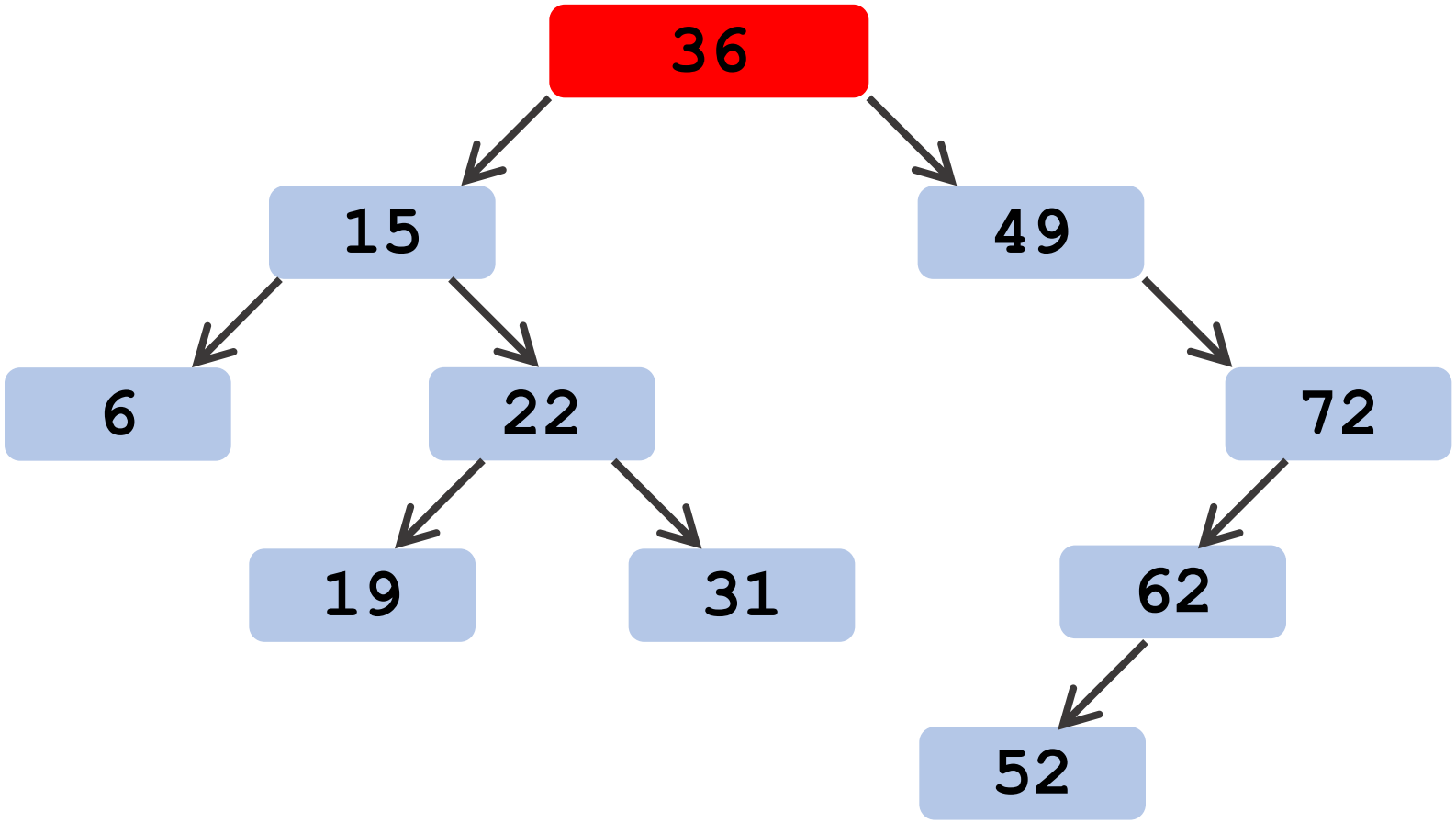
Binary Search Tree (2/2): Successor and Deletion

Shusen Wang

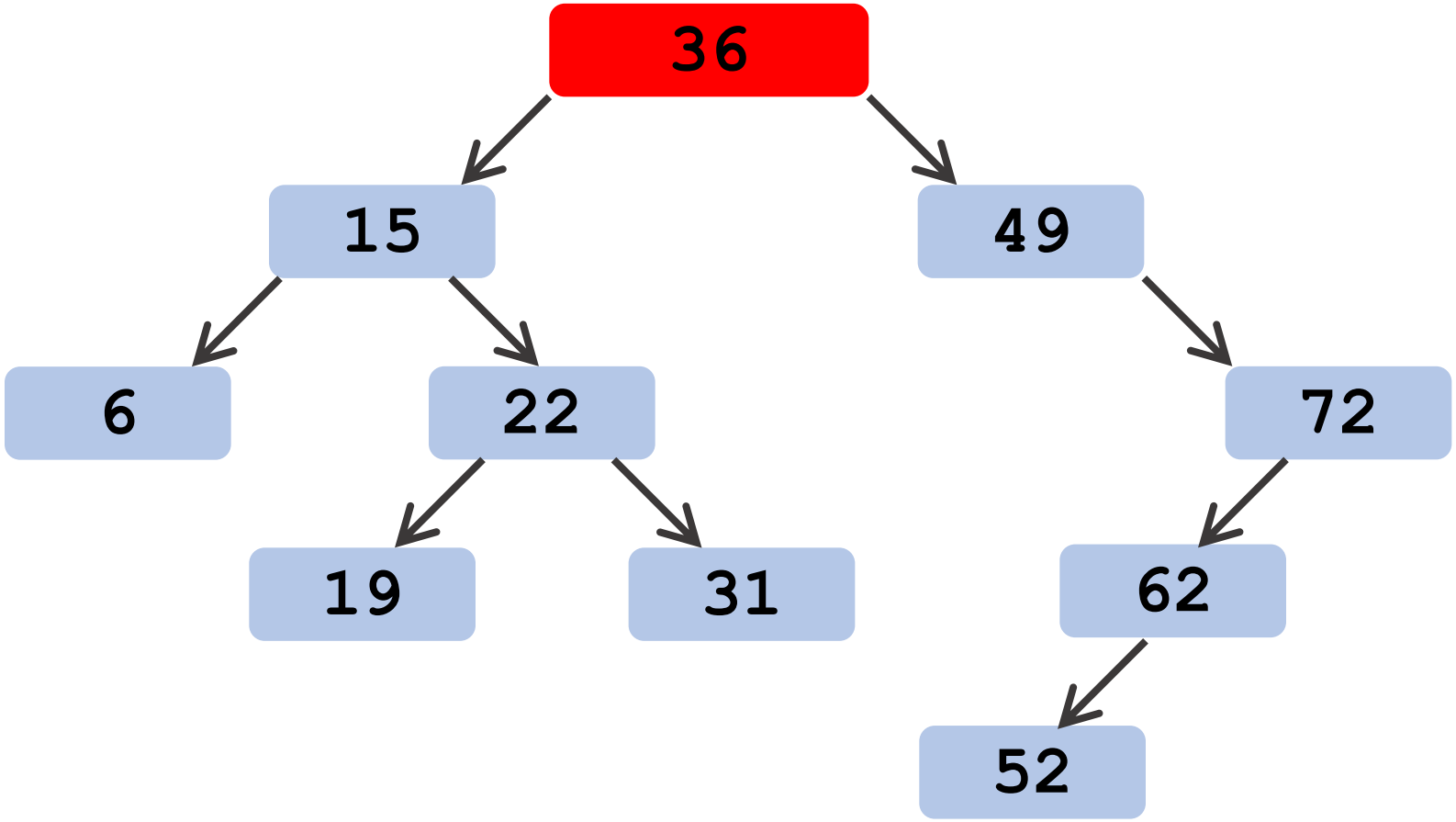
Find Successor

What is the successor of 36?



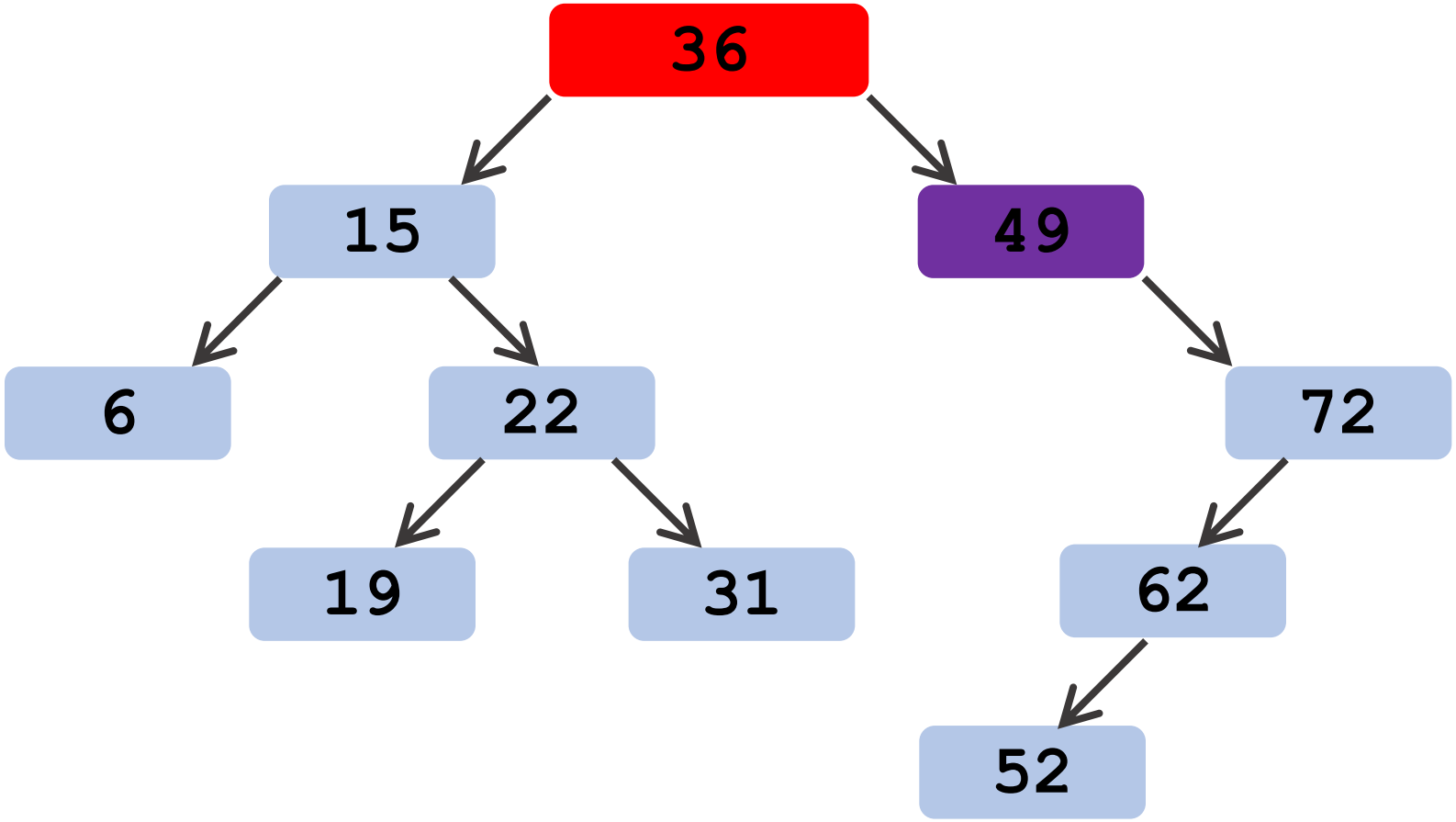
What is the successor of 36?

Inorder traversal: 6 | 15 | 19 | 22 | 31 | **36** | 49 | 52 | 62 | 72



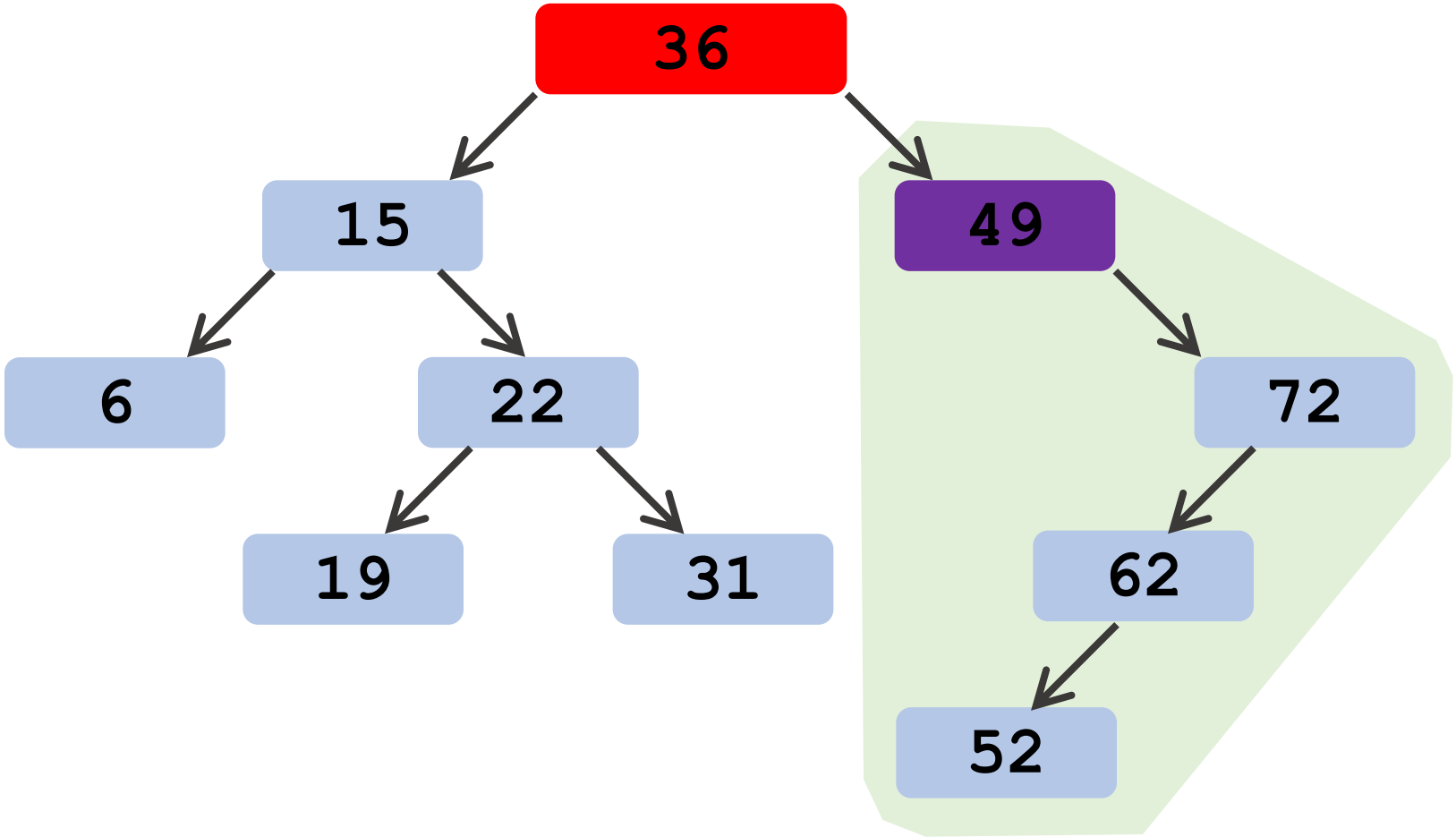
What is the successor of 36?

Inorder traversal: 6 | 15 | 19 | 22 | 31 | 36 | 49 | 52 | 62 | 72

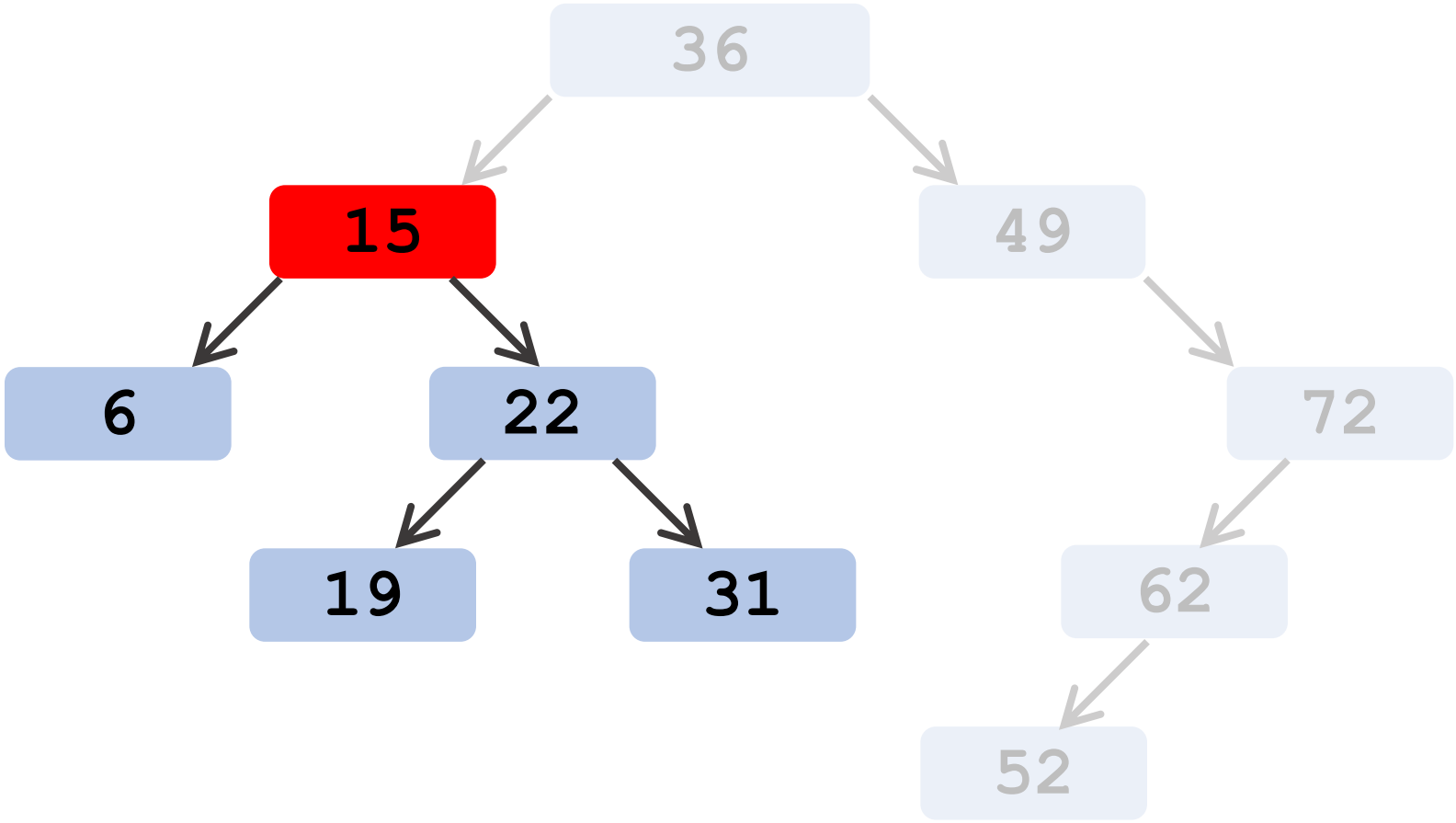


What is the successor of 36?

Successor is the leftmost vertex of the right sub-tree.

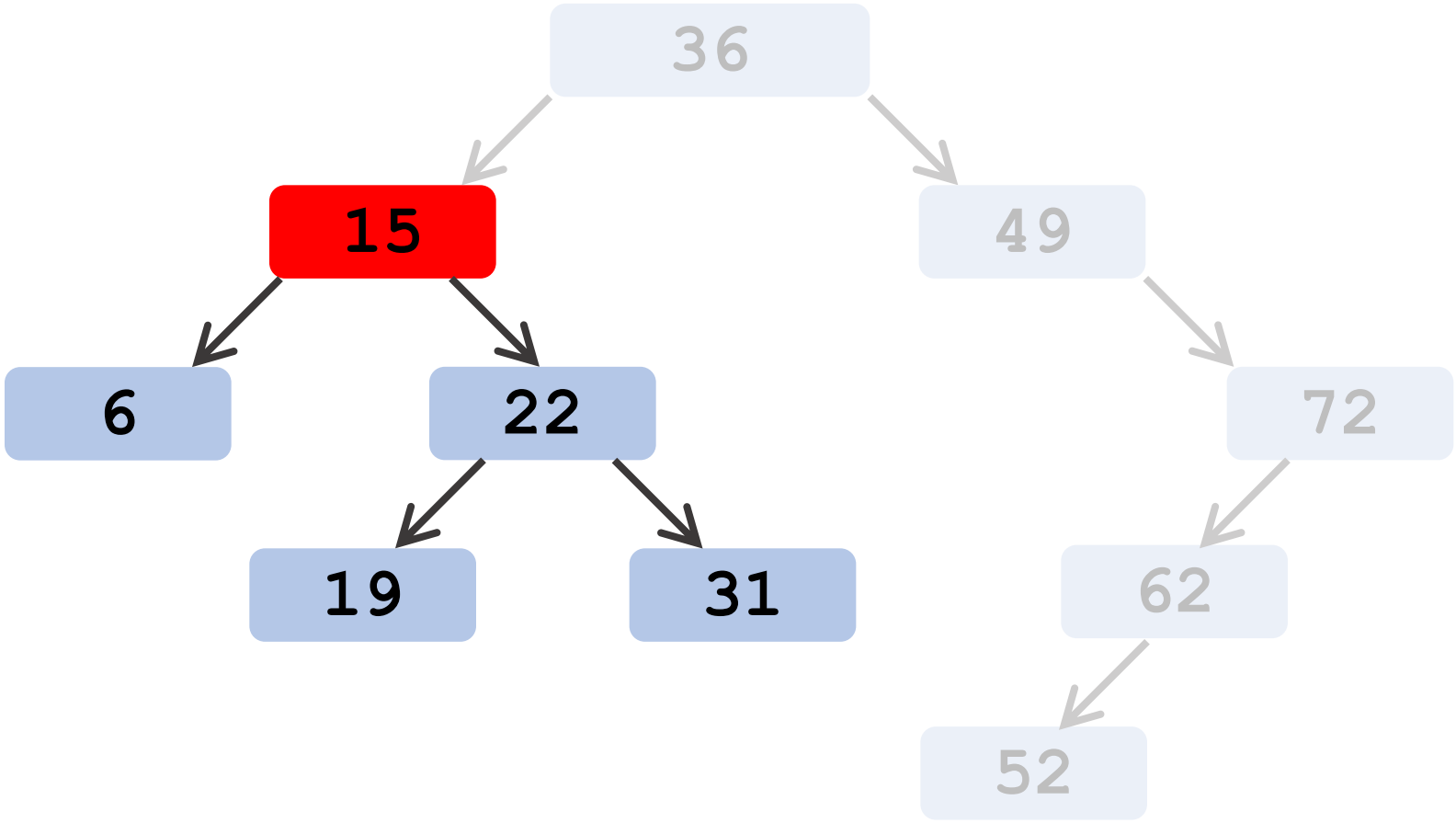


What is the successor of 15?



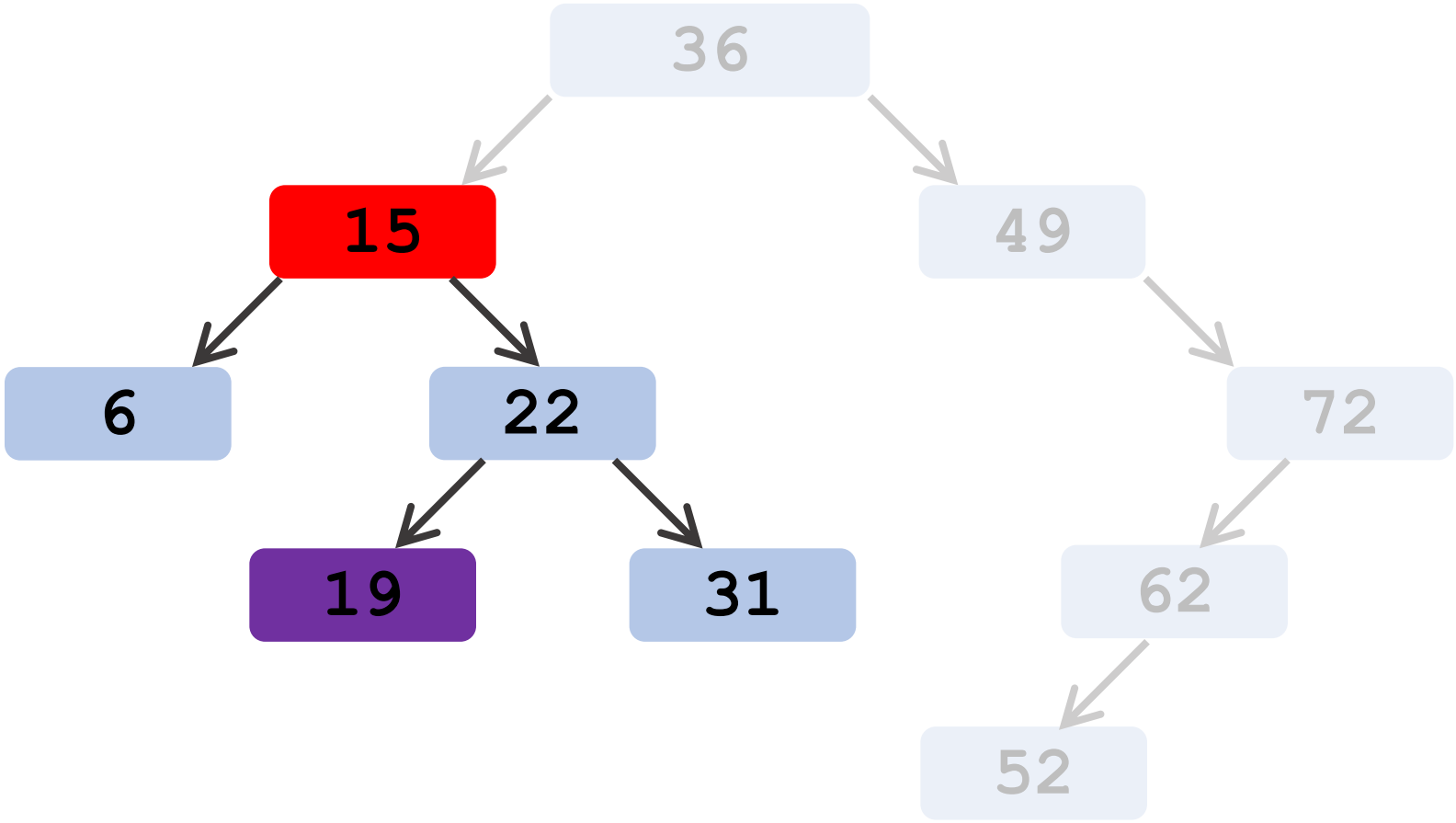
What is the successor of 15?

Inorder traversal: 6 | 15 | 19 | 22 | 31 | 36 | 49 | 52 | 62 | 72



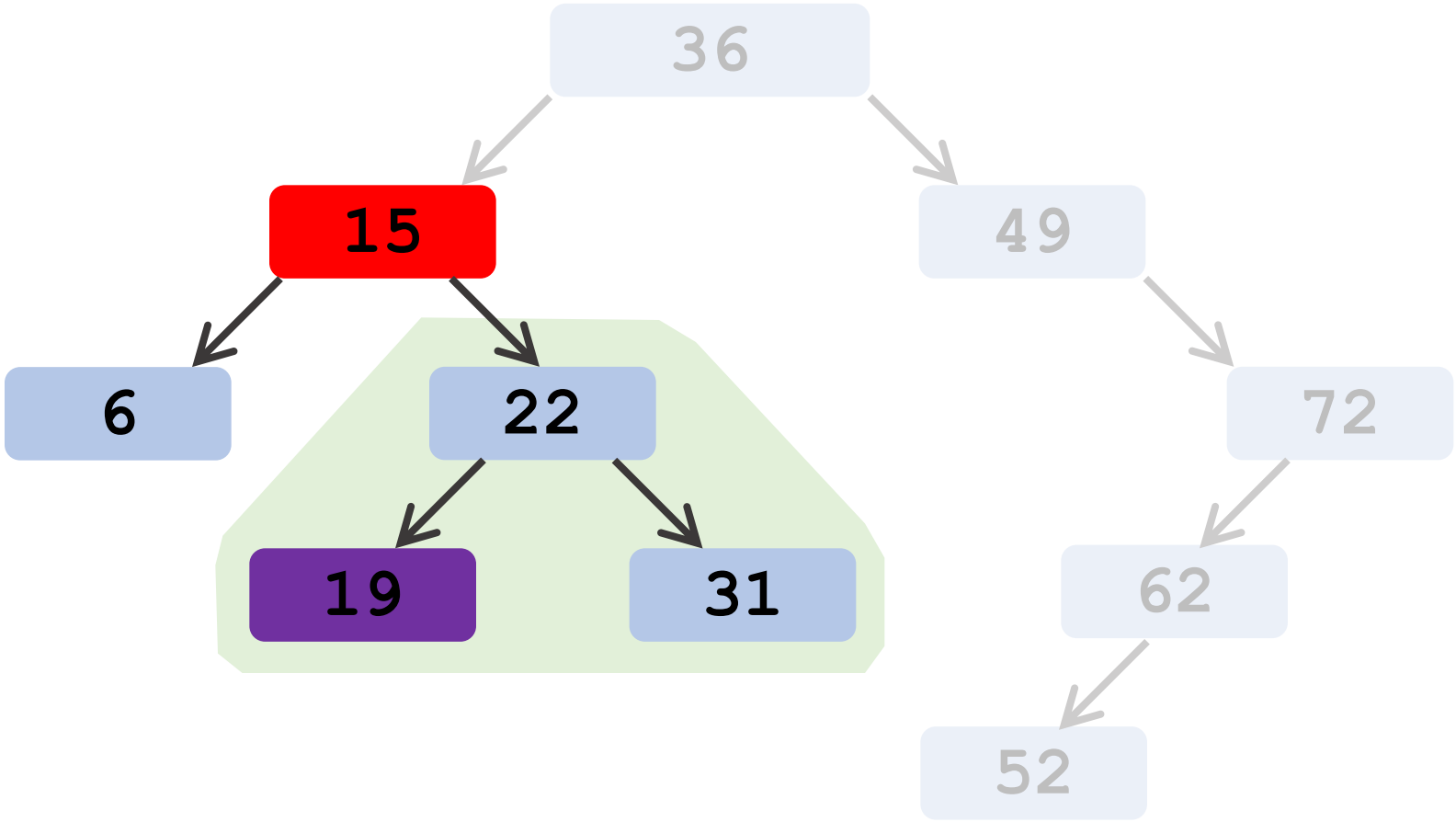
What is the successor of 15?

Inorder traversal: 6 | 15 | 19 | 22 | 31 | 36 | 49 | 52 | 62 | 72

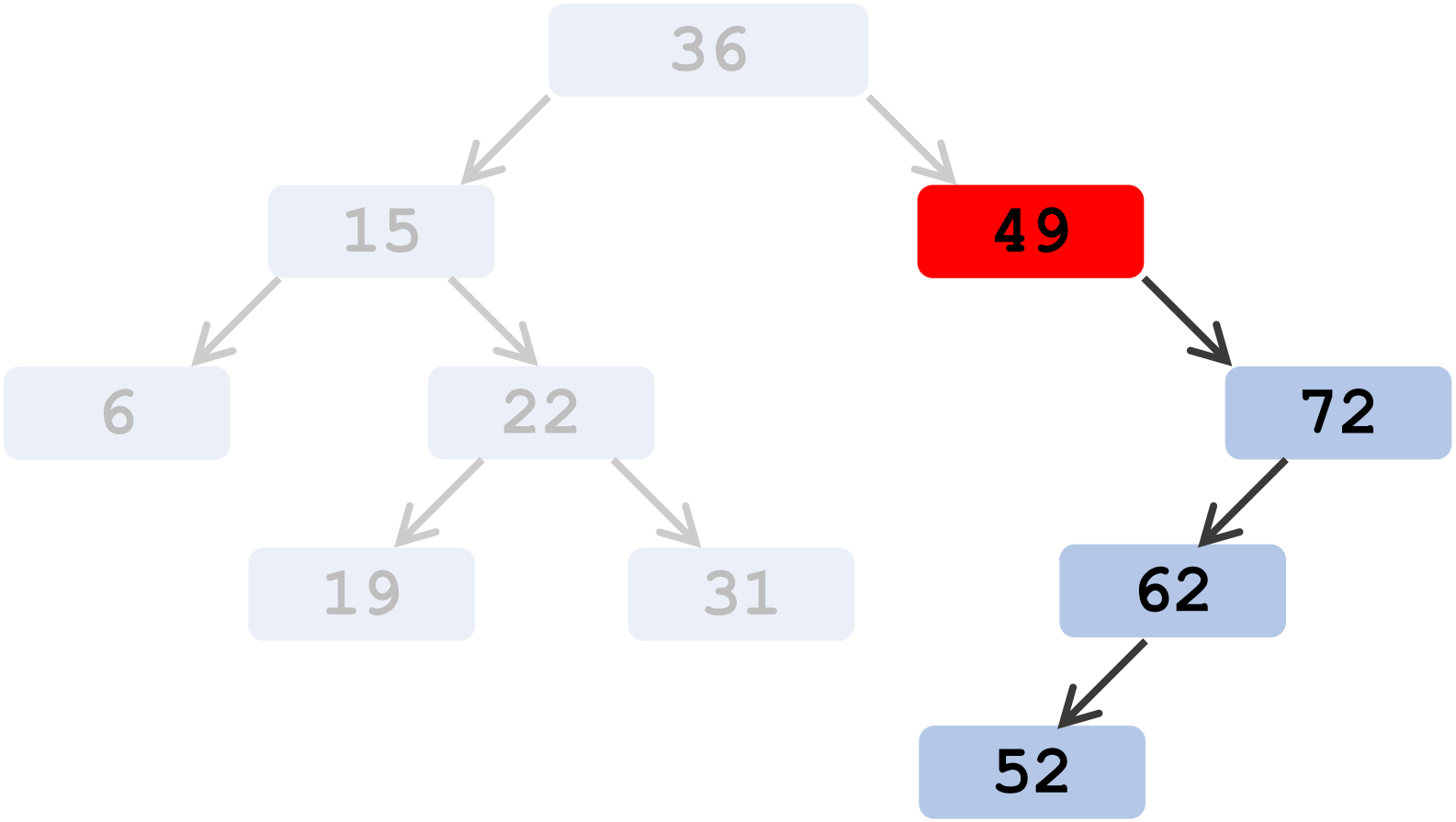


What is the successor of 15?

Successor is the leftmost vertex of the right sub-tree.

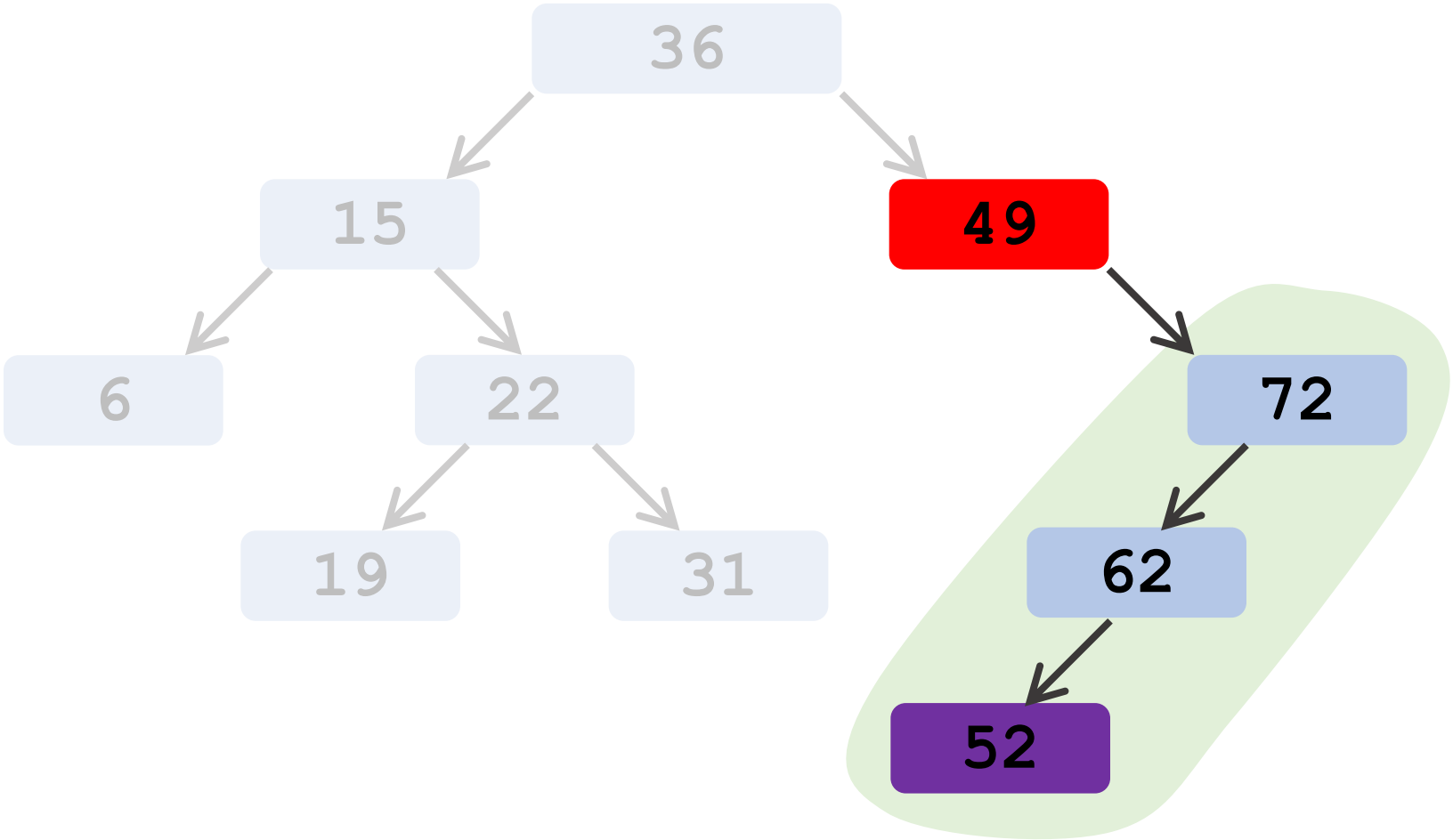


What is the successor of 49?

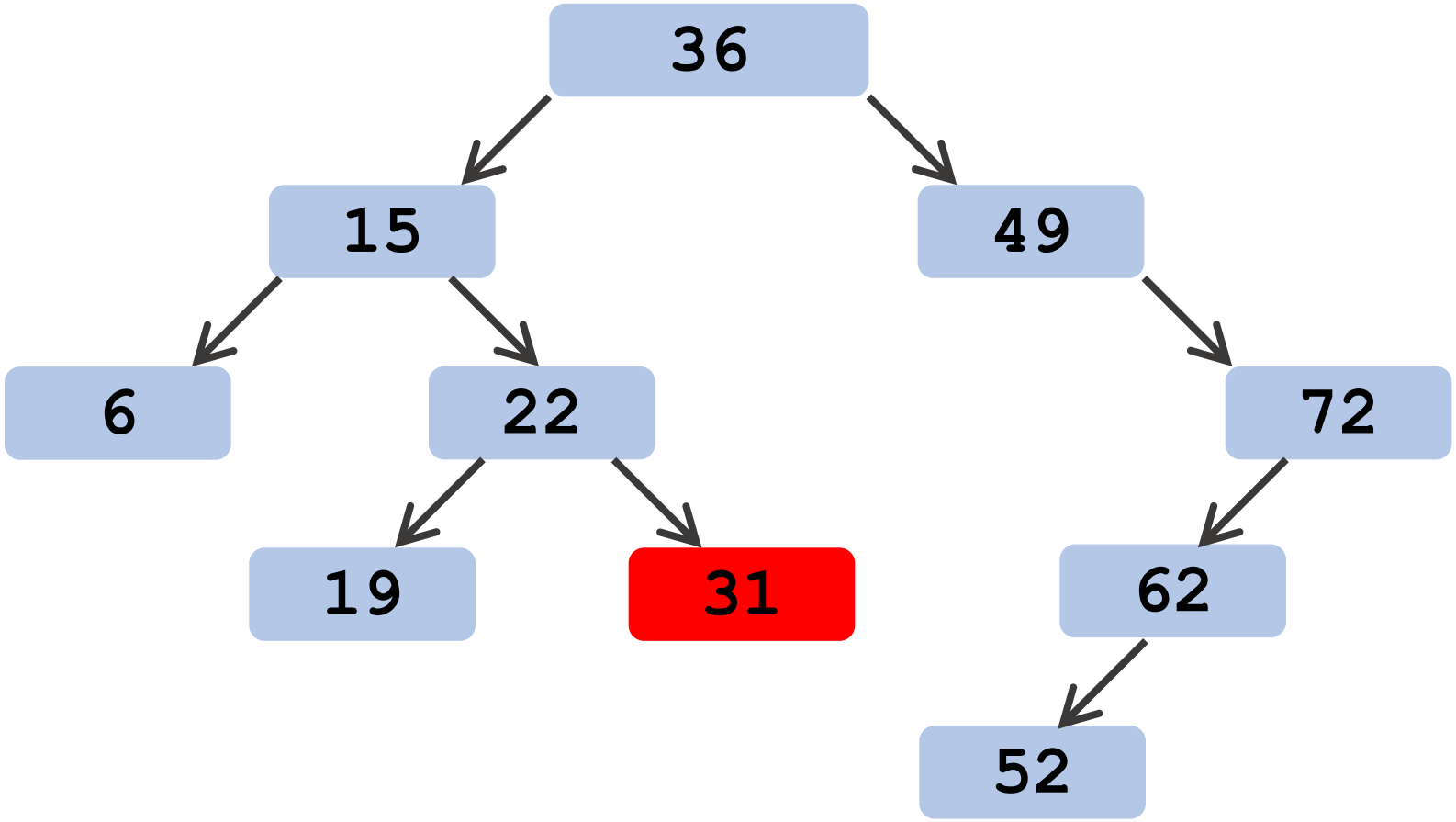


What is the successor of 49?

Successor is the leftmost vertex of the right sub-tree.

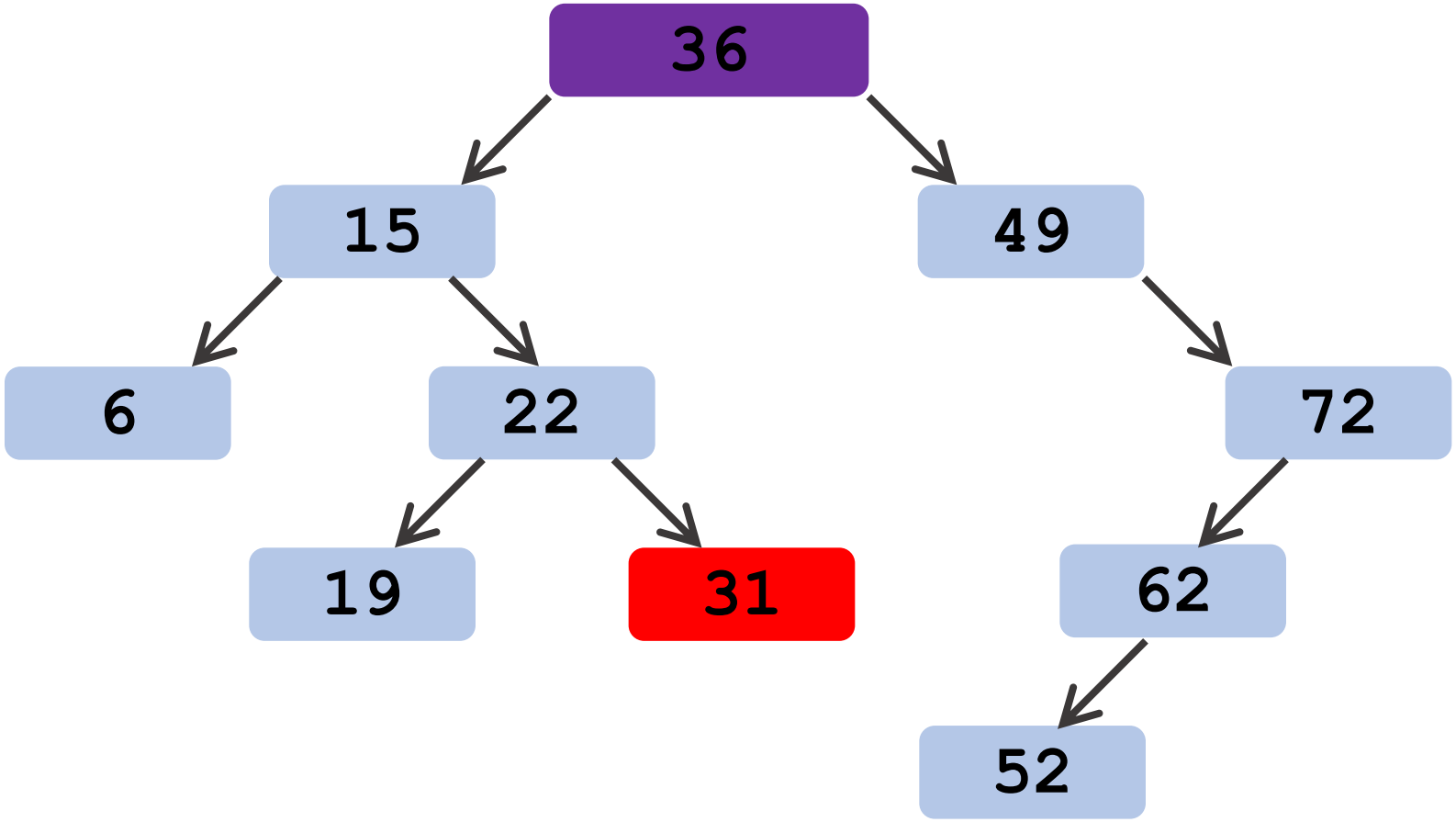


What is the successor of 31?



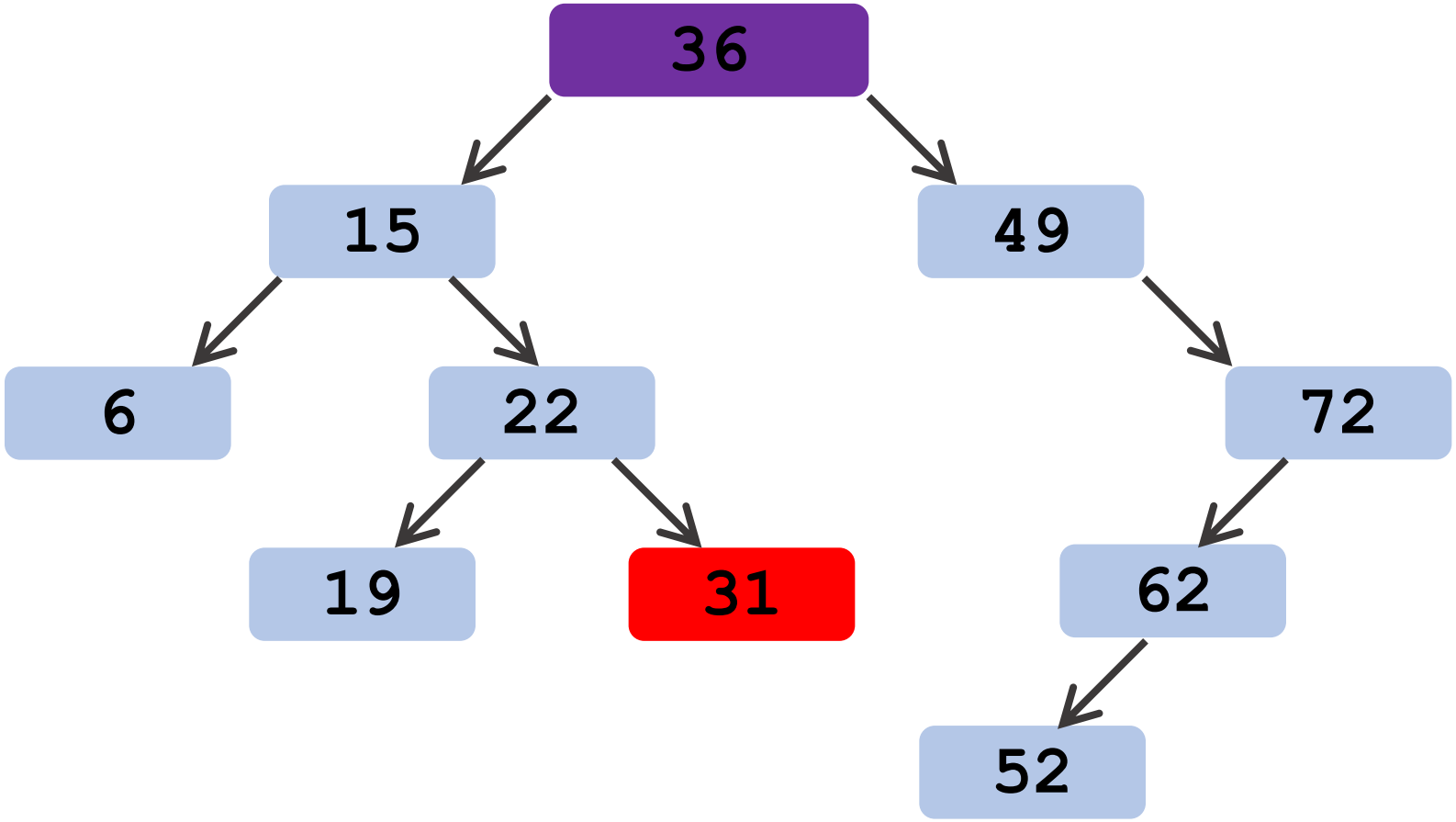
What is the successor of 31?

Inorder traversal: 6 | 15 | 19 | 22 | 31 | 36 | 49 | 52 | 62 | 72



What is the successor of 31?

Let’s ignore such cases. Study vertices with right sub-tree.

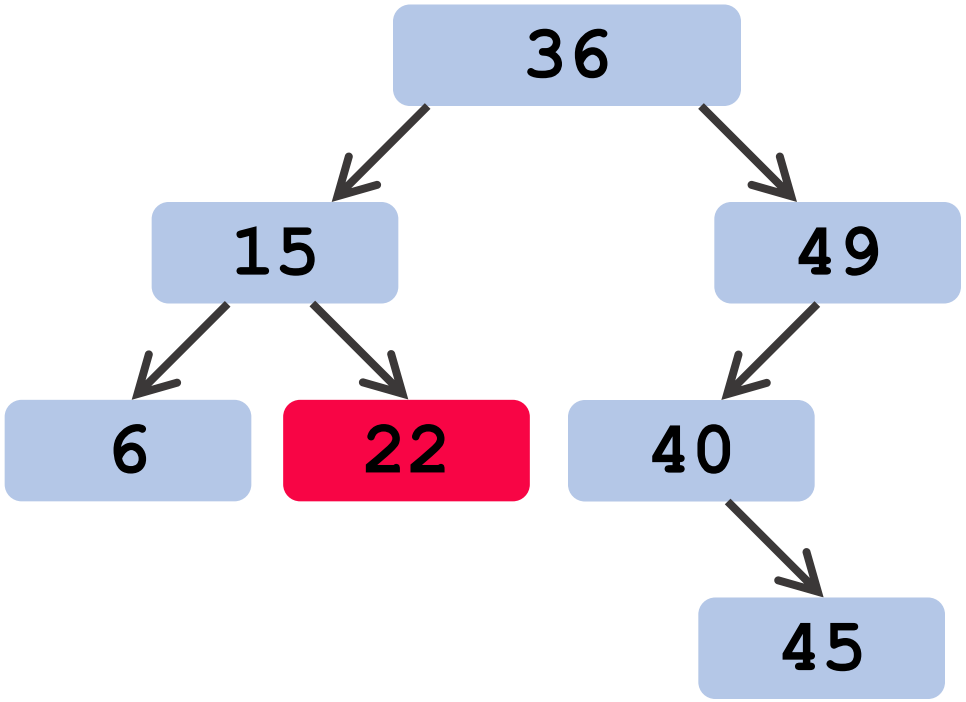


```
// find the leftmost vertex of a tree
struct vertex* leftmost(struct vertex* root) {
      struct vertex* current = root;
      while (current->left != NULL) {
            current = current->left;
      }
      return current;
}
```

```
// find the leftmost vertex of a tree
struct vertex* leftmost(struct vertex* root) {
    struct vertex* current = root;
    while (current->left != NULL) {
        current = current->left;
    }
    return current;
}
// assume vertex v has right child
// the successor of v
struct vertex* successor = leftmost(v->right);
```

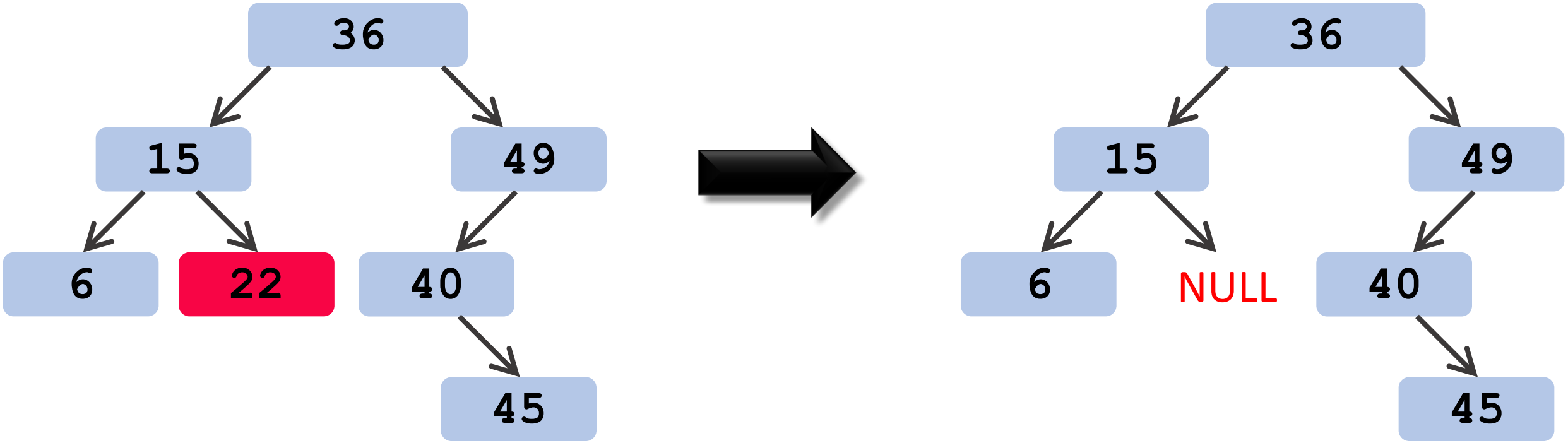
Delete Vertex

Case 1: Vertex to be deleted is leaf

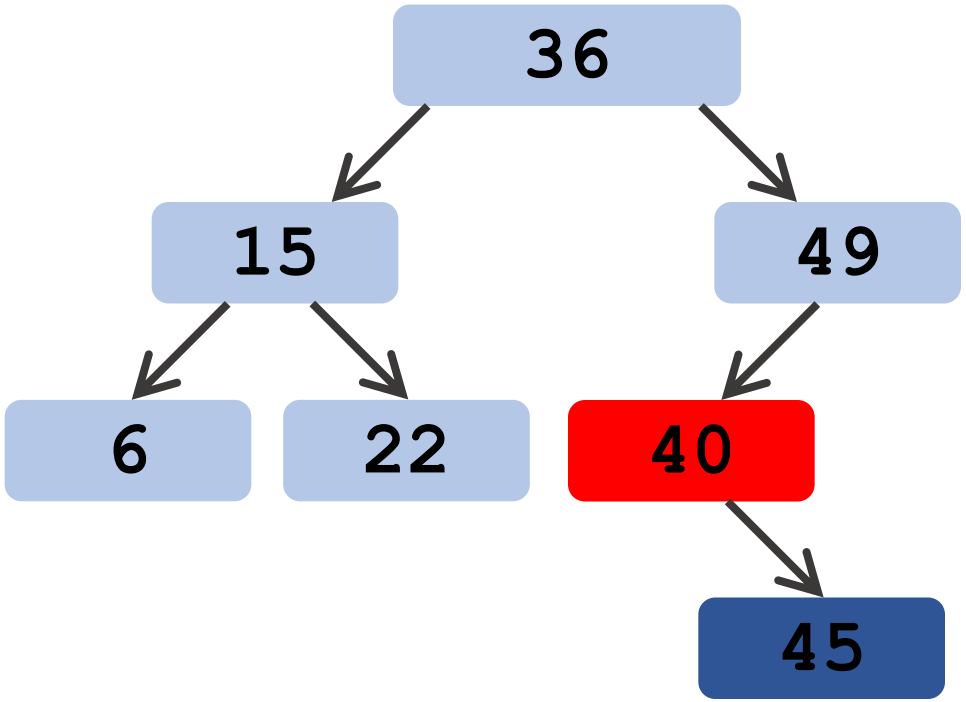


Case 1: Vertex to be deleted is leaf

Simply delete the vertex and set its parent's pointer to NULL.

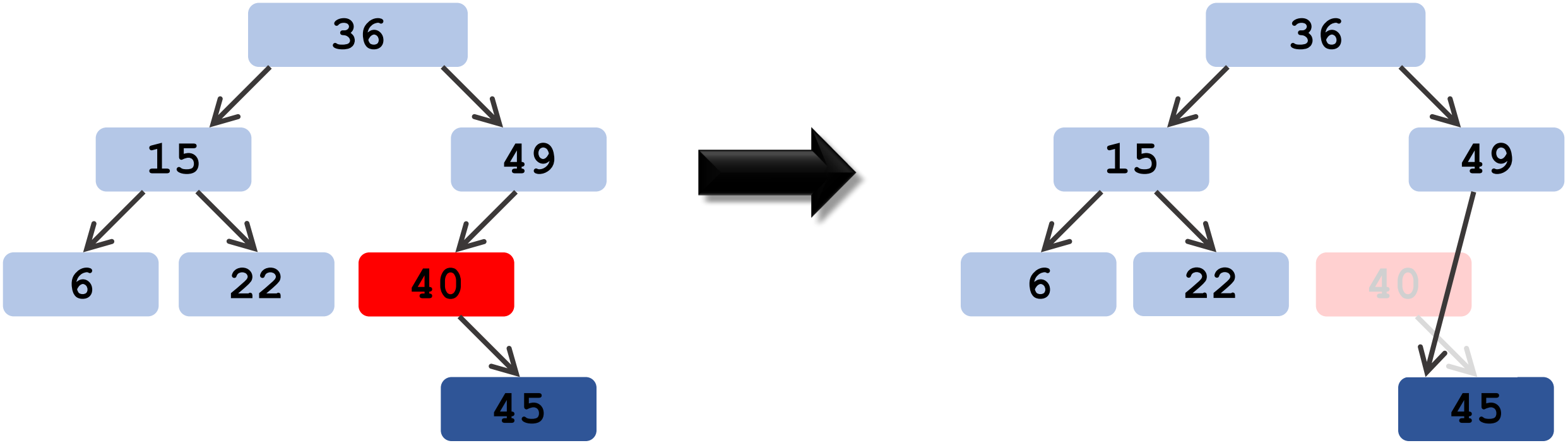


Case 2: Vertex to be deleted has one child



Case 2: Vertex to be deleted has one child

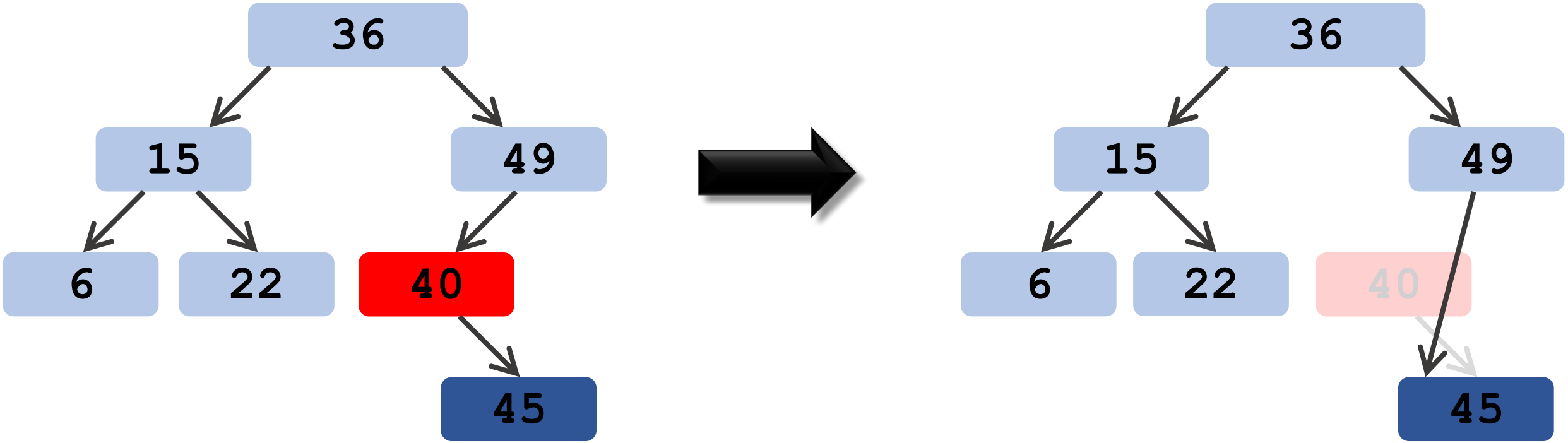
1. Let the vertex's parent point to the vertex's child.



Case 2: Vertex to be deleted has one child

1. Let the vertex's parent point to the vertex's child.

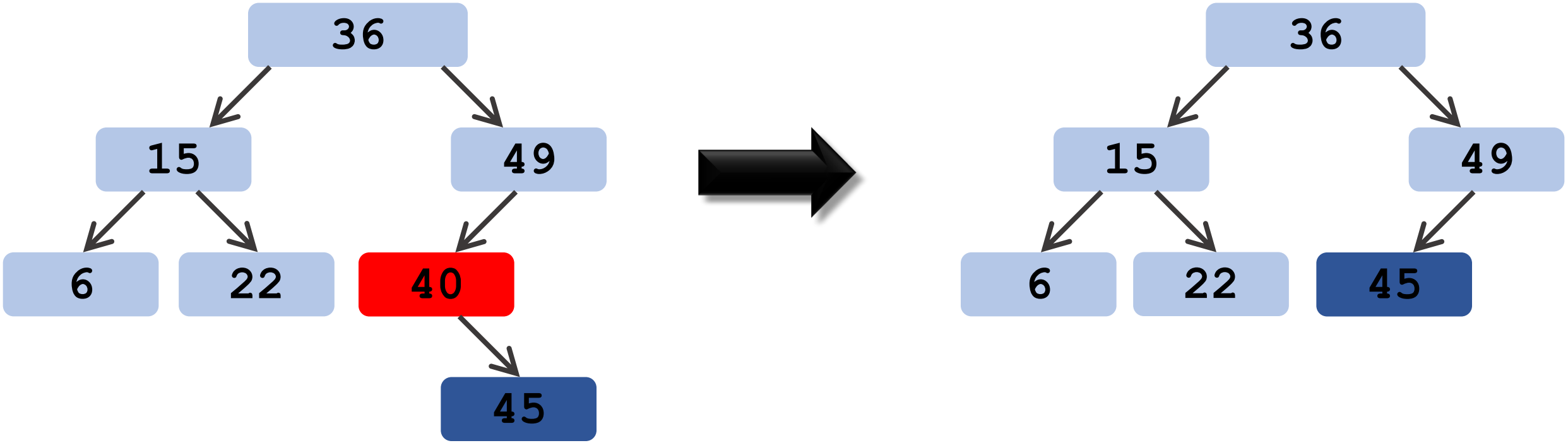
2. Free the vertex from memory.



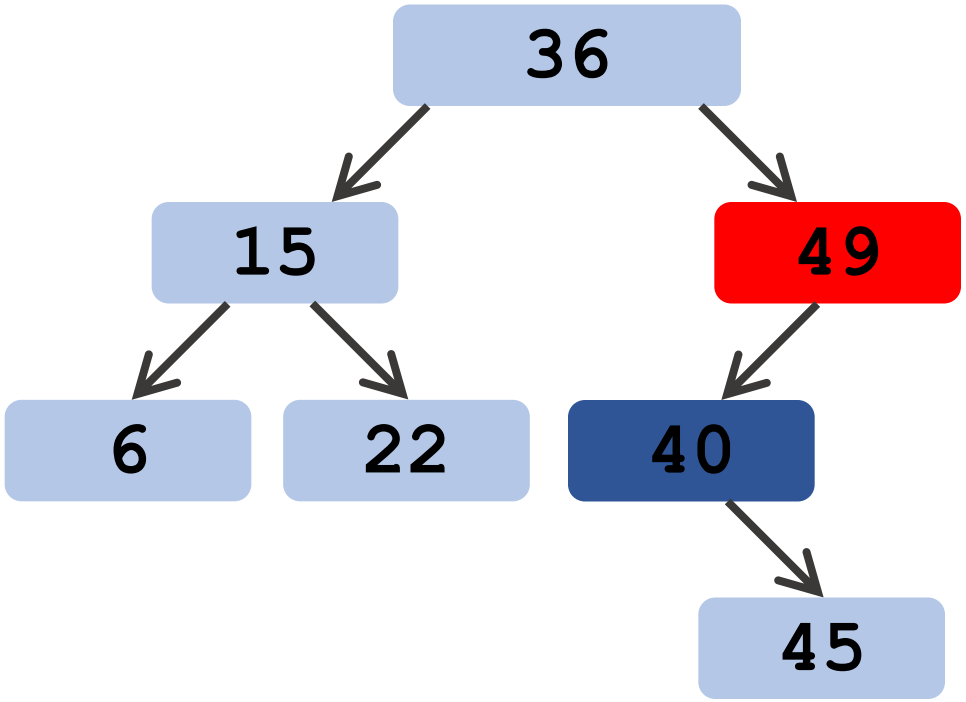
Case 2: Vertex to be deleted has one child

1. Let the vertex's parent point to the vertex's child.

2. Free the vertex from memory.

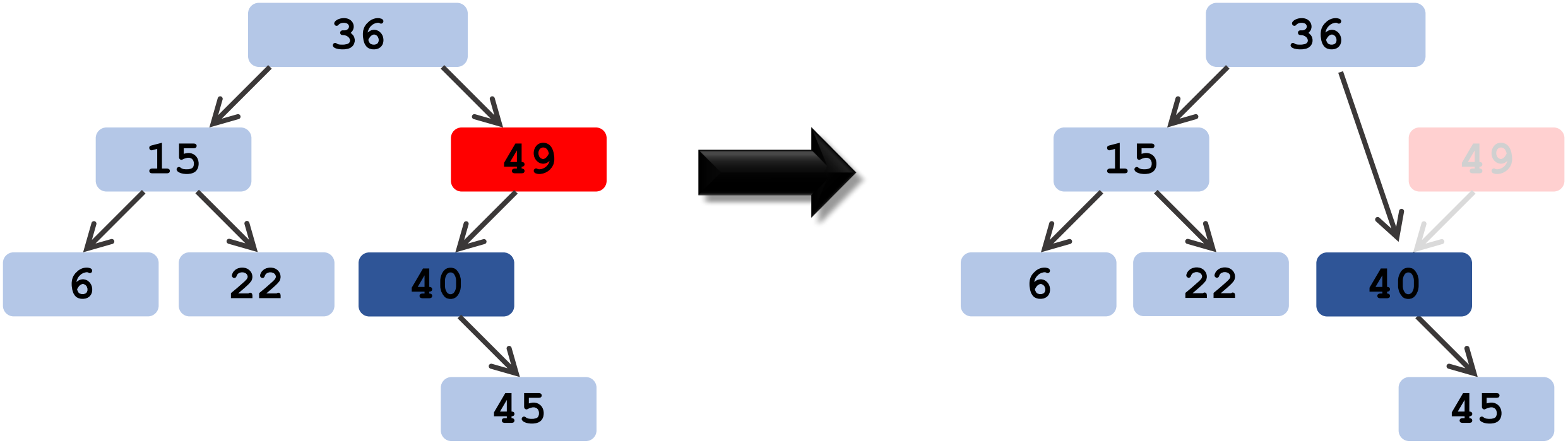


Case 2: Vertex to be deleted has one child



Case 2: Vertex to be deleted has one child

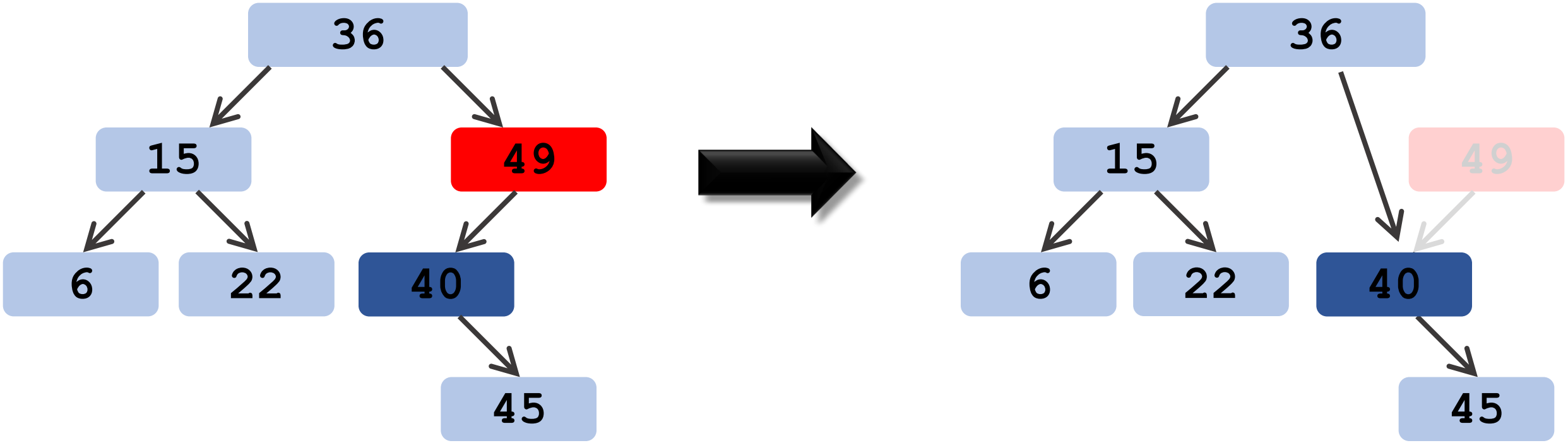
1. Let the vertex's parent point to the vertex's child.



Case 2: Vertex to be deleted has one child

1. Let the vertex's parent point to the vertex's child.

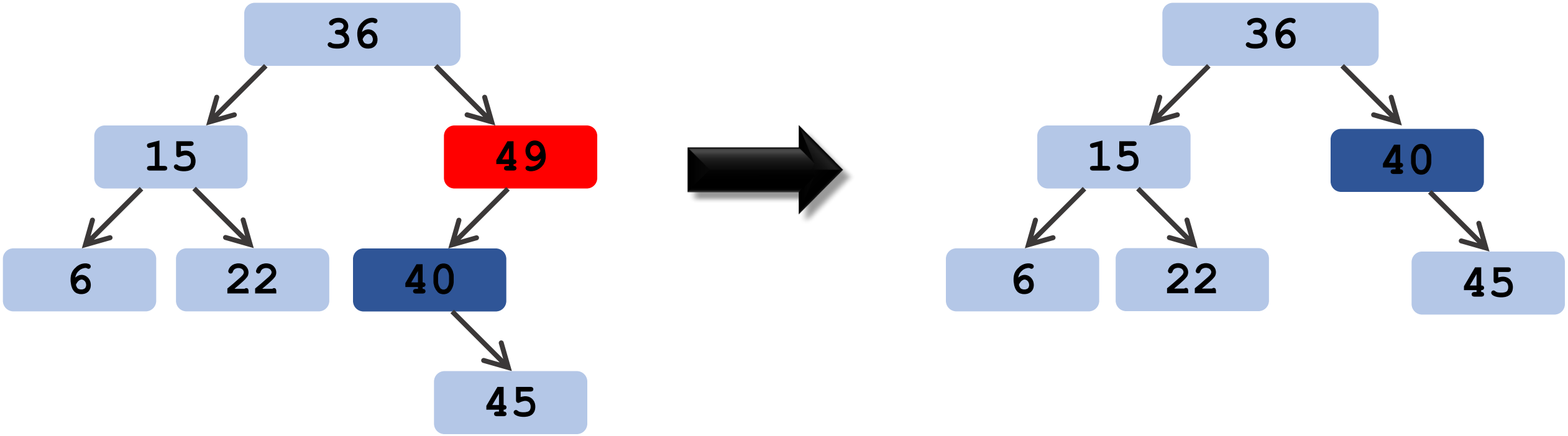
2. Free the vertex from memory.



Case 2: Vertex to be deleted has one child

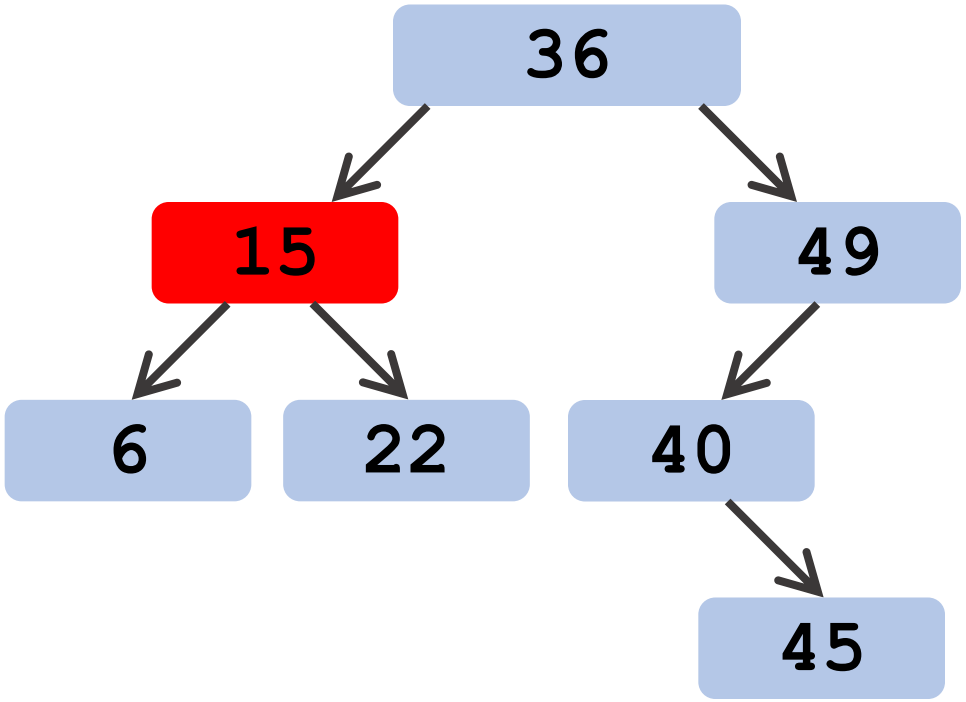
1. Let the vertex's parent point to the vertex's child.

2. Free the vertex from memory.



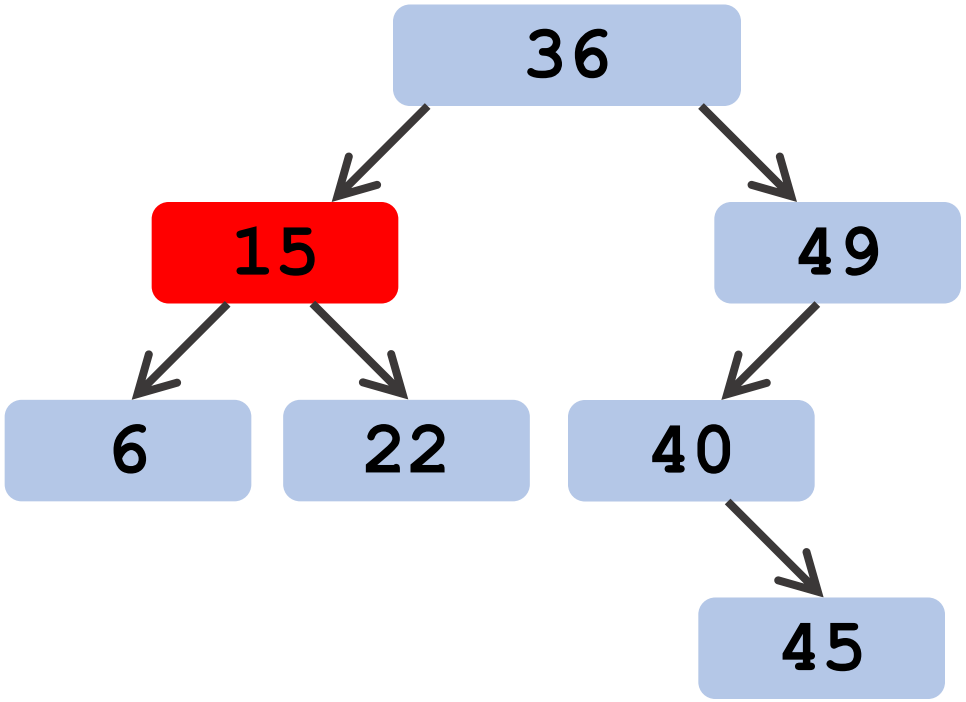
Case 3: Vertex to be deleted has two children

Basic idea: Replace the vertex to be deleted by its successor.



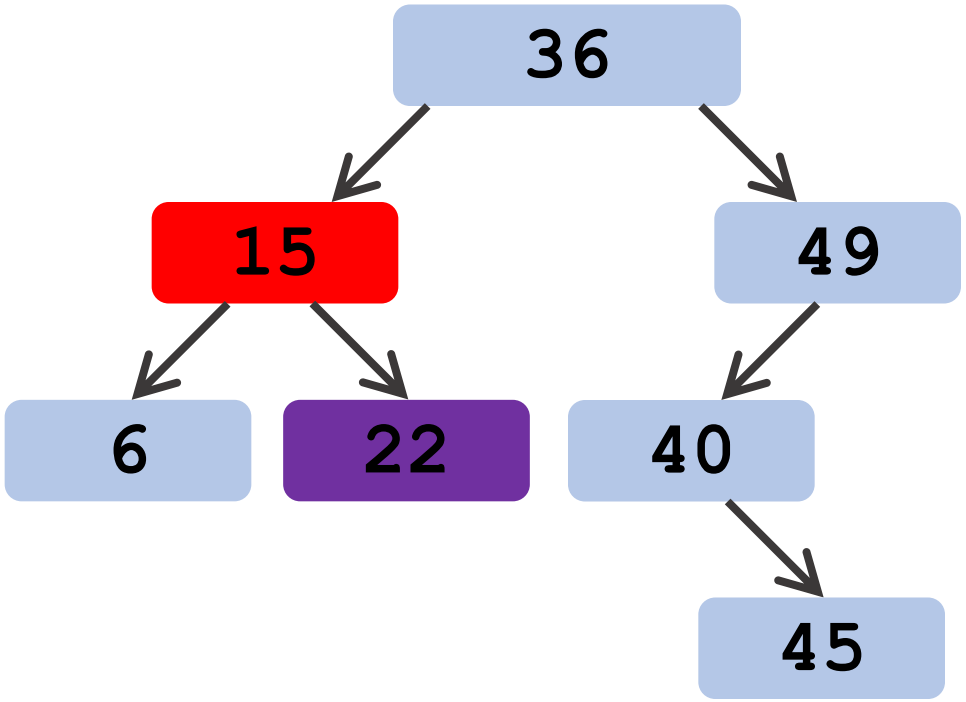
Case 3: Vertex to be deleted has two children

1. Find the successor of the vertex to be deleted.



Case 3: Vertex to be deleted has two children

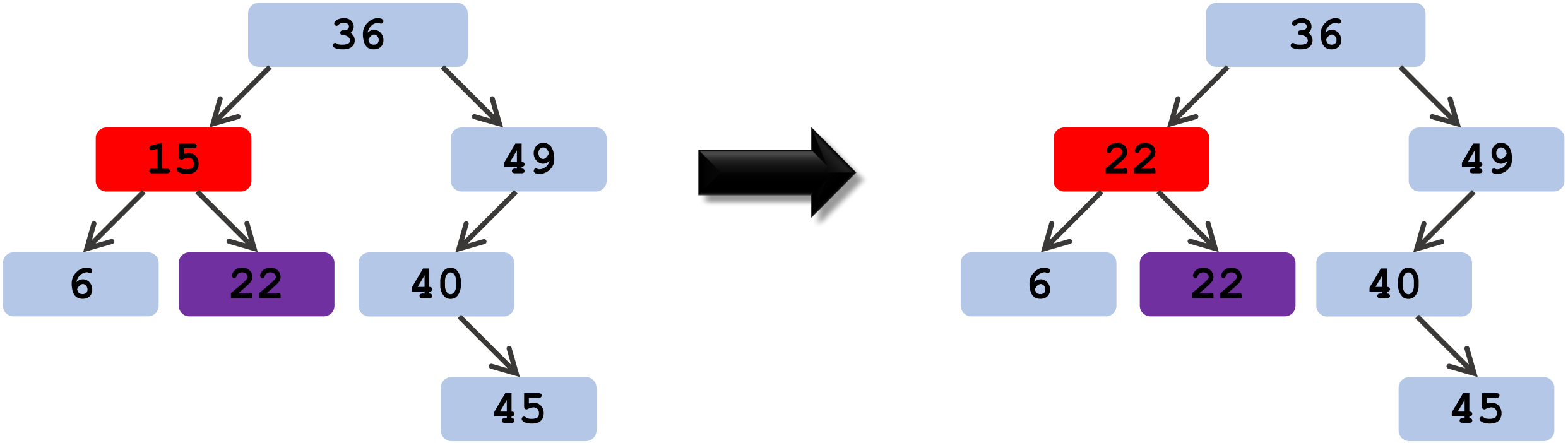
1. Find the successor of the vertex to be deleted.



Case 3: Vertex to be deleted has two children

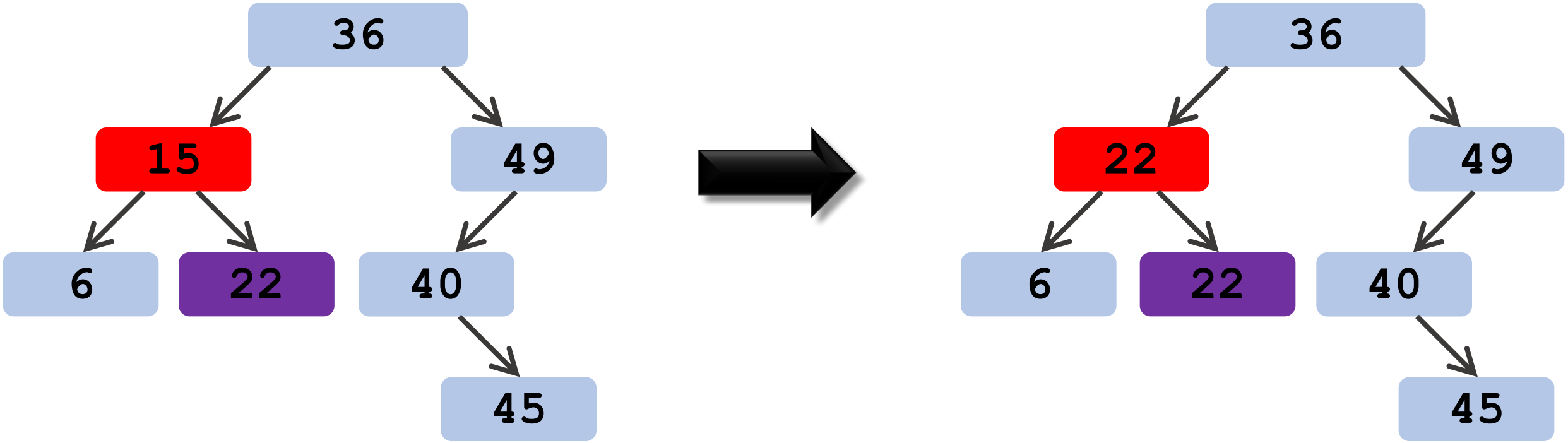
1. Find the successor of the vertex to be deleted.

2. Copy contents of the successor to the vertex.



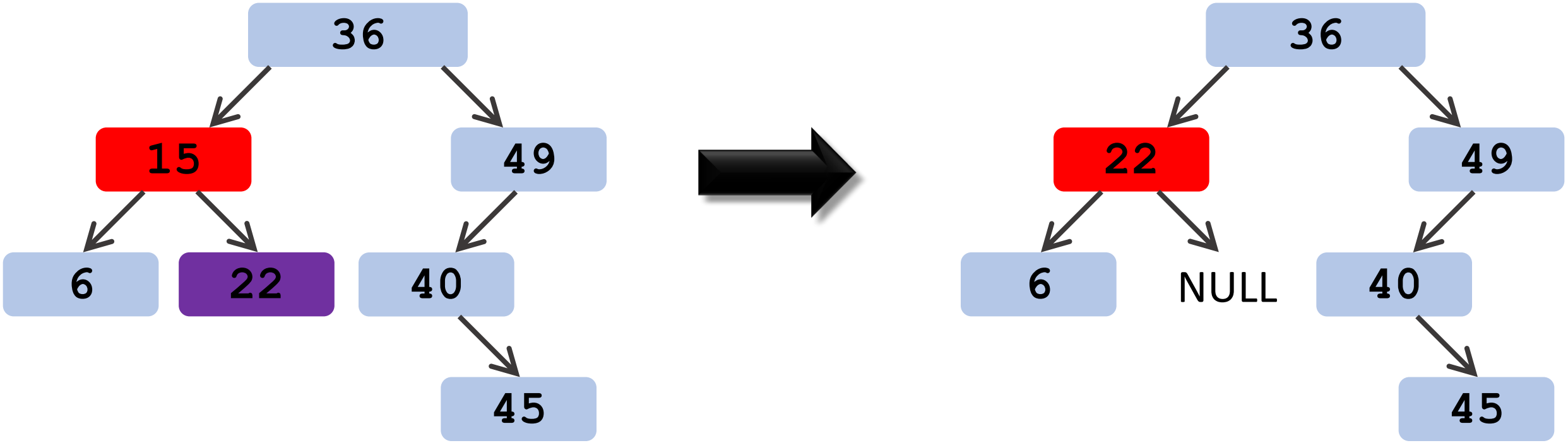
Case 3: Vertex to be deleted has two children

1. Find the successor of the vertex to be deleted.
2. Copy contents of the successor to the vertex.
3. Delete the successor.

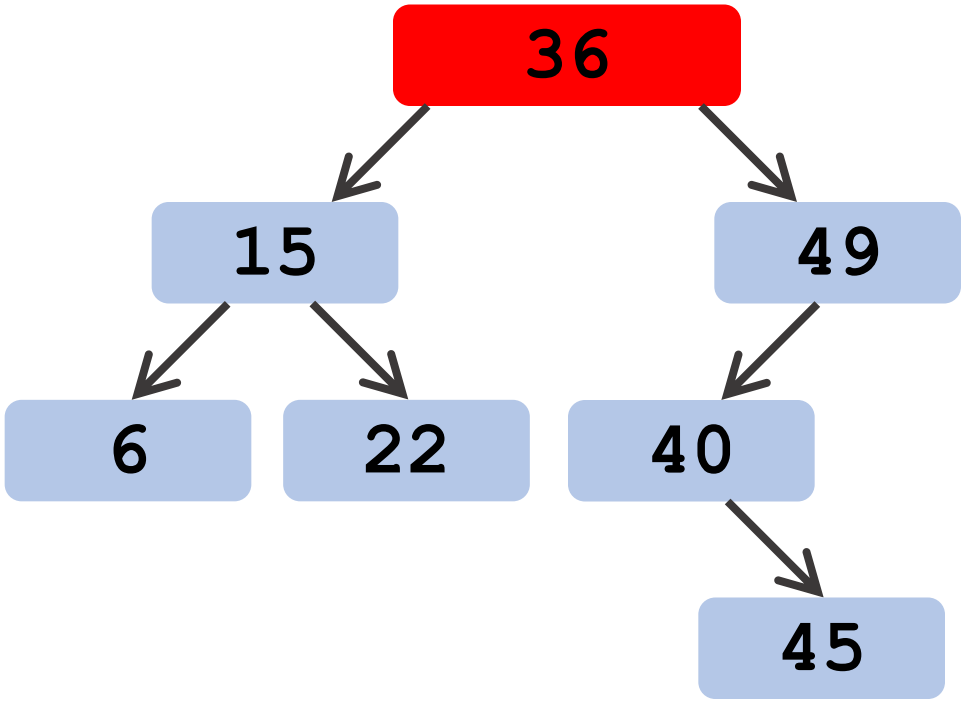


Case 3: Vertex to be deleted has two children

1. Find the successor of the vertex to be deleted.
2. Copy contents of the successor to the vertex.
3. Delete the successor.

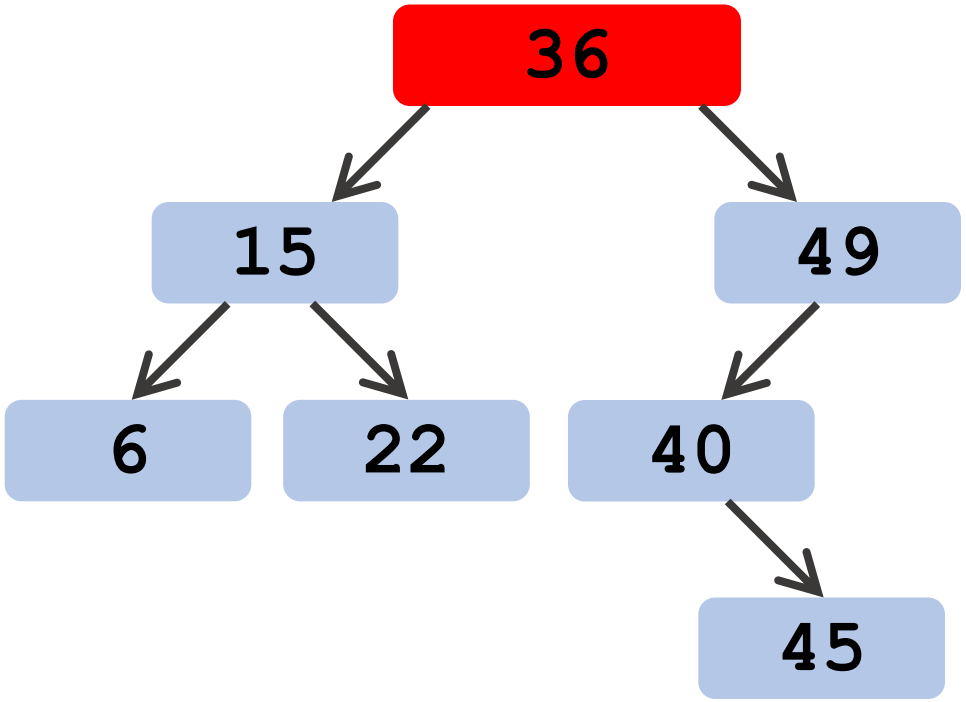


Case 3: Vertex to be deleted has two children



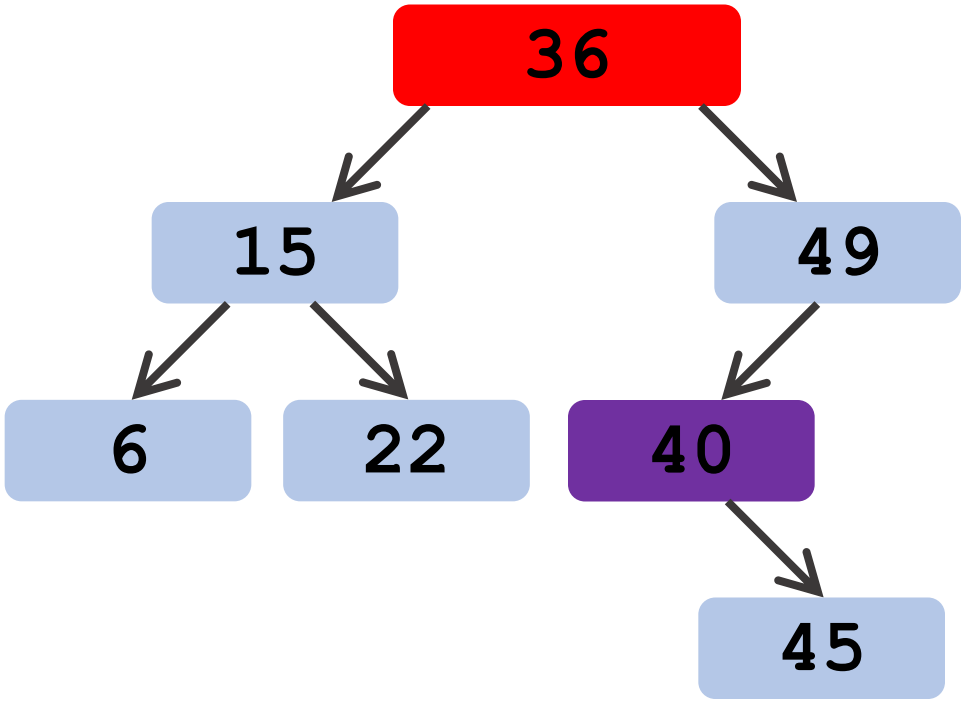
Case 3: Vertex to be deleted has two children

1. Find the successor of the vertex to be deleted.



Case 3: Vertex to be deleted has two children

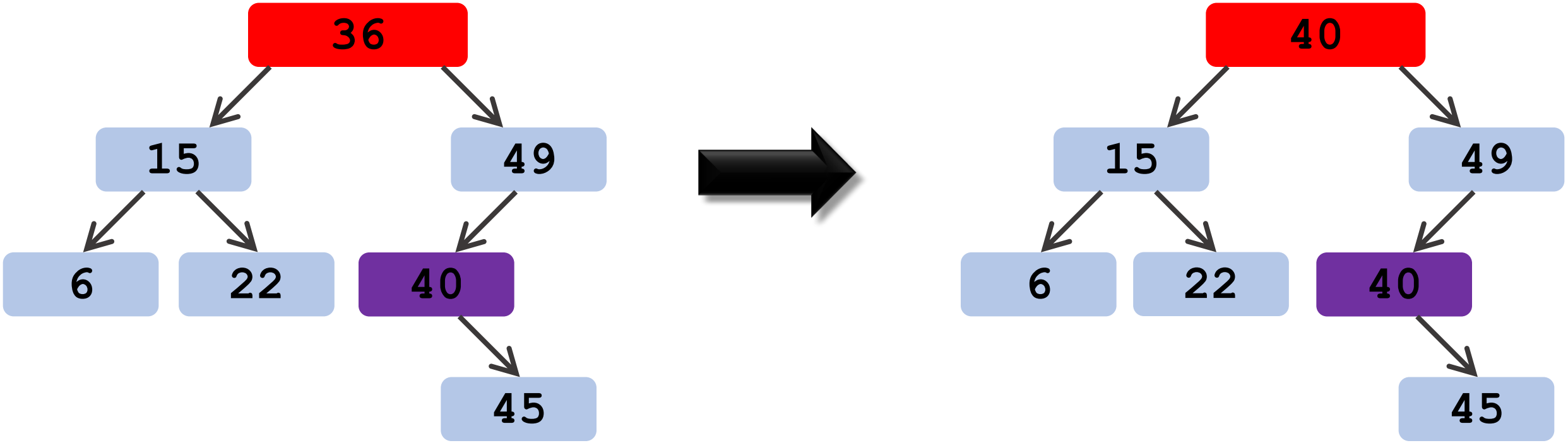
1. Find the successor of the vertex to be deleted.



Case 3: Vertex to be deleted has two children

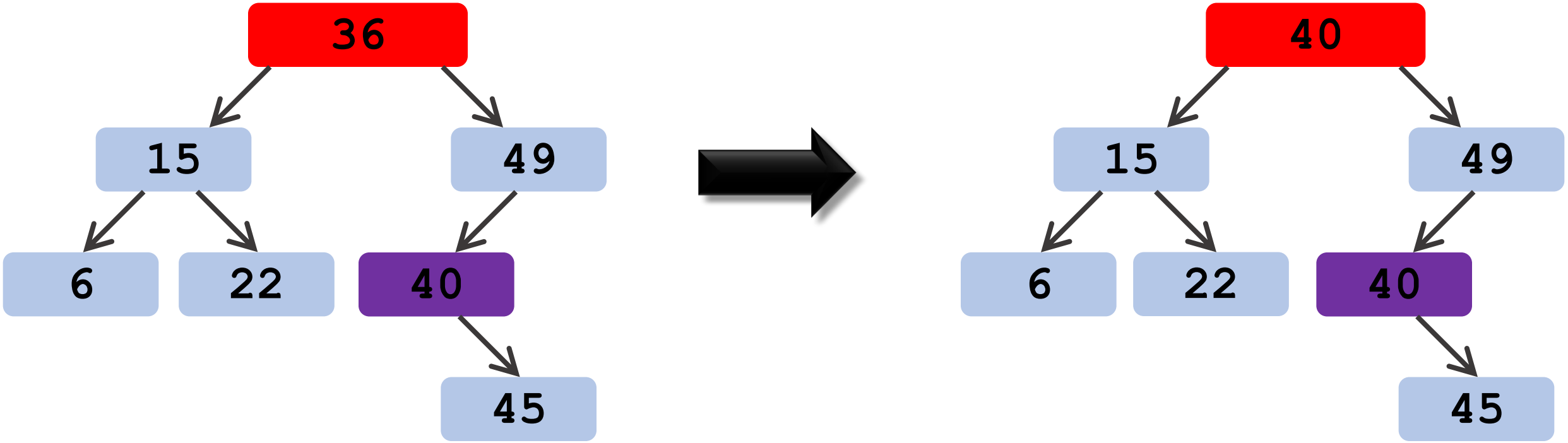
1. Find the successor of the vertex to be deleted.

2. Copy contents of the successor to the vertex.



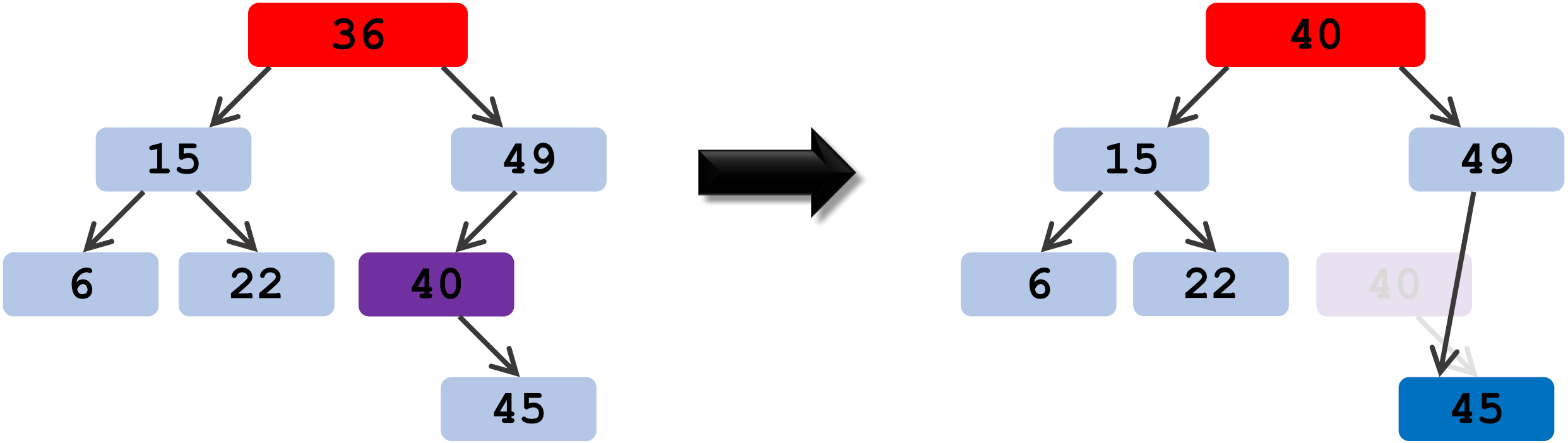
Case 3: Vertex to be deleted has two children

1. Find the successor of the vertex to be deleted.
2. Copy contents of the successor to the vertex.
3. Recursively delete the successor.



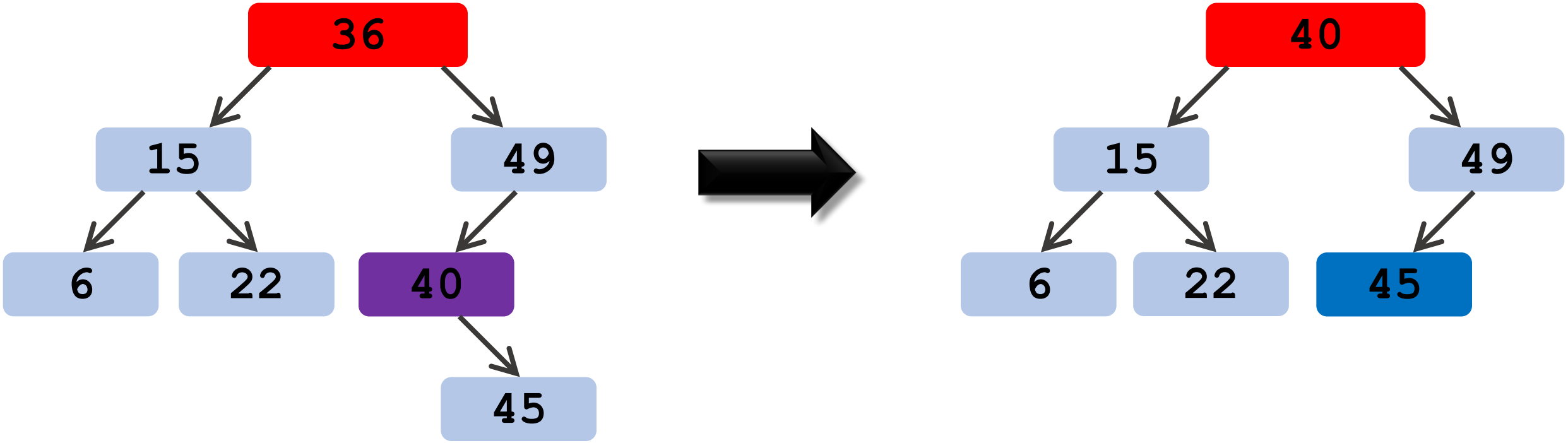
Case 3: Vertex to be deleted has two children

1. Find the successor of the vertex to be deleted.
2. Copy contents of the successor to the vertex.
3. Recursively delete the successor.



Case 3: Vertex to be deleted has two children

1. Find the successor of the vertex to be deleted.
2. Copy contents of the successor to the vertex.
3. Recursively delete the successor.



```
struct vertex* del(struct vertex* root, int key) {
      if (root == NULL) return NULL;
      // in the left sub-tree
      if (key < root->key) {
            root->left = del(root->left, key);
            return root;
      }
      // in the right sub-tree
      if (key > root->key) {
            root->right = del(root->right, key);
            return root;
      }
      // ... to continue
```

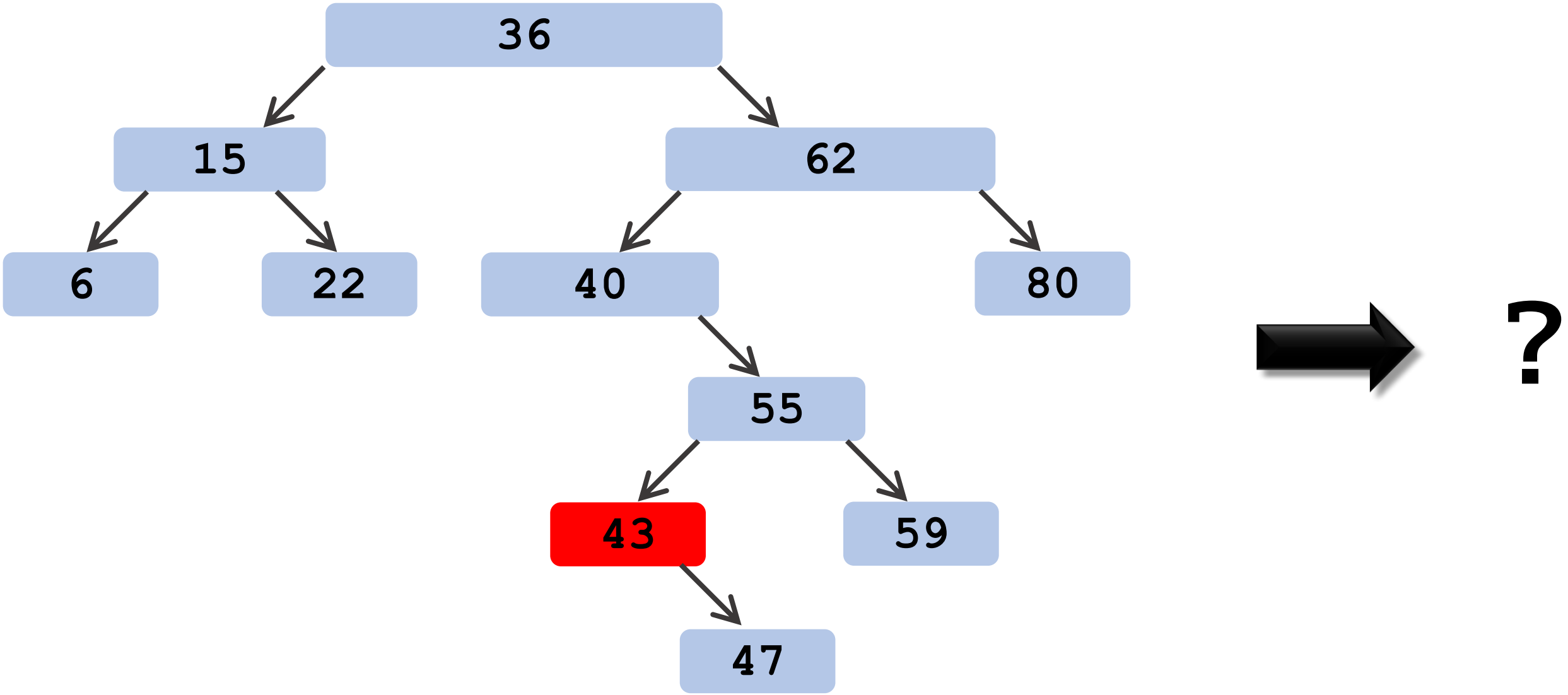
```
// the root is the vertex to be deleted

// the root has no child
if (root->left == NULL && root->right == NULL) {
      delete root;
      return NULL;
}




// ... to continue
```

```
// the root has only one child
else if (root->left == NULL) {
      struct vertex* v = root->right;
      delete root;
      return v;
}
else if (root->right == NULL) {
      struct vertex* v = root->left;
      delete root;
      return v;
}

// ... to continue
```

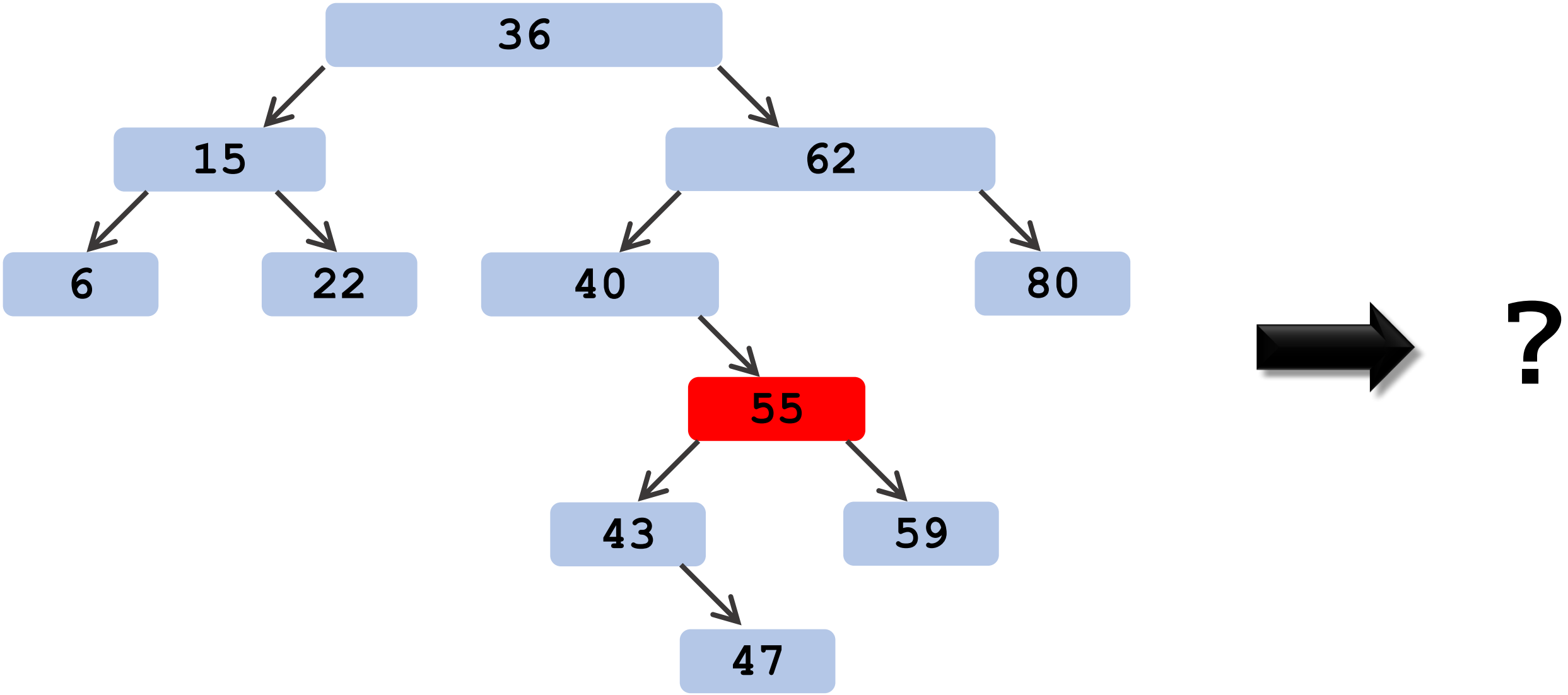
```
        // the root has two children
        else {
            // find the successor
            struct vertex* successor = leftmost(root->right);
            // copy the successor's content to this vertex
            root->key = successor->key;
            // recursively delete the successor
            root->right = del(root->right, successor->key);

            return root;
        }
}
```

Questions

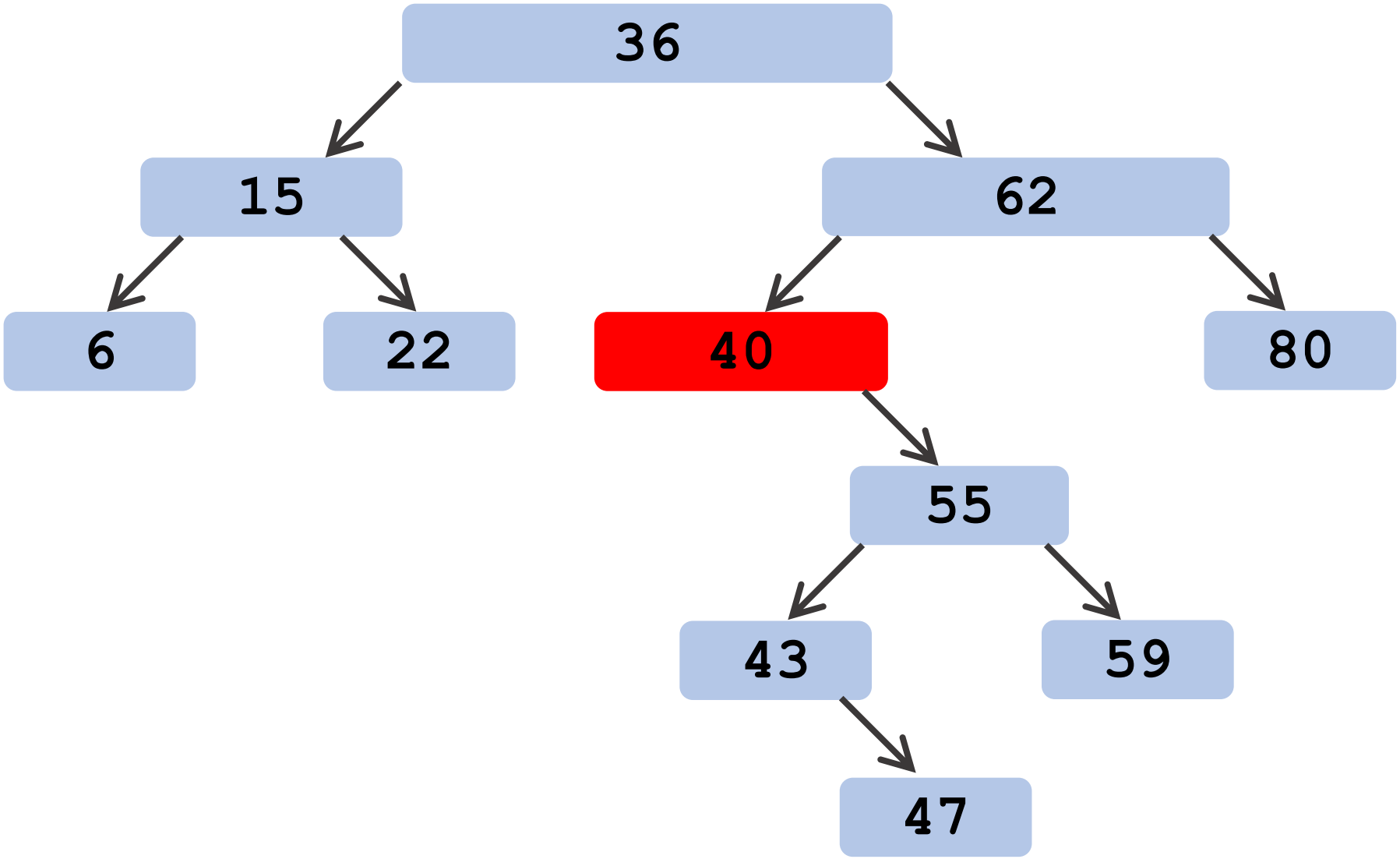
Question 1: Draw the tree after deleting 43

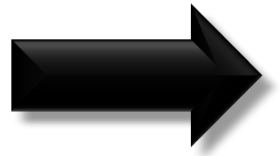


Question 2: Draw the tree after deleting 55



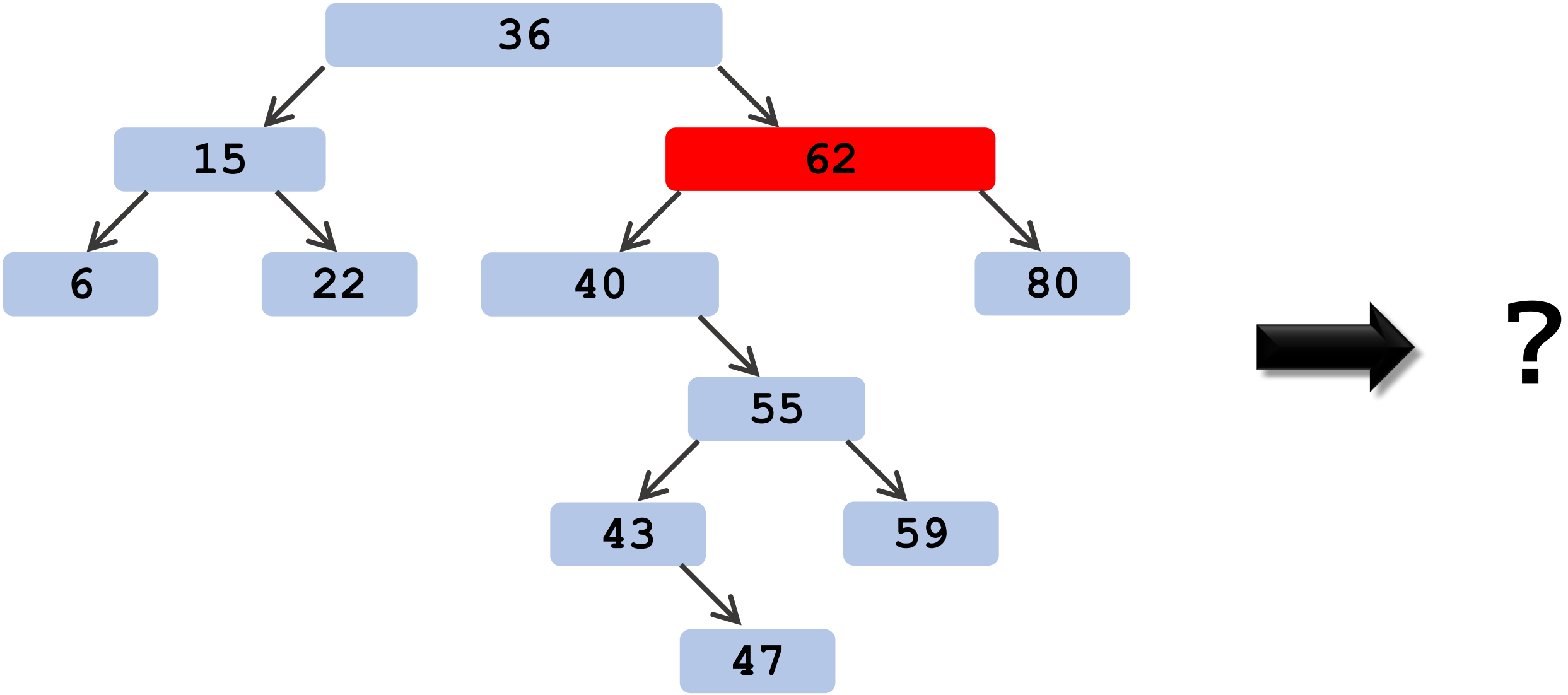
Question 3: Draw the tree after deleting 40



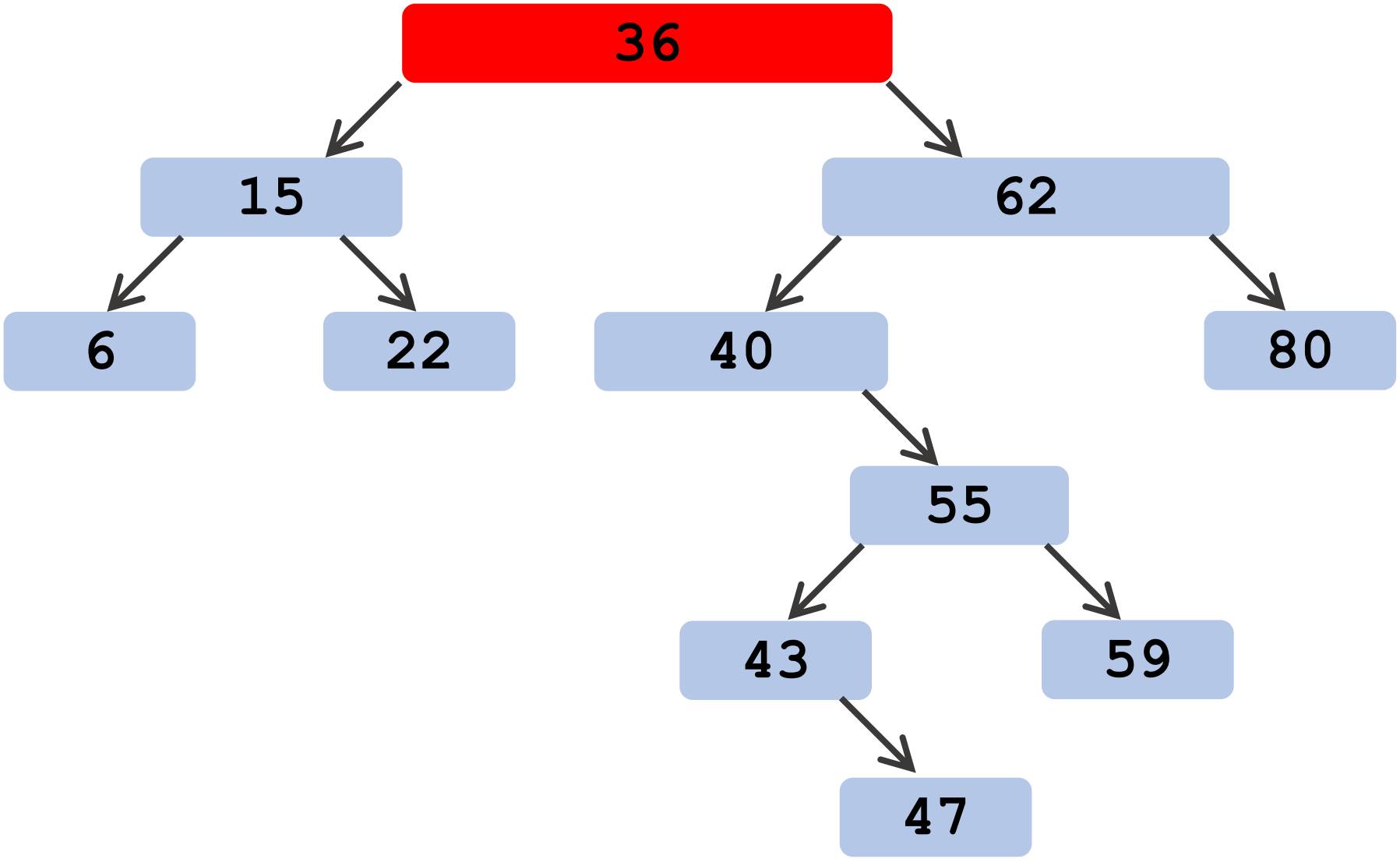


?

Question 4: Draw the tree after deleting 62



Question 5: Draw the tree after deleting 36



?

Thank You!