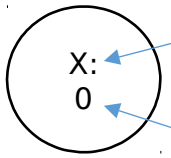
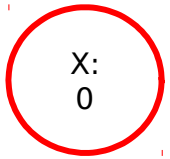
Diagrams to illustrate the cases referred to in the avl.c, wavl.c, twothree.c and twofour.c source files. These are the tree type balancing algorithms.

KEY

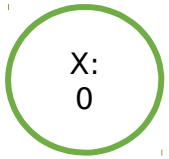


Name of node for demonstration purposes only

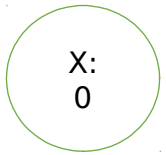
Rank difference of node



Bold red node indicates violation of rank rule

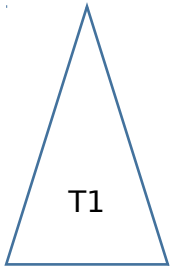


Bold green node indicates state of node tested in balancing loop



Not bold green node indicates an alternate state of node tested in balancing loop if swapped with bold green node

Black rectangular node is a missing or sentinel node. I have used a sentinel node in my code



Triangular node indicates any sub-tree including a sentinel node

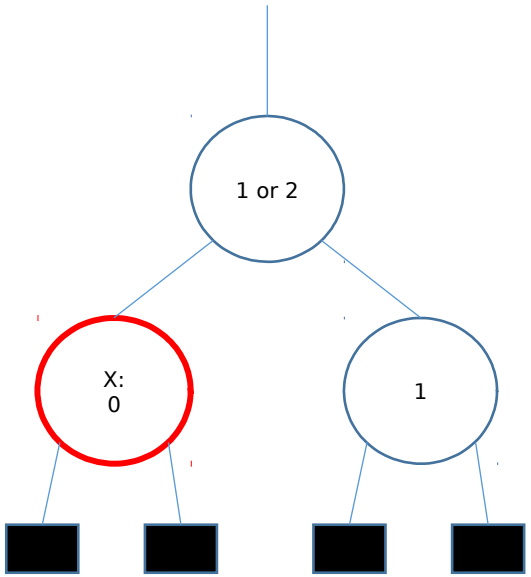
NOTES

- AVL and WAVL insertion balance are identical
- WAVL has one extra balancing case compared to AVL
- WAVL and AVL differ in two deletion cases only because the AVL must continue checking for further imbalance up the tree due to making 2,2-nodes into 1,1-nodes

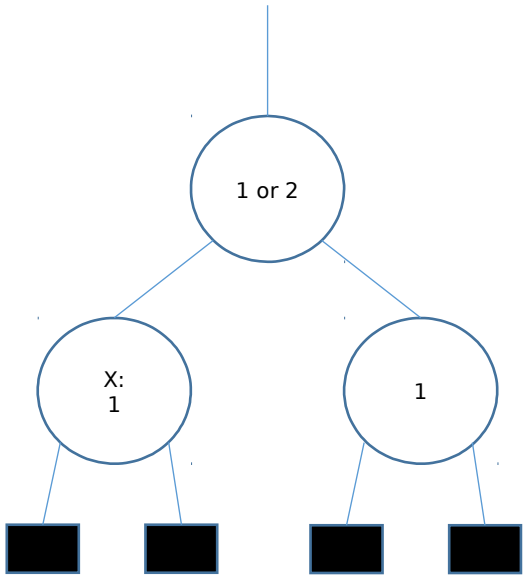
- 2-3 and 2-3-4 tree deletion re-balancing is identical
- 2-3 and 2-3-4 insertion is almost identical in every case. The 2-3-4 tree generally examines one extra node to determine what case to apply

- READ THE README.TXT FOR AN EXPLANATION ON HOW TO USE THE TREE PROGRAM
- AN EXAMPLE DATA INPUT.TXT HAS BEEN PROVIDED

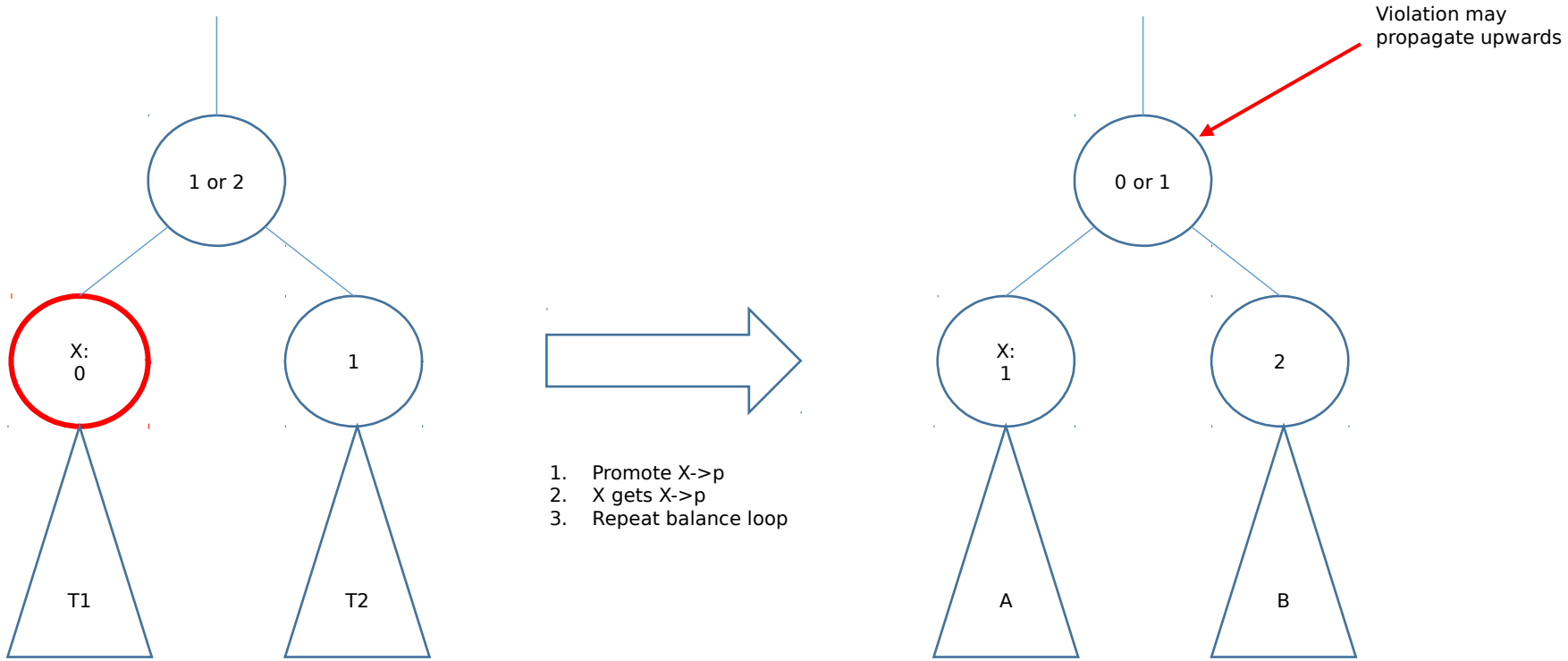
AVL/WAVL insertion – Case 0



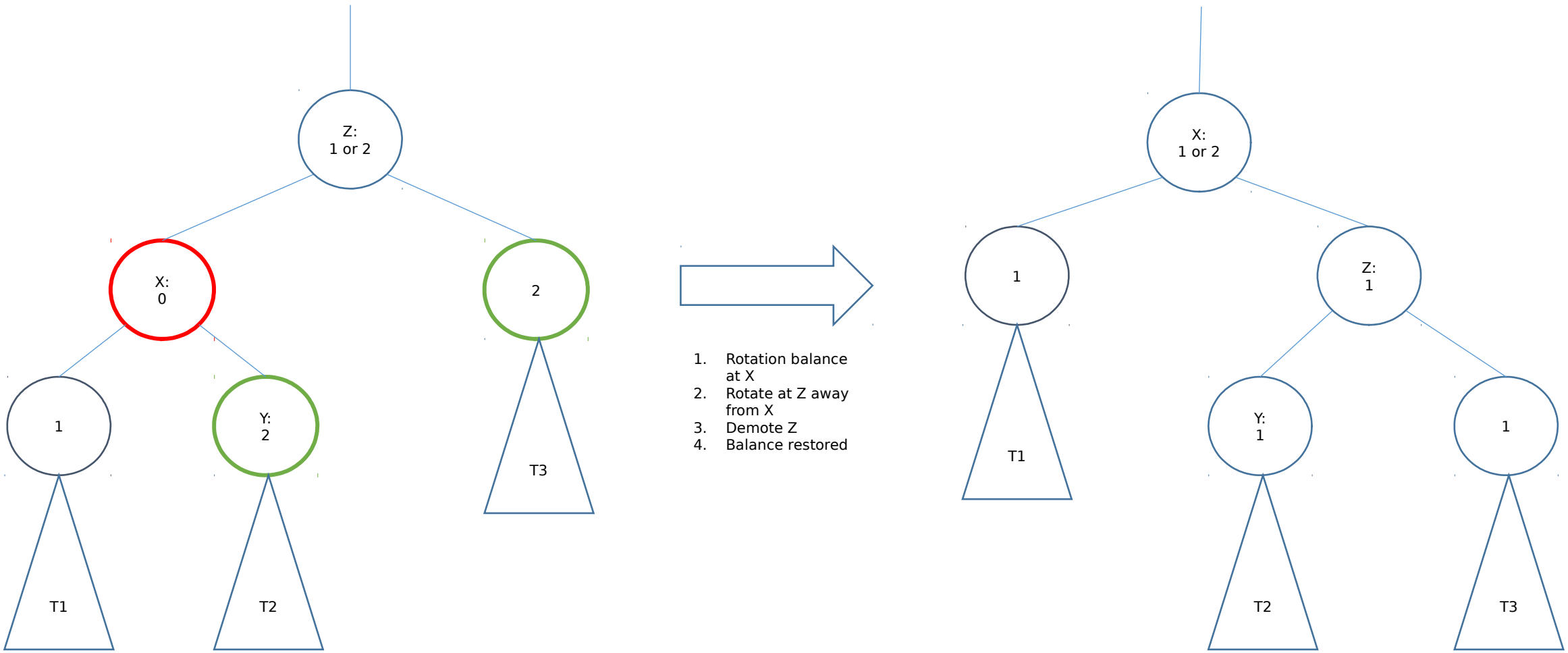
1. If X is added to a unary node, promote X and balancing is complete



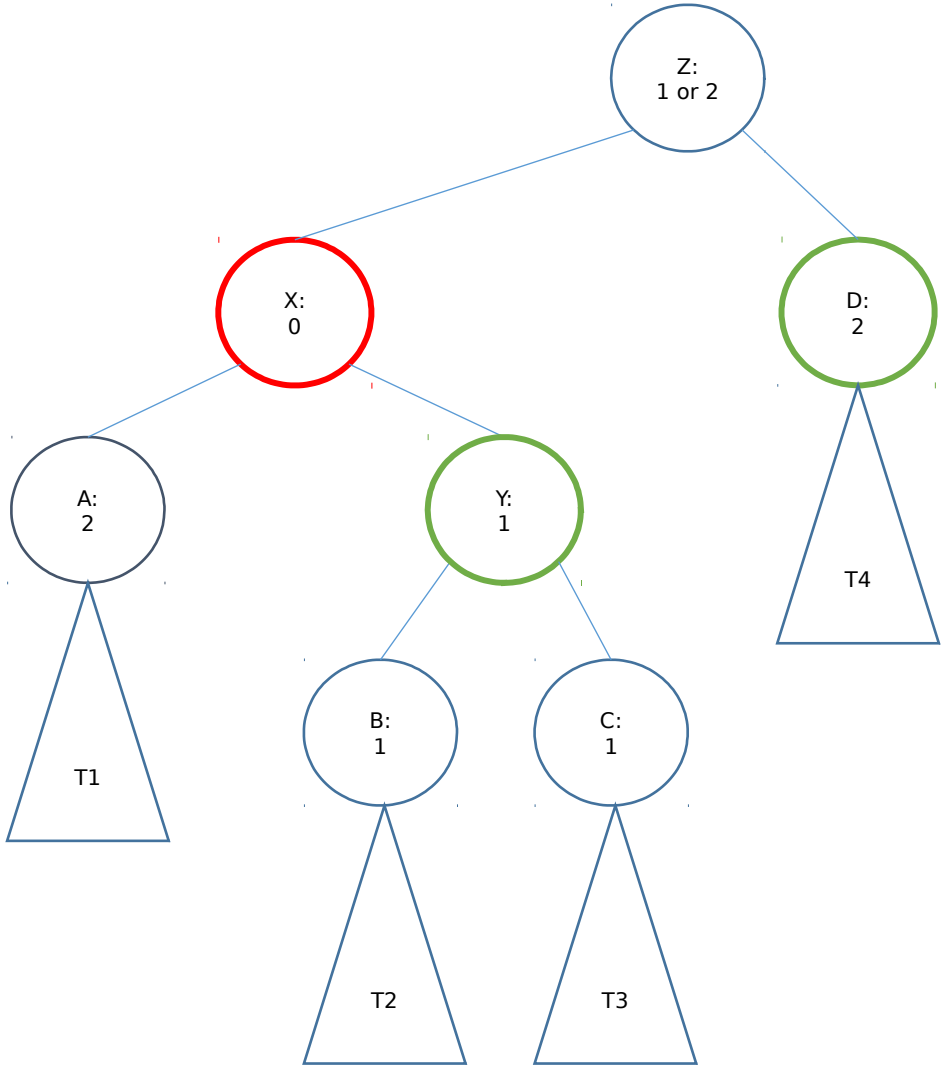
AVL/WAVL insertion – Case 1



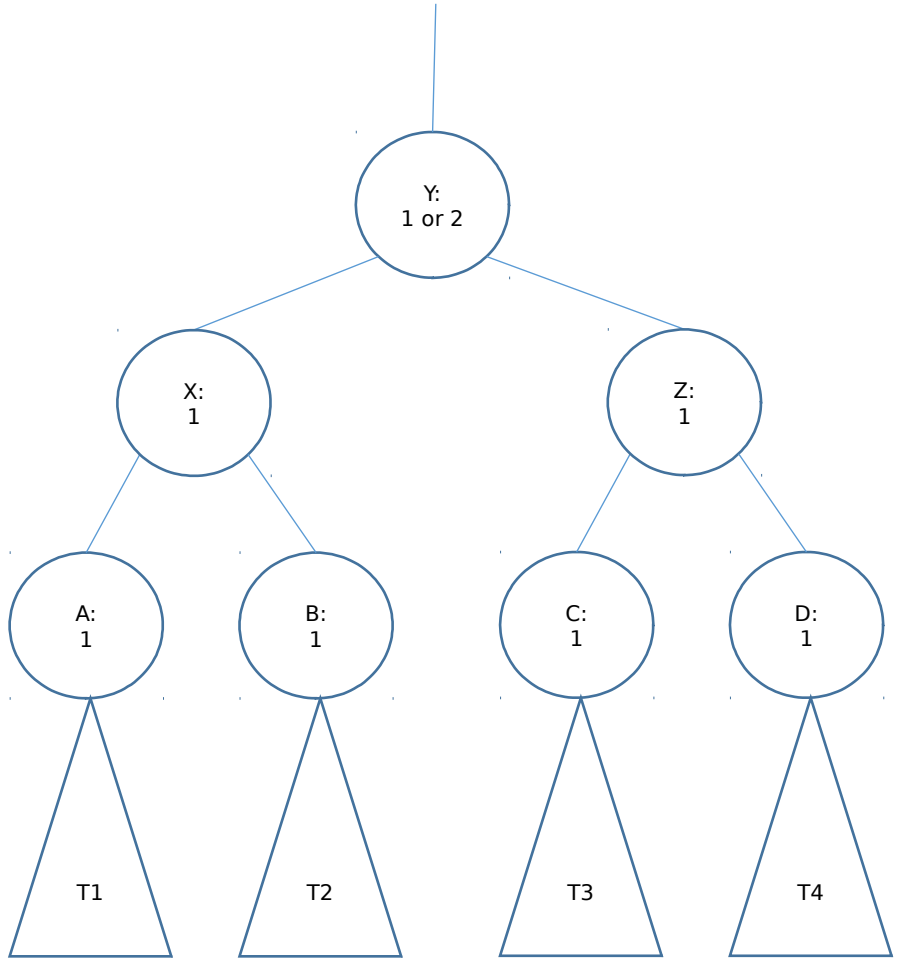
AVL/WAVL insertion – Case 2



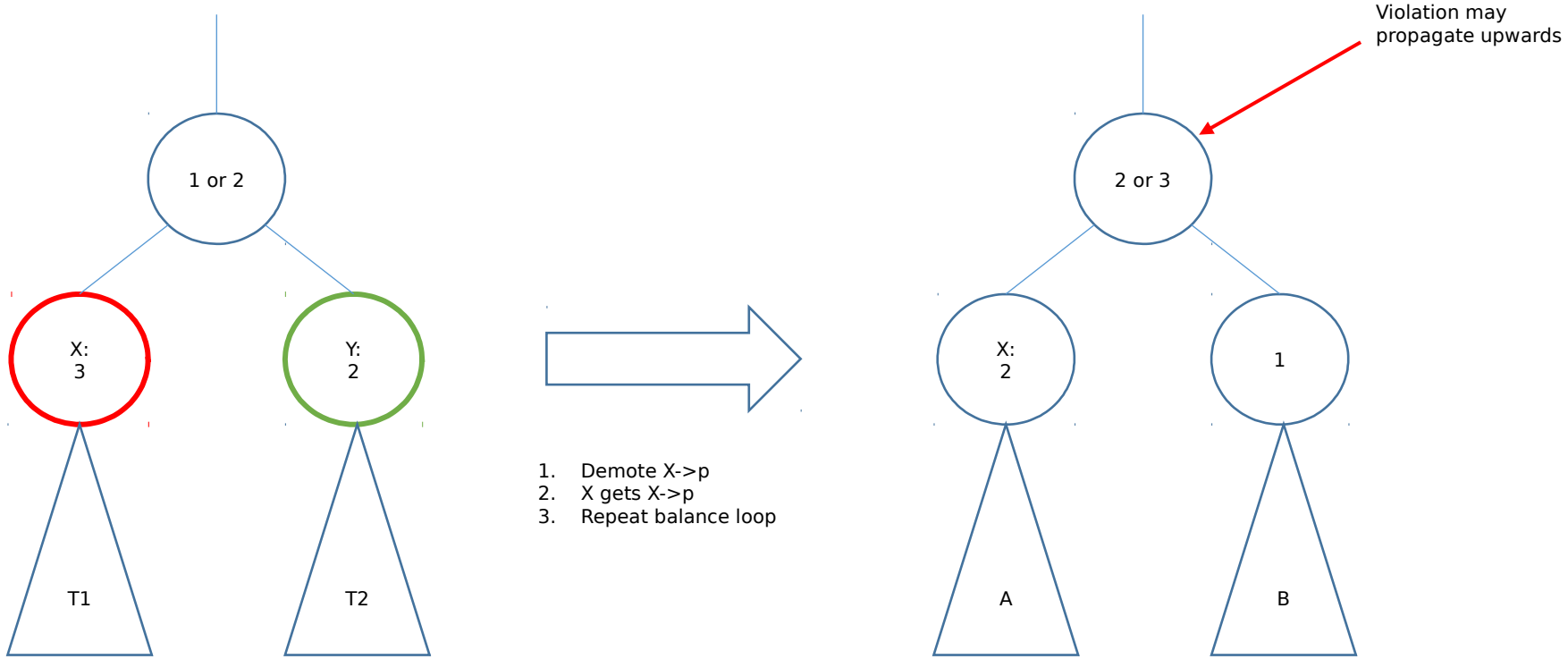
AVL/WAVL insertion – Case 3



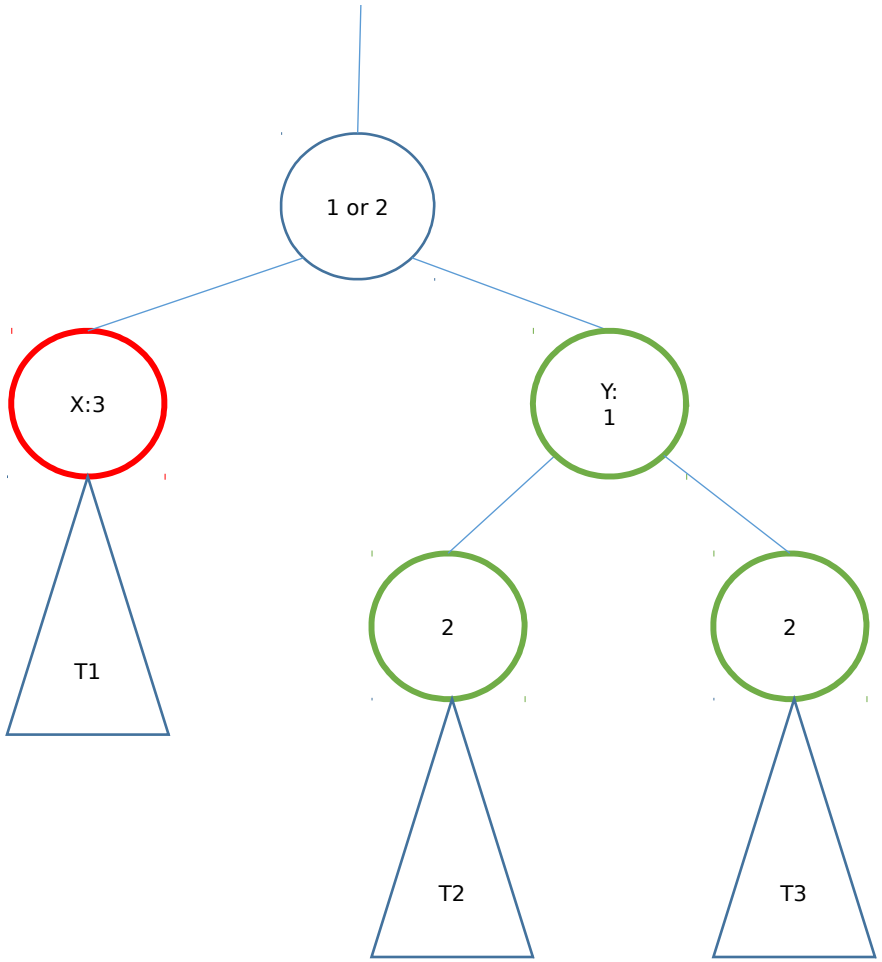
1. Rotation balance at Y
2. Rotate at X away from Y
3. Rotation balance at Y
4. Rotate at Z away from Y
5. Demote Z
6. Balanced Restored



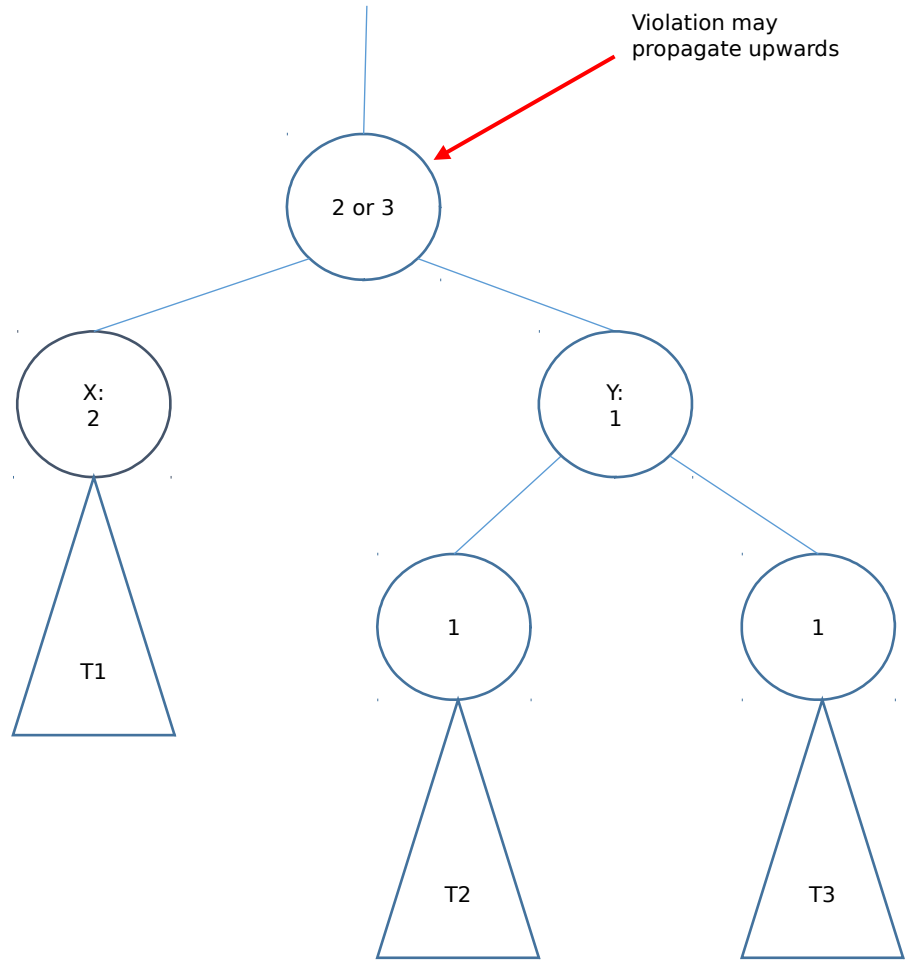
AVL/WAVL Deletion – Case 1



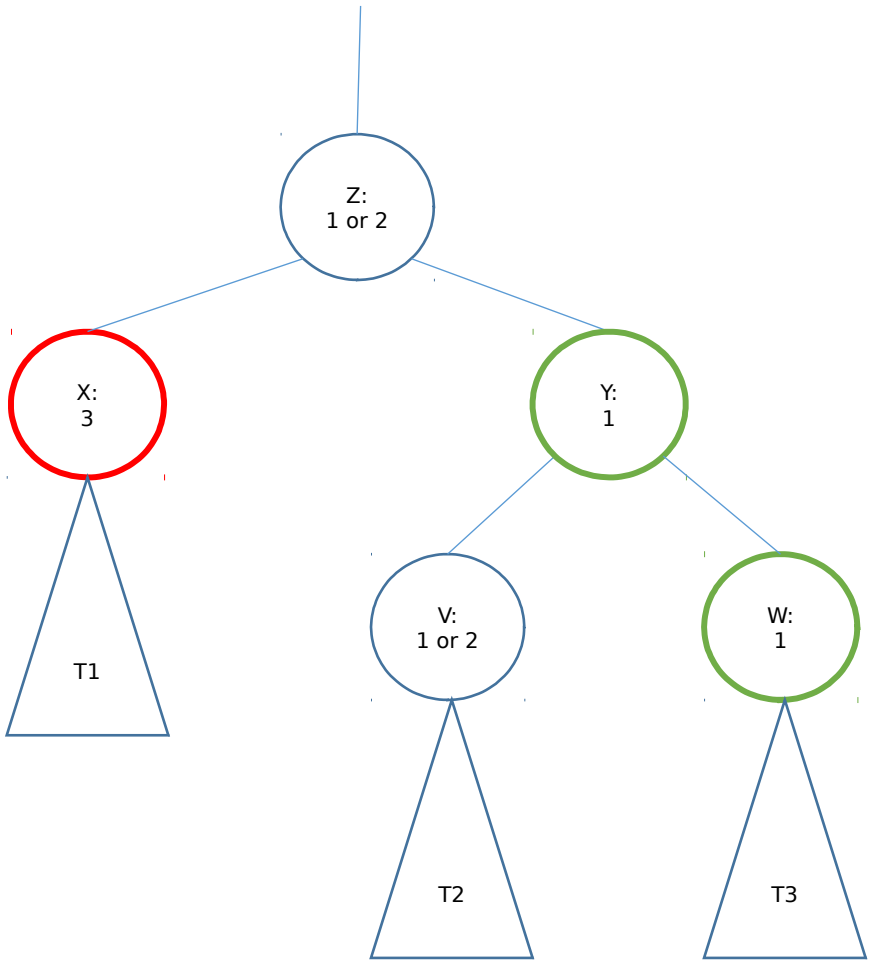
WAVL ONLY deletion – Case 1a



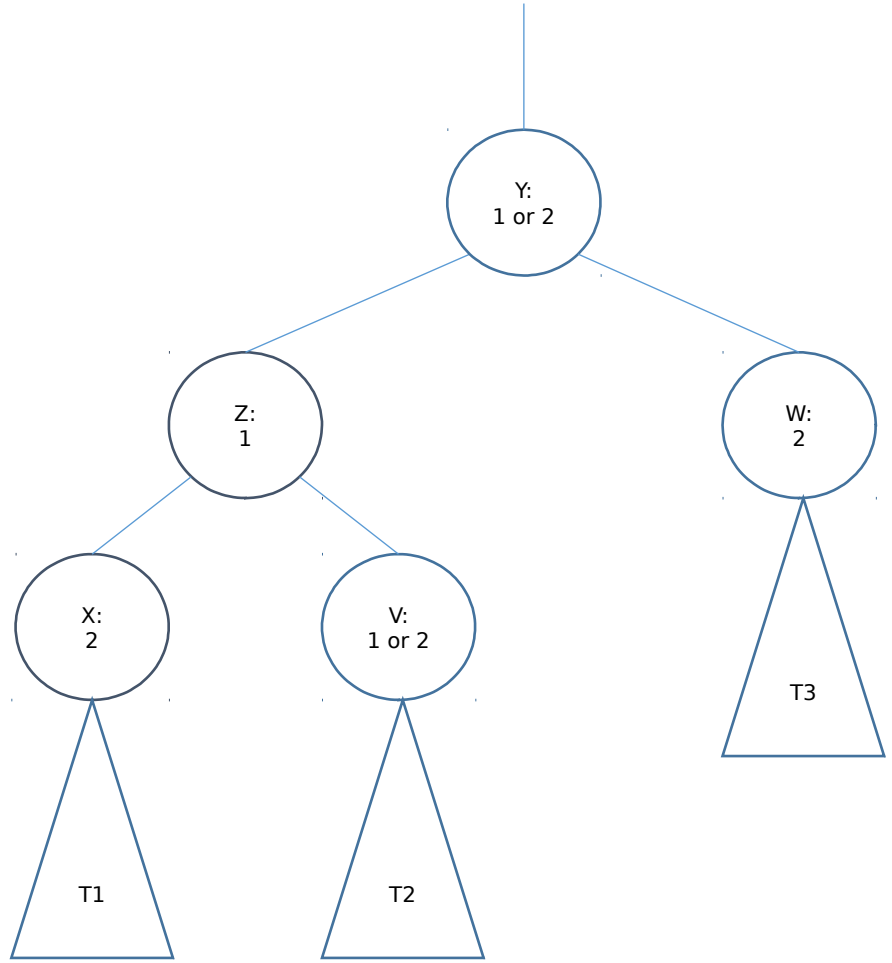
1. Demote Y
2. Demote X->p
3. X gets X->p
4. Repeat balance loop



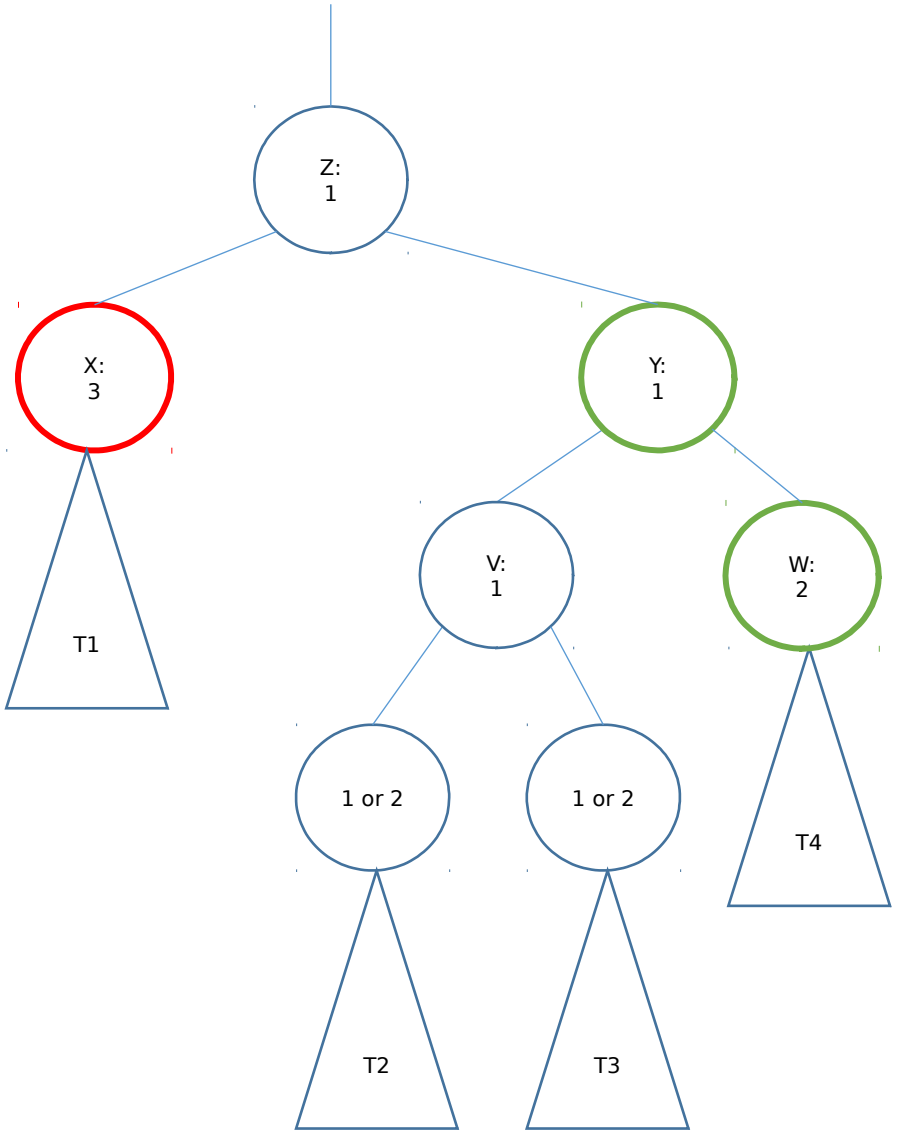
AVL/WAVL deletion – Case 2



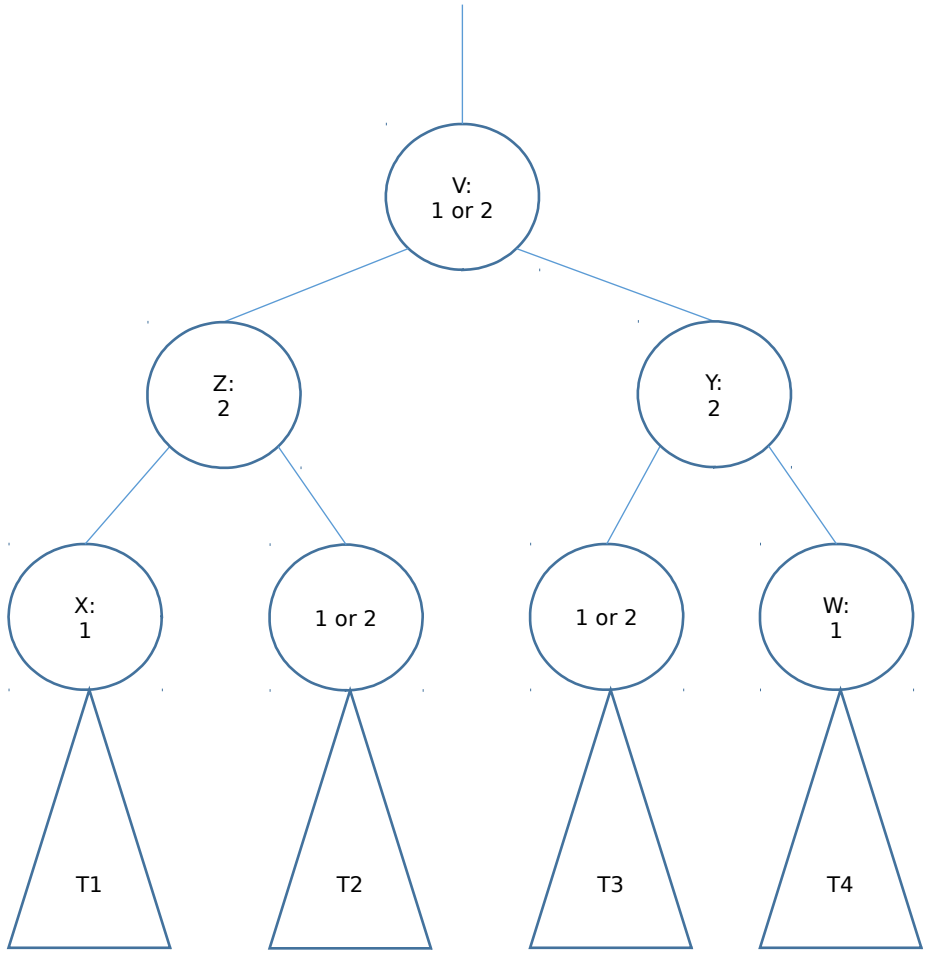
1. Rotation balance at Y
2. Rotate at Z away from Y
3. WAVL - If Z is a leaf, demote Z
4. **AVL** - If Z is a 2,2-node then demote Z.
5. **WAVL** - Balance restored



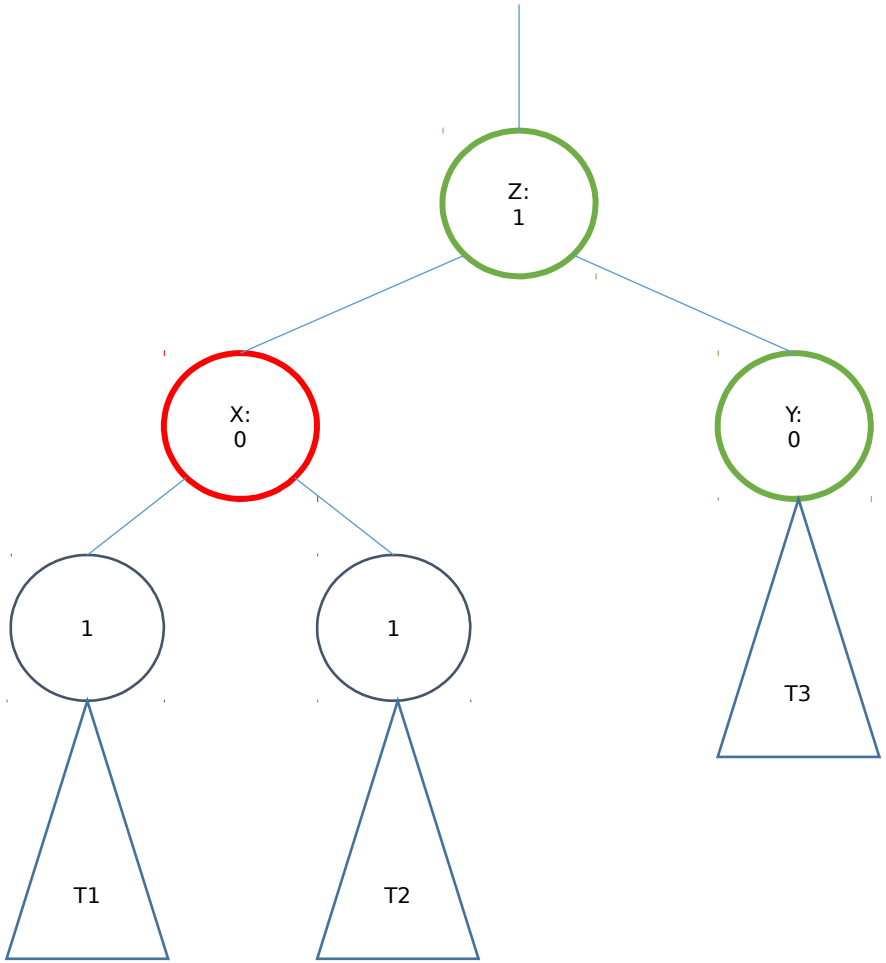
AVL/WAVL deletion – Case 3



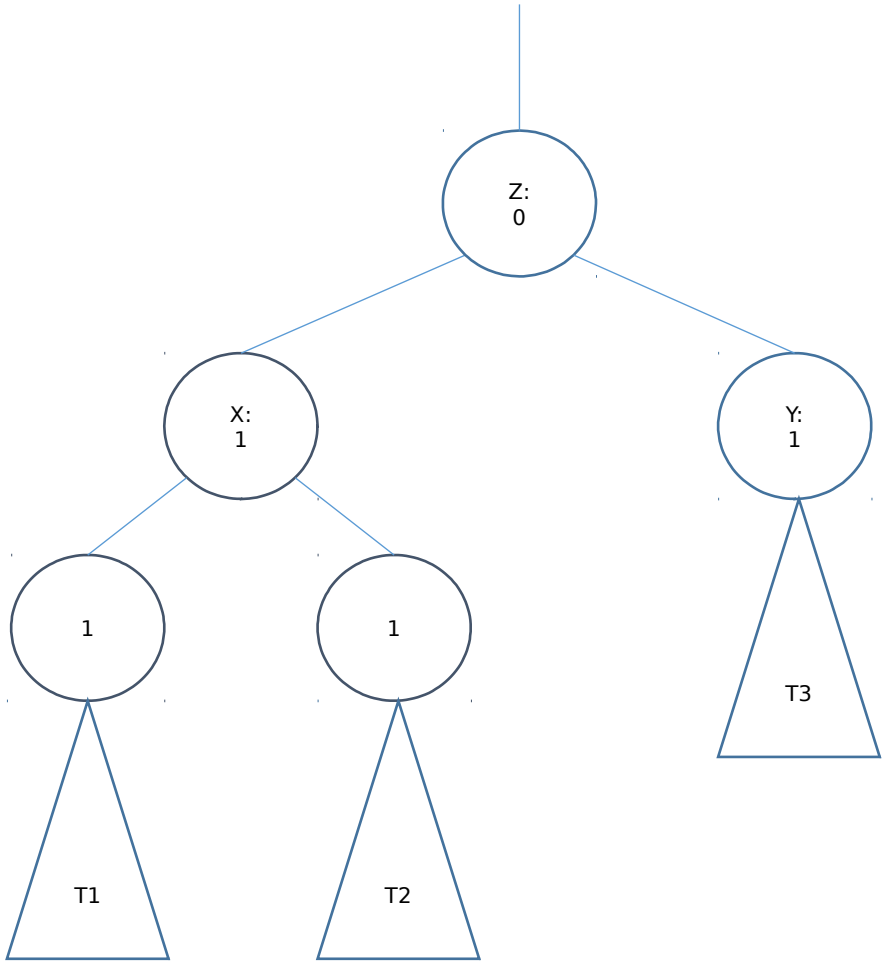
1. Rotation balance at V
2. Rotate at Y away from V
3. Rotation balance at v
4. Rotate at Z away from V
5. Demote Z
6. **AVL** – Demote V and continue balancing loop
7. **WAVL** – Balance restored



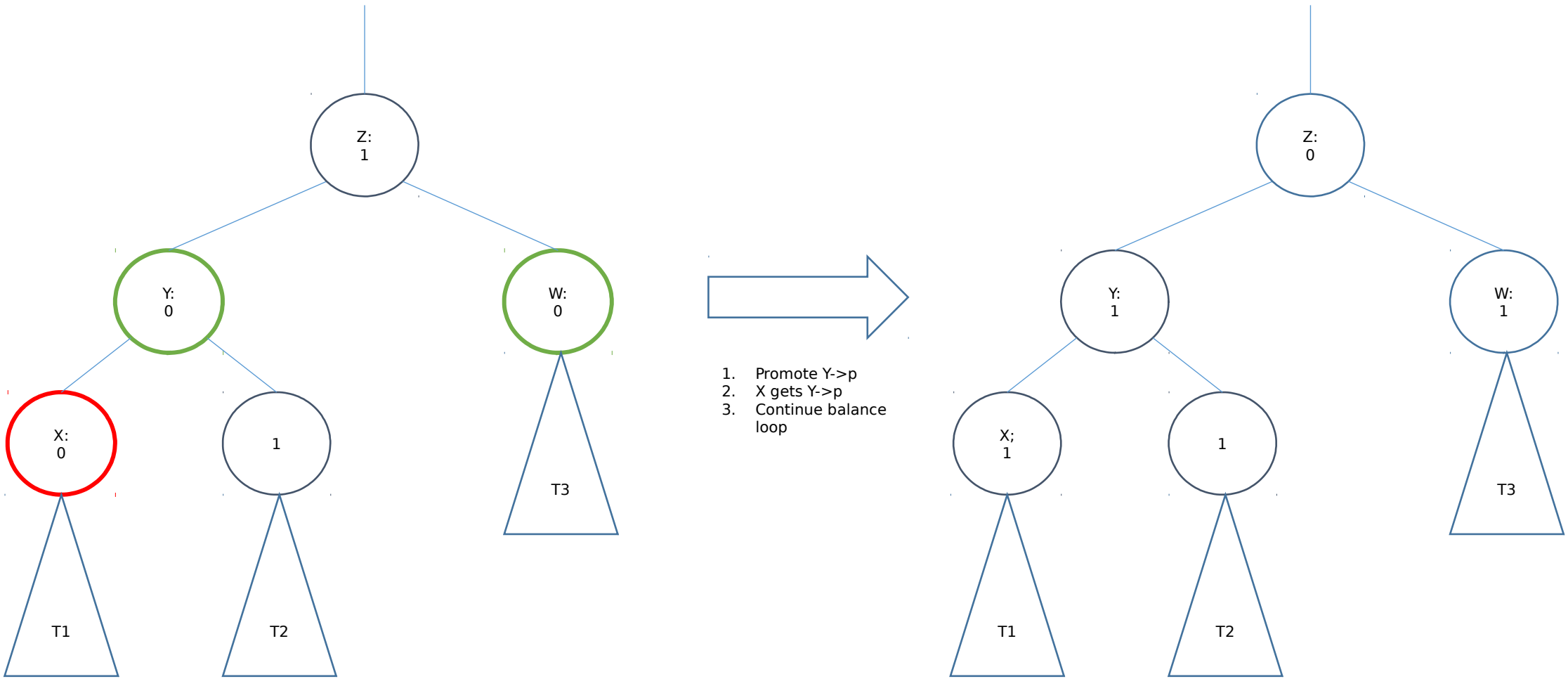
2-3 insertion – Case 1



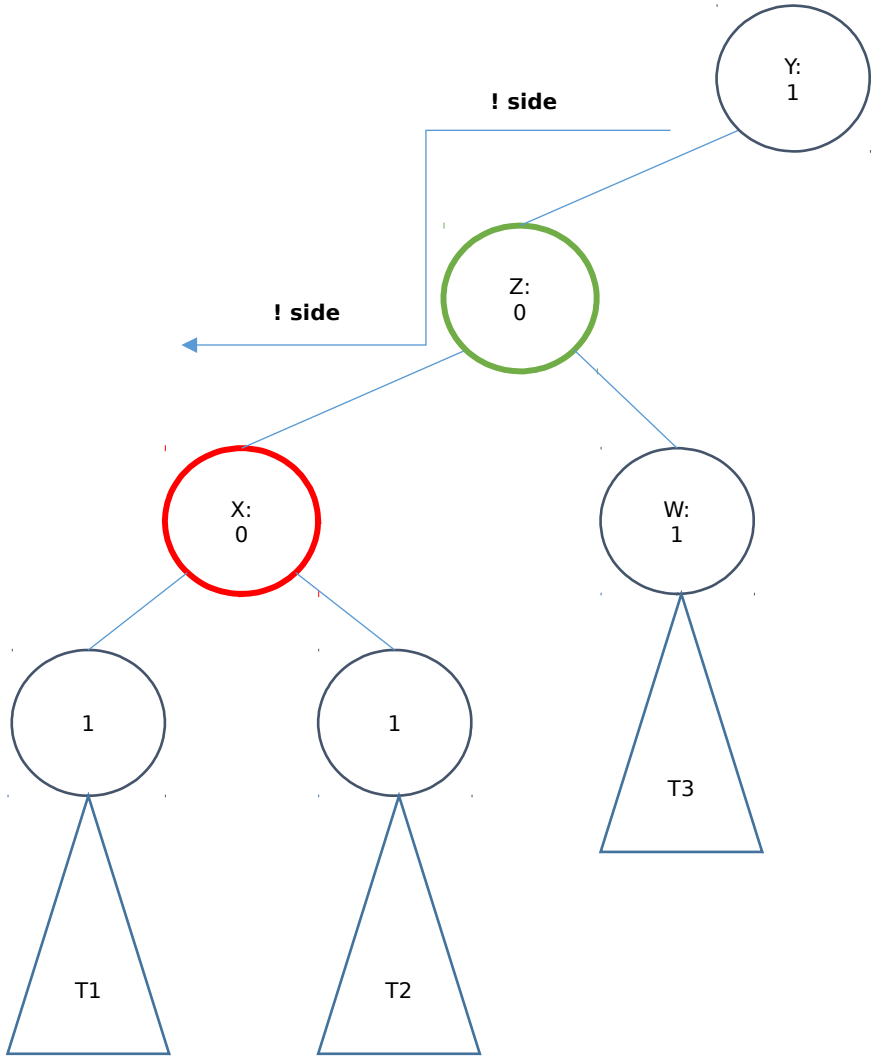
1. Promote X->p
2. X gets X->p
3. Continue balance loop



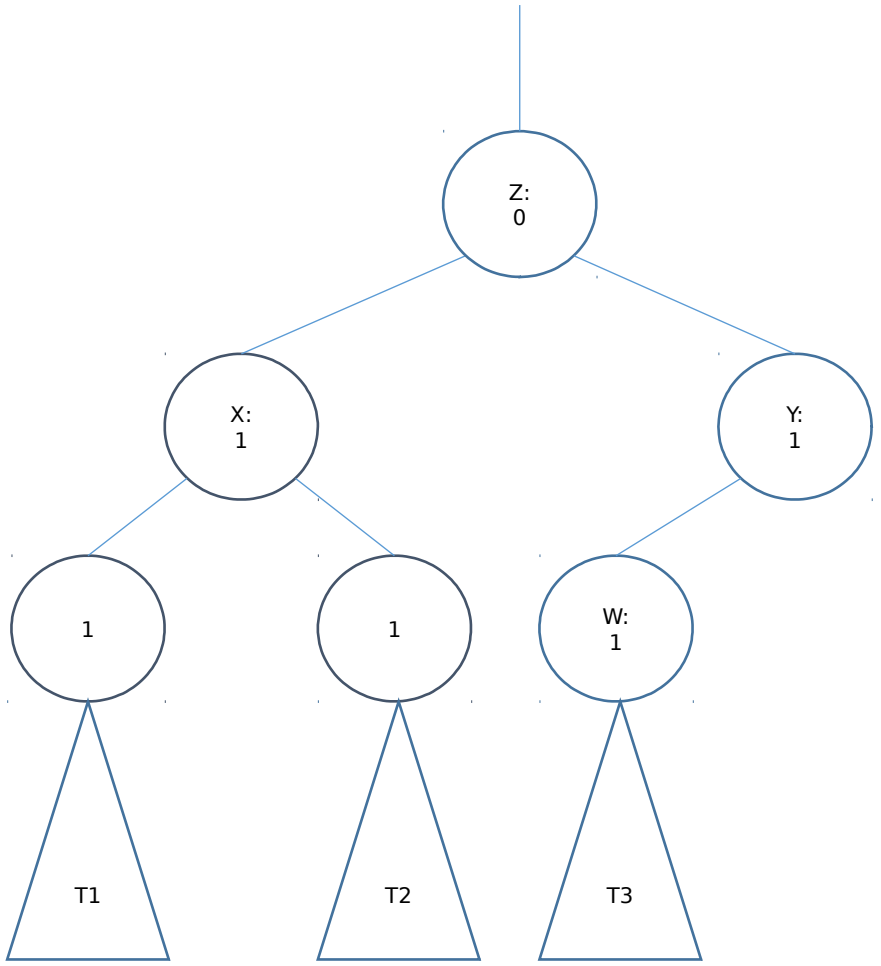
2-3-4 insertion – Case 1



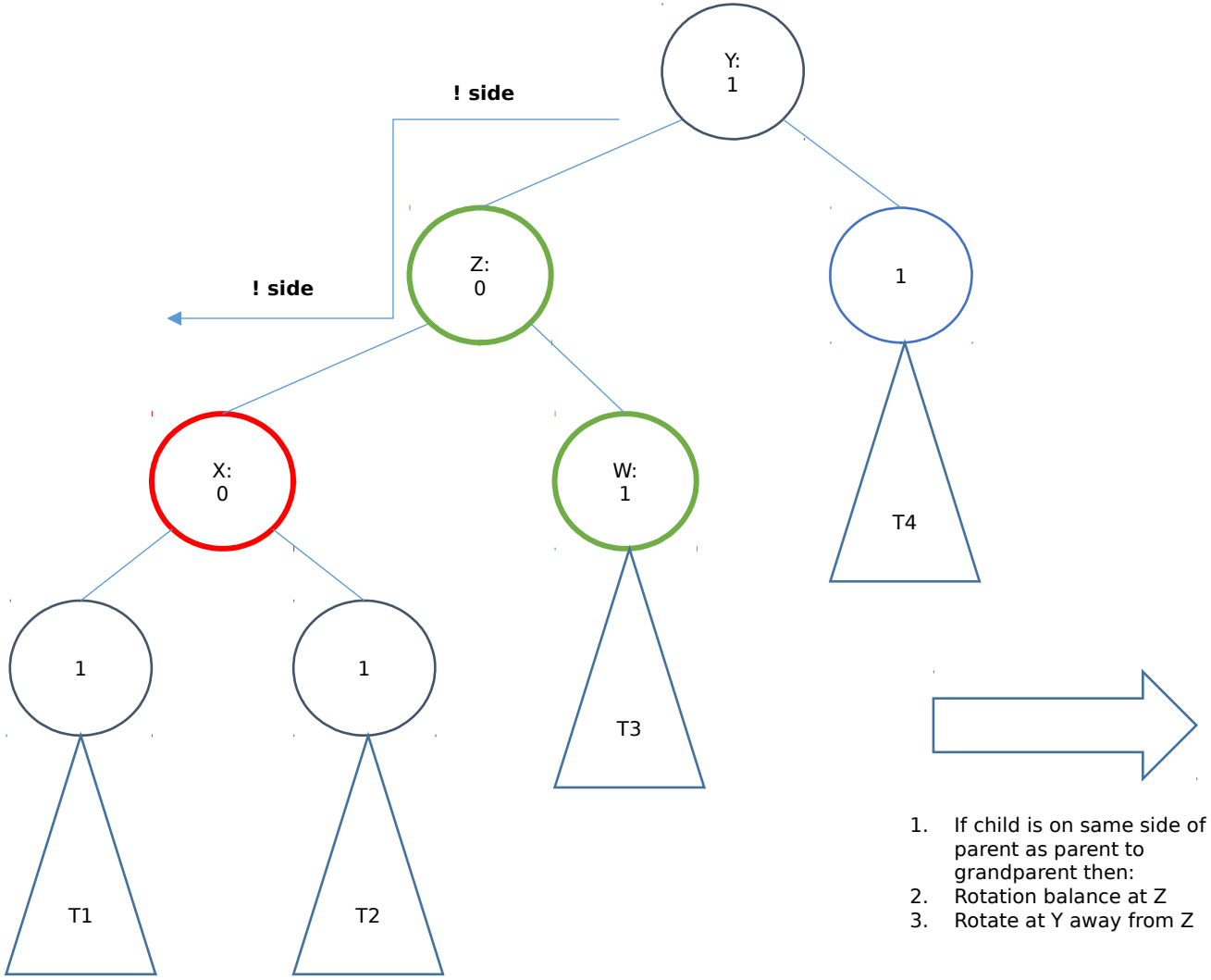
2-3 insertion – Case 2



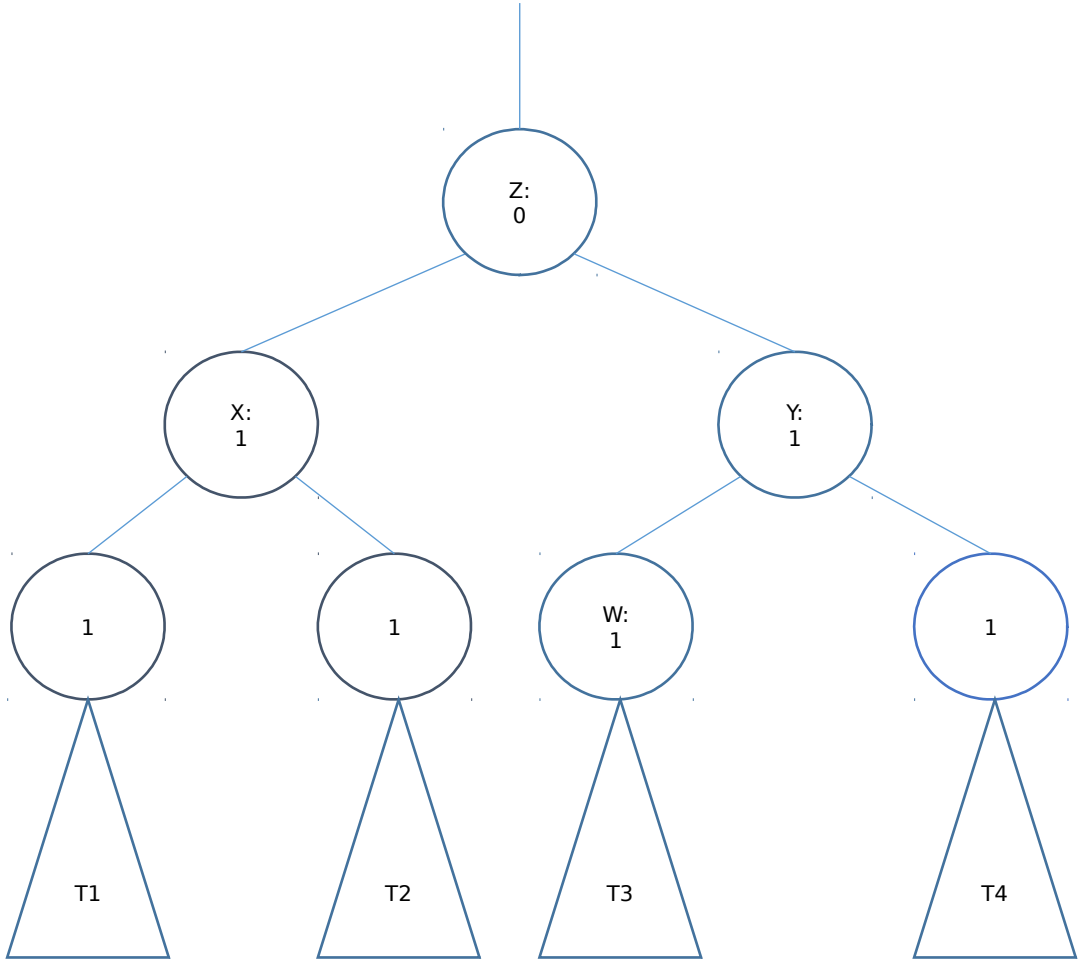
1. If child is on same side of parent as parent to grandparent then:
2. Rotation balance at Z
3. Rotate at Y away from Z
4. Promote Z
5. X gets X->p
6. Continue balancing loop



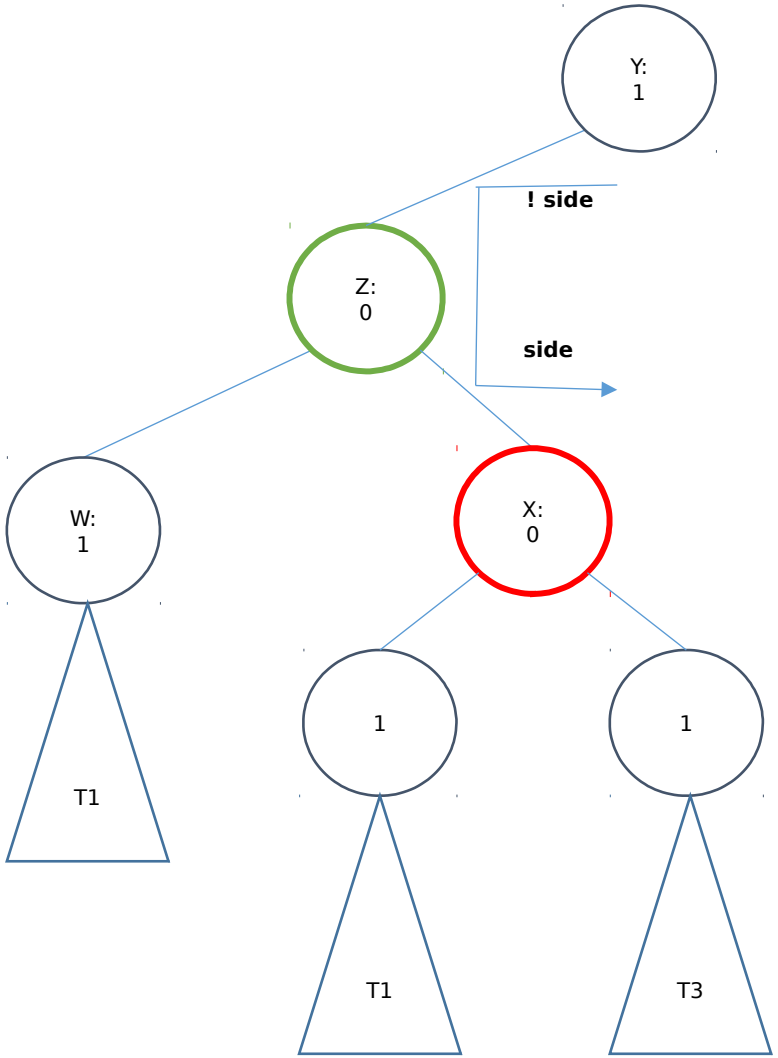
2-3-4 insertion – Case 2



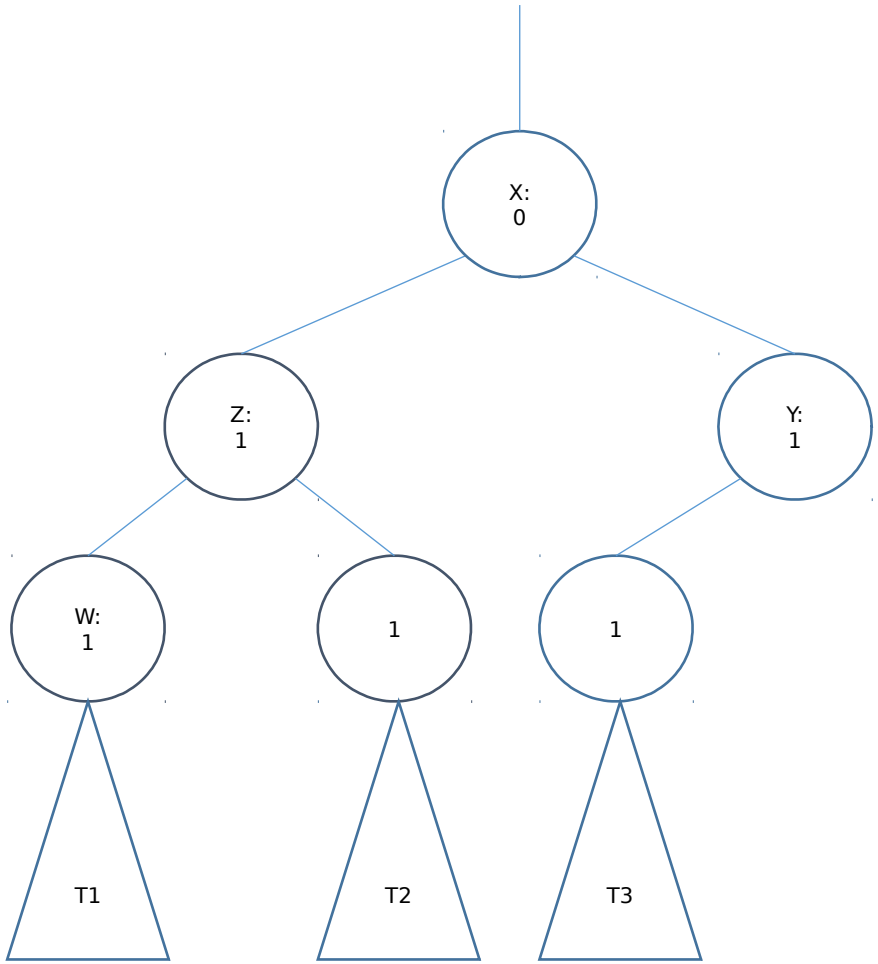
1. If child is on same side of parent as parent to grandparent then:
2. Rotation balance at Z
3. Rotate at Y away from Z



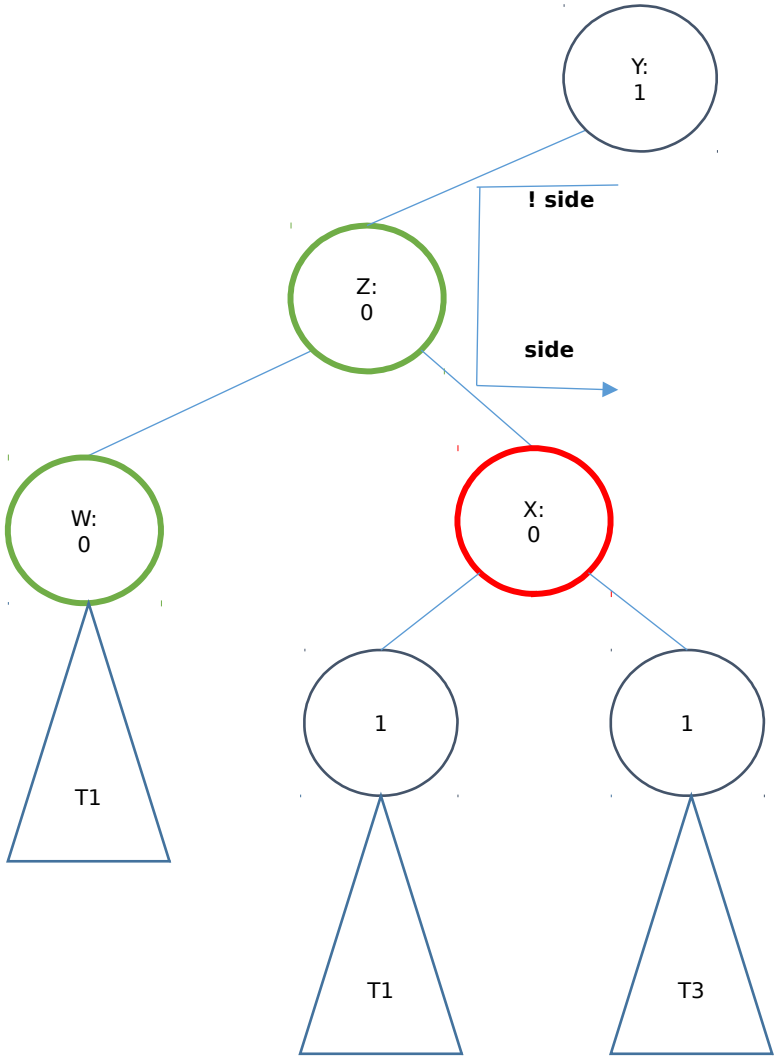
2-3 insertion – Case 3



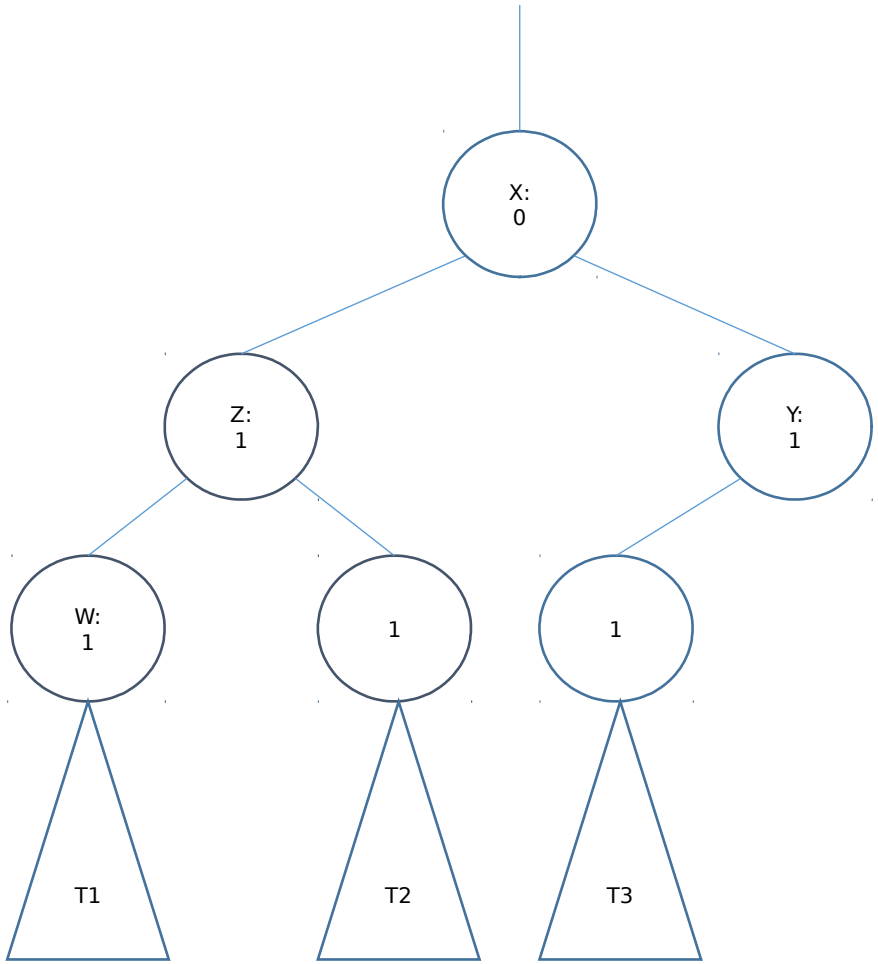
1. If child is on different side of parent as parent to grandparent then:
2. Rotation balance at X
3. Rotate at Z away from X
4. Rotation balance at X
5. Rotate at Y away from X
6. Promote X
7. X gets X->p
8. Continue balancing loop



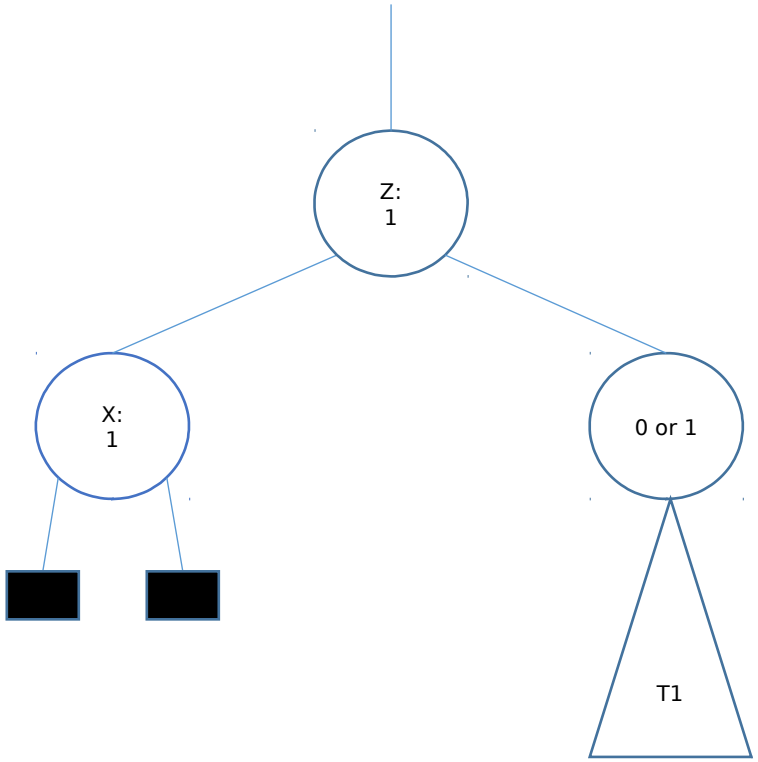
2-3-4 insertion – Case 3



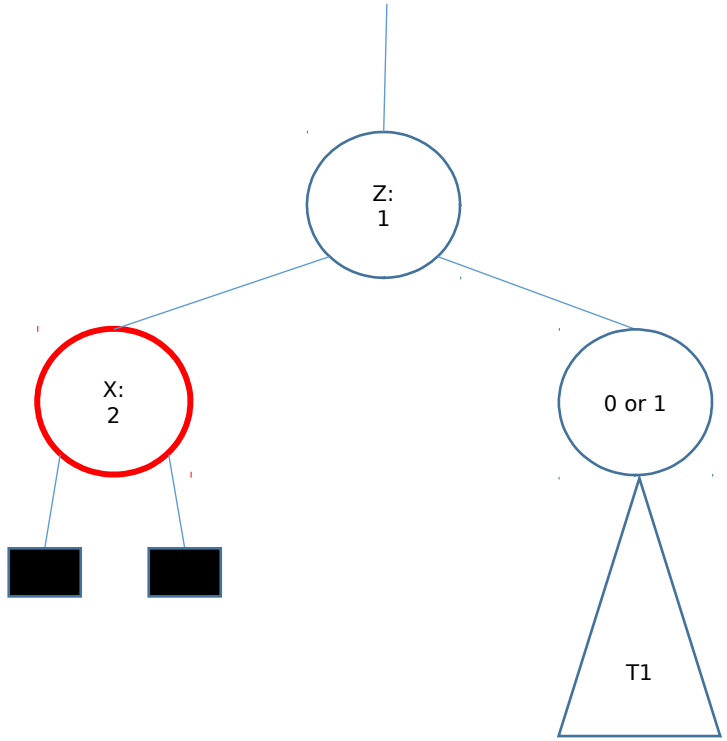
1. If child is on different side of parent as parent to grandparent then:
2. Rotation balance at X
3. Rotate at Z away from X
4. Rotation balance at X
5. Rotate at Y away from X
6. Promote X



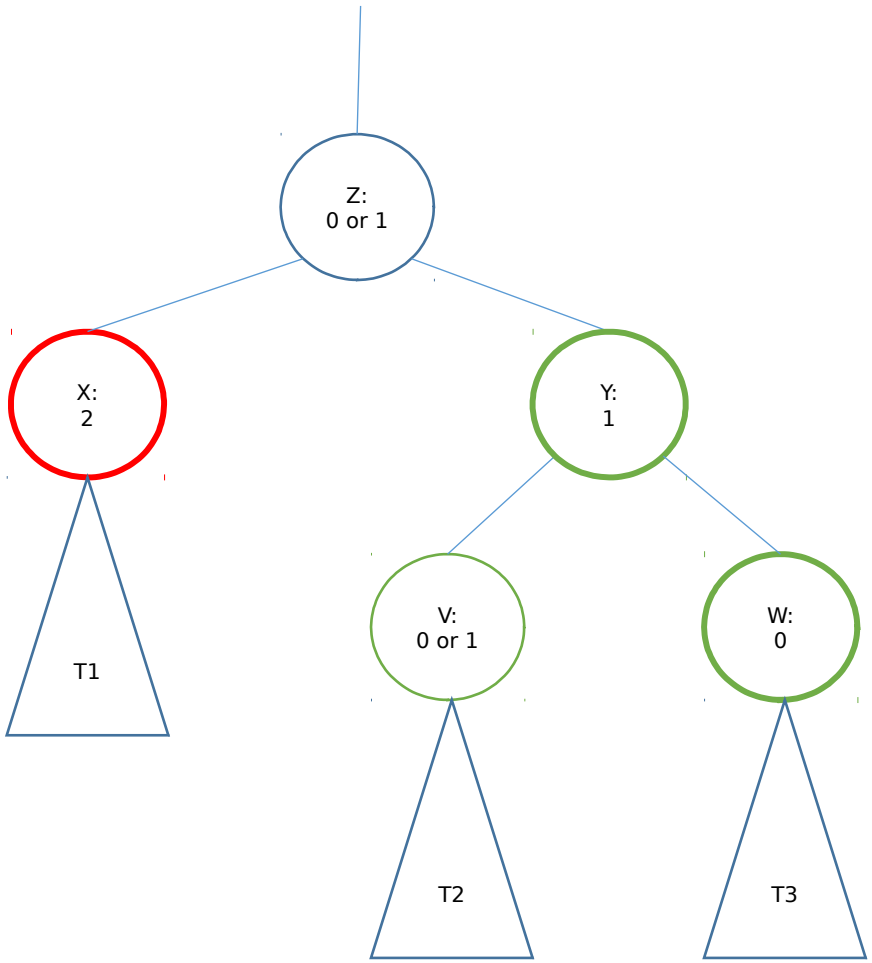
2-3/2-3-4 deletion – Preparation – occurs in delete function in Tree.c



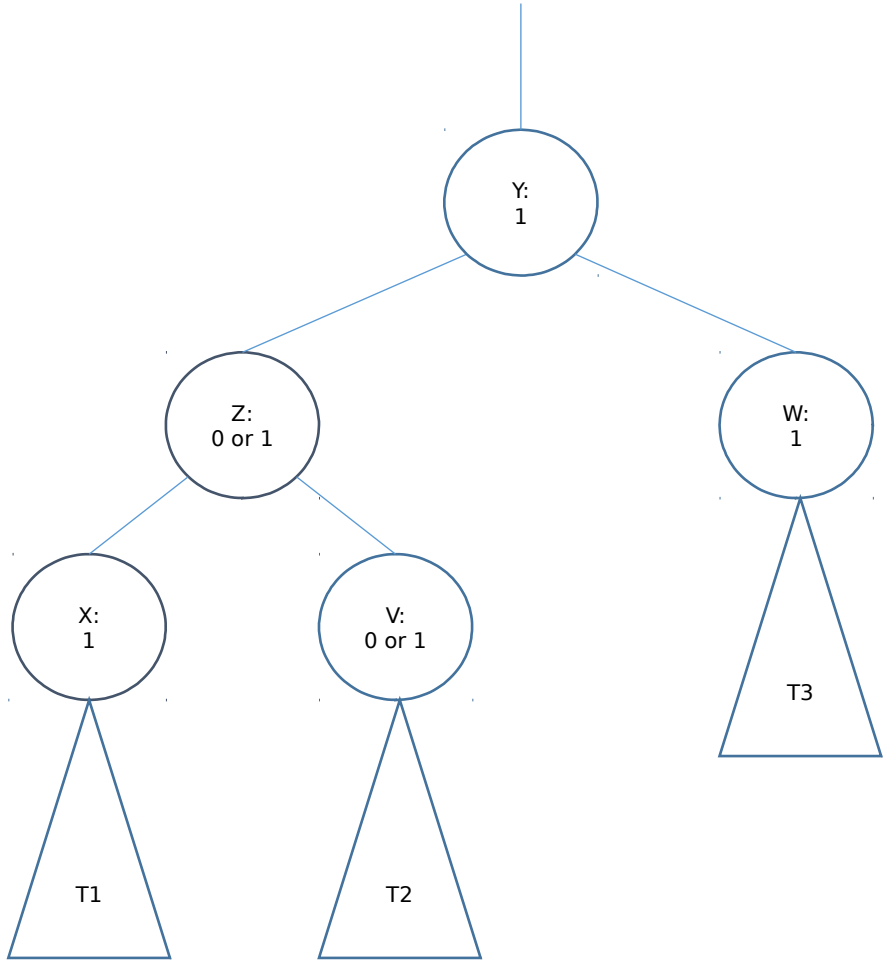
1. X is deleted and demoted to simulate the resulting rank difference Z would have with a missing node. If X had rank difference of 0, no balancing is required. If X had rank difference of 1, demotion makes it 2 so balancing is required
2. Enter balance loop with X
3. Continue balance loop



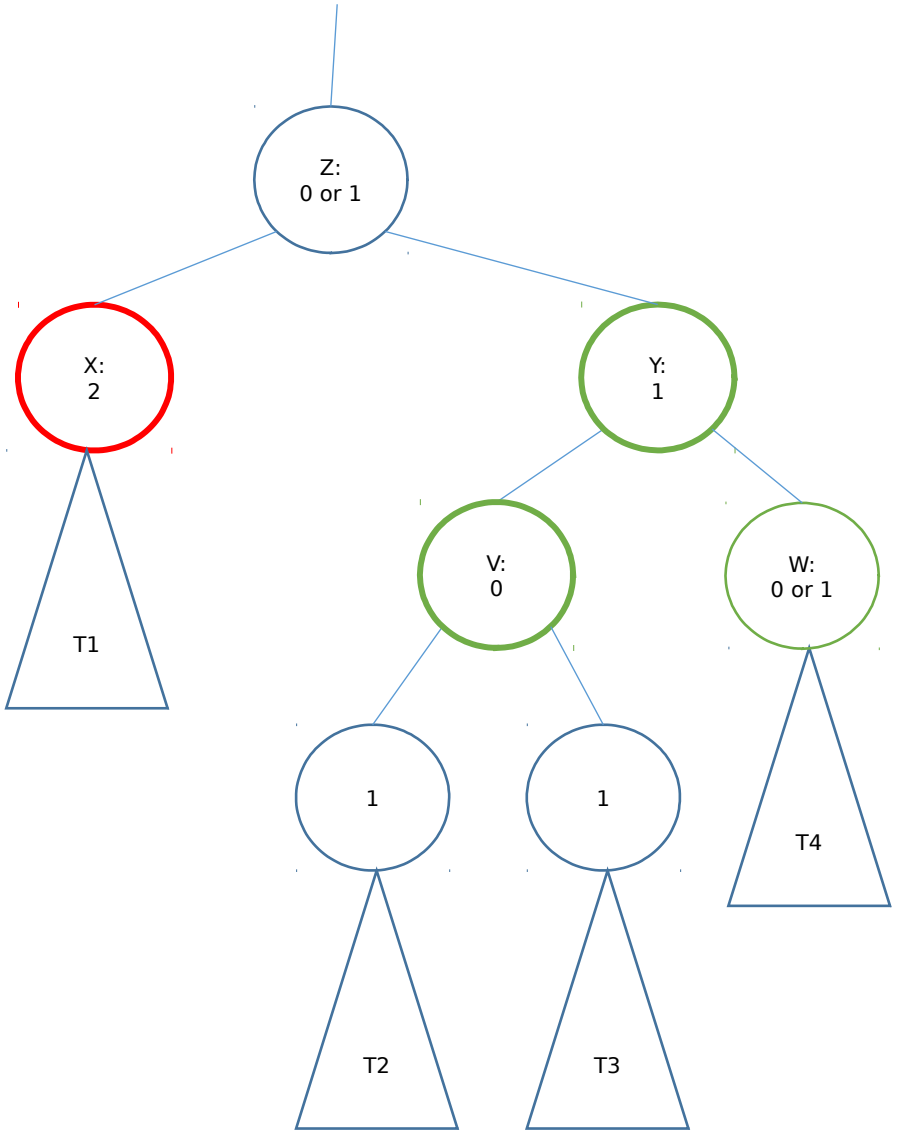
2-3/2-3-4 deletion – Case 1a



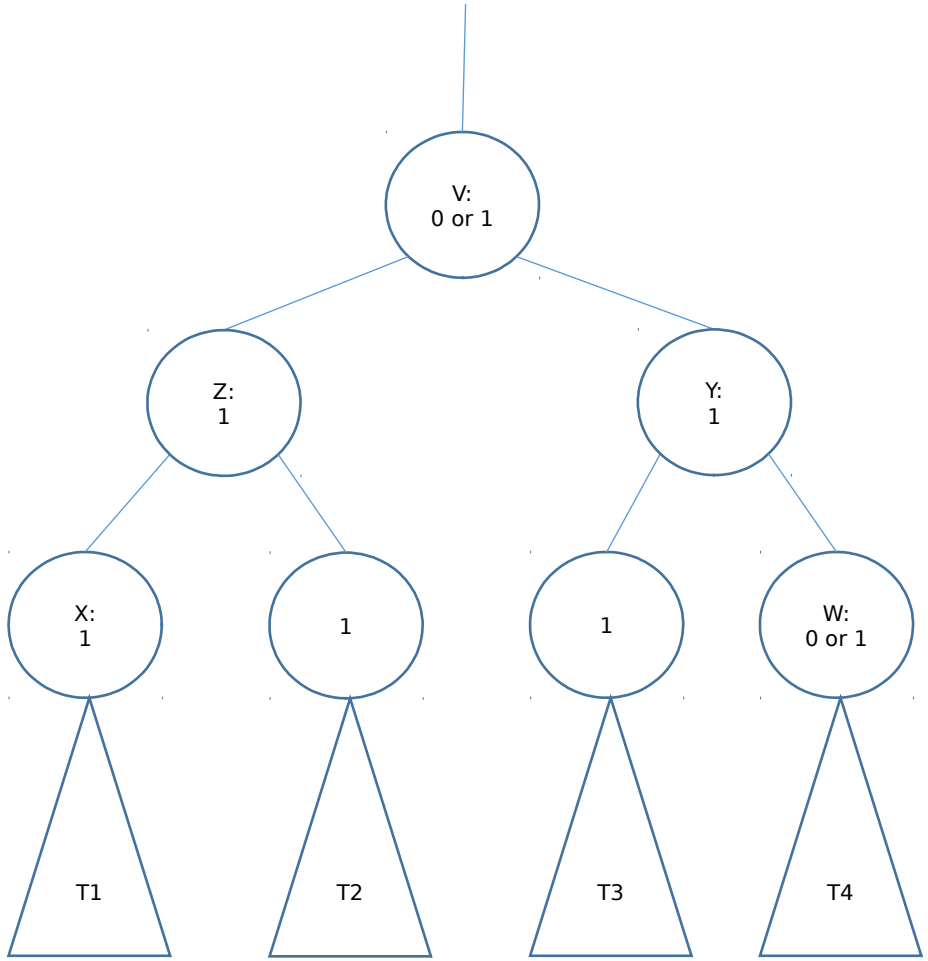
1. Either V or W have a rank difference of 0, W has in this example
2. Rotation balance at Y
3. Rotate at Z away from Y
4. Balance restored



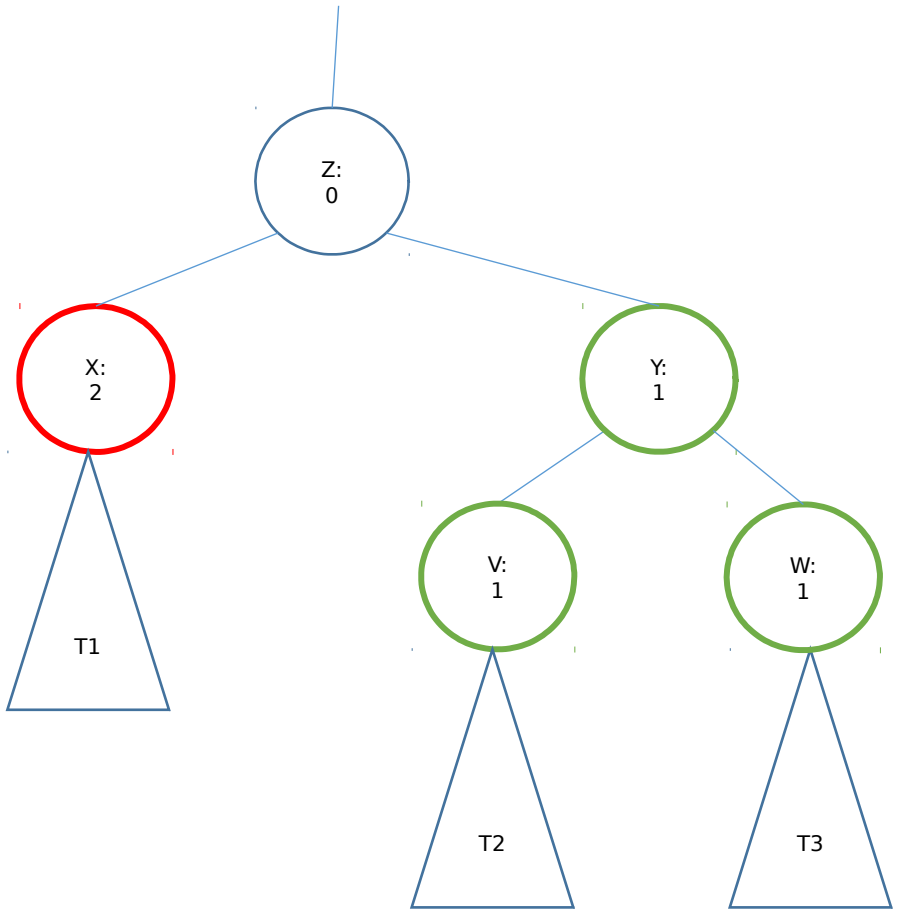
2-3/2-3-4 insertion – Case 1b



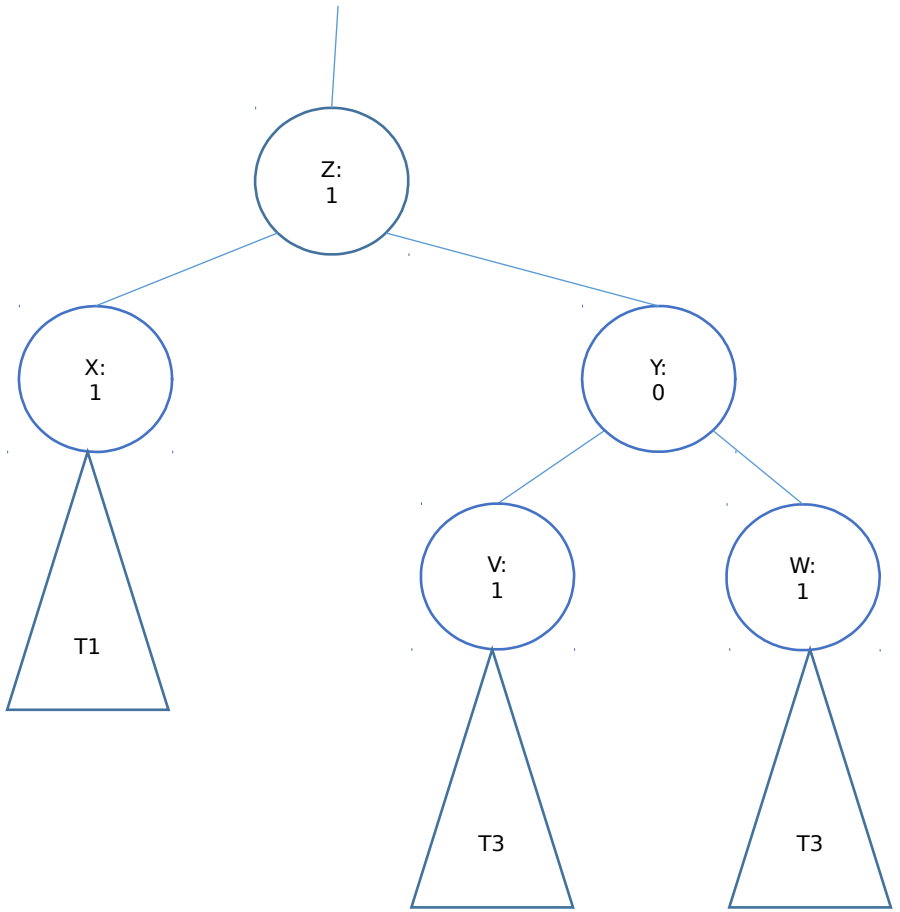
1. Either V or W have a rank difference of 0, V has in this example
2. Rotation balance at V
3. Rotate at Y away from V
4. Rotation balance at V
5. Rotate at Z away from V
6. Demote Z
7. AVL – Demote V and continue balancing loop
8. WAVL – Balance restored



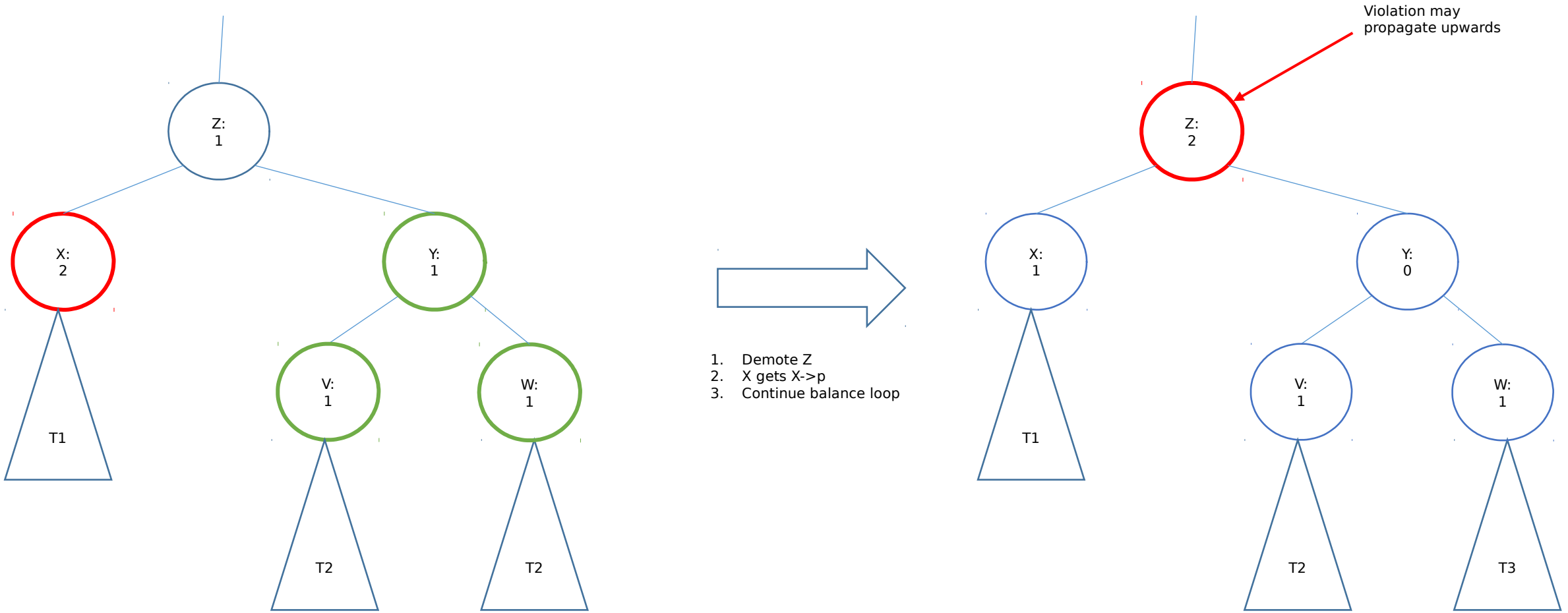
2-3/2-3-4 insertion – Case 2 – outcome (a)



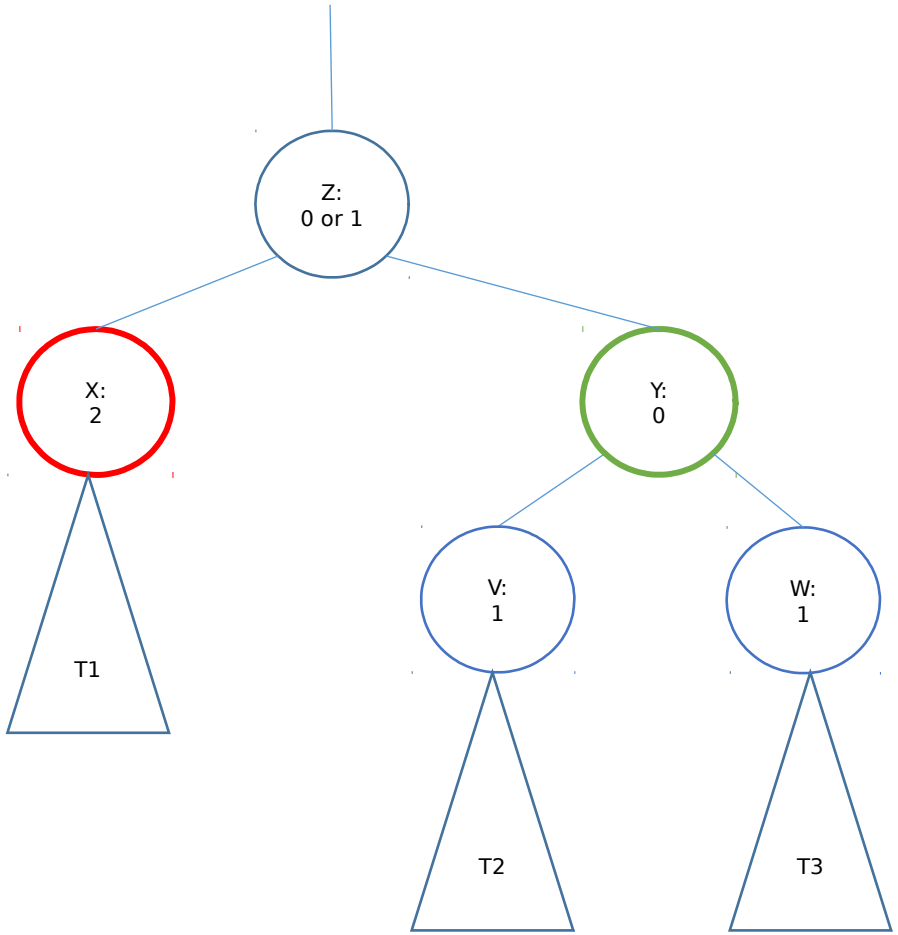
1. Demote Z
2. X gets X->p
3. Enter balance loop
4. Balance restored



2-3/2-3-4 insertion – Case 2 – outcome (b)



2-3/2-3-4 insertion – Case 3



1. Rotation balance at Y
2. Rotate Z away from Y
3. Continue balance loop