B+ Tree

- What is a B+ Tree
- Searching
- Insertion
- Deletion

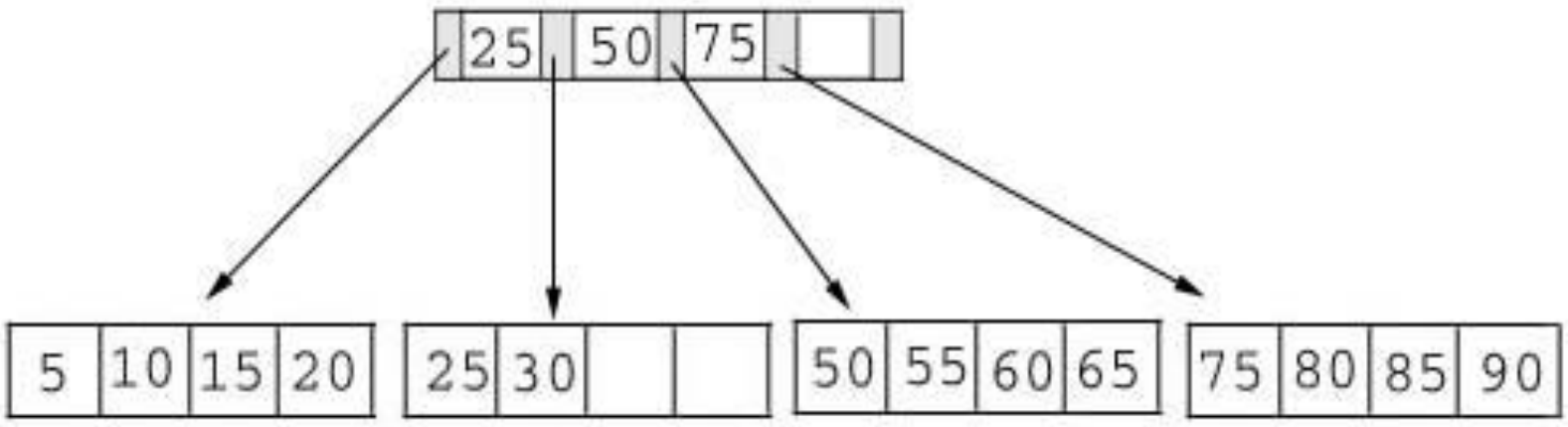
What is a B+ Tree

- Definition and benefits of a B+Tree

1.Definition: A B+tree is a balanced tree in which every path from the root of the tree to a leaf is of the same length, and each nonleaf node of the tree has between [n/2] and [n] children, where n is fixed for a particular tree. It contains index pages and data pages. The capacity of a leaf has to be 50% or more. For example: if n = 4, then the key for each node is between 2 to 4. The index page will be 4 + 1 = 5.

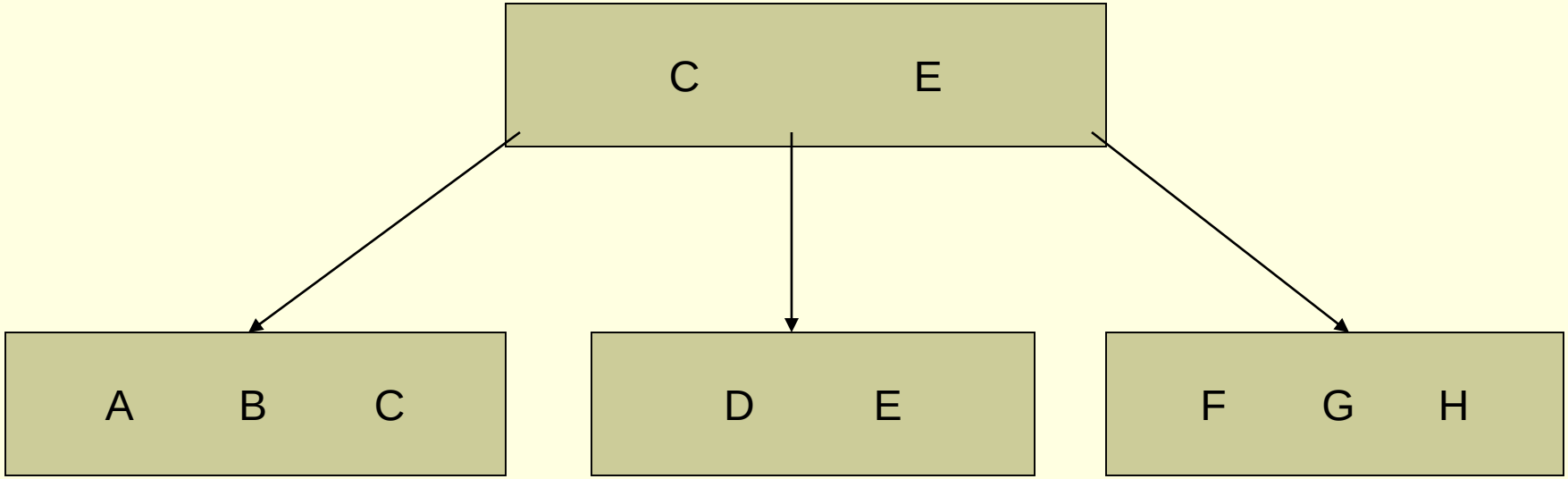
Example of a B+ tree with four keys (n = 4) looks like this:

What is a B+ Tree



What is a B+ Tree

- Question: Is this a valid B+ Tree?



What is a B+ Tree

Answer:

1. Both tree in slide 3 and slide 4 are valid; how you store data in B+ Tree depend on your algorithm when it is implemented.
2. As long as the number of data in each leaf are balanced, it doesn't matter how many data you stored in the leaves. For example: in the previous question, the n can be 3 or 4, but can not be 5 or more than 5.

What is a B+ Tree

- Benefit: Every data structure has it's benefit to solve a particular problem over other data structures. The two main benefits of B+ tree are:

1. Based on it's definition, it is easy to maintain it's balance. For example: Do you have to check your B+ tree's balance after you edit it?

   No, because all B+ trees are inherently balanced, which make it easy for us to manipulate the data.

What is a B+ Tree

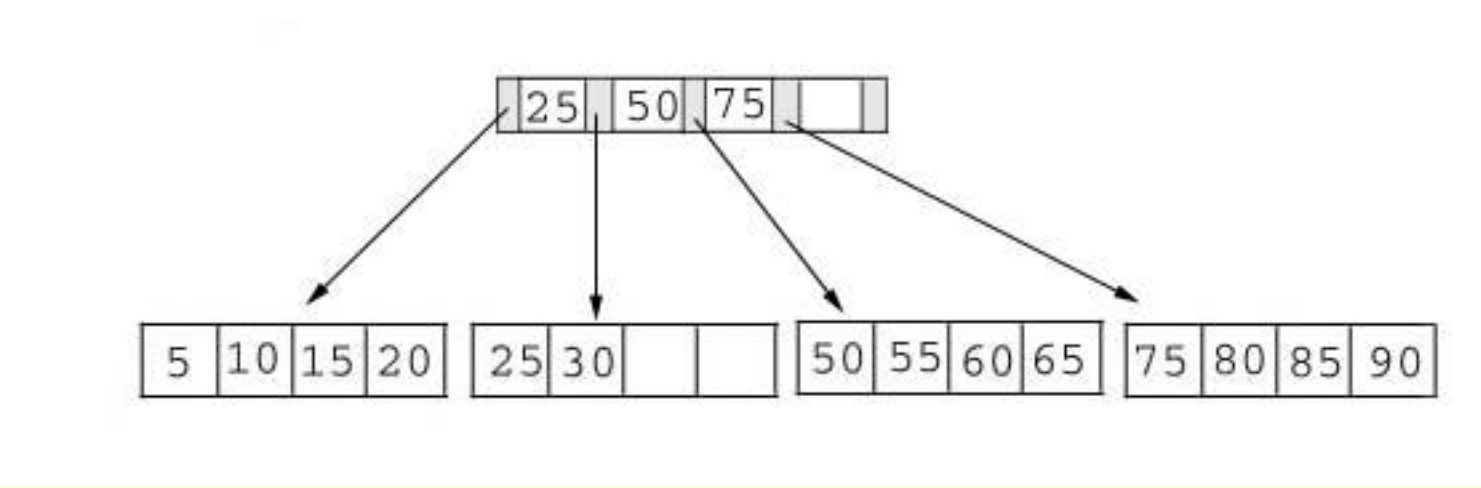
2. The searching time in a B+ tree is much shorter than most of other kinds of trees. For example: To search a data in one million key-values, a balanced binary requires about 20 block reads, in contrast only 4 block reads is required in B+ Tree.

   (The formula to calculate searching time can be found in the book. Page 492-493)

Searching

- Since no structure change in a B+ tree during a searching process, so just compare the key value with the data in the tree, then give the result back.

  For example: find the value 45, and 15 in below tree.



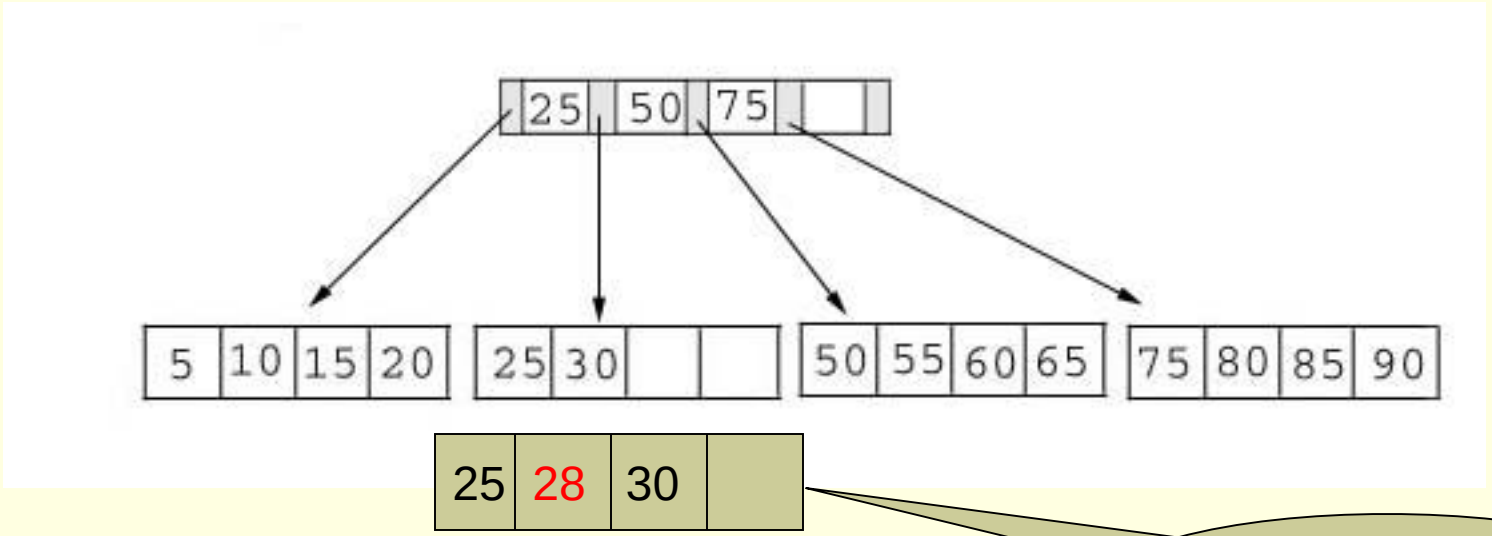
Searching

- Result:

  1. For the value of 45, not found.
  2. For the value of 15, return the position where the pointer located.

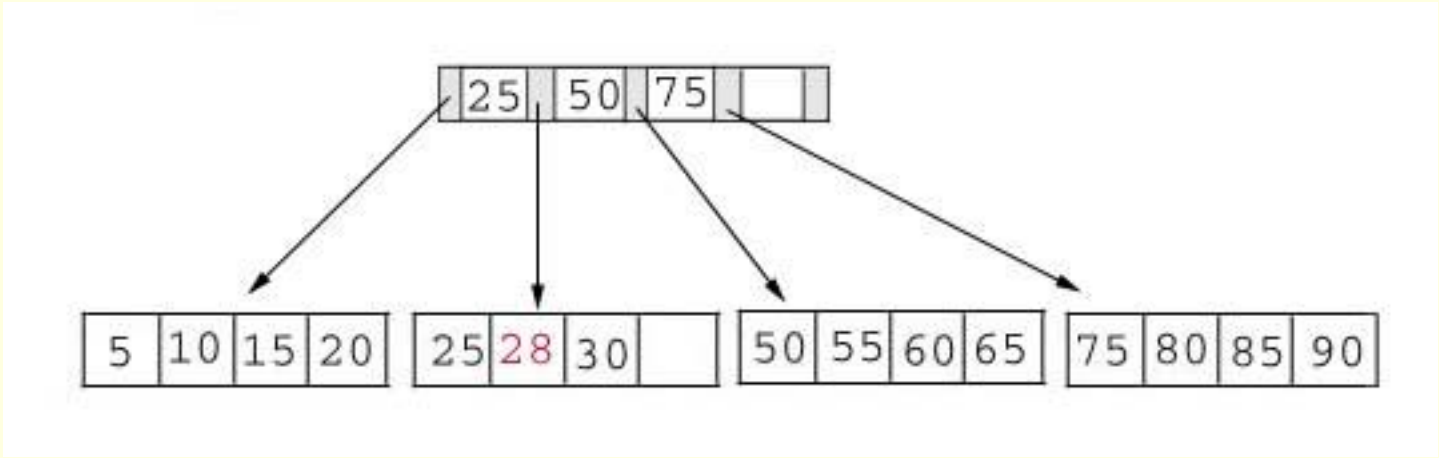
Insertion

- Since insert a value into a B+ tree may cause the tree unbalance, so rearrange the tree if needed.
- Example #1: insert 28 into the below tree.



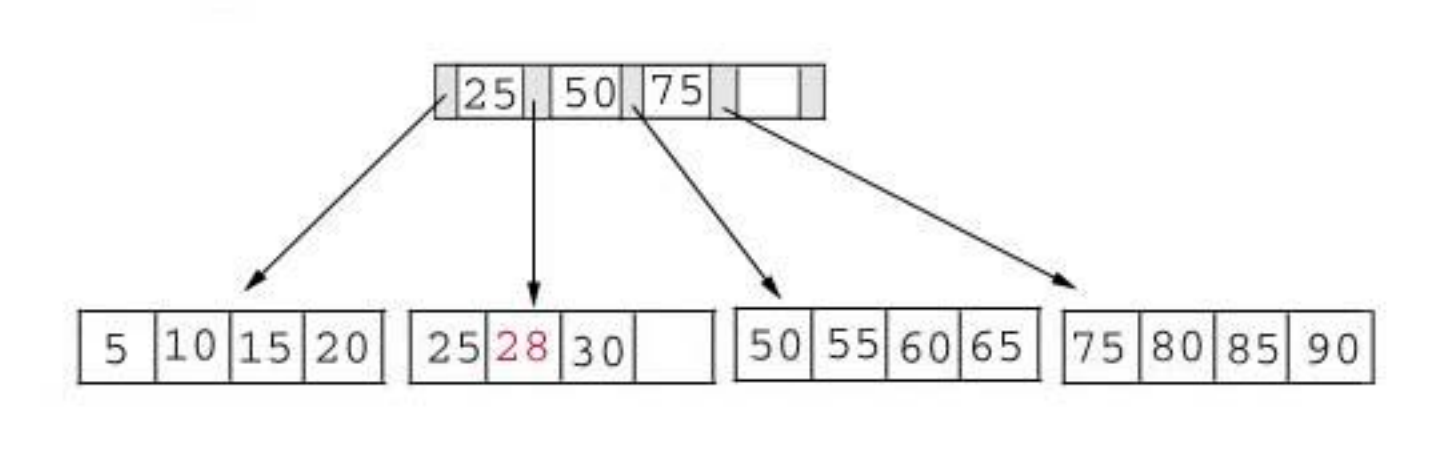
Insertion

- Result:



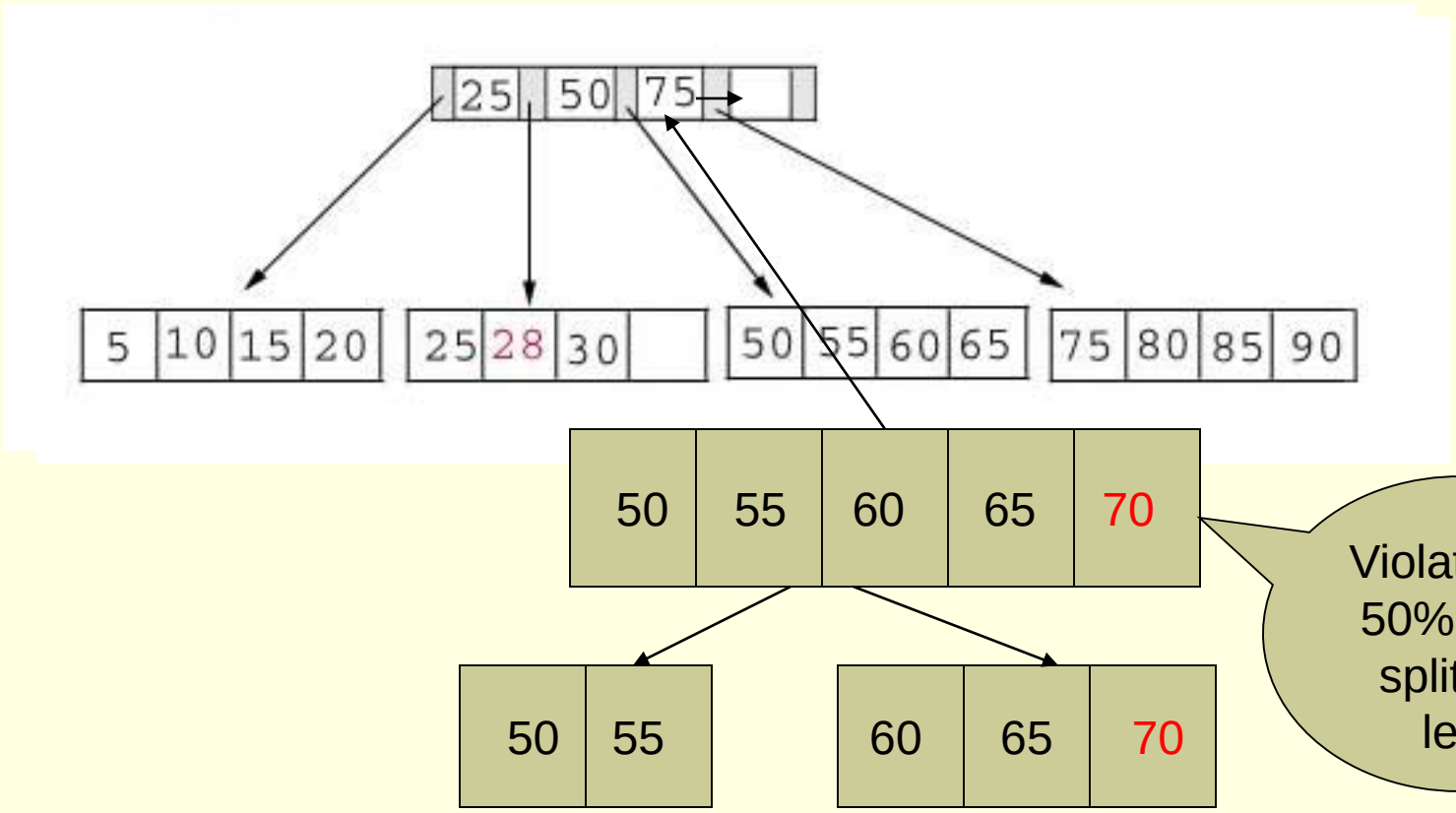
Insertion

- Example #2: insert 70 into below tree



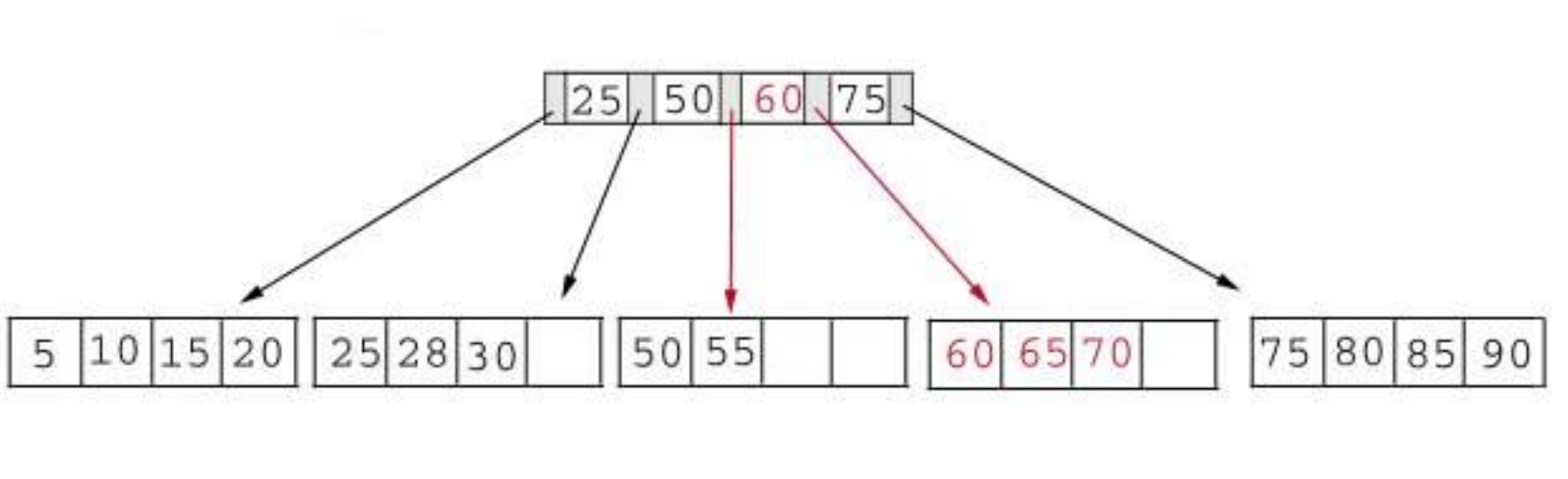
Insertion

- Process: split the tree



Insertion

- Result: chose the middle key 60, and place it in the index page between 50 and 75.



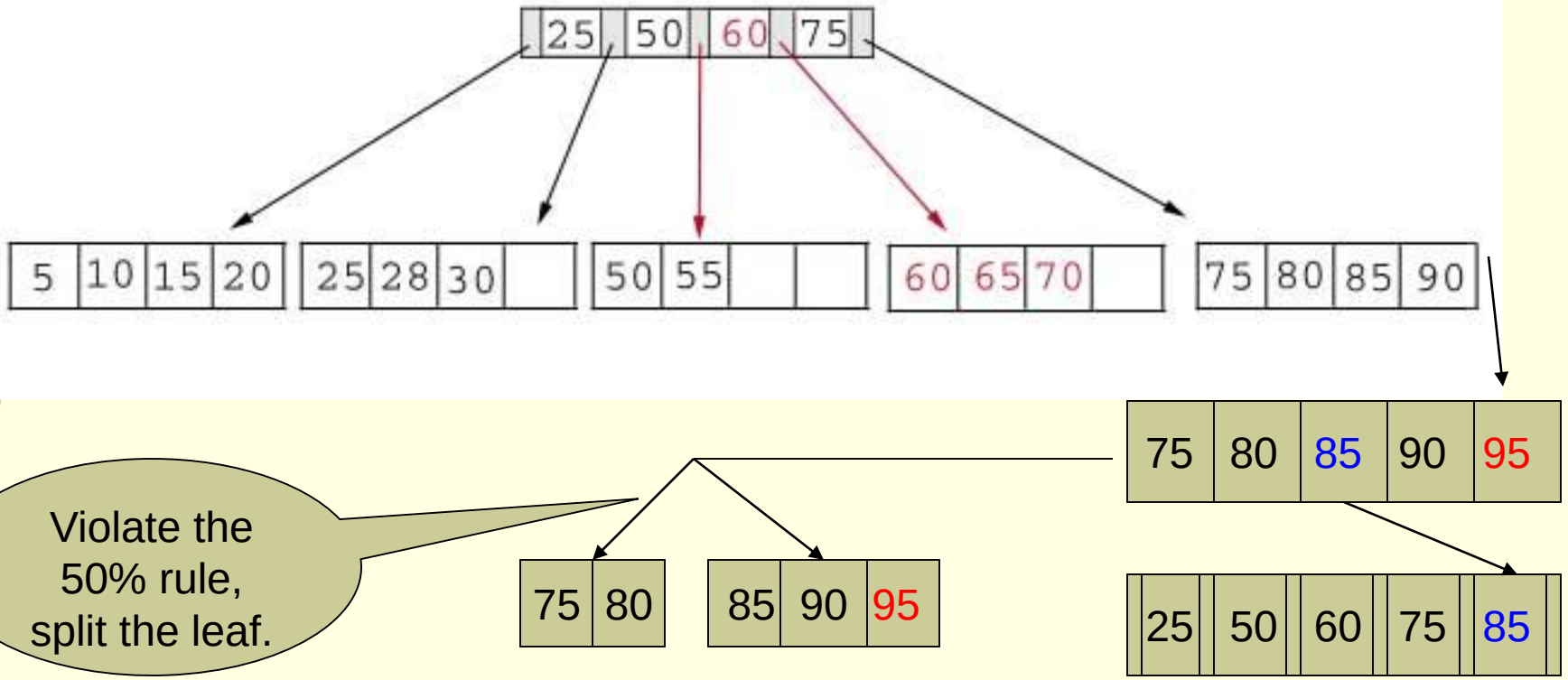
Insertion

The insert algorithm for B+ Tree

Data Page Full	Index Page Full	Action
NO	NO	Place the record in sorted position in the appropriate leaf page
YES	NO	1. Split the leaf page 2. Place Middle Key in the index page in sorted order. 3. Left leaf page contains records with keys below the middle key. 4. Right leaf page contains records with keys equal to or greater than the middle key.
YES	YES	1. Split the leaf page. 2. Records with keys < middle key go to the left leaf page. 3. Records with keys >= middle key go to the right leaf page. Split the index page. 4. Keys < middle key go to the left index page. 5. Keys > middle key go to the right index page. 6. The middle key goes to the next (higher level) index. IF the next level index page is full, continue splitting the index pages.

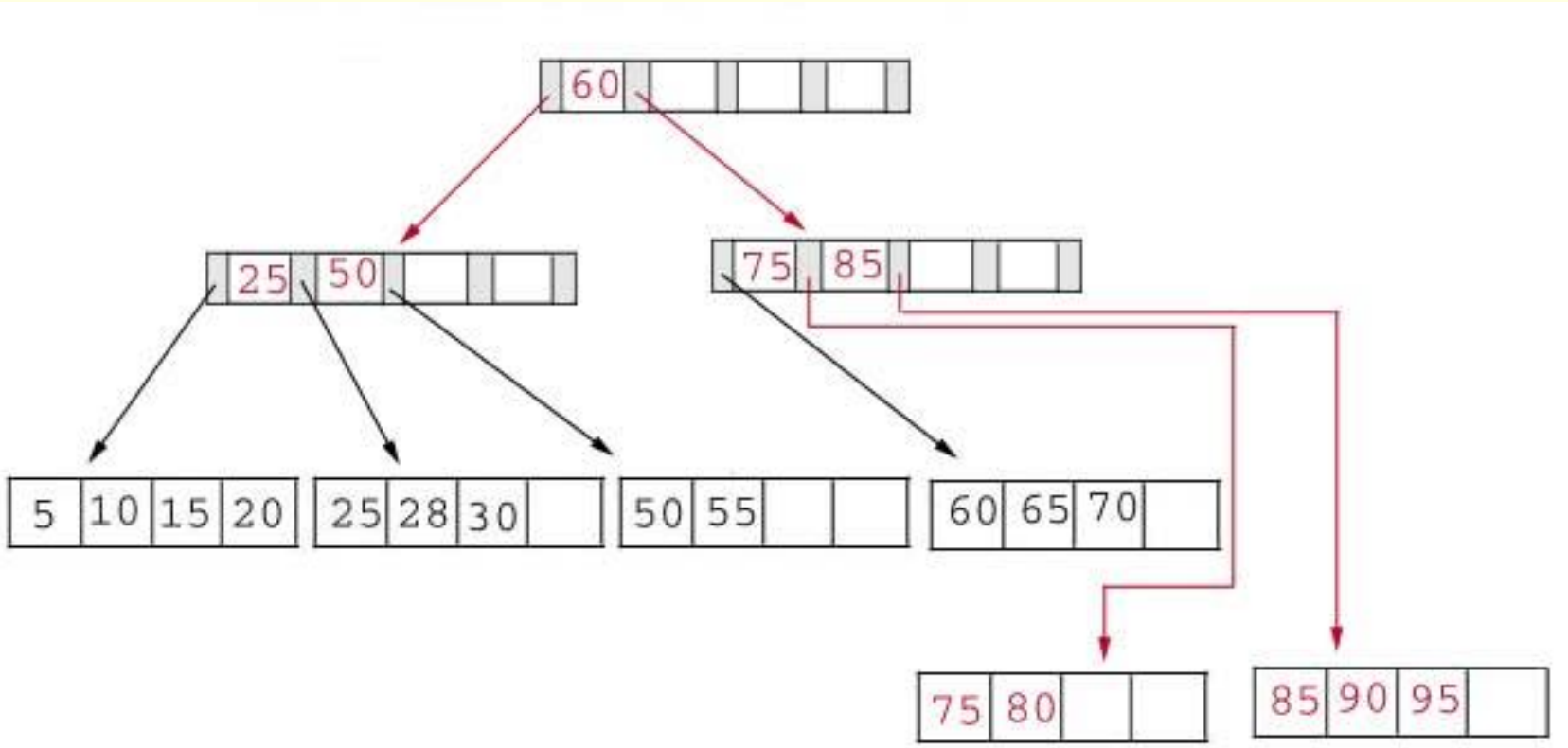
Insertion

- Exercise: add a key value 95 to the below tree.



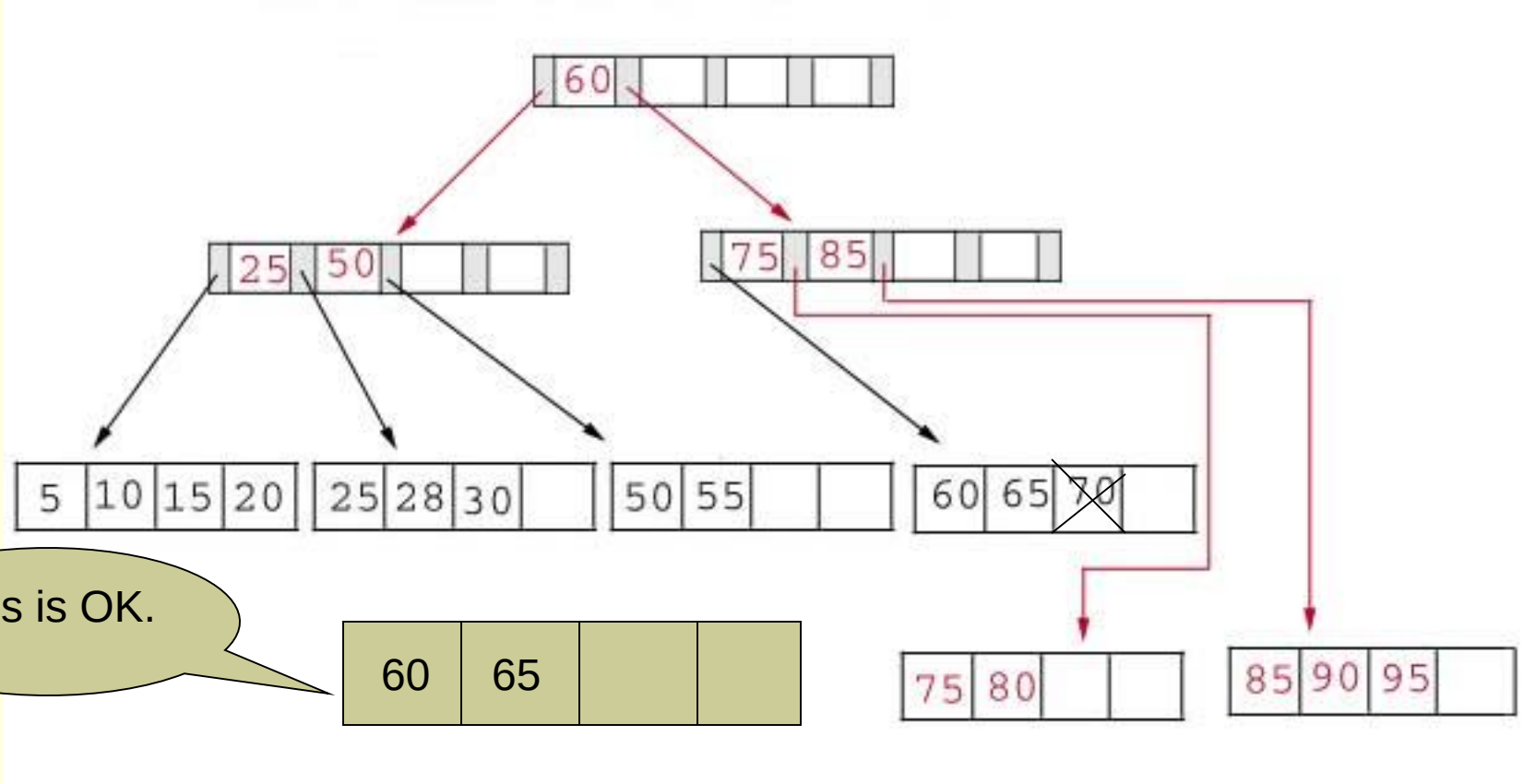
Insertion

- Result: again put the middle key 60 to the index page and rearrange the tree.



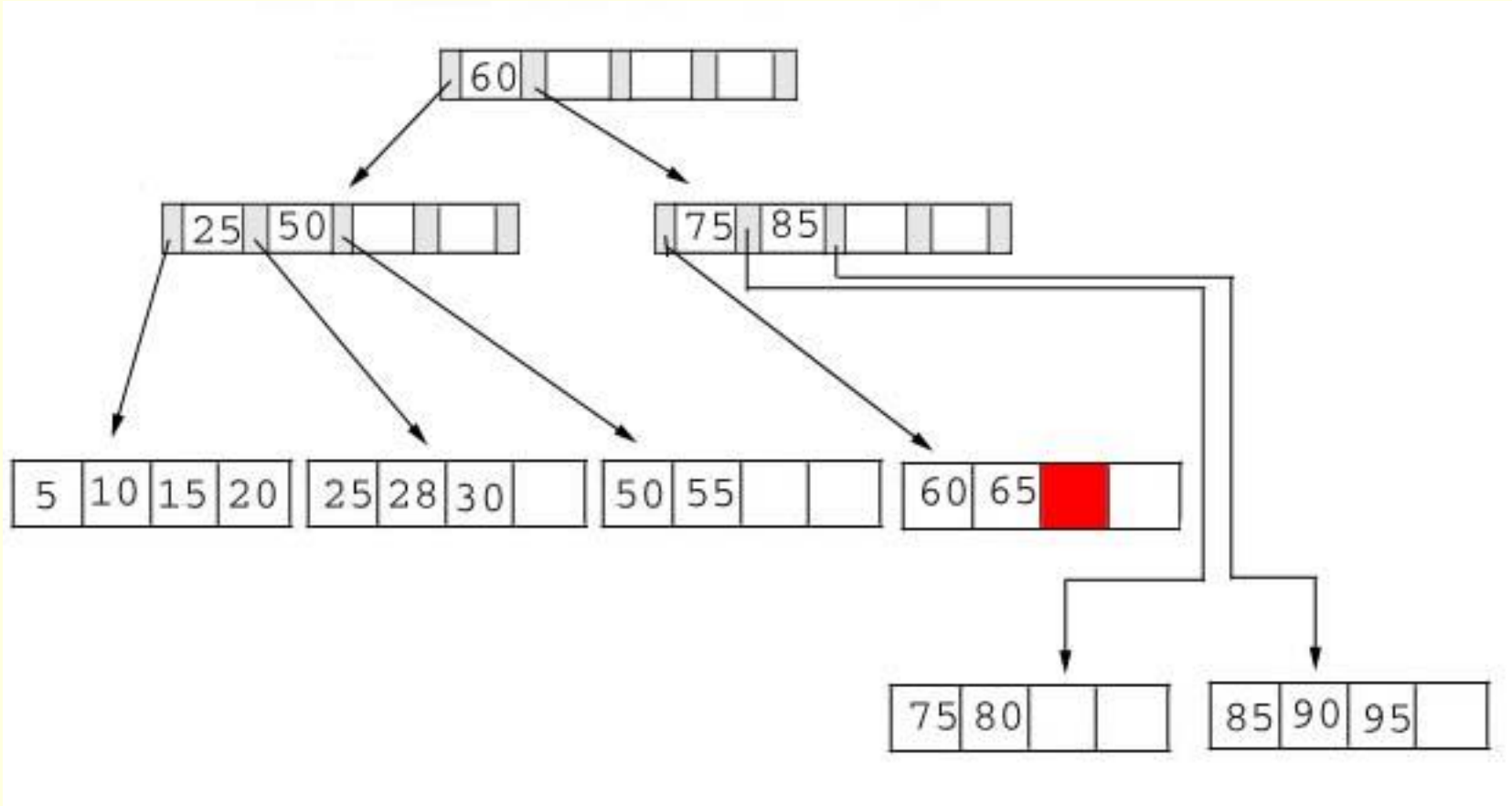
Deletion

- Same as insertion, the tree has to be rebuild if the deletion result violate the rule of B+ tree.
- Example #1: delete 70 from the tree



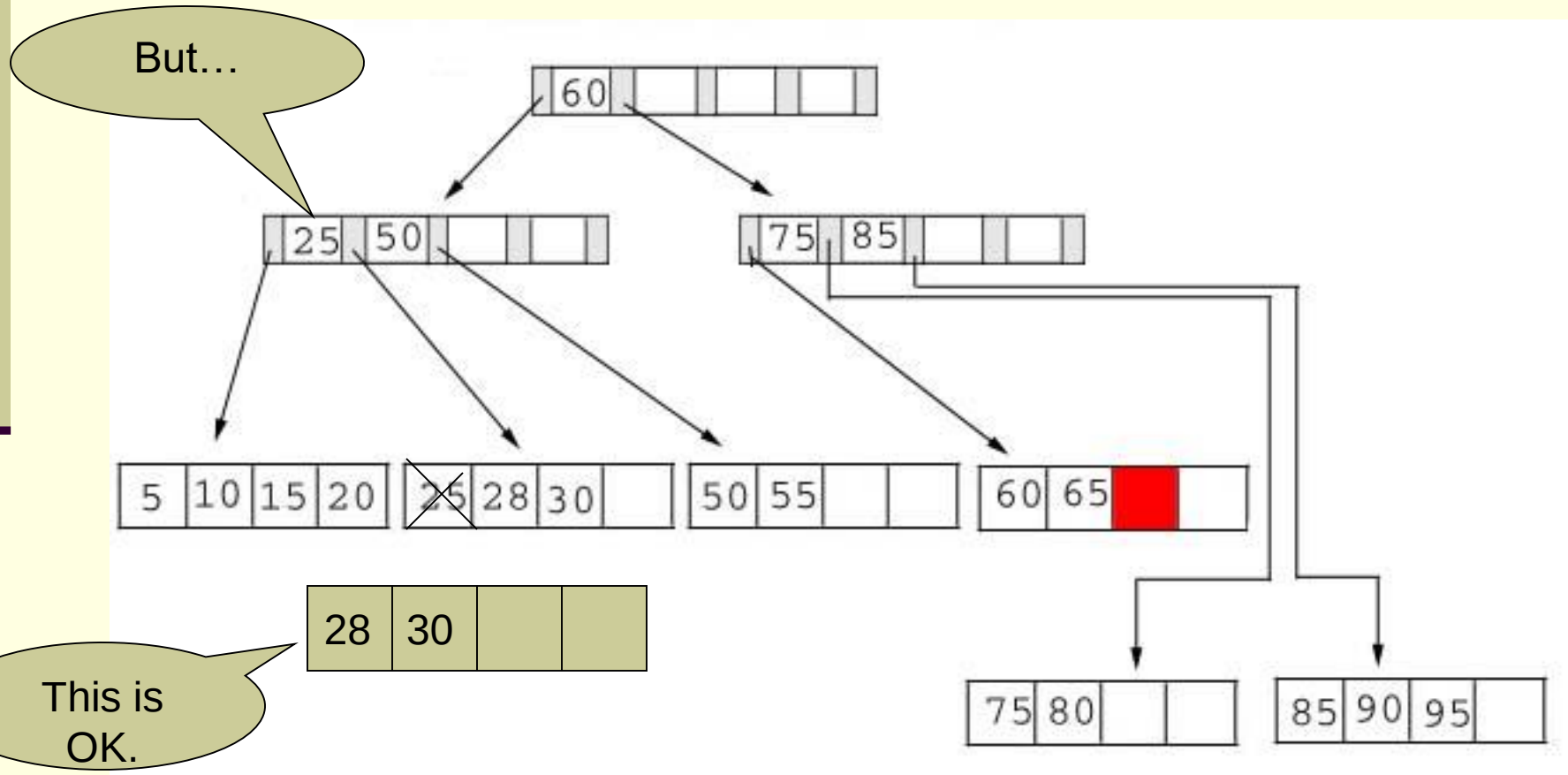
Deletion

- Result:



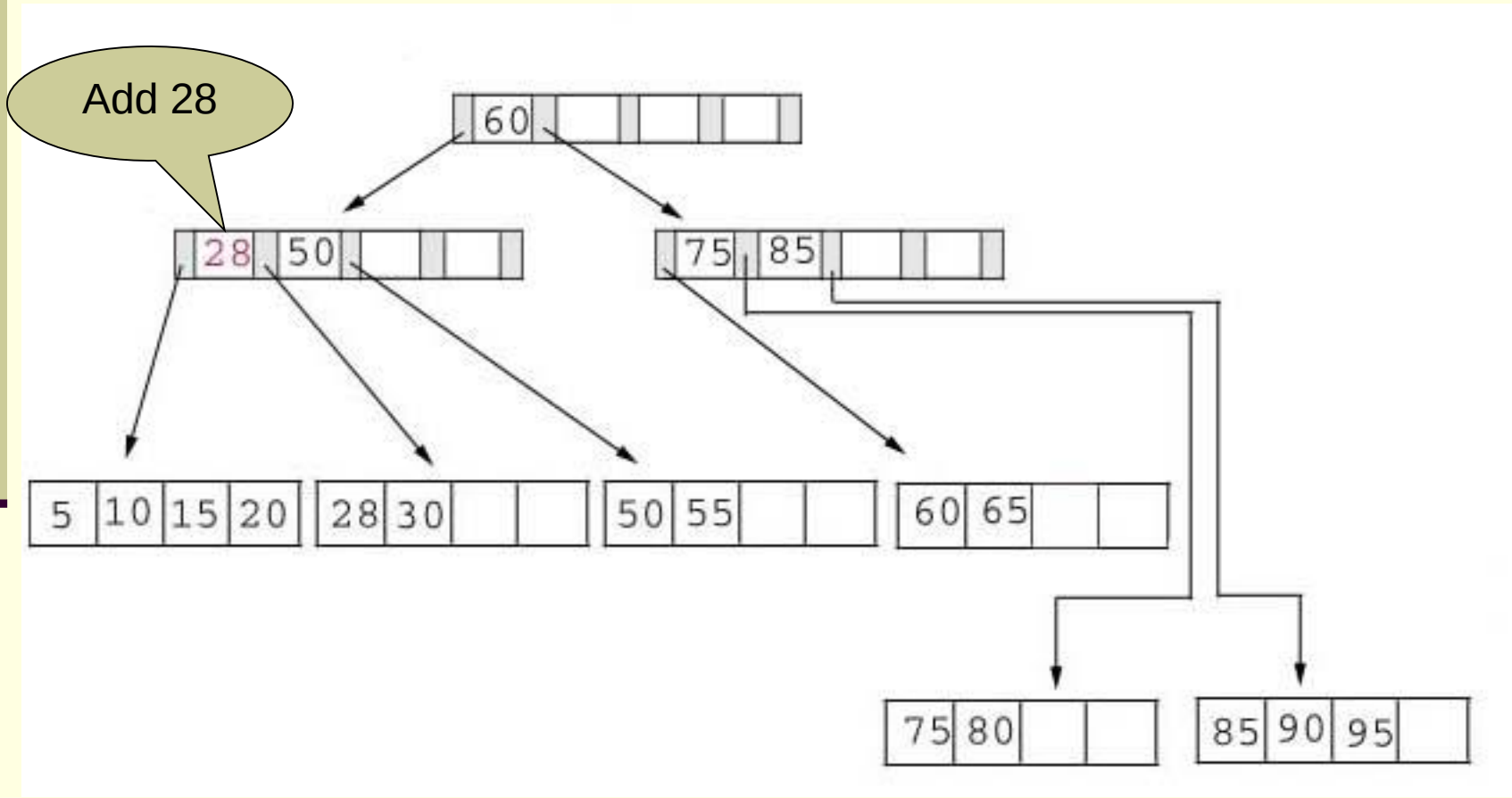
Deletion

Example #2: delete 25 from below tree, but 25 appears in the index page.



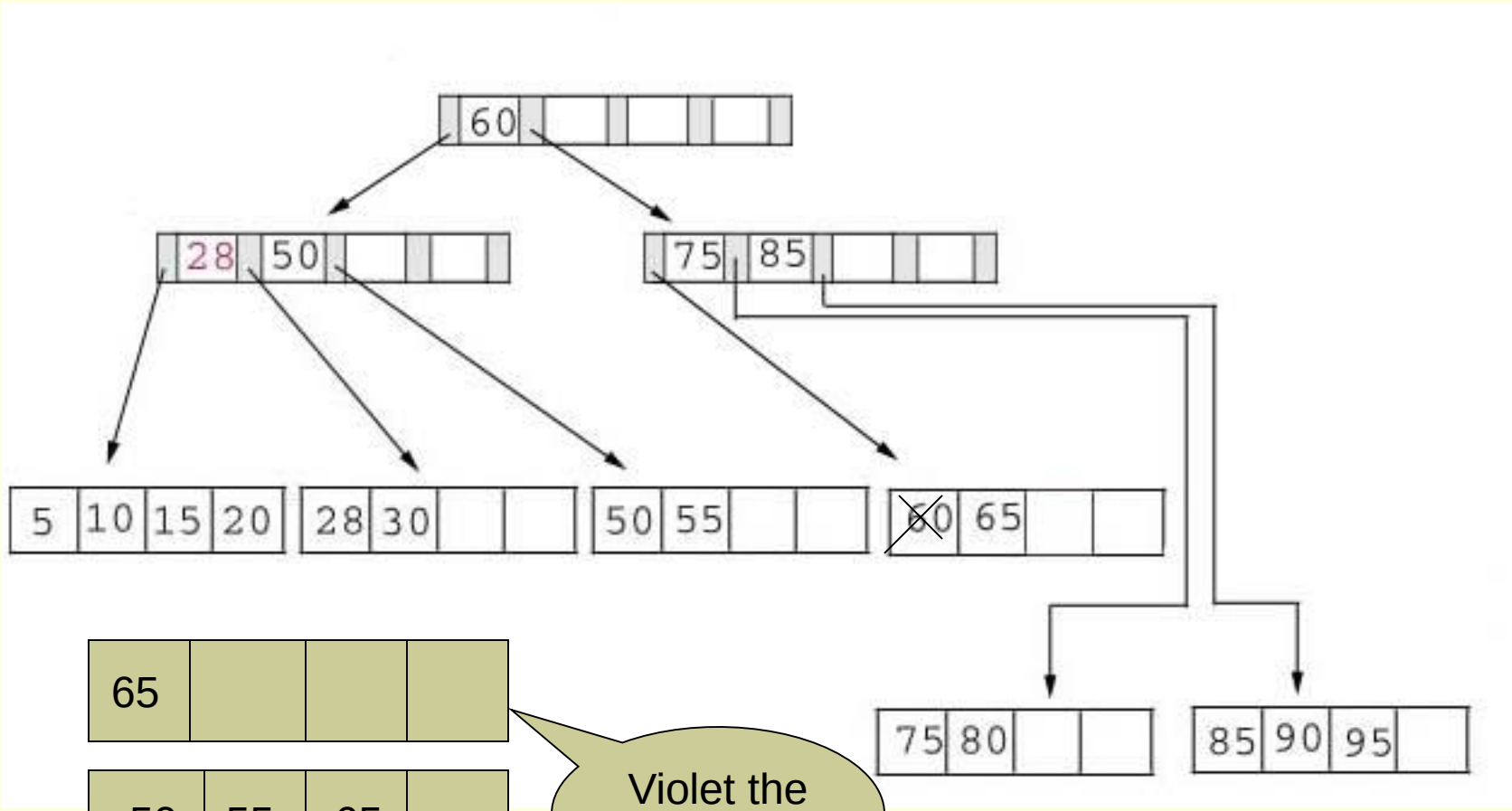
Deletion

- Result: replace 28 in the index page.



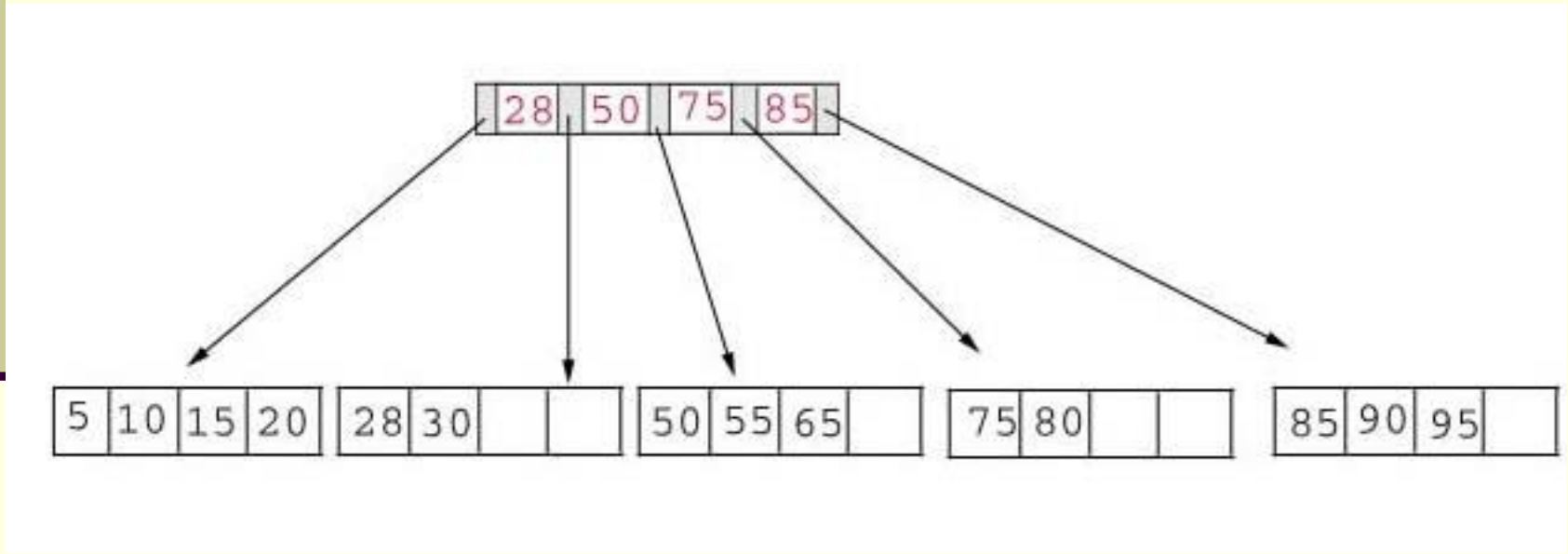
Deletion

- Example #3: delete 60 from the below tree



Deletion

- Result: delete 60 from the index page and combine the rest of index pages.



Deletion

- Delete algorithm for B+ trees

Data Page Below Fill Factor	Index Page Below Fill Factor	Action
NO	NO	Delete the record from the leaf page. Arrange keys in ascending order to fill void. If the key of the deleted record appears in the index page, use the next key to replace it.
YES	NO	Combine the leaf page and its sibling. Change the index page to reflect the change.
YES	YES	1. Combine the leaf page and its sibling. 2. Adjust the index page to reflect the change. 3. Combine the index page with its sibling. Continue combining index pages until you reach a page with the correct fill factor or you reach the root page.

Conclusion

- For a B+ Tree:
- It is easy to maintain it’s balance.
- The searching time is short than most of other types of trees.

Reference

- http://babbage.clarku.edu/~achou/cs160/B+Trees/B+Trees.htm
- www.csee.umbc.edu/~pmundur/courses/CMSC461-05/ch12.ppt