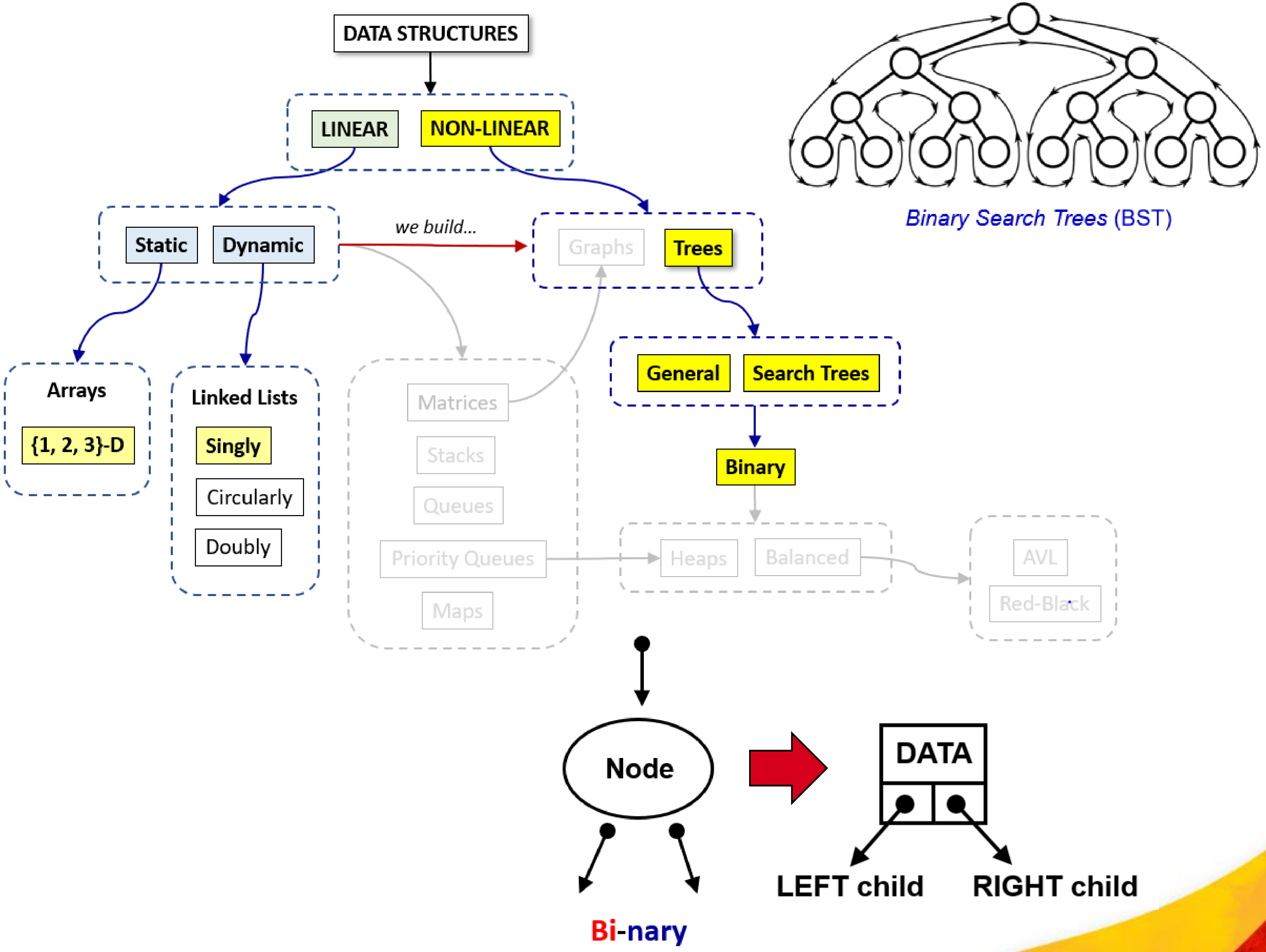
Binary Search Trees

UNIVERSITY OF CALGARY

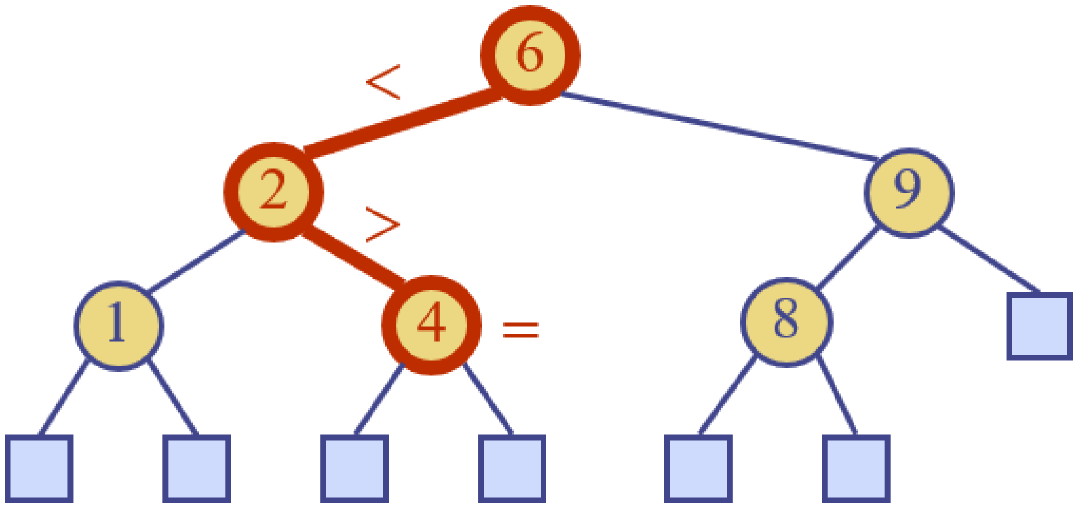


UNIVERSITY OF CALGARY

Binary Search Tree (BST)

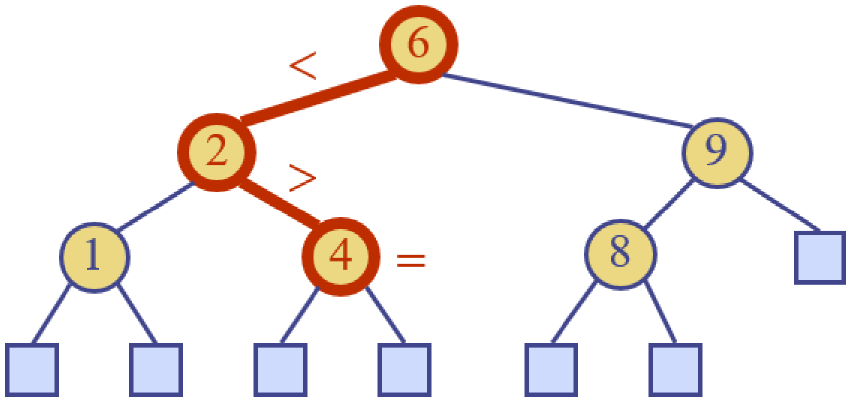
- A common form of the binary tree found in computer applications is the binary search tree.
- An ordered tree with a **search structure** where at any position in the tree all nodes less than the current node lie to the left and all nodes greater than the current node lie to the right

(LEFT child → DATA) < **parent** < **(RIGHT** child → DATA **)**



BST Insertions: Scenario

- **We have a Binary Search Tree**
  - It can be empty
  - Or have some elements in it already
- **We want to add an element to it**
  - Inserting/adding involves 2 steps:
    - Find the correct location
    - Do the steps to add a new node
- **Must maintain "search" structure:**
  (LEFT child→ DATA) < **parent** < **(RIGHT** child → DATA)

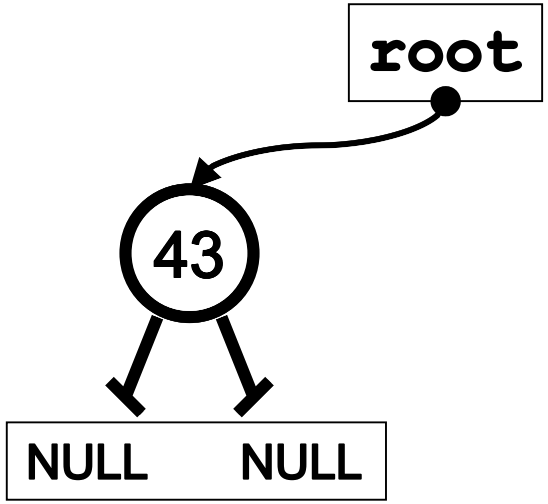


Example of BST Insertion

- What follows is a step by step visualization of BST insertion
- An array of integers is the input
- The tree begins empty

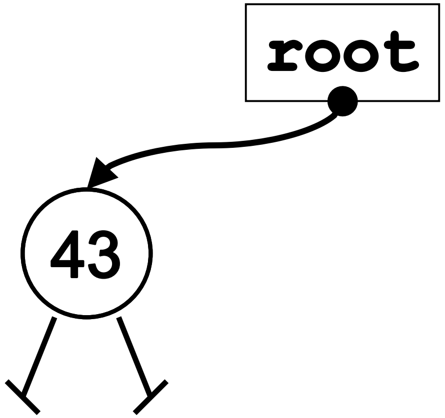
Example (1): BST from input array of integers

43 59 40 31 64 33 20 56 47 28 89



- Allocate Node: root
- Insert Data: root→DATA = 43
- Update Left and Right references (i.e., pointers) to NULL:
  - ✓ root→LEFT = NULL; root→RIGHT = NULL

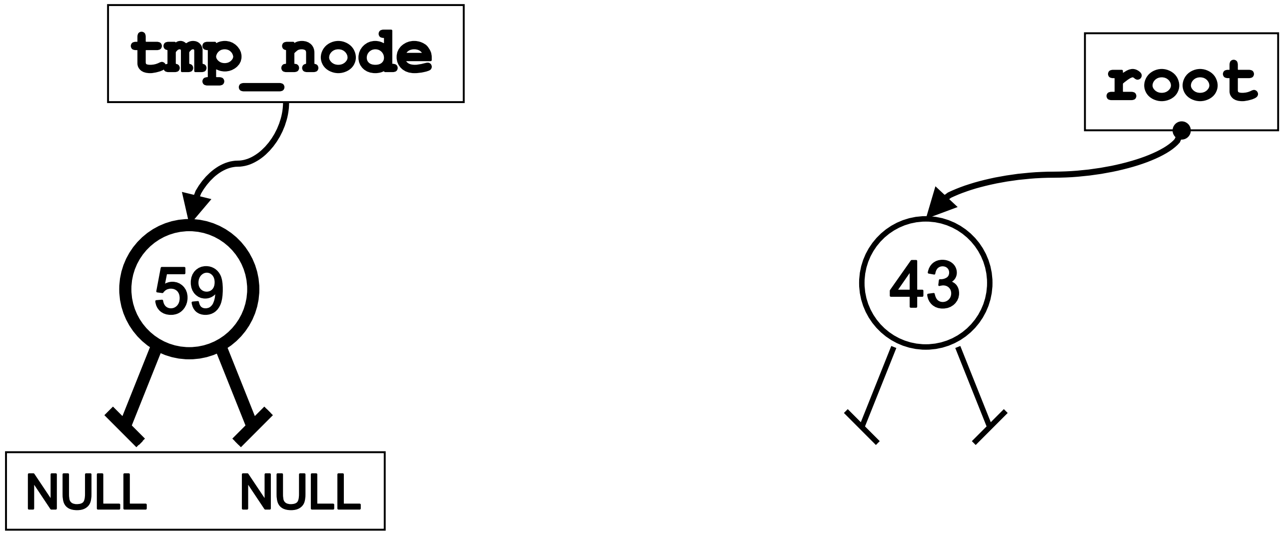
43	59	40	31	64	33	20	56	47	28	89



root node

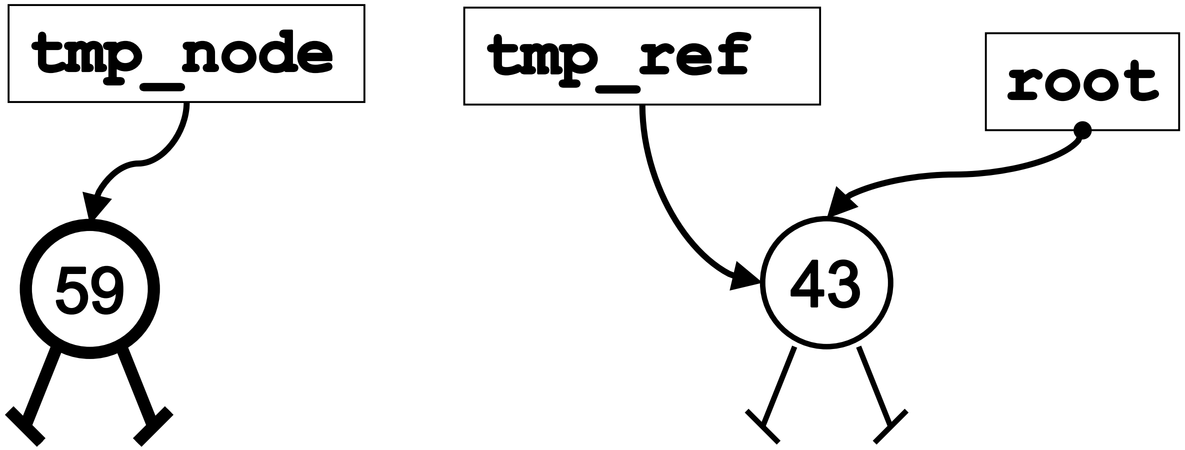
leaf node

43	59	40	31	64	33	20	56	47	28	89

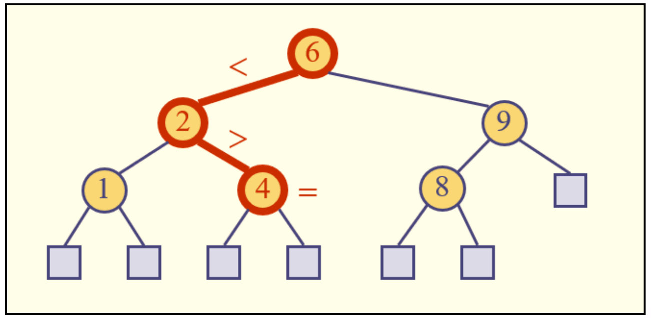


- Allocate New Node: tmp_node
- Insert Data in New Node: tmp_node→DATA = 59
- Update Left and Right references (i.e., pointers) to NULL:
  - ✓ tmp_node→LEFT = NULL; tmp_node→RIGHT = NULL

43	59	40	31	64	33	20	56	47	28	89



BST “search” structure

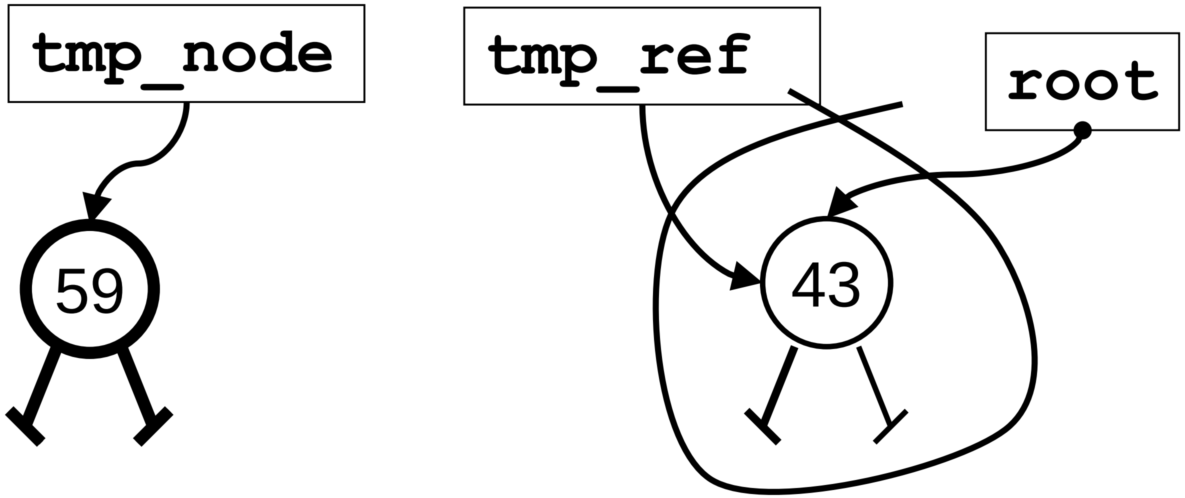


- (...)
- tmp_ref = root

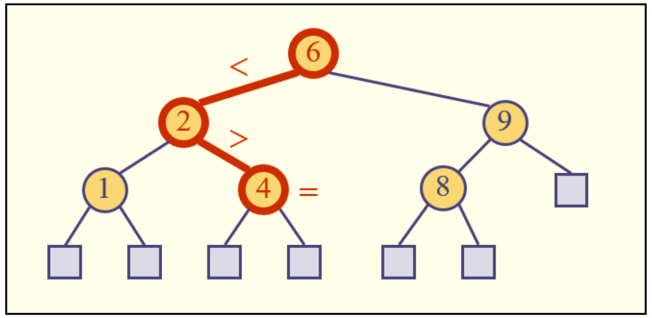
  i.e., tmp_ref will reference (i.e., point at)

  the location in the BST to insert the new node

43	59	40	31	64	33	20	56	47	28	89

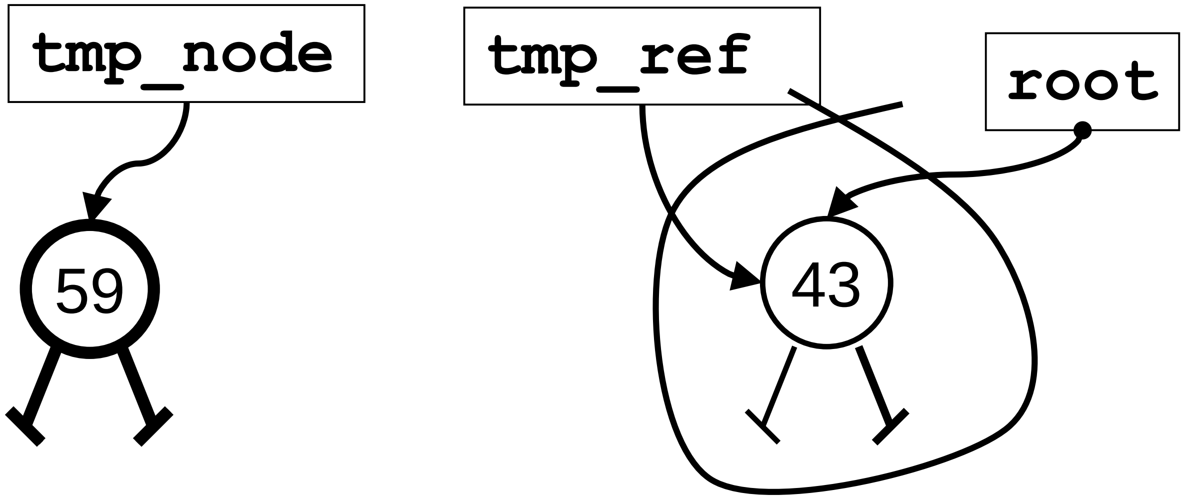


BST “search” structure

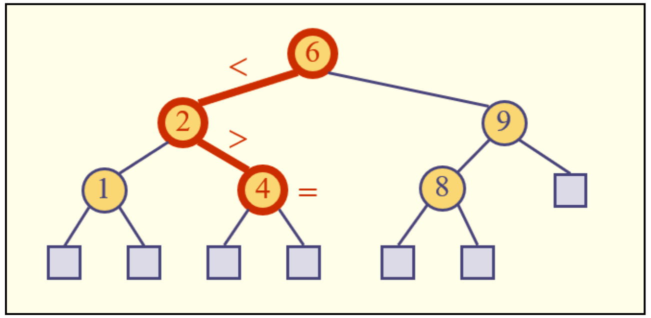


- (…)
- tmp_node→DATA < tmp_ref→DATA? Insert @ tmp_ref→ LEFT

43	59	40	31	64	33	20	56	47	28	89

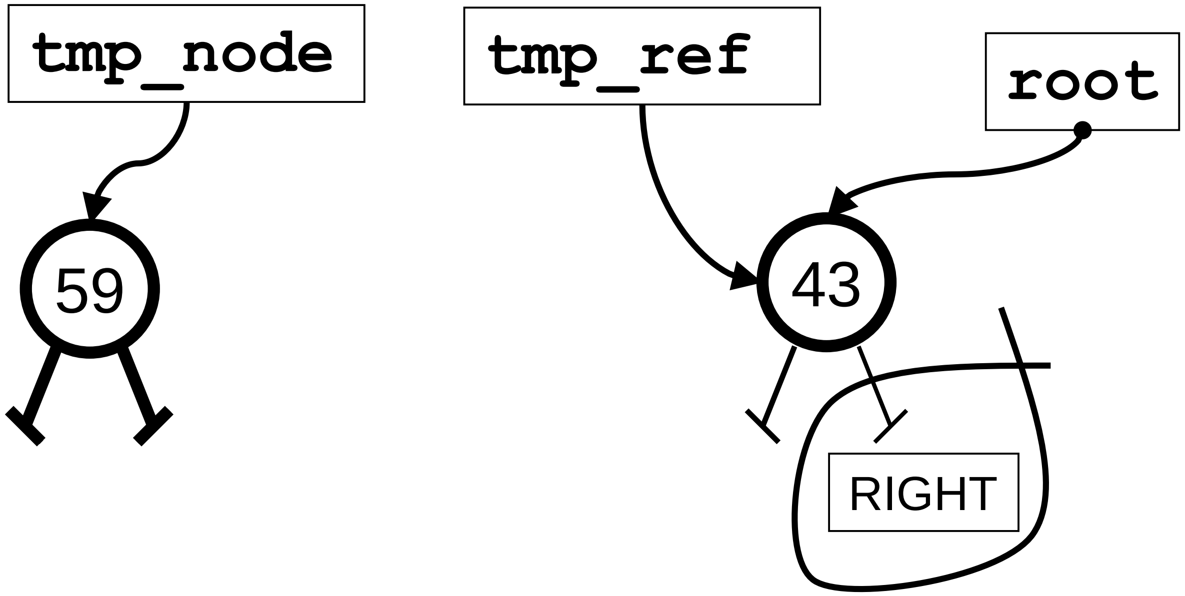


BST "search" structure



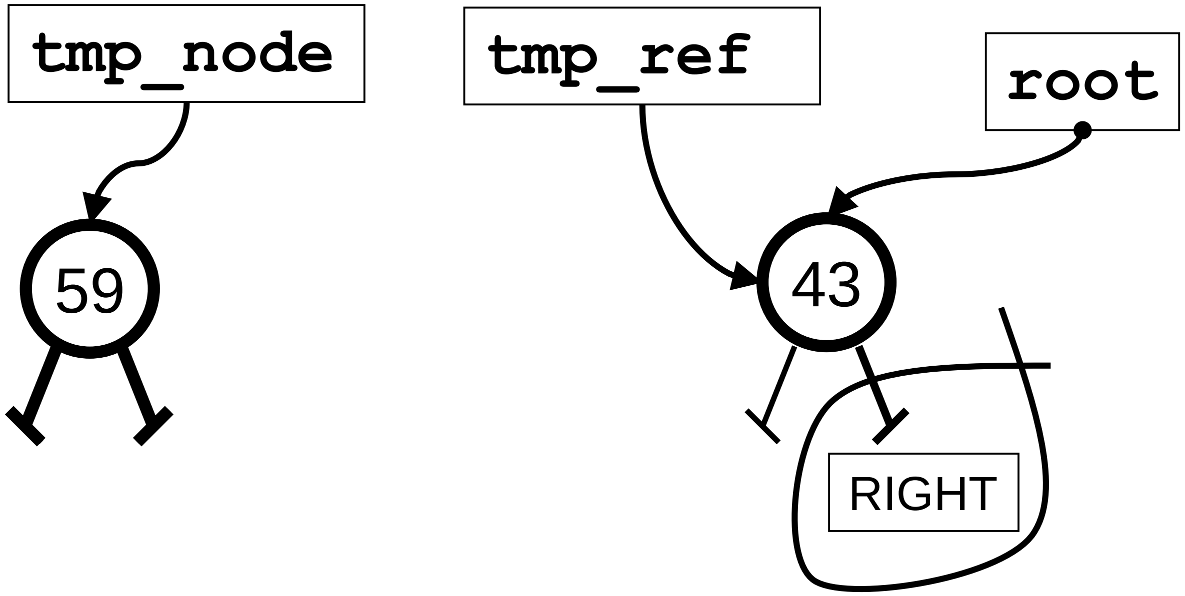
- (…)
- tmp_node→DATA < tmp_ref→DATA? Insert @ tmp_ref→ LEFT
- tmp_node→DATA > tmp_ref→DATA? Insert @ tmp_ref→ RIGHT

43	59	40	31	64	33	20	56	47	28	89



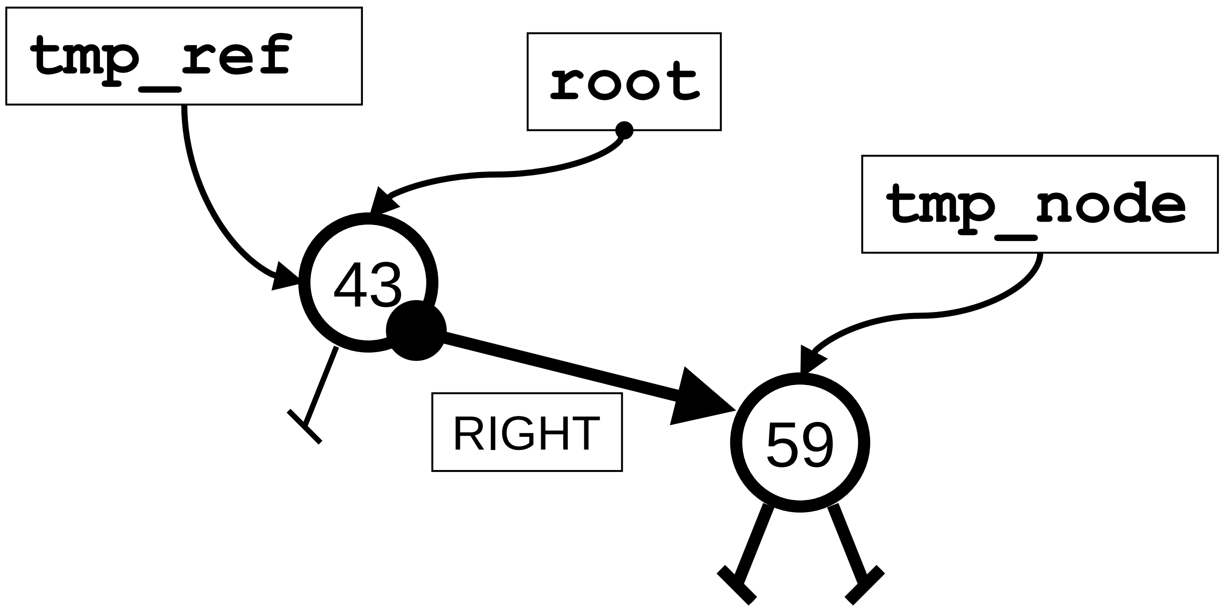
- (...)
- tmp_node→ 59 < tmp_ref→ 43? NO
- tmp_node→ 59 > tmp_ref→ 43? YES: Insert @ tmp_ref→ RIGHT

43	59	40	31	64	33	20	56	47	28	89



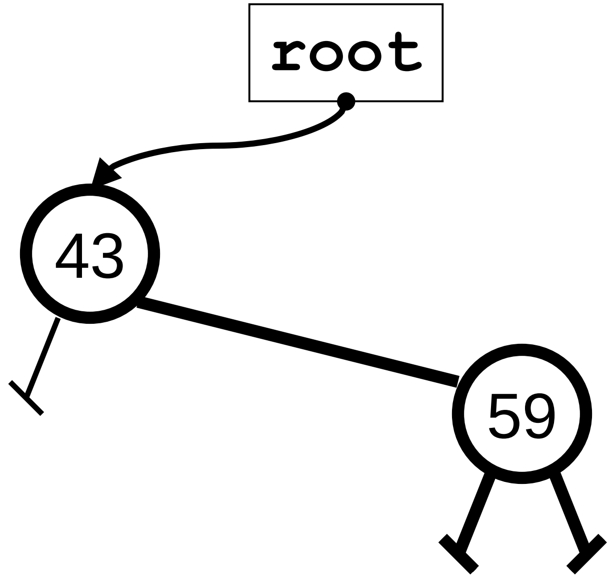
- (…)
- tmp_node→ 59 > tmp_ref→ 43? YES: Insert @ tmp_ref→ RIGHT
  - ✓ tmp_ref→ RIGHT = tmp_node

43	59	40	31	64	33	20	56	47	28	89



- (…)
- tmp_node→ 59 > tmp_ref→ 43? YES: Insert @ tmp_ref→ RIGHT
  - ✓ tmp_ref→ RIGHT = tmp_node

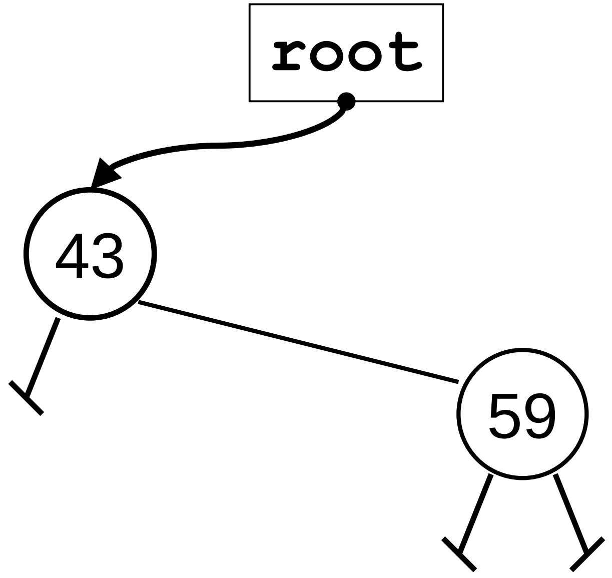
43	59	40	31	64	33	20	56	47	28	89



43	59	40	31	64	33	20	56	47	28	89

root node

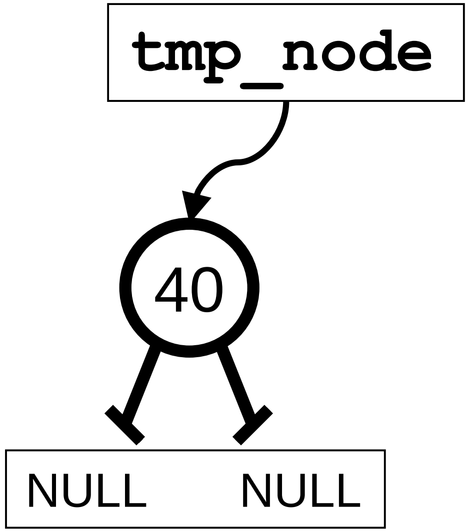
parent node

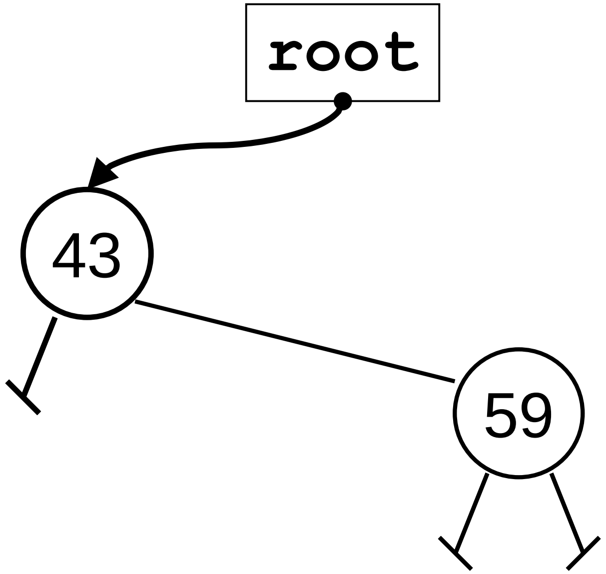


right child node

leaf node

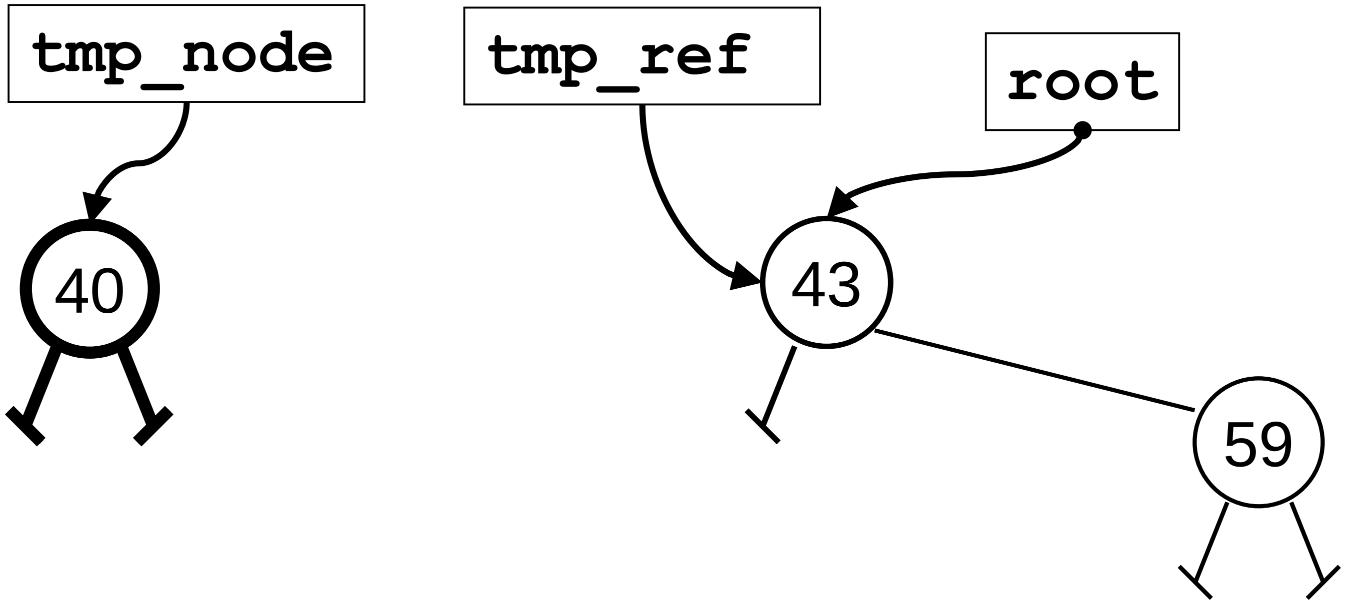
43	59	40	31	64	33	20	56	47	28	89





- Allocate New Node: tmp_node
- Insert Data in New Node: tmp_node→DATA = 40
- Update Left and Right references (i.e., pointers) to NULL:
  - ✓ tmp_node→LEFT = NULL; tmp_node→RIGHT = NULL

43	59	40	31	64	33	20	56	47	28	89

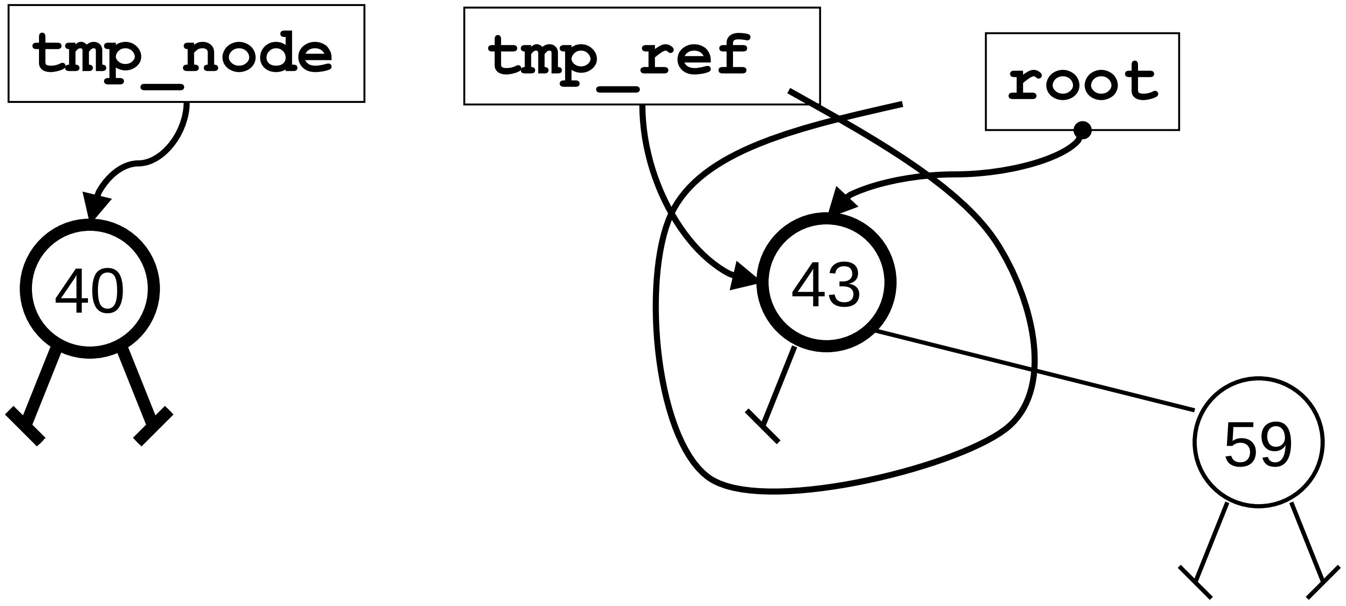


- (…)
- tmp_ref = root

  i.e., tmp_ref will reference (i.e., point at)

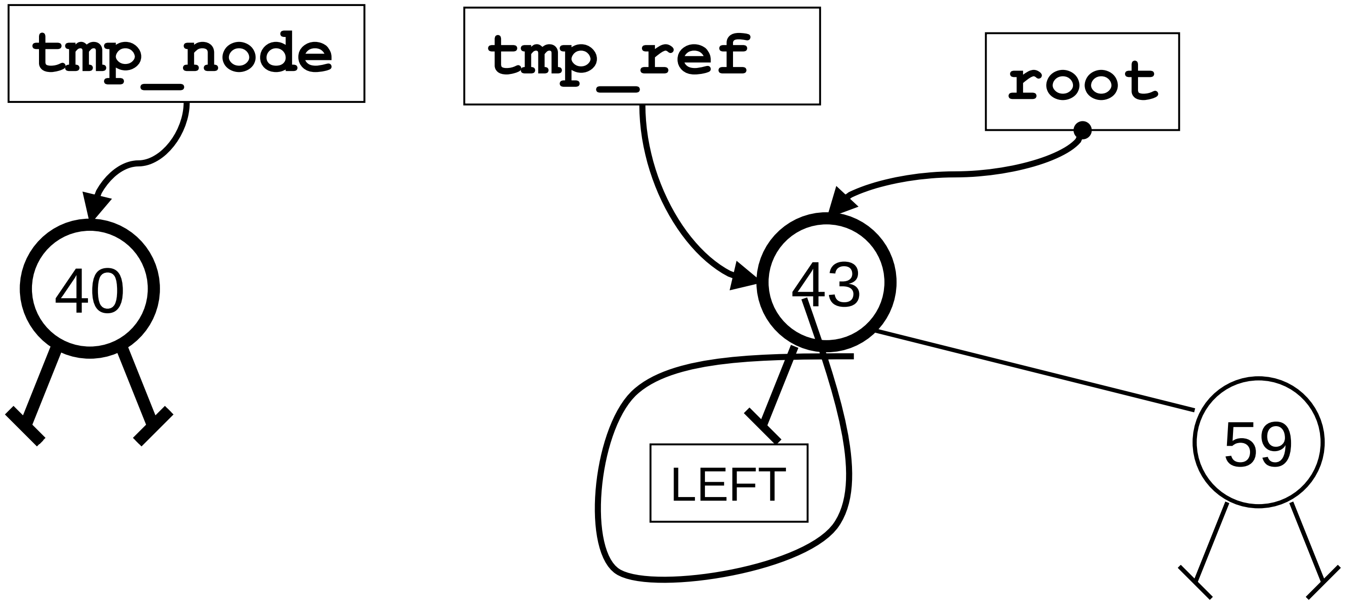
  the location in the BST to insert the new node

43	59	40	31	64	33	20	56	47	28	89



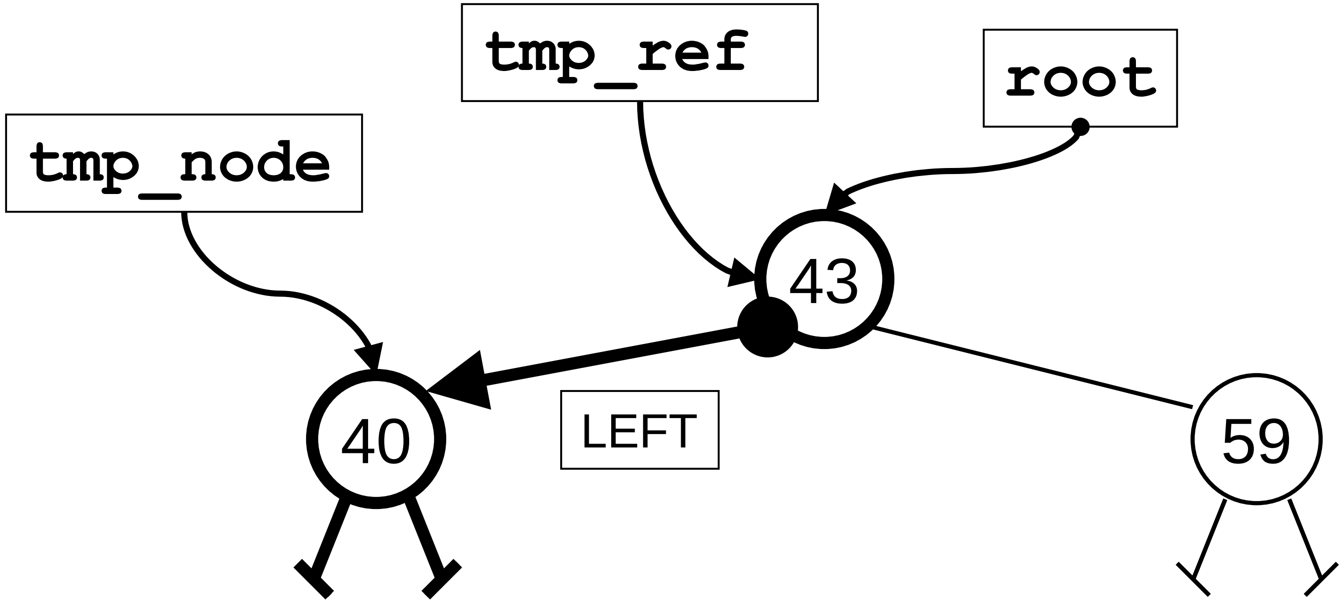
- (…)
- tmp_node→ DATA < tmp_ref→ DATA? Insert @ tmp_ref→ LEFT
- tmp_node→ DATA > tmp_ref → DATA? Insert @ tmp_ref→ RIGHT

43	59	40	31	64	33	20	56	47	28	89



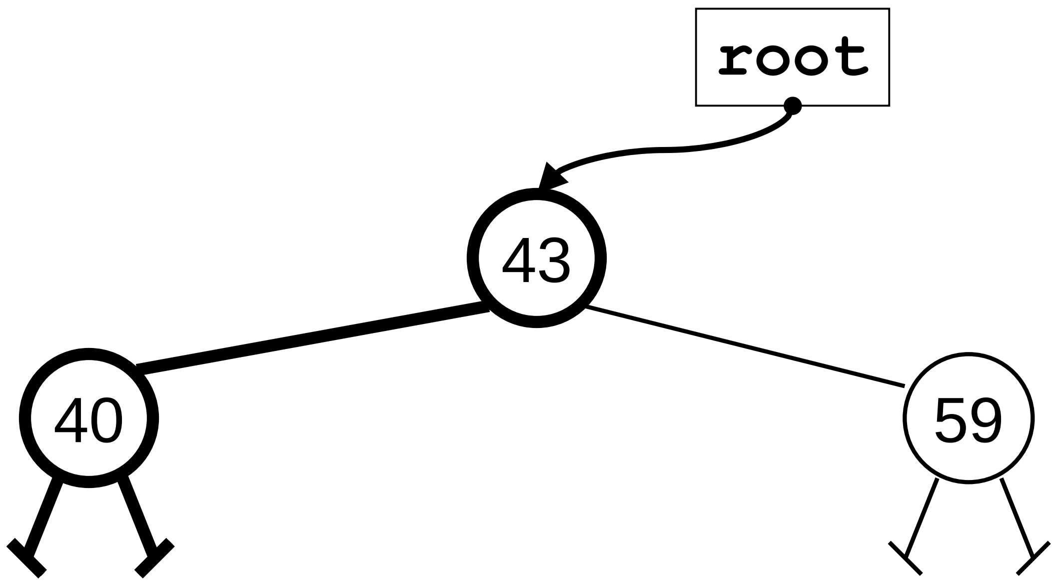
- (…)
- tmp_node→ 40 < tmp_ref→ 43? YES: Insert @ tmp_ref→ LEFT

43	59	40	31	64	33	20	56	47	28	89

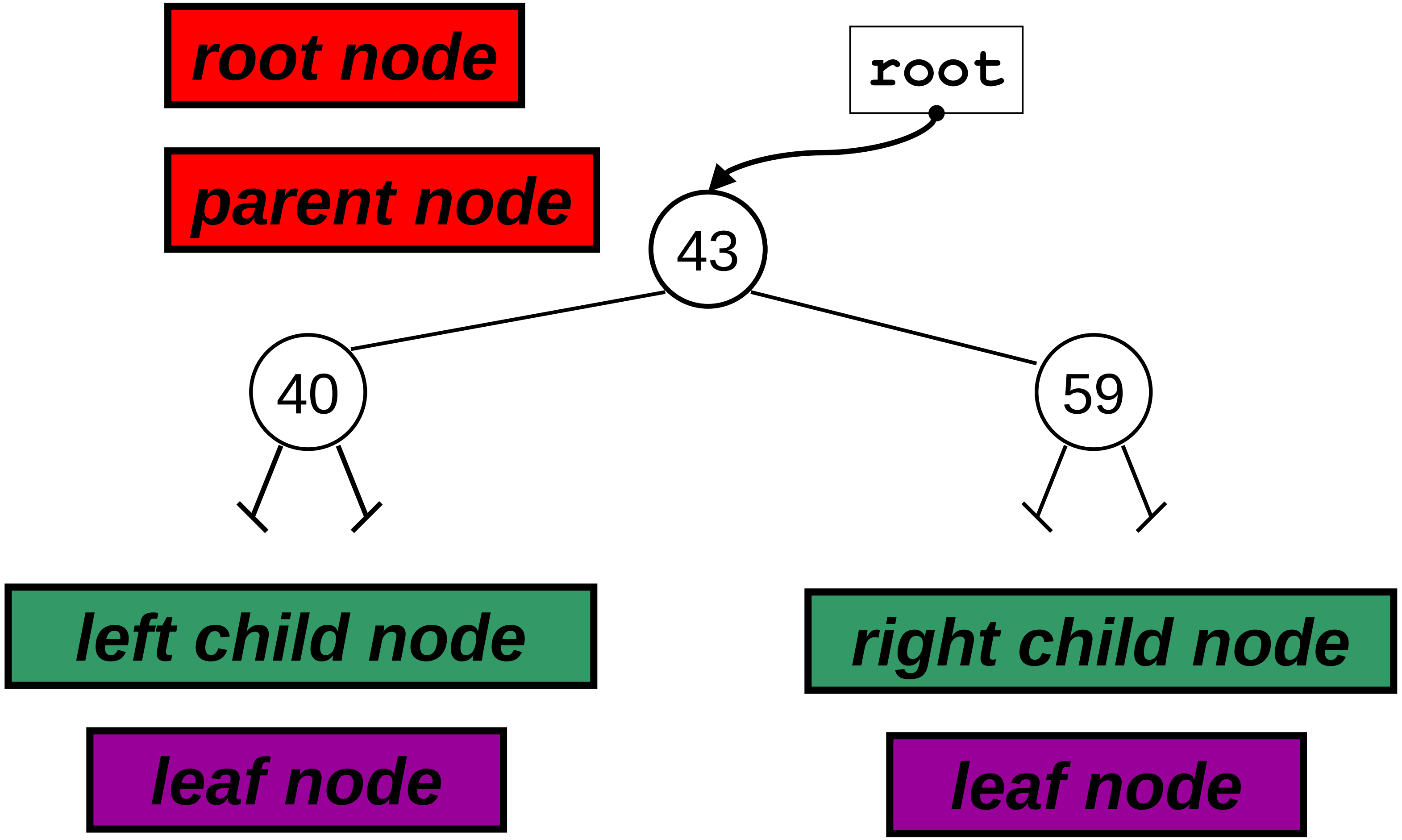


- (…)
- tmp_node→ 40 < tmp_ref→ 43? YES: Insert @ tmp_ref→ LEFT
  - ✓ tmp_ref→ LEFT = tmp_node

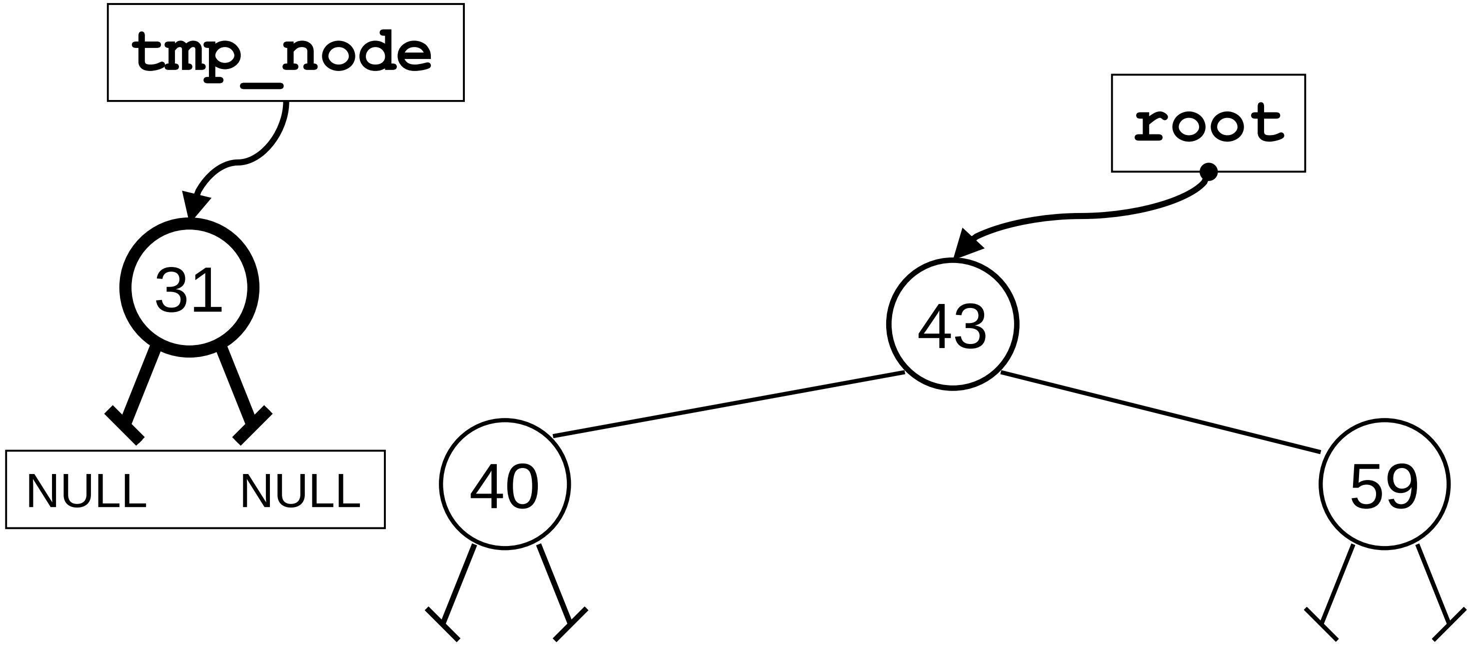
43	59	40	31	64	33	20	56	47	28	89



43	59	40	31	64	33	20	56	47	28	89

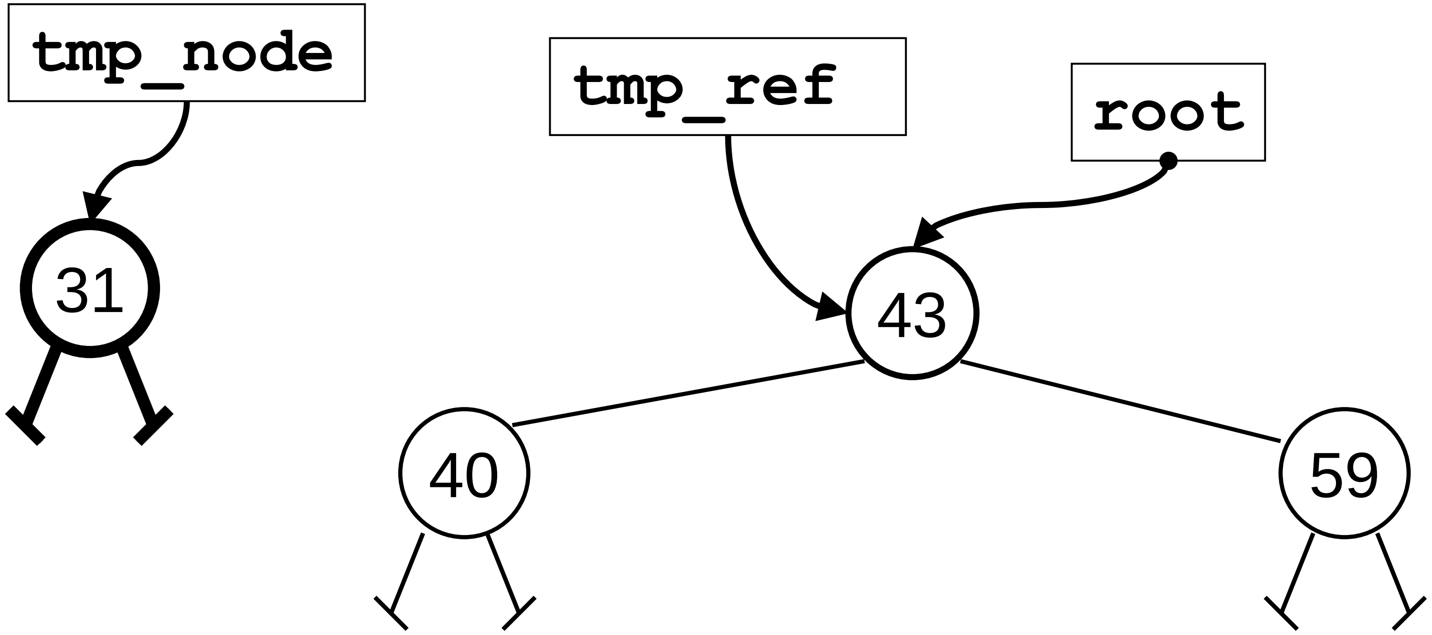


43	59	40	31	64	33	20	56	47	28	89



- Allocate New Node: tmp_node
- Insert Data in New Node: tmp_node→DATA = 31
- Update Left and Right references (i.e., pointers) to NULL:
  - ✓ tmp_node→LEFT = NULL; tmp_node→RIGHT = NULL

43	59	40	31	64	33	20	56	47	28	89

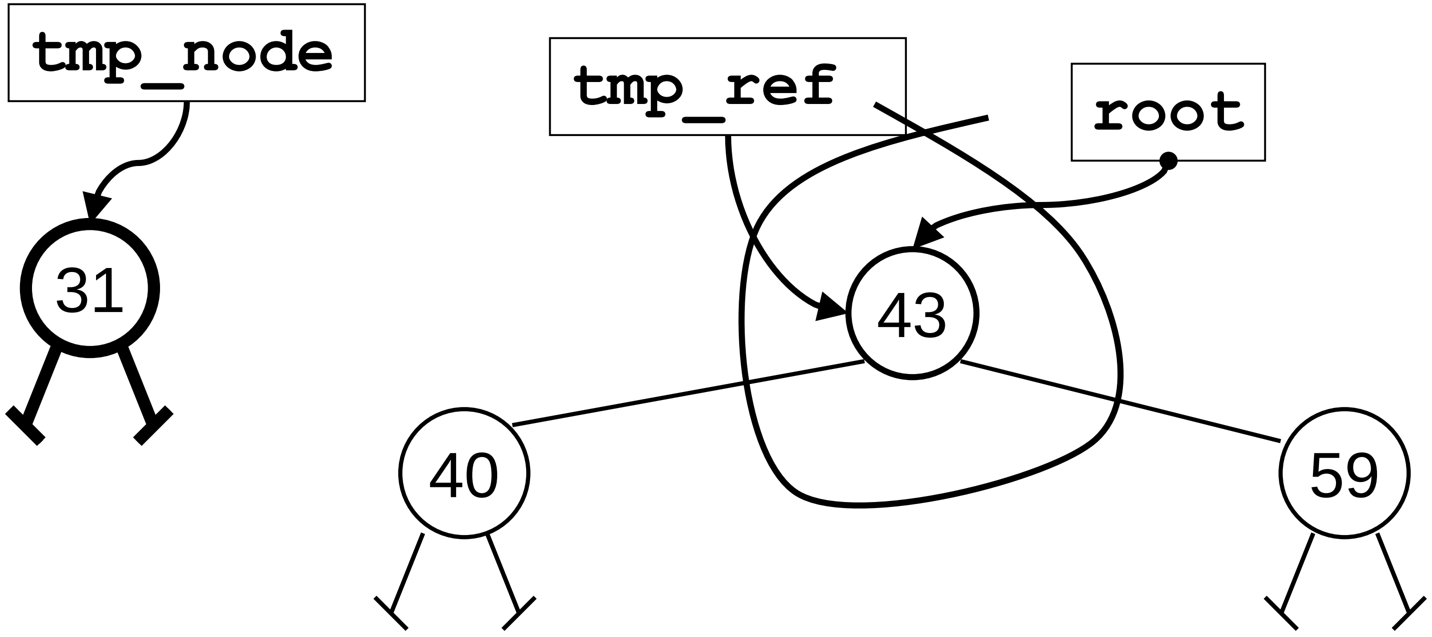


- (…)
- tmp_ref = root

  i.e., tmp_ref will reference (i.e., point at)

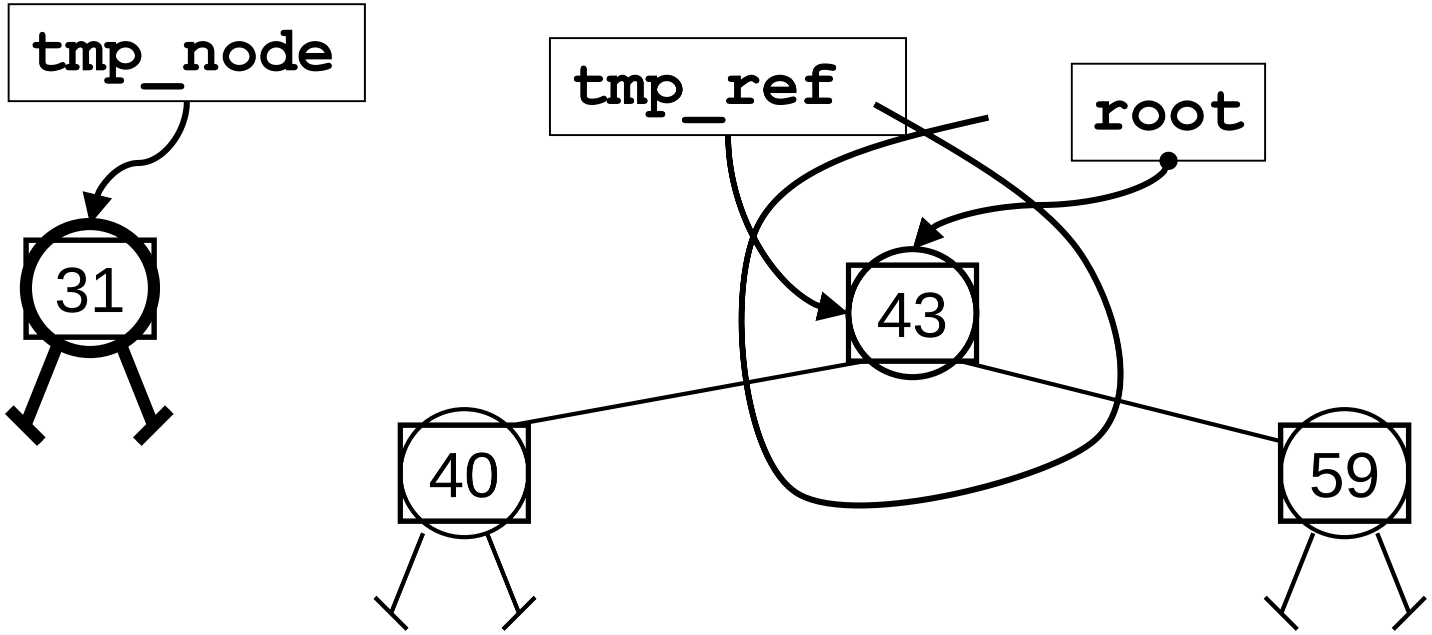
  the location in the BST to insert the new node

43	59	40	31	64	33	20	56	47	28	89



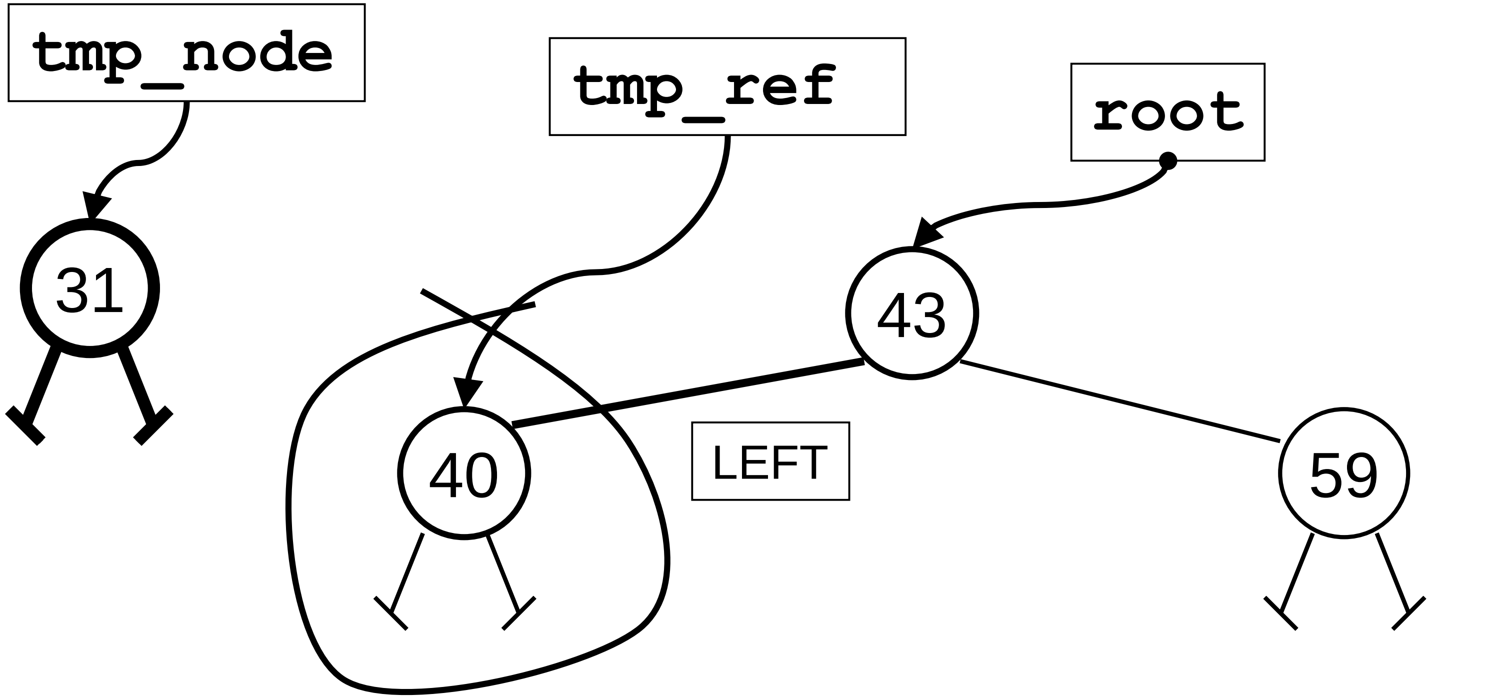
- (…)
- tmp_node→ DATA < tmp_ref→ DATA? Insert @ tmp_ref→ LEFT
- tmp_node→ DATA > tmp_ref → DATA? Insert @ tmp_ref→ RIGHT

43	59	40	31	64	33	20	56	47	28	89



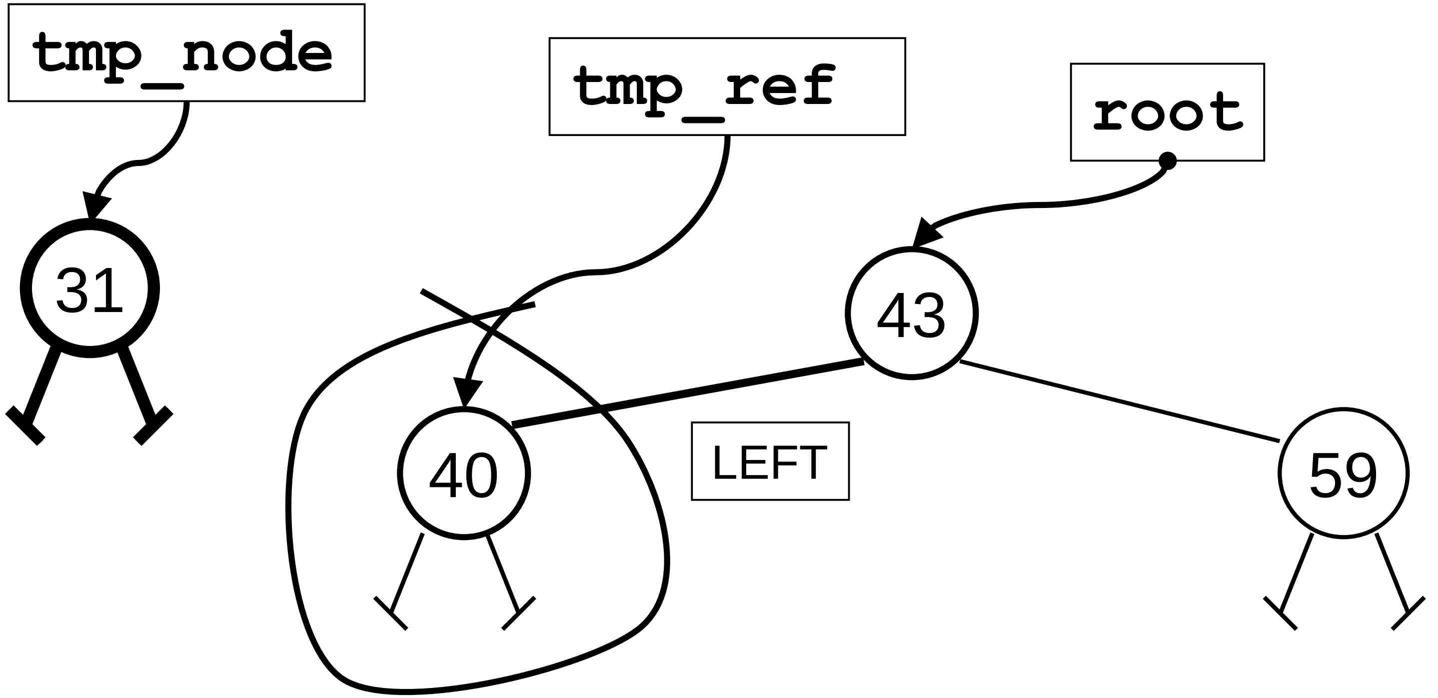
- (...)
- tmp_node→ 31 < tmp_ref→ 43? YES: Insert @ tmp_ref→ LEFT
  - ✓ tmp_ref→LEFT = NULL? // i.e., Leaf Node?
  - ✓ NO: tmp_ref = tmp_ref→LEFT

43	59	40	31	64	33	20	56	47	28	89



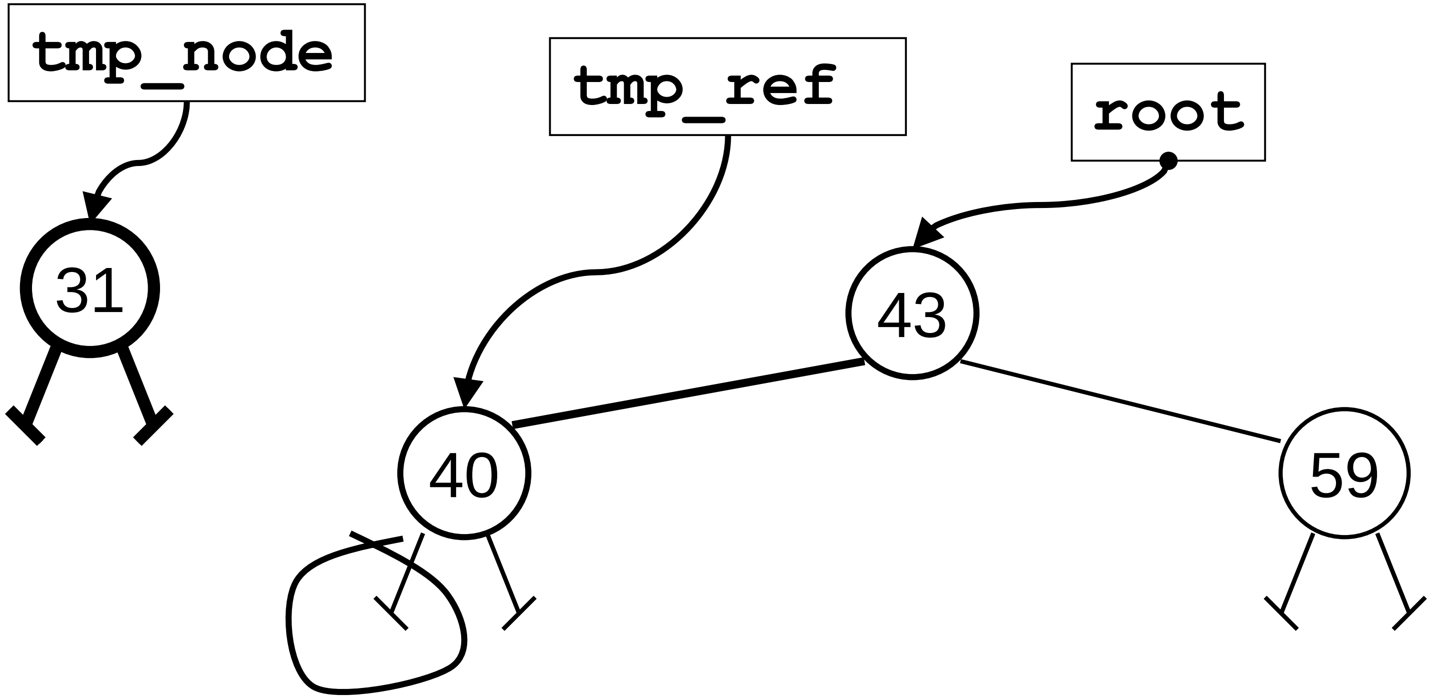
- (…)
- tmp_node→ 31 < tmp_ref→ 43? YES: Insert @ tmp_ref→ LEFT
  - ✓ tmp_ref→LEFT = NULL? // i.e., Leaf Node?
  - ✓ NO: tmp_ref = tmp_ref→LEFT

43	59	40	31	64	33	20	56	47	28	89



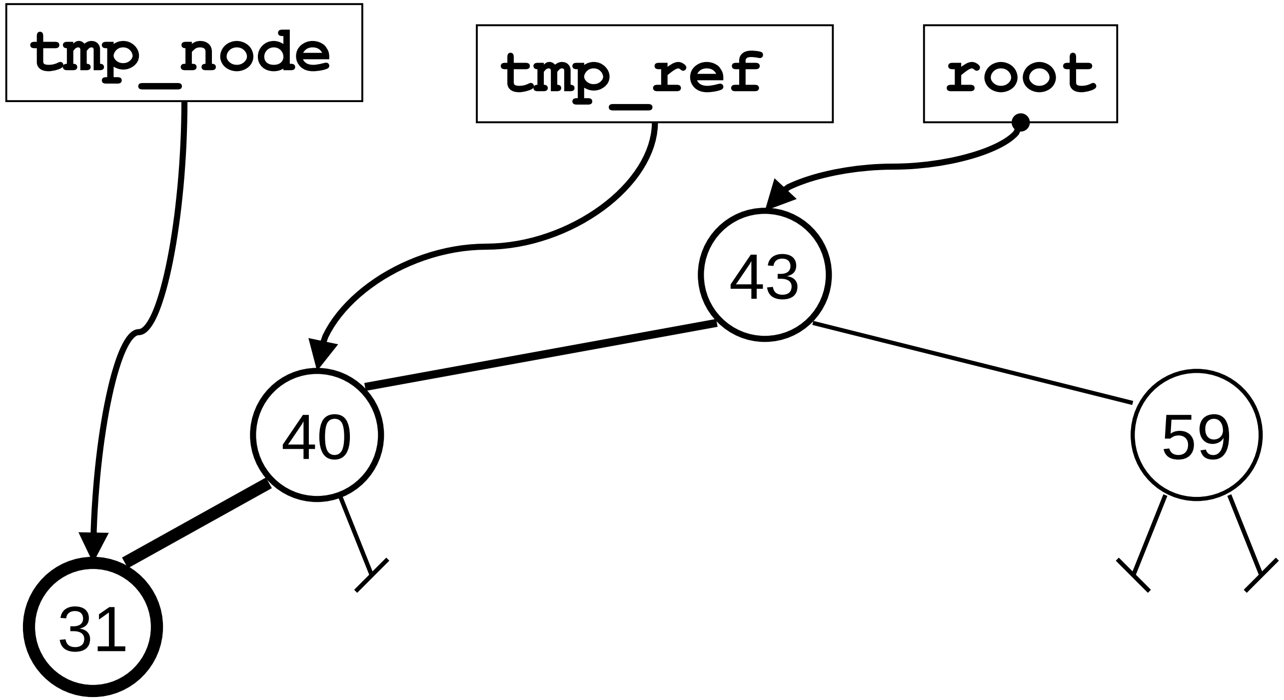
- (…)
- tmp_node→ 31 < tmp_ref→ 40? YES: Insert @ tmp_ref→ LEFT

43	59	40	31	64	33	20	56	47	28	89



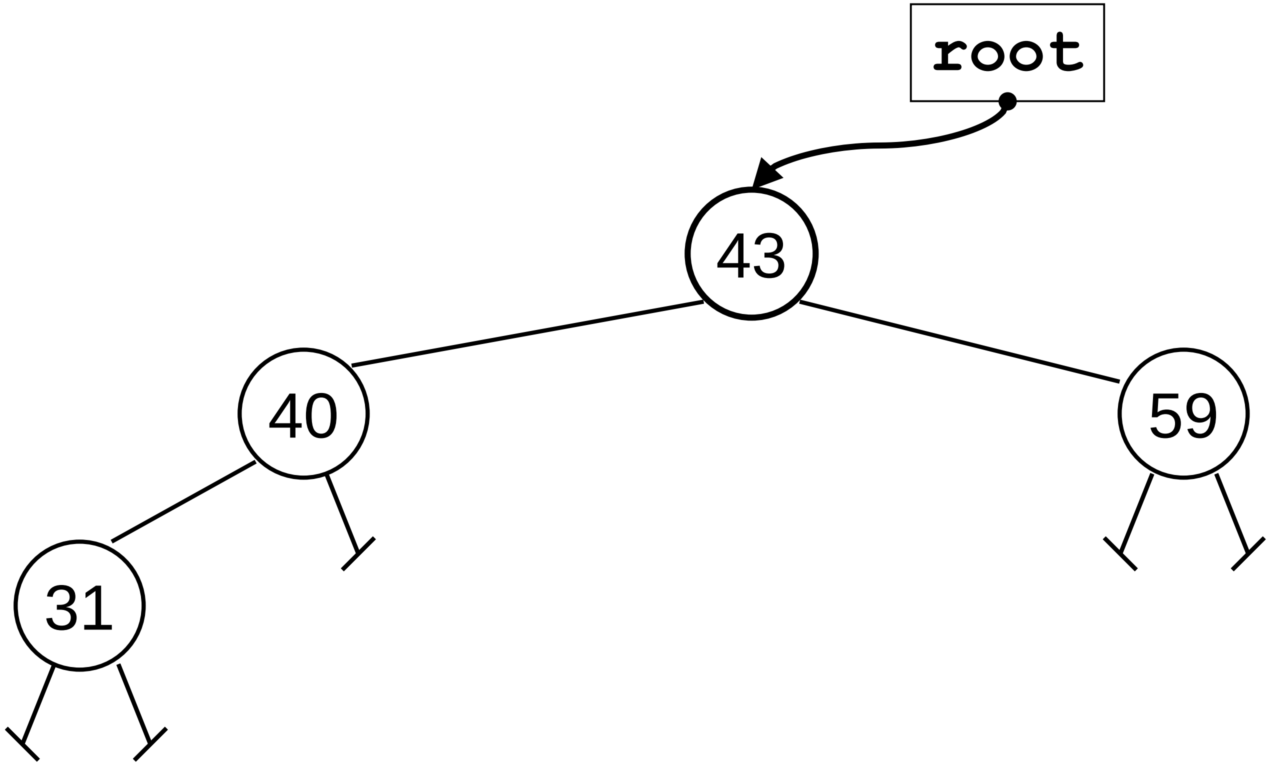
- (…)
- tmp_node→ 31 < tmp_ref→ 40? YES: Insert @ tmp_ref→ LEFT
  - ✓ tmp_ref→LEFT = NULL? // i.e., Leaf Node?
  - ✓ YES: tmp_ref = tmp_ref→LEFT

43	59	40	31	64	33	20	56	47	28	89

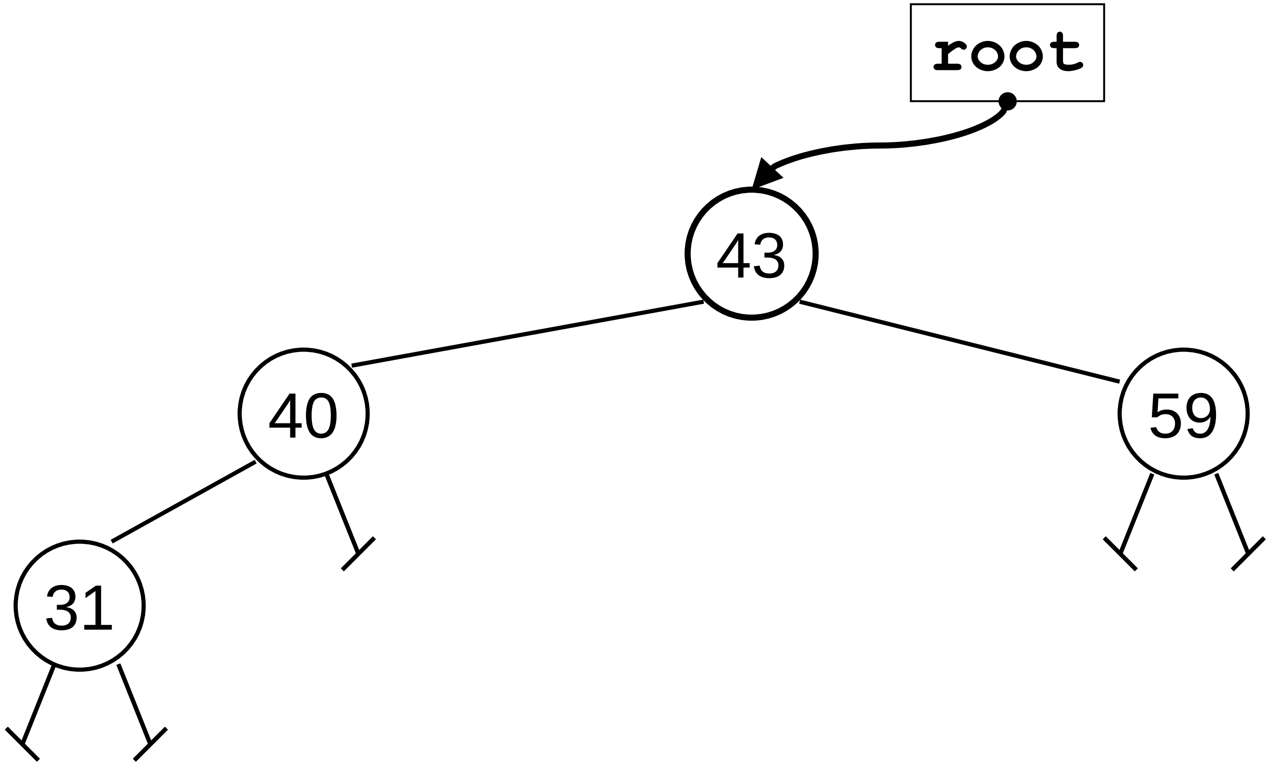


- (…)
- tmp_node→ 31 < tmp_ref→ 40? YES: Insert @ tmp_ref→ LEFT
  - ✓ tmp_ref→ LEFT = tmp_node

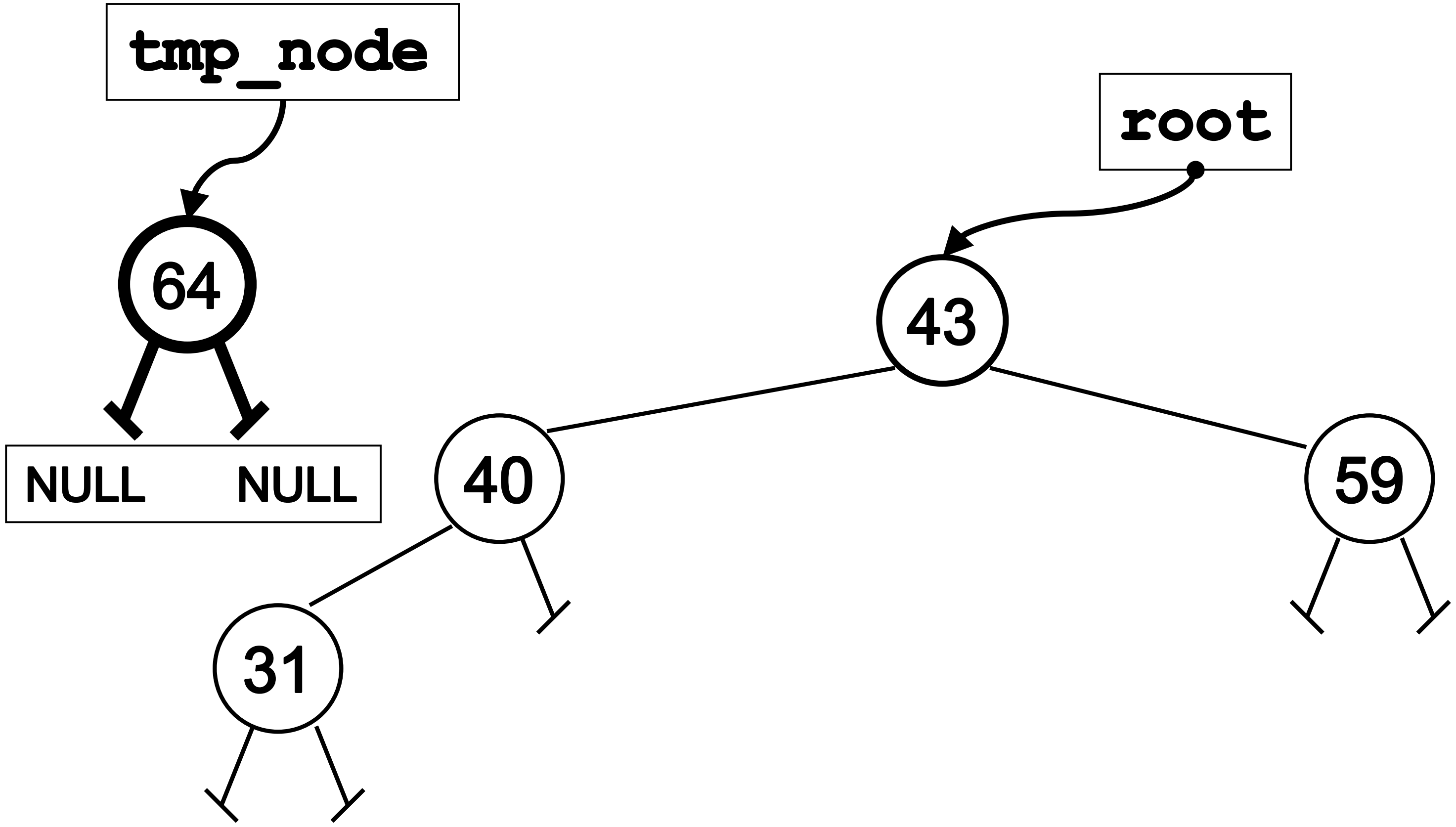
43	59	40	31	64	33	20	56	47	28	89



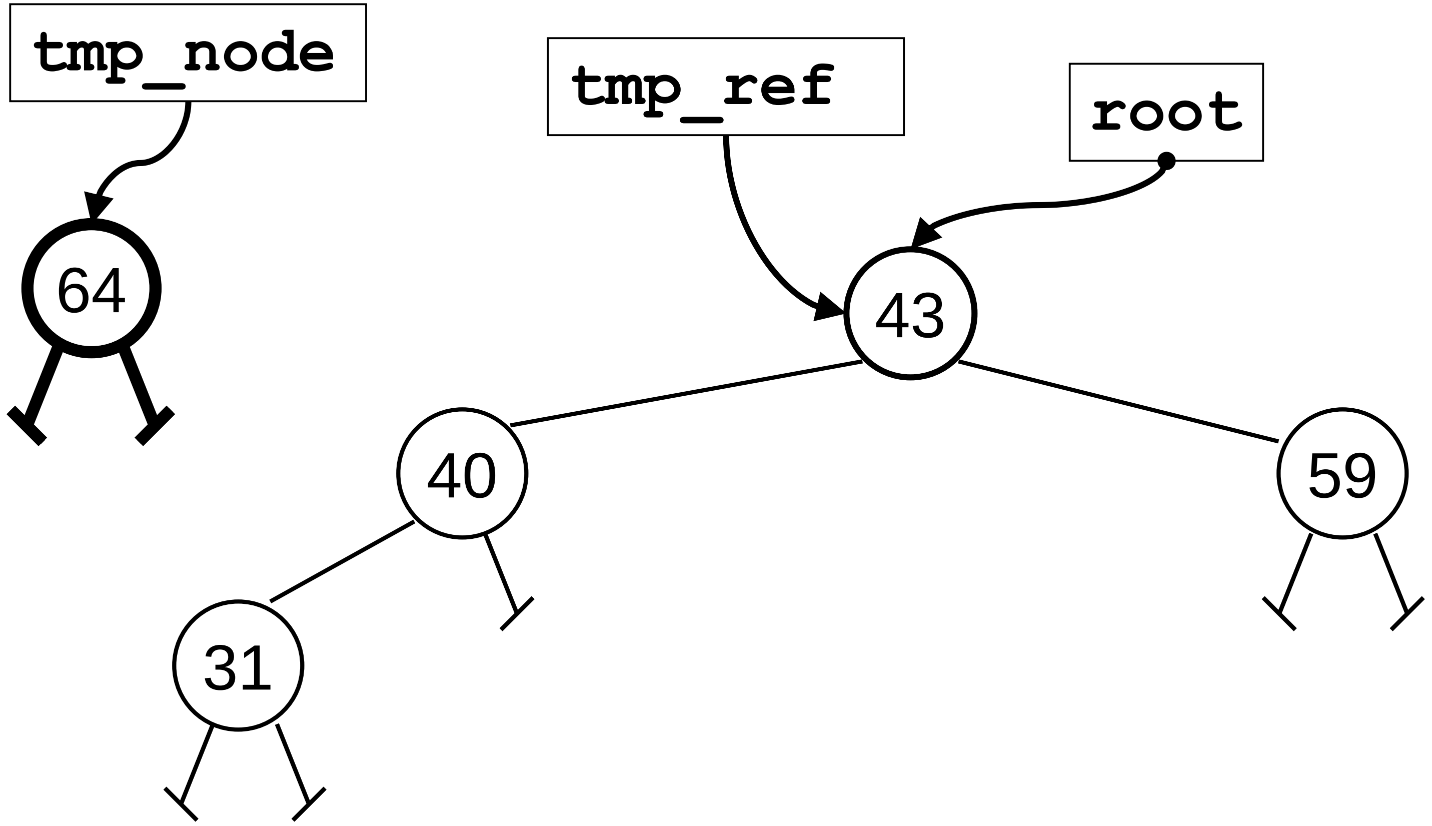
43	59	40	31	64	33	20	56	47	28	89



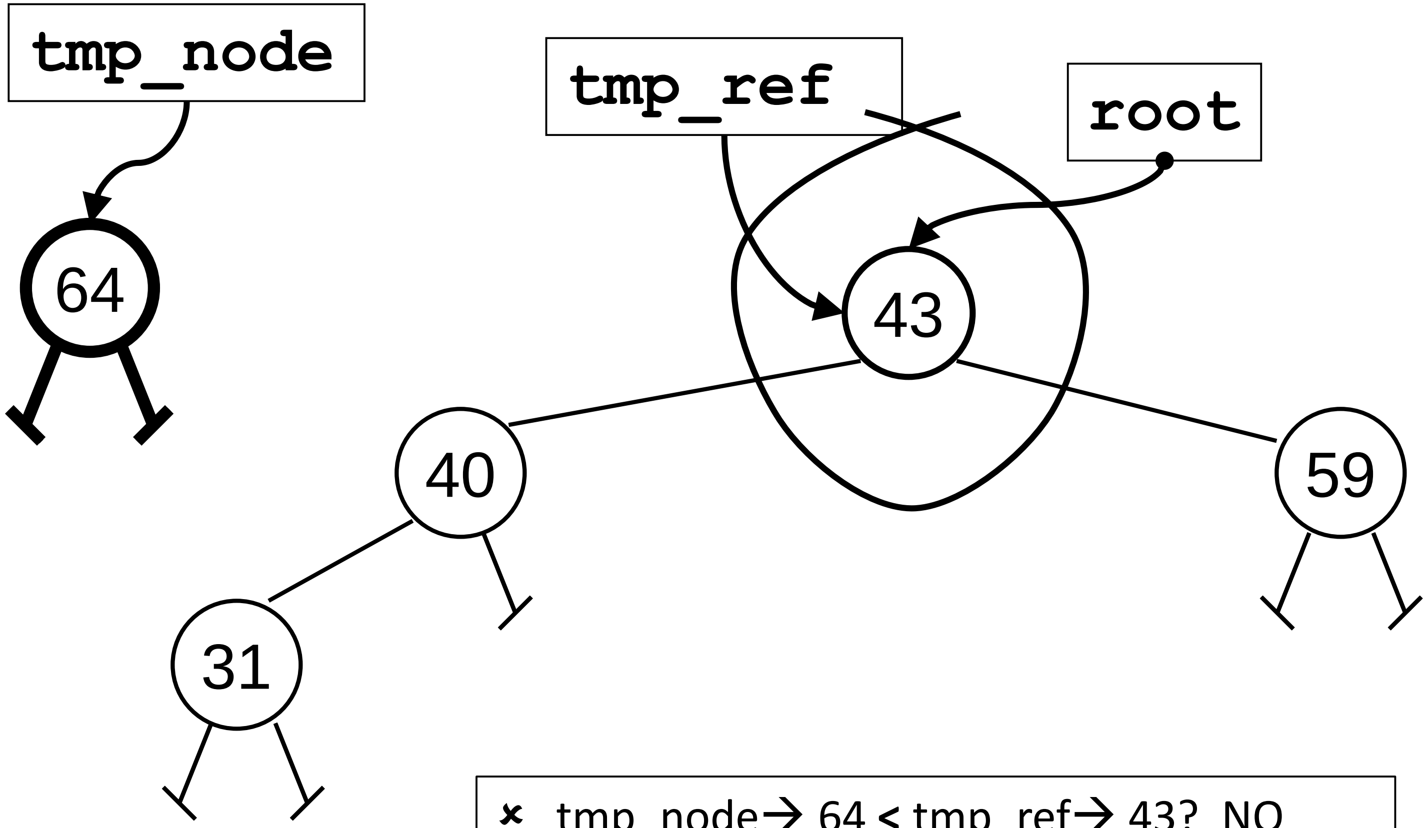
43	59	40	31	64	33	20	56	47	28	89



43	59	40	31	64	33	20	56	47	28	89

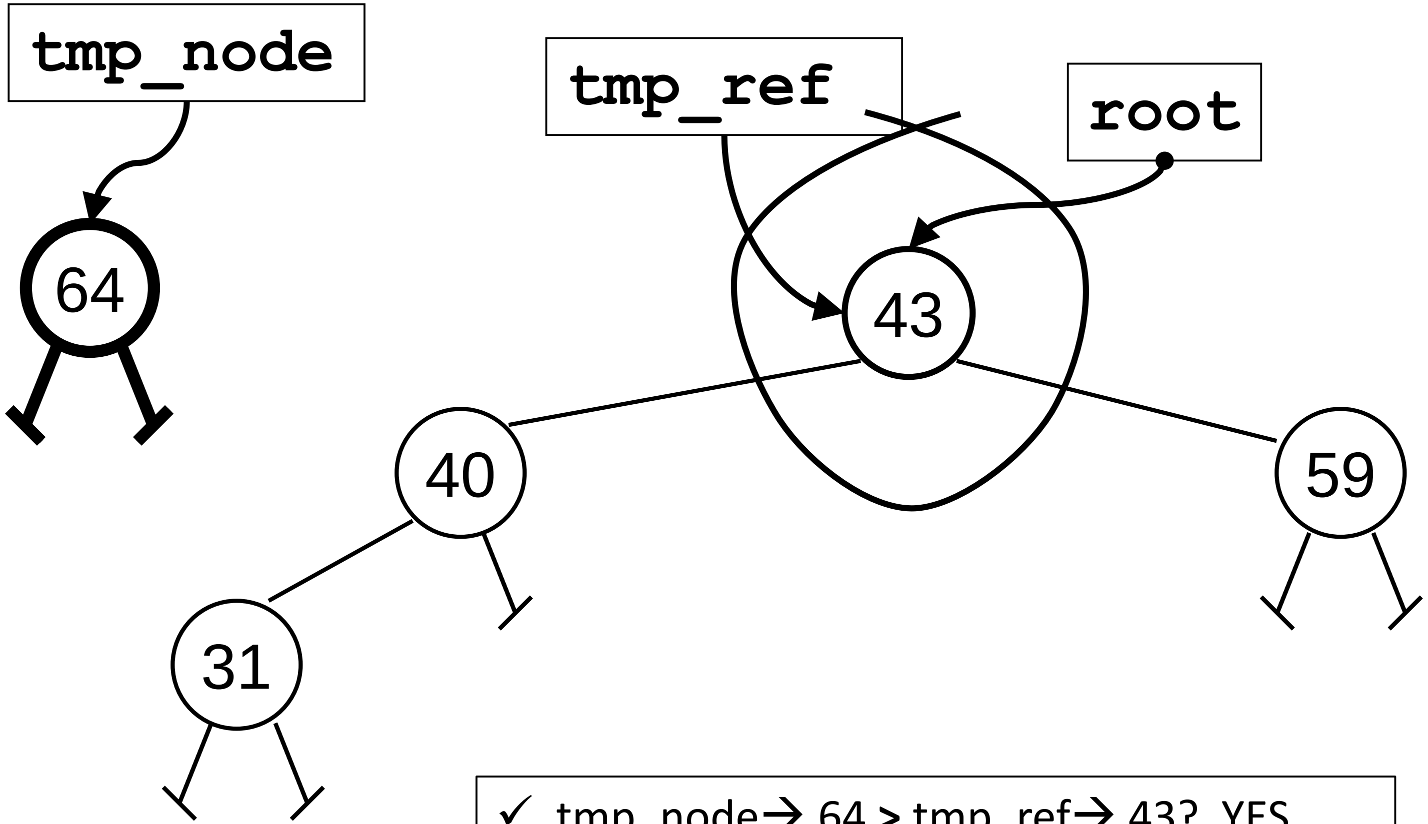


43	59	40	31	64	33	20	56	47	28	89



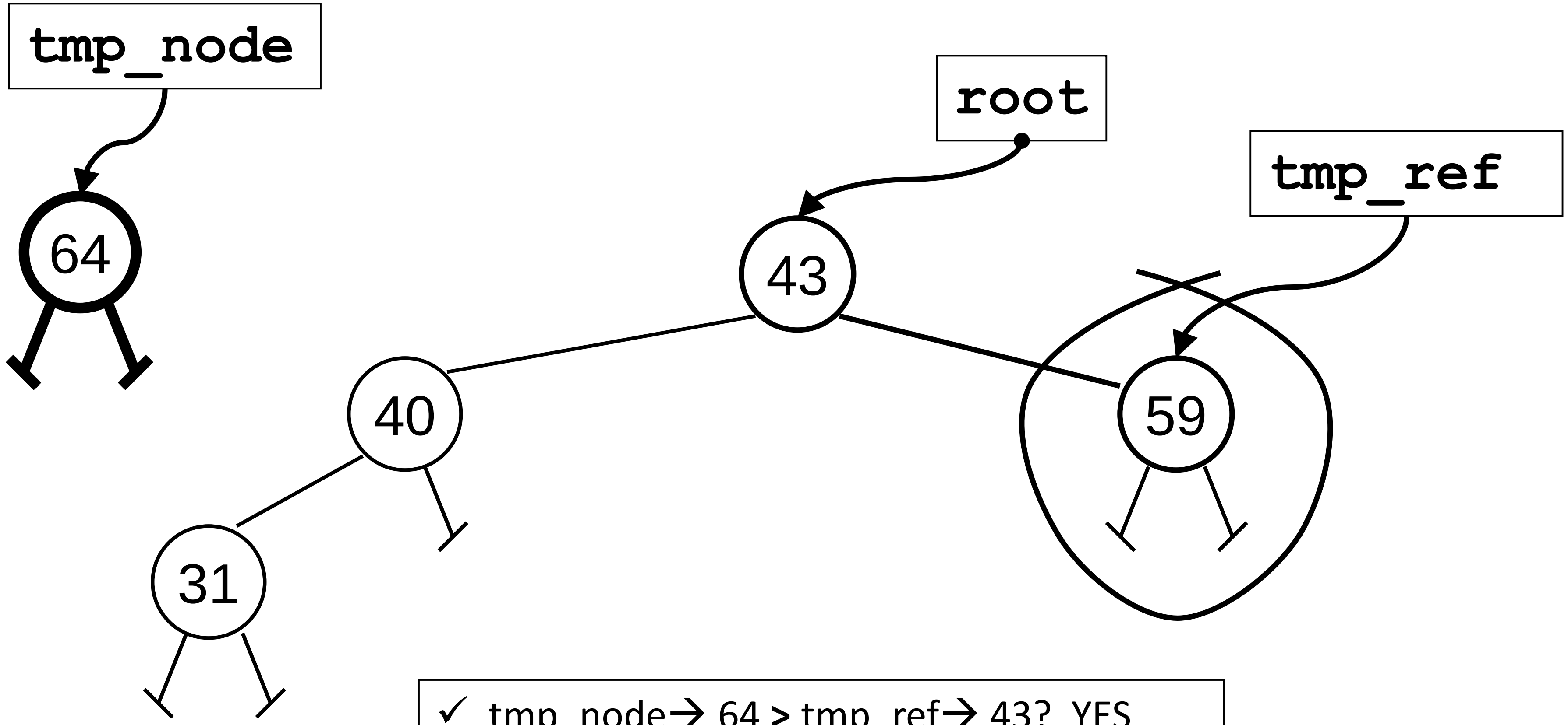
- ✗ tmp_node→ 64 < tmp_ref→ 43? NO
- ✓ tmp_node→ 64 > tmp_ref→ 43? YES

43	59	40	31	64	33	20	56	47	28	89



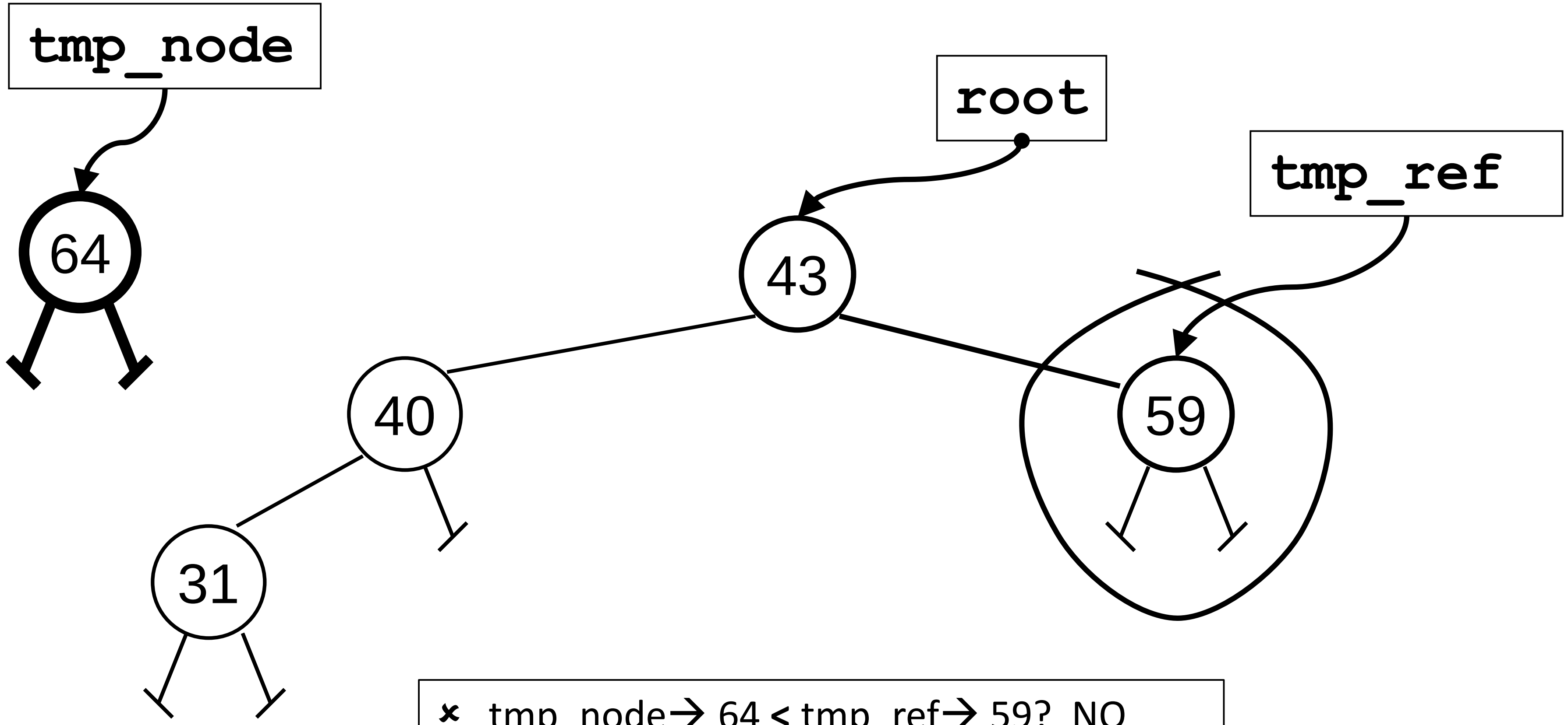
- ✓ tmp_node→ 64 > tmp_ref→ 43? YES
- ✓ tmp_ref = tmp_ref→ RIGHT

43	59	40	31	64	33	20	56	47	28	89



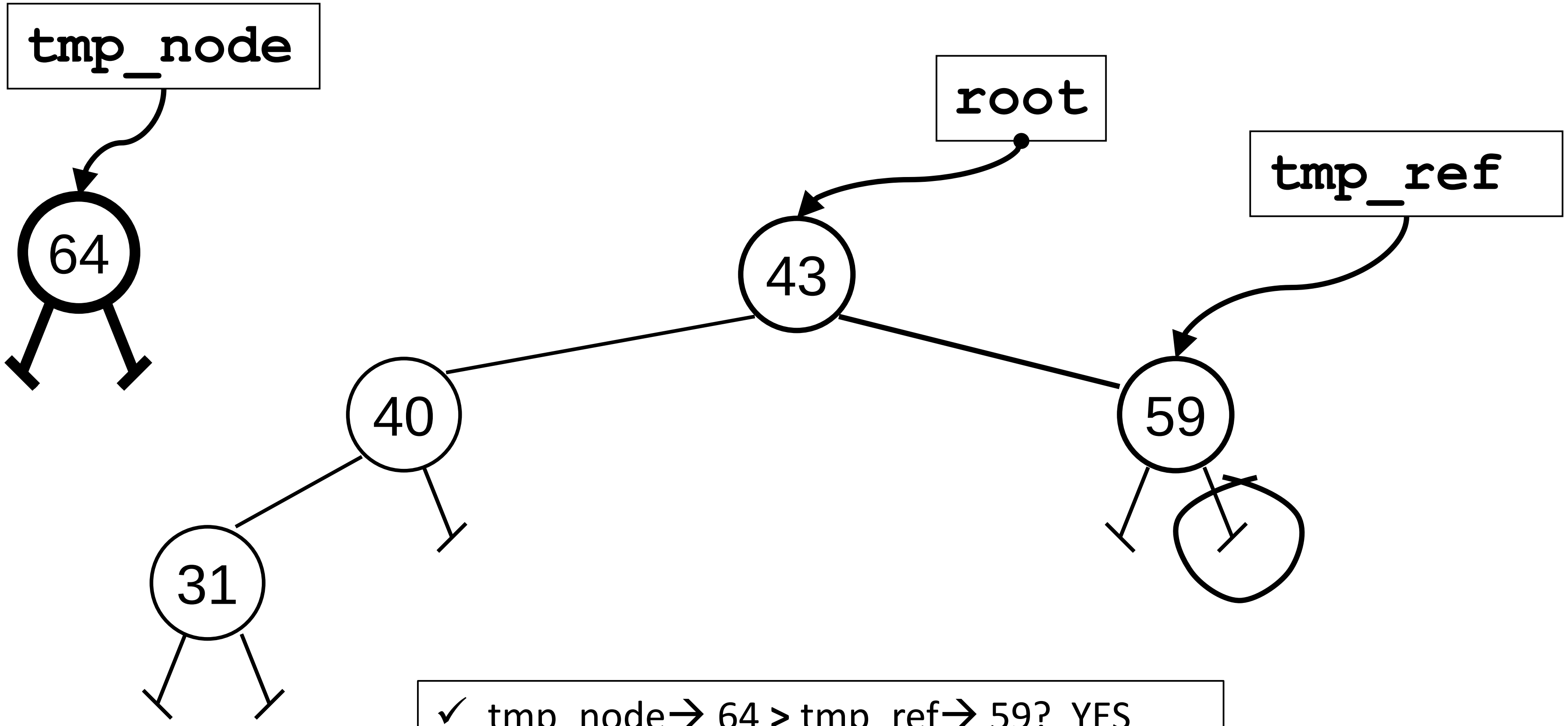
- ✓ tmp_node→ 64 > tmp_ref→ 43? YES
- ✓ tmp_ref = tmp_ref→ RIGHT

43	59	40	31	64	33	20	56	47	28	89



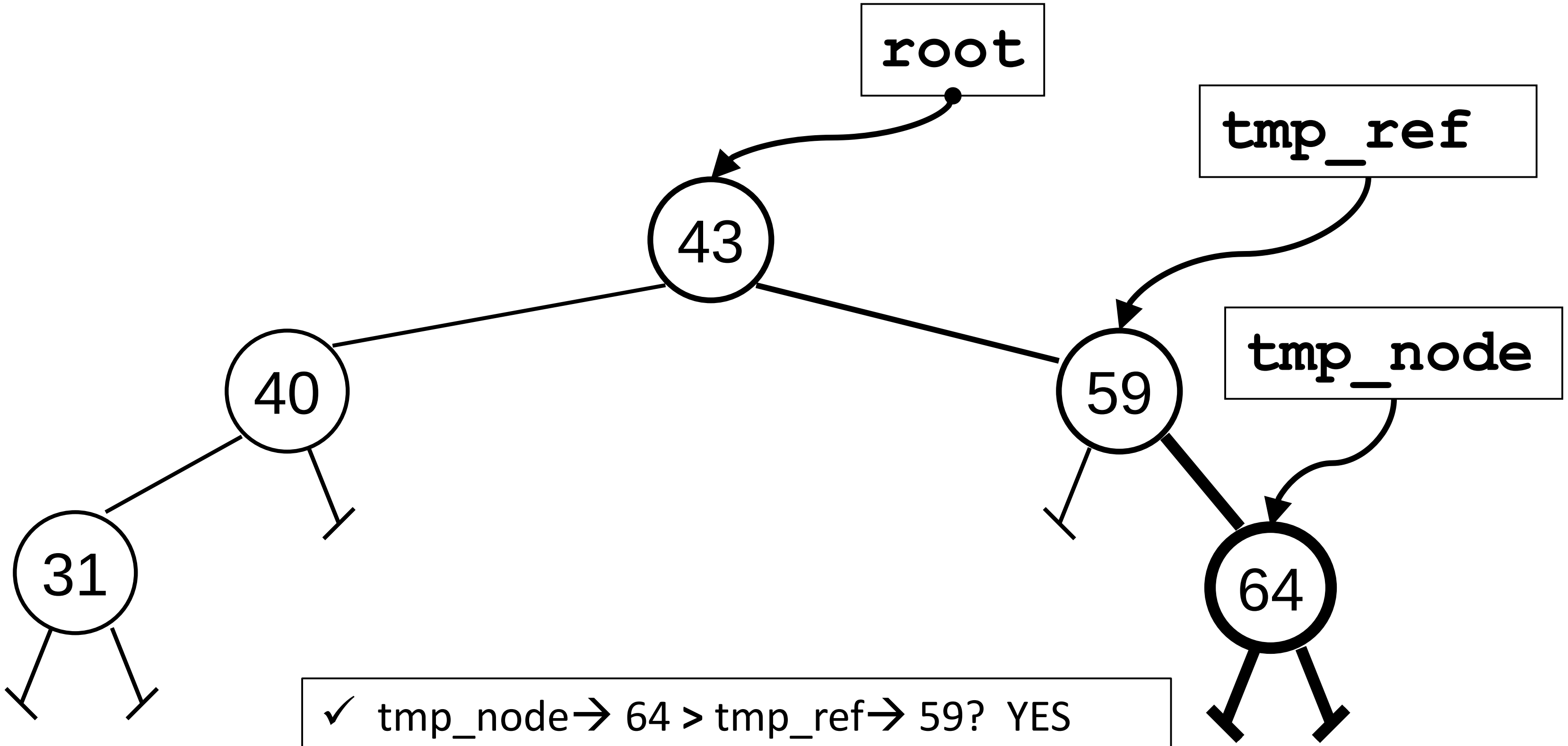
- ✗ tmp_node→ 64 < tmp_ref→ 59? NO
- ✓ tmp_node→ 64 > tmp_ref→ 59? YES

43	59	40	31	64	33	20	56	47	28	89



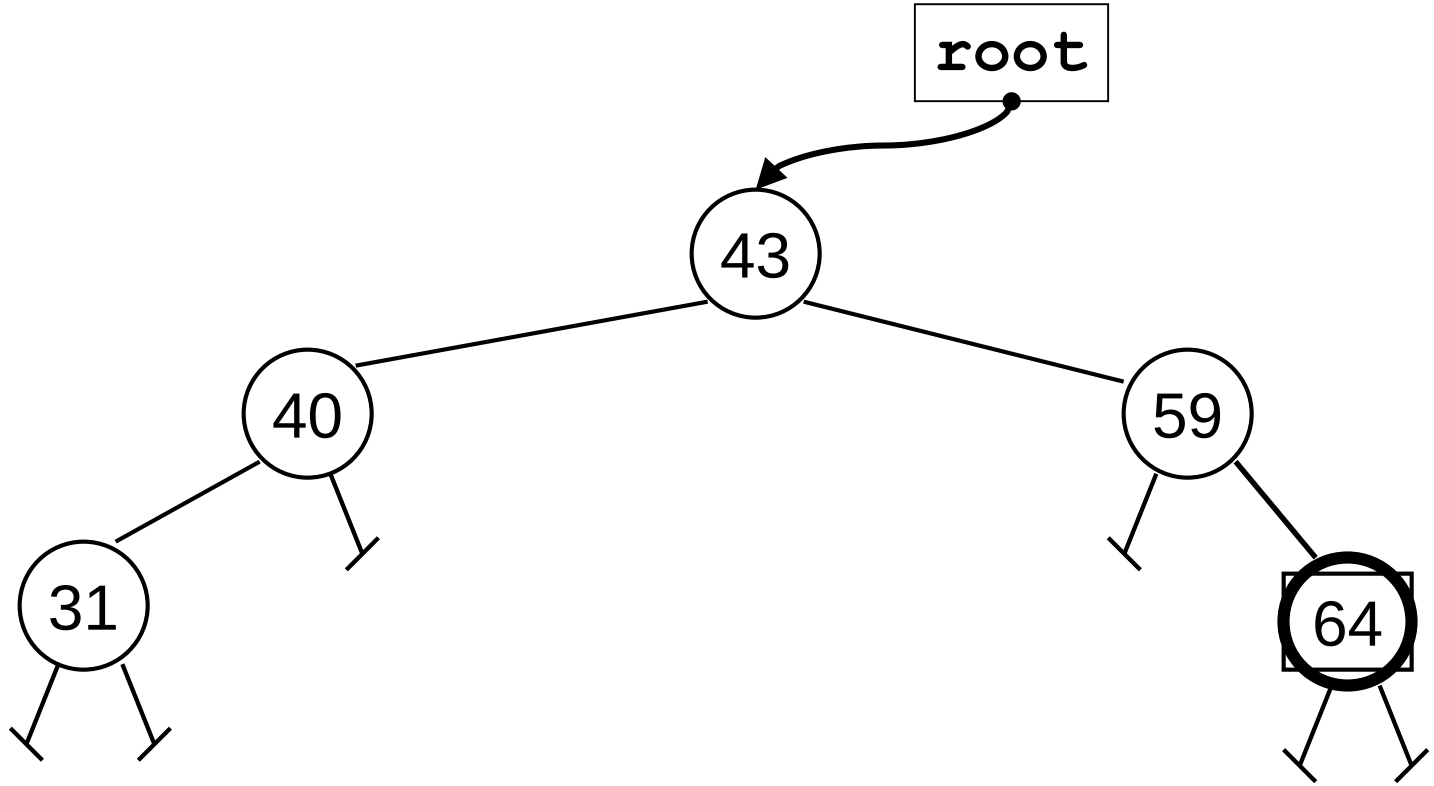
- ✓ tmp_node→ 64 > tmp_ref→ 59? YES
- ✓ tmp_ref→ RIGHT = tmp_node

43	59	40	31	64	33	20	56	47	28	89

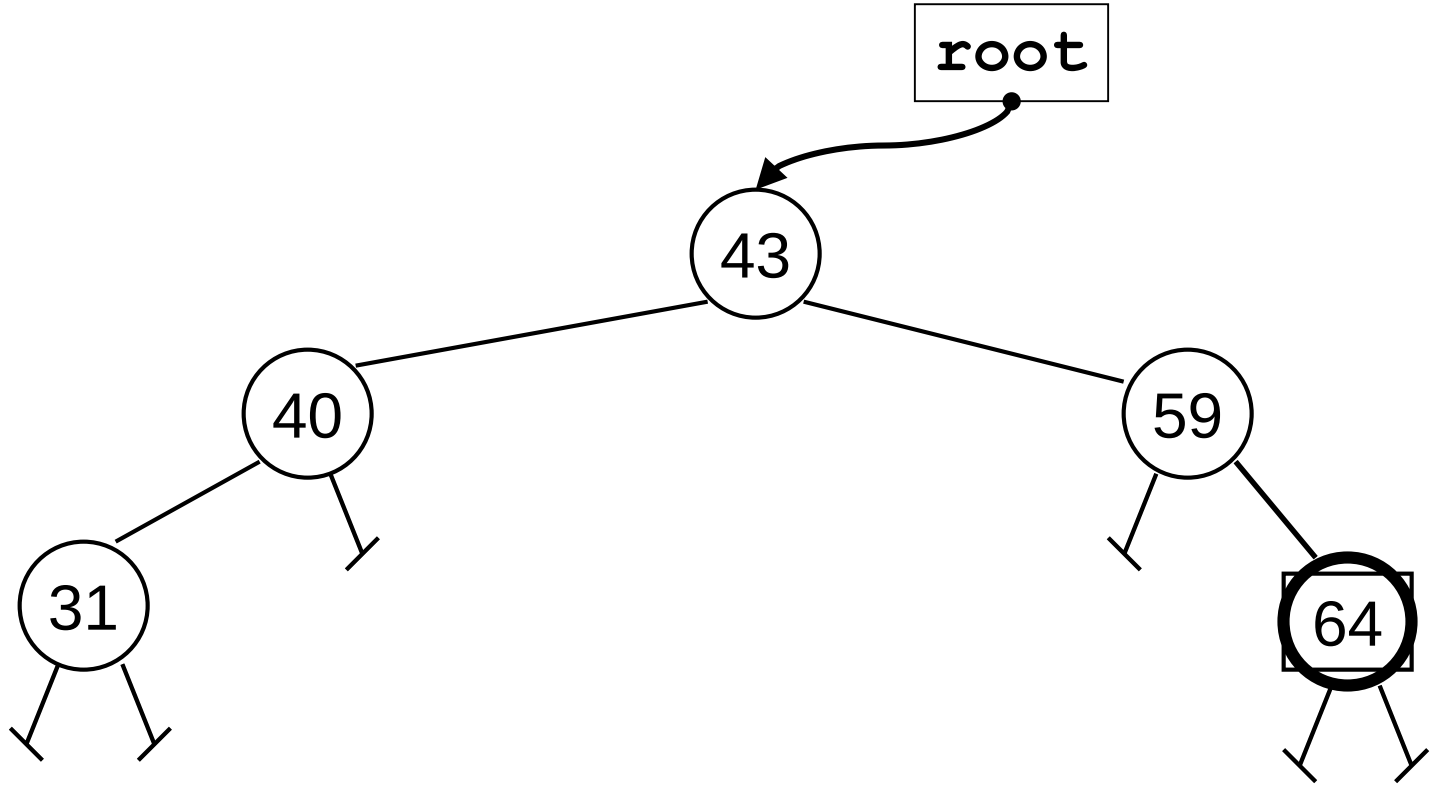


- ✓ tmp_node→ 64 > tmp_ref→ 59? YES
- ✓ tmp_ref→ RIGHT = tmp_node

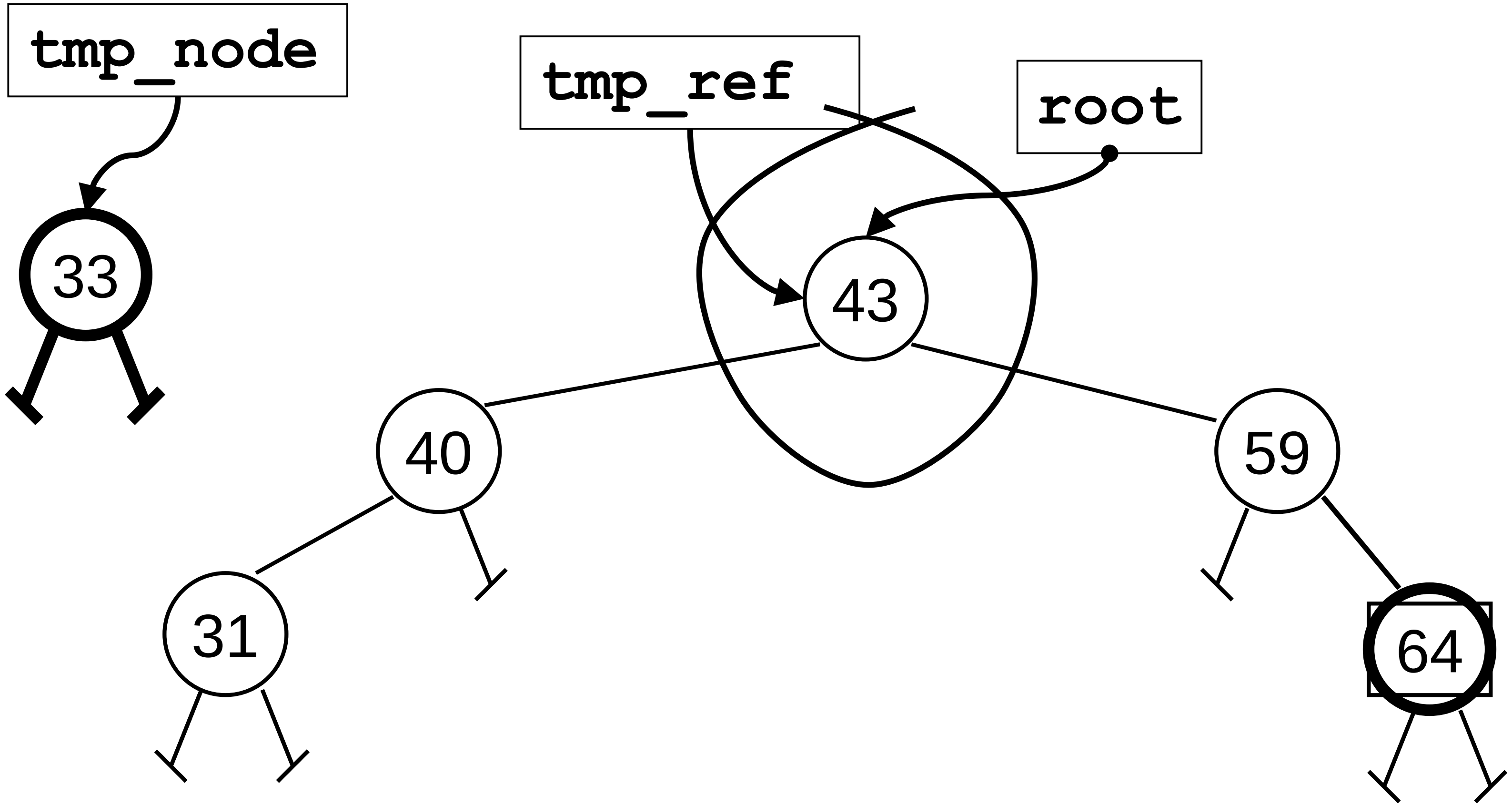
43	59	40	31	64	33	20	56	47	28	89



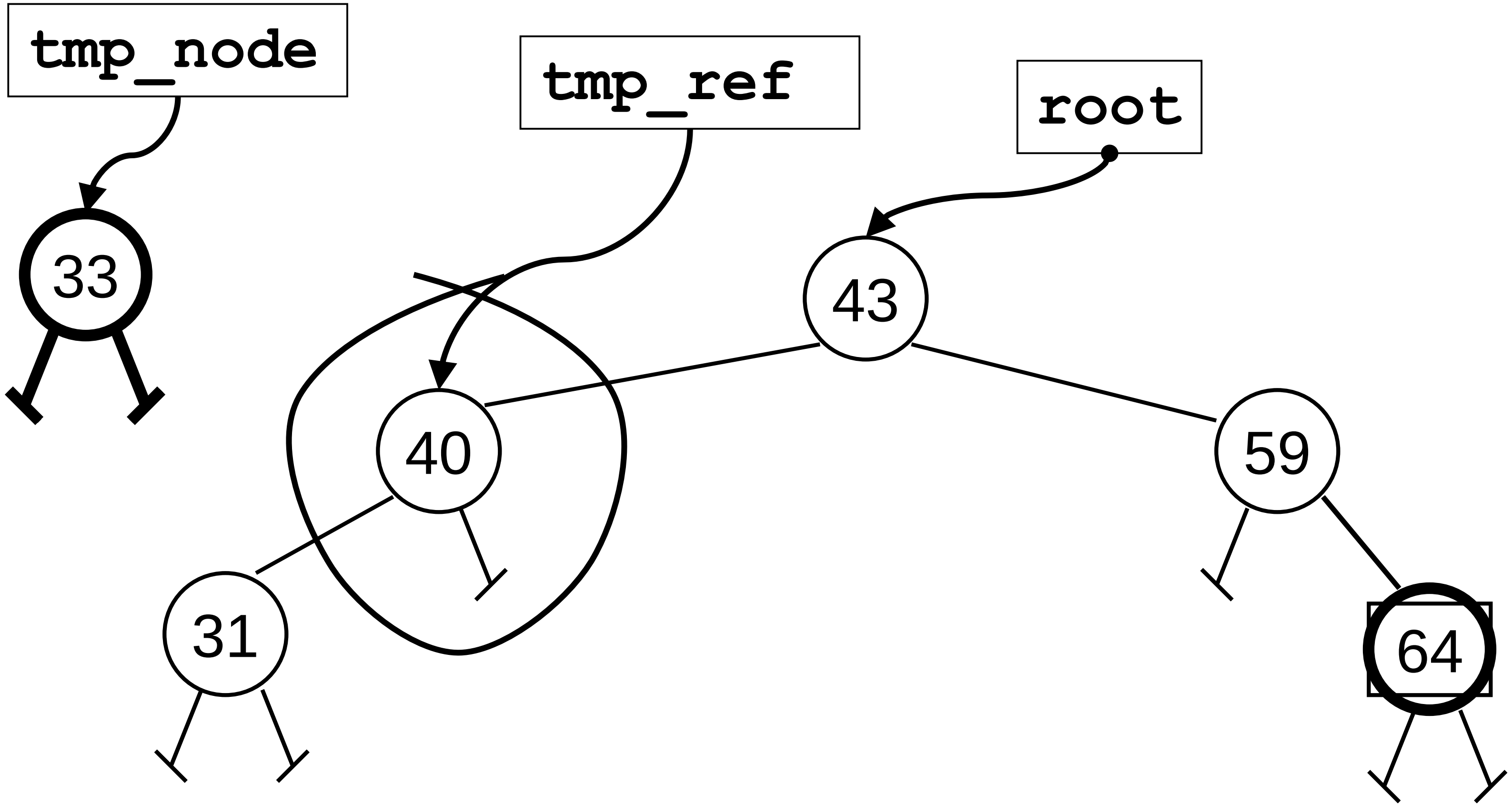
43	59	40	31	64	33	20	56	47	28	89



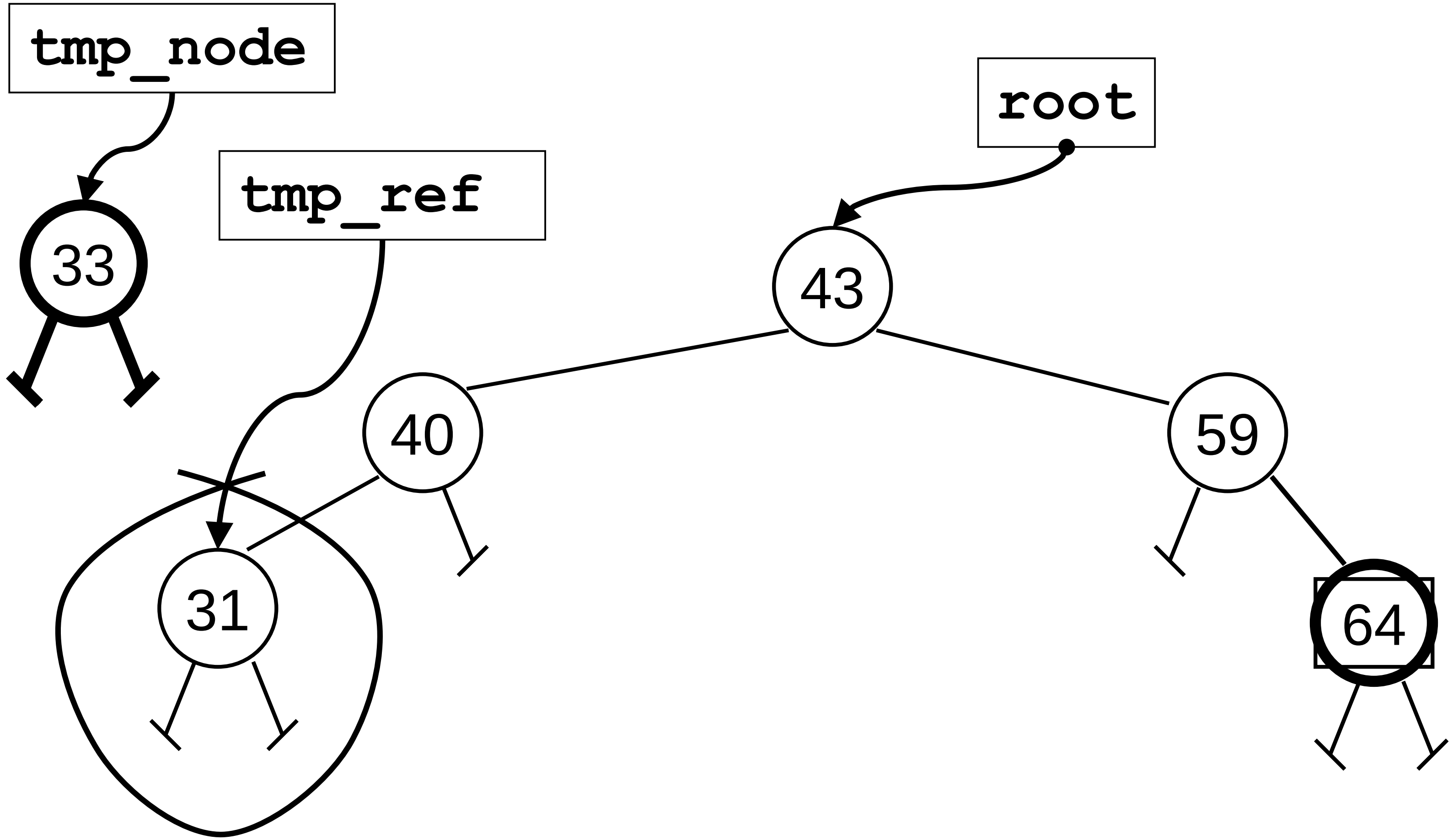
43	59	40	31	64	33	20	56	47	28	89



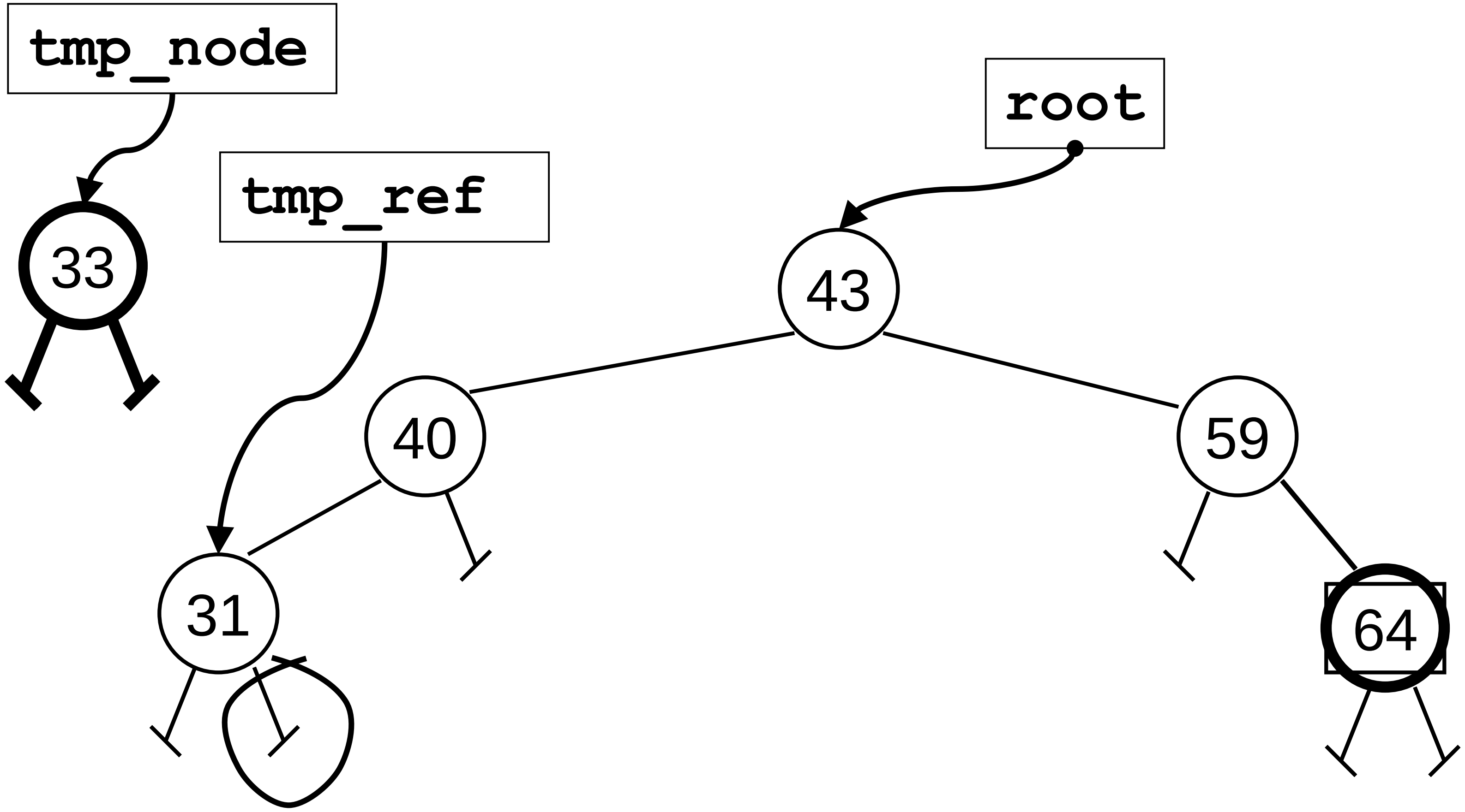
43	59	40	31	64	33	20	56	47	28	89



43	59	40	31	64	33	20	56	47	28	89

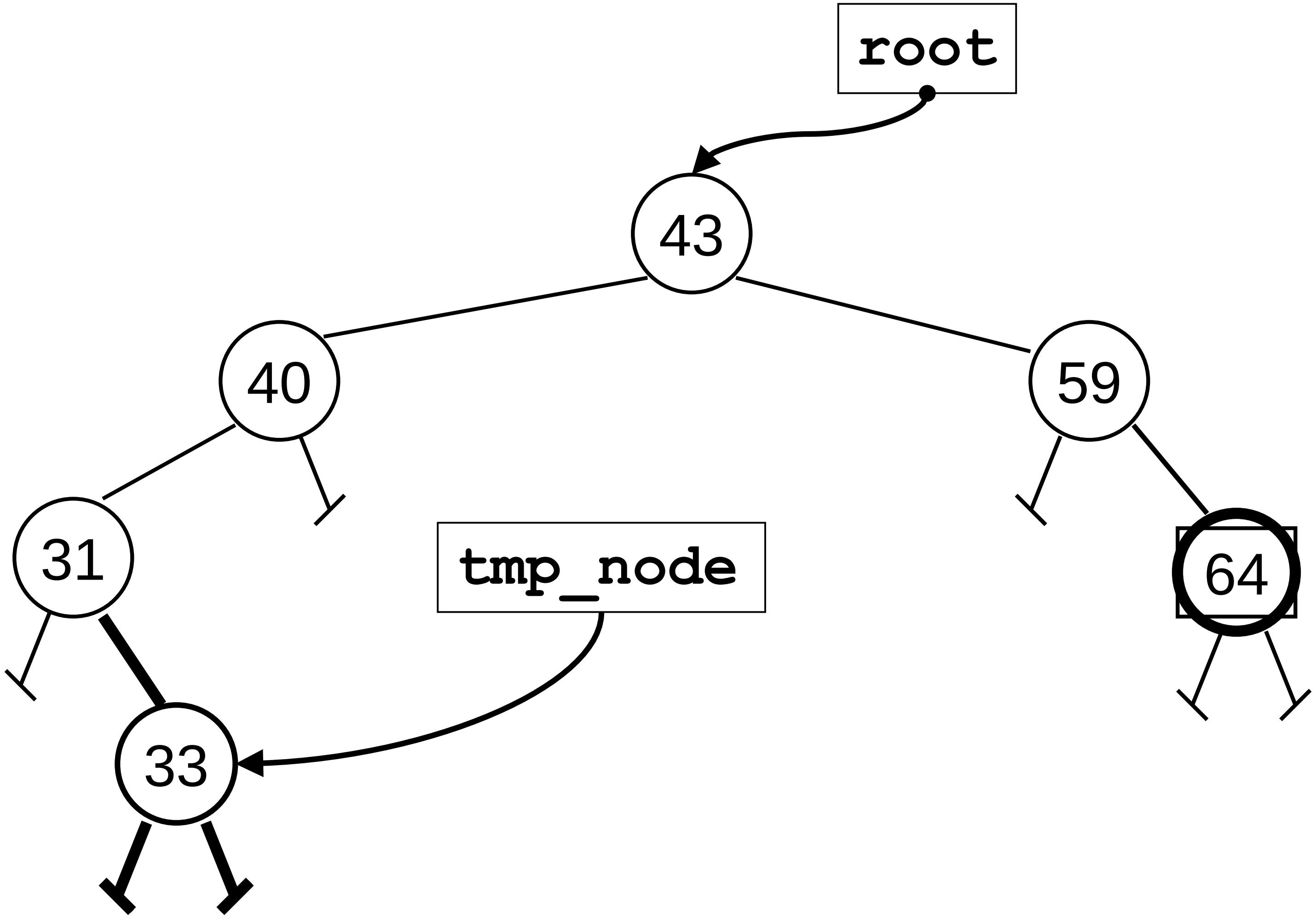


43	59	40	31	64	33	20	56	47	28	89

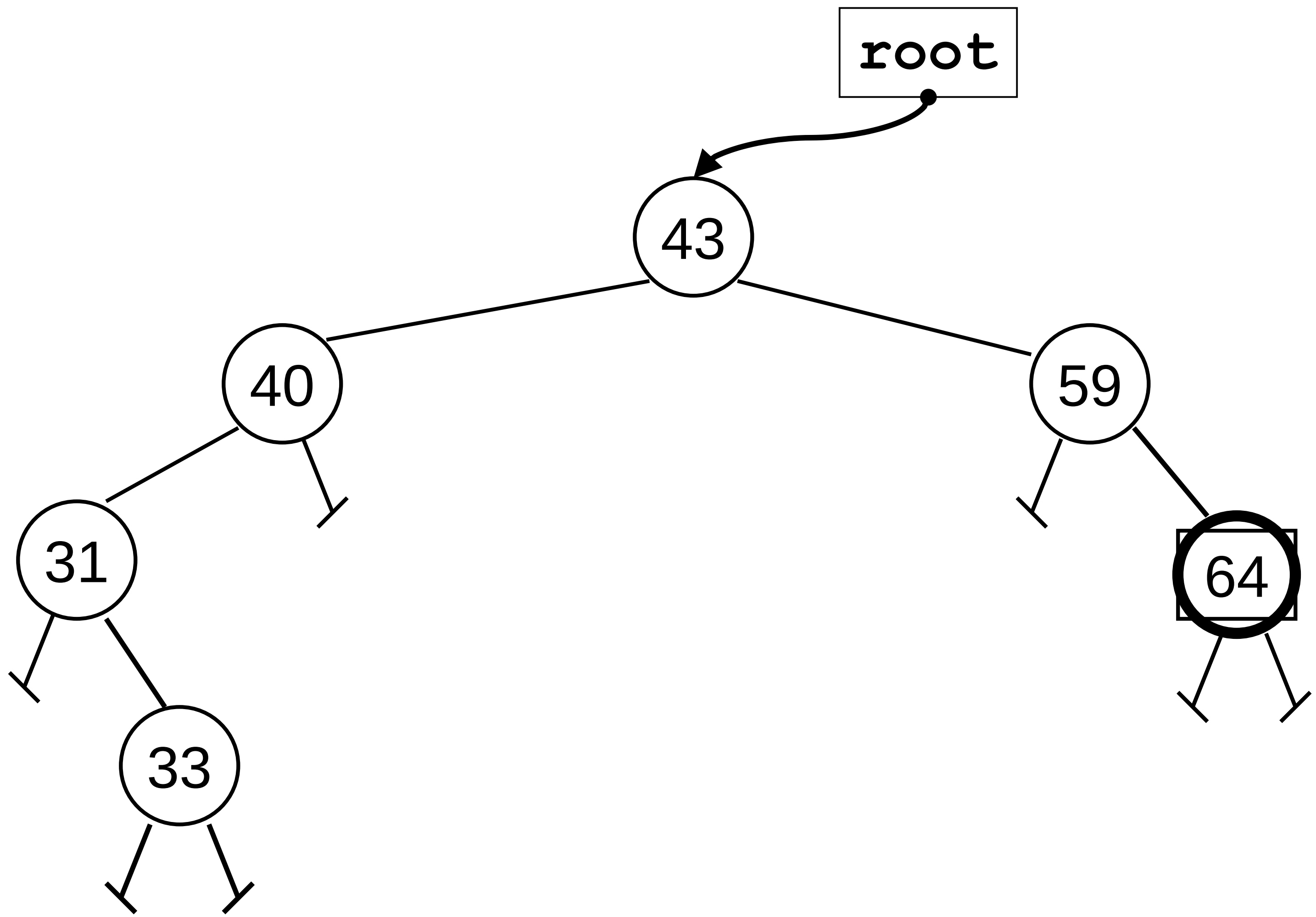


43	59	40	31	64	33	20	56	47	28	89

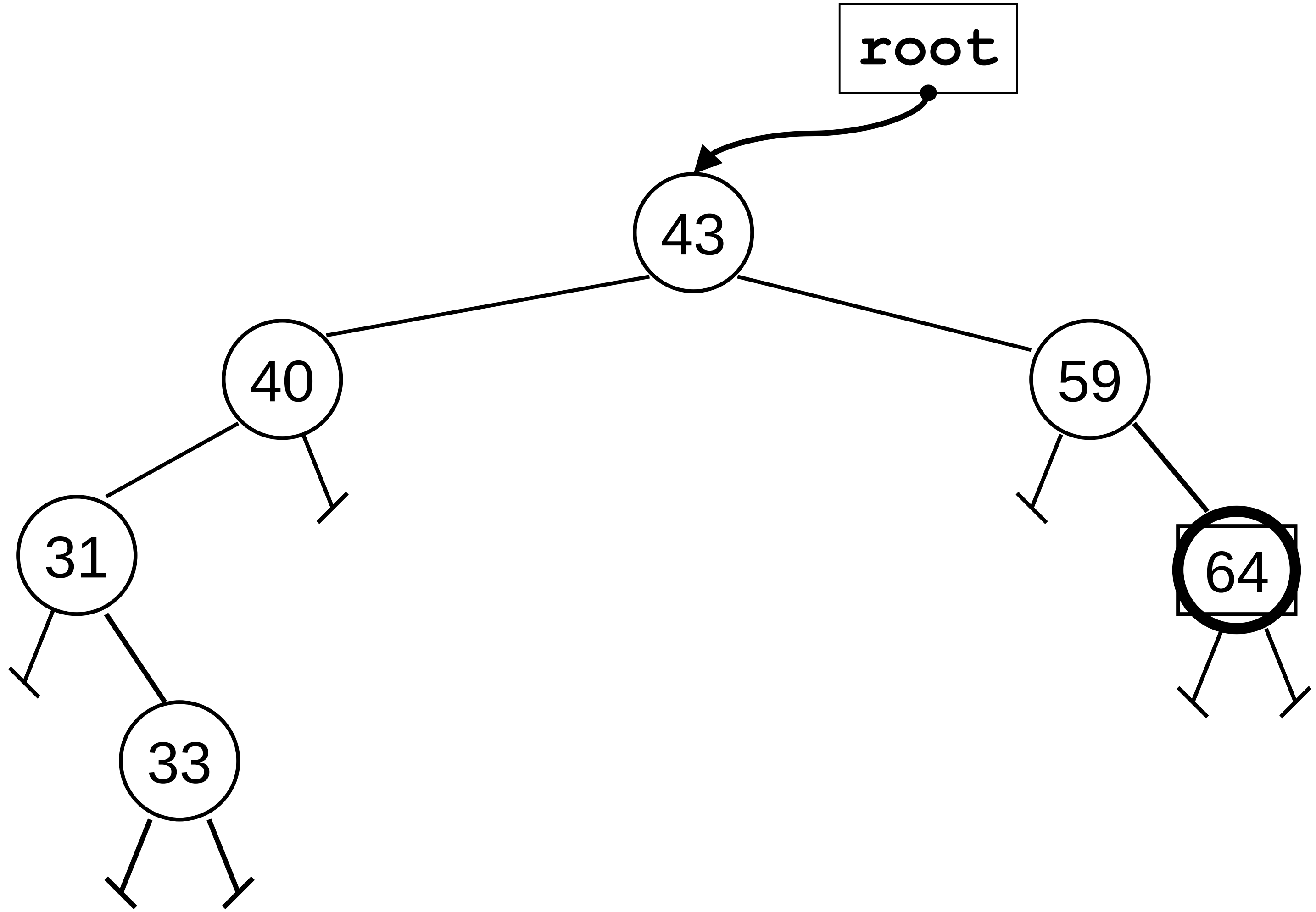
UNIVERSITY OF CALGARY



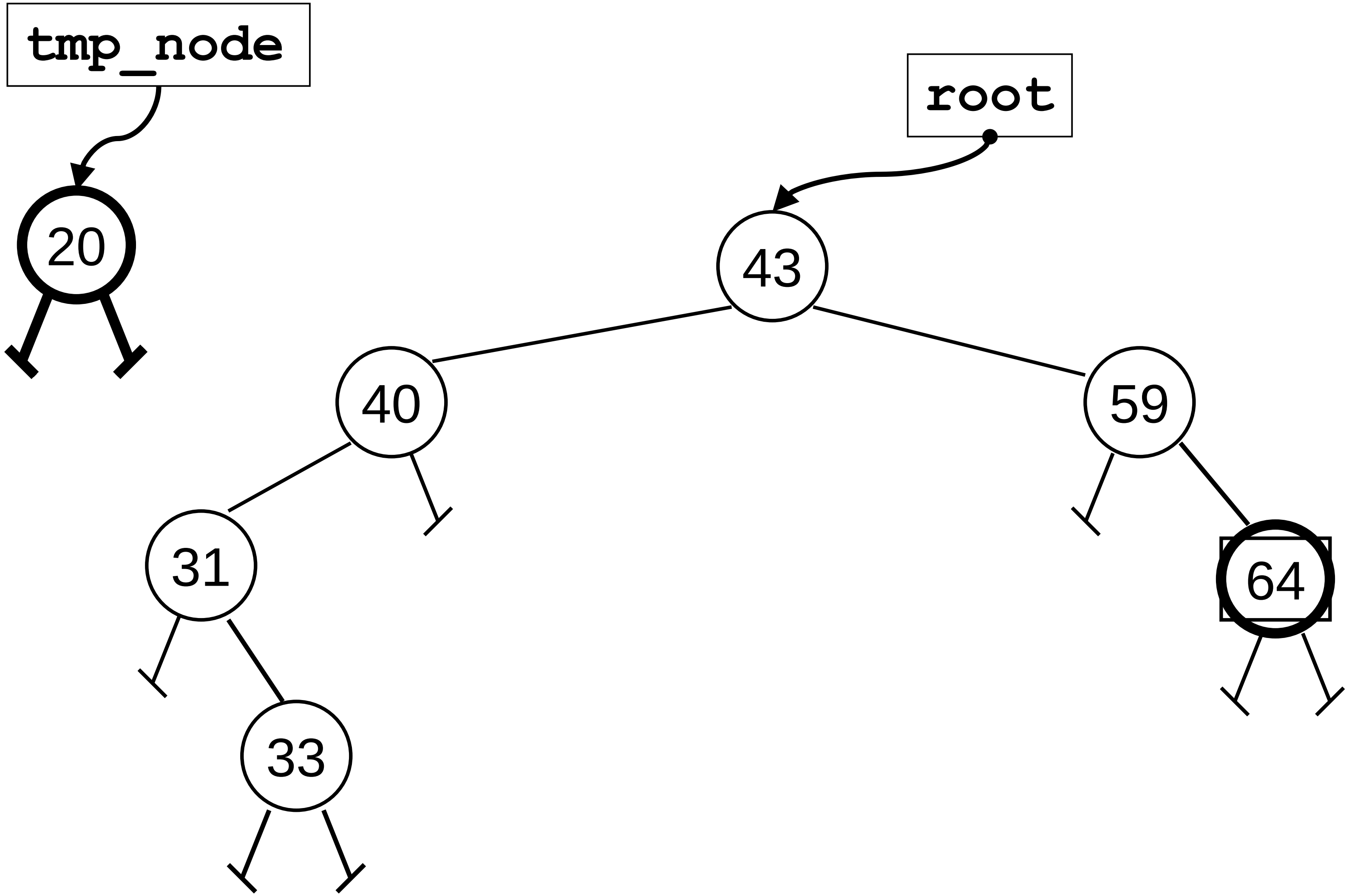
43	59	40	31	64	33	20	56	47	28	89



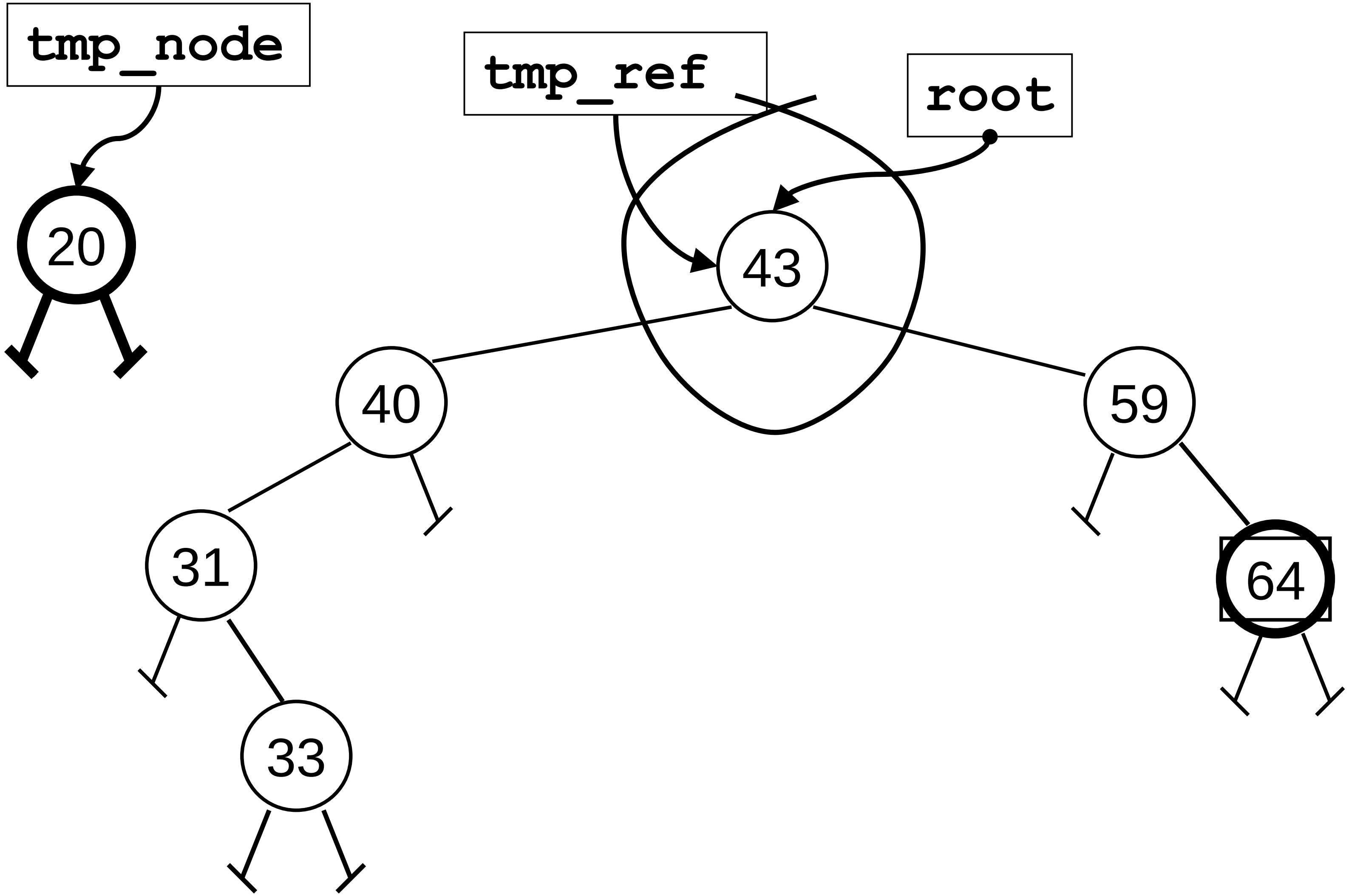
43	59	40	31	64	33	20	56	47	28	89



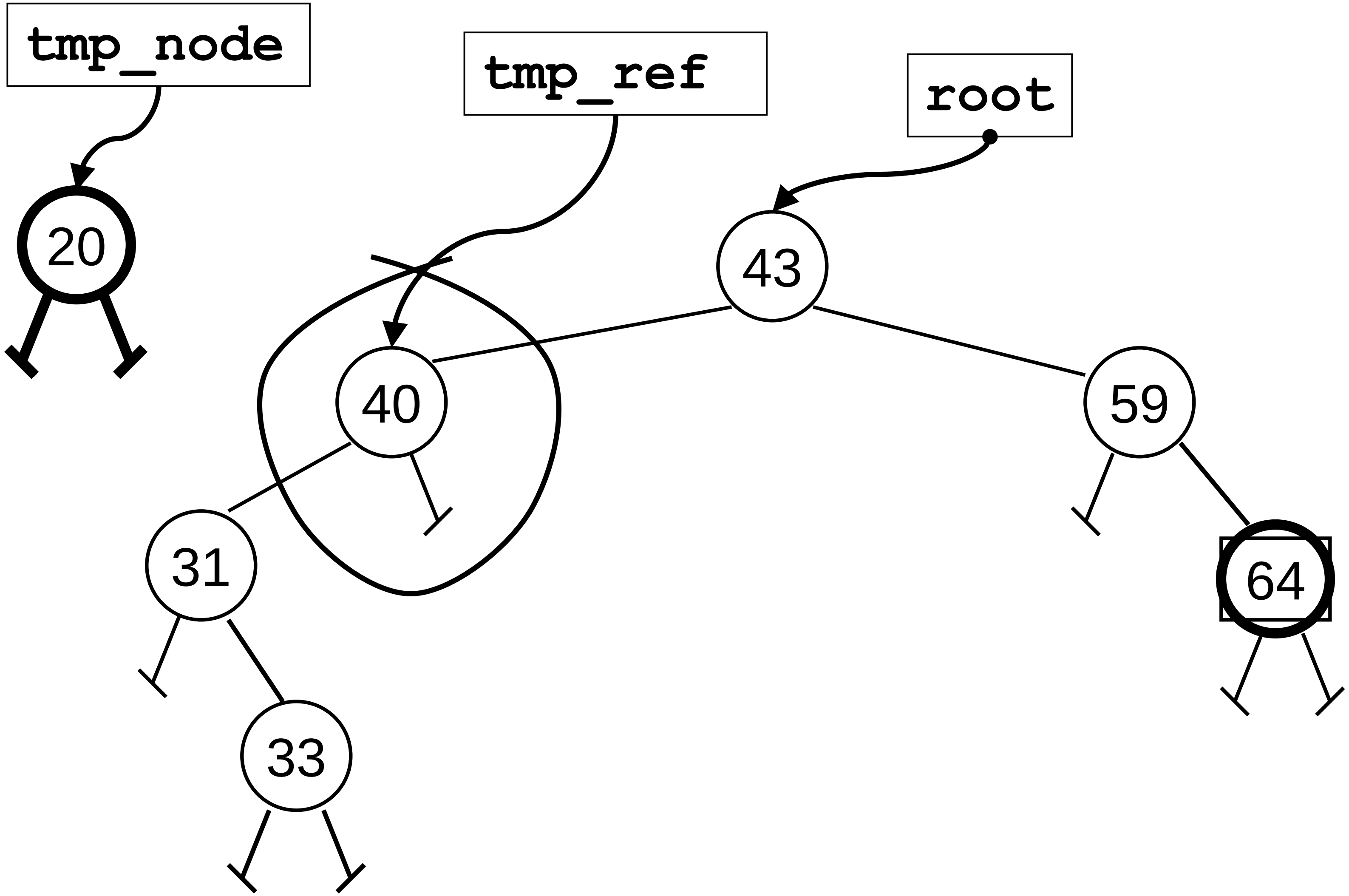
43	59	40	31	64	33	20	56	47	28	89



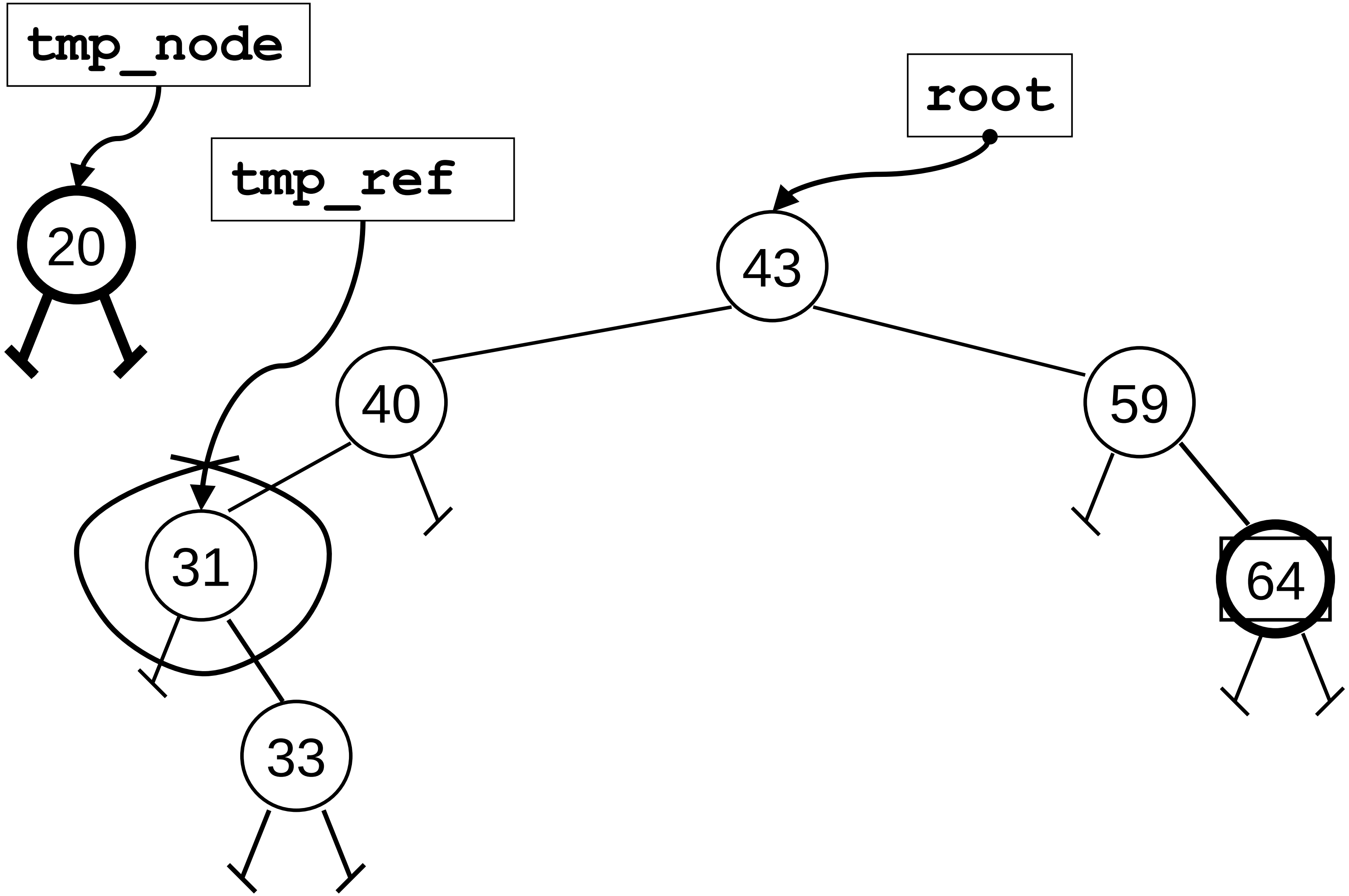
43	59	40	31	64	33	20	56	47	28	89



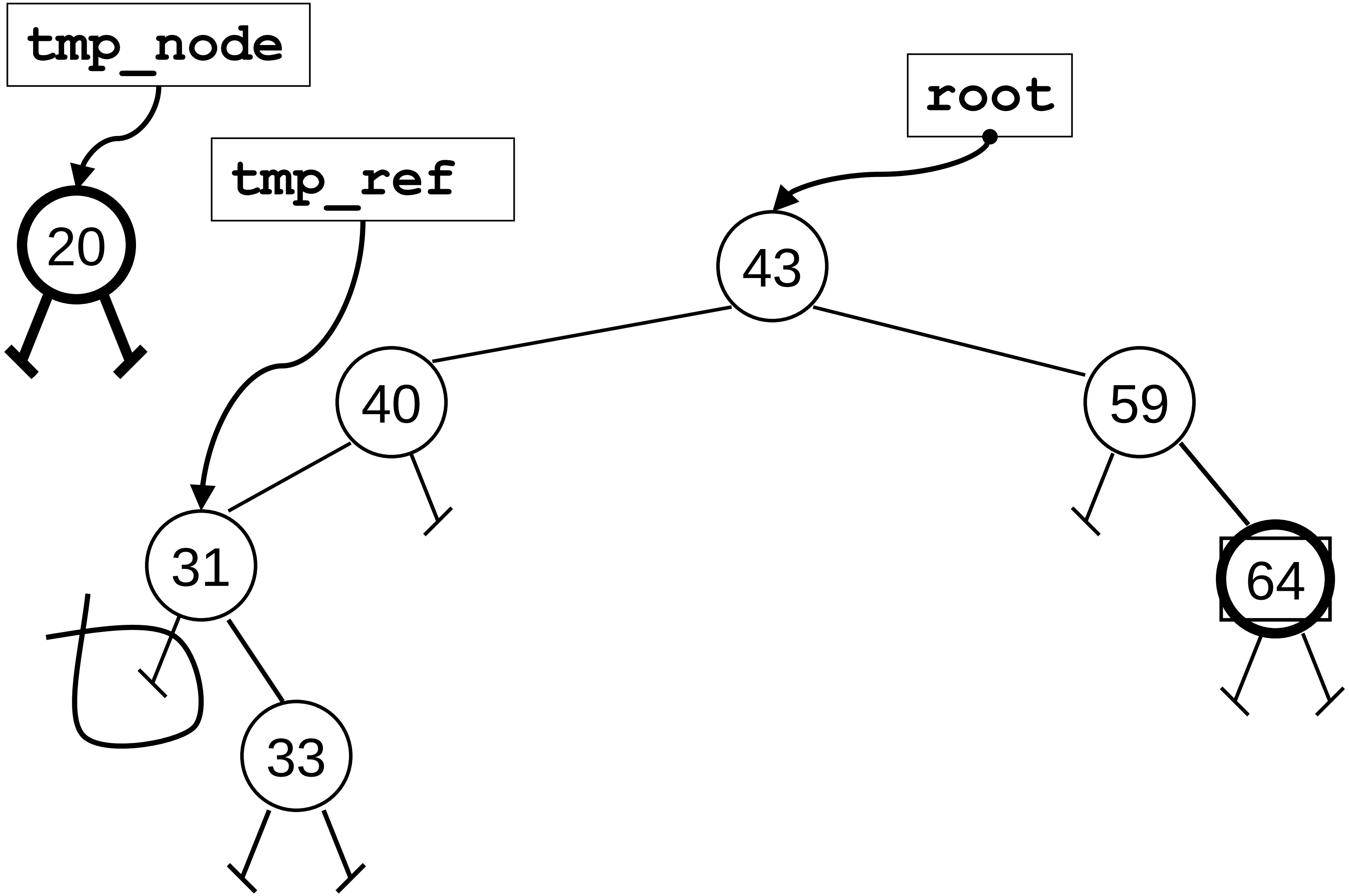
43	59	40	31	64	33	20	56	47	28	89



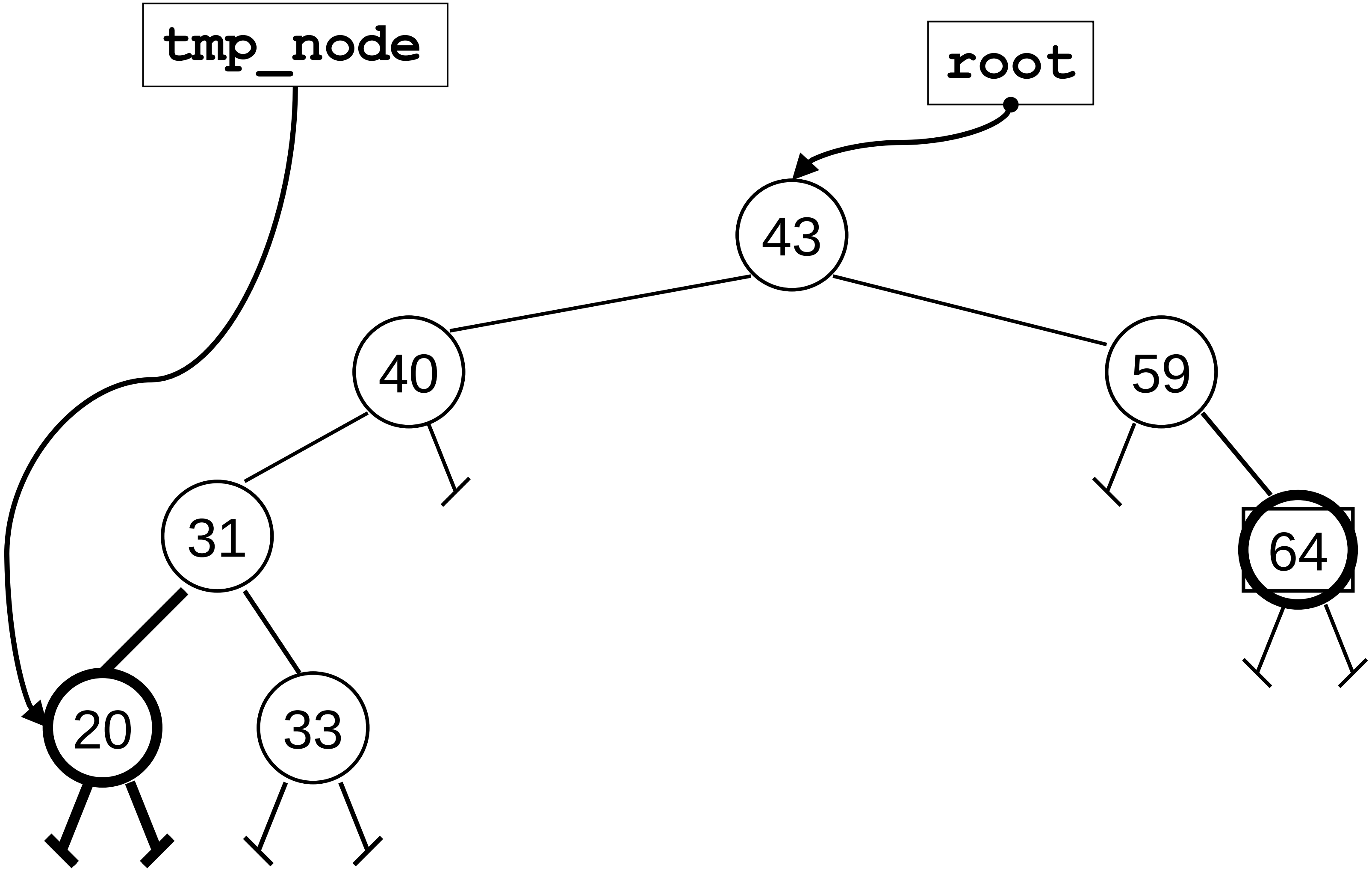
43	59	40	31	64	33	20	56	47	28	89



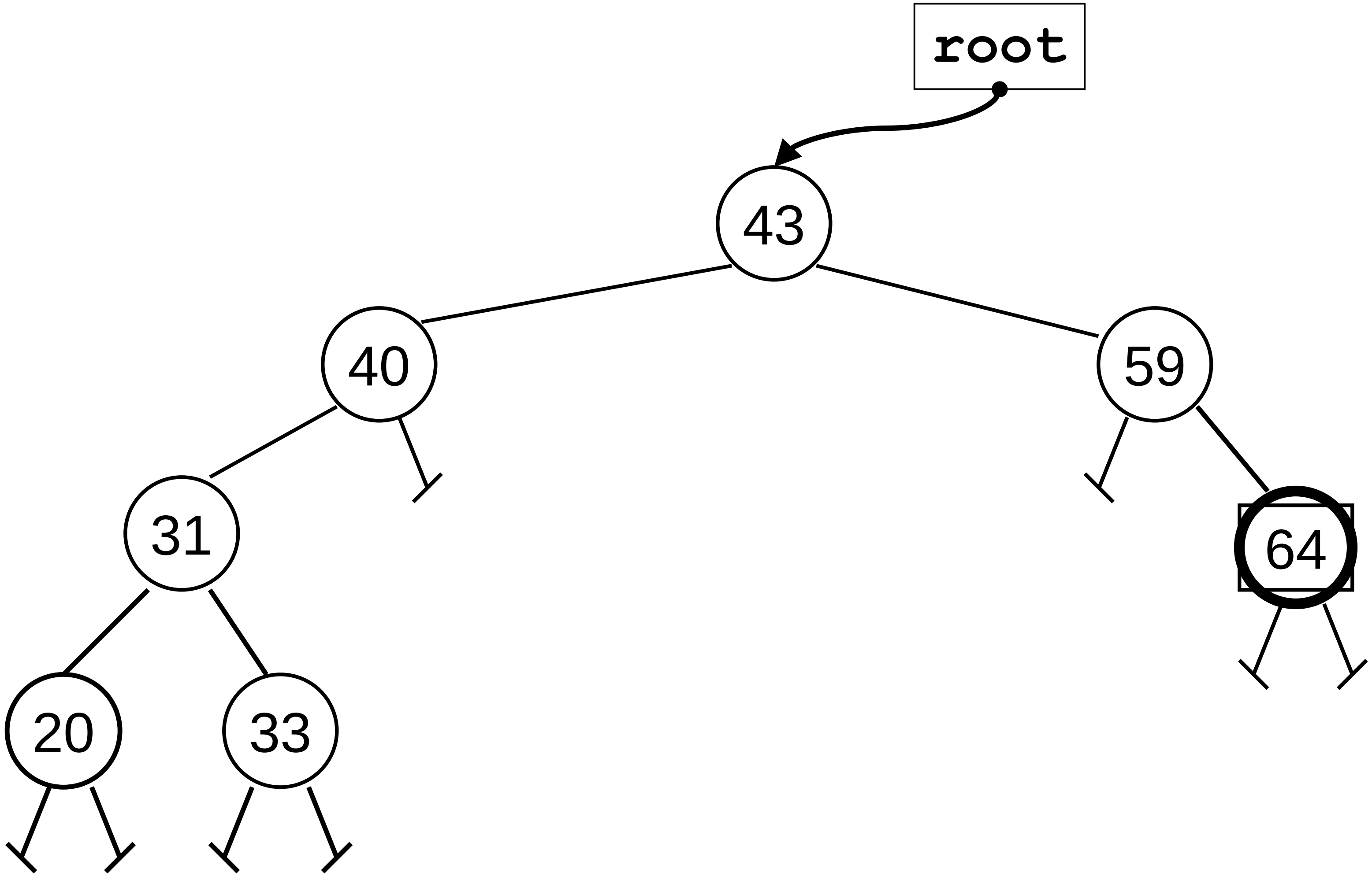
43	59	40	31	64	33	20	56	47	28	89



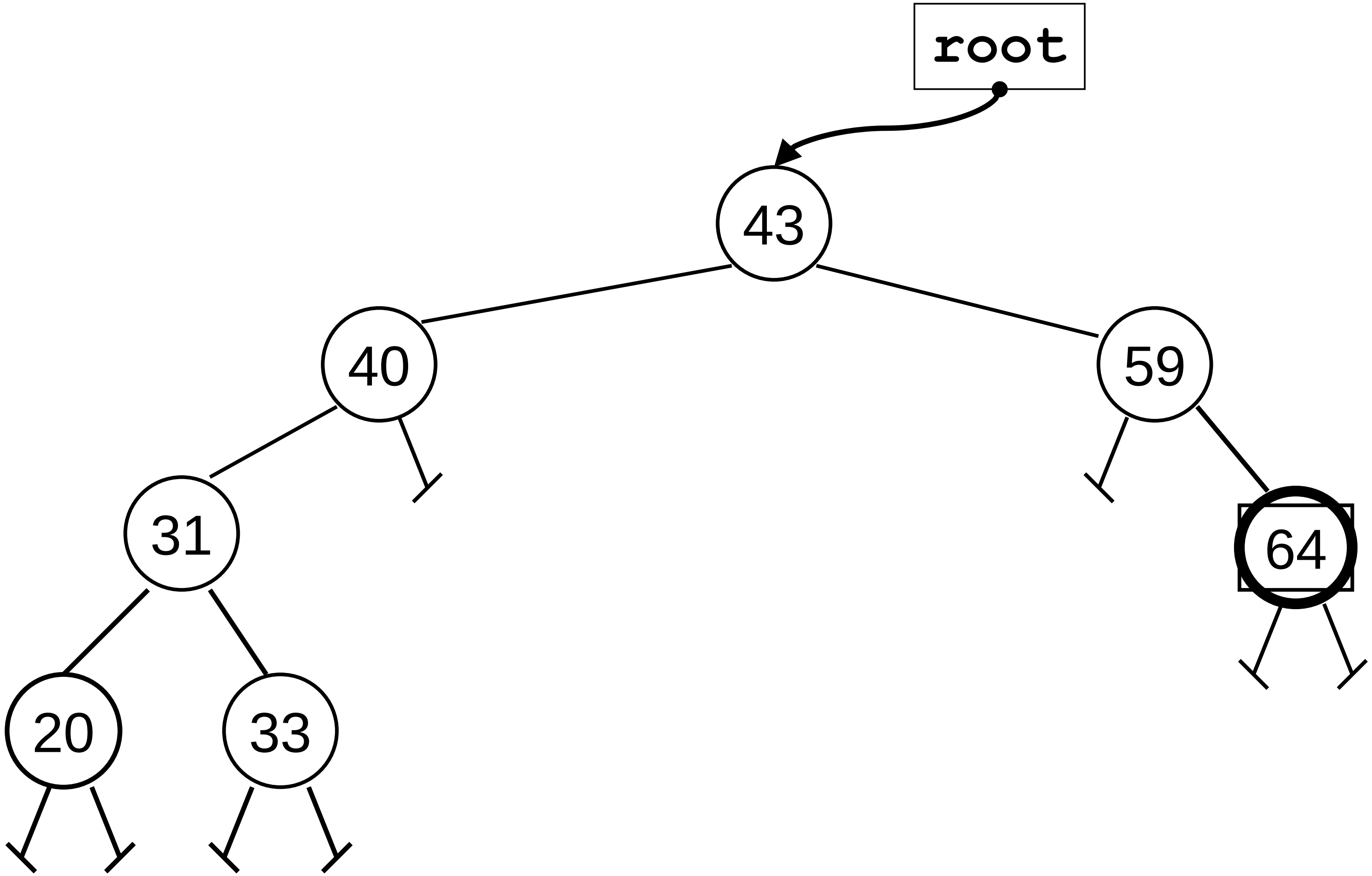
43	59	40	31	64	33	20	56	47	28	89



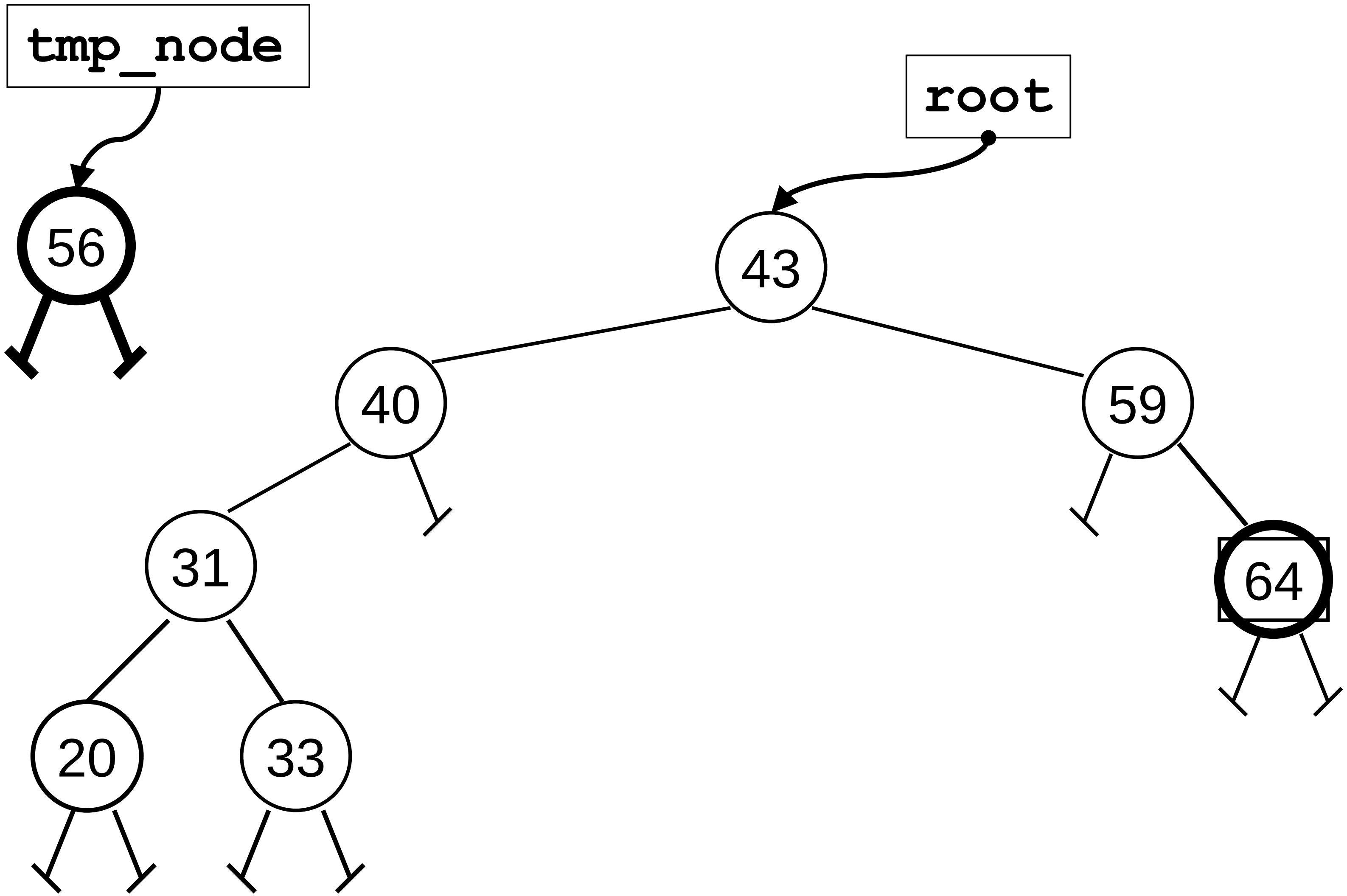
43	59	40	31	64	33	20	56	47	28	89



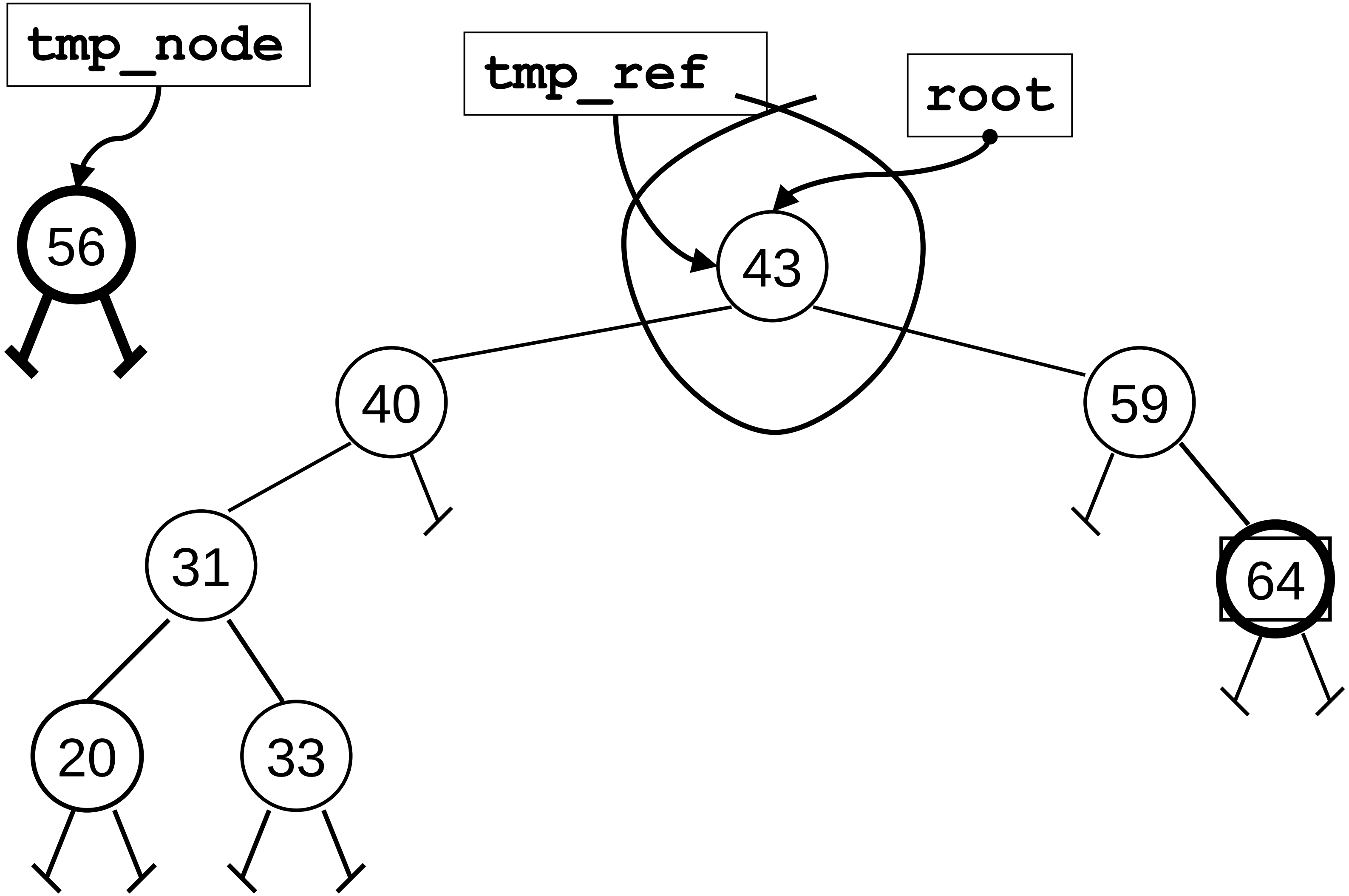
43	59	40	31	64	33	20	56	47	28	89



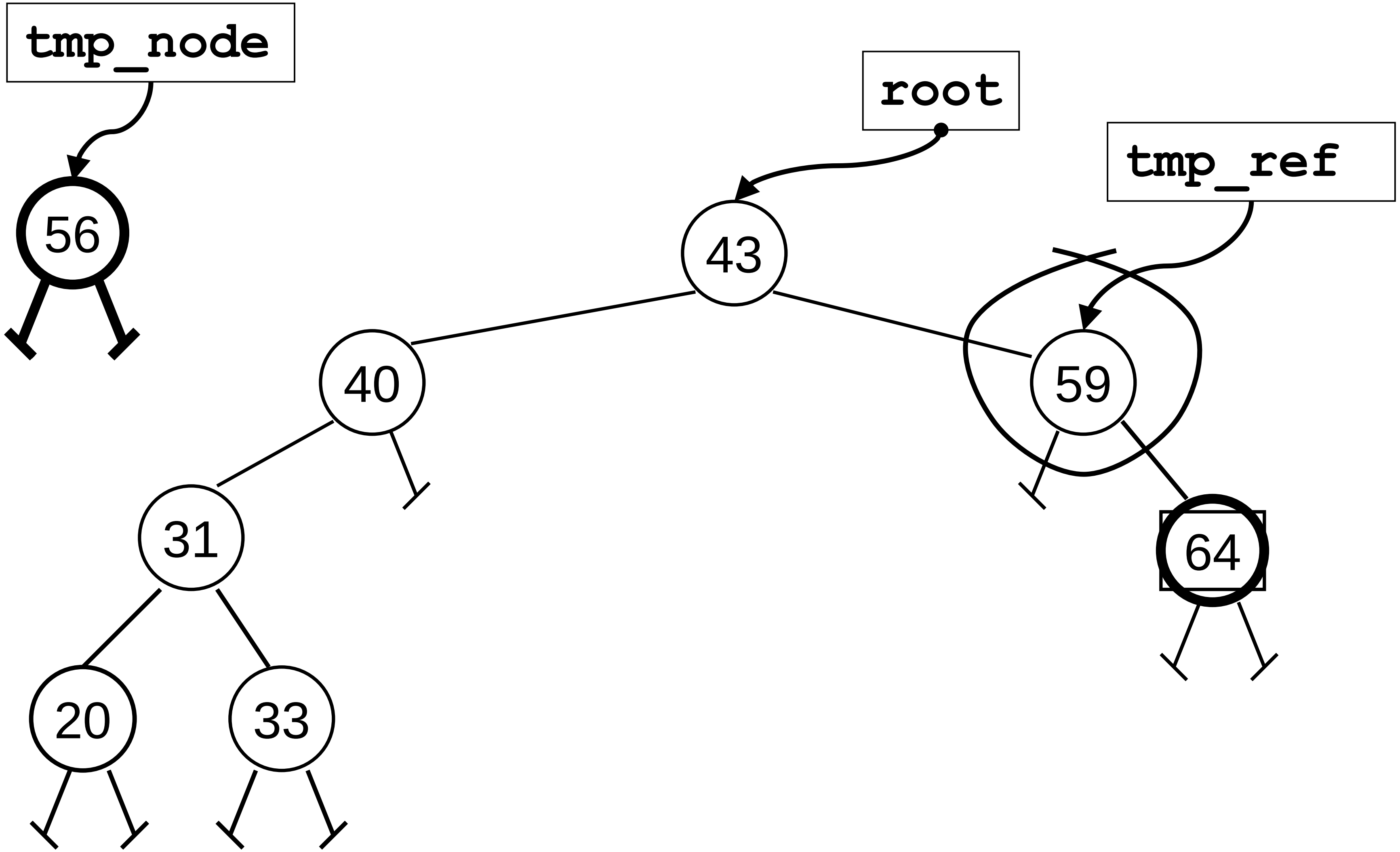
43	59	40	31	64	33	20	56	47	28	89



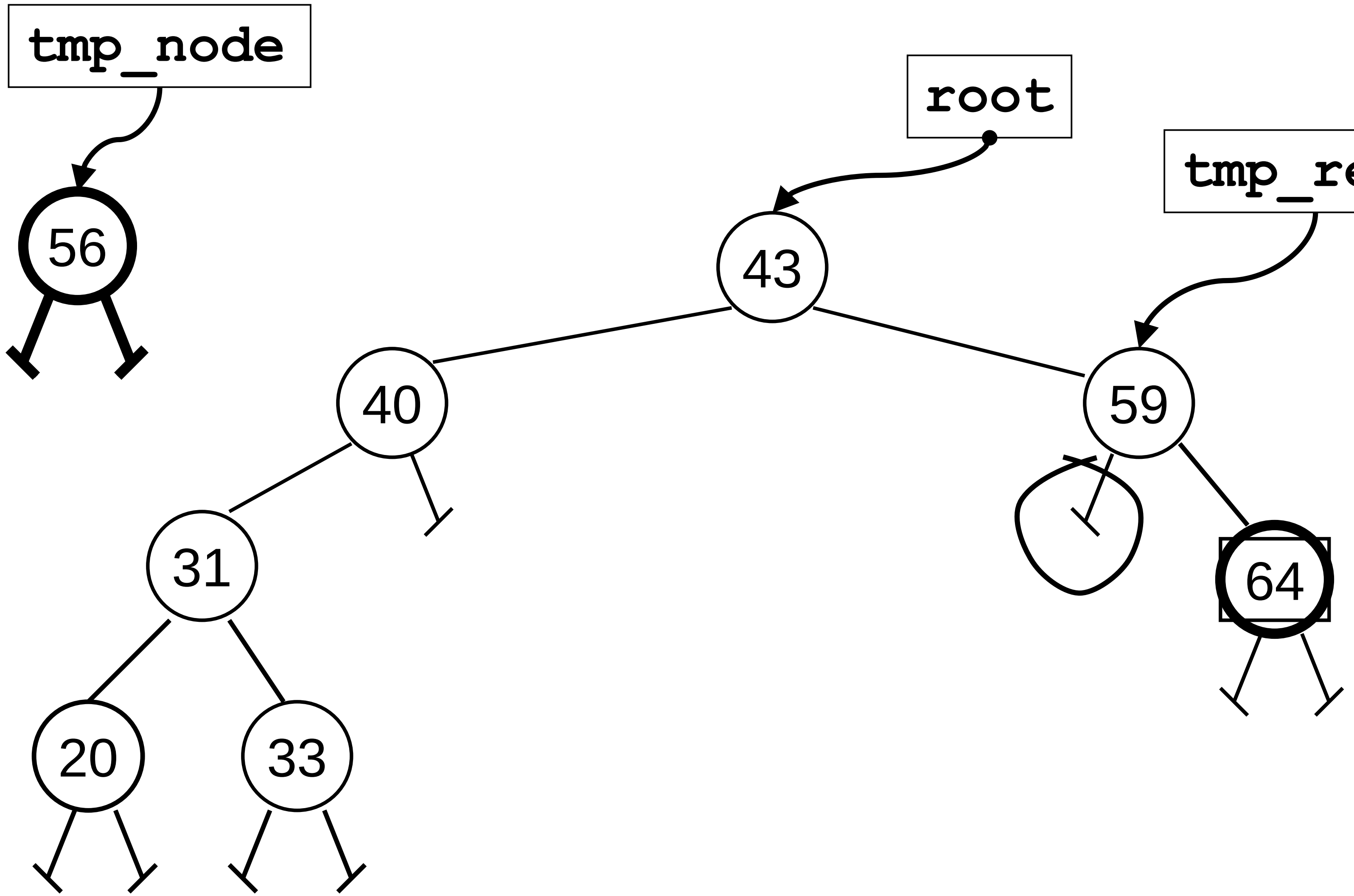
43	59	40	31	64	33	20	56	47	28	89



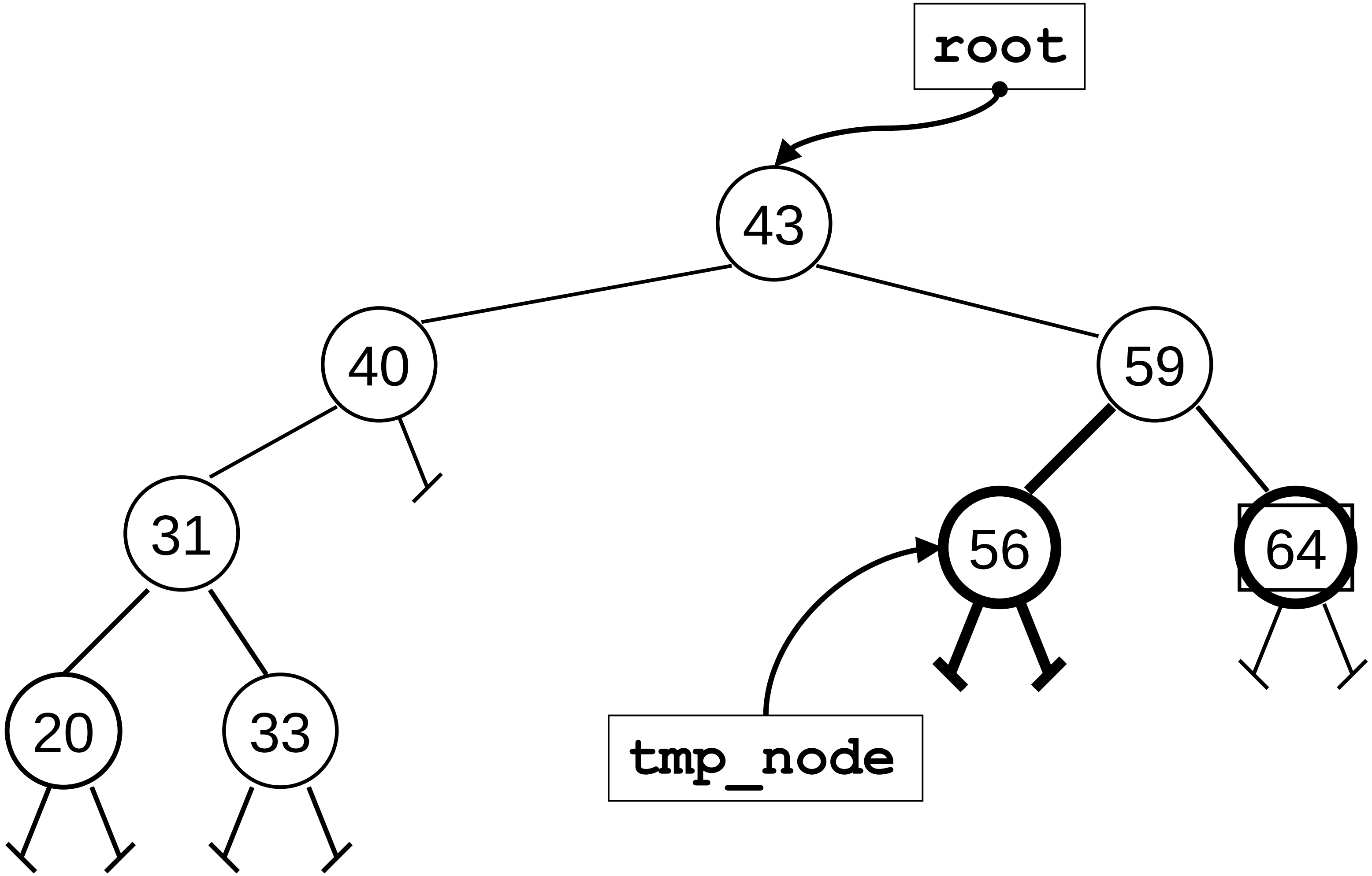
43	59	40	31	64	33	20	56	47	28	89



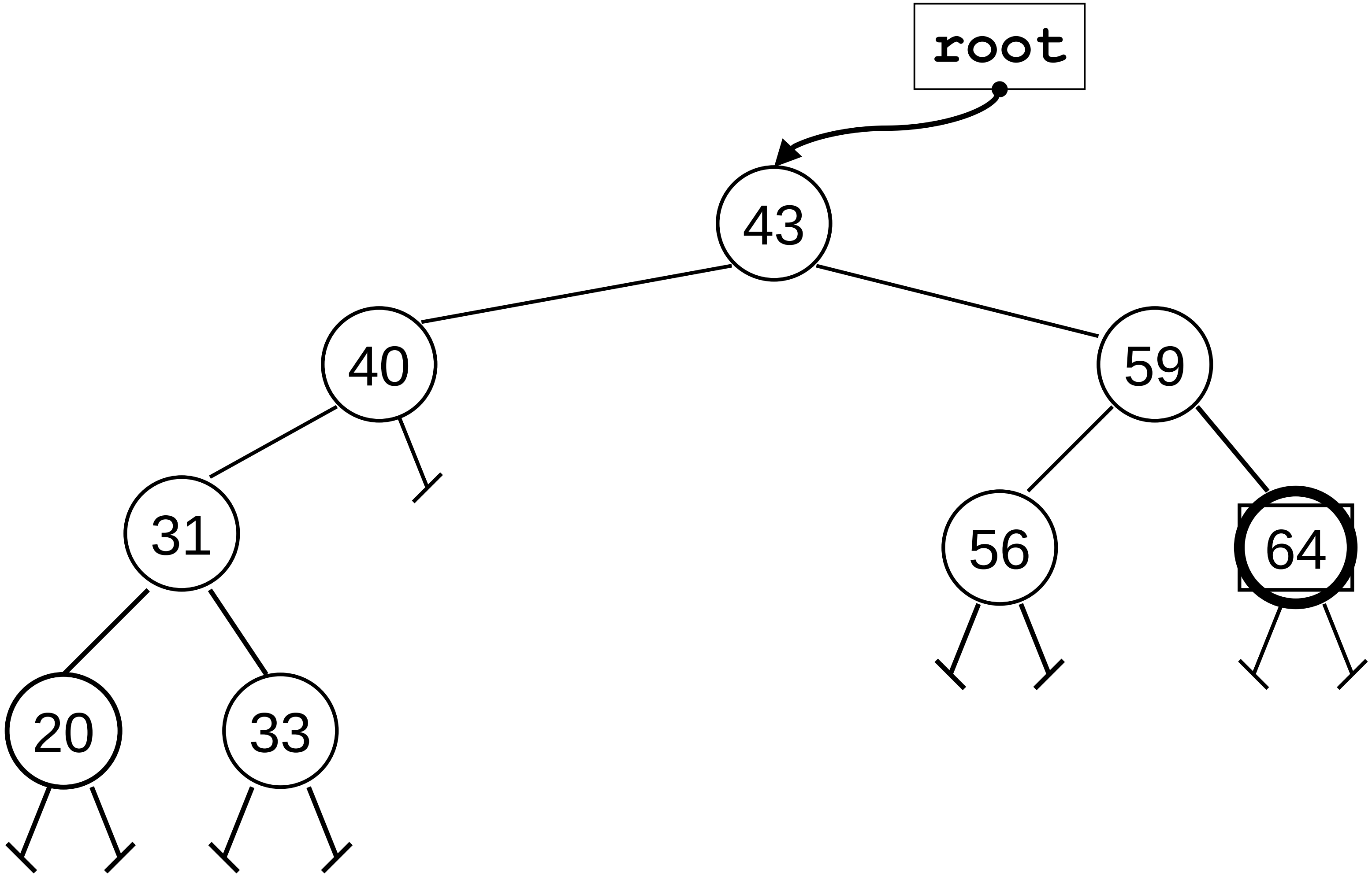
43	59	40	31	64	33	20	56	47	28	89



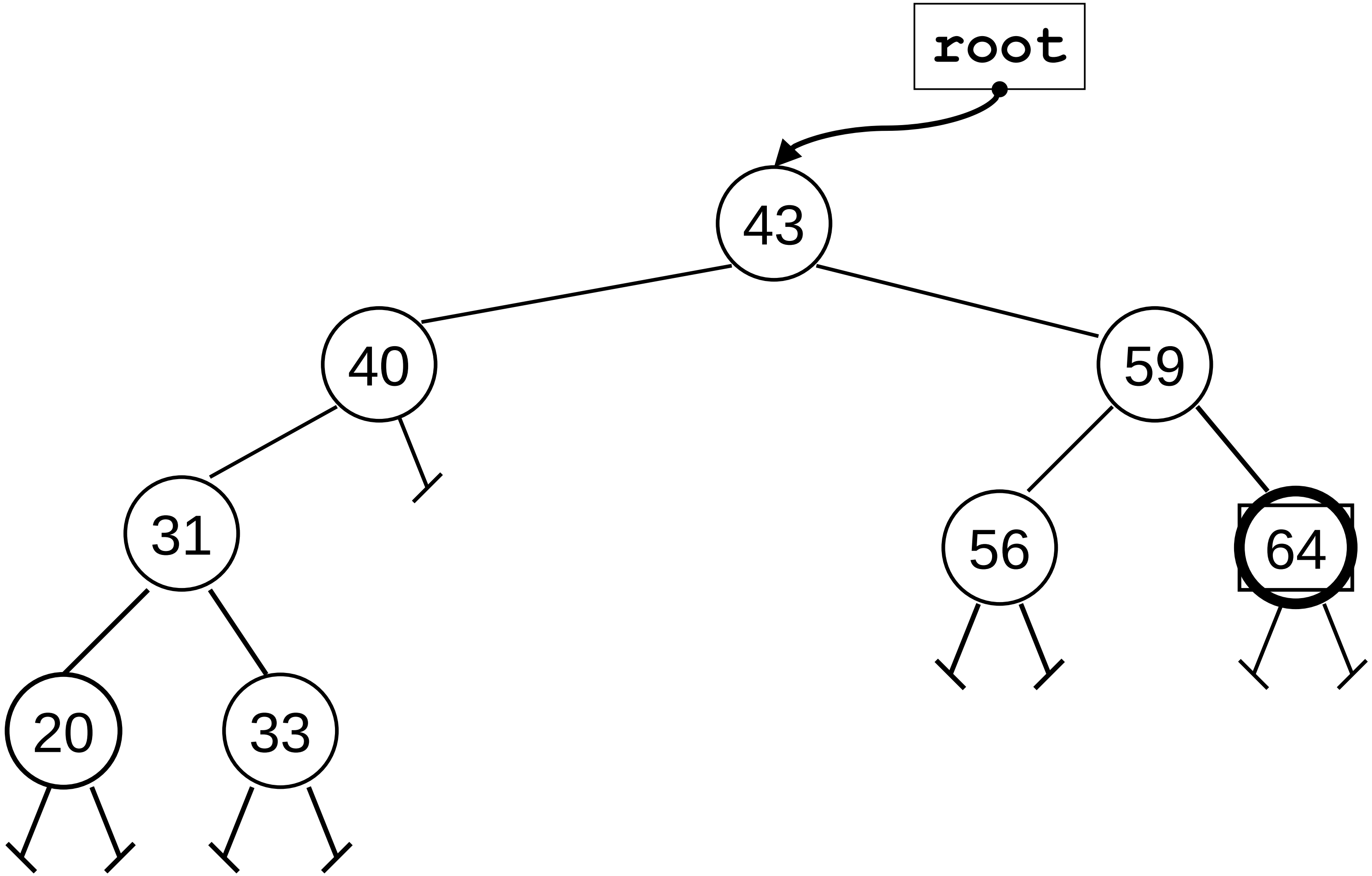
43	59	40	31	64	33	20	56	47	28	89



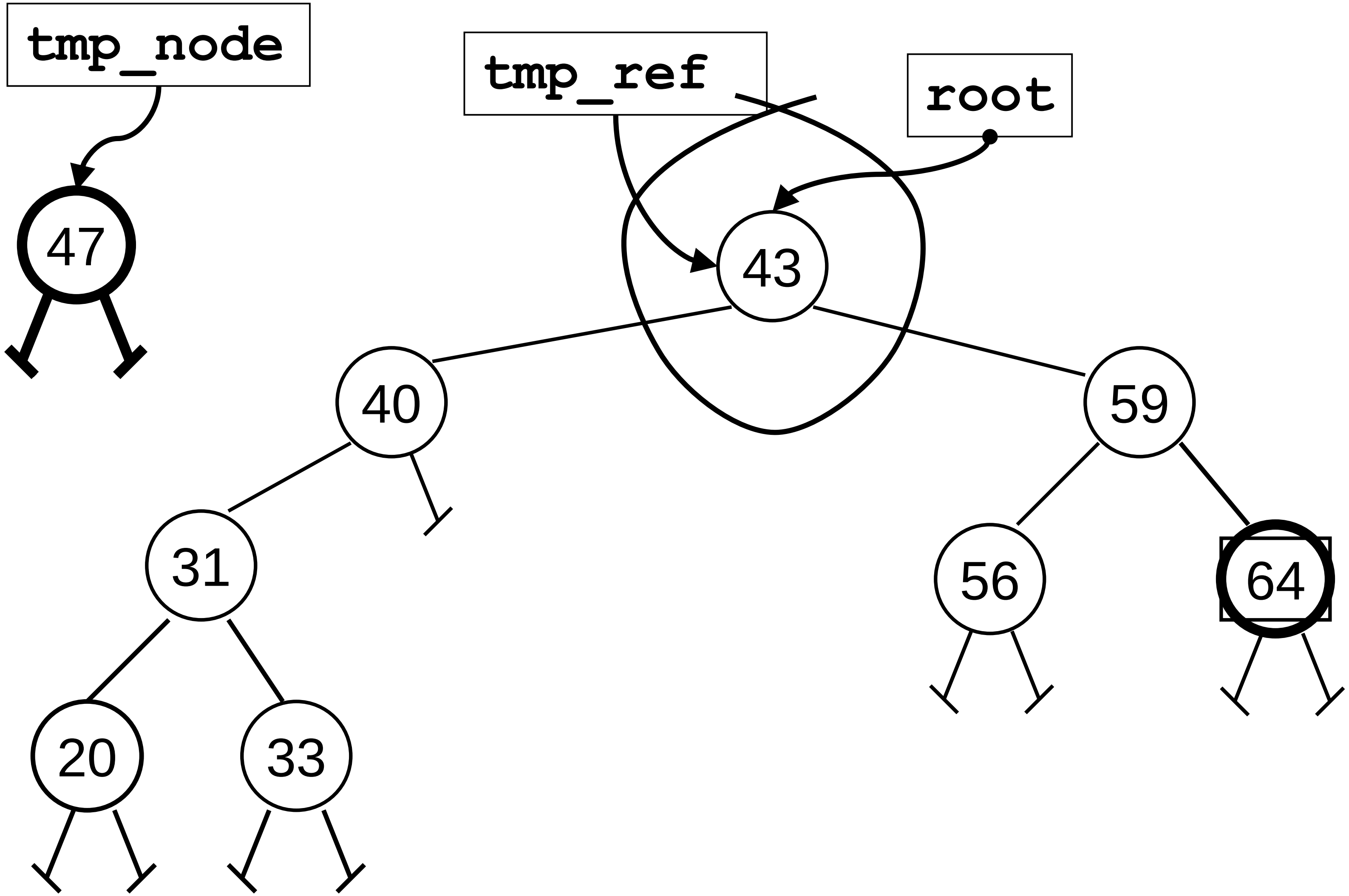
43	59	40	31	64	33	20	56	47	28	89



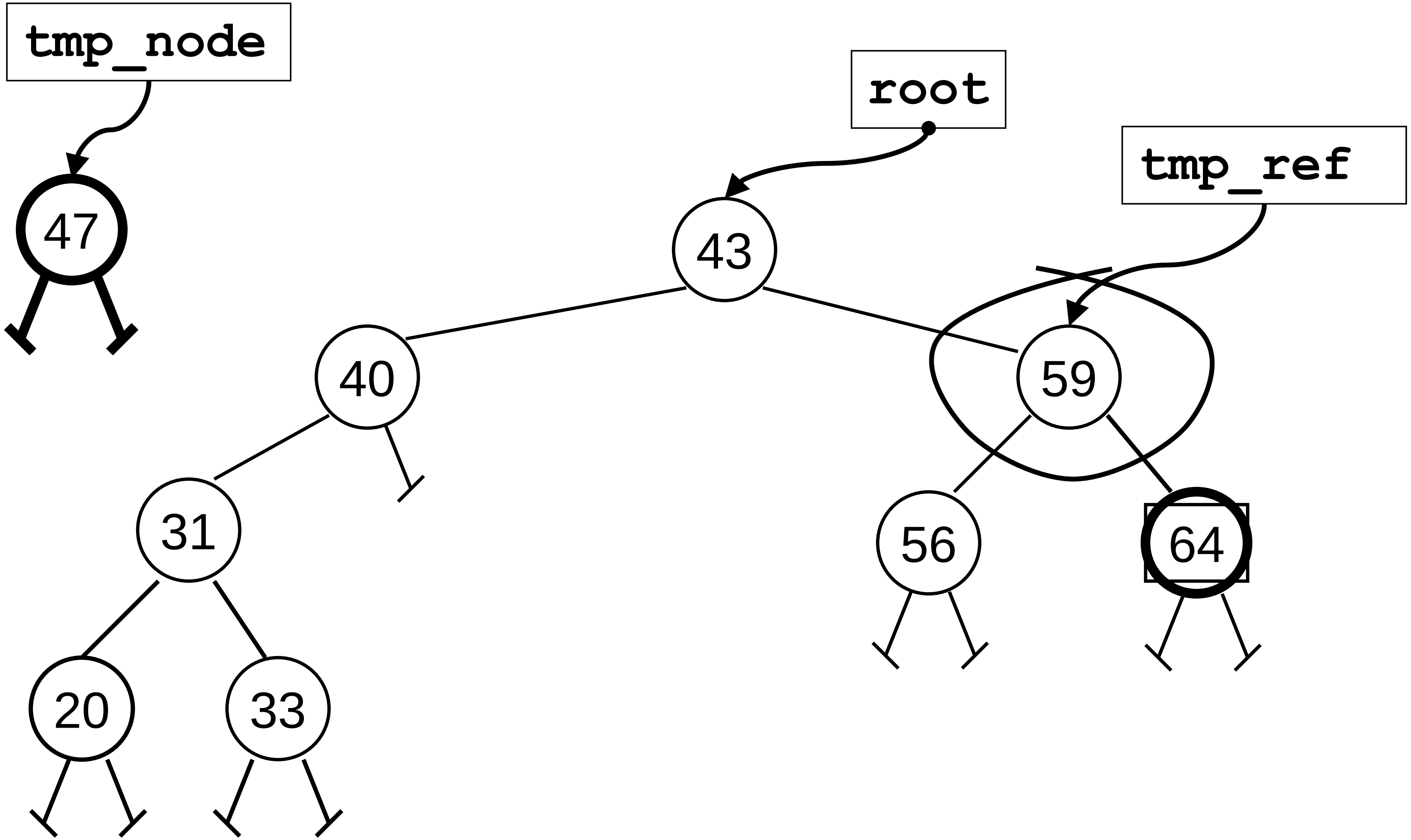
43	59	40	31	64	33	20	56	47	28	89



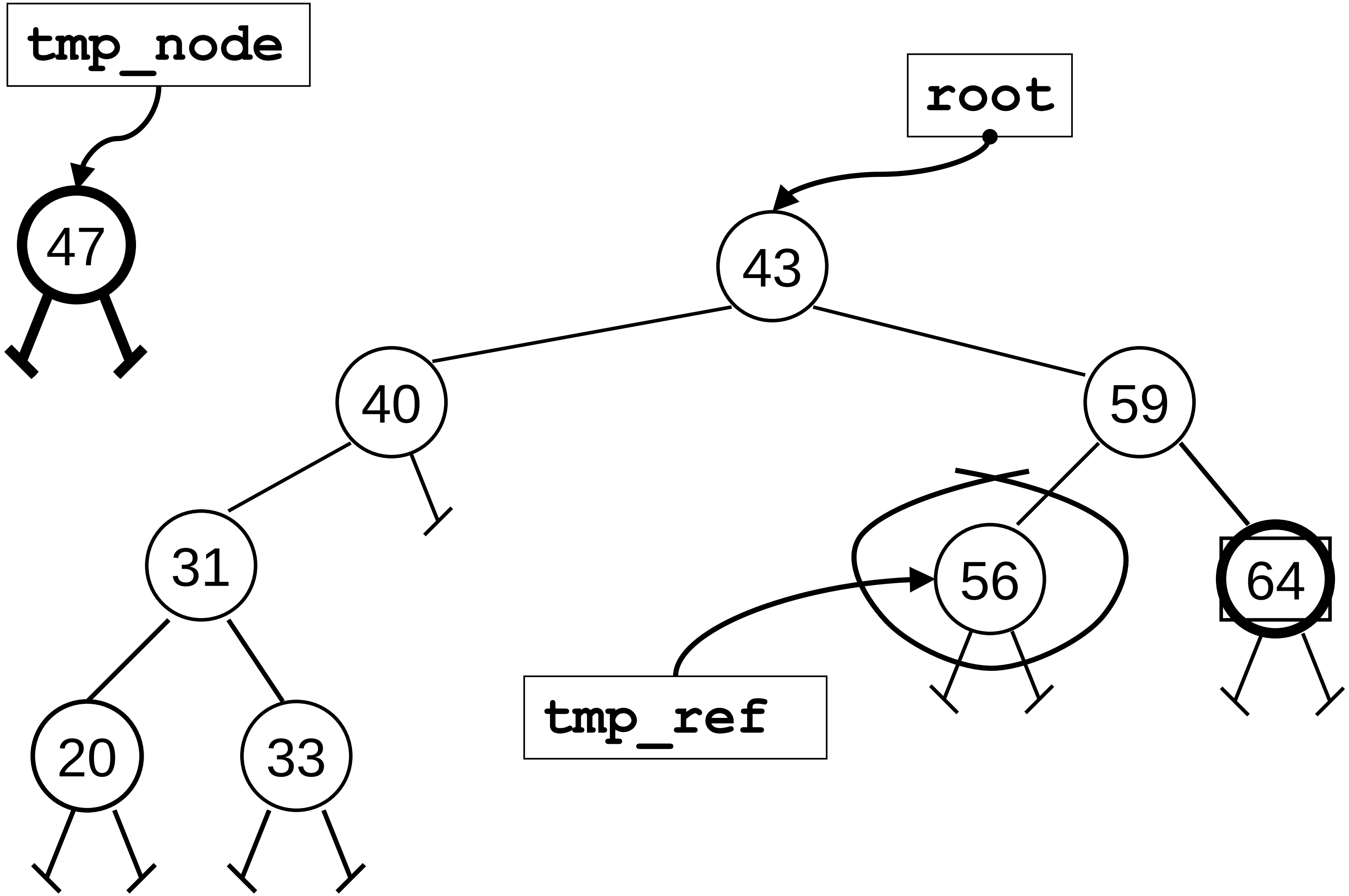
43	59	40	31	64	33	20	56	47	28	89



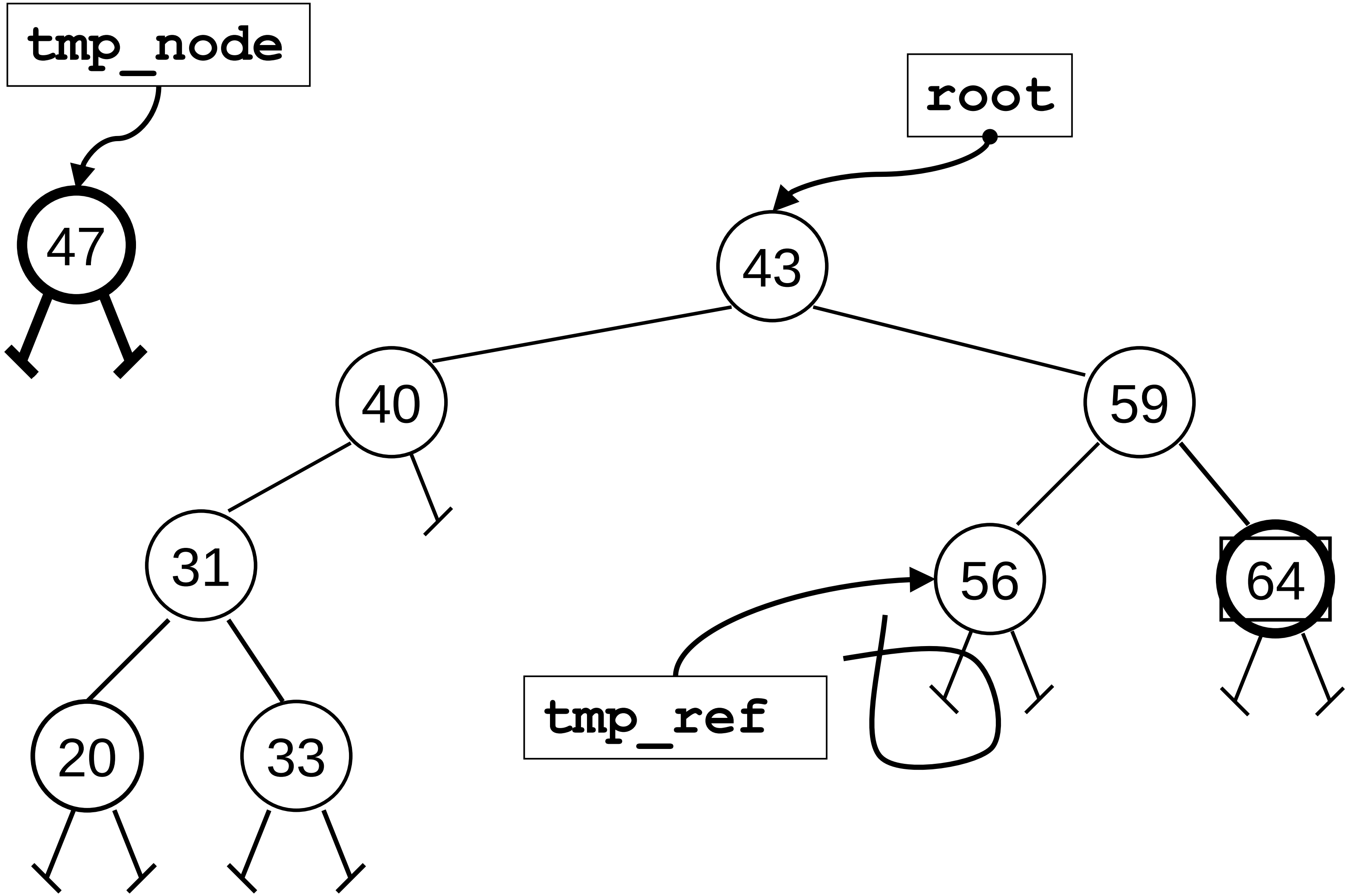
43	59	40	31	64	33	20	56	47	28	89



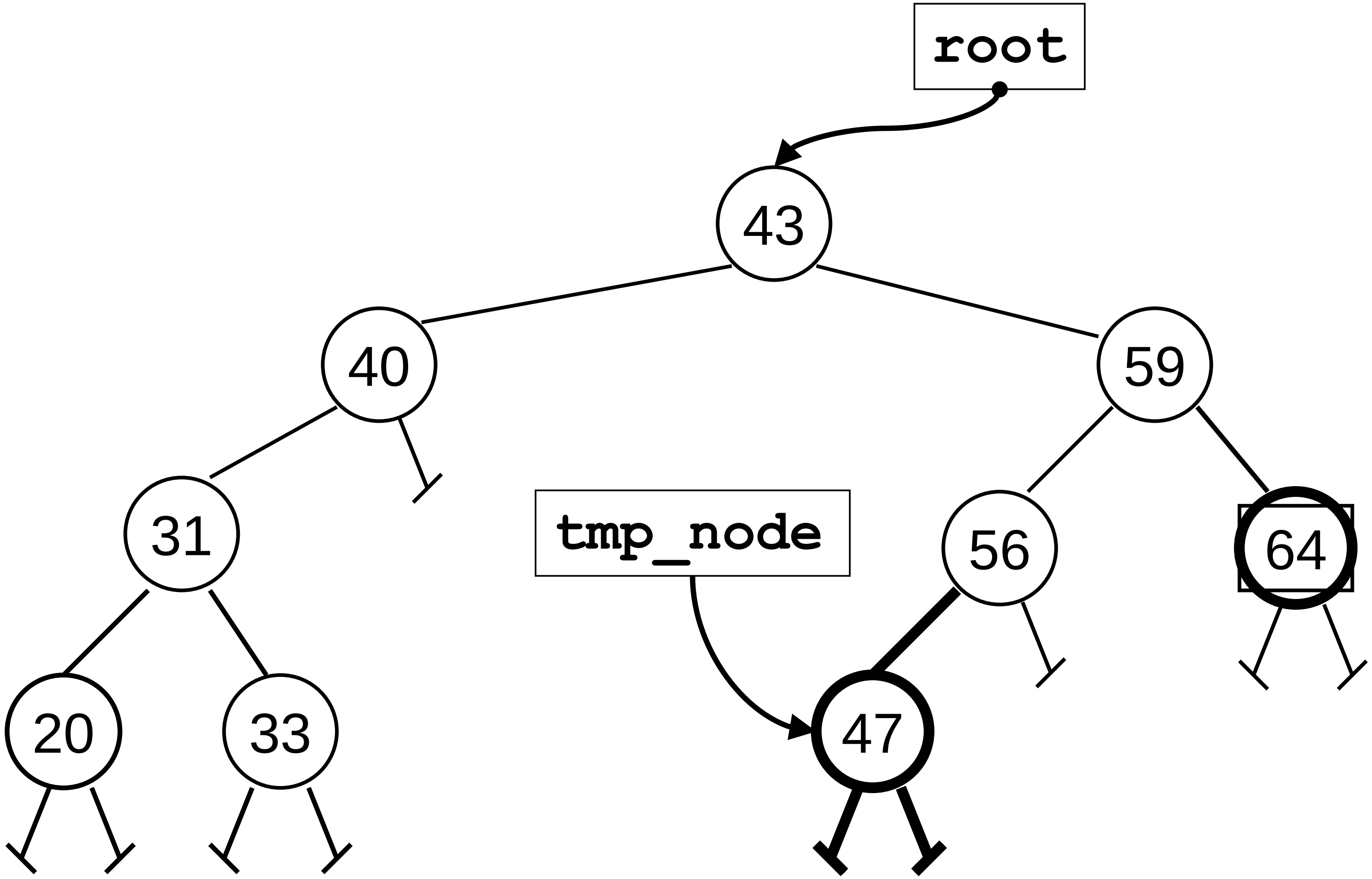
43	59	40	31	64	33	20	56	47	28	89



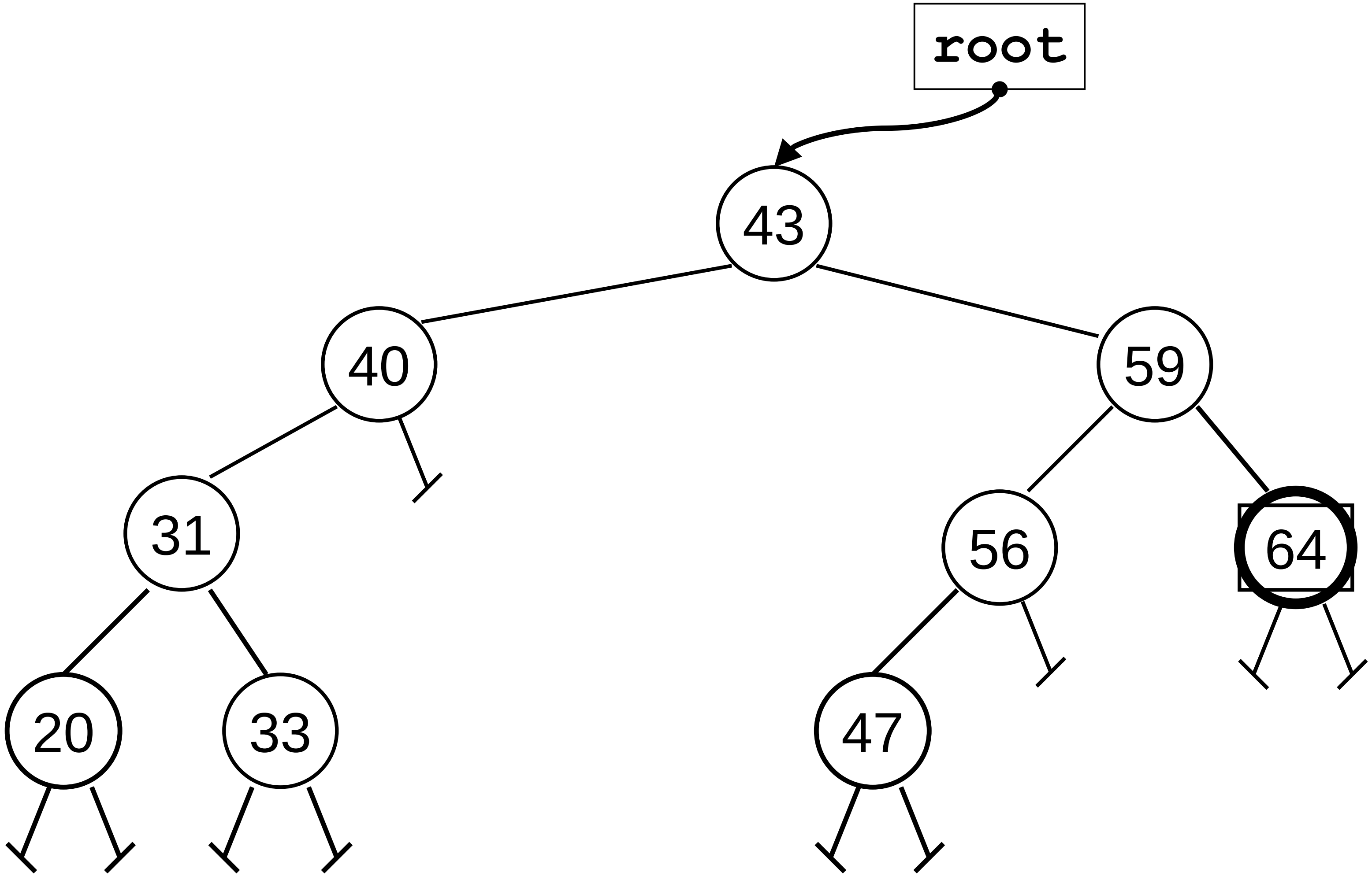
43	59	40	31	64	33	20	56	47	28	89



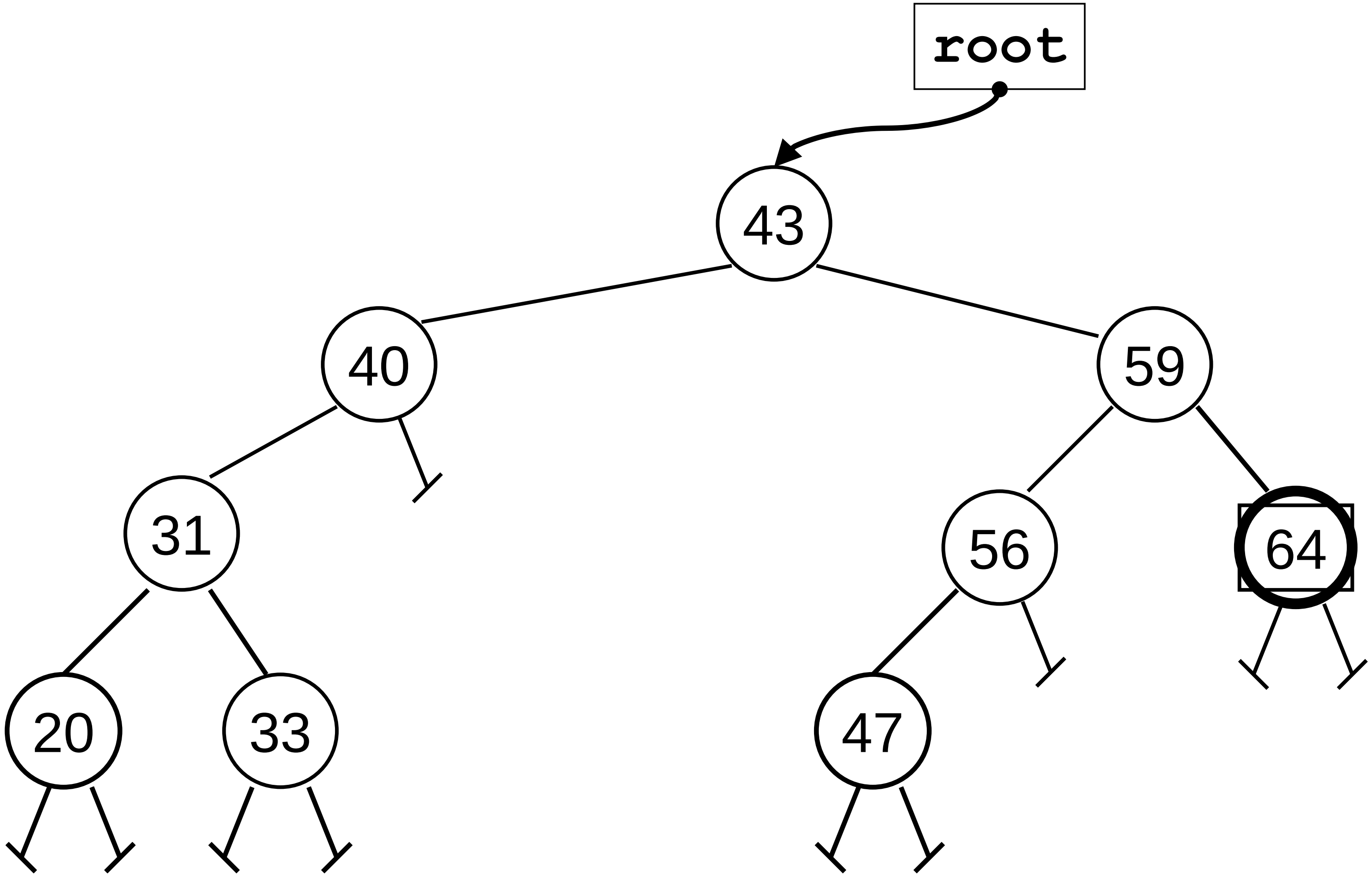
43	59	40	31	64	33	20	56	47	28	89



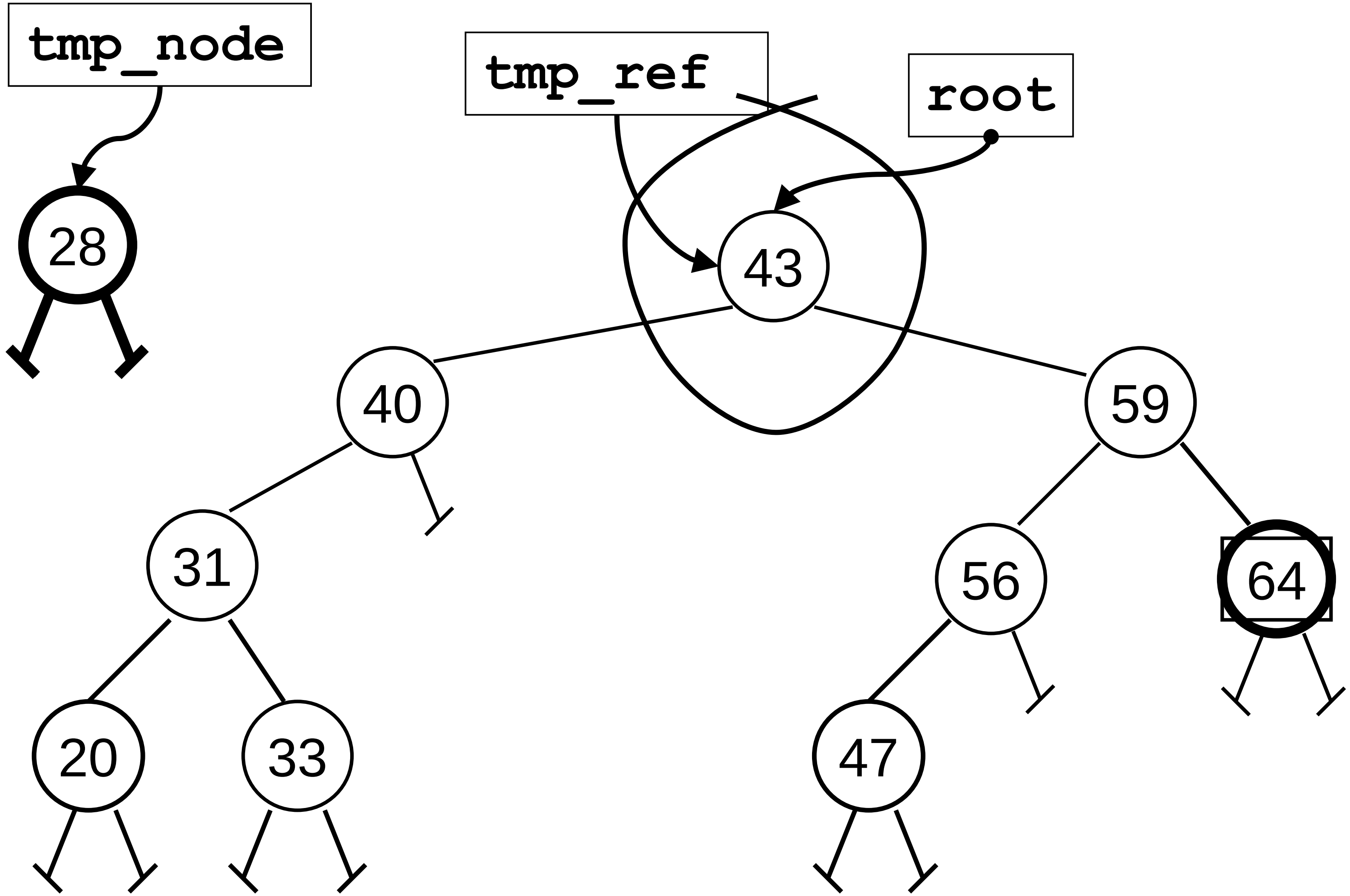
43	59	40	31	64	33	20	56	47	28	89



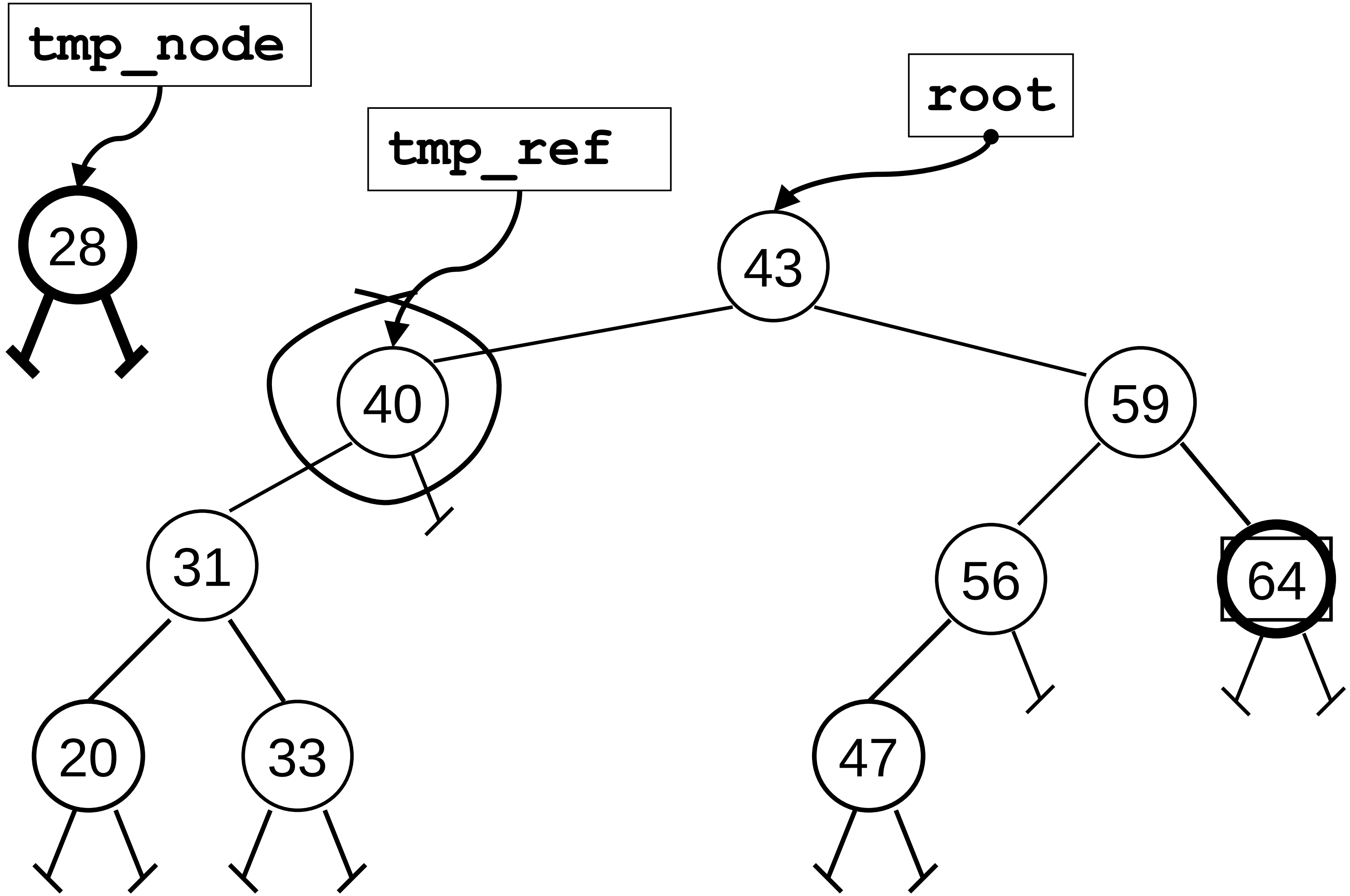
43	59	40	31	64	33	20	56	47	28	89



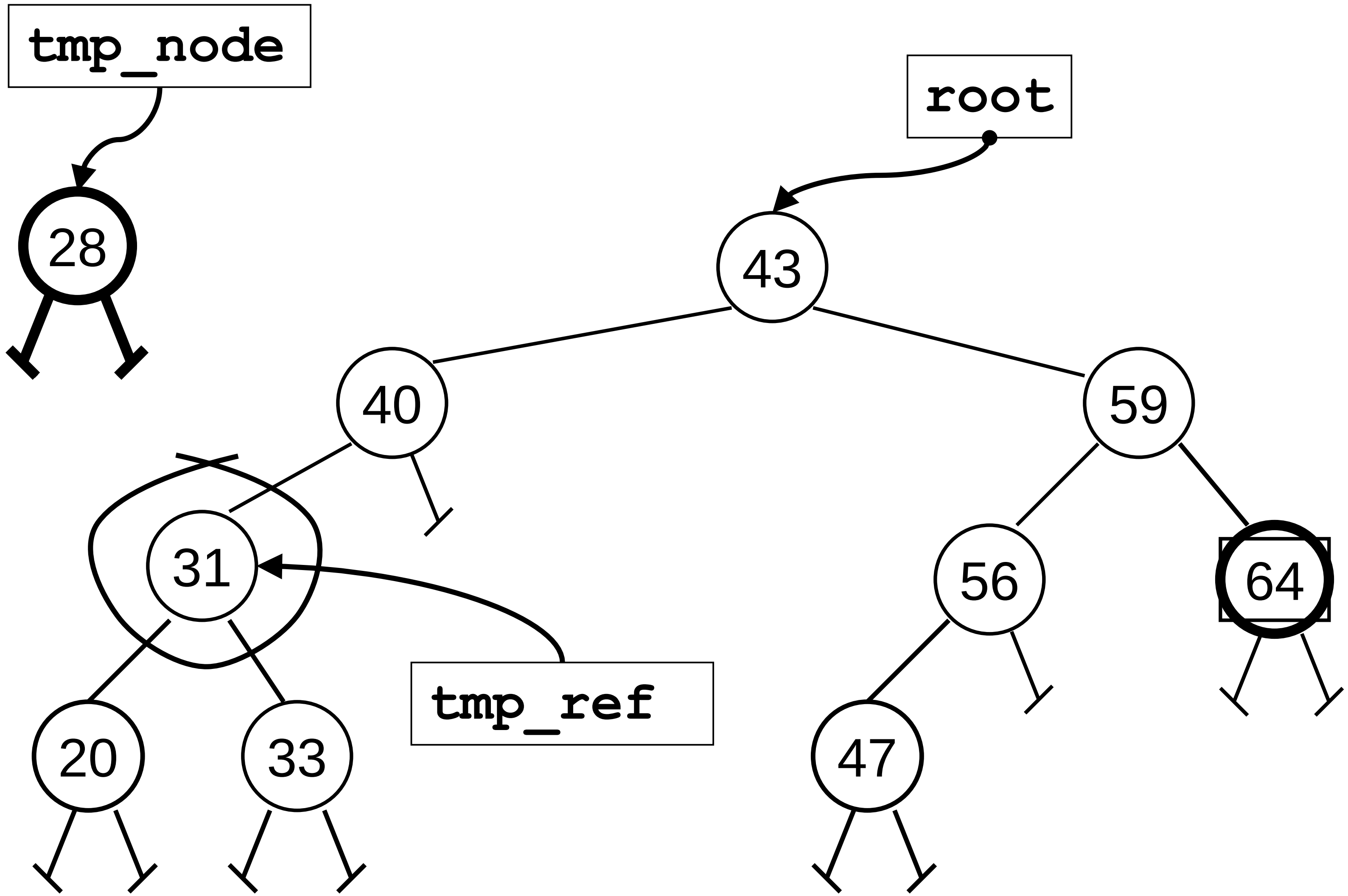
43	59	40	31	64	33	20	56	47	28	89



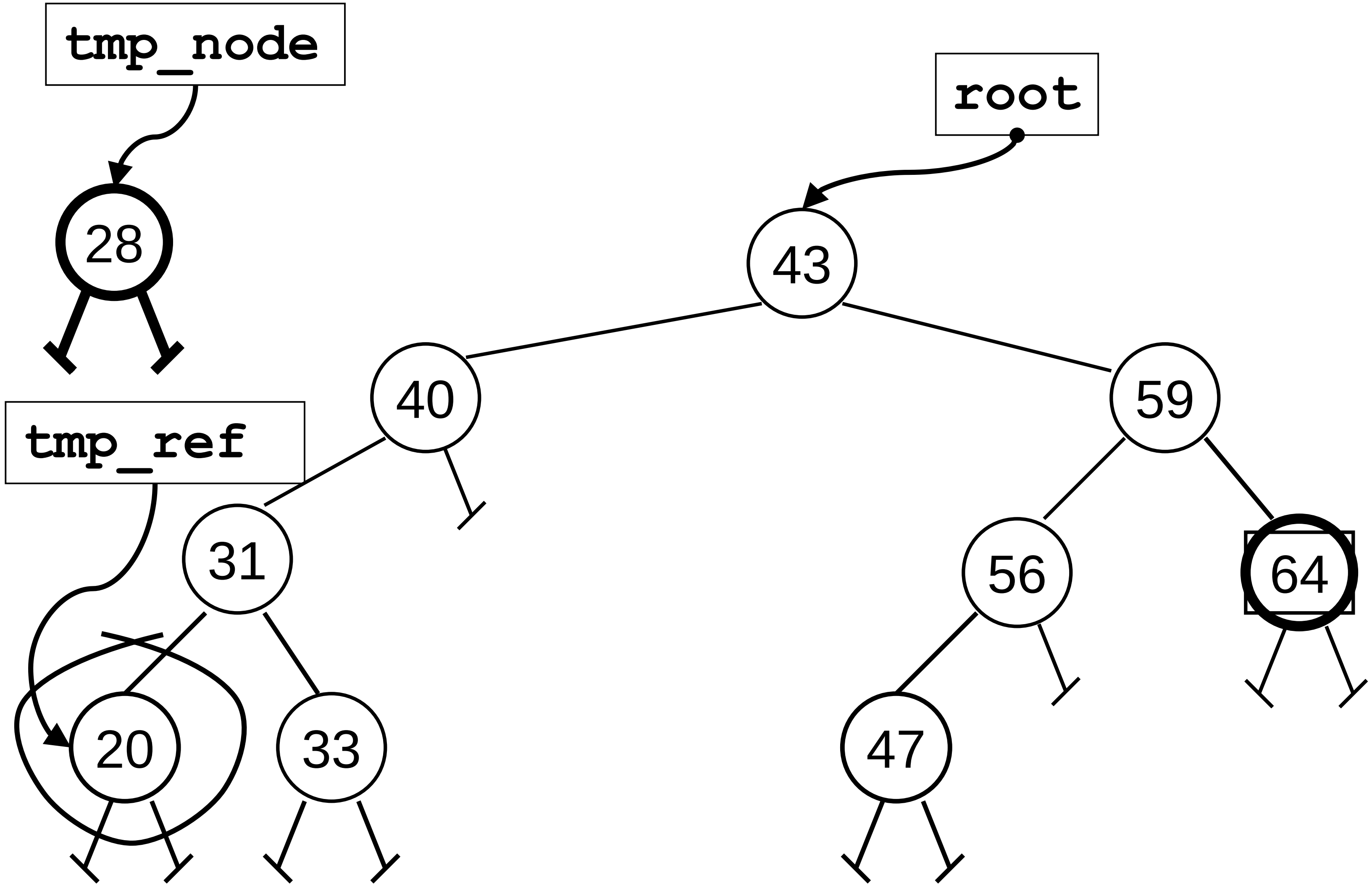
43	59	40	31	64	33	20	56	47	28	89



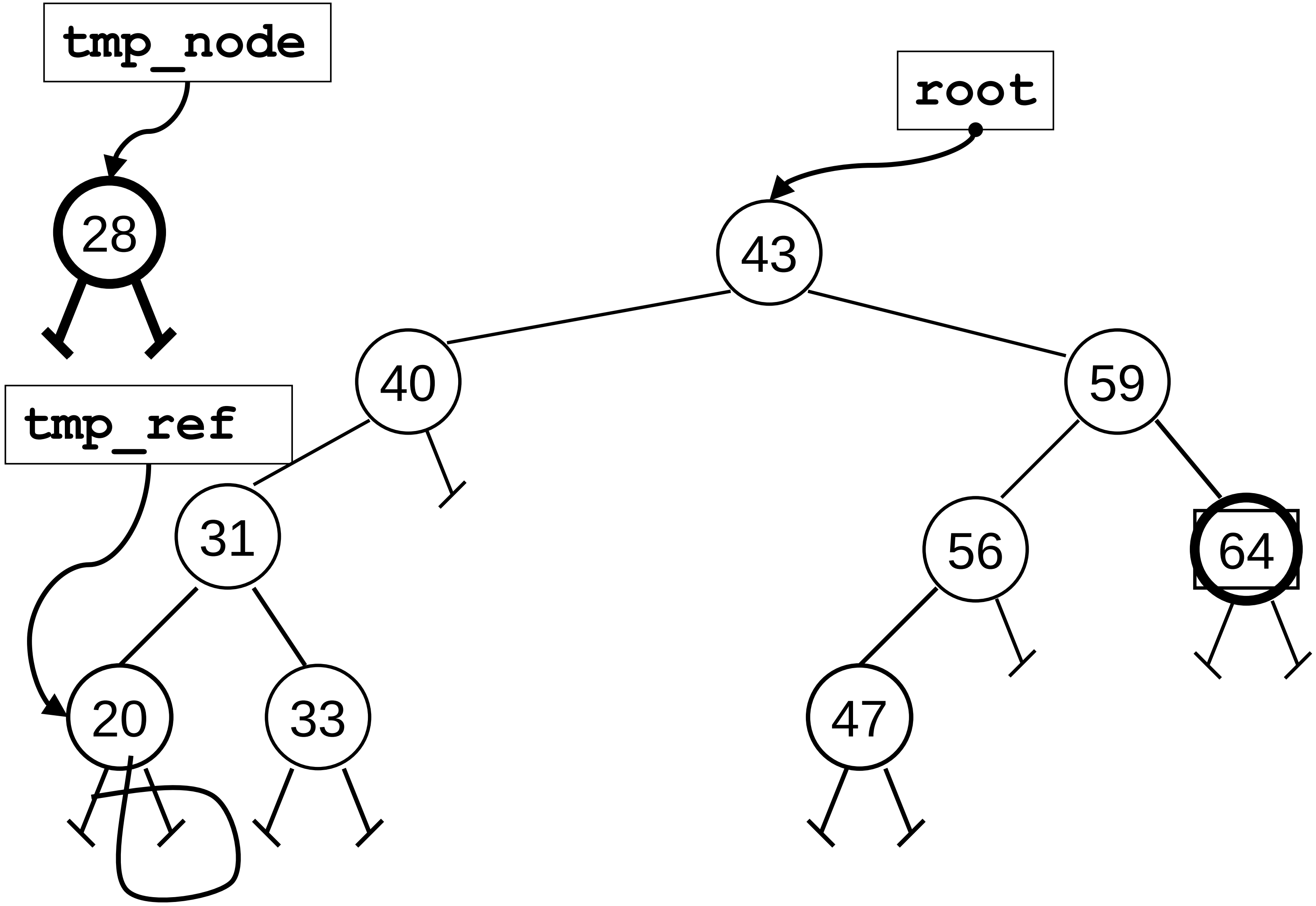
43	59	40	31	64	33	20	56	47	28	89



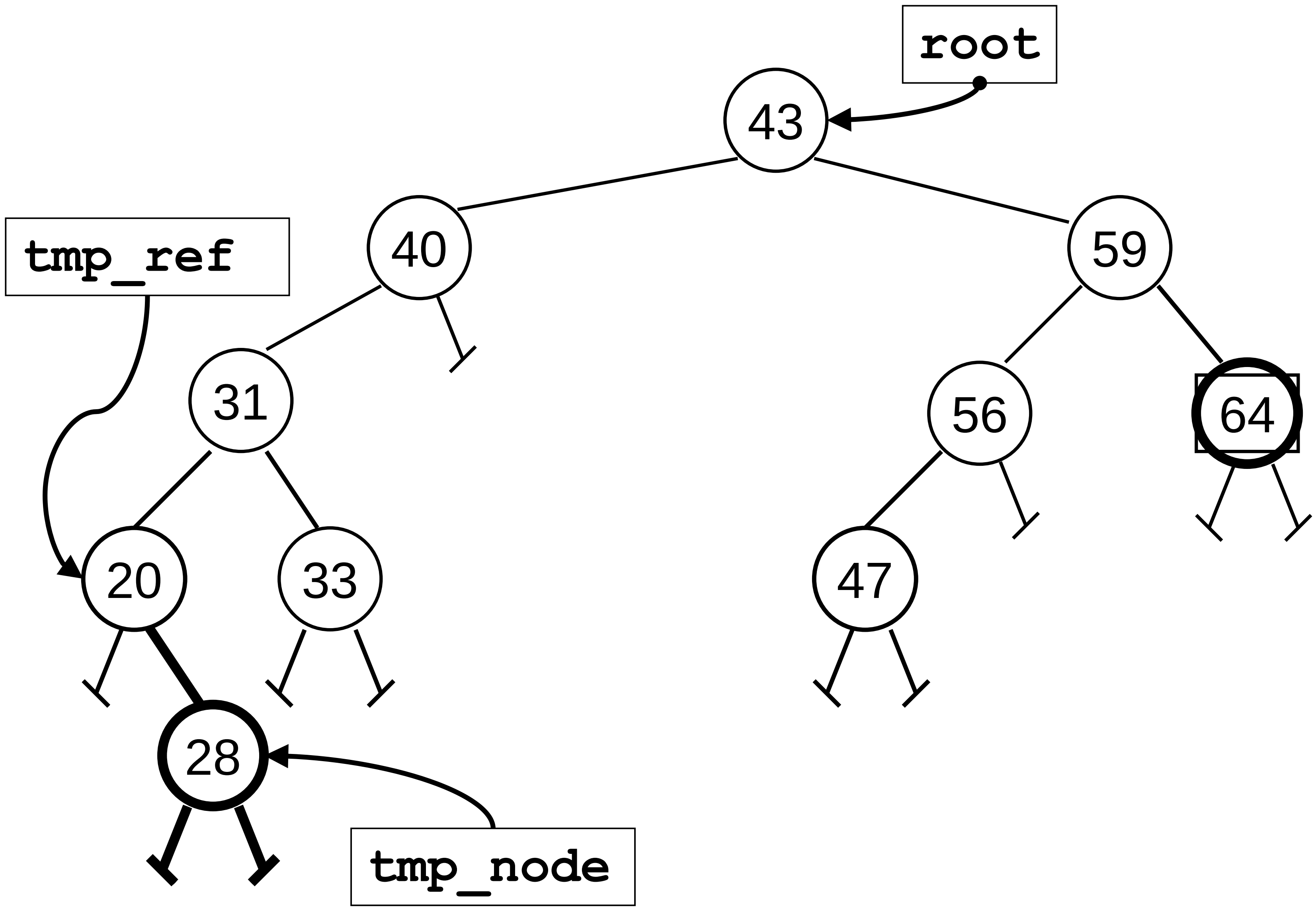
43	59	40	31	64	33	20	56	47	28	89



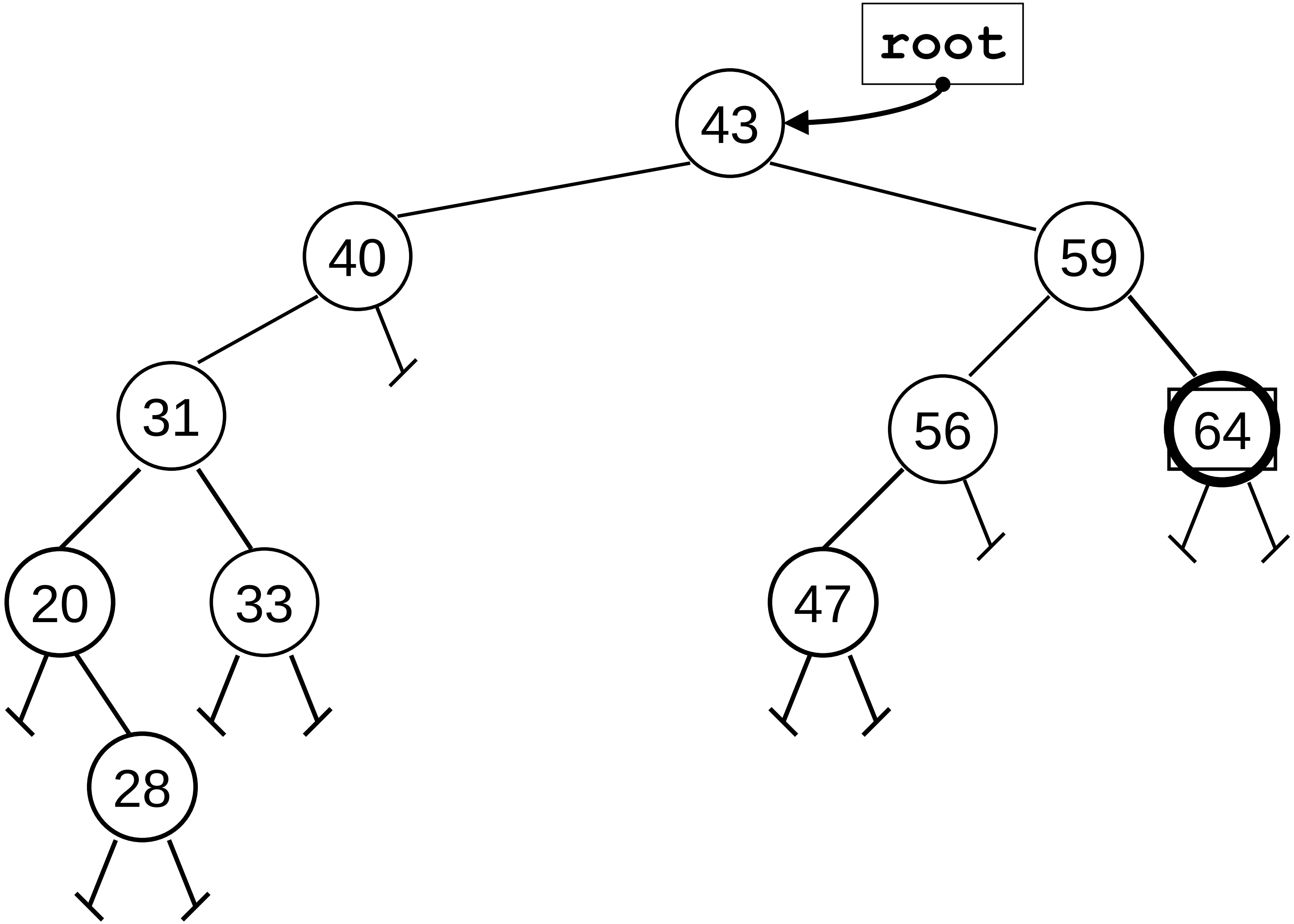
43	59	40	31	64	33	20	56	47	28	89



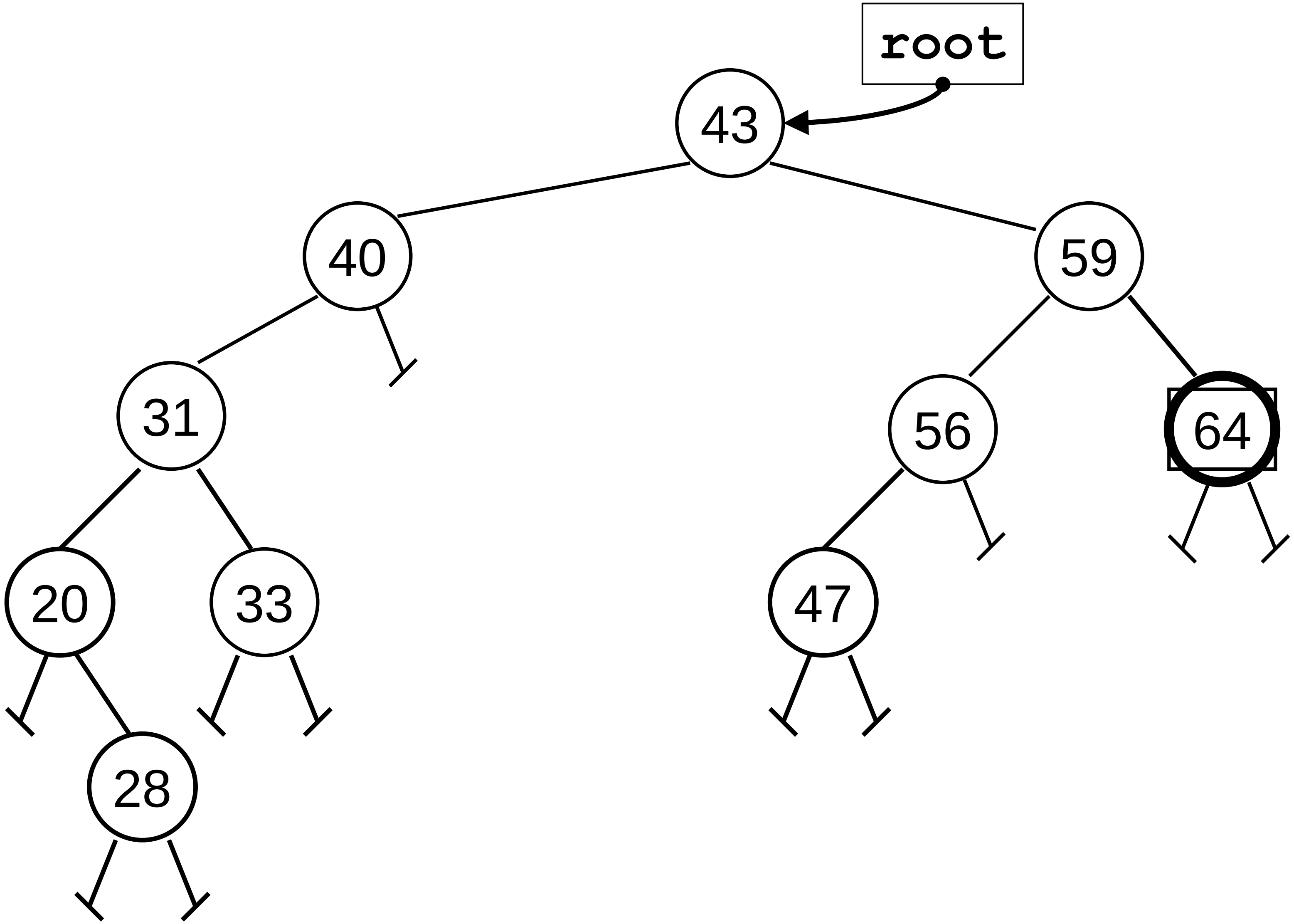
43	59	40	31	64	33	20	56	47	28	89



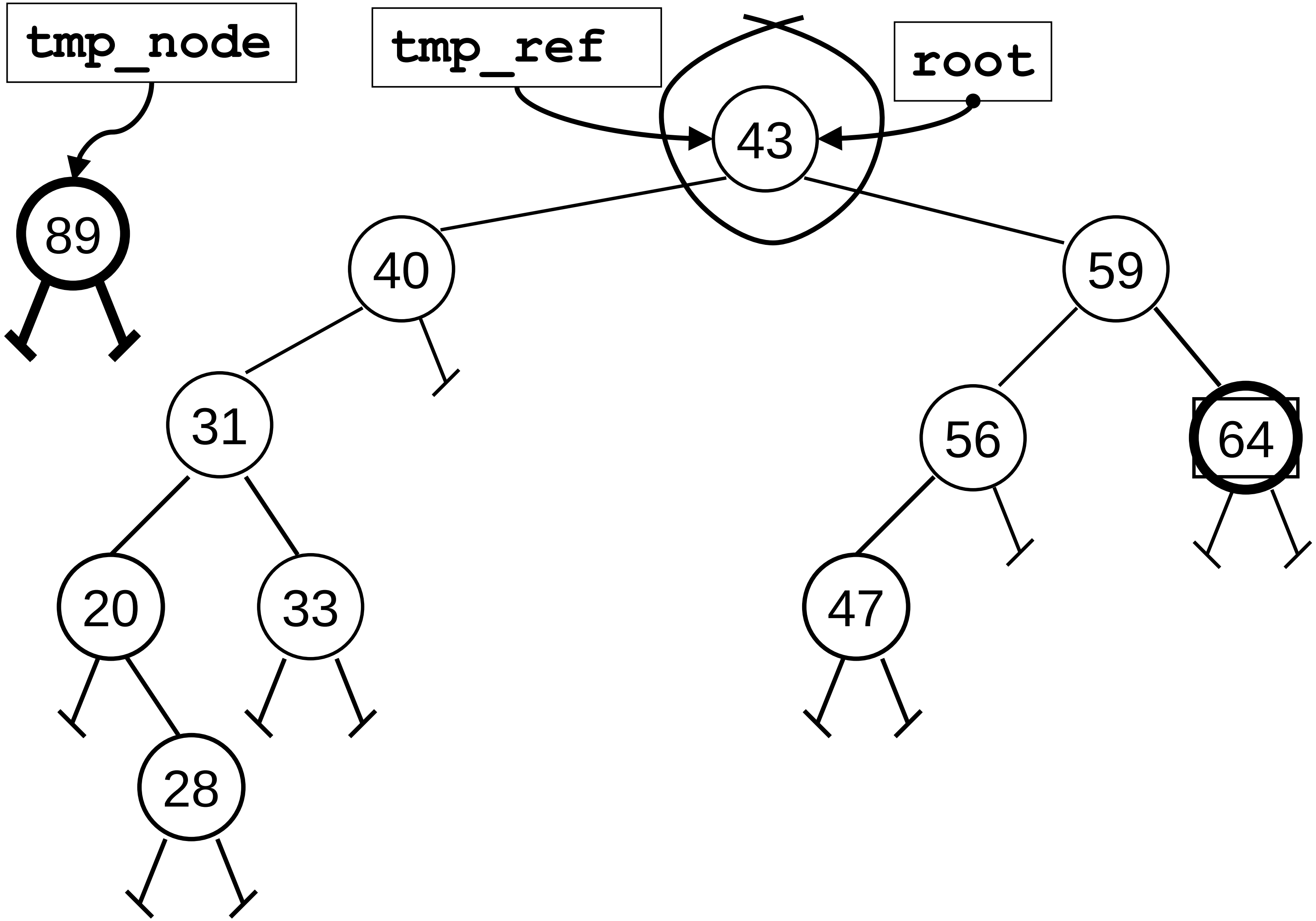
43	59	40	31	64	33	20	56	47	28	89



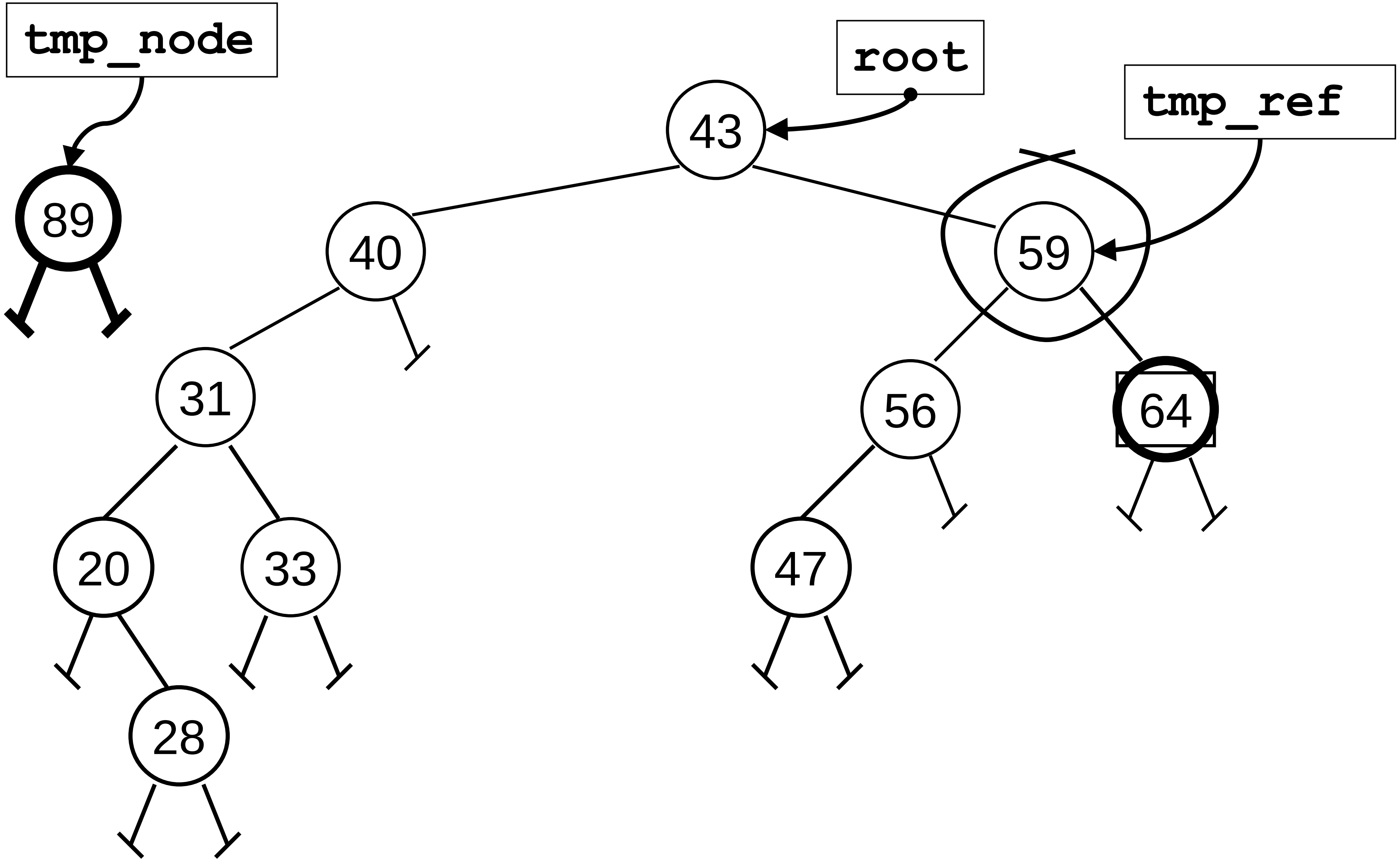
43	59	40	31	64	33	20	56	47	28	89



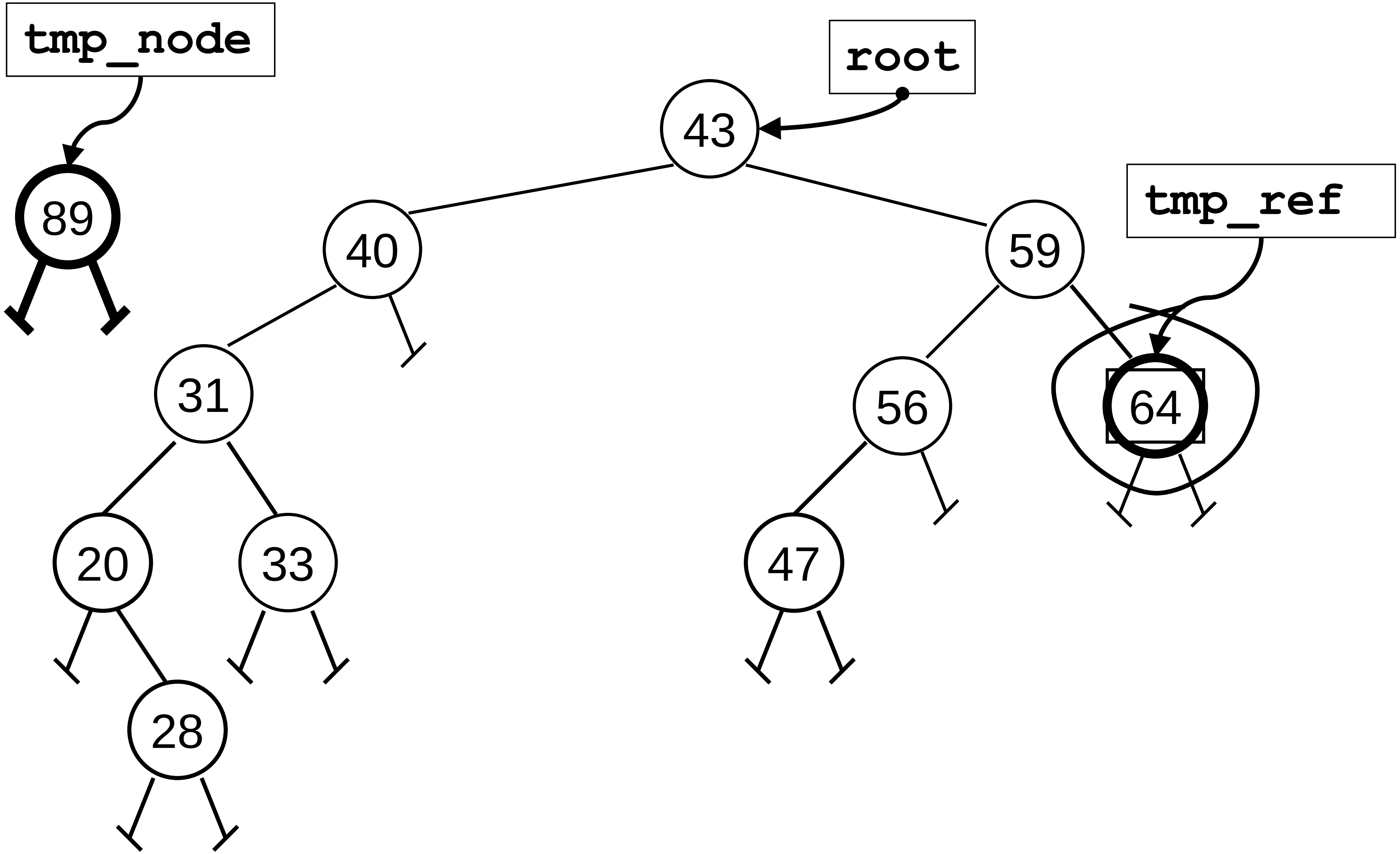
43	59	40	31	64	33	20	56	47	28	89



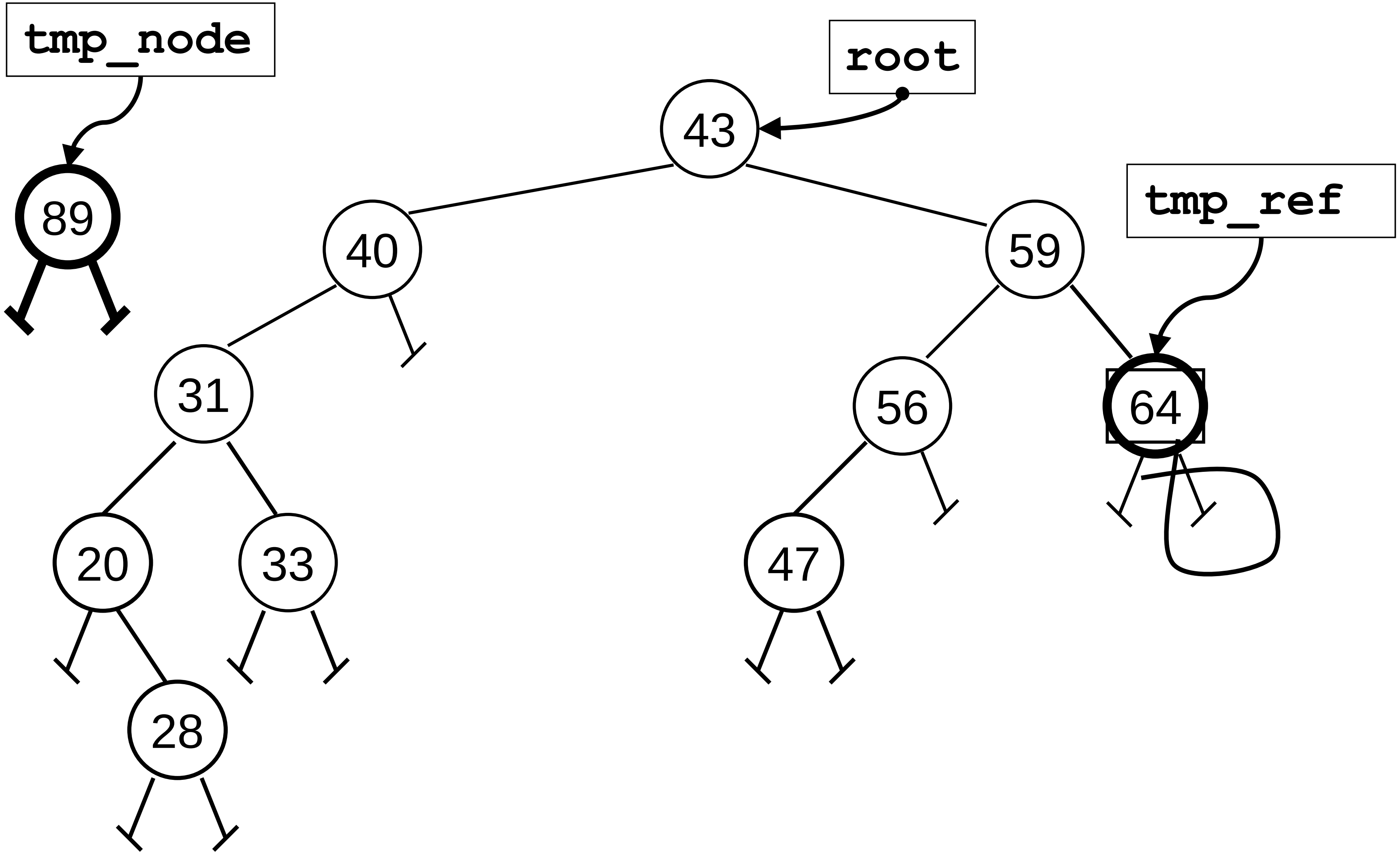
43	59	40	31	64	33	20	56	47	28	89



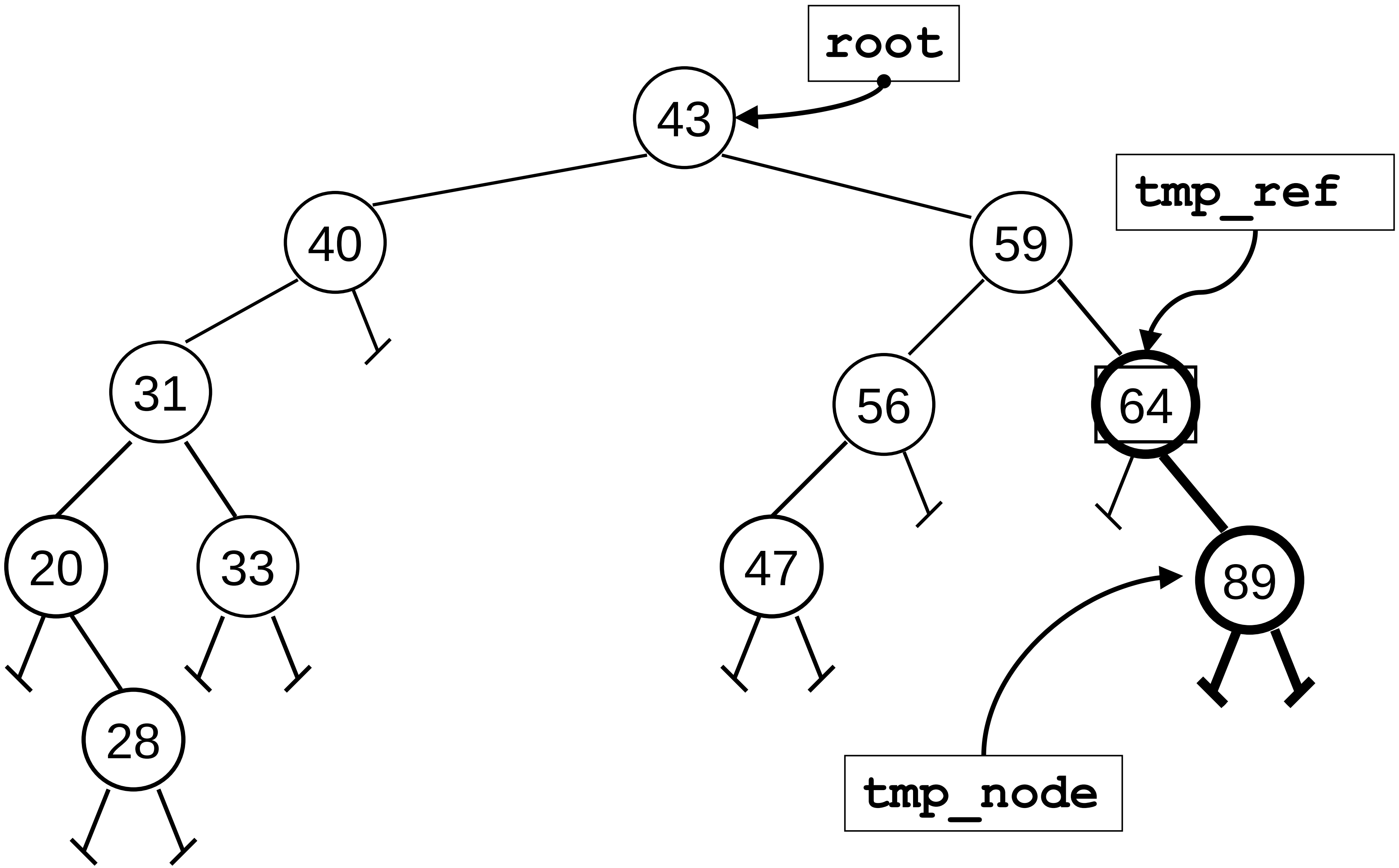
43	59	40	31	64	33	20	56	47	28	89



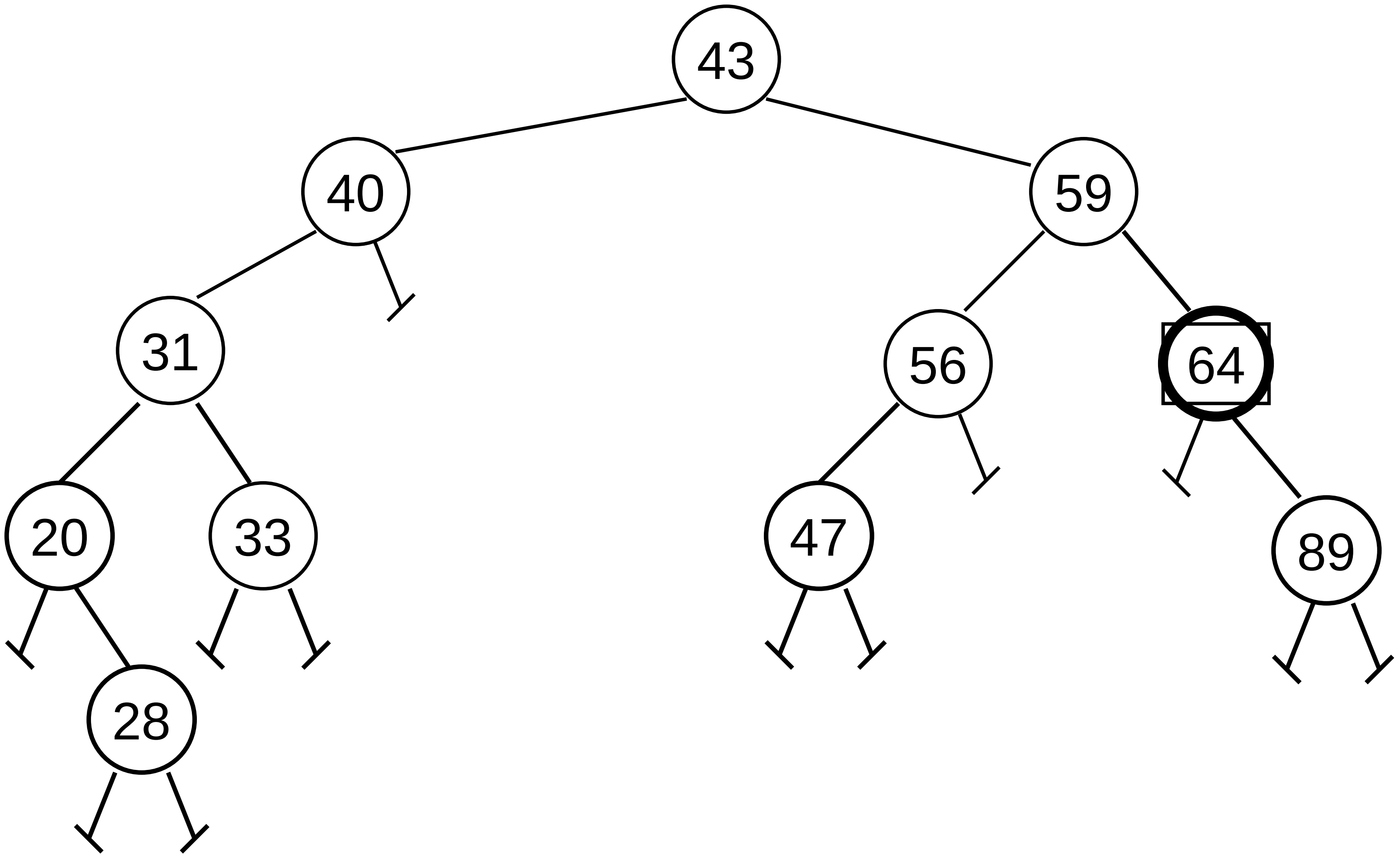
43	59	40	31	64	33	20	56	47	28	89



43	59	40	31	64	33	20	56	47	28	89



43	59	40	31	64	33	20	56	47	28	89



43	59	40	31	64	33	20	56	47	28	89

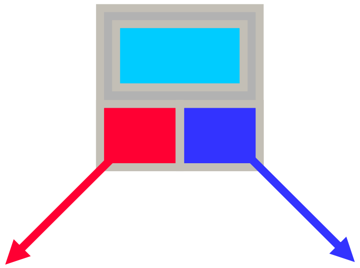
Summary of Insertion

- Preserve “search” structure!
- Inserting involves 2 steps:
  1. Find the correct location
     - For a BST insert, always insert at the “bottom” of the tree (i.e., LEFT or RIGHT child of a leaf node)
  2. Do commands to add node
     - Create node
     - Add DATA
     - Make LEFT and RIGHT pointers point to NULL

Traversal

- **Now we will explore ‘traversal’ of a binary tree**
- **Traversal: How do we move around the tree to interact with (or simply print) the data stored inside of it?**

In-Order Traversal: The Scenario



- Imagine we have a binary tree
- We want to traverse the tree
  - It's not linear
  - We need a way to visit all nodes

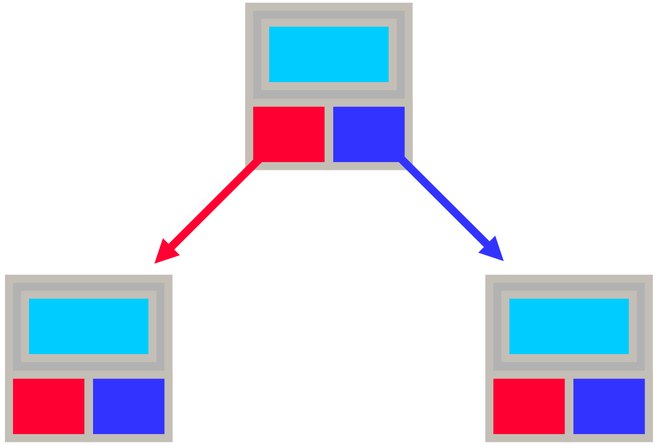
- Three things must happen:
  - Deal with the entire left sub-tree
  - Deal with the current node
  - Deal with the entire right sub-tree

- Result is IN-ORDER traversal

Outline of In-Order Traversal

- Three principle steps:
  - Traverse Left
  - Do work (Current)
  - Traverse Right

- Work can be anything
- Separate work from traversal

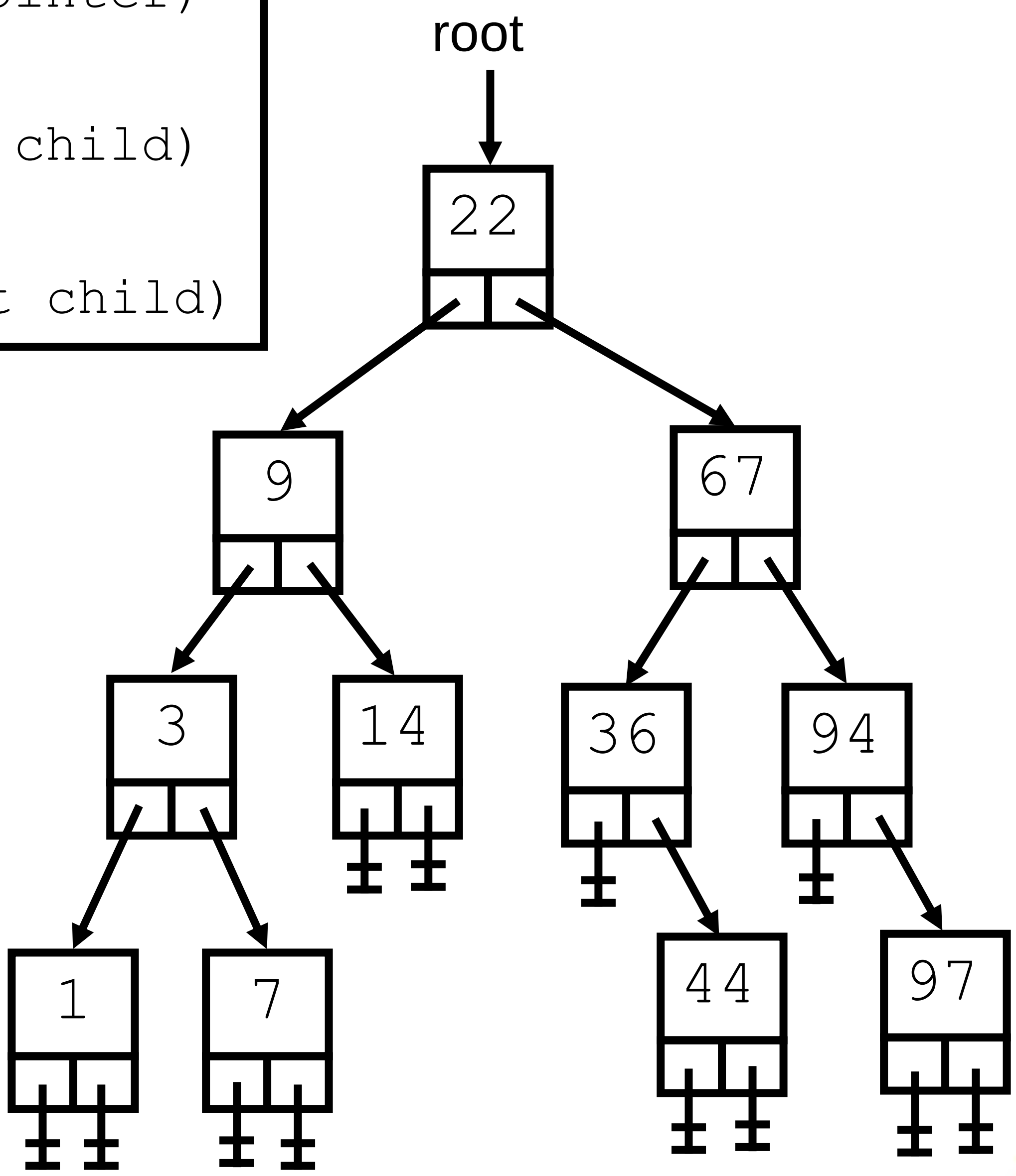


- Traverse the tree "In order":
  - Visit the tree's left sub-tree
  - Visit the current and do work
  - Visit right sub-tree

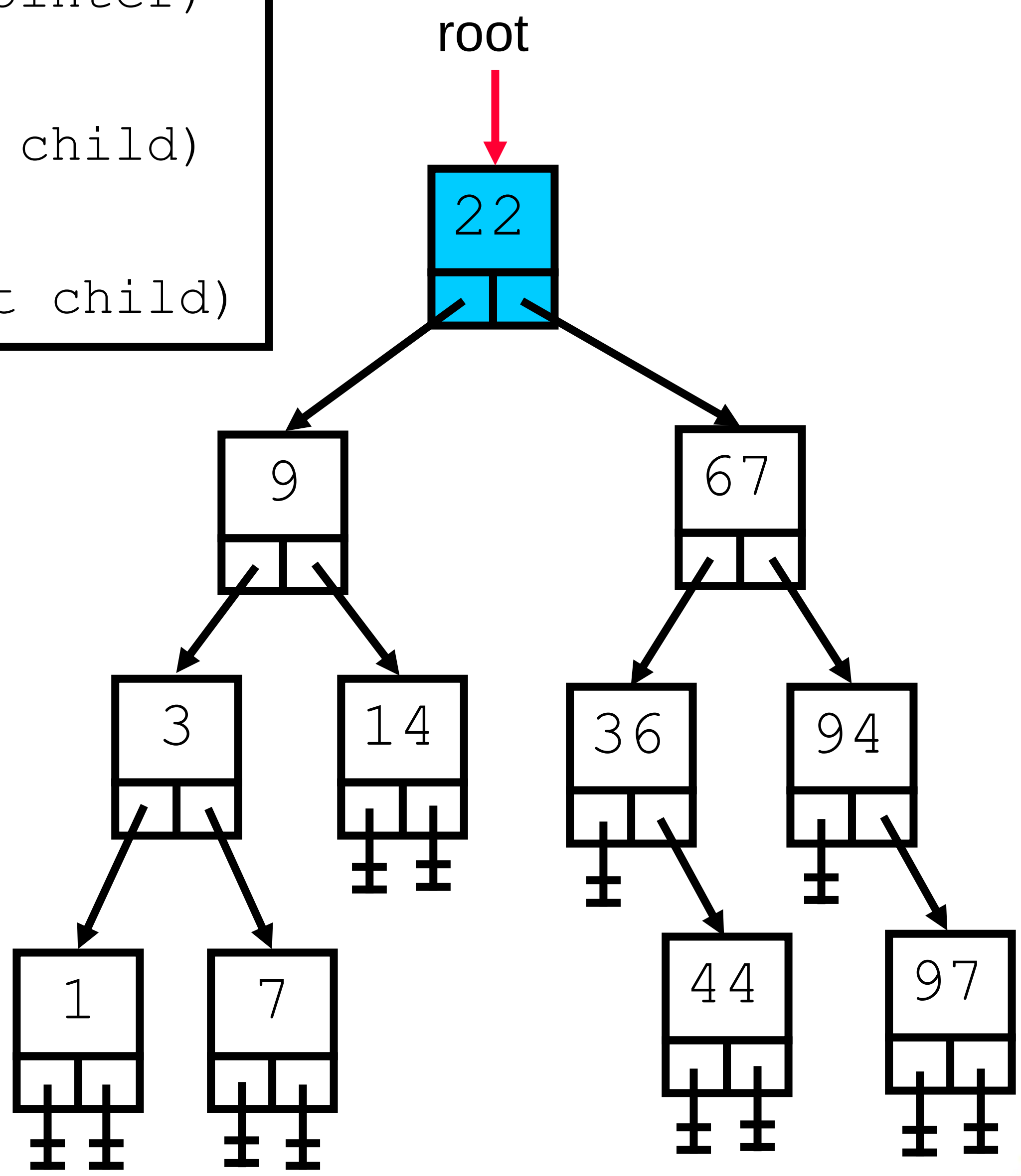
In-Order Traversal Procedure

```
procedure In_Order(cur iot in Ptr toa Tree_Node)
// Purpose: perform in-order traversal, call
//          Do_Something for each node
// Preconditions: cur points to a binary tree
// Postcondition: Do_Something on each tree
//                node in “in-order” order
  if( cur <> NULL ) then
    In_Order( cur^.left_child )
    Do_Something( cur^.data )
    In_Order( cur^.right_child )
  endif
endprocedure   // In_Order
```

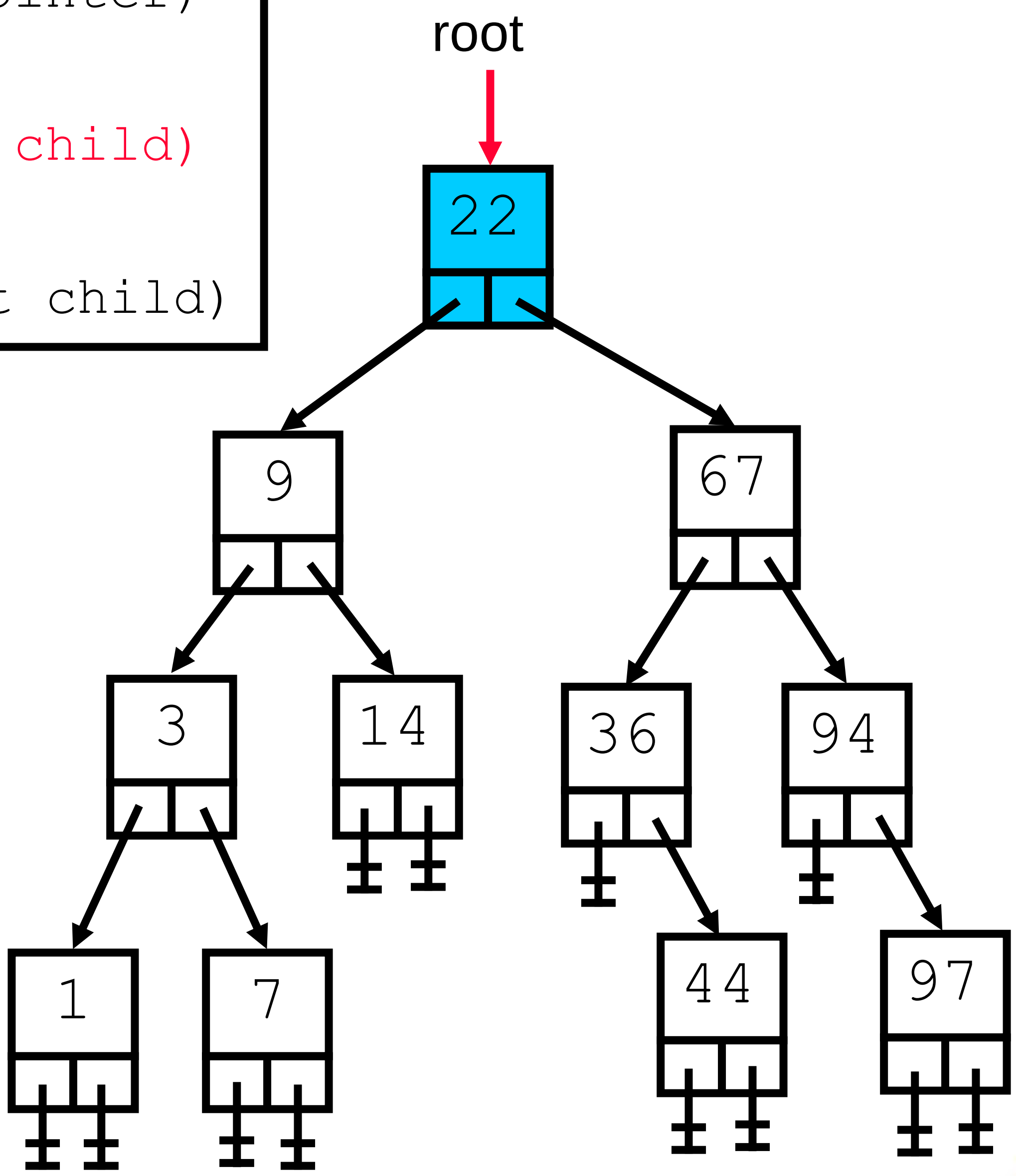
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```



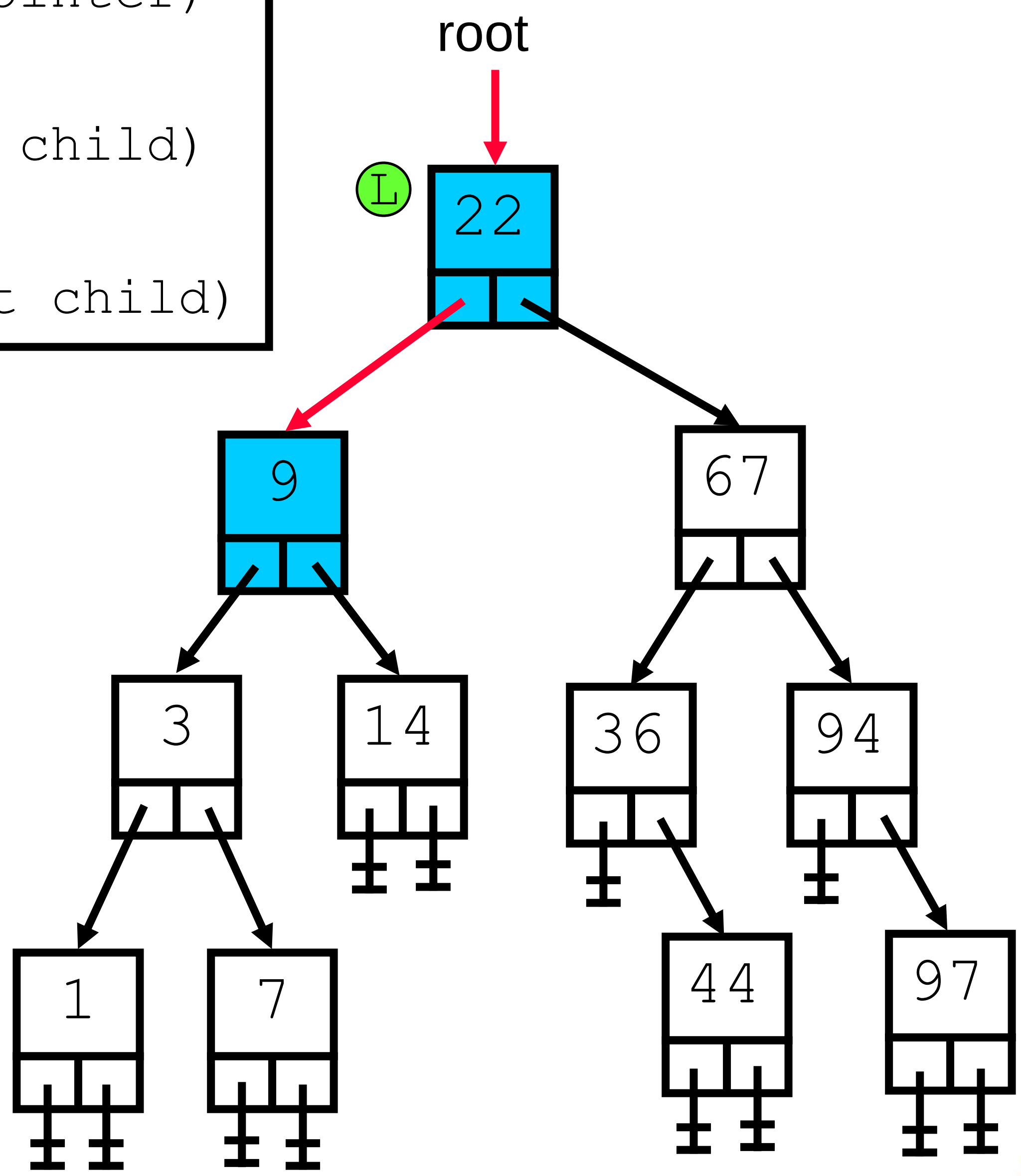
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```



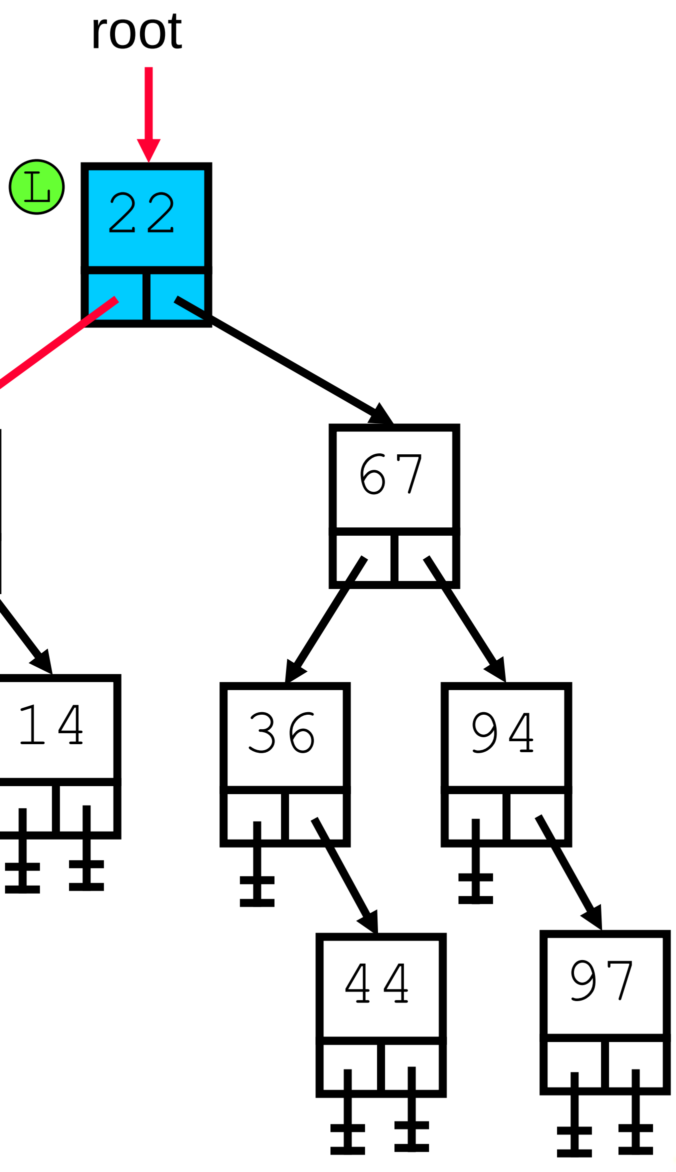
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```



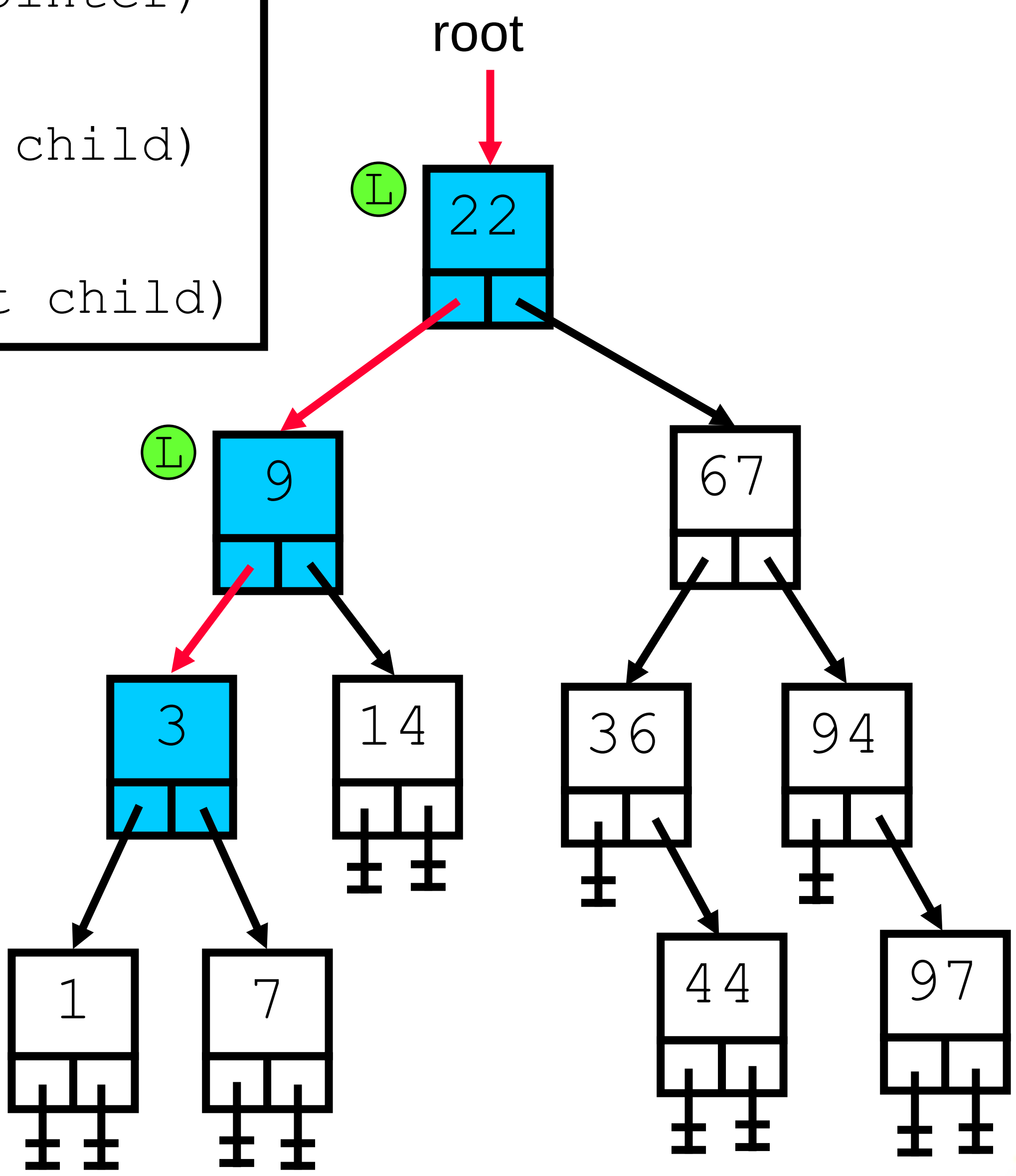
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```



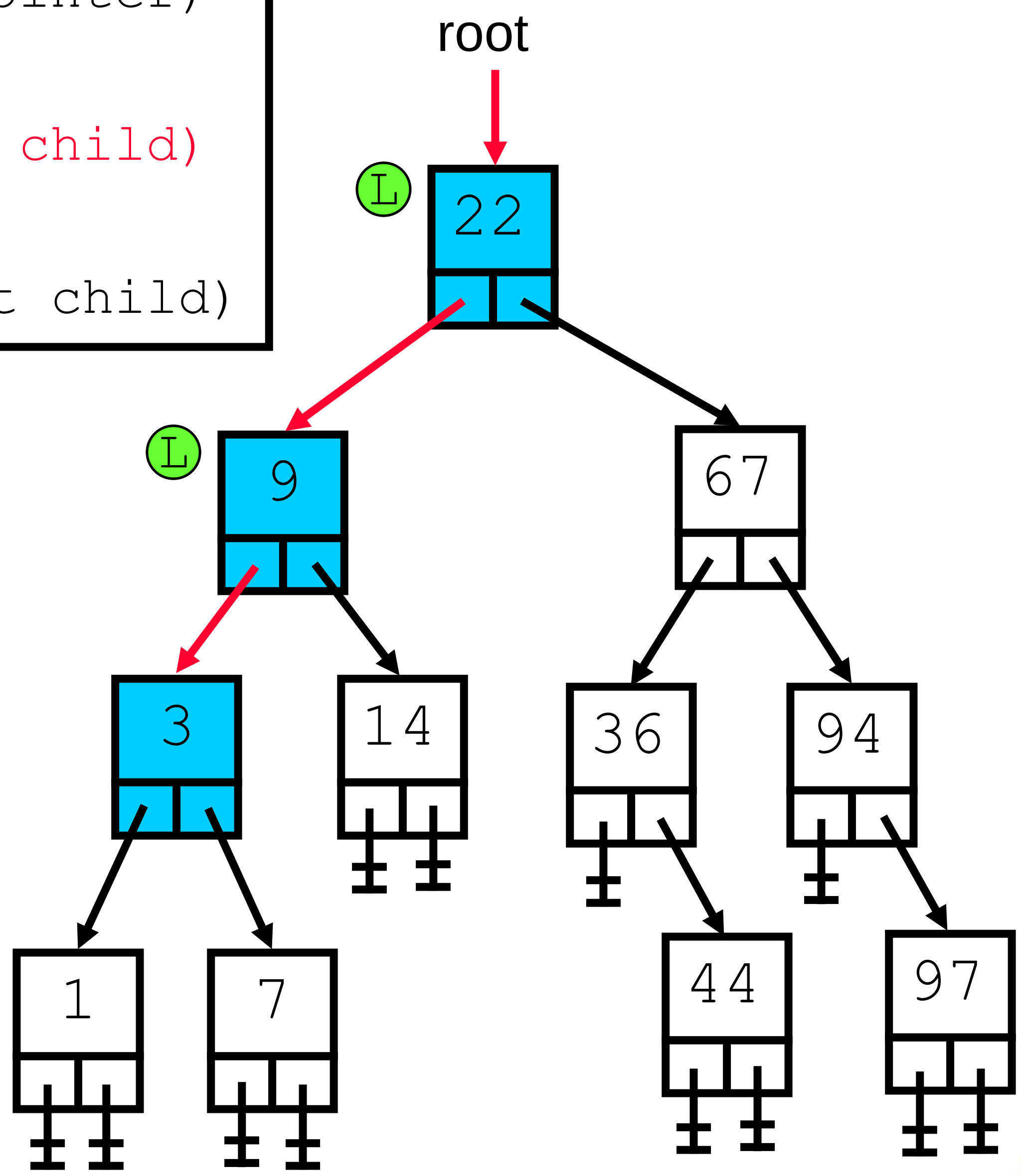
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```



```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

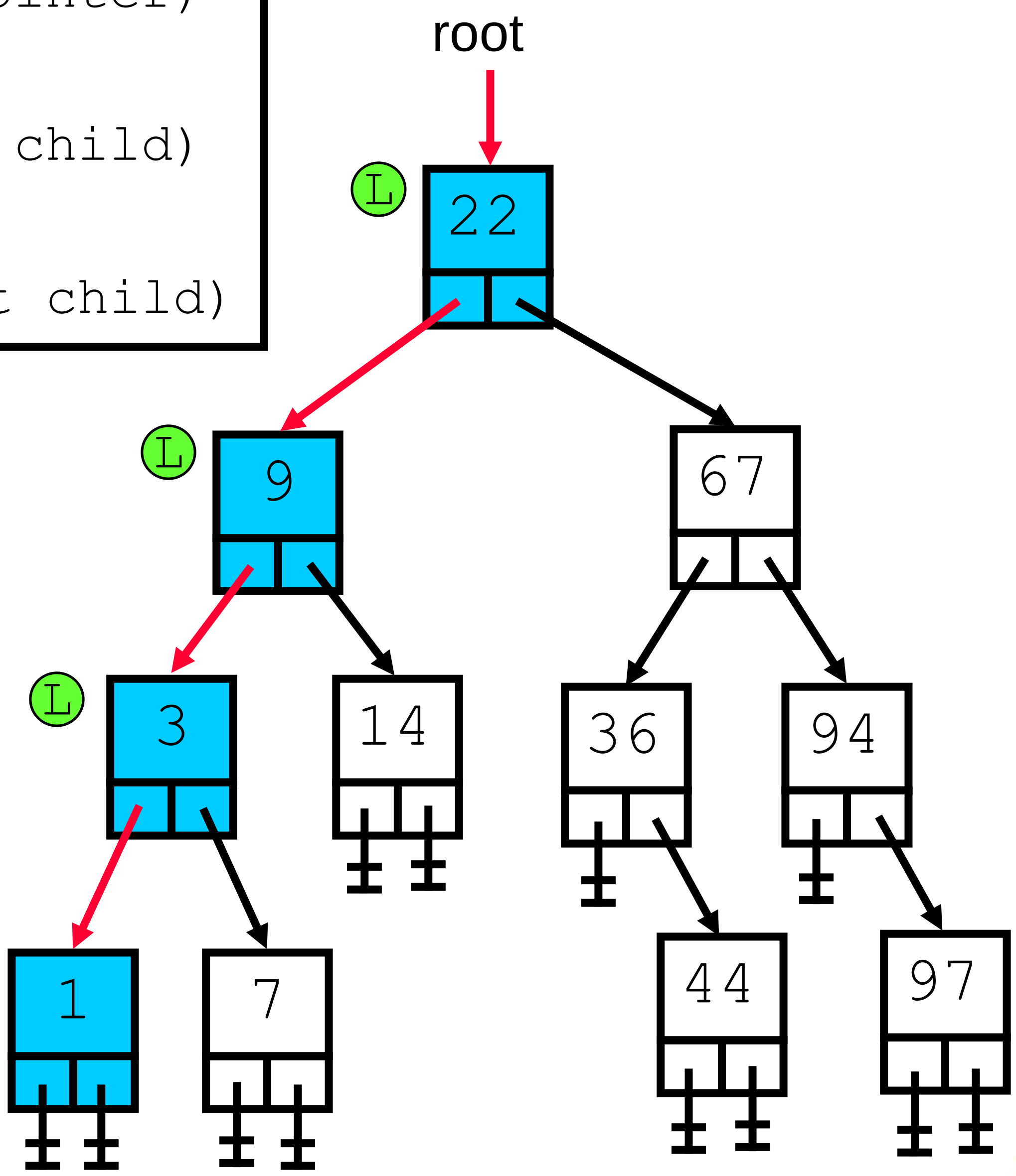


```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

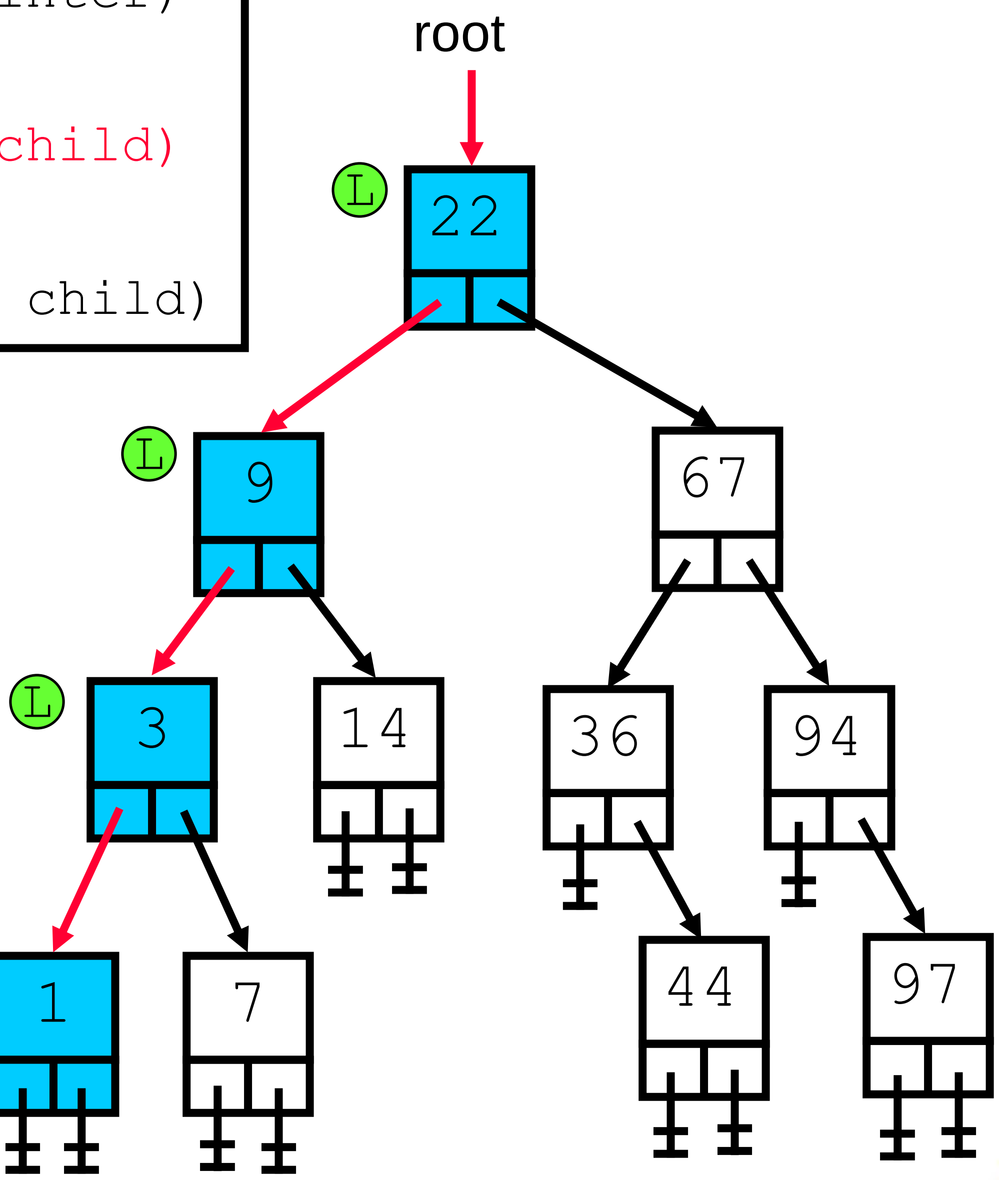


UNIVERSITY OF CALGARY

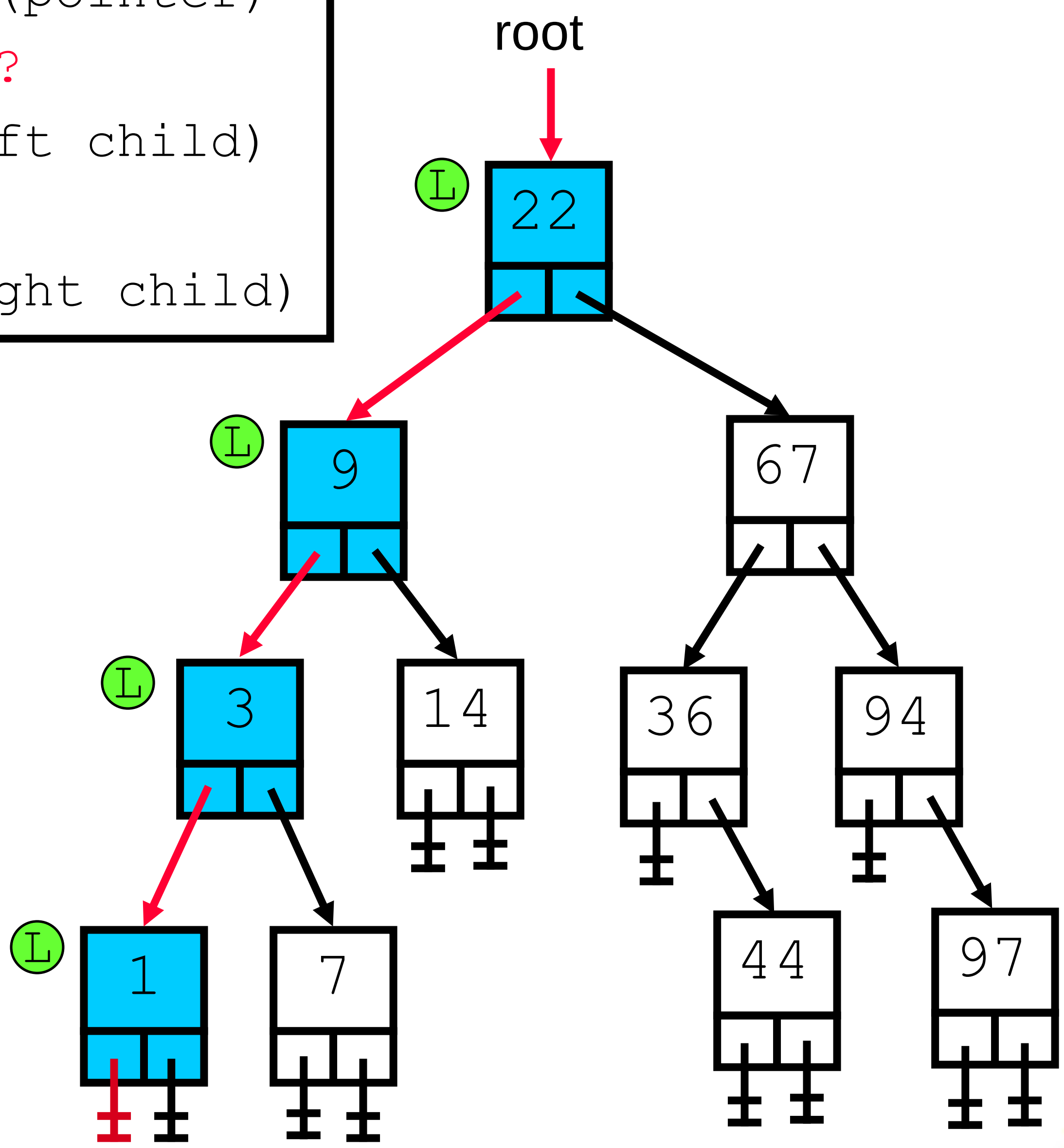
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```



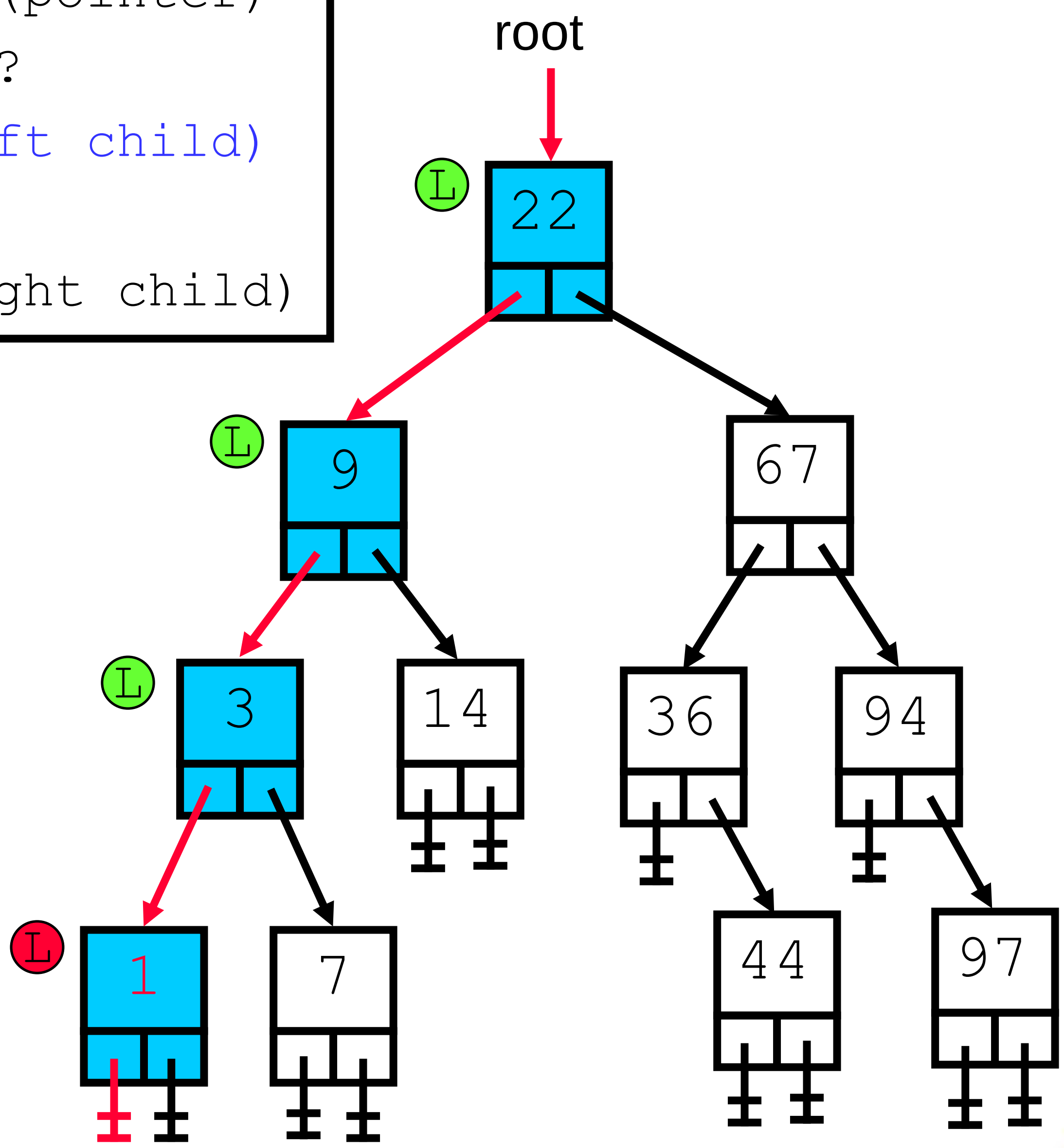
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```



```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

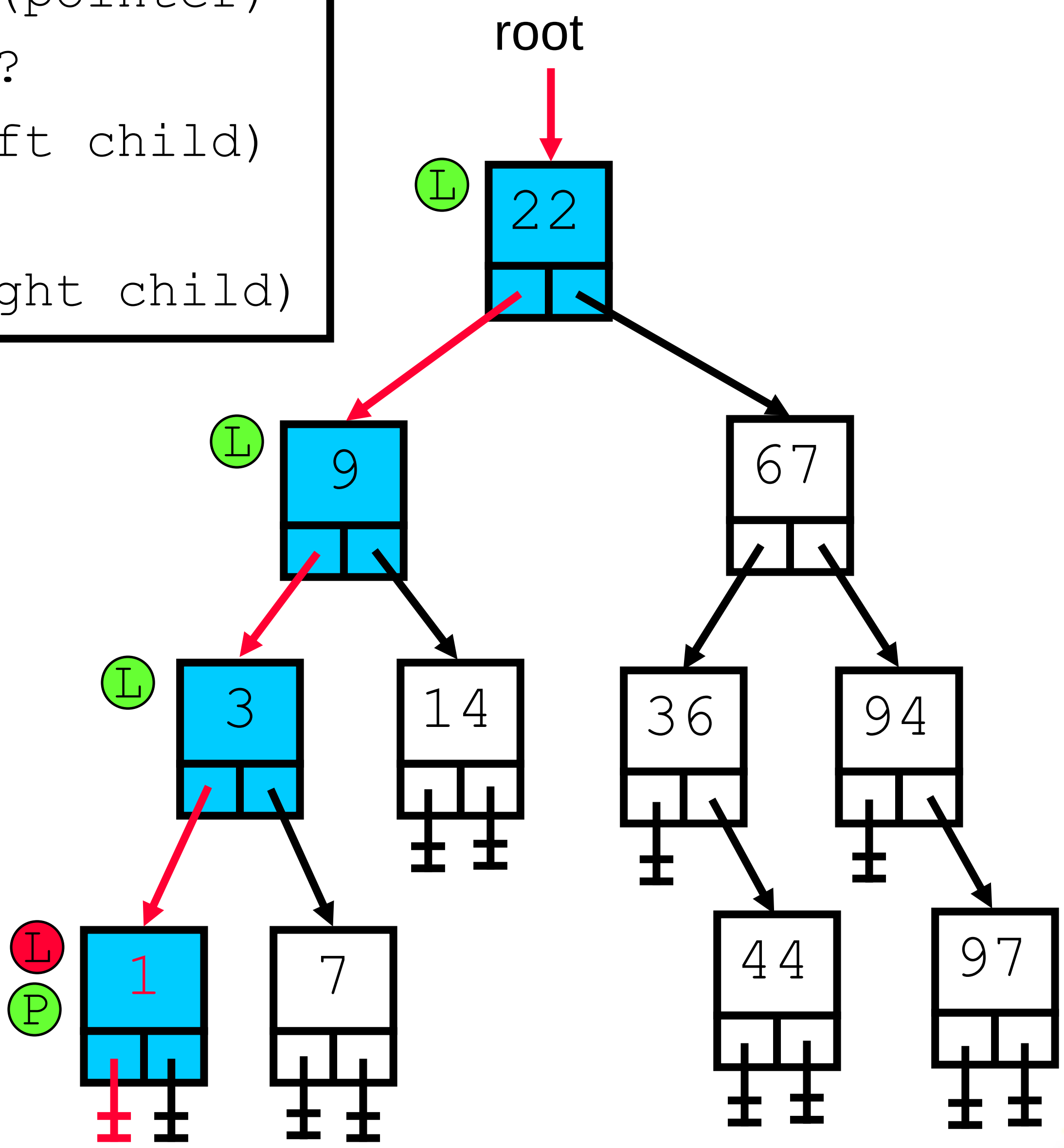


```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```



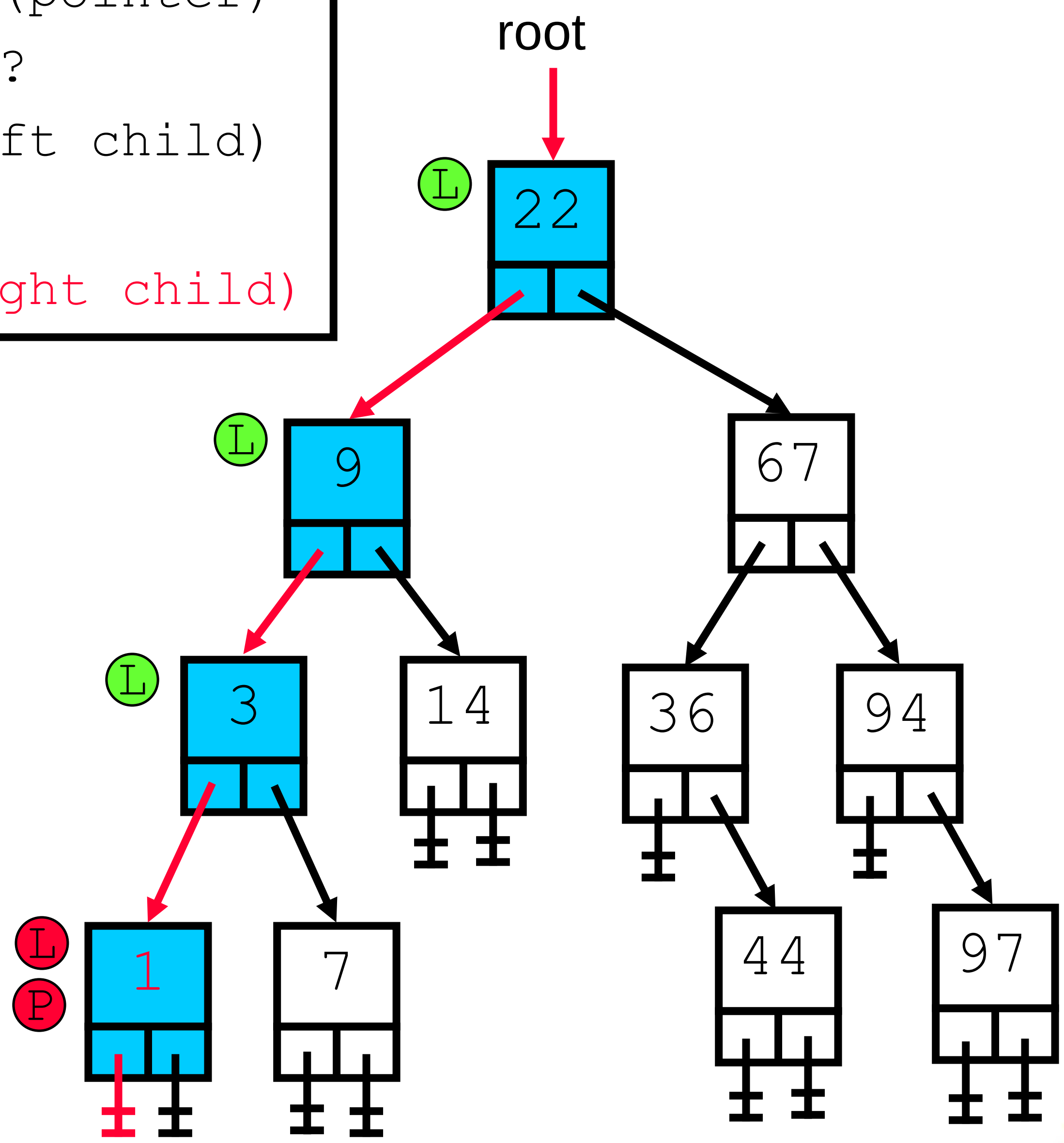
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1
```



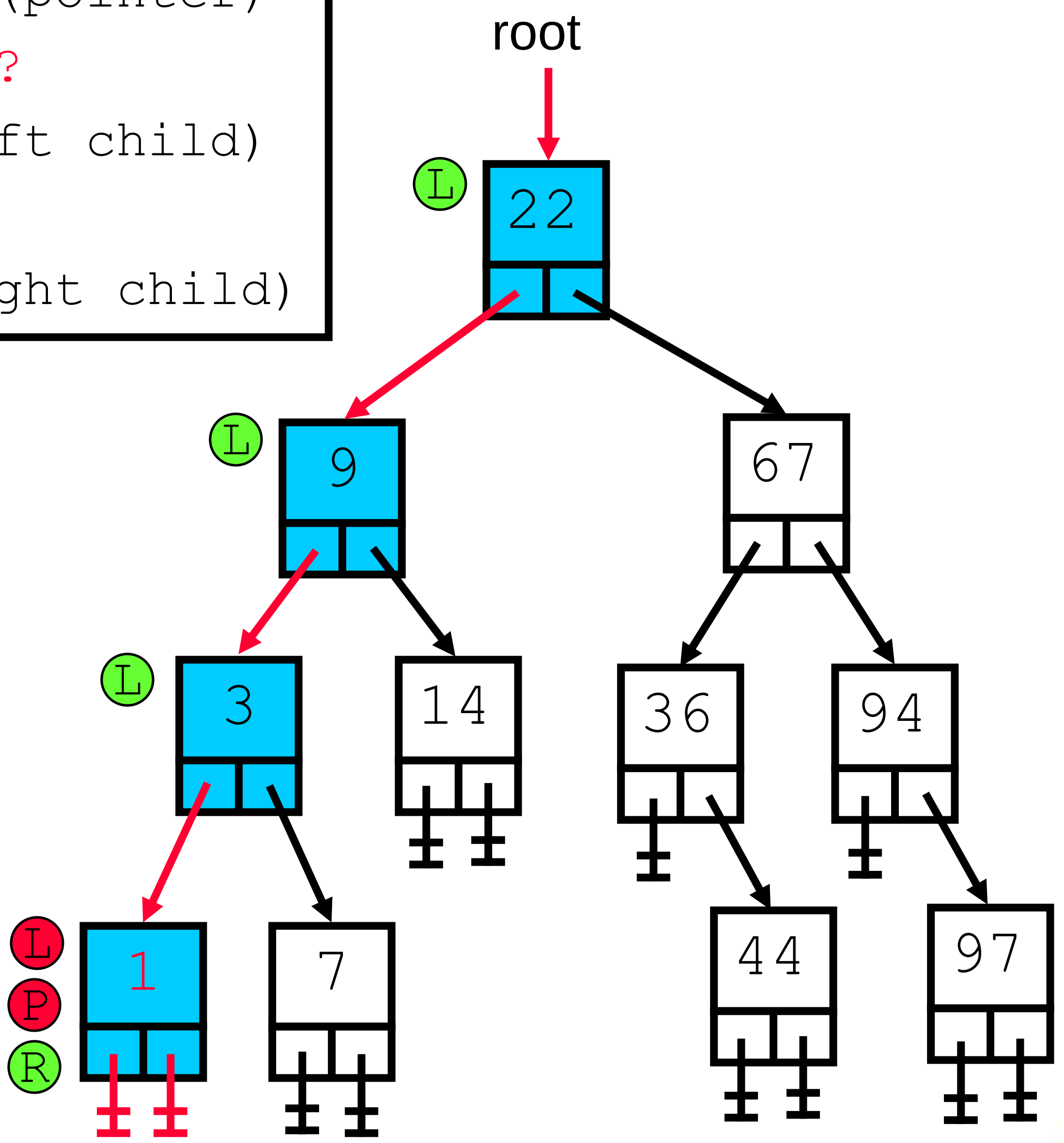
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1
```



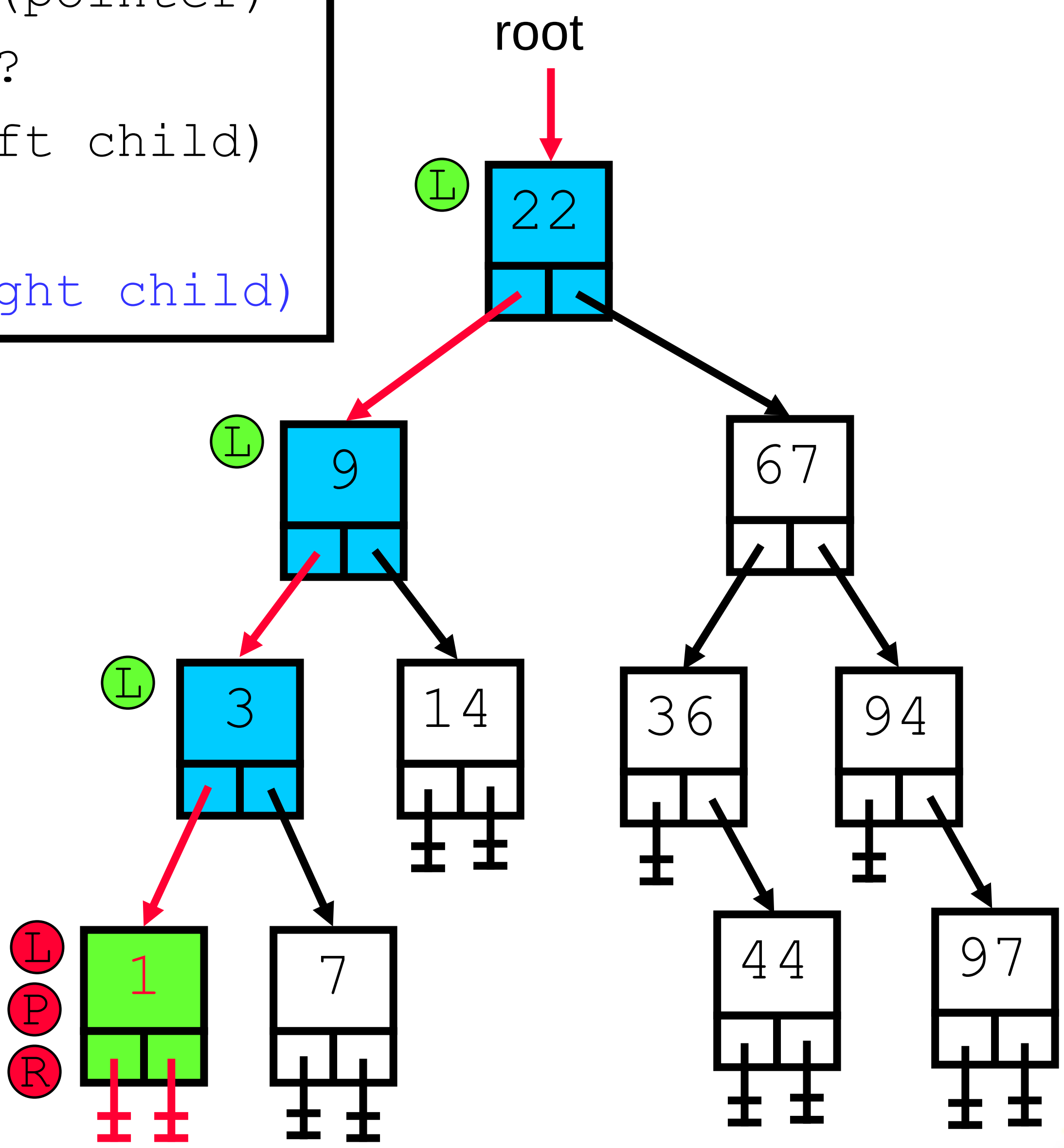
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1
```



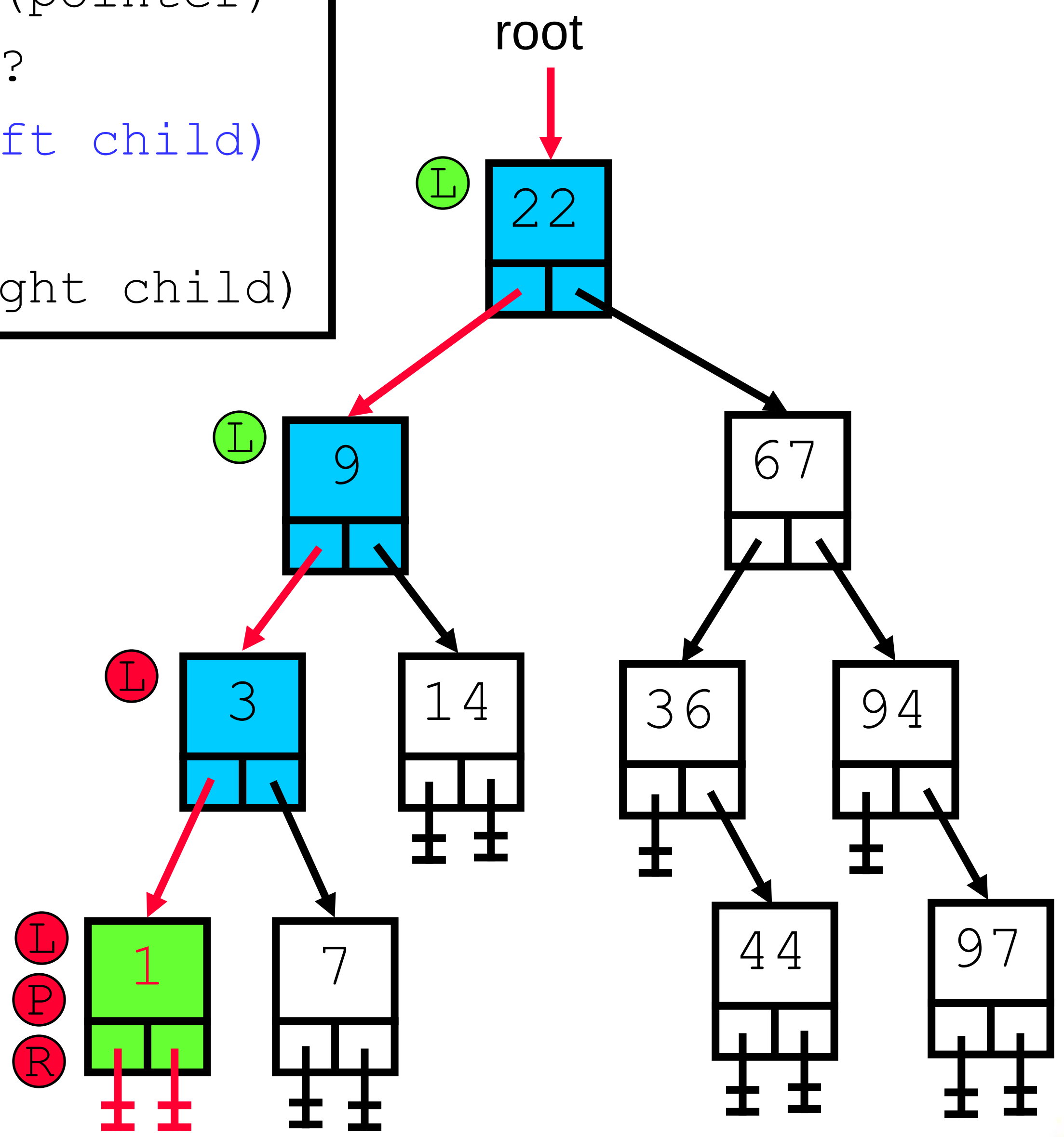
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1
```



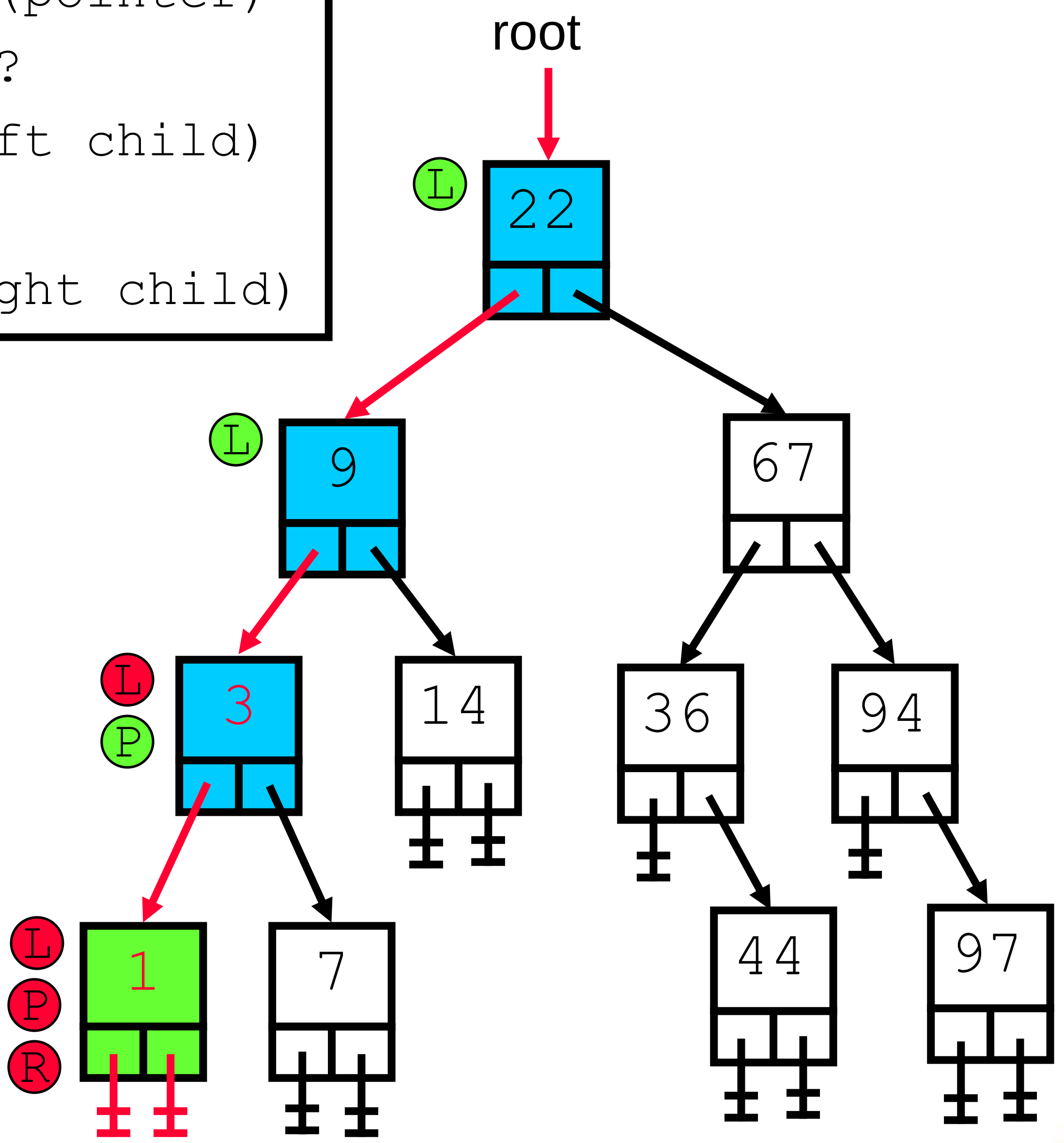
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1
```



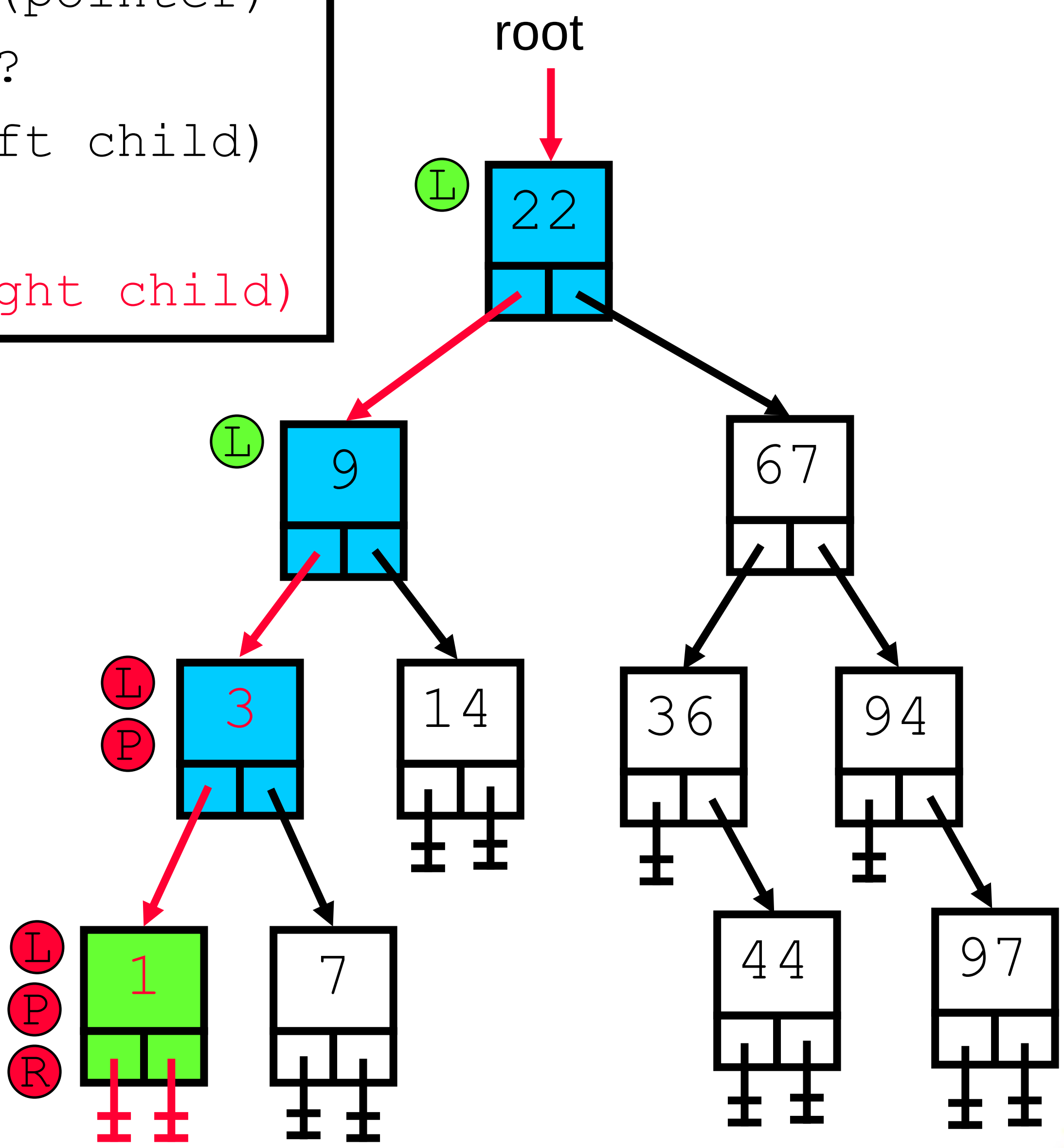
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3
```



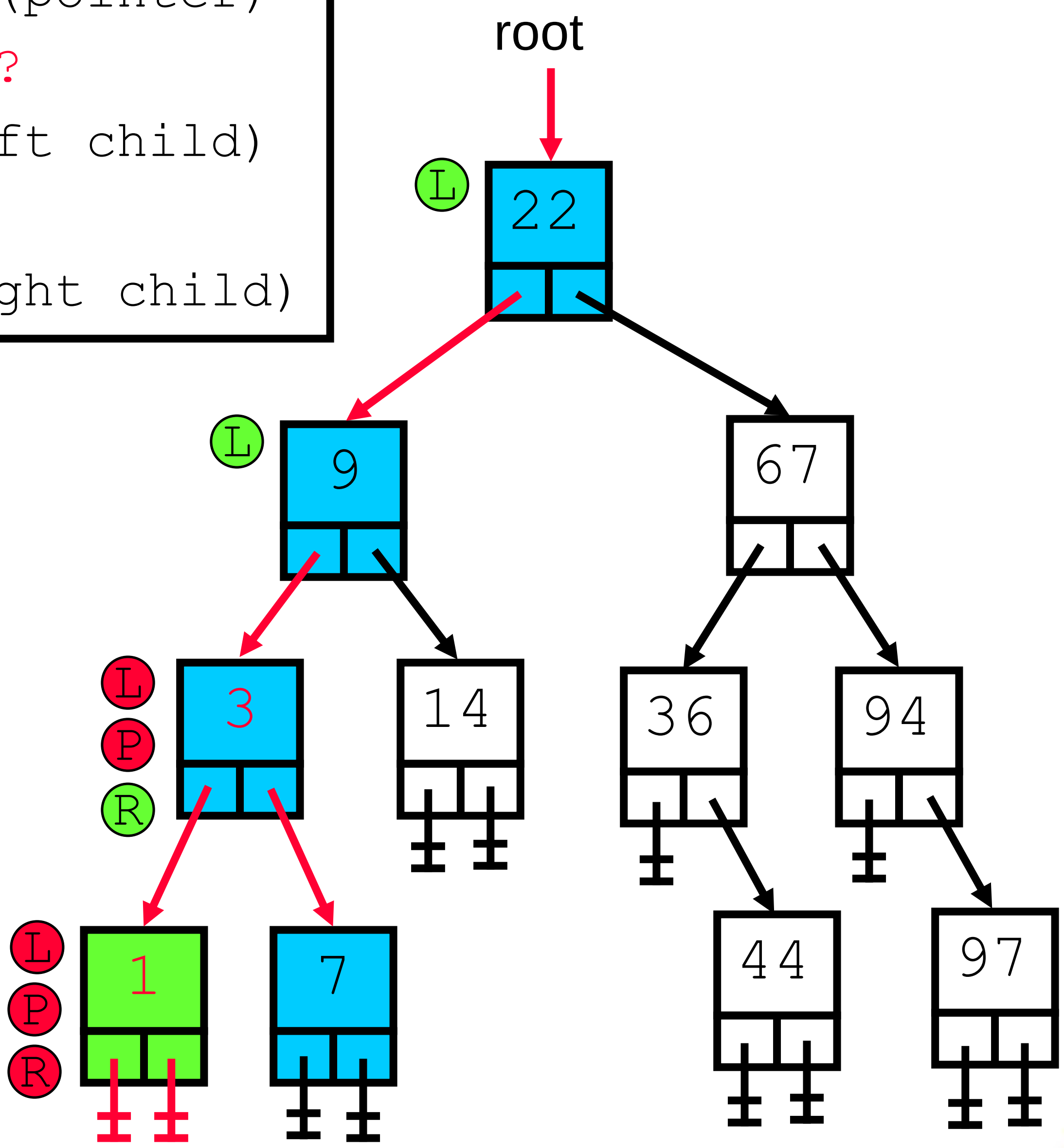
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3
```



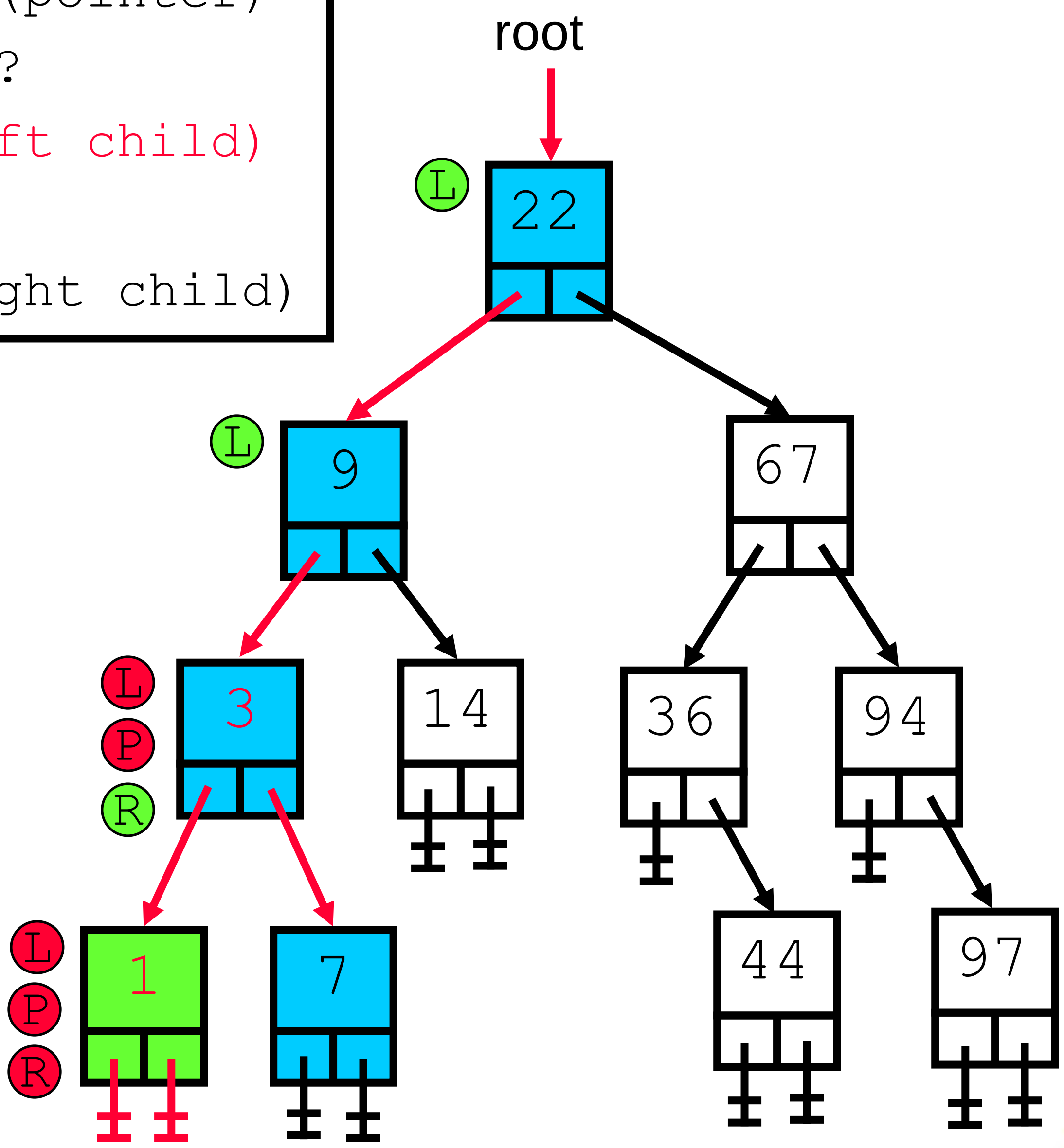
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3
```



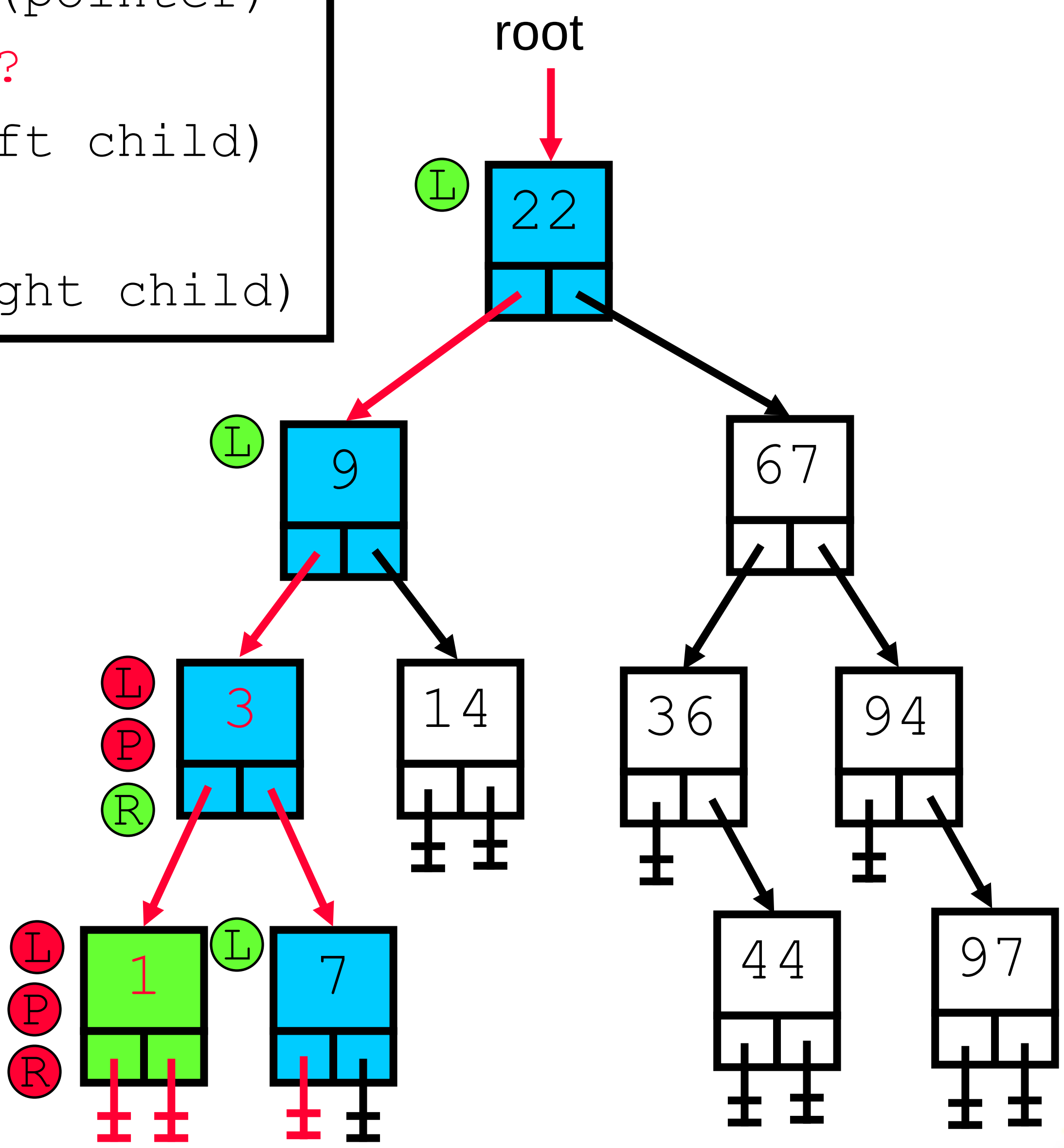
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3
```



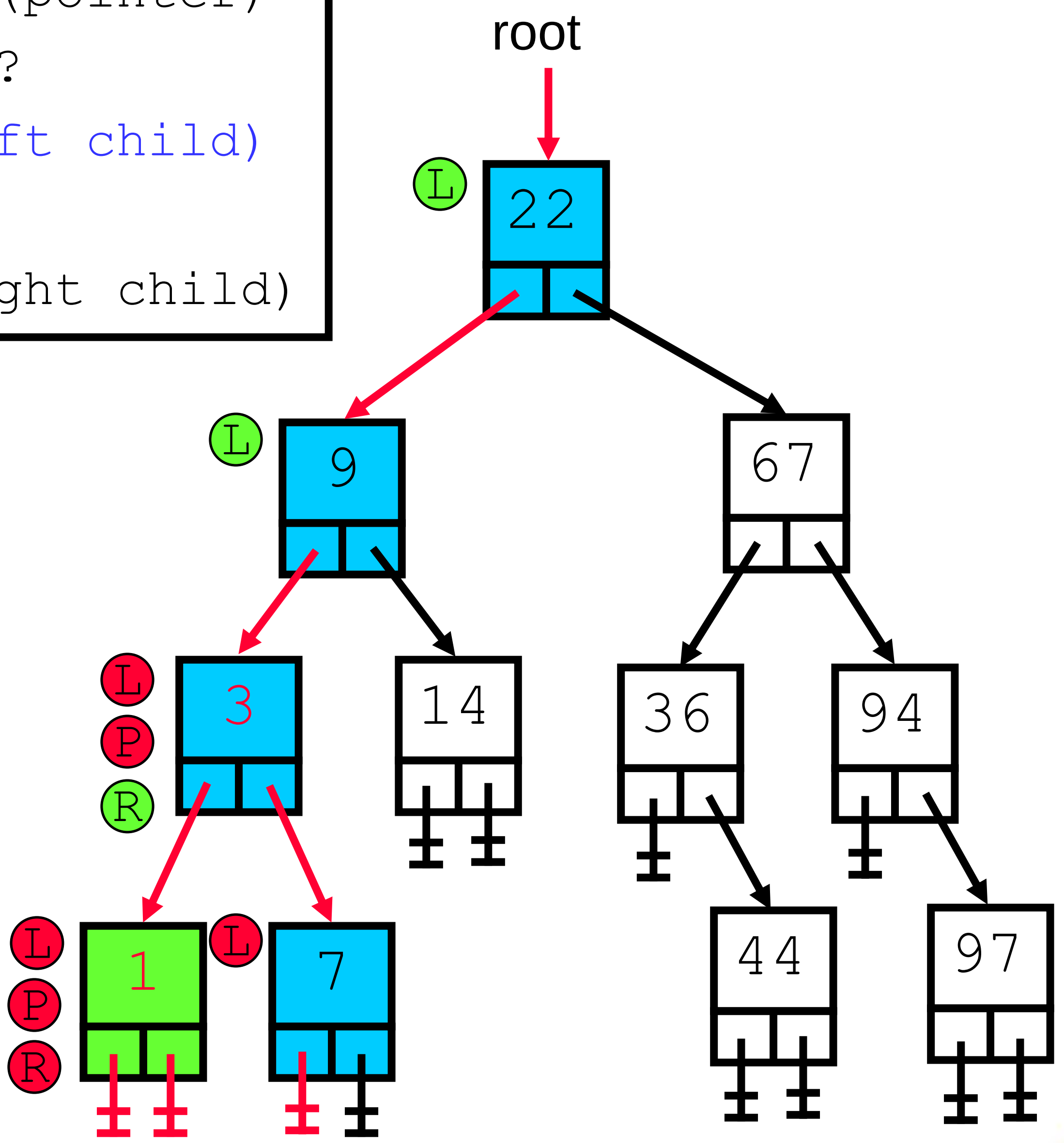
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3
```



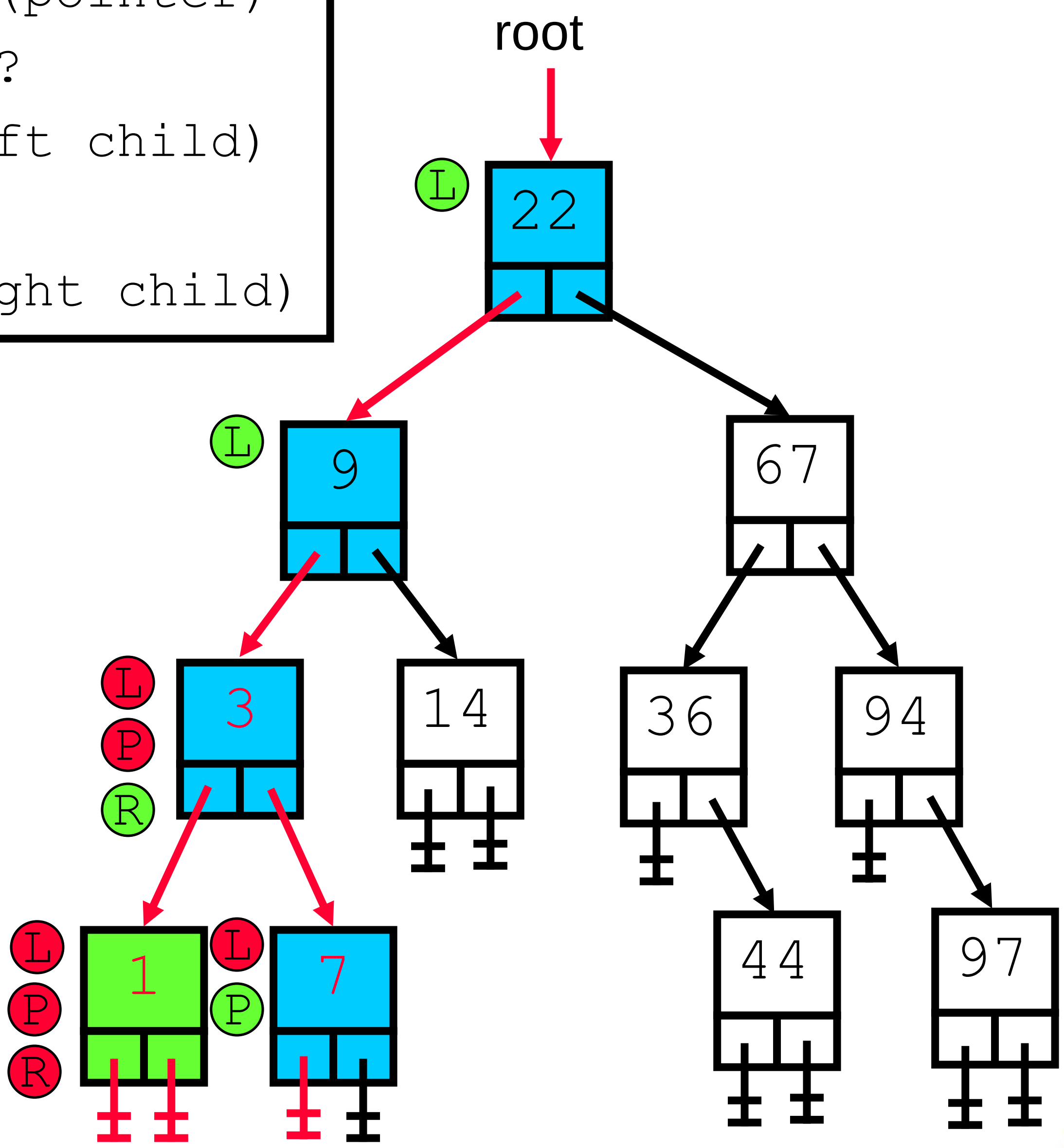
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3
```



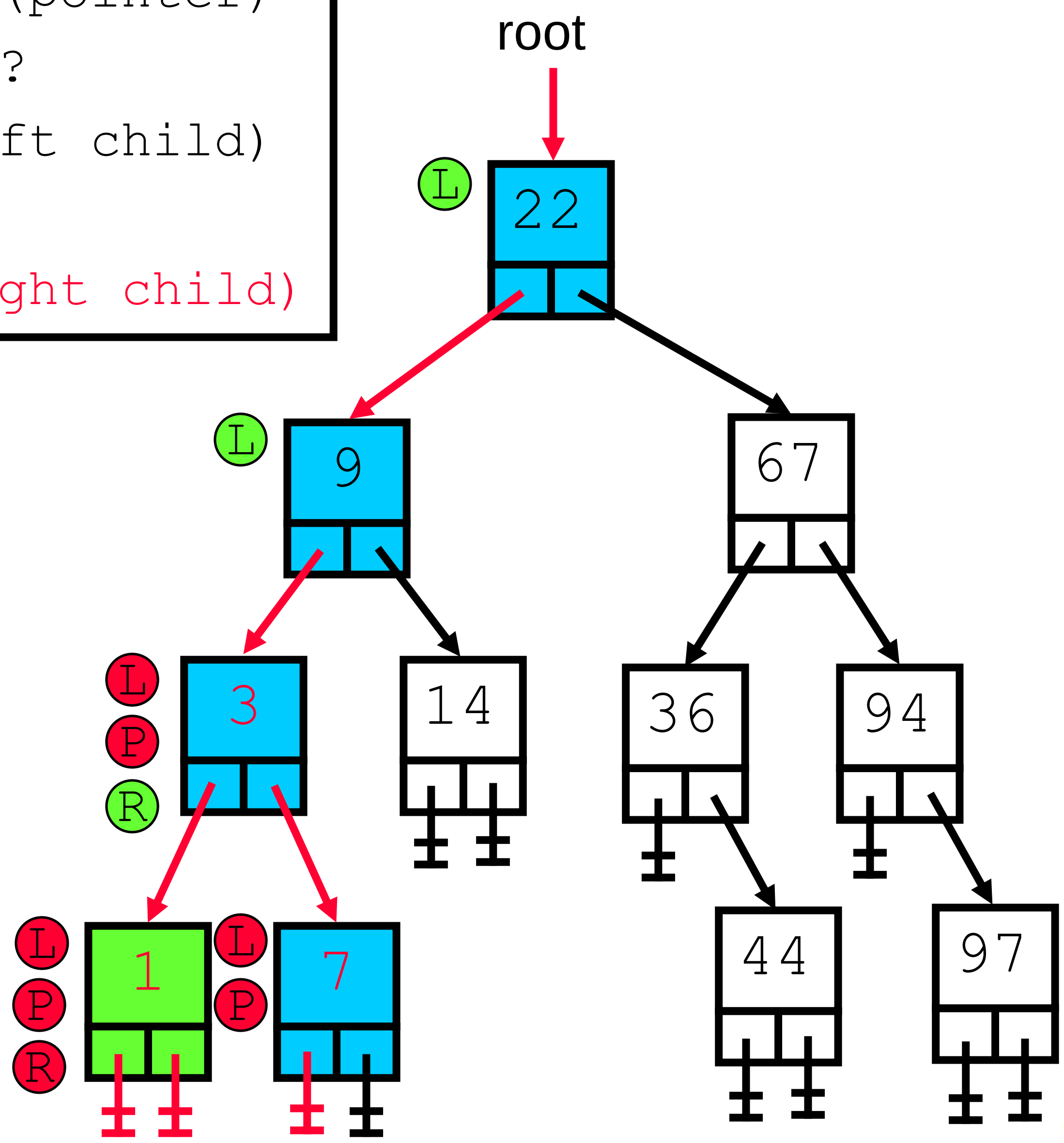
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

Output: 1 3 7



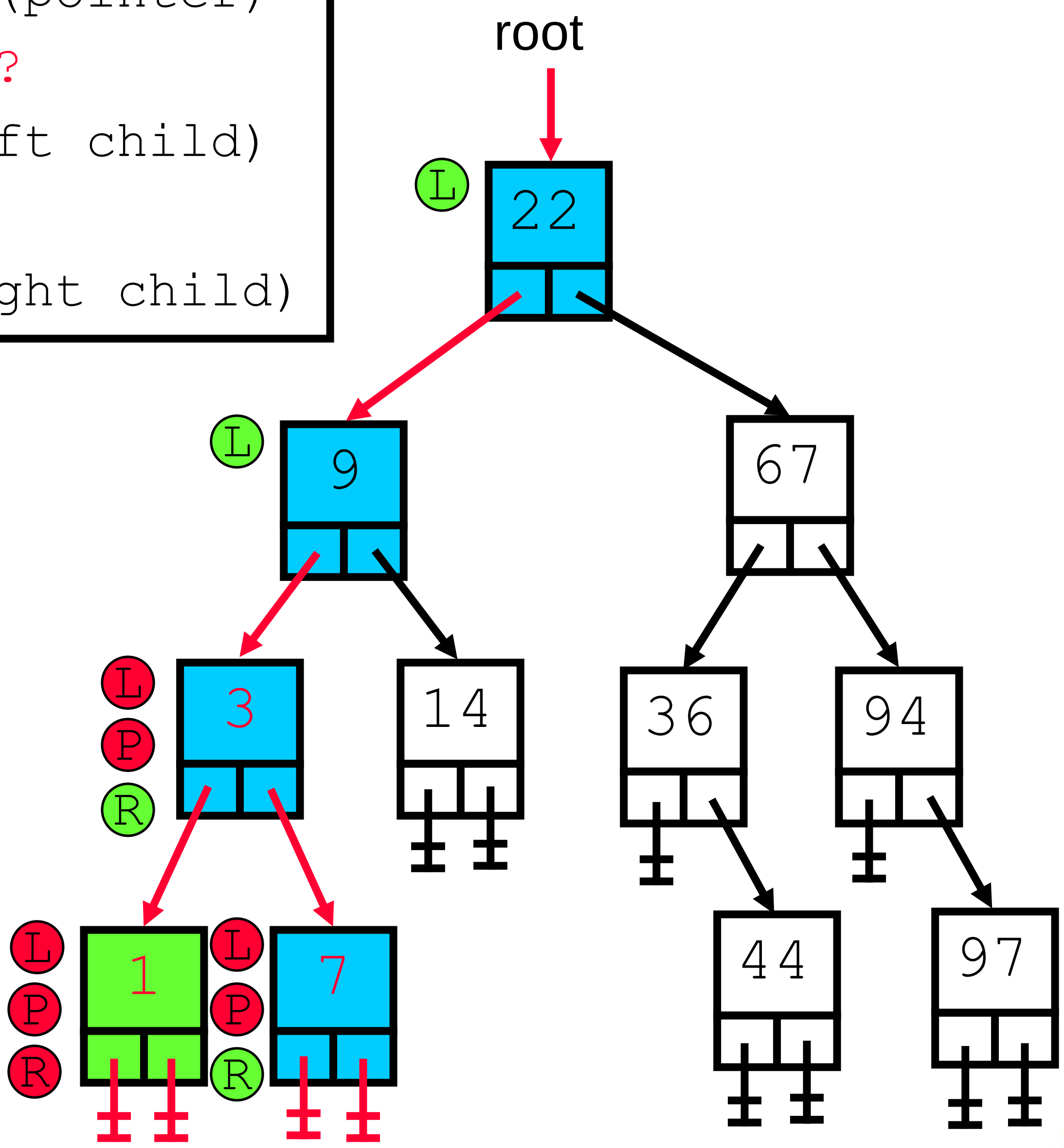
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7
```



```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

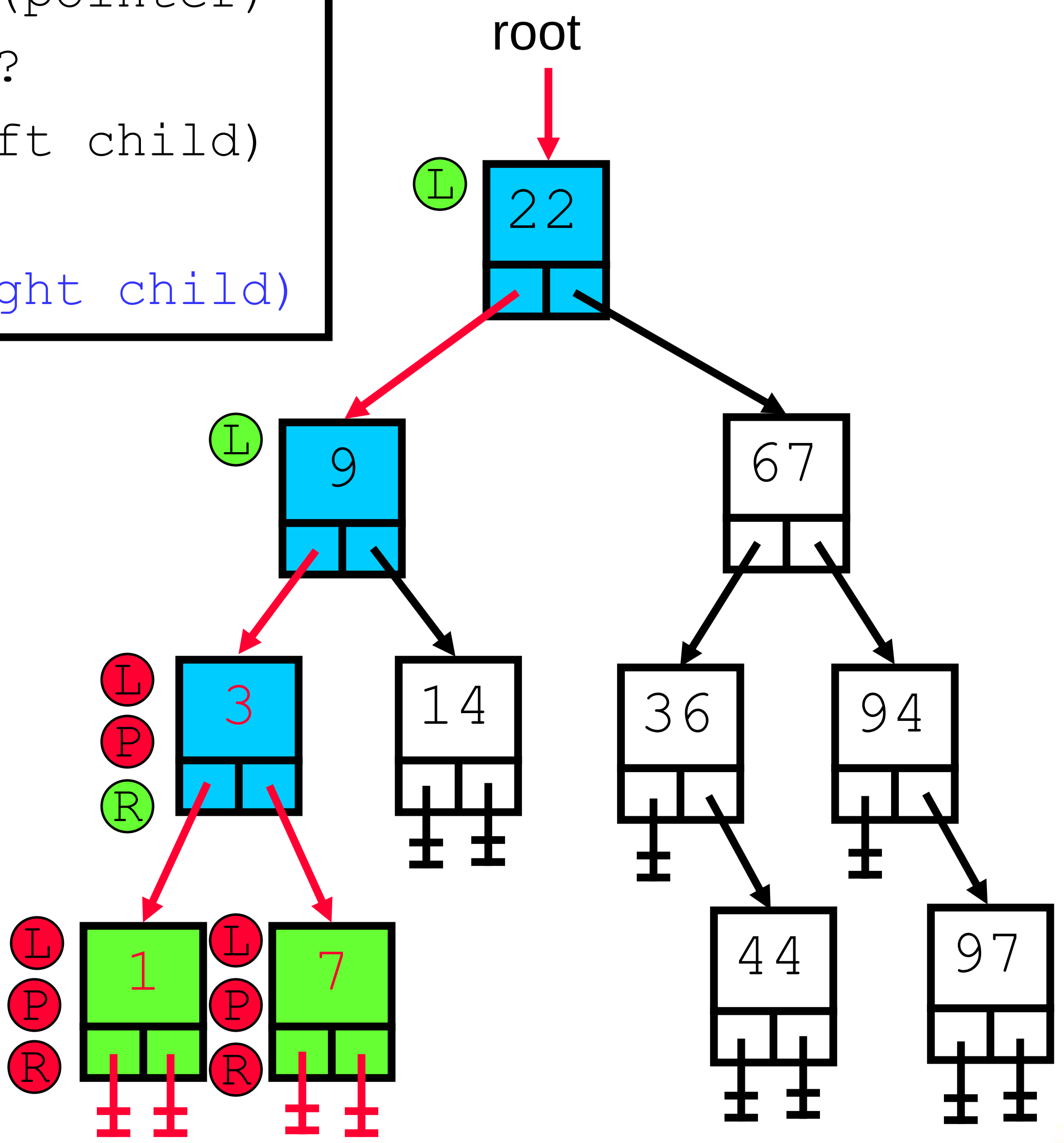
```
Output: 1 3 7
```



UNIVERSITY OF CALGARY

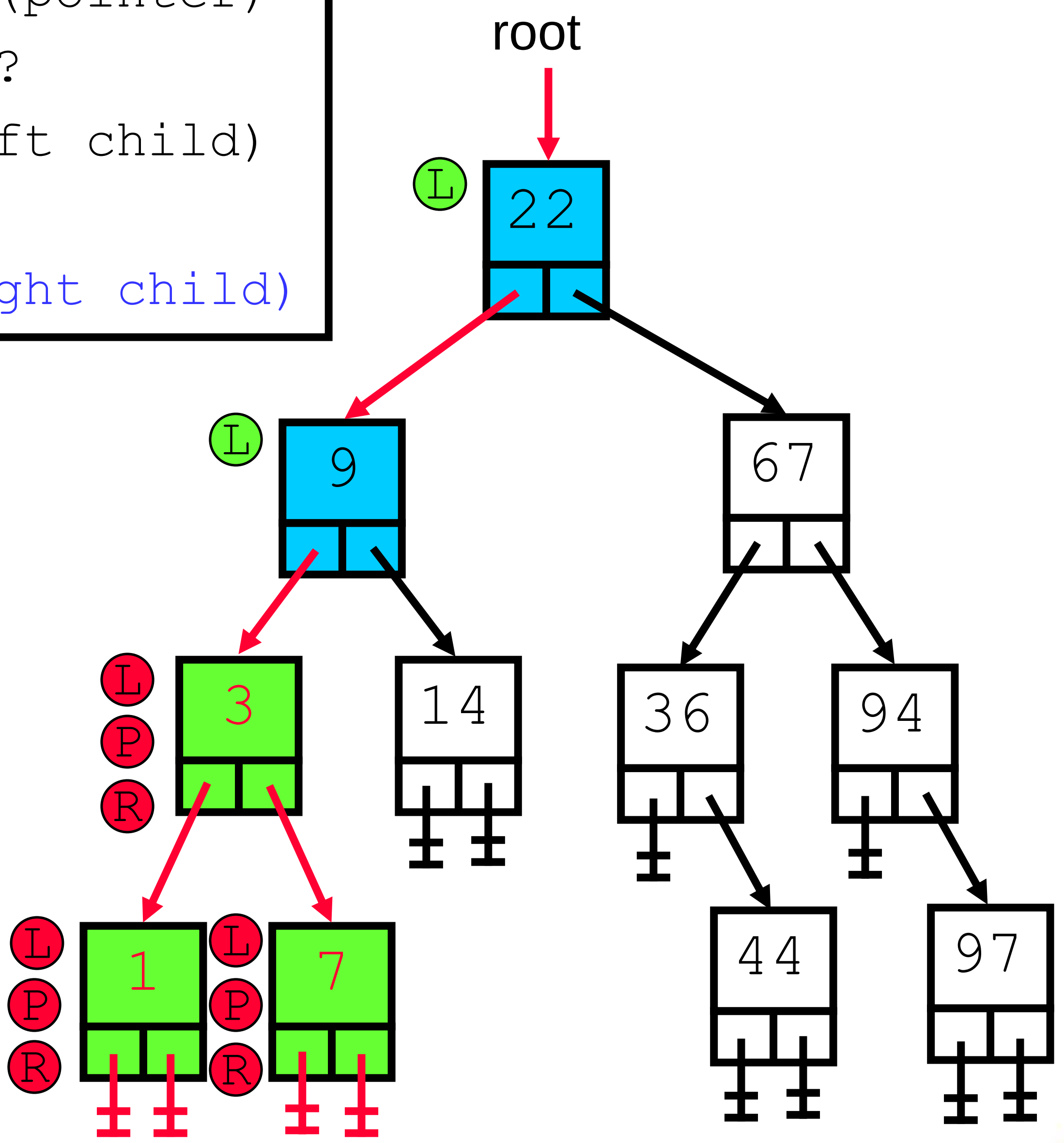
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7
```



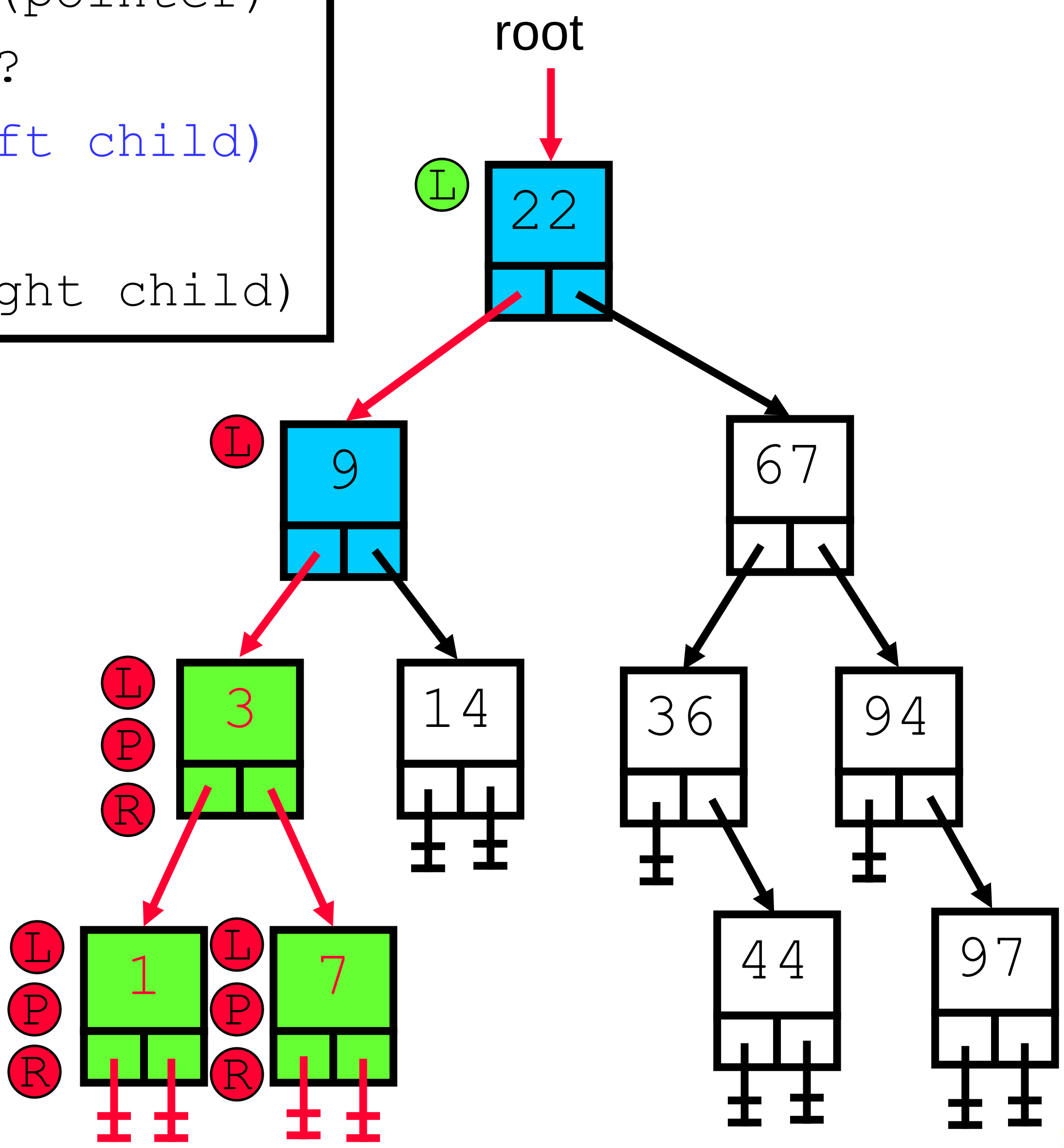
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7
```



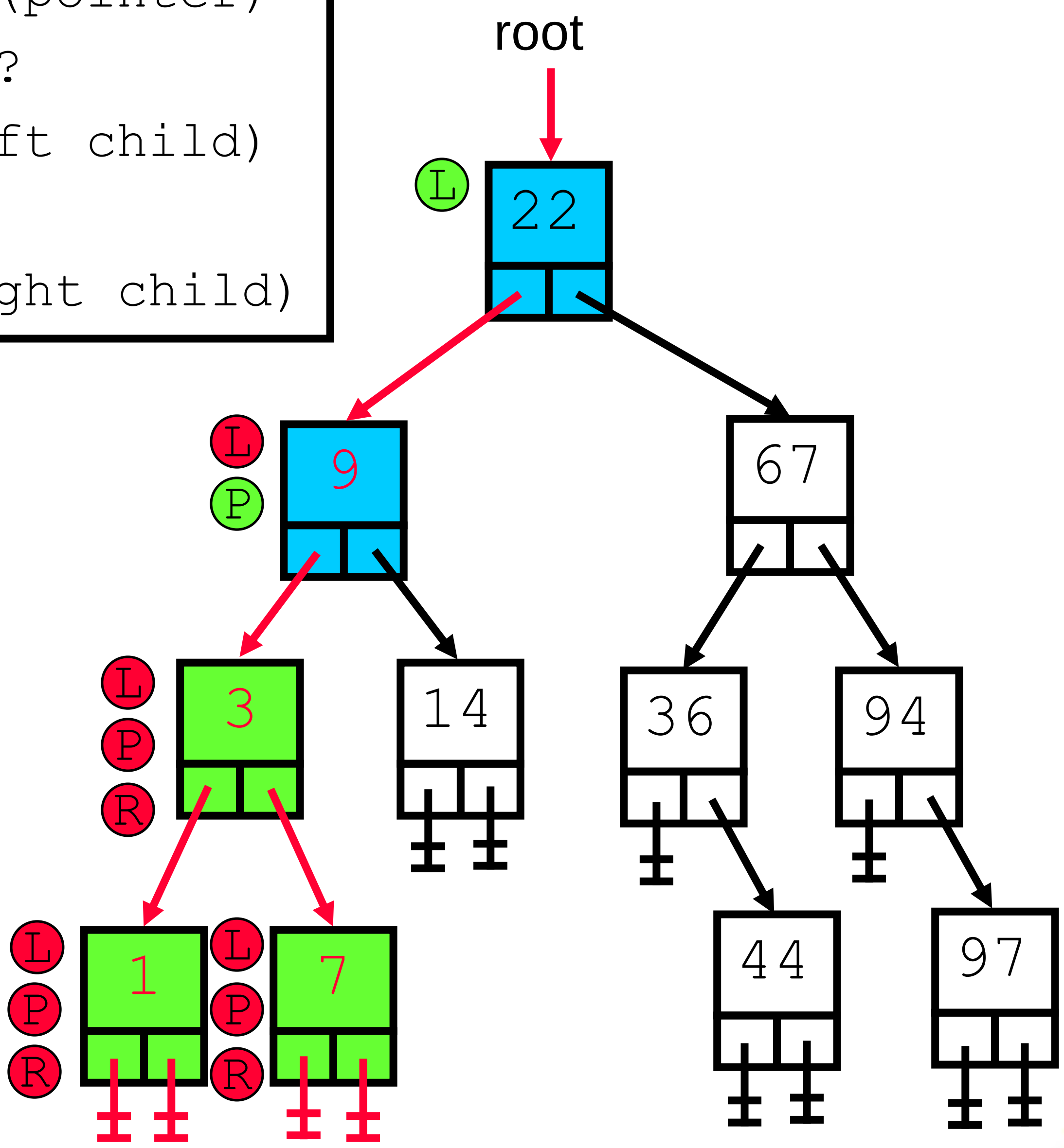
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7
```



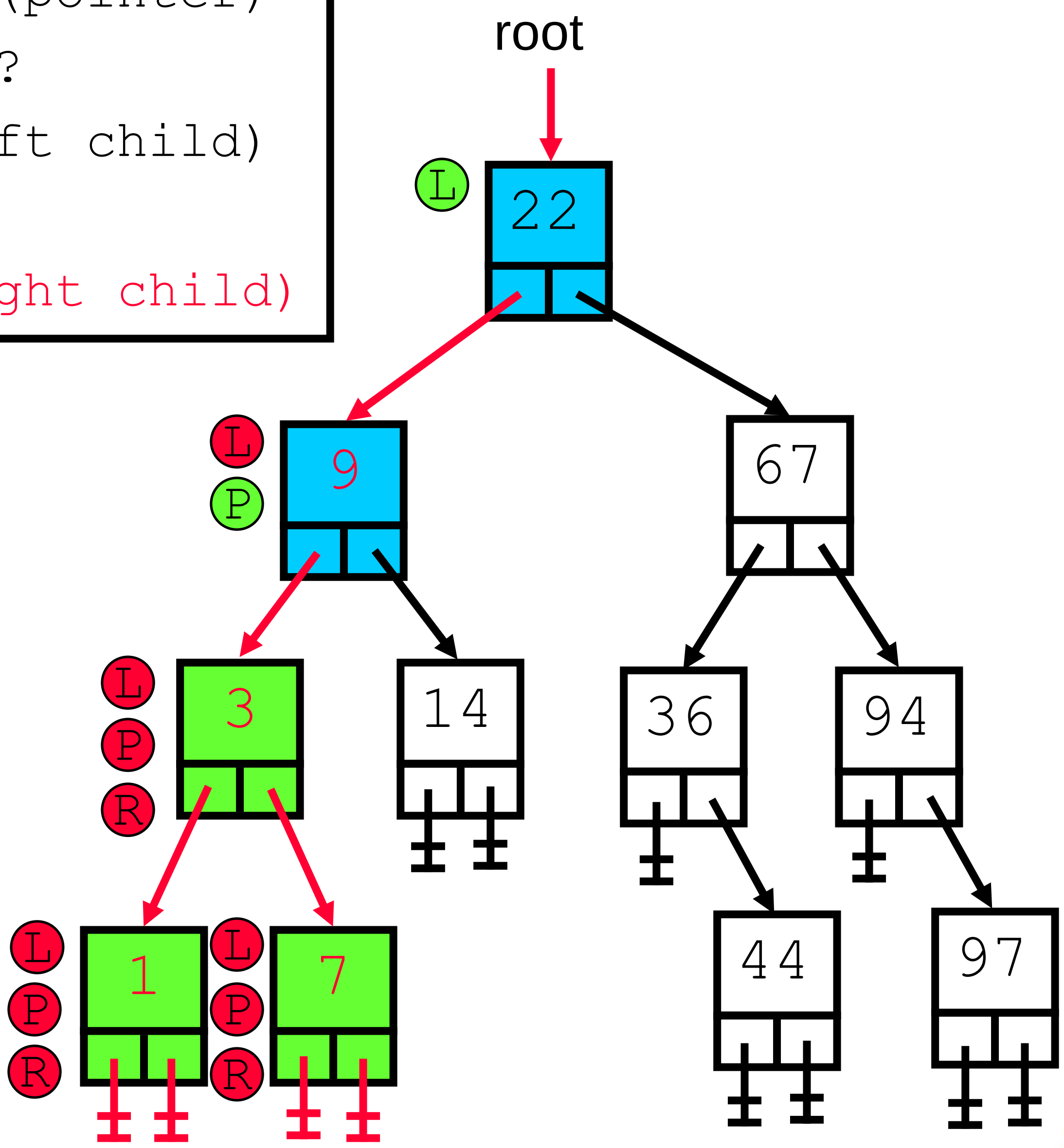
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9
```



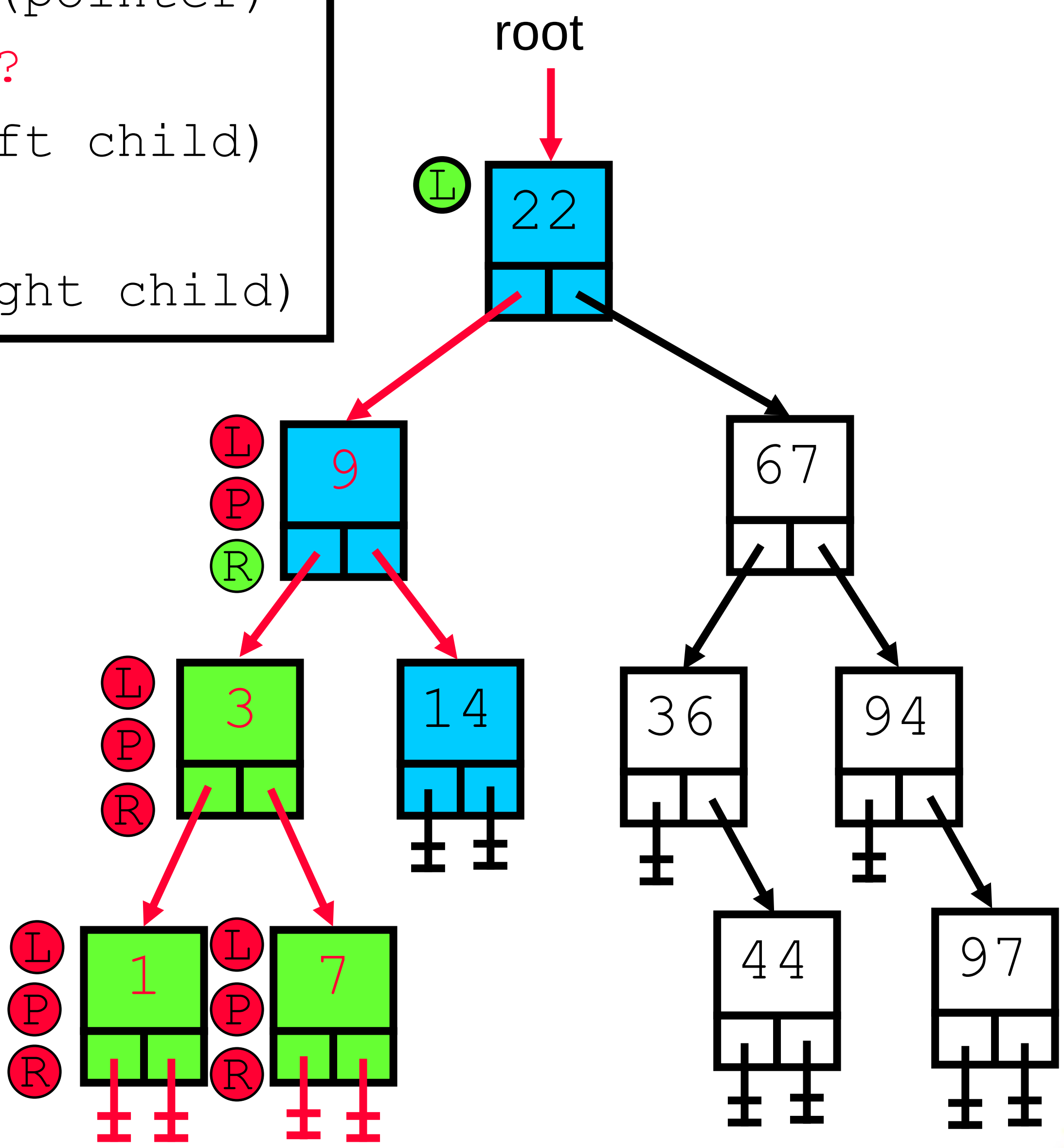
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9
```



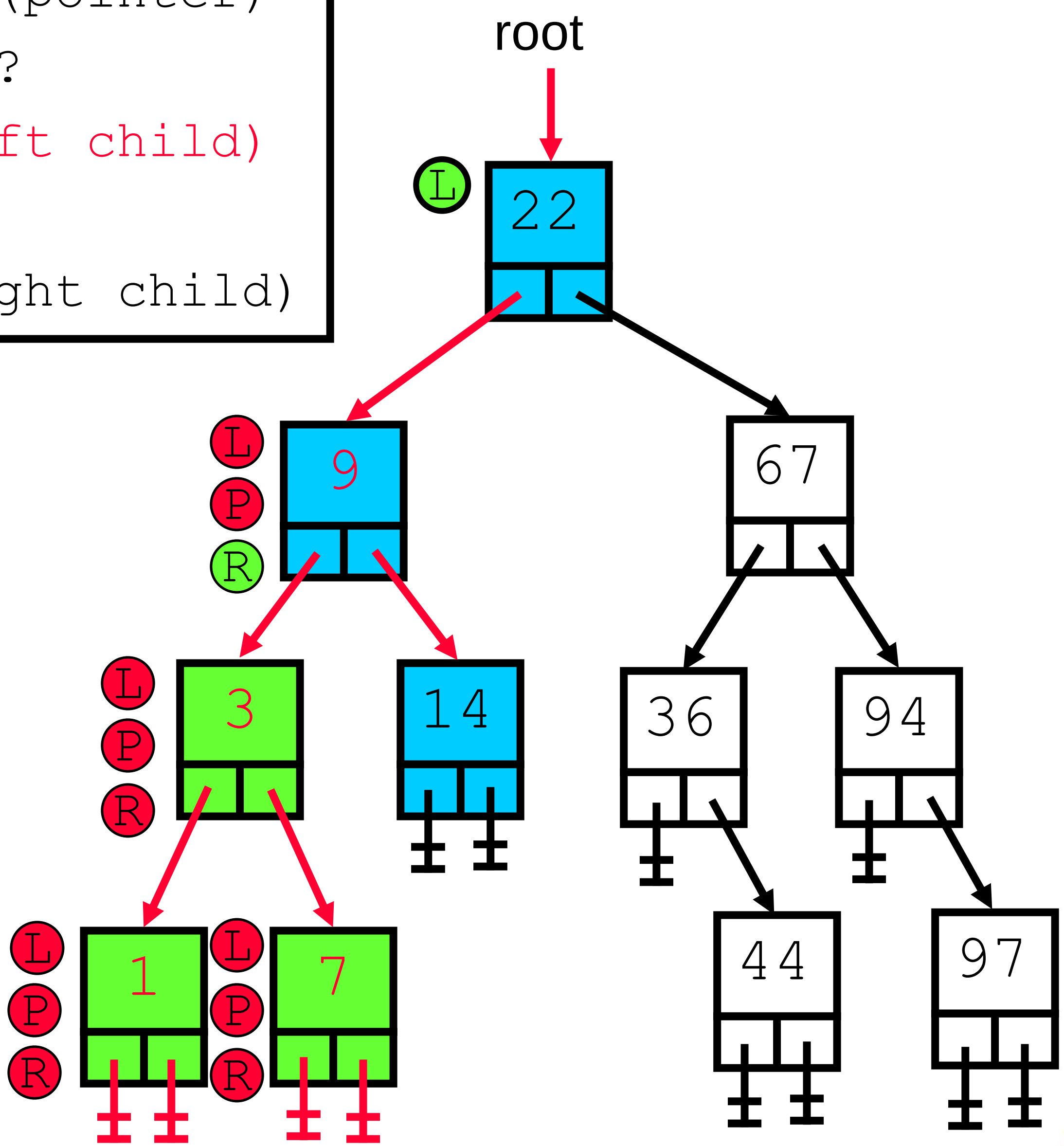
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9
```



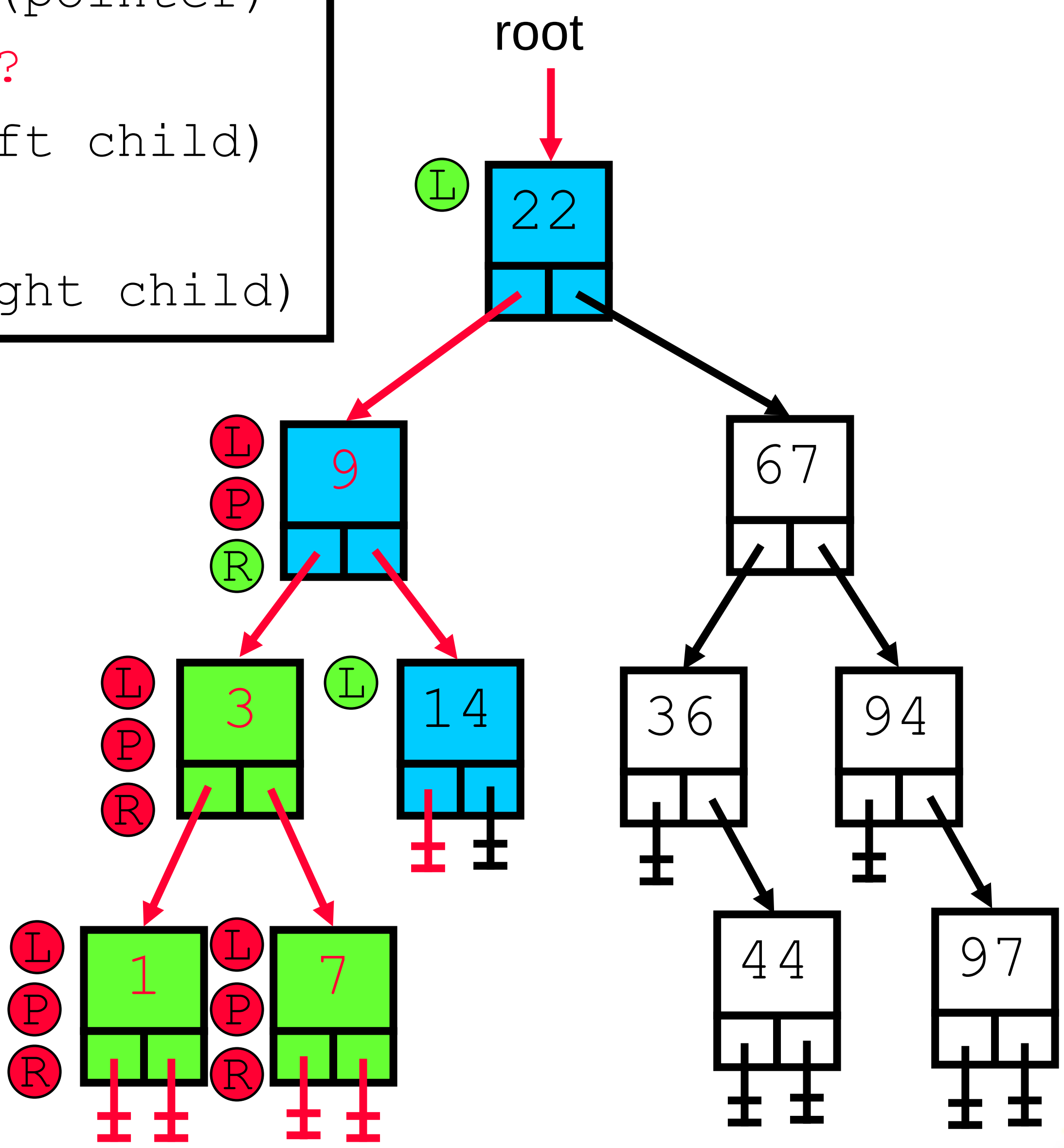
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

Output: 1 3 7 9



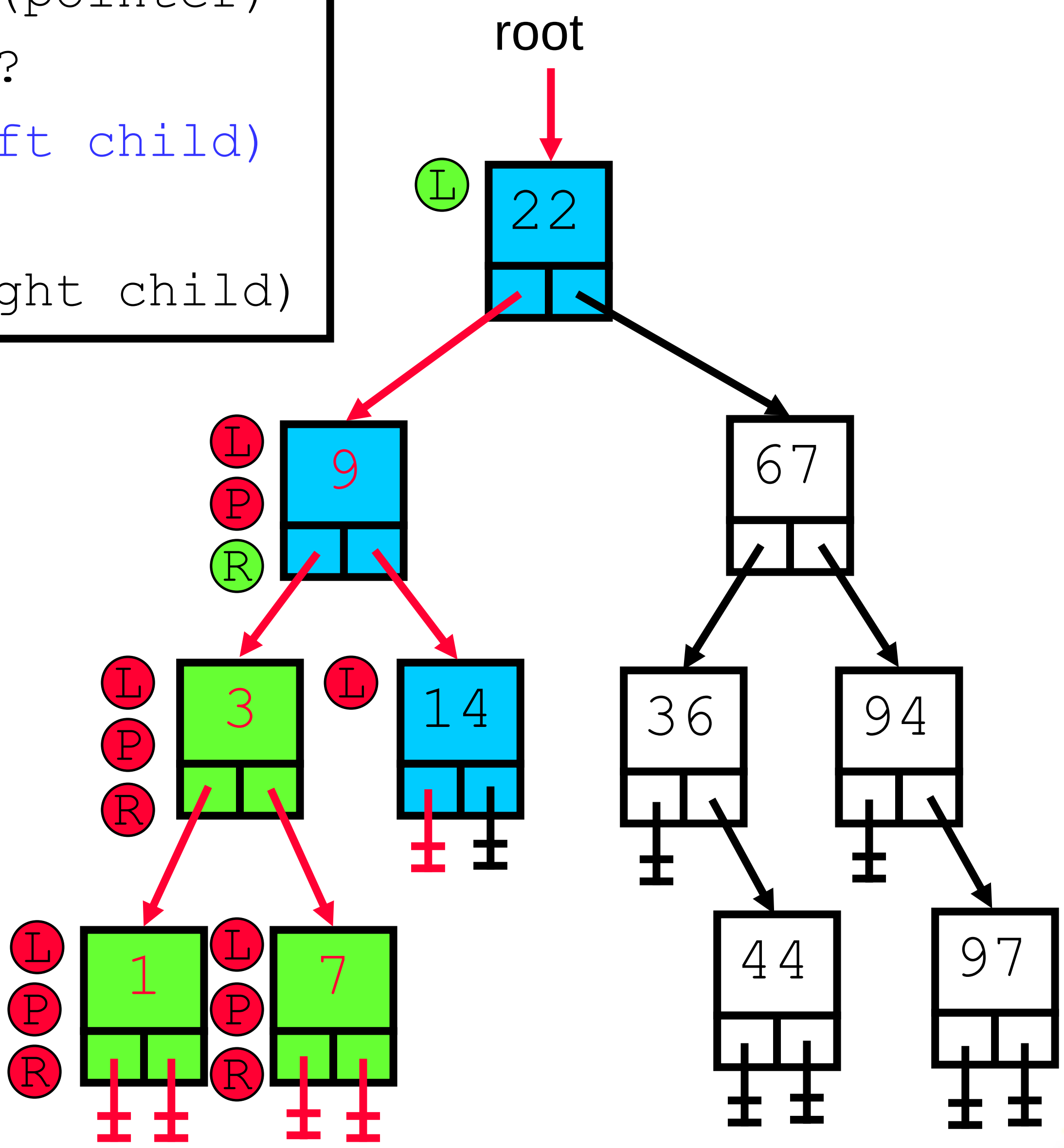
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9
```



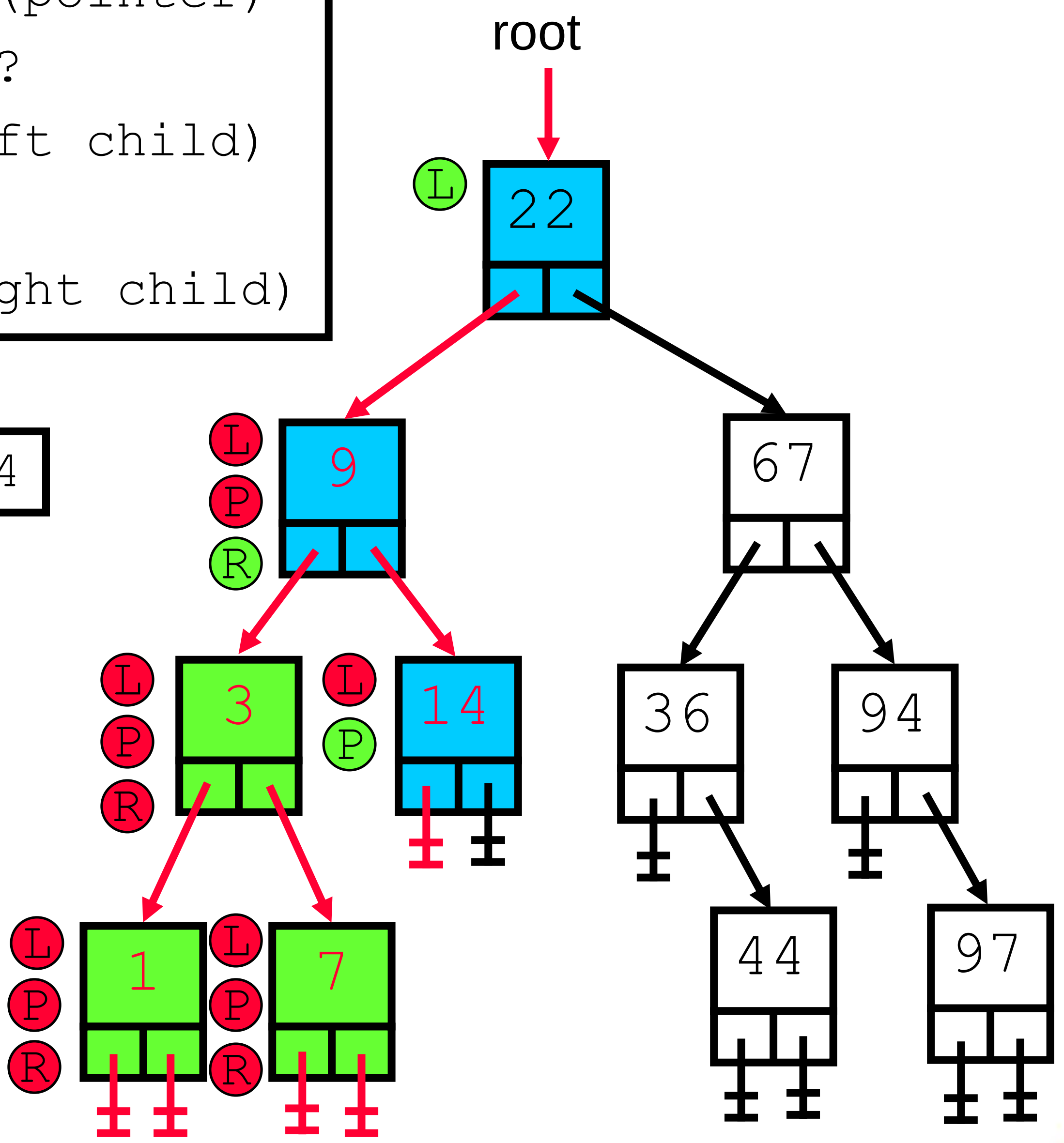
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9
```



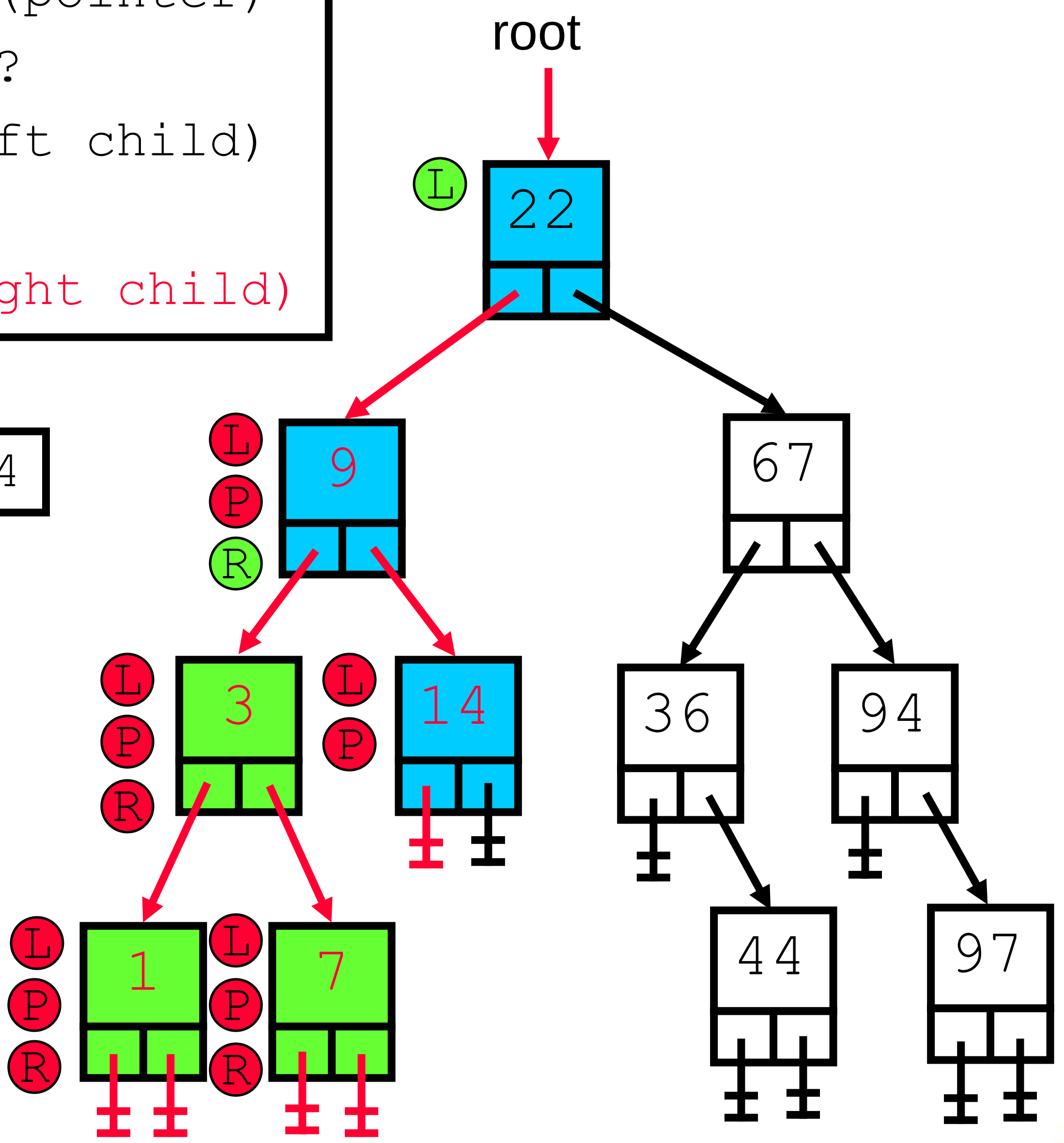
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14
```



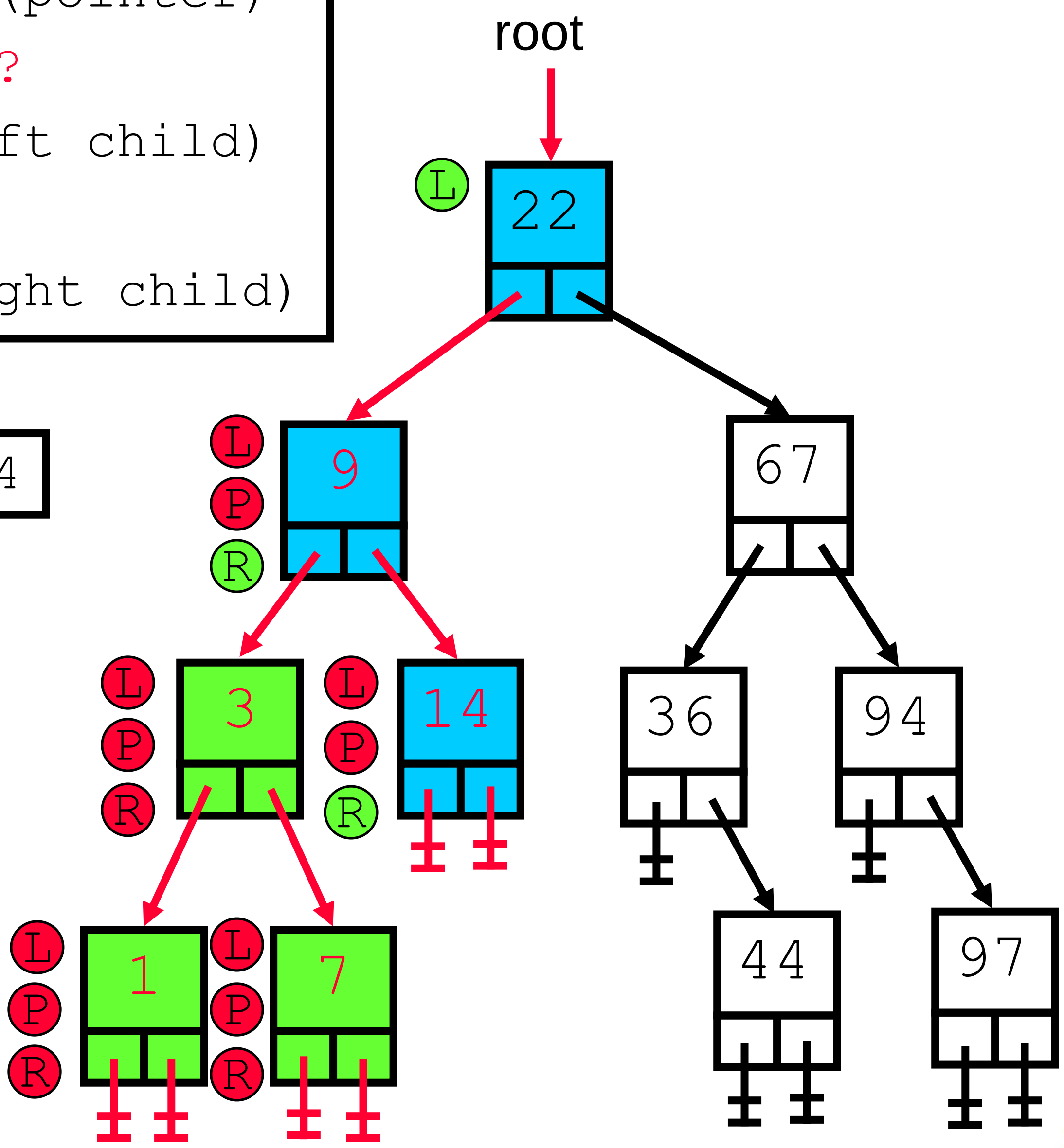
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14
```



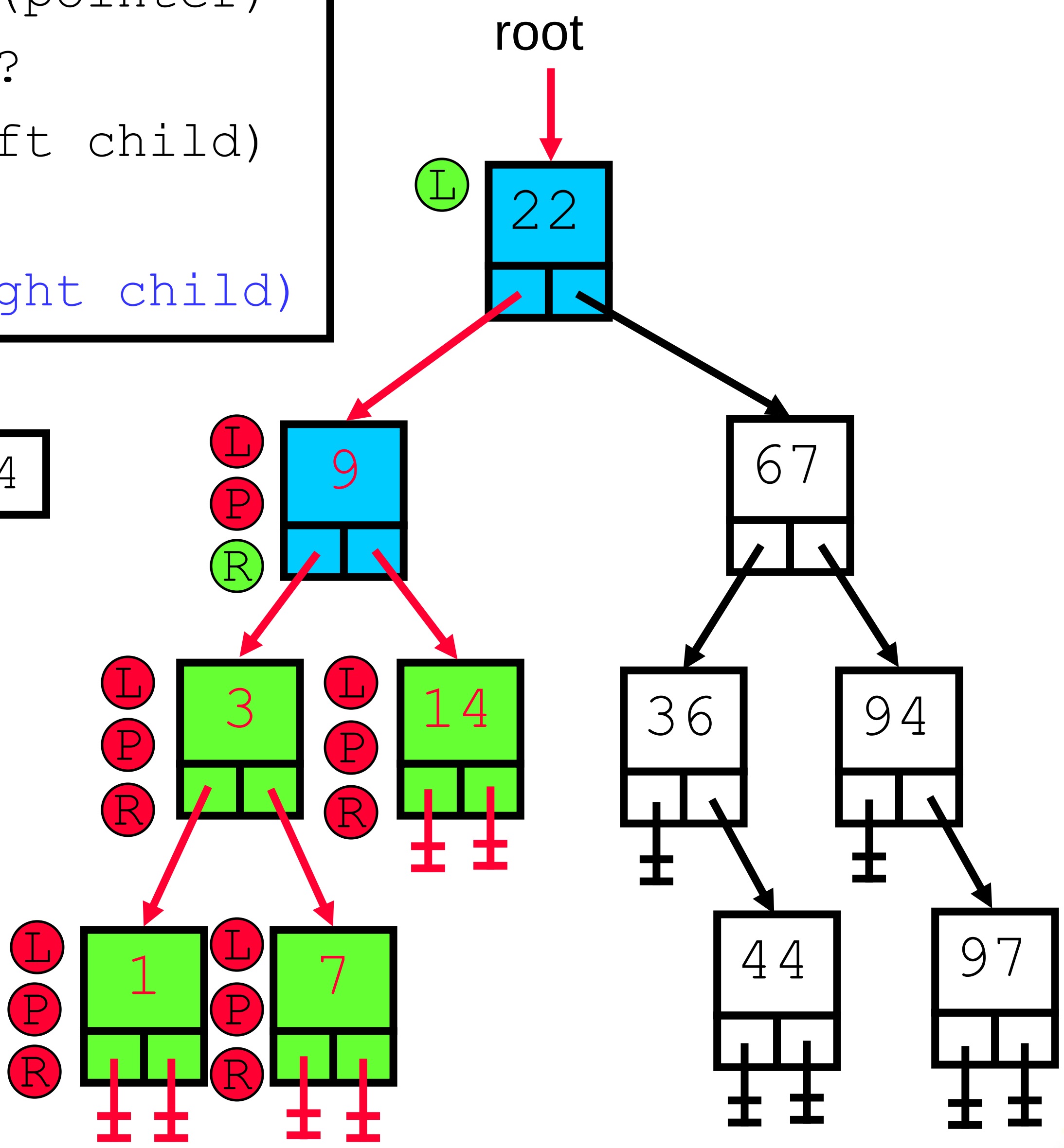
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14
```



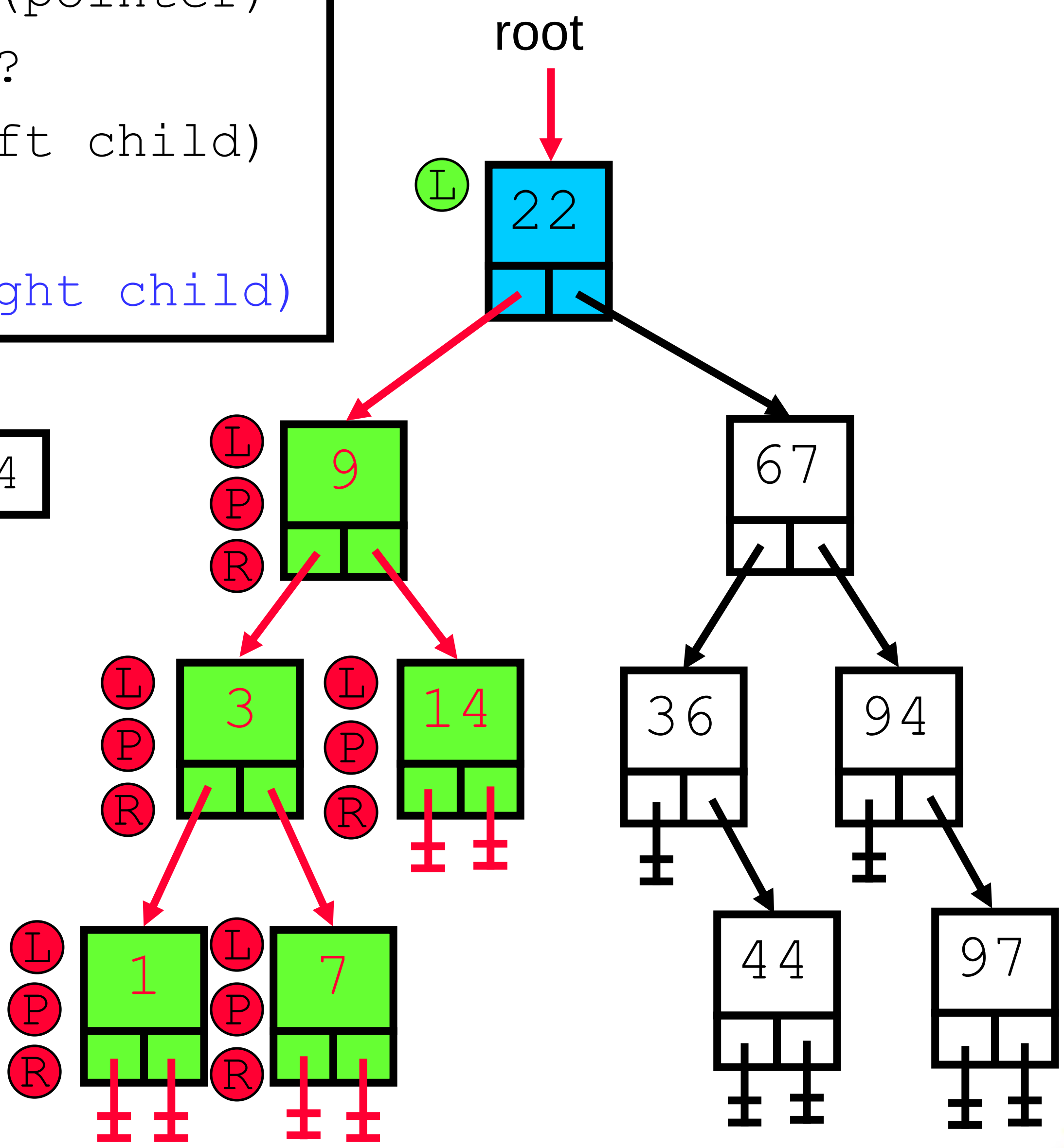
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14
```



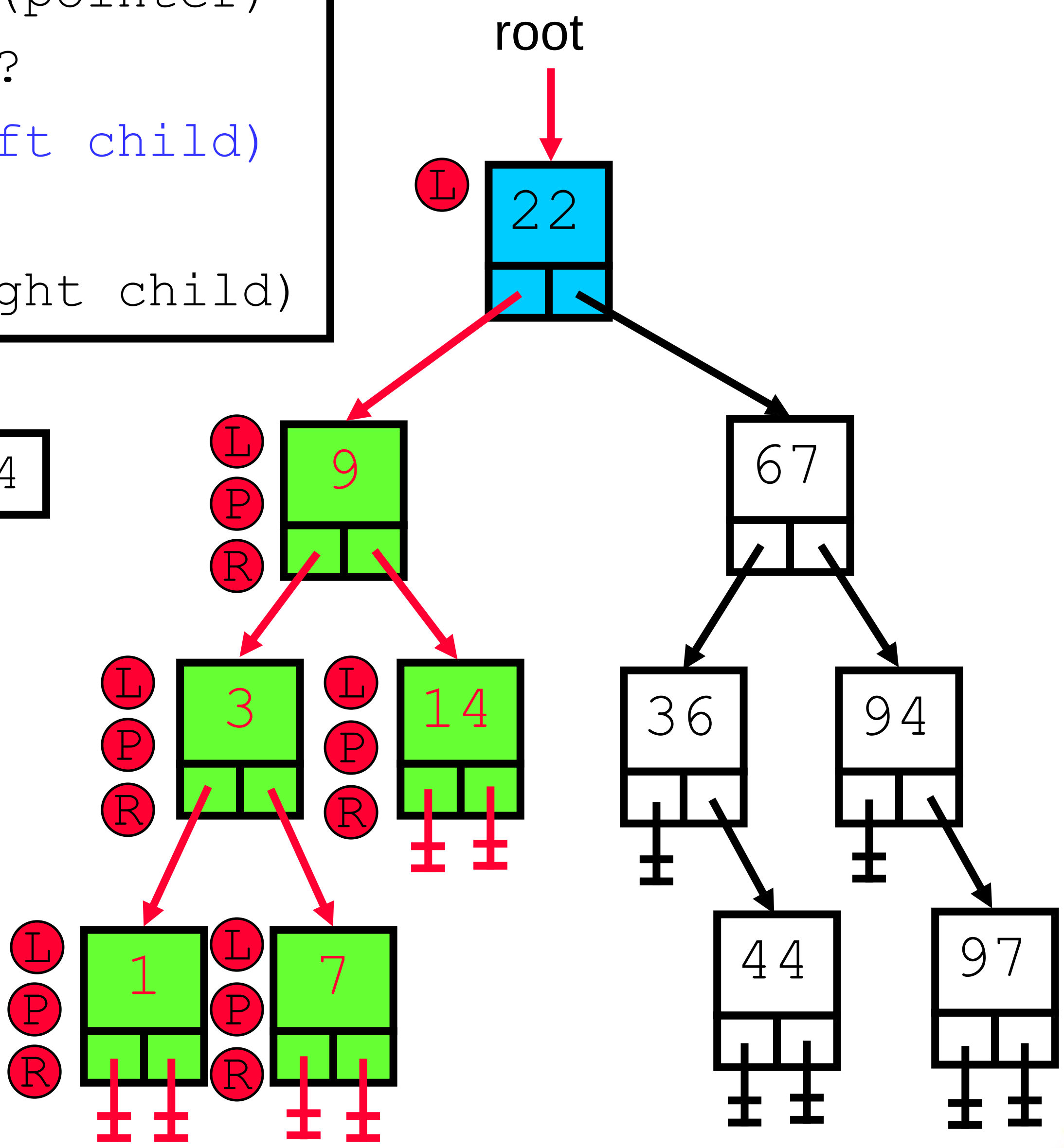
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14
```



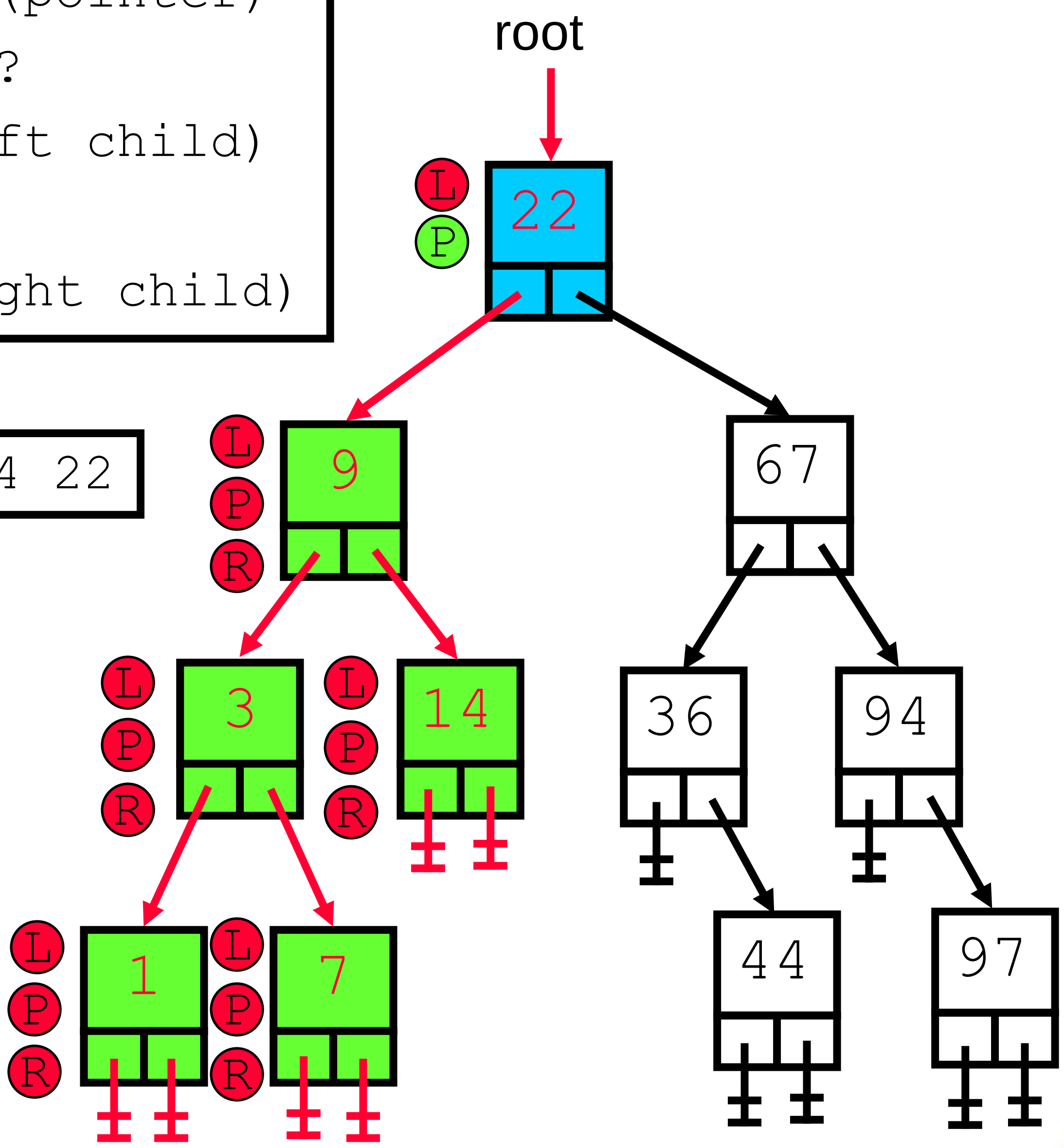
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14
```



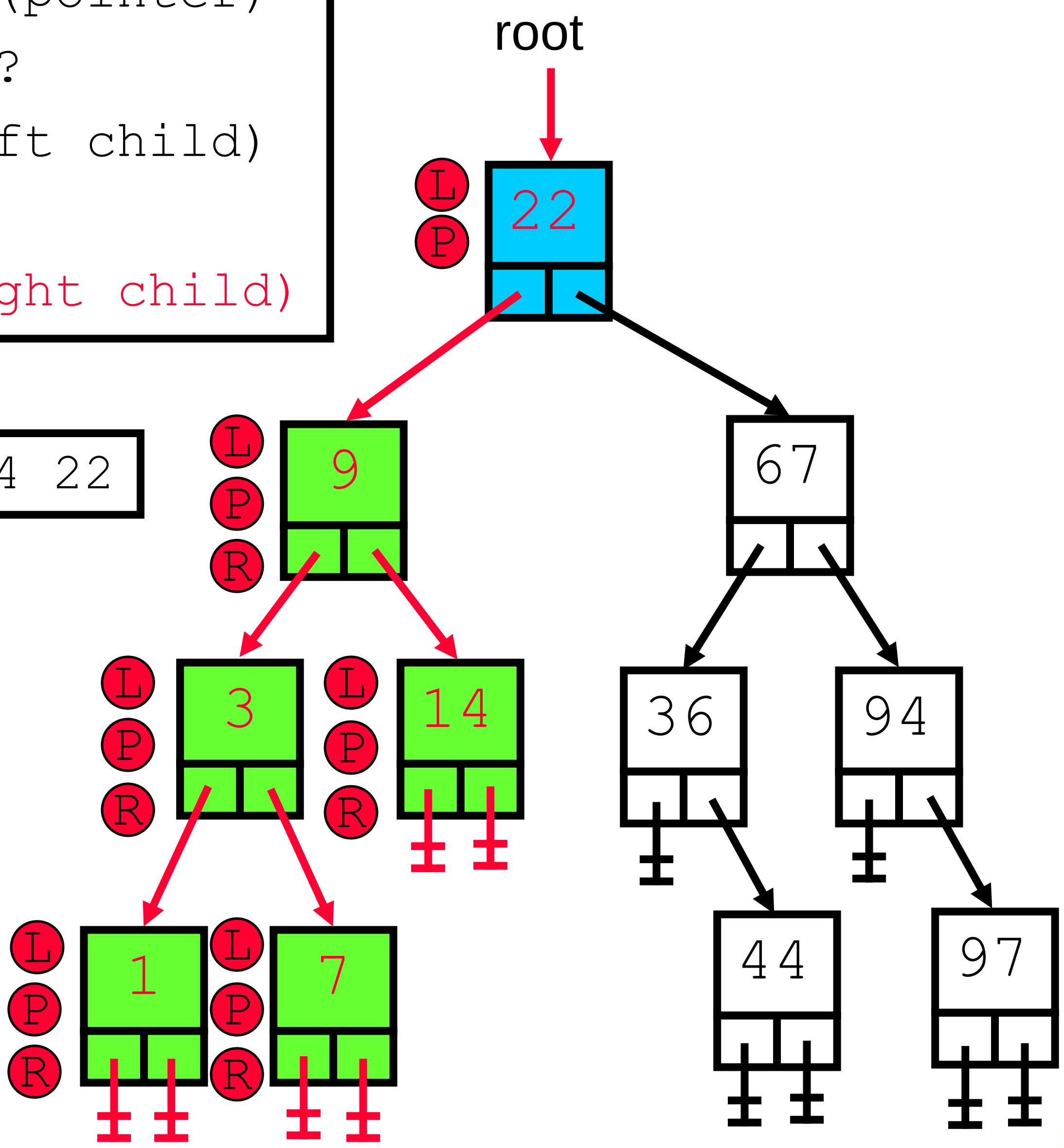
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
```



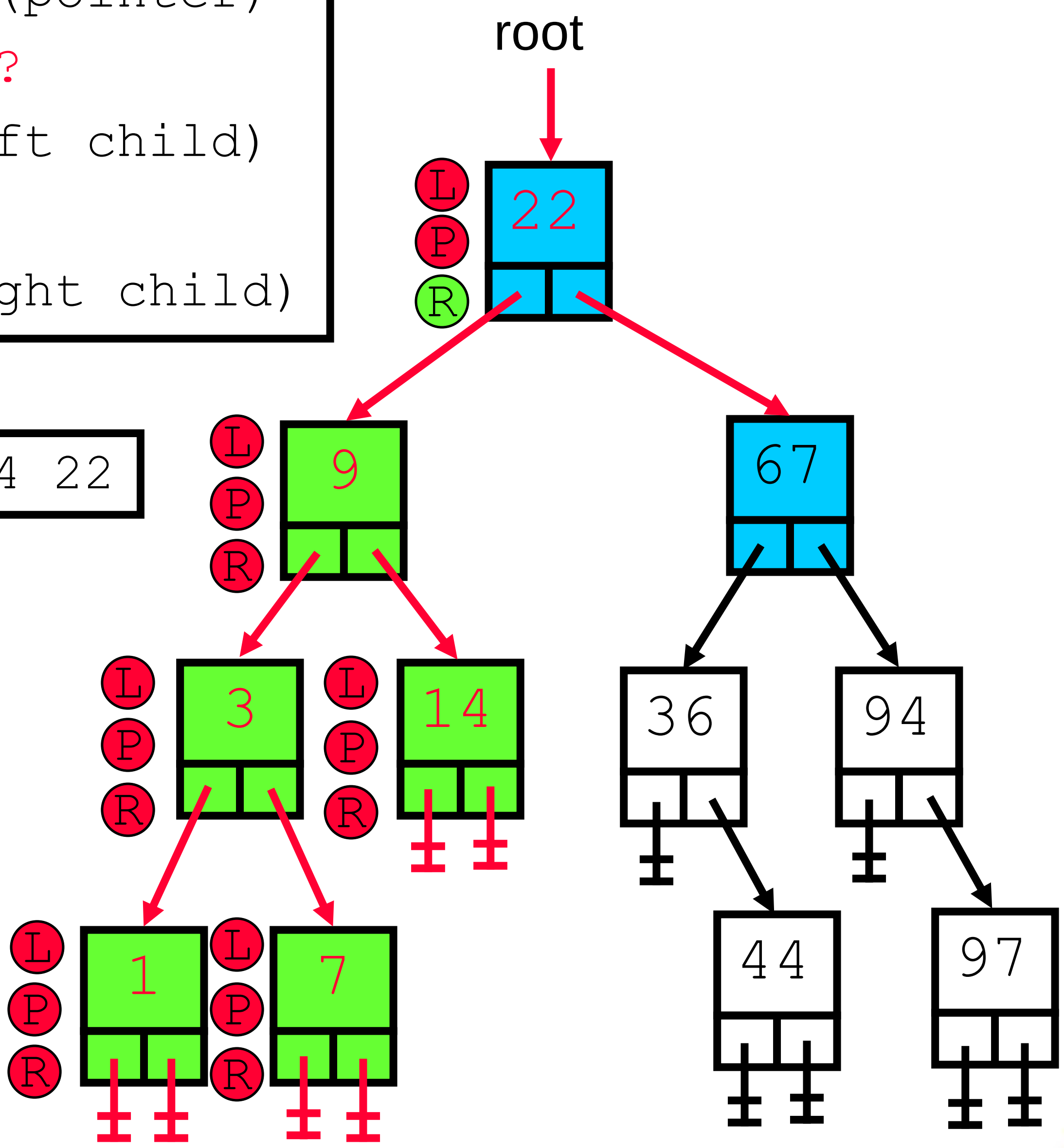
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
```



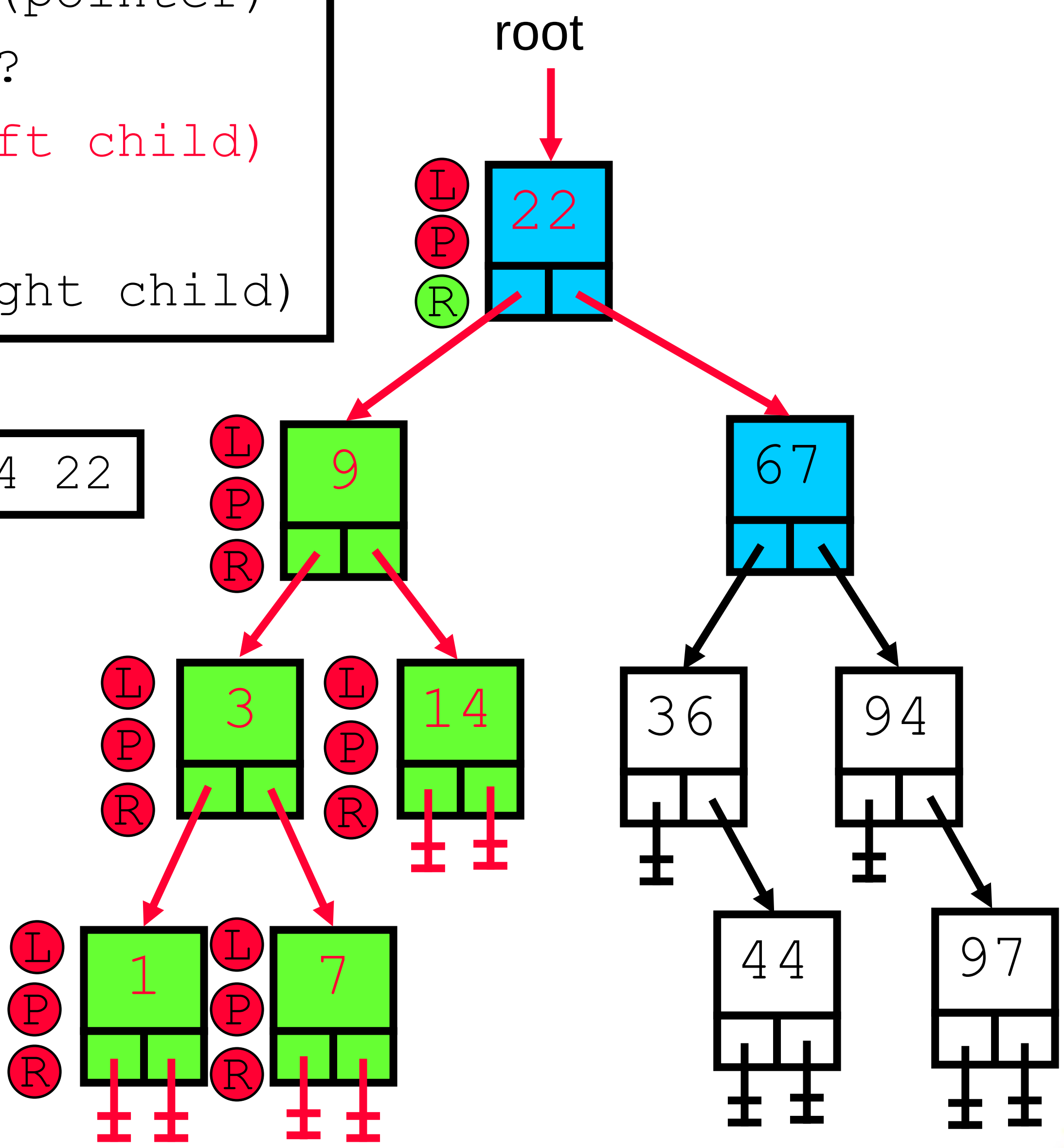
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
```



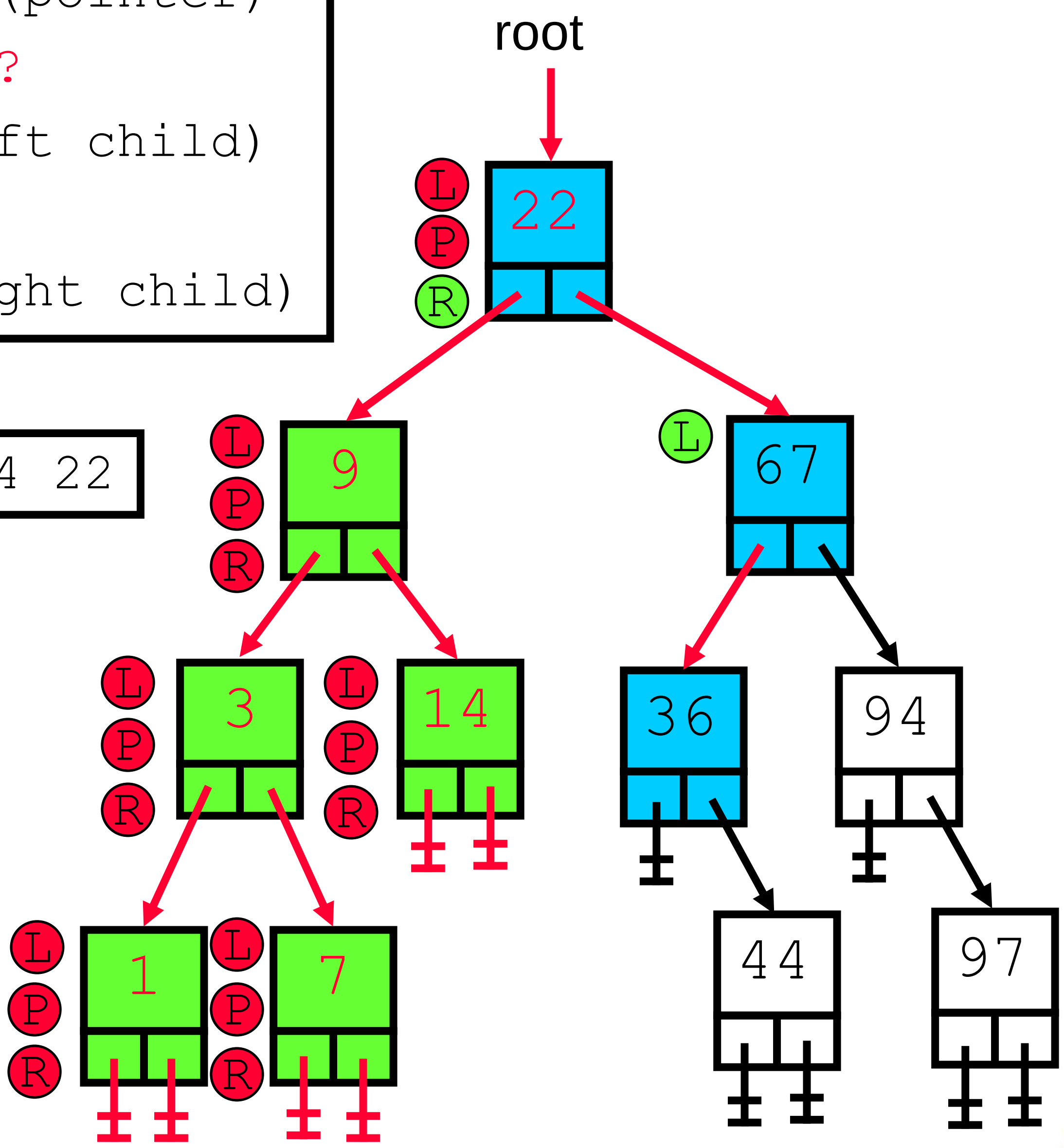
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
```



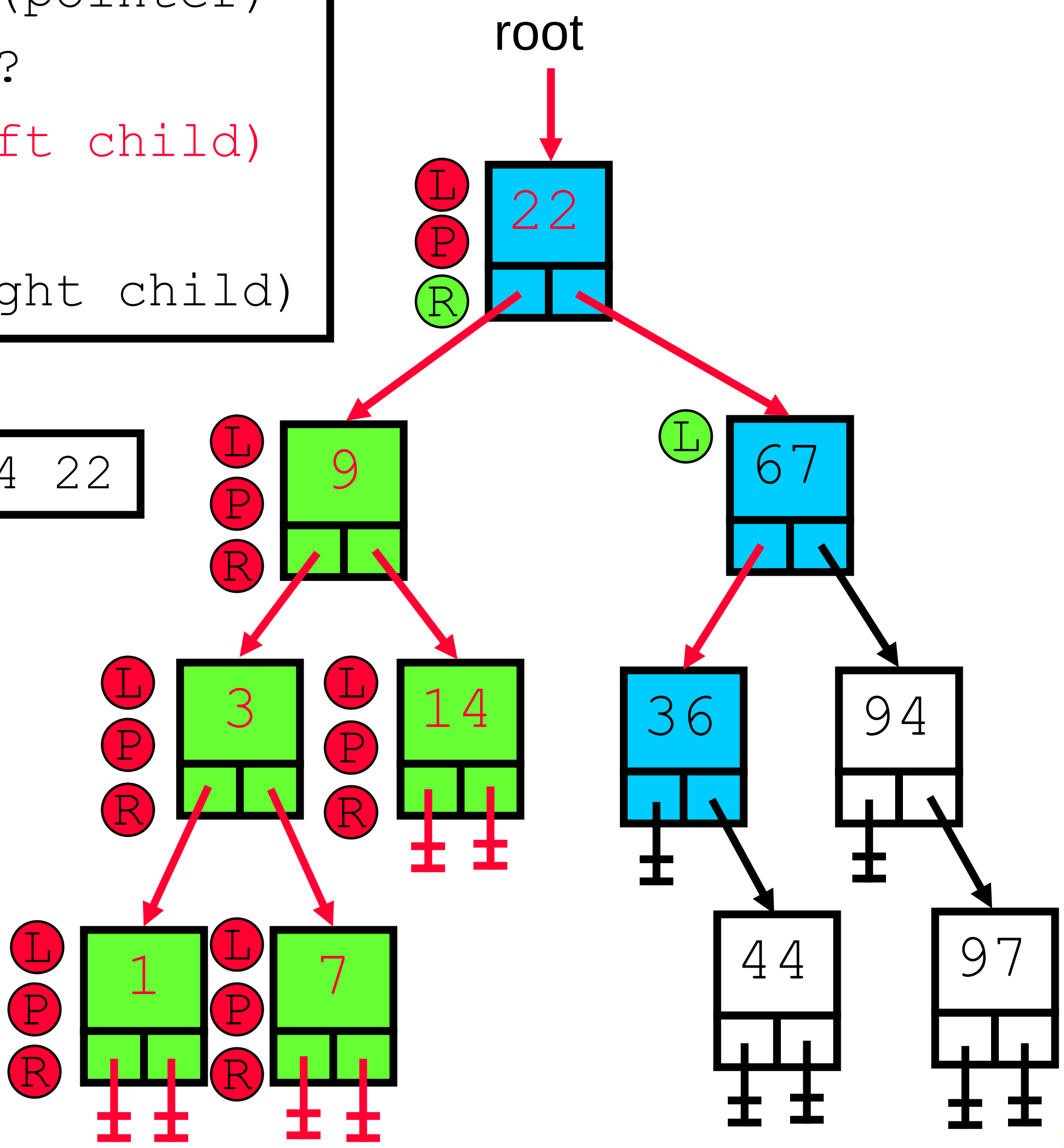
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
```



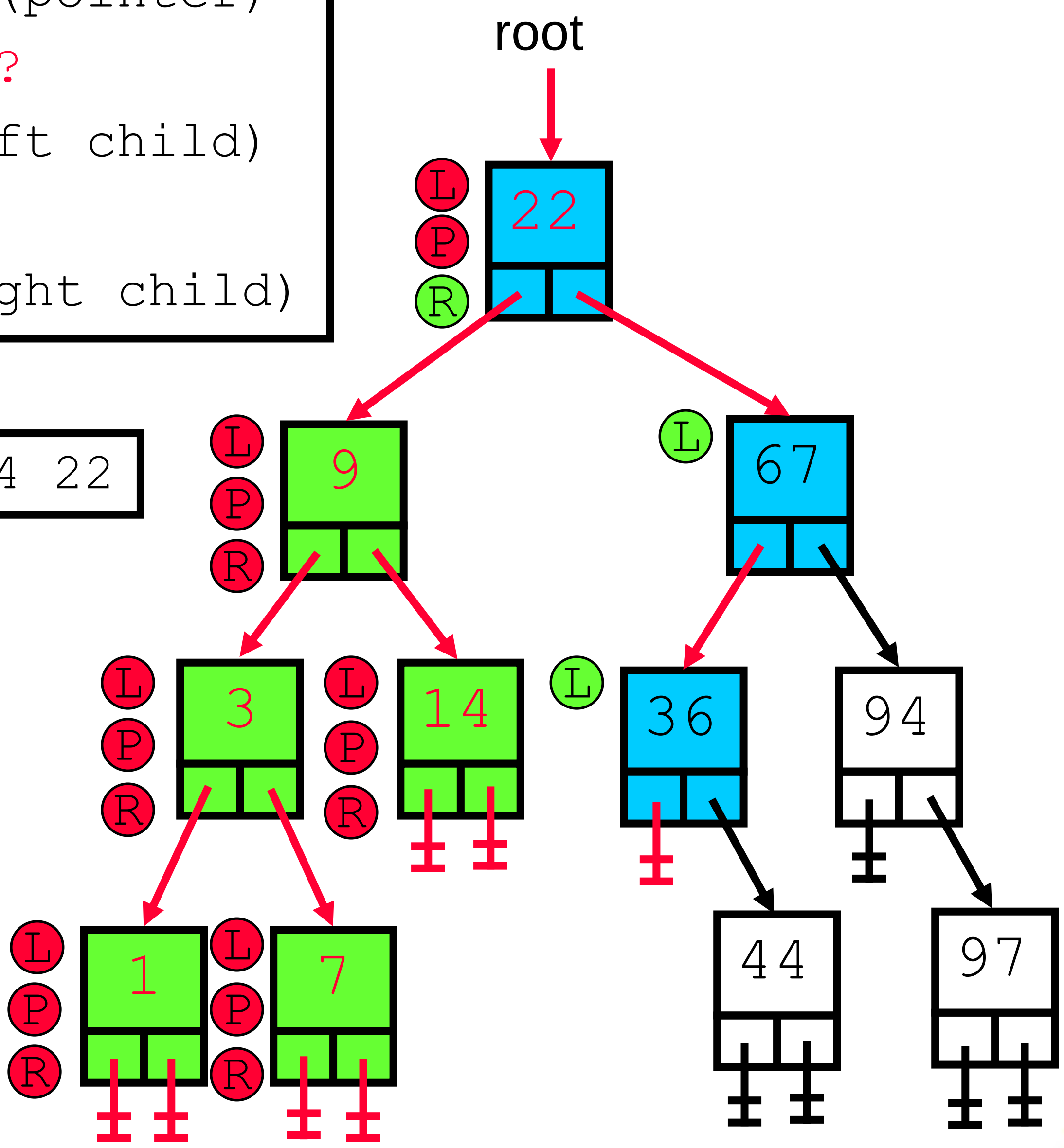
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
```



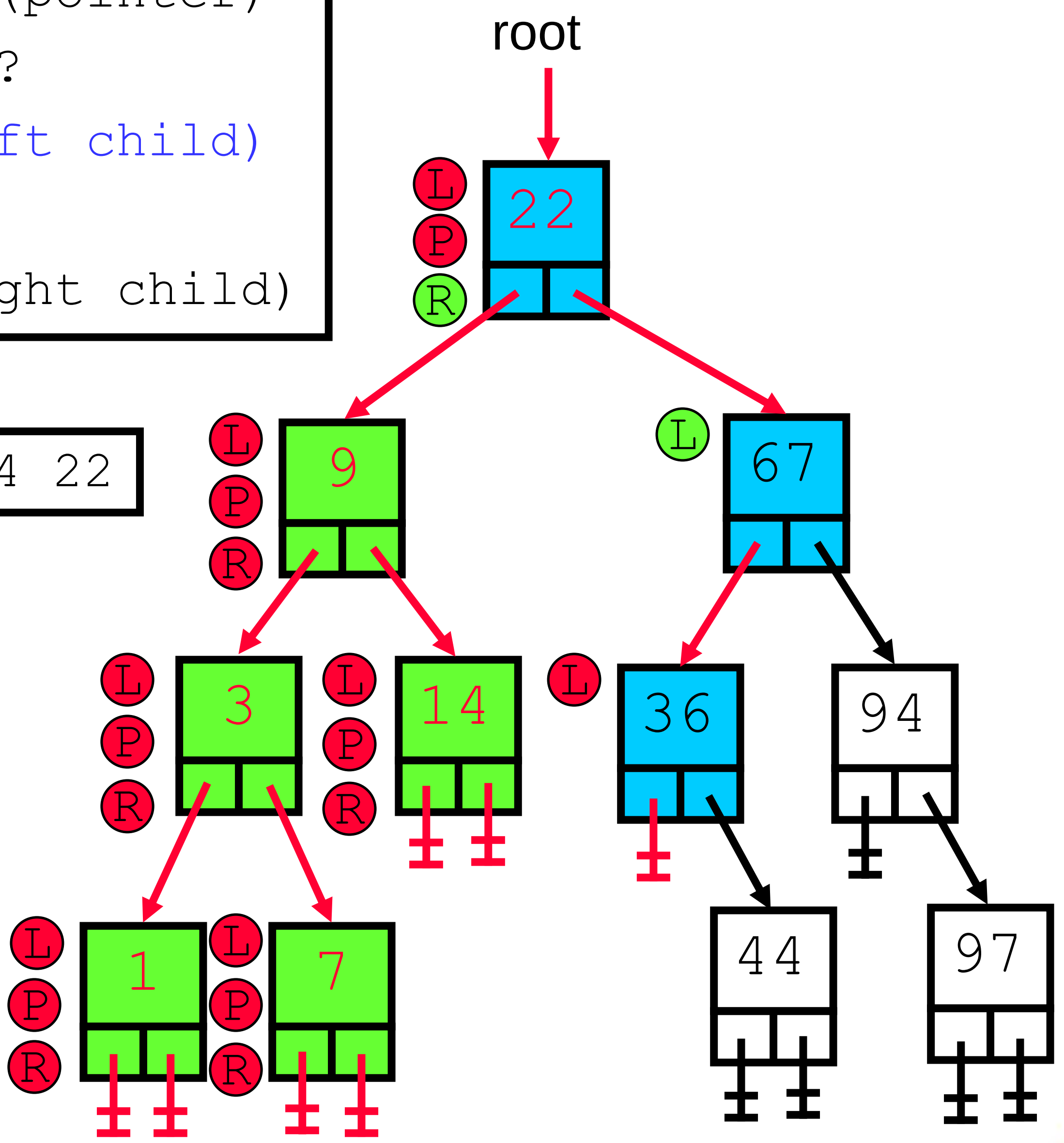
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
```



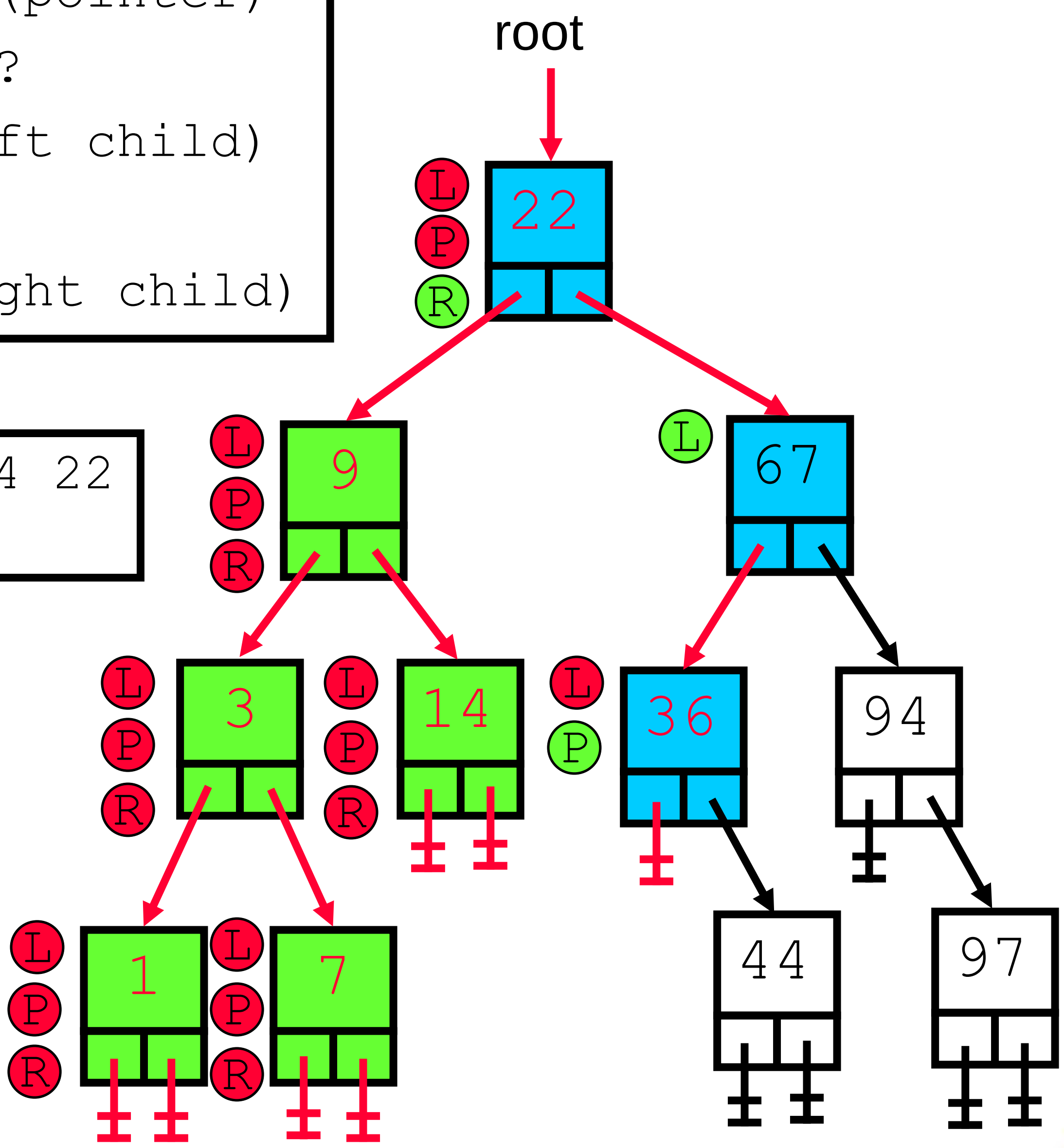
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
```



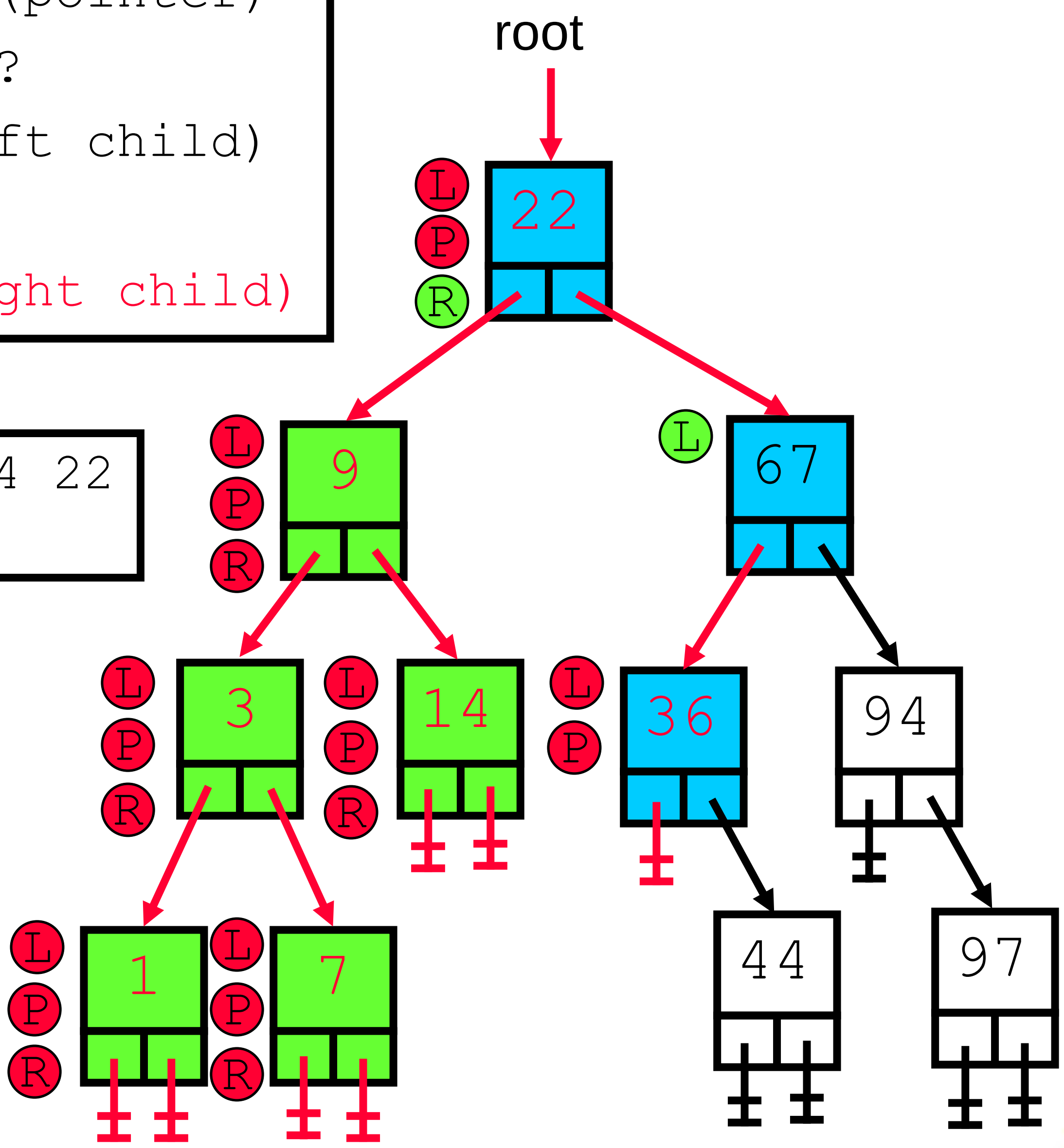
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
        36
```



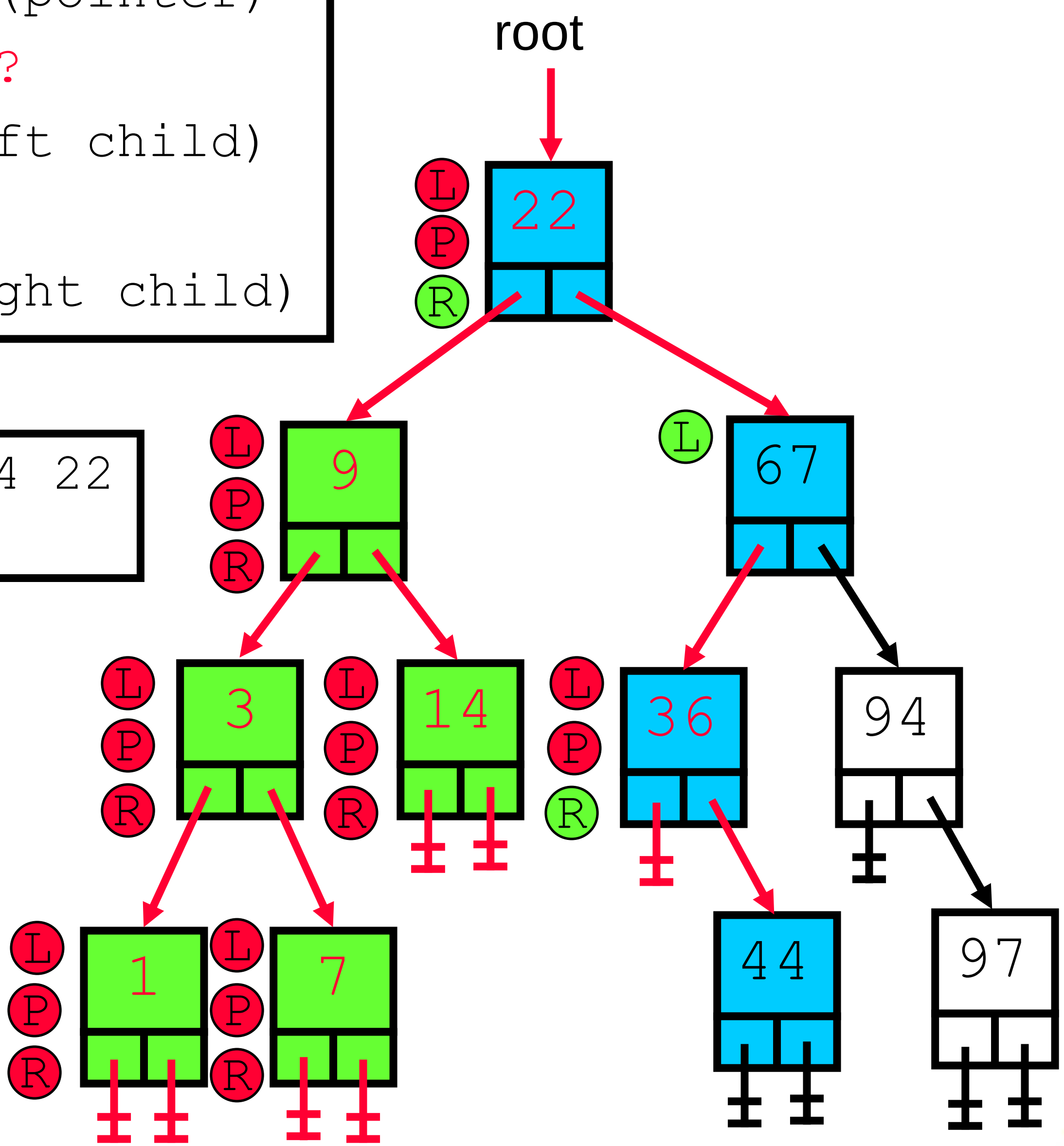
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

Output: 1 3 7 9 14 22 36



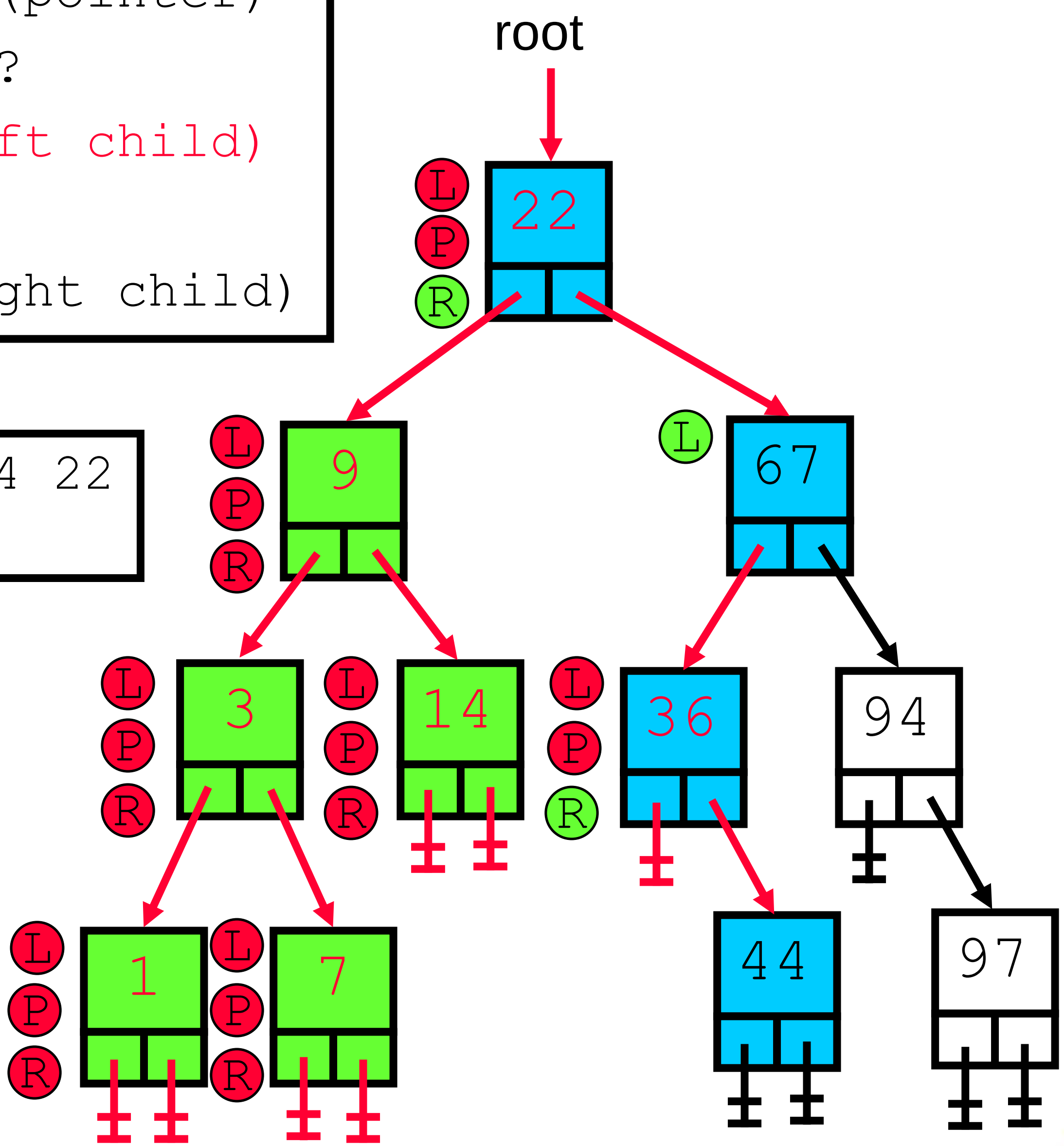
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
        36
```



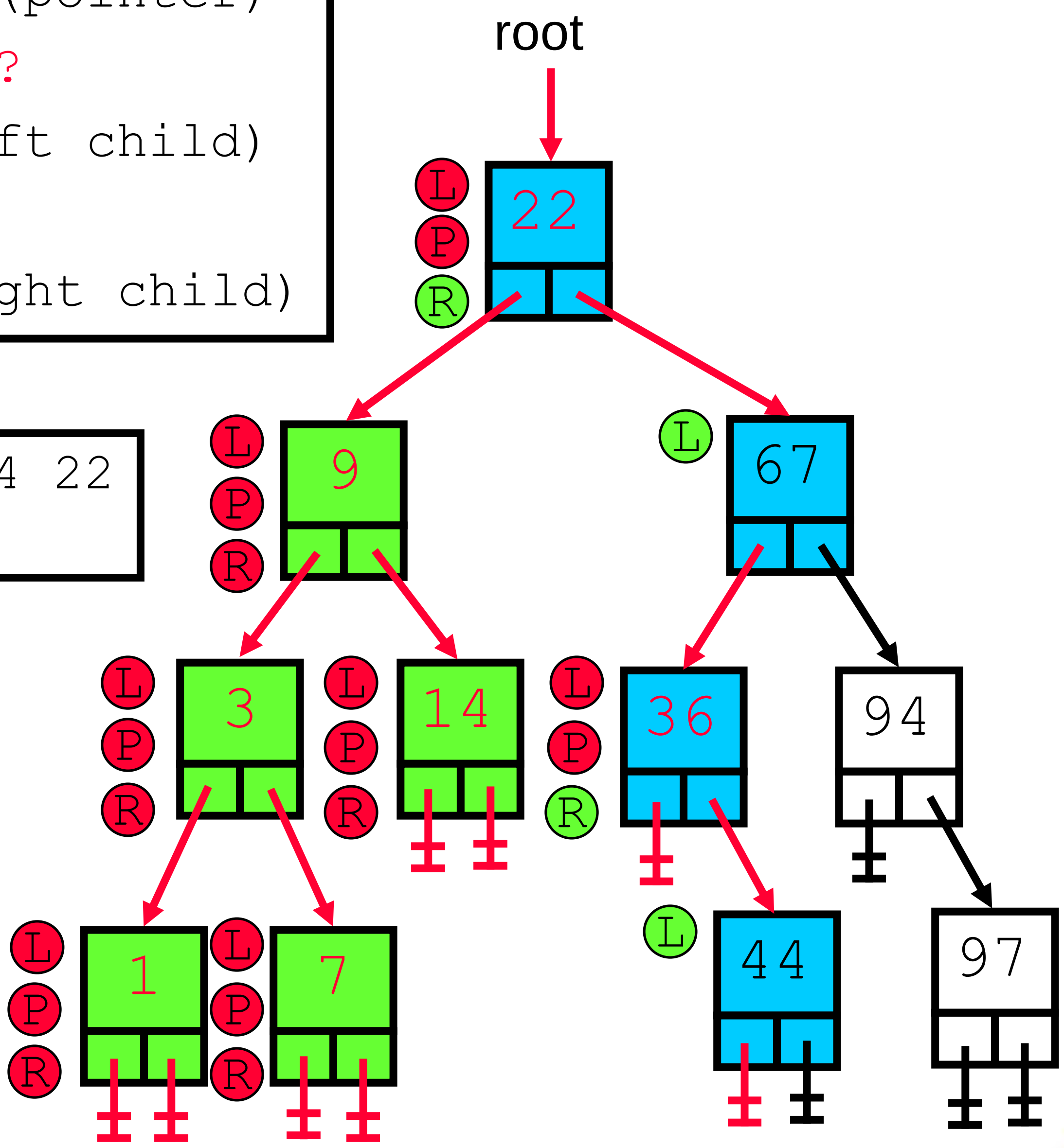
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
        36
```



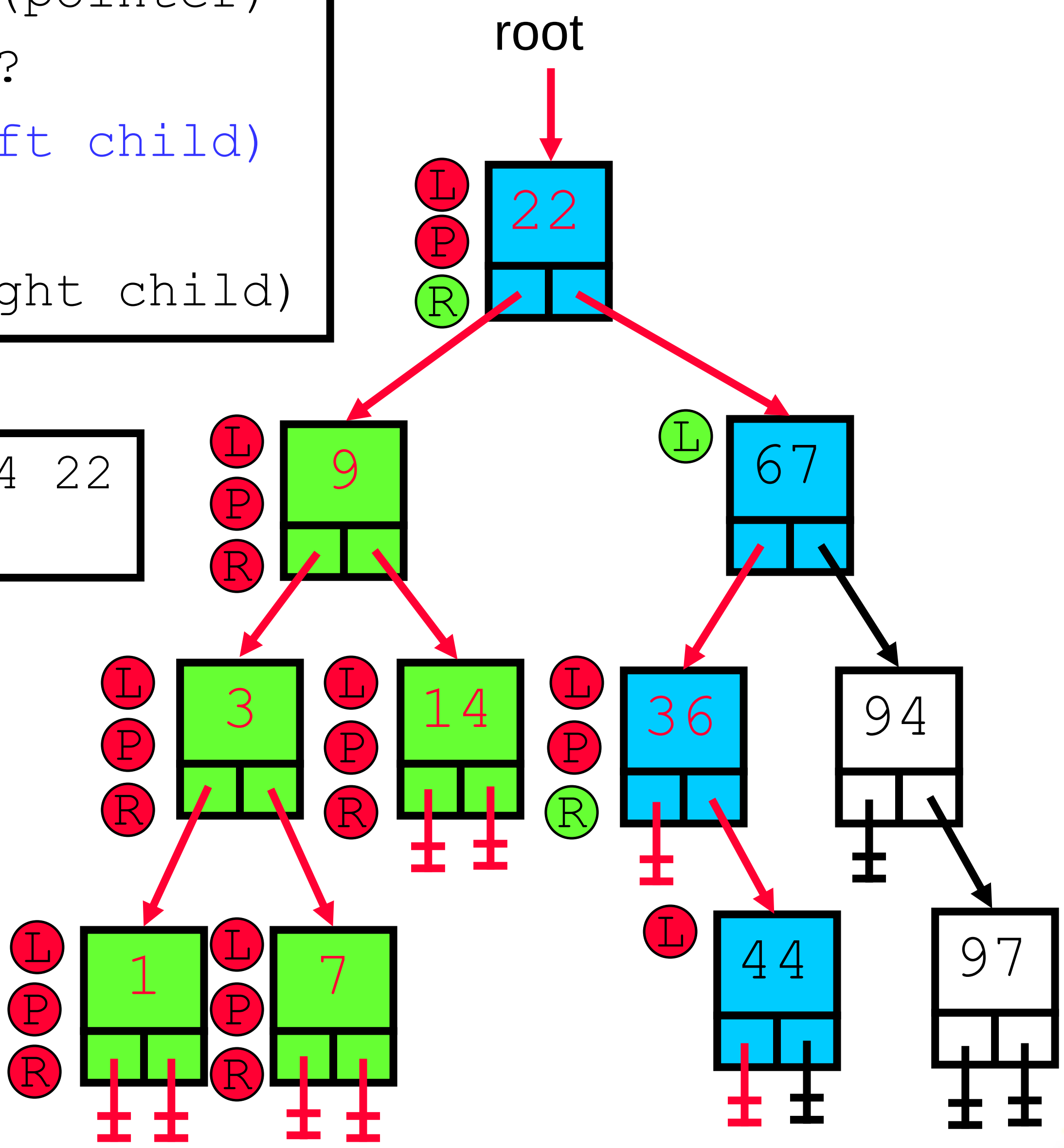
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
        36
```



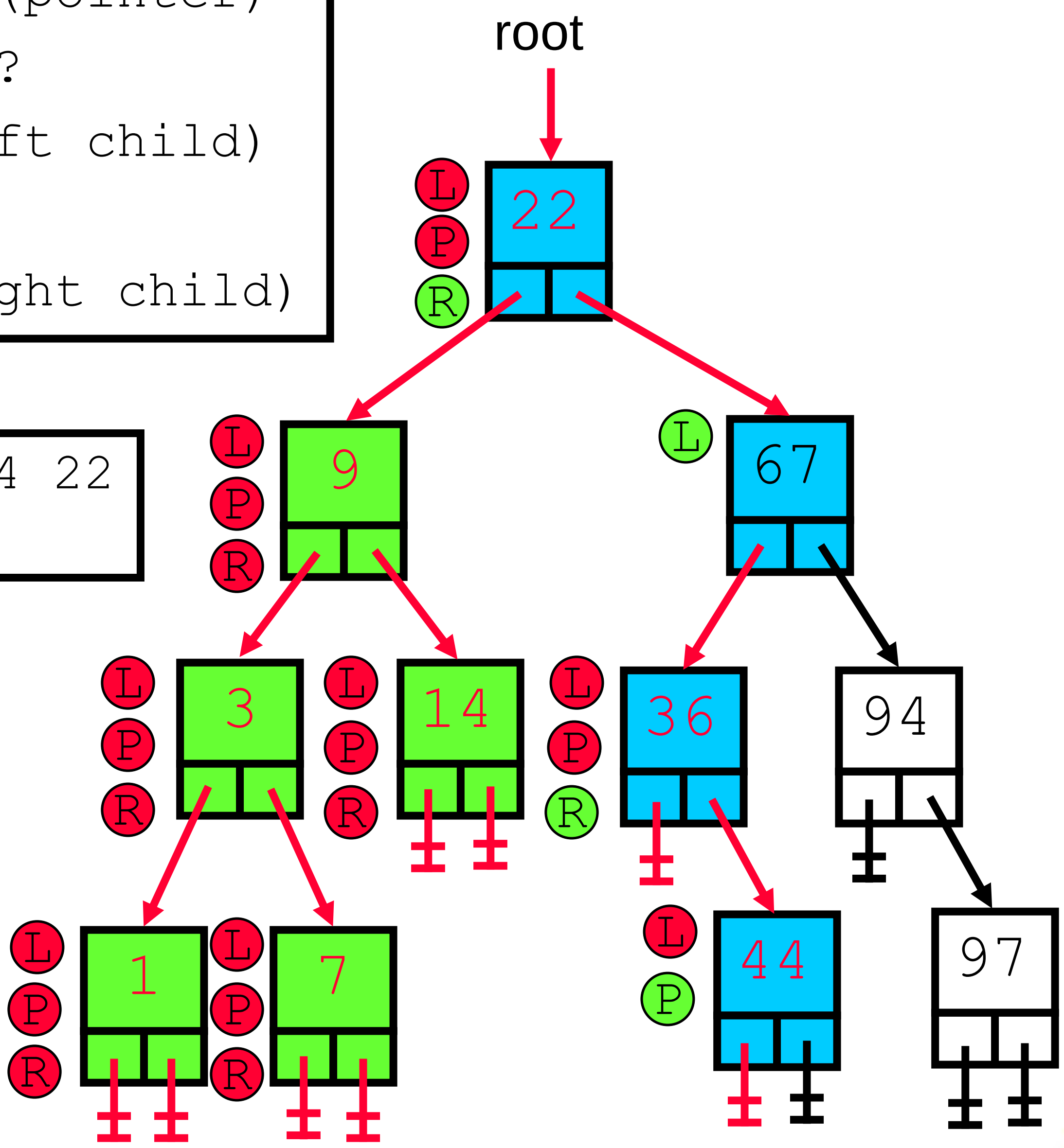
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
        36
```



```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
(L) InOrderPrint(left child)
(P) print(data)
(R) InOrderPrint(right child)
```

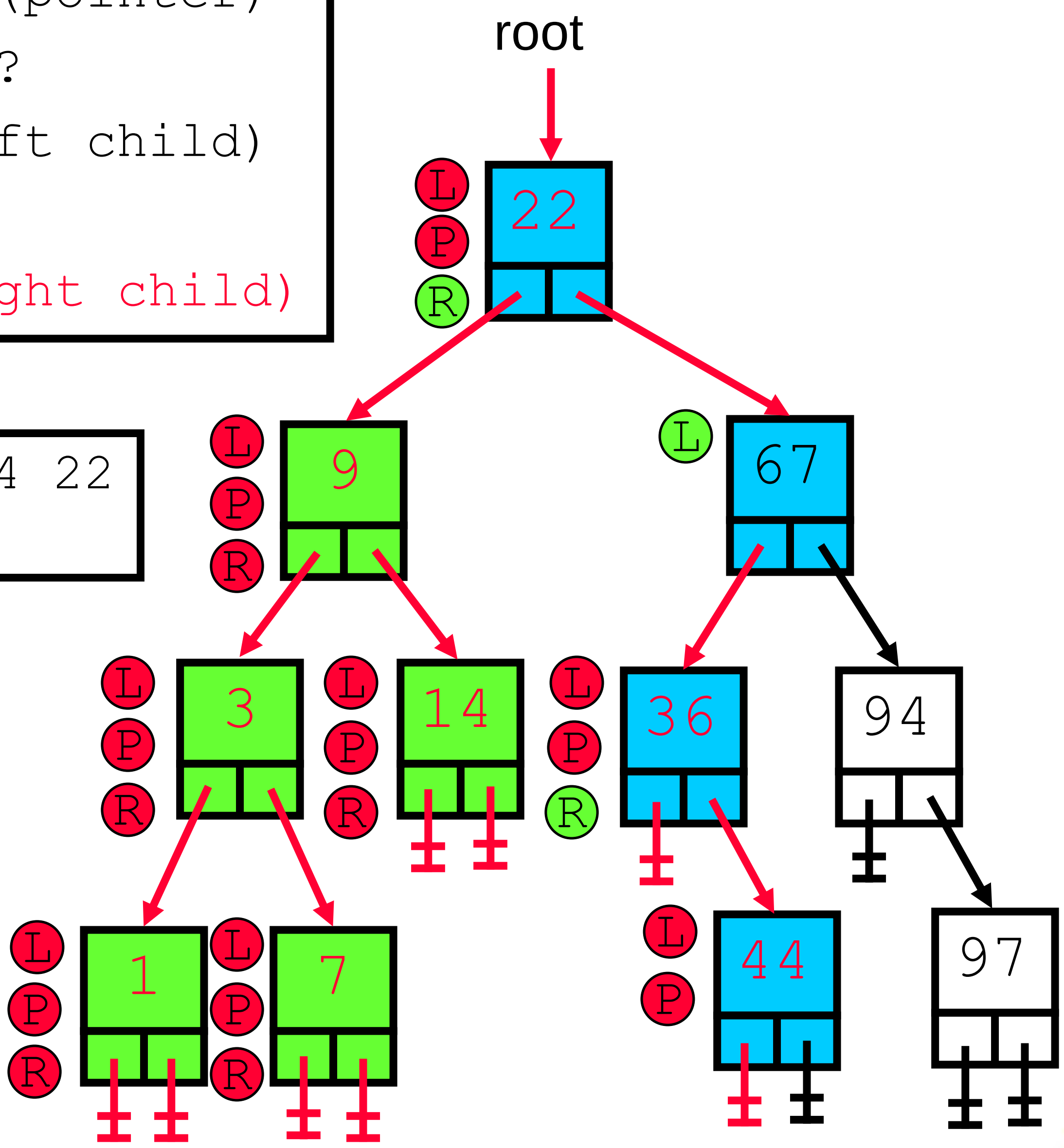
Output: 1 3 7 9 14 22 36 44



UNIVERSITY OF CALGARY

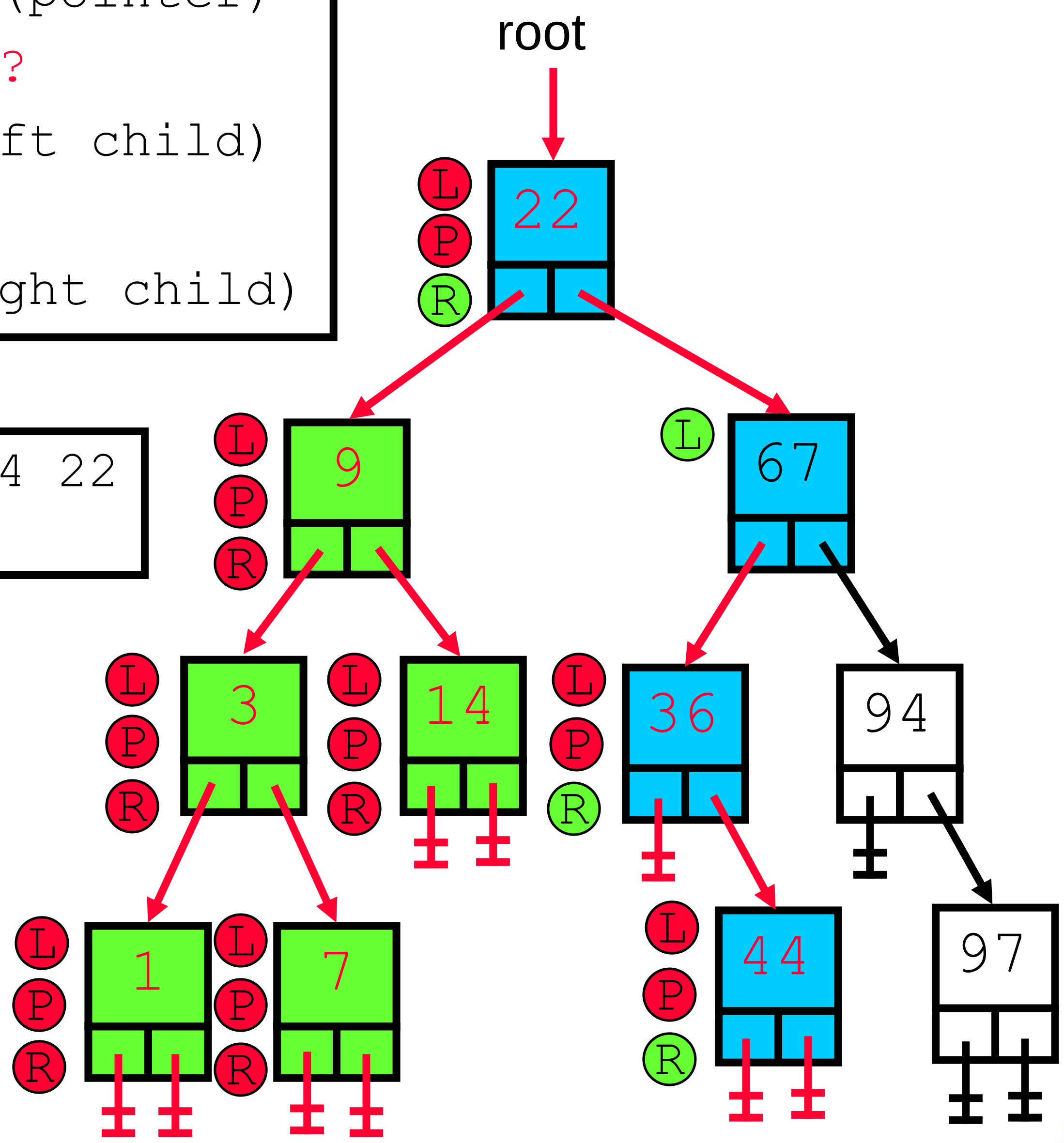
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
        36 44
```



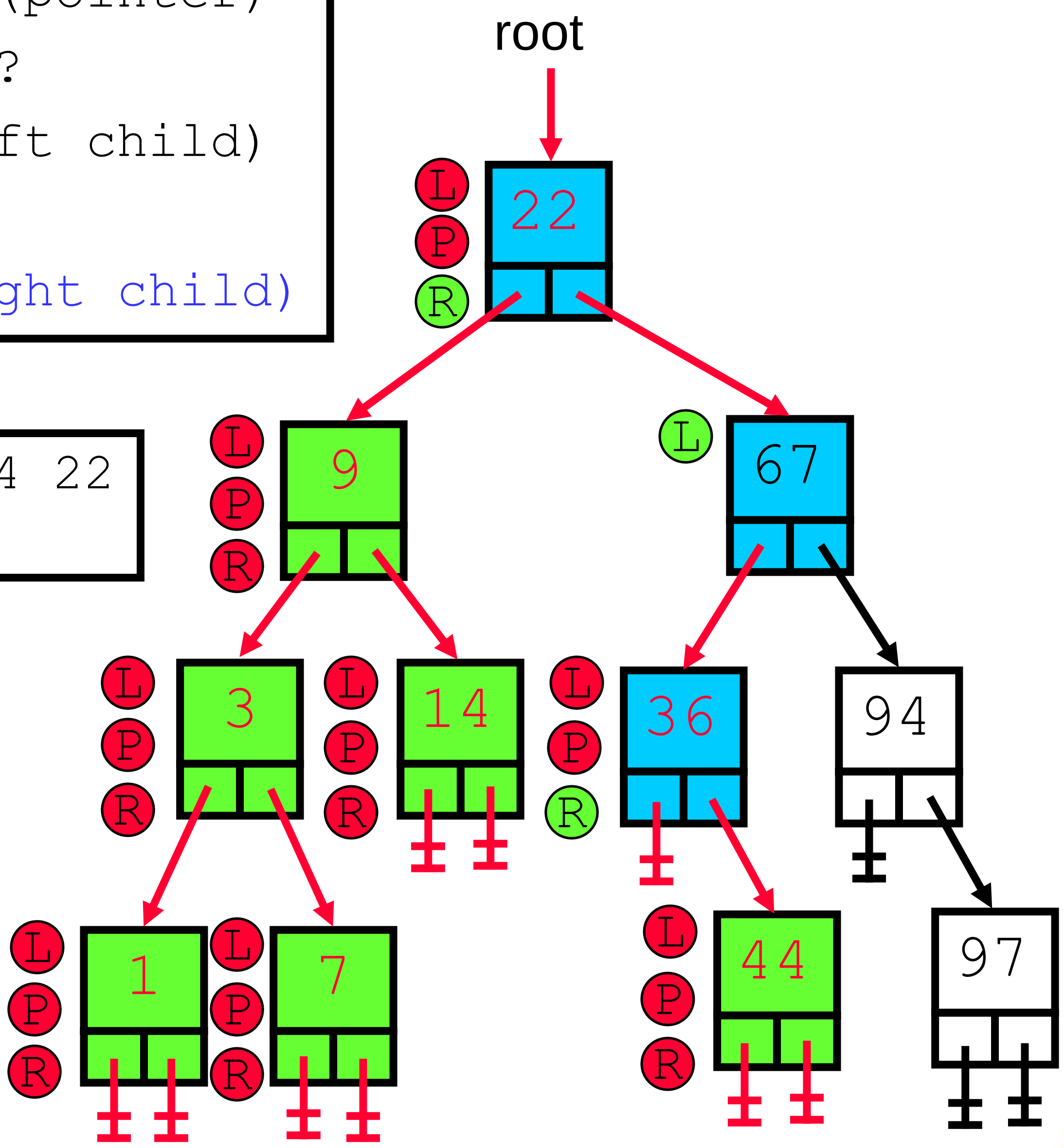
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
        36 44
```



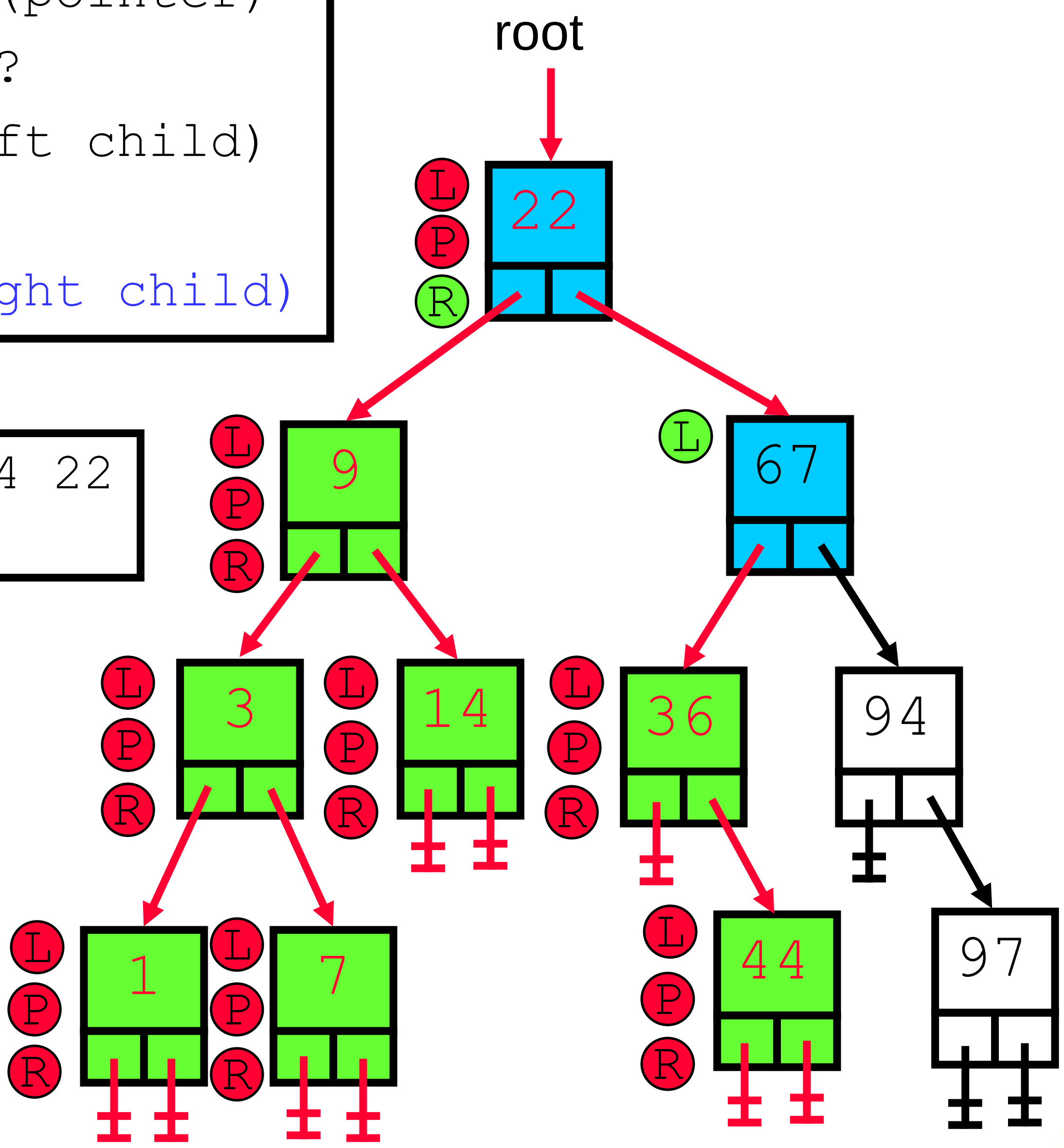
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

Output: 1 3 7 9 14 22 36 44



```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

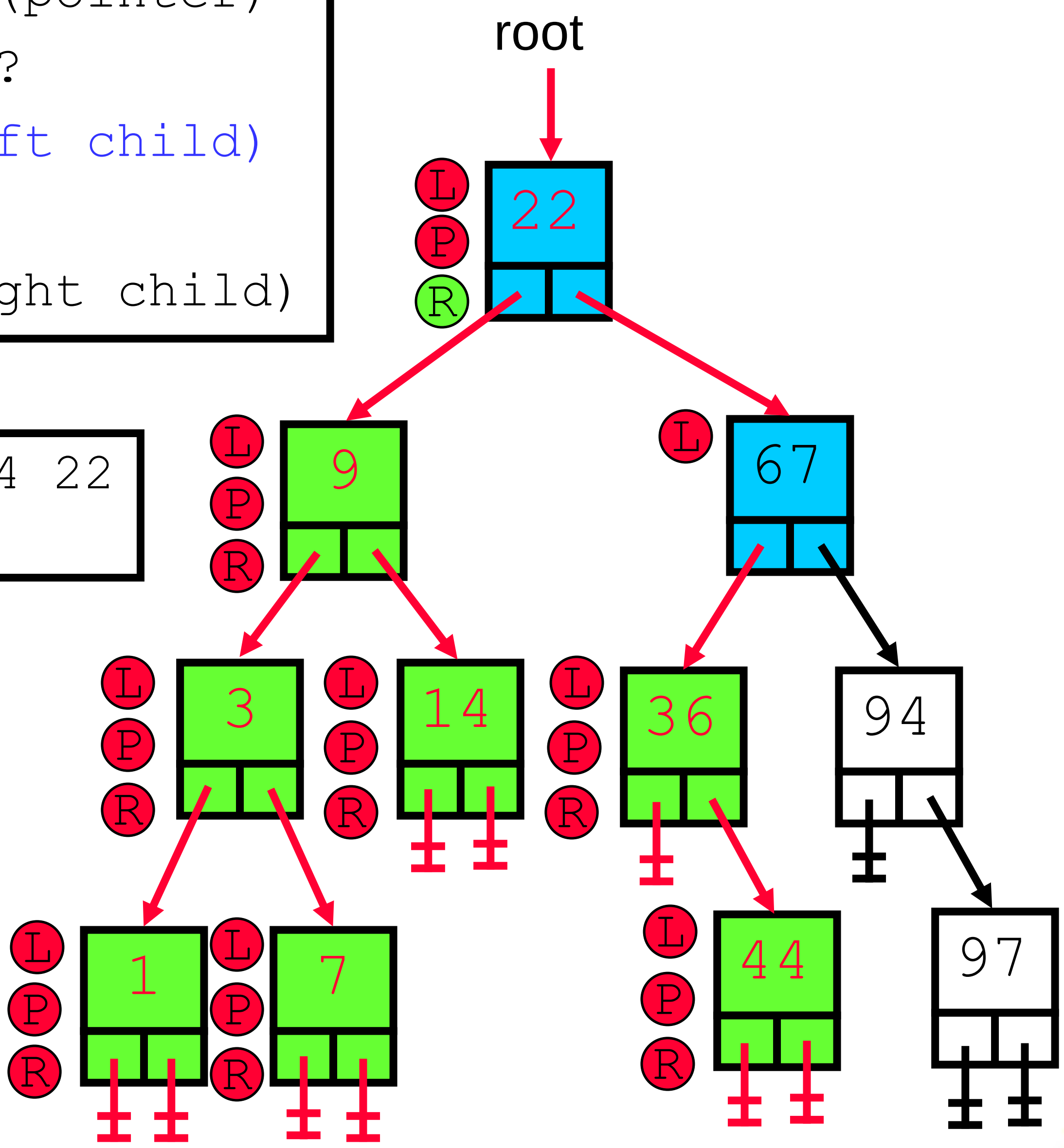
```
Output: 1 3 7 9 14 22
        36 44
```



UNIVERSITY OF CALGARY

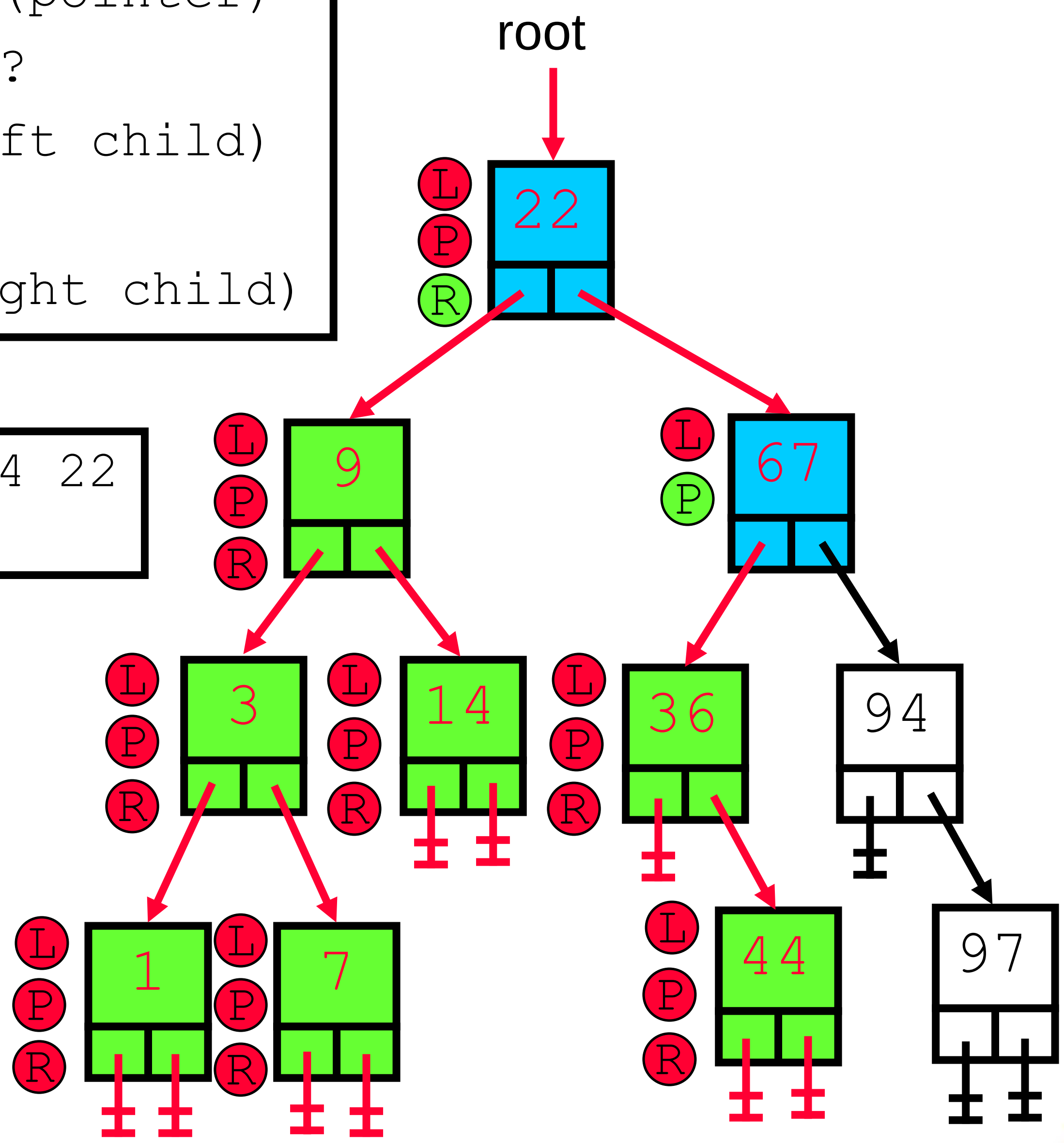
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
(L) InOrderPrint(left child)
(P) print(data)
(R) InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
        36 44
```



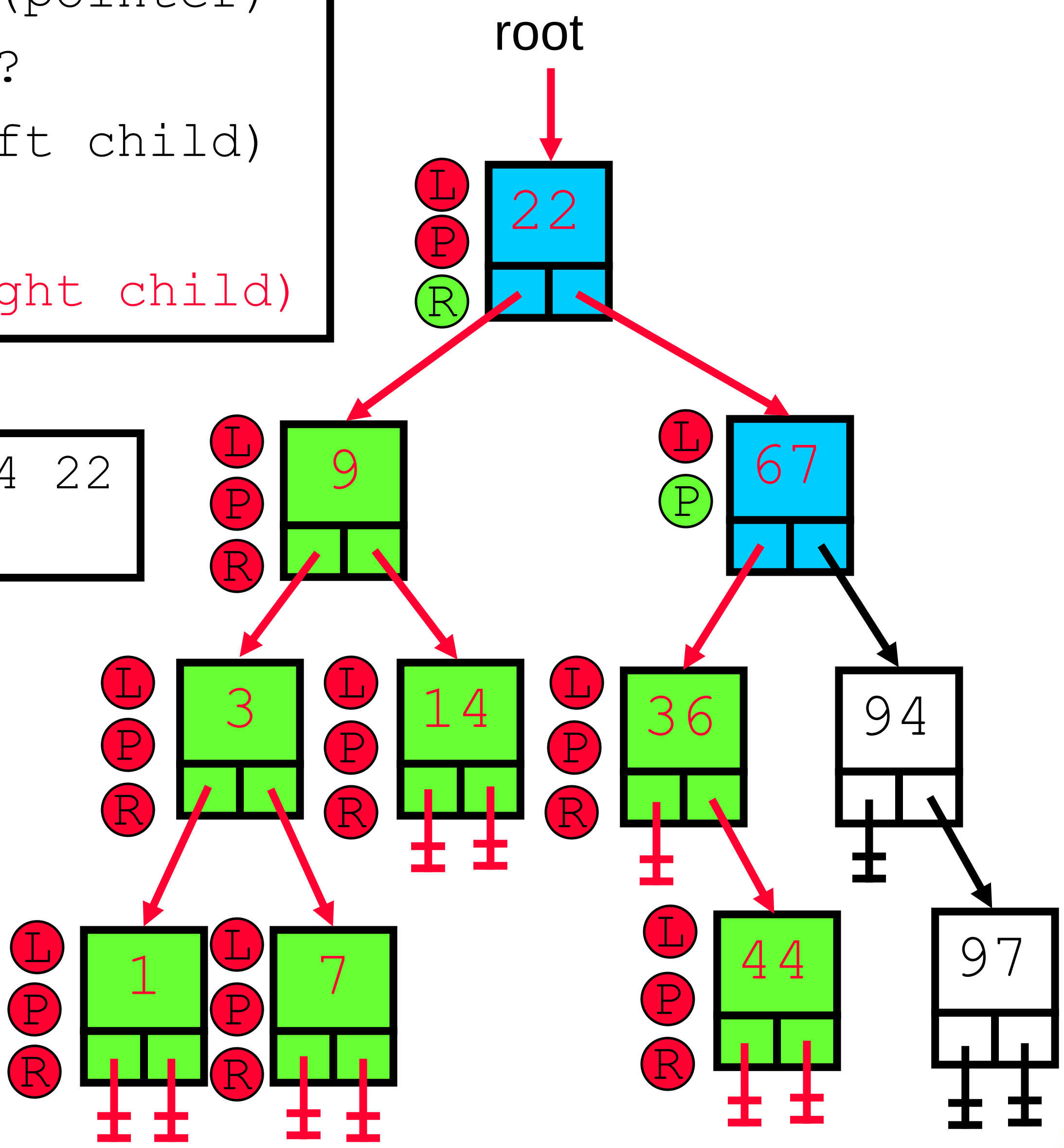
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
        36 44 67
```



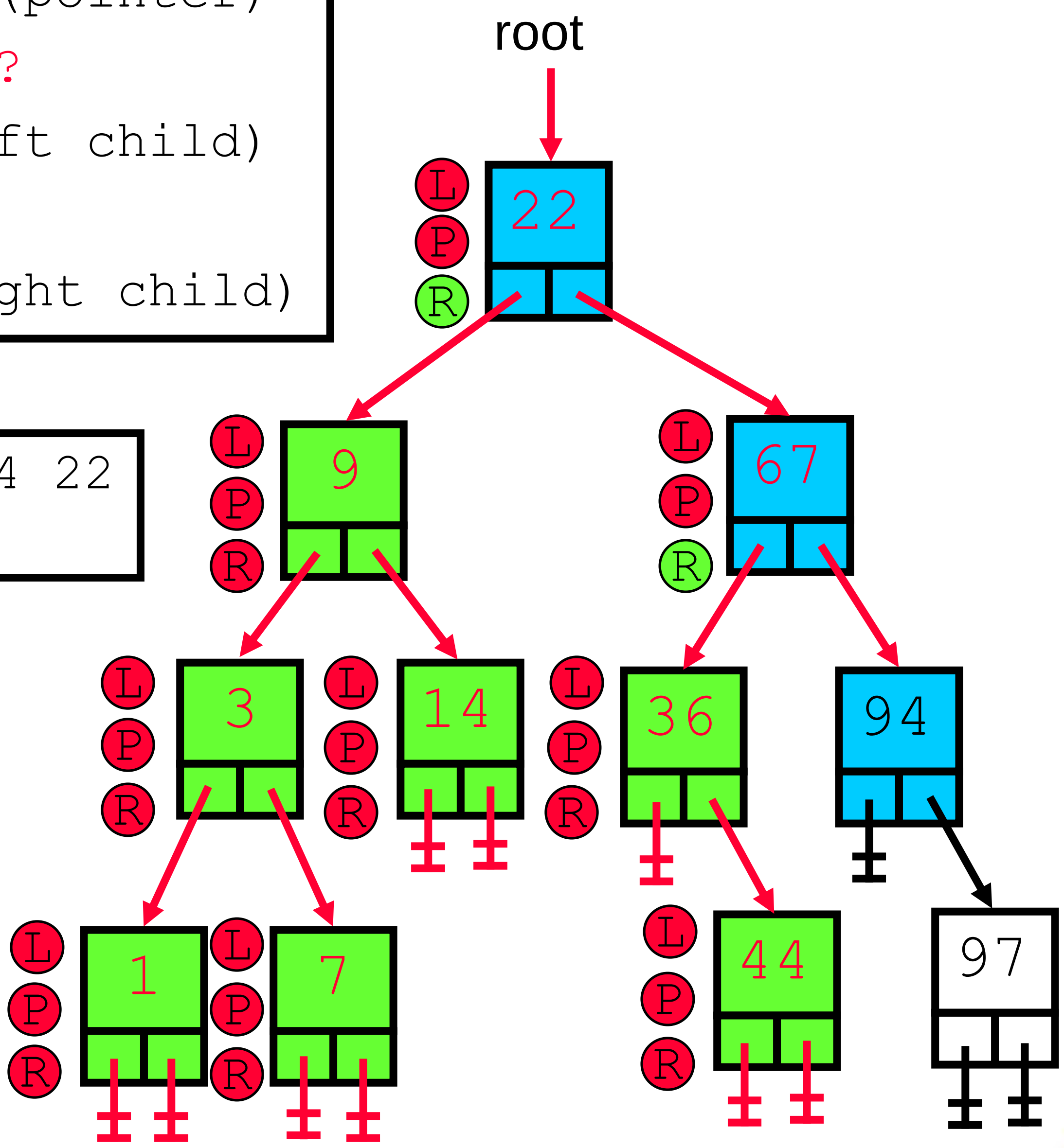
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
        36 44 67
```



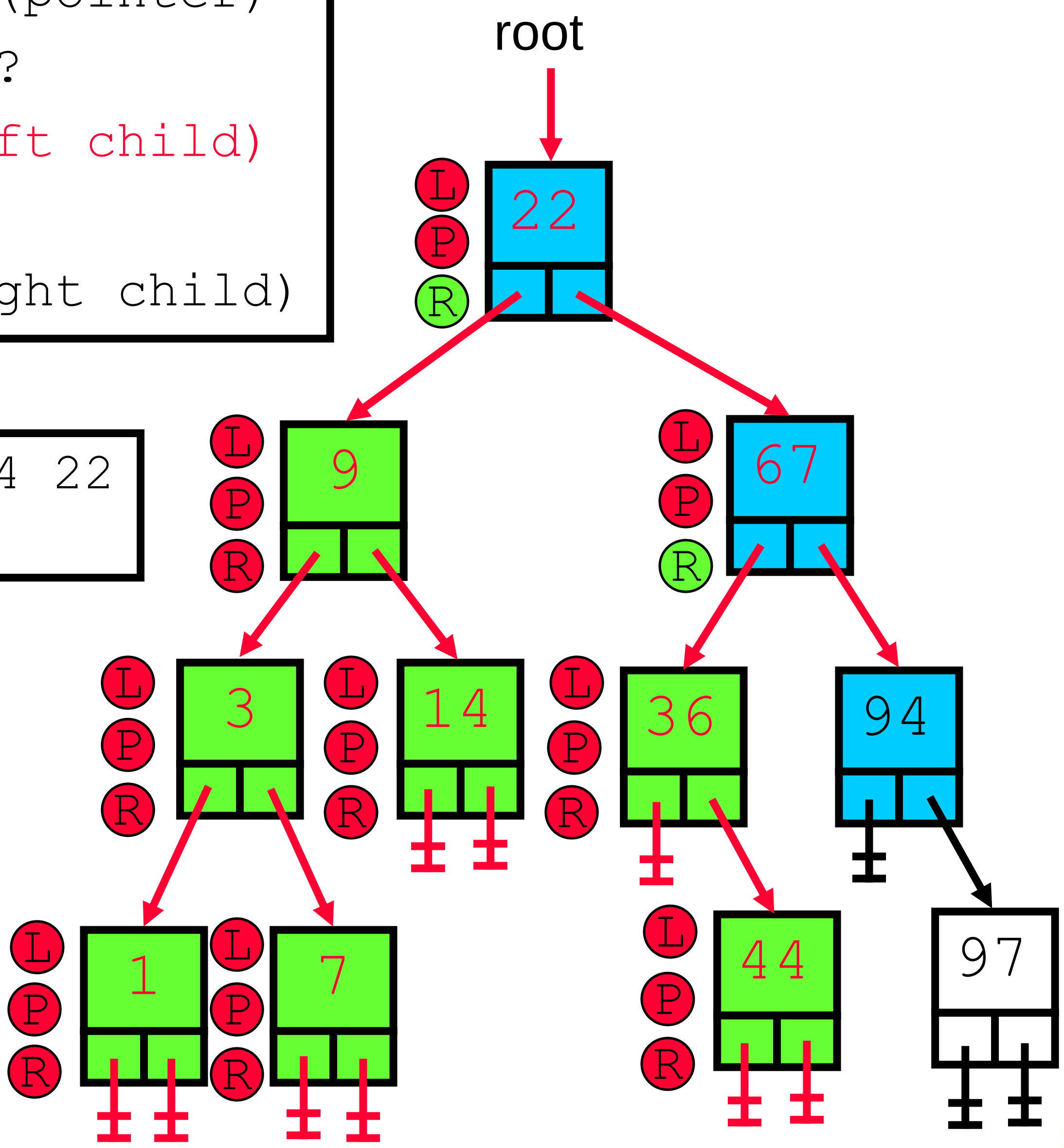
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
        36 44 67
```



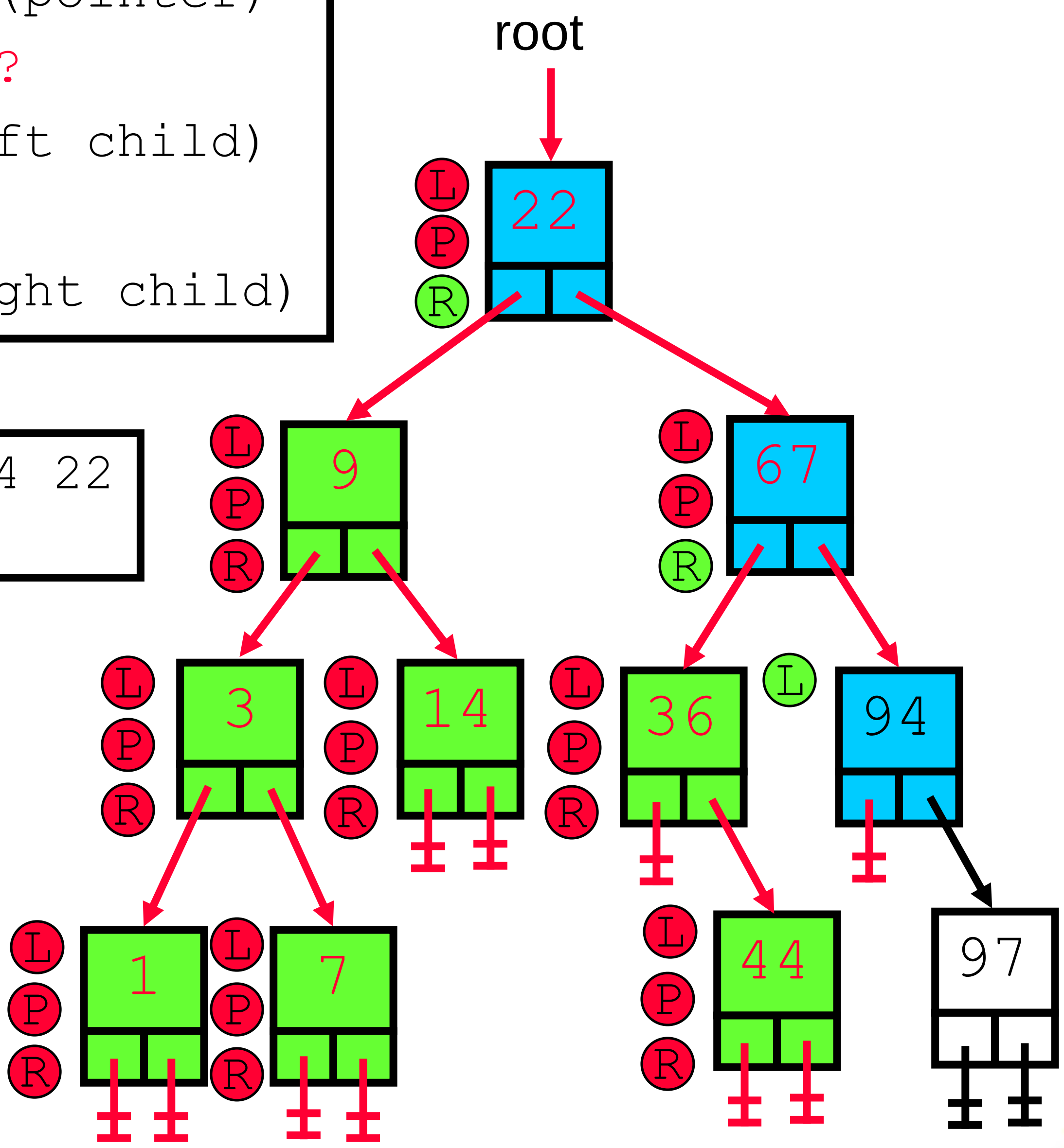
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
        36 44 67
```



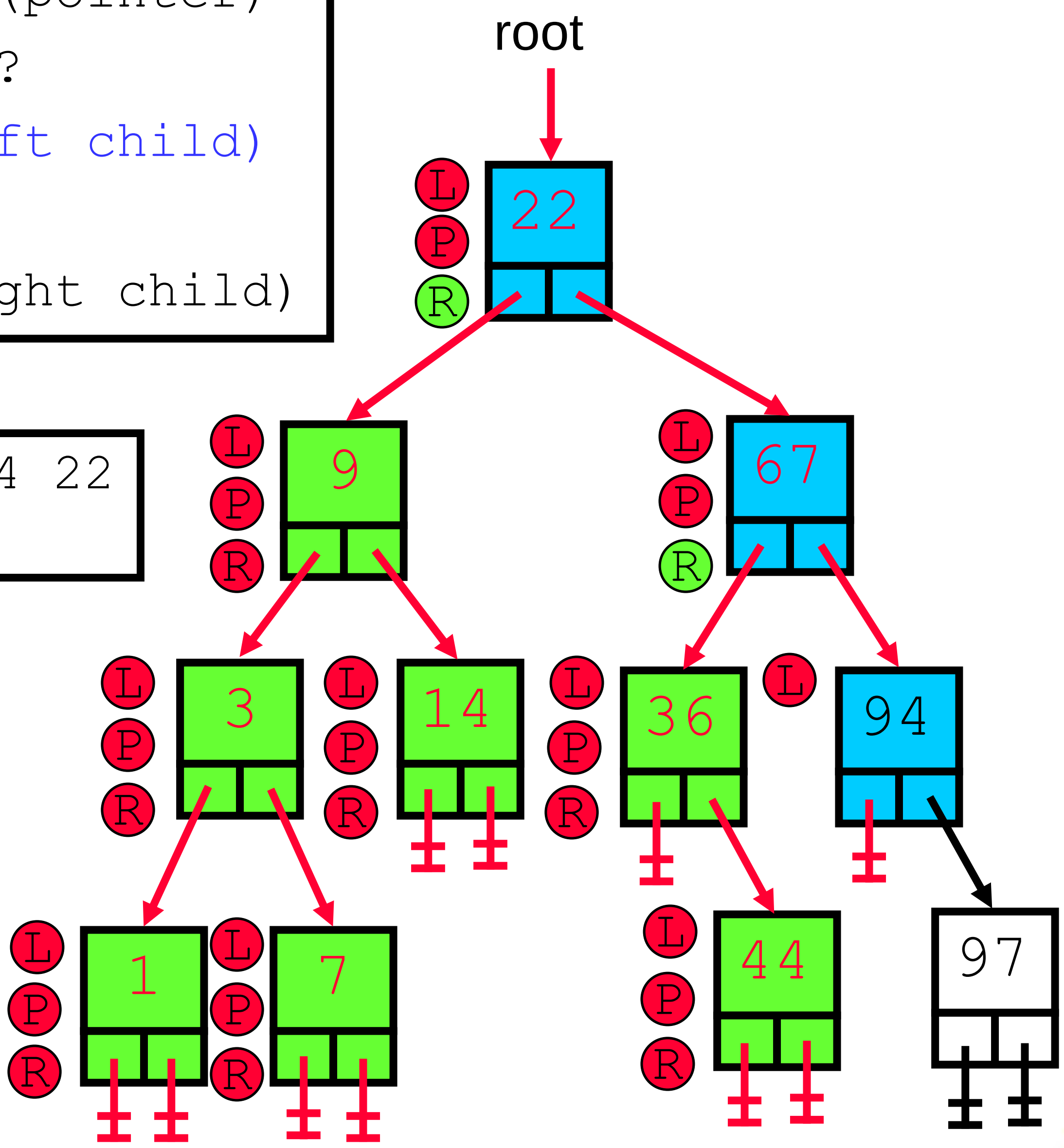
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
        36 44 67
```



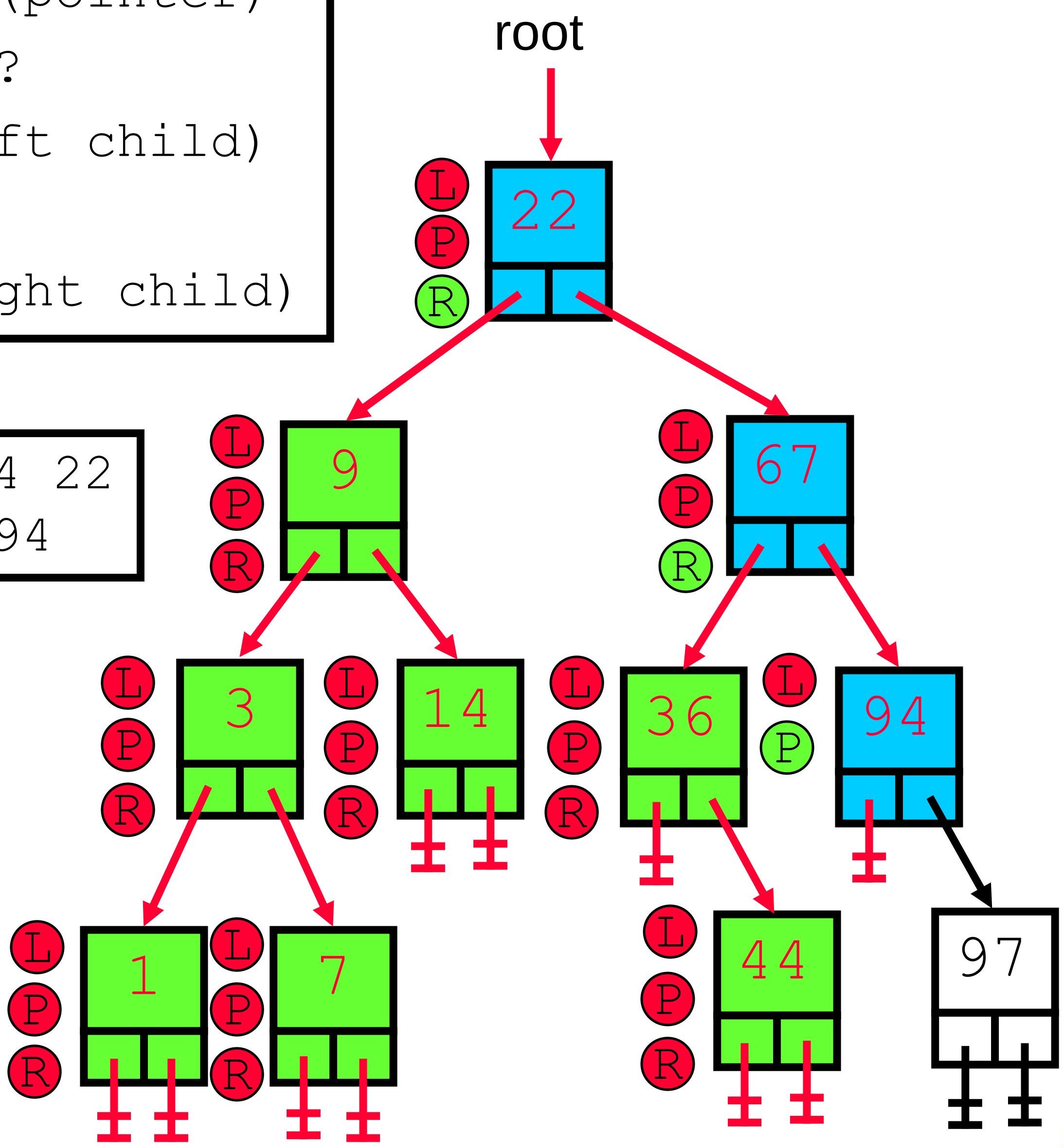
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
        36 44 67
```



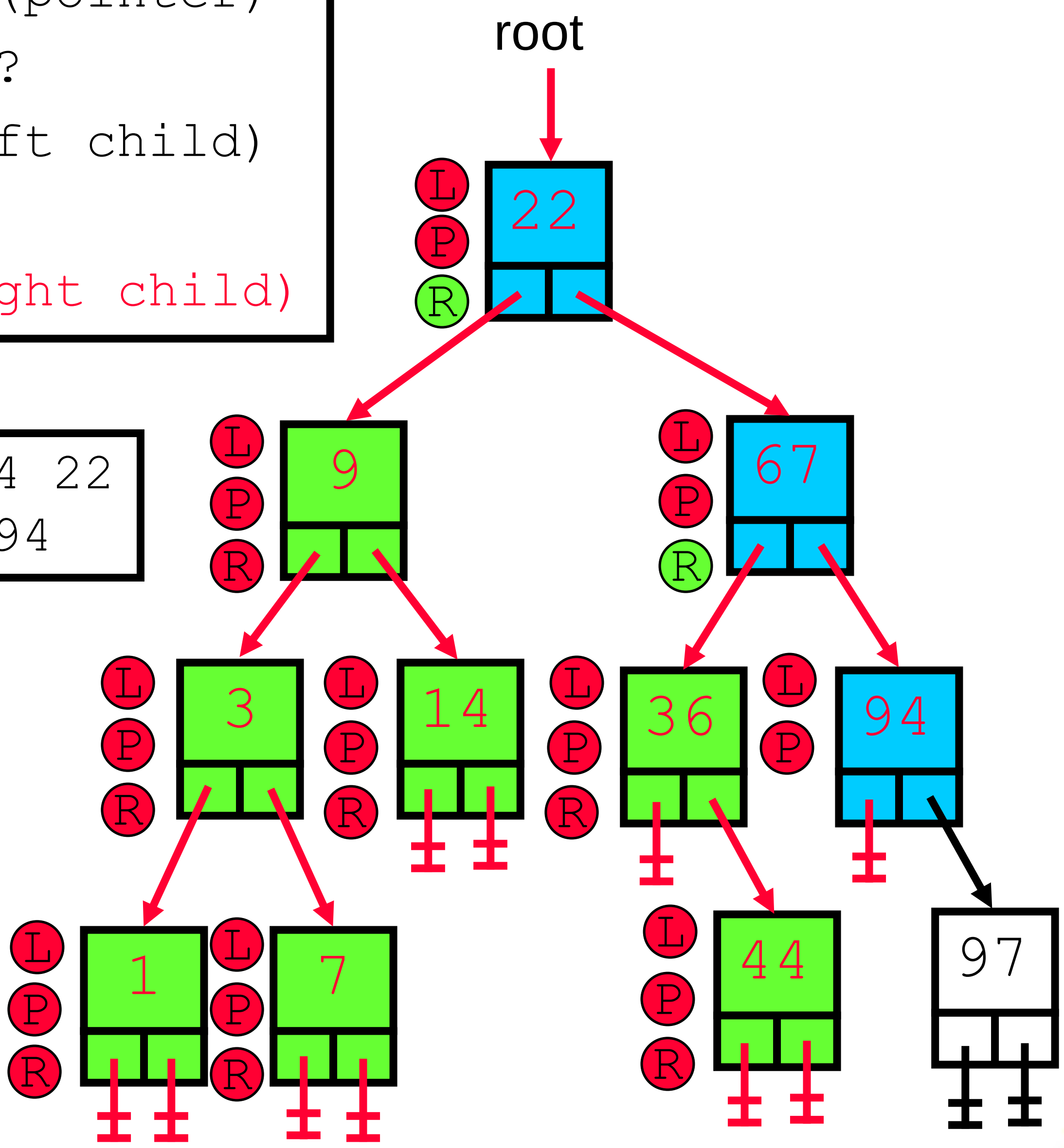
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
        36 44 67 94
```



```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

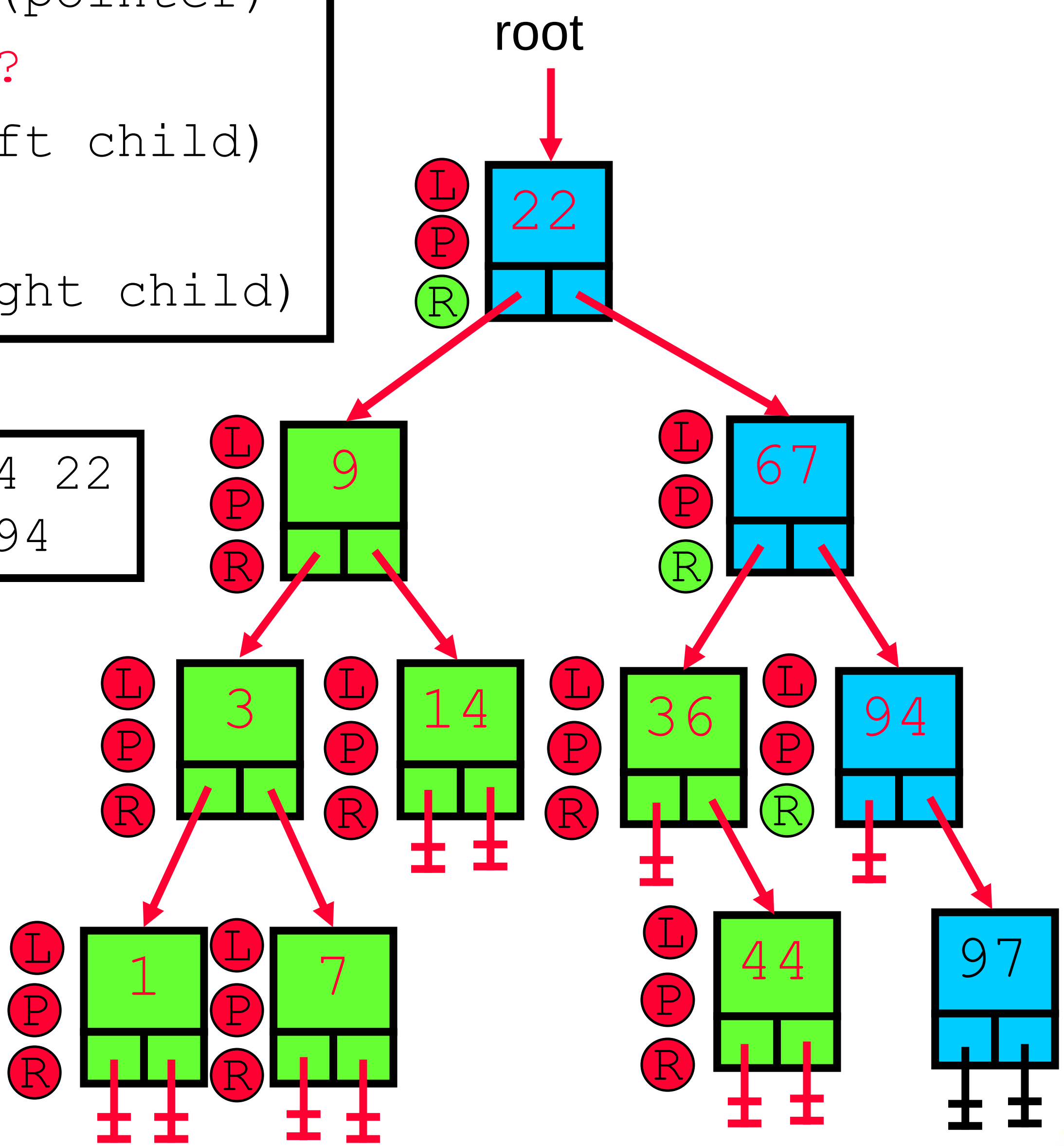
```
Output: 1 3 7 9 14 22
        36 44 67 94
```



UNIVERSITY OF CALGARY

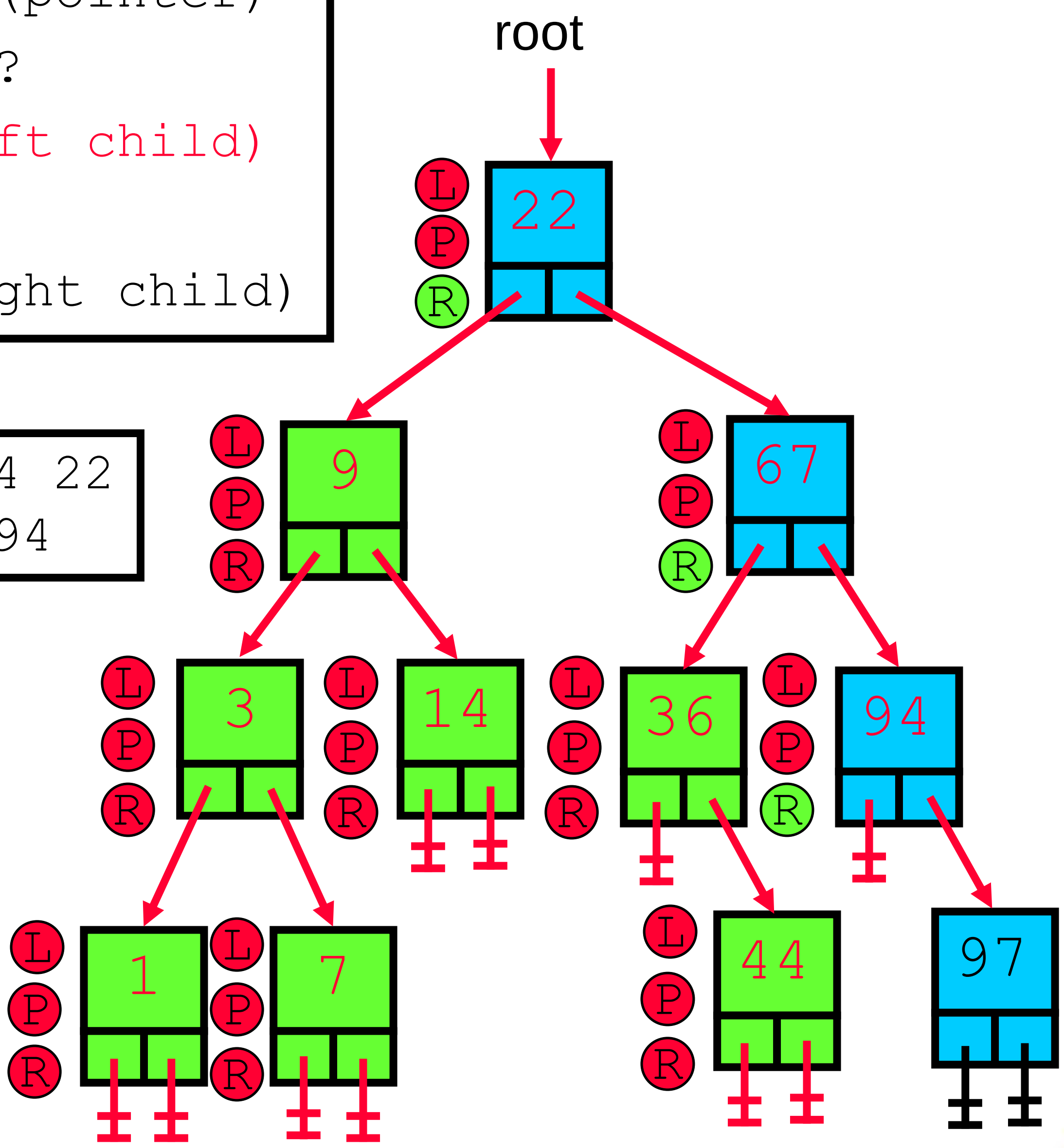
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
        36 44 67 94
```



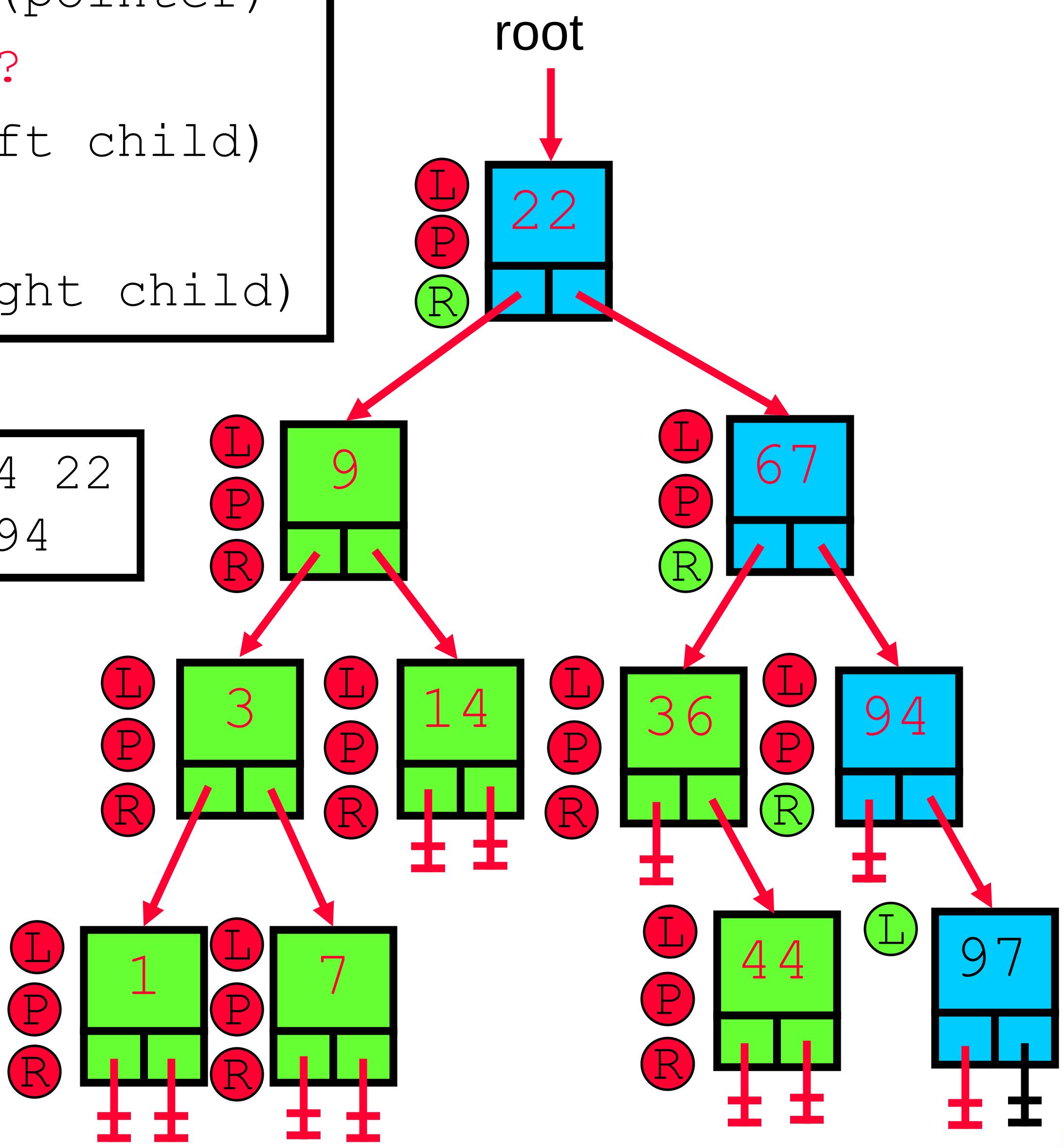
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
        36 44 67 94
```



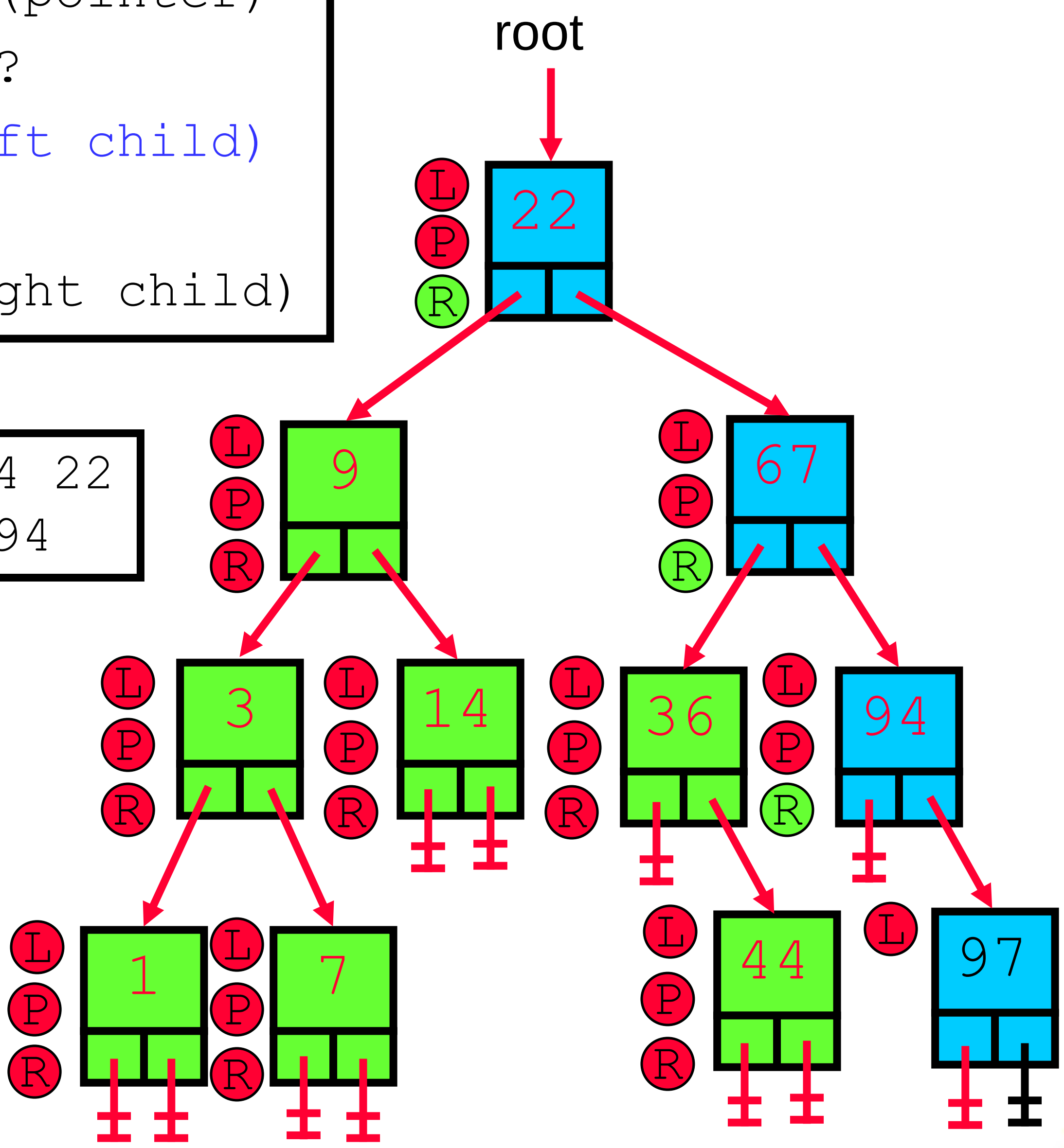
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
        36 44 67 94
```



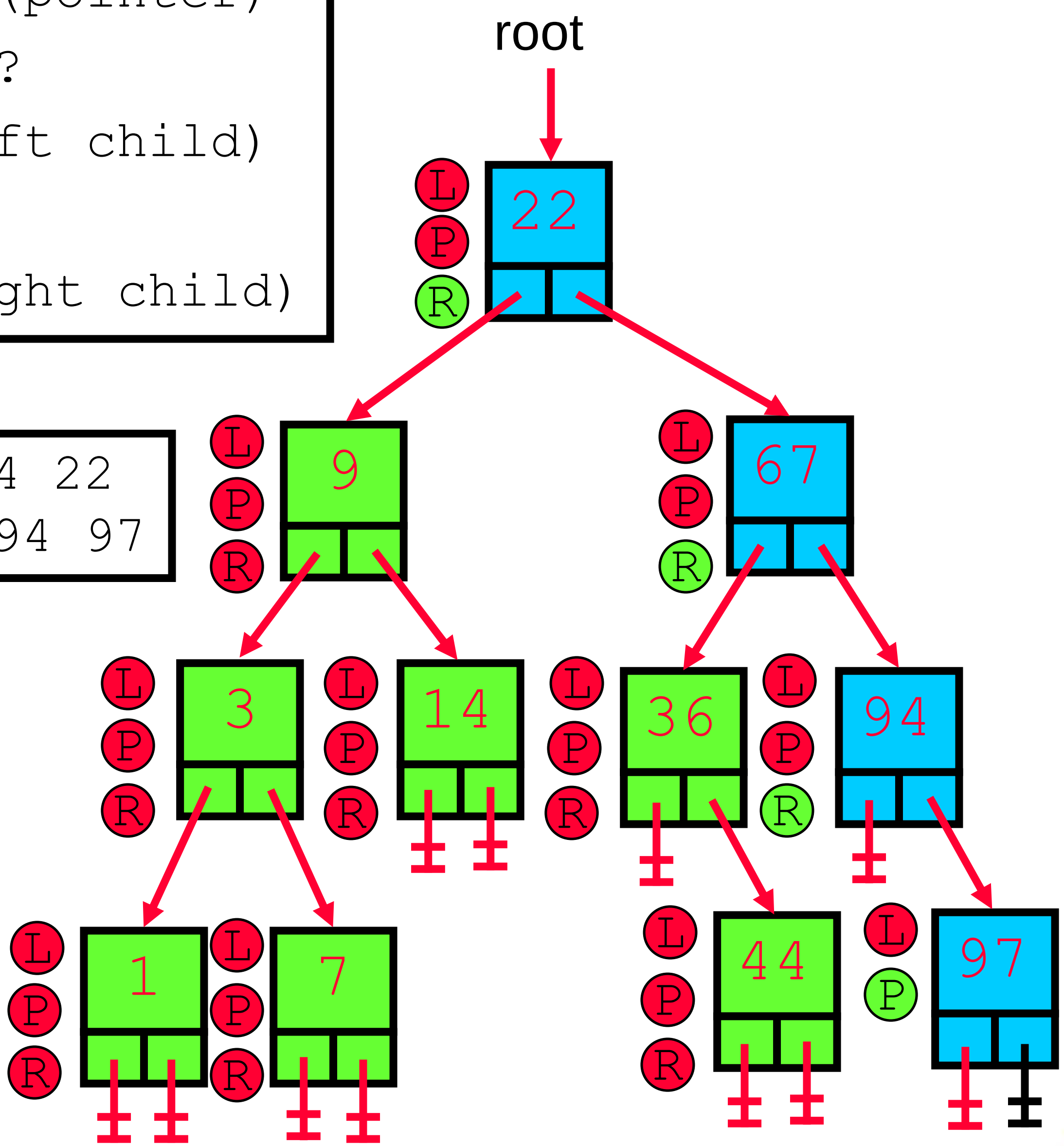
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
        36 44 67 94
```



```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

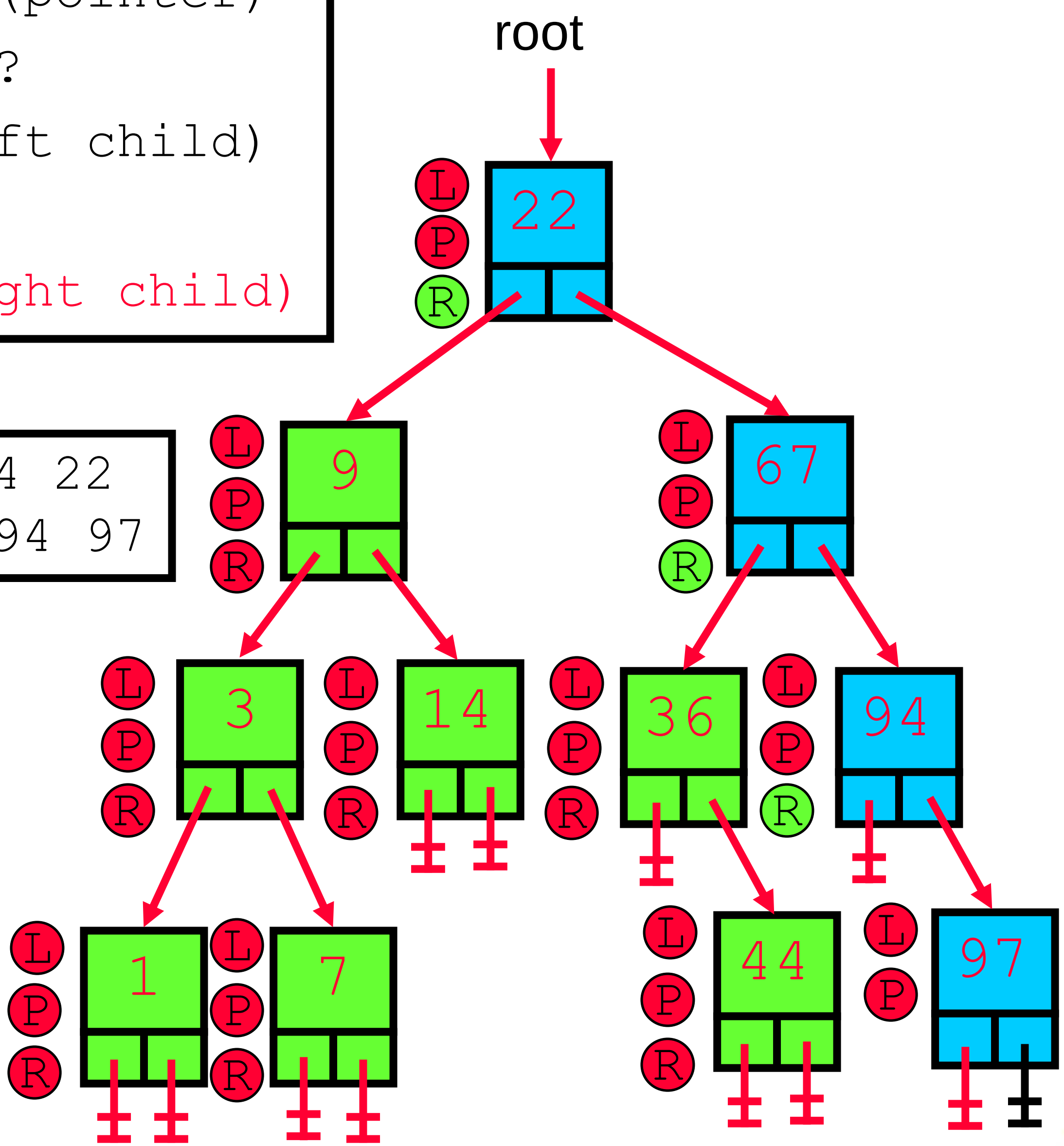
```
Output: 1 3 7 9 14 22
        36 44 67 94 97
```



UNIVERSITY OF CALGARY

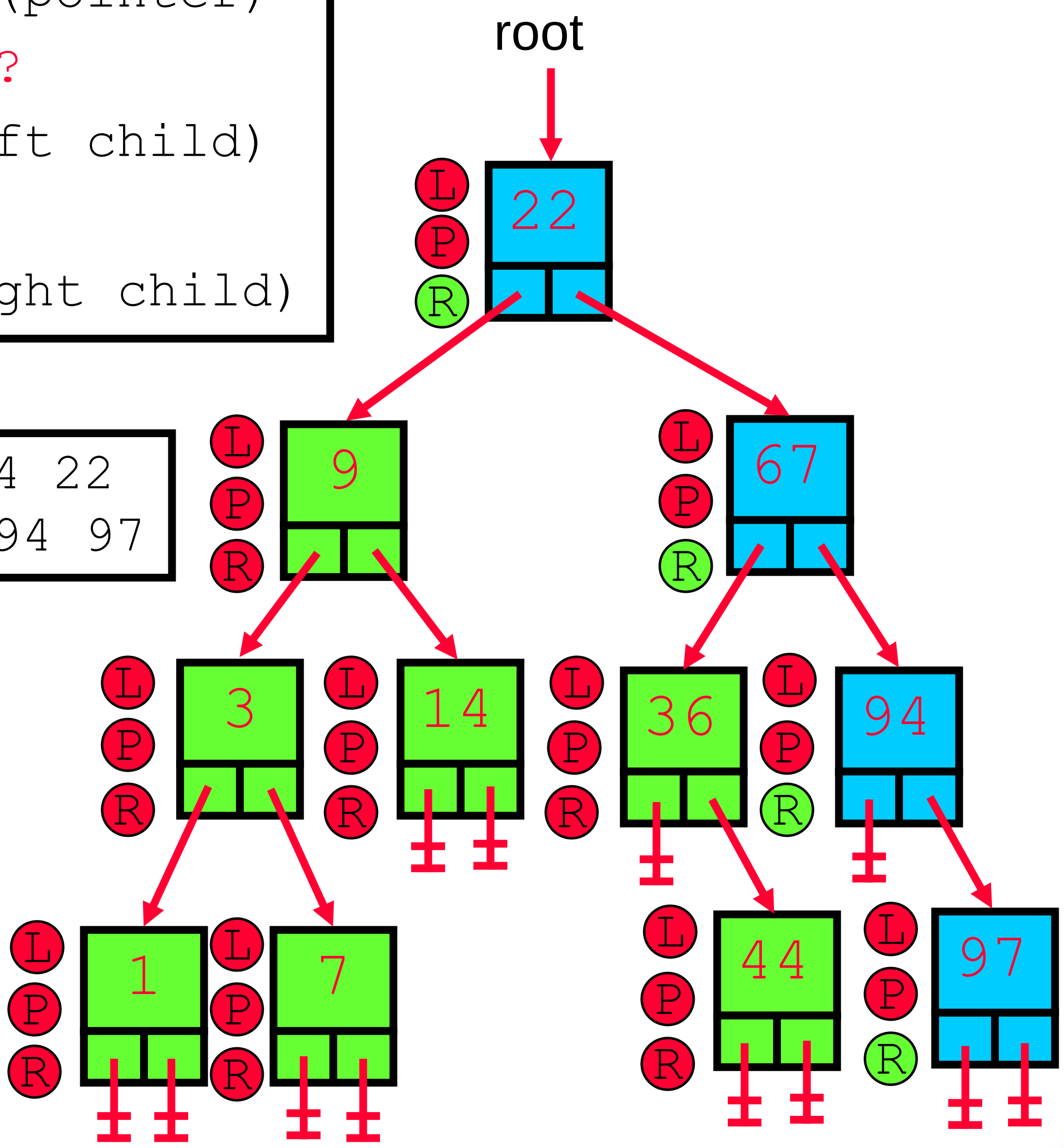
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
        36 44 67 94 97
```



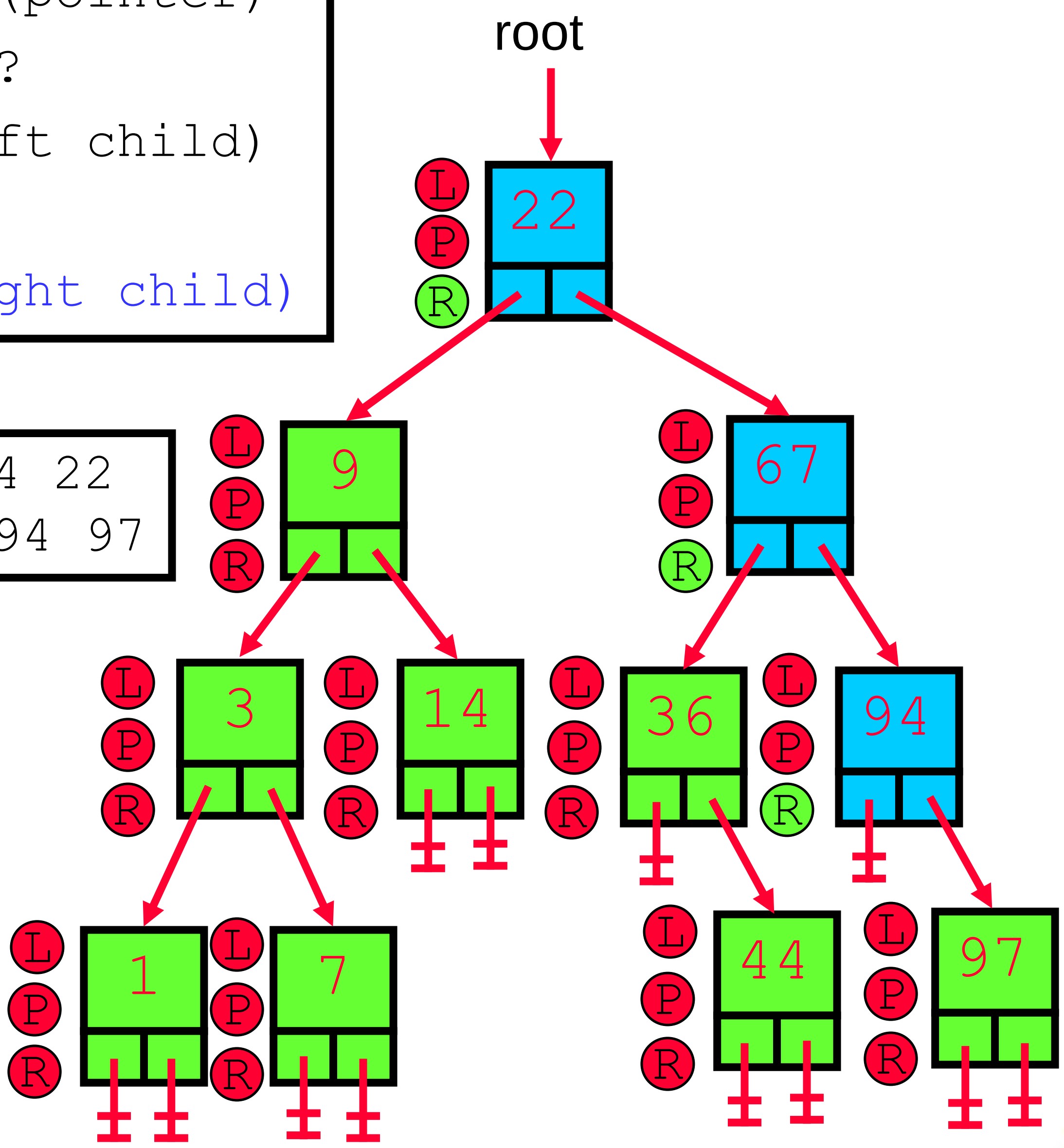
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
Ⓛ InOrderPrint(left child)
Ⓟ print(data)
Ⓡ InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
        36 44 67 94 97
```



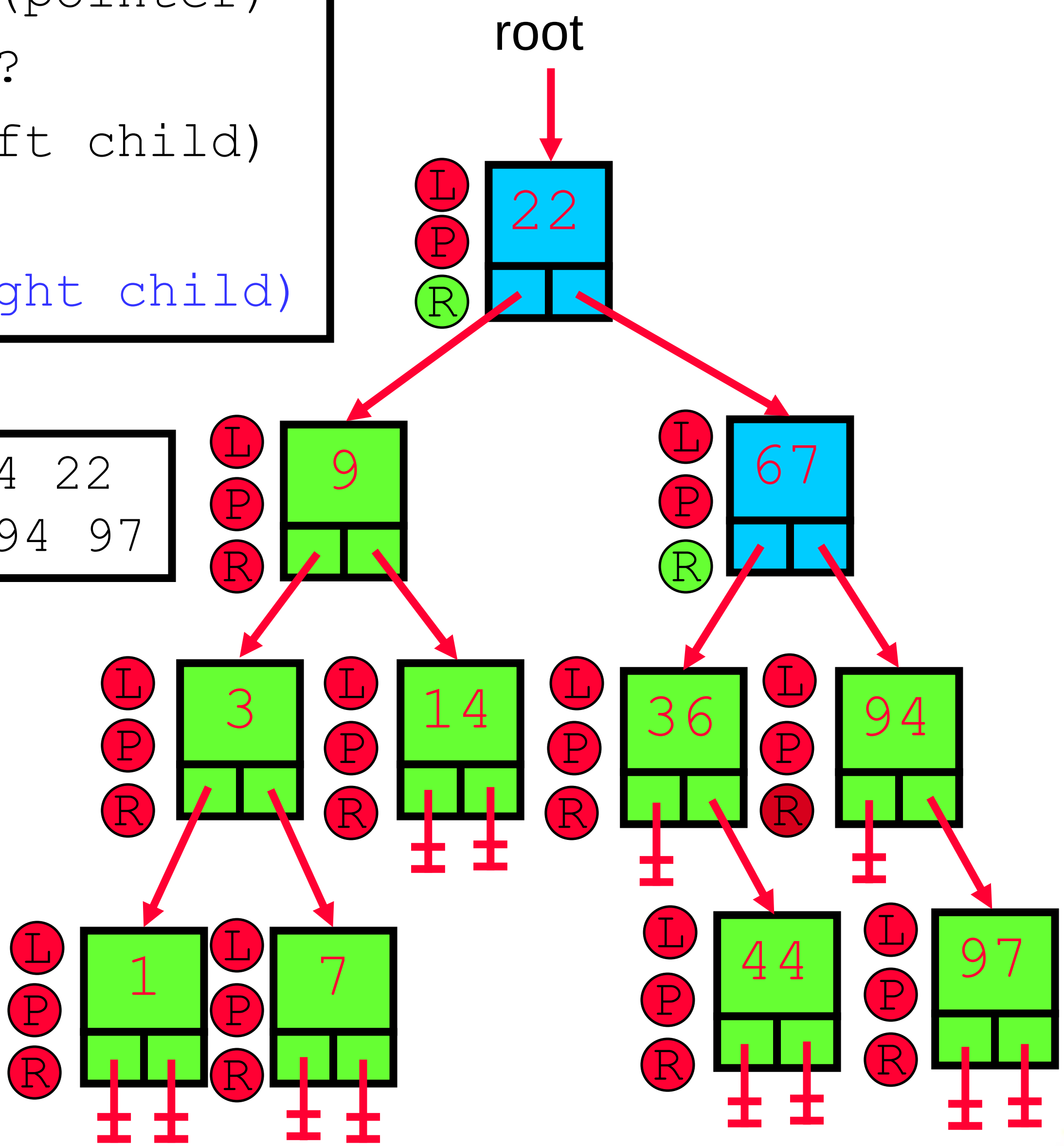
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
        36 44 67 94 97
```



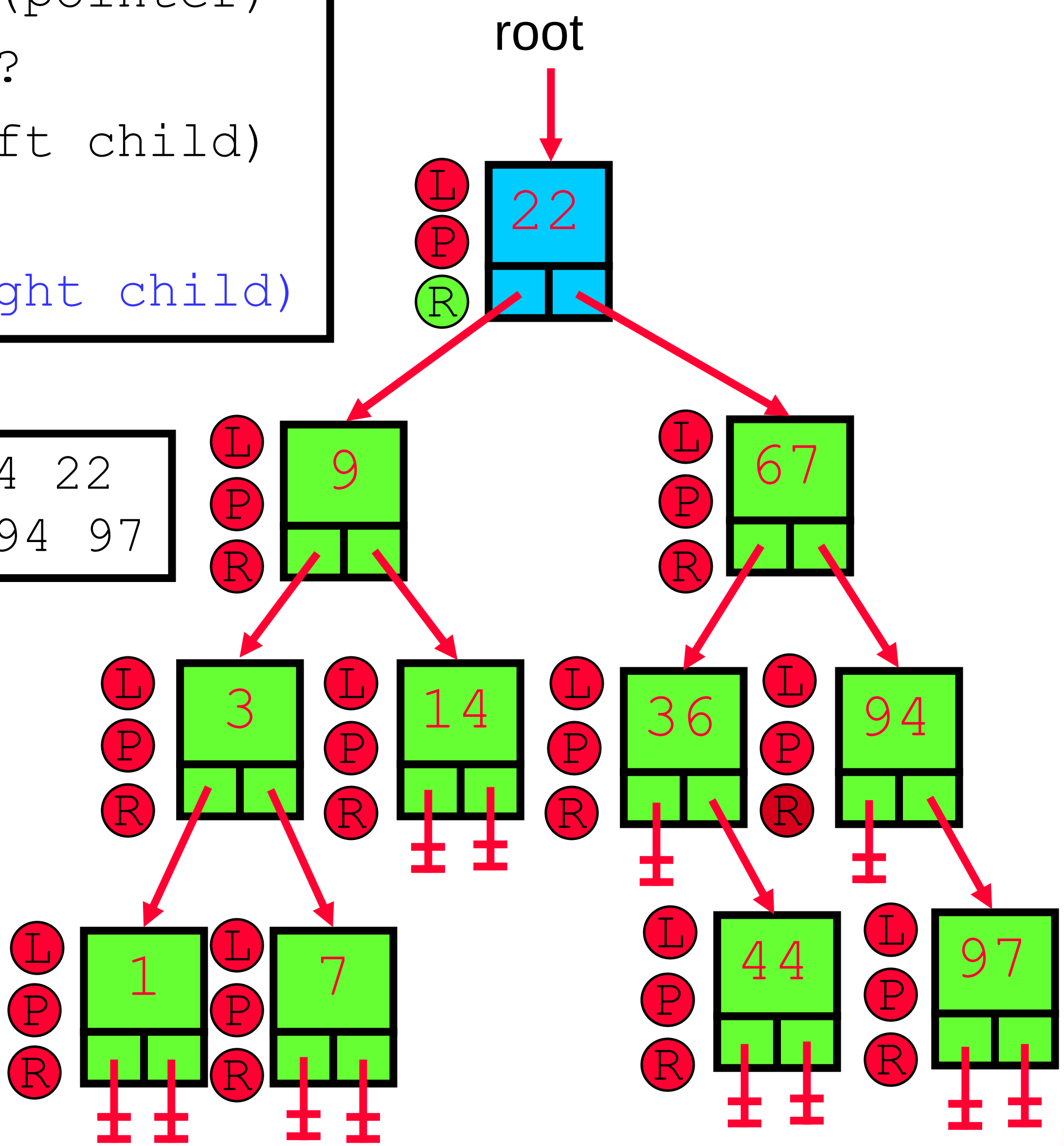
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
        36 44 67 94 97
```



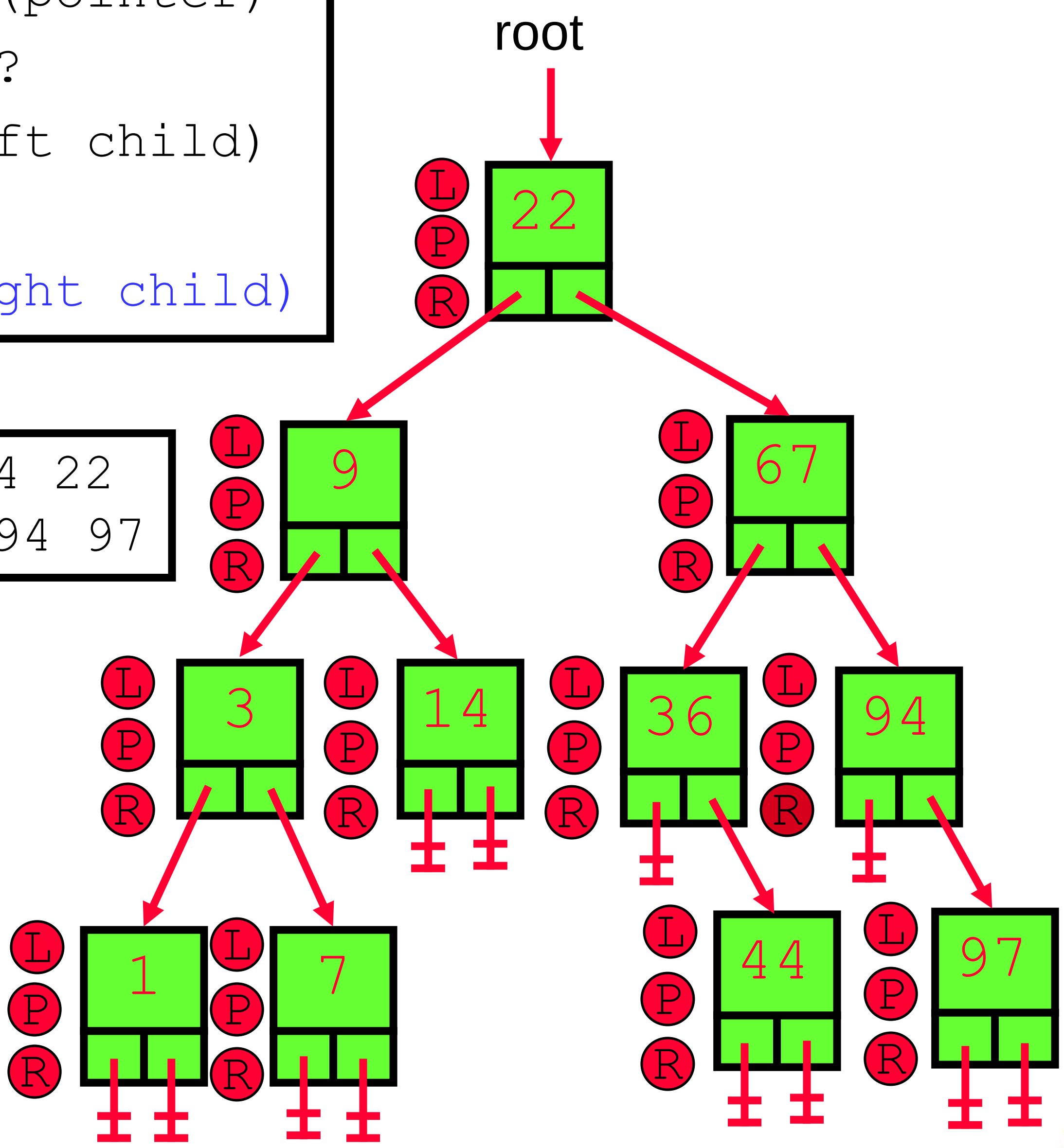
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
        36 44 67 94 97
```



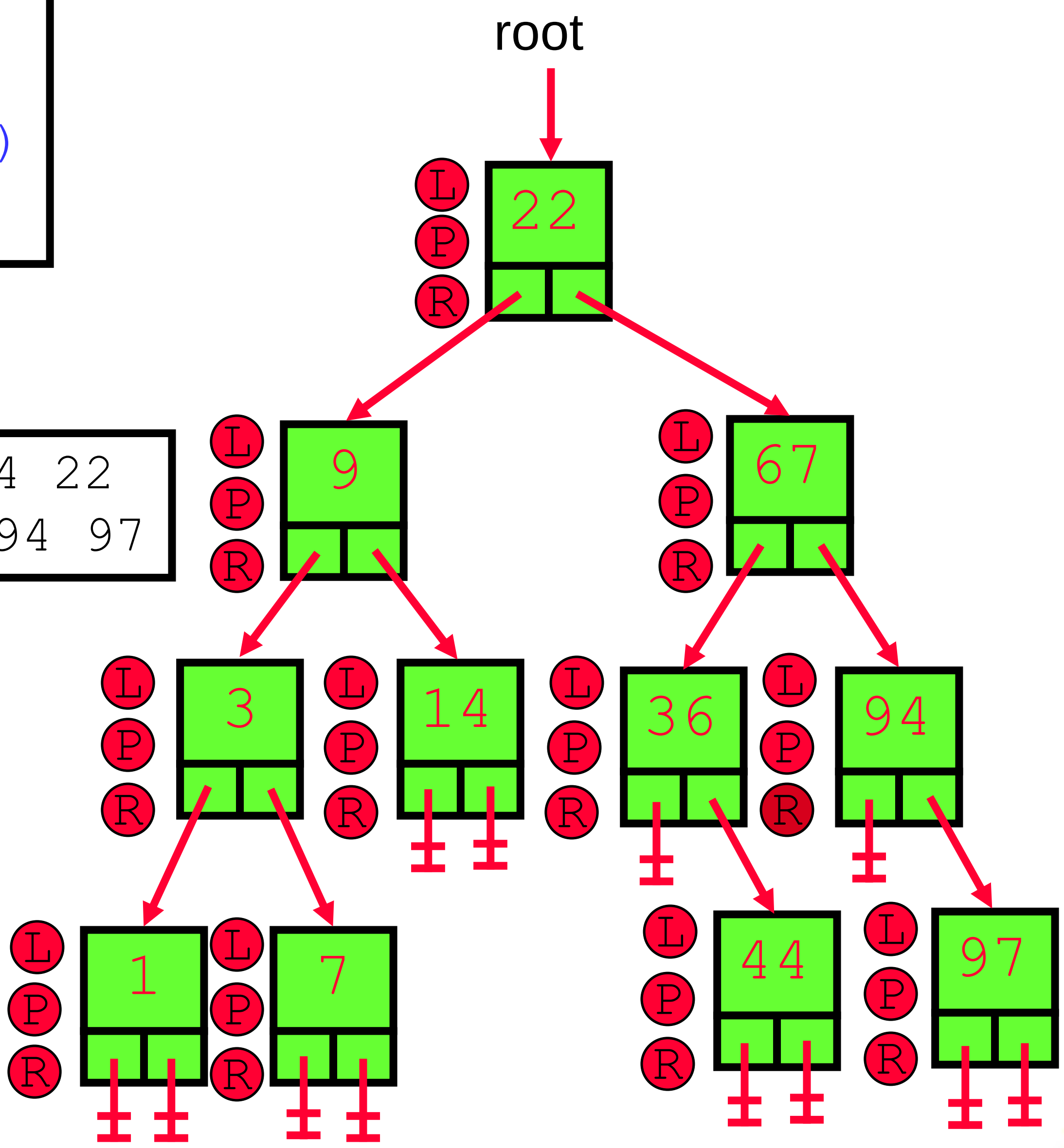
```
Proc InOrderPrint(pointer)
 pointer NOT NULL?
L InOrderPrint(left child)
P print(data)
R InOrderPrint(right child)
```

```
Output: 1 3 7 9 14 22
        36 44 67 94 97
```



```
Algorithm Example
. . .
InOrderPrint(root)
. . .
```

```
Output: 1 3 7 9 14 22
        36 44 67 94 97
```



Summary

- An In-Order traversal visits every node
  - Recurse left first
  - Do something with current
  - Recurse right last

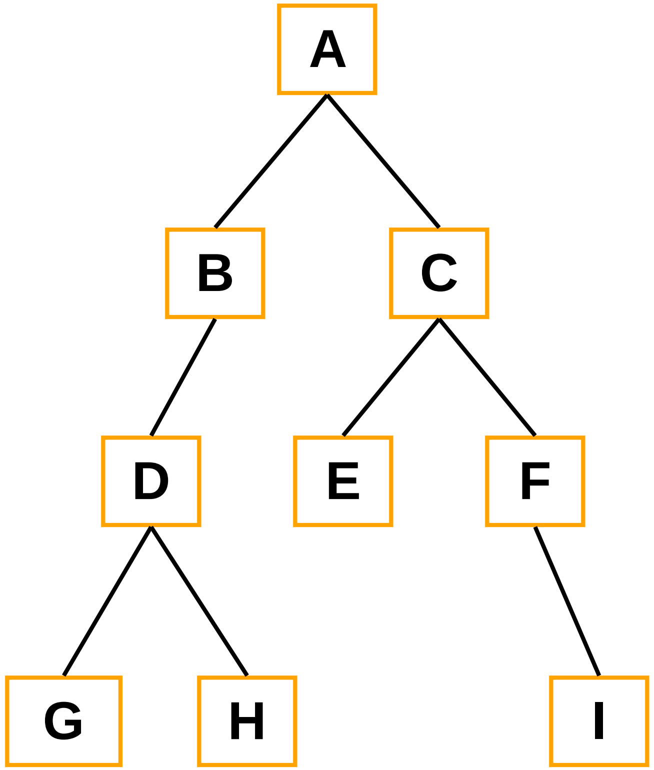
- The “left, current, right” logic is repeated recursively at every node.

- For a BST, an in-order traversal accesses the elements in ascending order.

Traversing a Binary Tree

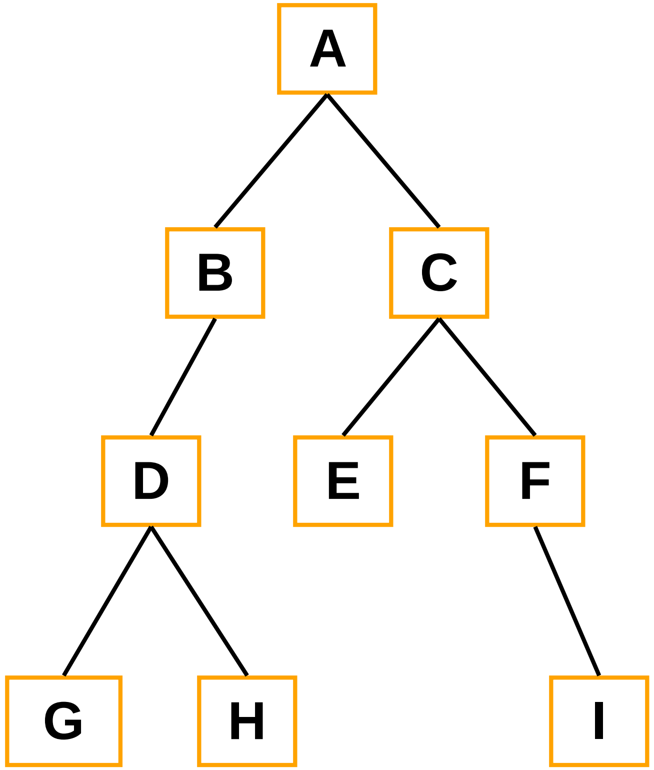
Level Order Traversal /Breadth First Traversal

- What if we want to print everything on one level of the tree at a time?
- We can’t do simple recursion
- We need to deal with all siblings on a level before we do any of those siblings children



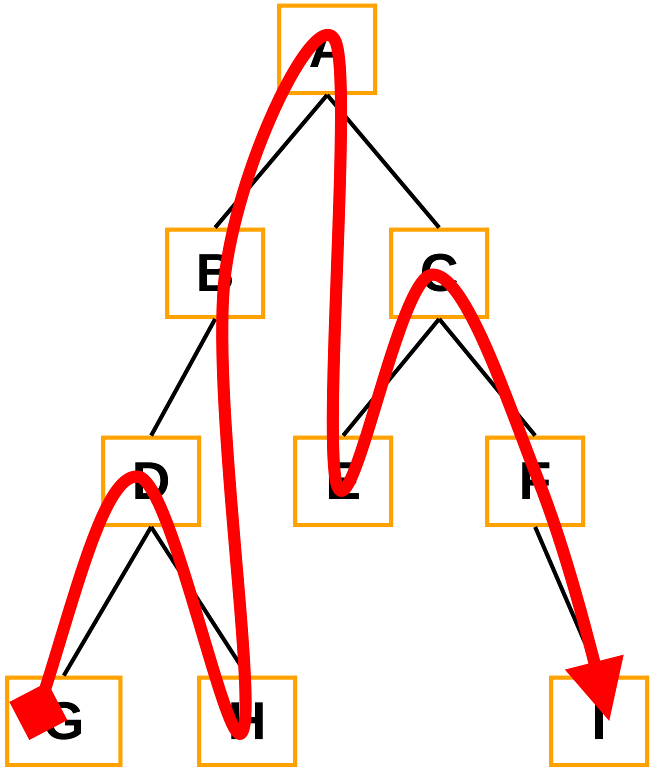
How can we get the following output?

A B C D E F G H I



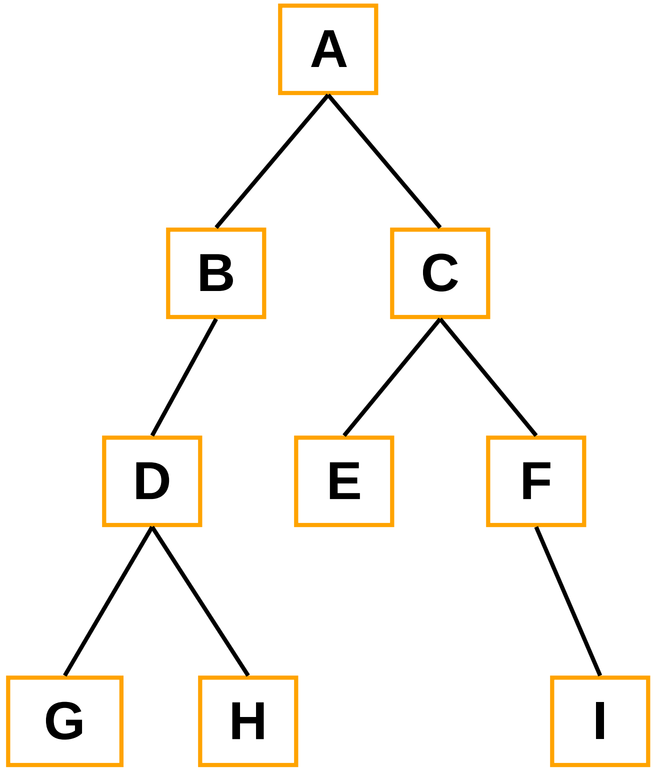
Inorder (left, node, right)?

G D H B A E C F I



Inorder (left, node, right)?

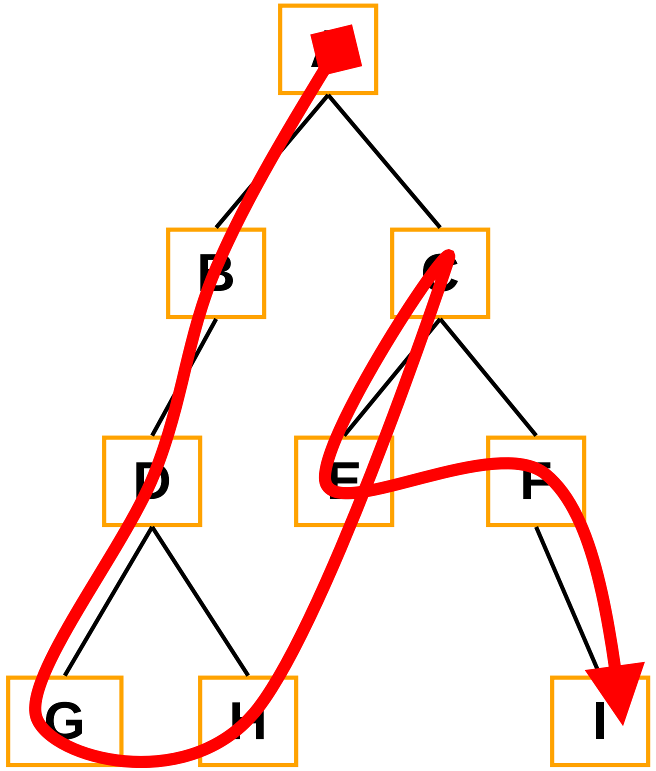
G D H B A E C F I



Preorder (node, left, right)?

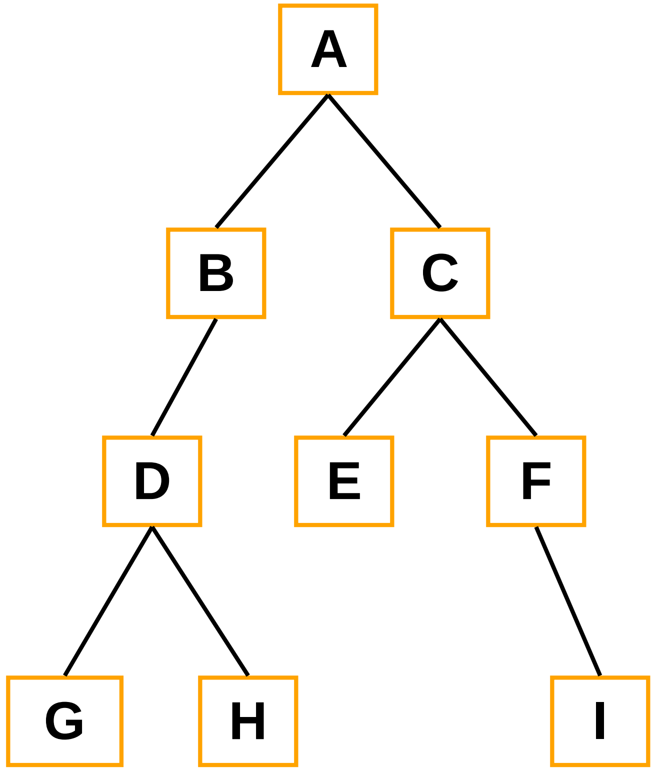
A B D G H C E F I

UNIVERSITY OF CALGARY



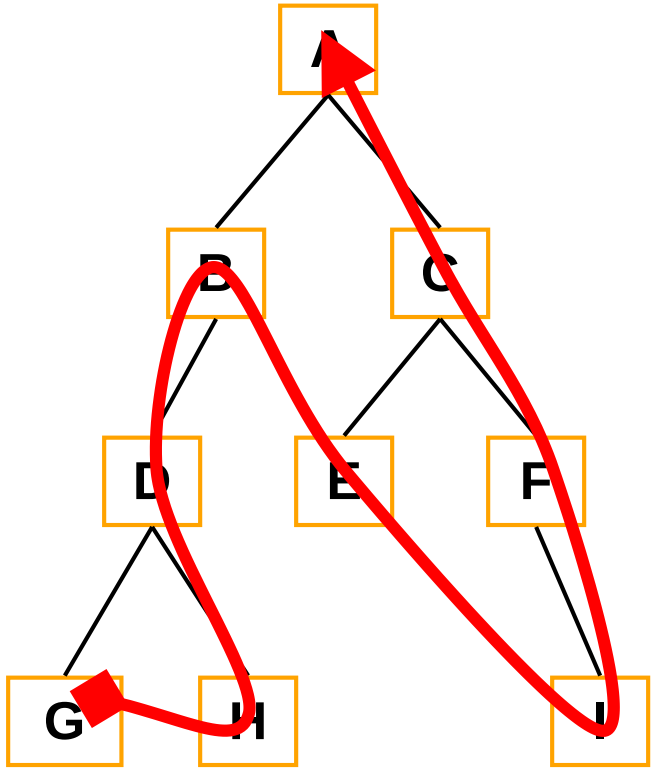
Preorder (node, left, right)?

A B D G H C E F I



Postorder (left, right, node)?

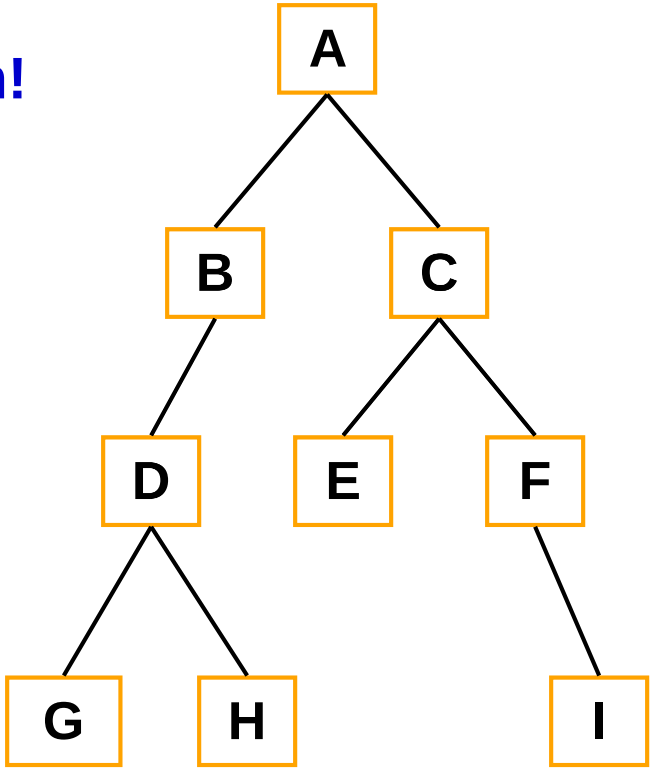
G H D B E I F C A



Postorder (left, right, node)?

G H D B E I F C A

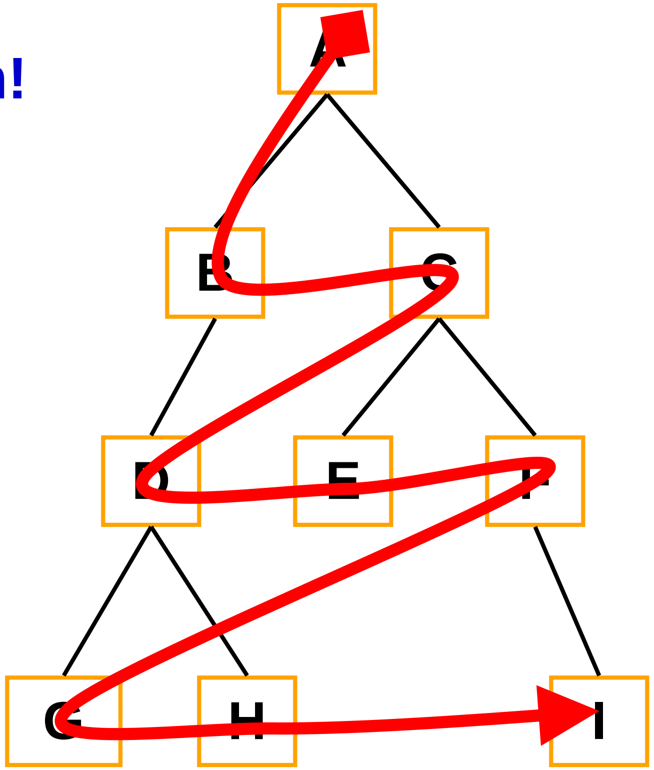
We need a new traversal algorithm!



How can we get the following output?

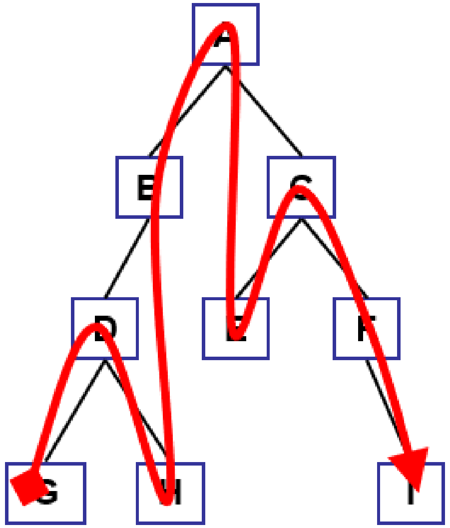
A B C D E F G H I

We need a new traversal algorithm!

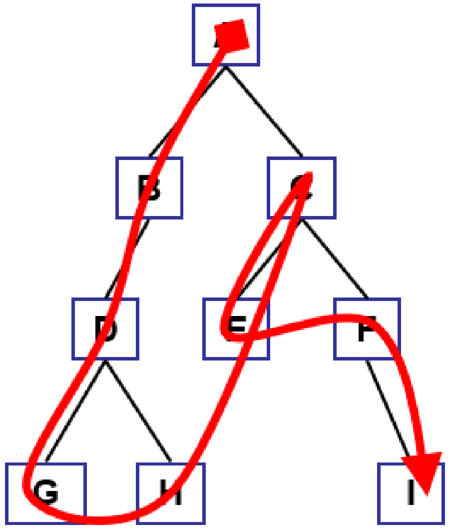


How can we get the following output?

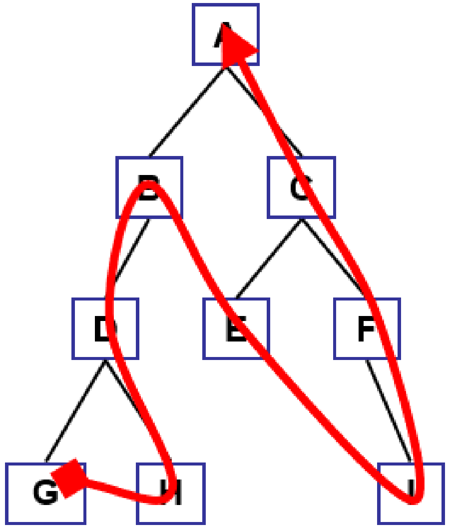
A B C D E F G H I



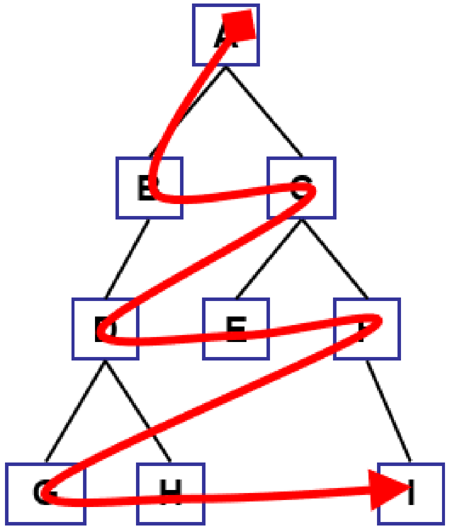
In-order



Pre-order



Post-order



Level-order

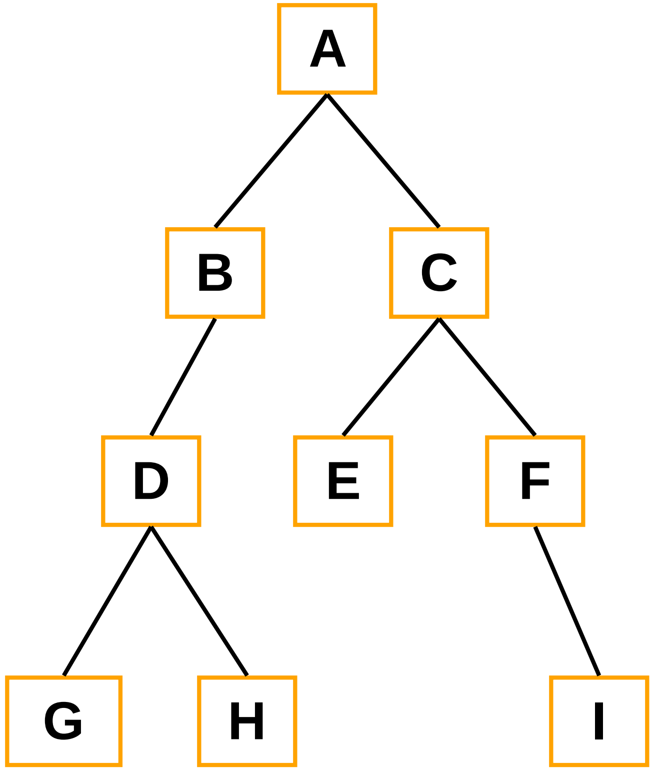
Level Order Algorithm

- Use a *queue* to track unvisited nodes

- For each node that is dequeued,
  - enqueue each of its children
  - until queue empty

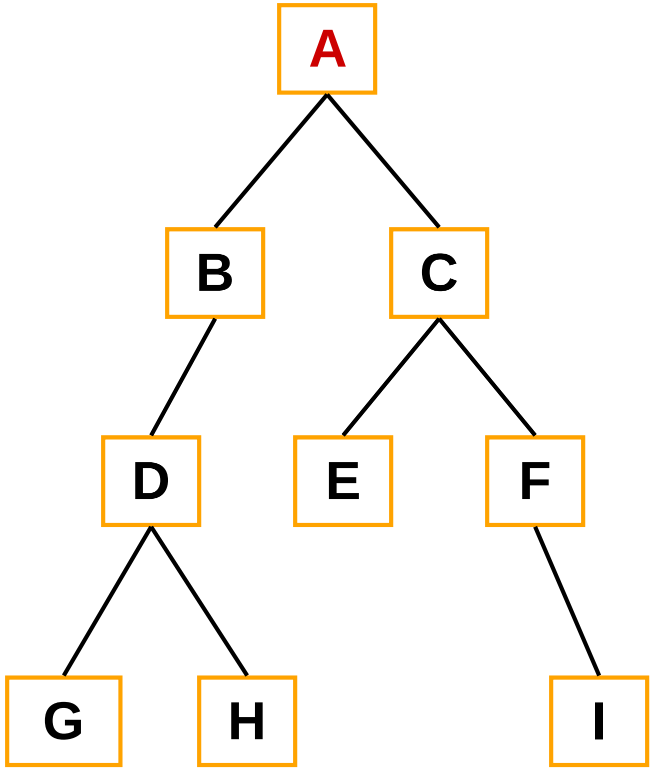
- Also called: Breadth-First traversal

LevelOrder

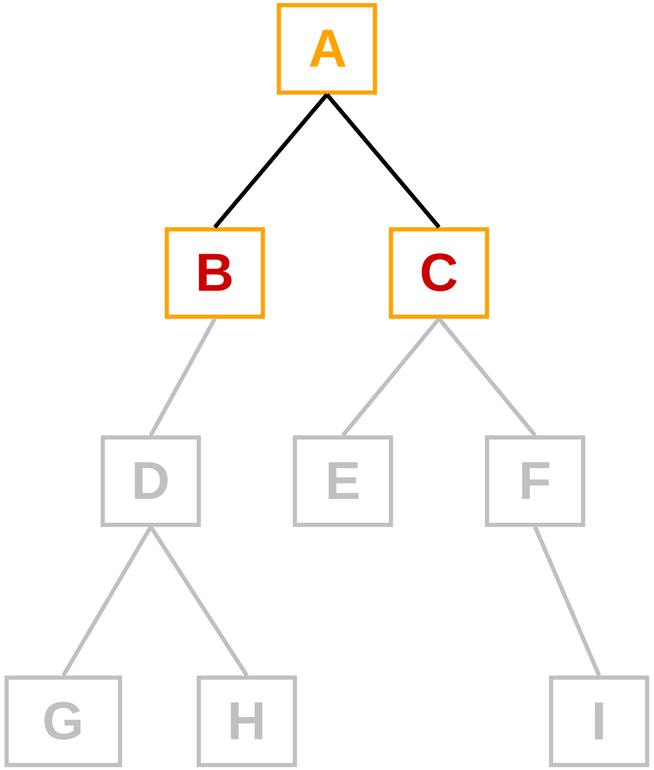


Queue Output

LevelOrder

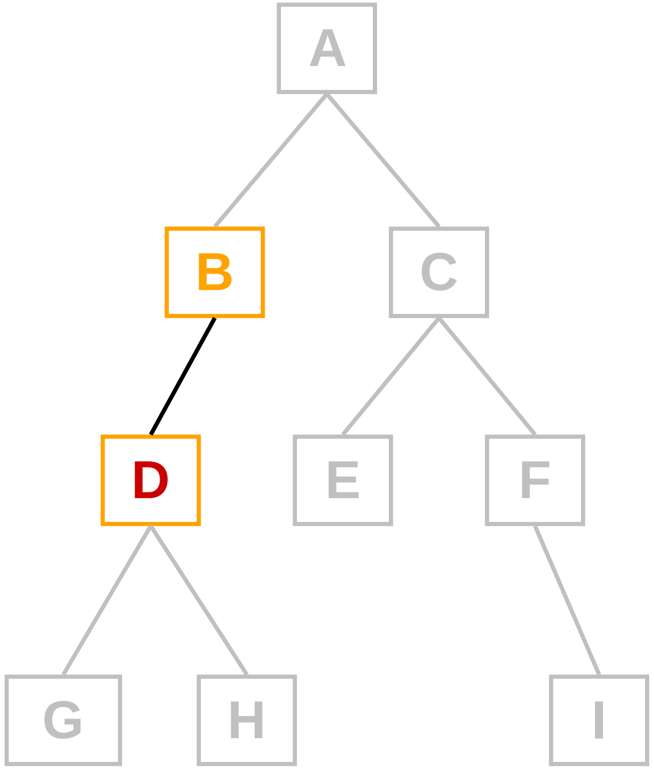


	Queue	Output
Init	[A]	-



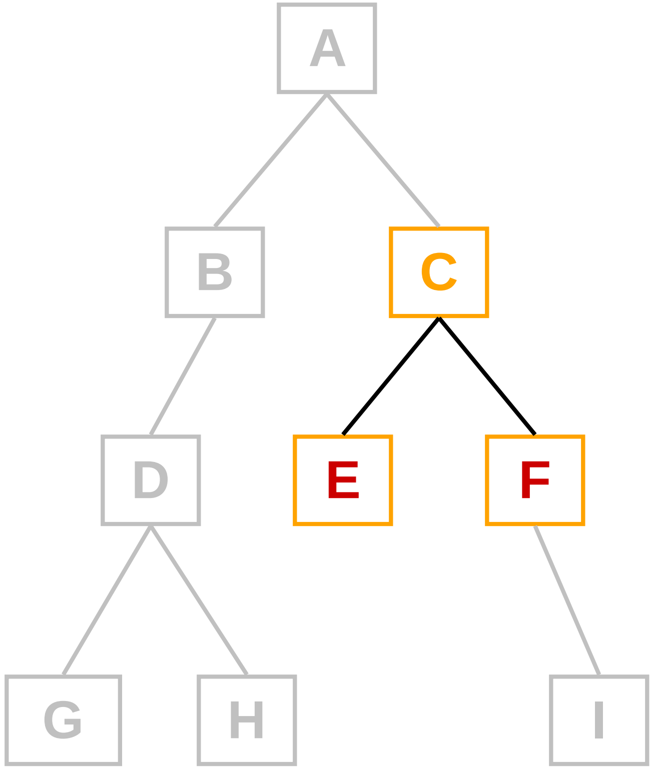
	Queue	Output
Init	[A]	-
Step 1	[B,C]	**A**

Dequeue A
Print A
Enqueue children of A



	Queue	Output
Init	[A]	-
Step 1	[B,C]	A
Step 2	[C,D]	A **B**

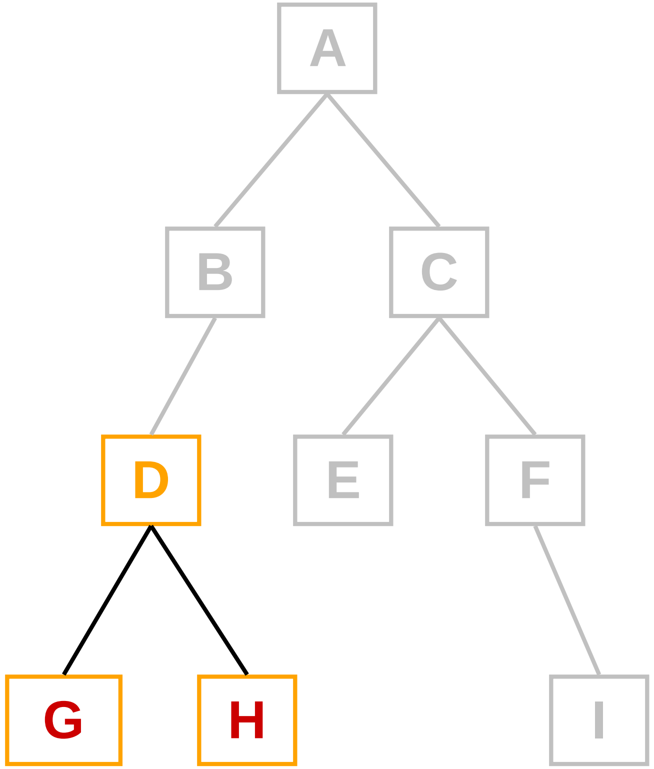
Dequeue B
Print B
Enqueue children of B



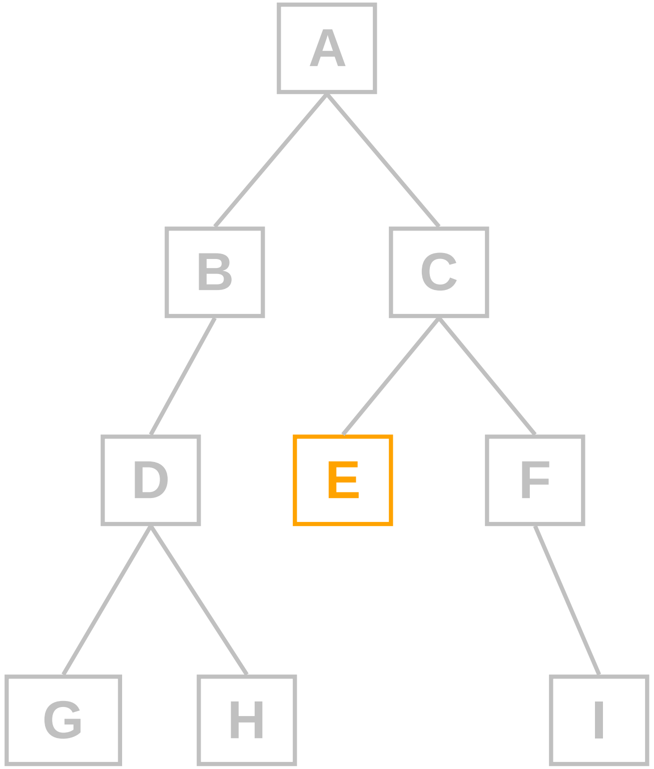
	Queue	Output
Init	[A]	-
Step 1	[B,C]	A
Step 2	[C,D]	A B
Step 3	[D,E,F]	A B **C**

Dequeue C
Print C
Enqueue children of C

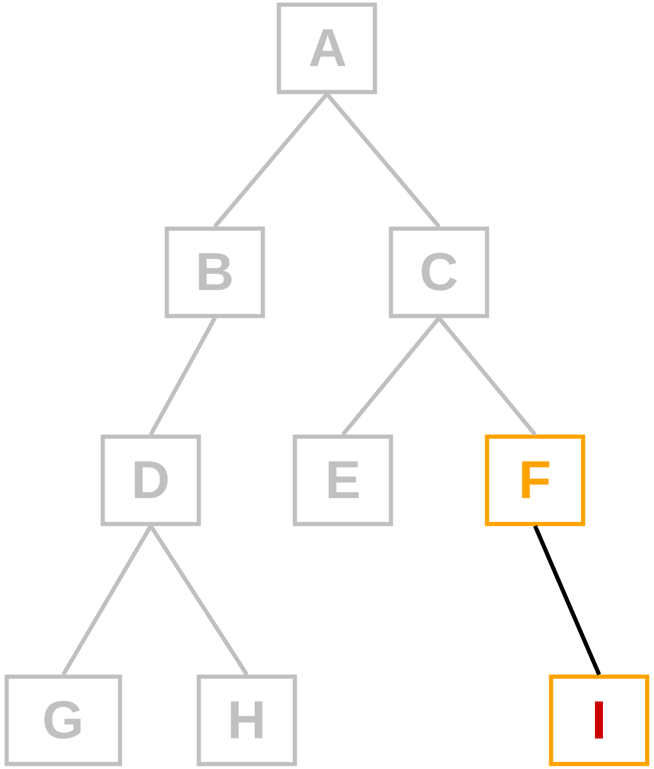
...



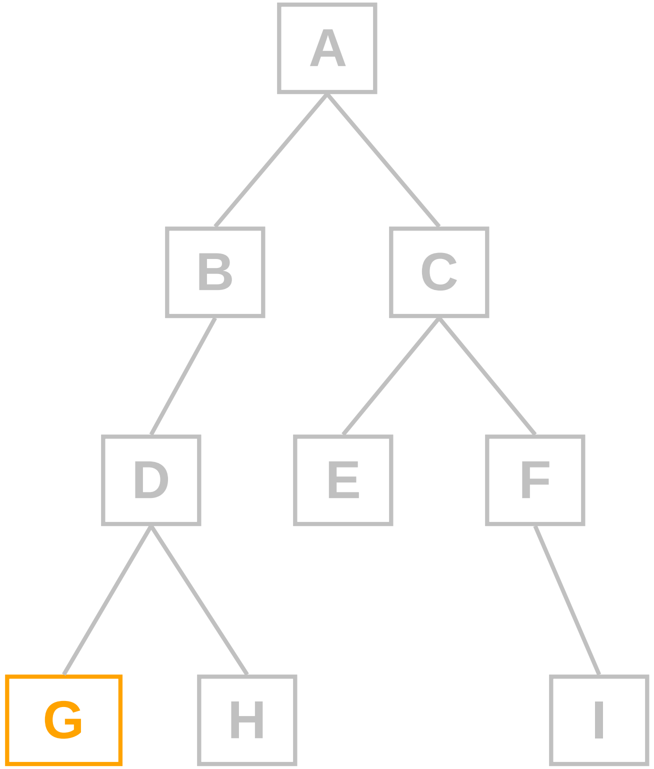
	Queue	Output
Init	[A]	-
Step 1	[B,C]	A
Step 2	[C,D]	A B
Step 3	[D,E,F]	A B C
Step 4	[E,F,G,H]	A B C **D**



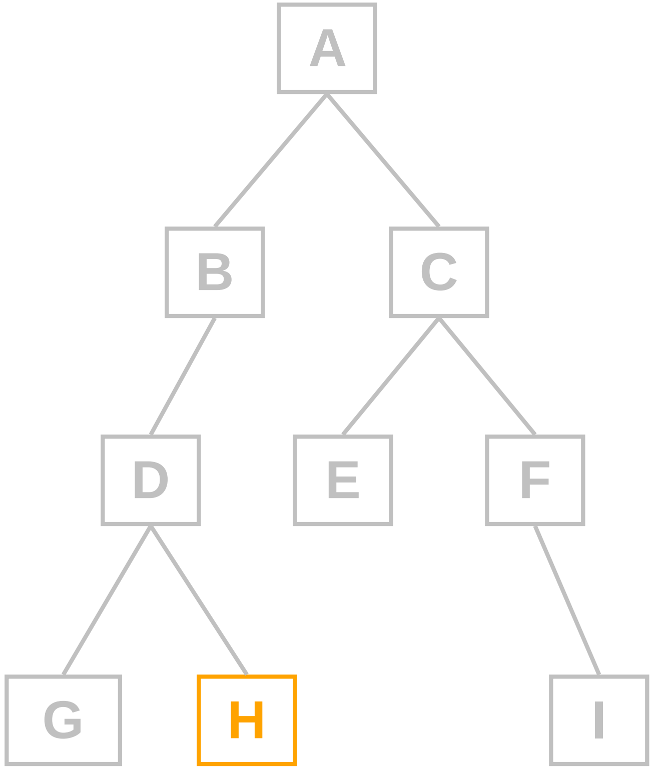
	Queue	Output
Init	[A]	-
Step 1	[B,C]	A
Step 2	[C,D]	A B
Step 3	[D,E,F]	A B C
Step 4	[E,F,G,H]	A B C D
Step 5	[F,G,H]	A B C D **E**



	Queue	Output
Init	[A]	-
Step 1	[B,C]	A
Step 2	[C,D]	A B
Step 3	[D,E,F]	A B C
Step 4	[E,F,G,H]	A B C D
Step 5	[F,G,H]	A B C D E
Step 6	[G,H,I]	A B C D E **F**

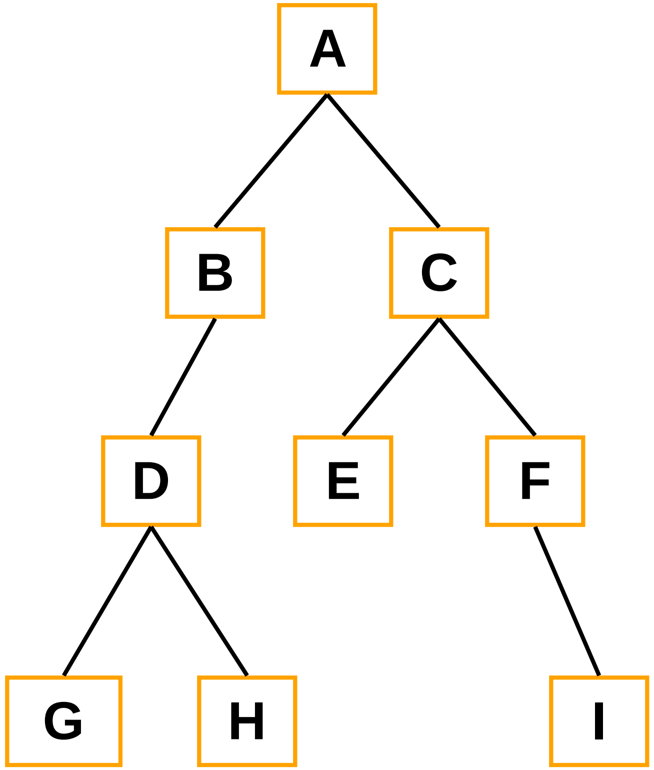


	Queue	Output
Init	[A]	-
Step 1	[B,C]	A
Step 2	[C,D]	A B
Step 3	[D,E,F]	A B C
Step 4	[E,F,G,H]	A B C D
Step 5	[F,G,H]	A B C D E
Step 6	[G,H,I]	A B C D E F
Step 7	[H,I]	A B C D E F **G**



	Queue	Output
Init	[A]	-
Step 1	[B,C]	A
Step 2	[C,D]	A B
Step 3	[D,E,F]	A B C
Step 4	[E,F,G,H]	A B C D
Step 5	[F,G,H]	A B C D E
Step 6	[G,H,I]	A B C D E F
Step 7	[H,I]	A B C D E F G
Step 8	[I]	A B C D E F G **H**

LevelOrder



	Queue	Output
Init	[A]	-
Step 1	[B,C]	A
Step 2	[C,D]	A B
Step 3	[D,E,F]	A B C
Step 4	[E,F,G,H]	A B C D
Step 5	[F,G,H]	A B C D E
Step 6	[G,H,I]	A B C D E F
Step 7	[H,I]	A B C D E F G
Step 8	[I]	A B C D E F G H
Step 9	[ ]	A B C D E F G H I

The Scenario

- We have a Binary Search Tree and want to remove some element based upon a match.

- Must preserve “search” property
- Must not lose any elements (i.e. only remove the one element)

BST Deletion

UNIVERSITY OF CALGARY

- Search for desired item.

- If not found, then return NULL or print error.

- If found, perform steps necessary to accomplish removal from the tree.

Four Cases for Deletion

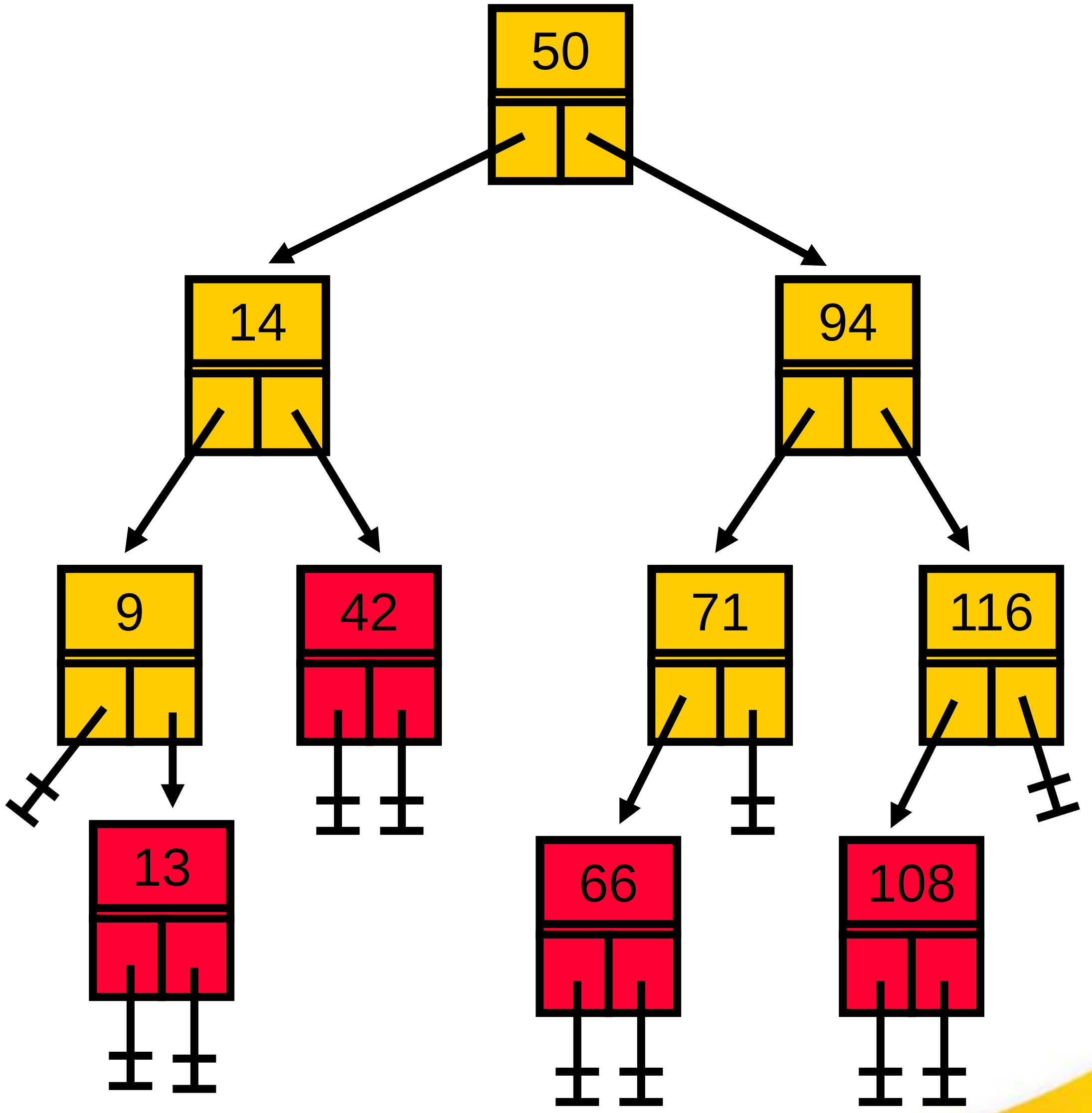
- Delete a leaf node
- Delete a node with only one child (left)
- Delete a node with only one child (right)
- Delete a node with two children

- Cases 2 and 3 are comparable and only need slight changes in the conditional statement used

Delete a Leaf Node

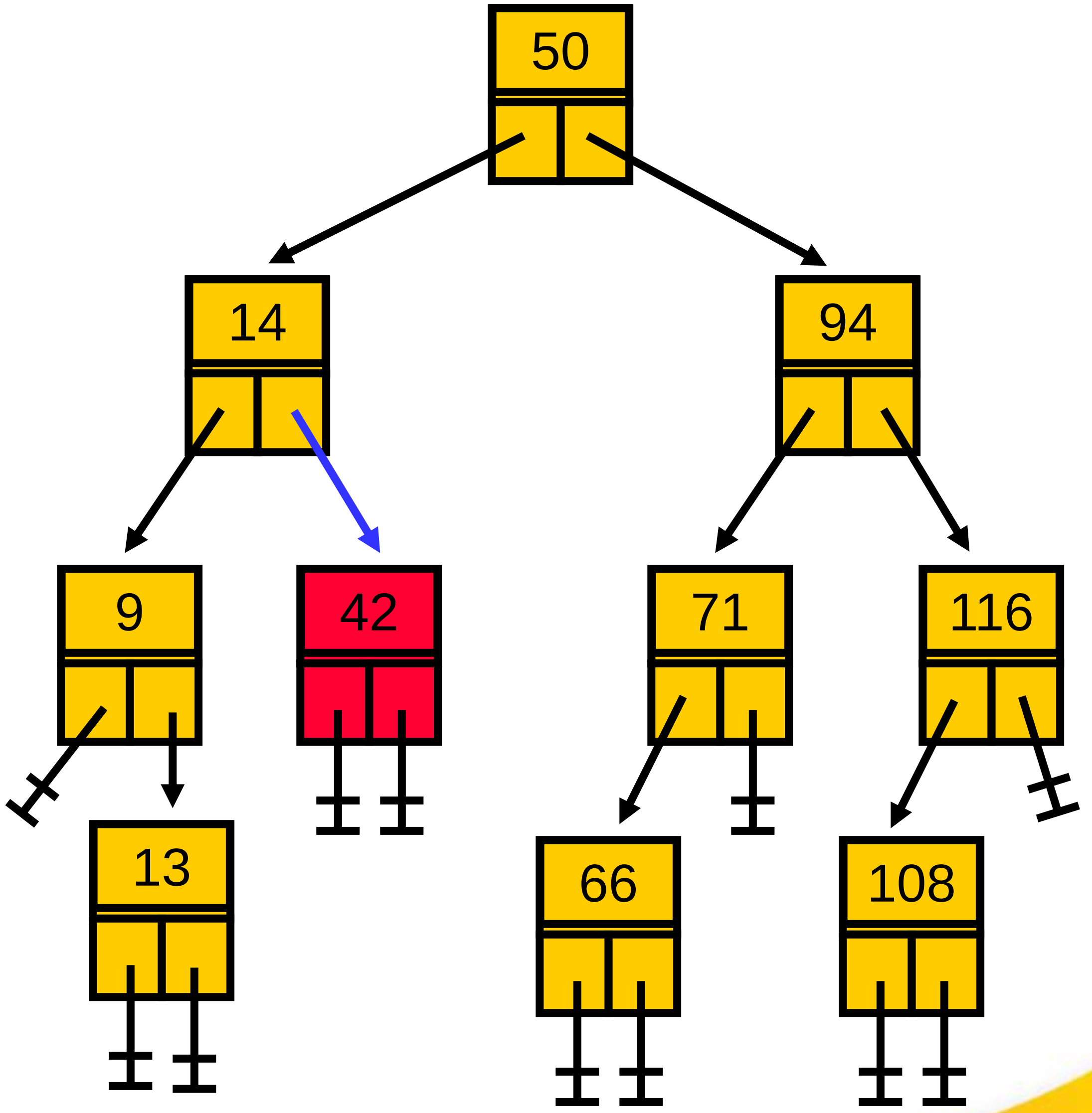
Set the parent node's child pointer to null

This will remove the node from the tree.

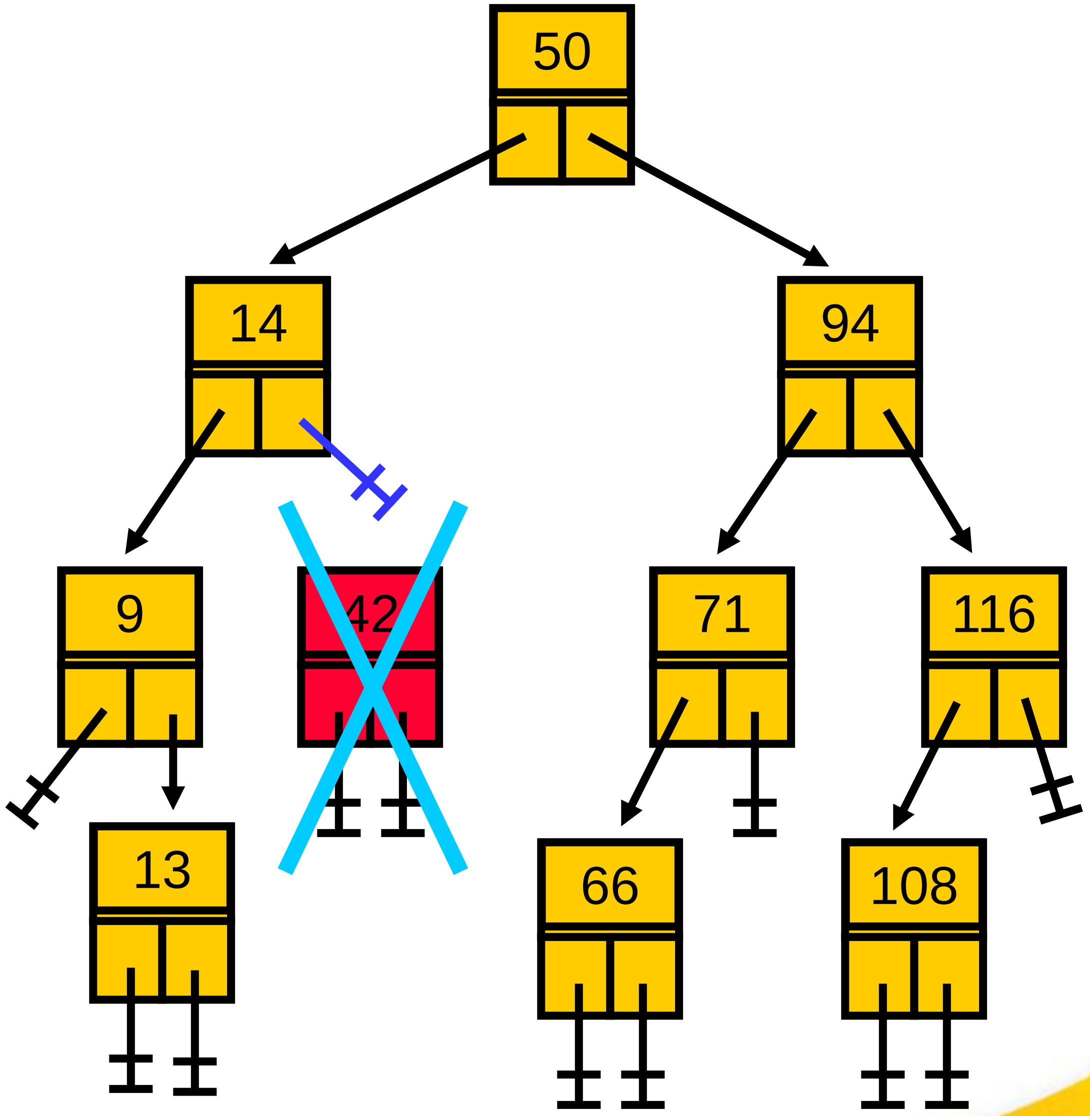


Delete a Leaf Node

Let's delete 42.

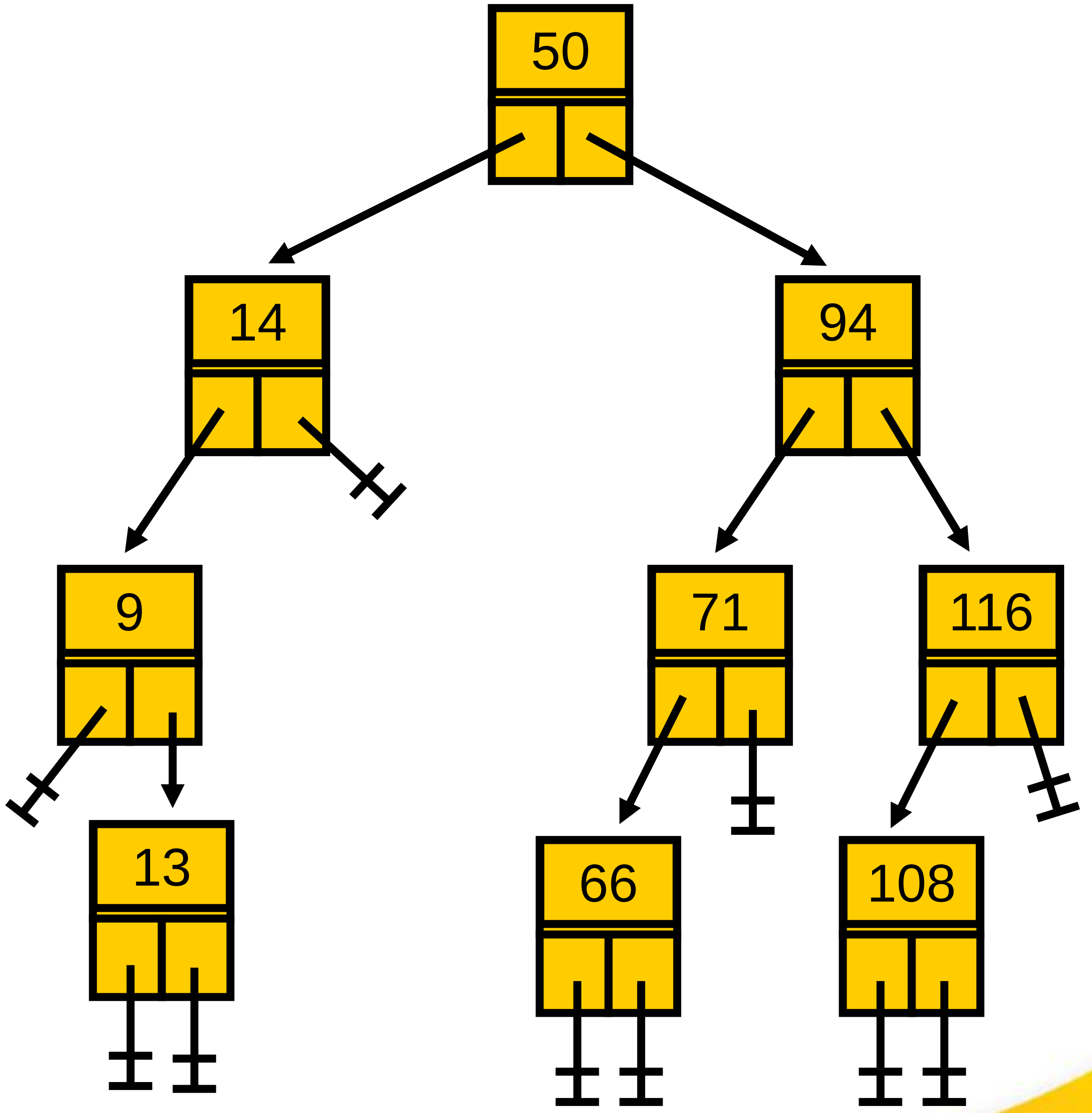


Delete a Leaf Node

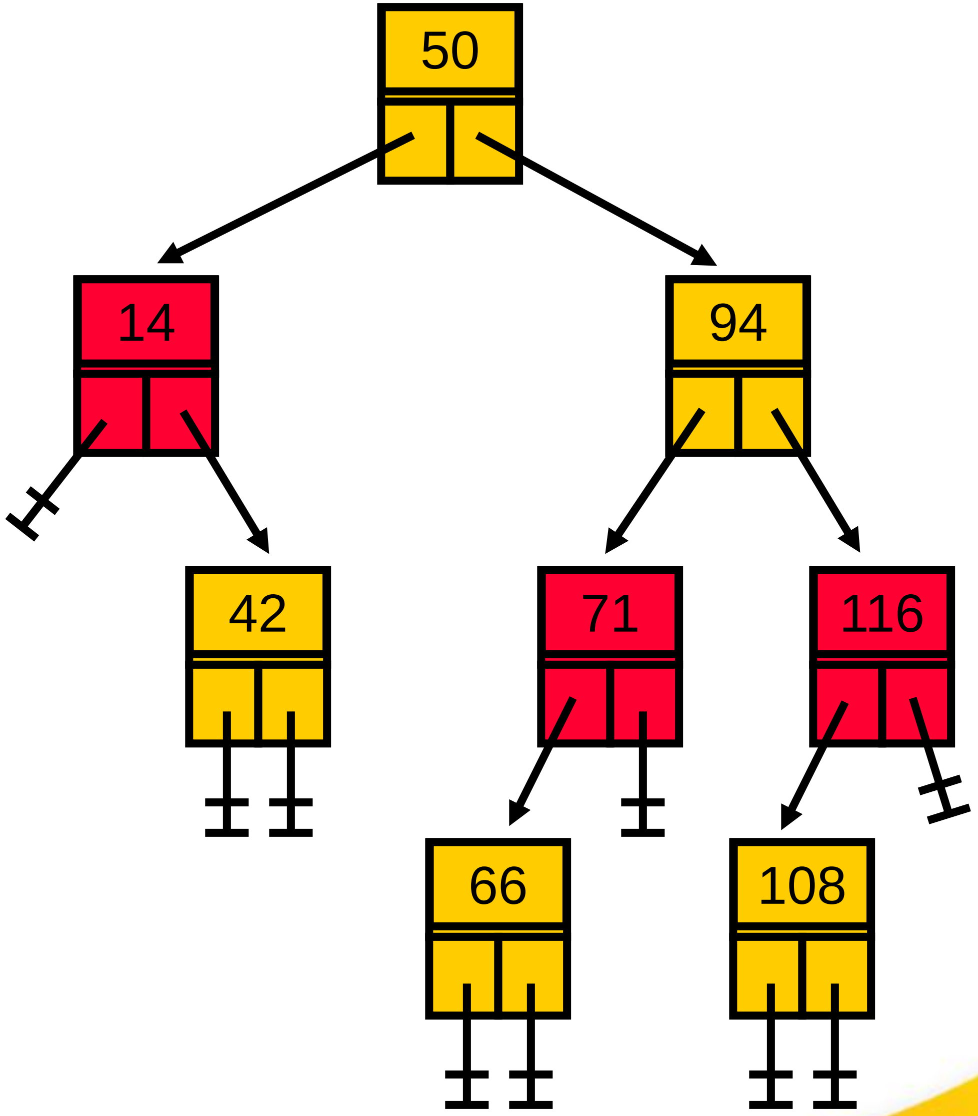


Delete a Leaf Node

The resulting tree.



Delete a Node with One Child



Determine if it has a left or a right child.

Point the current pointer to the appropriate child:

```
cur <- cur^.left_child
```

or

```
cur <- cur^.right_child
```

Delete a Node with One Child

Determine if it has a left or a right child.

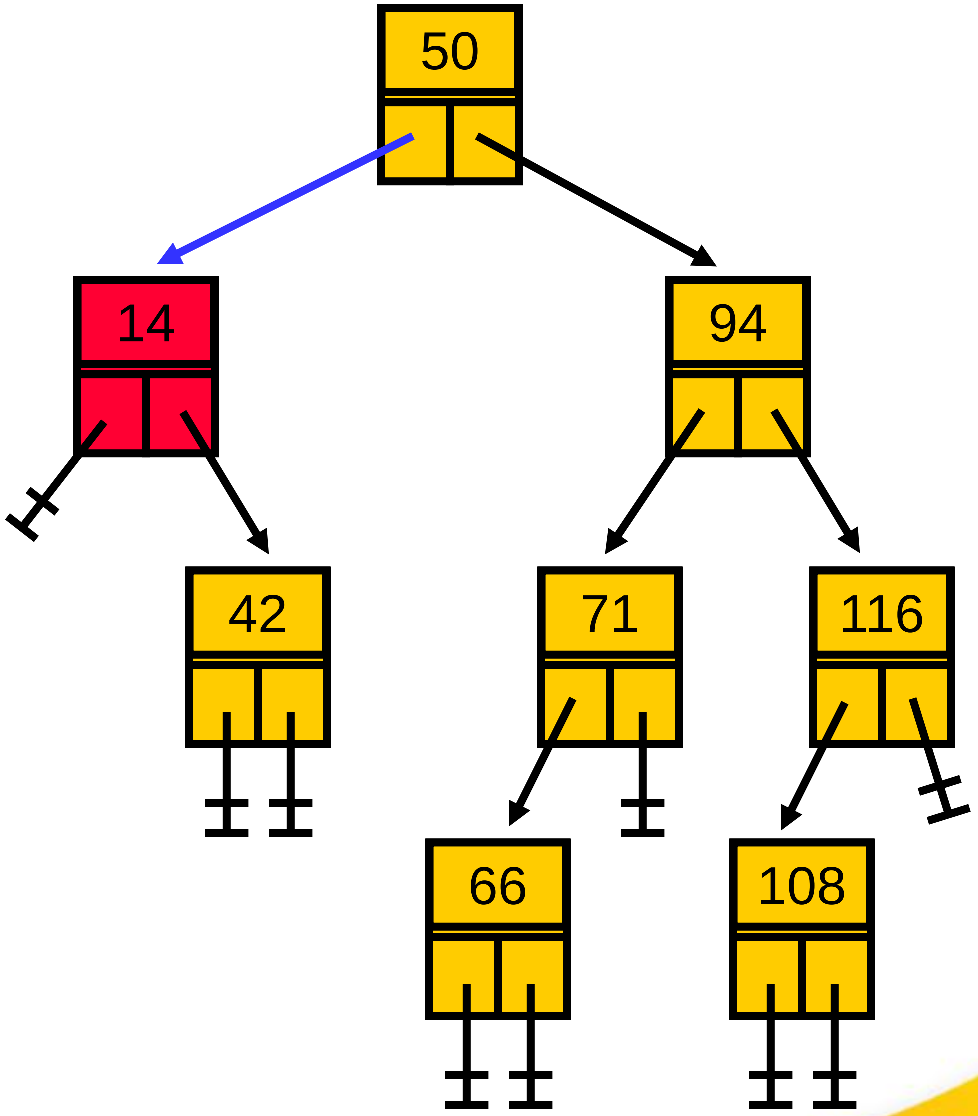
Point the current pointer to the appropriate child:

```
cur <- cur^.left_child
```

or

```
cur <- cur^.right_child
```

Let's delete 14.



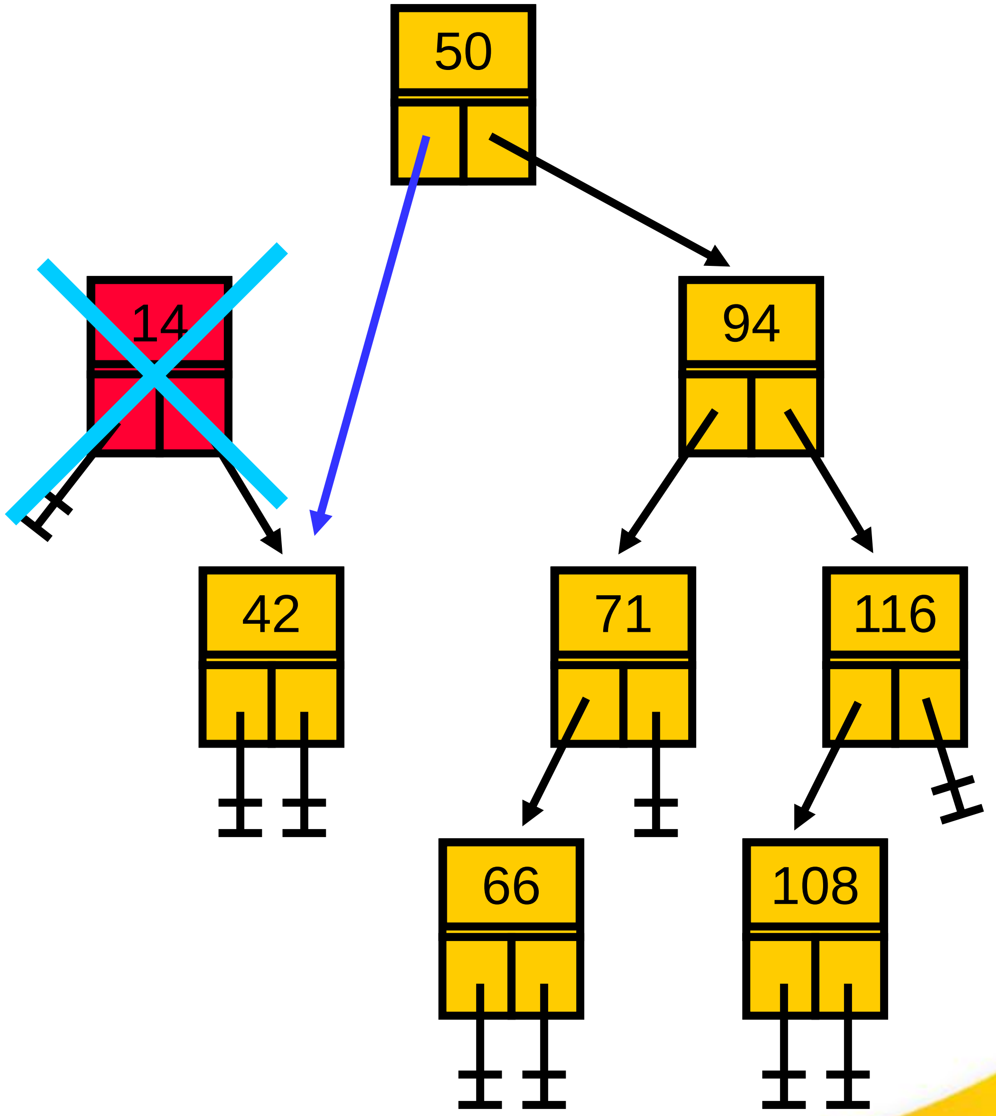
Delete a Node with One Child

Determine if it has a left or a right child.

Point the current pointer to the appropriate child:

```
cur <- cur^.right_child
```

Move the pointer; now nothing points to the node.



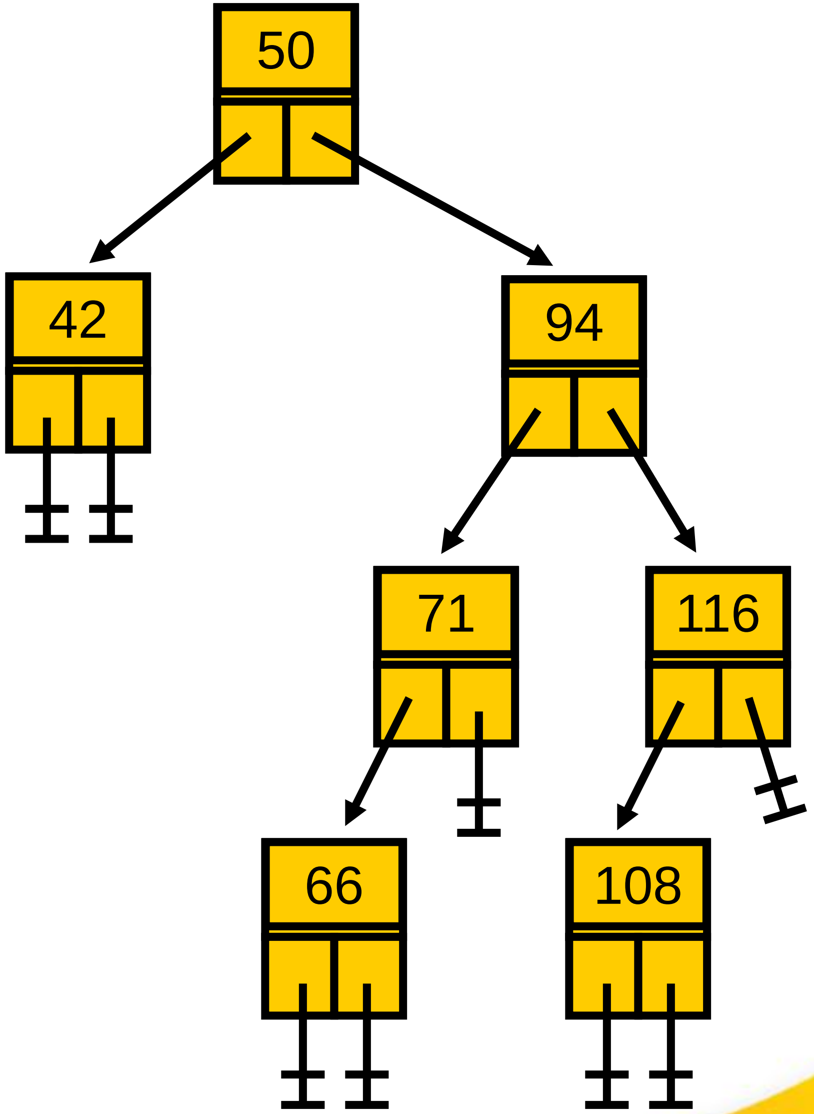
Delete a Node with One Child

Determine if it has a left or a right child.

Point the current pointer to the appropriate child:

```
cur <- cur^.right_child
```

The resulting tree.



Delete a Node with One Child

Determine if it has a left or a right child.

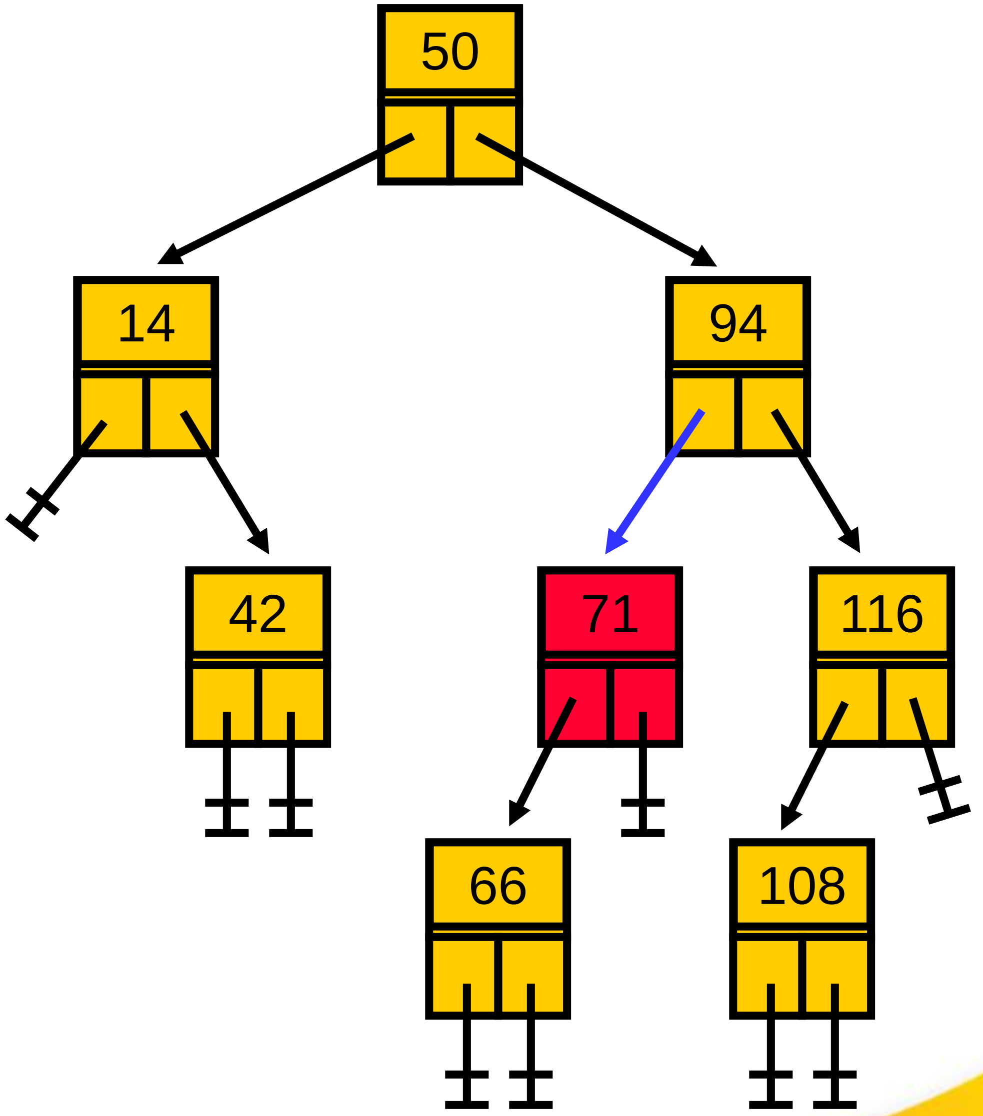
Point the current pointer to the appropriate child:

```
cur <- cur^.left_child
```

or

```
cur <- cur^.right_child
```

Let's delete 71.



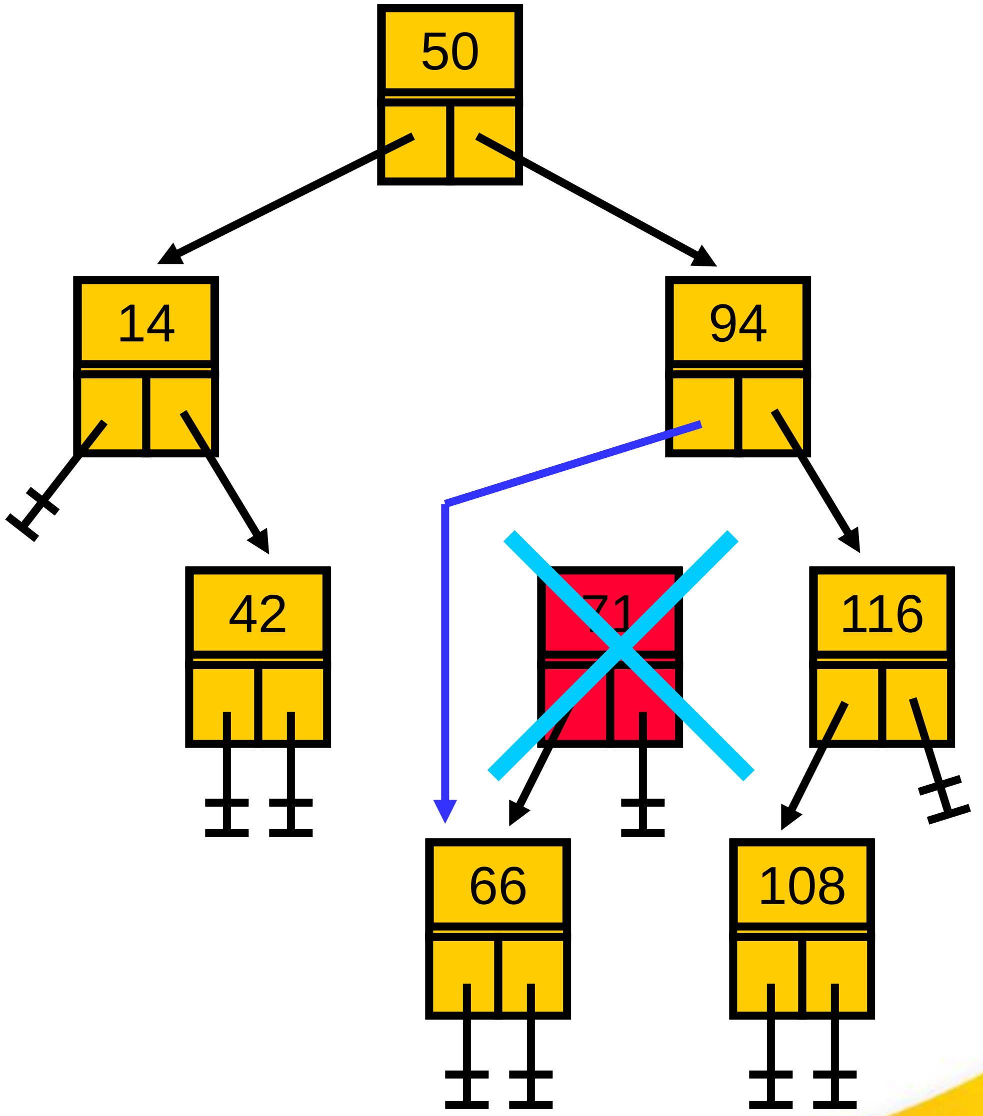
Delete a Node with One Child

Determine if it has a left or a right child.

Point the current pointer to the appropriate child:

```
cur <- cur^.left_child
```

Move the pointer; now nothing points to the node.



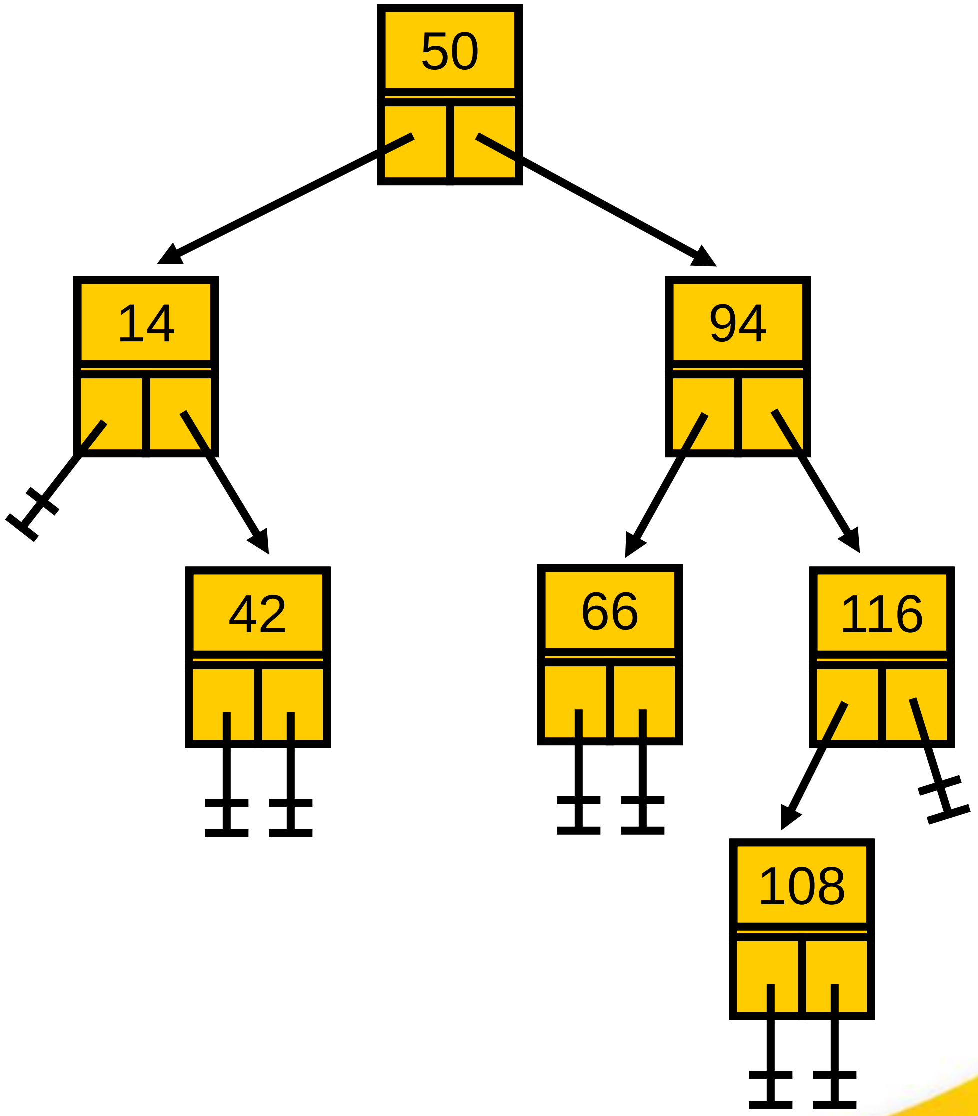
Delete a Node with One Child

Determine if it has a left or a right child.

Point the current pointer to the appropriate child:

```
cur <- cur^.left_child
```

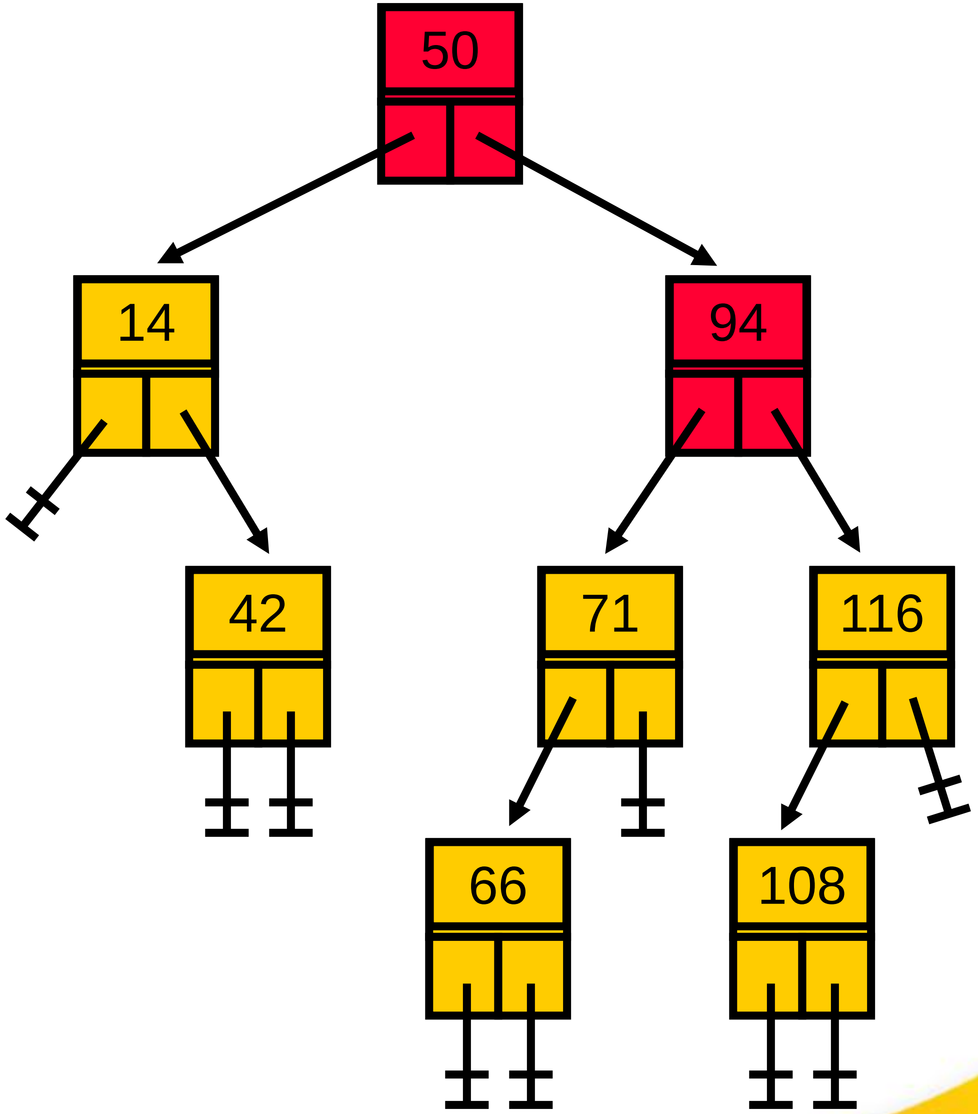
The resulting tree.



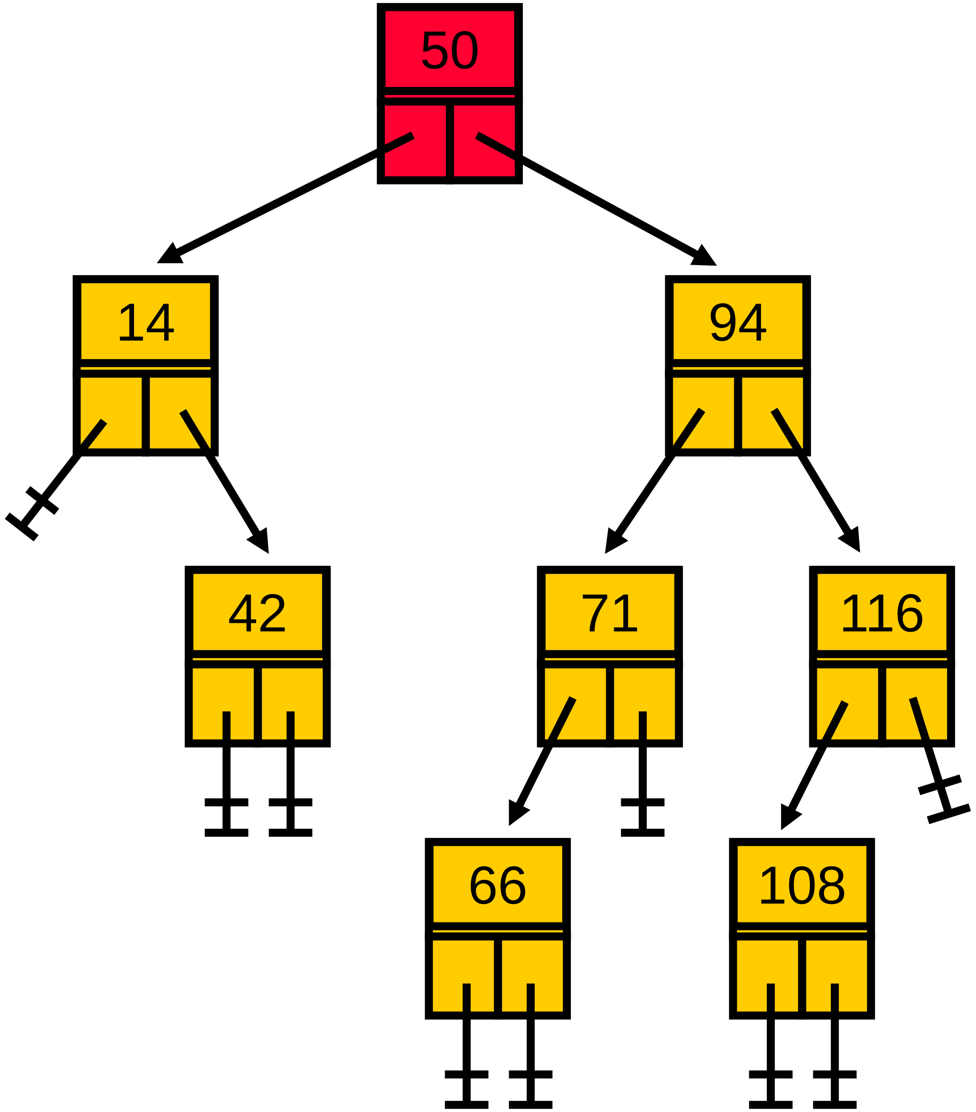
Delete a Node with Two Children

Copy a replacement value from a descendant node.
- Largest from left
- Smallest from right

Then delete that descendant node to remove the duplicate value.
- We know this will be an easier case.

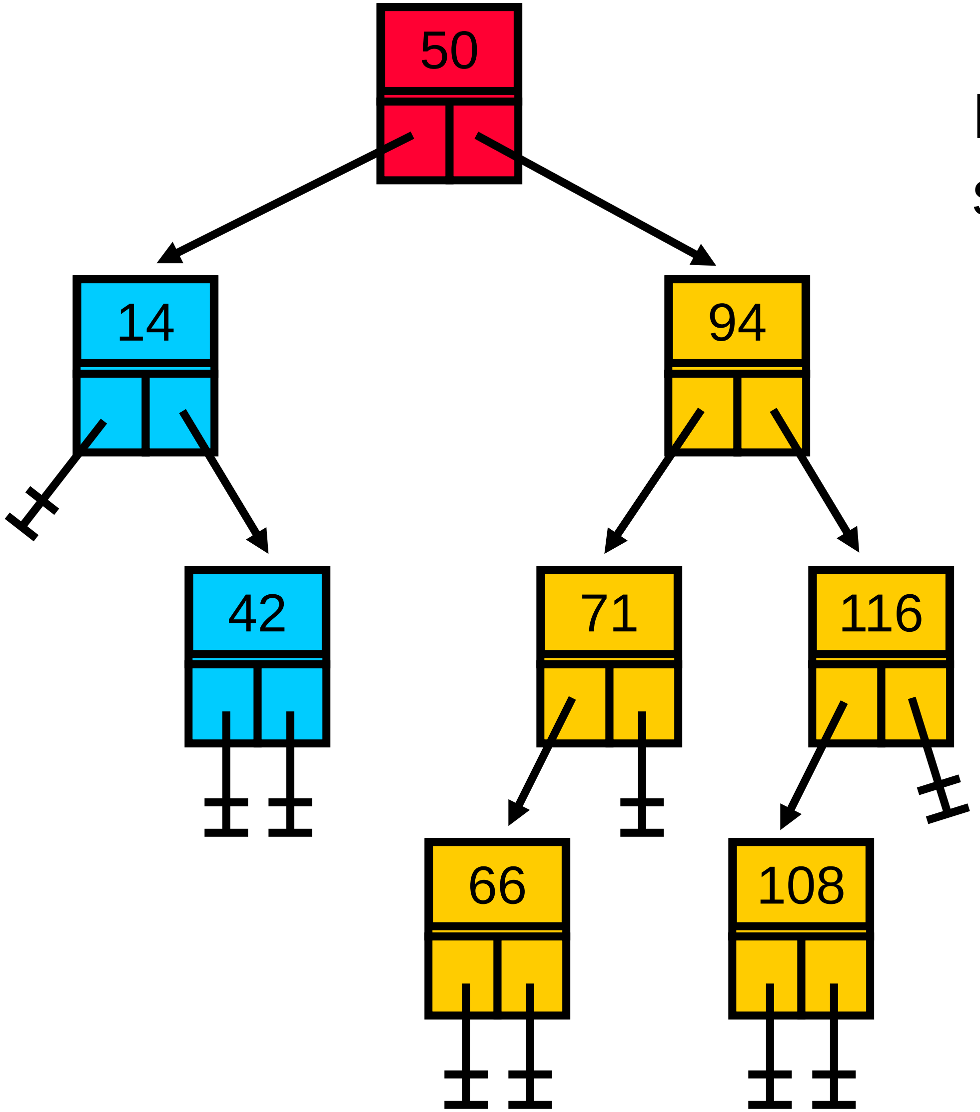


Delete a Node with Two Children



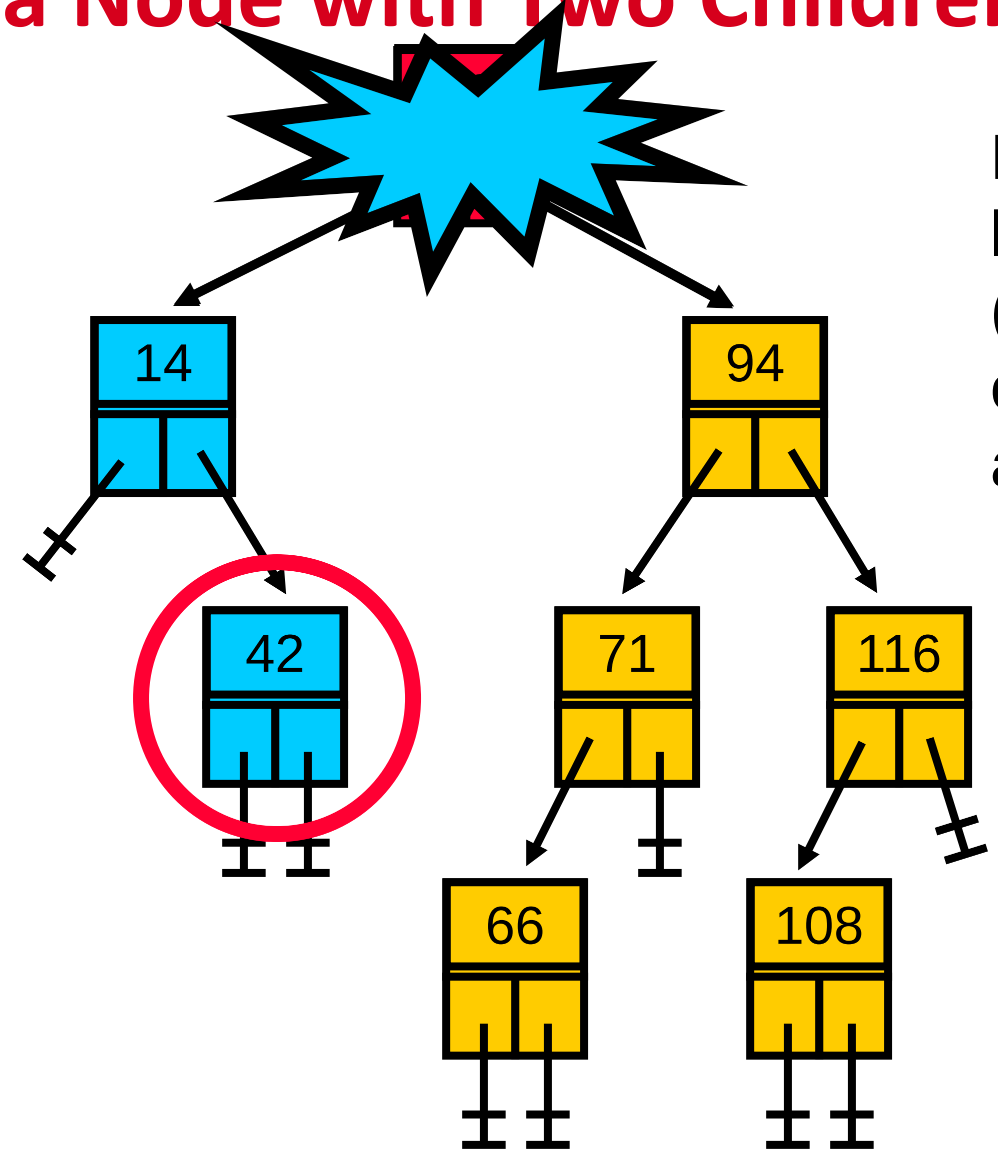
Let’s delete 50.

Delete a Node with Two Children



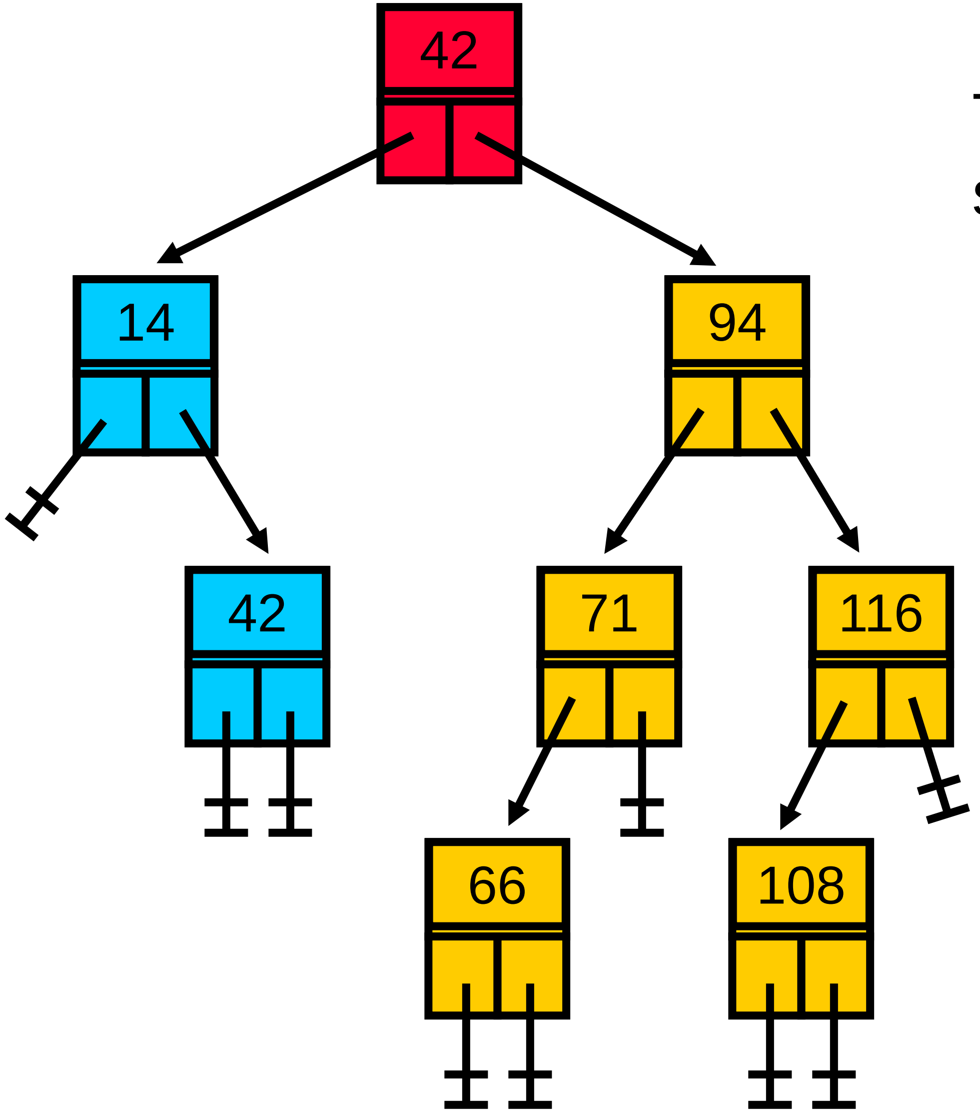
Look to the left sub-tree.

Delete a Node with Two Children



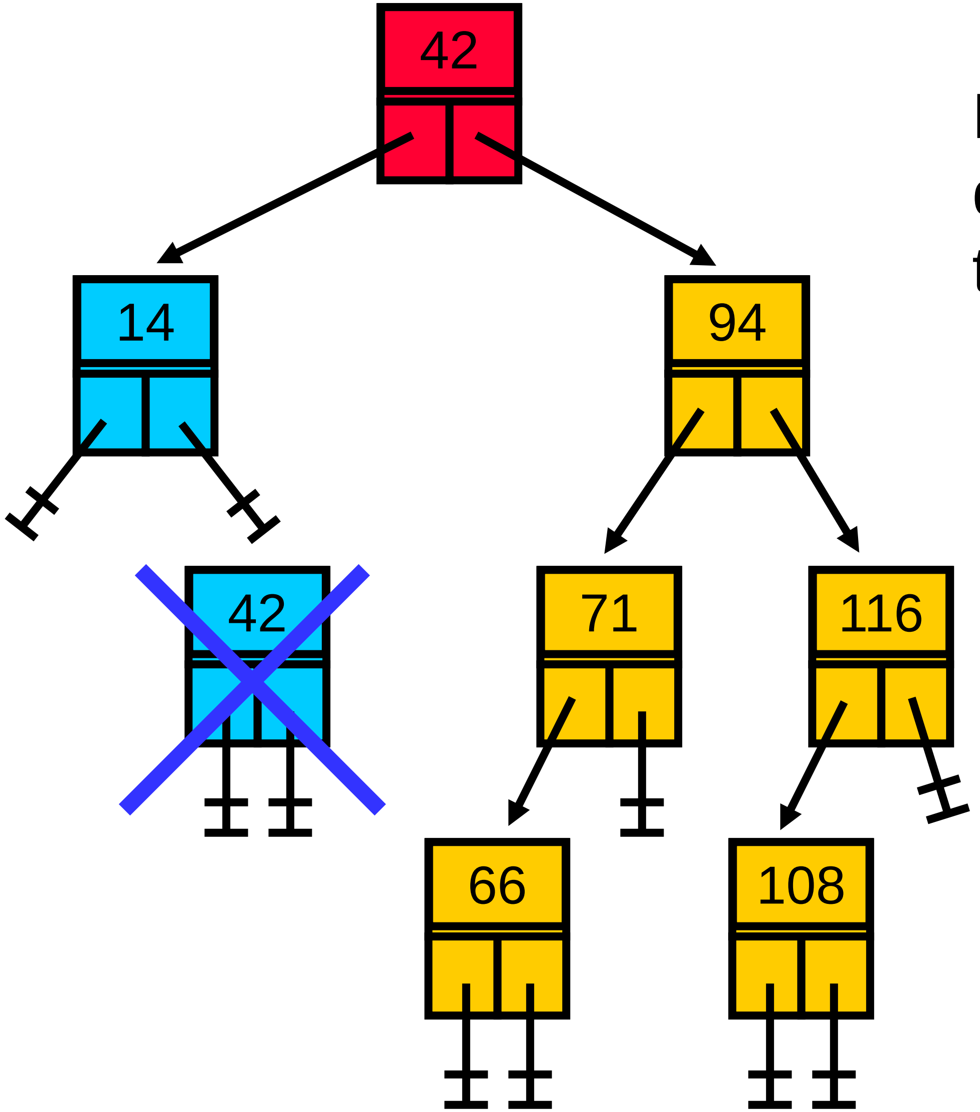
Find and copy the largest value
(this will erase the old value but creates a duplicate).

Delete a Node with Two Children



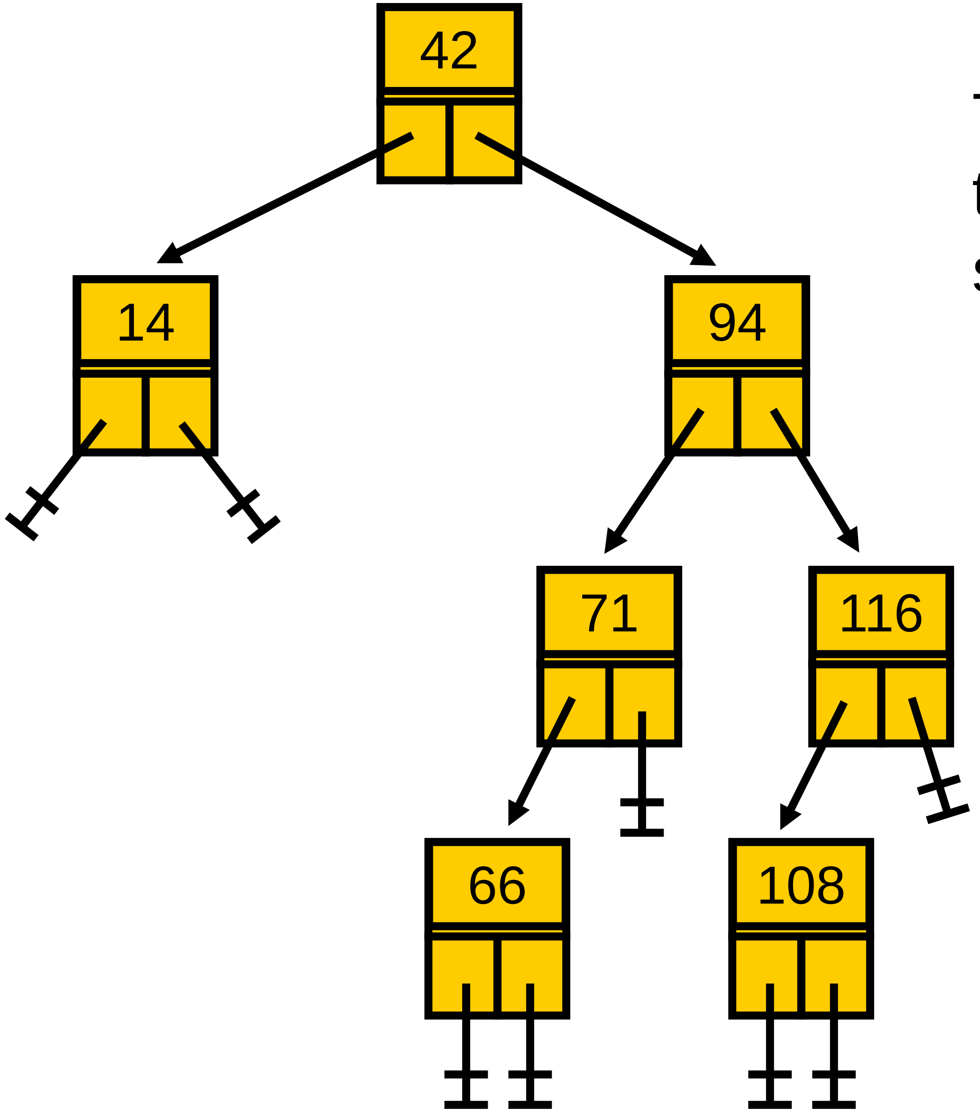
The resulting tree so far.

Delete a Node with Two Children



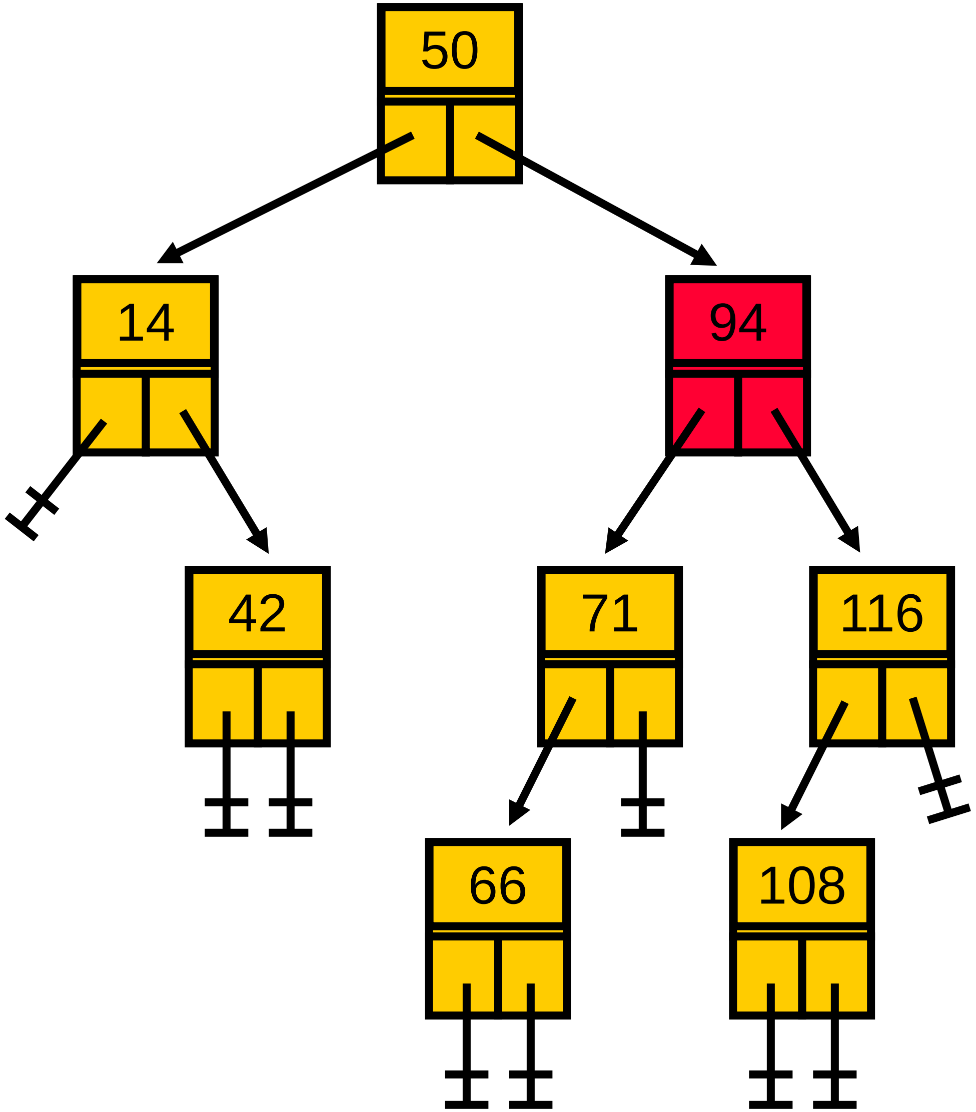
Now delete the duplicate from the left sub-tree.

Delete a Node with Two Children



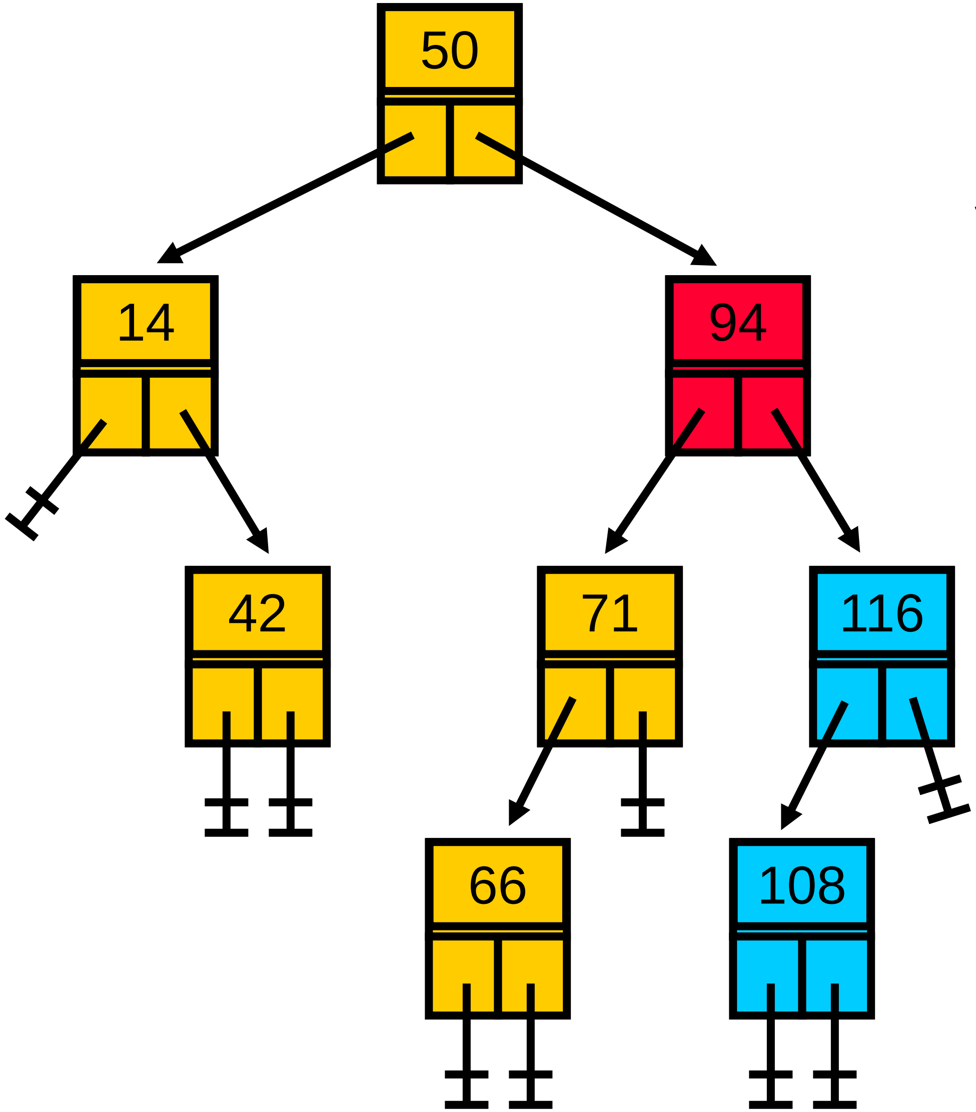
The final resulting tree – still has search structure.

Delete a Node with Two Children



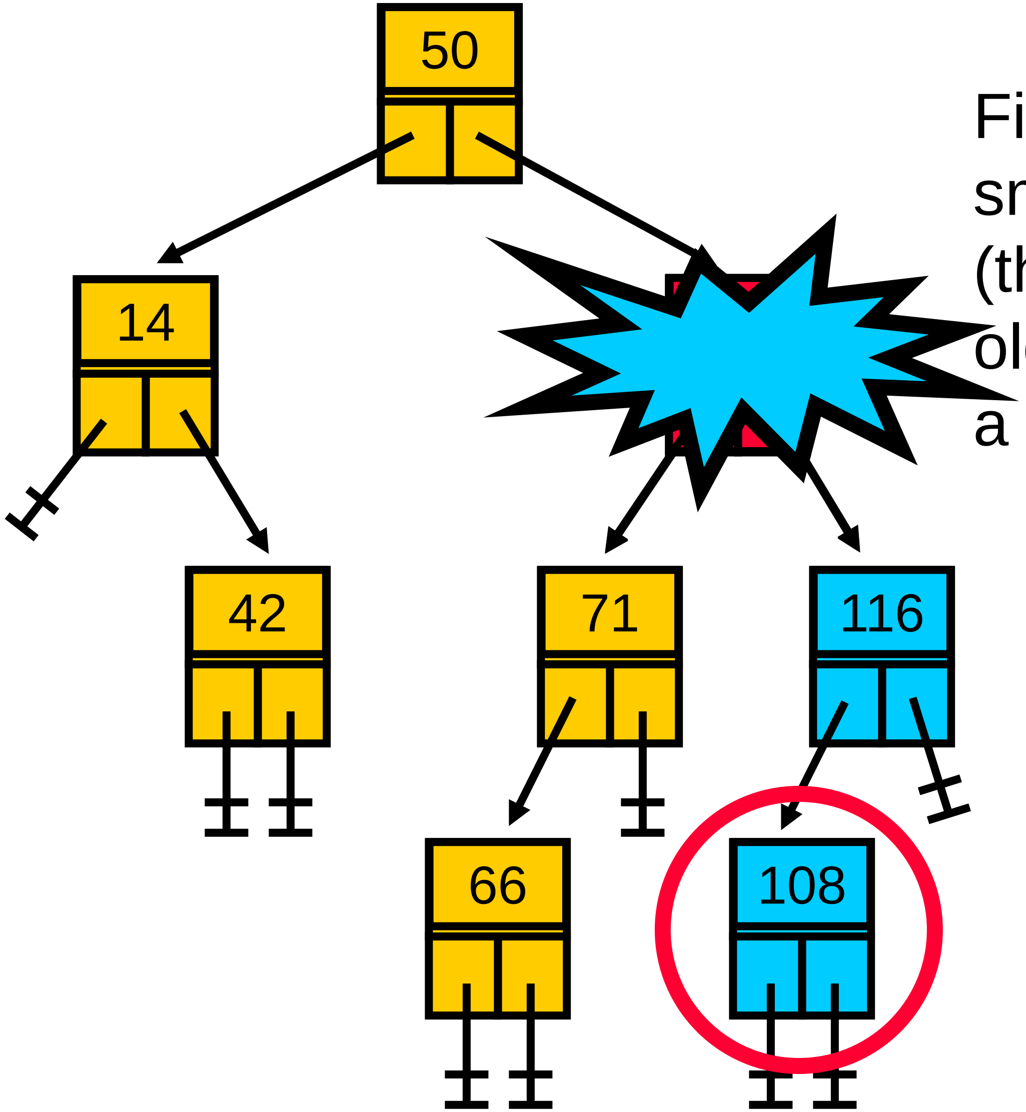
Let's delete 94.

Delete a Node with Two Children



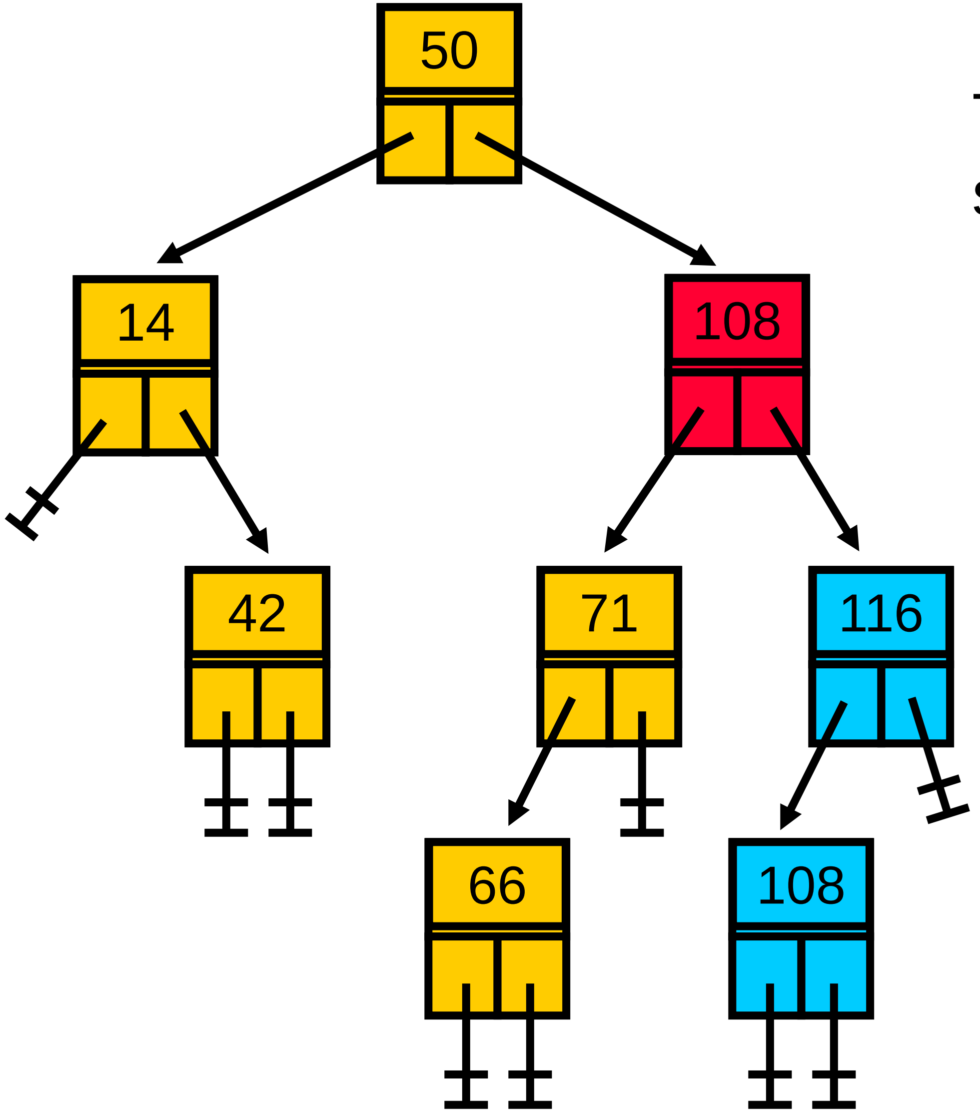
Look to the right sub-tree.

Delete a Node with Two Children



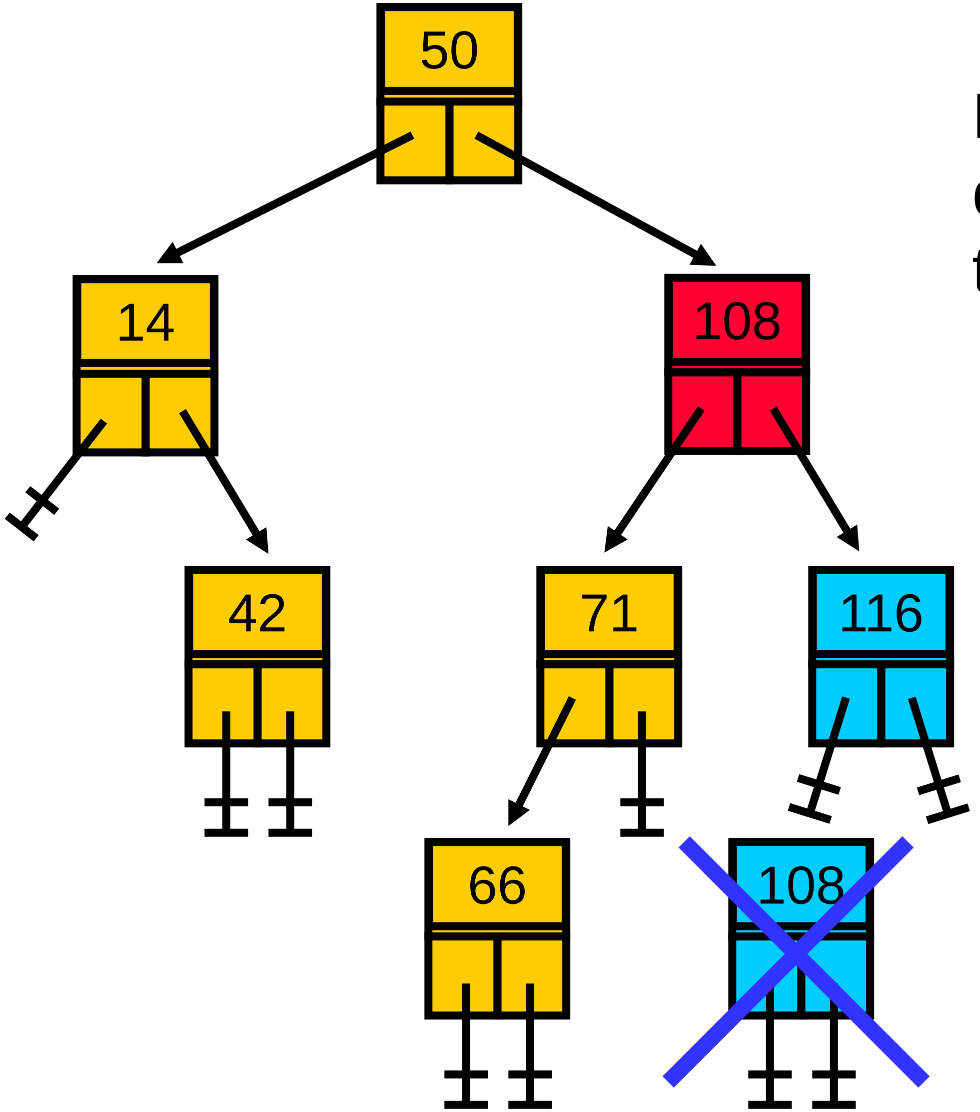
Find and copy the smallest value
(this will erase the old value but creates a duplicate).

Delete a Node with Two Children



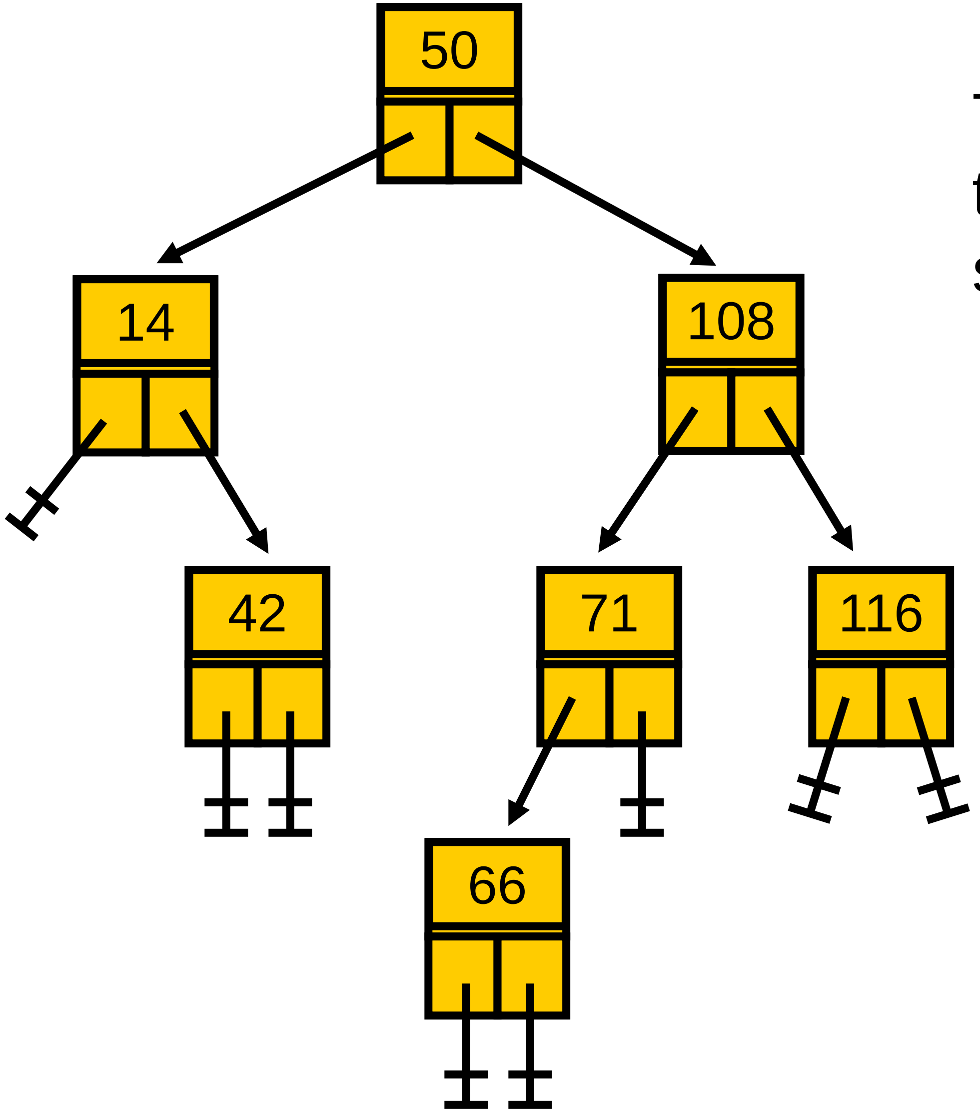
The resulting tree so far.

Delete a Node with Two Children



Now delete the duplicate from the left sub-tree.

Delete a Node with Two Children



The final resulting tree – still has search structure.

Summary

- Deleting a node from a binary search tree involves two steps:
  - Search for the element
  - Then perform the deletion
- We must preserve the search structure and only delete the element which matches.
- Four cases:
  - Deleting a leaf node
  - Deleting a node with only the left child
  - Deleting a node with only the right child
  - Deleting a node with both children