Algoritmos y Estructuras de Datos I

Universidad Católica del Uruguay

Resumen de clase

Jueves 27 de mayo de 2021

Algoritmos de árboles binarios basados en su recorrido

Universidad Católica del Uruguay

En TNodoArbolBinario elMetodo (): de algún tipo

COM

*HACER ALGO EN PREORDEN (inicializar **variables locales**, o comprobar algo y devolver)*

Si hijoIzq != nulo entonces
resultadoIzquierdo = hijoIzq.elMetodo()
fin si

HACER ALGO EN INORDEN, por ejemplo decidir si se llama al hijo derecho o no

Si hijoDer != nulo entonces
resultadoDerecho = hijoDer.elMetodo()
fin si

*HACER ALGO EN POSTORDEN (por ejemplo **componer ambos resultados**, devolver)*

FIN

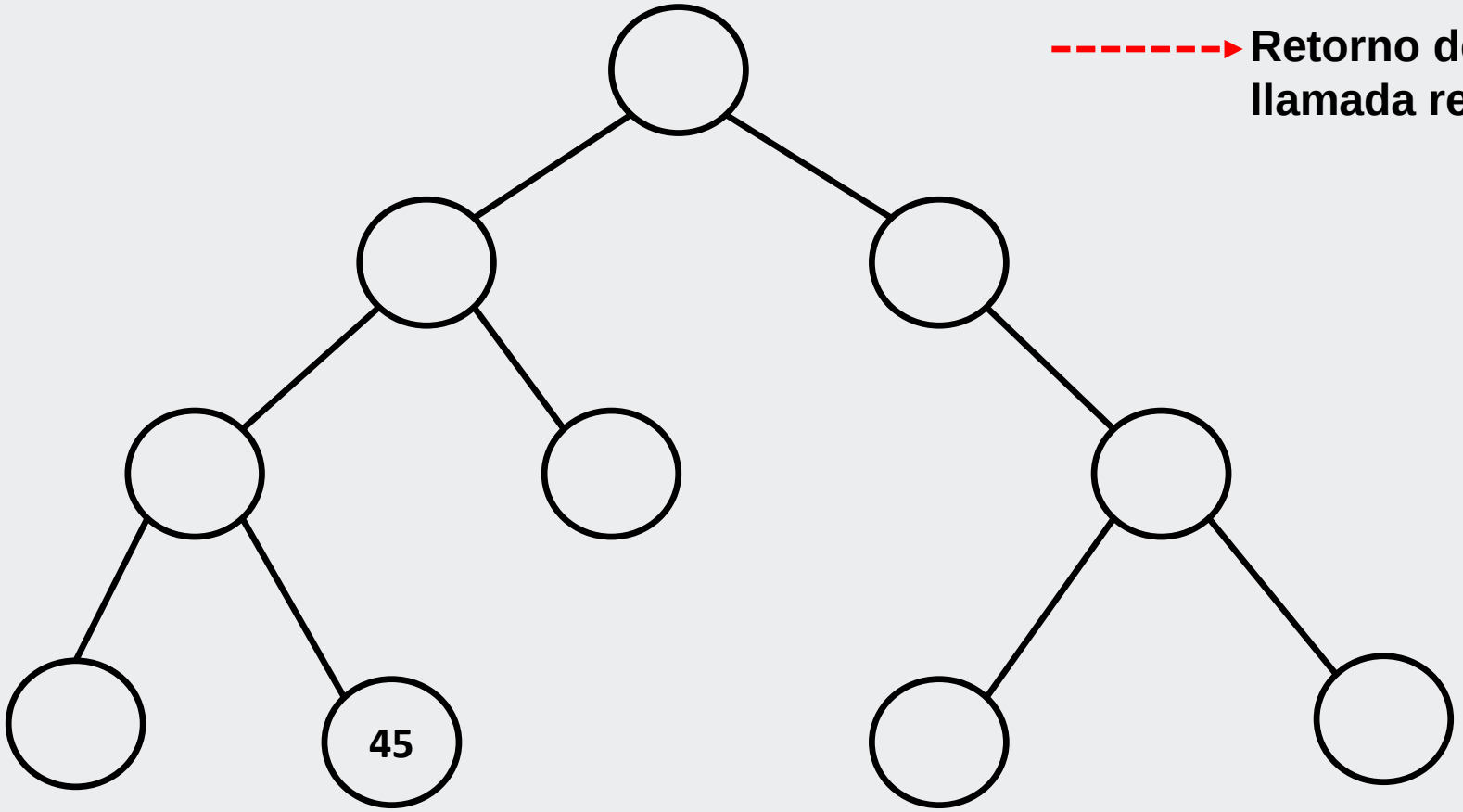
EJEMPLO: BUSQUEDA LINEAL (el árbol binario no es de búsqueda)

Universidad Católica del Uruguay

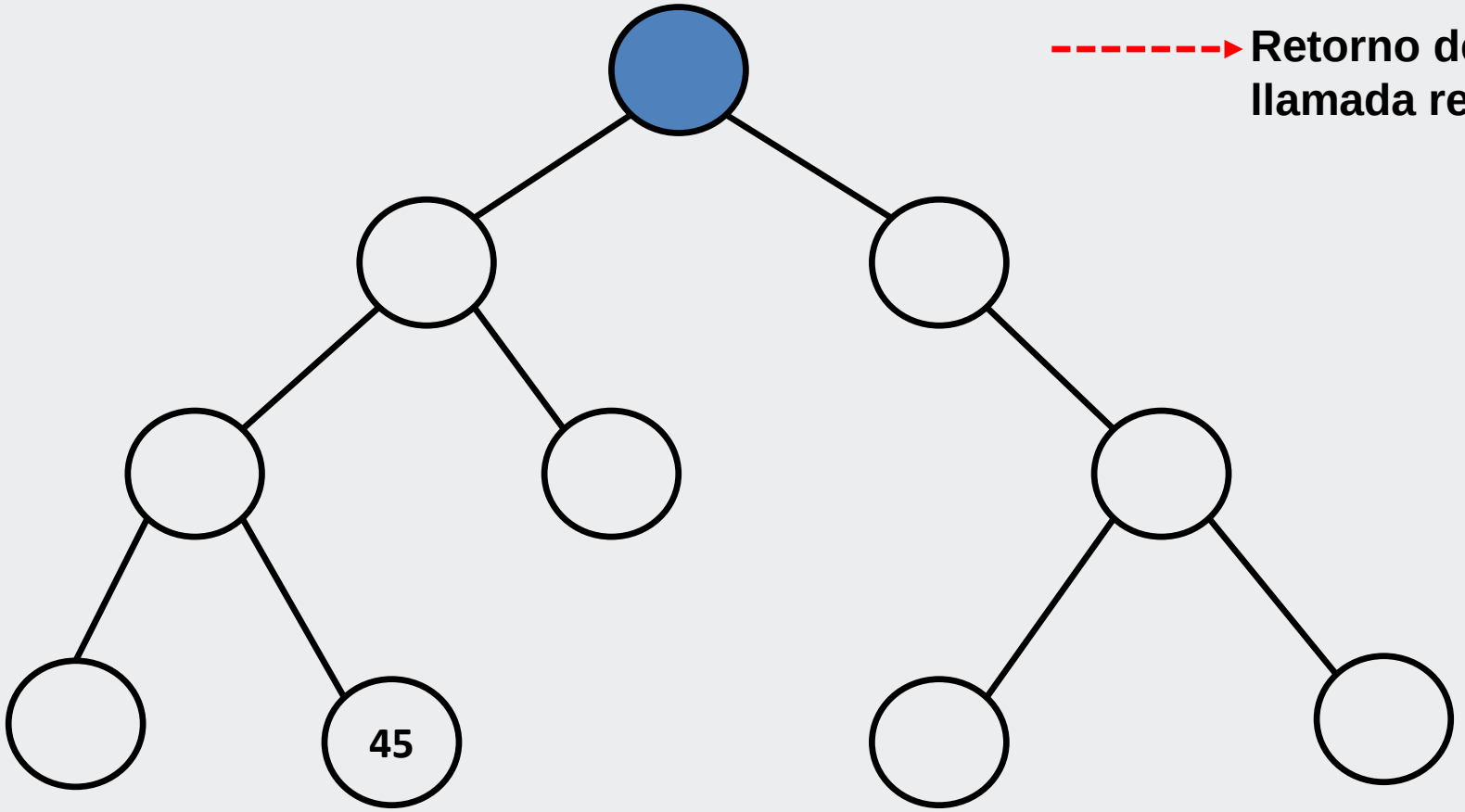
```
tipoElementoAB.encuentraLinealUno( Comparable unaEti): de tipo boolean
//devuelve verdadero si hay un nodo que tenga esa etiqueta, y falso en caso contrario.
Comienzo
 Si this.etiqueta = unaEti entonces
     Devolver Verdadero
 Fin si
 estáEnIzq ← Falso; estáEnDer ← Falso
 Si hijoIzquierdo <> nulo entonces
     estáEnIzq ← hijoIzquierdo.encuentraLinealUno(unaEti)
 Fin si
 Si hijoDerecho <> nulo entonces
     estáEnDer ← hijoDerechoencuentraLinealUno(unaEti)
 Fin si
 Devolver (estáEnIzq OR estáEnDer)
Fin
```

“encuentra lineal uno” con “unaEti” = 45

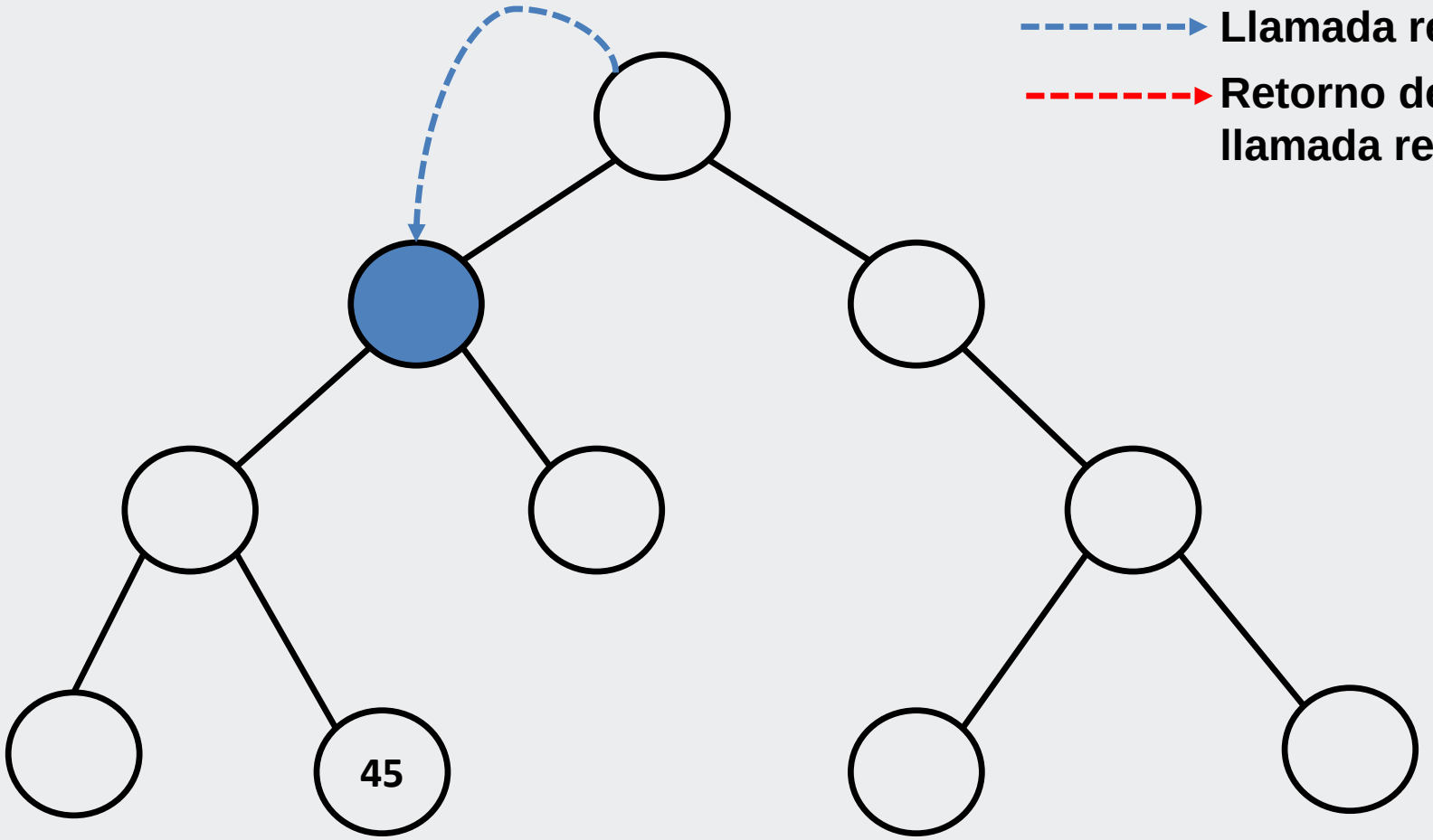
Universidad Católica del Uruguay



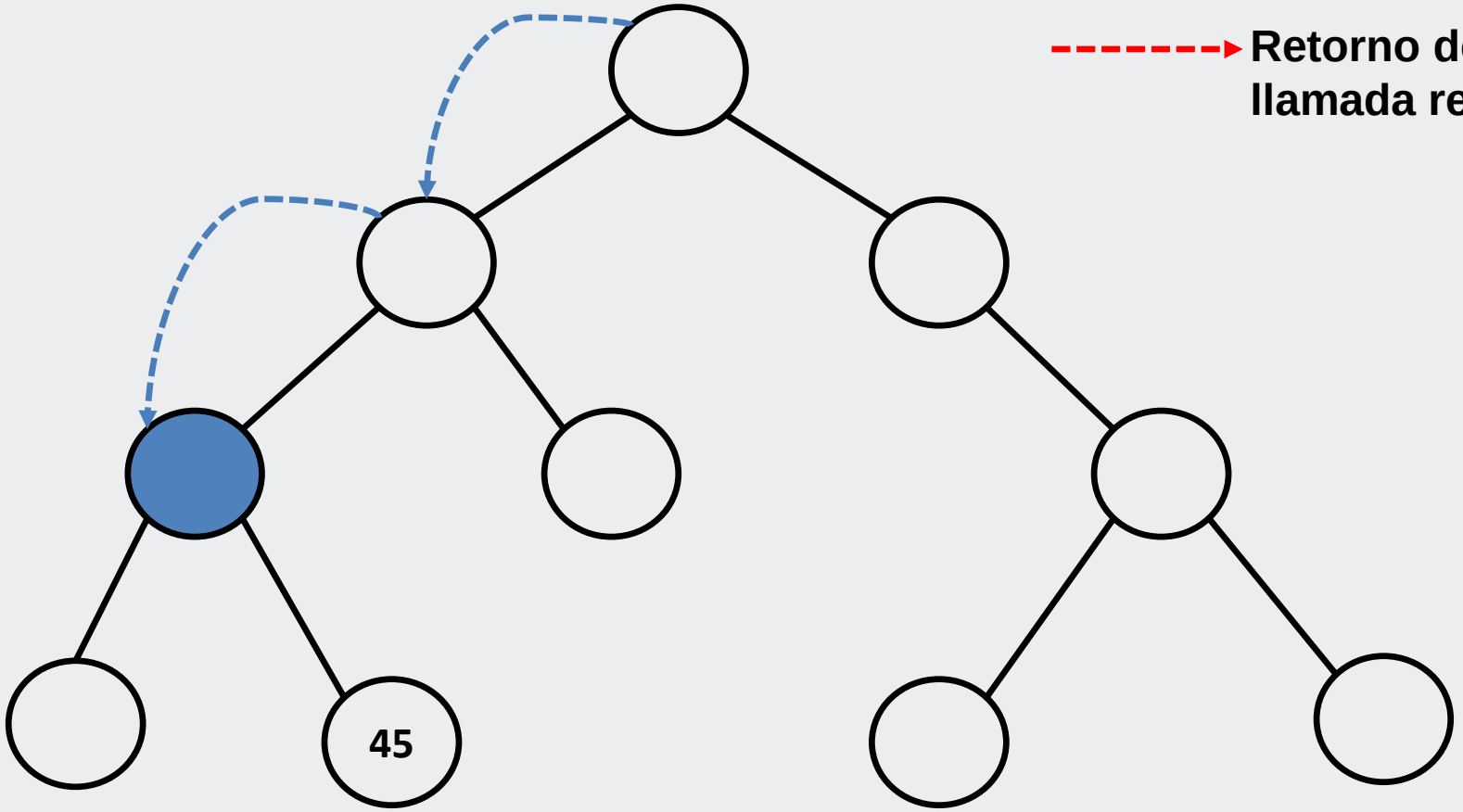
“encuentra lineal uno” con “unaEti” = 45



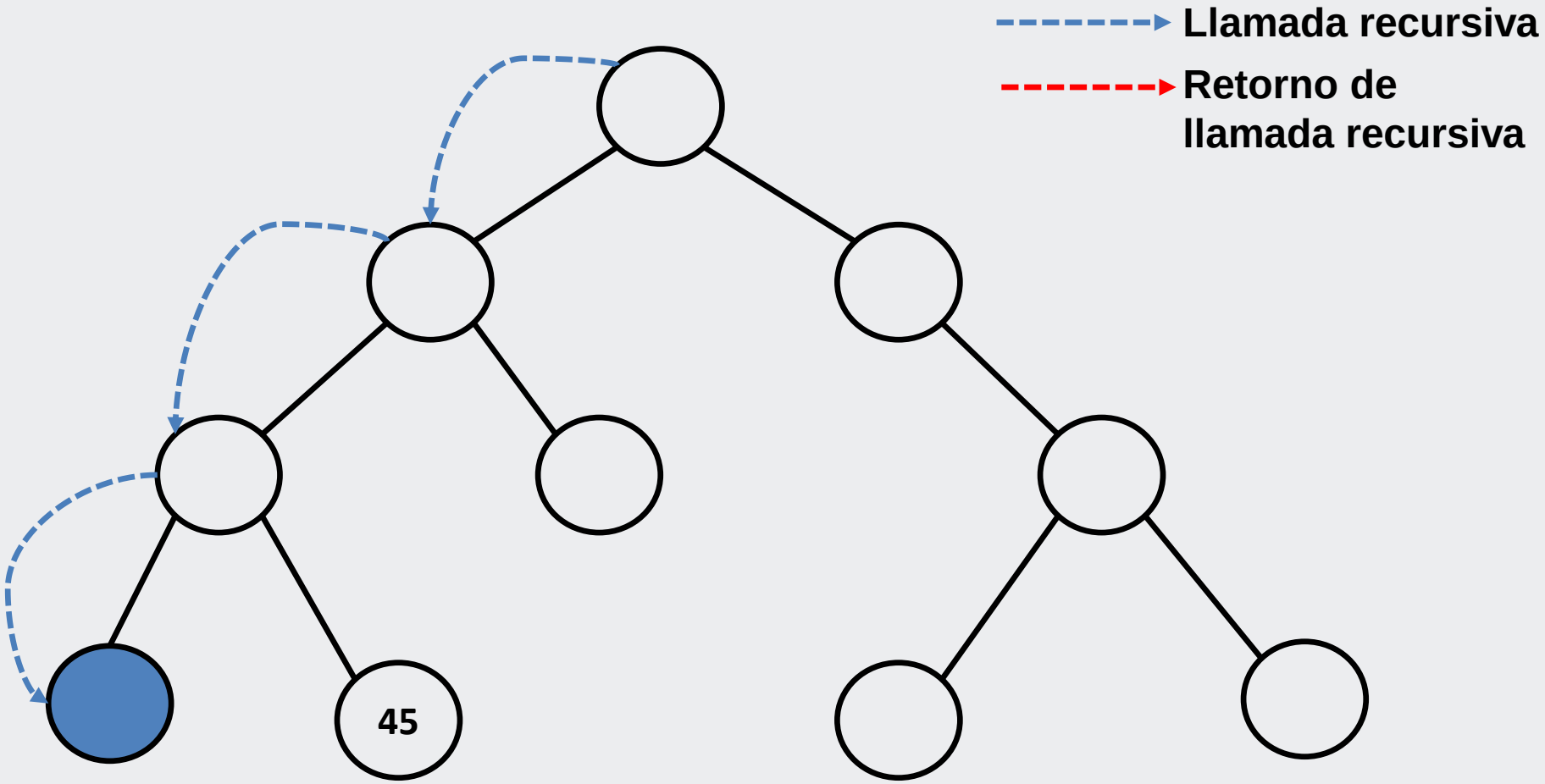
“encuentra lineal uno” con “unaEti” = 45



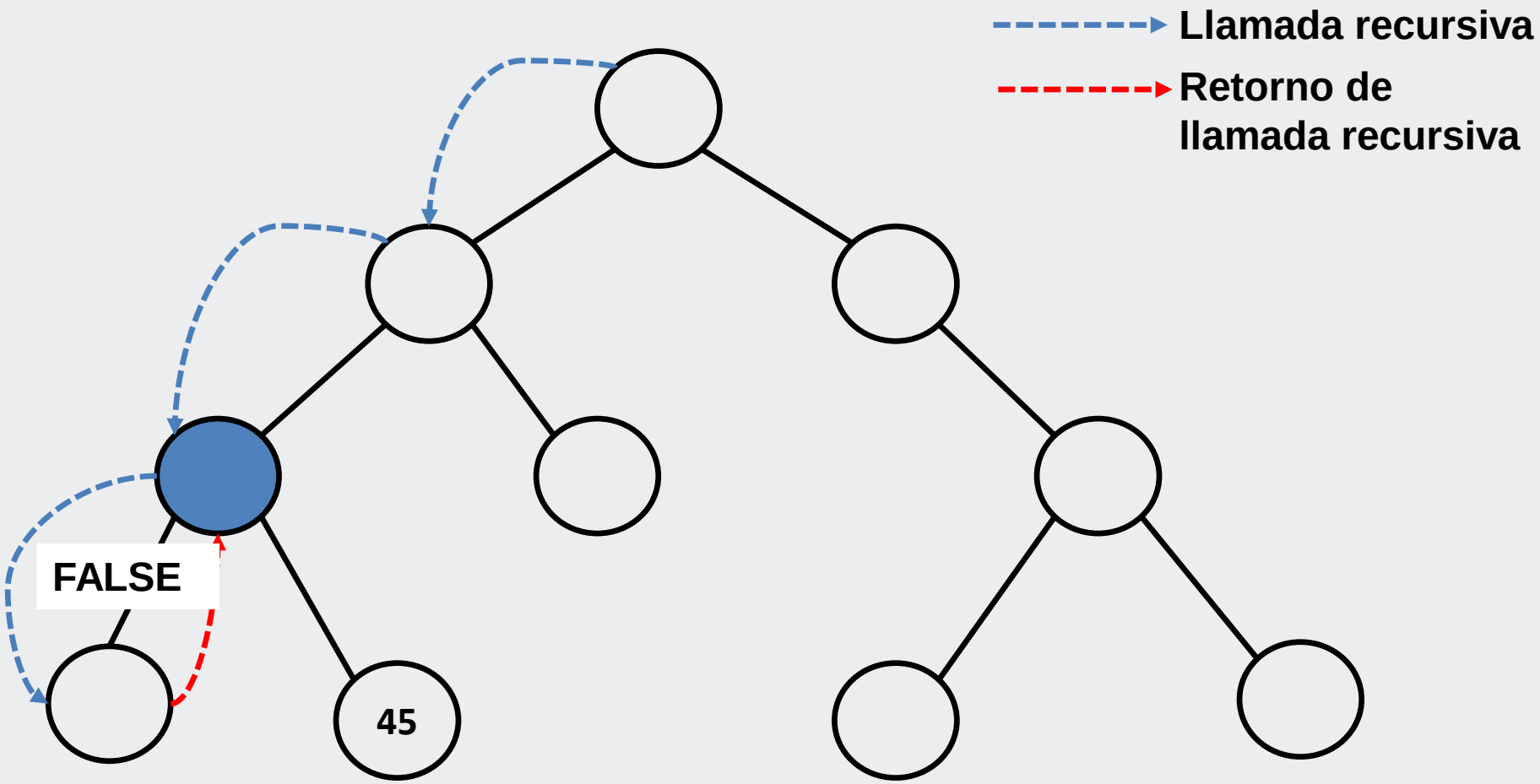
“encuentra lineal uno” con “unaEti” = 45



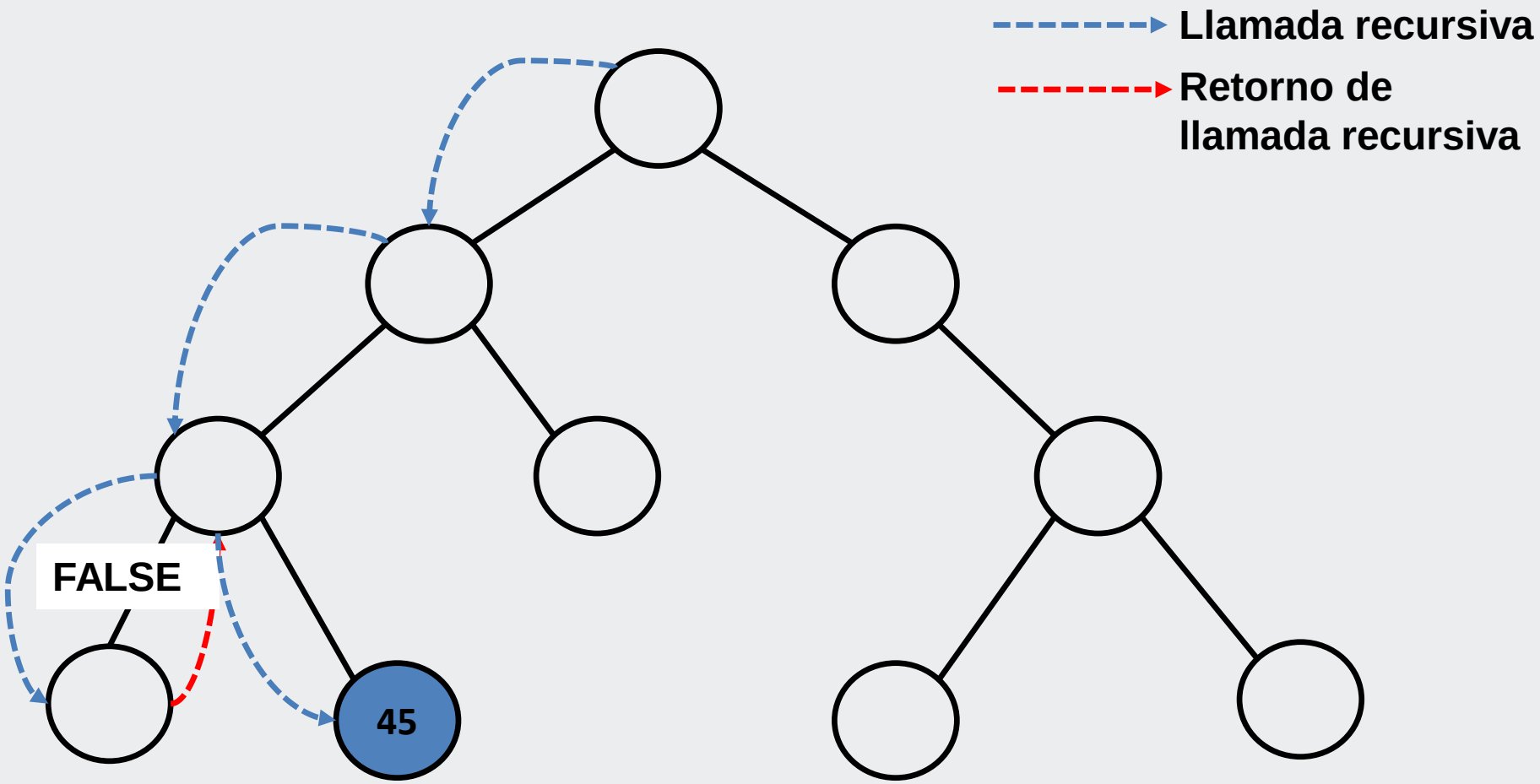
“encuentra lineal uno” con “unaEti” = 45



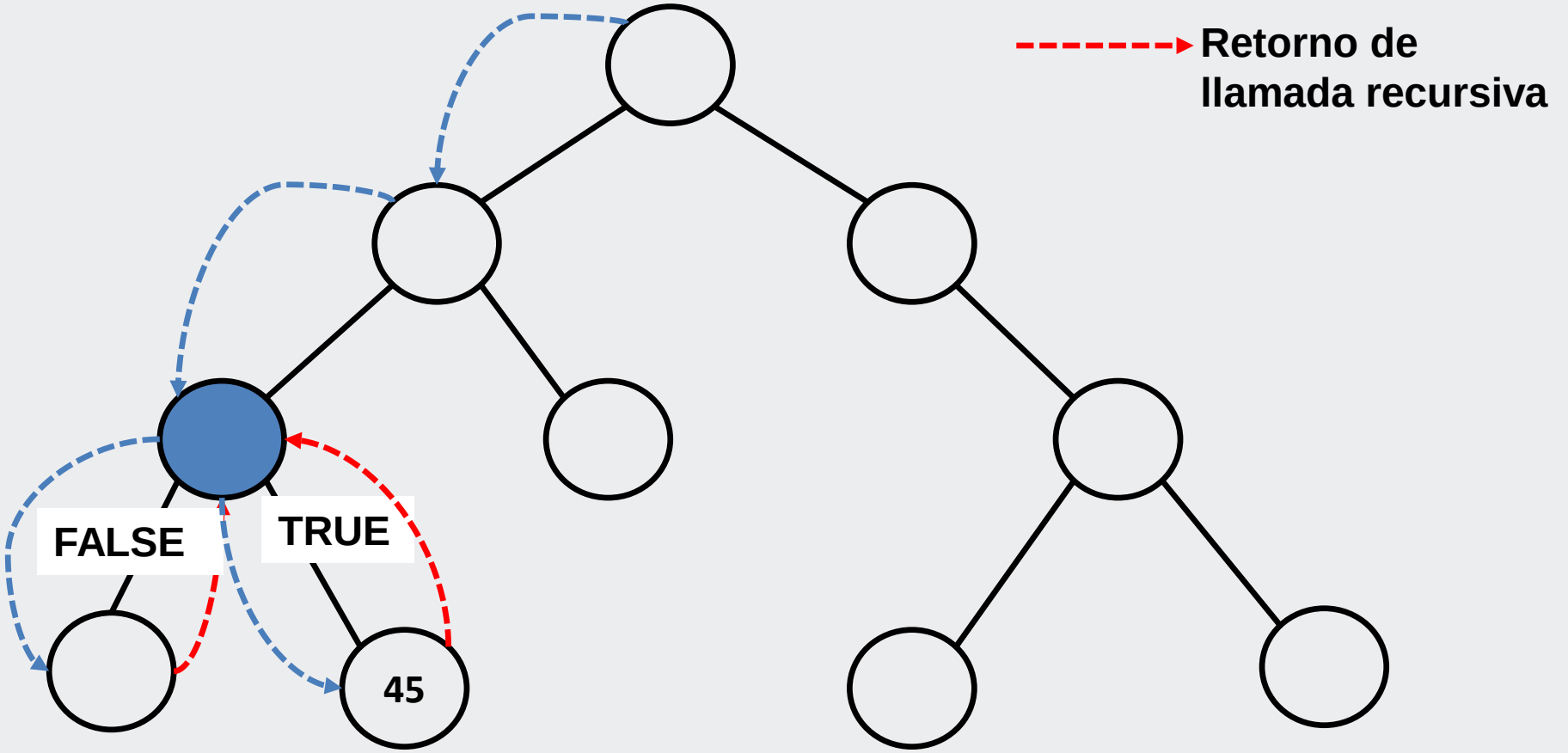
“encuentra lineal uno” con “unaEti” = 45



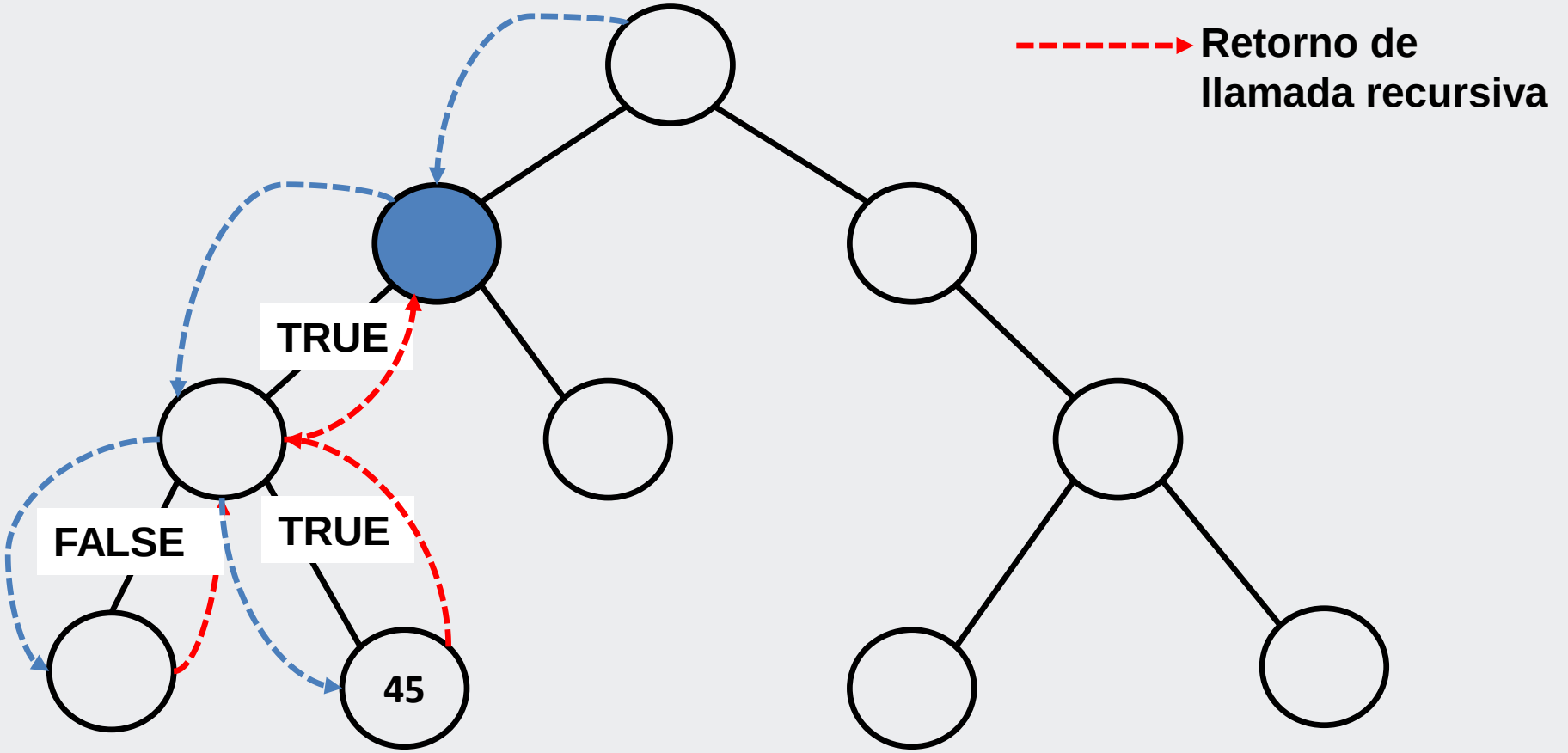
“encuentra lineal uno” con “unaEti” = 45



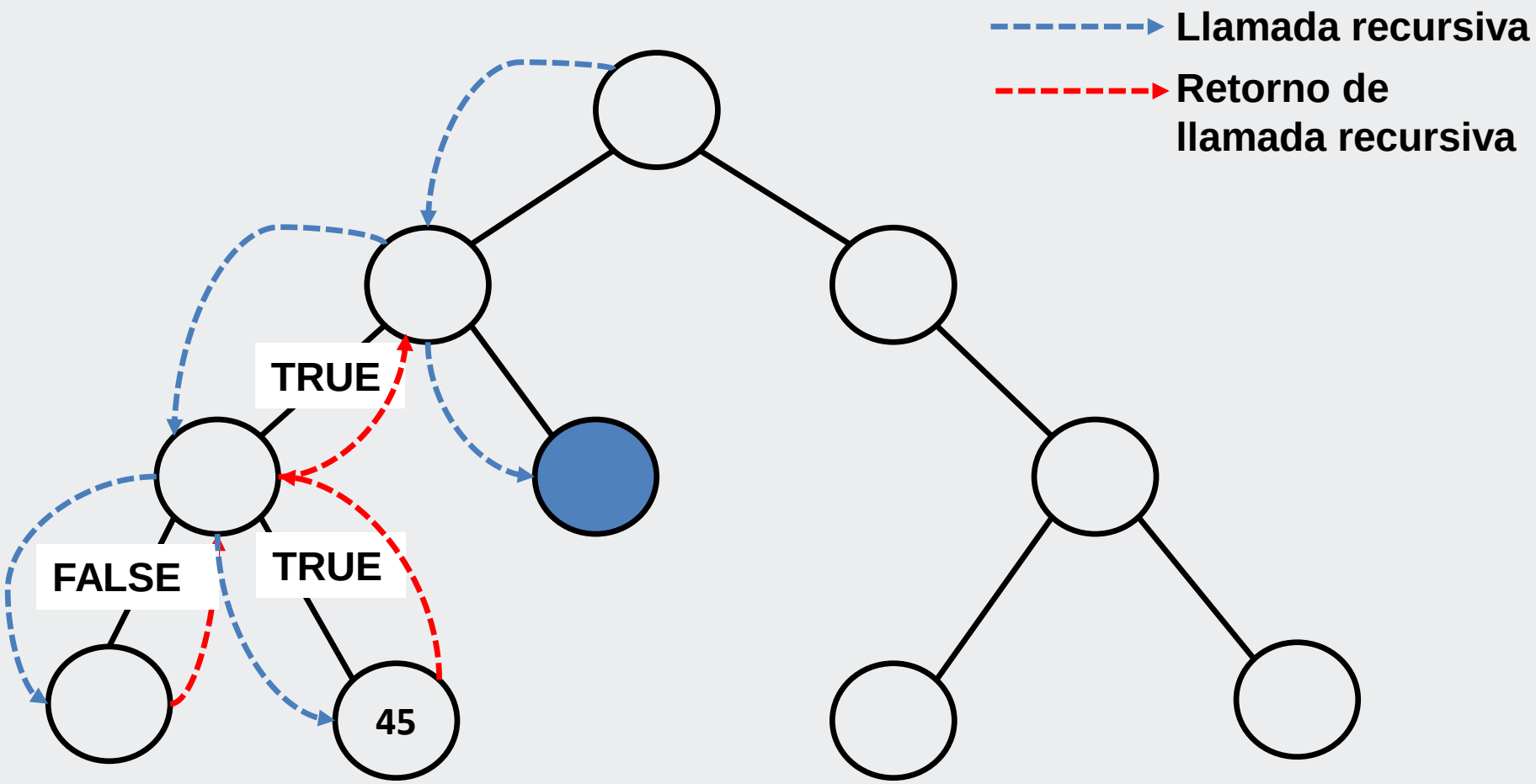
“encuentra lineal uno” con “unaEti” = 45



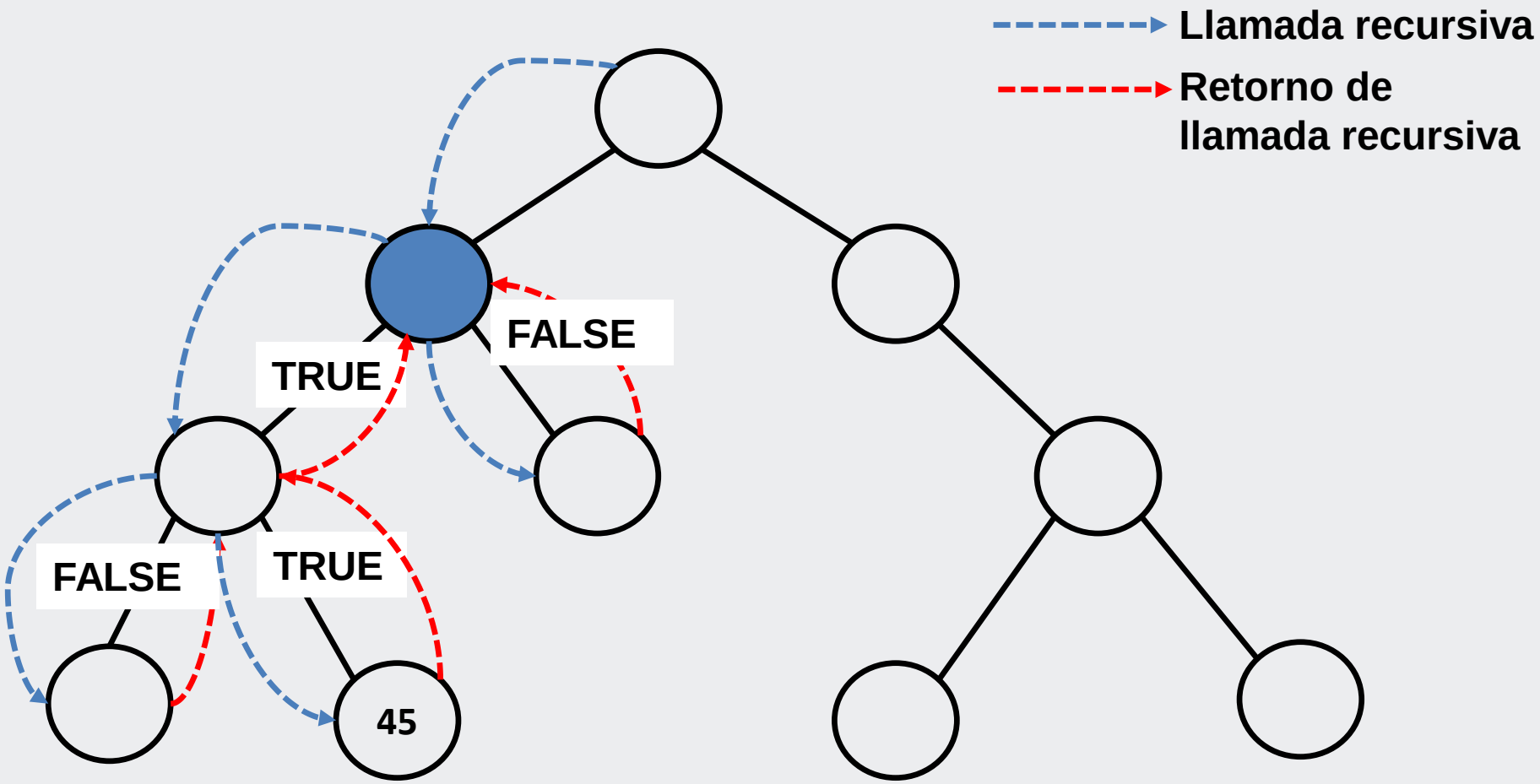
“encuentra lineal uno” con “unaEti” = 45



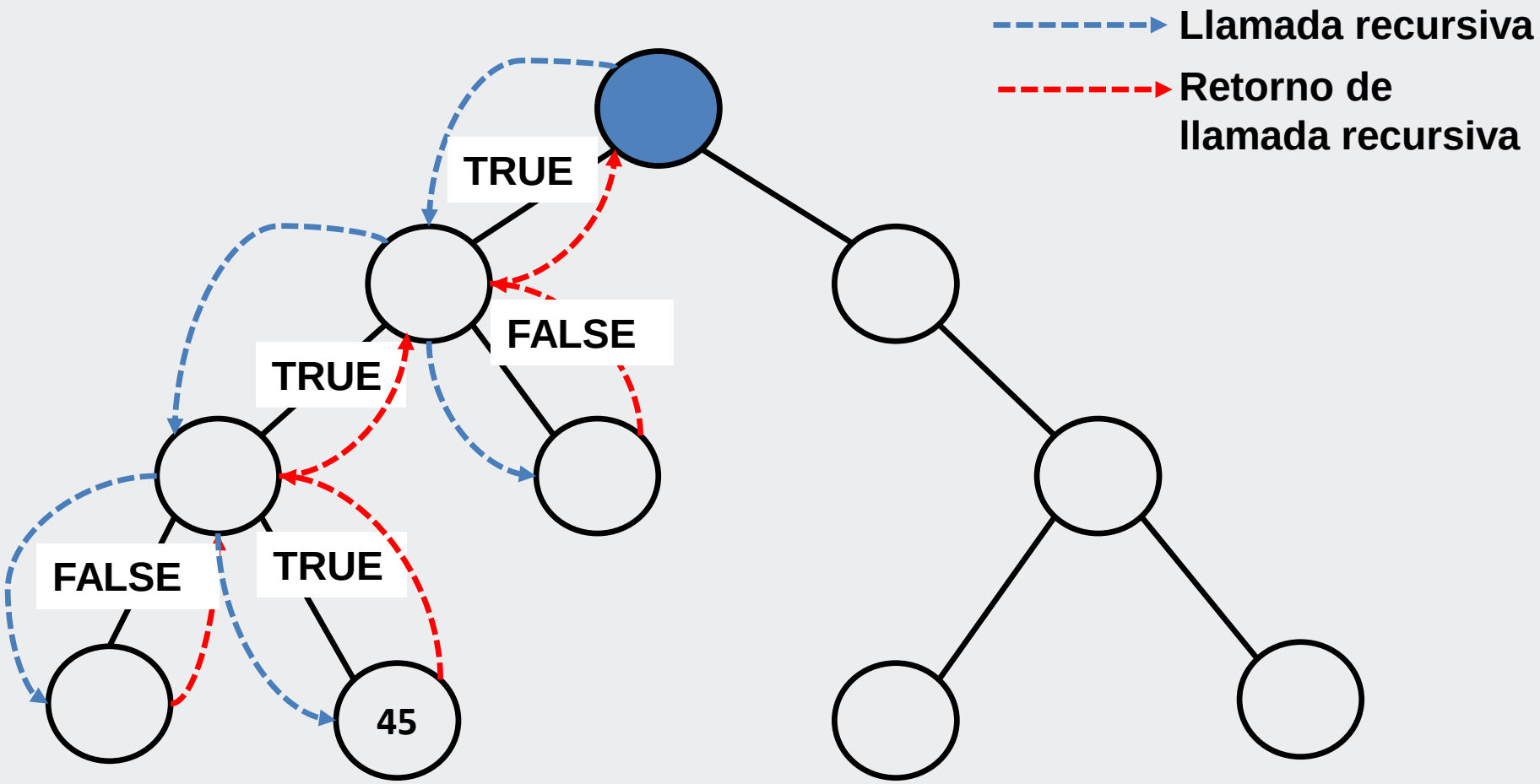
“encuentra lineal uno” con “unaEti” = 45



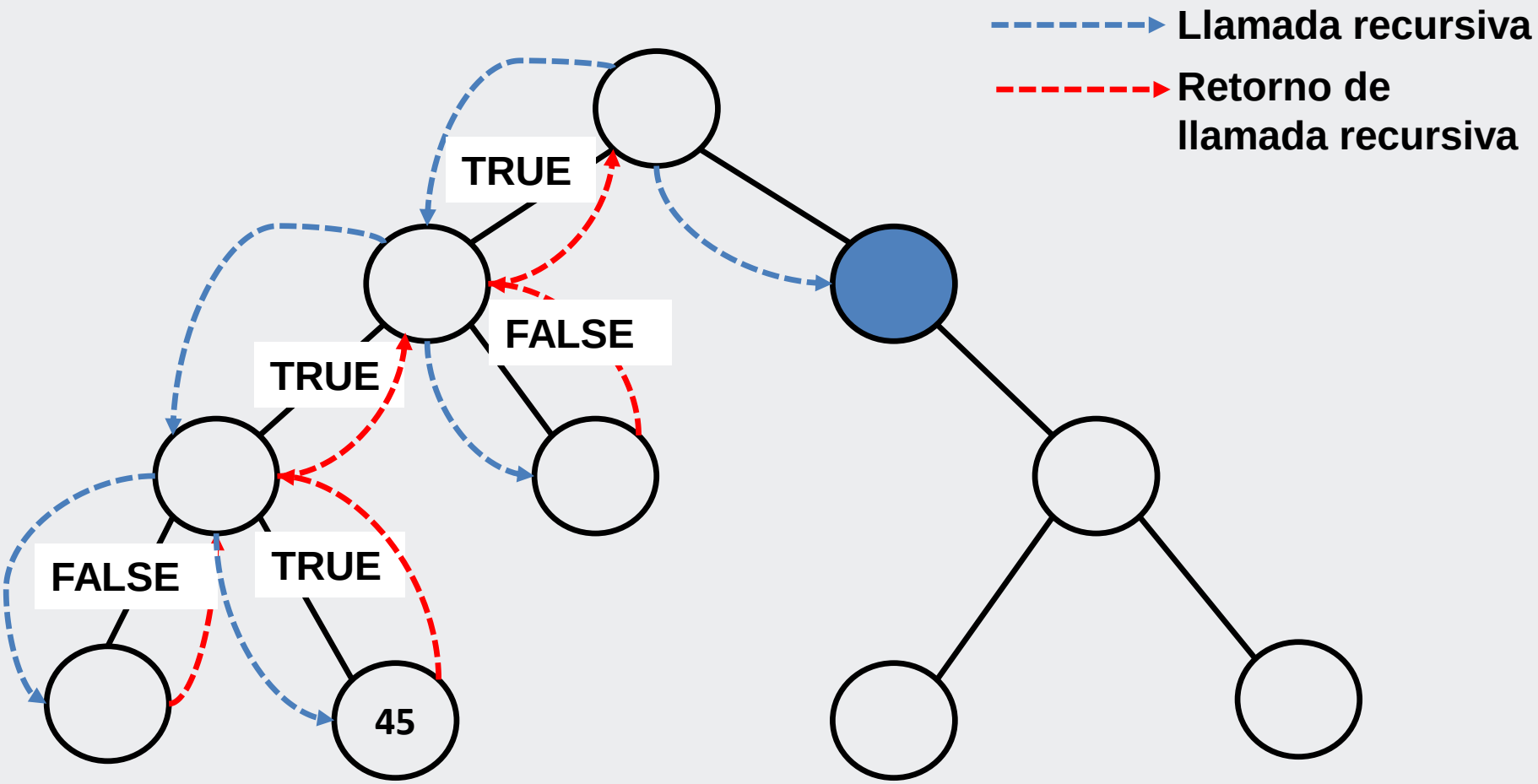
“encuentra lineal uno” con “unaEti” = 45



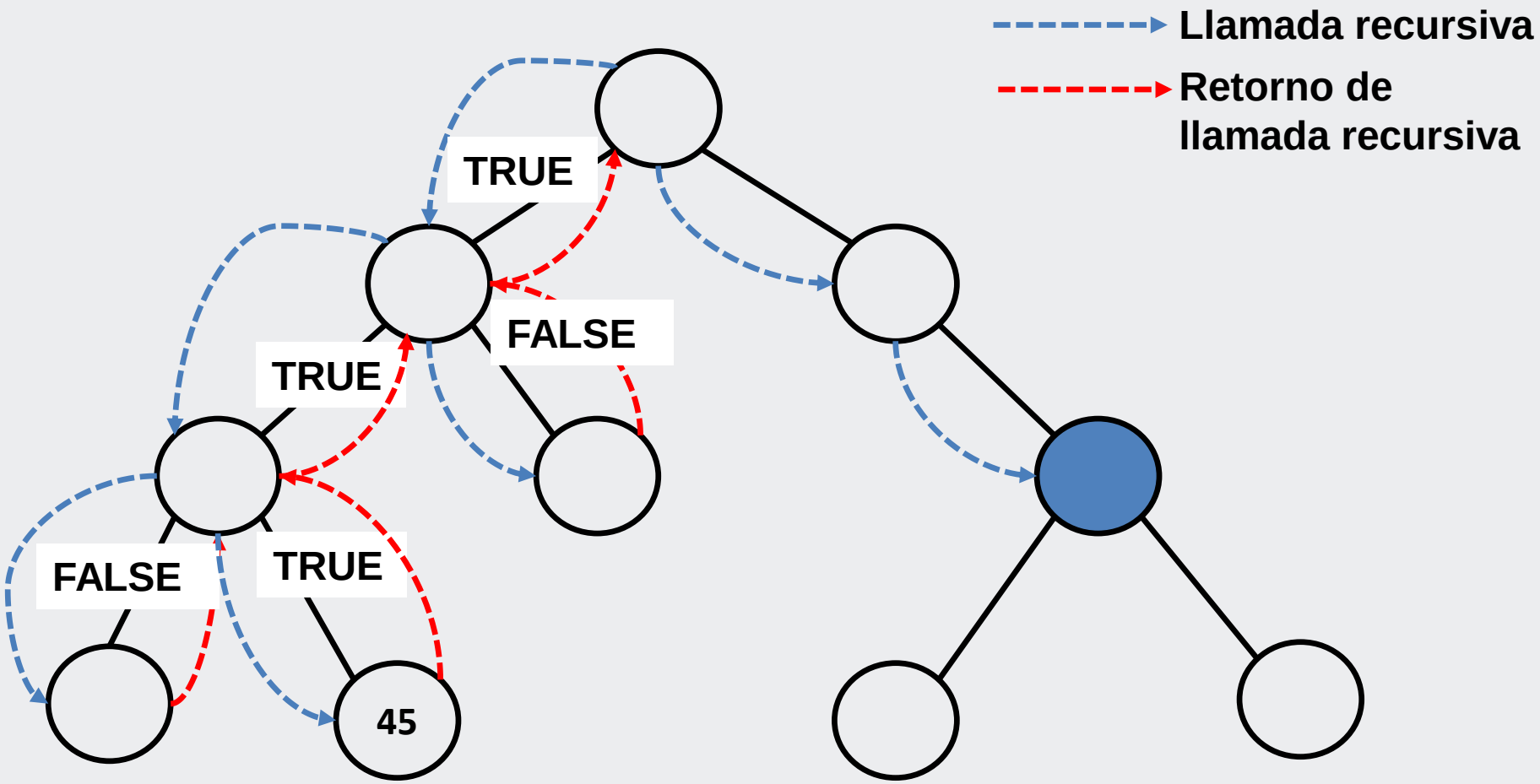
“encuentra lineal uno” con “unaEti” = 45



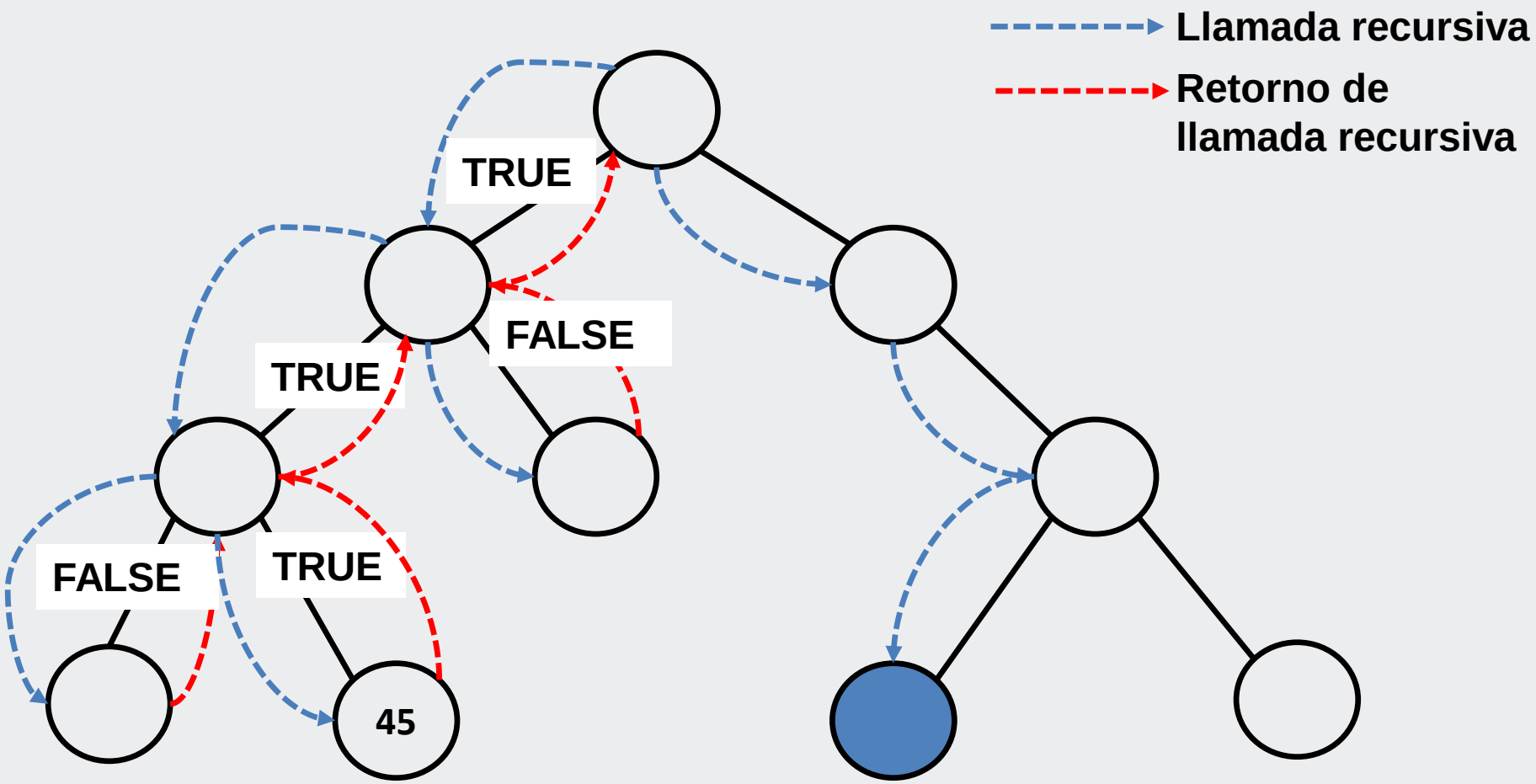
“encuentra lineal uno” con “unaEti” = 45



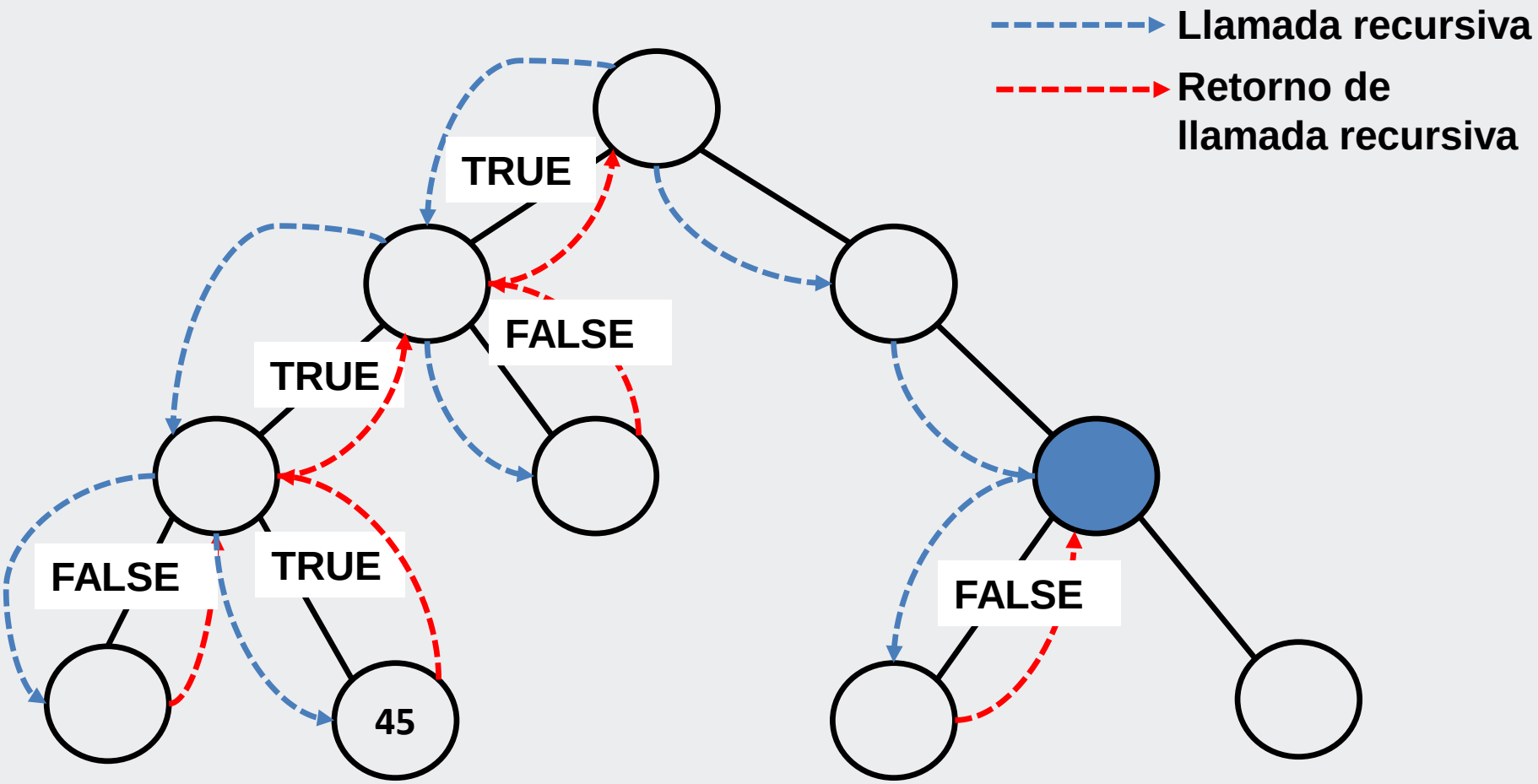
“encuentra lineal uno” con “unaEti” = 45



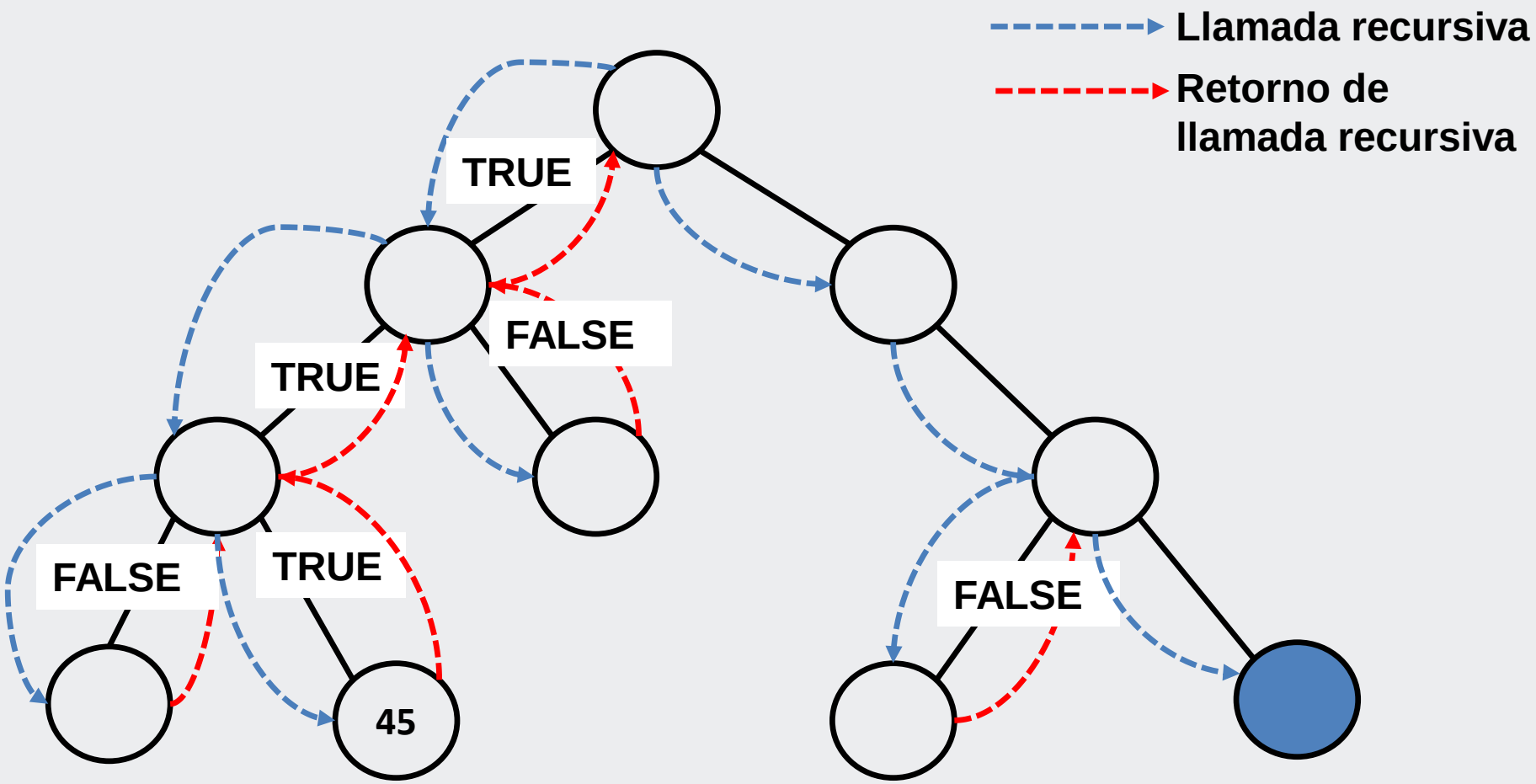
“encuentra lineal uno” con “unaEti” = 45



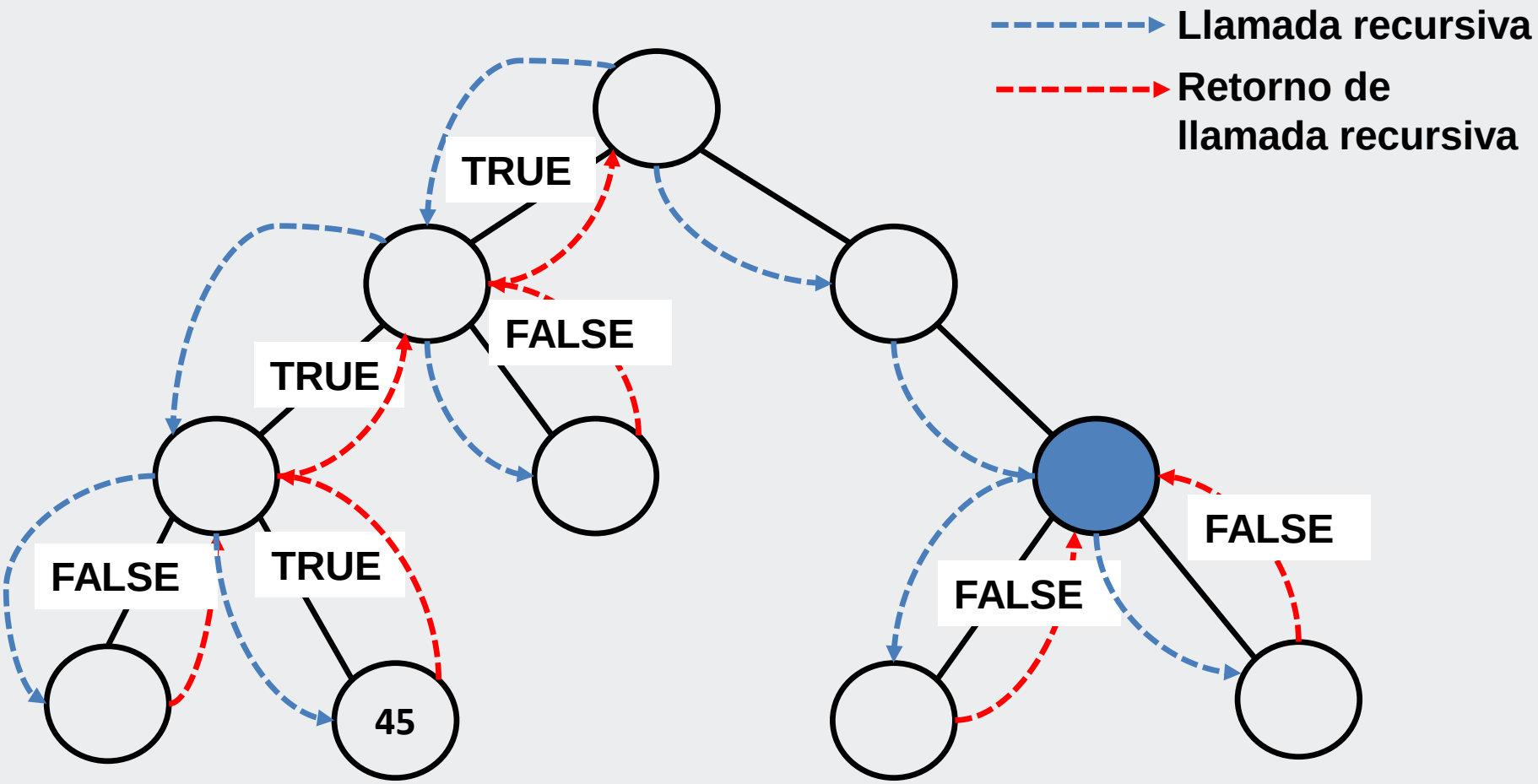
“encuentra lineal uno” con “unaEti” = 45



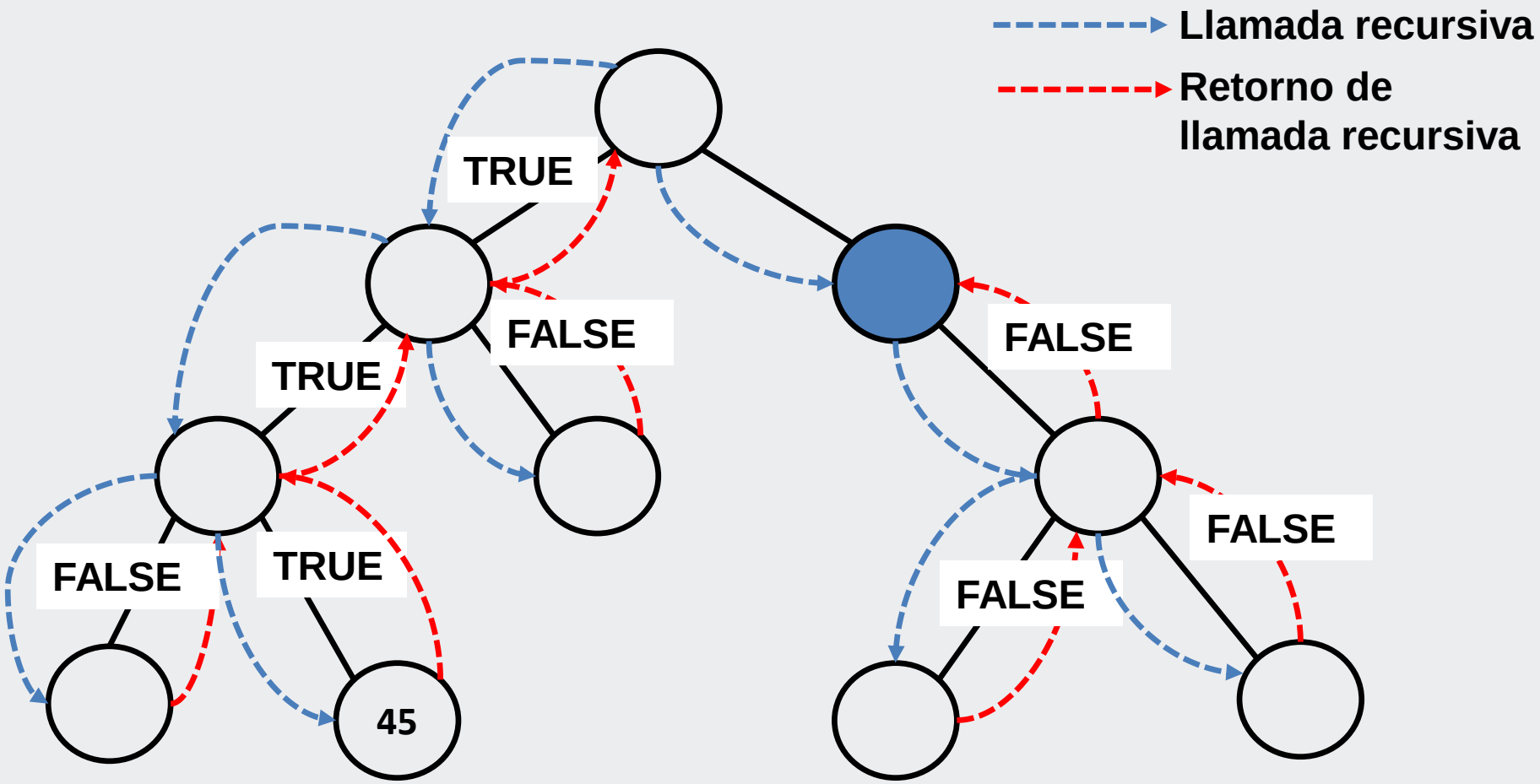
“encuentra lineal uno” con “unaEti” = 45



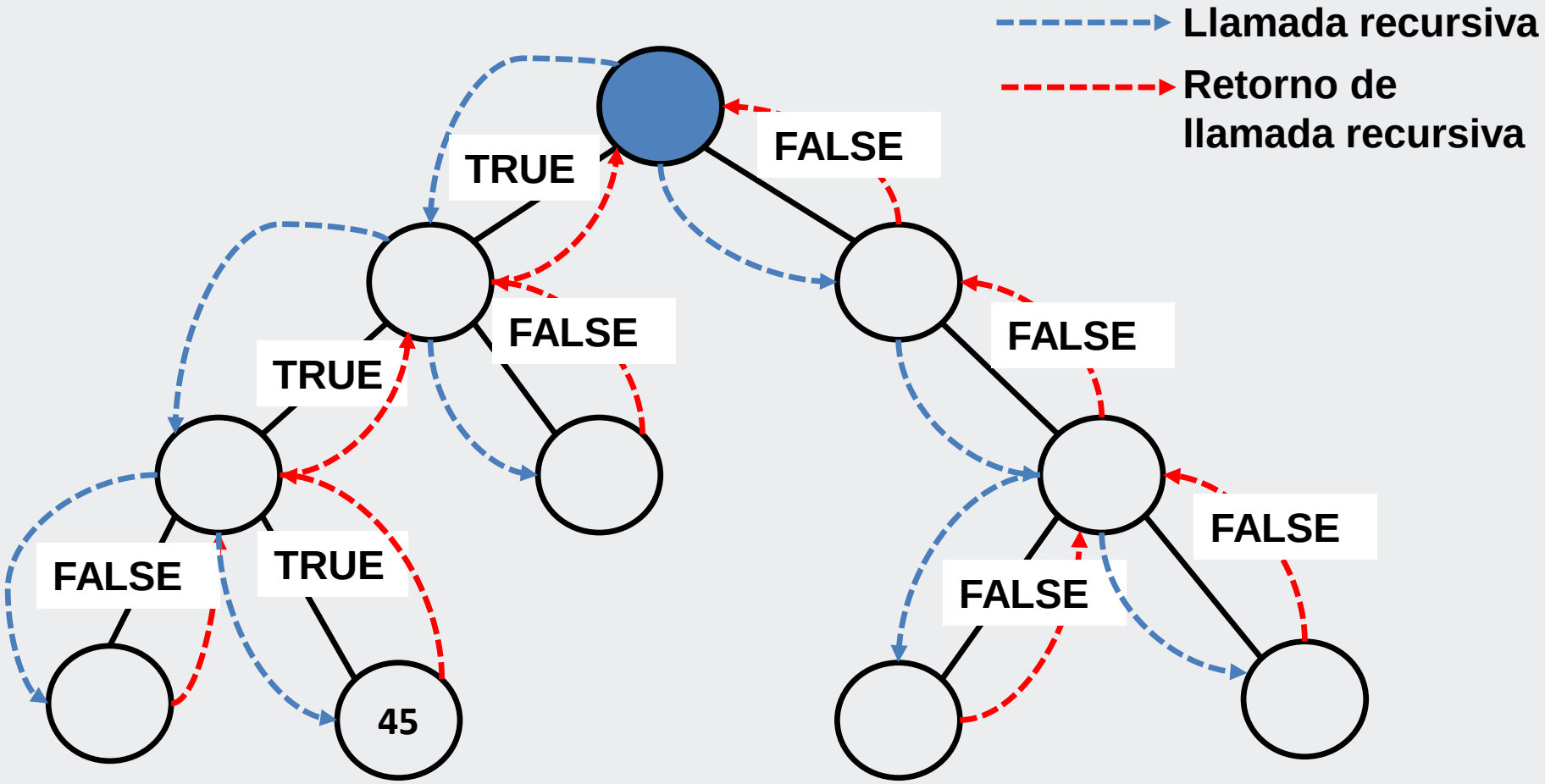
“encuentra lineal uno” con “unaEti” = 45



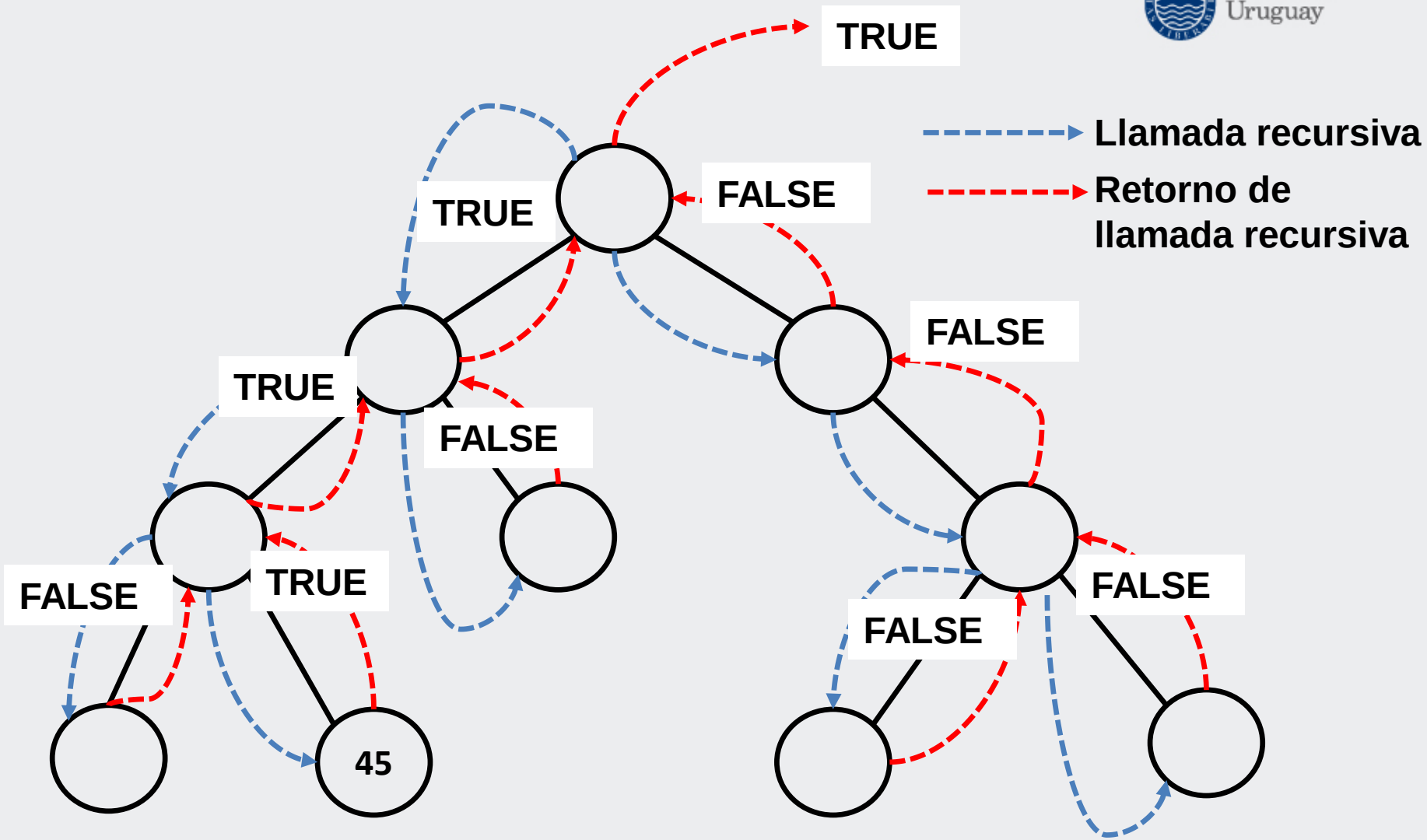
“encuentra lineal uno” con “unaEti” = 45



“encuentra lineal uno” con “unaEti” = 45



“encuentra lineal uno” con “unaEti” = 45

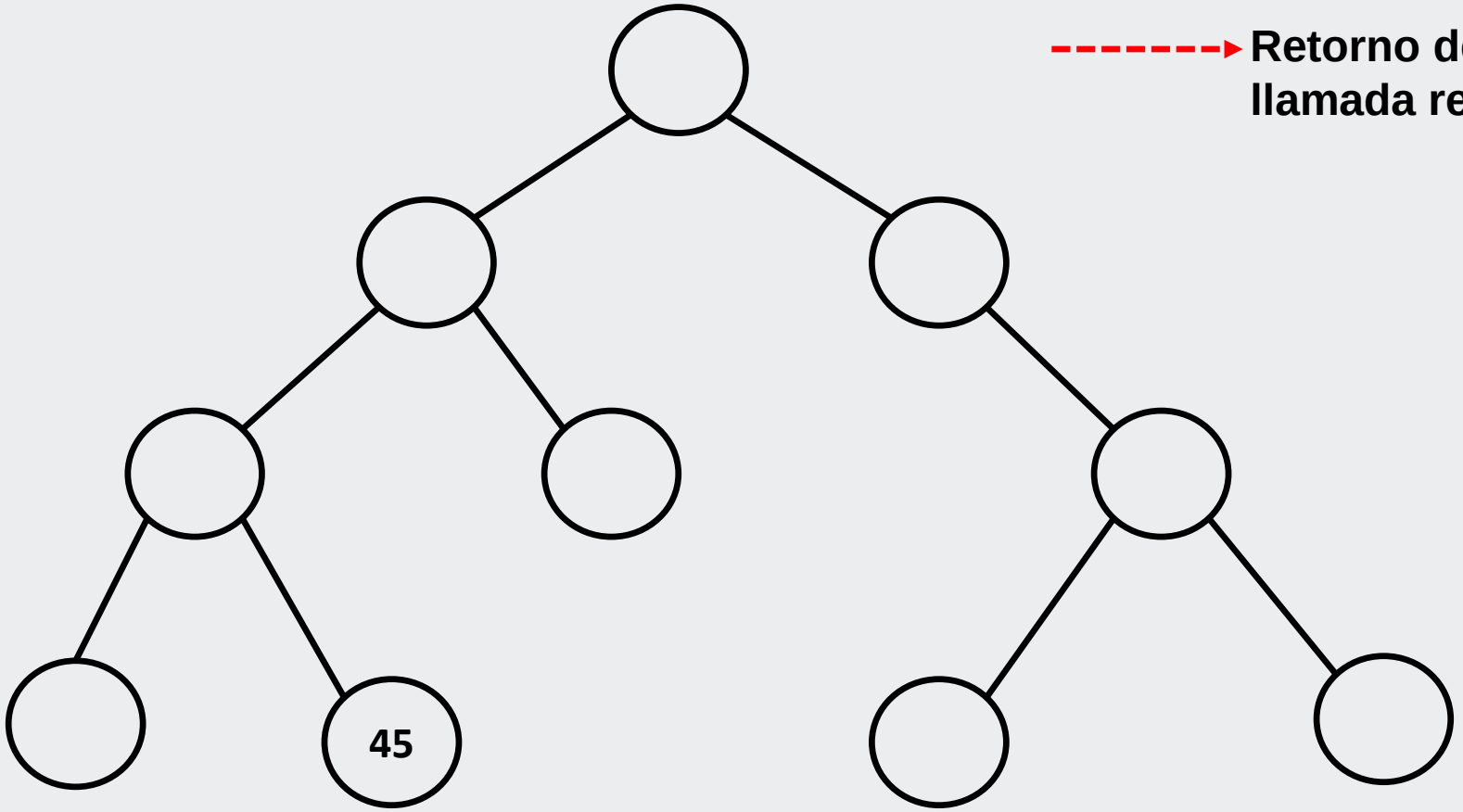


EJEMPLO: BUSQUEDA LINEAL (el árbol binario no es de búsqueda)

Universidad Católica del Uruguay

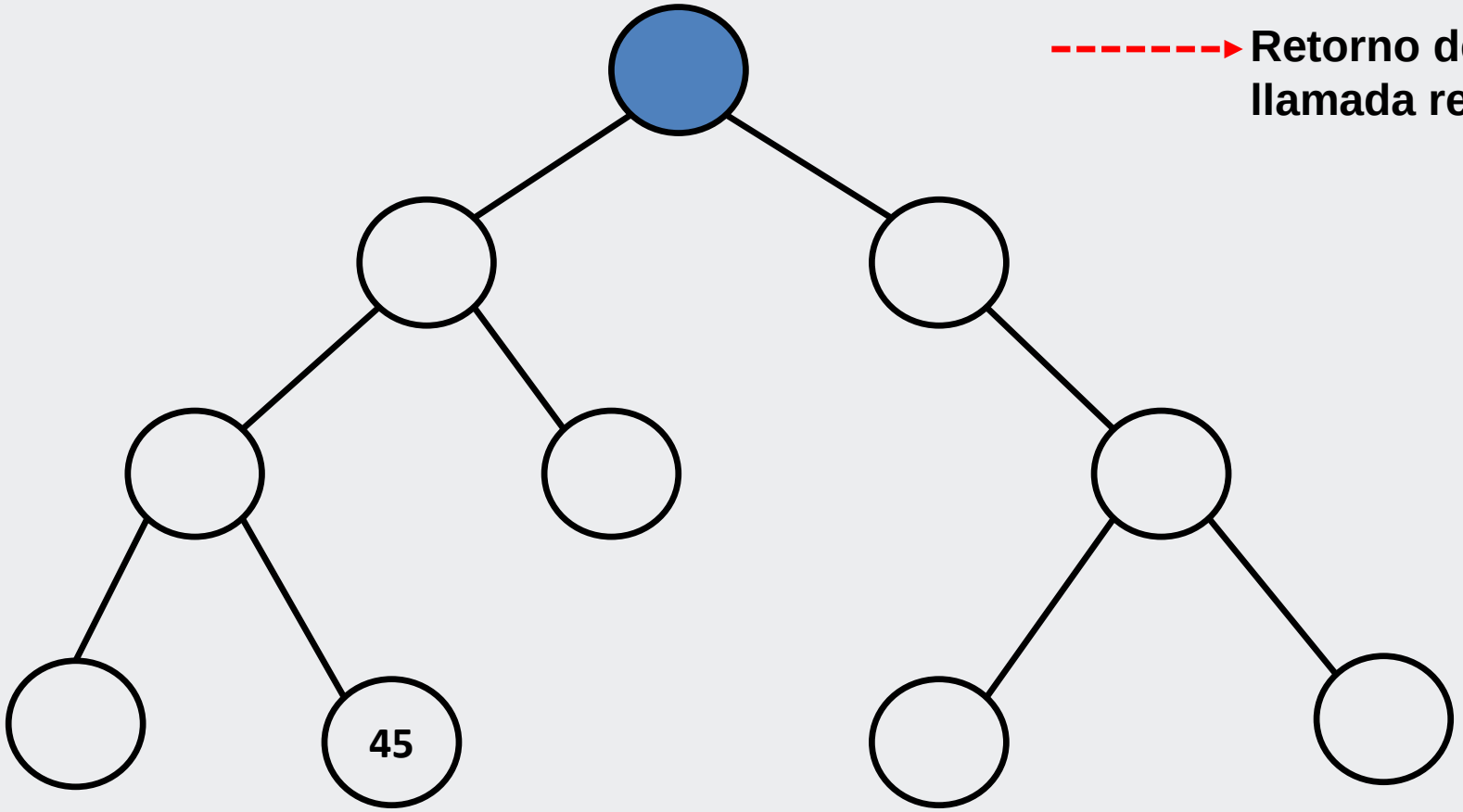
```
tipoElementoAB.encuentraLinealDos( Comparable unaEti): de tipo boolean
//devuelve verdadero si hay un nodo que tenga esa etiqueta, y falso en caso contrario.
Comienzo
 Si this.etiqueta = unaEti entonces
     Devolver Verdadero
 Fin si
 está ← Falso
 Si hijoIzquierdo <> nulo entonces
     está ← hijoIzquierdo.encuentraLinealDos(unaEti)
 Fin si
 Si hijoDerecho <> nulo Y no(está) entonces
     está ← hijoDerechoencuentraLinealDos(unaEti)
 Fin si
 devolver está
Fin
```

“encuentra lineal uno” con “unaEti” = 45

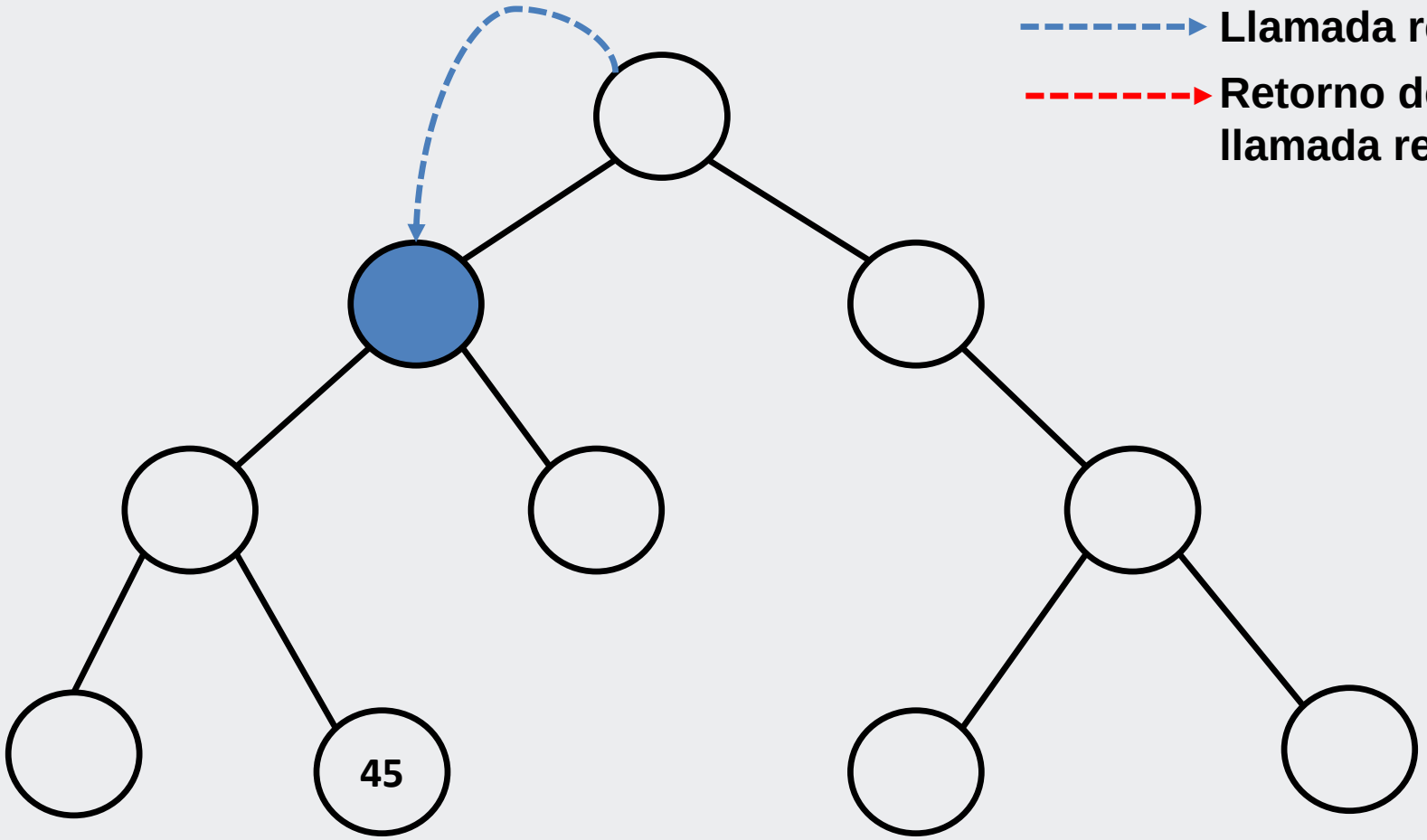


“encuentra lineal uno” con “unaEti” = 45

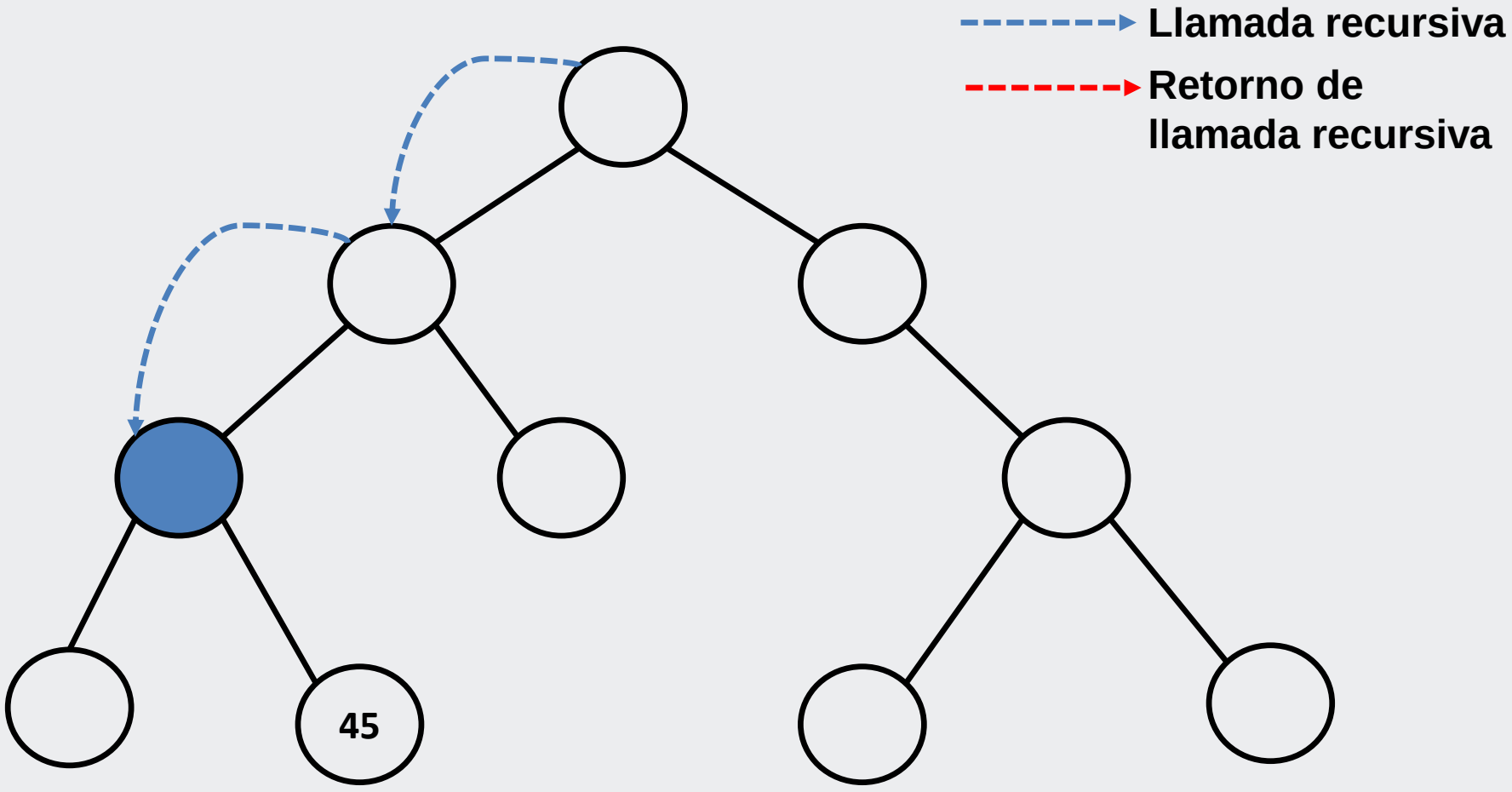
Universidad Católica del Uruguay



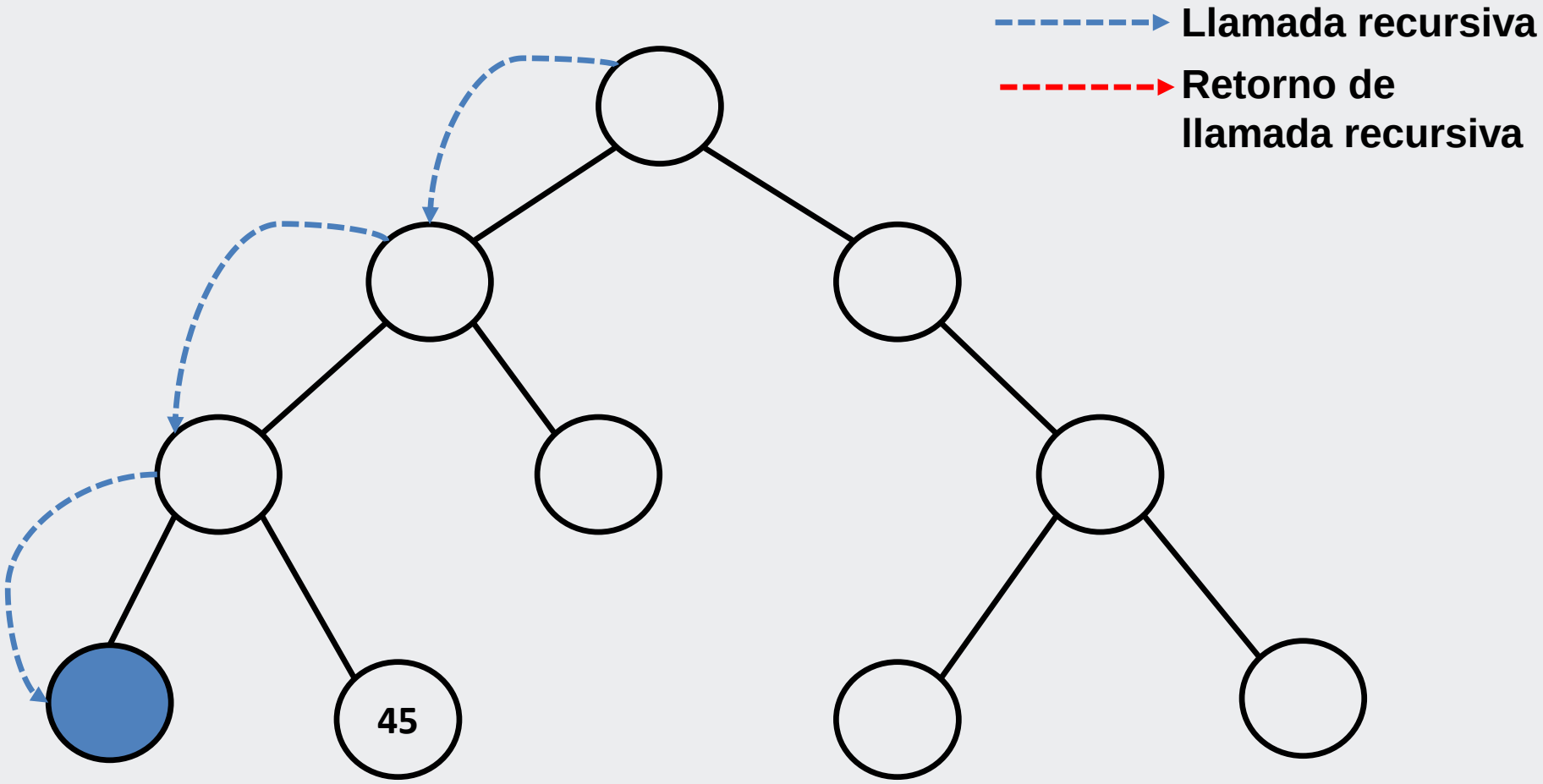
“encuentra lineal uno” con “unaEti” = 45



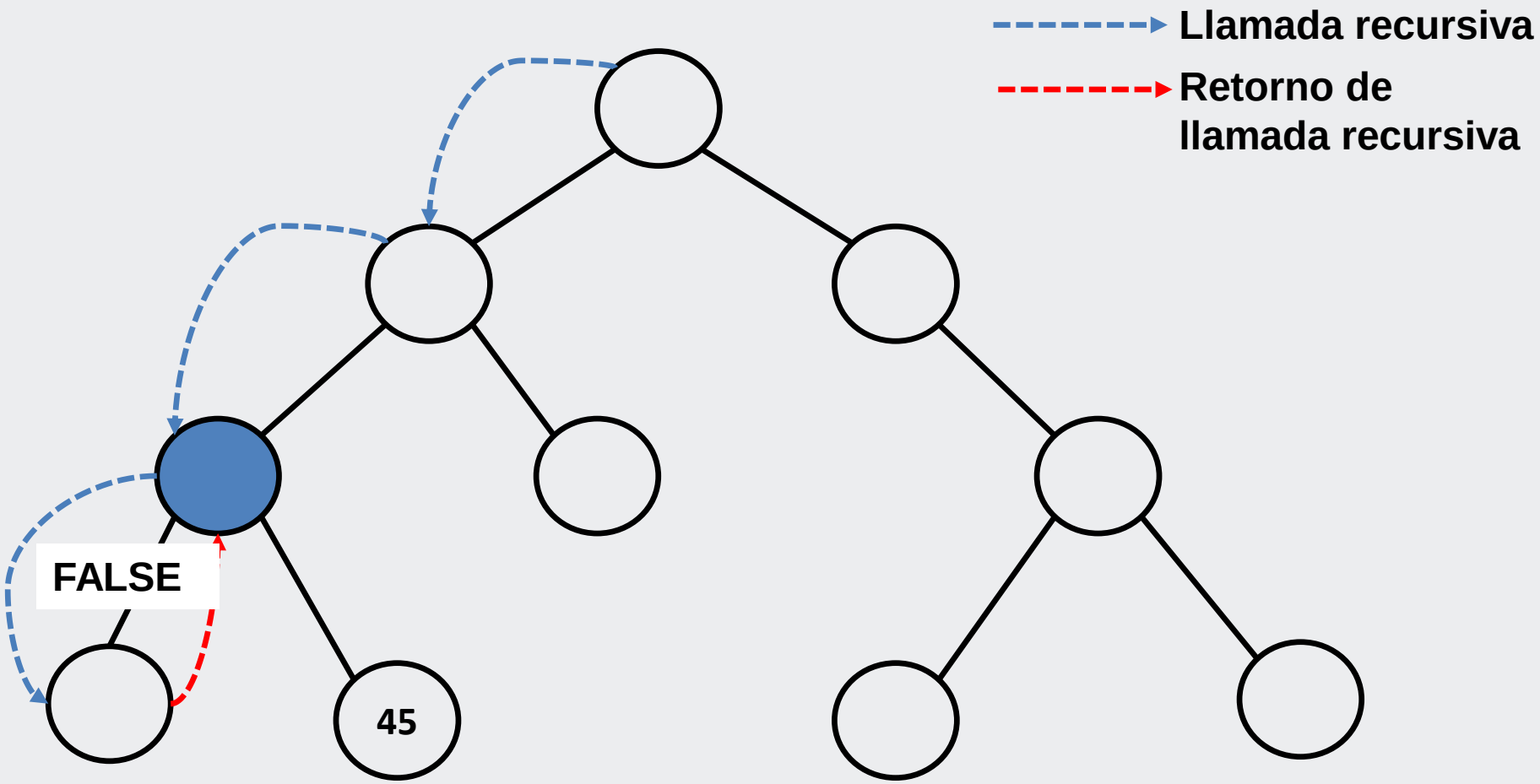
“encuentra lineal uno” con “unaEti” = 45



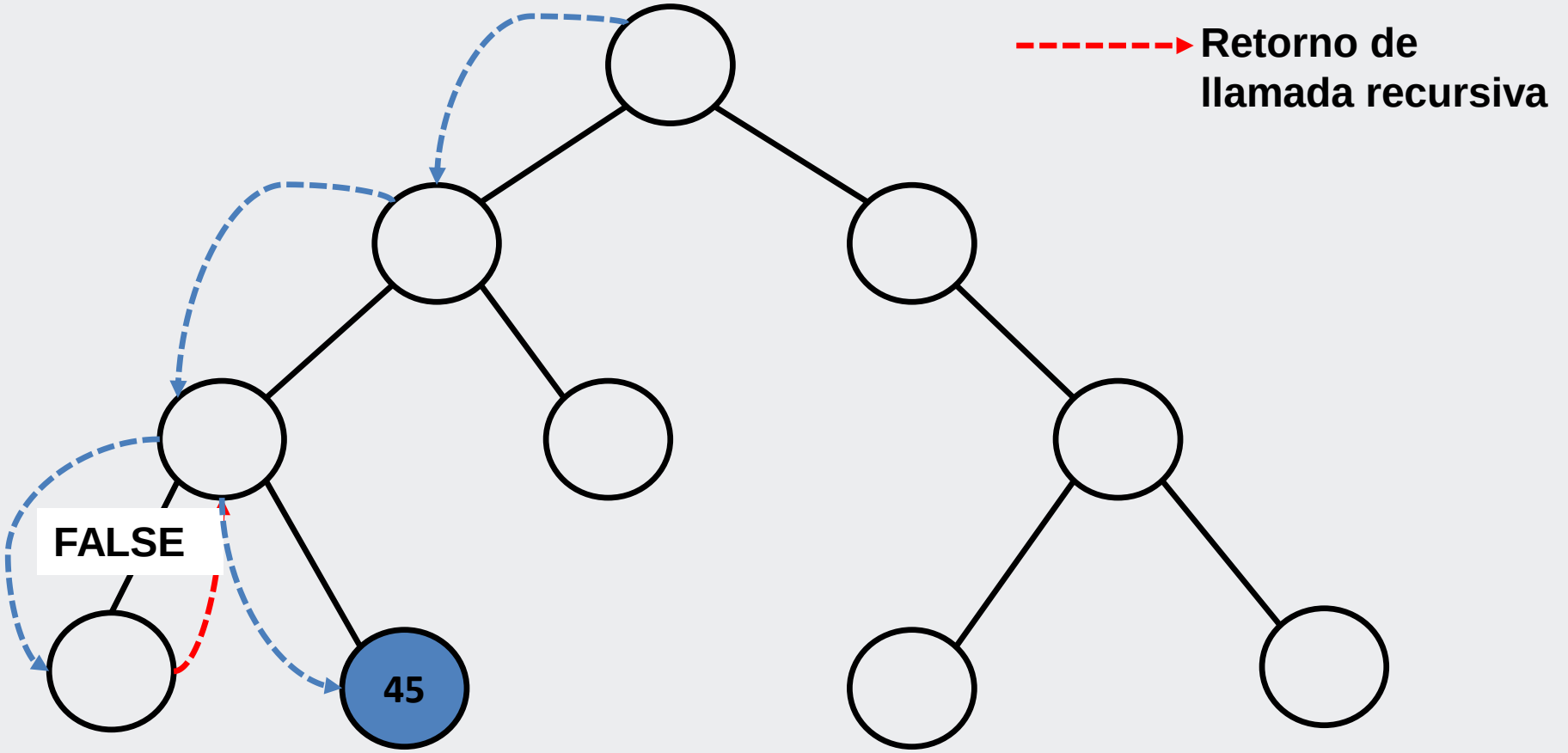
“encuentra lineal uno” con “unaEti” = 45



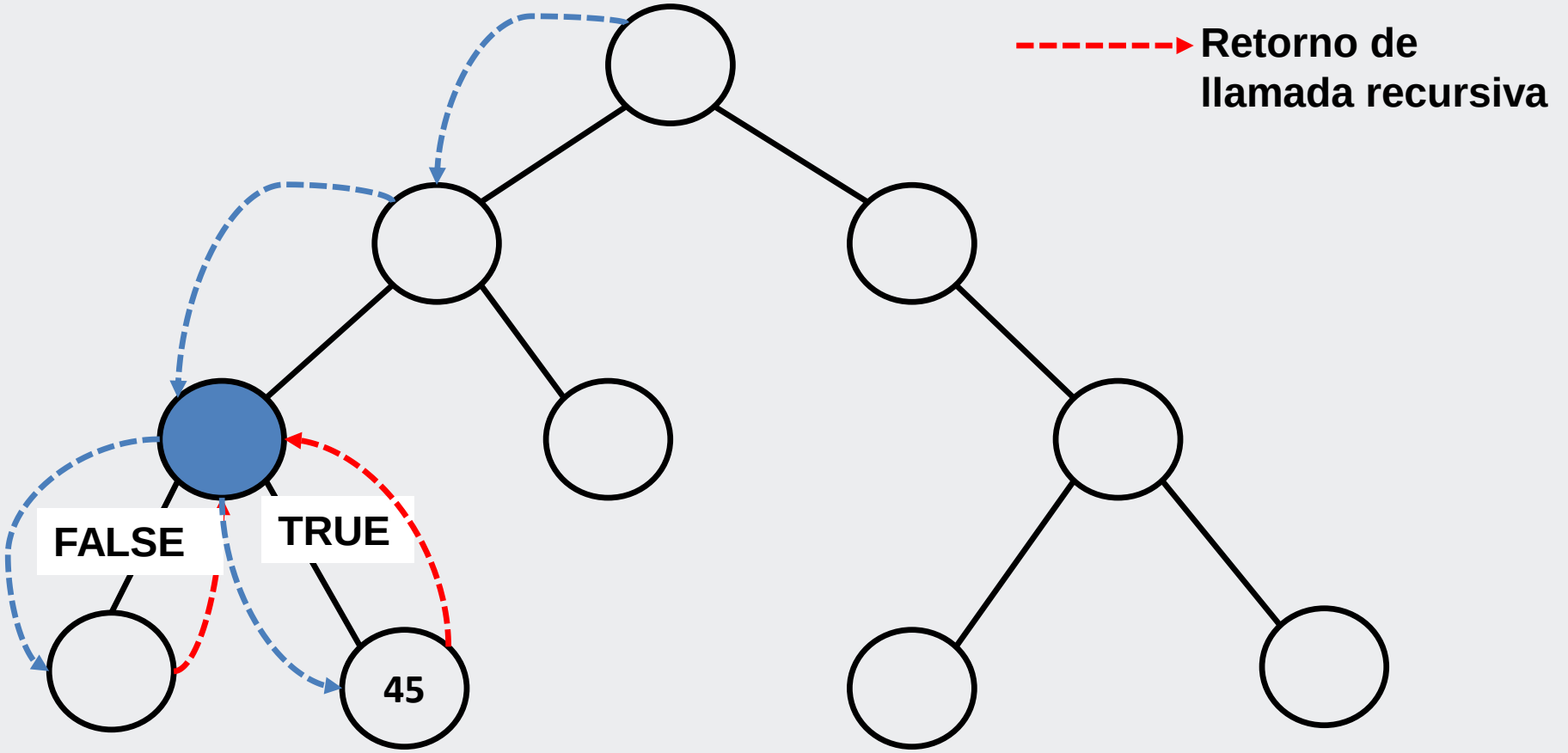
“encuentra lineal uno” con “unaEti” = 45



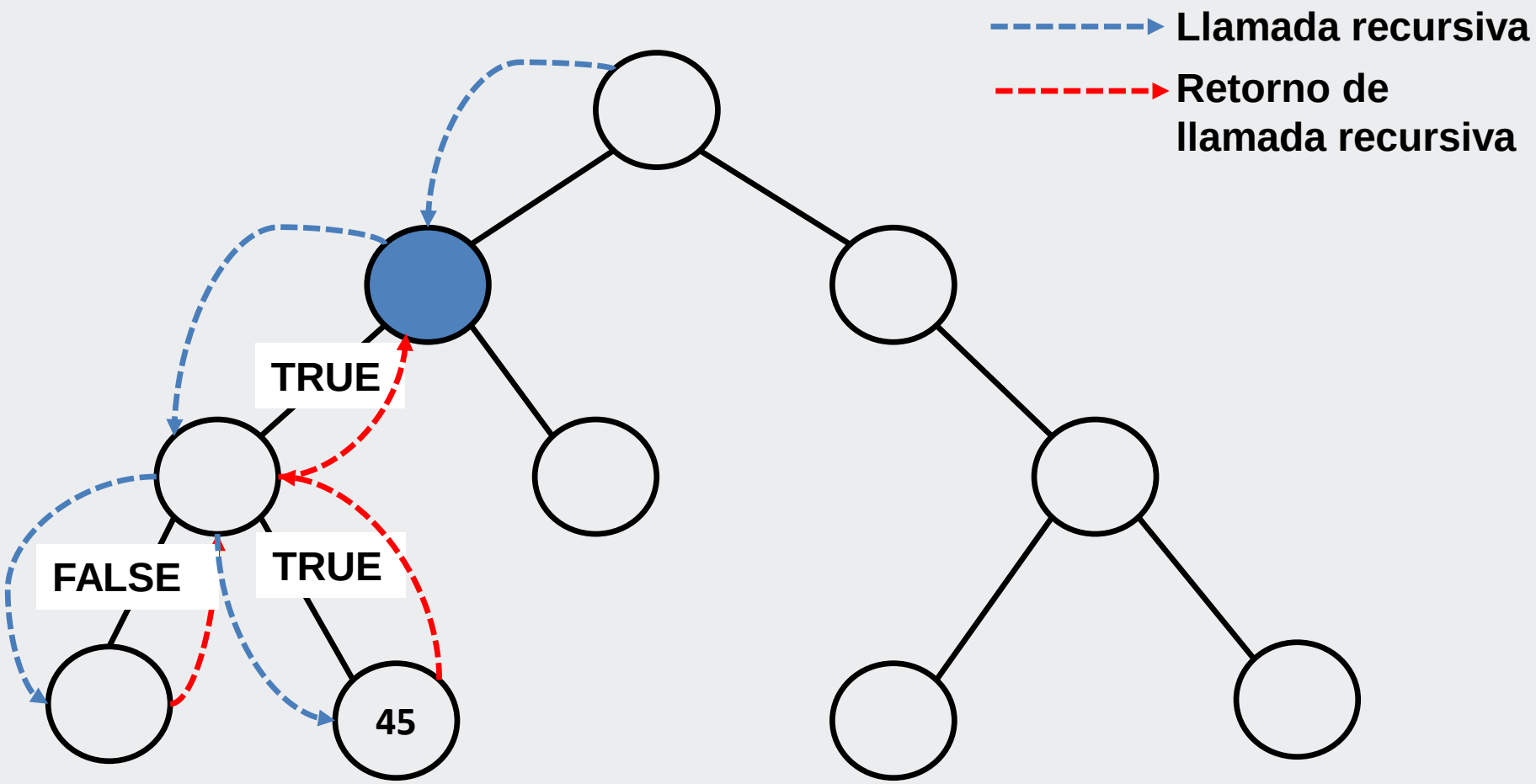
“encuentra lineal uno” con “unaEti” = 45



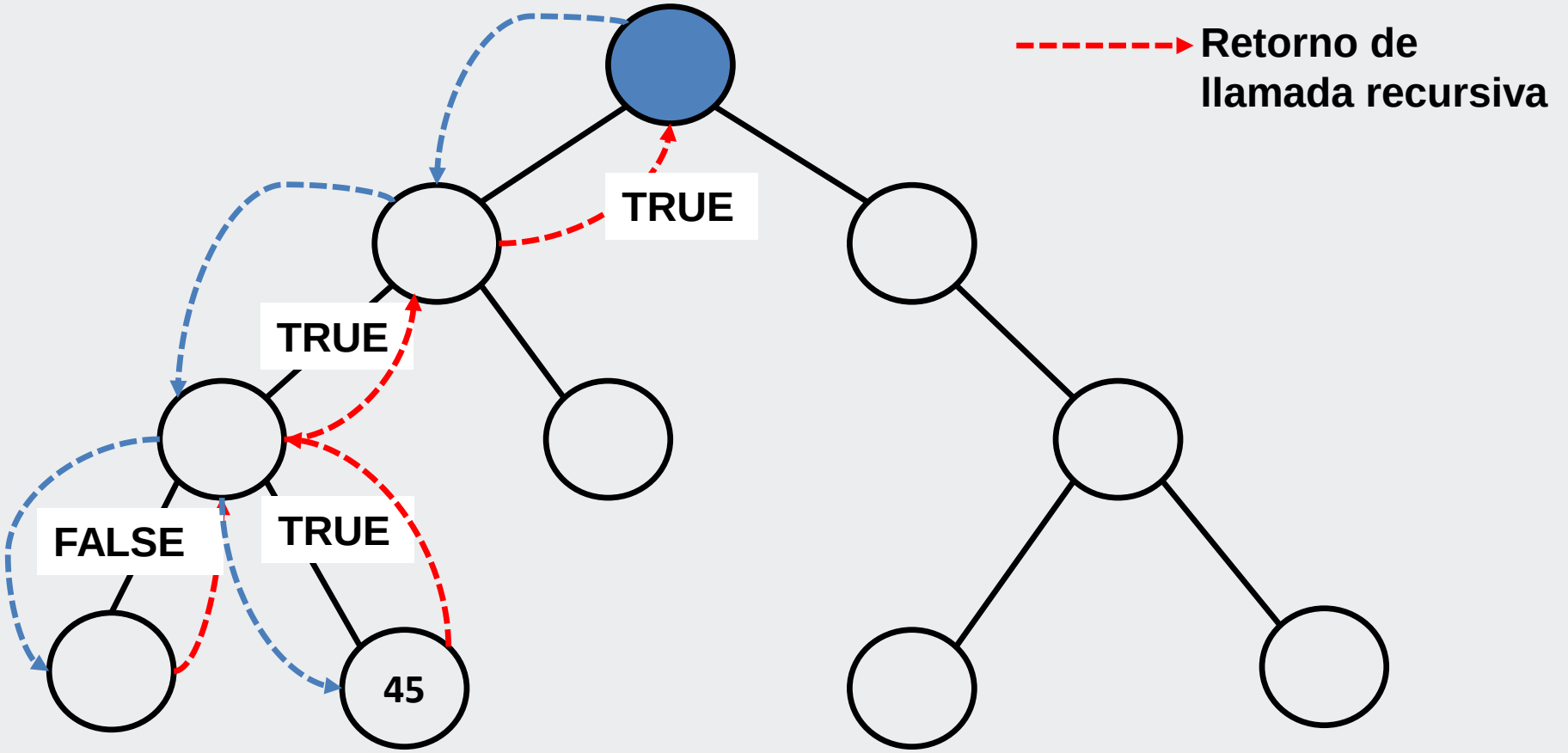
“encuentra lineal uno” con “unaEti” = 45



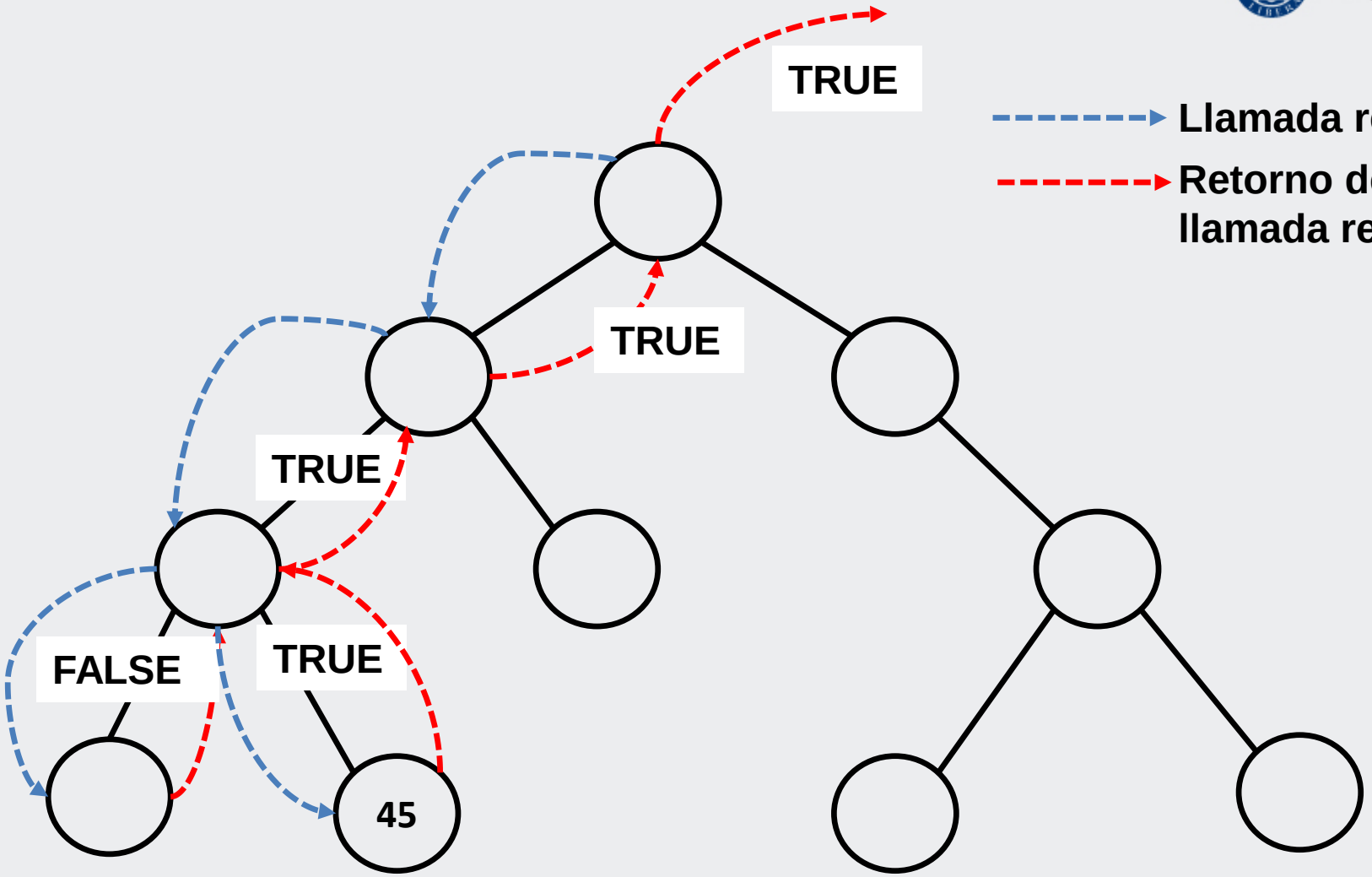
“encuentra lineal uno” con “unaEti” = 45



“encuentra lineal uno” con “unaEti” = 45



“encuentra lineal uno” con “unaEti” = 45



Universidad Católica del Uruguay

ARBOLES BINARIOS DE BÚSQUEDA

ARBOLES BINARIOS DE BÚSQUEDA

Universidad Católica del Uruguay

- Representan una de las aplicaciones más usadas de los árboles binarios.
- Son usados para almacenar un cierto conjunto de elementos de datos, con el propósito de que la búsqueda de un elemento se resuelva en un orden del tiempo de ejecución logarítmico.

ORDEN LINEAL vs ORDEN LOGARÍTMICO

Universidad Católica del Uruguay

- En una lista (encadenada o en array) la búsqueda de un elemento es de orden lineal.
- En un árbol binario de búsqueda se espera que la búsqueda de un elemento sea de orden logarítmico.
- Por ejemplo, si N fuera 1.000.000, logaritmo en base de 1.000.000 es aproximadamente 20.
- Es decir, que una búsqueda en vez de resolverse con un millón de comparaciones, se resolverá con 20.

ARBOLES BINARIOS DE BUSQUEDA

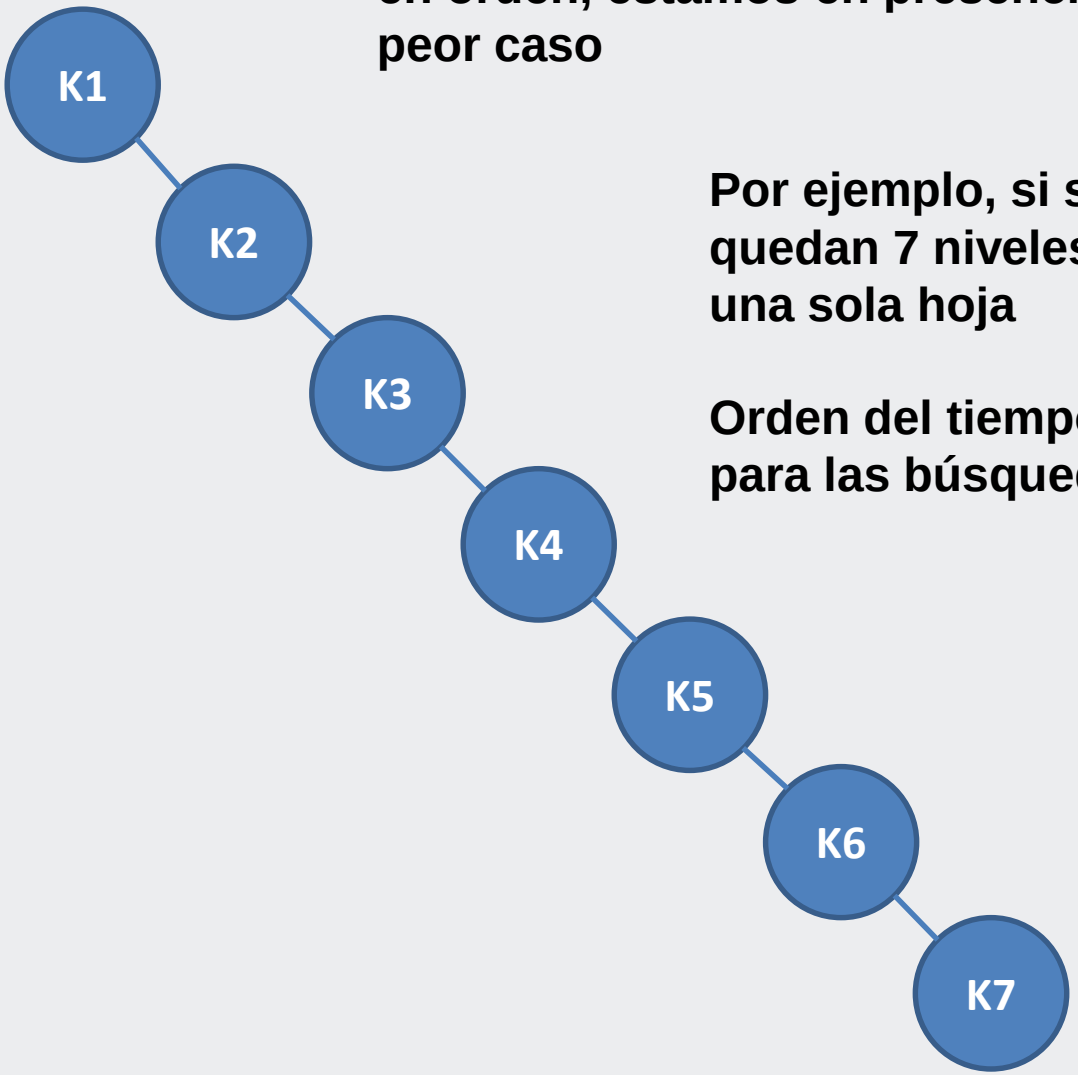
Universidad Católica del Uruguay

- El problema es que, dependiendo de cómo se realice la inserción (y eliminación) de los elementos, el árbol puede quedar "desbalanceado", es decir con una altura mayor a la deseable, con elementos cuyas diferencia de altura entre sus dos subárboles es muy grande.
- El caso **ideal** es que el árbol quede perfectamente balanceado, con todas sus hojas al mismo nivel.

ARBOLES BINARIOS DE BUSQUEDA

- Los criterios para que los árboles que se puedan auto balancear al insertar y al eliminar, los estudiaremos en la próxima unidad.
- Veremos ahora cómo se puede medir cómo quedó armado el árbol, si se acerca más al caso ideal (todas las hojas al mismo nivel) o si se acerca más al peor caso (una sola hoja en el último nivel).

si las claves se insertaran
en orden, estamos en presencia del
peor caso



Por ejemplo, si se insertaran 7 claves:
quedan 7 niveles (altura 6)
una sola hoja

Orden del tiempo de ejecución queda lineal
para las búsquedas (inserción y eliminación)

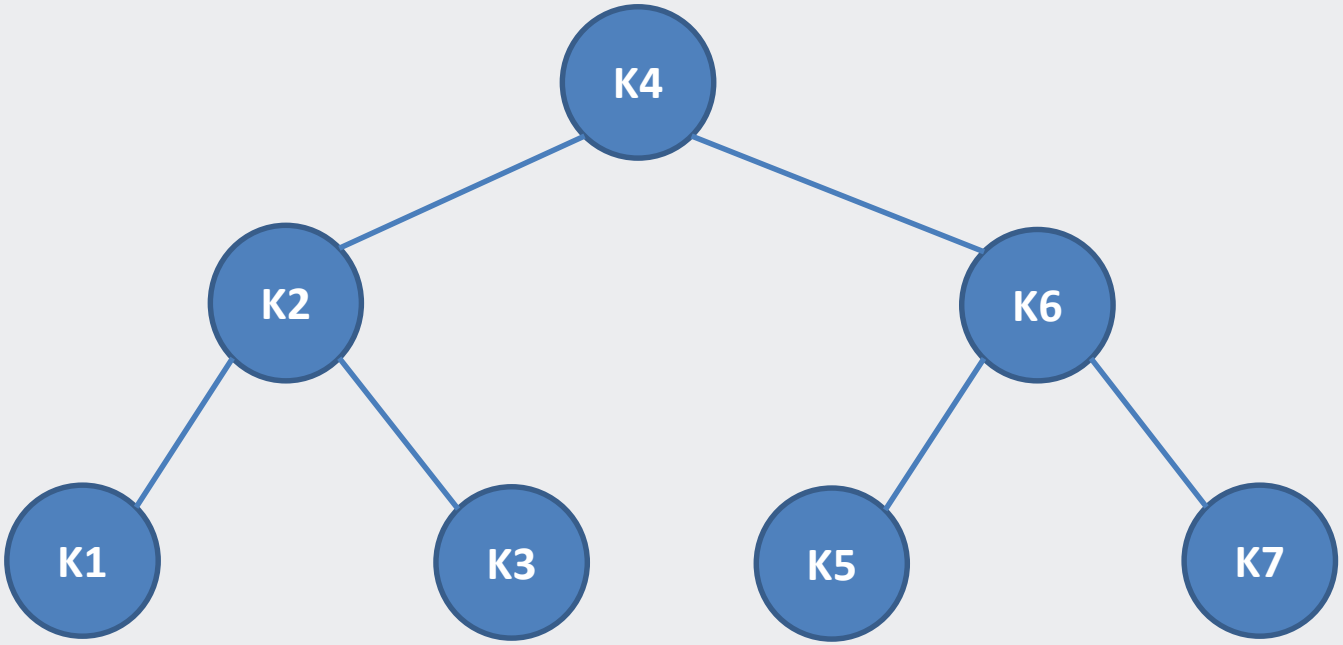
PEOR CASO

El mejor caso sería si se insertan de tal manera que el árbol quede con todas sus hojas al mismo nivel.

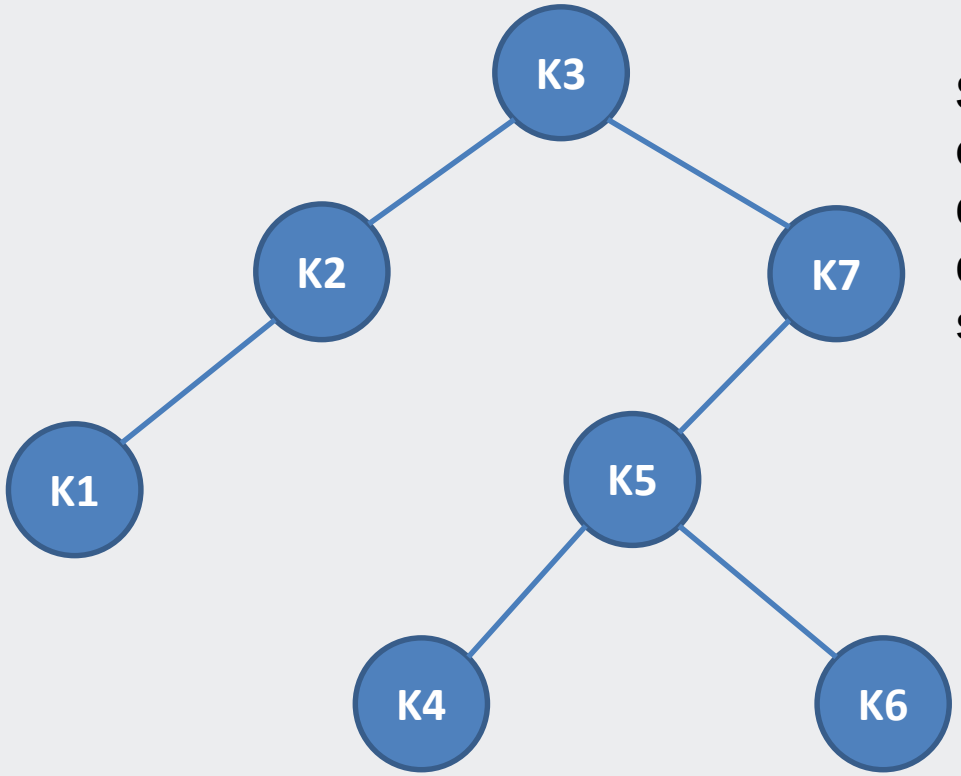
Por ejemplo, si se insertaran 7 claves:
quedan 3 niveles (altura 2)
4 hojas (una más que los nodos internos)

Orden del tiempo de ejecución logarítmico para las búsquedas (inserción y eliminación)

MEJOR CASO



¿CASO PROMEDIO?



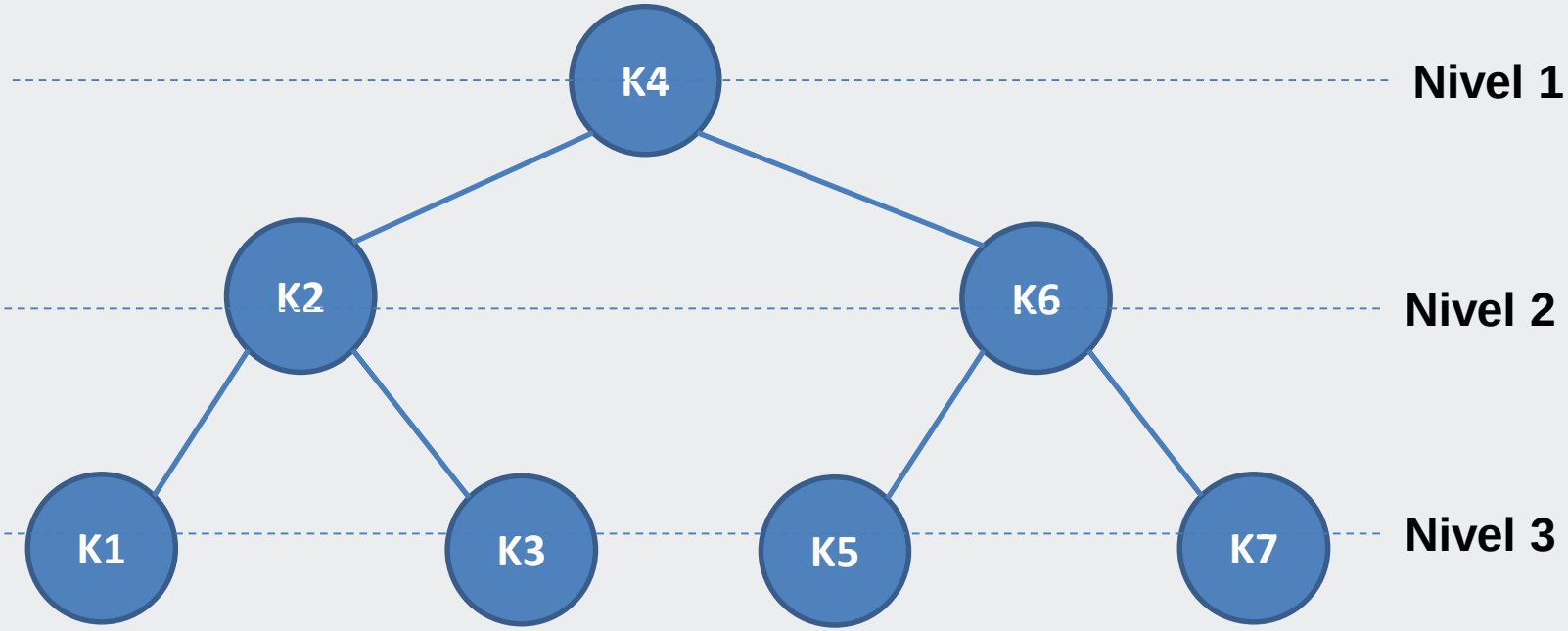
Si las claves se van insertando en orden aleatorio, es decir, siguen una distribución uniforme, cabe esperar que la eficiencia de las búsquedas sean un 39% peor que el mejor caso[1]

1.- Algorithms and Data Structures, Niklaus Wirth, 1985

EFICIENCIA DE LA BÚSQUEDA

- La cantidad de comparaciones necesarias para resolver una búsqueda, está determinada por el nivel en el que se encuentra la clave.
- Por ello, la cantidad de comparaciones necesarias para encontrar a todas las claves, será igual a la suma de los niveles de los nodos.
- A la suma de los niveles de todos los nodos se le conoce como

LONGITUD DE TRAYECTORIA INTERNA



1 nodo en el nivel 1, 2 nodos en el nivel 2, 4 nodos en el nivel 3

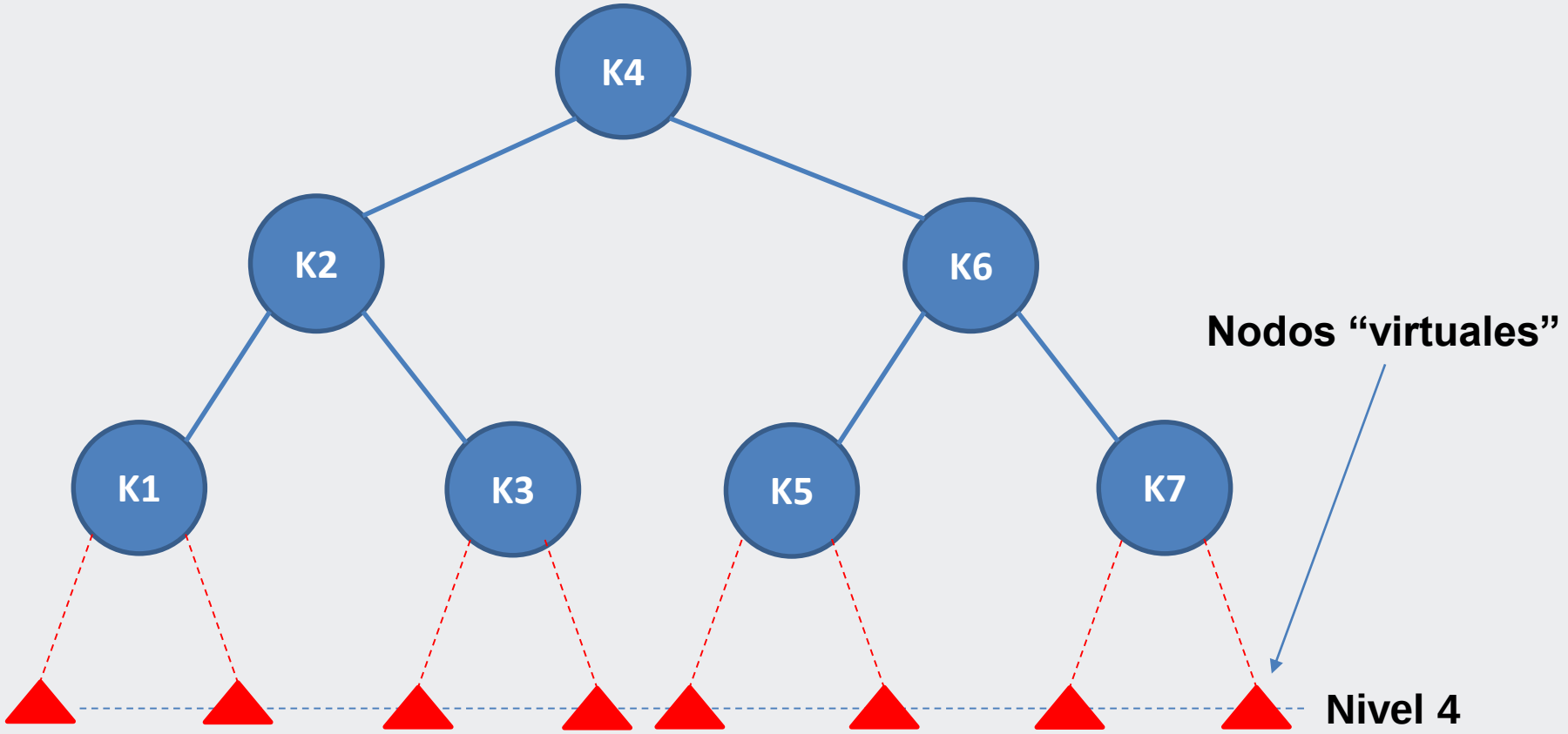
LTI = 1 + 2*2 + 4*3 = 17

LTI = 1 + 2 + 3 + 4 + 5 + 6 + 7 = 28 **(7 por 8 sobre 2)**

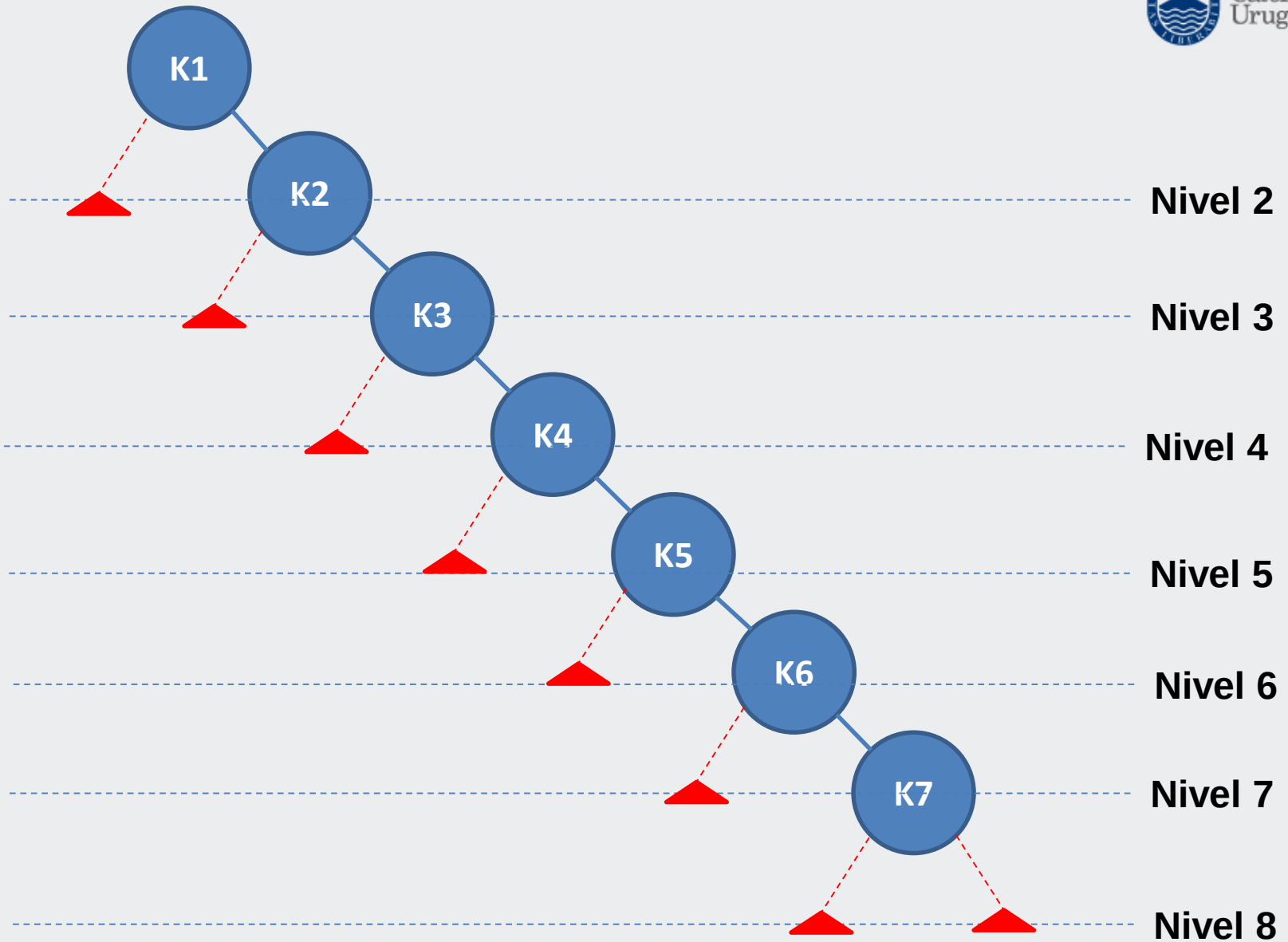
EFICIENCIA DE LA BÚSQUEDA

- La LONGITUD DE TRAYECTORIA INTERNA representa el esfuerzo para resolver las búsquedas EXITOSAS
- Las búsquedas INFRUCTUOSAS son resueltas cuando se encuentra un subárbol vacío.
- A la suma de los niveles de todos los puntos que resuelven las búsquedas infructuosas se le llama:

LONGITUD DE TRAYECTORIA EXTERNA



LTE = 8 * 4 = 32



LTE = 2 + 3 + 4 + 5 + 6 + 7 + 8 + 8 = 43

EFICIENCIA DE LA BÚSQUEDA

- La LONGITUD DE TRAYECTORIA INTERNA PROMEDIO se obtiene dividiendo la LTI por el tamaño.
- La LONGITUD DE TRAYECTORIA EXTERNA PROMEDIO se obtiene dividiendo la LTE por el tamaño más uno.