

AVL trees

Today

- AVL Deletes and rotations, then testing your knowledge of these concepts!
- Before I get into details, I want to show you some animated operations in an AVL tree.
- I think it's important to just get the gears turning in your mind.
- We'll look at some animations *again* after we study (some) details.
- Interactive web applet

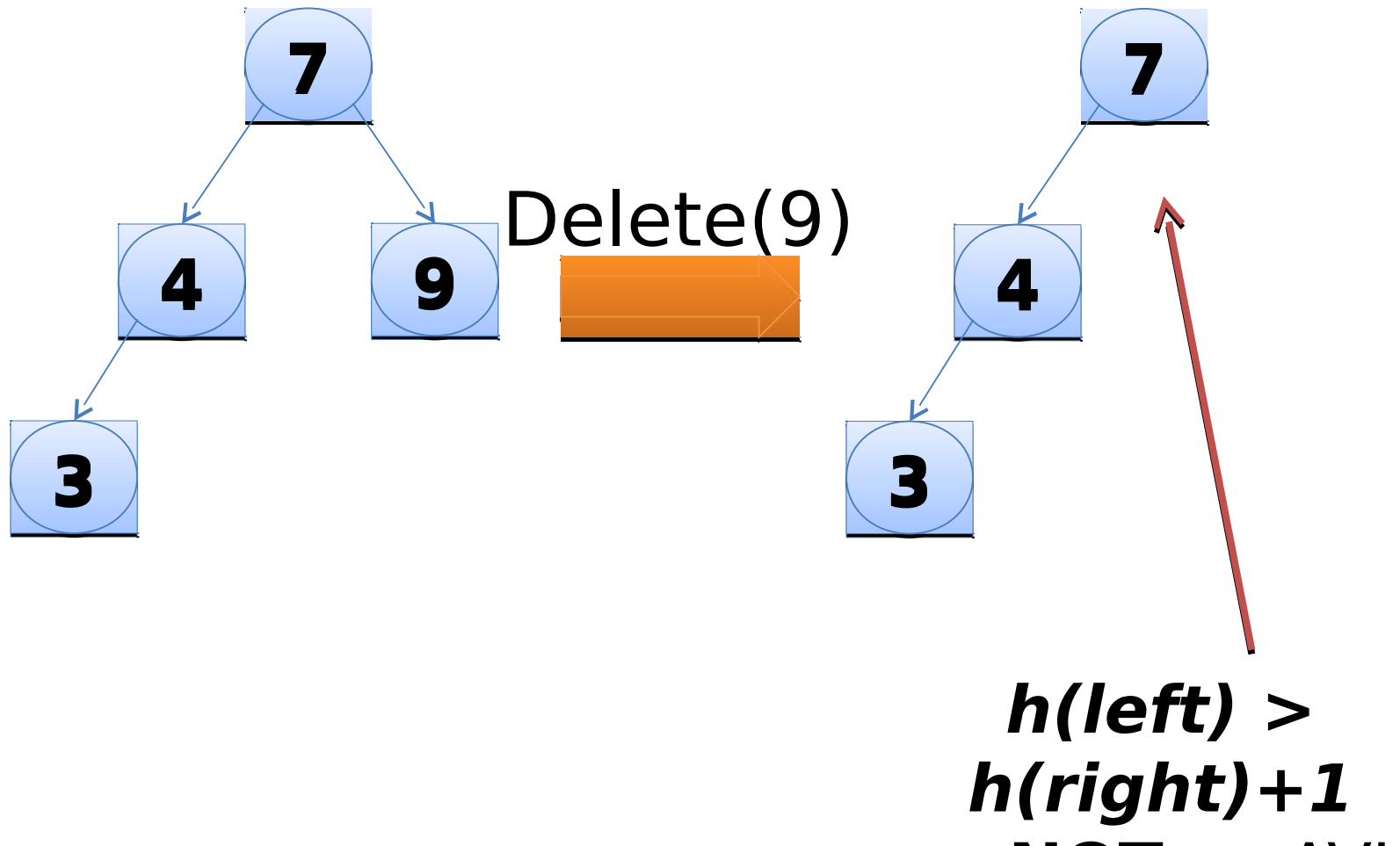
AVL tree

- Is a binary search tree
- Has an additional ***height constraint***:
 - For each node x in the tree, $\text{Height}(x.\text{left})$ differs from $\text{Height}(x.\text{right})$ by at most 1
- I promise:
 - If you satisfy the ***height constraint***, then the **height of the tree is $O(\lg n)$** .
 - (Proof is easy, but no time! =])

AVL tree

- To be an AVL tree, must **always**:
 - (1) Be a ***binary search tree***
 - (2) Satisfy the ***height constraint***
- Suppose we start with an AVL tree, then delete as if we're in a regular BST.
- Will the tree be an AVL tree after the delete?
 - (1) It will still be a BST... that's one part.
 - (2) Will it satisfy the ***height constraint***?
- (Not covering insert, since you already did in class)

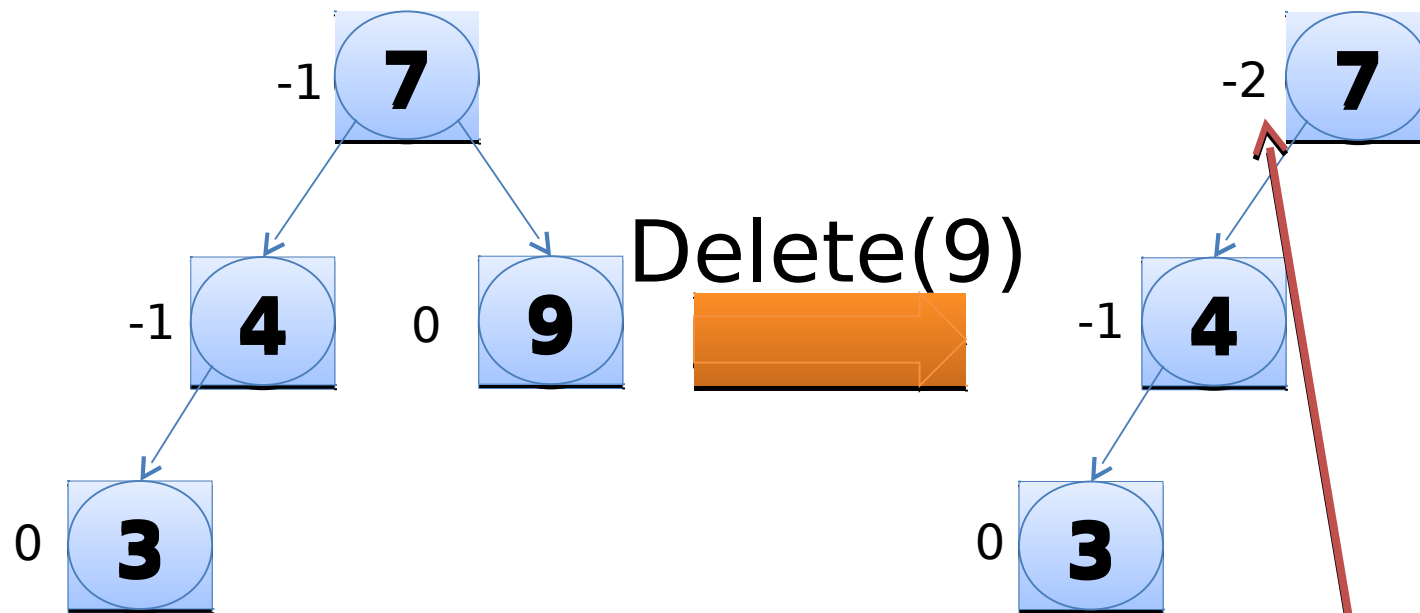
BST Delete breaks an AVL tree



Balance factors

- To check the **balance constraint**, we have to know the ***height h*** of each node
- Or do we?
- In fact, we can store ***balance factors*** instead.
- The balance factor $bf(x) = h(x.right) - h(x.left)$
 - $bf(x)$ values -1, 0, and 1 are allowed.
 - If $bf(x) < -1$ or $bf(x) > 1$ then tree is **NOT AVL**

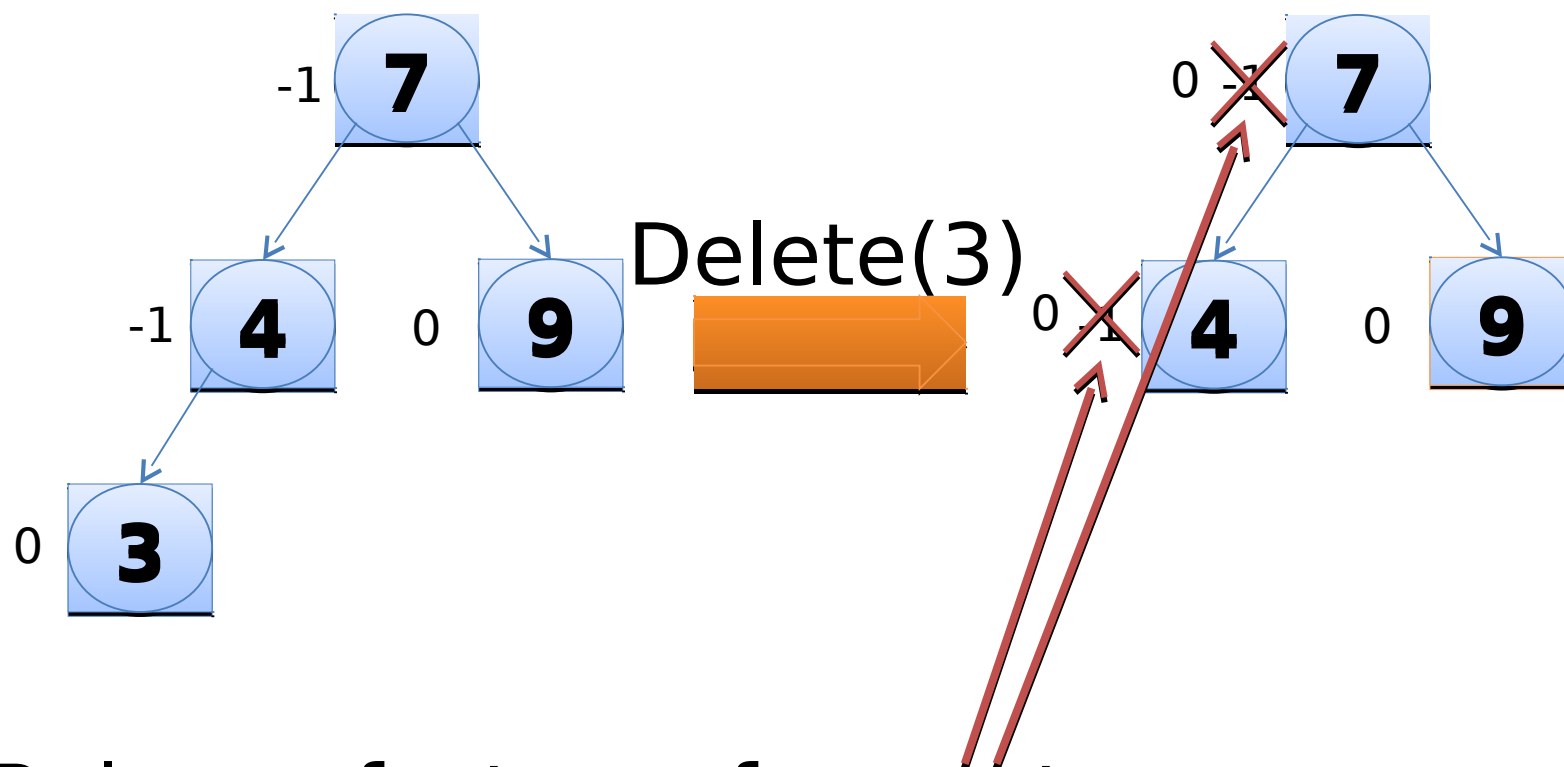
Same example with **bf(x)**,
not h(x)



bf < -1

so **NOT** an AVL tree

What else can BST Delete break?



- Balance factors of ancestors...

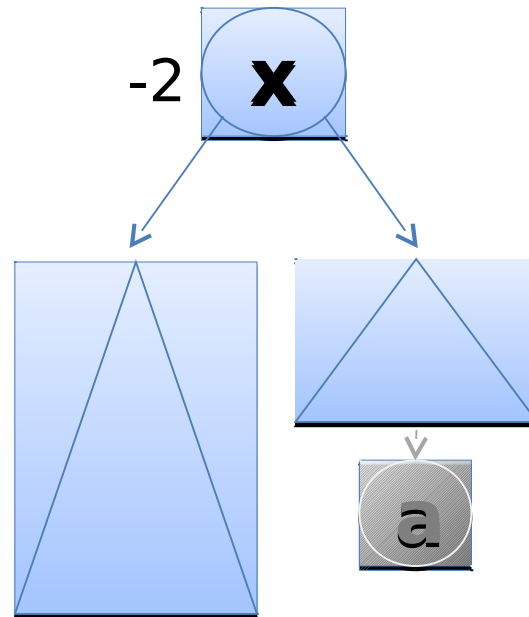
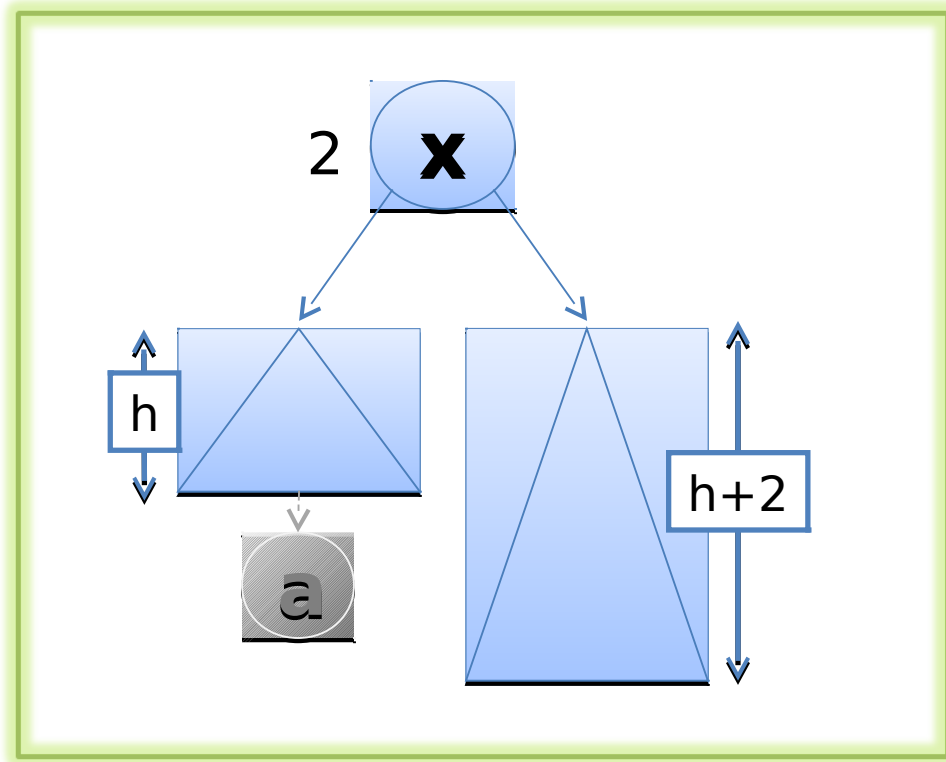
Need a new Delete algorithm

- We are starting to see what our delete algorithm must look like.
- **Goal:** if tree is AVL before Delete, then tree is AVL after Delete.
- **Step 1:** do BST delete.
 - This maintains the *BST property*, but can BREAK the *balance factors of ancestors!*
- **Step 2:** fix the balance constraint.
 - Do something that maintains the BST property, but fixes any balance factors that are < -1 or > 1 .

Bad balance factors

- Start with an AVL tree, then do a BST Delete.
- What bad values can $bf(x)$ take on?
 - Delete can reduce a subtree's height by 1.
 - So, it might increase or decrease $h(x.right) - h(x.left)$ by 1.
 - So, $bf(x)$ might increase or decrease by 1.
 - This means:
 - if $bf(x) = 1$ before Delete, it might become 2. **2 cases**
 - If $bf(x) = -1$ before Delete, it might become -2. **2 cases**
 - If $bf(x) = 0$ before Delete, then it is still -1, 0 or 1. **OK.**

Problematic cases for Delete(a)



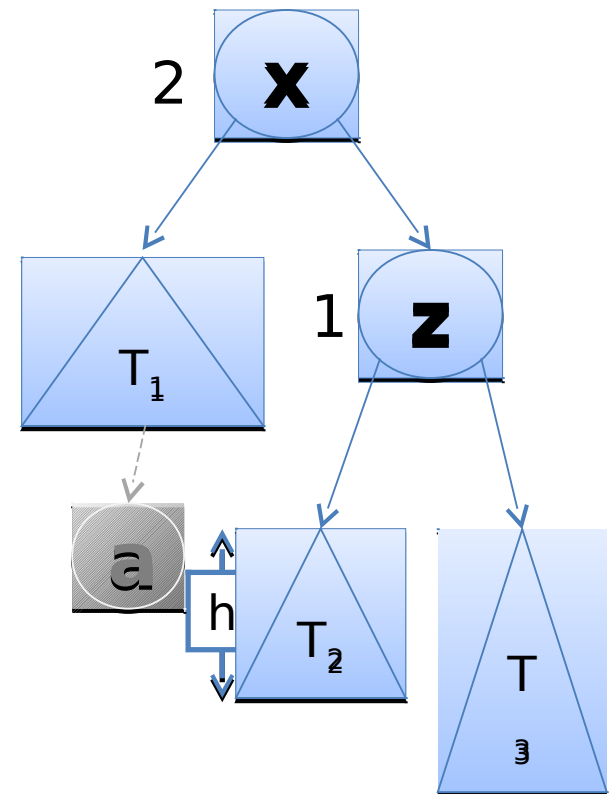
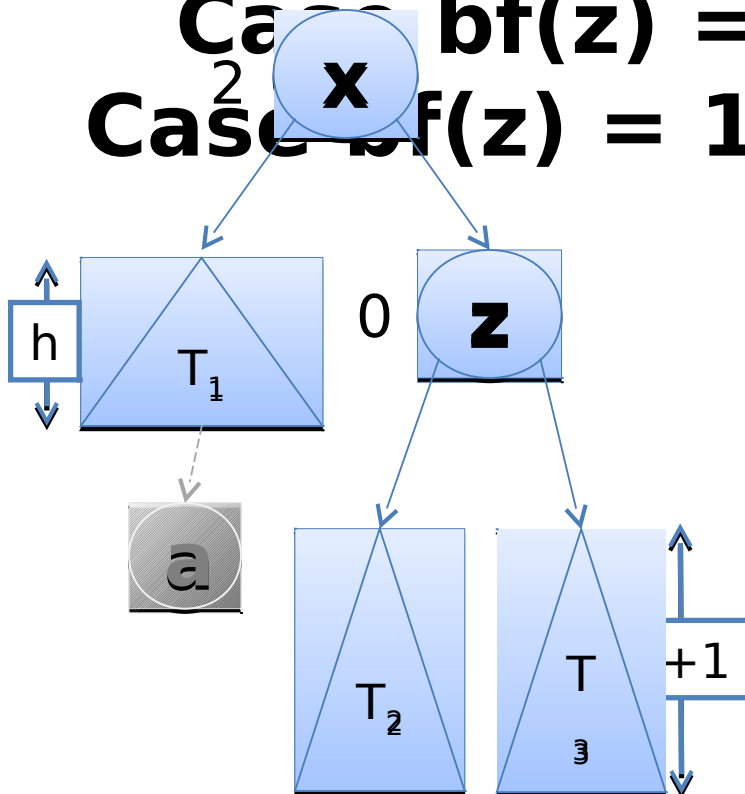
- $bf(x) = -2$ is just **symmetric** to $bf(x) = 2$.
- So, we just look at $bf(x) = 2$.

Delete(a): 3 subcases for $bf(x)=2$

- Since tree was AVL before, **$bf(z) = -1, 0$ or 1**

Case $bf(z) = 0$

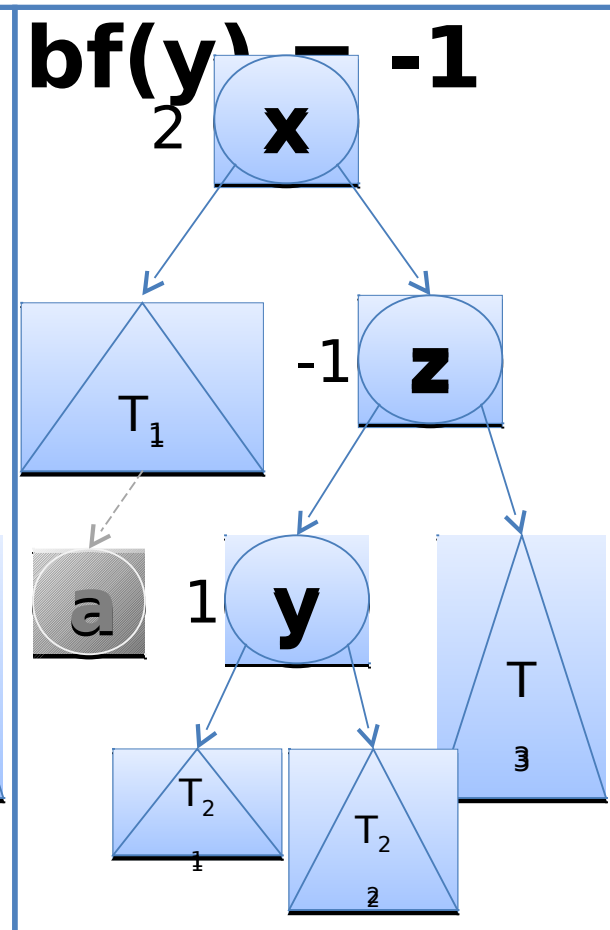
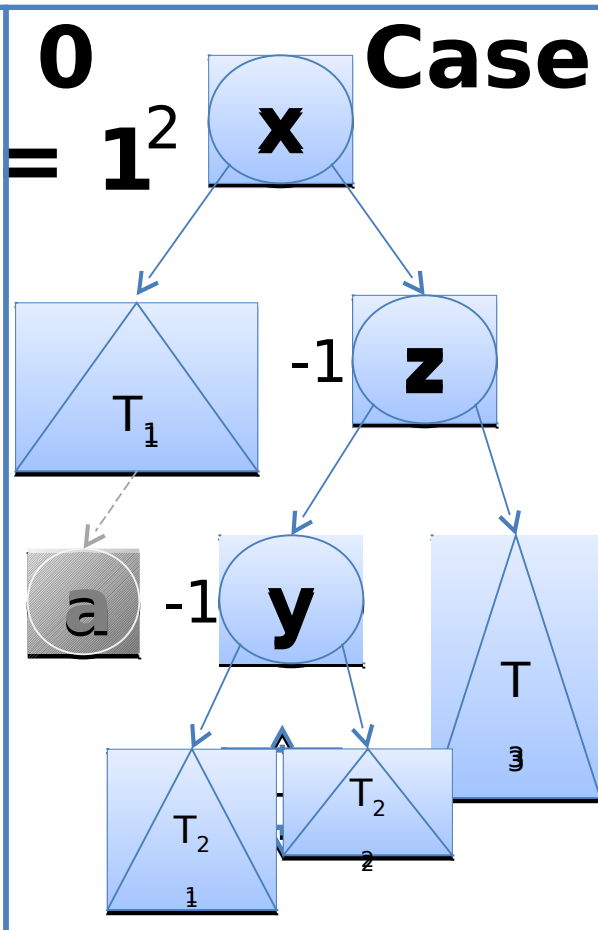
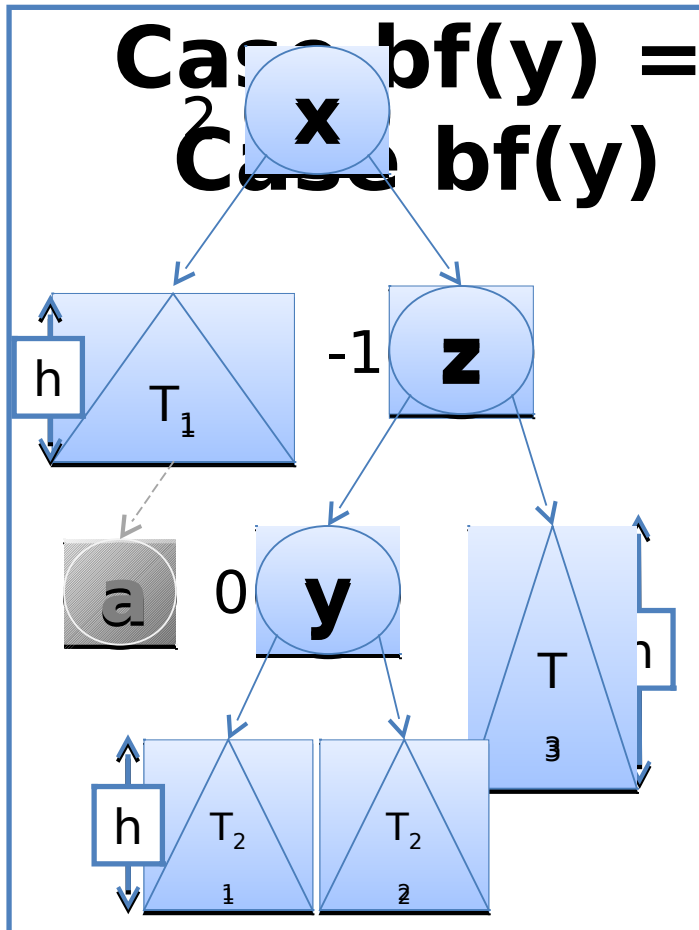
Case $bf(z) = 1$



Delete(a): final subcase for $bf(x)=2$

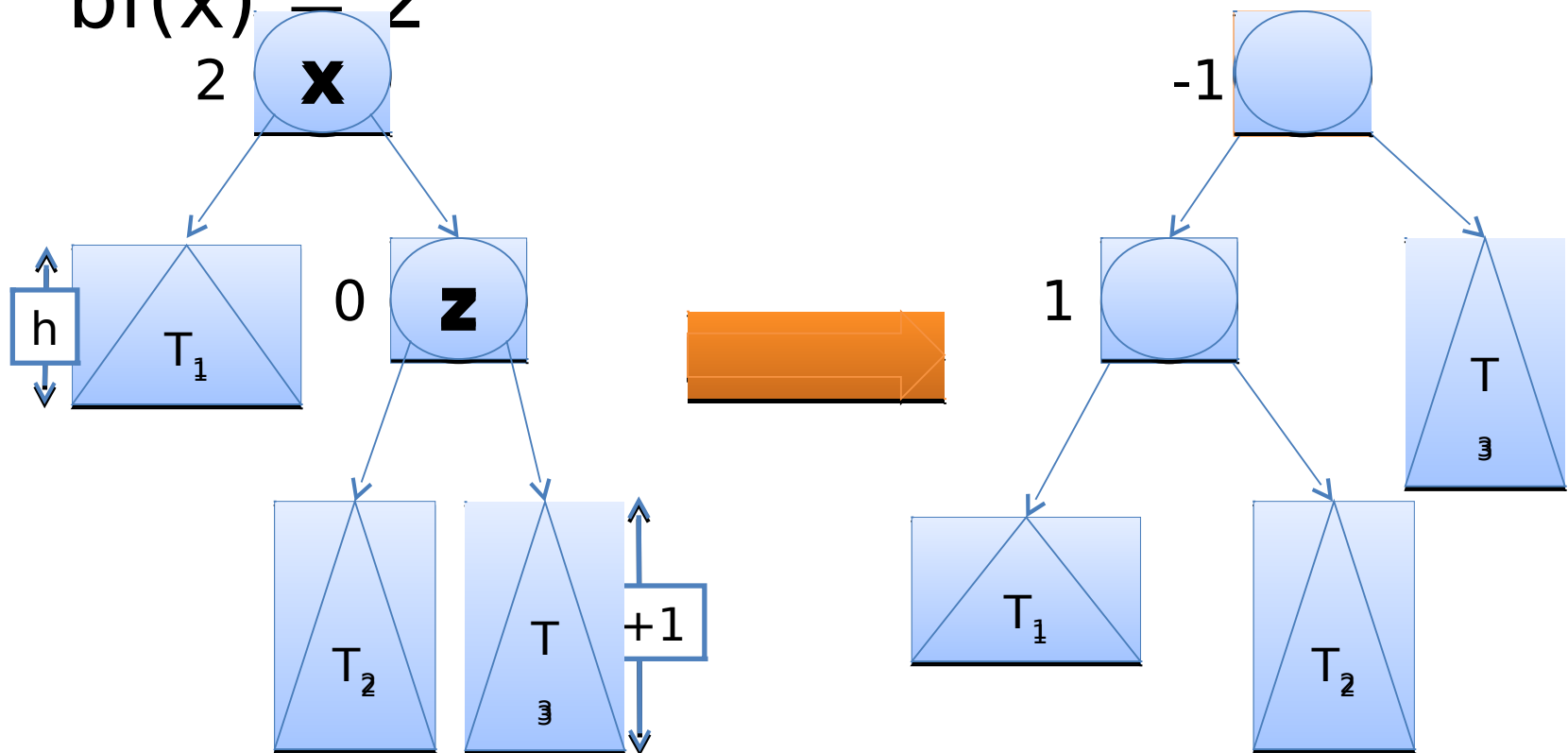
Case $bf(z) = -1$: we have 3 subcases.
(More details)

Case $bf(y) = 0$
Case $bf(y) = 1$



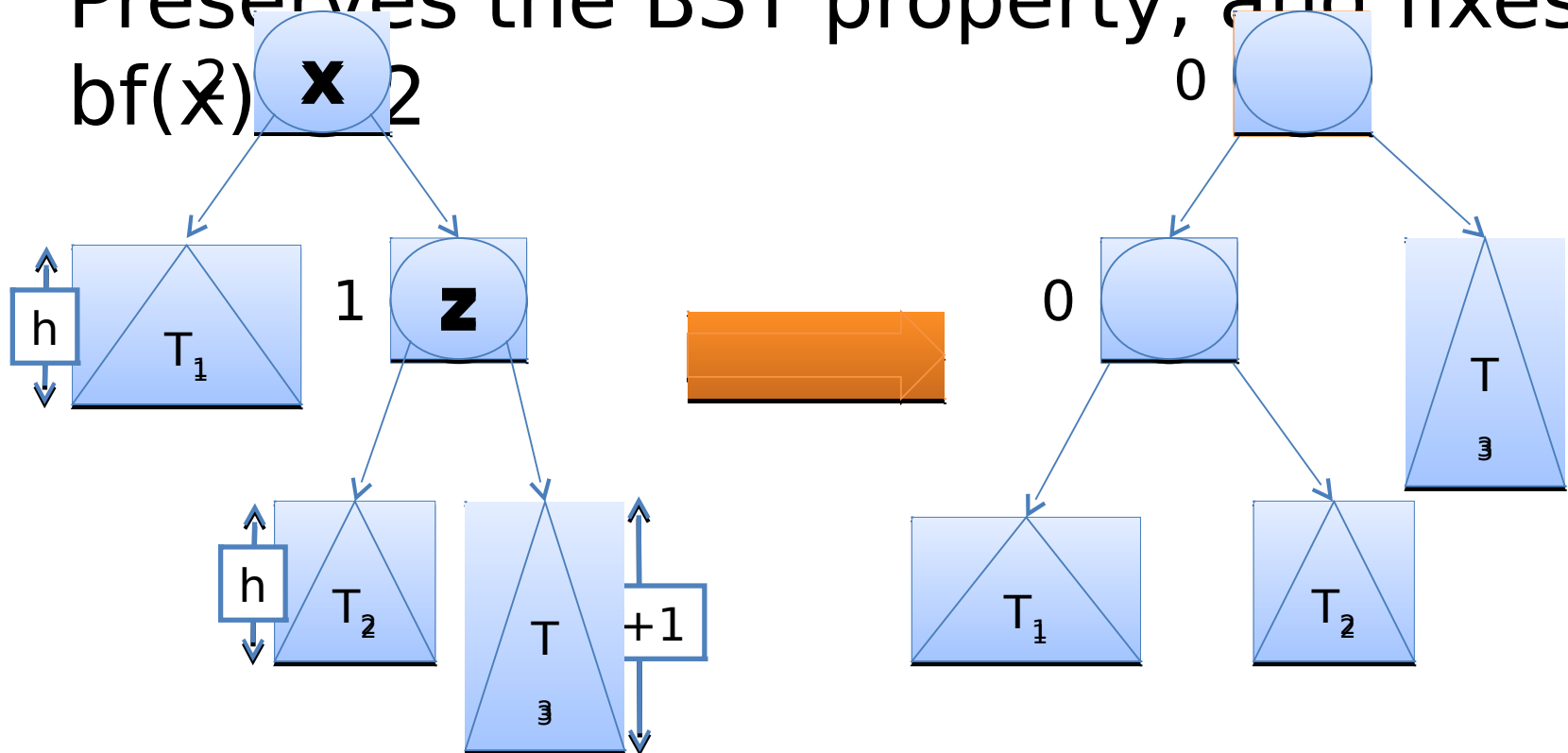
Fixing case $\text{bf}(x) = 2, \text{bf}(z) = 0$

- We do a ***single left rotation***
- Preserves the BST property, and fixes $\text{bf}(x) = 2$



Fixing case $bf(x) = 2, bf(z) = 1$

- We do a ***single left rotation*** (same as last case)
- Preserves the BST property, and fixes $bf(x)$



Interactive AVL Deletes

- Interactive web applet
- Video of this applet being used to show most cases for insert / delete