Binary Search Tree (3/3): Deletion

Shusen Wang

Stevens Institute of Technology

http://wangshusen.github.io/

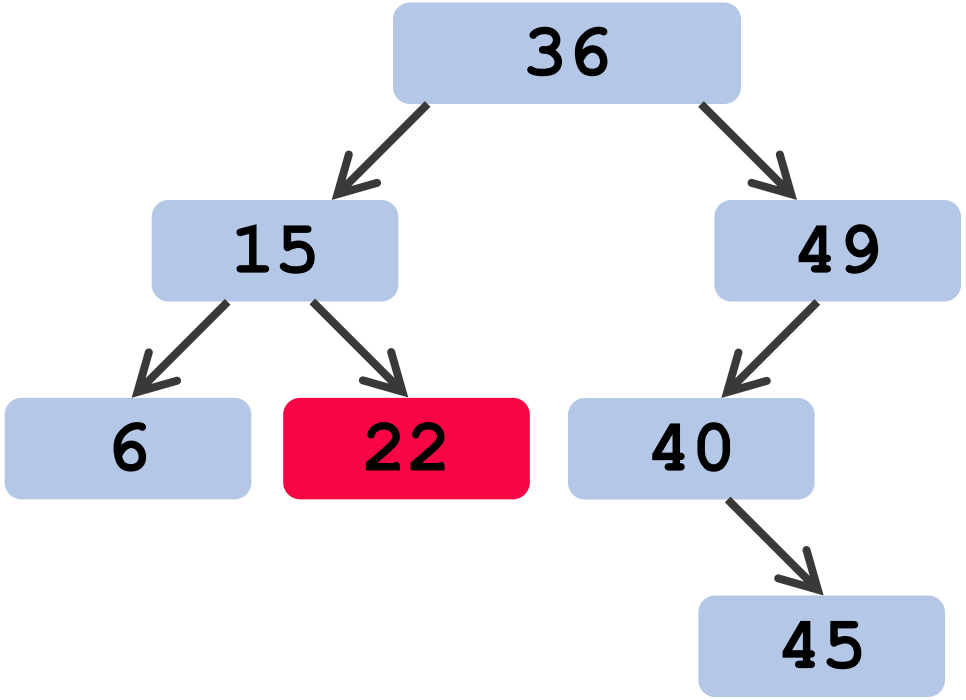
Deletion

- **Inputs:** **root** (of the tree) and **key** (to be matched).
- **Goal:** Deleting the vertex without destroying the properties of binary search tree.
- **Time complexity:** depth of the tree.

Three Cases

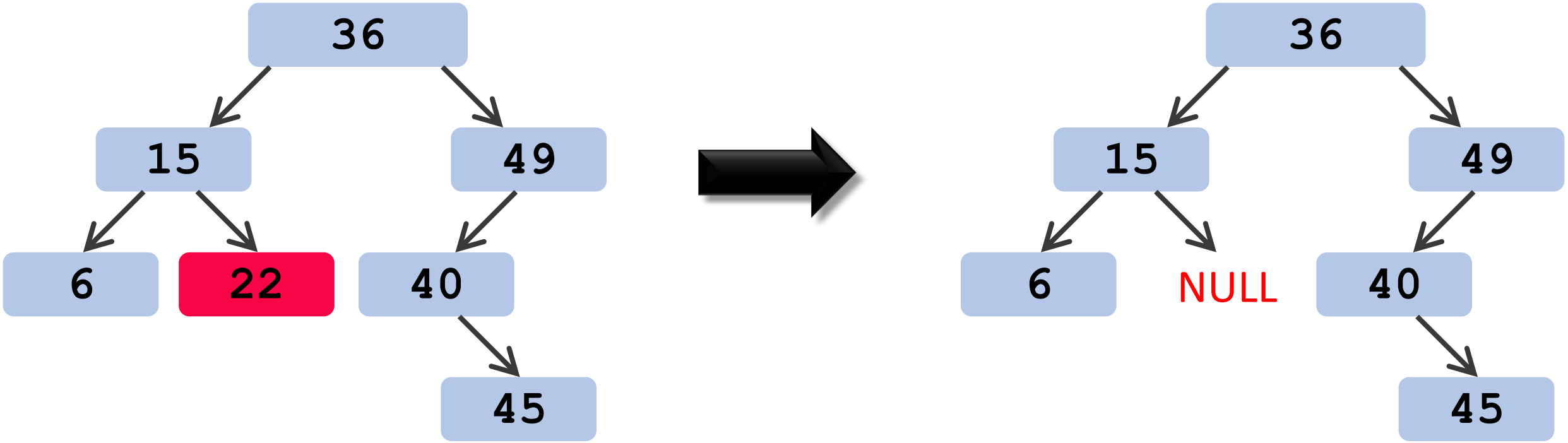
1. **Case 1:** Vertex to be deleted is a leaf (which has no child.)
2. **Case 2:** Vertex to be deleted has one child.
3. **Case 3:** Vertex to be deleted has two children.

Case 1: Vertex to be deleted is leaf

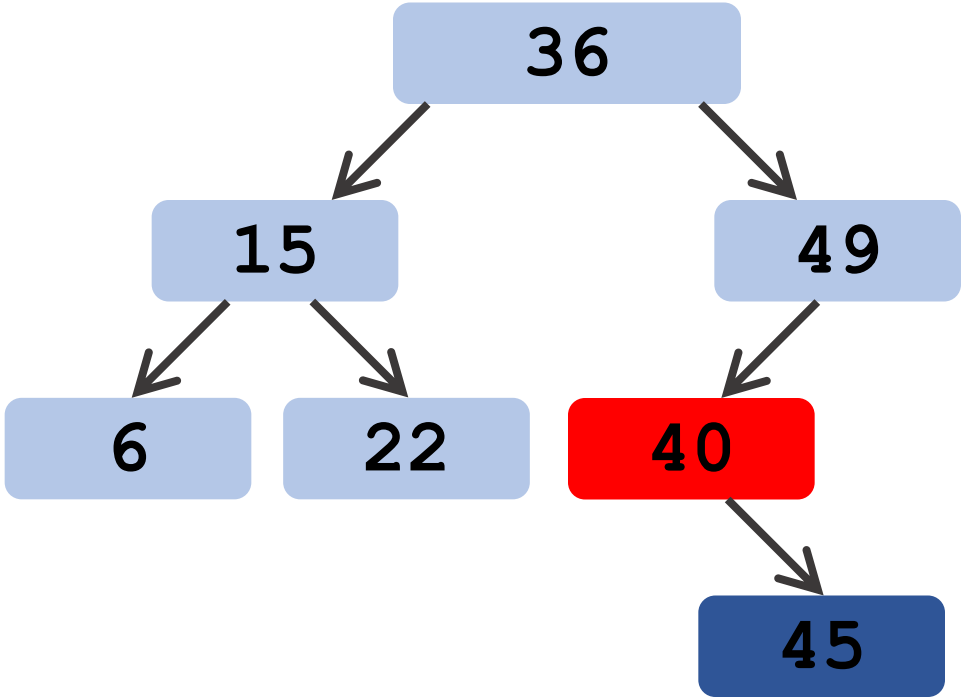


Case 1: Vertex to be deleted is leaf

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

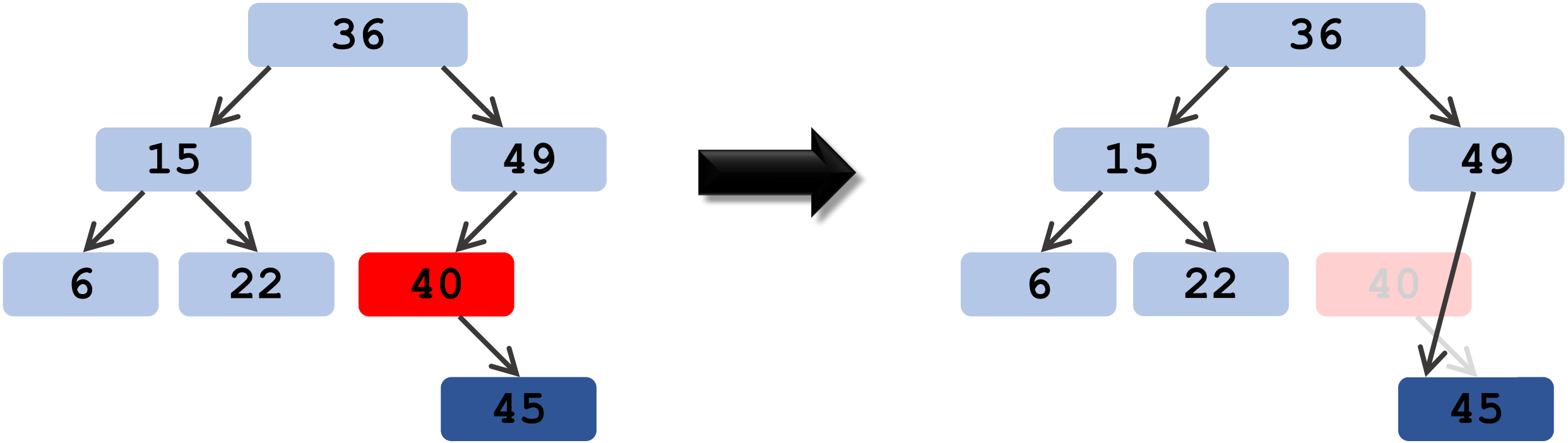


Case 2: Vertex to be deleted has one child



Case 2: Vertex to be deleted has one child

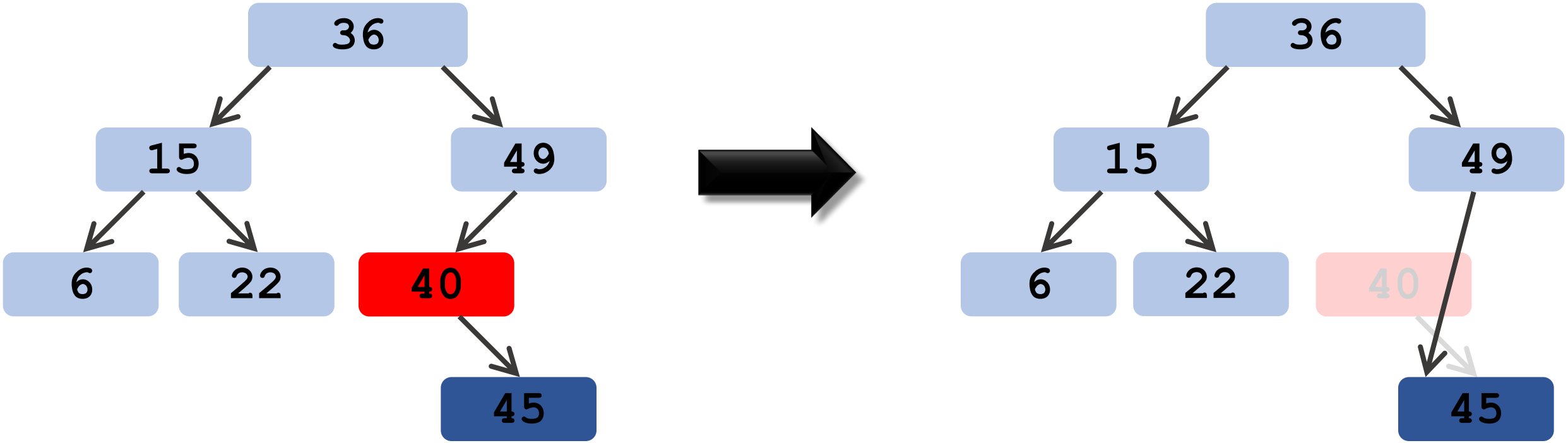
1. Let the vertex's parent point to the vertex's child.



Case 2: Vertex to be deleted has one child

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

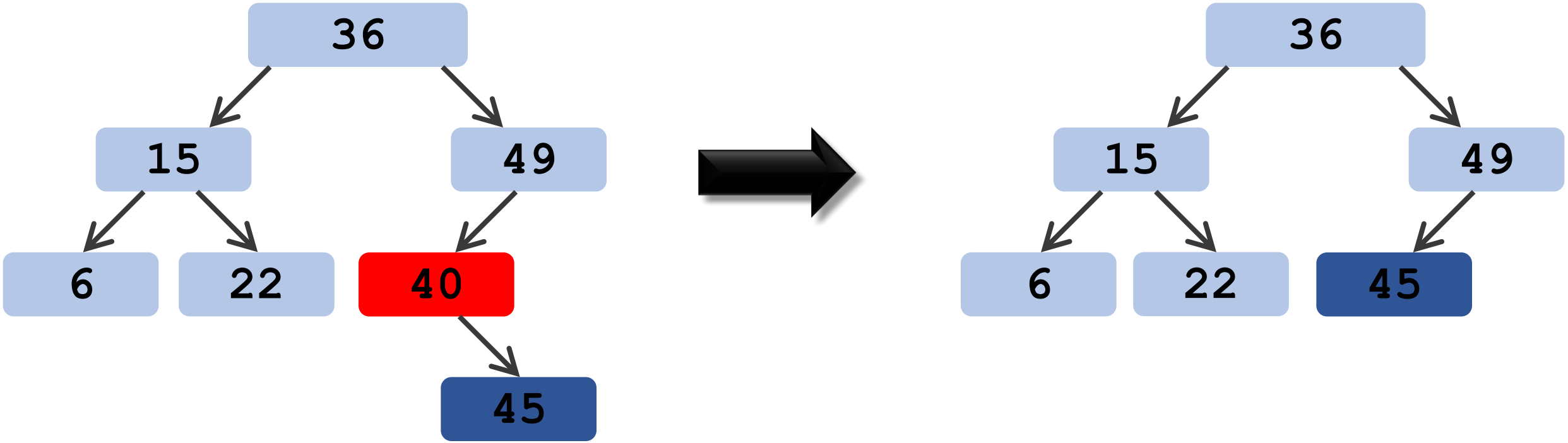
2. Free the vertex from memory.



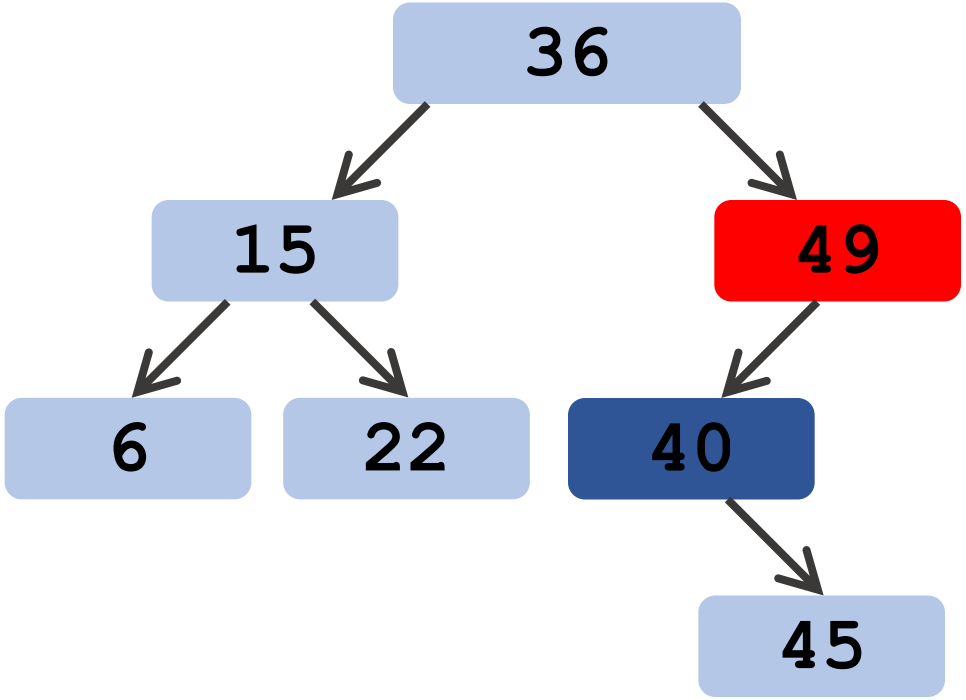
Case 2: Vertex to be deleted has one child

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

2. Free the vertex from memory.

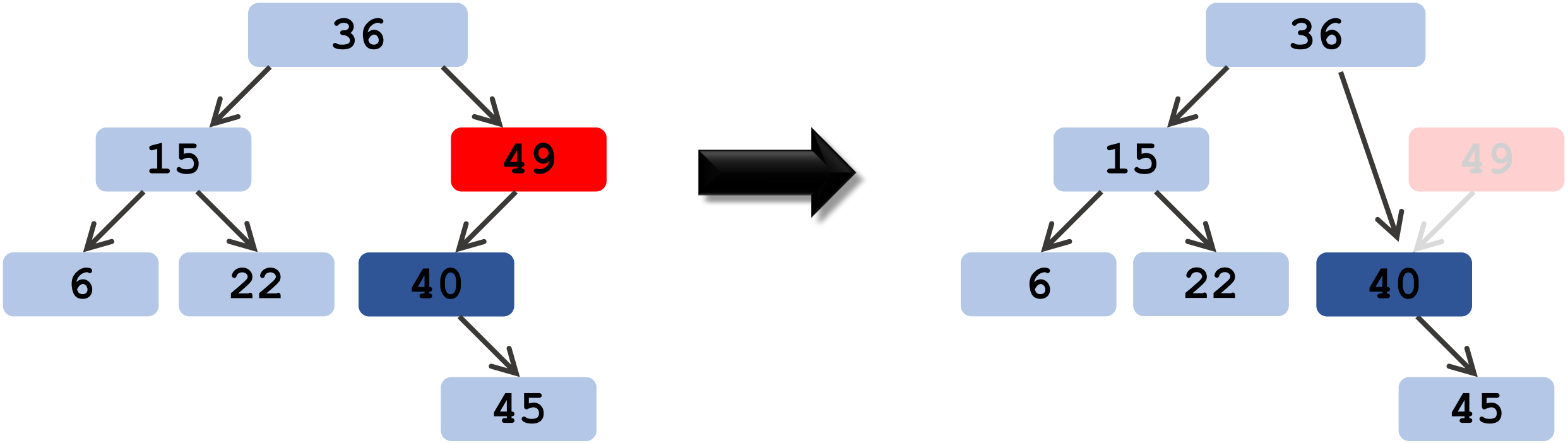


Case 2: Vertex to be deleted has one child



Case 2: Vertex to be deleted has one child

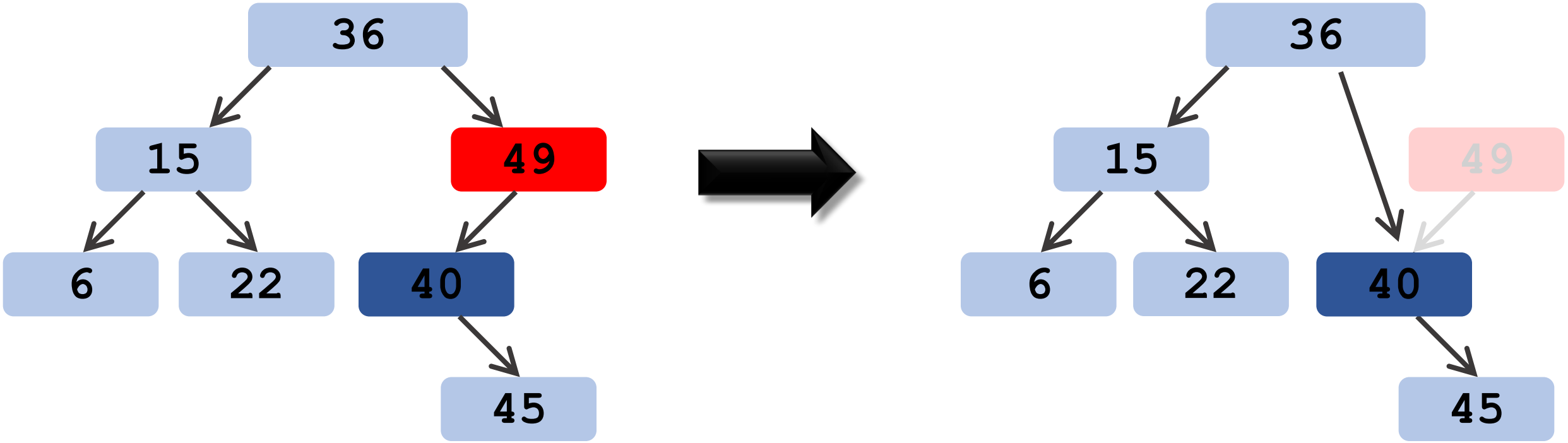
1. Let the vertex's parent point to the vertex's child.



Case 2: Vertex to be deleted has one child

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

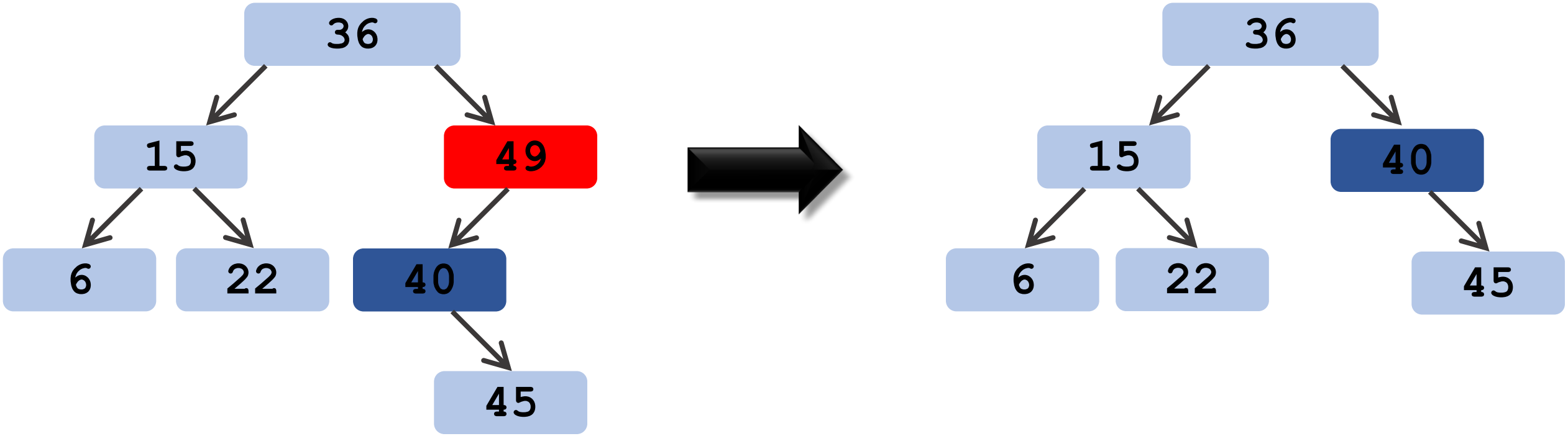
2. Free the vertex from memory.



Case 2: Vertex to be deleted has one child

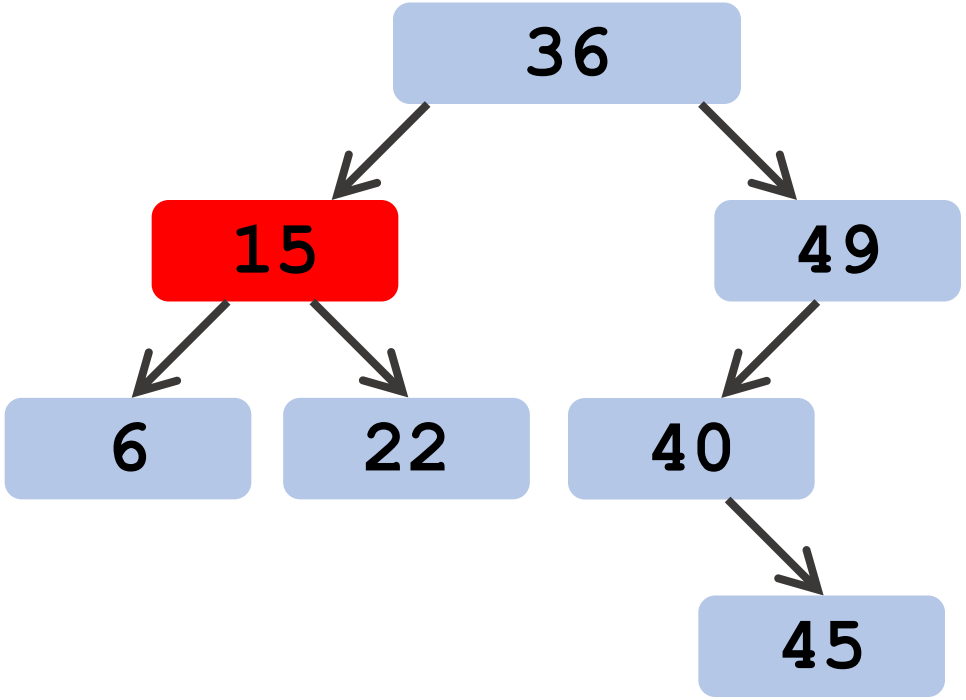
1. Let the vertex's parent point to the vertex's child.

2. Free the vertex from memory.



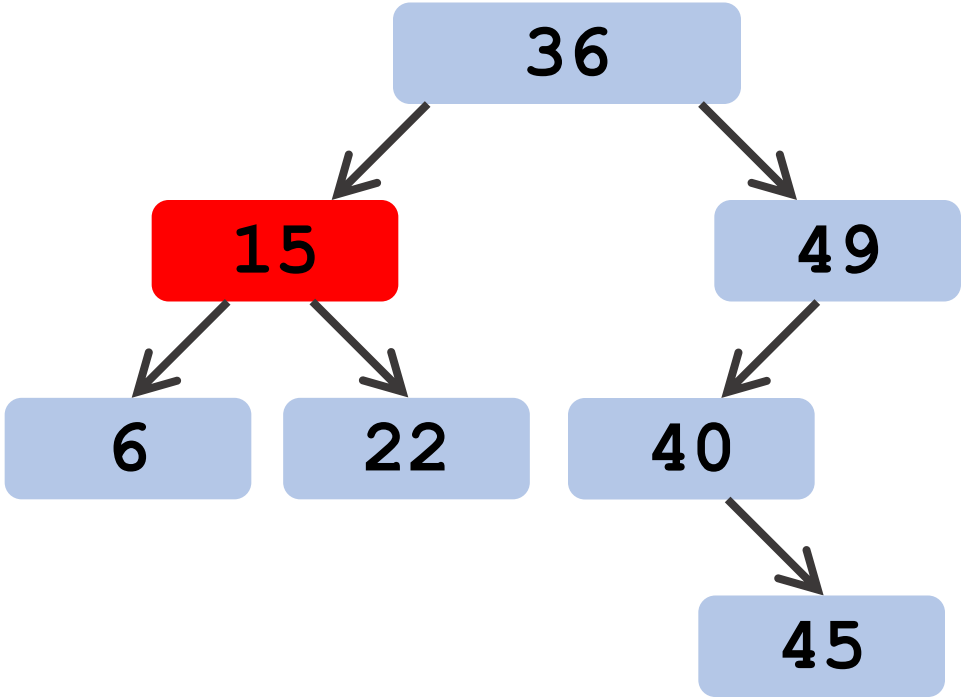
Case 3: Vertex to be deleted has two children

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



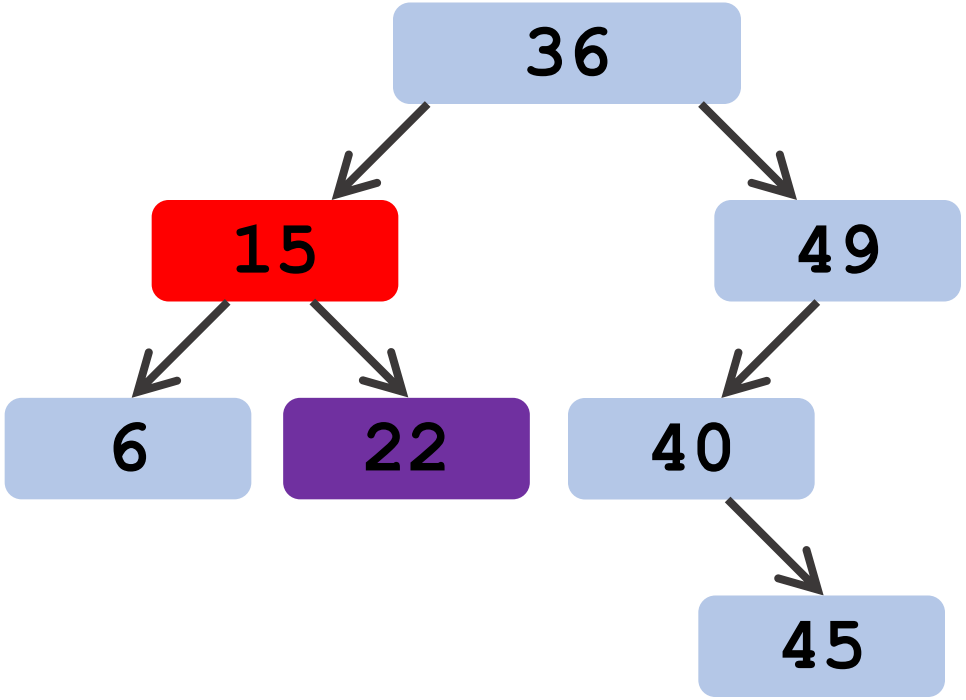
Case 3: Vertex to be deleted has two children

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



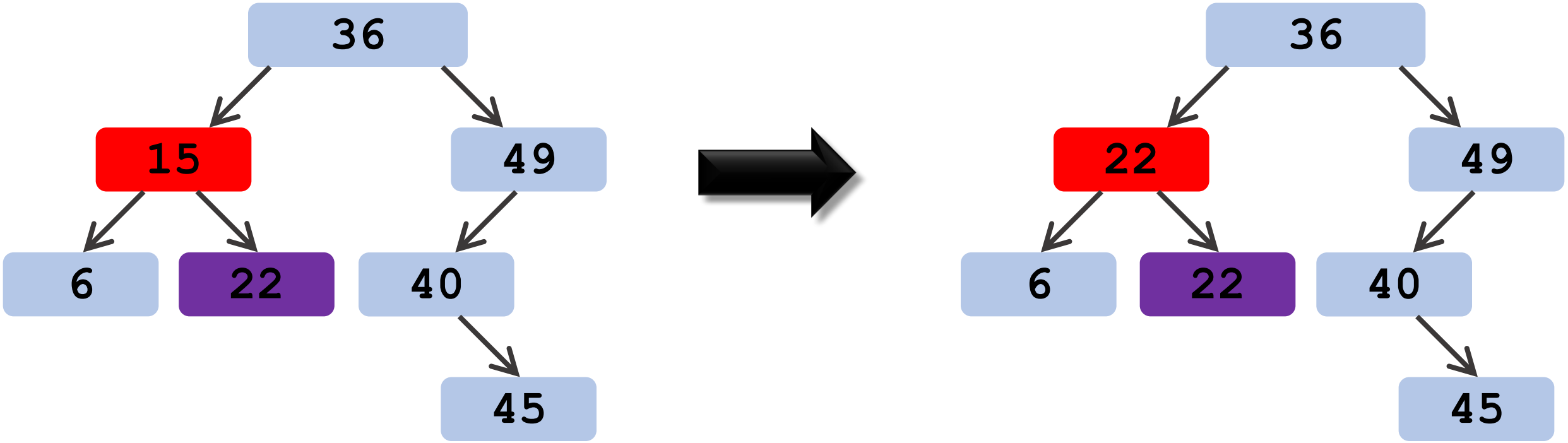
Case 3: Vertex to be deleted has two children

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



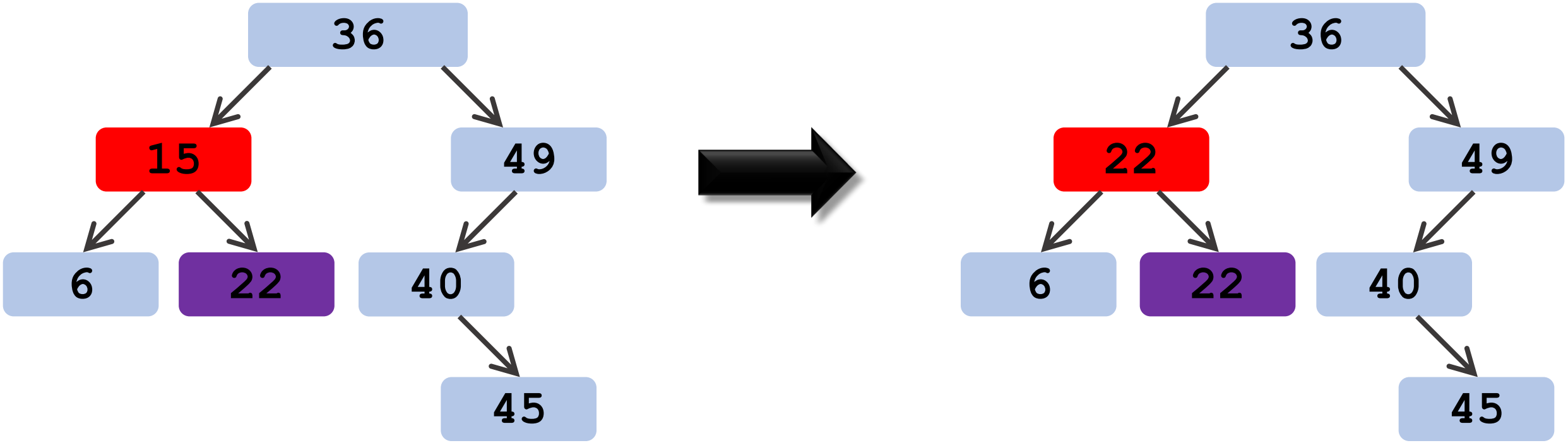
Case 3: Vertex to be deleted has two children

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



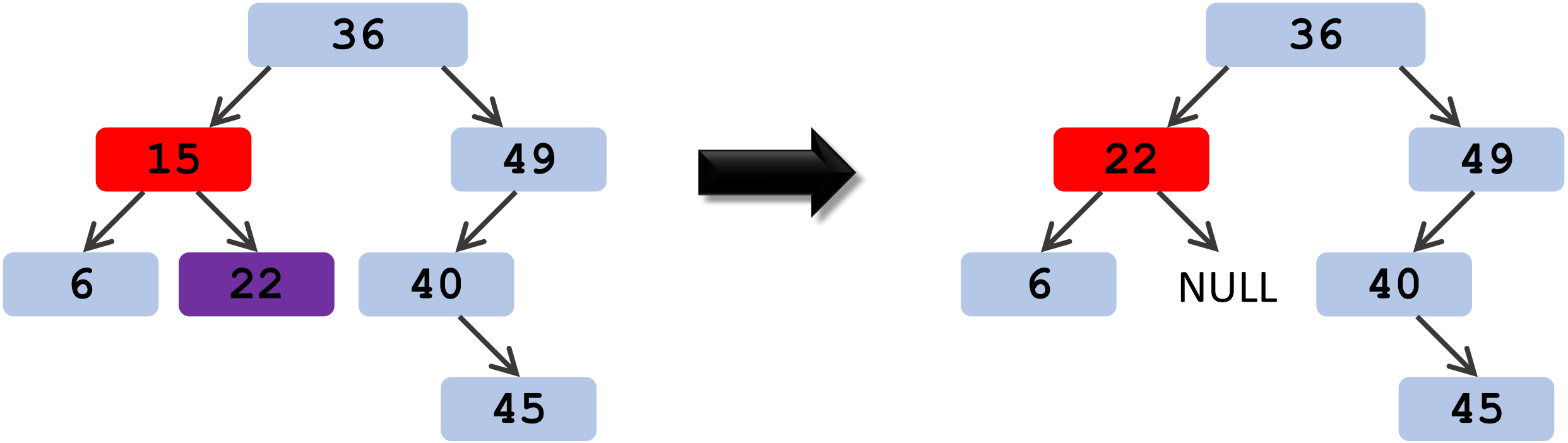
Case 3: Vertex to be deleted has two children

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

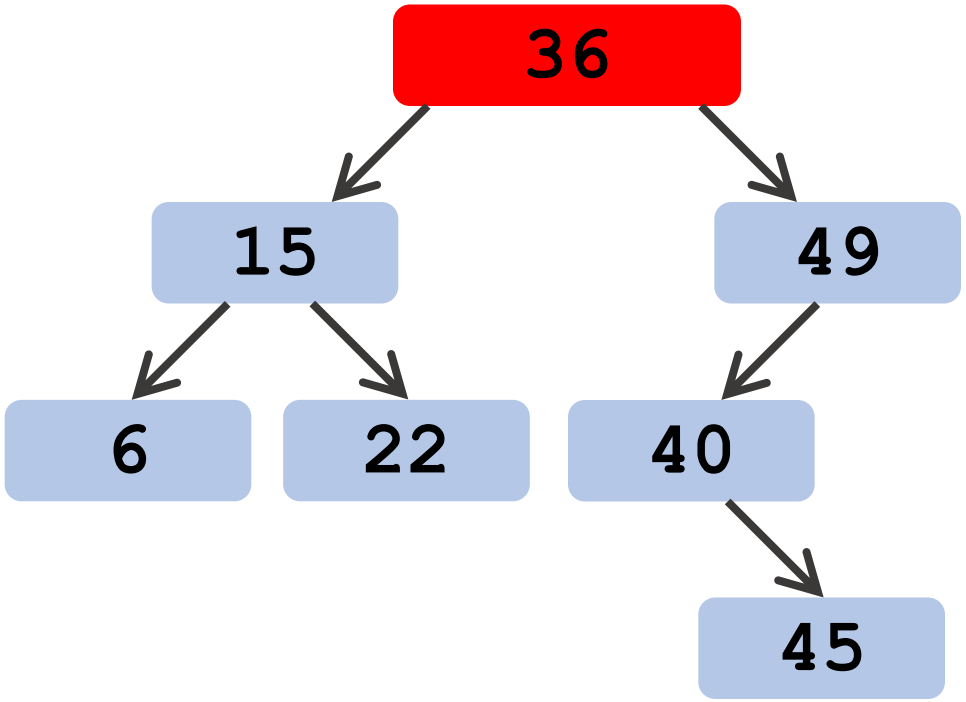


Case 3: Vertex to be deleted has two children

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

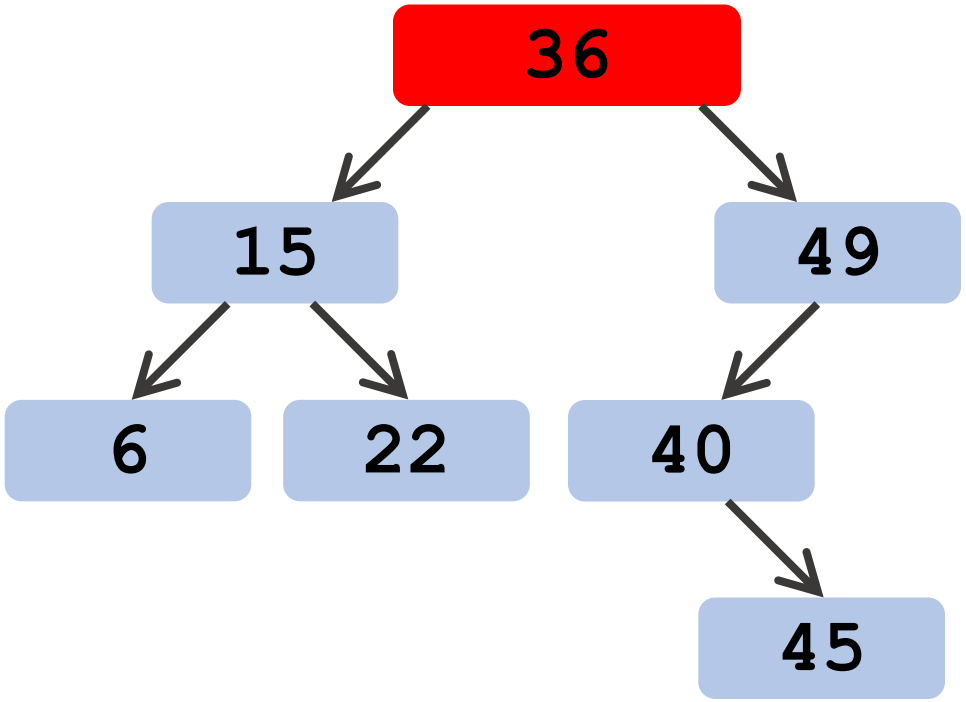


Case 3: Vertex to be deleted has two children



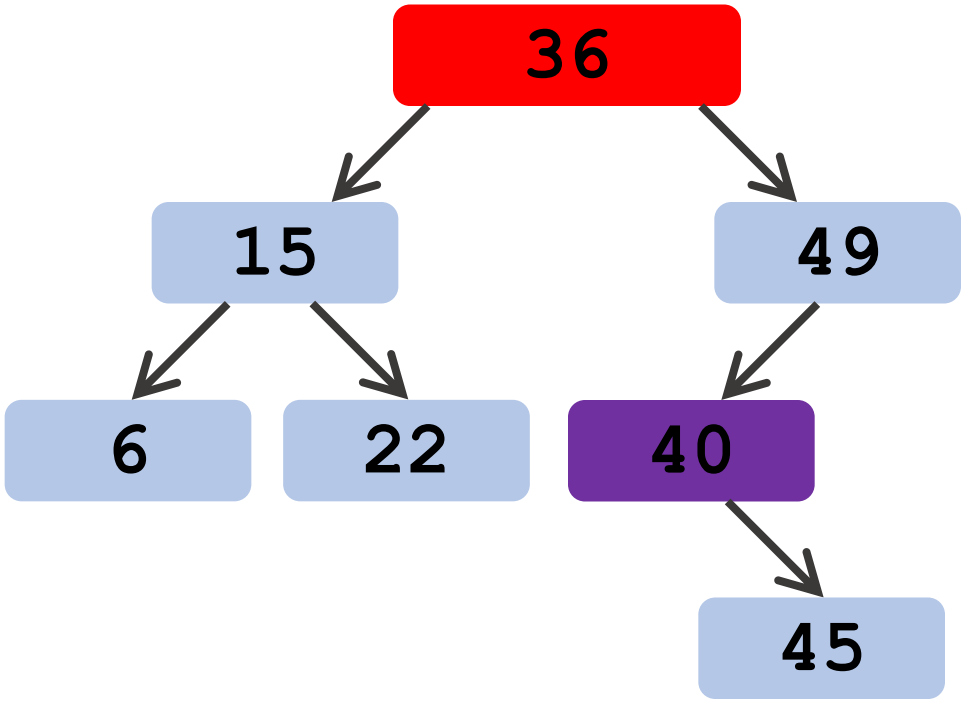
Case 3: Vertex to be deleted has two children

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



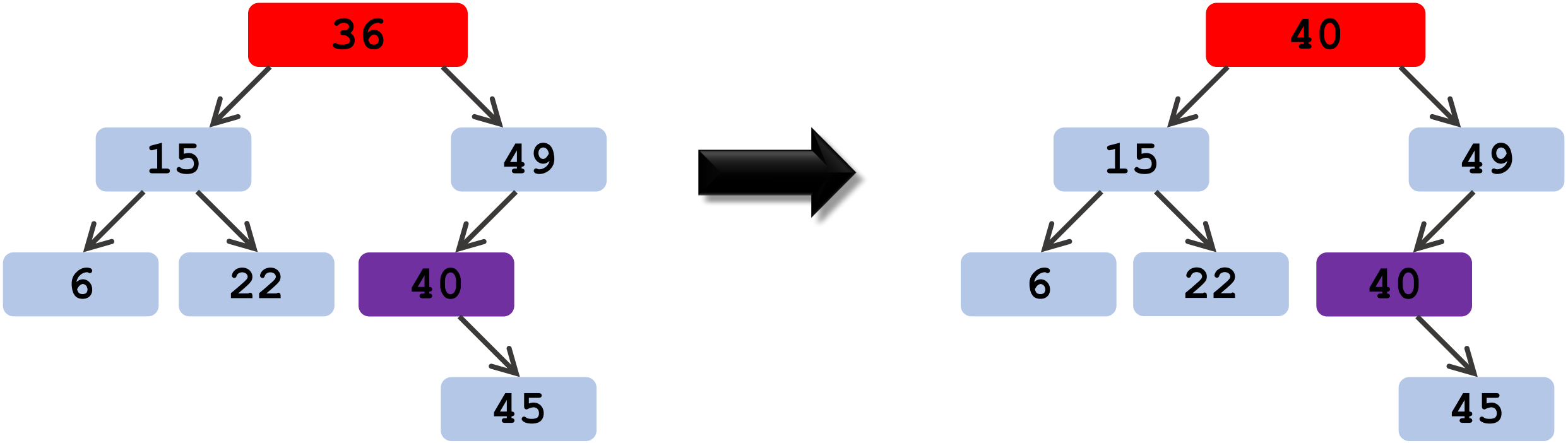
Case 3: Vertex to be deleted has two children

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



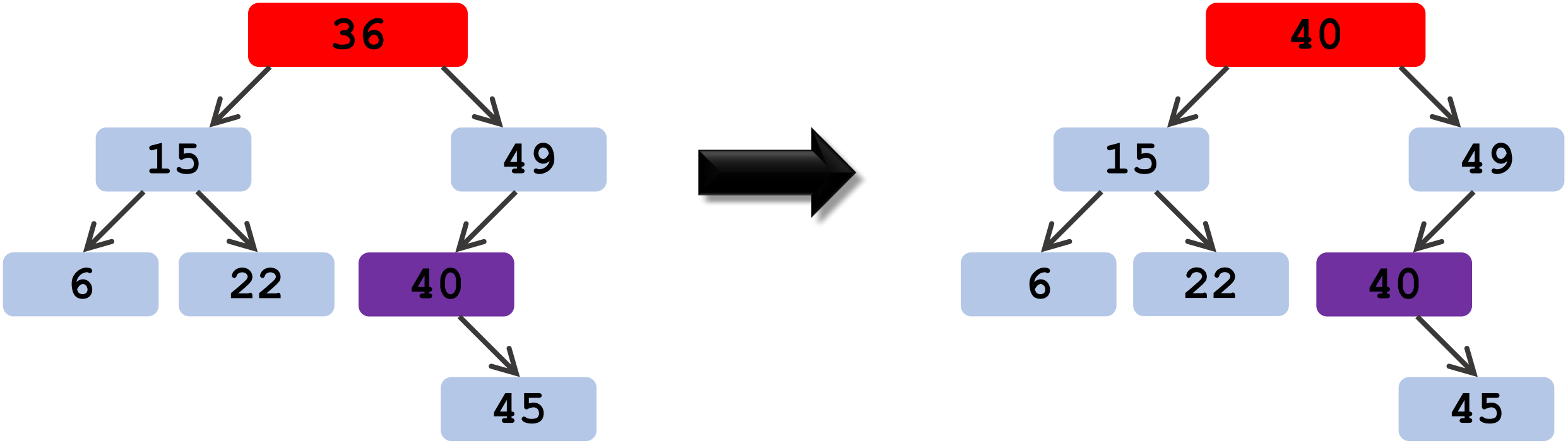
Case 3: Vertex to be deleted has two children

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



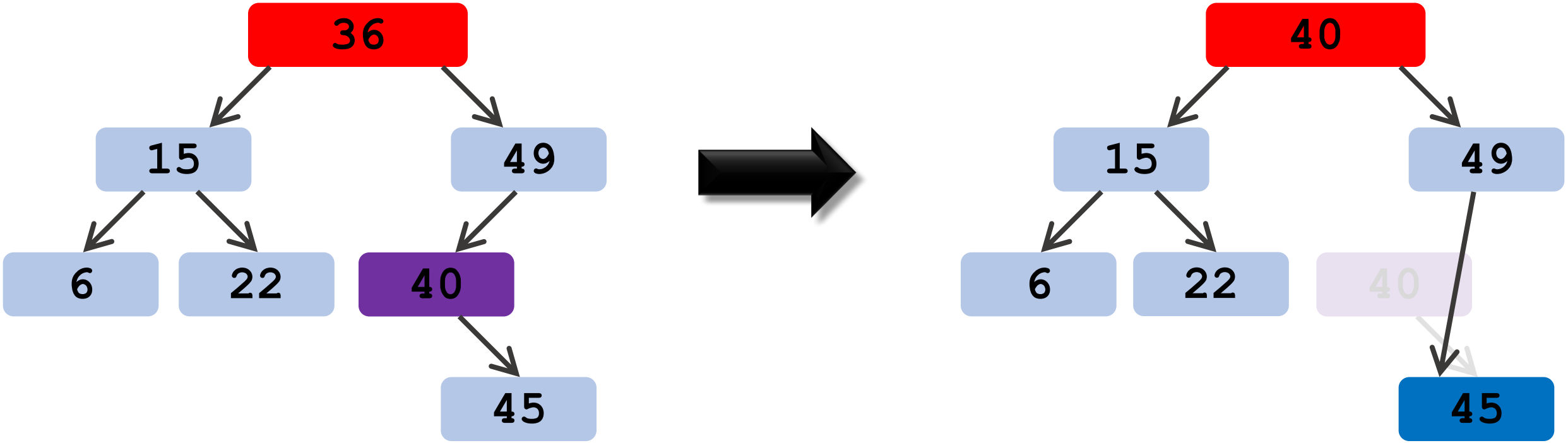
Case 3: Vertex to be deleted has two children

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



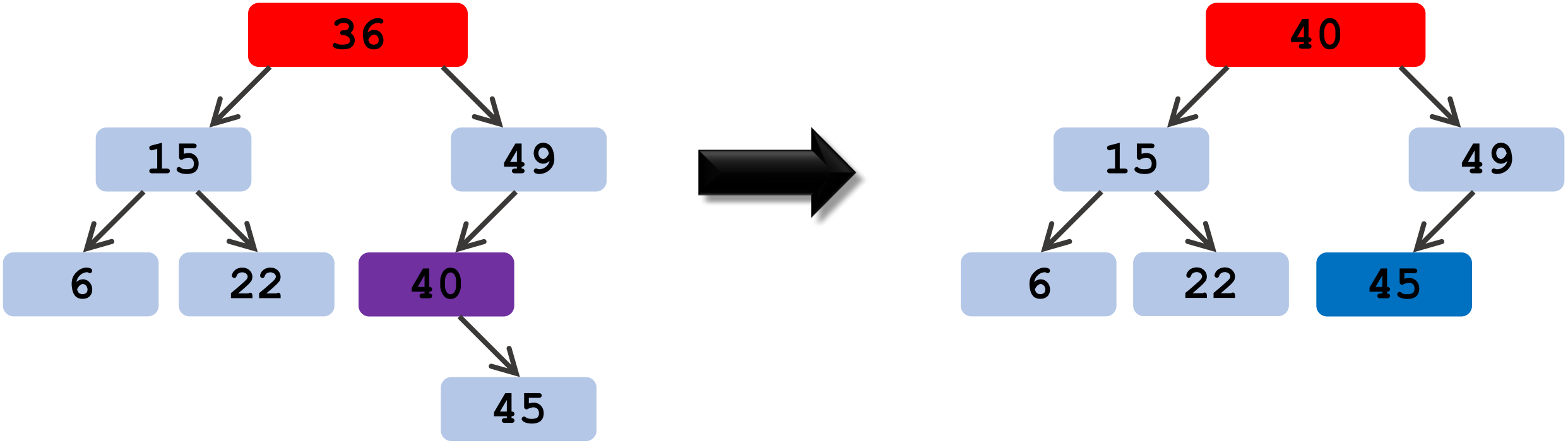
Case 3: Vertex to be deleted has two children

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.



```c
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
```

```
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
```

```
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
```

```
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
```

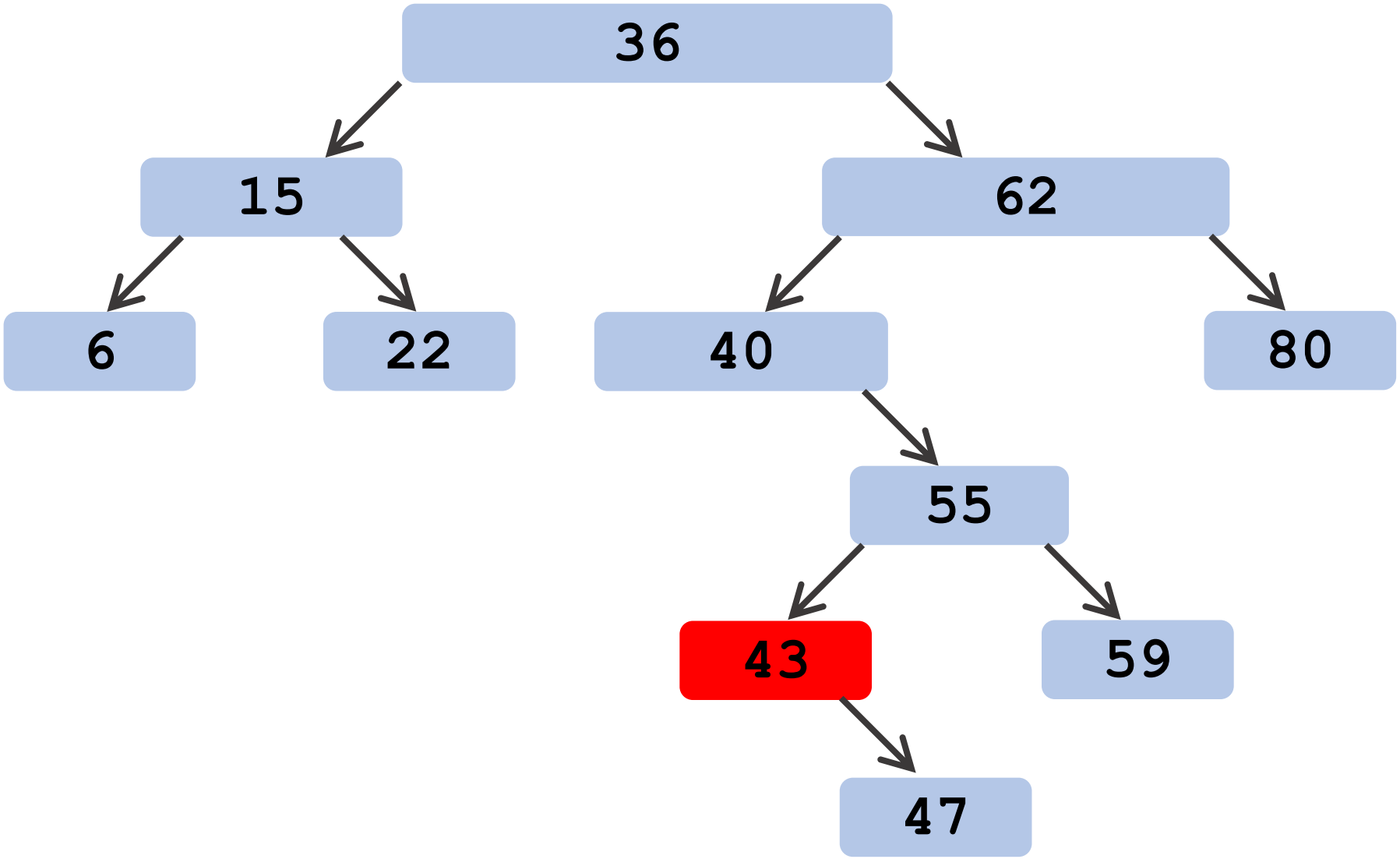
```
// the root has only one child
else if (root->left == NULL) {
    struct vertex* v = root->right;
    delete root;
    return v;
}
else if (root->right == NULL) {
    struct vertex* v = root->left;
    delete root;
    return v;
}

// ... to continue
```

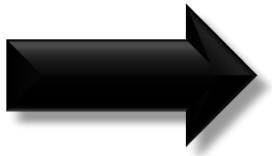
```
// the root has only one child
else if (root->left == NULL) {
      struct vertex* v = root->right;
      delete root;
      return v;
}
else if (root->right == NULL) {
      struct vertex* v = root->left;
      delete root;
      return v;
}

// ... 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

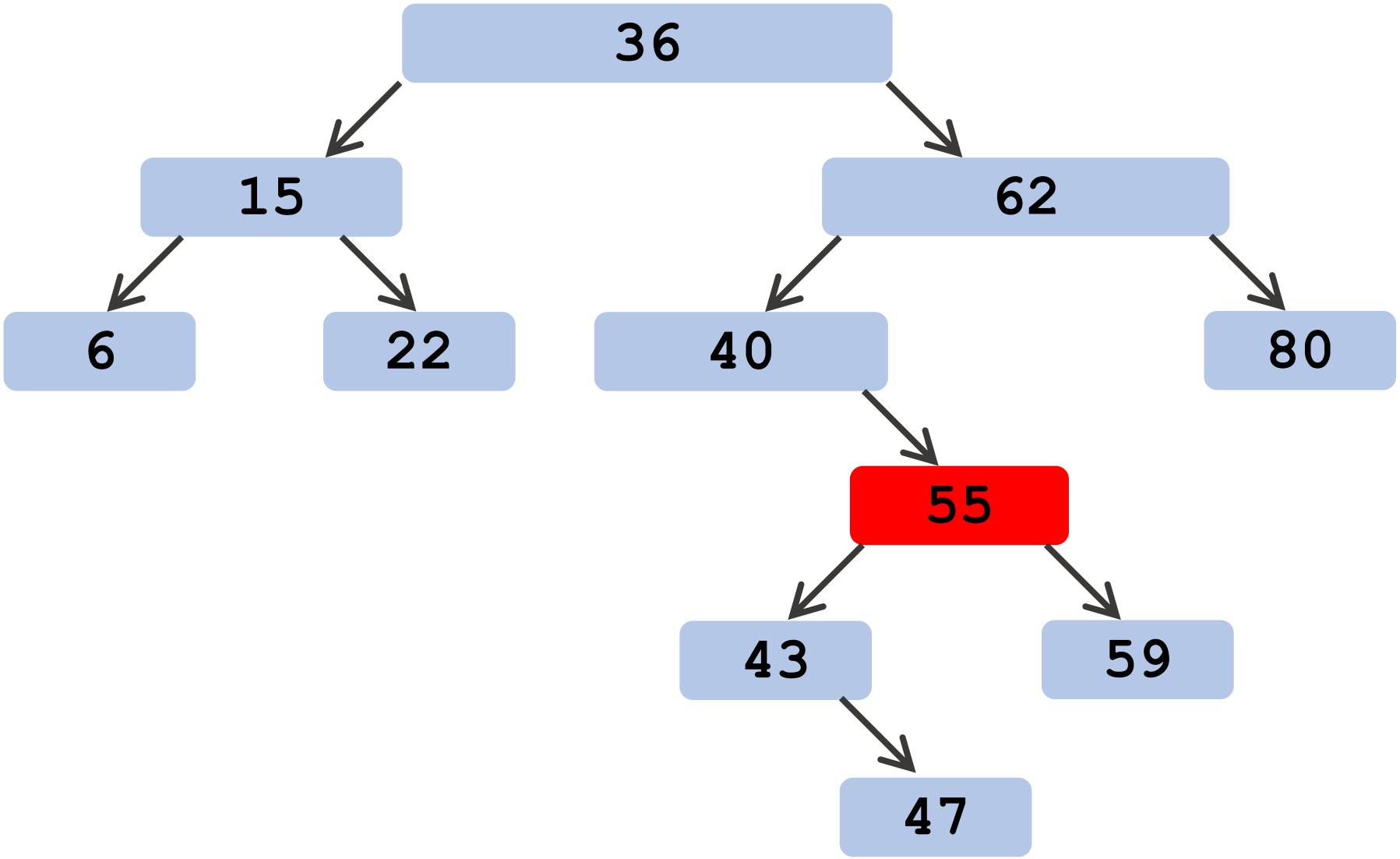
Question 1: Draw the tree after deleting 43





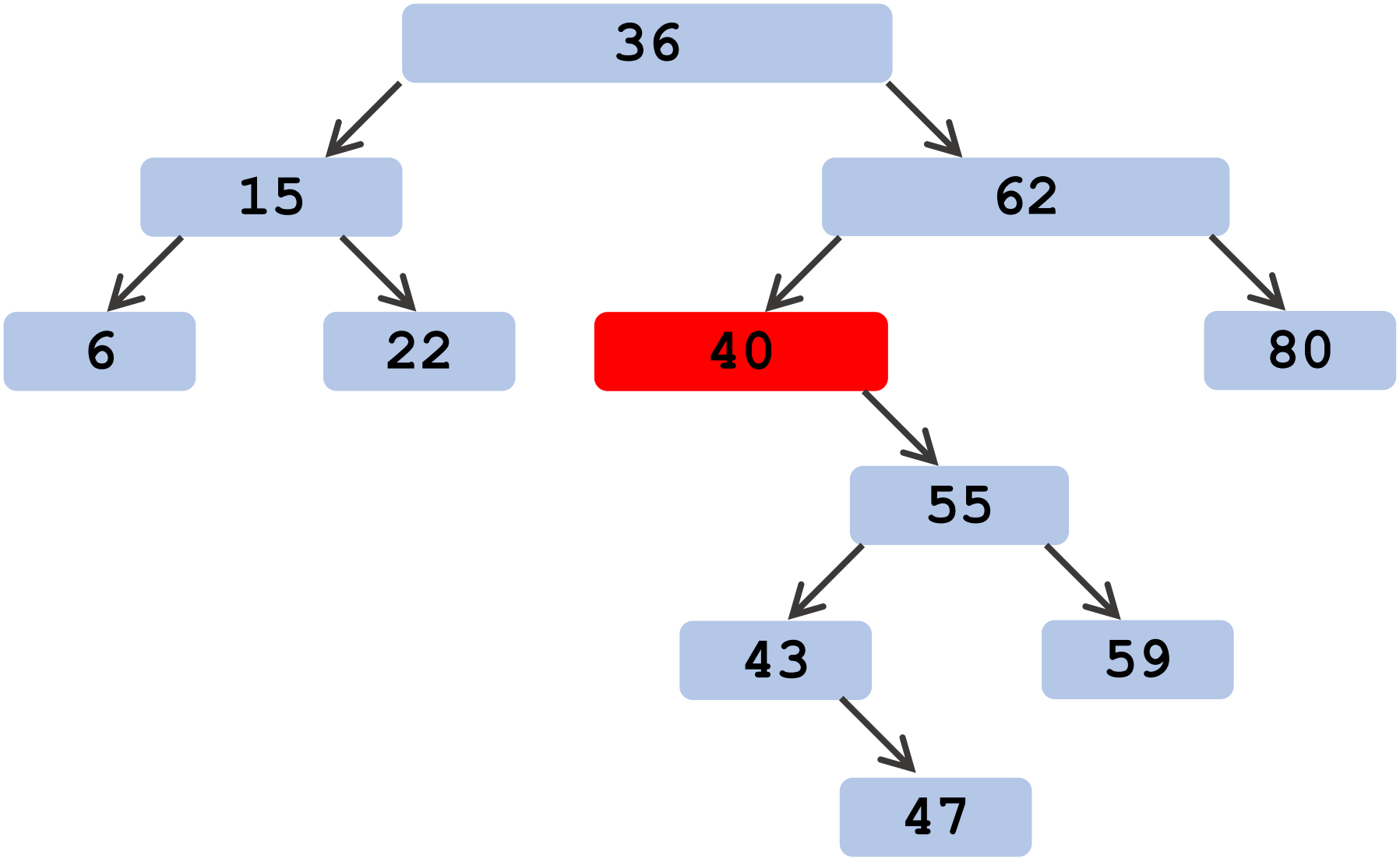
?

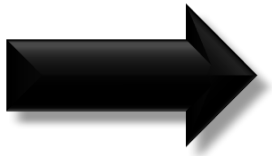
Question 2: Draw the tree after deleting 55



?

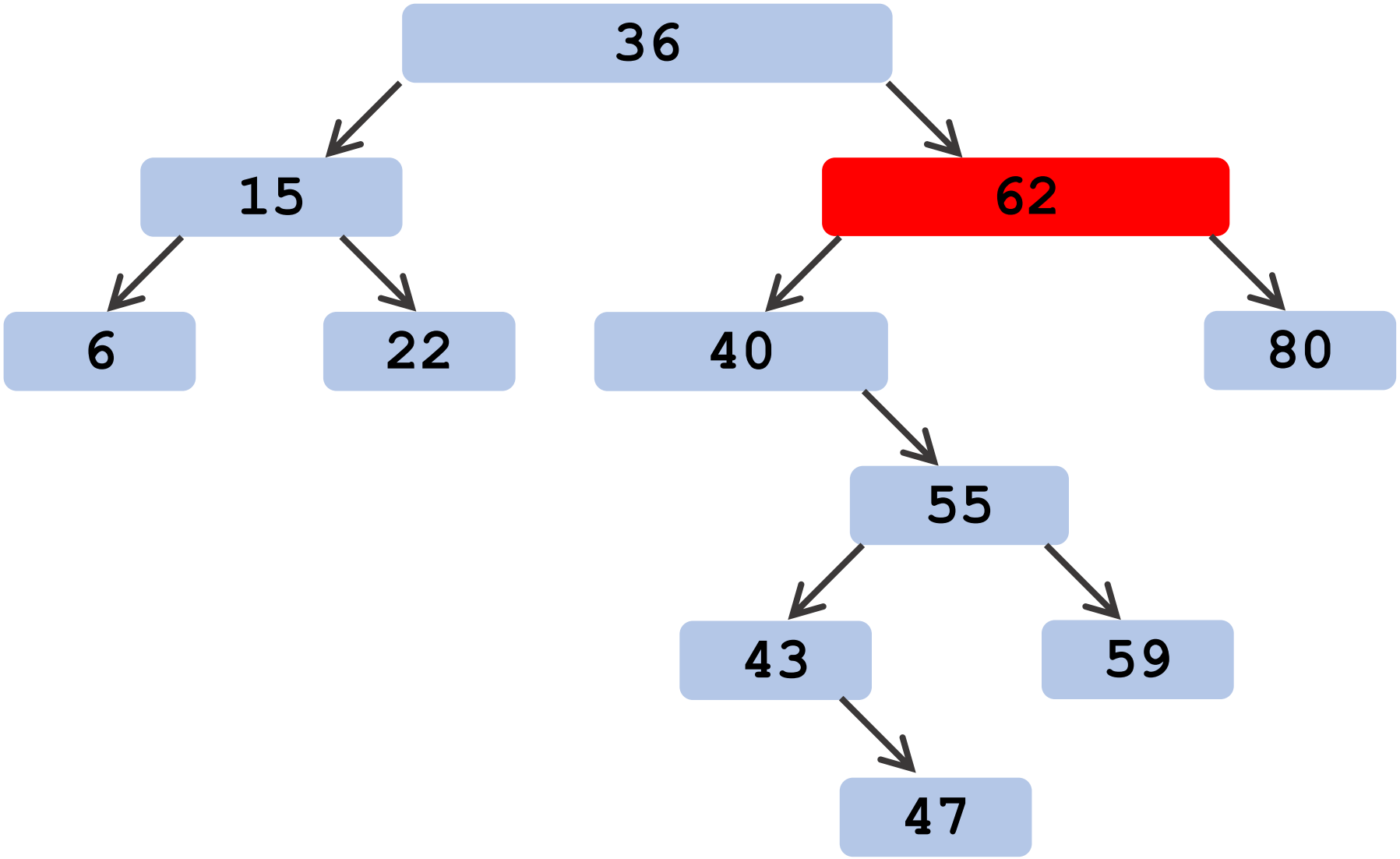
Question 3: Draw the tree after deleting 40





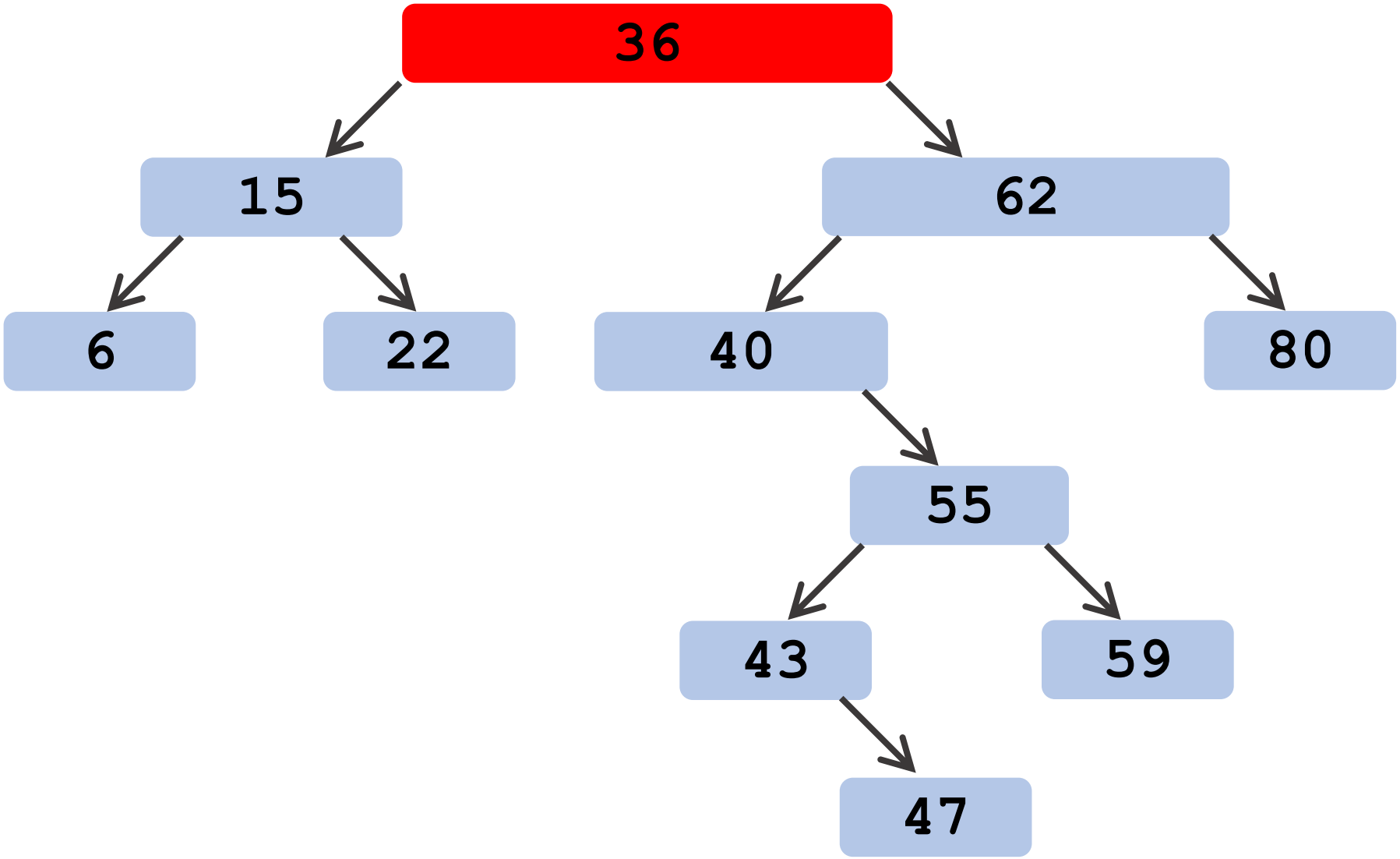
?

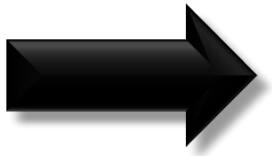
Question 4: Draw the tree after deleting 62



?

Question 5: Draw the tree after deleting 36





?

Thank You!

http://wangshusen.github.io/