Lecture – 14
AVL Trees

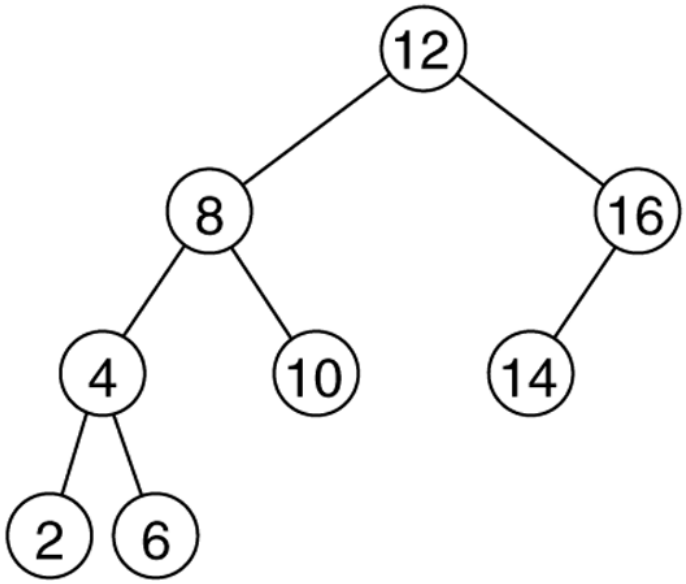
AVL Trees

- An AVL tree is a binary search tree with a *balance* condition.
- AVL is named for its inventors: **A**del'son-**V**el'skii and **L**andis
- AVL tree *approximates* the ideal tree (completely balanced tree).
- AVL Tree maintains a height close to the minimum.

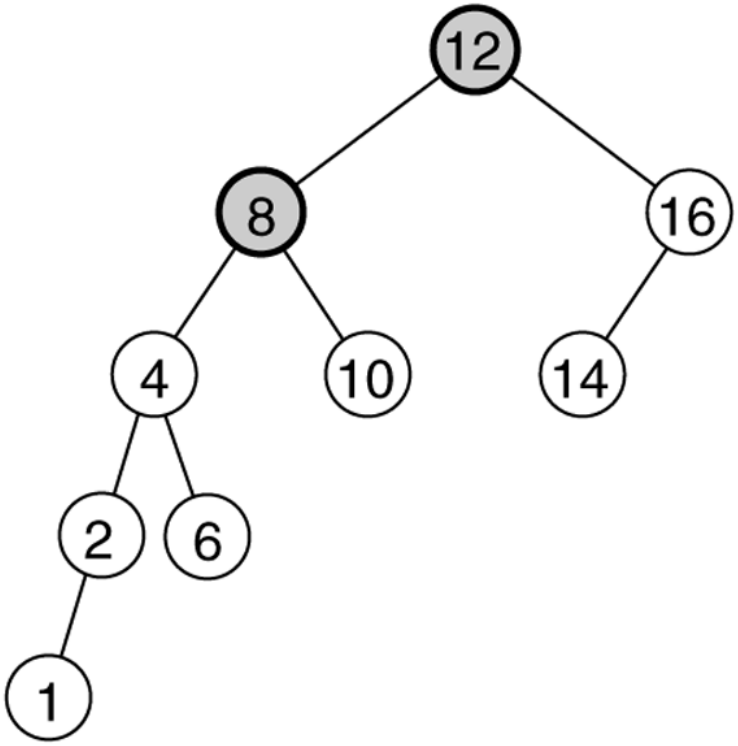
Definition:

An AVL tree is a binary search tree such that
for any node in the tree, the height of the left and
right subtrees can differ by at most 1.

Two binary search trees: (a) an AVL tree; (b) not an AVL tree (unbalanced nodes are darkened)

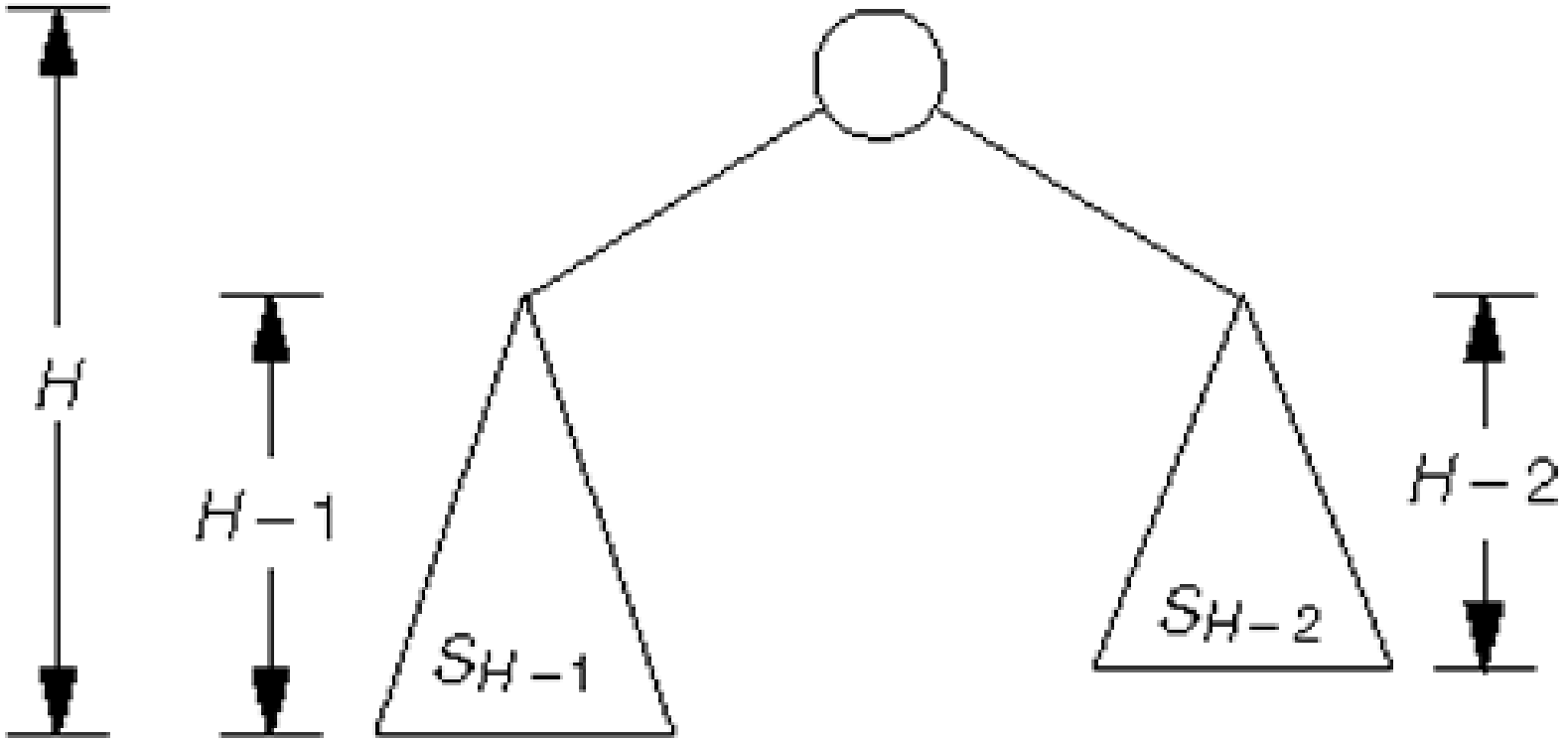


(a)



(b)

Minimum tree of height H



Properties

- The depth of a typical node in an AVL tree is very close to the optimal *log N*.
- Consequently, all searching operations in an AVL tree have logarithmic worst-case bounds.
- An update (insert or remove) in an AVL tree could destroy the balance. It must then be rebalanced before the operation can be considered complete.
- After an insertion, only nodes that are on the path from the insertion point to the root can have their balances altered.

Rebalancing

- Suppose the node to be rebalanced is X. There are 4 cases that we might have to fix (two are the mirror images of the other two):
  1. An insertion in the left subtree of the left child of X,
  2. An insertion in the right subtree of the left child of X,
  3. An insertion in the left subtree of the right child of X, or
  4. An insertion in the right subtree of the right child of X.
- Balancing of the tree can be restored by tree *rotations*.

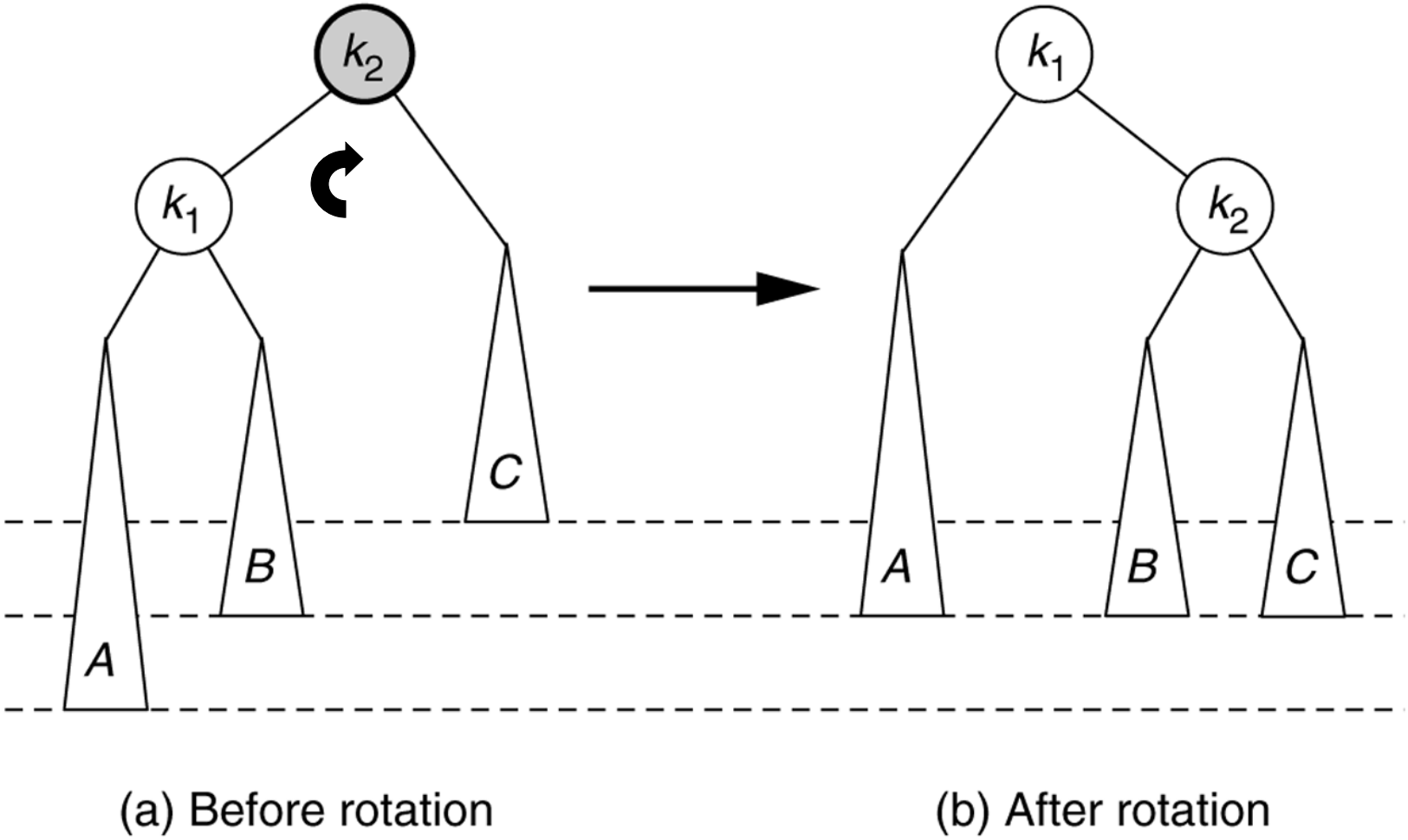
Balancing Operations: Rotations

- Case 1 and case 4 are symmetric and requires the same operation for balance.
  - Cases 1,4 are handled by *single rotation*.
- Case 2 and case 3 are symmetric and requires the same operation for balance.
  - Cases 2,3 are handled by *double rotation*.

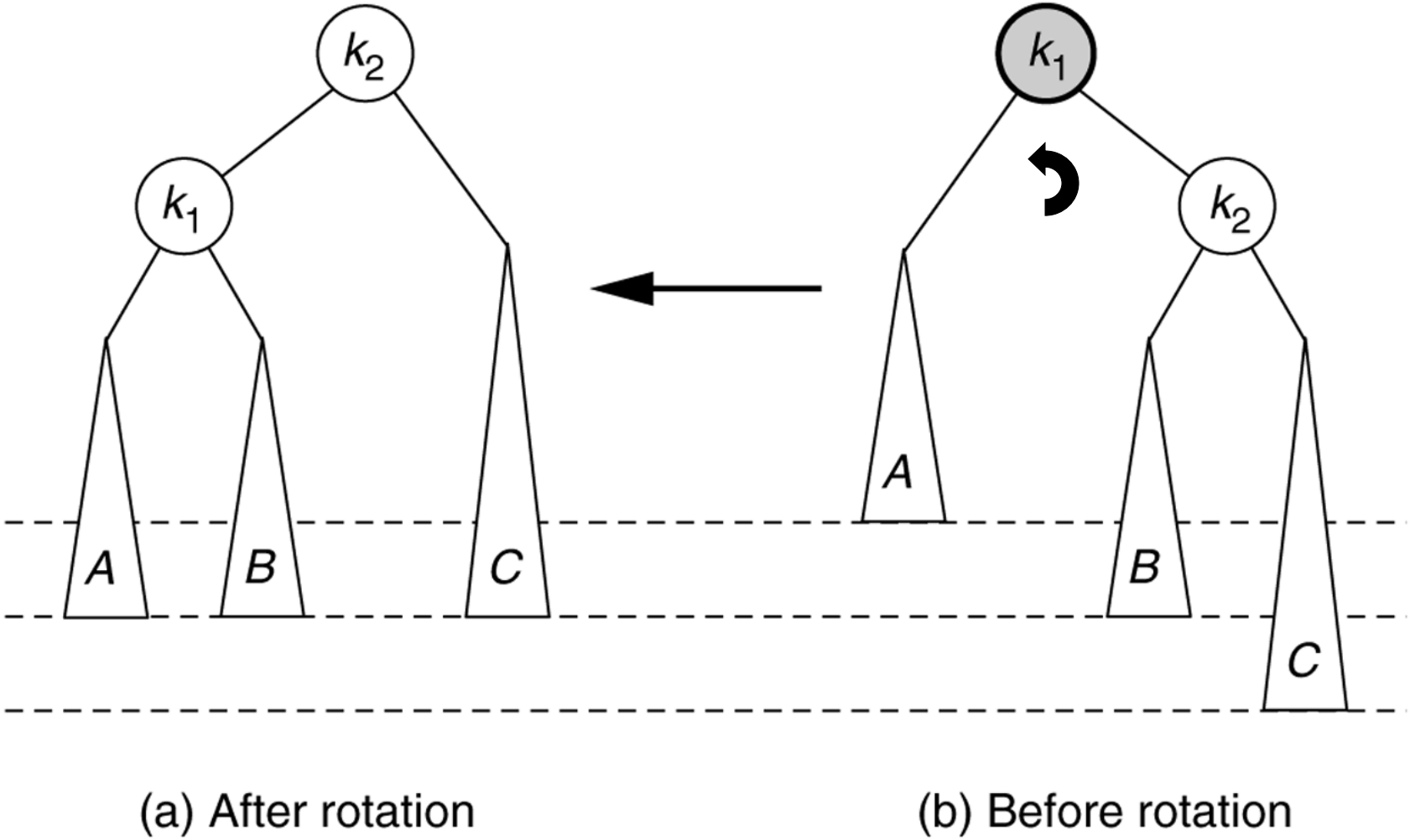
Single Rotation

- A single rotation switches the roles of the parent and child while maintaining the search order.
- Single rotation handles the outside cases (i.e. 1 and 4).
- We rotate between a node and its child.
  - Child becomes parent. Parent becomes right child in case 1, left child in case 4.
- The result is a binary search tree that satisfies the AVL property.

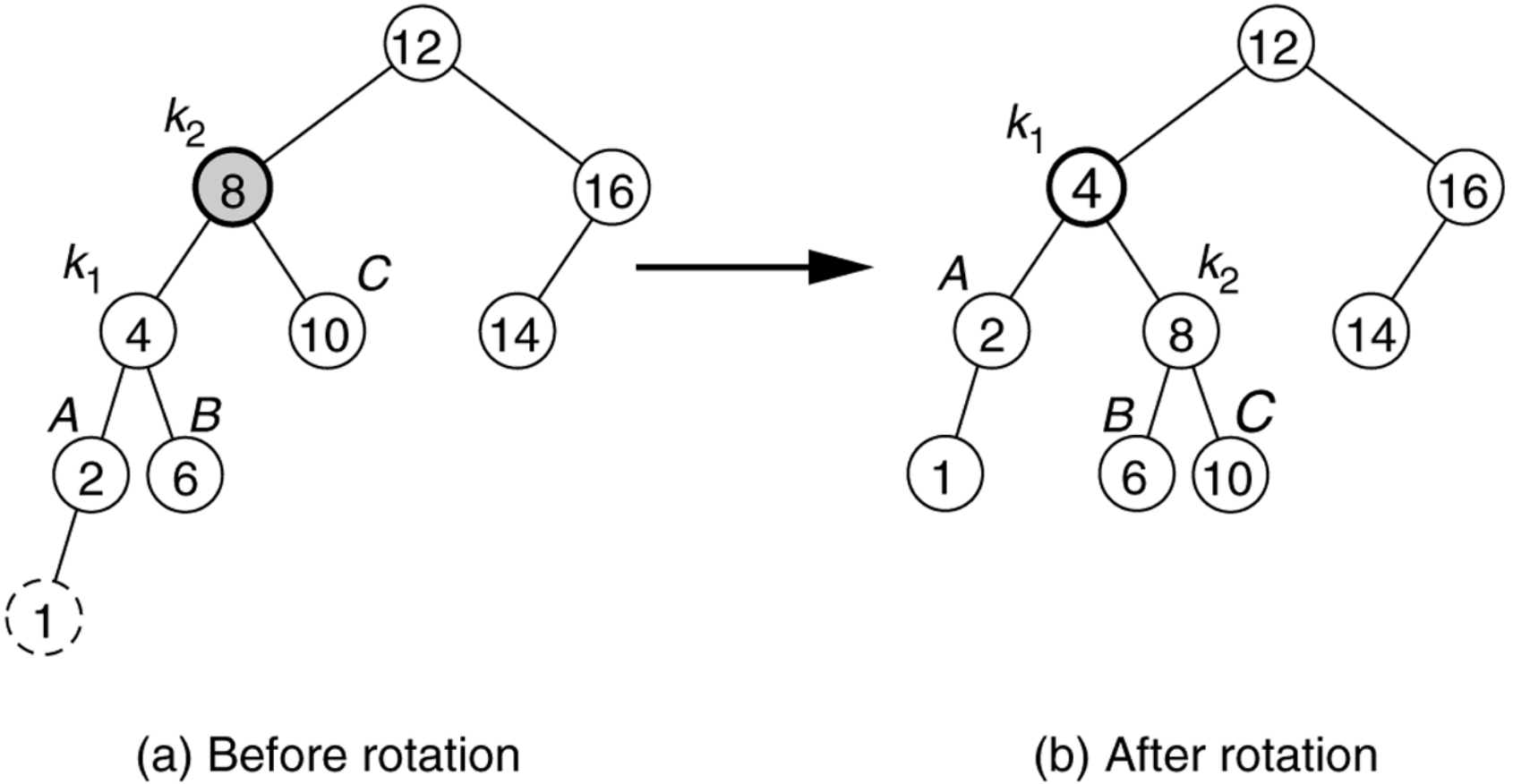
Single rotation to fix case 1: Rotate right



Symmetric single rotation to fix case 4 : Rotate left



Single rotation fixes an AVL tree after insertion of 1.

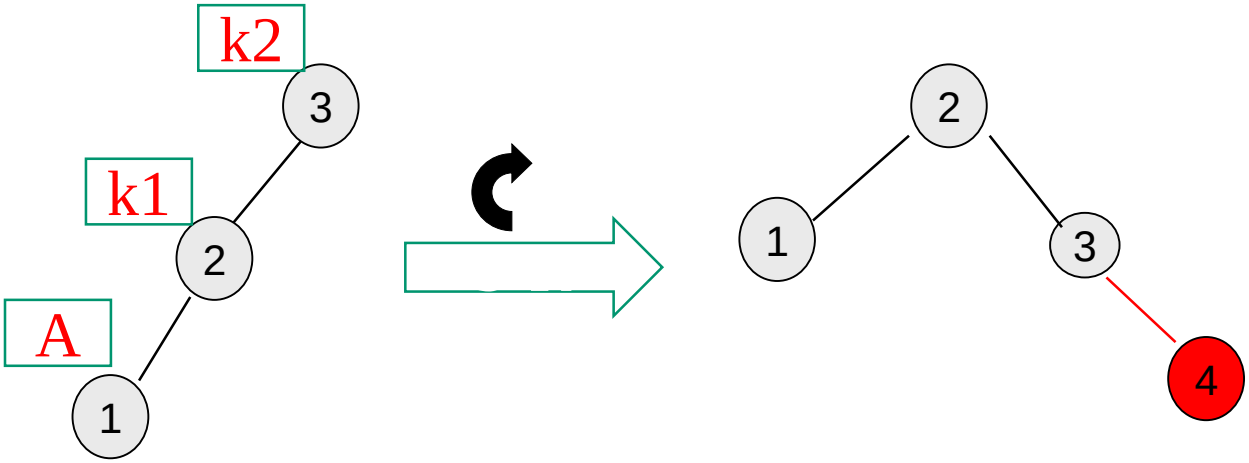


(a) Before rotation

(b) After rotation

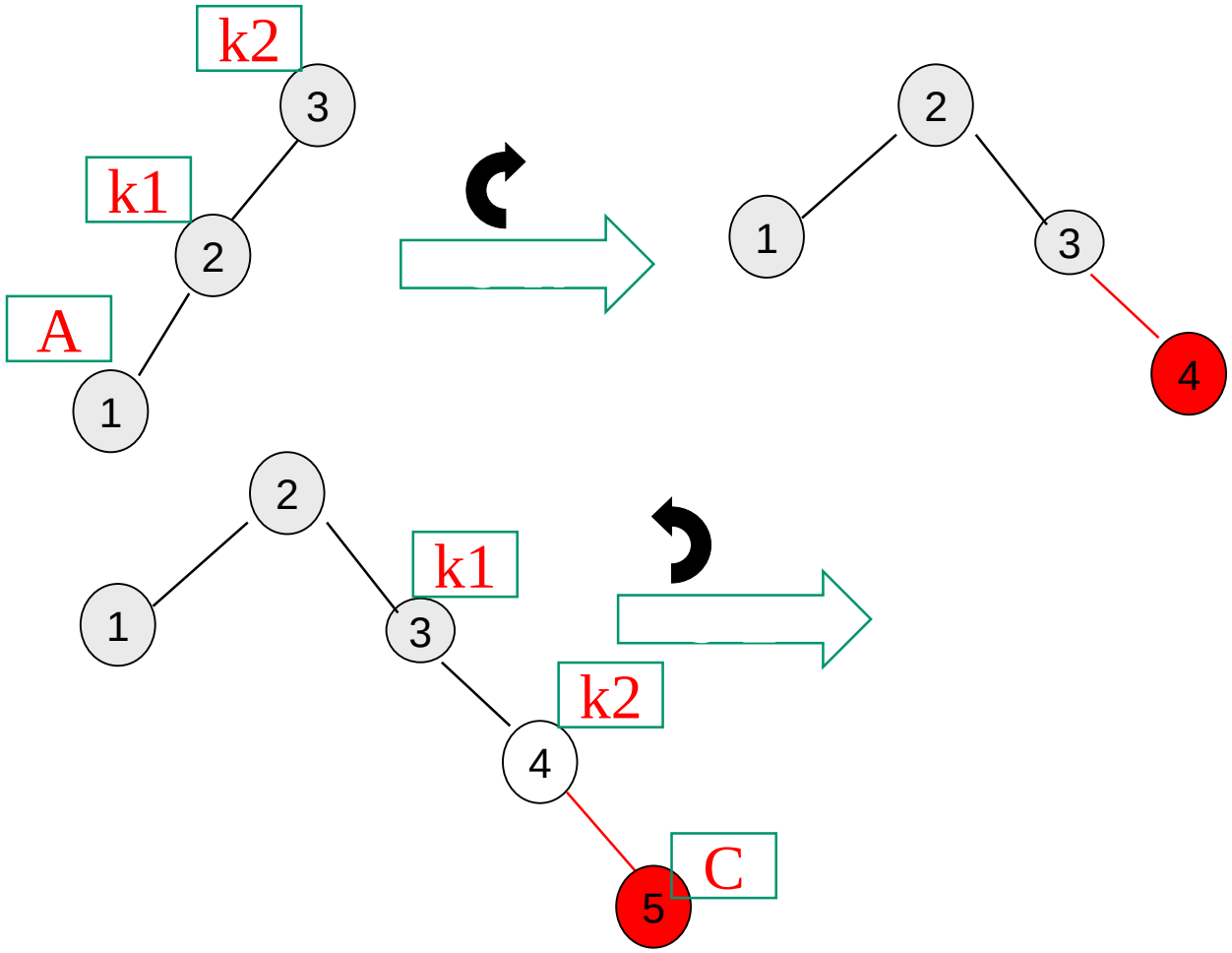
Example

Start with an empty AVL tree and insert the items 3,2,1, and then 4 through 7 in sequential order.



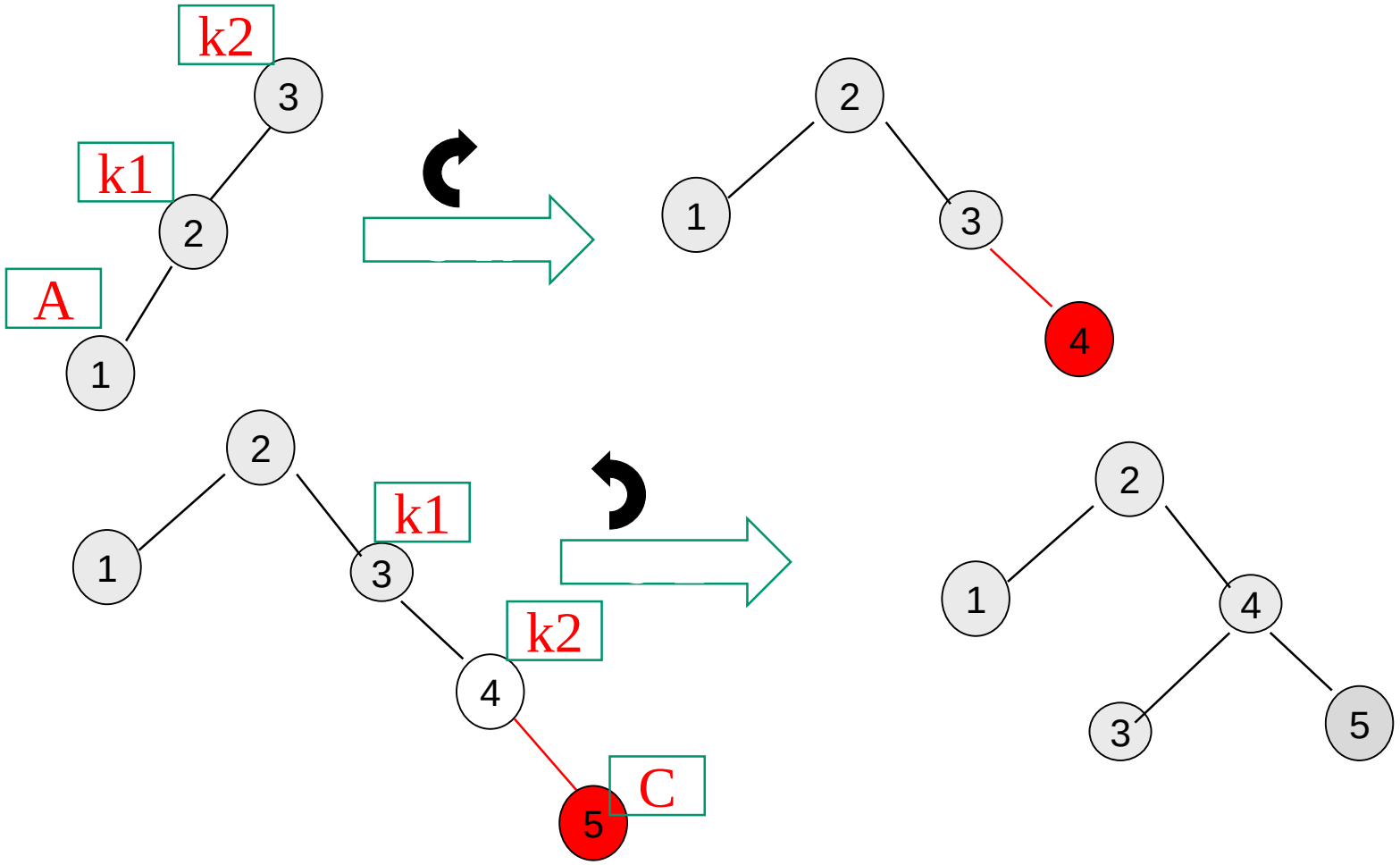
Example

Start with an empty AVL tree and insert the items 3,2,1, and then 4 through 7 in sequential order.

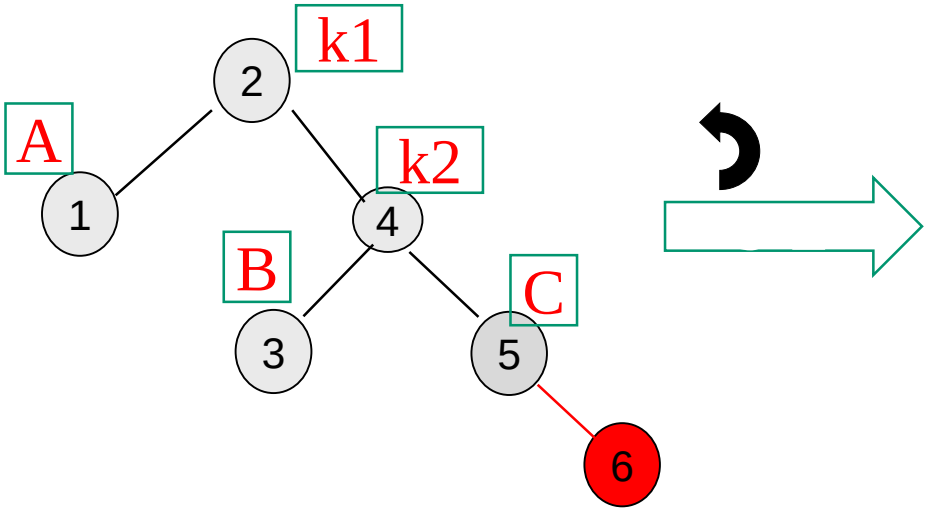


Example

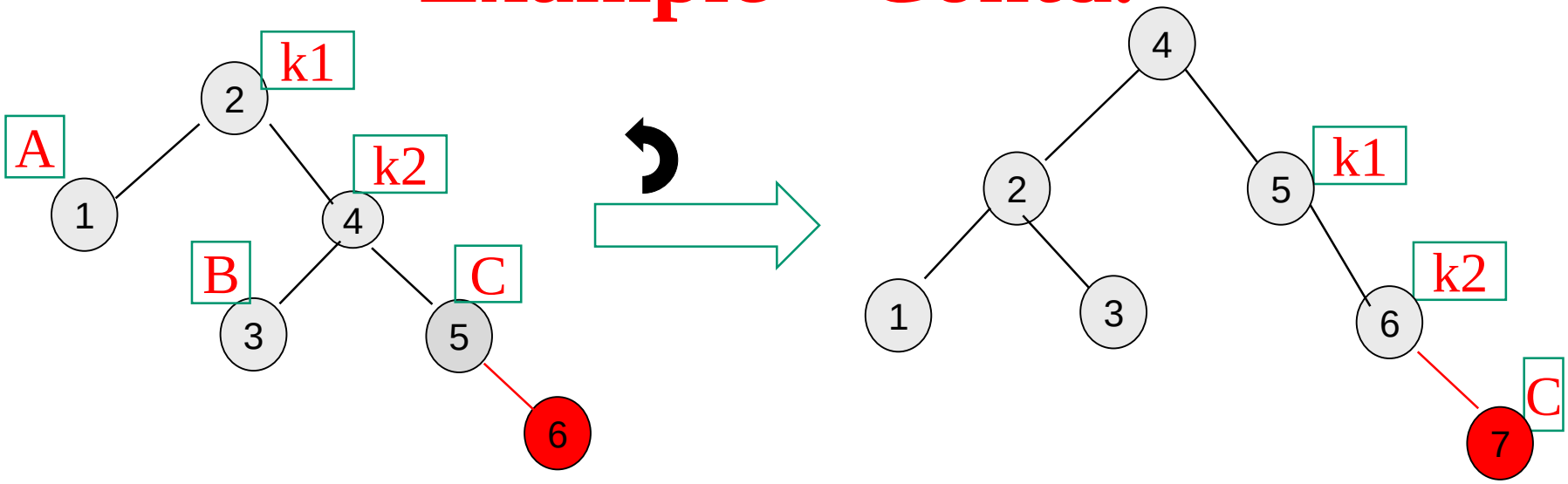
Start with an empty AVL tree and insert the items 3,2,1, and then 4 through 7 in sequential order.



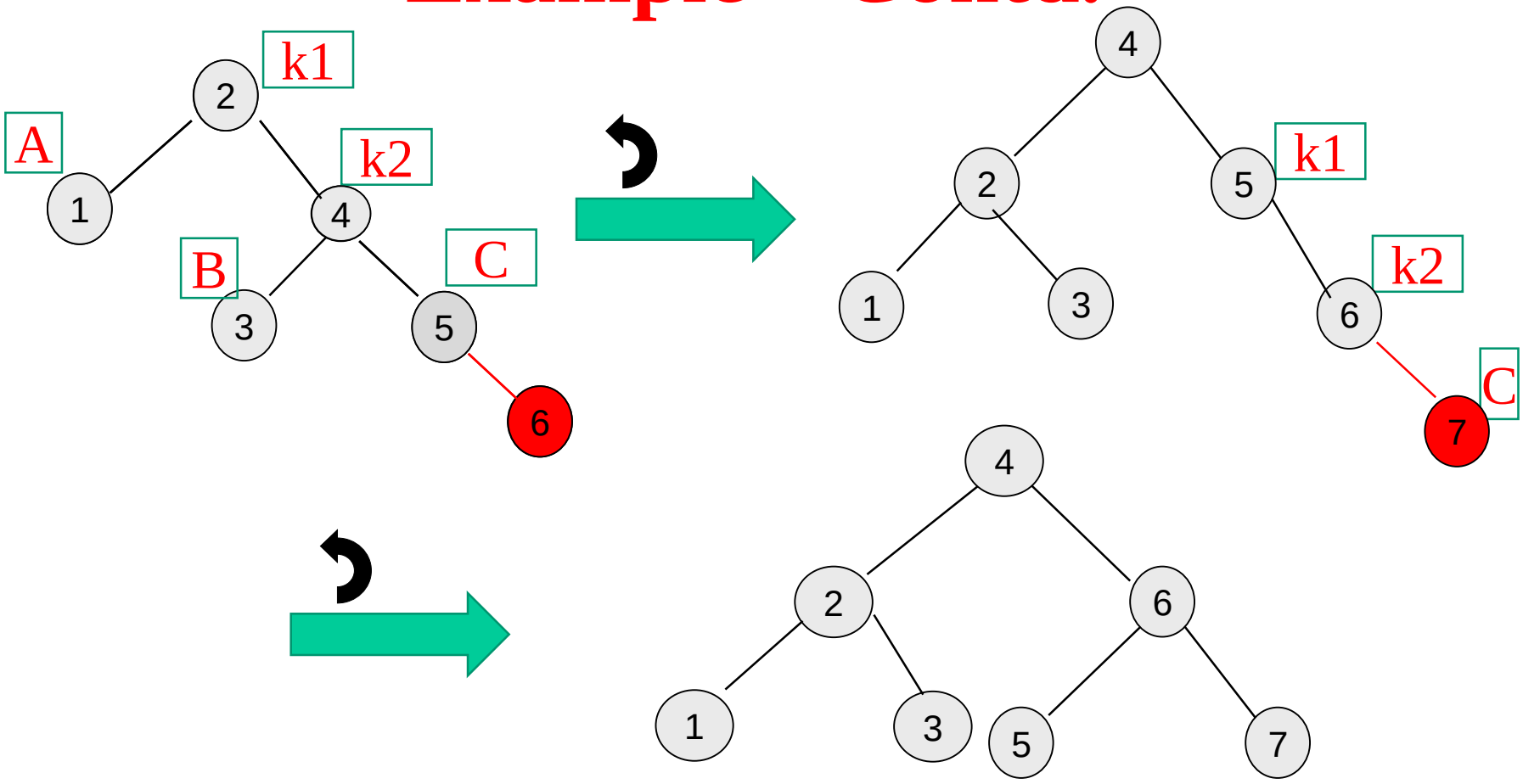
Example – Contd.



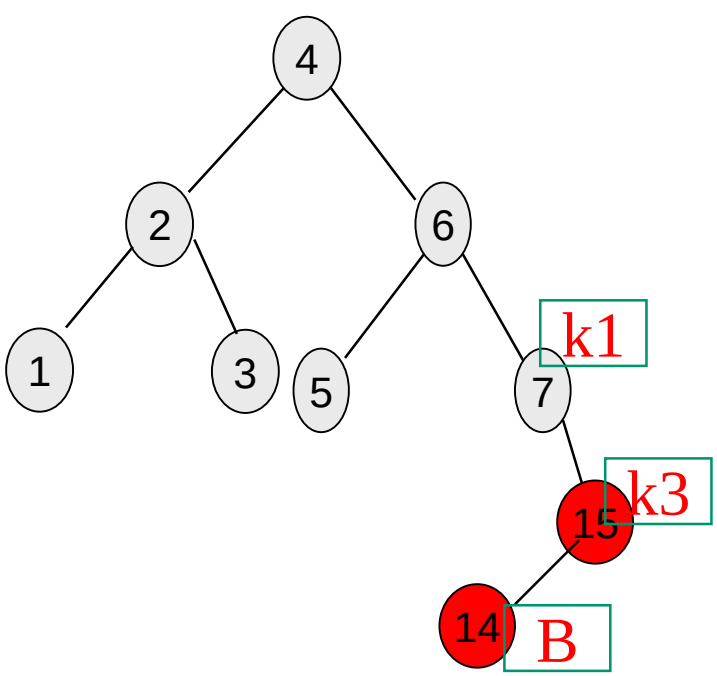
Example – Contd.

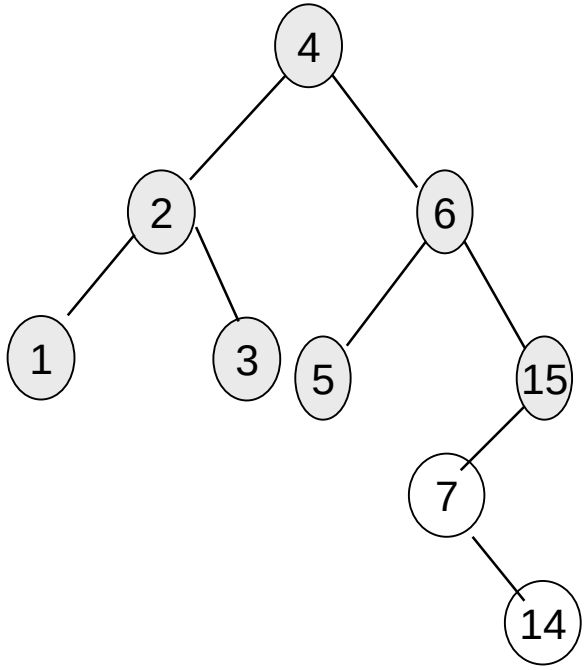


Example – Contd.



Insert 15, 14



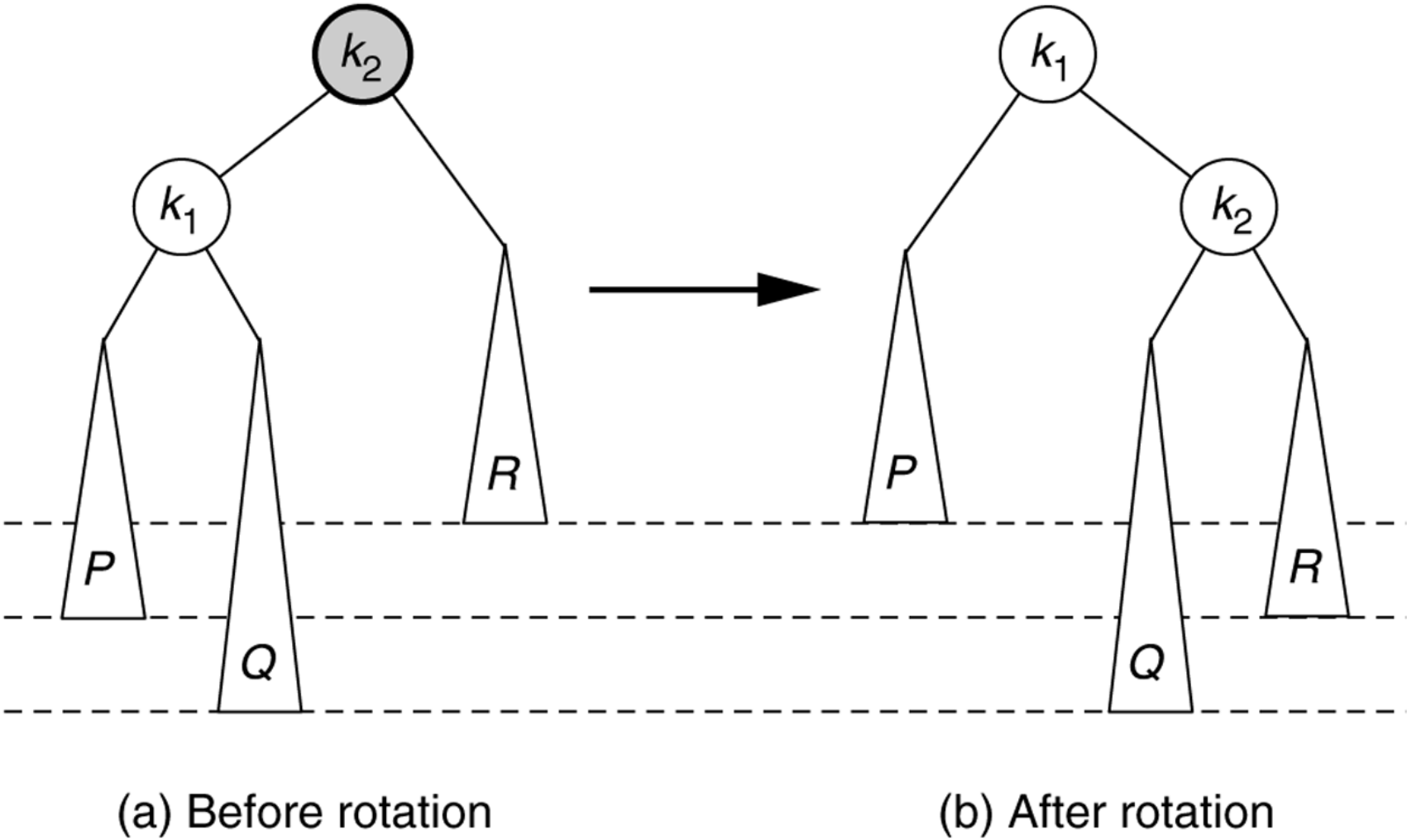


Single rotation cannot fix the height balance – Insertion of element is between trouble node and its child.

Double Rotation

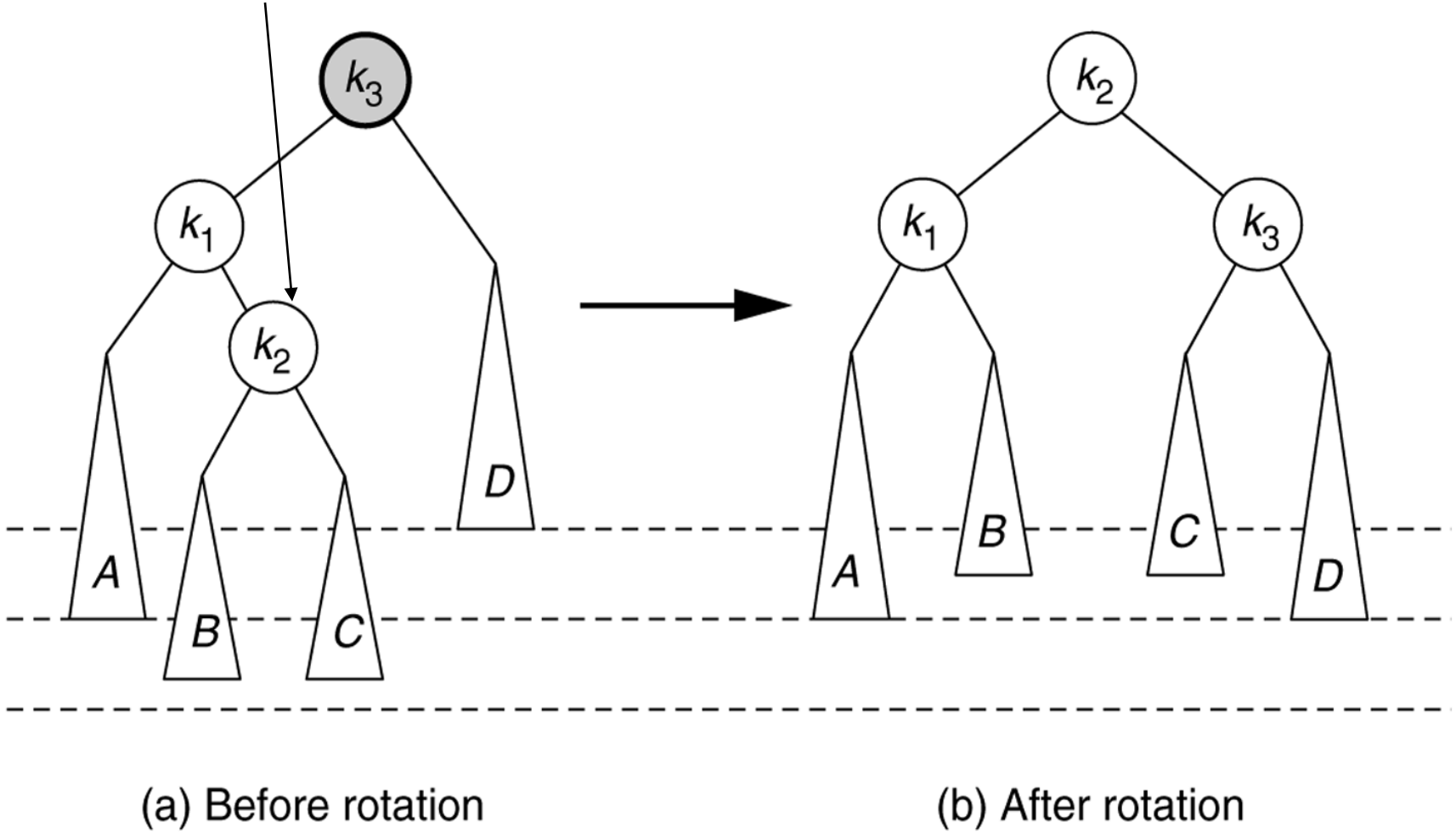
- Single rotation does not fix the inside cases (2 and 3).
- These cases require a *double* rotation, involving three nodes and four subtrees.

Single rotation **does not** fix case 2.



Left–right double rotation to fix case 2

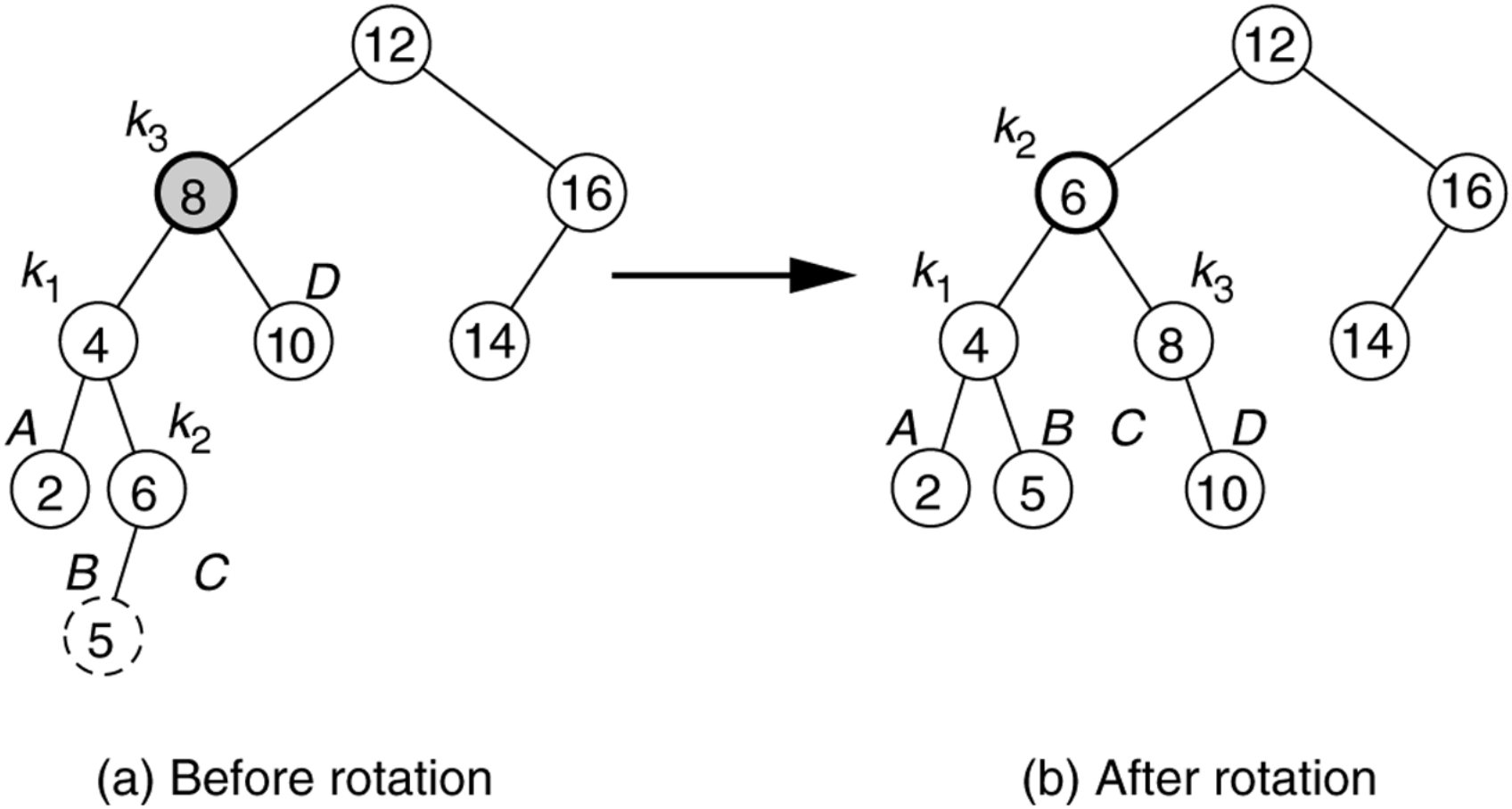
Lift this up:
first rotate left between (k_1,k_2),
then rotate right betwen (k_3,k_2)



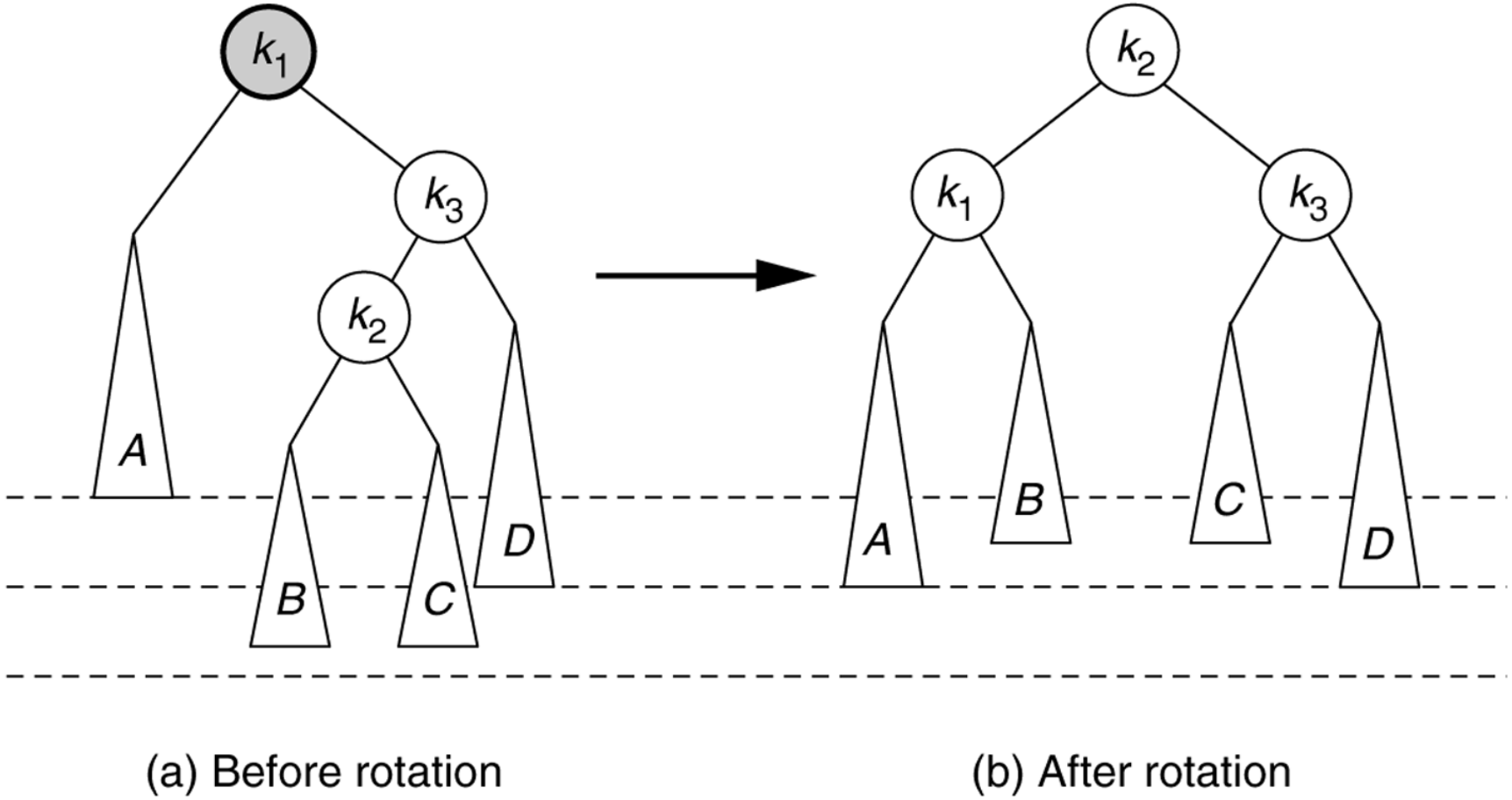
Left-Right Double Rotation

- A left-right double rotation is equivalent to a sequence of two single rotations:
  - 1st rotation on the original tree:
    a *left* rotation between X's left-child and grandchild
  - 2nd rotation on the new tree:
    a *right* rotation between X and its new left child.

Double rotation fixes AVL tree after the insertion of 5.



Right–Left double rotation to fix case 3.

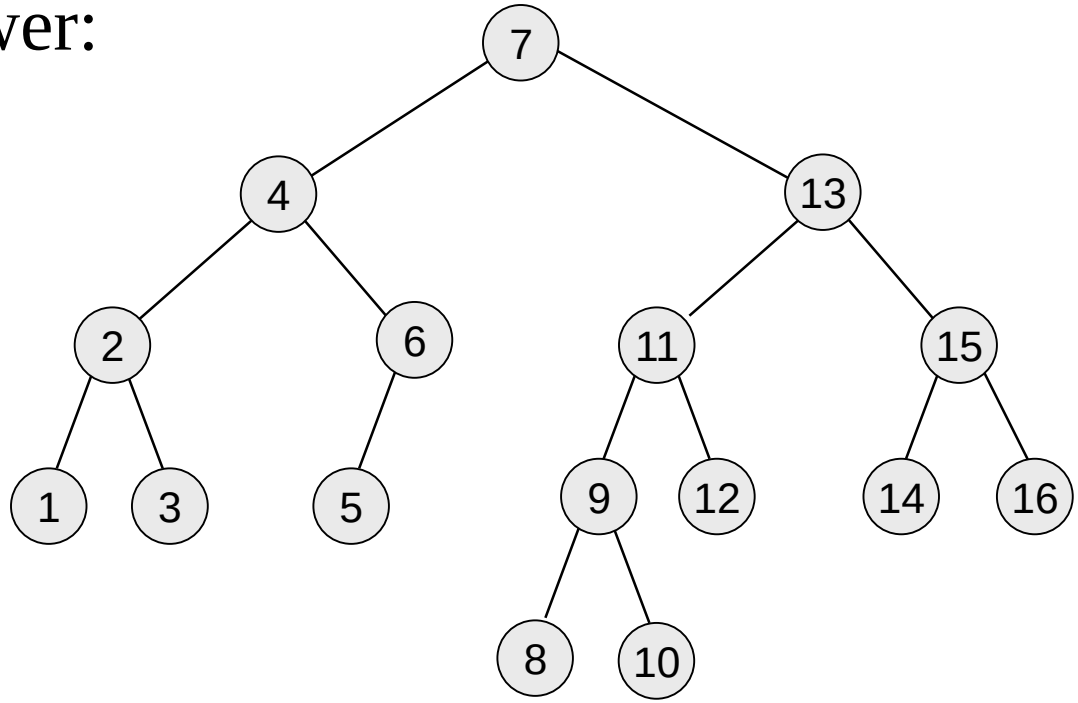


(a) Before rotation

(b) After rotation

Example

- Insert 16, 15, 14, 13, 12, 11, 10, 9, and 8 to the previous tree obtained in the previous single rotation example.
- Answer:



Node declaration for AVL trees

```
struct AvlNode
{

   int element;
   struct AvlNode*   left;
   struct AvlNode* right;
   int  height;
};
```