Advanced Data Structures and Algorithms

Multi-way Search Trees

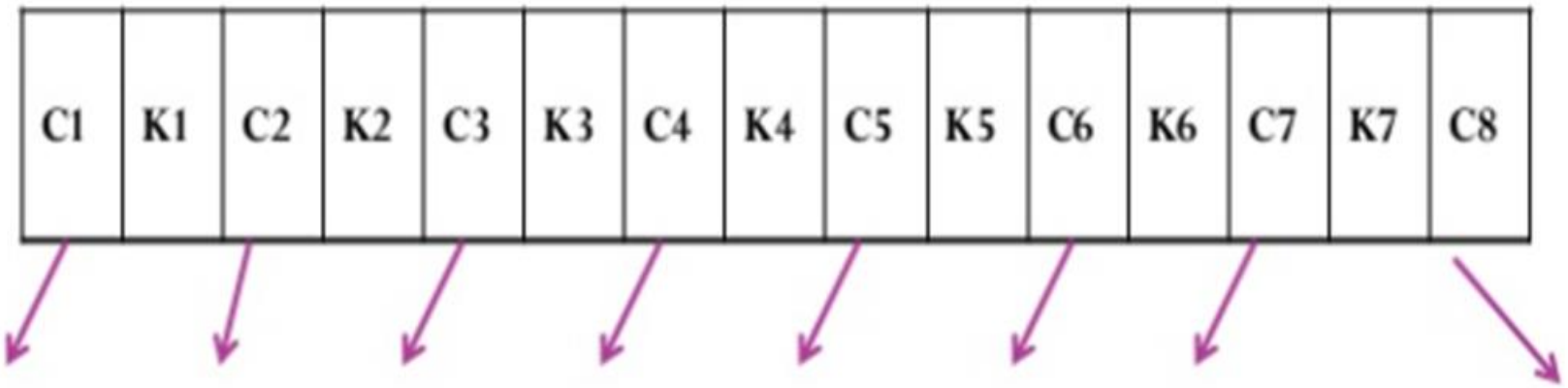
Dr G.Kalyani

Department of Information Technology

VR Siddhartha engineering College

Multi-way Tree

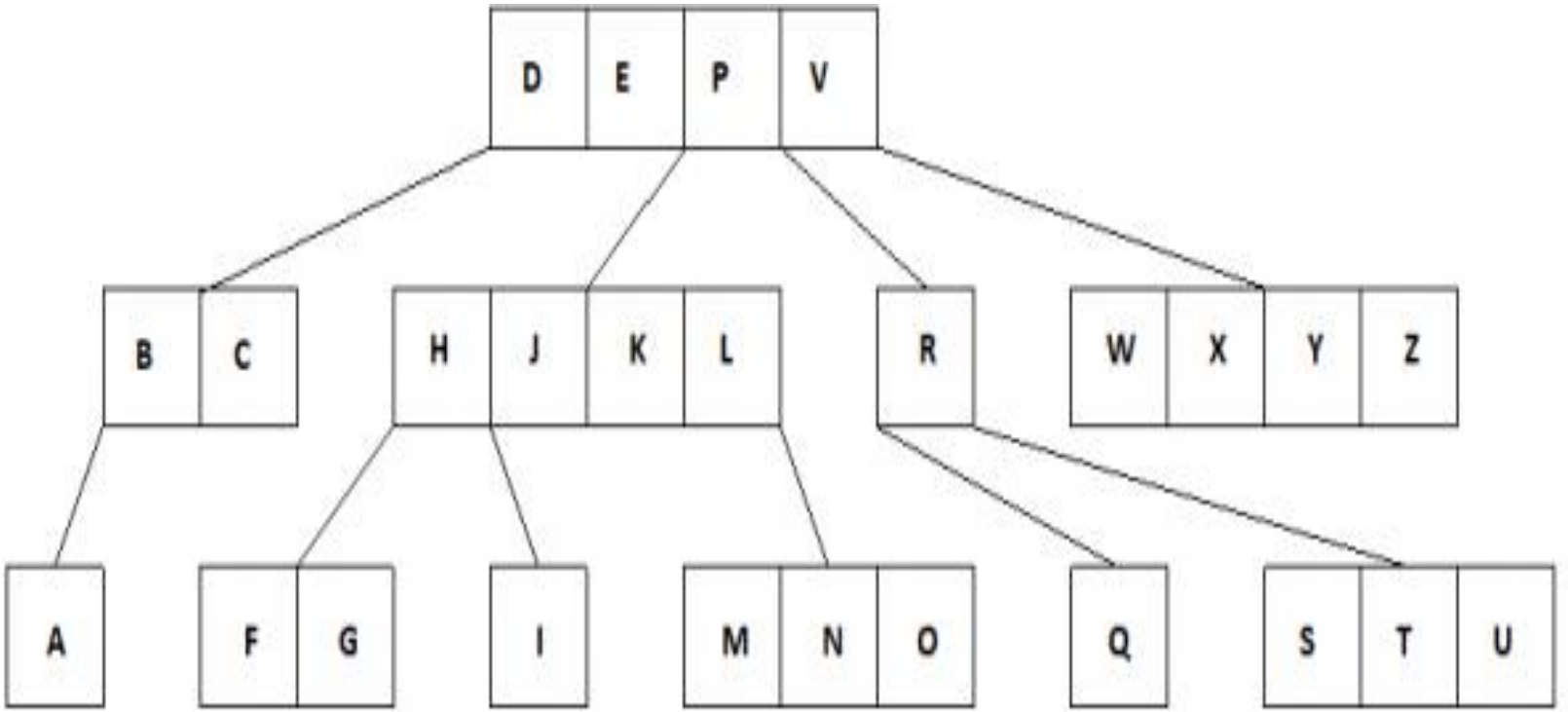
- **A multiway tree is defined as a tree that can have more than two children.**
- If a tree can have **maximum m children, then this tree is called as multiway tree of order m (or an m-way tree).**
- As with the other trees that have been studied, the nodes in an **m-way tree will be made up of (m-1) key fields and m pointers** to children.



1. This node has the capacity to hold 7 keys and 8 children.

Multi-way Tree

- Multiway tree of order 5



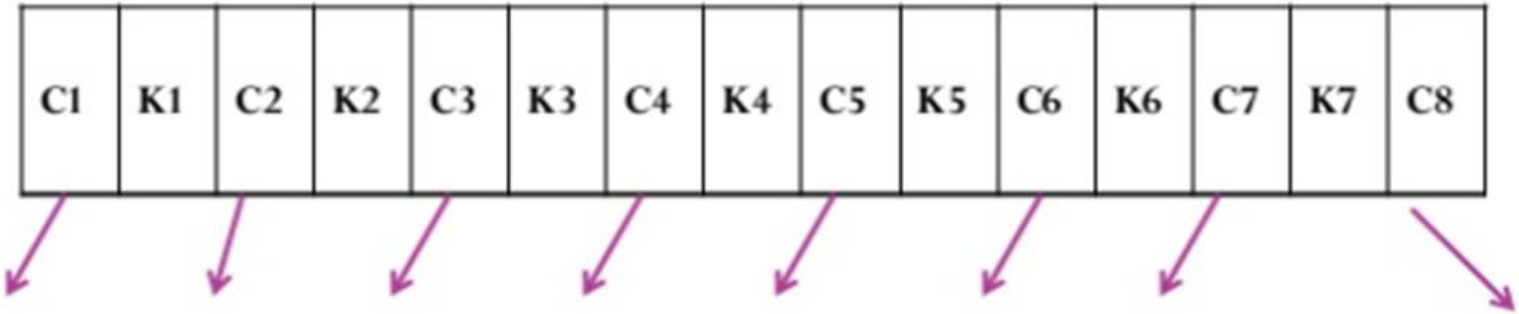
Multi-way Search Tree

- To make the processing of m-way trees easier **order will be imposed on the keys within each node, resulting in a multiway search tree of order m** (or an m-way search tree).
- By definition an **m-way search tree is a m-way tree in which following condition should be satisfied.**
  - **Each node is associated with m children and (m-1) key fields**
  - **The keys in each node are arranged in ascending order.**
  - **The subtree in the left pointer are less than the key and sub tree in the right pointer are greater than the key.**

Multi-way Search Tree node of order 8

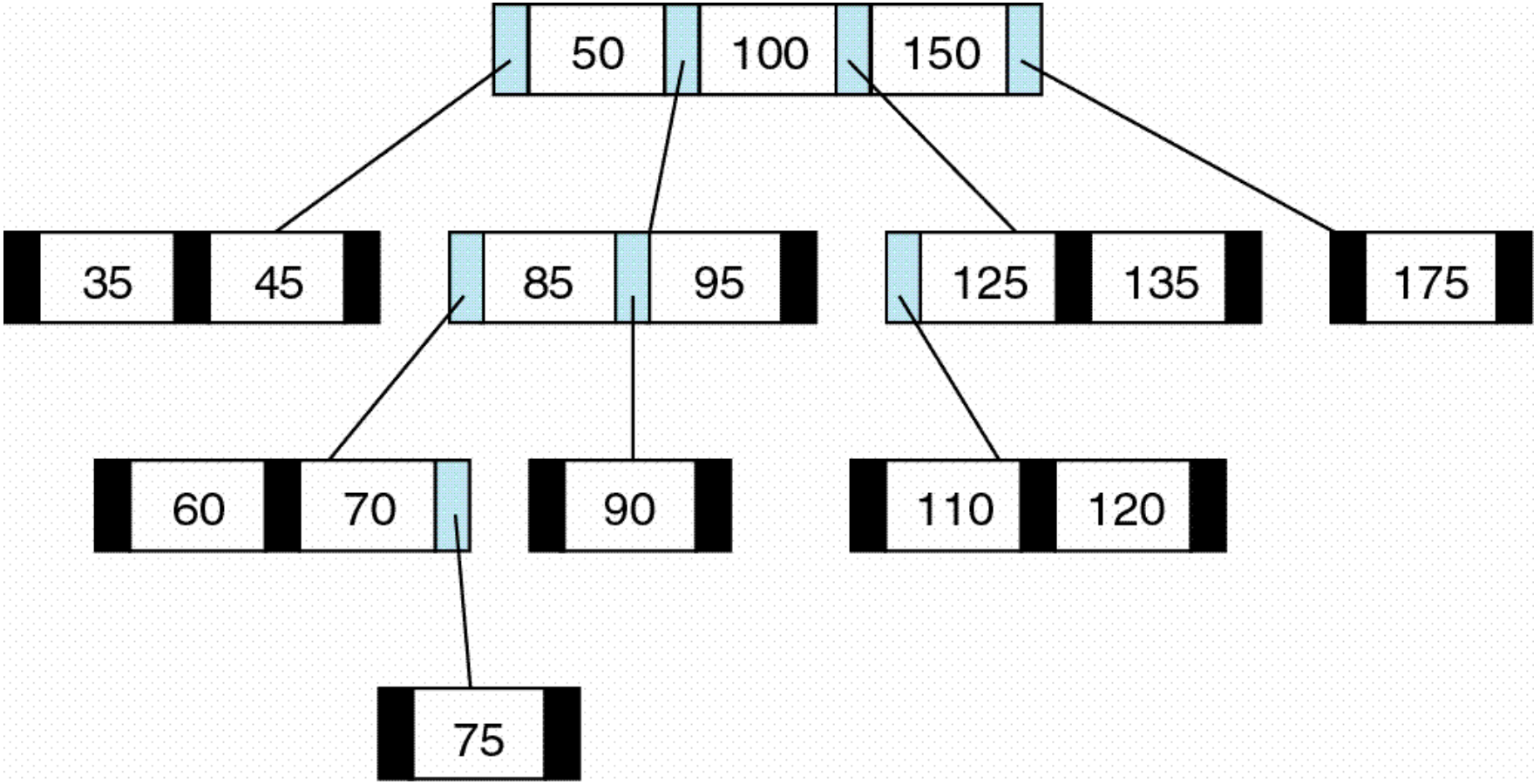
Example

Consider a node of m-way search tree of order 8



1. This node has the capacity to hold 7 keys and 8 children.
2. K1 < K2 < K3 < K4 < K5 < K6 < K7
3. The key K1 is greater then all the keys in subtree pointed to by C1 and less than all the keys in subtree pointed to by pointer C2. Similarly this relation holds true for other keys also.
4. Keys(C1) < K1< Keys(C2) < K2 < Keys(C3) < K3 < Keys(C4)< K4 ….

Multi-way Search Tree of order 4

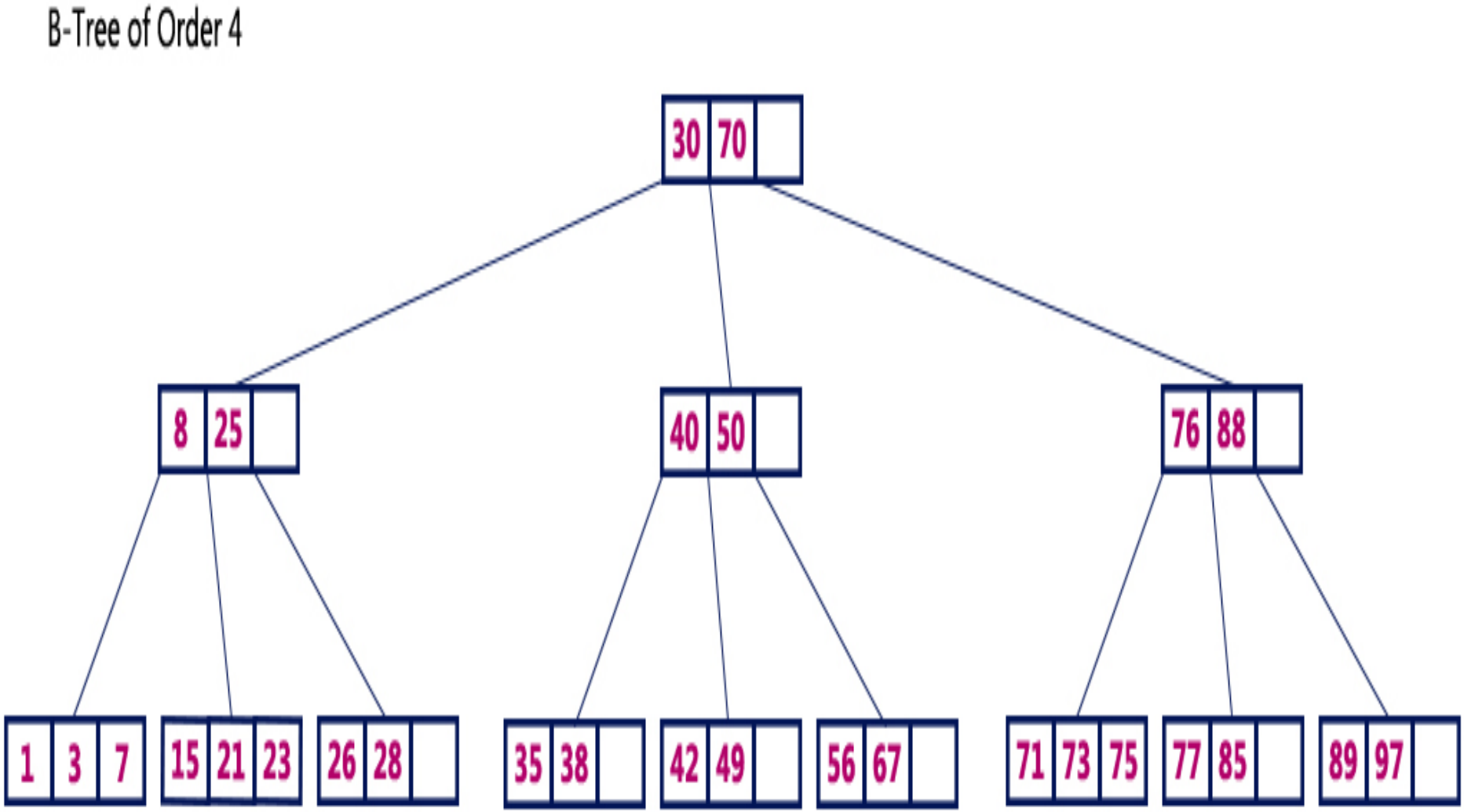


B-Trees

- B-Tree is also known as **Height Balanced m-way Search Tree**

- A **B-tree of order m can be defined as an m-way search tree which is either empty or satisfies the following properties:-**
  - **All leaf nodes are at the same level.**
  - **All non-leaf nodes (except root node) should have at least $\mathbf{m/2}$ children.**
  - **All nodes (except root node) should have at least $[(\mathbf{m/2}) - \mathbf{1}]$ keys.**
  - **If the root node is a leaf node, then it will have at least one key. If the root node is a non-leaf node, then it will have at least 2 children and one key.**
  - **A non-leaf node with $n - 1$ keys values should have n non NULL children.**

B-Tree of order 4



Operations on B-Tree

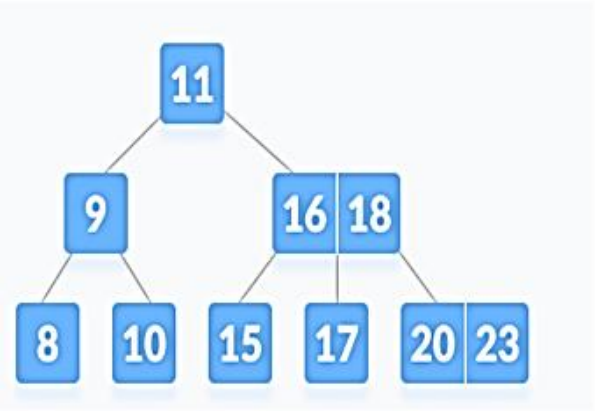
- **Searching for an element**
- **Insertion of an element**
- **Deletion of an element**
- **Traversing the tree**

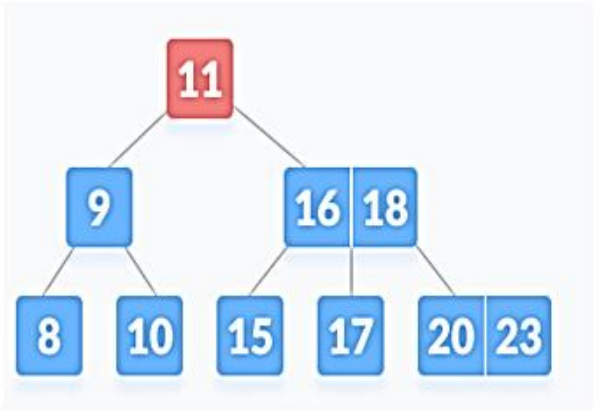
Searching in B-TREE

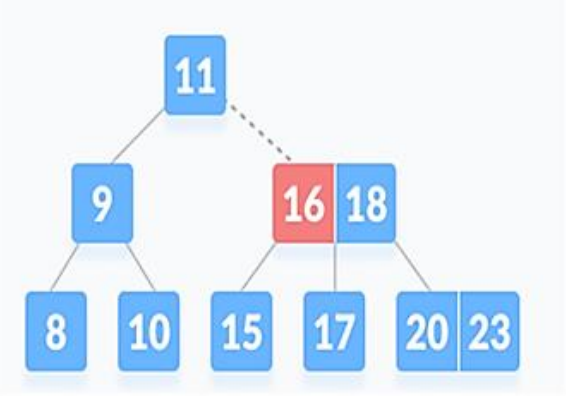
- If we want to search for a value say **X** in an M-way search tree and currently we are at a root node that contains key values from **Y1, Y2, Y3,.....,Yk**. Then in total 4 cases are possible to deal with this scenario, these are:
  - If **X < Y1**, then we need to recursively traverse the left subtree of **Y1**.
  - If **X > Yk**, then we need to recursively traverse the right subtree of **Yk**.
  - If **X = Yi**, for some **i**, then we are done, and can return.
  - Last case is that when for some **i** we have **Yi < X < Y(i+1)**, then in this case we need to recursively traverse the subtree that is present in between **Yi** and **Y(i+1)**.

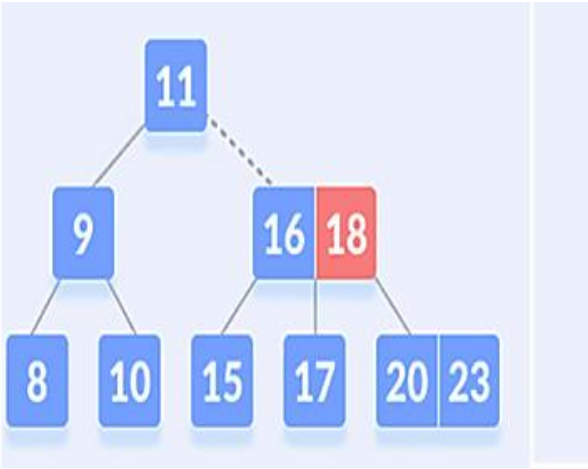
Example search in B-Tree

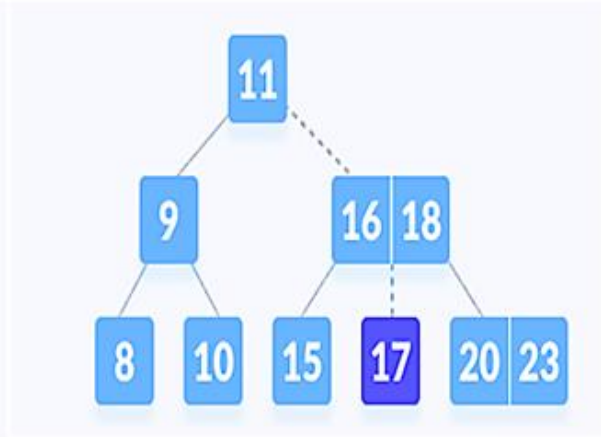
Search for 17 in B Tree of order 3





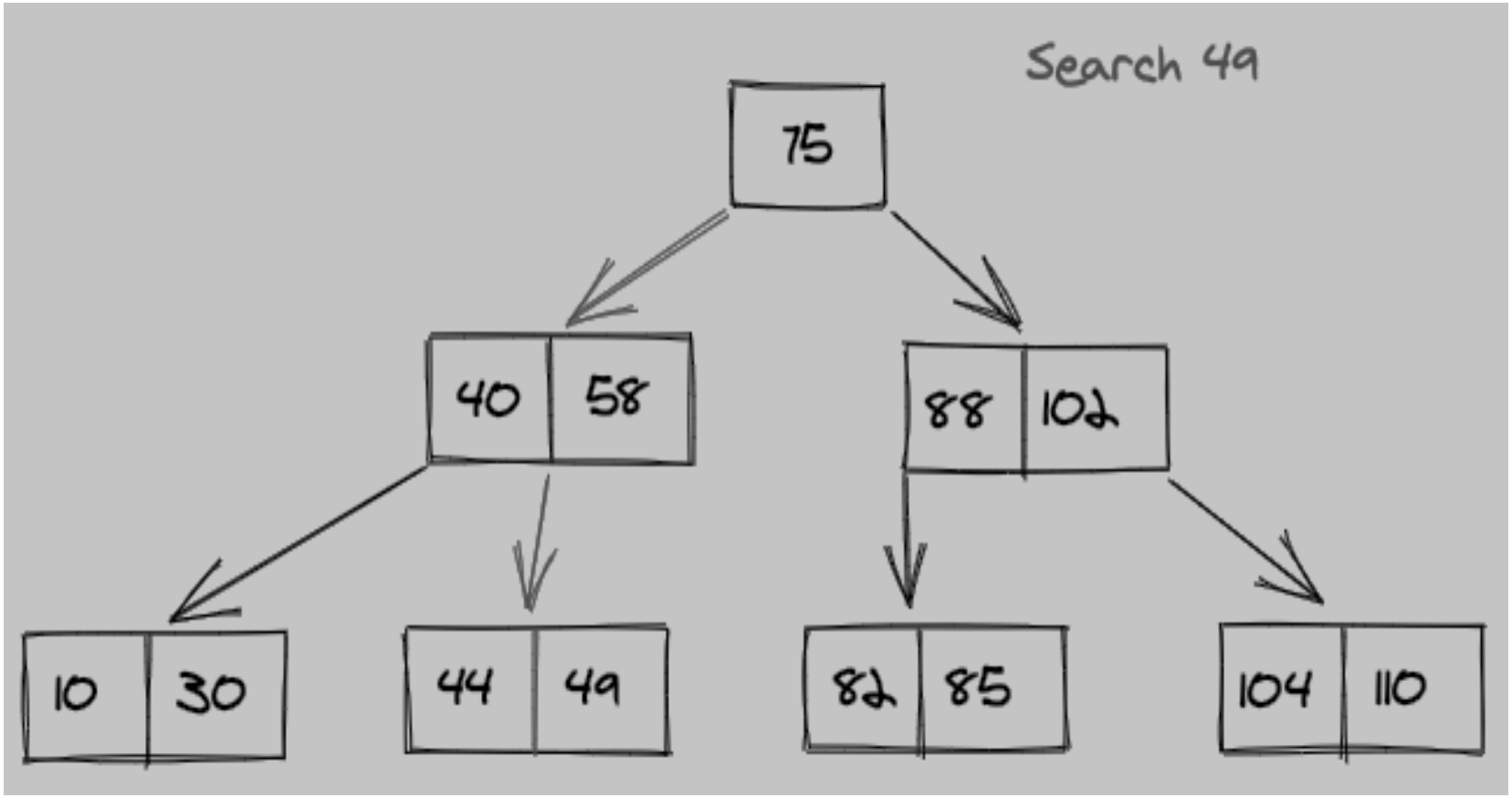






Example search in B-Tree

- Compare item **49 with root node 75**. Since **49 < 75** hence, move to its left sub-tree.
- Since, **40<49<58**, traverse right sub-tree of 40.
- **49>44**, move to right. Compare **49**.
- We have found 49, hence returning.

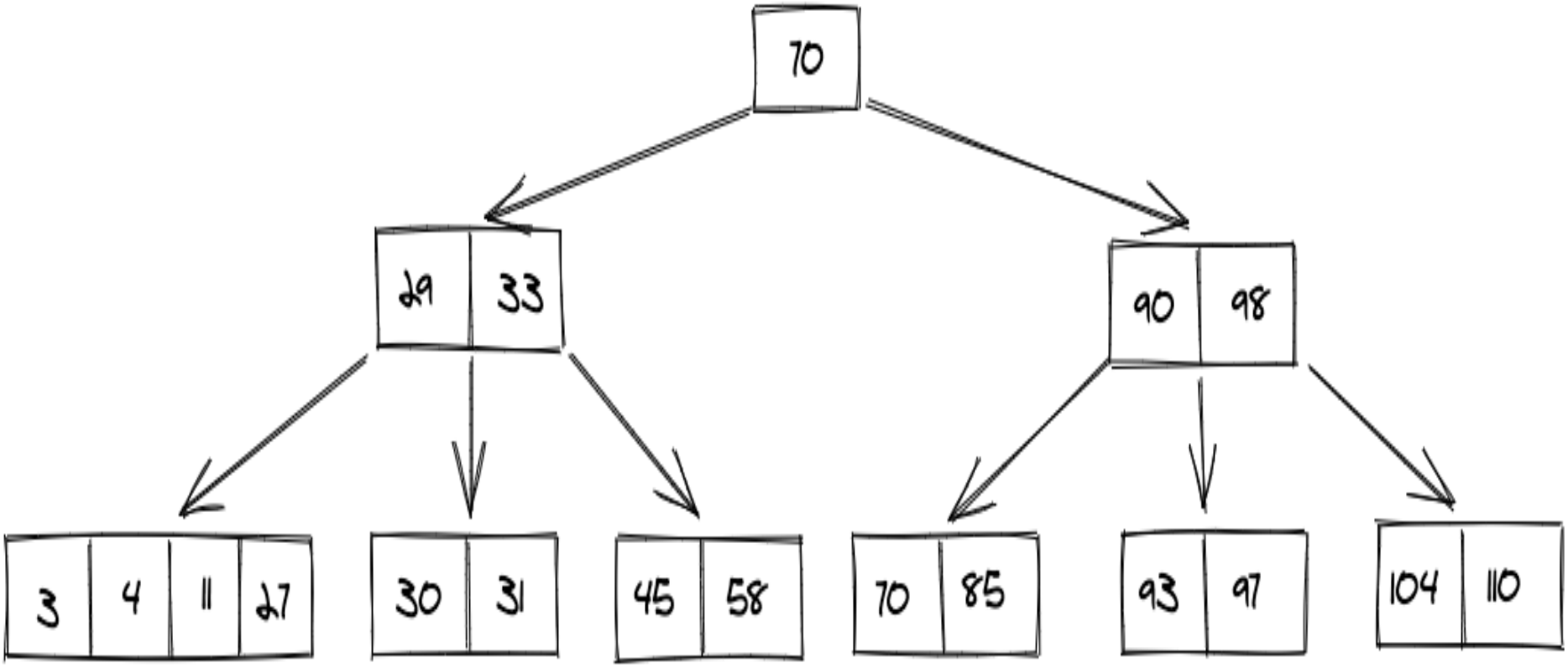


Insertion in B-Tree

- Inserting in a B-Tree is done at the leaf node level. We follow the given steps to make sure that after the insertion the B tree is valid, these are:
- First, we traverse the B tree to find the appropriate node where the inserted key will fit.
- If that node contains less than **M-1** keys, then we insert the key in an increasing order.
- If that node contains exactly **M-1** keys, then apply the following steps:
  - Insert the new element in increasing order,
  - split the nodes into two nodes through the median, push the median element up to its parent node, and
  - finally if the parent node also contains **M-1 keys**, then we need to repeat these steps.

Insertion in B-Tree

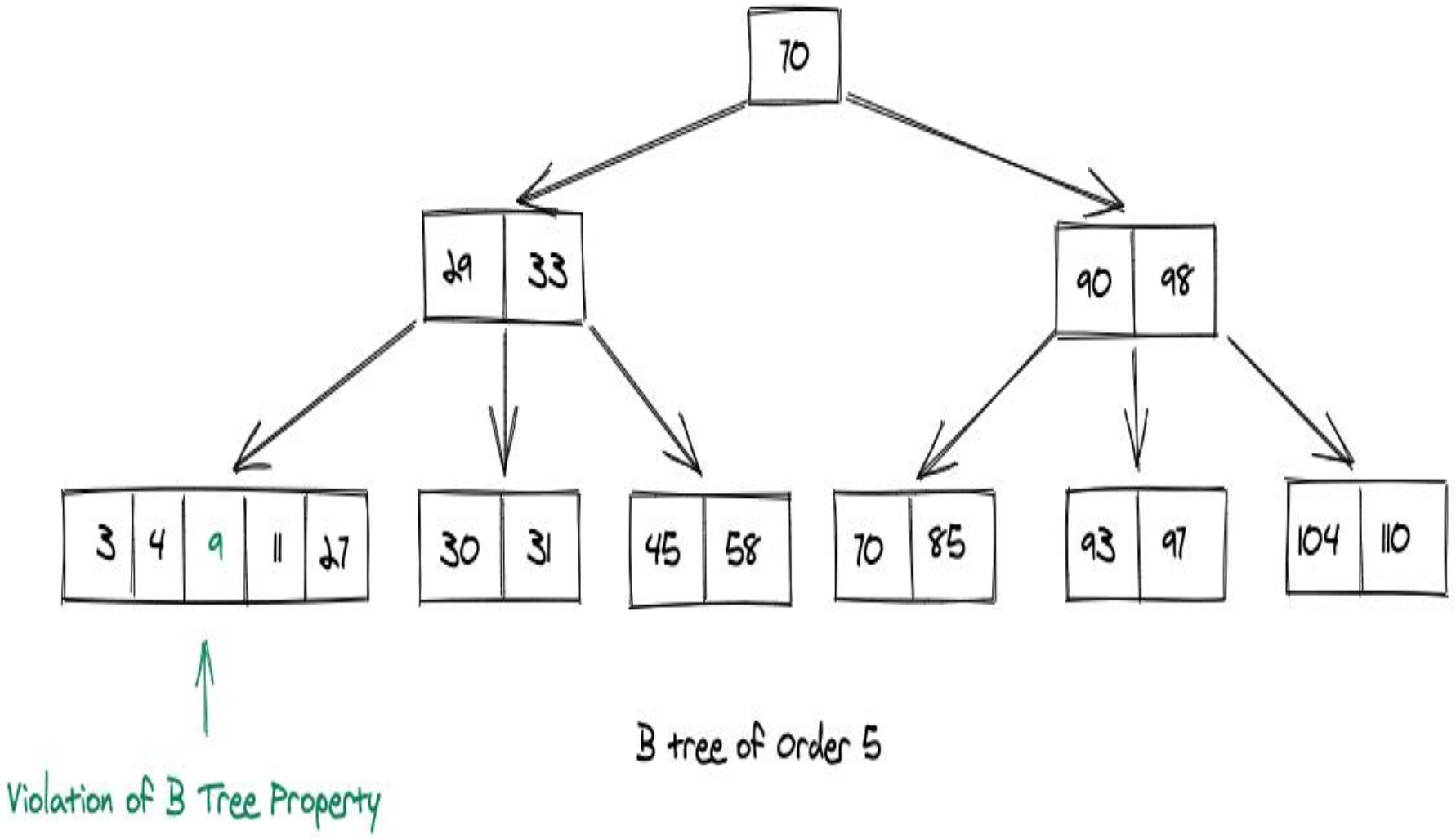
Insert 9 in to the following B-Tree of order 5:



B tree of order 5

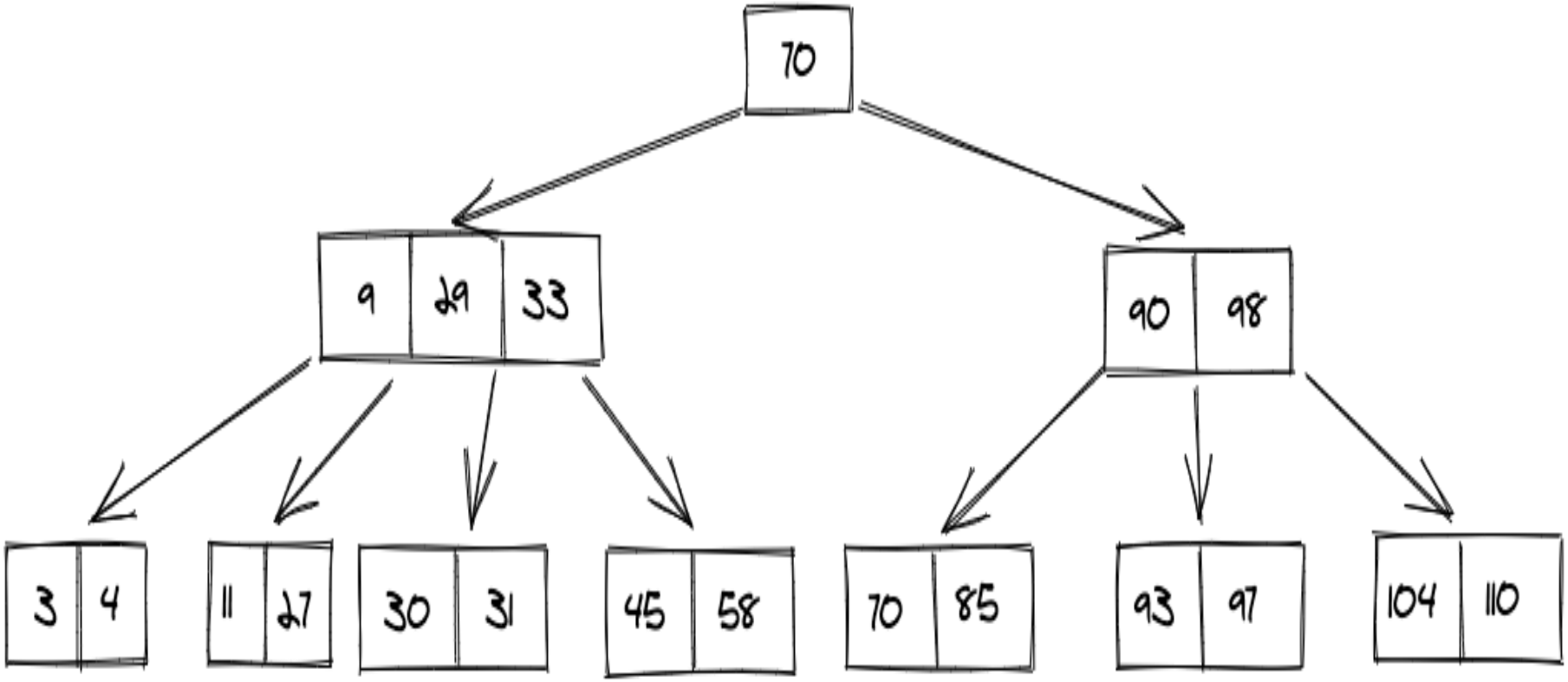
Insertion in B-Tree

Insert 9 in to the following B-Tree of order 5:



Insertion in B-Tree

Since, a violation occurred, we need to push the median node to the parent node, and then split the node in two parts, hence the final look of B tree is:

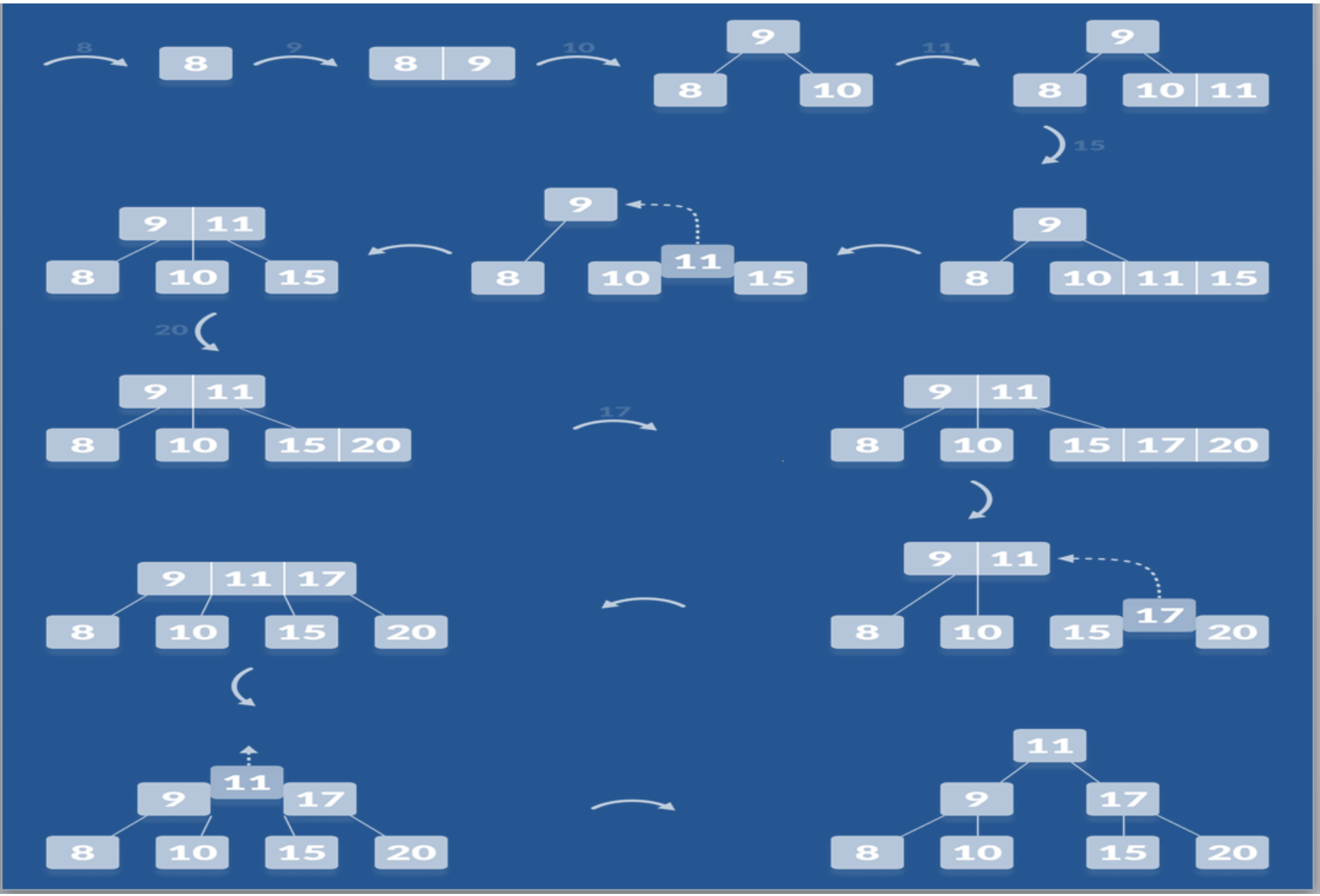


B tree of order 5

Creating a B-Tree of order 3

- **Create a B-Tree of order 3 with the elements : 8,9,10,11,15,20,17,25,7,65,48,31,2,19,23,14,6**

Creating a B-Tree of order 3



In Class Exercise1

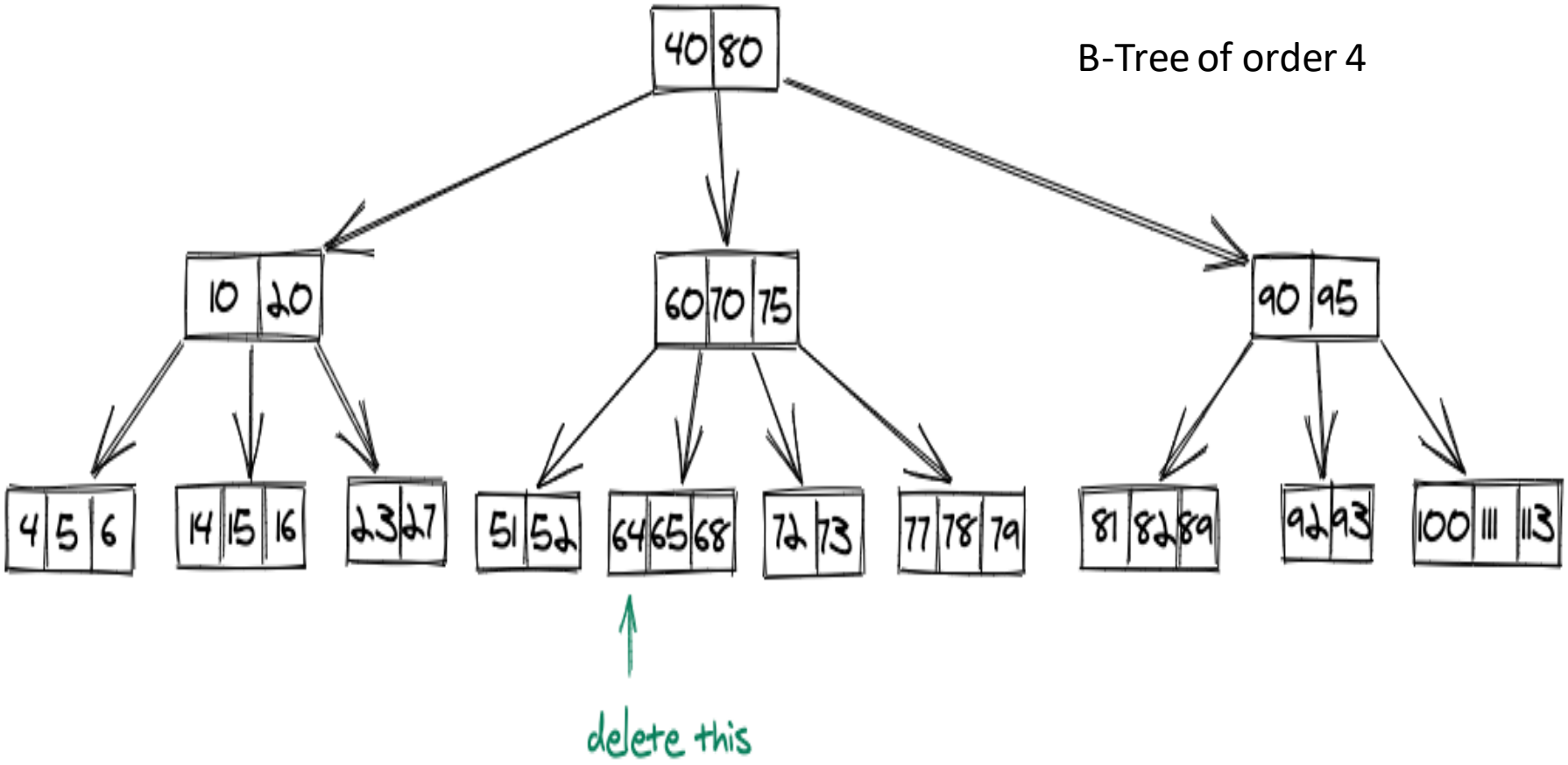
- Q: Construct an B-Tree of order 4 with the following values.
  - **12,24,8,9,56,78,41,5,6,1,4,14,89,2,28,19,32,65,21, 17,25.**

B-Tree Deletion

- Deletion of a key in a B-Tree includes two cases, these are:
  - **Deletion of key from a leaf node**
  - **Deletion of a key from an Internal node**

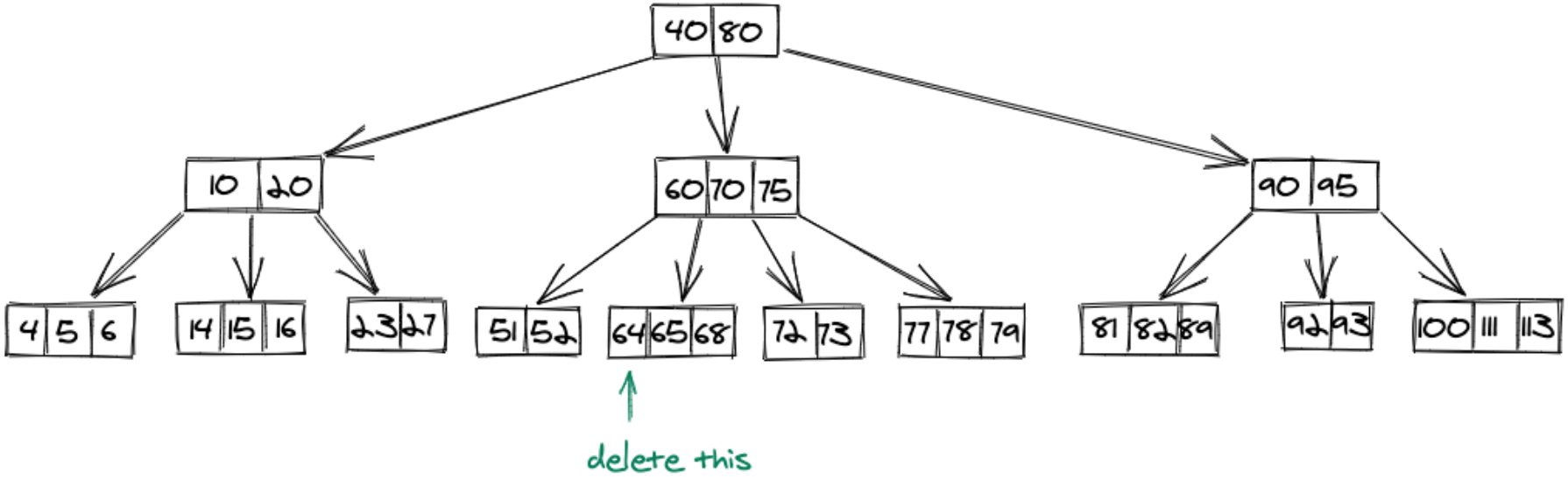
B-Tree Deletion-Leaf Node

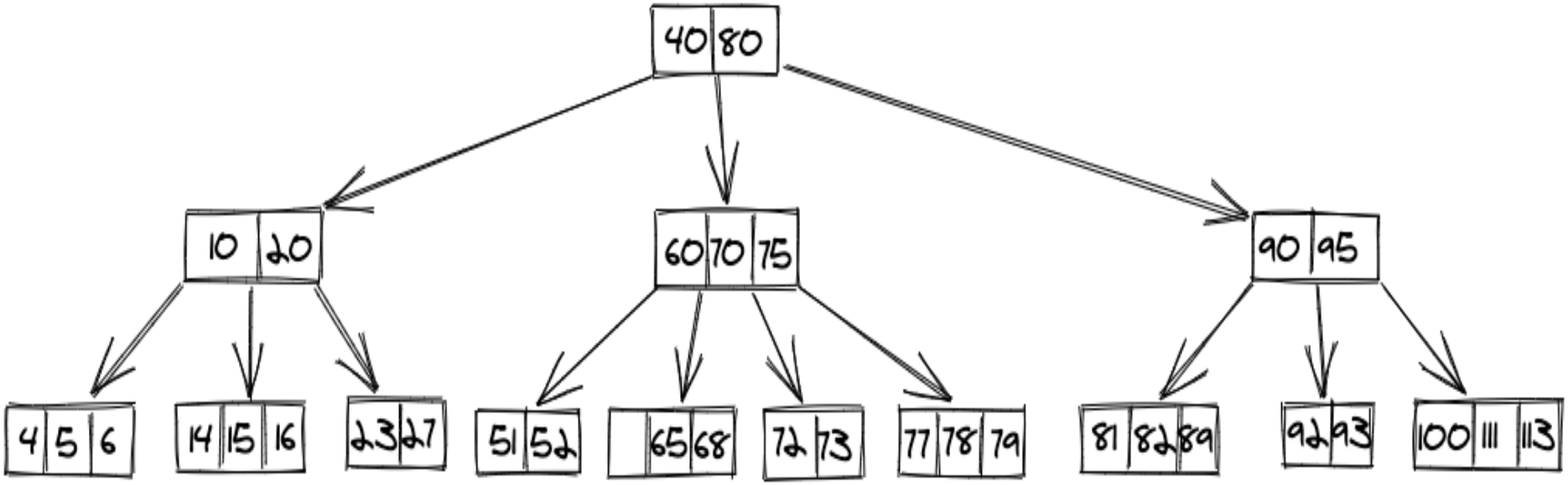
- If we want to delete a key that is present in a leaf node of a B tree, then we have two cases possible, these are:
  - **CASE-1:** If the node that contains the key that we want to delete, in turn **contains more than the minimum number of keys required** for the valid B tree, then we can simply delete that key.



B-Tree of order 4

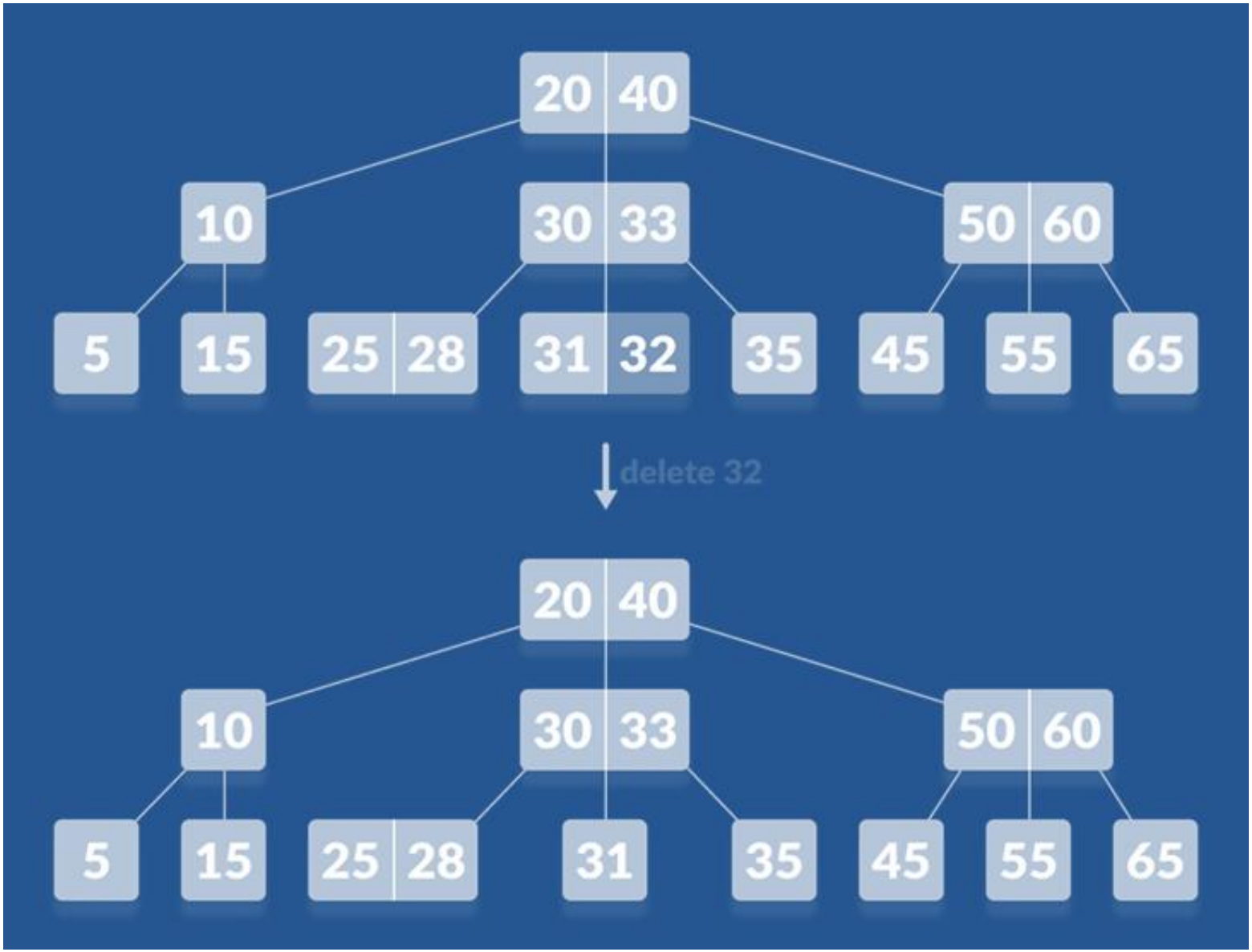
B-Tree Deletion-Leaf Node





B-Tree Deletion-Leaf Node

B-Tree of order 3



B-Tree Deletion-leaf node

- **CASE 2:** If the node that contains the key that we want to delete, in turn **not contains the minimum number of keys required** for the valid B tree, then three options are possible:

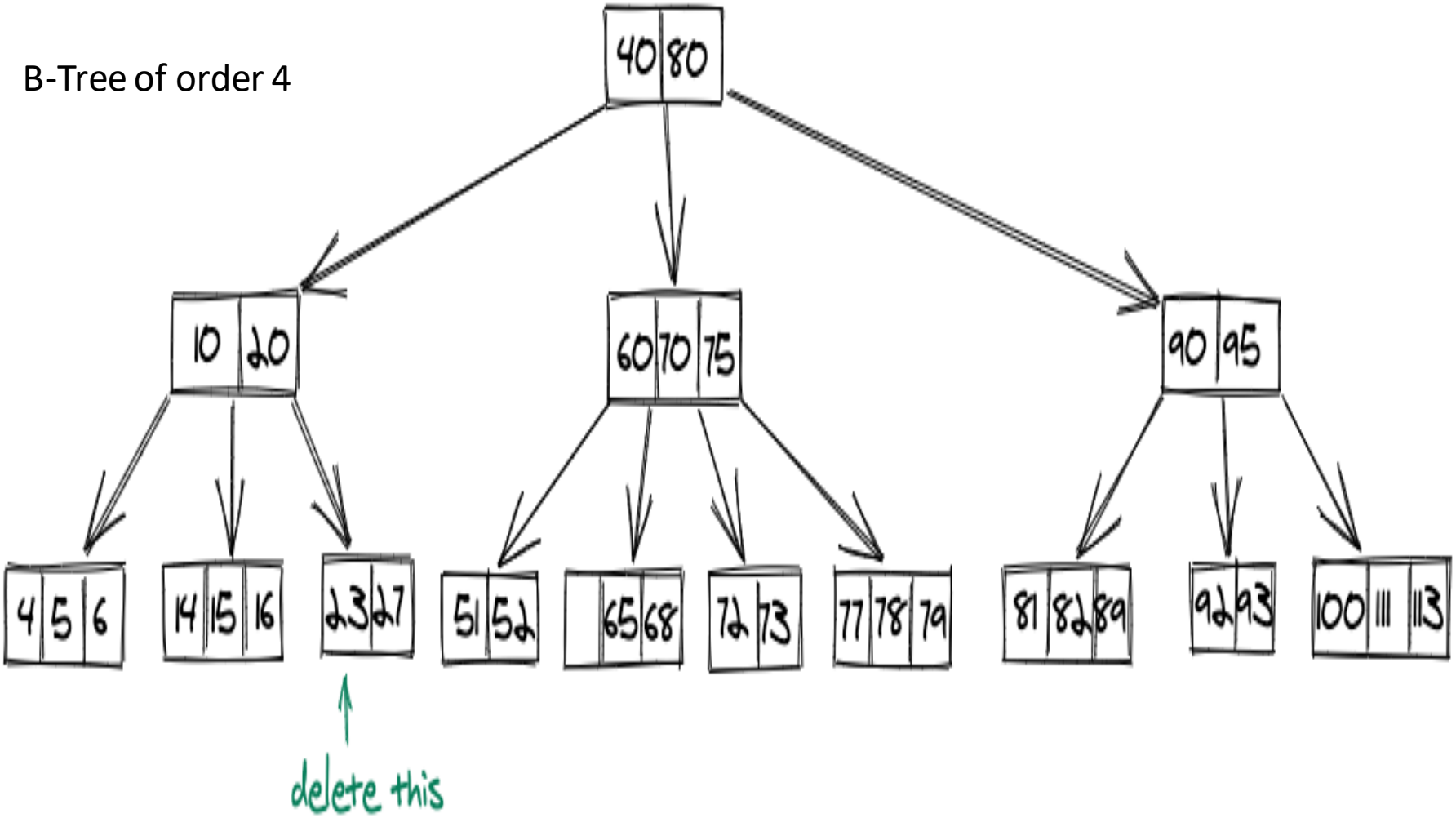
  - **Option-1: borrow a key from the left sibling.** The process is that we move the highest value key from the left sibling to the parent, and then the highest value parent key to the node from which we just deleted our key.

  - **Option-2: borrow a key from the right sibling.** The process is that we move the lowest value key from the right sibling to the parent node, and then the highest value parent key to the node from which we just deleted our key.

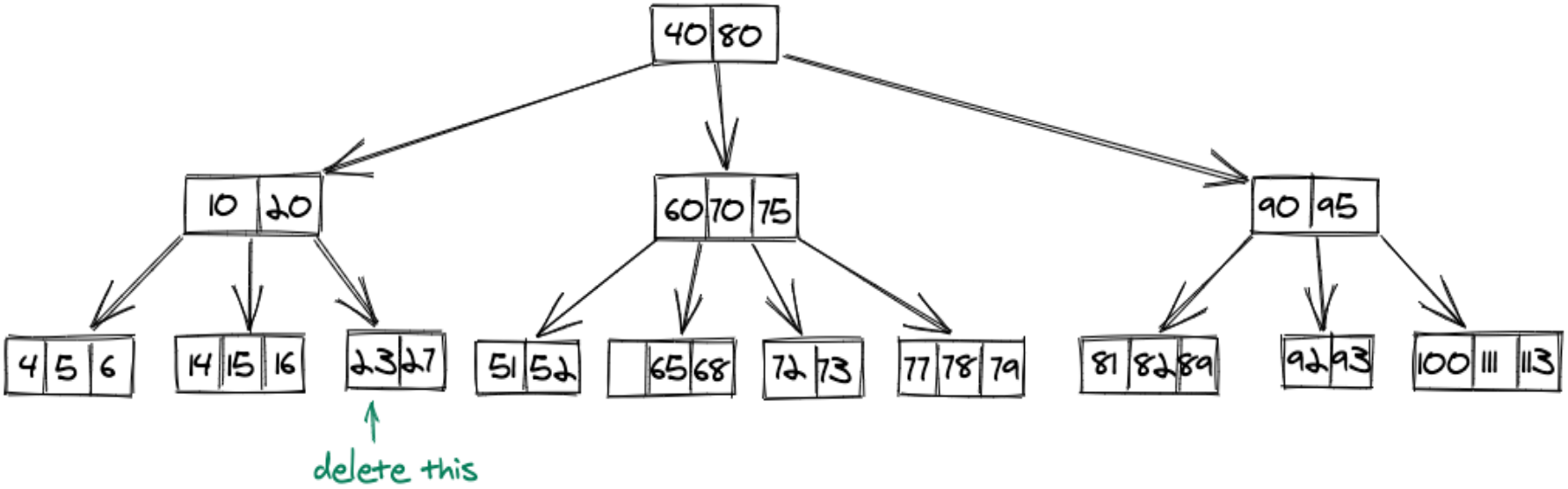
  - **Option-3:** Last case would be that neither the left sibling or the right sibling are in a state to give the current node any value, so in this step we will do a merge with either one of them, and the merge will also include a key from the parent, and then we can delete that key from the node.

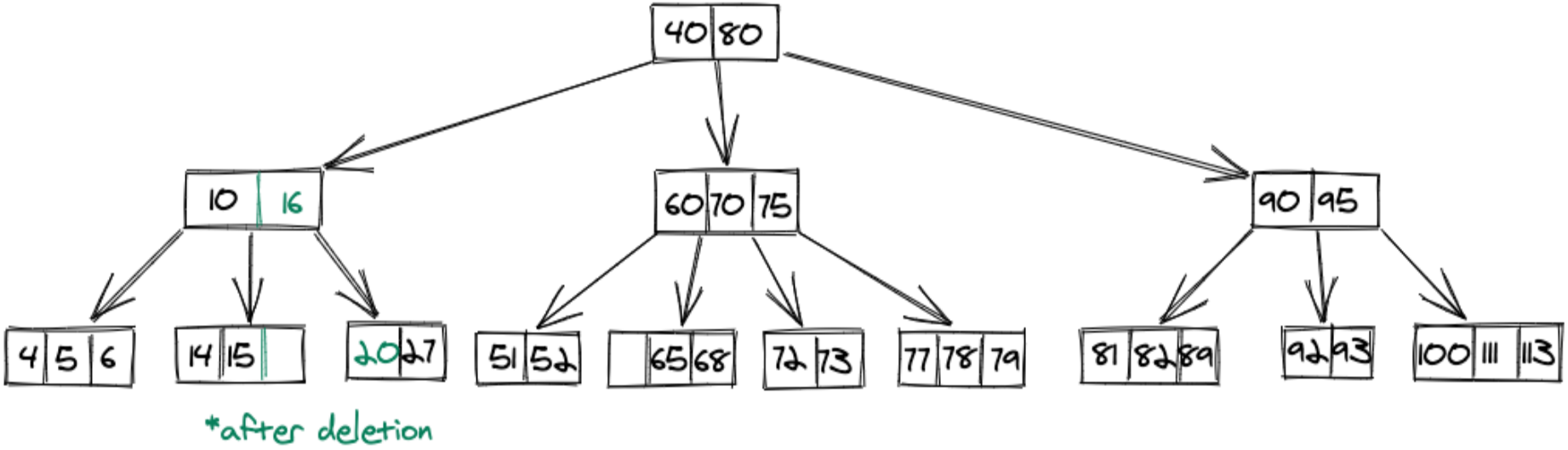
B-Tree Deletion-Leaf node

B-Tree of order 4

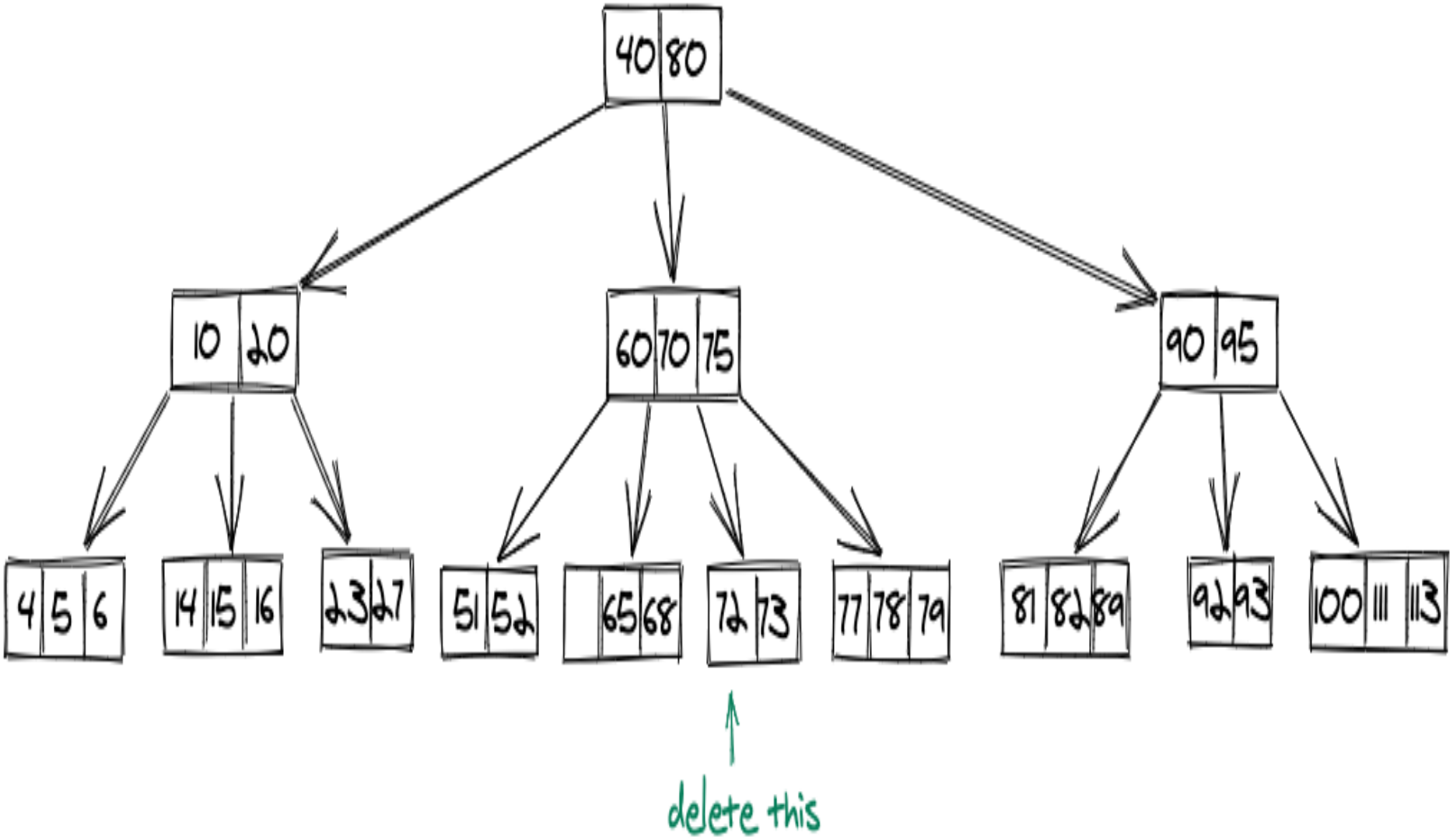


B-Tree Deletion-Leaf node

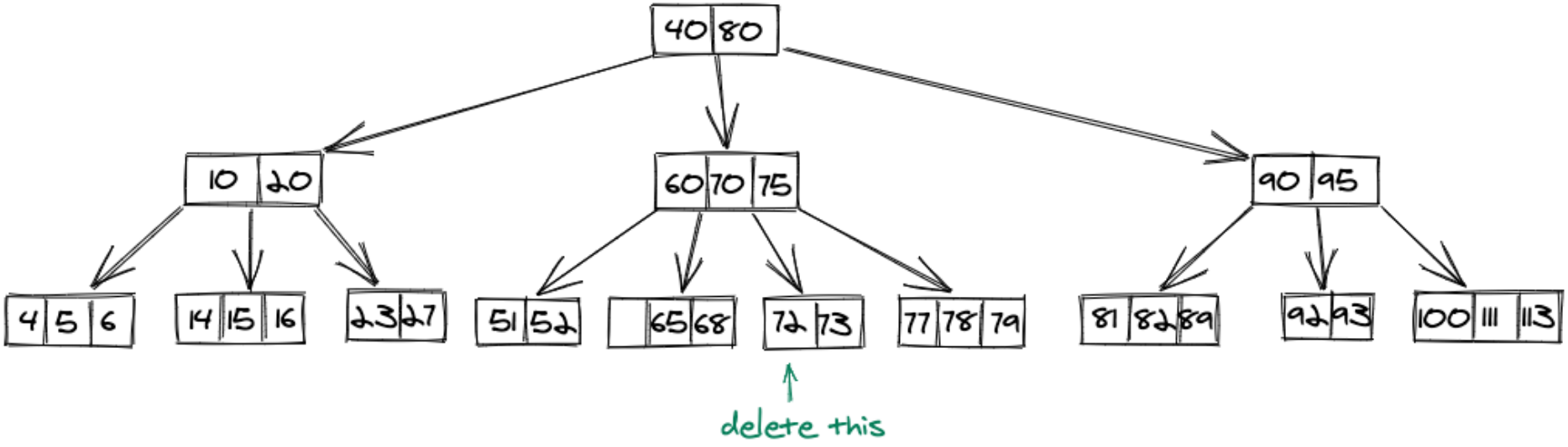


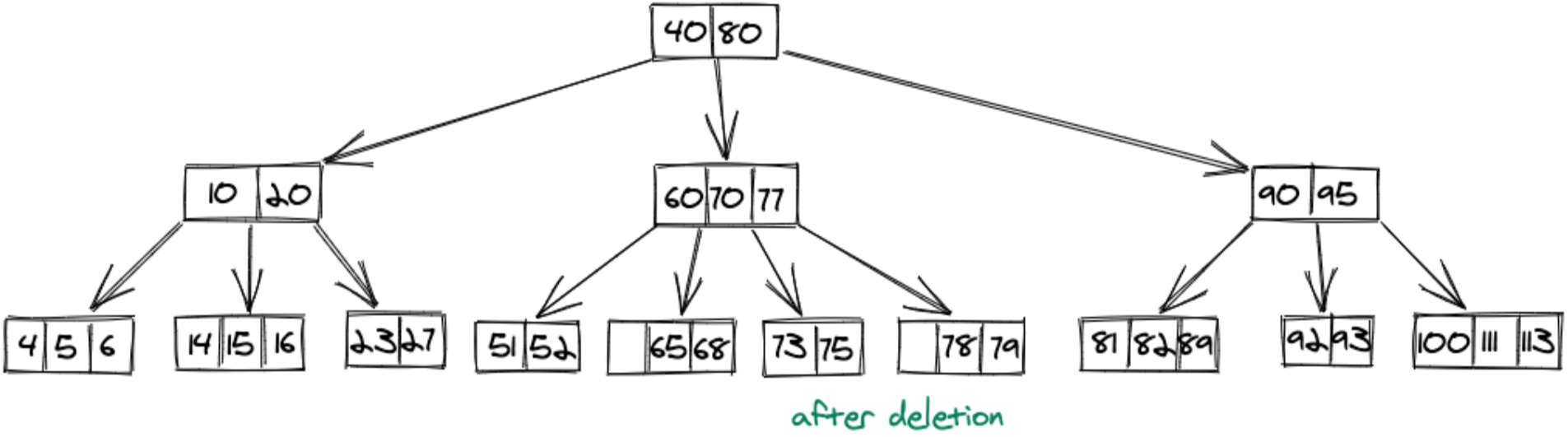


B-Tree Deletion-Leaf node

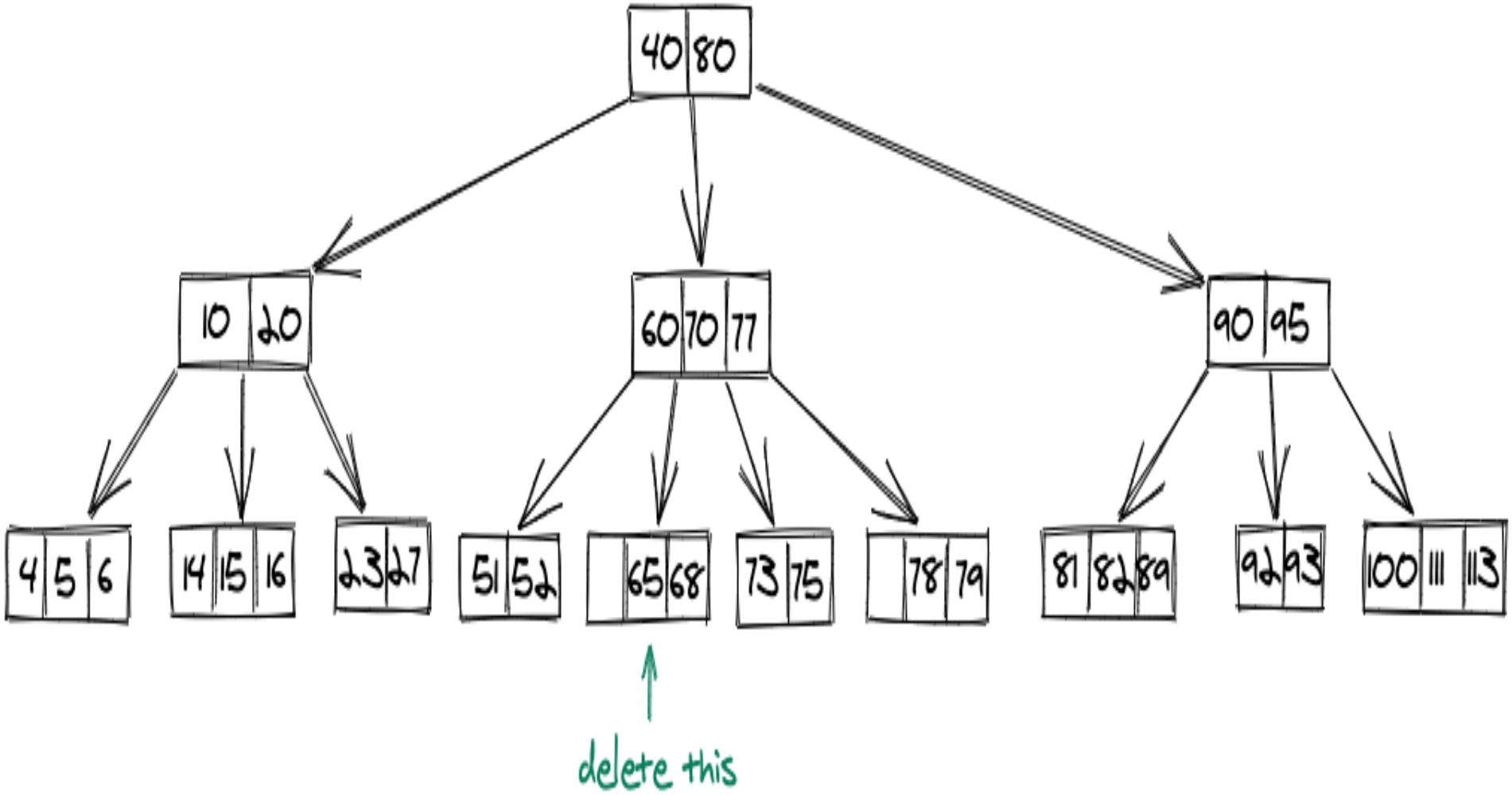


B-Tree Deletion-Leaf node

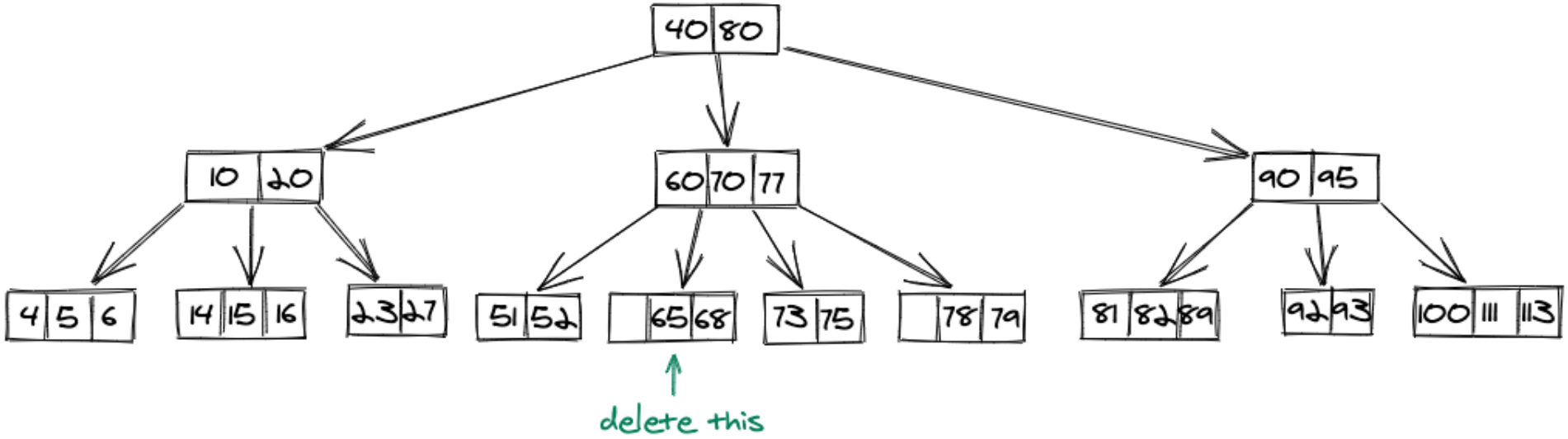


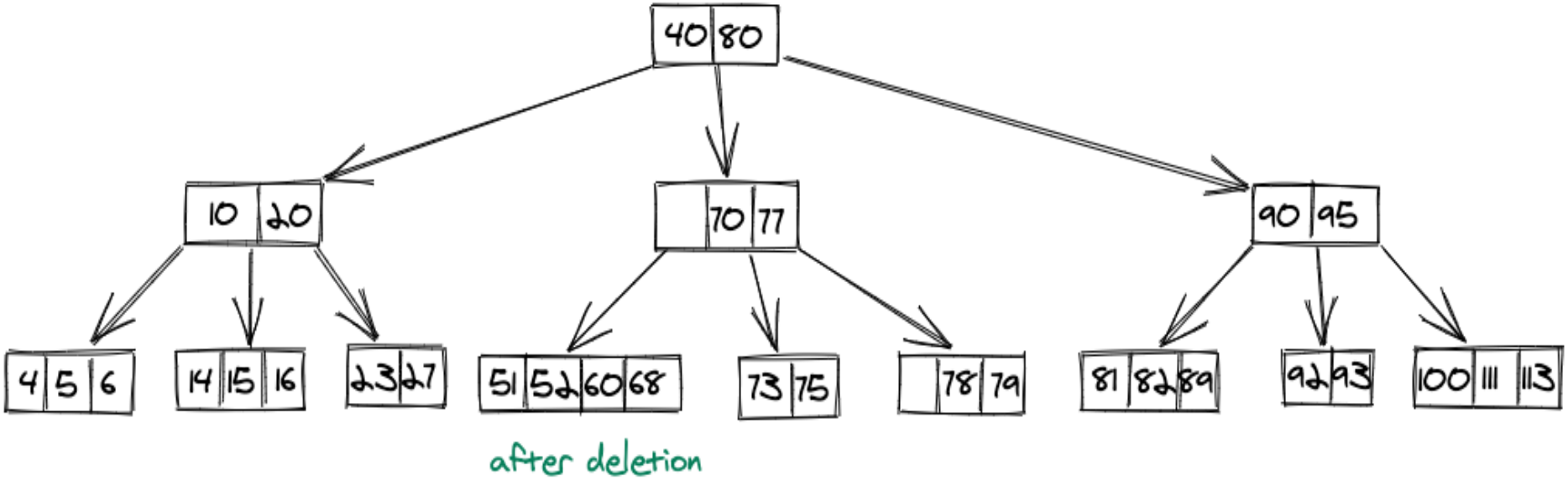


B-Tree Deletion-Leaf Node

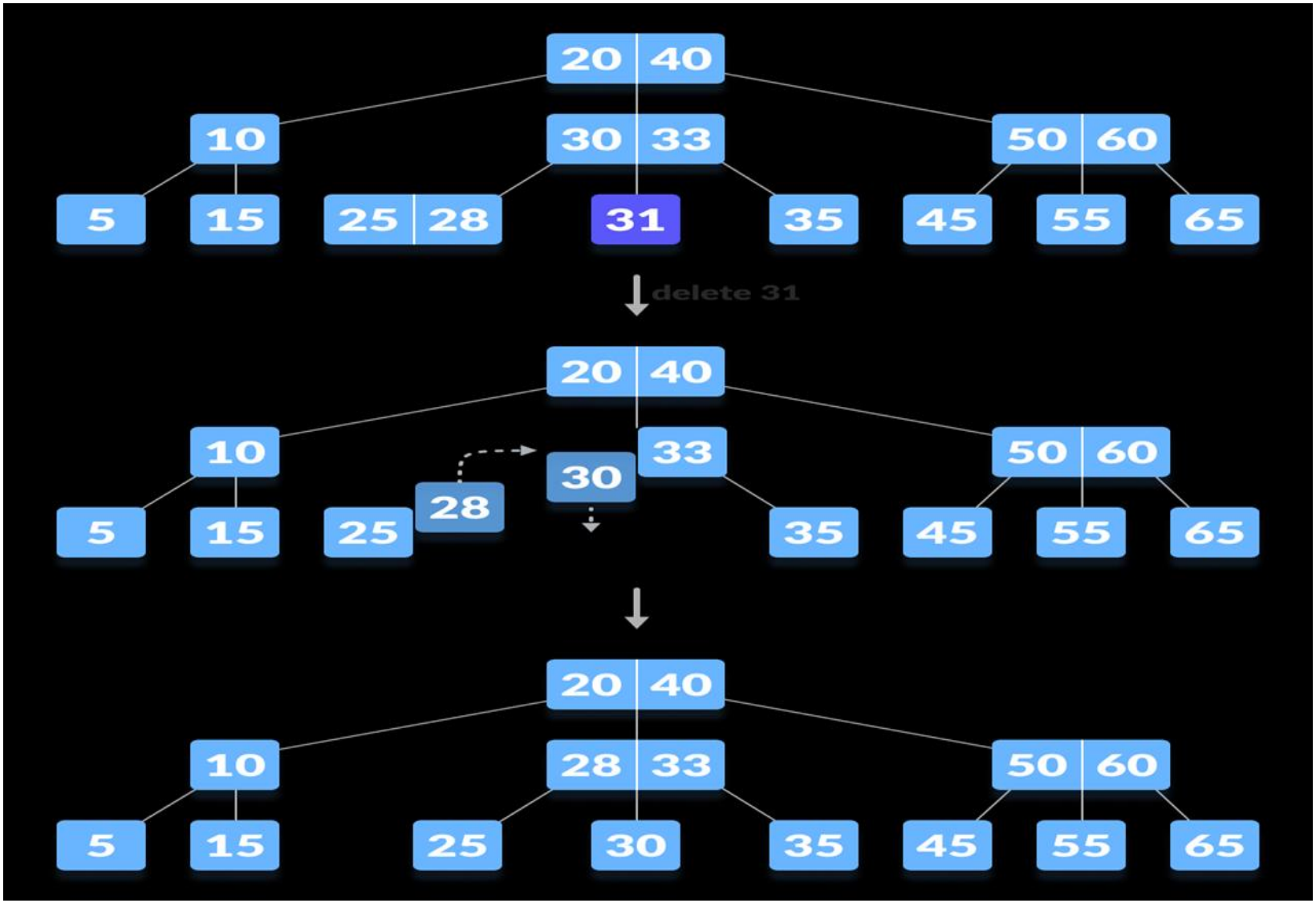


B-Tree Deletion-Leaf Node

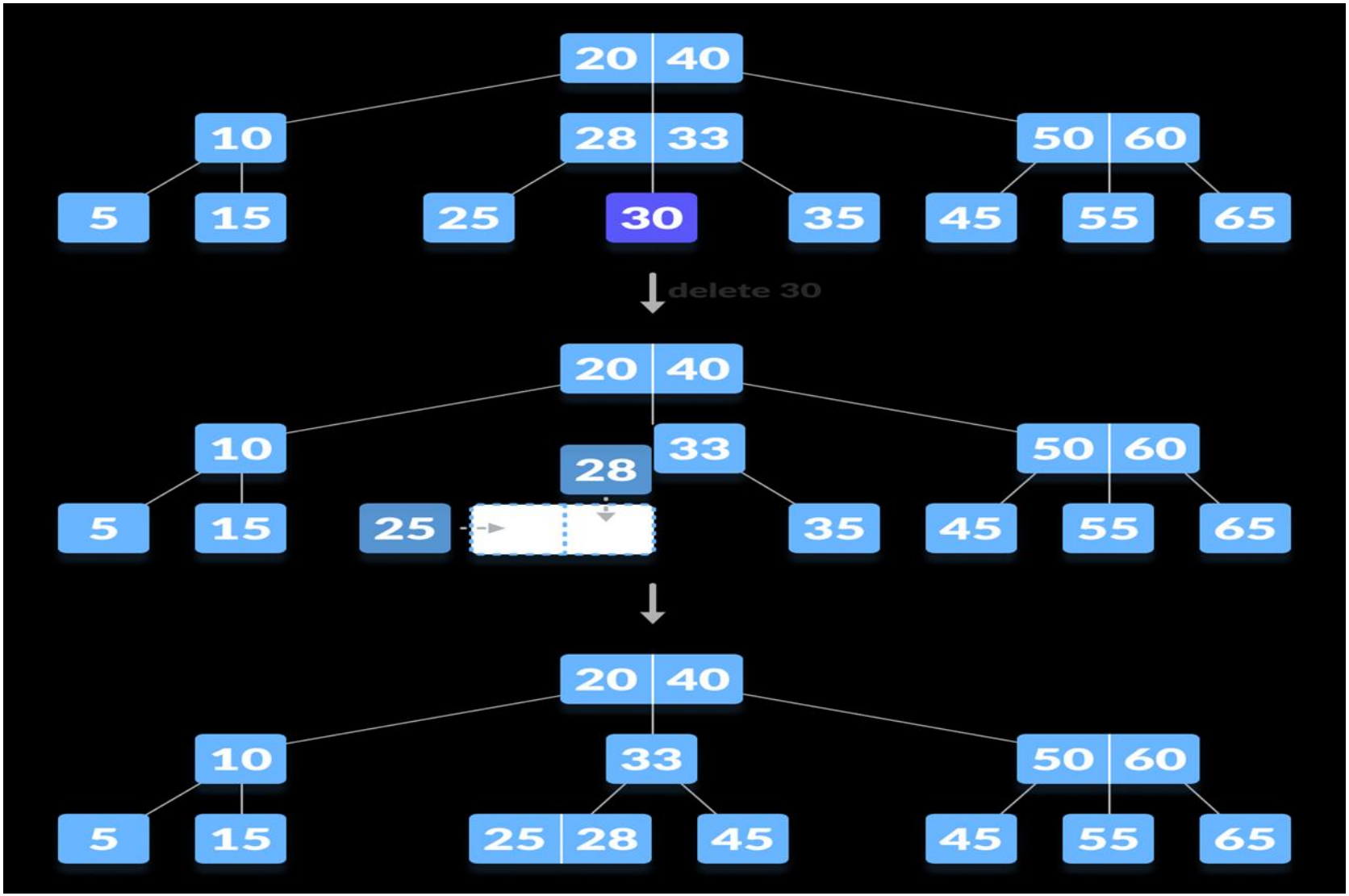




B-Tree Deletion-Leaf Node



B-Tree Deletion-Leaf Node



B-Tree Deletion

- Deletion of a key in a B-Tree includes two cases, these are:
  - **Deletion of key from a leaf node**
  - **Deletion of a key from an Internal node**

B-Tree Deletion-Internal Node

- If we want to delete a key that is present in an internal node, then we can either
  - take the value which is **inorder predecessor** of this key or
  - if taking that inorder predecessor violates the B tree property we can take the **inorder successor** of the key.
  - **If both are not possible merge both the child's.**

- In the **inorder predecessor** approach, we extract the **highest value in the left children** node of the node where our key is present.

- In the **inorder successor** approach, we extract the **lowest value in the right children** node of the node where our key is present.

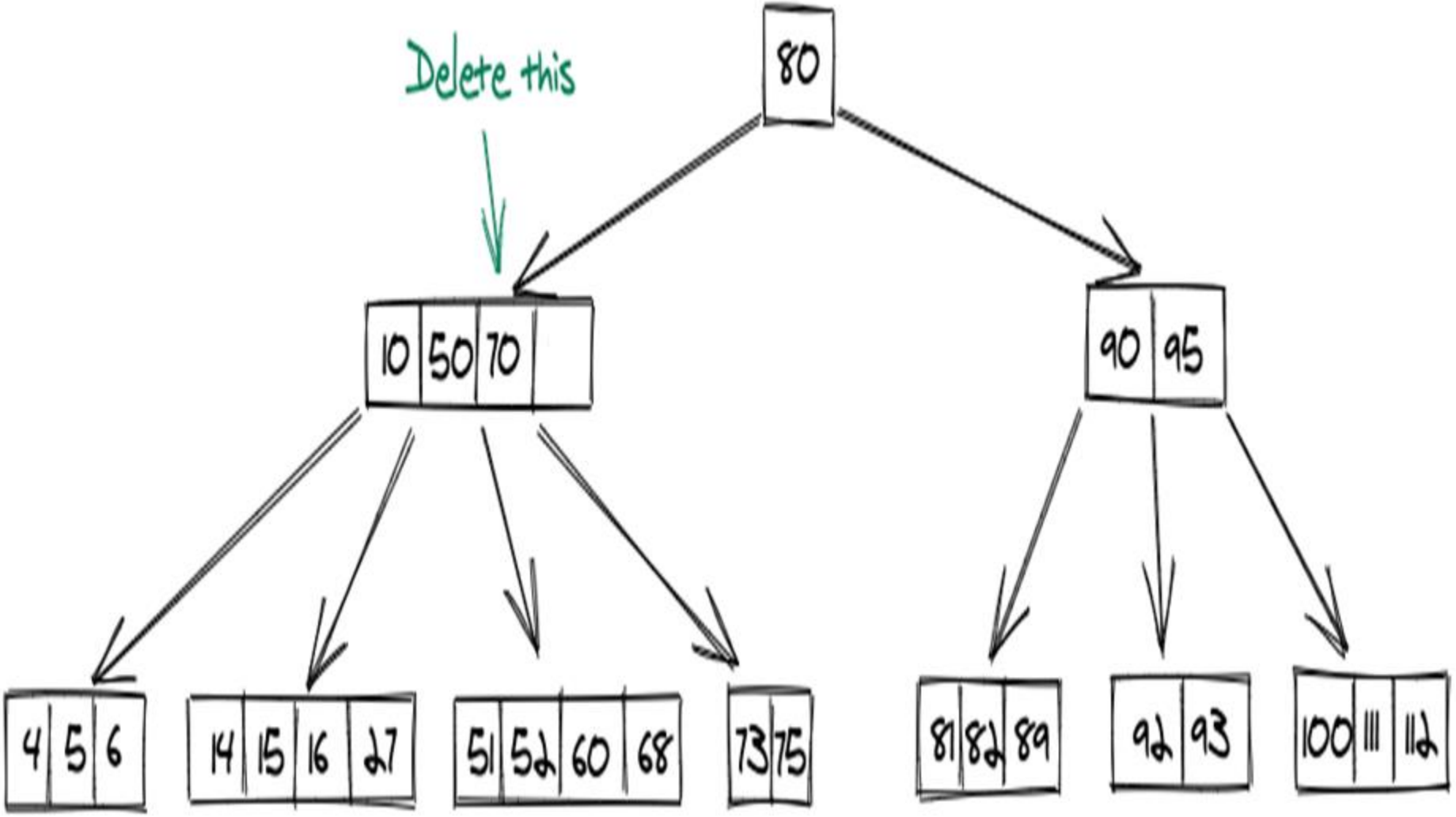
B-Tree Deletion-Internal Node

- After selecting the inorder predecessor or inorder successor of the element in that copy the selected element in the place of the element to be deleted.

- Delete the selected element from the leaf node.

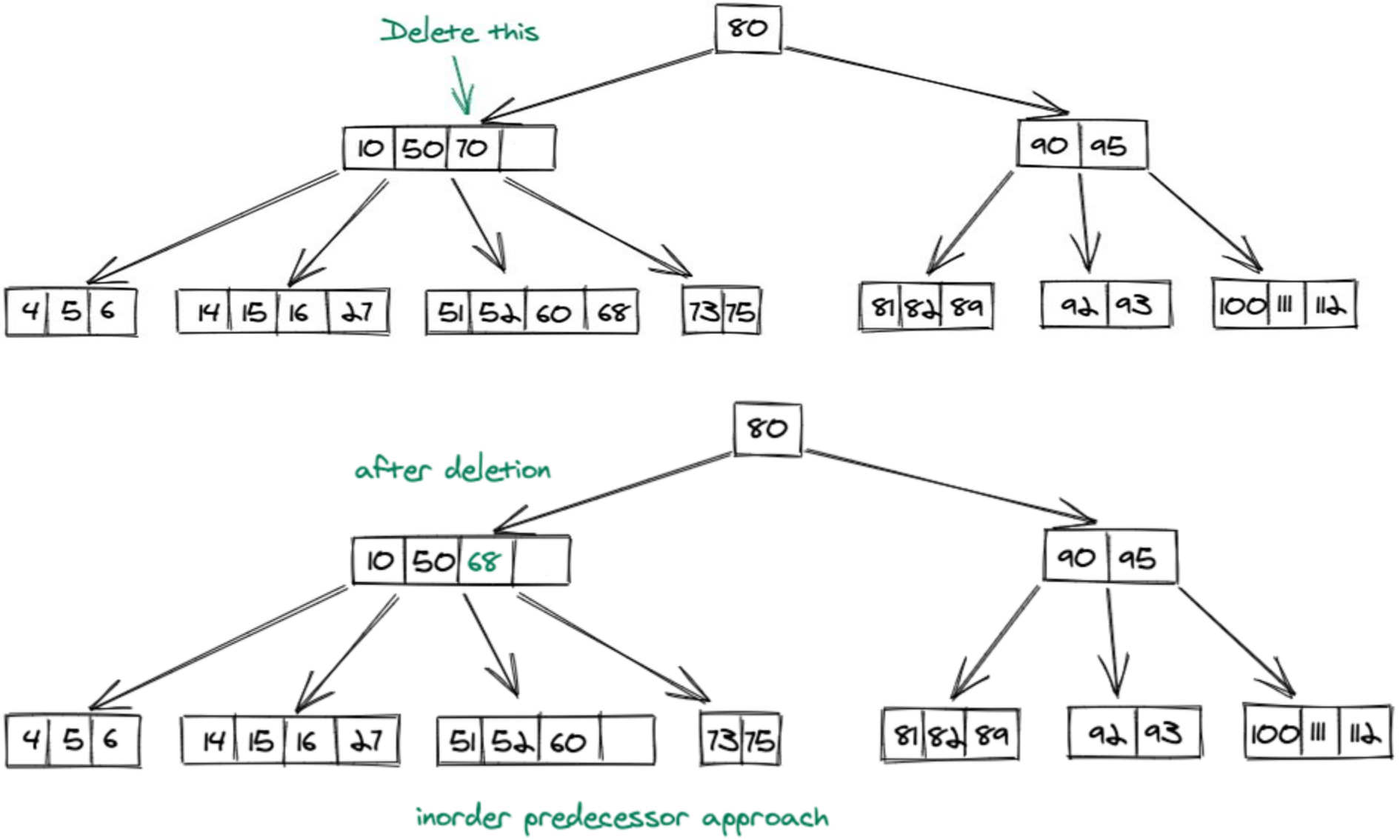
B-Tree Deletion-Internal Node

B-Tree of order 5: **Inorder predecessor approach**



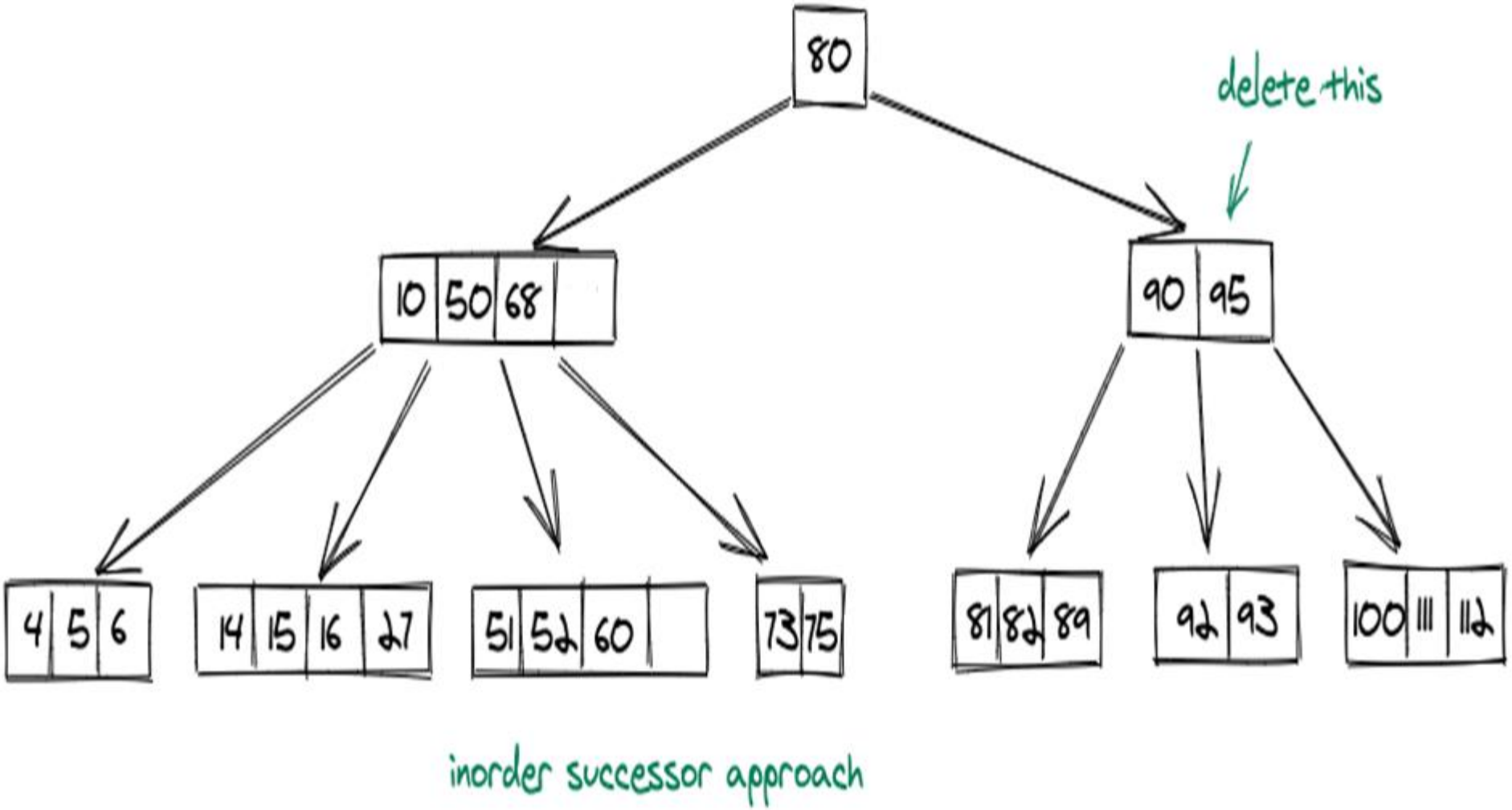
B-Tree Deletion-Internal Node

Inorder predecessor approach



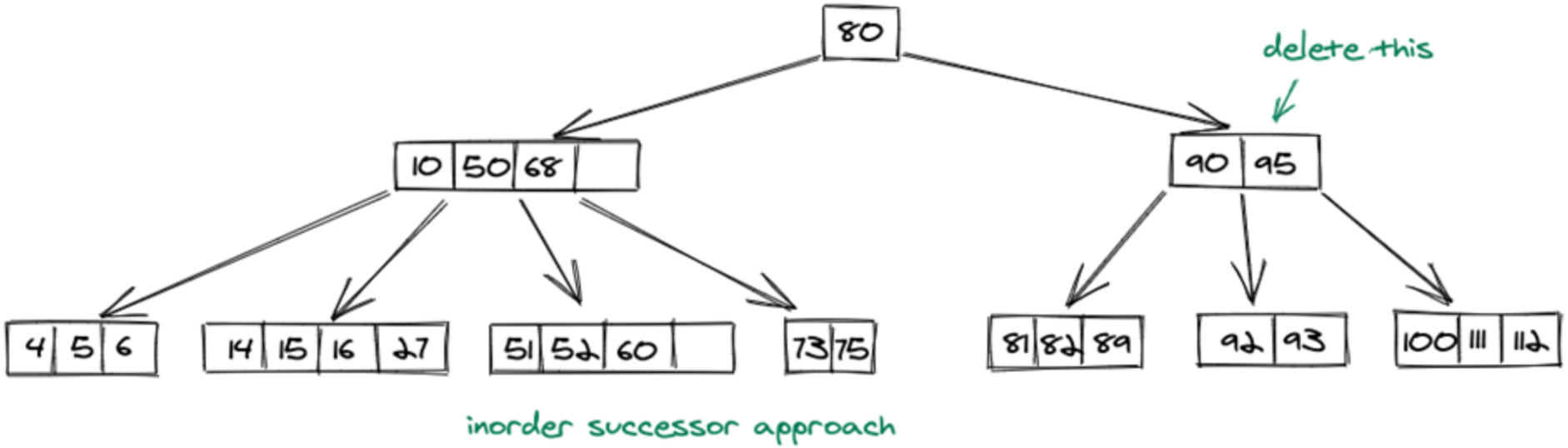
B-Tree Deletion-Internal Node

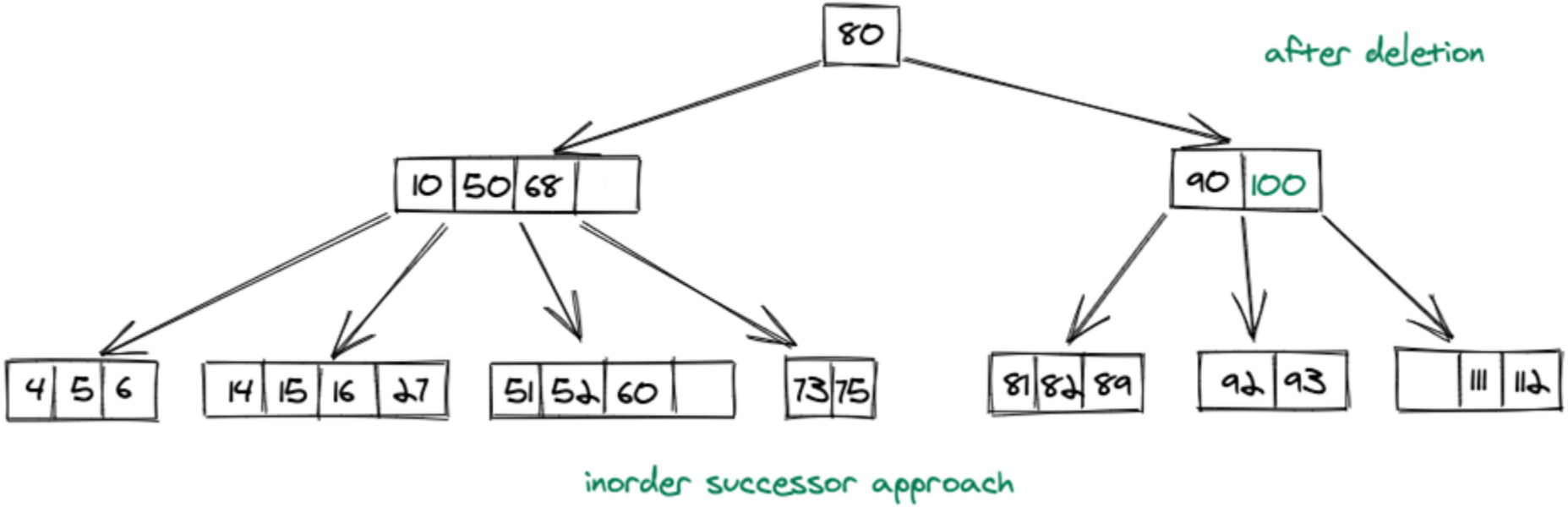
Inorder successor approach



B-Tree Deletion-Internal Node

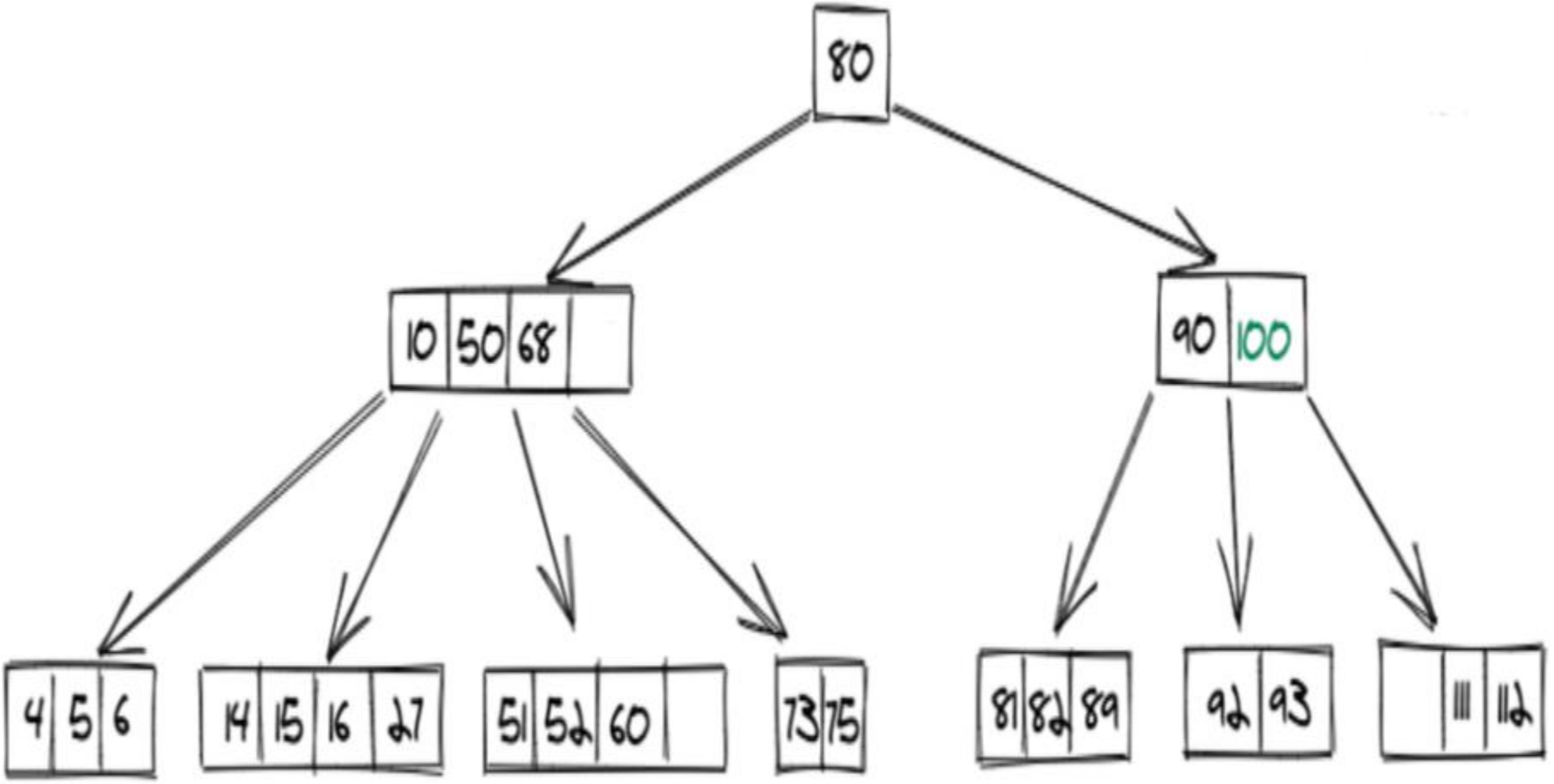
Inorder successor approach





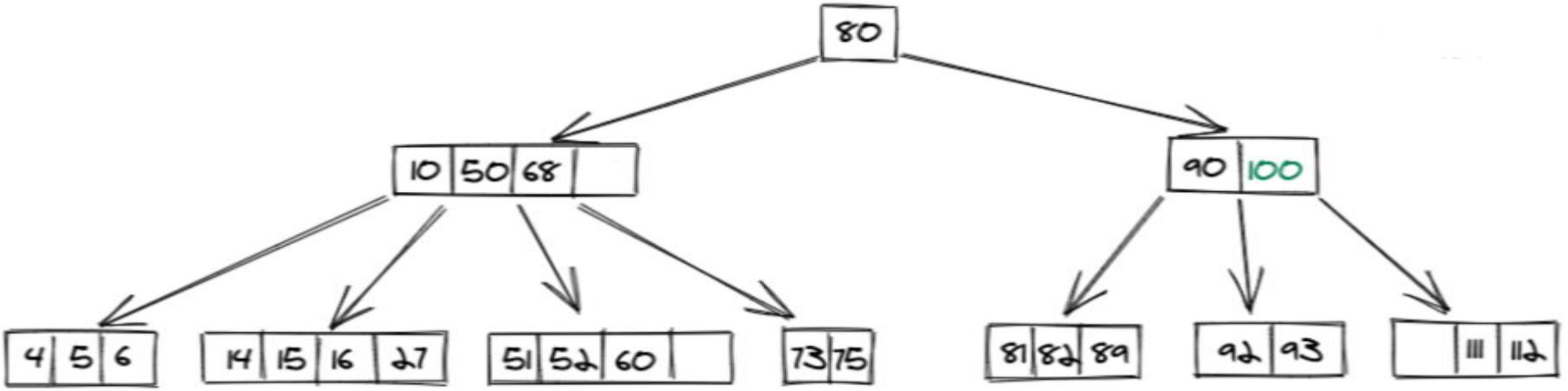
B-Tree Deletion-Internal Node

Delete 100

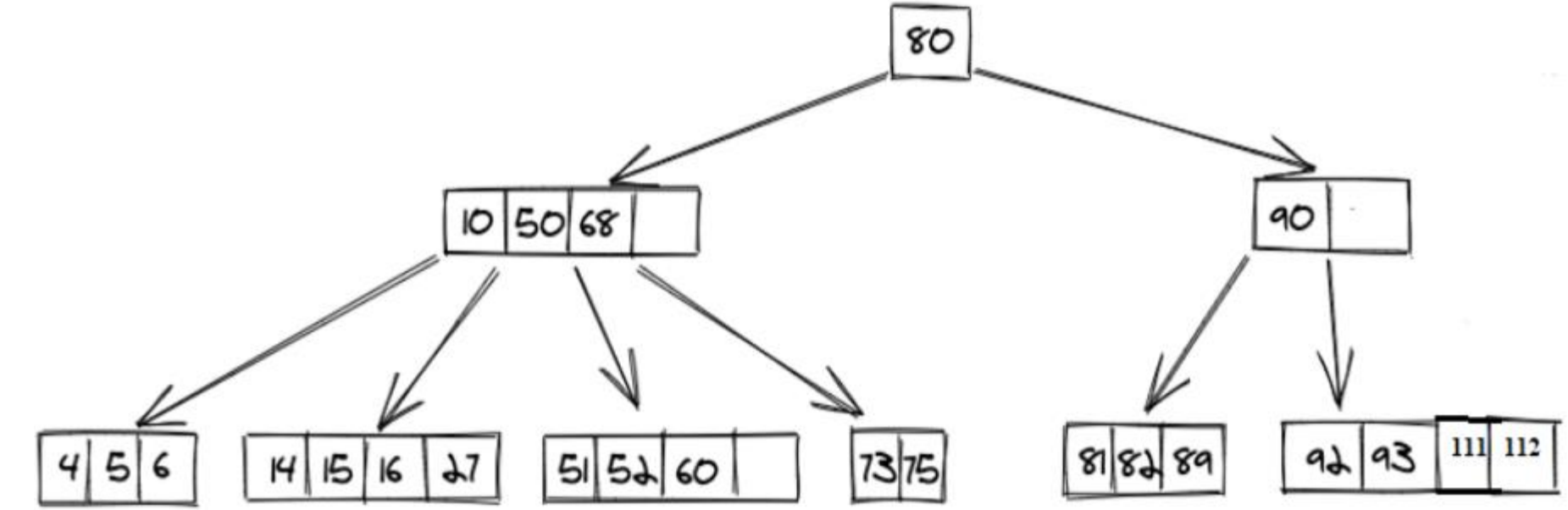


B-Tree Deletion-Internal Node

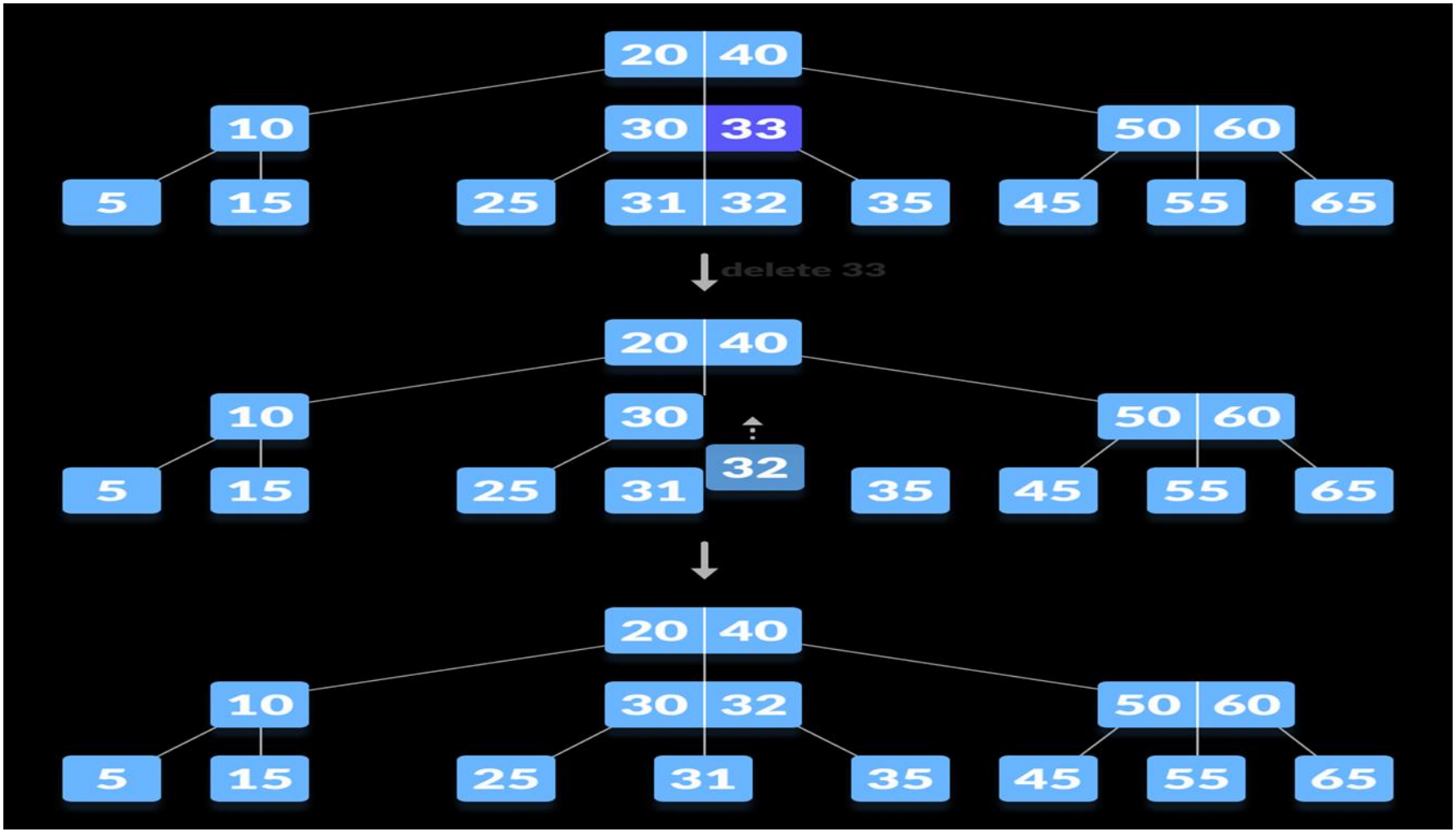
Delete 100



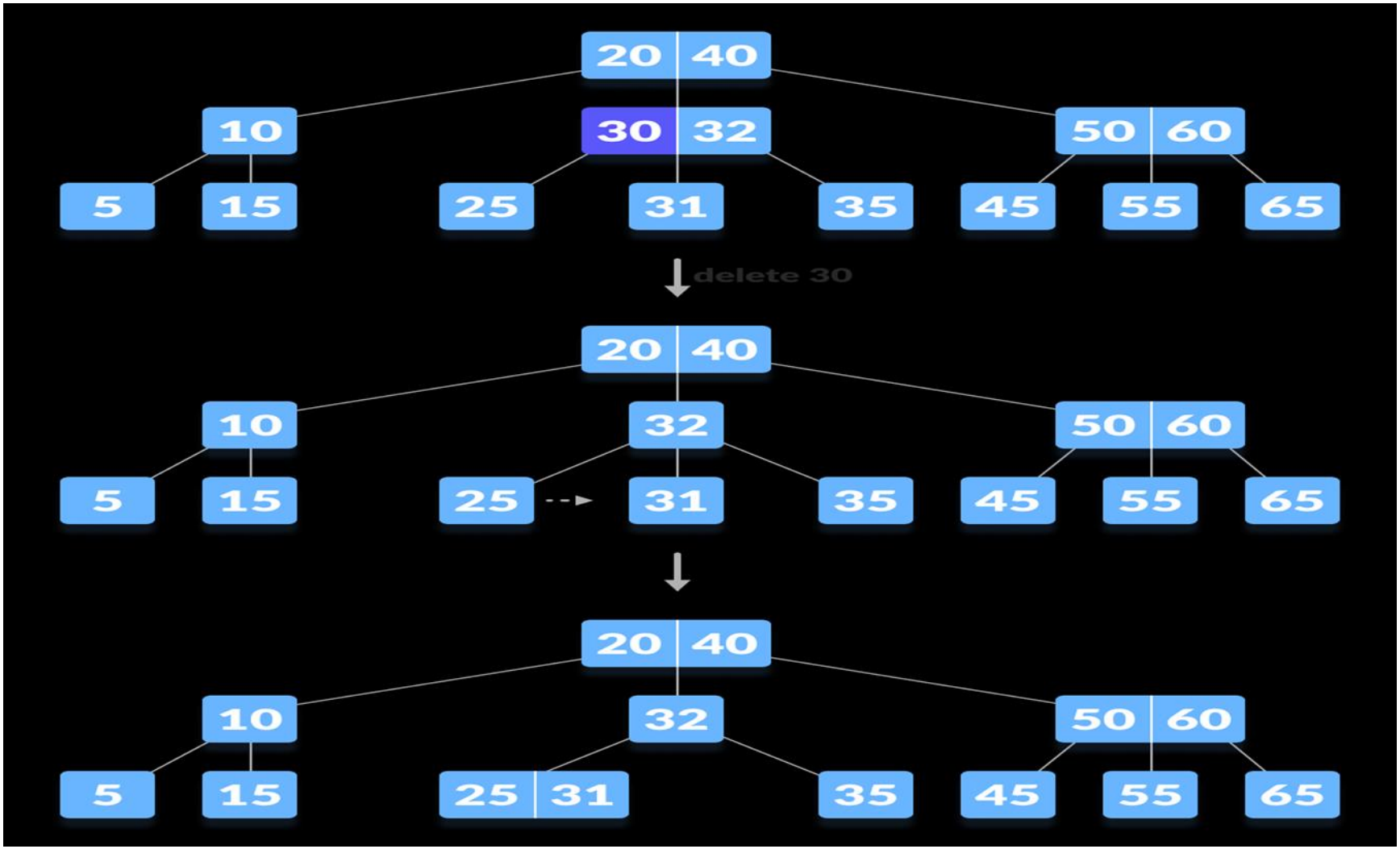
After Deletion of 100



B-Tree Deletion-Internal Node

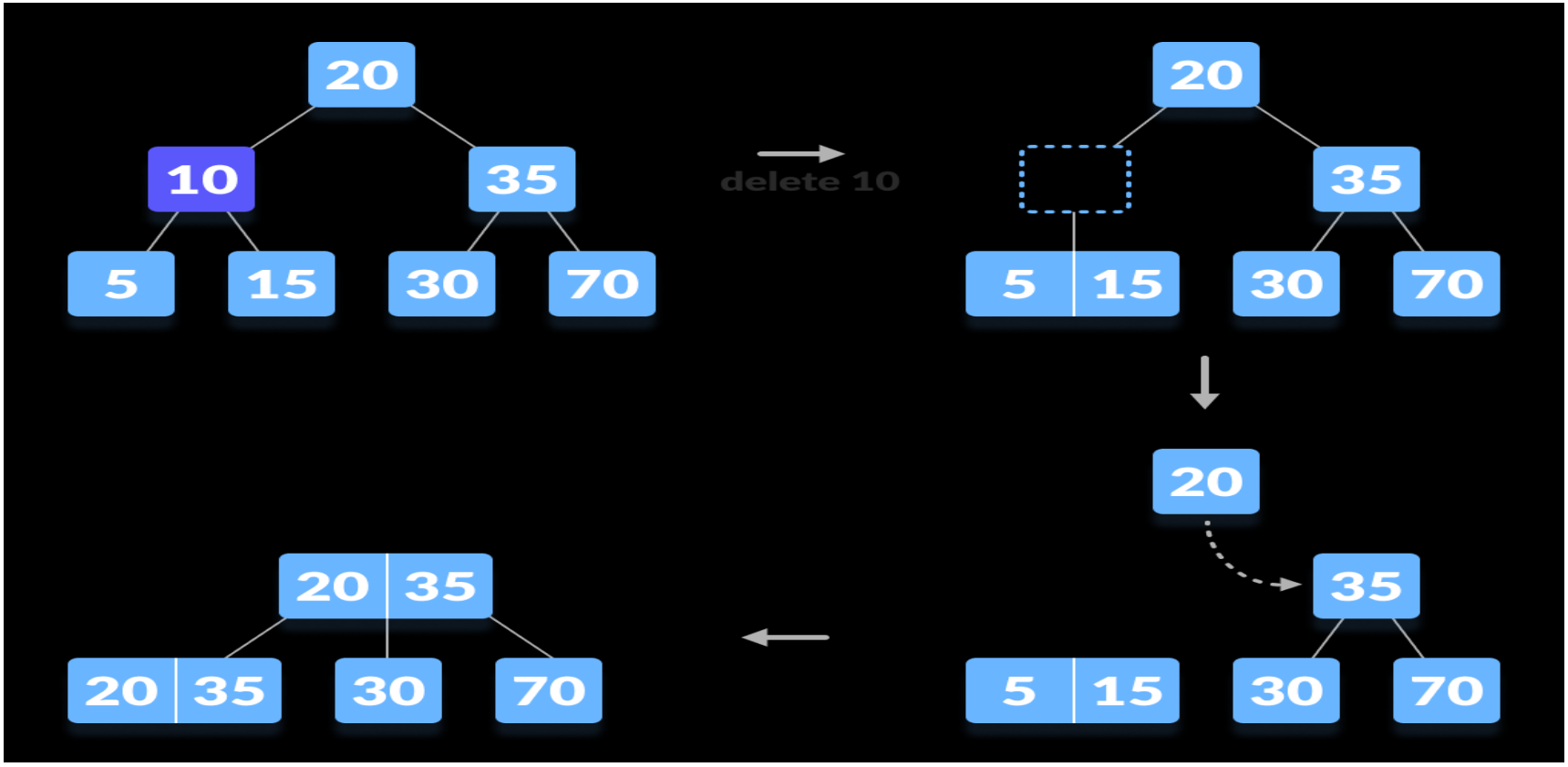


B-Tree Deletion-Internal Node



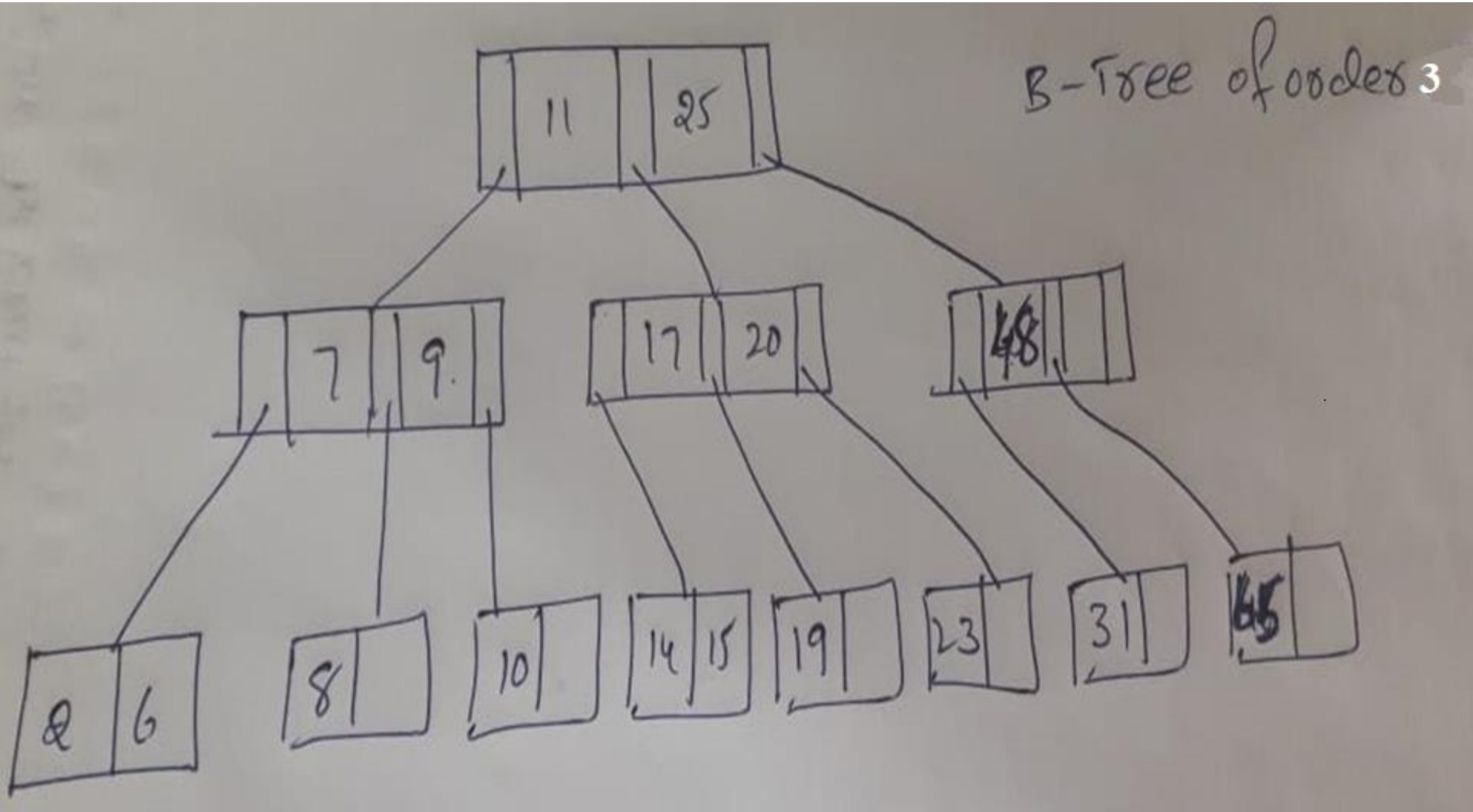
B-Tree Deletion-Internal Node spl case

- **In this case, the height of the tree shrinks**. If the target key lies in an internal node, and the deletion of the key leads to a fewer number of keys in the node then look for the inorder predecessor and the inorder successor. If both the children contain a minimum number of keys then, borrowing cannot take place. This leads to merging the children.
- Again, look for the sibling to borrow a key. But, if the sibling also has only a minimum number of keys then, merge the node with the sibling along with the parent. Arrange the children accordingly (increasing order).



Exercise on B-Tree Deletion

- **Make the tree empty by Deleting the elements in the sequence : 8,9,10,11,15,20,17,25,7,65,48,31,2,19,23,14,6**



Exercise on B-Trees

- Construct the B-tree of order 5 for the following list of elements

1, 7, 2, 9, 4, 6, 3, 10, 5, 11, 13, 17, 12, 25, 32, 78, 21, 29, 91,84,71,15,69 and make the tree empty by deleting the values in the same sequence.

- Constuct a B-Tree of order 5 for the following data.54, 4, 44, 3, 6, 7, 8, 12, 33, 56, 87, 52, 53, 9, 17, 28, 26, 16. **8M**