Liste concatenate (*linked lists*)

(https://github.com/Cyofanni/high-school-cs-class/tree/main/C/data_structures)

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

Liste concatenate (*linked lists*)

- Vogliamo costruire una struttura dati la cui dimensione cambia in base alle necessità del programma a *runtime*, immaginando una lettura da *standard input* di una sequenza di interi

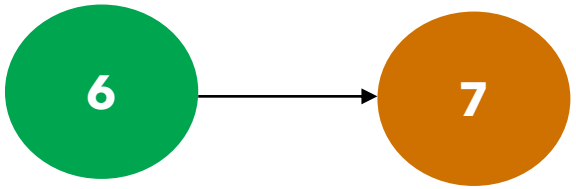
l'utente inserisce l'intero 6

viene allocato in memoria un oggetto contenente l'intero 6, conservando però gli oggetti precedenti in ordine di inserimento

Liste concatenate (*linked lists*)

l'utente inserisce l'intero 7

viene allocato in memoria un oggetto contenente l'intero 7, conservando però gli oggetti precedenti in ordine di inserimento



Liste concatenate (*linked lists*)

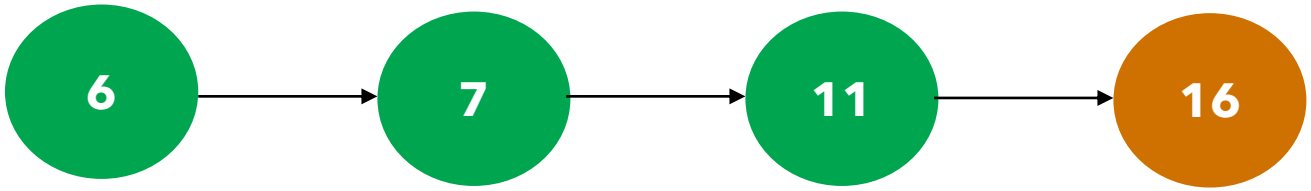
l'utente inserisce l'intero 11

viene allocato in memoria un oggetto contenente l'intero 11, conservando però gli oggetti precedenti in ordine di inserimento

Liste concatenate (*linked lists*)

l'utente inserisce l'intero 16

viene allocato in memoria un oggetto contenente l'intero 16, conservando però gli oggetti precedenti in ordine di inserimento



Liste concatenate (*linked lists*)

- Dobbiamo creare un nuovo tipo che rappresenti questi *nodi* in memoria

- Per creare un nuovo tipo, dobbiamo chiederci da cosa è caratterizzato il concetto che vogliamo rappresentare. Un nodo è caratterizzata da:
  - il dato che contiene, che chiameremo **chiave**. Nell'esempio si tratta di una chiave intera
  - una **freccia**, un qualcosa che permette di conservare l'ordine dei nodi e trovarli. Qualcosa che permetta di andare dal nodo con chiave 6 al nodo con chiave 7

Liste concatenate (*linked lists*)

- Un nodo deve avere al suo interno un **puntatore** al nodo successore
- E se un nodo non ha successori? Ossia, se è l'ultimo nodo della lista?
  - abbiamo già visto che esiste il puntatore che non punta a niente, il famoso puntatore `NULL`
- **NB**: i nodi di una linked list non sono necessariamente contigui in memoria, a differenza delle celle di un array

Liste concatenate (*linked lists*)

- Possiamo sfruttare i puntatori per creare la nostra struttura dati dinamica. Chiameremo la struttura ***singly linked list*** (*lista concatenata semplice*)
- Il concatenamento è *semplice* perché ciascun nodo punta al successore, ma non al precedente (vedremo anche il concatenamento doppio)

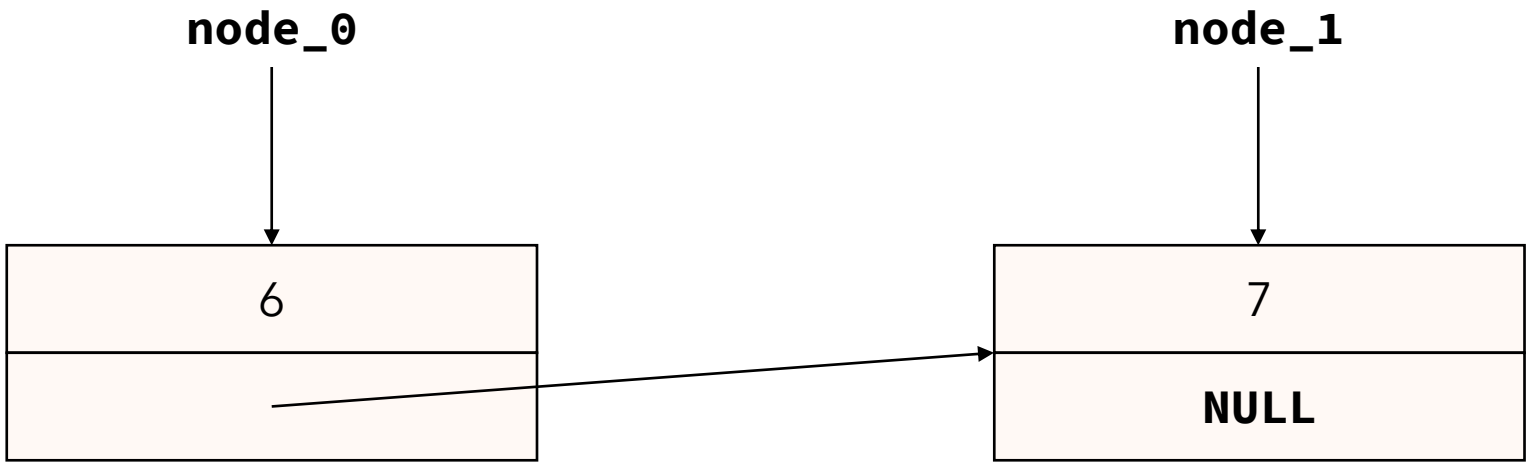
```
typedef struct list_node {
  int key;
  struct list_node* next;
} L_NODE;
```

Il nodo successore è a sua volta un nodo. Quindi deve avere necessariamente lo stesso tipo.
Il tipo che stiamo creando è dunque autoreferenziale, o ricorsivo

Liste concatenate (*linked lists*)

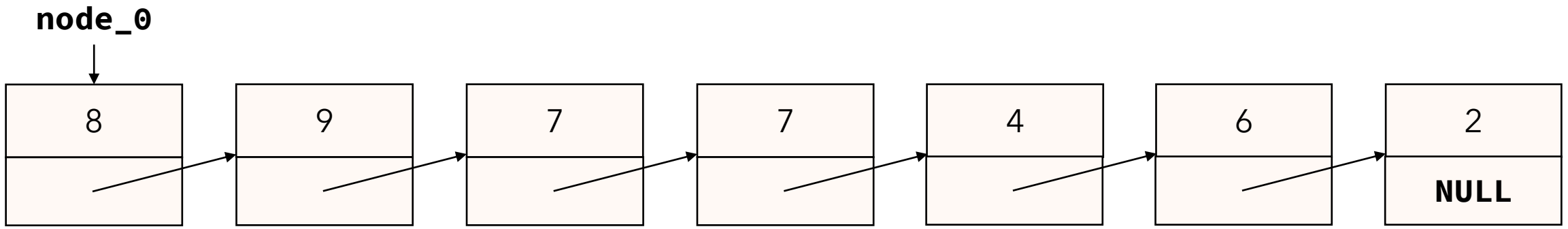
- Possiamo scegliere se allocare i nodi sullo stack o sull'heap
- Nei primi esempi li allocheremo sullo stack, poi sull'heap per avere maggior flessibilità
- Ripasso: allocazione dinamica in C
- Utilizziamo la memoria heap per poter creare nodi all'interno di funzioni e restituirli, cosa che sarebbe impossibile sullo stack

Liste concatenate (*linked lists*)



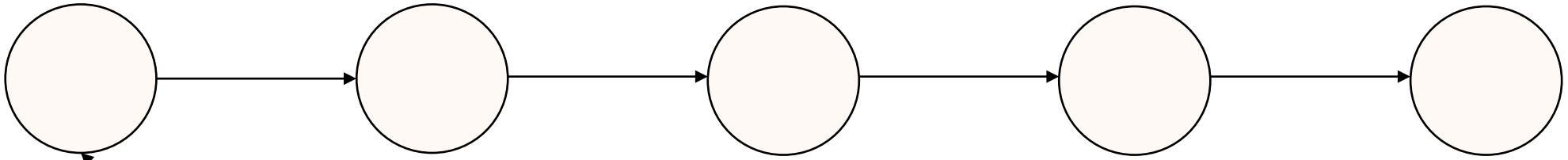
Liste concatenate (*linked lists*)

- Creare una lista concatenata di 7 nodi di chiavi intere (manualmente, uno alla volta). Prima sullo stack, poi sull'heap



- Hint: partire dall'ultimo nodo

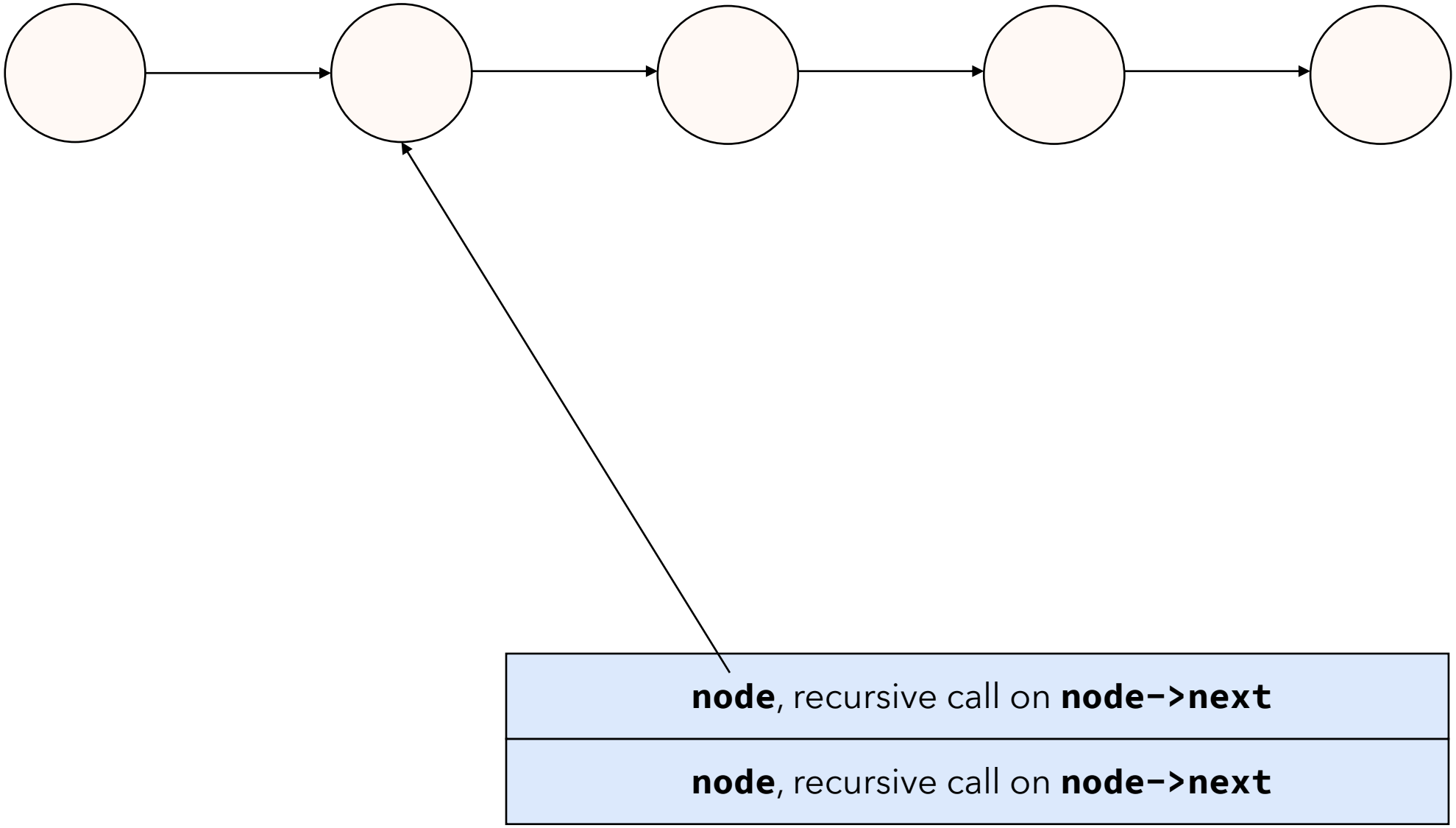
Calcolo ricorsivo della lunghezza



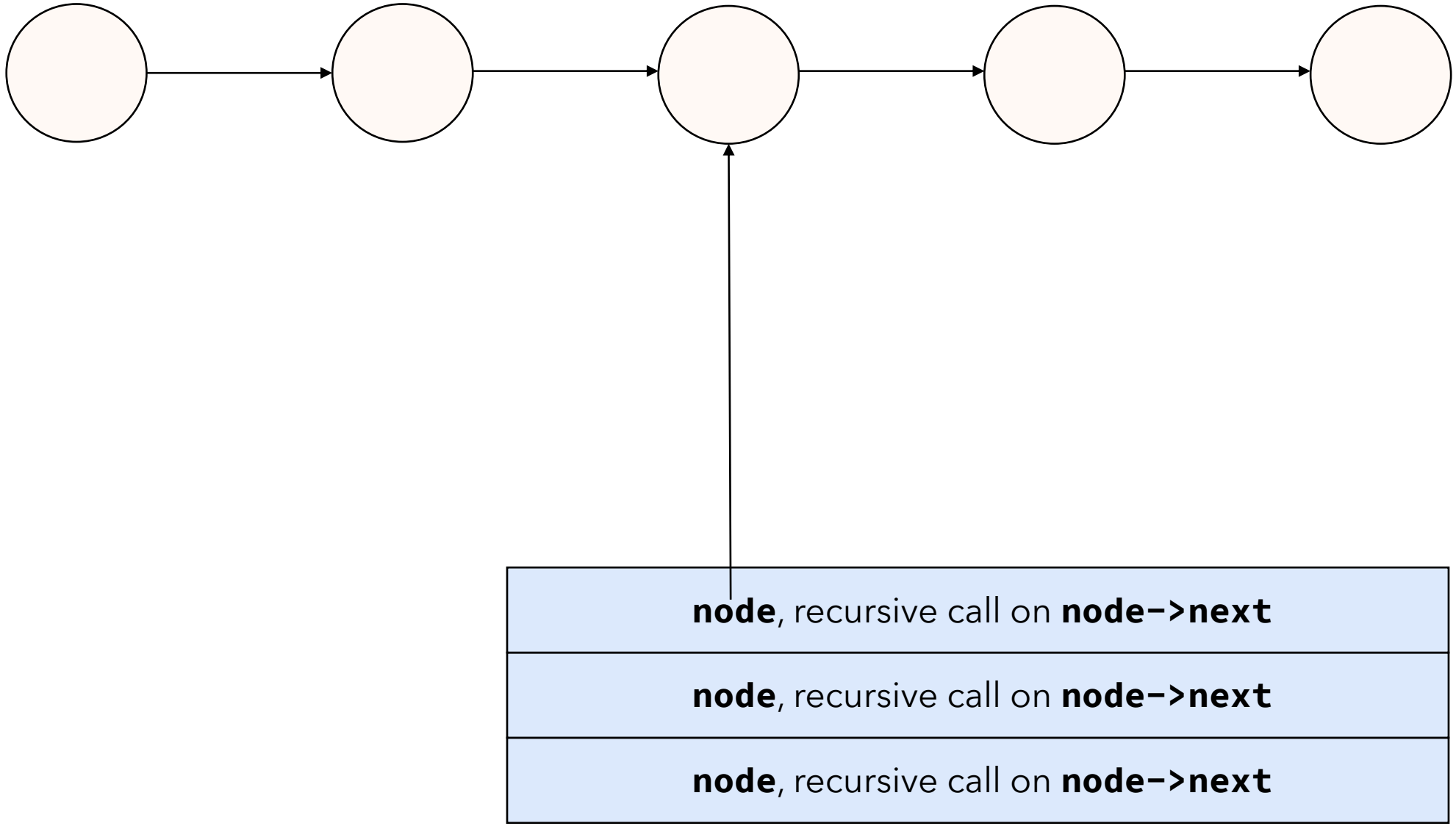
Scriviamo il codice dopo aver visto cosa dovrà succedere sullo stack.
Si tratterà di una funzione ricorsiva molto simile ad altre che abbiamo visto e rivisto

node, recursive call on **node->next**

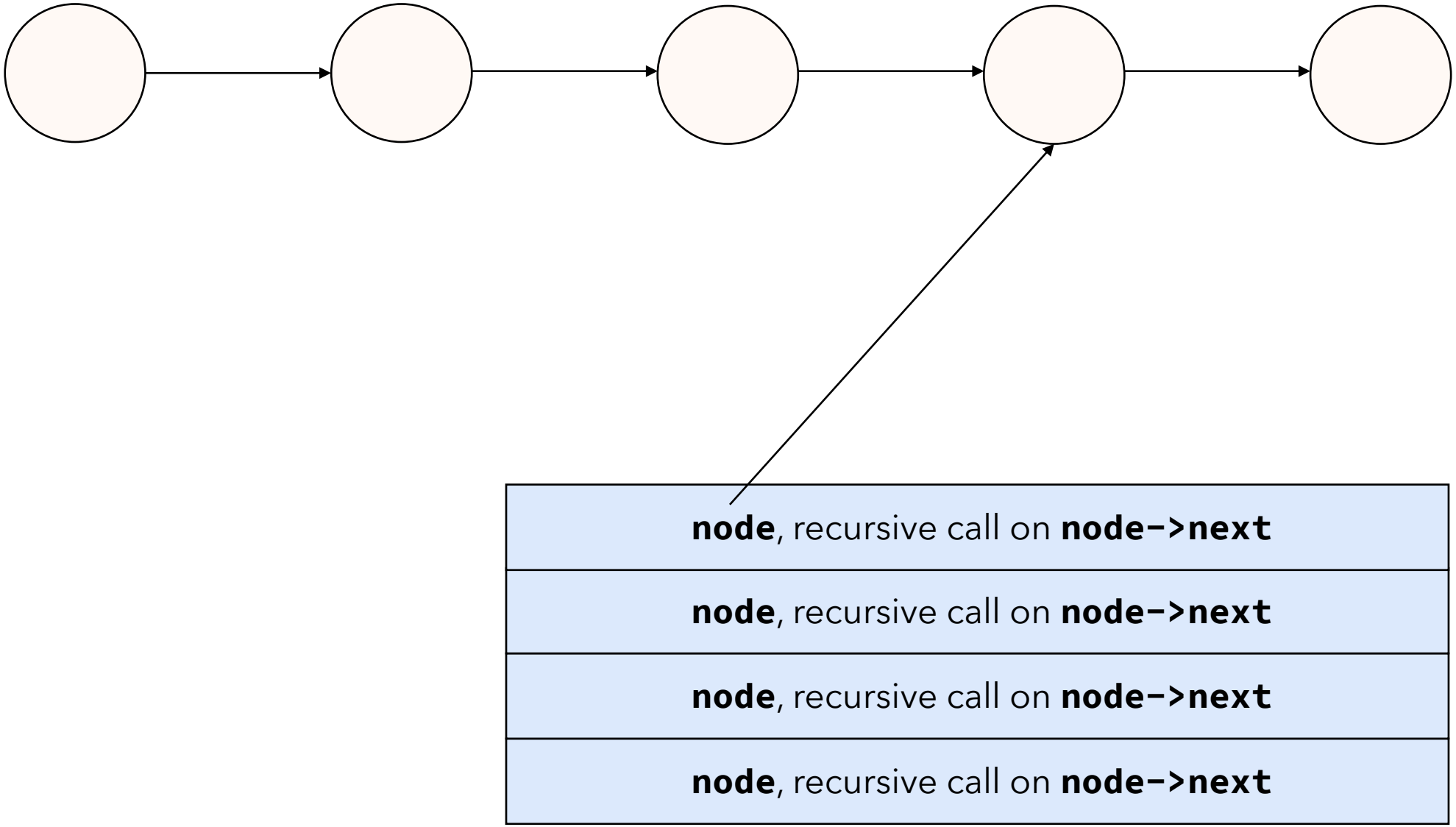
Calcolo ricorsivo della lunghezza



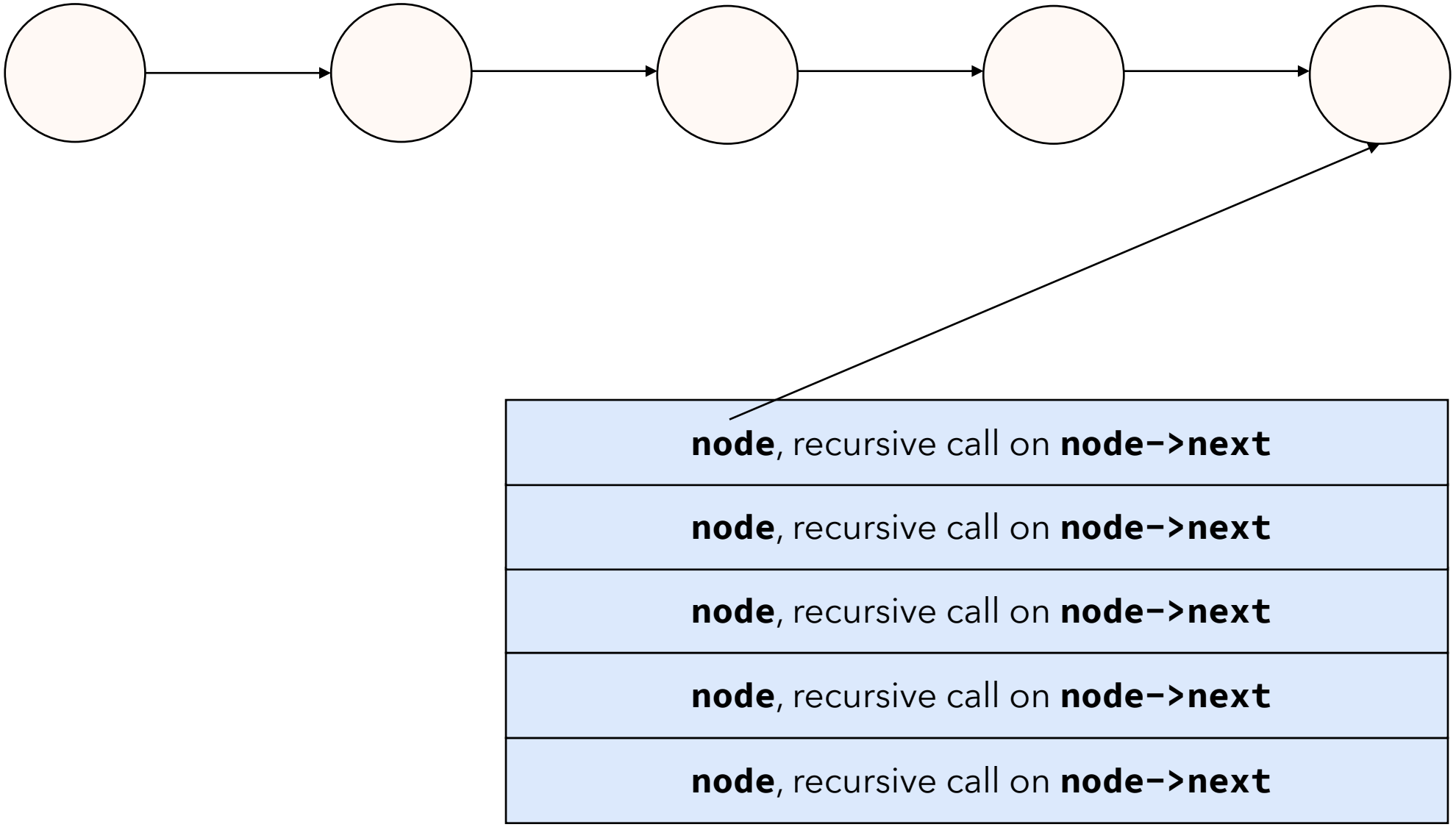
Calcolo ricorsivo della lunghezza



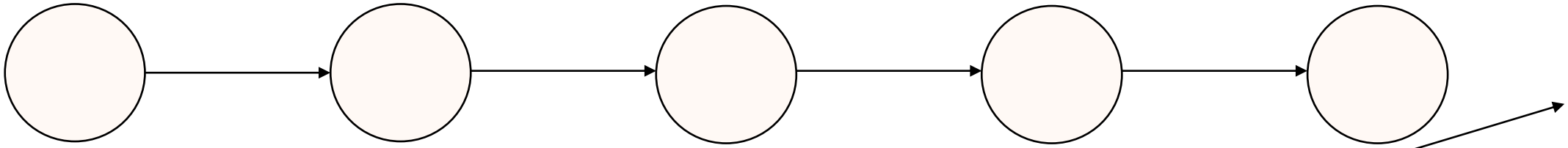
Calcolo ricorsivo della lunghezza

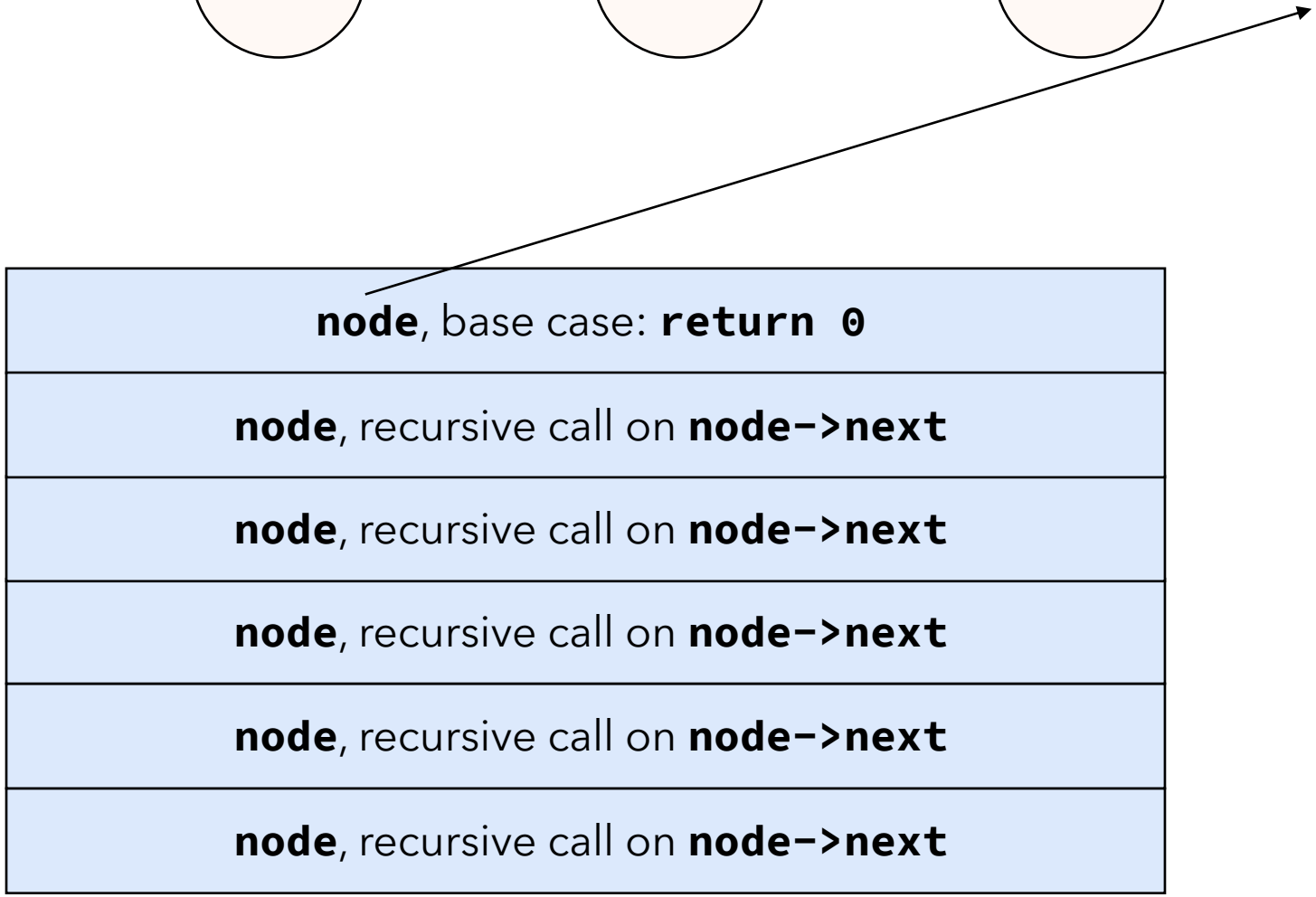


Calcolo ricorsivo della lunghezza

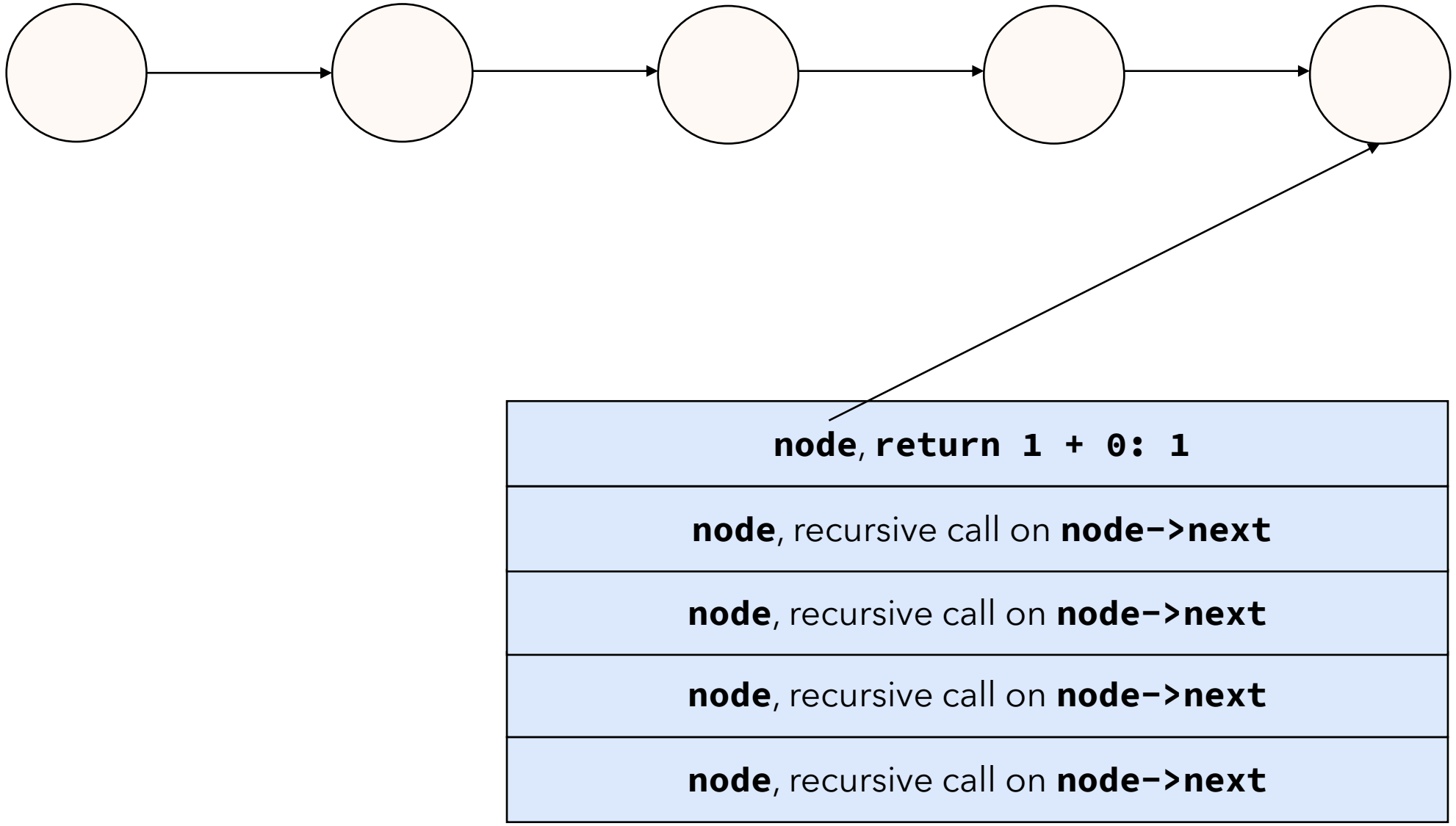


Calcolo ricorsivo della lunghezza

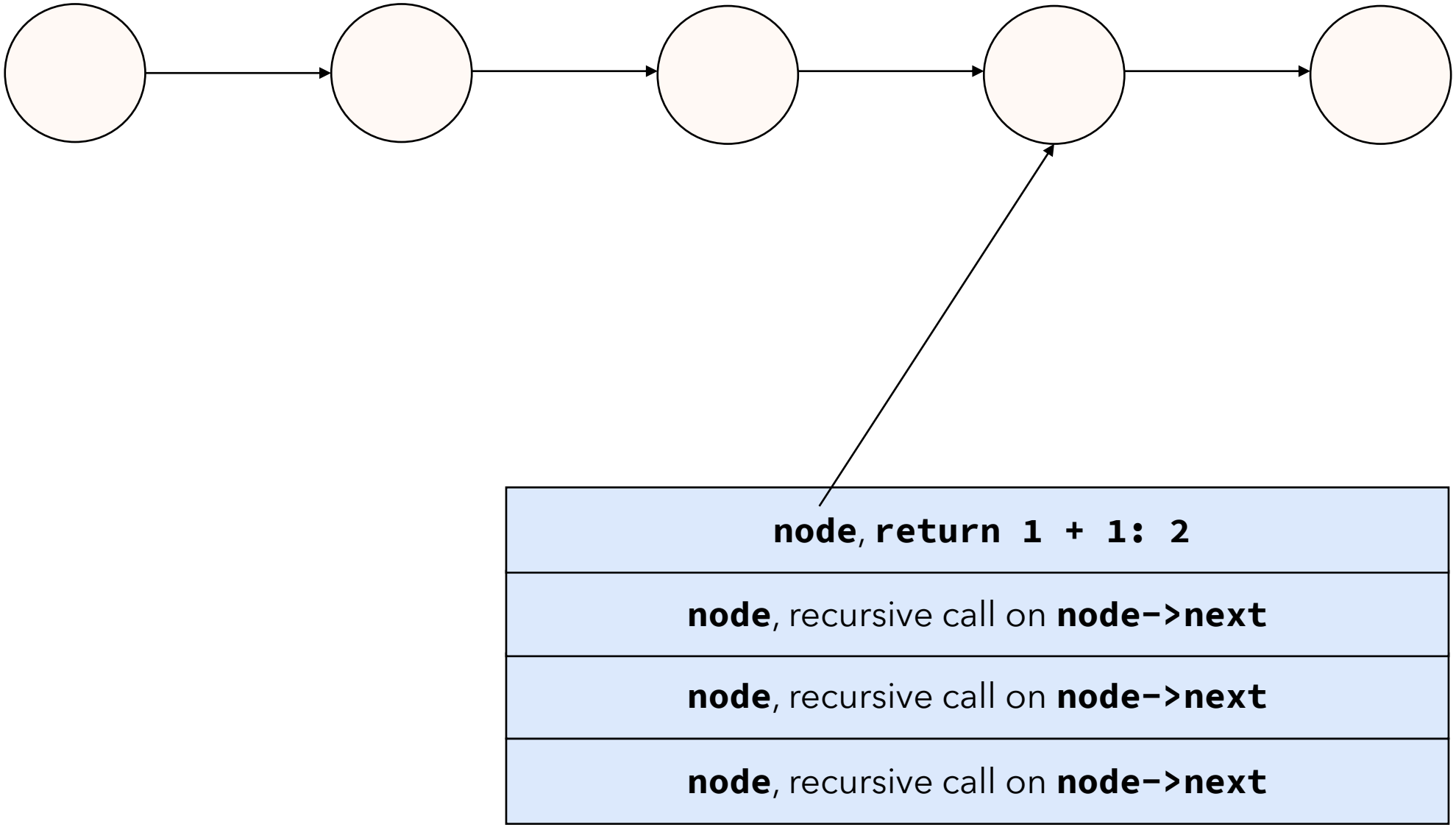




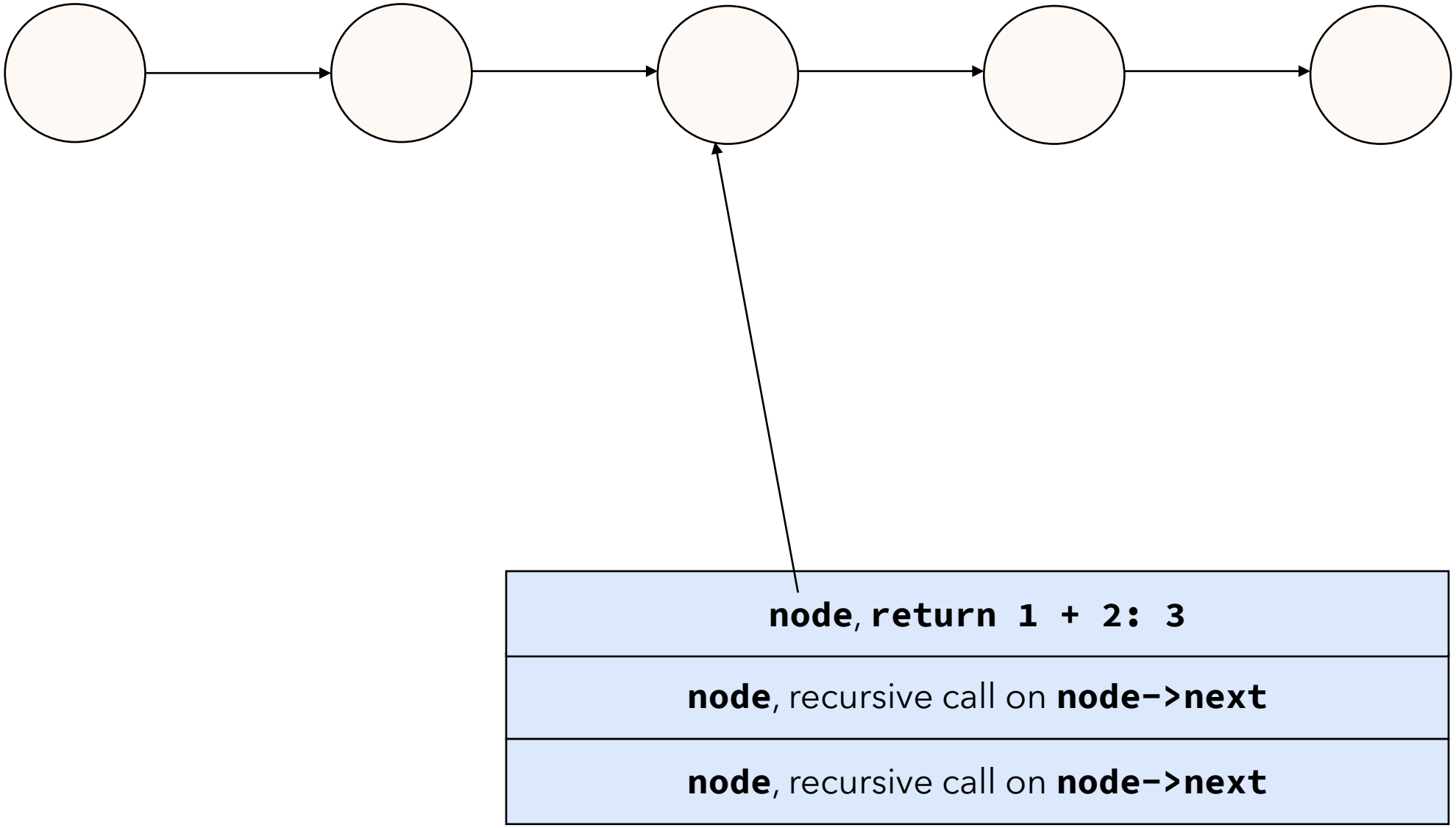
Calcolo ricorsivo della lunghezza



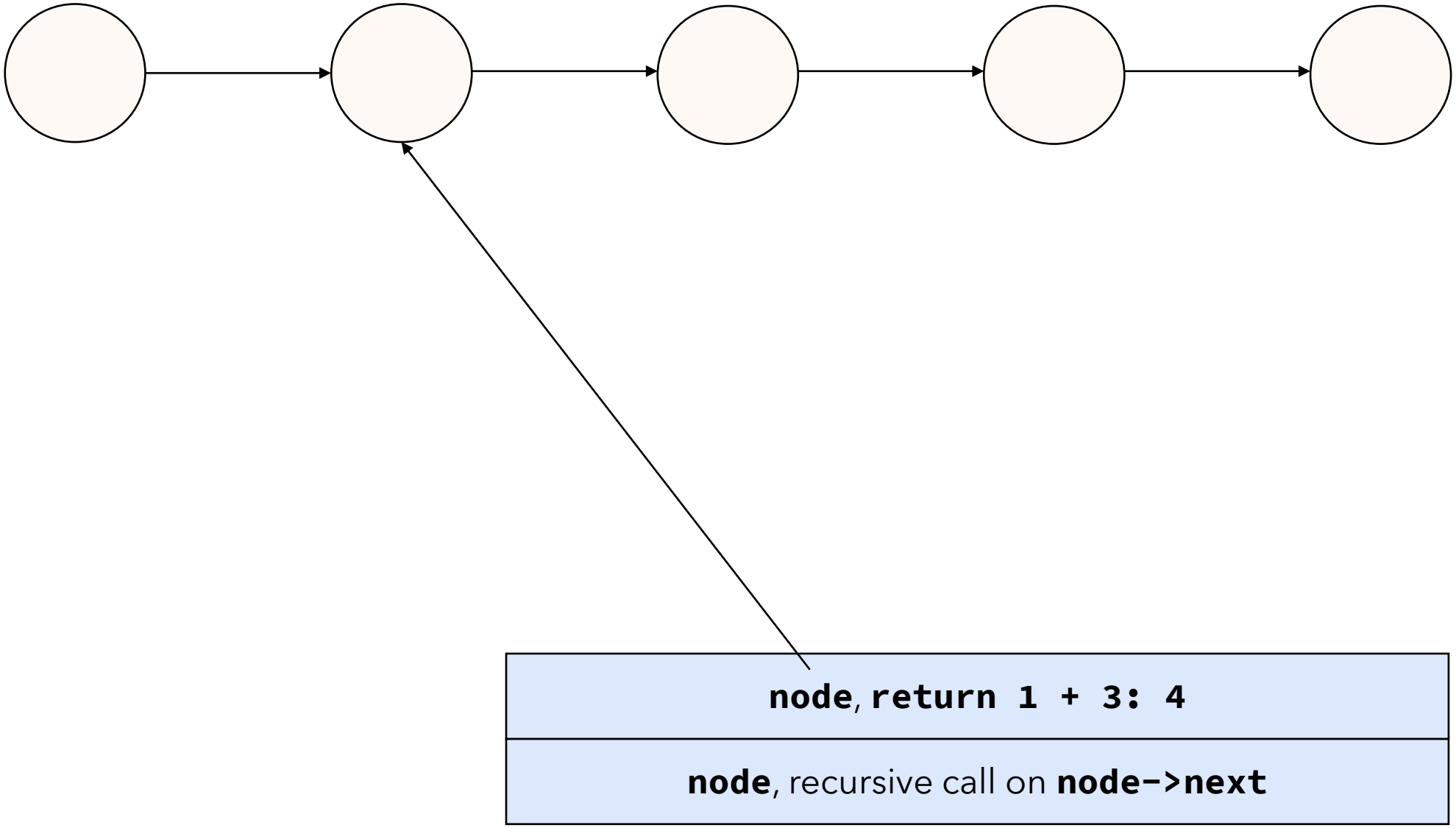
Calcolo ricorsivo della lunghezza



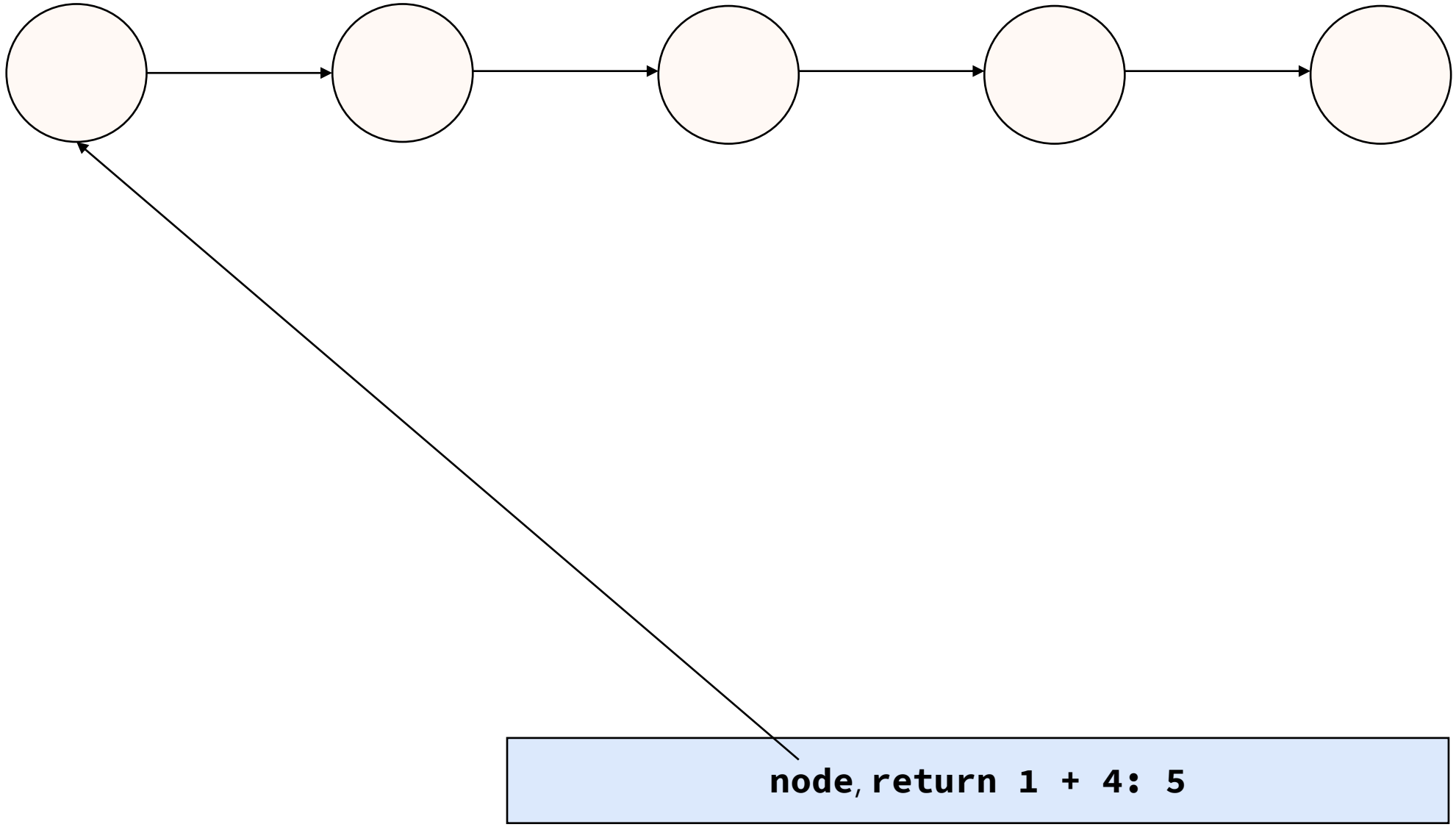
Calcolo ricorsivo della lunghezza



Calcolo ricorsivo della lunghezza



Calcolo ricorsivo della lunghezza

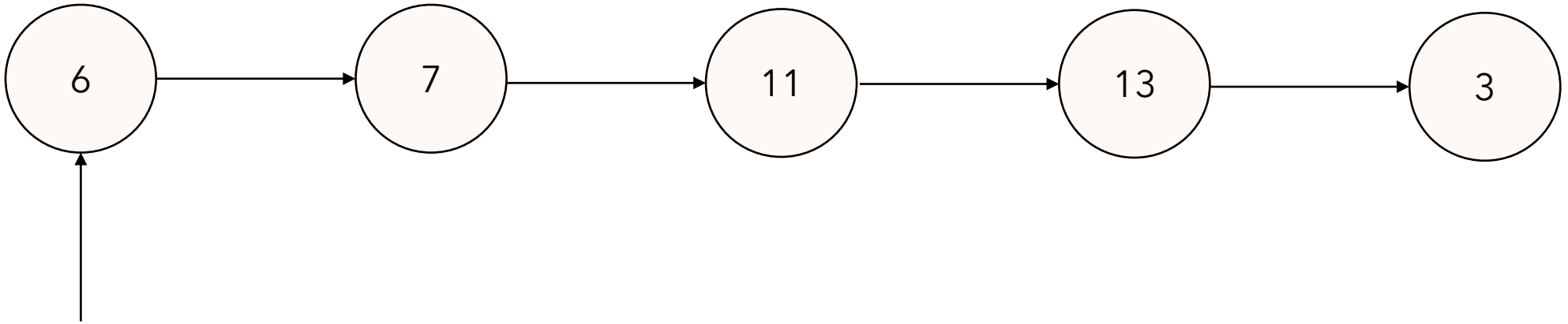


Calcolo ricorsivo della lunghezza

Pseudocode

```
length(node):
  if node is null:
    return 0
  return 1 + length(node.next)
```

Ricerca lineare iterativa

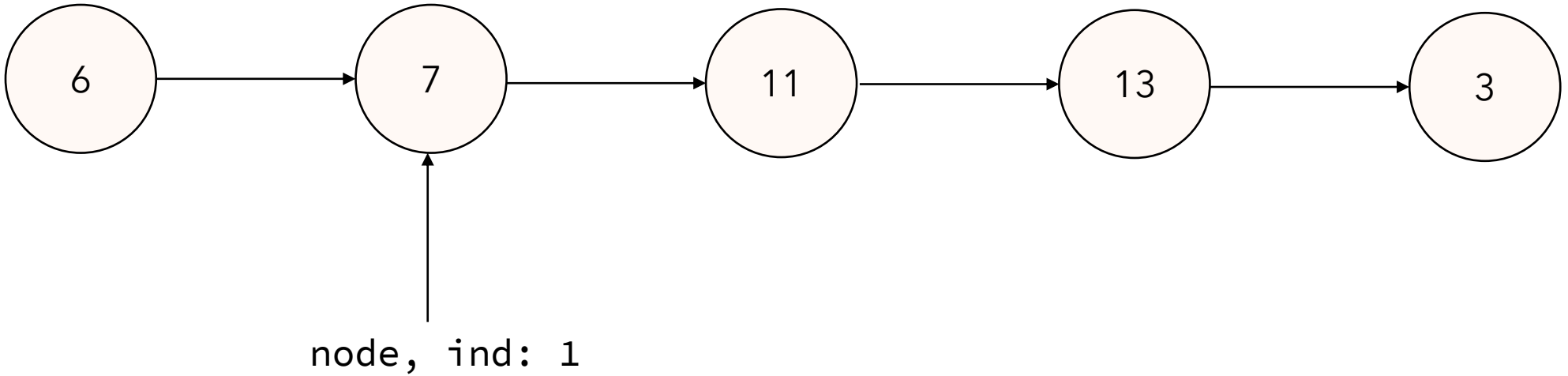


key_position: -1

cerchiamo la prima occorrenza della chiave 11 e restituiamo l'indice del nodo che la contiene (diamo al primo nodo indice 0)

prima di iniziare a cercare, diciamo che la chiave si trova nella posizione fittizia -1

Ricerca lineare iterativa

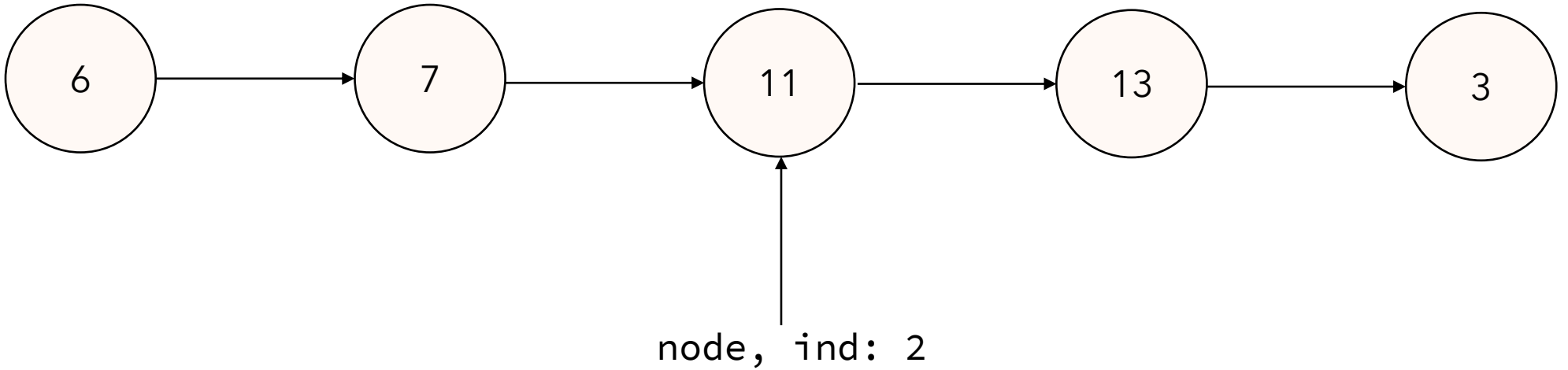



```
key_position: -1
```

cerchiamo la prima occorrenza della chiave 11 e restituiamo l'indice del nodo che la contiene (diamo al primo nodo indice 0)

prima di iniziare a cercare, diciamo che la chiave si trova nella posizione fittizia -1

Ricerca lineare iterativa



key_position: 2

prima di iniziare a cercare, diciamo che la chiave si trova nella posizione fittizia -1

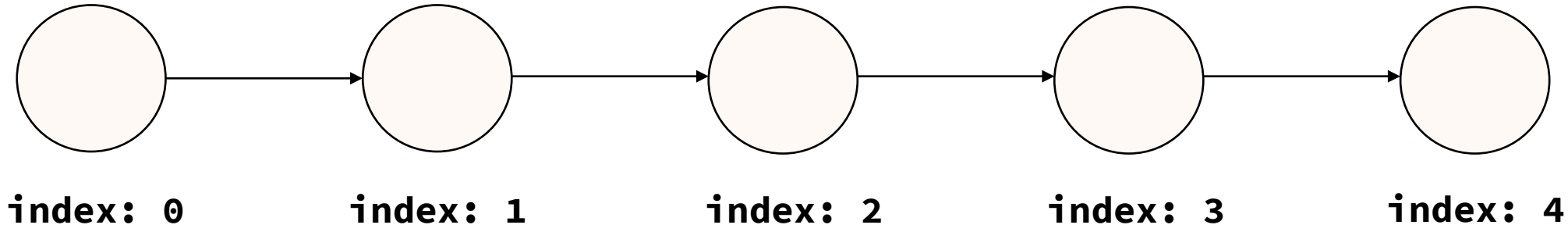
cerchiamo la prima occorrenza della chiave 11 e restituiamo l'indice del nodo che la contiene (diamo al primo nodo indice 0)

cosa fare quando la chiave non è presente nella lista?

Accesso sequenziale iterativo

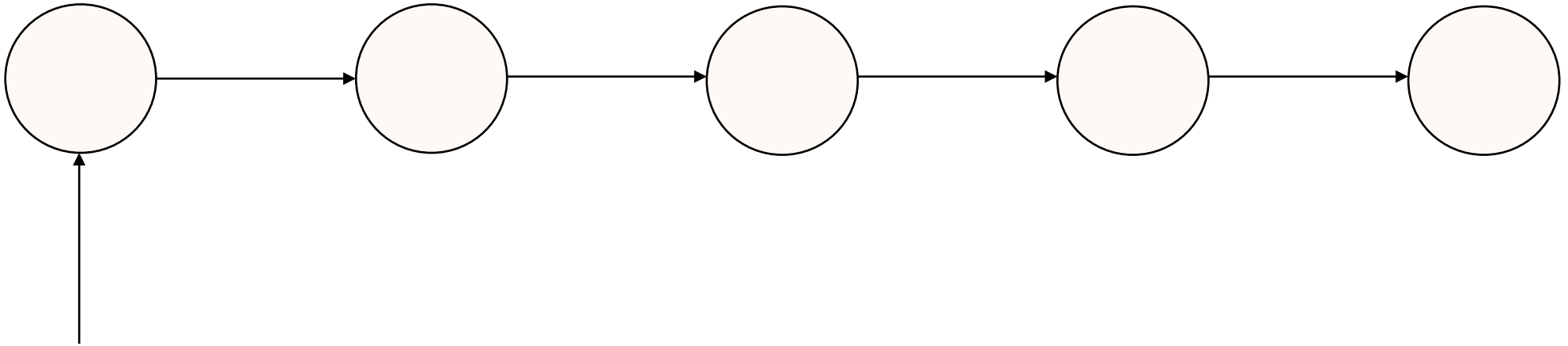
- Scriviamo un algoritmo iterativo che restituisca il puntatore al nodo di indice n
- Indicizziamo le liste, come gli array, partendo da 0
- Se viene richiesto un indice *illegale* (negativo, o maggiore della dimensione della lista - 1), restituiamo **`NULL`**
- L'accesso è **sequenziale**: per accedere all'n-esimo nodo, bisogna scorrere tutti i nodi precedenti (l'accesso per gli array invece è **diretto** - ***random-access***)
- Si dice che l'accesso avviene in tempo lineare sulla dimensione della lista:
  - nel caso peggiore (*worst case*, ossia accesso all'ultimo nodo) bisogna scorrerla tutta

Accesso sequenziale iterativo



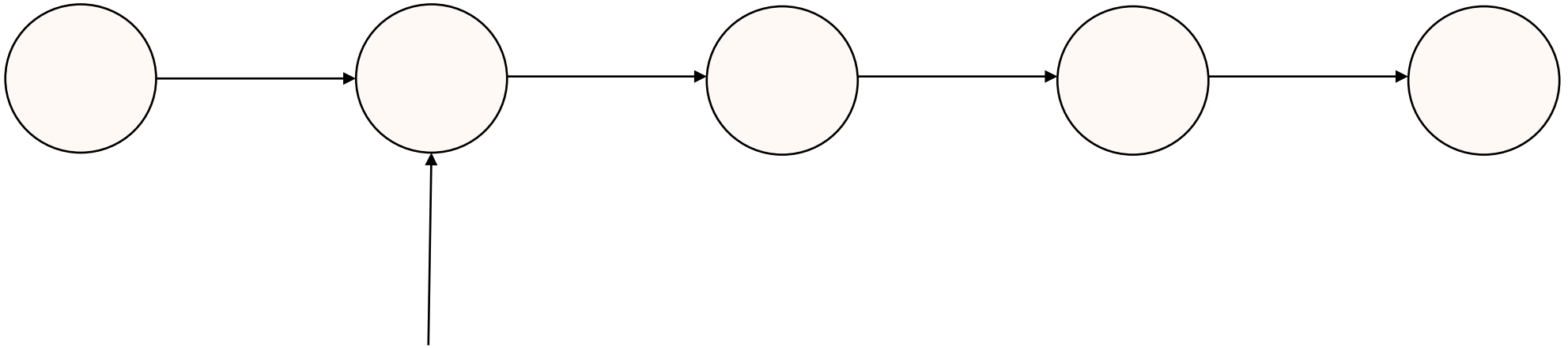
eseguiamo graficamente la procedura per ottenere il puntatore al nodo di indice 3. Inizializziamo un contatore a 0

Accesso sequenziale iterativo



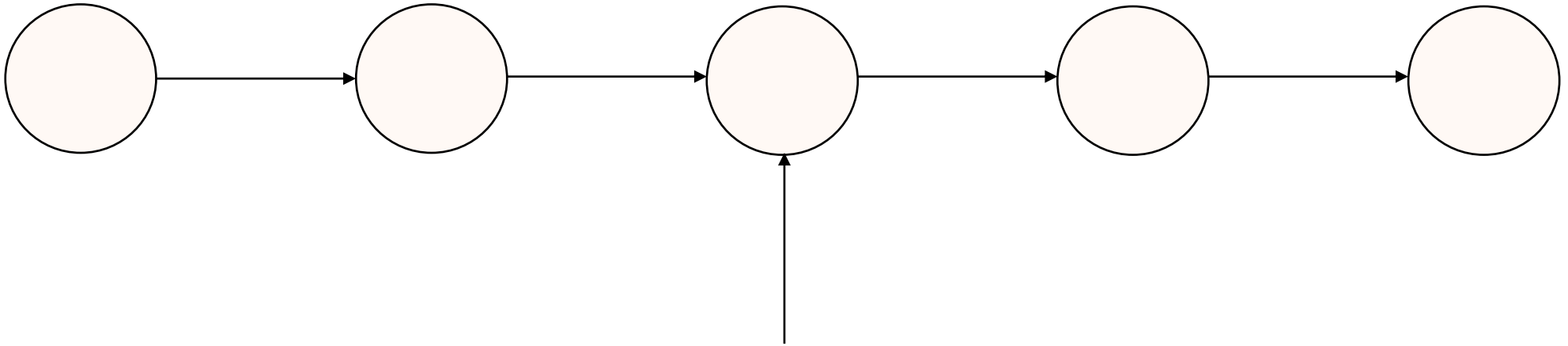
```
count: 0
```

Accesso sequenziale iterativo



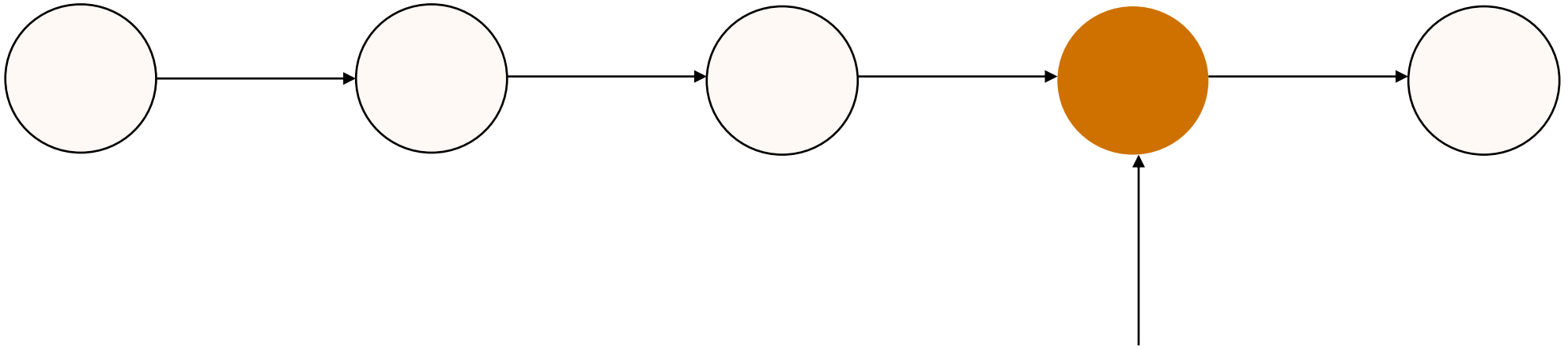
```
count: 1
```

Accesso sequenziale iterativo



```
count: 2
```

Accesso sequenziale iterativo



cosa fare quando l'indice richiesto è maggiore dell'ultimo indice legale?

```
count: 3
```

Algoritmi notevoli sulle liste

- Alcuni algoritmi notevoli da implementare come esercizi a casa e a scuola:
  - somma delle chiavi
  - inserimento di un nodo in testa
  - inserimento di un nodo in coda
  - inserimento di un nodo in posizione n
  - cancellazione del nodo in posizione n
  - creazione del tipo che rappresenta il nodo di una lista doppiamente concatenata

Da vedere a casa

- Arrays vs Linked Lists – Computerphile
- Linked Lists - Computerphile