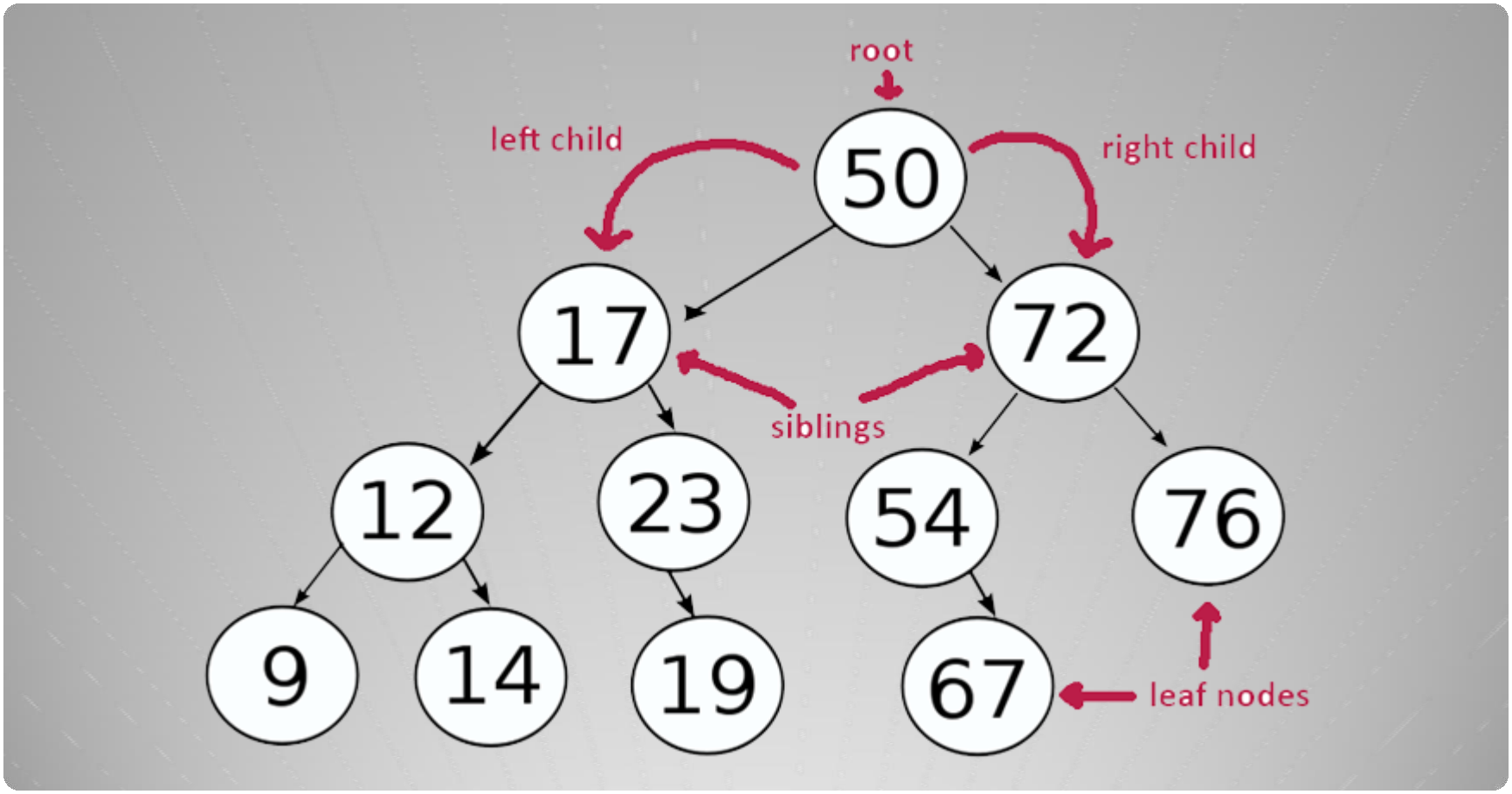
Data
Structures
Fibonacci Heap
FM-Index
Hash Table
Trie
Beap
Splay Tree
Top Tree
Tree
Heap
BST
queue
Soft Heap
Stack
Array
Binary Heap
Soft Heap
Weak Heap
Suffix Tree
Doubly Linked List
Stack
Trie
Beap
Rope
BIT
BST
Tango Tree
array
Queue
Array
Trie
Treap
Tree
Array
Binomial Heap
Trie
ডেটা স্ট্রাকচার শিখি বাংলাতে...
Queue
Beap
Rope
Heap
Treap
BIT
queue
Suffix Array
BST
Tree
Splay Tree
Soft Heap
Linked List

Agenda

- Introduction to Tree.
- How the Tree works?
- Binary Tree Representation.
- Operations performed on the Tree.
- Tree Implementation.


root
left child
right child
50
17
72
siblings
12
23
54
76
9
14
19
67
leaf nodes

Tree

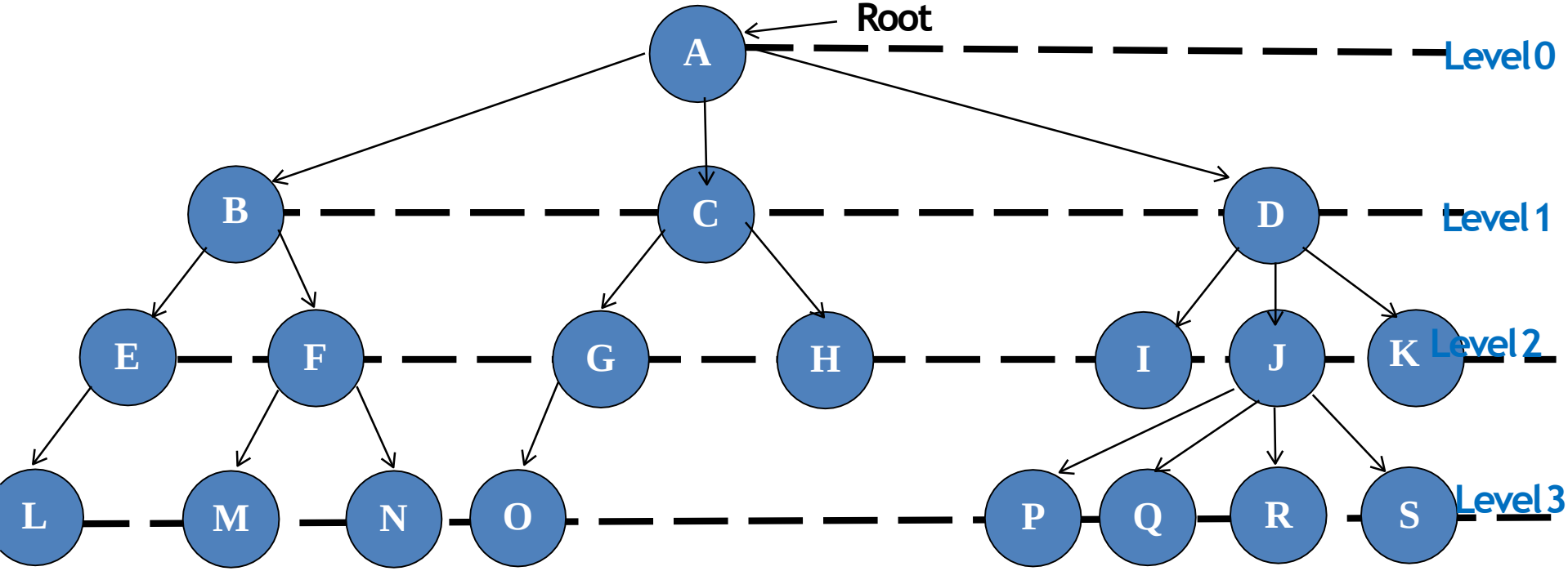
What is a Tree?

- A tree is a collection of nodes arranged in a hierarchical structure such that:

1) There is a special node called the root of the tree.

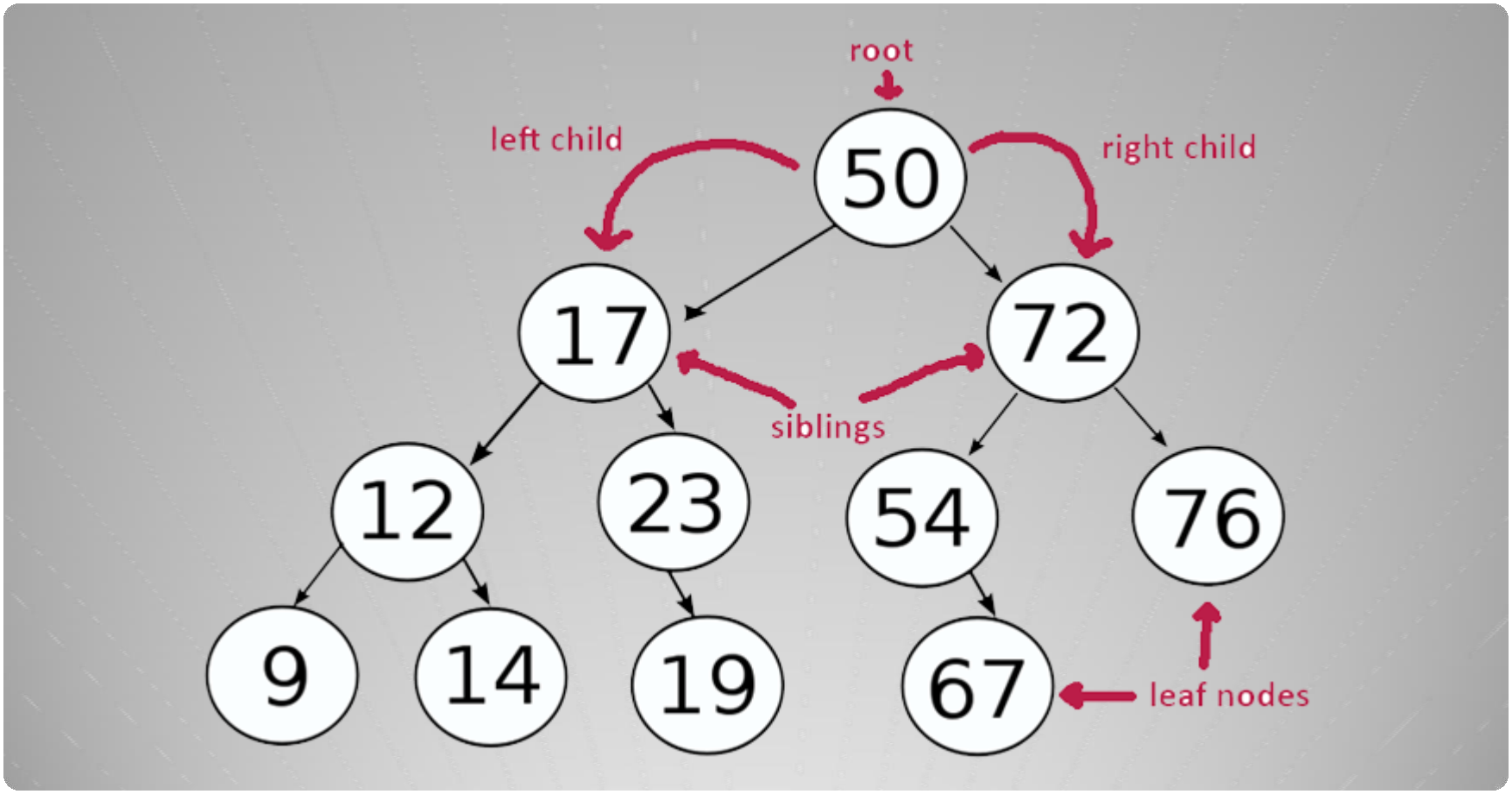
2) Remaining nodes (or data item) are partitioned into number of disjoined subsets each of which is itself a tree, are called sub tree.

Introduction to the Tree



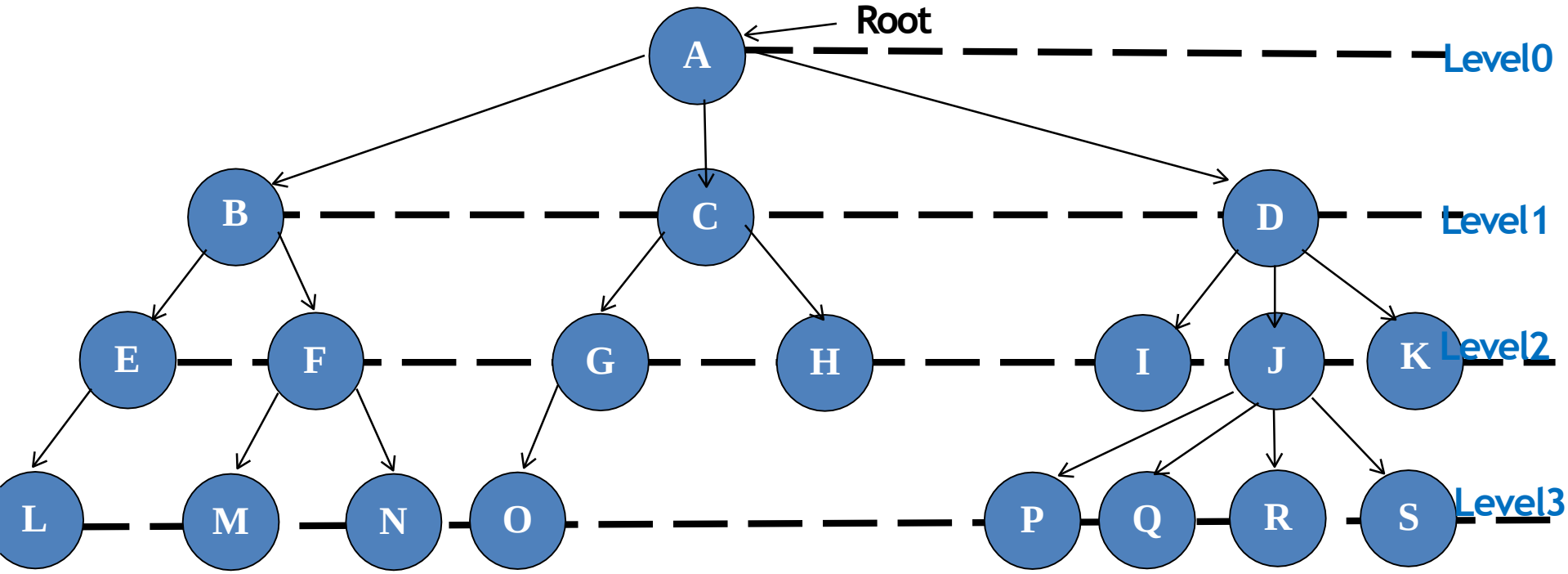
- Root is a specially designed node (or data items) in a tree. It is the first node in the hierarchical arrangement of the data items.

- 'A' is a root node in the previous Figure. Each data item in a tree is called a node. It specifies the data information and links (branches) to other data items.



Basic Terminologies

Basic Terminologies



The degree of a tree =4

The height(depth) of the tree=3

Basic Terminologies

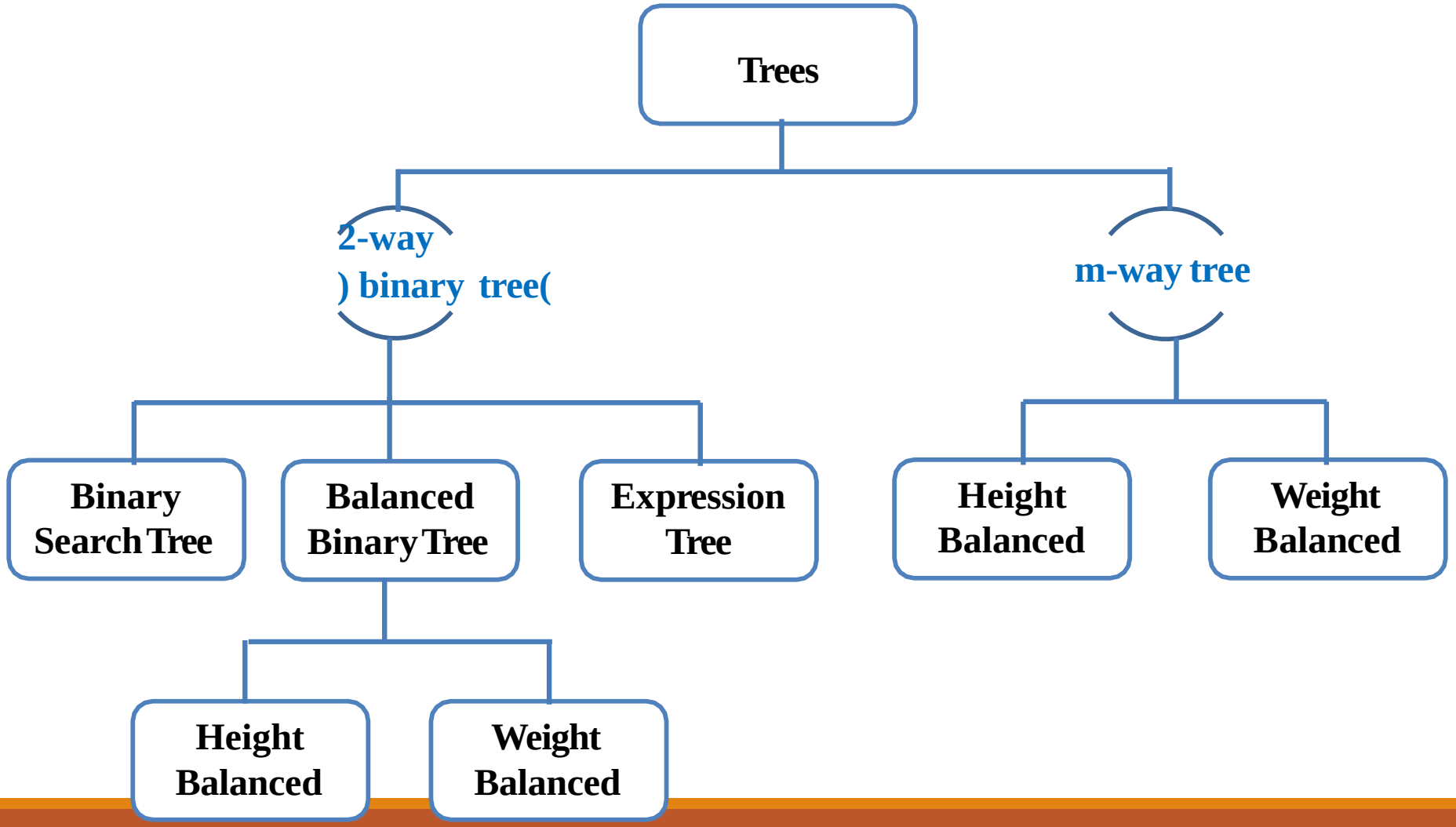
- Degree of a node is the number of sub-trees of a node in a given tree.
- Examples:
  - The degree of node A is 3
  - The degree of node B is 2
  - The degree of node C is 2
  - The degree of node D is 3
- The degree of a tree is the maximum degree of node in a given tree.
- The degree of node J is 4 while all the other nodes have less or equal degree. So the degree of the above tree is 4.
- A node with degree Zero is called a terminal node or a leaf. For example M, N, I, O etc. are leaf nodes.
- Any node whose degree is not zero is called non-terminal node.

Basic Terminologies

- The tree is structured in different levels. The entire tree is leveled in a way that the root node is always of level 0 .
- Then, its immediate children are at level 1 and their immediate children are at level 2 and so on up to the terminal nodes. that is, if a node is at level n, then its children will be at level n+1.
- Depth of a tree (or height) is *the maximum level of any node in a given tree*. That is a number of level one can descend the tree from its root node to the terminal nodes (leaves).
- The term height is also used to denote the depth.

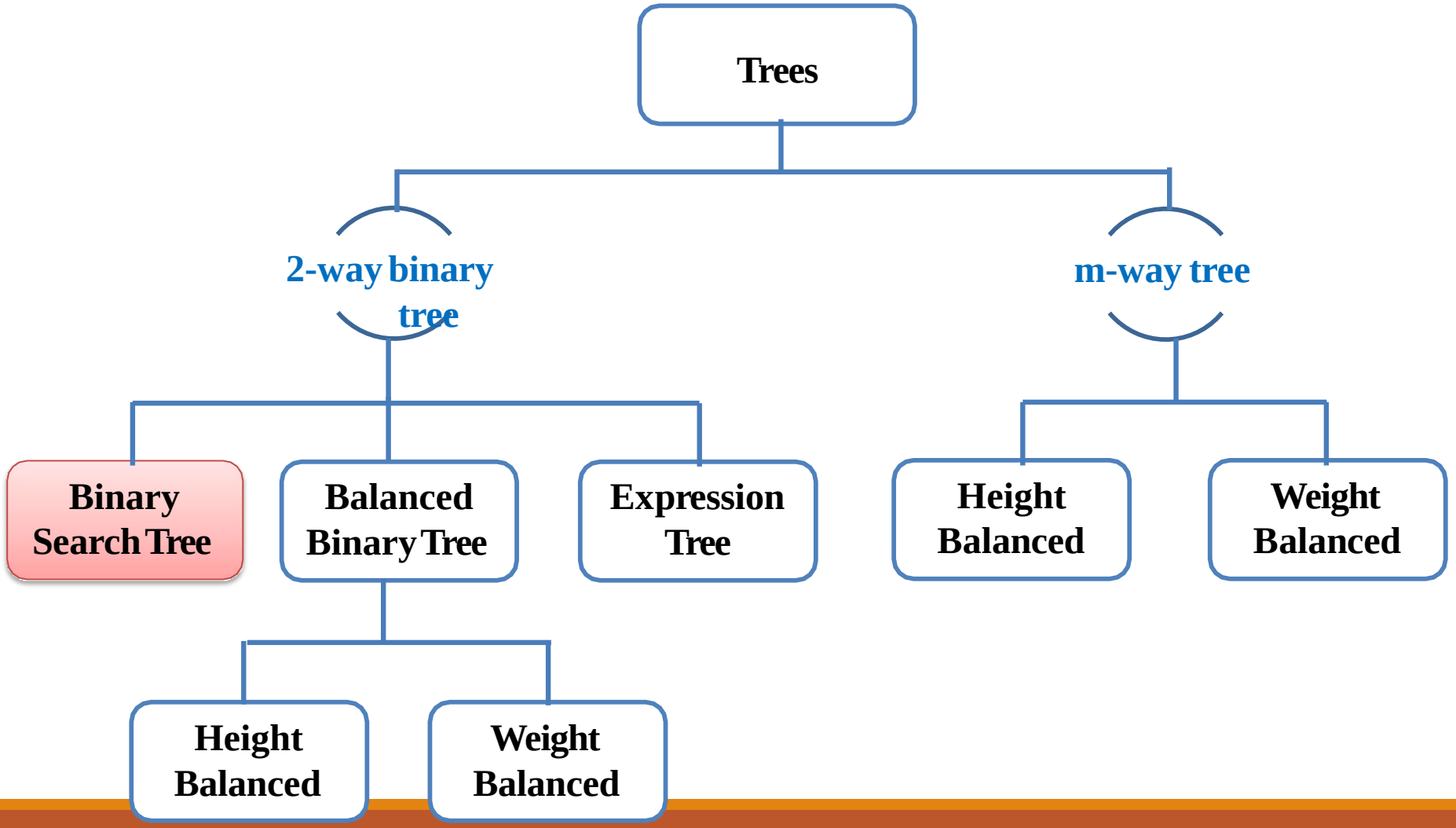
Basic Terminologies

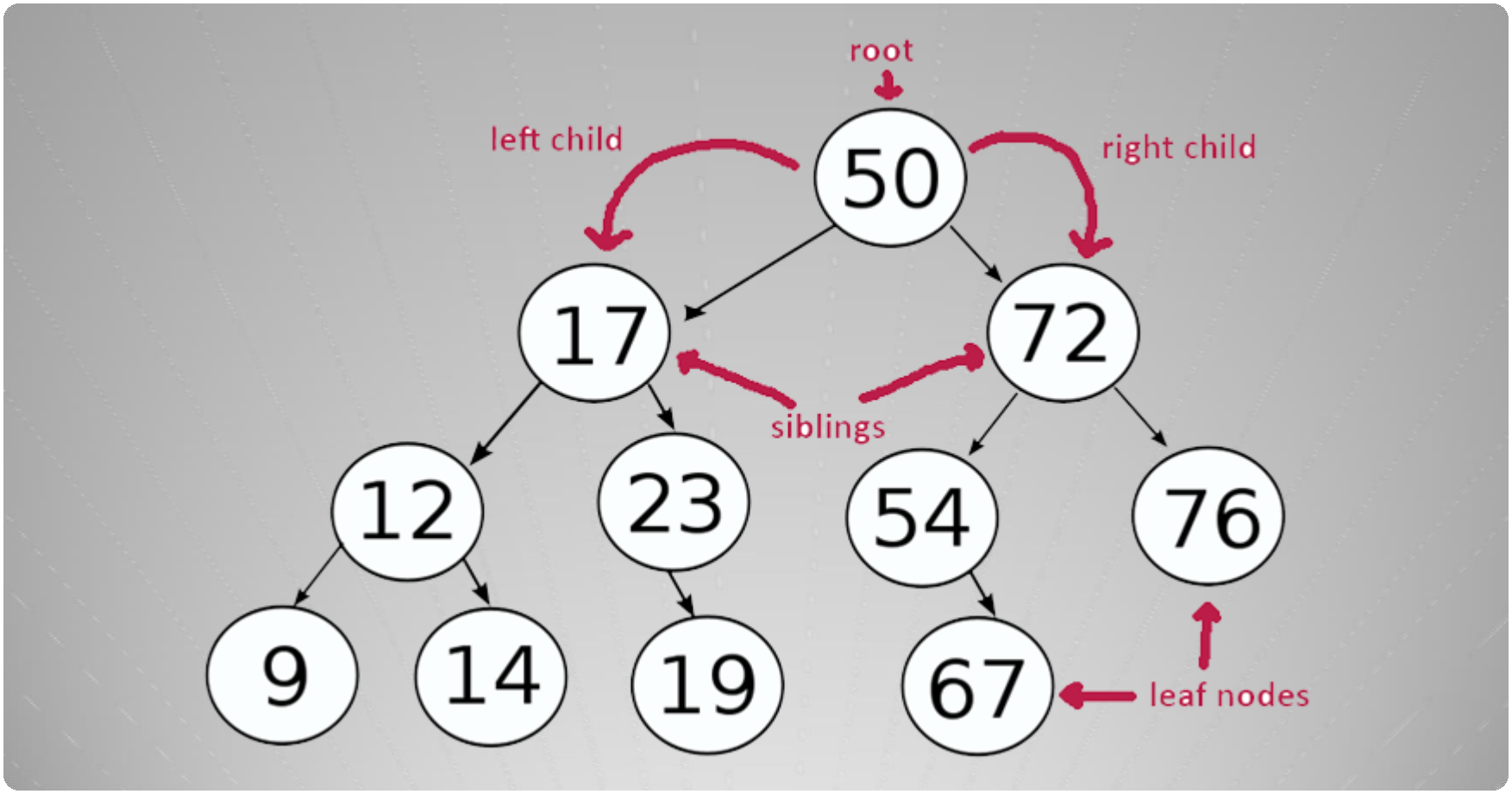
- Trees can be divided in different classes as follows:



Basic Terminologies

- Trees can be divided in different classes as follows:

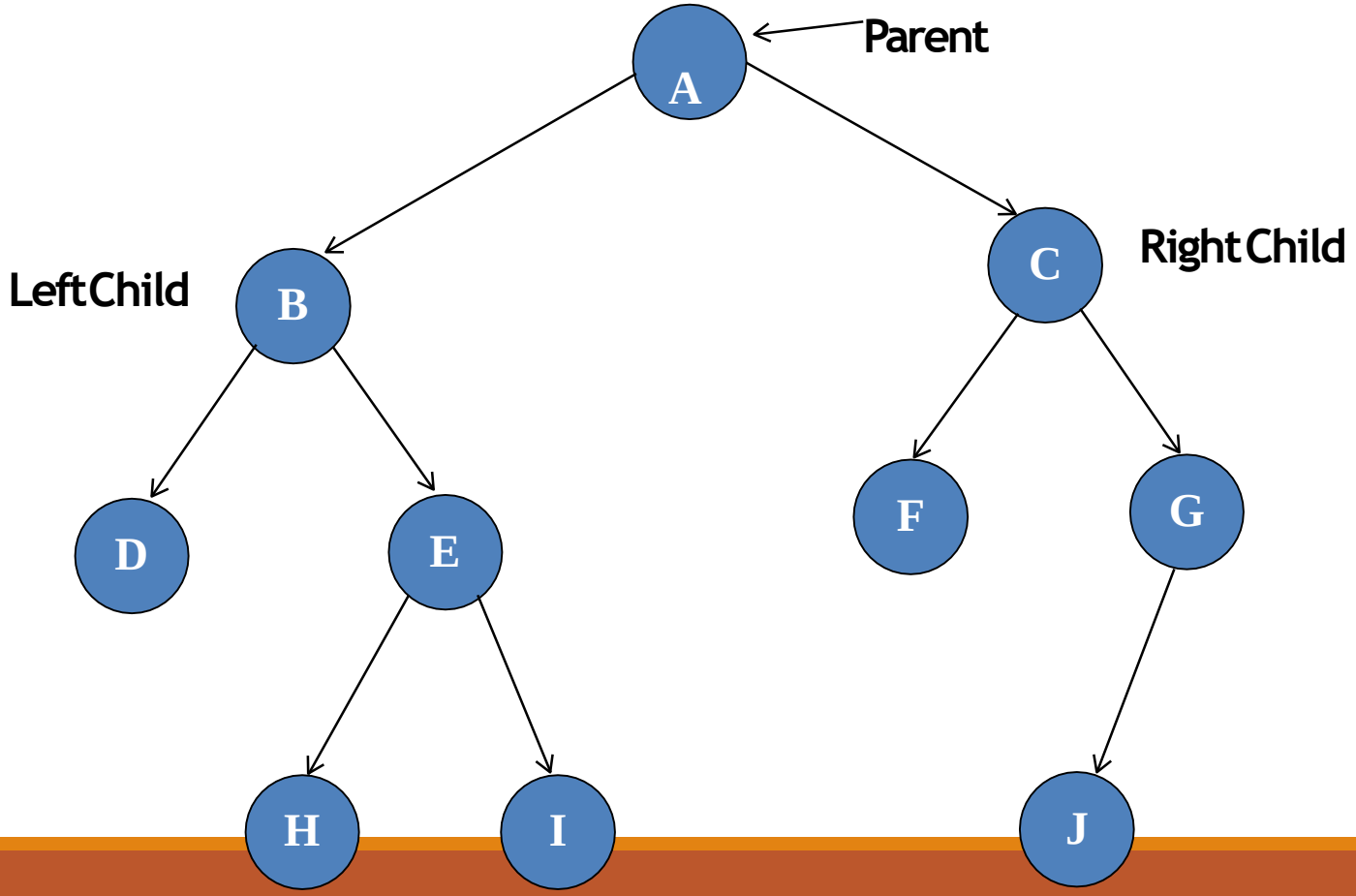




Binary Trees

Binary Trees

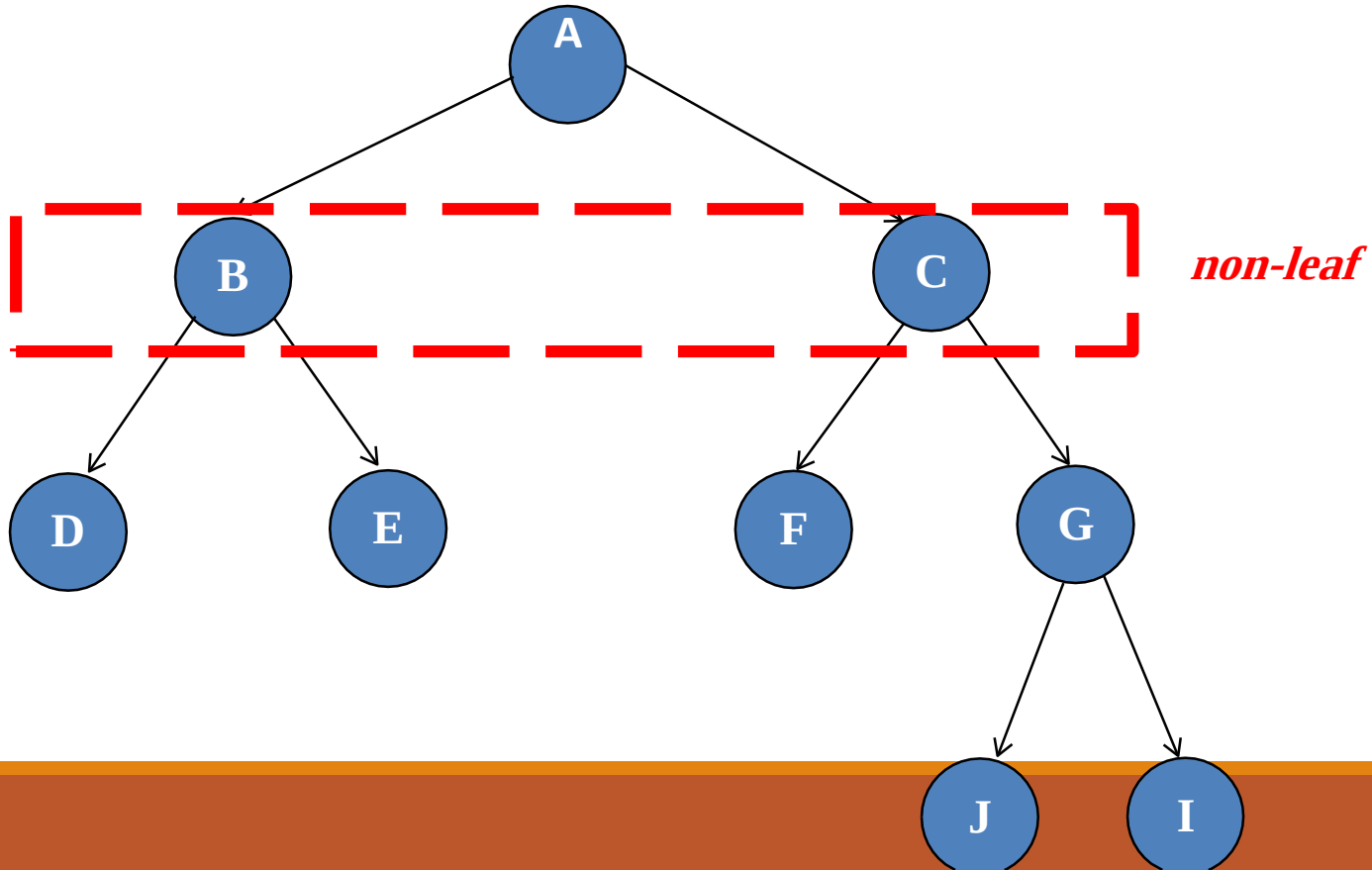
- A binary tree is a tree in which no node can have *more than two children*. Typically these children are described as "left child" and "right child" of the parent node.



Strictly Binary Tree

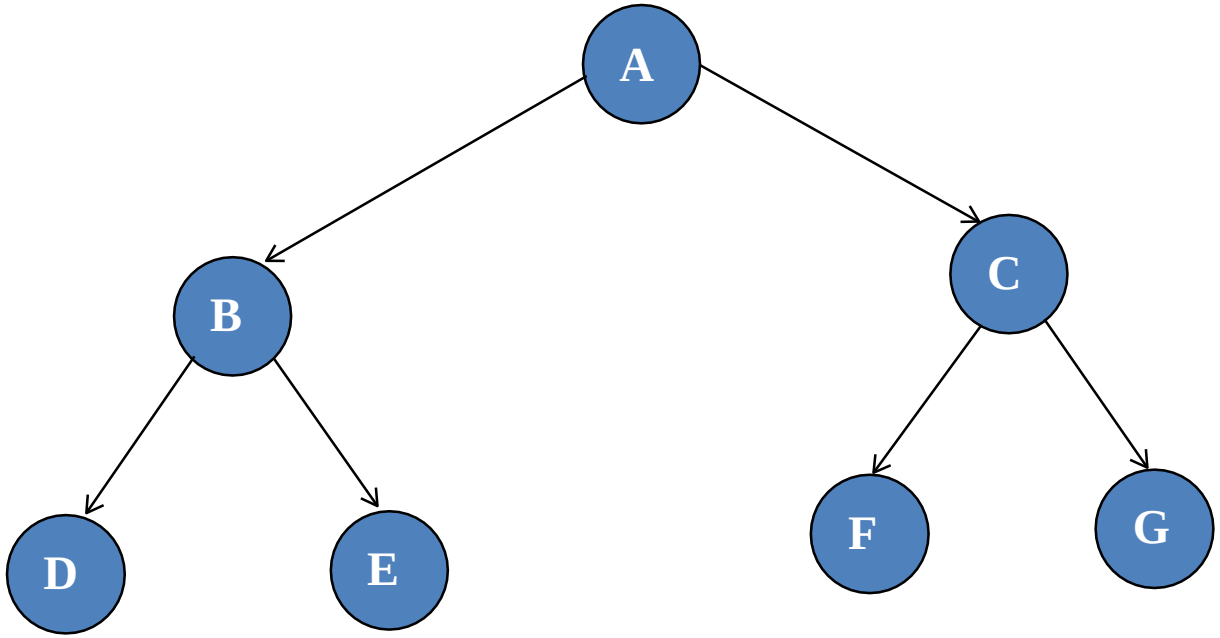
- The *tree* is said to be *strictly binary tree*, if *every non-leaf* made *in a binary tree* has *non-empty left* and *right sub trees.*

- A *strictly binary tree* with *n leaves* always contains *2n–1 nodes*.



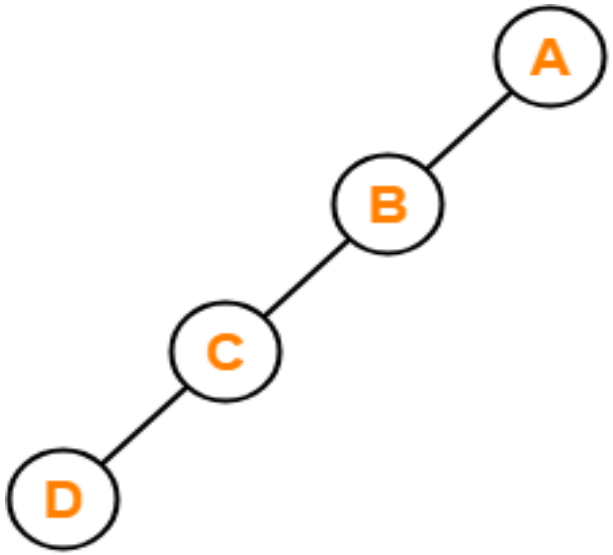
Complete Binary Tree

- A complete binary tree at depth '*d*' is the *strictly binary tree*, where *all the leaves are at level d*.

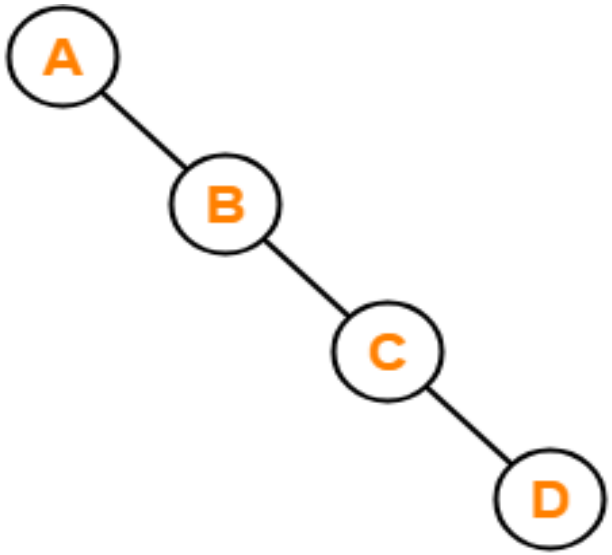


Binary Tree Versus Ordinary Tree

- If a binary tree has only left sub trees, then it is called left skewed binary tree and If a binary tree has only right sub trees, then it is called right skewed binary tree.



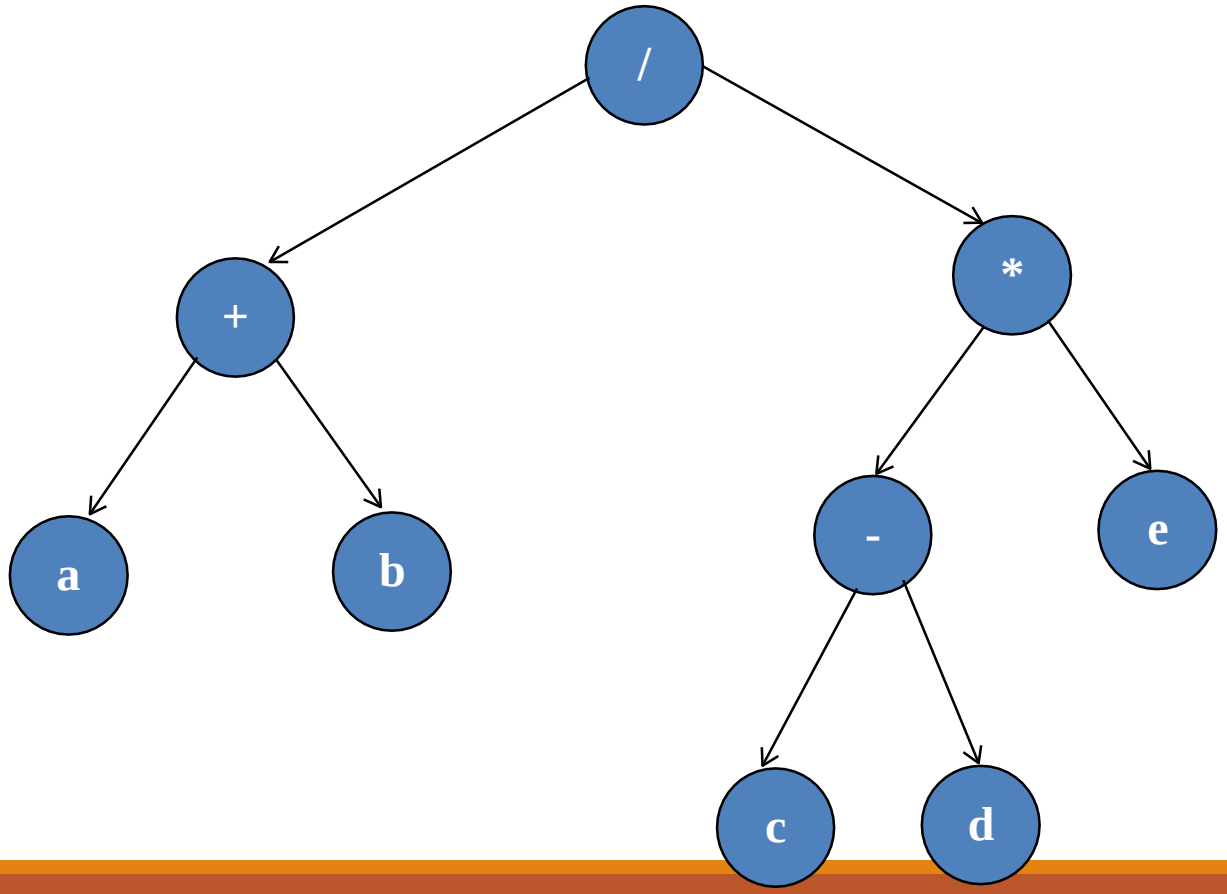
Left Skewed Binary Tree

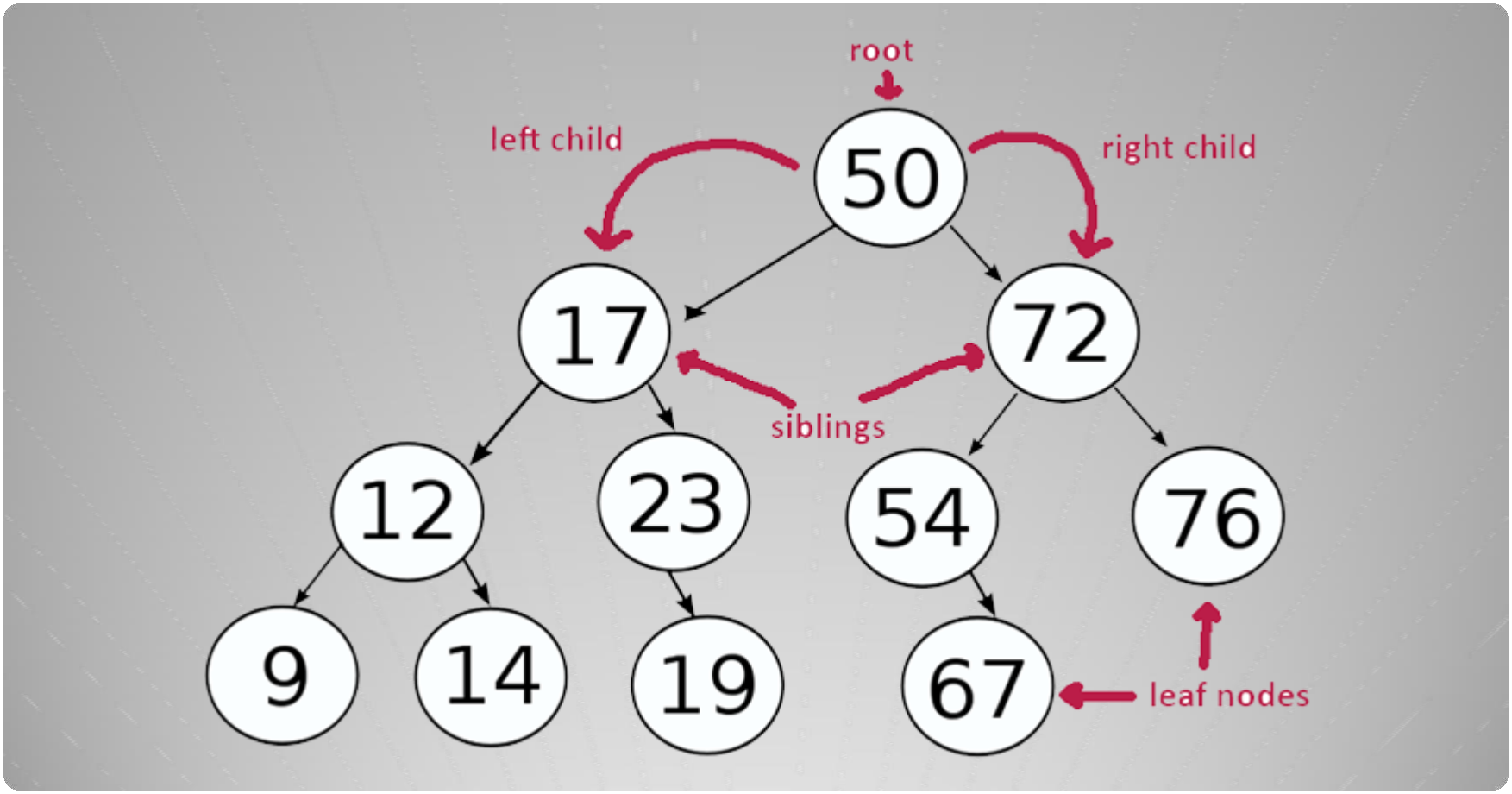


Right Skewed Binary Tree

Expression Binary Tree

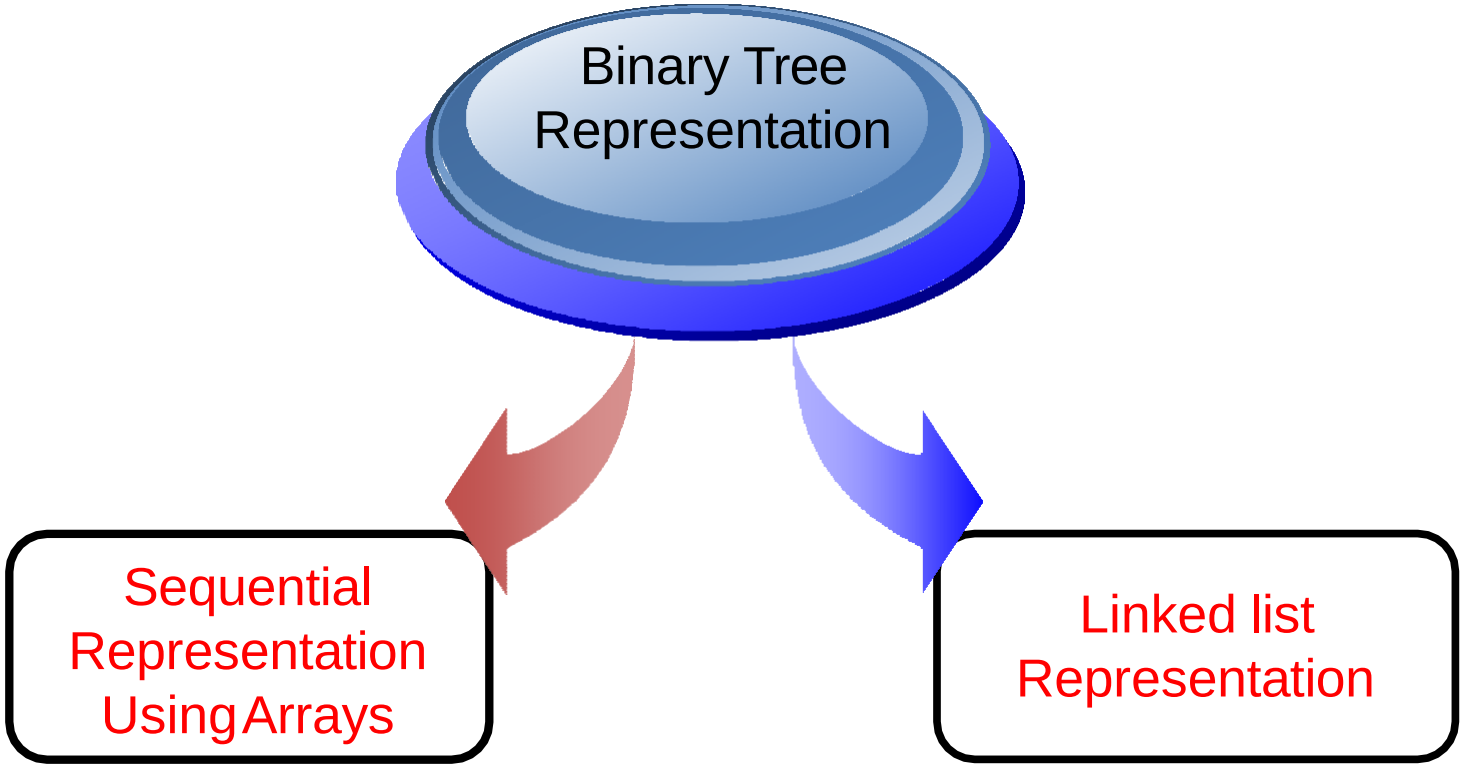
- For example: consider an algebraic expression E.
  where E = (a + b)/((c – d)*e)
- So, the Expression tree is

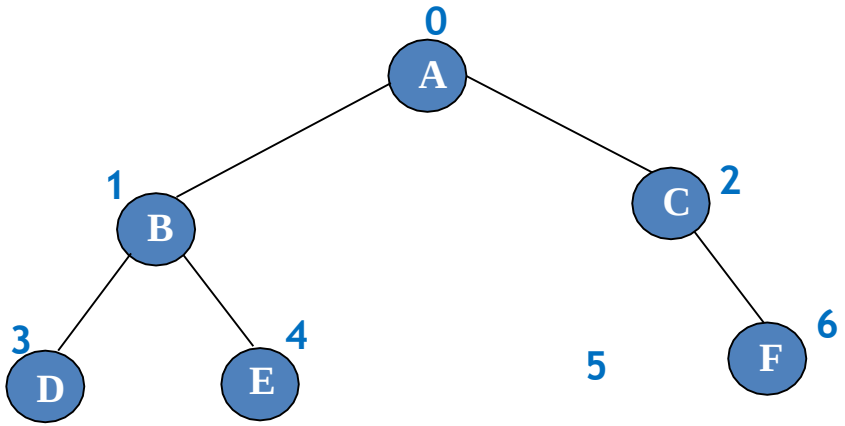




Binary Tree Representation

Binary Tree Representation





A[ ]	A	B	C	D	E		F
	[0]	[1]	[2]	[3]	[4]	[5]	[6]

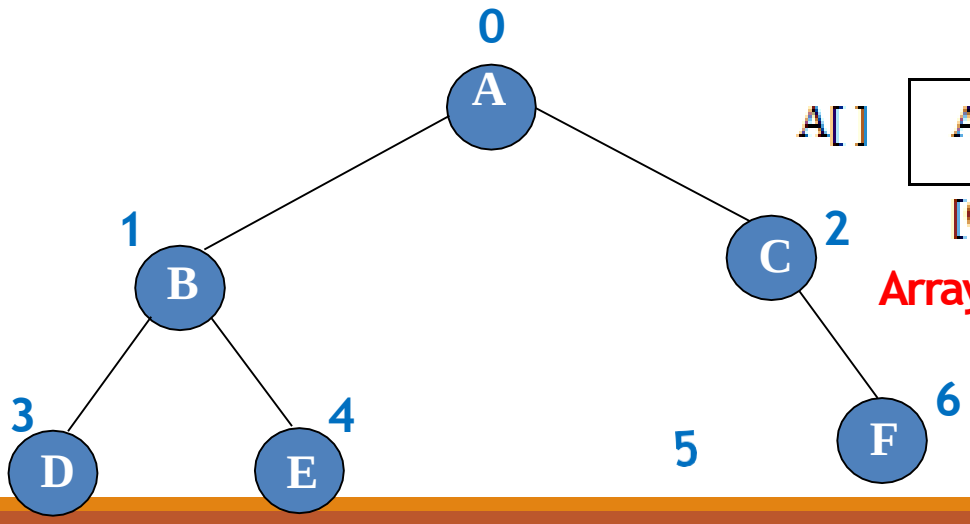
Array Representation

Array Representation

- An array can be used to store the nodes of a binary tree. The nodes stored in an array of memory can be accessed sequentially.

- Suppose a binary tree T of height d. Then at most $(2^{d+1} - 1)$ nodes can be there in T.(i.e., SIZE = $2^{d+1}-1$) So the array of size "SIZE" to represent the binary tree.

- Consider a binary tree of height 2. Then SIZE = $2^3 - 1 = 7$. Then the array A[7] is declared to hold the nodes.



Binary tree of depth 3

A[ ]	A	B	C	D	E		F
	[0]	[1]	[2]	[3]	[4]	[5]	[6]

Array representation of the binary tree

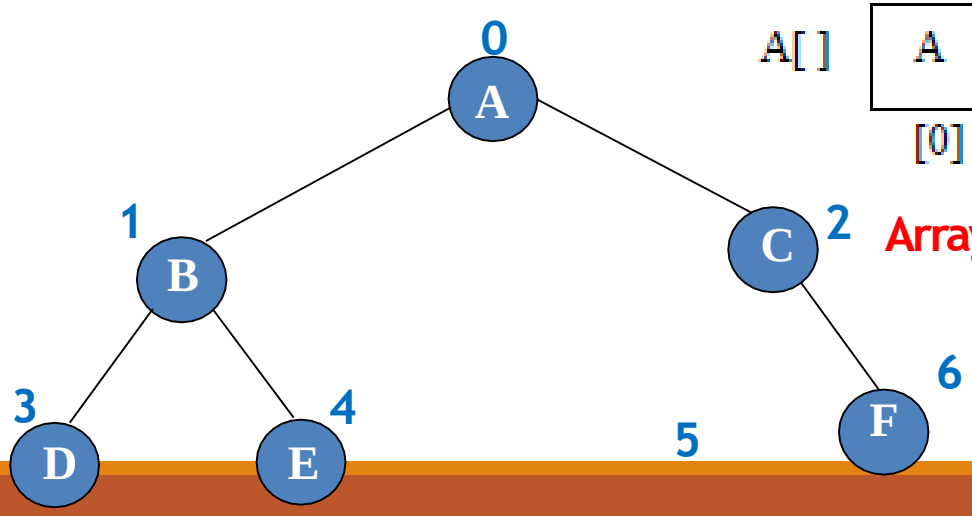
Array Representation

- In order to perform any operation often we have to identify the father, the left child and right child of an arbitrary node.

1. The left child of a node having index n can be obtained by (2n+1)

2.The right child of a node having array index n can be obtained by the formula (2n+ 2)

3.If the left child is at array index n, then its right brother is at (n + 1). Similarly, if the right child is at index n, then its left brother is at (n-1)



Binary tree of depth 3

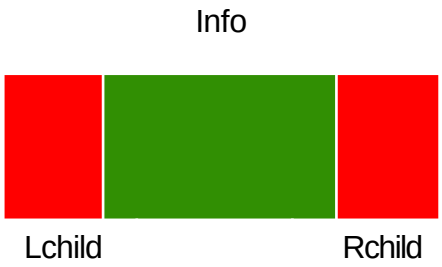
A[ ]	A	B	C	D	E		F
	[0]	[1]	[2]	[3]	[4]	[5]	[6]

Array representation of the binary tree

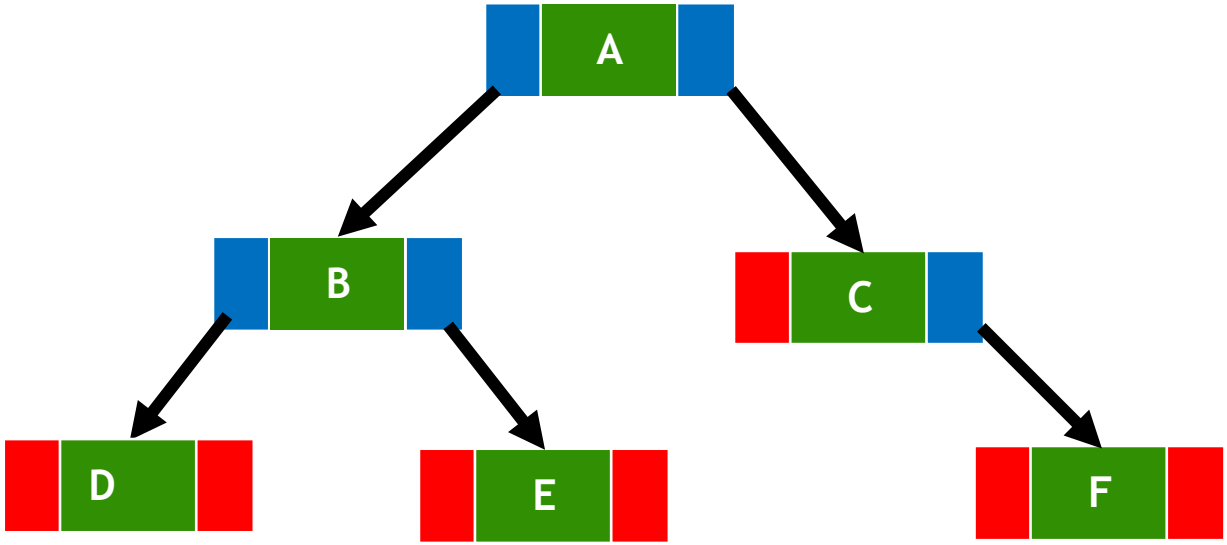
Linked List Representation

Linked List Representation

- The most popular and practical way of representing a binary tree is using linked list (or pointers). In linked list, every element is represented as nodes. A node consists of three fields such as:
  (a) Left Child (Lchild(
  (b) Information of the Node (Info(
  (c) Right Child (Rchild(

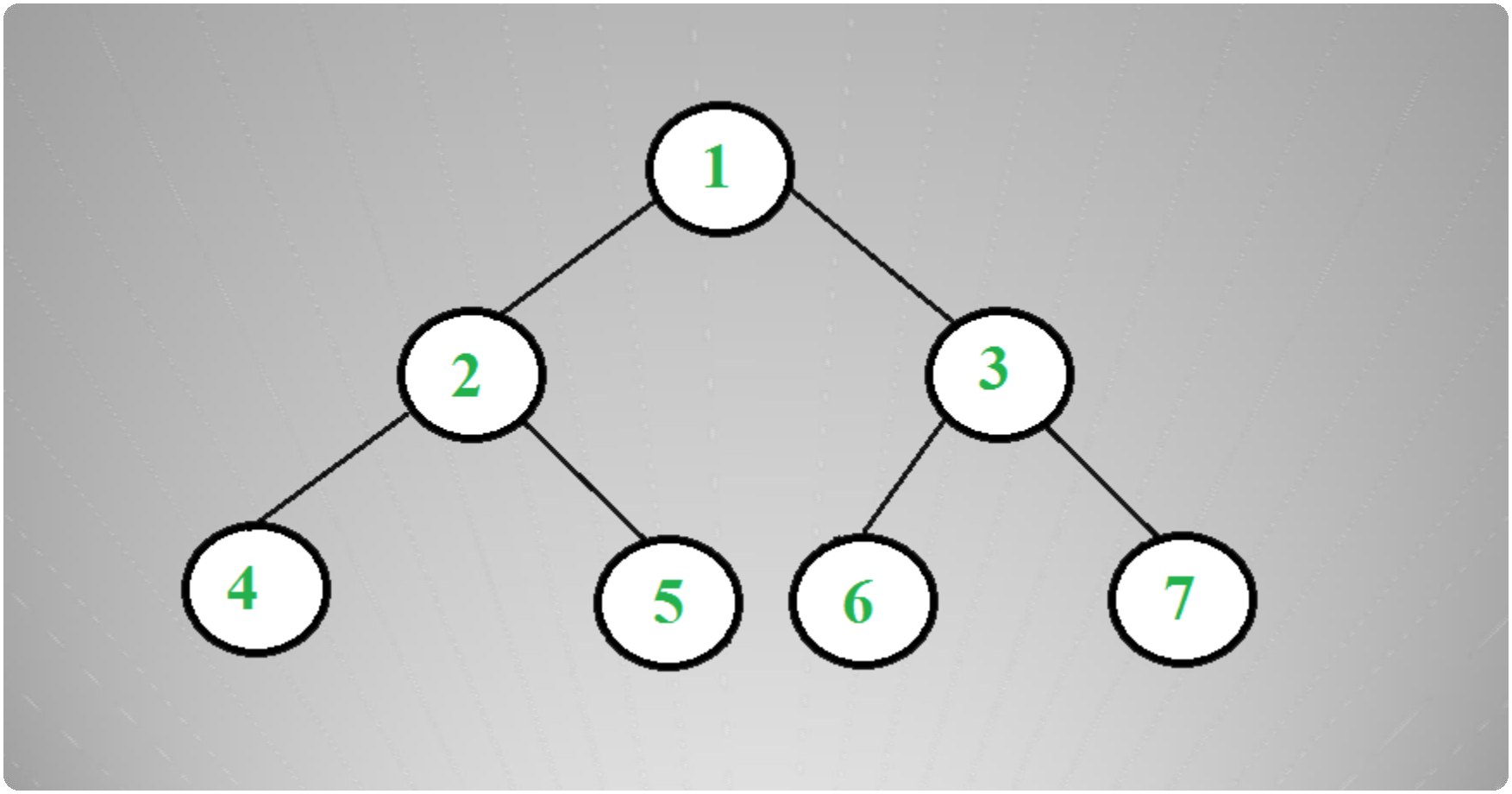



```
struct node
{
        int Info;
        node *Lchild;
        node *Rchild;
};
```



linked list representation of the binarytree

- If a node has left or/and right node, corresponding LChild or/and RChild are assigned to NULL.



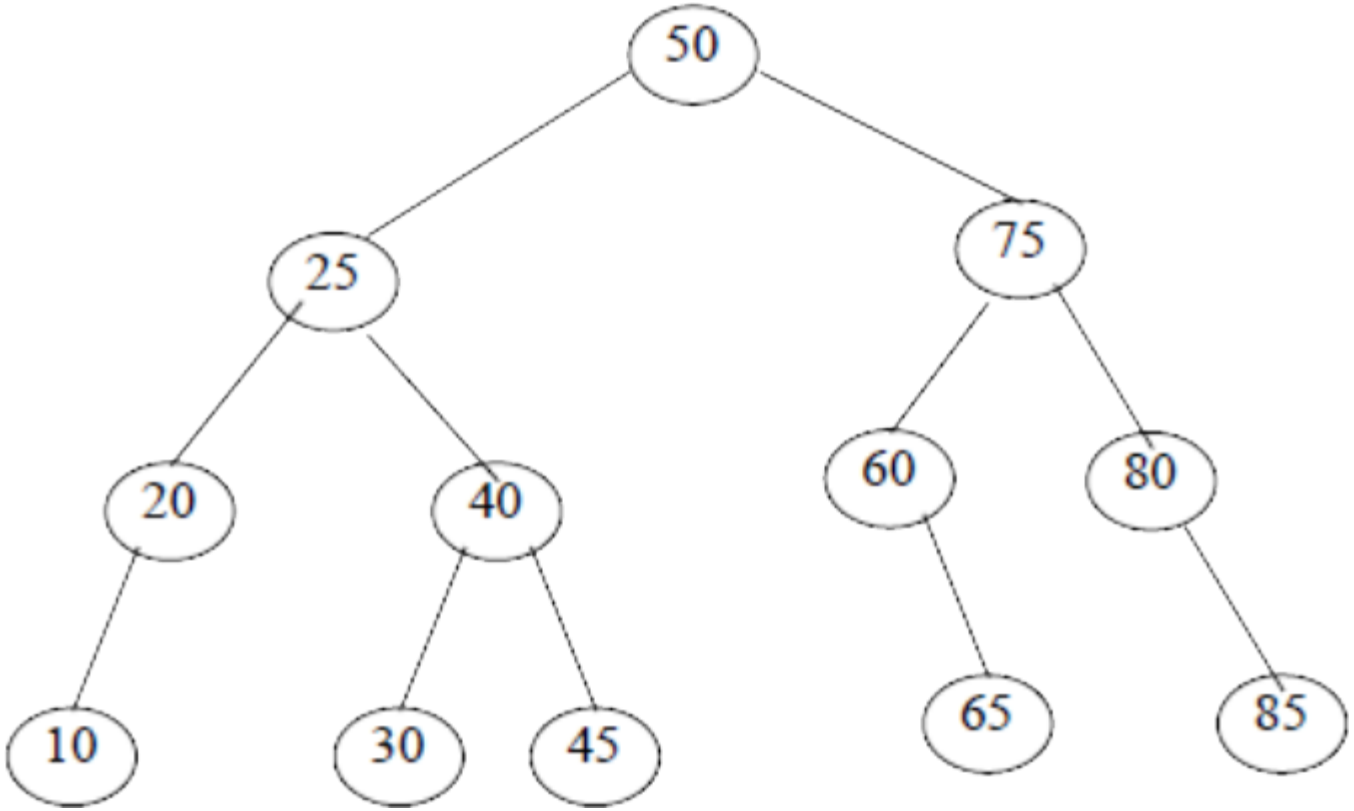
Operations on the Binary Tree

Operations Performed on theTree

- The basic operations that are commonly performed an binary tree is listed below:

1. Create an empty Binary Tree.
2. Traversing a Binary Tree.
3. Inserting a New Node.
4. Deleting a Node.
5. Searching for a Node.
6. Determine the total no of Nodes.
7. Determine the total no leaf Nodes.
8. Determine the total no non-leaf Nodes.
9. Find the smallest element .
10. Finding the largest element.
11. Find the Height of the tree.
12. Finding the Father/Left Child/Right Child/Brother of an arbitrary node.

Binary Search Trees (BST)



Binary Search Trees (BST)

- A Binary Search Tree is a binary tree, which is either empty or satisfies the following properties:

1.Every node has a value and no two nodes have the same value (i.e., all the values are unique)

2.If there exists a left child or left sub tree then its value is less than the value of the root.

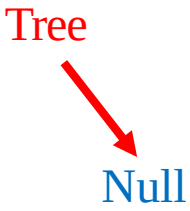
3.The value(s) in the right child or right sub tree is larger than the value of the root node.

Binary Search Tree Construction

- Construct a binary search tree using the following sequence:

  50,75,25,20,60,40,30,10,45,65,80,85

- The BST Tree:

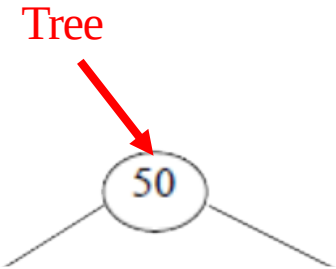


Binary Search Tree Construction

- Construct a binary search tree using the following sequence:

50,75,25,20,60,40,30,10,45,65,80,85

- The BST Tree:

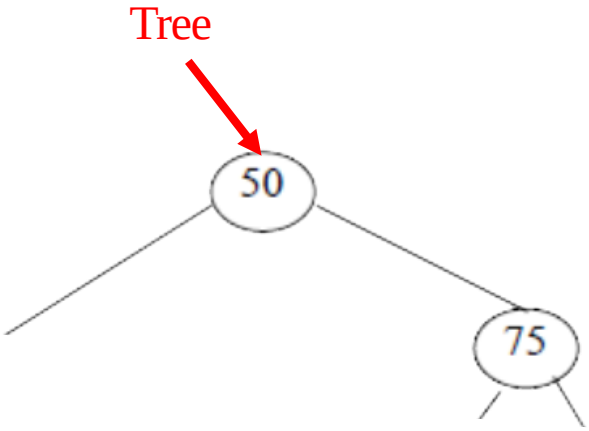


Binary Search Tree Construction

- Construct a binary search tree using the following sequence:

50,75,25,20,60,40,30,10,45,65,80,85

- The BST Tree:

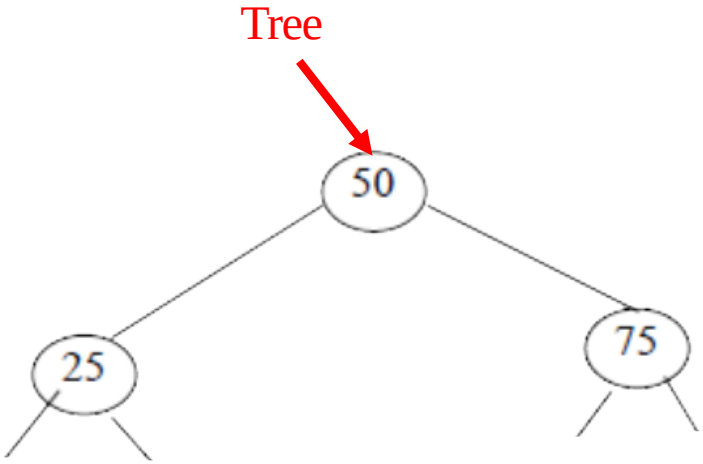


Binary Search Tree Construction

- Construct a binary search tree using the following sequence:

50,75,25,20,60,40,30,10,45,65,80,85

- The BST Tree:

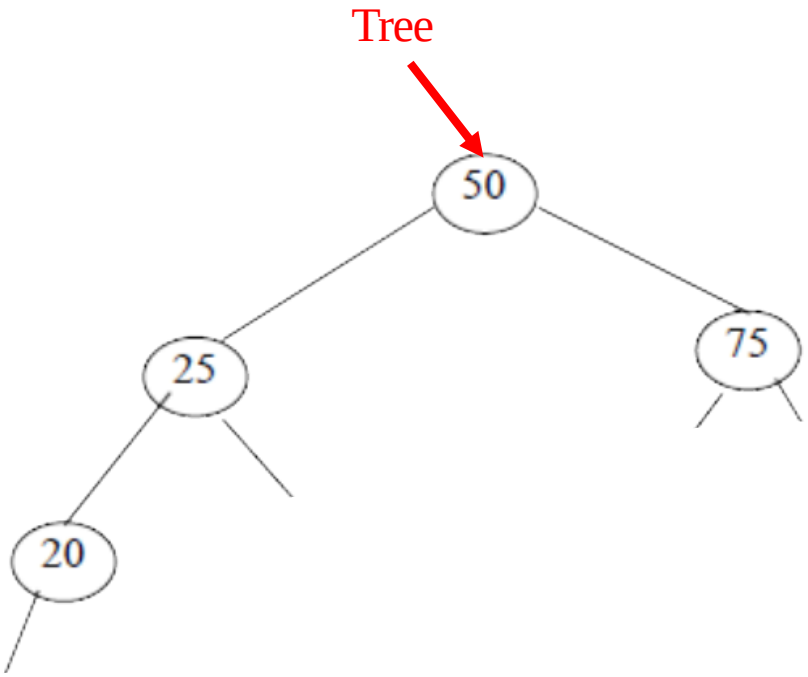


Binary Search Tree Construction

- Construct a binary search tree using the following sequence:

50,75,25,20,60,40,30,10,45,65,80,85

- The BST Tree:

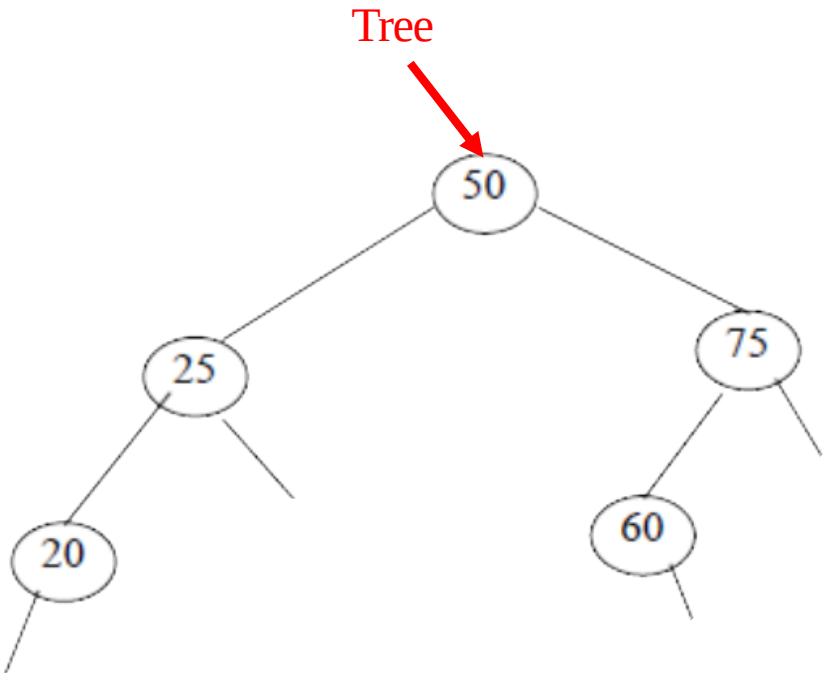


Binary Search Tree Construction

- Construct a binary search tree using the following sequence:

50,75,25,20,60,40,30,10,45,65,80,85

- The BST Tree:

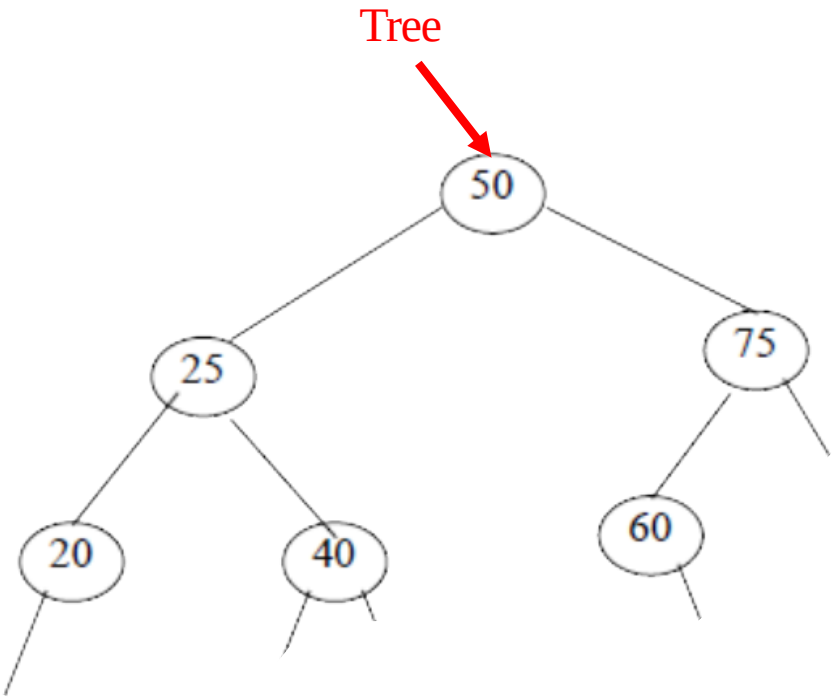


Binary Search Tree Construction

- Construct a binary search tree using the following sequence:

50,75,25,20,60,40,30,10,45,65,80,85

- The BST Tree:

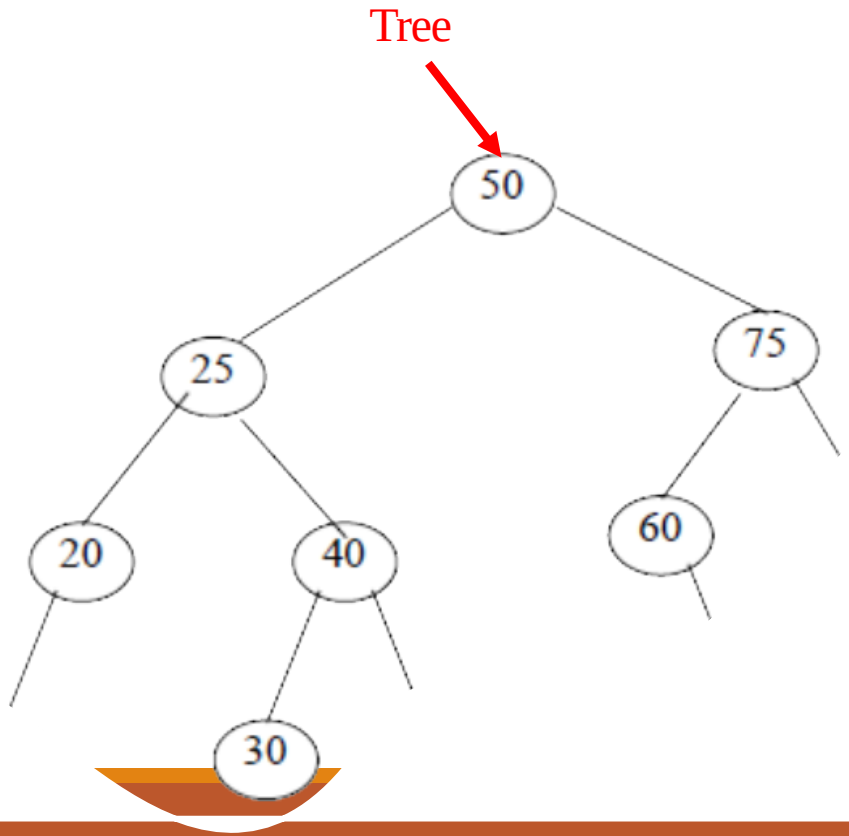


Binary Search Tree Construction

- Construct a binary search tree using the following sequence:

50,75,25,20,60,40,30,10,45,65,80,85

- The BST Tree:

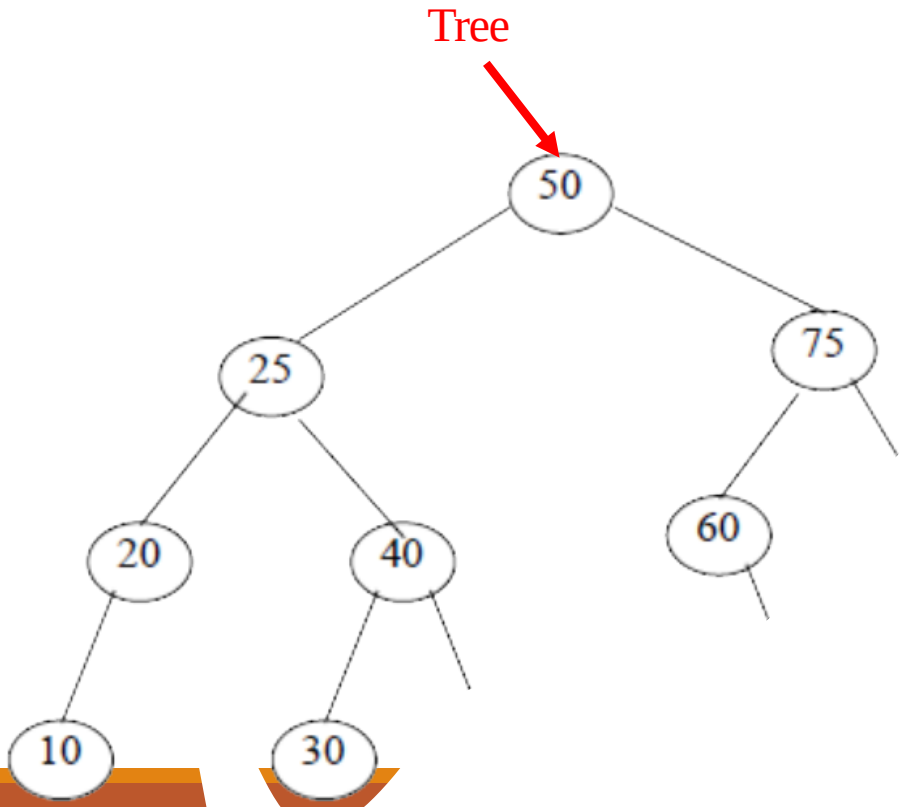


Binary Search Tree Construction

- Construct a binary search tree using the following sequence:

50,75,25,20,60,40,30,10,45,65,80,85

- The BST Tree:

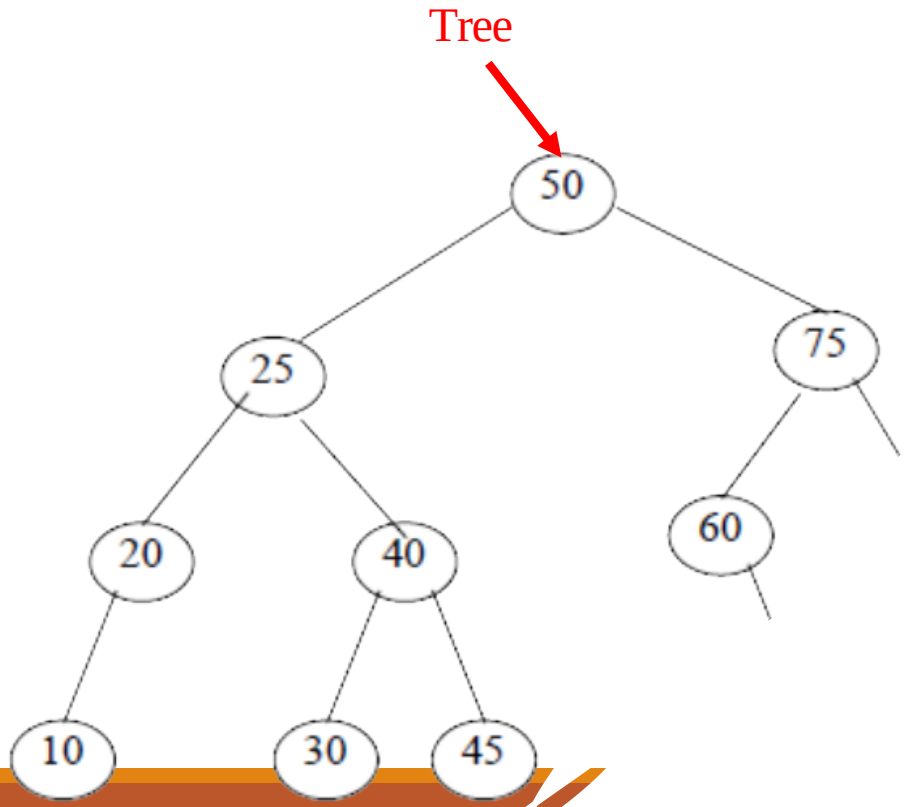


Binary Search Tree Construction

- Construct a binary search tree using the following sequence:

50,75,25,20,60,40,30,10,45,65,80,85

- The BST Tree:

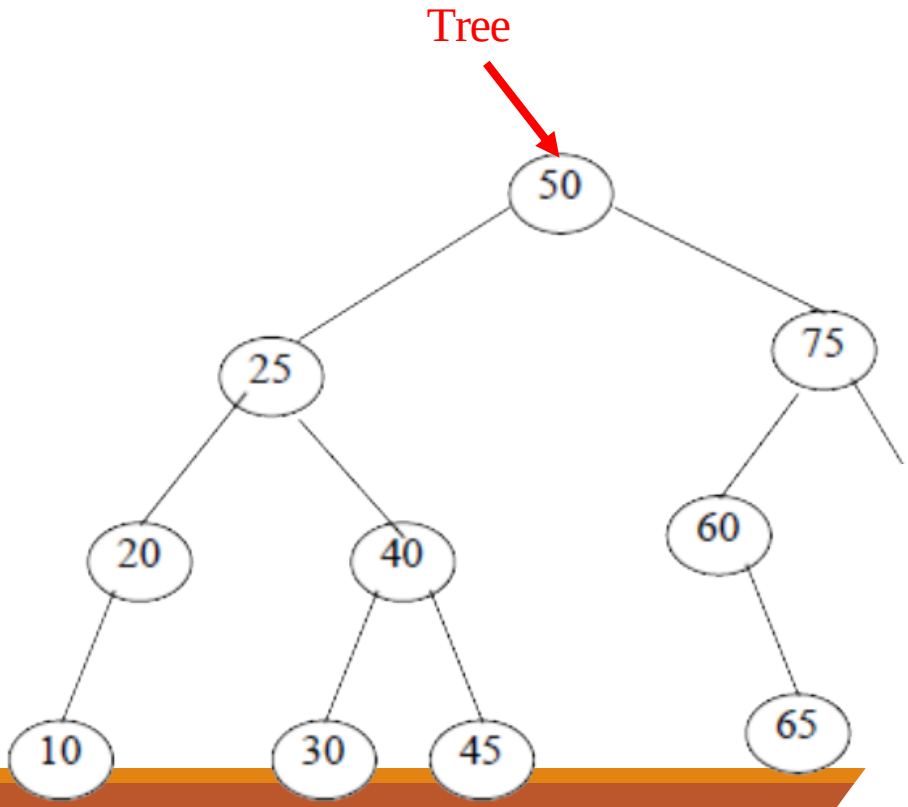


Binary Search Tree Construction

- Construct a binary search tree using the following sequence:

50,75,25,20,60,40,30,10,45,65,80,85

- The BST Tree:

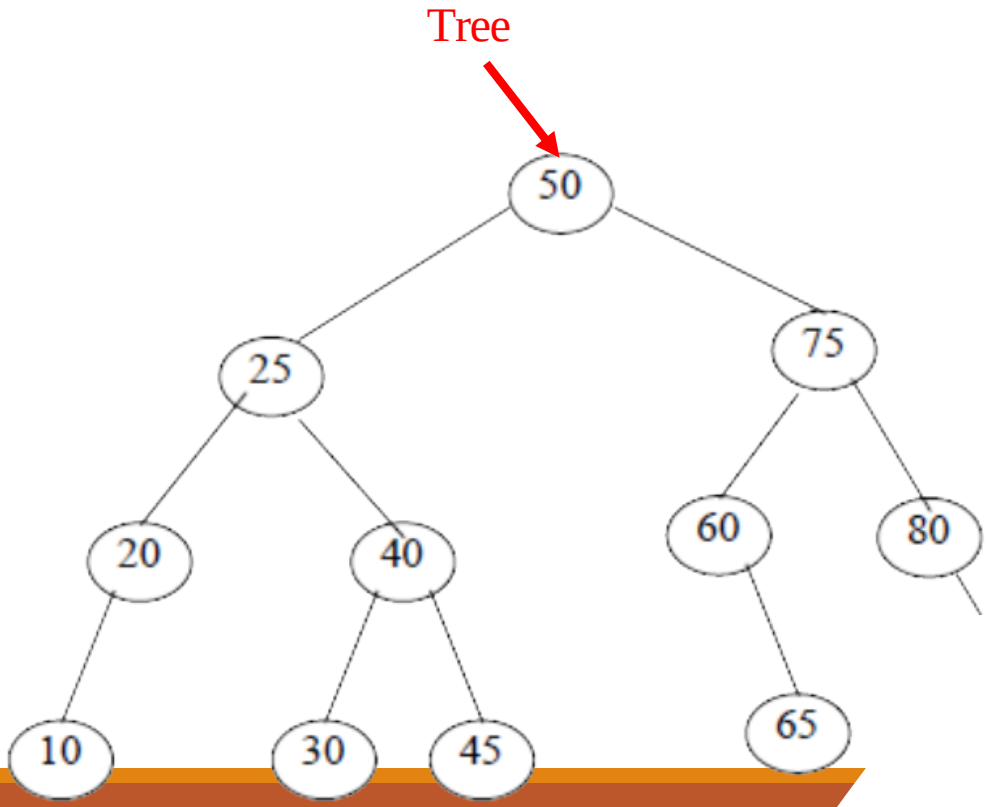


Binary Search Tree Construction

- Construct a binary search tree using the following sequence:

50,75,25,20,60,40,30,10,45,65,80,85

- The BST Tree:

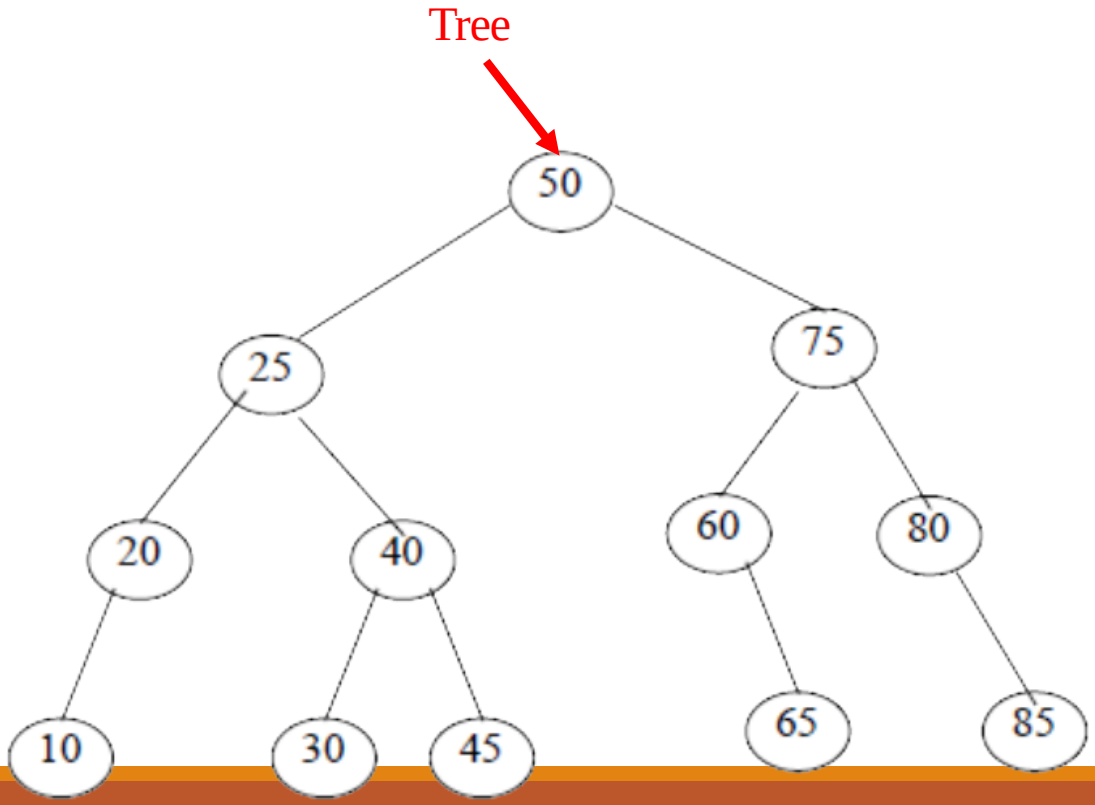


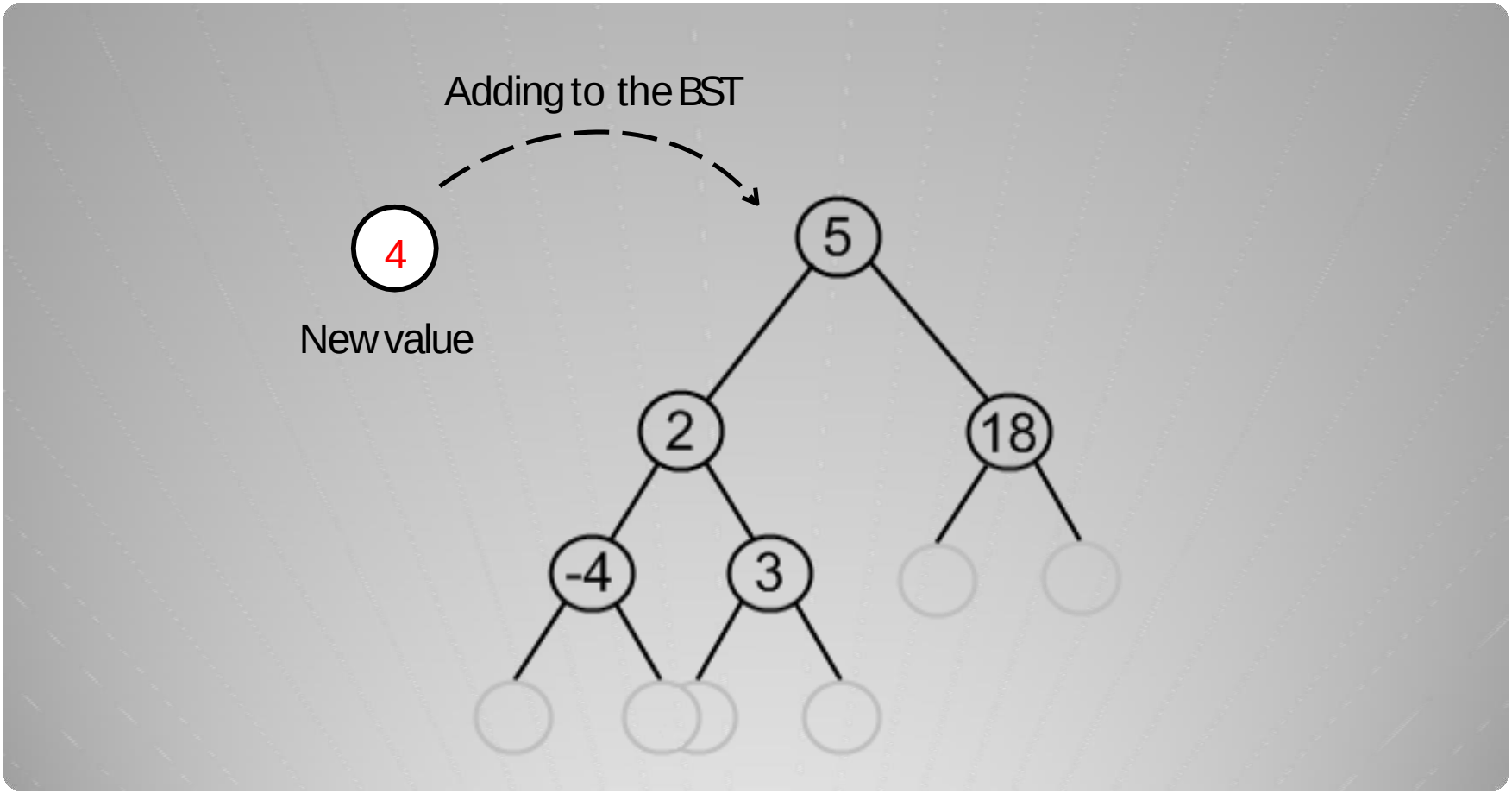
Binary Search Tree Construction

- Construct a binary search tree using the following sequence:

50,75,25,20,60,40,30,10,45,65,80,85

- The BST Tree:





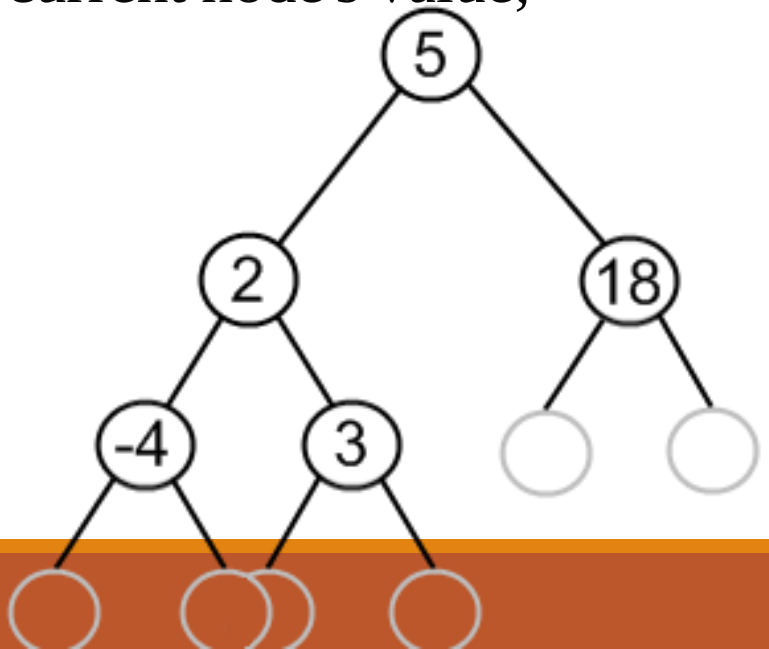
Adding a value to the BST

Adding a value to theBST

- Adding a value to BST can be divided into two stages:

1. search for a place to put a new element;
2. insert the new element to this place.

- If a new value is less, than the current node's value, go to the left sub-tree,
- else
  go to the right sub-tree



Adding a value to the BSTAlg.

- Starting from the root,

1. Check, whether value in current node and a new value are equal. If so, duplicate is found. Otherwise,

2. If a new value is less than the node's value:

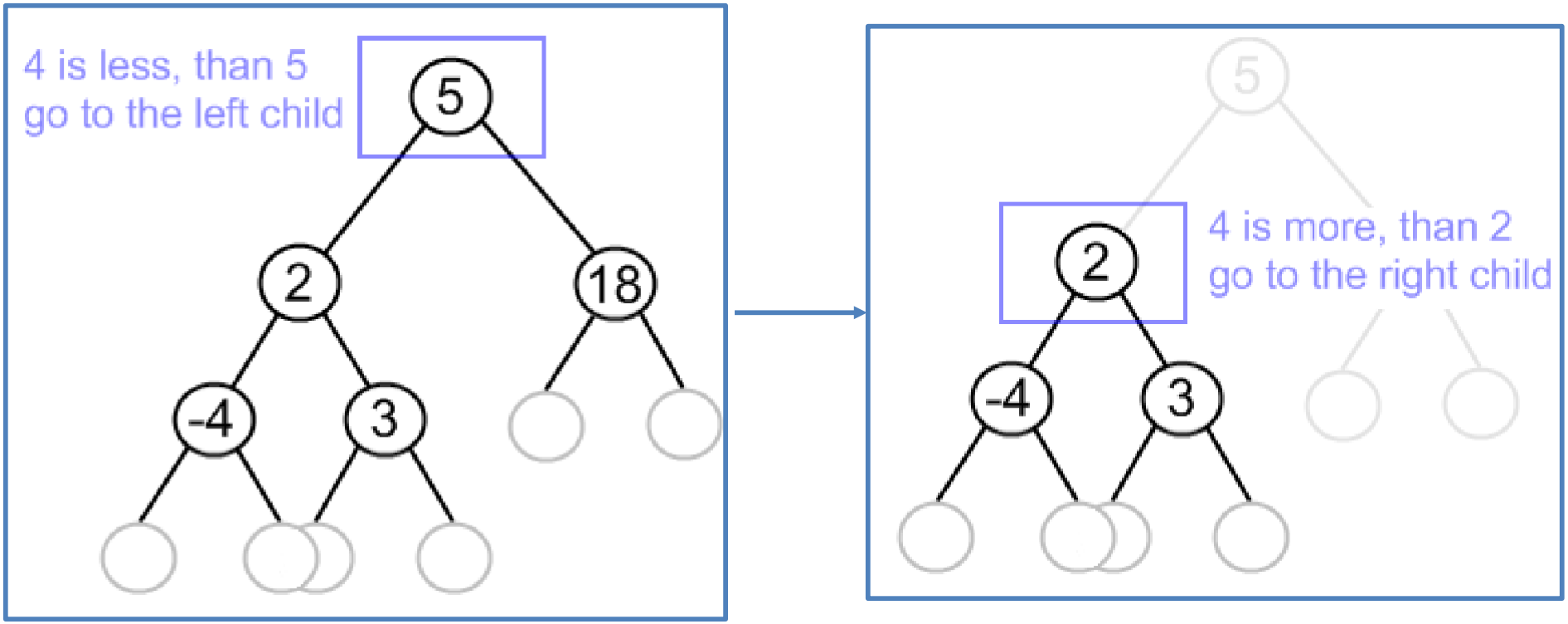
   - If a current node has no left child, place for insertion has been found;
   - Otherwise, handle the left child with the same algorithm.

3. If a new value is greater than the node's value:

   - If a current node has no right child, place for insertion has been found;
   - Otherwise, handle the right child with the same algorithm.

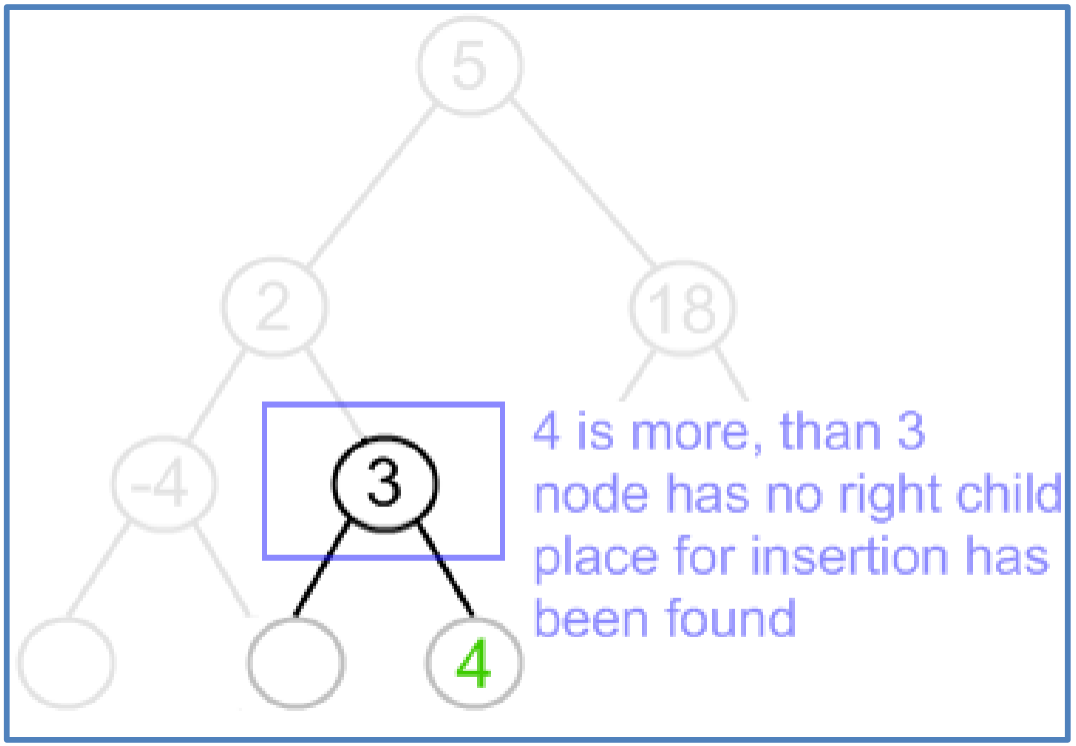
Example for Addinga value to the BST

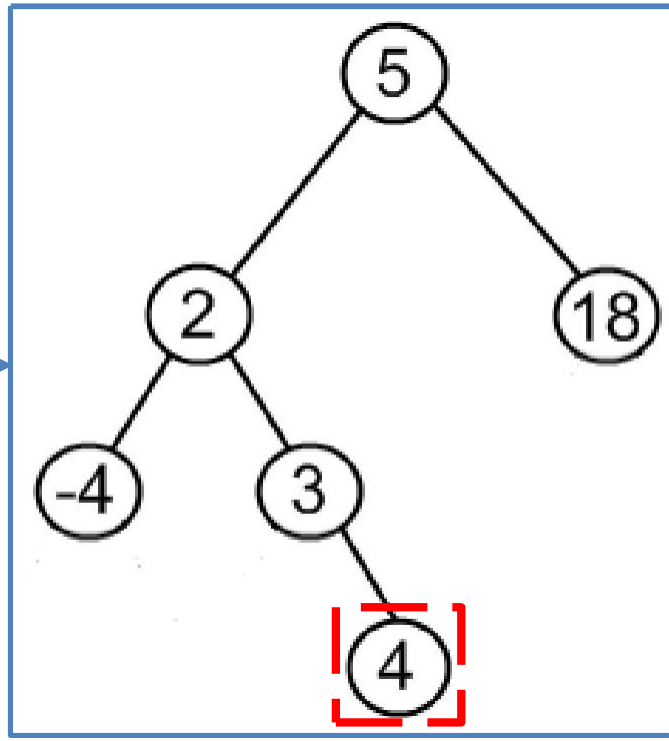
- Insert 4 to the tree:



Example for Addinga value to the BST

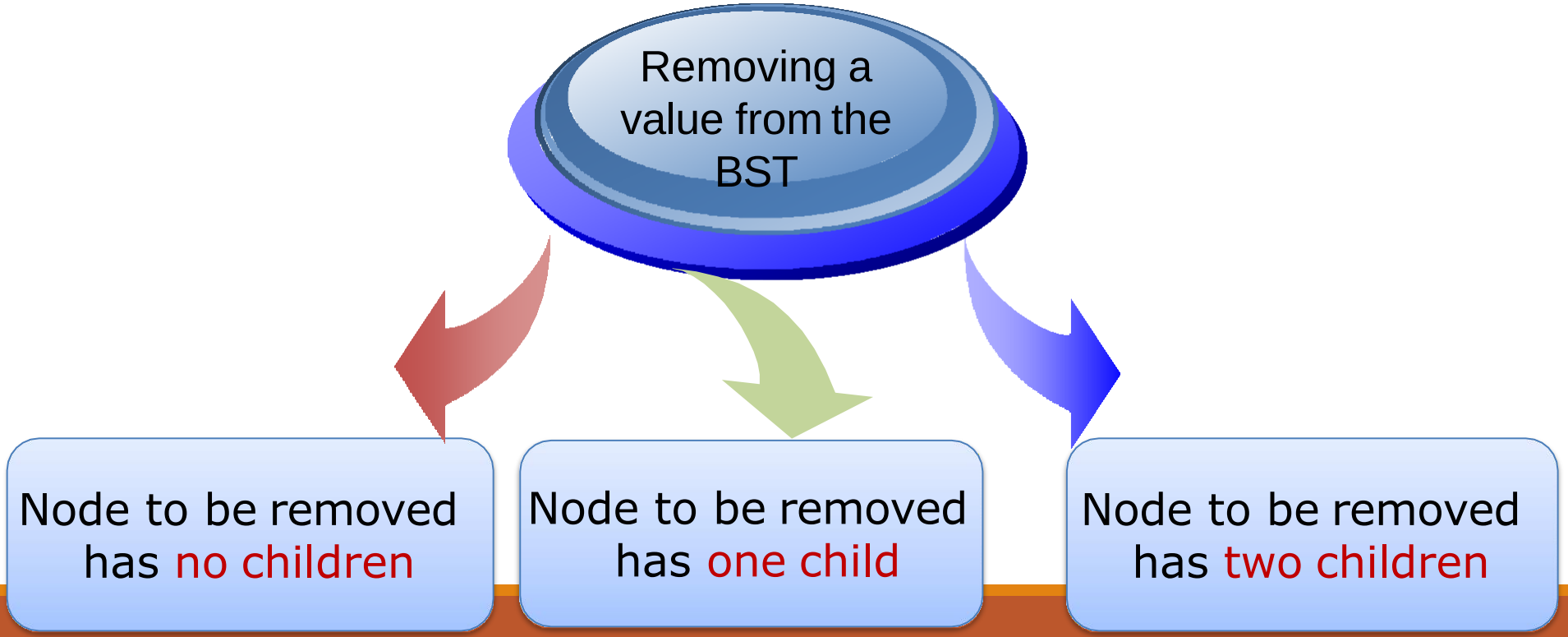
- Insert 4 to the tree:





Removing a value from theBST

- Remove operation on binary search tree is more complicated, than add and search. Basically, in can be divided into two stages:
  1. search for a node to remove;
  2. if the node is found, run remove algorithm.

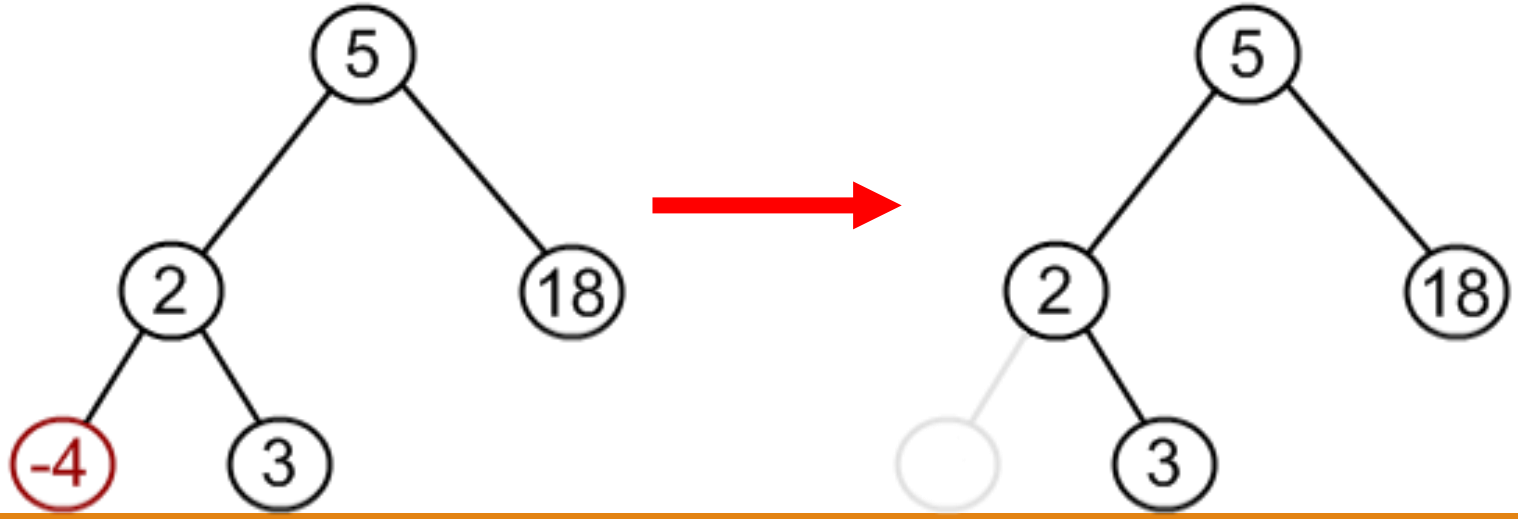


Removing a value from theBST

1. **<u>Node to be removed has no children</u>: (Case 1)**

This case is quite simple. Algorithm setscorresponding link of the parent to NULL and disposes the node.

❑ **<u>Example</u>**: Remove -4 from a BST.

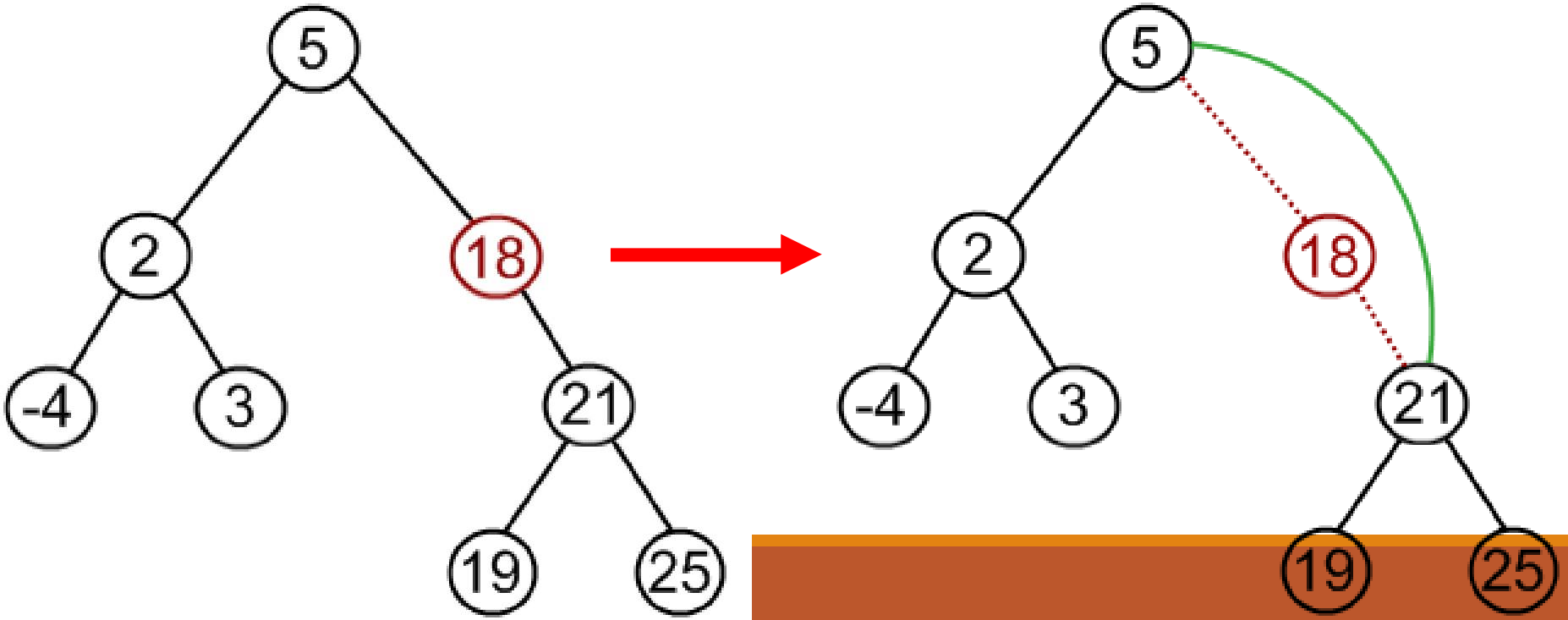


Removing a value from theBST

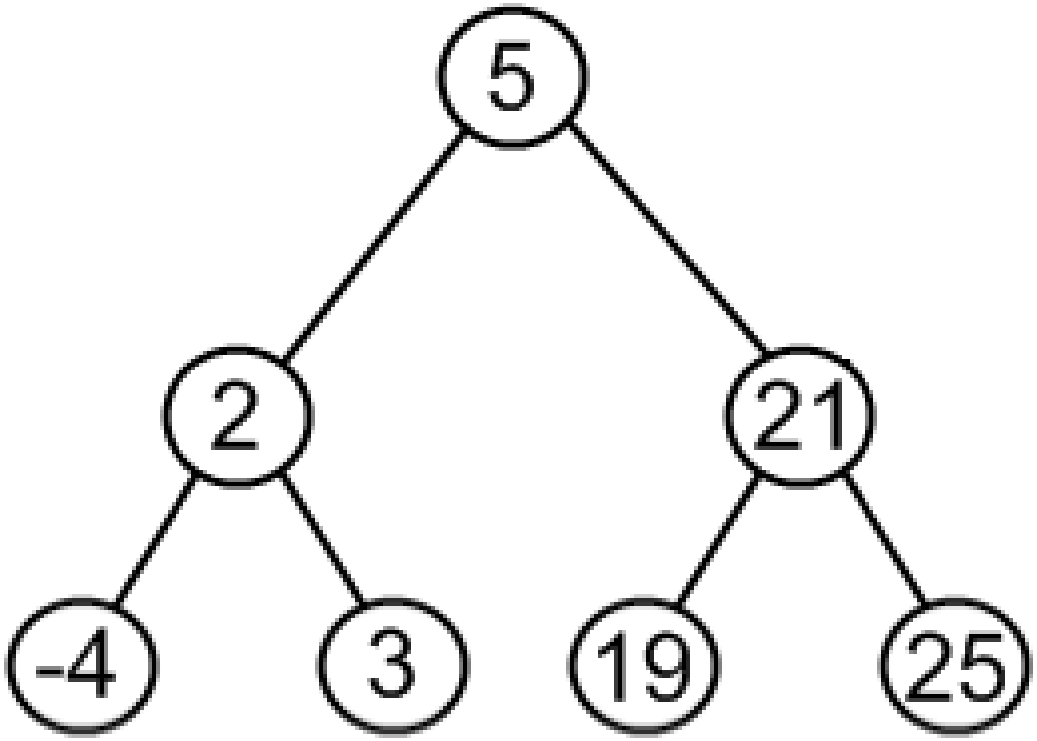
2. <u>Node to be removed has one child:</u> (Case 2)

- It this case, node is cut from the tree and algorithm links single child (with it's sub-tree) directly to the parent of the removed node.

- **<u>Example:</u>** Remove 18 from a BST



Removing a value from theBST

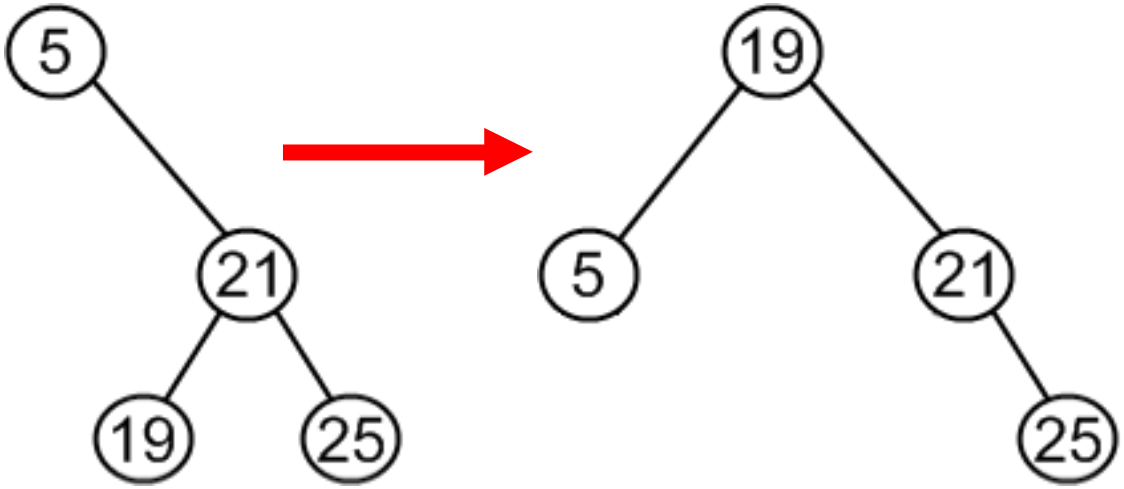


Removing a value from theBST

3. <u>Node to be removed has two children:</u> (Case 3)

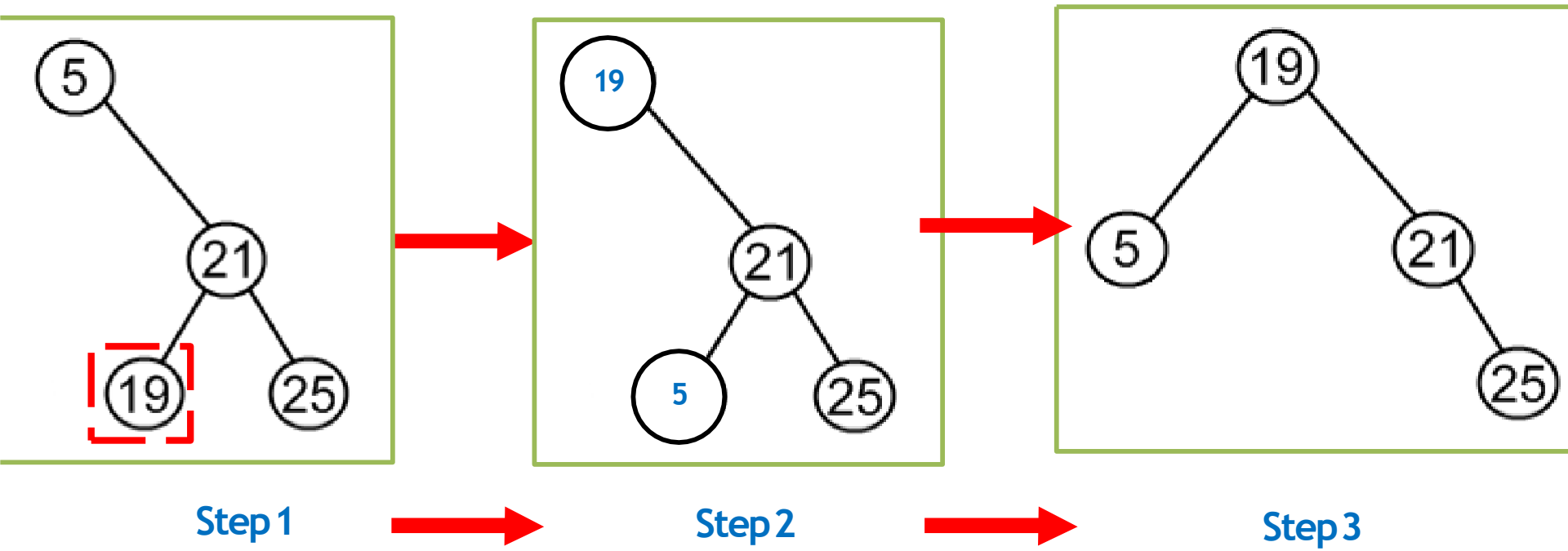
- This is the most complex case. To solve it, let us see one useful BST property first.

- We are going to use the idea, that the same set of values may be represented as different binary-search trees. For example those BSTs:



Two different trees with the samevalues

Removing a value from theBST



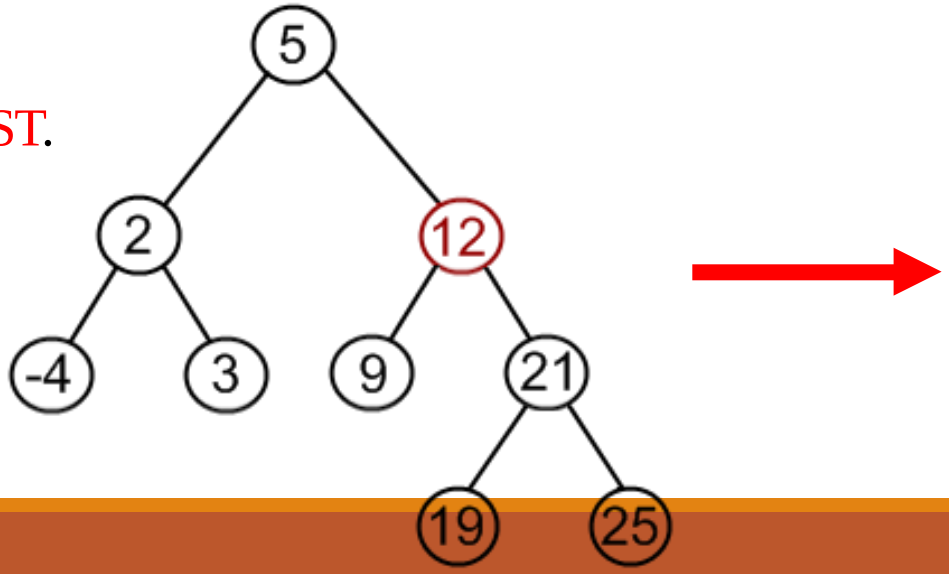
- contains the same values {5, 19, 21, 25}. To transform first tree into second one, we can do following:
  - ✓ **Step 1:** Choose minimum element from the right sub-tree (19 in the example(
  - ✓ **Step 2:** Replace 5 by 19;
  - ✓ **Step 3:** Hang 5 as a left child.

Removing a value from theBST

- The same approach can be utilized to remove a node, which has two children:
  - Find a minimum value in the right sub-tree;
  - Replace value of the node to be removed with found minimum. Now, right sub-tree contains a duplicate!
  - Apply remove to the right sub-tree to remove a duplicate.

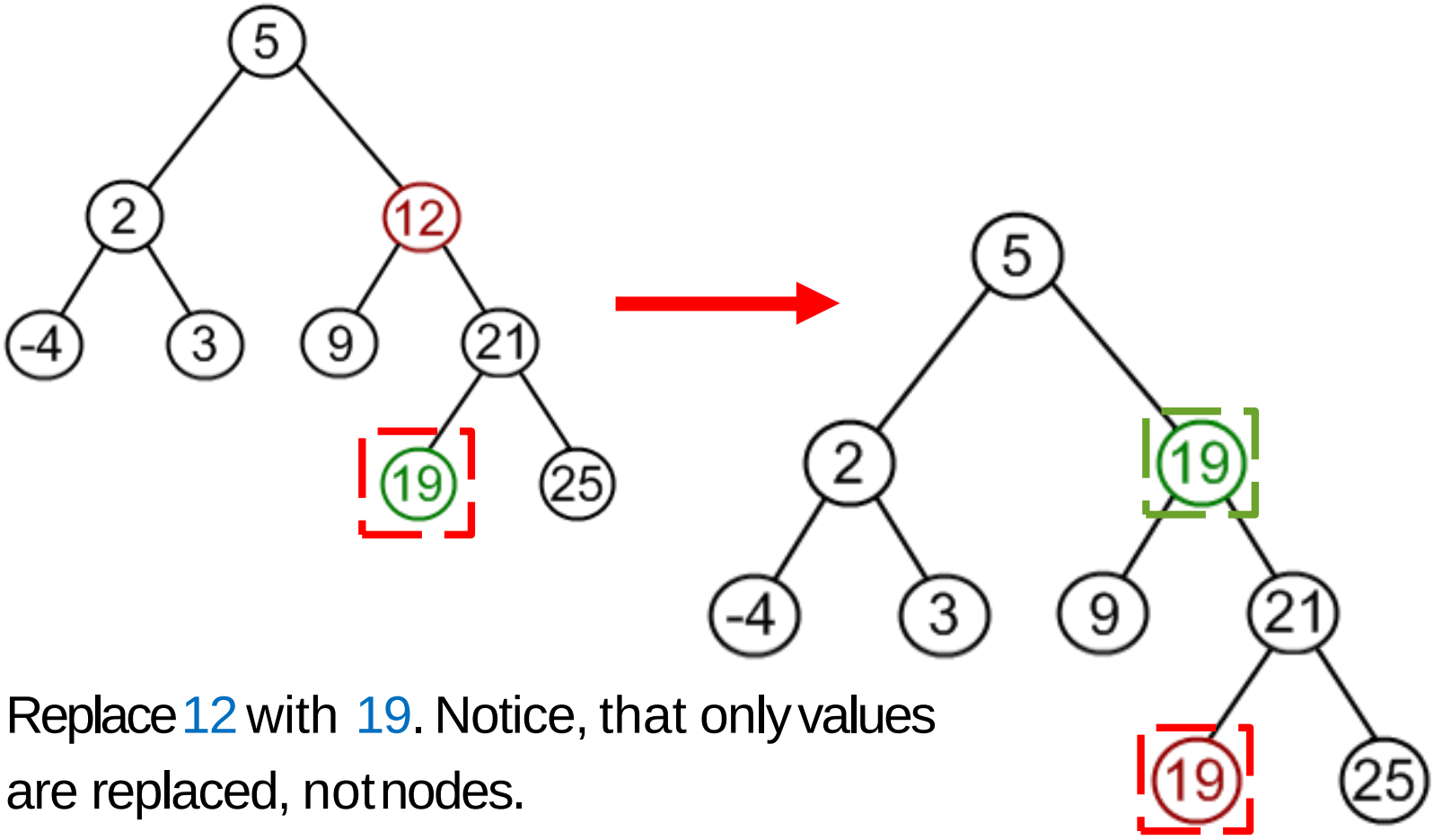
- Notice, that the node with minimum value has no left child and, therefore, it's removal may result in first or second cases only.

- Example. Remove 12 from a BST.



Removing a value from theBST

- Find minimum element in the right sub-tree of the node to be removed. In current example it is19

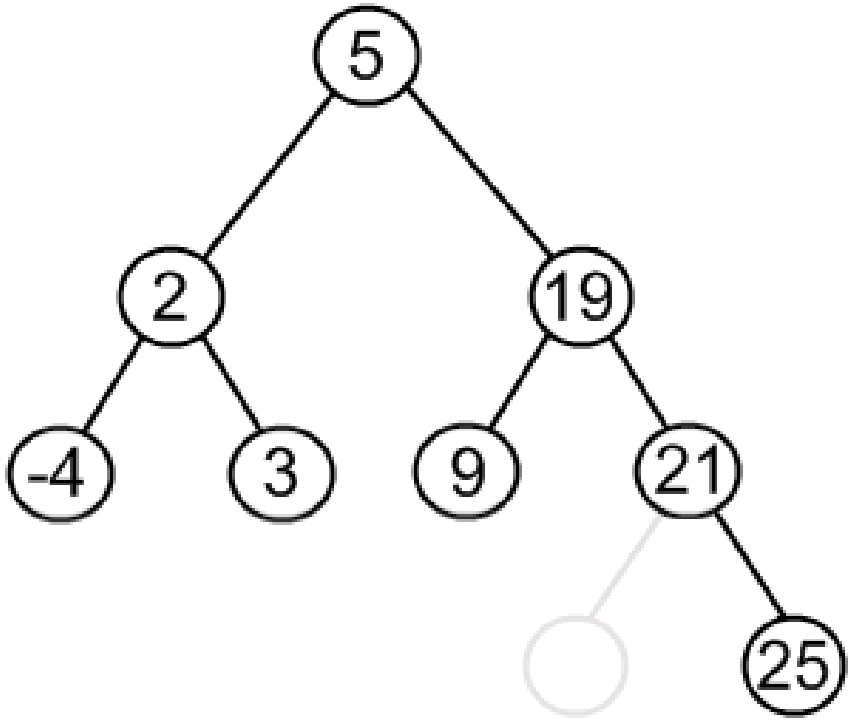


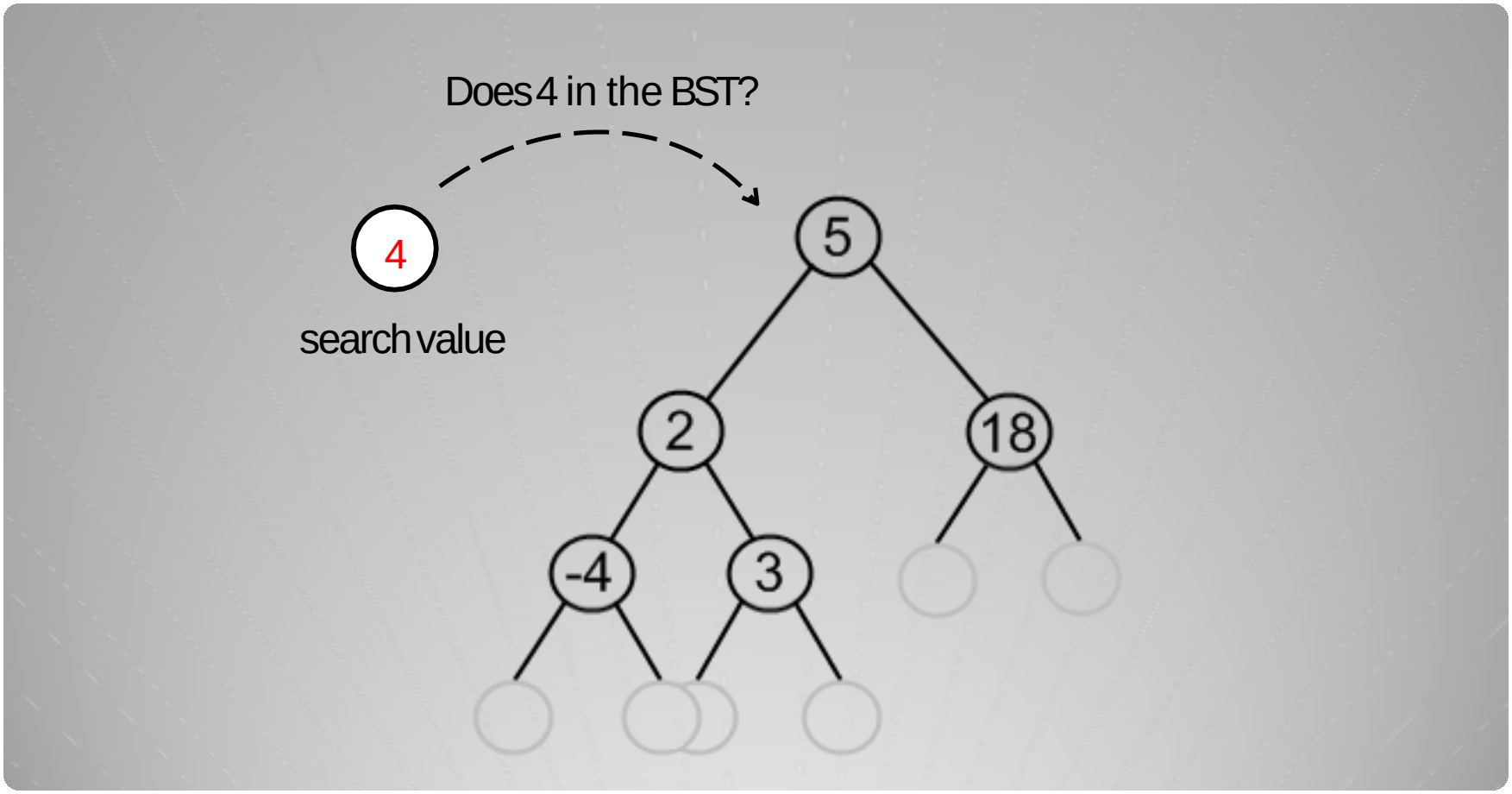
- Replace 12 with 19. Notice, that only values are replaced, not nodes.

- Now we have two nodes with the samevalue.

Removing a value from theBST

- Remove 19 from the left sub-tree.





Searching for a value in the BST

Searching for a value in theBST

- Searching for a value in a BST is very similar to add operation.

- Search algorithm traverses the tree "in-depth", choosing appropriate way to go, following binary search tree property and compares value of each visited node with the one, we are looking for.

- Algorithm stops in two cases:
  - A node with necessary value is found;
  - The algorithm has no way to go.

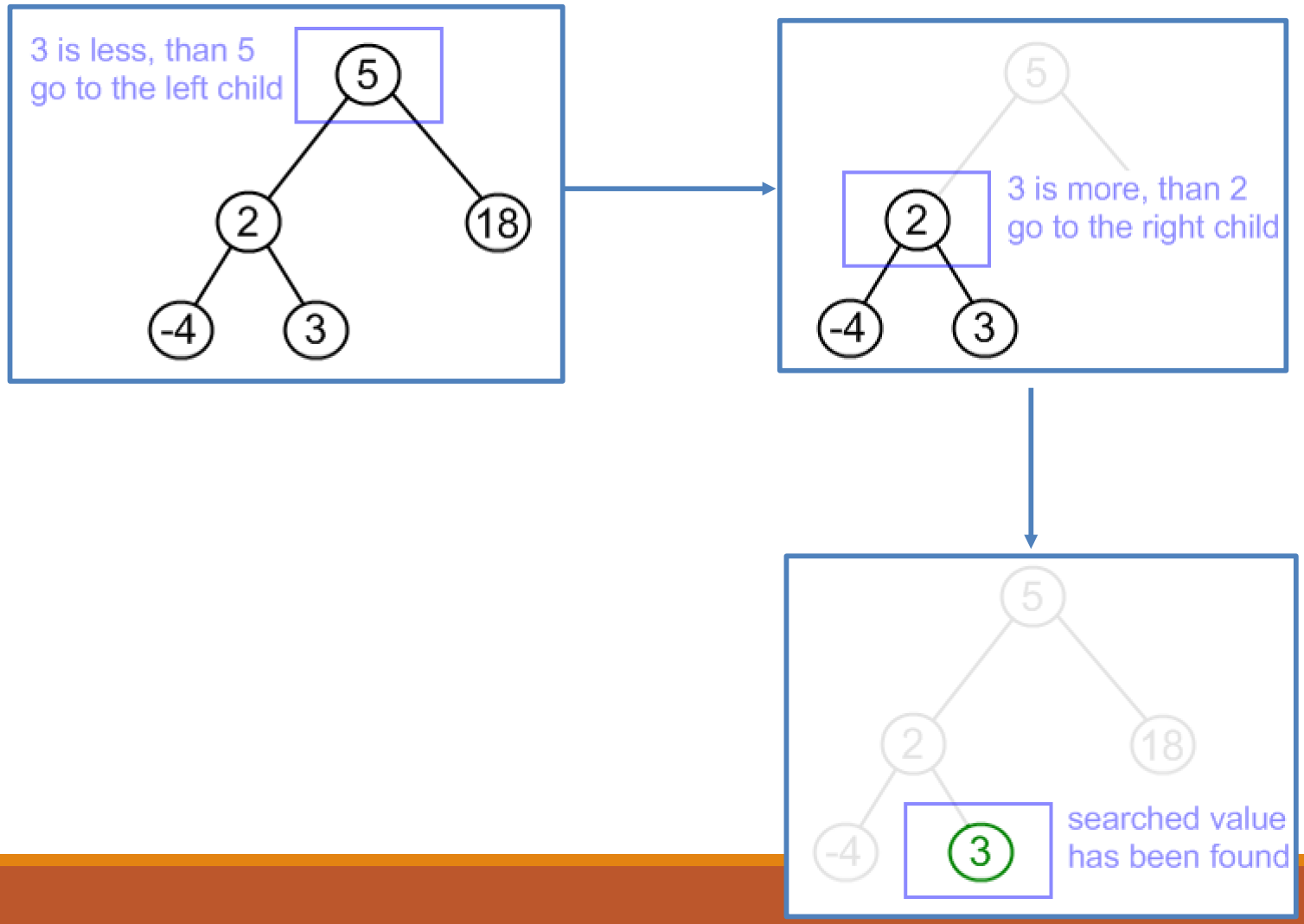
Searching for a value in theBST

- Search algorithm: Like an add operation, and almost every operation on BST, search algorithm utilizes recursion. Starting from the root,

1. check, whether value in current node and searched value are equal. If so, value is found. Otherwise,
2. if searched value is less than the node's value:
    1. if current node has no left child, searched value doesn't exist in the BST;
    2. otherwise, handle the left child with the same algorithm.
3. if a new value is greater than the node's value:
    1. if current node has no right child, searched value doesn't exist in the BST;
    2. otherwise, handle the right child with the same algorithm.

Example for Searching a value in theBST

- Search for 3 in the tree, shown above.



THANK
YOU

Any
Question?