UNIVERSITAT DE BARCELONA

Estructura de datos

Árboles de Búsqueda Binaria

Santi Seguí | 2014-15

Indicie

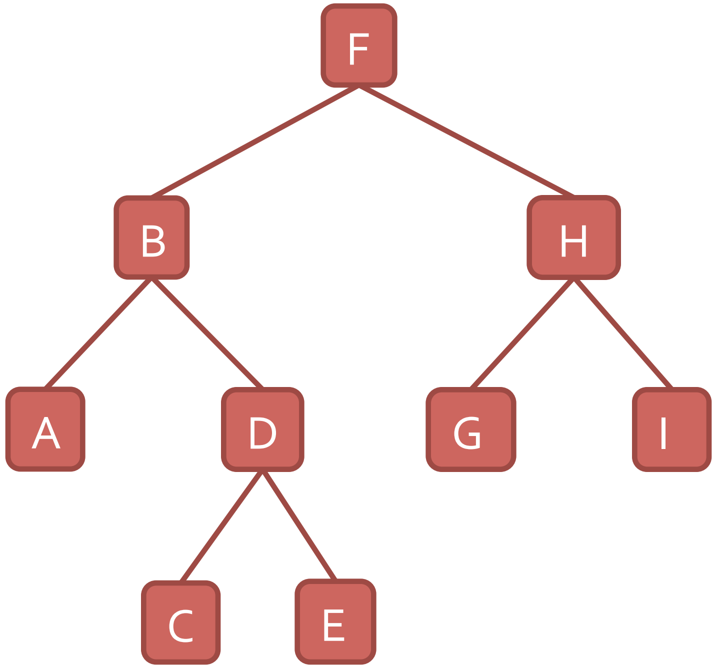
- Árboles de Búsqueda Binaria (ABB)
- Búsqueda en ABB
- Añadir en ABBs
- Eliminar en ABBs
- Análisis de los ABBs
- Balencear ABBs

Operaciones sobre árboles

	Representacion Vectorial	Representacion con nodos
Size		
Buscar		
Remplazar		
Eliminar		
Insertar		

Árbol de Búsqueda Binaria

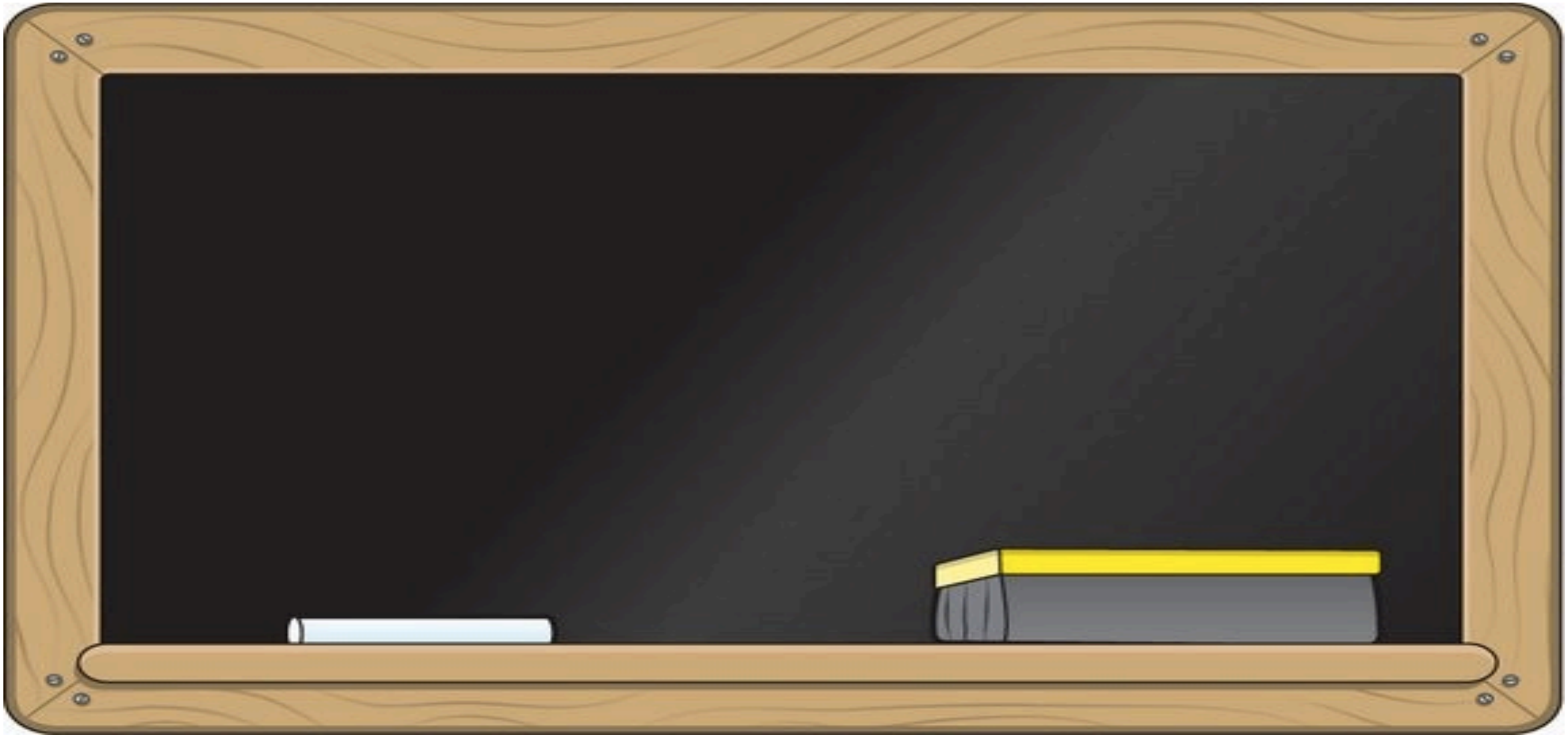
- Los árboles de búsqueda binarios (ABBs) son árboles binarios con unas propiedades especiales. Para cada nodo:
  - Todos los descendientes de su **subárbol izquierdo** tienen un valor **menor**.
  - Todos los descendientes de su **subárbol derecho** tienen un valor **mayor**.
- ¿Qué recorrido nos dará la respuesta ordenada por el valor?
  - **Recorrido In-order**



Árbol ABB

Queremos crear un ABB donde cada **nodo** corresponda a **una palabra** de la siguiente frase:

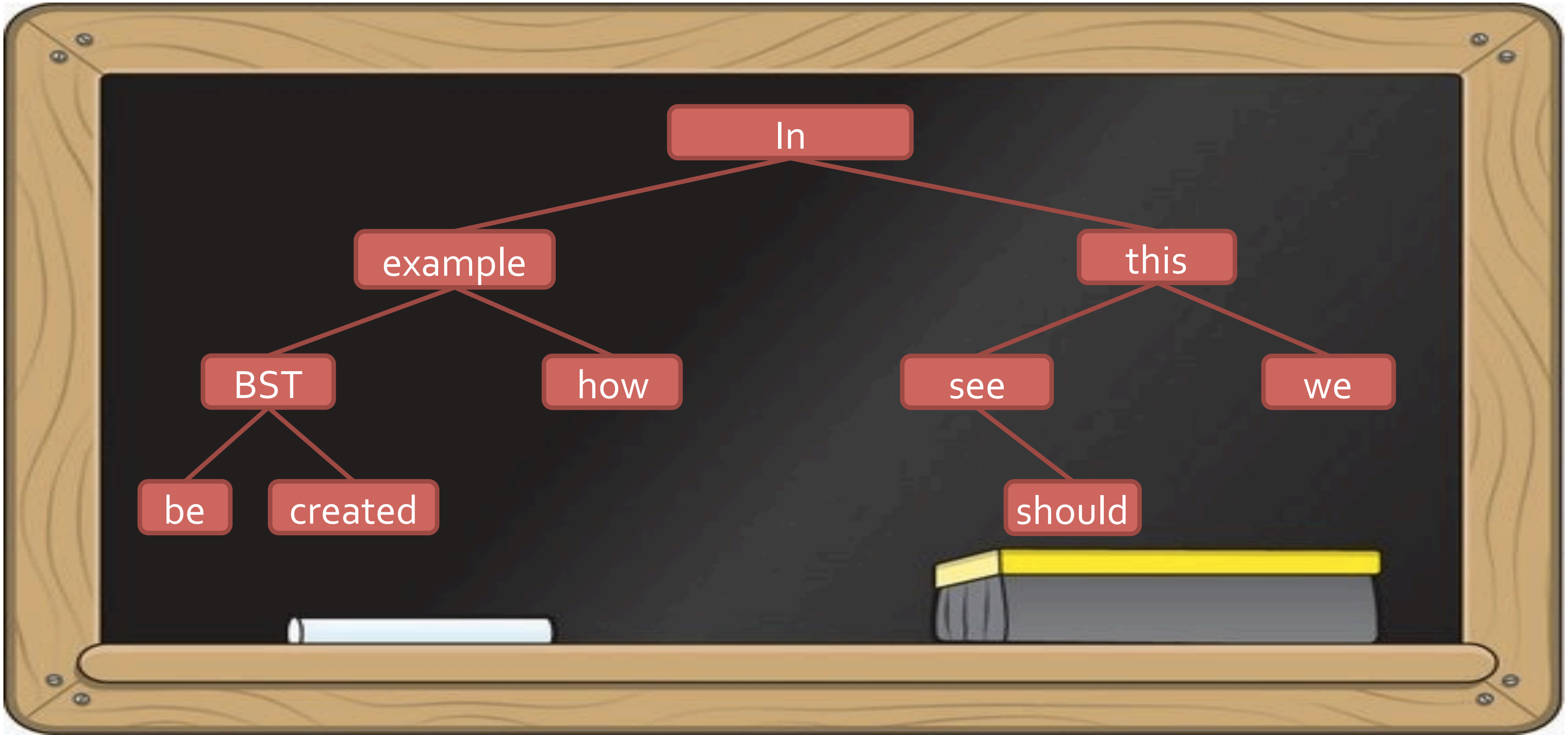
" In this example we see how BST should be created"



Árbol ABB

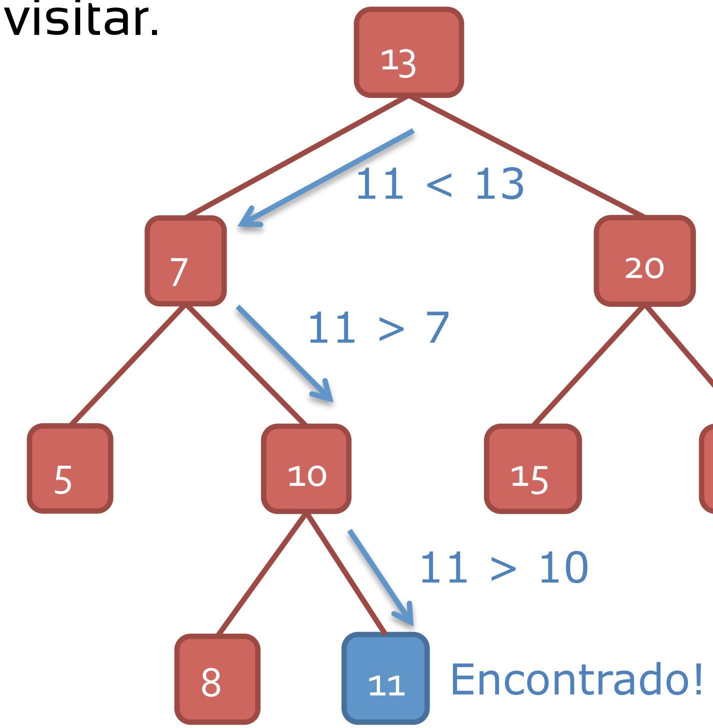
Queremos crear un ABB donde cada **nodo** corresponda a **una palabra** de la siguiente frase:

" In this example we see how BST should be created"



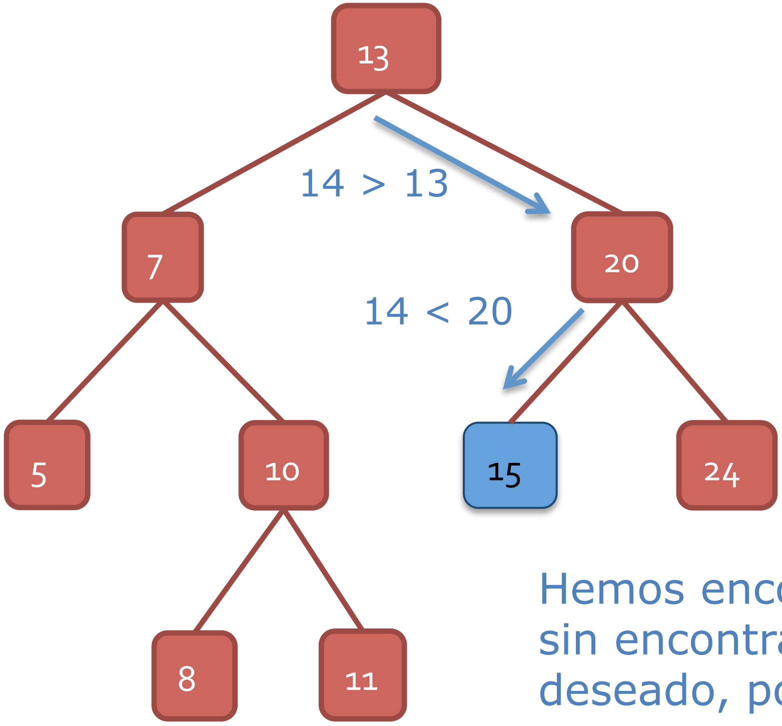
Búsqueda en ABBs

- Como podemos implementar el método **contains**() utilizado un ABB?
- Supón que queremos encontrar el nodo con **valor 11** en el siguiente árbol
- Empezando por la raíz, cada comparación no dirá que subárbol tenemos que visitar.



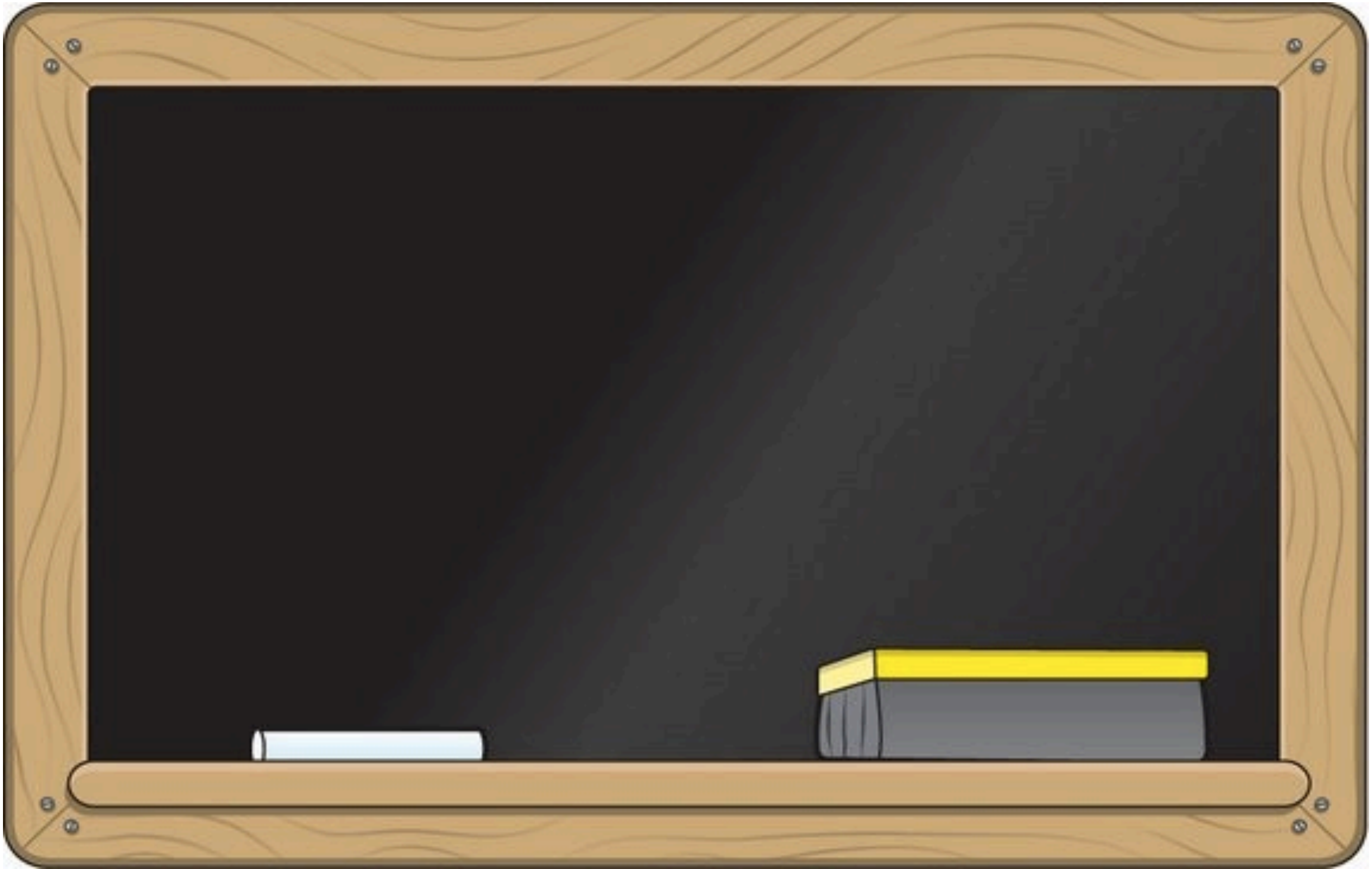
Búsqueda en ABBs (2)

- ¿Qué passa si el elemento no sé encuentra en el árbol?
- Supon que estamos buscando el número 14



Hemos encontrado una hoja sin encontrar el número deseado, por tanto el número no se encuentra en el árbol

Búsqueda en ABBs: Pseudo-Código



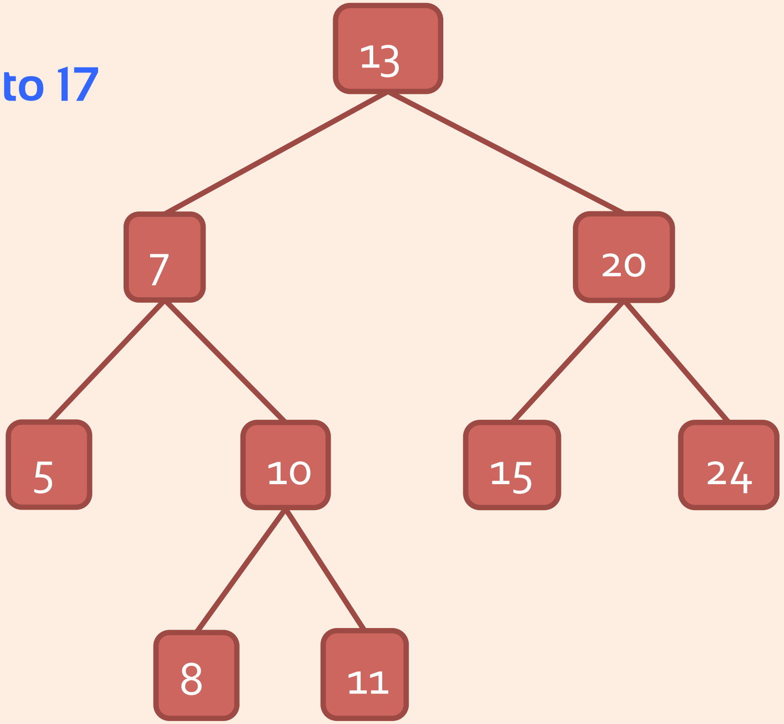
Búsqueda en ABBs: Pseudo-Código

```
function contains(node, toFind):
    // Input: node - root node of tree
    //        toFind - data of the node you're trying to find
    // Output: the node with data toFind or null if toFind is not
    //          in BST

    if node.data == toFind:
        return node

    else if toFind < node.data and node.left != null:
        return contains(node.left, toFind)

    else if toFind > node.data and node.right != null:
        return contains(node.right, toFind)

    return null
```

Añadir en un ABB

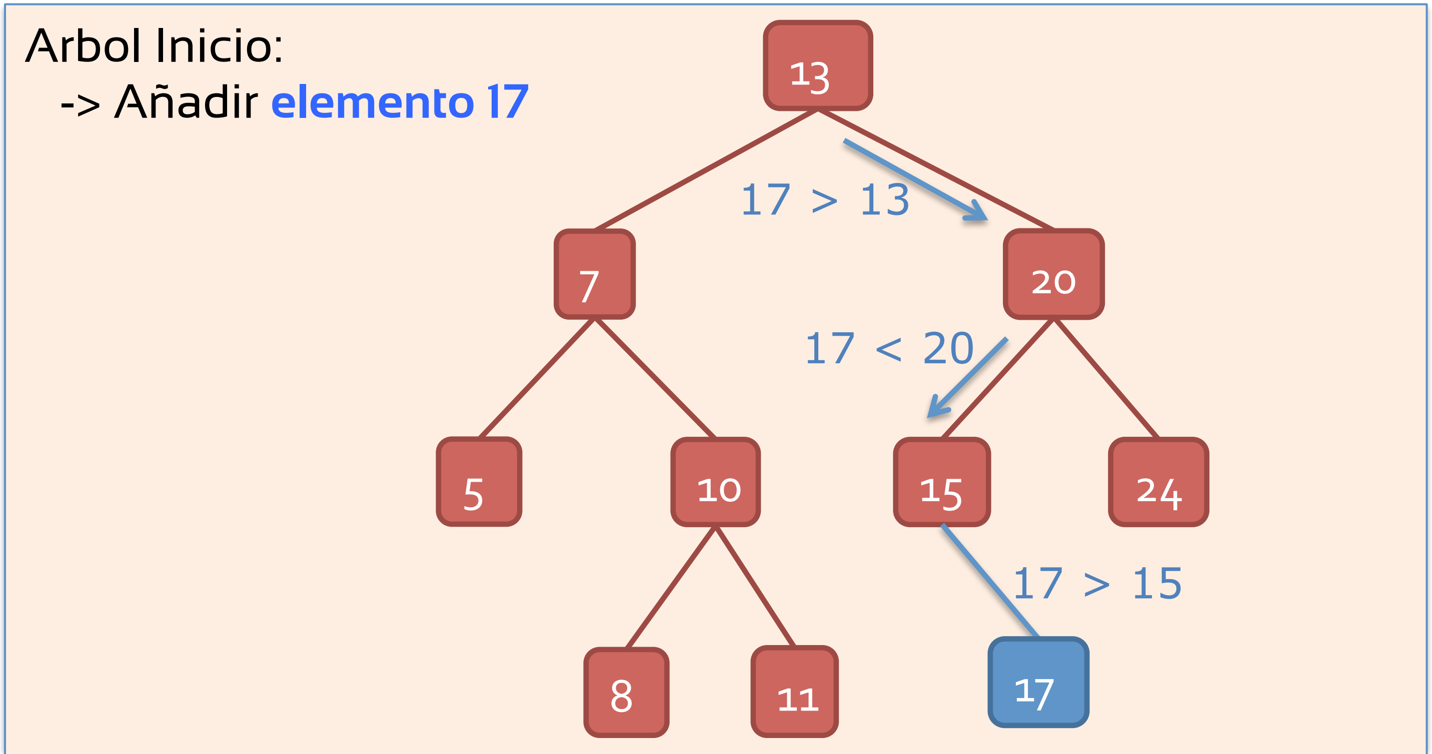
- Para añadir un elemento en el ABB, utilizamos la misma estrategia que la utilizada para buscar.
- Un nuevo ítem siempre se añade como una nueva hoja.

Arbol Inicio:
-> Añadir **elemento 17**



Añadir en un ABB

- Para añadir un elemento en el ABB, utilizamos la misma estrategia que la utilizada para buscar.
- Un nuevo ítem siempre se añade como una nueva hoja.



Método para añadir elementos en un ABB

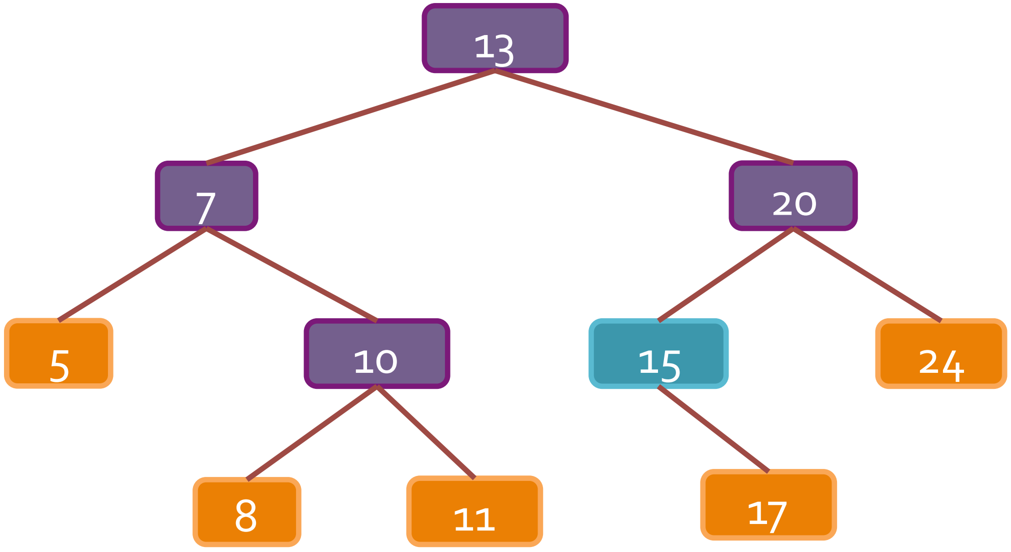
```
function insert(node, toInsert):
    // Input: node - root node of tree
    //        toInsert - data you are trying to insert
```

Método para añadir elementos en un ABB

```
function insert(node, toInsert):
   // Input: node - root node of tree
   //        toInsert - data you are trying to insert

   if node.data == toInsert: // data already in tree
      return

   if toInsert < node.data:
      if node.left == null: // add as left child
         node.addLeft(toInsert)
      else:
         insert(node.left, toInsert)
   else:
      if node.right == null: // add as right child
         node.addRight(toInsert)
      else:
         insert(node.right, toInsert)
```

Eliminación en ABB's

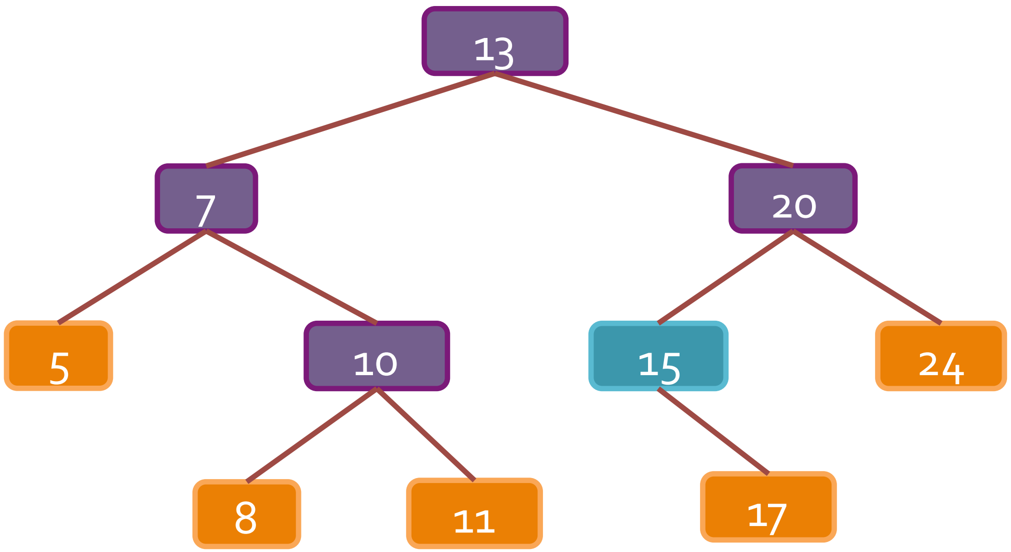
- La eliminación de elementos en ABB es un algoritmo mas complejo.
- Hay que considerar 3 casos:
  - 1) Eliminar una **hoja**. Eliminar el nodo correspondiente
  - 2) Eliminar un **nodo interno con un hijo**
  - 3) Eliminar un **nodo interno con dos hijos**



Eliminar en ABB's. Caso 2

Caso 2: Eliminar un nodo interno con un hijo.

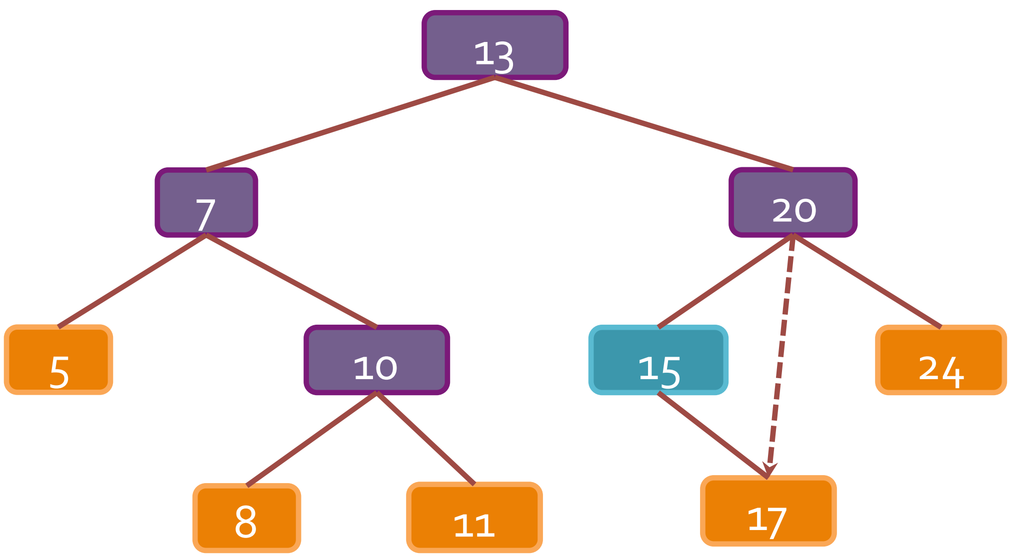
- Estrategia General:
  - Eliminar el elemento correspondiente conectado el nodo del padre con el nodo del hijo del nodo a eliminar
- Ejemplo: **remove(15)**



Eliminar en ABB's. Caso 2

Caso 2: Eliminar un nodo interno con un hijo.

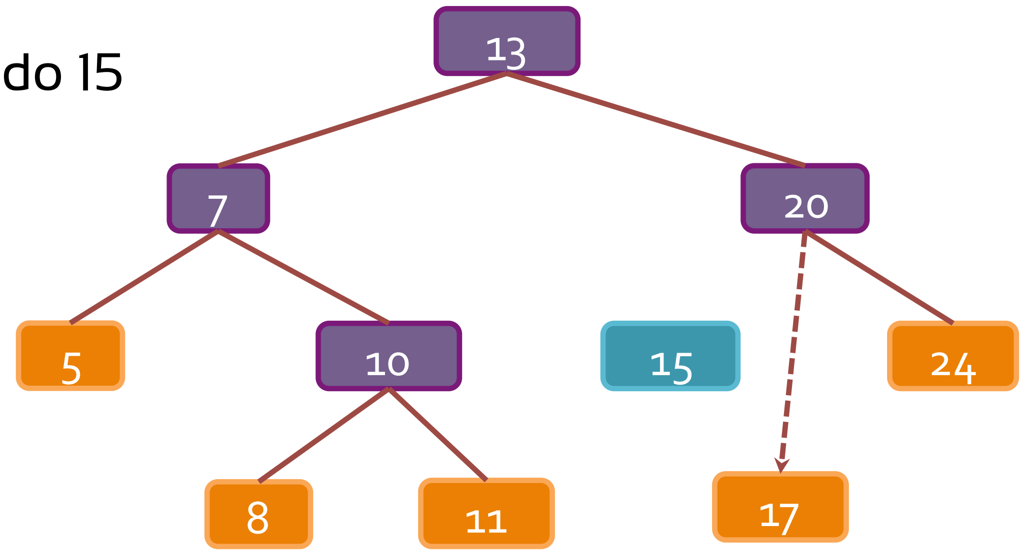
- Estrategia General:
  - Eliminar el elemento correspondiente conectado el nodo del padre con el nodo del hijo del nodo a eliminar
- Ejemplo: **remove(15)**
  - Asignamos a la variable `left` del nodo (20) padre la referencia del hijo del nodo 15



Eliminar en ABB's. Caso 2

Caso 2: Eliminar un nodo interno con un hijo.

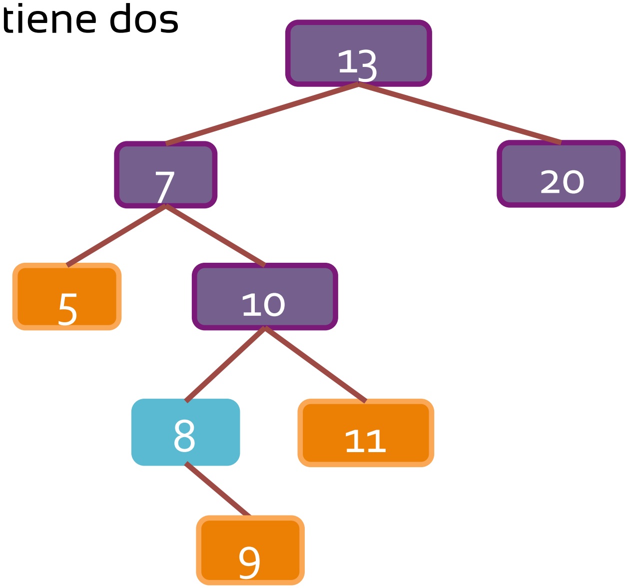
- Estrategia General:
  - Eliminar el elemento correspondiente conectado el nodo del padre con el nodo del hijo del nodo a eliminar
- Ejemplo: **remove(15)**
  - Asignamos a la variable `left` del nodo (20) padre la referencia del hijo del nodo 15
  - Eliminamos el objecto nodo 15



Eliminar en ABB's. Caso 3

Caso 3: Eliminar un nodo interno con dos hijos

- Estrategia general:
  - Cambiar los datos del nodo a eliminar por los datos del sucesor del nodo.
    - i.e. Cambiar los datos del nodo a eliminar con los datos del nodo con valor menor de todo el subárbol
    - Teniendo en cuenta que sabemos que el nodo tiene dos hijos sabemos que su sucesor se encontrará en el subárbol derecho
  - Eliminar el nodo sucesor.



Eliminar en ABB's. Caso 3

Caso 3: Eliminar un nodo interno con dos hijos

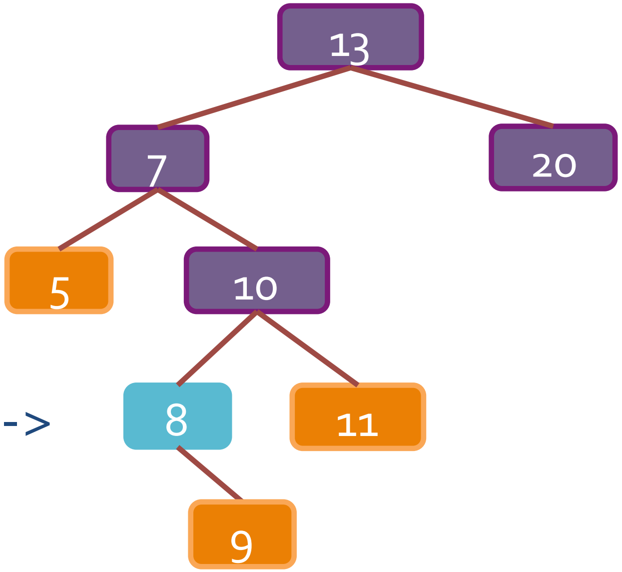
Ejemplo: **remove(7)**

- Primero, buscamos el succesor del nodo dado
  - El sucesor del nodo 7 es el nodo 8

  Código:

```
successor(node):
   // Input: node - the node for
   // which to find the successor
   curr = node.right
   while (curr.left != null):
      curr = curr.left
   return curr
```

Nodo sucesor del 7 ->



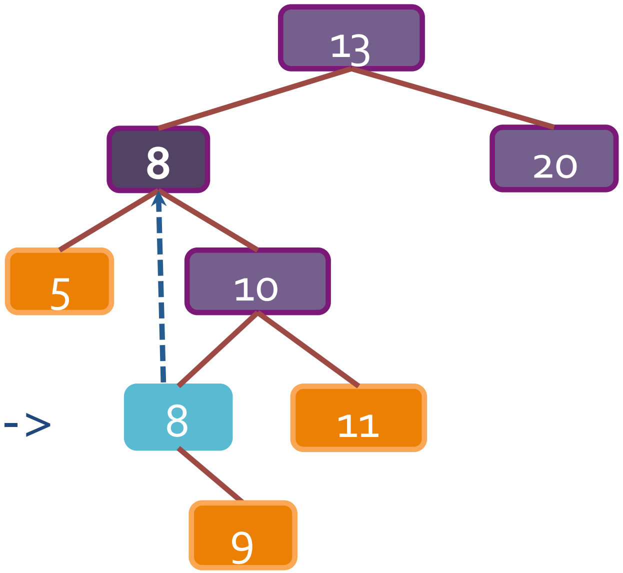
Eliminar en ABB's. Caso 3

Caso 3: Eliminar un nodo interno con dos hijos

Ejemplo: **remove(7)**

- Segundo, remplazamos los datos del nodo a eliminar por los datos del nodo sucesor

Nodo sucesor del 7 ->

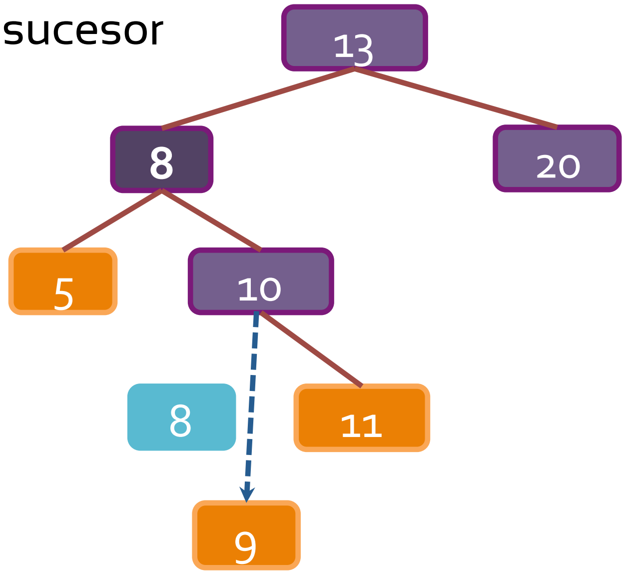


Eliminar en ABB's. Caso 3

Caso 3: Eliminar un nodo interno con dos hijos

Ejemplo: **remove(7)**

- Por último, eliminar el nodo sucesor
  - **IMPORTANTE** : Tenemos la seguridad que el nodo sucesor no tiene hijo izquierdo, ya que en ese caso, su hijo derecho hubiese sido el nodo sucesor. Por tanto, el nodo sucesor como mucho tiene un hijo izquierdo.
  - Podemos eliminar el nodo sucesor aplicando el caso 1 o caso 2

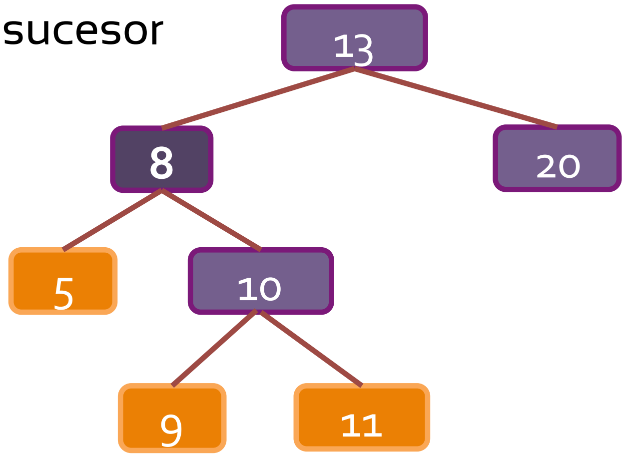


Eliminar en ABB's. Caso 3

Caso 3: Eliminar un nodo interno con dos hijos

Ejemplo: **remove(7)**

- Por último, eliminar el nodo sucesor
  - **IMPORTANTE** : Tenemos la seguridad que el nodo sucesor no tiene hijo izquierdo, ya que en ese caso, su hijo derecho hubiese sido el nodo sucesor. Por tanto, el nodo sucesor como mucho tiene un hijo izquierdo.
  - Podemos eliminar el nodo sucesor aplicando el caso 1 o caso 2



Eliminar en ABB's.

Eliminar en ABB's.

```
function remove(node):
   // Input: node - the node we are trying to remove. We can find this node
   //                    by calling find()
   if node has no children: // case 1 - node is a leaf
      node.parent.removeChild(node)
   else if node only has left child: // case 2a - only left child
      if node.parent.left == node: // if node is a left child
         node.parent.left = node.left
      else:
         node.parent.right = node.left
   else if node only has right child: // case 2b - only right child
      if node.parent.left == node:
         node.parent.left = node.right
      else:
         node.parent.right = node.right
   else:  // case 3 - node has two children
      nextNode = successor(node)
      node.data = nextNode.data // replace node's data with that of nextNode
      remove(nextNode) // nextNode guaranteed to have at most one child
```

¿Cuanto de rápidas son las funciones con ABB's?

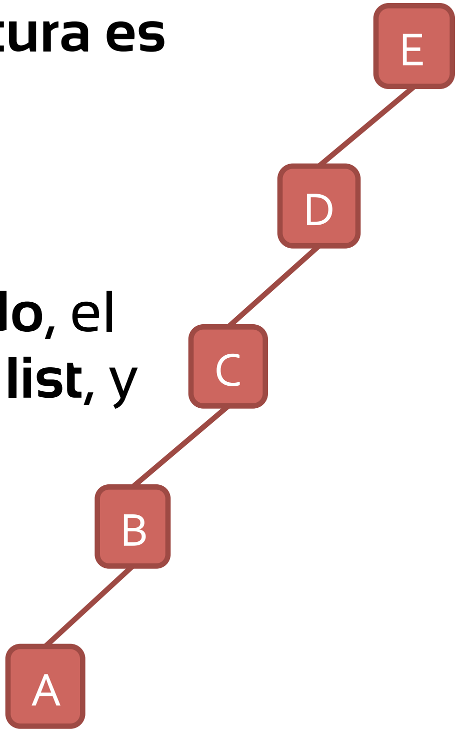
	Representacion Vectorial	**Representacion con nodos No Balanceado**	**Representacion con nodos Balanceado**
Size			
Buscar			
Remplazar			
Eliminar			
Insertar			

¿Cuanto de rápidas son las funciones con ABB's?

- Depende de la altura del árbol. En el **peor caso,** tendremos un árbol con una **altura igual que al número de nodos**.

- Si el árbol está **perfectamente** **balanceado**, la **altura es $\mathbf{log_2 n}$**, lo que nos hace que la complejidad de la funciones sea O(log n) . **MENOS que lineal!!**

- En el caso de un **árbol totalmente desbalanceado**, el **árbol de búsqueda binaria en una sorted linked list**, y sus funciones tiene una complejidad de O(n).



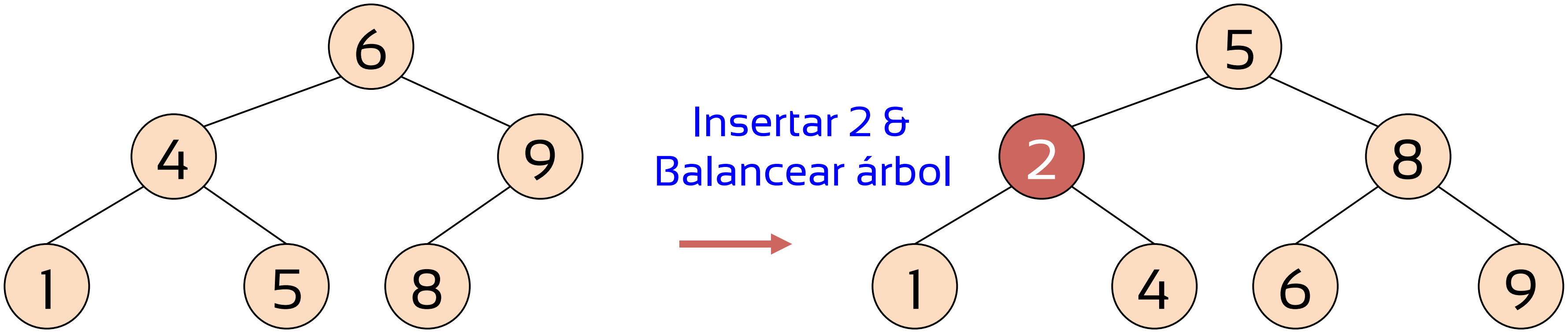
¿Qué son los árboles balanceados?

- Árbol de Búsqueda **perfectamente balanceado**:
  - Para todo nodo, la **cantidad de nodos** de su subárbol izquierdo difiere como máximo de 1 de la cantidad de nodos del subárbol derecho.

- **Árbol balanceado o AVL** (Adelson – Velskii y Landis)
  - Condición de balanceo más débil para que no sea tan costoso el proceso de balancear un árbol.
  - Definición: Para todo nodo, **la altura** de sus subárboles difiere como máximo en 1.

Árbol perfectamente balanceado

- Queremos un árbol completo después de cada iteración
- Esta operación conlleva una alta complejidad
  - Por ejemplo, insertar 2 en el árbol izquierdo y luego reconstruirlo a un árbol completo



Árbol balanceado o AVL

- Los árboles AVL son llamados ABB de altura balanceada

Factor de Balance de un nodo:

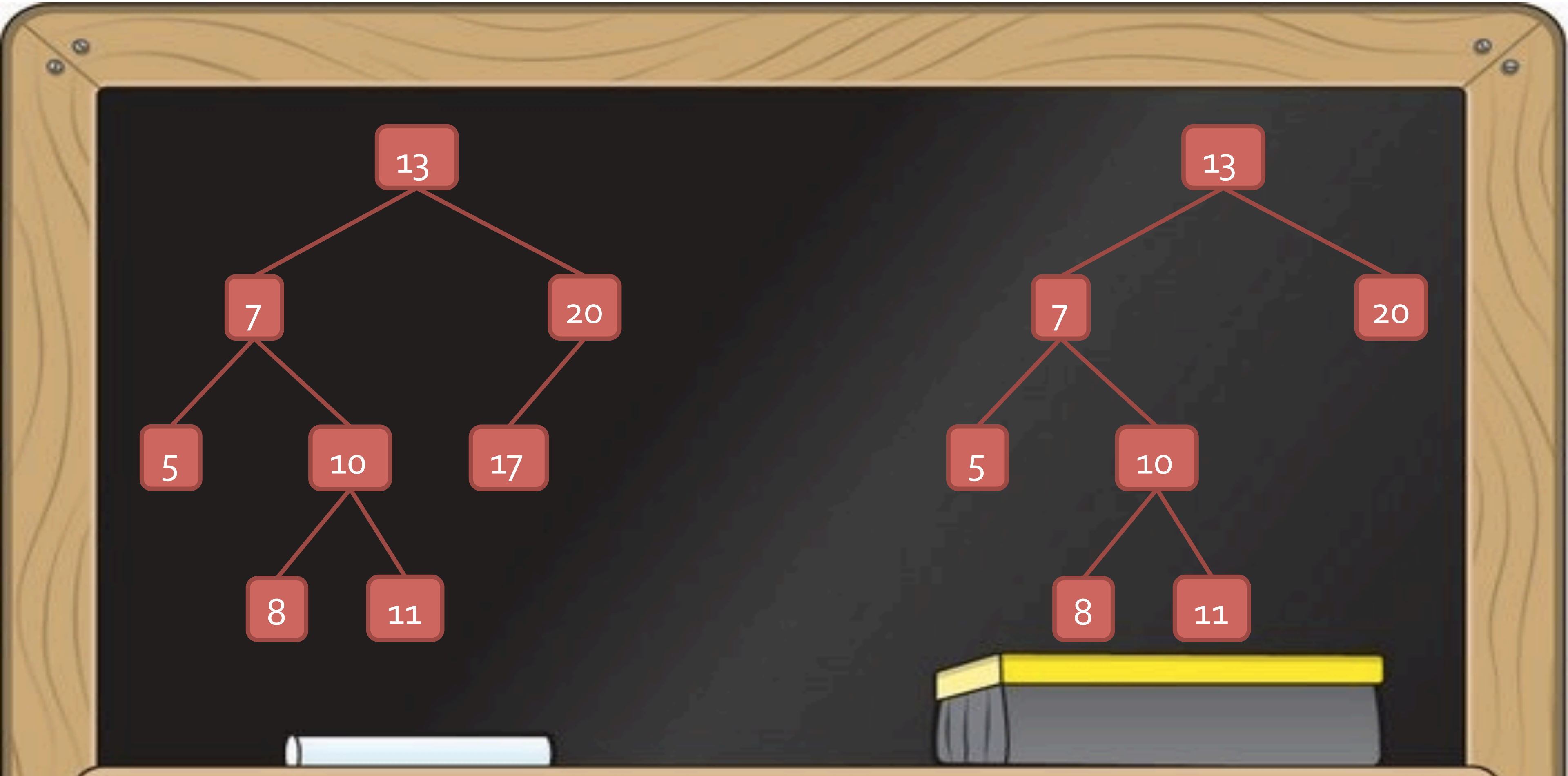
Altura(subárbol izquierdo) - Altura(subárbol derecho)

- Un **árbol AVL** tiene el factor de balance calculado en cada nodo.
  - **Para cada nodo, la altura del subárbol izquierdo y derecho solo puede diferir de 1**
  - Guardamos la altura en cada nodo.

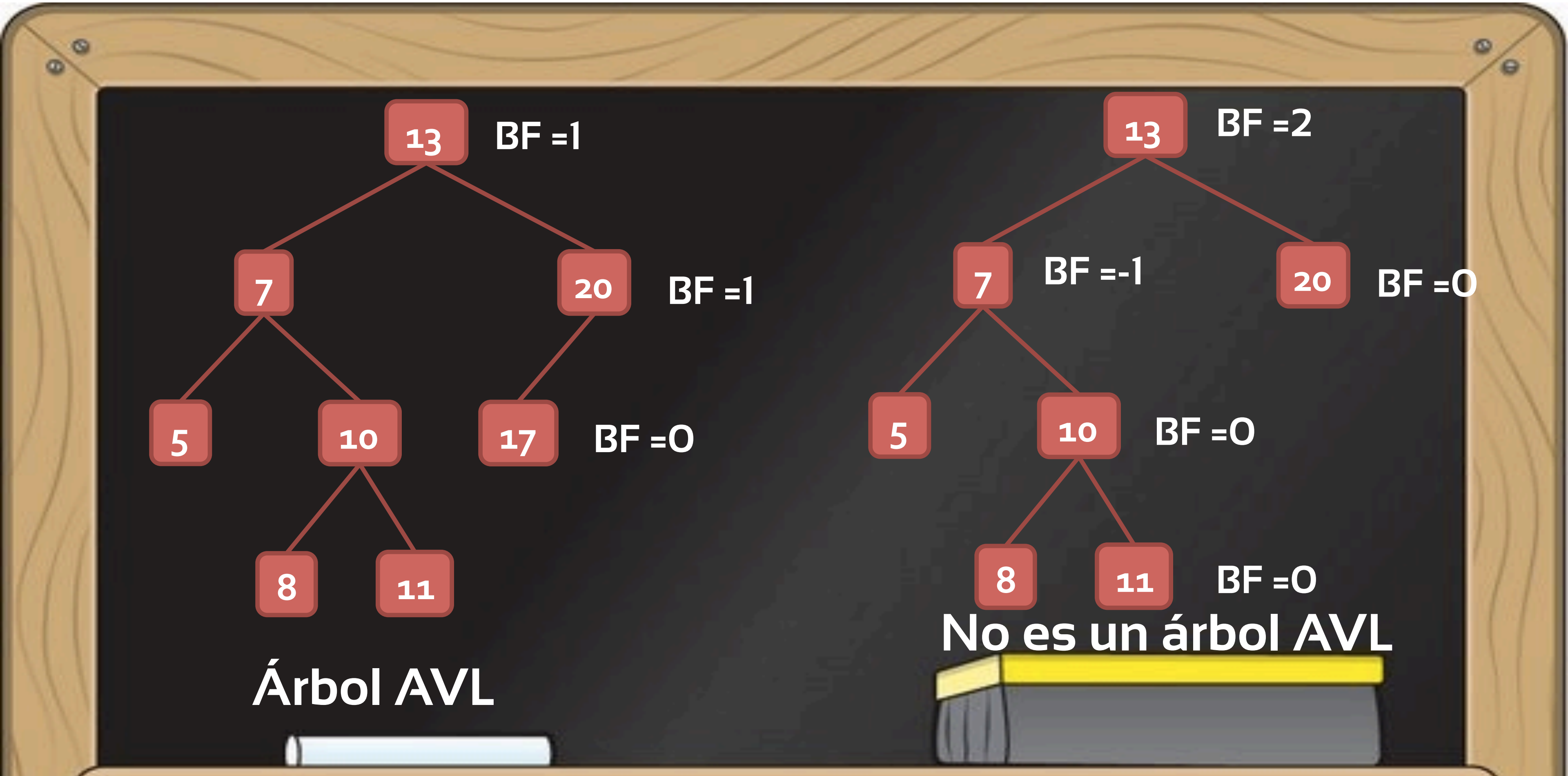
¿Perfectamente Balanceado? ¿AVL?

Balance factor de un nodo:

Altura (subárbol izquierdo) - Altura(subárbol derecho)

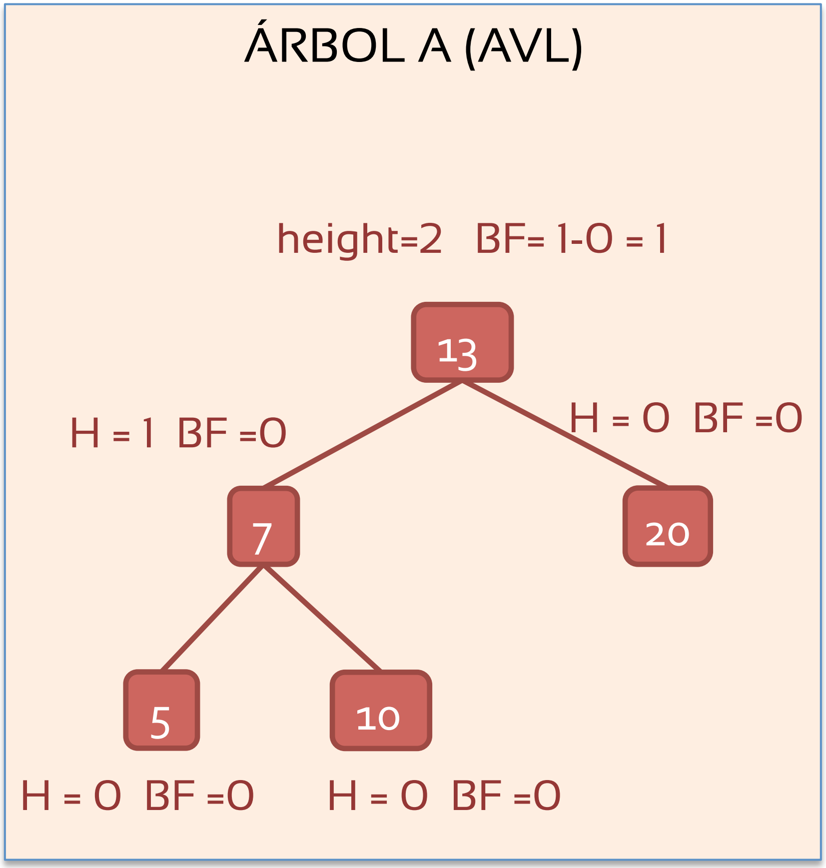


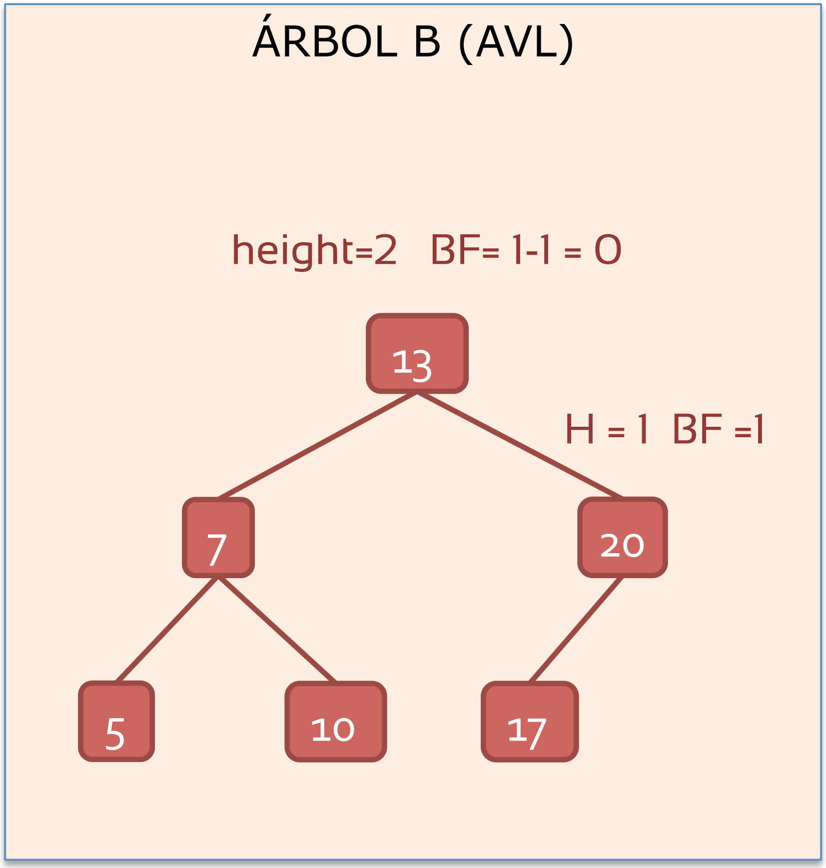
¿Perfectamente Balanceado?
¿AVL?



Altura (Height) de un nodo

height of node = h
balance factor = h_{left}-h_{right}
empty height = -1



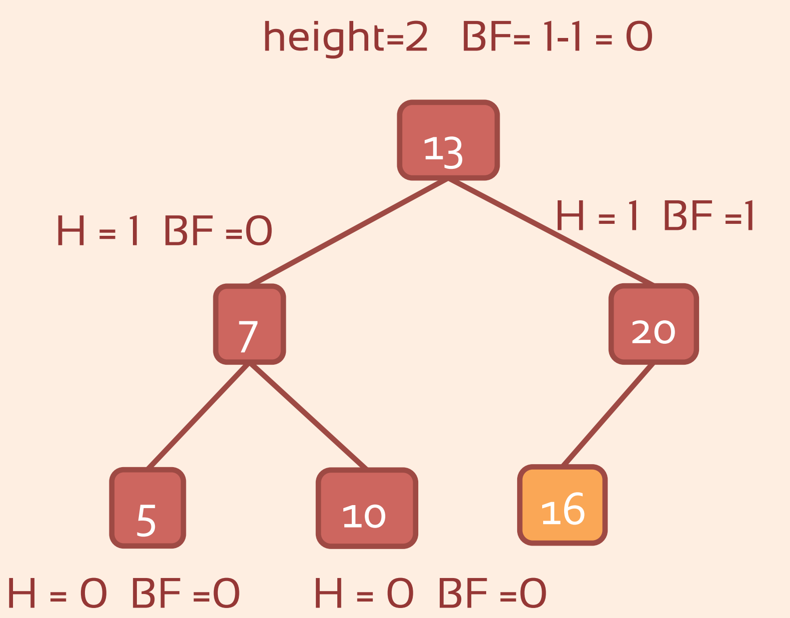


Altura (Height) de un nodo

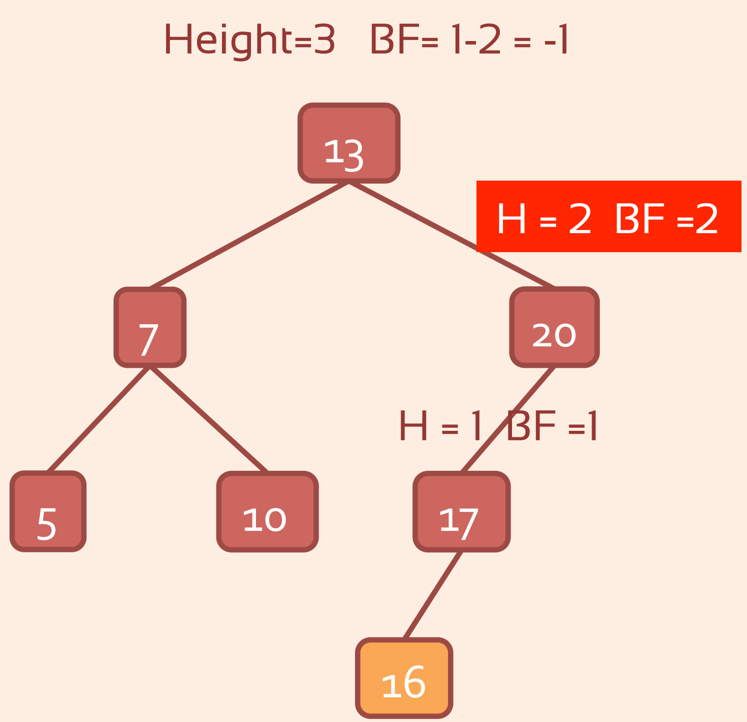
height of node = h
balance factor = $h_{left}-h_{right}$
empty height = -1

Despues de añadir elemento 16

ÁRBOL A (AVL)



ÁRBOL B (AVL)

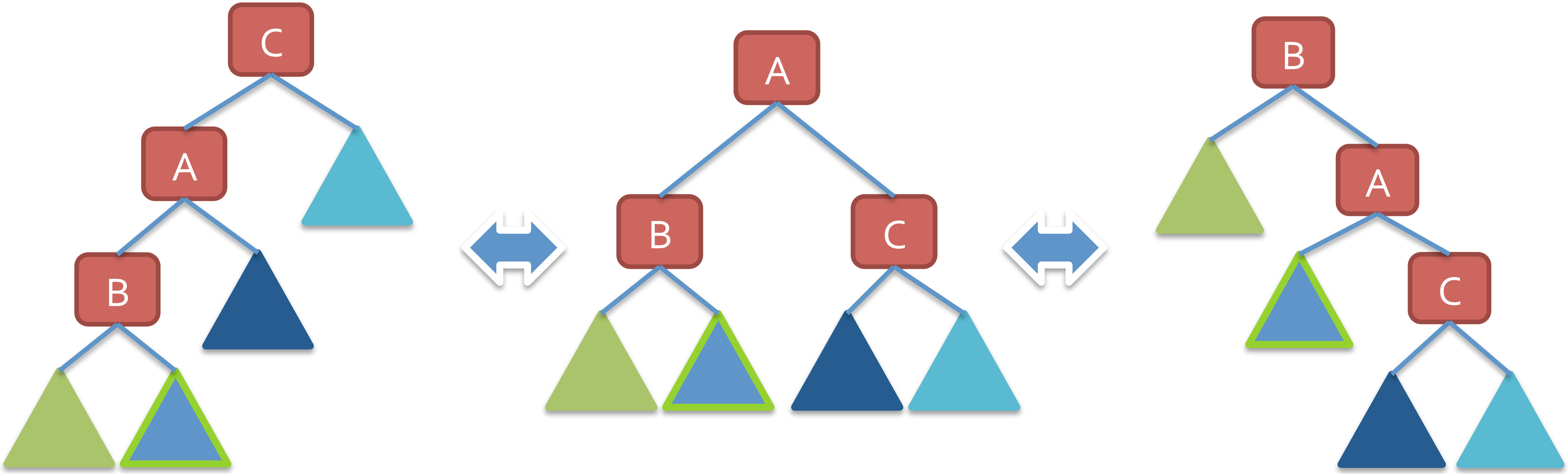


Insertar y rotar árboles AVL

- La inserción de un elemento puede causar que el factor de balance se convierta en 2 o -2 por algún nodo.
  - Solo los nodos desde el punto de inserción hacia el elemento raíz tienen posibilidad de cambiar su altura.
  - Por tanto la inserción, **va hacia arriba** hasta la raíz pasando por cada nodo actualizado su altura.
  - Si el nuevo factor de balance es 2 o -2, tenemos que ajustar el árbol mediante **rotaciones** del nodo.

Balanceado de ABB: AVL

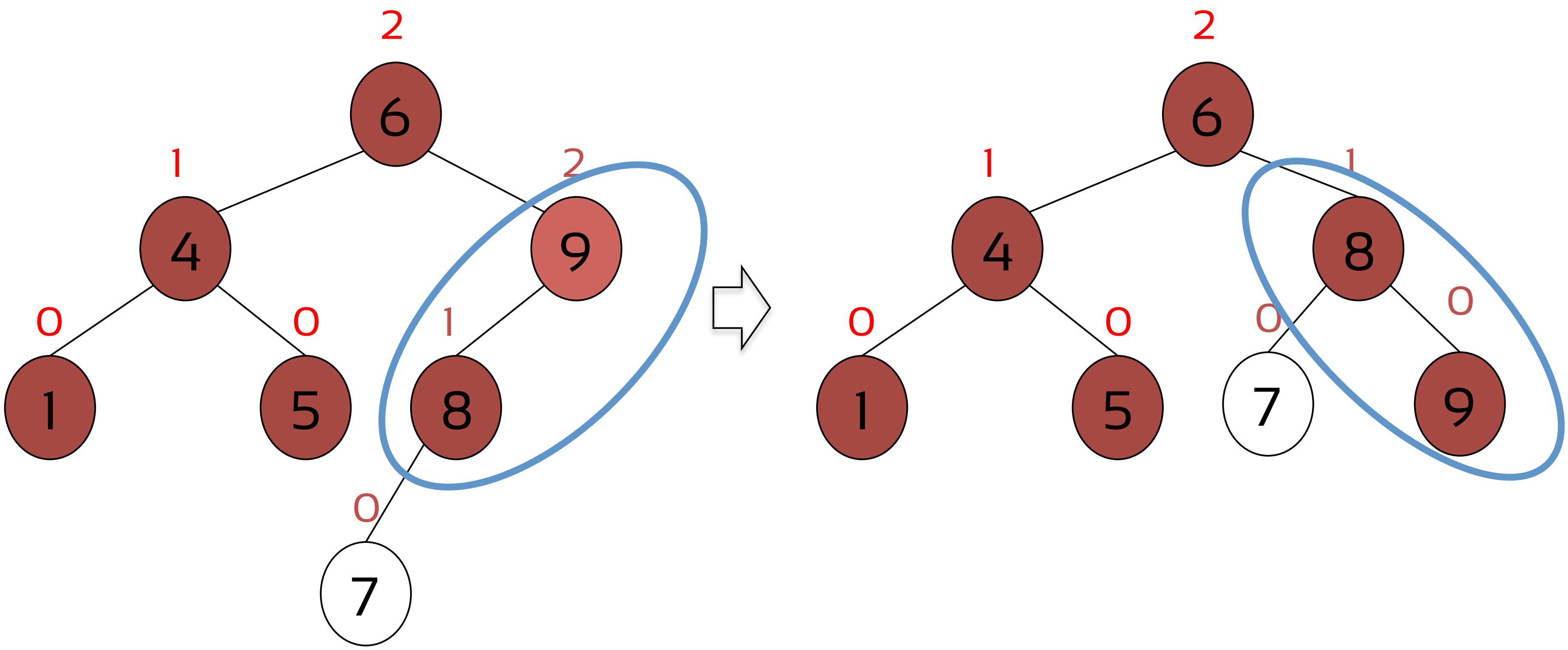
- Si el árbol binario se convierte en un árbol no balanceado, podemos revertir esto mediante una serie de rotaciones.



Podemos ver que con el recorrio in-orden obtemos el mismo resultado

Lo que significa que el orden de los ABB's se mantiende con las rotaciones

Rotaciones en un árbol AVL



Inserción en árboles AVL

Dado un nodo α que necesita re-balanceó.

Hay 4 casos distinto:

Casos externos (requiere una rotación simple) :

1. Inserción en el árbol izquierdo del hijo izquierdo de α.
2. Inserción en el árbol derecho del hijo derecho de α.

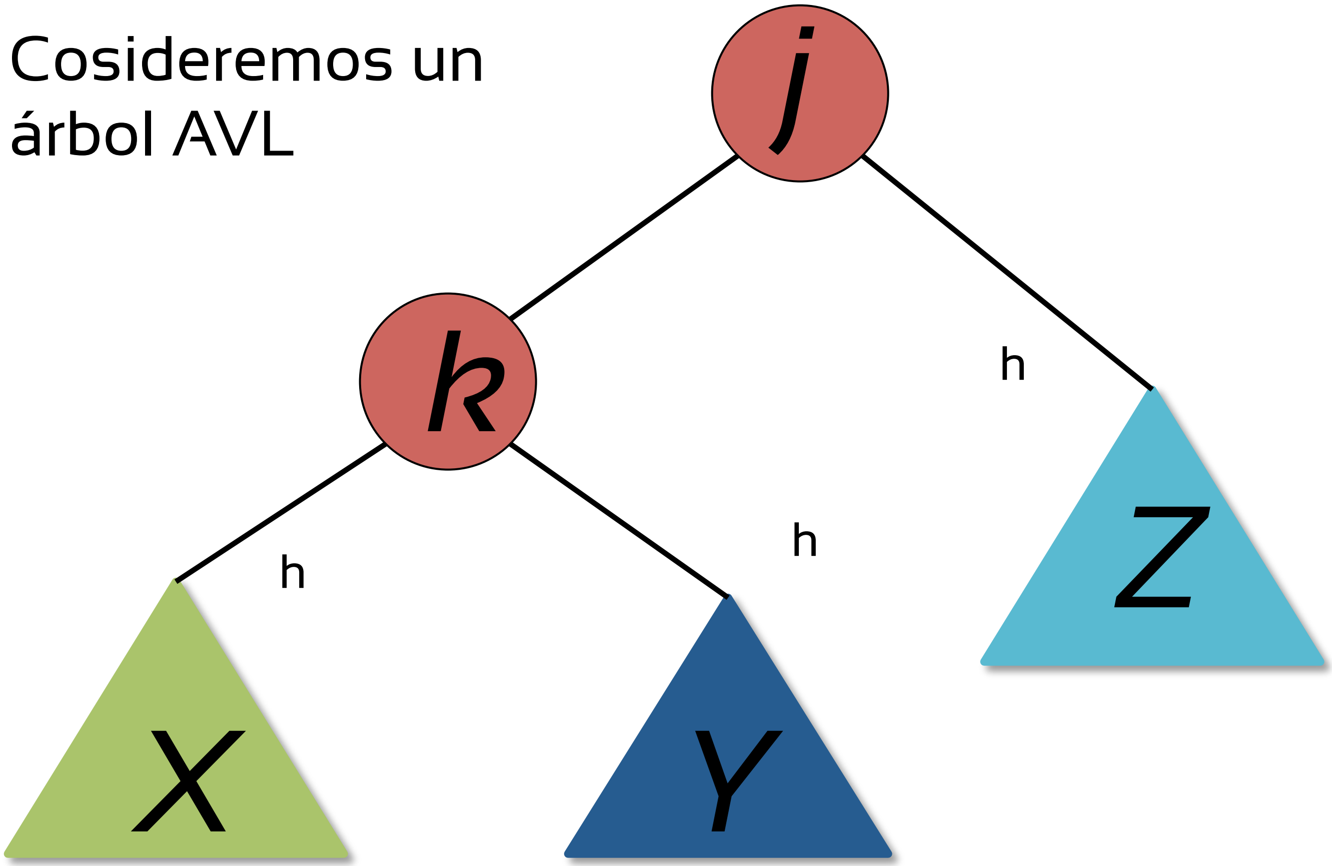
Casos Internos (requiere una rotación doble) :

3. Inserción en el árbol derecho del hijo izquierdo de α.
4. Inserción en el árbol izquierdo del hijo derecho de α.

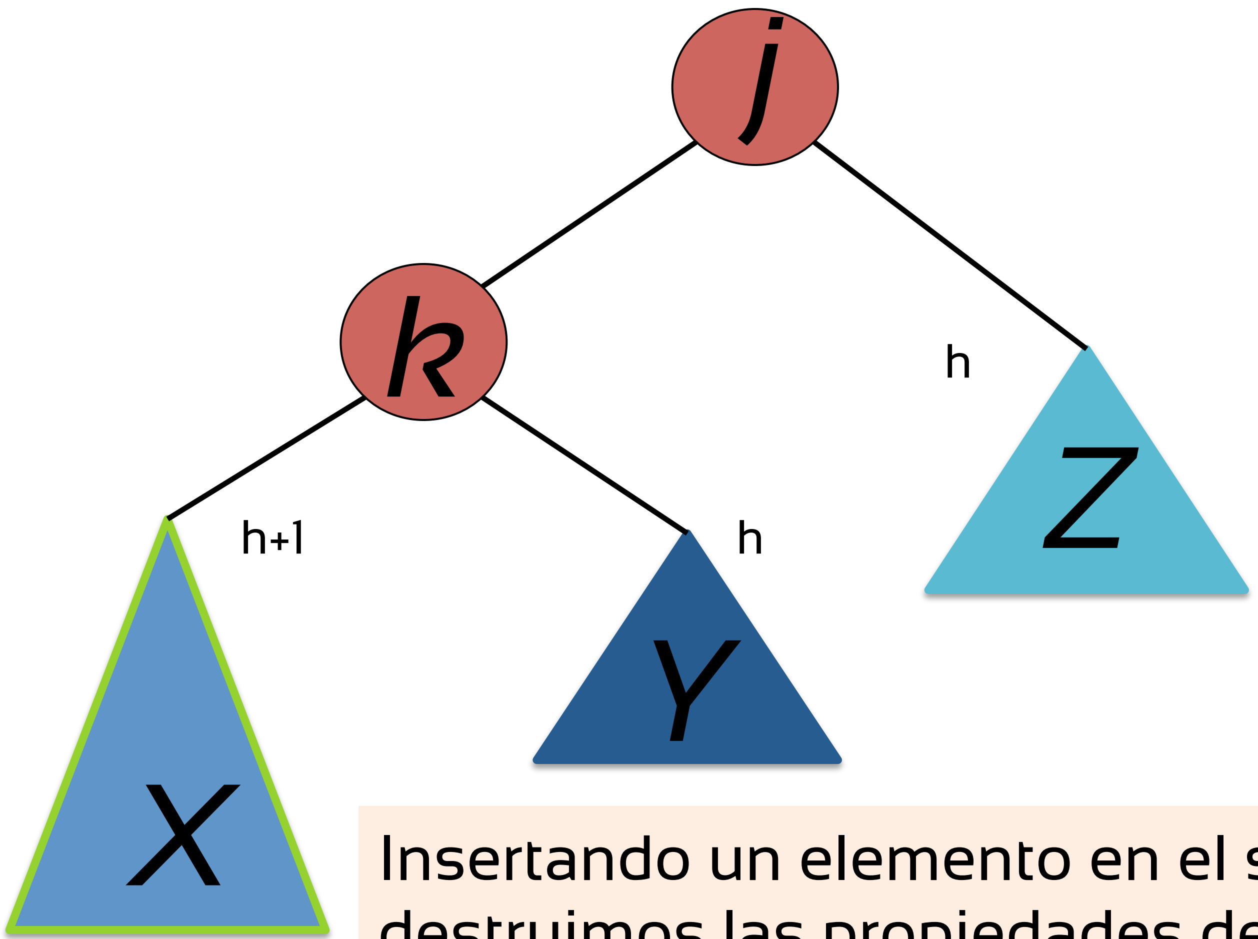
El re-balanceo se realiza mediante 4 algoritmos de rotación distintos

Inserción en un AVL: Caso externo

Cosideremos un árbol AVL

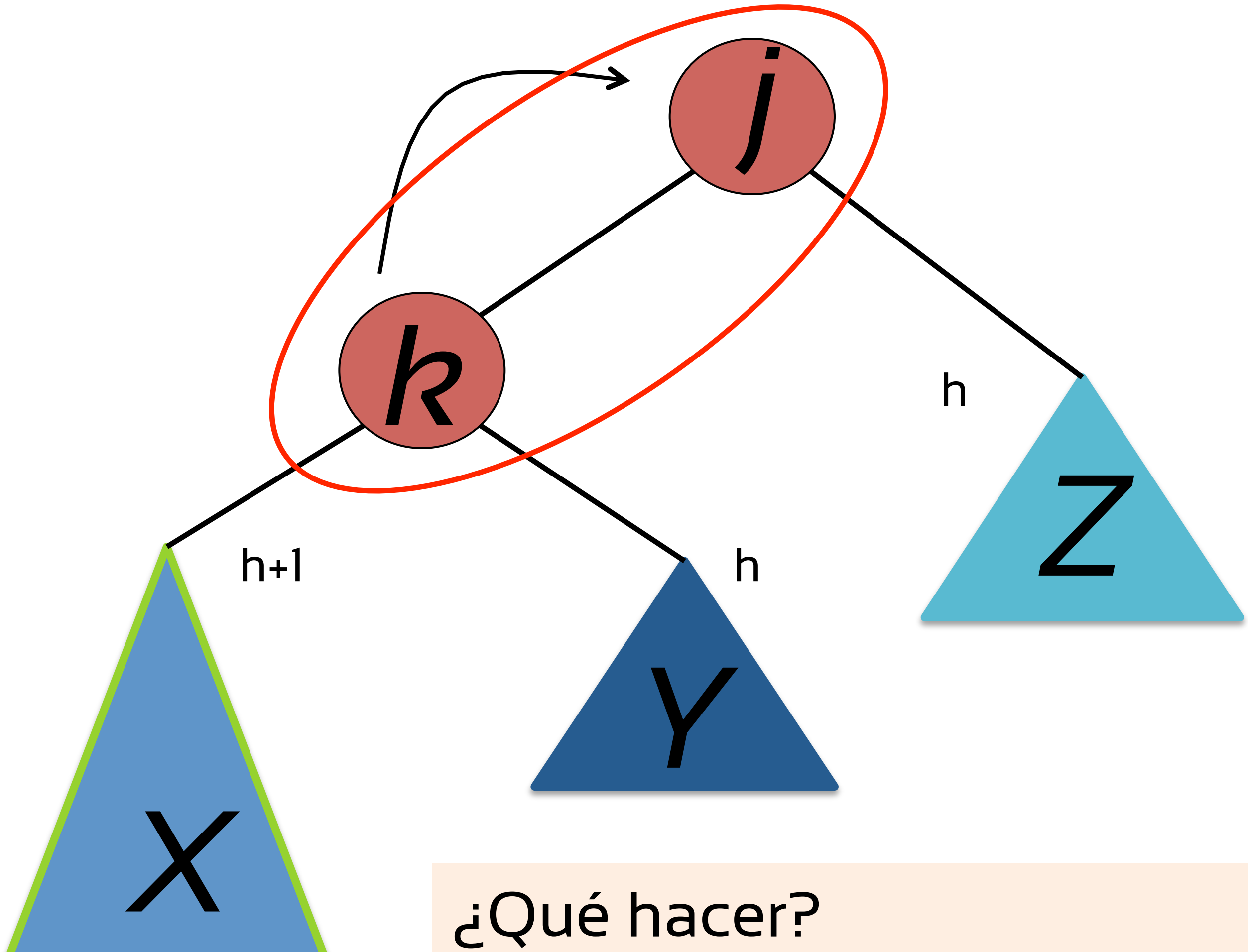


Inserción en un AVL: Caso externo



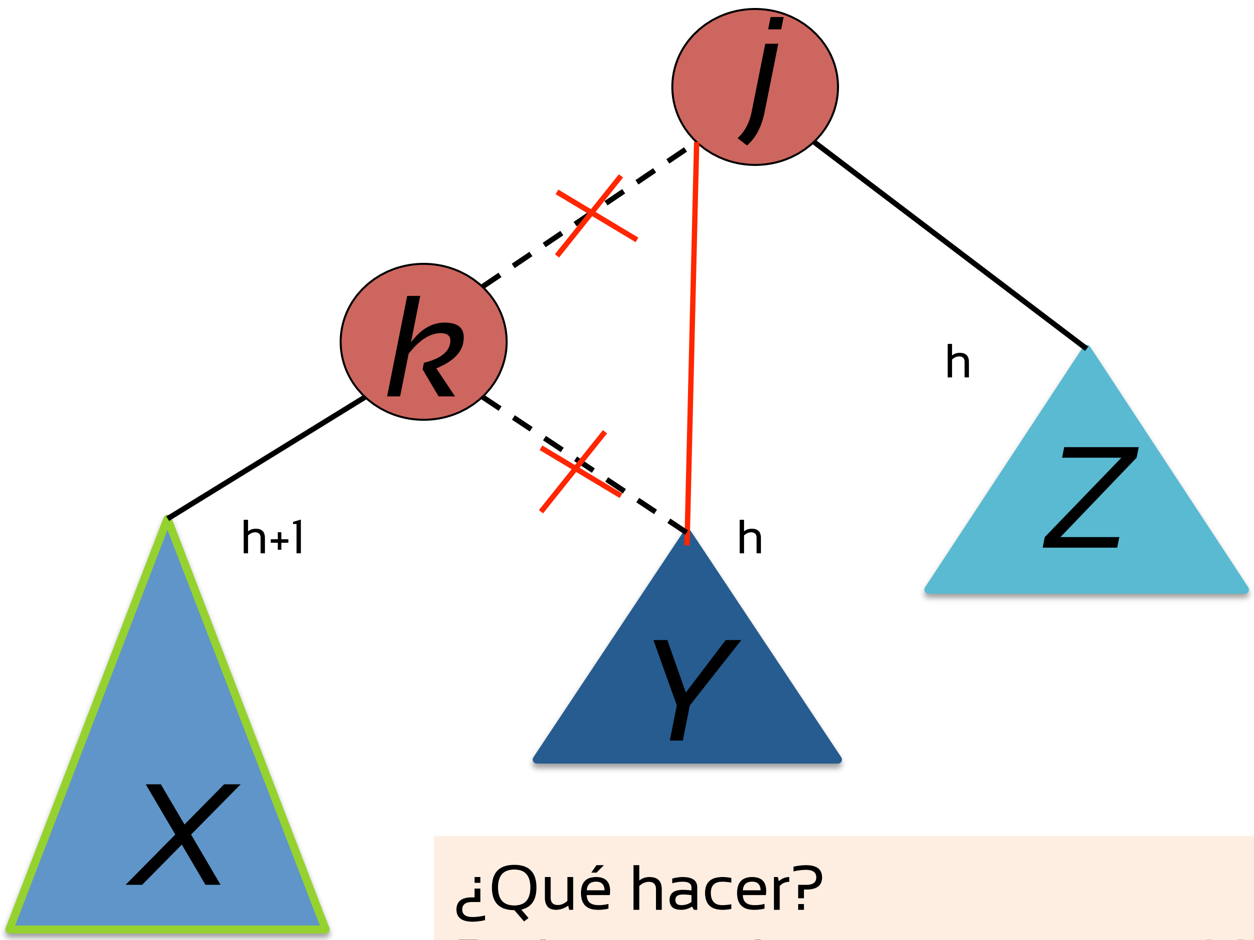
Insertando un elemento en el subárbol **X** destruimos las propiedades del árbol AVL en el nodo **j**

Inserción en un AVL: Caso externo



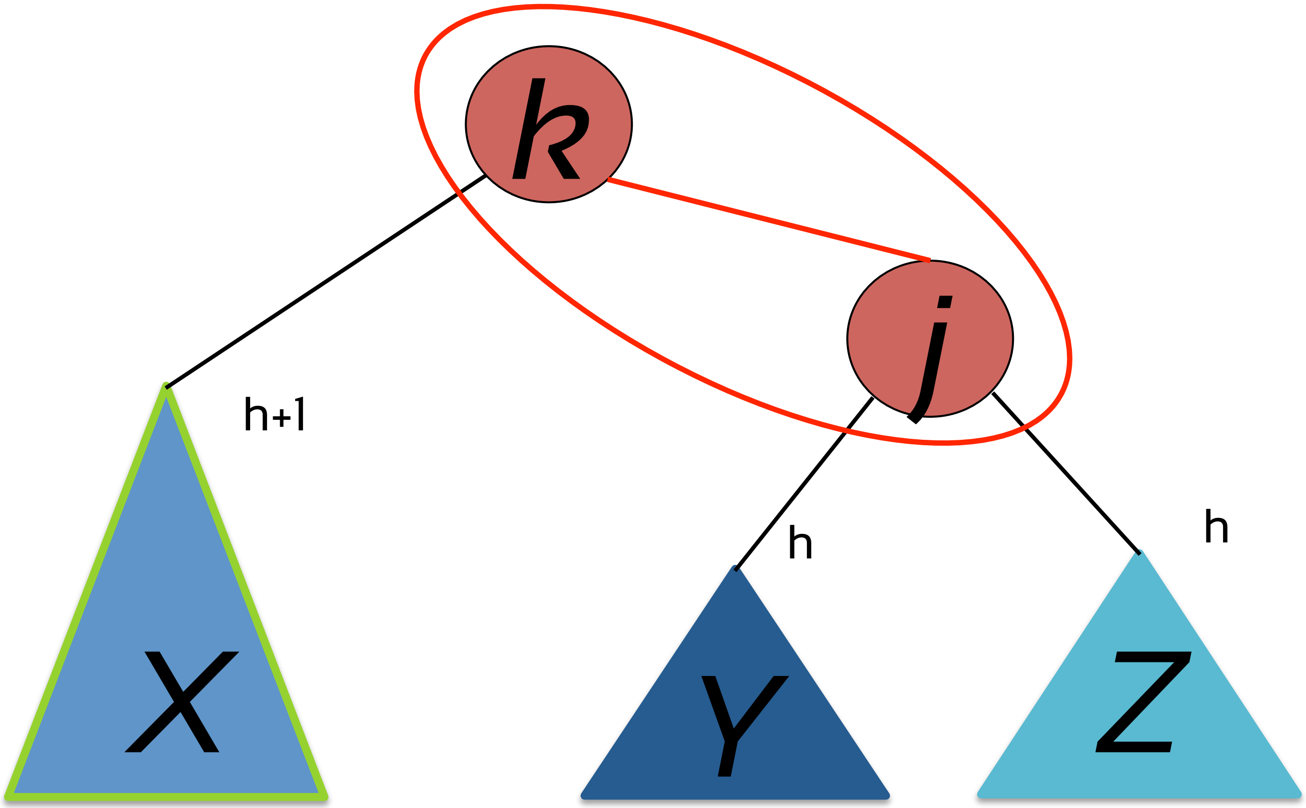
¿Qué hacer?
Debemos hacer una rotación derecha

Rotación izquierda simple



¿Qué hacer?
Debemos hacer una rotación derecha

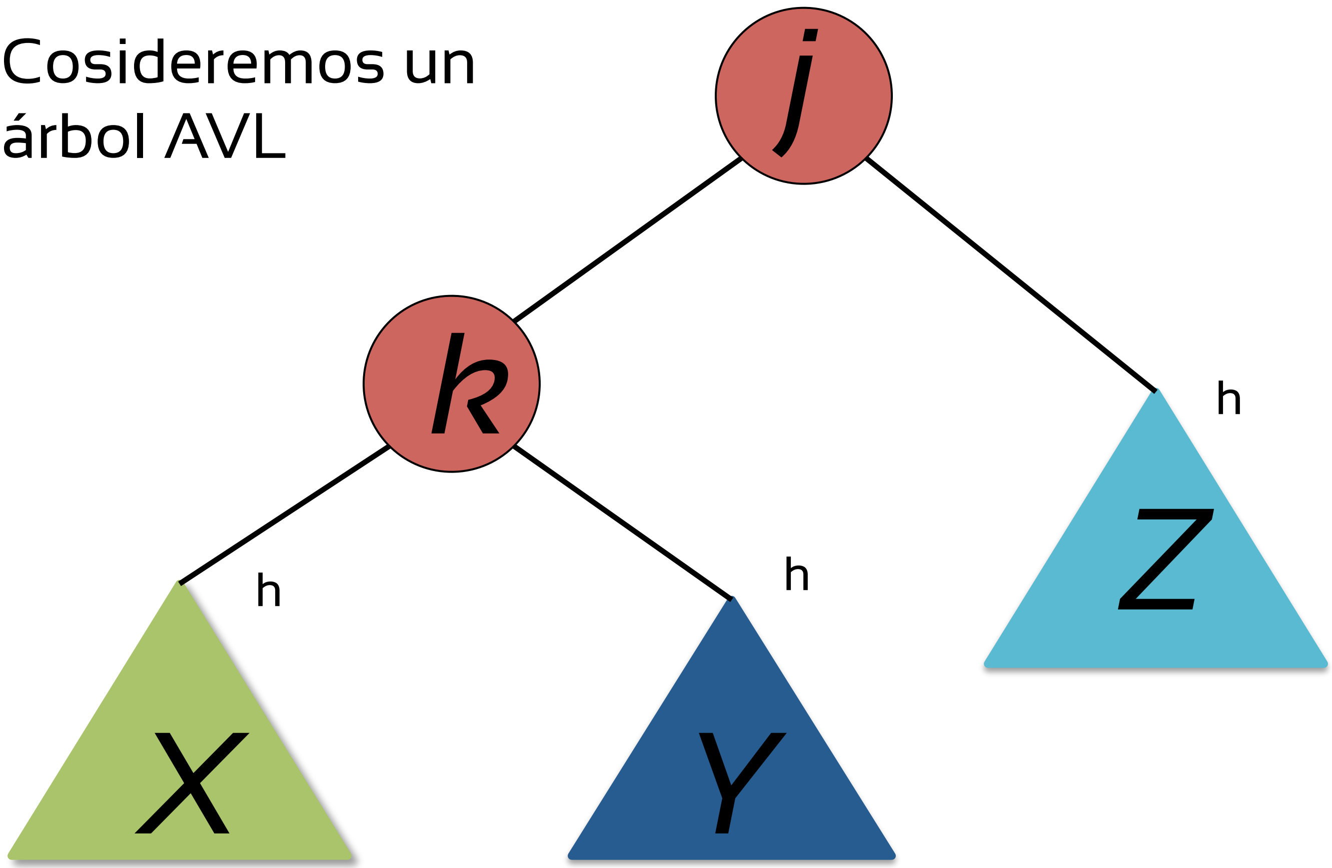
Caso externo completado!!



"Rotación derecha" realizada!
("Rotación izquierda" es un caso simétrico)
Las propiedades del árbol AVL se ha restaurado

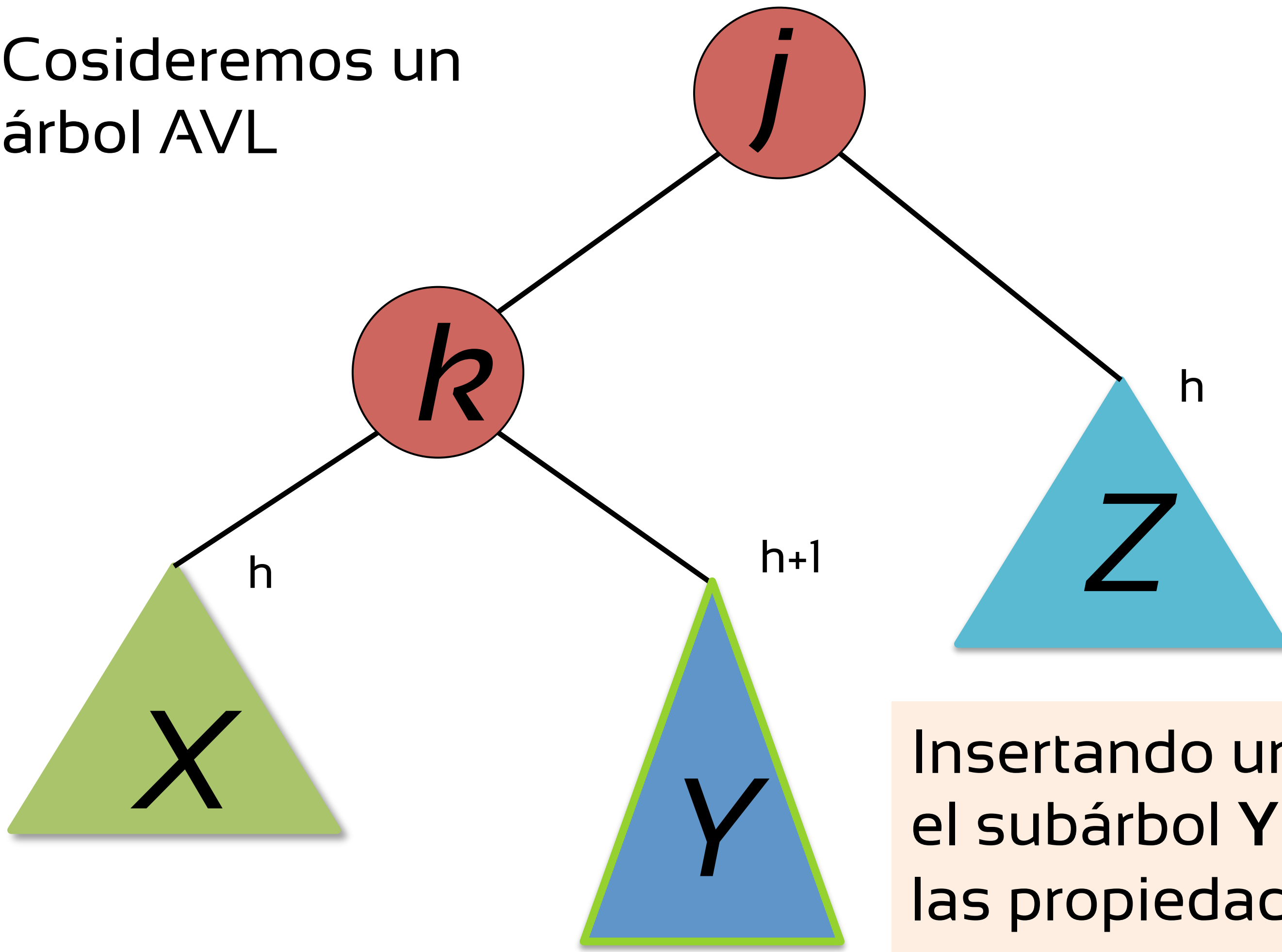
Inserción en un AVL: Caso interno

Cosideremos un árbol AVL



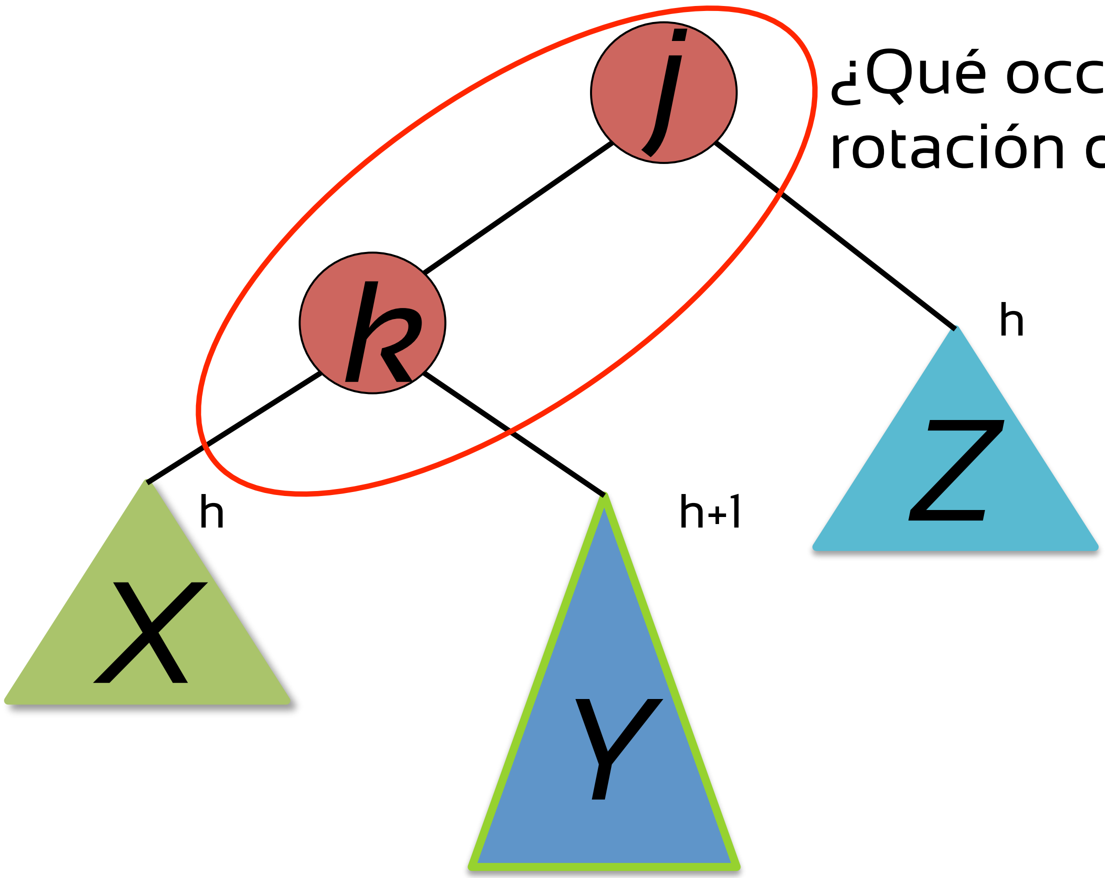
Inserción en un AVL: Caso interno

Cosideremos un árbol AVL



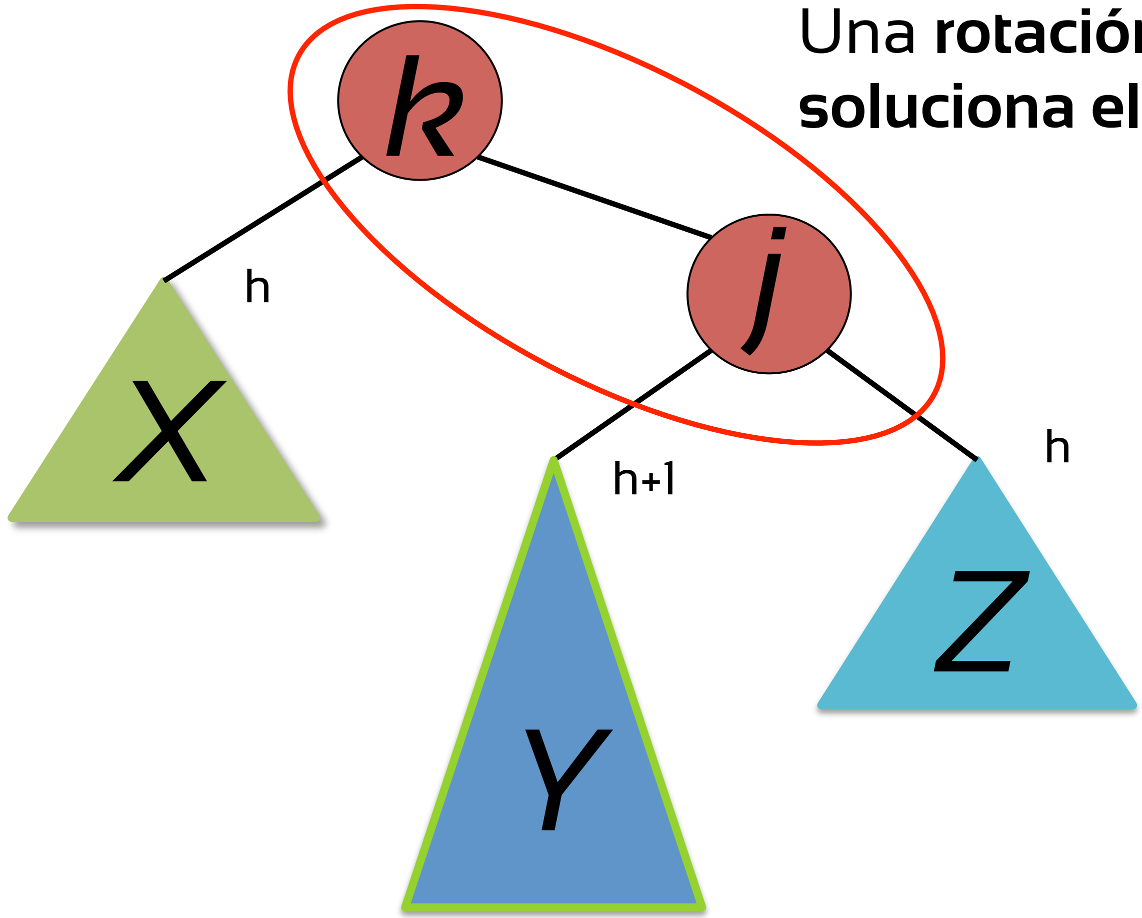
Insertando un elemento en el subárbol **Y** destruimos las propiedades del árbol AVL en el nodo **j**

Inserción en un AVL: Caso interno



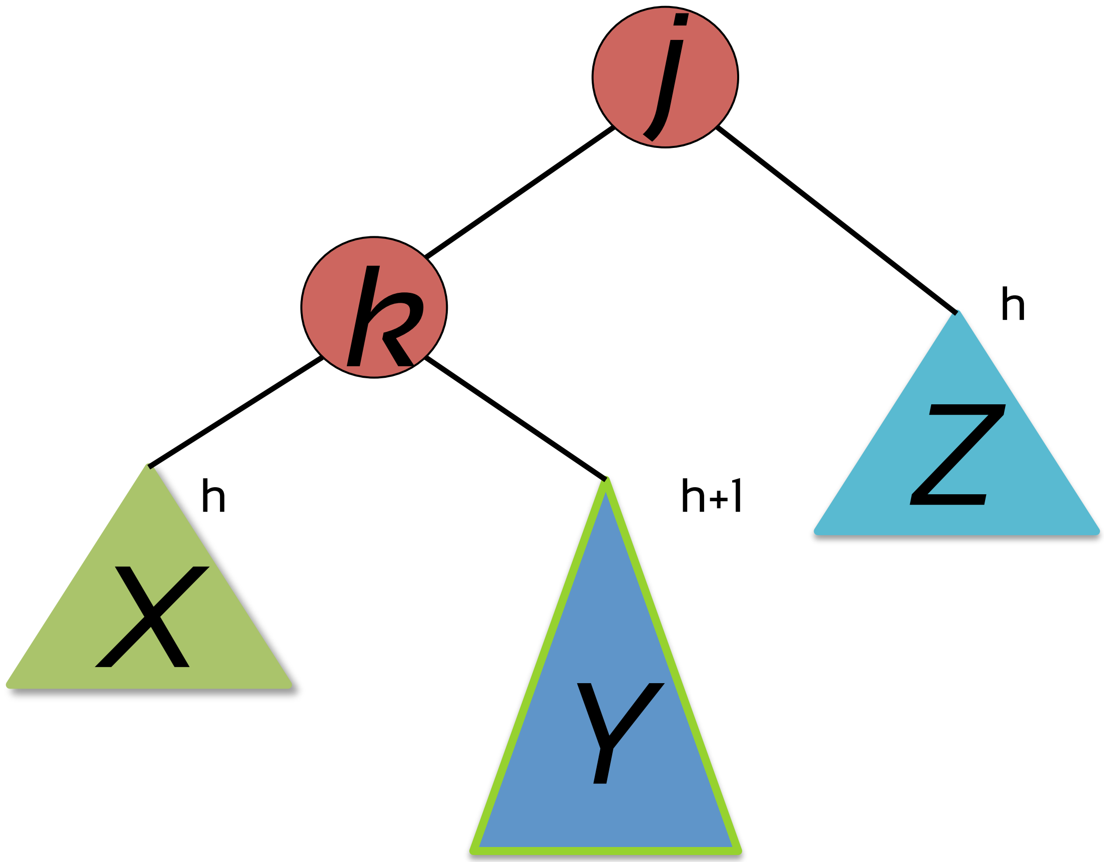
¿Qué occurre si hacemos una rotación derecha?

Inserción en un AVL: Caso interno



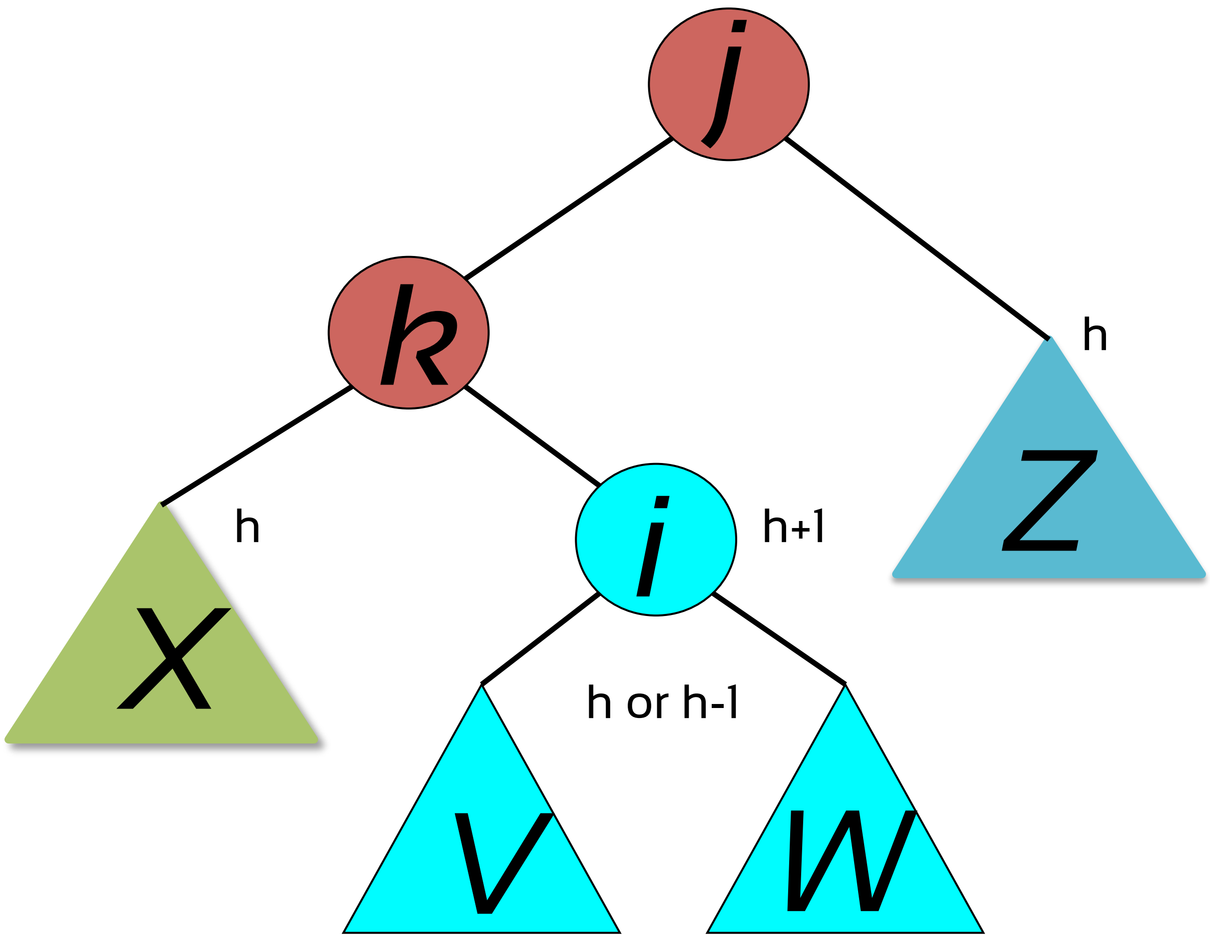
Una **rotación derecha no soluciona el desbalanceado**

Inserción en un AVL: Caso interno



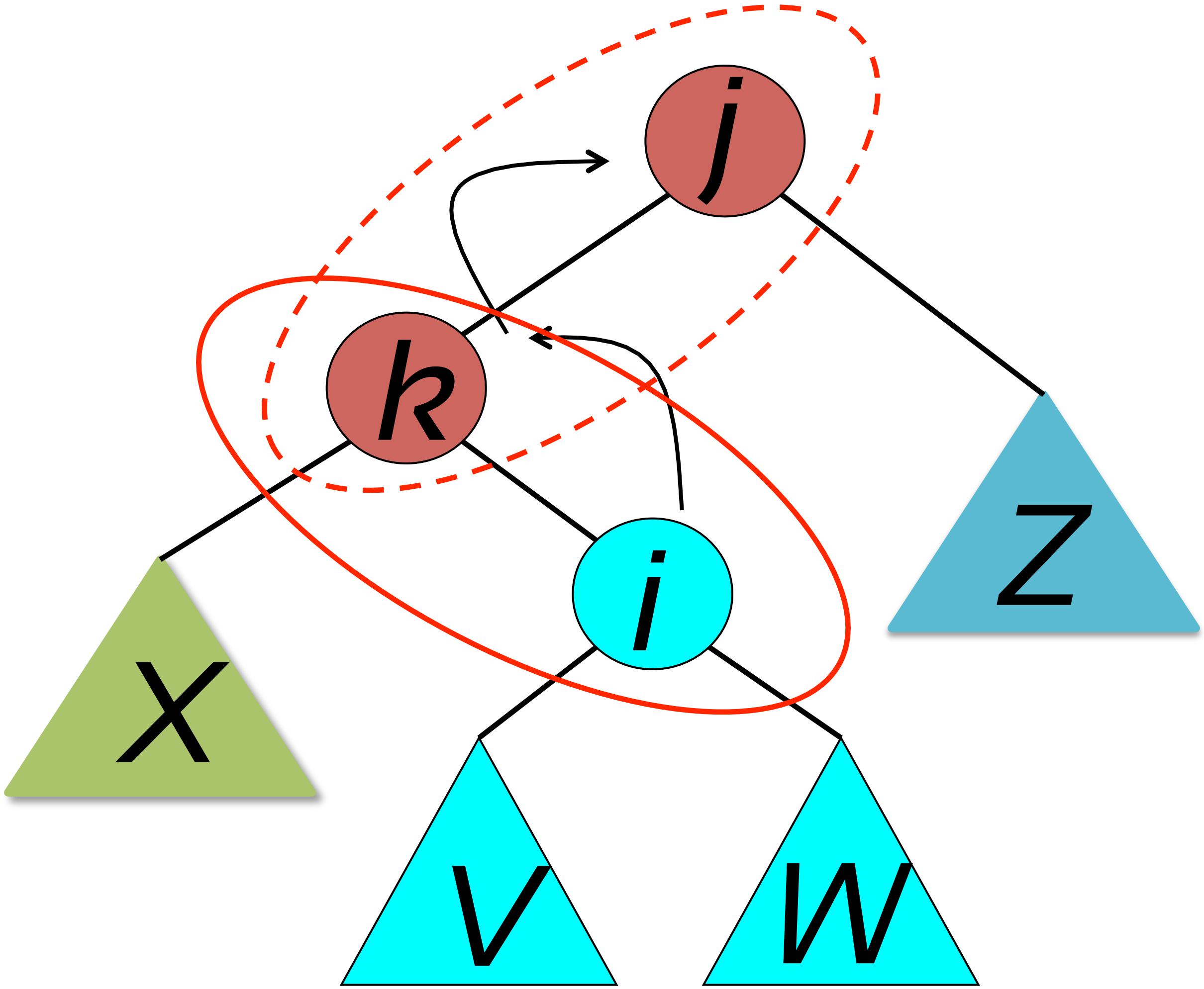
Consideremos la estructura defindia por el subárbol Y

Inserción en un AVL: Caso interno



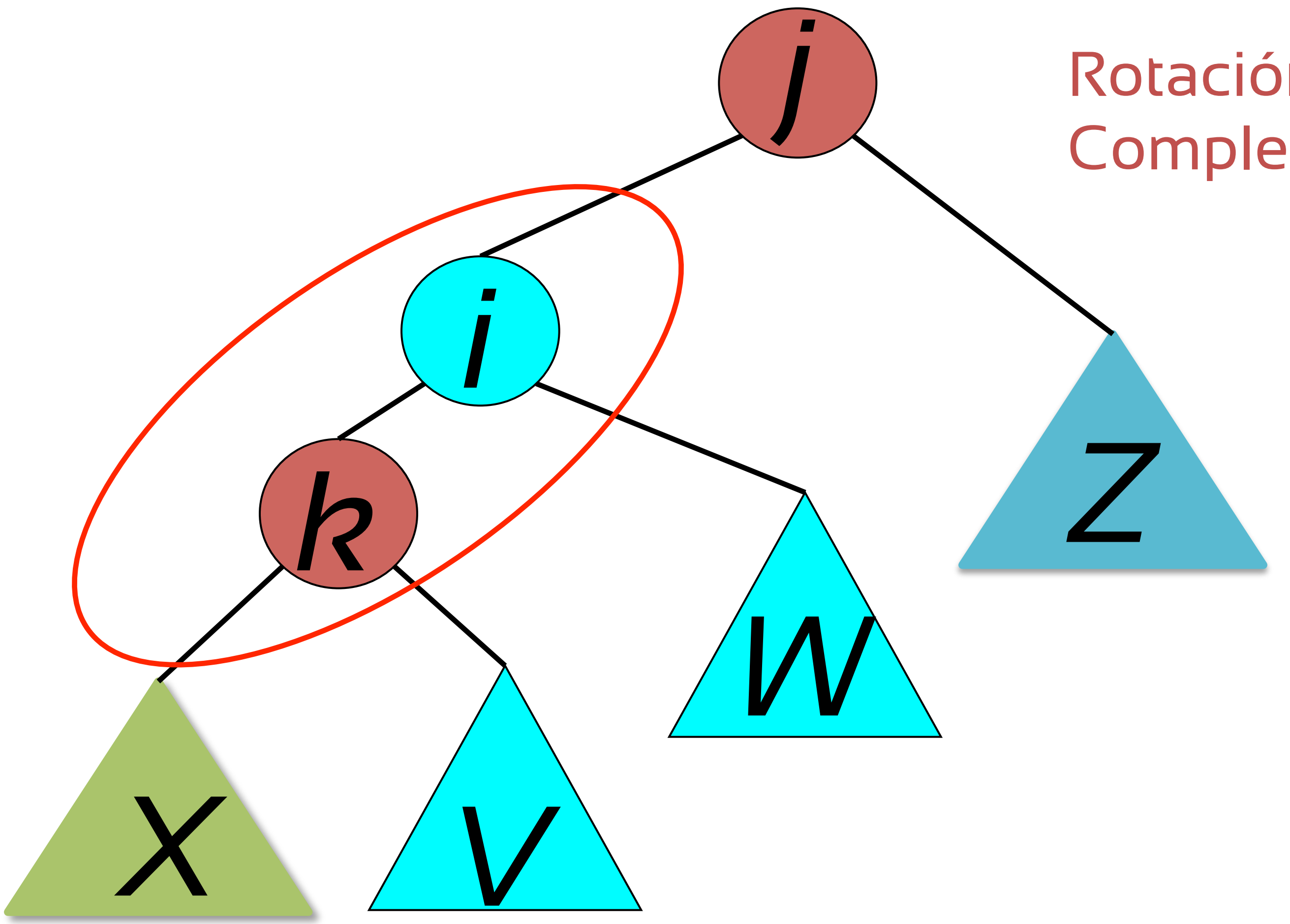
Y == nodo i & subárboles V and W

Inserción en un AVL: Caso interno



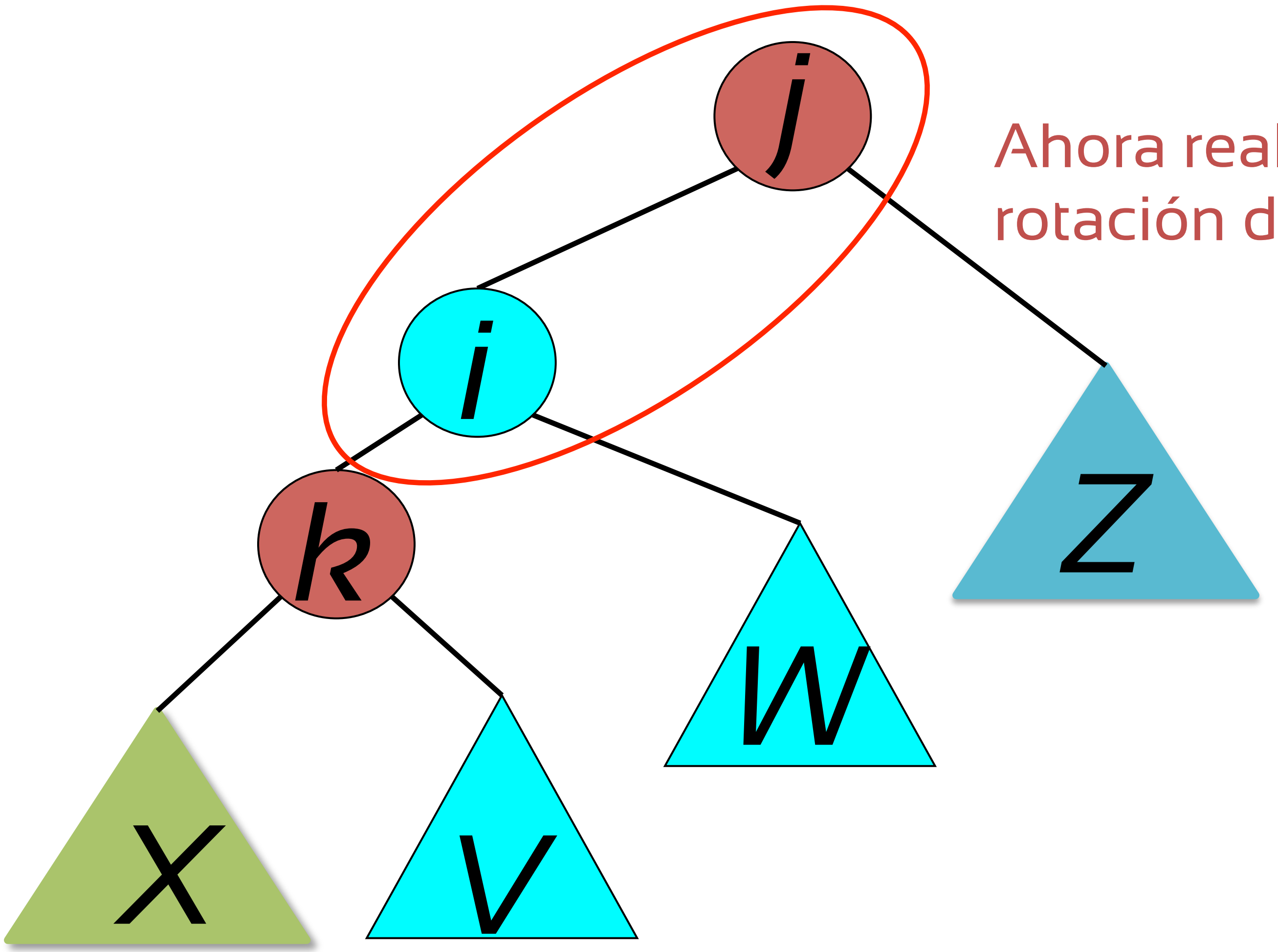
Vamos a realizar una **doble rotación izquierda-derecha...**

Rotación Doble: Primera rotación



Rotación Izquierda Completada

Rotación Doble: Segunda rotación

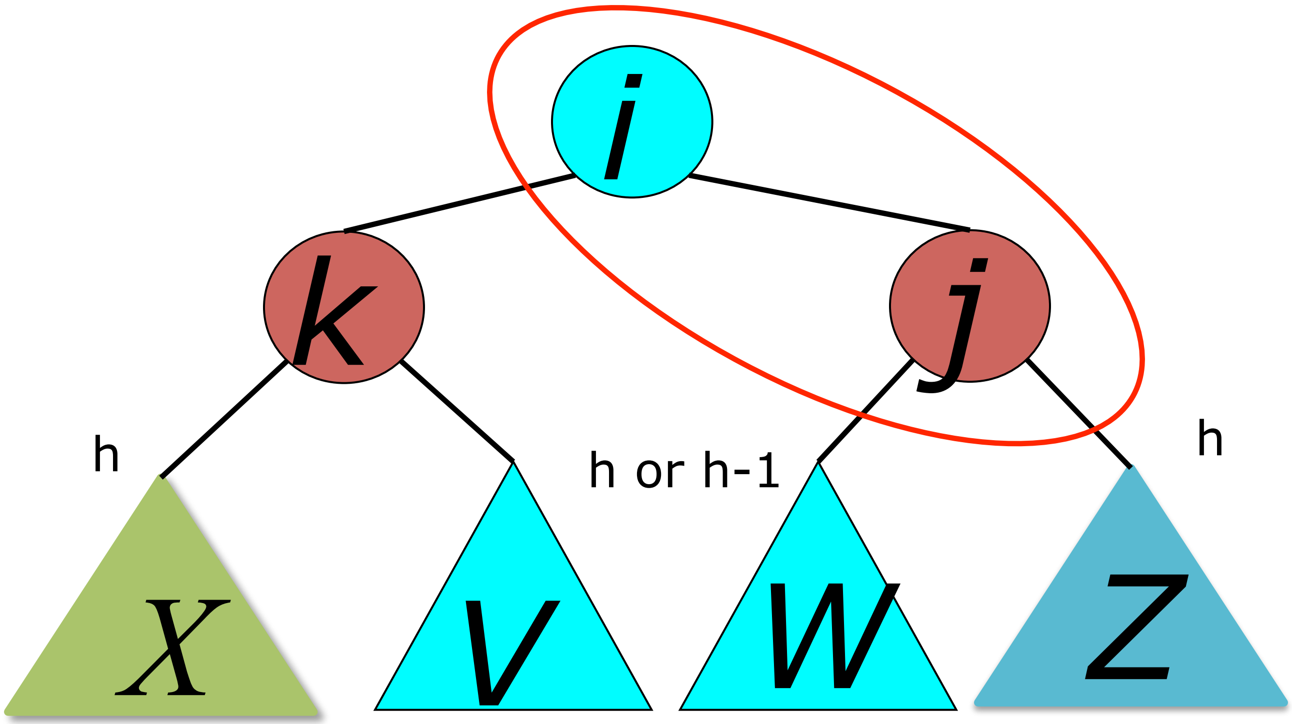


Ahora realizaremos una rotación derecha

Rotación Doble: Segunda rotación

Rotación derecha completada

Árbol balanceado recuperado

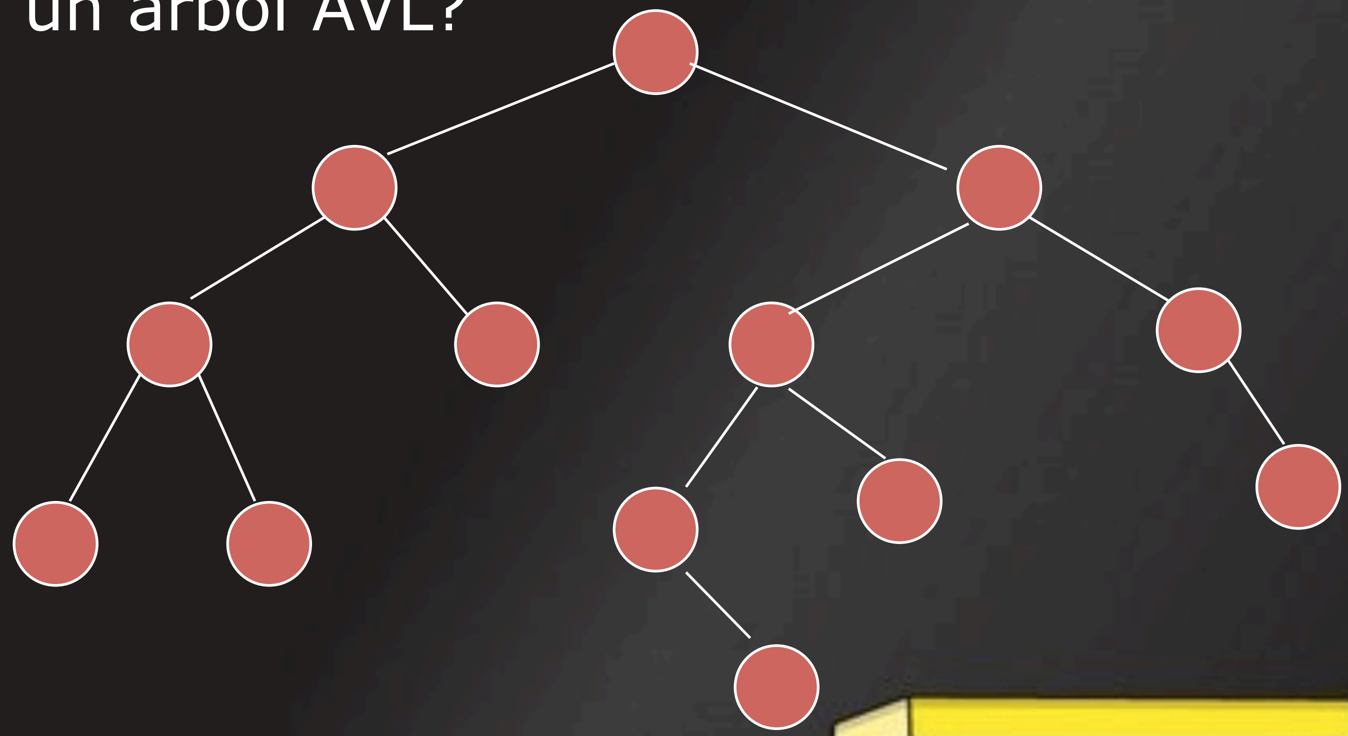


Inserión AVL: Pseudo-Código

```
function insert(node, toInsert):
   // Input: node - root node of tree
   //        toInsert - data you are trying to insert

   if node.data == toInsert: // data already in tree
      return

   if toInsert < node.data:
      if node.left == null: // add as left child
         node.addLeft(toInsert)
      else:
         insert(node.left, toInsert)
         updateBalance(node.left) //update the BalanceFactor
   else:
      if node.right == null: // add as right child
         node.addRight(toInsert)
      else:
         insert(node.right, toInsert)
         updateBalance(node.right) //update the BalanceFactor
```

Factor de Balance para un árbol AVL

¿Cual es el factor de Balance de cada nodo?
¿Es un árbol AVL?



Construcción de un árbol AVL

¿Qué árbol obtendremos al crear un AVL añadiendo los siguientes números?

- 8, 9, 10, 2, 1, 5, 3, 6, 4, 7, 11, 12