Unit 5 Part 1
Multi-Way Search Trees

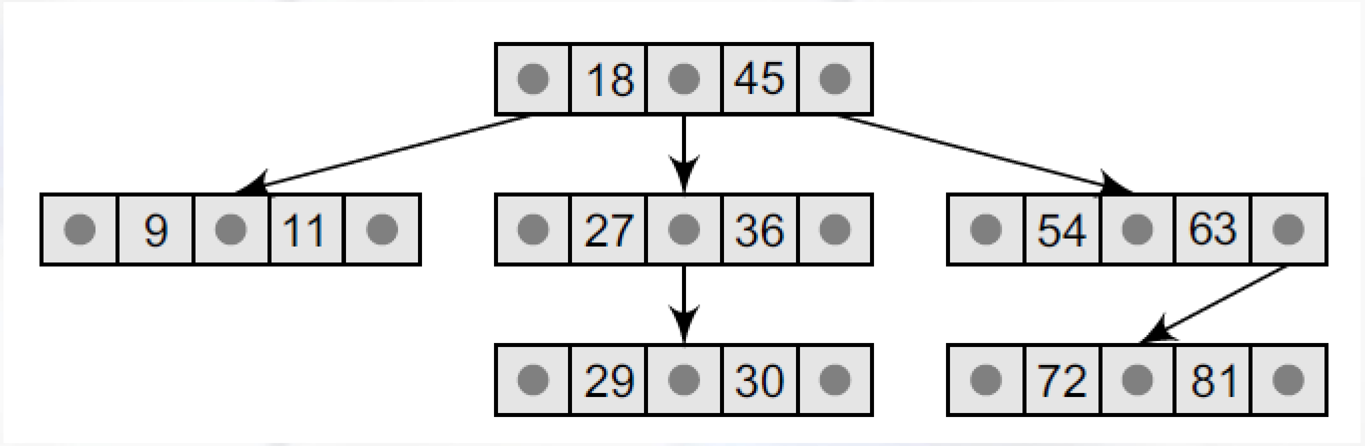
Introduction

- A binary search tree contains **one key value** and **two pointers**, left and right, which point to the node's left and right sub-trees, respectively
- Same concept can be extended to an M-way search tree which has **M subtrees** and **M – 1 key values** per node.
- A BST in that way is a specific case of M-way tree where M = 2.
- An M-way search tree consists of pointers to M sub-trees and contains M – 1 keys, where M > 2.

Structure of an M-way Tree node

P_0	K_0	P_1	K_1	P_2	K_2	....	P_{n-1}	K_{n-1}	P_n

- P_0, P_1, P_2, ..., P_n are pointers to the node's sub-trees and K0, K_1, K_2, ..., K_{n-1} are the key values of the node.
- All the key values are stored in **ascending order**.
- M-way search tree of **order 3** is shown below:



Structure of an M-way Tree node

- In an M-way search tree, it is not compulsory that every node has exactly M–1 key values and M subtrees.
- Rather, the node can have anywhere from 1 to M–1 key values

B-Trees

- A B-tree is a specialized m- way tree that is widely used for disk access.
- A B tree of order **M** can have maximum **M** pointers to its sub-trees and **M-1** keys values.
- Storing a large number of keys in a single node keeps the height of the tree relatively small.
- A B-tree is designed to store data in sorted order and allows search, insert, and delete operations to be performed in logarithmic time.
- Invented by Rudolf Bayer and Edward M. McCreight in Boeing Research Labs
- 'B' in B Tree is never clarified by it's inventors, thought to be *Boeing*, *balanced*, *between*, *broad*, *bushy*, *Bayer* etc.

B-Trees

- A B-tree of **order *M*** (the maximum number of children that each node can have) is a tree with all the properties of an m-way search tree and in addition has the following properties:
  1. Every node in the B-tree has at most ***M*** children.
  2. Every node in the B-tree (except the root node and leaf nodes) will have minimum **CIEL(*M*/2)** children and Maximum **M** children.
  3. Minimum key values will be **CIEL(*M*/2) - 1** and maximum will be **M-1**.
  4. The root node has at least two children if it is not a terminal (leaf) node.
  5. All leaf nodes are at the same level.

B-Trees

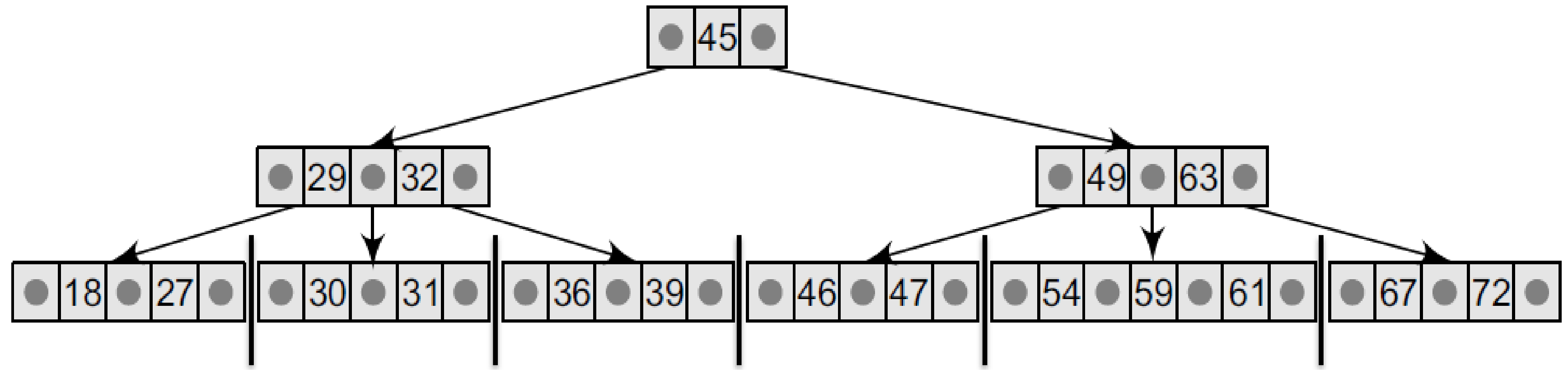
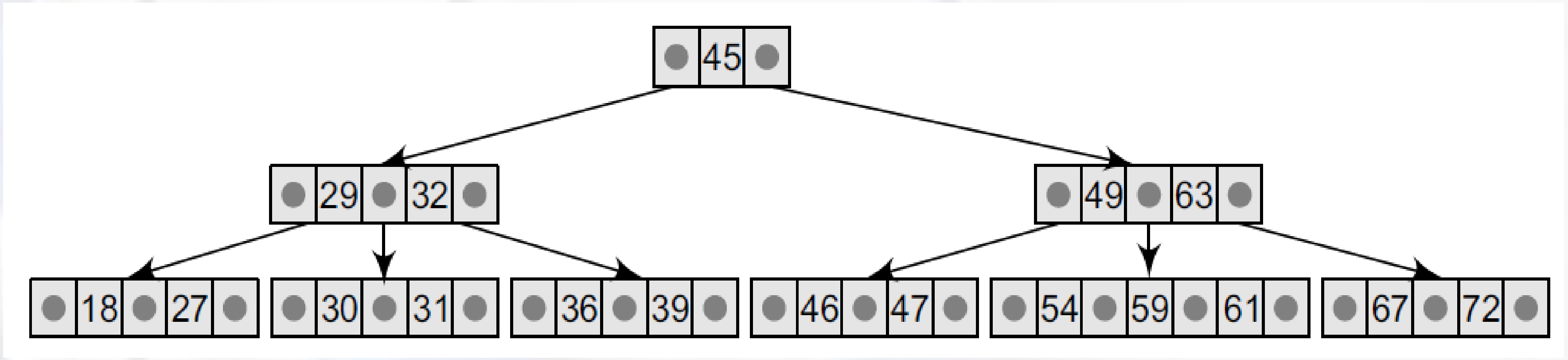


Figure 11.4 B tree of order 4

Searching for an Element in a B Tree

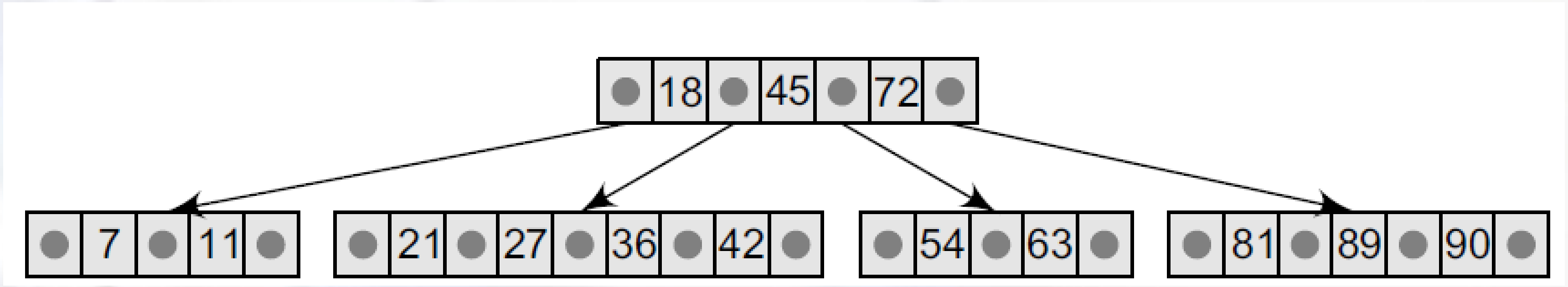
- Searching for an element in a B tree is similar to that in binary search trees.
- Search for 59 and 9



Inserting a New Element in a B Tree

In a B tree, **all insertions are done at the leaf node level**. A new value is inserted in the B tree using the algorithm given below.

Step 1: Search the B tree to find the leaf node where the new key value should be inserted.

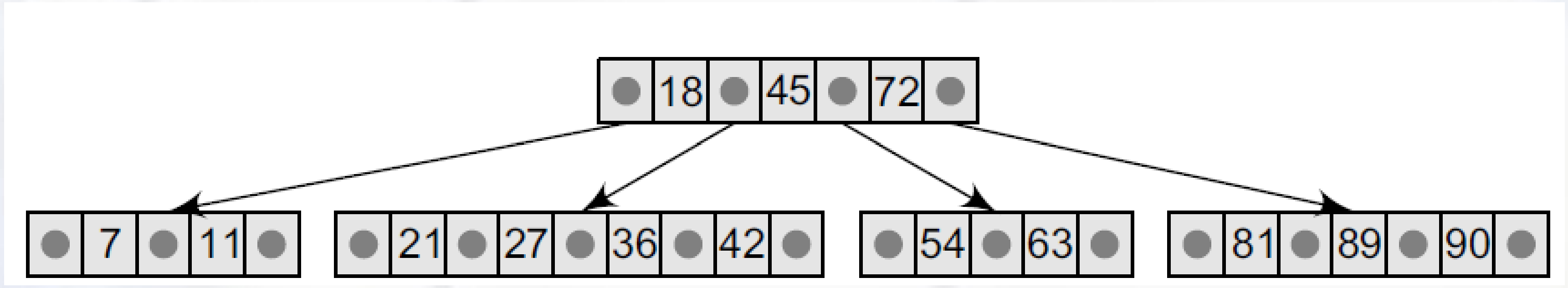


Insert 8

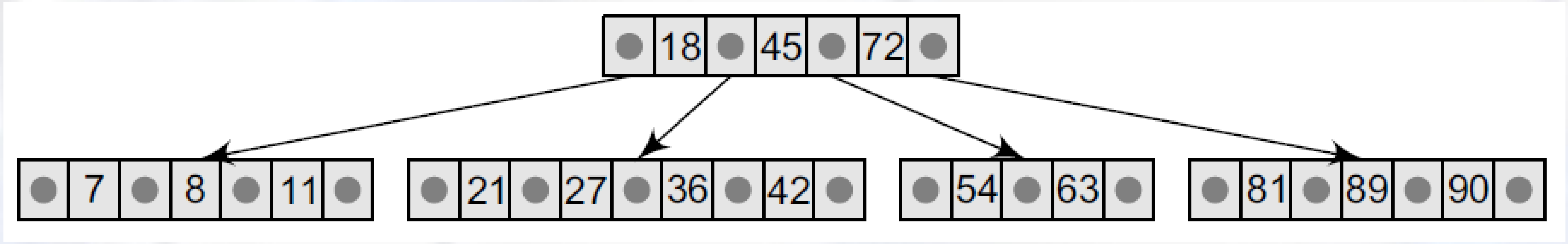
Order M = 5,
Max sub-tree = M = 5,
Min sub-tree= CEIL(M/2) = 3,
Max keys = M-1 = 4,
Min Keys = CEIL(M/2) - 1 = 2

Step 2: If the leaf node is not full, i.e., it contains less than m–1 key values, then insert the new element in the node **keeping the node's elements in sorted order**.

Example:



Insert 8



Inserting a New Element in a B Tree

3. If the leaf node is full, that is, the leaf node already contains m–1 key values, then

   (a) Insert the new value in order into the existing set of keys.

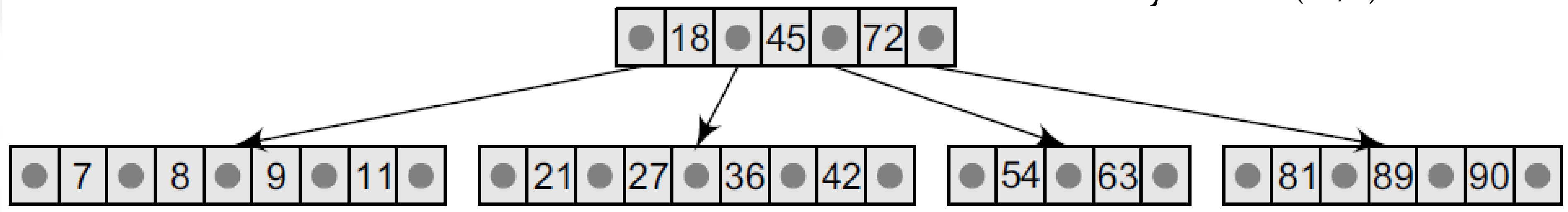
   (b) Split the node at its median into two nodes (note that the split nodes are half full), and

   (c) Push the median element up to its parent's node.

   If the parent's node is already full, then split the parent node by following the same steps

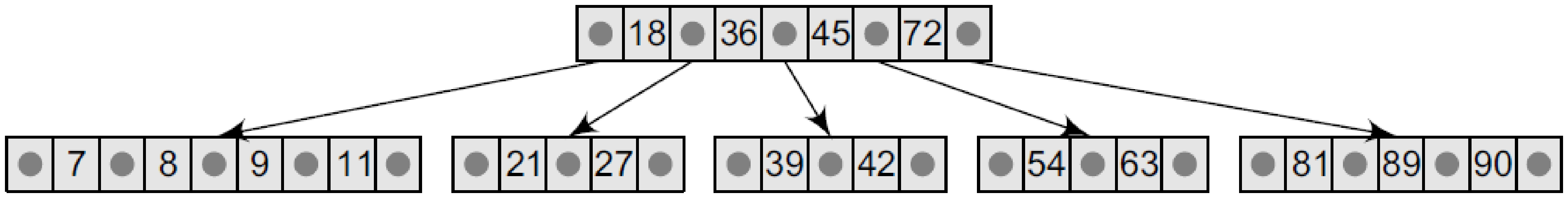
Order M = 5,
Max sub-tree = M = 5,
Min sub-tree= CEIL(M/2) = 3,
Max keys = M-1 = 4,
Min Keys = CEIL(M/2) - 1 = 2

Step 3 Example



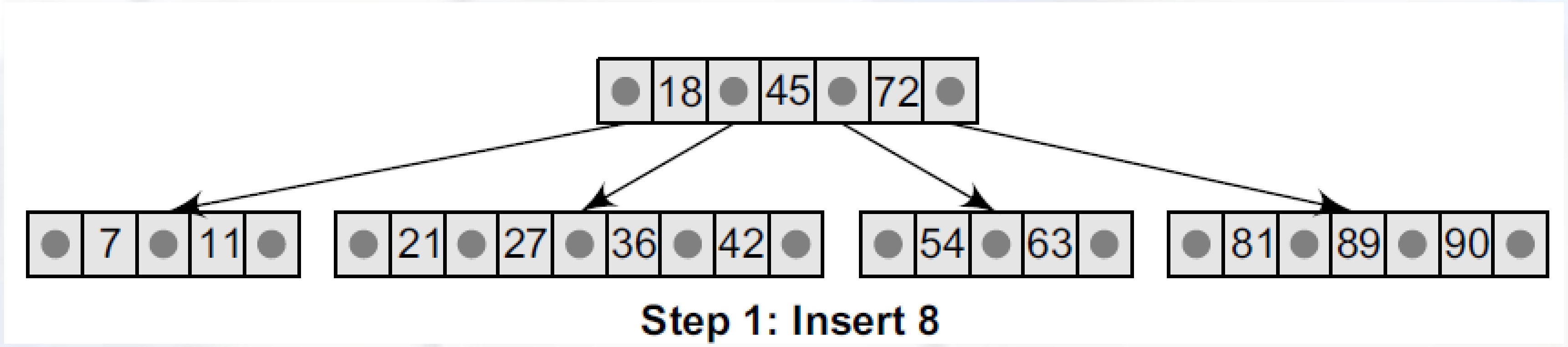
Step 3: Insert 39

- Need to split the second leaf node since it is already full with 4 key values.
- Take the median value of 21,27,**36**,39,42. i.e., 36.
- Move it to parent node.

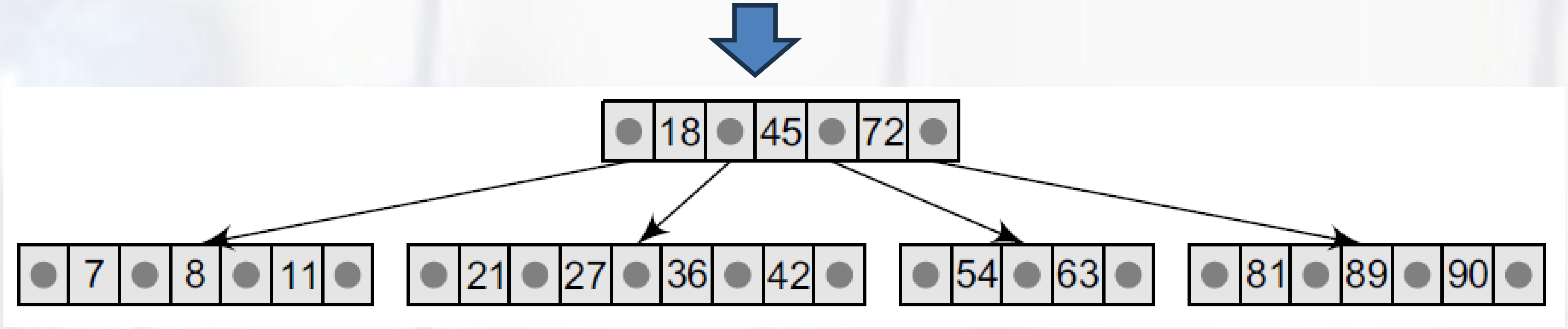


Inserting a New Elements in a B Tree

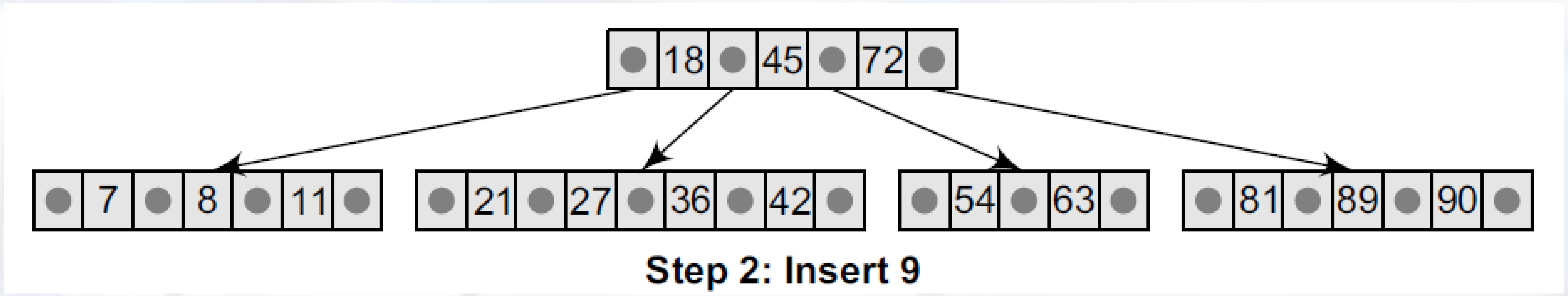
Problem: Consider the B tree of order 5 given below and insert 8, 9, 39, and 4 into it.



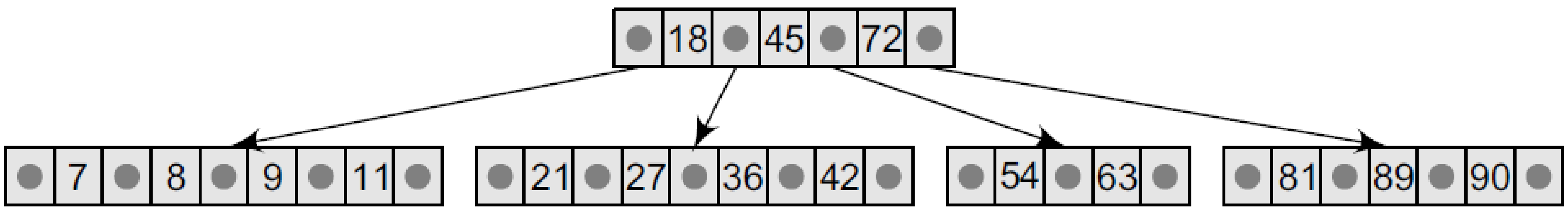
Step 1: Insert 8



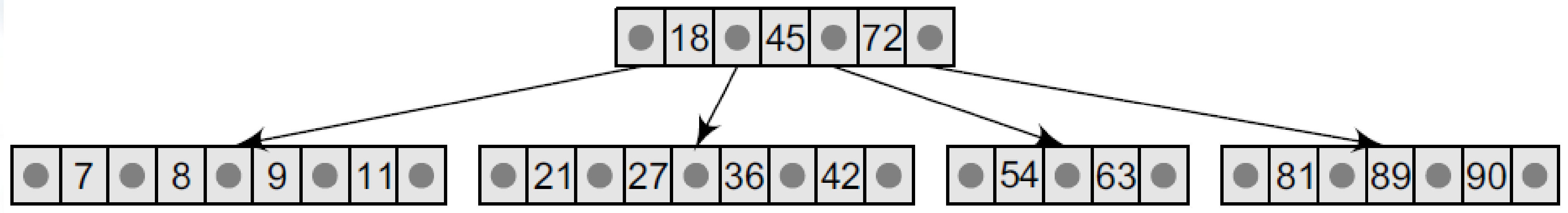
Inserting a New Element in a B Tree



Step 2: Insert 9

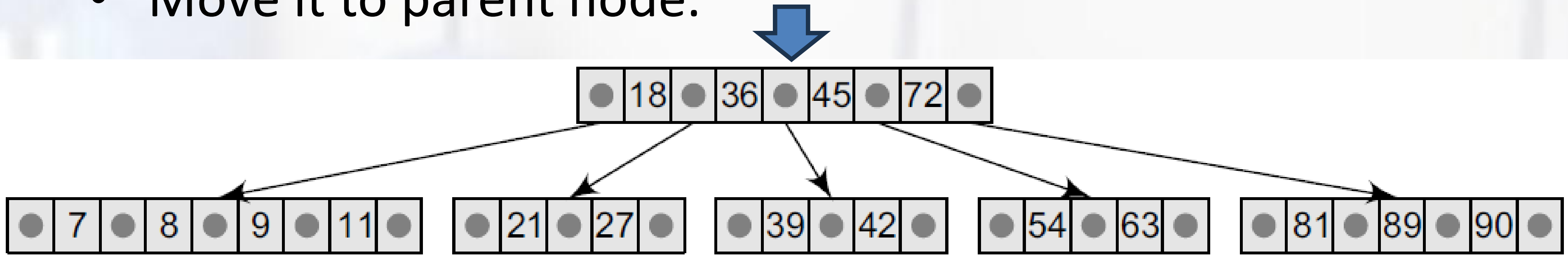


Inserting a New Element in a B Tree

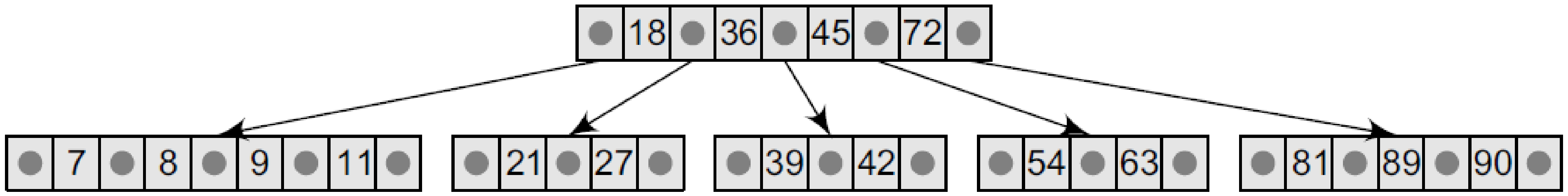


Step 3: Insert 39

- Need to split the second leaf node since it is full already with 4 values.
- Take the median value of 21,27,**36**,39,42. i.e., 36.
- Move it to parent node.



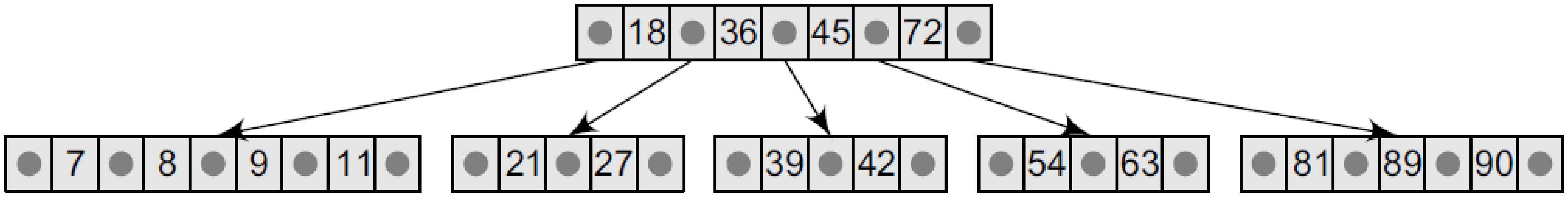
Inserting a New Element in a B Tree



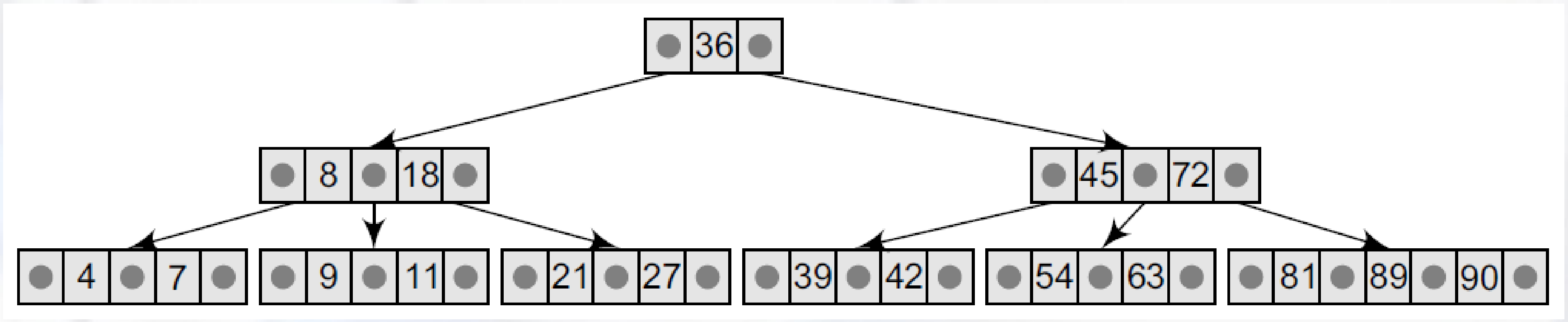
Step 4: Insert 4

- Need to split the first leaf node since it is full already with 4 values.
- Take the median value of 4,7,**8**,9,11,. i.e. 8. Move 8 it to parent node.
- This will now need splitting parent node as well since it is also full.
- Take the median value of 8,18,**36**,45,72,. I.e. 36. Create new parent node(root) and move 36 into it.

Inserting a New Element in a B Tree



Step 4: Insert 4

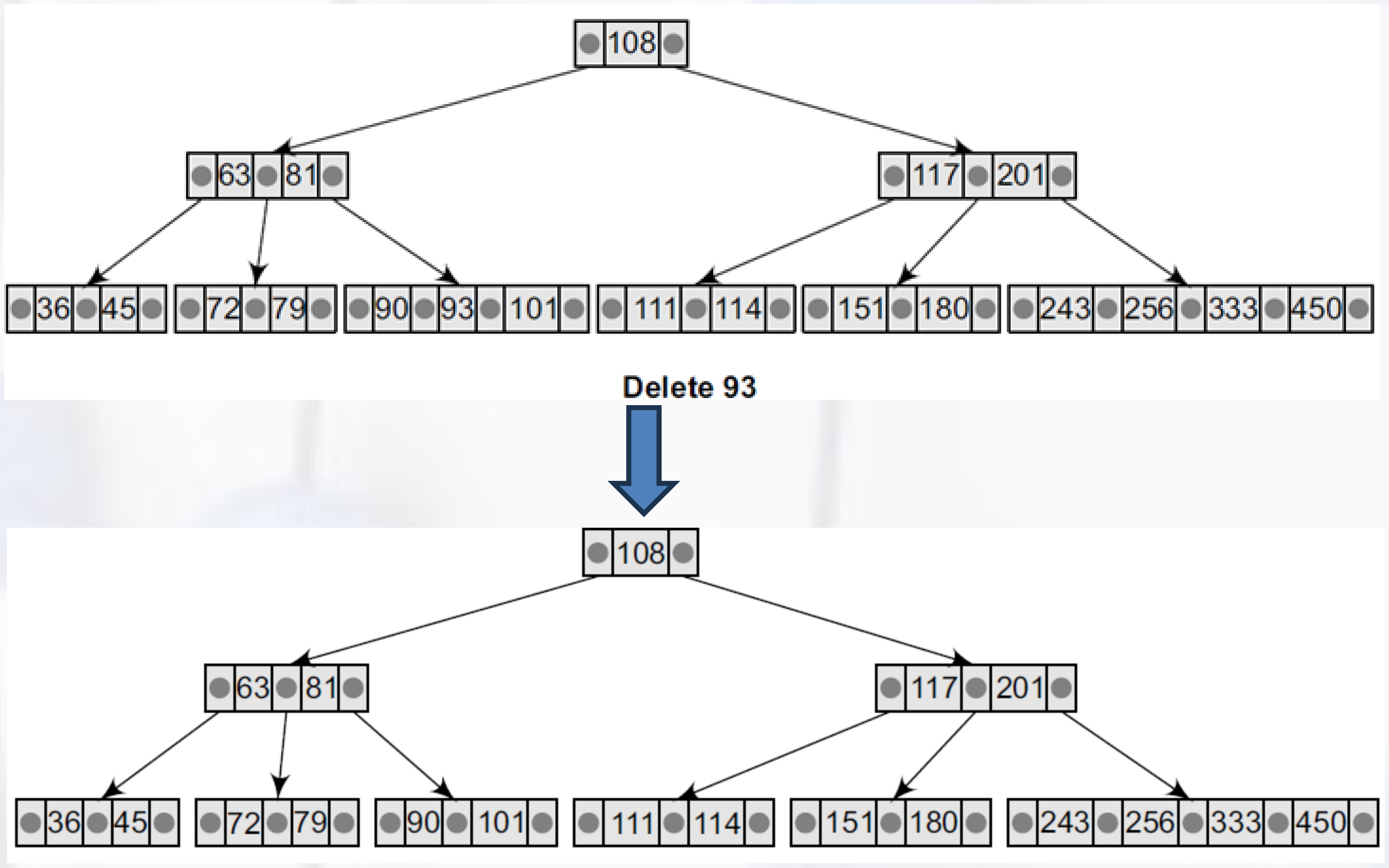


Deleting an Element from a B Tree(**Leaf node**)

Step 1. Locate the leaf node which has to be deleted.

Step 2. If the leaf node contains more than the minimum number of key values, then simply delete the key.

Order M = 5, Max sub-tree = 5, Min sub-tree= CEIL(M/2) = 3, Max keys = M-1 = 4, Min Keys = CEIL(M/2) - 1 = 2



Deleting an Element from a B Tree(**Leaf node**)

Step 3. Else if the leaf node contain minimum required elements, then fill the node by borrowing an element either from the left or right sibling **through the parent**.

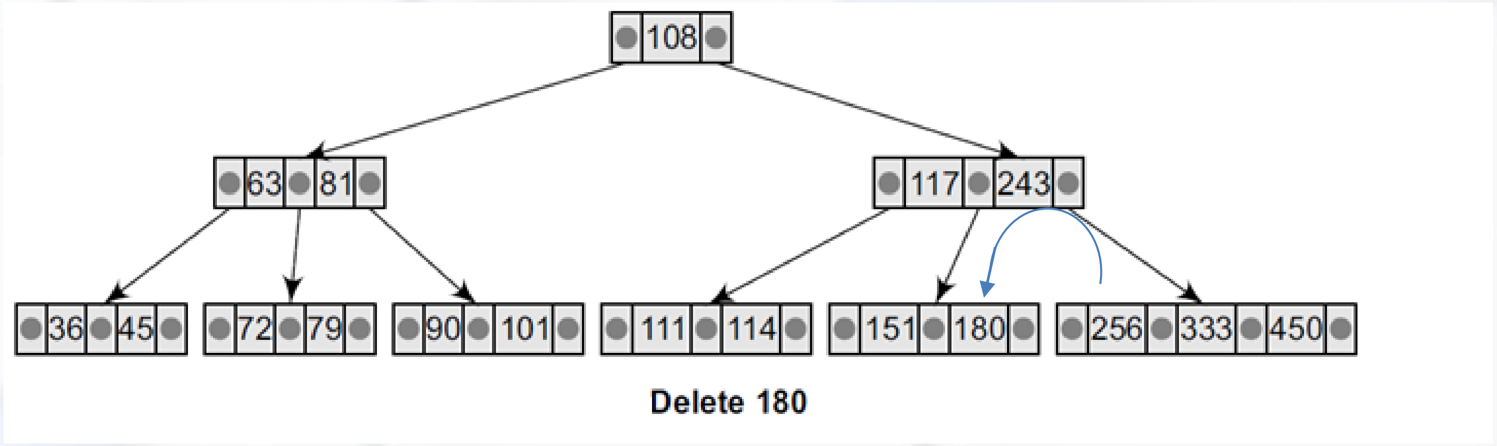
(a) If the left sibling has more than the minimum number of key values: Push its largest key into its parent’s node and pull down the intervening element from the parent node to the leaf node where the key is deleted. (Left biased B Tree)

(b) Else, if the right sibling has more than the minimum number of key values: Push its smallest key into its parent node and pull down the intervening element from the parent node to the leaf node where the key is deleted. (Right biased B Tree)

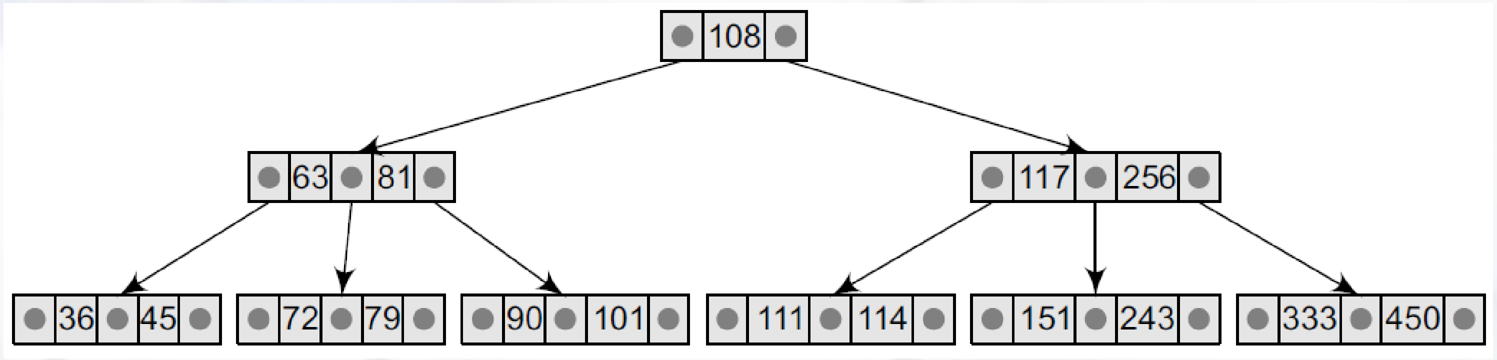
Deleting an Element in a B Tree: Example

- Deleting 180 leaves the node with < Minimum required keys. So, borrow from right sibling **through parent node**.

Order M = 5, Max sub-tree = 5, Min sub-tree= CEIL(M/2) = 3, Max keys = M-1 = 4, Min Keys = CEIL(M/2) - 1 = 2



Apply Algorithm step 3b



Deleting an Element from a B Tree(Leaf node)

Step 3 (c) if both left and right siblings contain only the minimum number of elements.

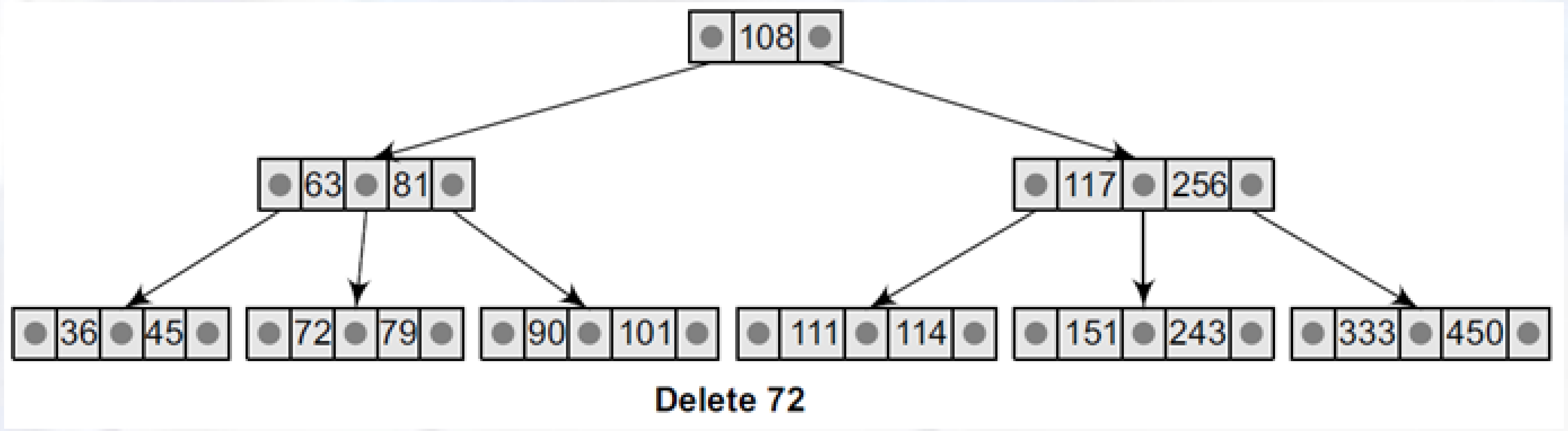
Then, create a new leaf node by combining the **two leaf nodes and the intervening element of the parent node** (ensuring that the number of elements does not exceed the maximum number of elements a node can have, that is, m).

If pulling the intervening element from the parent node leaves it with less than the minimum number of keys in the node, then propagate the process upwards, thereby reducing the height of the B tree.

Deleting an Element in a B Tree: Example

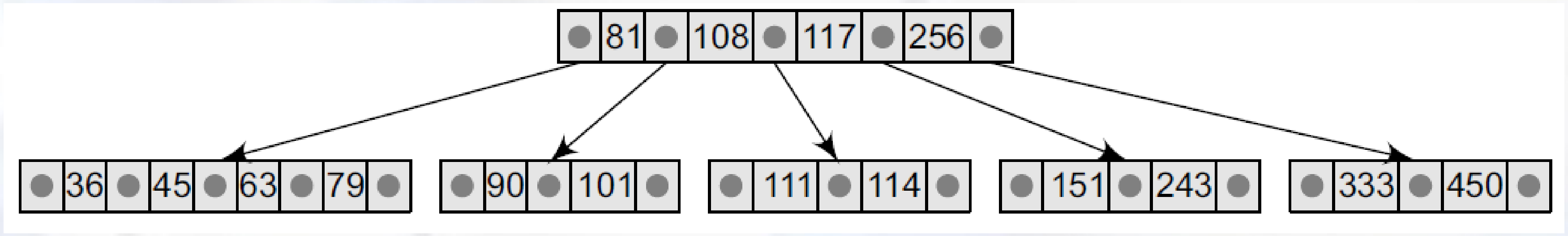
If both left and right siblings contain only the minimum number of elements.

Order M = 5, Max sub-tree = 5, Min sub-tree= CEIL(M/2) = 3, Max keys = M-1 = 4, Min Keys = CEIL(M/2) - 1 = 2



Delete 72

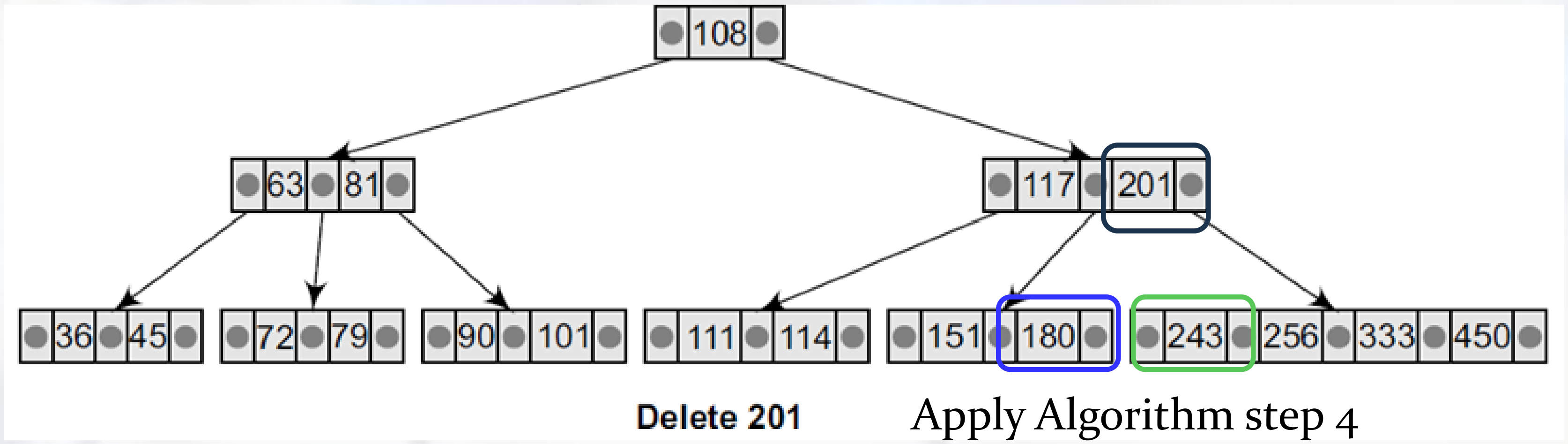
Apply Algorithm step 3(c)



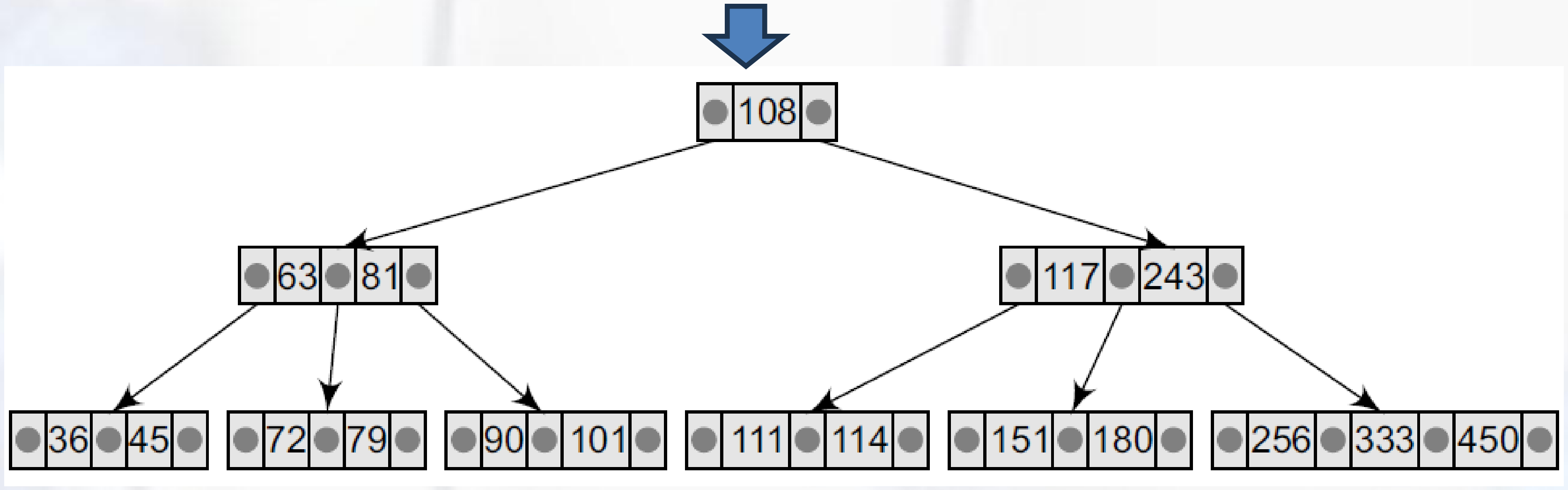
Deleting an Element from a B Tree(Internal node)

Step 4. To delete an internal node, promote the **inorder successor** or **inorder predecessor** of the key to be deleted to occupy the position of the deleted key.

Order M = 5, Max sub-tree = 5, Min sub-tree= CEIL(M/2) = 3, Max keys = M-1 = 4, Min Keys = CEIL(M/2) - 1 = 2



Apply Algorithm step 4



Deleting an Element from a B Tree(**Leaf node**)

1. Locate the leaf node which has to be deleted.

2. If the leaf node contains more than the minimum number of key values (i.e., more than CEIL(M/2)-1 elements), then simply delete the value.

3. Else if the leaf node contain minimum required elements, then fill the node by taking an element either from the left or from the right sibling **through the parent**.

(a) If the left sibling has more than the minimum number of key values, push its largest key into its parent’s node and pull down the intervening element from the parent node to the leaf node where the key is deleted. (Left biased B Tree)

(b) Else, if the right sibling has more than the minimum number of key values, push its smallest key into its parent node and pull down the intervening element from the parent node to the leaf node where the key is deleted. (Right biased B Tree)

Deleting an Element from a B Tree(Leaf node)

(c) Else, if both left and right siblings contain only the minimum number of elements, then create a new leaf node by combining the **two leaf nodes and the intervening element of the parent node** (ensuring that the number of elements does not exceed the maximum number of elements a node can have, that is, m). If pulling the intervening element from the parent node leaves it with less than the minimum number of keys in the node, then propagate the process upwards, thereby reducing the height of the B tree.

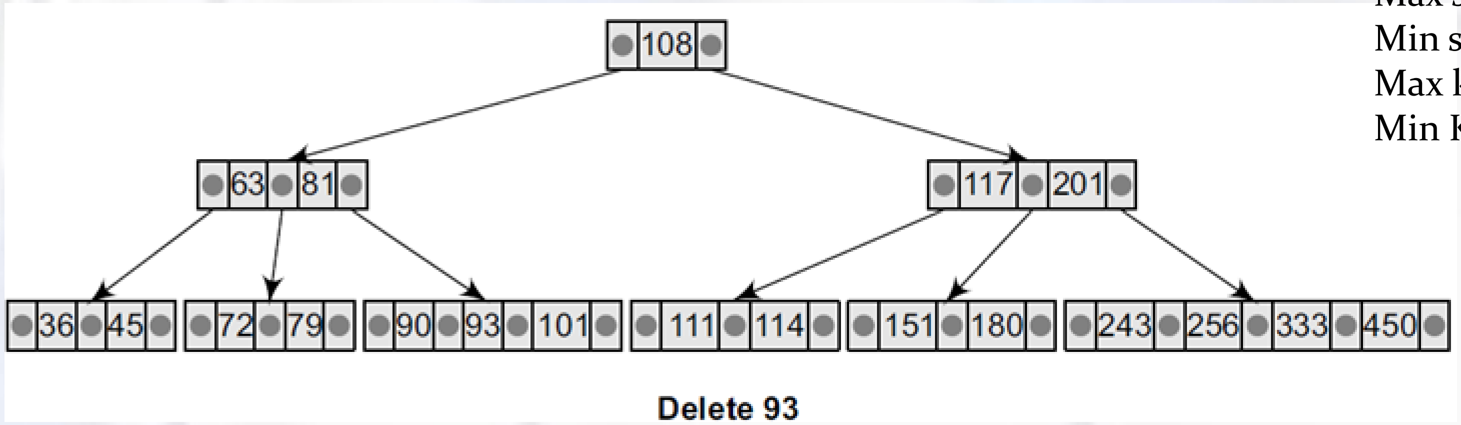
Deleting an Element from a B Tree(Internal node)

4. To delete an internal node, promote the **inorder successor** or **inorder predecessor** of the key to be deleted to occupy the position of the deleted key.

Deleting an Element in a B Tree: Example

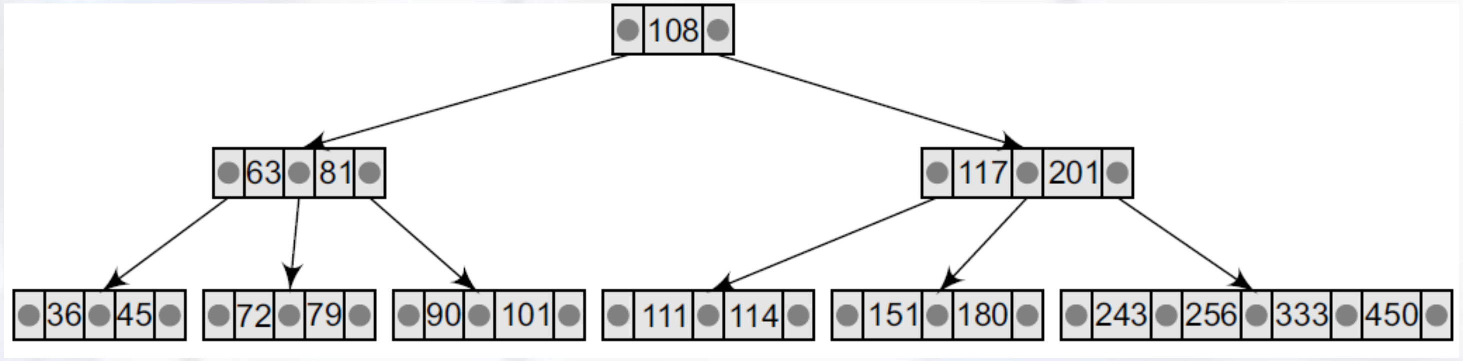
- Consider the following B tree of order 5 and delete values 93, 201, 180, and 72 from it

Order M = 5,
Max sub-tree = M = 5,
Min sub-tree= CEIL(M/2) = 3,
Max keys = M-1 = 4,
Min Keys = CEIL(M/2) - 1 = 2



Delete 93

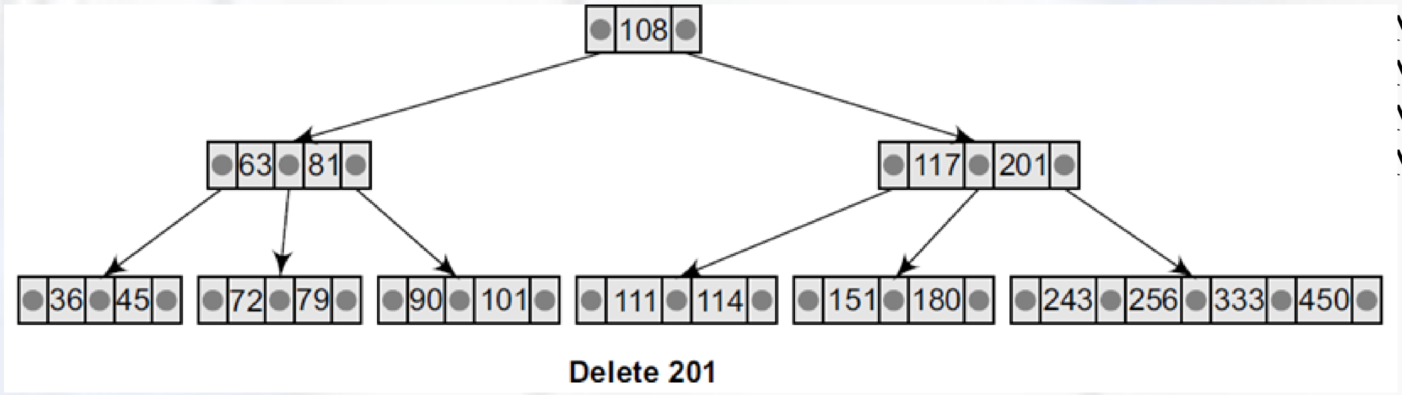
Apply Algorithm step 2



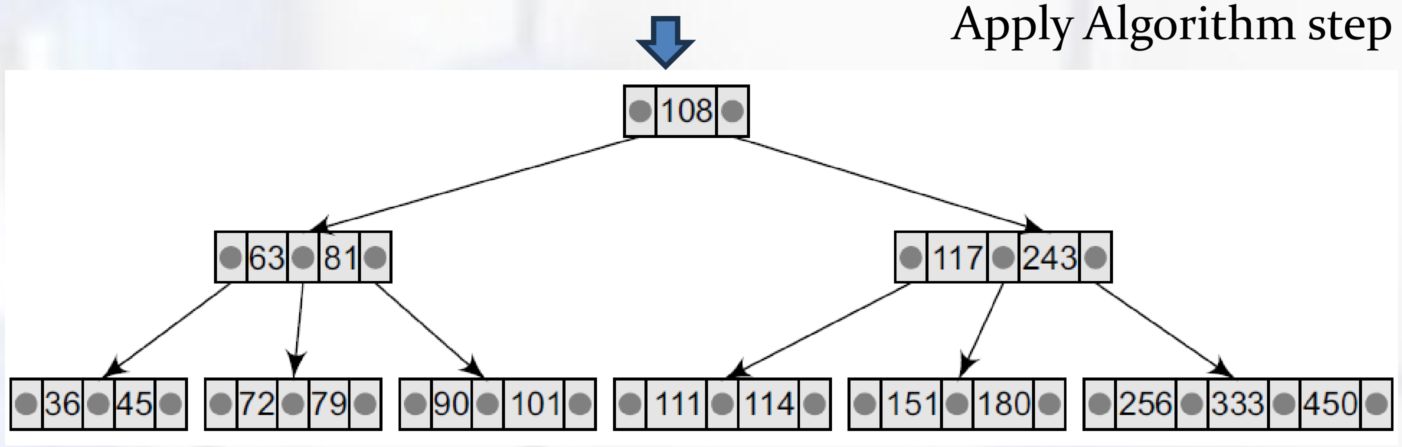
Deleting an Element in a B Tree: Example

- Consider the following B tree of order 5 and delete values 93, 201, 180, and 72 from it
- Use **Inorder successor**(243) to replace 201.

Order M = 5,
Max sub-tree = M = 5,
Min sub-tree= CEIL(M/2) = 3,
Max keys = M-1 = 4,
Min Keys = CEIL(M/2) - 1 = 2

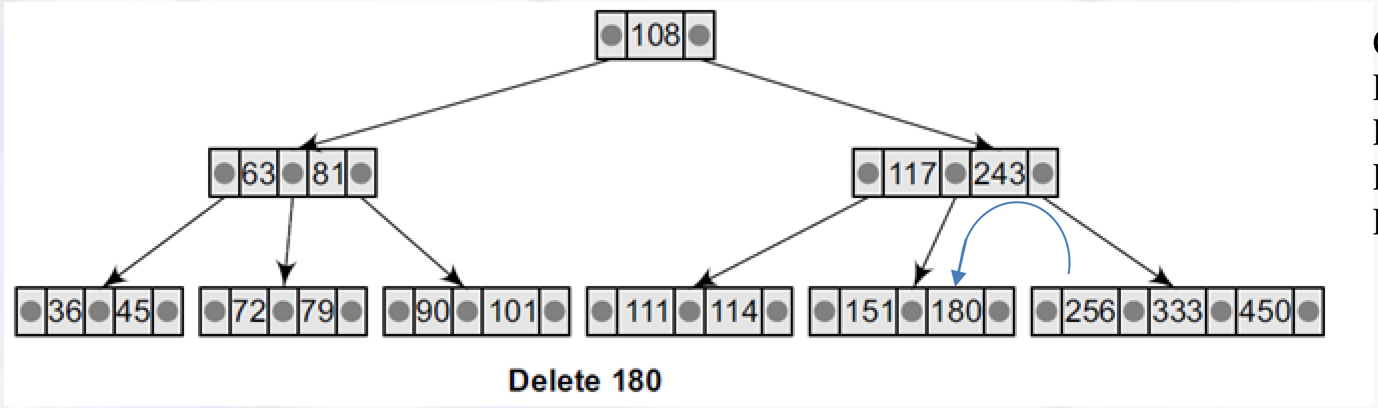


Apply Algorithm step 4



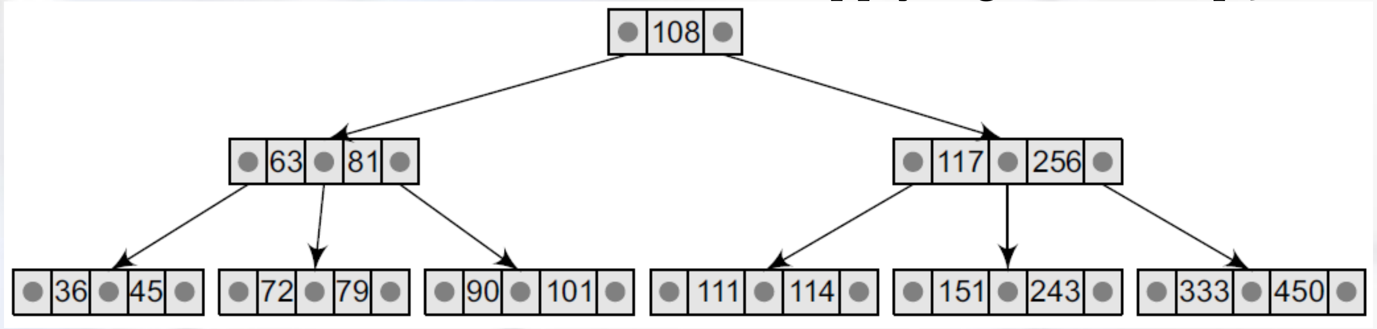
Deleting an Element in a B Tree: Example

- Consider the following B tree of order 5 and delete values 93, 201, 180, and 72 from it
- Deleting 180 leaves the node with < Minimum required keys. So, borrow from right sibling **through parent node**.



Order M = 5,
Max sub-tree = M = 5,
Min sub-tree= CEIL(M/2) = 3,
Max keys = M-1 = 4,
Min Keys = CEIL(M/2) - 1 = 2

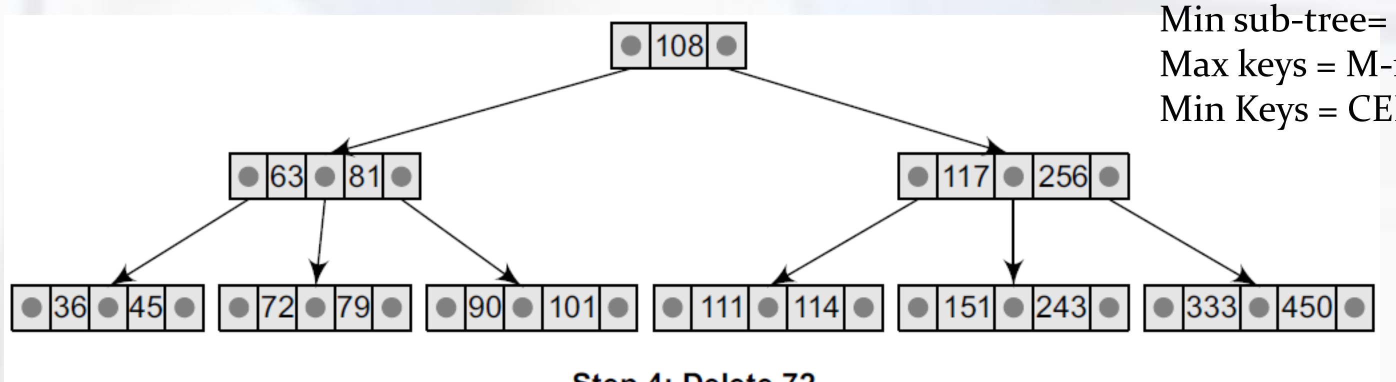
Apply Algorithm step 3b



Deleting an Element in a B Tree: Example

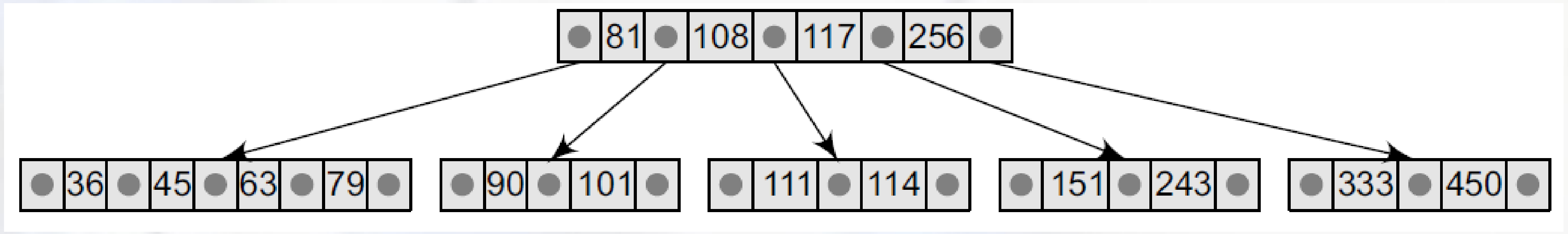
- Consider the following B tree of order 5 and delete values 93, 201, 180, and 72 from it

Order M = 5,
Max sub-tree = M = 5,
Min sub-tree= CEIL(M/2) = 3,
Max keys = M-1 = 4,
Min Keys = CEIL(M/2) - 1 = 2



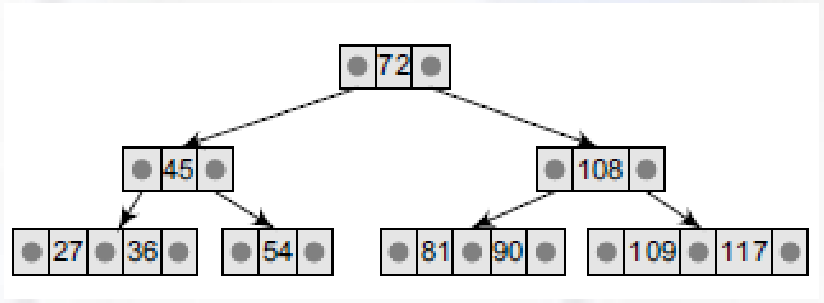
Step 4: Delete 72

Apply Algorithm step 3(c)

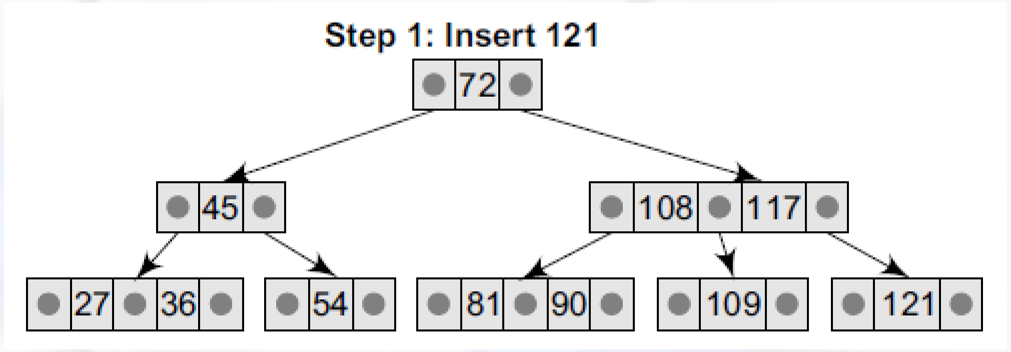


Operations on B Tree-Example

- **Example 11.3** Consider the B tree of order 3 given below and perform the following operations: (a) insert 121, 87 and then (b) delete 36, 109.

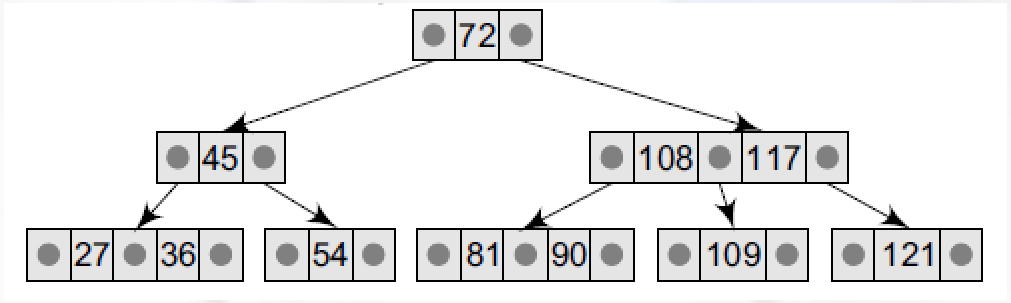


Order M = 3,
Max sub-tree = M = 3,
Min sub-tree= CEIL(M/2) = 2,
Max keys = M-1 = 2,
Min Keys = CEIL(M/2) - 1 = 1

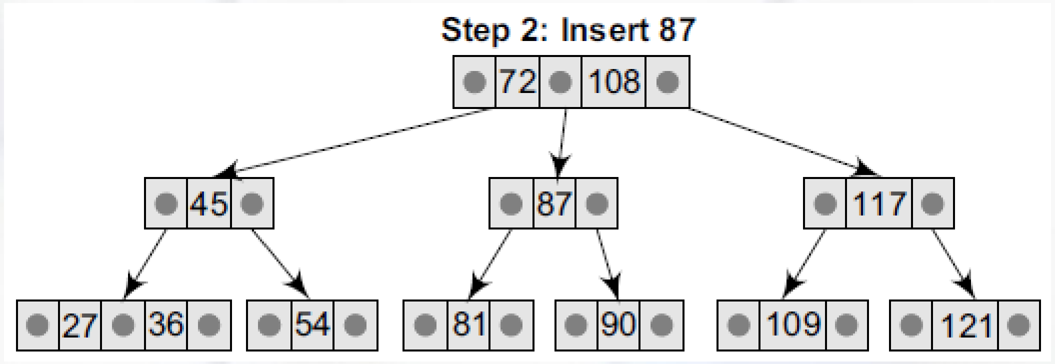


Operations on B Tree-Example

- **Example 11.3** Consider the B tree of order 3 given below and perform the following operations: (a) insert 121, 87 and then (b) delete 36, 109.

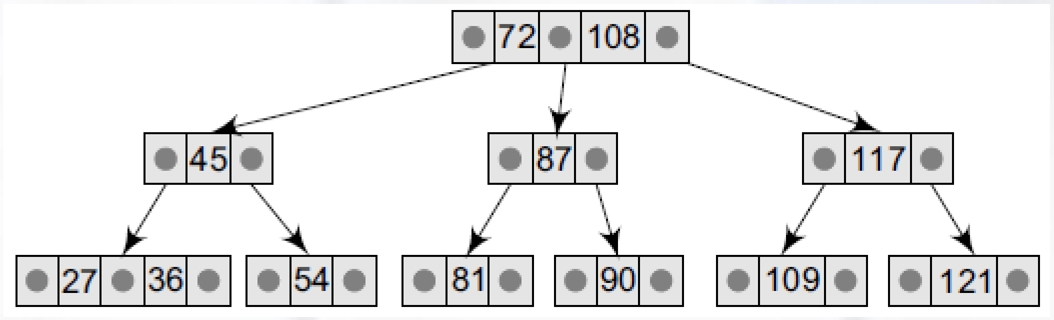


Order M = 3,
Max sub-tree = M = 3,
Min sub-tree= CEIL(M/2) = 2,
Max keys = M-1 = 2,
Min Keys = CEIL(M/2) - 1 = 1

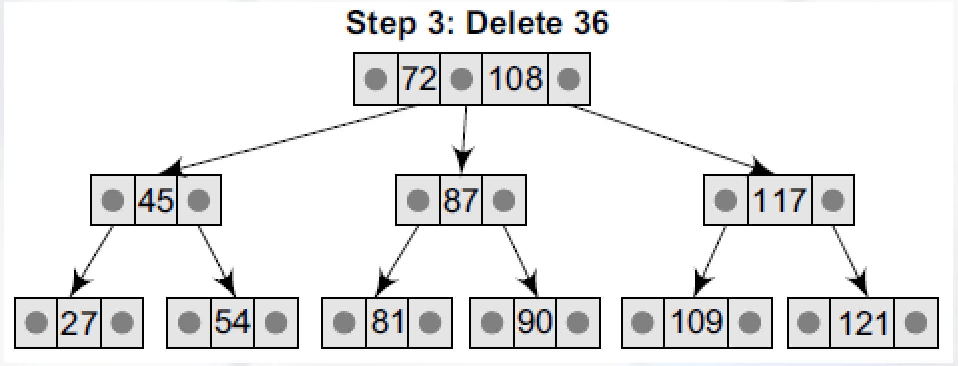


Operations on B Tree-Example

- **Example 11.3** Consider the B tree of order 3 given below and perform the following operations: (a) insert 121, 87 and then (b) delete 36, 109.



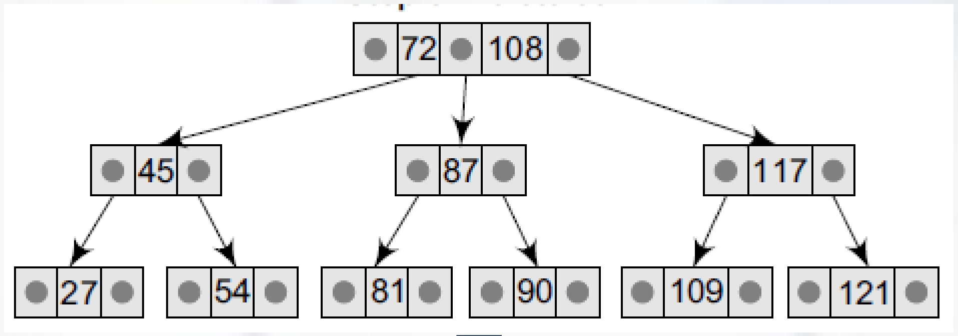
Order M = 3,
Max sub-tree = M = 3,
Min sub-tree= CEIL(M/2) = 2,
Max keys = M-1 = 2,
Min Keys = CEIL(M/2) - 1 = 1

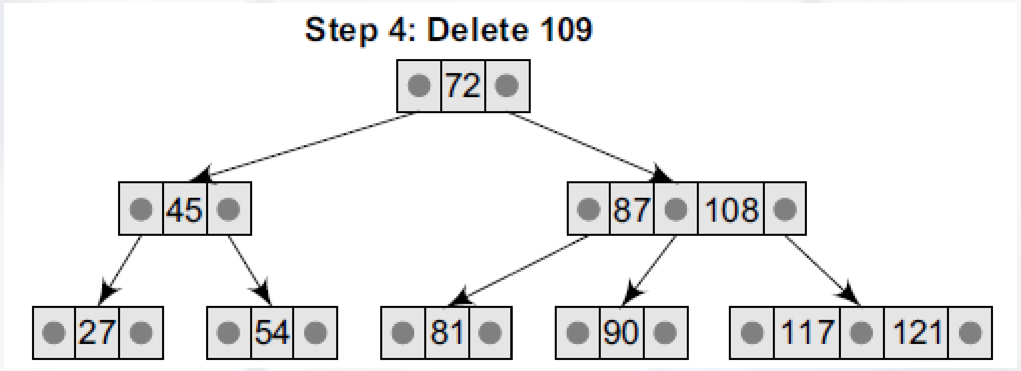


Operations on B Tree-Example

- **Example 11.3** Consider the B tree of order 3 given below and perform the following operations: (a) insert 121, 87 and then (b) delete 36, **109**.

Order M = 3,
Max sub-tree = M = 3,
Min sub-tree= CEIL(M/2) = 2,
Max keys = M-1 = 2,
Min Keys = CEIL(M/2) - 1 = 1





Creation of B Tree

- Example 11.4 Create a B tree of order 5 by inserting the following elements: 3, 14, 7, 1, 8, 5, 11, 17, 13, 6, 23, 12, 20, 26, 4, 16, 18, 24, 25, and 19.

Order M = 5:

Max sub-tree = M = 5

Min sub-tree= CEIL(M/2) = 3

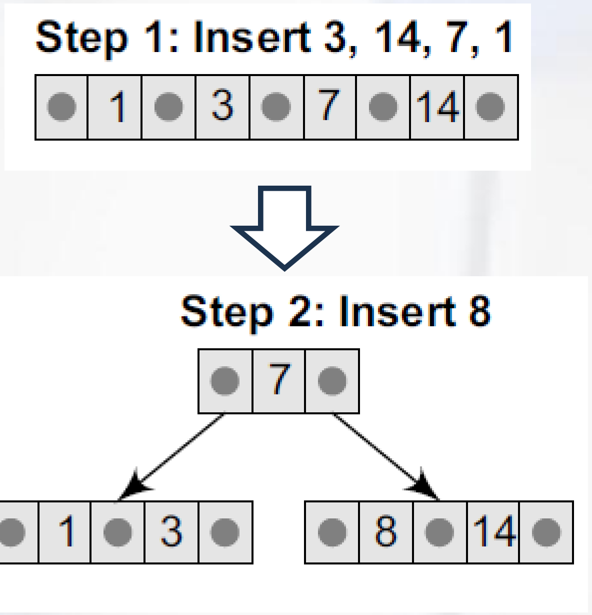
Max keys = M-1 = 4

Min Keys = CEIL(M/2) - 1 = 2

Creation of B Tree

- Example 11.4 Create a B tree of order 5 by inserting the following elements: 3, 14, 7, 1, **8**, 5, 11, 17, 13, 6, 23, 12, 20, 26, 4, 16, 18, 24, 25, and 19.

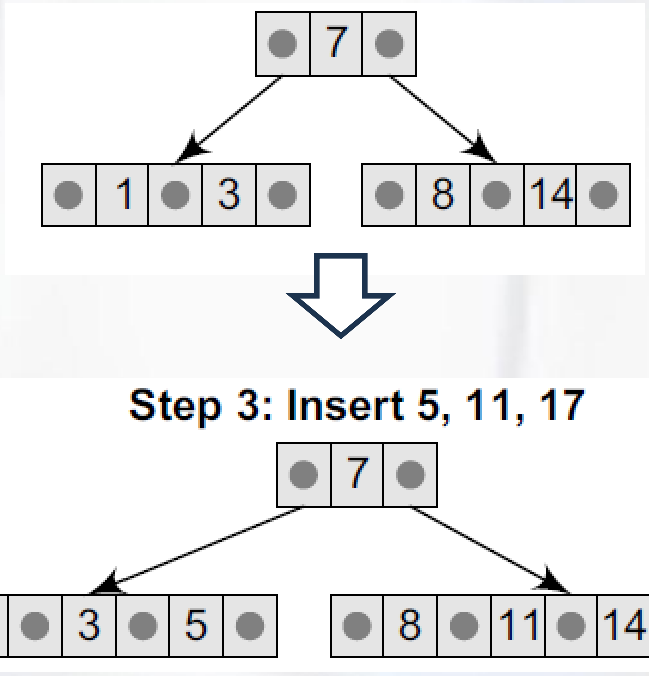
Order M = 5, Max sub-tree = M = 5, Min sub-tree= CEIL(M/2) = 3, Max keys = M-1 = 4, Min Keys = CEIL(M/2) - 1 = 2



Creation of B Tree

- Example 11.4 Create a B tree of order M = 5 by inserting the following elements: 3, 14, 7, 1, 8, <u>5, 11, 17</u>, 13, 6, 23, 12, 20, 26, 4, 16, 18, 24, 25, and 19.

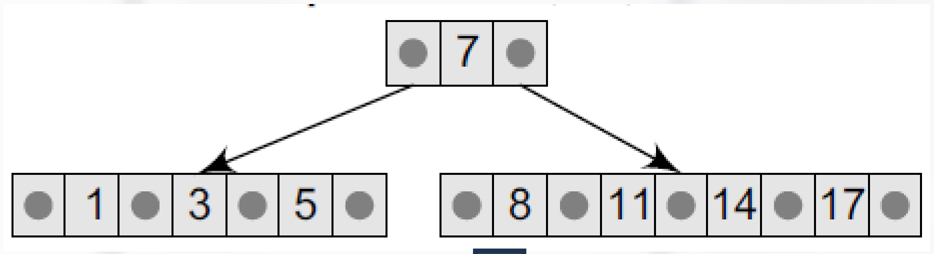
Order M = 5, Max sub-tree = M = 5, Min sub-tree= CEIL(M/2) = 3, Max keys = M-1 = 4, Min Keys = CEIL(M/2) - 1 = 2

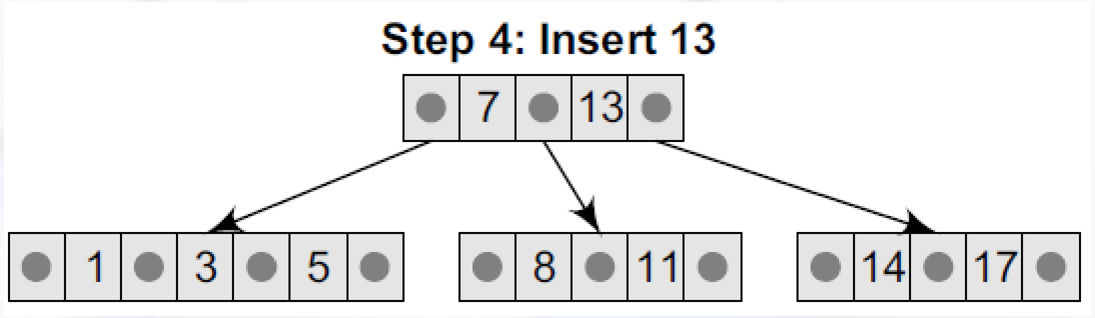


Creation of B Tree

- Example 11.4 Create a B tree of order M = 5 by inserting the following elements: 3, 14, 7, 1, 8, 5, 11, 17, 13, 6, 23, 12, 20, 26, 4, 16, 18, 24, 25, and 19.

Order M = 5, Max sub-tree = M = 5, Min sub-tree= CEIL(M/2) = 3, Max keys = M-1 = 4, Min Keys = CEIL(M/2) - 1 = 2

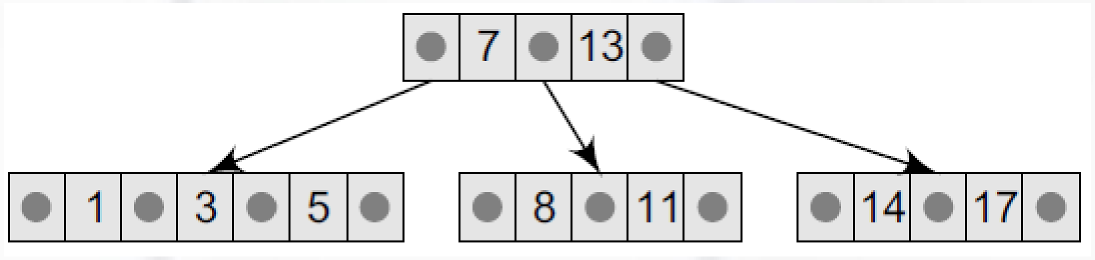


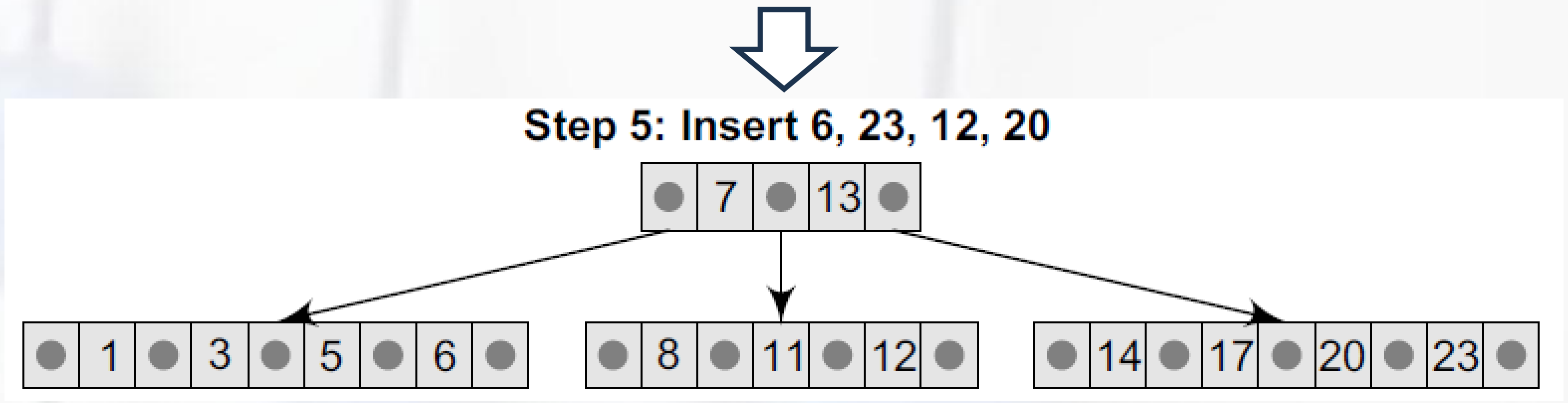


Creation of B Tree

- Example 11.4 Create a B tree of order M = 5 by inserting the following elements: 3, 14, 7, 1, 8, 5, 11, 17, 13, 6, 23, 12, 20, 26, 4, 16, 18, 24, 25, and 19.

Order M = 5, Max sub-tree = M = 5, Min sub-tree= CEIL(M/2) = 3, Max keys = M-1 = 4, Min Keys = CEIL(M/2) - 1 = 2

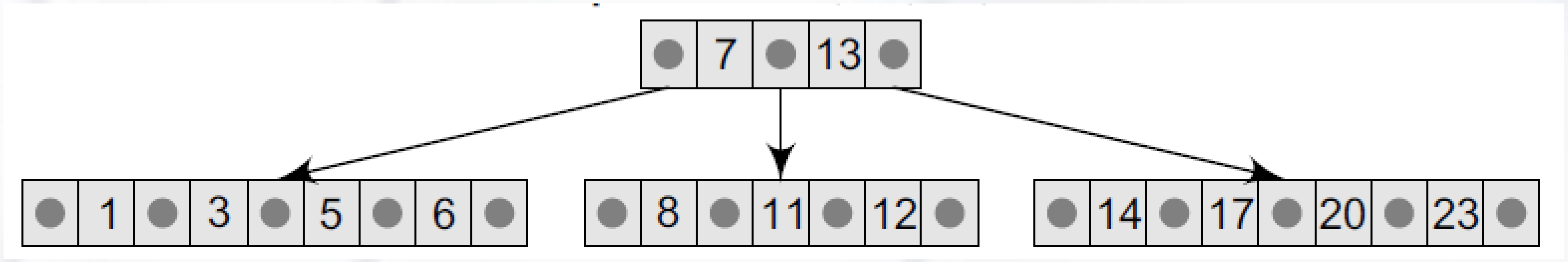


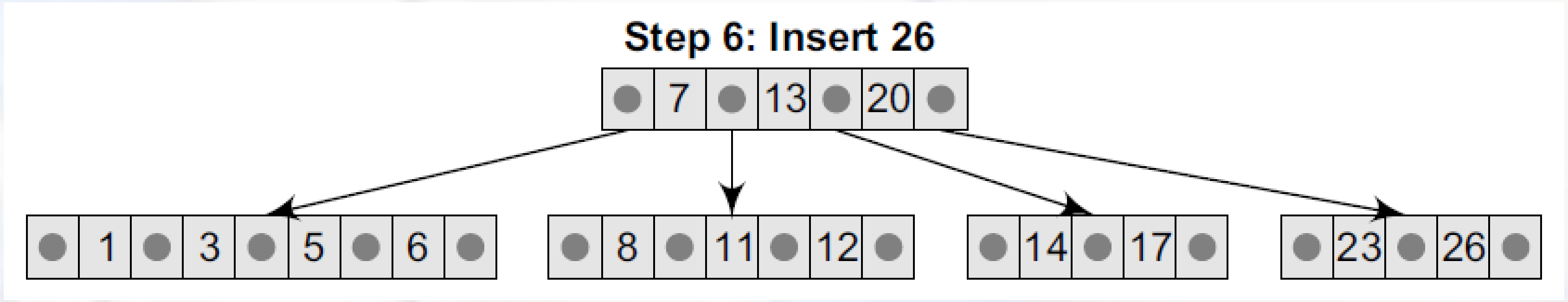


Creation of B Tree

- Example 11.4 Create a B tree of order M = 5 by inserting the following elements: 3, 14, 7, 1, 8, 5, 11, 17, 13, 6, 23, 12, 20, 26, 4, 16, 18, 24, 25, and 19.

Order M = 5, Max sub-tree = M = 5, Min sub-tree= CEIL(M/2) = 3, Max keys = M-1 = 4, Min Keys = CEIL(M/2) - 1 = 2

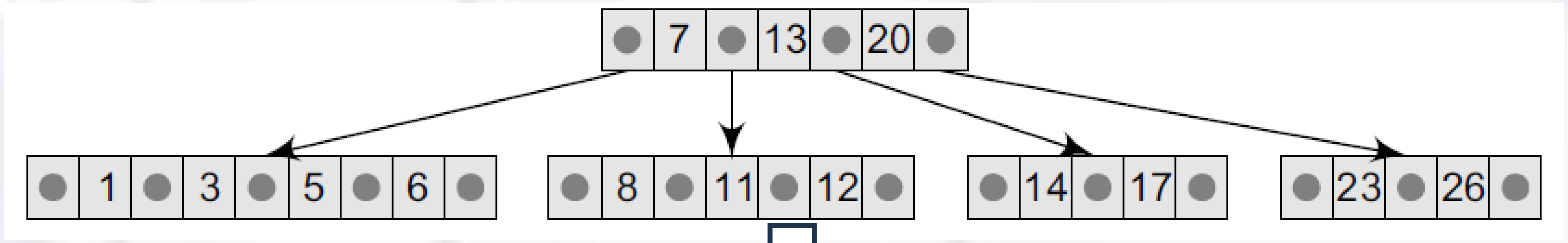


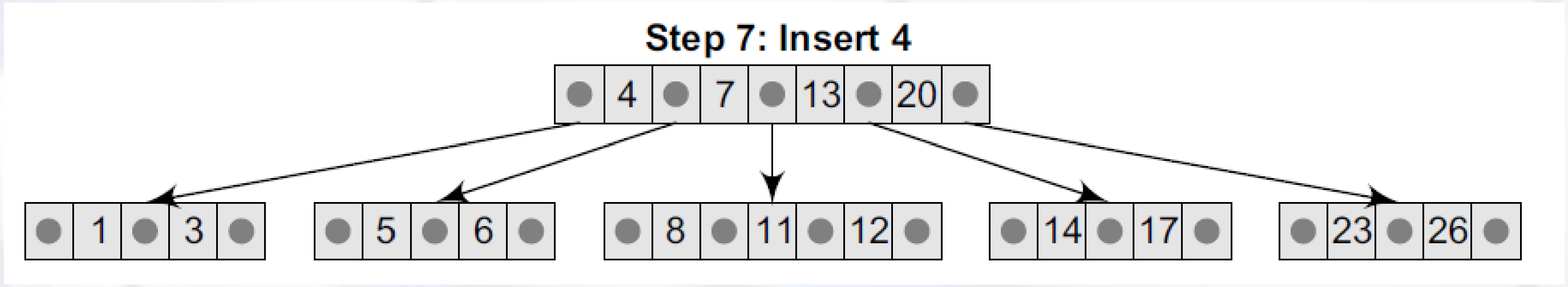


Creation of B Tree

- Example 11.4 Create a B tree of order M = 5 by inserting the following elements: 3, 14, 7, 1, 8, 5, 11, 17, 13, 6, 23, 12, 20, 26, <u>4</u>, 16, 18, 24, 25, and 19.

Order M = 5, Max sub-tree = M = 5, Min sub-tree= CEIL(M/2) = 3, Max keys = M-1 = 4, Min Keys = CEIL(M/2) - 1 = 2

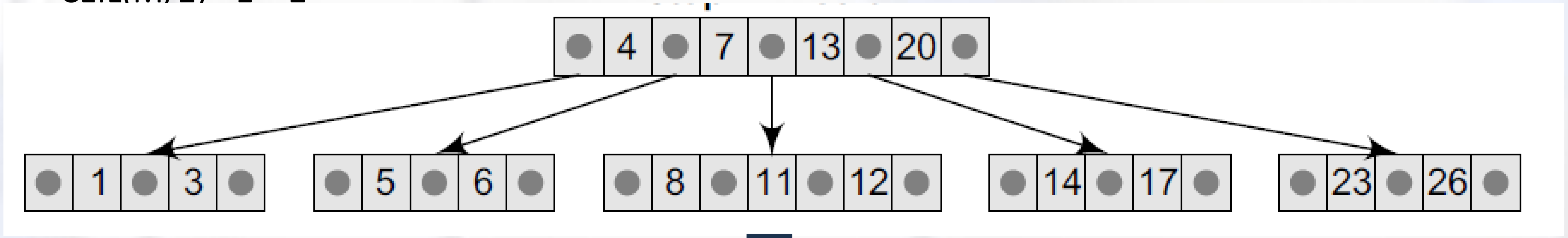




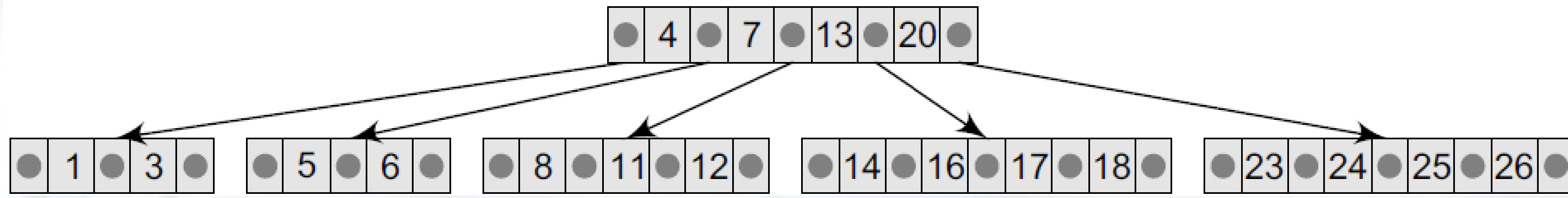
Creation of B Tree

- Example 11.4 Create a B tree of order M = 5 by inserting the following elements: 3, 14, 7, 1, 8, 5, 11, 17, 13, 6, 23, 12, 20, 26, 4, 16, 18, 24, 25, and 19.

Order M = 5, Max sub-tree = M = 5, Min sub-tree= CEIL(M/2) = 3, Max keys = M-1 = 4, Min Keys = CEIL(M/2) - 1 = 2



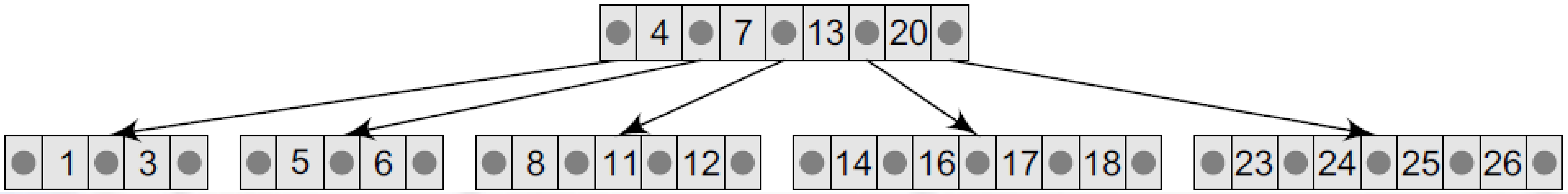
Step 8: Insert 16, 18, 24, 25



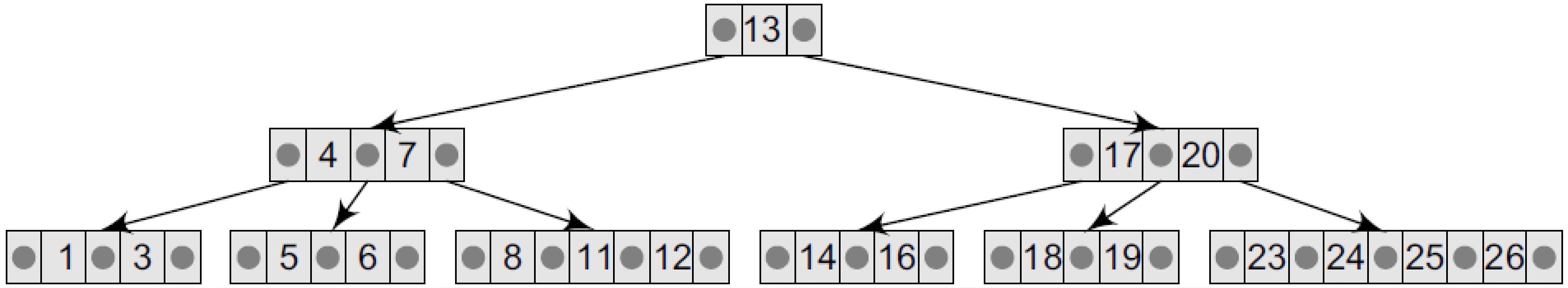
Creation of B Tree

- Example 11.4 Create a B tree of order M = 5 by inserting the following elements: 3, 14, 7, 1, 8, 5, 11, 17, 13, 6, 23, 12, 20, 26, 4, 16, 18, 24, 25, and **19**.

Order M = 5, Max sub-tree = M = 5, Min sub-tree= CEIL(M/2) = 3, Max keys = M-1 = 4, Min Keys = CEIL(M/2) - 1 = 2



Step 9: Insert 19

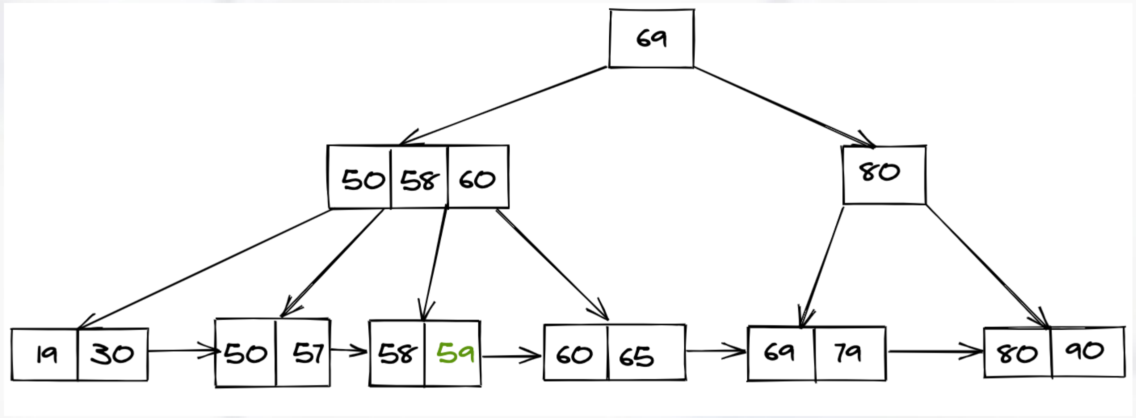


Application of B-Trees

- Storage and access to data in secondary storage in Database Management Systems through B-Tree based indexing.

B+ Trees

- A B+ tree is a variant of a B tree which stores sorted data in a way that allows for efficient insertion, retrieval, and removal of records, each of which is identified by a key.
- B tree can store both keys and records in its interior nodes, a B+ tree, in contrast, **stores all the records at the leaf level of the tree**; only **keys/index are stored in the interior nodes**.
- Leaf nodes of a **B+ tree** are often linked to one another in a linked list. This has an added advantage of making the algorithms simpler and more efficient.



B+ Trees

- B+ trees are used to store large amounts of data that cannot be stored in the main memory.
- With B+ trees, the secondary storage is used to store the leaf nodes of trees and the internal nodes of trees are stored in the main memory.
- B+ trees store data only in the leaf nodes. All other nodes (internal nodes) are called index nodes or i-nodes and store index values.
- Many database systems are implemented using B+ tree structure because of its simplicity.

Comparison Between B Trees and B+ Trees

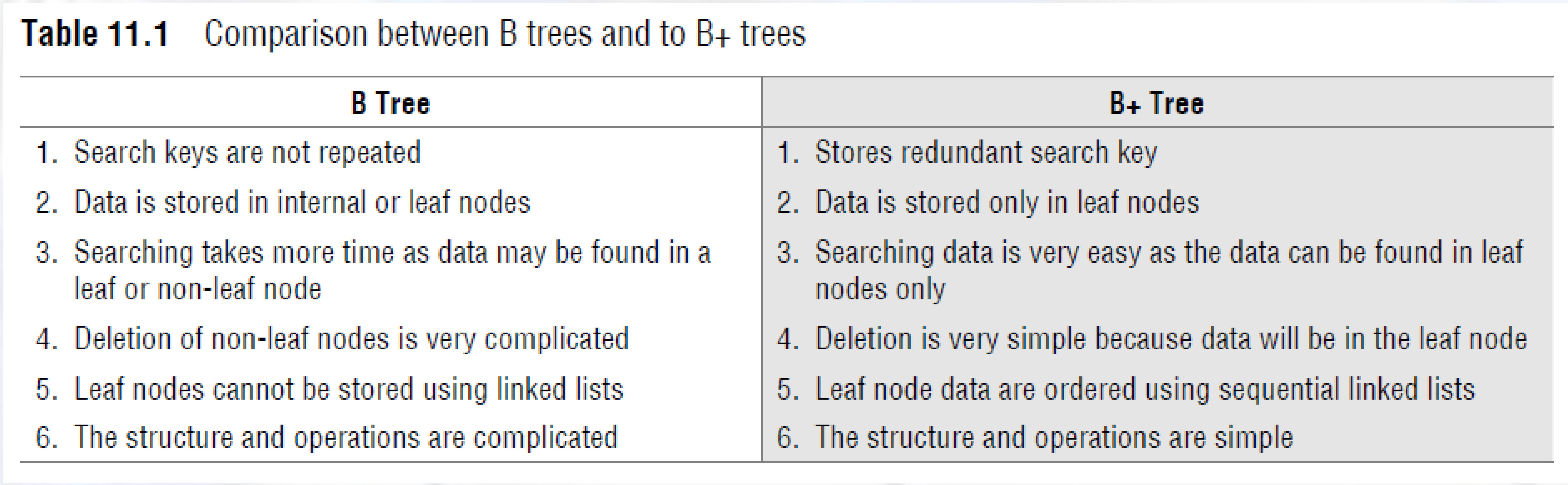


Table 11.1 Comparison between B trees and to B+ trees

B Tree	B+ Tree
1. Search keys are not repeated	1. Stores redundant search key
2. Data is stored in internal or leaf nodes	2. Data is stored only in leaf nodes
3. Searching takes more time as data may be found in a leaf or non-leaf node	3. Searching data is very easy as the data can be found in leaf nodes only
4. Deletion of non-leaf nodes is very complicated	4. Deletion is very simple because data will be in the leaf node
5. Leaf nodes cannot be stored using linked lists	5. Leaf node data are ordered using sequential linked lists
6. The structure and operations are complicated	6. The structure and operations are simple

References

- https://www.youtube.com/watch?v=94ErZ5K8XZg&list=PLdo5W4Nhv31bbKJzrsKfMpo_grxuLl8LU&index=71&pp=iAQB (Intro to B Tree)
- https://www.youtube.com/watch?v=Ay2AbTk_QEg&list=PLdo5W4Nhv31bbKJzrsKfMpo_grxuLl8LU&index=72&pp=iAQB (Insertion in B Tree)
- https://www.youtube.com/watch?v=GKa_t7fF8o0&list=PLdo5W4Nhv31bbKJzrsKfMpo_grxuLl8LU (Deletion in B Tree)
- https://www.youtube.com/playlist?list=PLdo5W4Nhv31bbKJzrsKfMpo_grxuLl8LU (Jenny’s Lecture’s- Playlist: Data Structures and Algorithms)
- https://www.geeksforgeeks.org/introduction-of-b-tree/?ref=lbp (GFG B+ Trees)
- https://www.youtube.com/@JennyslecturesCSIT (Channel Jenny’s Lectures CS/IT)