Árboles AVL

CSE 373

Data Structures

Lecture 8

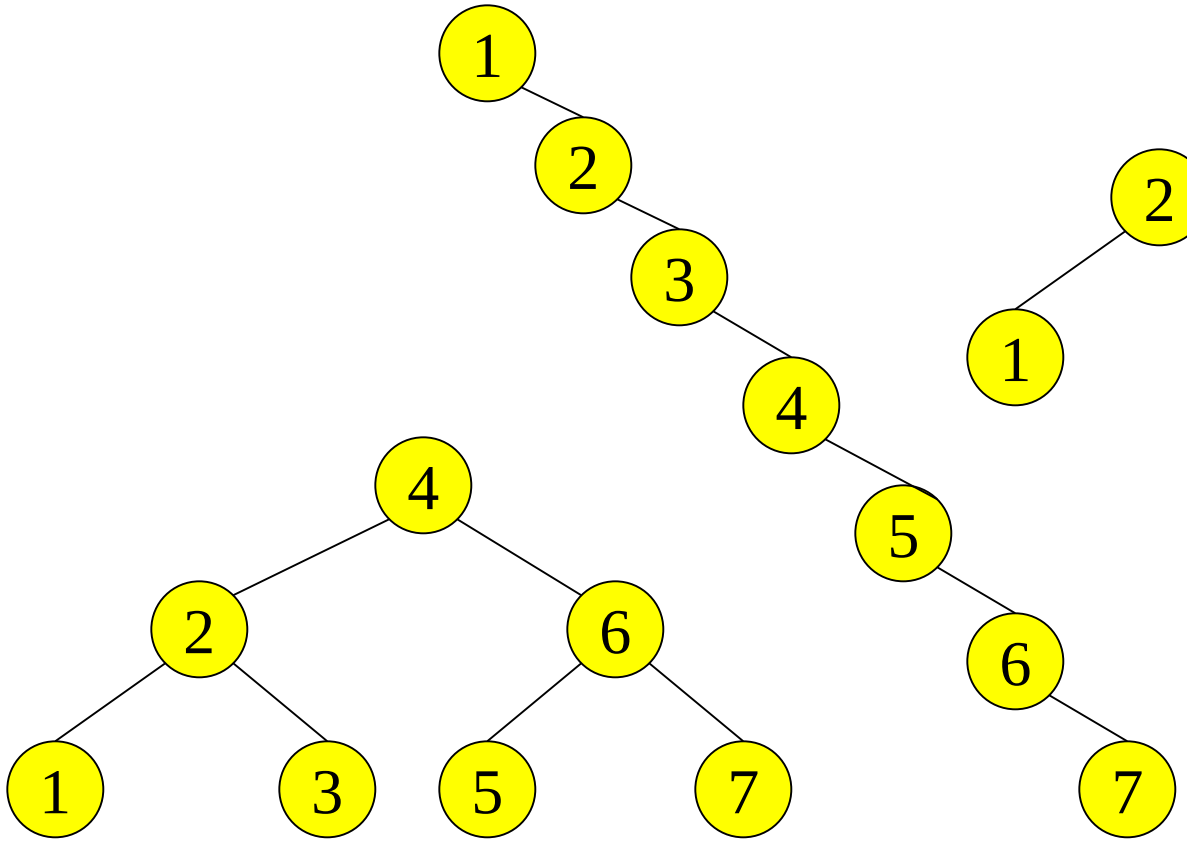
Árbol de búsqueda binario, el tiempo

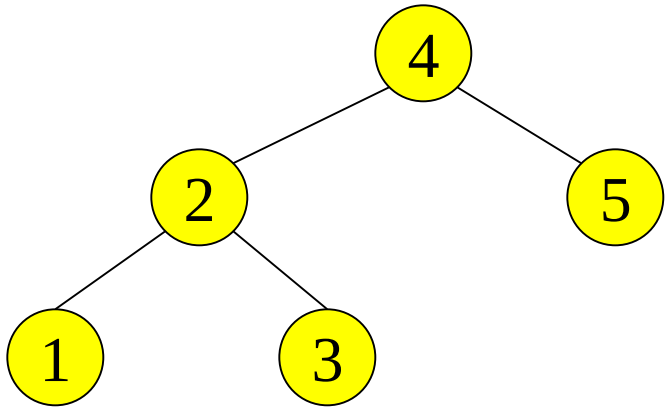
- Todas las operaciones sobre los ABB son O(d), donde d es la profundidad del árbol
- La d mínima es $d = \lfloor \log_2 N \rfloor$ para un árbol binario con N nodos
  - ¿Cuál es el mejor caso?
  - ¿Cuál es el peor caso?
- El mejor tiempo de ejecución en las operaciones de un ABB es O(log N)

Búsqueda en un árbol binario- el peor tiempo

- El peor caso en tiempo de ejecución es O(N)
  - › ¿Qué ocurre cuando se insertan los elementos en orden ascendente?
    - Inserte: 2, 4, 6, 8, 10, 12 en un BST vacío
  - › Problema: Falta de “balance”:
    - compare las profundidades del subárbol izquierdo y el derecho
  - › Árbol desbalanceado degenerado

BST balanceado y desbalanceado





¿está balanceado?

Estrategias para árboles balanceados

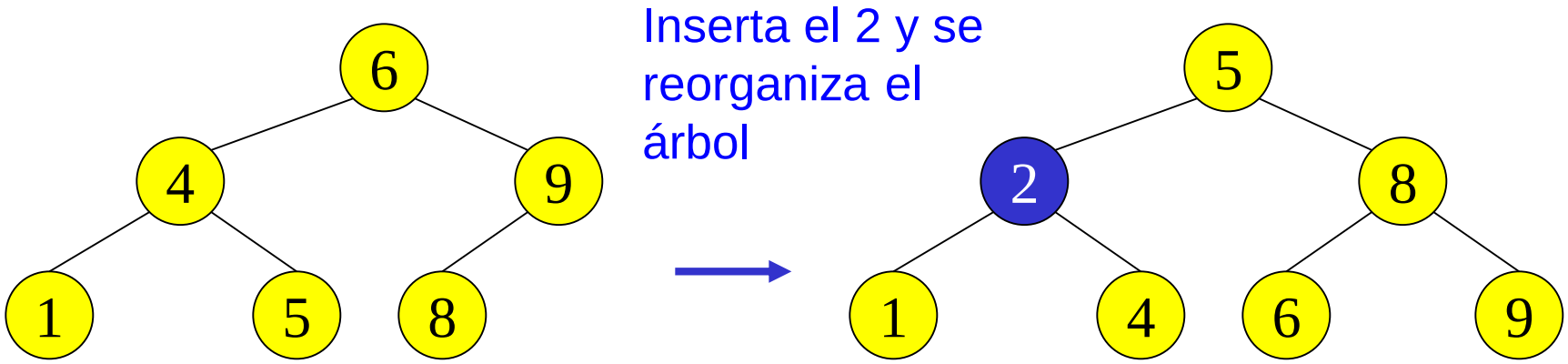
- No balancearlos
  - › Pueden terminar con nodos muy profundos
- Balance estricto
  - › El árbol deberá de estar siempre perfectamente balanceado
- Un buen balance
  - › Sólo se permite un “poco” de desbalanceo
- Balanceando durante el acceso
  - › Auto-balanceado

Balanceo de árboles binarios de búsqueda

- Existen muchos algoritmos para mantener los árboles balanceados
  - › Árboles Adelson-Velskii y Landis (AVL) (árboles balanceados en altura)
  - › Árboles biselados (Splay trees) y otros árboles auto-balanceados
  - › Árboles B (B-trees) y otros árboles de búsqueda multicaminos

Balance perfecto

- Se desea un árbol completo después de cada operación
  - › El árbol está lleno excepto posiblemente en el nivel inferior a la derecha
- Esto es costoso
  - › Por ejemplo, insertar el 2 en el árbol de la izquierda y después reconstruir el árbol completamente. Ojo: se eligió una nueva raíz

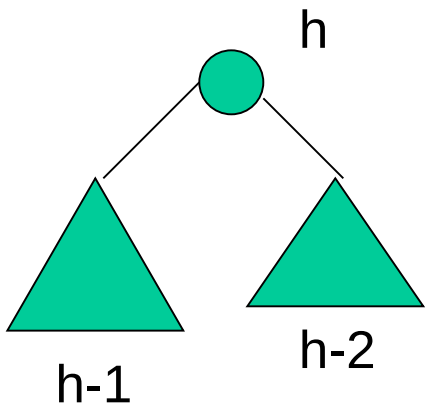


AVL – Bueno, pero no perfecto Balance

- Los árboles AVL son árboles de búsqueda balanceados en altura
- El factor de balance de un nodo es:
  - altura(subárbol izquierdo) - altura(subárbol derecho)
- Un árbol AVL tiene un factor de balance calculado en cada nodo
  - Para cada nodo, la altura de los subárboles izquierdo y derecho no pueden diferir por más de 1.
  - Se almacenan las alturas en cada nodo

Altura de un árbol AVL

- N(h) = número mínimo de nodos en un árbol AVL de altura h.
- Básico
  - N(0) = 1, N(1) = 2
- Inducción
  - N(h) = N(h-1) + N(h-2) + 1
- Solución aproximada (recordando el análisis
- de Fibonacci)
  - $N(h) \geq \phi^h \quad (\phi \approx 1.62)$
  - La relación dorada

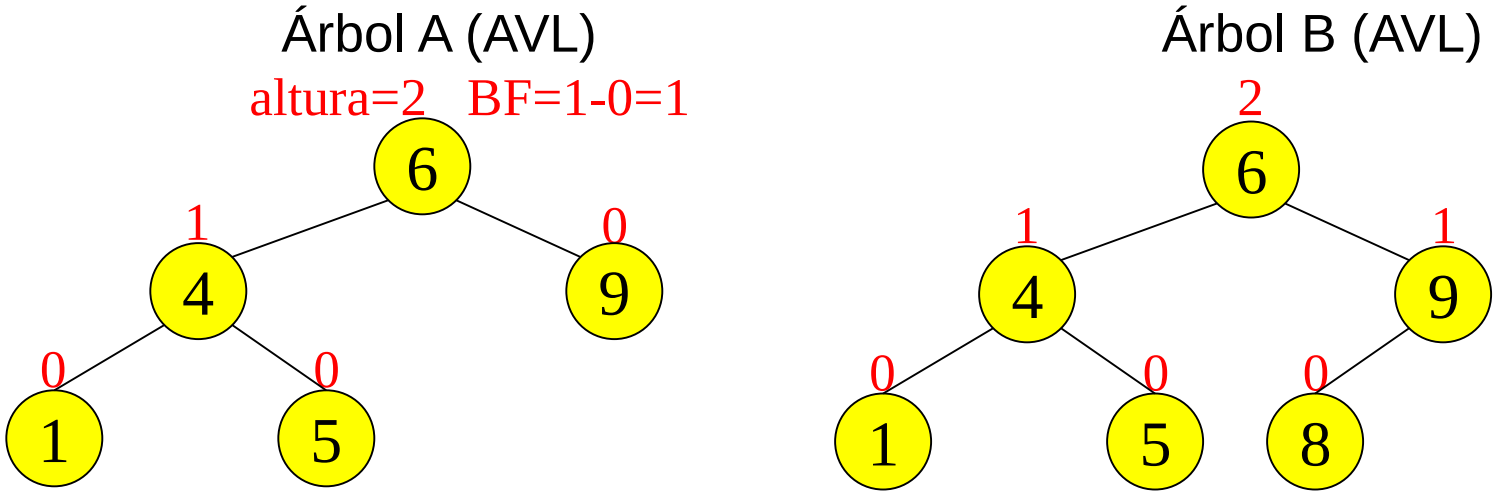


Altura de un árbol AVL

- $N(h) \geq \phi^h \quad (\phi \approx 1.62)$
- Suponga que tenemos n nodos en un árbol AVL de altura h.
  - › $n \geq N(h)$ (porque N(h) fue el mínimo)
  - › $n \geq \phi^h$ dado que $\log_\phi n \geq h$ (un árbol relativamente bien balanceado!!)
  - › $h \leq 1.44 \log_2 n$ (i.e., O(*log*n))

$$\log_\phi n = \log_2 n / \log_2 \phi$$

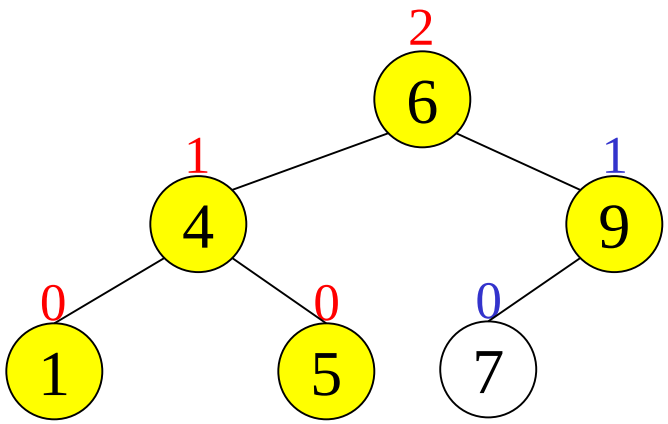
Altura del nodo



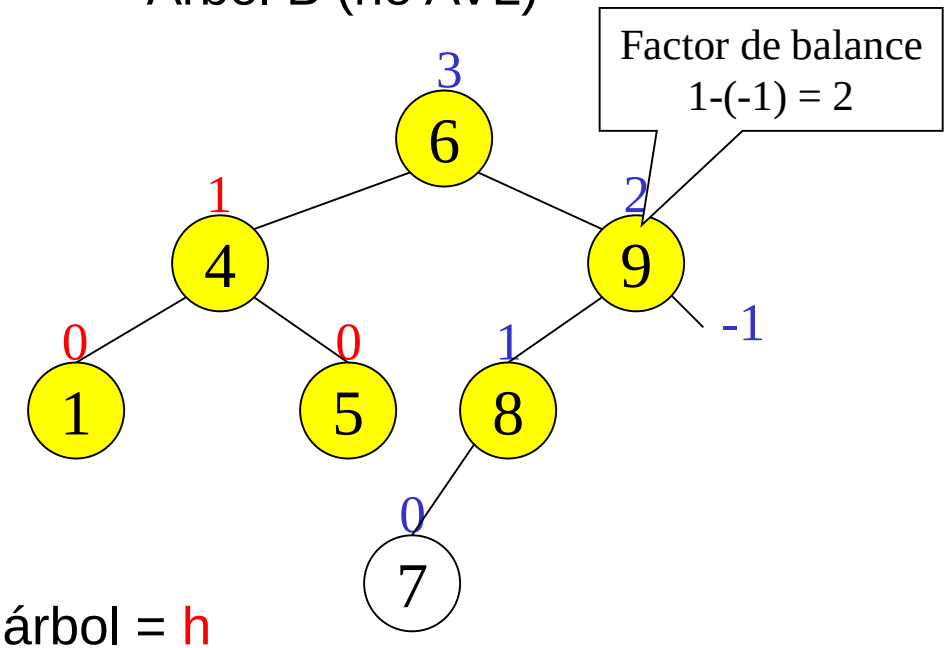
Altura del nodo= h
Factor de balance = h_{izq}-h_{der}
Altura vacío = -1

Altura del nodo después de insertar el 7

Árbol A (AVL)



Árbol B (no AVL)



Altura del árbol = h

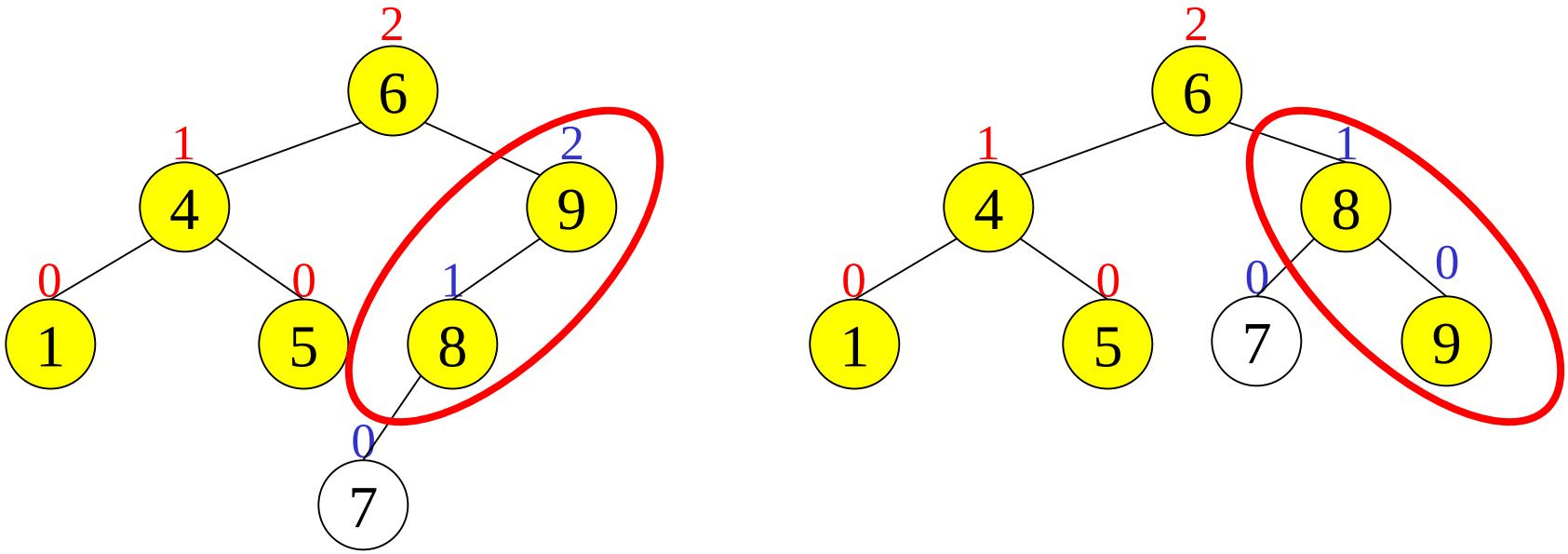
Factor de balance = h_{izq}-h_{der}

Altura vacío = -1

Inserción y rotación en árboles AVL

- La operación de inserción puede causar que el factor de balance alcance 2 o -2 en algún nodo
  - › Solamente los nodos en la trayectoria del punto de inserción al nodo raíz pueden cambiar de altura
  - › Después de insertar, regresar hacia el nodo raíz nodo por nodo actualizando las alturas
  - › Si un nuevo factor de balance (h_{izq}-h_{der}) es 2 o –2, se deberá ajustar el árbol con una rotación alrededor del nodo que presenta el desbalance

Rotación simple en un árbol AVL



Inserción en árboles AVL

Dado el nodo α que necesita rebalanceo.

Se tienen 4 casos:

Casos externos (requieren una rotación simple) :

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

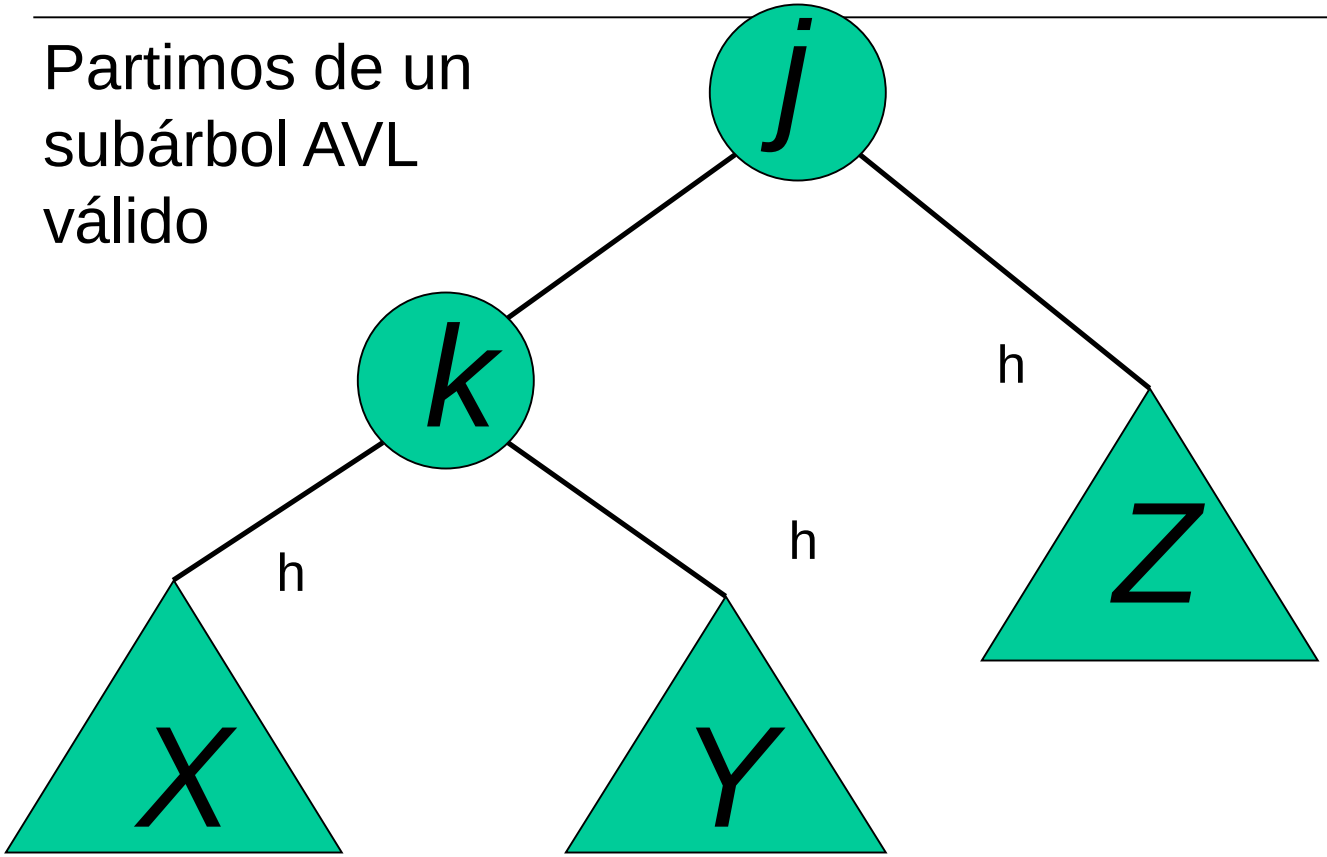
Casos internos (requieren una doble rotación) :

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

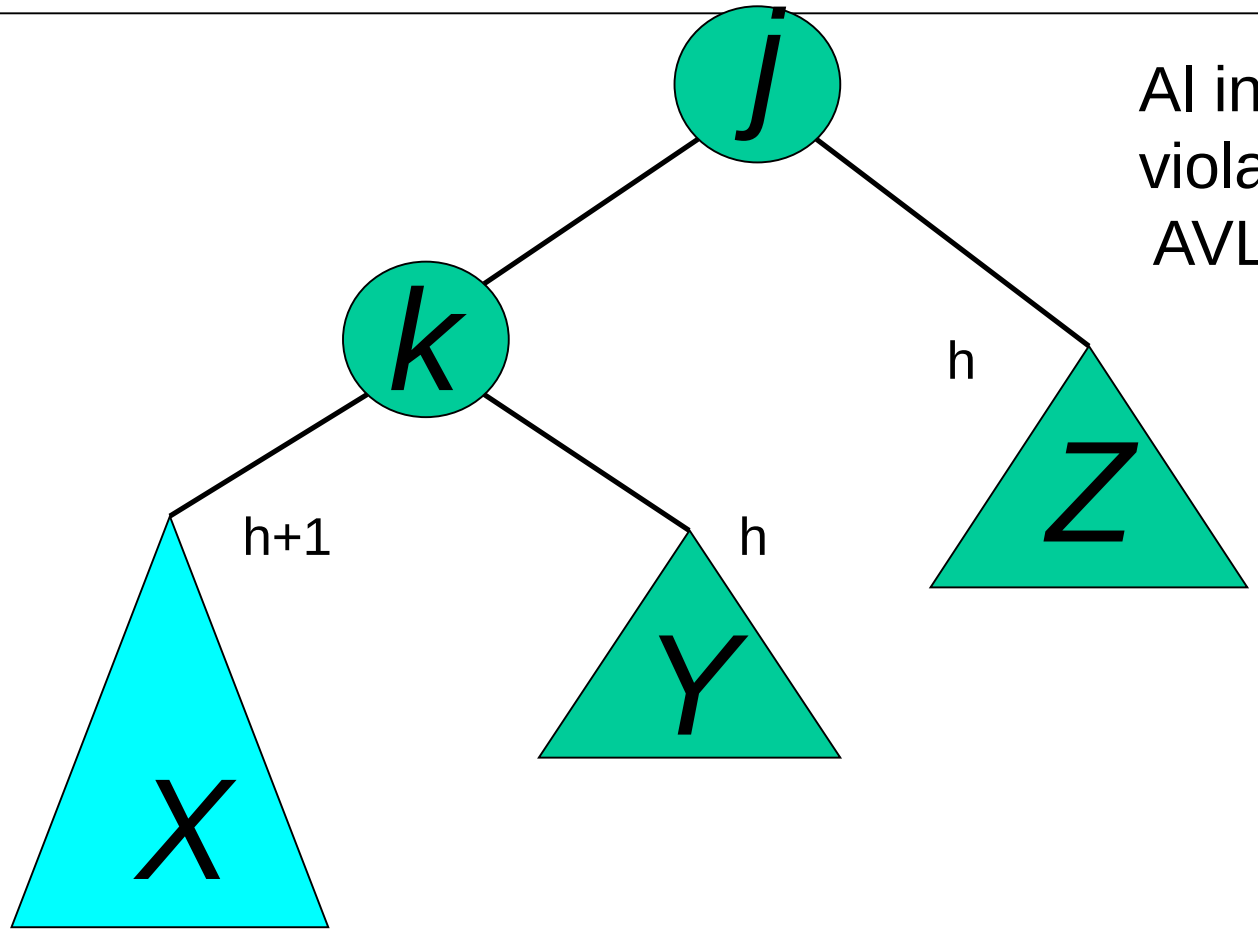
El rebalanceo se realiza utilizando cuatro algoritmos distintos de rotación.

Inserción en un AVL: Caso externo

Partimos de un subárbol AVL válido

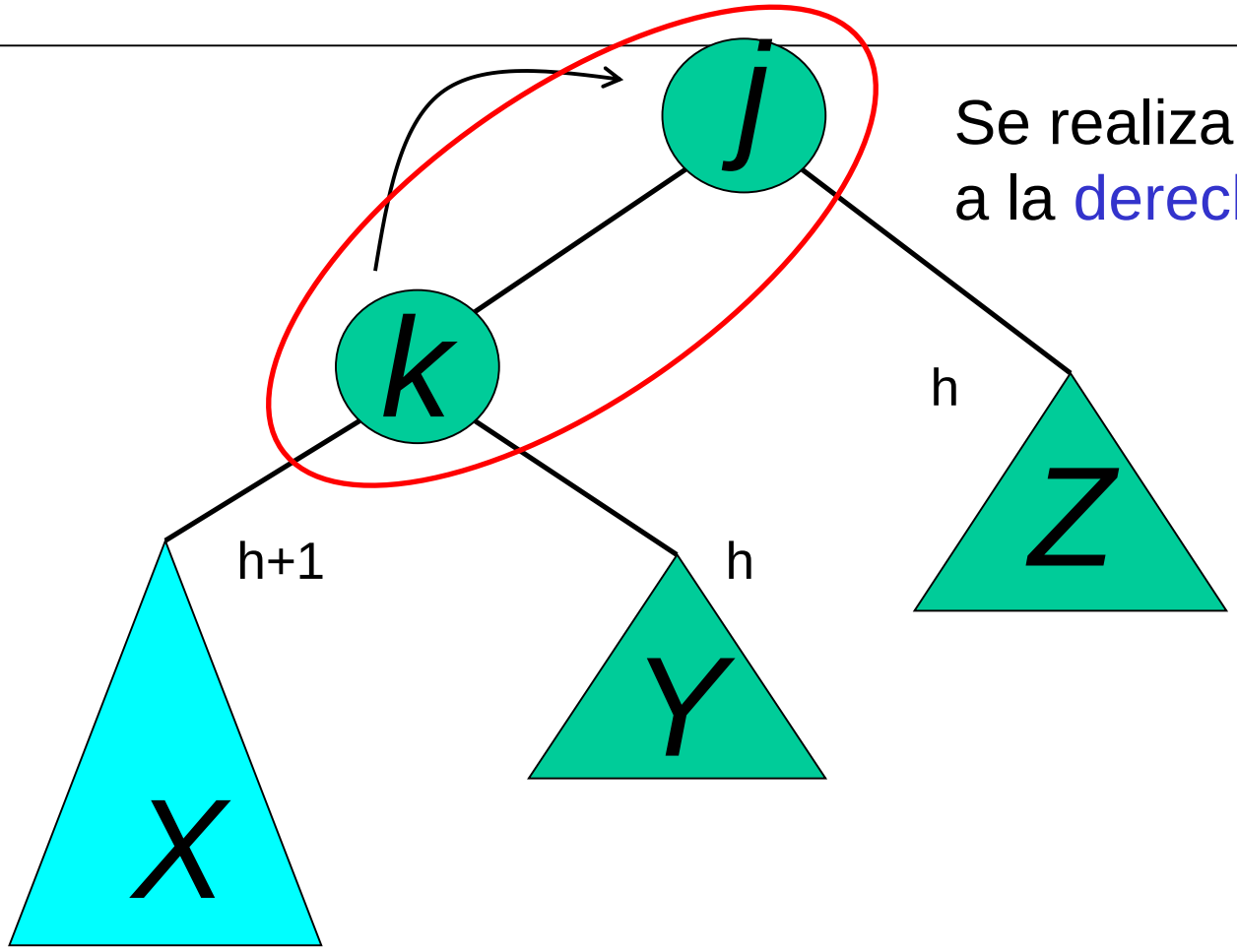


Inserción en un AVL: Caso externo



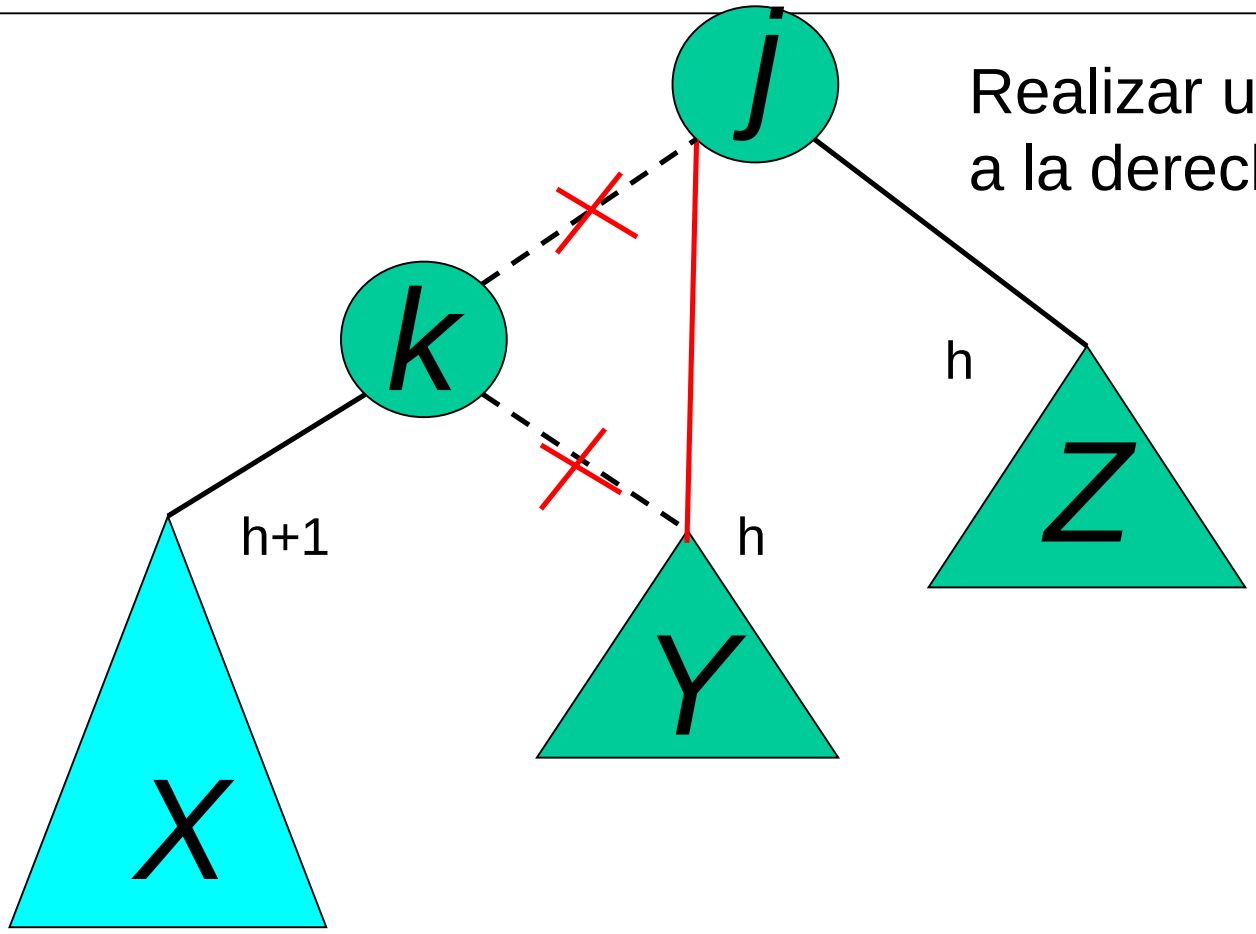
Al insertar en X, se viola la propiedad de AVL del nodo j

Inserción en un AVL: Caso externo



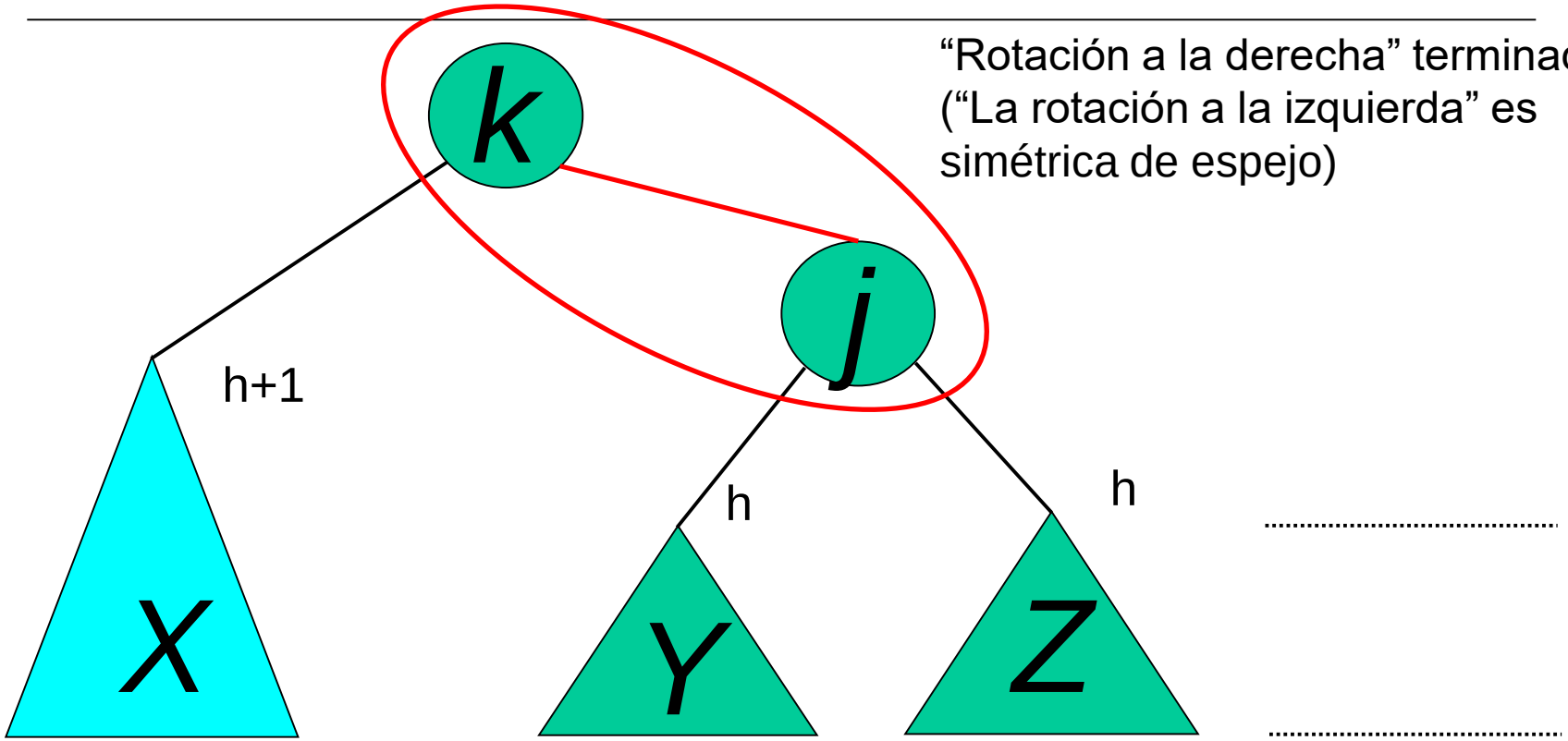
Se realiza una rotación a la derecha

Rotación simple a la derecha



Realizar una rotación a la derecha

Caso externo completo



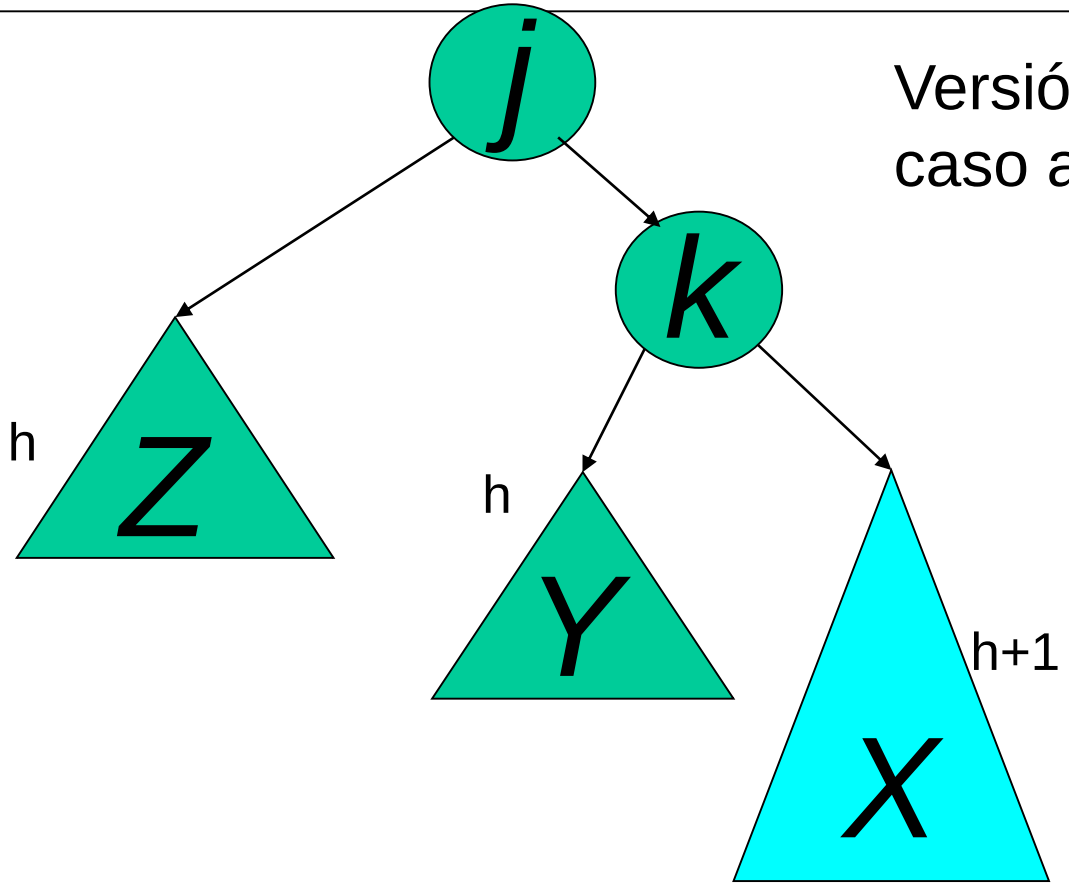
"Rotación a la derecha" terminada!
("La rotación a la izquierda" es simétrica de espejo)

La propiedad de AVL se restauró

Caso externo por la izquierda

h(j)=h+3
h(k)=h+2

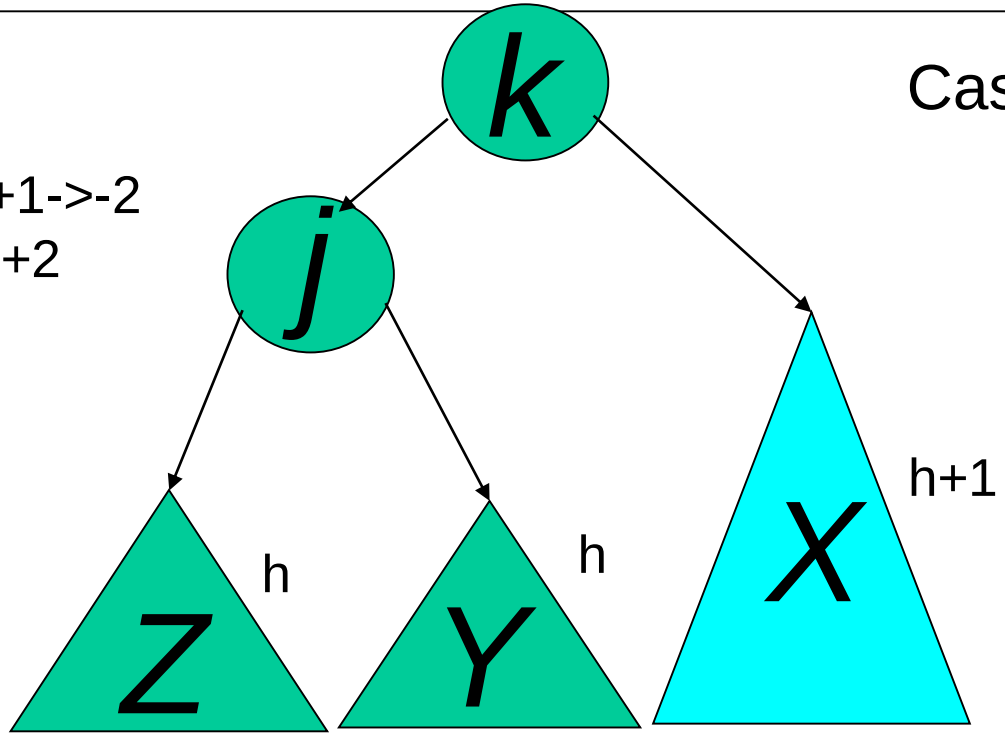
Versión Espejo del caso anterior



Caso externo por la izquierda

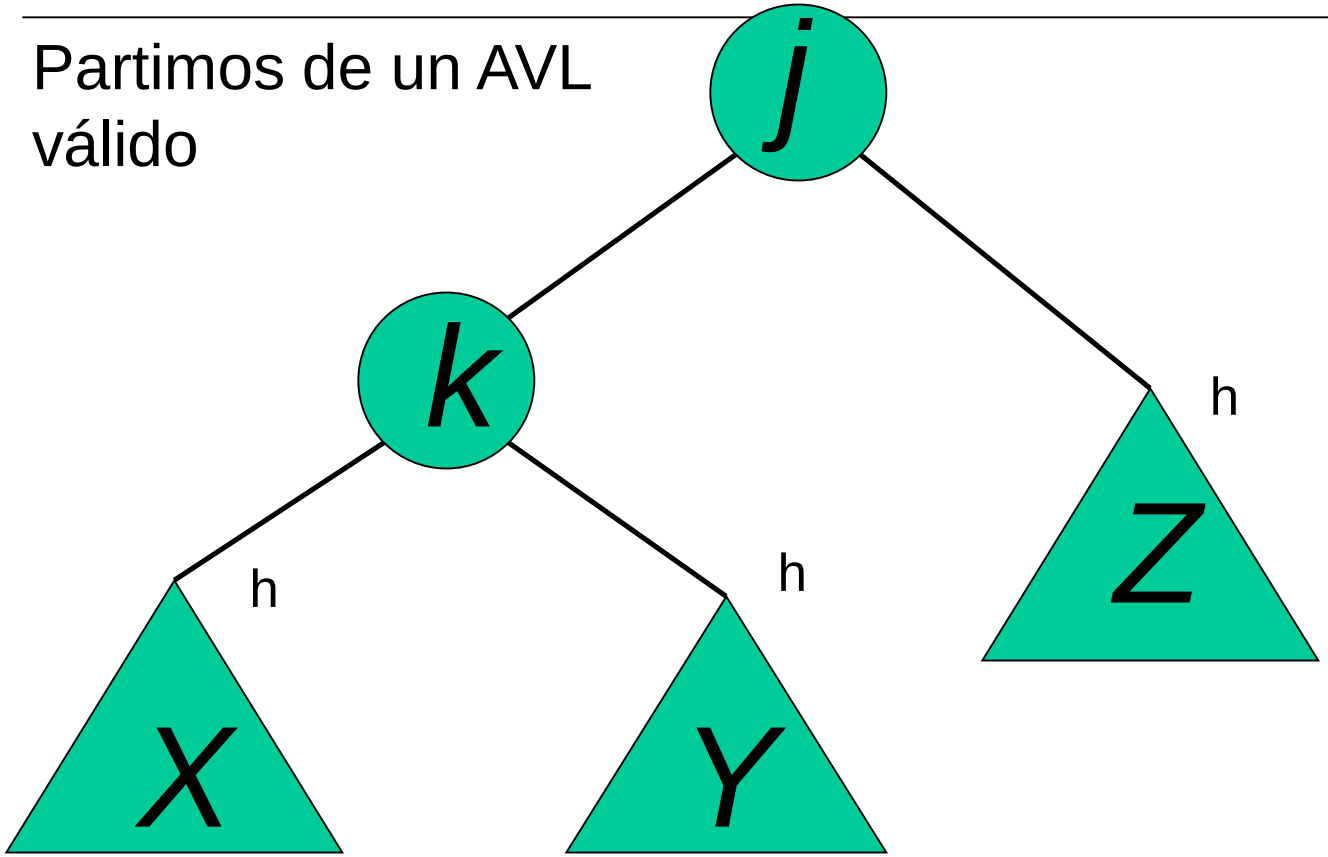
Caso terminado

h(j)=h+3->h+1->-2
h(k)=h+2->h+2



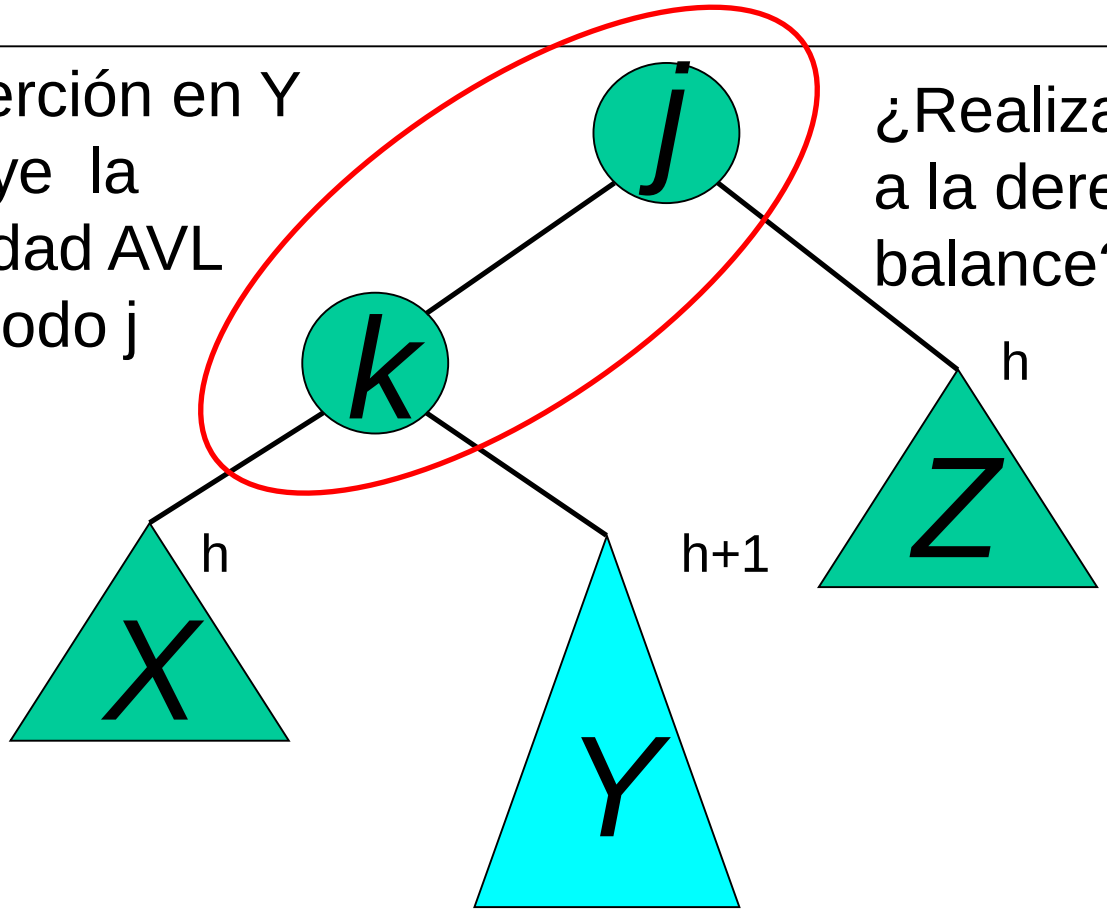
Inserción en un AVL: caso interno

Partimos de un AVL válido



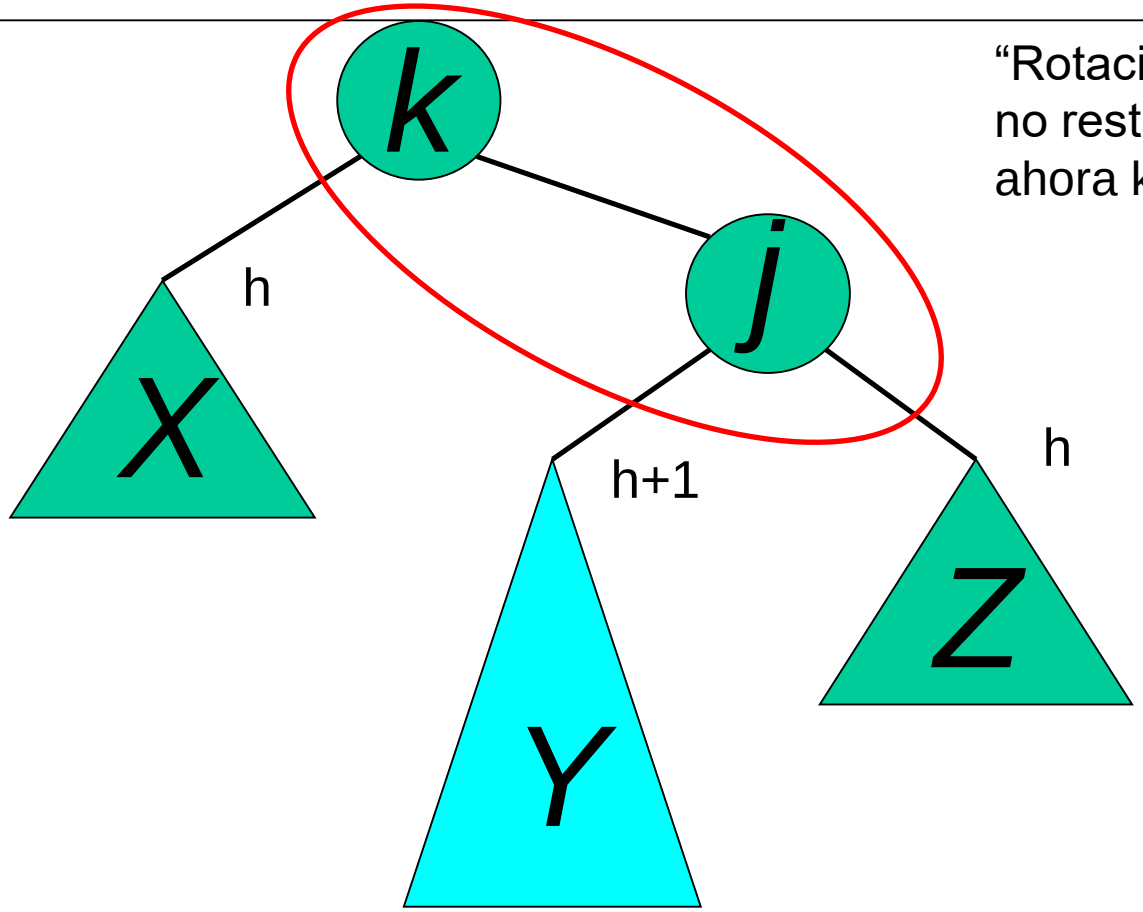
Inserción en un AVL: caso interno

La inserción en Y destruye la propiedad AVL en el nodo j



¿Realizar una rotación a la derecha restaura el balance?

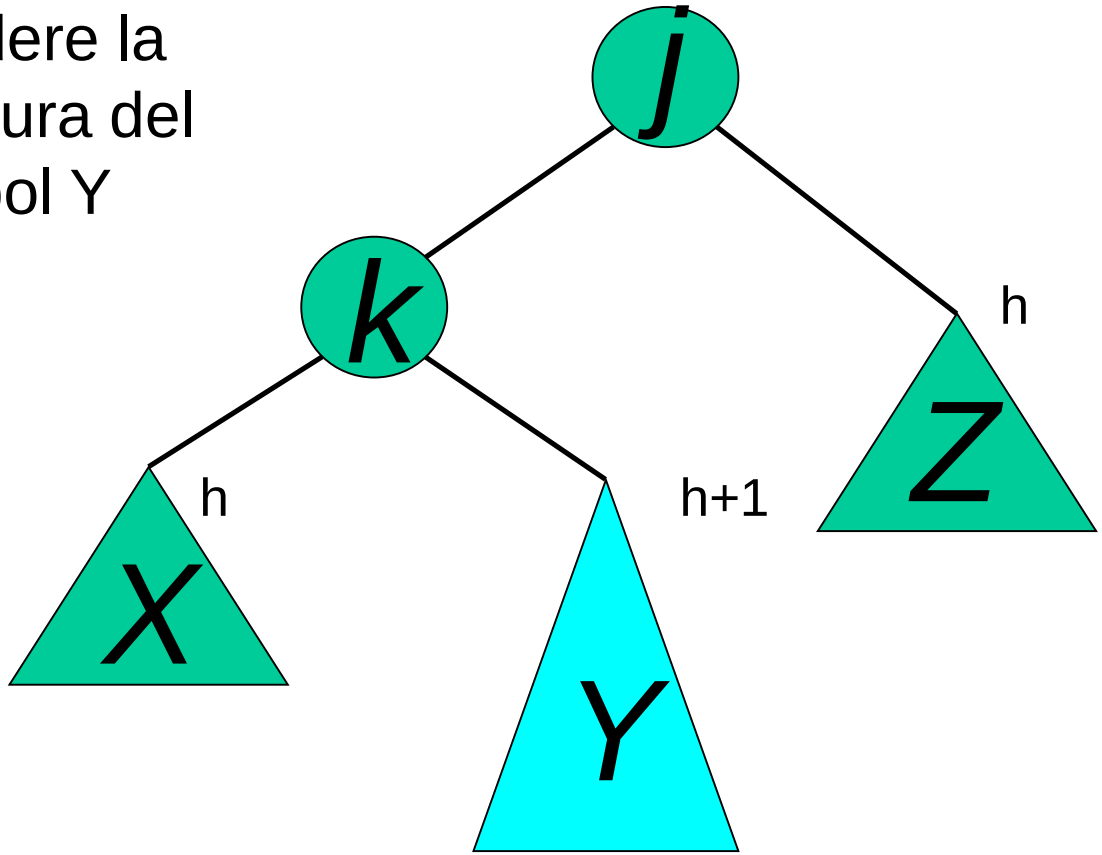
Inserción en un AVL: caso interno



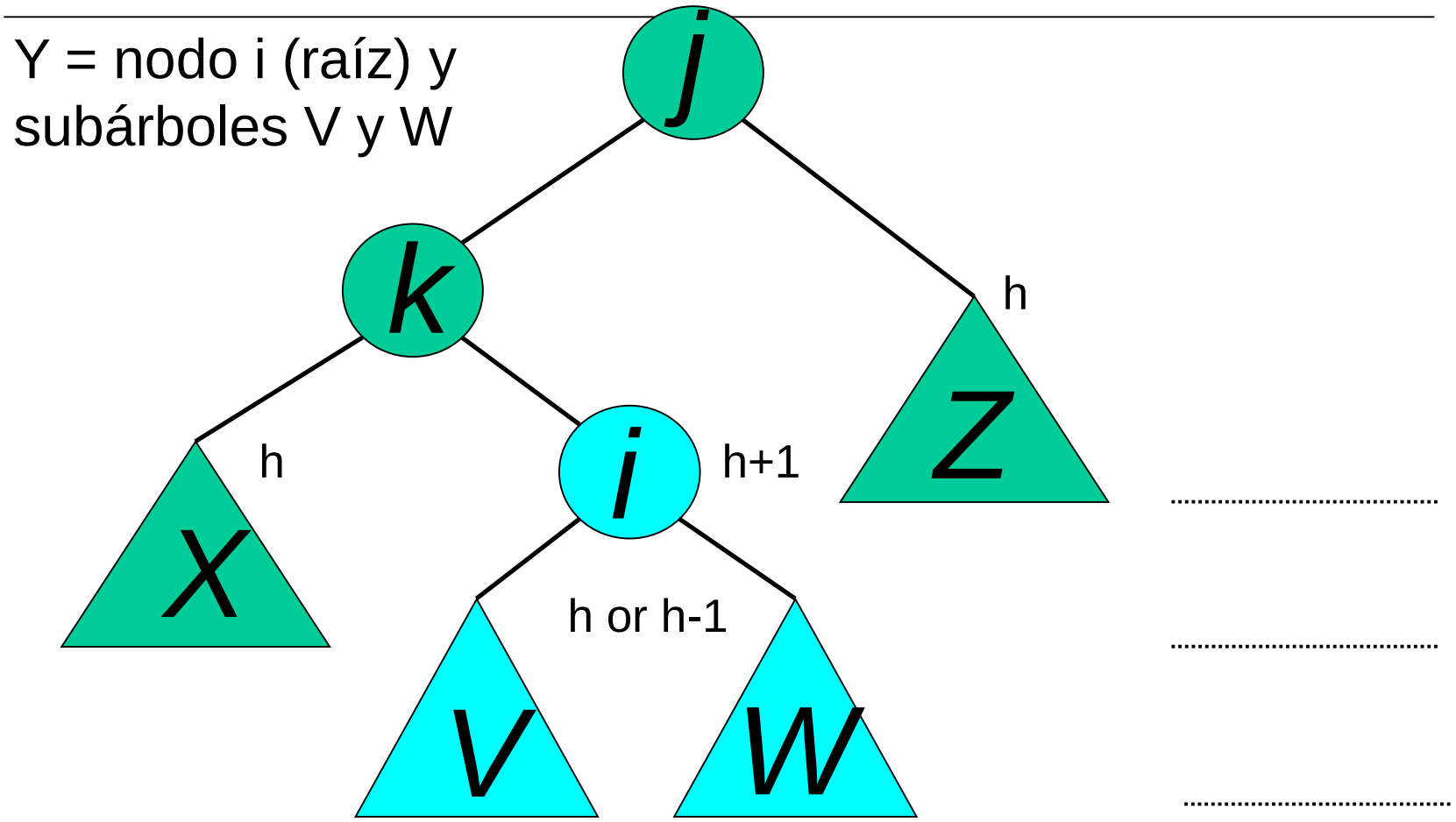
"Rotación a la derecha"
no restaura el balance…
ahora k está desbalancedo

Inserción en un AVL: caso interno

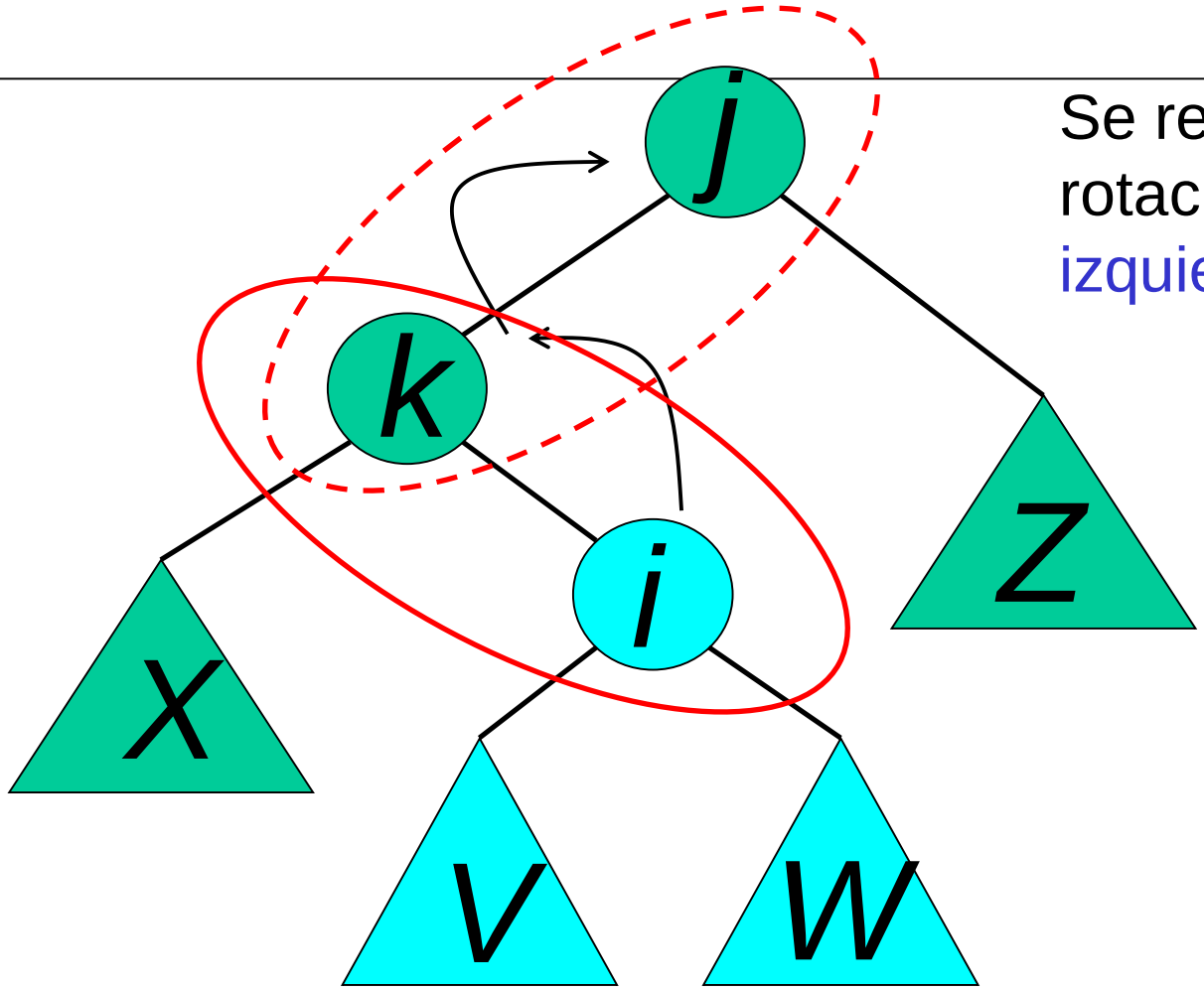
Considere la estructura del subárbol Y



Inserción en un AVL: caso interno

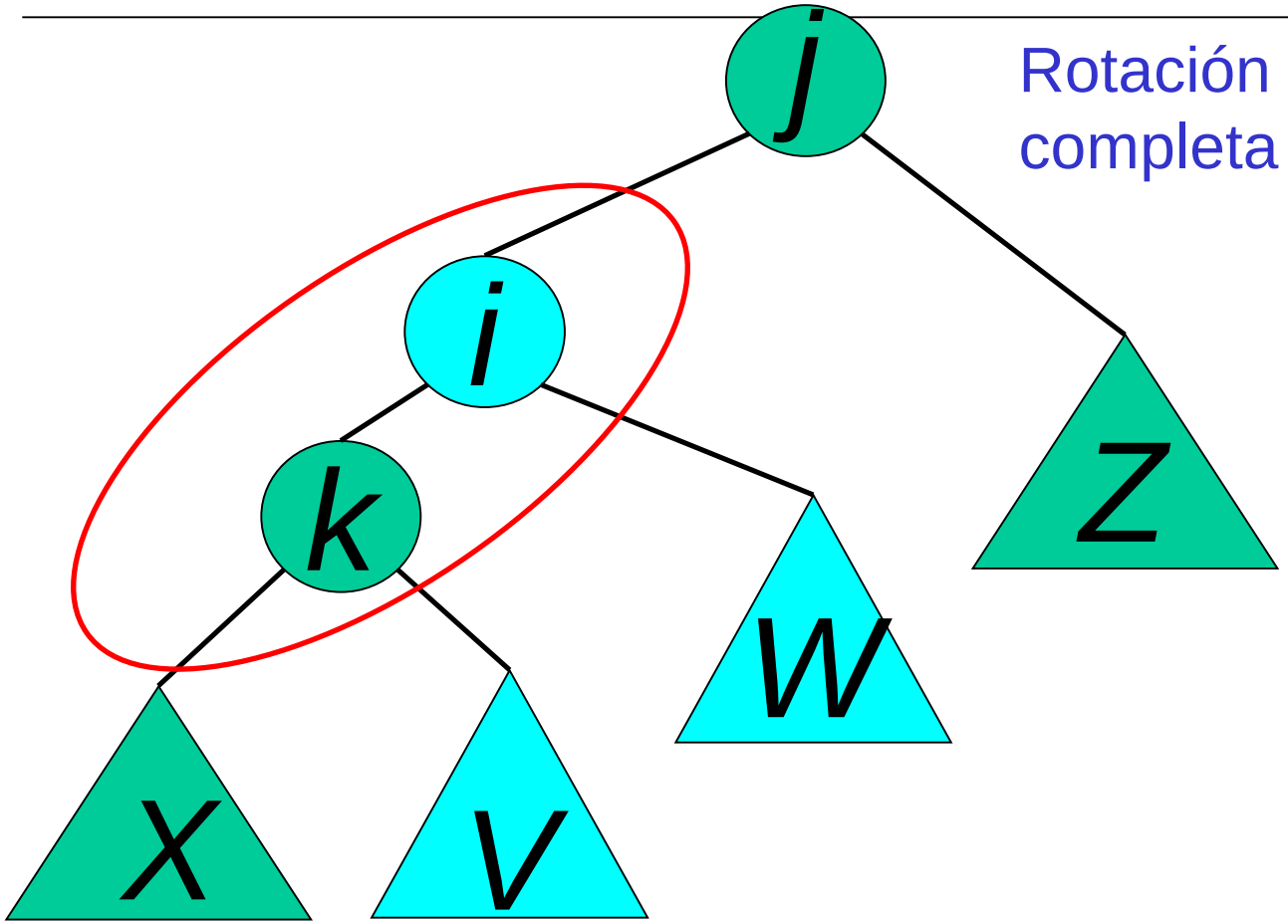


Inserción en un AVL: caso interno



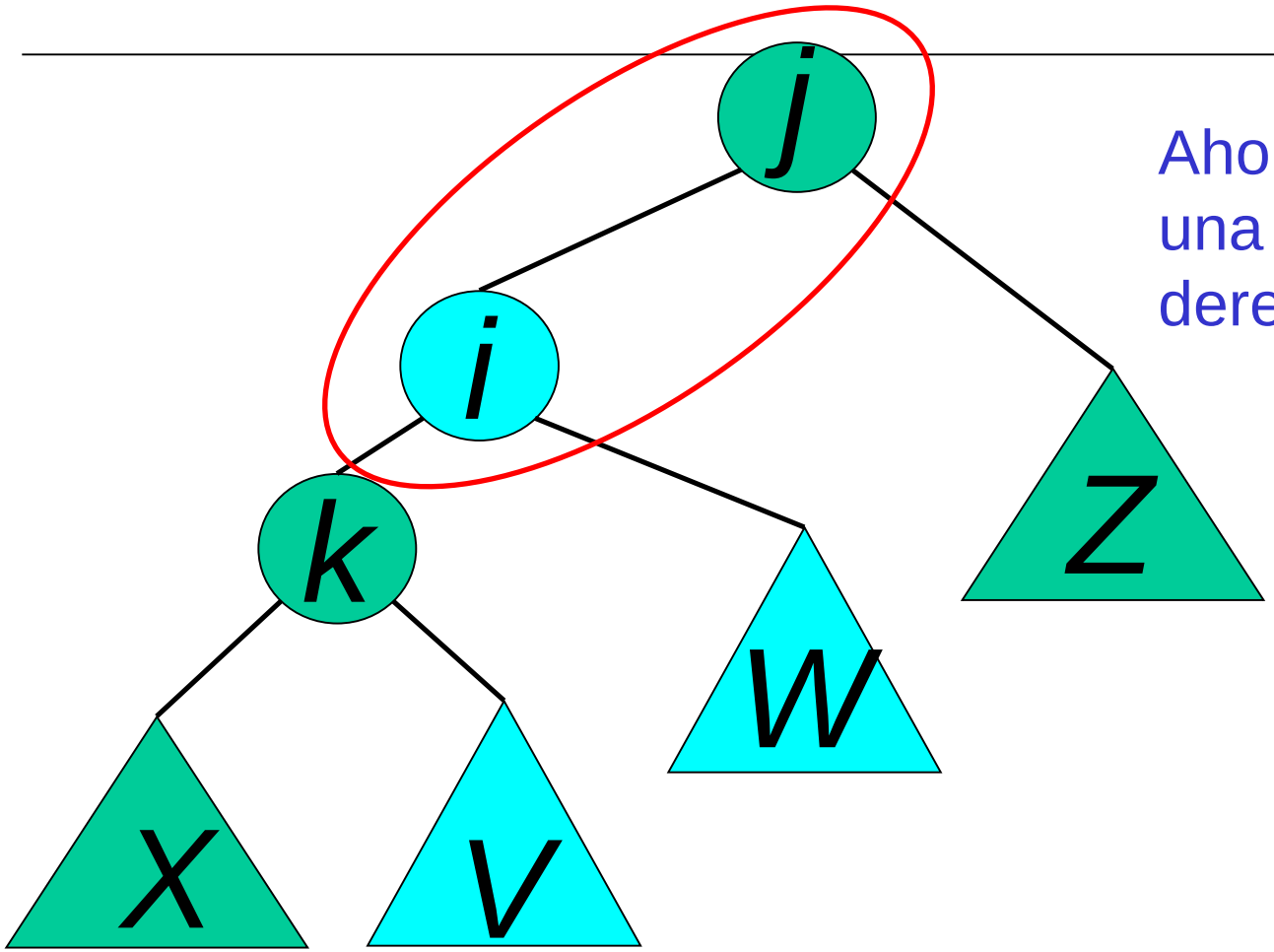
Se realiza una doble rotación derecha-izquierda . . .

Doble rotación : la primera



Rotación a la izquierda completa

Doble rotación: la segunda

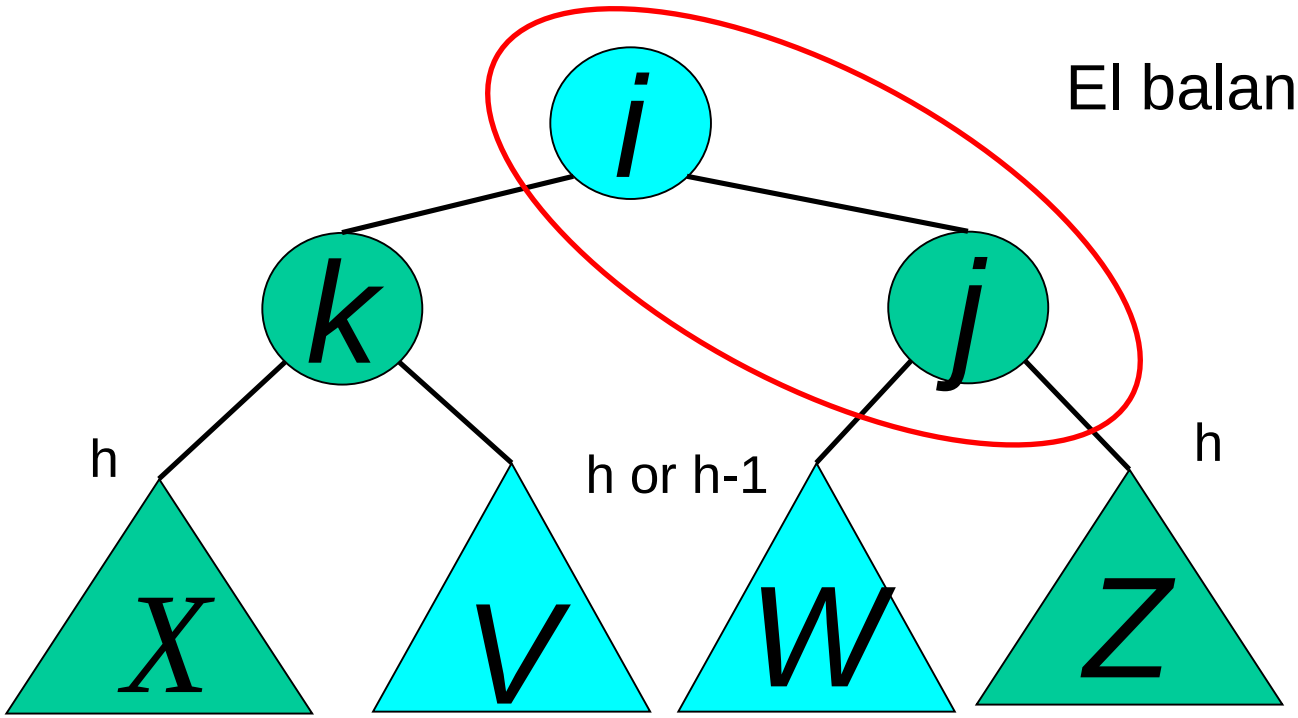


Ahora se hace una rotación a la derecha

Doble rotación: la segunda

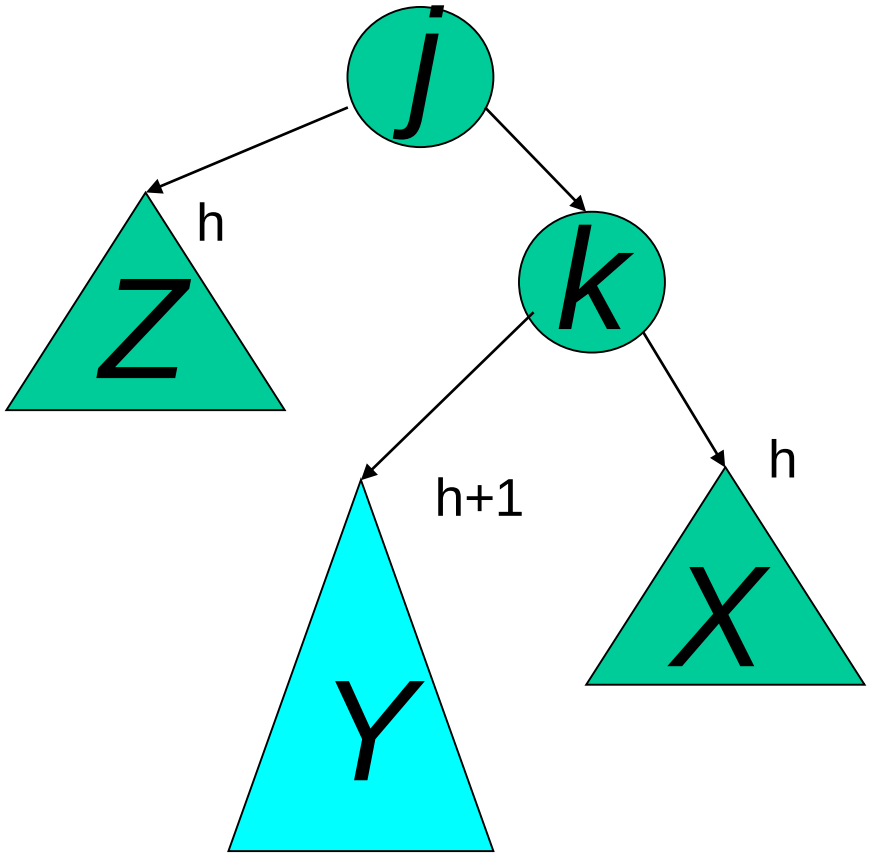
Rotación a la derecha completa

El balance se restauró



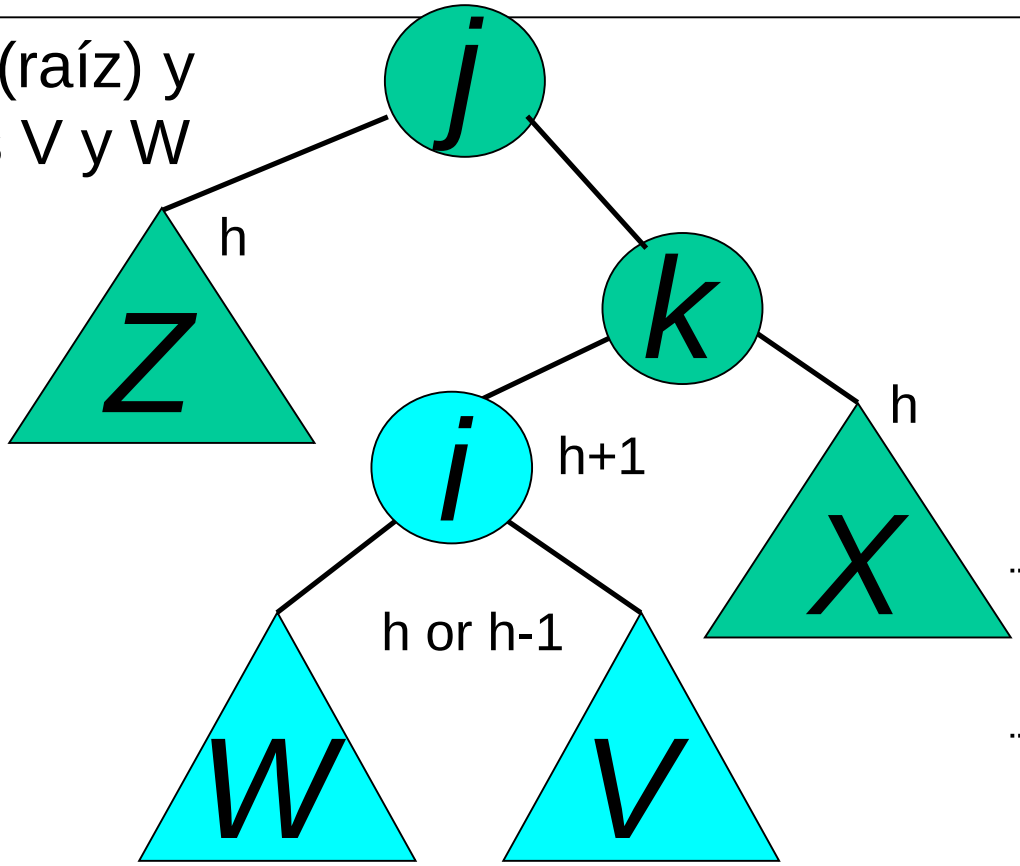
Caso interno, espejo del anterior

Considere la estructura del subárbol Y



Caso interno, espejo

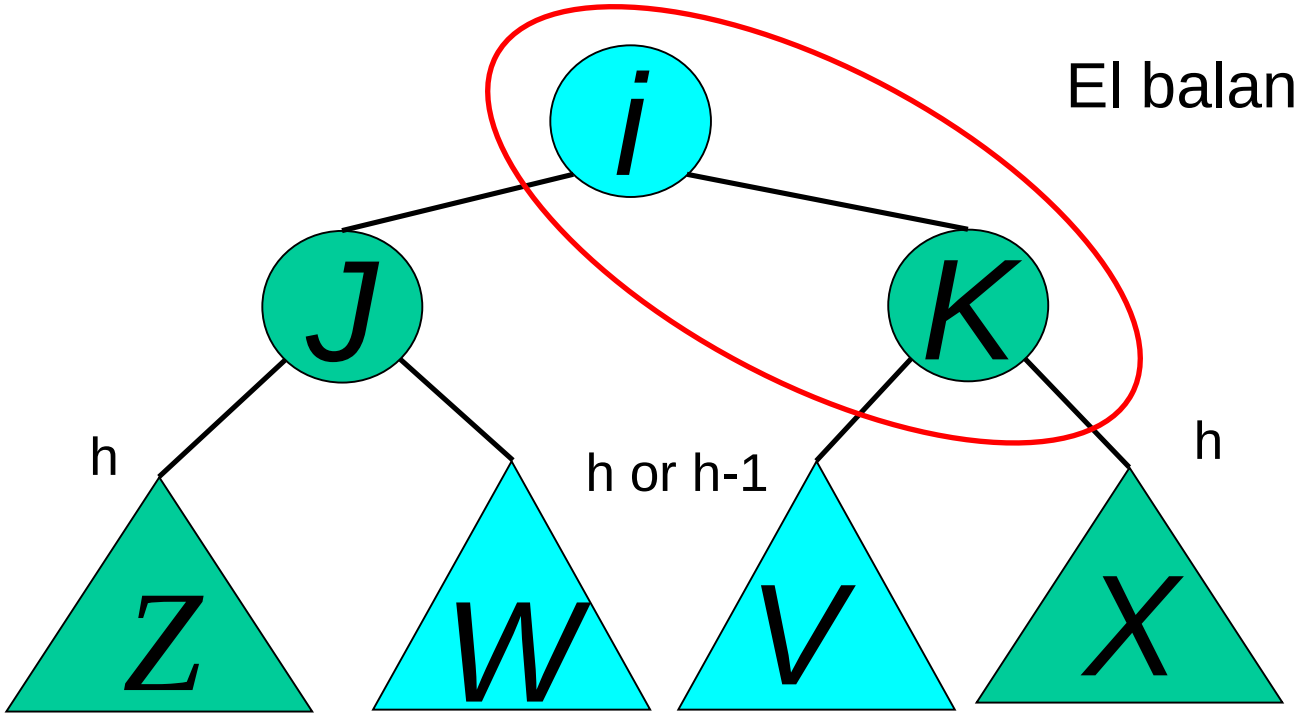
Y = nodo i (raíz) y subárboles V y W



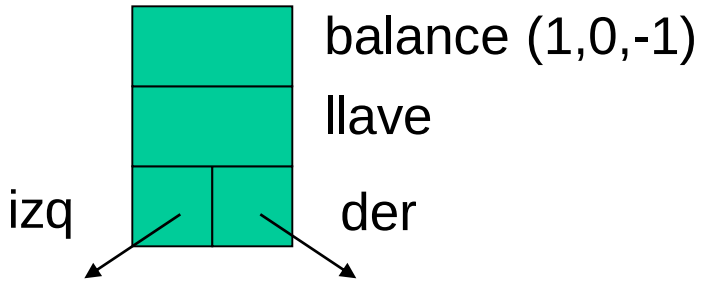
Caso interno espejo

Rotación a la derecha completa

El balance se restauró



Implementación



No se necesita almacenar la altura; sólo la diferencia en la altura, i.e. el factor de balance;

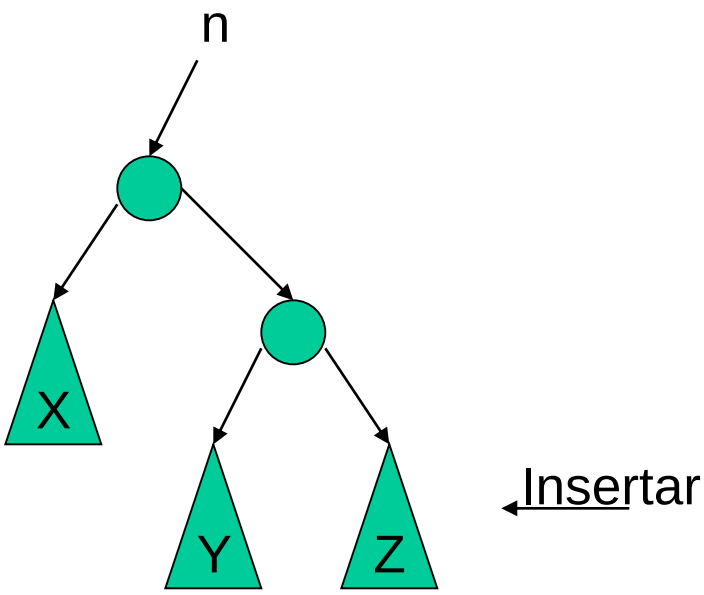
Este deberá de modificarse en la trayectoria de inserción aún si si no se realizan rotaciones

Una vez que se ha realizado una rotación, (simple o doble) ya no es necesario continuar regresando hacia arriba en el árbol

Rotación simple

```
RotarAlaDerecha(Nodo n) {
Nodo p;
p = n.der;
n.Der = p.izq;
p.Izq = n;
N = p
}
```

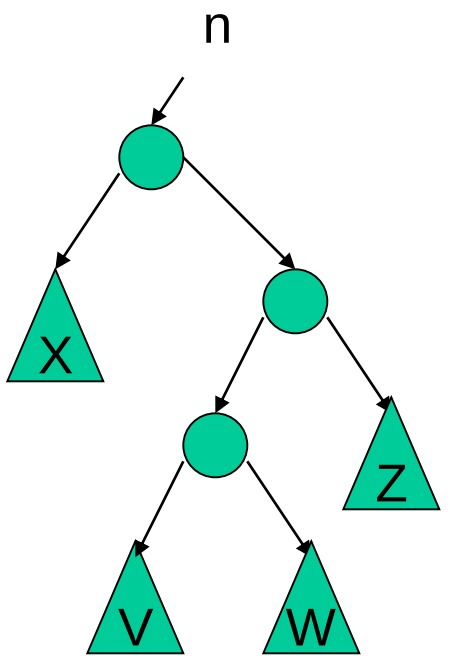
También es necesario recalcular los factores de balance para los nodos n y p



Rotación doble

- Implemente la doble rotación en dos líneas.

```
DobleRotacionDerecha(Nodo n) {
????
}
```



Inserción en áboles AVL

- Insertar al nivel hoja (como en todos los BST)
  - › Solamente los nodos en la trayectoria desde el punto de inserción a al raíz posiblemente cambien en altura
  - › Después de insertar, regrese hacia arriba al nodo raíz nodo por nodo, actualizando las alturas
  - › Si un nuevo factor de balance (h_{izq}-h_{der}) es 2 o –2, ajuste el árbol por rotación alrededor del nodo

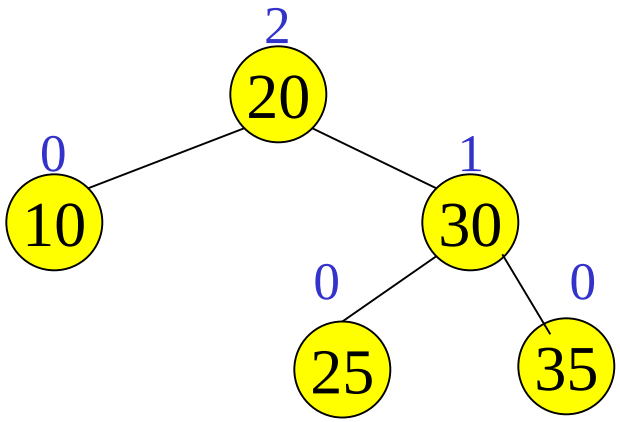
Insertar en un BST

```
public int Insert(Nodo T, Dato x) {
if (T == null) {
  T = new Nodo();
  T.dato = x; return 1;//the links to
}
  if (T.dato == x) {return 0;} //Duplicado
  if (T.dato > x) {return Insert(T.izq, x);}
  if (T.dato < x) {return Insert(T.der, x);}
  return 0; //Sólo para que no marque error
}
```

Insertar en un árbol AVL

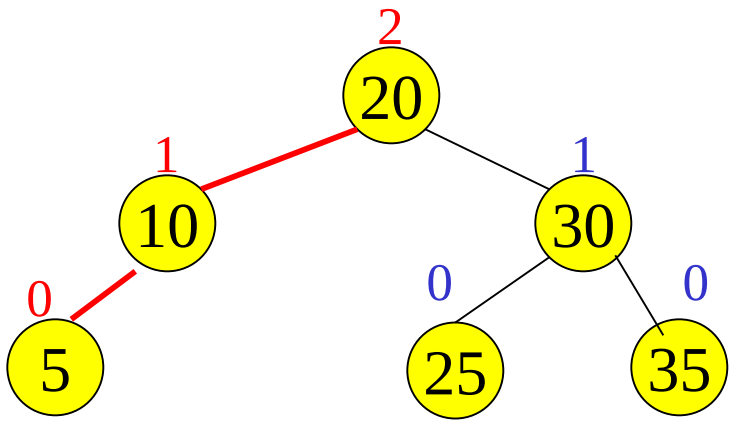
```
Insert(T : reference tree pointer, x : element) : {
if T = null then
  {T := new tree; T.data := x; height := 0; return;}
case
  T.data = x : return ; //Duplicate do nothing
  T.data > x : Insert(T.left, x);
               if ((height(T.left)- height(T.right)) = 2){
                  if (T.left.data > x ) then //outside case
                         T = RotatefromLeft (T);
                  else                       //inside case
                         T = DoubleRotatefromLeft (T);}
  T.data < x :  Insert(T.right, x);
                code similar to the left case
Endcase
  T.height := max(height(T.left),height(T.right)) +1;
  return;
}
```

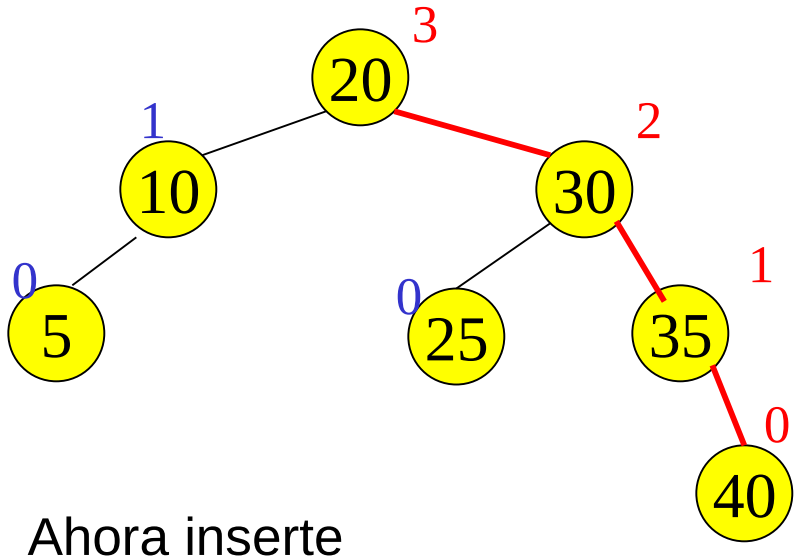
Ejemplo de inserción en un árbol AVL



Inserte 5, 40

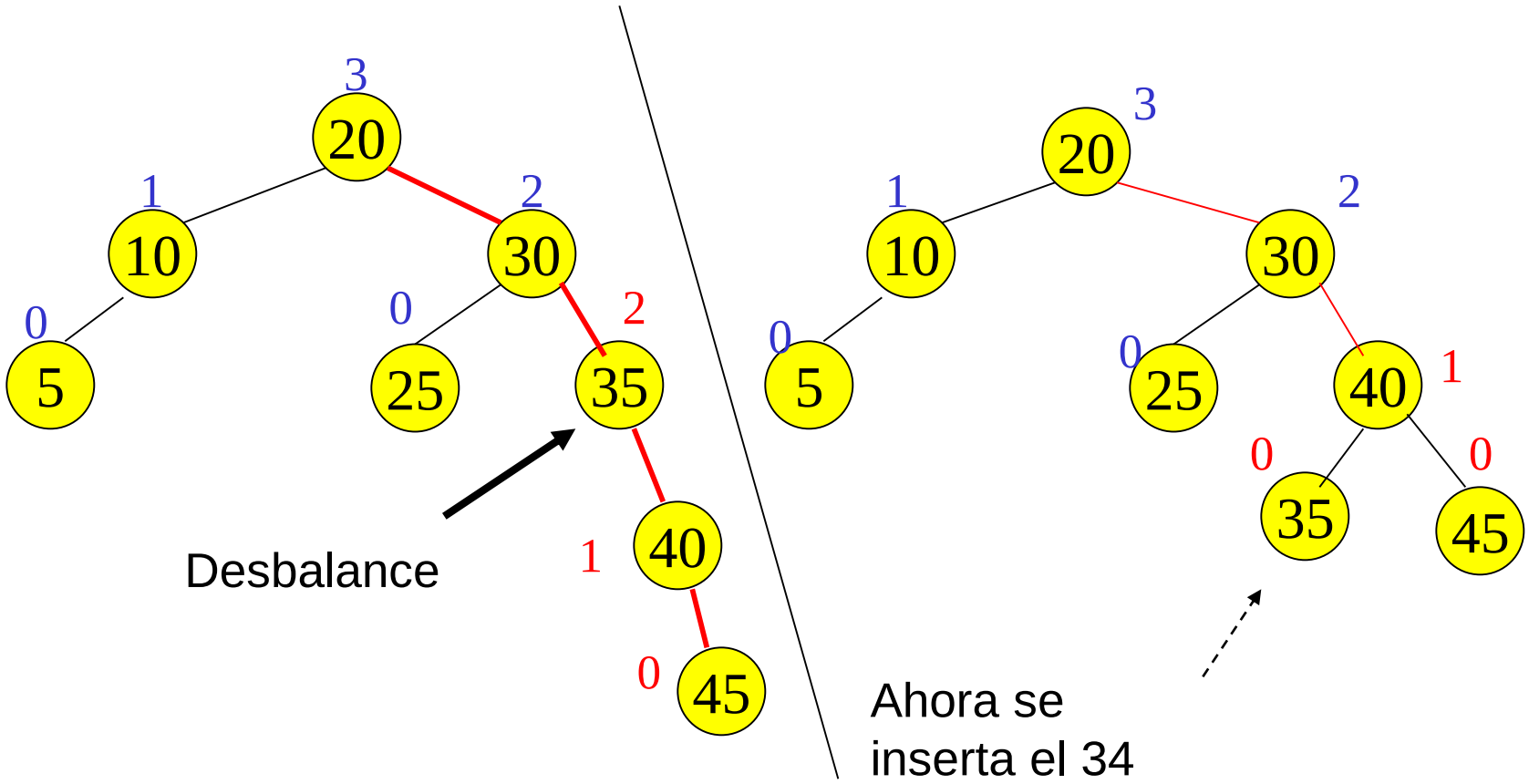
Ejemplo de inserción en un árbol AVL



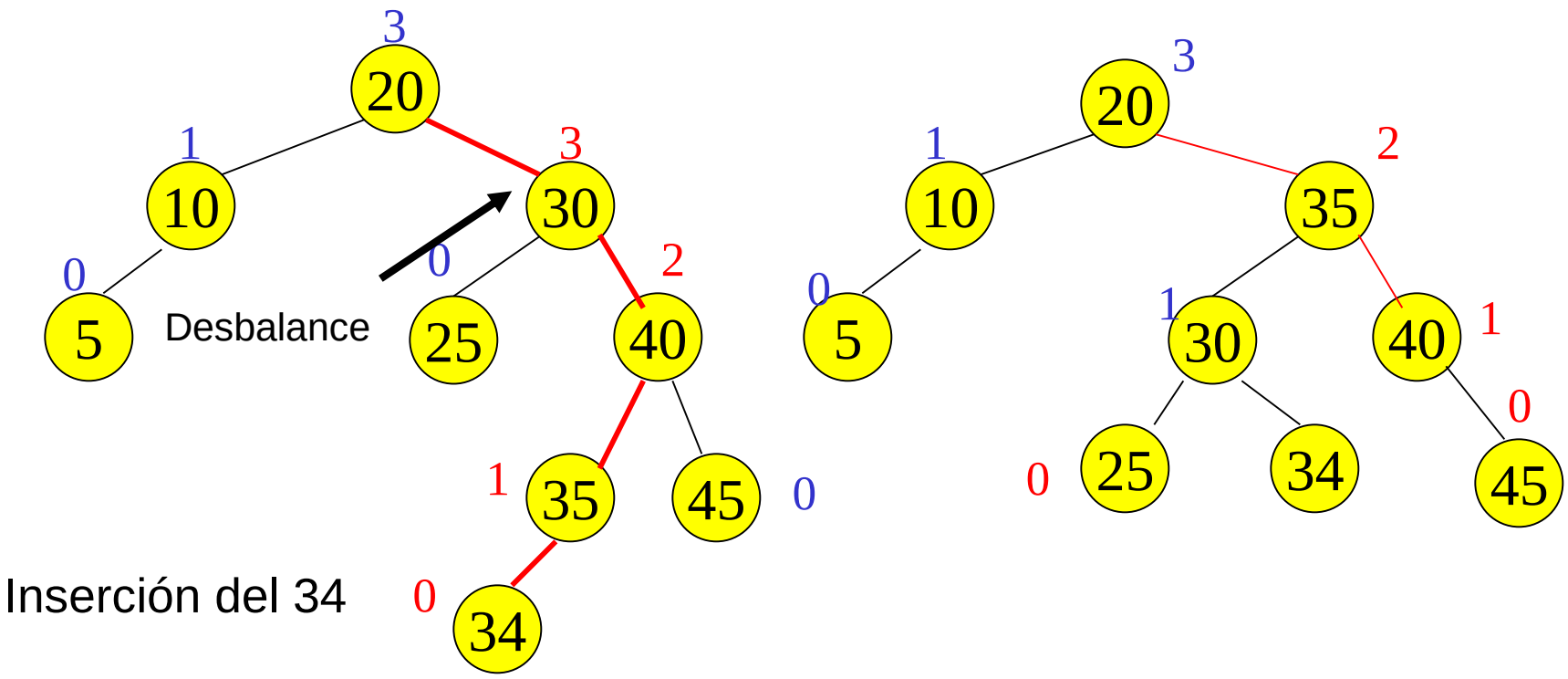


Ahora inserte el 45

Rotación simple (caso externo)



Rotación doble (caso interno)



Borrado en un árbol AVL

- Es similar pero más complejo que la inserción
  - Son necesarias las rotaciones y las doble rotaciones para rebalancear
  - El desbalance se puede propagar hacia arriba por lo que puede ser necesario realizar varias rotaciones.

Pros and Cons de los árboles AVL

A favor del uso de los árboles AVL:

1. La búsqueda es O(log N) ya que los árboles AVL están siempre balanceados.
2. La inserción y el borrado también son O(log N)
3. El balanceo en altura agrega a lo más un factor constante en la velocidad de inserción.

En contra del uso de los árboles AVL:

1. Difícil de programar y depurar; más espacio para el factor de balance.
2. Asíntoticamente rápido pero el rebalanceo cuesta tiempo.
3. La mayoría de las búsquedas grandes se realizan en sistemas de base de datos almacenadas en disco y se utilizan otras estructuras (e.g., árboles B).
4. Puede estar bien tener un O(N) para una sóla operación si el tiempo total de ejecución para mucho accesos consecutivos es rápido (e.g, Splay trees árboles biselados).

Solución de la doble rotación

```
DoubleRotateFromRight(Nodo n) {
RotateFromLeft(n.right);
RotateFromRight(n);
}
```