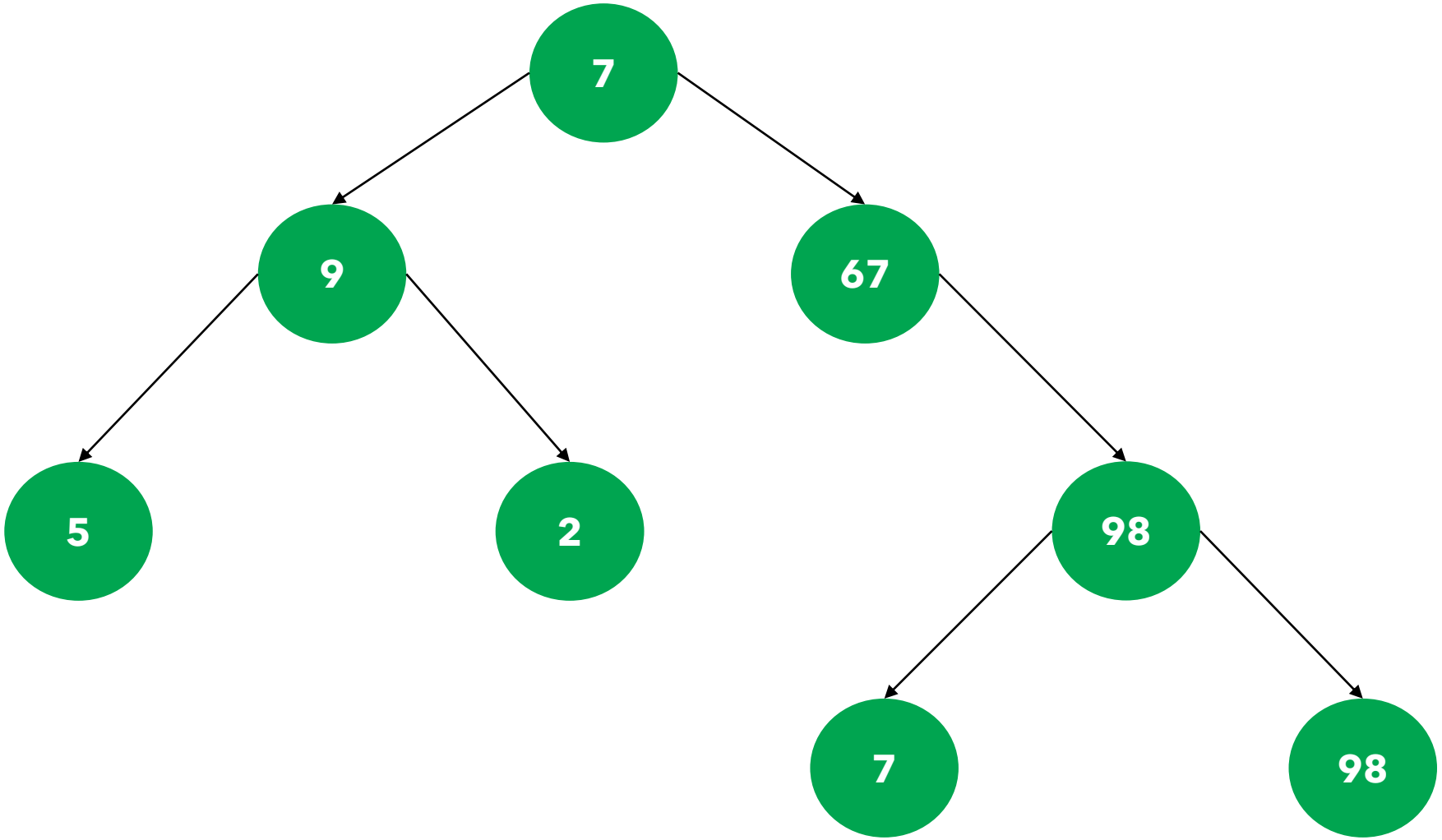
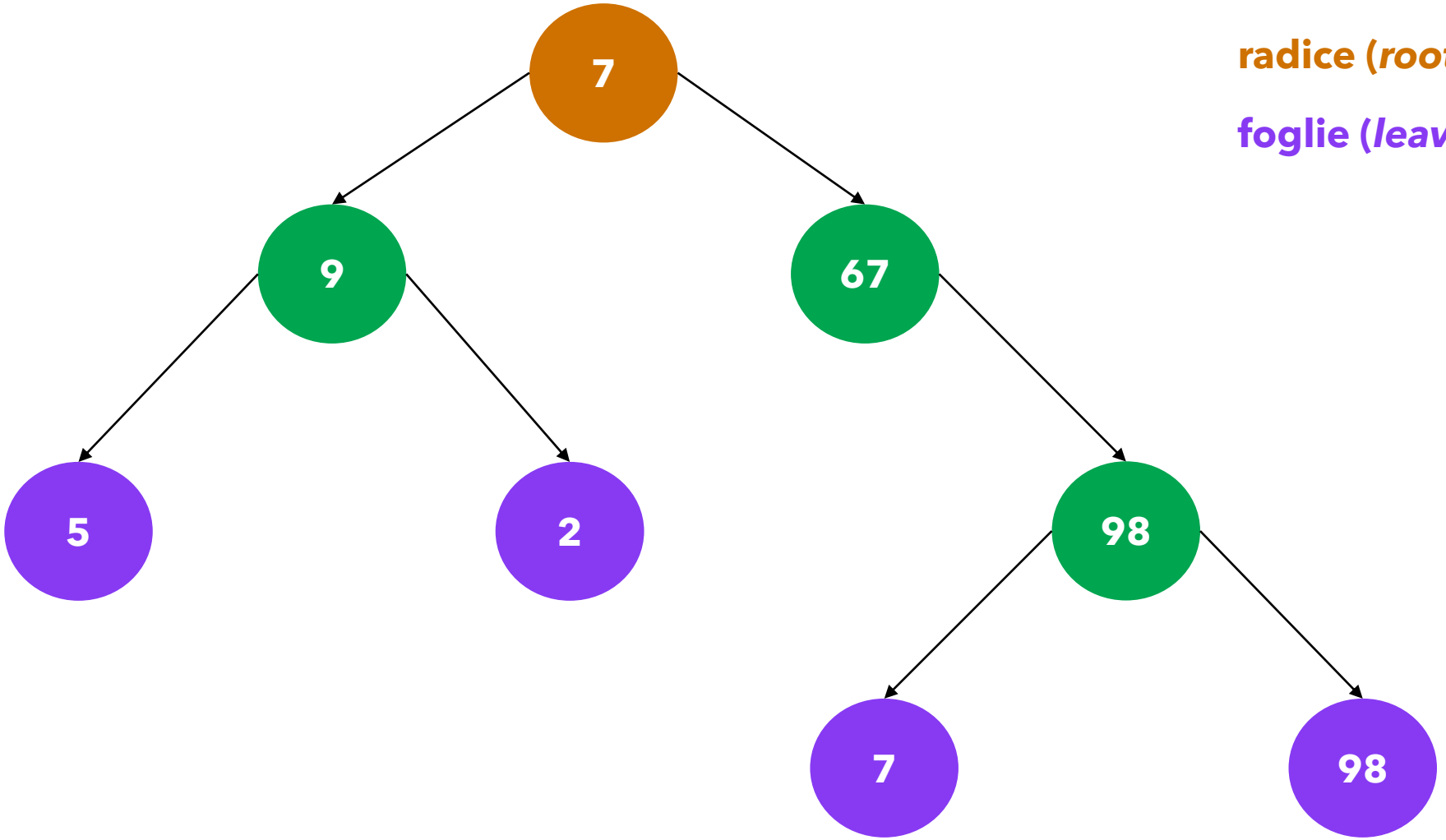
Alberi binari
(*binary trees*)

Liceo G.B. Brocchi – Bassano del Grappa (VI)
Liceo Scientifico – opzione scienze applicate
Giovanni Mazzocchin

Alberi binari



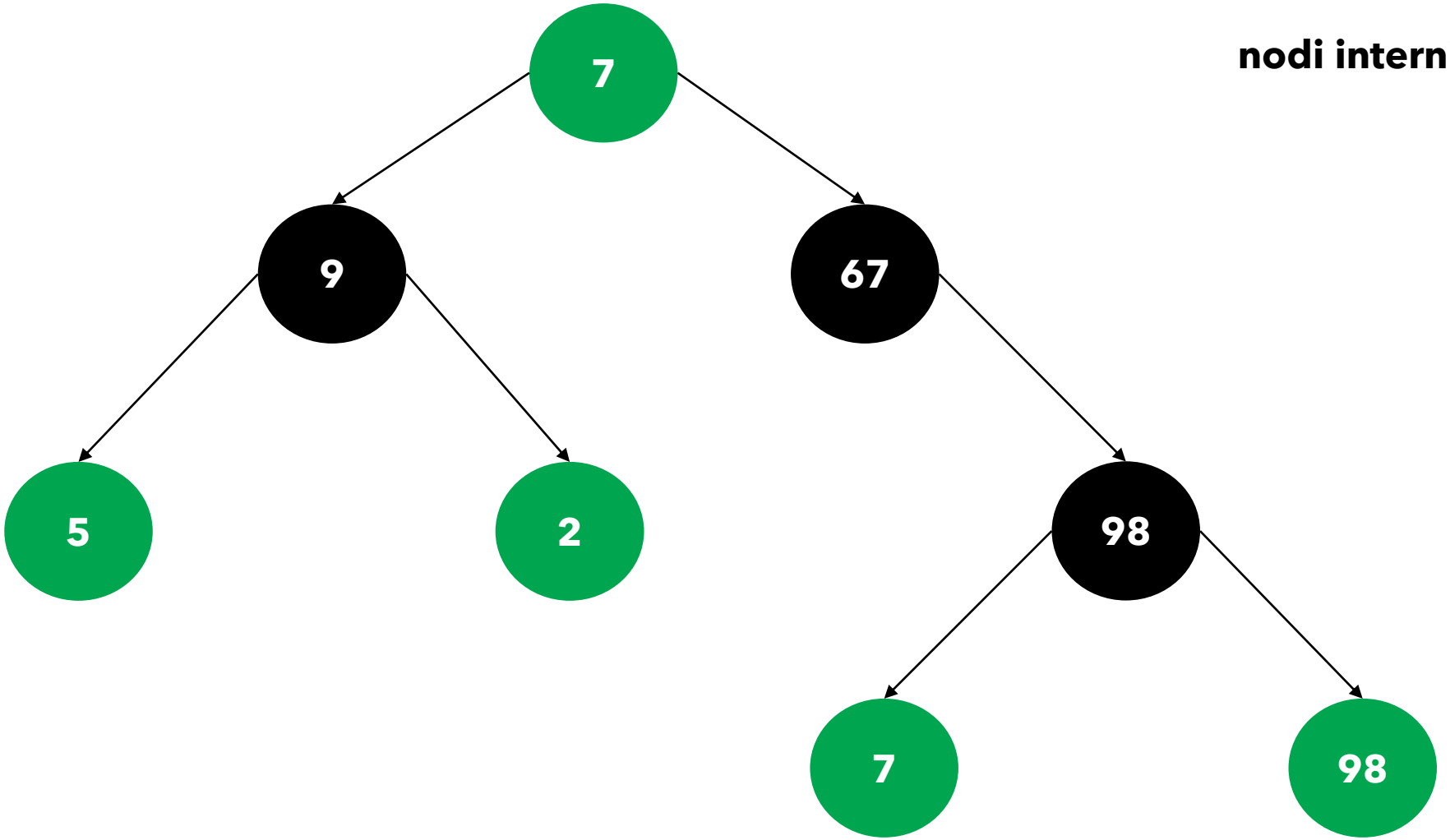
Alberi binari



radice (*root*)

foglie (*leaves*)

Alberi binari



nodi interni (*inner nodes*)

Definizione ricorsiva

Un **albero binario** è:

- un albero senza alcun nodo

oppure

- un nodo che punta a due **alberi binari**

Implementazione in C

```
typedef struct tree_node {
  int key;
  struct tree_node* left;
  struct tree_node* right;
} T_NODE;
```

Alberi binari di ricerca (*binary search trees*)

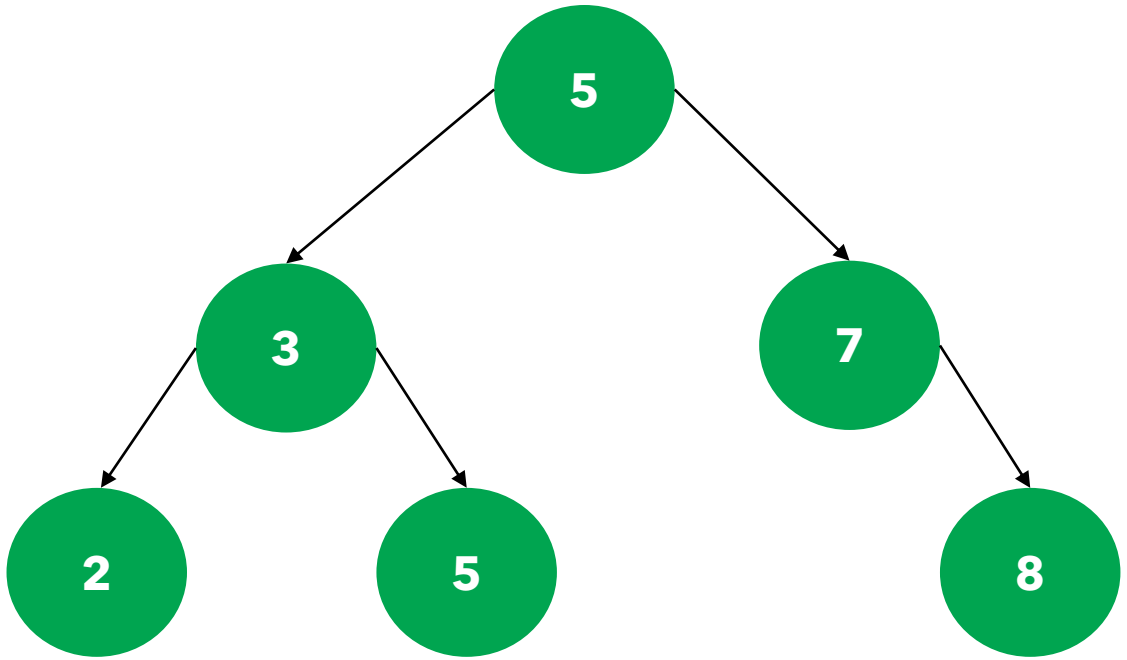
- **Binary-search-tree property**:

  se n è un nodo di un albero binario di ricerca, `n.key` è la chiave di n, `n.left` è la radice del sottoalbero sinistro di n, e `n.right` è la radice del sottoalbero destro di n, allora:

  - per ogni nodo `nl` del sottoalbero radicato in `n.left`, è vero che `nl.key <= n.key`
  - per ogni nodo `nr` del sottoalbero radicato in `n.right`, è vero che `n.key < nr.key`

  una struttura del genere è molto utile per ricercare informazioni

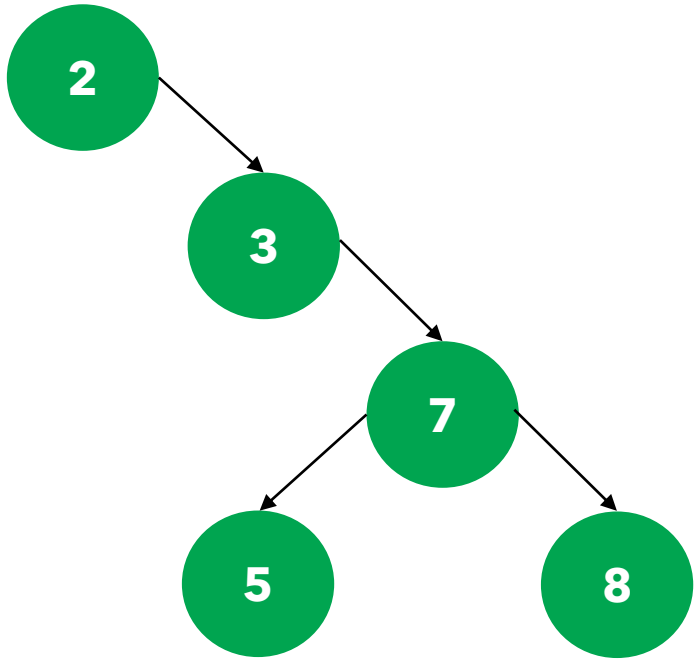
Alberi binari - altezza



questo BST ha **altezza 2**: l'altezza di un albero binario è la distanza del percorso più lungo dalla radice ad una foglia. In questo caso, abbiamo 3 percorsi radice foglia di lunghezza 2 (la lunghezza del percorso è il numero delle frecce):

- `5 -> 3 -> 2; 5 -> 3 -> 5; 5 -> 7 -> 8`

Alberi binari - altezza



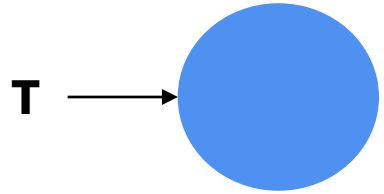
questo BST ha **altezza 3**

Calcolo ricorsivo dell'altezza

se T è un albero vuoto, ossia un puntatore NULL:

`height(T) = 0`

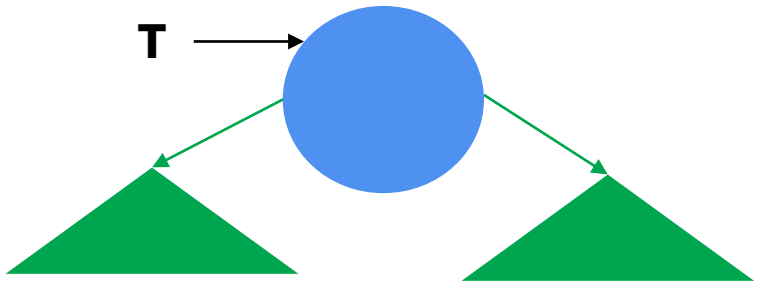
Calcolo ricorsivo dell'altezza



se T è un albero composto da 1 solo nodo **senza sottoalberi**, allora:

`height(T) = 0`

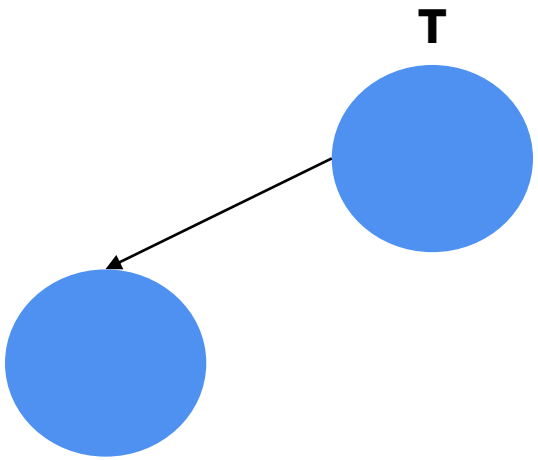
Calcolo ricorsivo dell'altezza



se T ha almeno 1 sottoalbero:

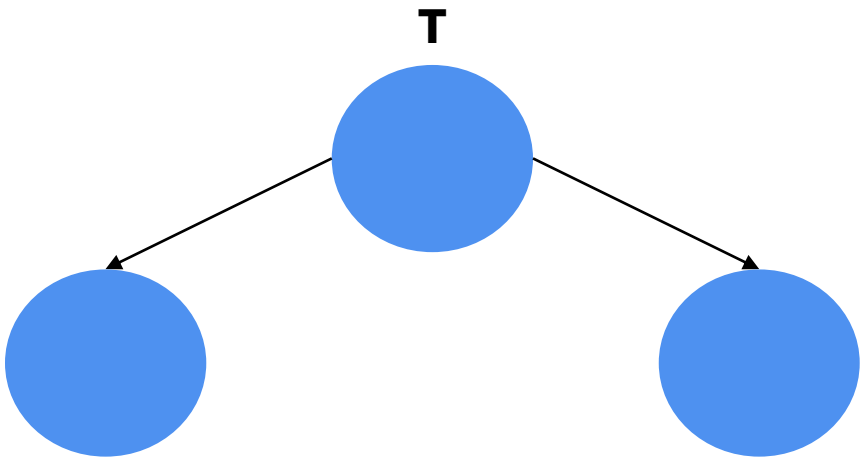
```
height(T) =
max(height(T.left), height(T.right)) + 1
```

Calcolo ricorsivo dell'altezza



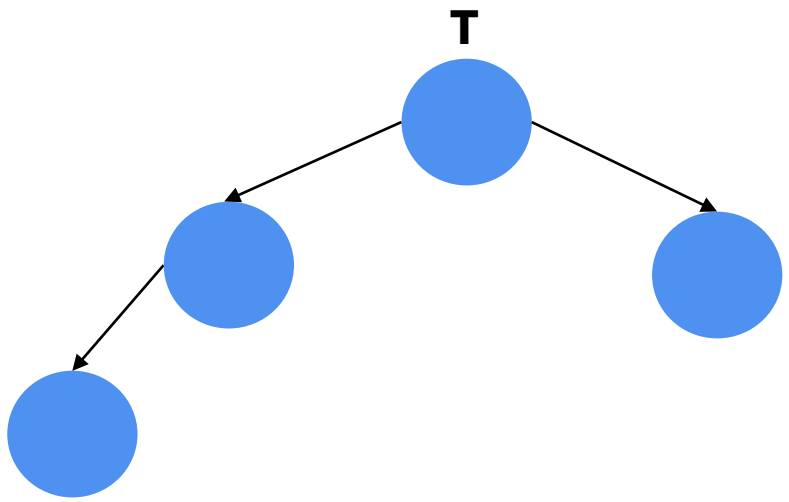

```
height(T) = max(height(T.left), height(T.right)) + 1 =
max(0, 0) + 1 = 0 + 1 = 1
```

Calcolo ricorsivo dell'altezza




```
height(T) = max(height(T.left), height(T.right)) + 1 =
max(0, 0) + 1 = 0 + 1 = 1
```

Calcolo ricorsivo dell'altezza




```
height(T) = max(height(T.left), height(T.right)) + 1 =
max(max(height(T.left.left), height(T.left.right)) + 1,
height(T.right)) + 1 =
max(max(0, 0) + 1, 0) + 1 = max(0 + 1, 0) + 1 = max(1, 0) + 1 =
1 + 1 = 2
```

Calcolo ricorsivo dell'altezza

```
height(T): returns int
  if T is nil:
    return 0
  if T.left is nil and T.right is nil:
    return 0
  heightLeftSubTree = height(T.left)
  heightRightSubTree = height(T.right)
  return max(heightLeftSubTree, heightRightSubTree) + 1
```

Analogia: lunghezza di una linked list

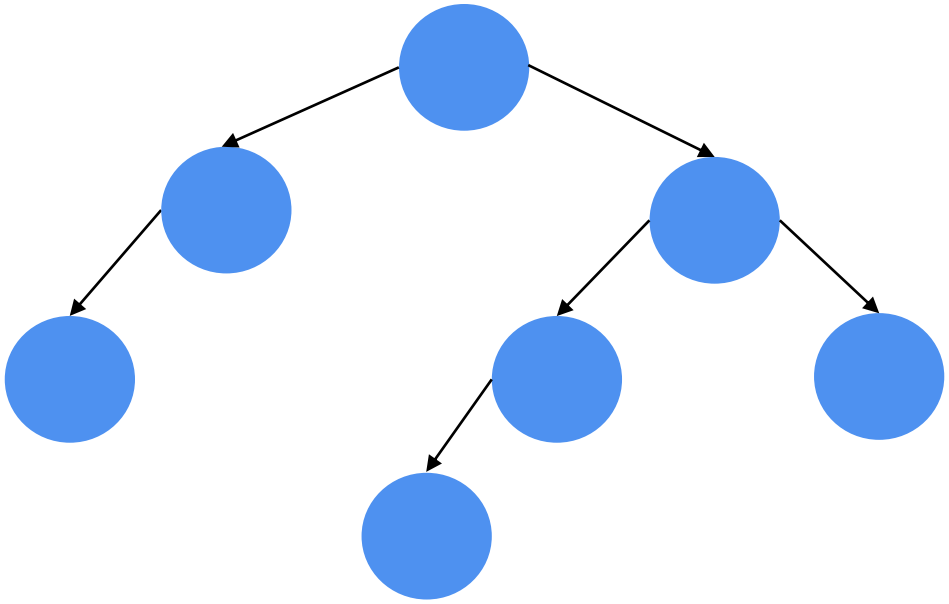
```
length(L): returns int
  if L is nil:
    return 0
  length_rem = length(L.next)
  return length_rem + 1
```

Calcolo dell'altezza: evoluzione del call stack

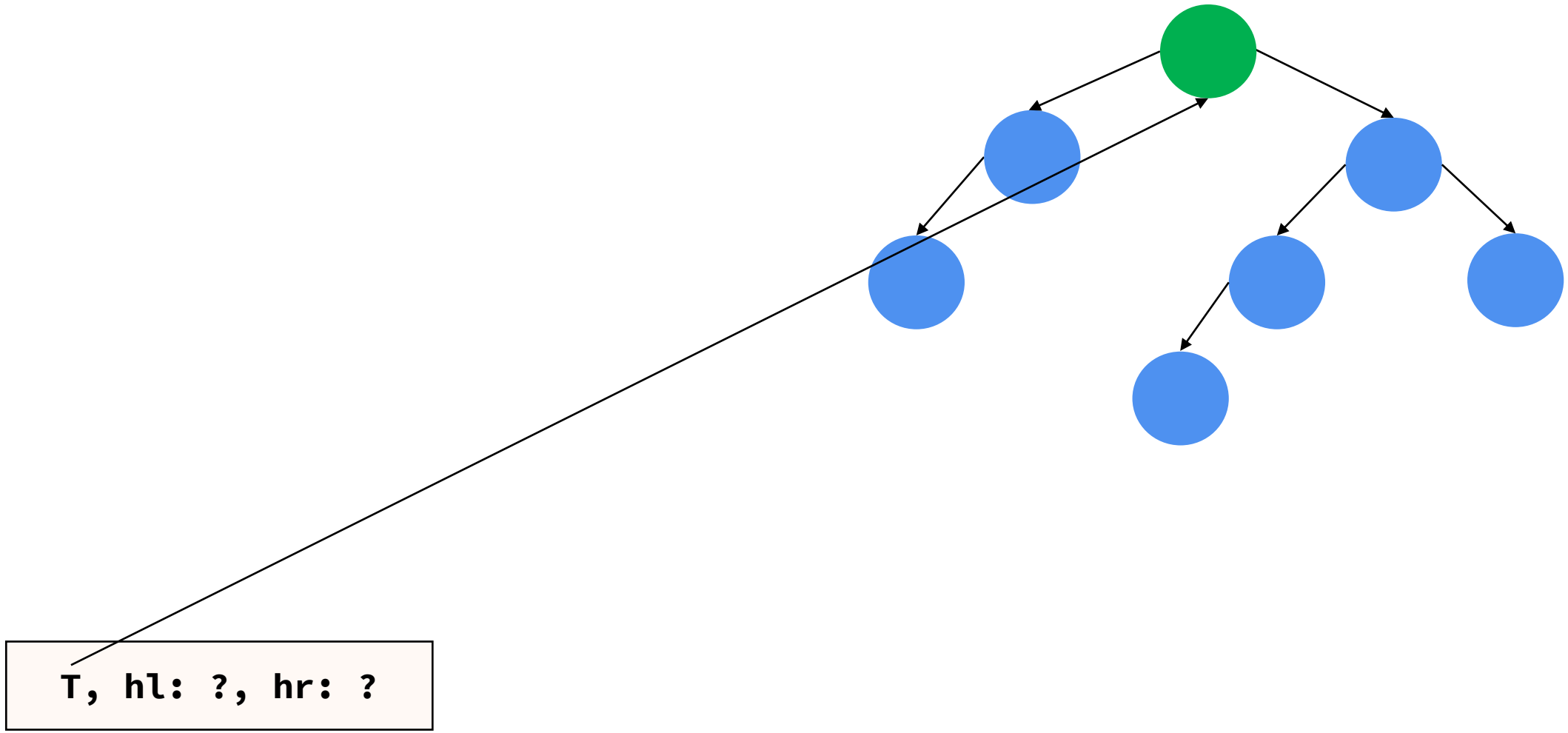
nodo radice dell'invocazione corrente

nodo per cui l'altezza è stata calcolata completamente

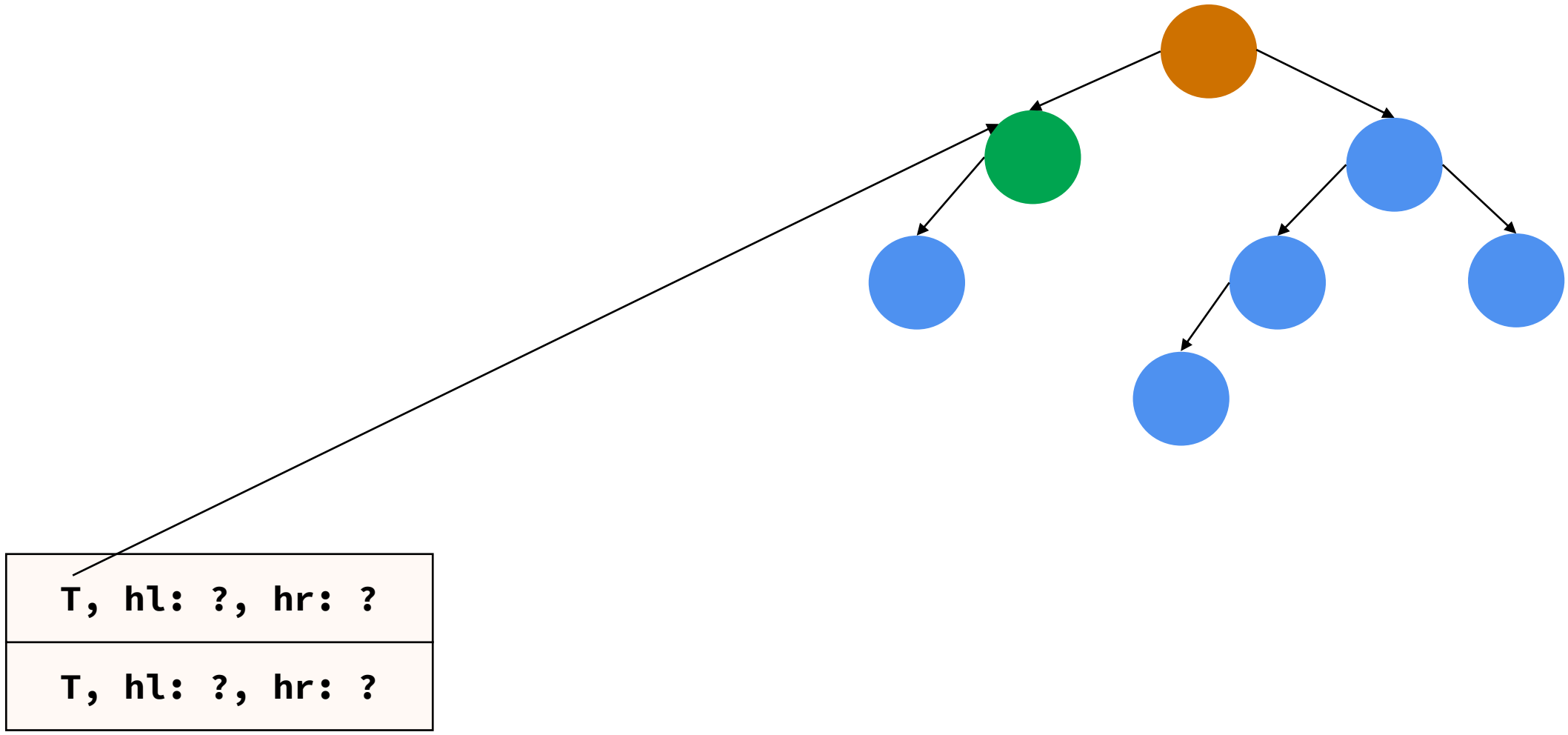
nodo per cui l'altezza non è stata calcolata completamente



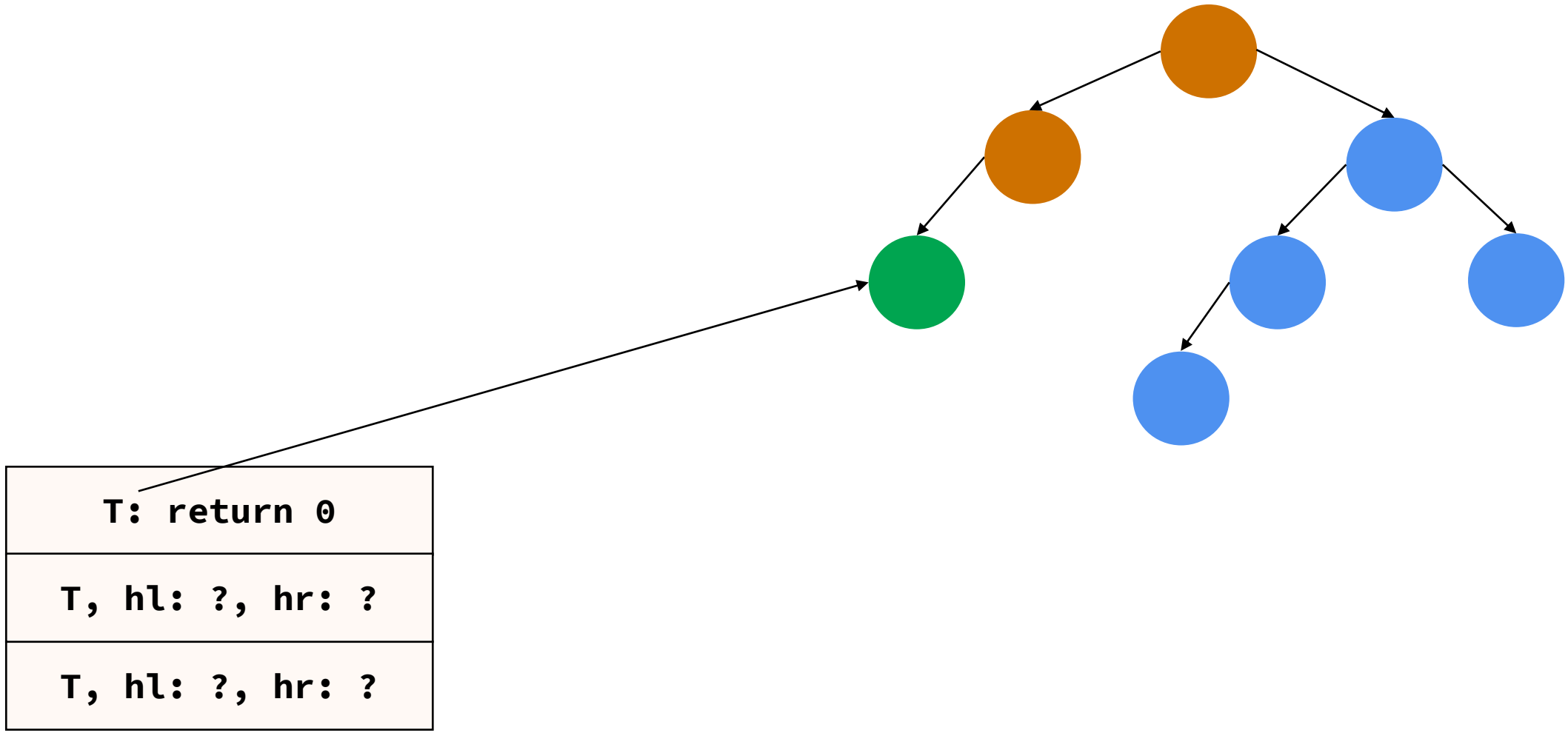
Calcolo dell'altezza: evoluzione del call stack



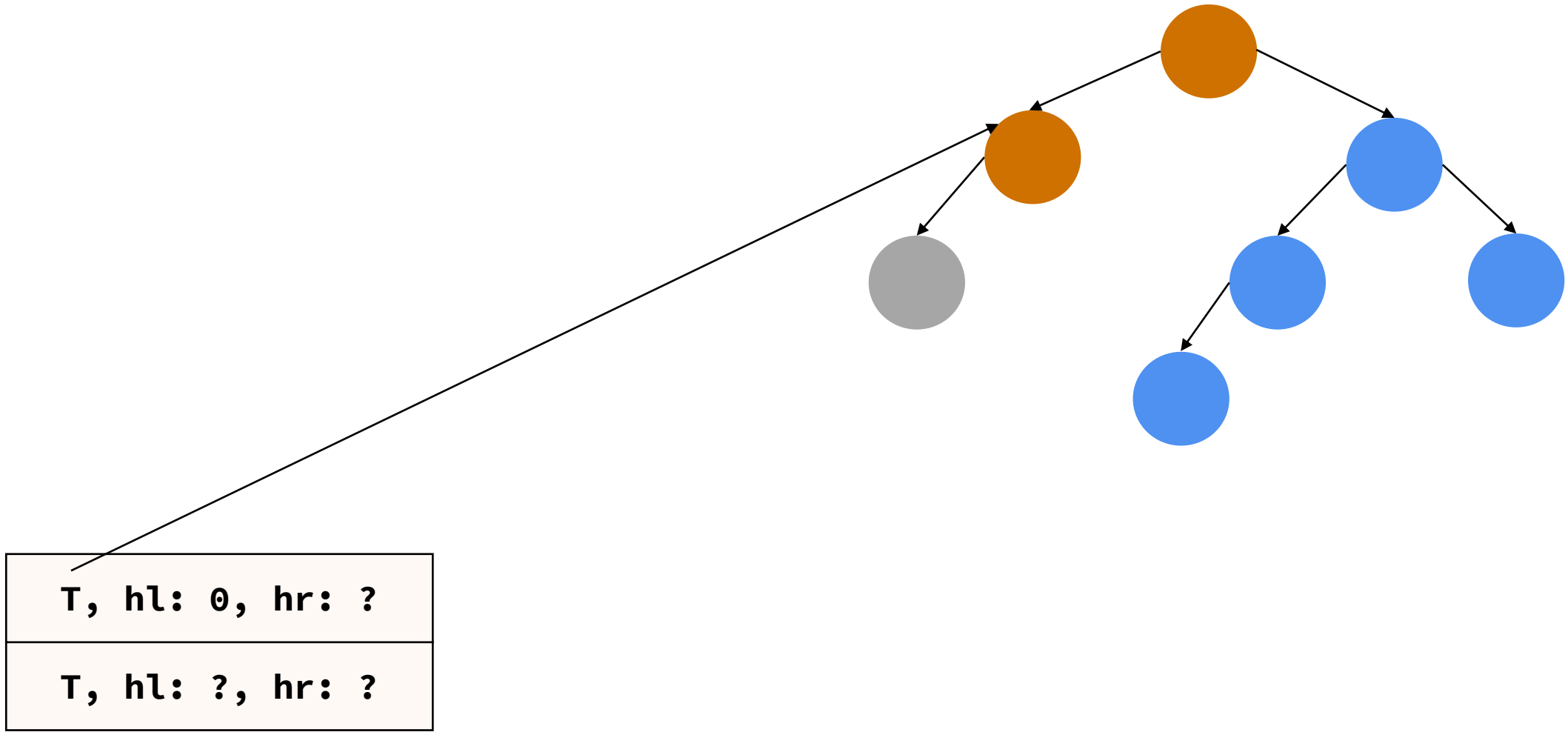
Calcolo dell'altezza: evoluzione del call stack



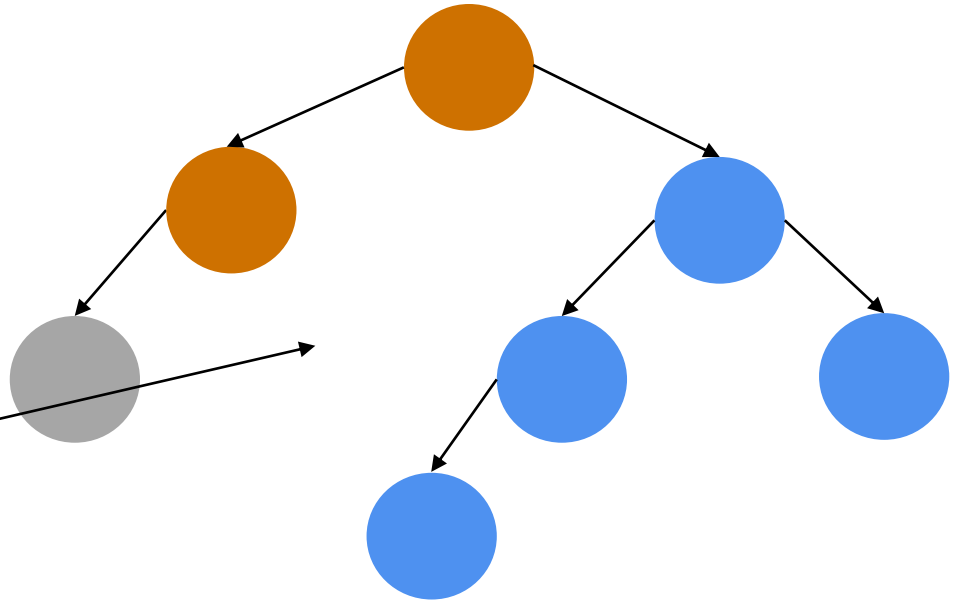
Calcolo dell'altezza: evoluzione del call stack



Calcolo dell'altezza: evoluzione del call stack

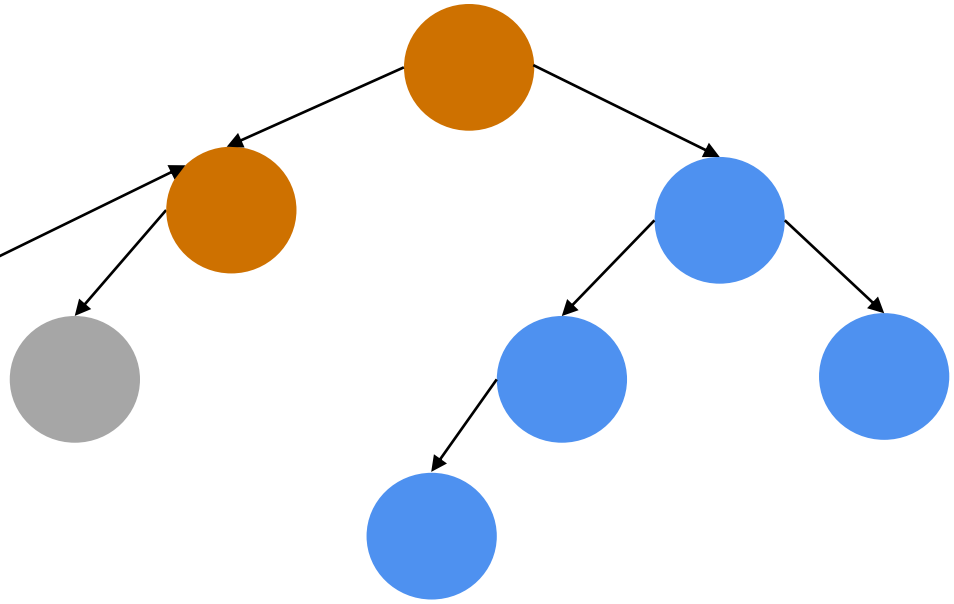


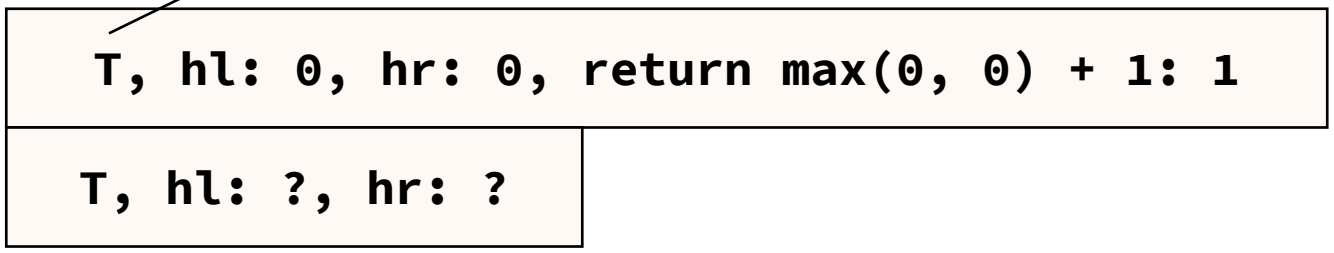
Calcolo dell'altezza: evoluzione del call stack



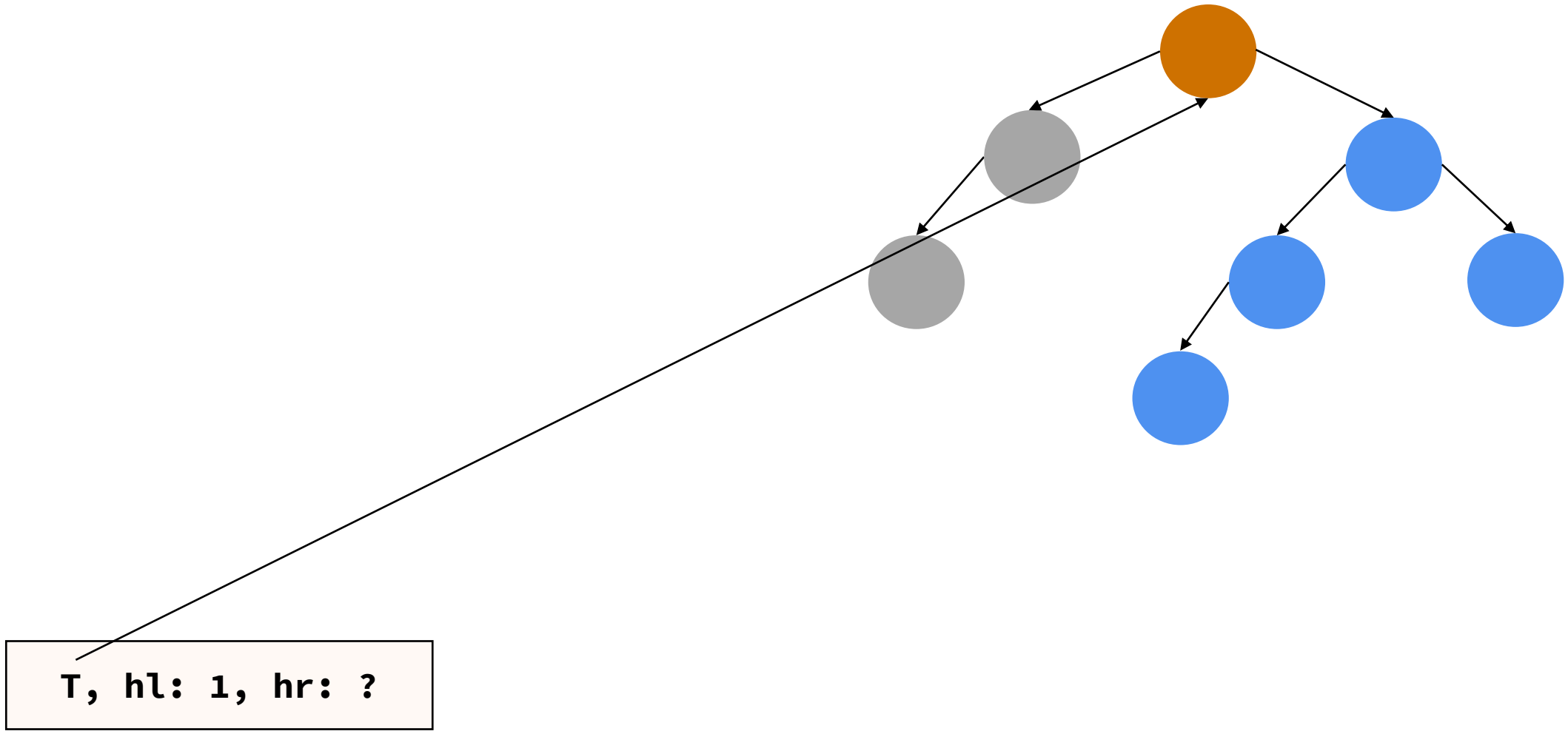
T: return 0
T, hl: 0, hr: ?
T, hl: ?, hr: ?

Calcolo dell'altezza: evoluzione del call stack

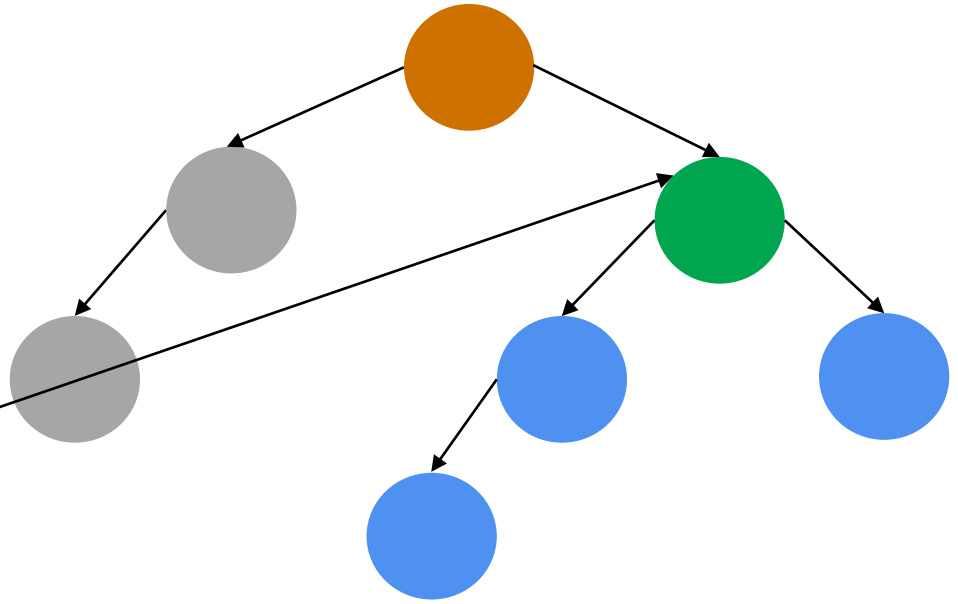


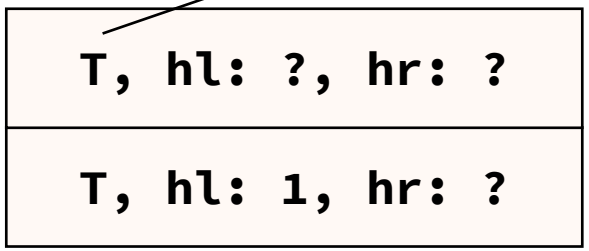


Calcolo dell'altezza: evoluzione del call stack

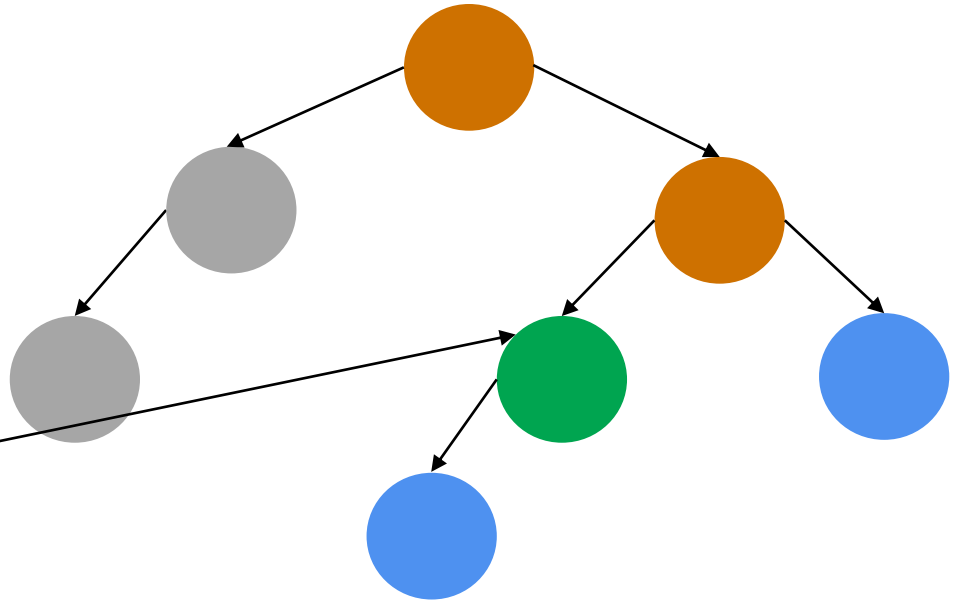


Calcolo dell'altezza: evoluzione del call stack



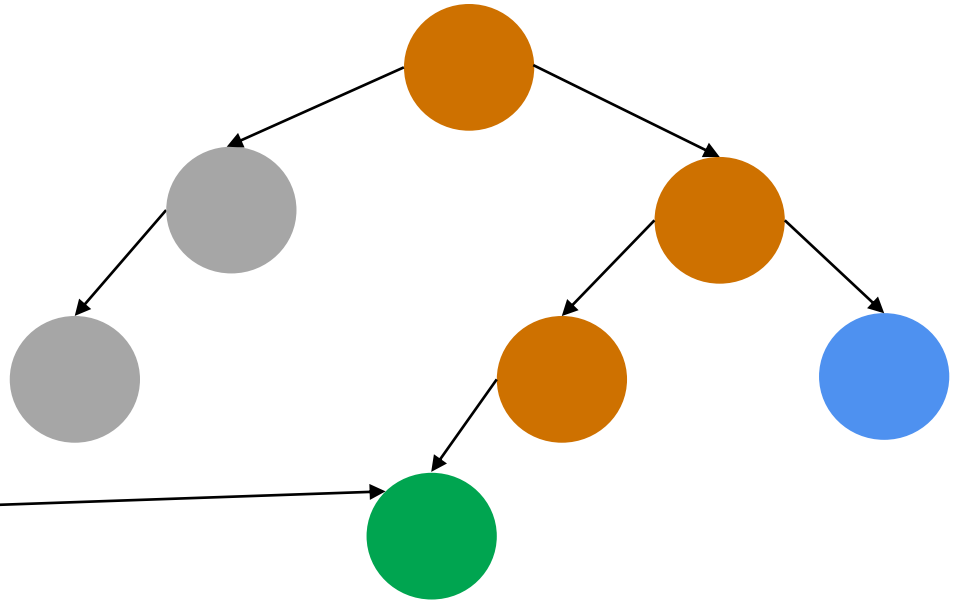


Calcolo dell'altezza: evoluzione del call stack



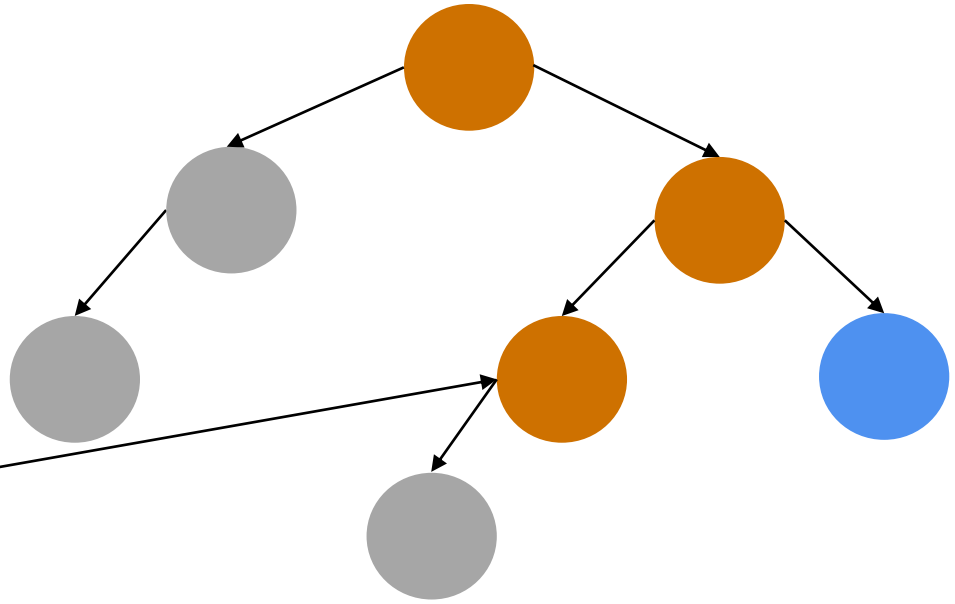
T, hl: ?, hr: ?
T, hl: ?, hr: ?
T, hl: 1, hr: ?

Calcolo dell'altezza: evoluzione del call stack



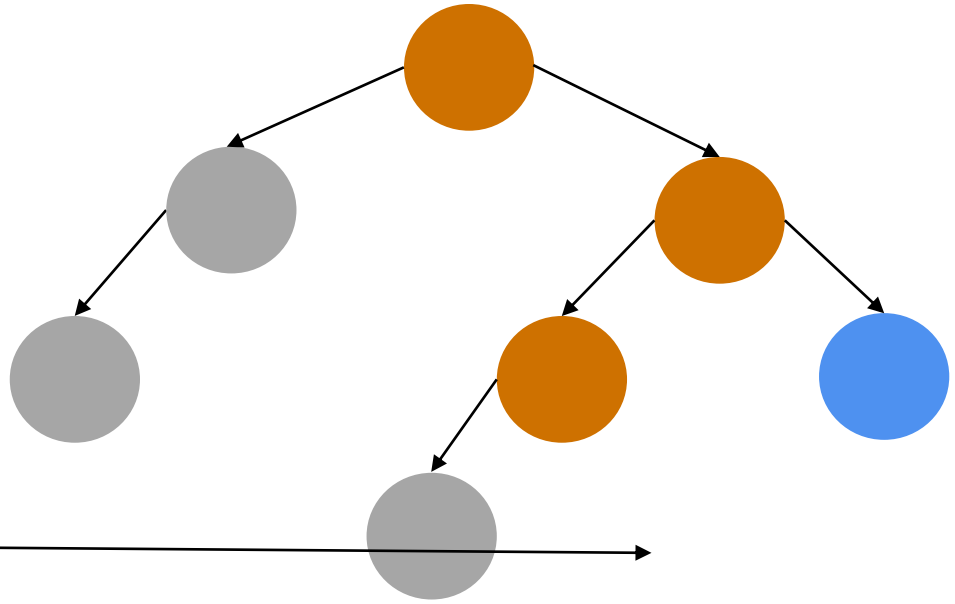
T, return 0
T, hl: ?, hr: ?
T, hl: ?, hr: ?
T, hl: 1, hr: ?

Calcolo dell'altezza: evoluzione del call stack



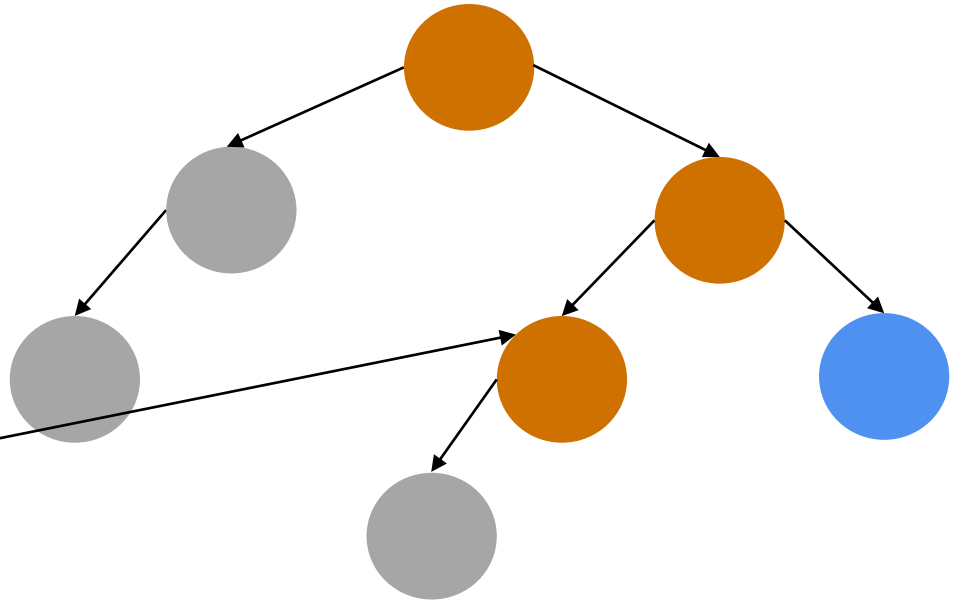
T, hl: 0, hr: ?
T, hl: ?, hr: ?
T, hl: 1, hr: ?

Calcolo dell'altezza: evoluzione del call stack



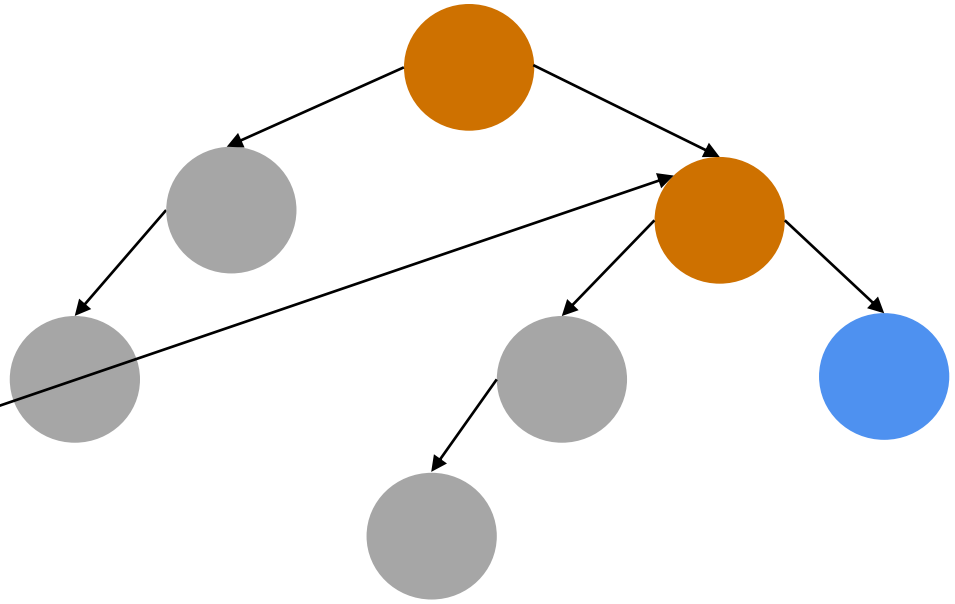
T, return 0
T, hl: 0, hr: ?
T, hl: ?, hr: ?
T, hl: 1, hr: ?

Calcolo dell'altezza: evoluzione del call stack



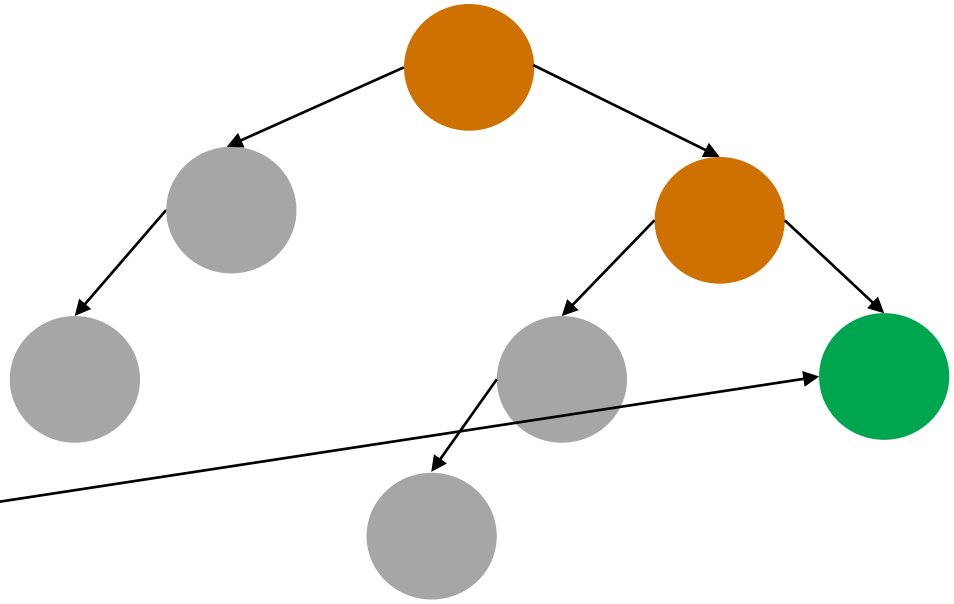
```
T, hl: 0, hr: 0, return max(0, 0) + 1: 1
T, hl: ?, hr: ?
T, hl: 1, hr: ?
```

Calcolo dell'altezza: evoluzione del call stack



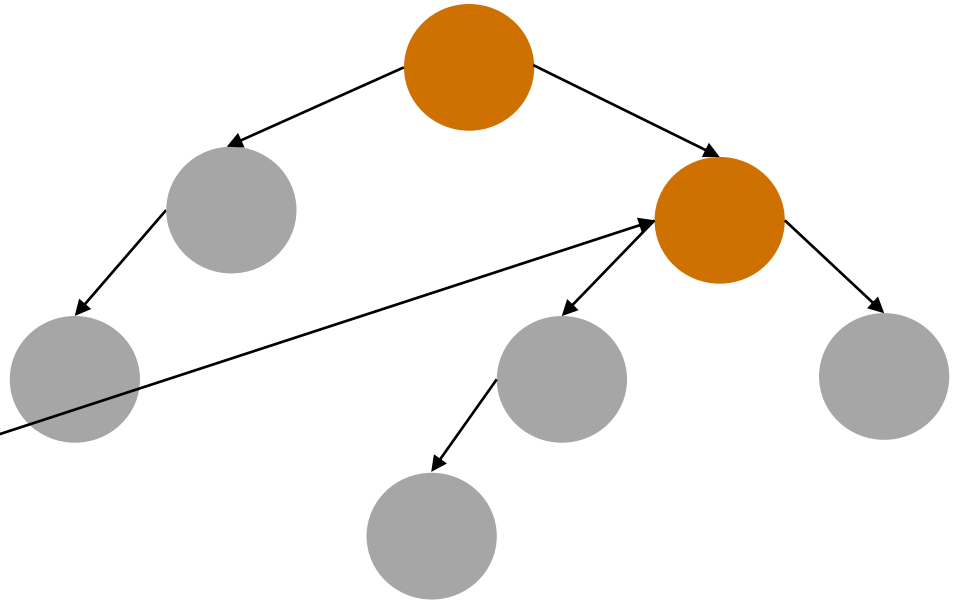
T, hl: 1, hr: ?
T, hl: 1, hr: ?

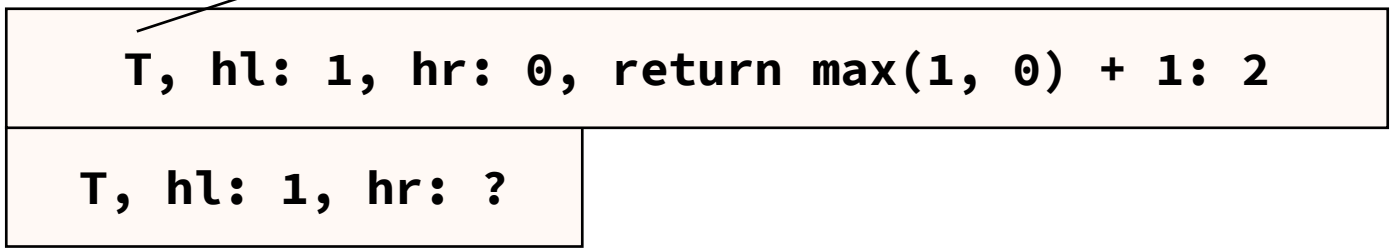
Calcolo dell'altezza: evoluzione del call stack



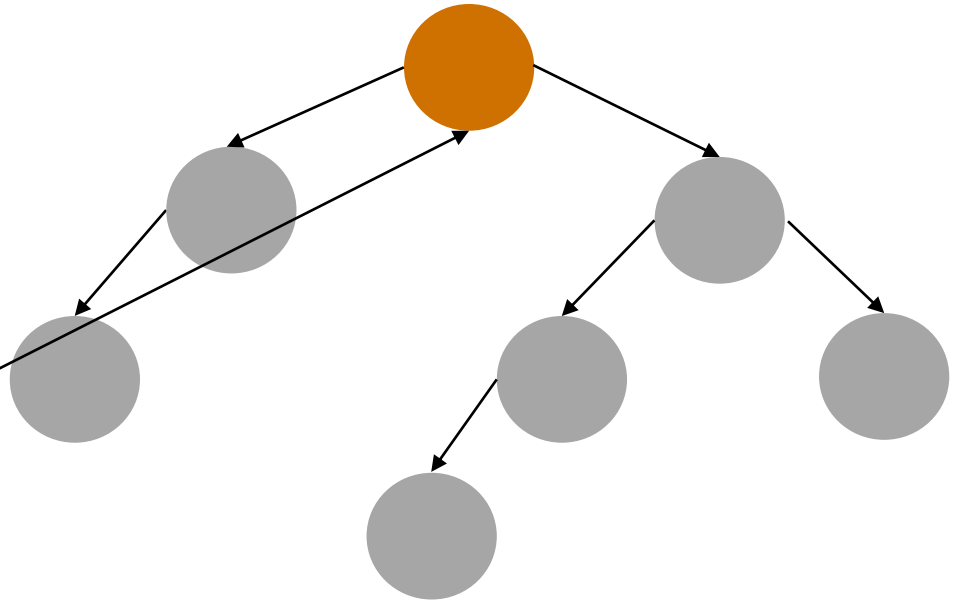
T, return 0
T, hl: 1, hr: ?
T, hl: 1, hr: ?

Calcolo dell'altezza: evoluzione del call stack



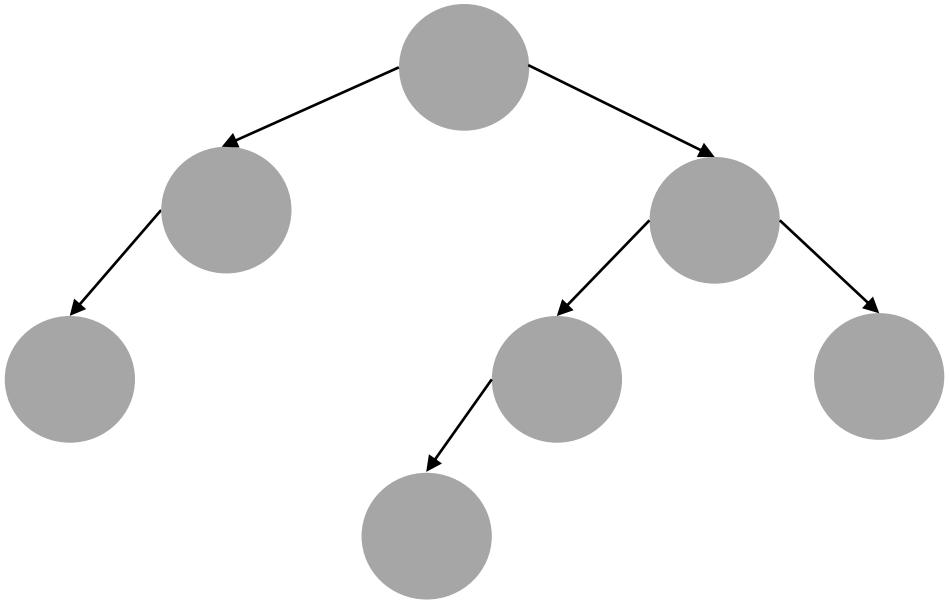


Calcolo dell'altezza: evoluzione del call stack



```
T, hl: 1, hr: 2, return max(1, 2) + 1: 3
```

Calcolo dell'altezza: evoluzione del call stack



Implementazione

- https://github.com/Cyofanni/high-school-cs-class/blob/main/C/data_structures/bin_tree.c