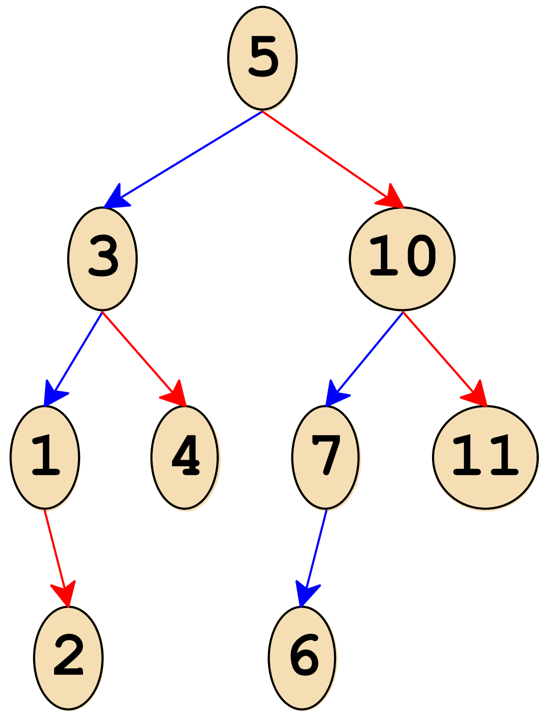
Adding to a Sorted Sequence

What if you need to frequently **find** and **insert** ordered items?

- Array: can find in O(log n) time, but takes O(n) time to insert into the middle
- Doubly-linked list: can insert in O(1) time, but takes O(n) time to find position

A ***binary search tree*** can make both find and insert O(log n) time

Binary Search Tree



Binary Search Tree

```
; An X-tree is either
;  - empty
;  - (make-node X X-tree X-tree)
(define-struct node (value left right))

(define (leaf v) (make-node v empty empty))
(define (branch v l r) (make-node v l r))

(define num-tree
  (branch 5
          (branch 3
                  (branch 1 empty (leaf 2))
                  (leaf 4))
          (branch 10
                  (branch 7 (leaf 6) empty)
                  (leaf 11))))
```

Binary Search Tree

```
; A dir is either 'too-big, 'too-small, or 'same

; btsearch X-tree (X -> dir) -> X-or-false
(define (btsearch t check)
  (cond
    [(empty? t) false]
    [else
     (define d (check (node-value t)))
     (cond
       [(eq? d 'too-big)
        (btsearch (node-left t) check)]
       [(eq? d 'too-small)
        (btsearch (node-right t) check)]
       [else (node-value t)])]))
```

Binary Search Tree

See **`btsearch`** in **`btsearch.c`**

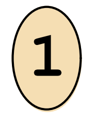
Binary Search Tree Inserts

```
; btinsert X-tree X (X -> dir) -> X-tree
(define (btinsert t v check)
  (cond
    [(empty? t) (leaf v)]
    [else
     (define d (check (node-value t)))
     (cond
       [(eq? d 'too-big)
        (branch (node-value t)
                (btinsert (node-left t) v check)
                (node-right t))]
       [(eq? d 'too-small)
        (branch (node-value t)
                (node-left t)
                (btinsert (node-right t) v check))]
       [else t])]))
```

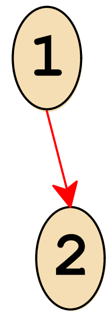
Binary Search Tree Inserts

See **`btinsert`** in **`btsearch.c`**

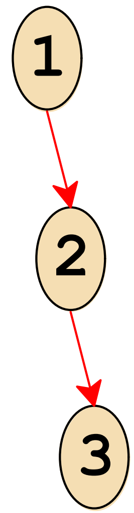
Unbalanced Tree



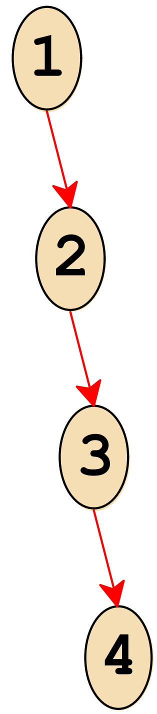
Unbalanced Tree



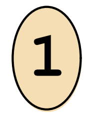
Unbalanced Tree



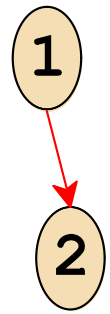
Unbalanced Tree



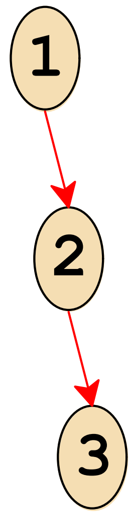
Balancing a Tree



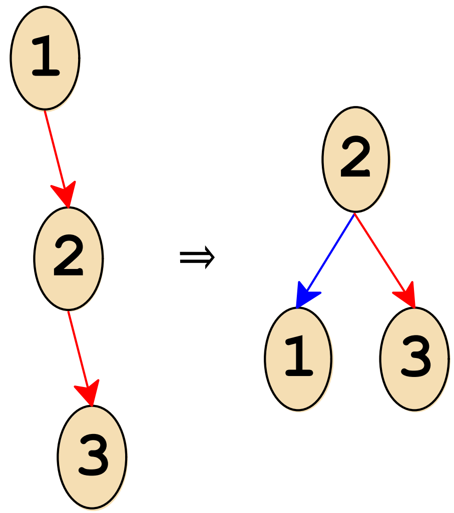
Balancing a Tree



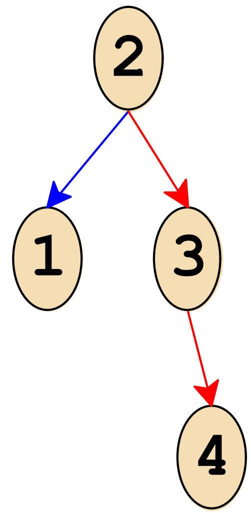
Balancing a Tree



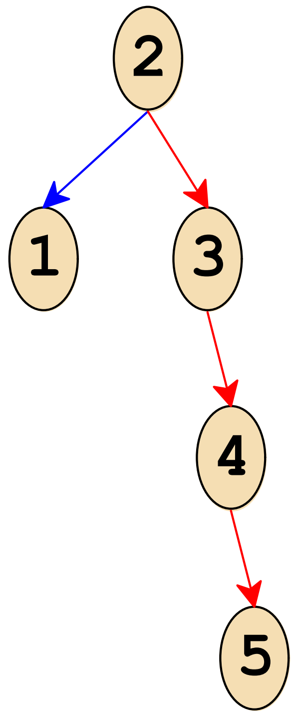
Balancing a Tree



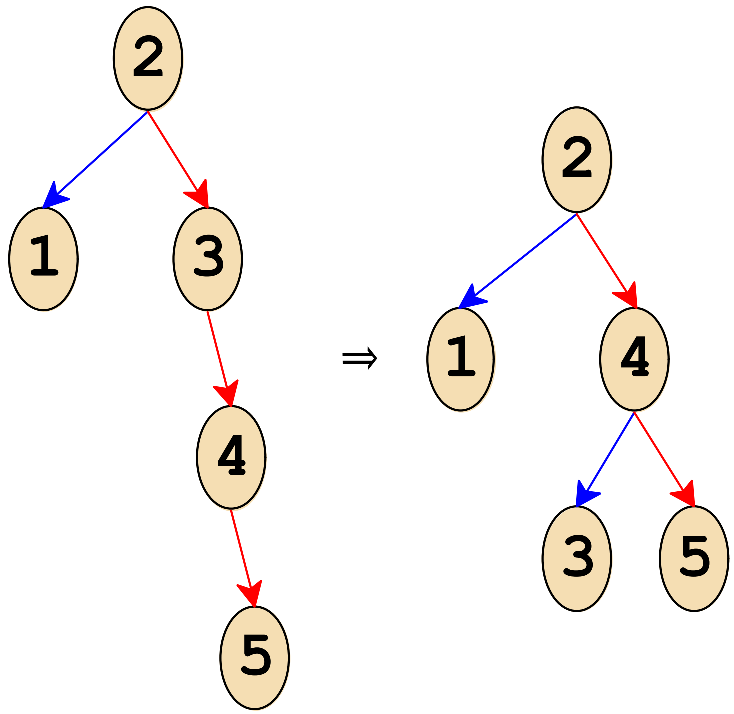
Balancing a Tree



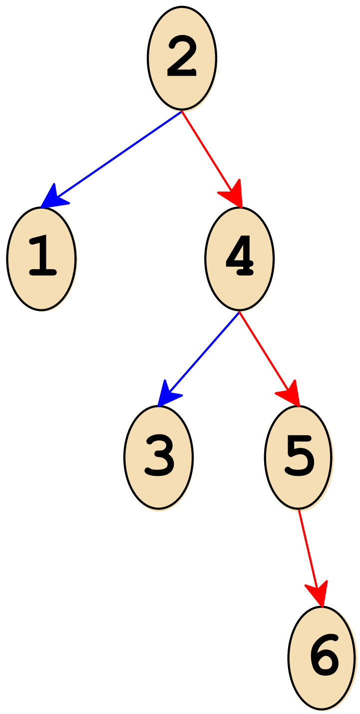
Balancing a Tree



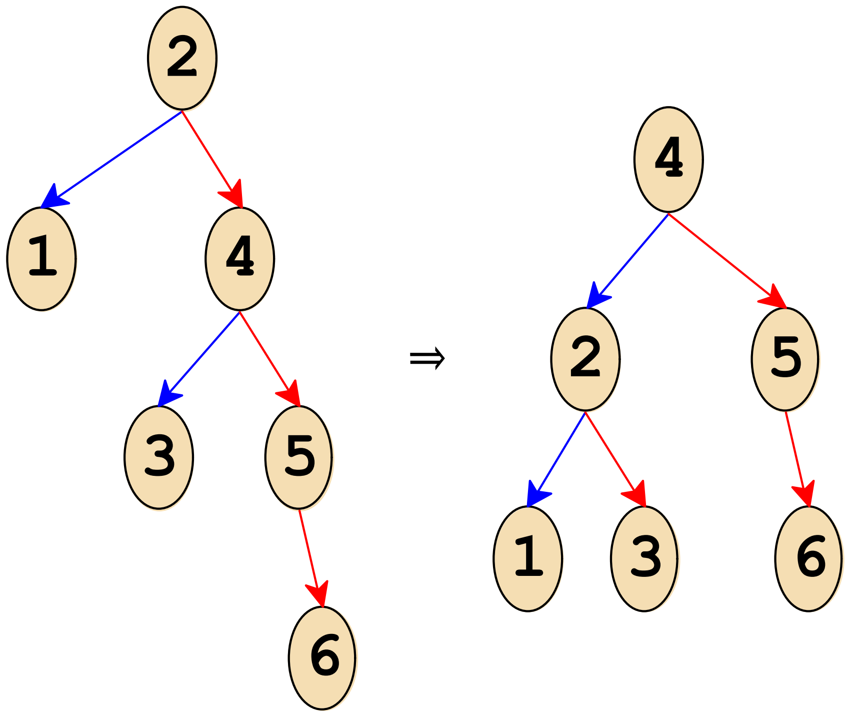
Balancing a Tree



Balancing a Tree

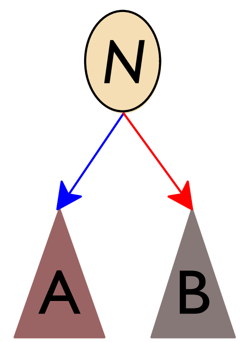


Balancing a Tree



AVL Trees

An ***AVL tree*** uses a particular balancing strategy

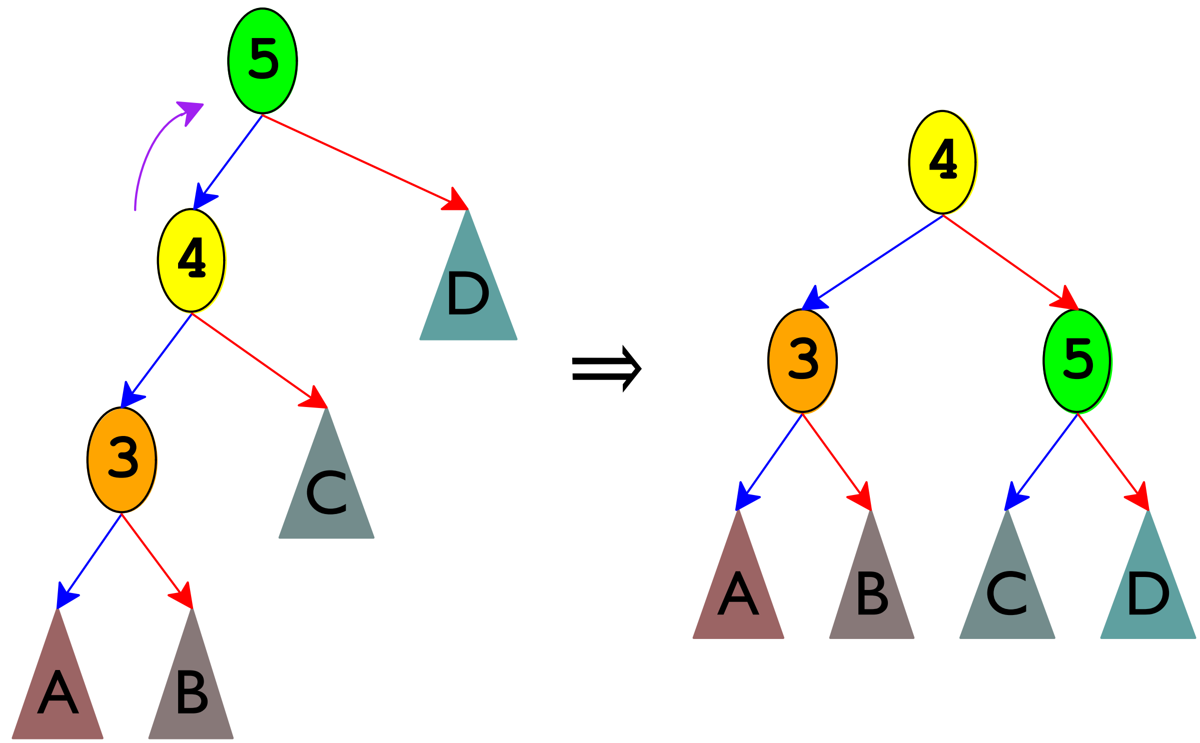


Define ***balance*** at *N* as

$$\text{height}(A) \quad - \quad \text{height}(B)$$

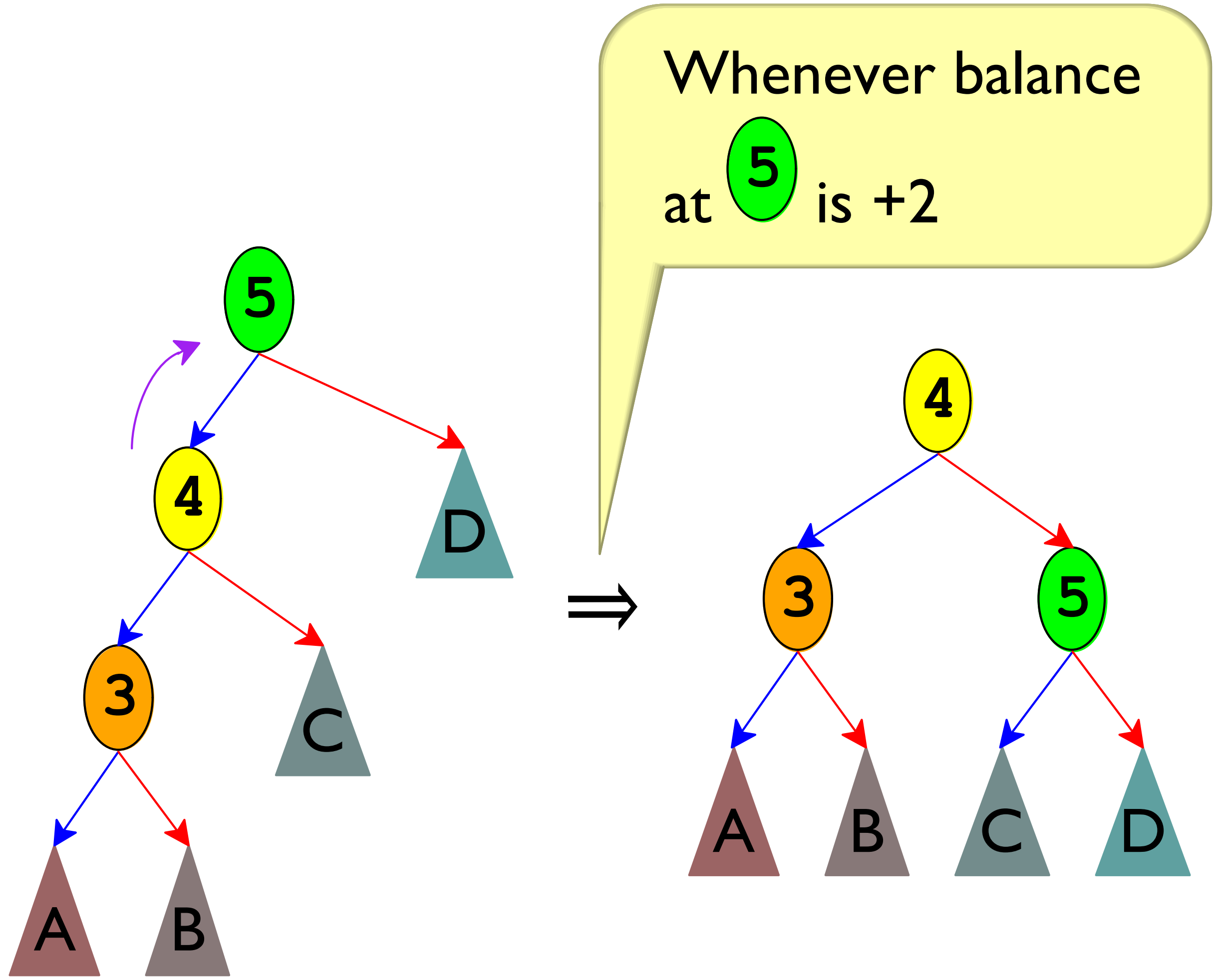
After insert, a balance of ±2 triggers rotations

AVL Trees



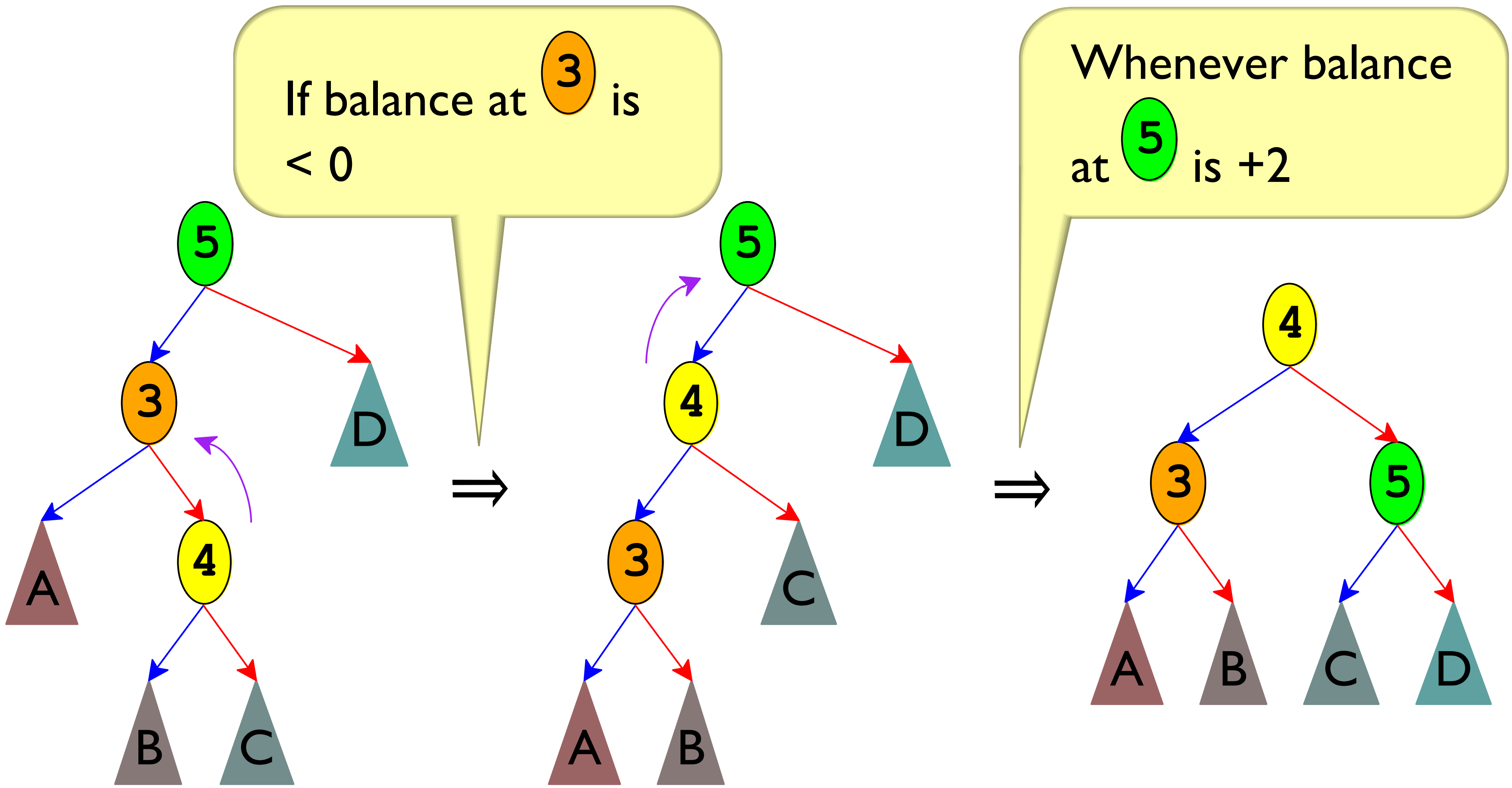
picture based on **`http://en.wikipedia.org/wiki/AVL_tree`**

AVL Trees



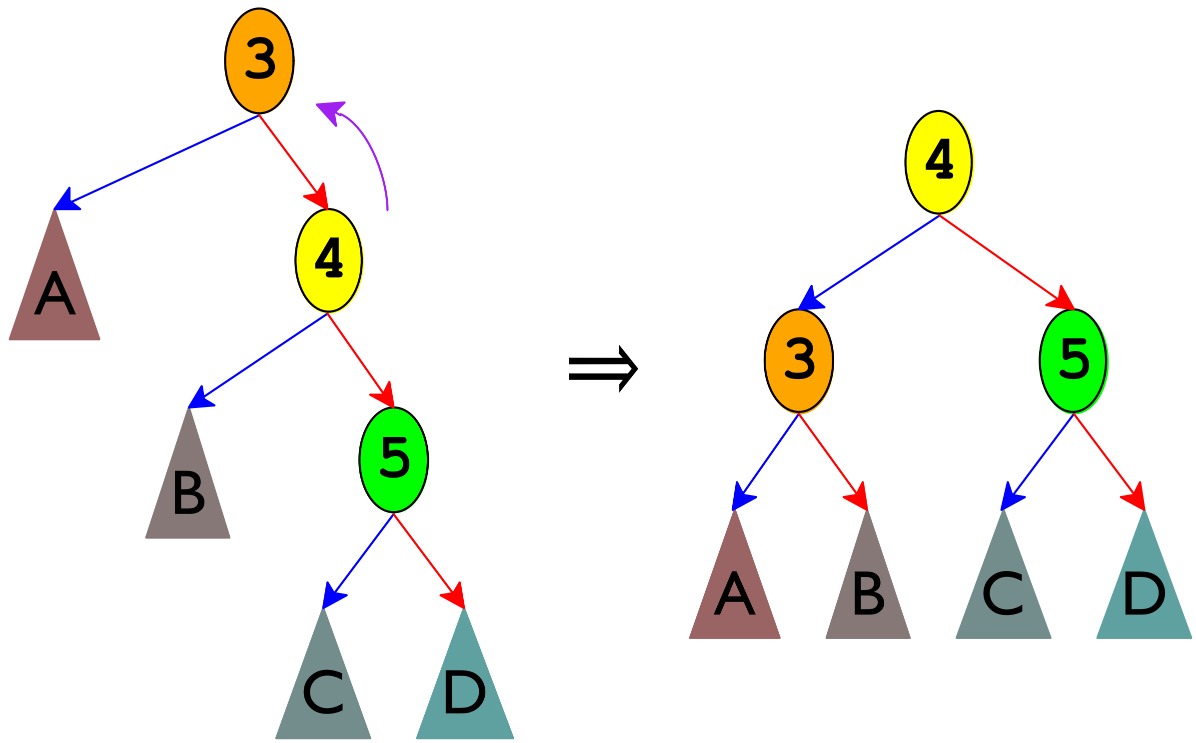
picture based on **`http://en.wikipedia.org/wiki/AVL_tree`**

AVL Trees



picture based on **http://en.wikipedia.org/wiki/AVL_tree**

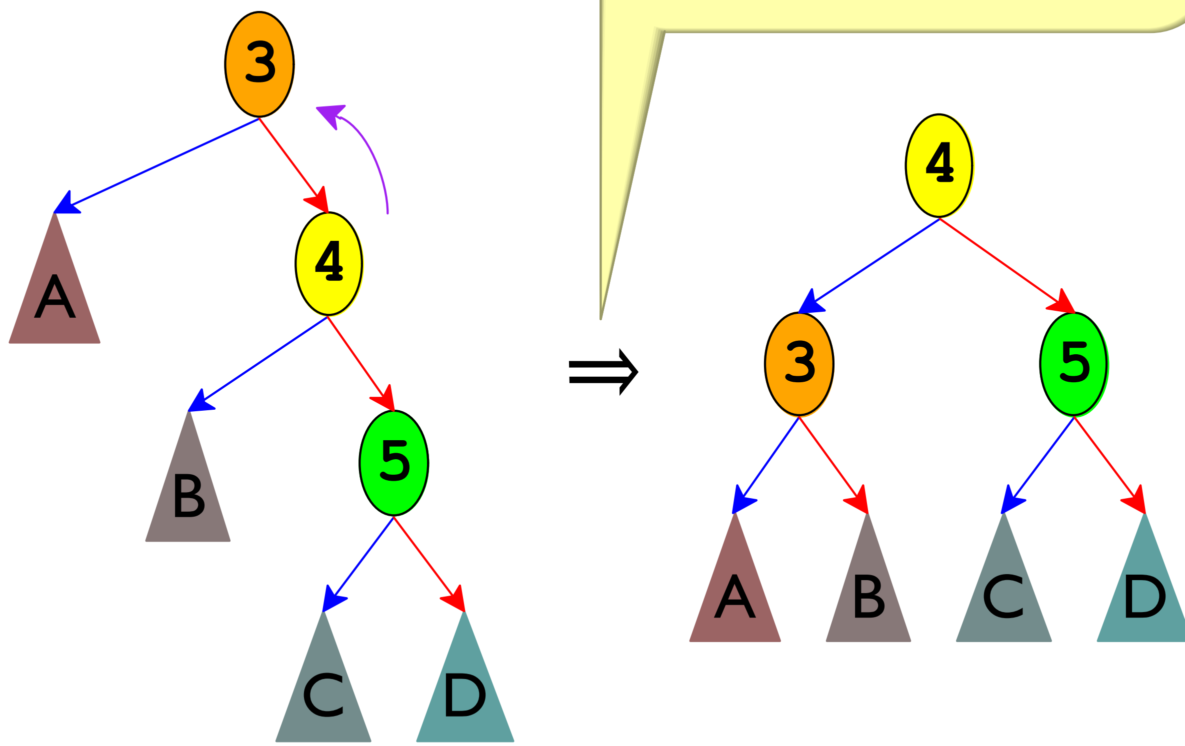
AVL Trees



picture based on **`http://en.wikipedia.org/wiki/AVL_tree`**

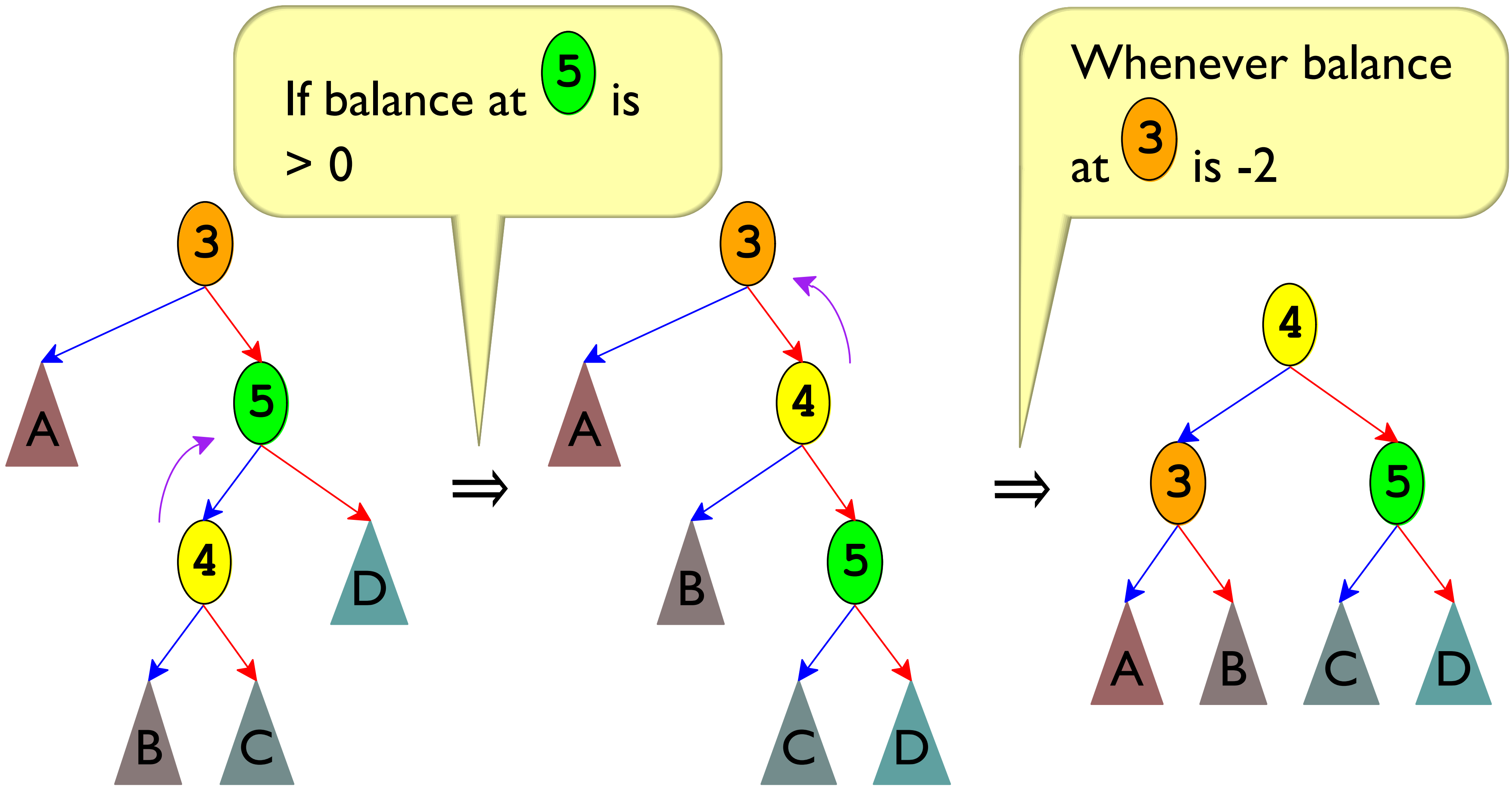
AVL Trees

Whenever balance at 3 is -2



picture based on `http://en.wikipedia.org/wiki/AVL_tree`

AVL Trees



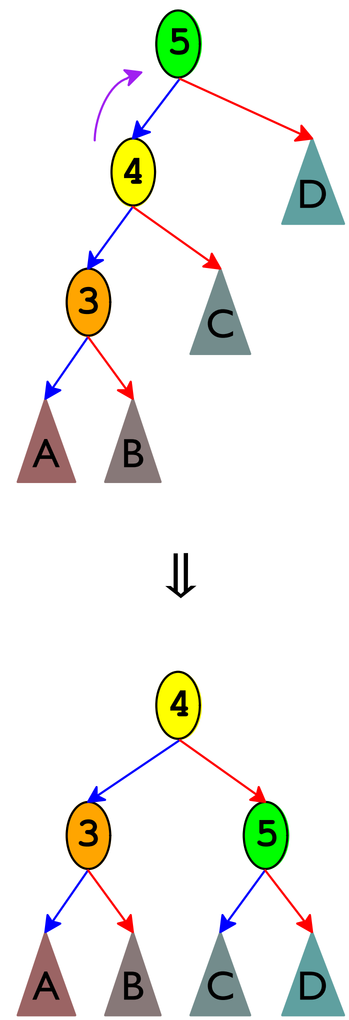
picture based on **`http://en.wikipedia.org/wiki/AVL_tree`**

AVL Trees

See **`avl.c`**

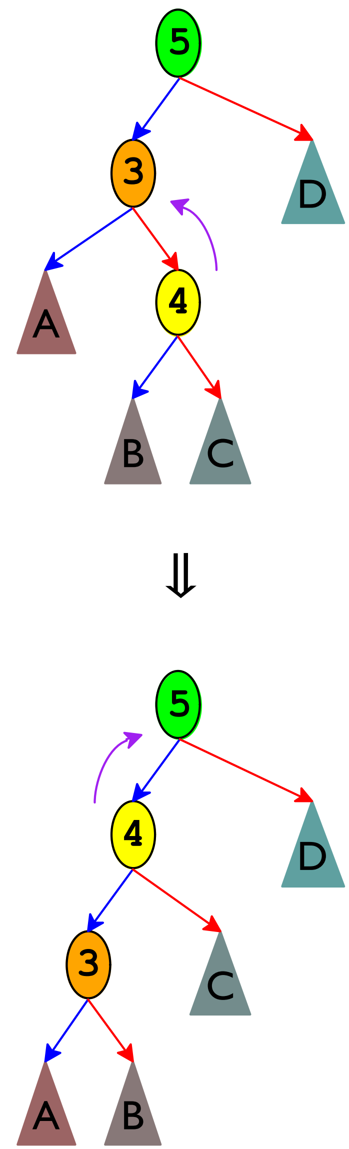
AVL Rotation Code

```
if (get_balance(t) == 2) {
  /* need to rotate right */
  tree left = t->left;
  if (get_balance(left) < 0) {
    /* double right rotation */
    tree left_right = left->right;
    left->right = left_right->left;
    left_right->left = left;
    fix_height(left);
    left = left_right;
  }
  t->left = left->right;
  left->right = t;
  fix_height(t);
  fix_height(left);
  return left;
}
```



AVL Rotation Code

```
if (get_balance(t) == 2) {
  /* need to rotate right */
  tree left = t->left;
  if (get_balance(left) < 0) {
    /* double right rotation */
    tree left_right = left->right;
    left->right = left_right->left;
    left_right->left = left;
    fix_height(left);
    left = left_right;
  }
  t->left = left->right;
  left->right = t;
  fix_height(t);
  fix_height(left);
  return left;
}
```



JFYI: Red–Black Trees

A ***red-black tree*** uses a similar but different rebalancing strategy

It is often implemented with **`for`** loops instead of recusion, which is/was useful in some settings

JFYI: Splay Trees

A ***splay tree*** uses another balancing approach

Instead of rebalancing after an insert, a splay tree rotates all lookups and inserts to the root