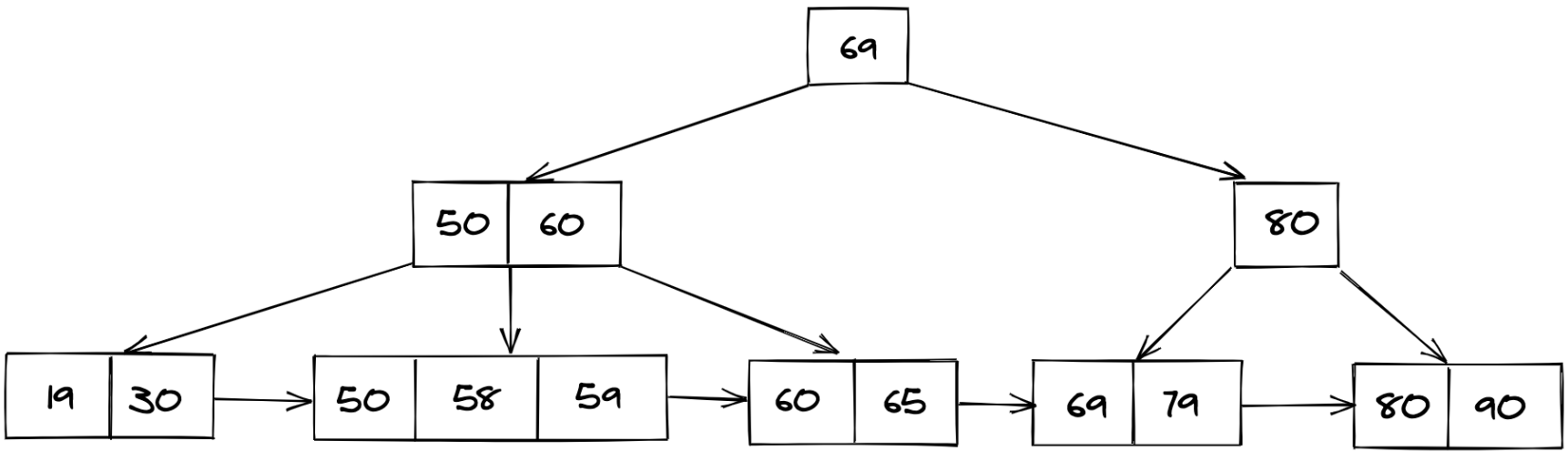
B+ Trees Data Structure

A **B+ tree** is an extension of a B tree which makes the search, insert and delete operations more efficient. We know that B trees allow both the data pointers and the key values in internal nodes as well as leaf nodes, this certainly becomes a drawback for B trees as the ability to insert the nodes at a particular level is decreased thus increase the node levels in it, which is certainly of no good. B+ trees reduce this drawback by simply **storing the data pointers at the leaf node level** and only storing the key values in the internal nodes. It should also be noted that the nodes at the leaf level are linked with each other, hence making the traversal of the data pointers easy and more efficient.

B+ trees come in handy when we want to store **a large amount of data** in the main memory. Since we know that the size of the main memory is not that large, so make use of the B+ trees, whose internal nodes that store the key(to access the records) are stored in the main memory whereas, the leaf nodes that contain the data pointers are actually stored in the secondary memory.

A pictorial representation of a B+ tree is given below:



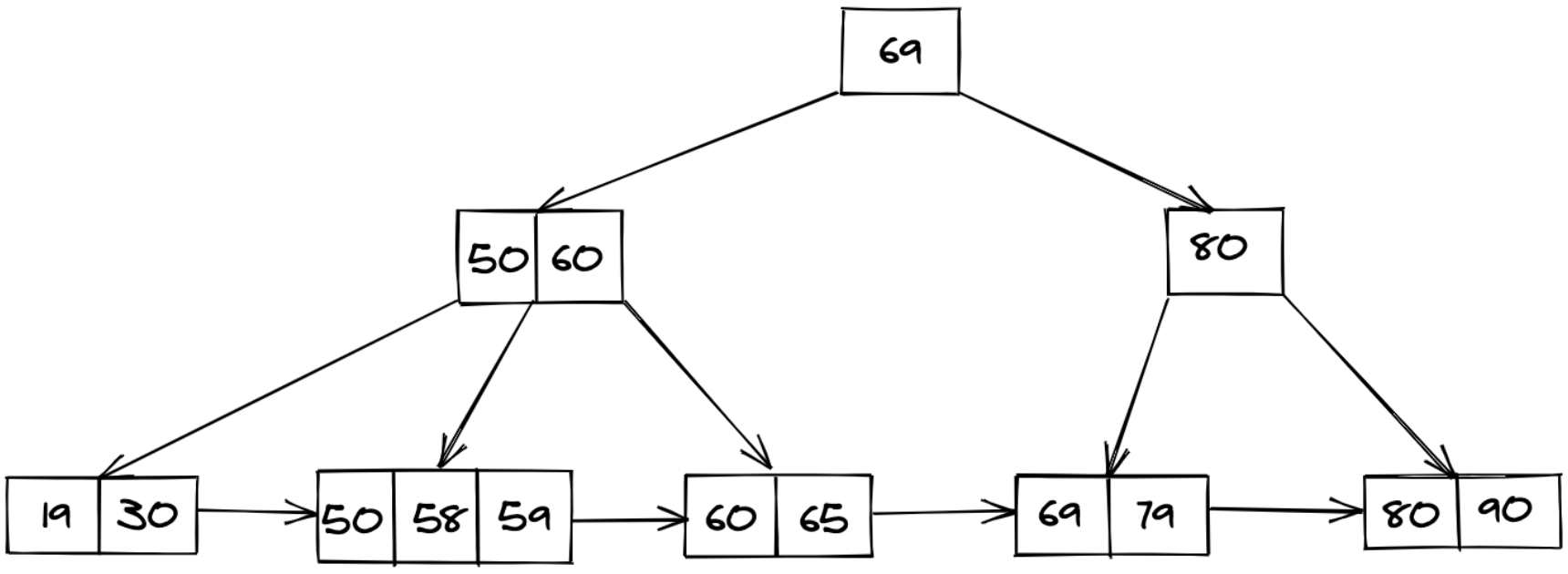
Why B+ trees?

- B+ trees store the records which later can be fetched in an equal number of disk accesses.
- The height of the B+ tree **remains balanced** and very less as when compared to B trees even if the number of records store in them is the same.
- Having less number of levels makes the accessing of records very easy.
- As the leaf nodes are connected with each other **like a linked list**, we can easily search elements in sequential manners.

Inserting in B+ tree

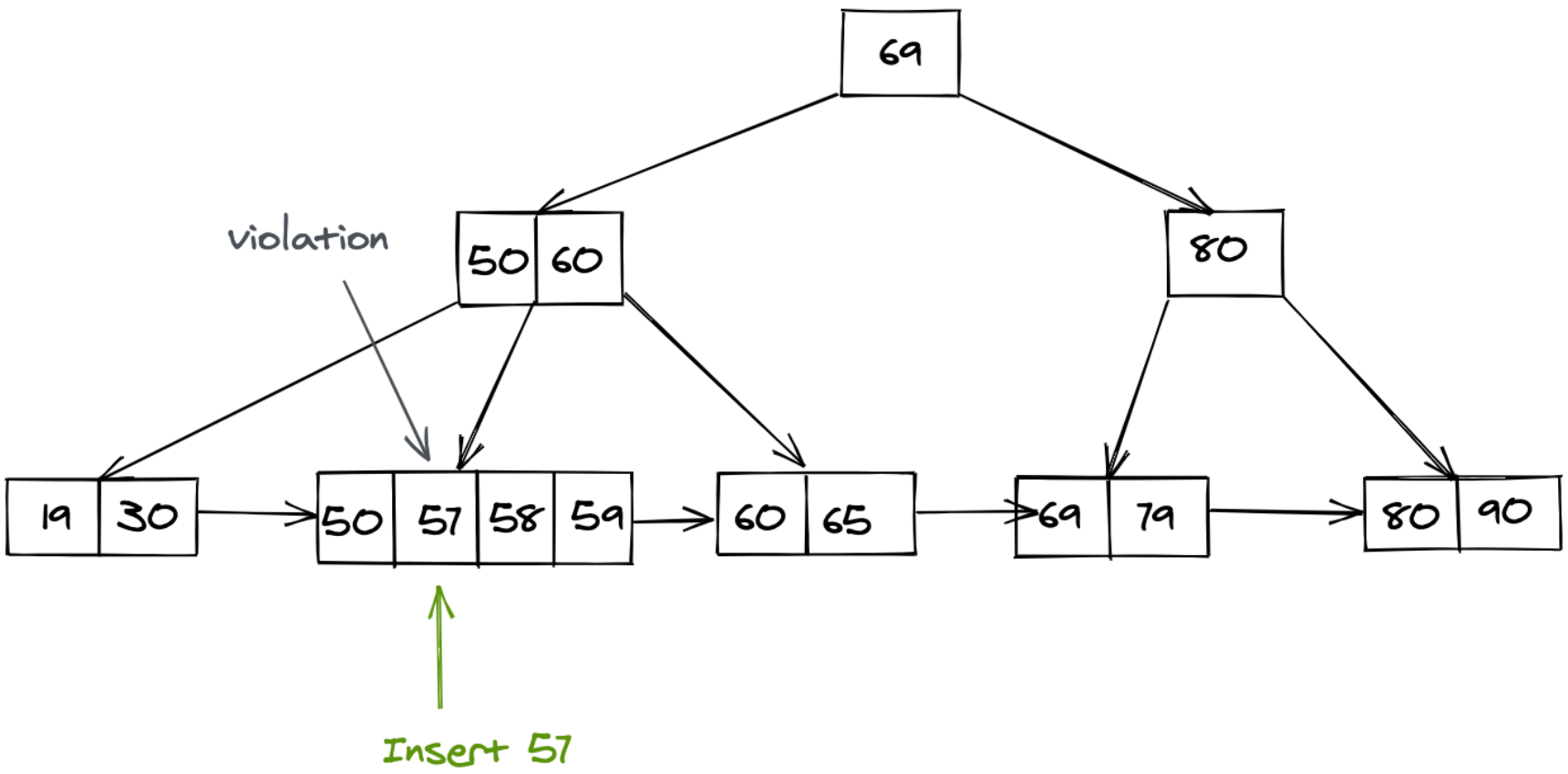
- Perform a search operation in the B+ tree to check the **ideal bucket location** where this new node should go to.
- If the bucket is not full(does not violate the B+ tree property), then add that node into this bucket.
- Otherwise split the nodes into two nodes and push the middle node(median node to be precise) to the parent node and then insert the new node.
- Repeat the above steps if the parent node is there and the current node keeps getting full.

Consider the pictorial representations shown below to understand the Insertion operation in the B+ tree:

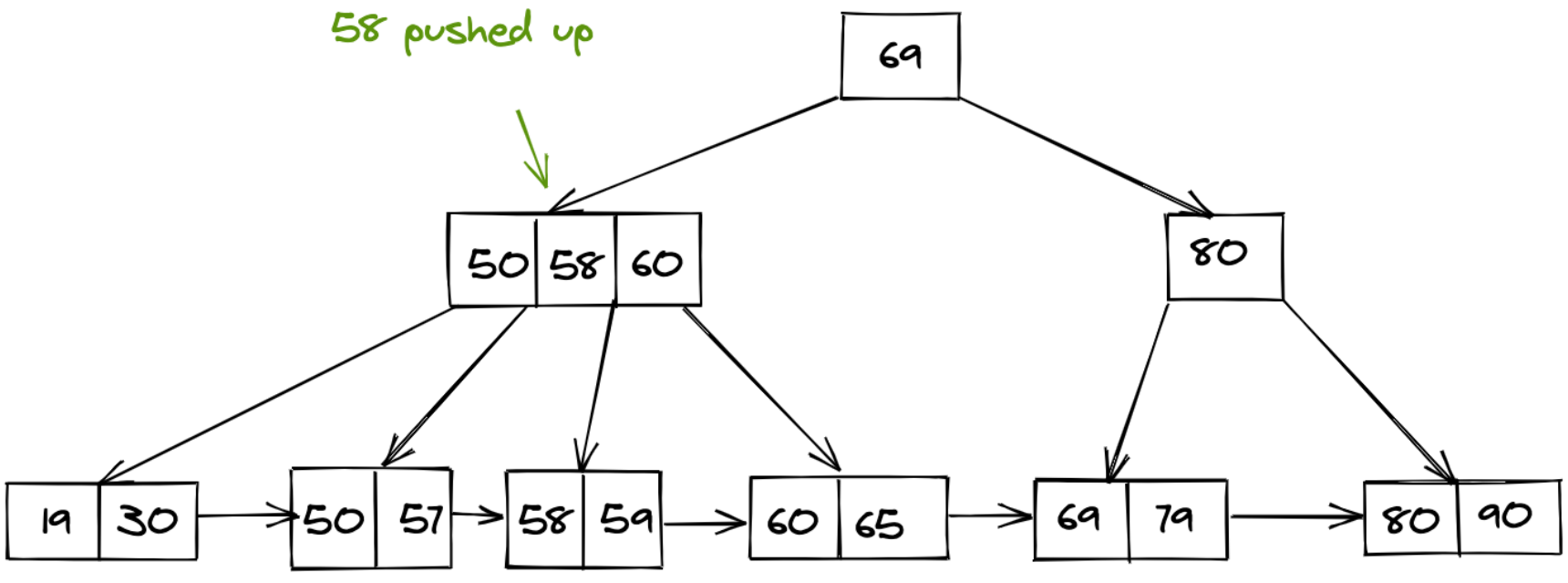


B+ tree of order 4

Let us try to insert 57 inside the above-shown B+ tree, the resultant B+ tree will look like this:



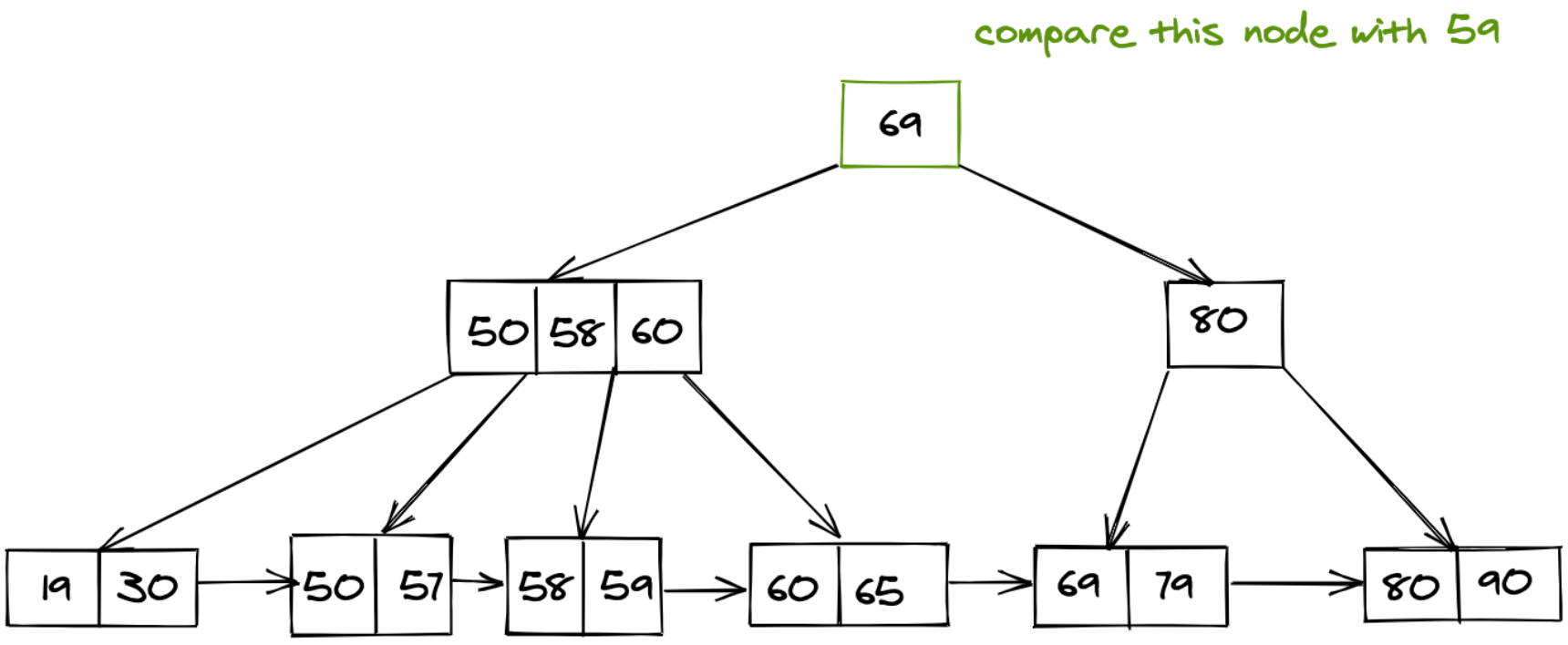
We know that the bucket(node) where we inserted the key with value 57 is now violating the property of the B+ tree, hence we need to split this node as mentioned in the steps above. After splitting we will push the median node to the parent node, and the resulting B+ tree will look like this:



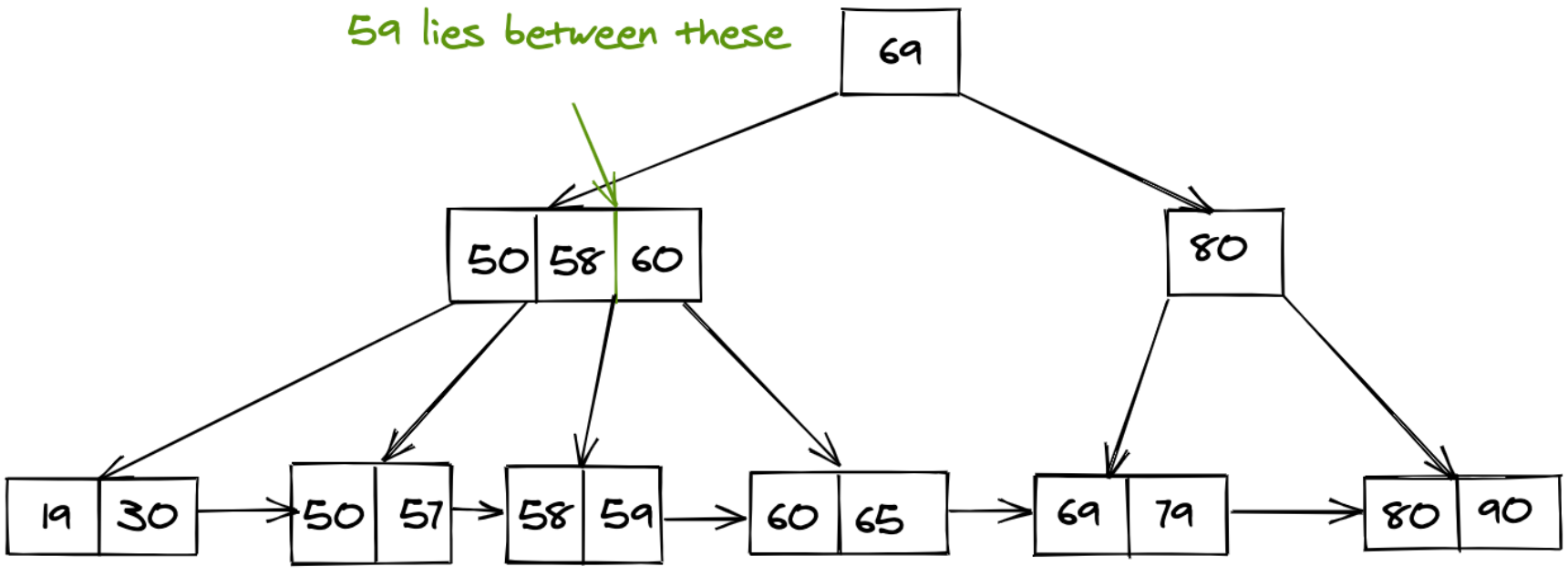
Searching in B+ tree:

Searching in a B+ tree is similar to searching in a BST. If the current value is less than the searching key, then traverse the left subtree, and if greater than first traverse this current bucket(node) and then check where the ideal location is.

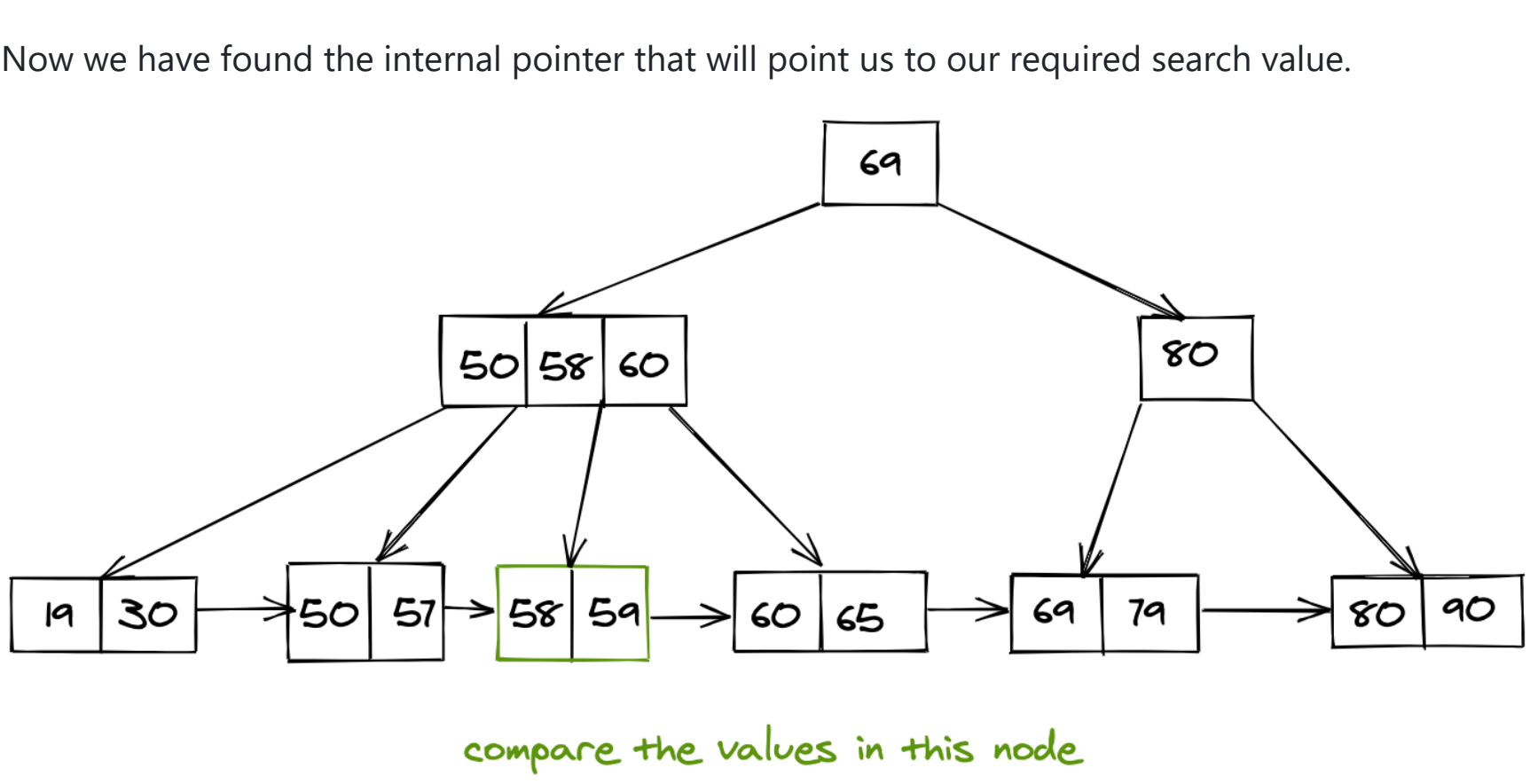
Consider the below representation of a B+ tree to understand the searching procedure. Suppose we want to search for a key with a value equal to 59 in the below given B+ tree.



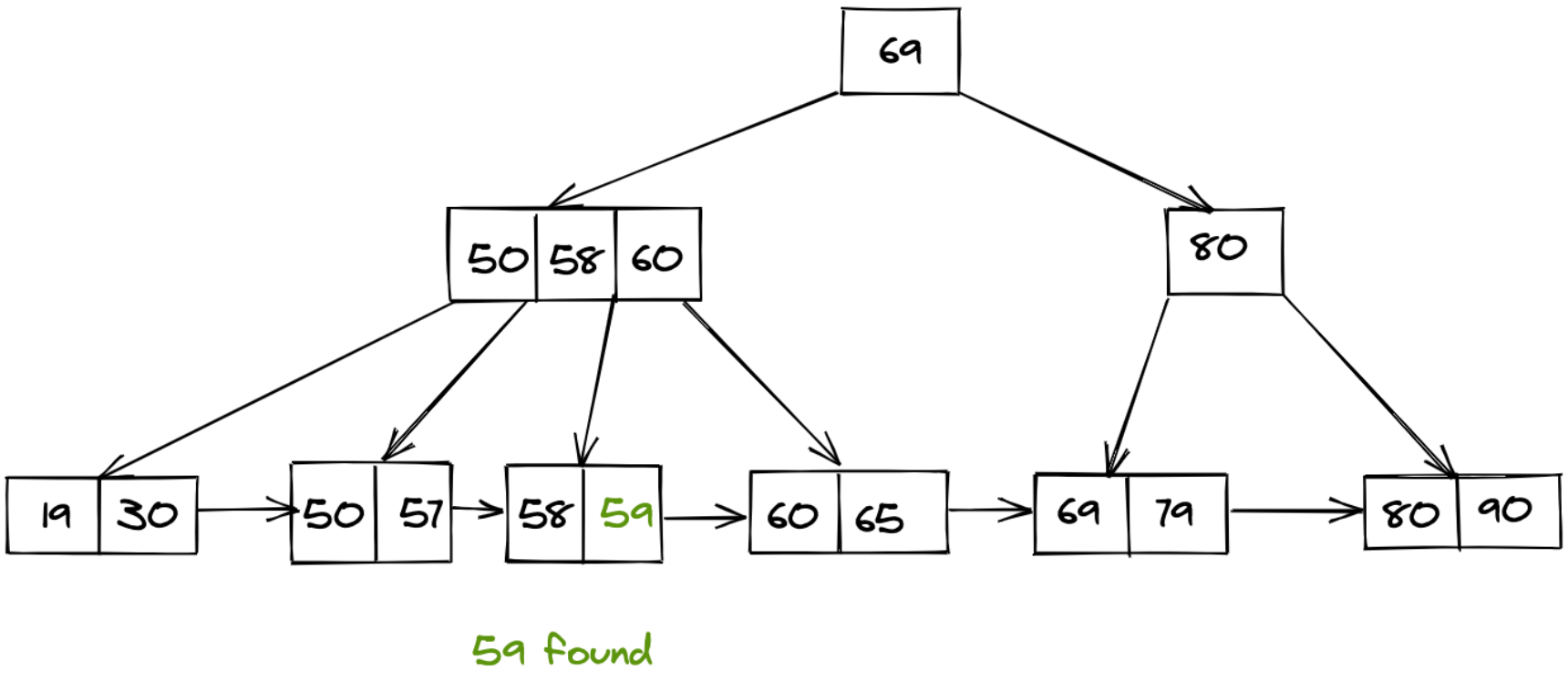
Now we know that 59 < 69, hence we traverse the left subtree.



Now we have found the internal pointer that will point us to our required search value.



Finally, we traverse this bucket in a linear fashion to get to our required search value.



Deletion in B+ tree:

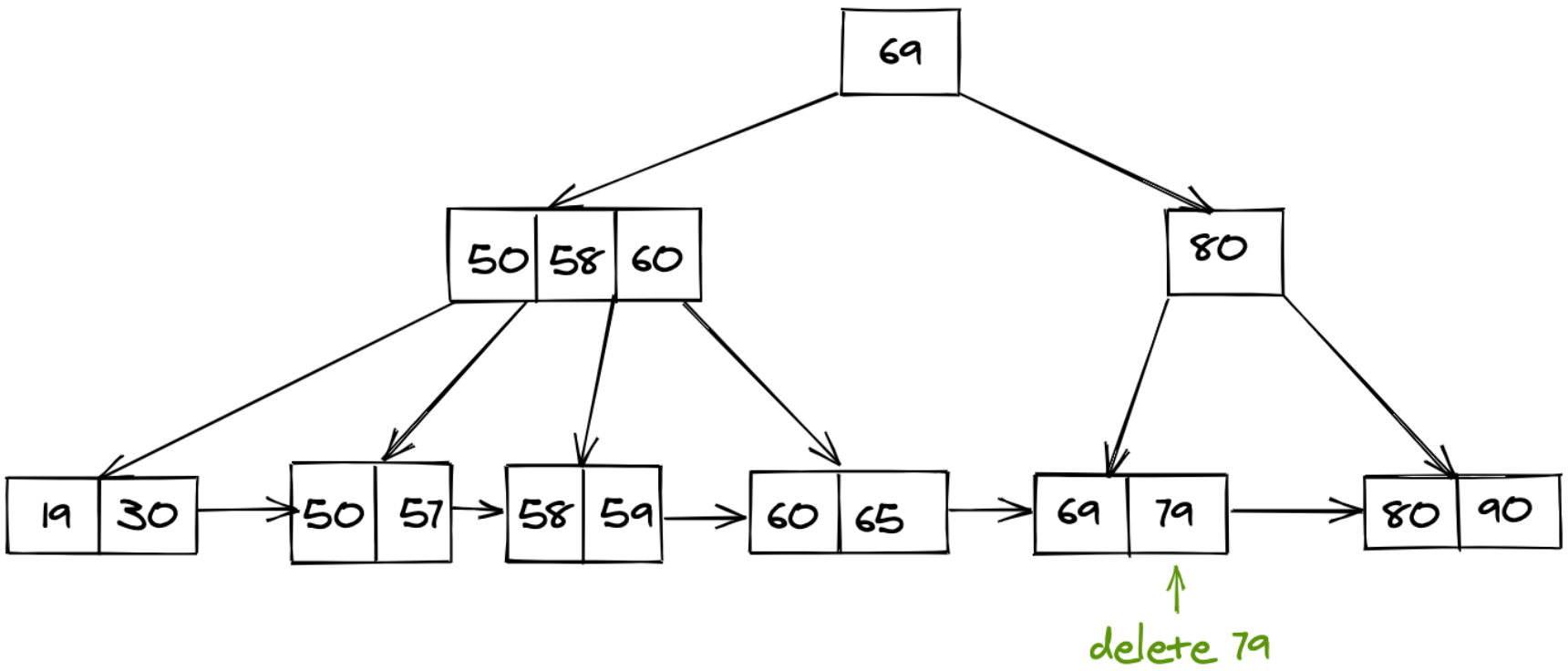
Deletion is a bit complicated process, two case arises:

- It is only present at the **leaf level**
- or, it contains a pointer from an internal node also.

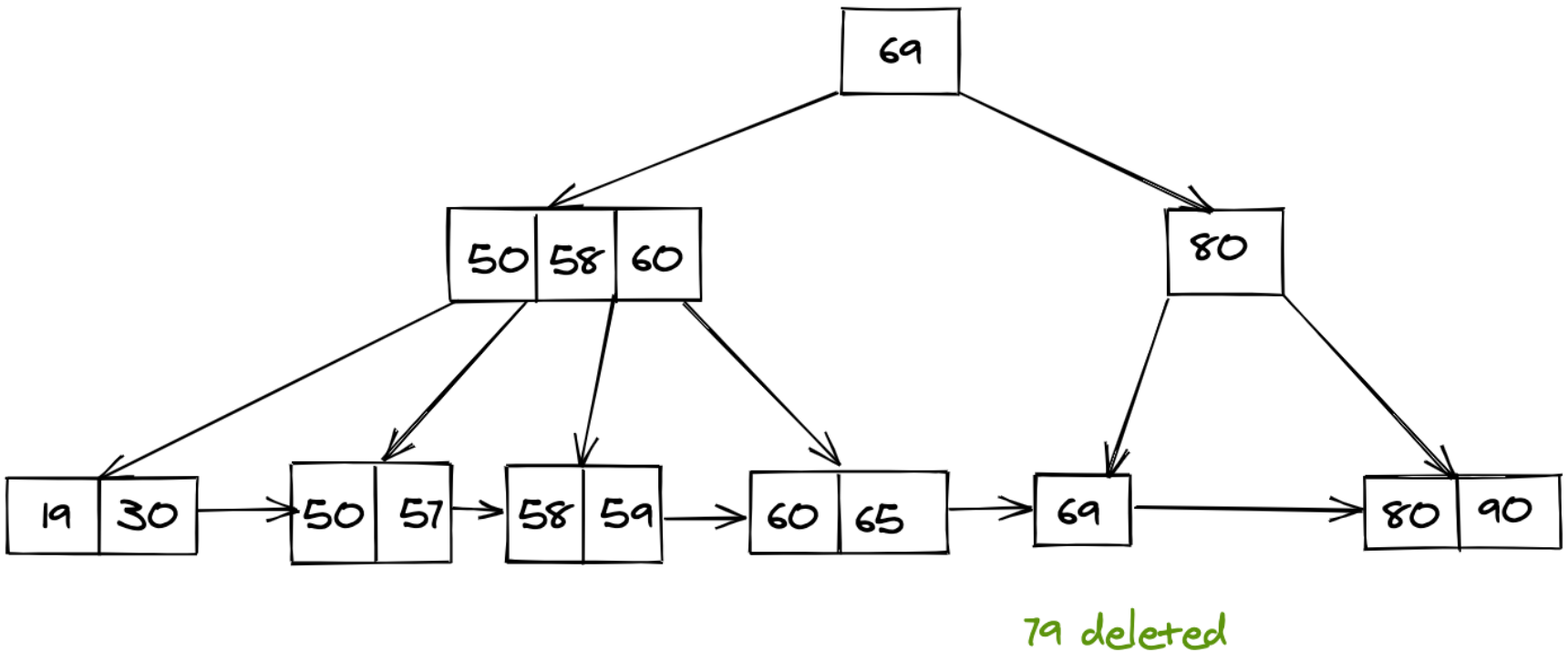
Deletion of only the leaf-node:

If it is present only as a **leaf node position**, then we can simply delete it, for which we first do the search operation and then delete it.

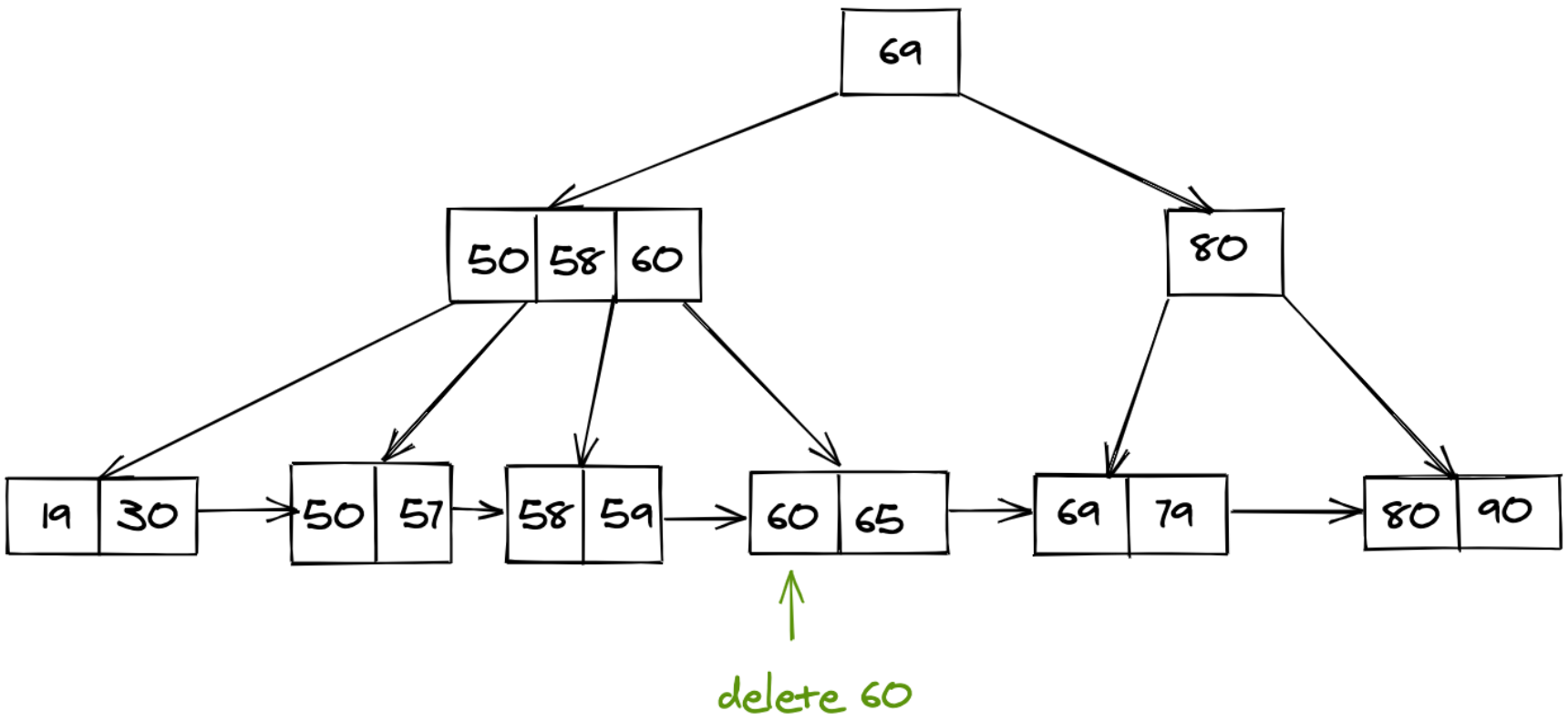
Consider the pictorial representation shown below:



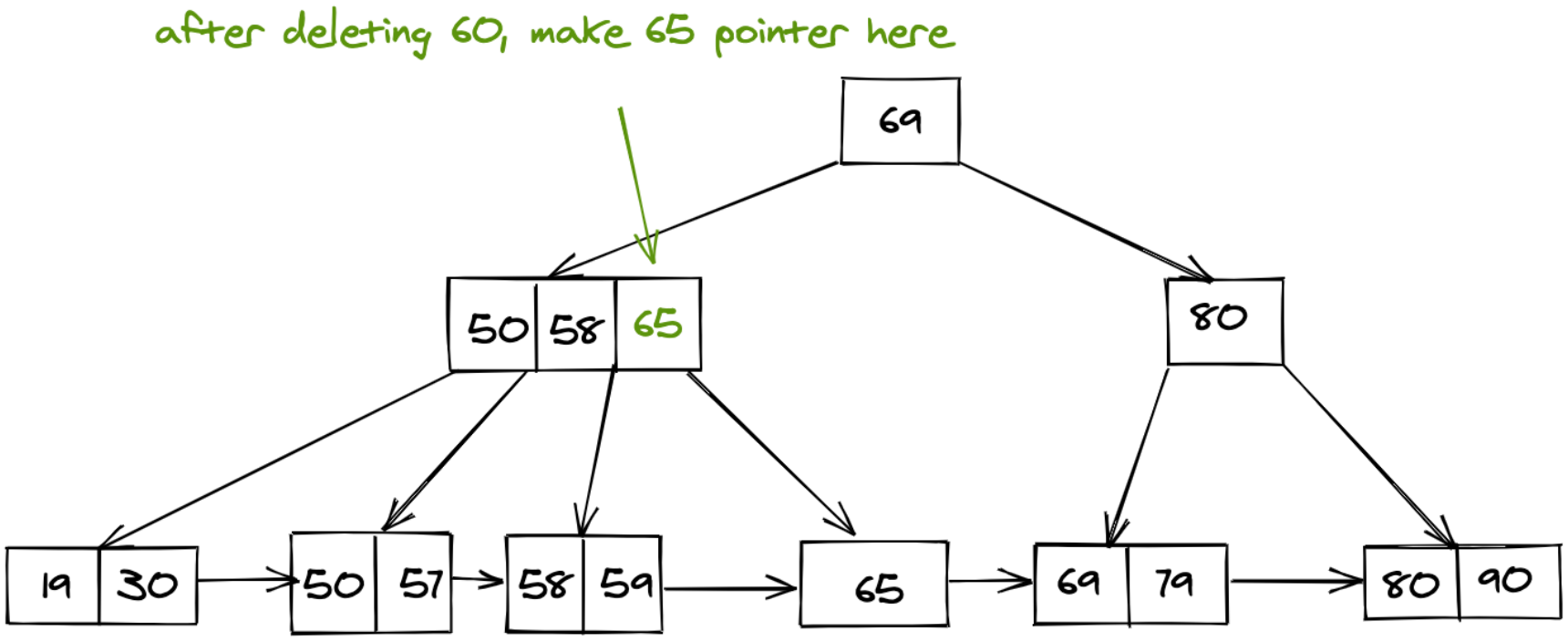
After the deletion of 79, we are left with the following B+ tree.



Deletion if a pointer to a leaf node is there:



After we locate the node we want to delete we must also delete the internal pointer that points to this node, and then we finally need to move the next node pointer to move to the parent node.



Conclusion:

- We learned what a B+ tree is, and how it is different from a B tree.

- Then we learned why we need a B+ tree.
- Lastly, we learned different operations that we do on a B+ tree-like searching, Insertion, and deletion of a record(key).