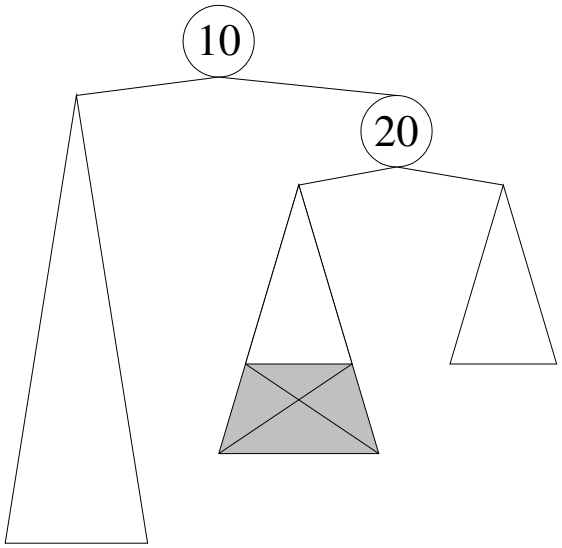
COMP 282

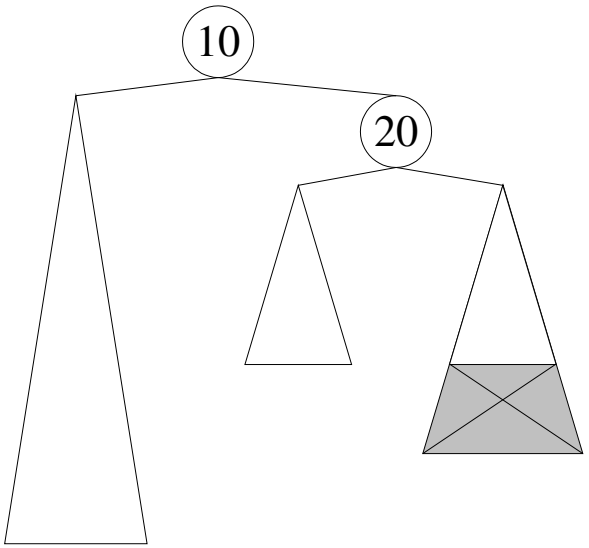
AVL Deletion

Trees can only decrease like so...

- Deletion may cause a reduction in height.

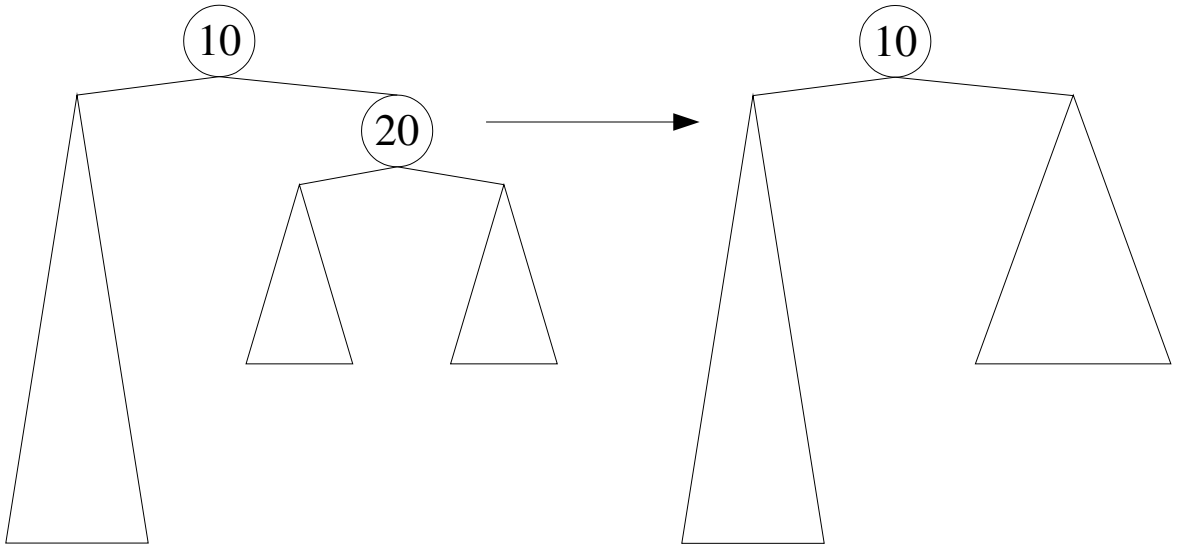


Item deleted from
Inner subtree



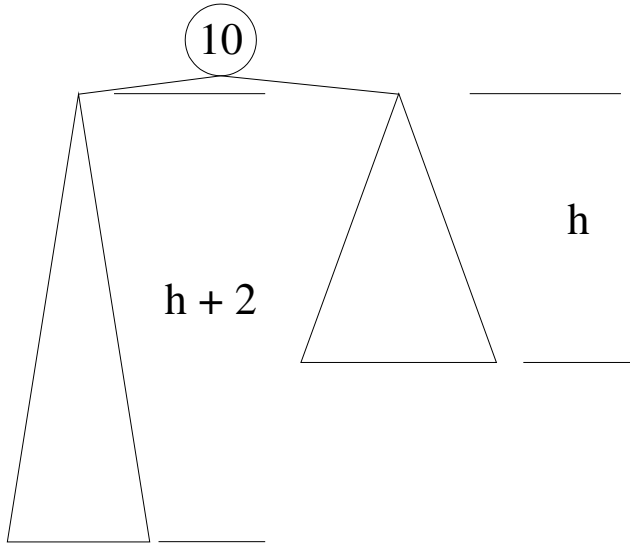
Item deleted from
Outer subtree

- In either case the shortened subtree has equal height subtrees itself.
- View this subtree as just a single subtree



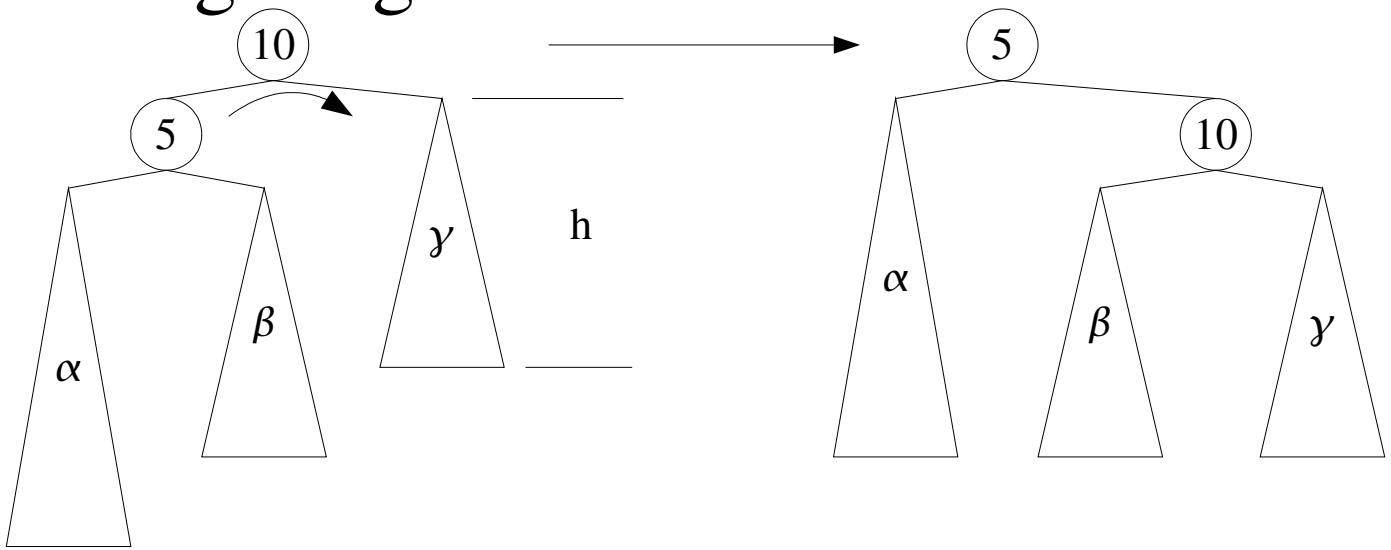
How does this help?

- We can apply the general “Is this balanced?” test to the root node. Just as is done during insertion.

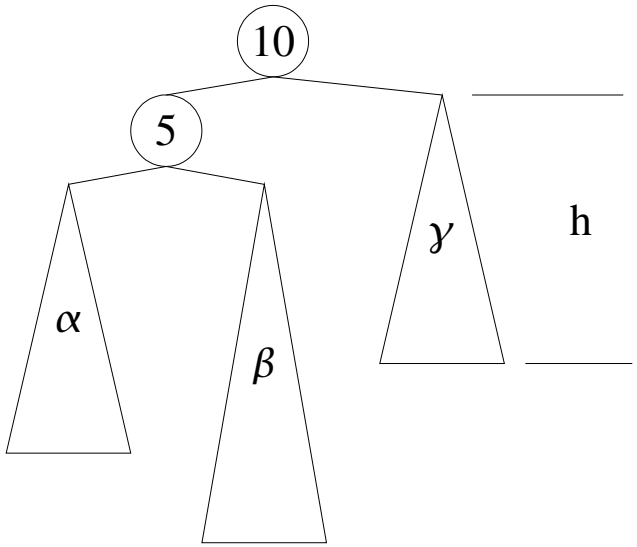


Look at left subtree in detail

- The case where the left subtree's height is determined by the left-left (or “outer”) tree...
- A single right rotation is sufficient.

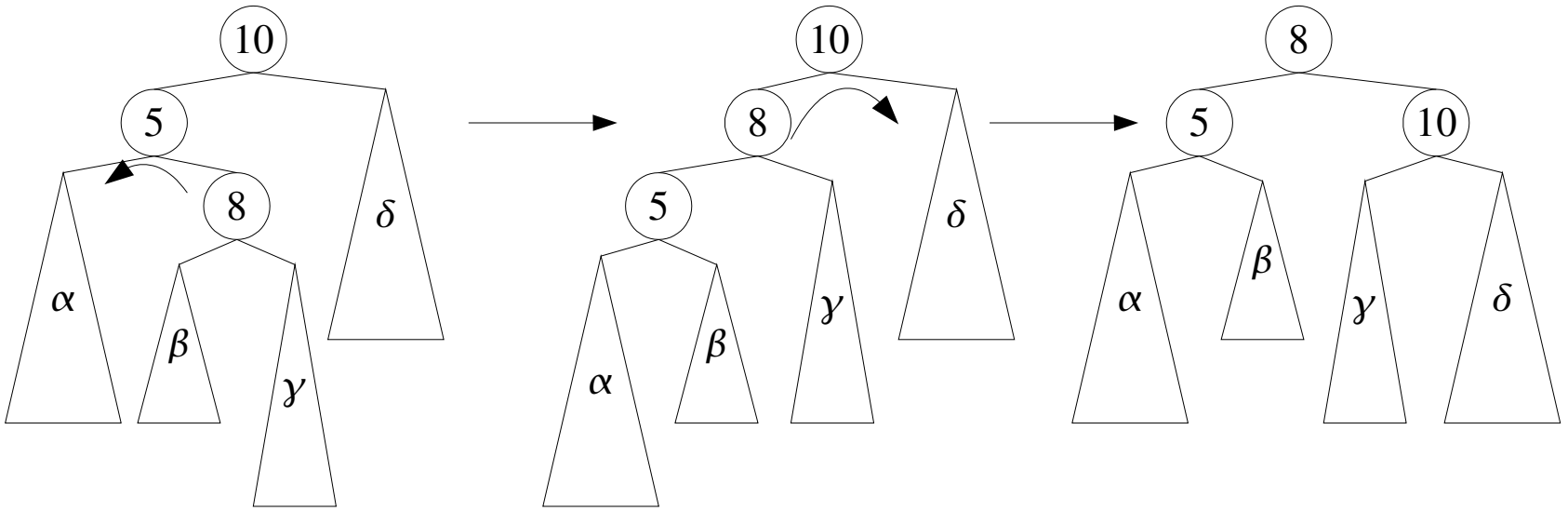


- The case where the left subtree's height is determined by the “inner” subtree.



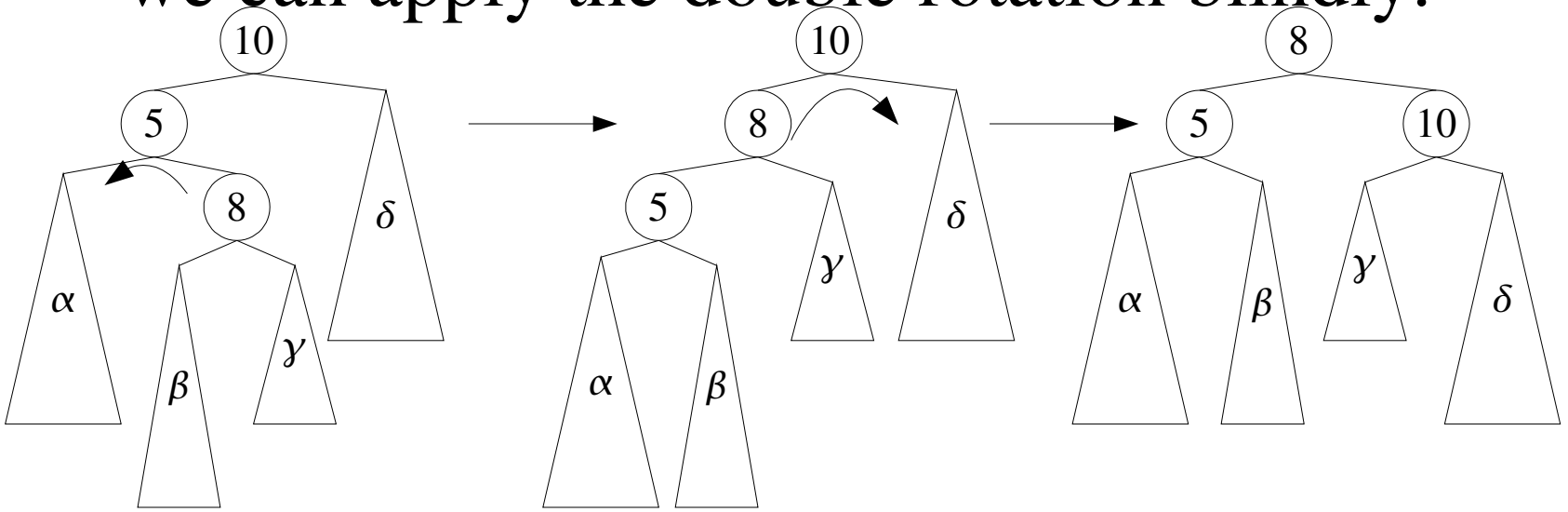
γ determines height at 8

- A left rotation of the subtree followed by a right rotation restores balance.



β determines height at 8

- Again a left rotation of the subtree followed by a right rotation restores balance. Therefore we can apply the double rotation blindly.



Hmm...

- These are the exact same conditions as we identified for imbalances that occur during insertion.
- balance can be restored using the exact same rotation actions that were used to restore balance caused by insertion.
- There is no difference.

- Programming wise then we can apply the same logic as the last step of recursion as we back out of the tree.

- JAVA doesn't tolerate null “this” references very well and we have to examine the height of possibly empty subtrees regularly.
- method parameters can be null however so it helps to create a private method that can be passed a (possibly null) node reference and determines the appropriate height.
- Its not the best Object Oriented design but it is too convenient to ignore.
- ```
  private int determineHeight(AVLNode n)
  {
     if (n == null)
        return 0;
     else
        return n.height;
  }
  ```
  ```

- ```
  private int selectBalanceAction()
  {
     int diff = determineHeight(left) - determineHeight(right);
     int subdiff;

     if (diff < -1)     // right subtree is too tall
        if ((determineHeight(right.left) -
             determineHeight(right.right)) < 0) // outer tall
           return 2; // outer subtree already taller
        else
           return 4; // inner subtree is the problem
     else if (diff > 1) // left subtree is too tall
        if ((determineHeight(left.left) -
             determineHeight(left.right)) > 0) // outer tall
           return 1; // outer subtree already taller
        else
           return 3; // inner subtree is the problem
     else
        return 0; // no rebalance necessary
  }
  ```
  ```

- ```
  private void rebalance()
  {
     switch (selectBalanceAction())
     {
        case 3:
           left.rotateLeft();
           left.recomputeHeight();
        case 1:
           rotateRight();
           recomputeHeight();
           break;
        case 4:
           right.rotateRight();
           right.recomputeHeight();
        case 2:
           rotateLeft();
           recomputeHeight();
        case 0: // don't really have to do anything
           break;
     }
  }
  ```
  ```