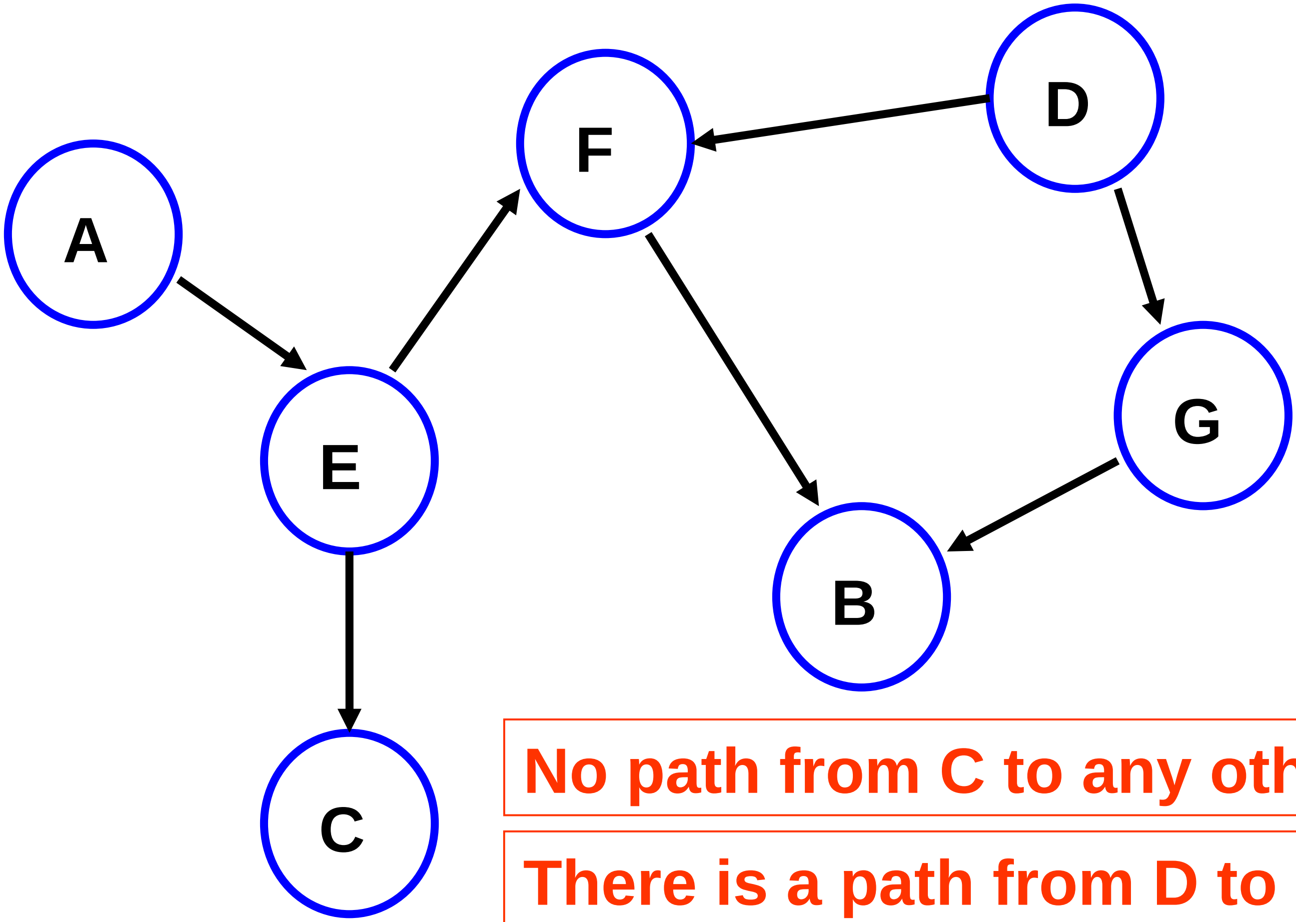
Lec 03

Binary Search Tree

Definition of Tree

- A ***tree*** is a set of linked nodes, such that there is one and only one ***path*** from a unique node (called the ***root*** node) to every other node in the tree.
- A path exists from node A to node B if one can follow a **chain of pointers** to travel from node A to node B.

Paths



No path from C to any other node.

There is a path from D to B.

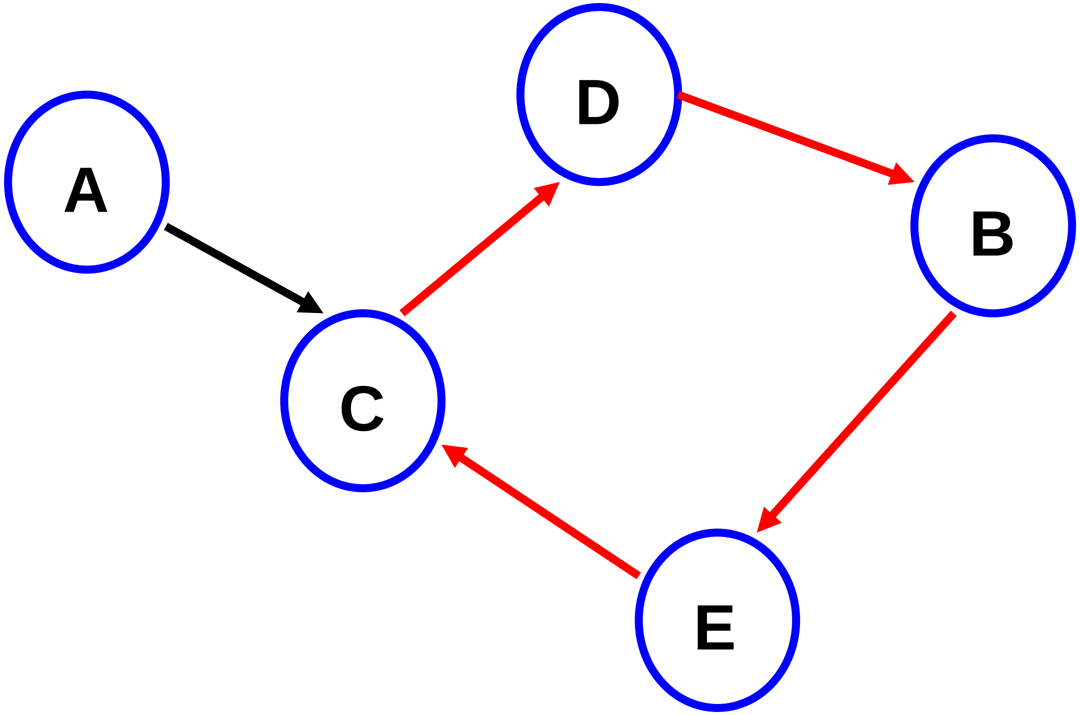
There is also a second path from D to B.

Cycles

- There is no ***cycle*** (circle of pointers) in a tree.

- Any linked structure that has a cycle would have more than one path from the root node to another node.

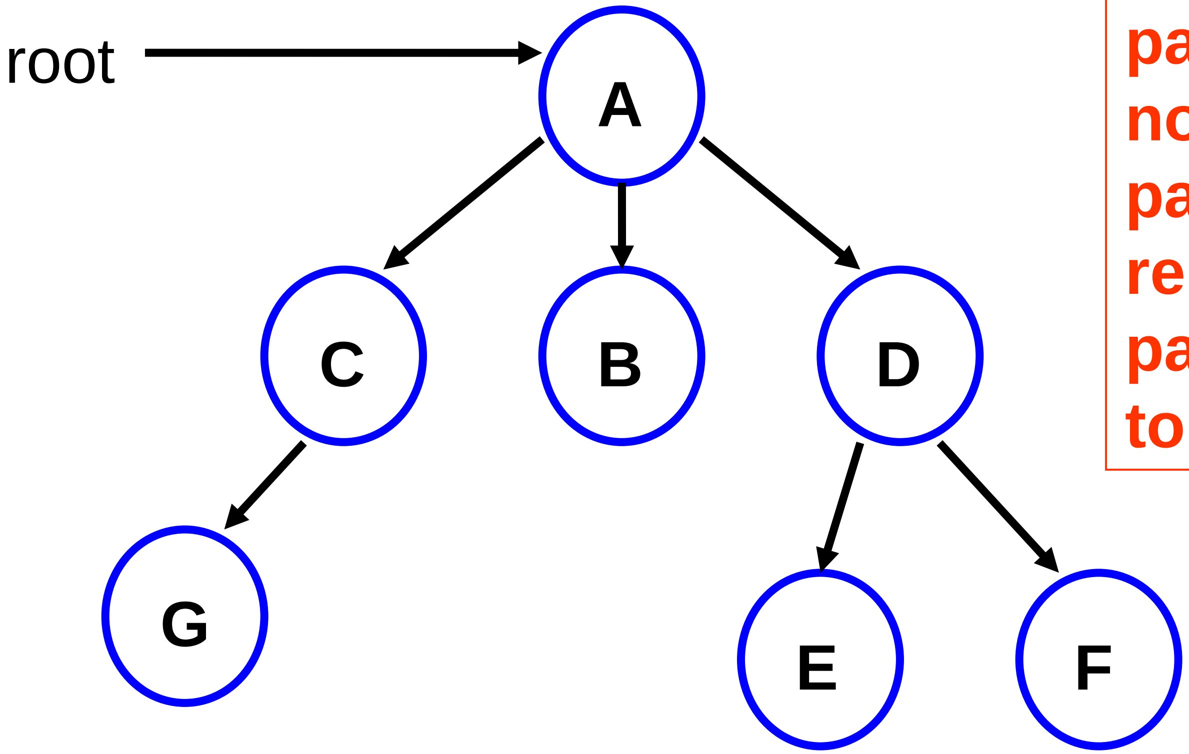
Example of a Cycle



Cycle: C → D → B → E → C

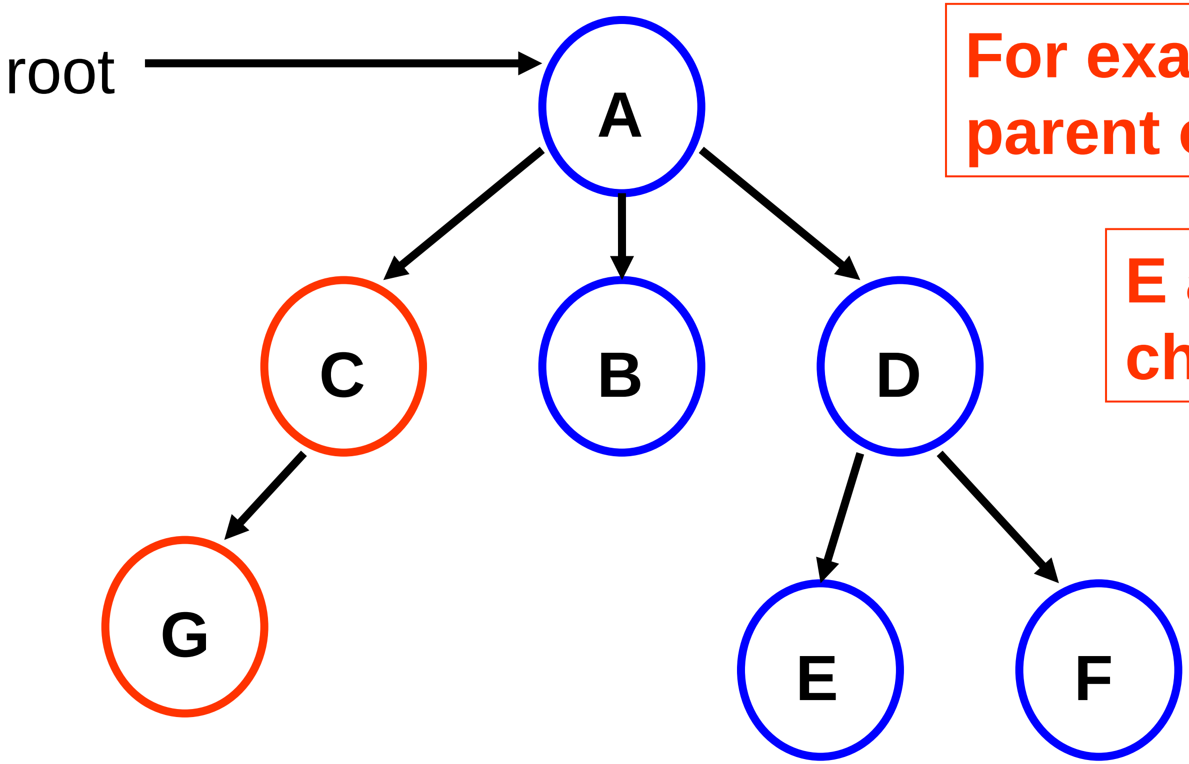
Tree cannot have a cycle.

Example of a Tree



In a tree, every pair of linked nodes have a parent-child relationship (the parent is closer to the root)

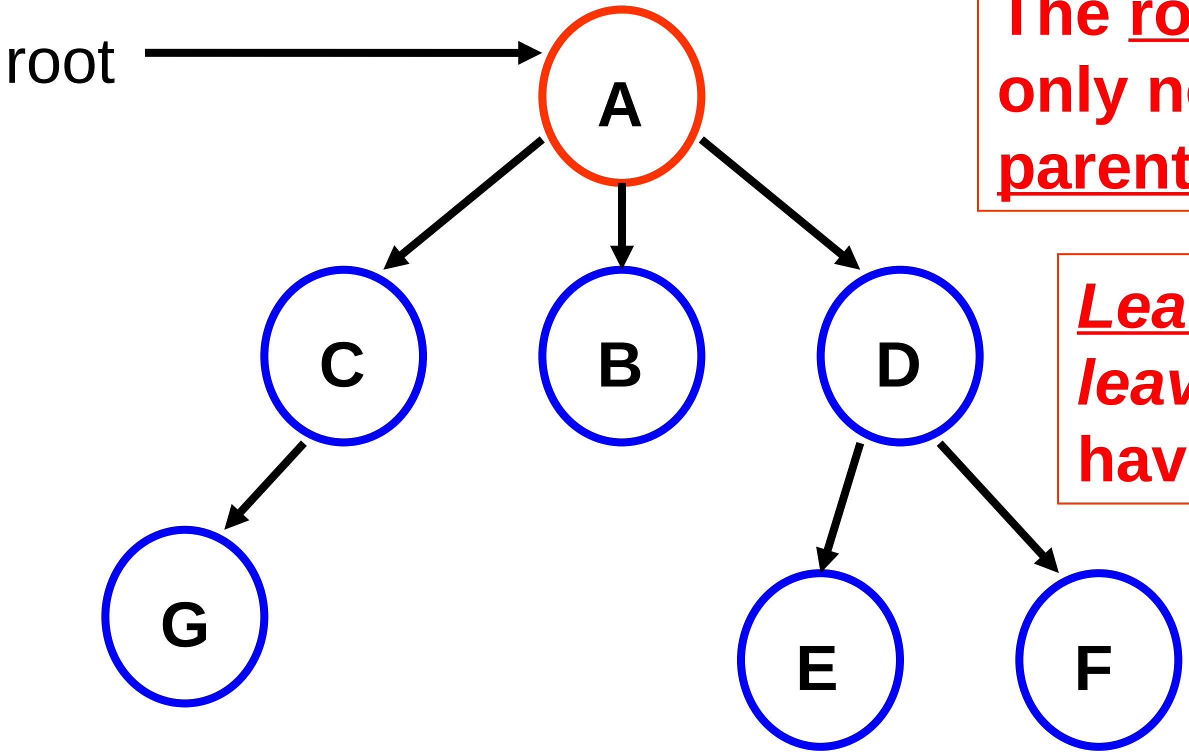
Example of a Tree (cont.)



For example, C is a parent of G

E and F are children of D

Example of a Tree (cont.)



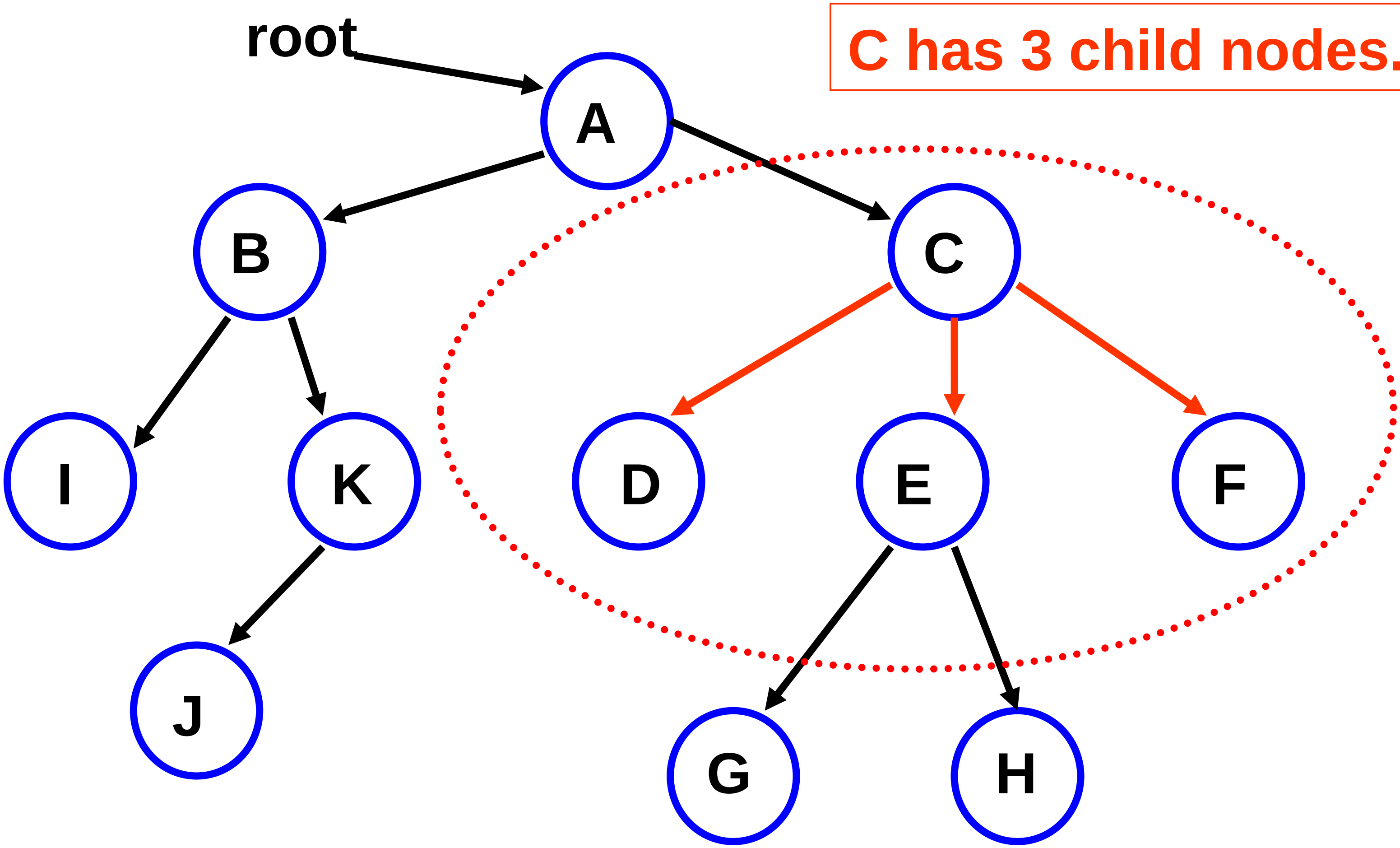
The <u>root</u> node is the only node that has <u>no parent</u>.

<u>Leaf</u> nodes **(or *leaves* for short) have <u>no children</u>.**

Binary Trees

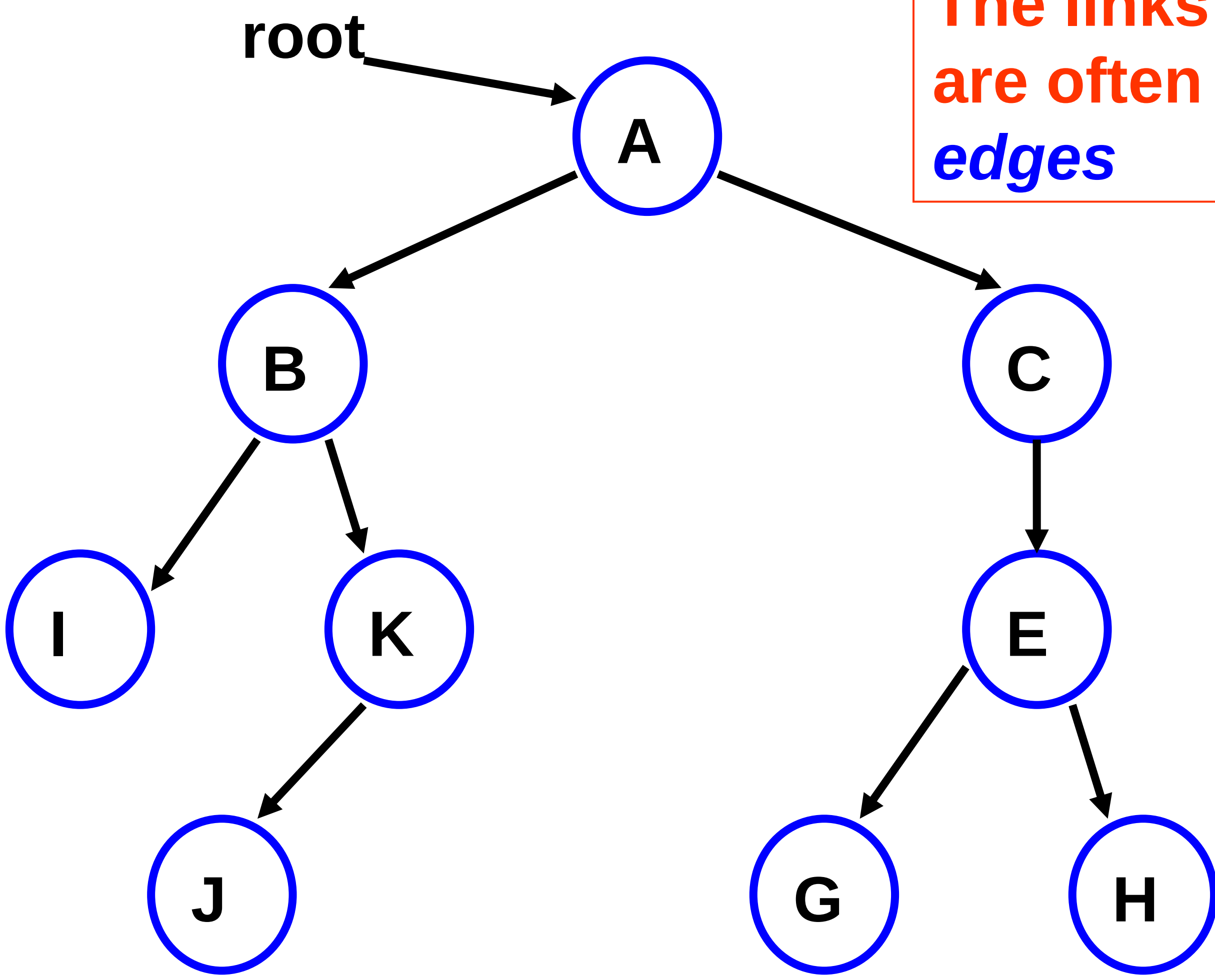
- A ***binary tree*** is a tree in which each node can only have **up to** two children…

NOT a Binary Tree



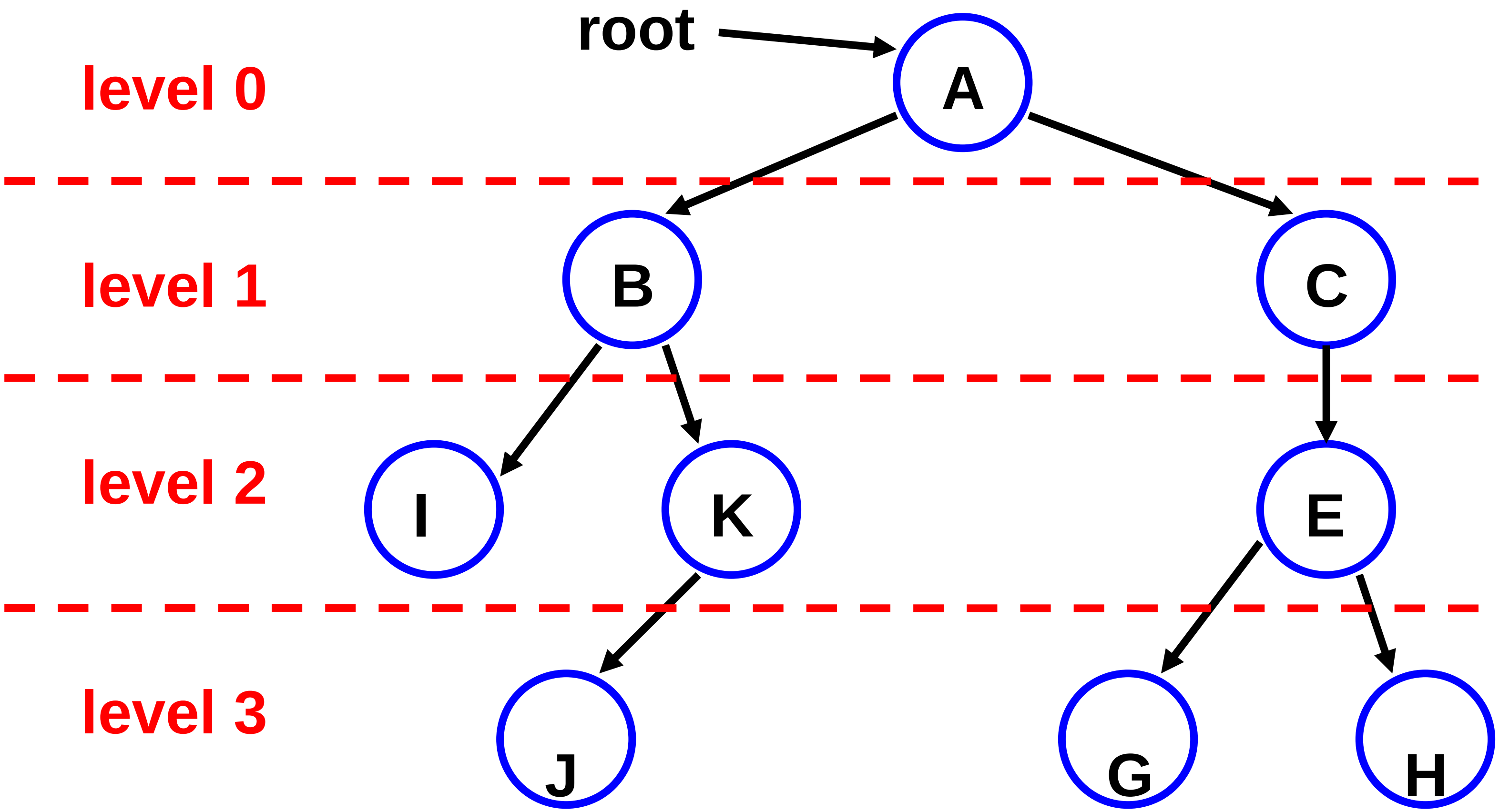
C has 3 child nodes.

Example of a Binary Tree



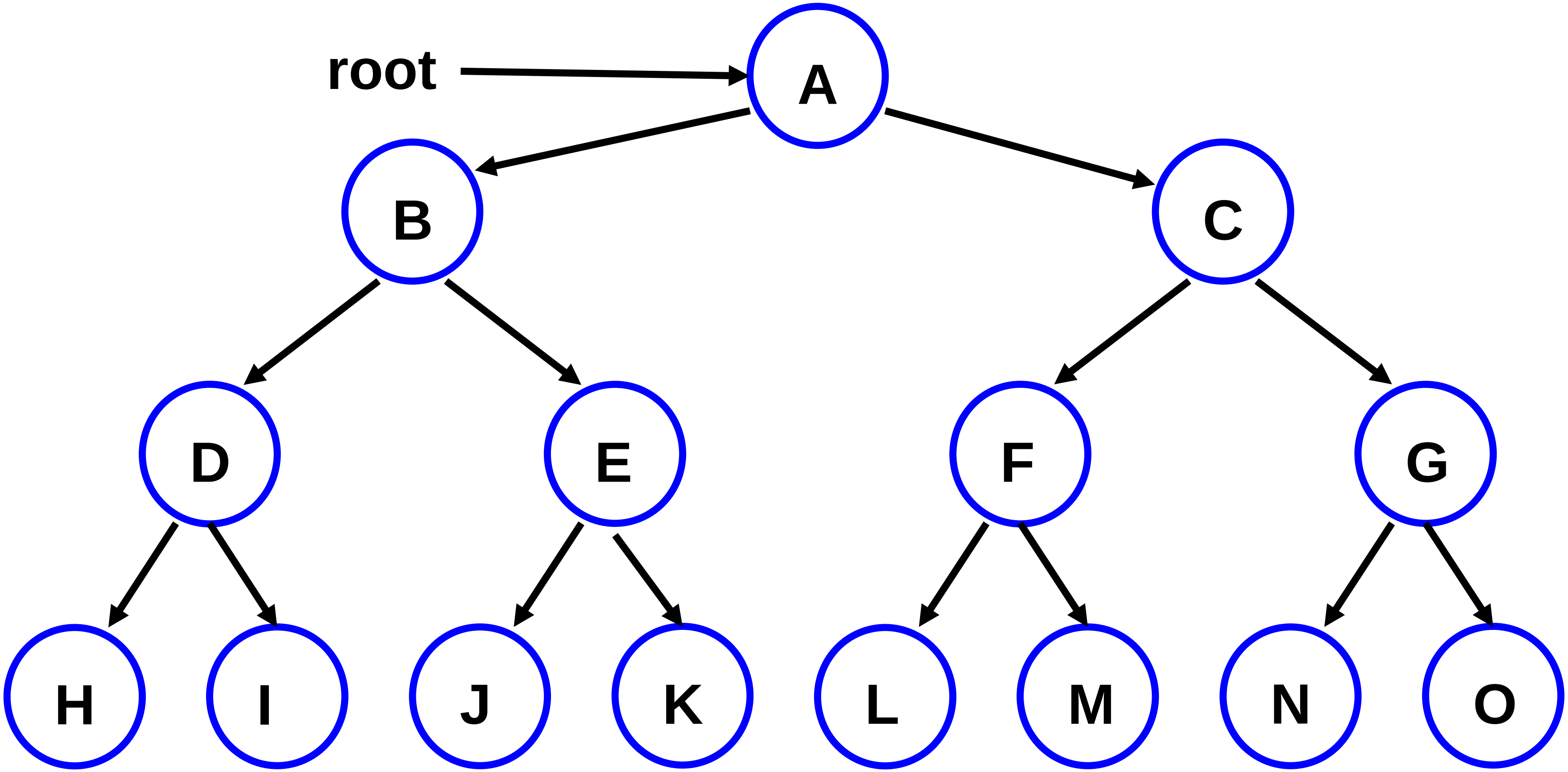
The links in a tree are often called *edges*

Levels



The *level* of a node is the number of edges in the path from the root node to this node

Full Binary Tree

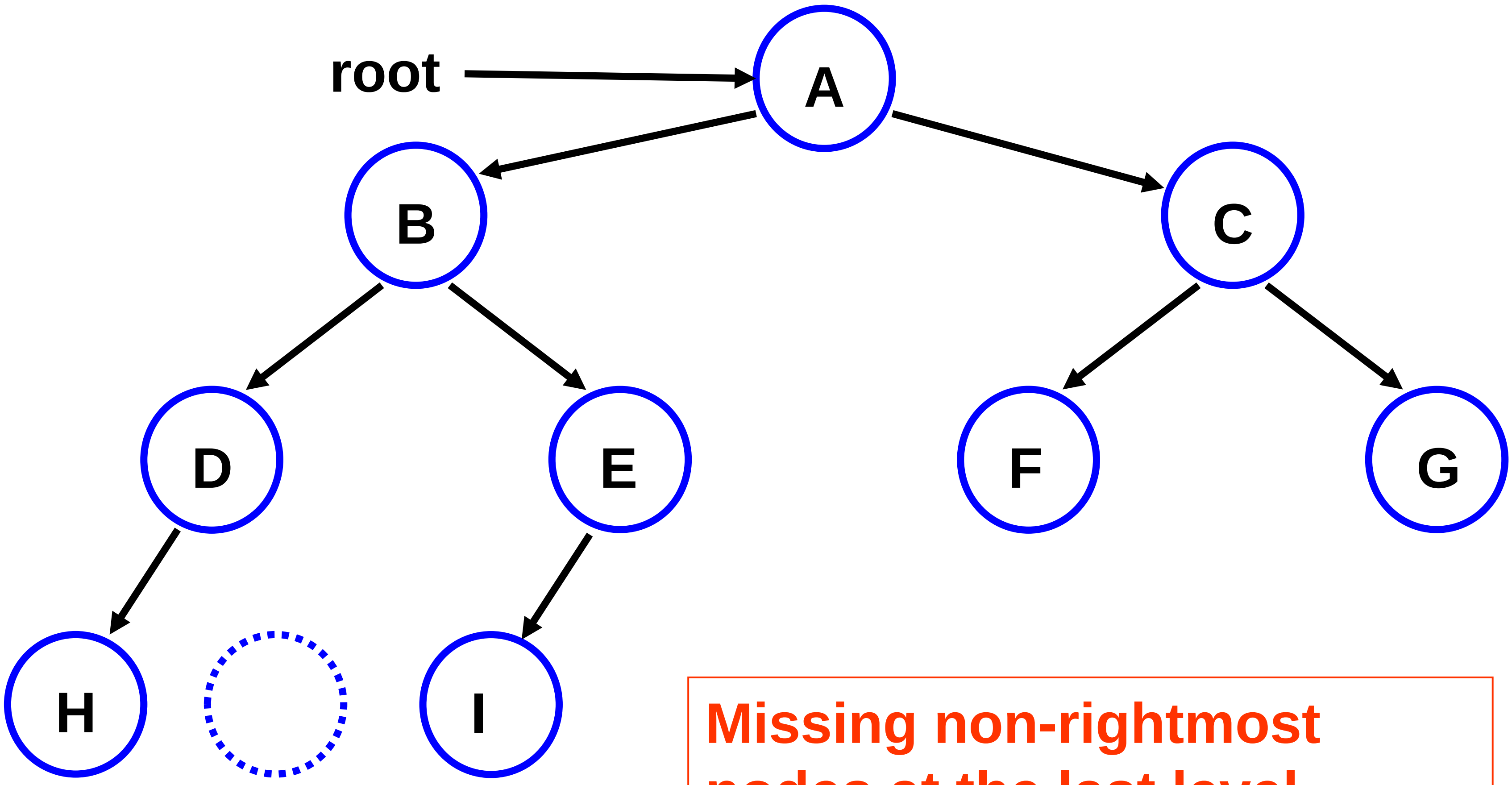


In a ***full binary tree***, each node has two children except for the nodes on the last level, which are leaf nodes

Complete Binary Trees

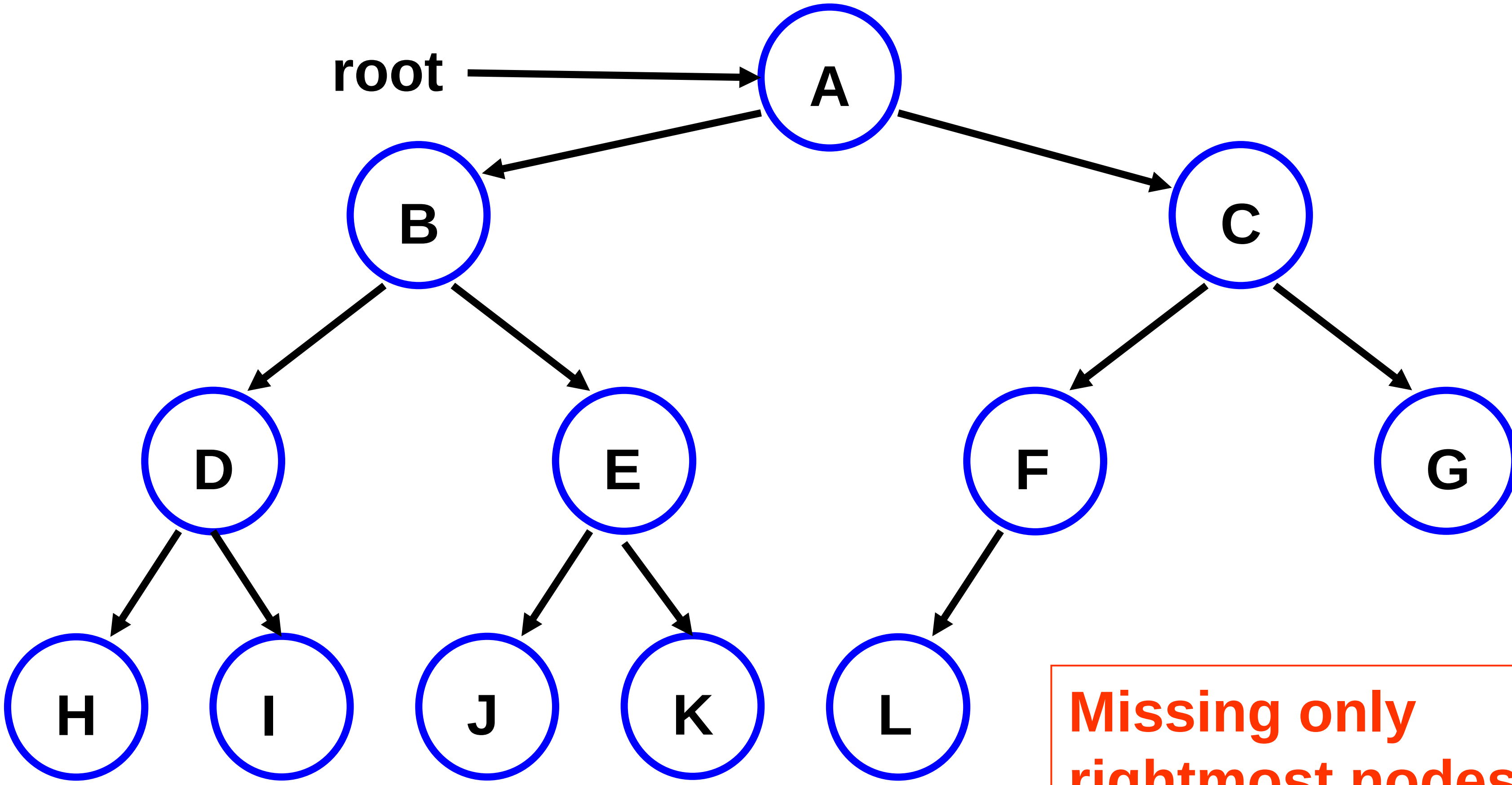
- A ***complete binary tree*** is a binary tree that is either
  - a full binary tree
  - OR
  - a tree that would be a full binary tree but it is missing the rightmost nodes on the last level

NOT a Complete Binary Trees



Missing non-rightmost nodes at the last level

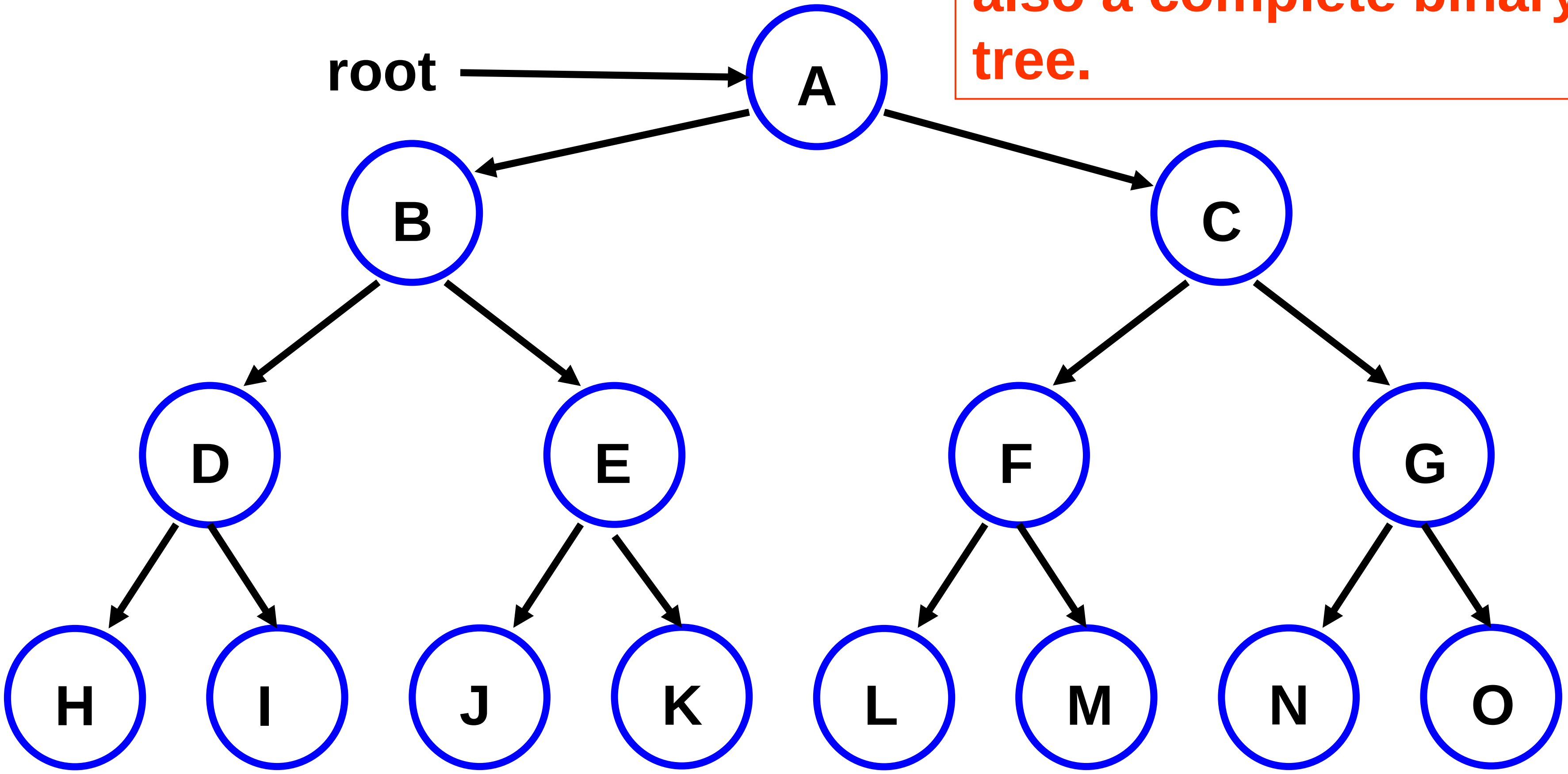
Complete Binary Trees (cont.)



Missing only rightmost nodes at the last level

Complete Binary Trees (cont.)

A full binary tree is also a complete binary tree.



Binary Search Trees

- A ***binary search tree*** is a binary tree that allows us to search for values that can be anywhere in the tree.

- Usually, we search for a certain key value, and once we find the node that contains it, we retrieve the rest of the info at that node.

Properties of Binary Search Trees

- A binary search tree does not have to be a complete binary tree.
- For any particular node,
  - the key in its left child (if any) is less than its key.
  - the key in its right child (if any) is greater than or equal to its key.
- (Left < Parent <= Right) or (Left <= Parent < Right)

Inserting Nodes Into a BST

root:
NULL

BST starts off empty

Objects that need to be inserted (only key values are shown):

37, 2, 45, 48, 41, 29, 20, 30, 49, 7

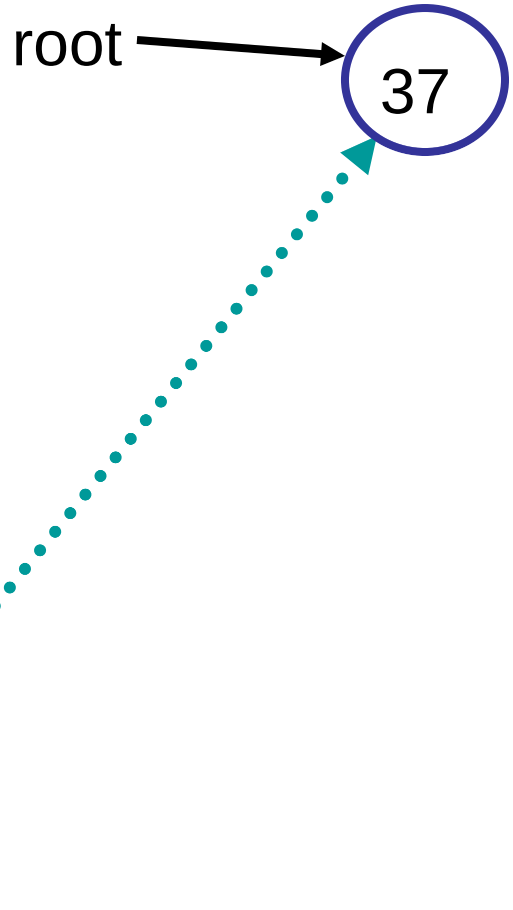
Pseudocode

Algorithm ***insertElement*** **(*p*, *e*)**

Input: ***e*** is the element to be inserted under vertex ***p***, without rotation.

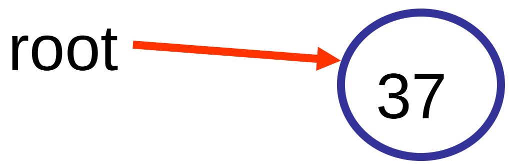
```
if p is null
    p = e
else if (e < p)
    insertElement (leftChild (p), e)
else
    insertElement (rightChild (p), e);
```

Inserting Nodes Into a BST (cont.)



37, 2, 45, 48, 41, 29, 20, 30, 49, 7

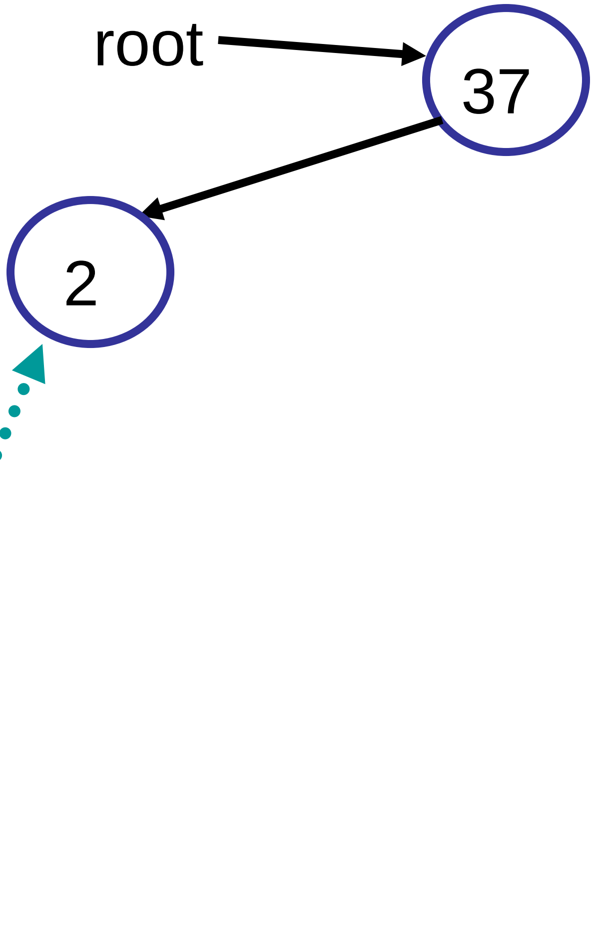
Inserting Nodes Into a BST (cont.)



2 < 37, so insert 2 on the left side of 37

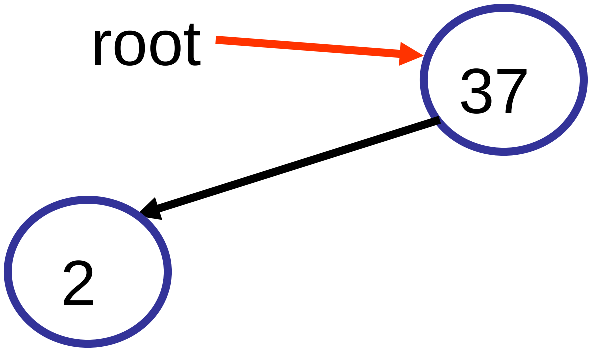
2, 45, 48, 41, 29, 20, 30, 49, 7

Inserting Nodes Into a BST (cont.)



2, 45, 48, 41, 29, 20, 30, 49, 7

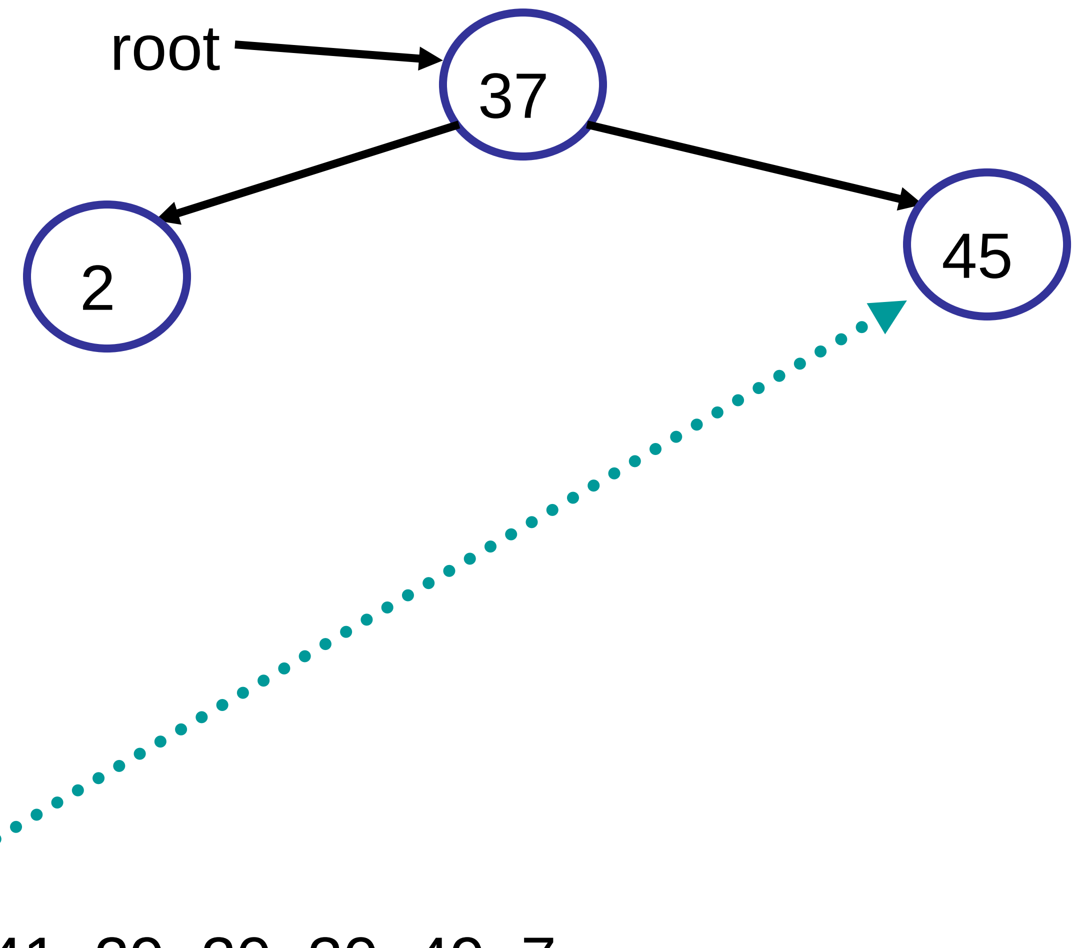
Inserting Nodes Into a BST (cont.)



45 > 37, so insert it at the right of 37

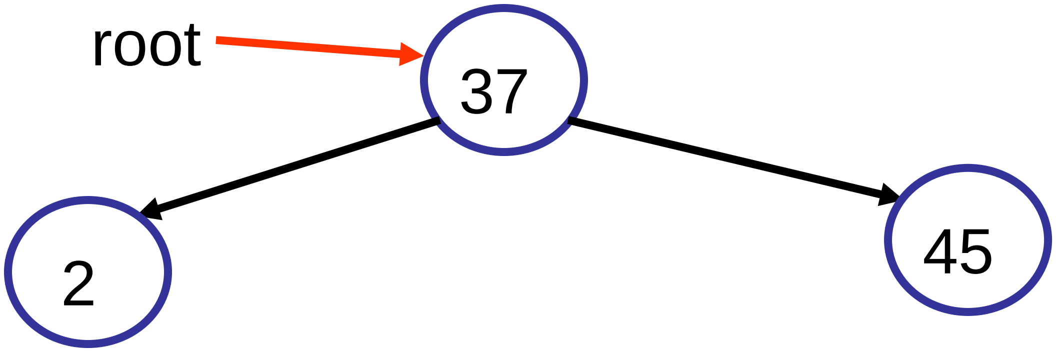
45, 48, 41, 29, 20, 30, 49, 7

Inserting Nodes Into a BST (cont.)



45, 48, 41, 29, 20, 30, 49, 7

Inserting Nodes Into a BST (cont.)

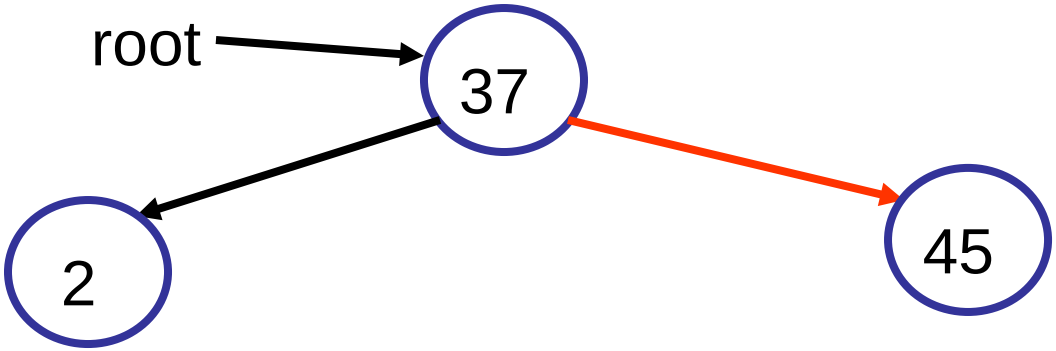


When comparing, we always start at the root node

48 > 37, so look to the right

48, 41, 29, 20, 30, 49, 7

Inserting Nodes Into a BST (cont.)

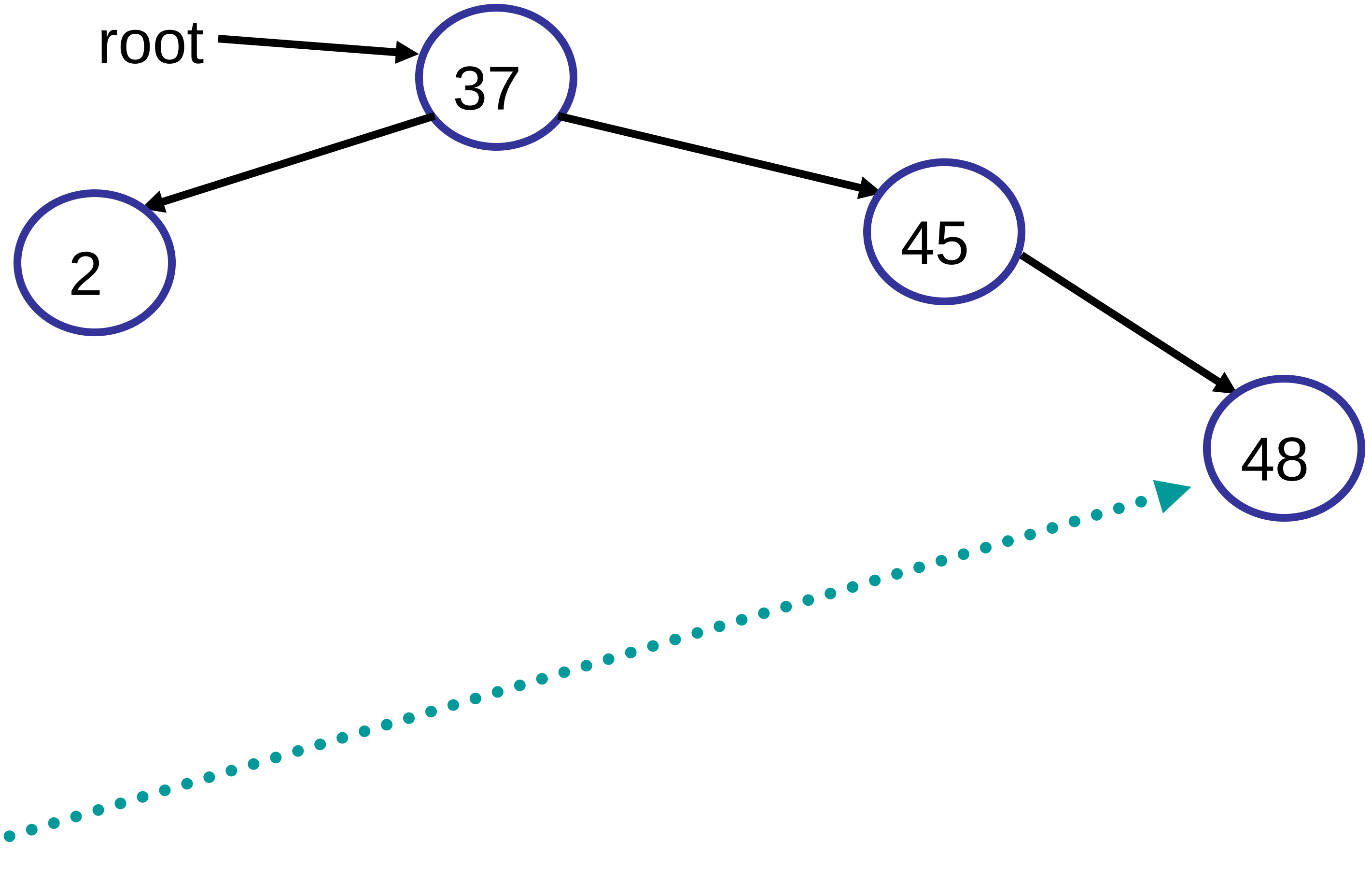


This time, there is a node already to the right of the root node. We then compare 48 to this node

48 > 45, and 45 has no right child, so we insert 48 on the right of 45

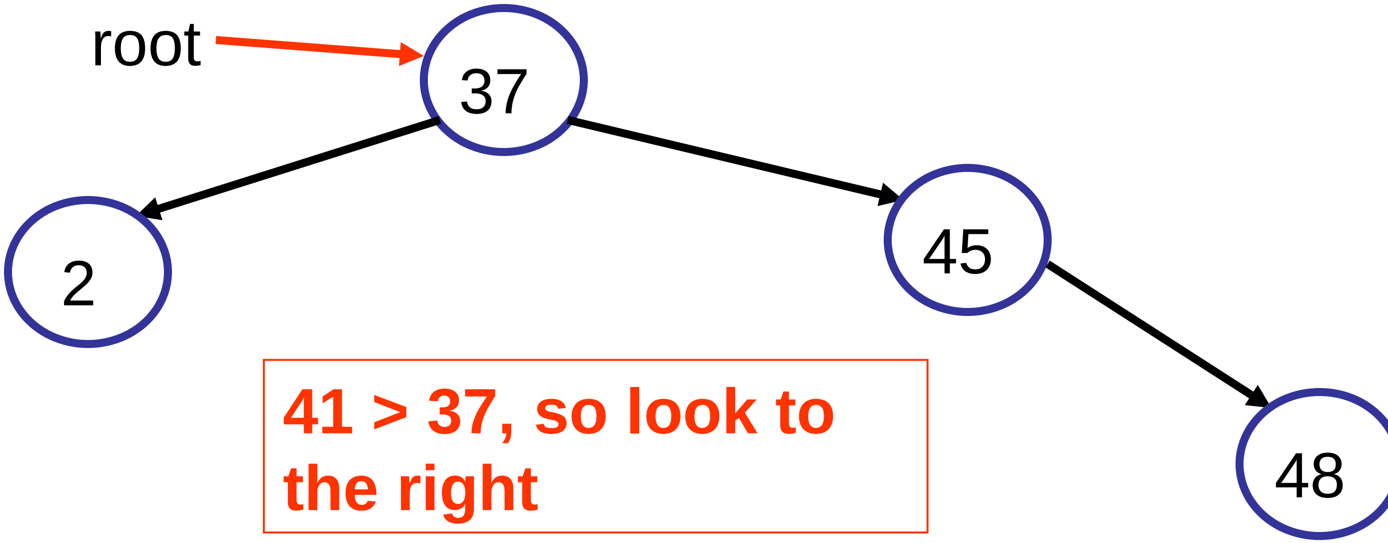
48, 41, 29, 20, 30, 49, 7

Inserting Nodes Into a BST (cont.)



48, 41, 29, 20, 30, 49, 7

Inserting Nodes Into a BST (cont.)

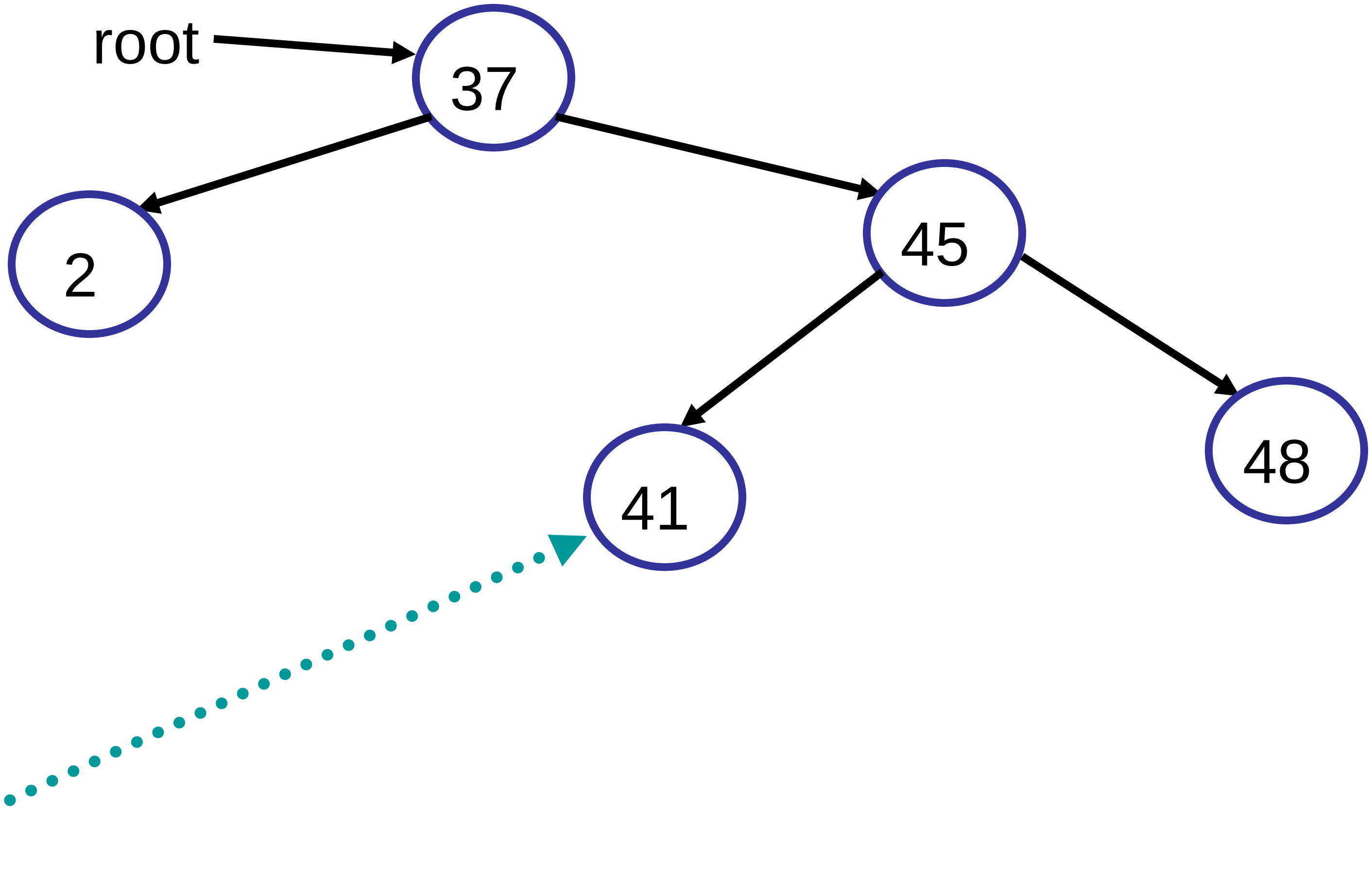


41 > 37, so look to the right

41 < 45, so look to the left – there is no left child, so insert

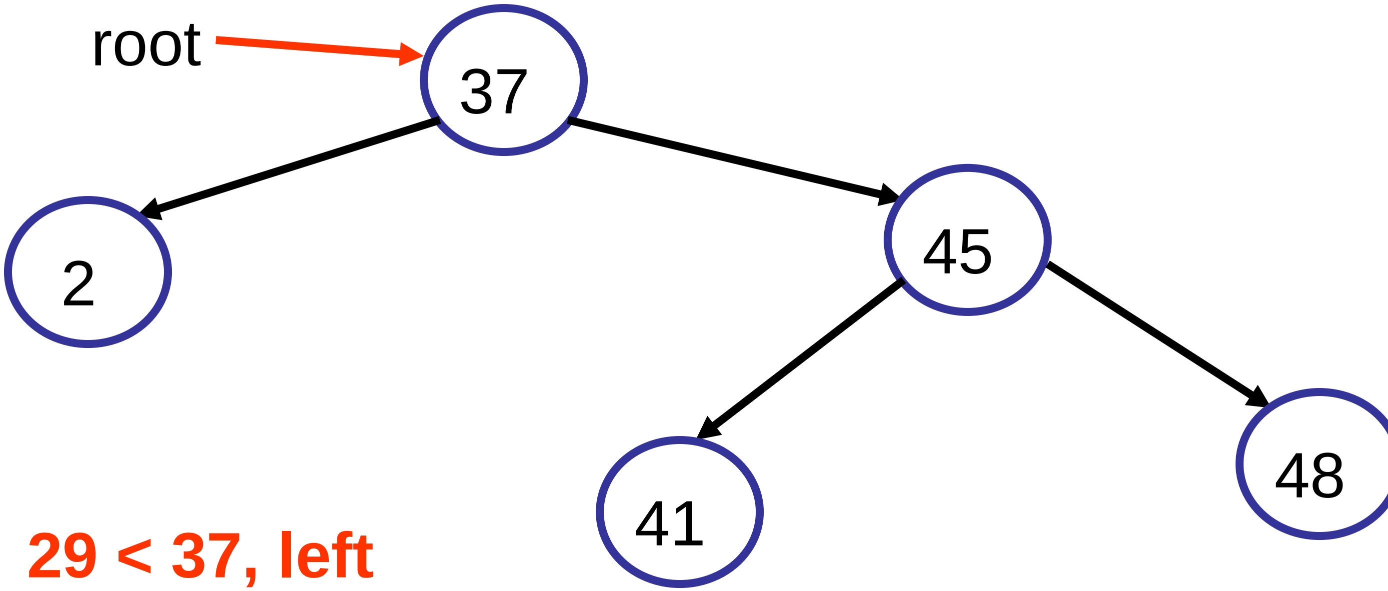
41, 29, 20, 30, 49, 7

Inserting Nodes Into a BST (cont.)



41, 29, 20, 30, 49, 7

Inserting Nodes Into a BST (cont.)

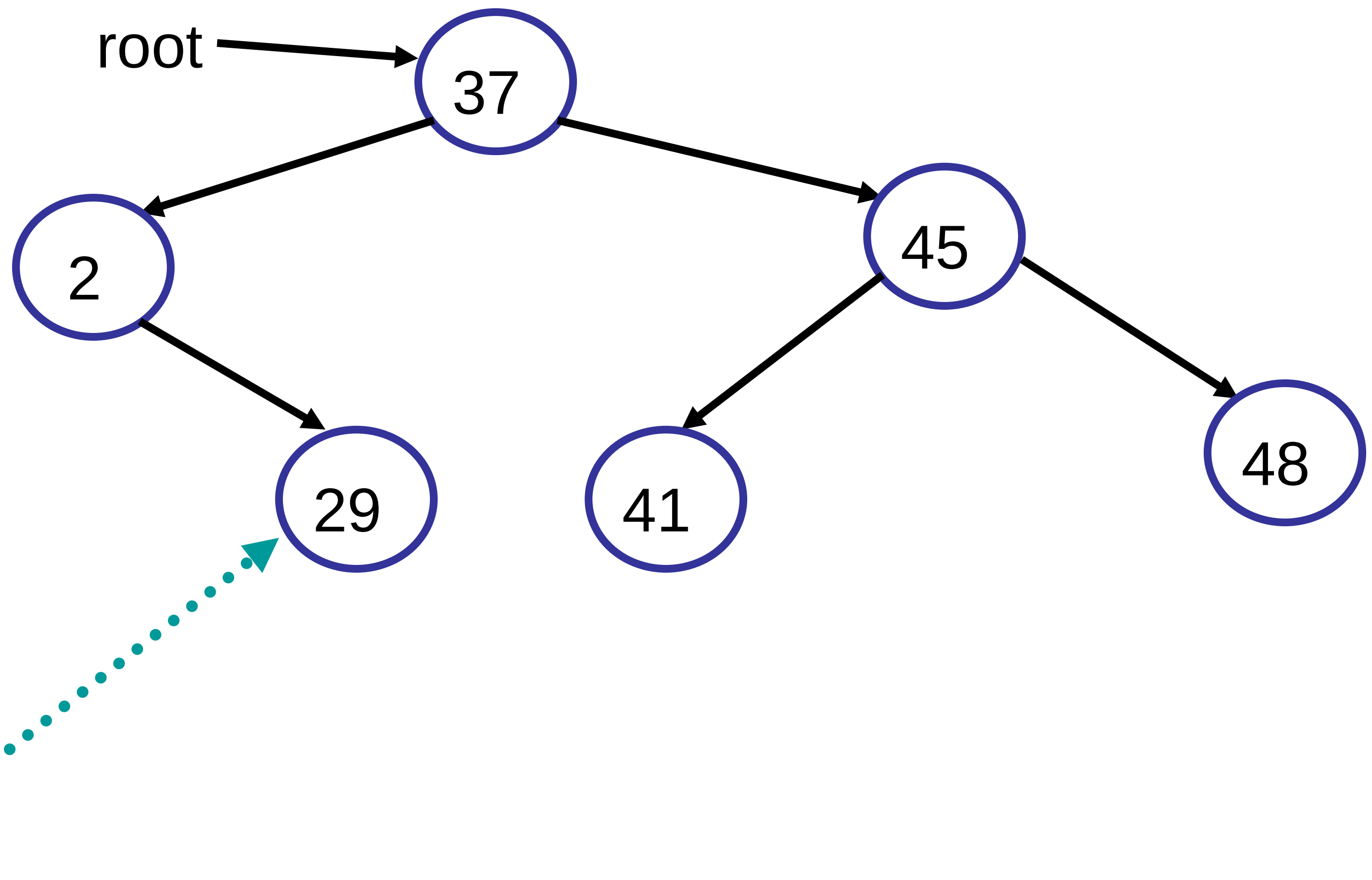


29 < 37, left

29 > 2, right

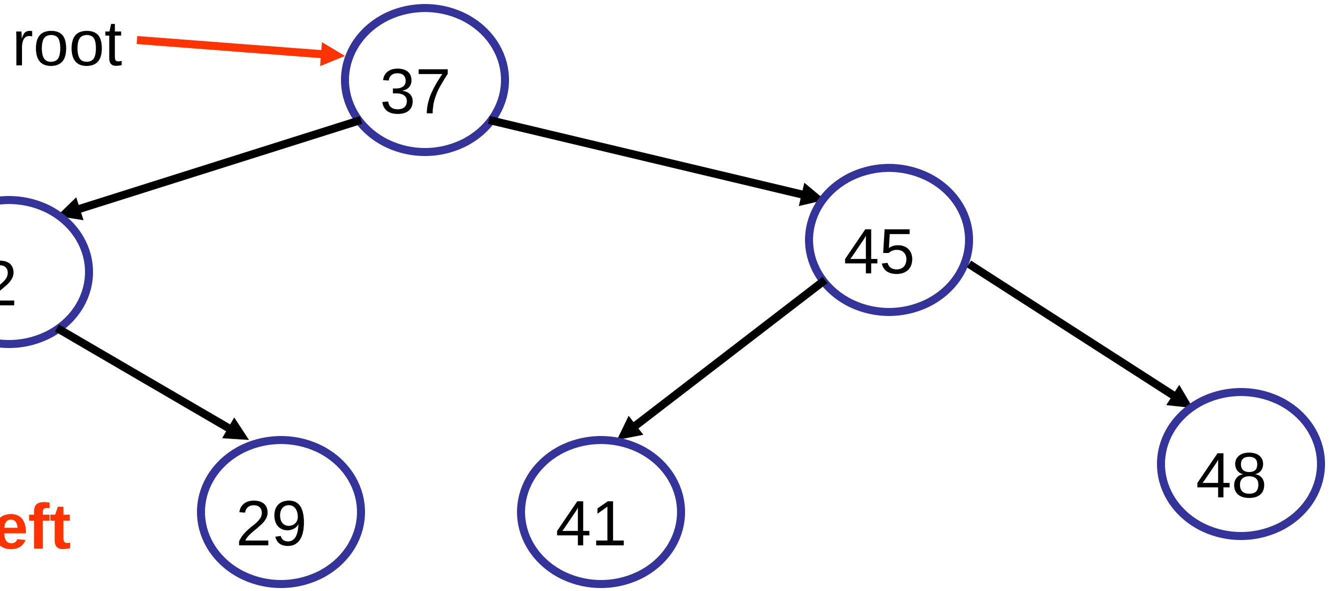
29, 20, 30, 49, 7

Inserting Nodes Into a BST (cont.)



29, 20, 30, 49, 7

Inserting Nodes Into a BST (cont.)



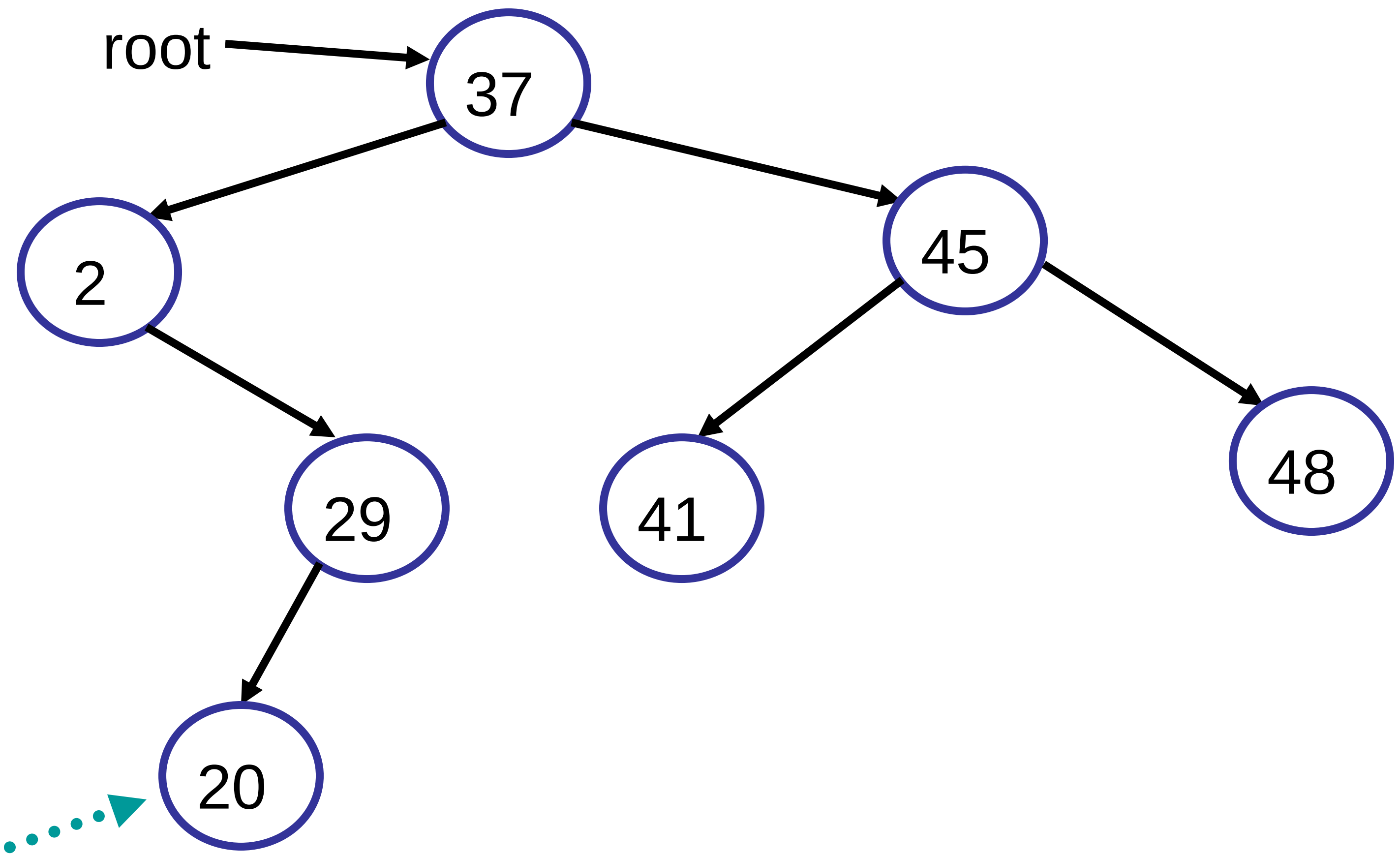
20 < 37, left

20 > 2, right

20 < 29, left

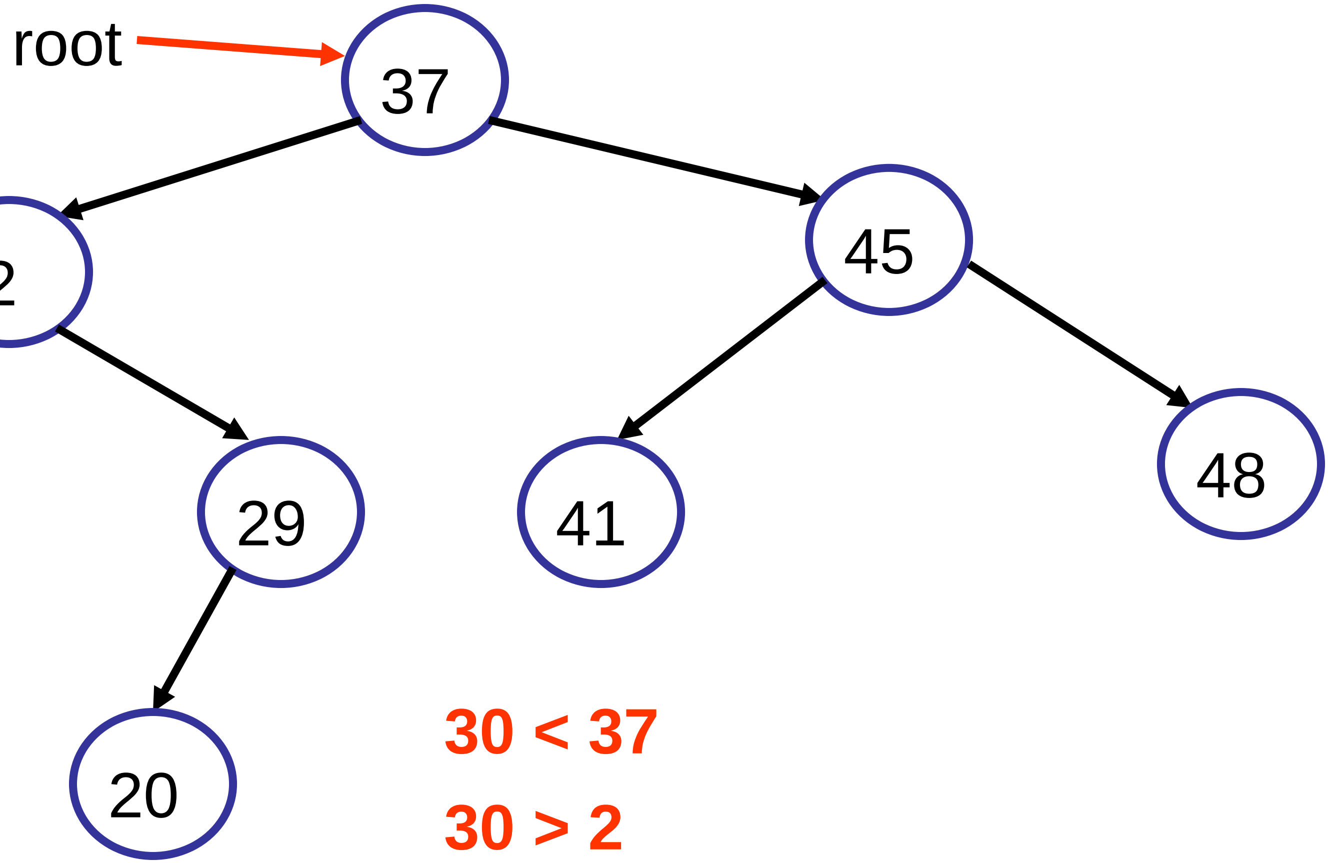
20, 30, 49, 7

Inserting Nodes Into a BST (cont.)



20, 30, 49, 7

Inserting Nodes Into a BST (cont.)



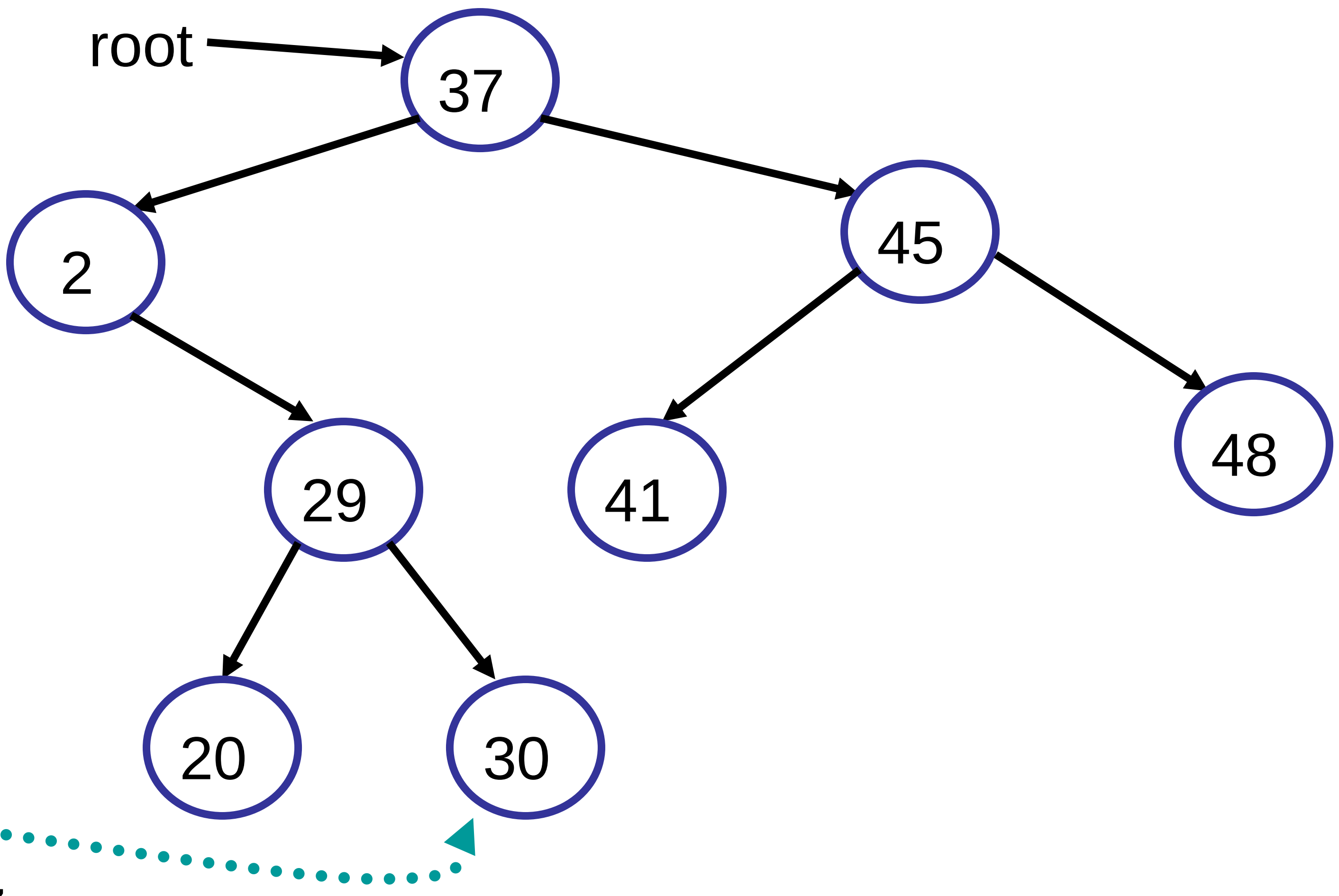
30 < 37

30 > 2

30 > 29

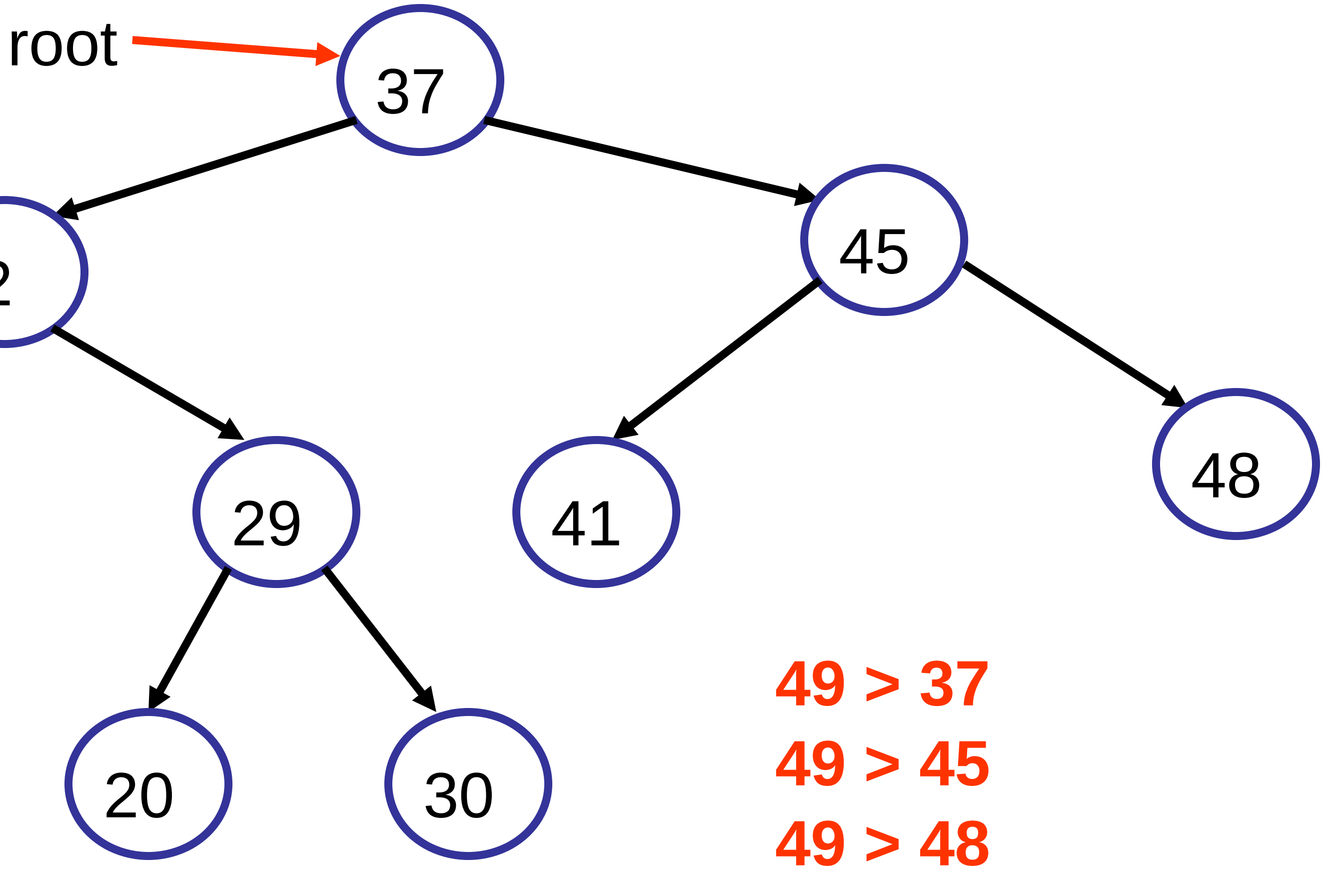
30, 49, 7

Inserting Nodes Into a BST (cont.)



30, 49, 7

Inserting Nodes Into a BST (cont.)



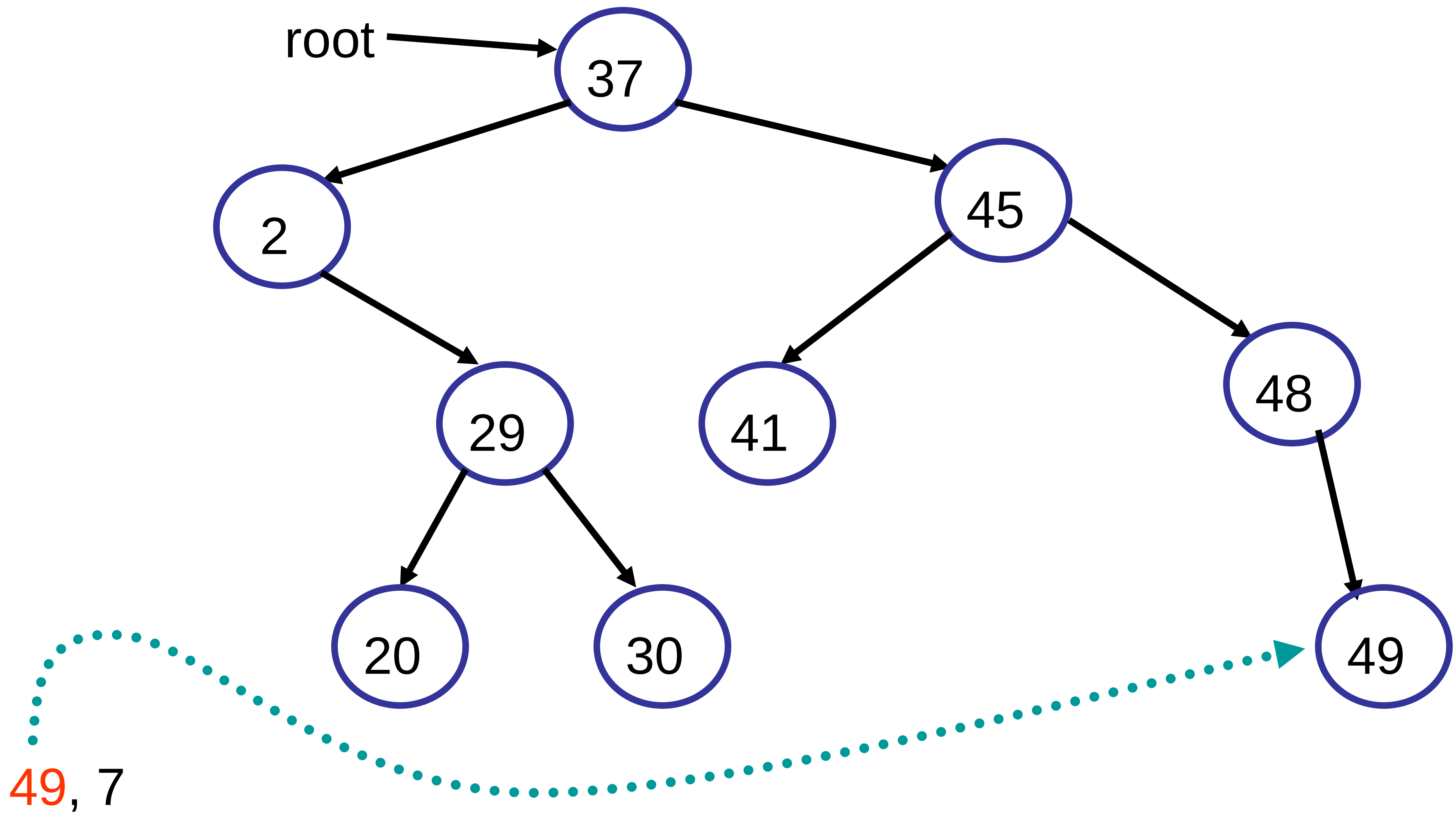
49 > 37

49 > 45

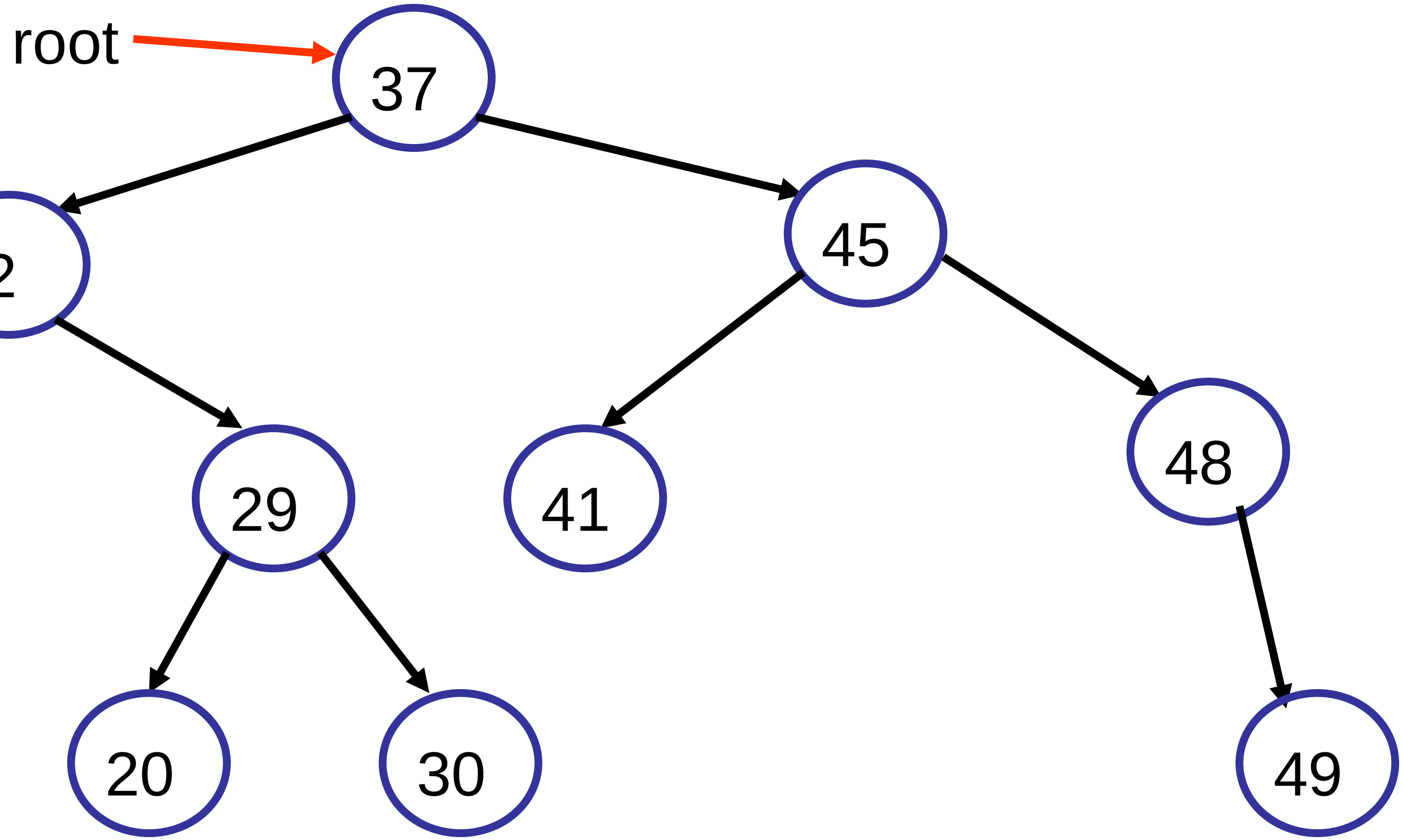
49 > 48

49, 7

Inserting Nodes Into a BST (cont.)



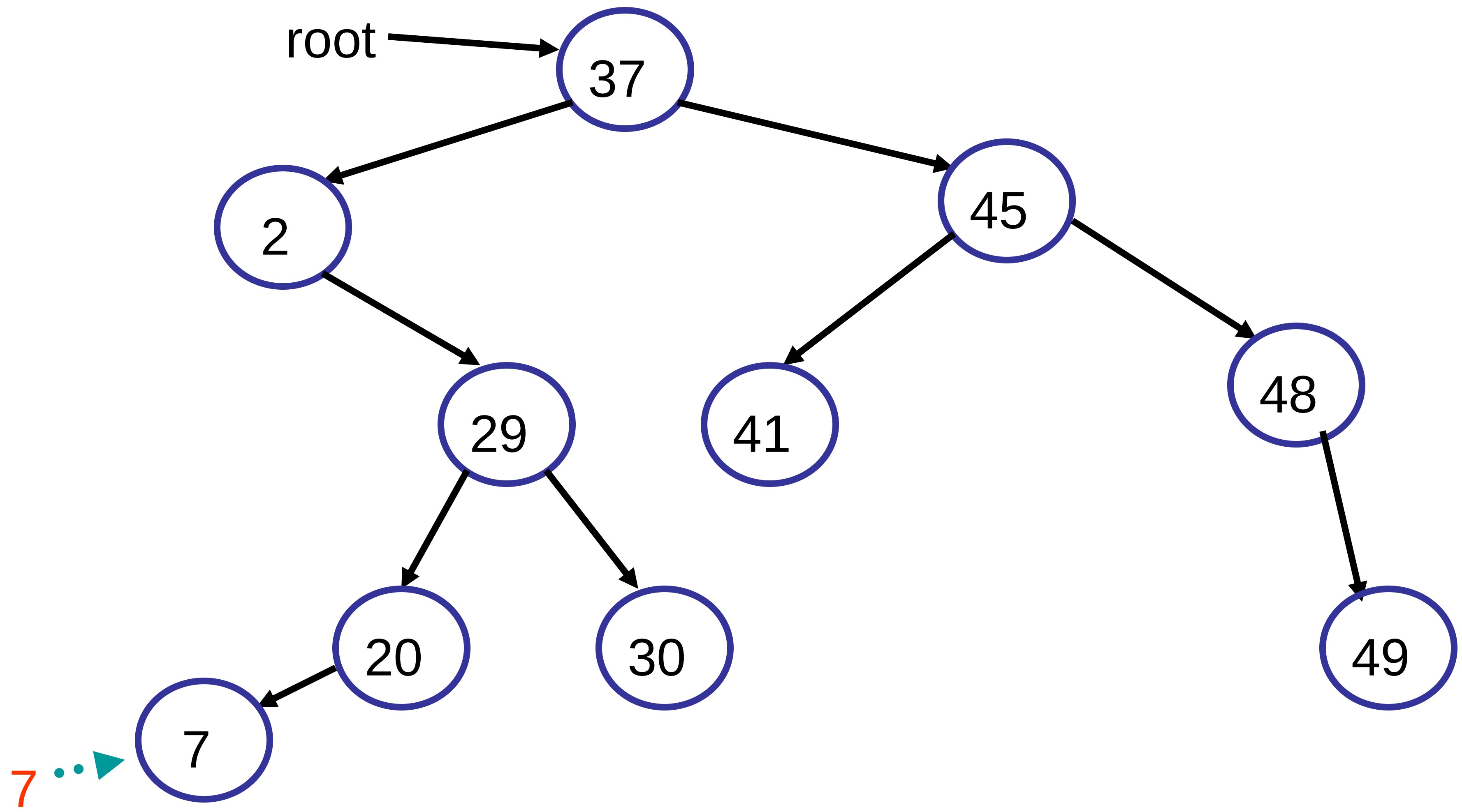
Inserting Nodes Into a BST (cont.)



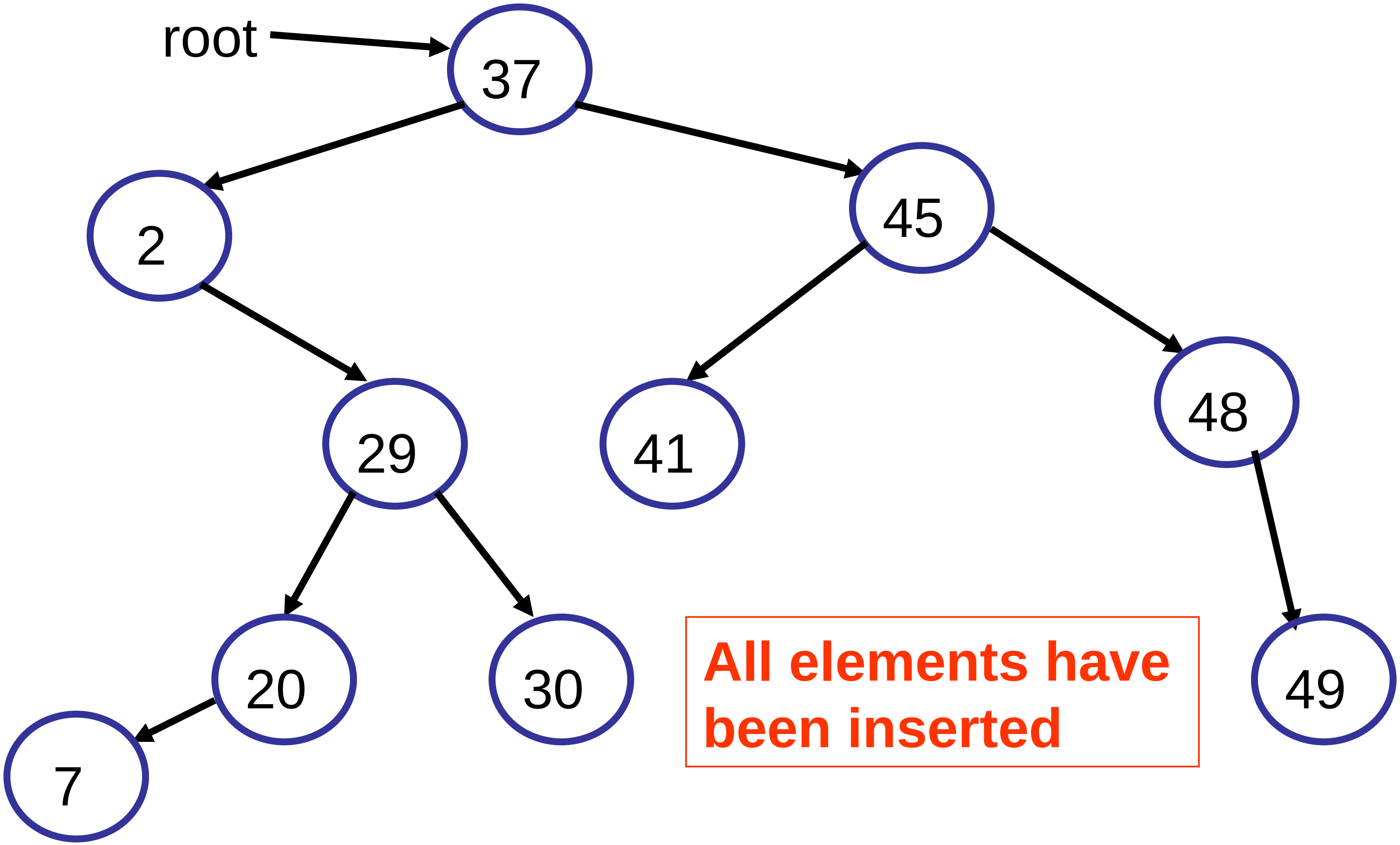
7 < 37
7 > 2
7 < 29
7 < 20

7

Inserting Nodes Into a BST (cont.)



Inserting Nodes Into a BST (cont.)



All elements have been inserted

Searching for a Key in a BST

Algorithm ***findElement*** **(*k*, *p*)**

Input: Find target ***k*** from vertex ***p*** and its children.

```
if p is null
    return NO_SUCH_KEY
if k == p
    return p
else if k < p
    return findElement (k, leftChild (p))
else // k > p
    return findElement (k, rightChild (p))
```

Searching for a Key in a BST

Searching means to find or locate a specific element or node in a data structure. In Binary search tree, searching a node is easy because elements in BST are stored in a specific order.

The steps of searching a node in Binary Search tree are listed as follows :

1. First, compare the element to be searched with the root element of the tree.

2. If root is matched with the target element, then return the node's location.

Searching for a Key in a BST

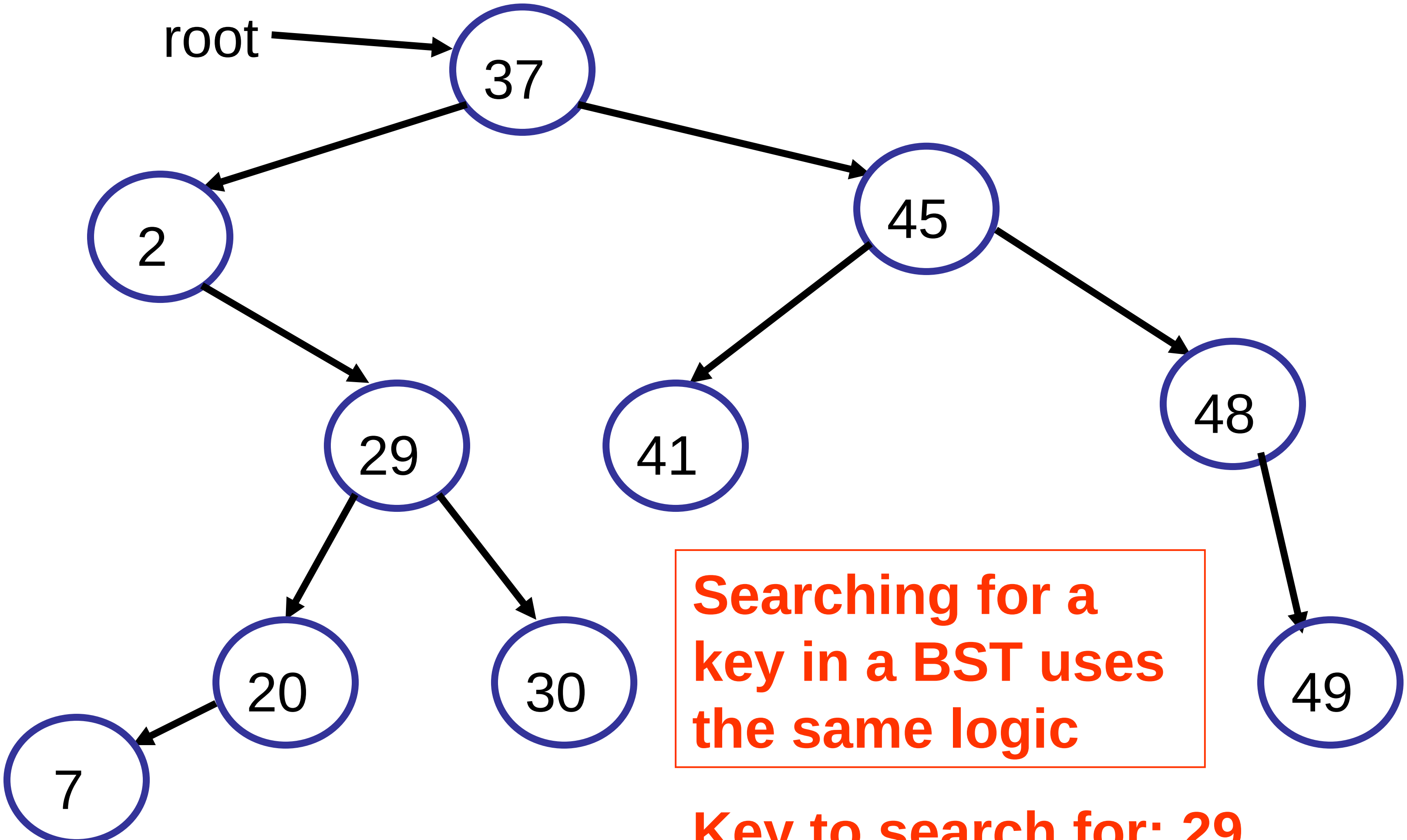
3. If it is not matched, then check whether the item is less than the root element, if it is smaller than the root element, then move to the left subtree.

4. If it is larger than the root element, then move to the right subtree.

5. Repeat the above procedure recursively until the match is found.

6. If the element is not found or not present in the tree, then return NULL.

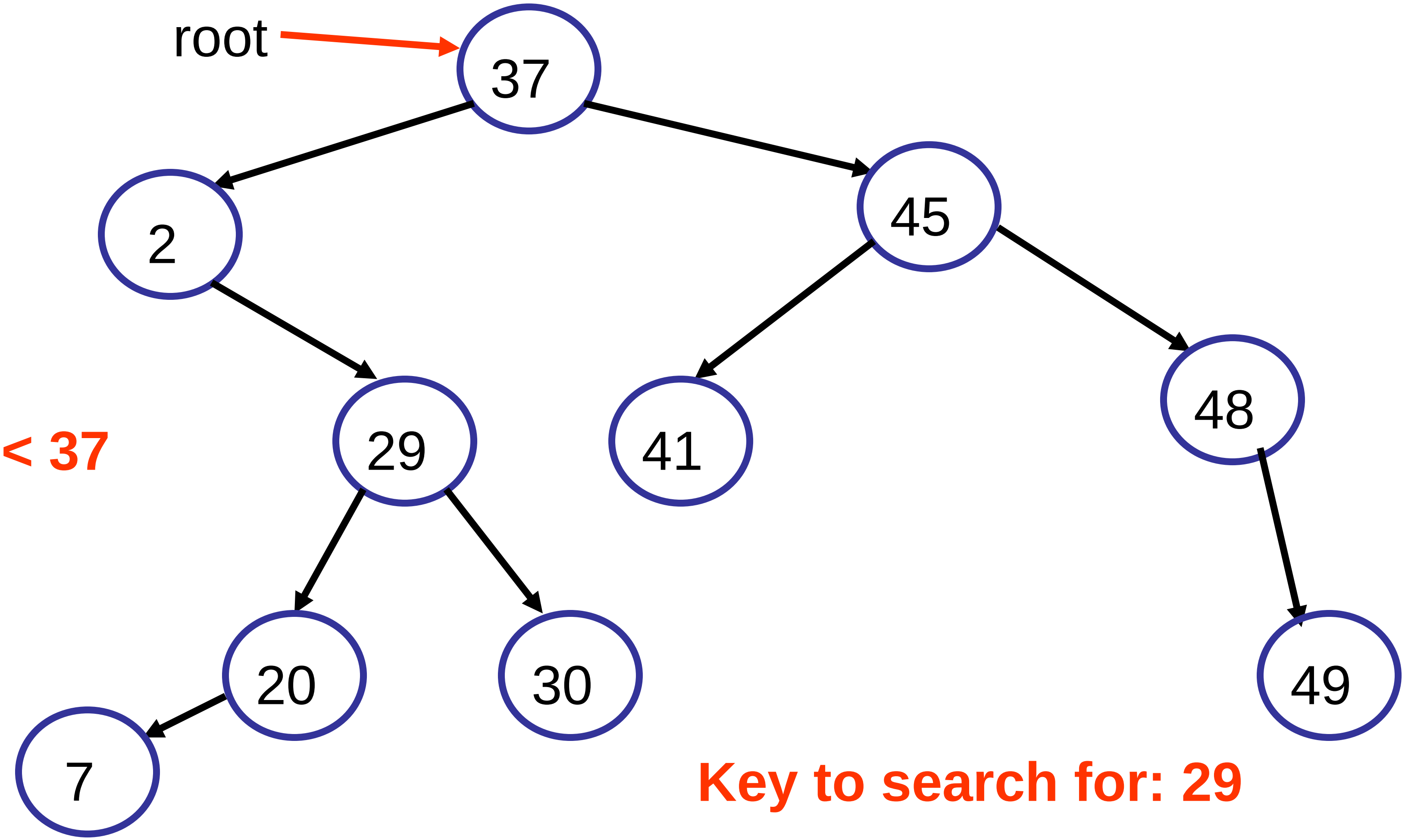
Searching for a Key in a BST



Searching for a key in a BST uses the same logic

Key to search for: 29

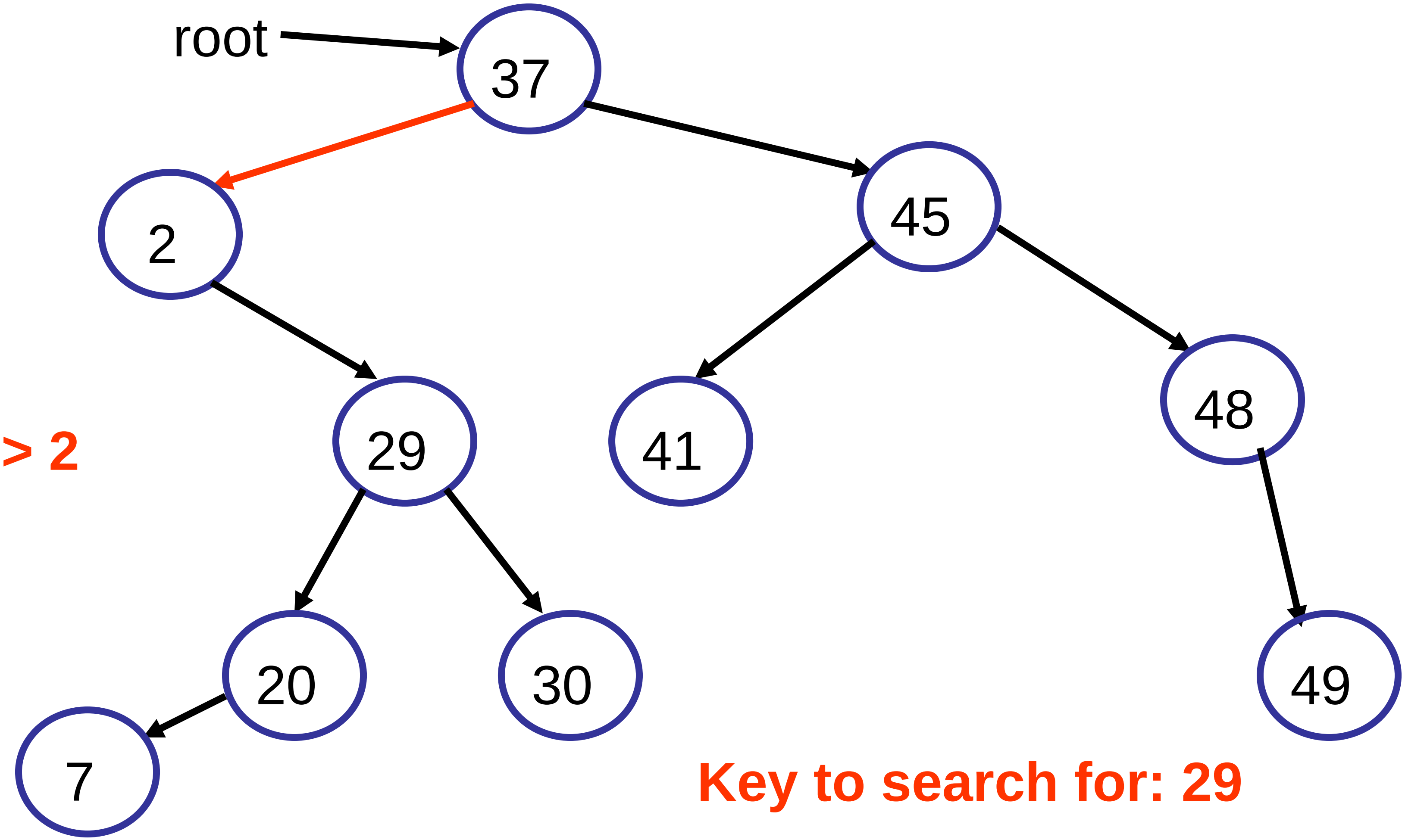
Searching for a Key in a BST (cont.)



29 < 37

Key to search for: 29

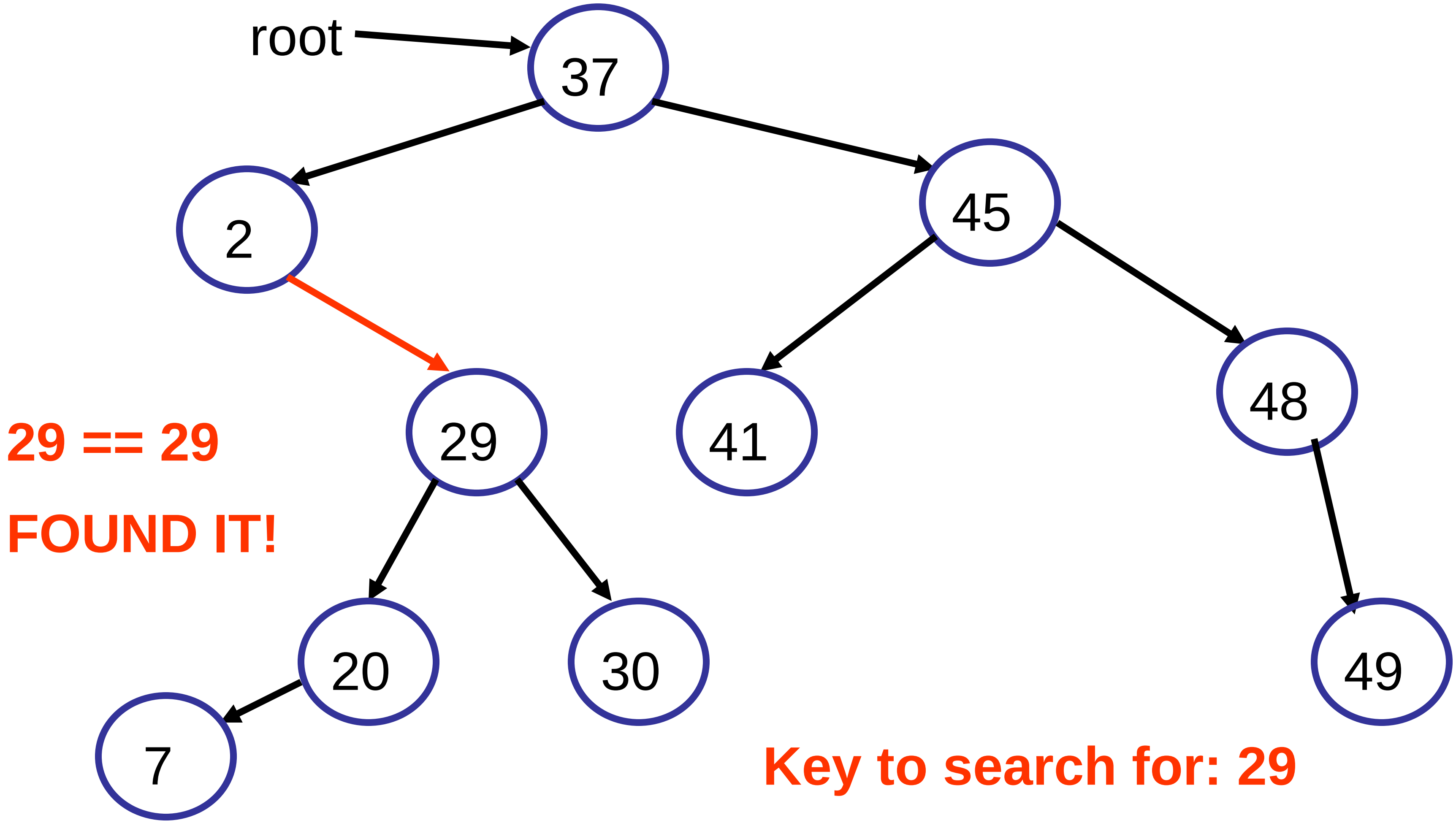
Searching for a Key in a BST (cont.)



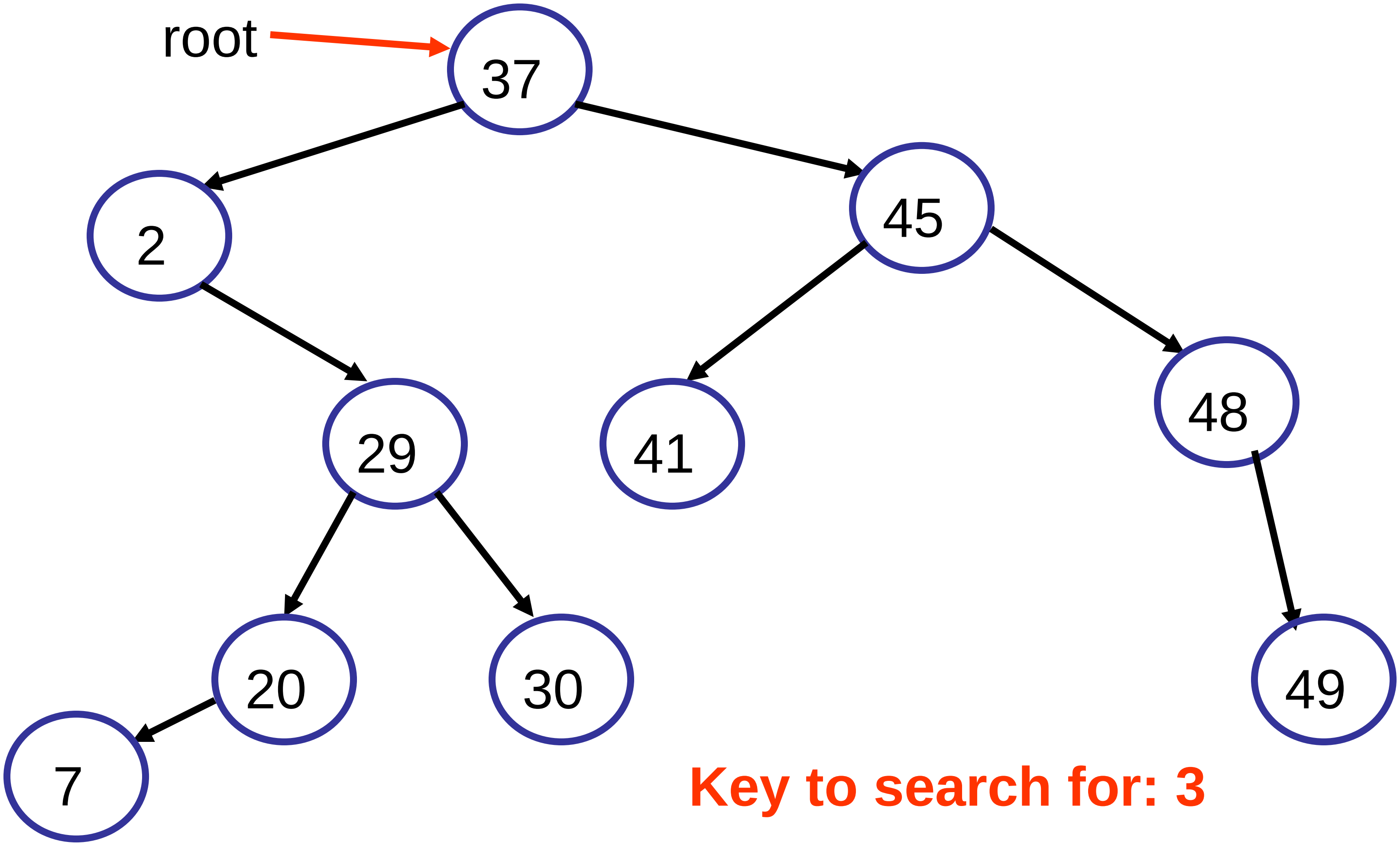
29 > 2

Key to search for: 29

Searching for a Key in a BST (cont.)



Searching for a Key in a BST (cont.)



Key to search for: 3

Searching for a Key in a BST (cont.)

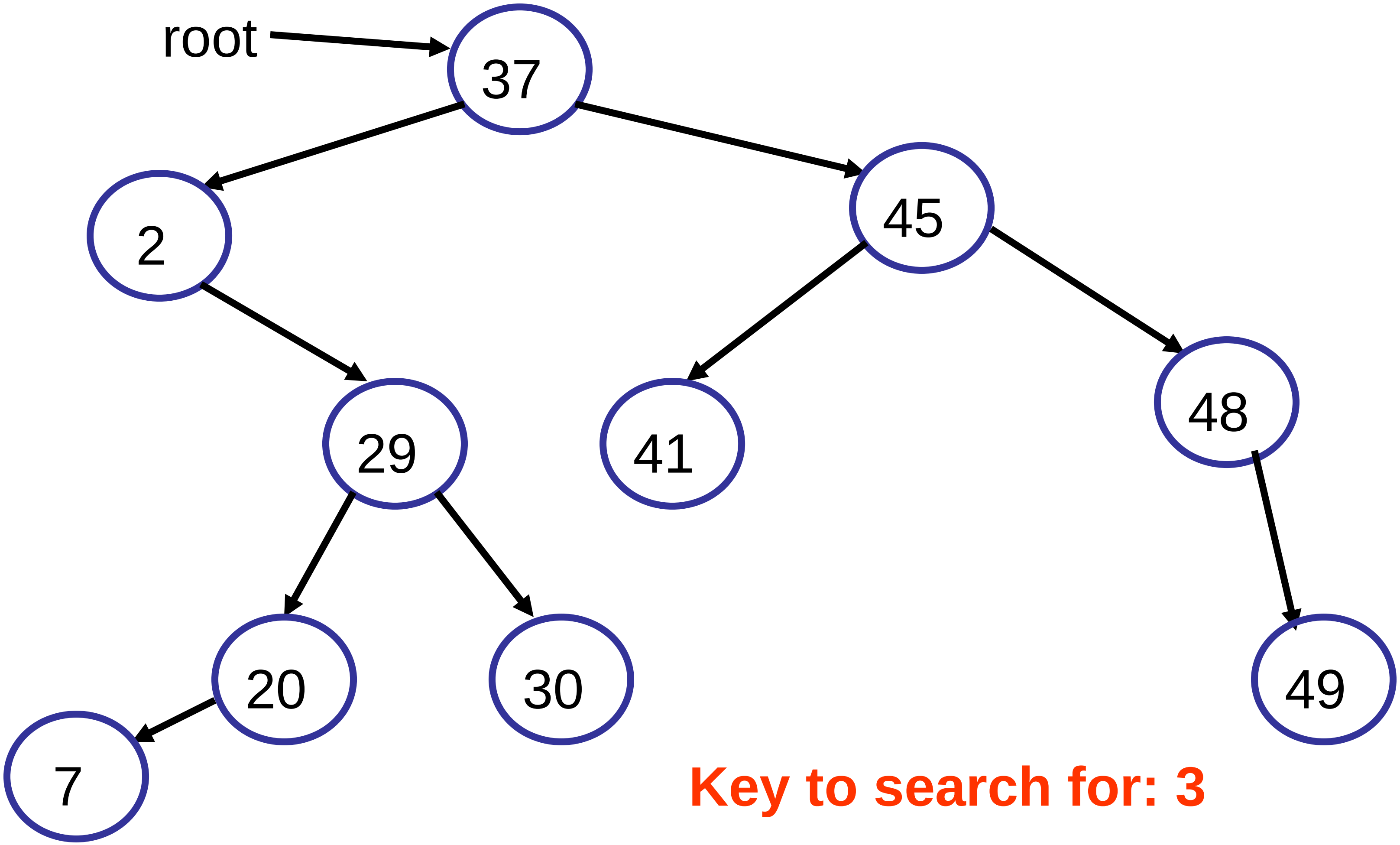
3 < 37

3 > 2

3 < 29

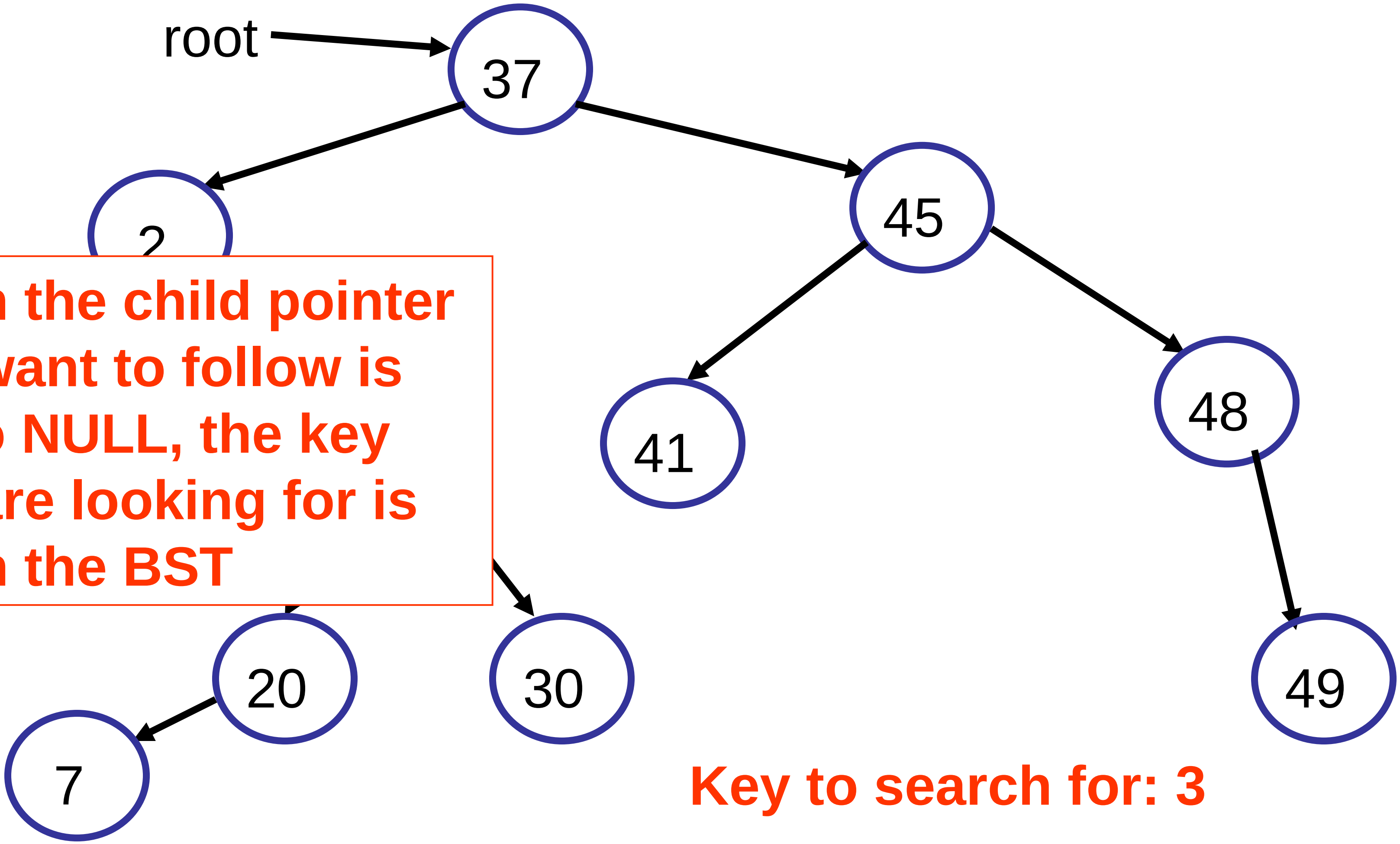
3 < 20

3 < 7



Key to search for: 3

Searching for a Key in a BST (cont.)



When the child pointer you want to follow is set to NULL, the key you are looking for is not in the BST

Key to search for: 3

Deletion in Binary Search tree

- In a binary search tree, we must delete a node from the tree by keeping in mind that the property of BST is not violated.

- To delete a node from BST, there are three possible situations occur –

    - The node to be deleted is the leaf node, or,

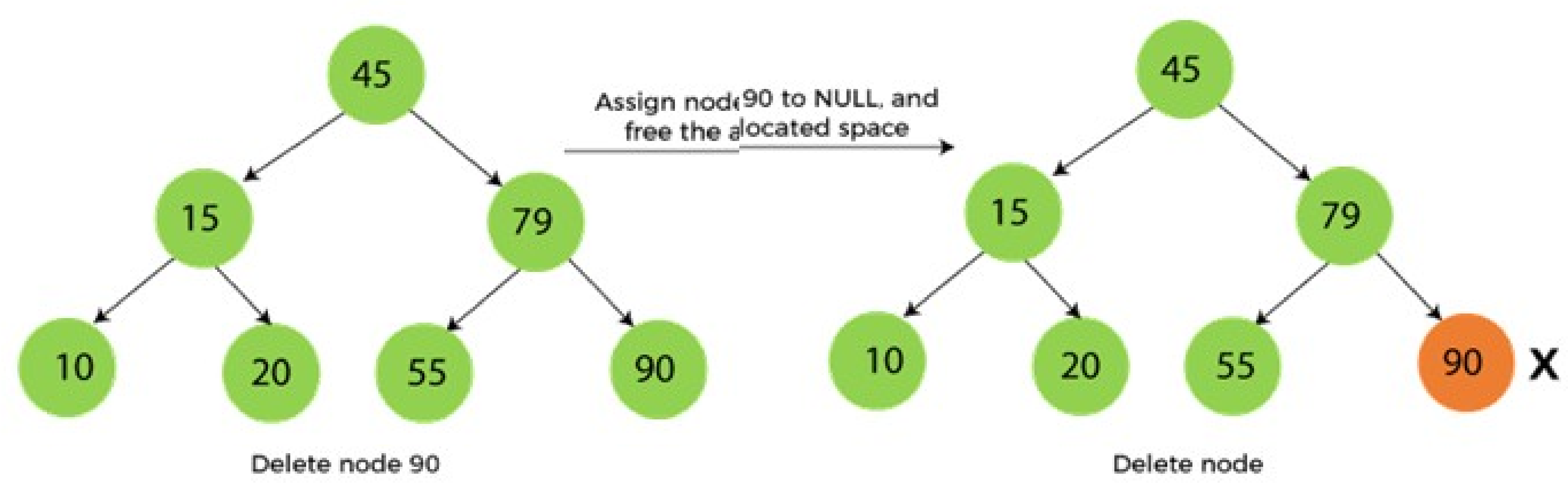
    - The node to be deleted has only one child, and,

    - The node to be deleted has two children

Deletion in Binary Search tree

- **When the node to be deleted is the leaf node** It is the simplest case to delete a node in BST.

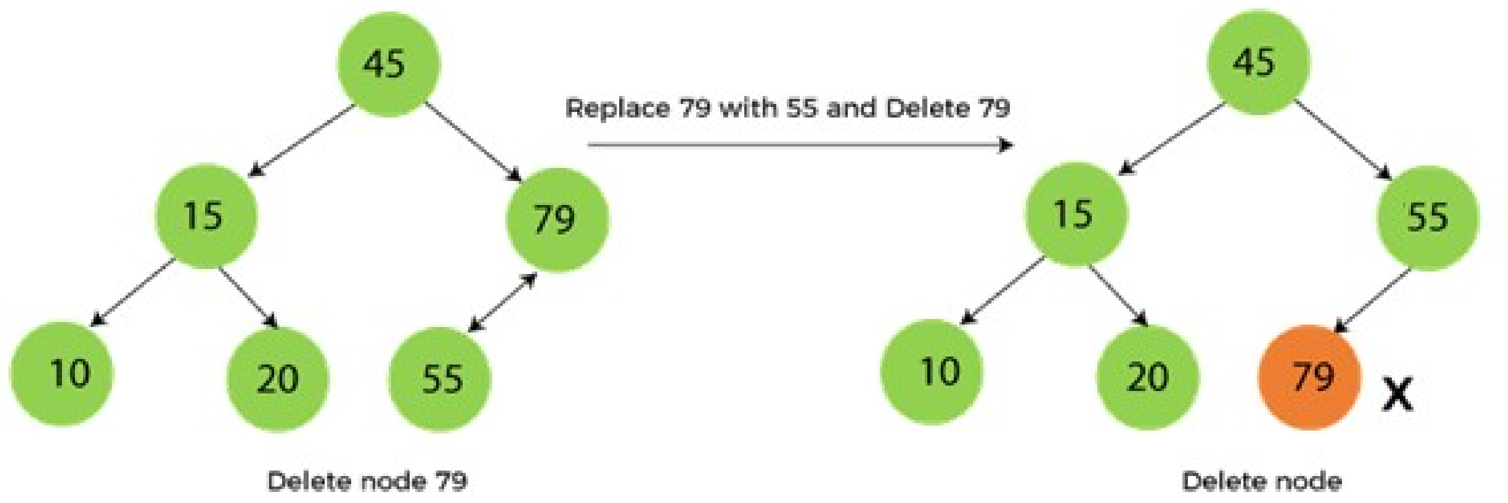
- Here, we have to replace the leaf node with NULL and simply free the allocated space.



Deletion in Binary Search tree

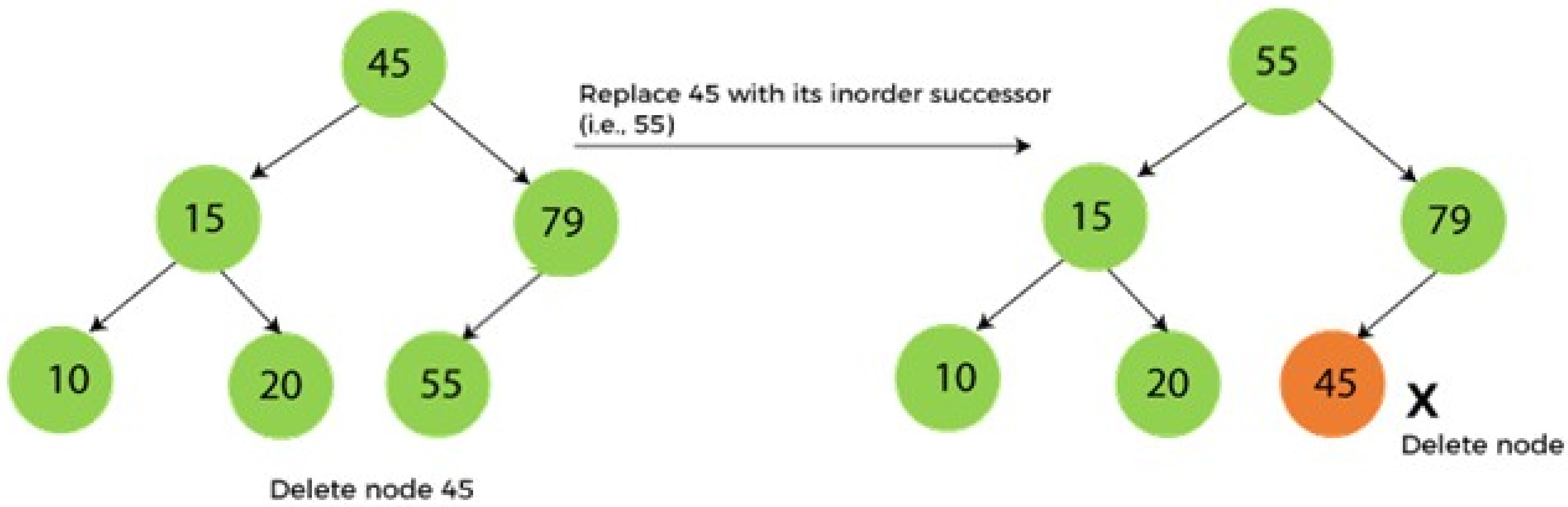
- **When the node to be deleted is the leaf node** It is the simplest case to delete a node in BST. Here, we have to replace the leaf node with NULL and simply free the allocated space.

- **The node to be deleted has only one child– delete 79**



Deletion in Binary Search tree

- The node to be deleted has two children---- **delete 79**



Time Complexities

Operations	Best case time complexity	Average case time complexity	Worst case time complexity
Insertion	O(log n)	O(log n)	O(n)
Deletion	O(log n)	O(log n)	O(n)
Search	O(log n)	O(log n)	O(n)

Time Complexities

Operations	Space complexity
Insertion	O(n)
Deletion	O(n)
Search	O(n)