B-trees And B^+-trees

- B-tree = Dynamic multi-level index.

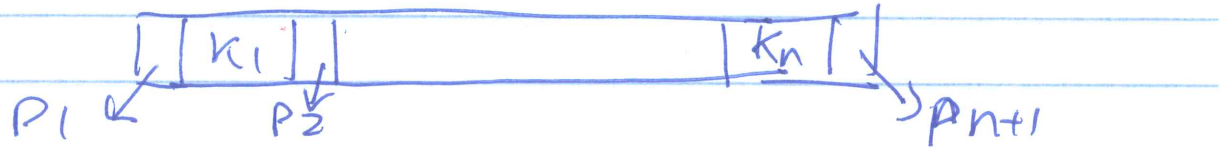
(# levels in a B-tree depends on # search keys stored).

Property: every block (node) in B-tree is ≥ (at least) ½ full.

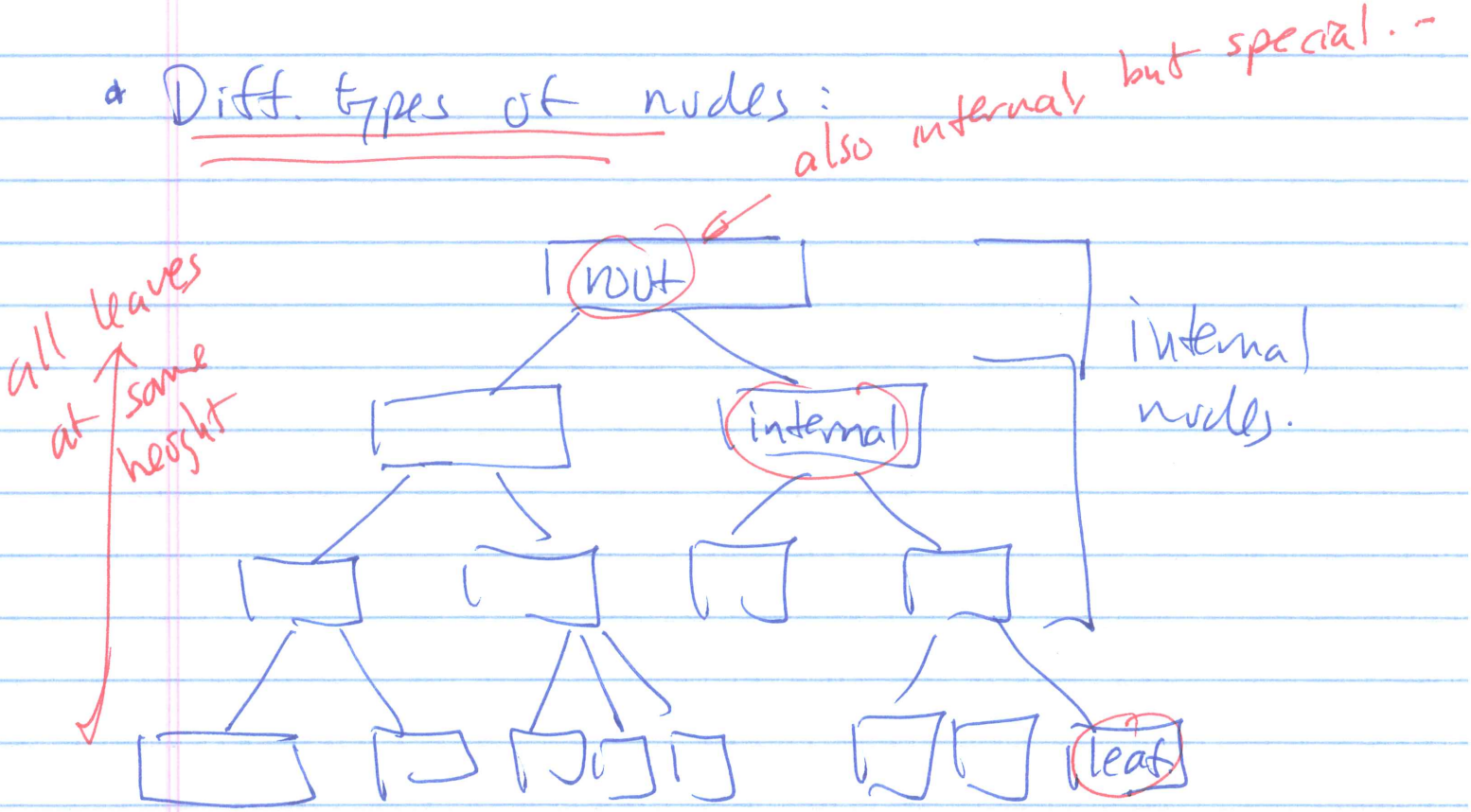
– except for the root node

- Different fill criteria for:
  (1) root node
  (2) internal node
  (3) leaf node

- Node structure (general) — n keys fields, n+1 pointer fields

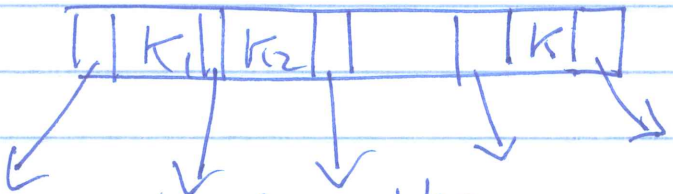


- Diff. types of nodes:



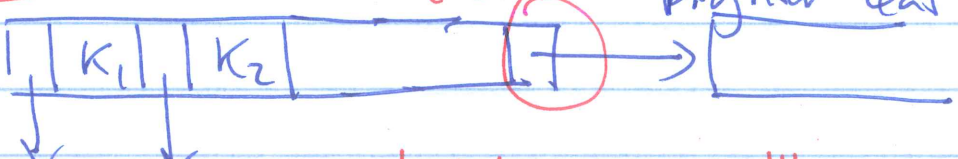
- Structure of nodes:

root + internal node:



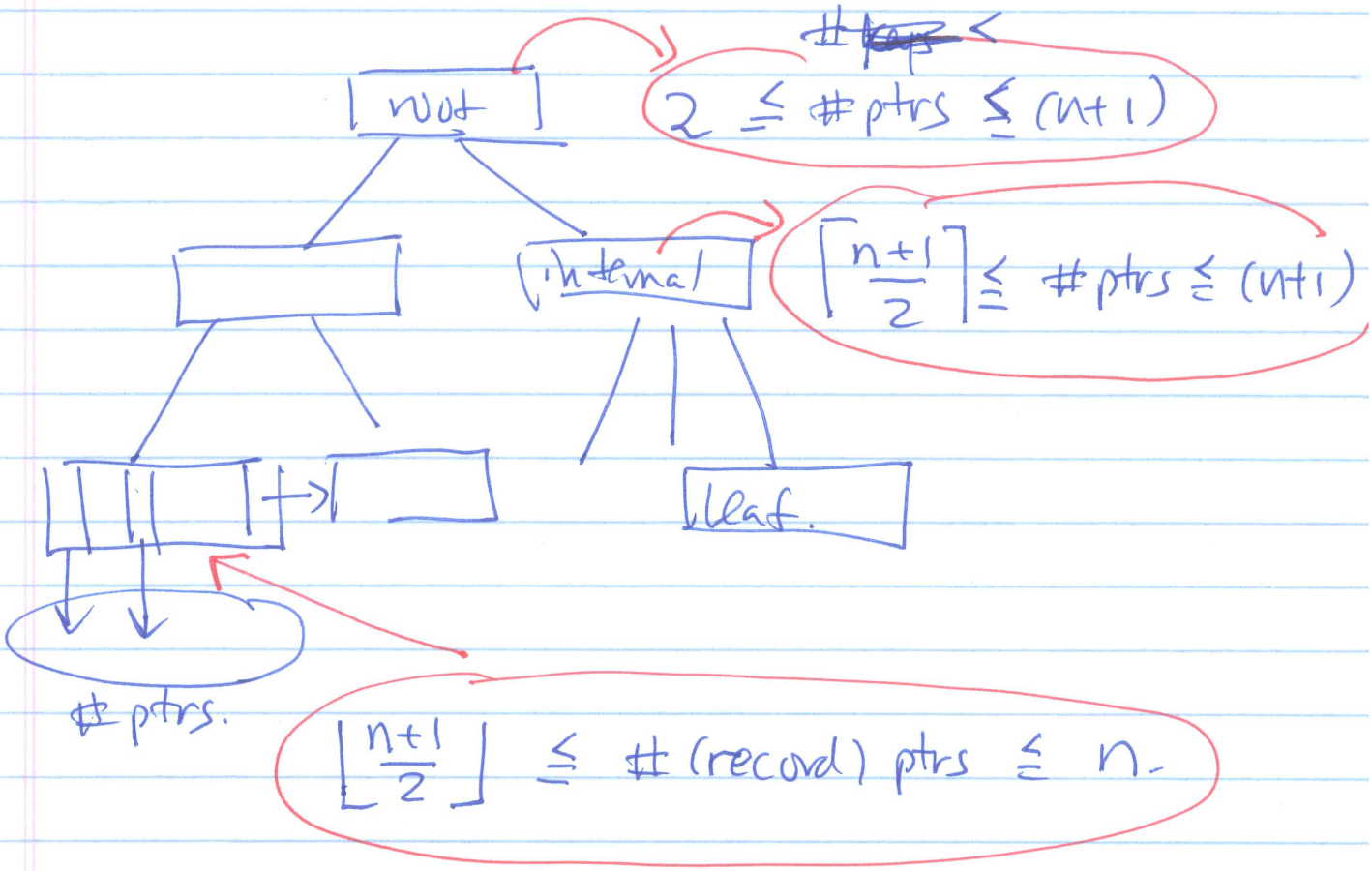
points to other nodes (block). in B-tree.

Leaf node:



points to ~~record~~ data record !!!

- Fill level constraints on diff. types of nodes:



NB:

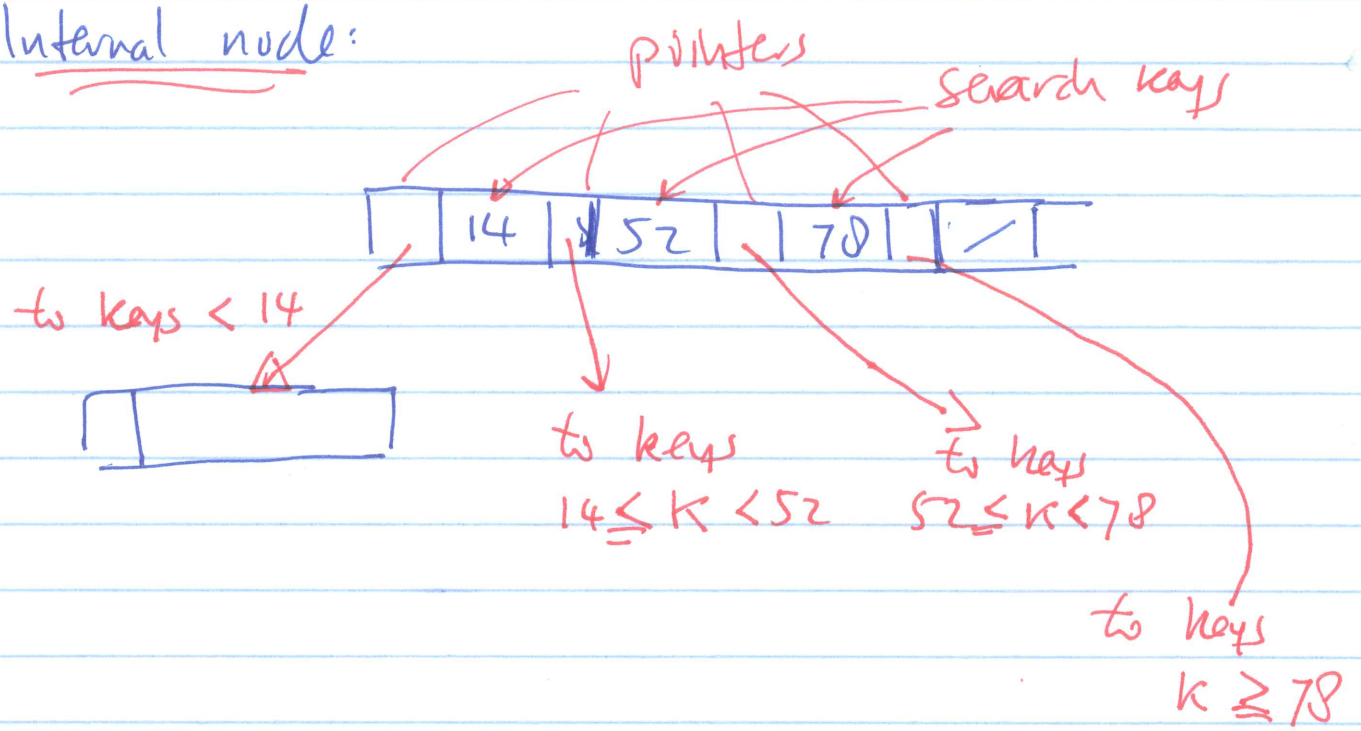
Internal node : # keys = # ptrs − 1.

Leaf node : # keys = # record pointers
(+ 1 sibling ptr).

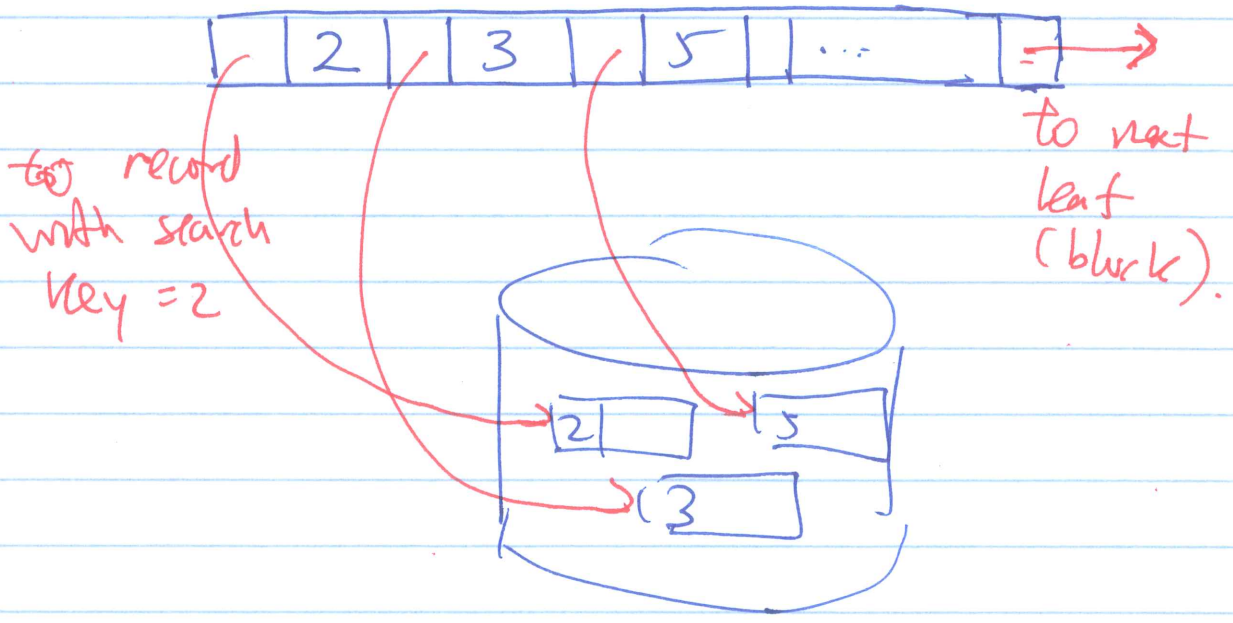
More detailed.

Structure of the nodes in a B^+-tree:

① Internal node:

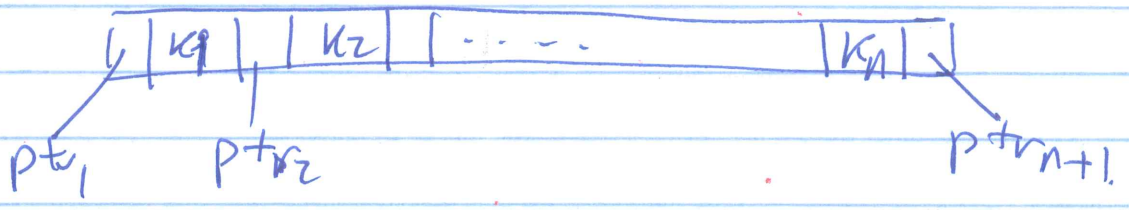


② Leaf node:



The parameter n (# search keys)

- Always pack a block with the max. # search keys + pointers



- 1 block contains:

n	search keys	(size = S_1)
n+1	pointers	(size = S_2.

Then: $n S_1 + (n+1) S_2 \leq$ block size

find largest n that fits!

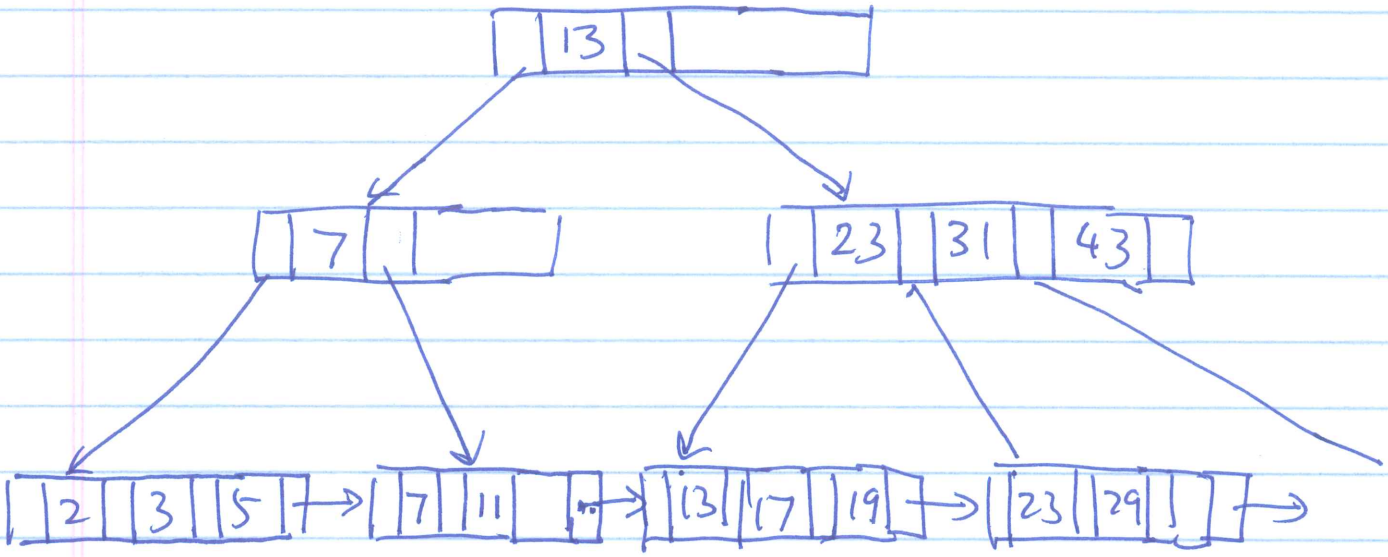
- Example:

Block size = 4096 bytes
key size = 4 bytes (int)
pointer size = 8 bytes

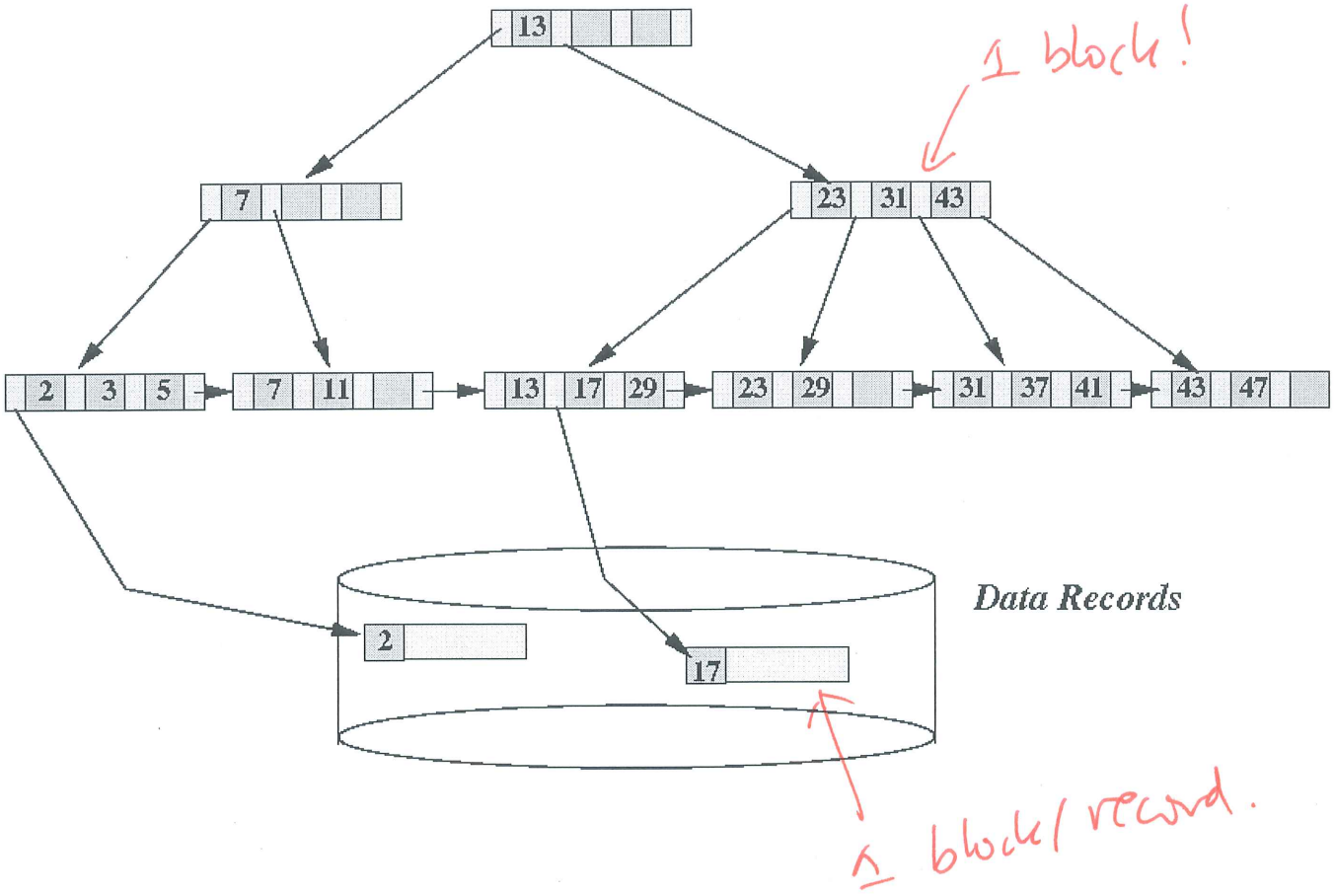
$4n + (8)(n+1) \leq 4096 \rightarrow n = 340$

$12n + 8 \leq 4096 \quad (12n \leq 4088)$

Example: B^+-tree (with $n = 3$).



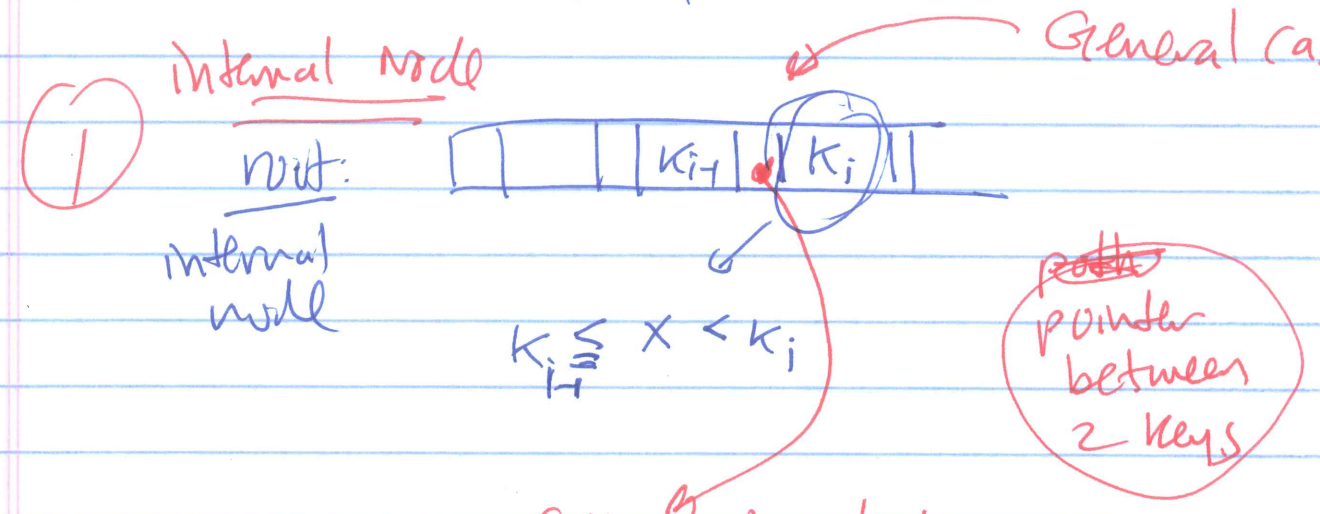
(see printout).


13
7
23 31 43
2 3 5
7 11
13 17 29
23 29
31 37 41
43 47
Data Records
2
17
1 block!
1 block/record.

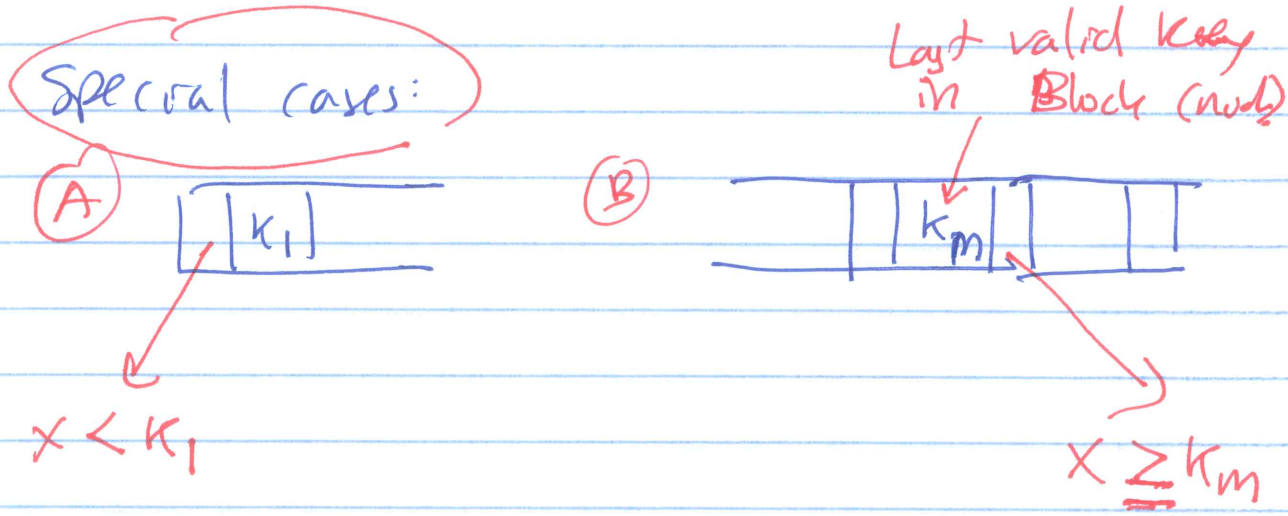
- Lookup in B+ trees:

Search for search key X:

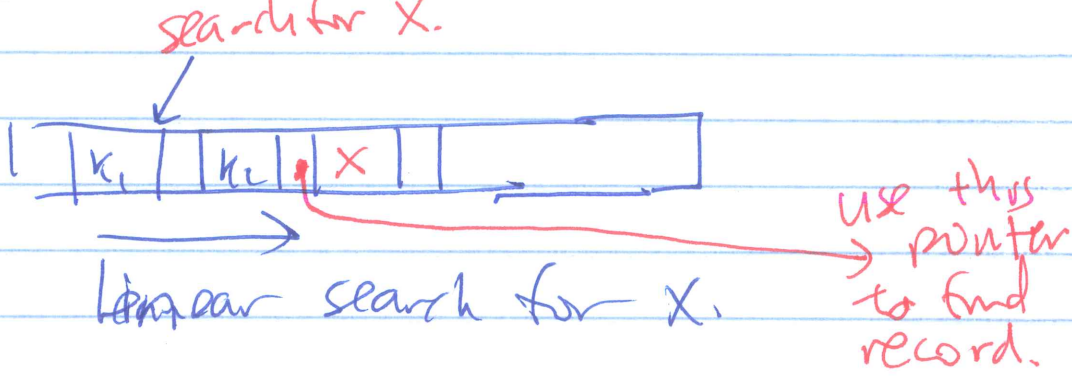
(1) Internal node



Special cases:

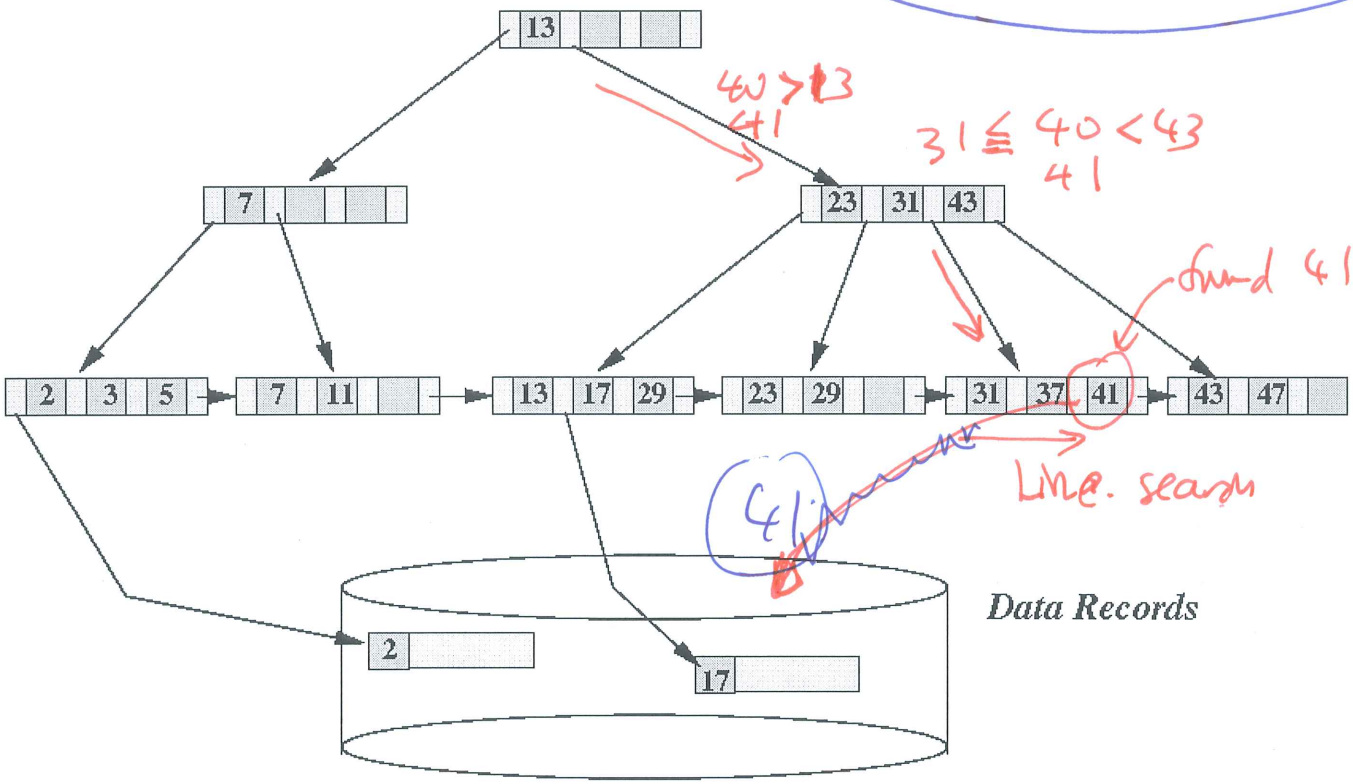


(2) External node:



Searching in B-tree

Example: search for search key 40 and 41



40 > 13
41

31 ≤ 40 < 43
41

found 41.

Line. search

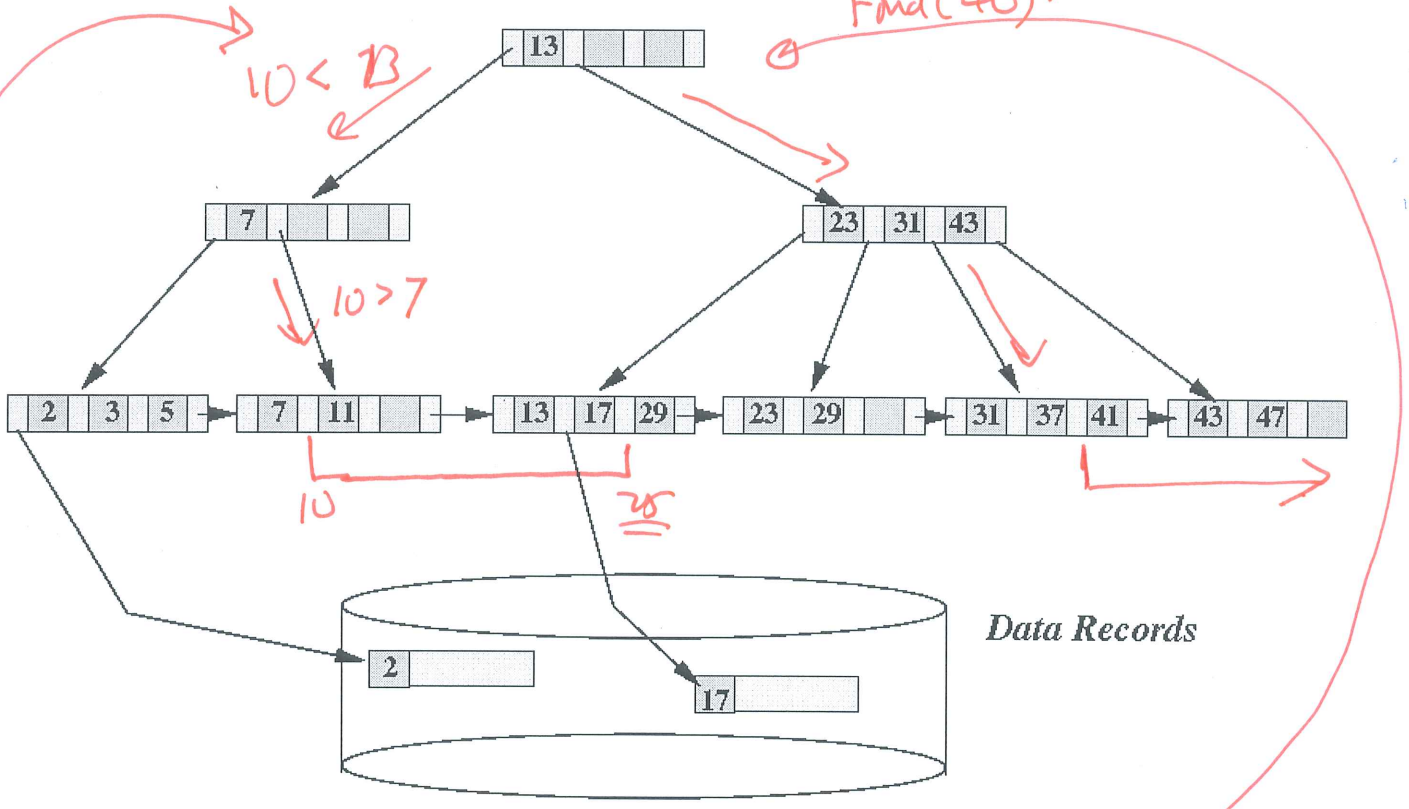
41

40 Not found → No record with key 40

☆ search for key 37 / 41.

land in same leaf.

Range Query using B+-tree



Query: select * From R
where R.key > 40.

Query: select * from R
Where R.key ≥ 10
and R.key ≤ 25.

You must use Right most ptr to go to the NEXT leaf!!!

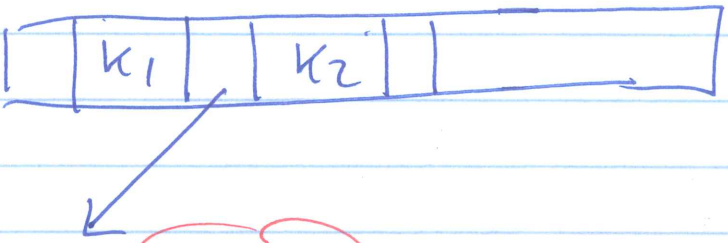
Variation in B-tree: duplicate key values

- When search key values can be duplicate

(i.e.: secondary dense index)

search key is NOT a key in the relation.

(1) then the meaning of the pointers are as following:



leads to FIRST occurrence of all keys

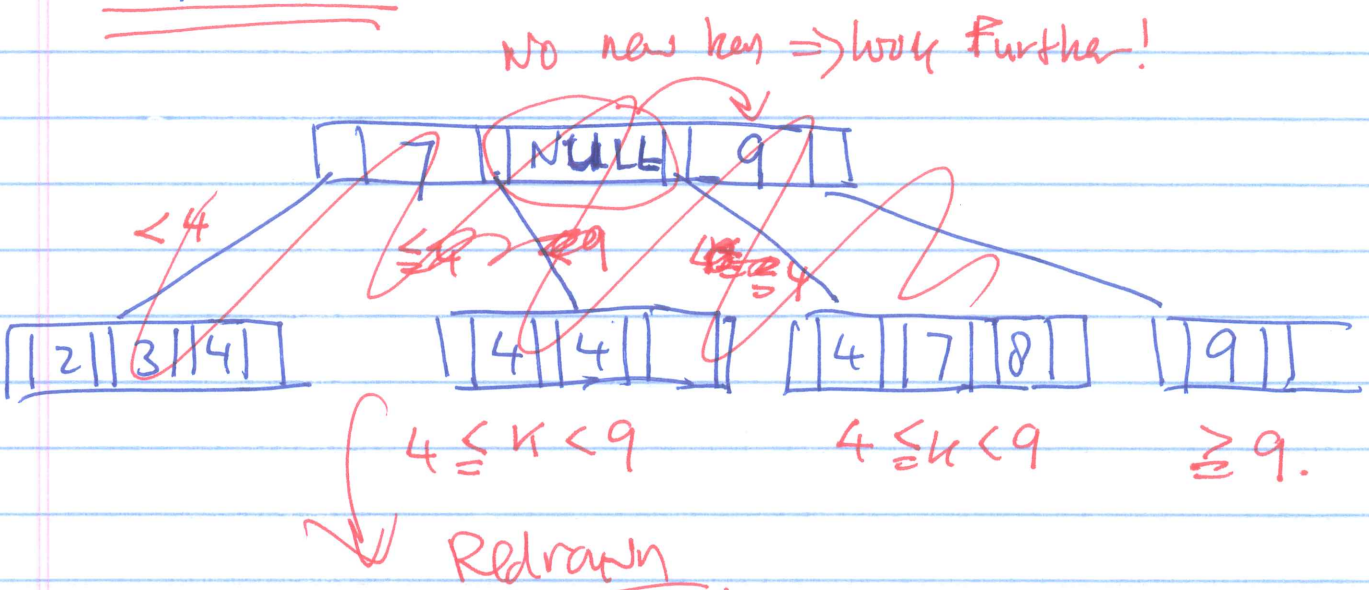
$$K_1 \leq K < K_2.$$

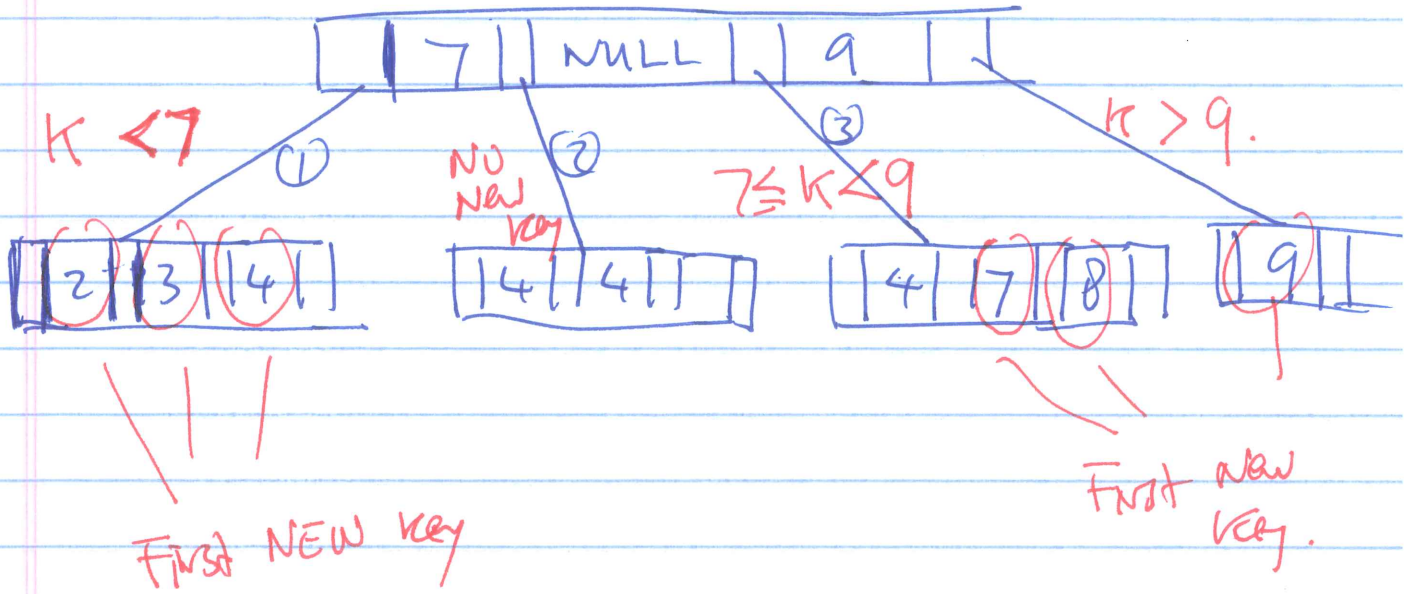
(2) We also introduce a NEW search key value:

NULL

meaning: There are NO new search keys

Example of use of NULL:





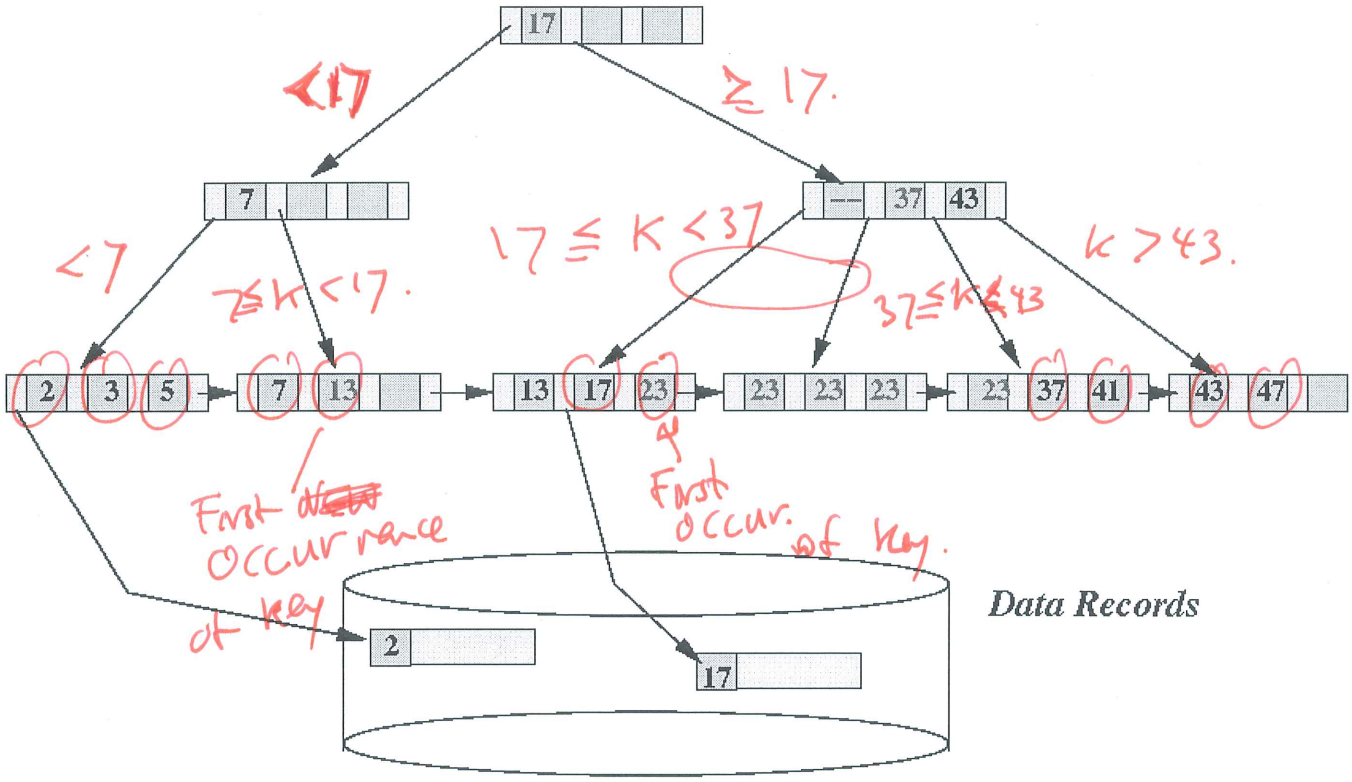
☆ IF you search for key = 8

① : 8 ≥ 7. don't follow ①

② : NO new keys : don't follow ②

③ 7 ≤ 8 < 9 → use ③.

Example: B^+-tree with duplicate keys.



Find 13: follow <17.

then 7≤K<17. → first occurrence of 13.

Find 23.: follow ≥17.

then FIRST pointer