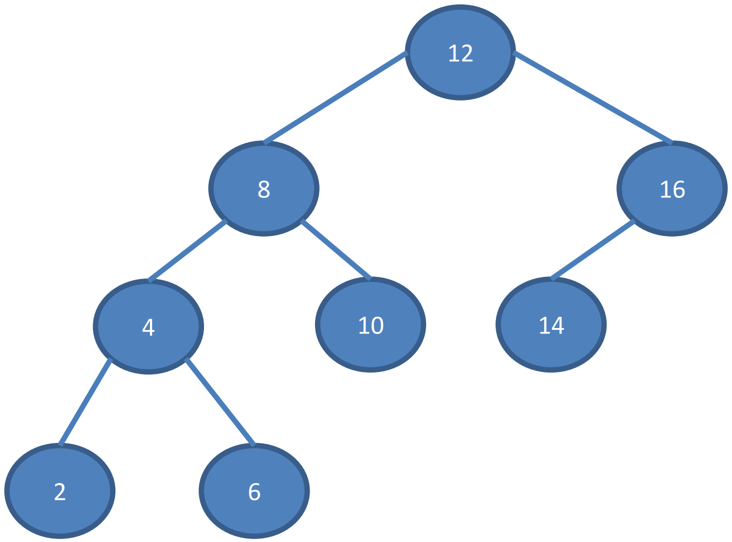
DIT181: Data Structures and Algorithms

AVL Trees

Gül Calikli
Email: calikli@chalmers.se

AVL Trees

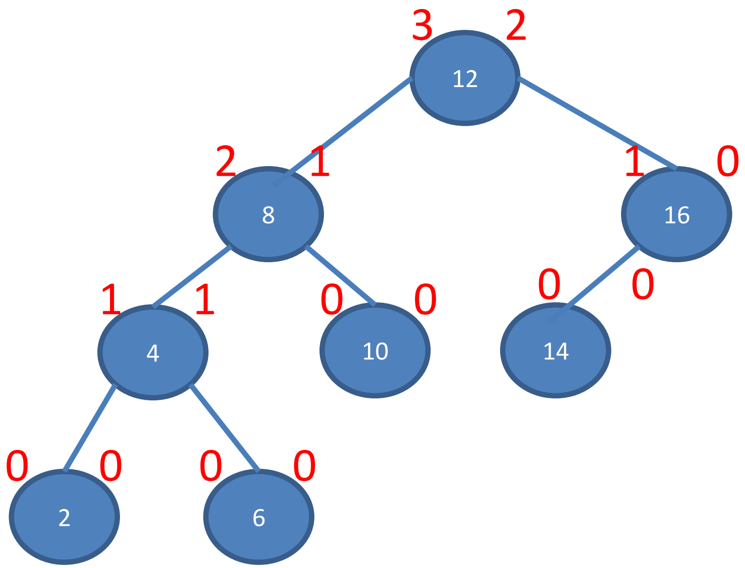
- Definition: An AVL tree is a binary search tree with additional balance property that for any node in the tree, the height of the left and right subtrees can differ by 1.



Question: Is this an AVL tree?

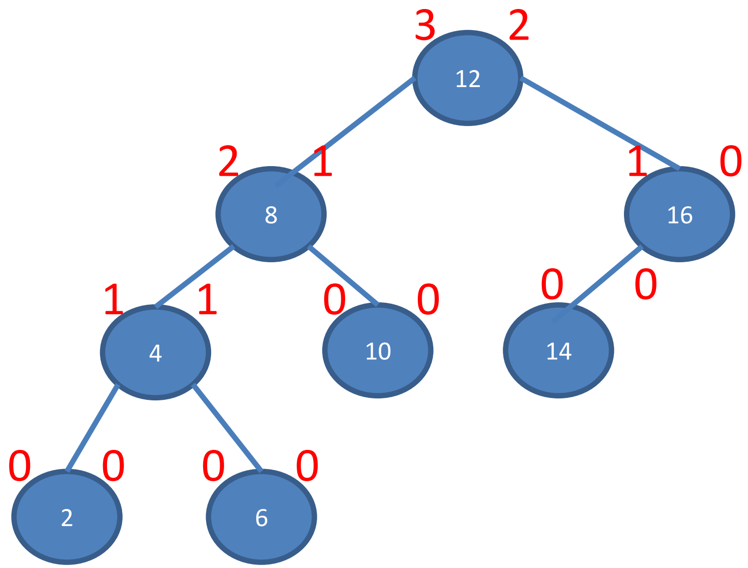
AVL Trees

- Definition: An AVL tree is a binary search tree with additional balance property that for any node in the tree, the height of the left and right subtrees can differ by 1.



AVL Trees

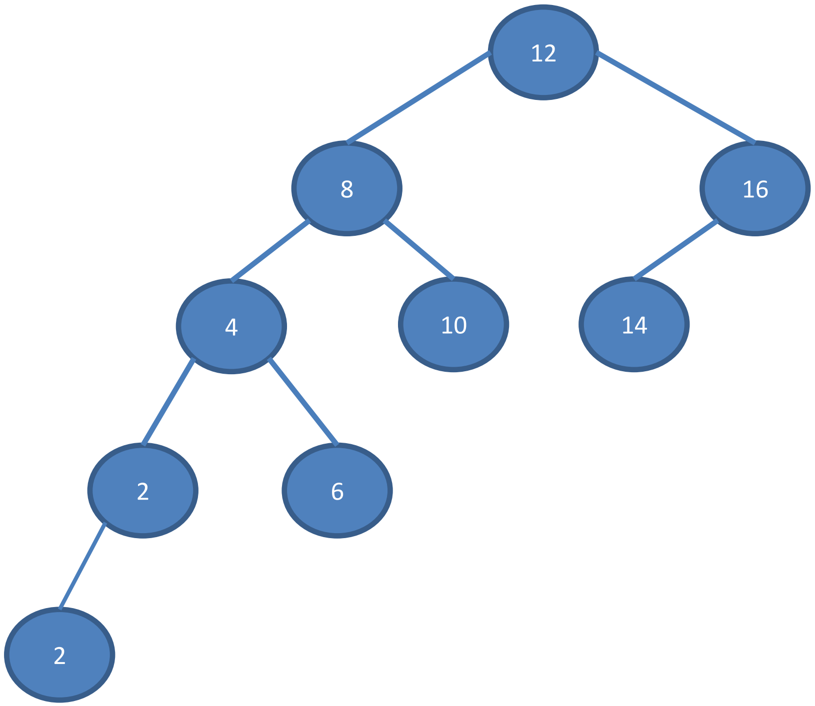
- Definition: An AVL tree is a binary search tree with additional balance property that for any node in the tree, the height of the left and right subtrees can differ by 1.



Answer: Acc. To definition, this is an AVL tree!

AVL Trees

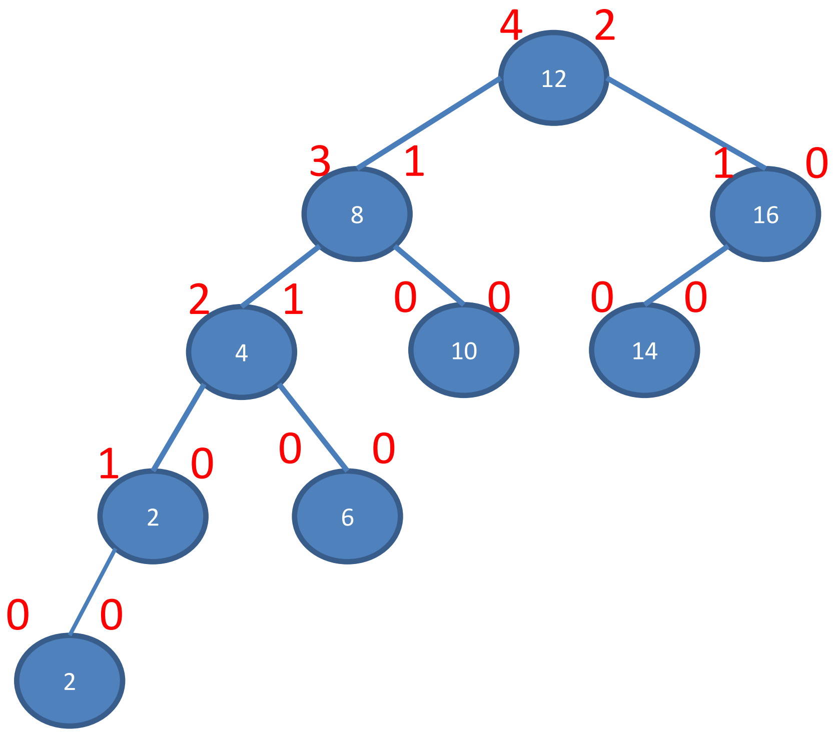
- Definition: An AVL tree is a binary search tree with additional balance property that for any node in the tree, the height of the left and right subtrees can differ by 1.



Question: Is this an AVL tree?

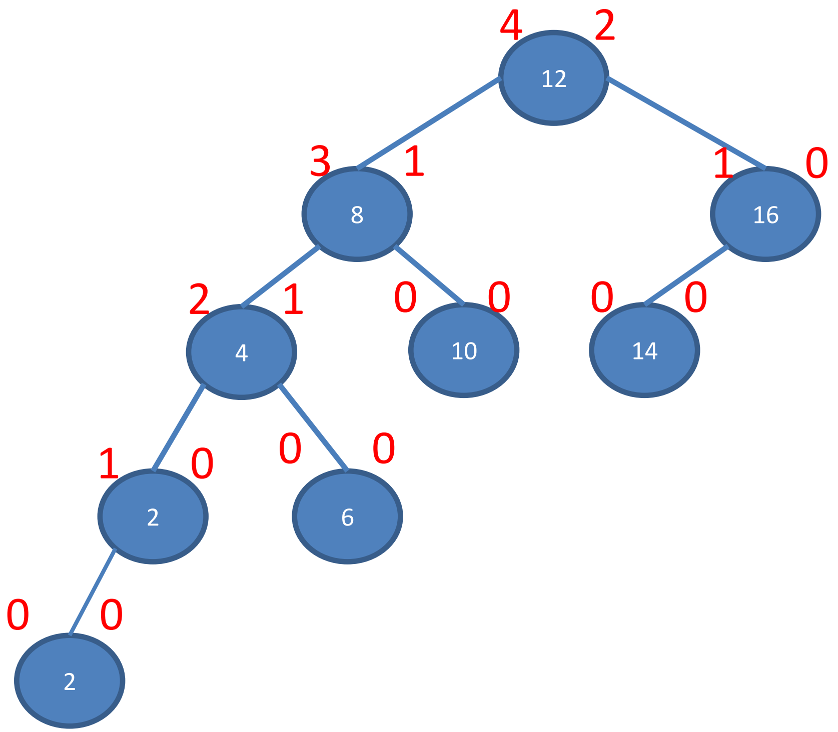
AVL Trees

- Definition: An AVL tree is a binary search tree with additional balance property that for any node in the tree, the height of the left and right subtrees can differ by 1.



AVL Trees

- Definition: An AVL tree is a binary search tree with additional balance property that for any node in the tree, the height of the left and right subtrees can differ by 1.



Answer: Height of the subtrees for both nodes “12” and “8” differ by 2. Hence, this is not an AVL tree.

AVL Trees: Properties

- Question: What is the height of an empty AVL tree?

AVL Trees: Properties

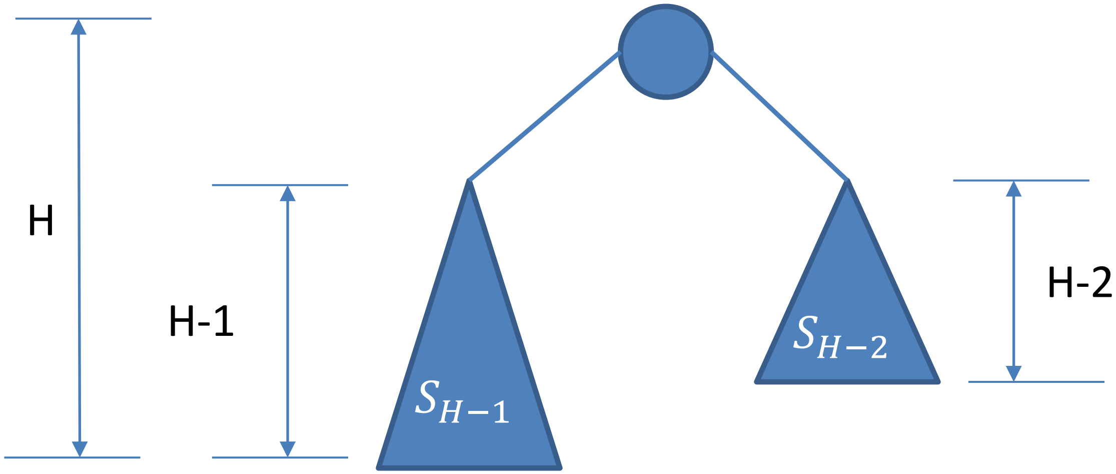
- Answer: The height of an empty AVL tree is **-1**.
- Also, height of an AVL tree is always logarithmic. → In order to prove this, let's prove the following theorem.

Theorem: An AVL tree of height H has at least $F_{H+3} - 1$, where F_i is the i^{th} Fibonacci number.

AVL Trees: Properties

Theorem: An AVL tree of height H has at least $F_{H+3} - 1$, where F_i is the i^{th} Fibonacci number.

- Let S_H be the minimum number of nodes in an AVL tree of height H can have.
- Then a minimum tree of height H will look like the following:

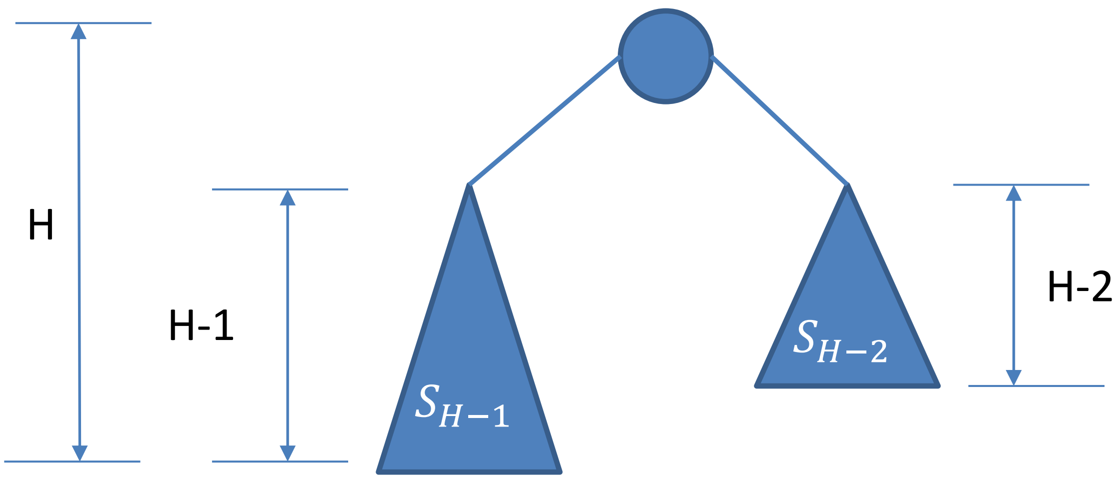


Question:
$S_0 = ?$
$S_1 = ?$

AVL Trees: Properties

Theorem: An AVL tree of height H has at least $F_{H+3} - 1$, where F_i is the i^{th} Fibonacci number.

- Let S_H be the minimum number of nodes in an AVL tree of height H can have.
- Then a minimum tree of height H will look like the following:

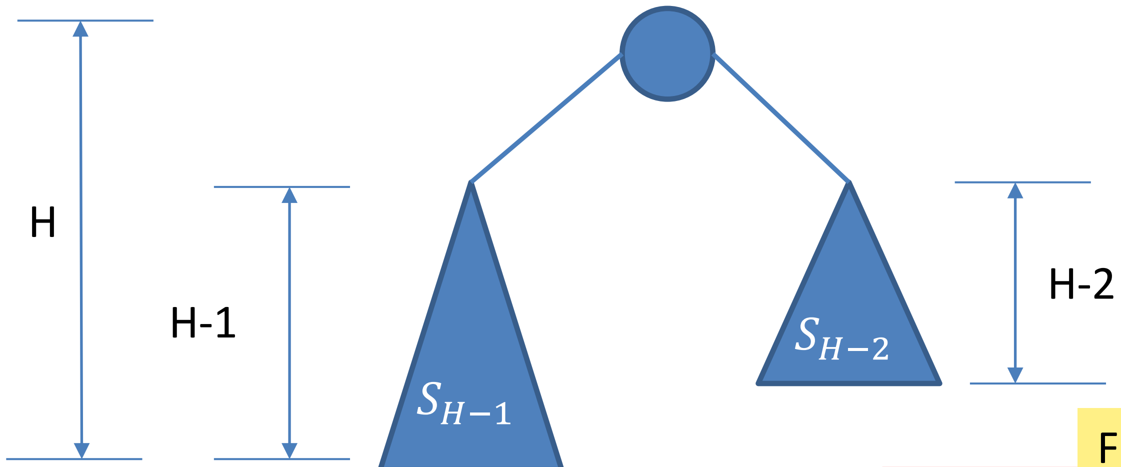


$$\begin{aligned} S_0 &= 1 \\ S_1 &= 2 \end{aligned}$$

$$S_H = S_{H-1} + S_{H-2} + 1$$

AVL Trees: Properties

Theorem: An AVL tree of height H has at least $F_{H+3} - 1$, where F_i is the i^{th} Fibonacci number.



$$S_0 = 1$$
$$S_1 = 2$$

$$S_H = S_{H-1} + S_{H-2} + 1$$

$$S_2 = S_1 + S_0 + 1 = 4 = F_5 - 1$$
$$S_3 = S_2 + S_1 + 1 = 7 = F_6 - 1$$
$$\ldots \ldots \ldots ..$$
$$S_H = S_{H-2} + S_{H-1} + 1 = F_{H+3} - 1$$

Fibonacci Numbers

$F_1 = 1$

$F_2 = 1$

$F_3 = F_1 + F_2 = 2$

$F_4 = F_2 + F_3 = 3$

$F_5 = F_3 + F_4 = 5$

$F_6 = F_4 + F_5 = 8$

$\ldots \ldots .$

$F_i = F_{i-2} + F_{i-1}$

AVL Trees: Properties

According to "Binet's formula" $F_i \approx \frac{\varphi^i}{\sqrt{5}}$, where φ is known as "Golden Ratio".

$$\varphi = \left(1+\sqrt{5}\right)/2 \approx 1.618$$

$$S_H = N \approx \varphi^{H+3}/\sqrt{5} = \frac{\varphi^3}{\sqrt{5}} * \varphi^H$$

$$\log(N+2) > \log\frac{\varphi^3}{\sqrt{5}} + \mathrm{H} * \log\varphi$$

$$\frac{\log(N+2)}{log\varphi} > \log\frac{\varphi^3}{\sqrt{5}} * \frac{1}{\log\varphi} + H$$ Multiply both sides of the equation by $\frac{1}{\log\varphi}$

$$\mathrm{H} < \frac{1}{\log\varphi} * \log(N+2) + \log\frac{\varphi^3}{\sqrt{5}} * \frac{1}{\log\varphi}$$

$$\boxed{\mathrm{H} < 1.44 * \log(N+2) - 1.328}$$

AVL Trees: Properties

The worst case height of an AVL tree is at most roughly 44% more than the minimum possible for binary trees.

$$\boxed{\mathrm{H} < 1.44 * \log(N+2) - 1.328}$$

AVL Trees: Insertion

- If a node is inserted to an AVL, then it may need to be rebalanced.
- Let the node to be rebalanced be X.
- Because any node has at most two children that the heights of X's two subtrees differ by 2, a violation might occur in any of the following cases:
- Case 1: An insertion in the left subtree of the left child of X
- Case 2: An insertion in the right subtree of the left child of X
- Case 3: An insertion in the left subtree of the right child of X
- Case 4: An insertion in the right subtree of the right child of X

AVL Trees: Insertion

- If a node is inserted to an AVL, then it may need to be rebalanced.
- Let the node to be rebalanced be X.
- Because any node has at most two children that the heights of X's two subtrees differ by 2, a violation might occur in any of the following cases:
- Case 1: An insertion in the left subtree of the left child of X
- Case 2: An insertion in the right subtree of the left child of X
- Case 3: An insertion in the left subtree of the right child of X
- Case 4: An insertion in the right subtree of the right child of X

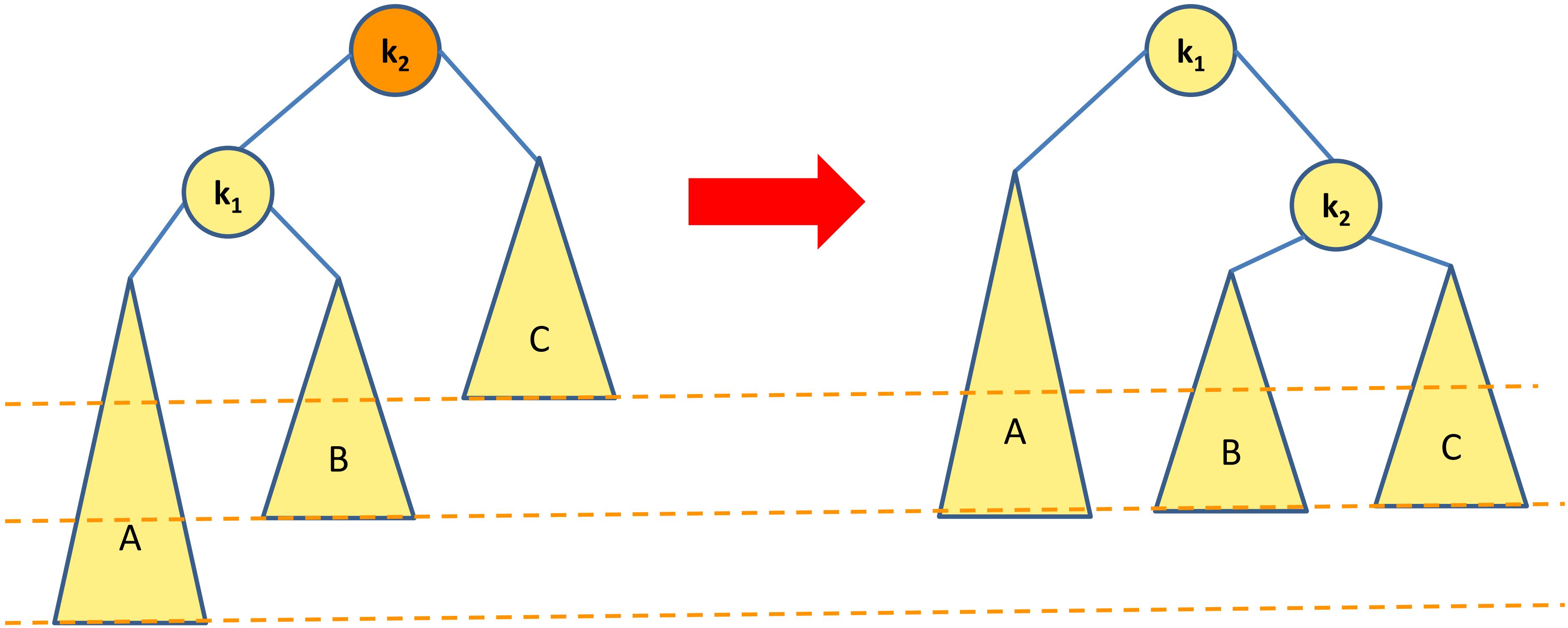
Case 1 and Case 4 are mirror image symmetries and both require **single rotation**.

AVL Trees: Insertion

- If a node is inserted to an AVL, then it may need to be rebalanced.
- Let the node to be rebalanced be X.
- Because any node has at most two children that the heights of X's two subtrees differ by 2, a violation might occur in any of the following cases:
- Case 1: An insertion in the left subtree of the left child of X
- Case 2: An insertion in the right subtree of the left child of X
- Case 3: An insertion in the left subtree of the right child of X
- Case 4: An insertion in the right subtree of the right child of X

Case 2 and Case 3 are mirror image symmetries and both require **double rotation**.

AVL Trees: Rebalancing w/ single rotation (Case 1)

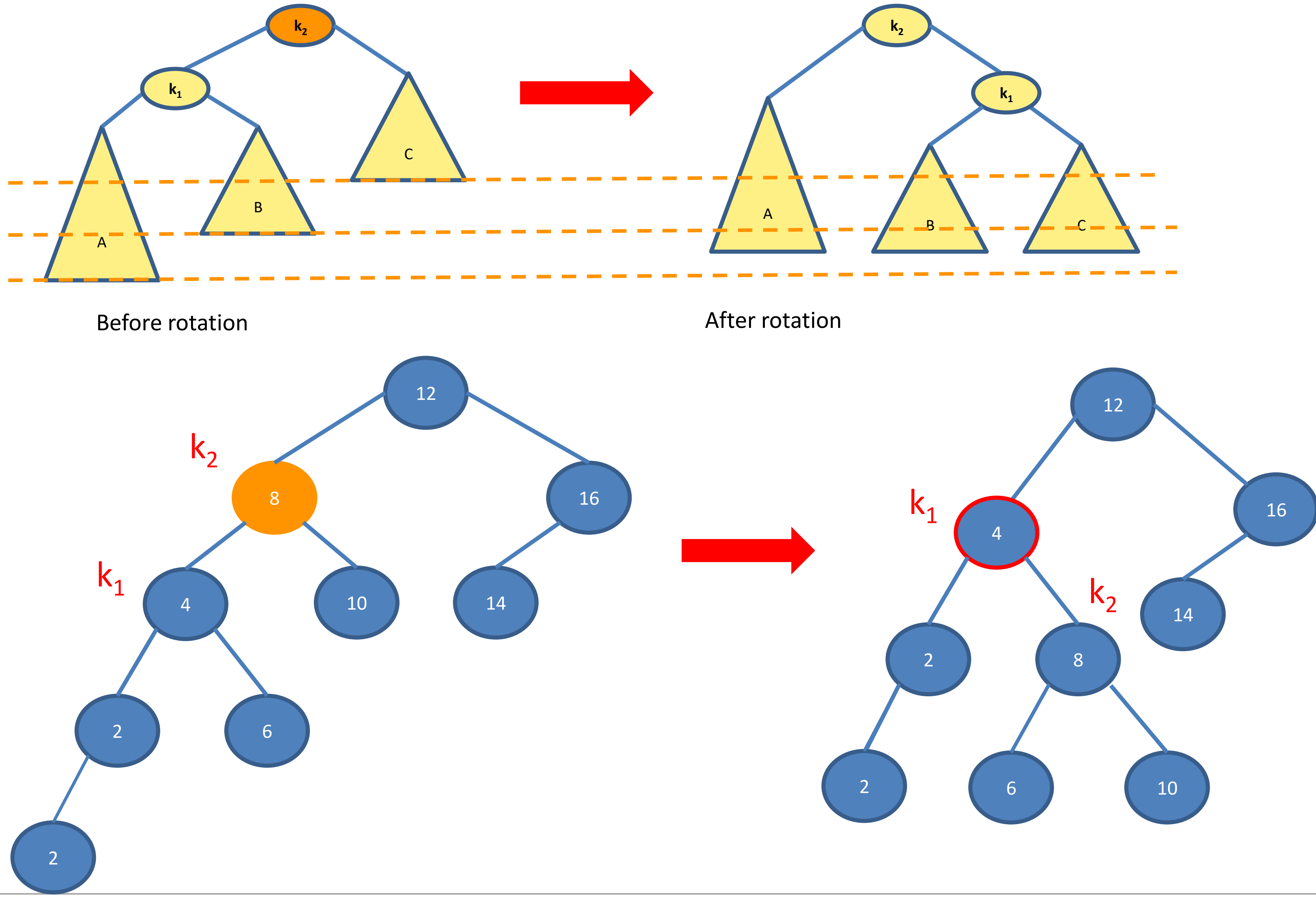


Before rotation

After rotation

Case 1: An insertion in the left subtree of the left child of X

AVL Trees: Rebalancing w/ single rotation (Case 1)



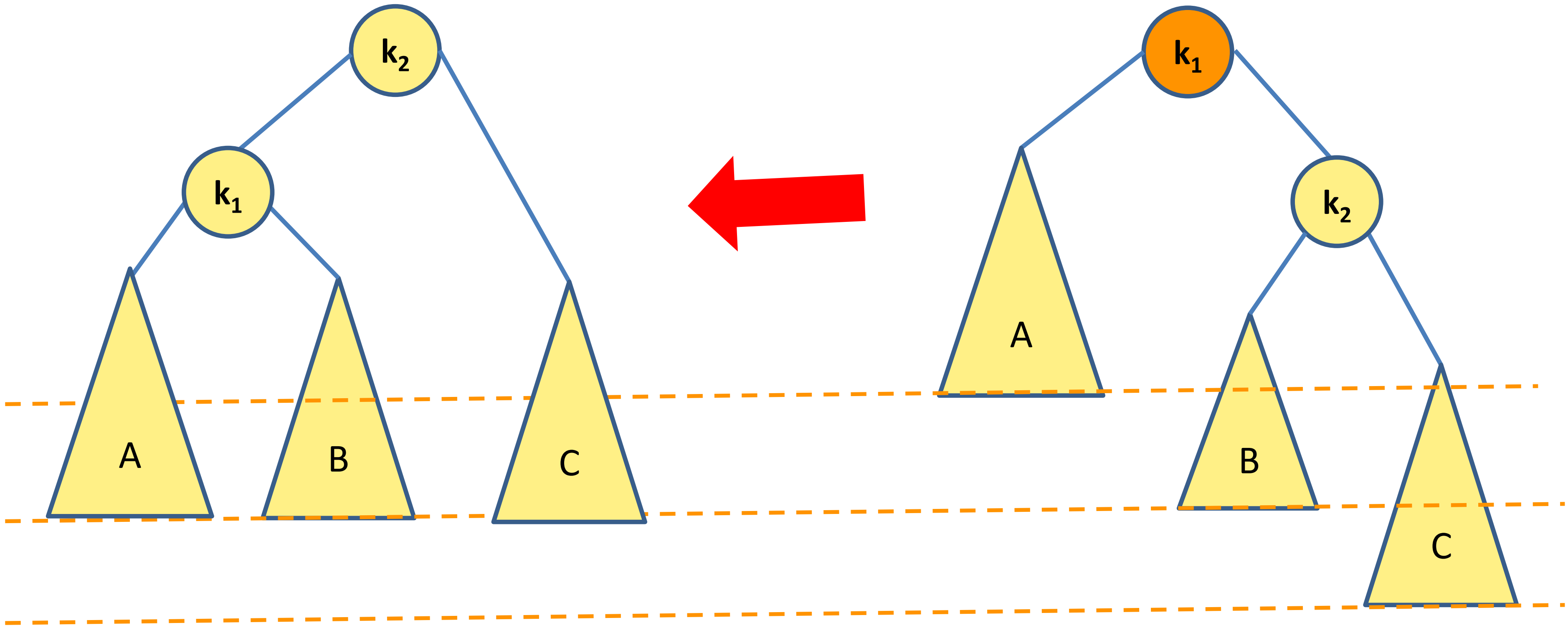
AVL Trees: Rebalancing w/ single rotation

- In-Class Exercise: Complete the following pseudocode for single rotation (Case 1)

```
/* Rotate binary tree node with left child
 * For all AVL trees this is single rotation for case 1 */
Static BinaryNode rotateWithLeftChild(BinaryNode k2) {
       …  …   …

}
```

AVL Trees: Rebalancing w/ single rotation (Case 4)

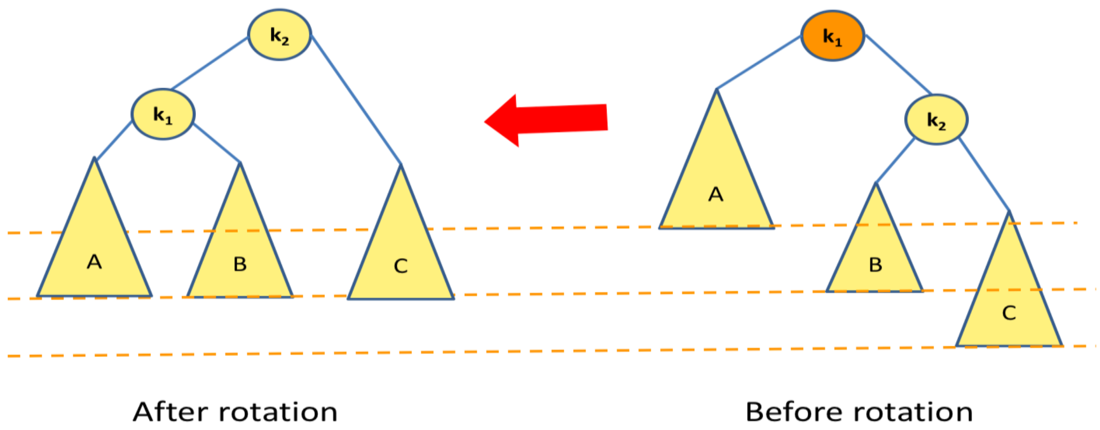


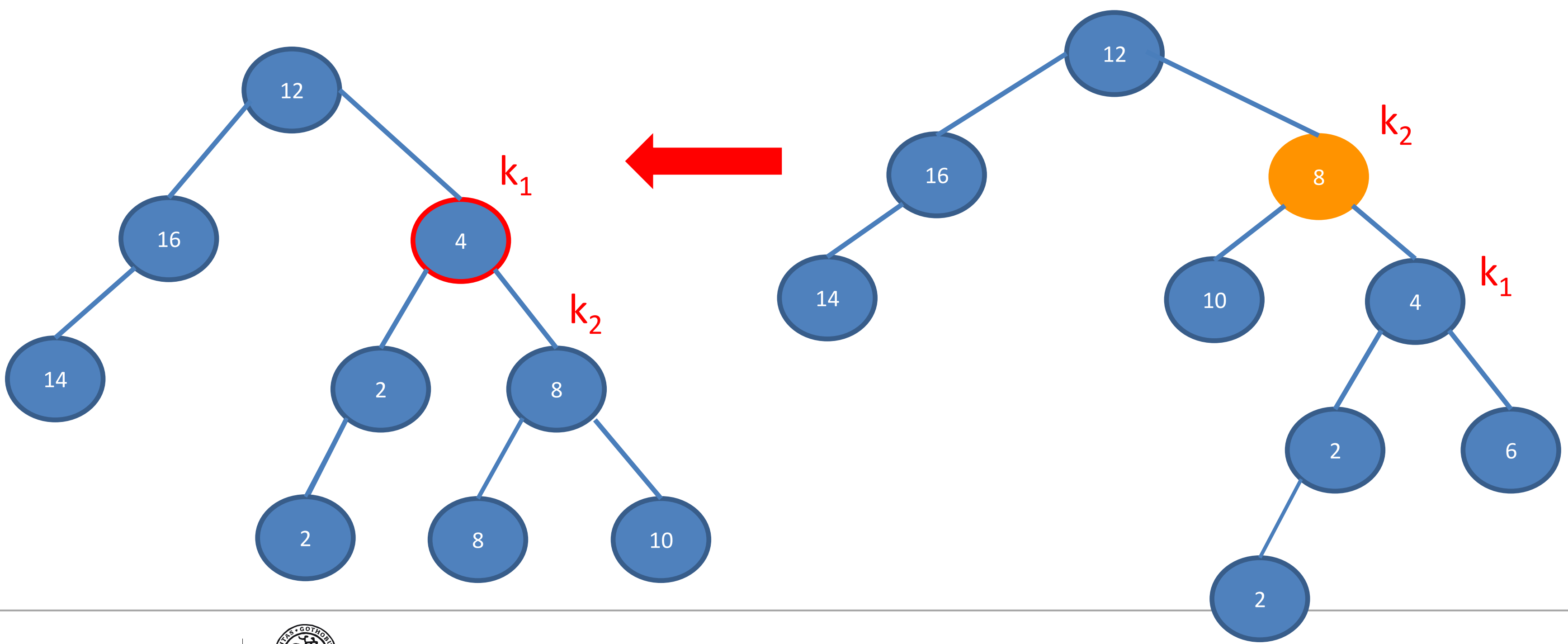
After rotation

Before rotation

Case 4: An insertion in the right subtree of the right child of X

AVL Trees: Rebalancing w/ single rotation (Case 4)





AVL Trees: Rebalancing w/ single rotation

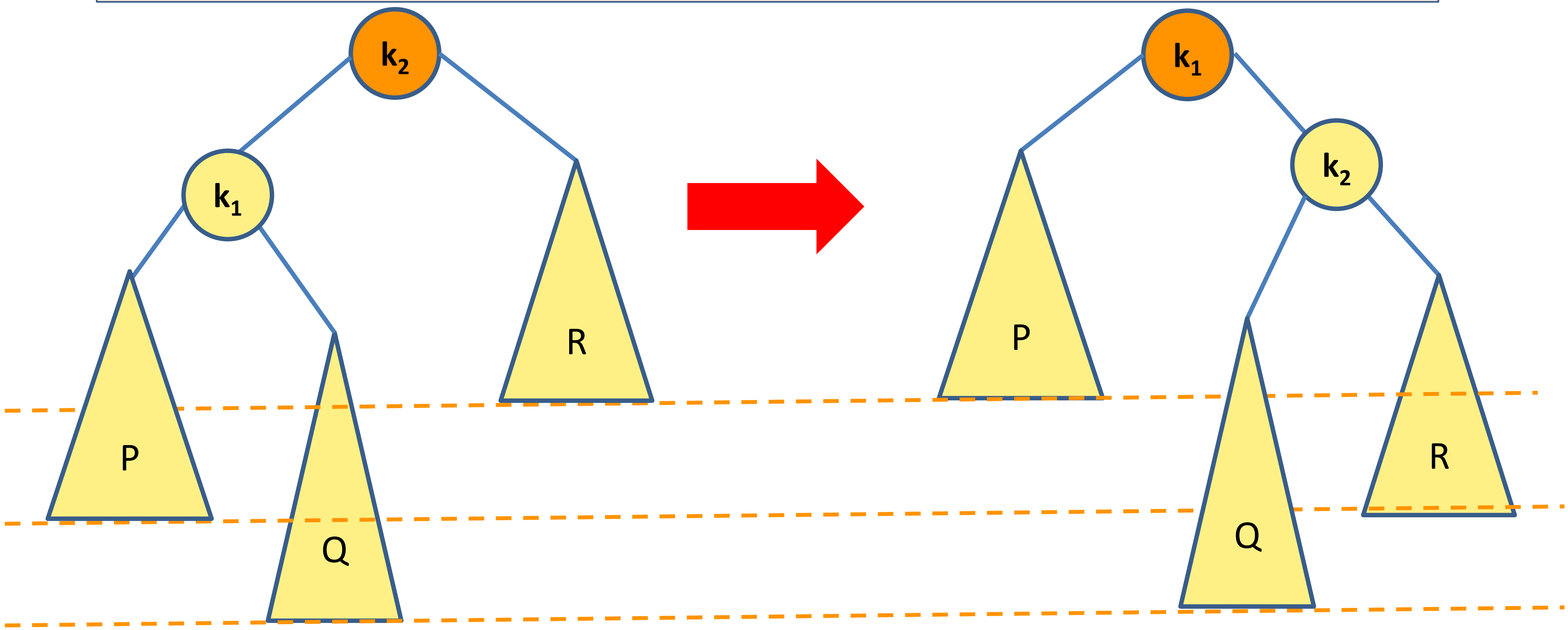
- In-Class Exercise: Complete the following pseudocode for single rotation (Case 4)

```
/* Rotate binary tree node with right child
 * For all AVL trees this is single rotation for case 1 */
Static BinaryNode rotateWithRightChild(BinaryNode k2) {
       …  …   …

}
```

AVL Trees: Rebalancing w/ single rotation (Case 2)

Case 2: An insertion in the right subtree of the left child of X

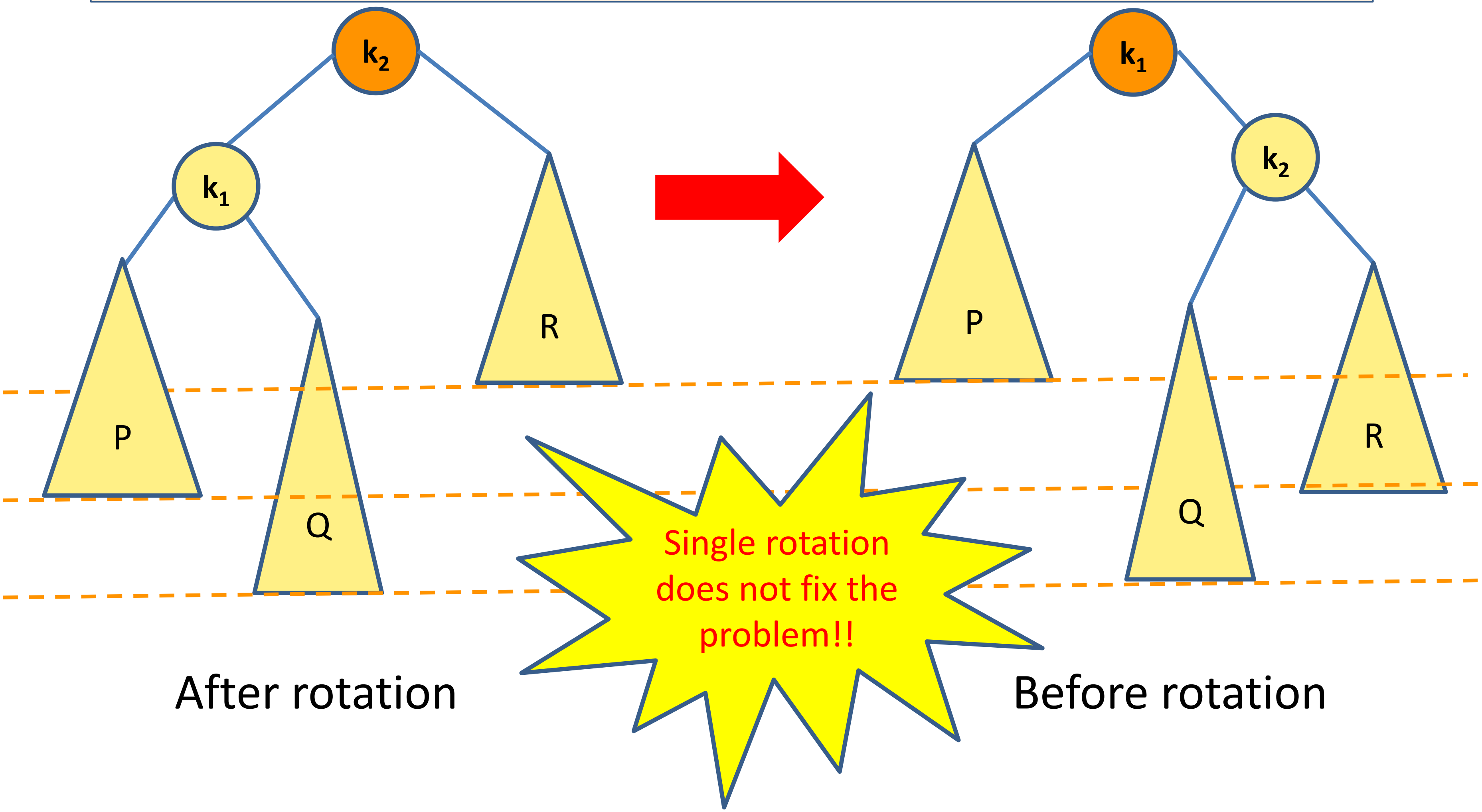


After rotation

Before rotation

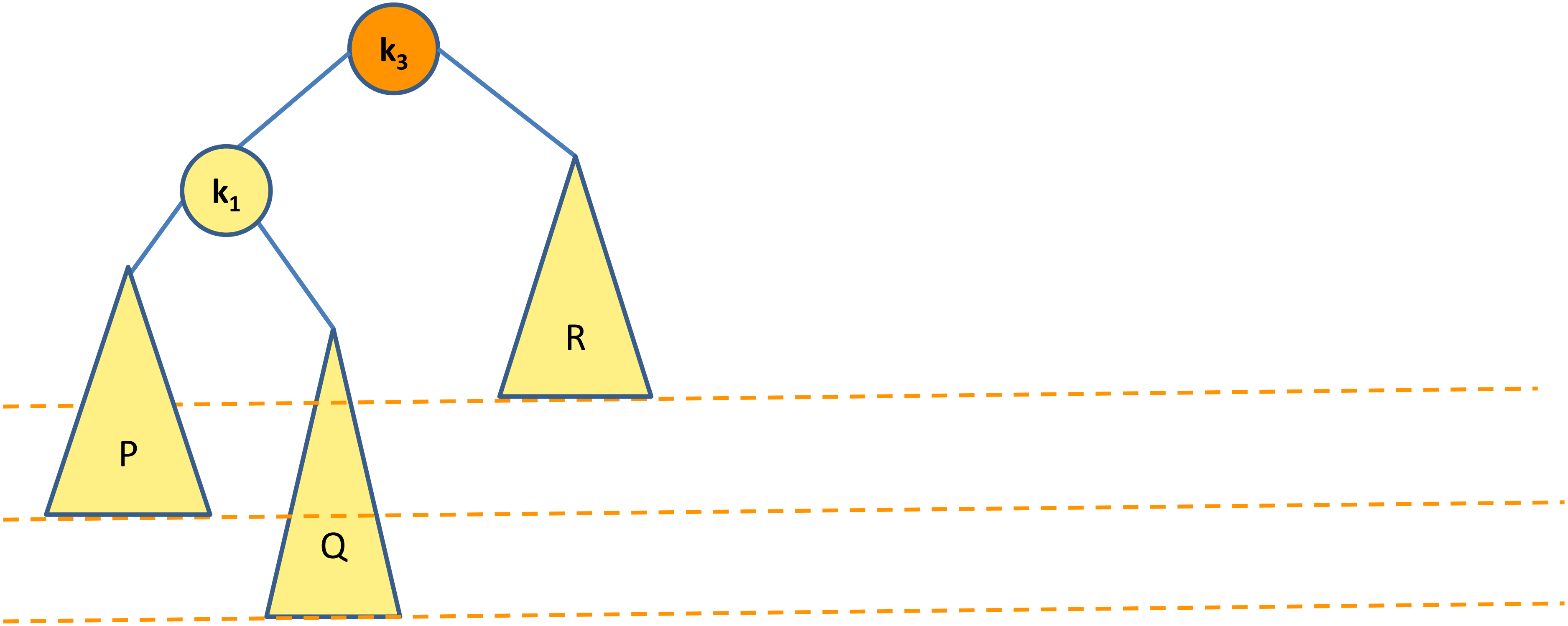
AVL Trees: Rebalancing w/ single rotation (Case 2)

Case 2: An insertion in the right subtree of the left child of X



AVL Trees: Rebalancing w/ single rotation (Case 2)

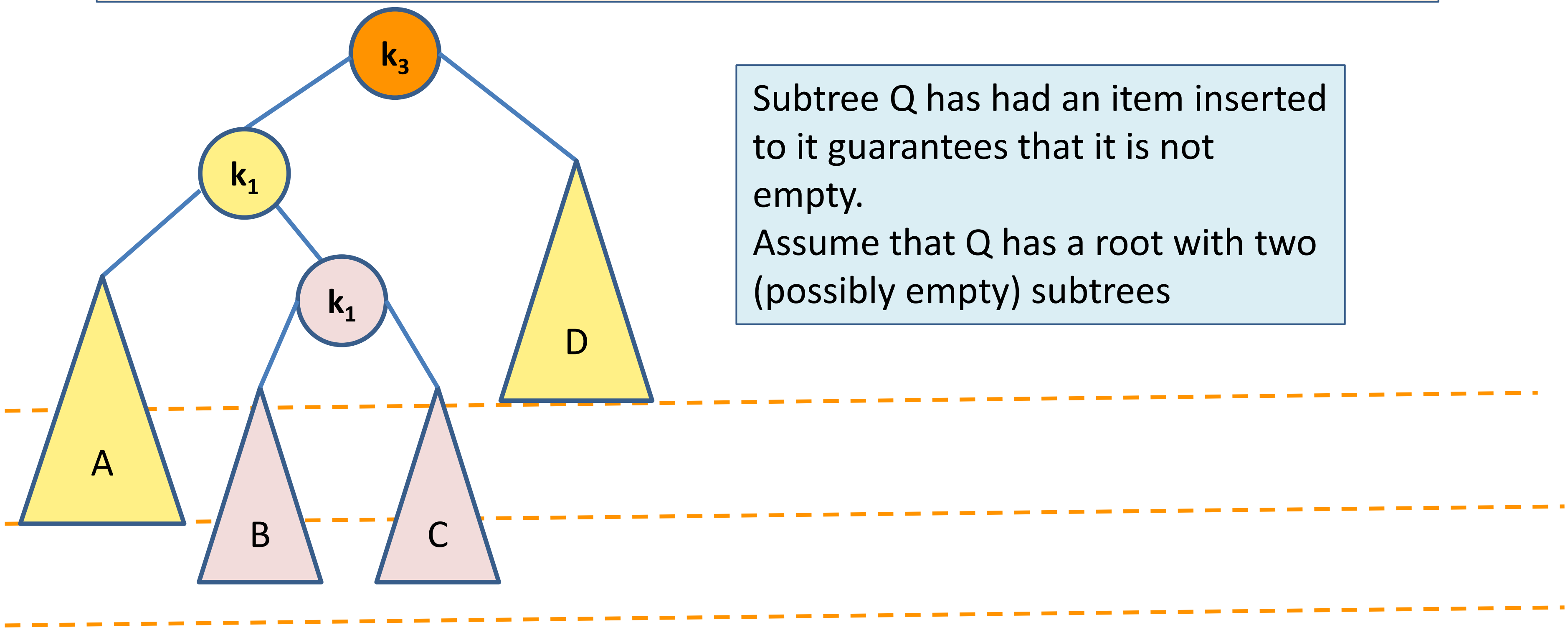
Case 2: An insertion in the right subtree of the left child of X



After rotation

AVL Trees: Rebalancing w/ single rotation (Case 2)

Case 2: An insertion in the right subtree of the left child of X

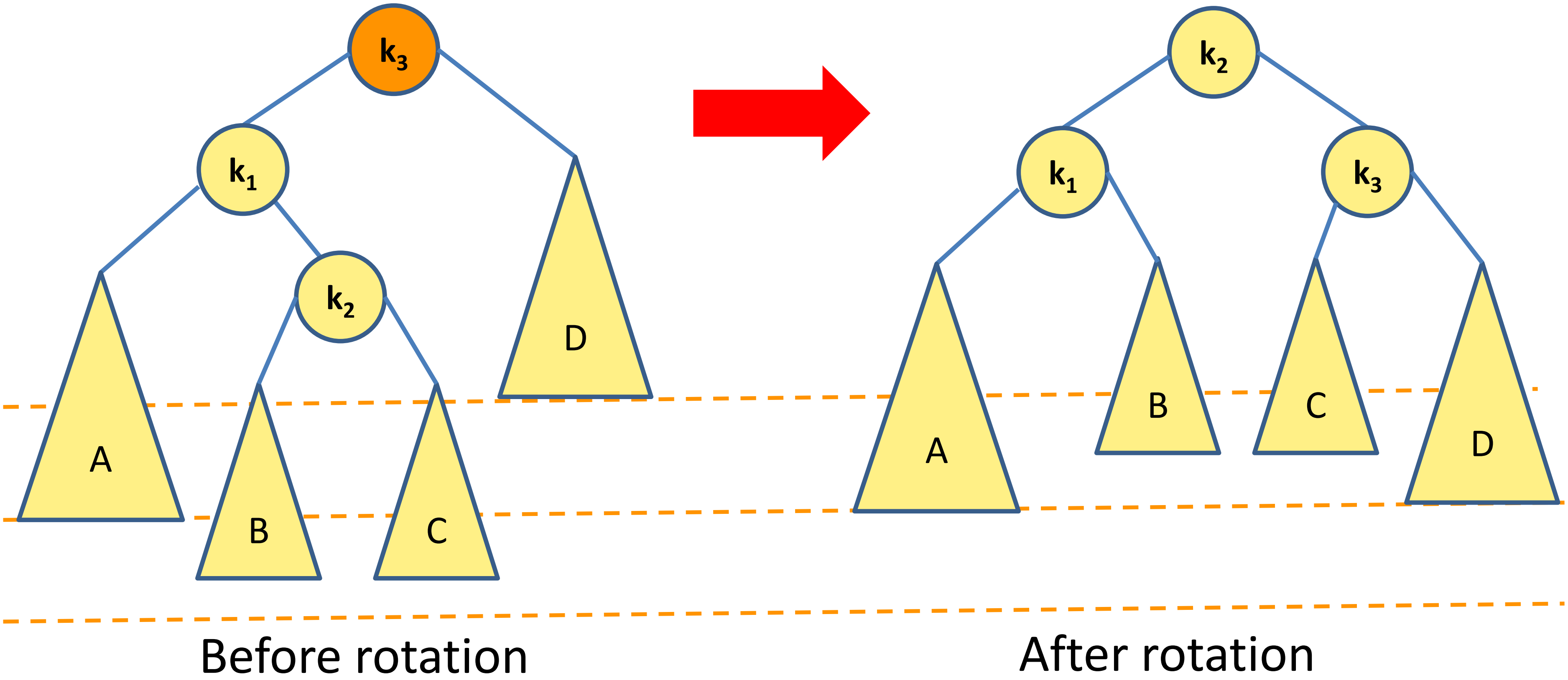


Subtree Q has had an item inserted to it guarantees that it is not empty.
Assume that Q has a root with two (possibly empty) subtrees

After rotation

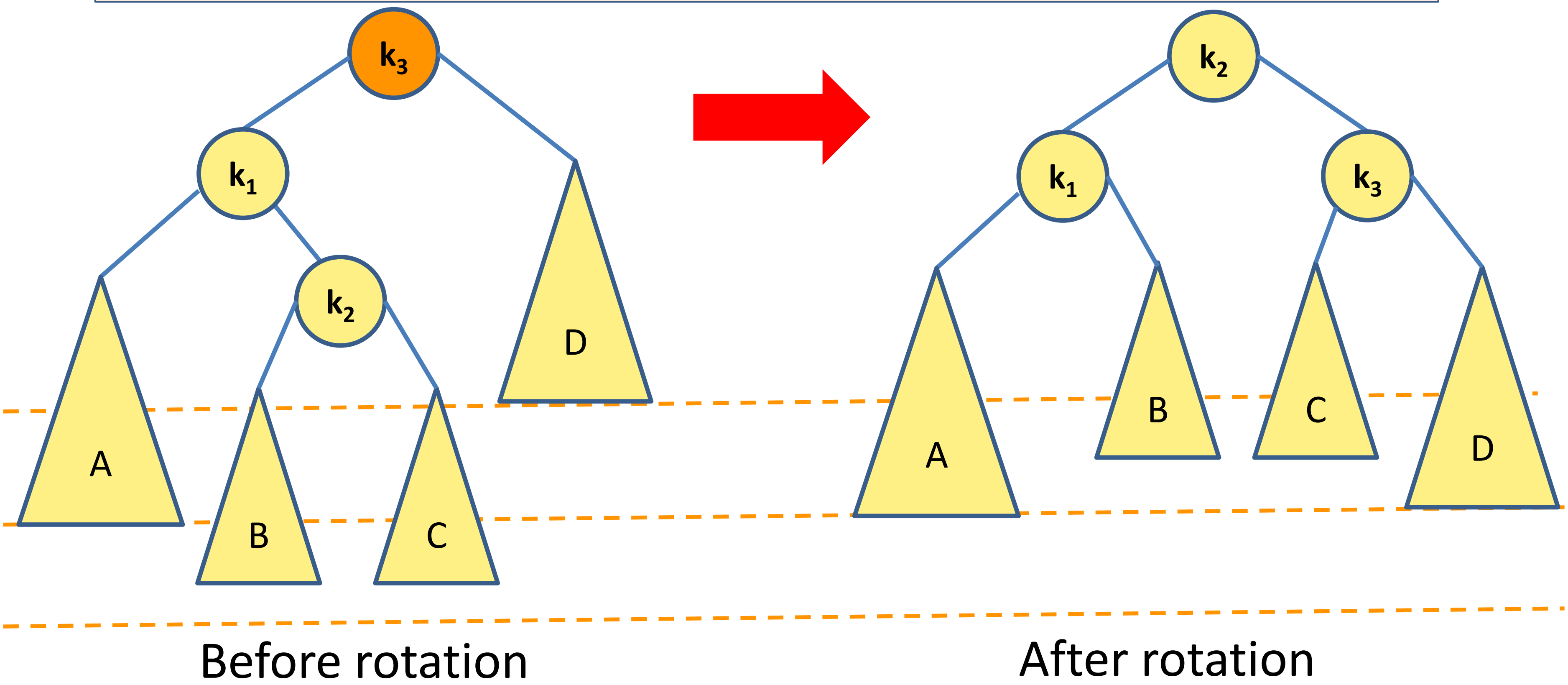
AVL Trees: Rebalancing w/ single rotation (Case 2)

Case 2: An insertion in the right subtree of the left child of X



AVL Trees: Rebalancing w/ single rotation (Case 2)

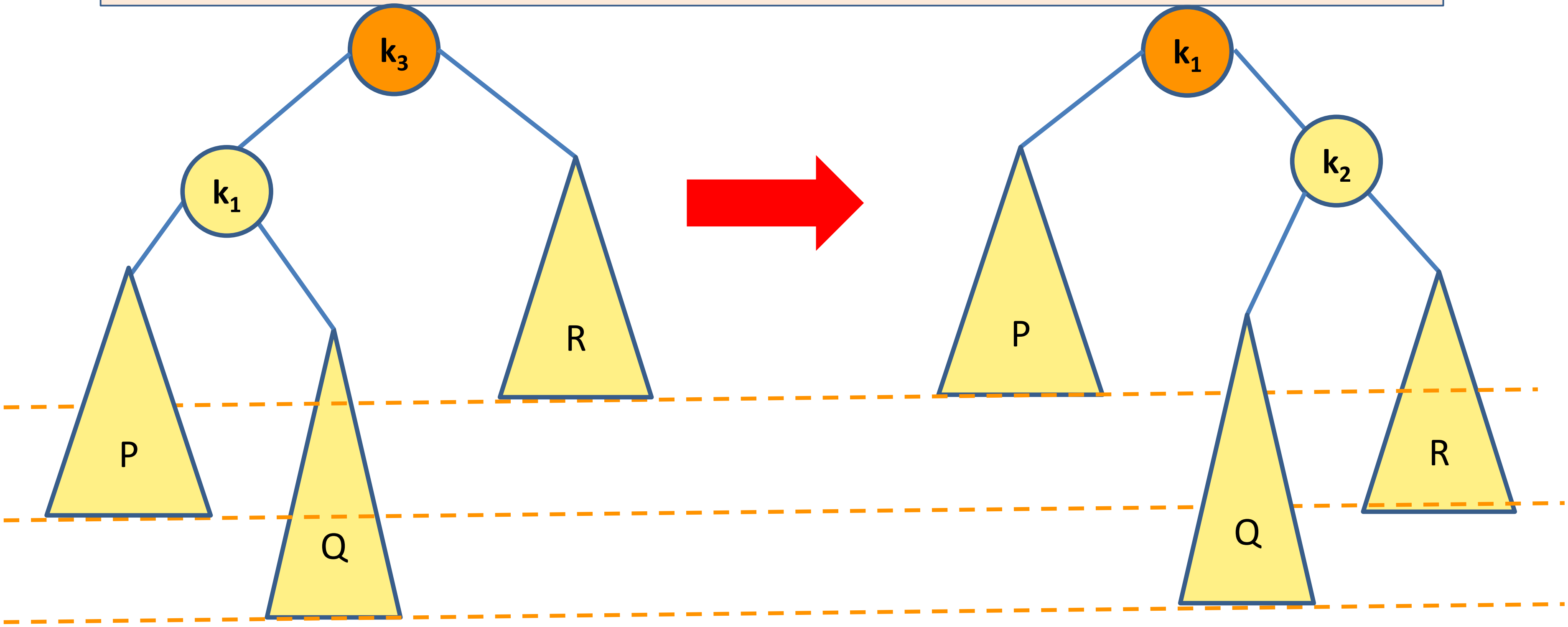
Case 2: An insertion in the right subtree of the left child of X



- **Left-Right double rotation** consists of:
- Single rotation between child of k_3 (i.e., k_1) and grandchild of k_3 (i.e., k_2)
- Single rotation between k_3 and its new child k_2

AVL Trees: Rebalancing w/ single rotation (Case 2)

Case 2: An insertion in the right subtree of the left child of X



After rotation

Before rotation

AVL Trees: Rebalancing w/ single rotation

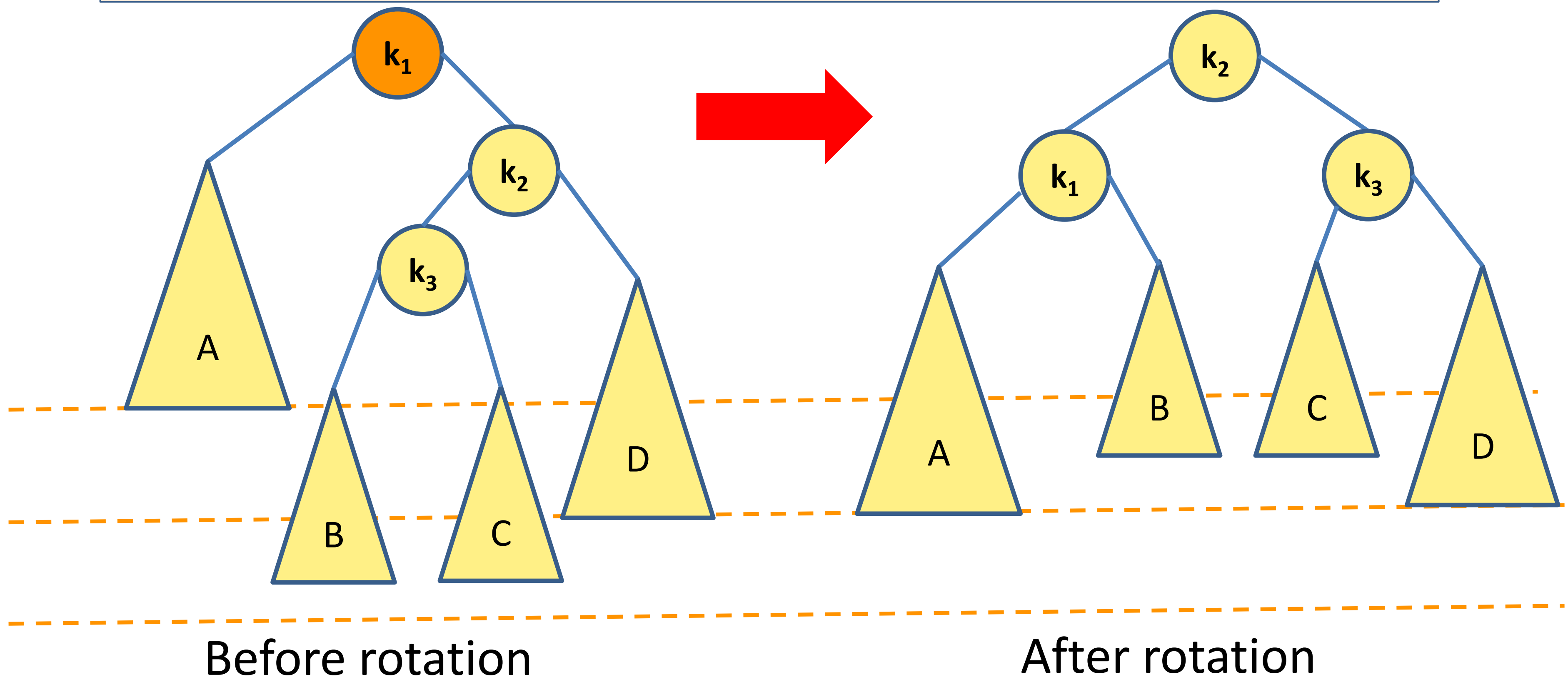
- In-Class Exercise: Complete the following pseudocode for left-right double rotation (Case 2)

```
/* Double Rotate binary tree node: first left child with its right child; then node k3
with its new left child. For all AVL trees this is left-right double rotation for case 2 */
Static BinaryNode doubleRotateWithLeftChild(BinaryNode k3) {
      …  …   …

}
```

AVL Trees: Rebalancing w/ single rotation (Case 2)

Case 3: An insertion in the left subtree of the right child of X



AVL Trees: Rebalancing w/ single rotation

- In-Class Exercise: Complete the following pseudocode for right-left double rotation (Case 3)

```
/* Double Rotate binary tree node: first right child with its left child; then node k1
with its new right child. For all AVL trees this is right-left double rotation for case 3 */

Static BinaryNode doubleRotateWithRightChild(BinaryNode k1) {
    …  …   …

}
```