CSE 3010 – Data Structures & Algorithms

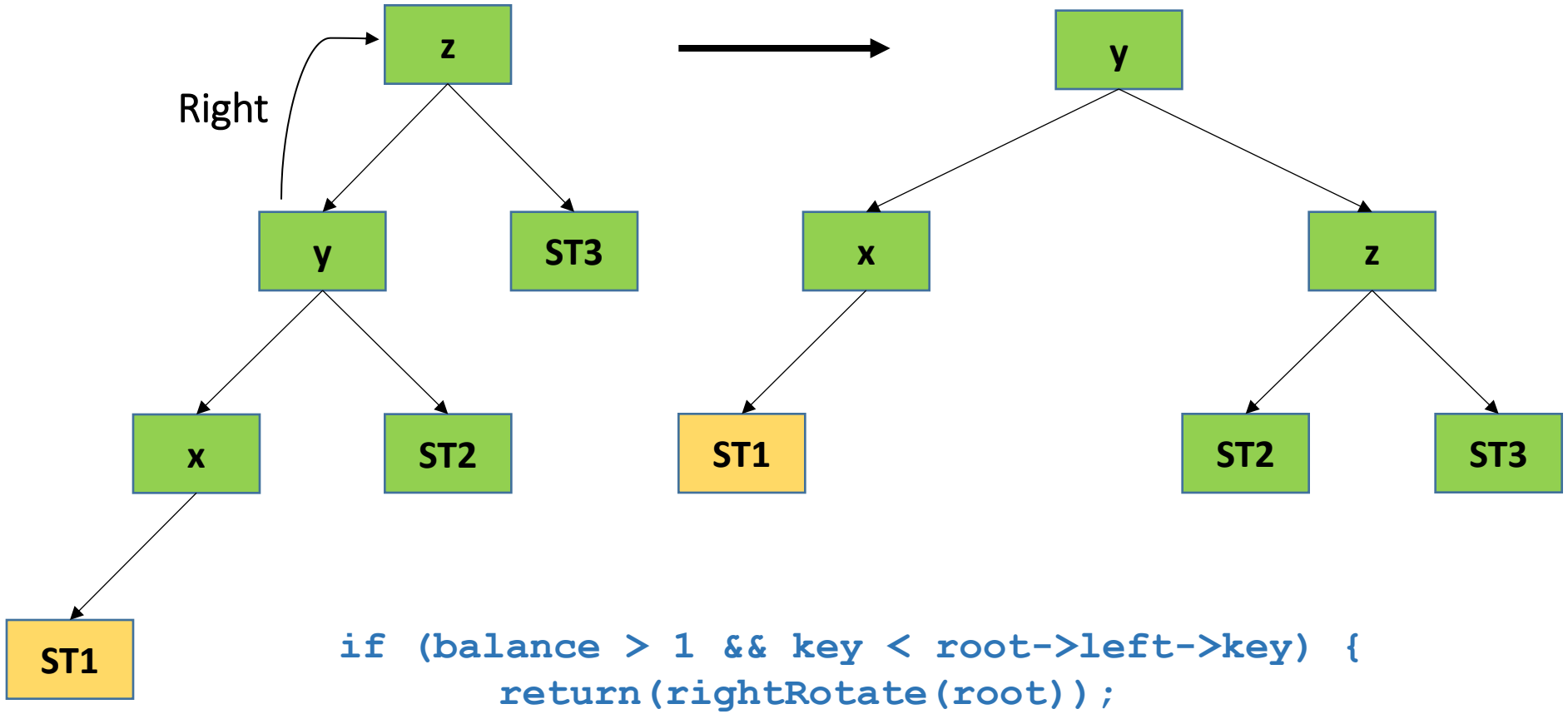
Lecture #43

What will be covered today

- Insertion of a new node in an AVL tree
- Deletion of a node in an AVL tree

Left Left Case

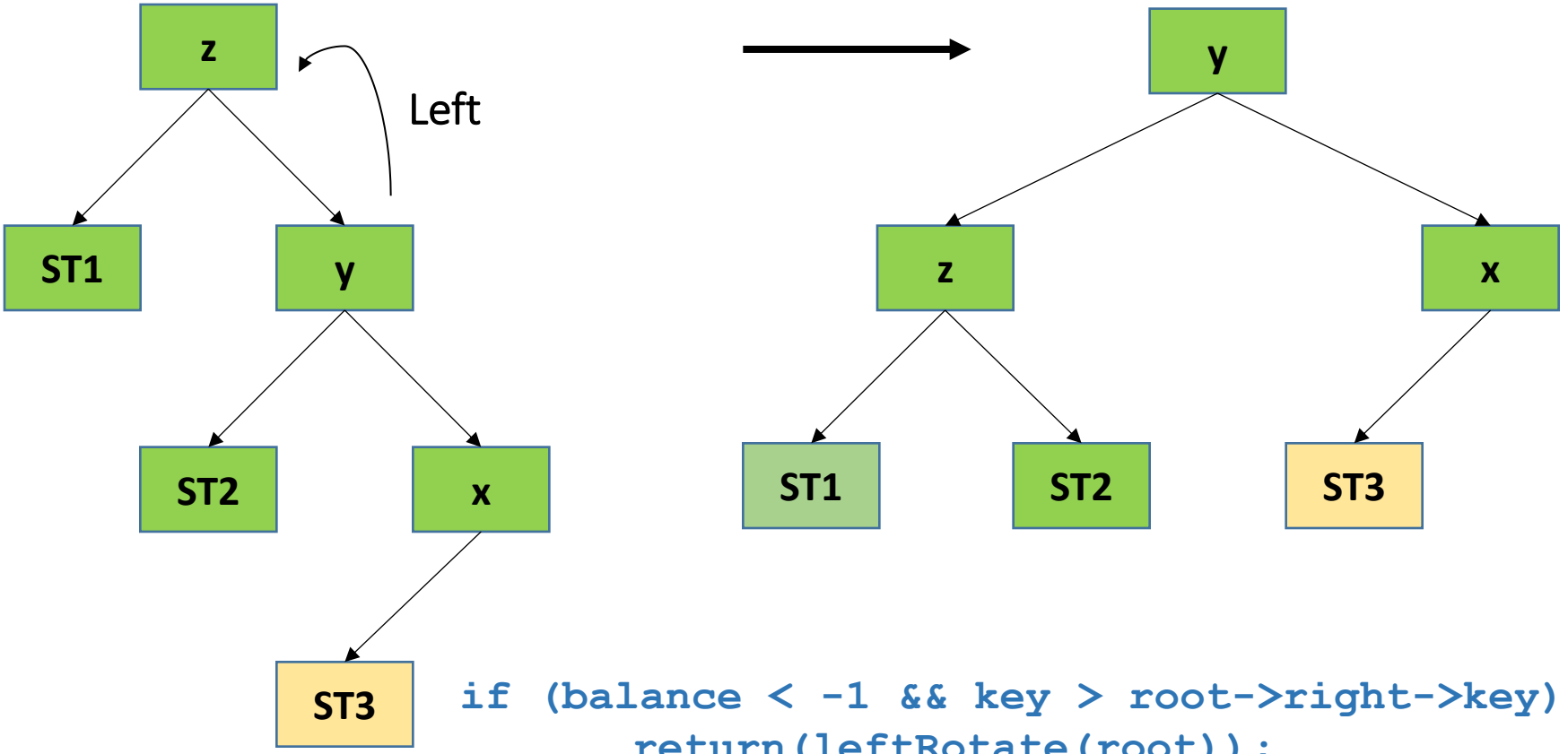
y is the left child of **z** and **x** is the left child of **y**




```
if (balance > 1 && key < root->left->key) {
        return(rightRotate(root));
}
```

Right Right Case

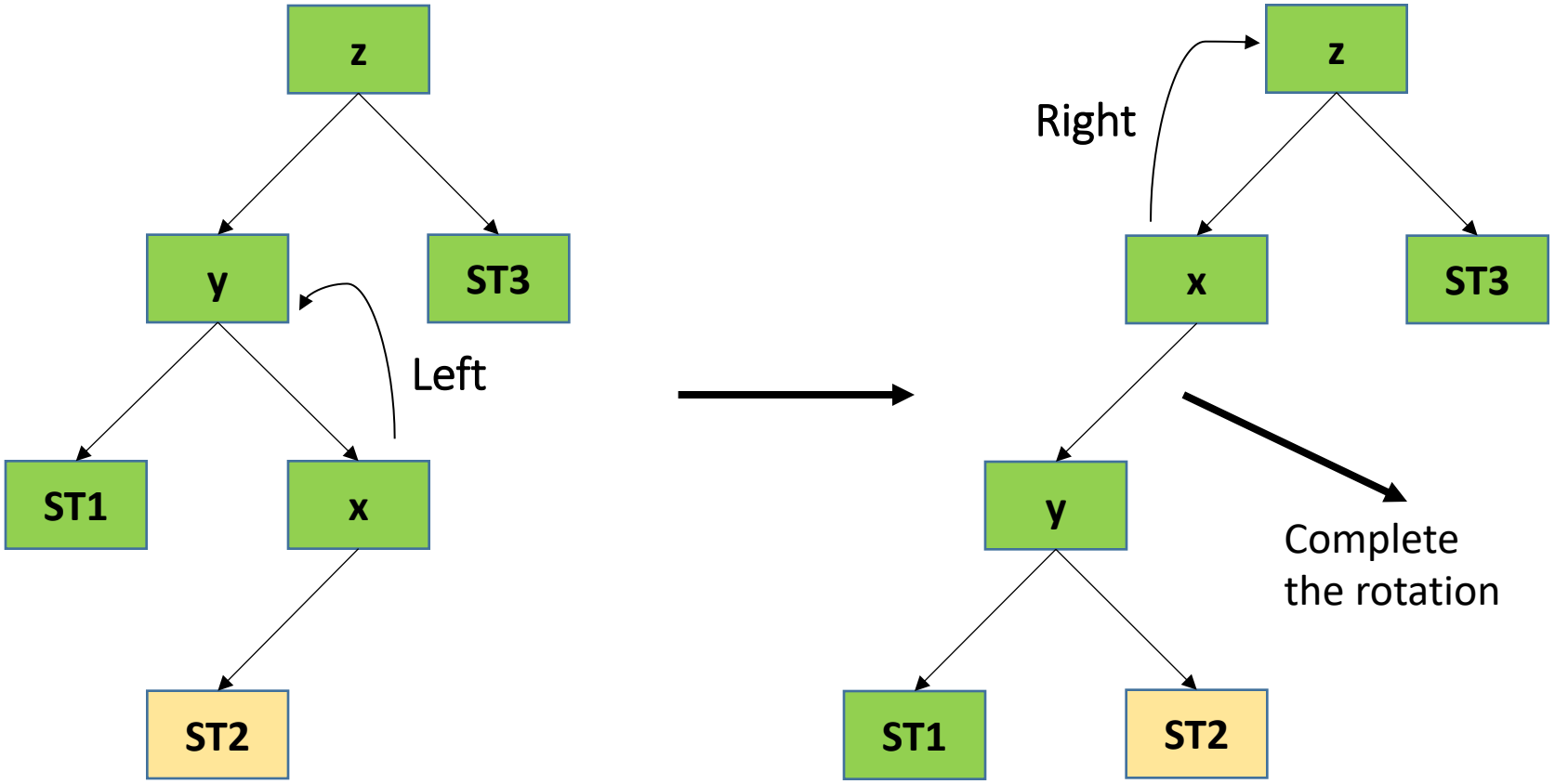
y is the right child of **z** and **x** is the right child of **y**




```
if (balance < -1 && key > root->right->key) {
        return(leftRotate(root));
}
```

Left Right Case

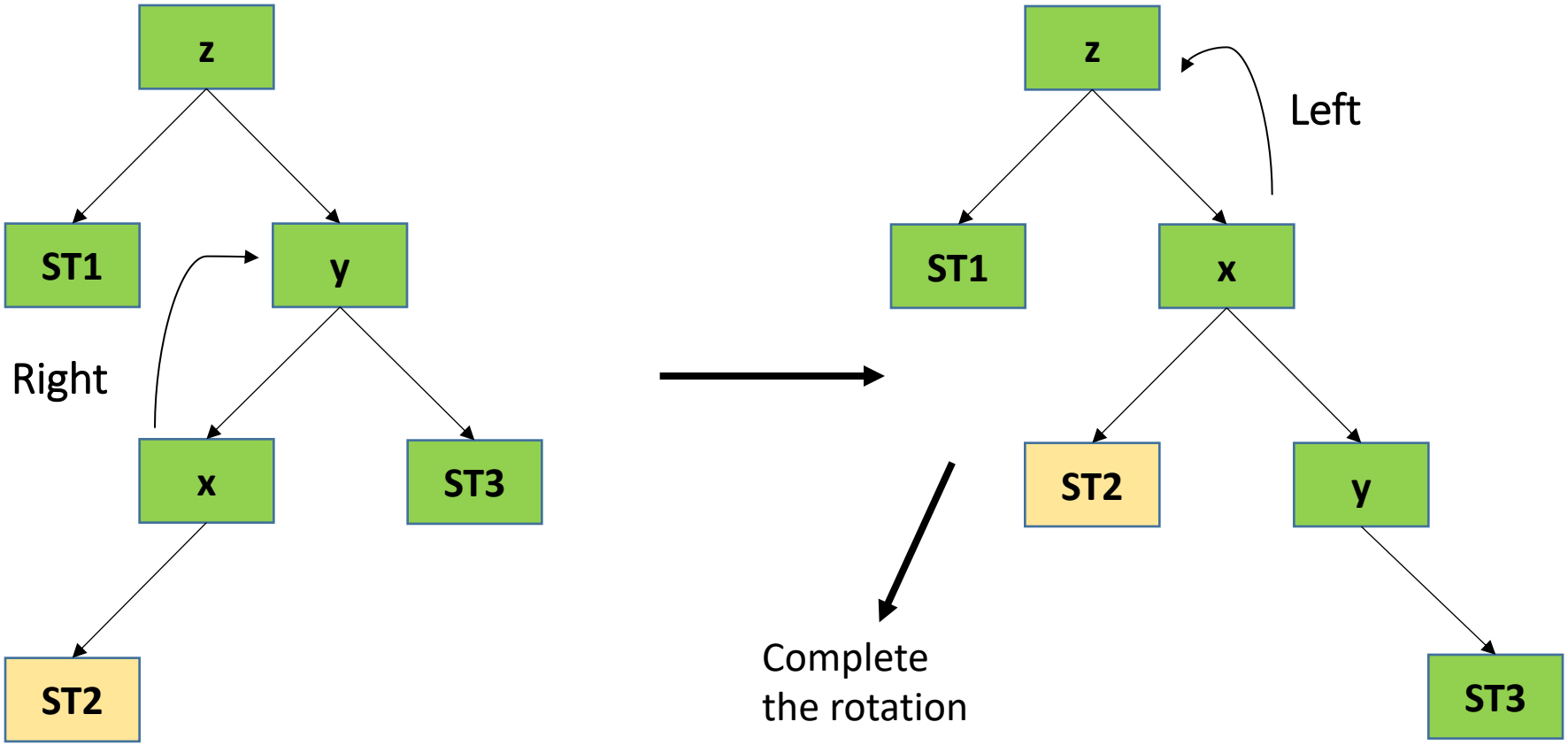
y is the left child of **z** and **x** is the right child of **y**




```
if (balance > 1 && key > root->left->key) {
        root->left = leftRotate(root->left);
        return(rightRotate(root));
}
```

Right Left Case

y is the right child of **z** and **x** is the left child of **y**




```
if (balance < -1 && key < root->right->key) {
        root->right = rightRotate(root->right);
        return(leftRotate(root));
}
```

Deletion in AVL Trees

- Delete a node as you would normally do in a BST. Let this node be **`w`**.
- Traverse up from **`w`** to **`root`**. Let **`z`** be the first unbalanced node in this path. Let **`y`** be the child of **`z`** with larger height and **`x`** be the grandchild of **`z`** with larger height.
- Rebalance the tree by performing one of the following operations depending on the relationship between **`x`**, **`y`**, and **`z`**
  - **Left Left Case**: **`y`** is the left child of **`z`** and **`x`** is the left child of **`y`**
  - **Left Right Case**: **`y`** is the left child of **`z`** and **`x`** is the right child of **`y`**
  - **Right Right Case**: **`y`** is the right child of **`z`** and **`x`** is the right child of **`y`**
  - **Right Left Case**: **`y`** is the right child of **`z`** and **`x`** is the left child of **`y`**
- If balancing the node **`z`** does not fix the AVL tree, balance the ancestors of **`z`** as well.