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

Árbol 2-3:

- Nodo 2, con una clave y, si no es una hoja, exactamente 2 hijos
- Nodo 3, con dos claves distintas y ordenadas y, si no es una hoja, exactamente 3 hijos
- Todas las hojas estén a la misma profundidad
- La inserción siempre se hace —inicialmente— en una hoja
- Si un nodo está lleno (ya tiene dos claves) y debe recibir una tercera clave, entonces se hace subir la clave que habría quedado al medio —la clave mediana— al nodo padre -> SPLIT
- El árbol sólo aumenta de altura cuando la raíz está llena y recibe una clave desde un hijo

Los árboles 2-3 son balanceados ... pero

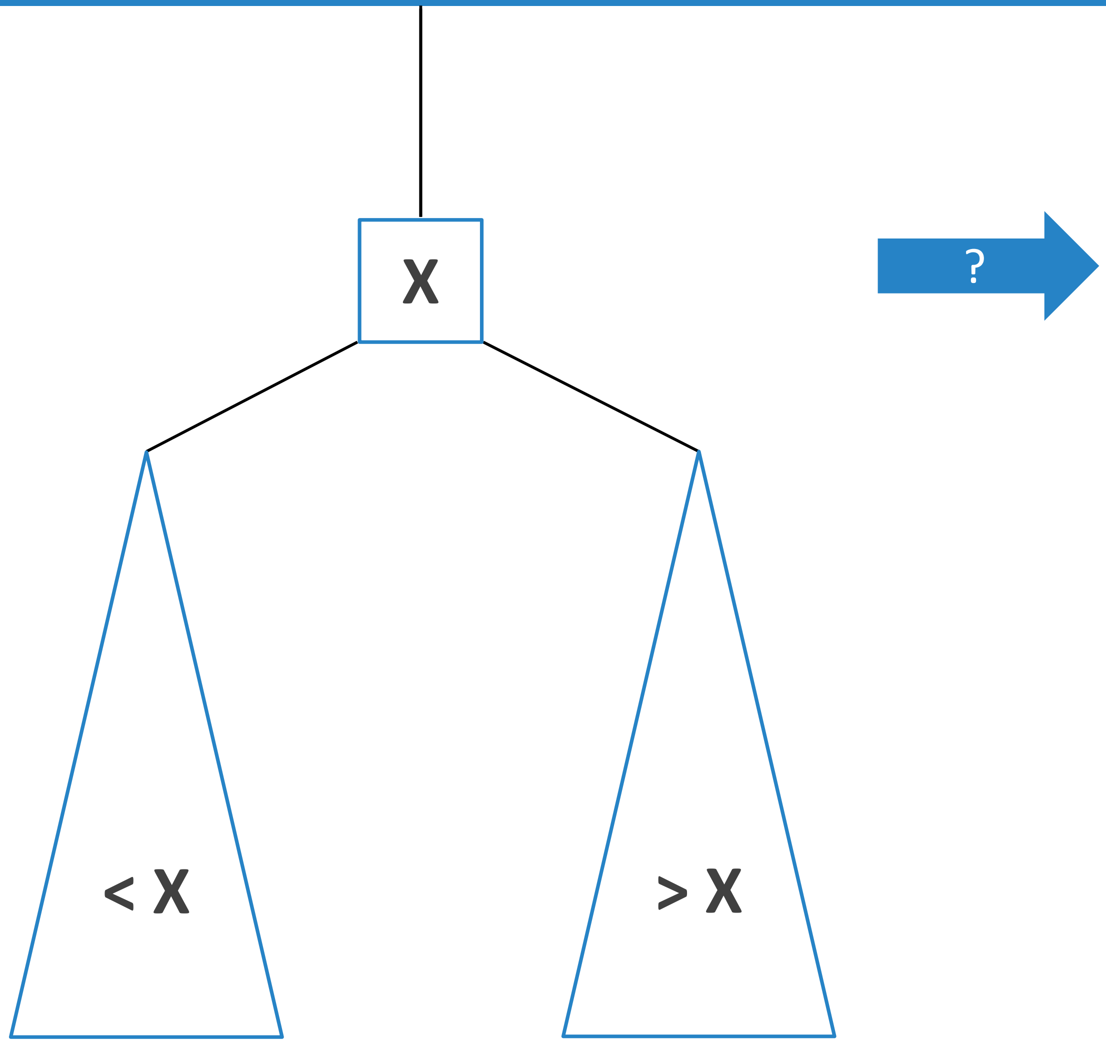
Las operaciones en un árbol 2-3, particularmente al insertar una nueva clave, tienen mucho *overhead*:

- durante el recorrido desde la raíz a la hoja, es posible que haya que hacer dos comparaciones en cada nodo (nodos 3)
- cuando se llega a la hoja, si es un nodo 2, hay que convertirlo en un nodo 3
- si es un nodo 3, hay que convertirlo en dos nodos 2 y hacer subir la clave mediana al nodo padre
- si el nodo padre es un nodo 2, hay que convertirlo en un nodo 3; si es un nodo 3, hay que aplicar recursivamente el paso anterior

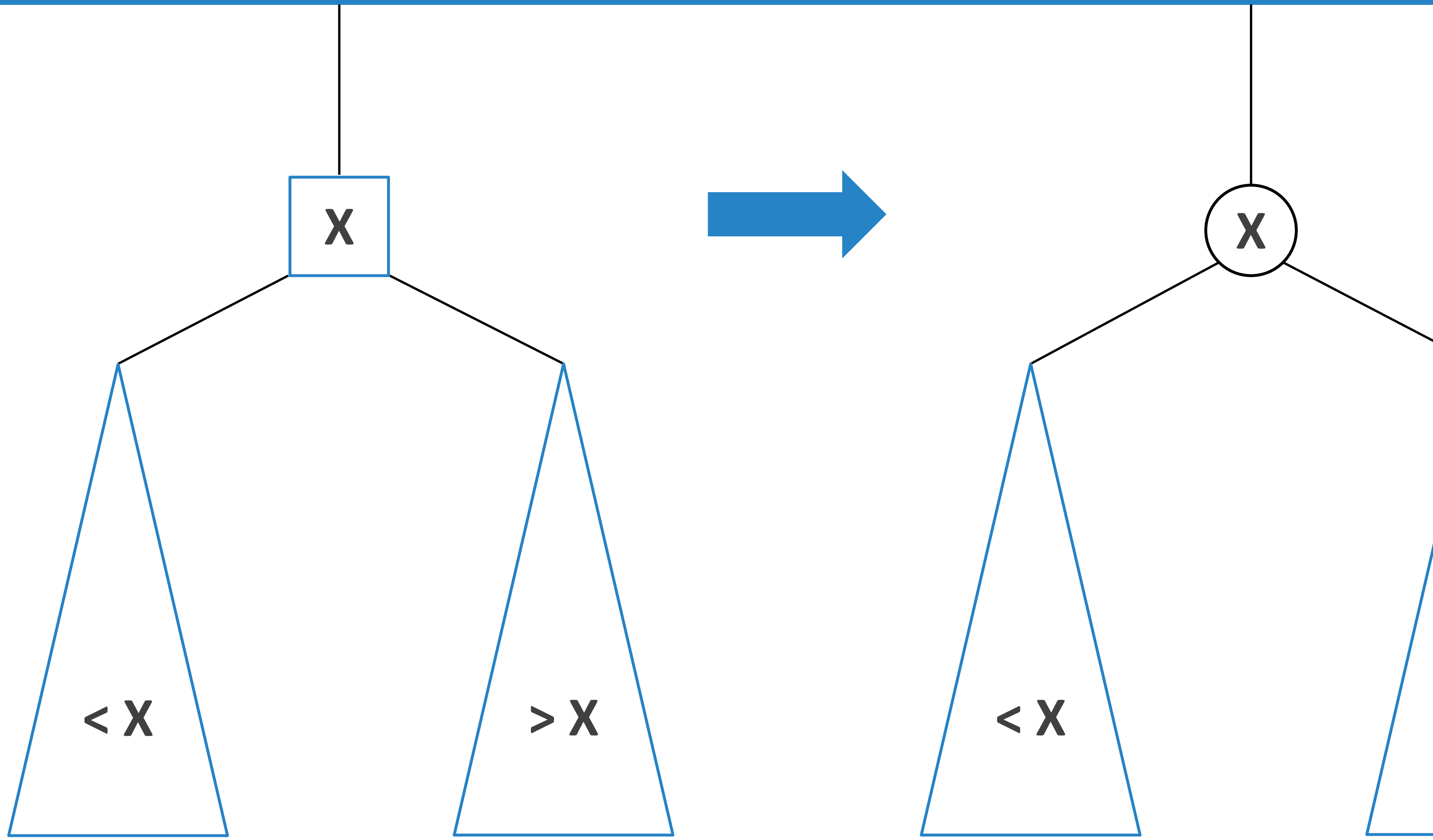
¿Será posible representar un árbol 2-3 como un ABB?

Nos interesa conservar toda la información del 2-3

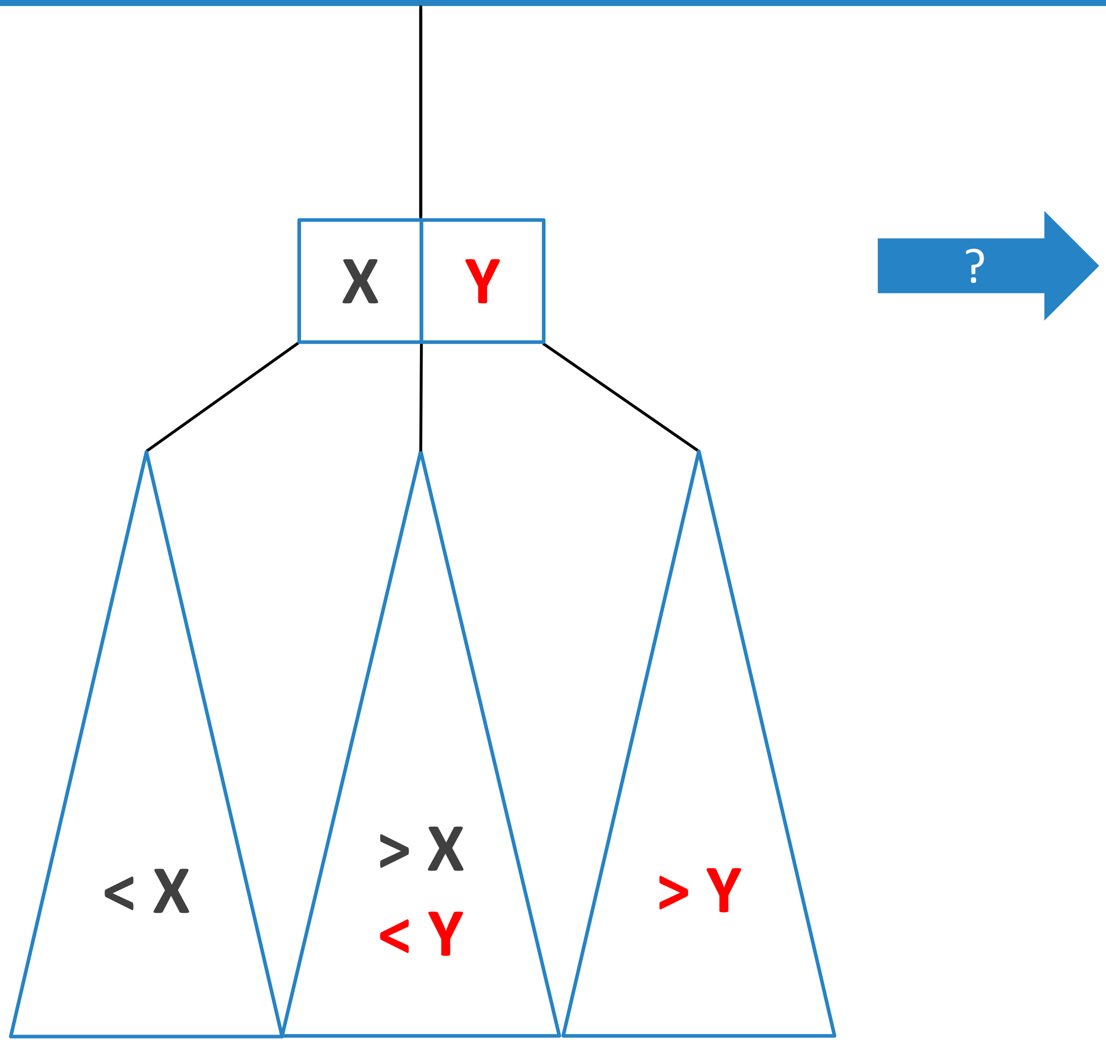
Nodo 2



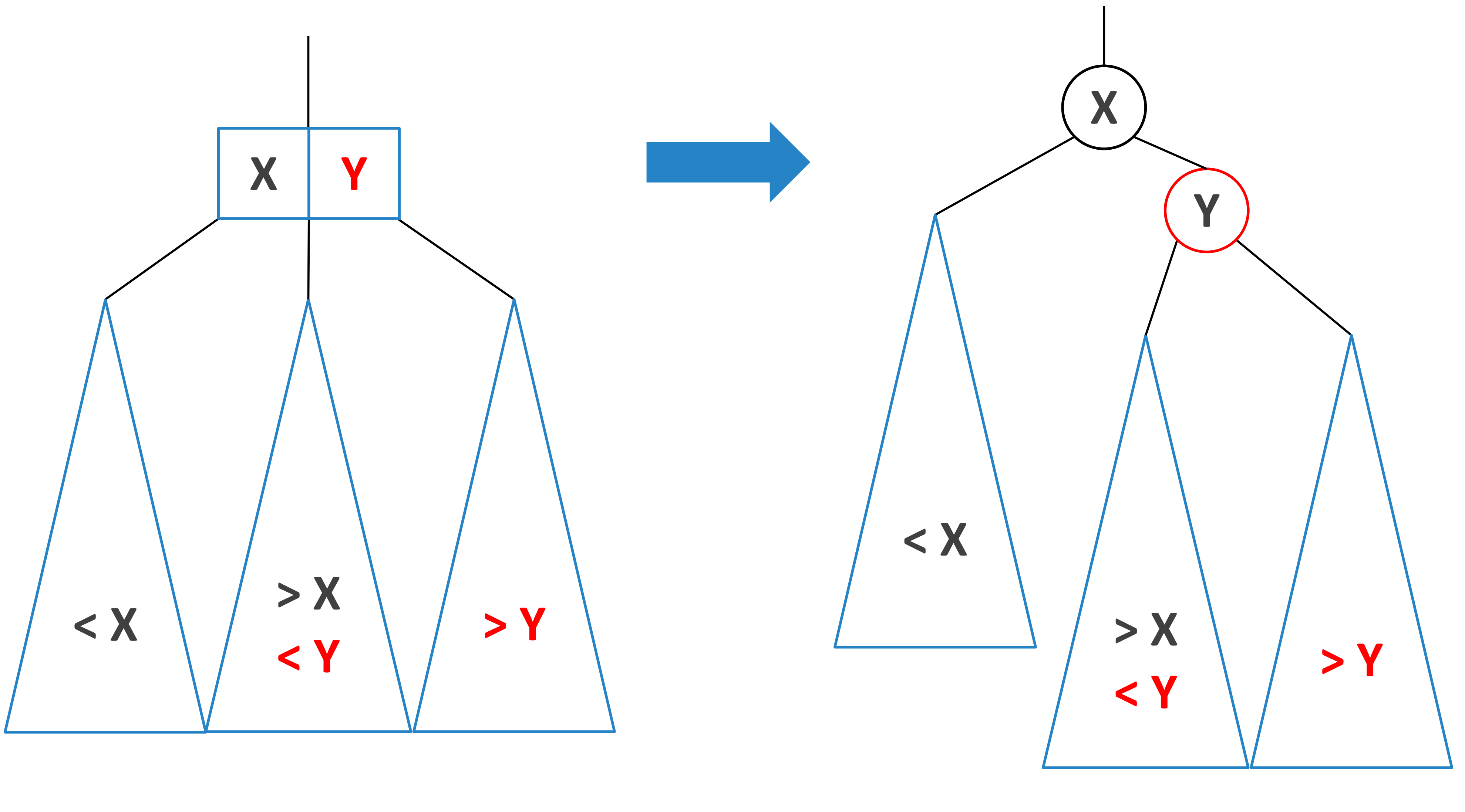
Nodo 2 como un nodo en un ABB



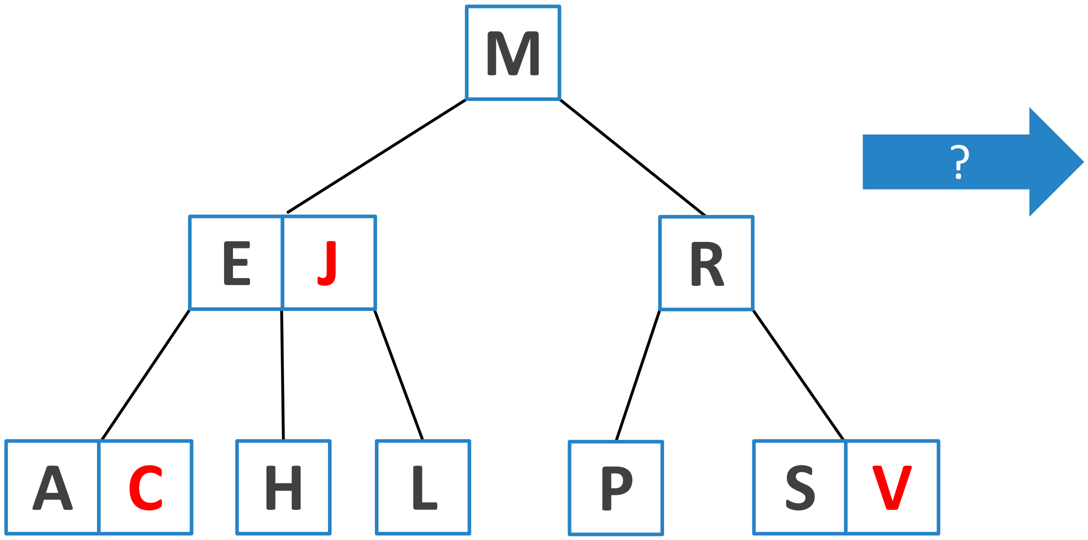
Nodo 3



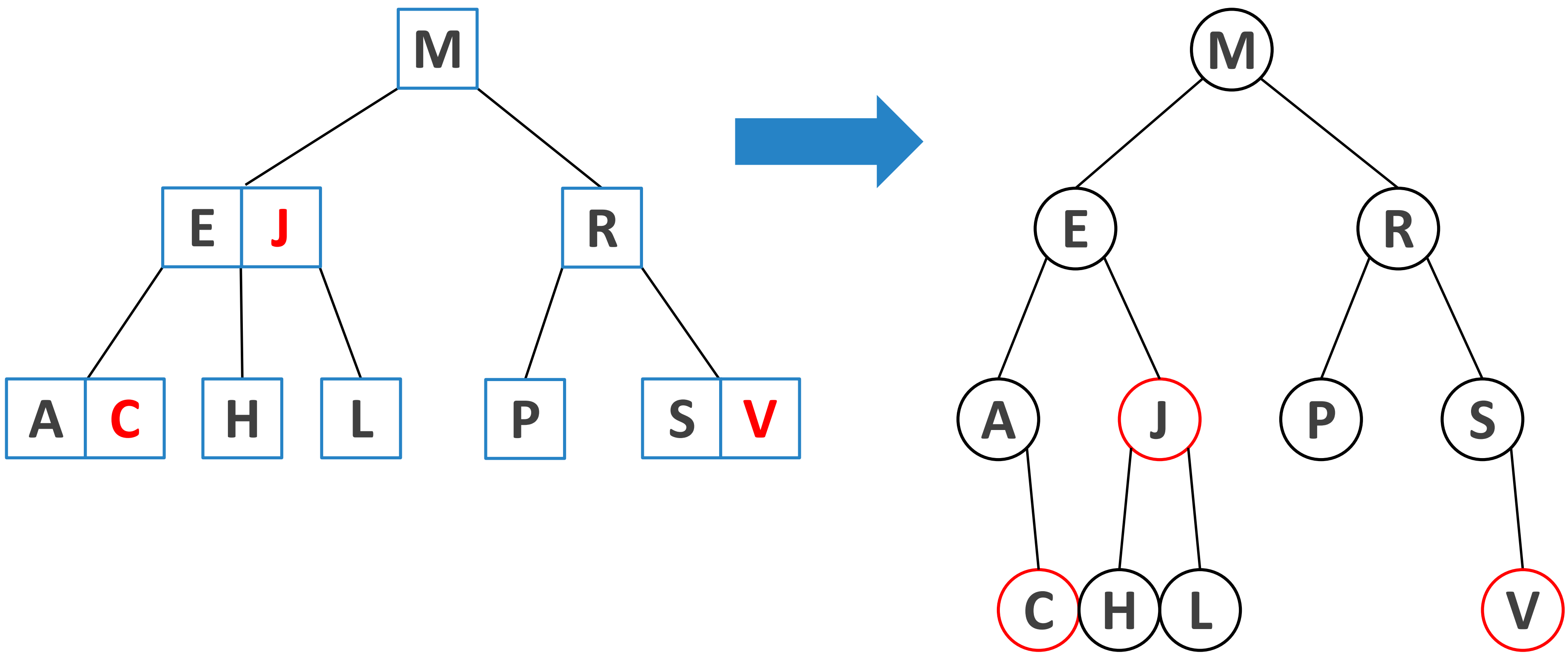
Nodo 3 como dos nodos en un ABB



Árbol 2-3 ...



Árbol 2-3 ... como ABB



El árbol resultante se conoce como **árbol rojo-negro**

Un **árbol rojo-negro** es un ABB que cumple cuatro propiedades:

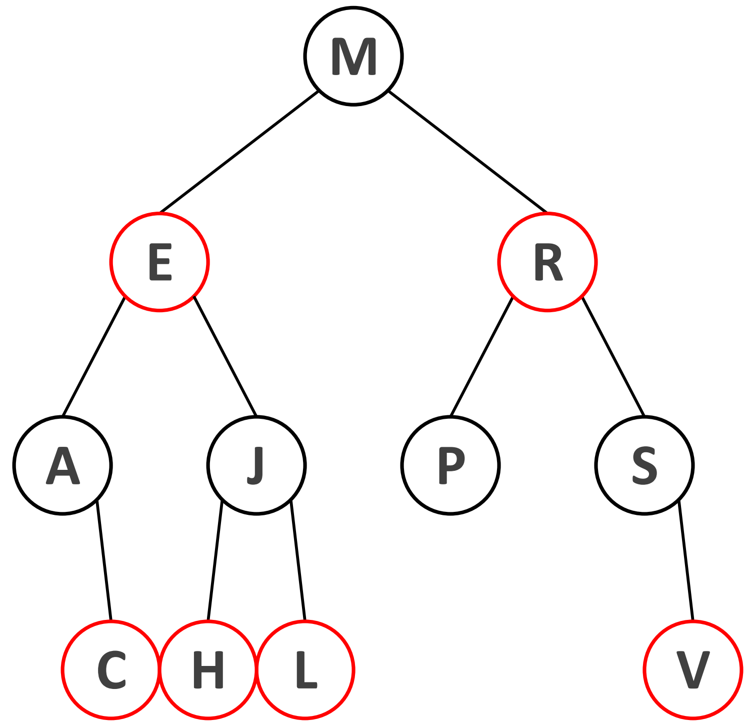
1) Cada nodo es ya sea **rojo** o **negro**
2) La raíz del árbol es **negra**
3) Si un nodo es **rojo**, sus hijos deben ser **negros**
4) La cantidad de nodos **negros** camino a cada hoja debe ser la misma

Las hojas nulas se consideran como nodos **negros**

El árbol resultante se conoce como **árbol rojo-negro**

Un **árbol rojo-negro** es un ABB que cumple cuatro propiedades:

1) Cada nodo es ya sea **rojo** o **negro**
2) La raíz del árbol es **negra**
3) Si un nodo es **rojo**, sus hijos deben ser **negros**
4) La cantidad de nodos **negros** camino a cada hoja debe ser la misma



Las hojas nulas se consideran como nodos **negros**

Inserción en un árbol rojo-negro

Una inserción puede violar las propiedades del árbol rojo-negro (así como ocurre en un árbol AVL)

Debemos restaurarlas, usando rotaciones (como en un AVL) y **cambios de color** (en lugar de ajustar el balance del nodo)

Es más fácil de ver si nos fijamos en el **árbol 2-3** equivalente

Equivalencia de árboles rojo-negro con los árboles ~~2-3~~ 2-4

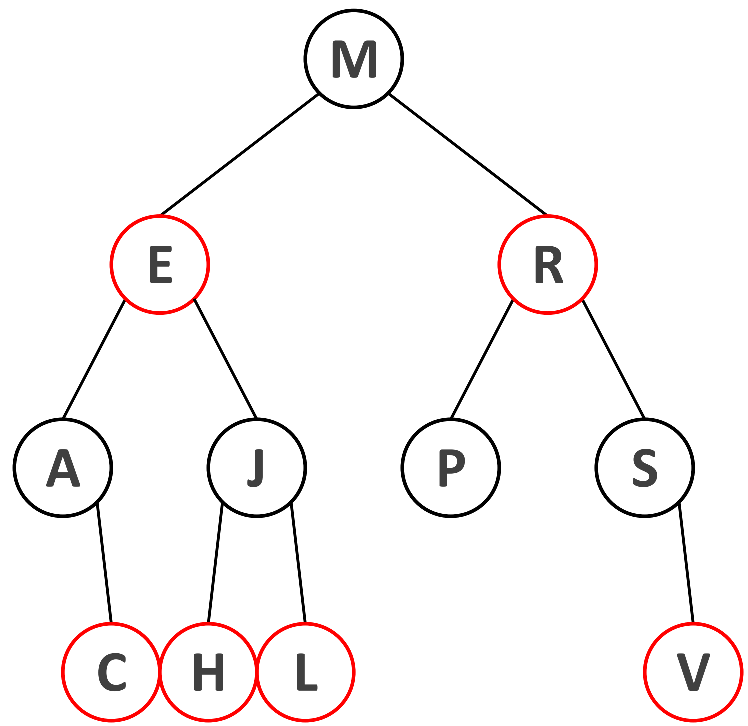
Bueno … no todos los árboles rojo-negro tienen un árbol 2-3 equivalente

…¡pero sí tienen un árbol 2-4 equivalente!

un **árbol 2-4** puede tener nodos 2 y nodos 3 (al igual que un árbol 2-3)

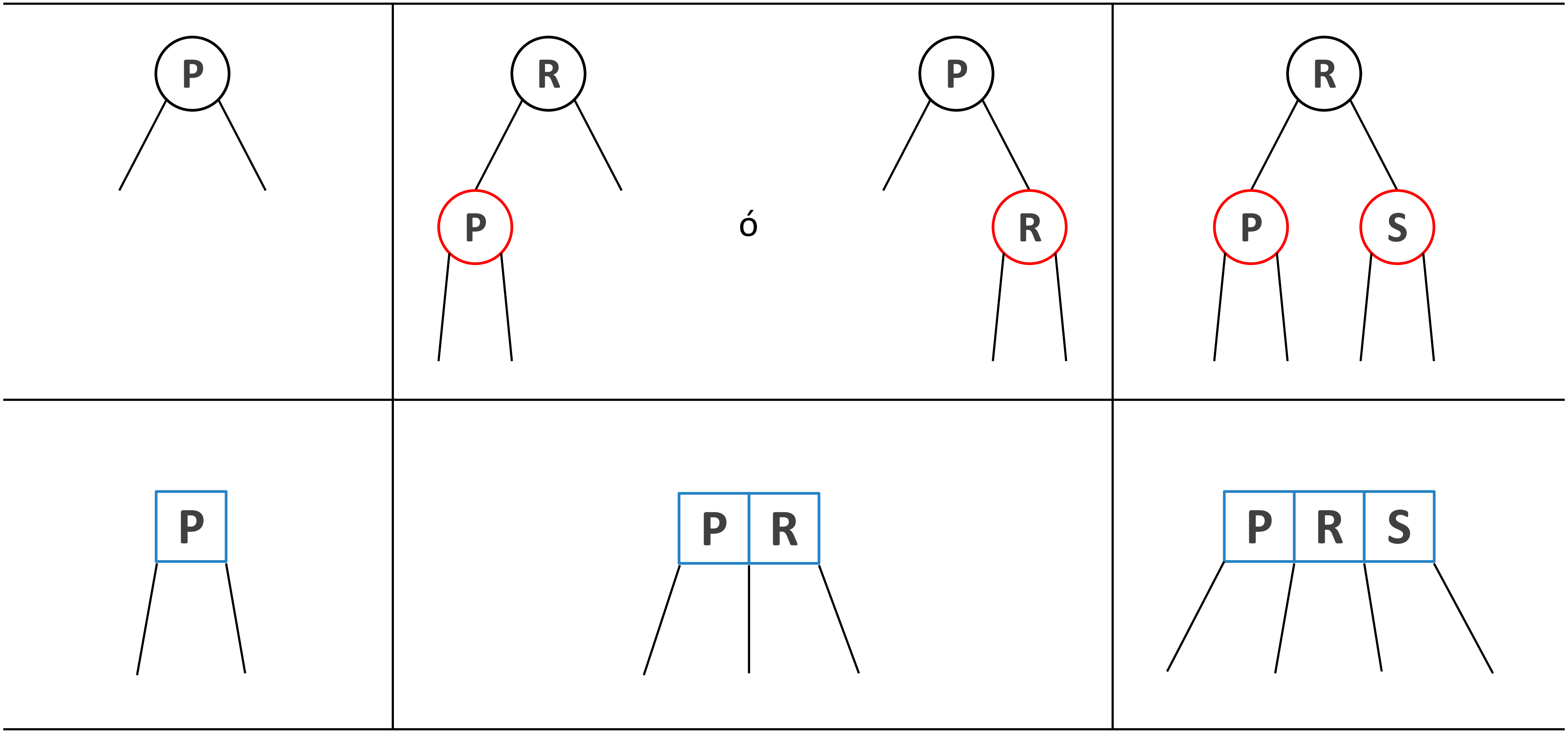
… y además puede tener **nodos 4**:

- 3 claves
- si no es una hoja, entonces 4 hijos



Equivalencia de los árboles rojo-negro con los **árboles 2-4**

Rojo Negro



2-4

¡Entonces hay que fijarse en el **árbol 2-4** equivalente!

(Un paréntesis

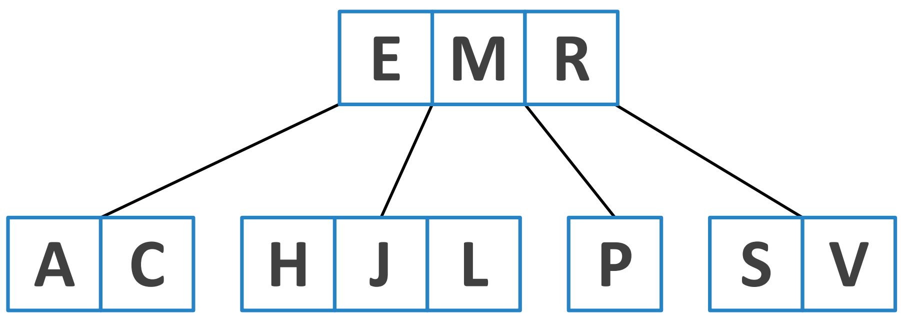
Para estudiar para las pruebas, simplemente revisar las diapositivas usadas en clases está **muy lejos de ser suficiente**:

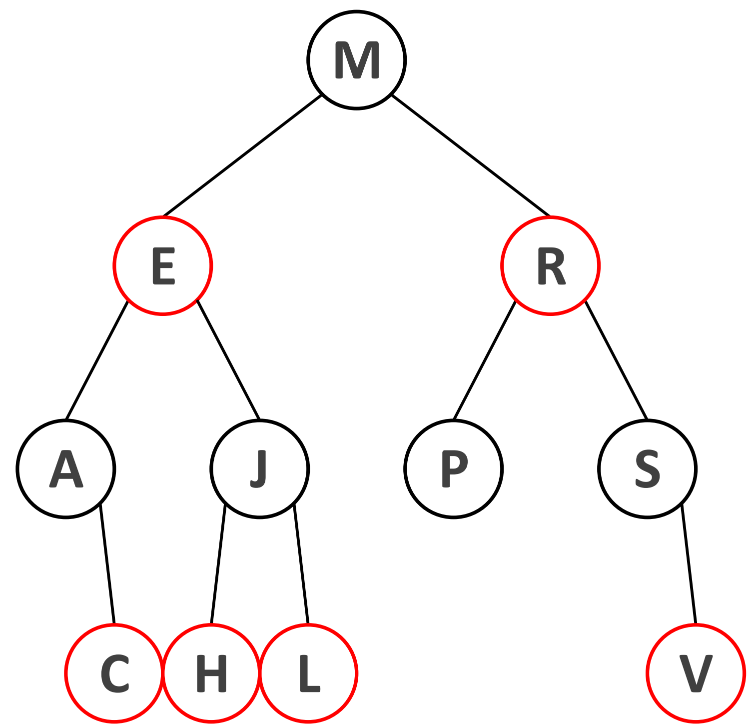
- estudiar los conceptos está bien
- … pero también hay que hacer muchos ejercicios

)

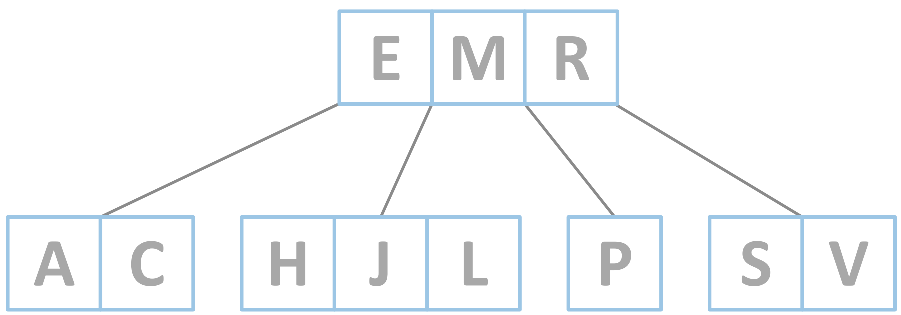
Ejemplo de inserción:

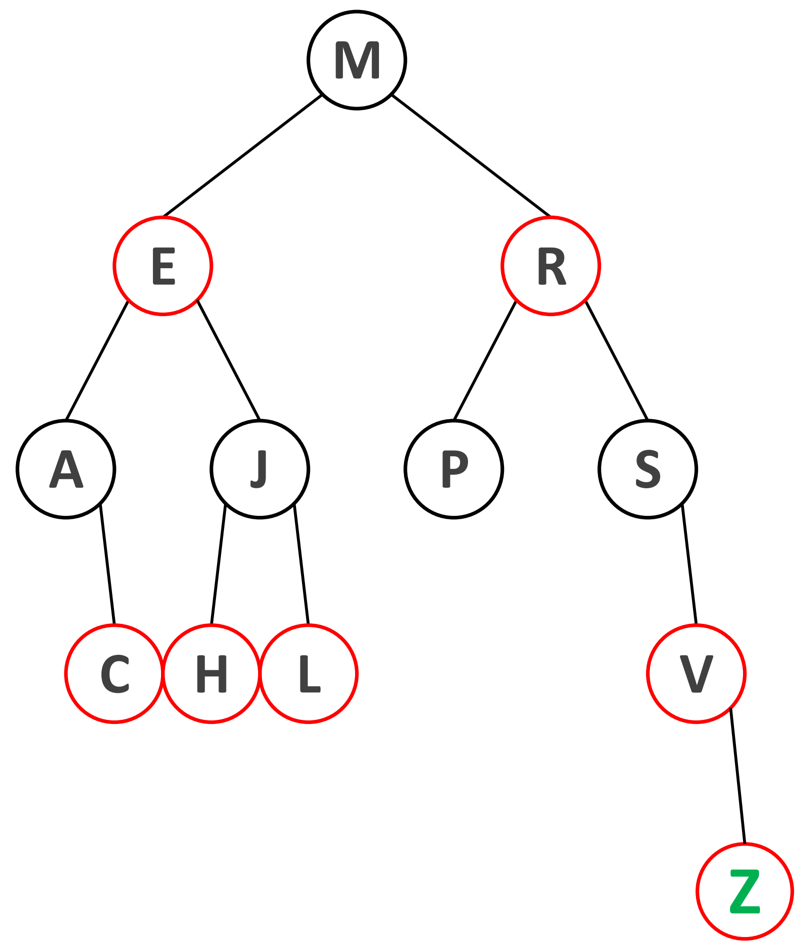
si insertamos la clave *Z*, ¿a dónde va a parar, inicialmente?





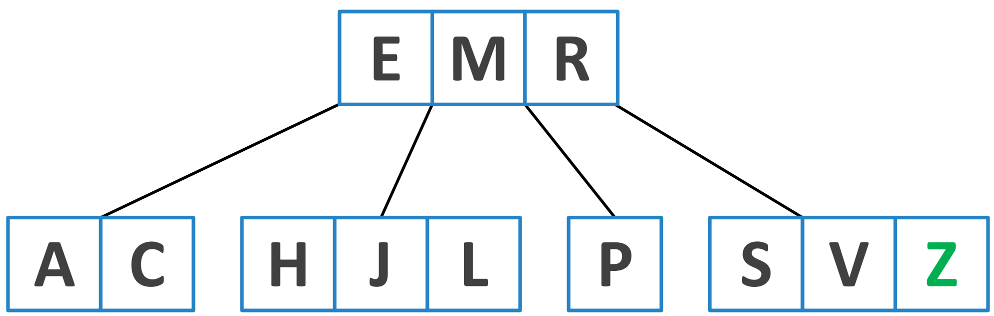
Insertemos la *Z* en el árbol rojo-negro

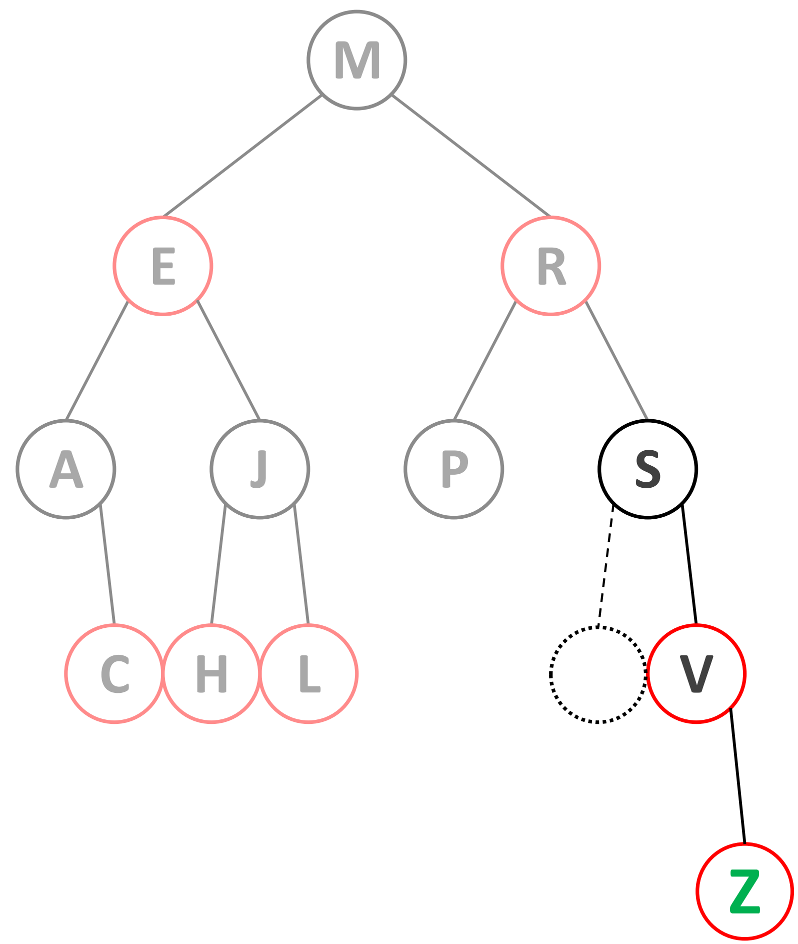




El nodo se inserta **rojo** (para no quebrantar la propiedad 4)

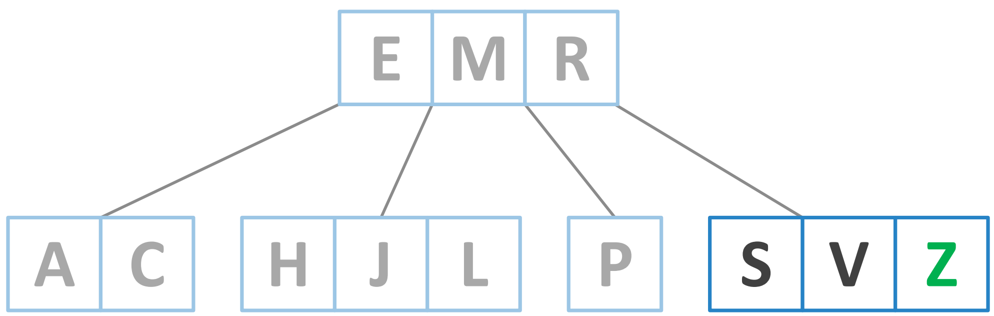
… y en el árbol 2-4

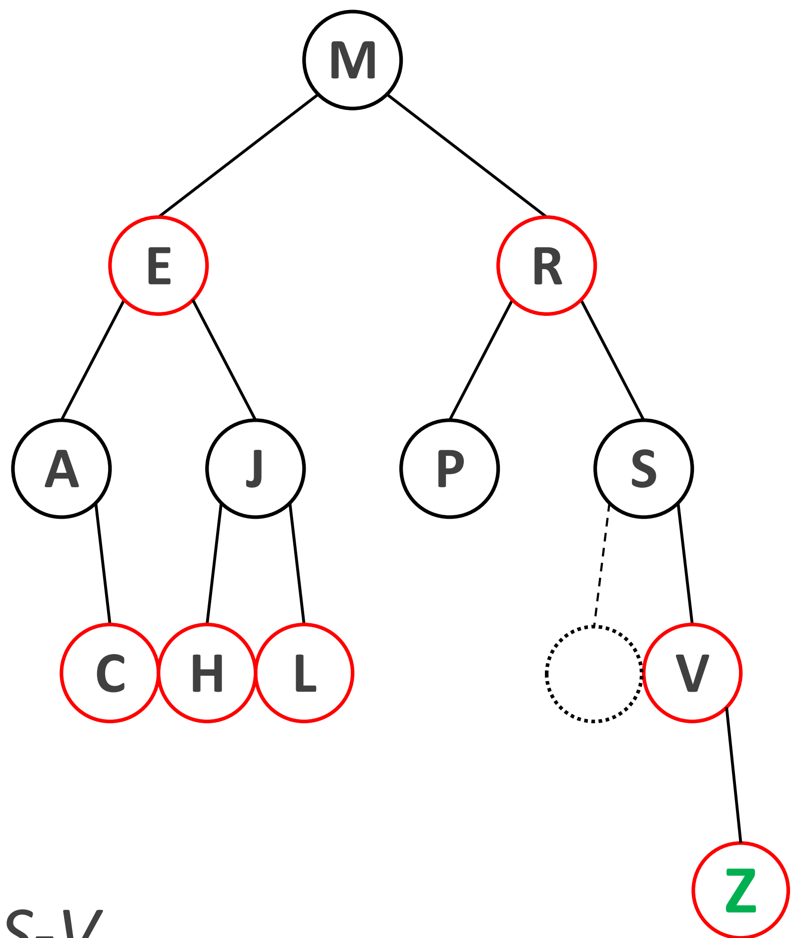




Observamos que el “tío” del nodo insertado es **negro**

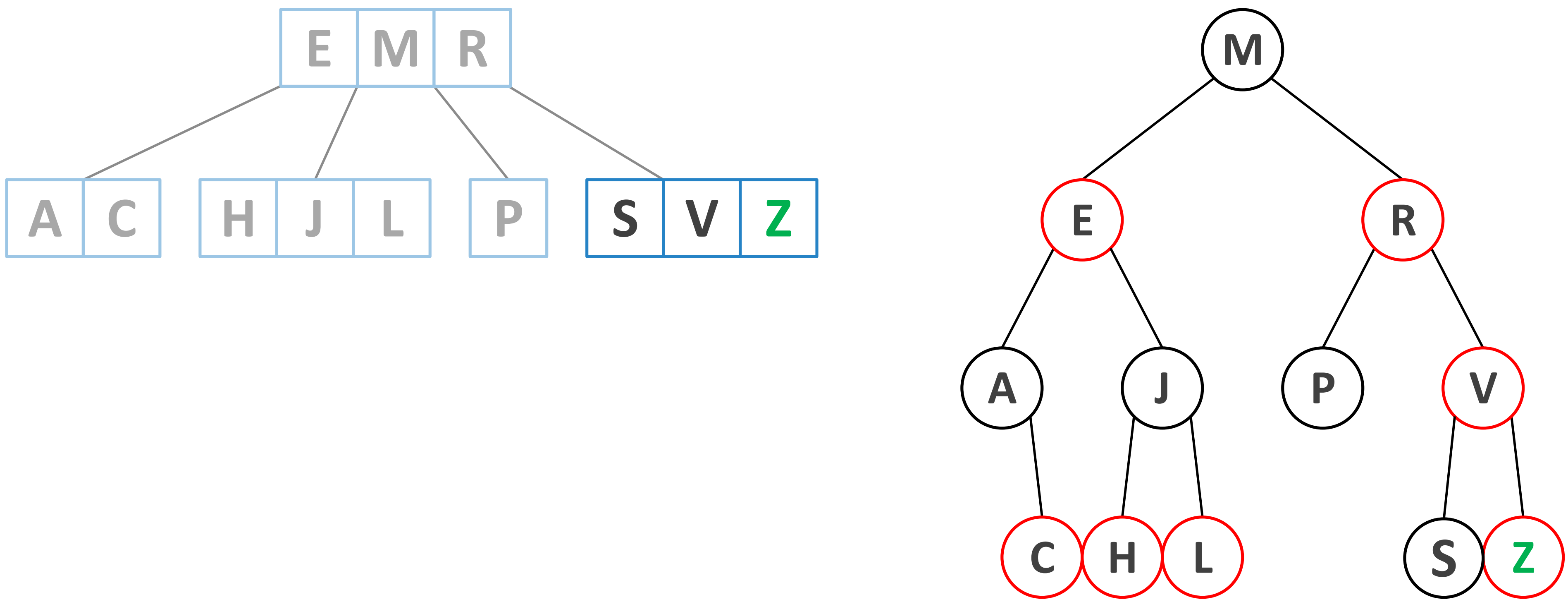
La configuración del nodo 4 "*S V Z*" nos sugiere qué hacer en el árbol rojo-negro





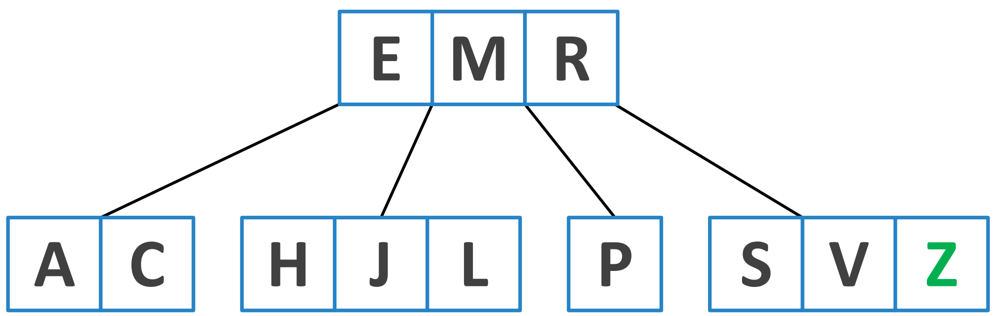
1) Rotación en torno a *S*-*V* ...

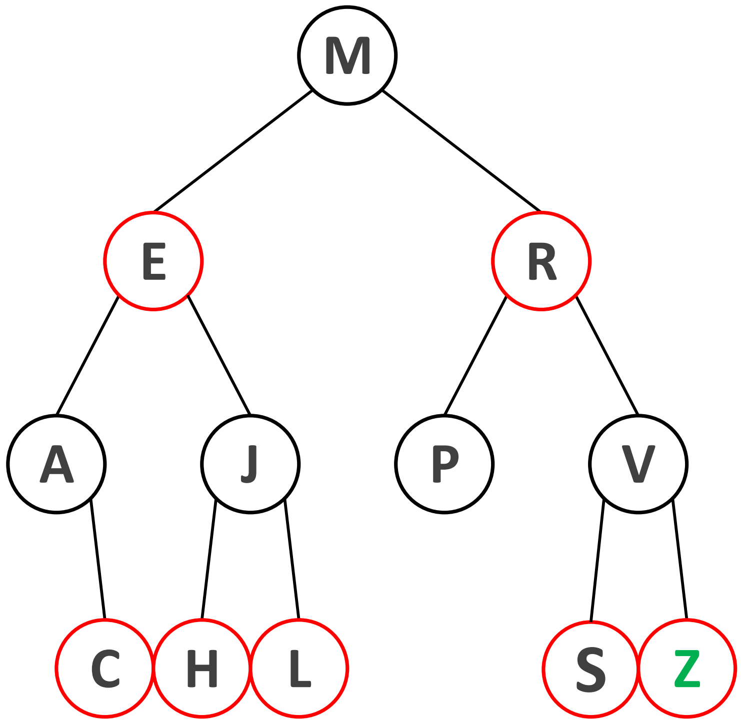
La sola rotación no es suficiente



2) Cambio de color a *S* y *V* ...

… también hay que cambiar colores

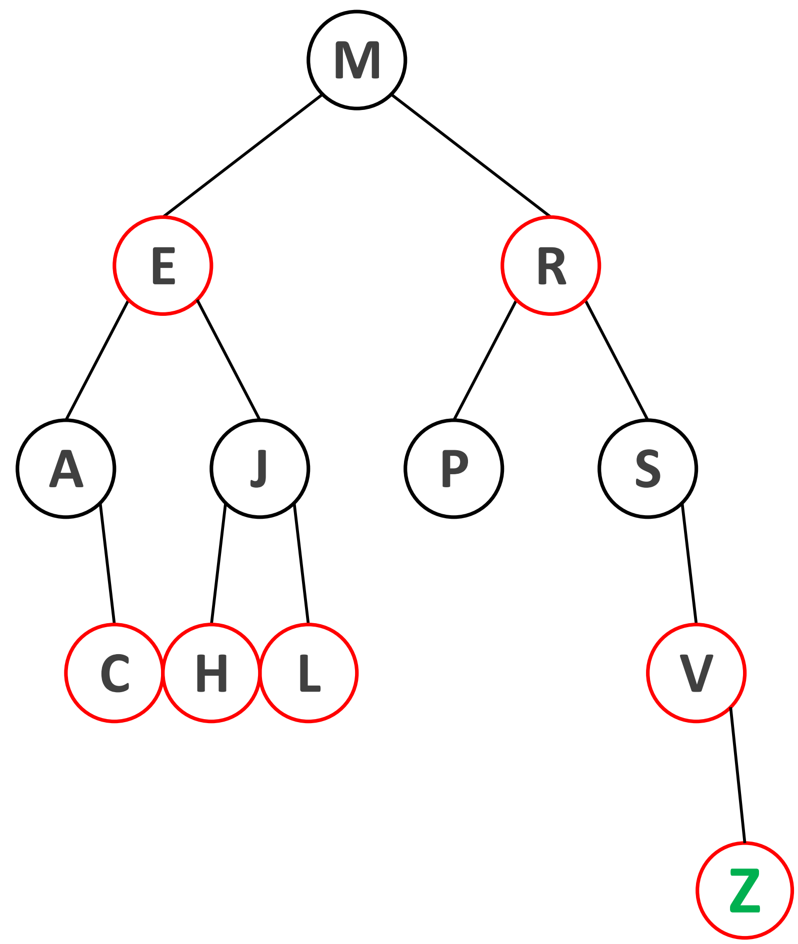




¡Listo!

Caso Z

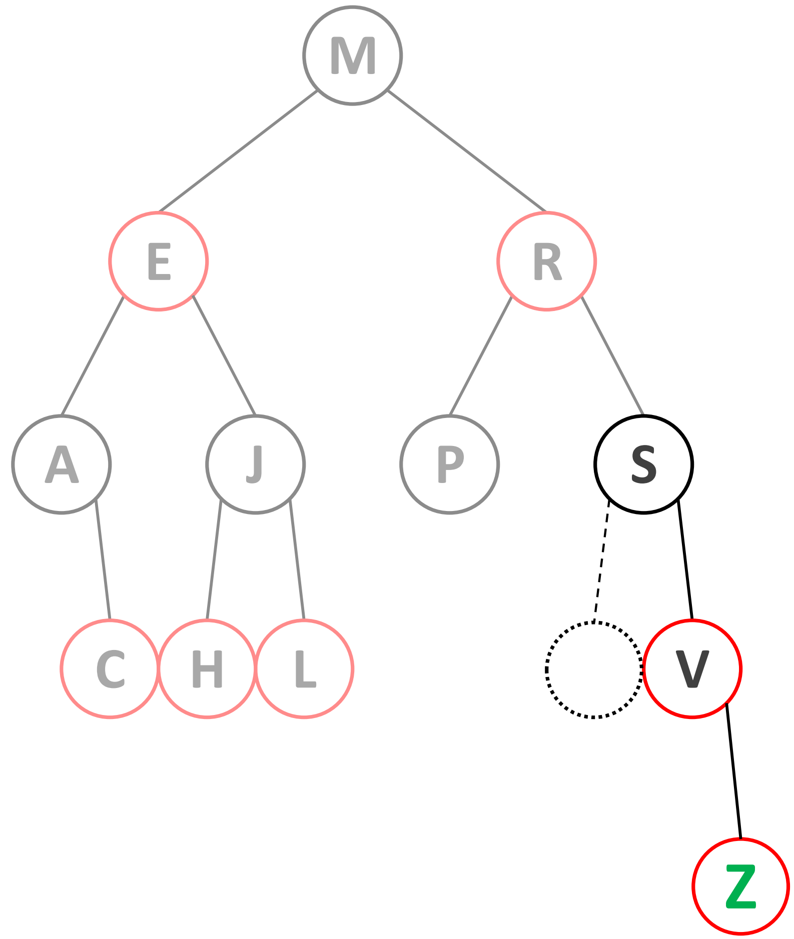
Si: insert "exterior"



Caso Z

Si: insert “exterior”

Si: tío “negro”

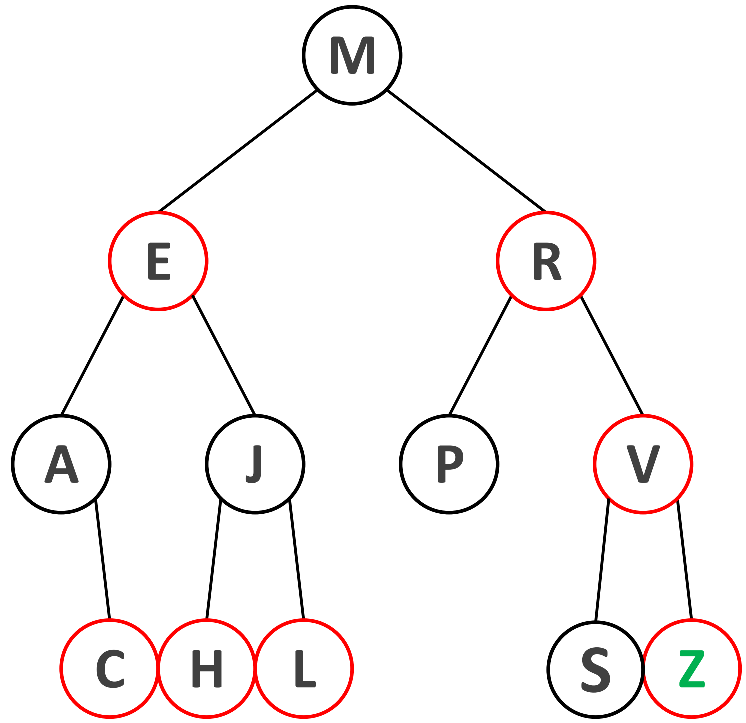


Caso Z

Si: insert "exterior"

Si: tío "negro"

rotación (abuelo z, padre z)



Caso Z

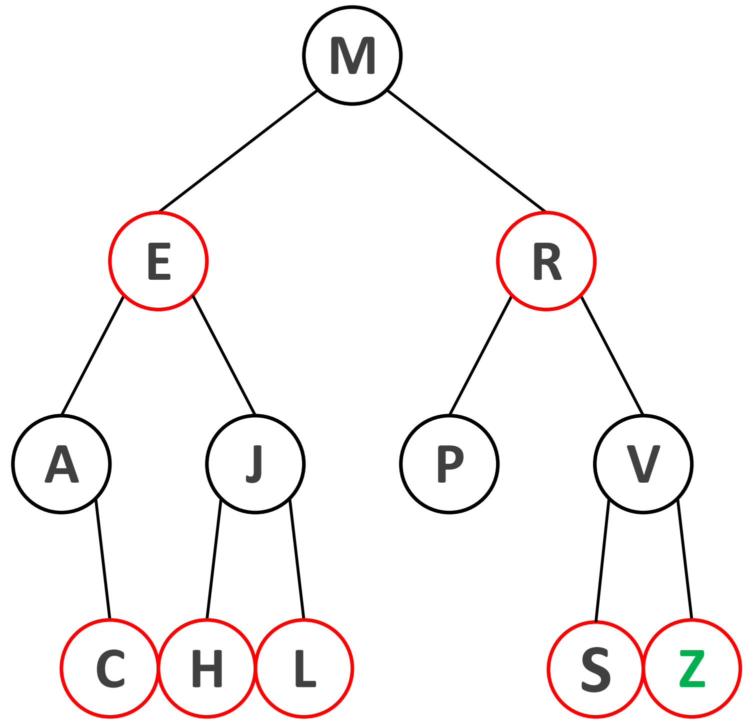
Si: insert "exterior"

Si: tío "negro"

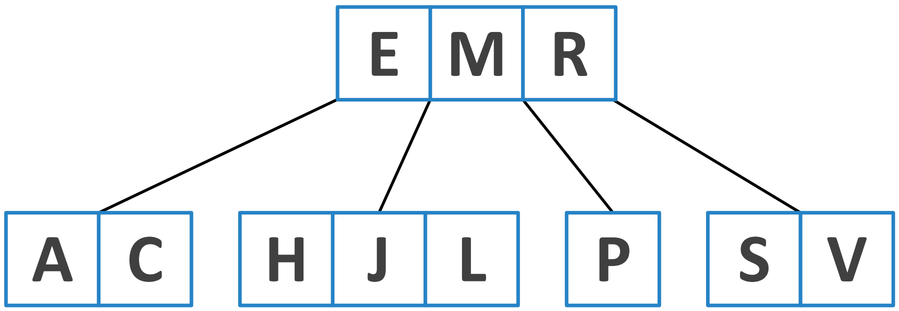
rotación (abuelo z, padre z)

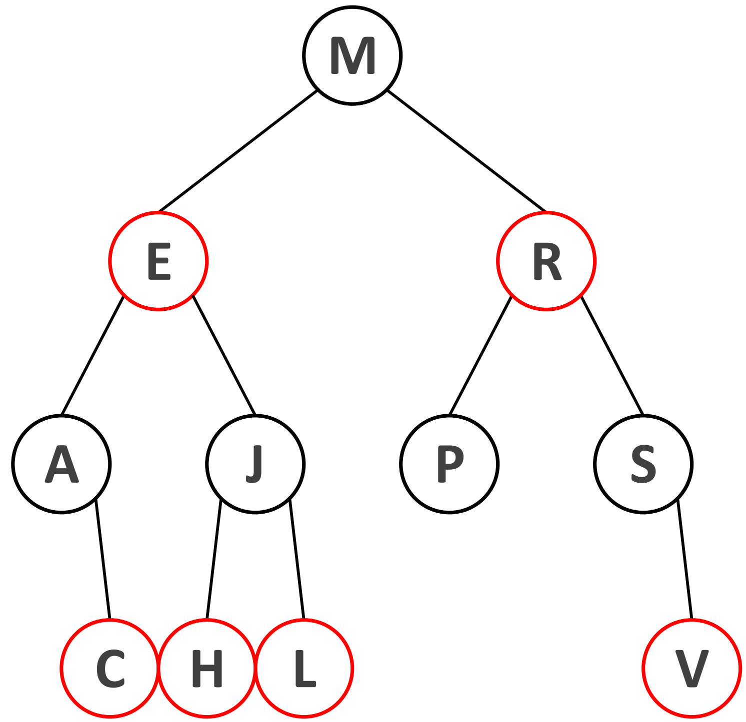
padre z <- negro

abuelo z <- rojo //original

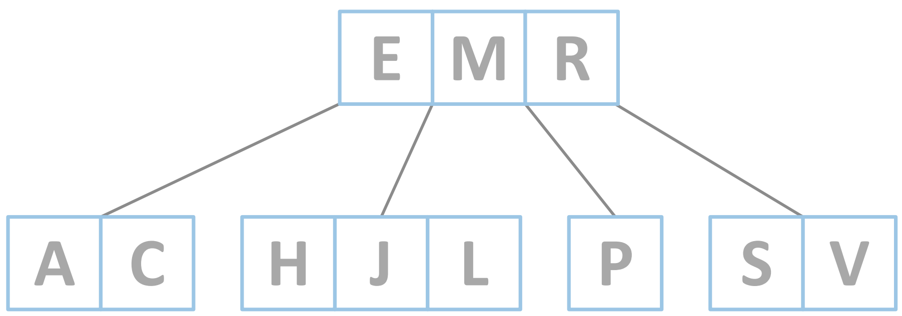


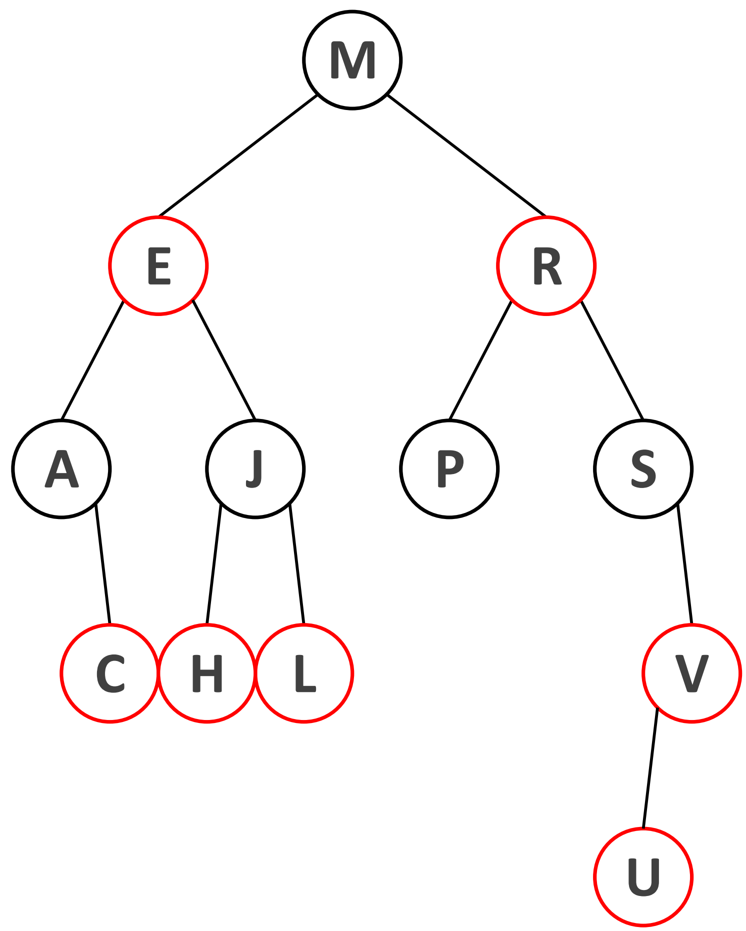
Veamos otra inserción en el árbol original: la clave *U*





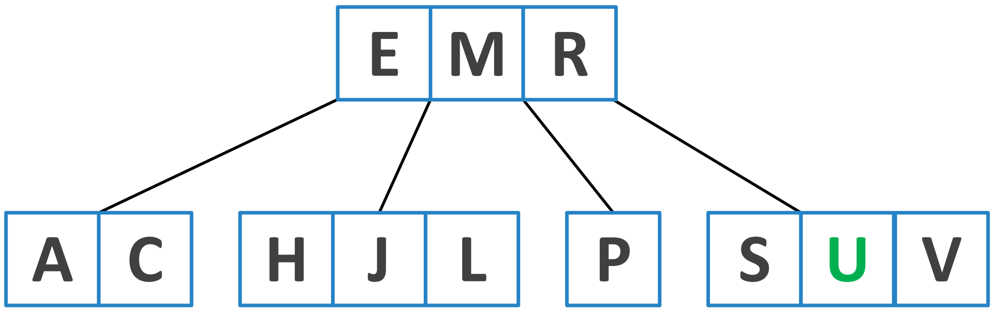
Insertemos la *U* en el rojo-negro

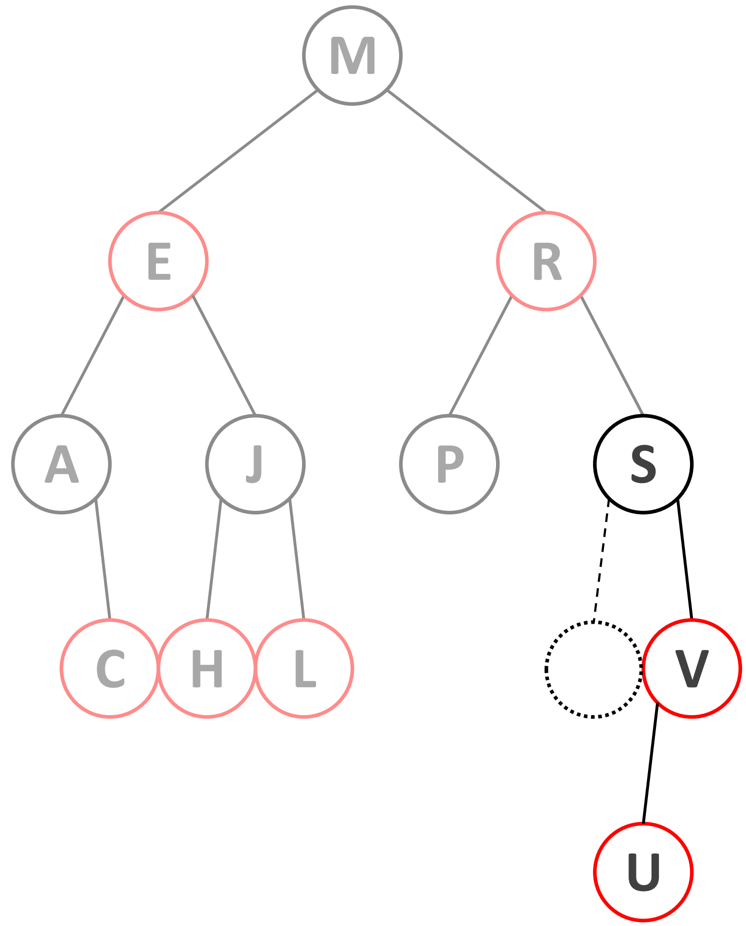




Nuevamente, el nodo recién insertado se pinta **rojo**

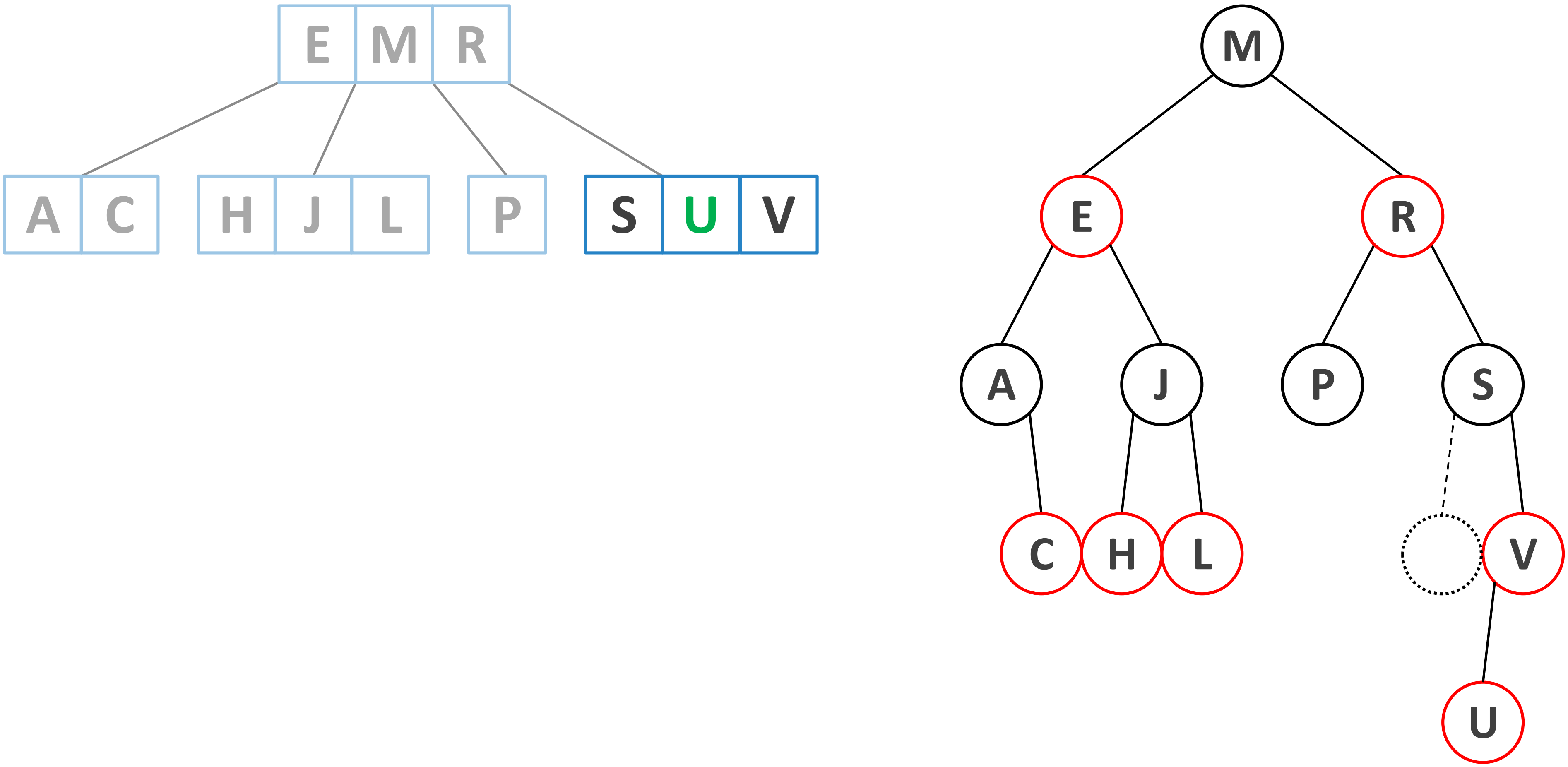
… y también en el 2-4





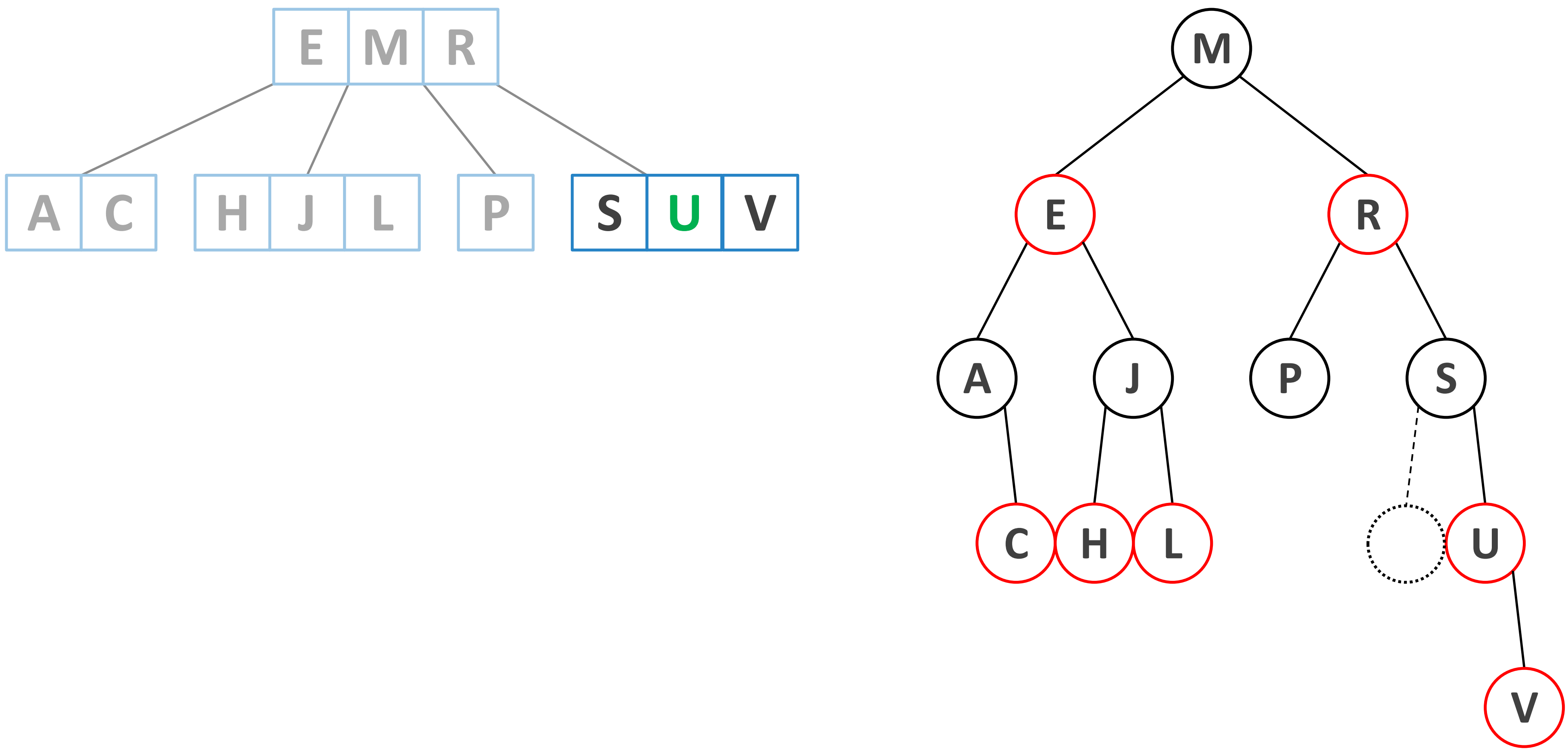
Nuevamente, el tío del nodo insertado es **negro**

La configuración del nodo “*S U V*” nos sugiere qué hacer en el árbol rojo-negro



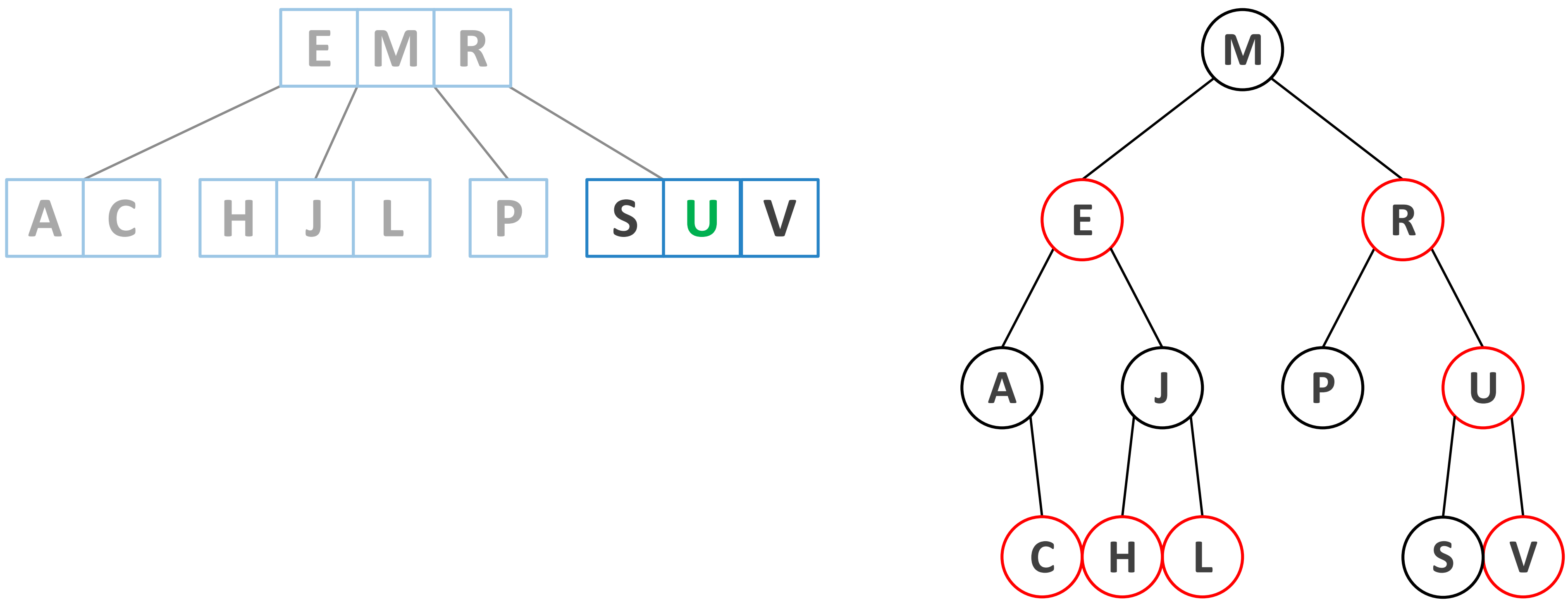
1) Rotación en torno a *U*-*V*

Una rotación no basta



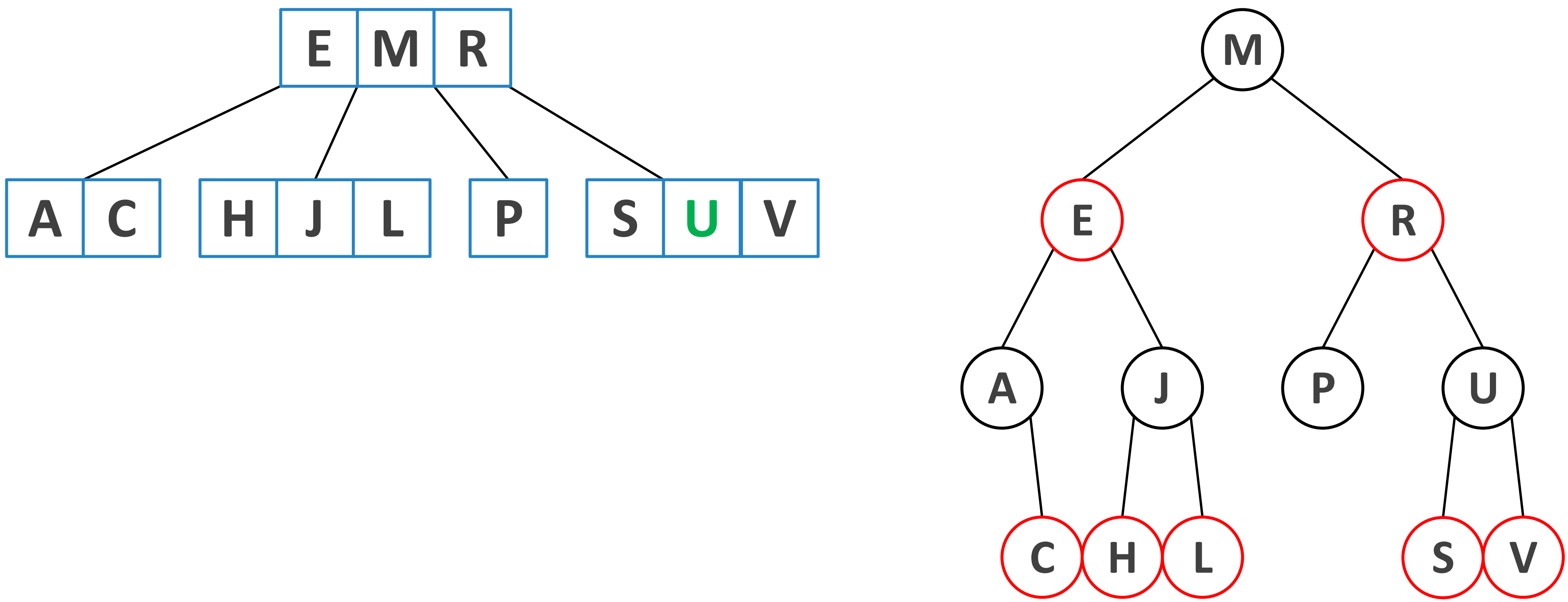
2) Segunda rotación, en torno a *S-U*

… hacemos una segunda rotación



3) Cambio de color de *S* y *U*

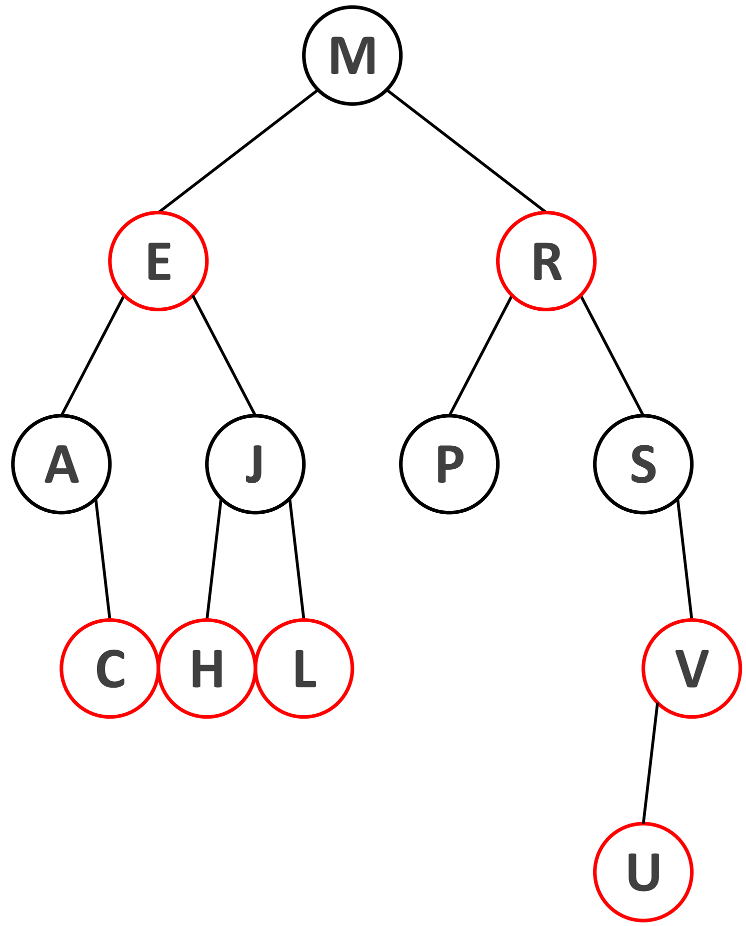
… y también cambiamos colores



¡Listo!

Caso U

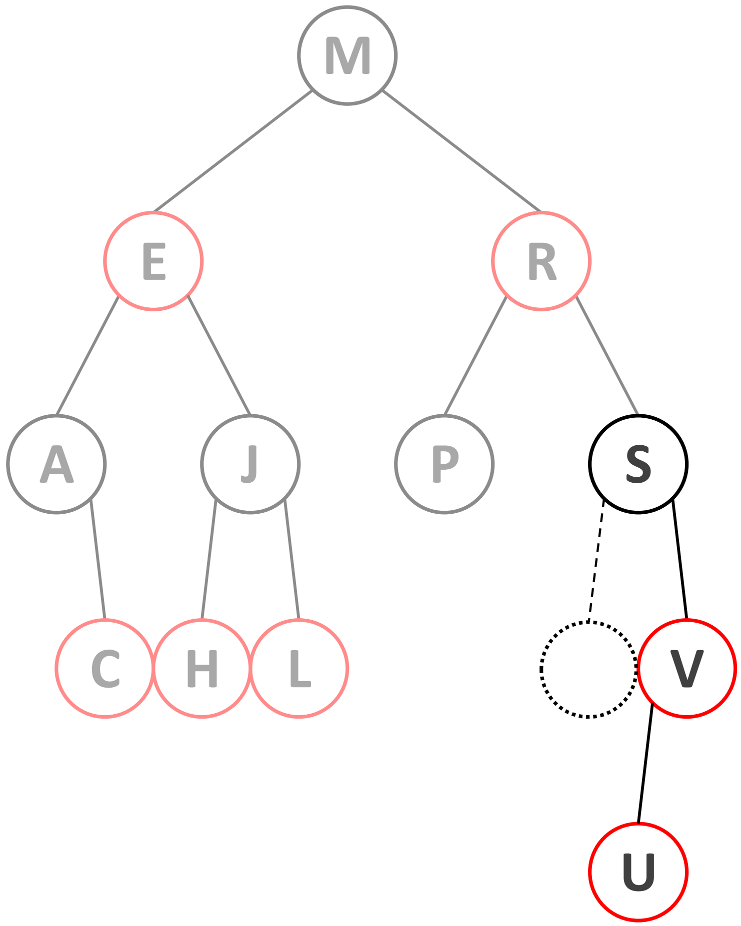
Si: insert “interior”



Caso U

Si: insert "interior"

Si: tío "negro"

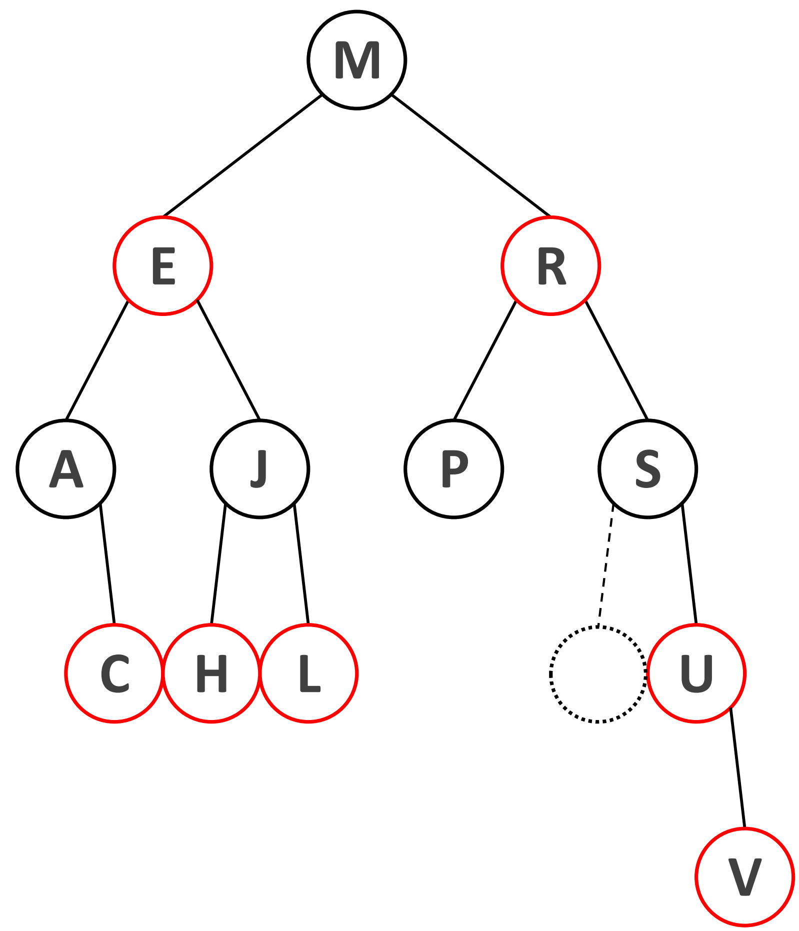


Caso U

Si: insert "interior"

Si: tío "negro"

rotación (padre u, u)



"¡ estamos en el mismo caso de Z !"

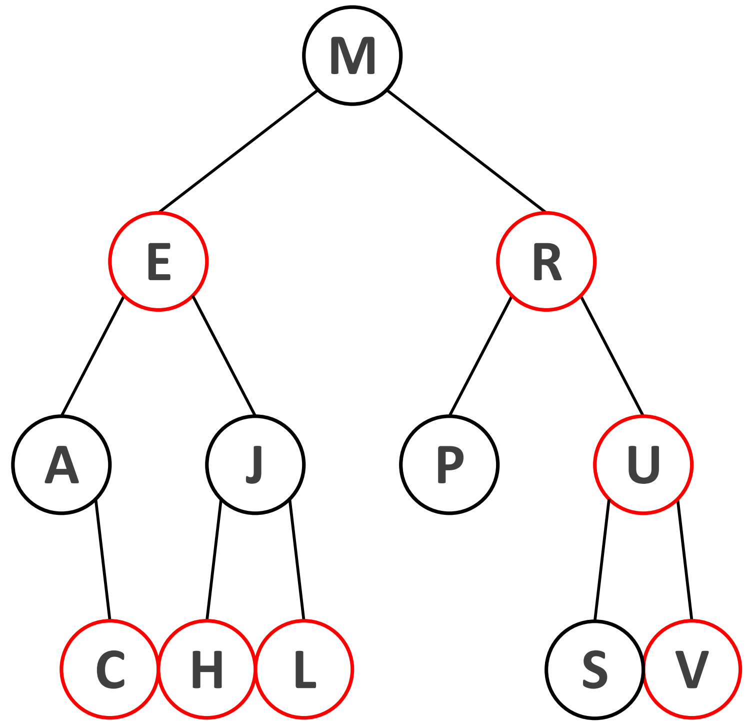
Caso U

Si: insert "interior"

Si: tío "negro"

rotación (padre u, u)

rotación (abuelo u, u)

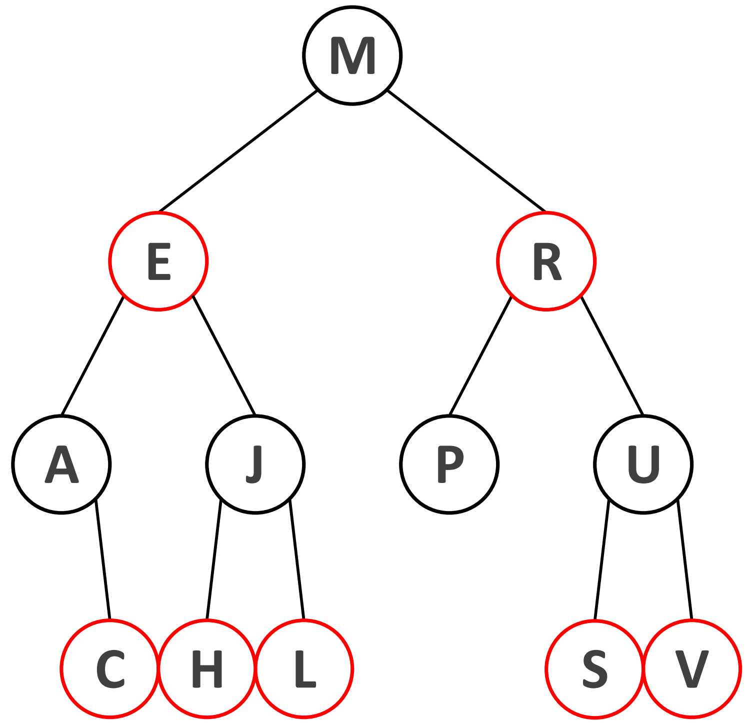


Caso U

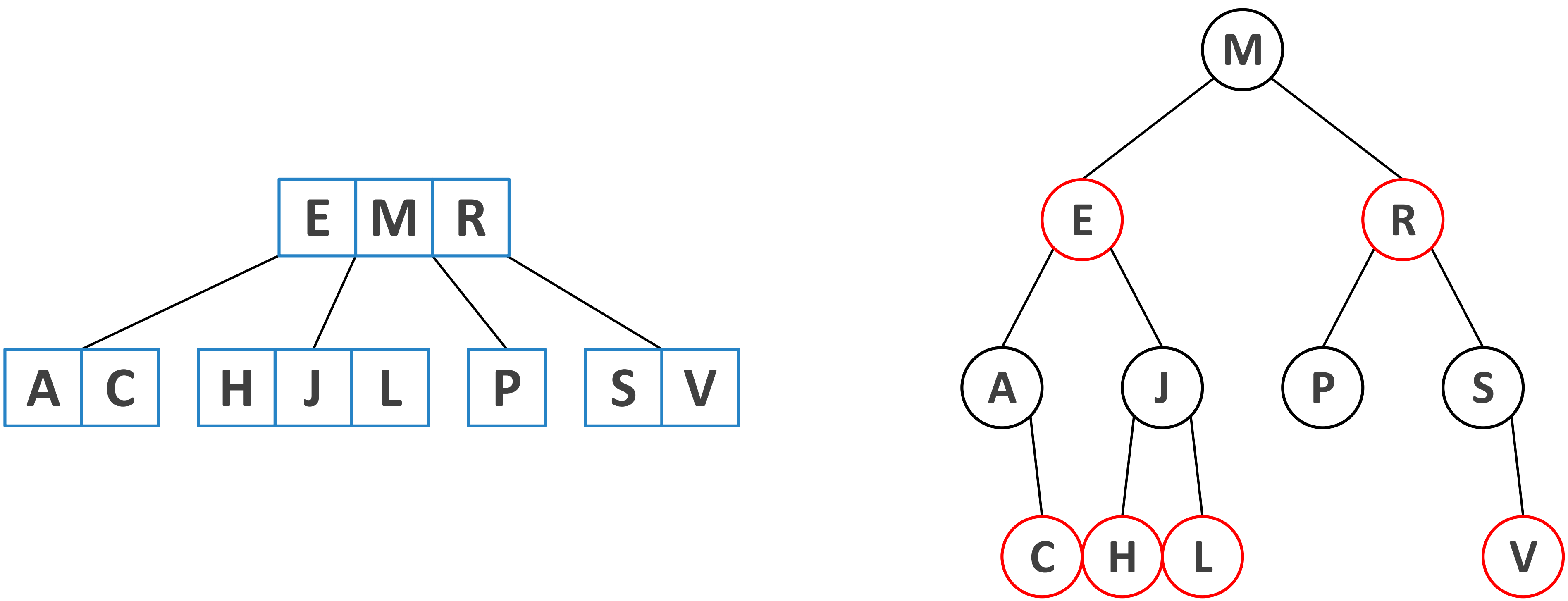
Si: insert “interior”

Si: tío “negro”

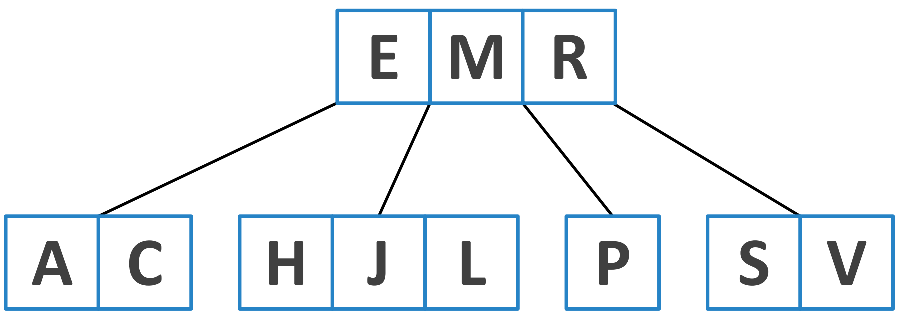
> **rotación (padre u, u)**
>
> **rotación (abuelo u, u)**
>
> **u <- Negro**
>
> **abuelo u <- rojo // original**

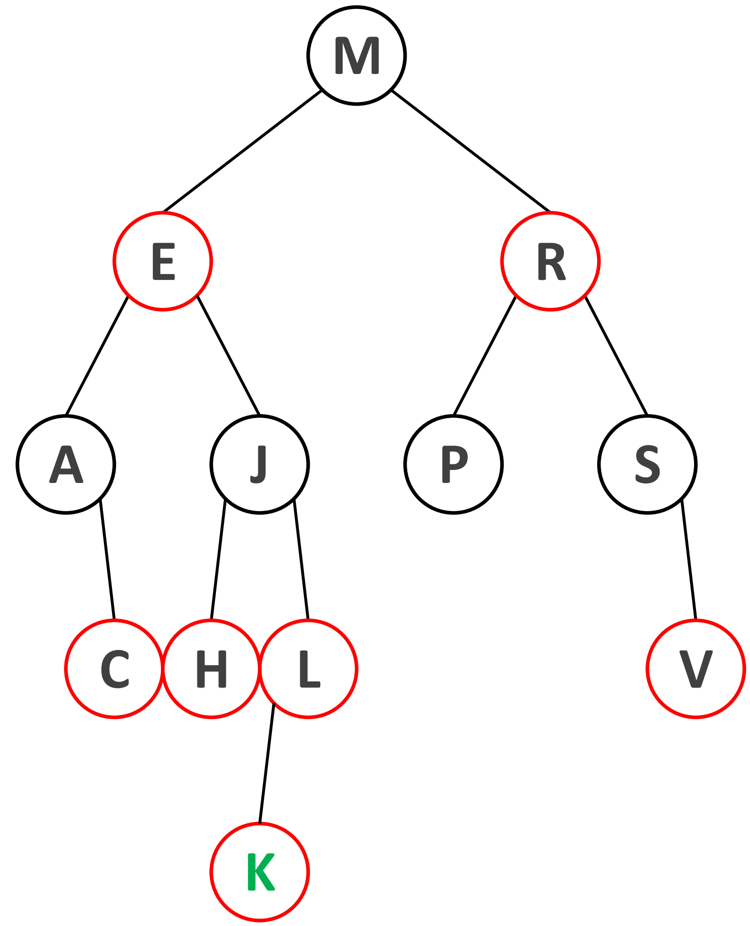


Hagamos una tercera inserción en el árbol original: la clave *K*



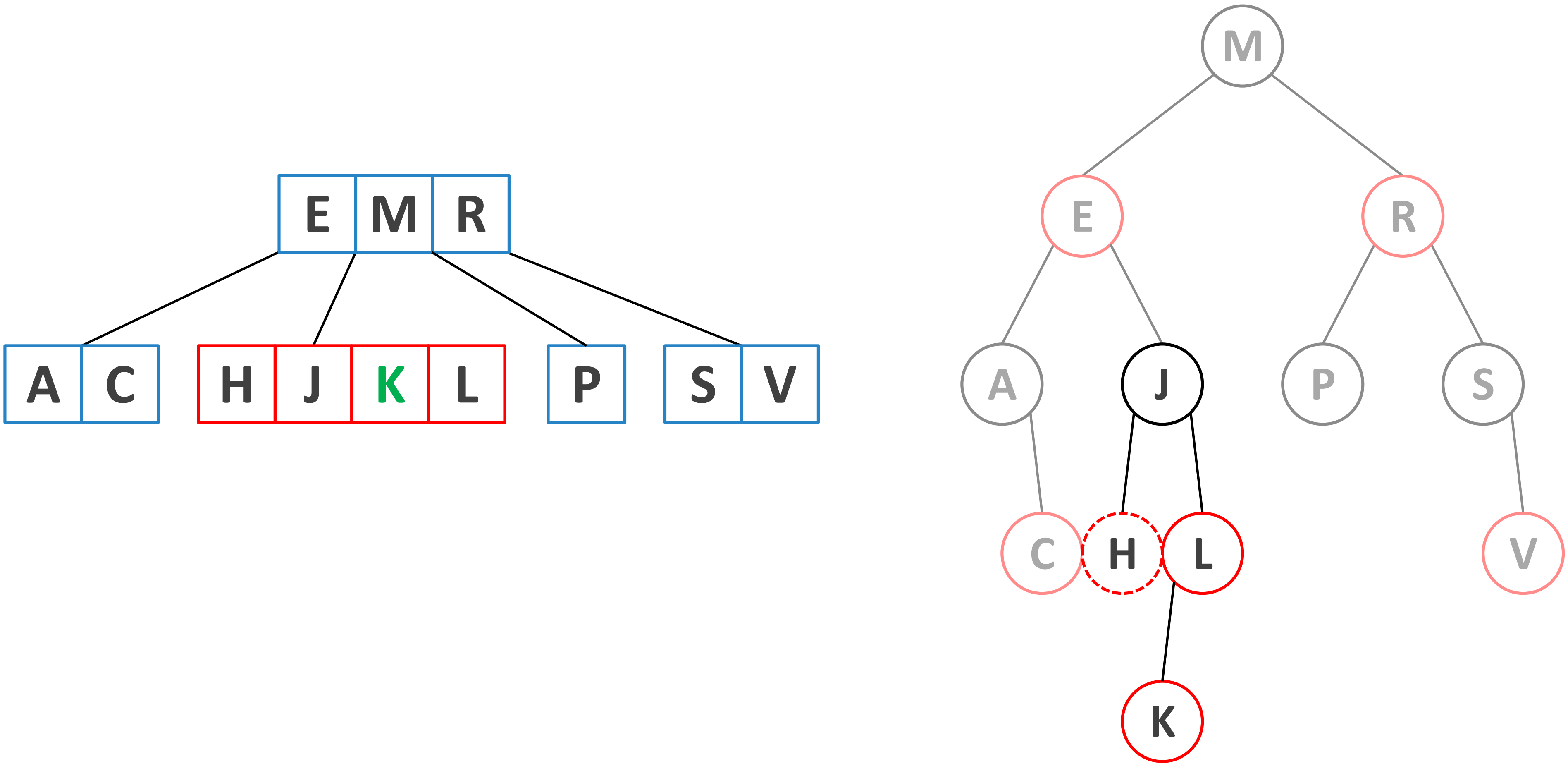
Insertemos la *K* en el árbol rojo-negro





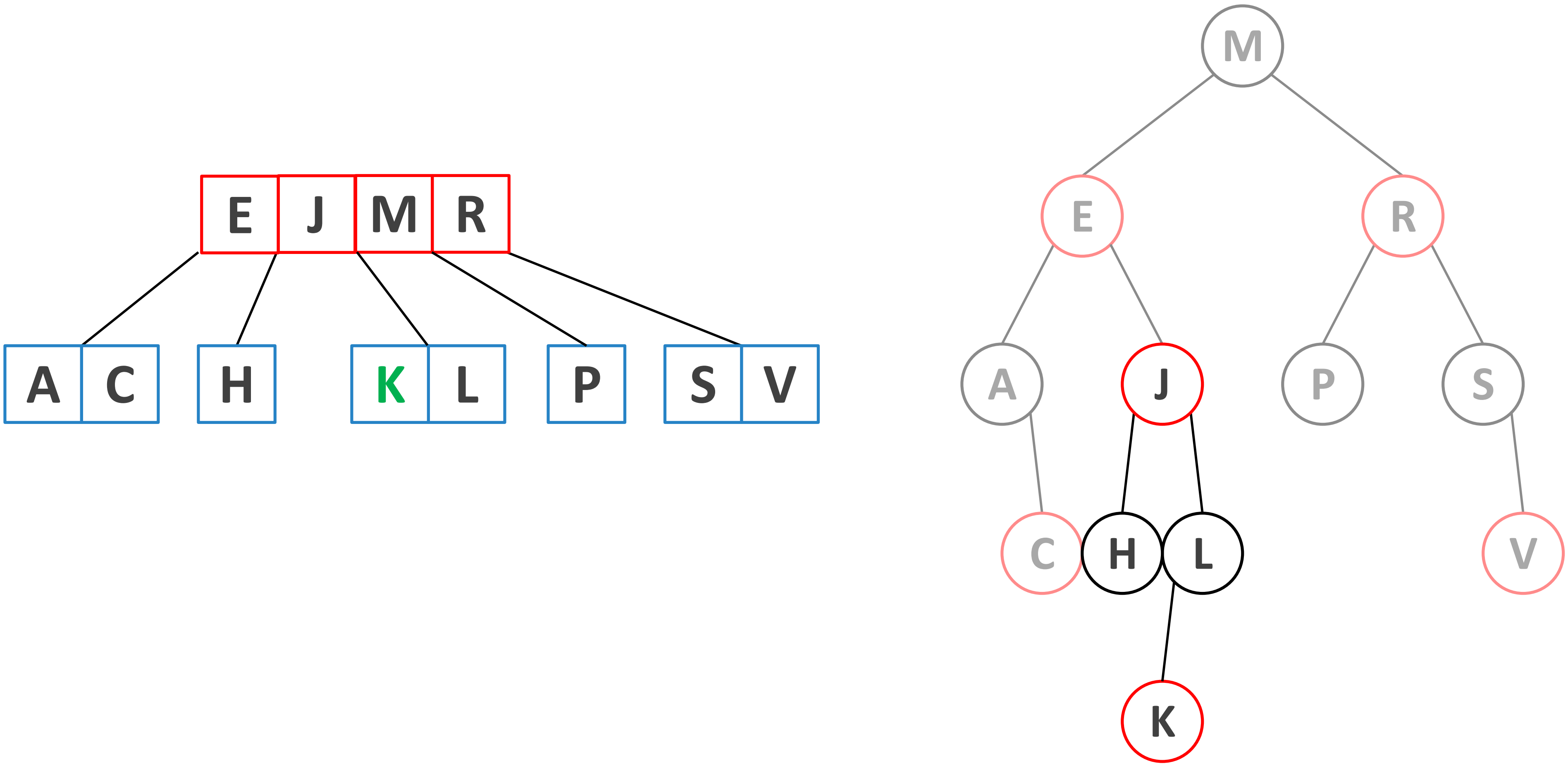
El nodo se inserta **rojo**

… y también en el árbol 2-4



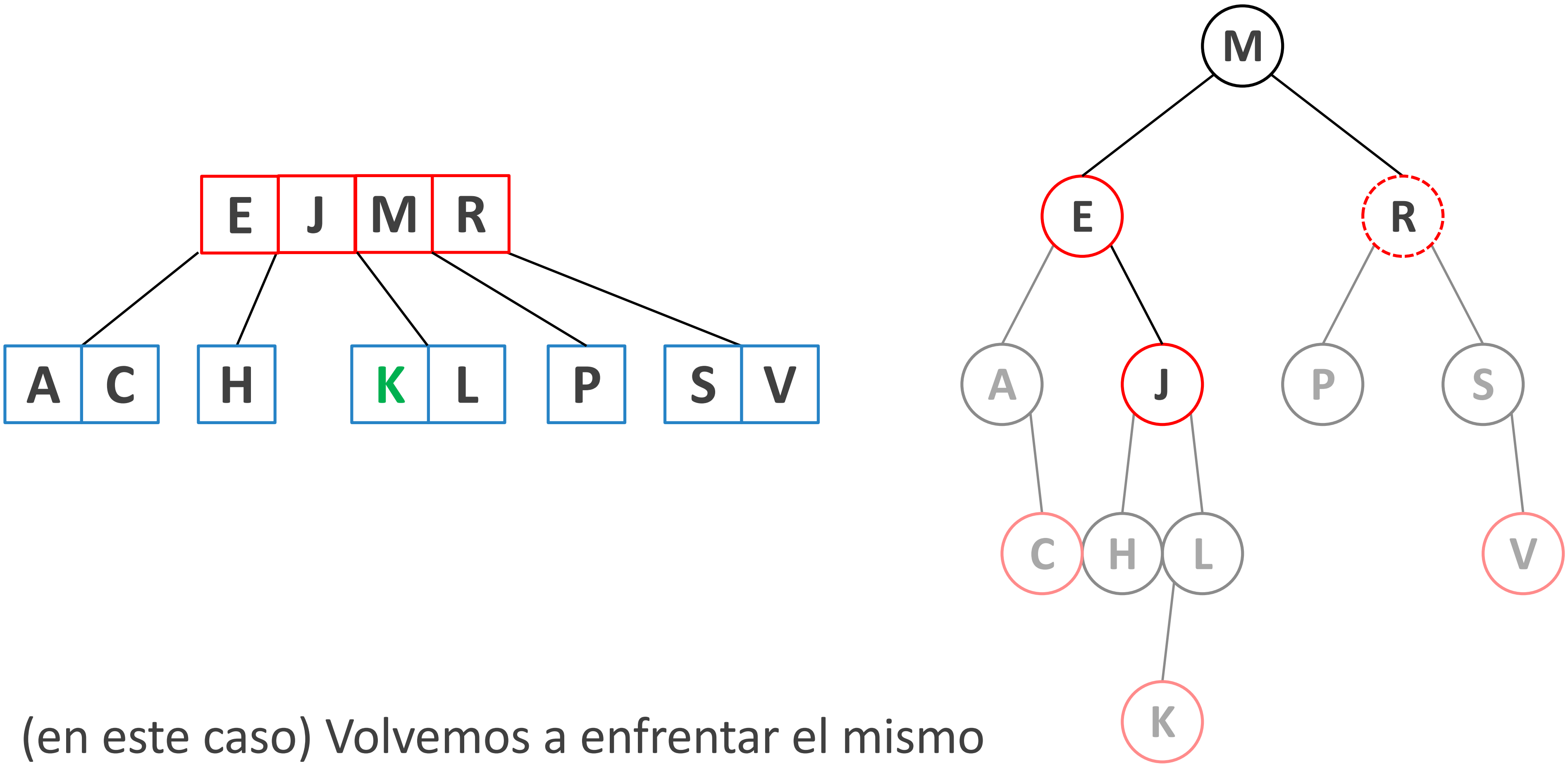
El tío del nodo insertado es **rojo**

¿Qué pasa en el árbol 2-4
y cómo se refleja en el árbol rojo-negro?



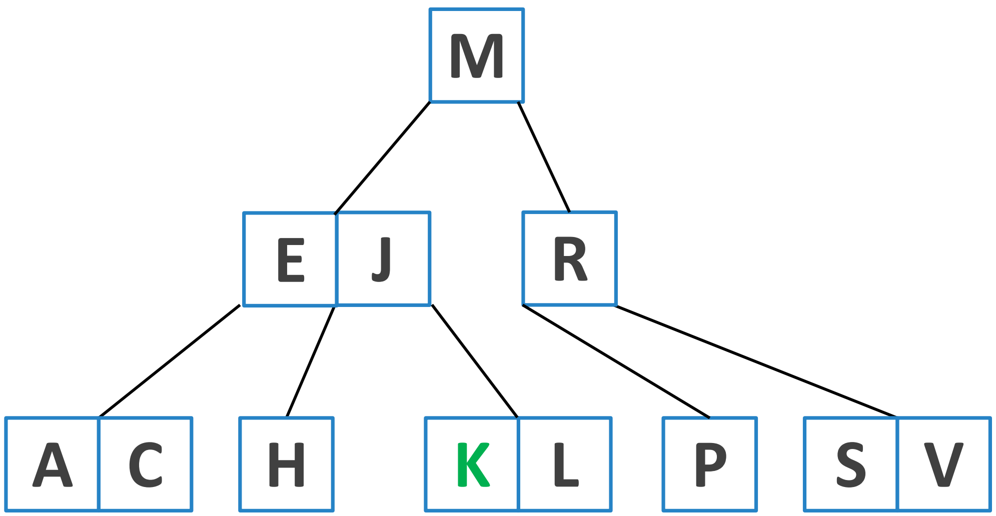
1) Cambio de color

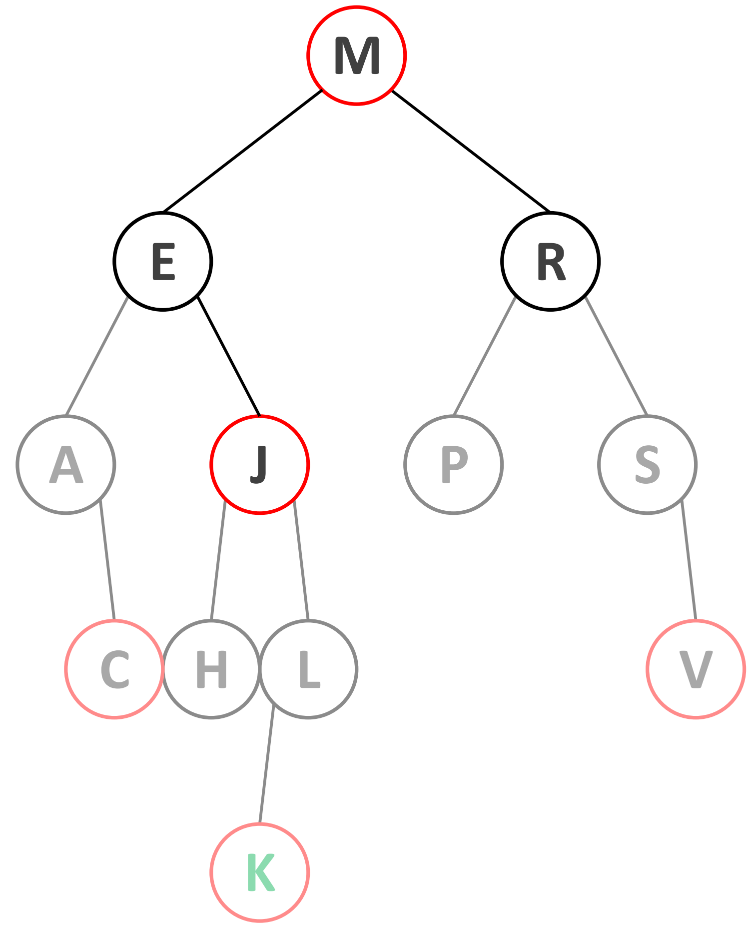
"Subimos" el problema de un nodo rojo con un hijo rojo



(en este caso) Volvemos a enfrentar el mismo problema: El tío del nodo con clave *J* es **rojo**

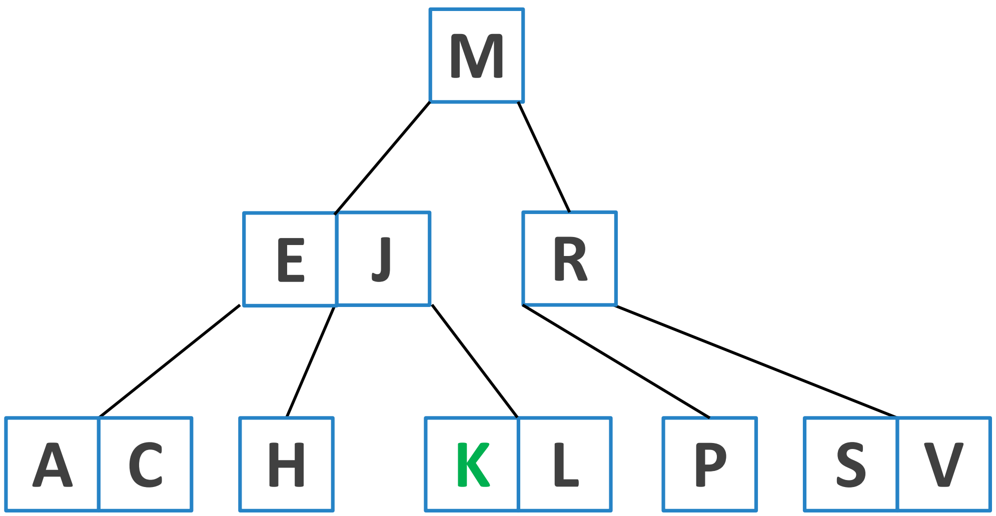
En el árbol 2-4 creamos una nueva raíz “arriba” de la que había

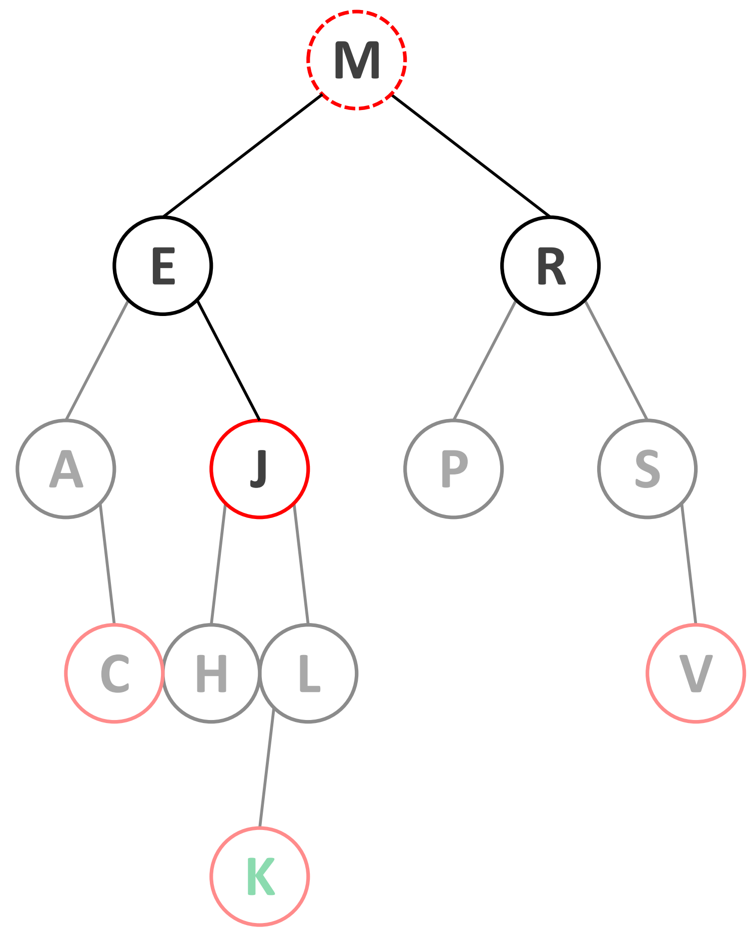




2) (recursivamente) Cambio de color

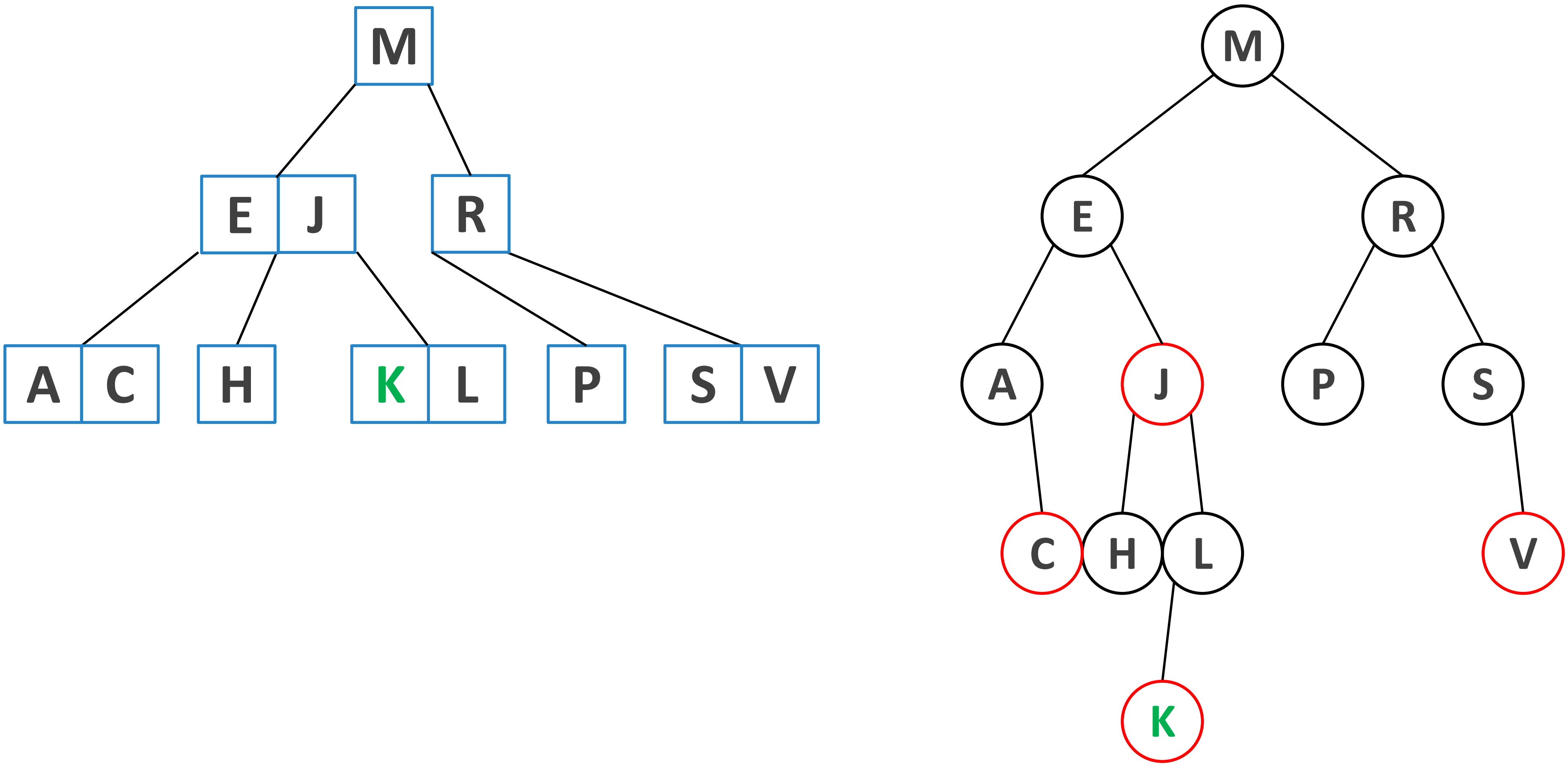
En el árbol rojo-negro, si la raíz se vuelve roja ...





3) La raíz es roja: se cambia a negro

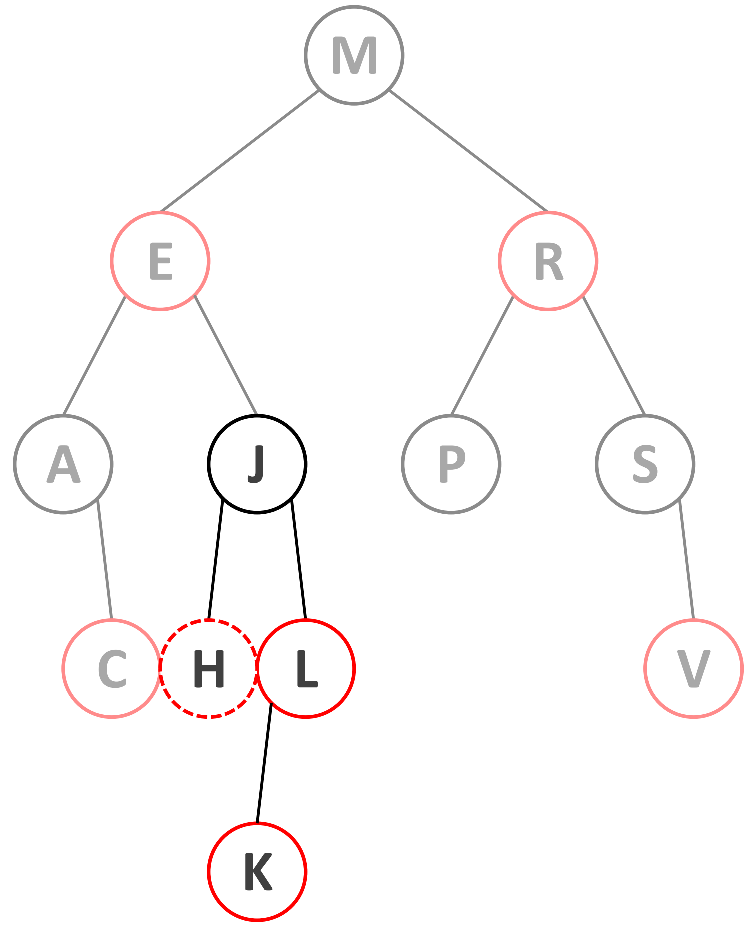
… simplemente la pintamos de negro



¡Listo!

Caso K

Si: tío "rojo"



El tío del nodo insertado es **rojo**

Caso K

Si: tío “rojo”

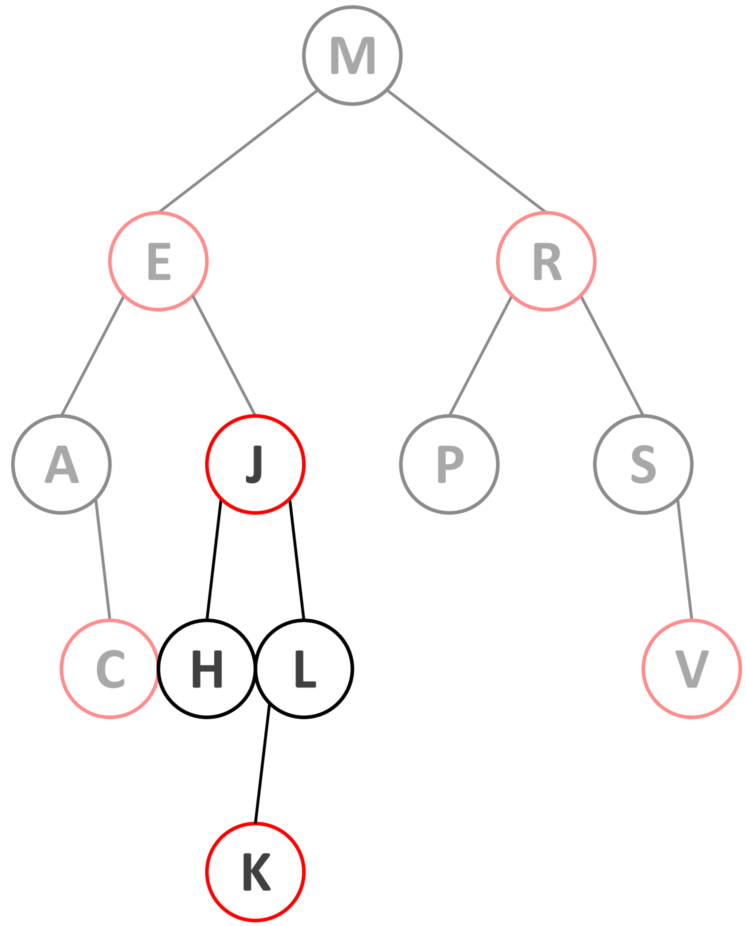
padre k <- negro

tío k <- negro

abuelo k <- rojo

// Volver a analizar para

// abuelo de k



Caso K

Si: tío “rojo” // ahora de j

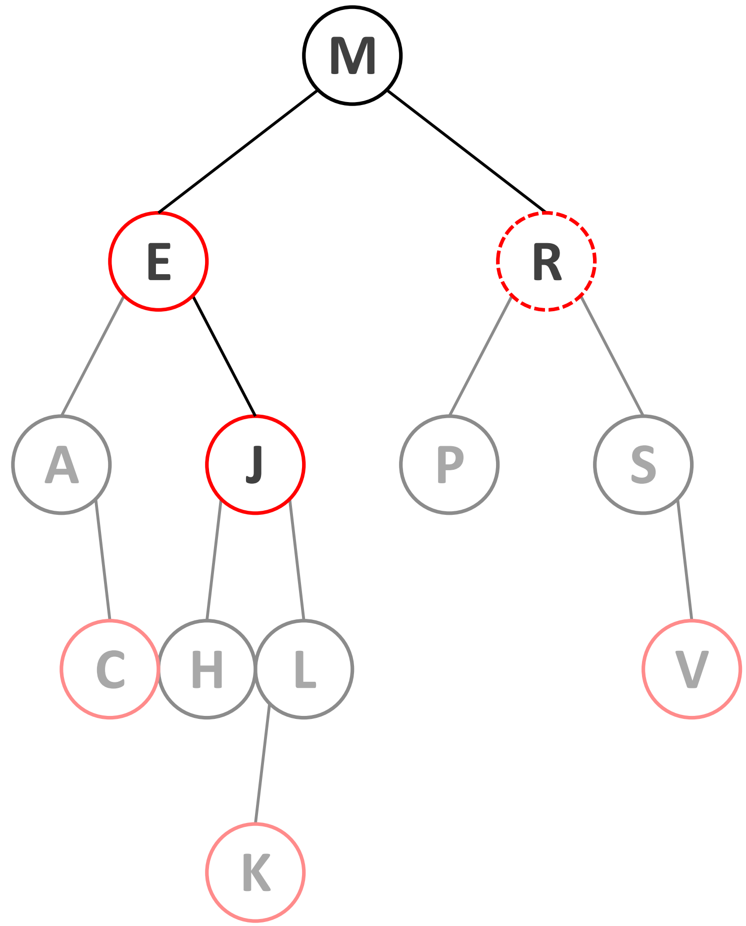
padre j <- negro

tío j <- negro

abuelo j <- rojo

// Volver a analizar para

// abuelo de j



Caso K

Si: tío “rojo” // ahora de m

padre j <- negro

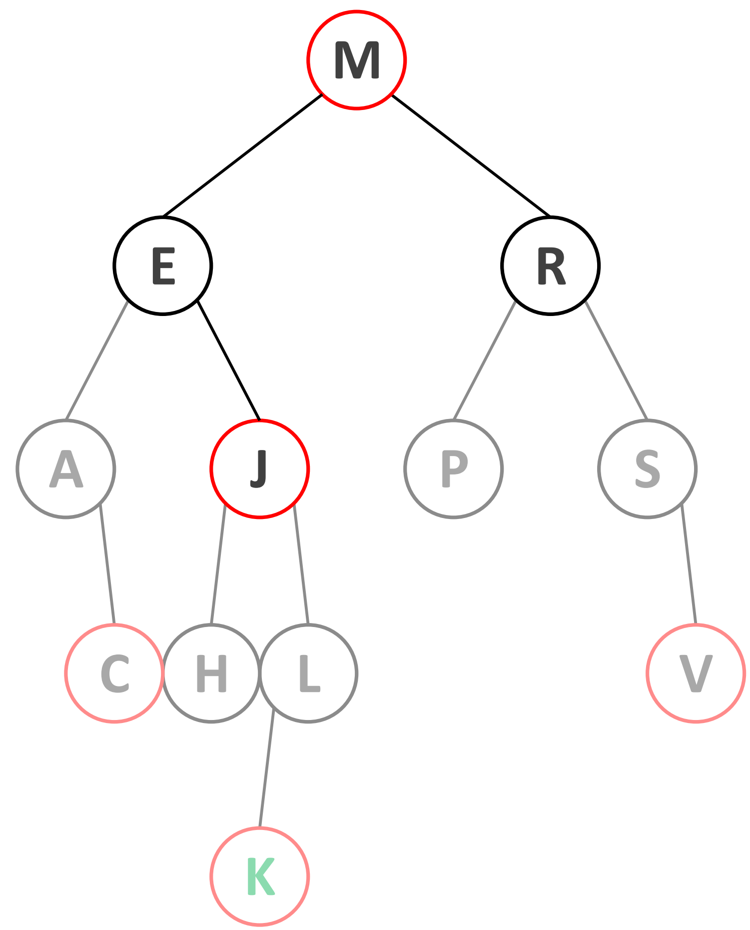
tío j <- negro

abuelo j <- rojo

// Volver a analizar para

// abuelo de j

Raiz <- negro



Caso K

Mientras padre actual es rojo

Si: tío actual es "rojo"

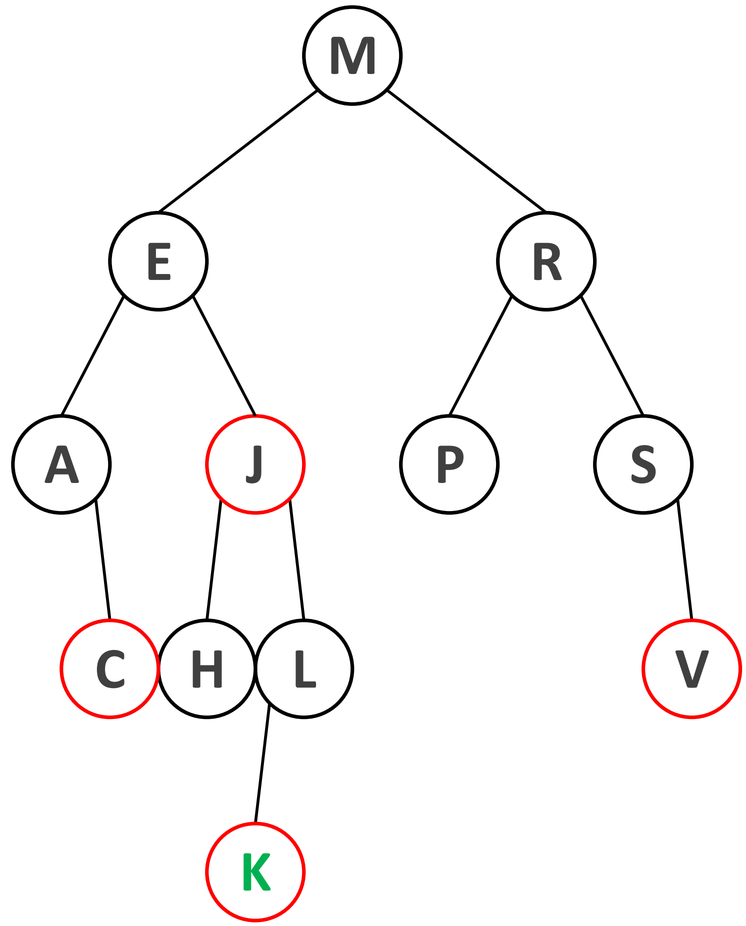
padre actual <- negro

tío actual <- negro

abuelo actual <- rojo

actual <- abuelo actual

Raiz <- negro



Inserción en árboles rojo-negros

Los nodos siempre se insertan rojos

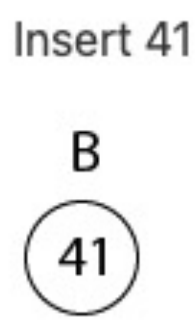
Si su padre es rojo, hay dos casos según el color del tío:

- Si el tío es negro, tenemos el aumento de grado en el nodo del 2-4
  - Se soluciona con rotaciones y cambios de color. No genera más conflictos.
- Si el tío es rojo, tenemos el caso en que el nodo del 2-4 rebalsa
  - Se soluciona cambiando colores. Puede generar el mismo caso hacia arriba.

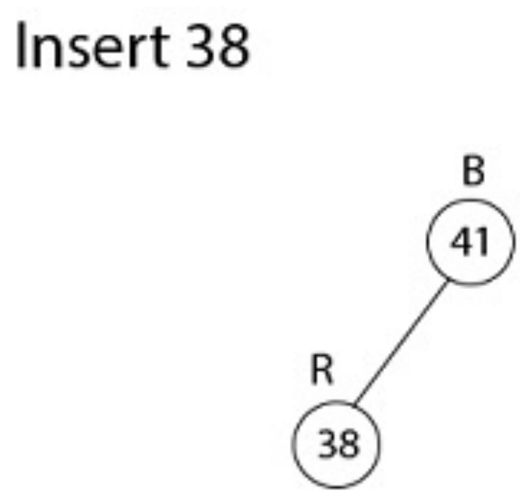
Inserción en árboles rojo-negros

```
Mientras padre actual es rojo
        Si: tío actual es "rojo"                                  // caso K
                padre actual <- negro
                tío actual <- negro
                abuelo actual <- rojo
                actual <- abuelo actual
        Si no,  si actual es hijo "interior"                      //caso U
                        rotacion padre actual, actual
                        actual <- padre actual
                padre actual <- negro                             // caso Z
                abuelo actual <- rojo
                rotacion abuelo actual, actual
Raiz <- negro
```

Ejercicio: Insert 41,38,31,12,19,8



Ejercicio: Insert 41,38,31,12,19,8

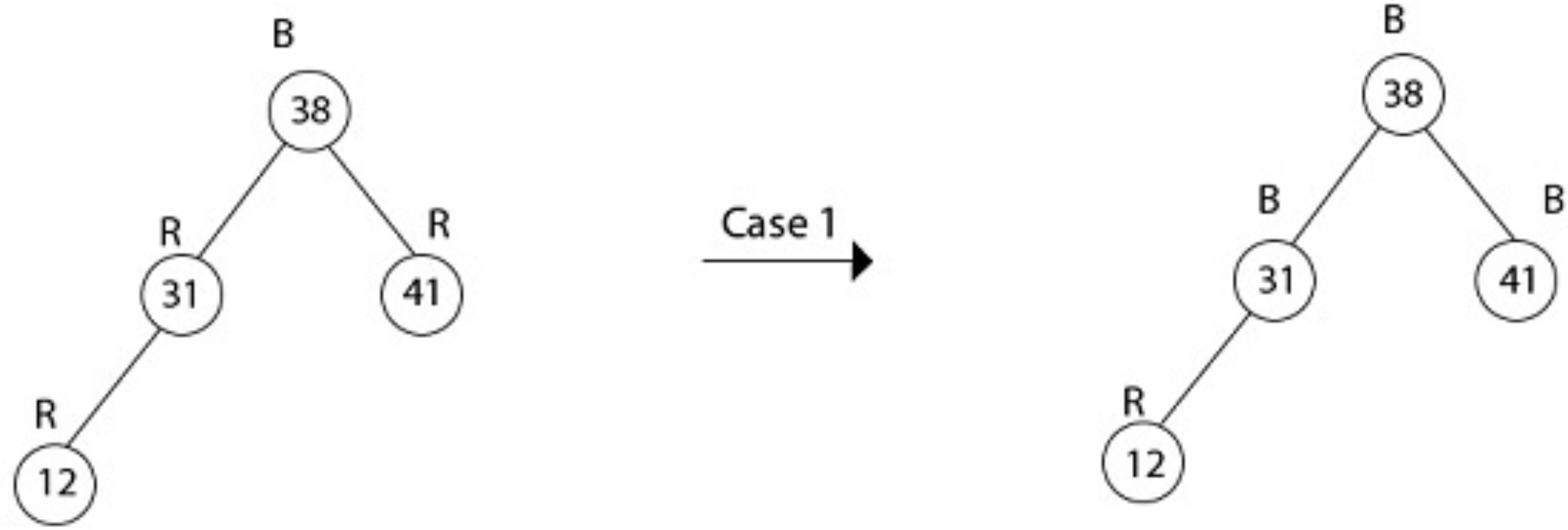


Ejercicio: Insert 41,38,31,12,19,8

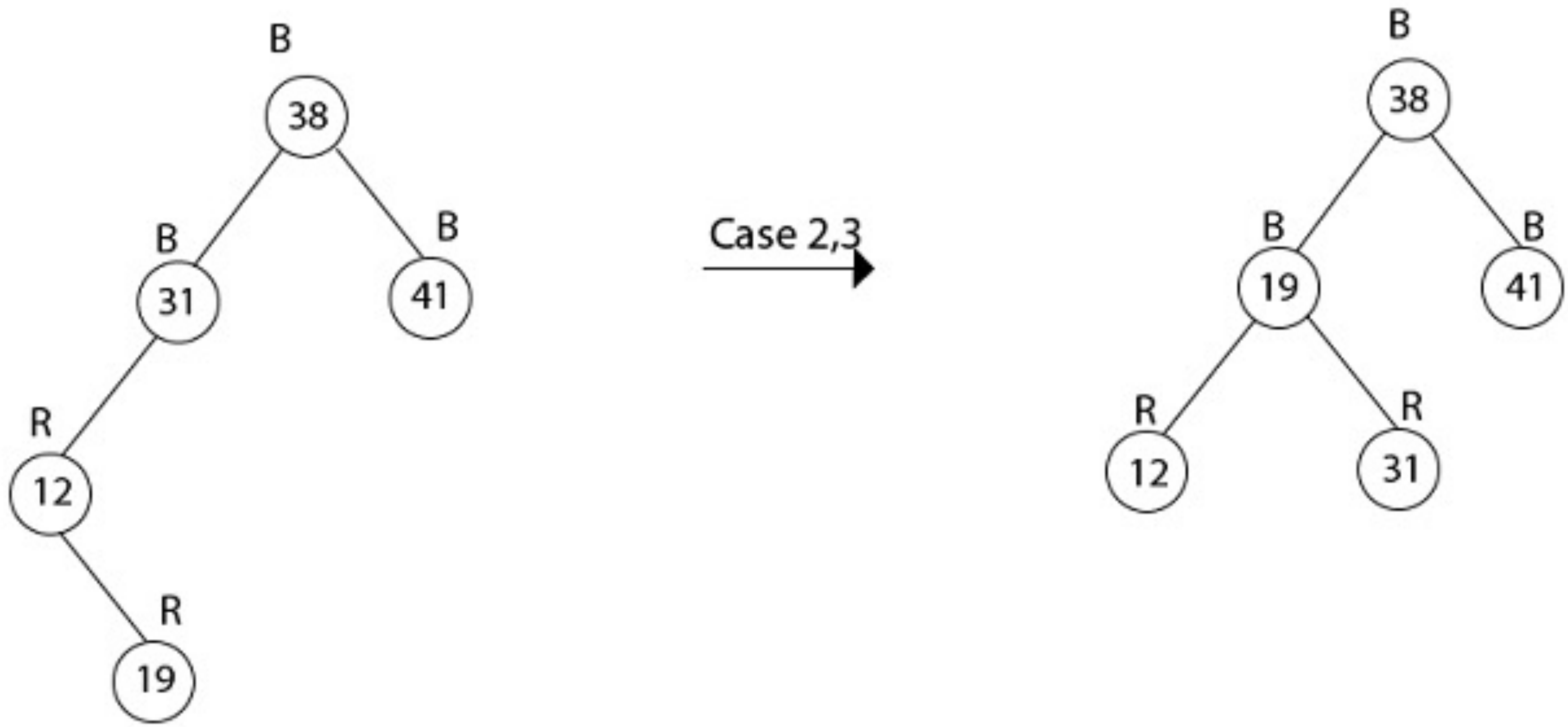
Insert 31

Ejercicio: Insert 41,38,31,12,19,8

Insert 12



Ejercicio: Insert 41,38,31,12,19,8



Ejercicio: Insert 41,38,31,12,19,8

Ejercicio propuesto

Demuestra que la altura de un árbol rojo-negro con n nodos es $O(\log n)$