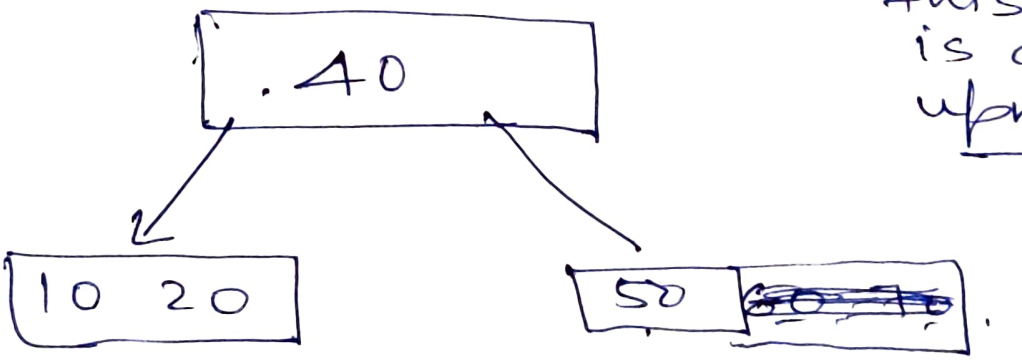
9/4/21

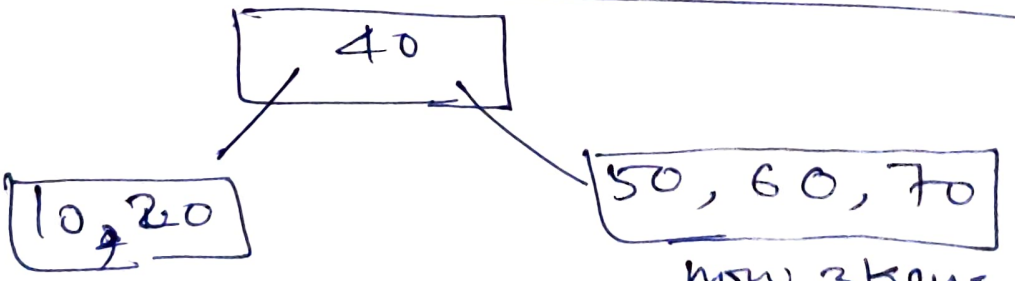
keys: 10, 20, 40, 50, 60, 70, 80



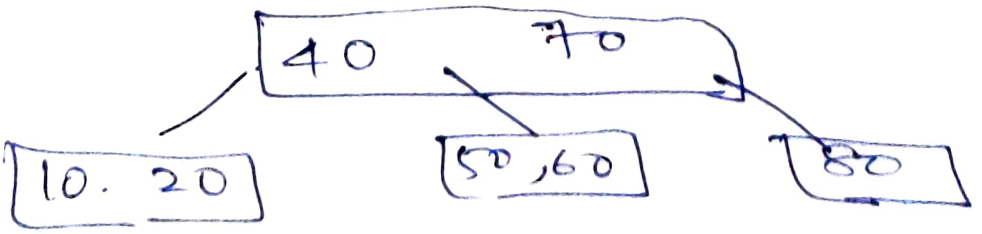
Now we can't insert 50 because only 3 keys it can have. So now split the node.

this tree is growing upward.





now 3 keys so split for 80

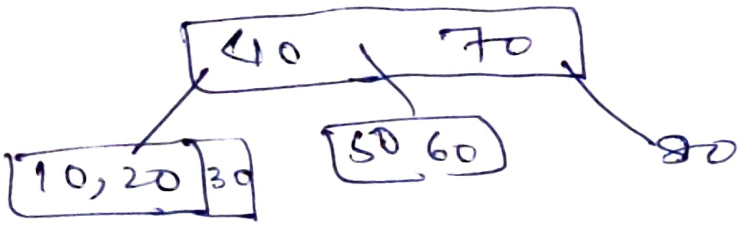


B-Tree

9|4|21

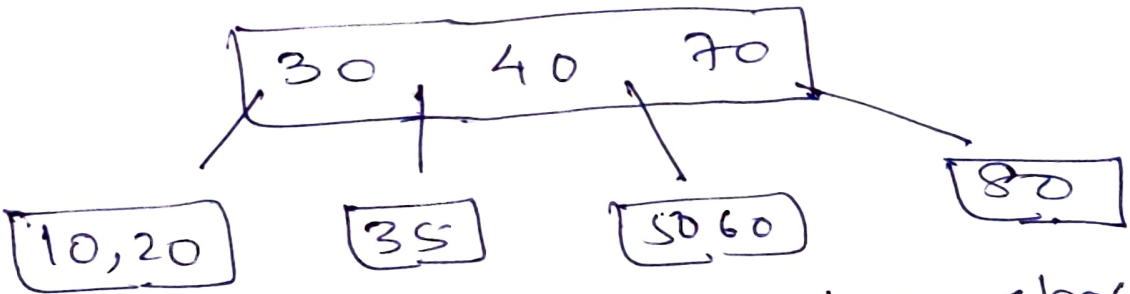
m=4 → keys = 3 only

keys : 10, 20, 40, 50, 60, 70, 80, 30



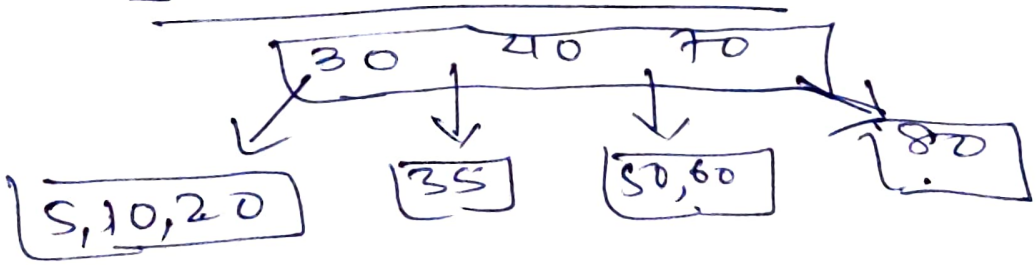
Now insert 35

there is no space in 10,20,30
so split, take 30 up.



whenever there is no space in child node then split child node and send last key of child node to parent node.

Now insert 5

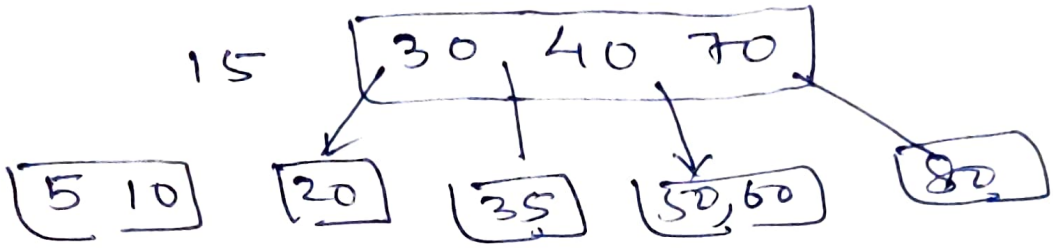


B-Tree

9/4/21

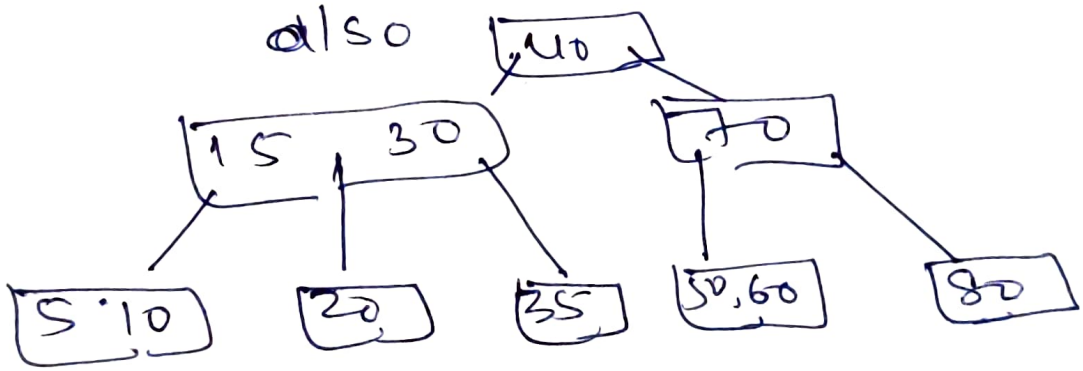
Insert 15

there is no space in [5,10,20] so split and send 15 to parent node.



[5,10,15,20]

Now above also no space for 15 so we split the parent node also



15 30 40 70