NPTEL MOOC,JAN-FEB 2015
Week 6, Module 2

DESIGN AND ANALYSIS OF ALGORITHMS

Balanced search trees

MADHAVAN MUKUND, CHENNAI MATHEMATICAL INSTITUTE
http://www.cmi.ac.in/~madhavan

Binary search trees

	Heap	Sorted array	Search tree
Find	O(n)	O(log n)	O(log n)
Min	O(1)	O(1)	O(log n)
Max	O(n)	O(1)	O(log n)
Insert	O(log n)	O(n)	O(log n)
Delete	O(log n)	O(n)	O(log n)
Pred	O(n)	O(1)	O(log n)
Succ	O(n)	O(1)	O(log n)

Complexity

- All operations on search trees walk down a single path
- Worst-case: height of the tree
- Balanced trees: height is O(log n) for n nodes
- How to maintain balance as the tree grows and shrinks?

Different notions of balance

- size(left) = size(right)
  - Too rigid, only complete binary trees
- | size(left) - size(right) | ≤ 1
  - More manageable but difficult to incrementally maintain this property

Height balance

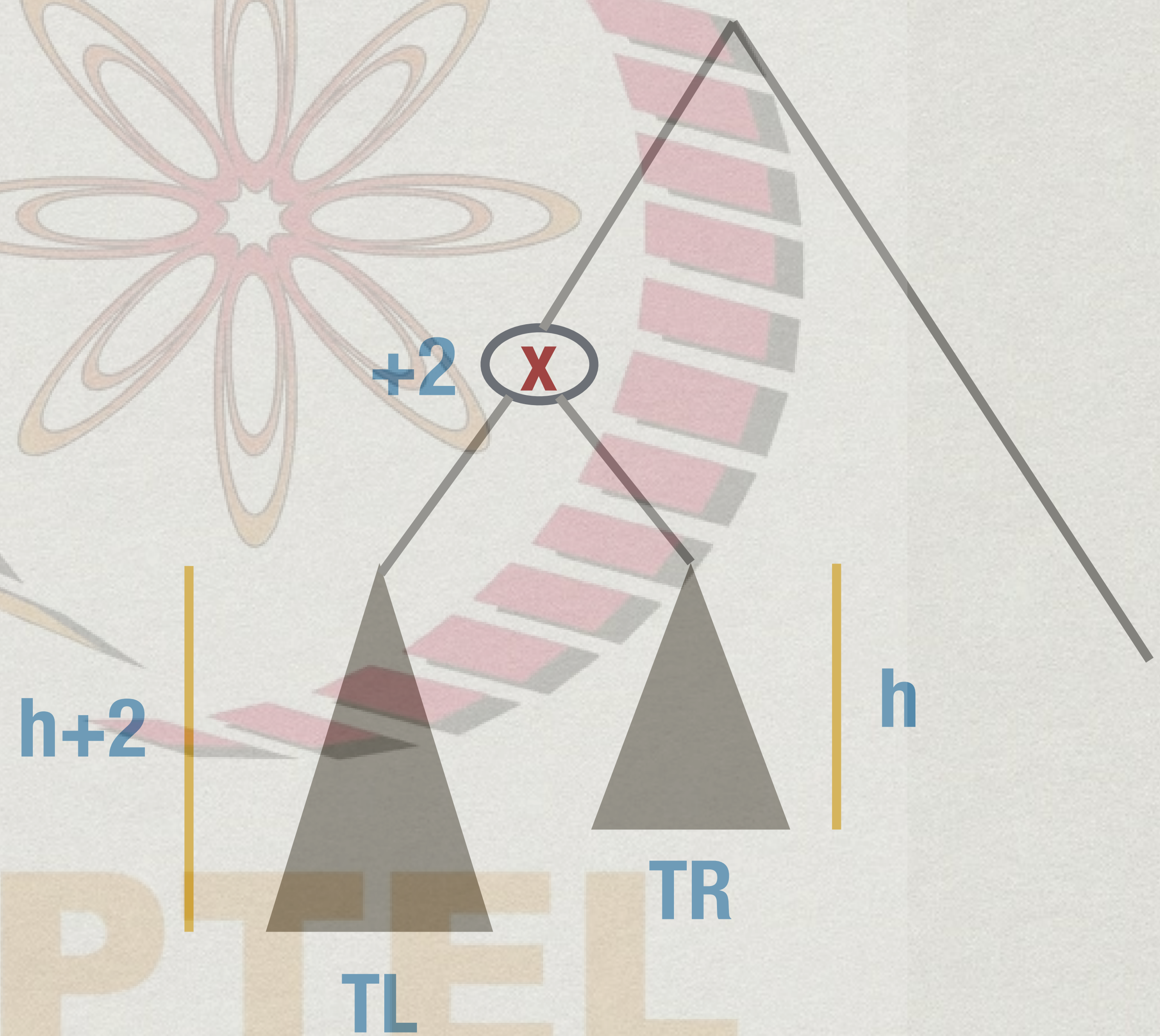
- height: number of nodes in longest path from root to leaf
    - empty tree: height = 0
    - only root: height = 1
- | height(left) - height(right) | ≤ 1
    - Height balanced trees
    - AVL trees (Adelson-Velsky and Landis)

Height balance

- Slope of a node : height(left) - height(right)
- Balanced tree
  - slope is within {-1,0,1} at each node
- insert(v)/delete(v) can disturb slope upto -2 or +2
- Sufficient to rebalance from slope {-2,-1,0,1,2} to {-1,0,1}
  - Rebalance bottom up — assume all lower nodes are balanced

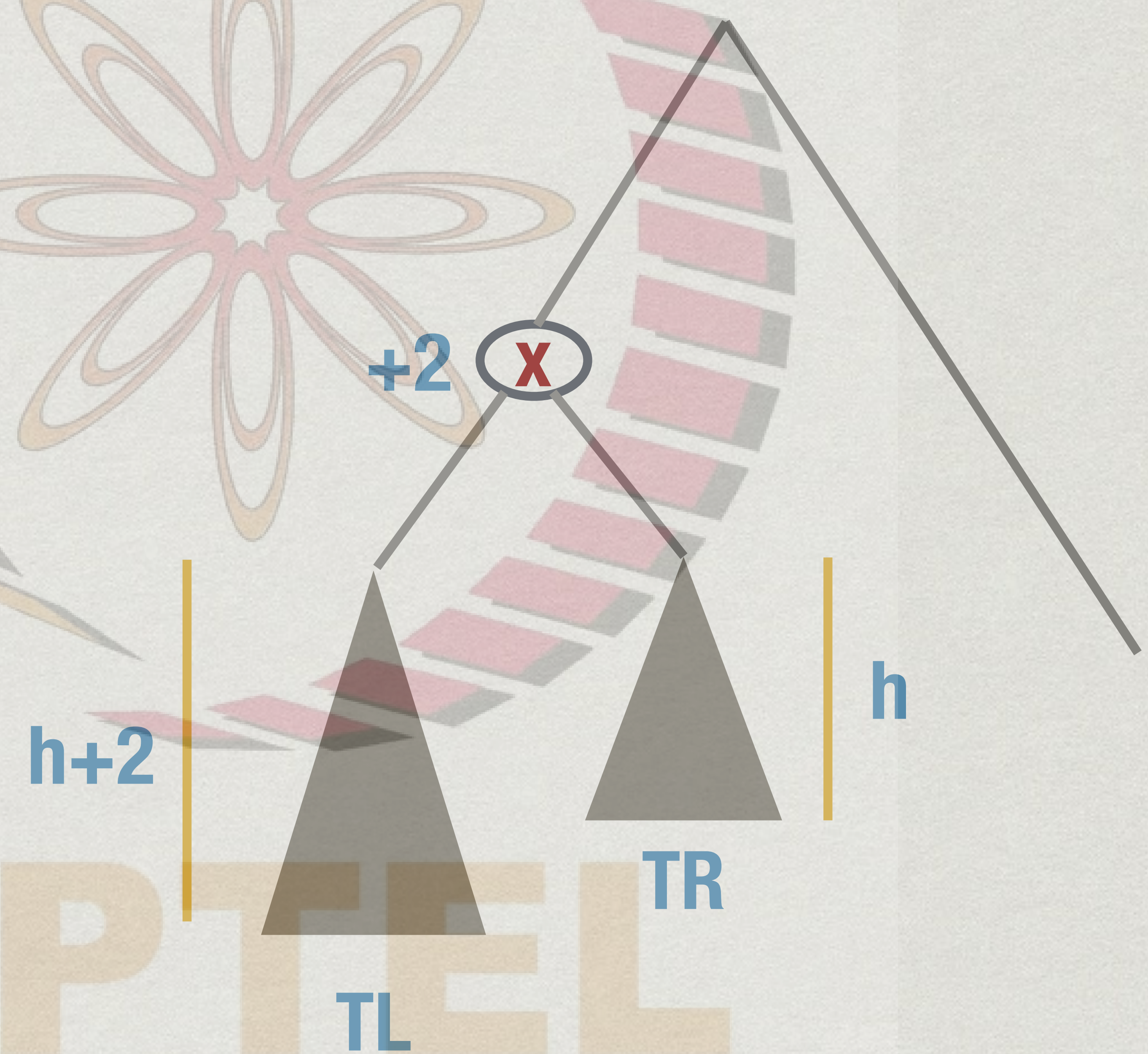
Unbalanced, slope +2

- Node x has slope +2
- Assume left and right subtrees are balanced
  - All slopes in {-1,0,+1}



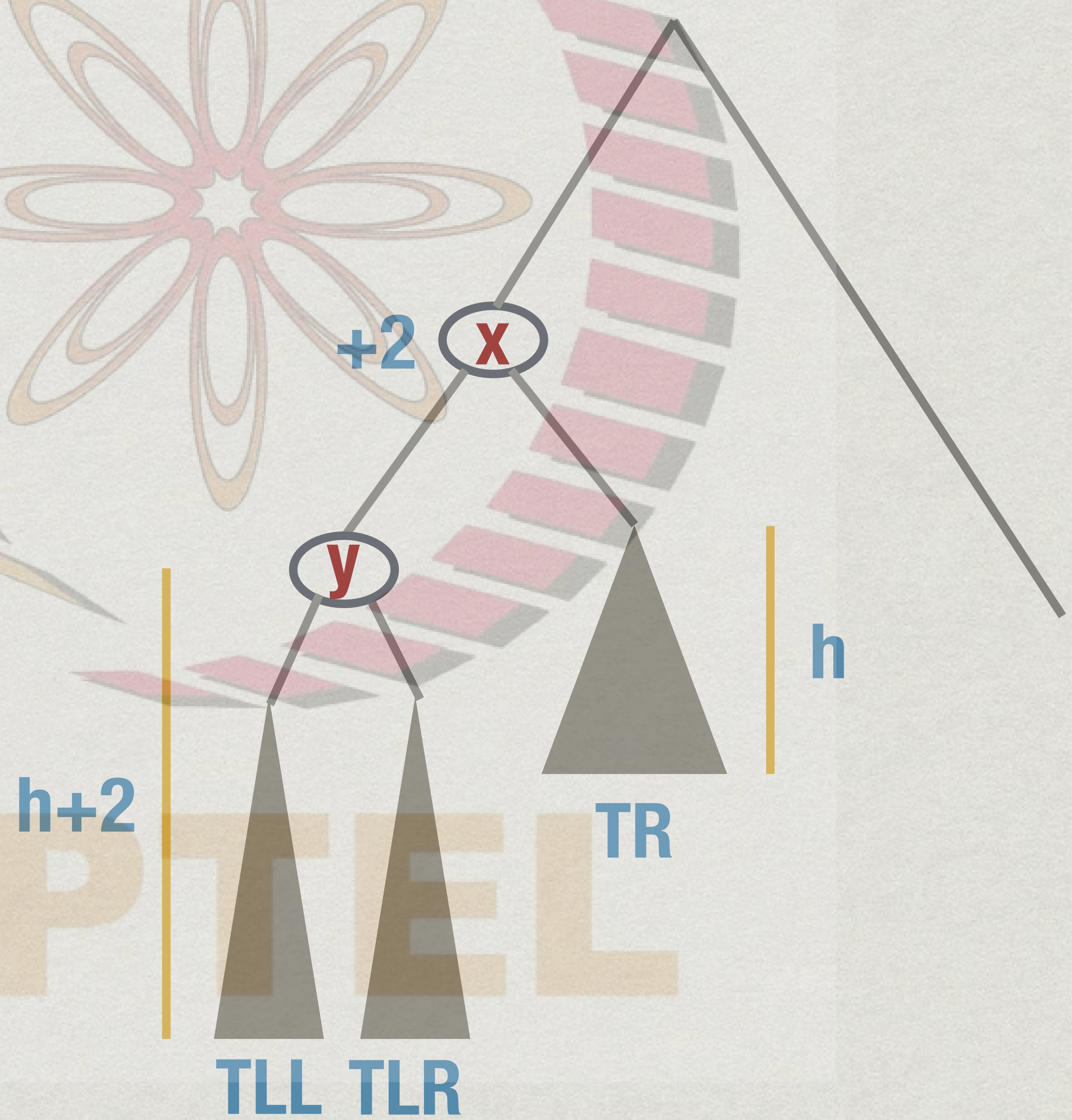
Unbalanced, slope +2

- TL is not empty: expand



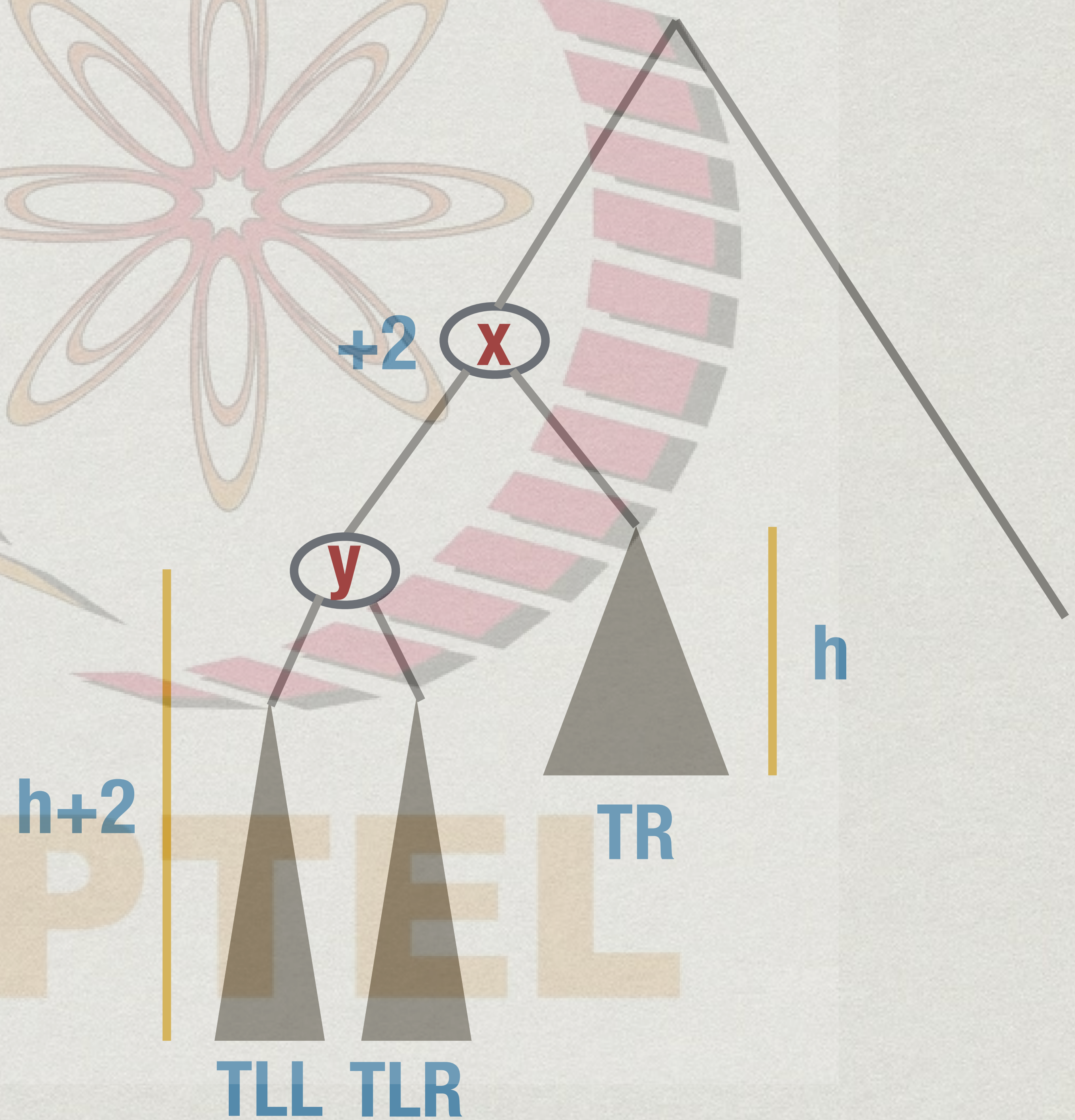
Unbalanced, slope +2

- TL is not empty: expand



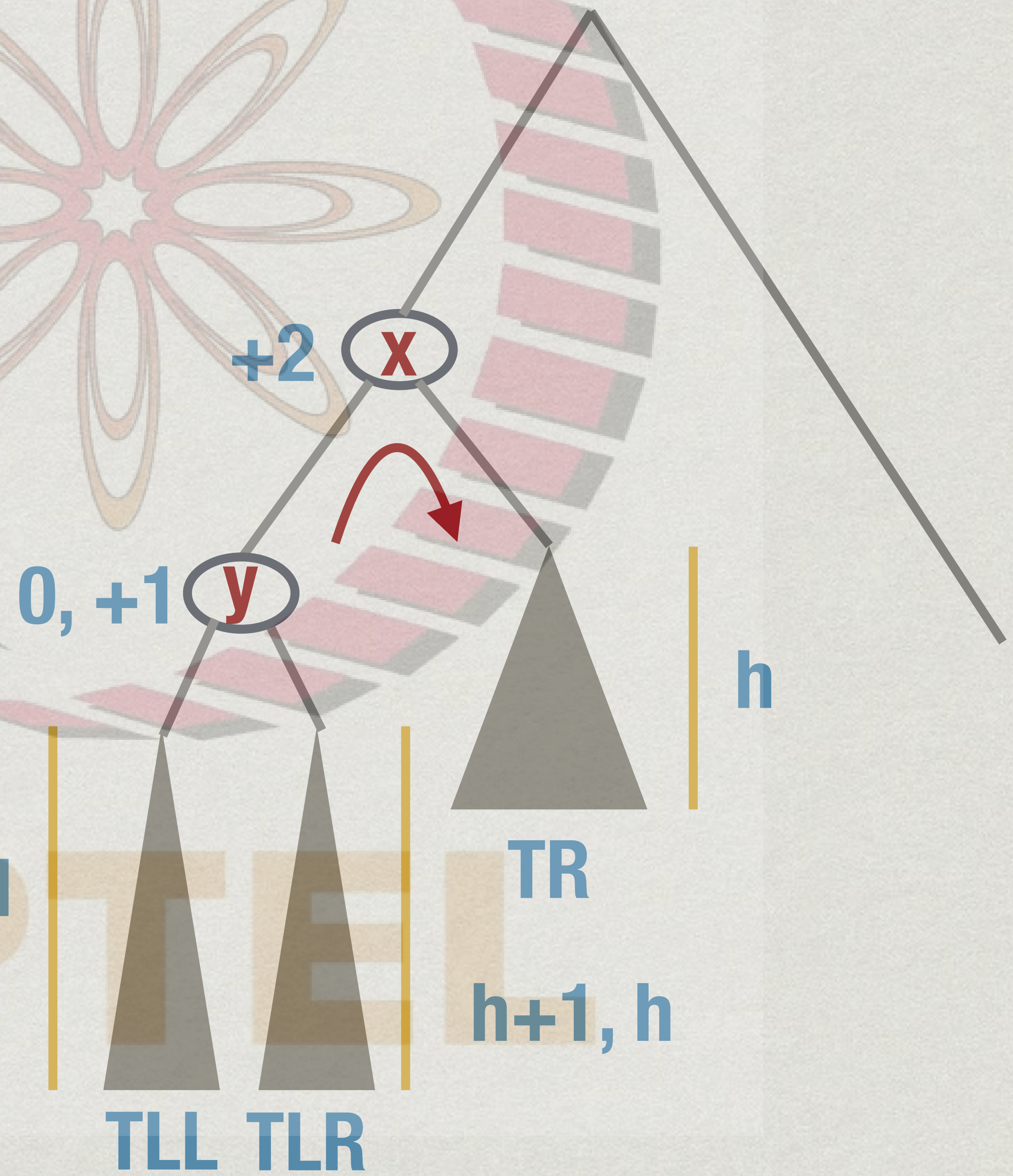
Unbalanced, slope +2

- TL is not empty: expand
- Slope of y is in {-1,0,+1}
  - Bottom up rebalancing
- Case analysis



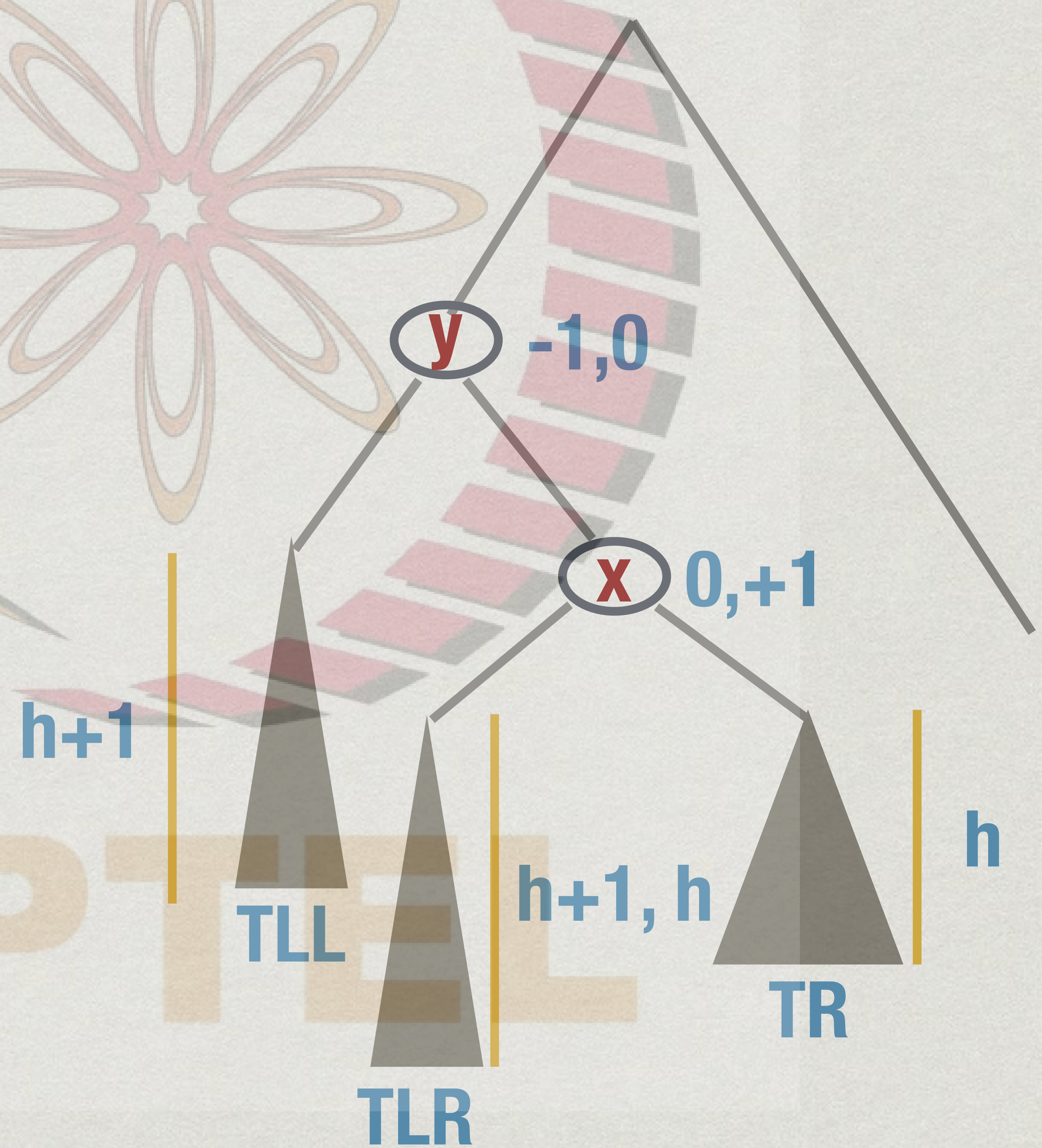
Unbalanced, slope +2

- Case 1: slope of y is {0,+1}
- Rotate the tree right at x



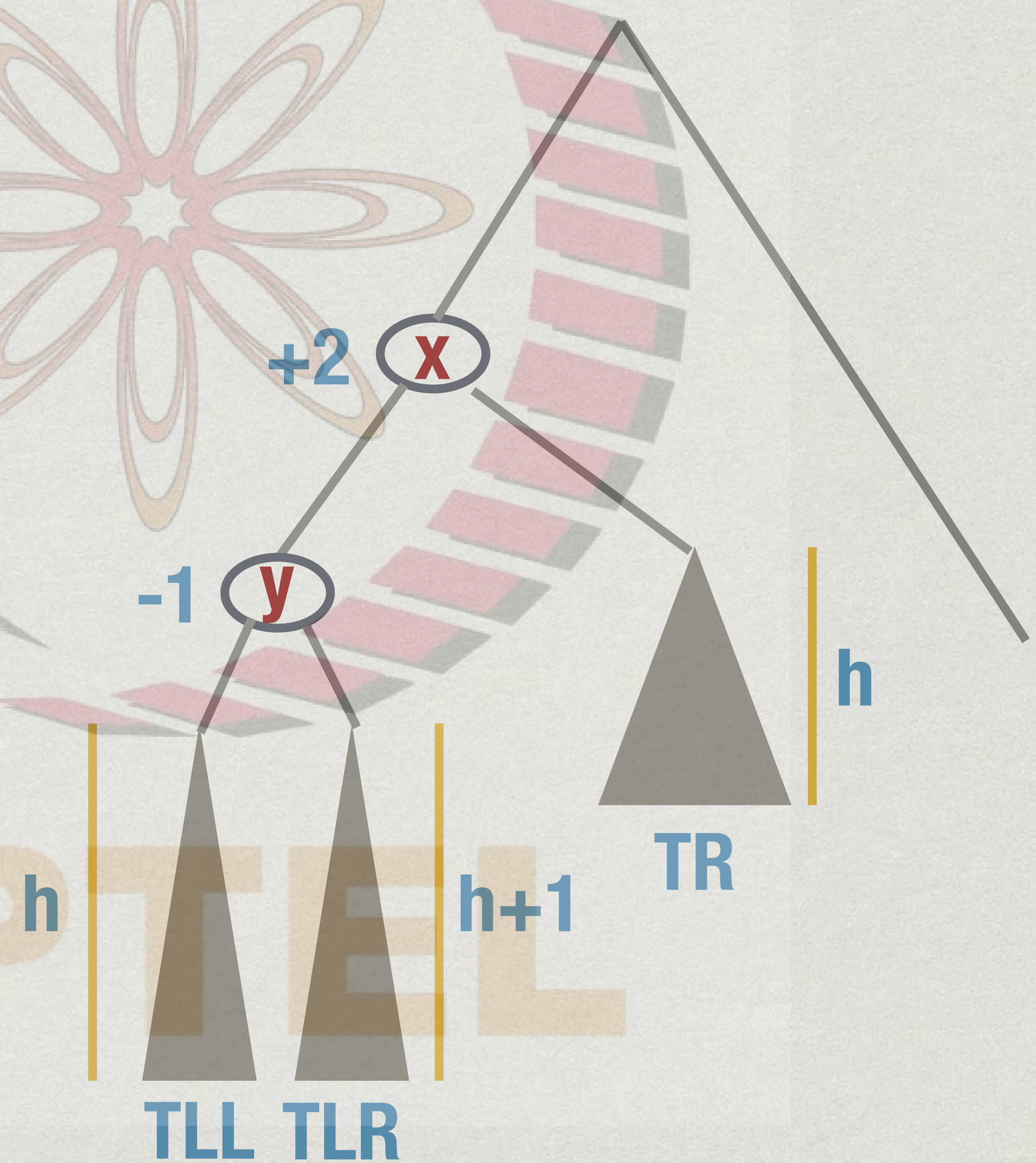
Unbalanced, slope +2

- Case 1: slope of y is {0,+1}
- Rotate the tree right at x
- Rebalanced!



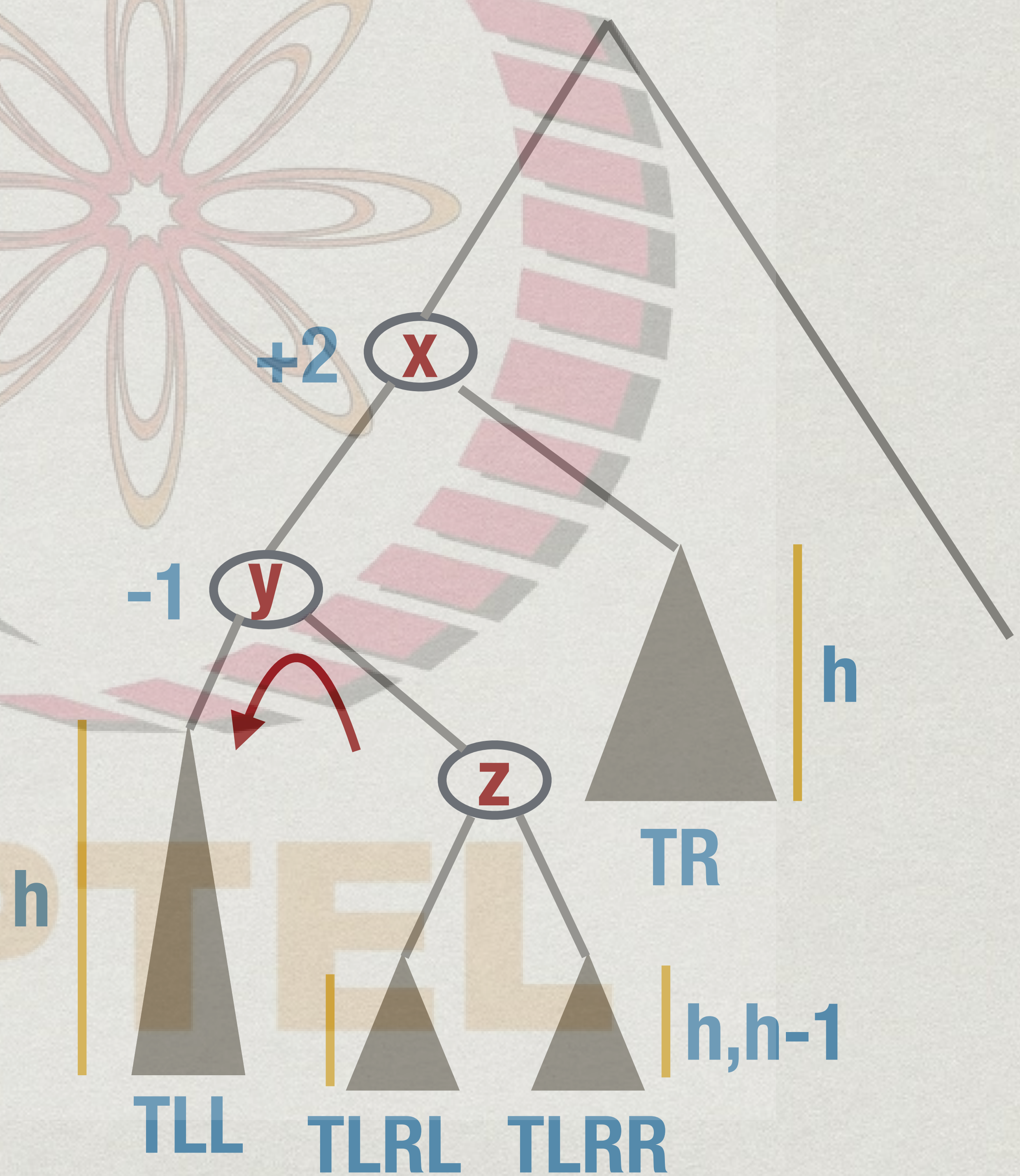
Unbalanced, slope +2

- Case 2: slope of y is {-1}
- Expand TLR



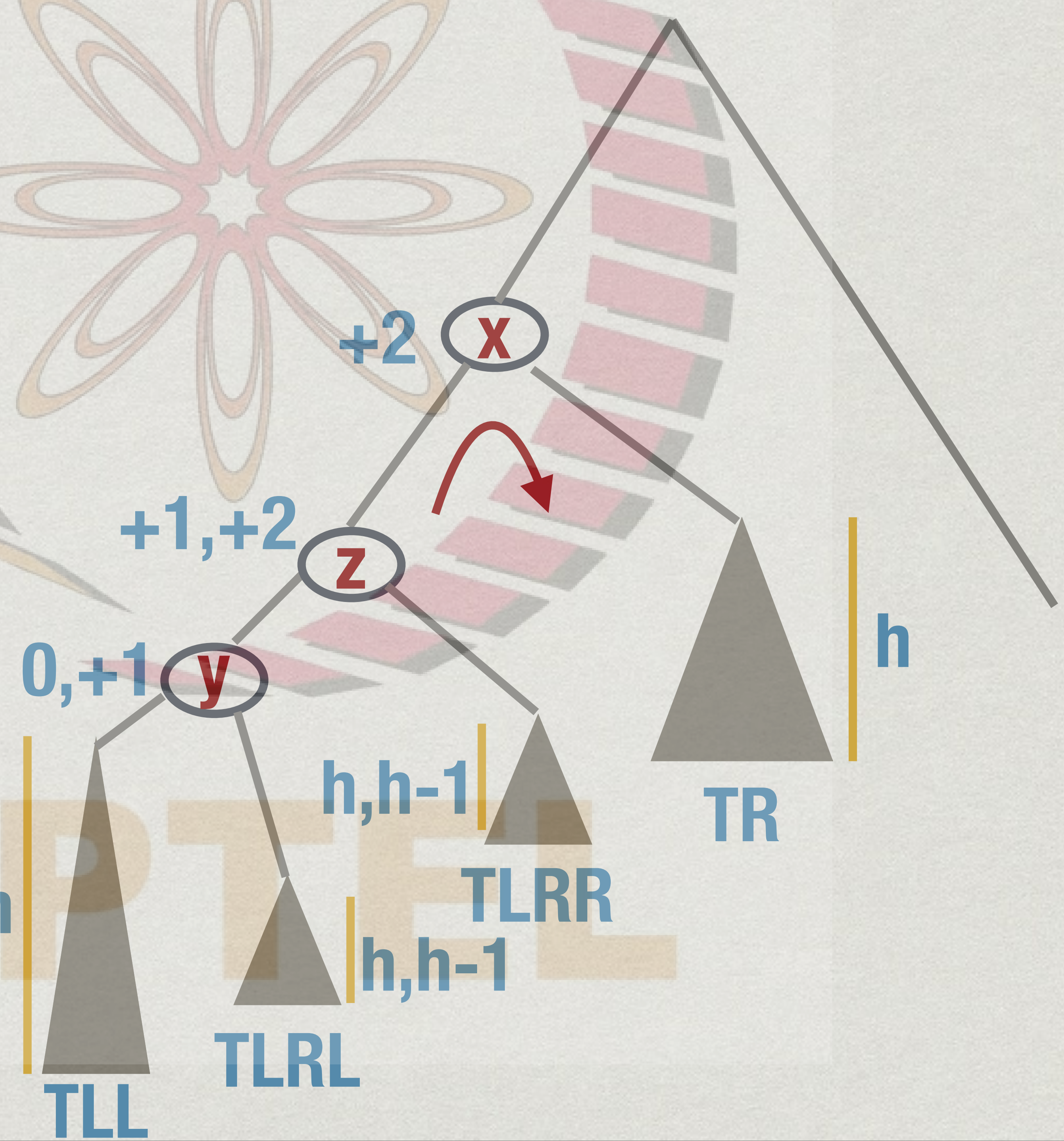
Unbalanced, slope +2

- Case 2: slope of y is {-1}
- Expand TLR
- Rotate left at y



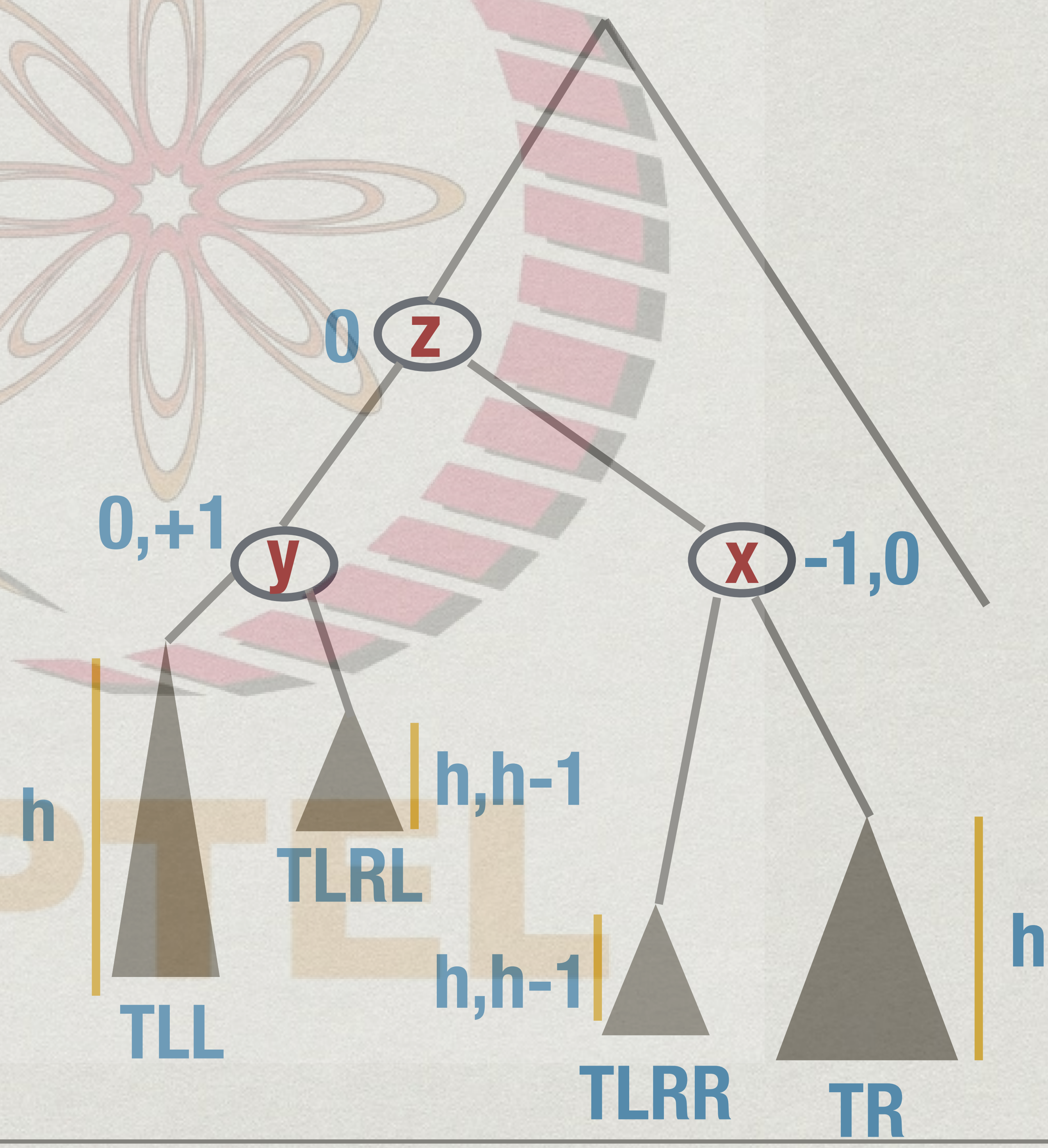
Unbalanced, slope +2

- Case 2: slope of y is {-1}
- Expand TLR
- Rotate left at y
- Rotate right at x



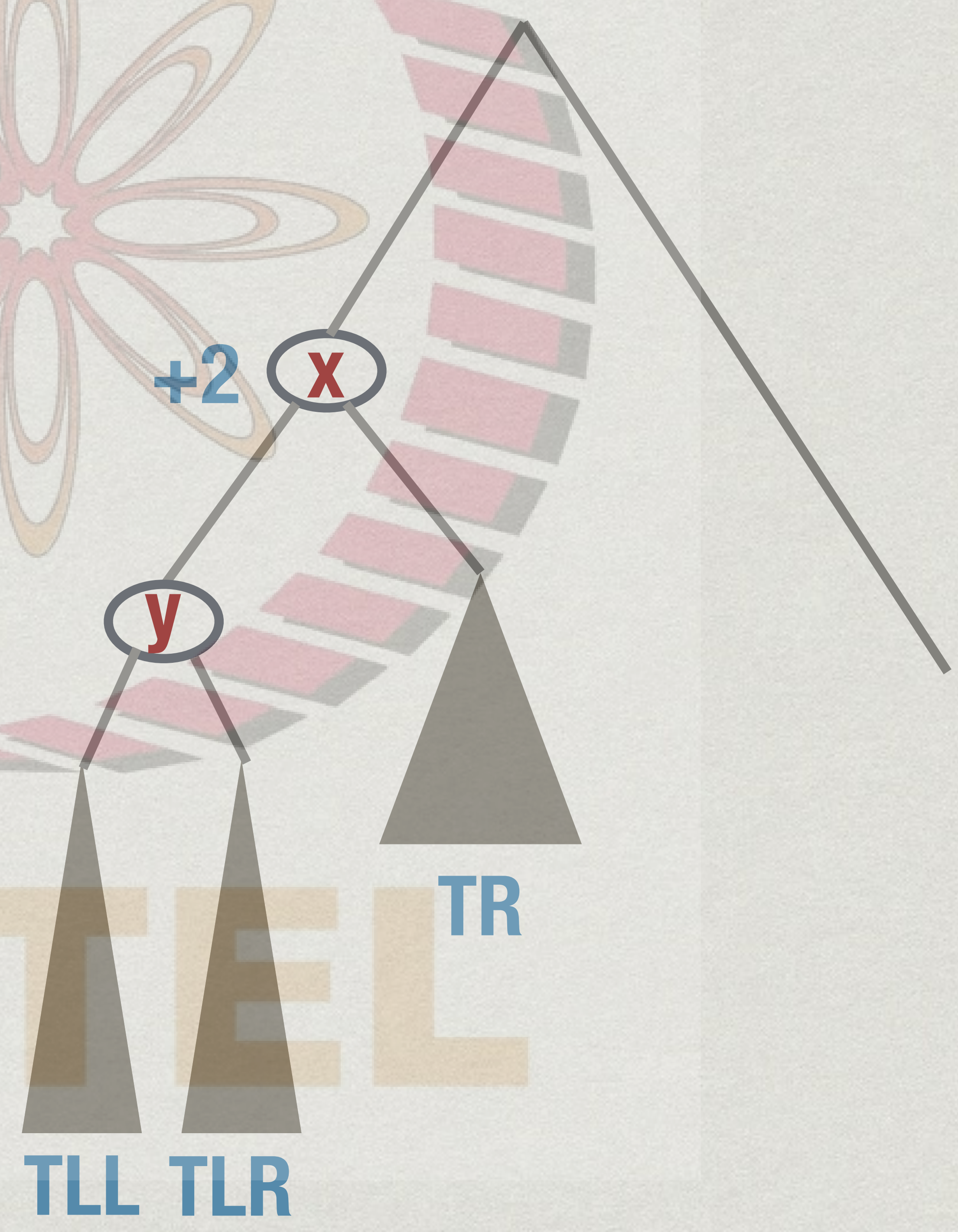
Unbalanced, slope +2

- Case 2: slope of y is {-1}
- Expand TLR
- Rotate left at y
- Rotate right at x



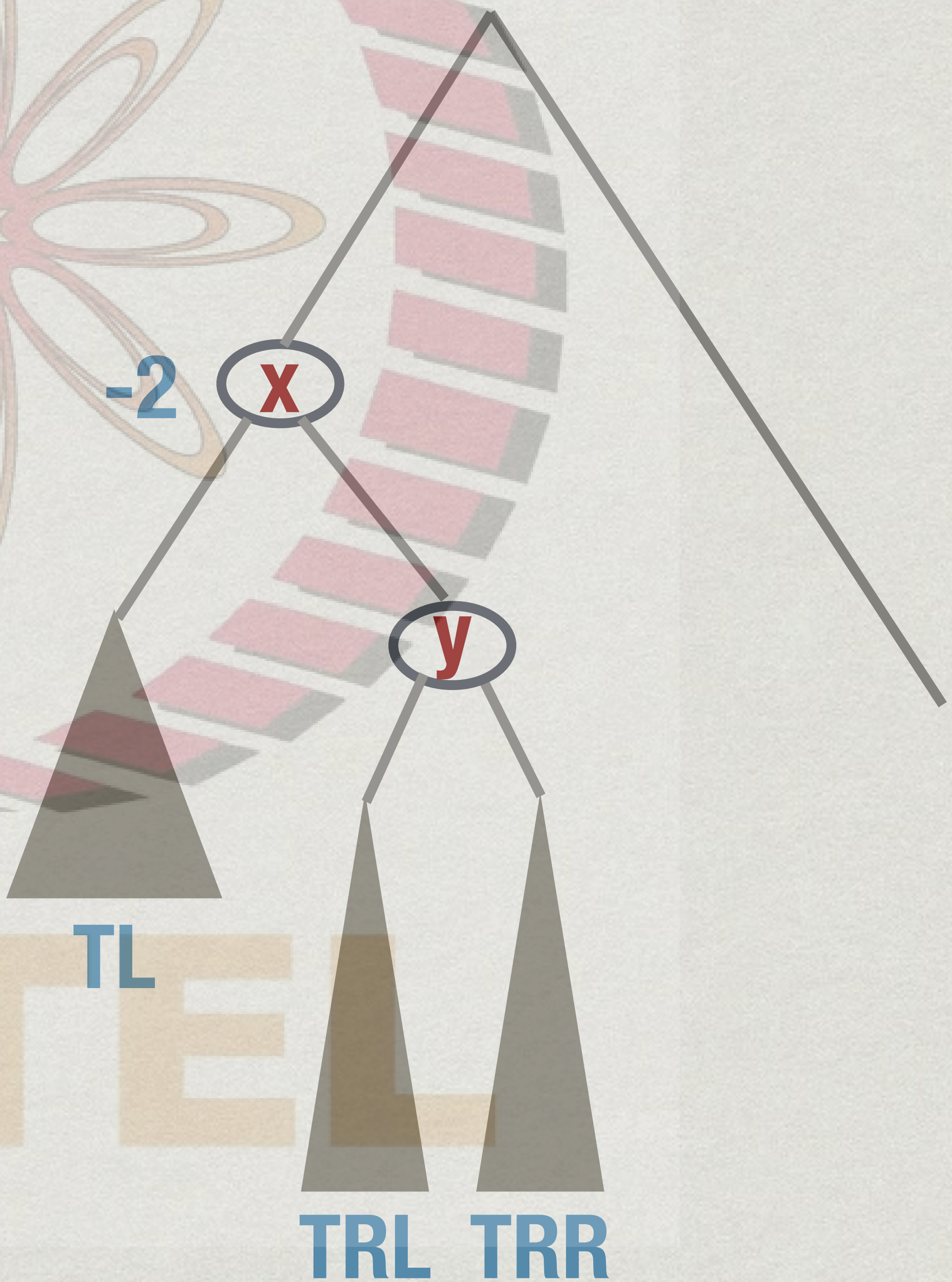
Unbalanced, slope +2

- Case 1:
  slope of y {0,+1}
- Rotate right at x
- Case 2:
  slope of y {-1}
- Rotate left at y
- Rotate right at x



Unbalanced, slope -2

- Case 1:
  slope of y {-1,0}
- Rotate left at x
- Case 2:
  slope of y {+1}
- Rotate right at y
- Rotate left at x

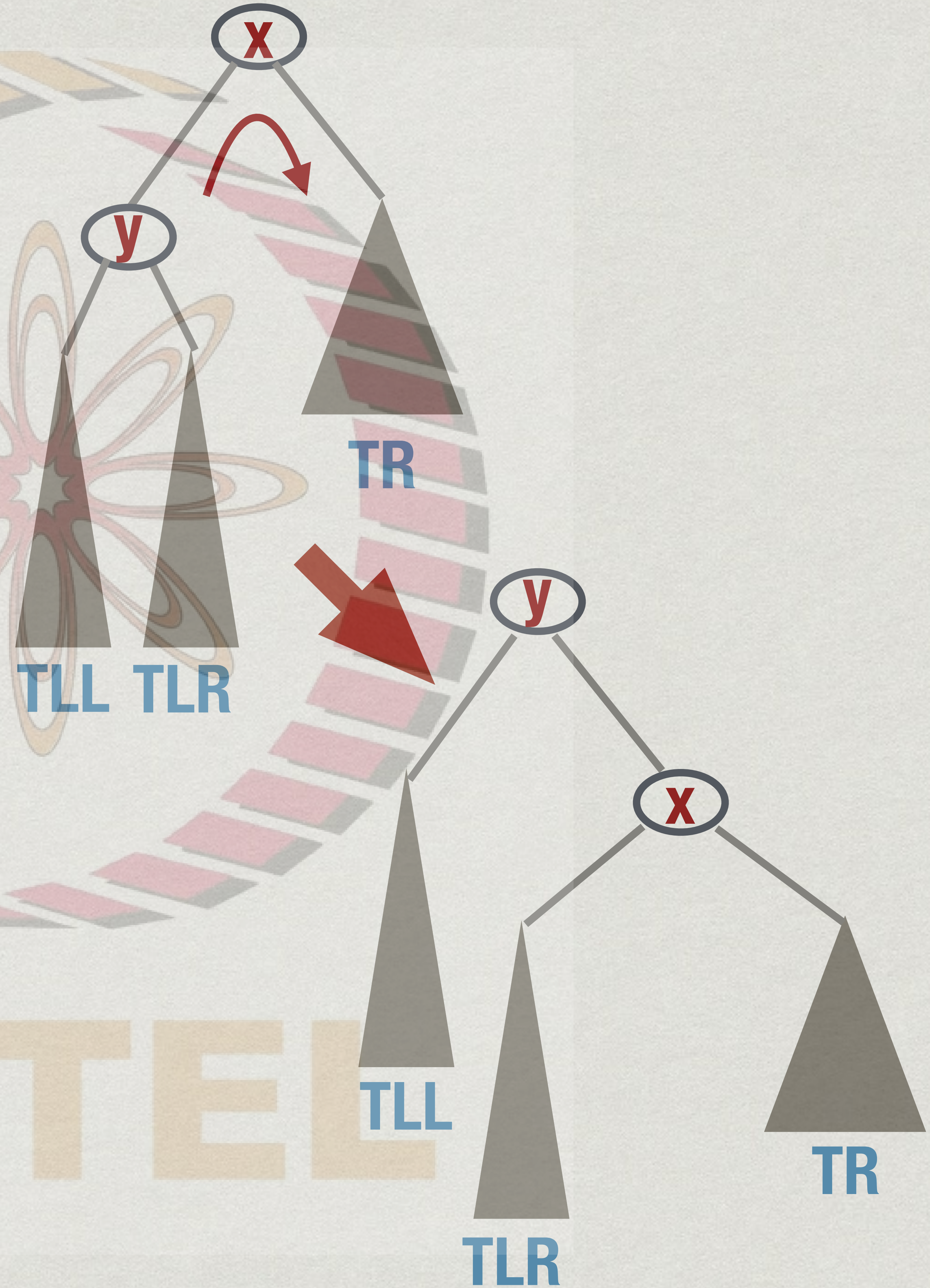


Rotate right

```
function rotateright(t)

x = t.value
y = t.left.value
TLL = t.left.left
TLR = t.left.right
TR = t.right

t.value = y
t.right = t.left
t.right.value = x
t.left = TLL
t.right.left = TLR
t.right.right = TR
```

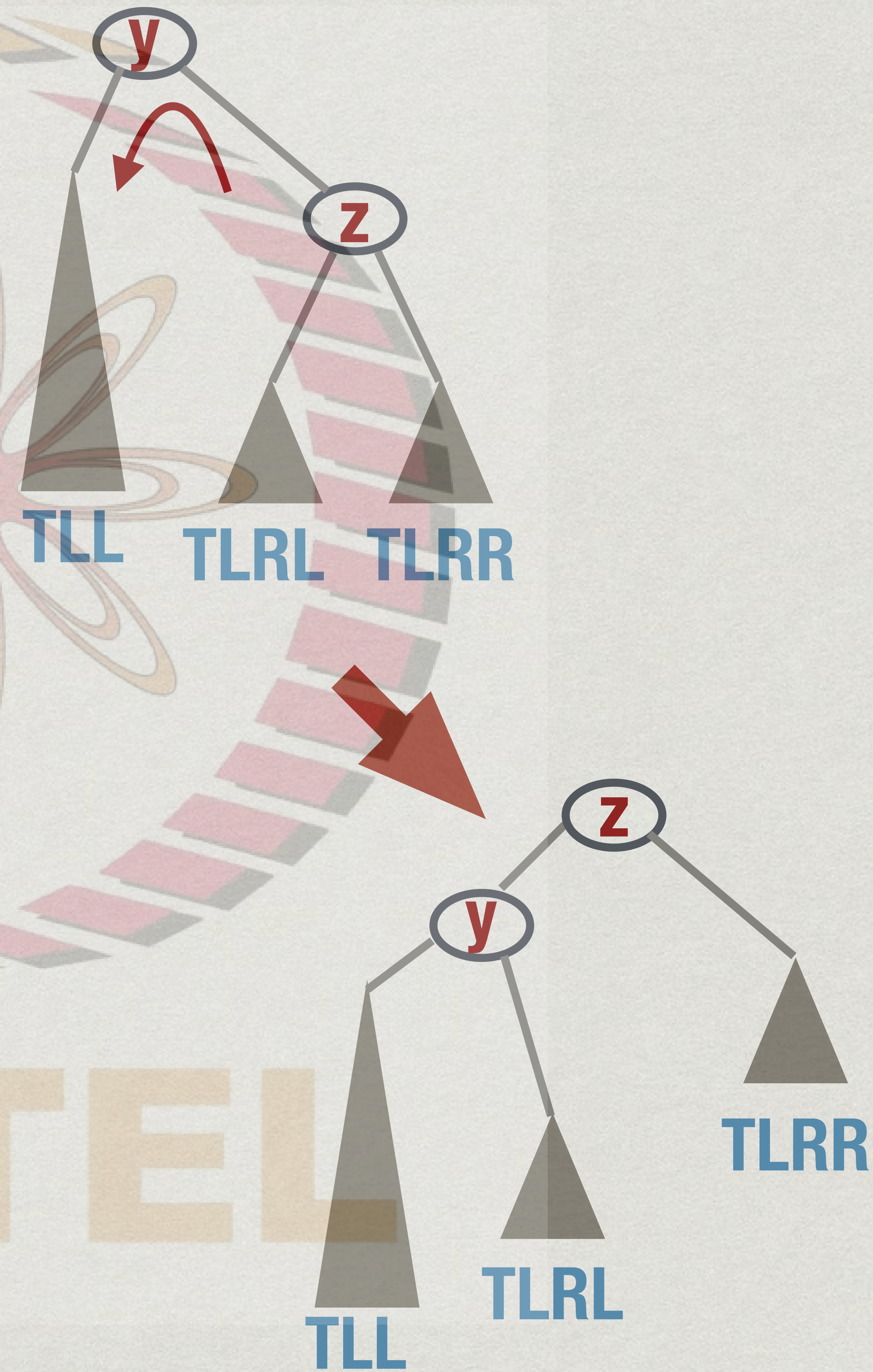


Rotate left

```
function rotateleft(t)

y = t.value
z = t.right.value
TLL = t.left
TLRL = t.right.left
TLRR = t.right.right

t.value = z
t.left = t.right
t.left.value = y
t.left.left = TLL
t.left.right = TLRL
t.right = TLRR
```



Rebalance

```
function rebalance(t)

if (slope(t) == 2)
  if (slope(t.left) == -1)
    rotateleft(t.left)
  rotateright(t)

if (slope(t) == -2)
  if (slope(t.right) == 1)
    rotateright(t.right)
  rotateleft(t)

return
```

Balanced insert(v)

```
function insert(t,v)
. . .

if (v < t.value)
  if (t.left == NIL)
    t.left = Node(v); t.left.parent = t; return
  else
    insert(t.left,v); rebalance(t.left); return
else
  if (t.right == NIL)
    t.right = Node(v); t.right.parent = t; return
  else
    insert(t.right,v); rebalance(t.right); return
```

Balanced delete(v)

```
function delete(t,v)

. . .

# Recursive cases, t.value != v
if (v < t.value)
  if (t.left != NIL)
    delete(t.left,v); rebalance(t.left)
  return

if (v > t.value)
  if (t.right != NIL)
    delete(t.right,v); rebalance(t.left)
  return
```

Balanced delete(v)

```
# Delete node with two children
# Copy pred(v) into current node

pv = pred(v)
t.value = pv

# Delete pv from left subtree
# - pv either leaf or has single child

delete(t.left,pv)
rebalance(t.left)
```

Computing slope

- slope =
  height(left) -
  height(right)
- Can compute height recursively, on demand
- Takes time O(n)!
  - Needs to traverse entire tree!

```
function height(t)

if (t == NIL)
  return(0)

return(
  1 +
  max(
    height(t.left),
    height(t.right))
  )
```

Computing slope

- Instead, maintain additional value t.height in each node
- Update t.height with each insert or delete
- Computing slope is now O(1)

```
function insert(t,v)
 . . .
 else
   insert(t.left,v);
   rebalance(t.left);
   t.height = 1 +
     max(
       t.left.height,
       t.right.height
     )
```

Summary

- Using rotations we can maintain height balanced binary search trees
- All operations on search trees then take O(log n) time