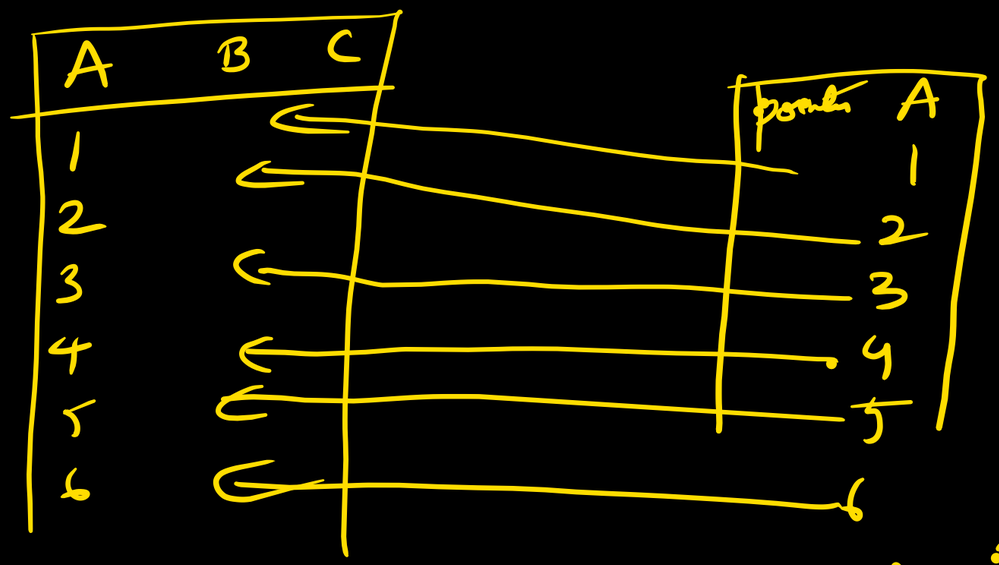
Search Key: fields used for indexing



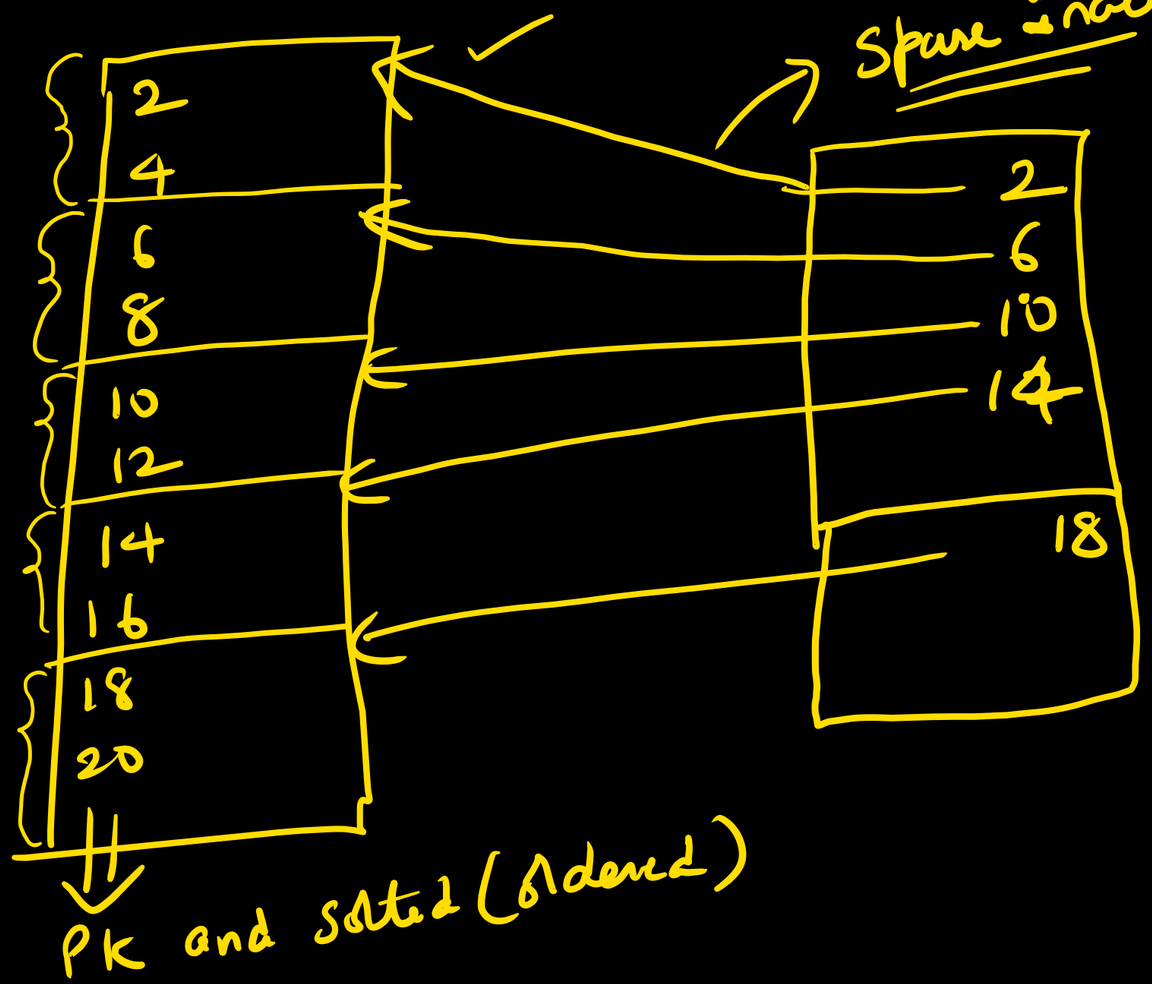
In this case A is search key.

Search key can be the key of DBT or non key of DBT

- Search key
  - Ordered field
    - Key
      - PK/AK
      - ① Primary index
    - Non key
      - ② clustering index
  - unordered field
    - key (PK/AK)
      - ③ Secondary index
    - non key
      - ④ Secondary index over non key.

Primary key: (Ordered field && Key)
↓ ↓
Sorted non repetition

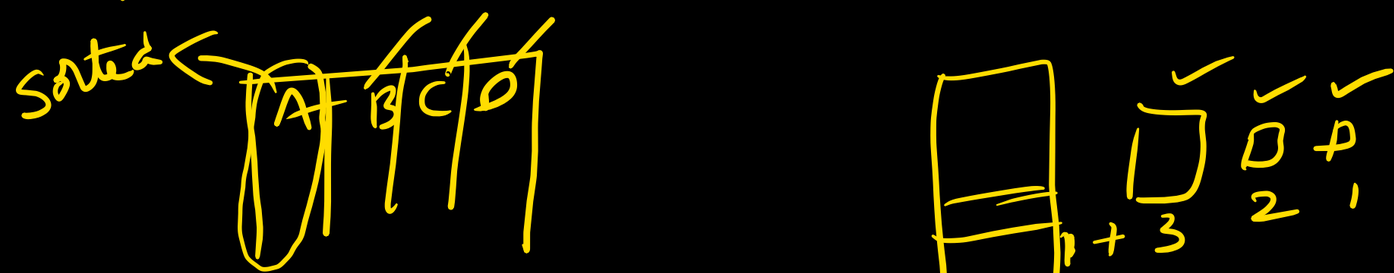
Sparse index → For many data records we have 1 index record



Important points on Primary index:

→ Primary index can be sparse of dense index

→ I/O cost PI with multi level indexing = $\underbrace{k+1}_{\text{blocks}}$ $k \Rightarrow$ no of levels.

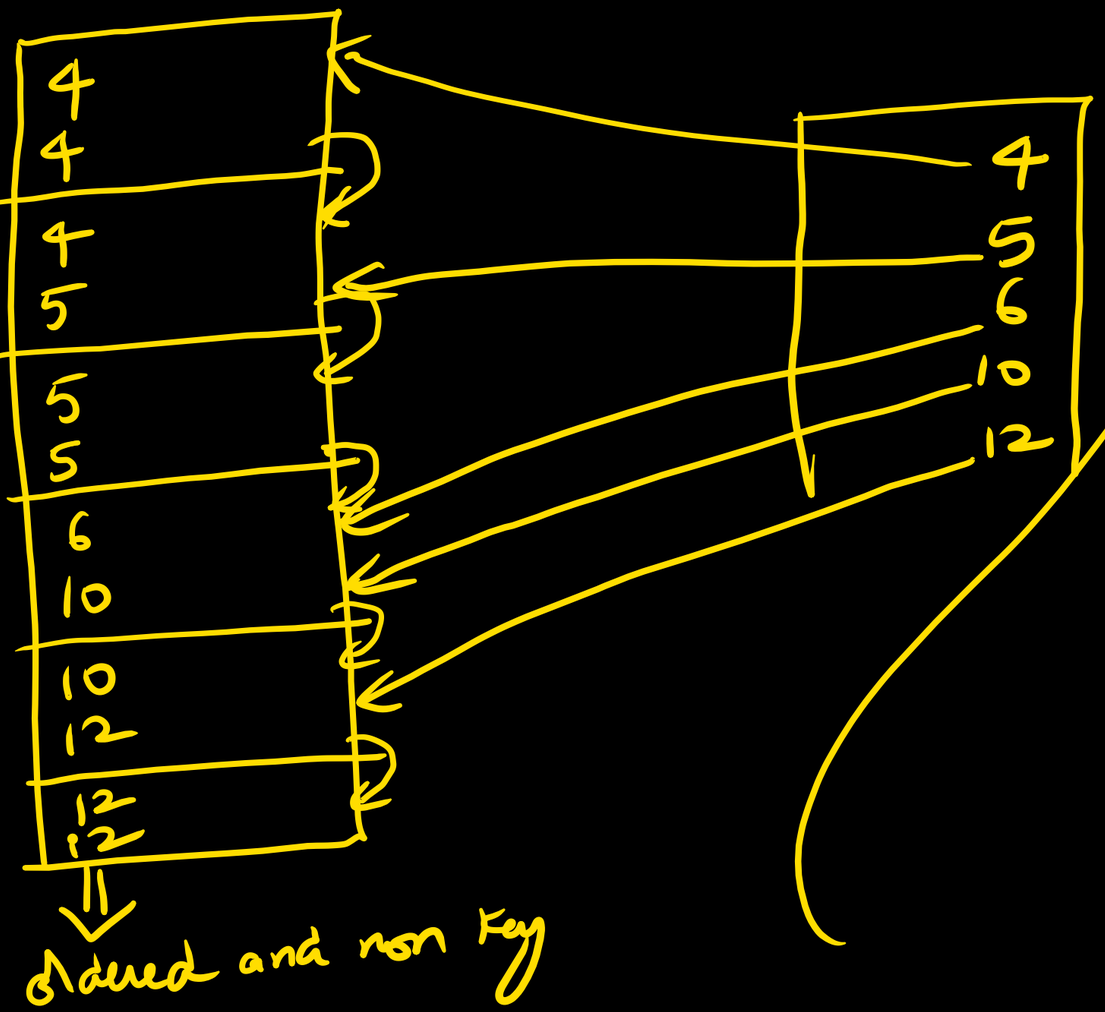


→ For a DB Table atmost one PK can be there.

Because PI is on ordered (sorted) field and we can have only one field on which table can be sorted.

Clustering index: (Ordered and not Key)

Repetitions



I/O cost with CI with multilevel

indexing = K + (one/node block to BT)

↓

levels of indexing.

4 → 2 blocks

5 → 2 block.

CI

Clustering index

Sparse

Repetition is there.

For every cluster we
use one index record

dense

Repetition is not there.

Then for every data record,
there is a index record

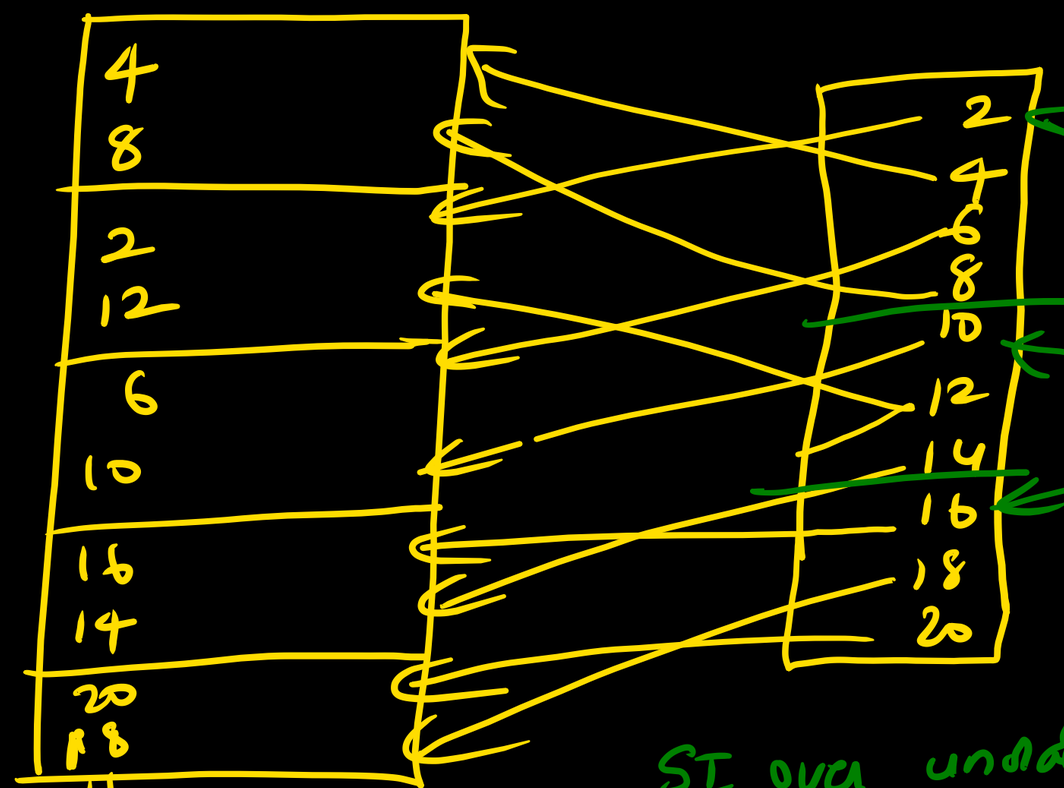
Atmost one clustering index is possible. (Because CI is on sorted field)

Either Primary index or clustering index, only one is possible

Secondary Index: (unordered field + key)

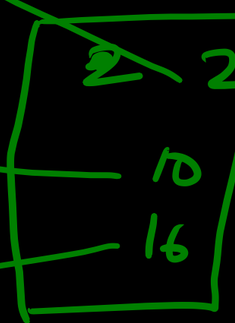
many SIs are possible for DBT because ordering is not required

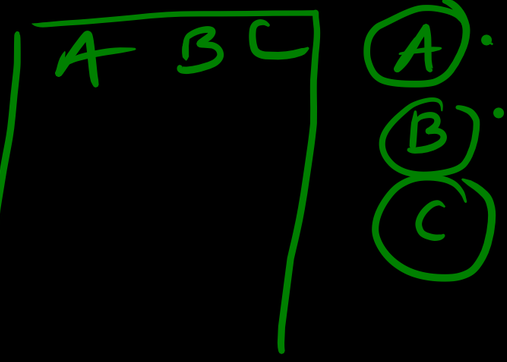
I/O cost = (K+1) → no of levels of indexing



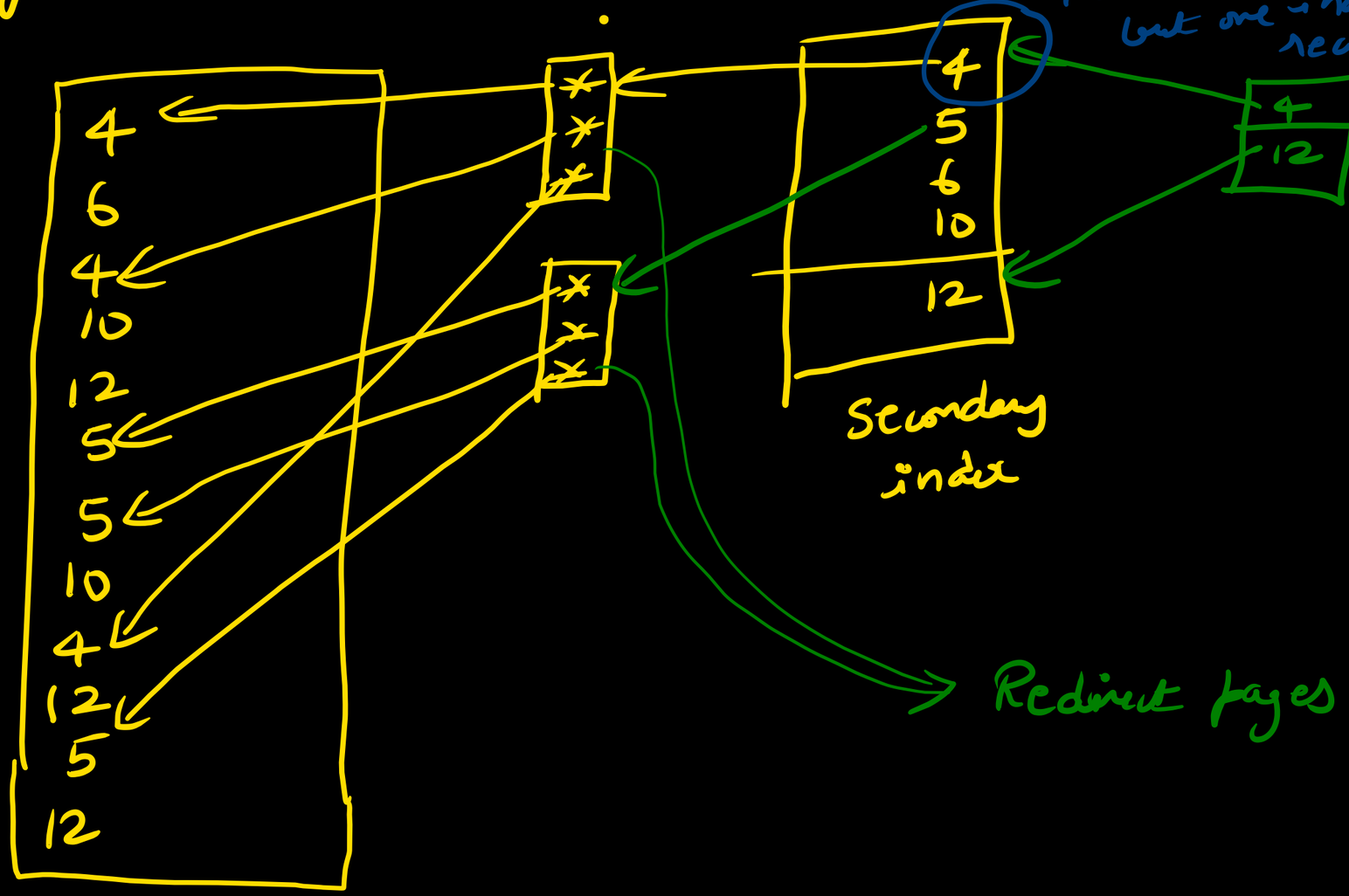
Key → no repetition
unordered

SI over unordered + key is always dense





Secondary index over non key (repetitions)



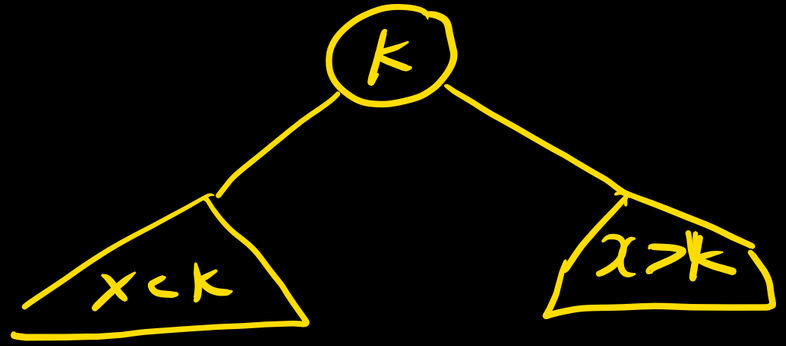
4 3 values & 4 but one index record

SI over non key is mostly sparse

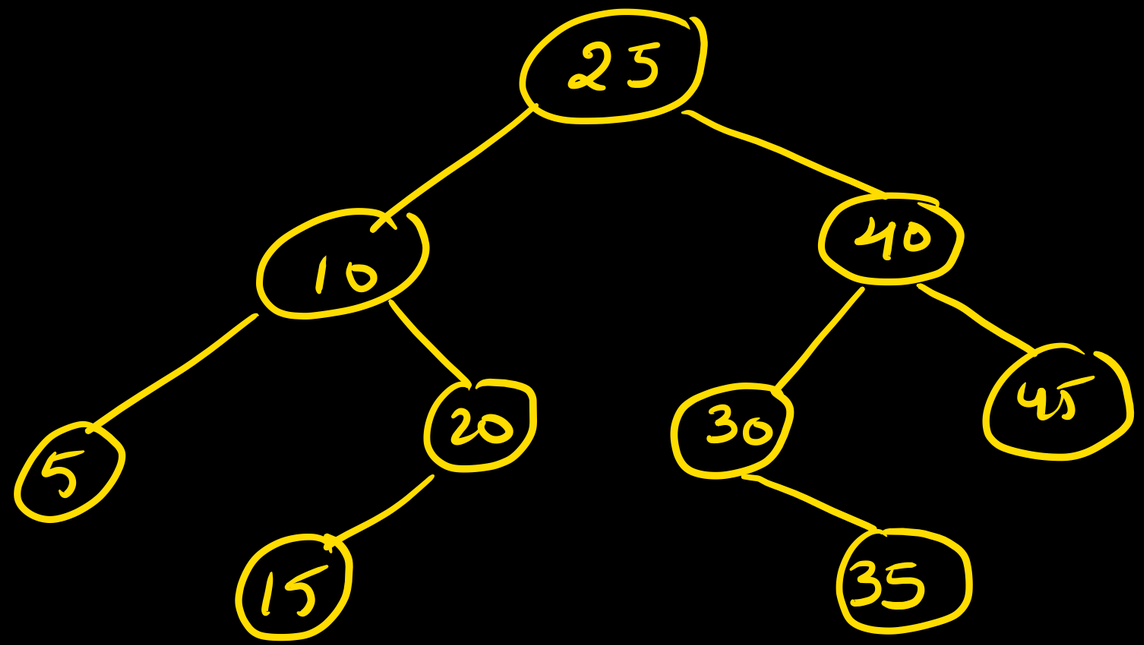
Dynamic ~~multi~~ multi level index:

B tree
B^+ tree
} Balanced seach trees.

Search tree: Data structure that organised keys



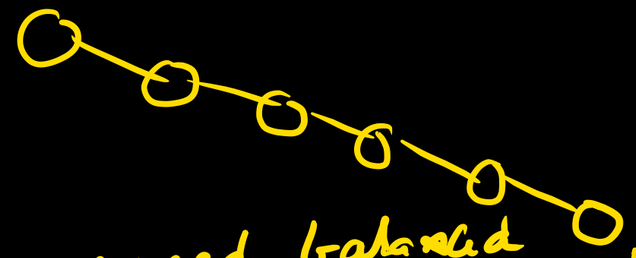
Binary search tree:



BST is optimized for searching.

Disadv of BST:

worst case time complexity is O(n) for search.



So we need balanced search tree which doesn't have O(n) worst case time.

we need balanced ~~search~~ search tree which has ~~worst~~ c

worst case time complexity as $O(\log n)$ not $O(n)$

Balanced search tree:

max height of search tree should not exceed $O(\log n)$

for n distinct keys.

worst case search cost does not exceed $O(\log n)$

Balanced Search tree

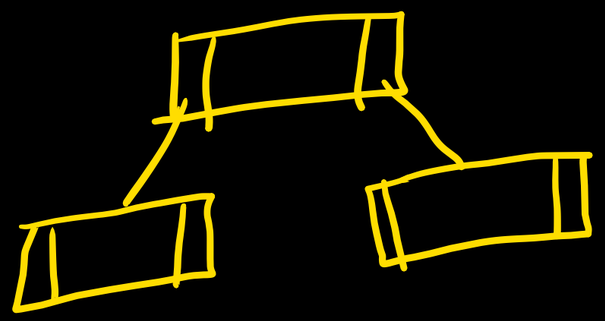
AVL

Red/black

Height restricted

Binary seach tree

Degree : 2



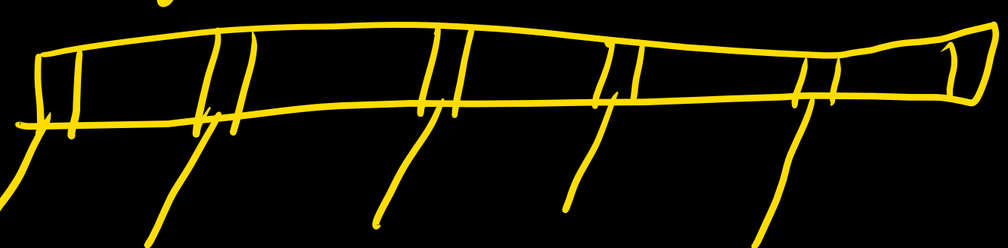
B tree

B^+ tree

prefered for OB

Height restricted seach tree

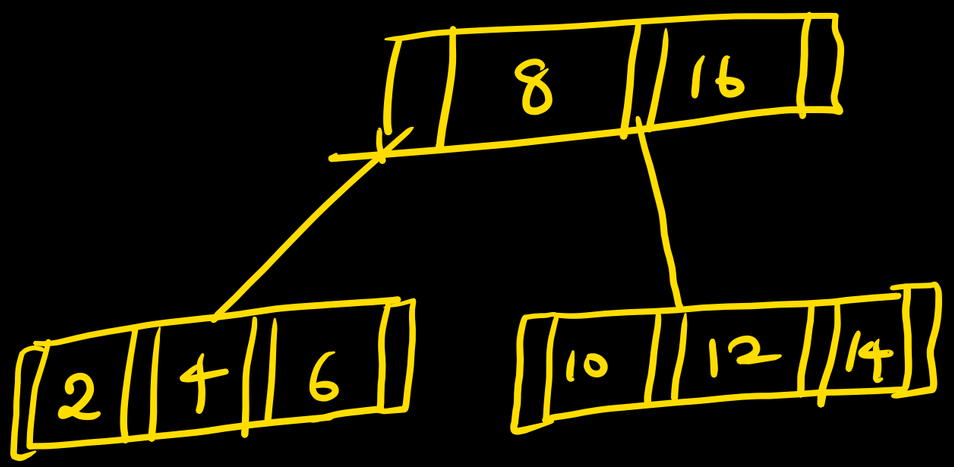
degree is p.



A node can have many keys

and many child pointers

End of Btree:

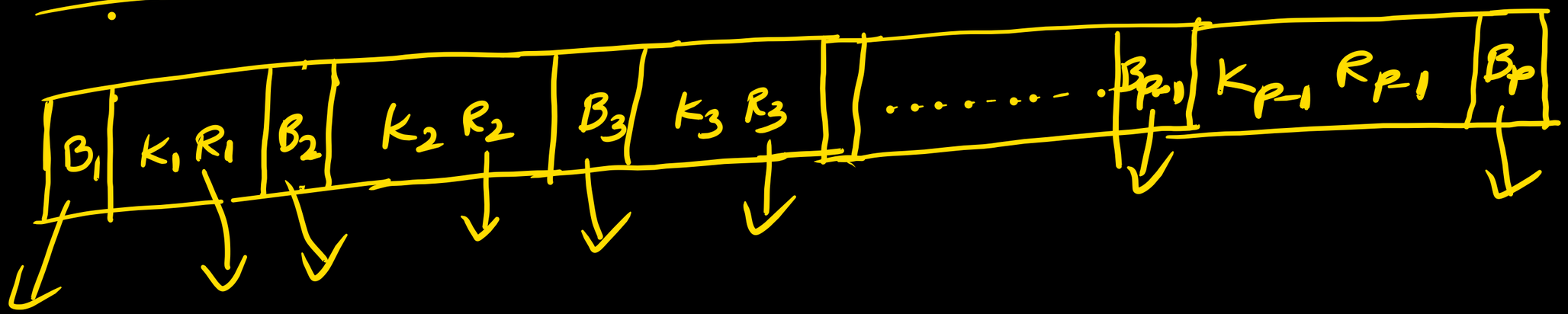


B tree definition :-

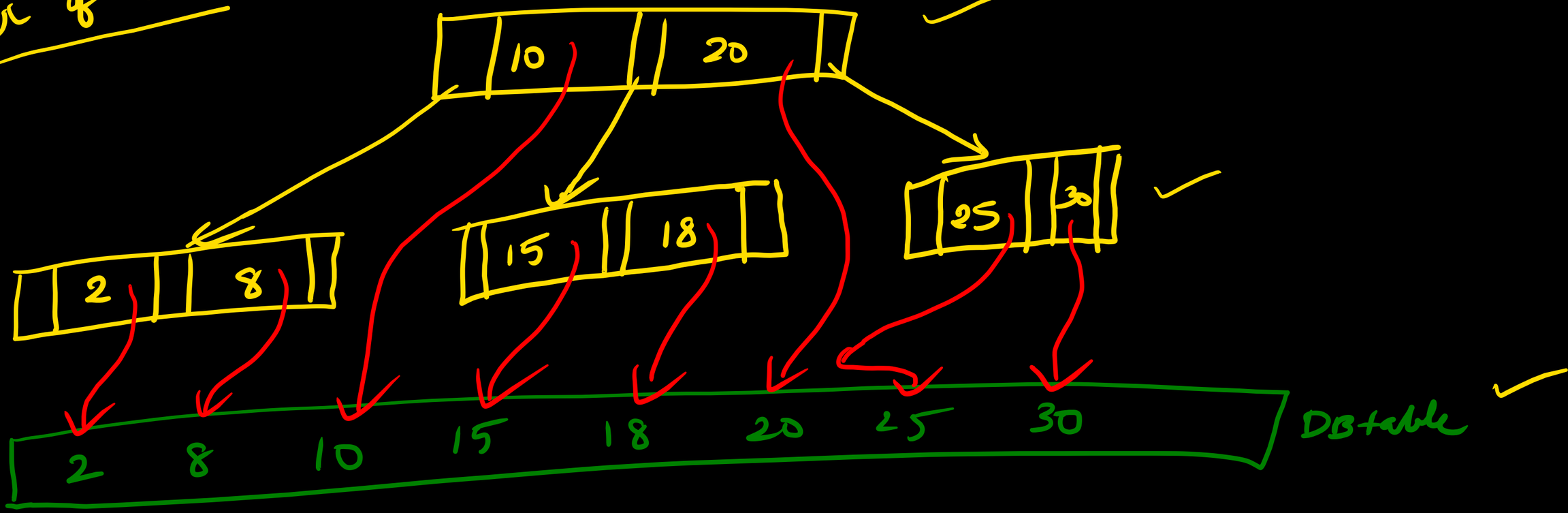
What is order P?

maximum possible block pointers which can be present in B tree node

1) Node structure: ✓



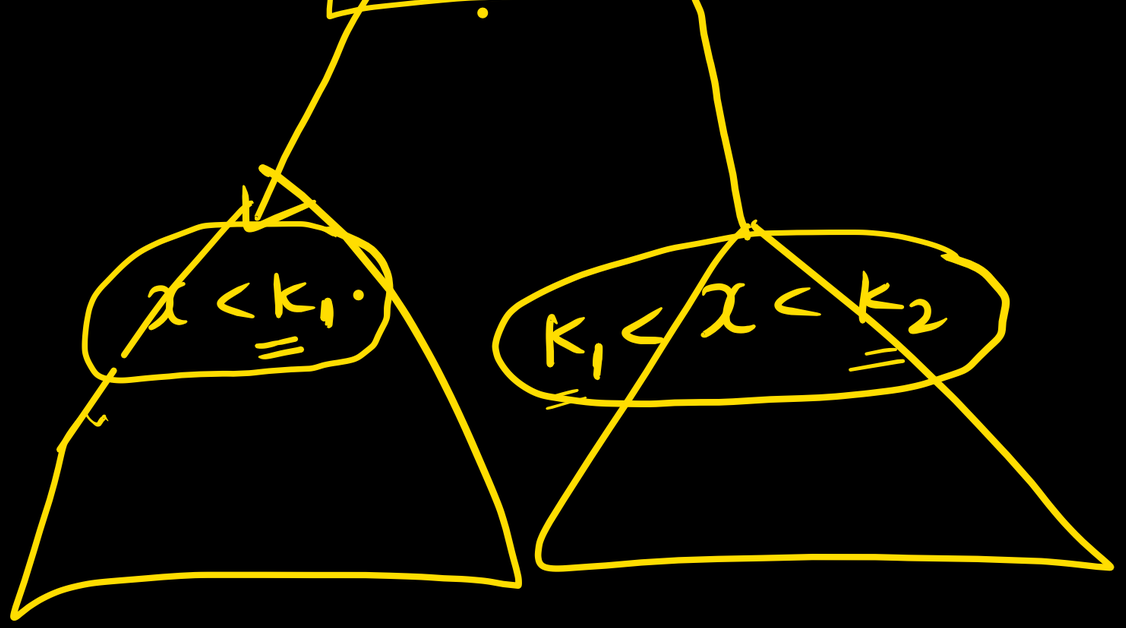
Ex of BT

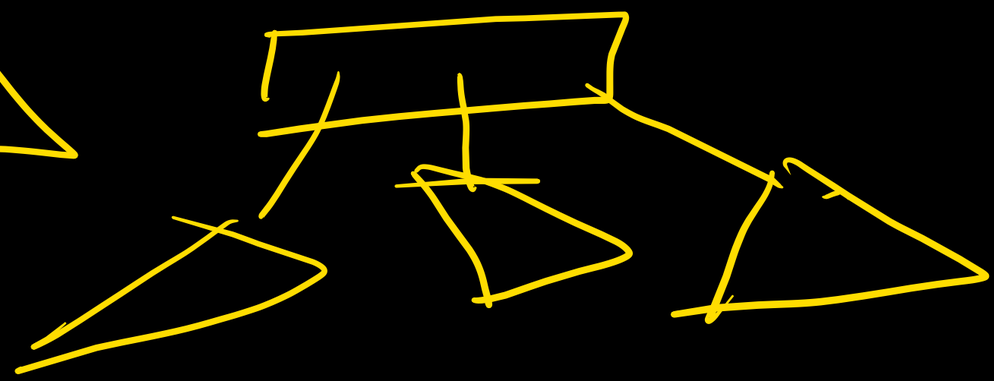


Yellow arrows are block pointers

Red arrows are record pointers

B tree node



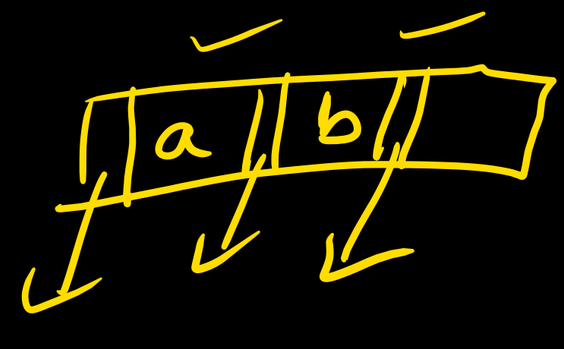


2) Every internal node except Root must be atleast $\lceil p/2 \rceil$ block pointers with $\lceil p/2 \rceil - 1$ keys.

Assume $p = 5$

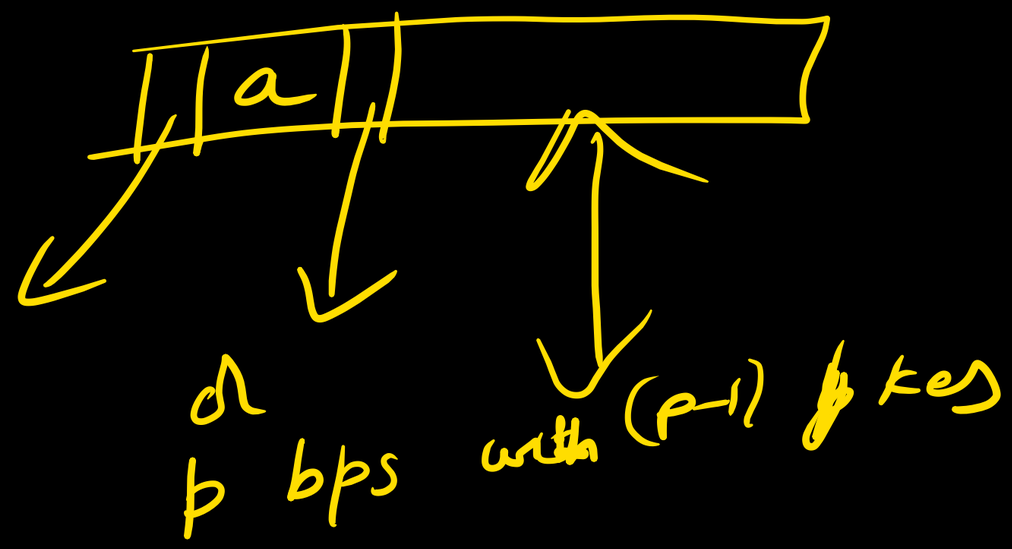
$\lceil p/2 \rceil = \lceil 2.5 \rceil = 3$

$\lceil p/2 \rceil - 1 = 2$



Every internal node can have atmost 'p' block pointers and (p-1) keys

3) Root node can have atleast 2 bp's with 1 key
and atmost p bps with p-1 keys



2 bps $\Longleftrightarrow$ p bps
min max
1 key $\Longleftrightarrow$ (p-1) keys

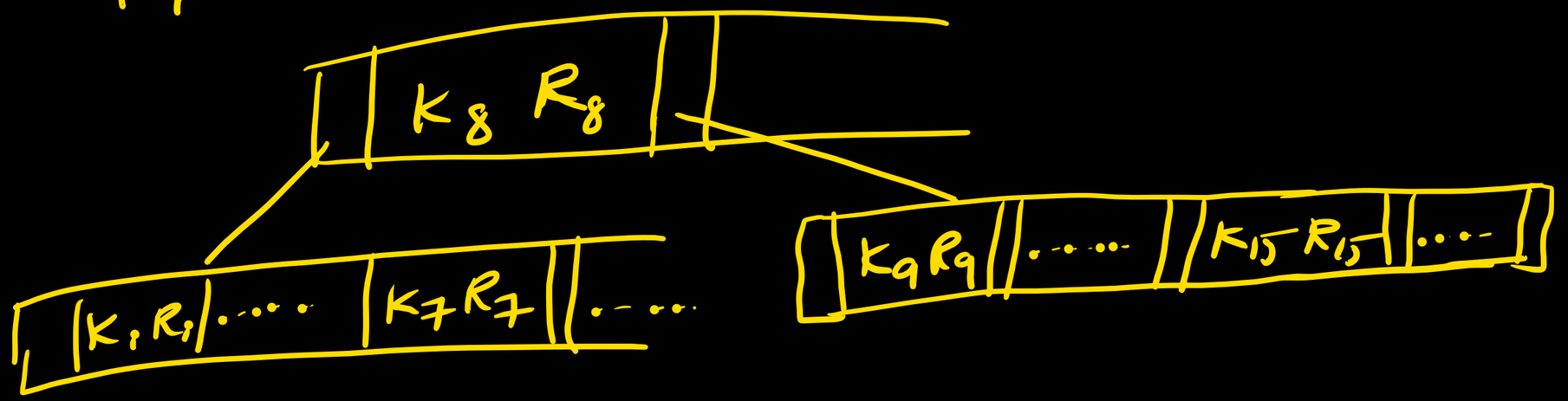
4) Every leaf node must be at the same level.

② ③ ④ are called balancing conditions

Ex: order P:15 (max bp's = 15)

The node is full. we want to insert K_{15} R_{15}

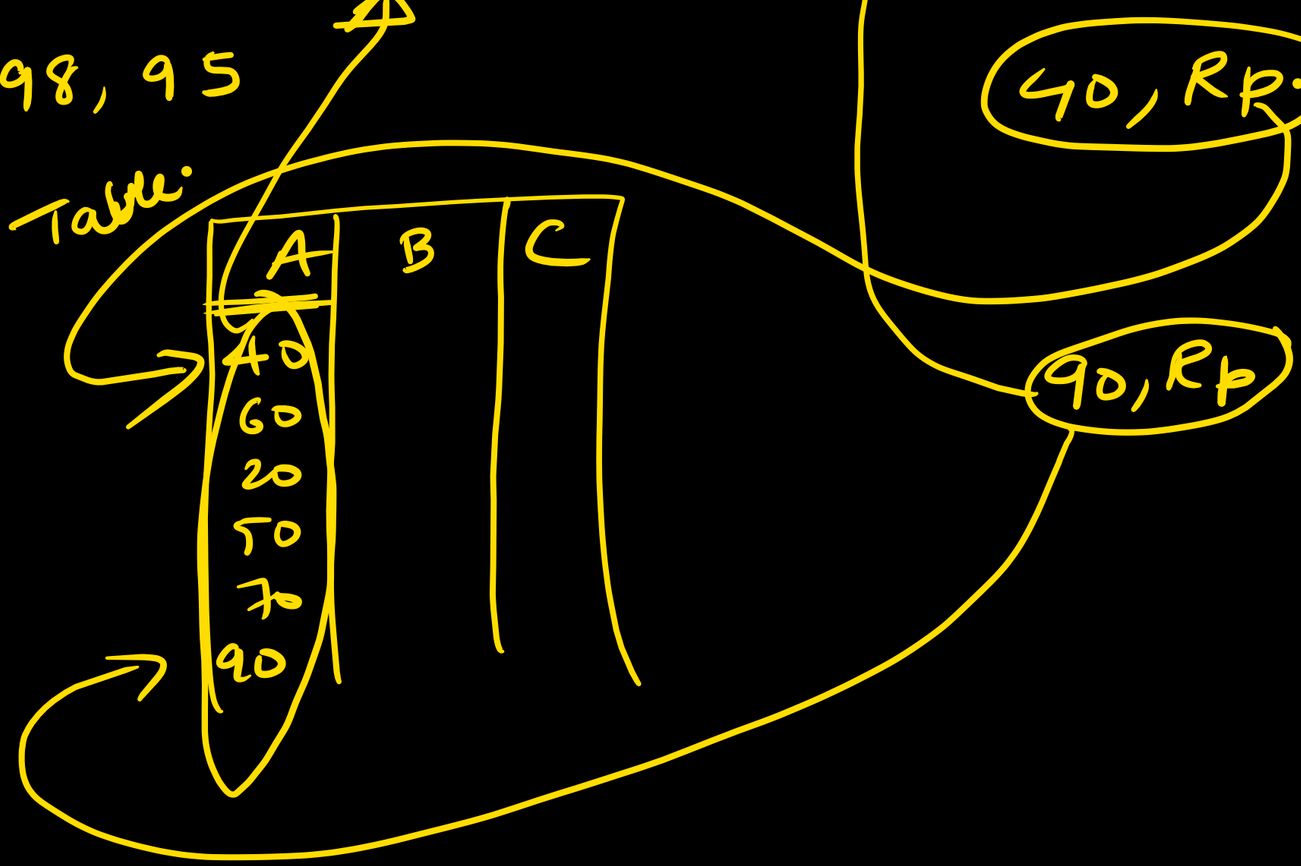
$K_1 < K_2 < K_3 \ldots < K_8 < \ldots \ldots K_{15}$



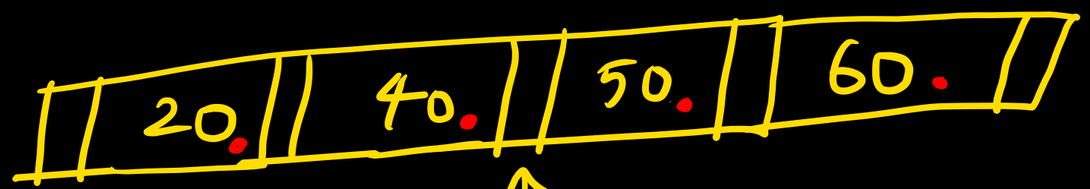
In B trees except Root node, all the nodes are atleast 50% occupied.

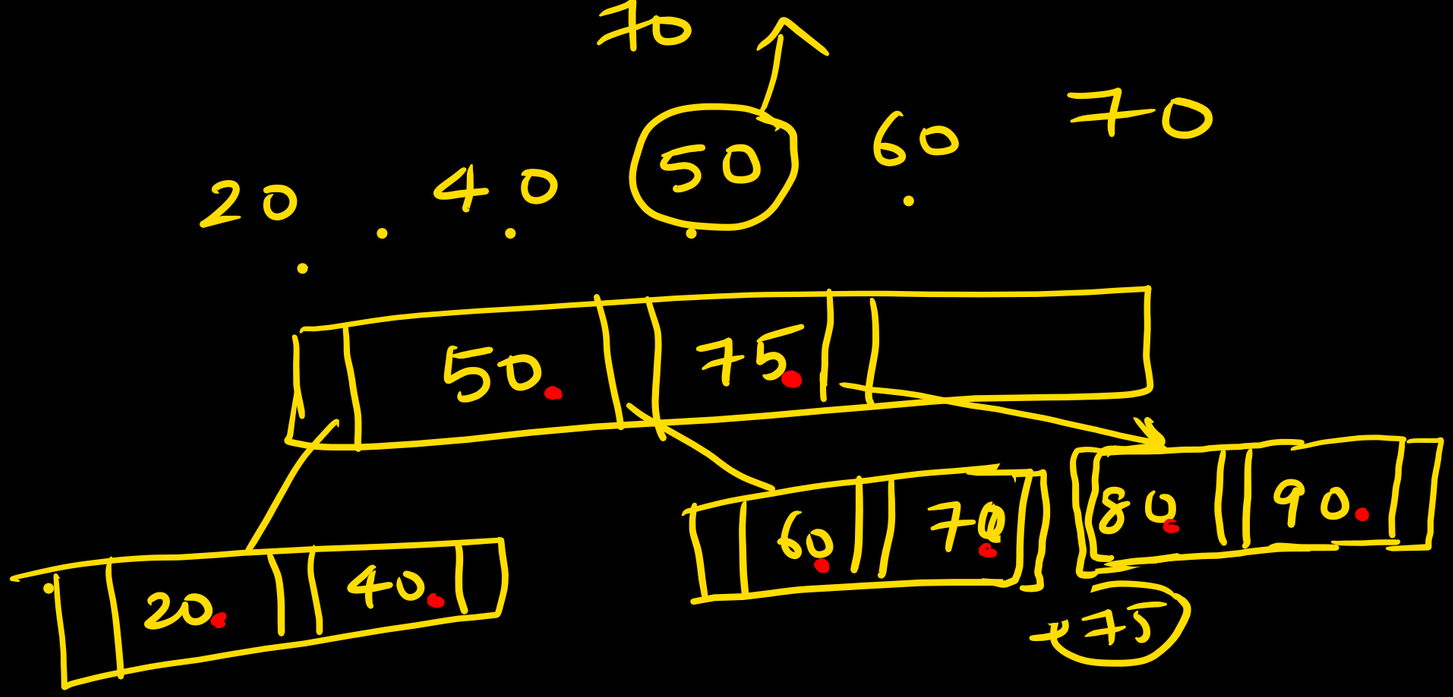
Construct a B tree order 5 : → (max BPs are 5, max keys = 4)

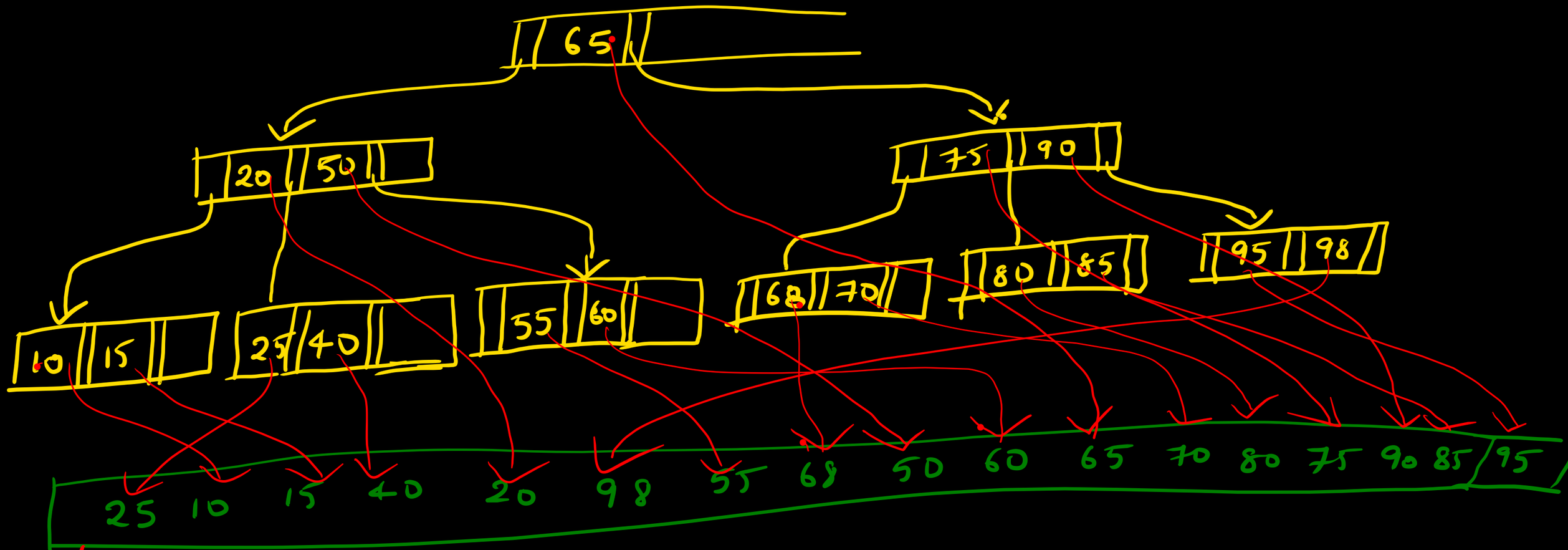
Keys: 40, 60, 20, 50, 70, 90, 80, 75, 10, 15, 25, 55, 65, 68, 85, 98, 95



40, 60, 20, 50, 70, 90, 80, 75







DBT

Btree index can be dense or sparse

In this example, it is dense

→ If the id indexing is done on ordered field, then we get sparse index and we need only one key ~~per~~ per block.

→ If the indexing is done on unordered field, then we get dense index. one key per record.

① B trees are suitable for random access of one record

Ex: Select * from R
where A = 40;

I/O cost : K + ①
levels of B tree

② B tree is not suitable for sequential access of a large no. of records.

queries . Select * from R where A ≥ 68 and A ≤ 85

68 — K+1
69 — K+1
⋮
85 — K+1

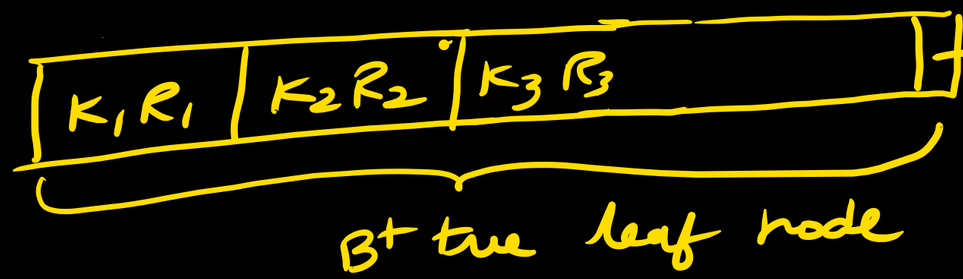
we go for B^+ trees where random access is possible and also range queries also possible in less time.

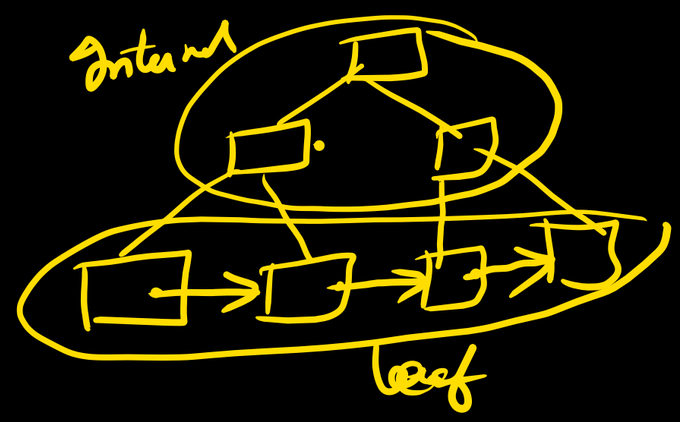
B+ tree:

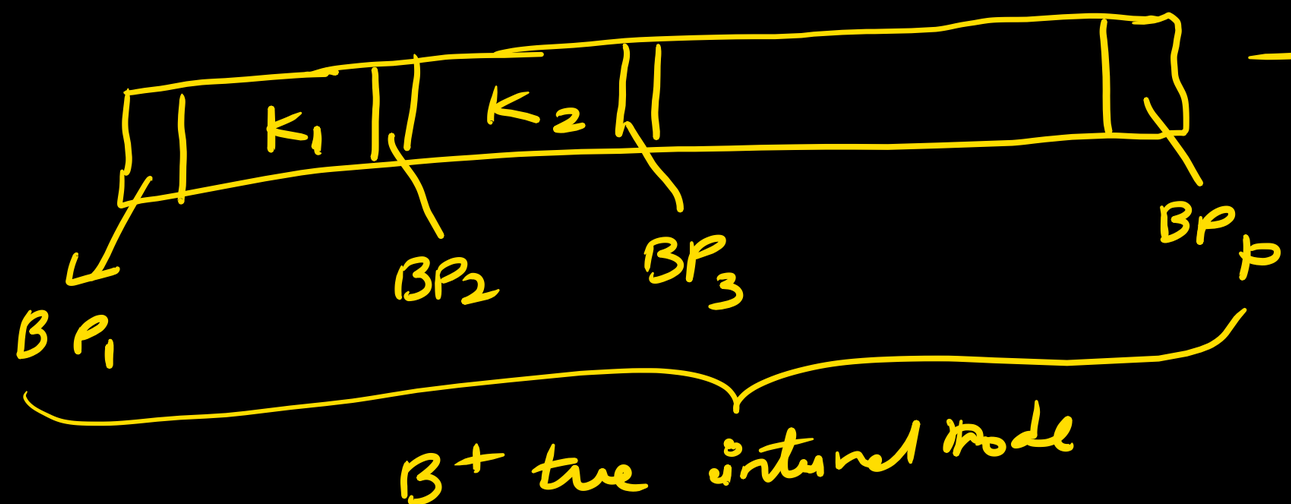
what is order P? P is maximum possible pointers that can be stored in B+ tree node.

1)

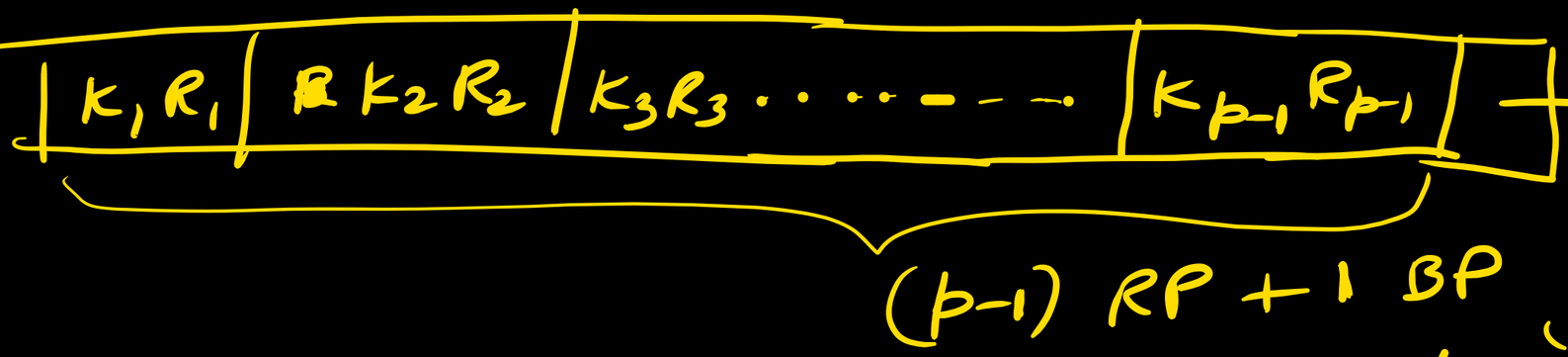
1) node structure







Leaf node: Set of (key, R_p) pairs and one block pointer which points to next leaf.



$\therefore$ total pointers $= p$.

Internal nodes:

only keys and block pointers