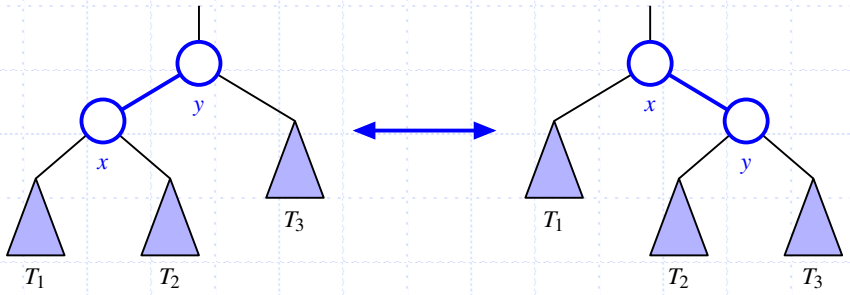
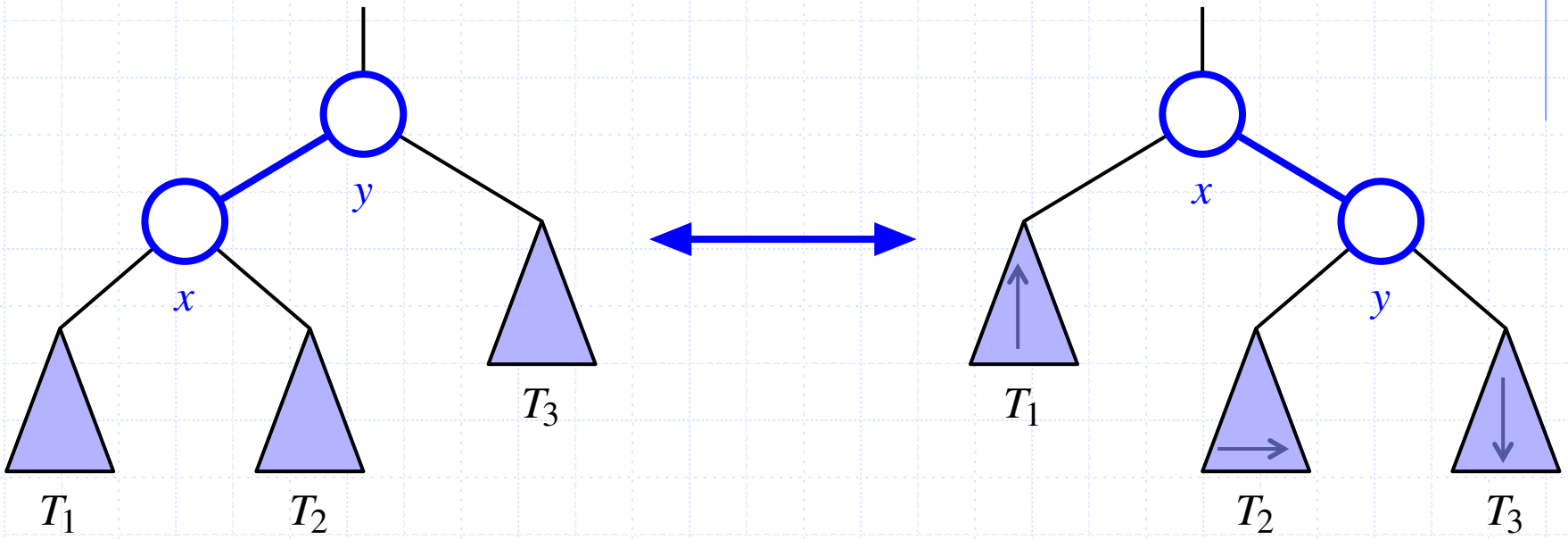
Balanced Search Trees



Rotation Operation

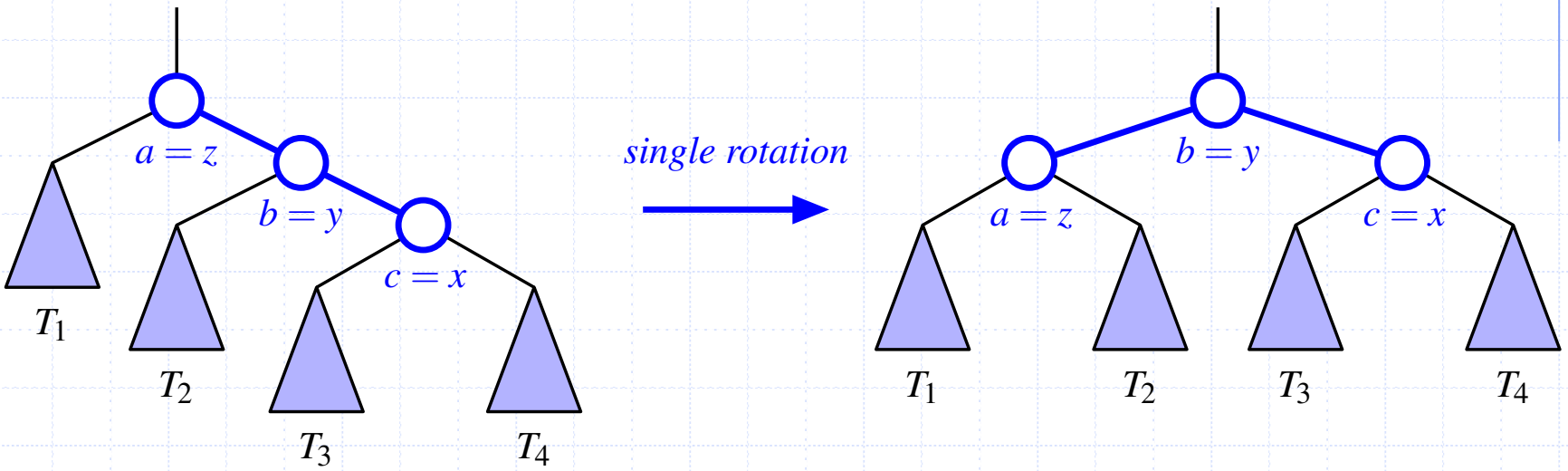


The primary operation to rebalance a BST is a rotation

- Alter the structure in O(1)
- Maintain the search tree property
- Can be combined to provide broader rebalancing

Trinode restructuring

Single Rotation



We consider a positionx, its pareny y, and its grandparent z.
The goal is to restructure the subtree rooted at z
to reduce the path length to x and its subtrees.

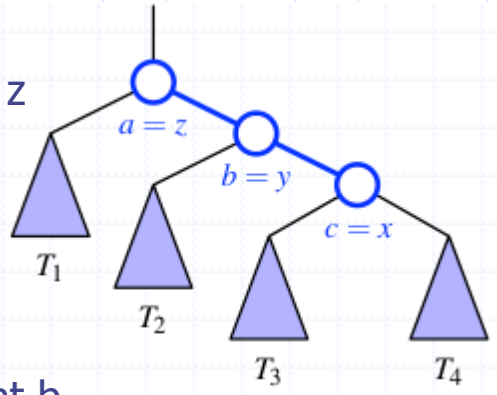
Trinode restructuring
Pseudo-code

restructure(x):

input: position x and BST T

output: T after trinode restructuring

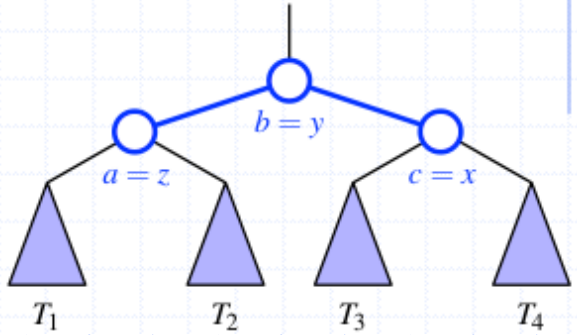
1: Let (a,b,c) be a left-to-right (inorder) listing of x, y and z and let (T1,T2,T3,T4) be an inorder listing of the four subtrees of x, y and z not rooted at x, y, or z.



2: replace subtree rooted at z with a new subtree rooted at b

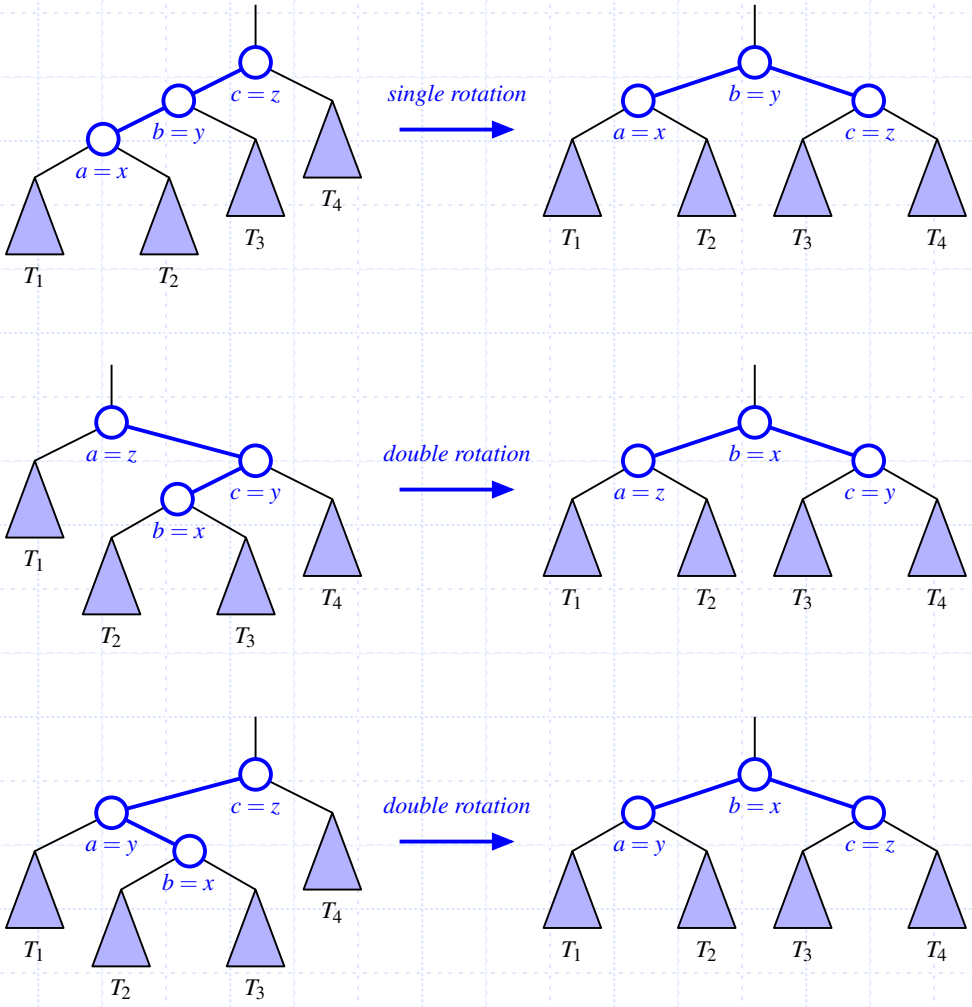
3: Let a be the left child of b and let T1 and T2 be the left and right subtrees of a, respectively

4: Let c be the right child of b and let T3 and T4 be the left and right subtrees of c, respectively

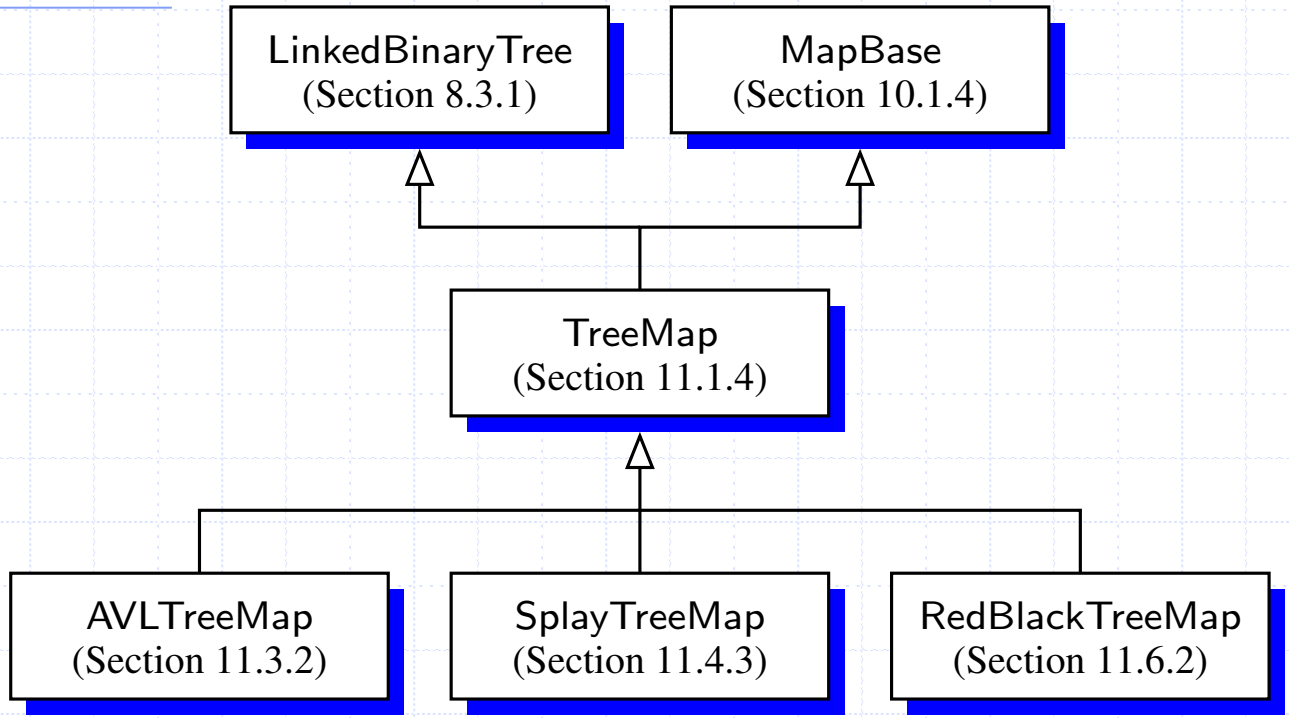


Trinode restructuring

Three other cases



Hierarchy of balanced search trees




```
def _rebalance_delete( self, p ):
    pass

def _rebalance_access( self, p ):
    pass

def _rebalance_insert( self, p ):
    pass
```

Trinode restructuring Python code…

```
def _restructure( self, x ):
    y = self.parent( x )
    z = self.parent( y )
    #check for case #1 and #2
    #single rotation
    if( x == self.right( y ) ) == (y == self.right( z ) ):
        self._rotate( y )
        return y
    #else case #3 or #4
    #double rotation
    else:
        self._rotate( x )
        self._rotate( x )
        return x
```

Trinode restructuring Python code…

```
def _rotate( self, p ):
    x = p._node
    y = x._parent
    z = y._parent
    if z is None: #x becomes root
        self._root = x
        x._parent = None
    else:
        #x becomes direct child of z
        self._relink( z, x, y == z._left )
  #rotate x and y, and transfer middle subtree
    if x == y._left:
        self._relink( y, x._right, True )
        self._relink( x, y, False )
    else:
        self._relink( y, x._left, False )
        self._relink( x, y, True )
```

Trinode restructuring Python code

```
#relink parent and child node (child can be None)
def _relink( self, parent, child, make_left_child ):
   #make it a left child
    if make_left_child:
        parent._left = child
   #make it a right child
    else:
        parent._right = child
   #make child point to parent
    if child is not None:
        child._parent = parent
```