COL106
Data Structures and Algorithms

Subodh Sharma and Rahul Garg

AVL Trees: Removal

Remove from BST

- Three cases
  - Case 1: Node to be removed is a leaf node
  - Case 2: Node to be removed has one child
  - Case 3: Node to be removed has two children

Case 1: Leaf Node Removal

- Remove 75

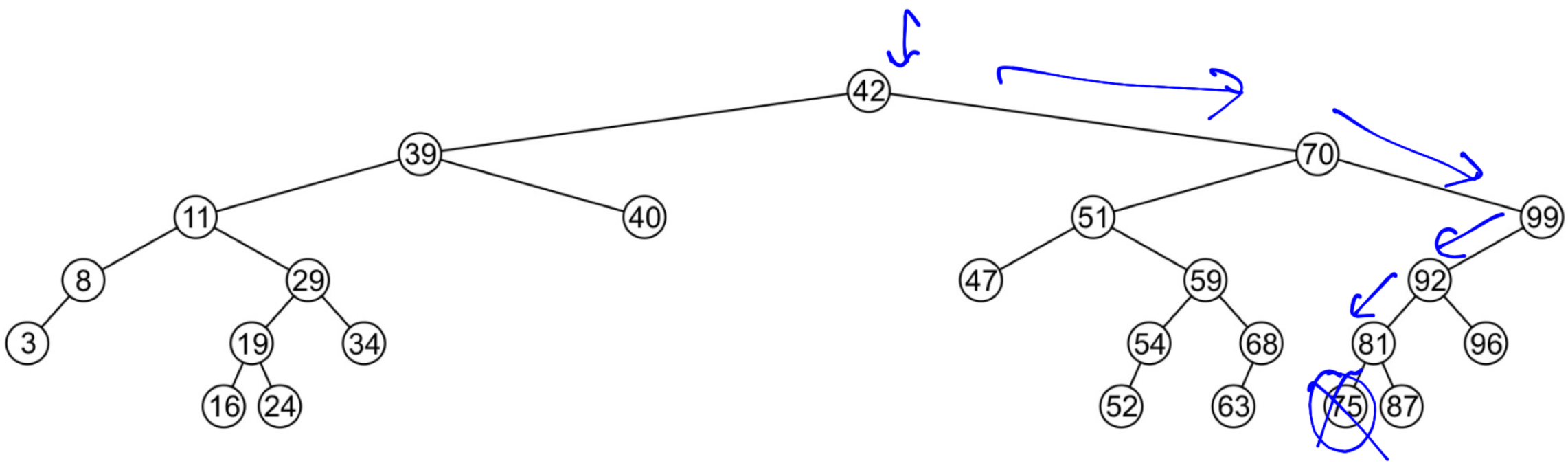


Figure courtesy: Douglas Wilhelm Harder, University of Waterloo, Canada

Case 1: Leaf Node Removal

- Set the left pointer of 81 to null
- Free the memory allocated for node 75

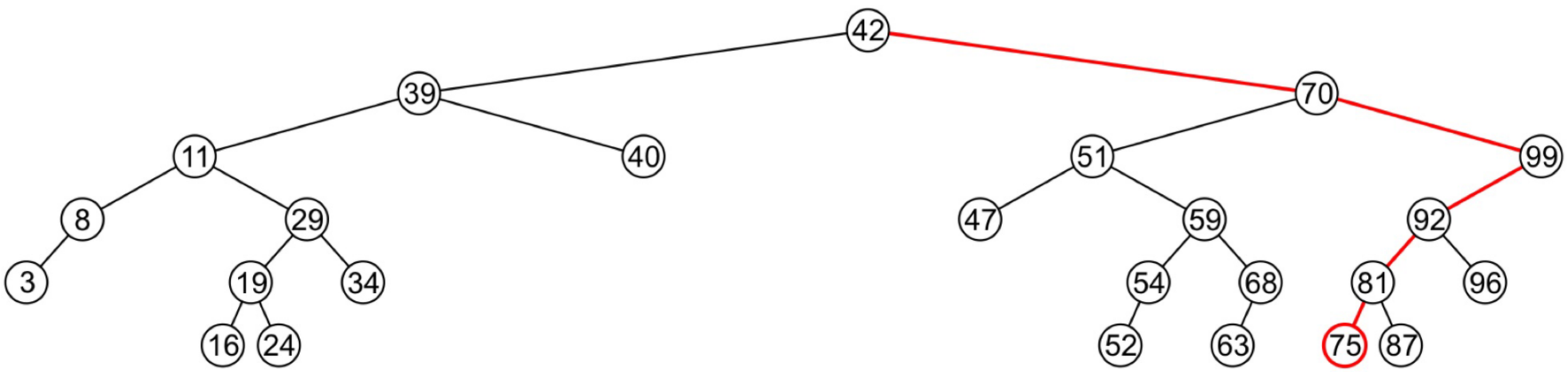


Figure courtesy: Douglas Wilhelm Harder, University of Waterloo, Canada

Case 1: Leaf Node Removal

- Remove 40 is similar

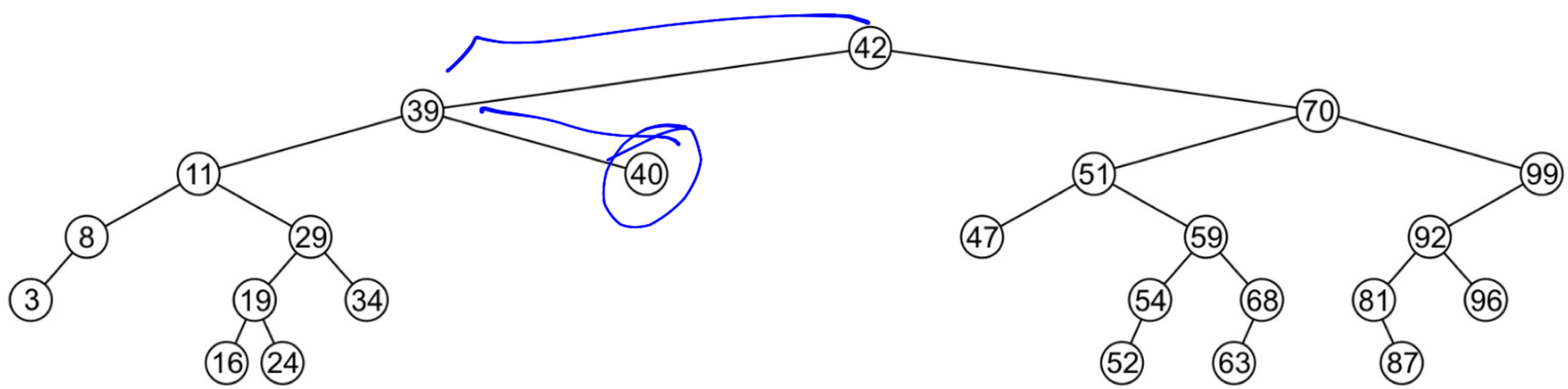


Figure courtesy: Douglas Wilhelm Harder, University of Waterloo, Canada

Case 1: Leaf Node Removal

- Free space for node 40
- Set right pointer of 29 to null

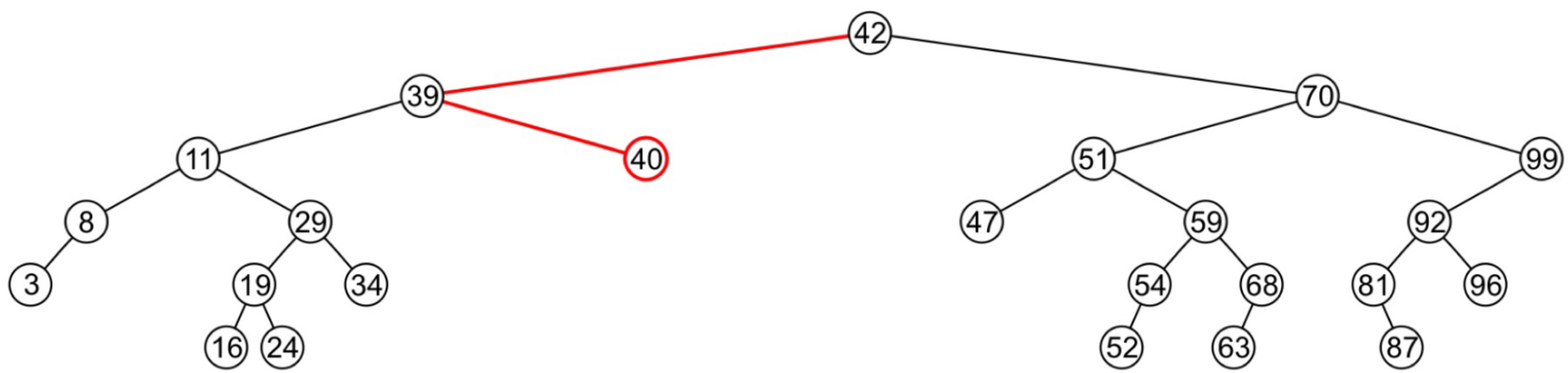


Figure courtesy: Douglas Wilhelm Harder, University of Waterloo, Canada

Case 2: Removing Single Child Node

- Remove 8

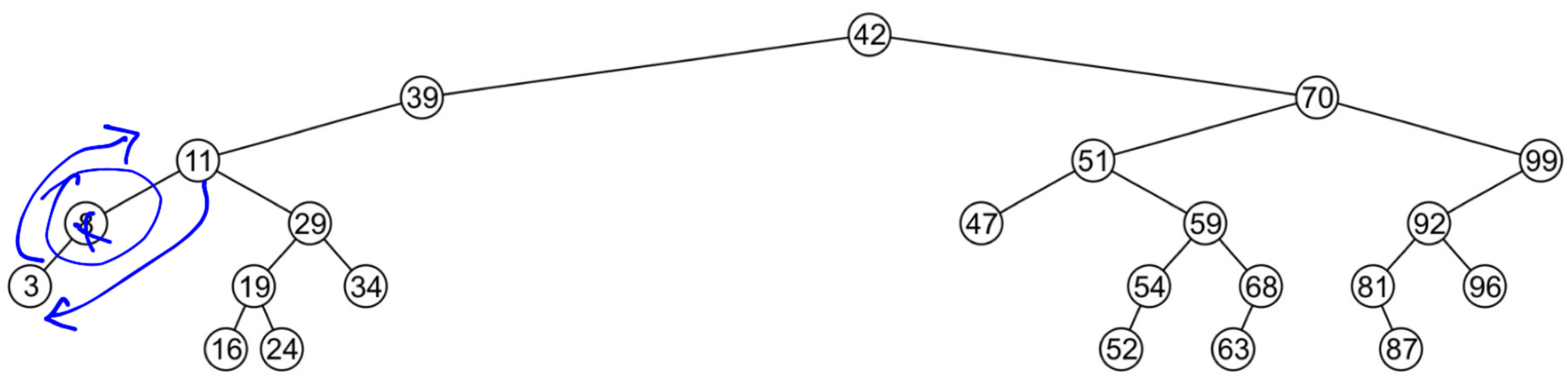


Figure courtesy: Douglas Wilhelm Harder, University of Waterloo, Canada

Case 2: Removing Single Child Node

- Remove 8
- Free space of node 8
- Promote the sub-tree at node 8

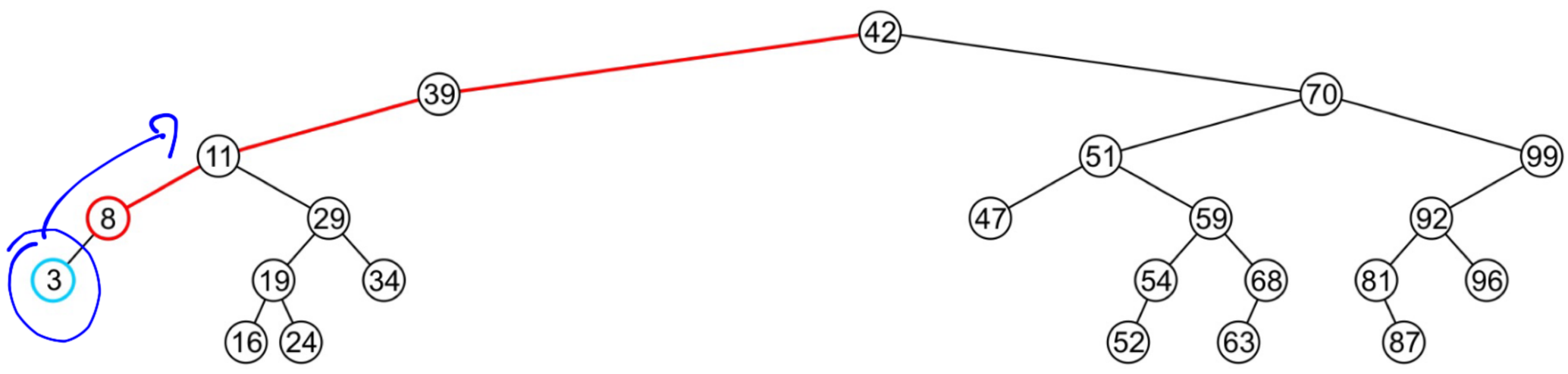


Figure courtesy: Douglas Wilhelm Harder, University of Waterloo, Canada

Case 2: Removing Single Child Node

- Remove 8
- Free space of node 8
- Make 11's left child point to 3
- Make 3's parent point to parent of 8 (i.e., 11)

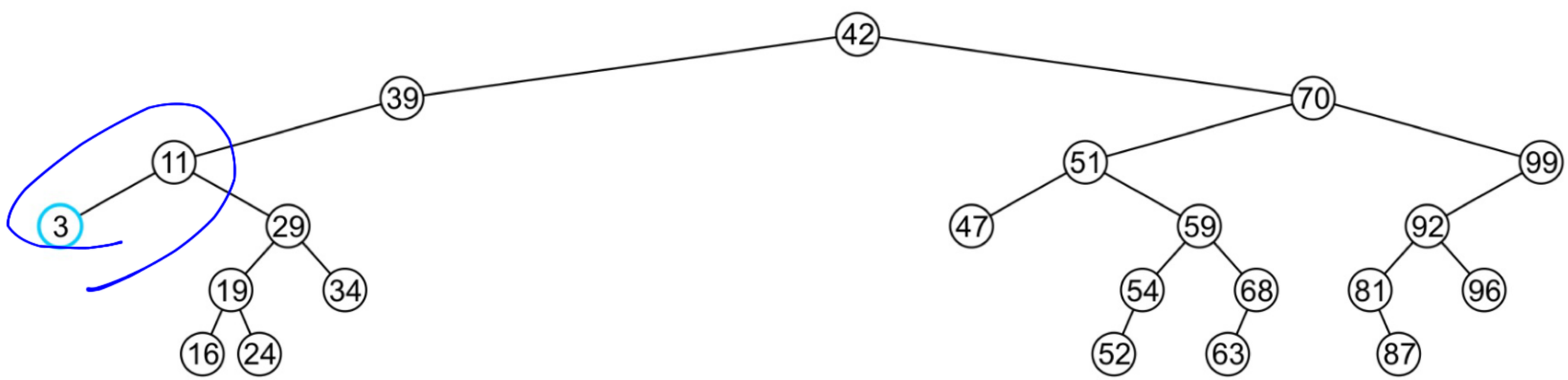


Figure courtesy: Douglas Wilhelm Harder, University of Waterloo, Canada

Case 2: Removing Single Child Node

- Remove 39
- No difference between promoting a node and promoting a subtree
- 39 has a single child 11

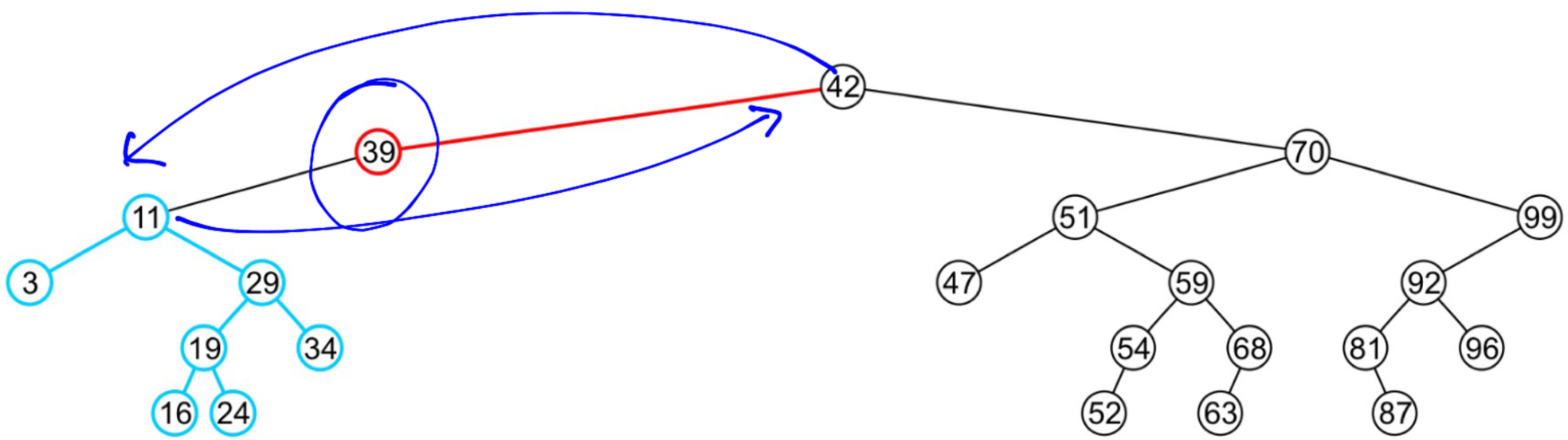


Figure courtesy: Douglas Wilhelm Harder, University of Waterloo, Canada

Case 2: Removing Single Child Node

- Remove 39
- 39 has a single left child 11
- Make 42's left point to 11
- Make 11's parent point to 42
- Is the tree still a BST?

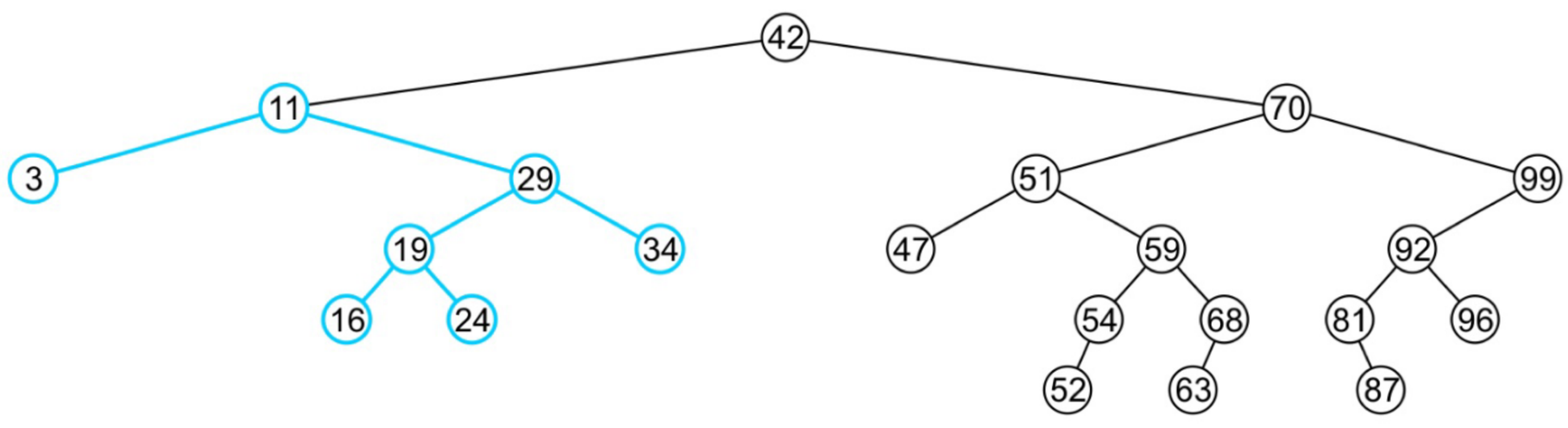


Figure courtesy: Douglas Wilhelm Harder, University of Waterloo, Canada

Case 2: Removing Single Child Node

- Remove 99

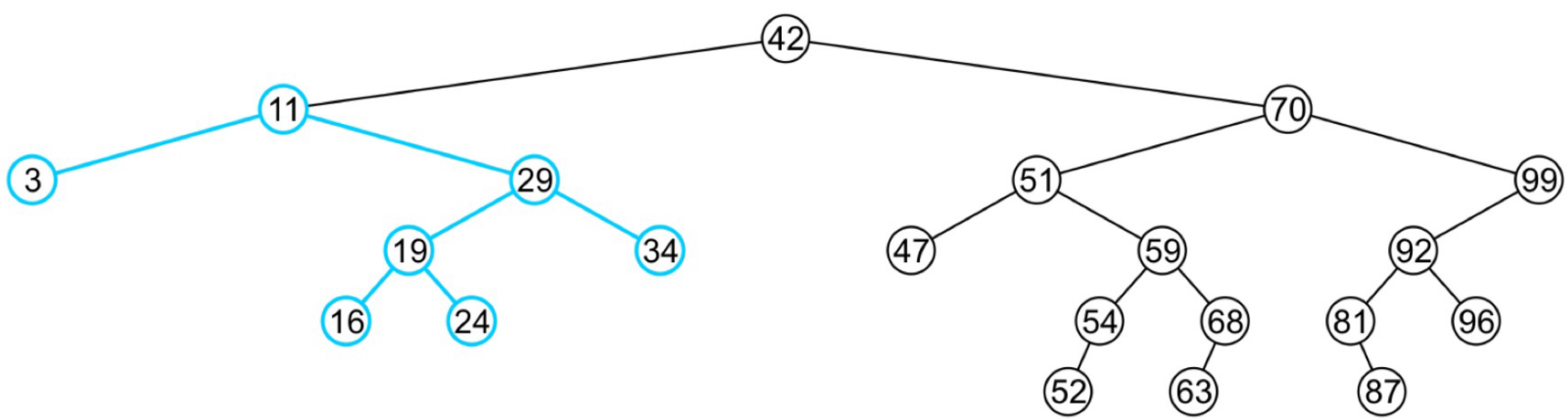


Figure courtesy: Douglas Wilhelm Harder, University of Waterloo, Canada

Case 2: Removing Single Child Node

- Remove 99

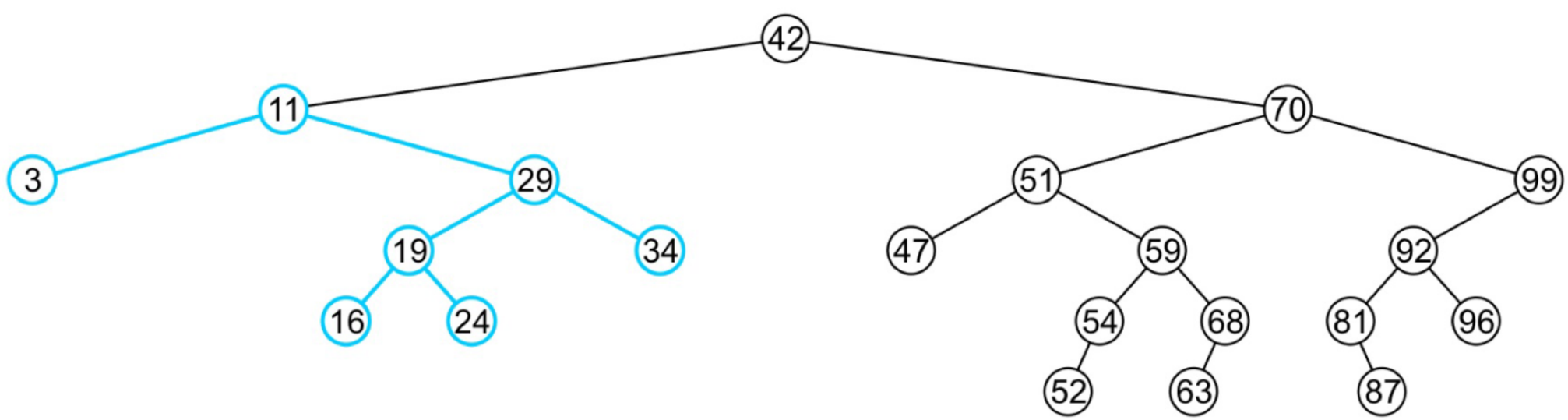


Figure courtesy: Douglas Wilhelm Harder, University of Waterloo, Canada

Case 2: Removing Single Child Node

- Remove 99

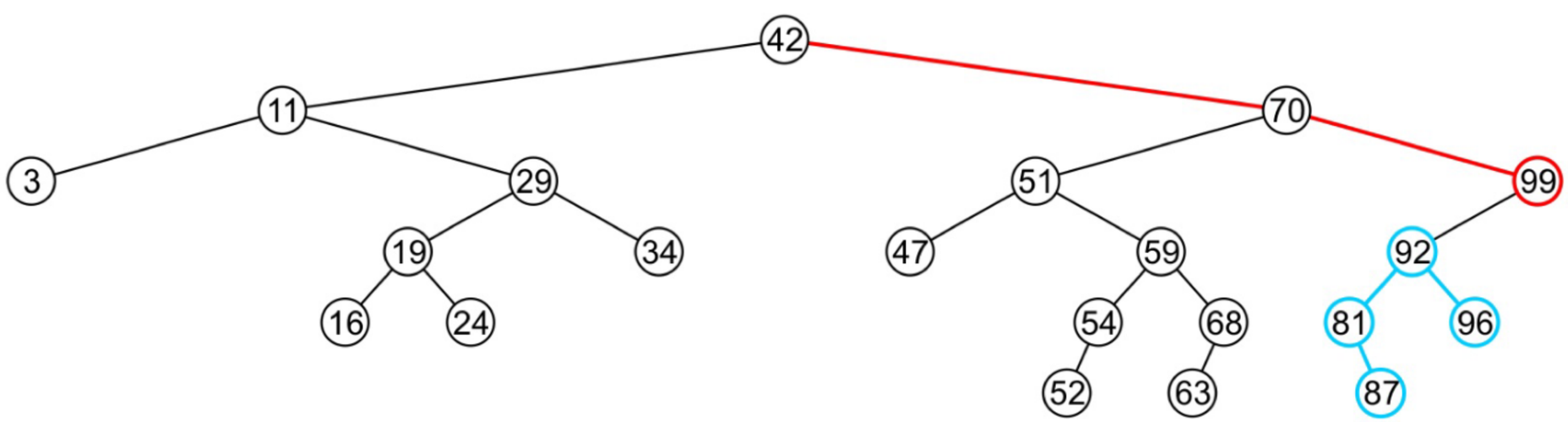


Figure courtesy: Douglas Wilhelm Harder, University of Waterloo, Canada

Case 2: Removing Single Child Node

- Remove 99
- The node is deleted and the left sub-tree is promoted
- The right child of 70 is set to point to 92, even though 92 was the left child of 99
- The tree is still BST and its order is maintained

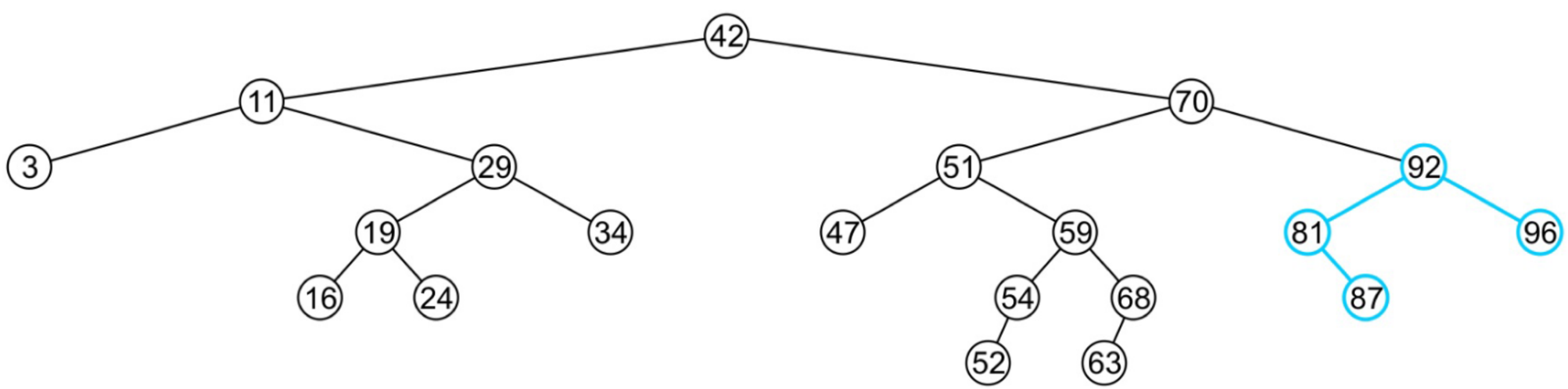


Figure courtesy: Douglas Wilhelm Harder, University of Waterloo, Canada

Case 3: Removing Node with two Children

- Remove 42

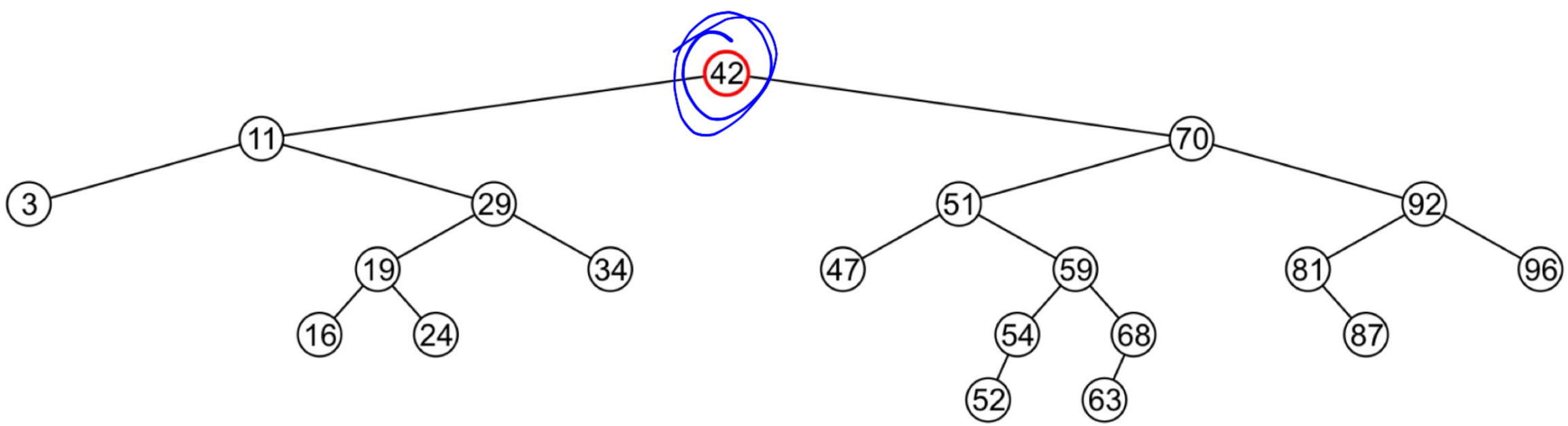


Figure courtesy: Douglas Wilhelm Harder, University of Waterloo, Canada

Case 3: Removing Node with two Children

- Remove 42
  - Step 1: Find successor of 42
  - Successor is 47

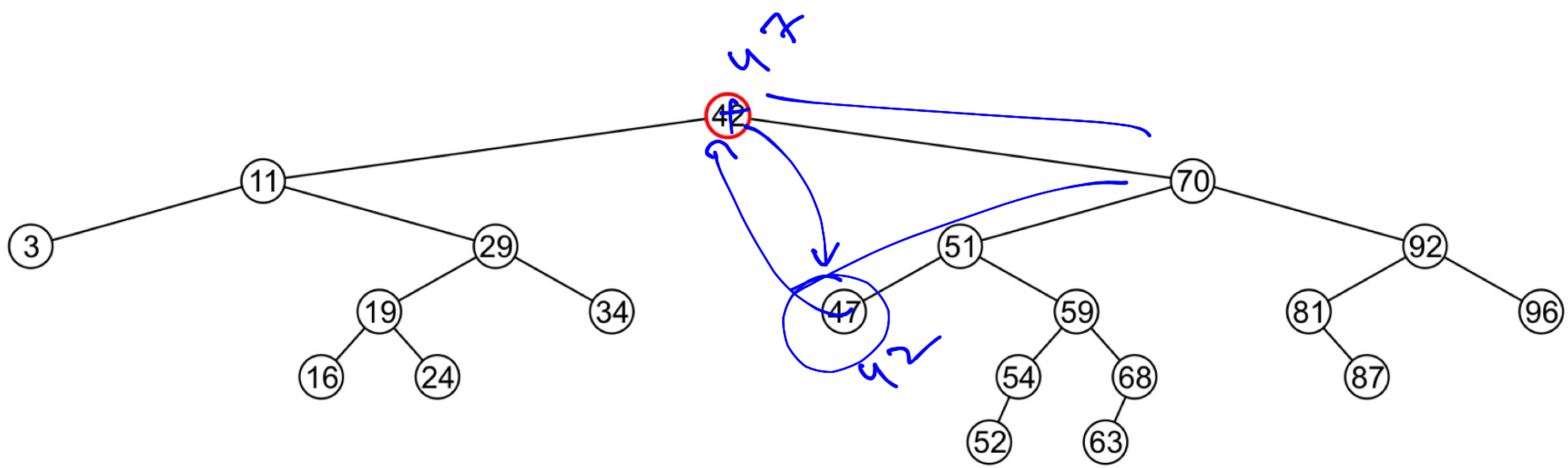


Figure courtesy: Douglas Wilhelm Harder, University of Waterloo, Canada

Case 3: Removing Node with two Children

- Remove 42
  - Step 1: Find successor of 42 (47)
  - Step 2: Copy 47 onto 42
  - We temporarily may have two copies of 47

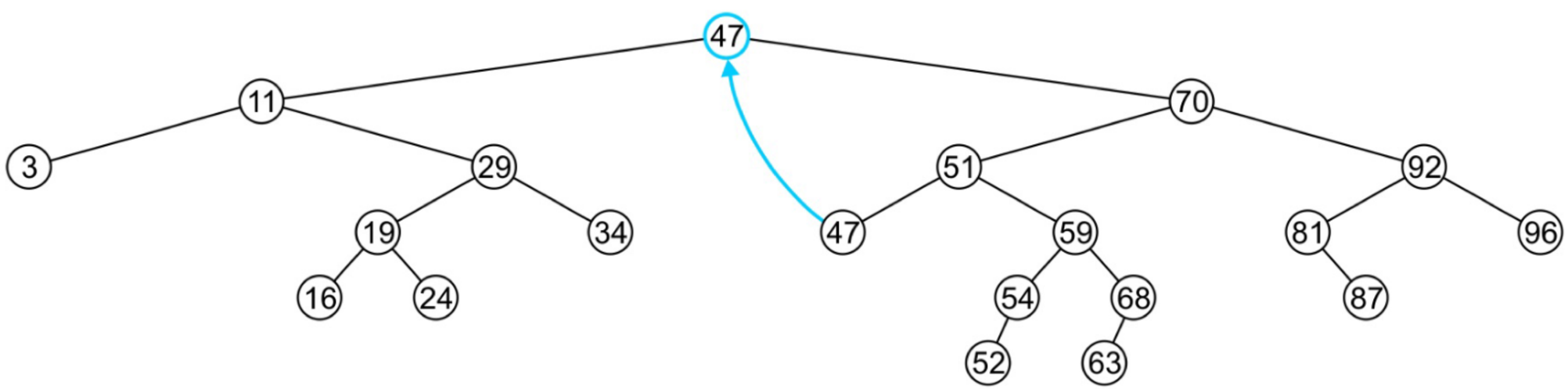


Figure courtesy: Douglas Wilhelm Harder, University of Waterloo, Canada

Case 3: Removing Node with two Children

- Remove 42
  - Step 1: Find successor of 42 (47)
  - Step 2: Copy 47 onto 42
  - Step 3: Now remove 47 from the right subtree
  - 47 is a leaf node in the right sub-tree
  - Note: The successor can never be a node with two children: Why?
  - So, removal of 47 can be done using previous cases

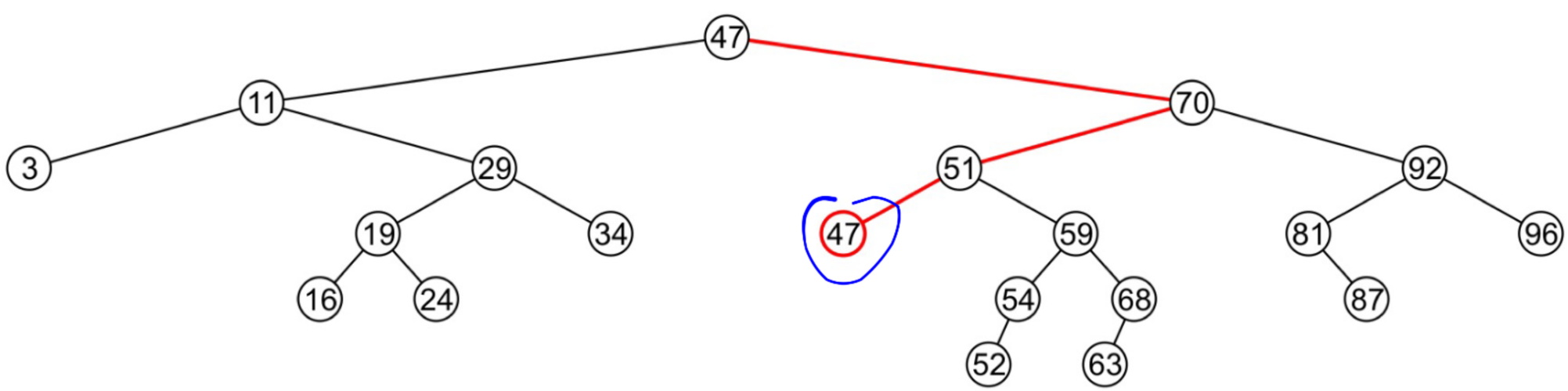


Figure courtesy: Douglas Wilhelm Harder, University of Waterloo, Canada

Case 3: Removing Node with two Children

- Lemma: The successor of a node with two children can never be a node with two children.
- Proof: Consider a node with two children
  - Its successor must be in its right subtree
  - Actually, it is the smallest node in the right subtree
  - If the left child of successor is not null, there are smaller nodes than the successor in the right-subtree..
  - Therefore, the left child of the successor must be null
  - Hence successor cannot have two children

Case 3: Removing Node with two Children

- Lemma: The successor of a node with two children can never be a node with two children
- Two cases of the successor
  - Case 1: Successor is a leaf node
  - Case 2: Successor has a non-empty right subtree

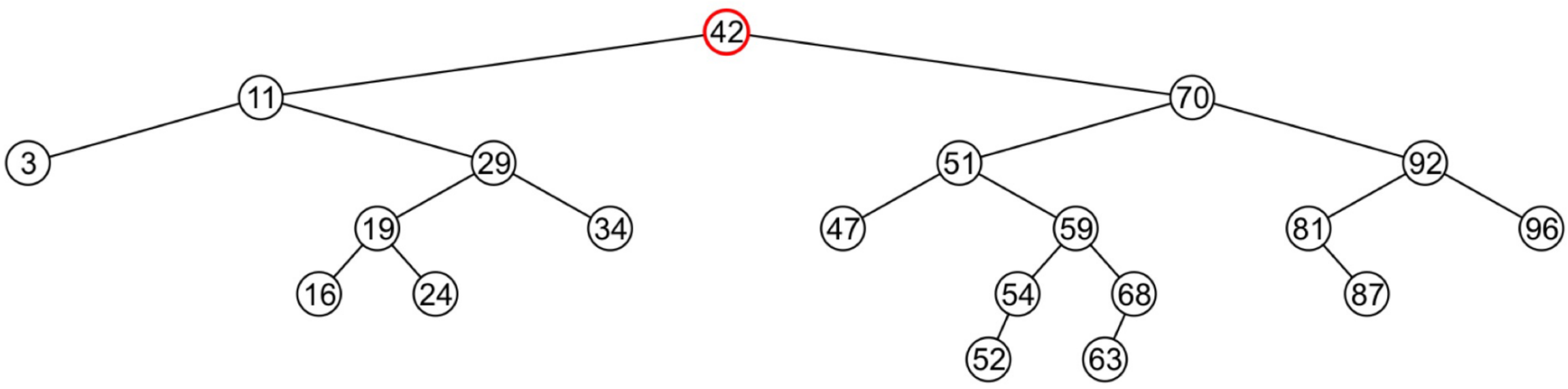


Figure courtesy: Douglas Wilhelm Harder, University of Waterloo, Canada

Case 3: Removing Node with two Children

- Remove 47
- Step 1: Find successor which is 51

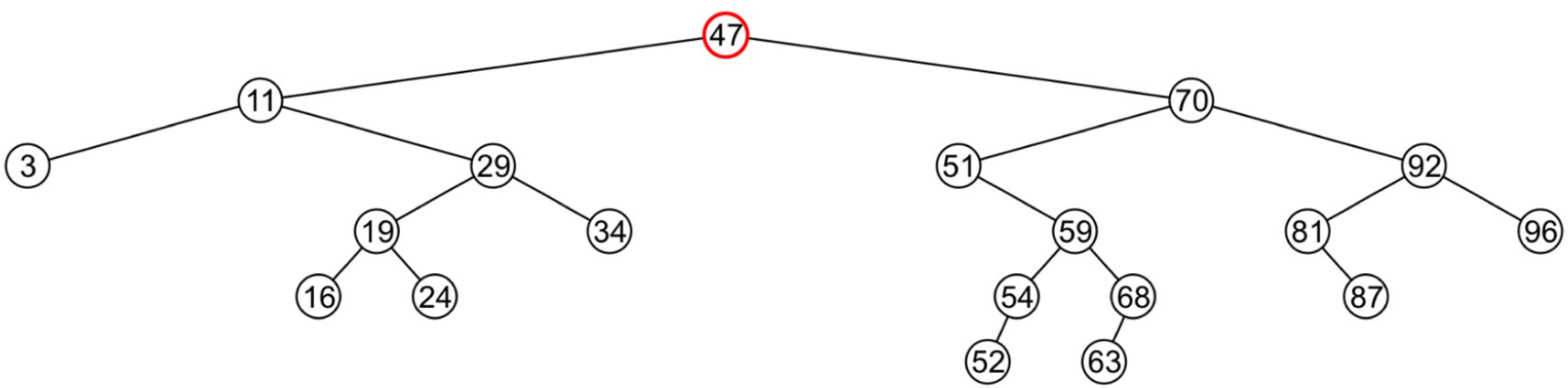


Figure courtesy: Douglas Wilhelm Harder, University of Waterloo, Canada

Case 3: Removing Node with two Children

- Remove 47
- Step 1: Find successor which is 51
- Step 2: Copy successor

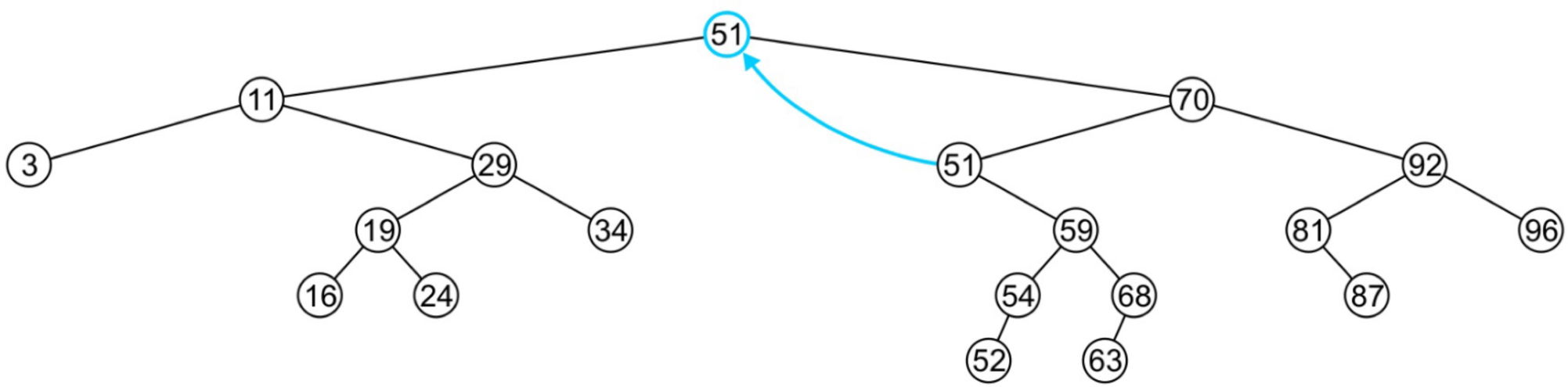


Figure courtesy: Douglas Wilhelm Harder, University of Waterloo, Canada

Case 3: Removing Node with two Children

- Remove 47
- Step 1: Find successor which is 51
- Step 2: Copy successor
- Step 3: Remove successor from the right subtree

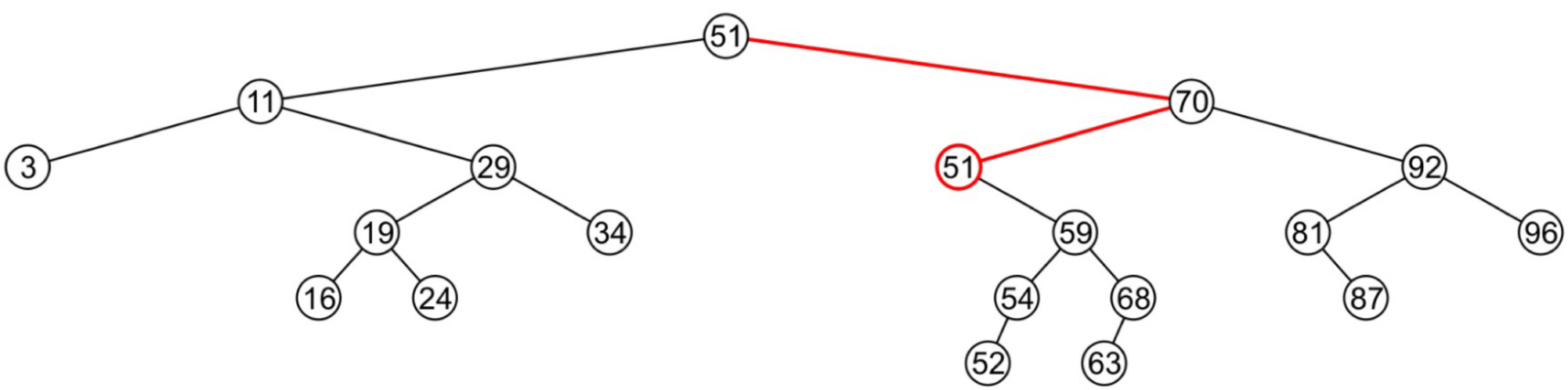


Figure courtesy: Douglas Wilhelm Harder, University of Waterloo, Canada

Case 3: Removing Node with two Children

- Remove 47
- Step 1: Find successor which is 51
- Step 2: Copy successor
- Step 3: Remove successor from the right subtree
  - Now 51 has an associated subtree
  - But it can be promoted as in case 2

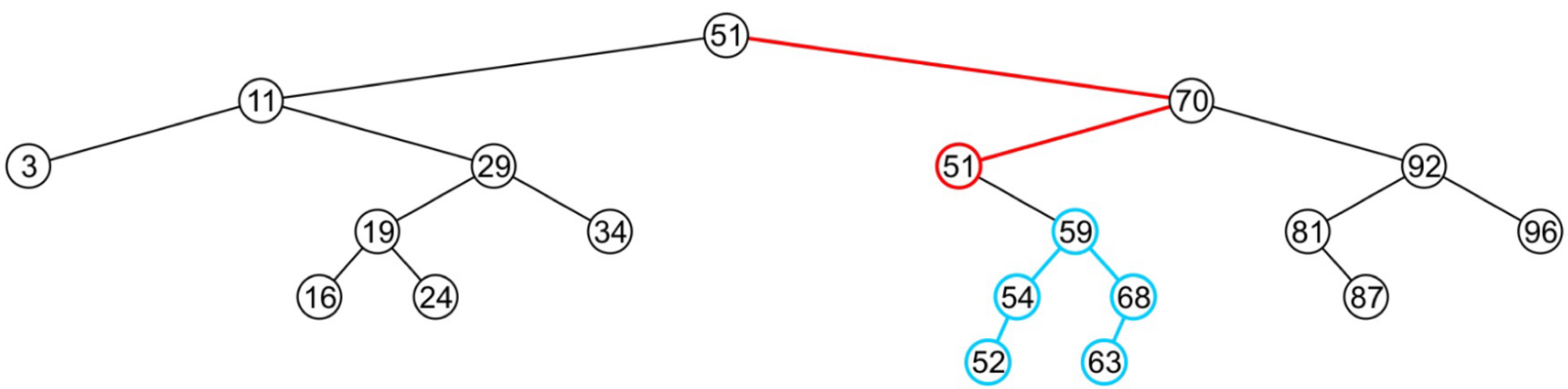


Figure courtesy: Douglas Wilhelm Harder, University of Waterloo, Canada

Case 3: Removing Node with two Children

- Remove 47
- Step 1: Find successor which is 51
- Step 2: Copy successor
- Step 3: Remove successor from the right subtree
  - Now 51 has an associated subtree
  - The subtree can be promoted as in case 2
  - Note that it is still a BST

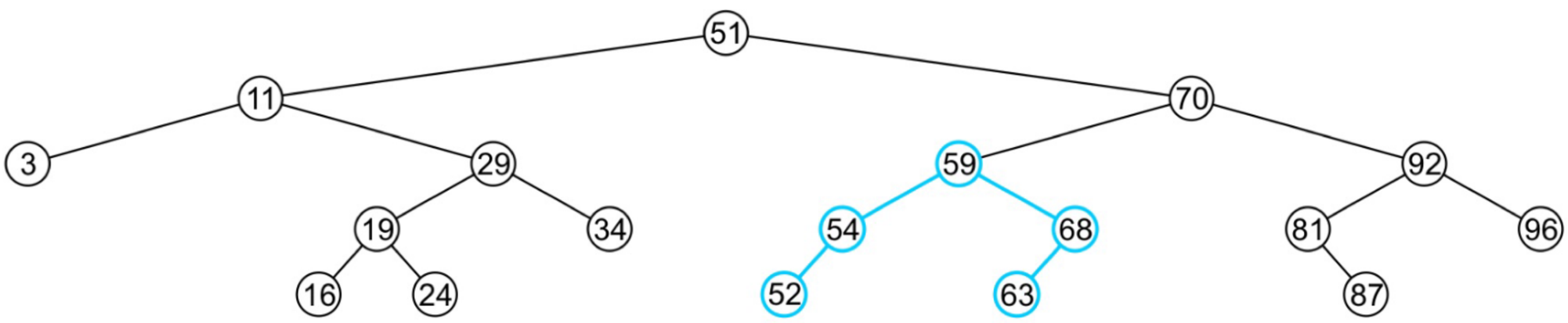


Figure courtesy: Douglas Wilhelm Harder, University of Waterloo, Canada

Ensuring Balance While Removing

- Case 3 of the removal is converted into
  - Case 1 (node with no children) or
  - Case 2 (note with one child)
  - Heights of the subtrees are not changed in copying step
- We only need to ensure balance in AVL trees in Case 1 and Case 2 of removals

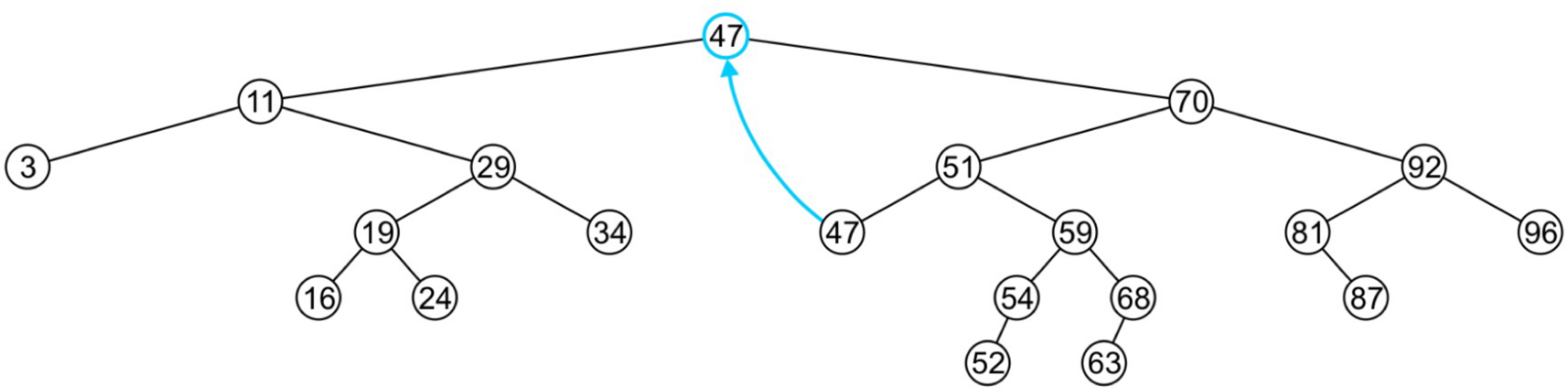
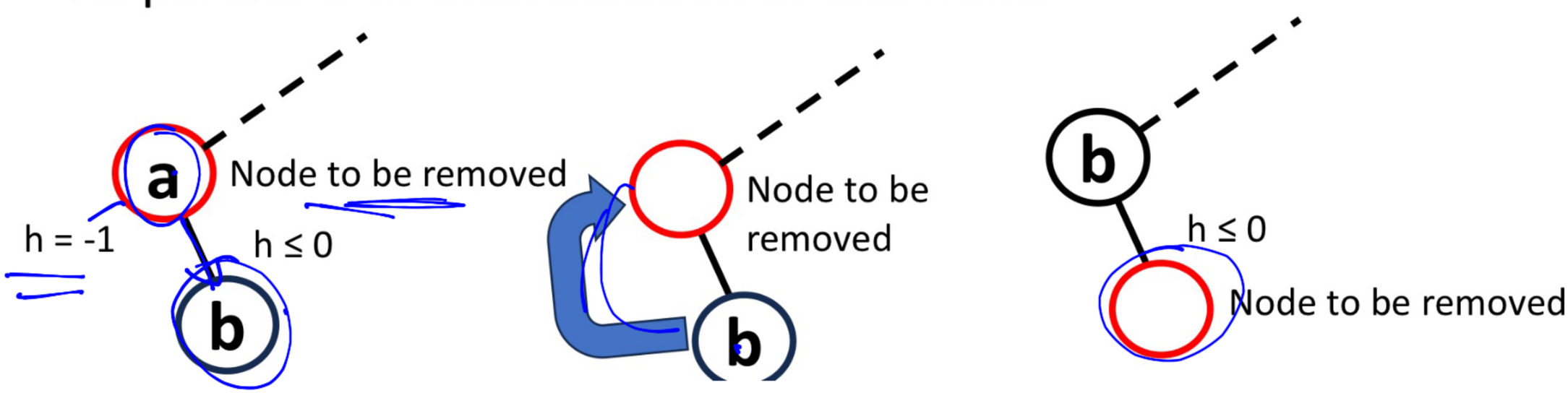


Figure courtesy: Douglas Wilhelm Harder, University of Waterloo, Canada

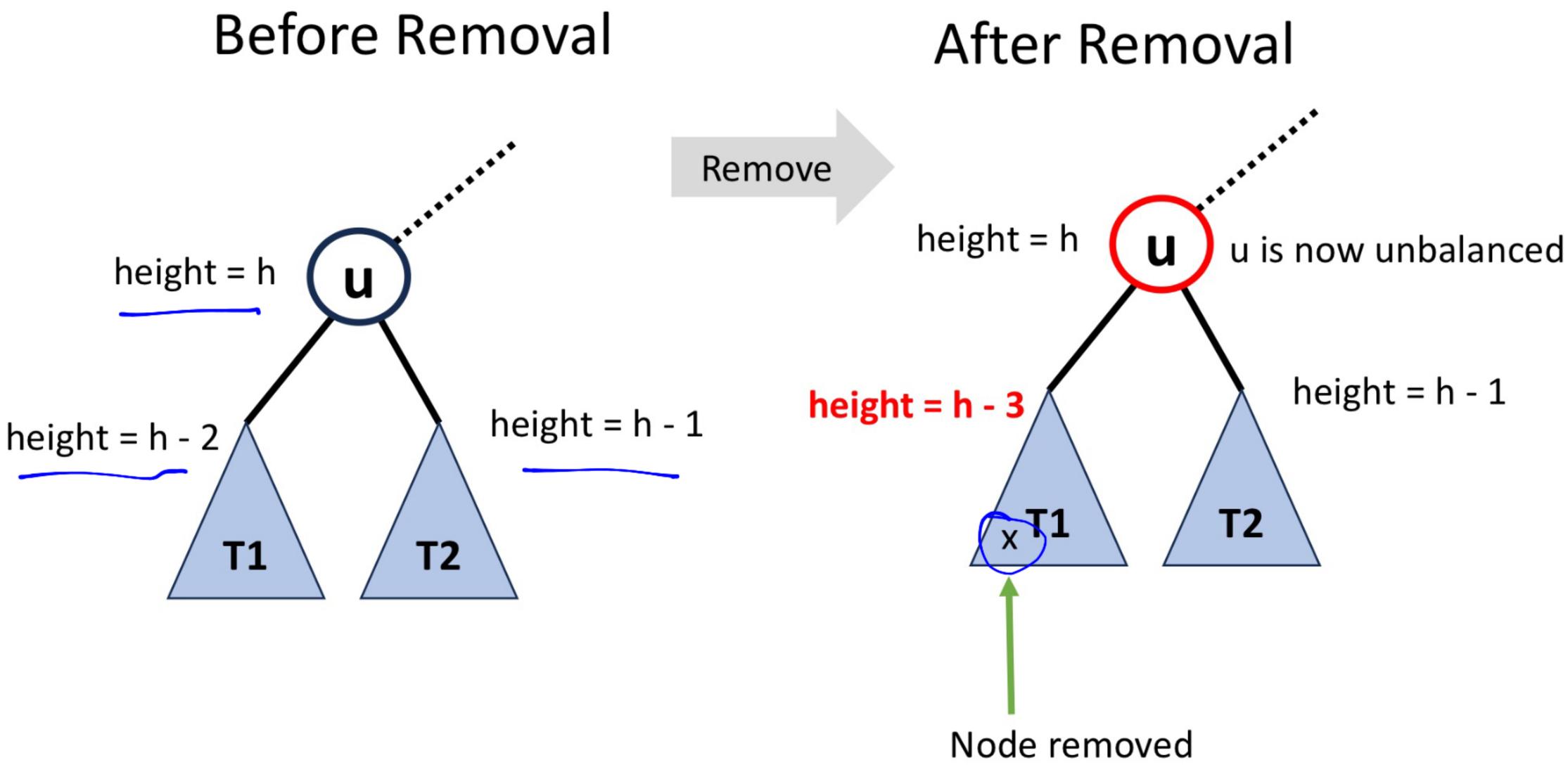
Removal in AVL Trees

- Remove a node in an AVL tree
- Case 1: Node to be removed is a leaf
- Case 2: Node to be removed has only one child
  - In this case, that child must be a leaf
- In AVL trees either remove a leaf (case 1) or parent of a leaf (case 2)

Presume case 2 to first copy from leaf node to its parent and then deletion of leaf node



Maintaining Height Balance After Removal



Maintaining Height Balance After Removal

Remove 1

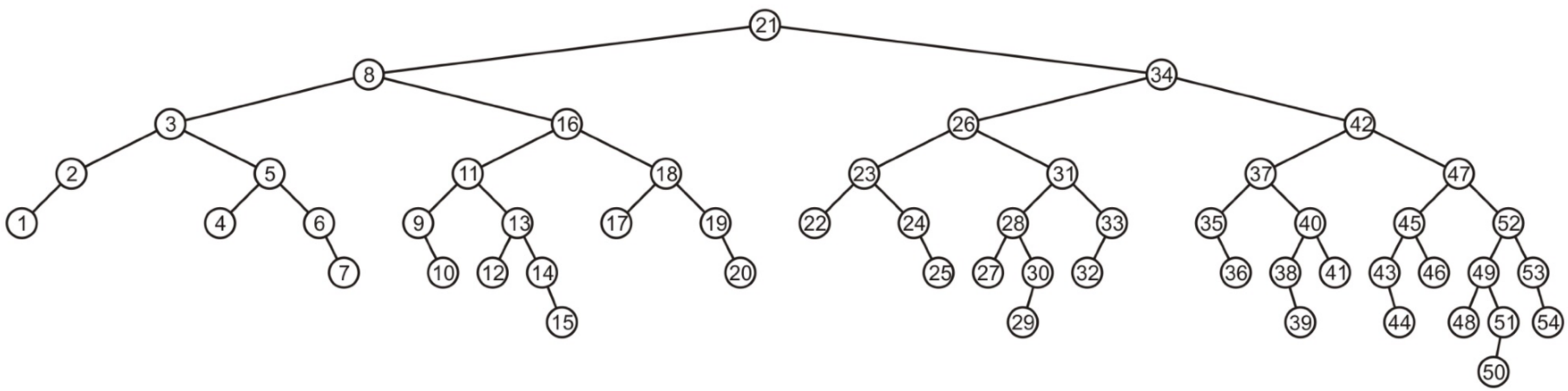


Figure courtesy: Douglas Wilhelm Harder, University of Waterloo, Canada

Maintaining Height Balance After Removal

Remove 1

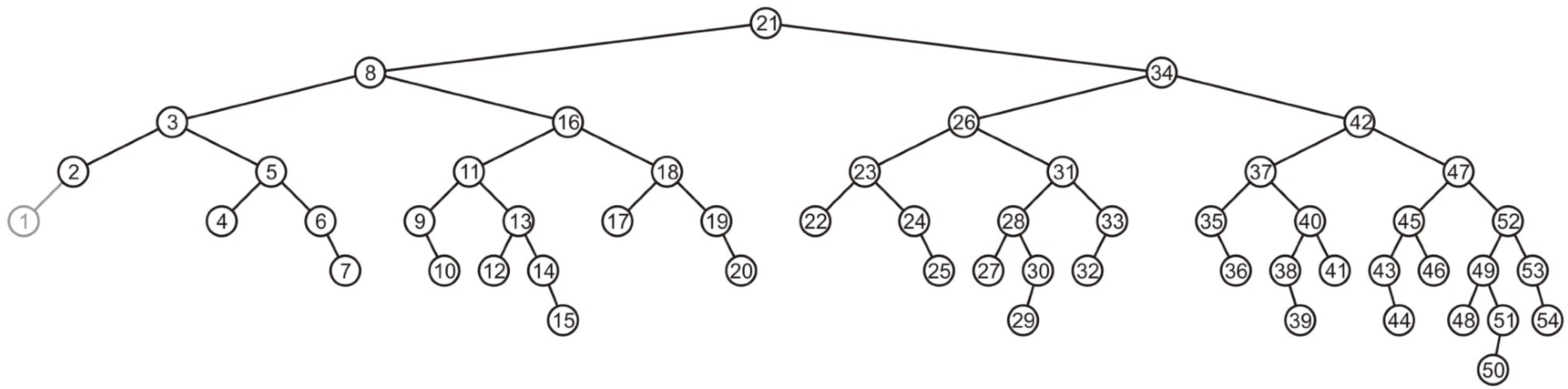


Figure courtesy: Douglas Wilhelm Harder, University of Waterloo, Canada

Maintaining Height Balance After Removal

- Remove 1

Patent of 1 which is 2, is not unbalanced, its grandparent 3 is imbalanced

The right-right subtree has more height

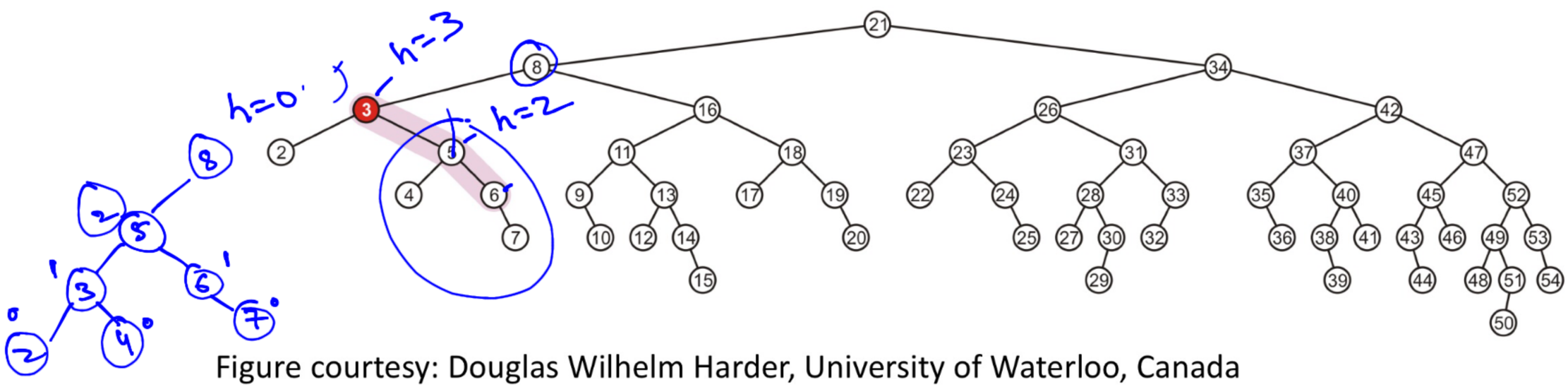


Figure courtesy: Douglas Wilhelm Harder, University of Waterloo, Canada

Maintaining Height Balance After Removal

Tree can be balanced by a rotation

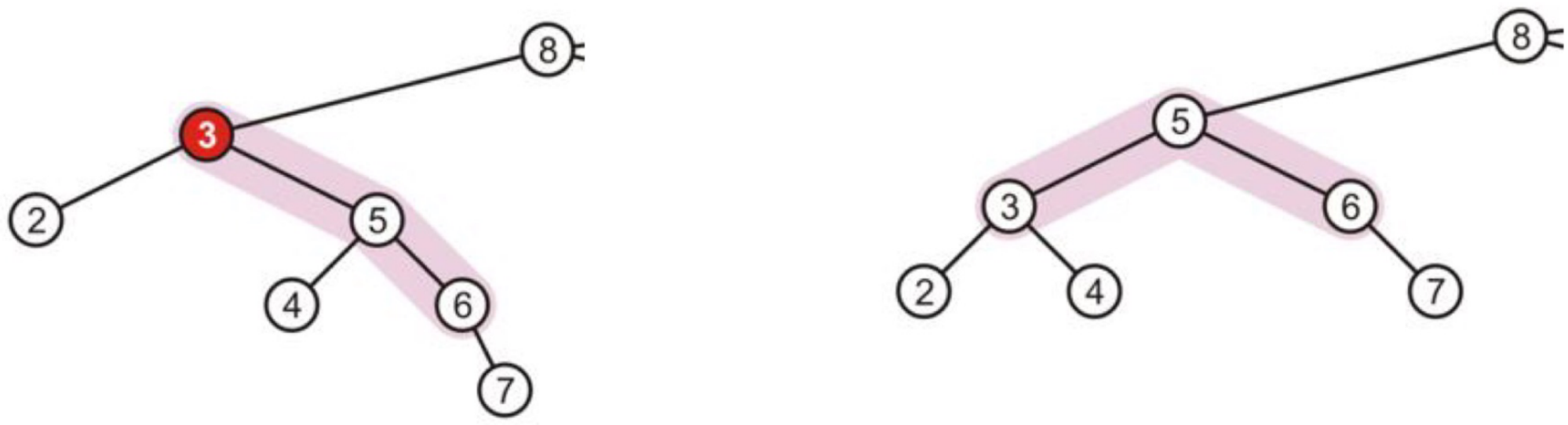


Figure courtesy: Douglas Wilhelm Harder, University of Waterloo, Canada

Maintaining Height Balance After Removal

More nodes become unbalanced after rotation

8 is now unbalanced

Right-left unbalance

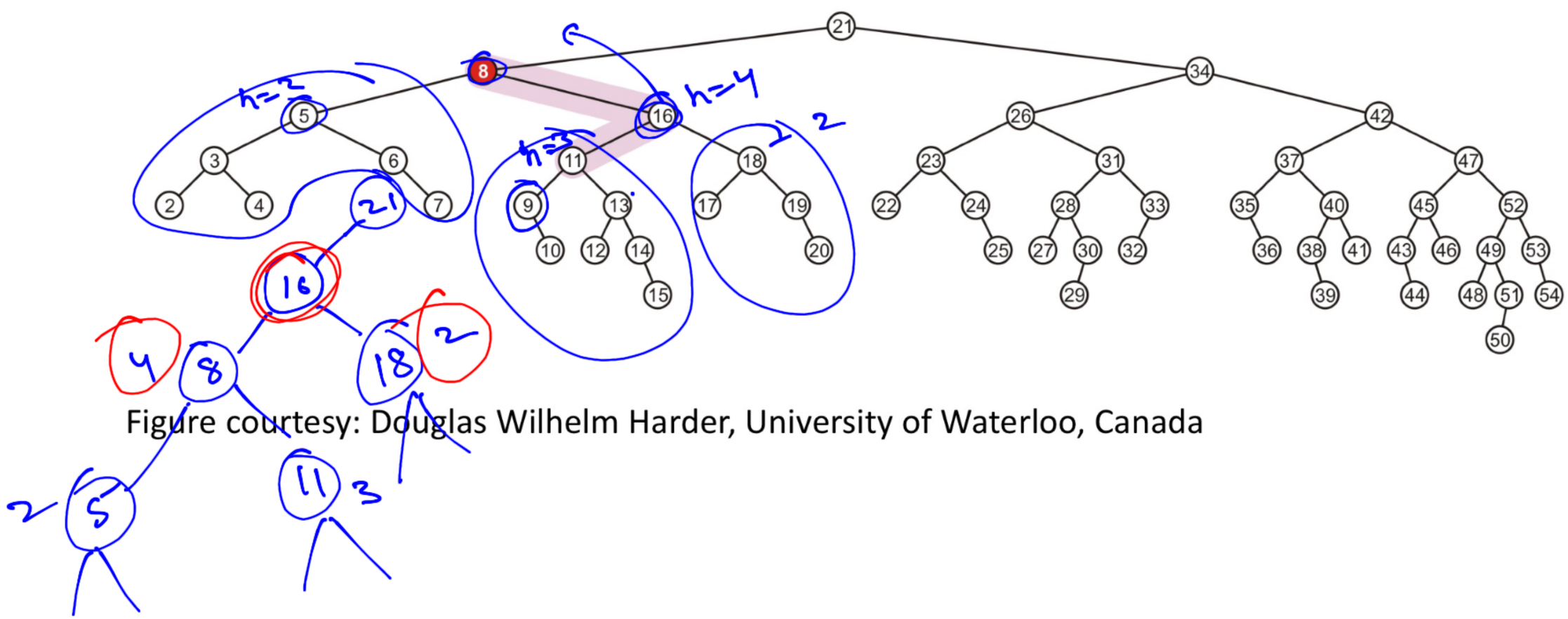


Figure courtesy: Douglas Wilhelm Harder, University of Waterloo, Canada

Maintaining Height Balance After Removal

More nodes become unbalanced after rotation

8 is now unbalanced

Moving 11 up; moving 8 down; splitting the tree rooted at 11 balances the tree

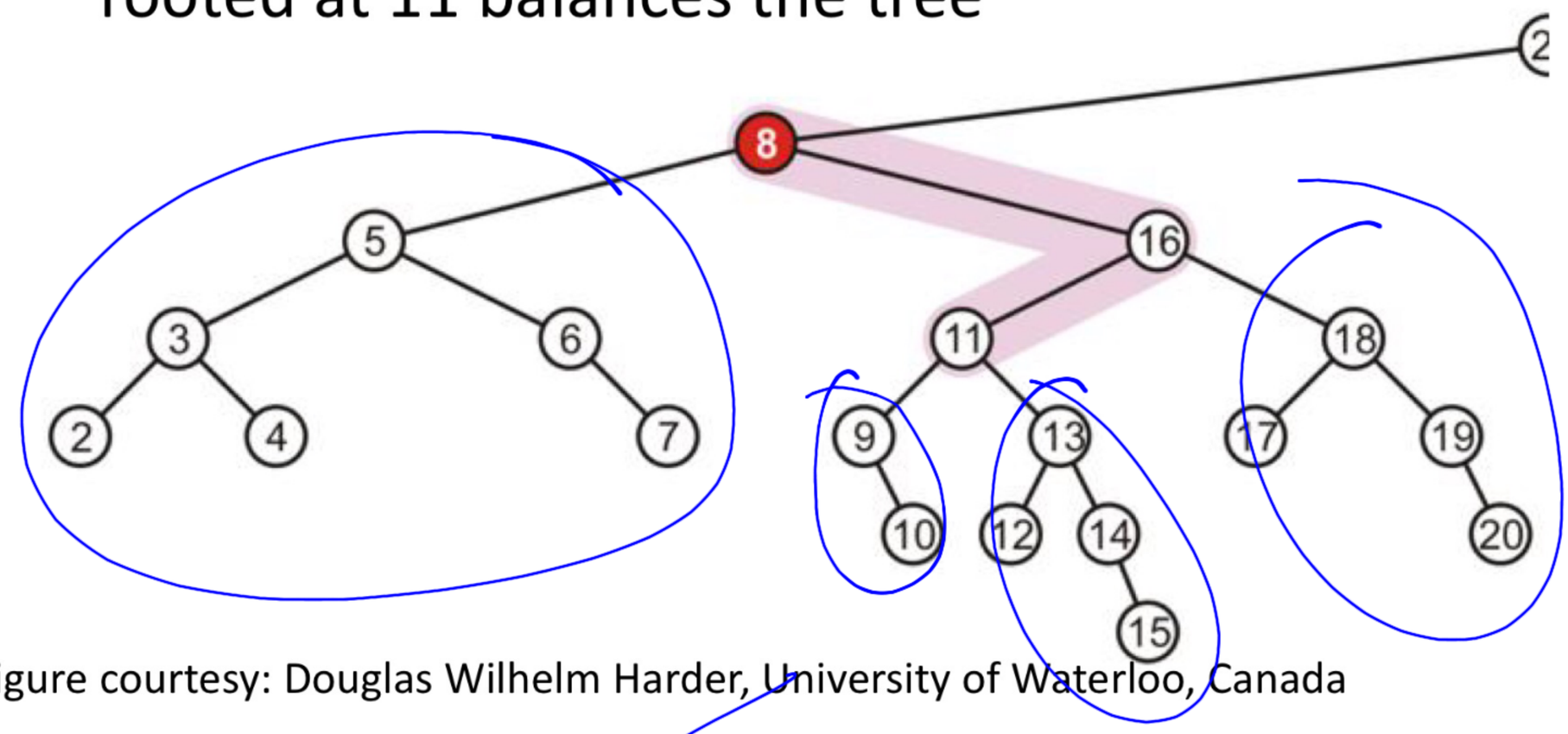
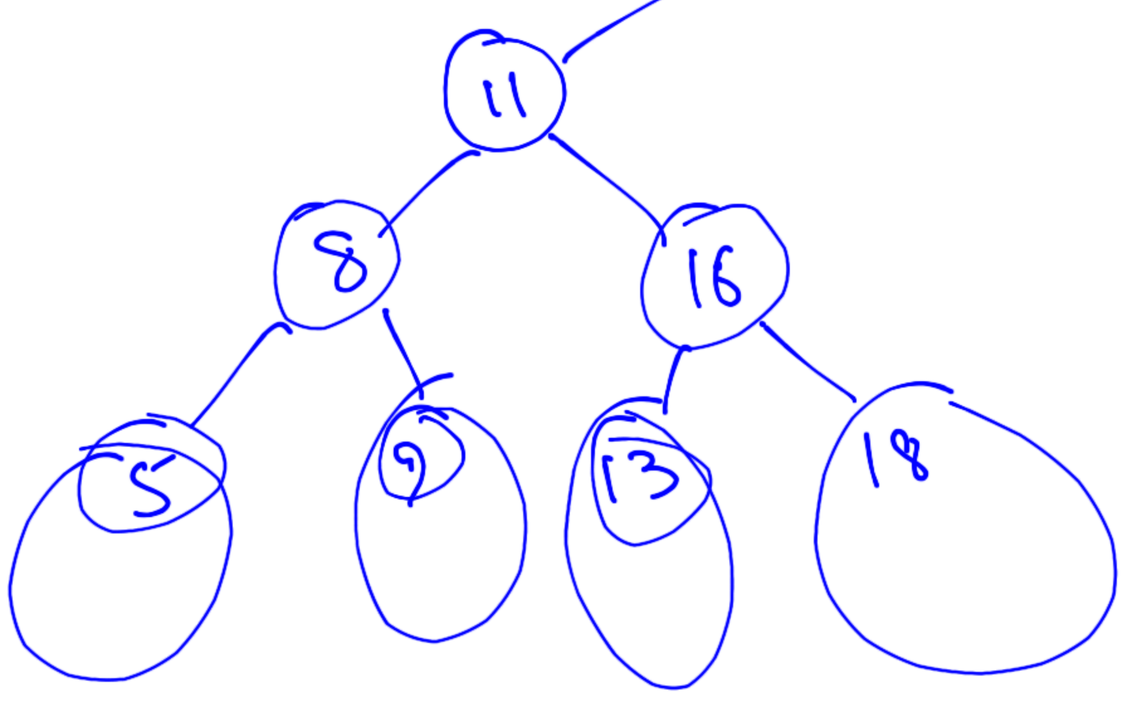


Figure courtesy: Douglas Wilhelm Harder, University of Waterloo, Canada



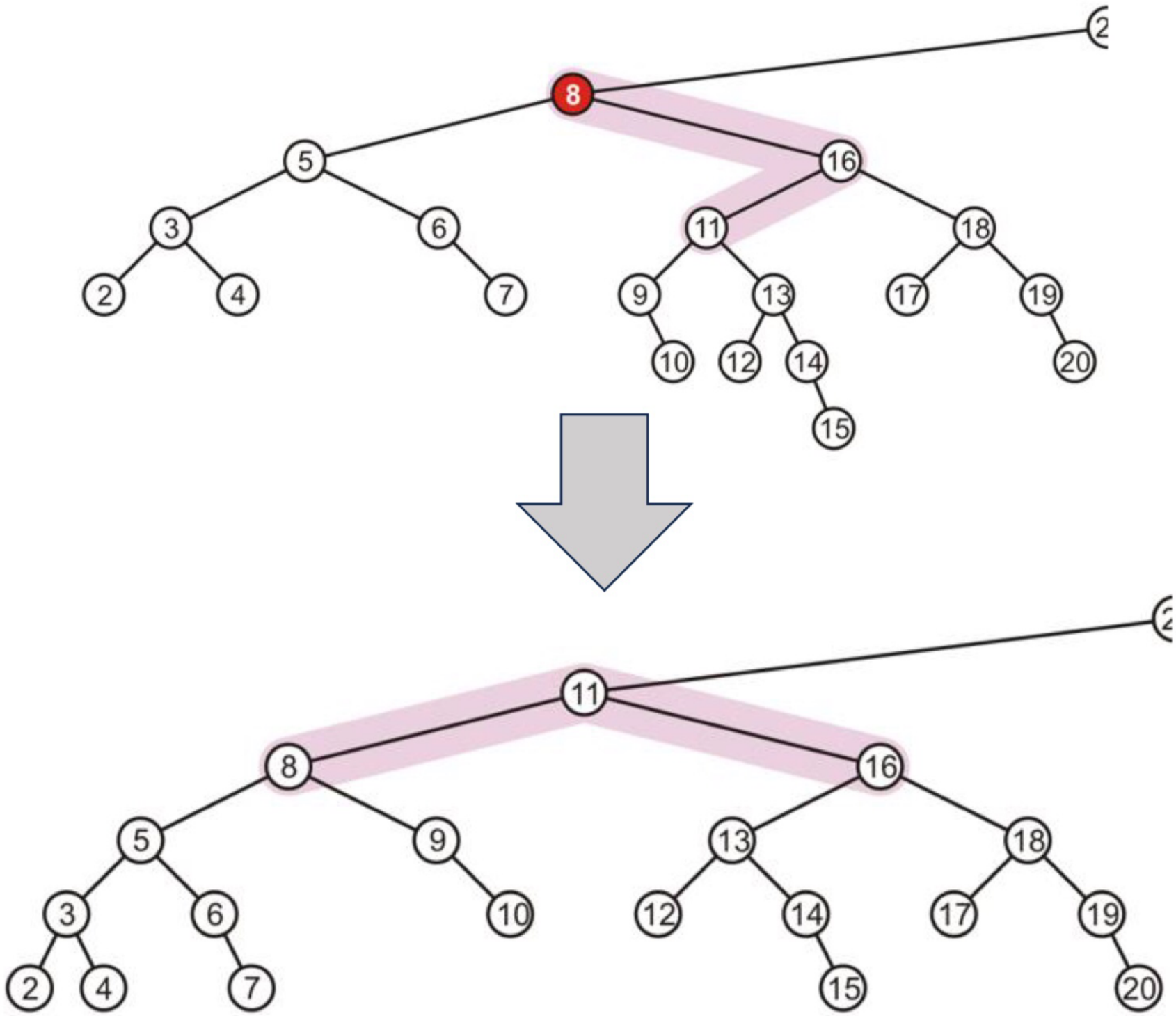


Figure courtesy: Douglas Wilhelm Harder, University of Waterloo, Canada

Maintaining Height Balance After Removal

Now root becomes imbalanced

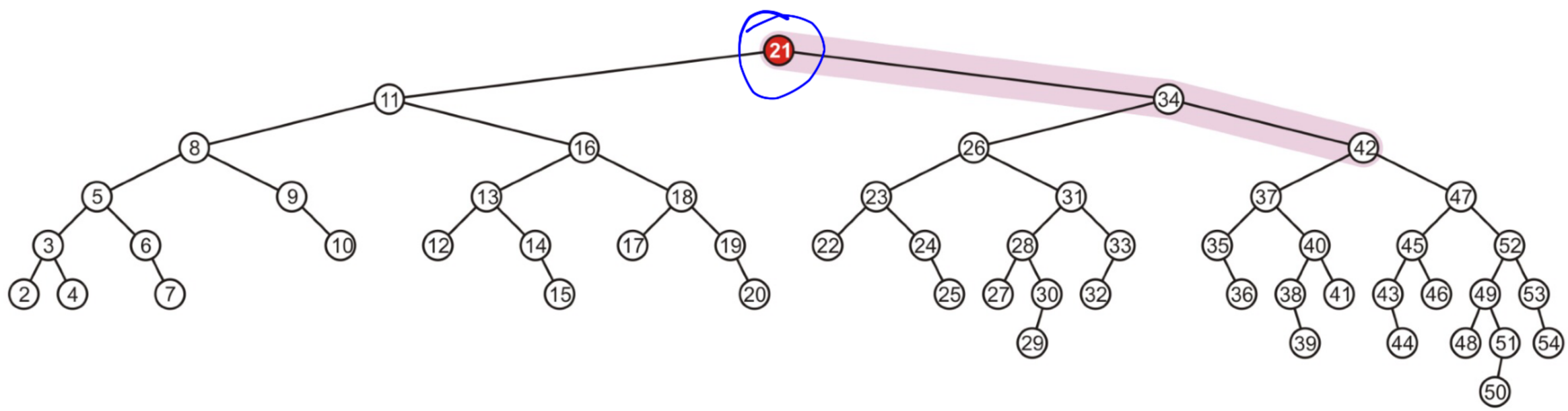


Figure courtesy: Douglas Wilhelm Harder, University of Waterloo, Canada

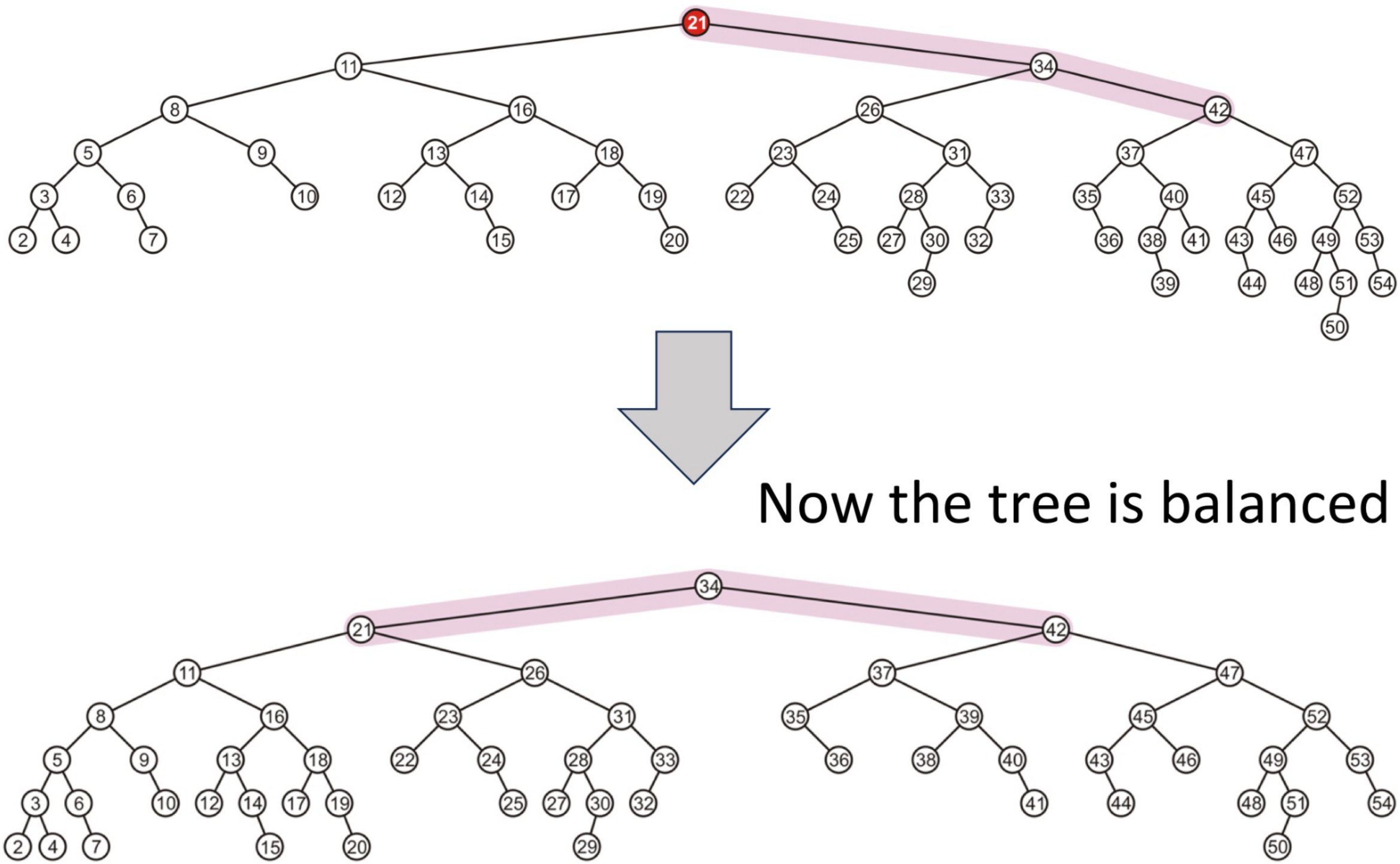
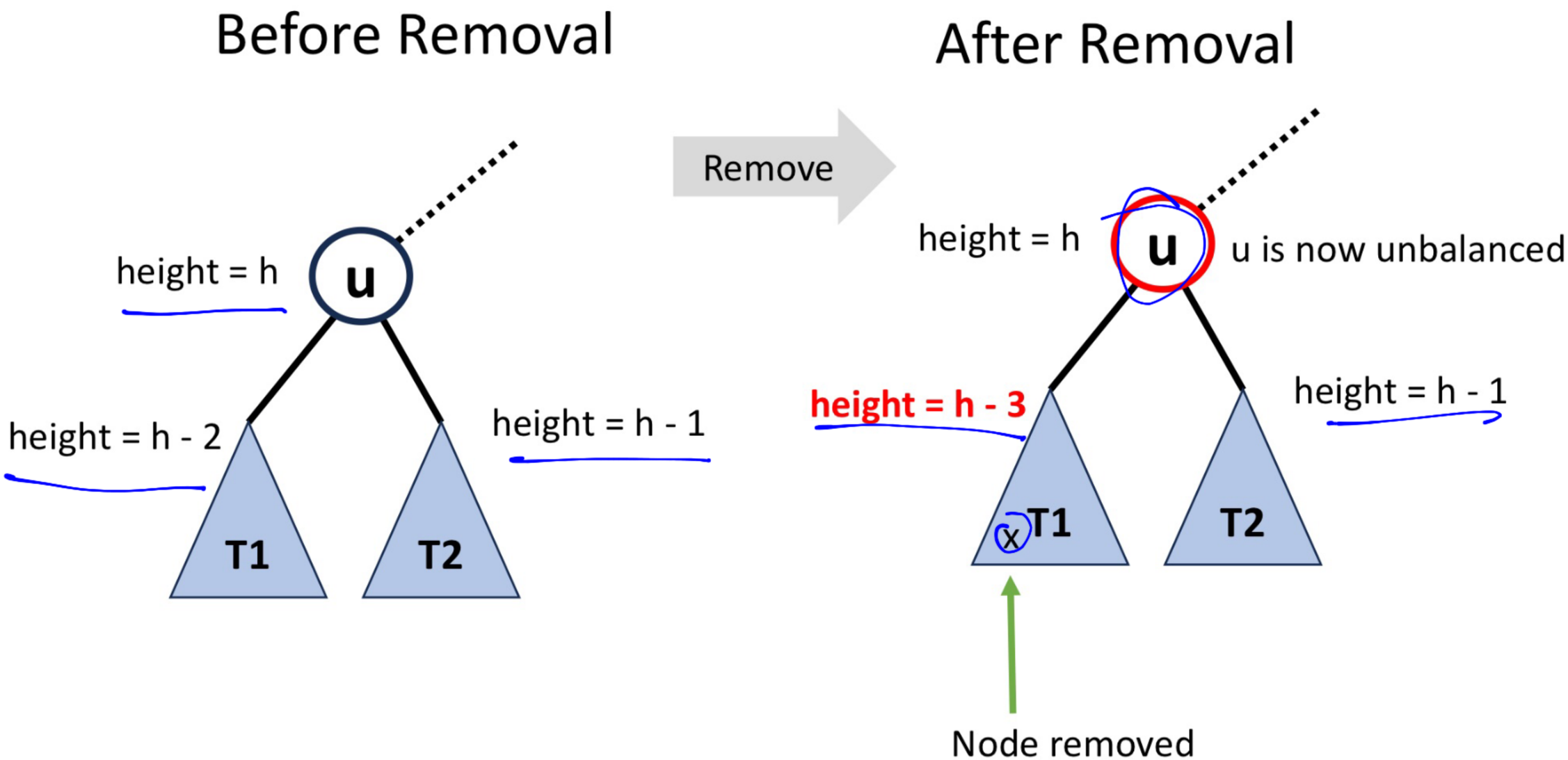


Figure courtesy: Douglas Wilhelm Harder, University of Waterloo, Canada

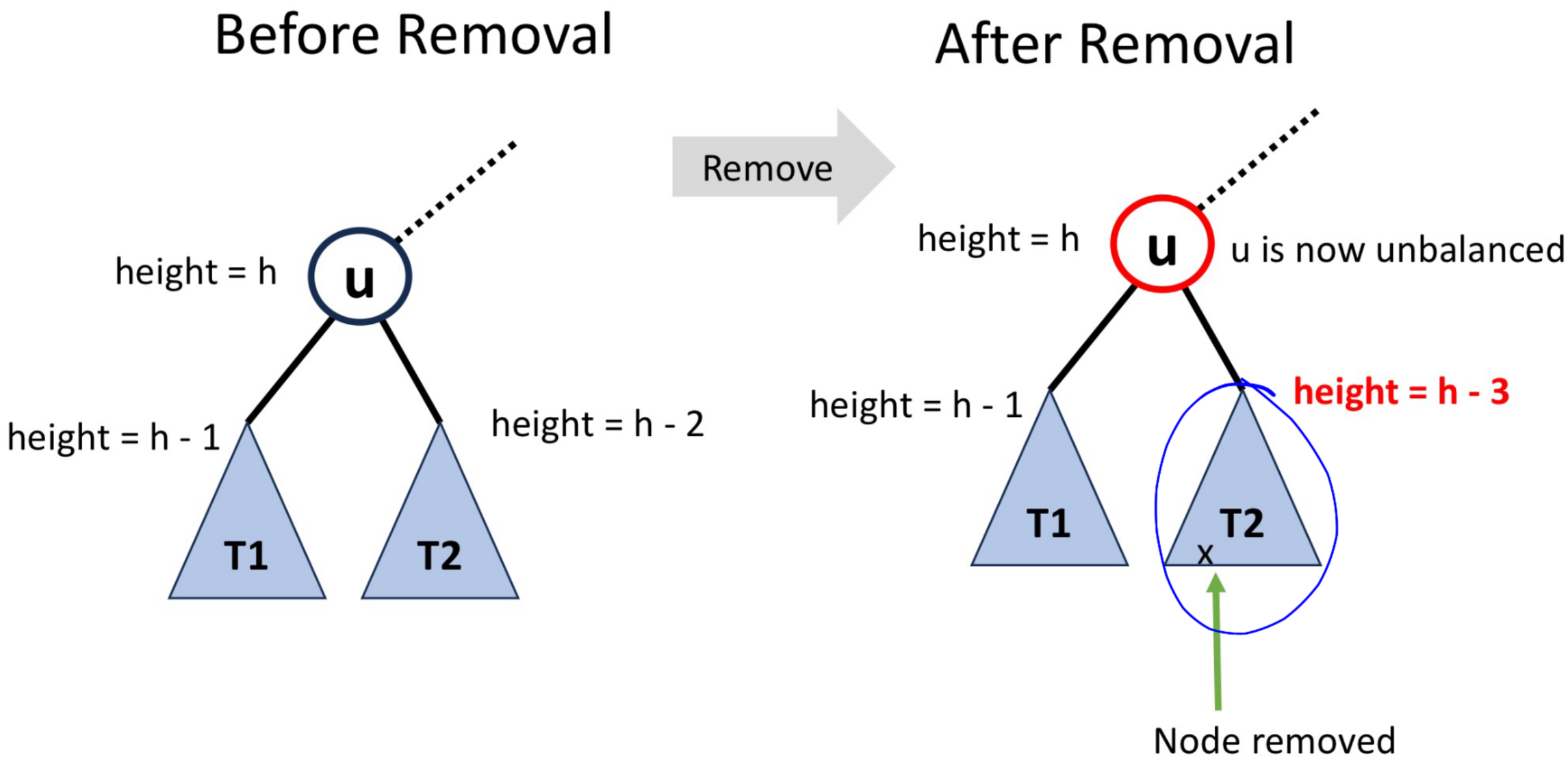
AVT Tree Deletion

- Let u be the first unbalanced node encountered while travelling up the tree from the deleted node
- Perform rotations to restore balance at the subtree rooted at u
- As this restructuring may upset the balance of another node higher in the tree
- We must continue checking for balance until the root of T is reached

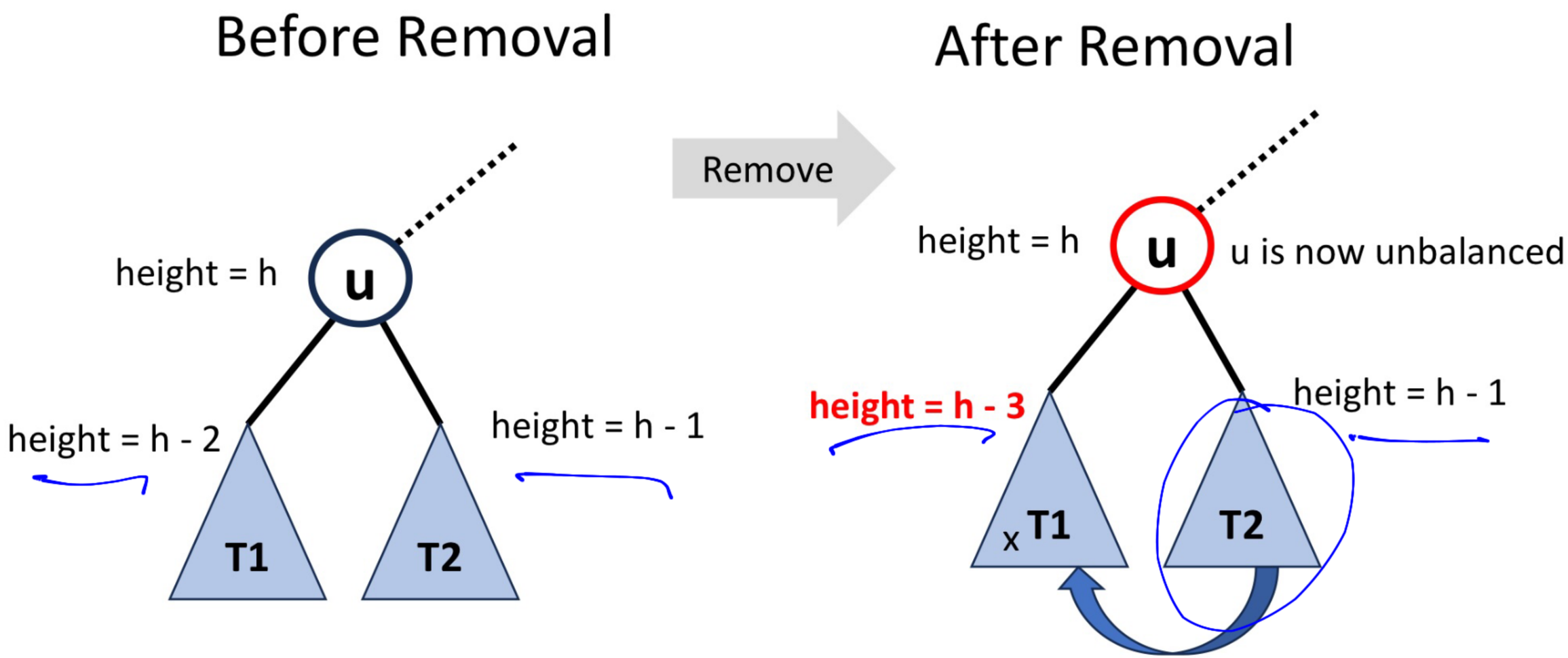
AVL Tree Removal Balancing



AVL Tree Removal Balancing (Symmetric Case – Homework)



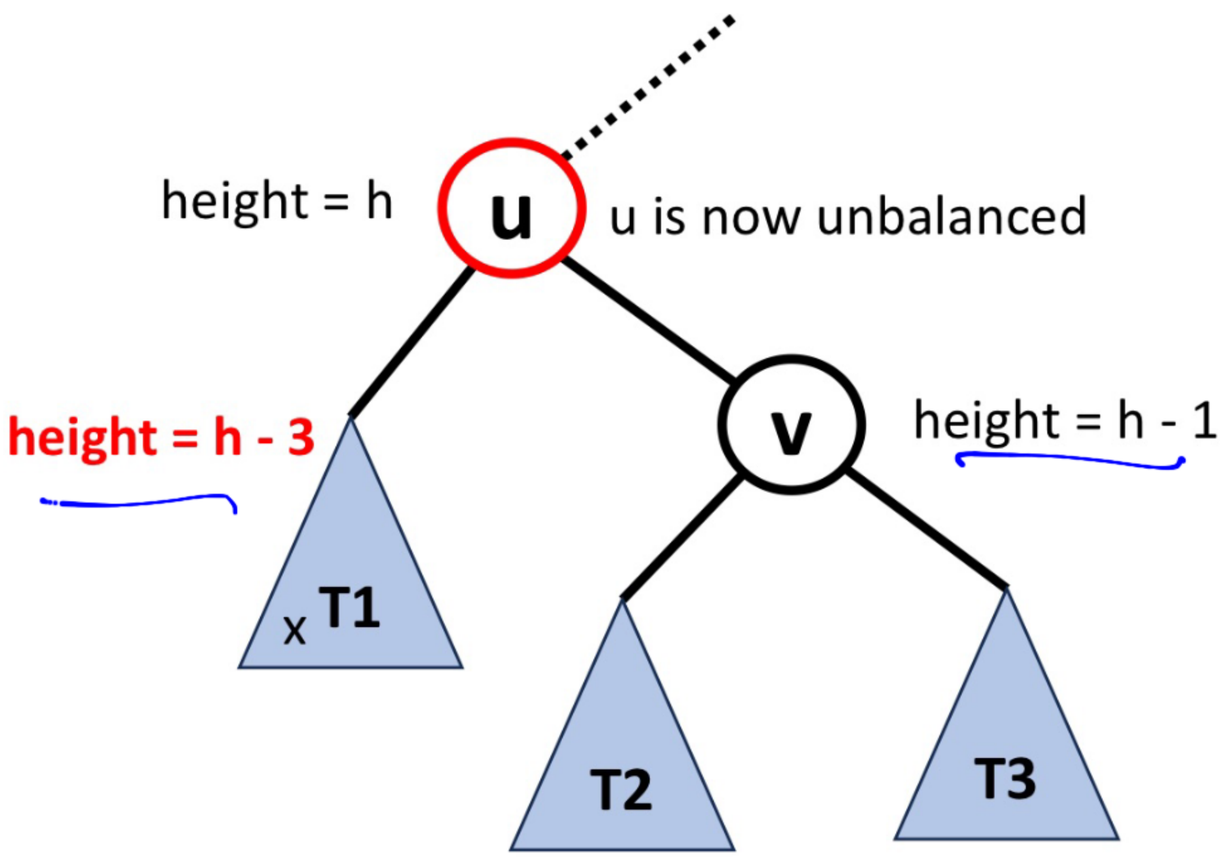
AVL Tree Removal Balancing



Break open tree T2
Transfer to T1

AVL Tree Removal Balancing

After Removal



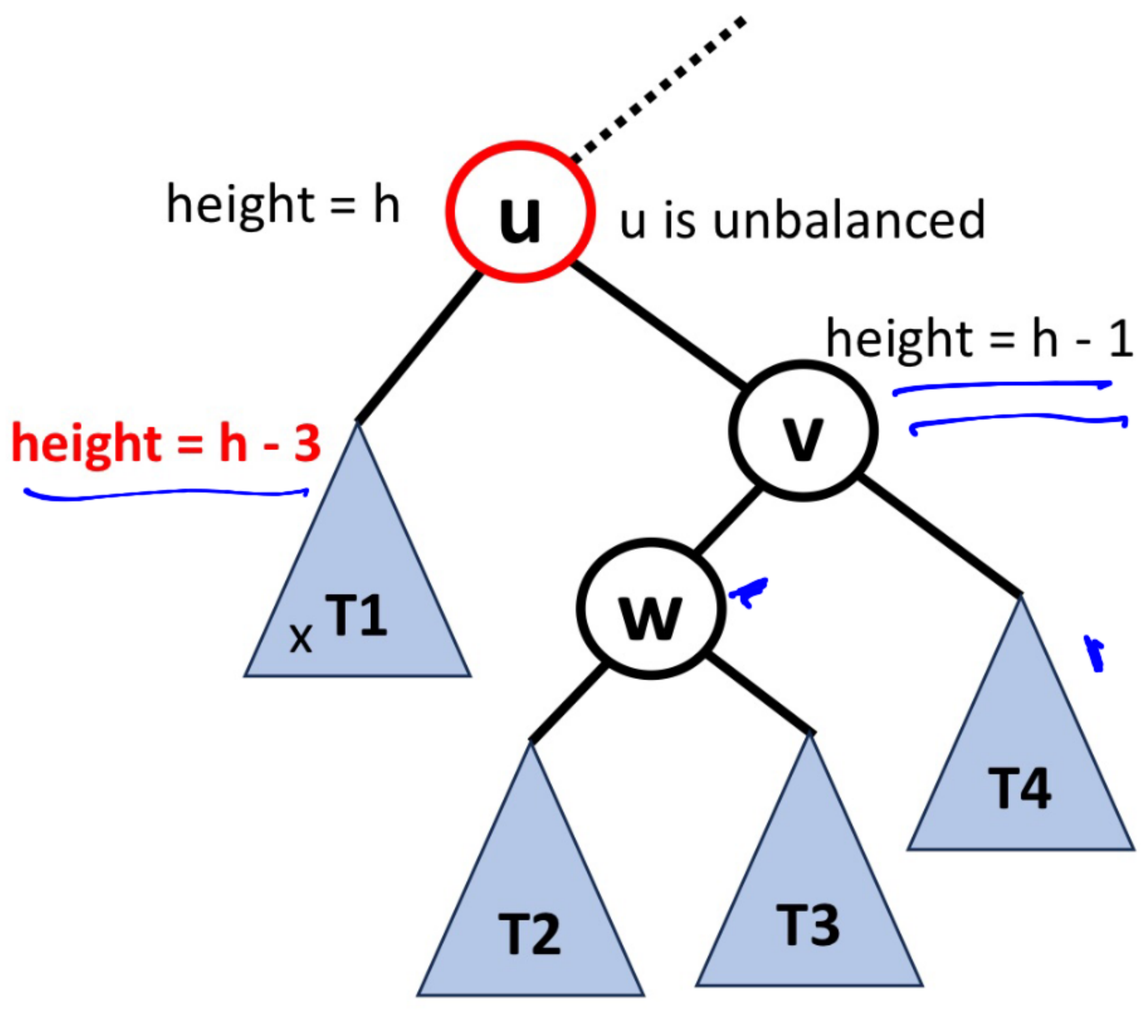
How to rebalance?

Should we break further?
Break T2 or T3?

Break open subtree(v)
Transfer to T1

AVL Tree Removal Balancing

After Removal



How to rebalance?

3 sub-cases at node v

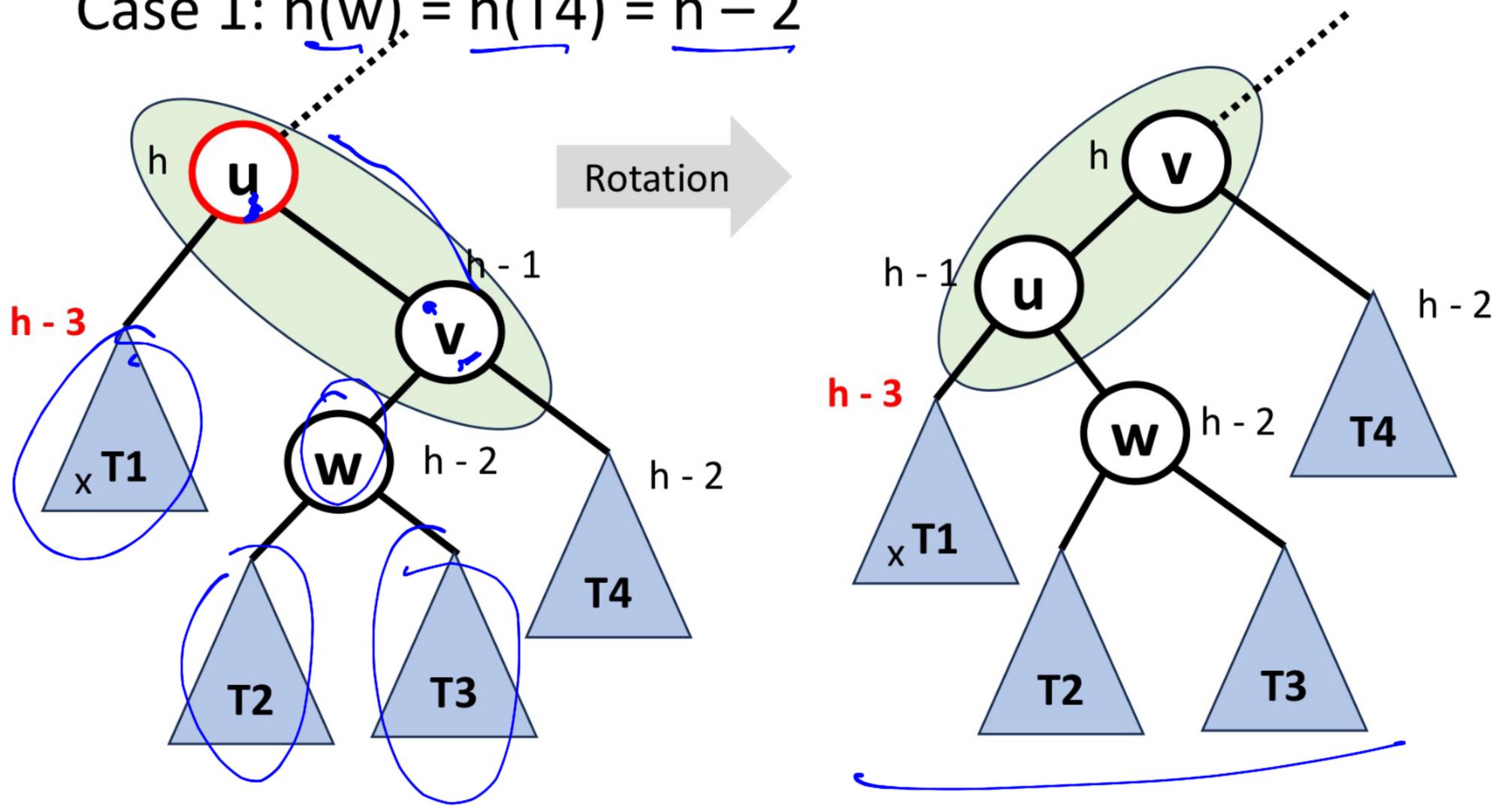
Depend on h(w) and h (T4)

1. h(w) = h(T4) = h – 2
2. h(w) = h – 3; h(T4) = h – 2
3. h(w) = h – 2; h(T4) = h – 3

AVL Tree Removal Balancing

keys(T1)	u	keys(T2)	w	keys(T3)	v	keys(T4)

Case 1: h(w) = h(T4) = h – 2

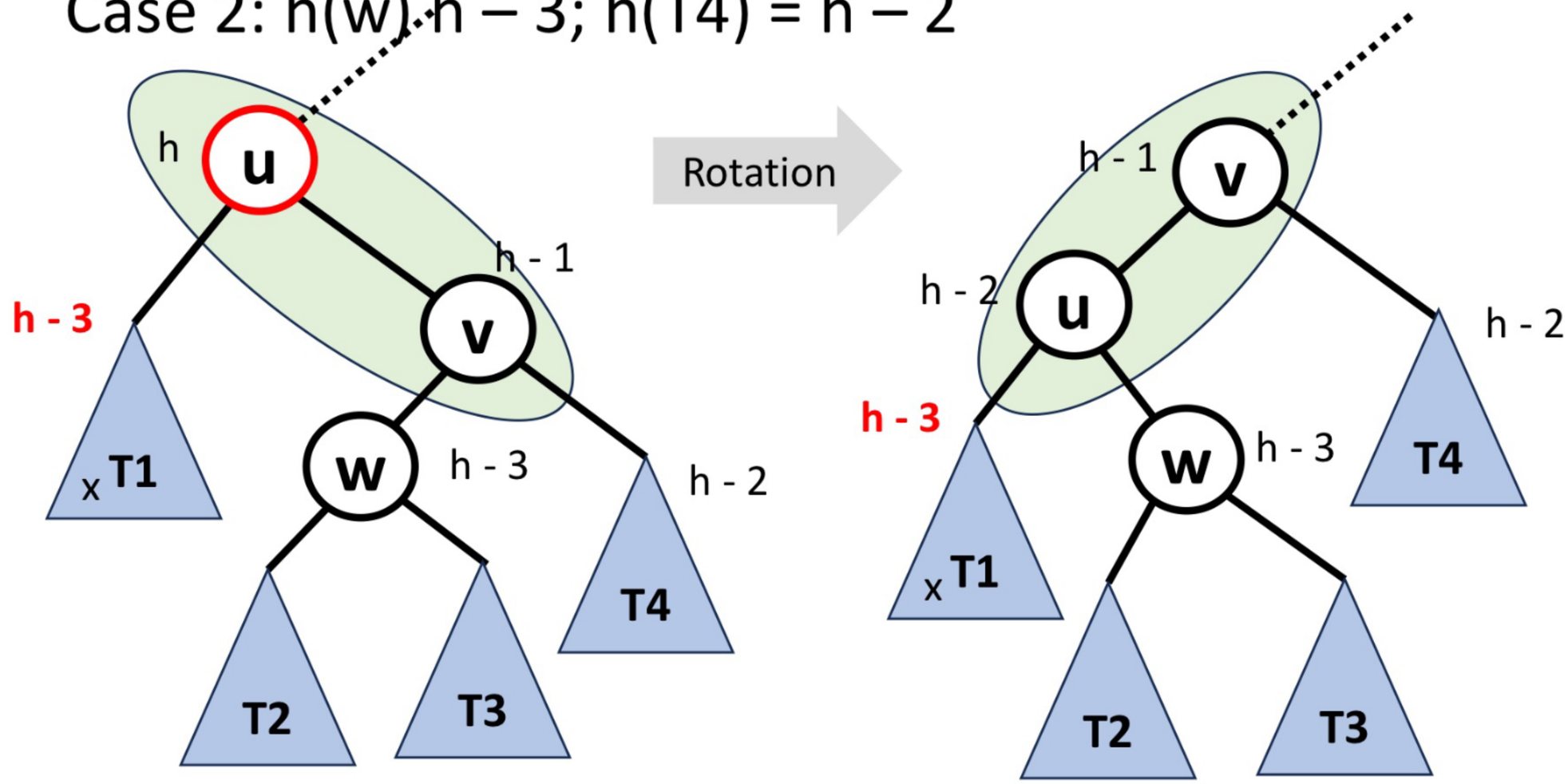


No further rotations are needed. Entire tree is balanced

AVL Tree Removal Balancing

keys(T1)	u	keys(T2)	w	keys(T3)	v	keys(T4)

Case 2: h(w) h – 3; h(T4) = h – 2

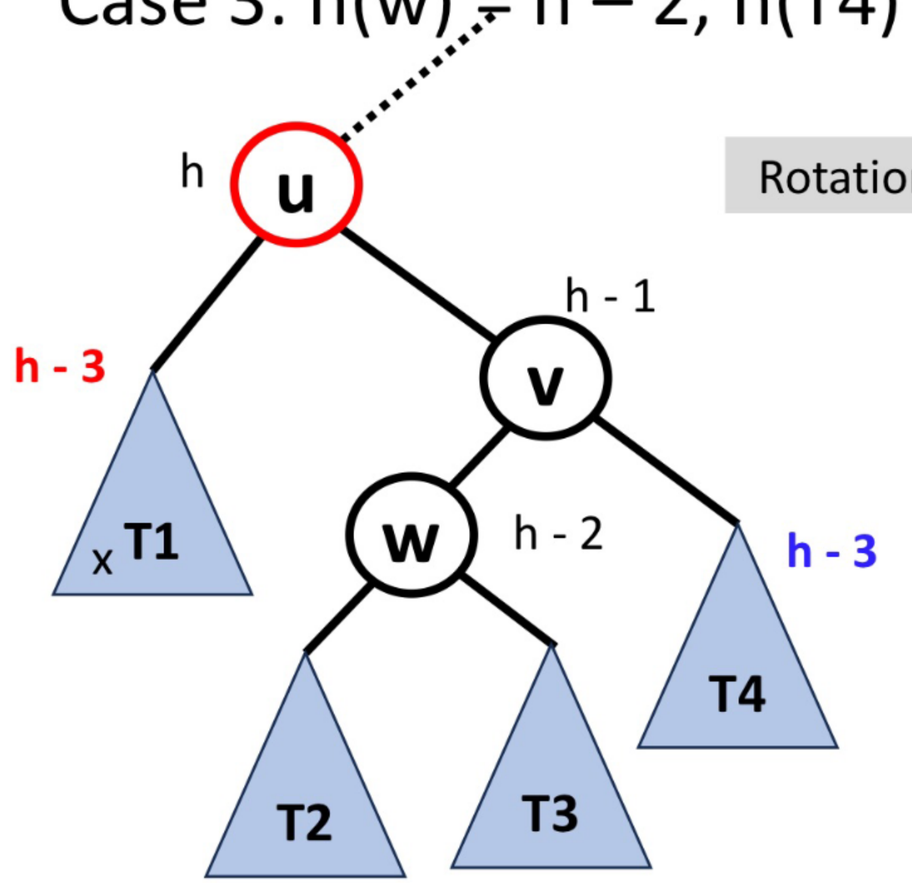


Note that height of the subtree has decreased. Need to propagate height and check balance at parents.

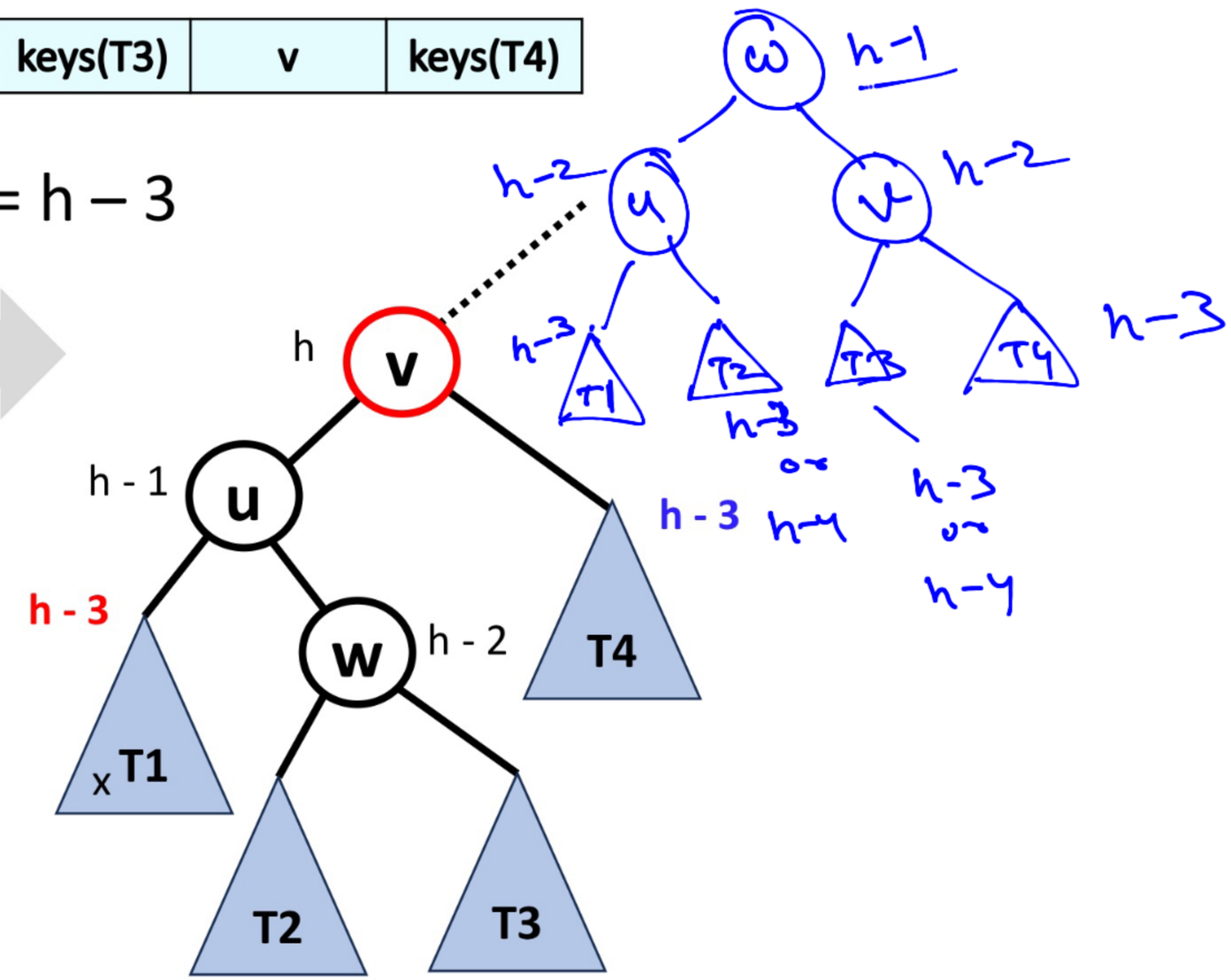
AVL Tree Removal Balancing

keys(T1)	u	keys(T2)	w	keys(T3)	v	keys(T4)

Case 3: h(w) = h – 2; h(T4) = h – 3



Rotation

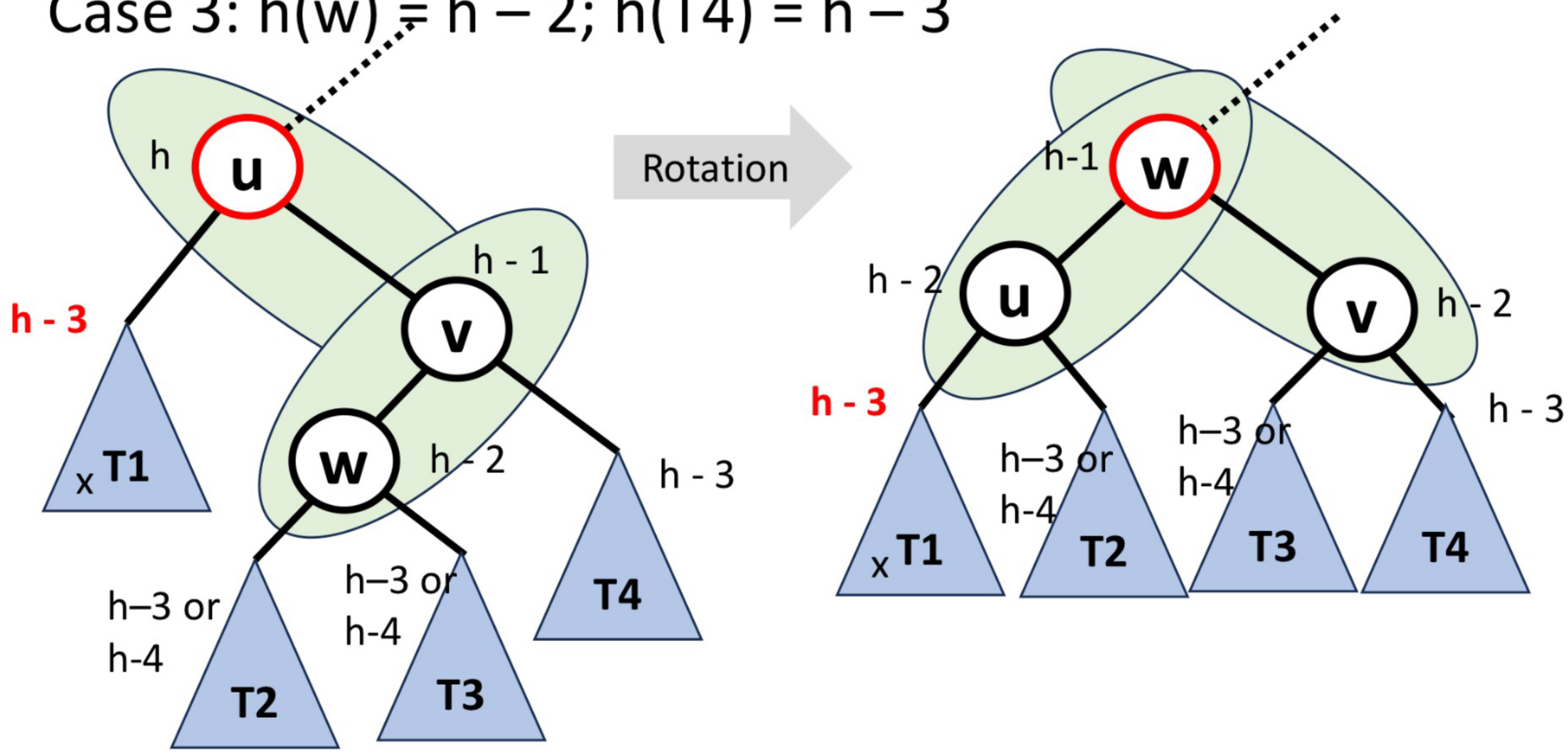


The same rotation will not work as node v will become unbalanced

AVL Tree Removal Balancing

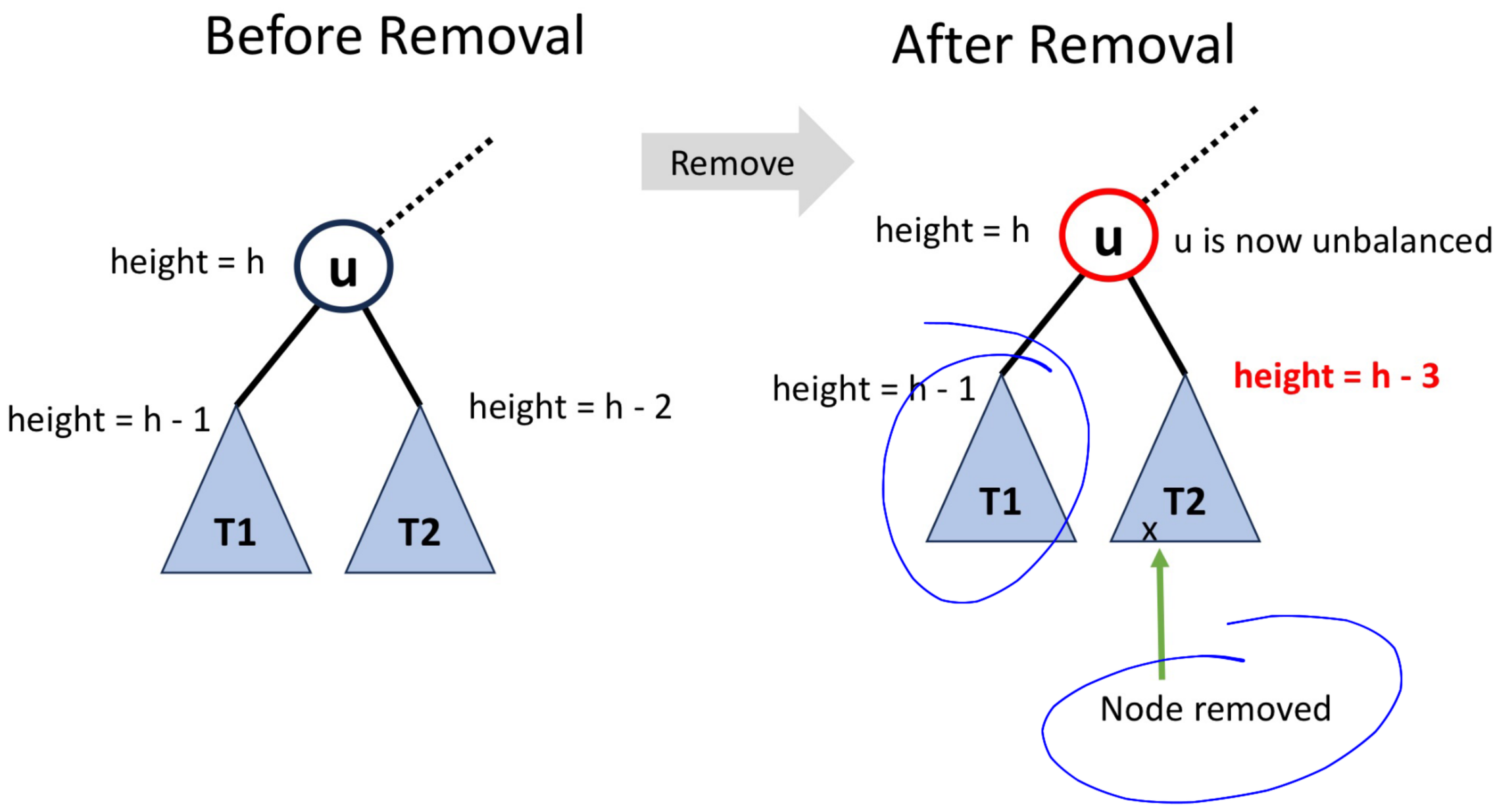
keys(T1)	u	keys(T2)	w	keys(T3)	v	keys(T4)

Case 3: h(w) = h – 2; h(T4) = h – 3



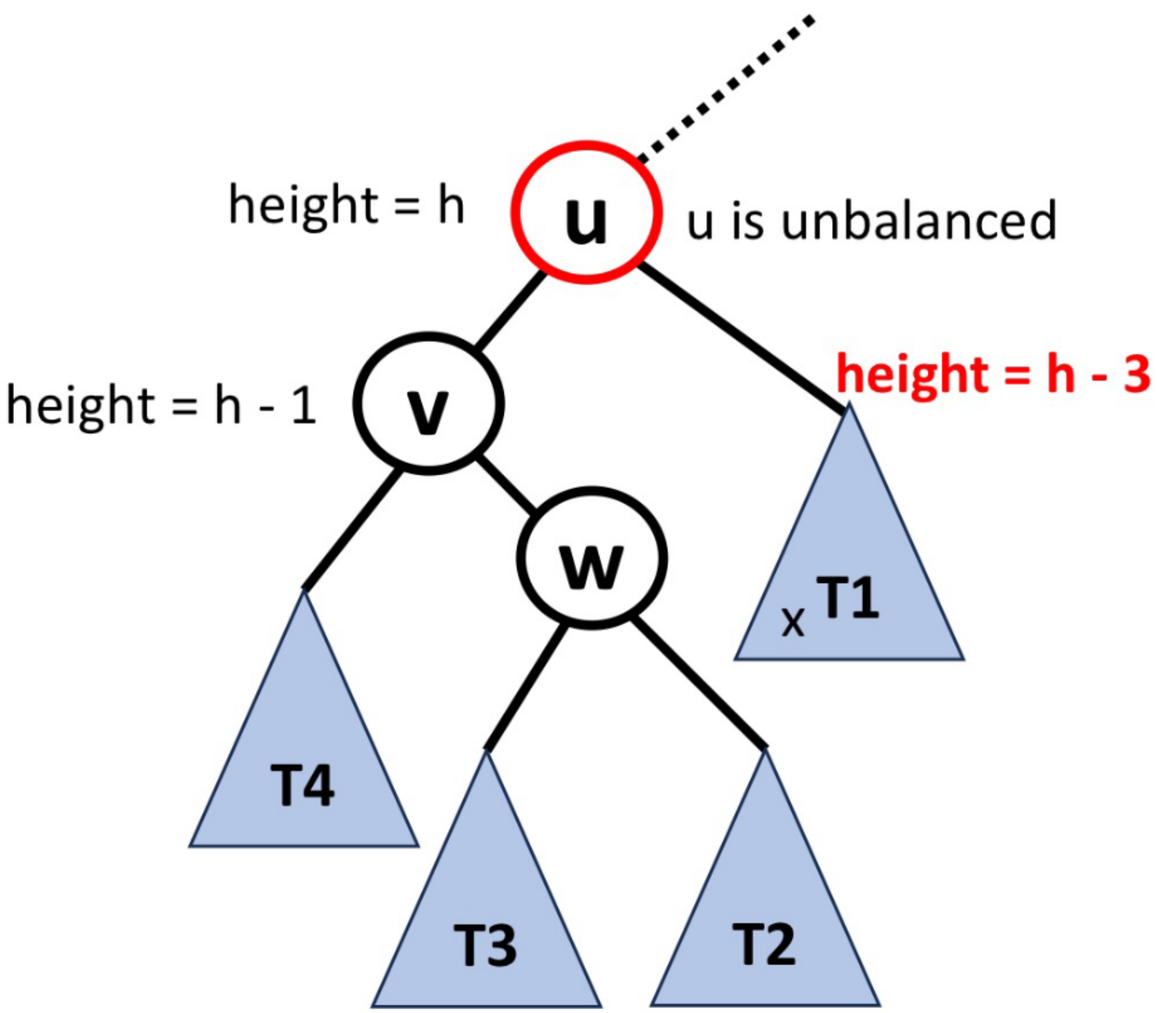
Note that height of the subtree has decreased. Need to propagate height and check balance at parents.

Symmetric Case – Homework



Symmetric Case – Homework

After Removal



How to rebalance?

3 sub-cases at node v

Depend on h(w) and h (T4)

1. h(w) = h(T4) = h – 2
2. h(w) = h – 3; h(T4) = h – 2
3. h(w) = h – 2; h(T4) = h – 3

Thank You