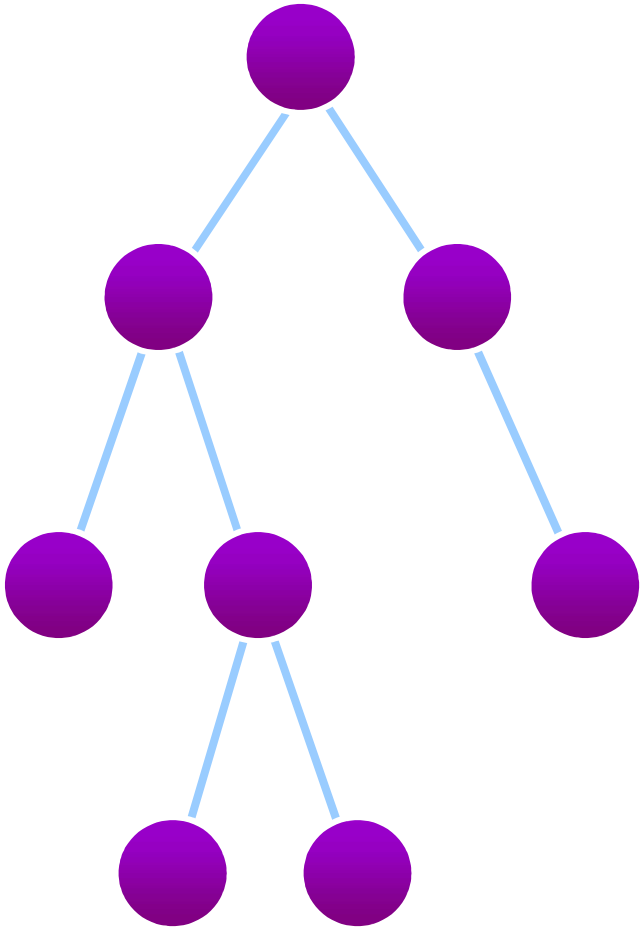
Binary and AVL Trees in C

CHEN Wang

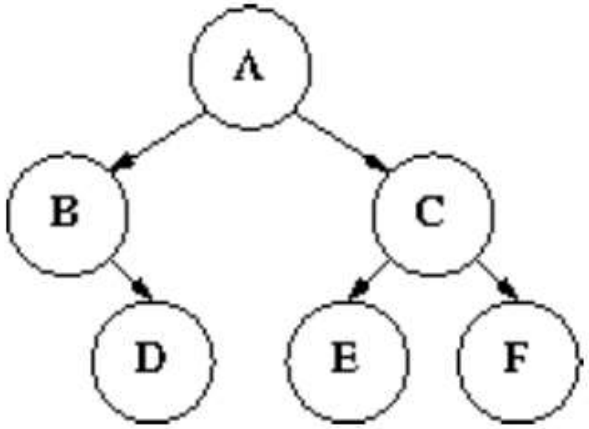
Overview

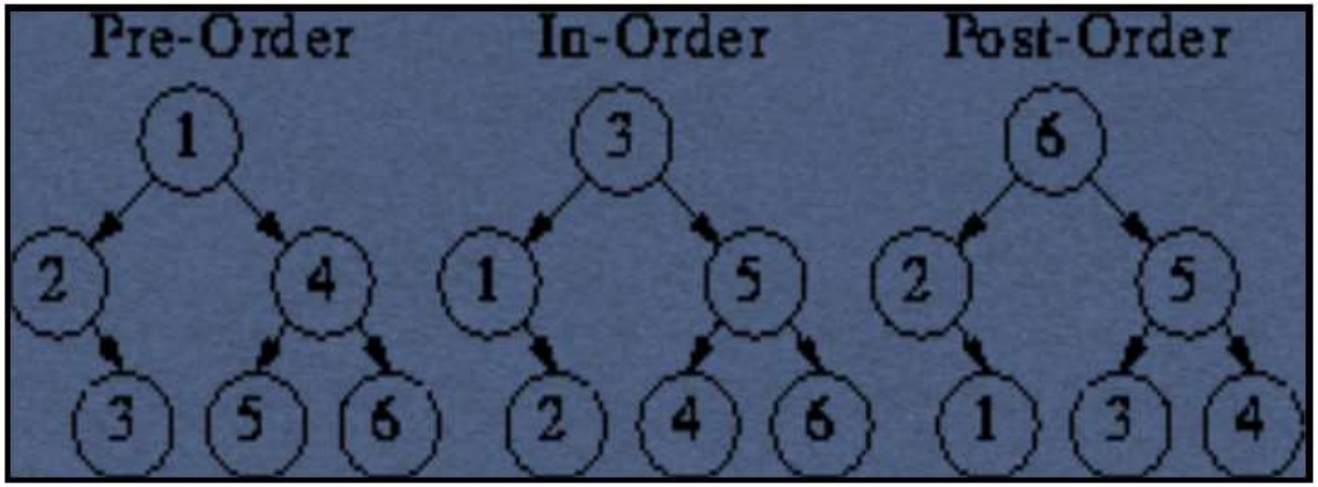
- Binary tree
  - Degree of tree is 2

```
struct node_s {
  Datatype element;
  struct node_s *leftChild;
  struct node_s *rightChild;
};
typedef struct node_s node;
```



Trees – traversal (Recursion Method)





Trees – traversal (Recursion Method)

- Preorder

```
void preorder(node *t) {
   if (t != NULL) {
     printf("%d ", t->element);     /* V */
     preorder(t->leftChild);        /* L */
     preorder(t->rightChild);       /* R */
   }
 }
```

Trees – traversal (Recursion Method)

- Inorder

```
void inorder(node *t) {
  if (t != NULL) {
    inorder(t->leftChild);          /* L */
    printf("%d ", t->element);      /* V */
    inorder(t->rightChild);         /* R */
  }
}
```

Trees – traversal (Recursion Method)

- Postorder

```
void postorder(node *t) {
   if (t != NULL) {
     postorder(t->leftChild);          /* L */
     postorder(t->rightChild);         /* R */
     printf("%d ", t->element);        /* V */
   }
 }
```

Trees - traversal

- Preorder

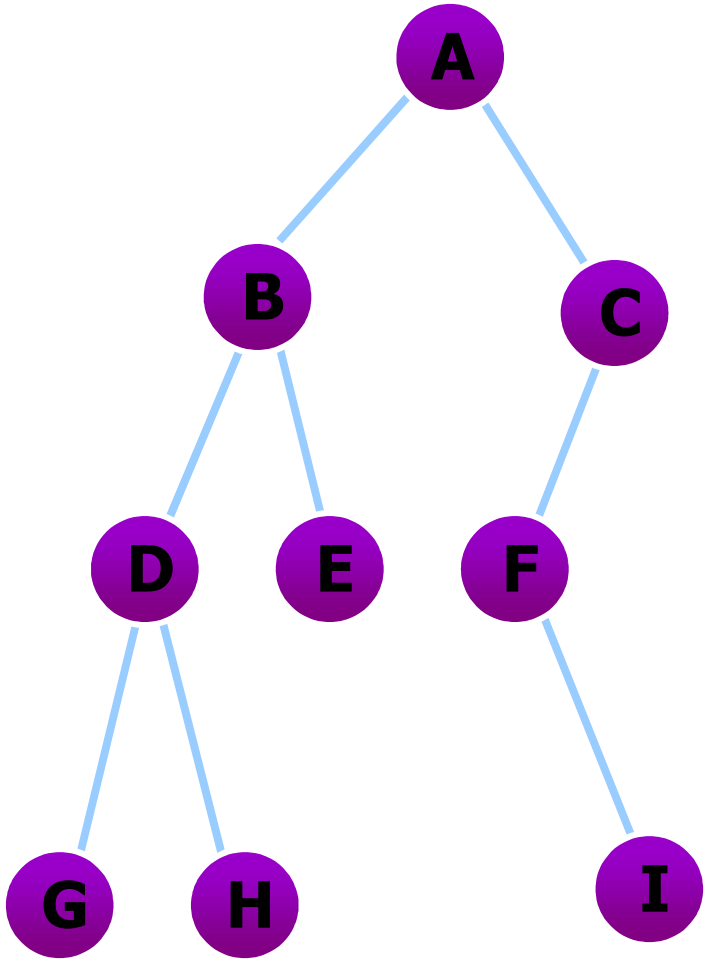
  A B D G H E C F I

- Inorder

  G D H B E A F I C

- Postorder

  G H D E B I F C A



AVL tree

For each node, the difference of height between left and right are no more than 1.

```
struct AVLnode_s {
  Datatype element;
  struct AVLnode *left;
  struct AVLnode *right;
};
typedef struct AVLnode_s AVLnode;
```

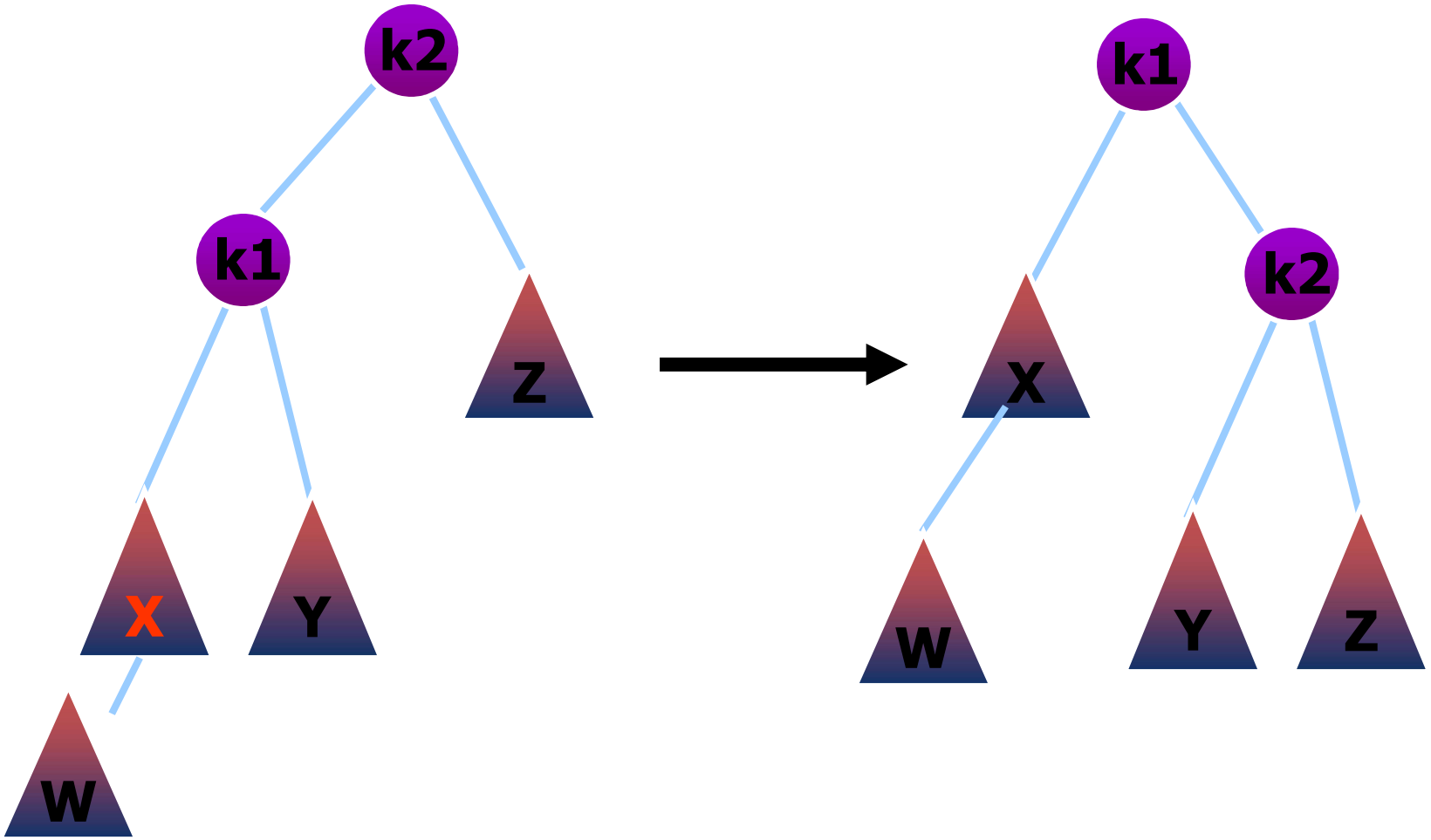
Four Models

- There are four models about the operation of AVL Tree:

  LL RR LR RL

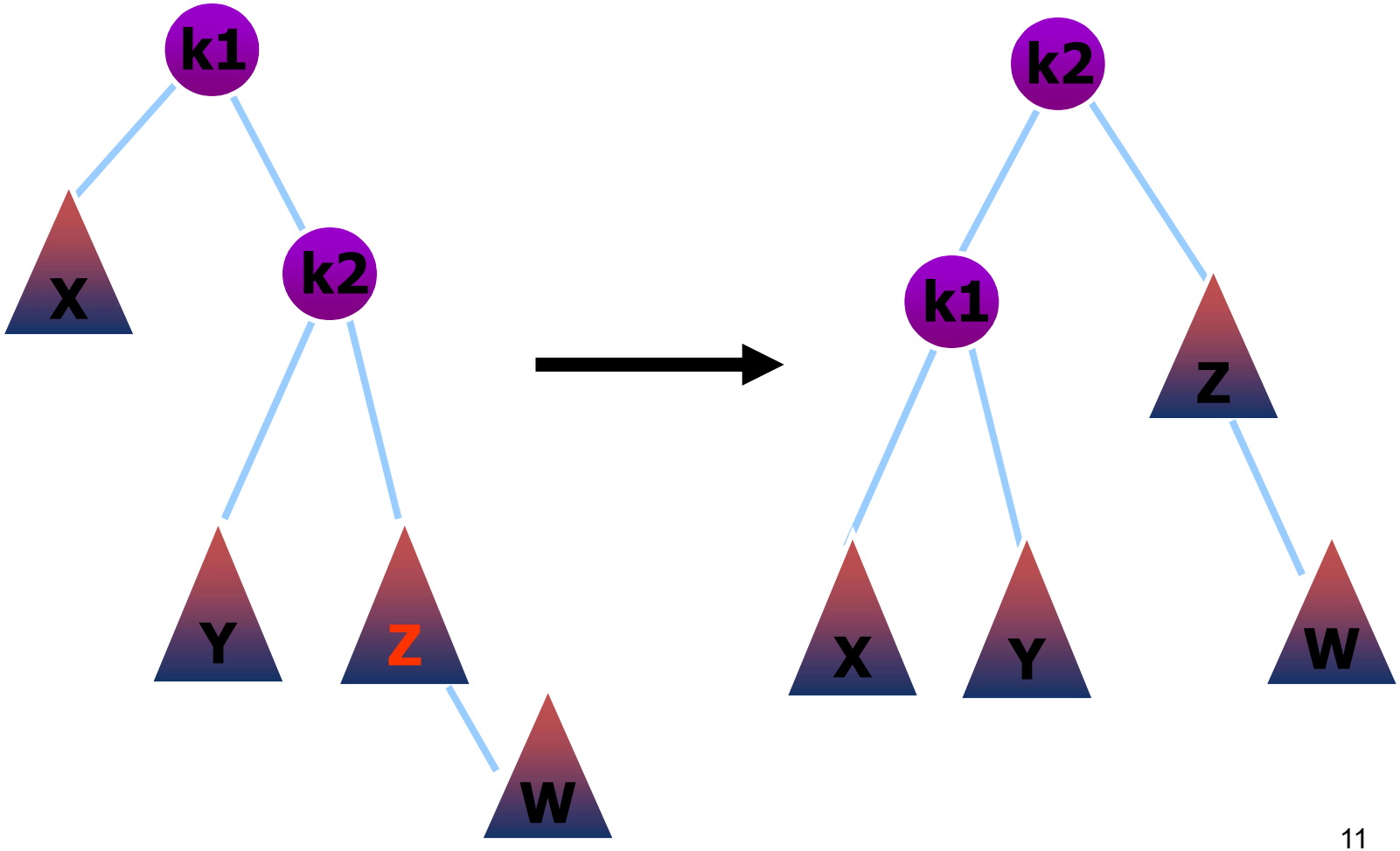
AVL tree

- Single rotation: left-left (LL)



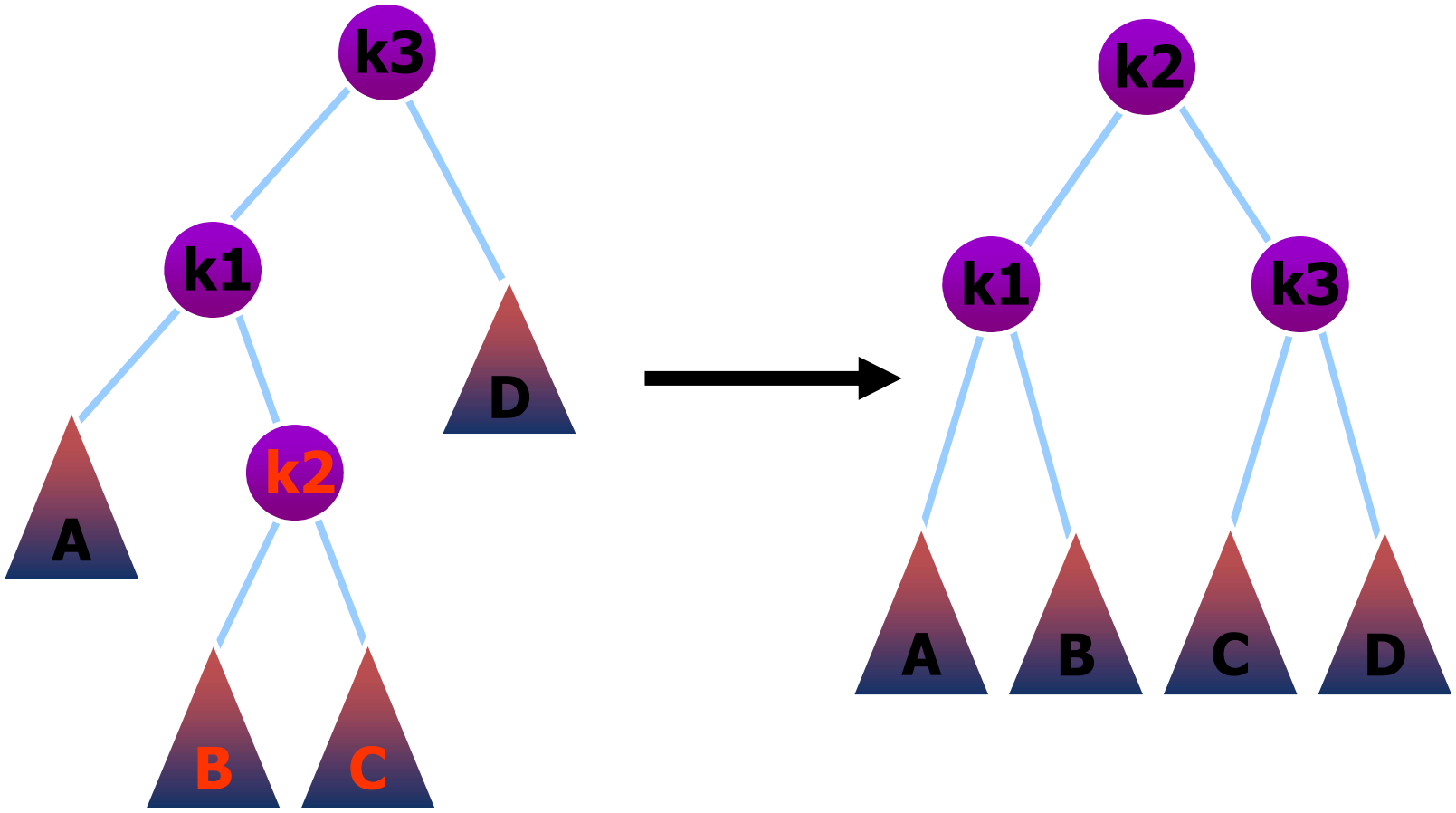
AVL tree

- Single rotation: right-right (RR)



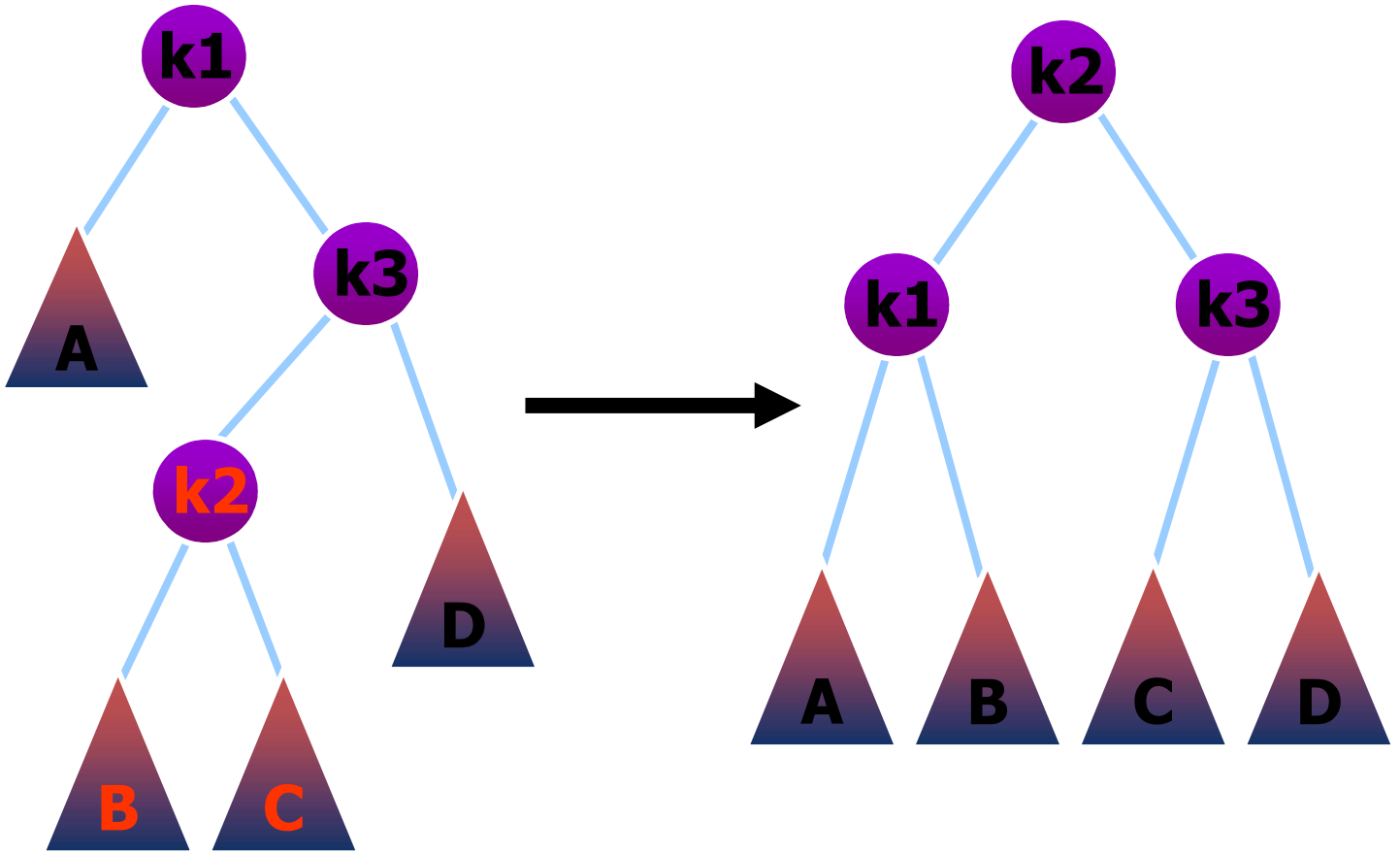
AVL tree

- Double rotation: left-right (LR)

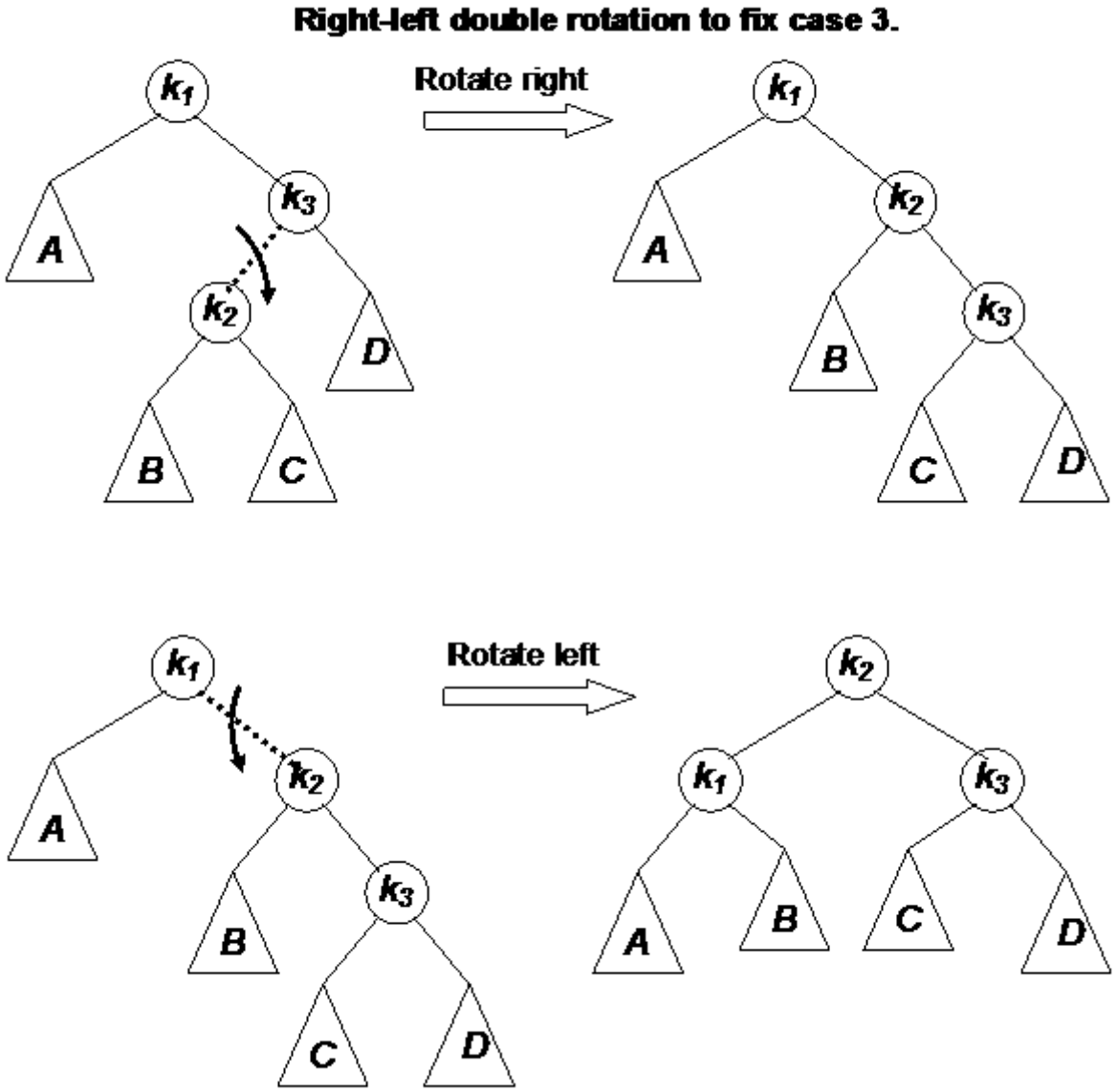


AVL tree

- Double rotation: right-left (RL)



Double rotation: right-left (RL)



Balancing an AVL tree after an insertion

- Begin at the node containing the item which was just inserted and move back along the access path toward the root.{
  - For each node determine its height and check the balance condition. {
    - If the tree is AVL balanced and no further nodes need be considered.
    - else If the node has become unbalanced, a rotation is needed to balance it.

    }

} we proceed to the next node on the access path.

```
AVLnode *insert(Datatype x, AVLnode *t) {
        if (t == NULL) {
                /* CreateNewNode */
        }
        else if (x < t->element) {
                t->left = insert(x, t->left);
                /* DoLeft */
        }
        else if (x > t->element) {
                t->right = insert(x, t->right);
                /* DoRight */
        }
}
```

AVL tree

- **CreateNewNode**

```
t = malloc(sizeof(struct AVLnode);
t->element = x;
t->left = NULL;
t->right = NULL;
```

AVL tree

- **DoLeft**

```
if (height(t->left) - height(t->right) == 2)
    if (x < t->left->element)
            t = singleRotateWithLeft(t); // LL
    else
            t = doubleRotateWithLeft(t); // LR
```

AVL tree

- **DoRight**

```
if (height(t->right) - height(t->left) == 2)
    if (x > t->right->element)
            t = singleRotateWithRight(t); // RR
    else
            t = doubleRotateWithRight(t); // RL
```

Kindly reminder of Midterm

- Programming environment would be Linux
- We will provide main stream editors: **Vi**/**Vim**, **Emacs**, **Gedit**, **Nano**. Please get familiar with at least one of them. (There is NO Visual Studio under Linux)
- You also need to learn how to compile program by command line
- e.g.: gcc myprog.c –o myprog
- And run it by command: ./myprog