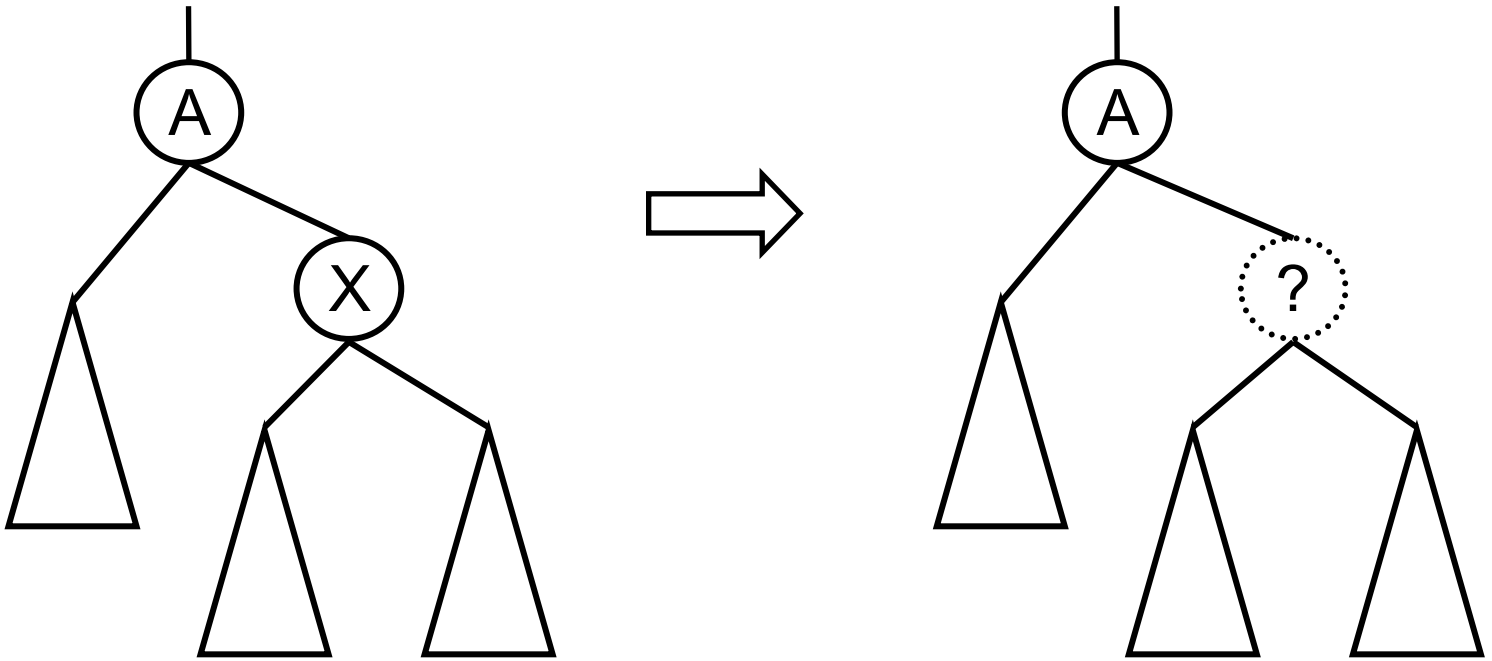
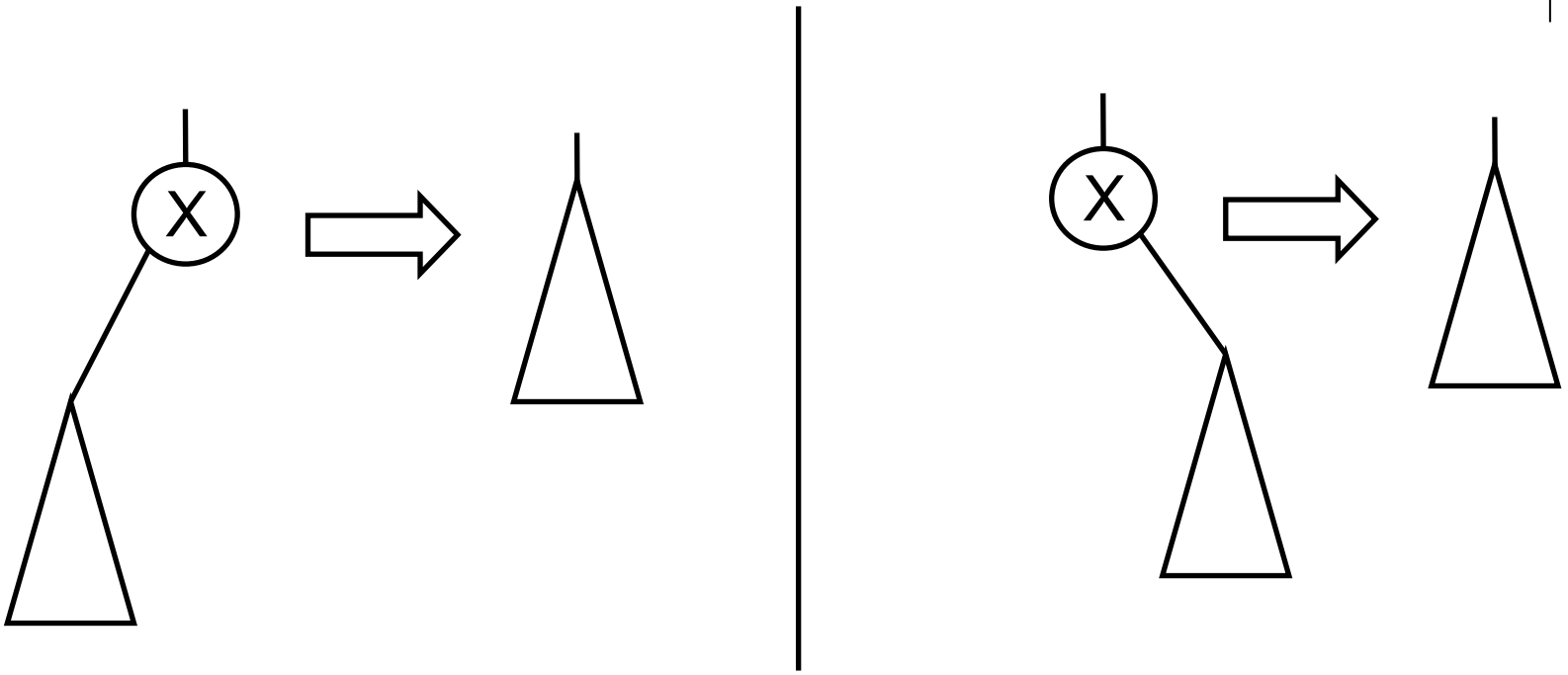
Deleting an element from an ordered binary tree



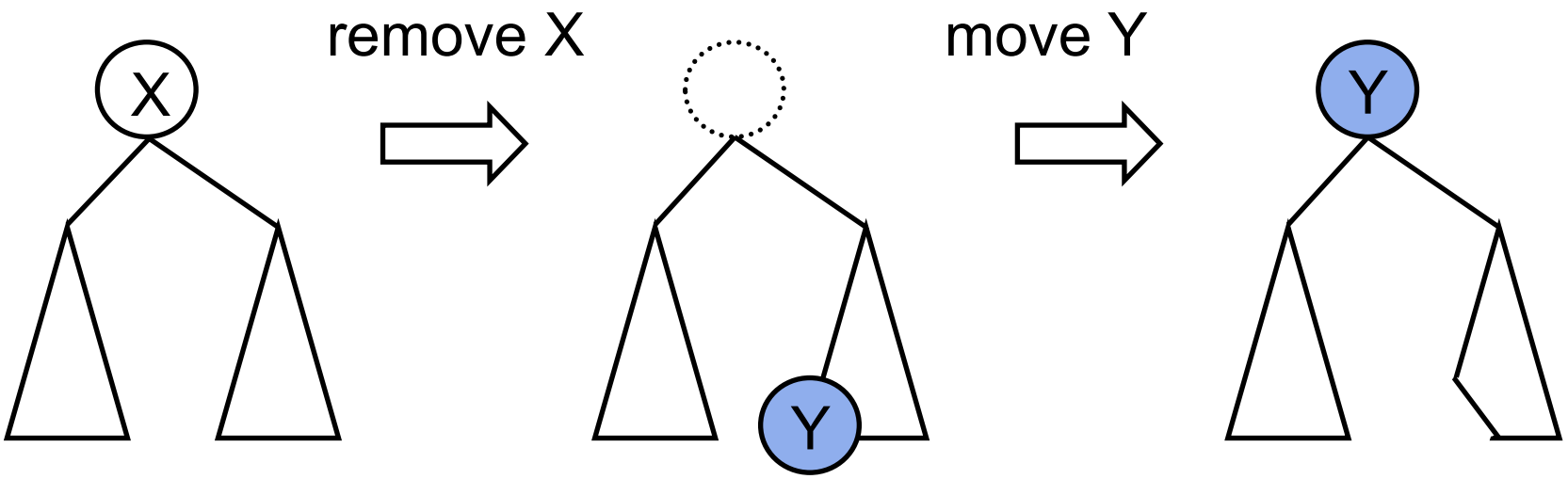
The problem is to repair the tree after X disappears

Deleting an element when one subtree is empty



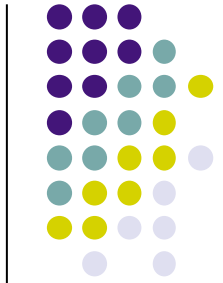
It’s easy when one of the subtrees is empty: just replace the tree by the other subtree

Deleting an element when both subtrees are not empty

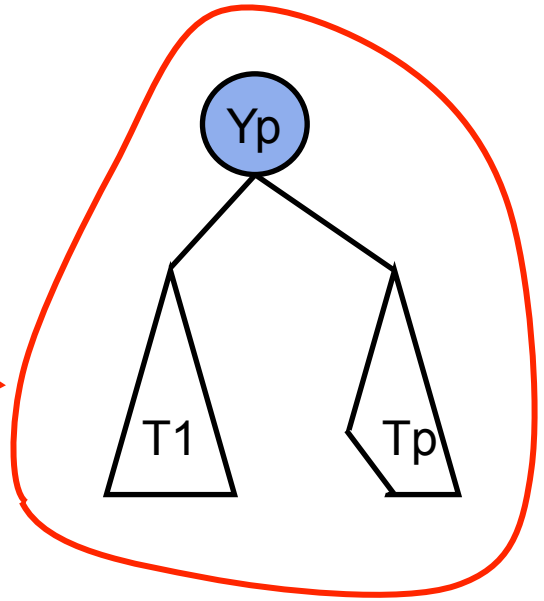


The idea is to fill the ”hole” that appears after X is removed. We can put there the smallest element in the right subtree, namely Y.

We need a new function: RemoveSmallest

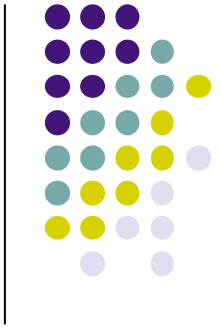


```
fun {Delete K T}
   case T
   of leaf then leaf
   [] tree(key:X value:V left:T1 right:T2) andthen K==X then
         case {RemoveSmallest T2}
         of none then T1
         [] triple(Tp Yp Vp) then
               tree(key:Yp value:Vp left:T1 right:Tp)
         end
   [] … end
end
```



- RemoveSmallest takes a tree and returns three values:
  - The new subtree Tp without the smallest element
  - The smallest element's key Yp
  - The smallest element's value Vp
- With these three values we can build the new tree where Yp is the root and Tp is the new right subtree

Recursive definition of RemoveSmallest



```
fun {RemoveSmallest T}
   case T
   of leaf then none
   [] tree(key:X value:V left:T1 right:T2) then
      case {RemoveSmallest T1}
      of none then triple(T2 X V)
      [] triple(Tp Xp Vp) then
         triple(tree(key:X value:V left:Tp right:T2) Xp Vp)
      end
   end
end
```

To understand this definition, draw diagrams with trees!

- RemoveSmallest takes a tree T and returns:
  - The atom none when T is empty
  - The record triple(Tp Xp Vp) when T is not empty