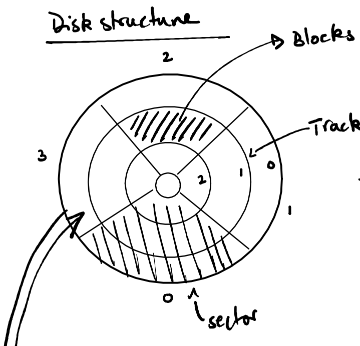
B-Trees & B+-Trees

Monday 17 June 2024 6:10 PM

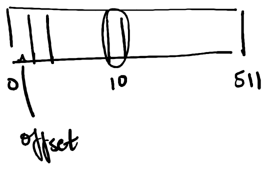
Contents

Disk structure



★ organising the data on the disk officiently so that it can be easily utilized is called **DBMS**

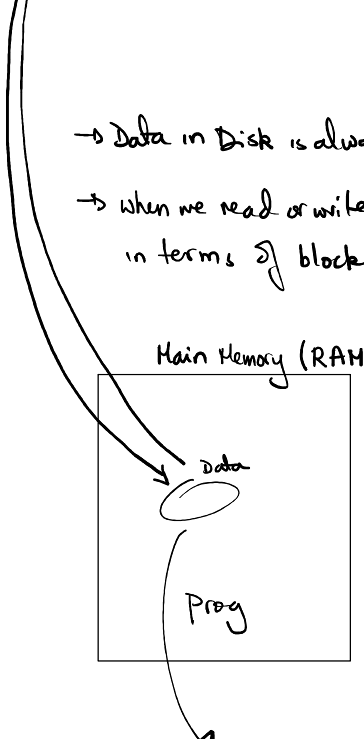
Block Address: (Track#, section#)



→ Data in Disk is always stored in terms of blocks

→ when we read or write from the disk, we always read & write in terms of blocks.

Main Memory (RAM)



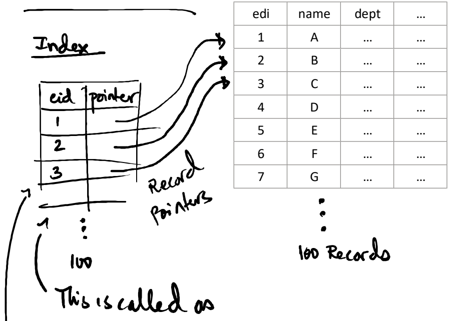
→ Data cannot be directly processed upon the disk, it has to be brought into the main memory & be accessed.

★ organising the data in the main memory that is directly used by the program that is data structures.

How is data organized on the disk in the form of database

How to reduce time?

Index



eid	pointer
1	
2	
3	
⋮	
100	

edi	name	dept	...
1	A	...	...
2	B	...	...
3	C	...	...
4	D	...	...
5	E	...	...
6	F	...	...
7	G	...	...
⋮			

100 Records

This is called as dense index

Employee

eid	10
name	50
dept	10
section	8
add	50

size = 128 Bytes

$\# \text{ of records / Block} = \frac{512}{128} = 4$

$\# \text{ of Blocks} : \frac{100}{4} = 25 \text{ Blocks}$

Time depends on # of Blocks we are accessing

Index is also stored inside the disk

eid – 10

pointer – 6

16 Bytes

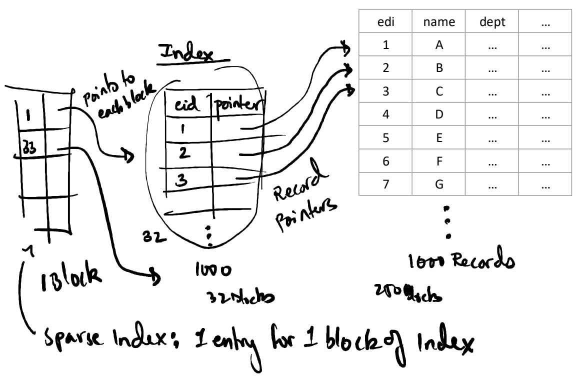
$\# \text{ entries / Block} = \frac{512}{16} = 32$

$\# \text{ of blocks} = \frac{100}{32} = 3.2$

4Blocks   Now to acces any record
we only need to acces 4+1 blocks

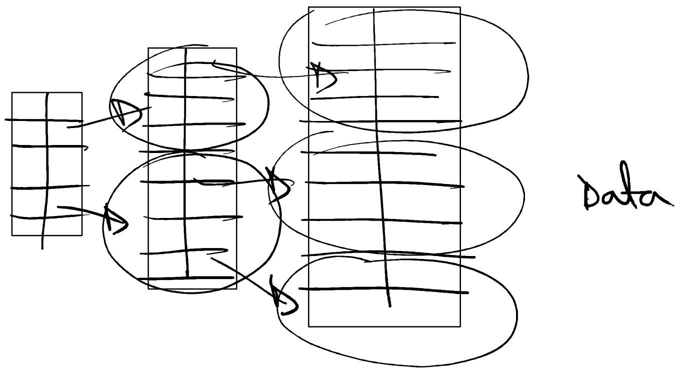
Multilevel Indexing

Suppose we now have 1000 records - 250 Blocks
we will has 1000 entrys in Index also - 32 Blocks

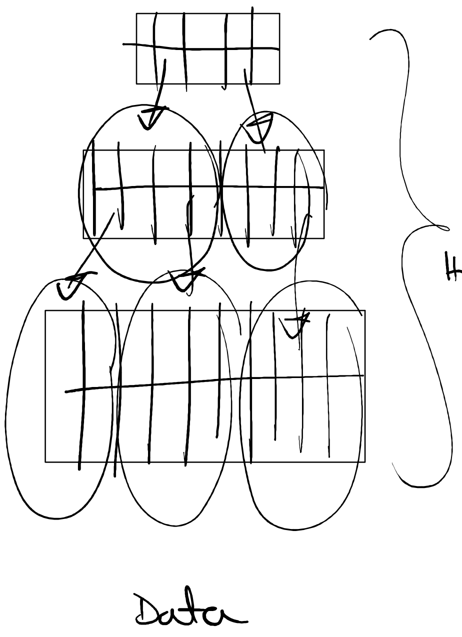


edi	name	dept	...
1	A	...	...
2	B	...	...
3	C	...	...
4	D	...	...
5	E	...	...
6	F	...	...
7	G	...	...

Sparse Index: 1 entry for 1 block of Index



Adding multilevel index will reduce the # of block acces



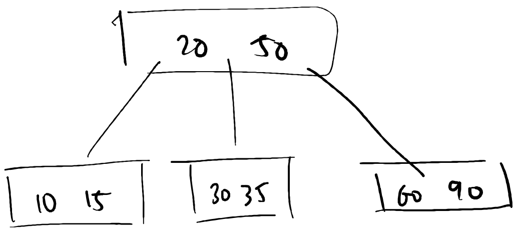
High level indices

→ we want the high level indices to added/deleted as Data grows & shrinks

→ self managed high-level indices/multi-level indexing

M-way search Tree

$k_1 < k_2 < k_3 \ldots$

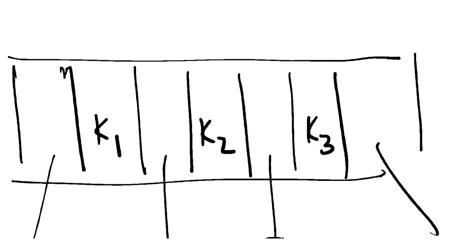


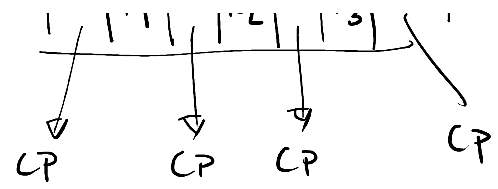
2 keys
3 children (at most 3 children)
3-way search Tree

M-way [Based on the degree of a node]
M-1 keys

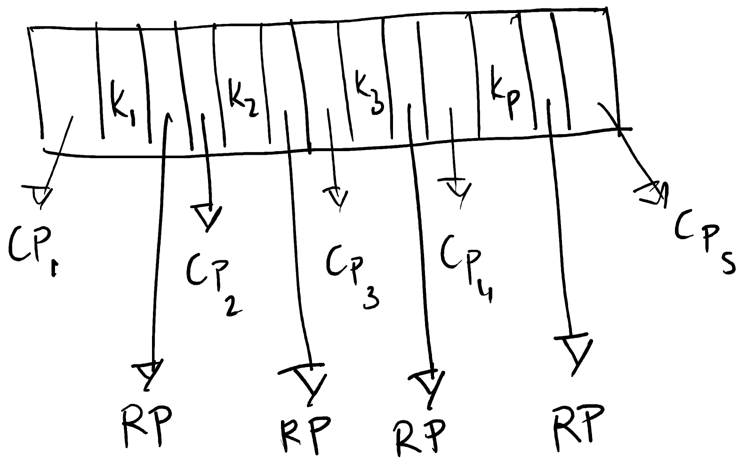
→ BST is a type of M-way ST with degree 2

e.g. 4-way ST



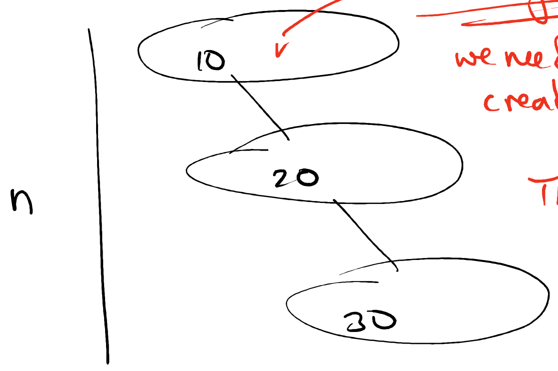


e.g. Using search trees to prepare Index



→ e.g. 10-way search Tree

Insert : 10, 20, 30



8 more key possible

we need to fill this node before creating nexte node

There is no Control on M-way search tree

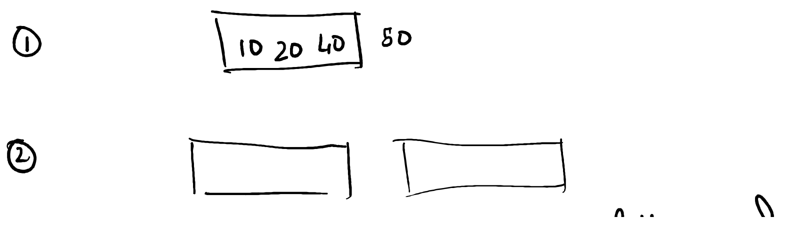
Problem: Creation process is not under any control

B-Trees (M-way search Trees with guidlines) (useful for implementing multi-level indexing)

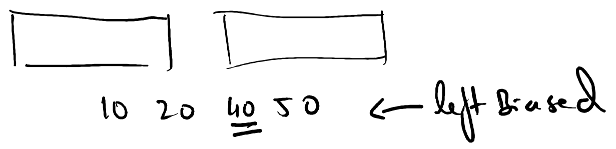
1. Every node must be have $\lceil m/2 \rceil$ children then only think of creating new node
2. Root can have minimum 2 children
3. All leaf nodes at same level
4. Creation process is Bottom up

e.g. m = 4

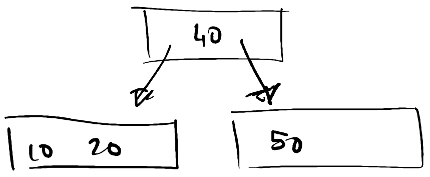
keys: 10, 20, 40, 50, 60, 70, 80, 30, 35, 5, 15



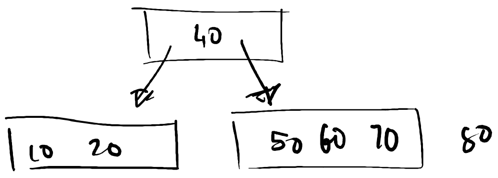
②


10 20 40 50
← left Biased

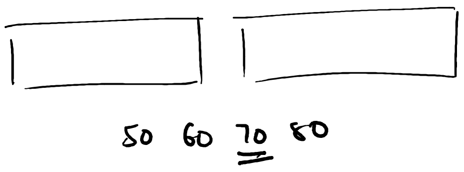
③


40
10 20
50

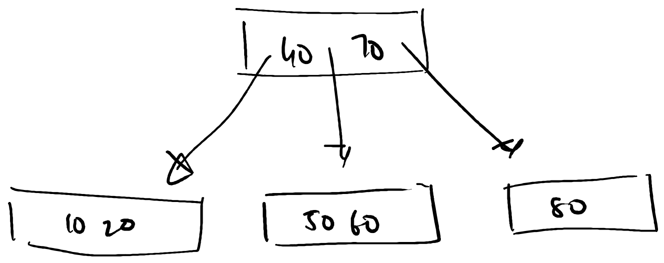
④


40
10 20
50 60 70
80

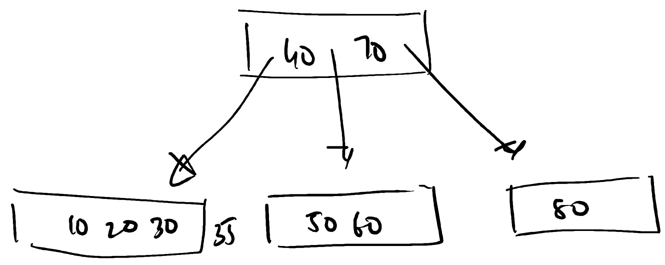
⑤


50 60 70 80

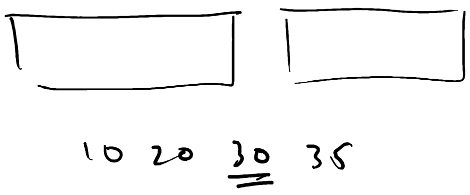
⑥


40 70
10 20
50 60
80

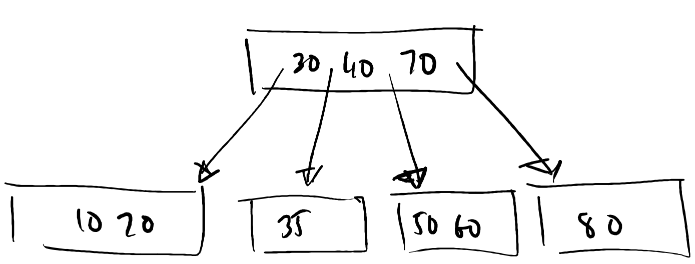
⑦


40 70
10 20 30
35
50 60
80

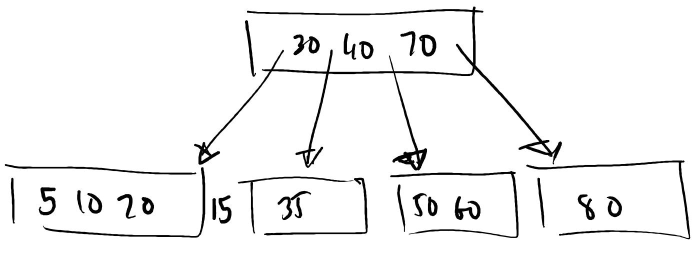
⑧


10 20 30 35

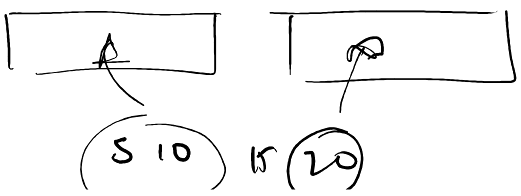
⑨

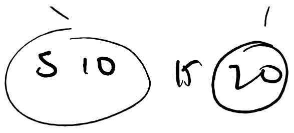

30 40 70
10 20
35
50 60
80

⑩

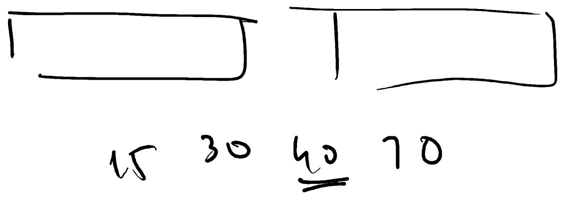

30 40 70
5 10 20
15
35
50 60
80

⑪

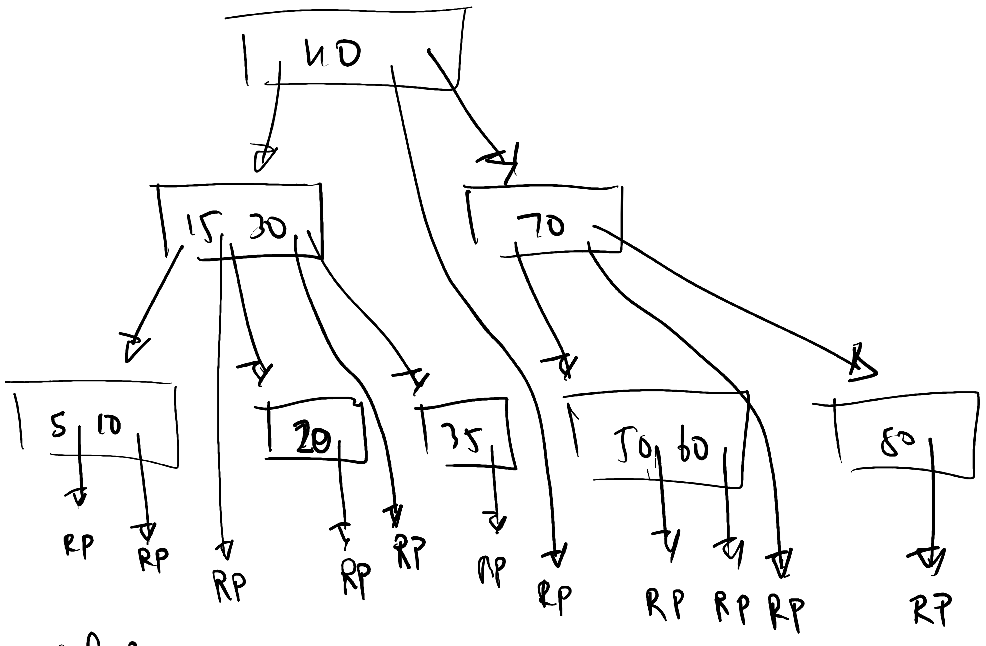

5 10
15
20



(12)



(13)

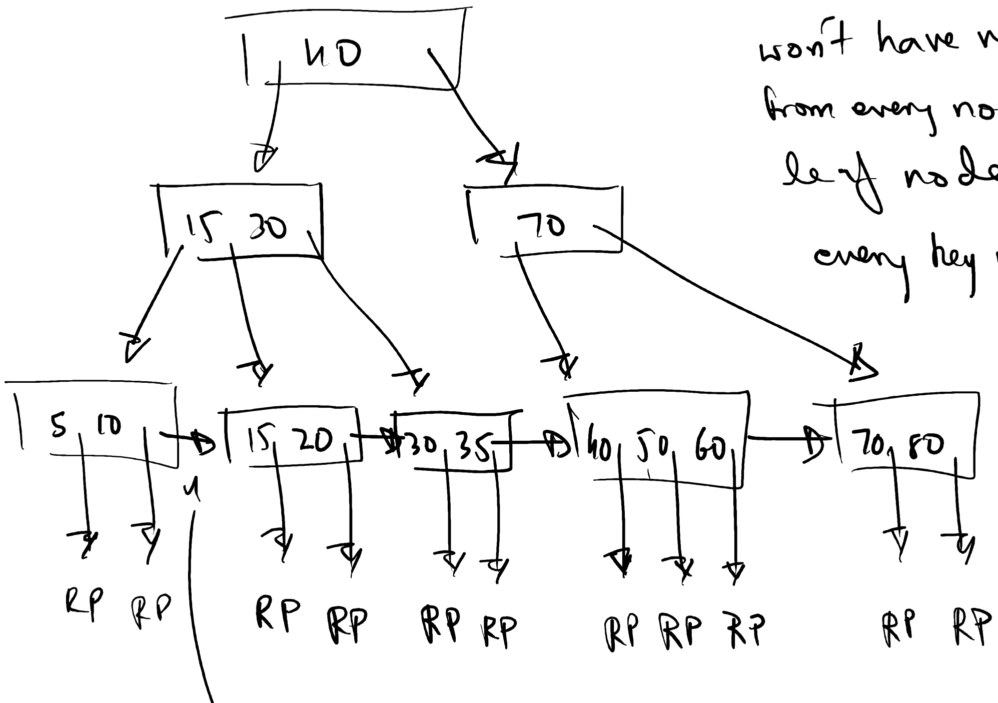


every Node:

→ Child pointer as a block pointer

→ Record pointer

B+ Trees



won't have record pointers from every node only from leaf node.

every key will have it's copy in the leaf node

} Dense index

leaf nodes are connected like a linked list

B+Tree is more like multi-level indexing