

Java: strutture dati dinamiche (liste concatenate)

Liceo G.B. Brocchi
Classi quarte Scientifico - opzione scienze applicate
Bassano del Grappa, Ottobre 2022
Prof. Giovanni Mazzocchin

I limiti degli array *fixed-size*

```
public static void main(String[] args) {  
    int[] v = new int[5];  
    Scanner in = new Scanner(System.in);
```

```
    int num = 0;  
    int index = 0;
```

```
    System.out.print("Enter an integer number: ");  
    num = in.nextInt();  
    while (num != -1) {  
        v[index] = num;  
        printArray(v, index);  
        System.out.print("Enter an integer number: ");  
        num = in.nextInt();  
        index++;  
    }
```

```
}
```

Il programma riceve da stdin un intero fintantoché l'intero inserito è diverso da -1. L'intero in input viene inserito nell'array v, che ha dimensione 5.

metodo definito nella slide seguente



I limiti degli array *fixed-size*

```
private static void printArray (int[] a, int lastFilledIndex) {  
    System.out.println("Array's length is: " + a.length);  
    System.out.print("Array's content is (default values preceded by *): ");  
  
    for (int i = 0; i < a.length; i++) {  
        if (i <= lastFilledIndex) {  
            System.out.print(a[i] + " ");  
        }  
        else {  
            System.out.print("*" + a[i] + " ");  
        }  
    }  
    System.out.println();  
}
```

I limiti degli array *fixed-size*

```
App
io
Enter an integer number: 8
Array's length is: 5
Array's content is (default values preceded by *): 8 *0 *0 *0 *0
va: str
Enter an integer number: 6
Array's length is: 5
Array's content is (default values preceded by *): 8 6 *0 *0 *0
Classi quart
Base
Enter an integer number: 9
Array's length is: 5
Array's content is (default values preceded by *): 8 6 9 *0 *0
oni
Enter an integer number: 1
Array's length is: 5
Array's content is (default values preceded by *): 8 6 9 1 *0
gli array
Enter an integer number: 6
Array's length is: 5
Array's content is (default values preceded by *): 8 6 9 1 6
Enter an integer number: 7
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 5 out of bounds for length 5
at Lecture3.main(Lecture3.java:29)
```

Un array v ha una dimensione fissa, stabilita al momento della sua inizializzazione. Se si prova ad accedere ad un indice superiore a v.length - 1 viene lanciata un'eccezione detta *ArrayIndexOutOfBoundsException*.

Le liste concatenate (*linked list*)

Vorremmo delle strutture dati la cui dimensione cresce in base alle necessità del programma durante la sua esecuzione;

Nell'esempio precedente con l'array, sarebbe stato utile se l'array si fosse «ridimensionato» dopo l'inserimento del quinto elemento;

Perché non allocare semplicemente un array di dimensione molto grande e chiudere qui la questione?

```
int[] v = new int[10000];
```

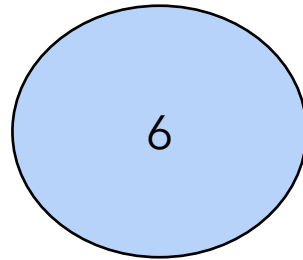


Le liste concatenate (*linked list*)

- L'ideale sarebbe avere una struttura dati la cui dimensione cambia in base alle necessità del programma a runtime, in questo modo:

L'utente inserisce l'intero 6:

Viene creato in memoria un oggetto che contiene l'intero 6:

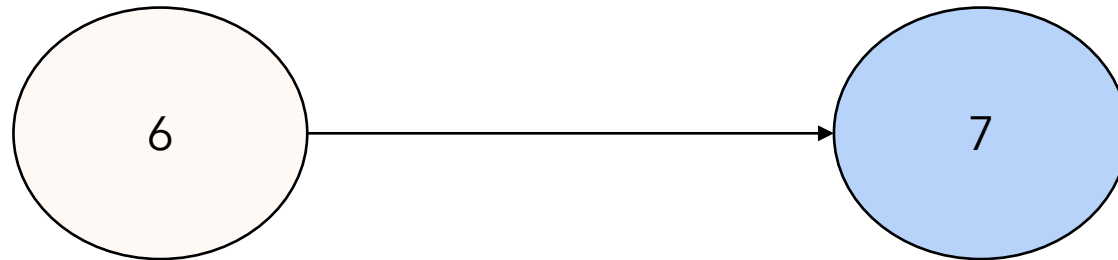


Le liste concatenate (*linked list*)

- L'ideale sarebbe avere una struttura dati la cui dimensione cambia in base alle necessità del programma a runtime, in questo modo:

L'utente inserisce l'intero 7:

Viene creato in memoria un oggetto che contiene l'intero 7, conservando però i precedenti in ordine di inserimento:

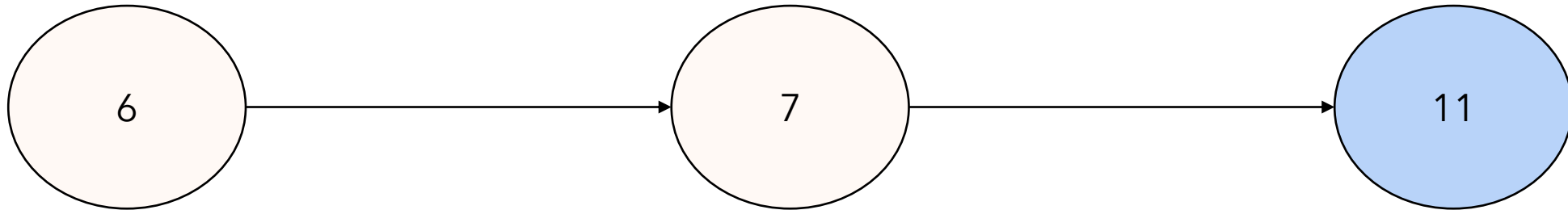


Le liste concatenate (*linked list*)

- L'ideale sarebbe avere una struttura dati la cui dimensione cambia in base alle necessità del programma a runtime, in questo modo:

L'utente inserisce l'intero 11:

Viene creato in memoria un oggetto che rappresenta l'intero 11, conservando però i precedenti in ordine di inserimento:

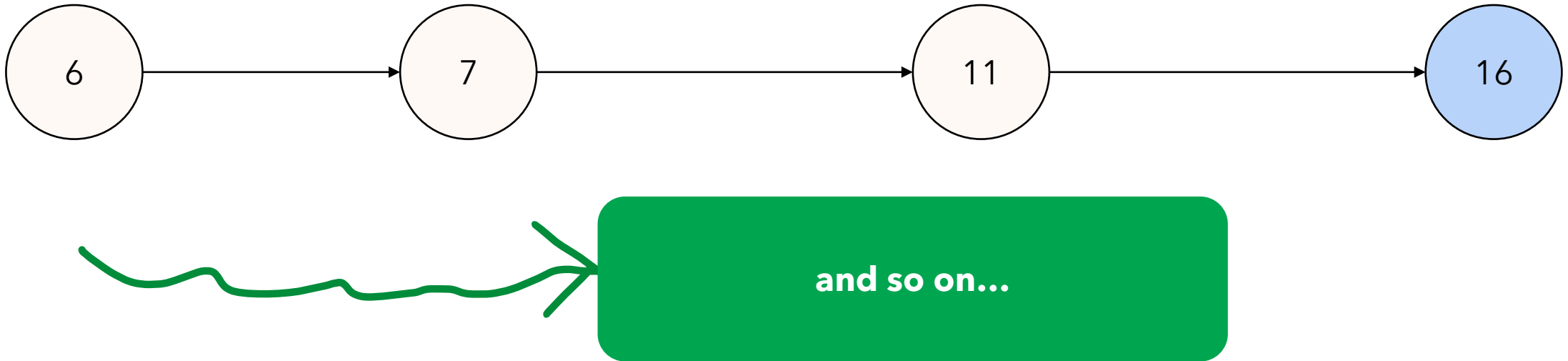


Le liste concatenate (*linked list*)

- L'ideale sarebbe avere una struttura dati la cui dimensione cambia in base alle necessità del programma a runtime, in questo modo:

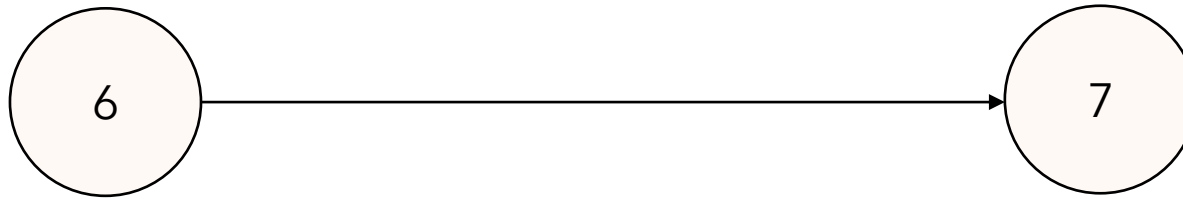
L'utente inserisce l'intero 16:

Viene creato in memoria un oggetto che rappresenta l'intero 16, conservando però i precedenti in ordine di inserimento:



Le liste concatenate (*linked list*)

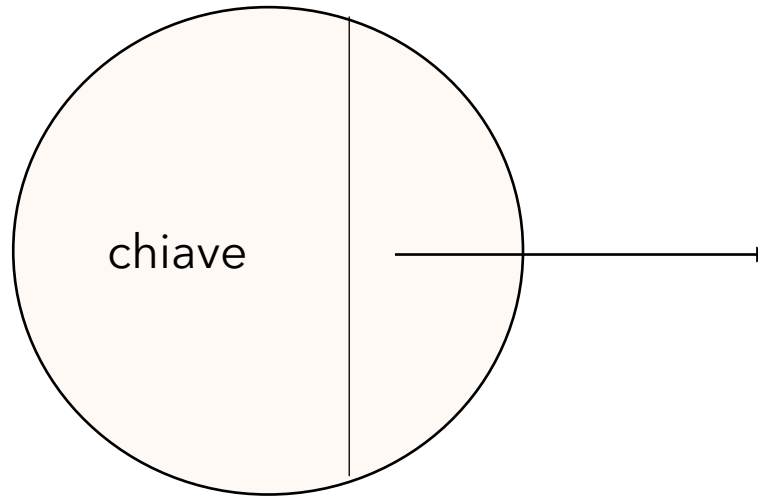
- Dobbiamo creare un nuovo tipo che rappresenti queste «palle» di memoria



- Per creare un nuovo tipo dobbiamo chiederci da cosa è caratterizzato il concetto che vogliamo rappresentare nel linguaggio. Una palla 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 delle palle e ricercarle. Qualcosa che permetta di andare dalla palla contenente 6 alla palla contenente 7.

Le liste concatenate (*linked list*)

- Dobbiamo creare un nuovo tipo che rappresenti queste «palle» di memoria



- Per creare un nuovo tipo dobbiamo chiederci da cosa è caratterizzato il concetto che vogliamo rappresentare nel linguaggio. Una palla è caratterizzata da:
 - il dato che contiene, che chiameremo «chiave». Nell'esempio si trattava di una chiave intera;
 - una freccia, un qualcosa che permette di conservare l'ordine delle palle e ricercarle. Qualcosa che permetta di andare dalla palla contenente 6 alla palla contenente 7.

Le liste concatenate (*linked list*)

- Dobbiamo creare un nuovo tipo che rappresenti queste «palle» di memoria



- Per creare un nuovo tipo dobbiamo chiederci da cosa è caratterizzato il concetto che vogliamo rappresentare nel linguaggio. Una palla è caratterizzata da:
 - il dato che contiene, che chiameremo «chiave». Nell'esempio si trattava di una chiave intera;
 - una freccia, un qualcosa che permette di conservare l'ordine delle palle e ricercarle. Qualcosa che permetta di andare dalla palla contenente 6 alla palla contenente 7.

Le liste concatenate (*linked list*)

- Dobbiamo creare un nuovo tipo che rappresenti queste «palle» di memoria



- Per creare un nuovo tipo dobbiamo chiederci da cosa è caratterizzato il concetto che vogliamo rappresentare nel linguaggio. Una palla è caratterizzata da:
 - il dato che contiene, che chiameremo «chiave». Nell'esempio si trattava di una chiave intera;
 - una freccia, un qualcosa che permette di conservare l'ordine delle palle e ricercarle. Qualcosa che permetta di andare dalla palla contenente 6 alla palla contenente 7
 - Che tipo deve avere la proprietà «freccia»?


Le liste concatenate (*linked list*)

- Dobbiamo creare un nuovo tipo che rappresenti queste «palle» di memoria
- Abbiamo già usato le frecce per rappresentare un concetto fondamentale in Java: i **riferimenti**!
- D'ora in poi chiameremo le palle di memoria **nodi**
- Diciamo che un nodo deve avere al suo interno un **riferimento** al nodo successore
- E se un nodo non ha successori? Ossia, se è l'ultimo nodo della lista?
 - abbiamo già visto che esiste il riferimento che non si riferisce a un bel niente, ed è il famoso riferimento **null**

Le liste concatenate (*linked list*)

- Possiamo sfruttare la potenza dei riferimenti per creare la nostra struttura dati dinamica. Chiameremo la nostra struttura **singly linked list** (lista concatenata semplice)
- Diciamo il concatenamento è semplice perché un nodo ha il riferimento al successore, ma non al precedente

```
class ListNode {  
    private int key;  
    private ListNode next;  
}
```



Il nodo successore è a sua volta un nodo. Quindi deve avere necessariamente lo stesso tipo. La particolarità è che stiamo usando il tipo `ListNode` per definire il tipo `ListNode`. Il tipo che stiamo creando è dunque autoreferenziale, o ricorsivo.

Le liste concatenate (*linked list*)

```
class ListNode {  
    private int key;  
    private ListNode next;  
  
    ListNode(int key, ListNode next) {  
        this.key = key;  
        this.next = next;  
    }  
  
    public int getKey(){  
        return key;  
    }  
    public ListNode getNext(){  
        return next;  
    }  
    public void setNext(ListNode next){  
        this.next = next;  
    }  
}
```

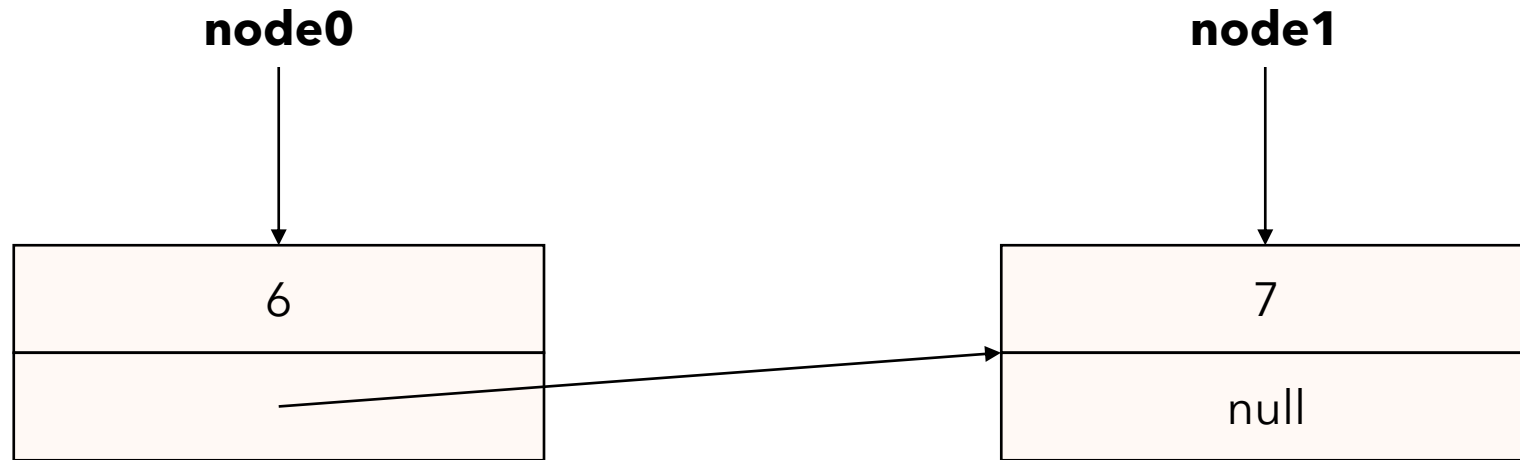
Nel main:

```
ListNode node1 = new ListNode(7, null);  
ListNode node0 = new ListNode(6, node1);  
System.out.println(n0.getNext() + " " + n1);
```

node0 next reference's value is: ListNode@2a3046da
node1's value is: ListNode@2a3046da

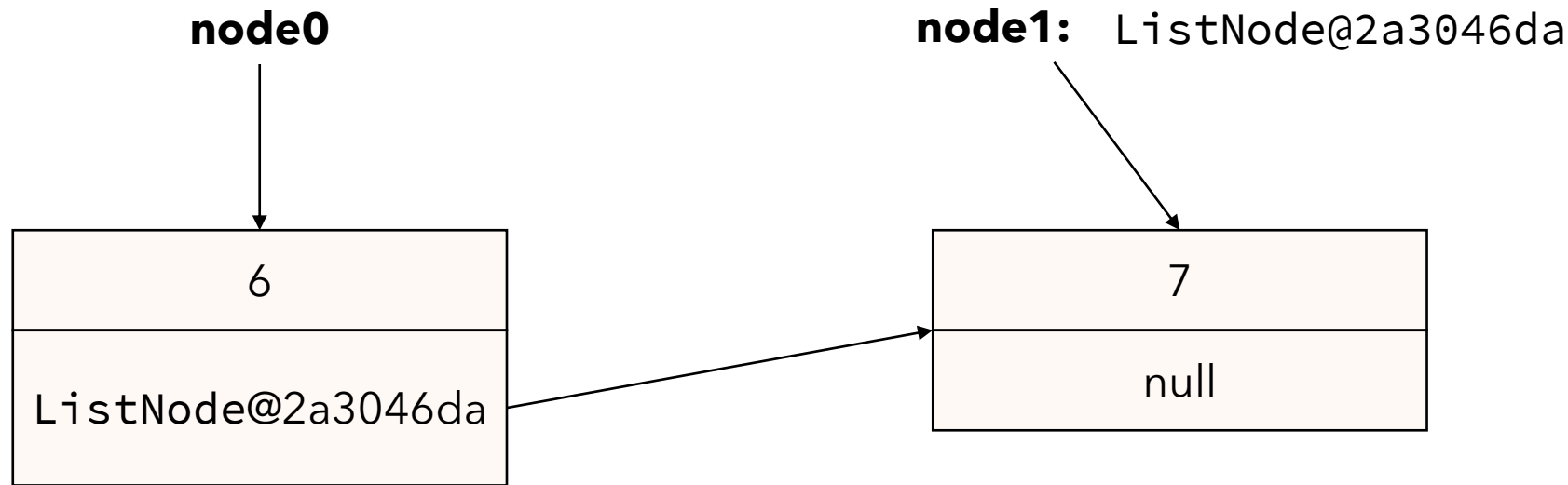
Le liste concatenate (*linked list*)

- Ora basta disegnare palle. Continuiamo con le care vecchie scatole e freccette.



Le liste concatenate (*linked list*)

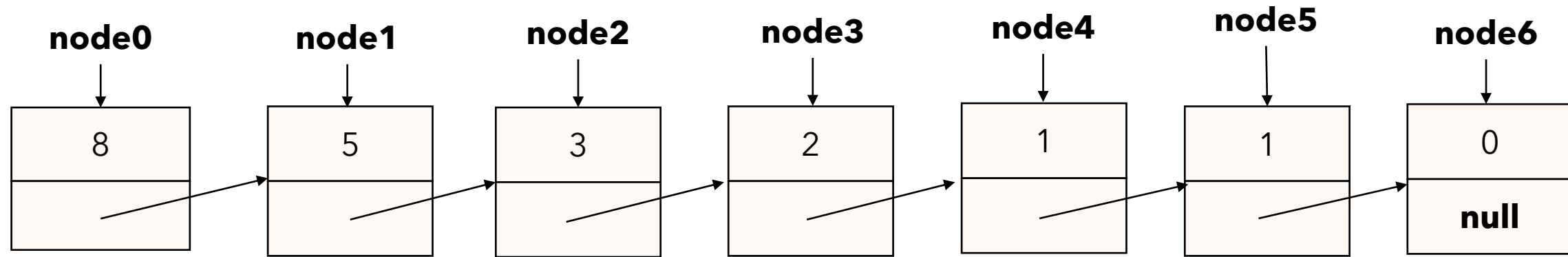
- Ora basta disegnare palle. Continuiamo con le care vecchie scatole e freccette.



node0.next e node1 sono riferimenti allo stesso oggetto di tipo `ListNode`

Le liste concatenate (*linked list*)

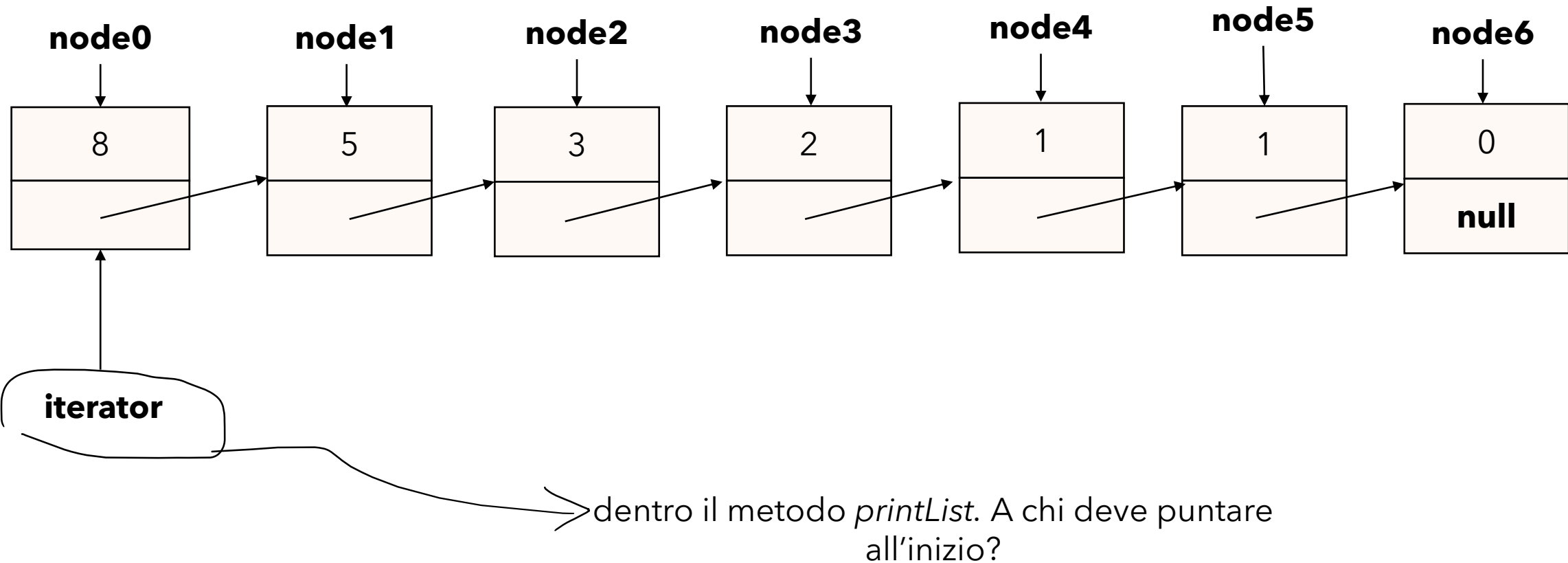
- Creare una lista concatenata di 7 nodi di chiave intera (manualmente, uno alla volta). Dare alle chiavi i primi 7 numeri di Fibonacci (partendo dall'ultimo nodo)



- Scrivere un metodo che stampi su stdout le chiavi della lista (partendo dall'oggetto di invocazioni fino all'ultimo nodo). L'algoritmo deve essere iterativo. NB: il riferimento **this** non può essere riassegnato.

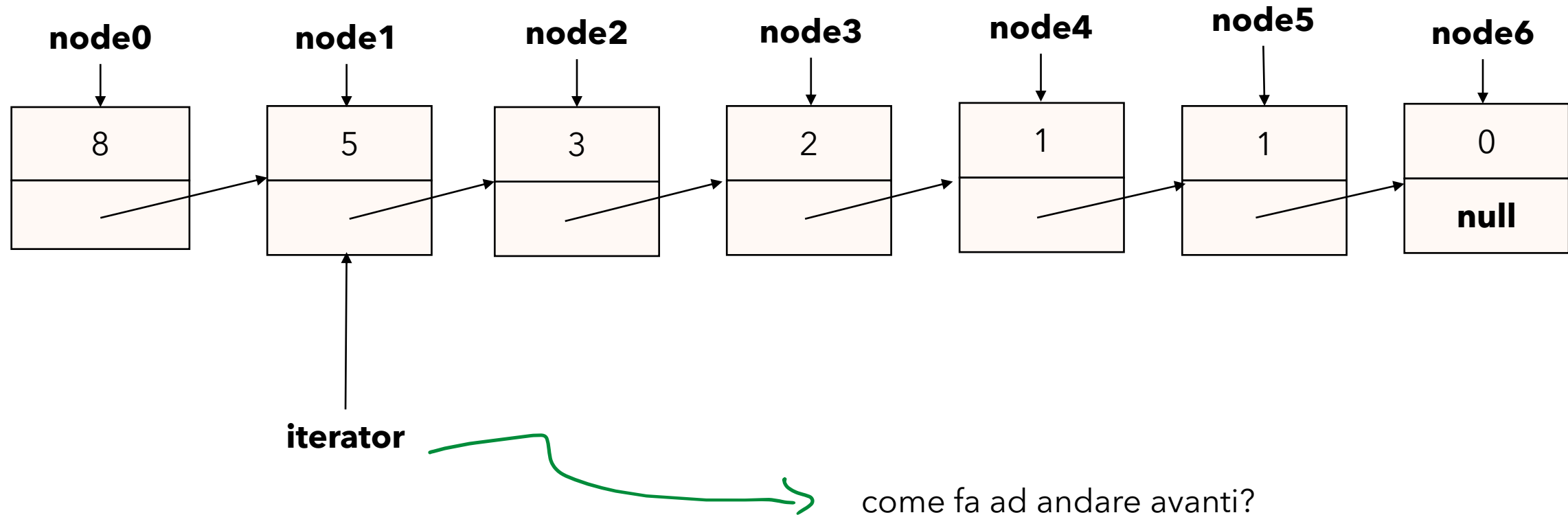
Le liste concatenate (*linked list*)

- Il metodo conosce il riferimento `this`, ma non può riassegnarlo. Bisogna creare una «testina» (in informatica si dice *iteratore*) che scorre sulla lista elemento per elemento.



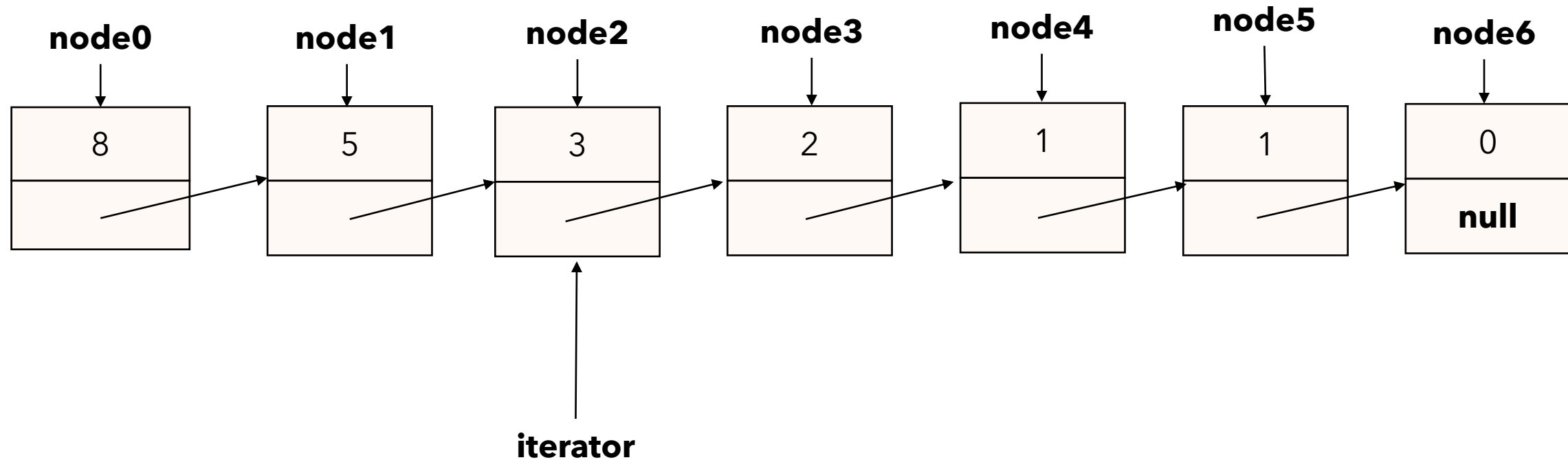
Le liste concatenate (*linked list*)

- Il metodo conosce il riferimento `this`, ma non può riassegnarlo. Bisogna creare una «testina» (in informatica si dice *iteratore*) che scorre sulla lista elemento per elemento.



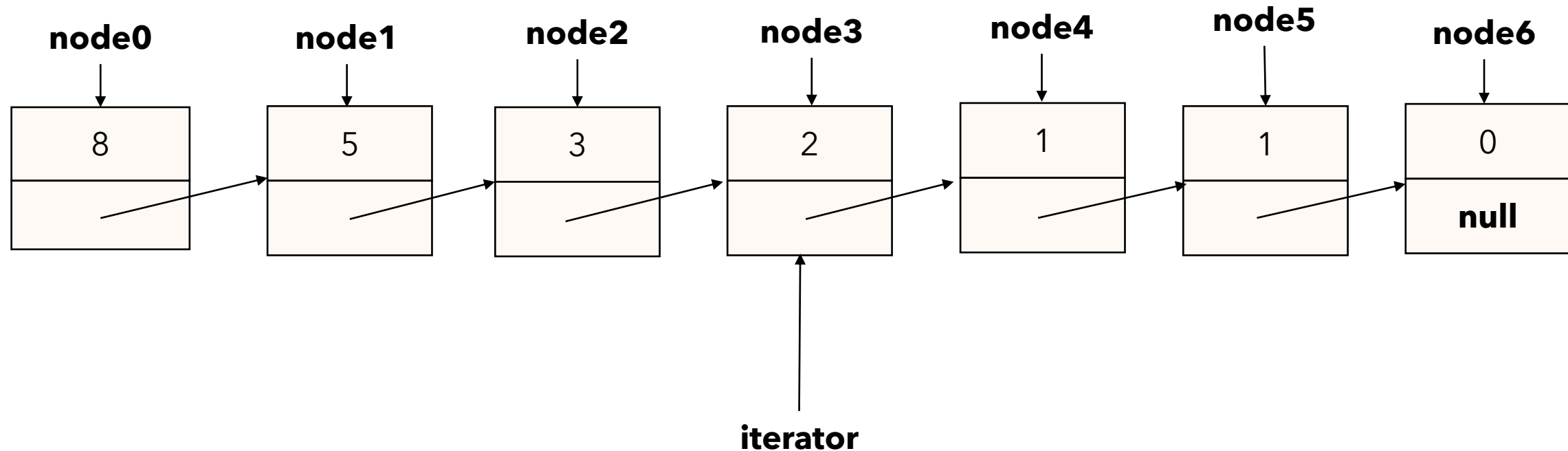
Le liste concatenate (*linked list*)

- Il metodo conosce il riferimento `this`, ma non può riassegnarlo. Bisogna creare una «testina» (in informatica si dice *iteratore*) che scorre sulla lista elemento per elemento.



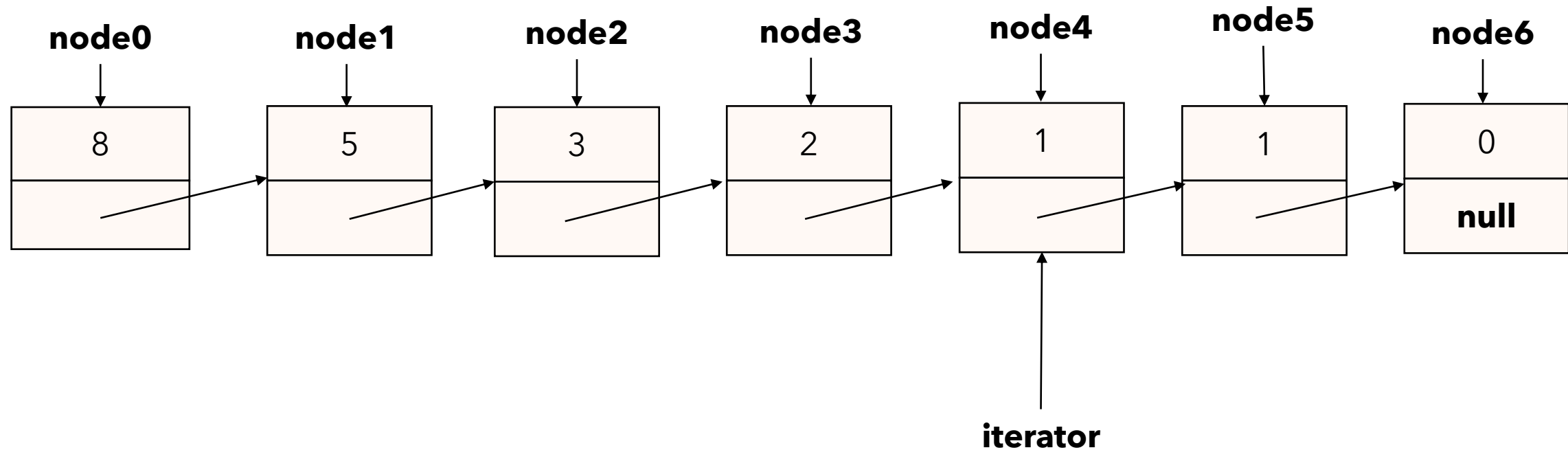
Le liste concatenate (*linked list*)

- Il metodo conosce il riferimento `this`, ma non può riassegnarlo. Bisogna creare una «testina» (in informatica si dice *iteratore*) che scorre sulla lista elemento per elemento.



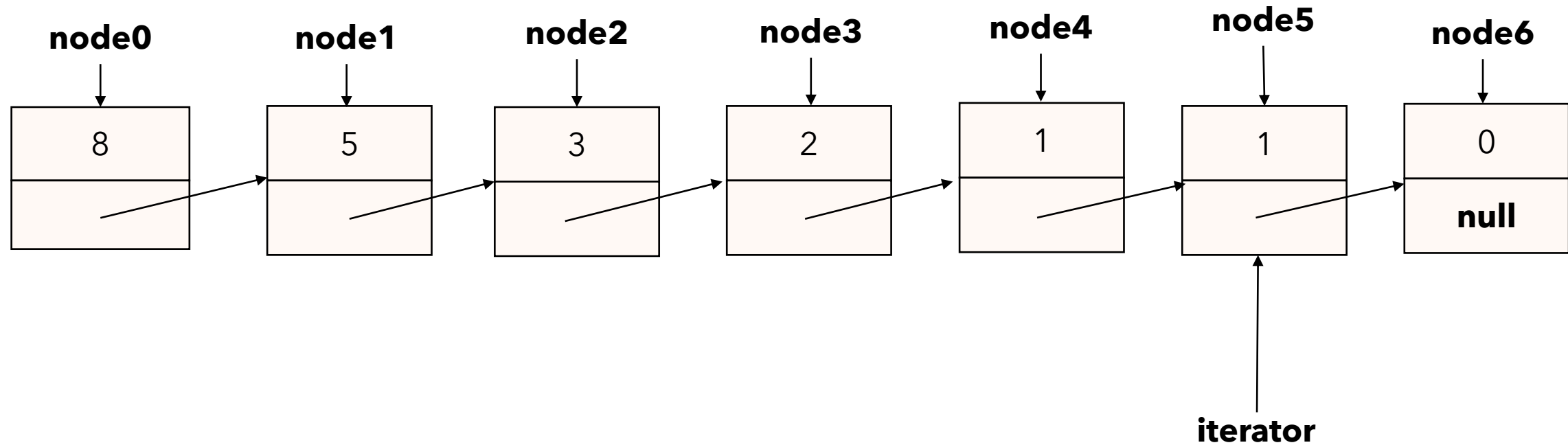
Le liste concatenate (*linked list*)

- Il metodo conosce il riferimento `this`, ma non può riassegnarlo. Bisogna creare una «testina» (in informatica si dice *iteratore*) che scorre sulla lista elemento per elemento.



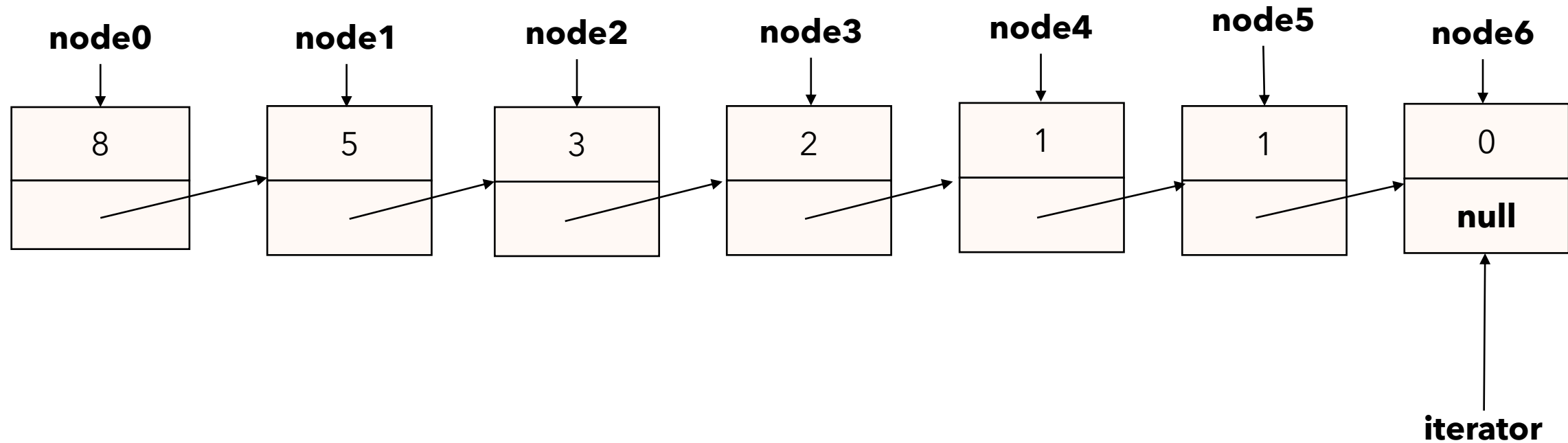
Le liste concatenate (*linked list*)

- Il metodo conosce il riferimento `this`, ma non può riassegnarlo. Bisogna creare una «testina» (in informatica si dice *iteratore*) che scorre sulla lista elemento per elemento.



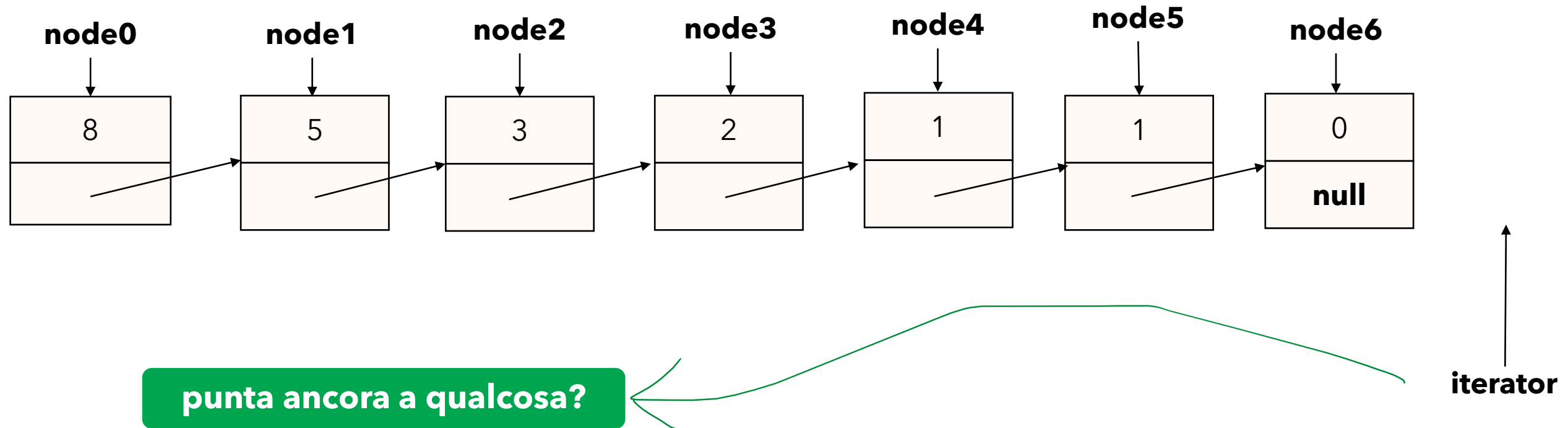
Le liste concatenate (*linked list*)

- Il metodo conosce il riferimento `this`, ma non può riassegnarlo. Bisogna creare una «testina» (in informatica si dice *iteratore*) che scorre sulla lista elemento per elemento.



Le liste concatenate (*linked list*)

- Il metodo conosce il riferimento `this`, ma non può riassegnarlo. Bisogna creare una «testina» (in informatica si dice *iteratore*) che scorre sulla lista elemento per elemento.



Le liste concatenate (*linked list*)

```
public void printList() {  
    System.out.print("Linked list's content is: ");  
  
    ListNode iterator = this;  
    while (iterator != null) {  
        System.out.print(iterator.key + " ");  
        iterator = iterator.next;  
    }  
    System.out.println();  
}
```

Le liste concatenate (*linked list*)

```
public void printList() {  
    System.out.print("Linked list's content is: ");  
  
    ListNode iterator = this;  
    while (iterator != null) {  
        System.out.print(iterator.key + " ");  
        iterator = iterator.next;  
    }  
    System.out.println();  
    System.out.println(iterator.key);  
}
```

✓ HI A ↑ HAPPENS?

Le liste concatenate (*linked list*)

```
Exception in thread "main" java.lang.NullPointerException  
    at ListNode.printList(LinkedList.java:37)  
    at LinkedList.main(LinkedList.java:97)
```

Potreste averlo trovato navigando un qualche sito web come errore 500 (Internal Server Error)

Le liste concatenate (*linked list*)

- Scrivere un metodo di ListNode che stampi le chiavi dei nodi della lista dal primo all'ultimo con un algoritmo **ricorsivo**
- Sfruttare la definizione **ricorsiva** di lista concatenata:
 - Una lista concatenata è:
 - la lista priva di alcun nodo
oppure
 - un nodo collegato al resto della lista concatenata
- Spiegare perché la definizione precedente è ricorsiva (si dice anche «induttiva»)

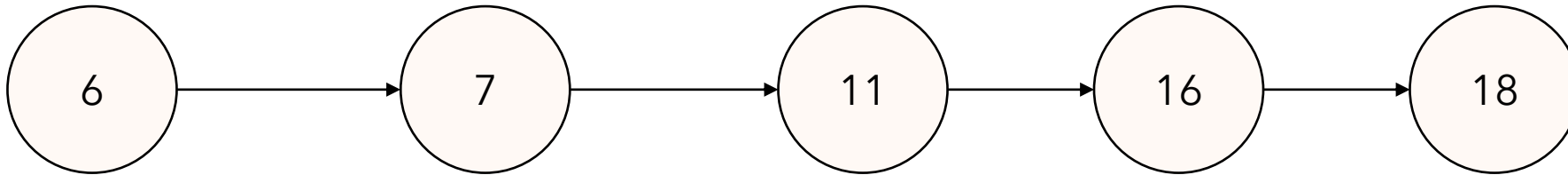
Le liste concatenate (*linked list*)

```
public void printListRecursive(ListNode it) {  
    if (it == null) {  
        System.out.println();  
        return;  
    }  
    System.out.print(it.key + " ");  
    printListRecursive(it.next);  
}
```

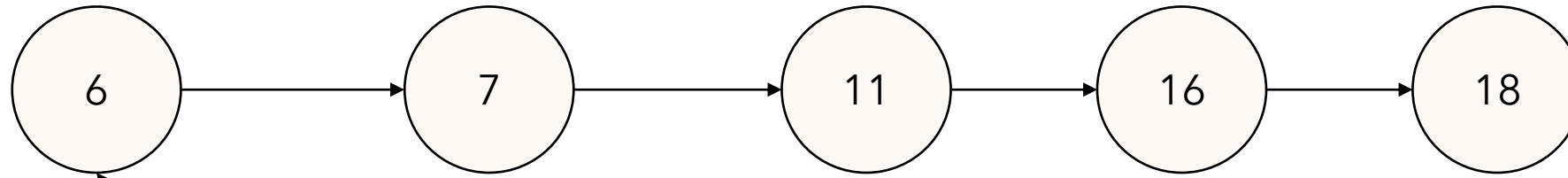

Calcolare la dimensione della lista ricorsivamente

La lista deve avere almeno un nodo. Altrimenti non sarebbe possibile chiamare il metodo.

```
public int getSizeRecursive() {  
    if (this.next == null) {  
        return 1;  
    }  
    int sizeOfRemainder = this.next.getSizeRecursive();  
    return sizeOfRemainder + 1;  
}
```



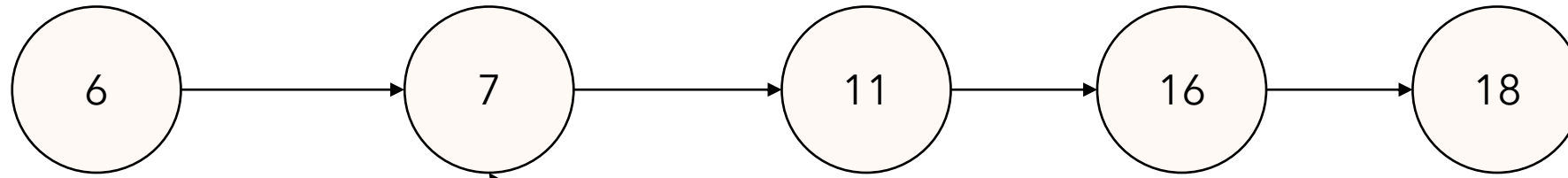
Calcolare la dimensione della lista ricorsivamente



pushed record of 1st,
non recursive, call

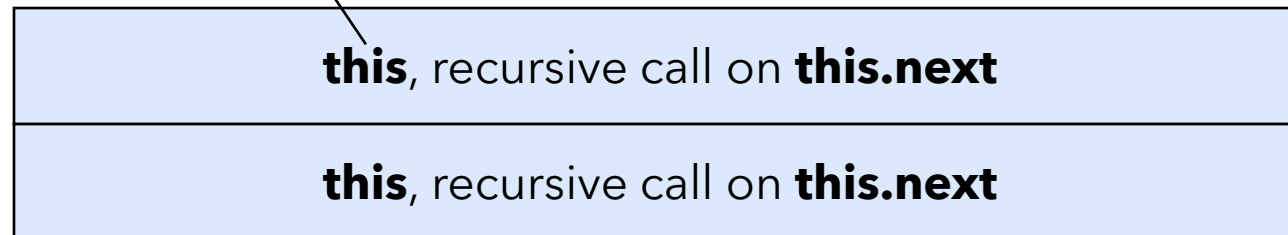
this, recursive call on **this.next**

Calcolare la dimensione della lista ricorsivamente

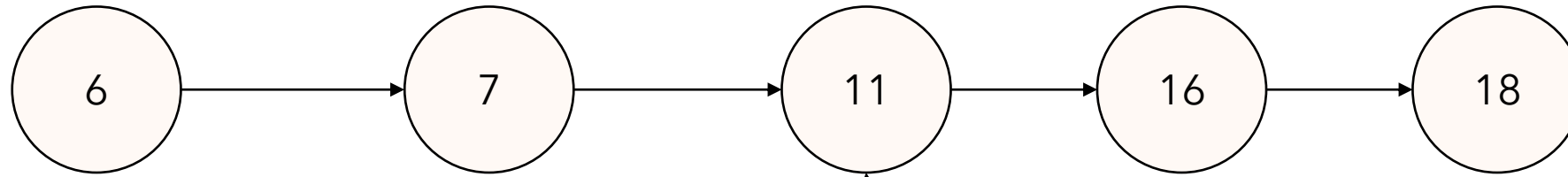


pushed record of 2nd,
recursive, call

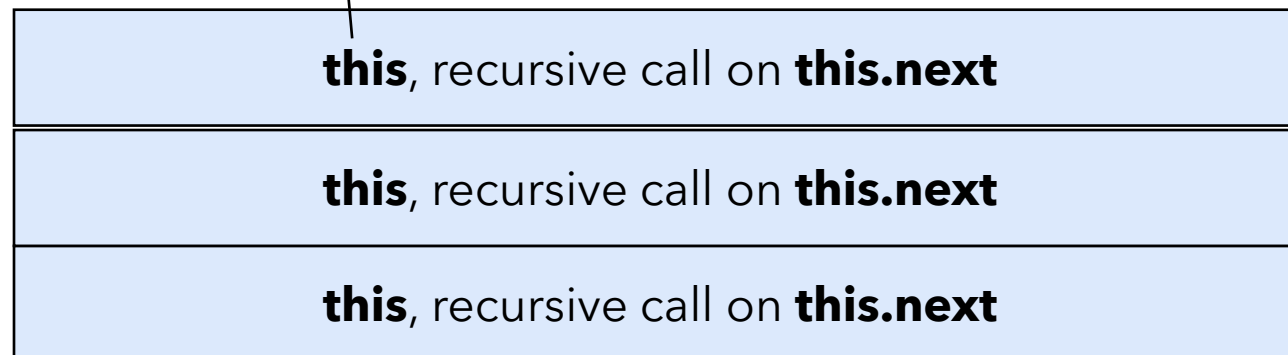
pushed record of 1st,
non recursive, call



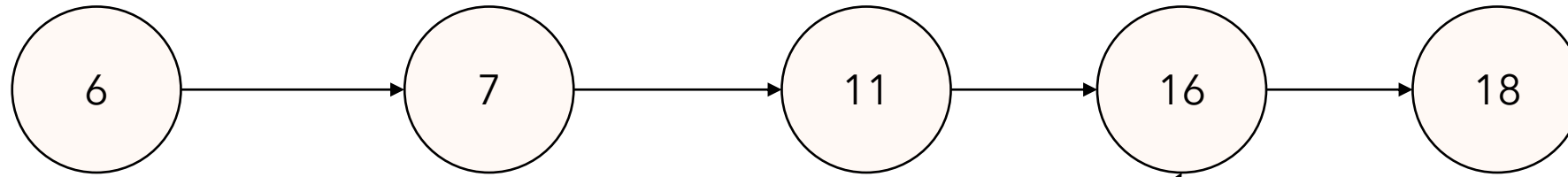
Calcolare la dimensione della lista ricorsivamente



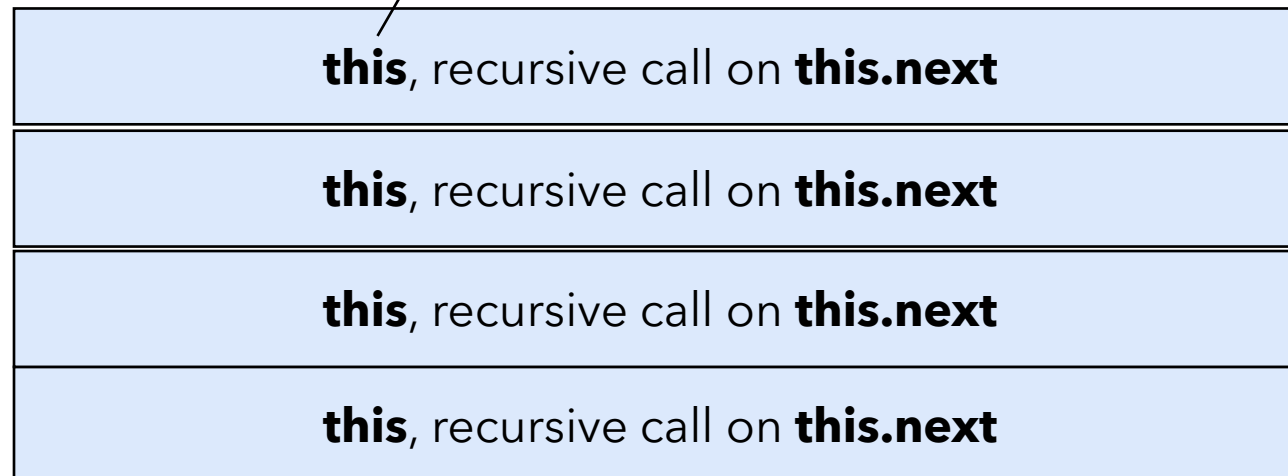
pushed record of 3rd,
recursive, call
pushed record of 2nd,
recursive, call
pushed record of 1st,
non recursive, call



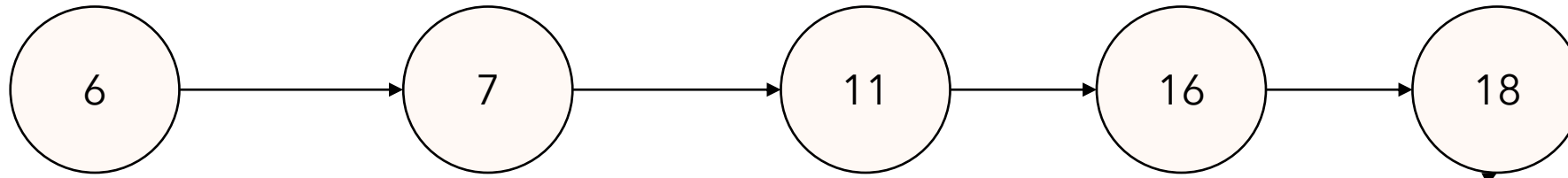
Calcolare la dimensione della lista ricorsivamente



pushed record of 4th,
recursive, call
pushed record of 3rd,
recursive, call
pushed record of 2nd,
recursive, call
pushed record of 1st,
non recursive, call



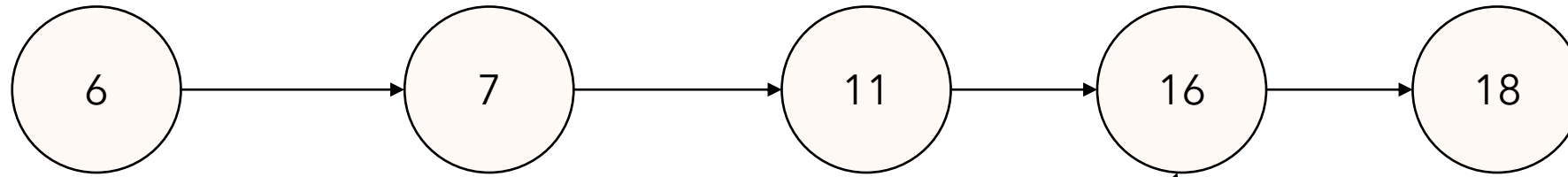
Calcolare la dimensione della lista ricorsivamente



pushed record of 5th,
recursive, call
pushed record of 4th,
recursive, call
pushed record of 3rd,
recursive, call
pushed record of 2nd,
recursive, call
pushed record of 1st,
non recursive, call

this , base case: return 1 to the caller
this , 1 + recursive call on this.next
this , 1 + recursive call on this.next
this , 1 + recursive call on this.next
this , 1 + recursive call on this.next

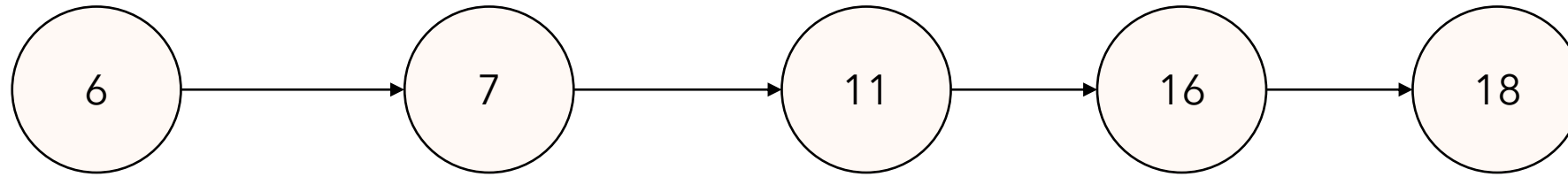
Calcolare la dimensione della lista ricorsivamente



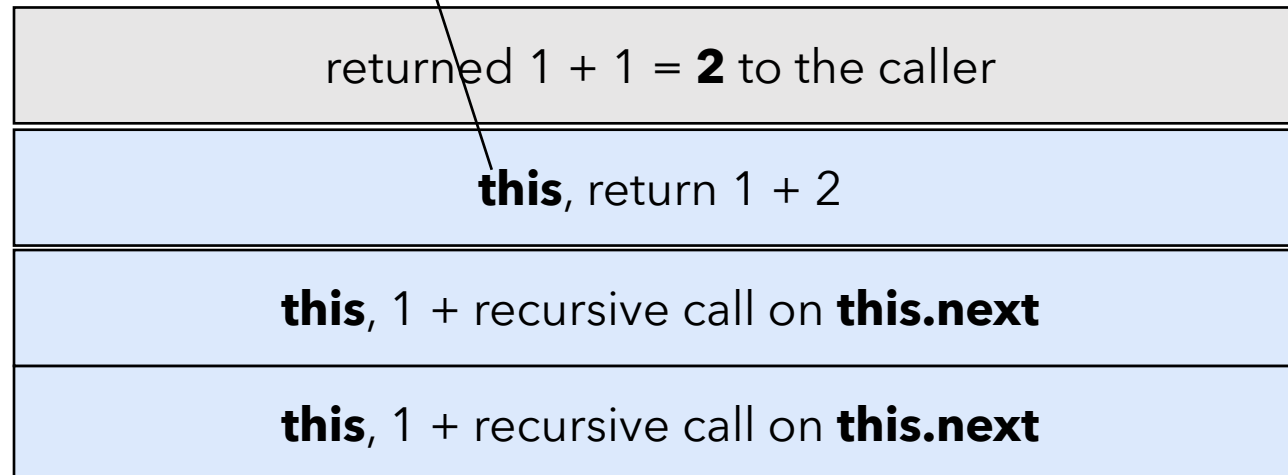
popped record

returned 1 to the caller
this , return $1 + 1 = 2$
this , $1 + \text{recursive call on } \mathbf{this.next}$
this , $1 + \text{recursive call on } \mathbf{this.next}$
this , $1 + \text{recursive call on } \mathbf{this.next}$

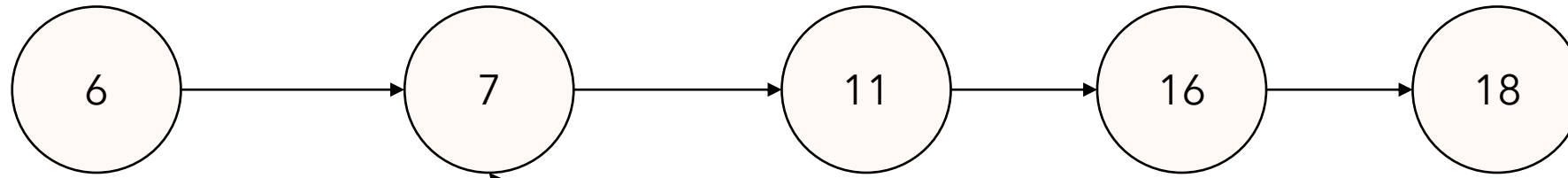
Calcolare la dimensione della lista ricorsivamente



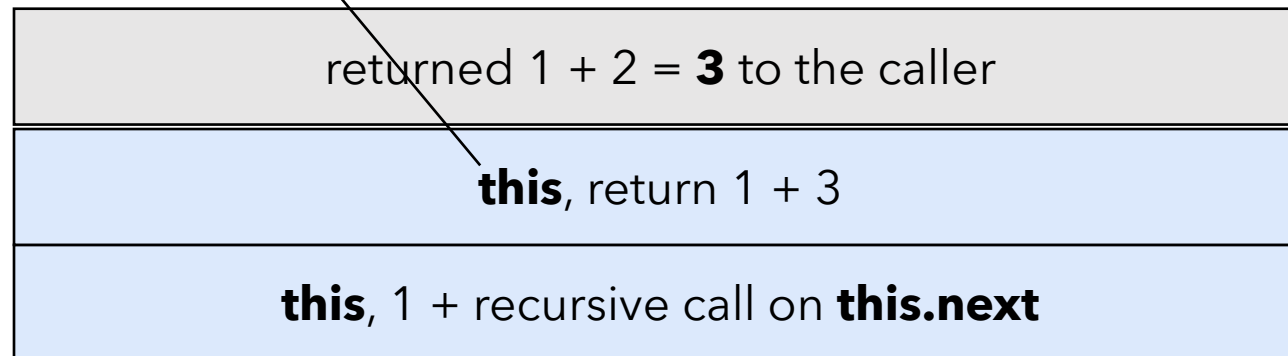
popped record



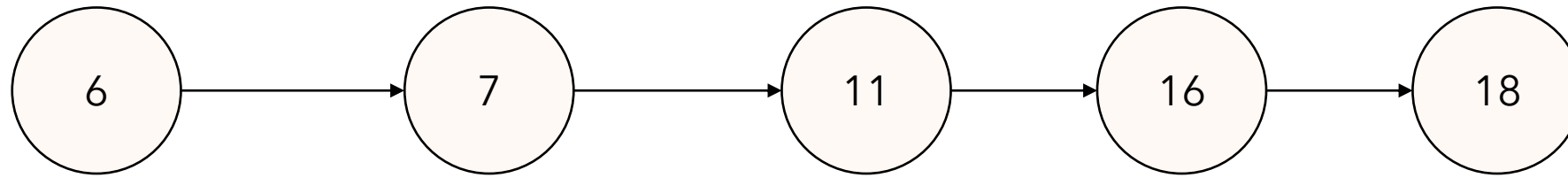
Calcolare la dimensione della lista ricorsivamente



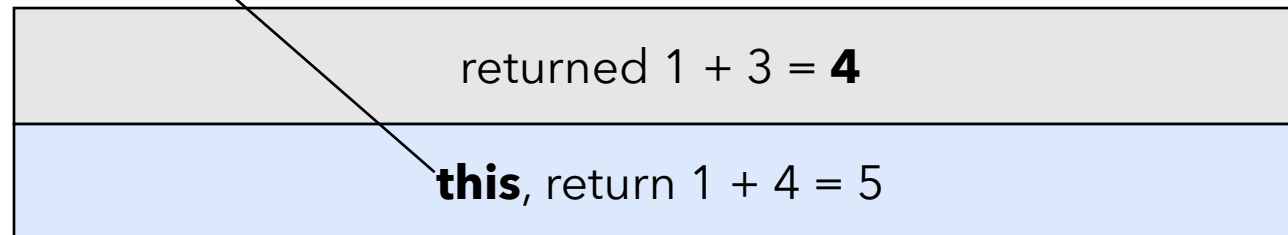
popped record



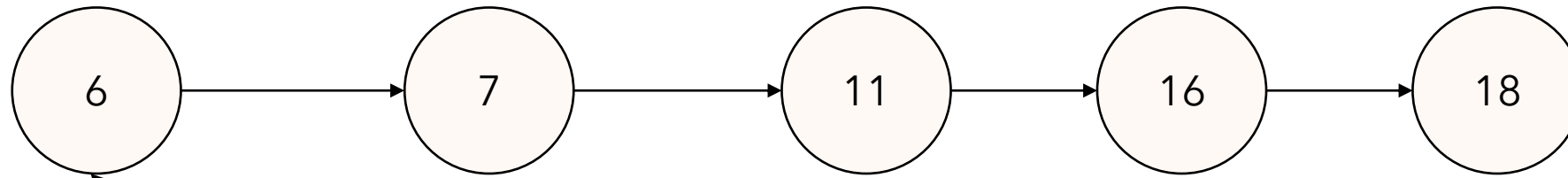
Calcolare la dimensione della lista ricorsivamente



popped record



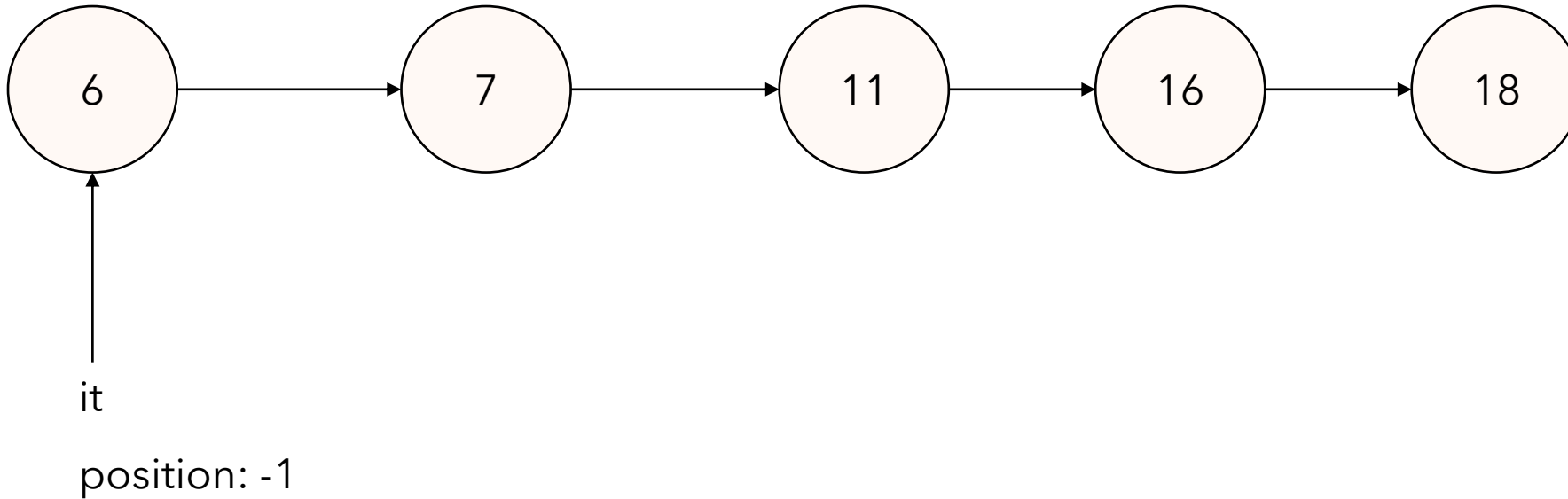
Calcolare la dimensione della lista ricorsivamente



popped record

this, returned $1 + 4 = \mathbf{5}$ to external caller

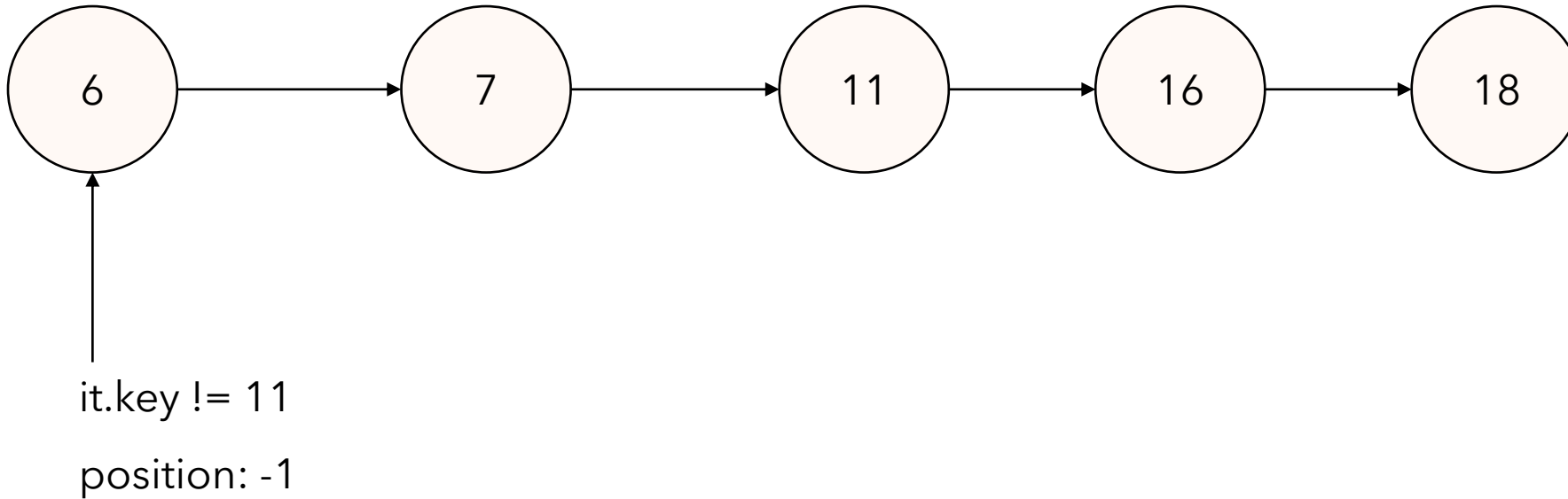
Ricerca lineare di una chiave



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

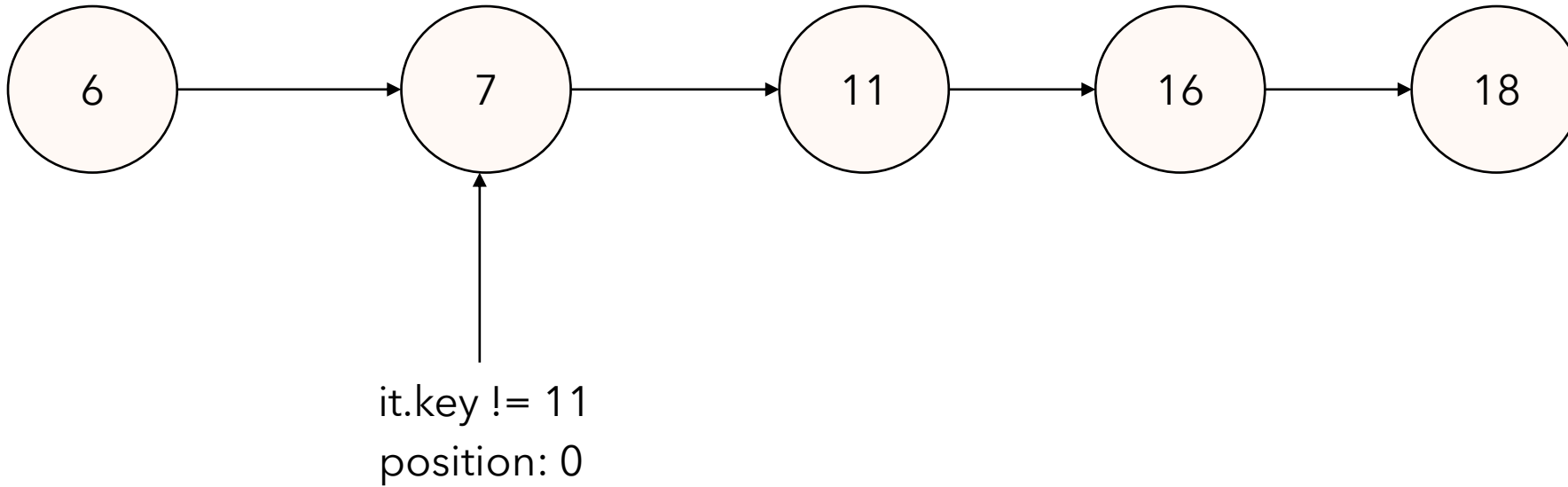
Prima di iniziare a cercare, diciamo che la chiave si trova nella posizione fittizia -1

Ricerca lineare di una chiave



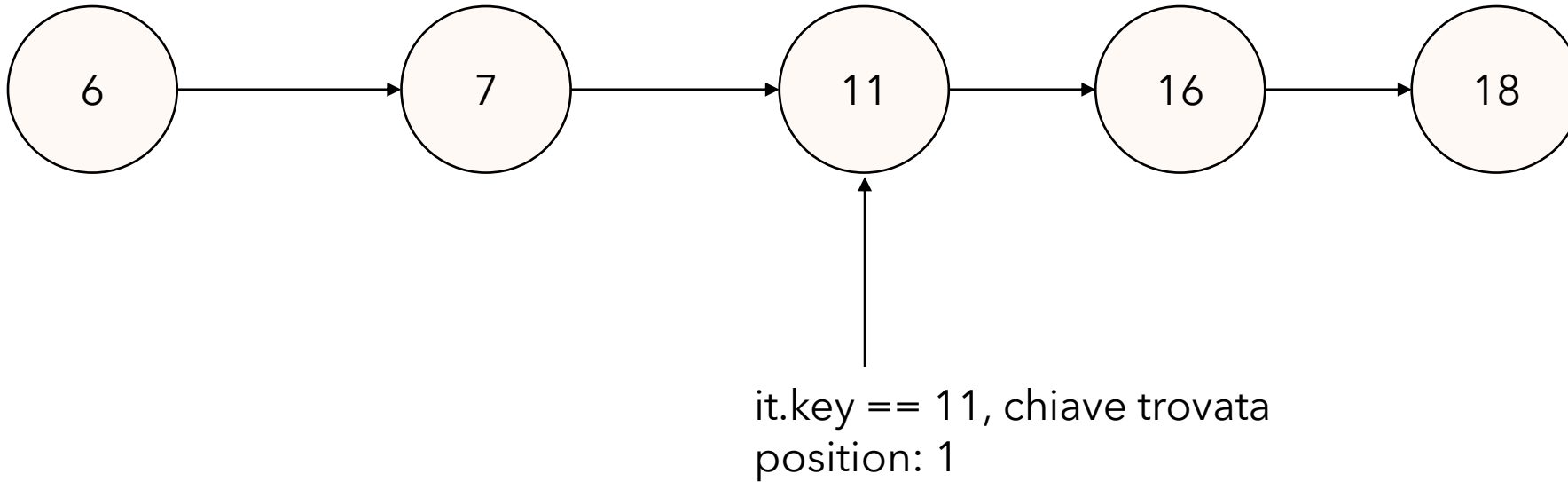
Cerchiamo la prima occorrenza della chiave 11 e restituiamo l'indice del nodo che la contiene (diamo al primo nodo indice 0)

Ricerca lineare di una chiave



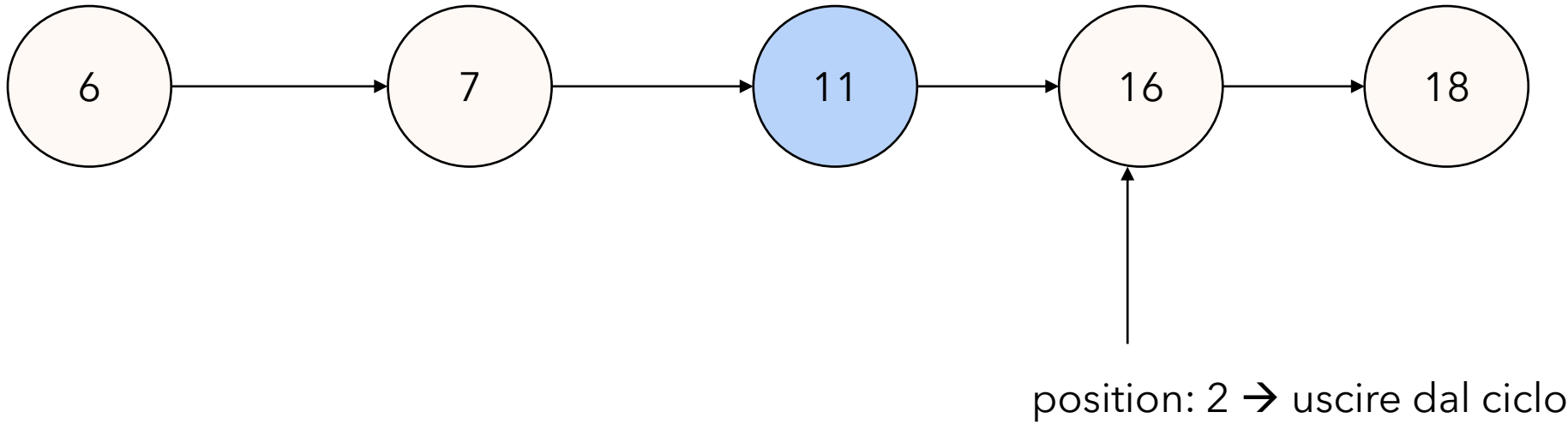
Cerchiamo la prima occorrenza della chiave 11 e restituiamo l'indice del nodo che la contiene (diamo al primo nodo indice 0)

Ricerca lineare di una chiave



Cerchiamo la prima occorrenza della chiave 11 e restituiamo l'indice del nodo che la contiene (diamo al primo nodo indice 0). La procedura deve fermarsi dopo aver trovato la chiave cercata.

Ricerca lineare di una chiave

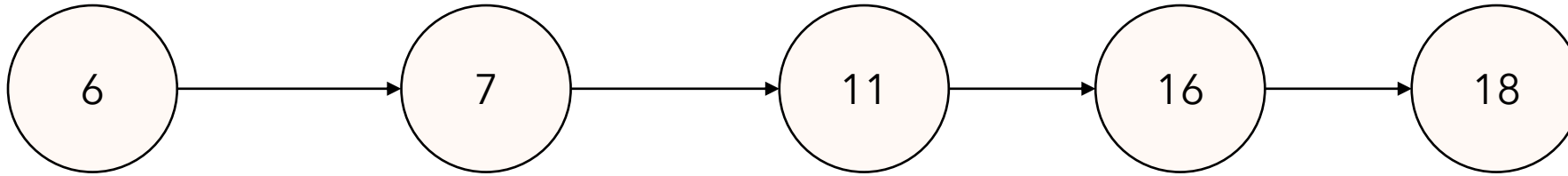


Cerchiamo la prima occorrenza della chiave 11 e restituiamo l'indice del nodo che la contiene (diamo al primo nodo indice 0). La procedura deve fermarsi dopo aver trovato la chiave cercata.

Ricerca lineare di una chiave

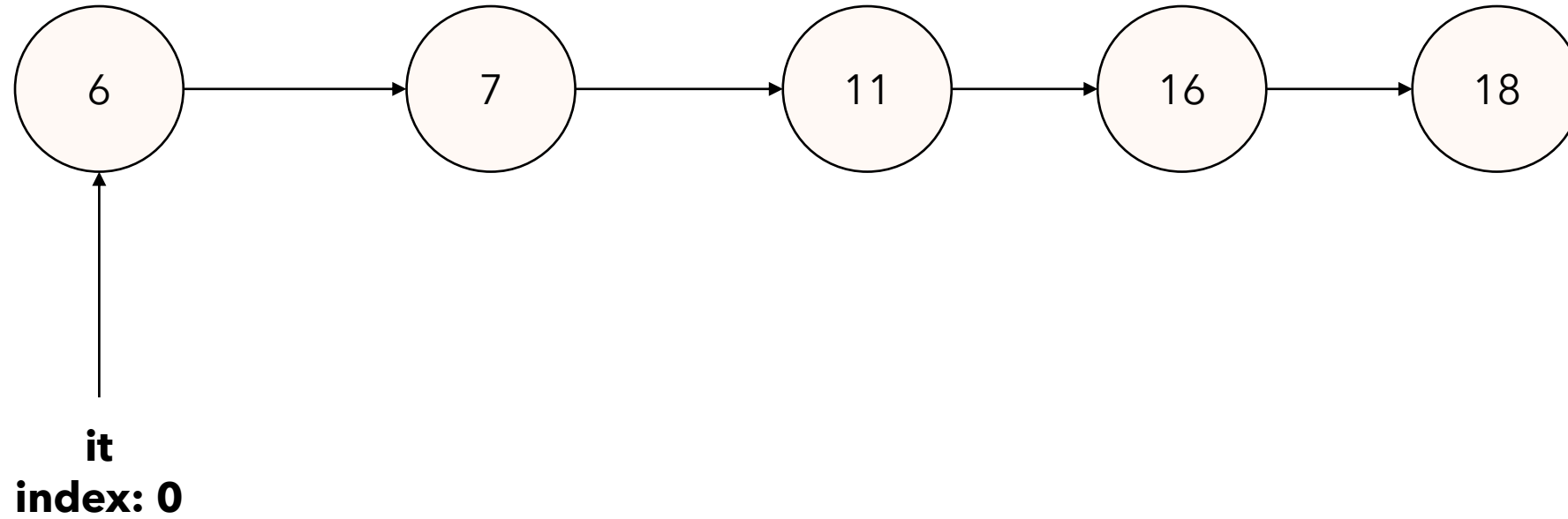
```
public int searchKey(int searchedKey) {  
    ListNode iterator = this;  
    int positionOfKey = -1;  
    boolean foundItem = false;  
  
    while (!foundItem && iterator != null) {  
        if (iterator.key == searchedKey) {  
            foundItem = true;  
        }  
        iterator = iterator.next;  
        positionOfKey++;  
    }  
  
    if (foundItem == false) {  
        positionOfKey = -1;  
    }  
  
    return positionOfKey;  
}
```

Accesso sequenziale all'n-esimo nodo



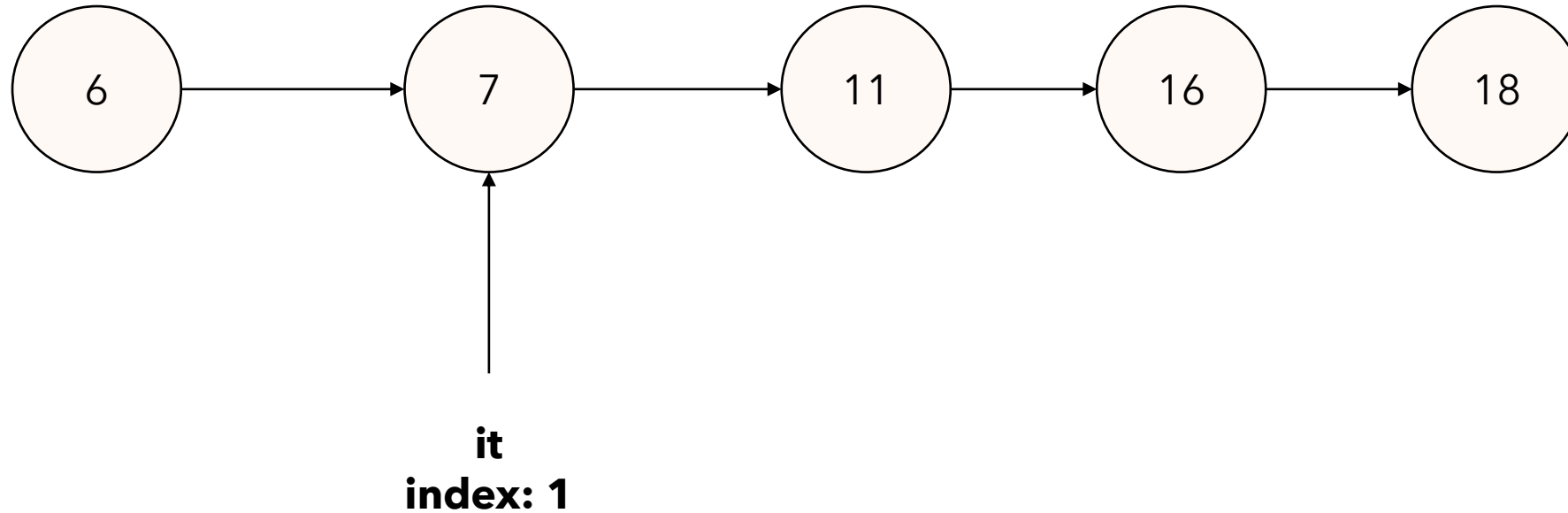
- Vogliamo un metodo che restituisca un riferimento al nodo di indice n
- Si parte sempre da 0
- Se viene richiesto un indice *illegale* (negativo o maggiore della dimensione della lista - 1), restituiamo il riferimento **null** (un comportamento che ricorda vagamente la *ArrayIndexOutOfBoundsException*)
- L'accesso è **sequenziale**: per accedere all' n -esimo nodo bisogna scorrere tutti i nodi precedenti (l'accesso per gli array invece è **diretto** (o **casuale**))
- Si dice che l'accesso avviene in tempo lineare sulla dimensione della lista:
 - se la lista ha n nodi, nel caso peggiore (*worst case*, ossia accesso all'ultimo nodo) devo scorrerla tutta

Accesso sequenziale all'n-esimo nodo



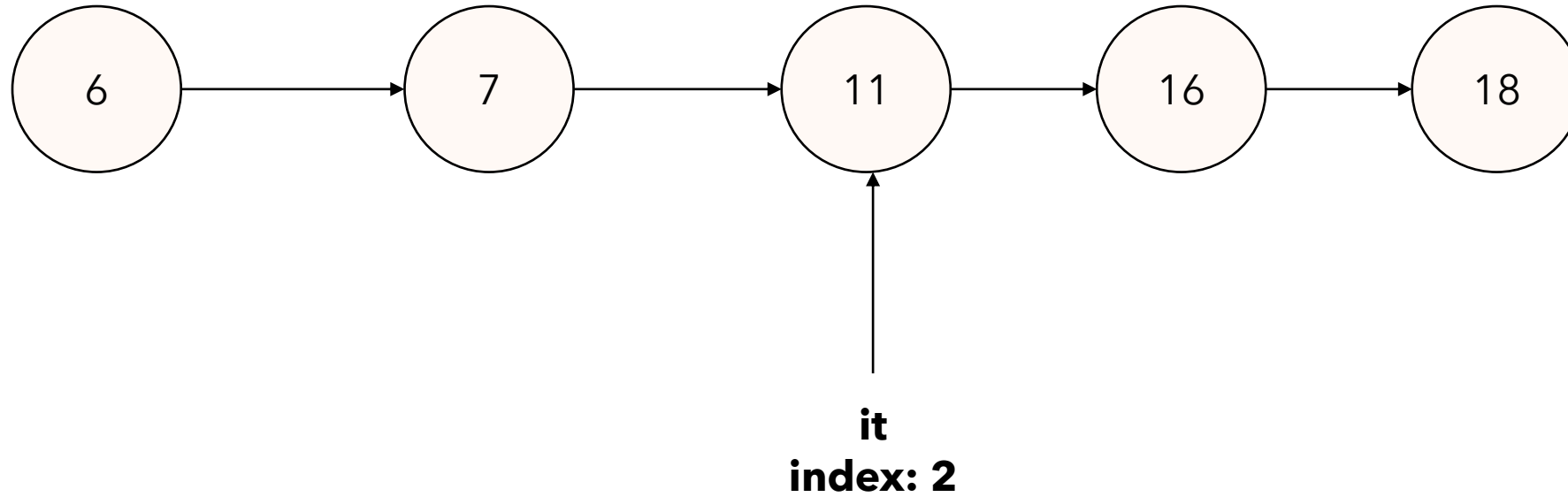
Vogliamo il nodo di indice 3

Accesso sequenziale all'n-esimo nodo



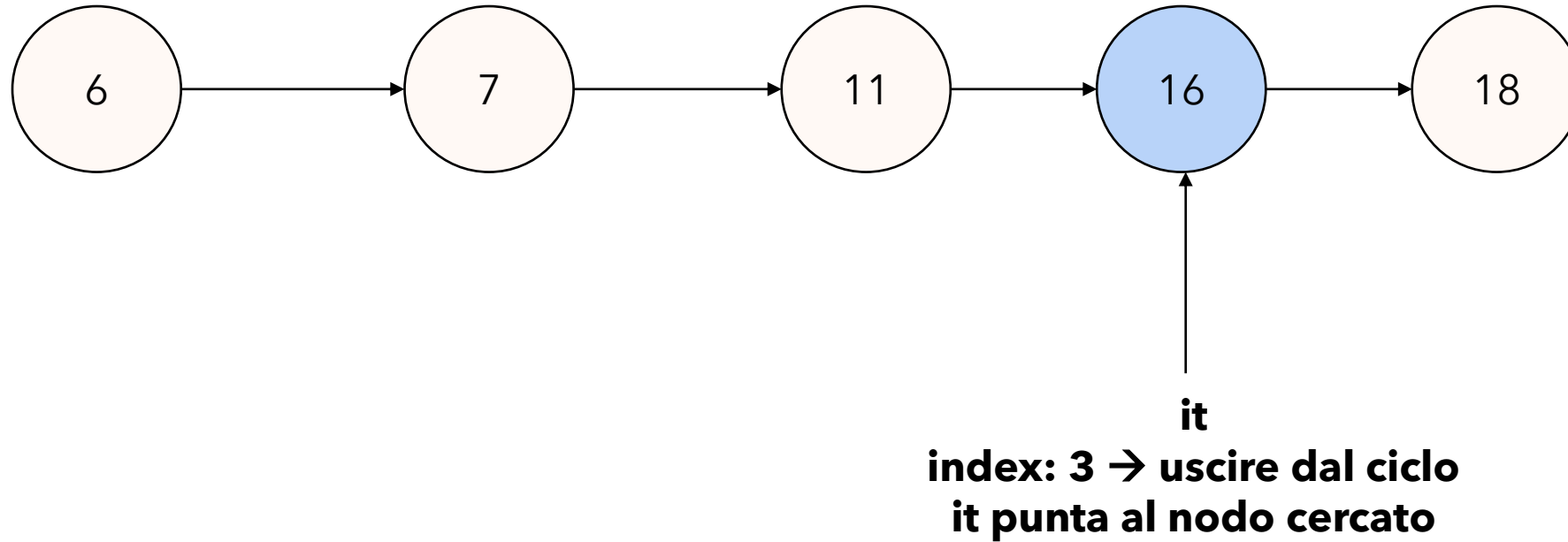
Vogliamo il nodo di indice 3

Accesso sequenziale all'n-esimo nodo



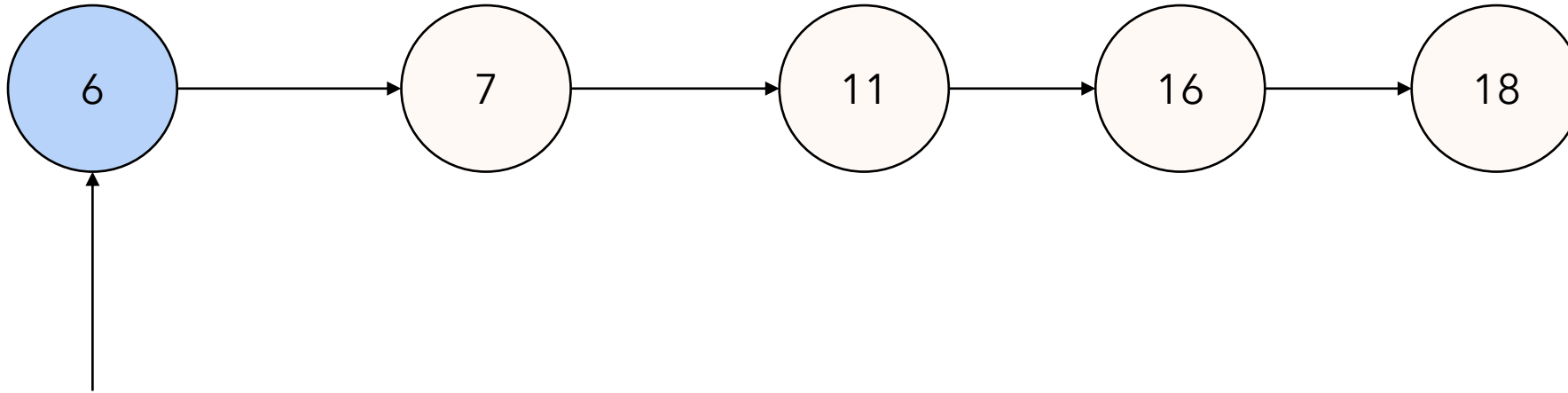
Vogliamo il nodo di indice 3

Accesso sequenziale all'n-esimo nodo



Vogliamo il nodo di indice 3

Accesso sequenziale all'n-esimo nodo



it

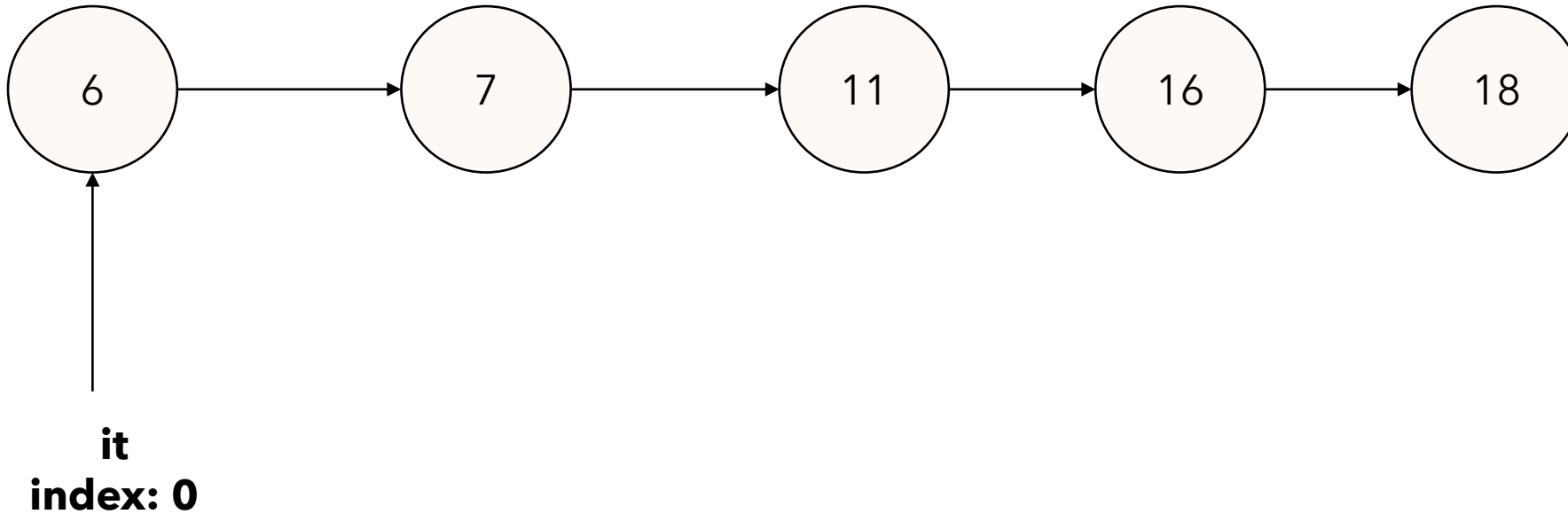
index: 0

Facciamo in modo di non entrare neanche nel ciclo

it punta già al nodo cercato

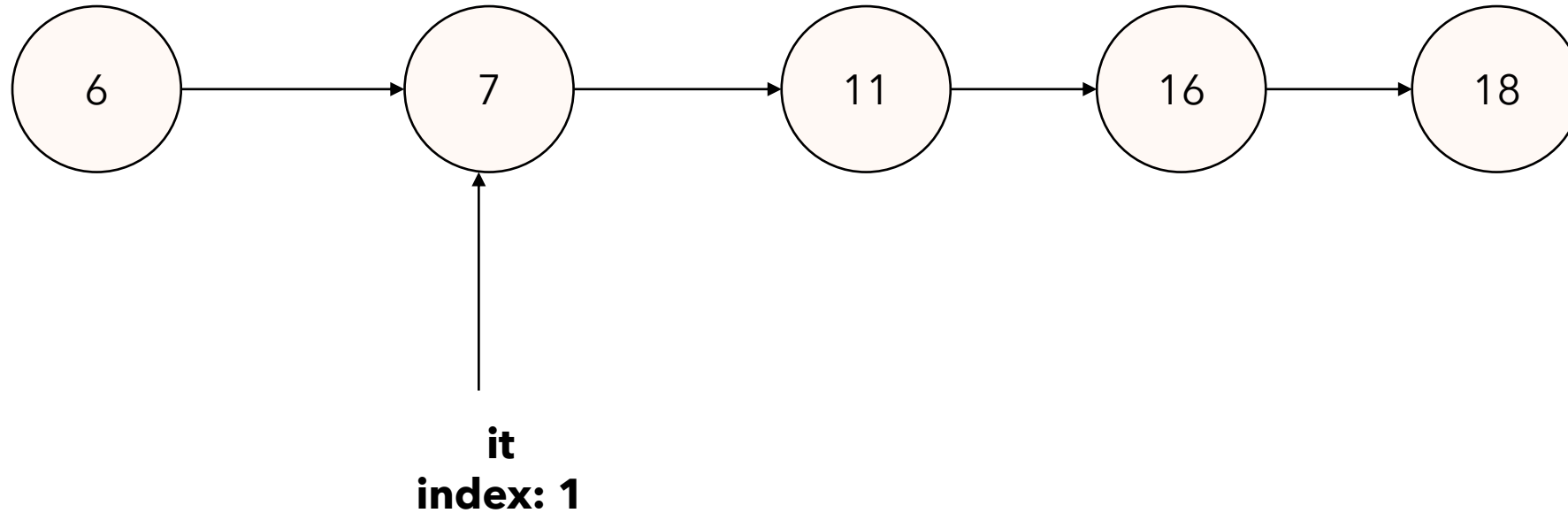
Vogliamo il nodo di indice 0

Accesso sequenziale all'n-esimo nodo



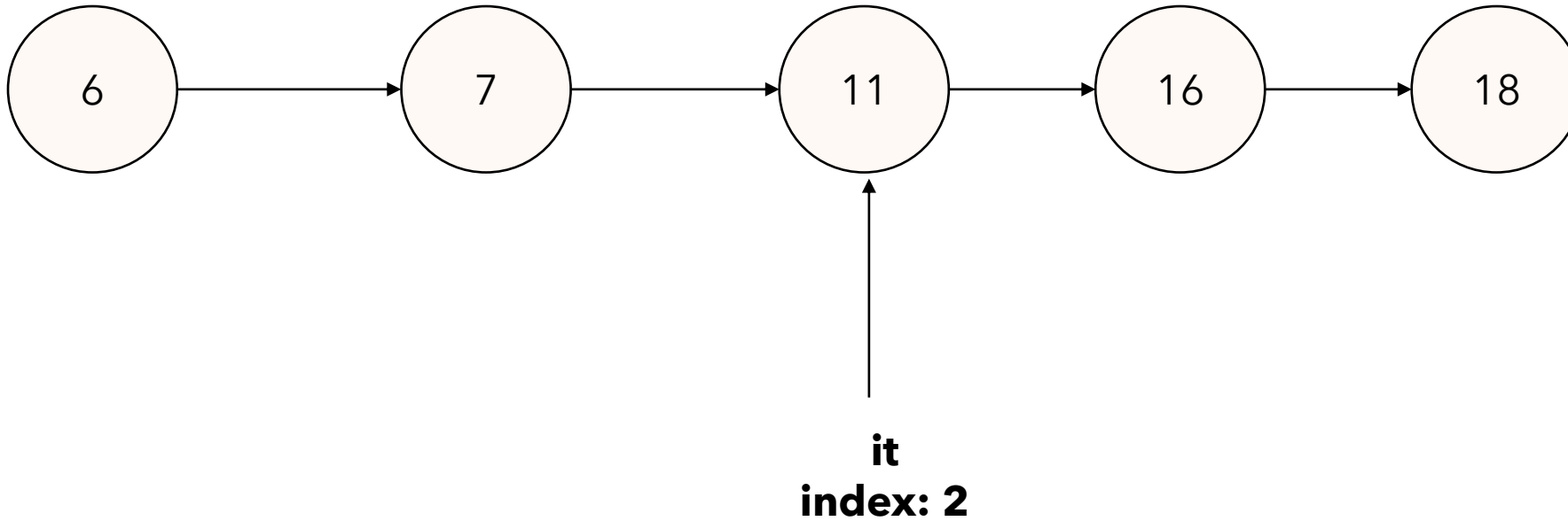
Vogliamo il nodo di indice 4

Accesso sequenziale all'n-esimo nodo



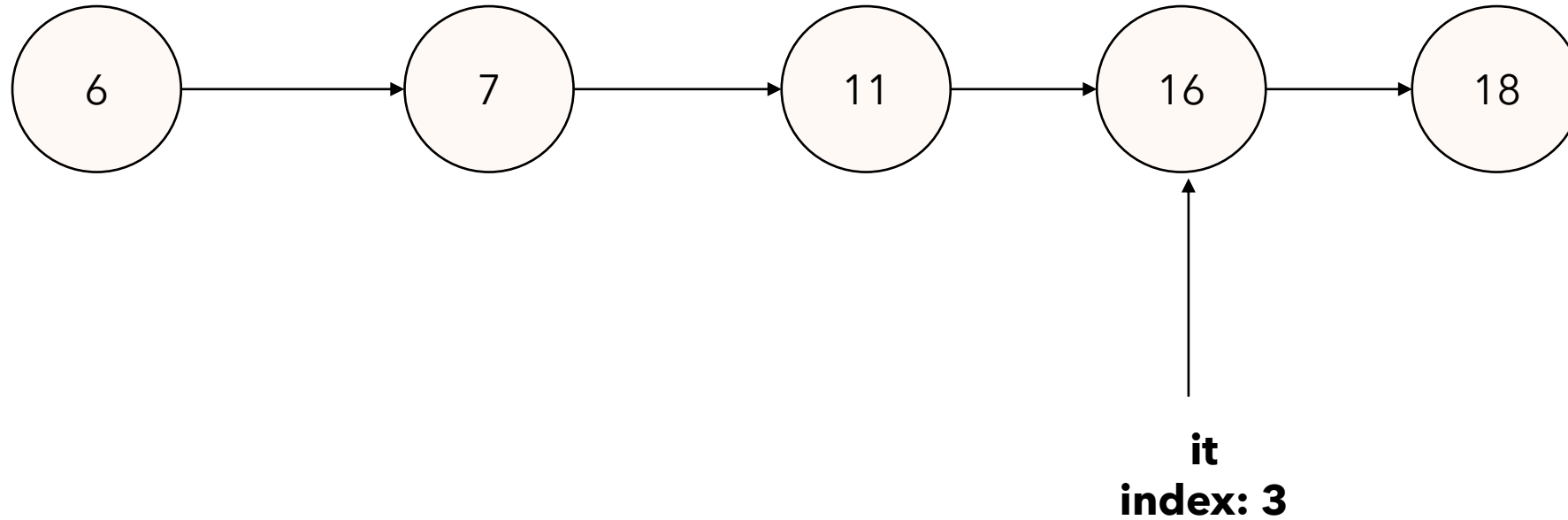
Vogliamo il nodo di indice 4

Accesso sequenziale all'n-esimo nodo



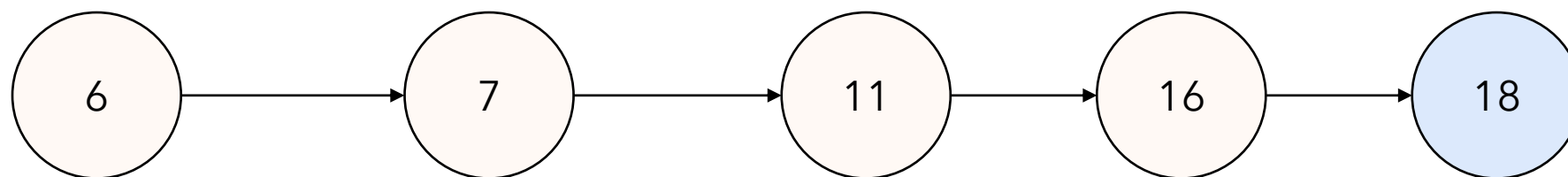
Vogliamo il nodo di indice 4

Accesso sequenziale all'n-esimo nodo



Vogliamo il nodo di indice 4

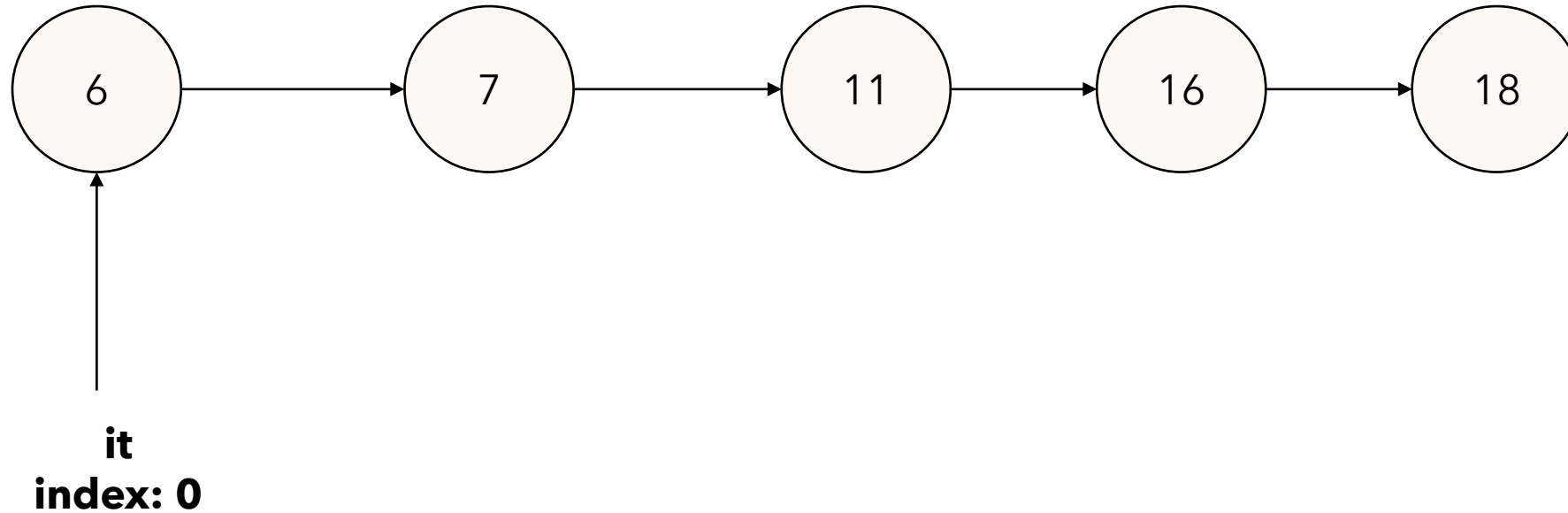
Accesso sequenziale all'n-esimo nodo



it
index: 4
Usciamo dal ciclo, it punta
al nodo cercato

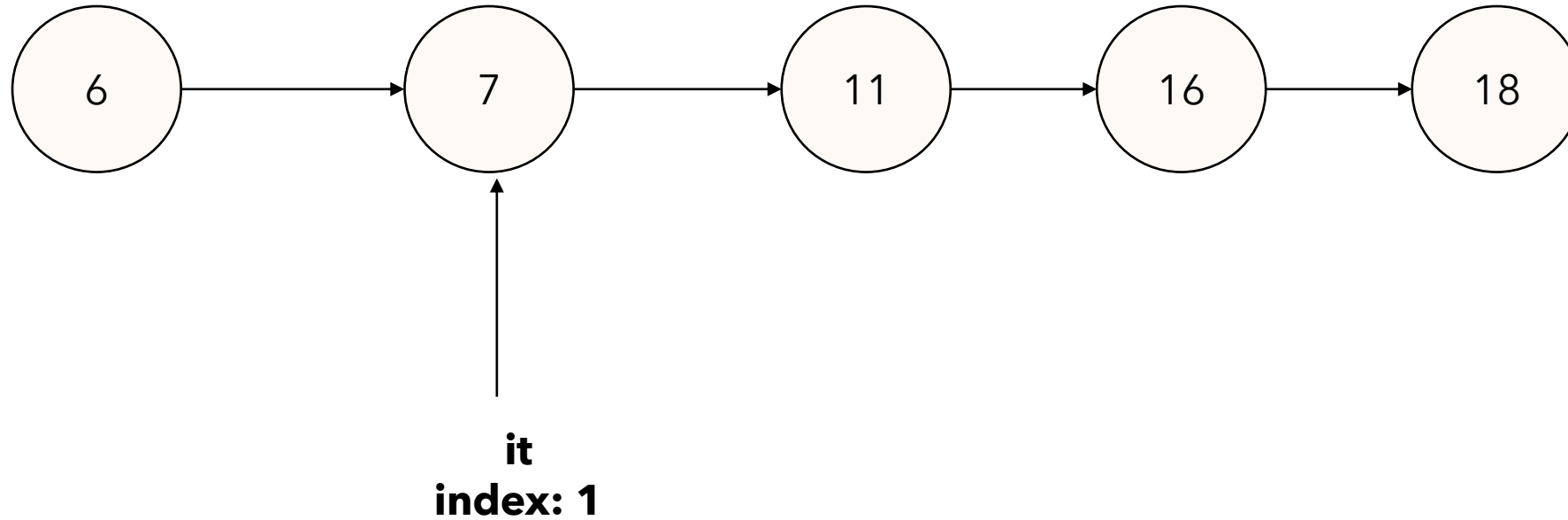
Vogliamo il nodo di indice 4

Accesso sequenziale all'n-esimo nodo



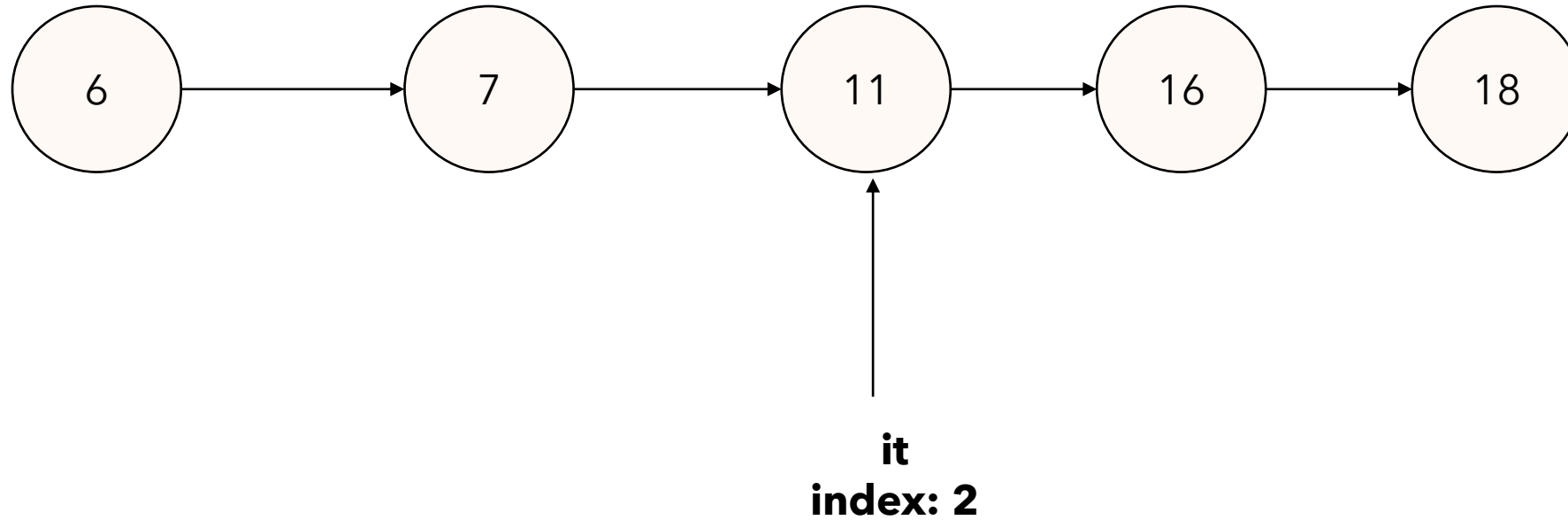
Vogliamo il nodo di indice 5 (non esistente)

Accesso sequenziale all'n-esimo nodo



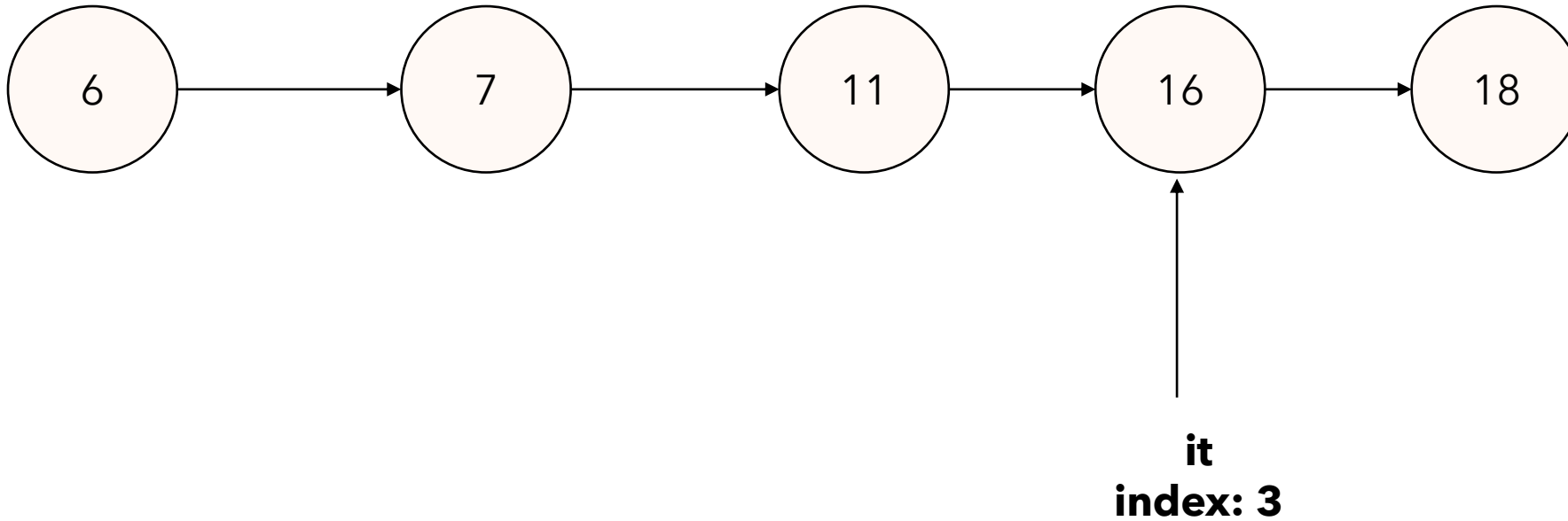
Vogliamo il nodo di indice 5 (non esistente)

Accesso sequenziale all'n-esimo nodo



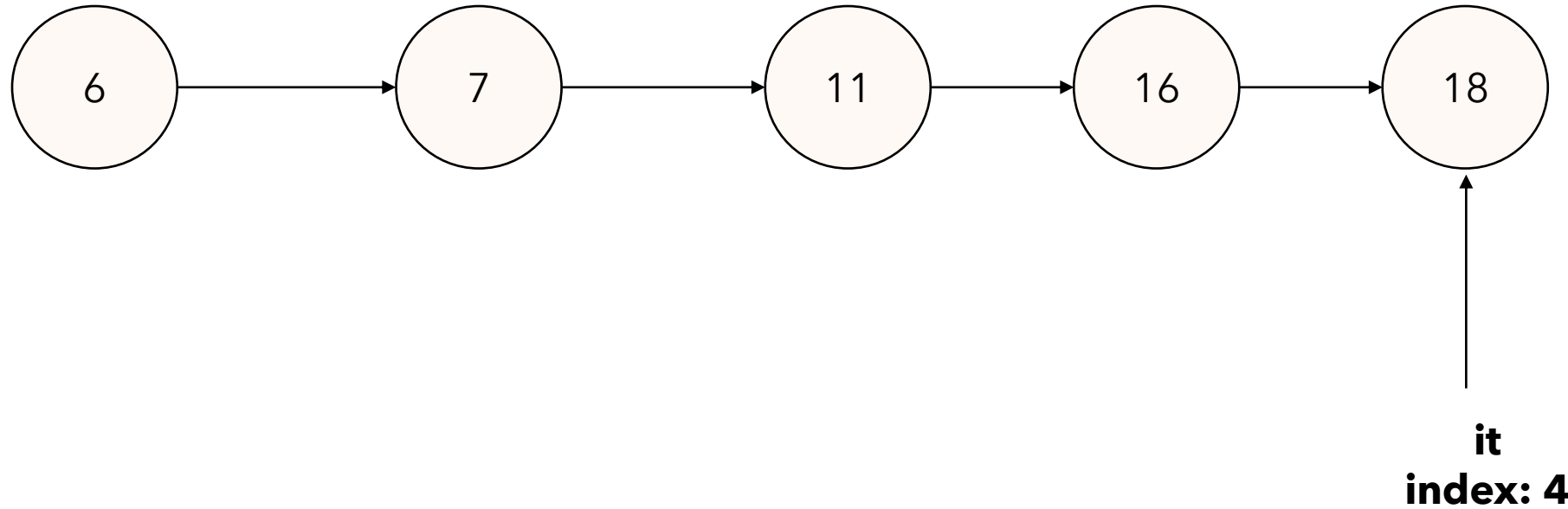
Vogliamo il nodo di indice 5 (non esistente)

Accesso sequenziale all'n-esimo nodo



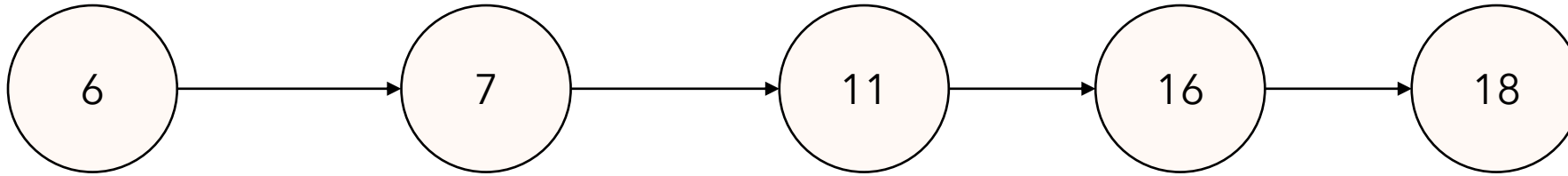
Vogliamo il nodo di indice 5 (non esistente)

Accesso sequenziale all'n-esimo nodo



Vogliamo il nodo di indice 5 (non esistente)

Accesso sequenziale all'n-esimo nodo



it
index: 5
Usciamo dal ciclo, it punta al
nodo cercato, ossia al nodo
che non esiste

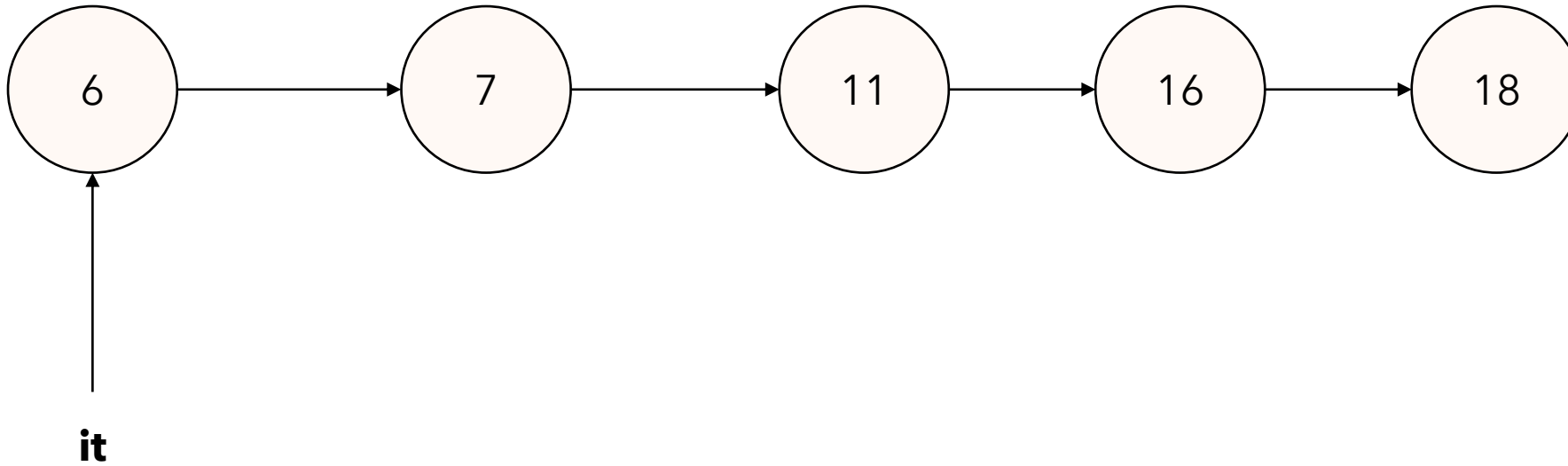
Vogliamo il nodo di indice 5 (non esistente)

Accesso sequenziale all'n-esimo nodo

```
public ListNode getNodeAt(int pos) {  
    ListNode it = this;  
    int index = 0;  
  
    while (index < pos && it != null) {  
        it = it.next;  
        index++;  
    }  
    if (pos < 0) {  
        it = null;  
    }  
  
    return it;  
}
```

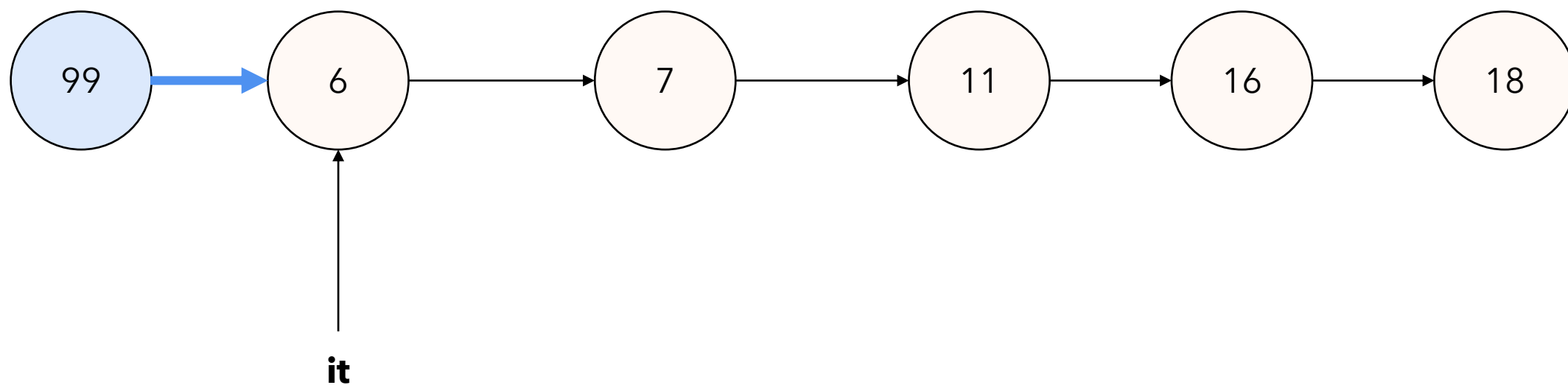
Il metodo ritorna il riferimento null se viene richiesto un indice illegale. È sensato: se chiedi una cosa che non esiste, ti restituisco un riferimento a qualcosa che non esiste.

Inserimento di un nodo in posizione n



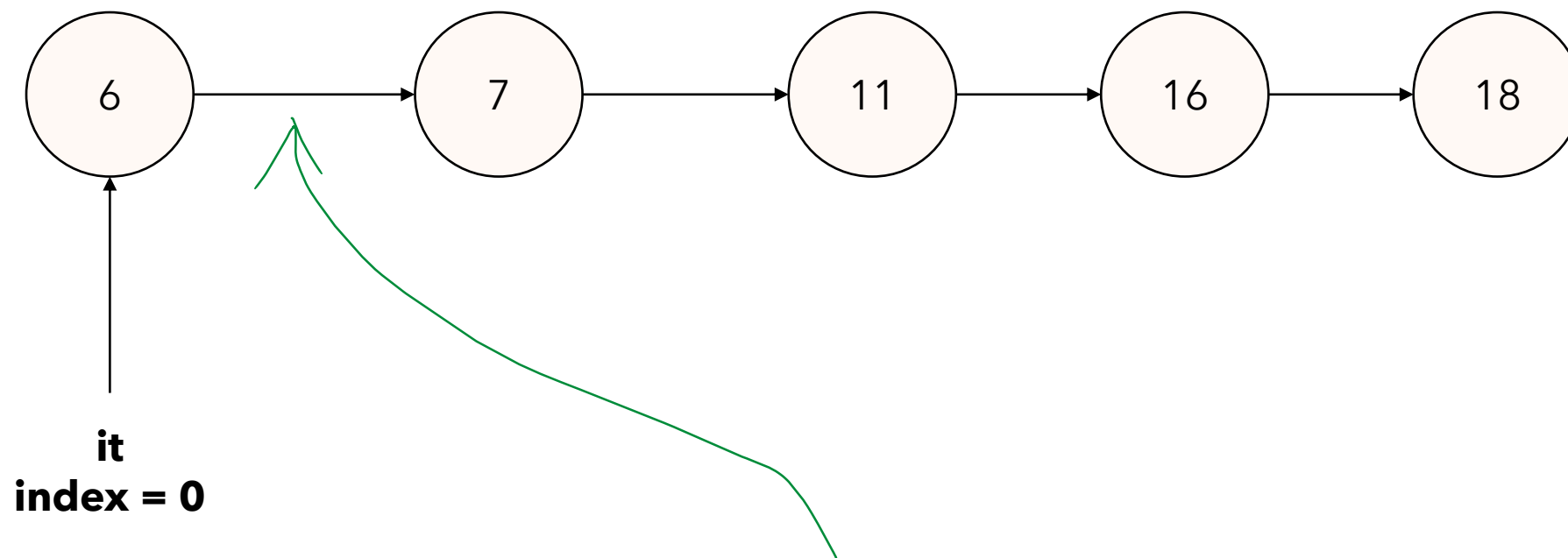
Vogliamo inserire un nuovo nodo con chiave 99 in posizione 0

Inserimento di un nodo in posizione n



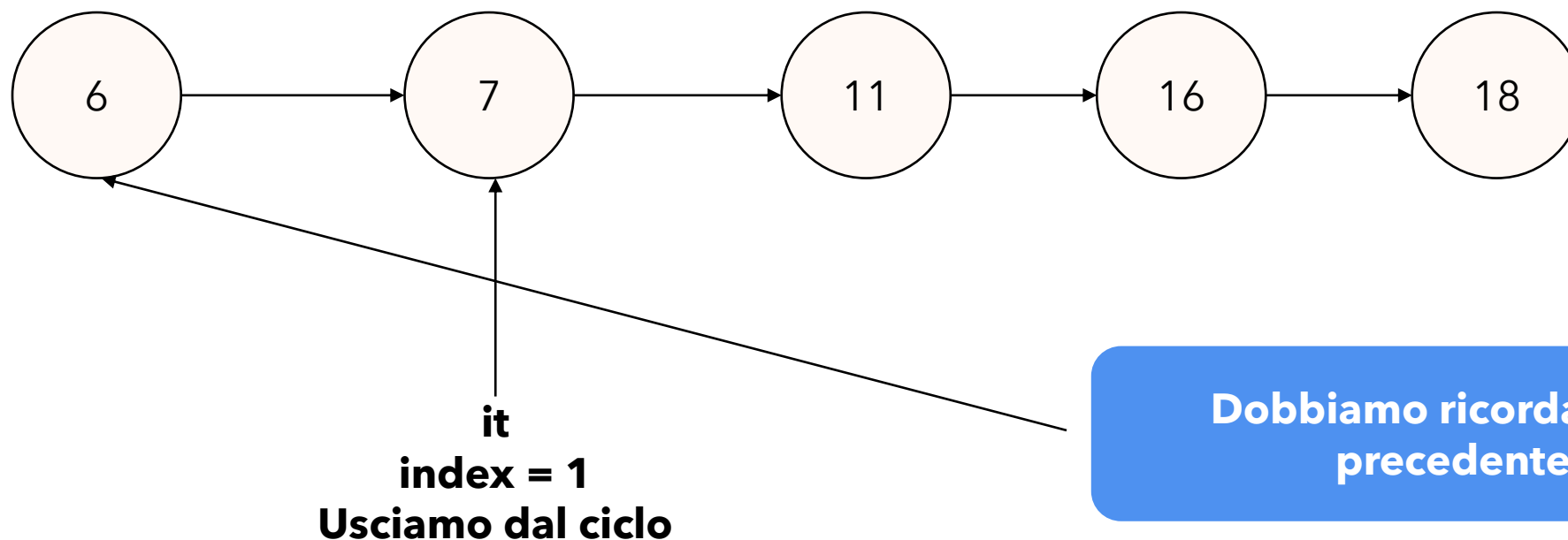
Vogliamo inserire un nuovo nodo in posizione 0
La testa della lista è cambiata. Il metodo dovrà ritornare un riferimento alla nuova testa.

Inserimento di un nodo in posizione n



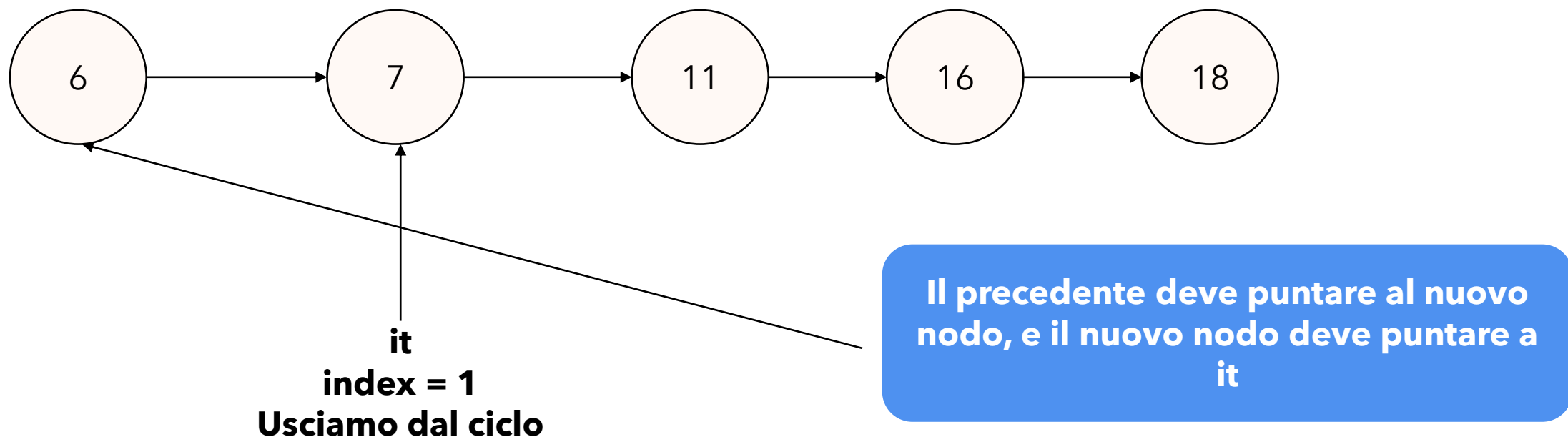
Vogliamo inserire un nuovo nodo con chiave 99 in posizione 1.
Il nodo andrà quindi inserito tra i nodi 0 e 1 (posizioni attuali).
In generale, se un nodo deve essere inserito in posizione n, va inserito tra i nodi che erano in posizione n - 1 e n

Inserimento di un nodo in posizione n



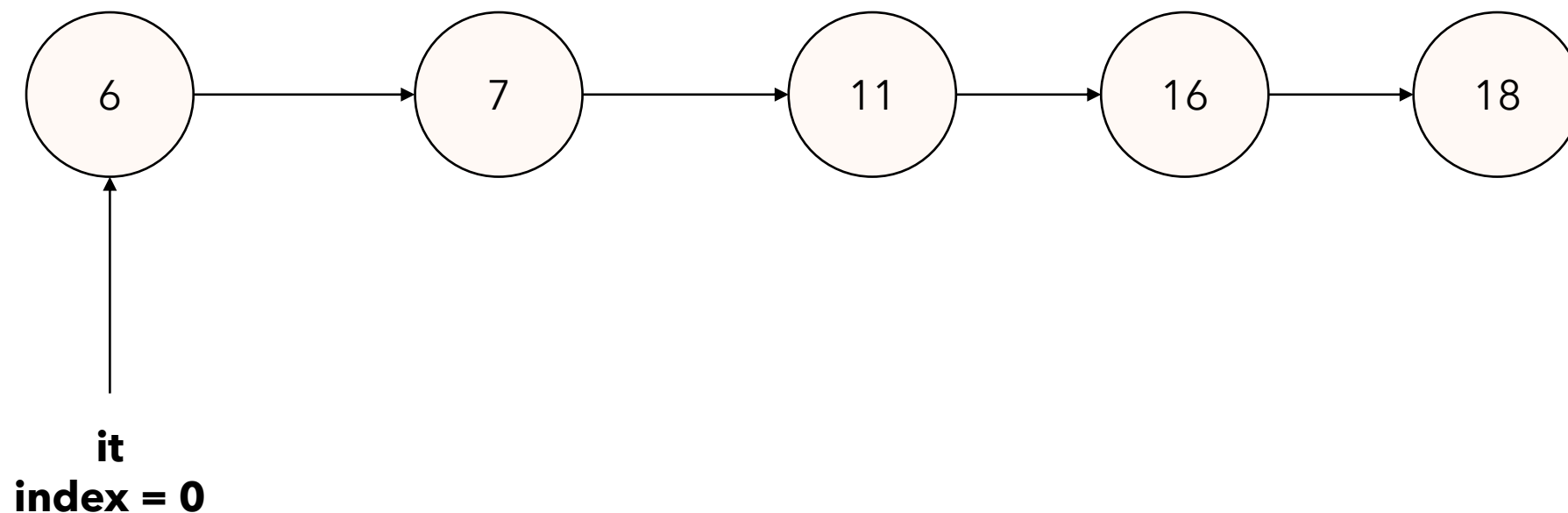
Vogliamo inserire un nuovo nodo con chiave 99 in posizione 1.
Il nodo andrà quindi inserito tra i nodi 0 e 1 (posizioni attuali).
In generale, se un nodo deve essere inserito in posizione n, va inserito tra i nodi che erano in posizione n - 1 e n

Inserimento di un nodo in posizione n



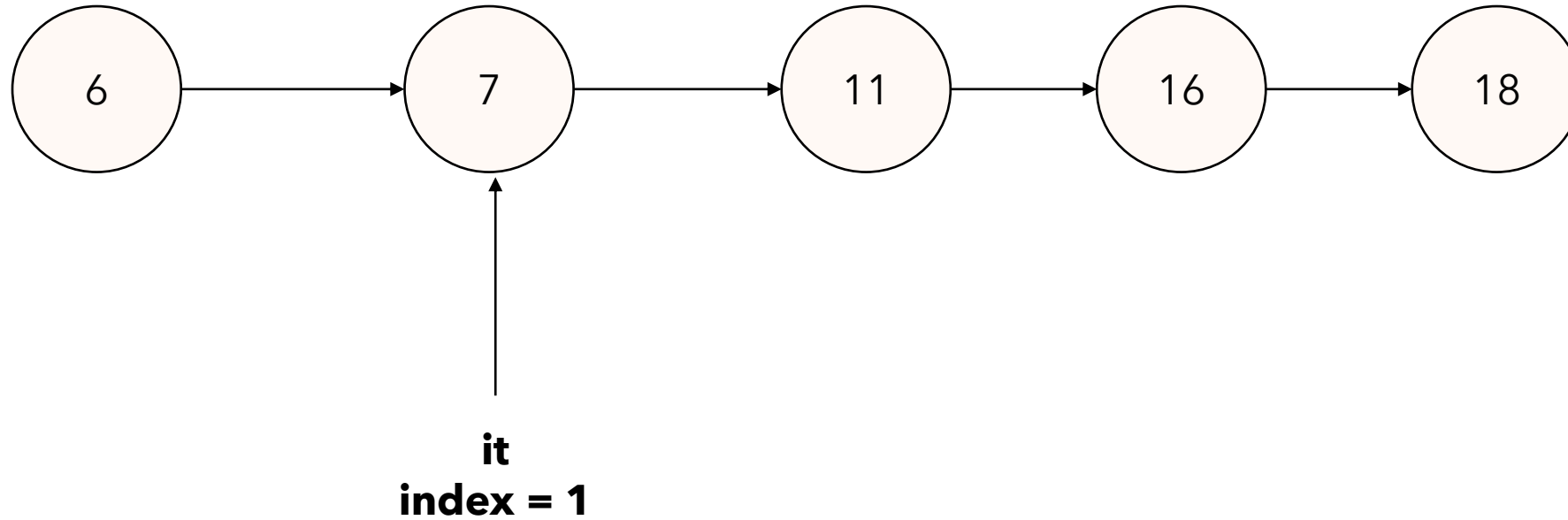
Vogliamo inserire un nuovo nodo con chiave 99 in posizione 1.
Il nodo andrà quindi inserito tra i nodi 0 e 1 (posizioni attuali).
In generale, se un nodo deve essere inserito in posizione n, va inserito tra i nodi che erano in posizione n - 1 e n

Inserimento di un nodo in posizione n



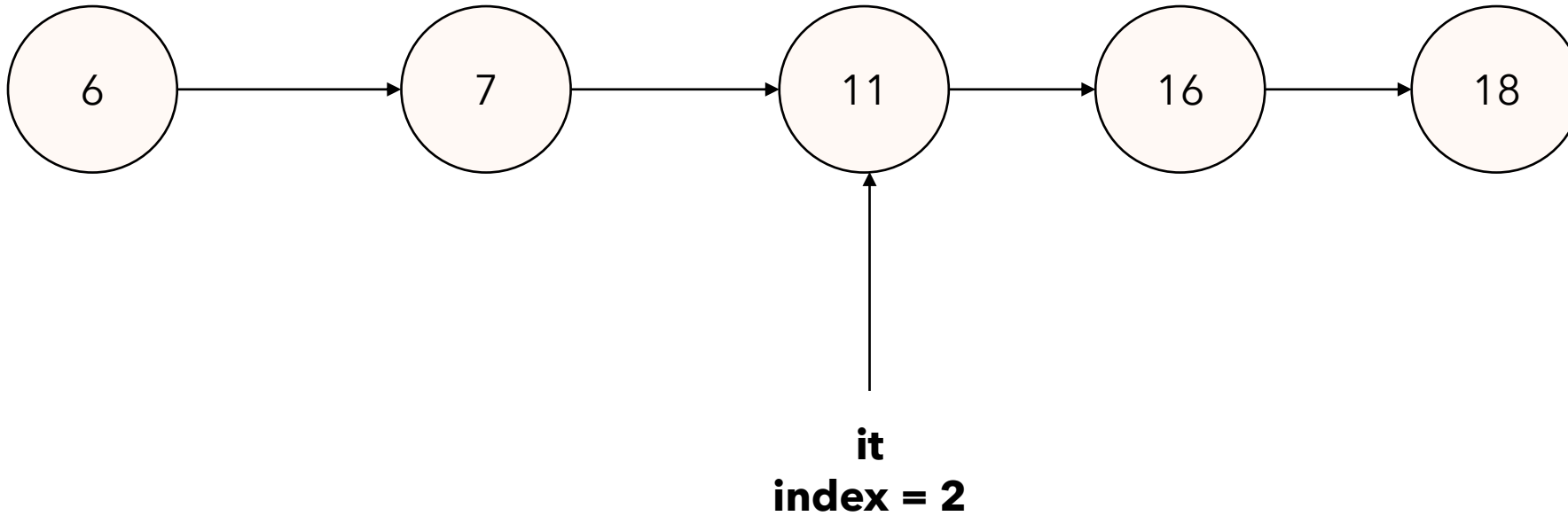
**Vogliamo inserire un nuovo nodo con chiave 99 in posizione 5.
Il nodo andrà quindi inserito dopo l'attuale nodo 4**

Inserimento di un nodo in posizione n



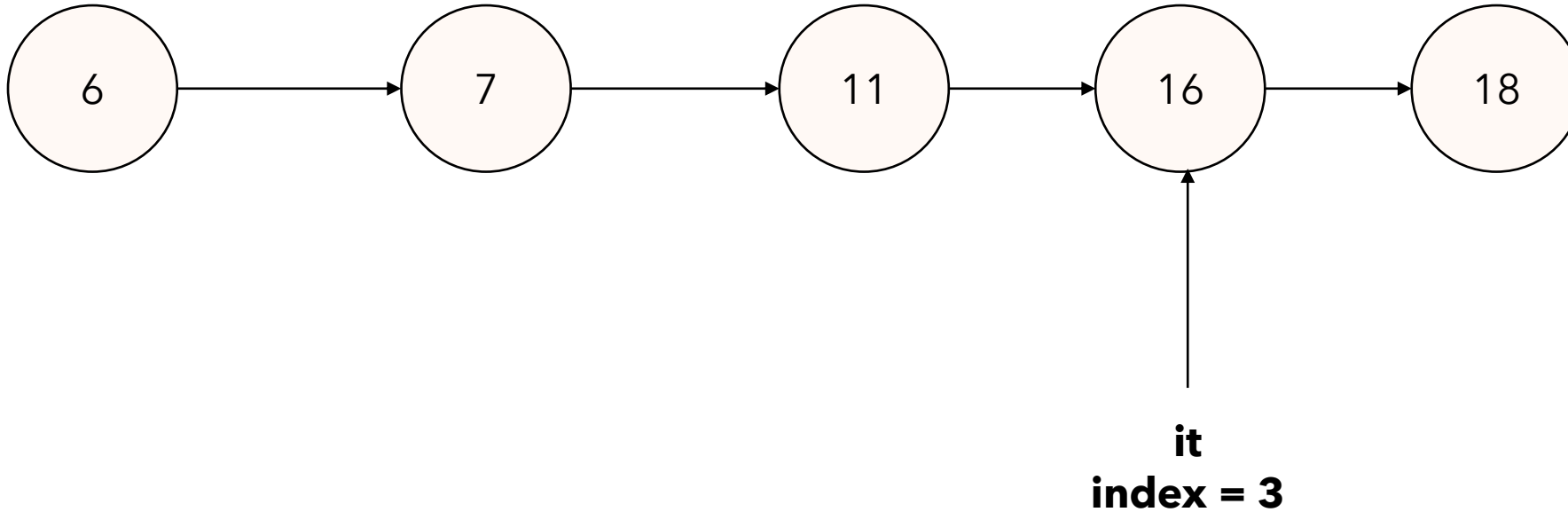
**Vogliamo inserire un nuovo nodo con chiave 99 in posizione 5.
Il nodo andrà quindi inserito dopo l'attuale nodo 4**

Inserimento di un nodo in posizione n



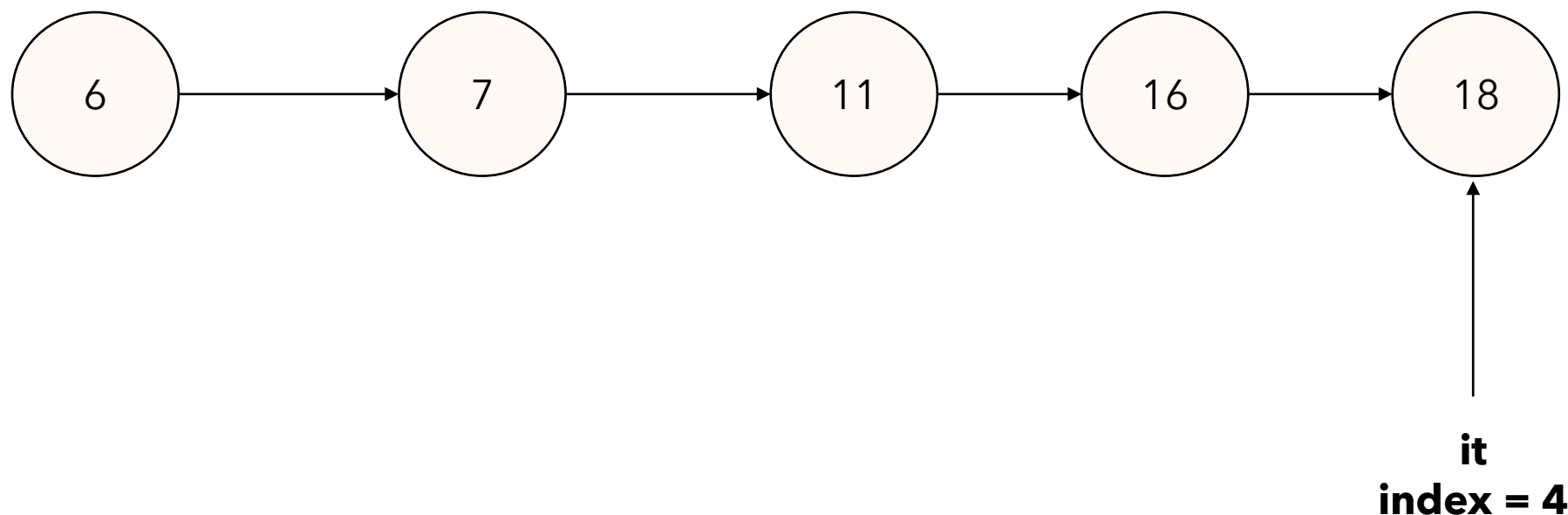
**Vogliamo inserire un nuovo nodo con chiave 99 in posizione 5.
Il nodo andrà quindi inserito dopo l'attuale nodo 4**

Inserimento di un nodo in posizione n



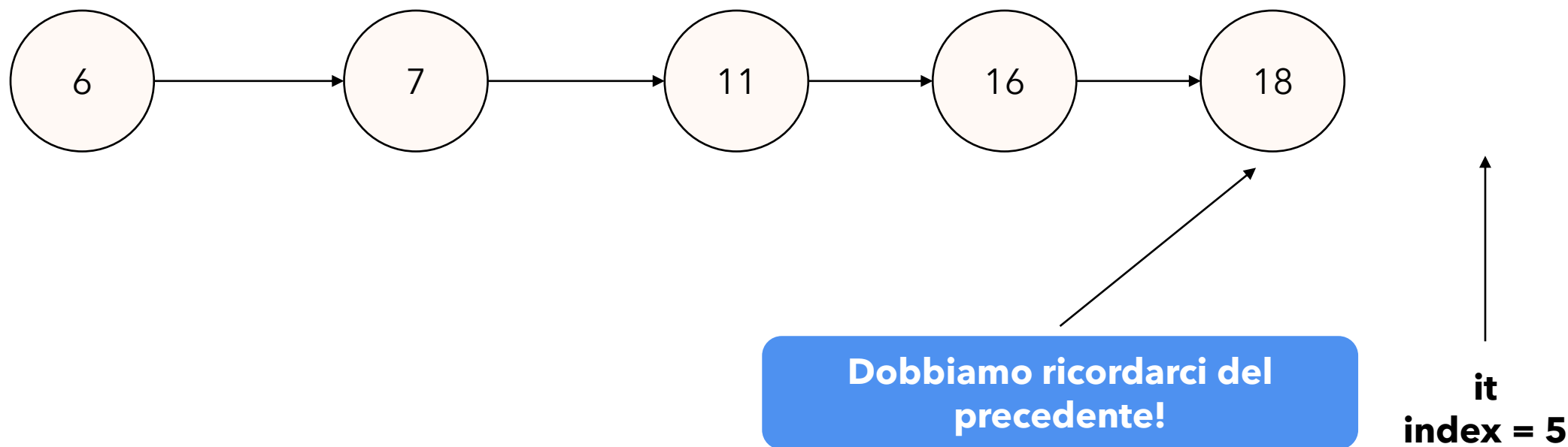
**Vogliamo inserire un nuovo nodo con chiave 99 in posizione 5.
Il nodo andrà quindi inserito dopo l'attuale nodo 4**

Inserimento di un nodo in posizione n



**Vogliamo inserire un nuovo nodo con chiave 99 in posizione 5.
Il nodo andrà quindi inserito dopo l'attuale nodo 4**

Inserimento di un nodo in posizione n



**Vogliamo inserire un nuovo nodo con chiave 99 in posizione 5.
Il nodo andrà quindi inserito dopo l'attuale nodo 4**

Inserimento di un nodo in posizione n

```
public ListNode insertAt(int key, int pos){
    ListNode it = this;
    int index = 0;
    ListNode retNode = this; //node that must be returned

    ListNode previousNode = null;

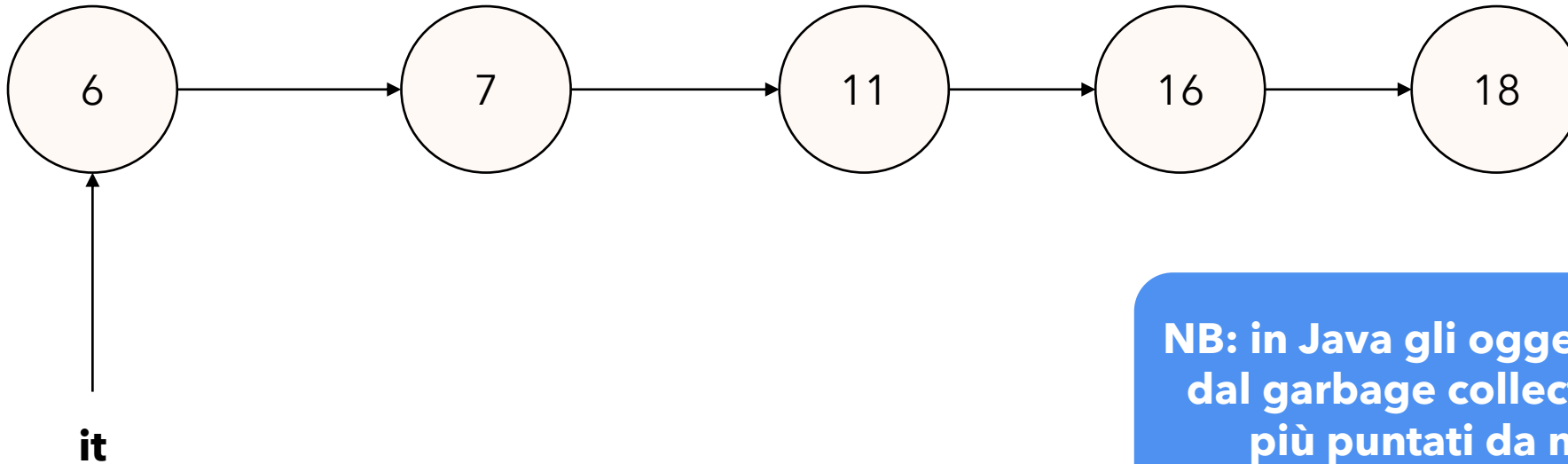
    while (it != null && index < pos) {
        previousNode = it;
        it = it.next;
        index++;
    }

    ListNode newNode = new ListNode(key, it);

    if (pos == 0){
        retNode = newNode;
    }
    else {
        previousNode.next = newNode;
    }

    return retNode;
}
```

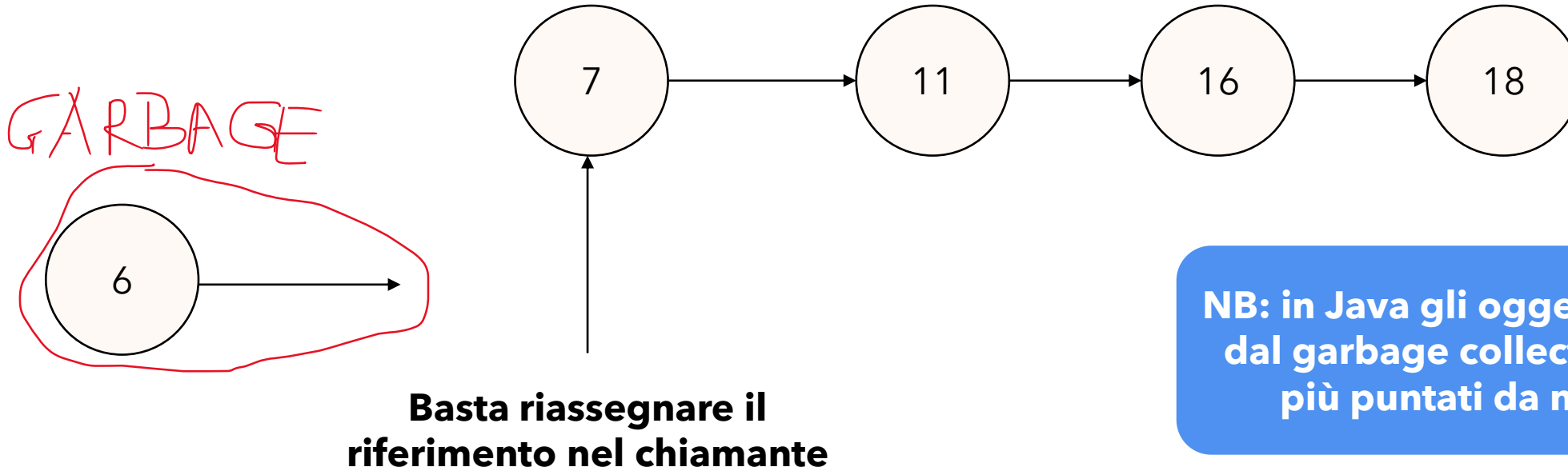
Cancellazione del nodo in posizione n



NB: in Java gli oggetti vengono cancellati dal garbage collector quando non sono più puntati da nessun riferimento

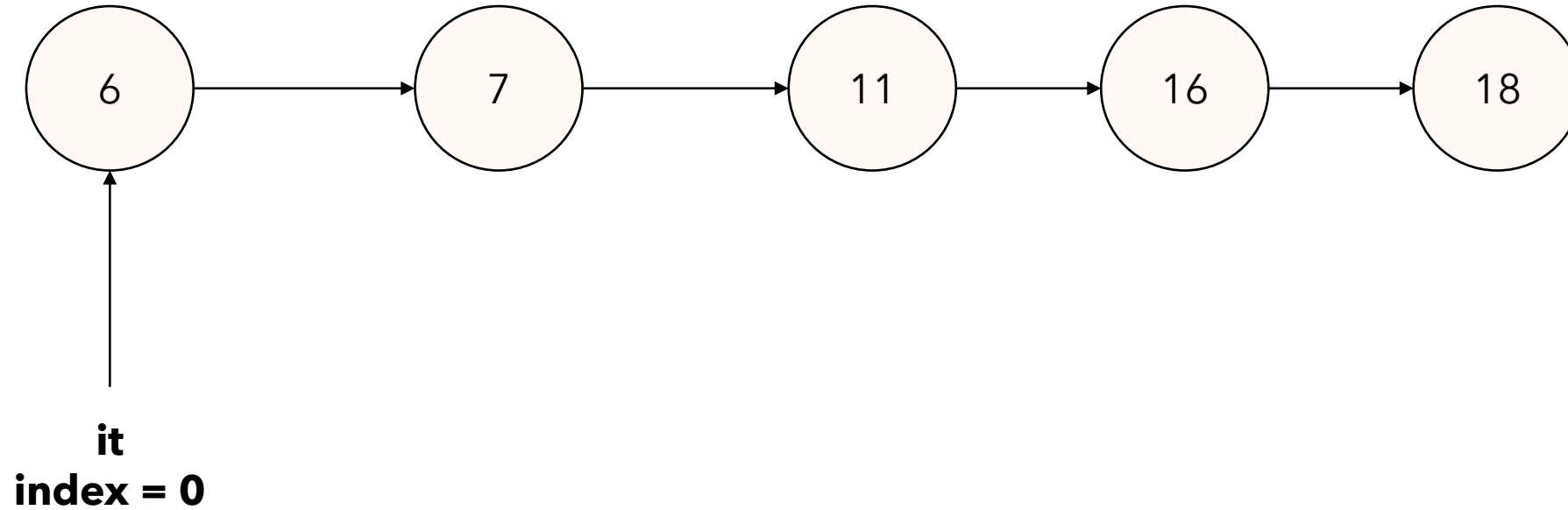
Vogliamo cancellare il nodo in posizione 0

Cancellazione del nodo in posizione n



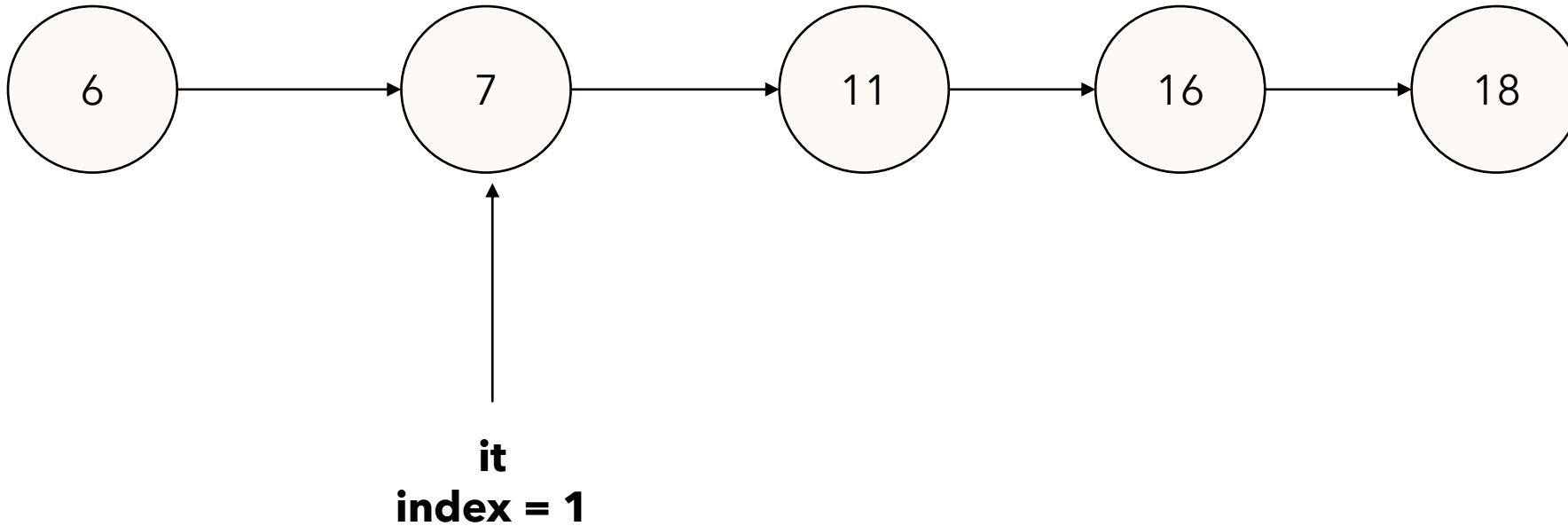
Vogliamo cancellare il nodo in posizione 0

Cancellazione del nodo in posizione n



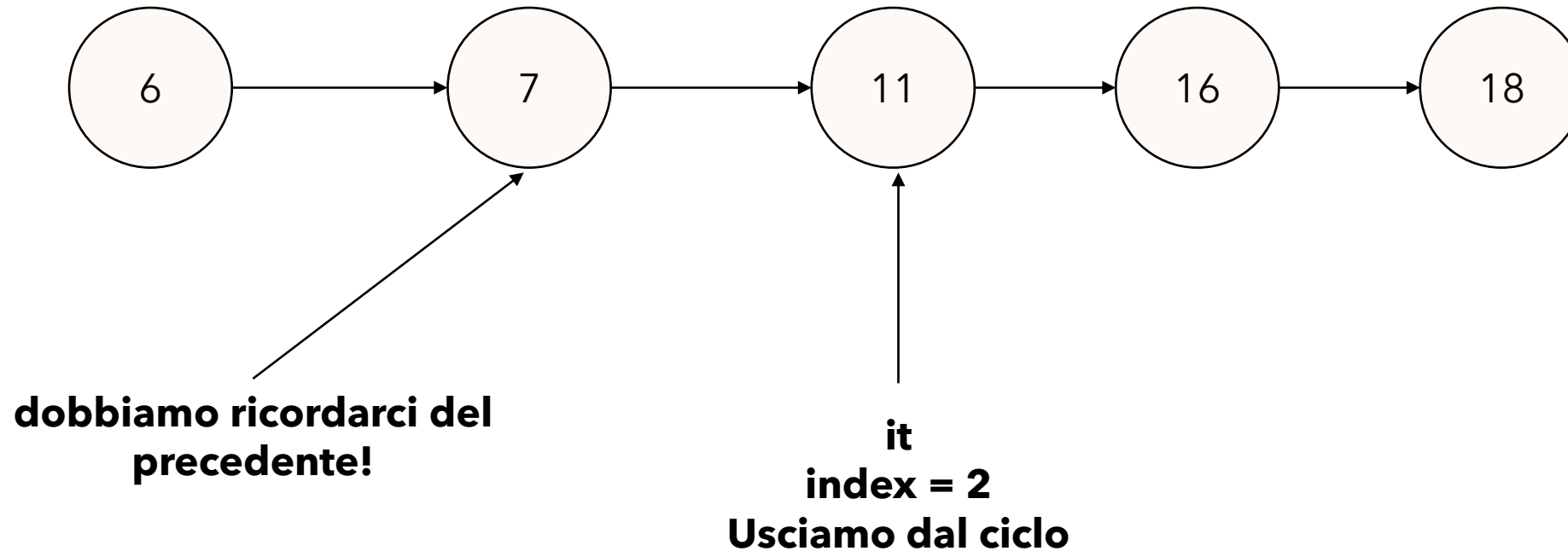
Vogliamo cancellare il nodo in posizione 2

Cancellazione del nodo in posizione n



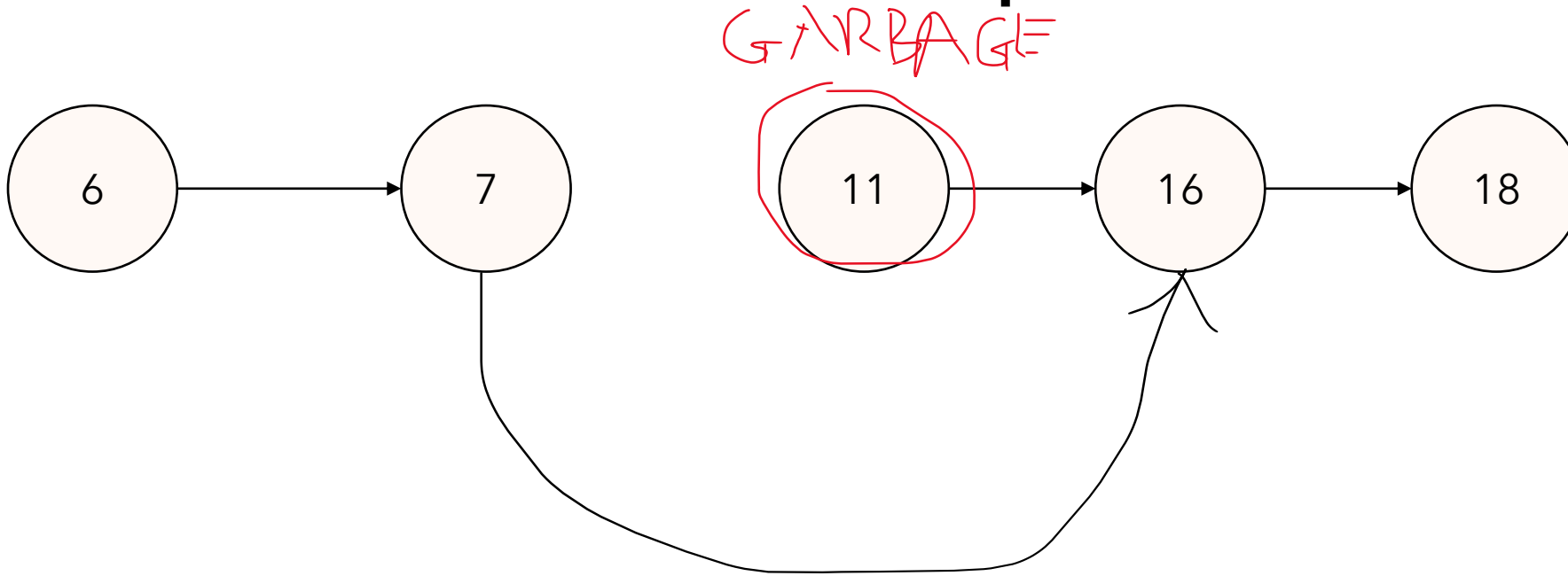
Vogliamo cancellare il nodo in posizione 2

Cancellazione del nodo in posizione n



Vogliamo cancellare il nodo in posizione 2

Cancellazione del nodo in posizione n



Vogliamo cancellare il nodo in posizione 2

Cancellazione del nodo in posizione n

```
public ListNode deleteAt (int pos){
    ListNode iterator = this;
    ListNode previousNode = null;
    int index = 0;
    ListNode retNode = this;
    while (iterator != null && index < pos) {
        previousNode = iterator;
        index++;
        iterator = iterator.next;
    }
    if (pos == 0) {
        retNode = this.next;
    }
    else if (pos < 0) {
        retNode = this;
    }
    else {
        //if didn't pass the end
        if (iterator != null) {
            previousNode.next = iterator.next;
        }
    }
    return retNode;
}
```

La classe LinkedList di java.util

```
LinkedList<String> list1 = new LinkedList<String>();  
list1.add("F");  
list1.add("B");  
list1.add("D");  
list1.add("E");  
list1.add("C");  
list1.addLast("Z");  
list1.addFirst("A");  
System.out.println("Content of list1: " + list1);  
list1.remove(0);  
System.out.println("Content of list1: " + list1);
```