Balanced BST

Balanced BST

- Goal: BST with $\Theta(\log n)$ height.

Balanced BST

- Goal: BST with $\Theta(\log n)$ height.
- Method: Balanced BSTs

Balanced BST

- Goal: BST with $\Theta(\log n)$ height.
- Method: Balanced BSTs
- Last lecture:
  - ✓ Defined AVL trees

Balanced BST

- Goal: BST with $\Theta(\log n)$ height.
- Method: Balanced BSTs
- Last lecture:
  - ✓ Defined AVL trees
  - ✓ Explained how to insert a key into an AVL tree:
    - ✓ High-level procedure
    - ✓ Examples to illustrate this procedure

Balanced BST

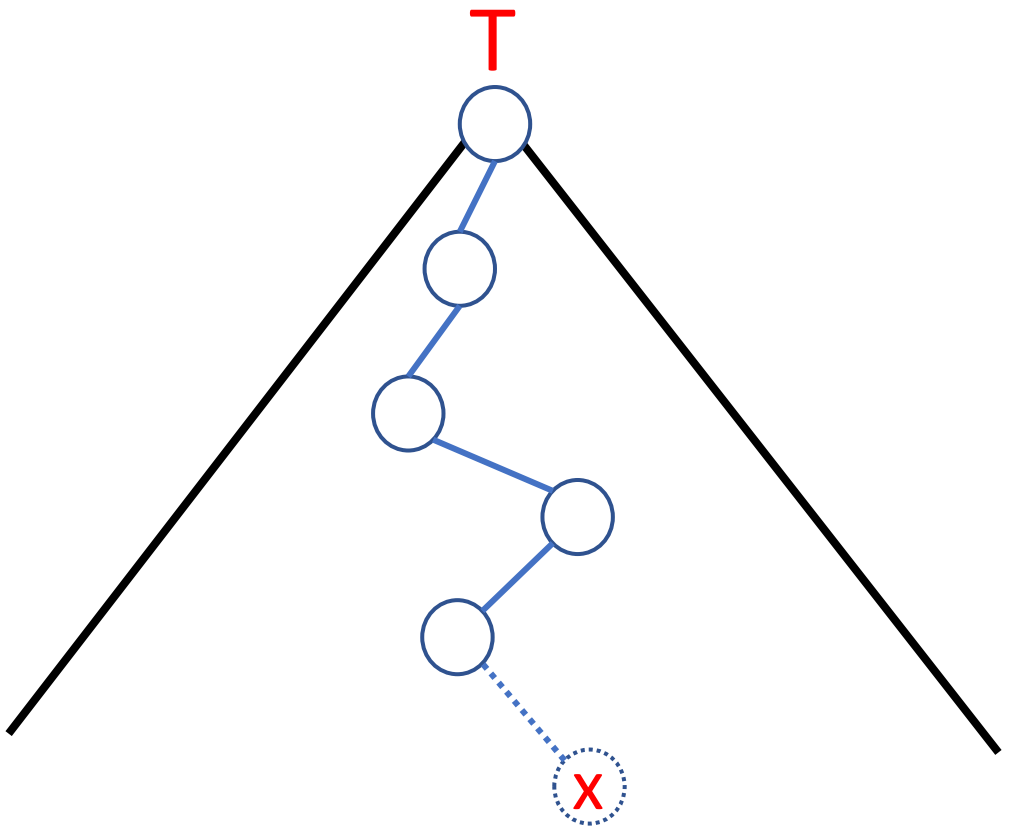
- Goal: BST with $\Theta(\log n)$ height.
- Method: Balanced BSTs
- Last lecture:
  - ✓ Defined AVL trees
  - ✓ Explained how to insert a key into an AVL tree:
    - ✓ High-level procedure
    - ✓ Examples to illustrate this procedure
  - ✓ From these examples we ``extracted'' an algorithm

Balanced BST

- Goal: BST with $\Theta(\log n)$ height.
- Method: Balanced BSTs
- Last lecture:
  - ✓ Defined AVL trees
  - ✓ Explained how to insert a key into an AVL tree:
    - ✓ High-level procedure
    - ✓ Examples to illustrate this procedure
  - ✓ From these examples we ``extracted'' an algorithm
- This lecture:
  - ✓ We show this algorithm works in all cases

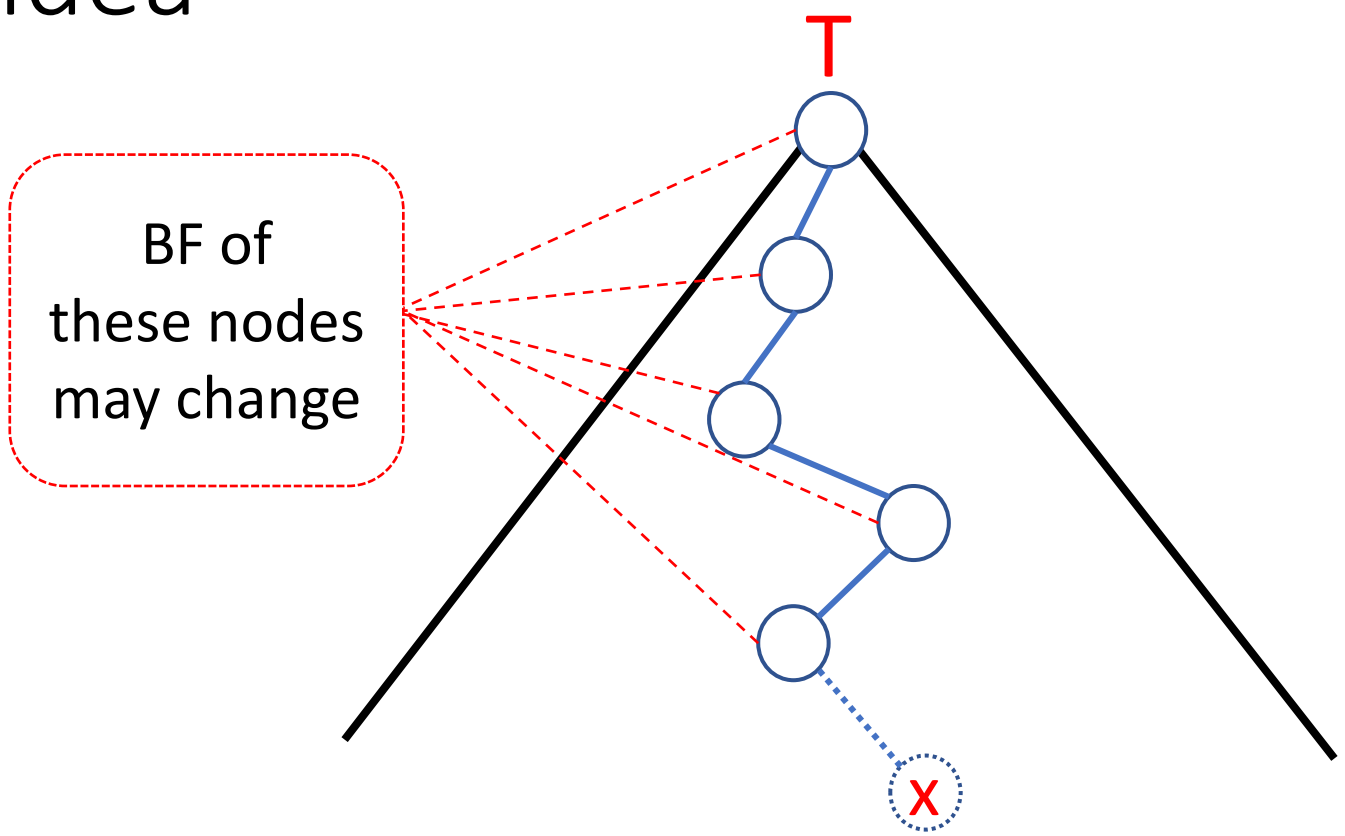
Insert(T, x) : General idea

- Insert x into T as in any BST :
  - x is now a leaf



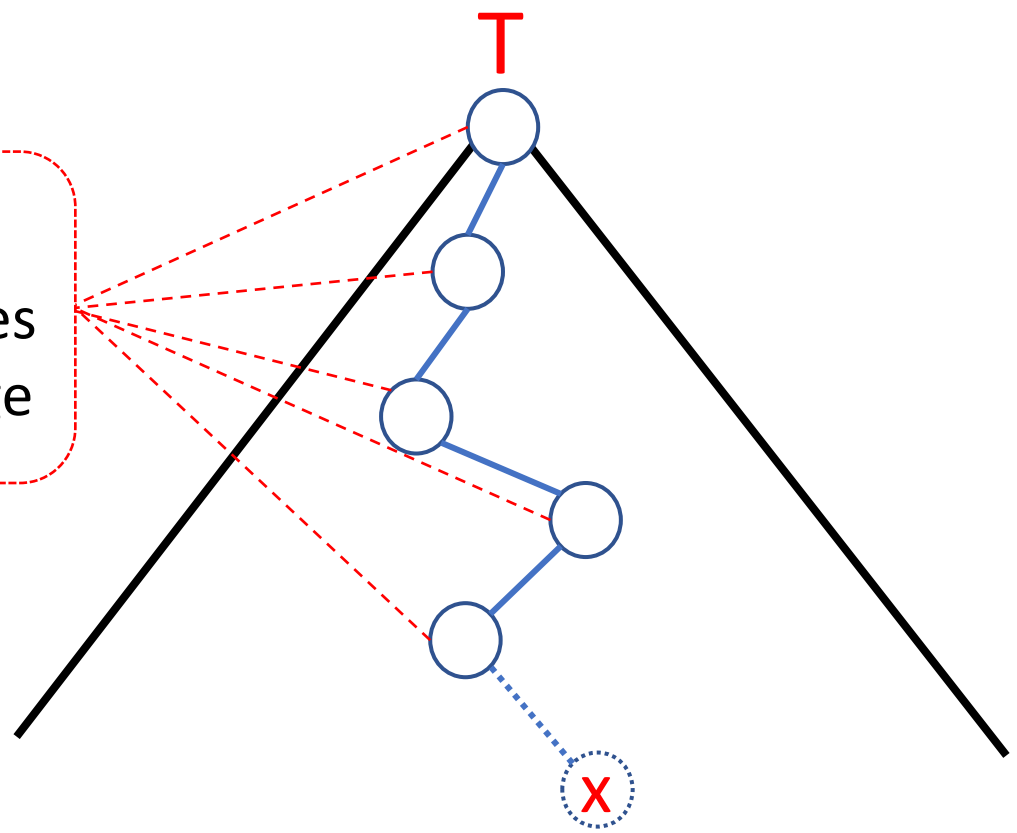
Insert(T, x) : General idea

- Insert x into T as in any BST :
  - x is now a leaf



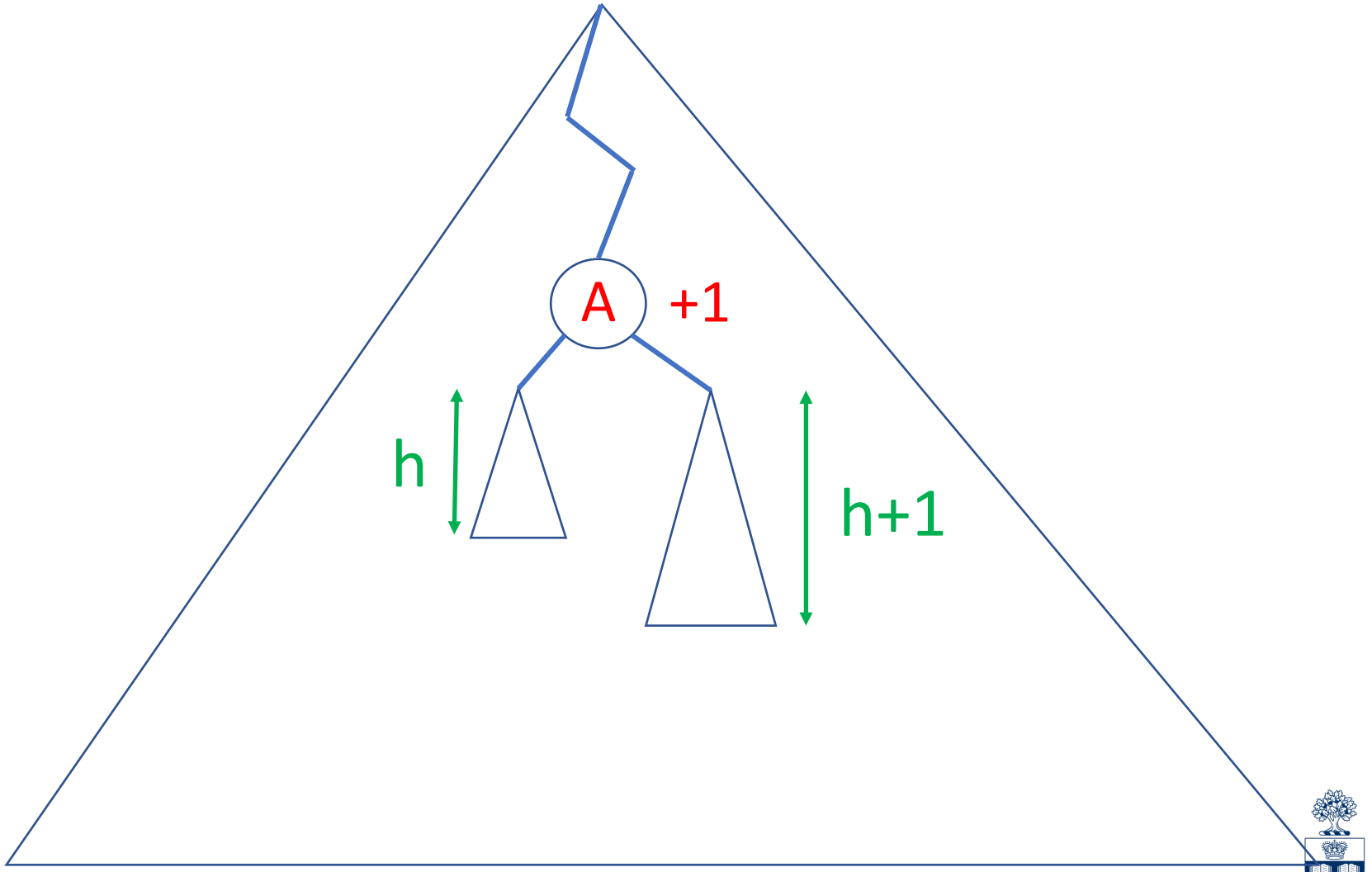
Insert(T, x) : General idea

- Insert x into T as in any BST :
  - x is now a leaf

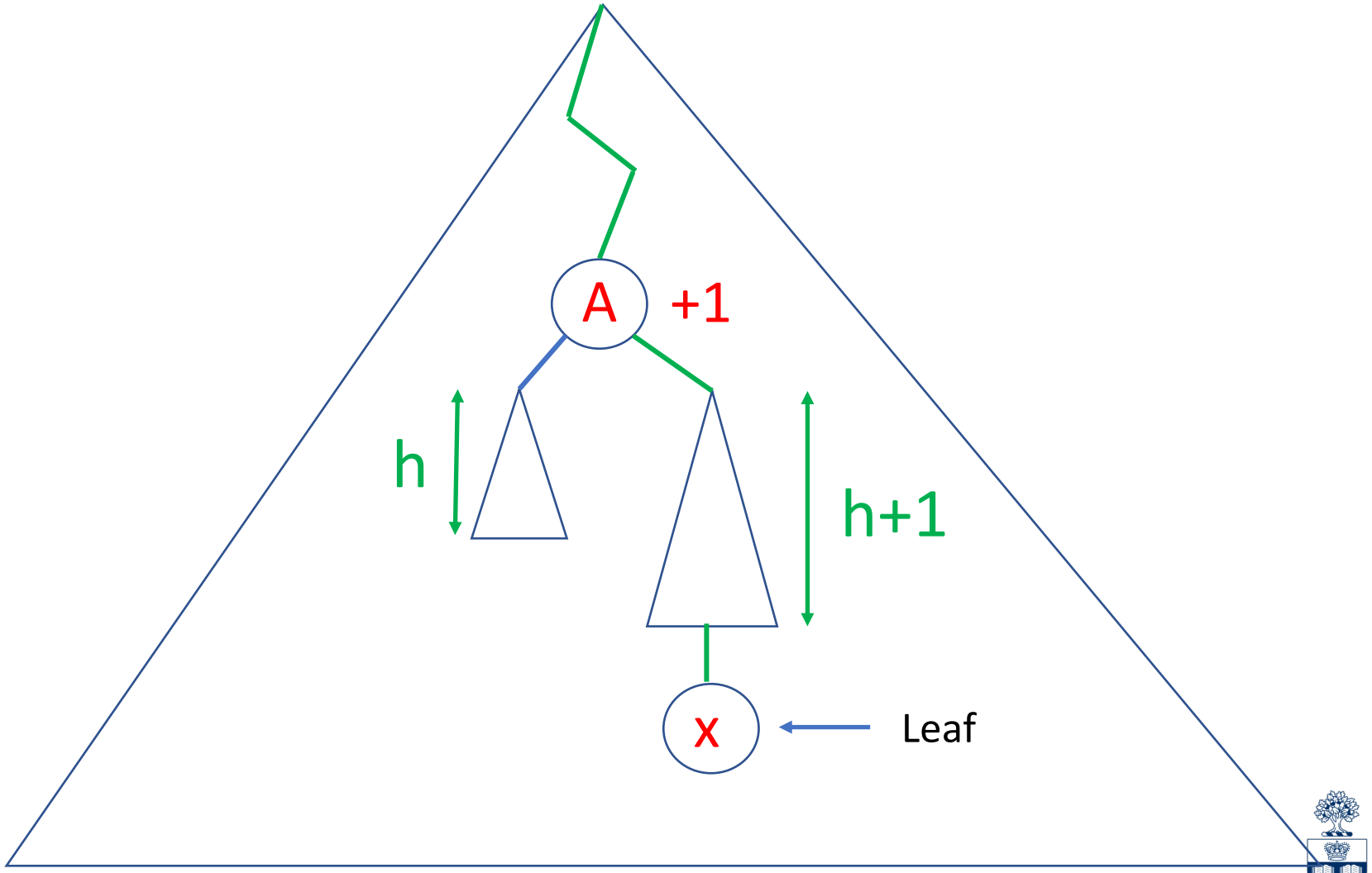


- For each node v from x to the root of T:
  - Adjust BF(v)
  - Rebalance if BF(v) > 1 or BF < -1

Inserting x into an AVL Tree T

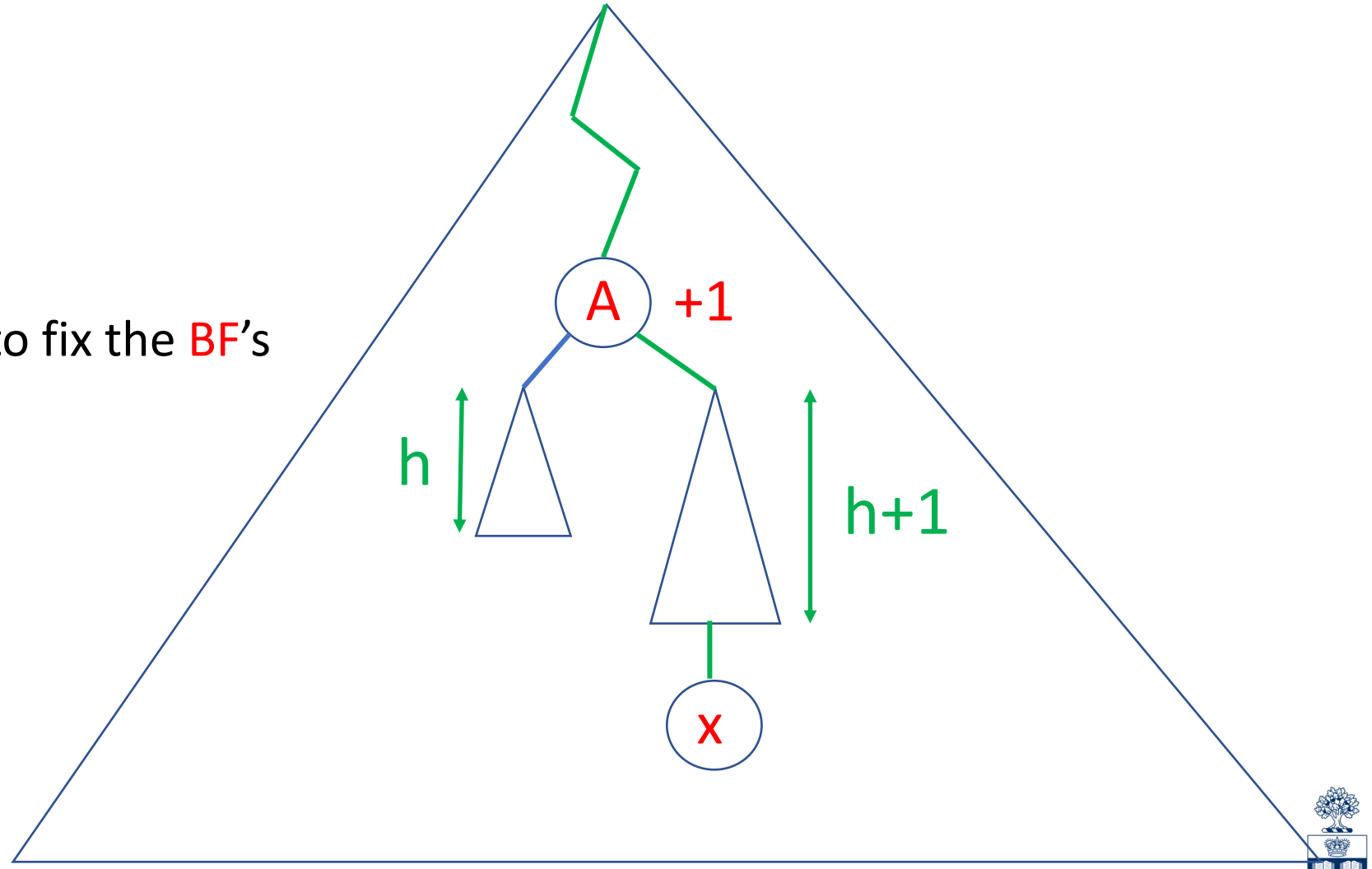


Inserting x into an AVL Tree T



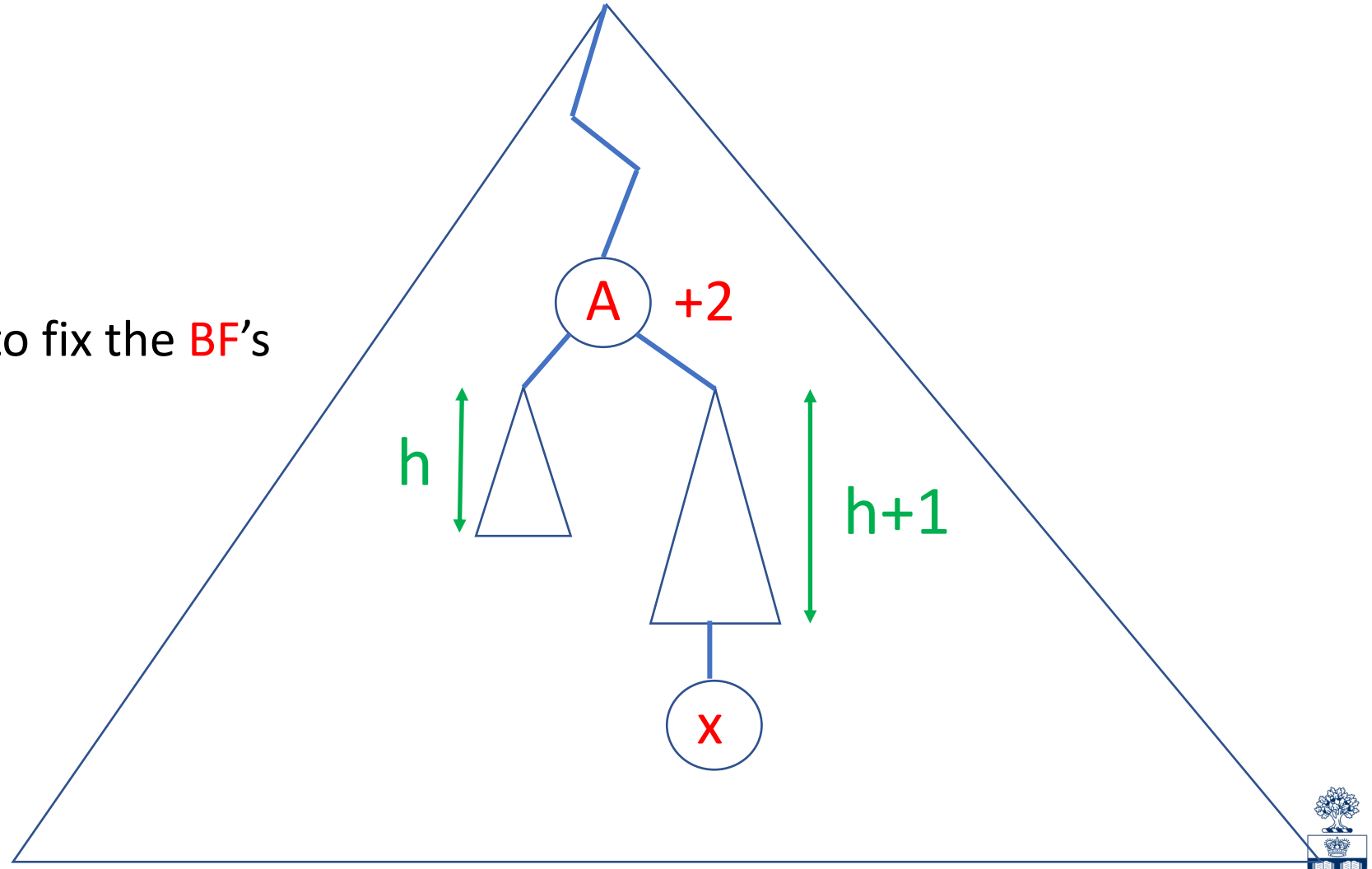
Inserting x into an AVL Tree T

Go up from x to the root of T to fix the BF's

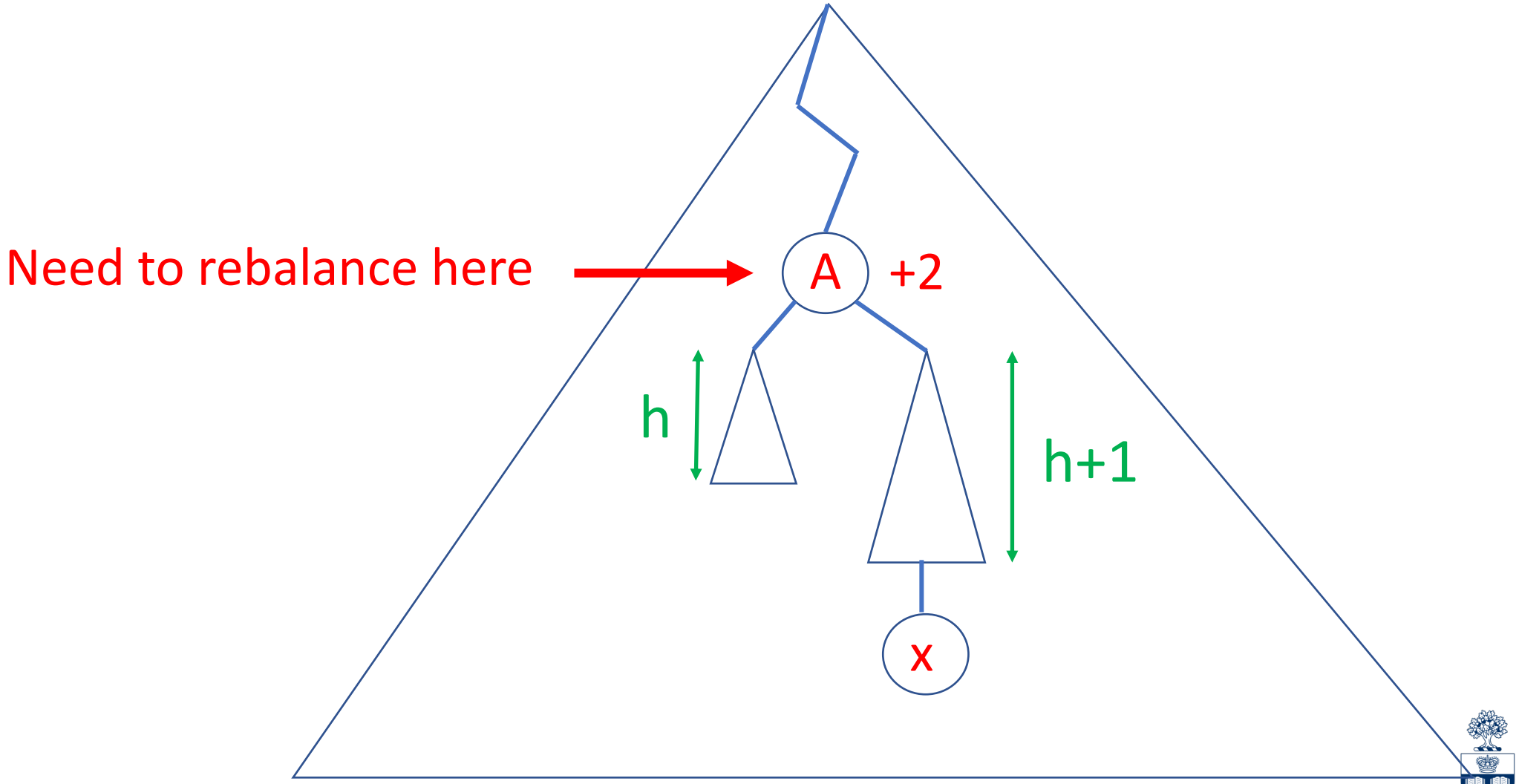


Inserting x into an AVL Tree T

Go up from x to the root of T to fix the BF's

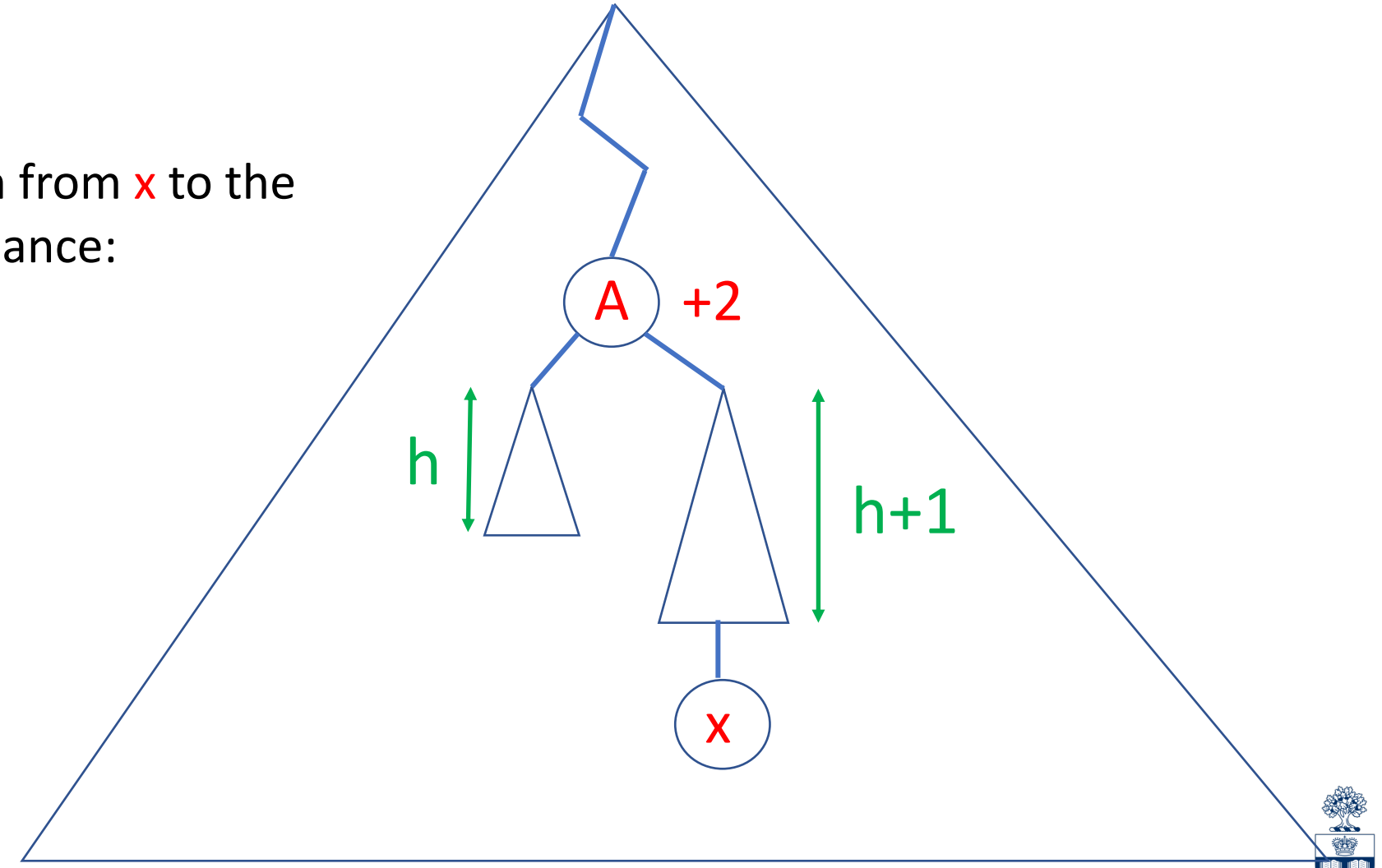


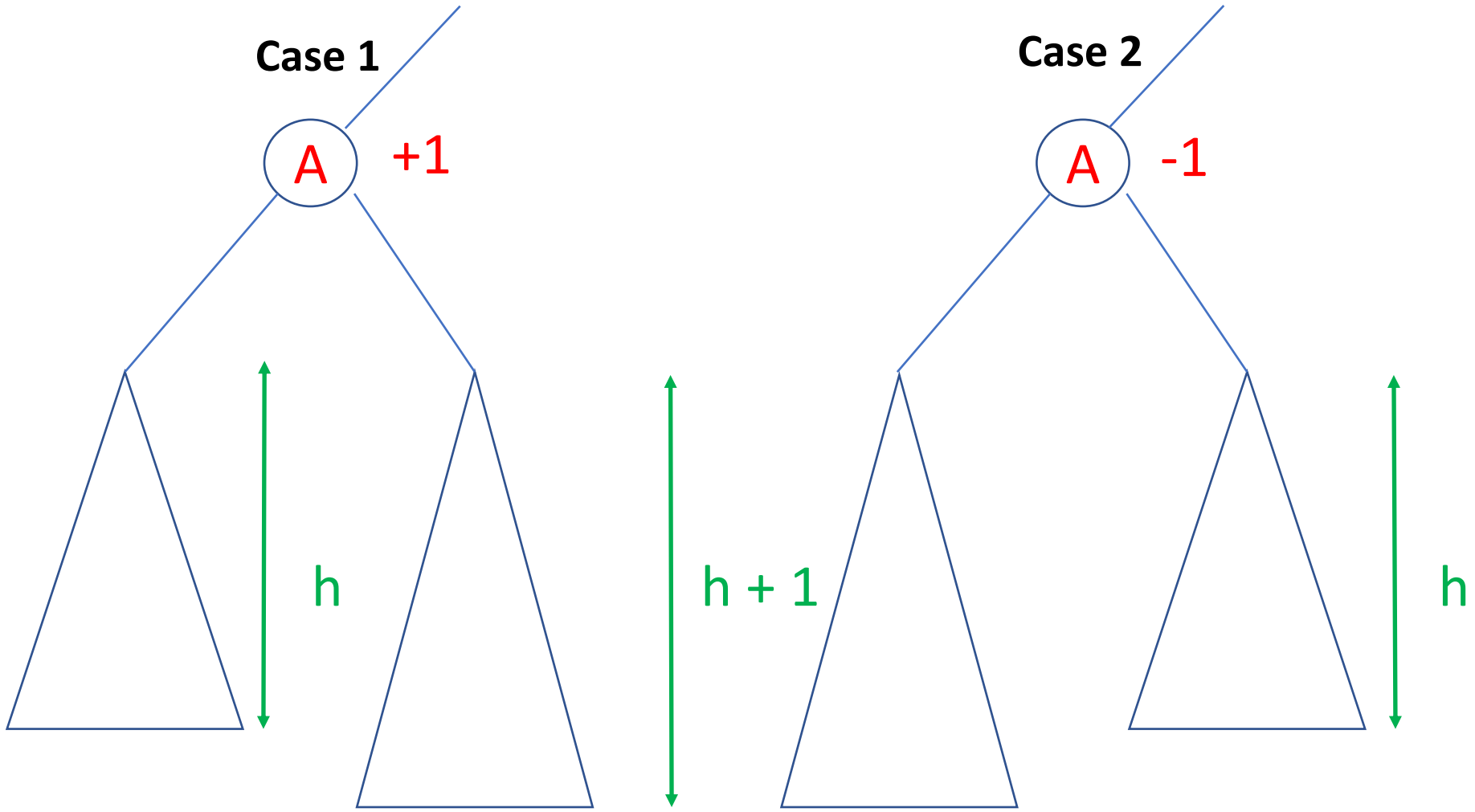
Inserting x into an AVL Tree T

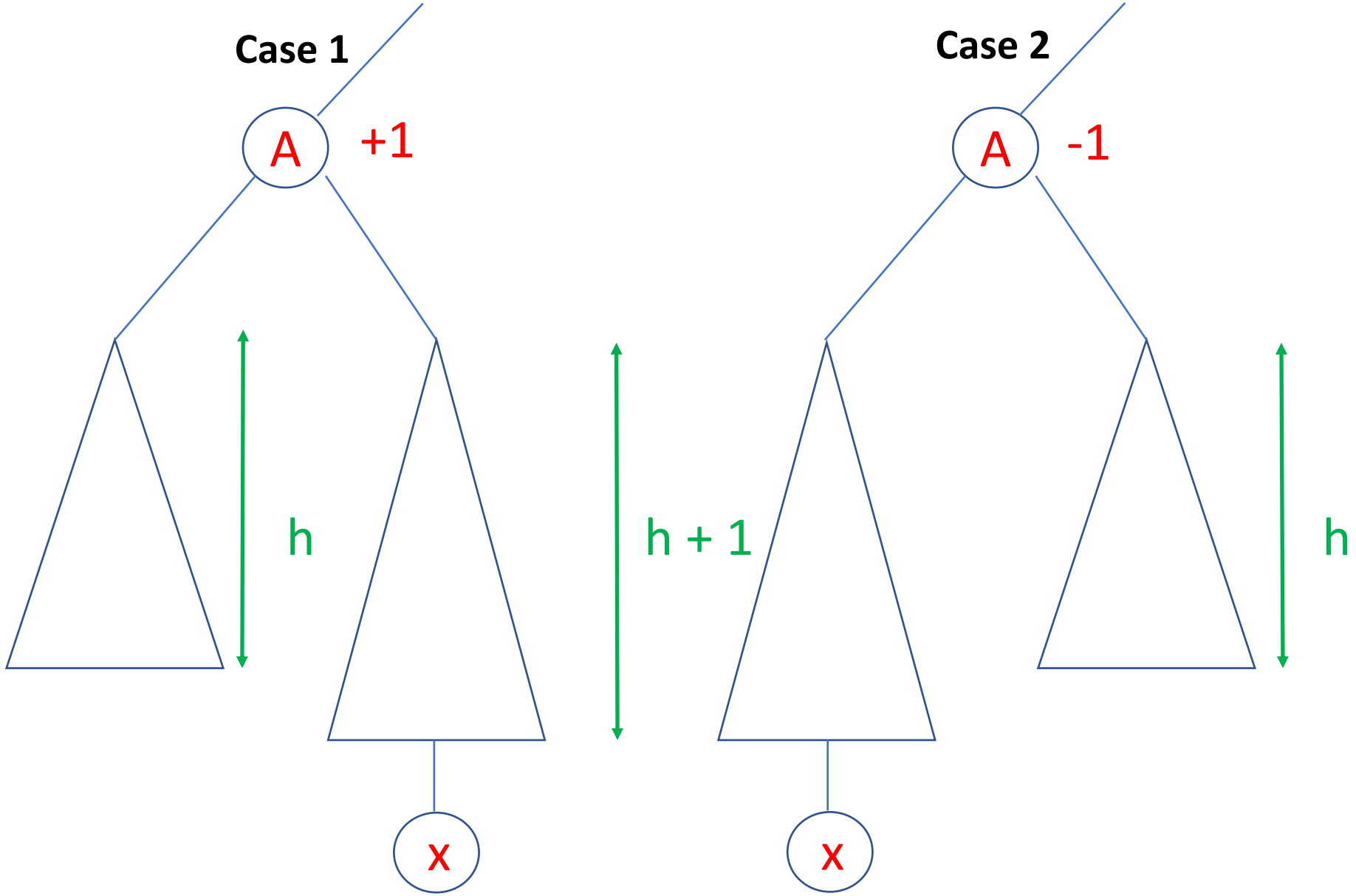


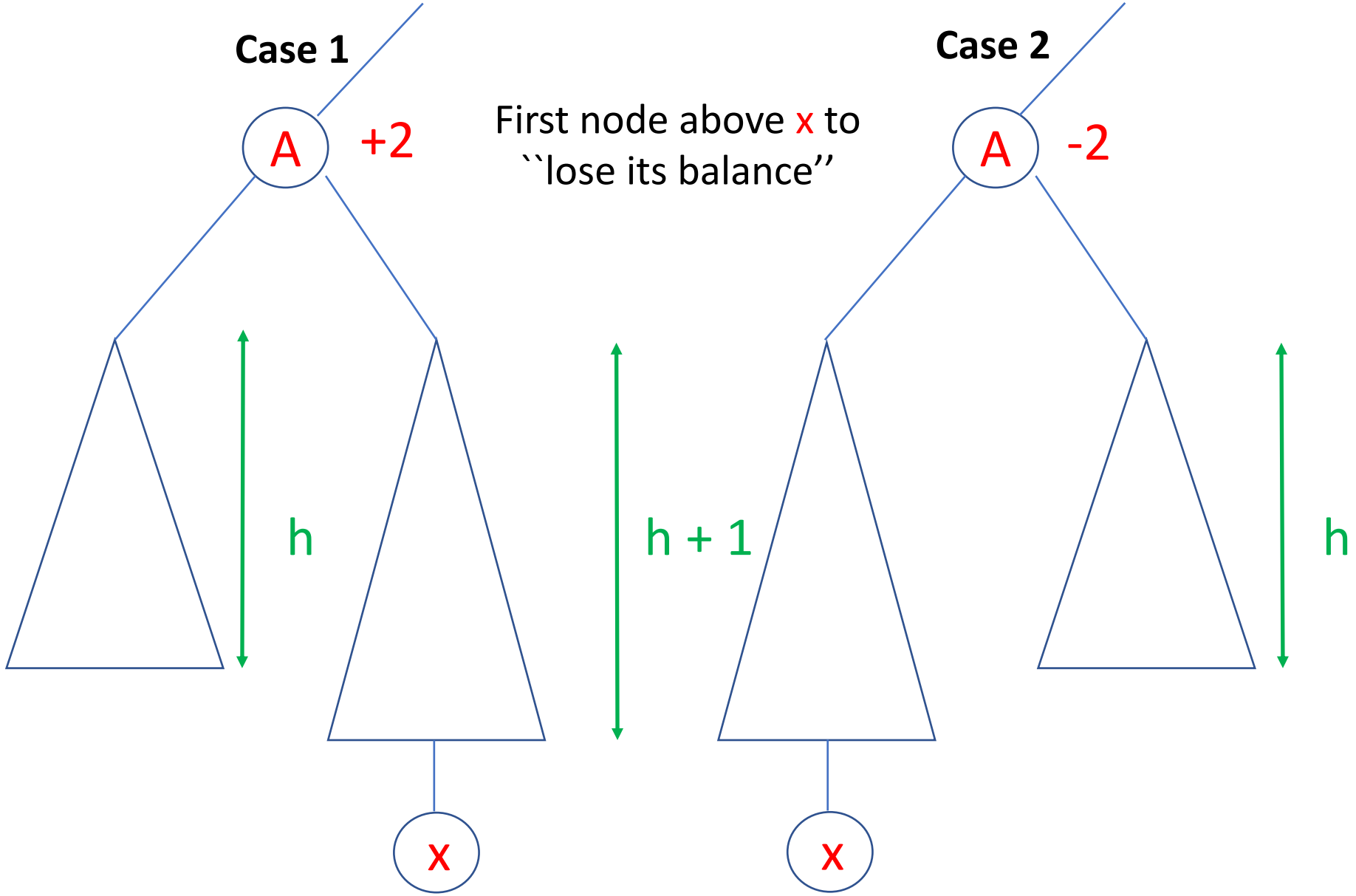
Inserting x into an AVL Tree T

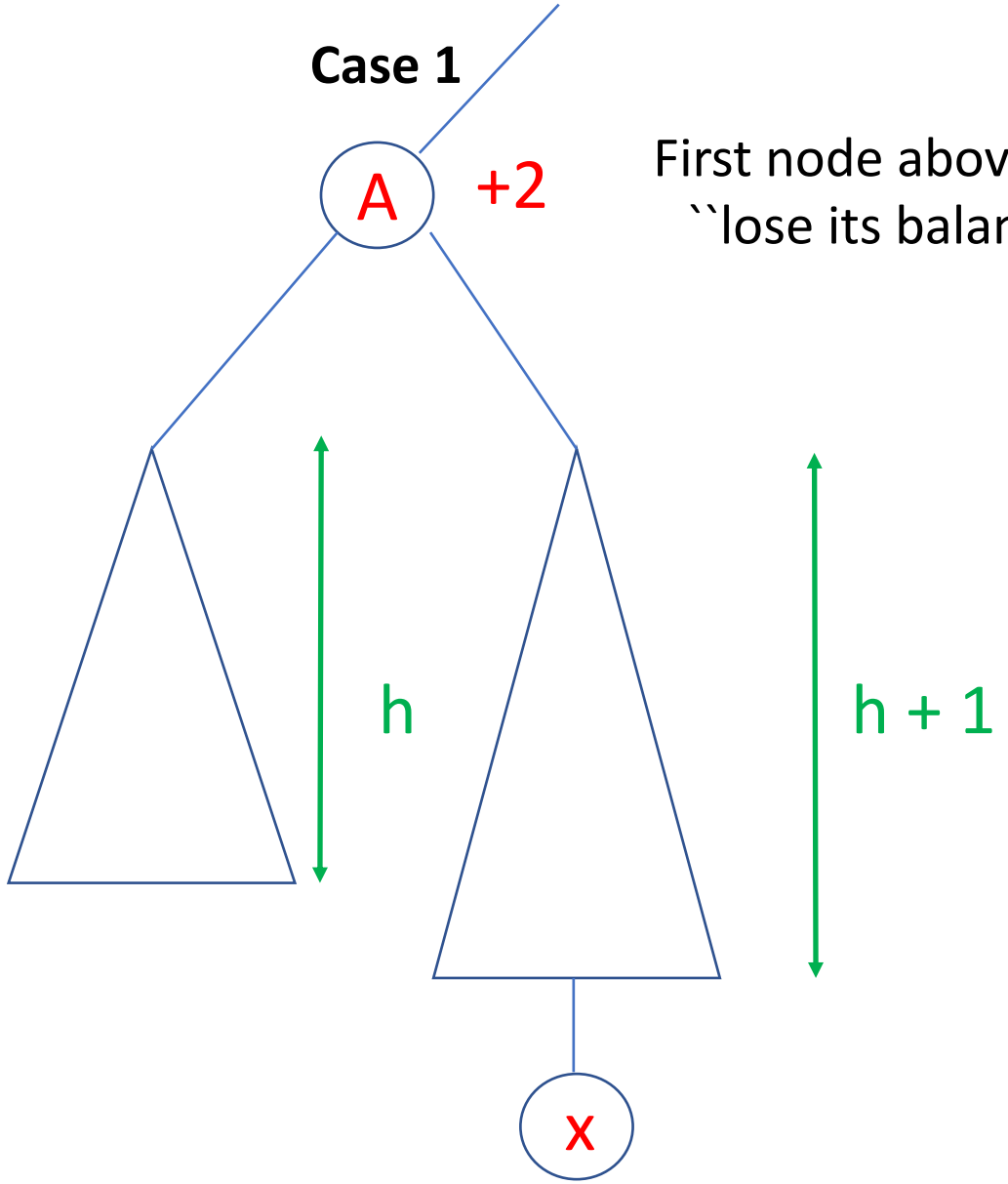
A is the **first** node on the path from x to the root of T that loses its AVL balance:
BF(A) becomes +2 or -2.

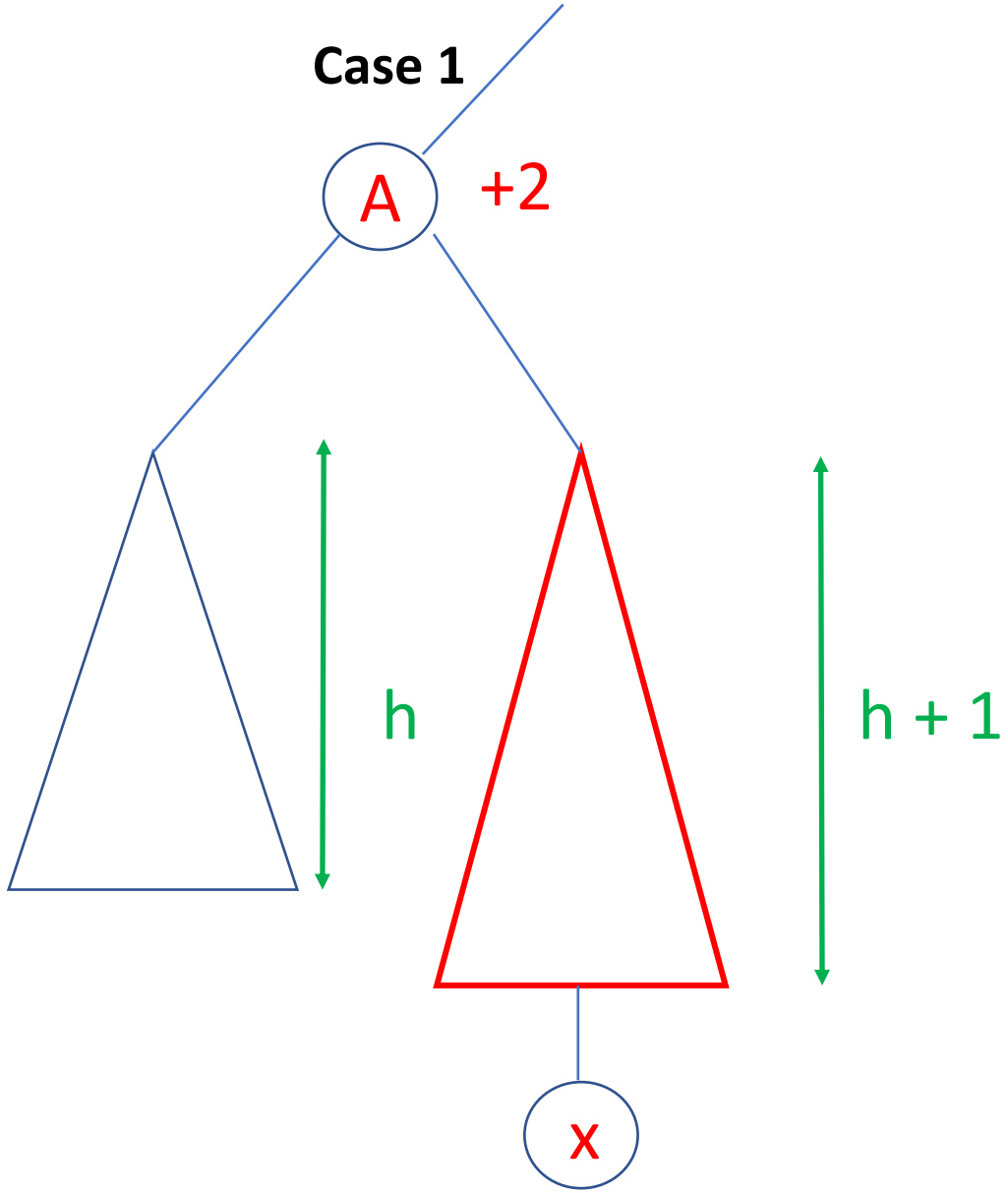




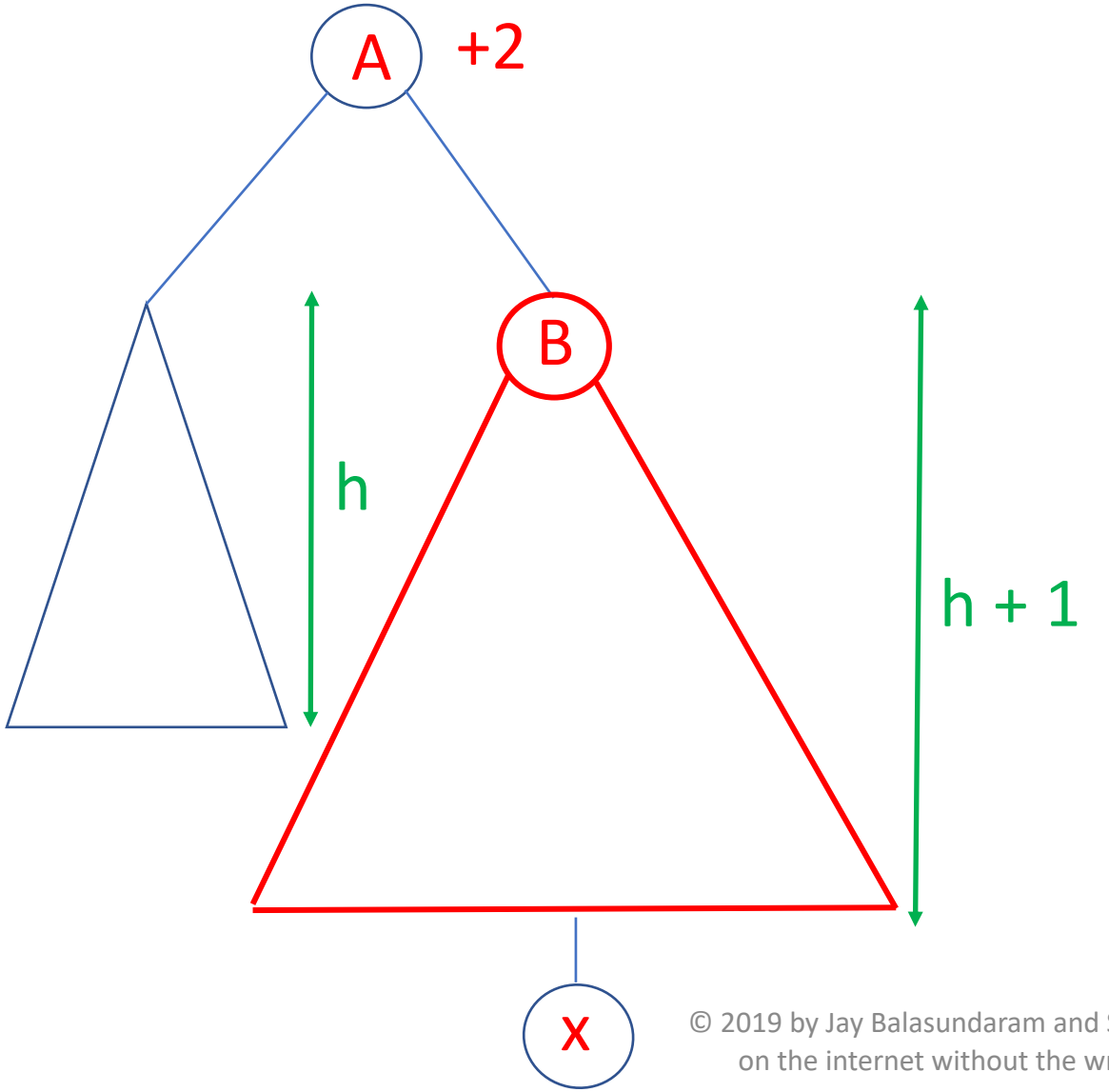




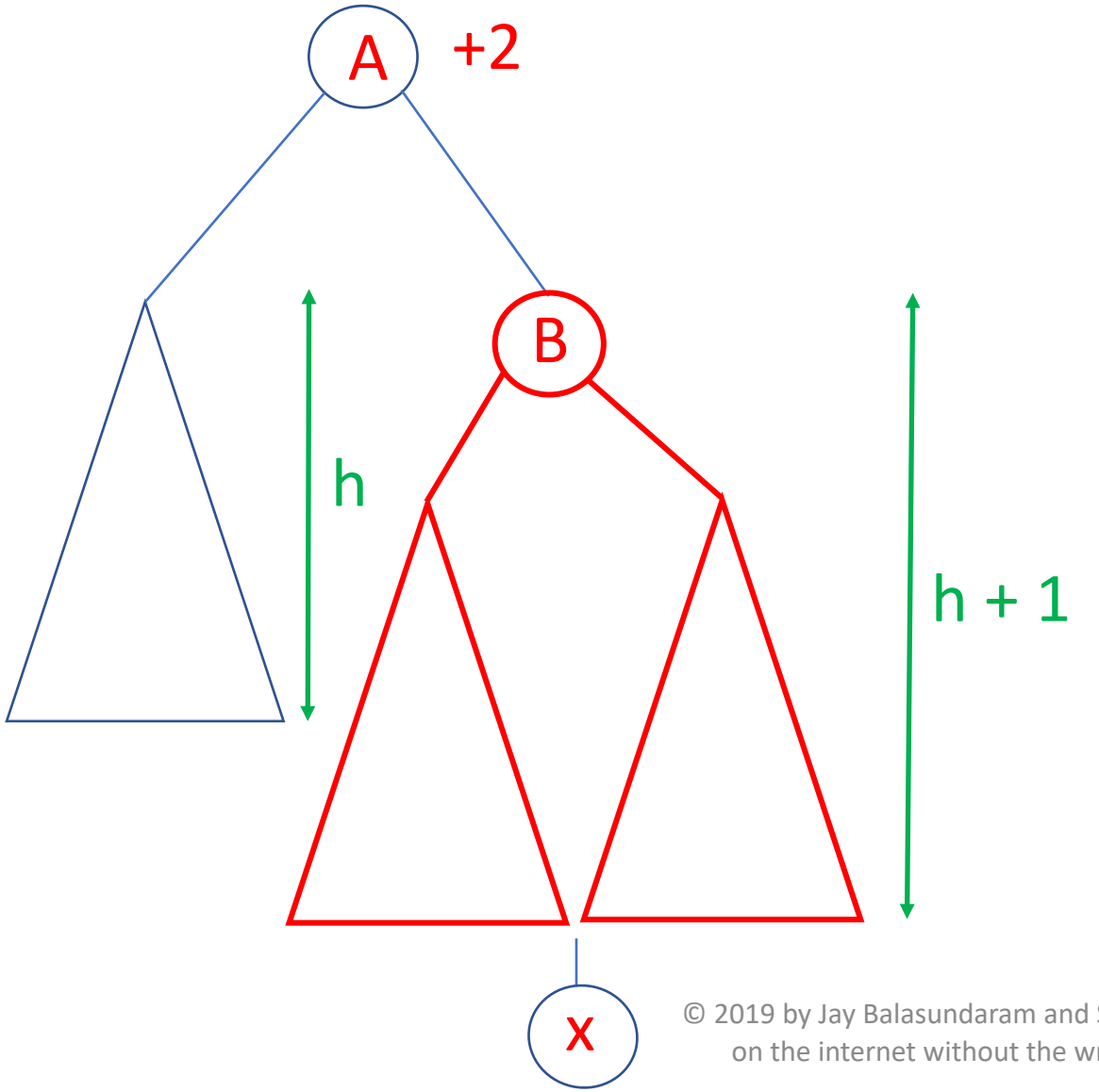




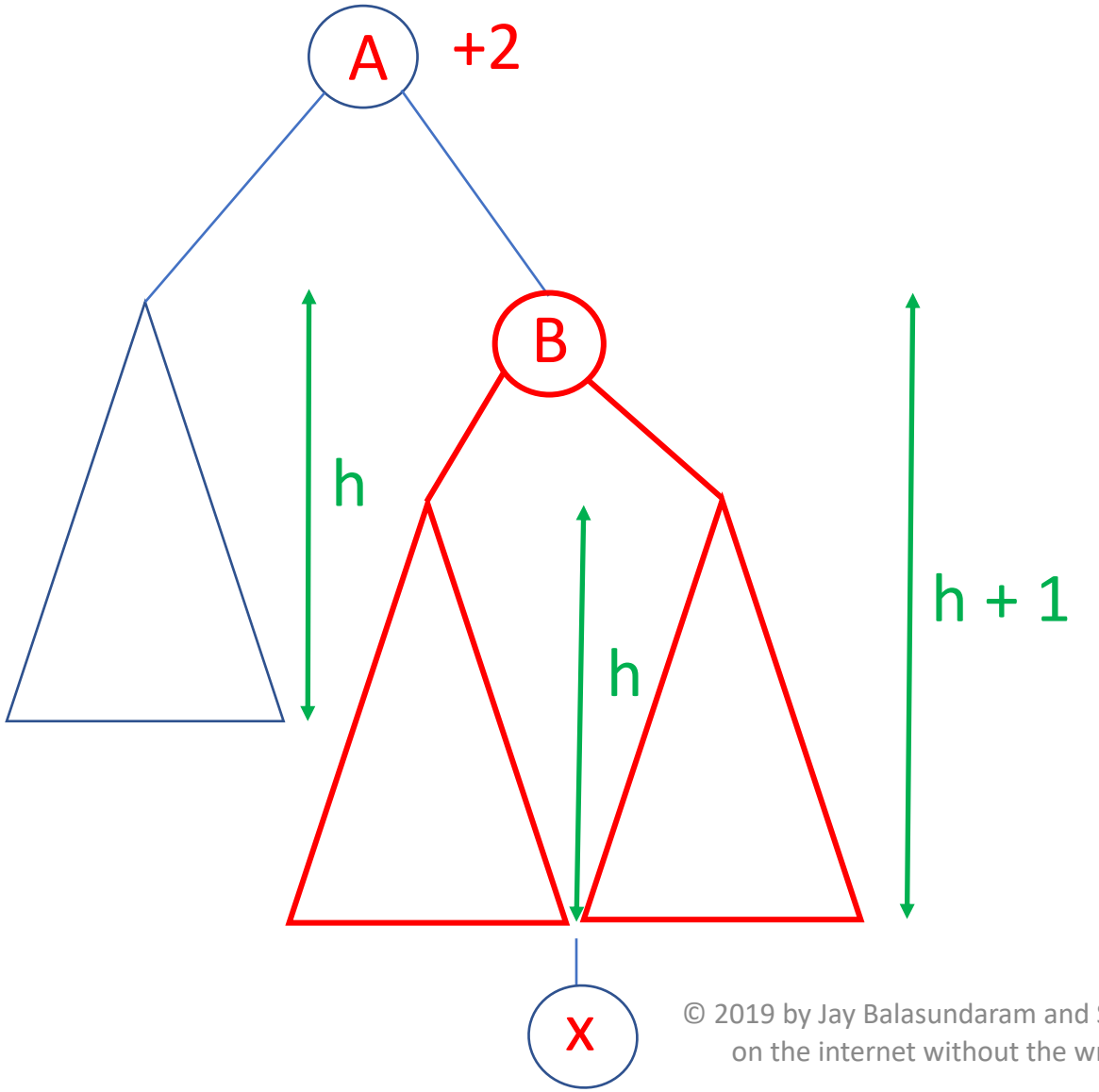
Case 1



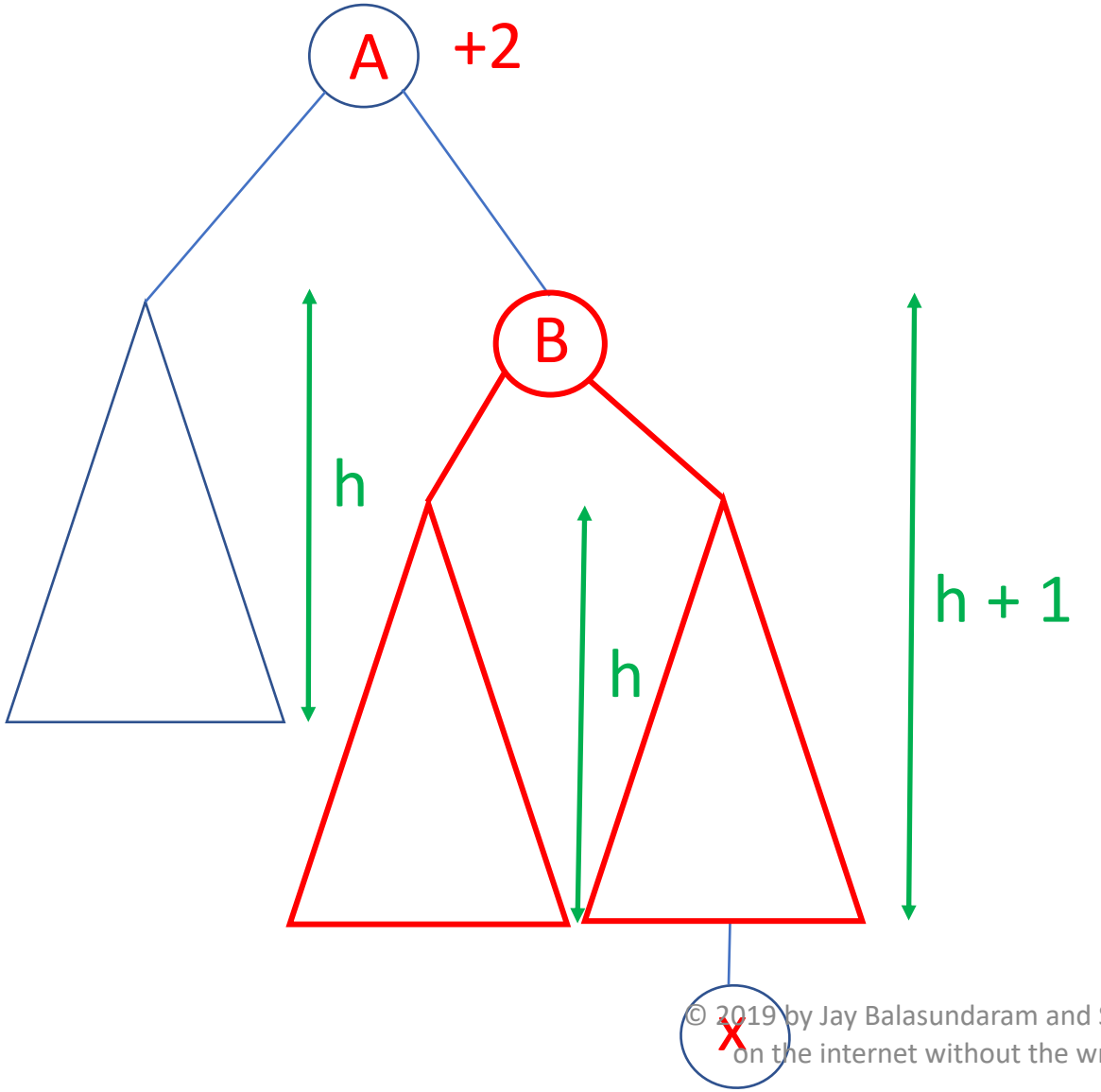
Case 1



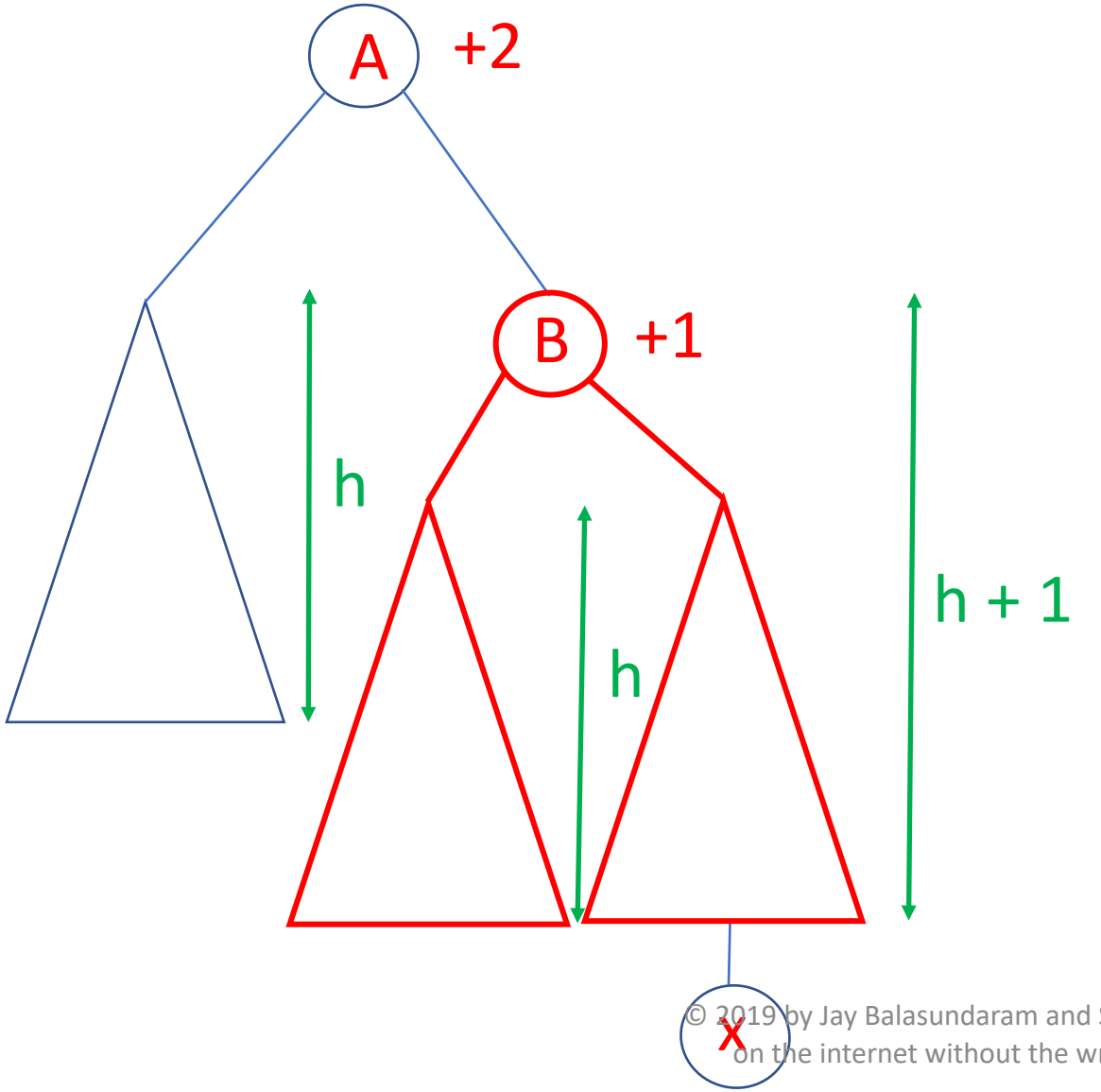
Case 1



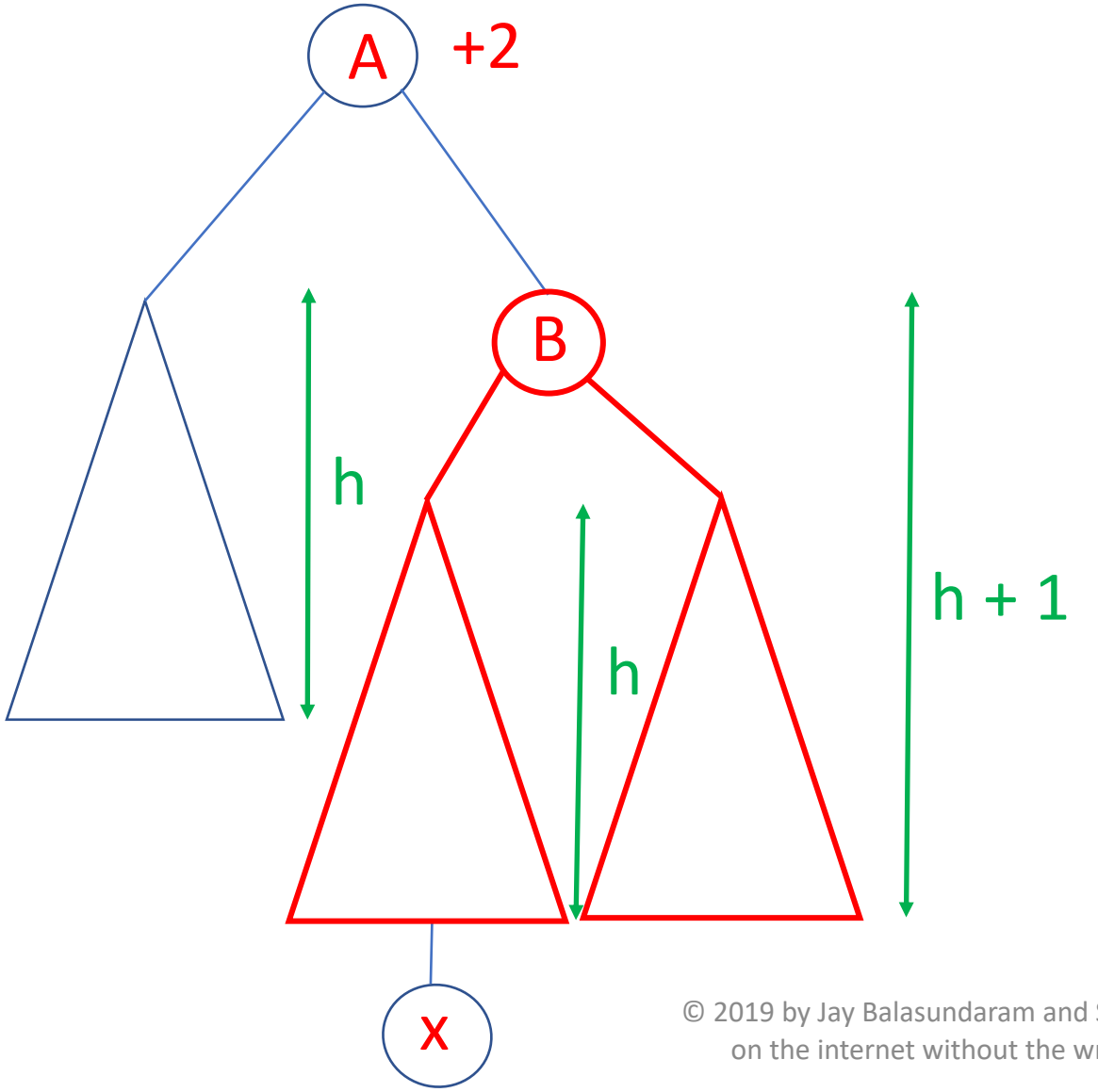
Case 1 (a)



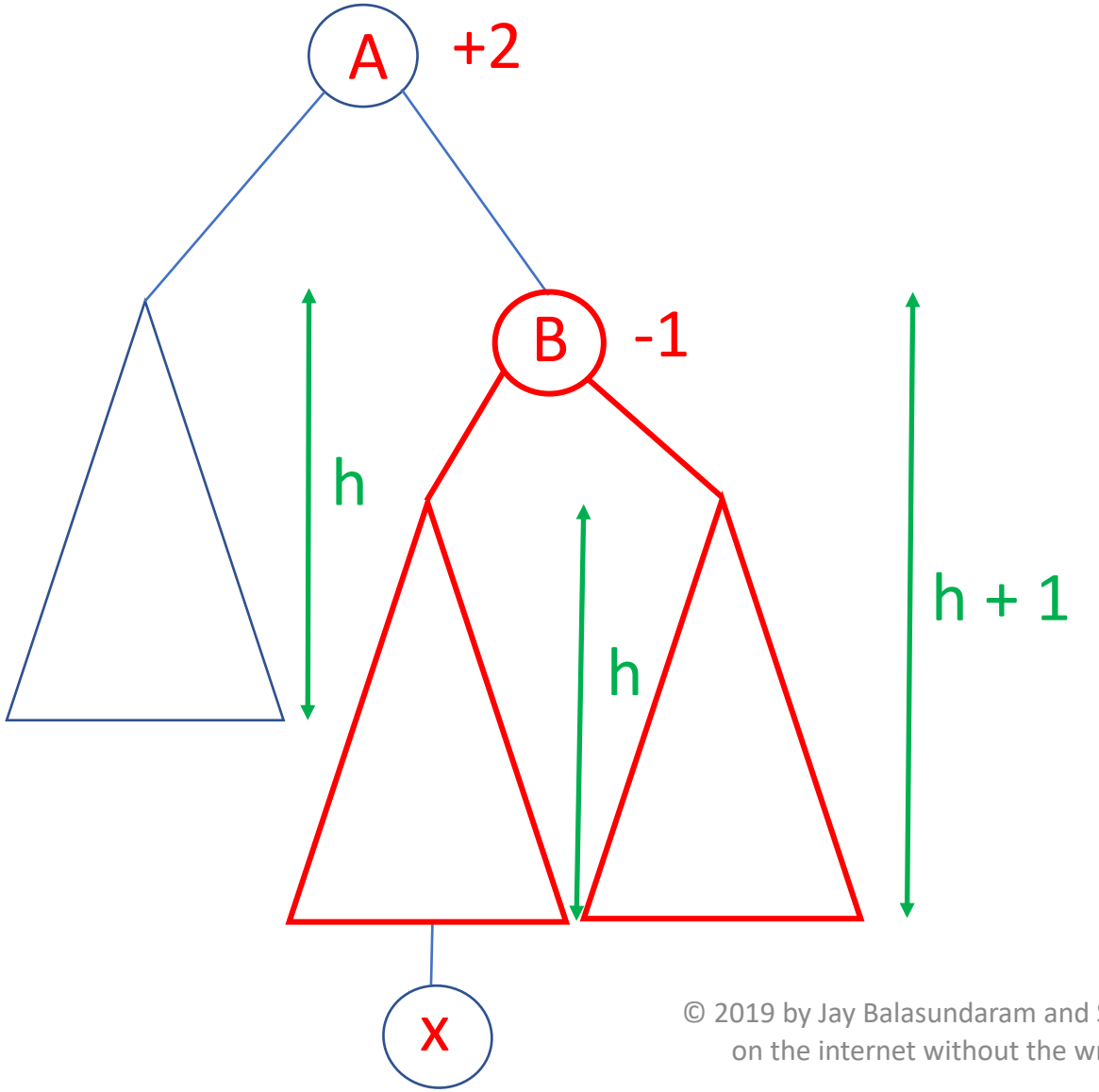
Case 1 (a)



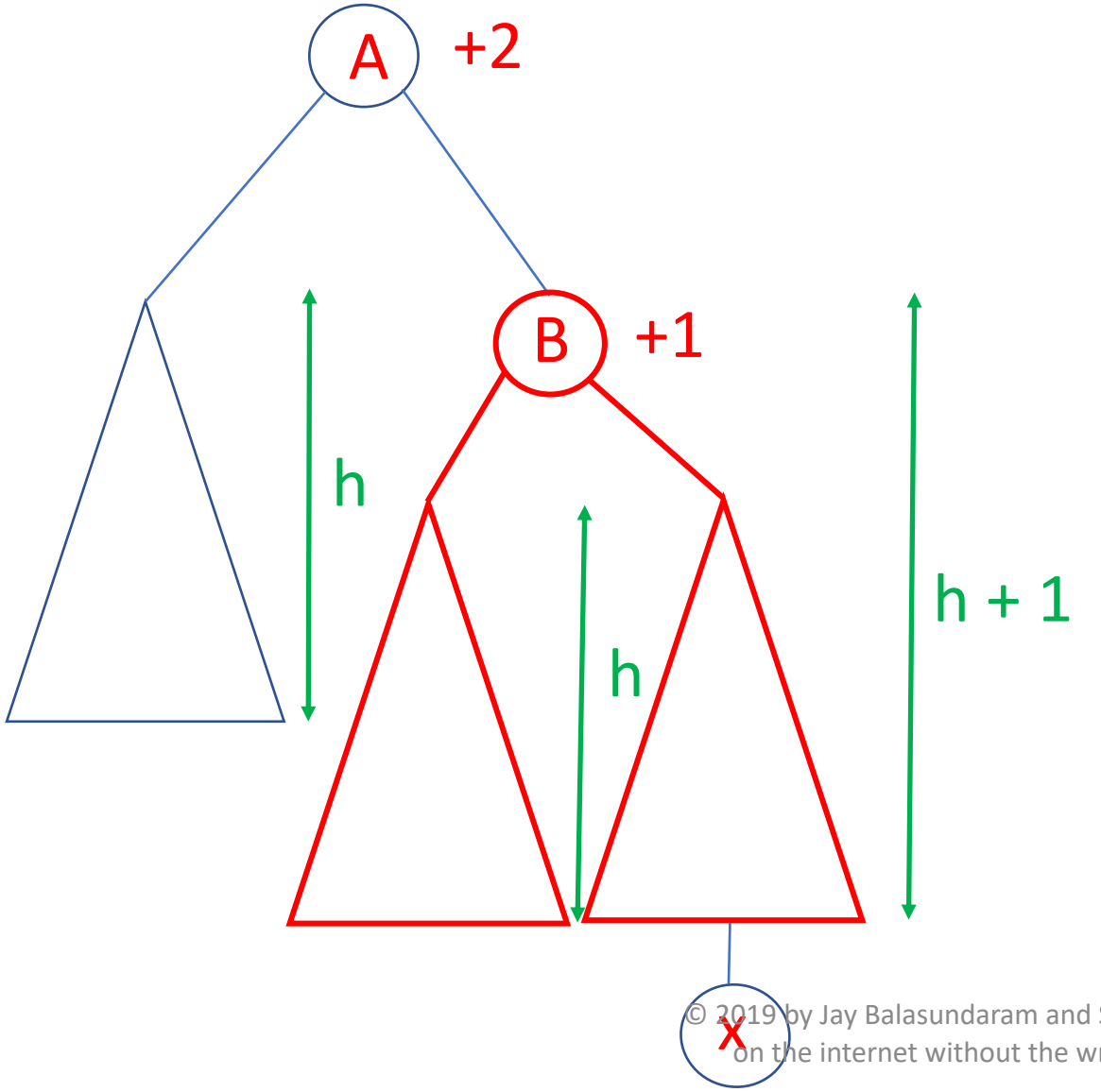
Case 1 (b)



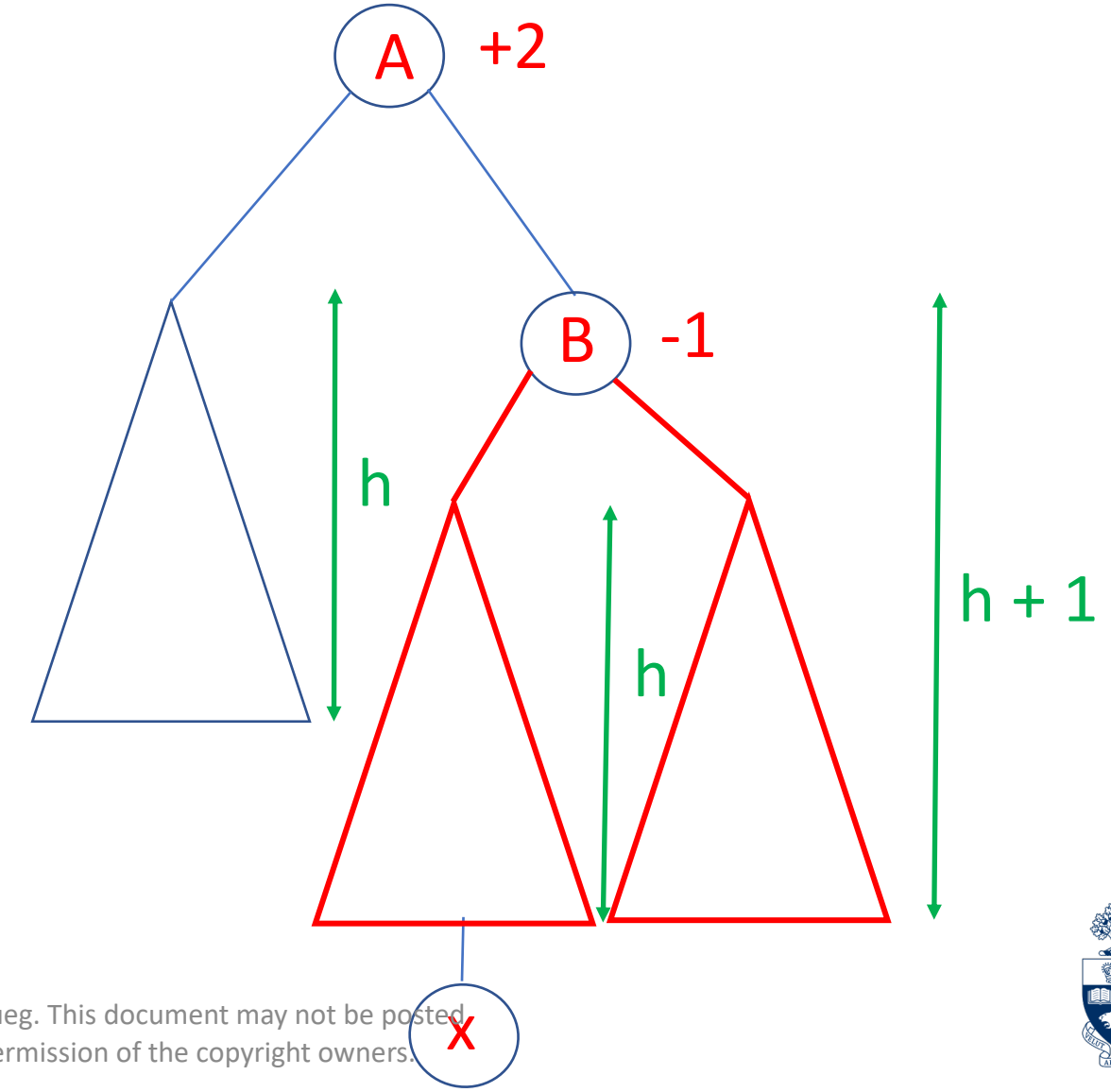
Case 1 (b)



Case 1 (a)



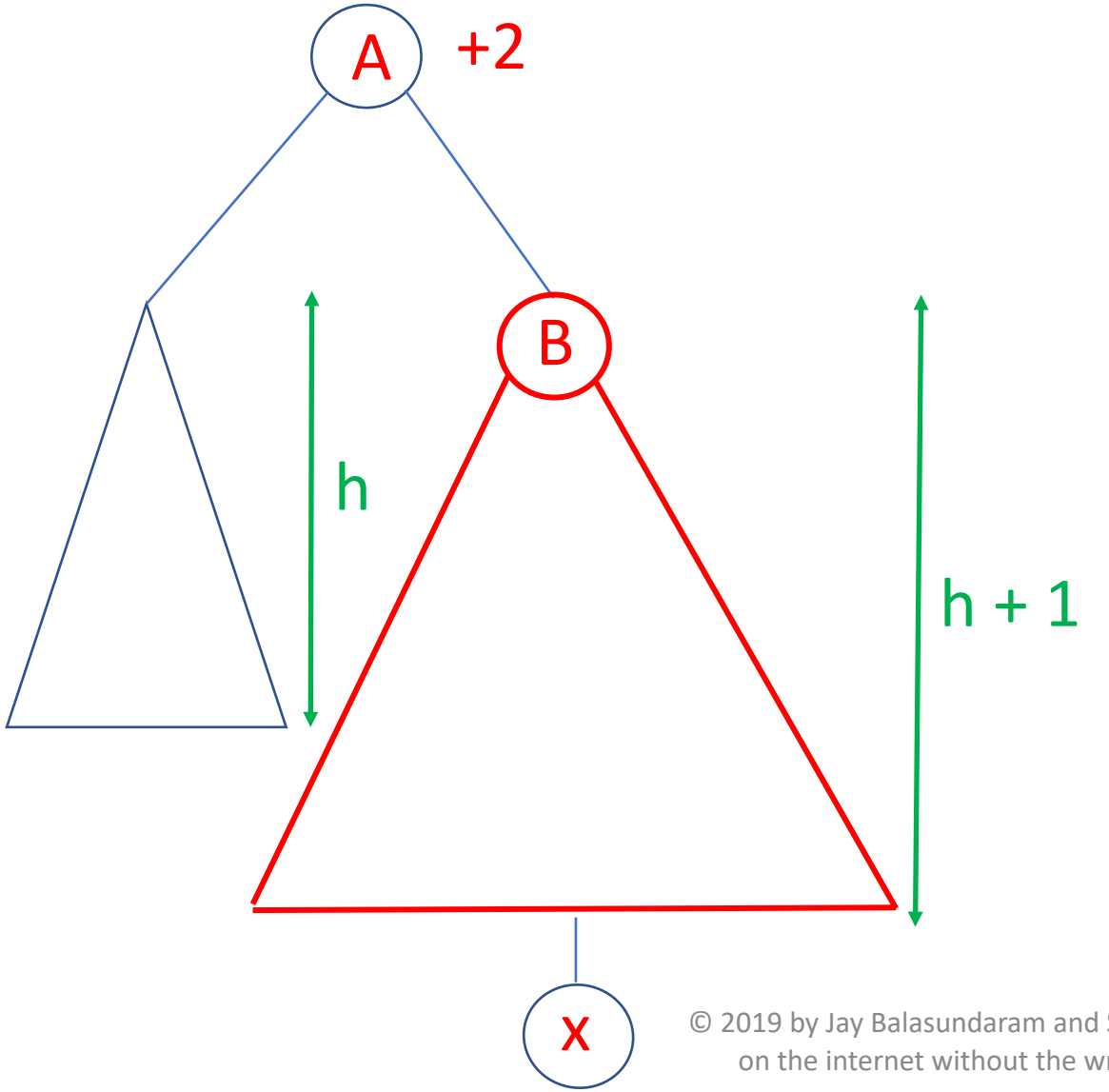
Case 1 (b)



We (sneakily :-) skipped a step in the argument!

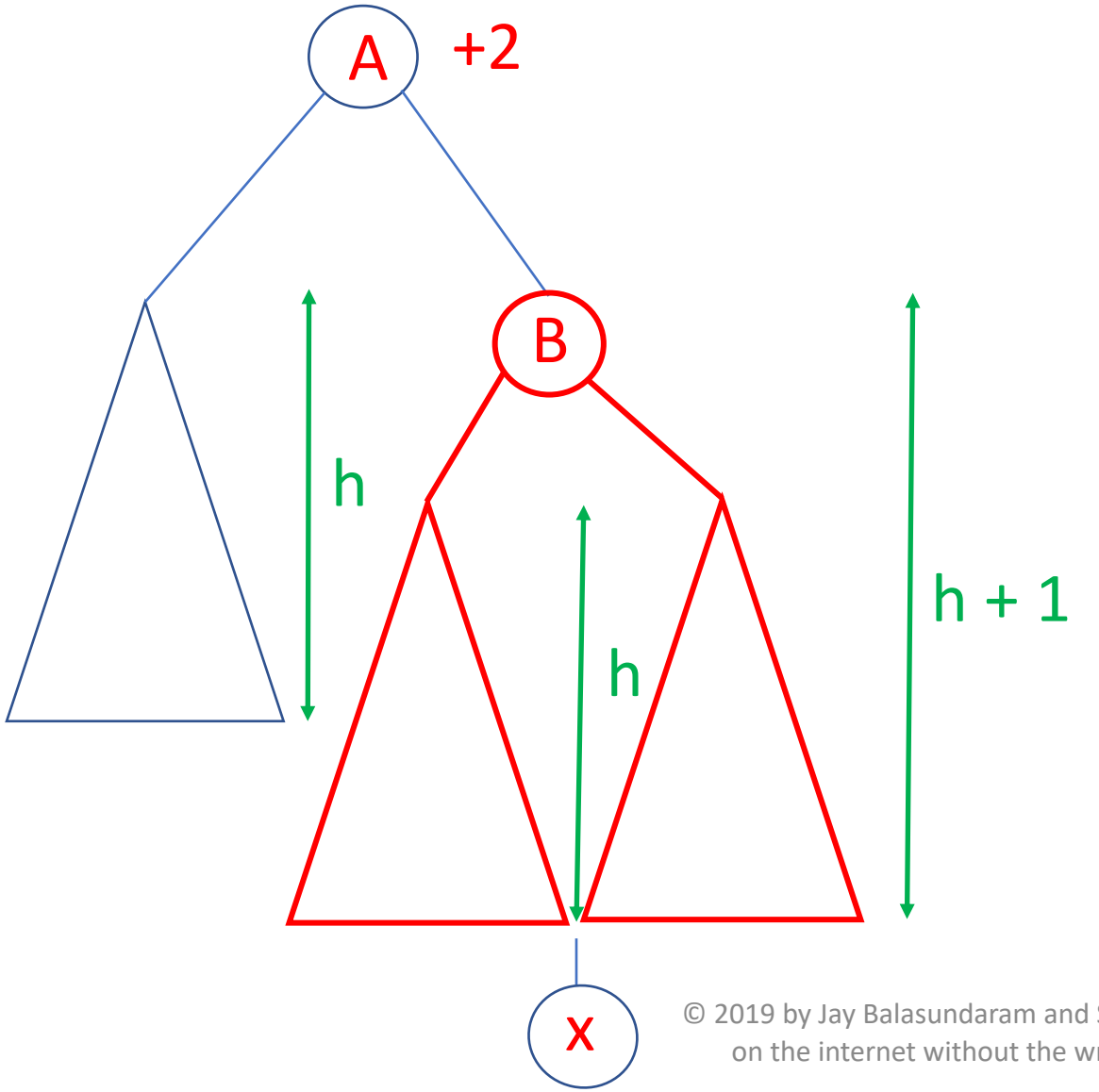
We (sneakily :-) skipped a step in the argument!

Case 1



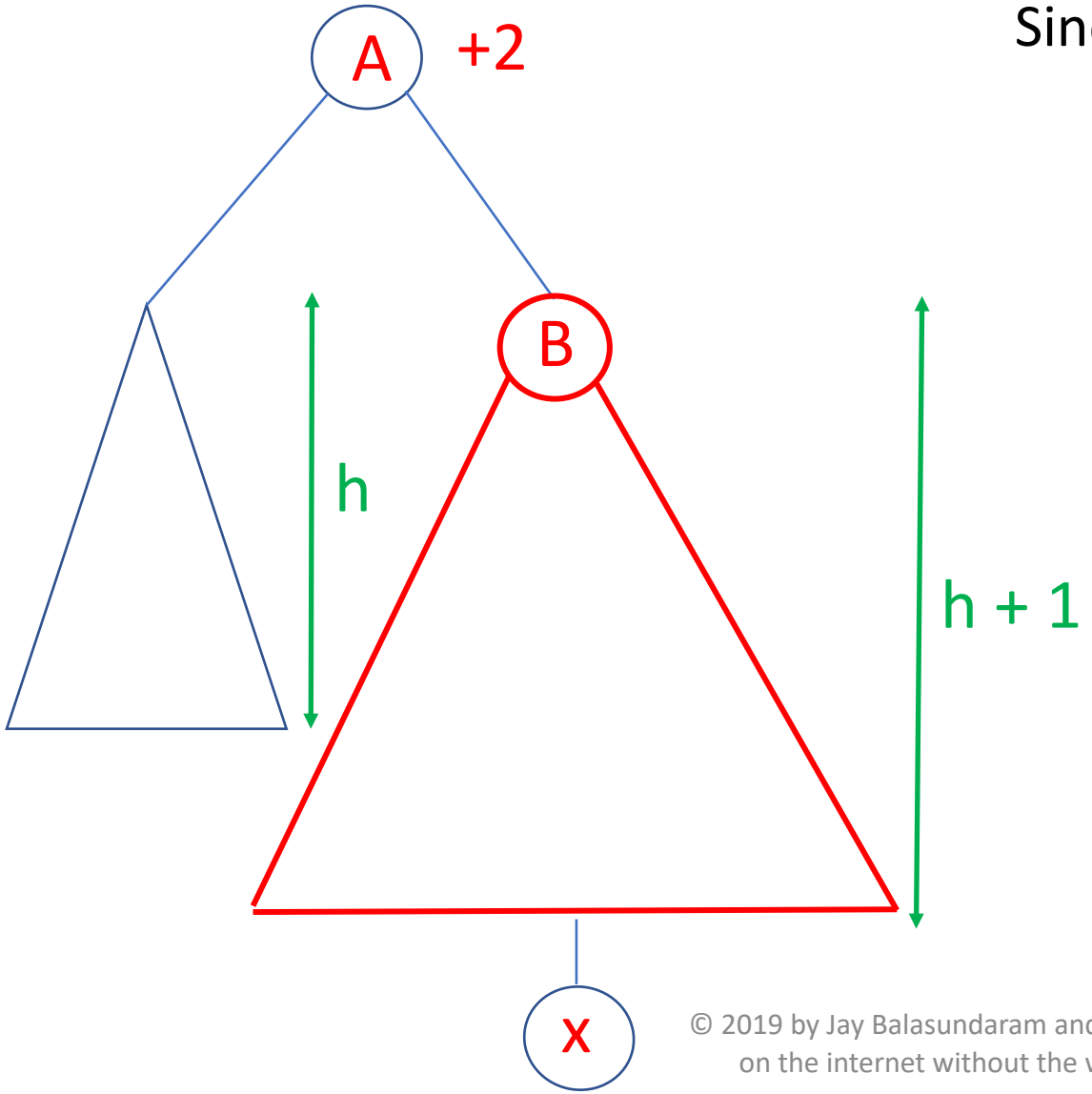
We (sneakily :-) skipped a step in the argument!

Case 1



We (sneakily :-) skipped a step in the argument!

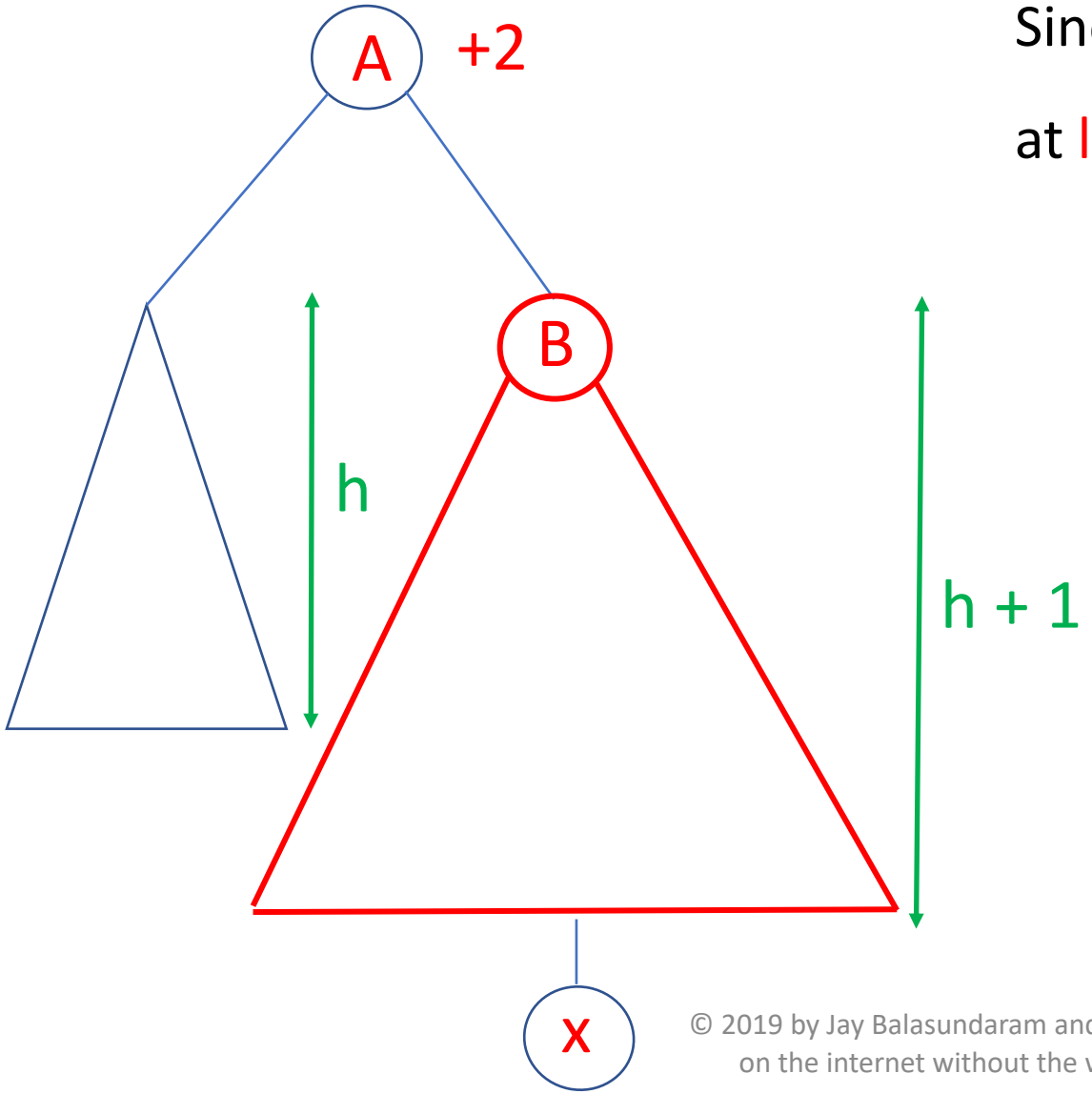
Case 1



Since B has height h+1:

We (sneakily :-) skipped a step in the argument!

Case 1

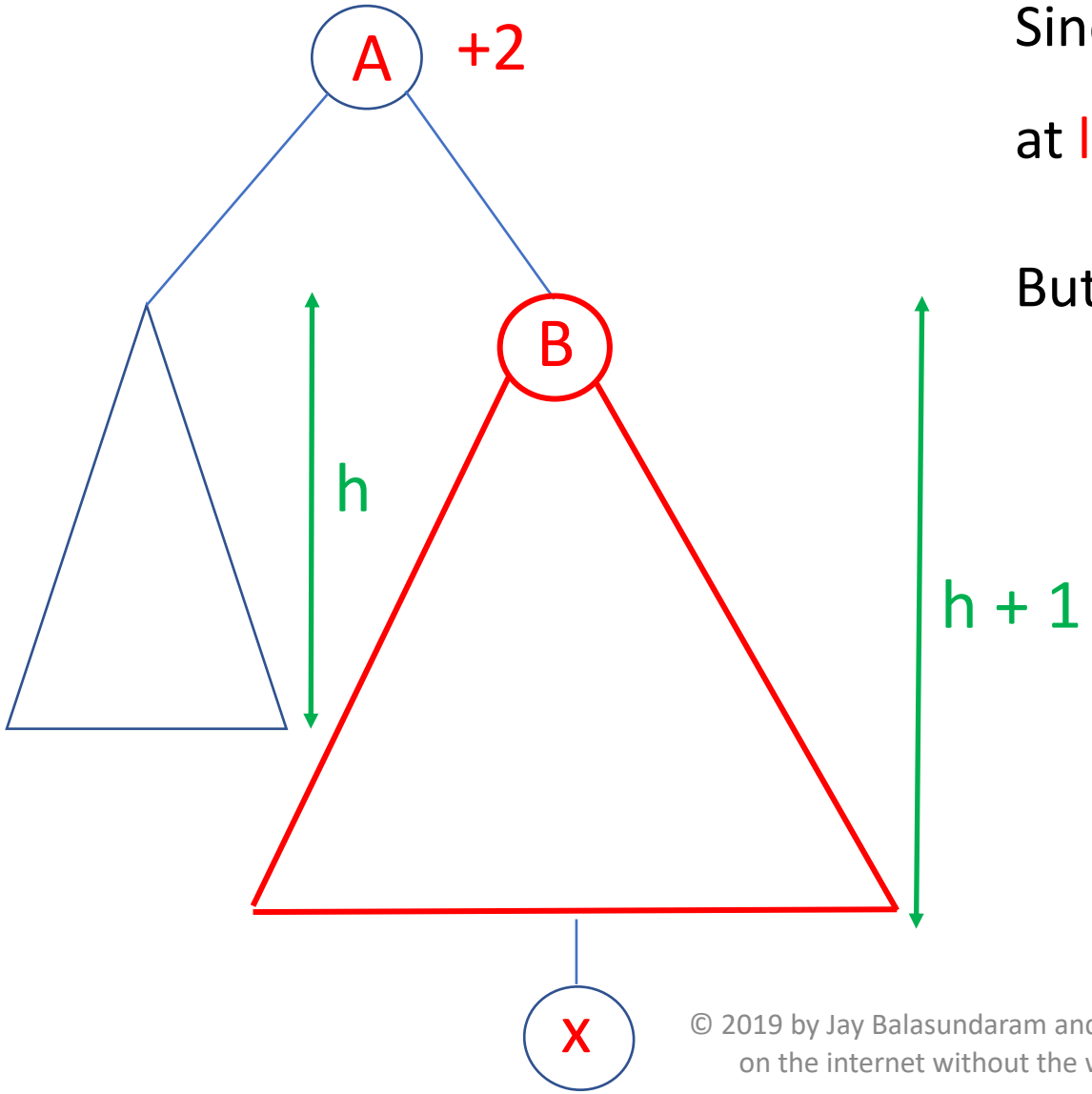


Since B has height h+1:

at least one subtrees of B has height h.

We (sneakily :-) skipped a step in the argument!

Case 1

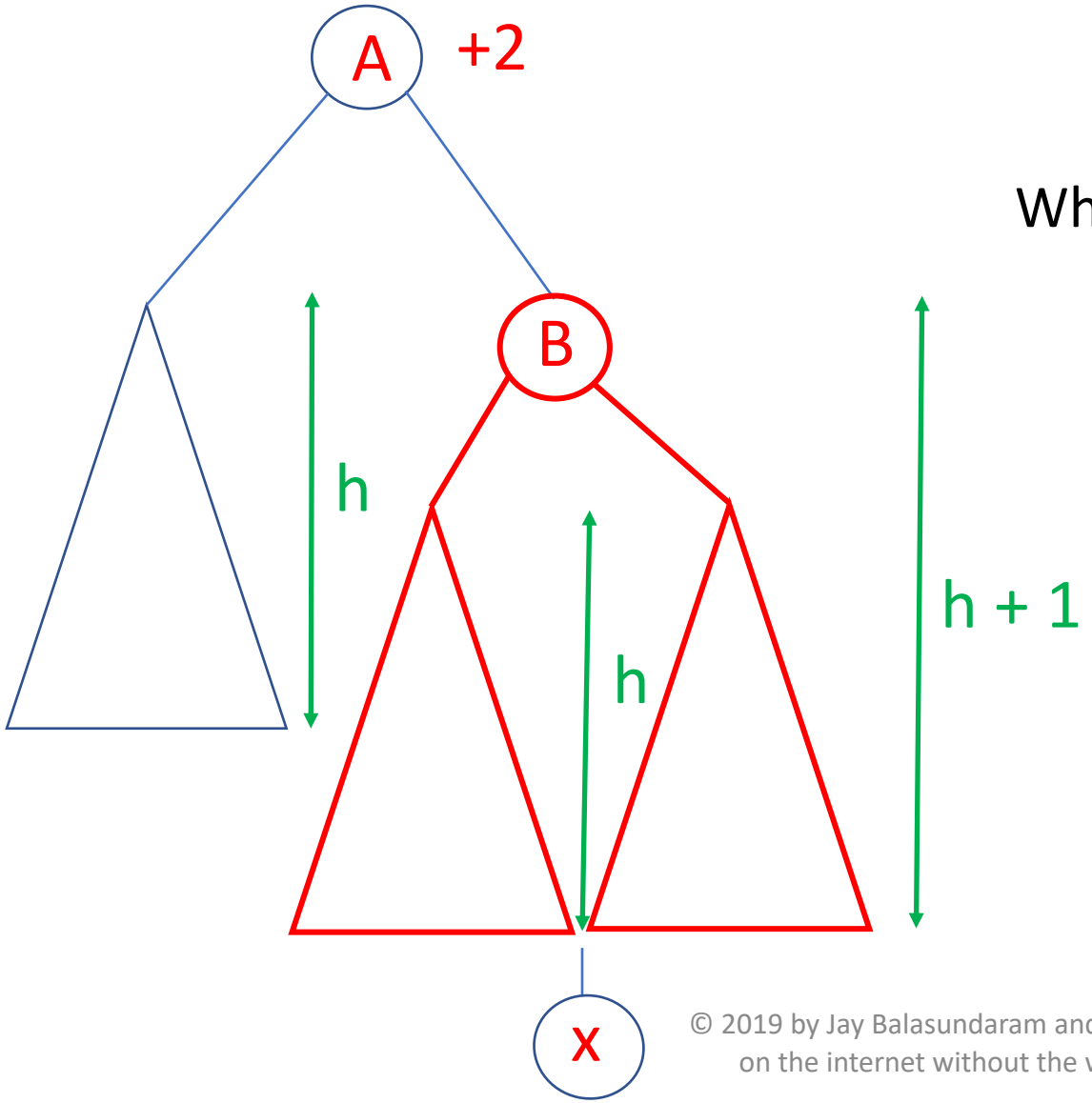


Since B has height h+1:

at least one subtrees of B has height h.

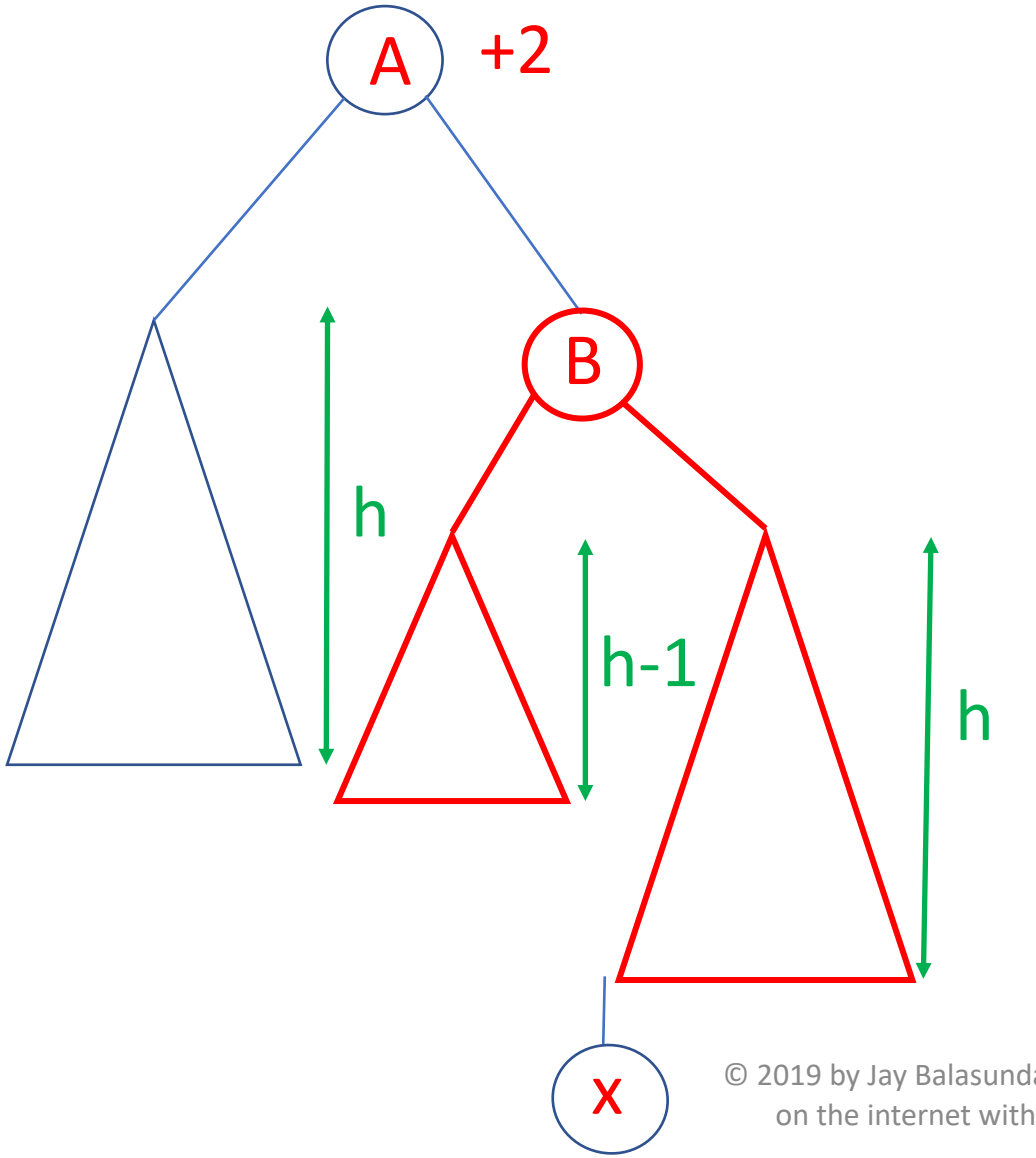
But why both subtrees of B have the same height h ?

Case 1



Why both subtrees of B have the same height h?

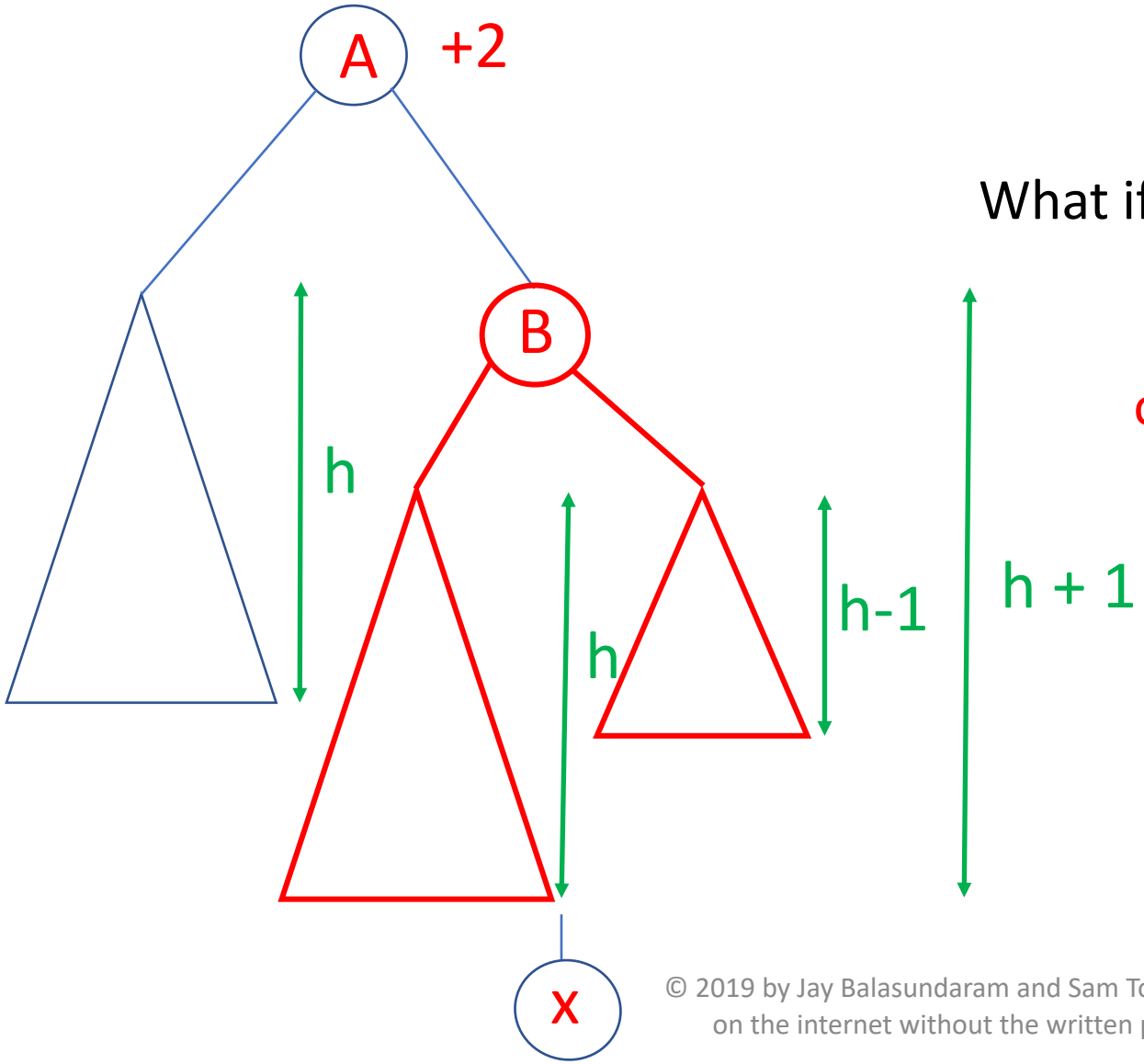
Case 1



What if only one of them has height h ?

like this

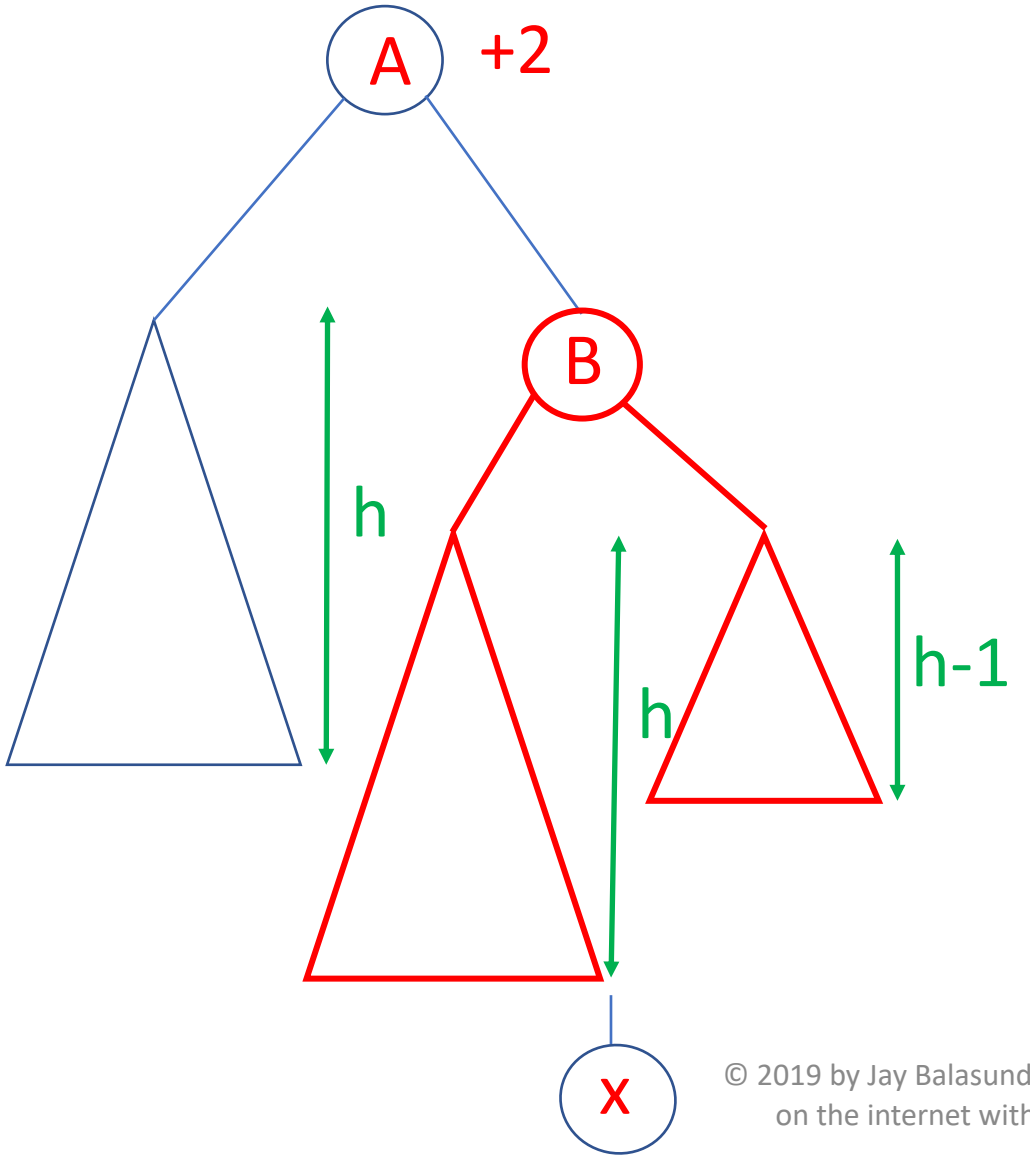
Case 1



What if only one of them has height h ?

or this

Case 1



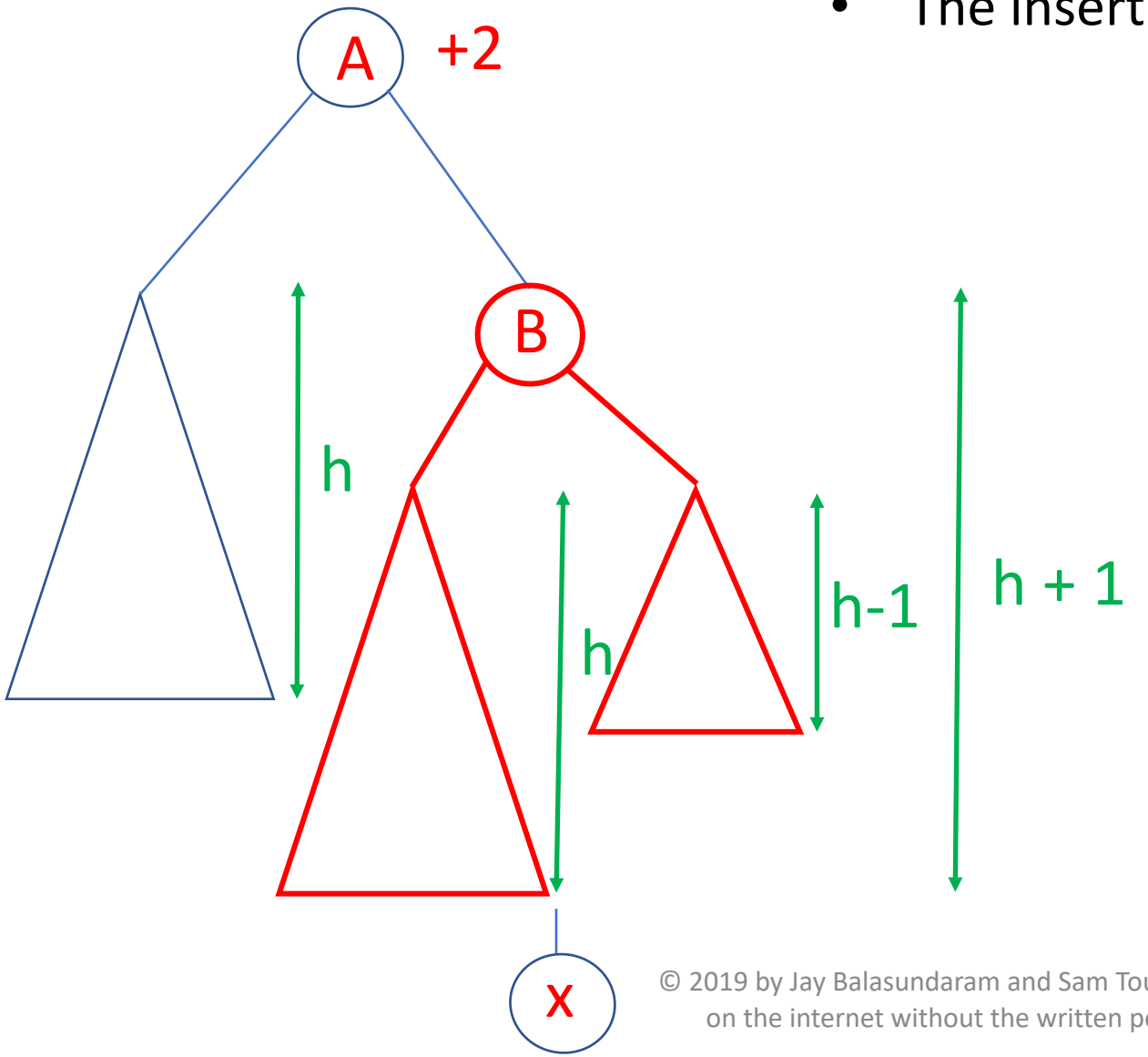
What if only one of them has height h ?

or this

h + 1

But this is impossible!

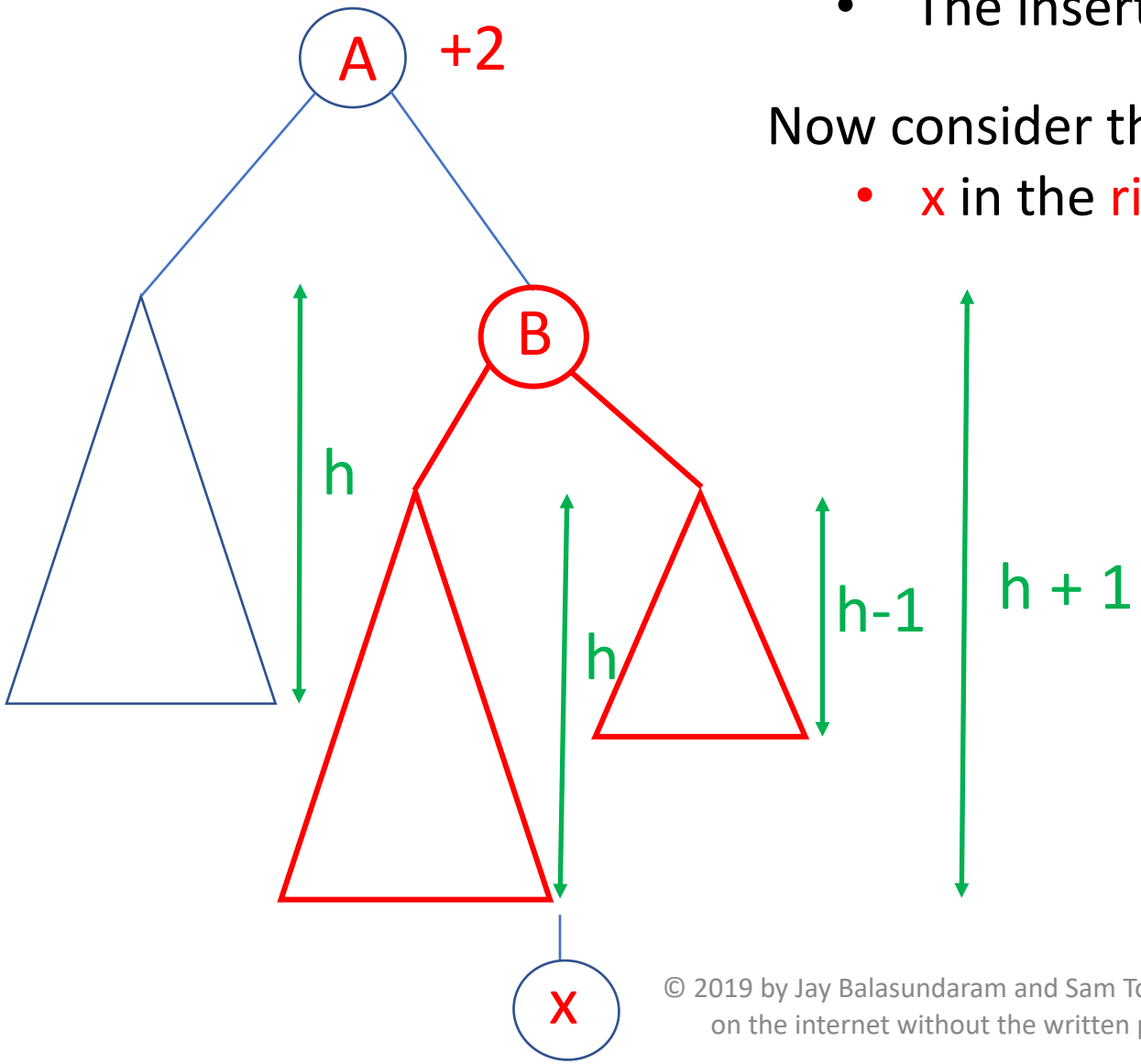
Case 1



To see why it is impossible, note that:

- The insertion of x must have increased the height of B

Case 1



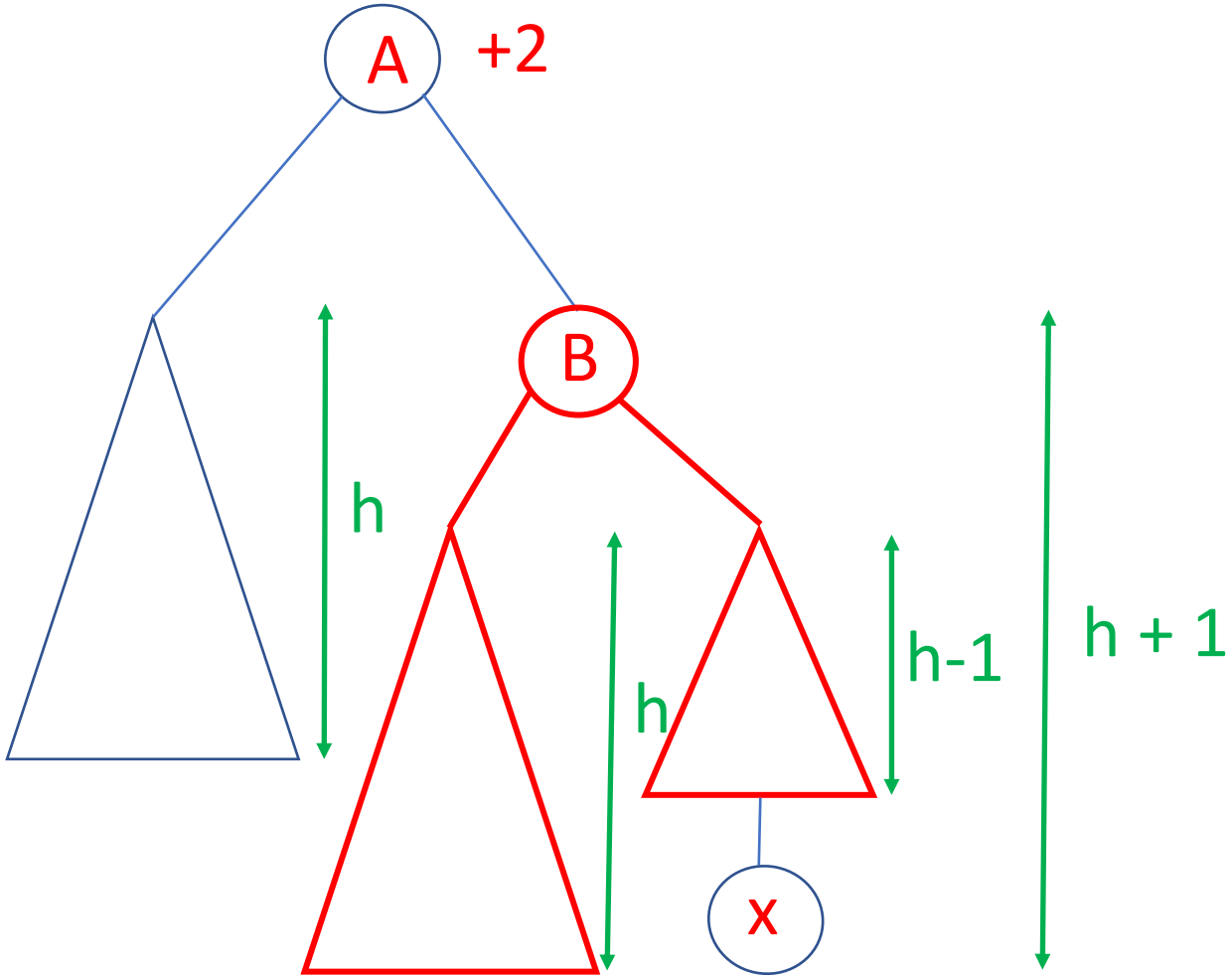
To see why it is impossible, note that:

- The insertion of x must have increased the height of B

Now consider the two possible cases:

- x in the right or left subtree of B

Case 1

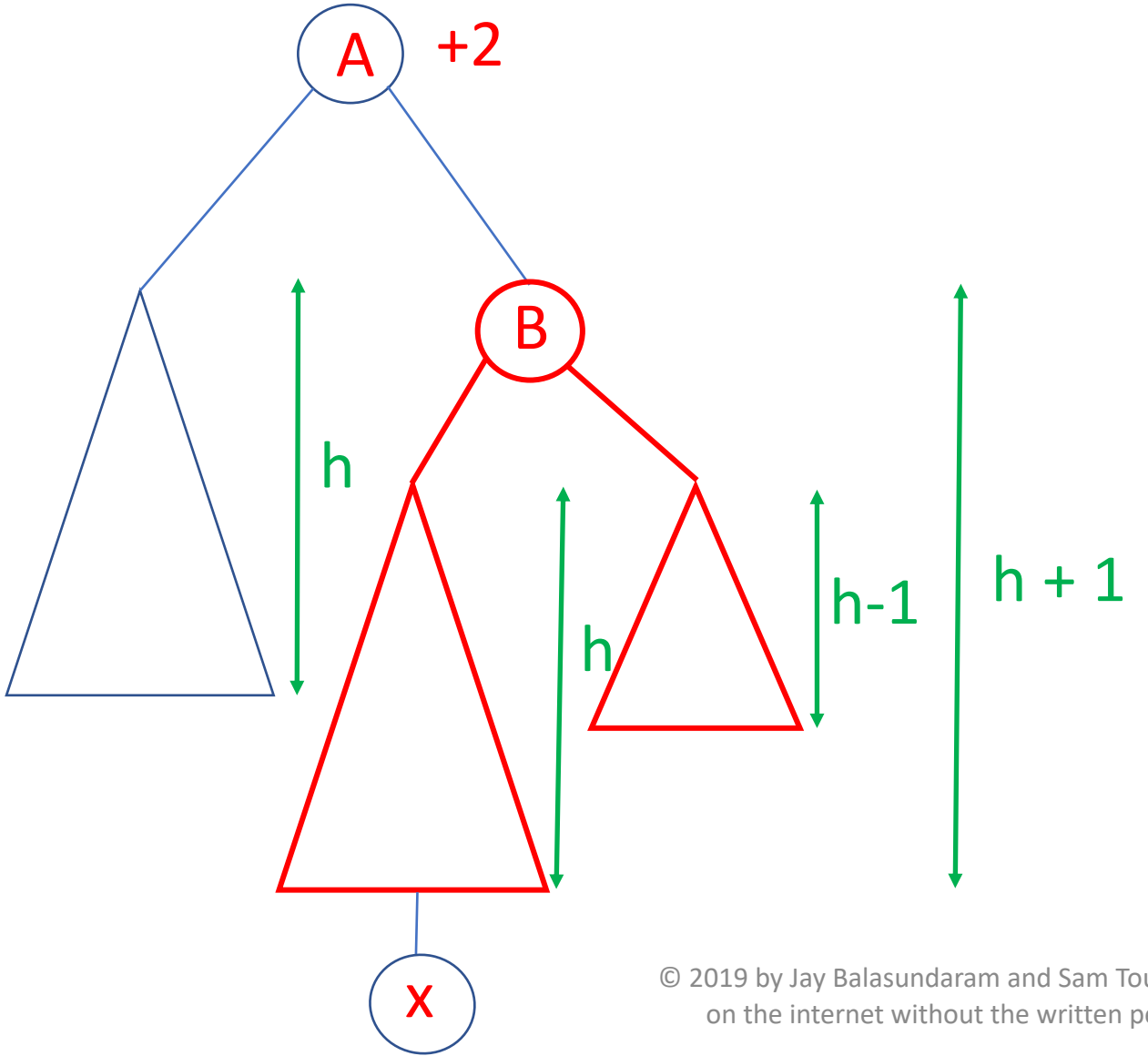


x in the right subtree of B

But this is impossible!

Do you see why?

Case 1

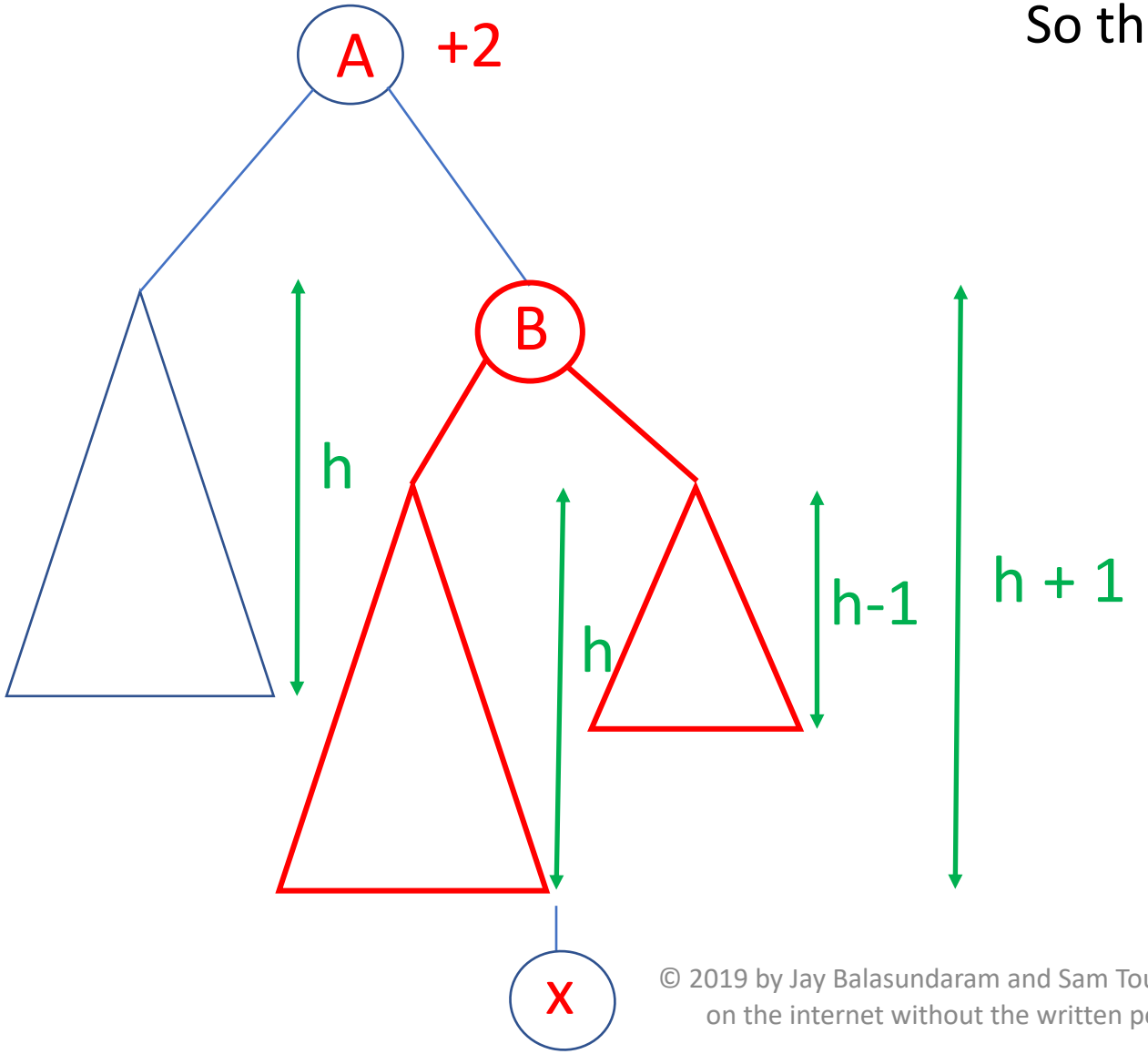


x in the left subtree of B

But this is also impossible!

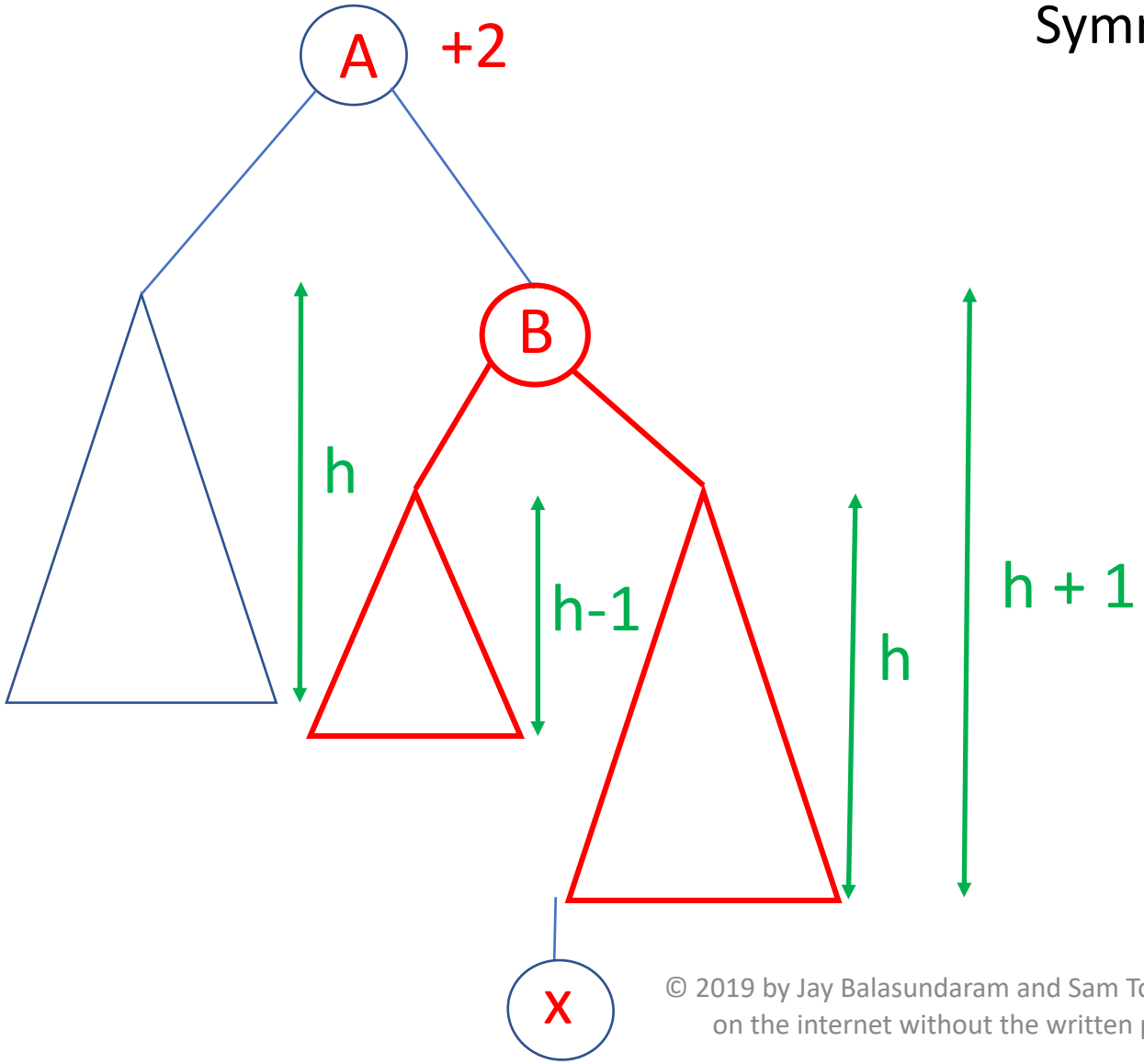
Do you see why?

Case 1



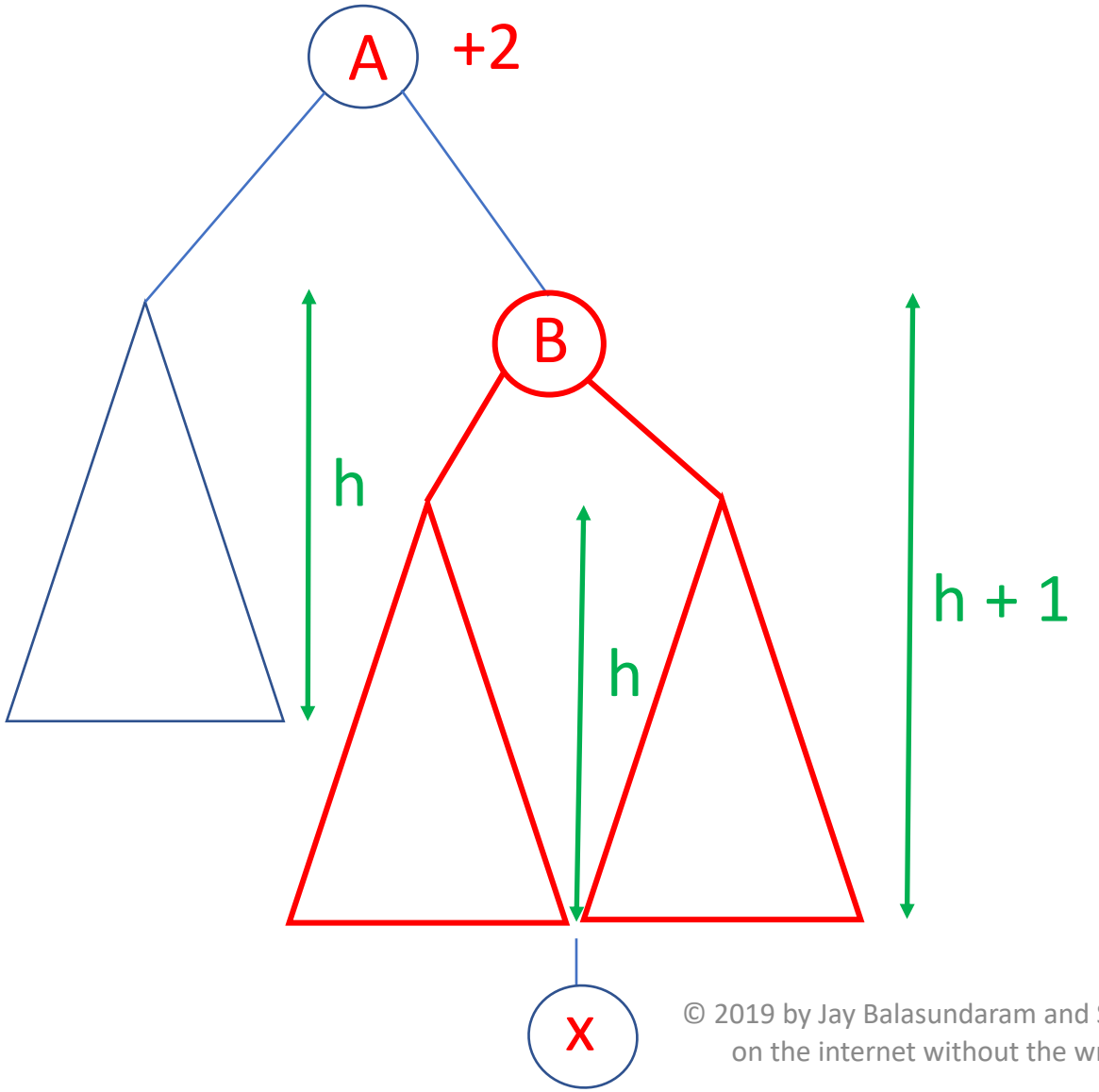
So this case is impossible!

Case 1



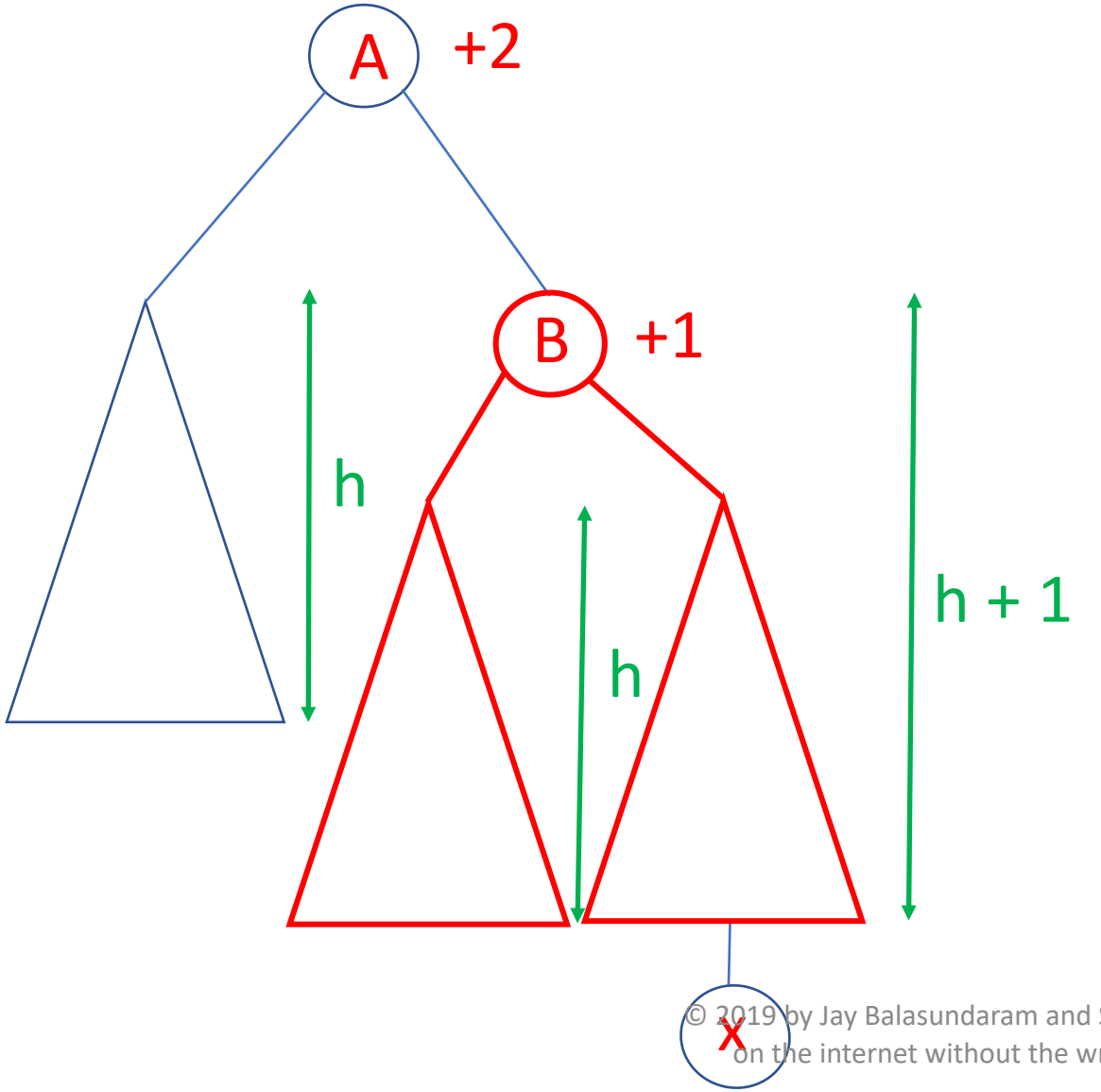
Symmetrically: this case is also impossible!

Case 1

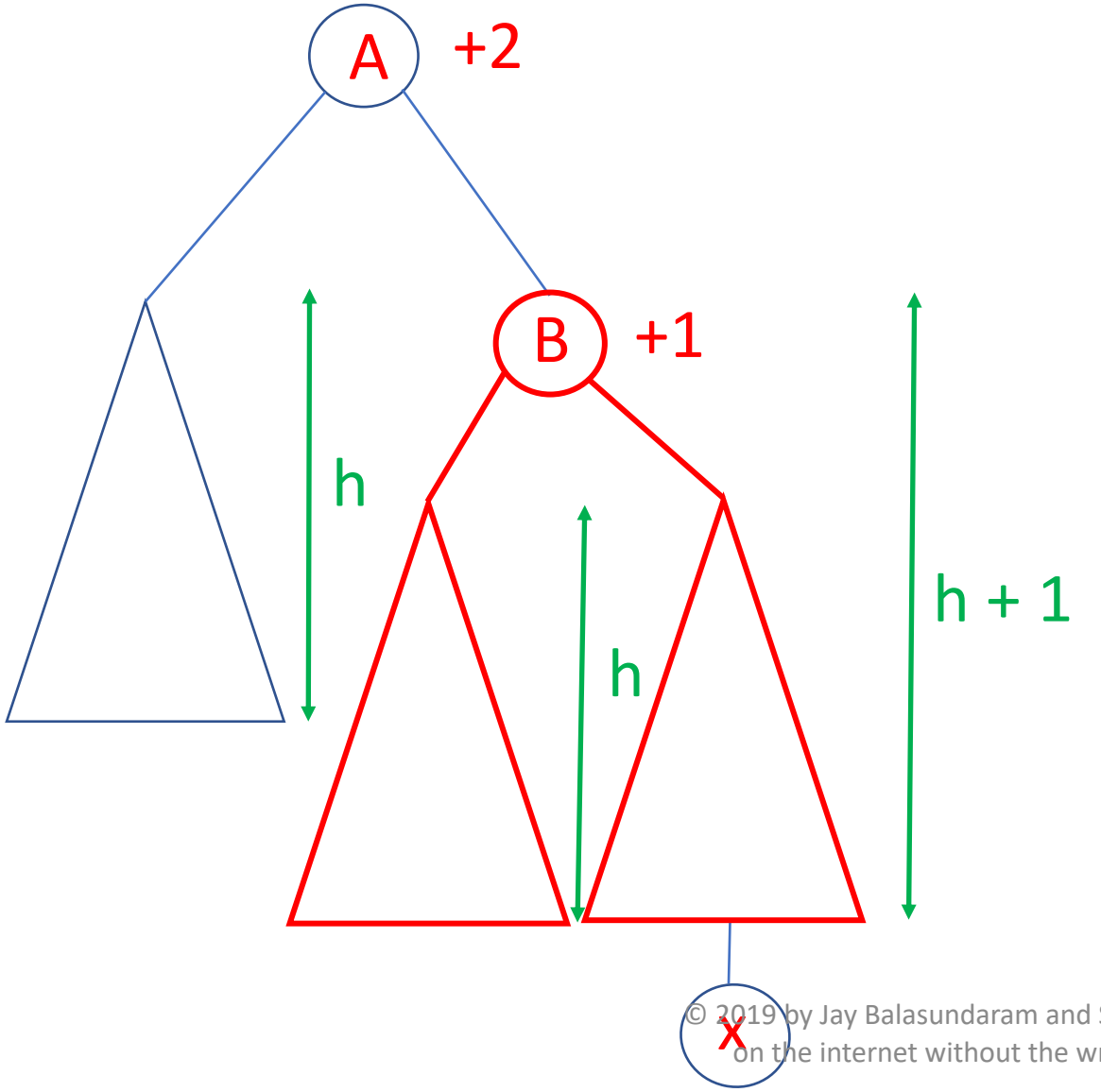


So this is indeed the only possible case 1

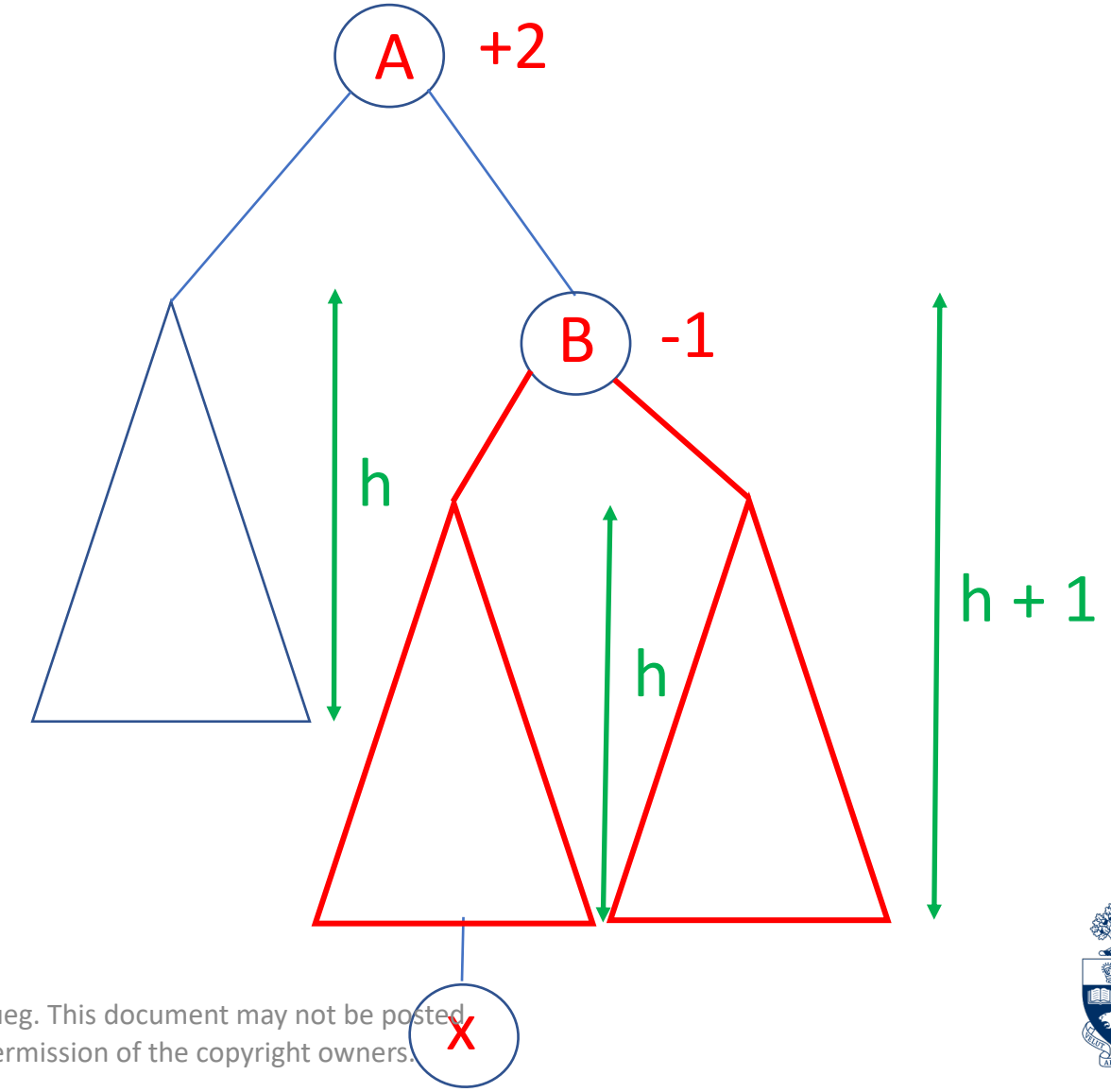
Case 1 (a)



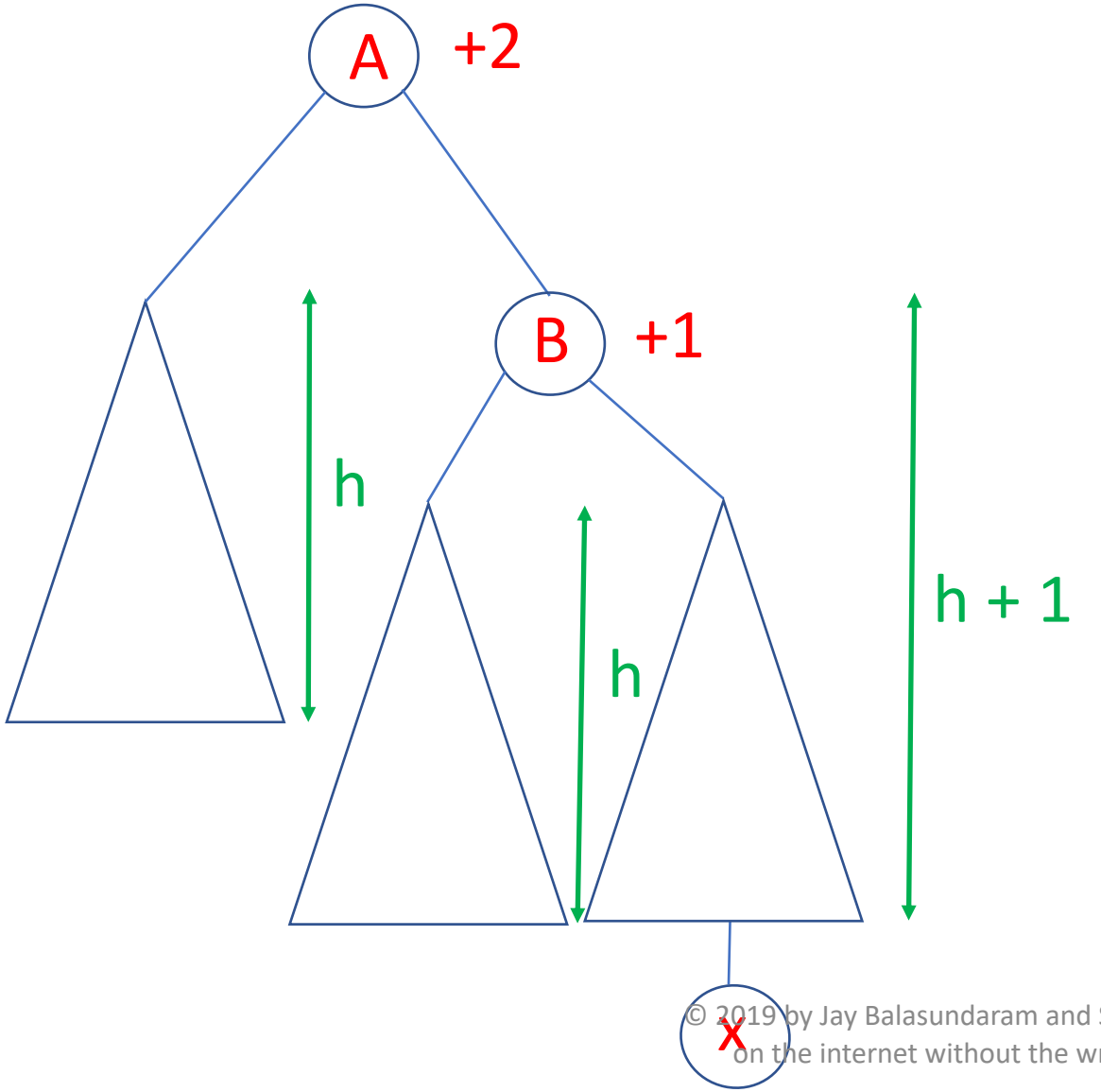
Case 1 (a)



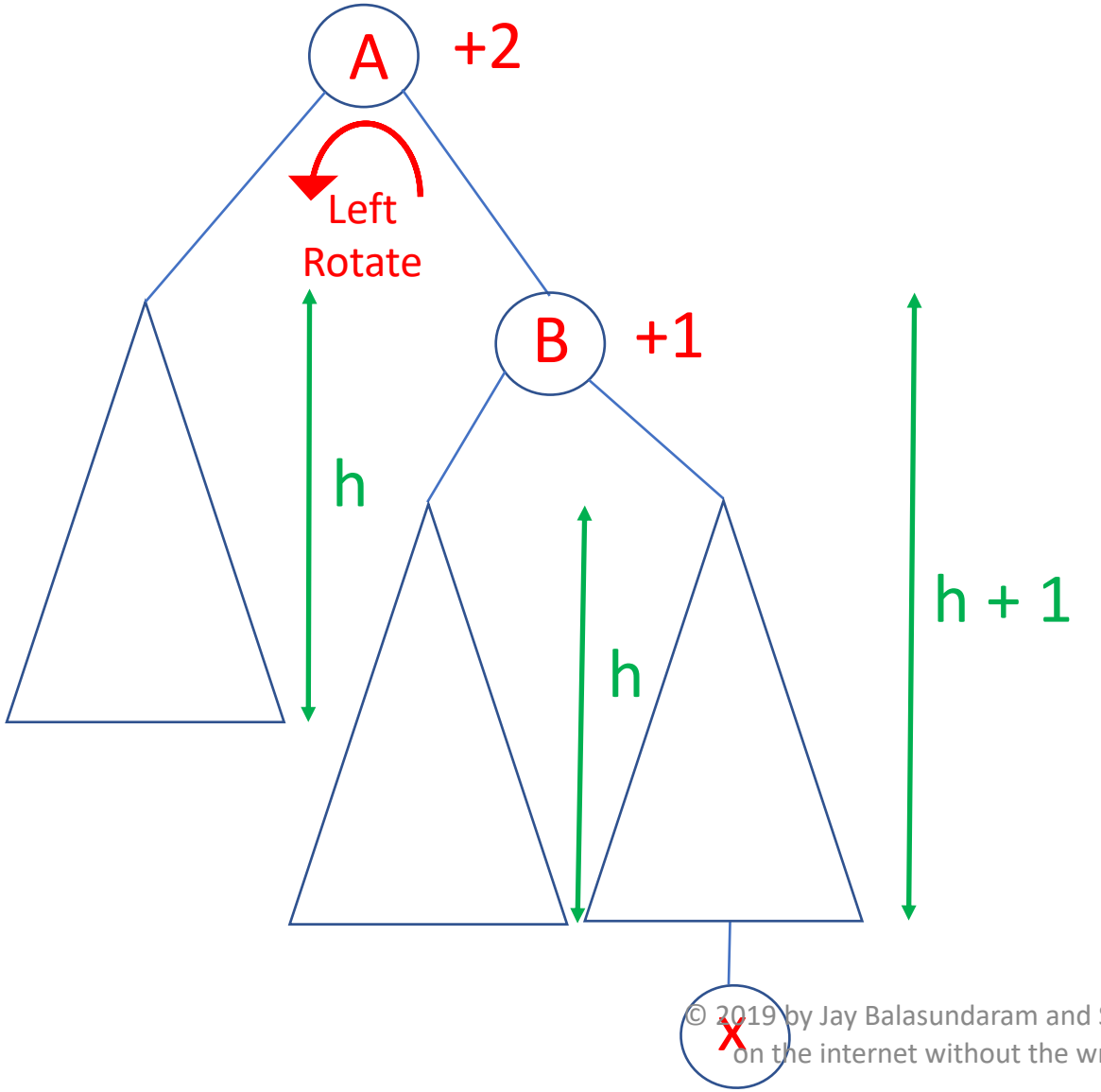
Case 1 (b)



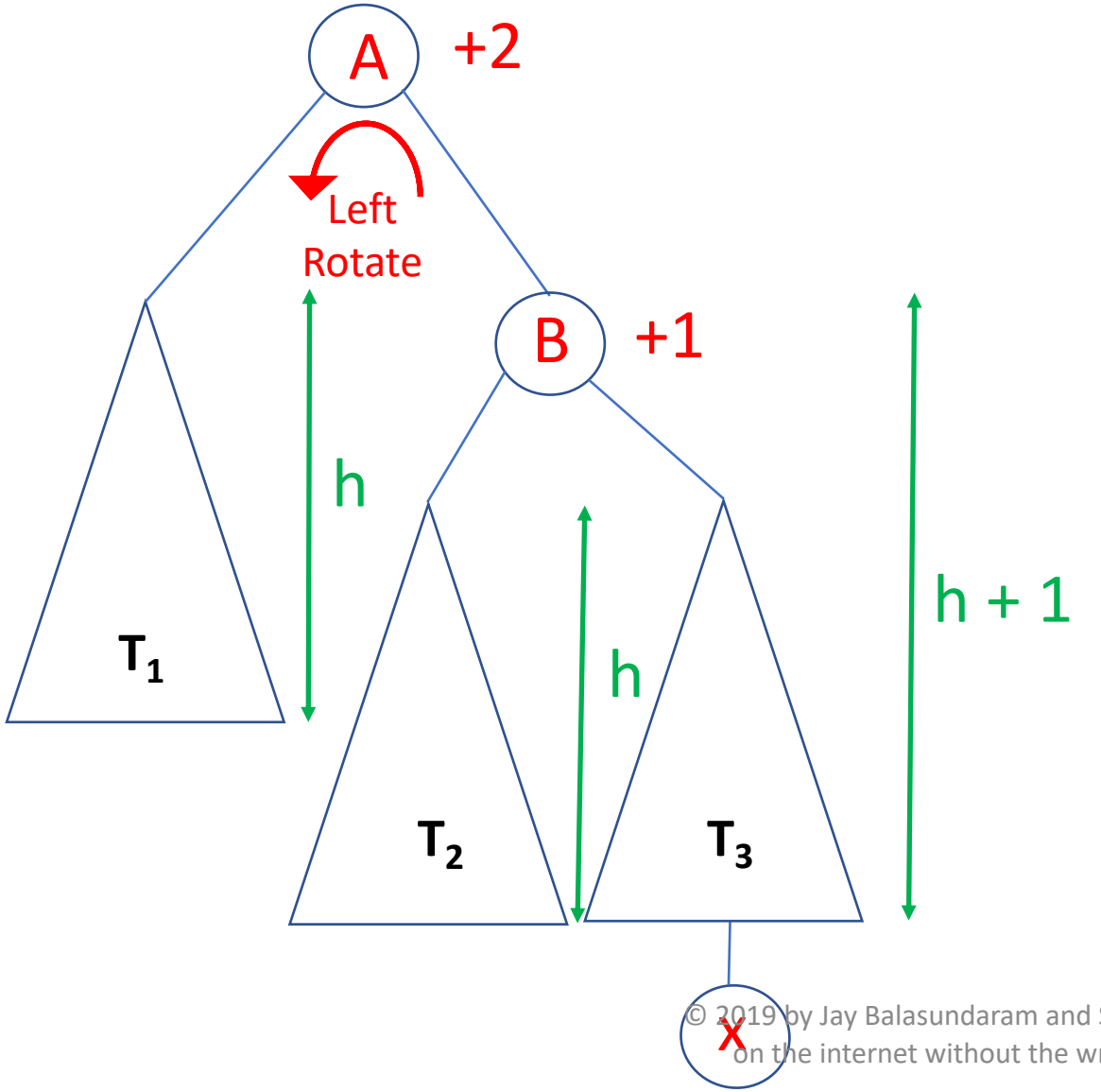
Case 1 (a)



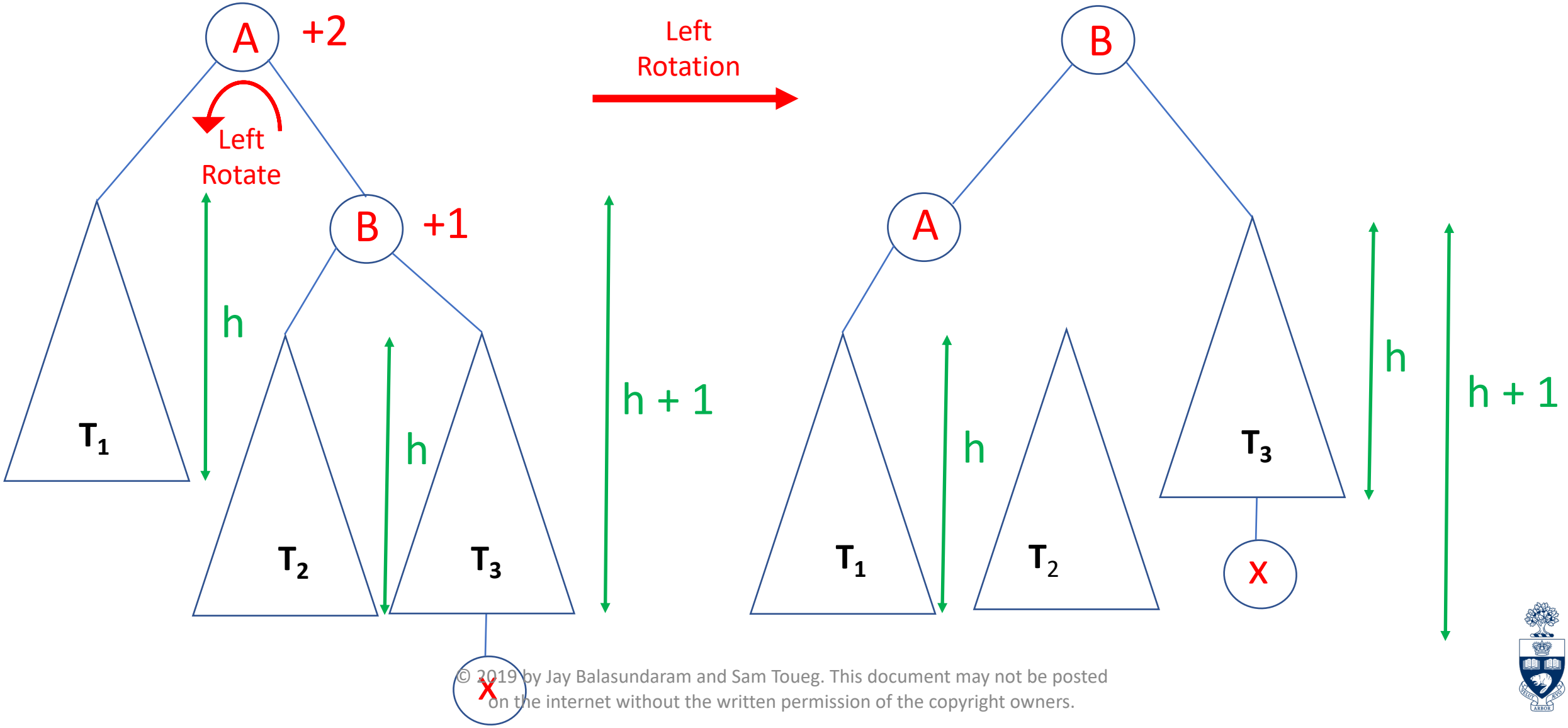
Case 1 (a)



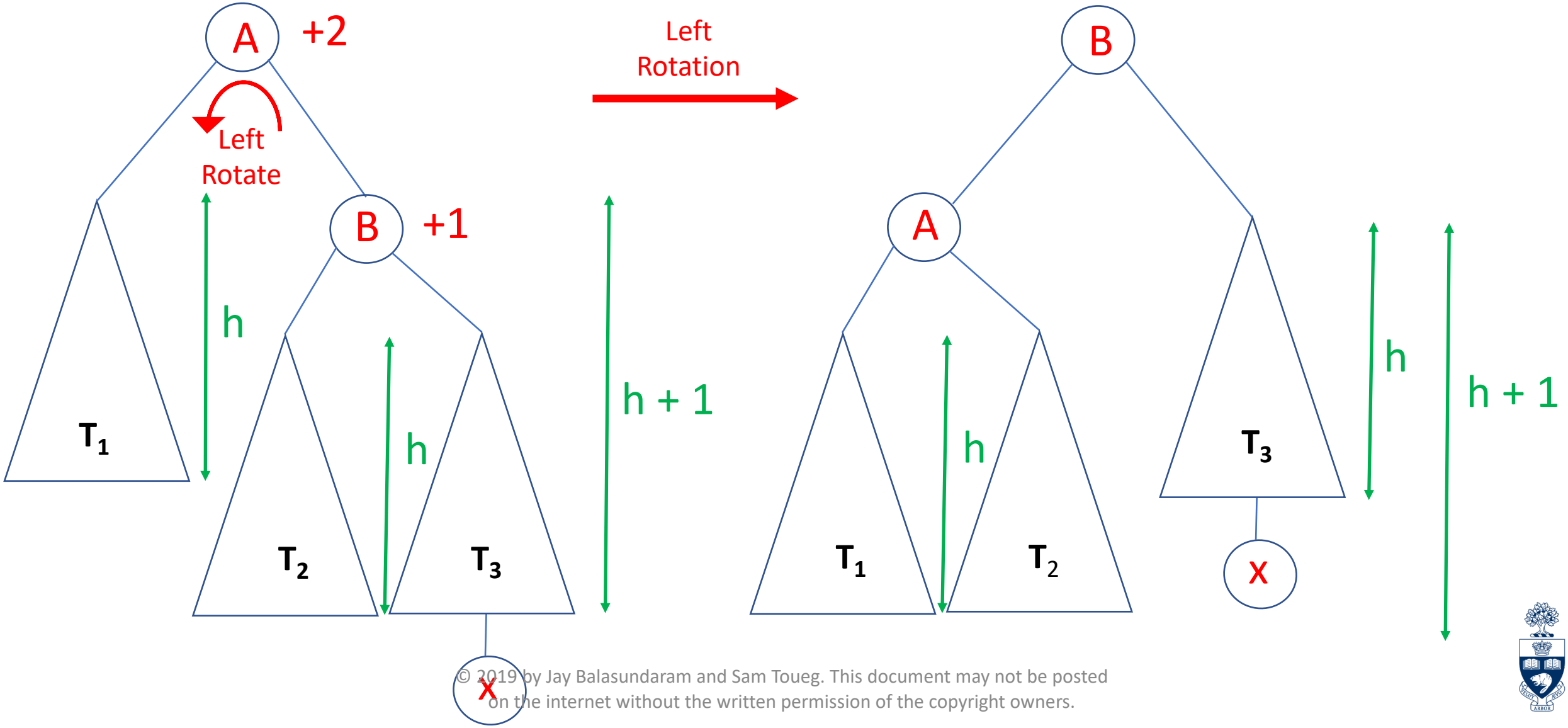
Case 1 (a)



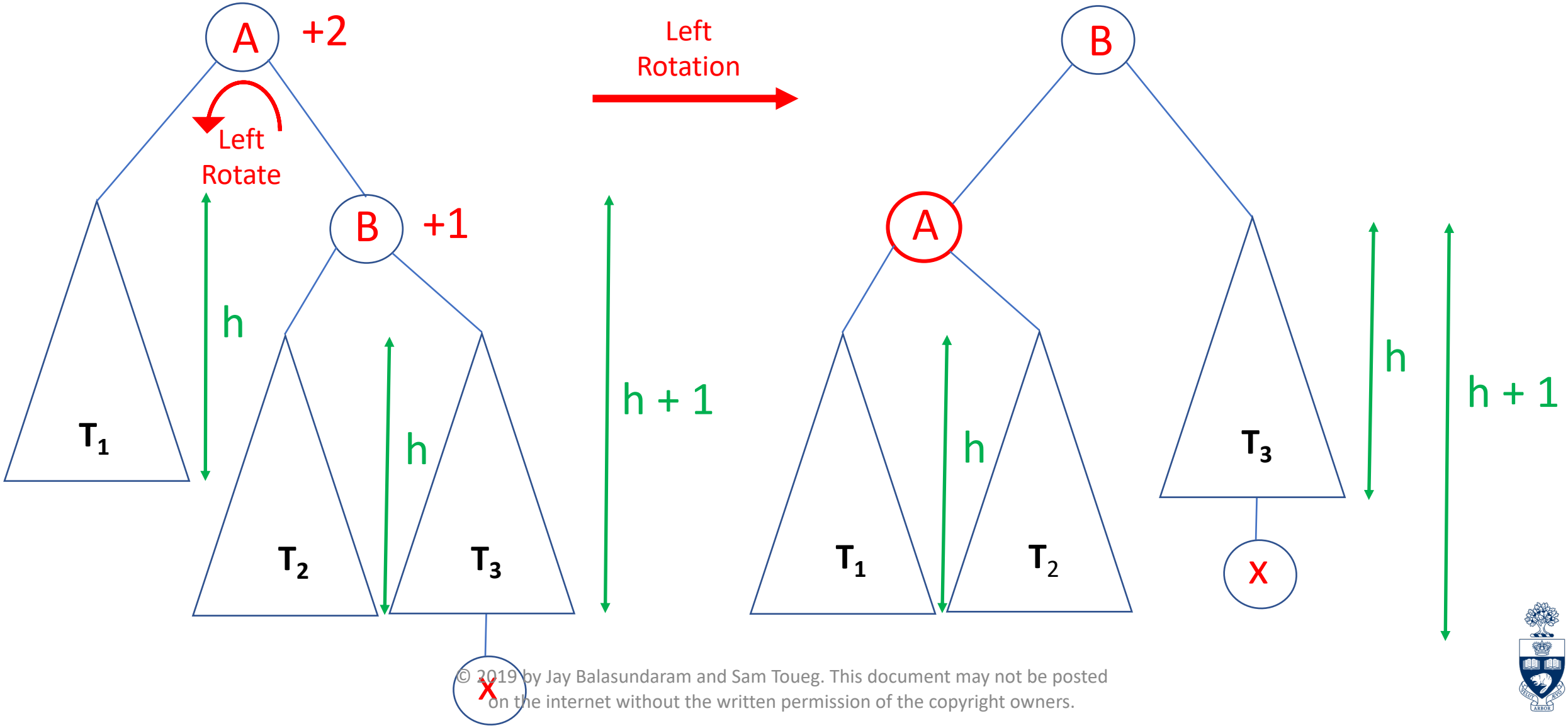
Case 1 (a)



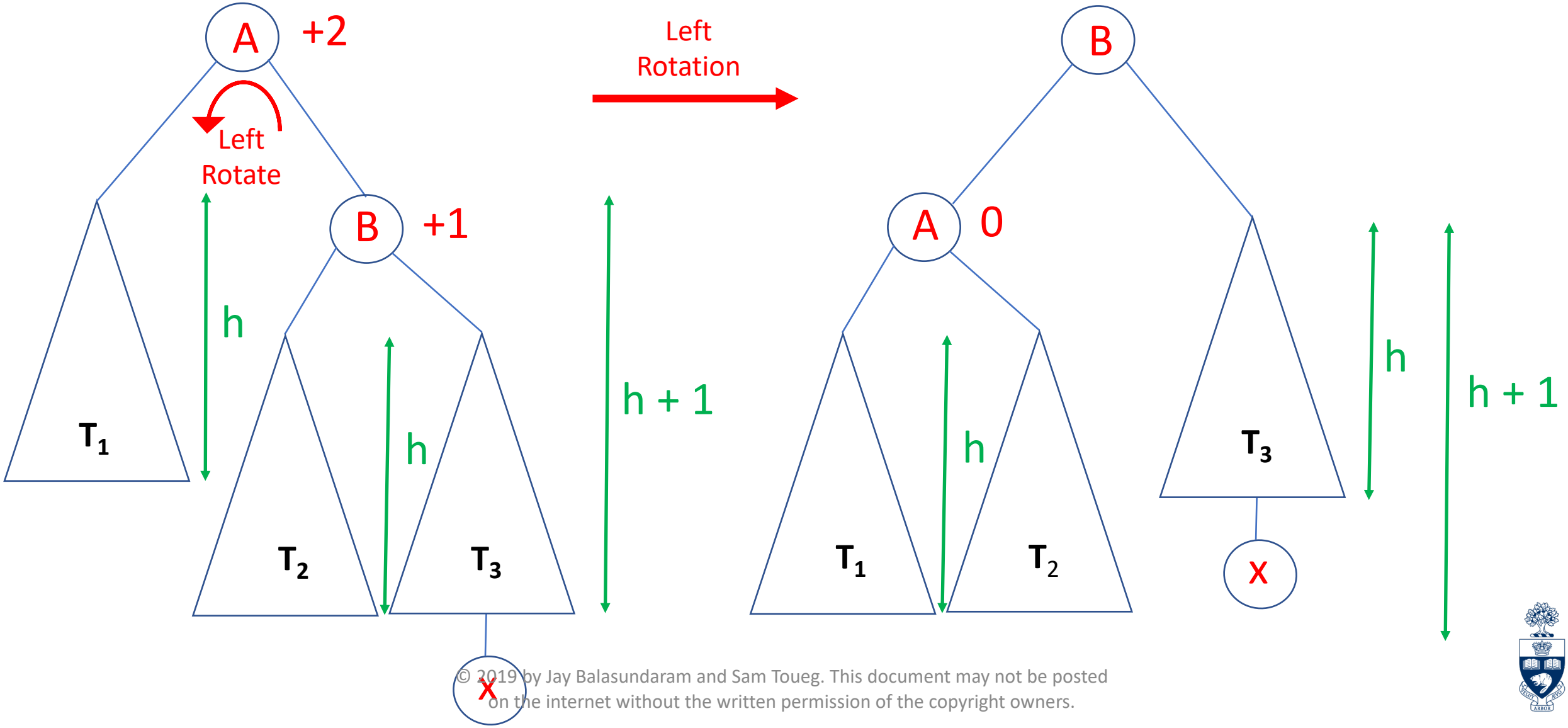
Case 1 (a)



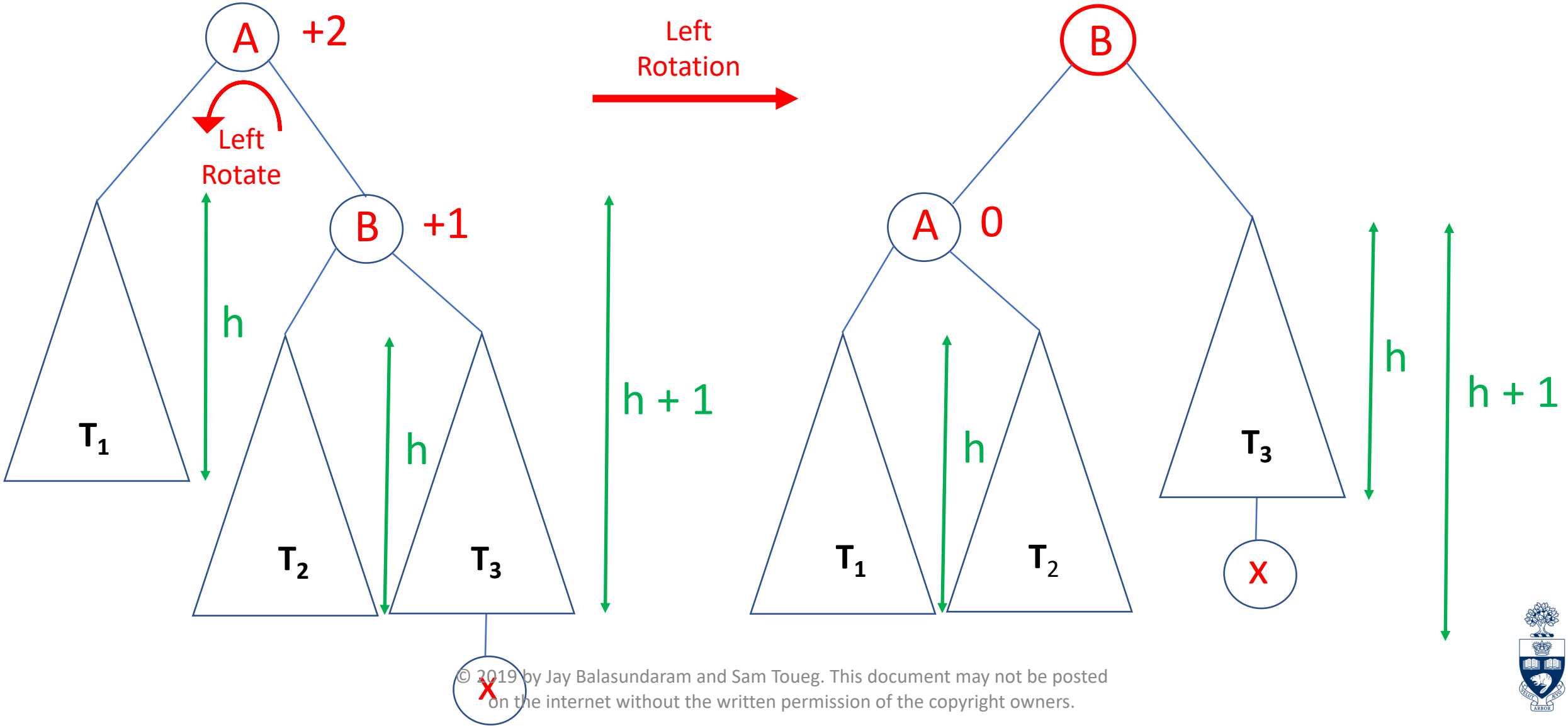
Case 1 (a)



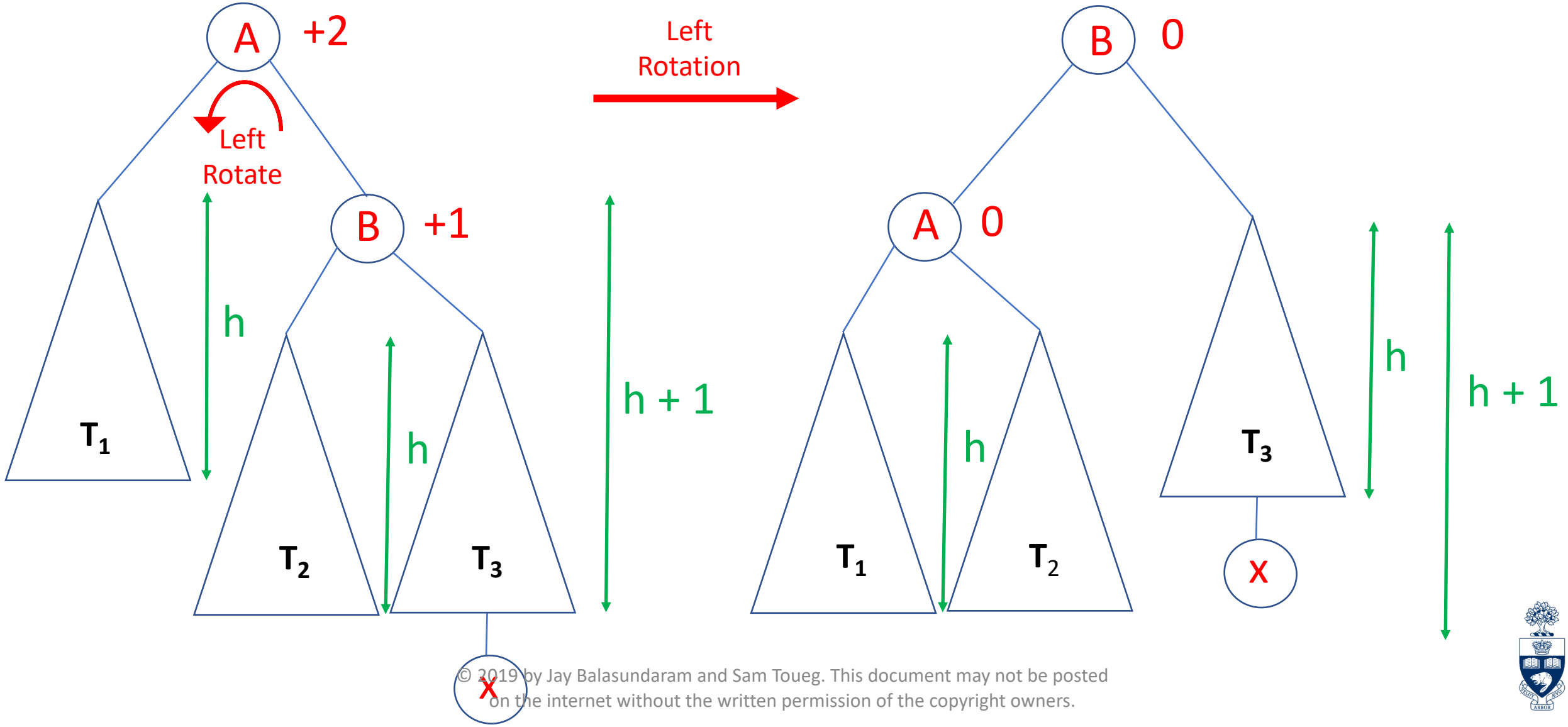
Case 1 (a)



Case 1 (a)

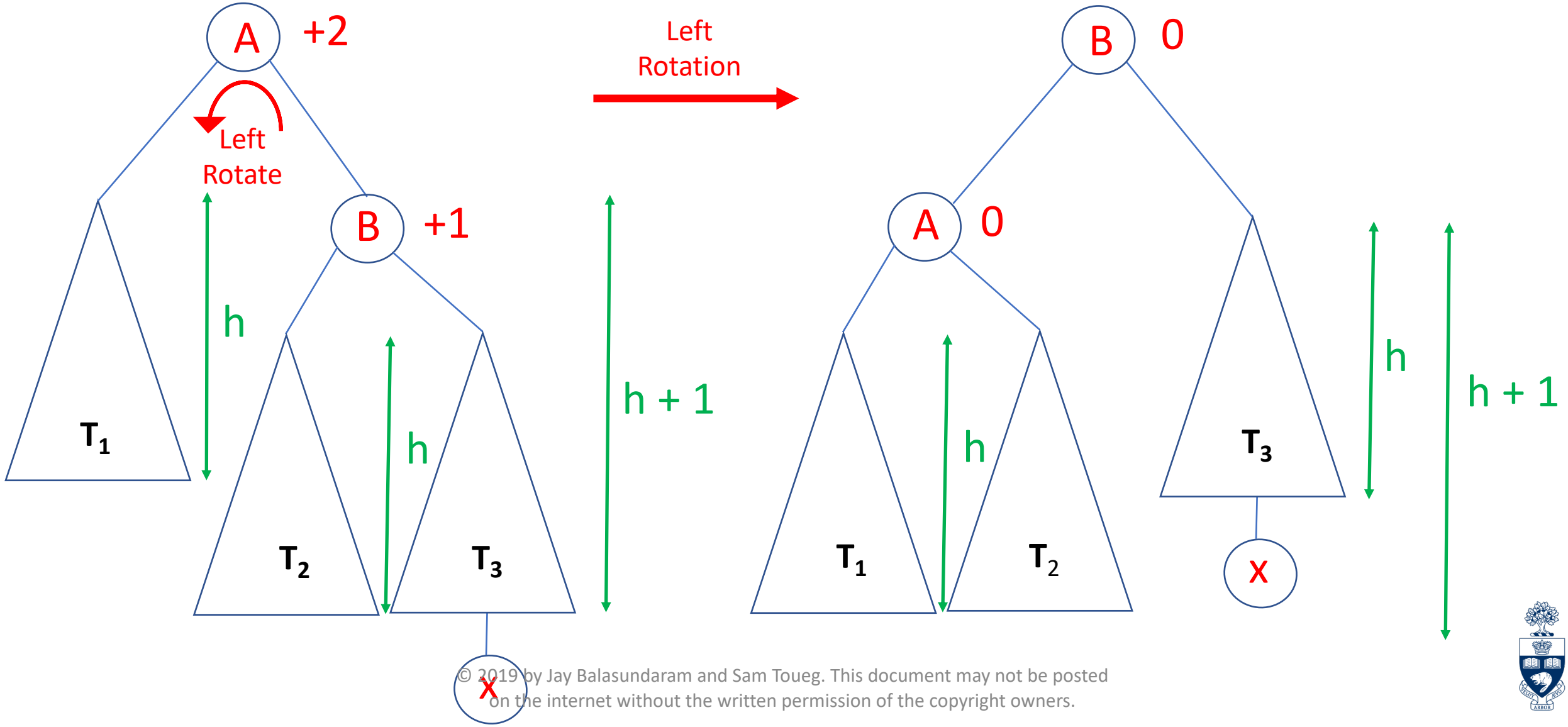


Case 1 (a)



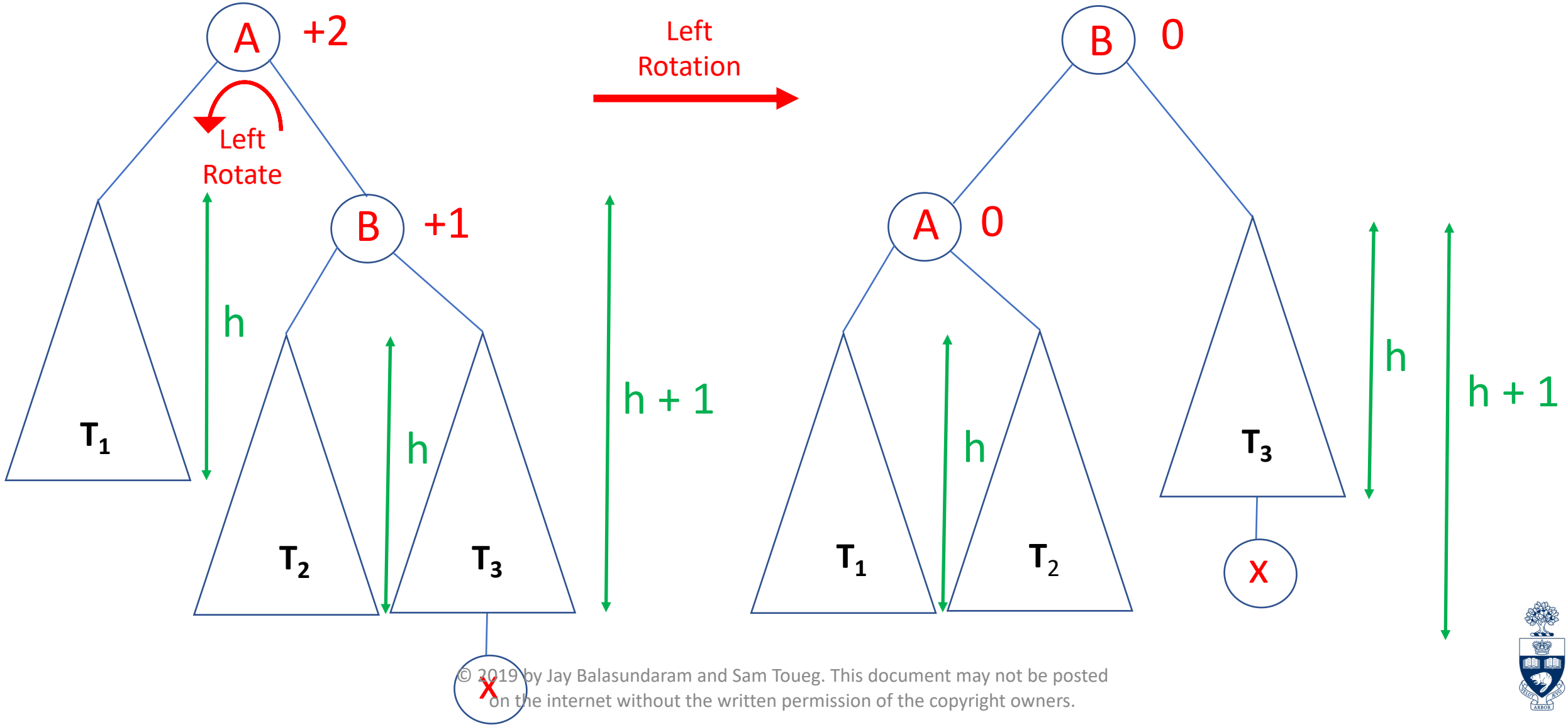
Rotation: (1) Rebalances the subtree

Case 1 (a)



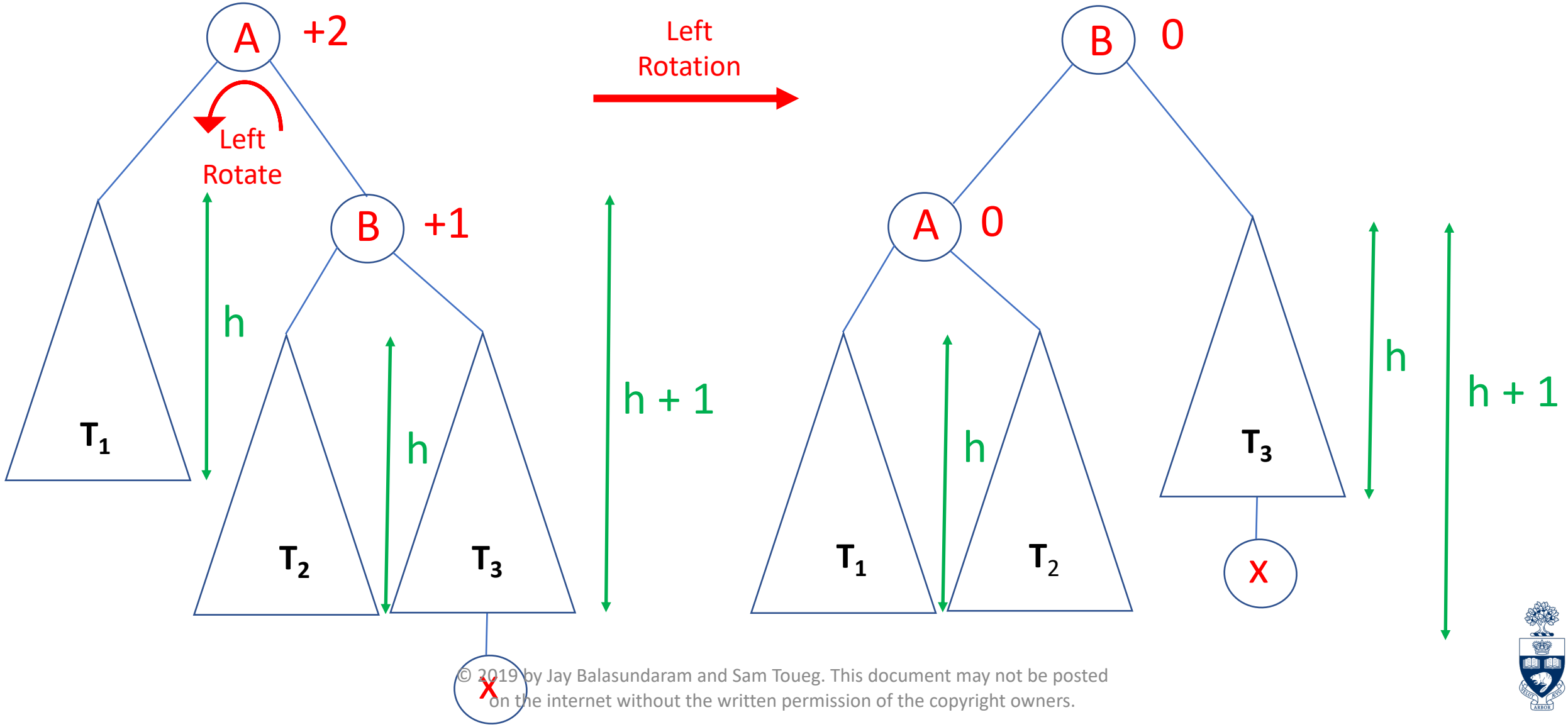
Rotation: (2) Preserves BST property

Case 1 (a)



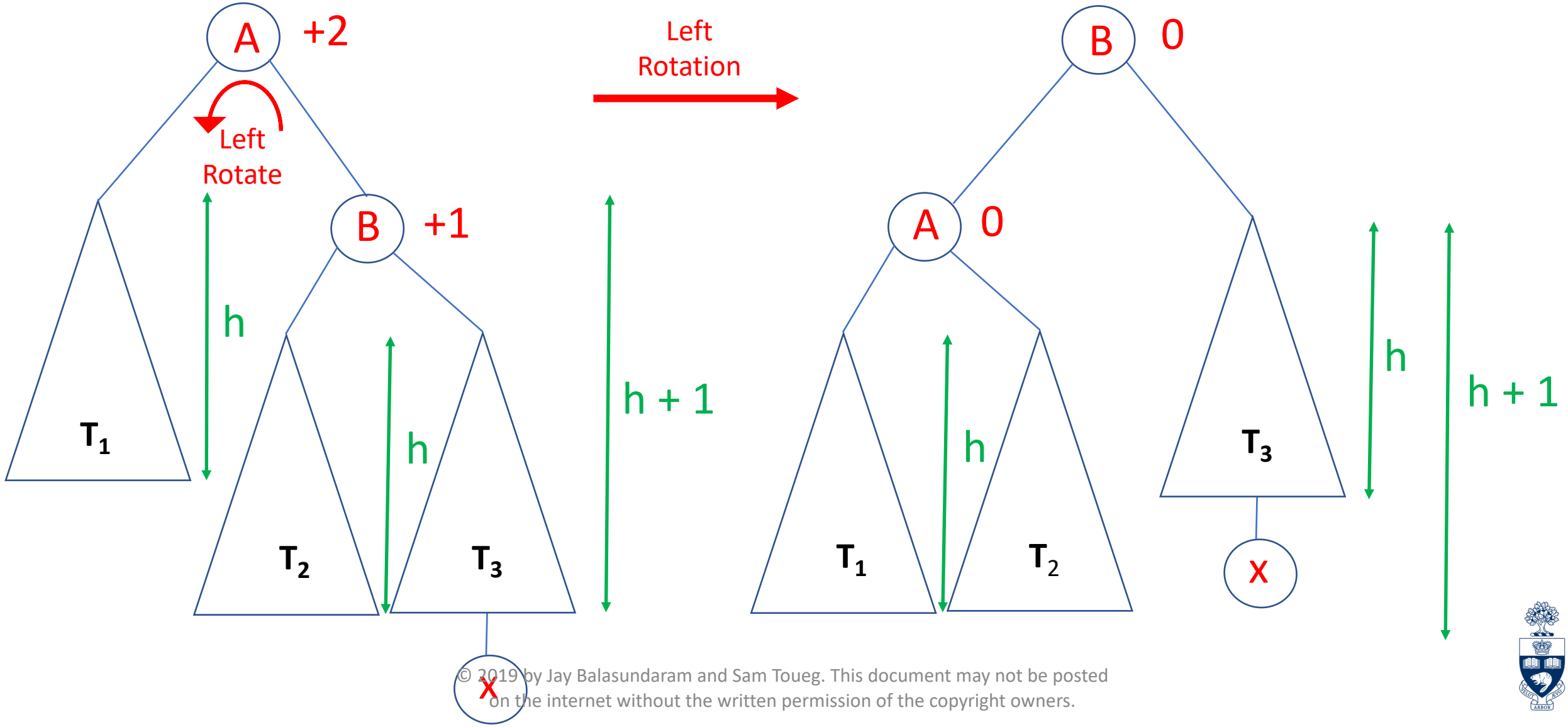
Rotation: (2) Preserves BST property (order:)

Case 1 (a)



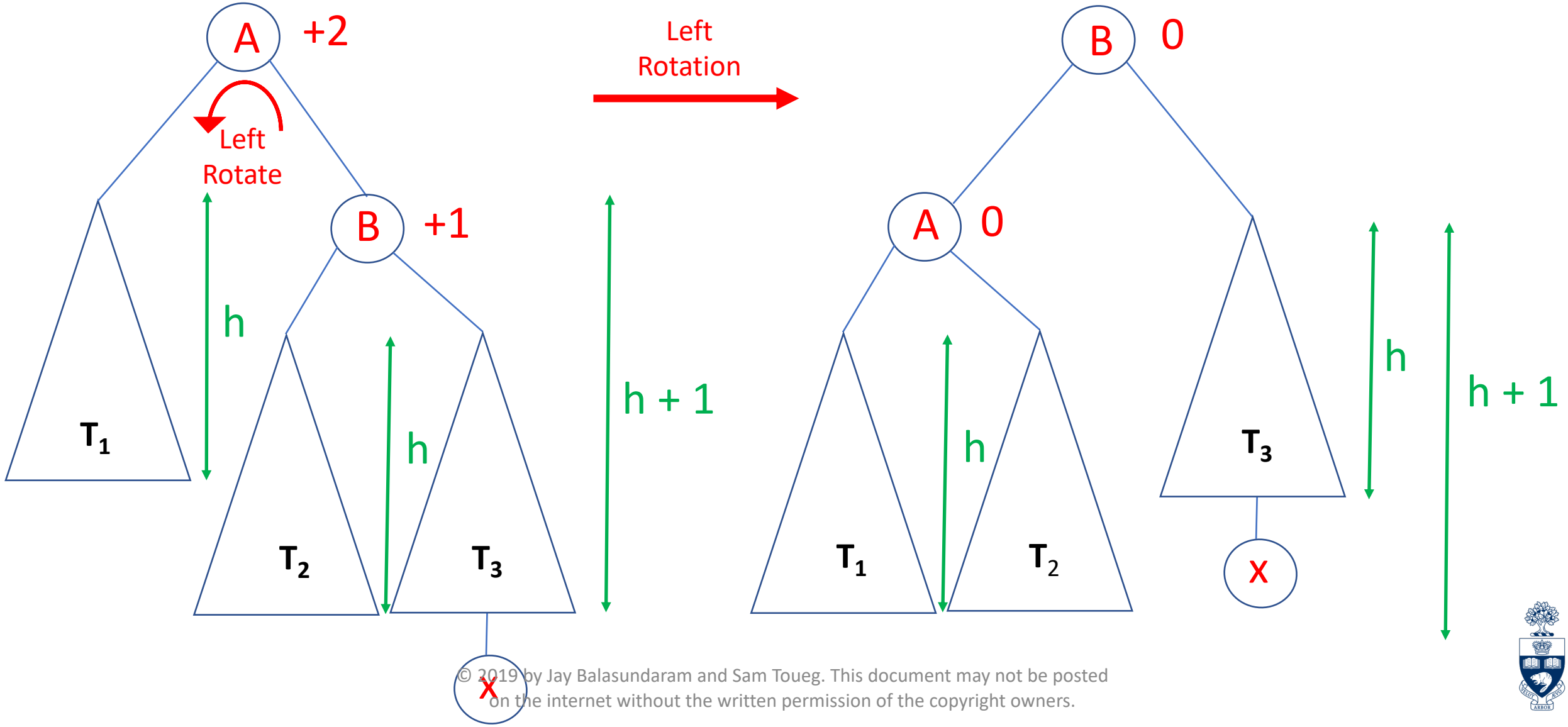
Rotation: (2) Preserves BST property (order: T_1)

Case 1 (a)



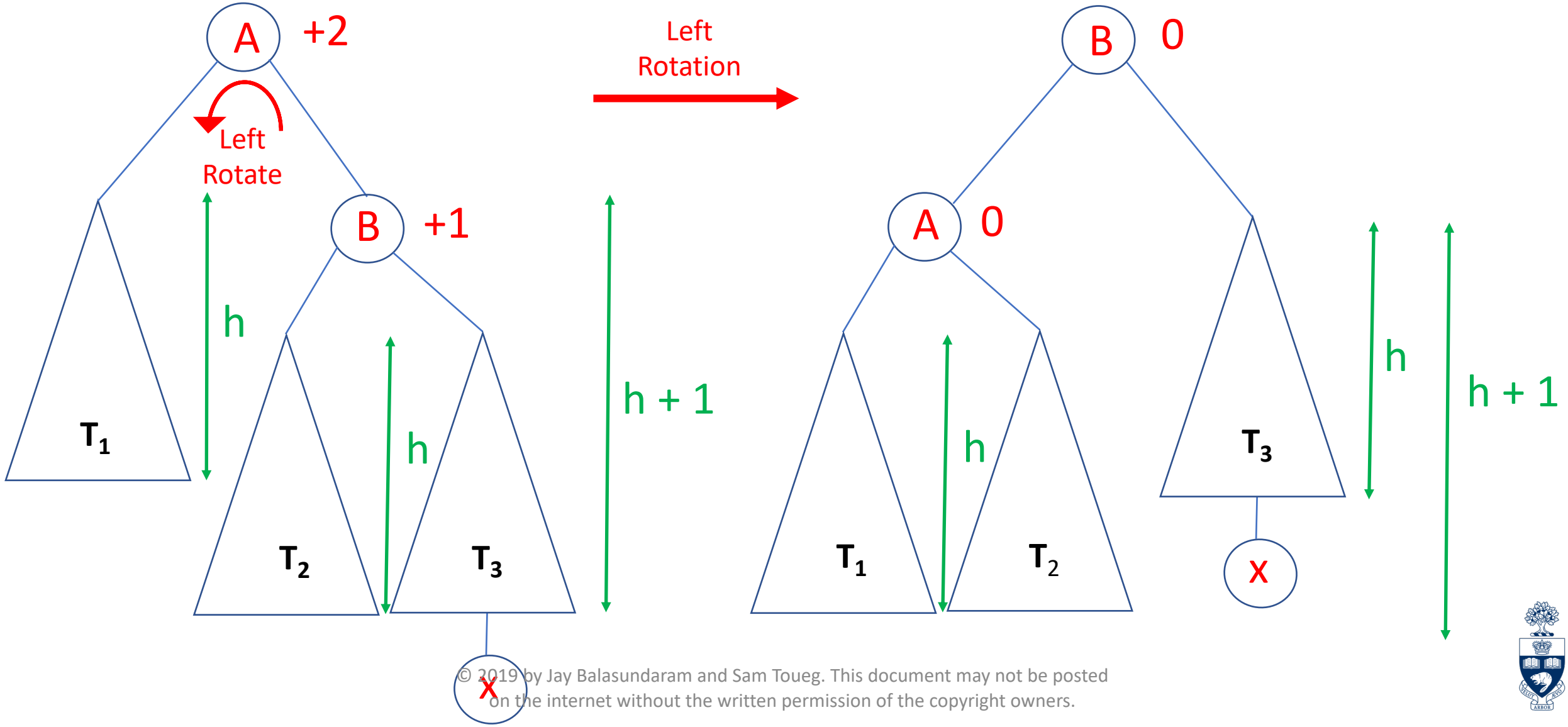
Rotation: (2) Preserves BST property (order: T_1 A)

Case 1 (a)



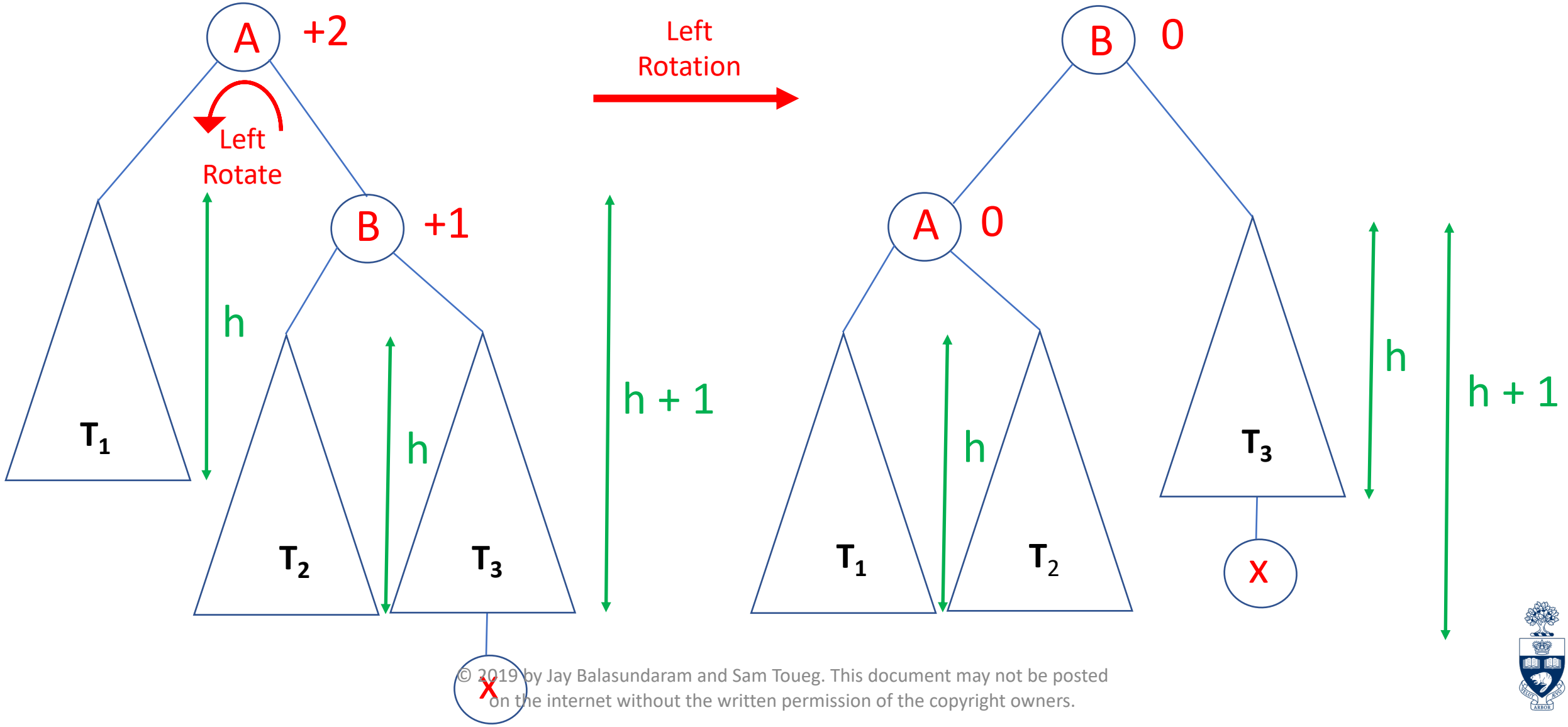
Rotation: (2) Preserves BST property (order: T_1 A T_2)

Case 1 (a)



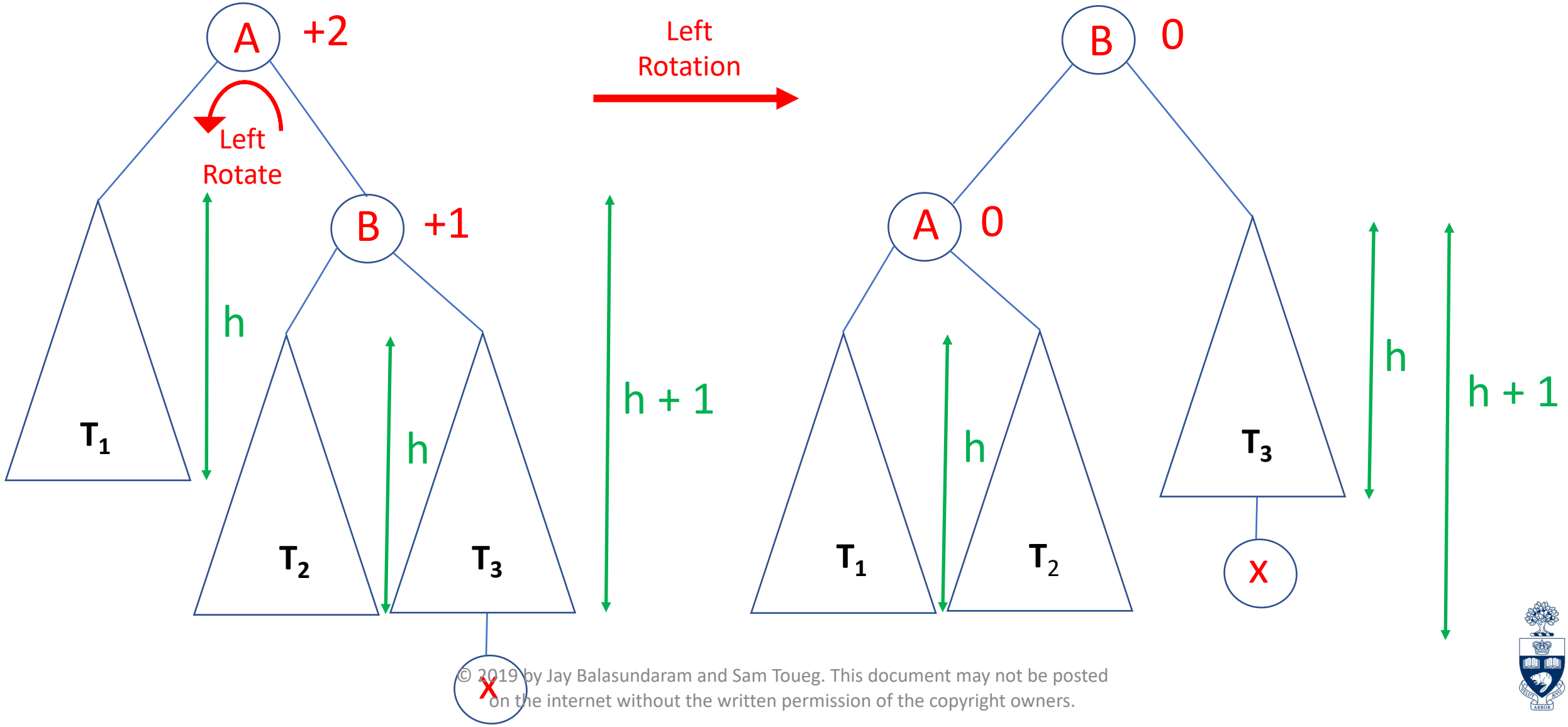
Rotation: (2) Preserves BST property (order: T_1 A T_2 B)

Case 1 (a)



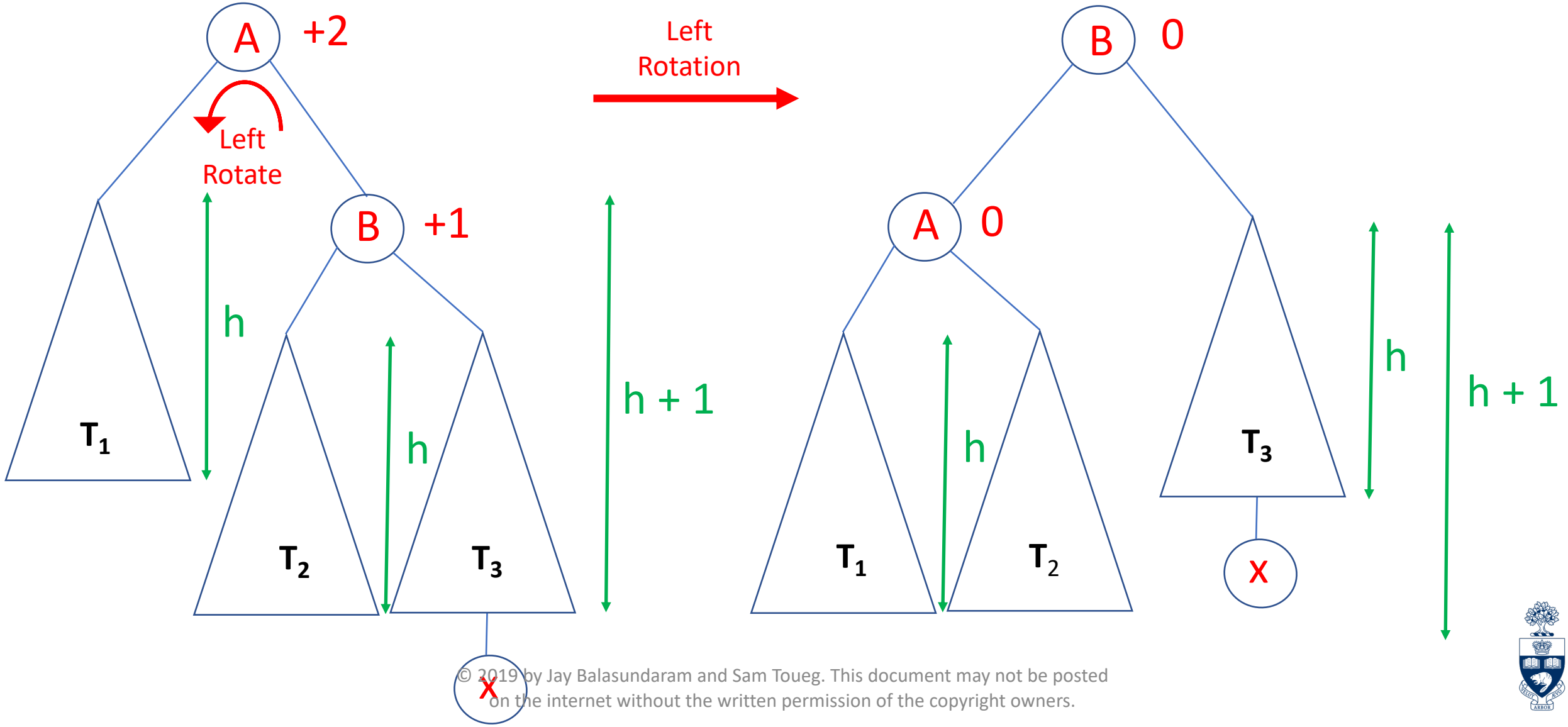
Rotation: (2) Preserves BST property (order: T_1 A T_2 B T_3)

Case 1 (a)



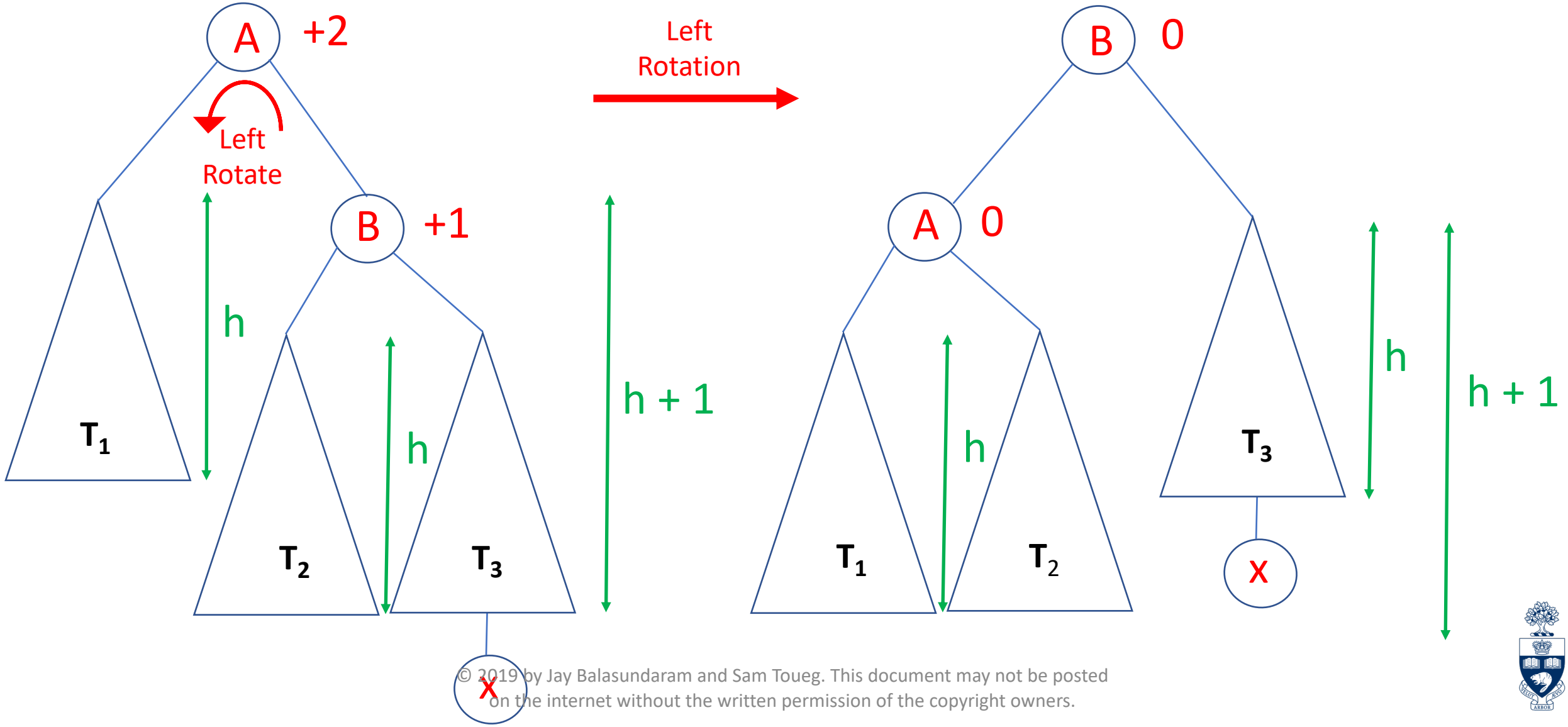
Rotation: (3) Bonus?

Case 1 (a)



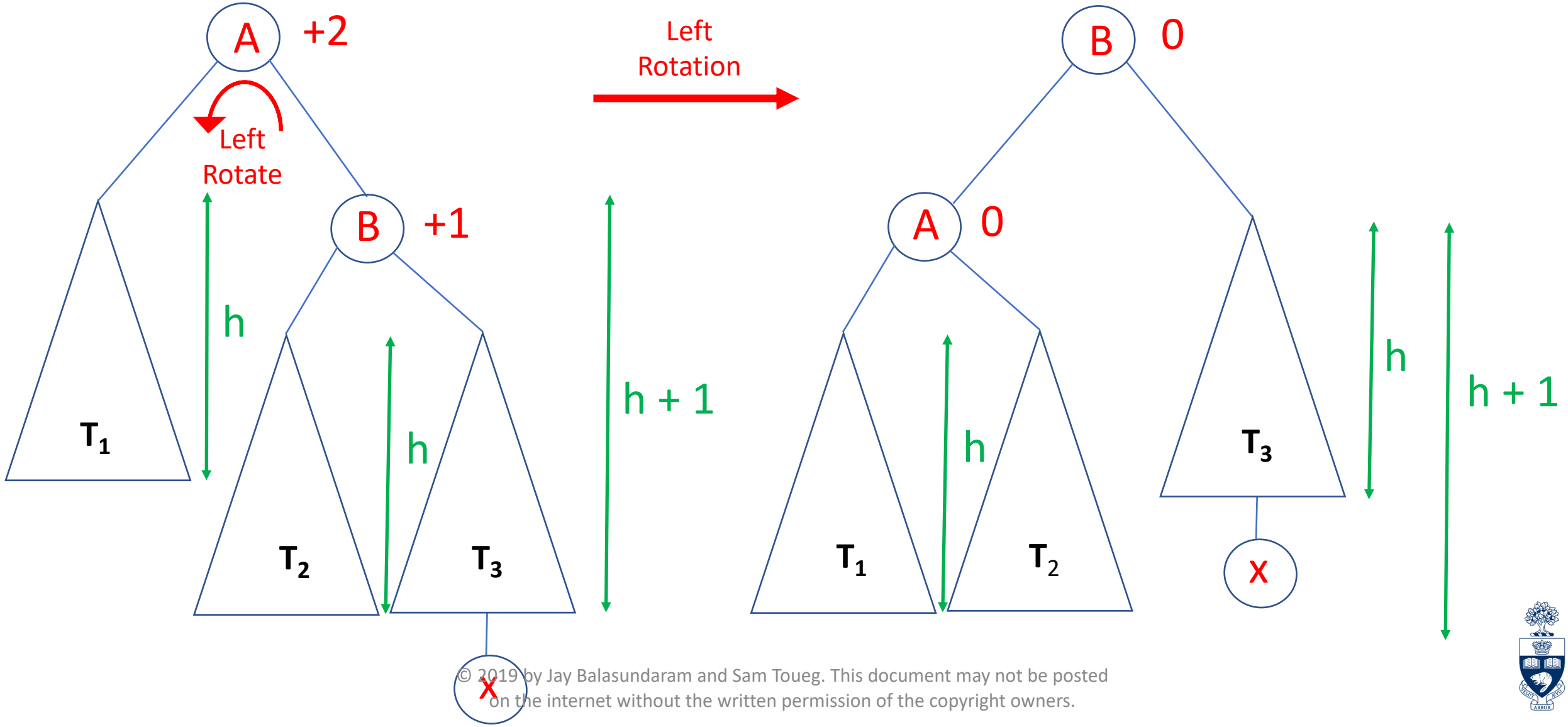
Rotation: (3) Bonus? YES!

Case 1 (a)



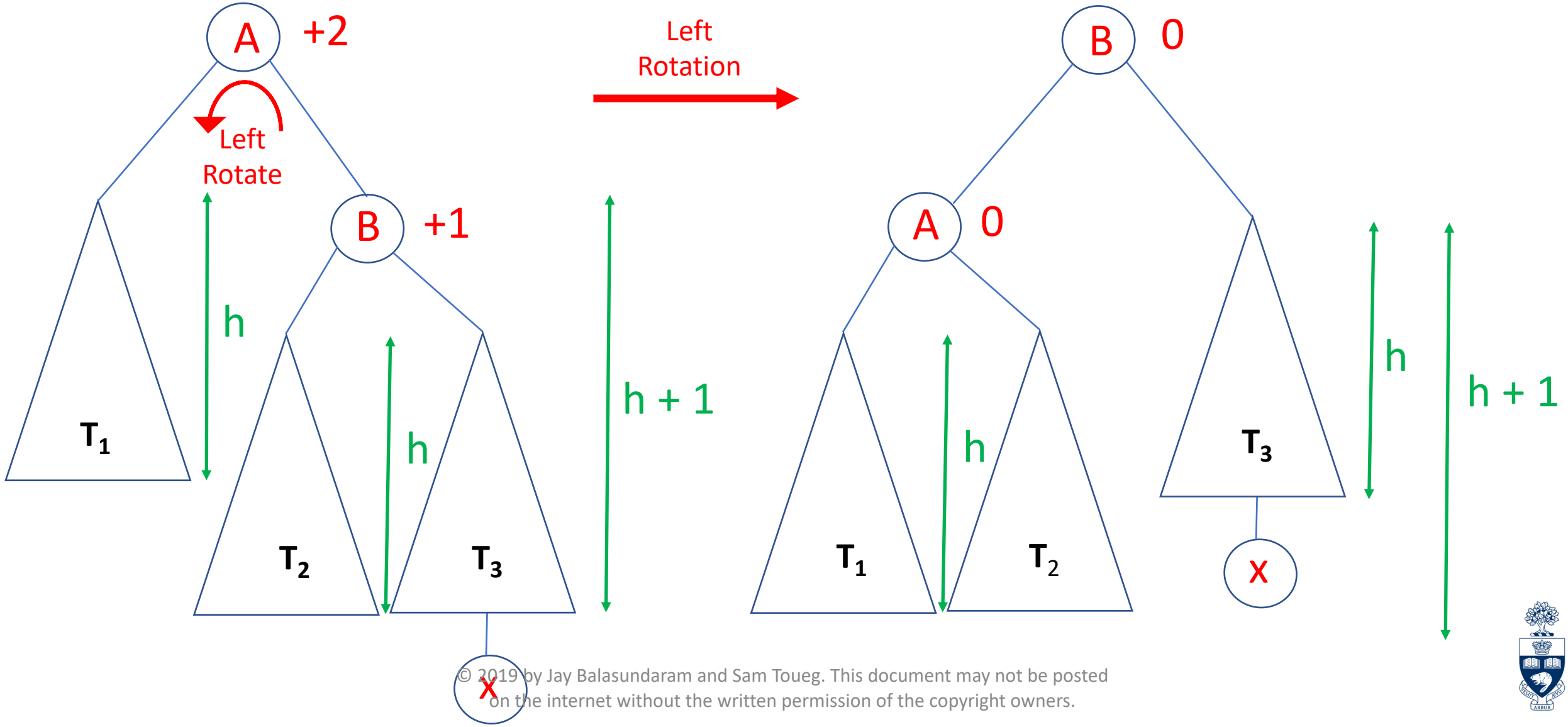
Rotation: (3) Bonus: height of this subtree is the same both before and after the insertion of x

Case 1 (a)



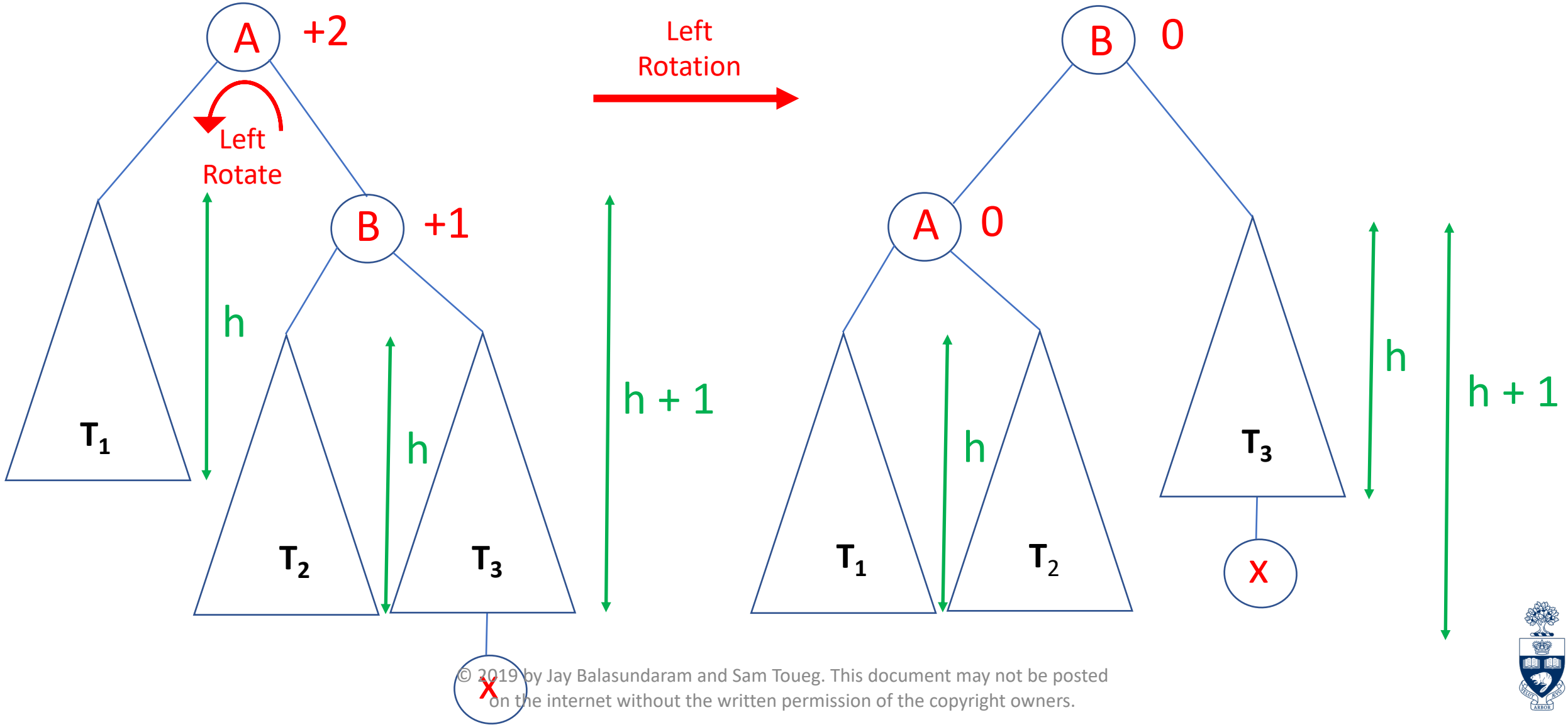
Rotation: (3) Bonus: height of this subtree is the same (h+2) both before and after the insertion of x

Case 1 (a)



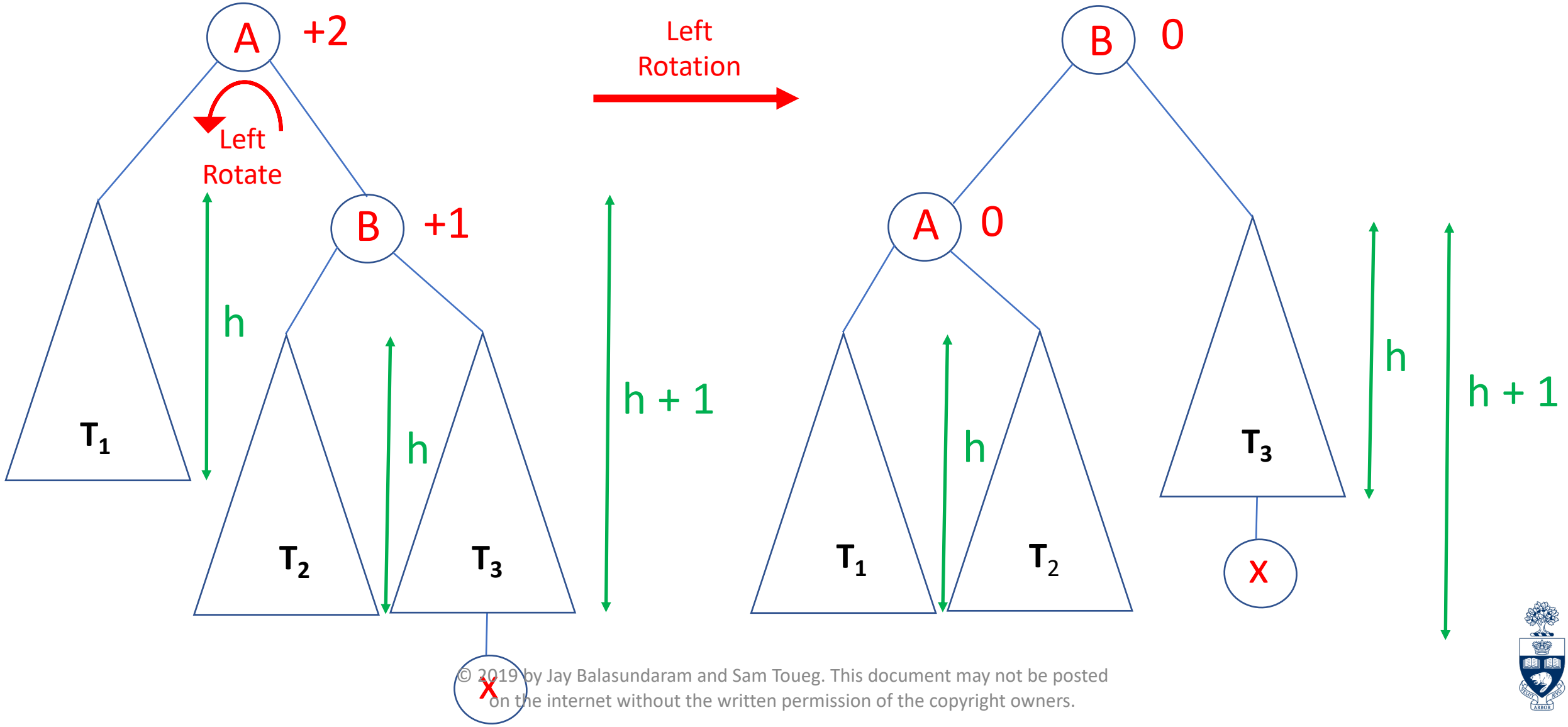
Rotation: (3) Bonus ⟹ insertion algorithm is done

Case 1 (a)



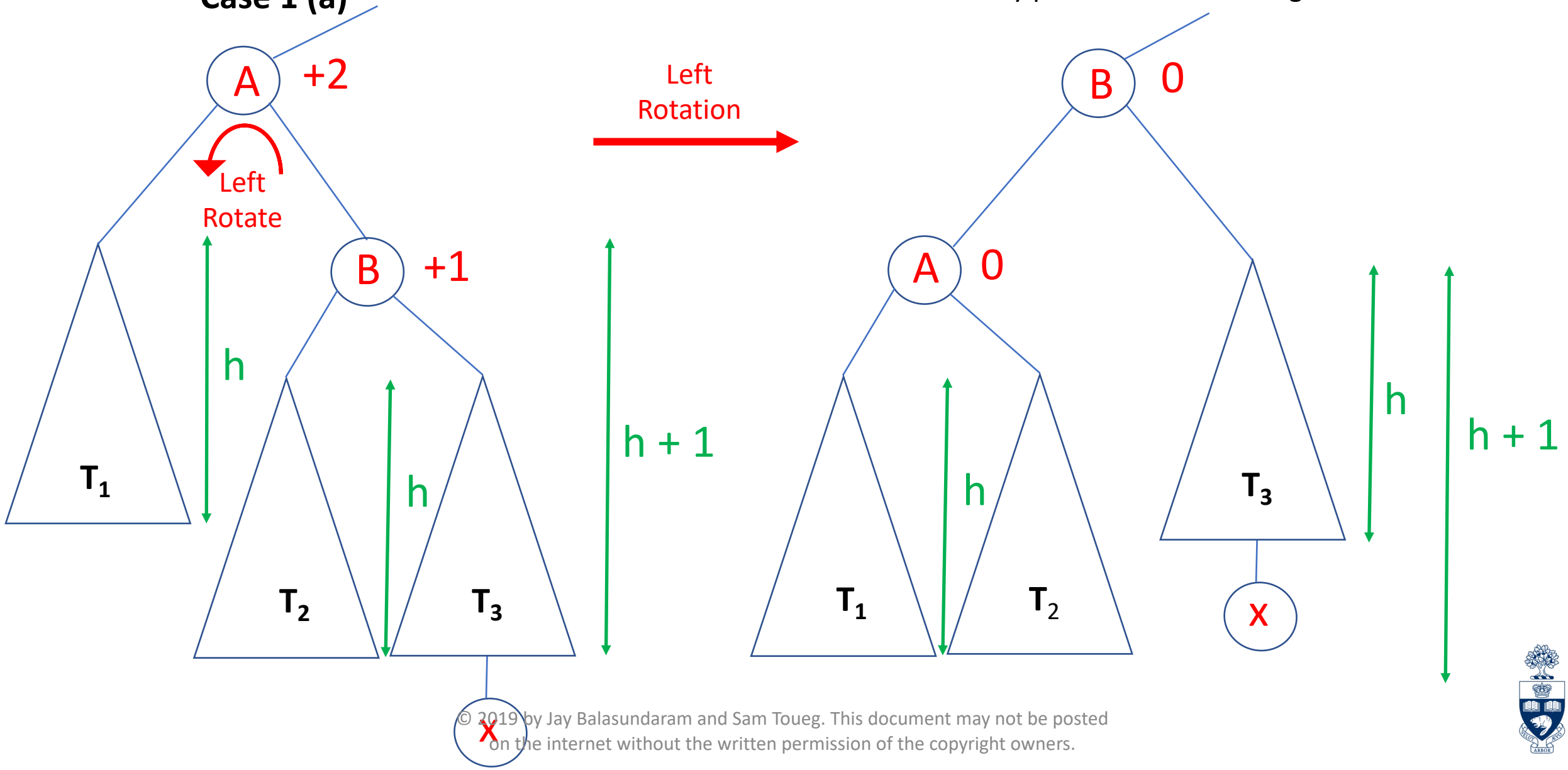
Rotation: What is the time complexity?

Case 1 (a)



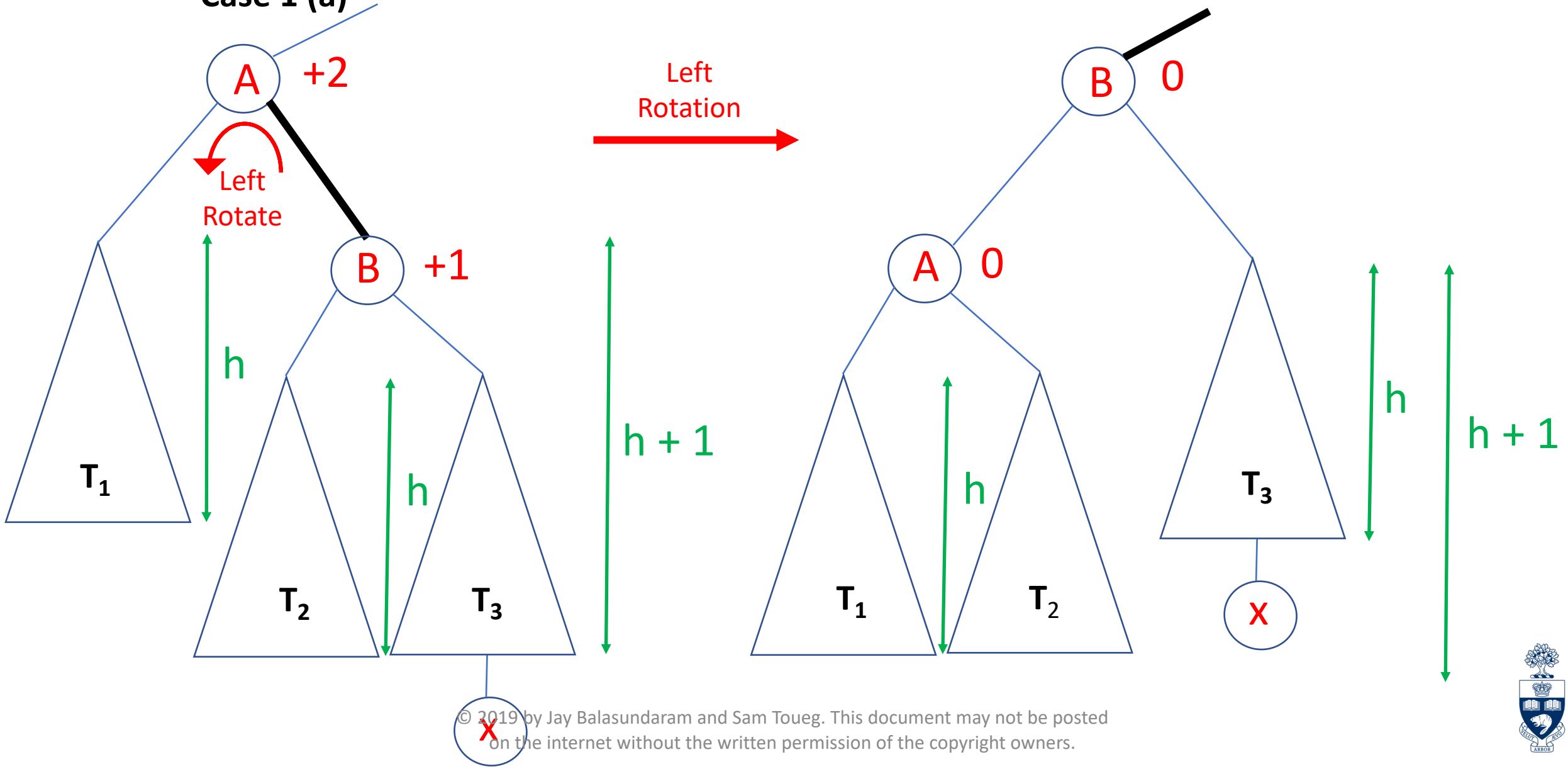
Rotation: What is the time complexity?
How many pointers did we change?

Case 1 (a)

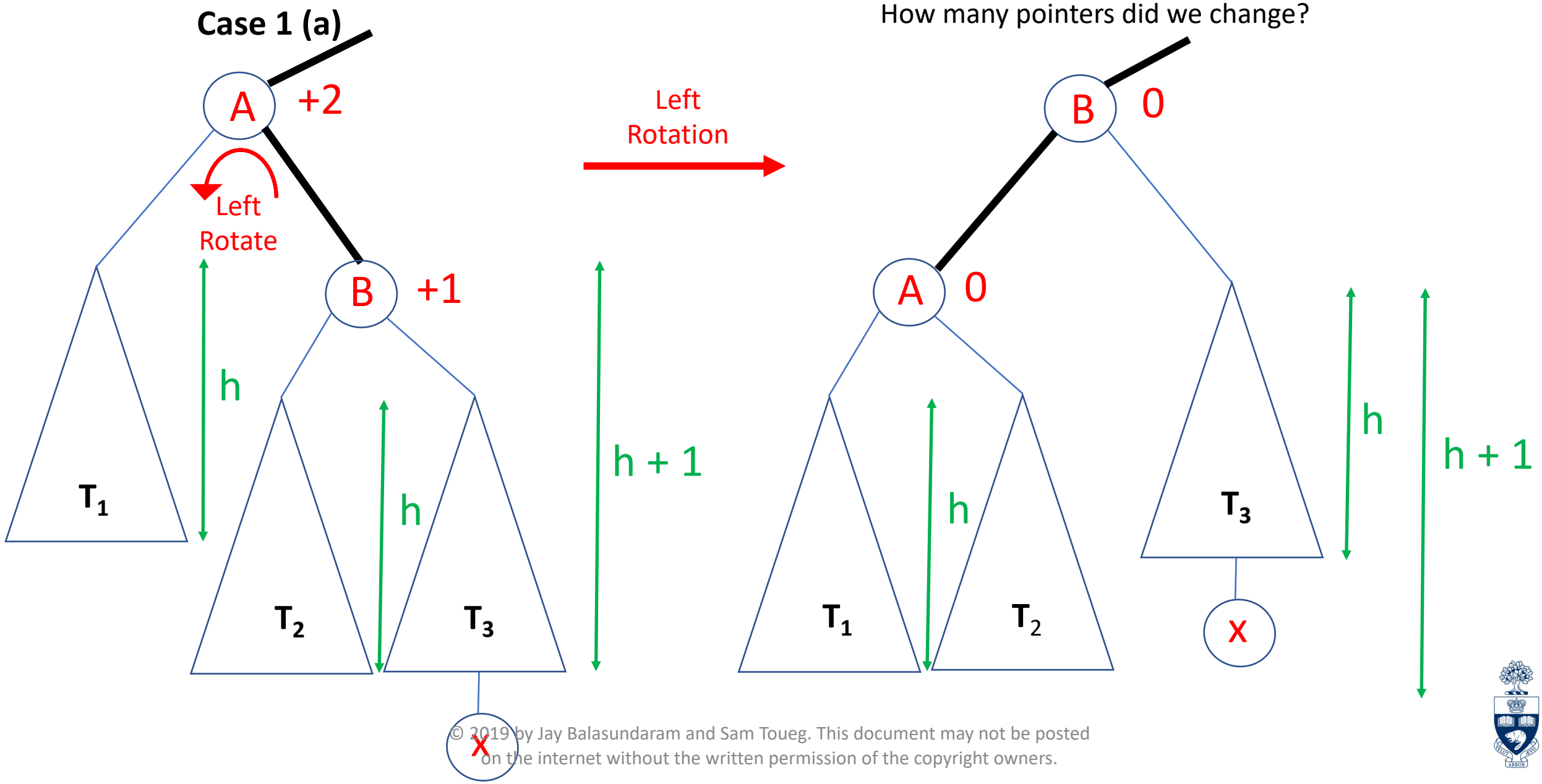


Rotation: What is the time complexity?
How many pointers did we change?

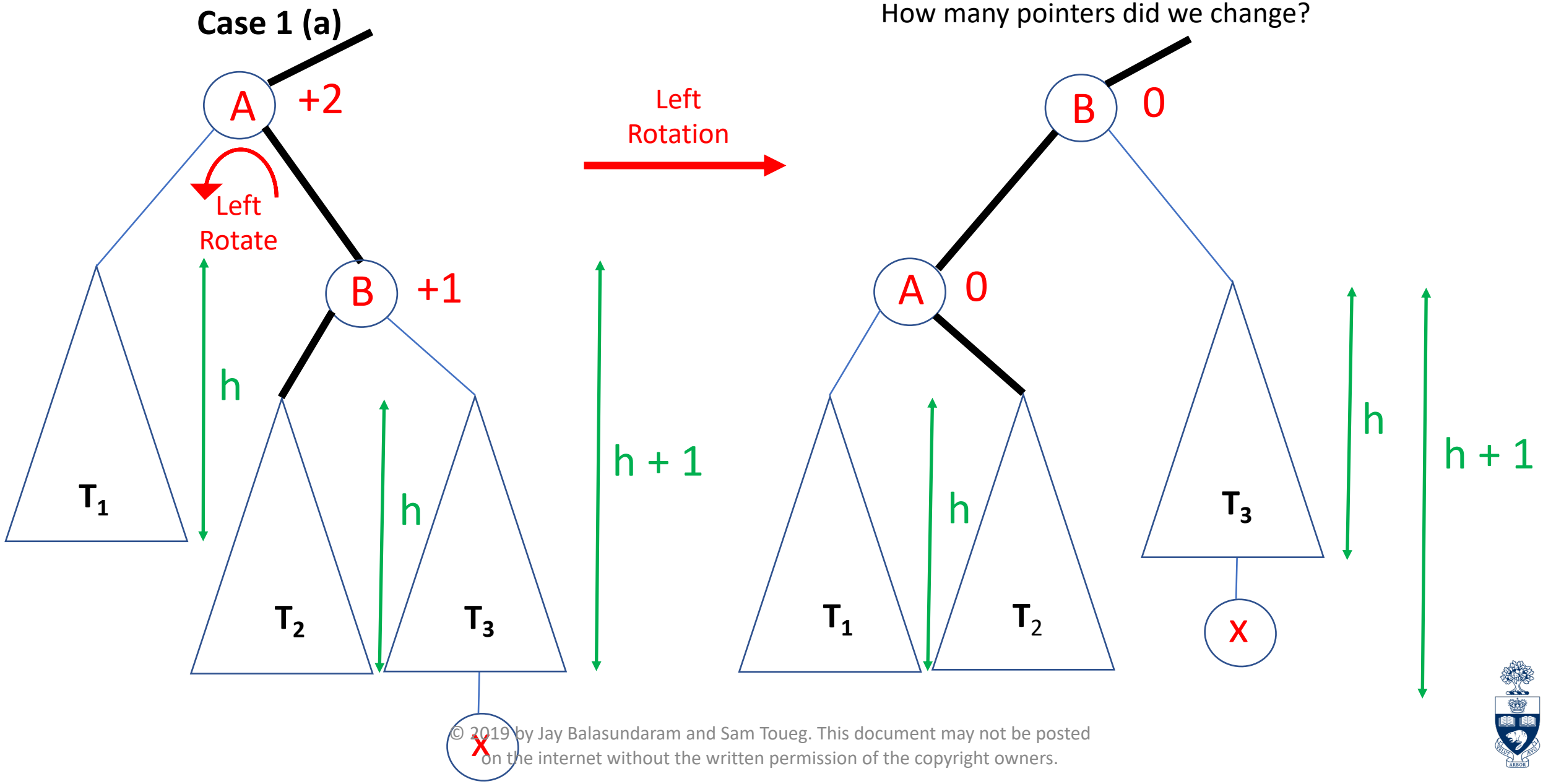
Case 1 (a)



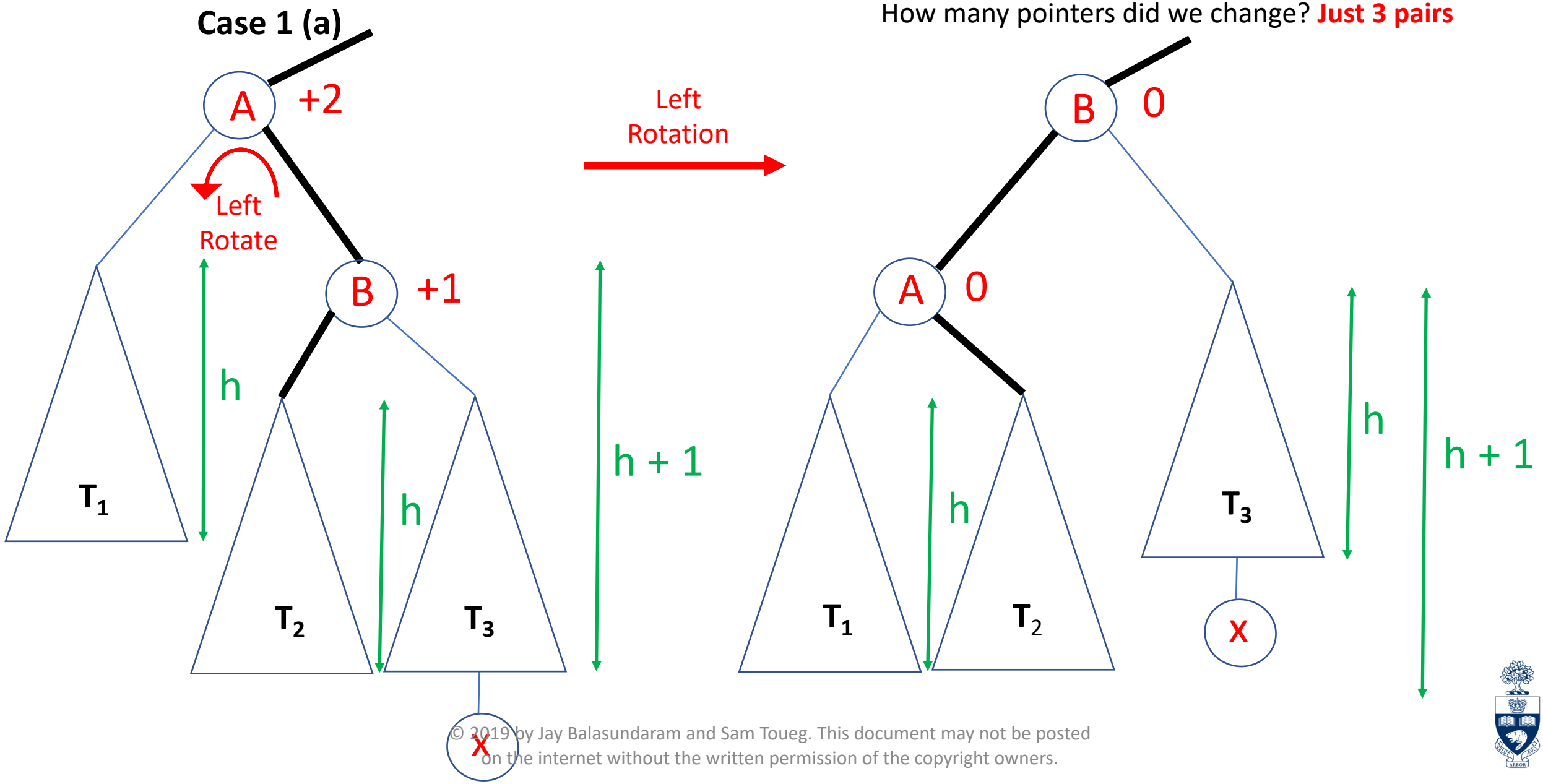
Rotation: What is the time complexity?
How many pointers did we change?



Rotation: What is the time complexity?
How many pointers did we change?

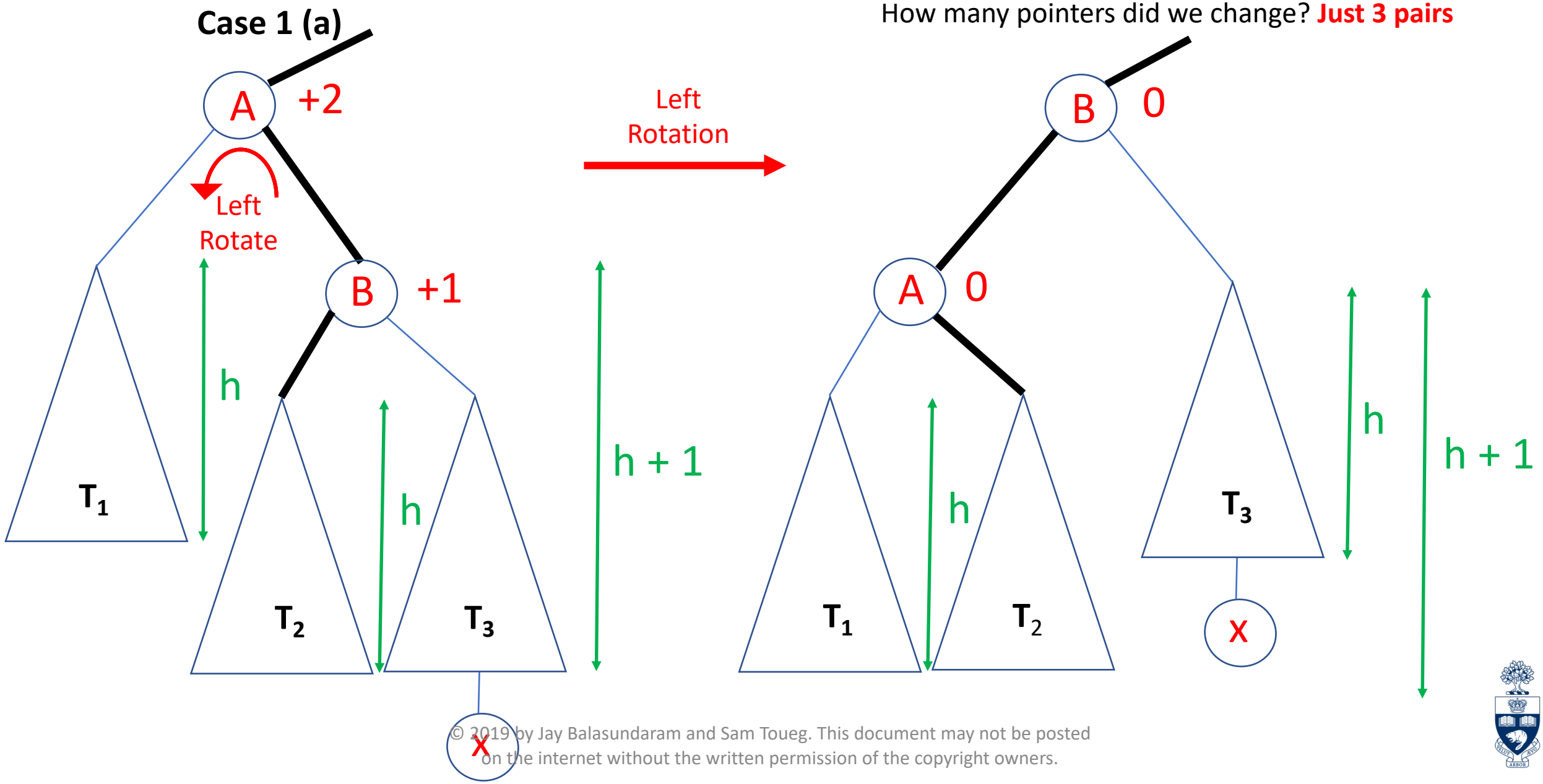


Rotation: What is the time complexity?
How many pointers did we change? **Just 3 pairs**

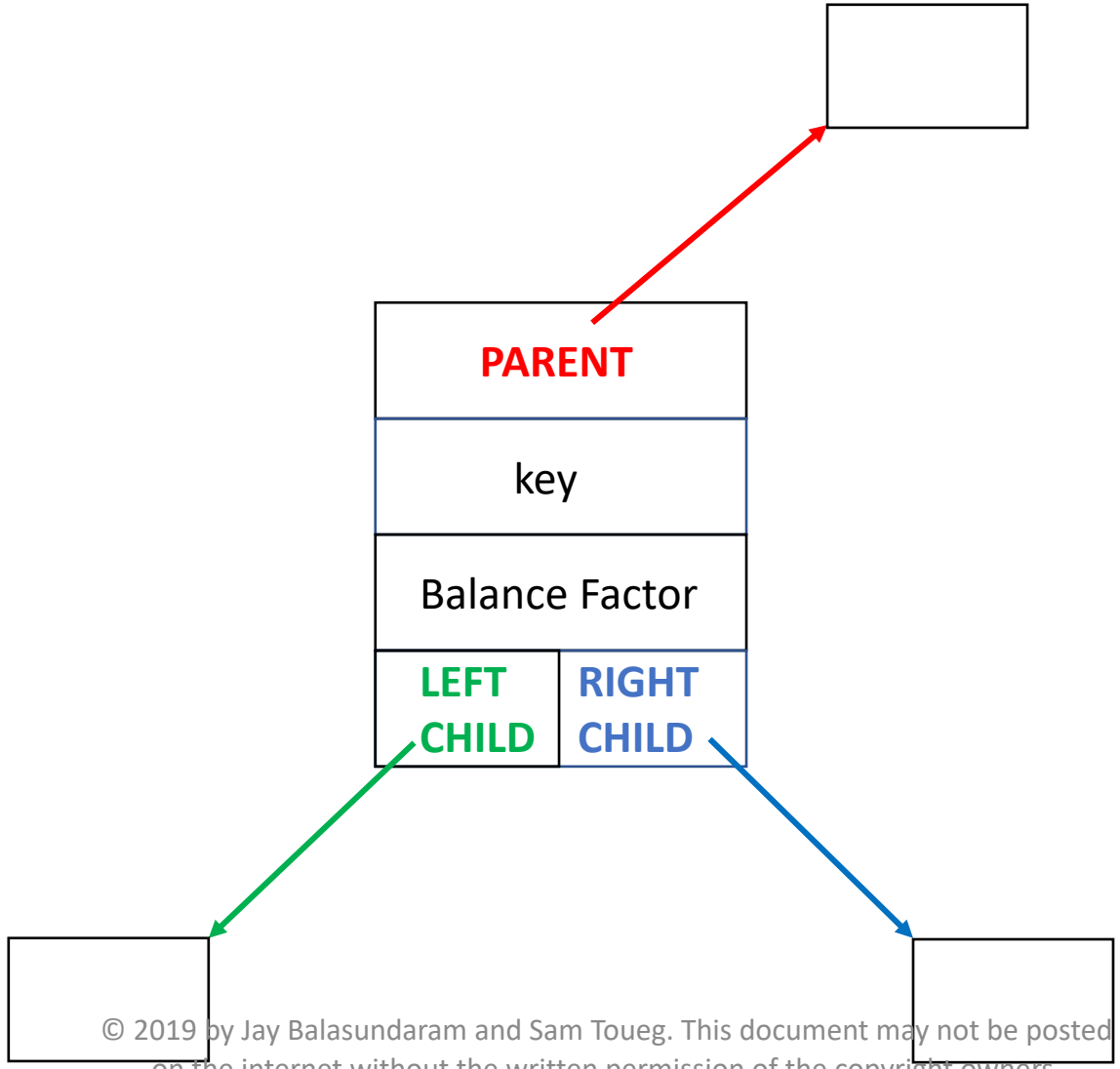


Rotation: What is the time complexity? **Θ(1)**

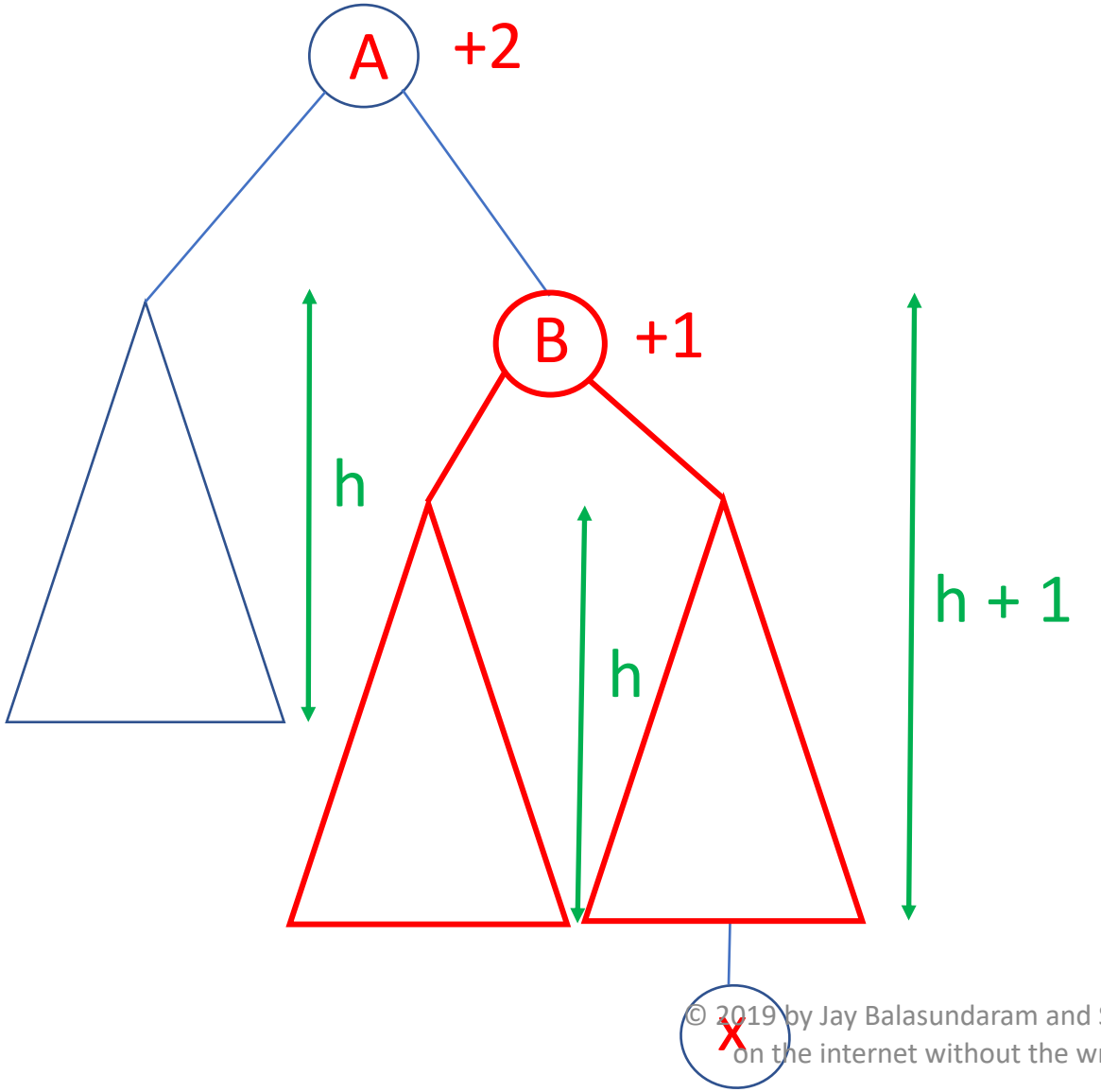
How many pointers did we change? **Just 3 pairs**



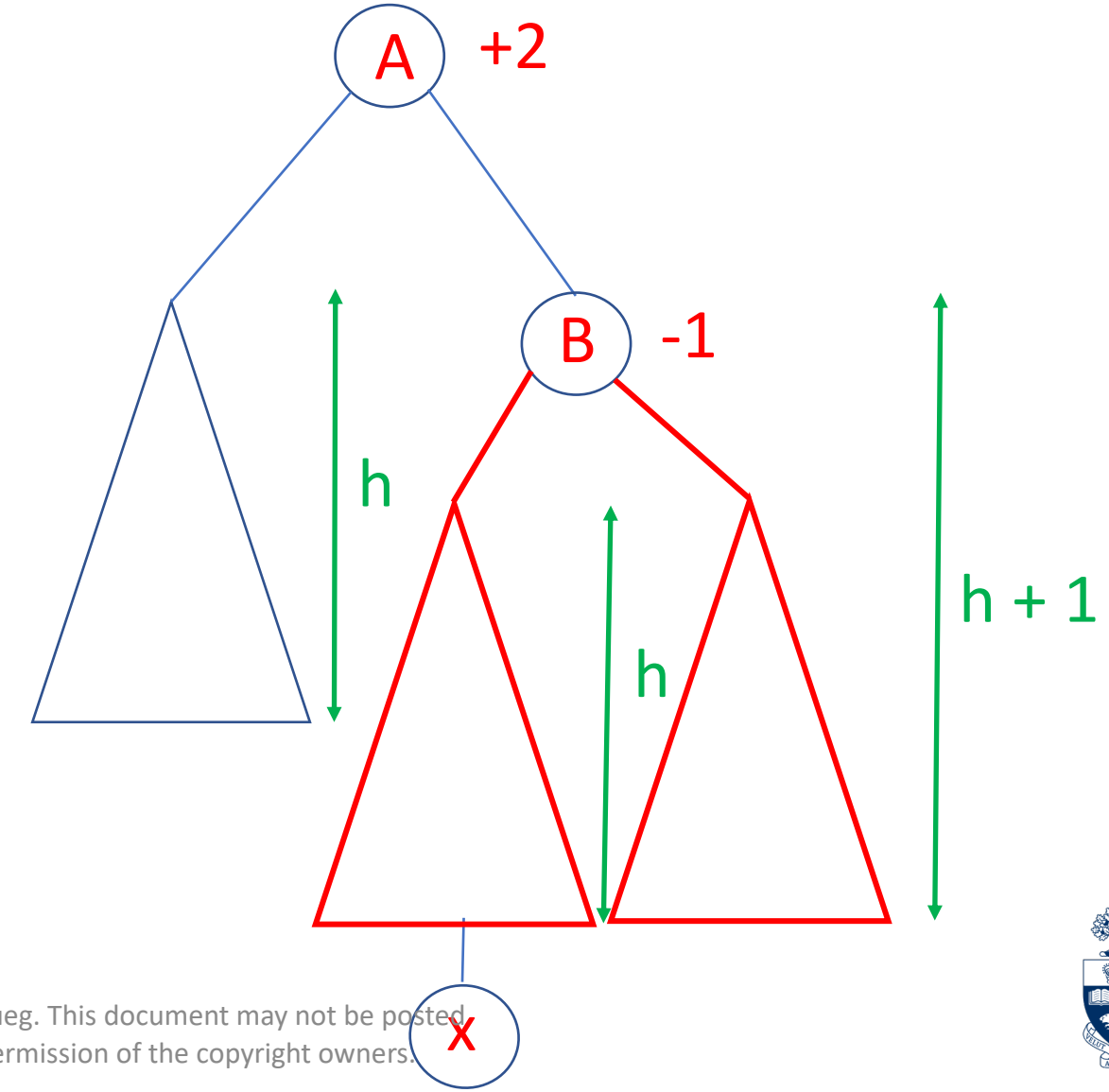
Contents of an AVL node



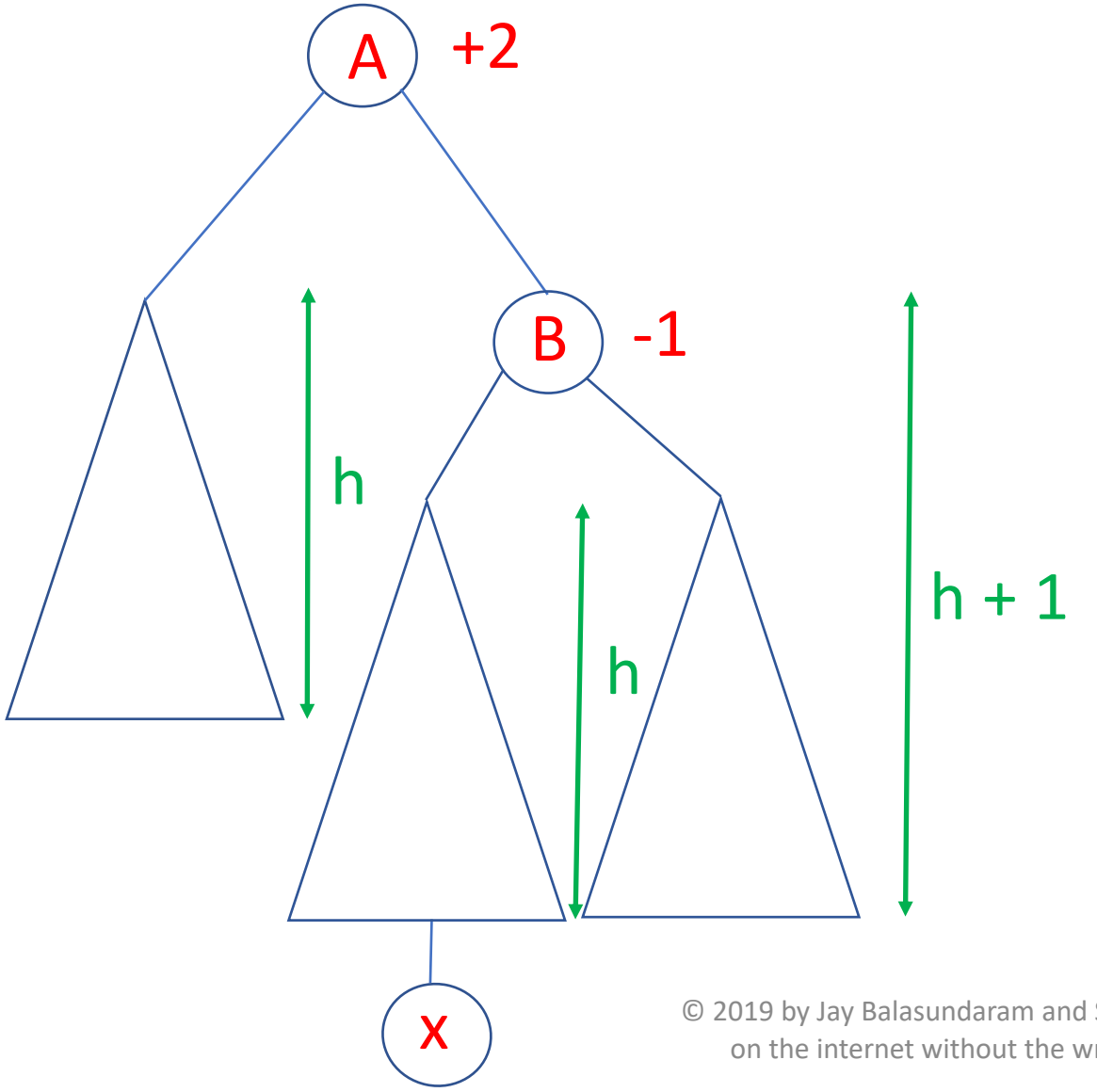
Case 1 (a) ✓



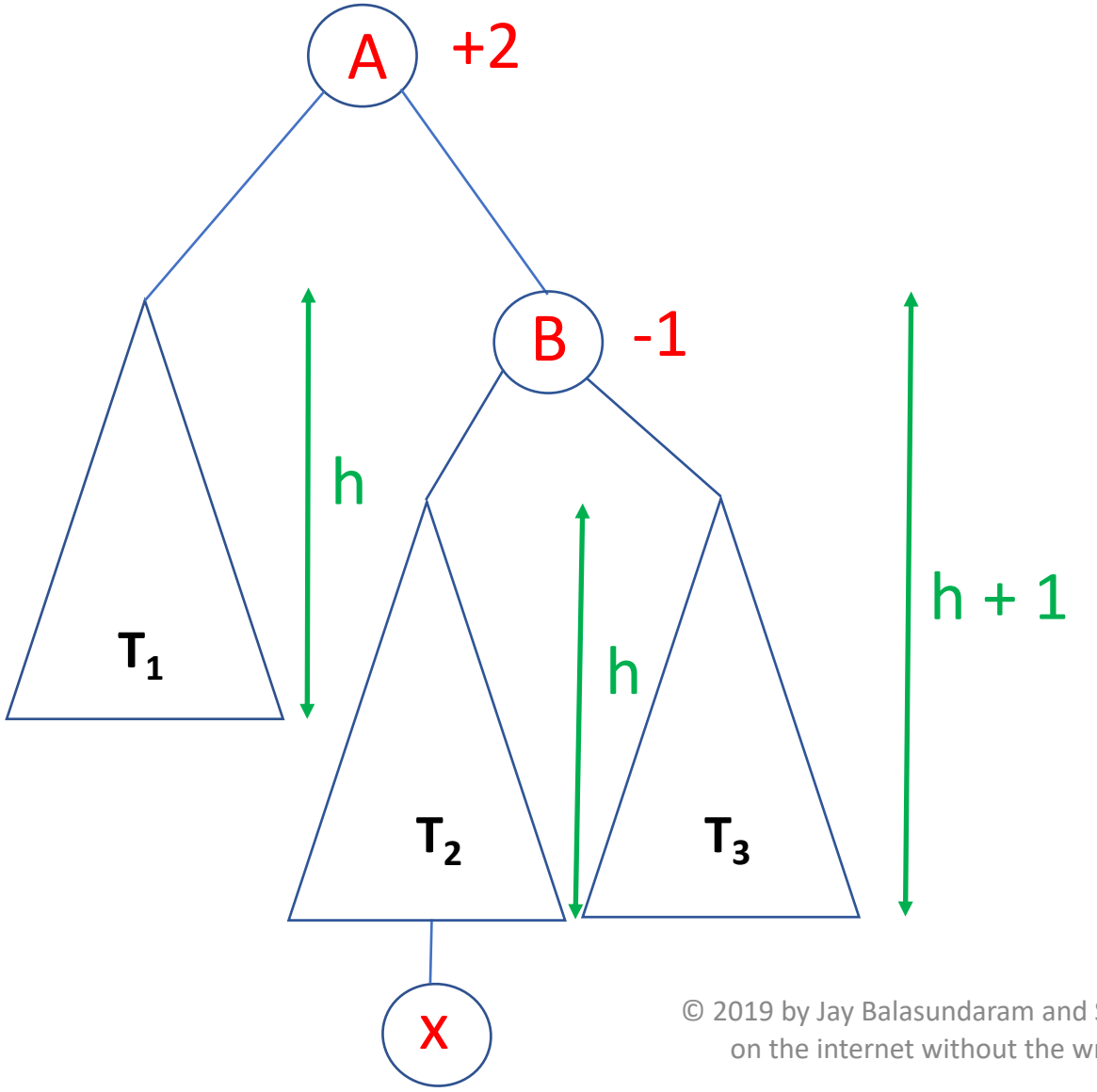
Case 1 (b)



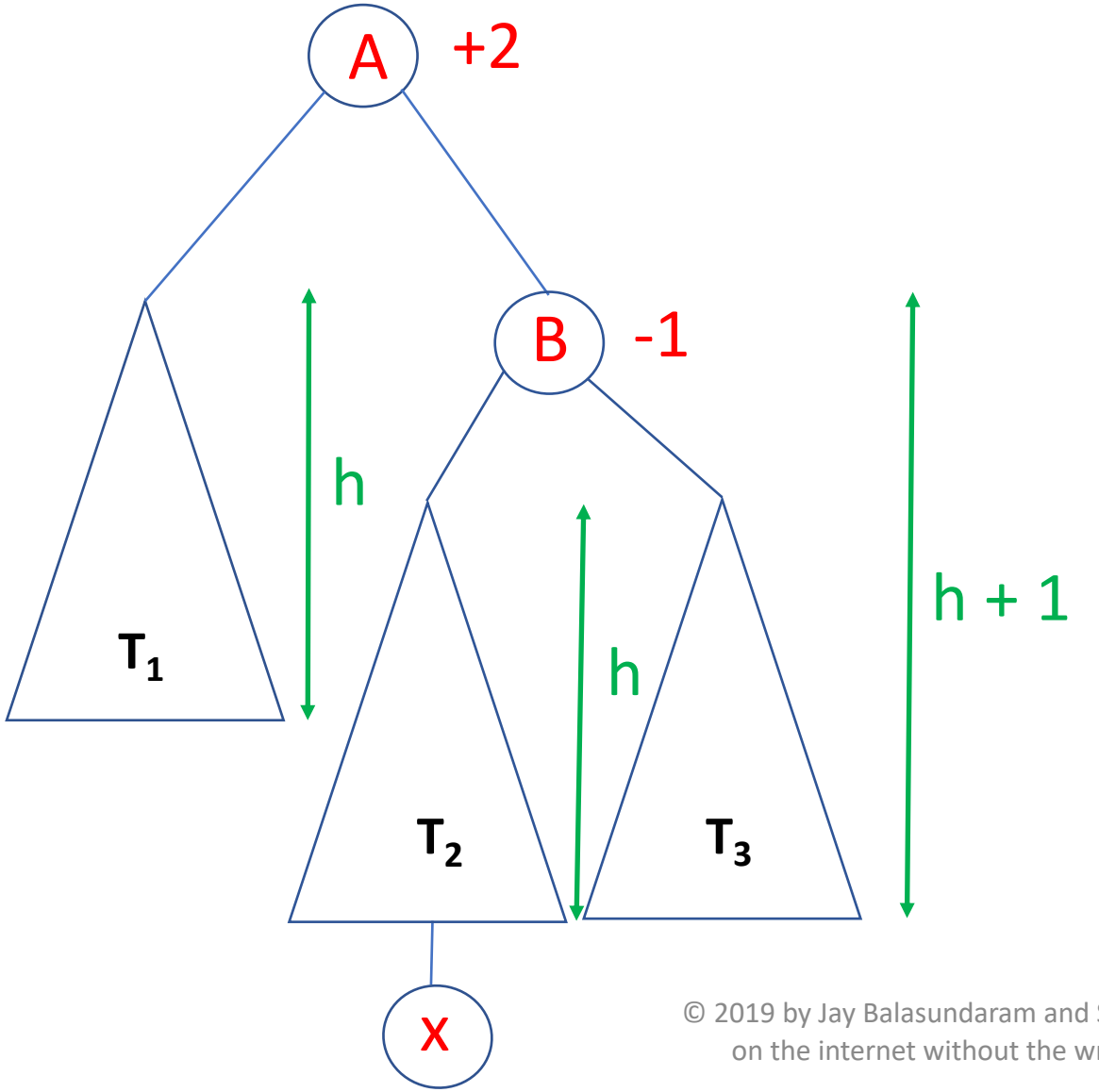
Case 1 (b)



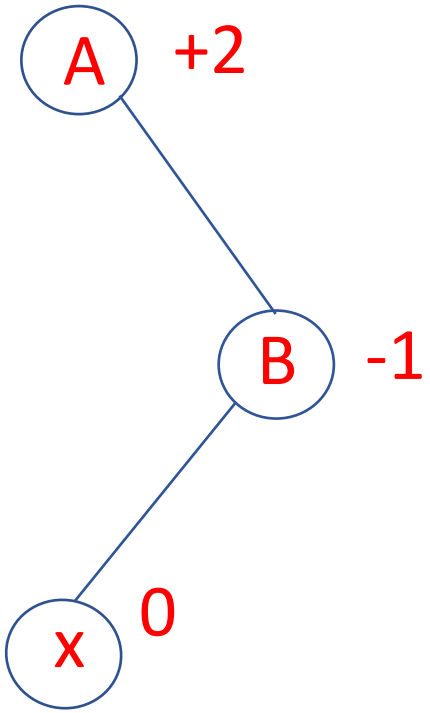
Case 1 (b)



Case 1 (b) : special case h = -1

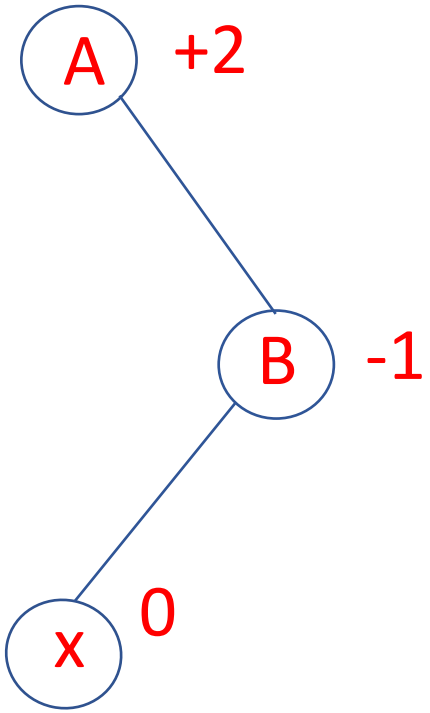


Case 1 (b) : special case h = -1



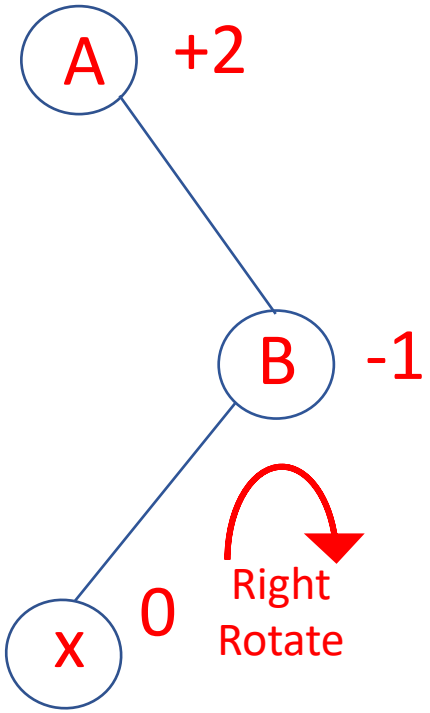
Order of keys: A x B

Case 1 (b) : special case h = -1



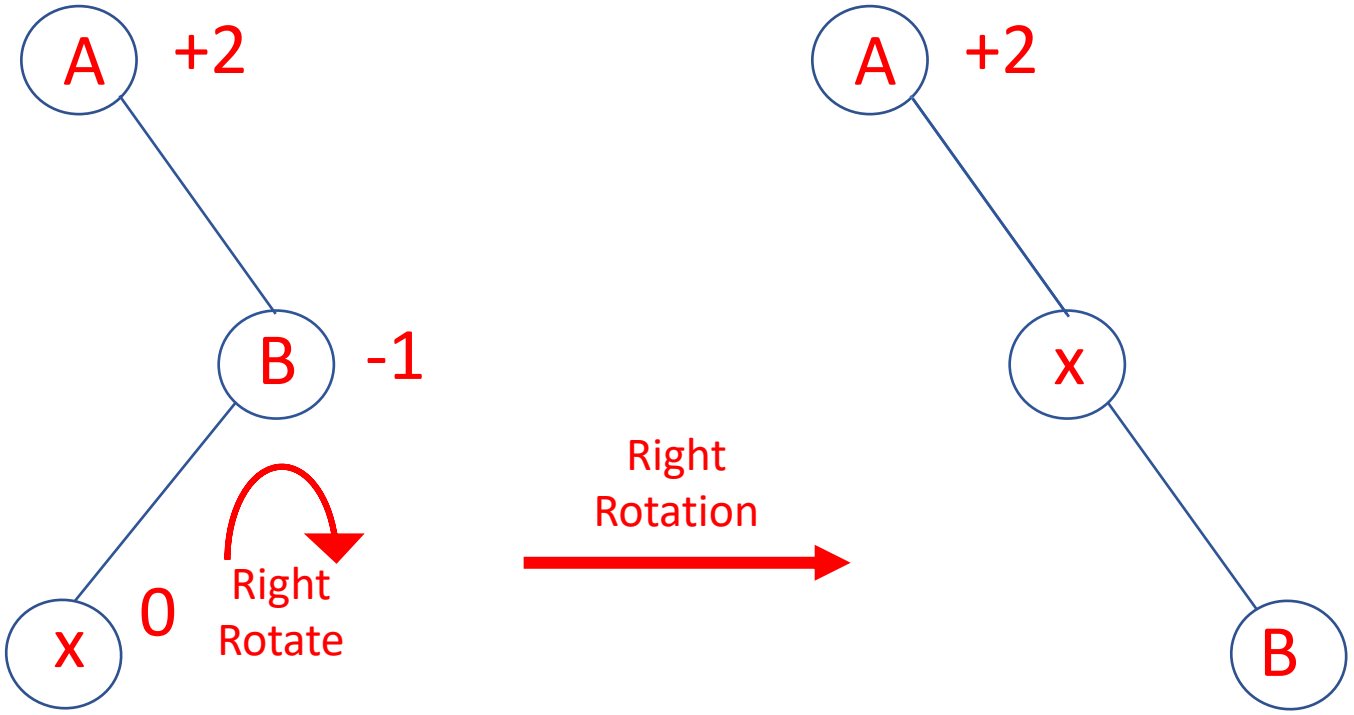
Order of keys: A x B

Case 1 (b) : special case h = -1



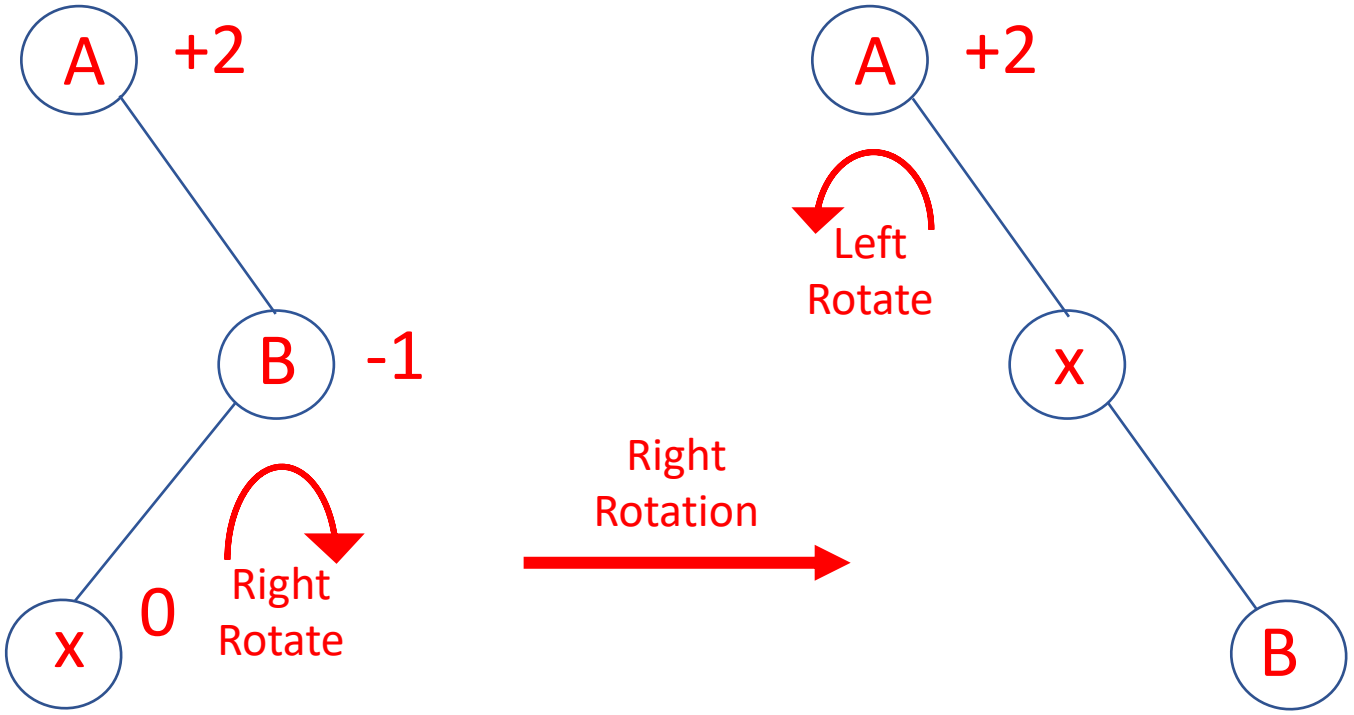
Order of keys: A x B

Case 1 (b) : special case h = -1



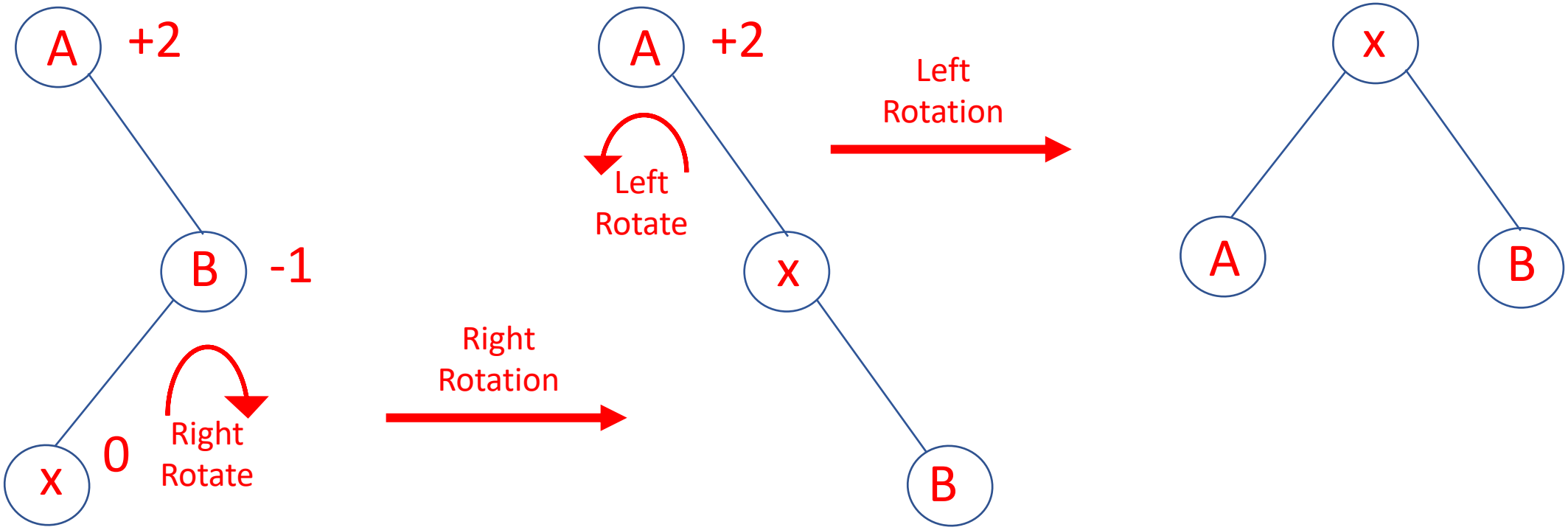
Order of keys: A x B

Case 1 (b) : special case h = -1



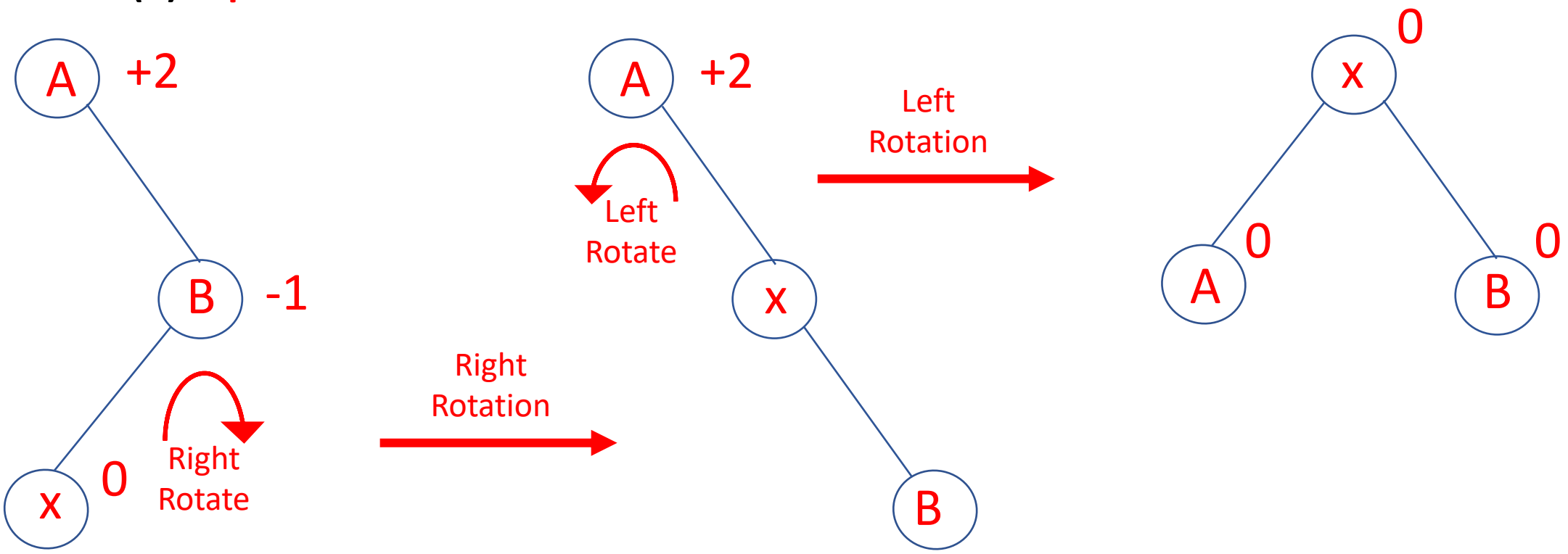
Order of keys: A x B

Case 1 (b) : special case h = -1



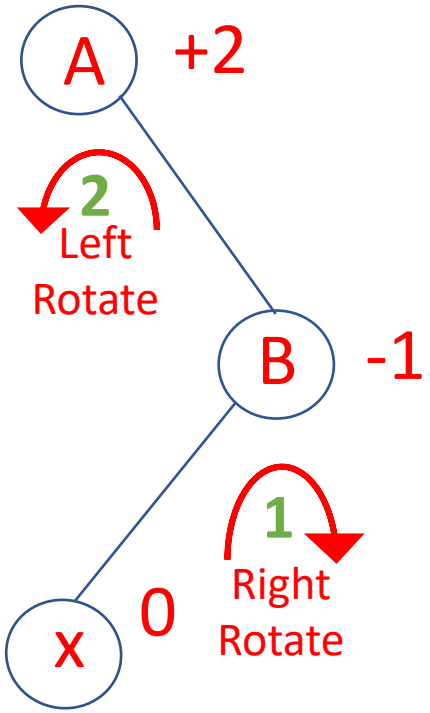
Order of keys: A x B

Case 1 (b) : special case h = -1

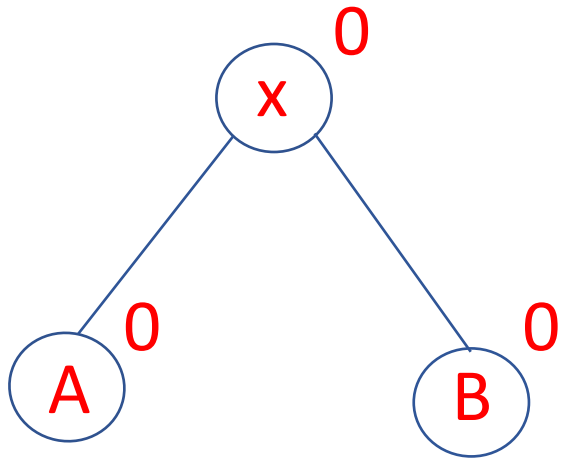


Order of keys: A x B

Case 1 (b) : special case h = -1

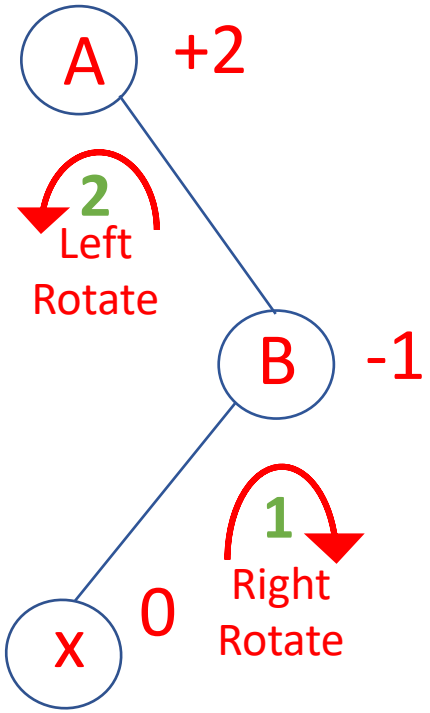


Double
Right-Left
Rotation

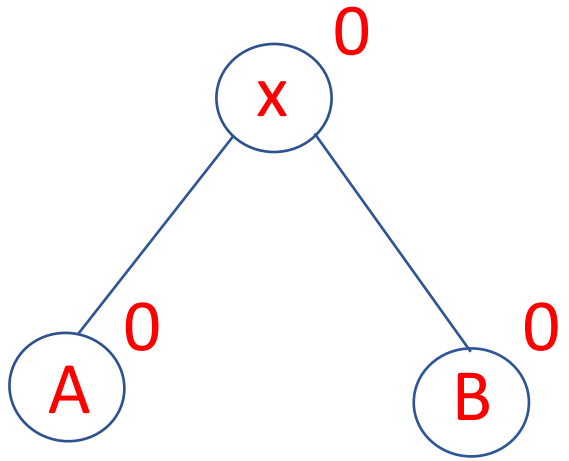


Rotation: (1) Rebalances the subtree

Case 1 (b) : special case h = -1

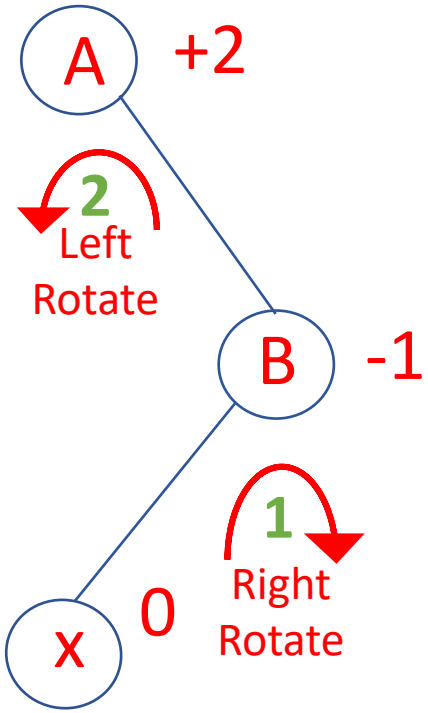


Double
Right-Left
Rotation

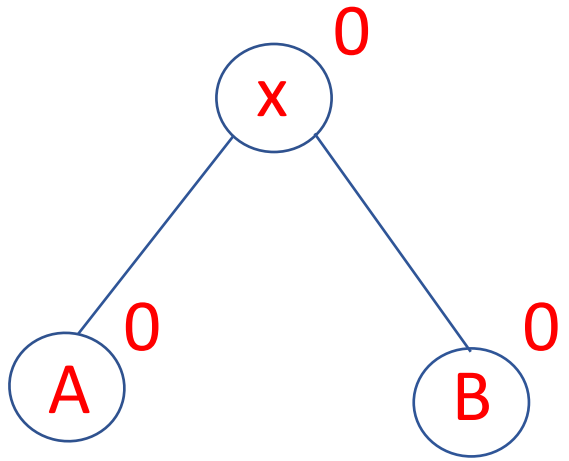


Rotation: (2) Preserves BST property (order: A x B)

Case 1 (b) : special case h = -1

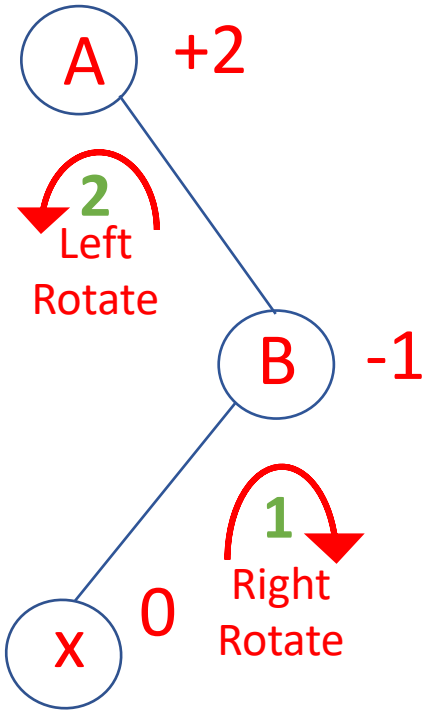


Double
Right-Left
Rotation

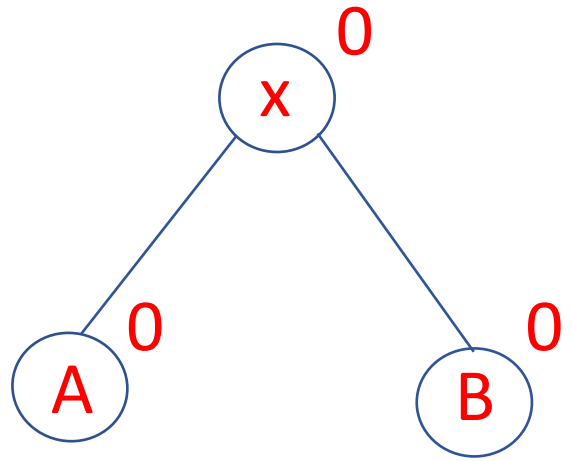


Rotation: (3) Bonus?

Case 1 (b) : special case h = -1

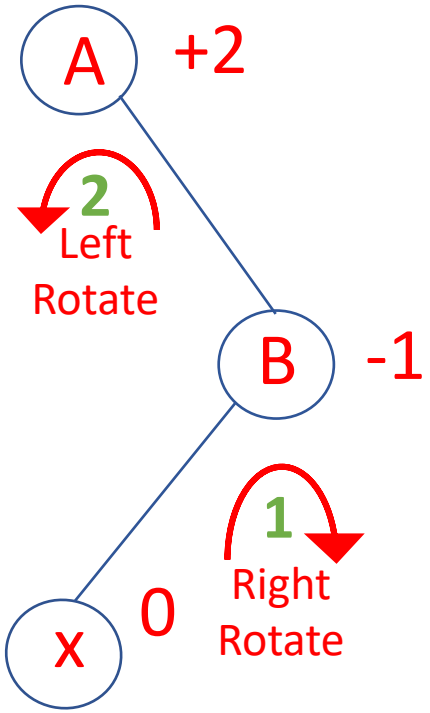


Double
Right-Left
Rotation

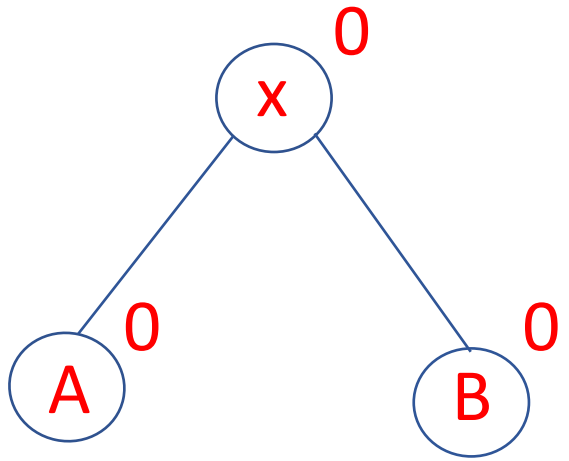


Rotation: (3) Bonus? YES!

Case 1 (b) : special case h = -1

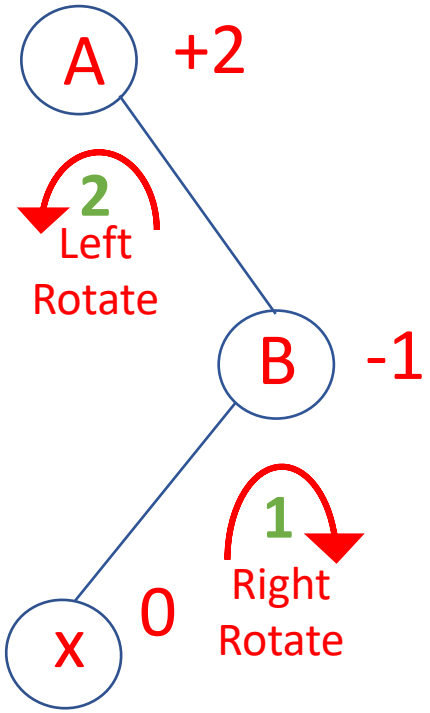


Double
Right-Left
Rotation

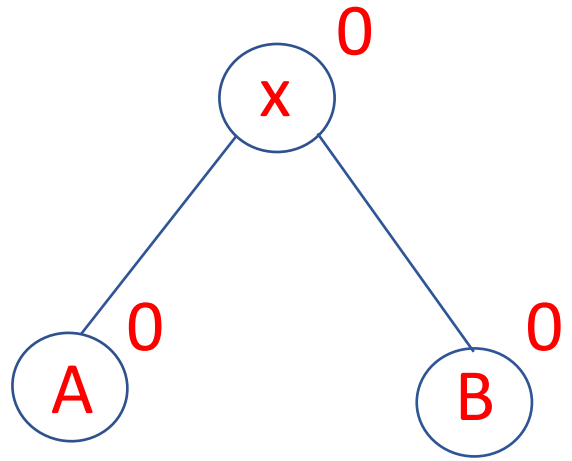


Case 1 (b) : special case h = -1

Rotation: (3) Bonus: height of this subtree is the same both before and after the insertion of x

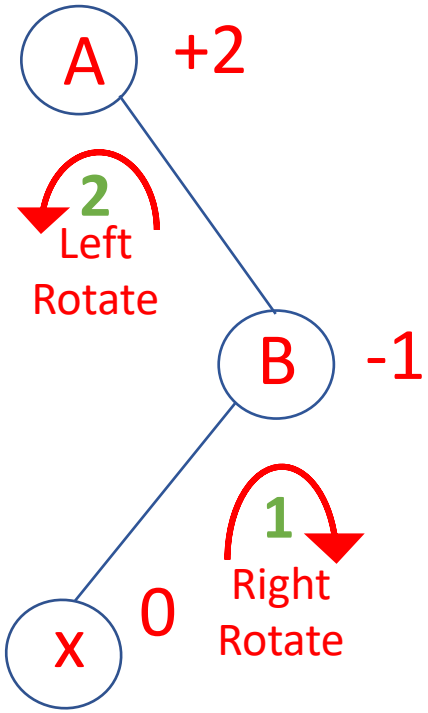


Double
Right-Left
Rotation

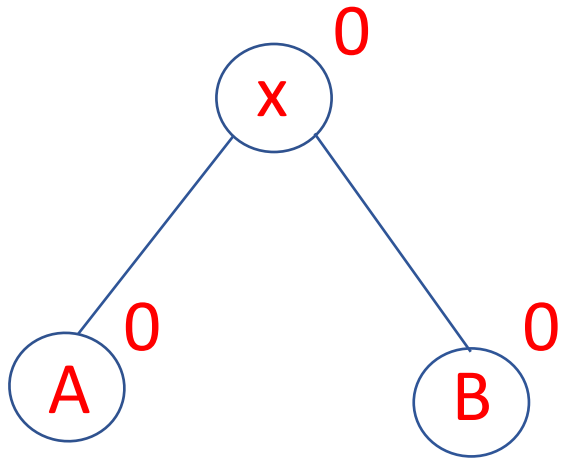


Rotation: (3) Bonus ⟹ insertion algorithm is done

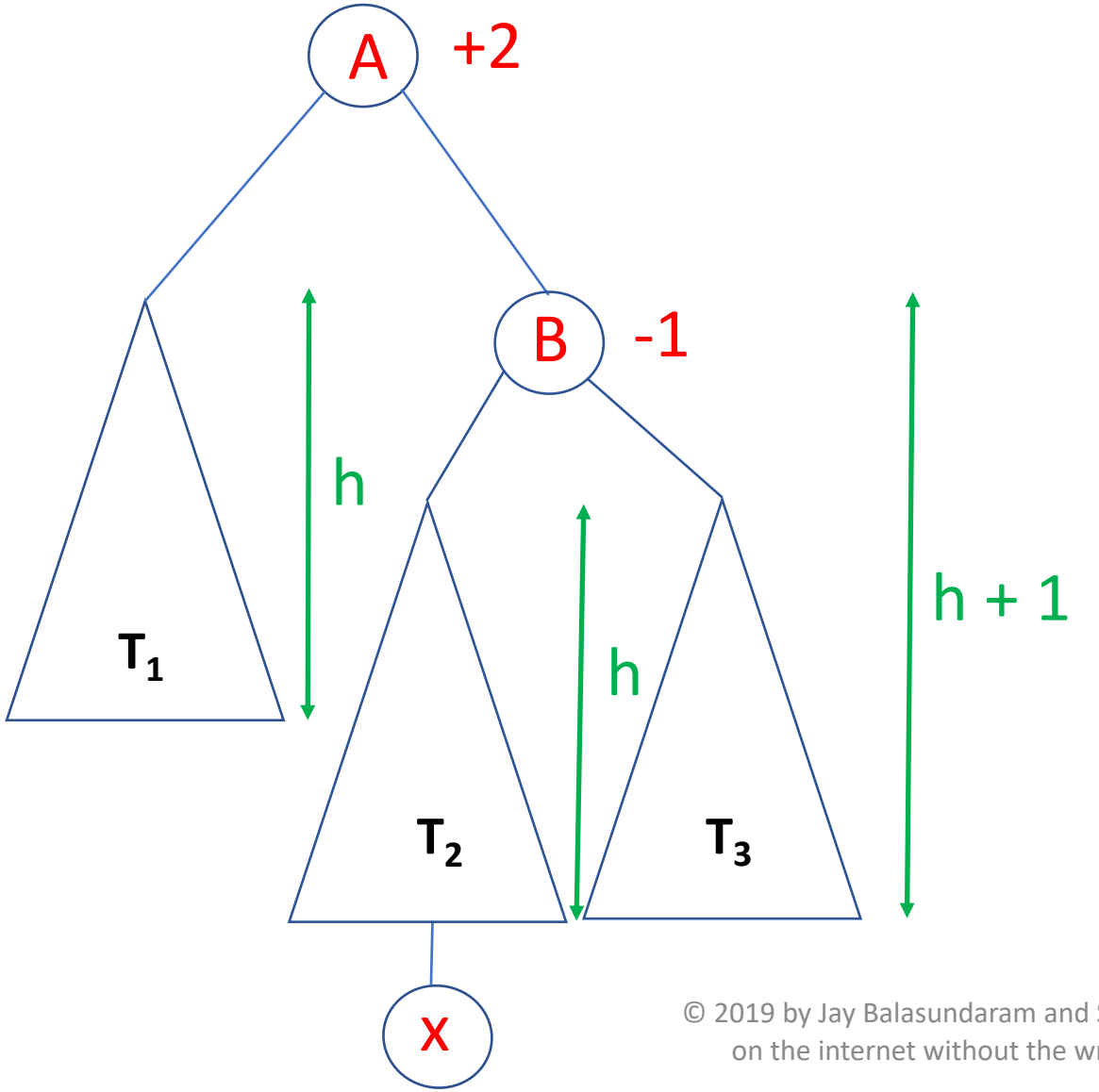
Case 1 (b) : special case h = -1



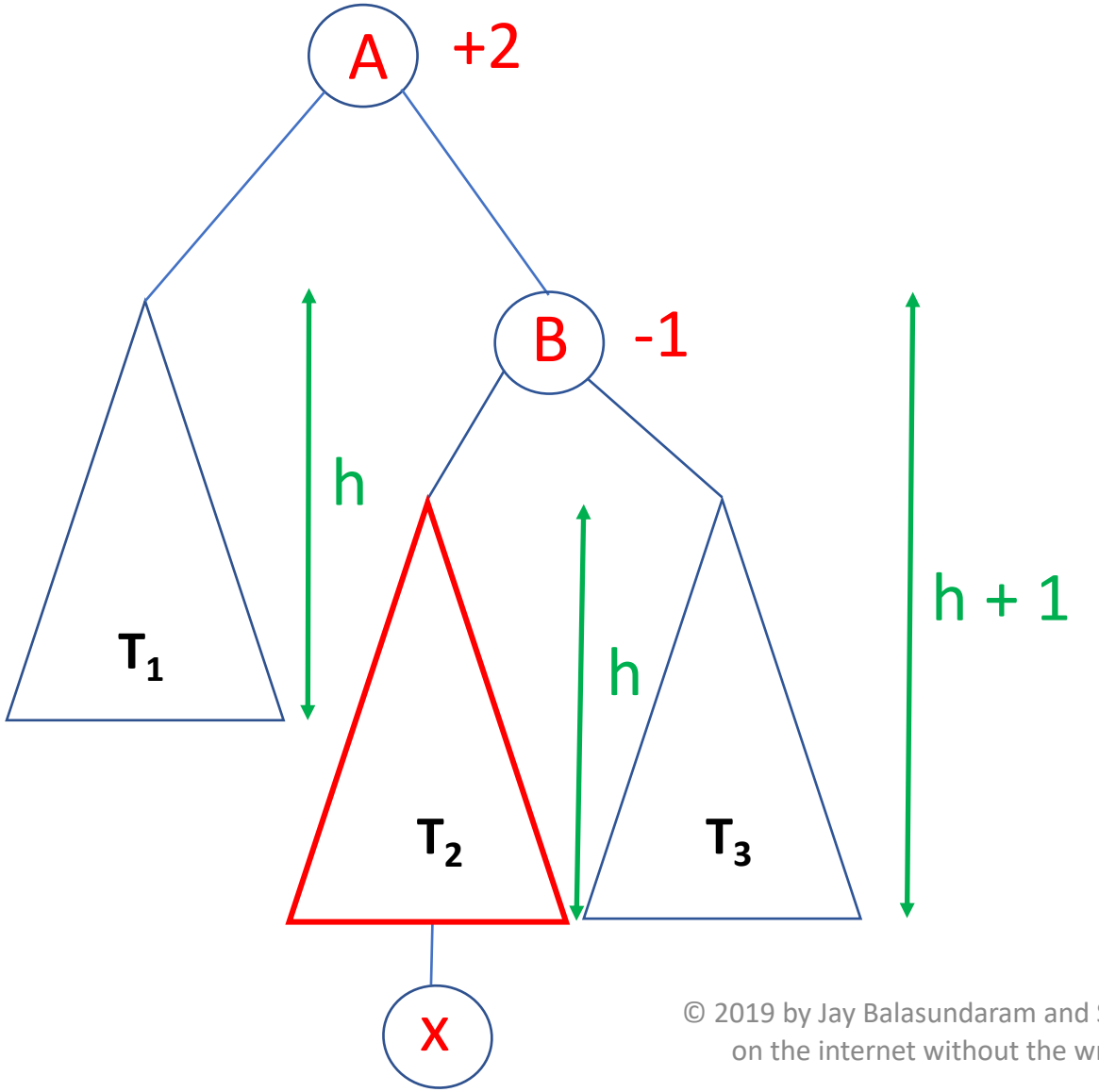
Double
Right-Left
Rotation



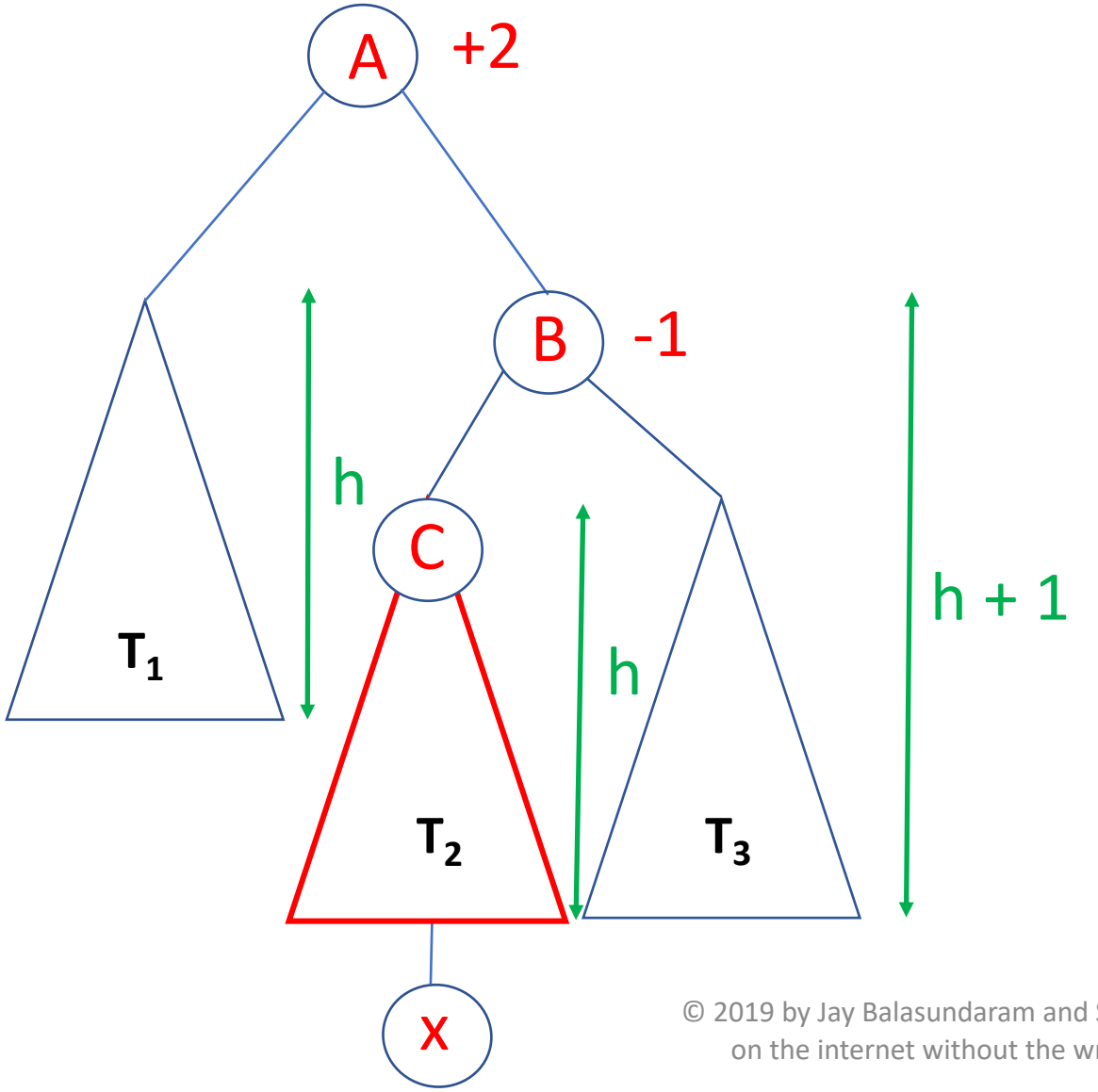
Case 1 (b) : h > -1



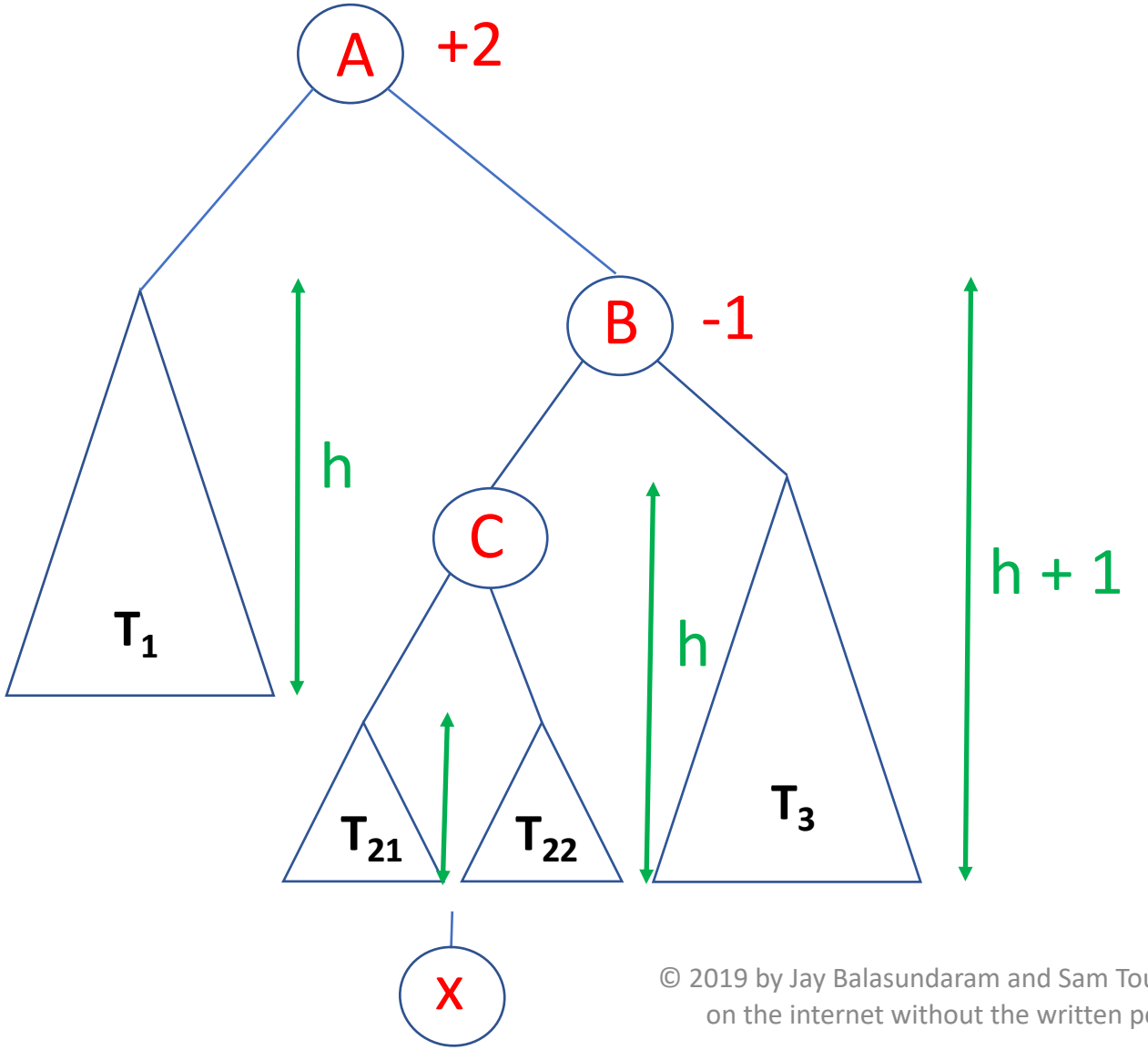
Case 1 (b) : h > -1



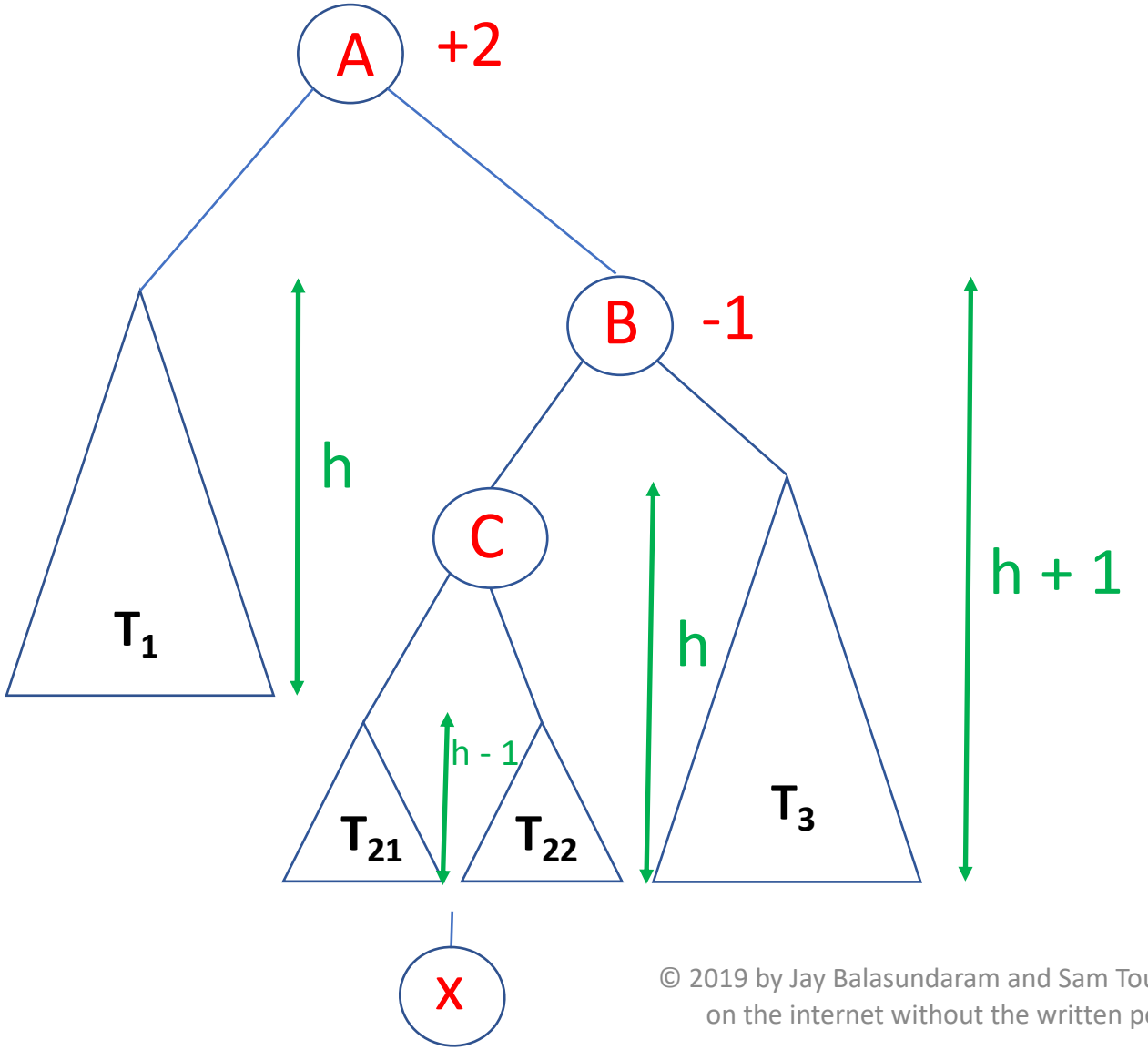
Case 1 (b) : h > -1



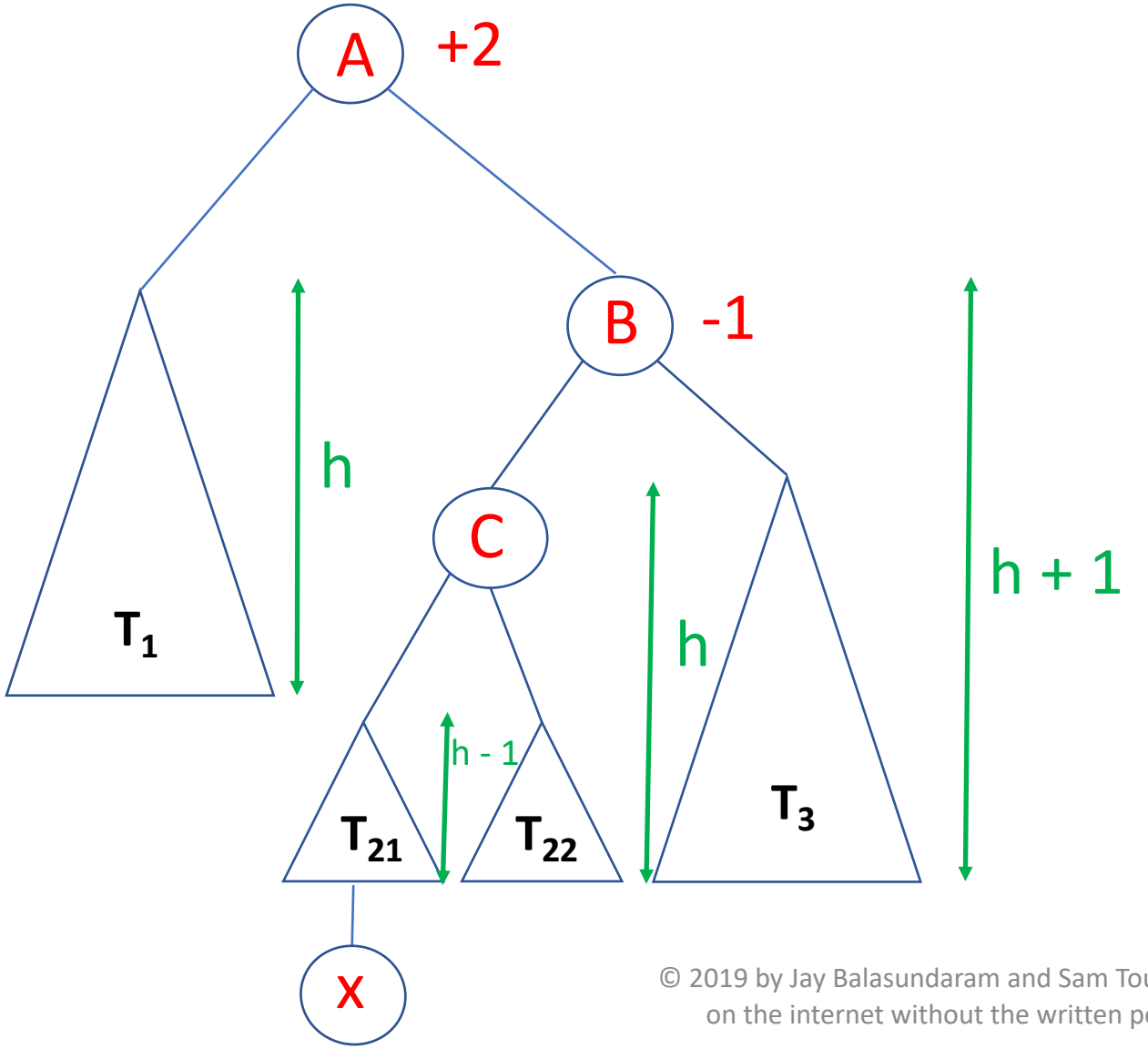
Case 1 (b) : h > -1



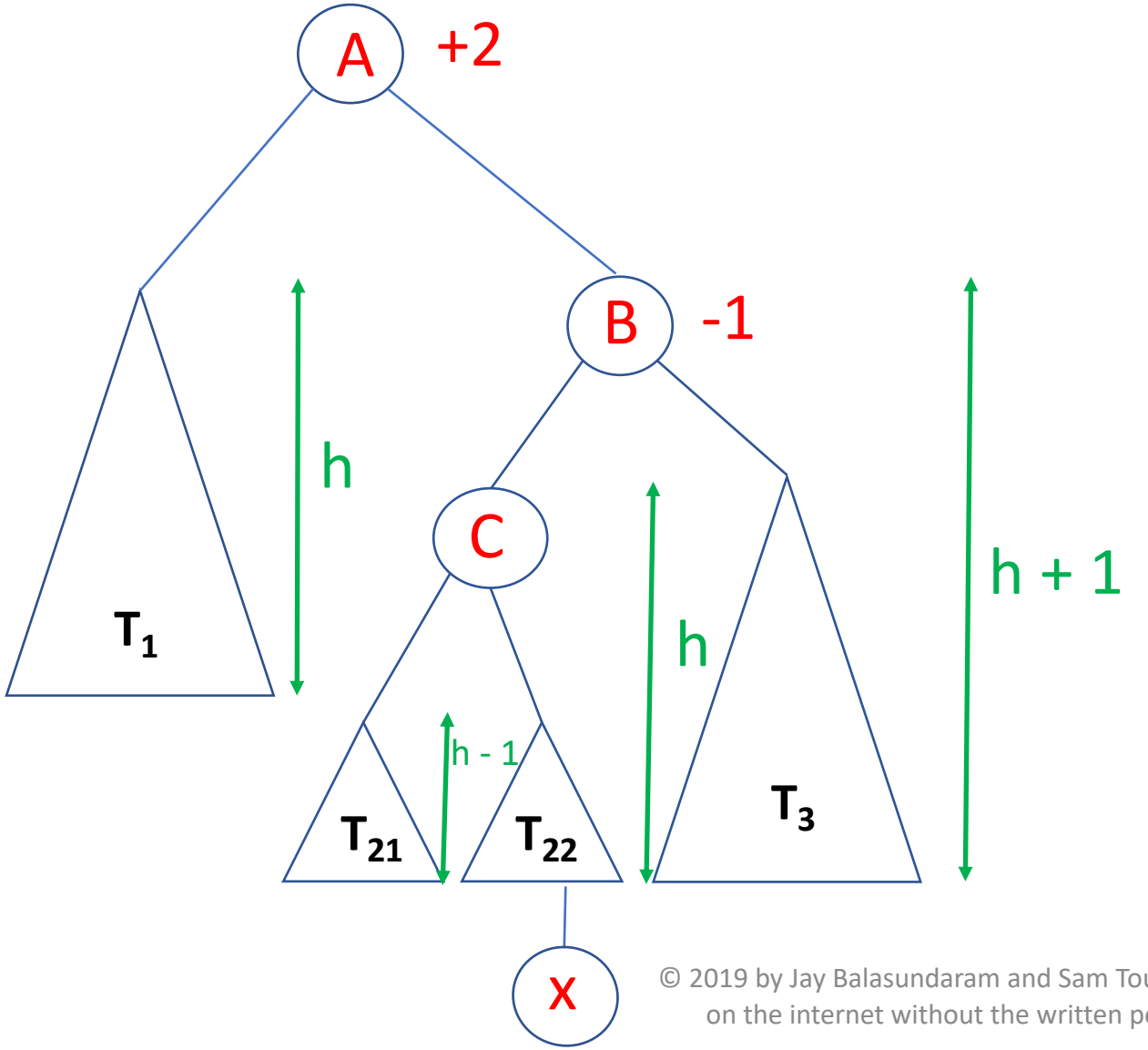
Case 1 (b) : h > -1



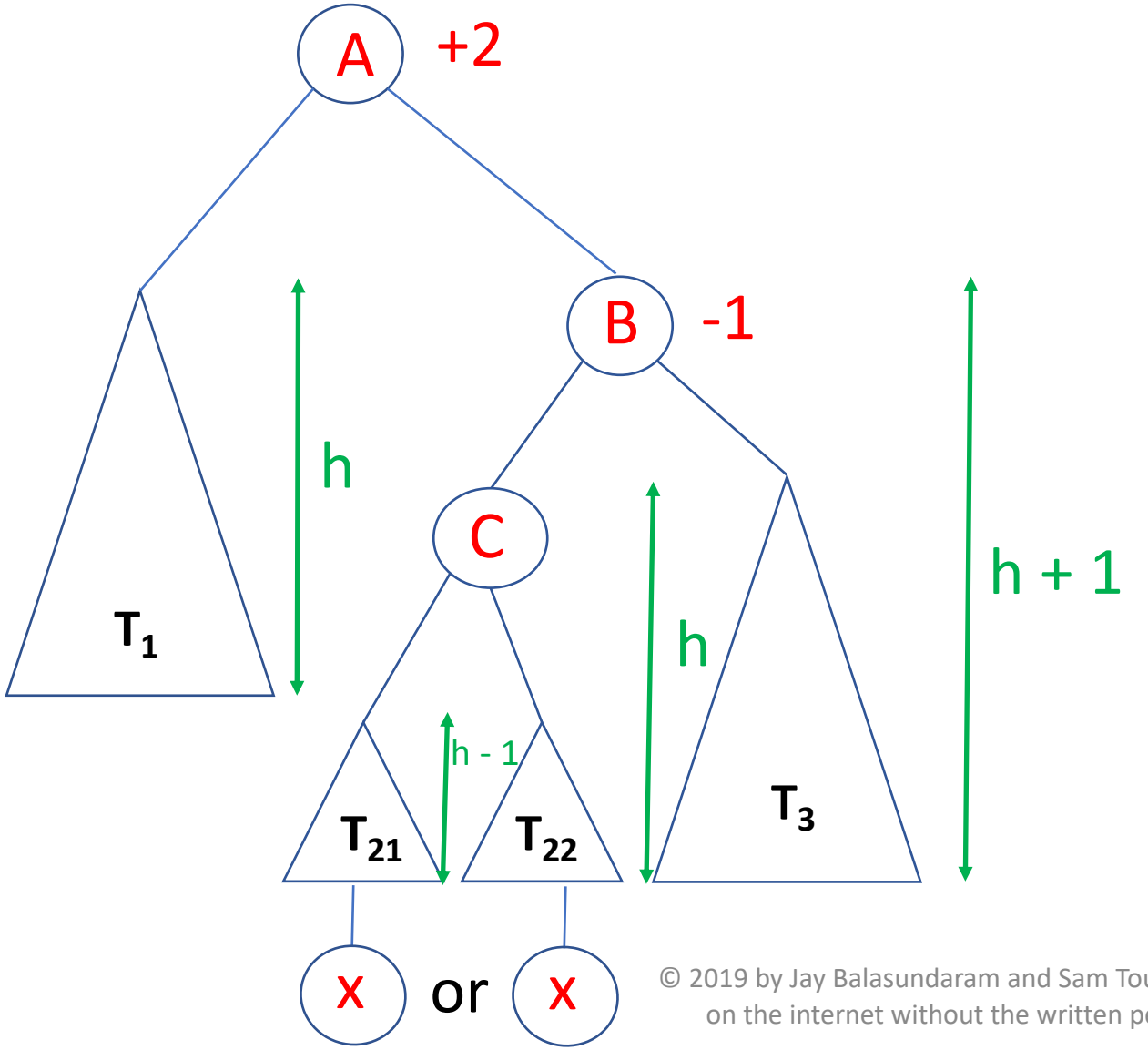
Case 1 (b) : h > -1



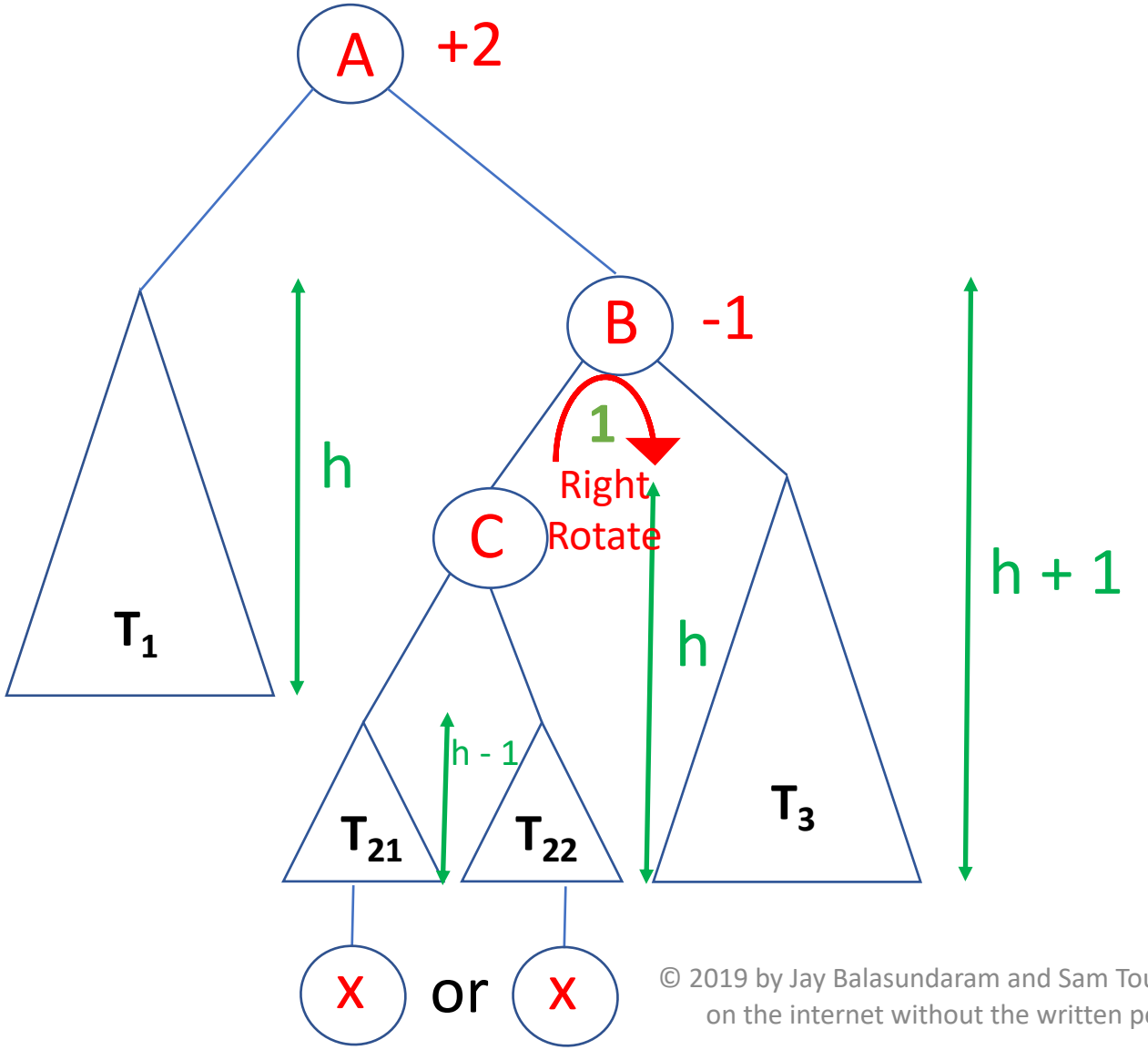
Case 1 (b) : h > -1



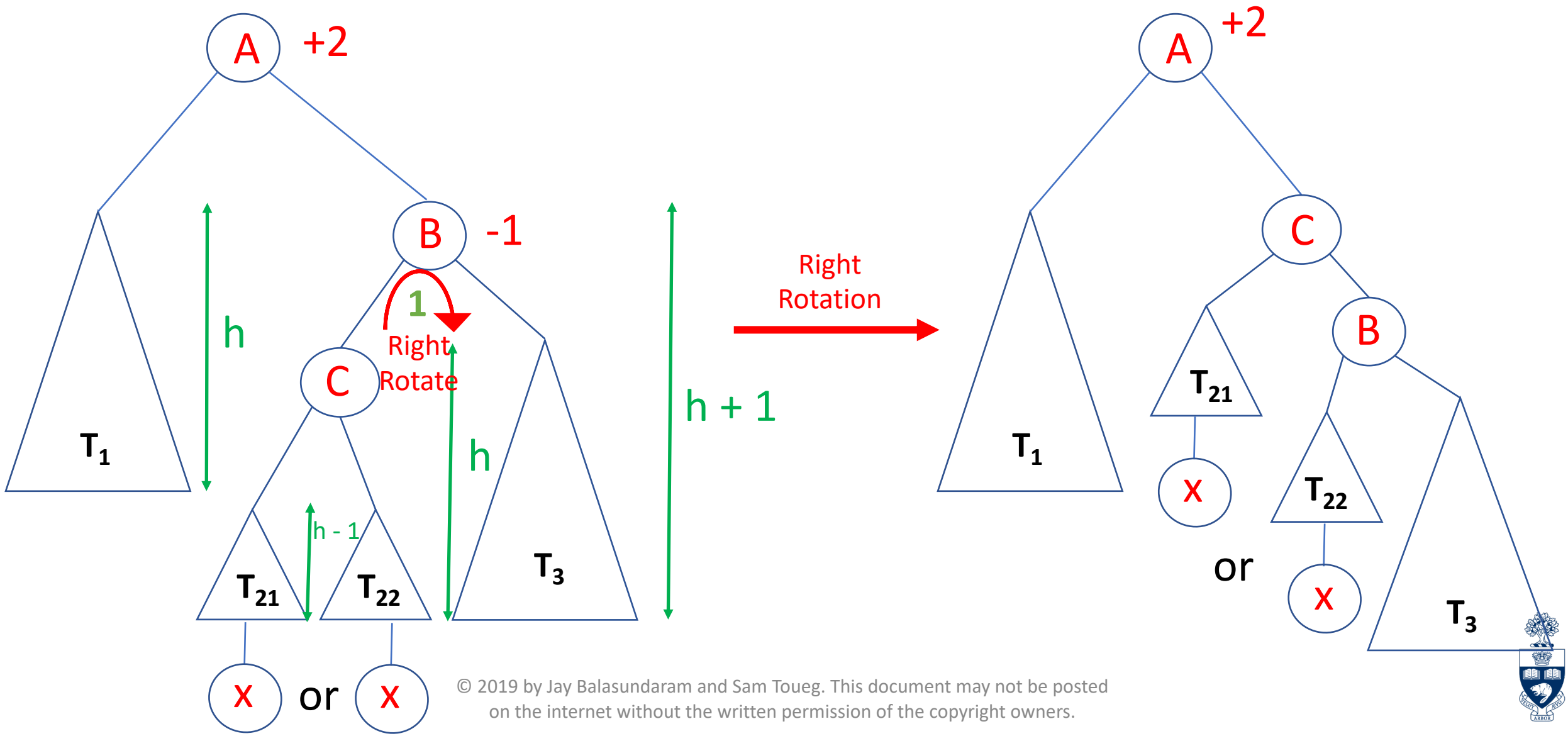
Case 1 (b) : h > -1



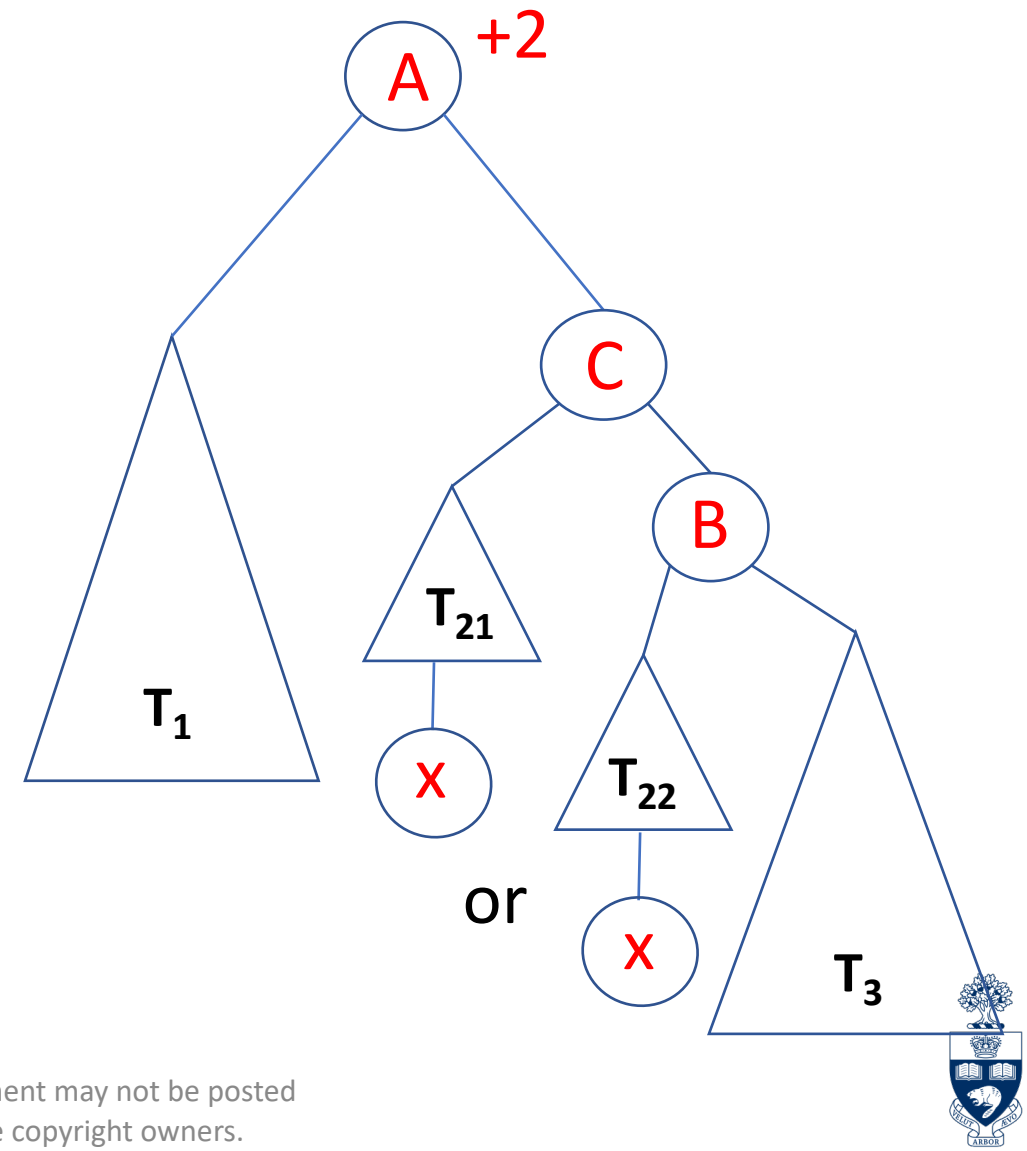
Case 1 (b) : h > -1



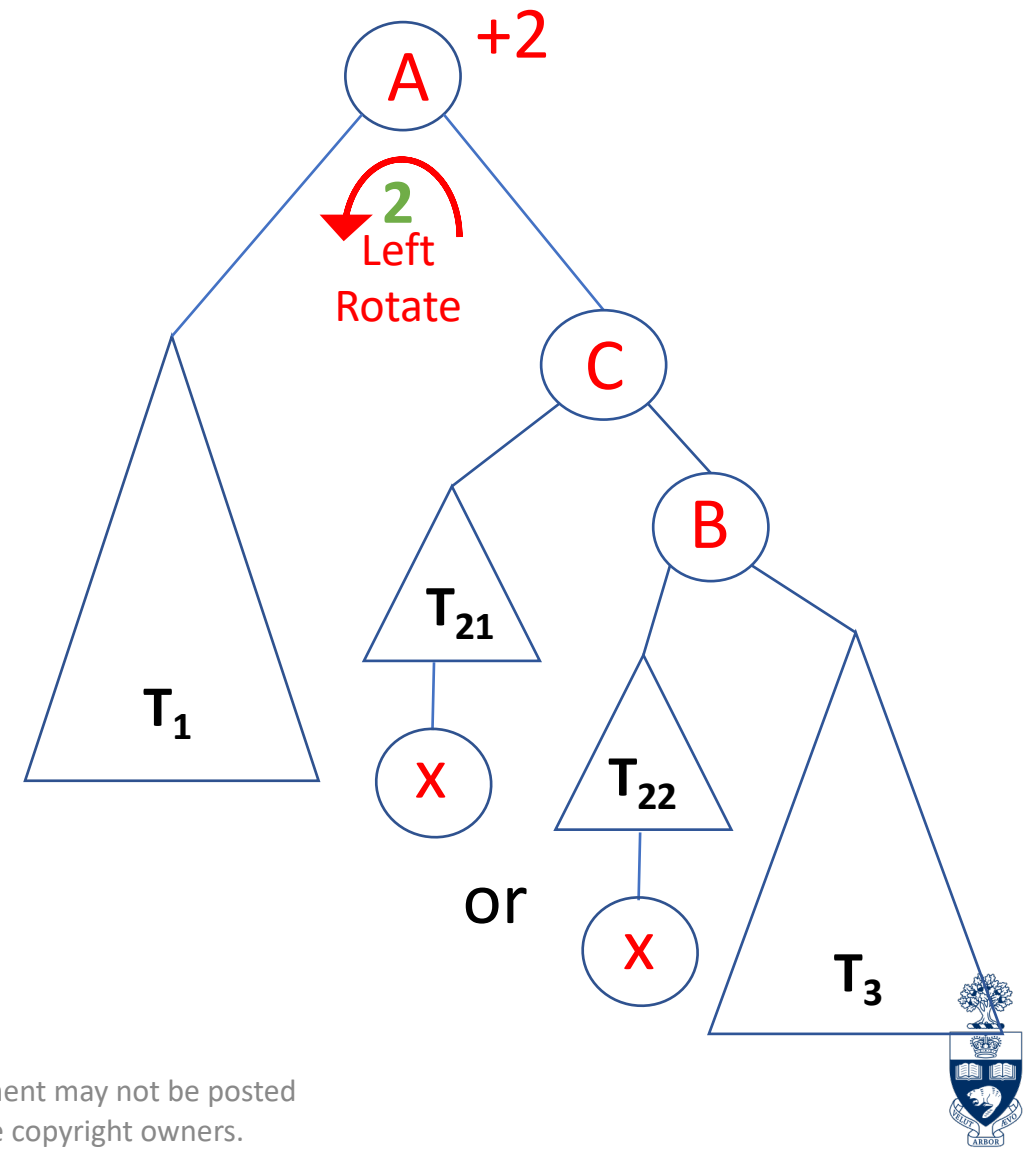
Case 1 (b) : h > -1



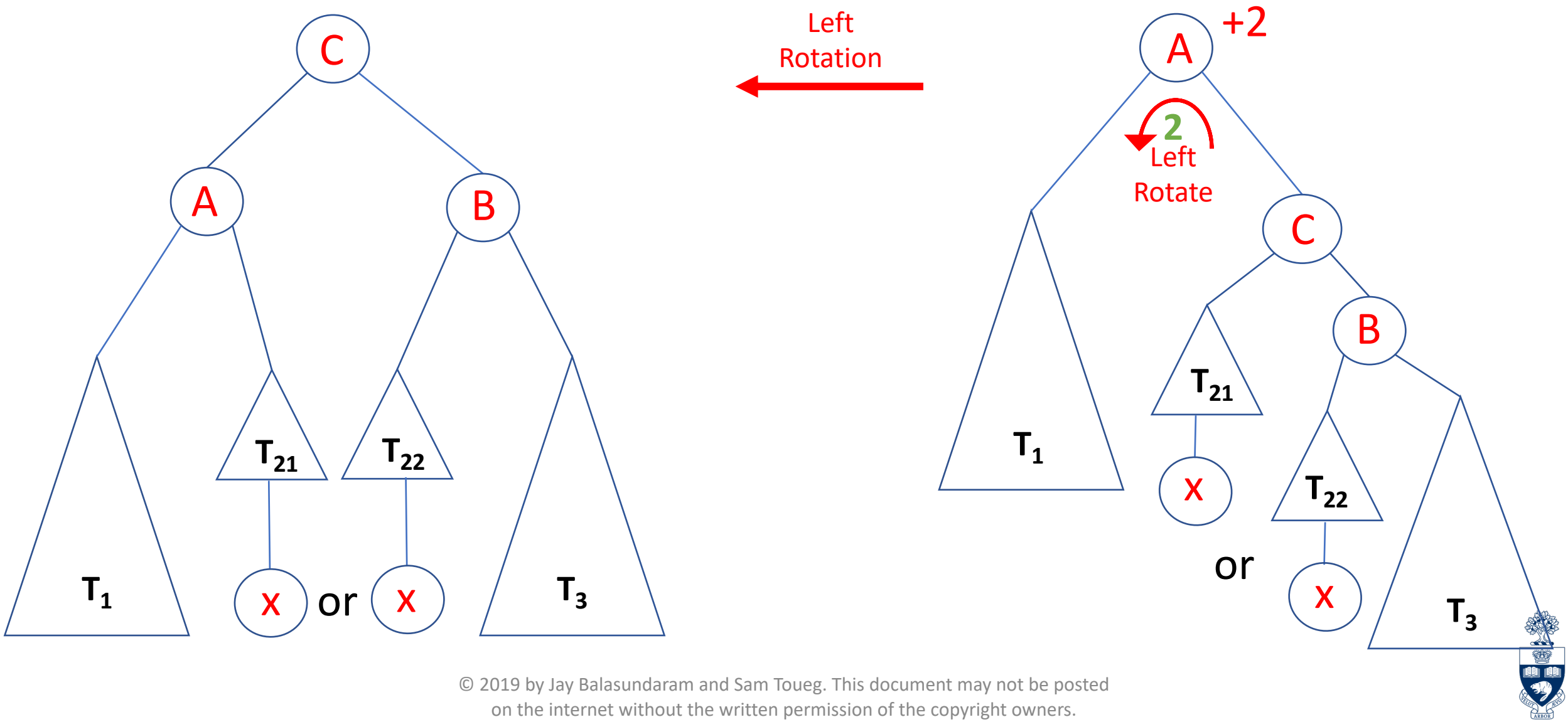
Case 1 (b) : h > -1



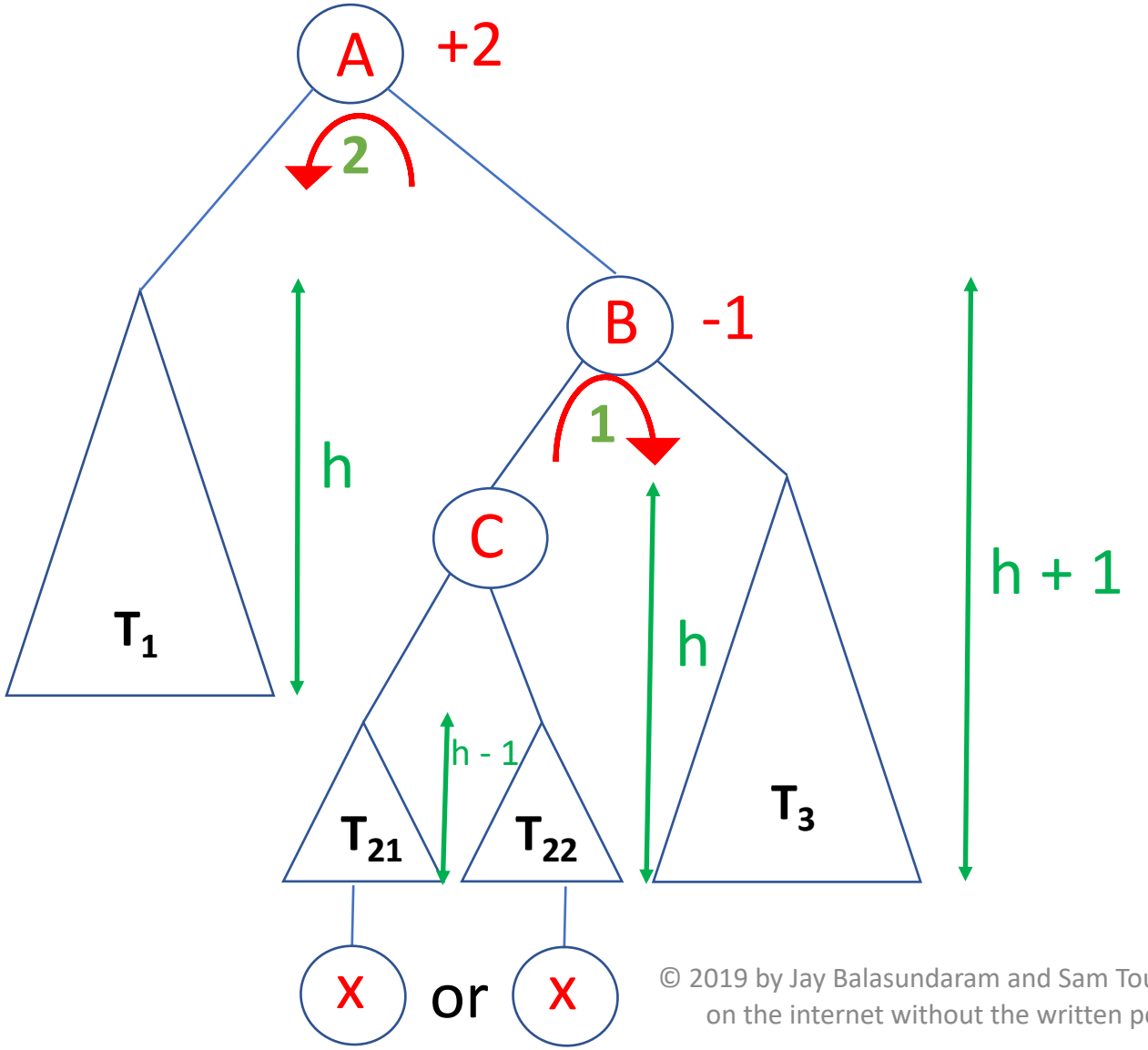
Case 1 (b) : h > -1



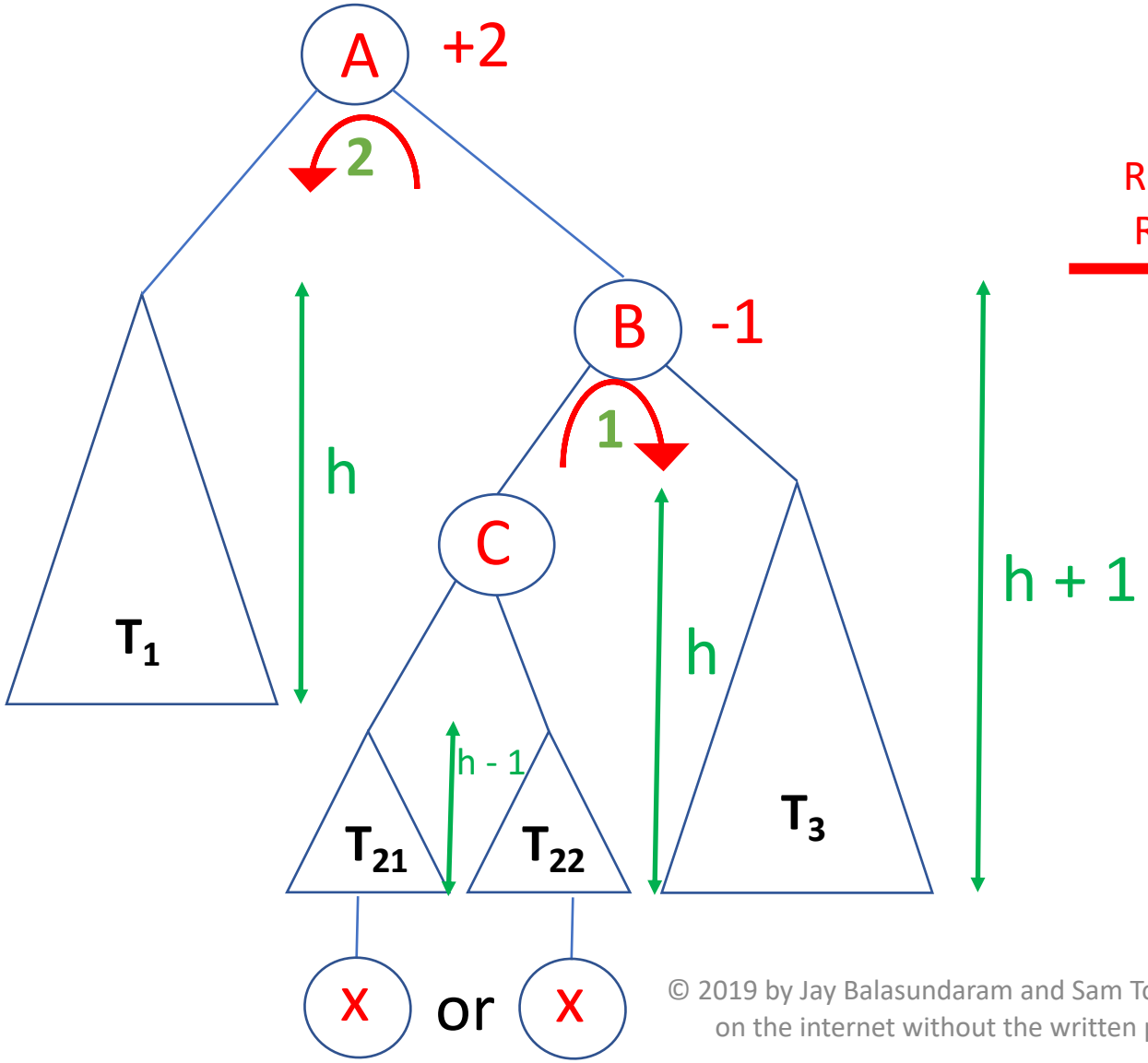
Case 1 (b) : h > -1



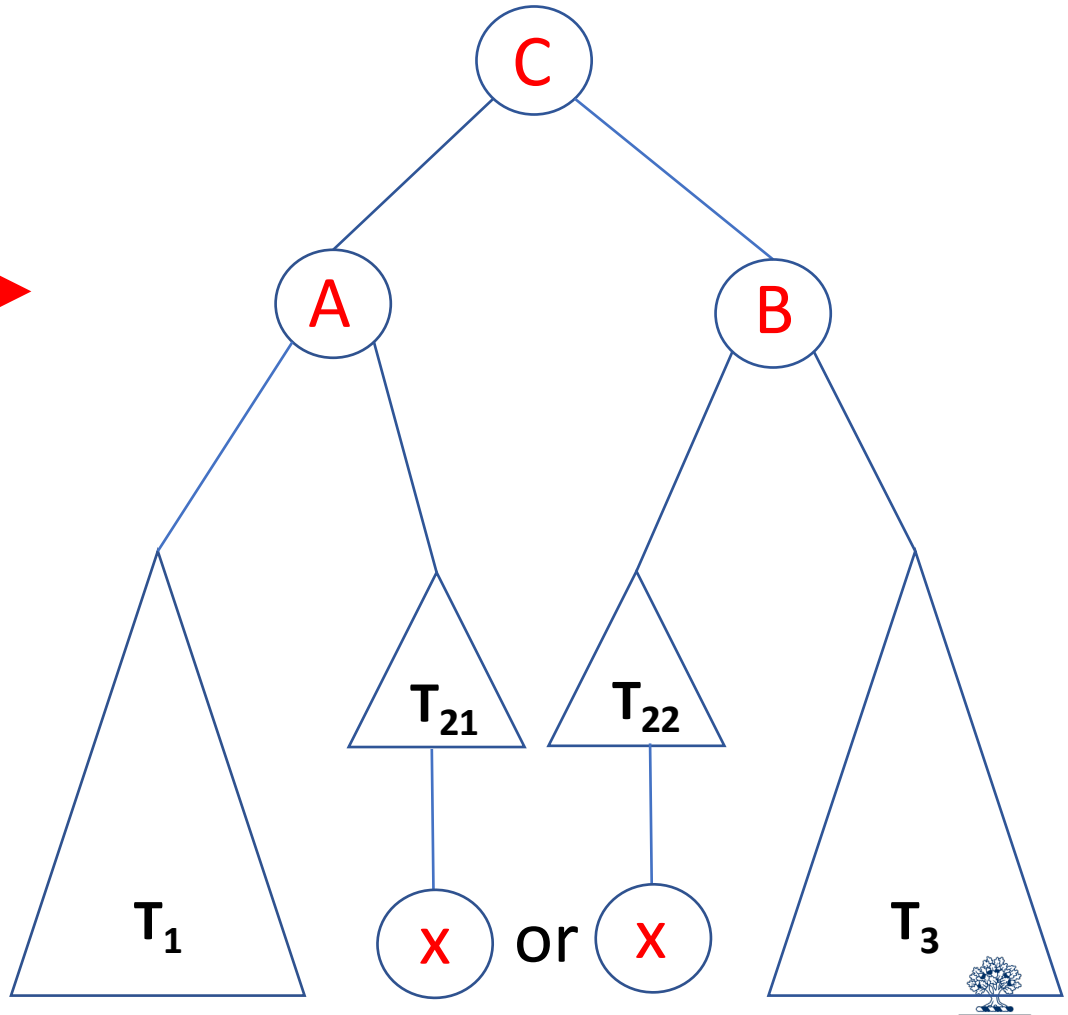
Case 1 (b) : h > -1



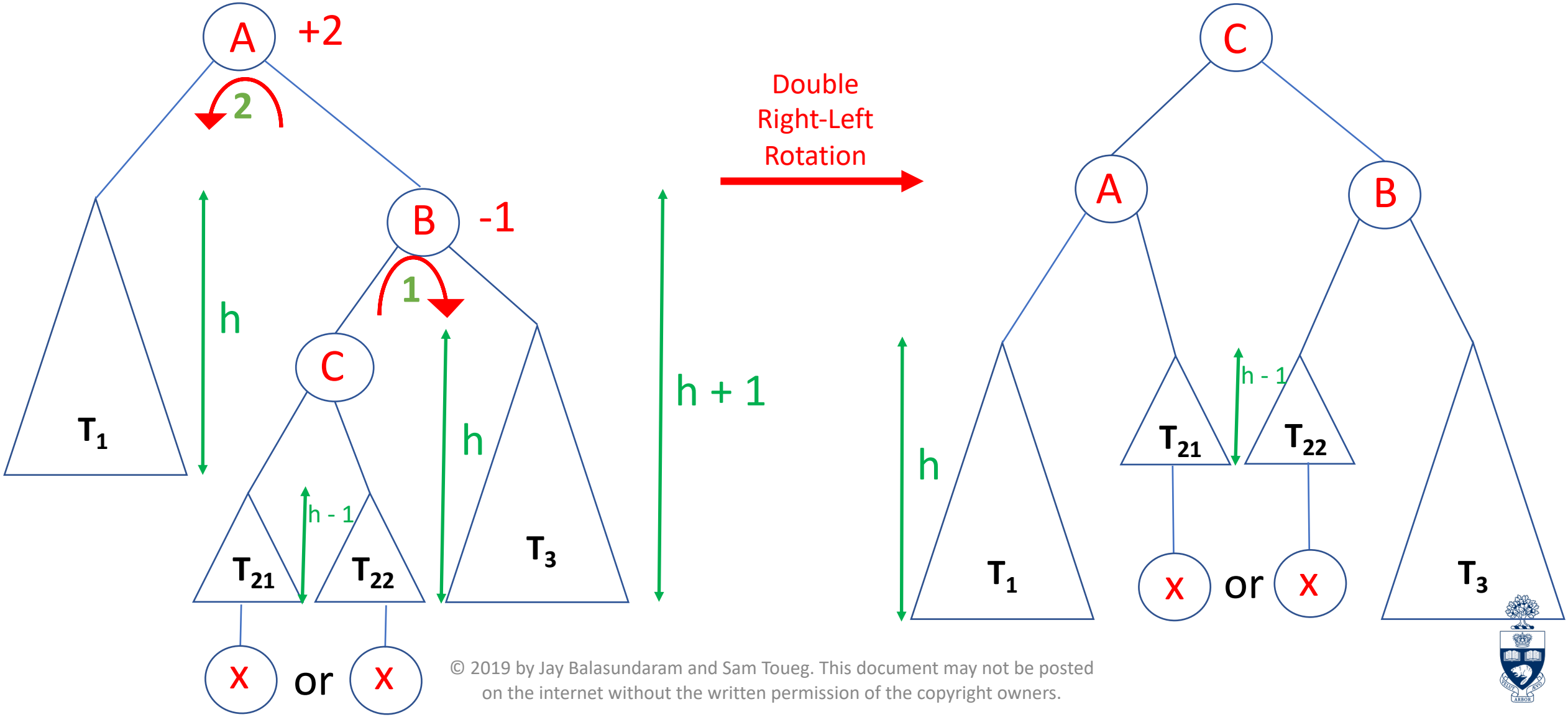
Case 1 (b) : h > -1



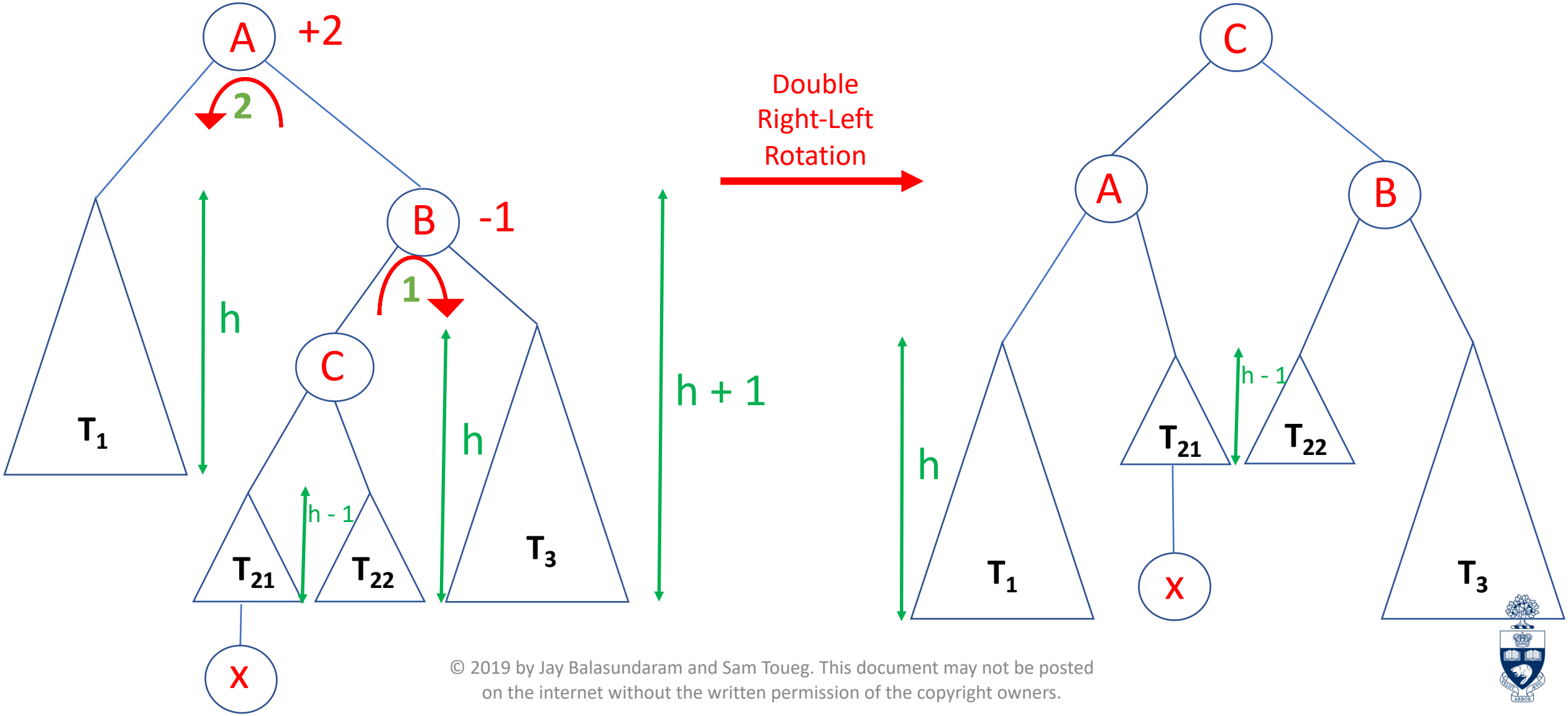
Double
Right-Left
Rotation



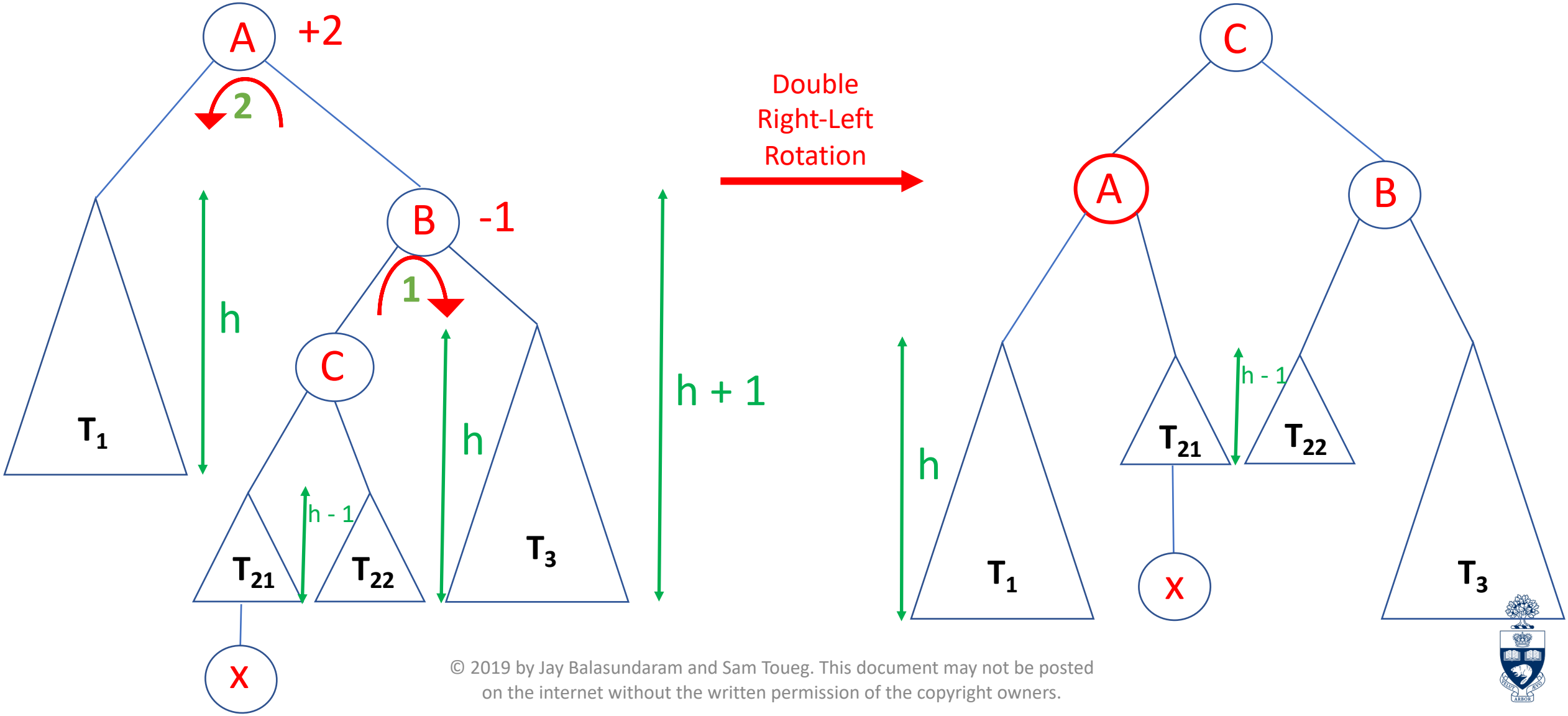
Case 1 (b) : h > -1



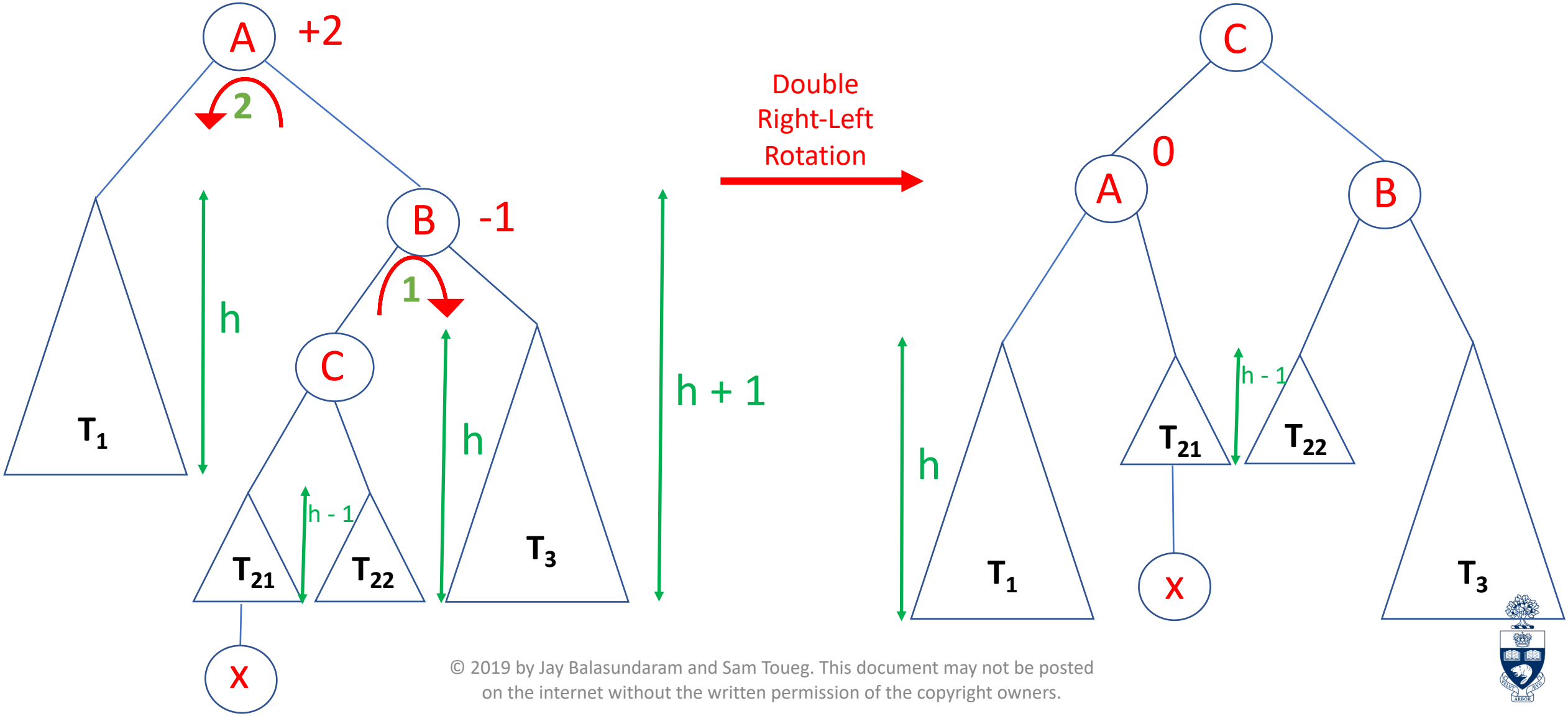
Case 1 (b) : h > -1



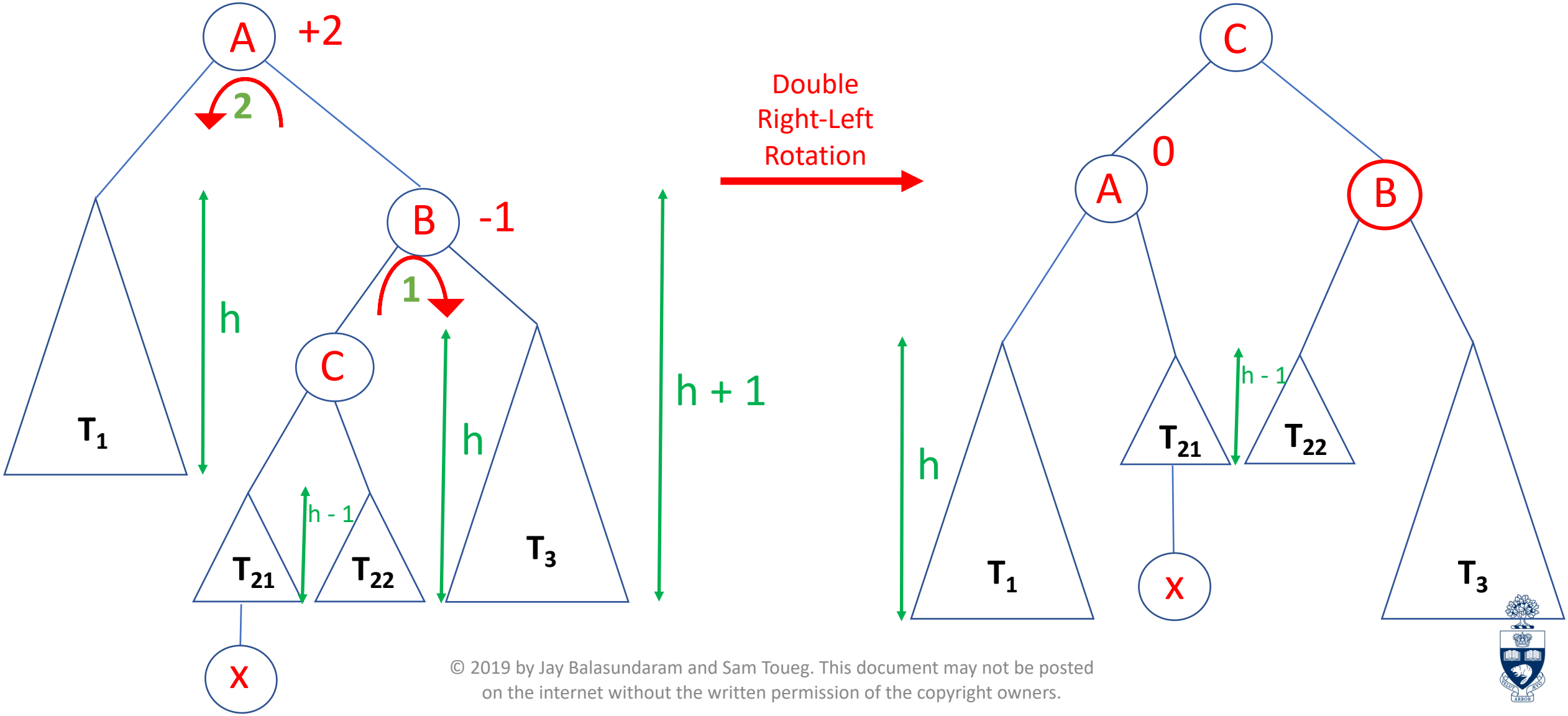
Case 1 (b) : h > -1



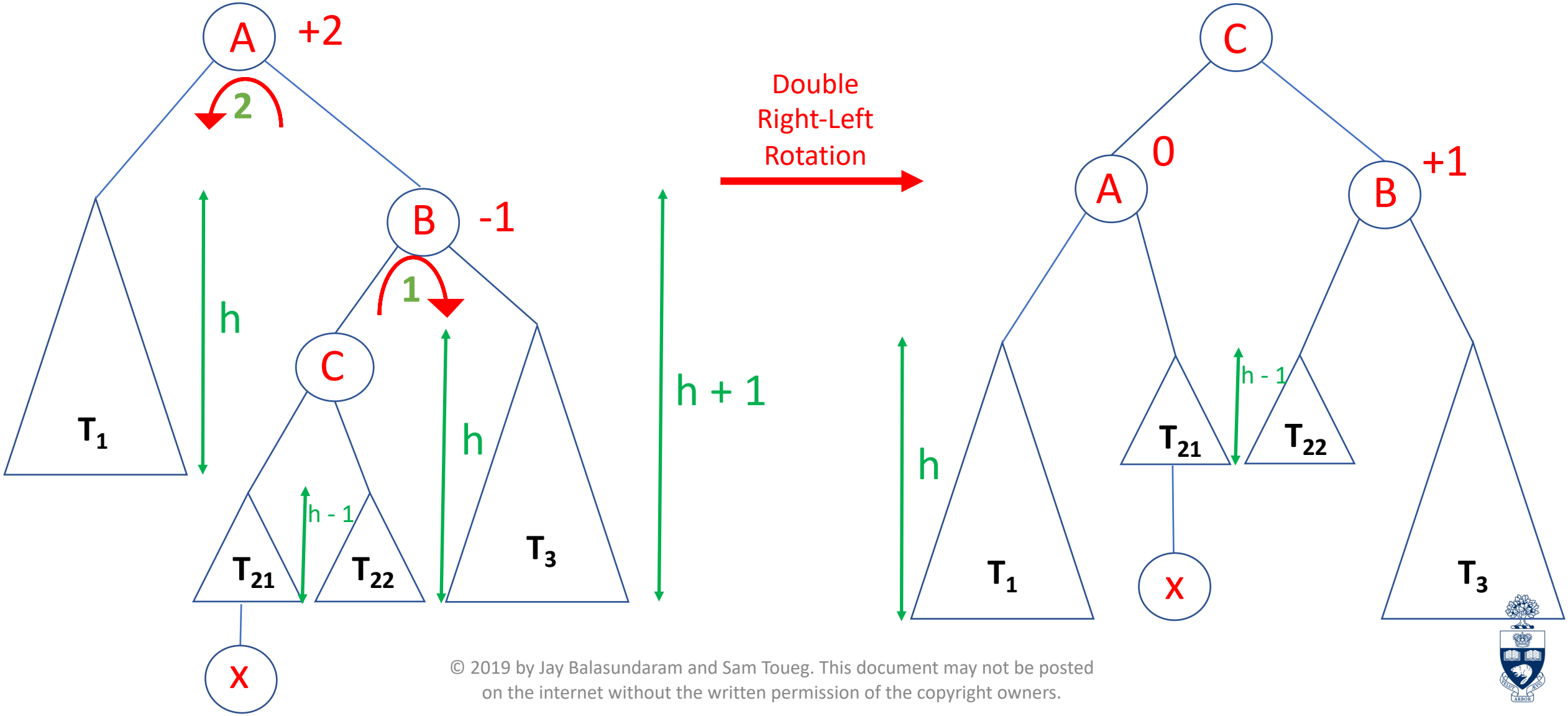
Case 1 (b) : h > -1



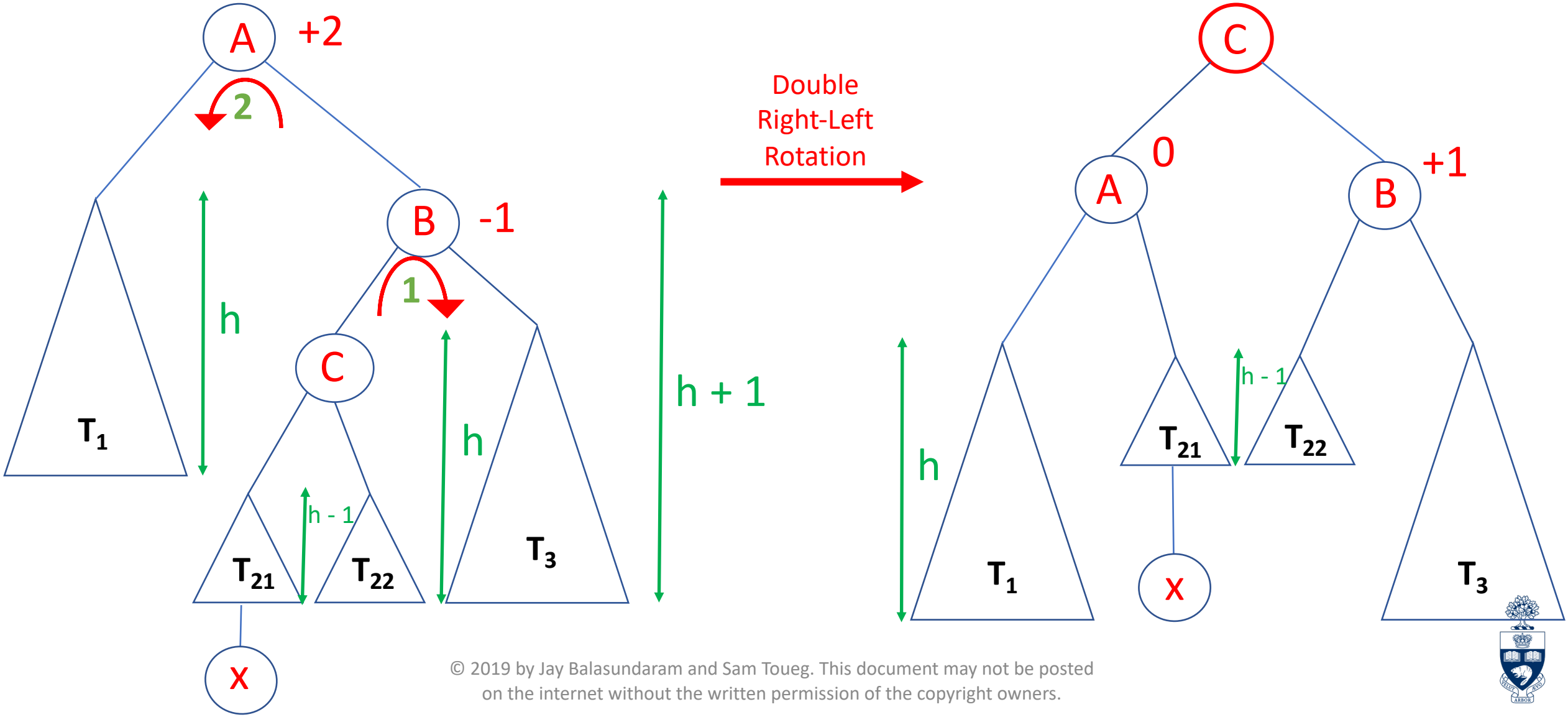
Case 1 (b) : h > -1



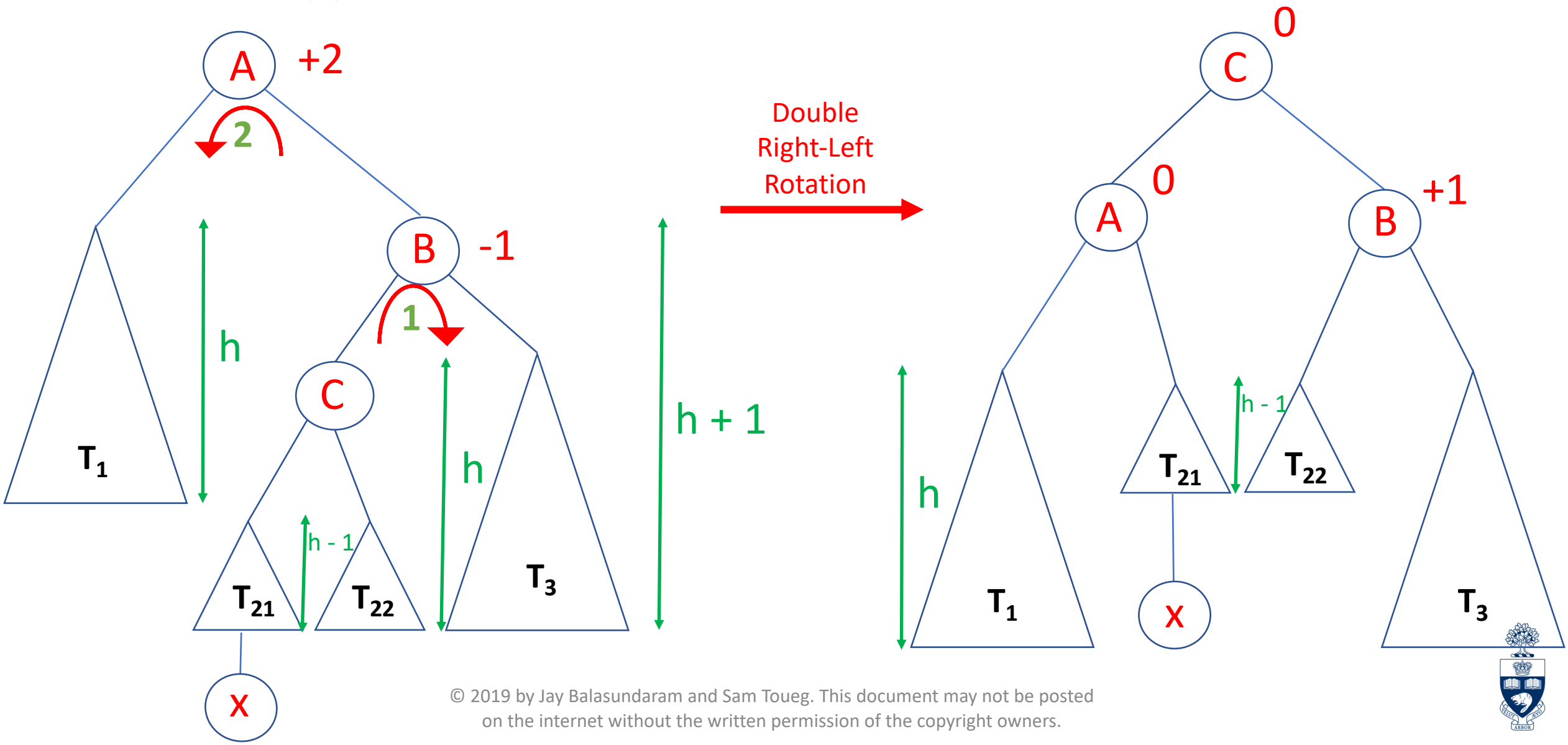
Case 1 (b) : h > -1



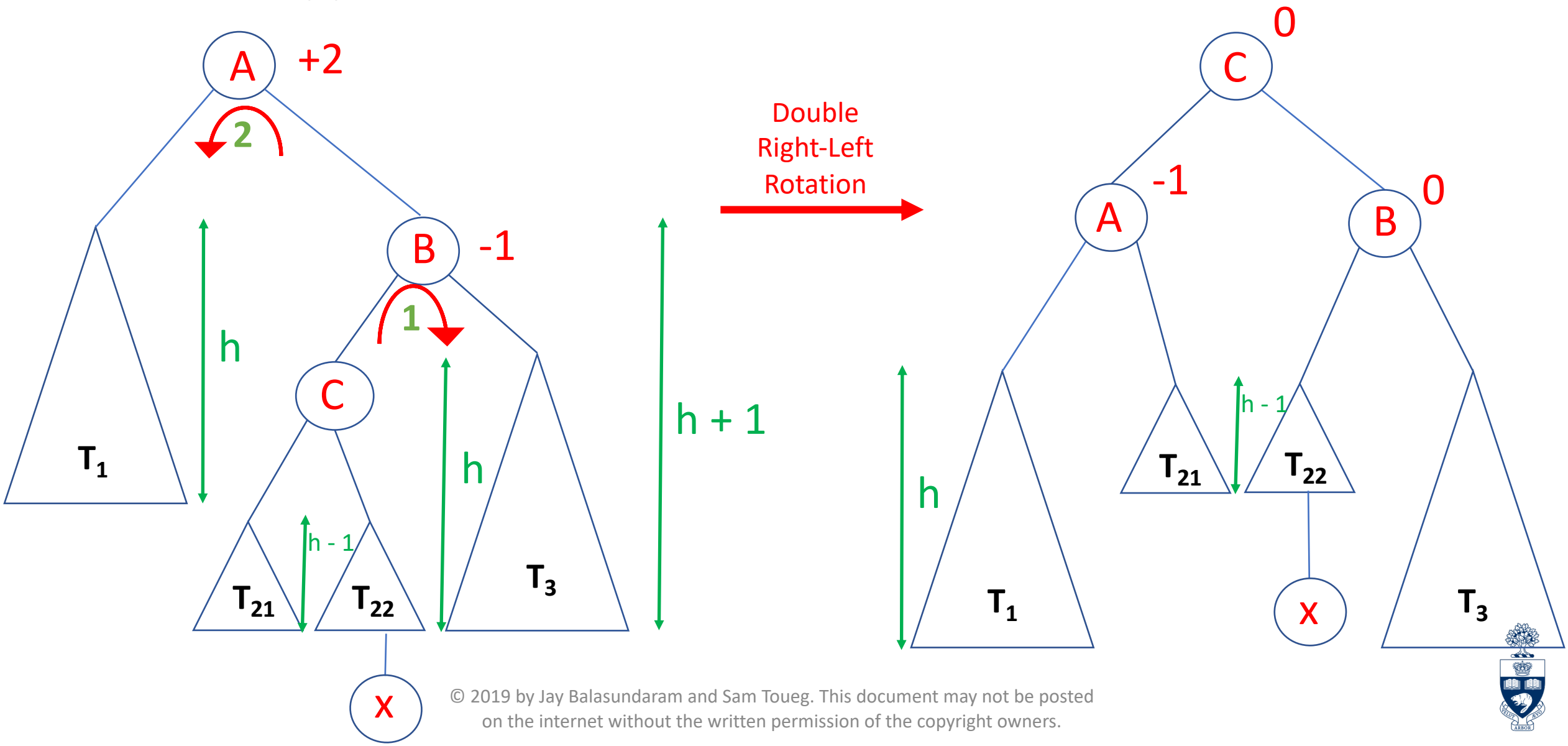
Case 1 (b) : h > -1



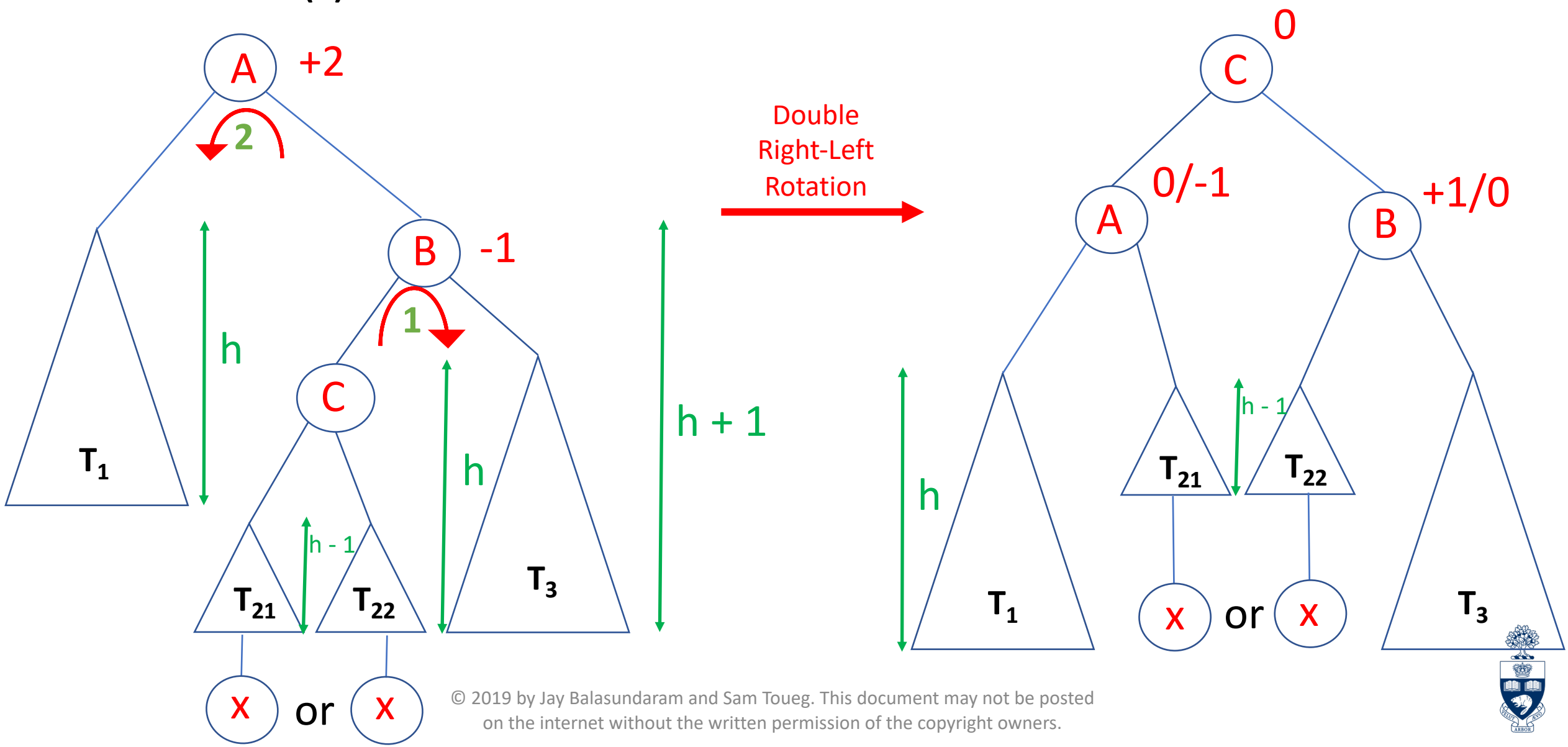
Case 1 (b) : h > -1



Case 1 (b) : h > -1

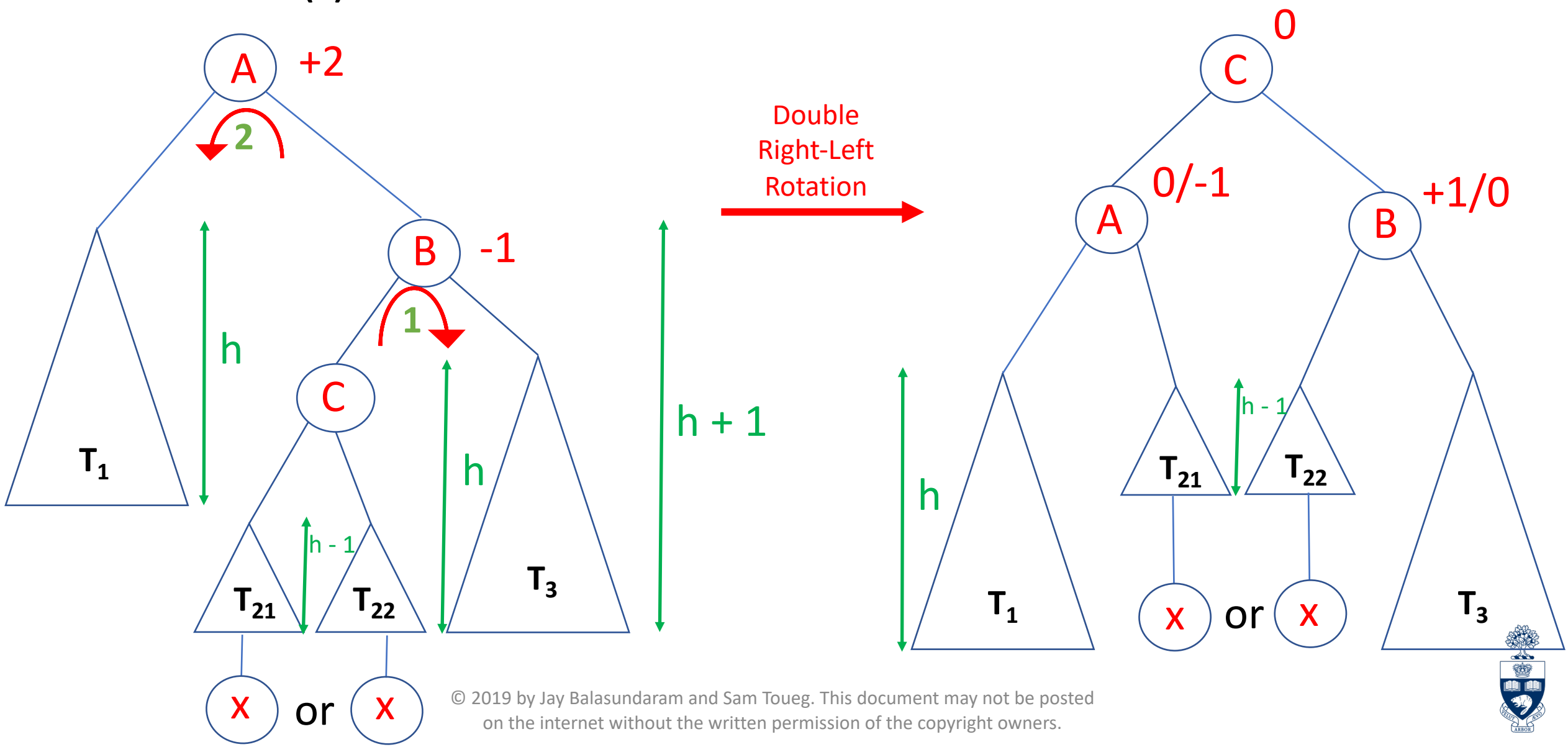


Case 1 (b) : h > -1



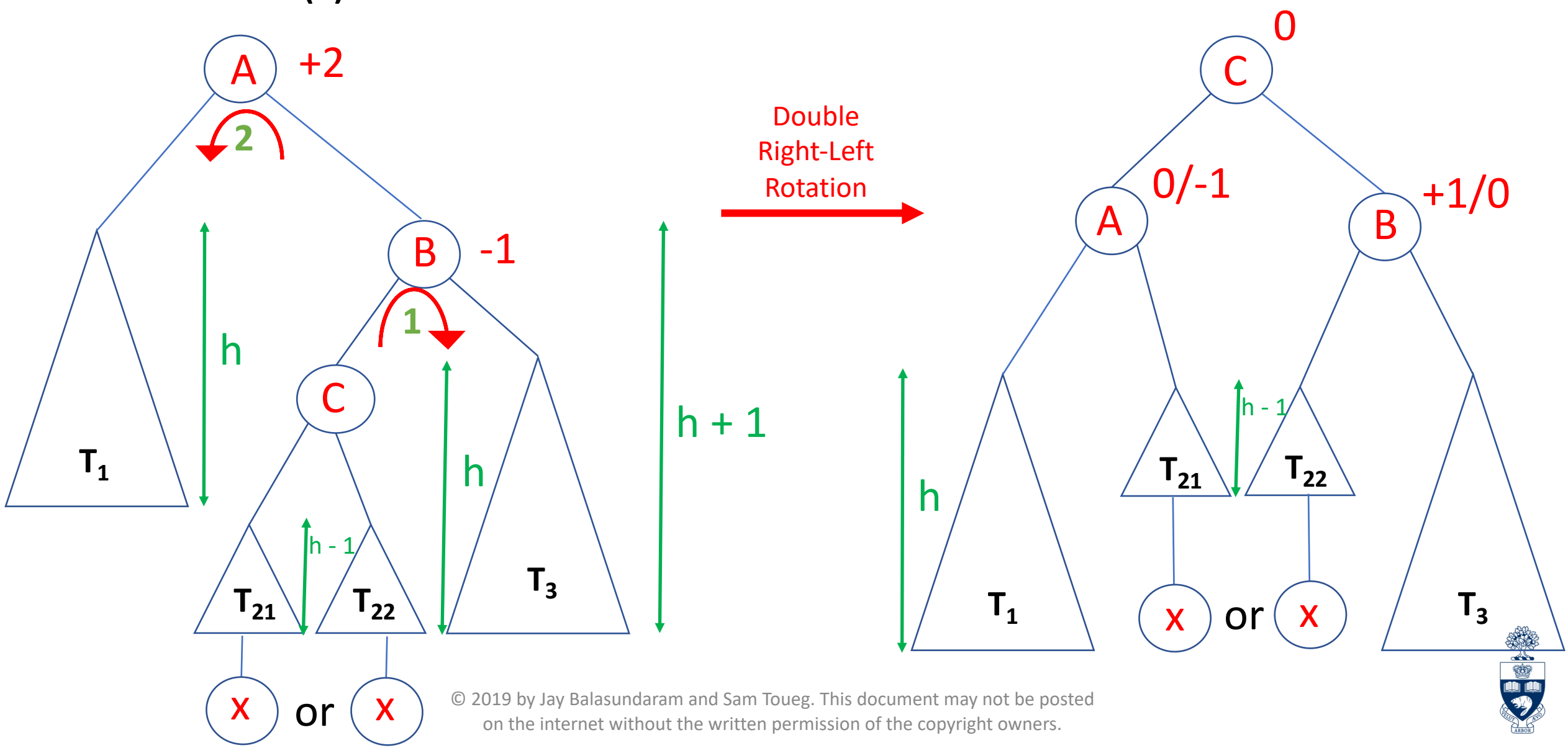
Rotation: (1) Rebalances the subtree

Case 1 (b) : h > -1



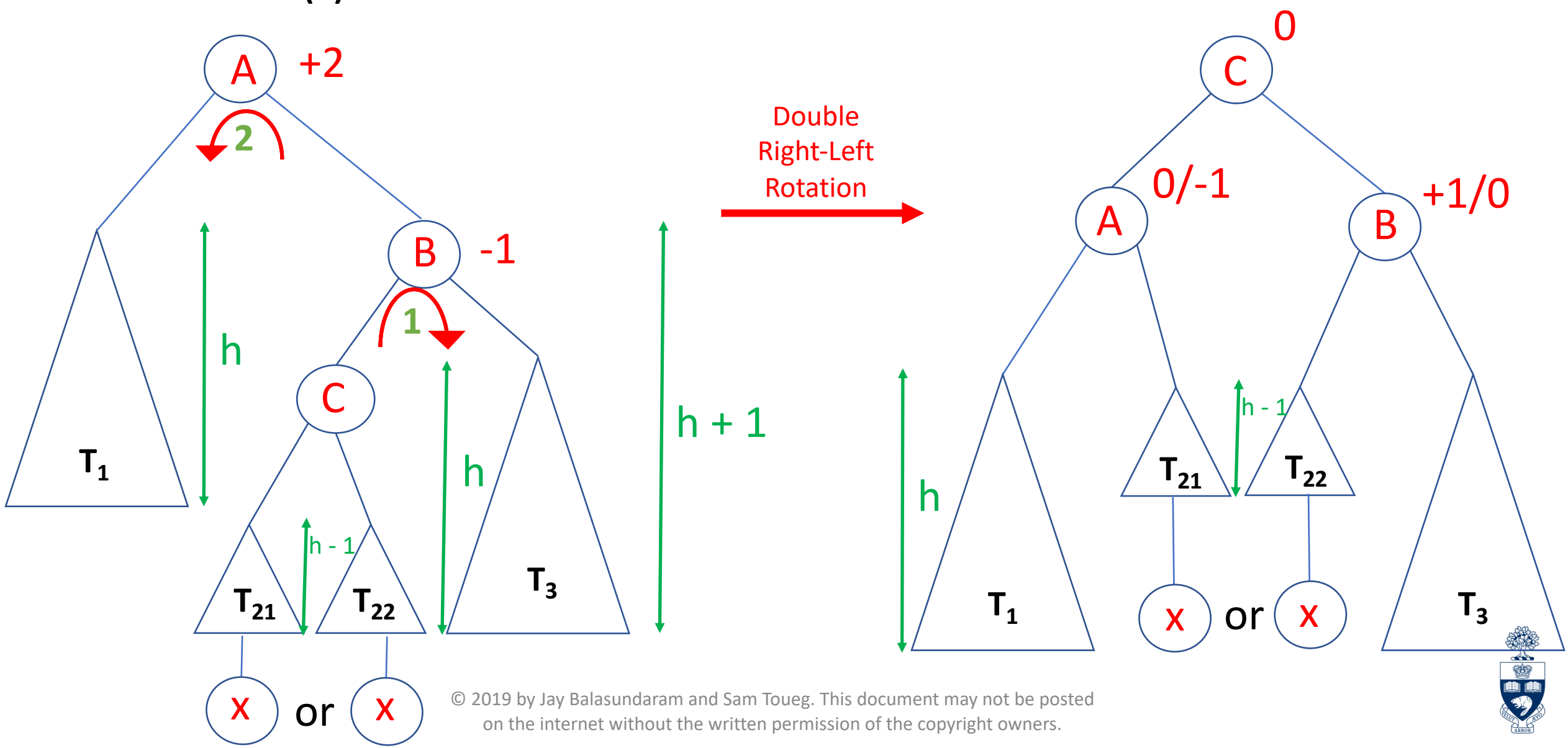
Rotation: (2) Preserves BST property

Case 1 (b) : h > -1



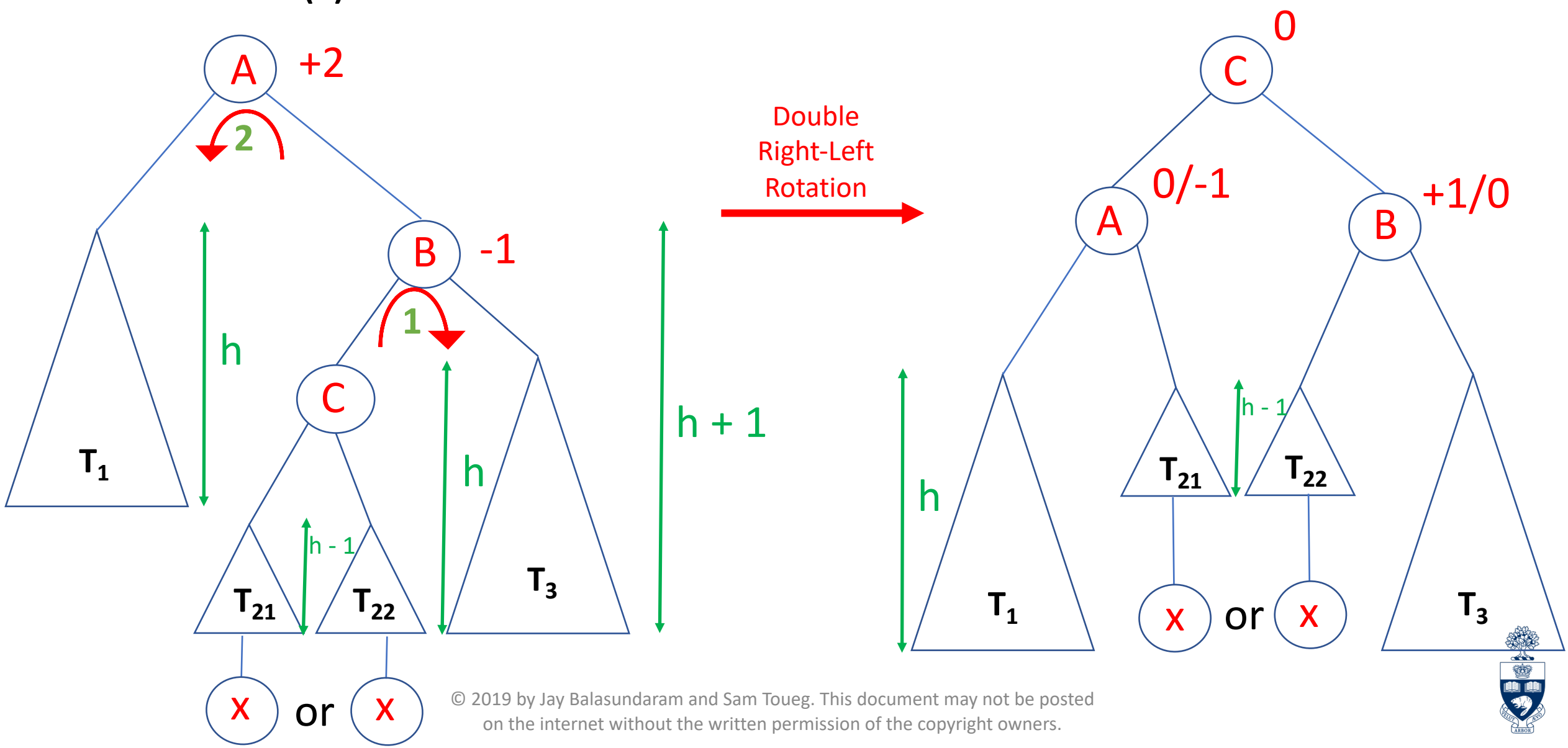
Rotation: (2) Preserves BST property (order:)

Case 1 (b) : h > -1



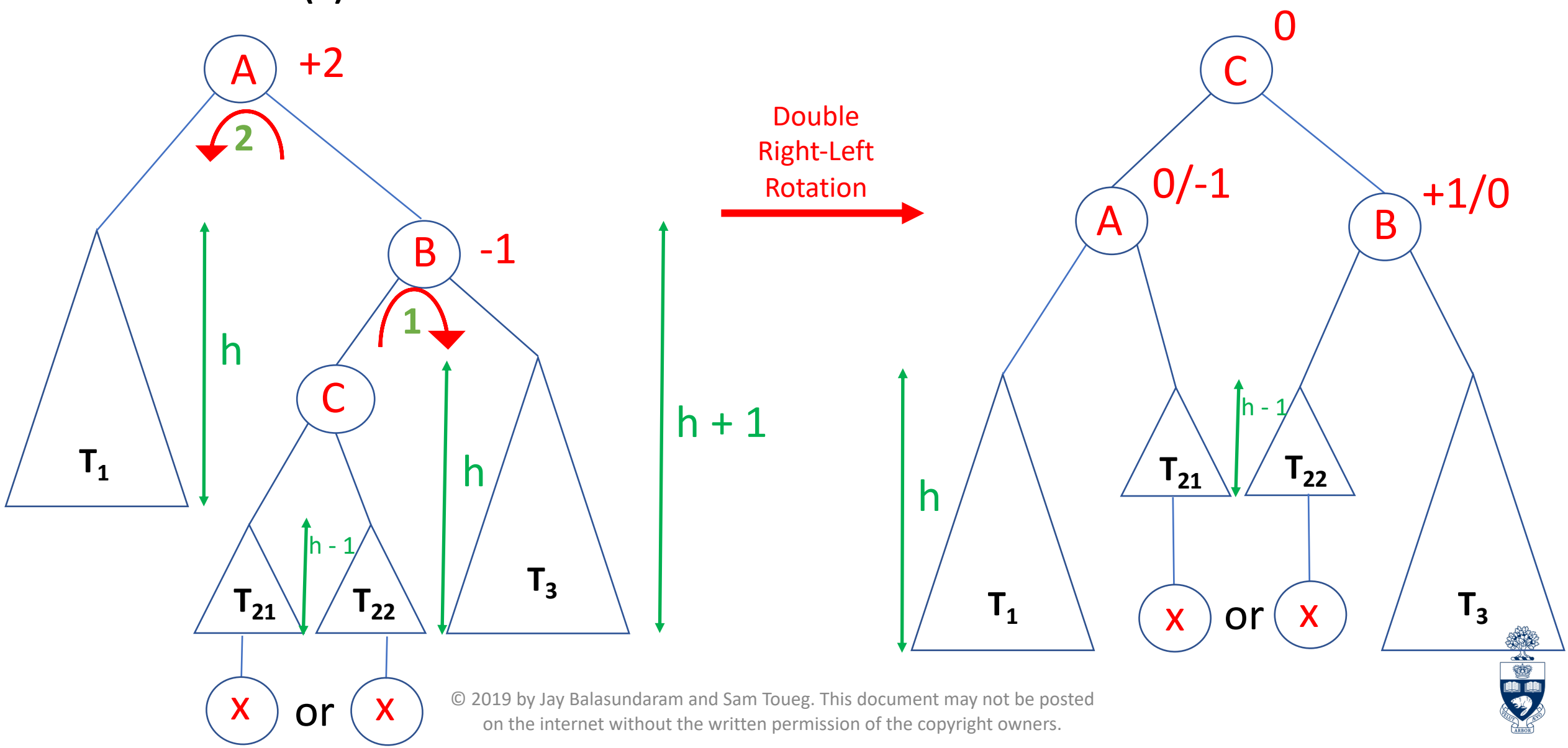
Rotation: (2) Preserves BST property (order: T_1)

Case 1 (b) : h > -1



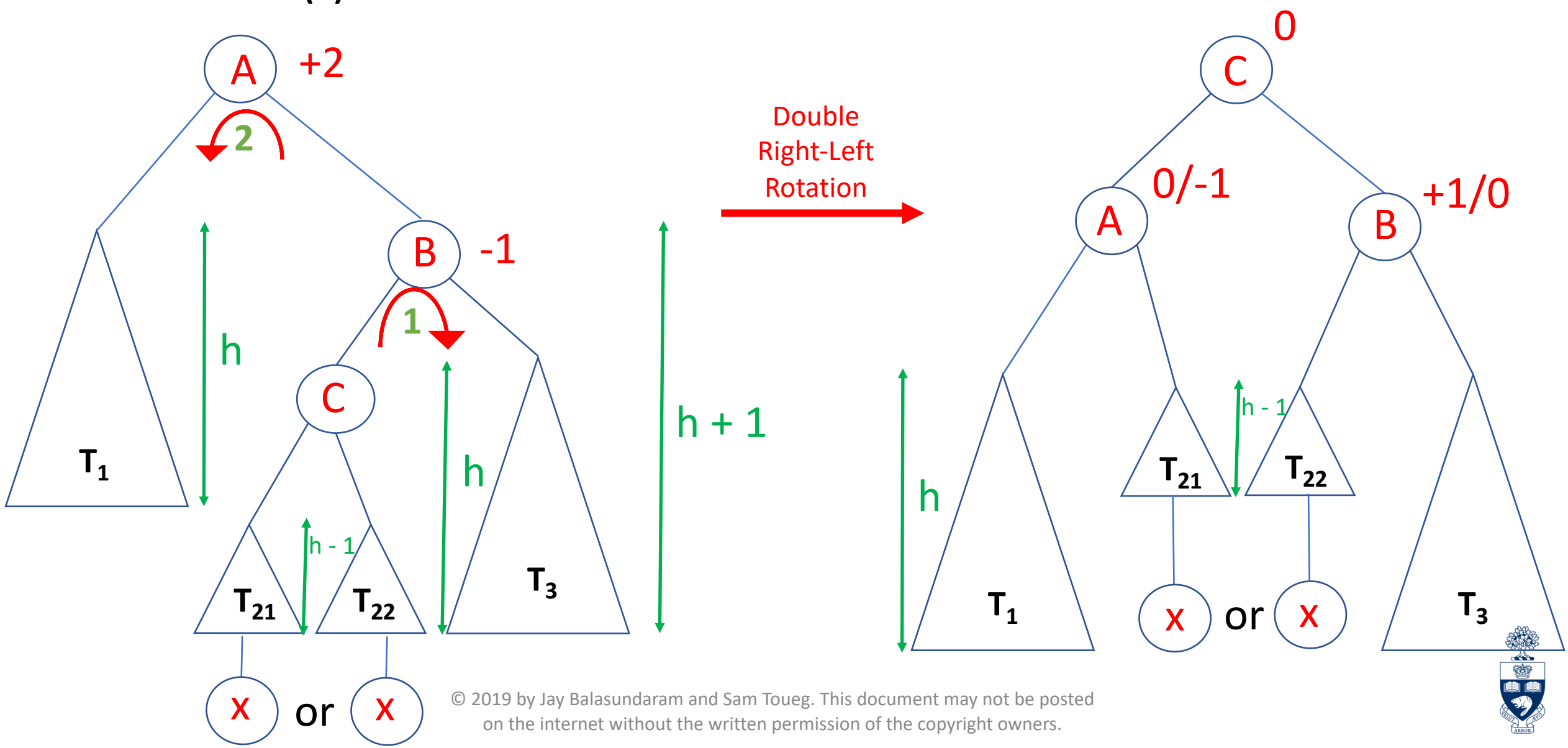
Rotation: (2) Preserves BST property (order: T_1 A)

Case 1 (b) : h > -1



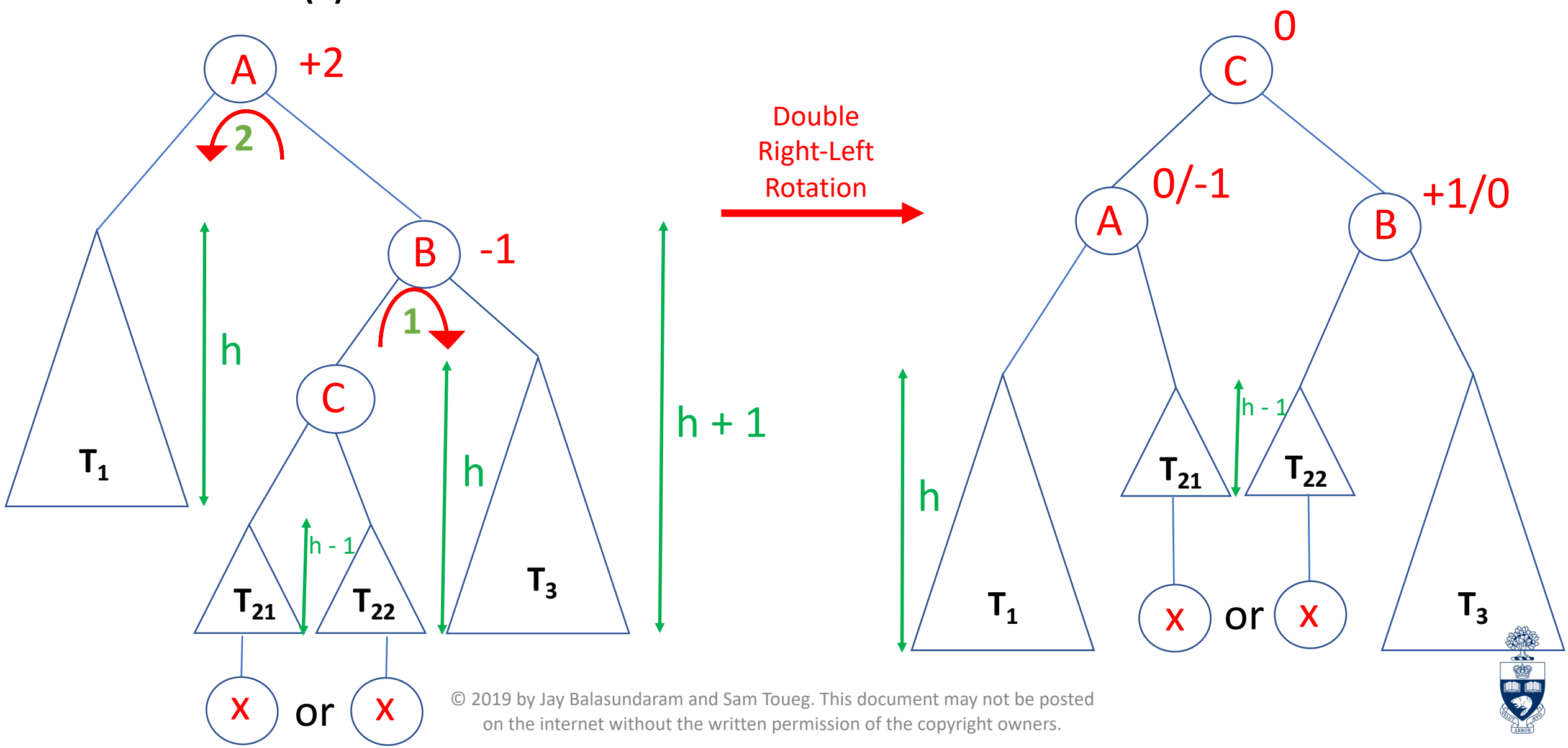
Rotation: (2) Preserves BST property (order: T_1 A T_{21})

Case 1 (b) : h > -1



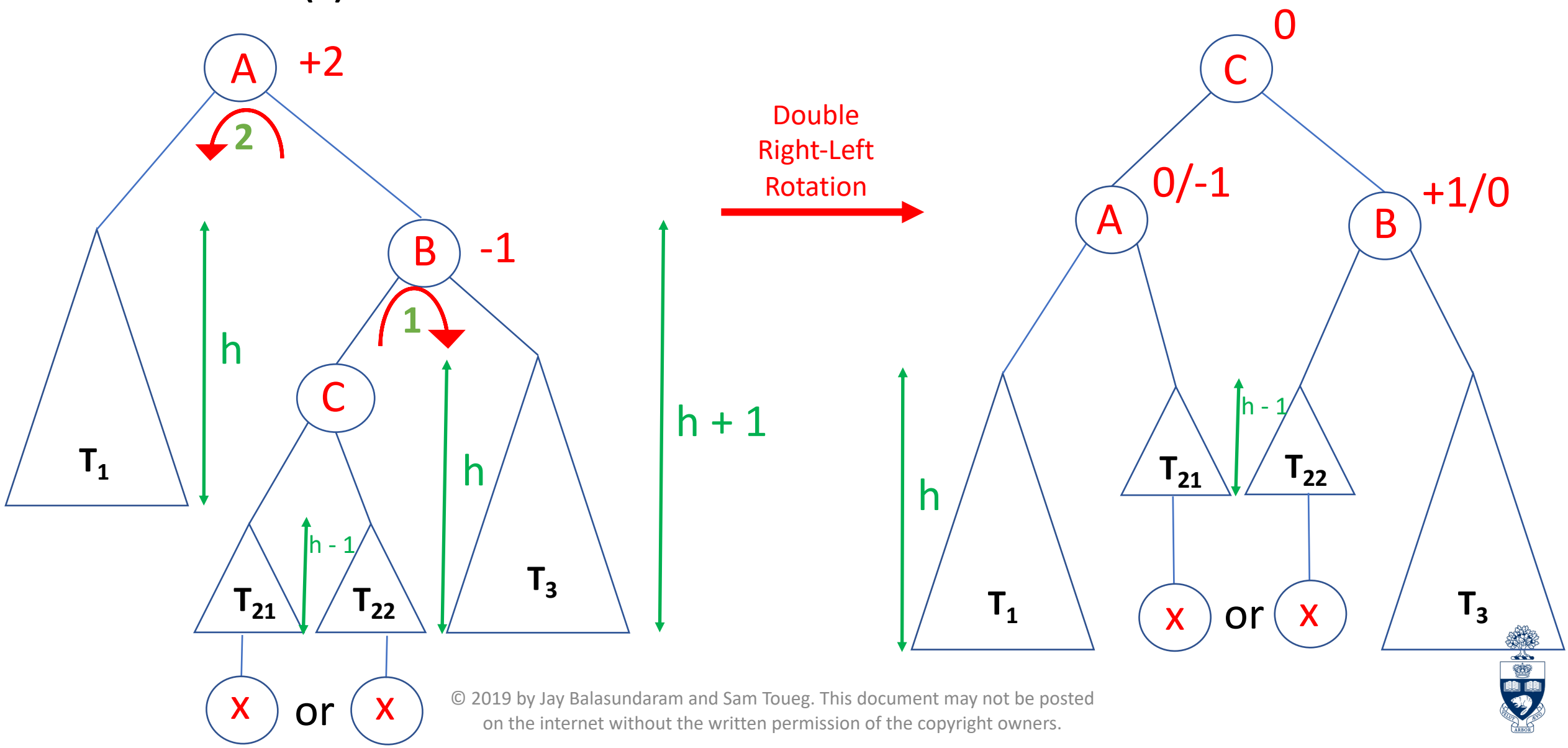
Rotation: (2) Preserves BST property (order: T_1 A T_{21} C T_{22} B T_3)

Case 1 (b) : h > -1



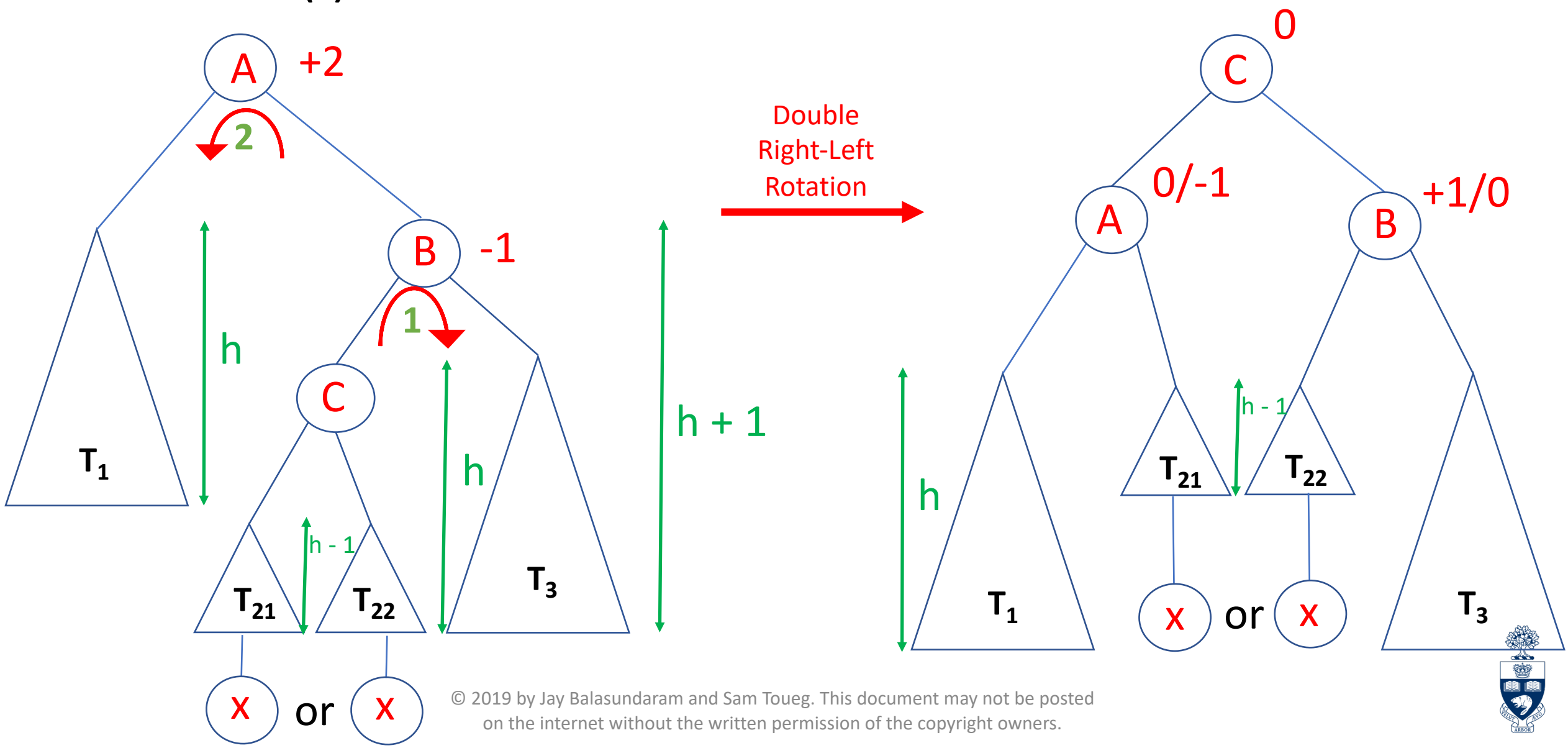
Rotation: (2) Preserves BST property (order: T_1 A T_{21} C T_{22})

Case 1 (b) : h > -1



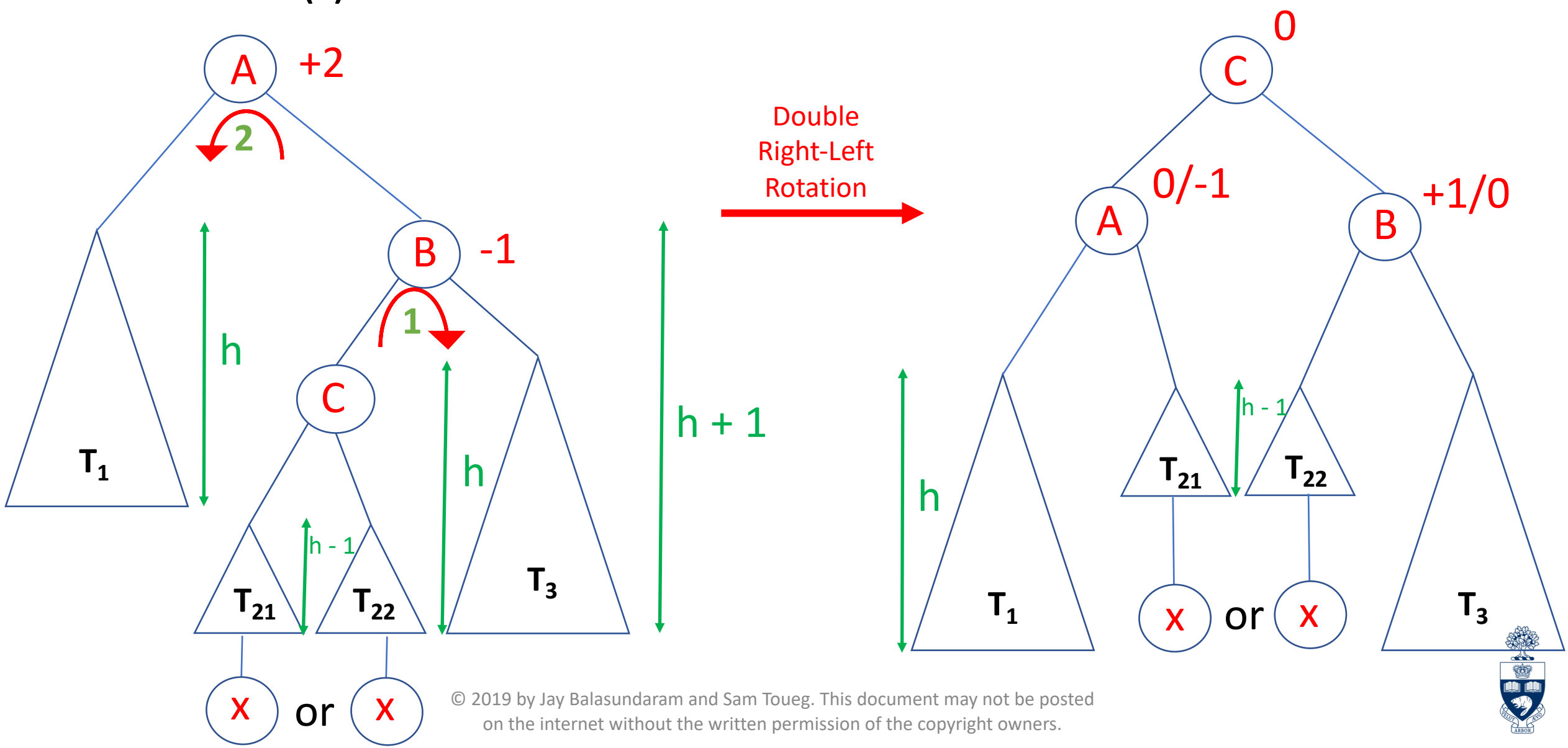
Rotation: (2) Preserves BST property (order: T_1 A T_{21} C T_{22} B)

Case 1 (b) : h > -1



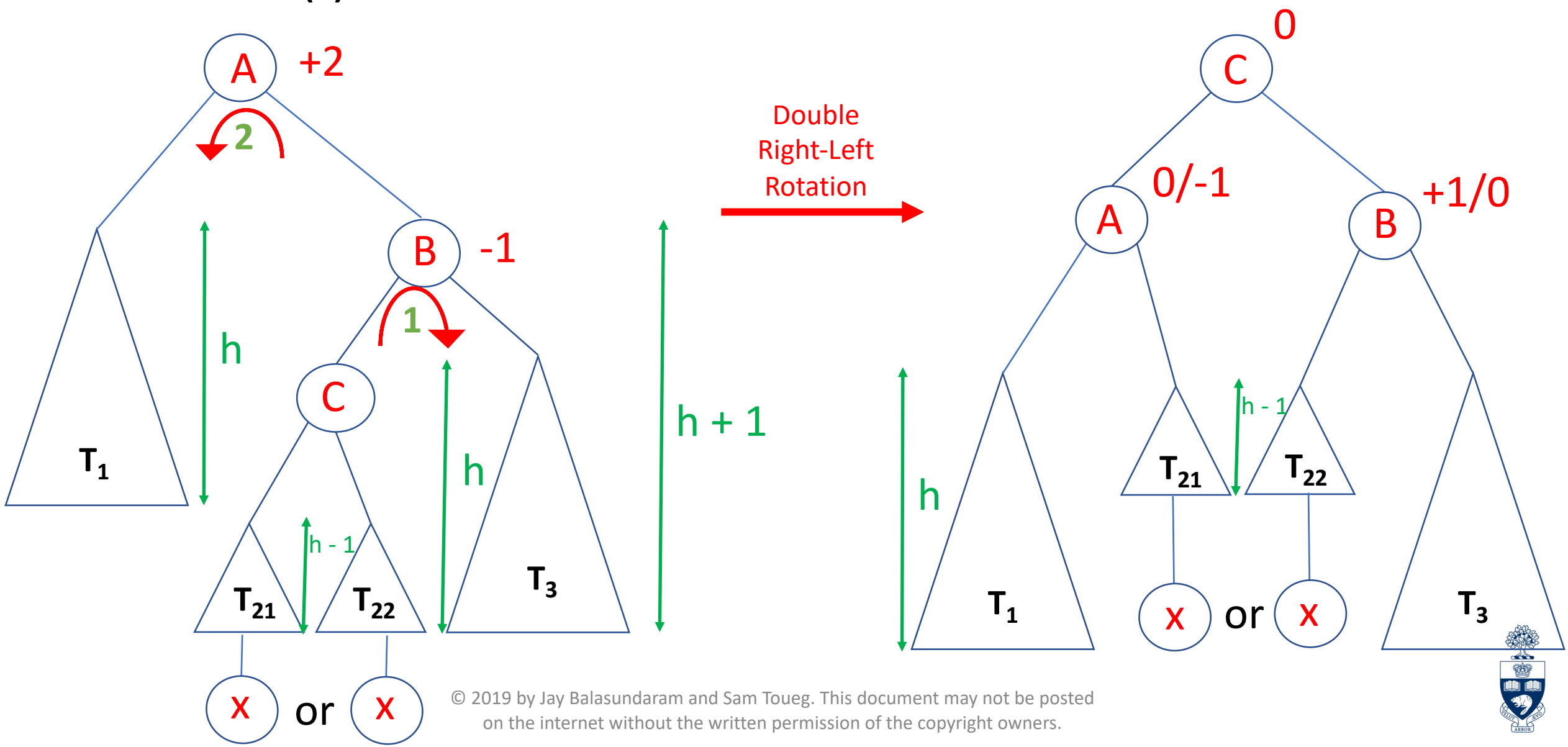
Rotation: (2) Preserves BST property (order: T_1 A T_{21} C T_{22} B T_3)

Case 1 (b) : h > -1



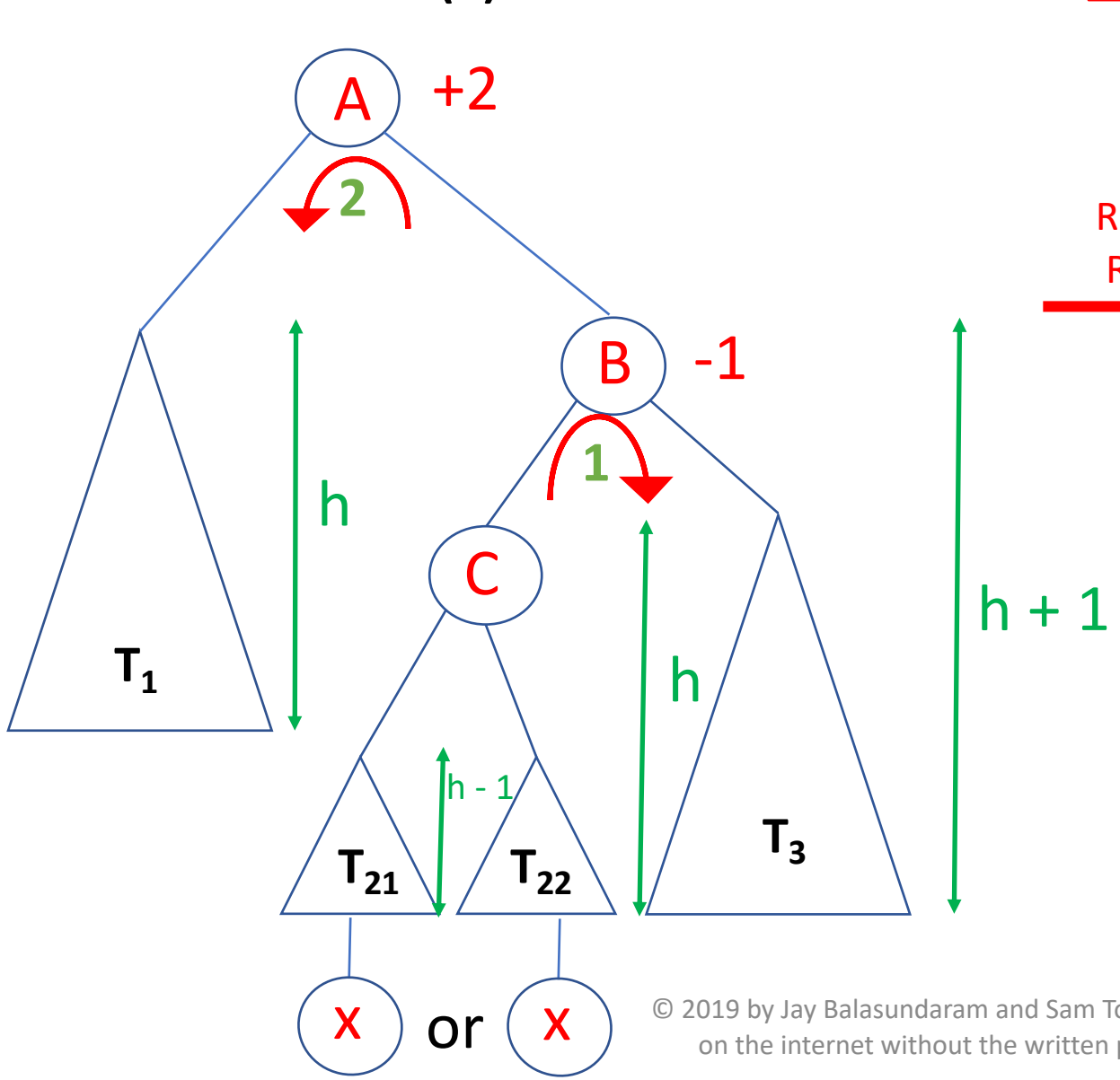
Rotation: (3) Bonus?

Case 1 (b) : h > -1

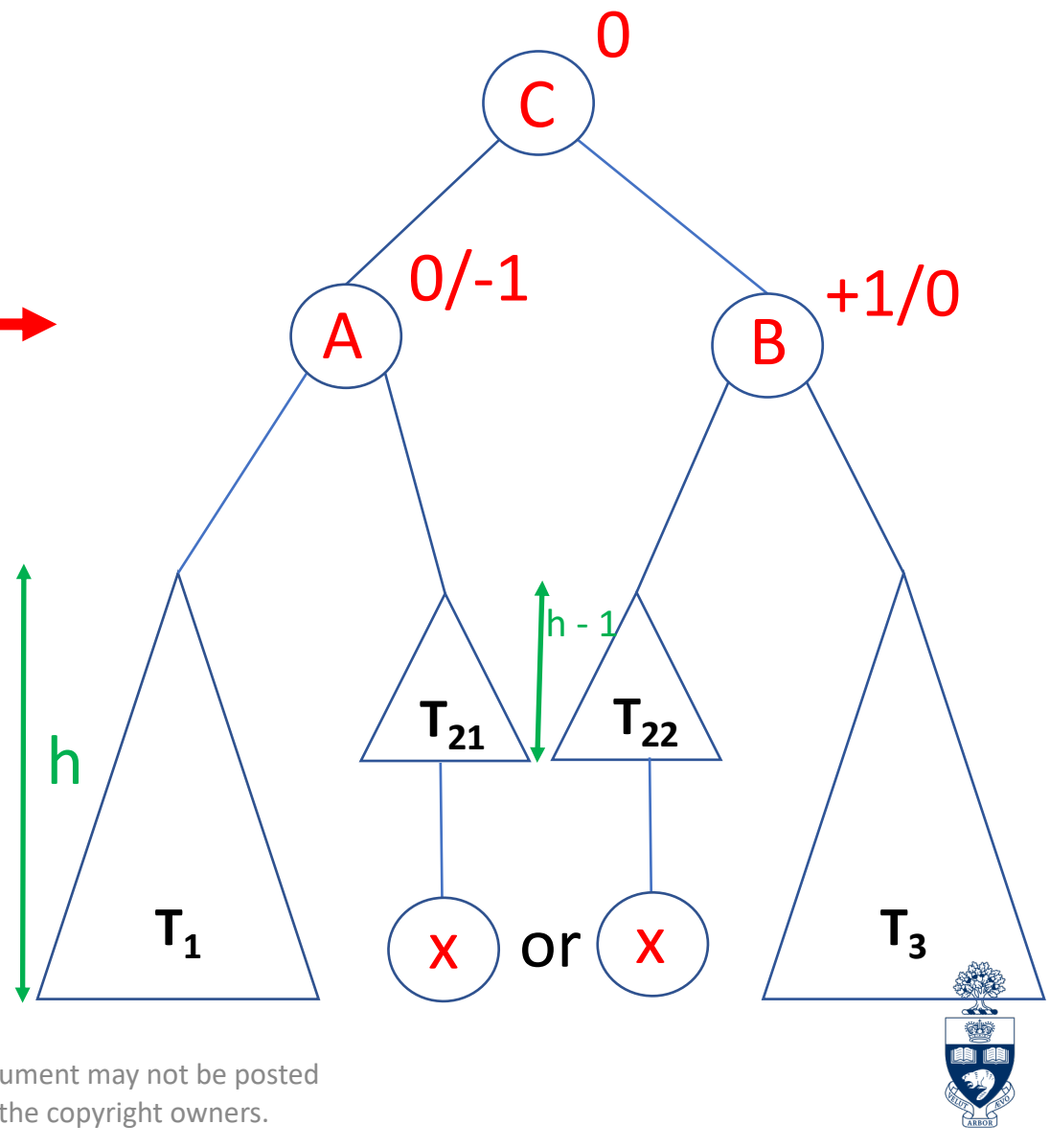


Rotation: (3) Bonus: height of this subtree is the same both before and after the insertion of x

Case 1 (b) : h > -1

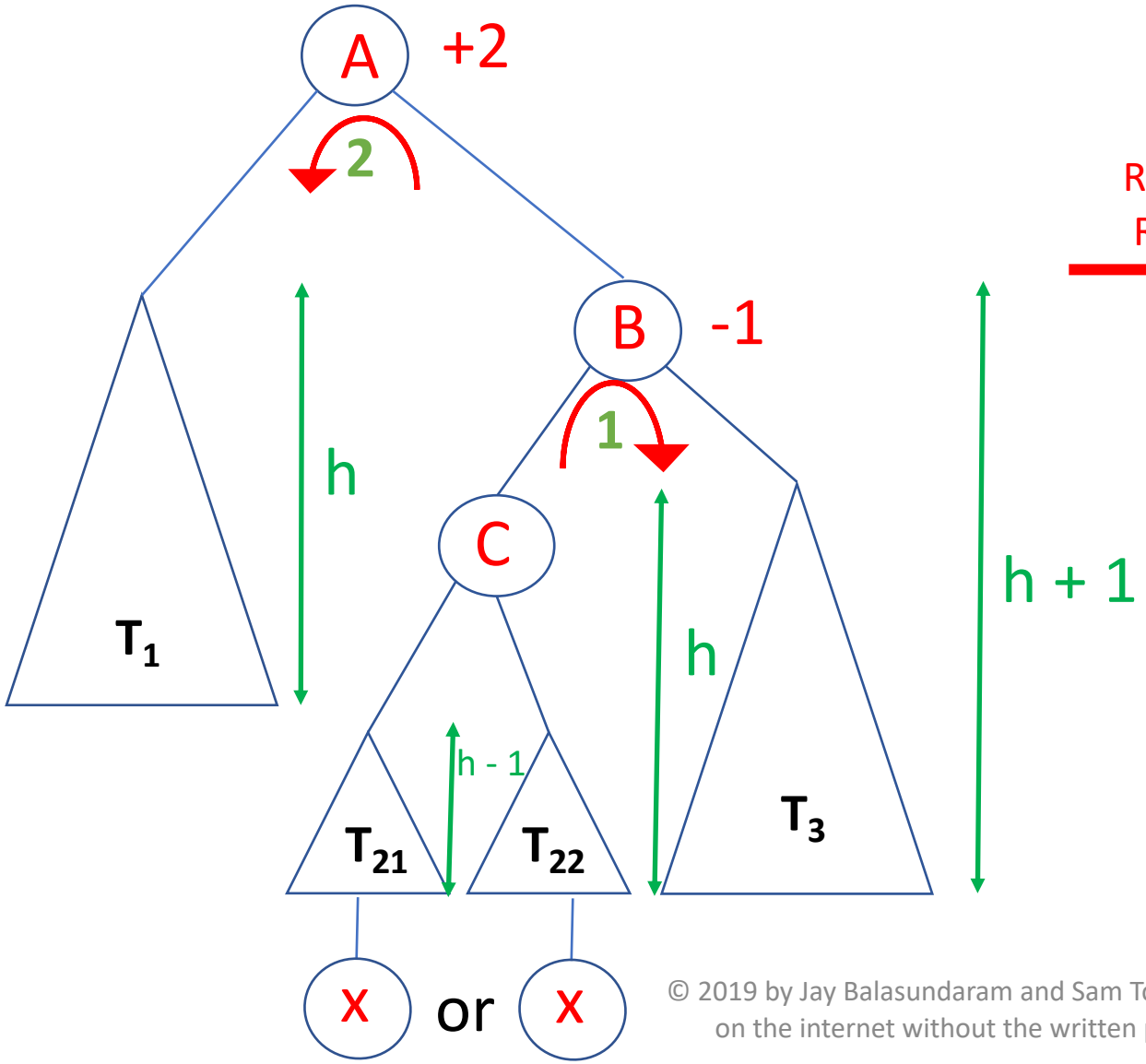


Double Right-Left Rotation

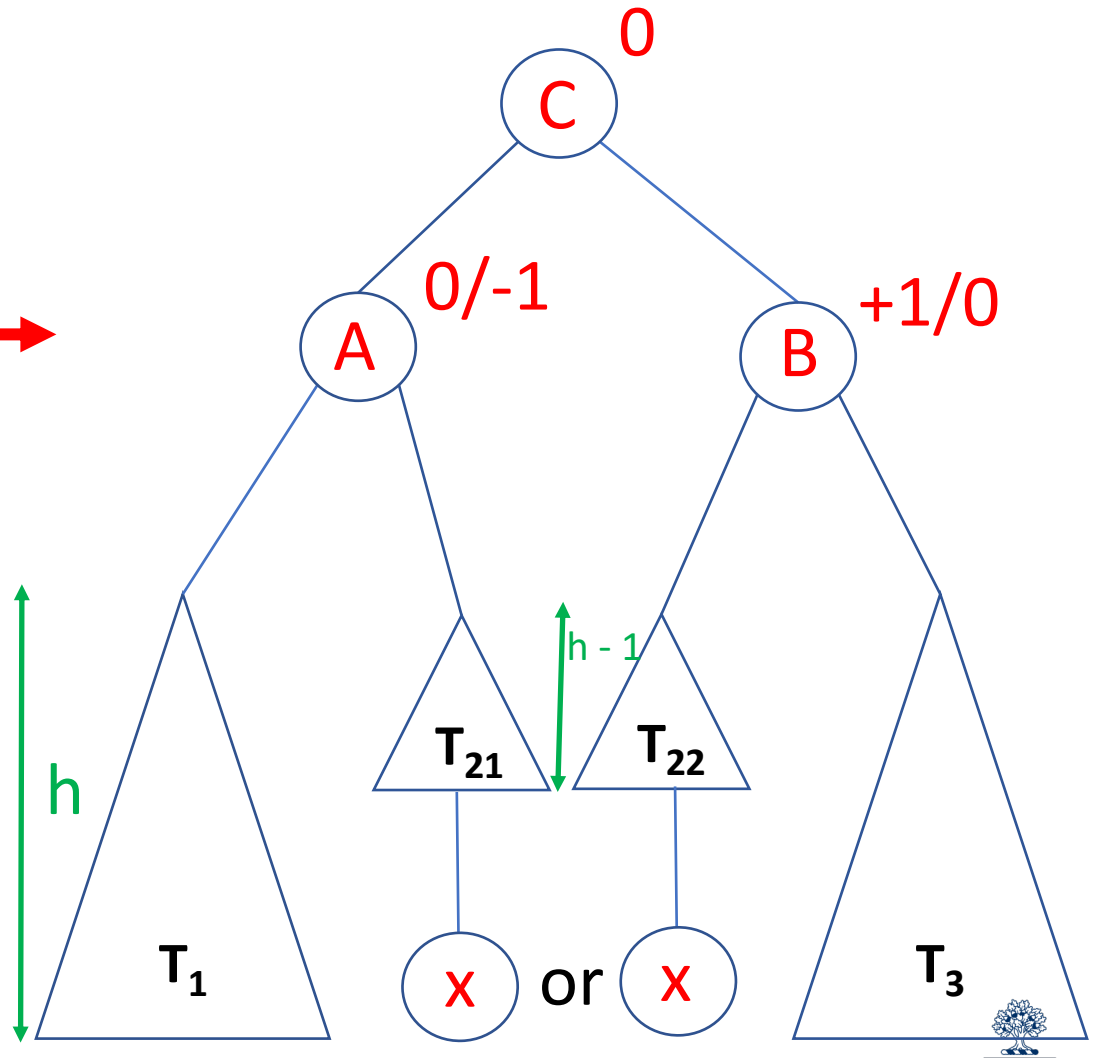


Rotation: (3) Bonus: height of this subtree is the same (h + 2) both before and after the insertion of x

Case 1 (b) : h > -1

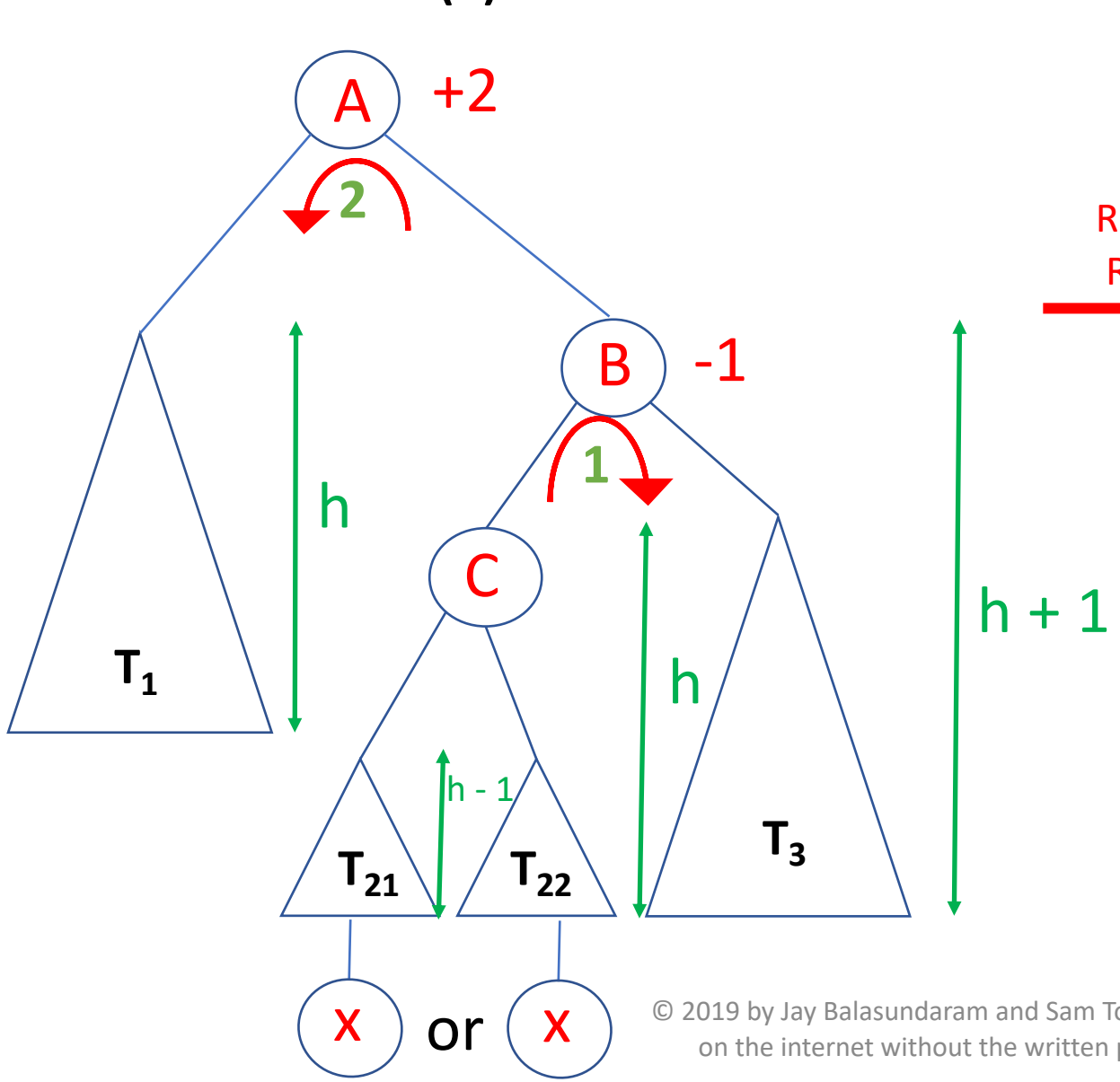


Double Right-Left Rotation

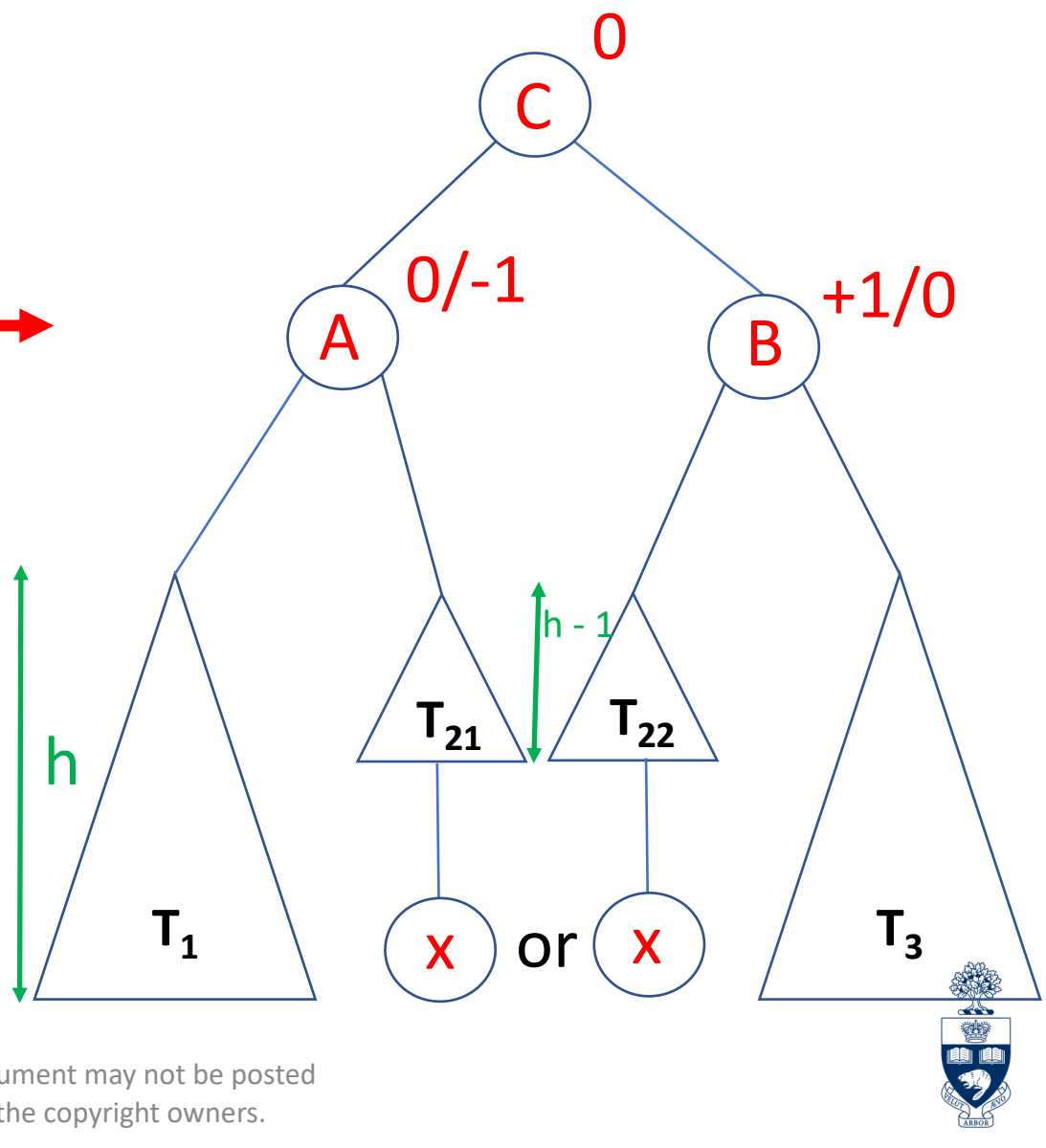


Rotation: (3) Bonus ⟹ insertion algorithm is done

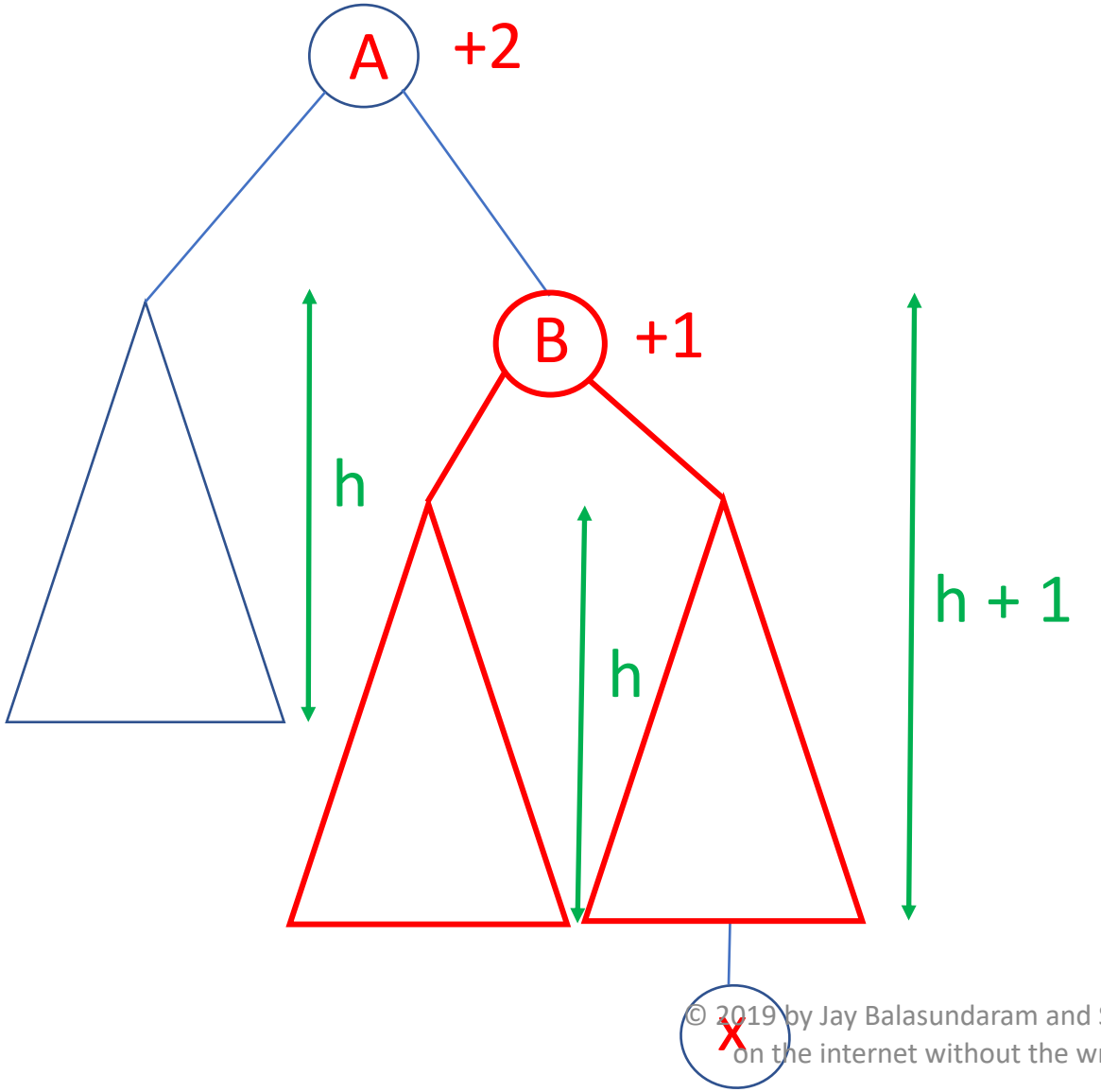
Case 1 (b) : h > -1



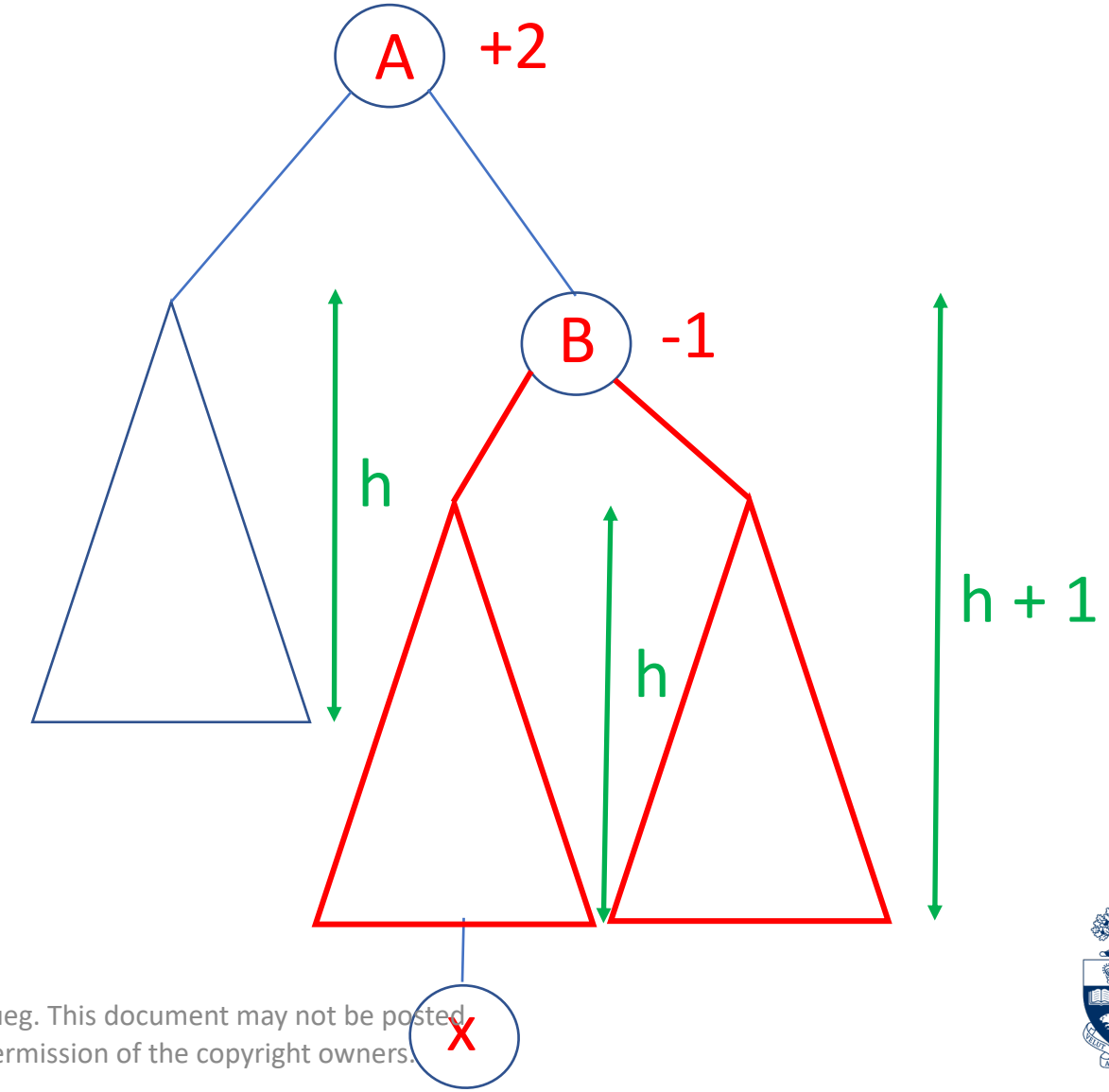
Double Right-Left Rotation



Case 1 (a) ✓



Case 1 (b) ✓



Insert(T, x) :

Insert(T, x) :

- Insert x into T as in any BST:

Insert(T, x) :

- Insert x into T as in any BST:
  - x is now a leaf

Insert(T, x) :

- Insert x into T as in any BST:
  - x is now a leaf
  - Set BF(x) to 0

Insert(T, x) :

- Insert x into T as in any BST:
  - x is now a leaf
  - Set BF(x) to 0
- Go up from x to the root and for each node v in this path

Insert(T, x) :

- Insert x into T as in any BST:
  - x is now a leaf
  - Set BF(x) to 0
- Go up from x to the root and for each node v in this path
  - Adjust BF(v):

Insert(T, x) :

- Insert x into T as in any BST:
  - x is now a leaf
  - Set BF(x) to 0
- Go up from x to the root and for each node v in this path
  - Adjust BF(v):

  - Rebalance if necessary:

Insert(T, x) :

- Insert x into T as in any BST:
  - x is now a leaf
  - Set BF(x) to 0
- Go up from x to the root and for each node v in this path
  - Adjust BF(v):
    if x is in right subtree of v :

  - Rebalance if necessary:

Insert(T, x) :

- Insert x into T as in any BST:
  - x is now a leaf
  - Set BF(x) to 0
- Go up from x to the root and for each node v in this path
  - Adjust BF(v):

    if x is in right subtree of v : Increment BF(v)

  - Rebalance if necessary:

Insert(T, x) :

- Insert x into T as in any BST:
  - x is now a leaf
  - Set BF(x) to 0
- Go up from x to the root and for each node v in this path
  - Adjust BF(v):
    if x is in right subtree of v : Increment BF(v)
    if x is in left subtree of v :
  - Rebalance if necessary:

Insert(T, x) :

- Insert x into T as in any BST:
  - x is now a leaf
  - Set BF(x) to 0
- Go up from x to the root and for each node v in this path
  - Adjust BF(v):

    if x is in right subtree of v : Increment BF(v)

    if x is in left subtree of v : Decrement BF(v)

  - Rebalance if necessary:

Insert(T, x) :

- Insert x into T as in any BST:
  - x is now a leaf
  - Set BF(x) to 0
- Go up from x to the root and for each node v in this path
  - Adjust BF(v):
    if x is in right subtree of v : Increment BF(v)
    if x is in left subtree of v : Decrement BF(v)
    if BF(v) = 0 :
  - Rebalance if necessary:

Insert(T, x) :

- Insert x into T as in any BST:
  - x is now a leaf
  - Set BF(x) to 0
- Go up from x to the root and for each node v in this path
  - Adjust BF(v):
    if x is in right subtree of v : Increment BF(v)
    if x is in left subtree of v : Decrement BF(v)
    if BF(v) = 0 : Stop
  - Rebalance if necessary:

Insert(T, x) :

- Insert x into T as in any BST:
  - x is now a leaf
  - Set BF(x) to 0
- Go up from x to the root and for each node v in this path
  - Adjust BF(v):
    if x is in right subtree of v : Increment BF(v)
    if x is in left subtree of v : Decrement BF(v)
    if BF(v) = 0 : Stop
  - Rebalance if necessary:
    if BF(v) = + 2

Insert(T, x) :

- Insert x into T as in any BST:
  - x is now a leaf
  - Set BF(x) to 0
- Go up from x to the root and for each node v in this path
  - Adjust BF(v):
    - if x is in right subtree of v : Increment BF(v)
    - if x is in left subtree of v : Decrement BF(v)
    - if BF(v) = 0 : Stop
  - Rebalance if necessary:
    - if BF(v) = + 2
      - if BF(v.right) = +1

Insert(T, x) :

- Insert x into T as in any BST:
  - x is now a leaf
  - Set BF(x) to 0
- Go up from x to the root and for each node v in this path
  - Adjust BF(v):
    - if x is in right subtree of v : Increment BF(v)
    - if x is in left subtree of v : Decrement BF(v)
    - if BF(v) = 0 : Stop
  - Rebalance if necessary:
    - if BF(v) = + 2
      - if BF(v.right) = +1
        - Do Left Rotation, update BFs of rotated nodes, and stop

Insert(T, x) :

- Insert x into T as in any BST:
  - x is now a leaf
  - Set BF(x) to 0
- Go up from x to the root and for each node v in this path
  - Adjust BF(v):
    - if x is in right subtree of v : Increment BF(v)
    - if x is in left subtree of v : Decrement BF(v)
    - if BF(v) = 0 : Stop
  - Rebalance if necessary:
    - if BF(v) = + 2
      - if BF(v.right) = +1
        - Do Left Rotation, update BFs of rotated nodes, and stop
      - if BF(v.right) = -1

Insert(T, x) :

- Insert x into T as in any BST:
  - x is now a leaf
  - Set BF(x) to 0
- Go up from x to the root and for each node v in this path
  - Adjust BF(v):
    - if x is in right subtree of v : Increment BF(v)
    - if x is in left subtree of v : Decrement BF(v)
    - if BF(v) = 0 : Stop
  - Rebalance if necessary:
    - if BF(v) = + 2
      - if BF(v.right) = +1
        - Do Left Rotation, update BFs of rotated nodes, and stop
      - if BF(v.right) = -1
        - Do Right-Left Rotation, update BFs of rotated nodes, and stop

Insert(T, x) :

- Insert x into T as in any BST:
  - x is now a leaf
  - Set BF(x) to 0
- Go up from x to the root and for each node v in this path
  - Adjust BF(v):
    - if x is in right subtree of v : Increment BF(v)
    - if x is in left subtree of v : Decrement BF(v)
    - if BF(v) = 0 : Stop
  - Rebalance if necessary:
    - if BF(v) = + 2
      - if BF(v.right) = +1
        - Do Left Rotation, update BFs of rotated nodes, and stop
      - if BF(v.right) = -1
        - Do Right-Left Rotation, update BFs of rotated nodes, and stop
    - if BF(v) = -2

Insert(T, x) :

- Insert x into T as in any BST:
  - x is now a leaf
  - Set BF(x) to 0
- Go up from x to the root and for each node v in this path
  - Adjust BF(v):
    - if x is in right subtree of v : Increment BF(v)
    - if x is in left subtree of v : Decrement BF(v)
    - if BF(v) = 0 : Stop
  - Rebalance if necessary:
    - if BF(v) = + 2
      - if BF(v.right) = +1
        - Do Left Rotation, update BFs of rotated nodes, and stop
      - if BF(v.right) = -1
        - Do Right-Left Rotation, update BFs of rotated nodes, and stop
    - if BF(v) = -2
      - Symmetric to above case