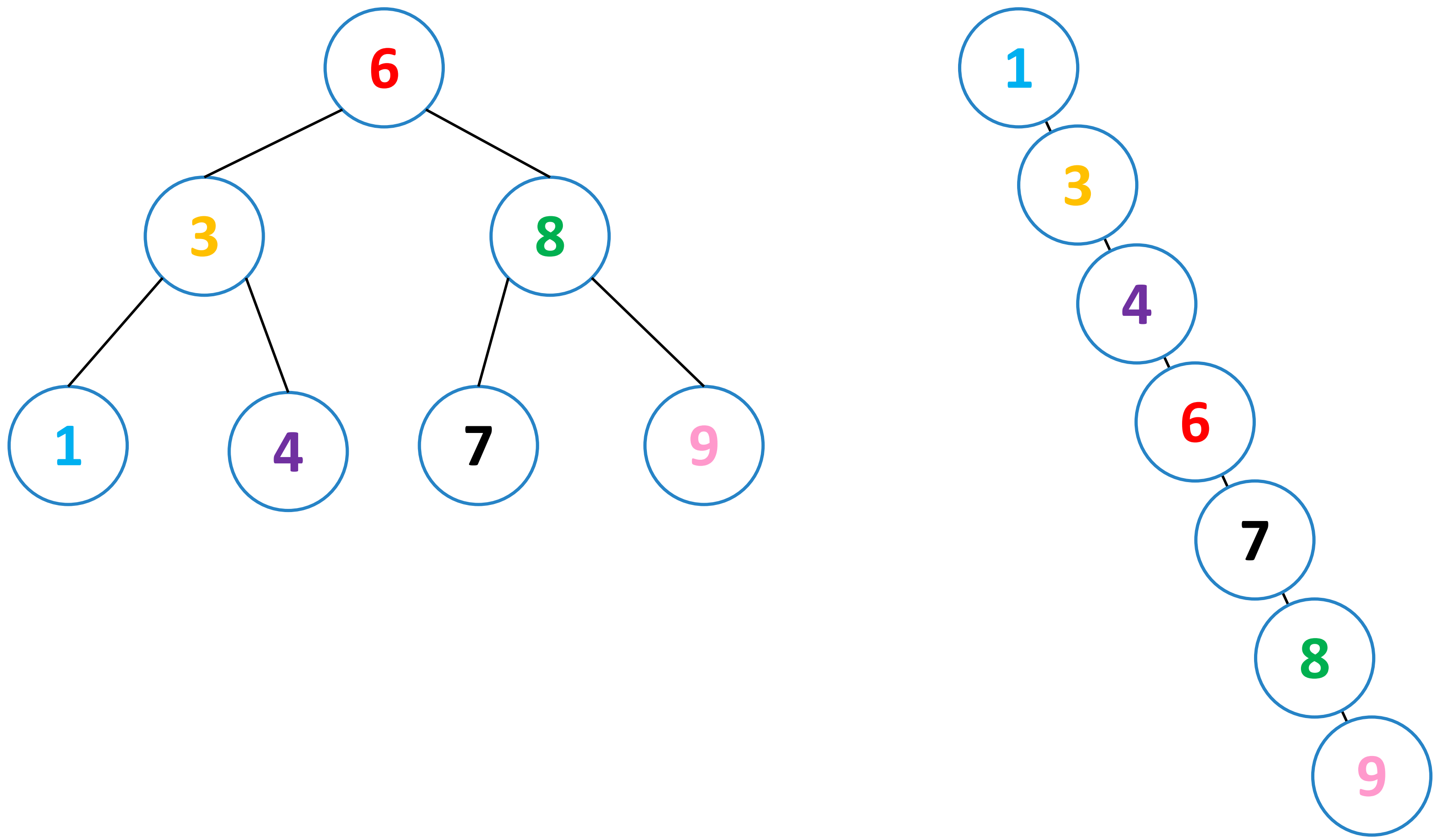
Hay raíces y raíces

¿Hay algunas raíces más convenientes que otras?

¿Qué pasa con el árbol si no queda una clave “conveniente” como raíz?

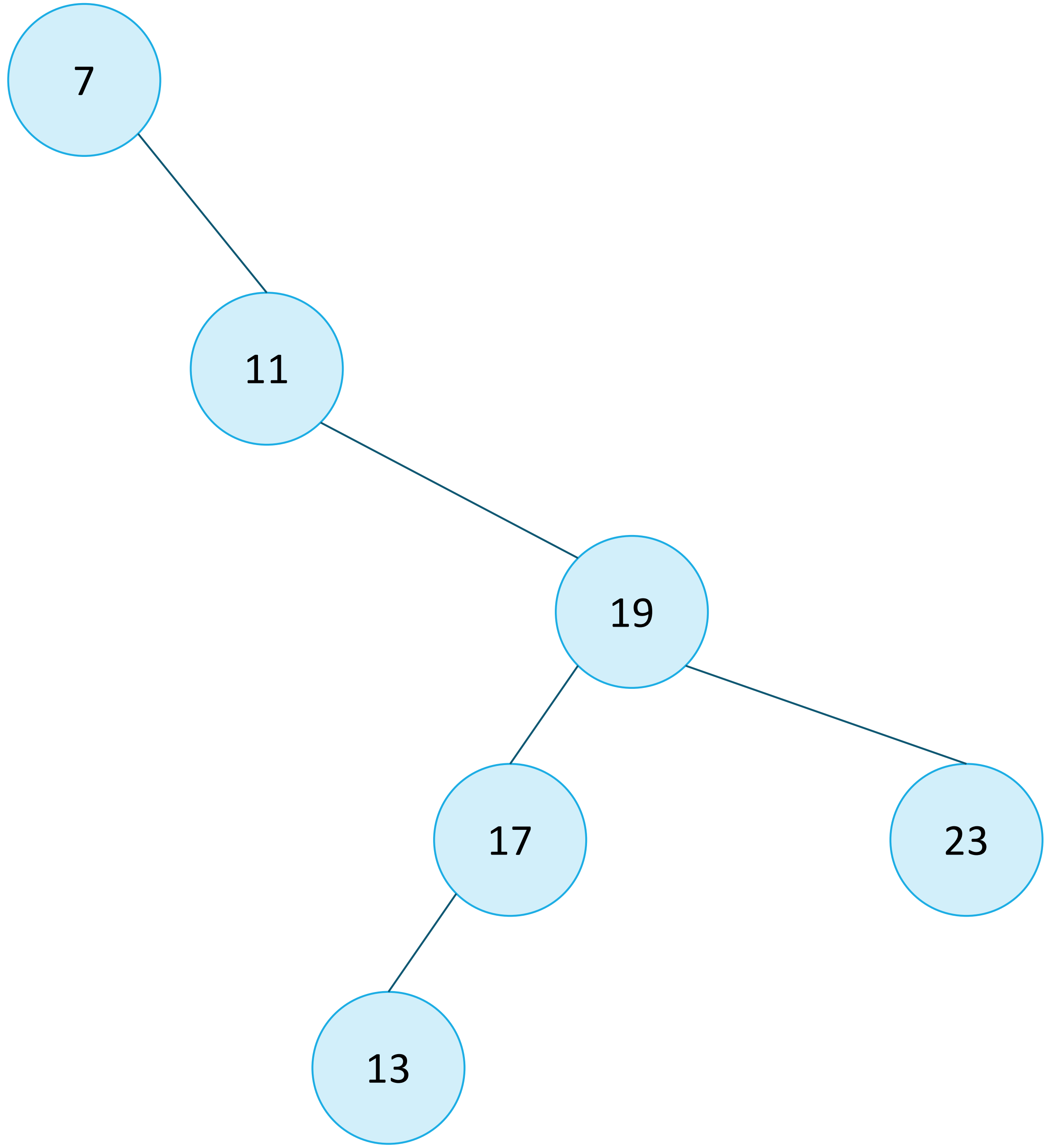
¿Cómo varía la complejidad de las operaciones?

Mismos datos, distintos árboles



¿Cuáles son las consecuencias de estas diferencias?

Si en el árbol de la derecha buscamos la clave 18:



Si en el árbol de la derecha buscamos la clave 18 a partir de la raíz:

- comparamos 18 con 7

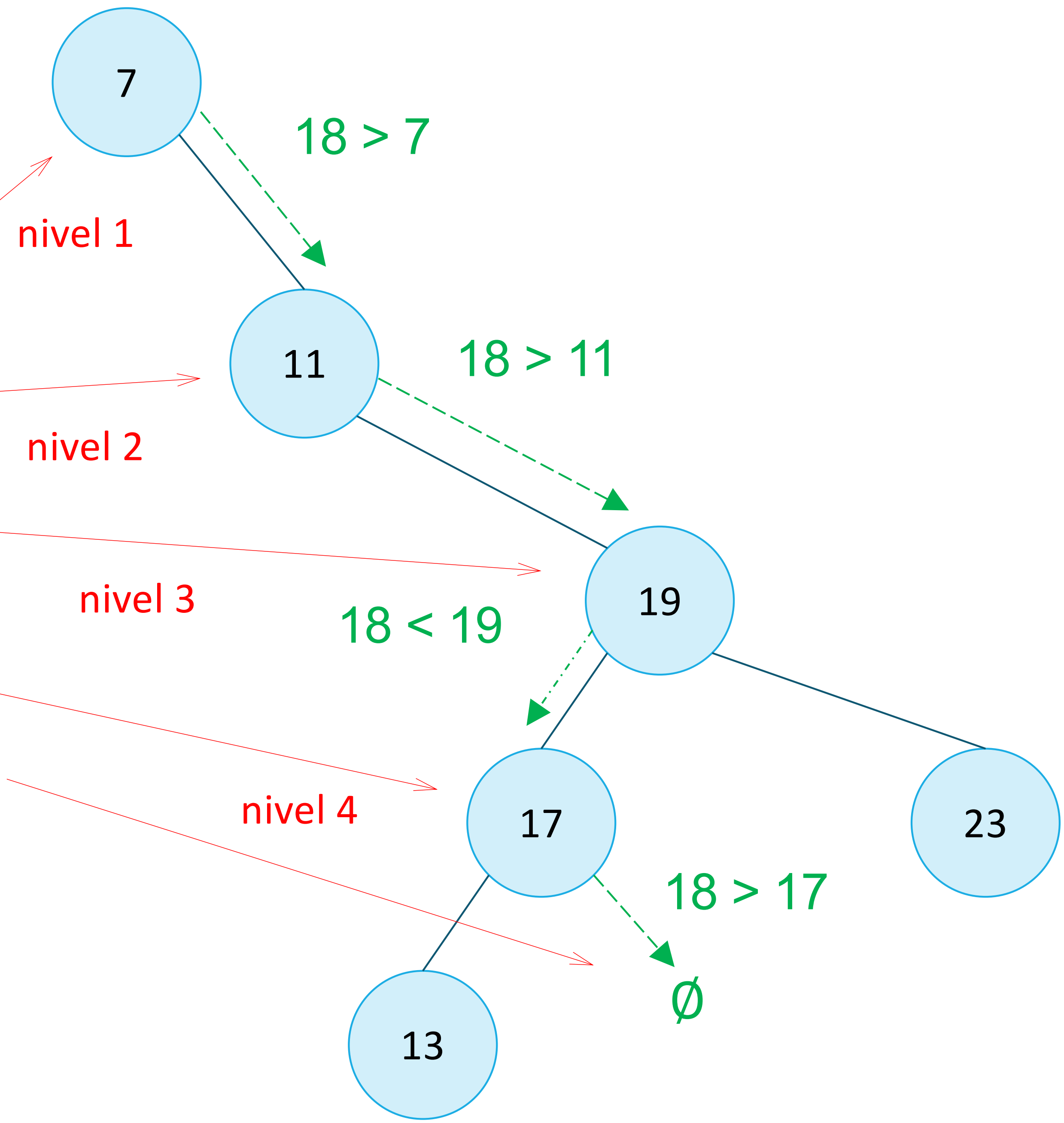
  ... luego con 11

  ... de ahí con 19

  ... con 17

  ... y tratamos de ir al hijo derecho del nodo con clave 17

- **(** este último nodo no existe → la clave 18 **no está** almacenada en el árbol, y por lo tanto devolvemos ∅ **)**



Complejidad de las operaciones

Todas las operaciones

buscar, *insertar* y *eliminar**

toman tiempo (o número de pasos) **proporcional a la *altura del árbol*** —el número máximo de niveles desde la raíz hasta la hoja “de más abajo”

… o la longitud de la rama más larga del árbol:

- la altura mínima de un ABB con n objetos es $O(\log n)$
- … aunque en general podría ser $O(n)$

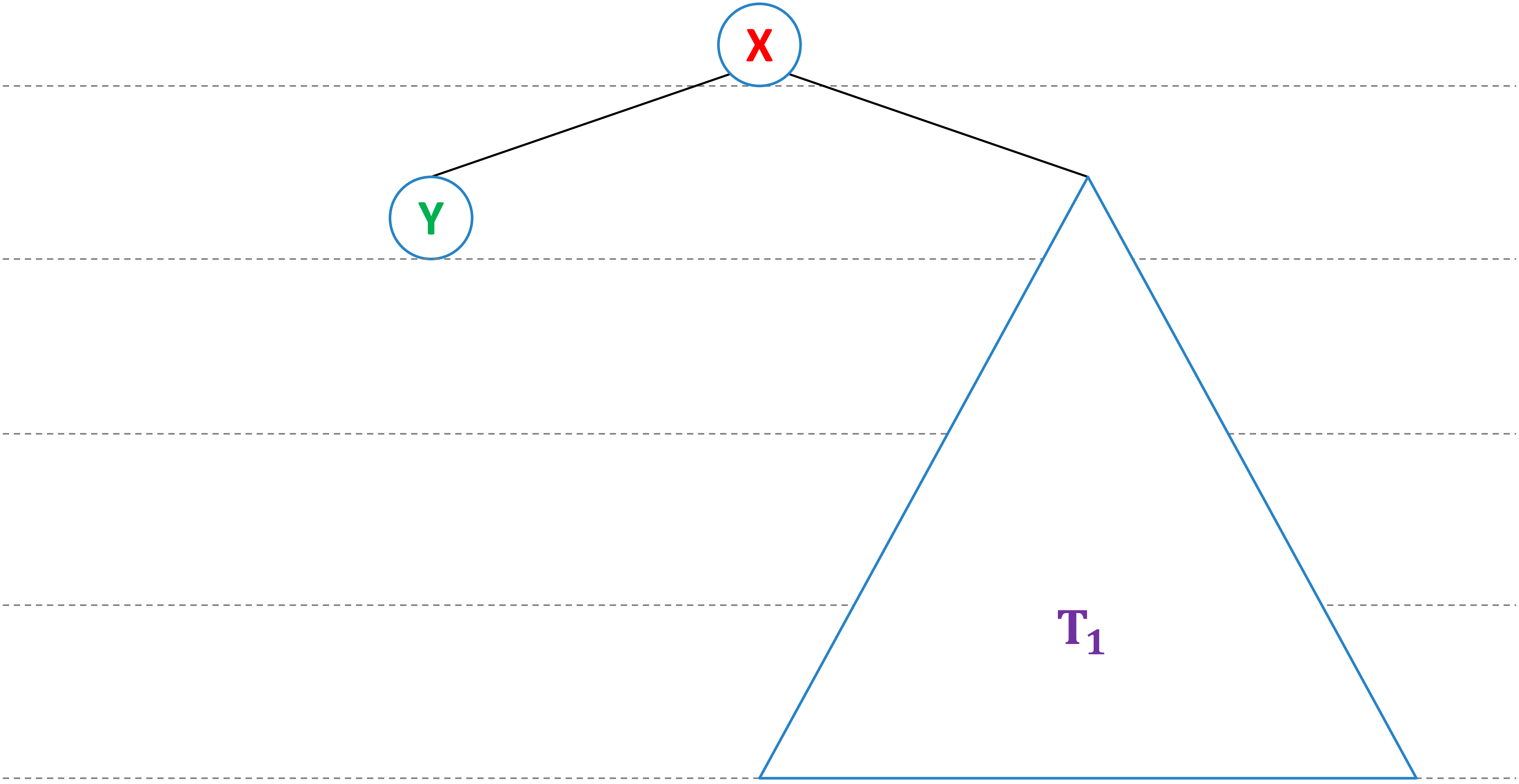
**insertar* y *eliminar* incluyen primero una búsqueda

Queremos asegurarnos de que el árbol esté **balanceado**

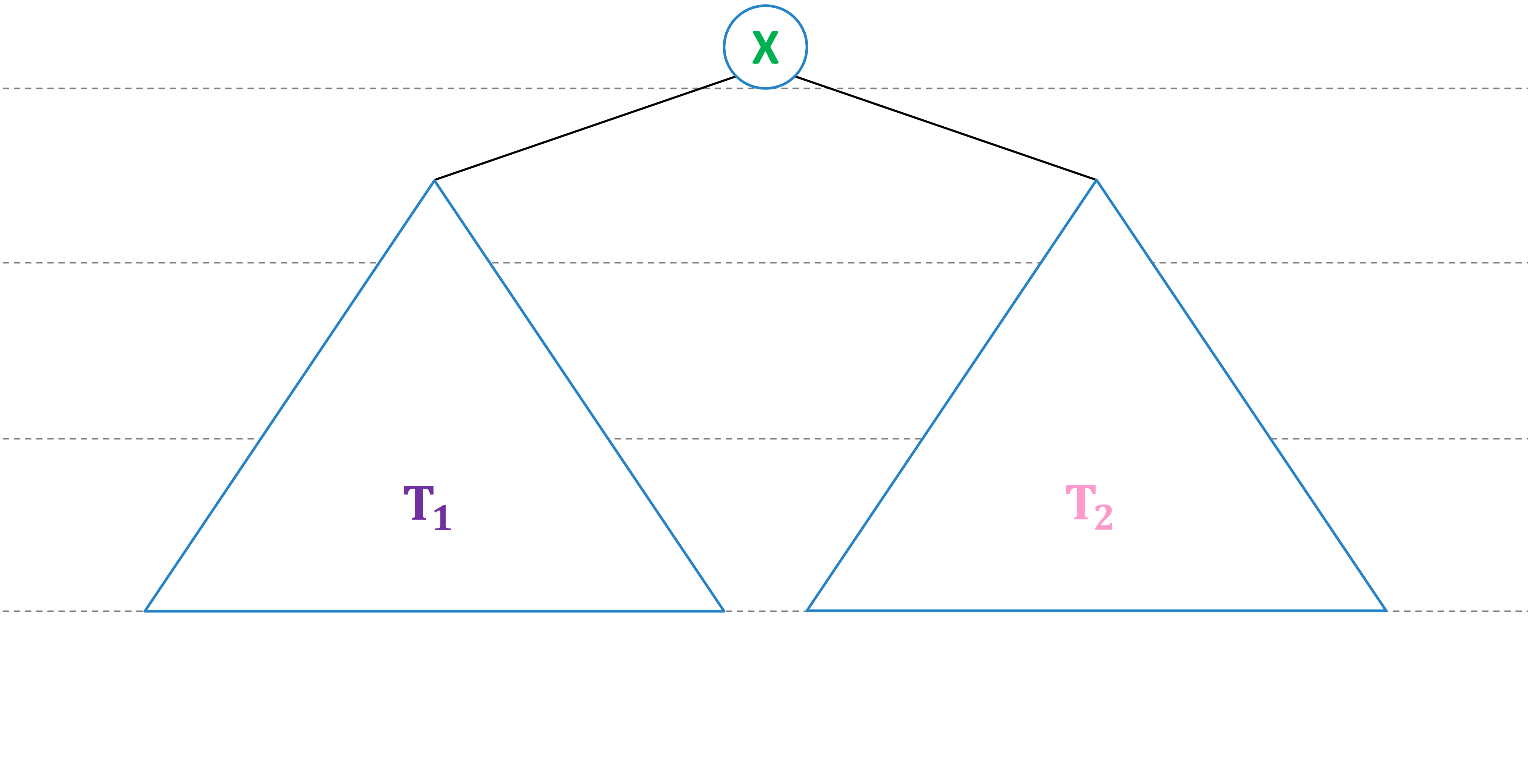
¿Cómo podemos definir esta noción?

Nos interesa que se cumpla recursivamente

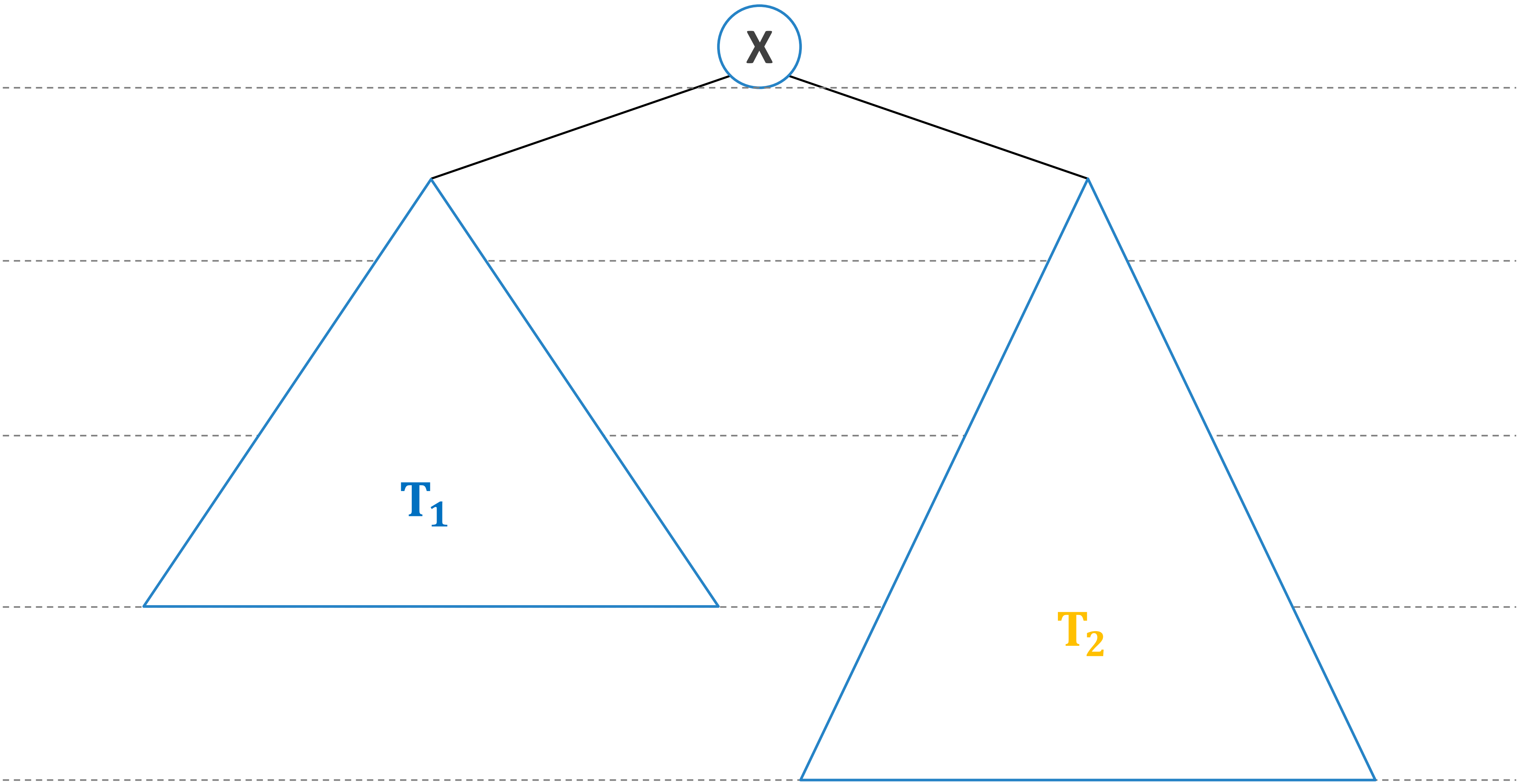
¿Está balanceado?



¿Está balanceado?



¿Está balanceado?



ABBs balanceados

Para ABBs, podemos garantizar que las operaciones de diccionario —*buscar*, *insertar* y *eliminar*— tomen tiempo $O(\log n)$ en el peor caso:

es necesario mantenerlos balanceados

La **propiedad de balance** debe cumplir dos condiciones:

- asegurar que la altura de un árbol con n nodos sea $O(\log n)$
- ser fácil de mantener —p.ej., la complejidad de (re)balancear el árbol después de una inserción no puede ser mayor que $O(\log n)$

Un ABB se llama **árbol AVL** si cumple la siguiente **propiedad de balance** AVL

Diremos que un ABB está **AVL-balanceado** si:

- las alturas de los hijos de la raíz difieren a lo más en 1 entre ellas
- cada hijo a su vez está **AVL-balanceado**

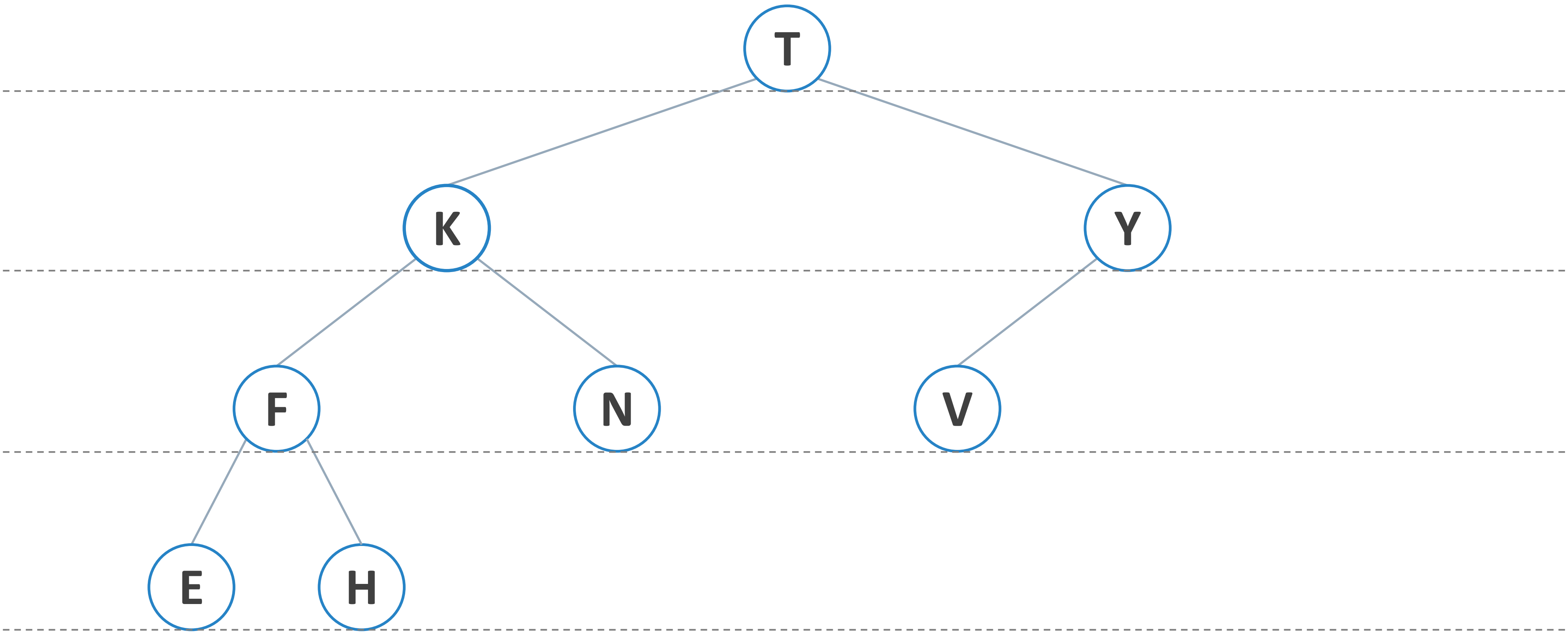
Al insertar o eliminar un nodo en un árbol AVL, es posible desbalancear el árbol

¿Cómo garantizamos el **balance** del árbol luego de cada operación?

Nos interesa conservar todas las propiedades de los ABB y además la propiedad de balance AVL:

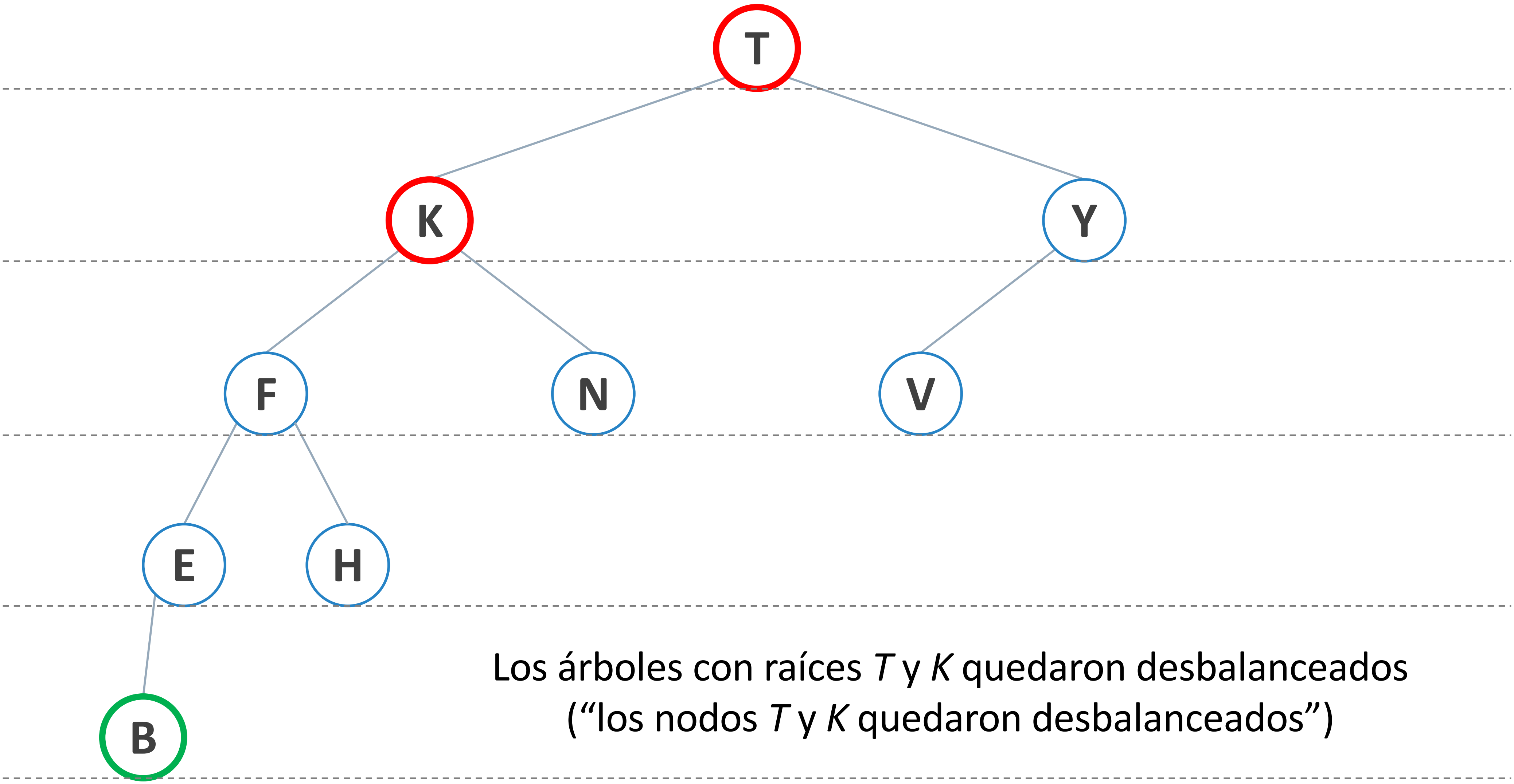
- en particular, el balance **debe ser restaurado** antes de que la operación —de inserción o eliminación— pueda considerarse completa

P.ej., árbol AVL inicial



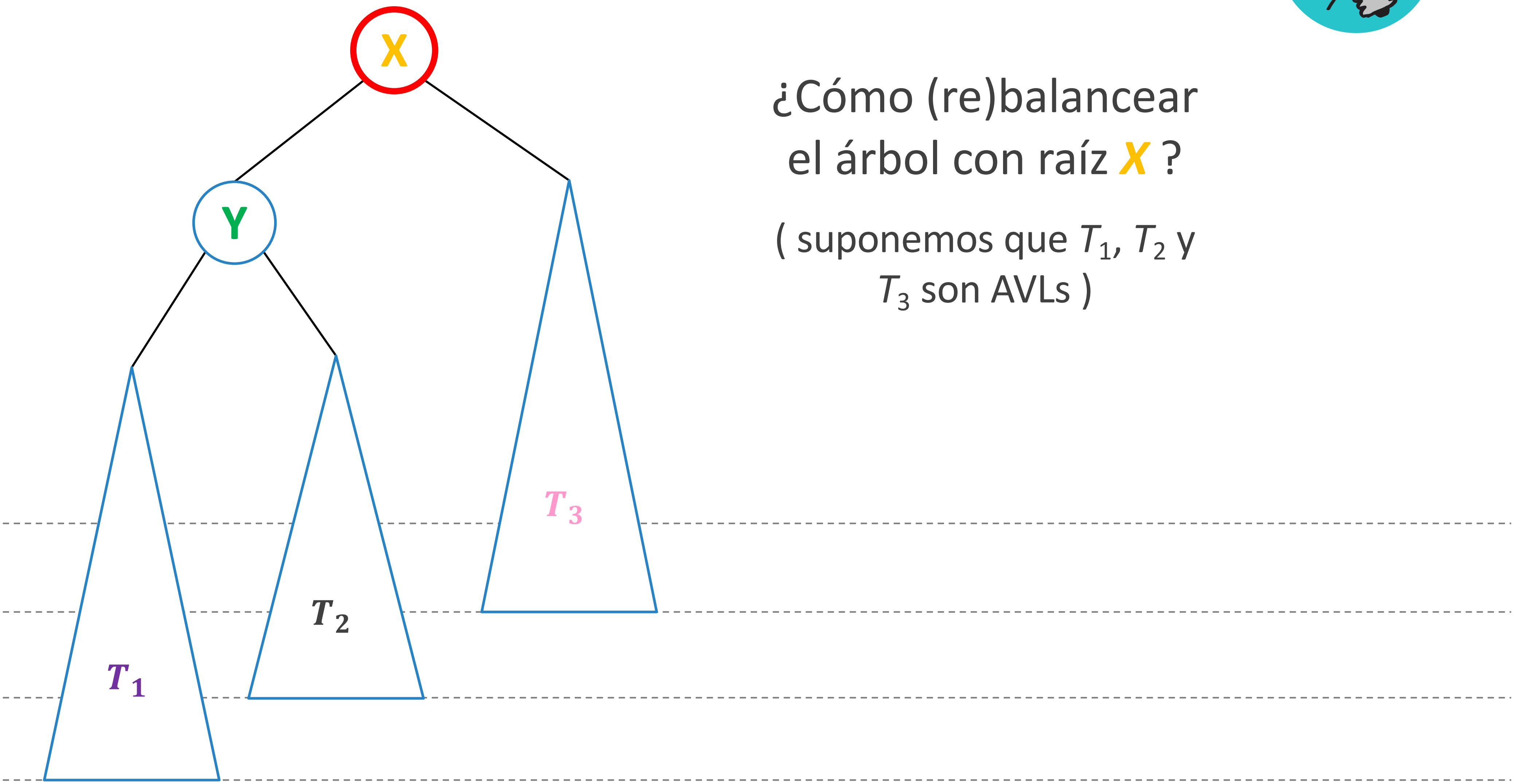
Para cualquier nodo, las alturas de sus hijos difieren a lo más en 1 entre ellas

… árbol luego de insertar *B*



Los árboles con raíces *T* y *K* quedaron desbalanceados
(“los nodos *T* y *K* quedaron desbalanceados”)

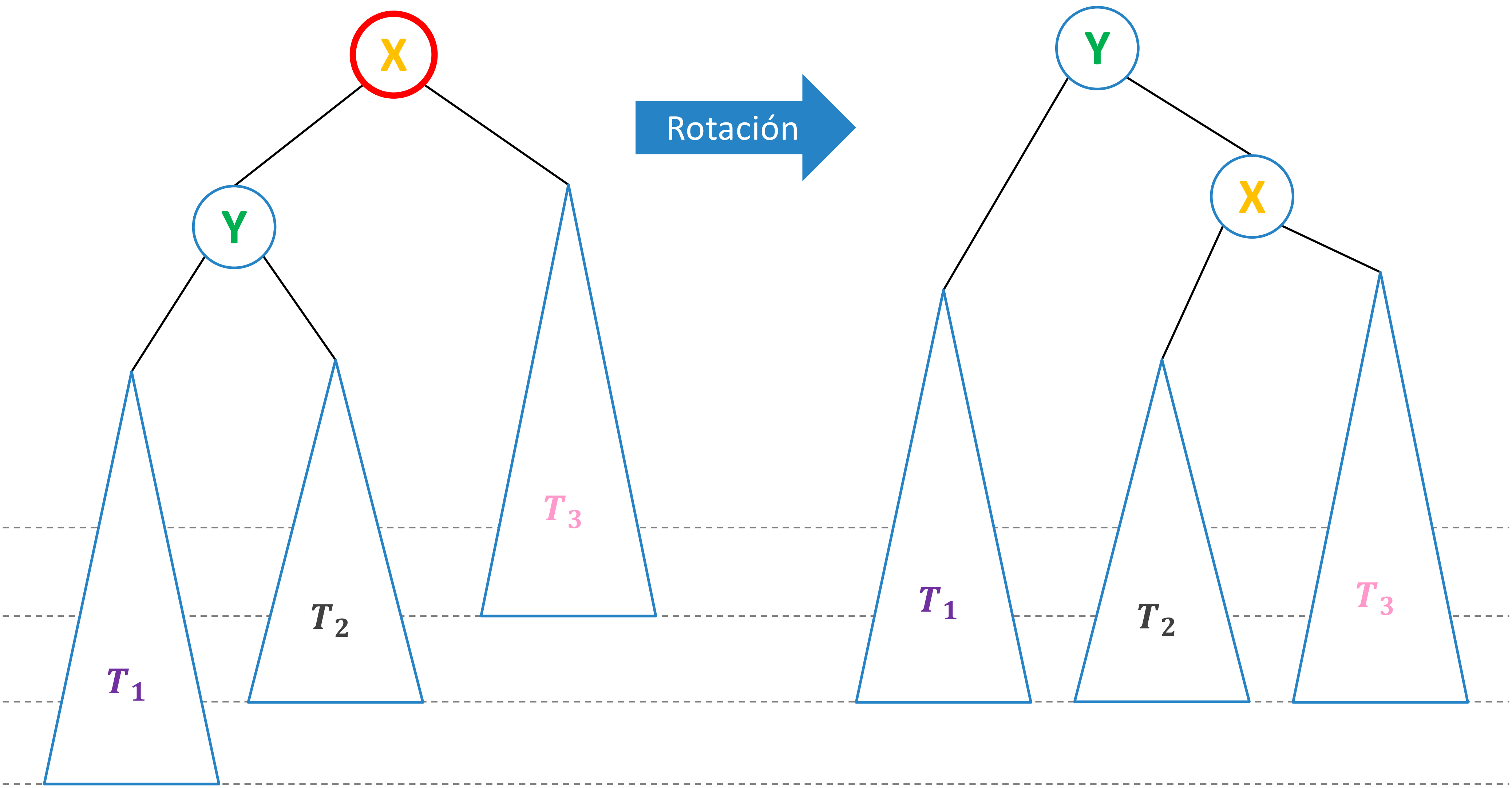
Más en general, luego de insertar en T_1



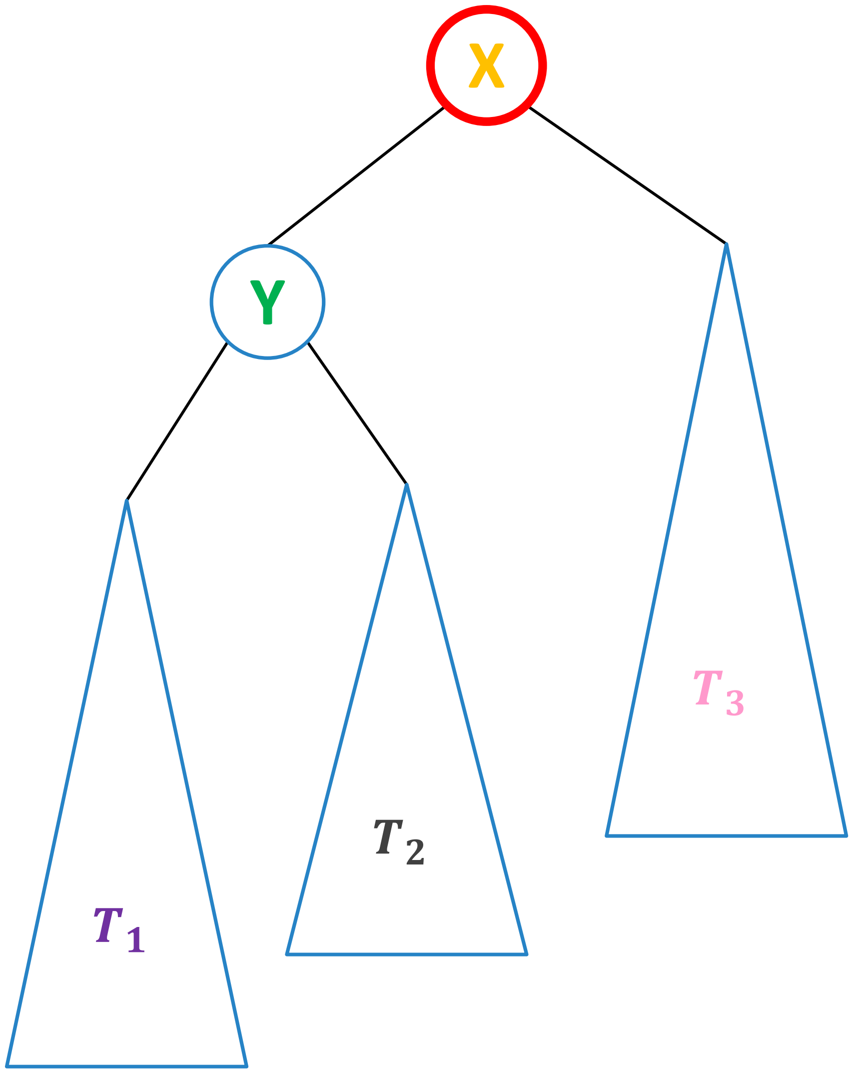
¿Cómo (re)balancear el árbol con raíz ***X*** ?

(suponemos que T_1, T_2 y T_3 son AVLs)

Rotación a la derecha en torno a *X*-*Y*



Rotación *X*-*Y*



¿Cómo encontramos los nodos *X* y *Y* para hacer la rotación?

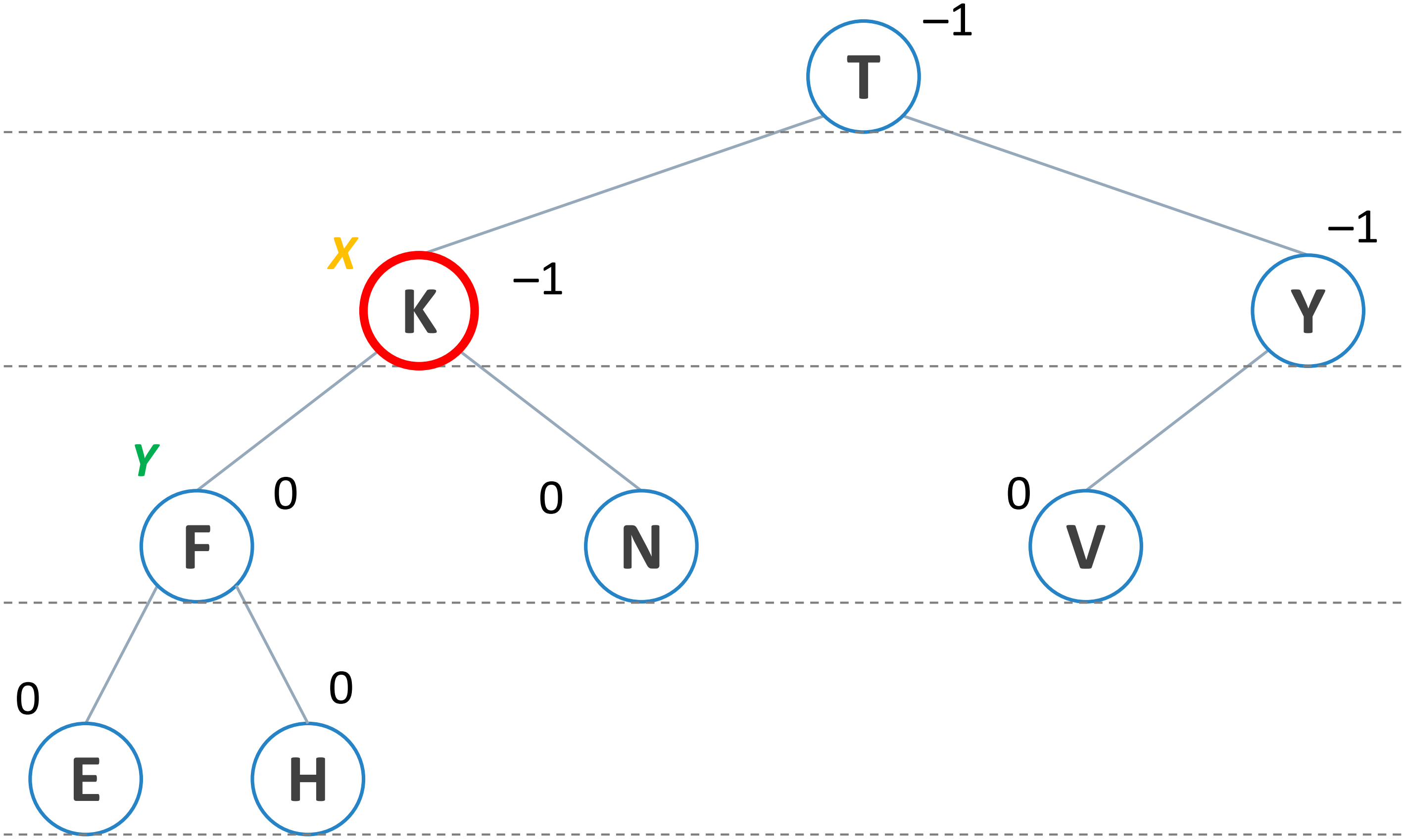
Agregamos a cada nodo *r* un **atributo de balance** (un campo adicional):

$$\boldsymbol{r.balance} = -1 \,/\, 0 \,/\, +1$$

… dependiendo de si:

- el subárbol izquierdo es más alto (–1)
- ambos subárboles tienen la misma altura (0)
- el subárbol derecho es más alto (+1)

Balances antes de insertar *B*

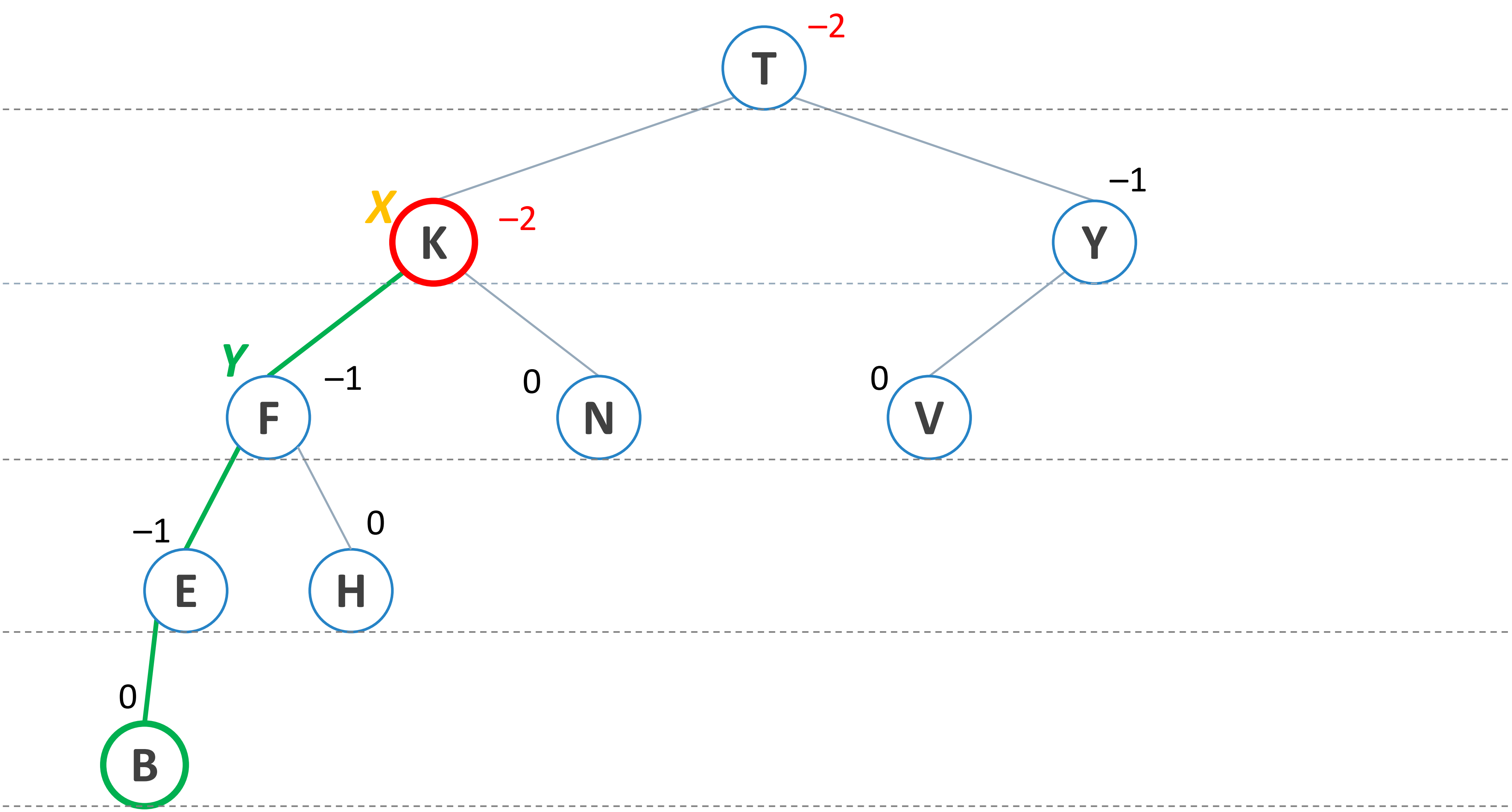


Luego de insertar, recorremos el árbol hacia arriba a lo largo de la **ruta de inserción**:

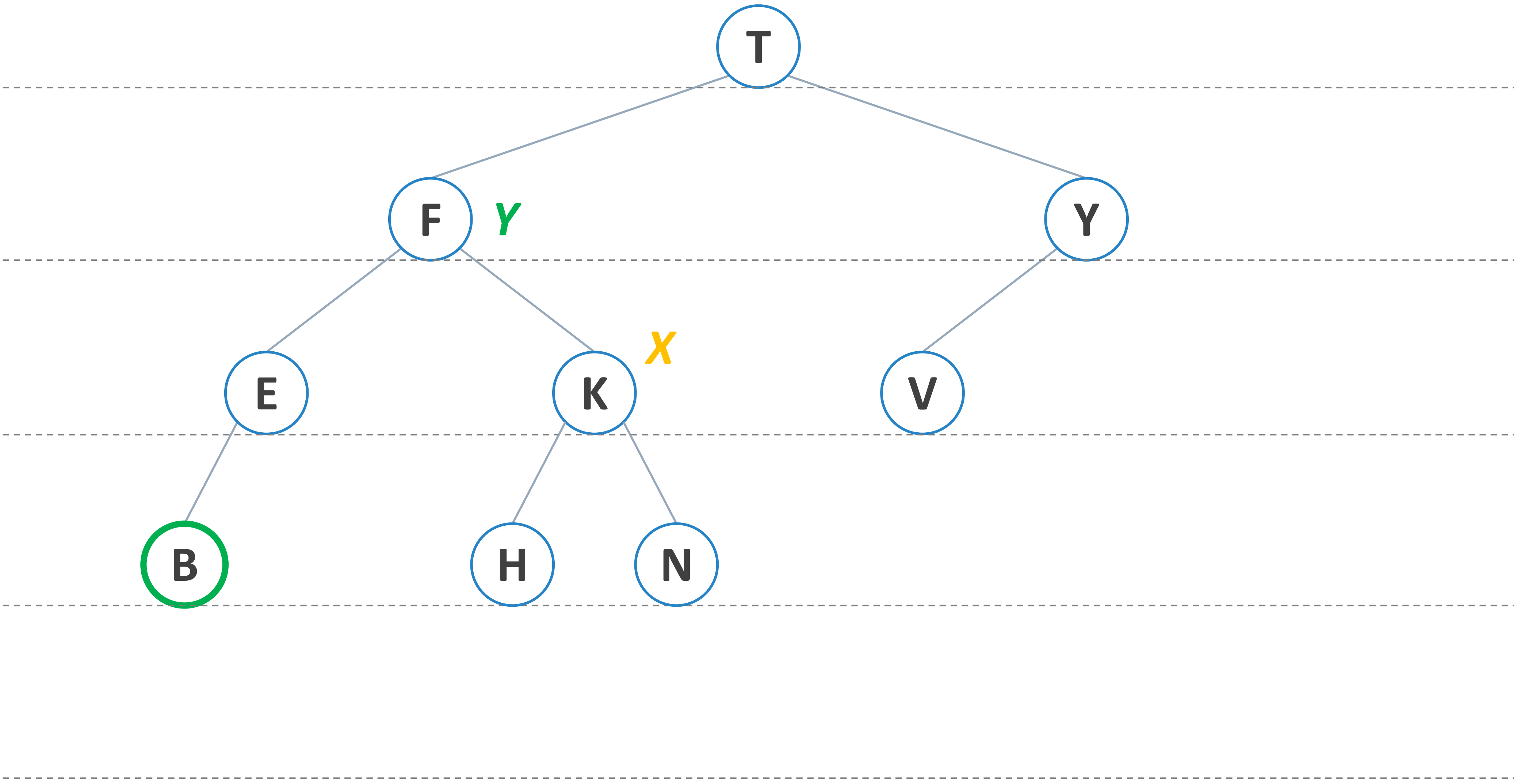
definimos ***X*** como la raíz del **primer** árbol desbalanceado que encontremos (o como el primer “nodo desbalanceado”),

… y ***Y*** como el hijo de ***X*** en la ruta de inserción

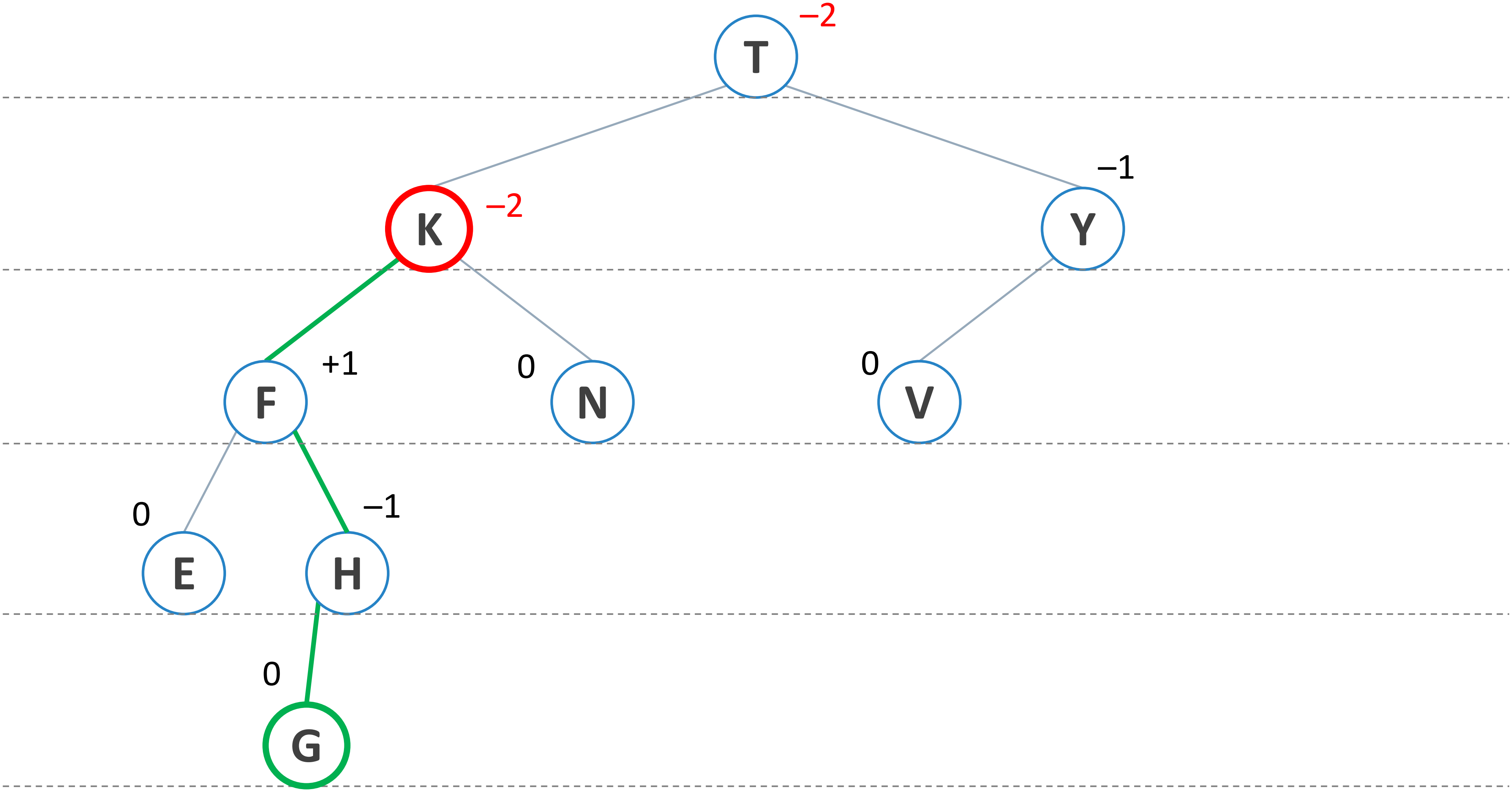
Luego de insertar *B*



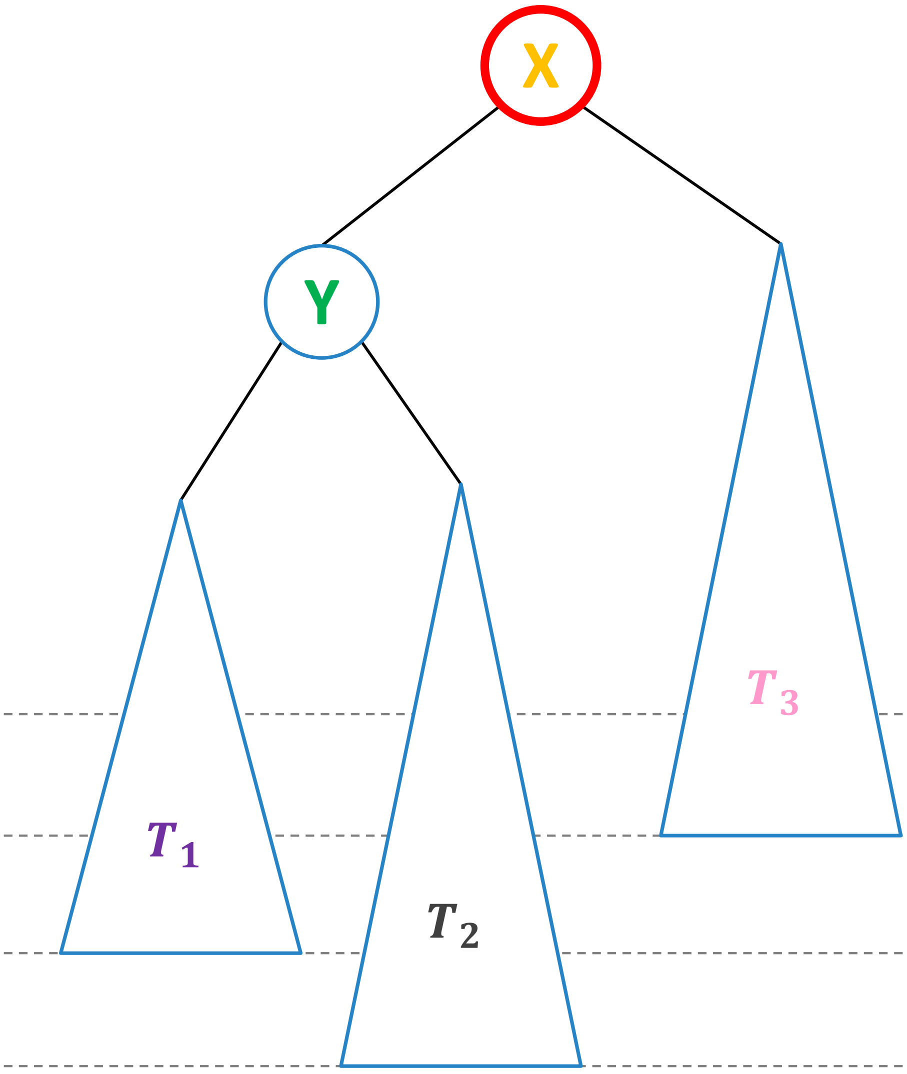
Rotación a la derecha en torno a *K-F*



Otro ej.: el árbol (inicial), luego de insertar *G*

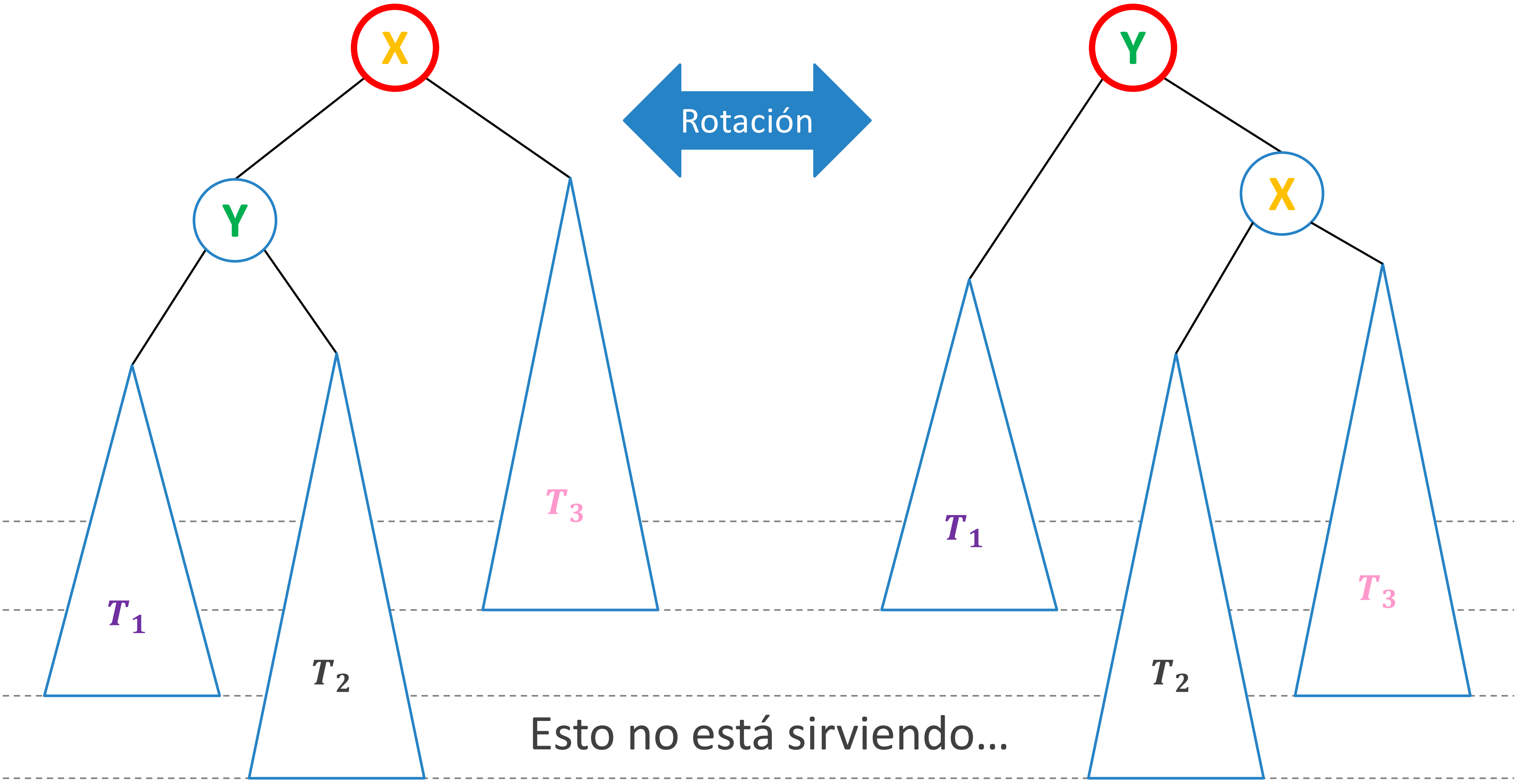


Más en general, el árbol luego de una inserción en T_2



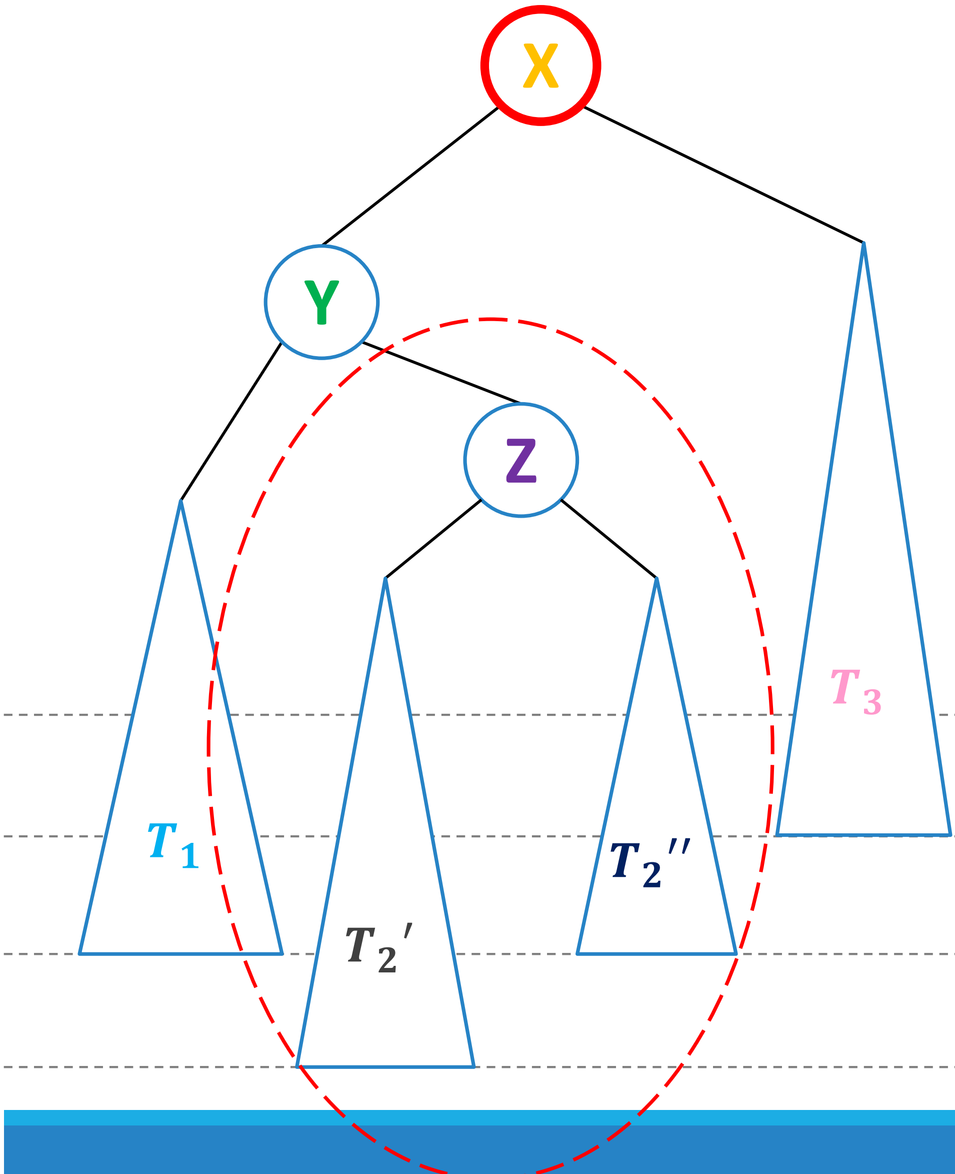
¿Cómo (re)balancear el árbol con raíz X?

¿Rotación a la derecha en torno a *X*-*Y* ?



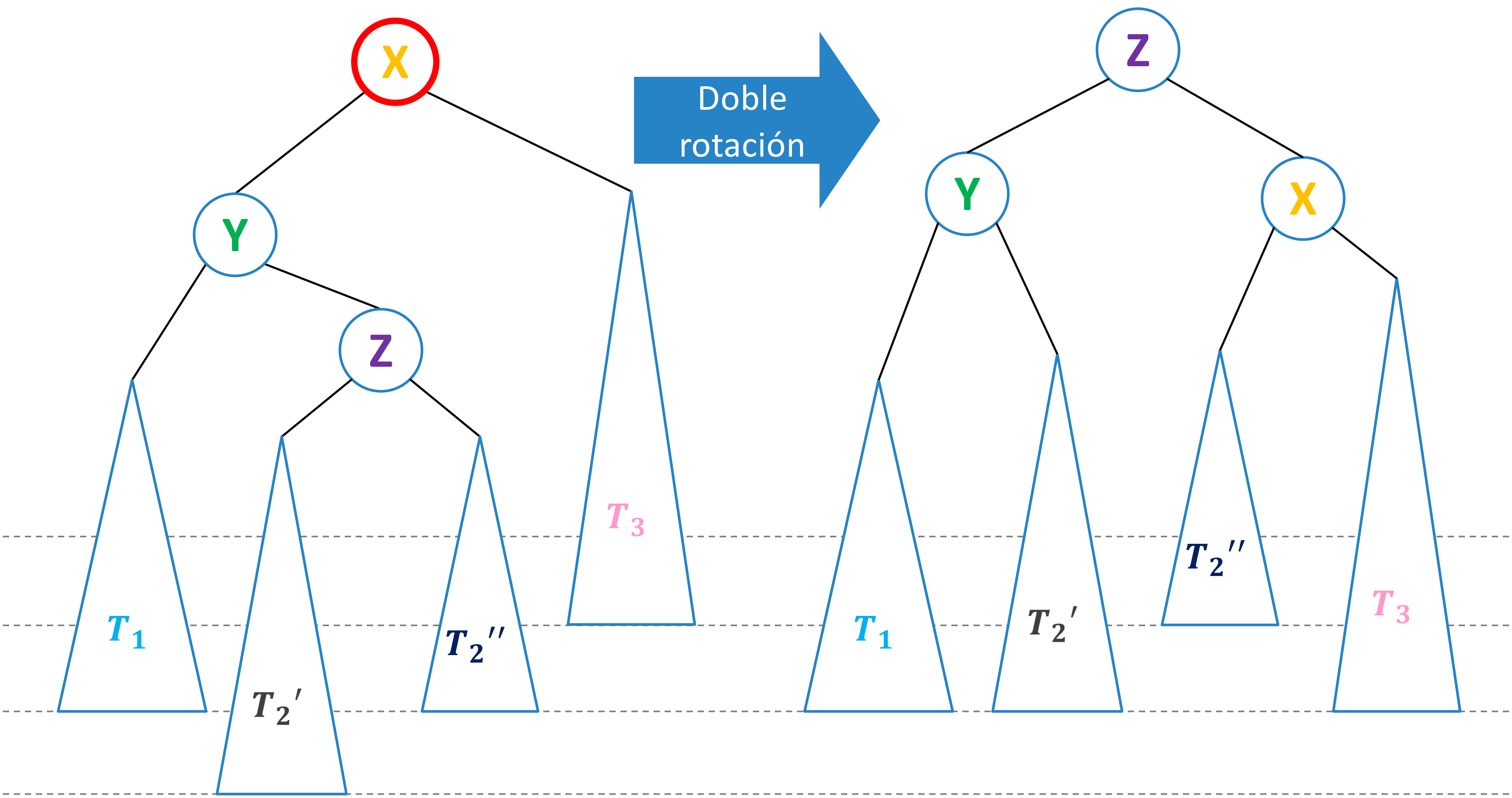
Esto no está sirviendo…

Hagamos doble click en T_2



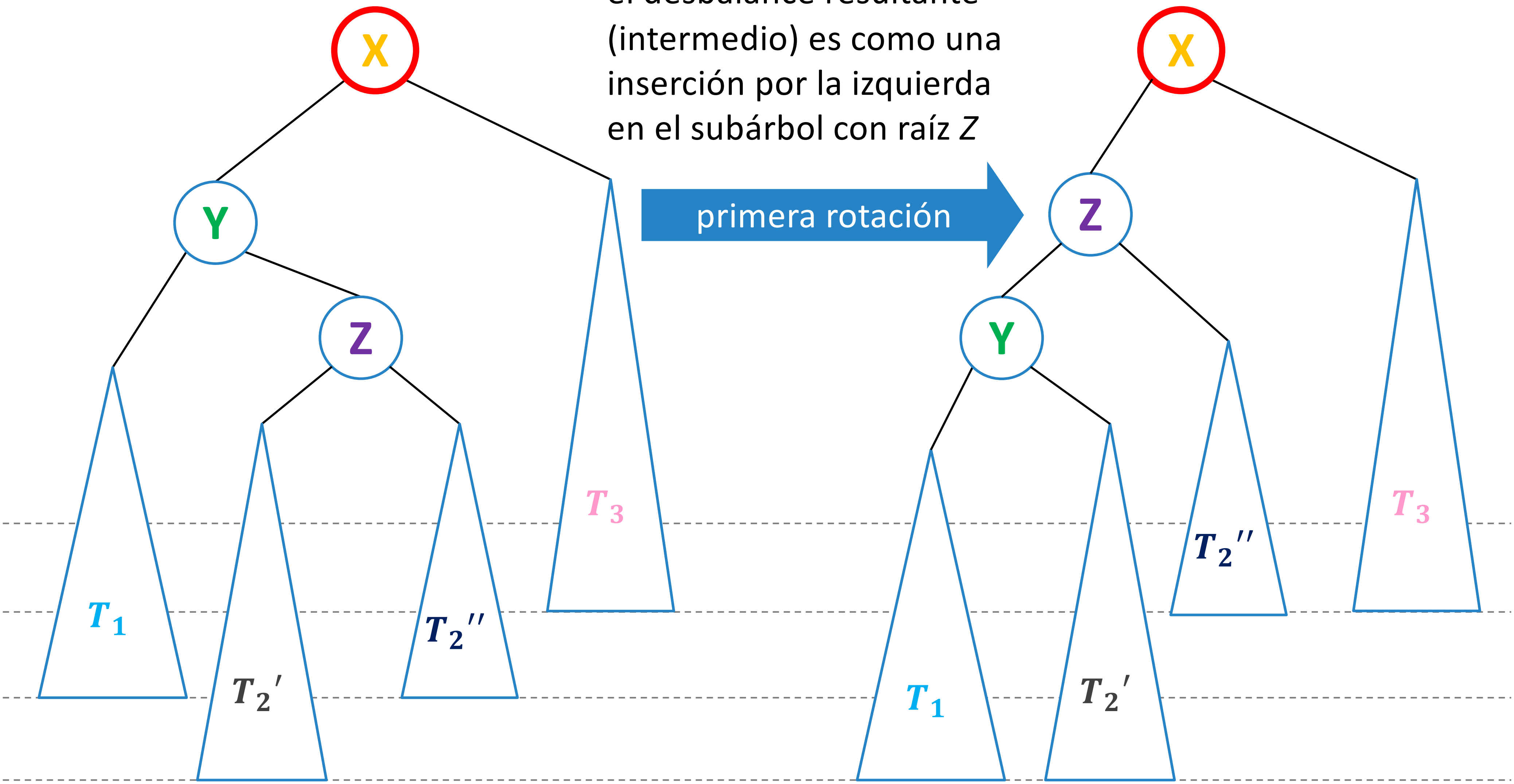
El nodo insertado podría estar en T_2' o en T_2''; en el caso del ej. de la diap. #22, está en T_2'

Rotación doble: primero a la izquierda en torno a *Y-Z* , luego a la derecha en torno a *X-Z*



(La primera rotación, a la izquierda, convierte el problema en uno similar al de la diap. #15)

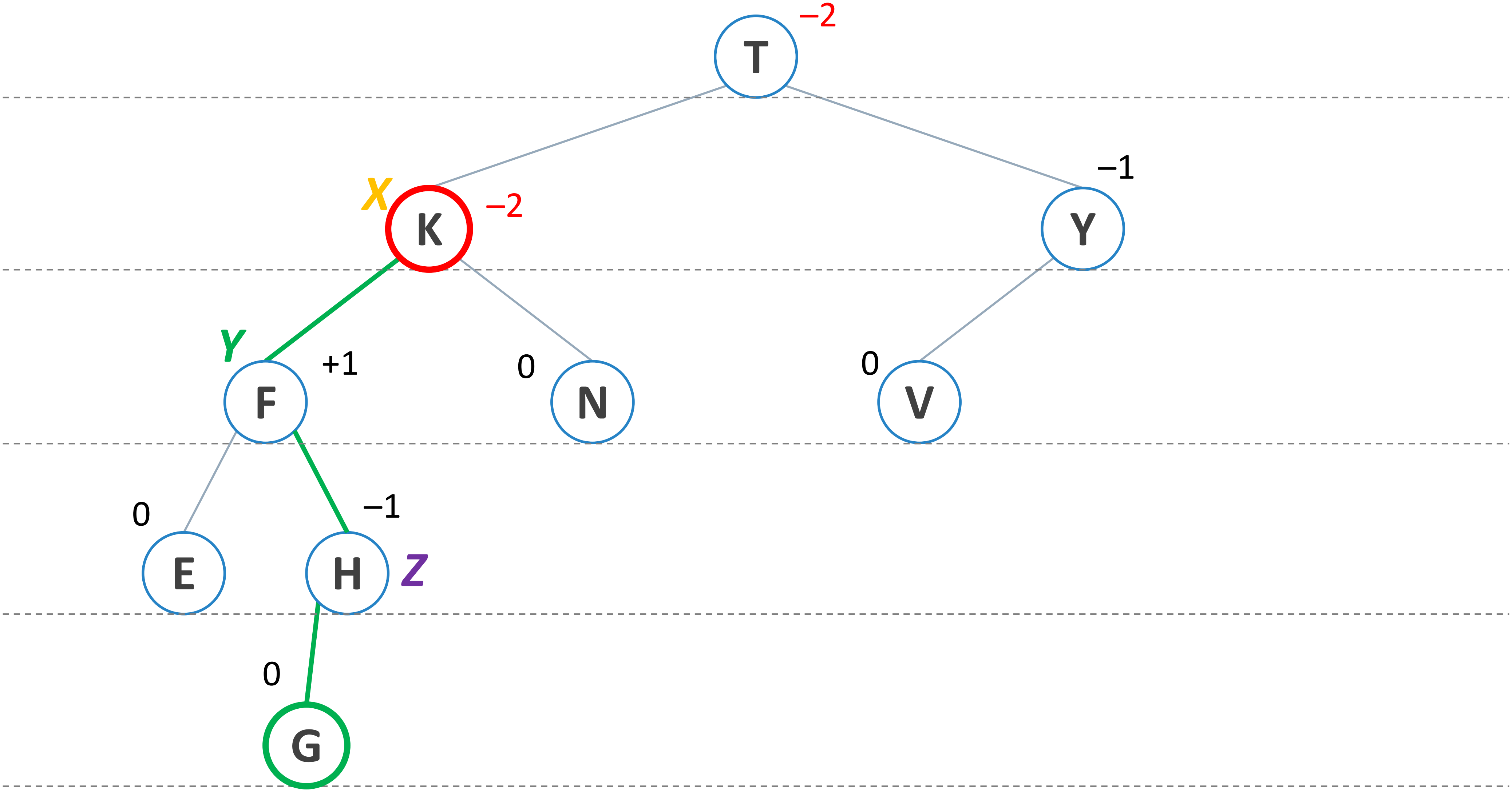
el desbalance resultante (intermedio) es como una inserción por la izquierda en el subárbol con raíz *Z*



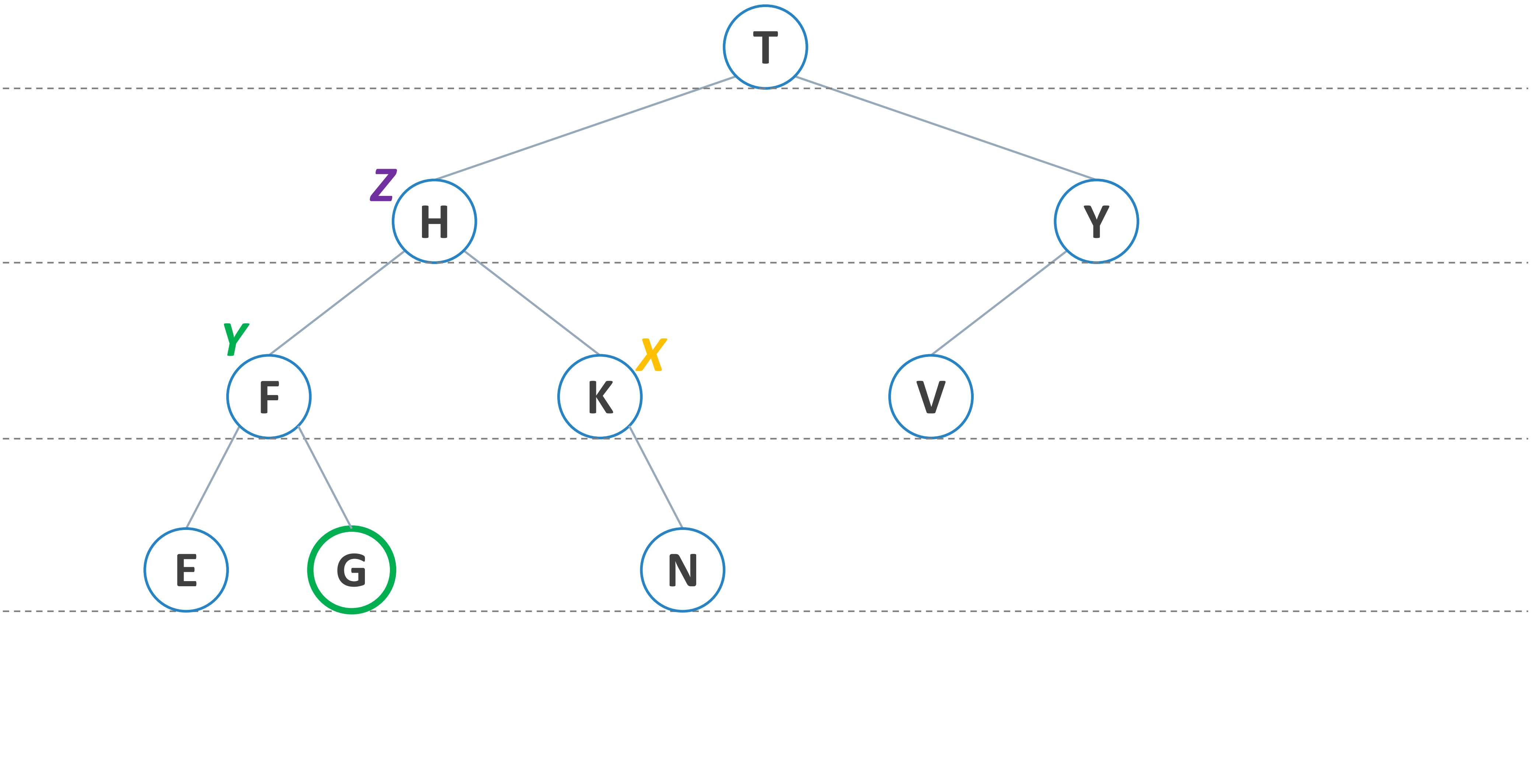
X y ***Y*** se definen igual que antes, y ***Z*** es el hijo de ***Y*** en la ruta de inserción

Tenemos definidos entonces ***X***, ***Y*** y ***Z*** en torno a los que hacemos las rotaciones

Vovliendo al ejemplo, luego de insertar *G*



¡ La rotación doble rebalancea el árbol !



Hay 4 casos de desbalance, según la ruta de inserción desde *X*

1. Izquierda + izquierda (LL): rotación simple

2. Izquierda + derecha (LR): rotación doble

3. Derecha + izquierda (RL): rotación doble

4. Derecha + derecha (RR): rotación simple

Los casos 1 y 4 son simétricos entre ellos; lo mismo para 2 y 3

Analicemos la restauración del balance

¿Qué tan costoso es rebalancear el árbol?

¿Cuántas rotaciones es necesario hacer en el peor caso?

Una rotación tiene **costo constante**:
no depende del número de nodos del árbol

Rotación simple: hay que cambiar 3 punteros

Rotación doble: hay que cambiar 5 punteros

Además, al hacer estas rotaciones para el ***X*** que definimos, se soluciona el desbalance de ***X*** **sin crear un nuevo desbalance**

… por lo que siempre **en el peor caso realizamos una sola rotación** (simple o doble) por cada inserción

El costo de una inserción sigue siendo $O(h)$

Luego de insertar, debemos revisar hacia arriba la ruta de inserción buscando el primer desbalance

El peor caso es que no haya un desbalance y lleguemos hasta la raíz

Toda **la inserción sigue siendo $\mathbf{O}(\boldsymbol{h})$**, siendo h la altura del árbol (aunque el número de pasos individuales más o menos se duplicó)

Entonces, la pregunta relevante ahora es

¿cuál es la altura de un árbol AVL en función del número de nodos?

Altura *h* de un árbol AVL con *n* nodos

Podemos pensarlo al revés:

- ¿cuál es el máximo número de nodos de un árbol de altura h?
- ¿y el mínimo?

Vamos a demostrar que en ambos casos el número de nodos n crece exponencialmente con la altura h

… lo que implica que la altura está acotada por el logaritmo del número de nodos → el árbol está balanceado

1) Máximo número de nodos

El máximo número de nodos en un árbol binario de altura h se da cuando el árbol está lleno, es decir, $n = 2^h - 1$

$n \in \Theta(2^h)$, por lo que $2^h \in \Theta(n)$

Como ambas funciones son crecientes, $h \in \Theta(\log n)$

2) Mínimo número de nodos

Sea $m(h)$ la cantidad mínima de nodos que puede tener un árbol AVL de altura h

Observamos que $m(1) = 1$ (sólo la raíz) y $m(2) = 2$ (la raíz y un hijo)

Este árbol debe tener subárboles de alturas $h - 1$ y $h - 2$:

- al menos un subárbol tiene altura $h - 1$ (para que el árbol original tenga altura h)
- las alturas de los subárboles pueden diferir a lo más en 1 (la propiedad de balance)

… y estos subárboles deben tener el menor número de nodos para sus alturas

$$m(h) = m(h - 1) + m(h - 2) + 1$$

Esta recurrencia es similar a la recurrencia para la secuencia de Fibonacci:

$$F(h) = F(h-1) + F(h-2), h \geq 2$$

Sabemos (de *Matemáticas Discretas*) que $F(h) \cong \frac{\varphi^h}{\sqrt{5}}$

… y se puede comprobar fácilmente (p.ej., por inducción) que

$$m(h) = F(h+3) - 1$$

… por lo que

$$m(h) = F(h+3) - 1 \cong \frac{\varphi^{h+3}}{\sqrt{5}} - 1$$

… y finalmente

$$h + 3 \cong \log_{\varphi}(\sqrt{5}(m(h) + 1))$$

Así, de 1) y 2) …

La altura h de un árbol AVL de n nodos es $O(\log n)$, tanto en el caso del mayor número de nodos, como en el caso del menor número de nodos

Por lo tanto, en un árbol AVL de altura h

$$h \in O(\log n)$$

Árbol binario:

- cada nodo *x* tiene a lo más dos hijos, uno izquierdo y otro derecho,

  … que, si están, son raíces de los subárboles izquierdo y derecho de *x*

Árbol binario de búsqueda (ABB):

- la clave en un nodo *x* es mayor que cualquiera de las claves en el subárbol izquierdo de *x*

  … y menor que cualquiera de las claves en el subárbol derecho de *x*

ABB balanceado (ABBB):

- cumple una propiedad adicional de balance
- p.ej., en un árbol AVL, para cualquier nodo del árbol, las alturas de los subárboles izquierdo y derecho pueden diferir a lo más en 1