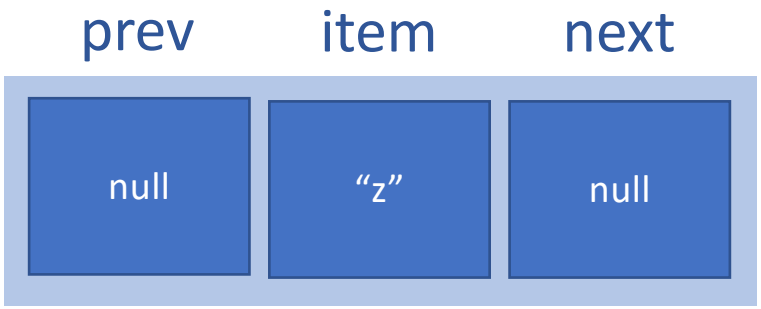
Comprendre les structures chaînées

Base de l'implémentation d'une LinkedList

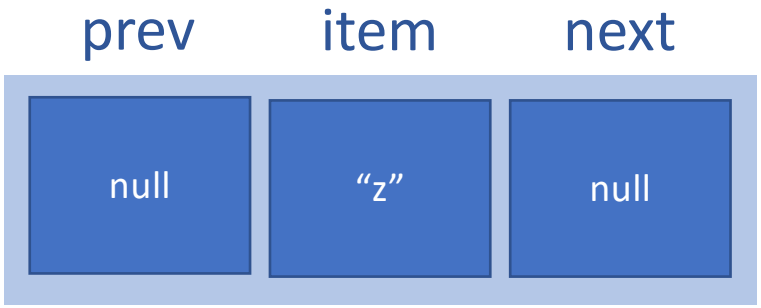
Université de Montpellier
Faculté des sciences
Mars 2020

Une chaîne constituée de noeuds successifs

- La structure de base est un NOEUD qui va porter une des valeurs que l'on veut stocker dans la liste
- Ici on veut stocker "z"

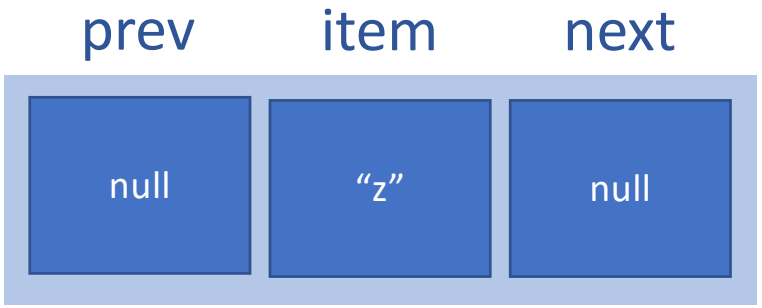


Une chaîne constituée de noeuds successifs



- Le nœud (parfois appelé cellule)
- est de type MyNode pour se rapprocher du nom Node de l’API Java
- contient :
  - Un item (la valeur stockée)
  - Une reference vers le nœud précédent (prev)
  - Une référence vers le nœud suivant (next)

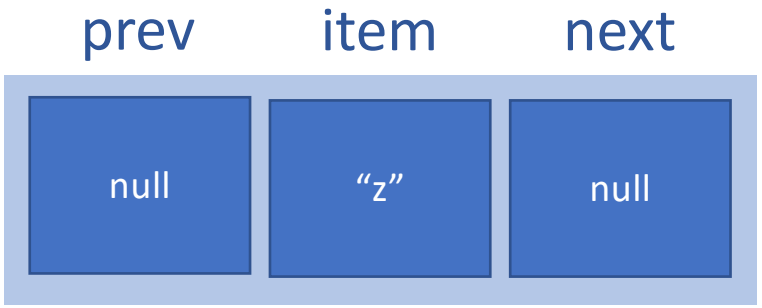
Une chaîne constituée de nœuds successifs




```
public class MyNode<E> {
        E item;
        MyNode<E> next;
        MyNode<E> prev;

    MyNode(MyNode<E> prev, E element, MyNode<E> next) {
        this.item = element; this.next = next; this.prev = prev;
    }
}
```

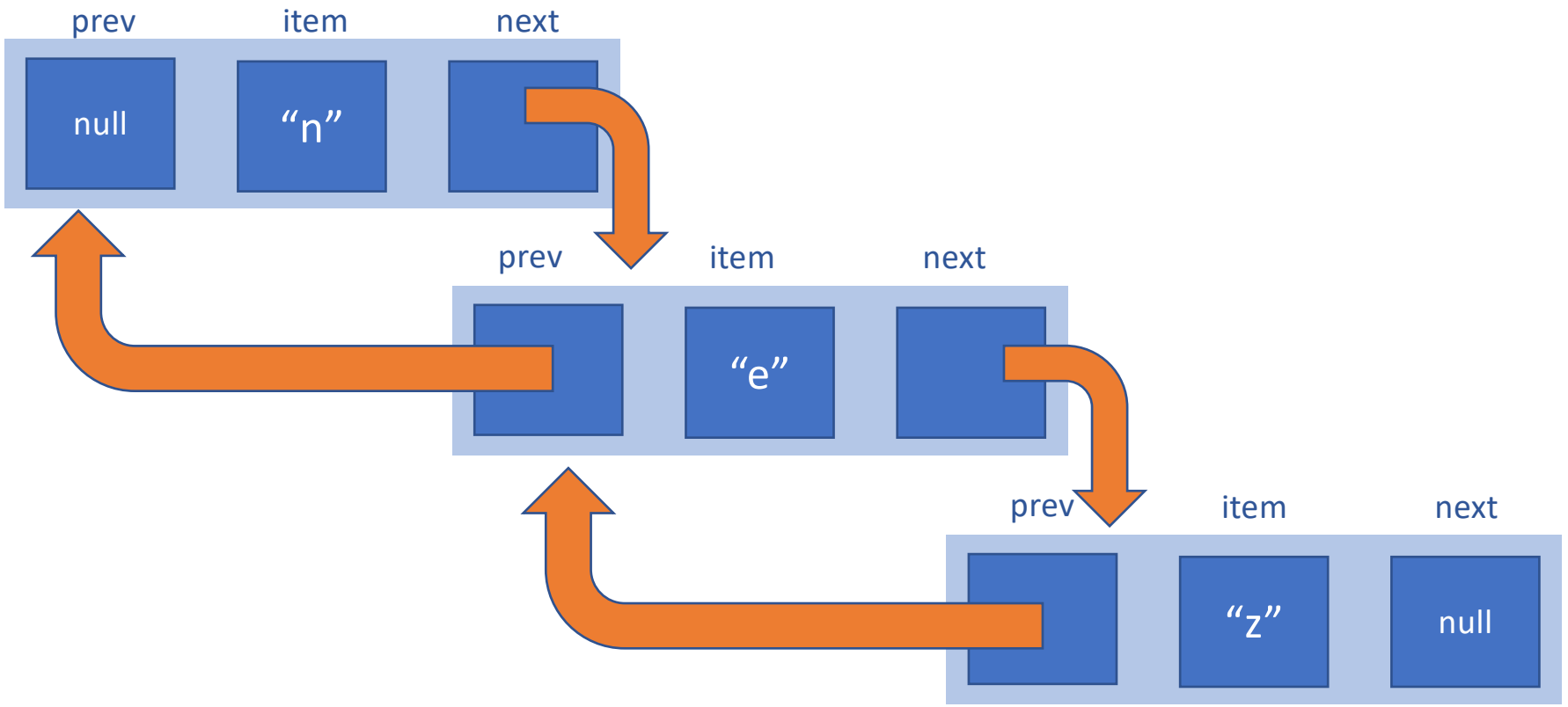
Une chaîne constituée de nœuds successifs



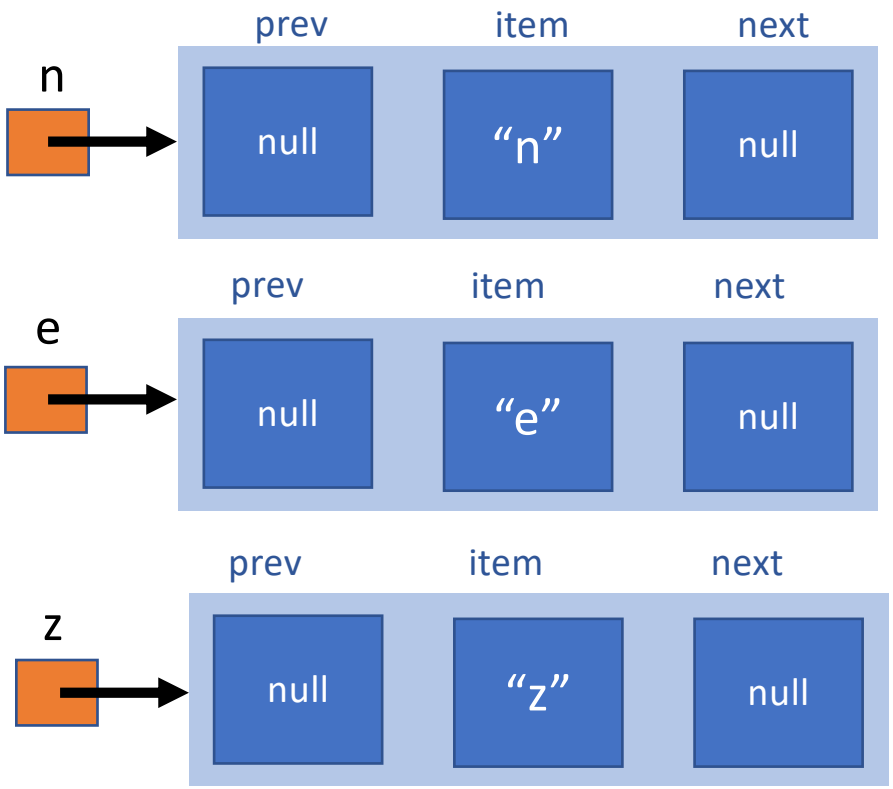

```
public class MyNode<E> {
        E item;
        MyNode<E> next;
        MyNode<E> prev;
}
```

Les attributs ne sont pas « private » pour que toute autre classe du même package puisse y accéder et simplifier le code de cette introduction aux structures chaînées (alternativement : classe imbriquée)

Une chaîne connectant des nœuds successifs



Une chaîne connectant des noeuds successifs



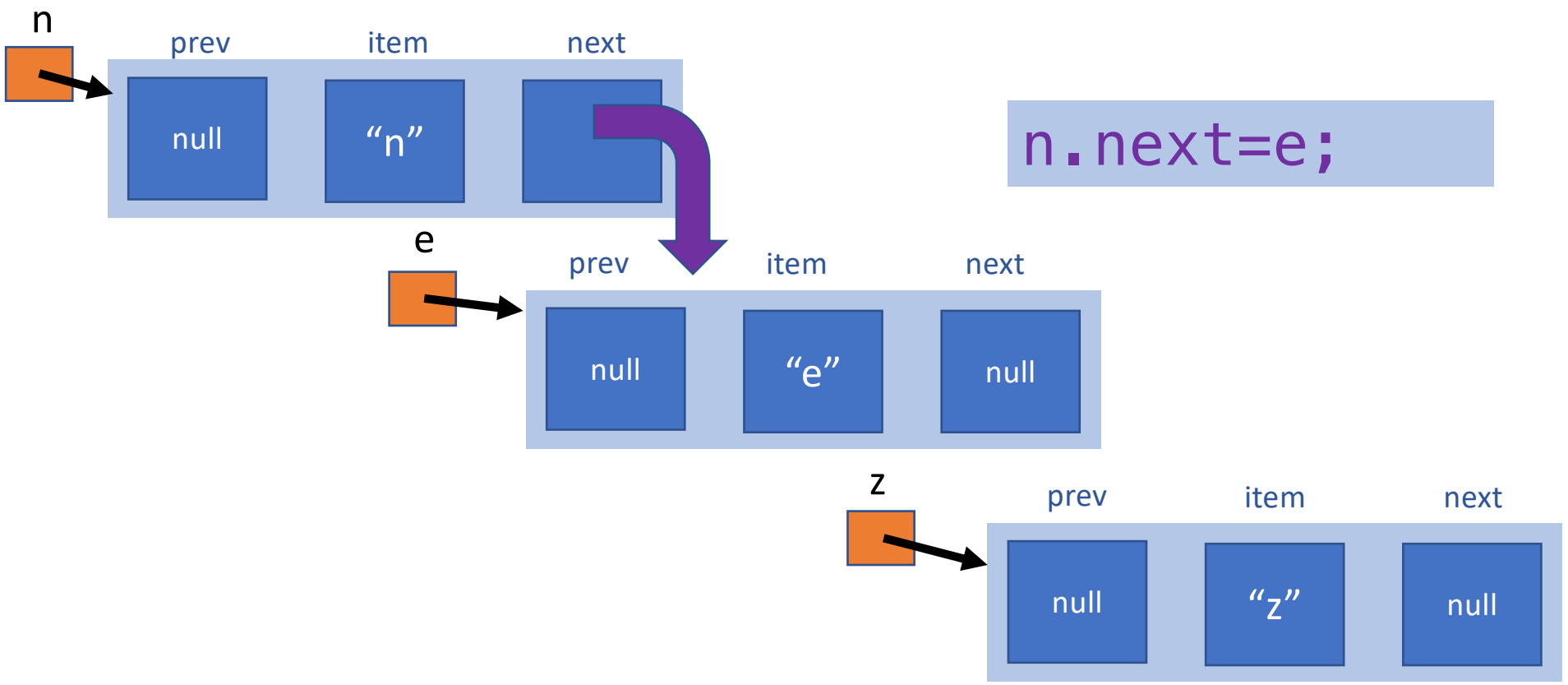
Création des nœuds

```
MyNode<String> n = new
MyNode<String>(null, "n", null);
```

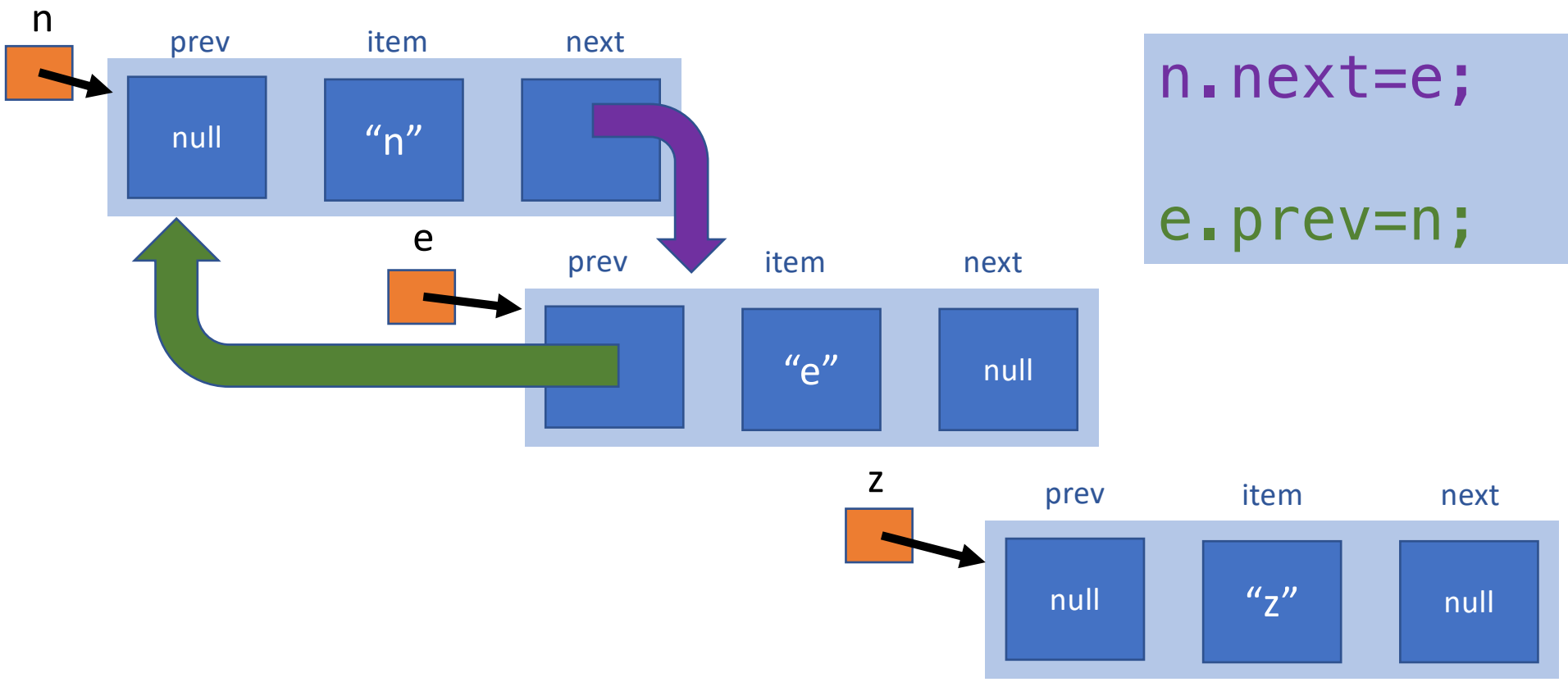
```
MyNode<String> e = new
MyNode<String>(null, "e", null);
```

```
MyNode<String> z = new
MyNode<String>(null, "z", null);
```

Connection des deux premiers noeuds



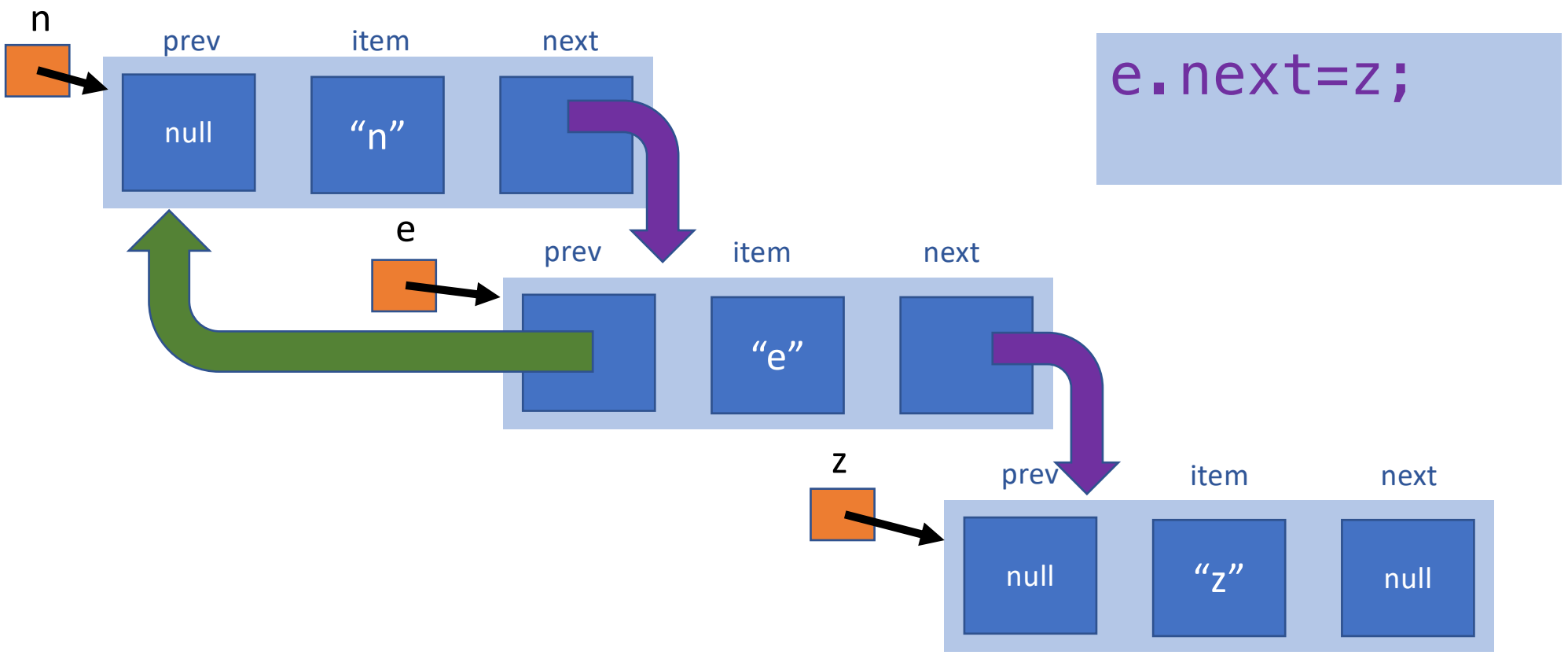
Connection des deux premiers noeuds



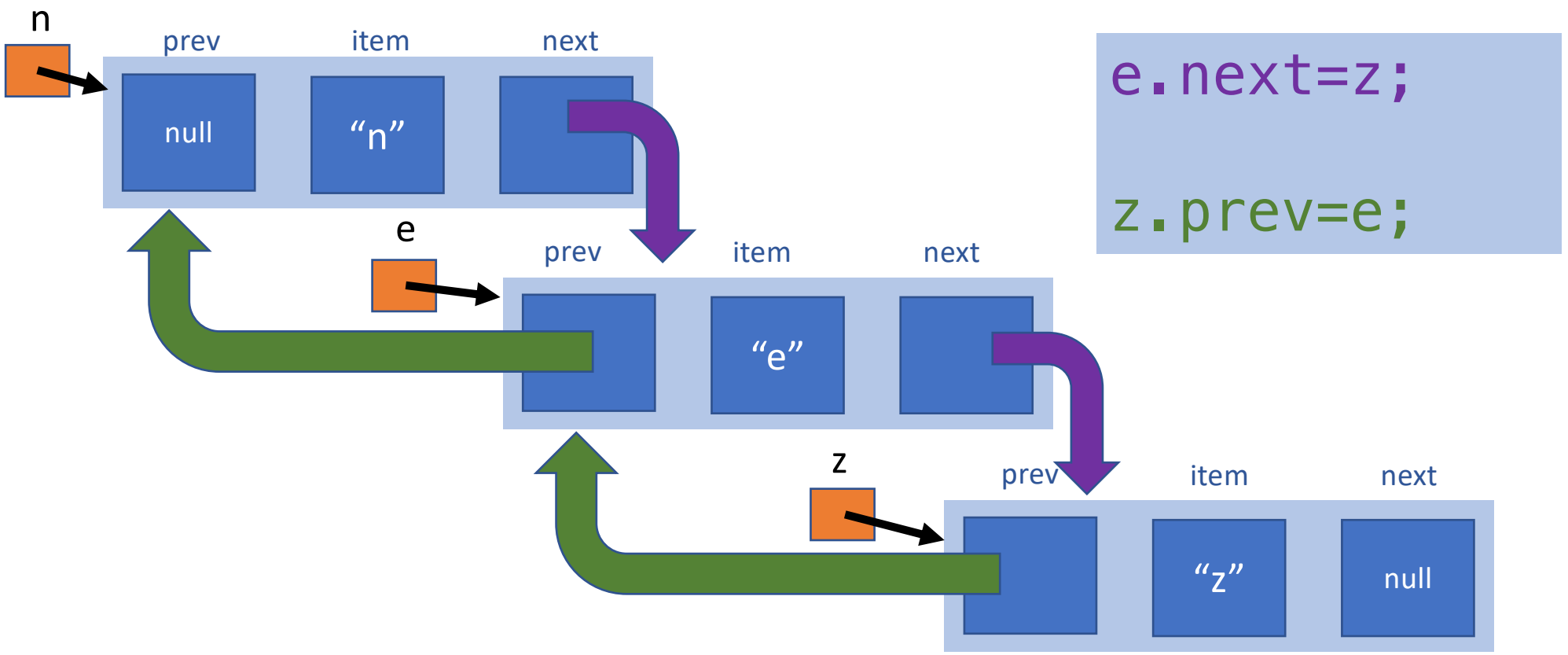

```
n.next=e;

e.prev=n;
```

Connection des deux derniers noeuds

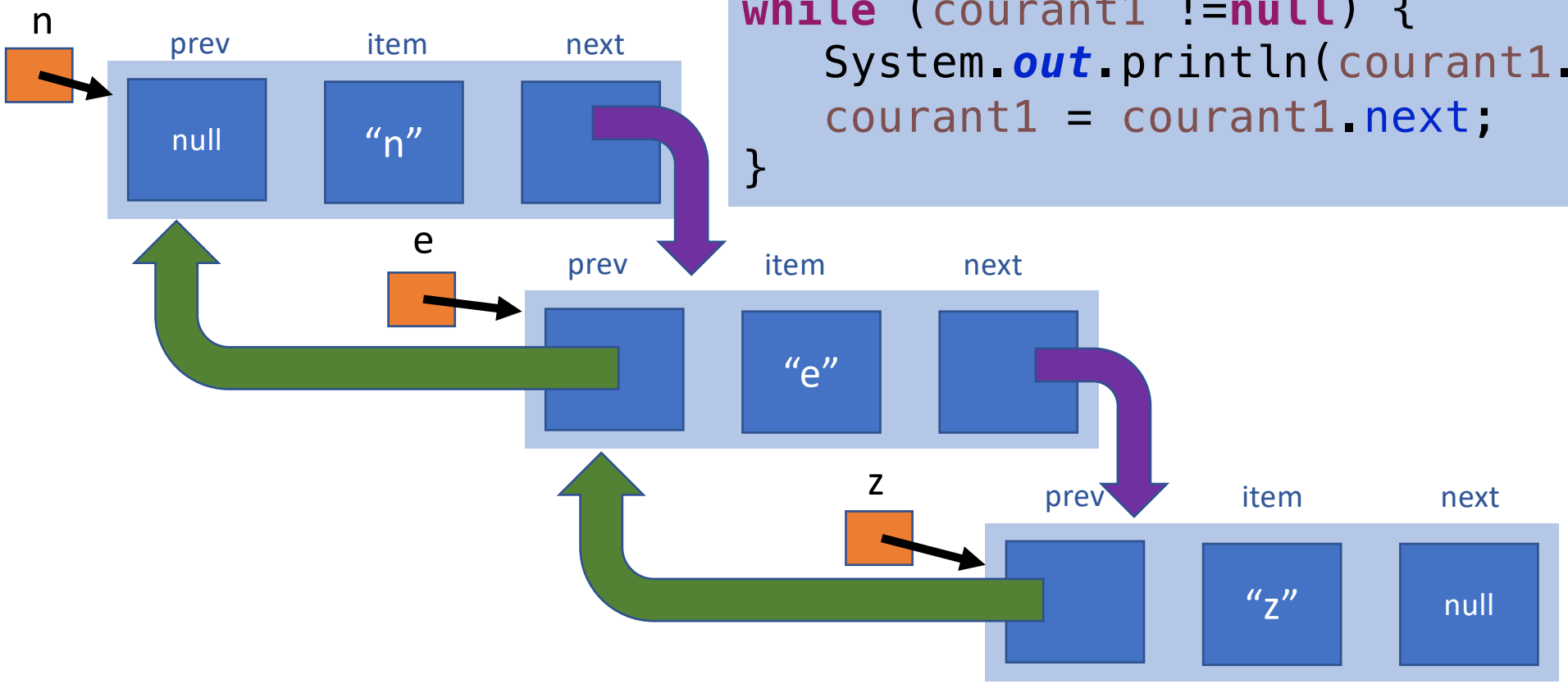


Connection des deux derniers noeuds

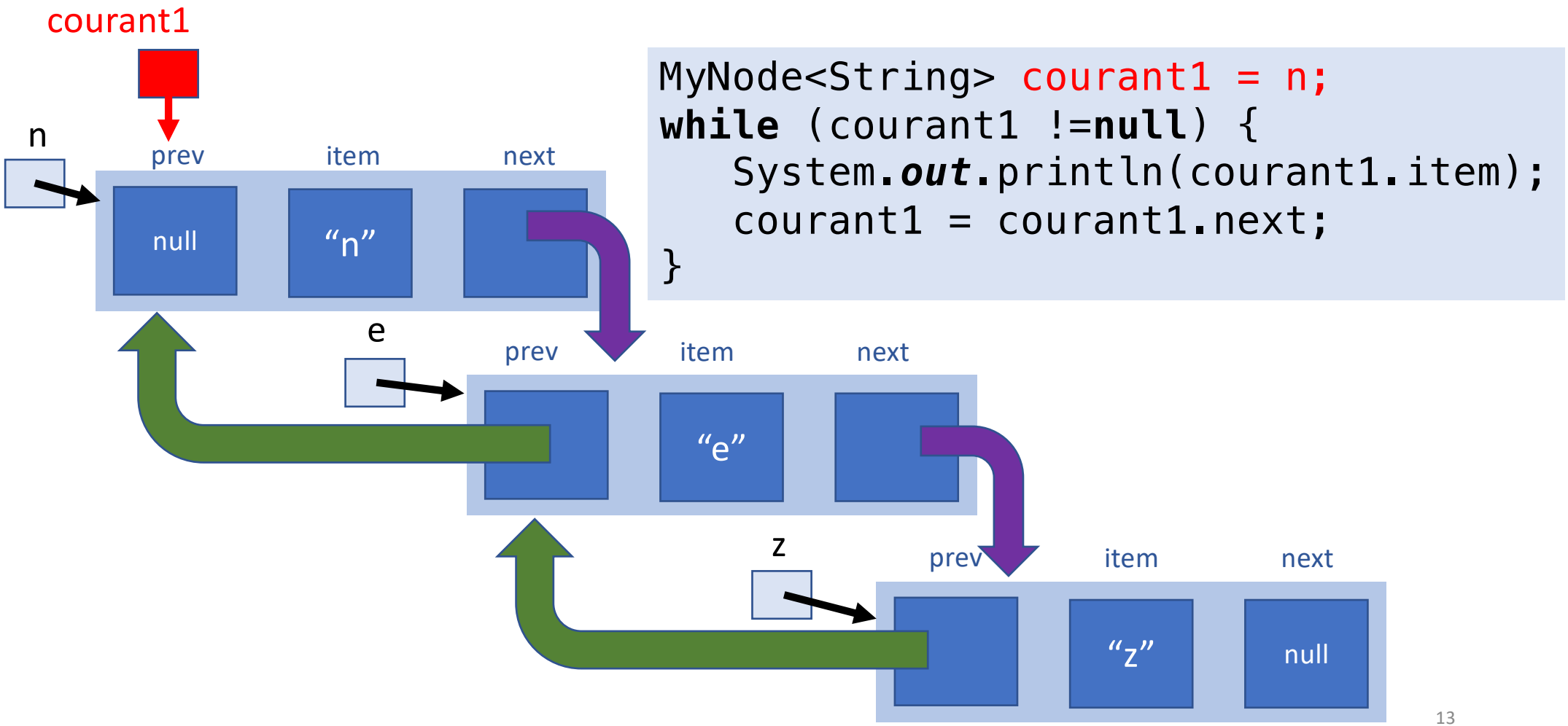


Parcourir la structure

```
MyNode<String> courant1 = n;
while (courant1 !=null) {
    System.out.println(courant1.item);
    courant1 = courant1.next;
}
```

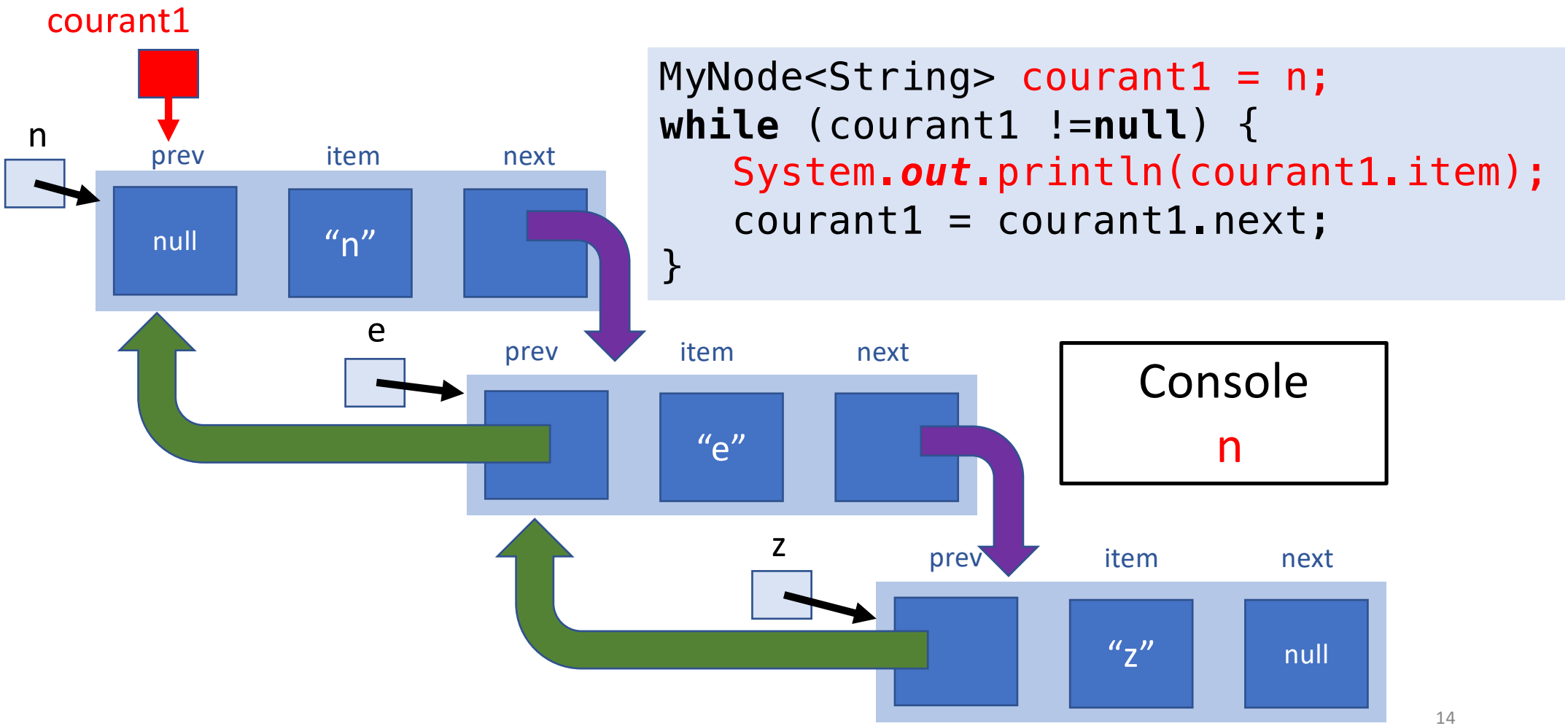


Parcourir la structure : initialisation




```
MyNode<String> courant1 = n;
while (courant1 !=null) {
    System.out.println(courant1.item);
    courant1 = courant1.next;
}
```

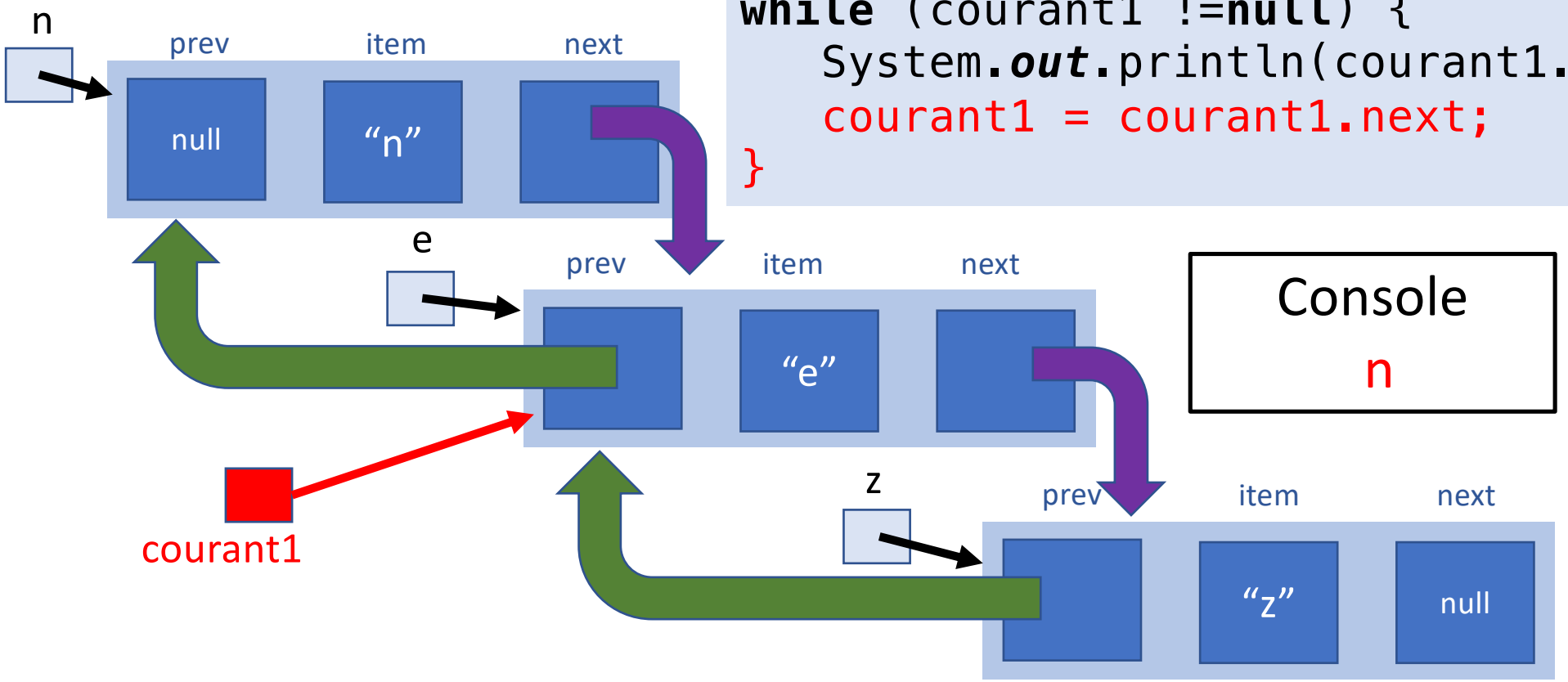
Parcourir la structure : affichage de la valeur




```
MyNode<String> courant1 = n;
while (courant1 !=null) {
    System.out.println(courant1.item);
    courant1 = courant1.next;
}
```

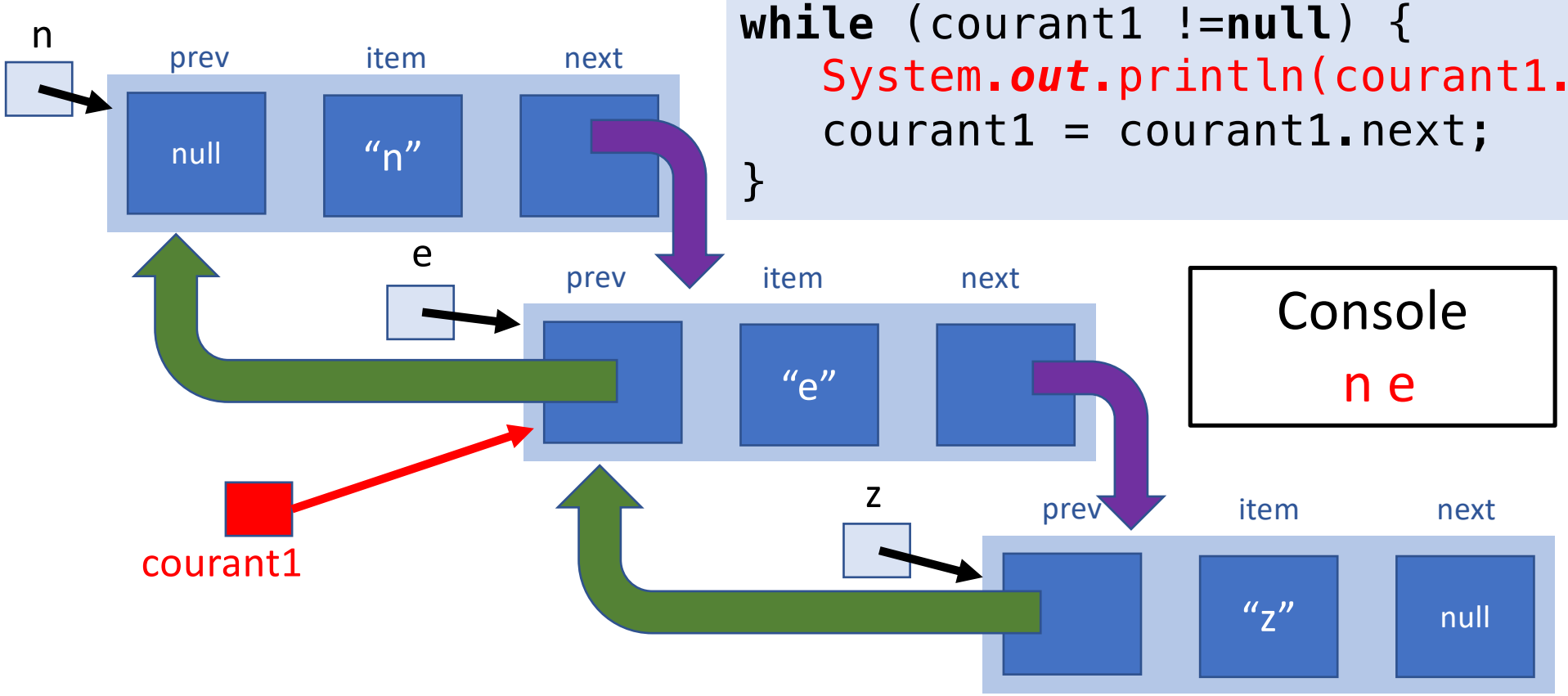
Parcourir la structure : passer au suivant

```
MyNode<String> courant1 = n;
while (courant1 !=null) {
    System.out.println(courant1.item);
    courant1 = courant1.next;
}
```



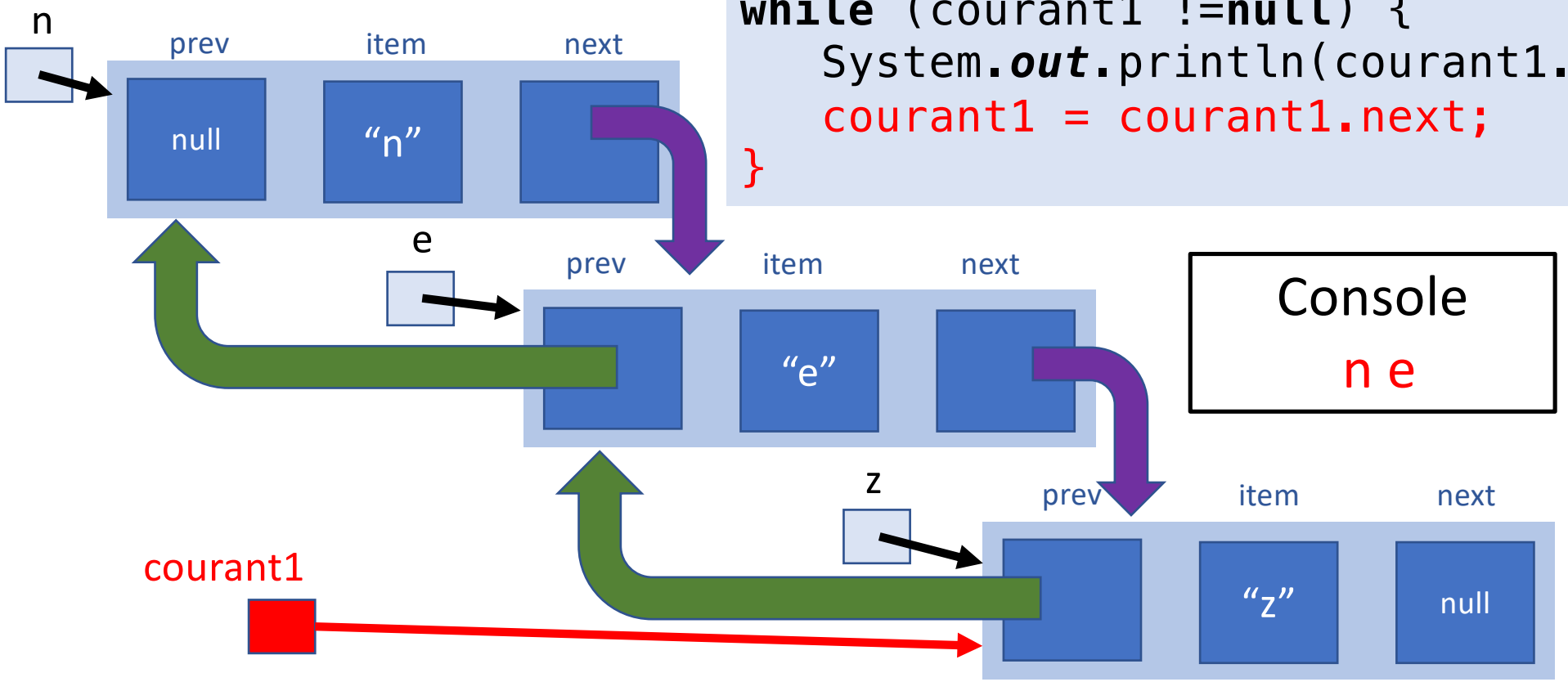
Parcourir la structure : affichage de la valeur

```
MyNode<String> courant1 = n;
while (courant1 !=null) {
    System.out.println(courant1.item);
    courant1 = courant1.next;
}
```



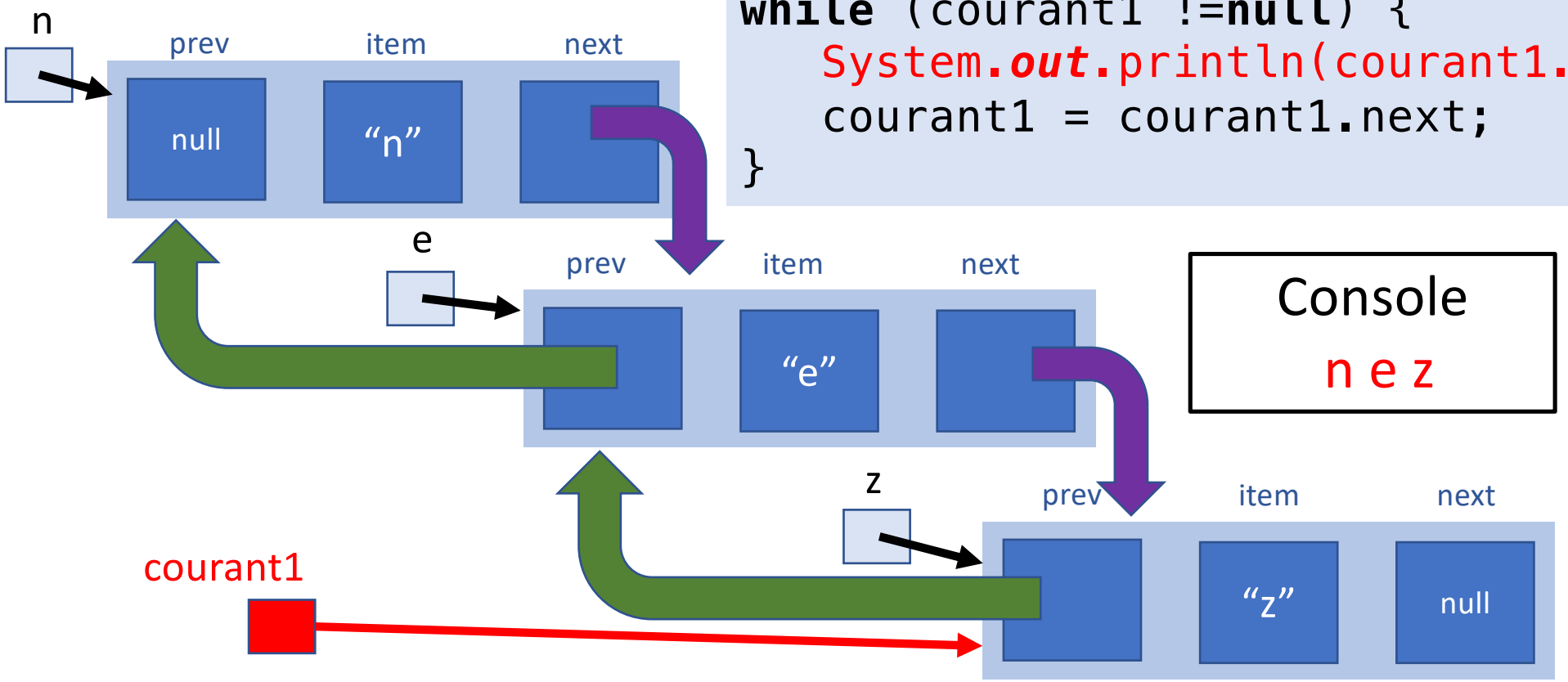
Parcourir la structure : passage au suivant

```
MyNode<String> courant1 = n;
while (courant1 !=null) {
    System.out.println(courant1.item);
    courant1 = courant1.next;
}
```



Parcourir la structure : affichage de la valeur

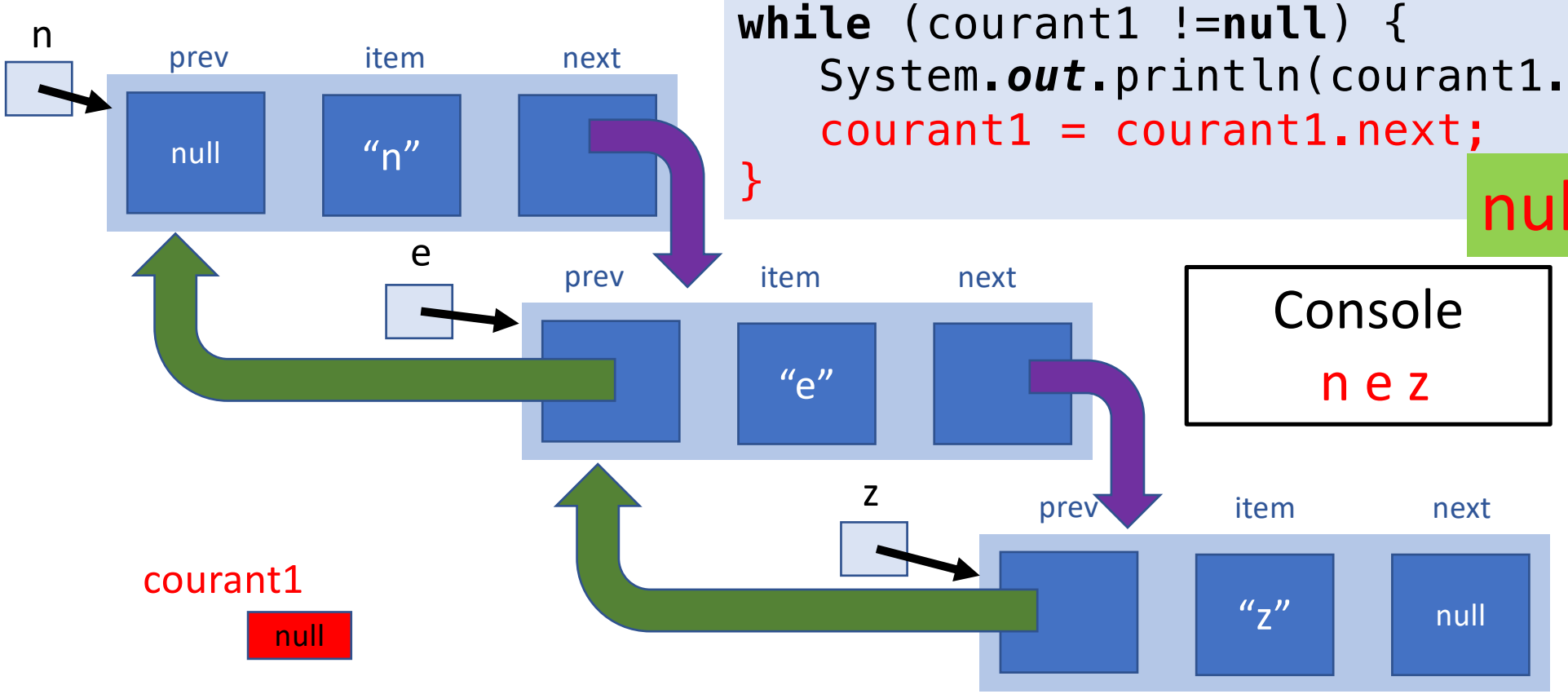
```
MyNode<String> courant1 = n;
while (courant1 !=null) {
    System.out.println(courant1.item);
    courant1 = courant1.next;
}
```



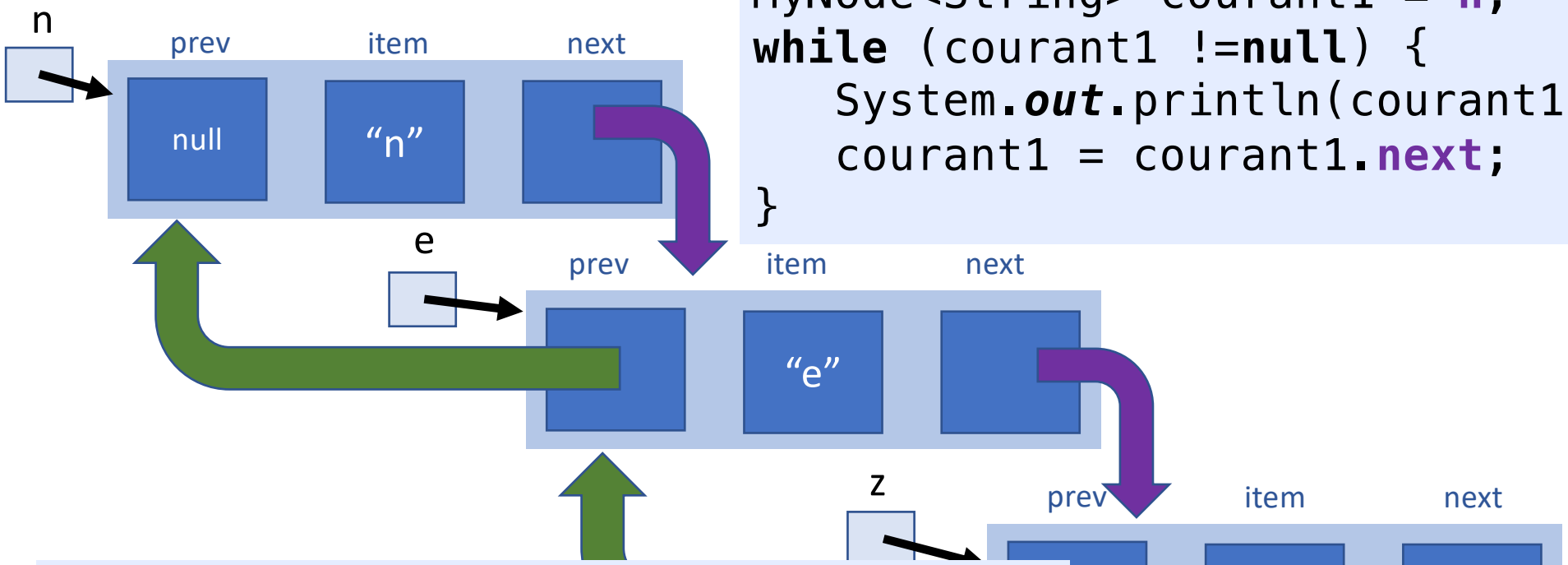
Parcourir la structure : fin de l'itération

```
MyNode<String> courant1 = n;
while (courant1 !=null) {
    System.out.println(courant1.item);
    courant1 = courant1.next;
}
```

null



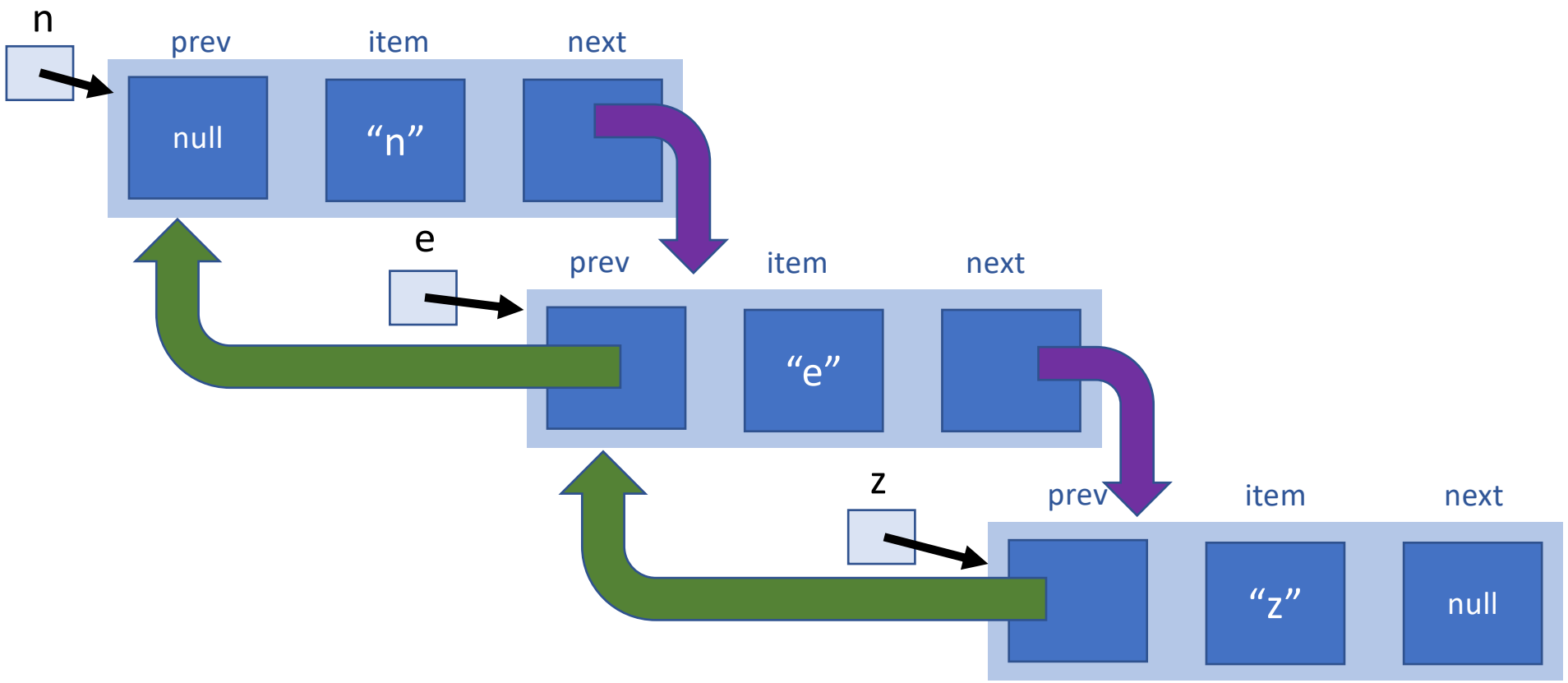
Parcourir la structure en sens inverse



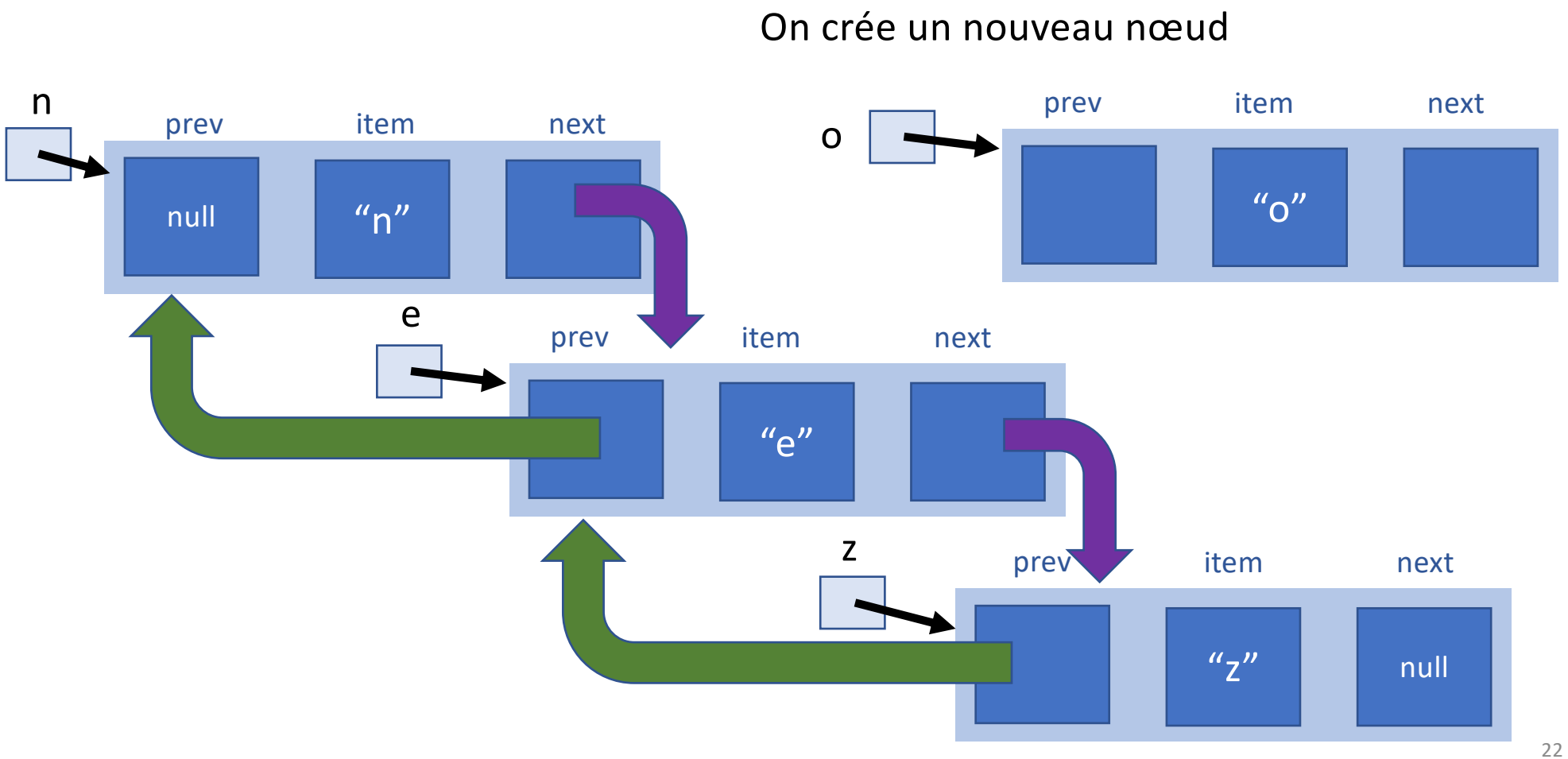

```
MyNode<String> courant1 = n;
while (courant1 !=null) {
    System.out.println(courant1.item);
    courant1 = courant1.next;
}
```

```
MyNode<String> courant2 = z;
while (courant2 !=null) {
    System.out.println(courant2.item);
    courant2 = courant2.prev;
}
```

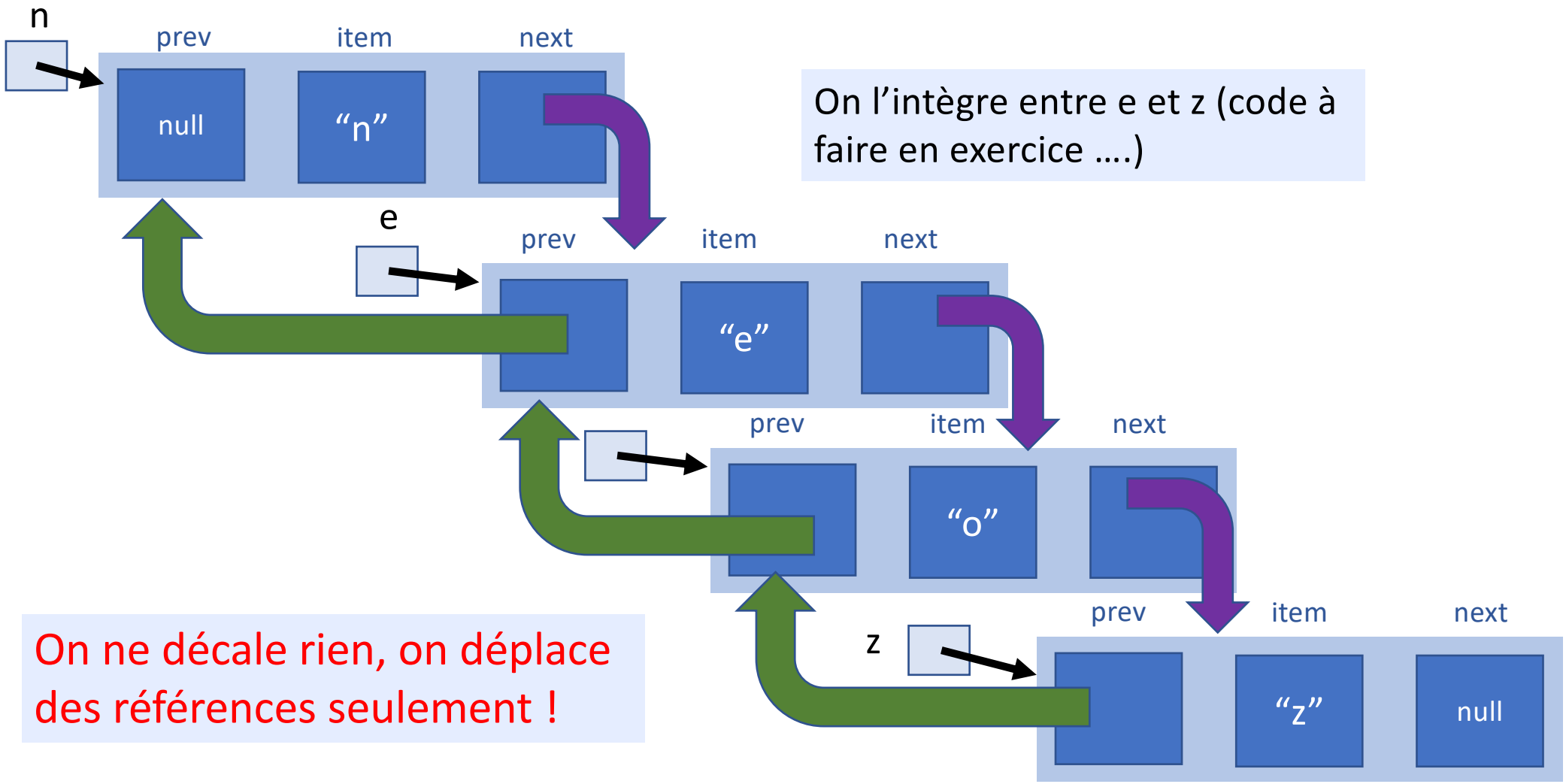
Insérer au milieu : ajouter “o” entre “e” et “z”



Insérer au milieu : ajouter “o” entre “e” et “z”



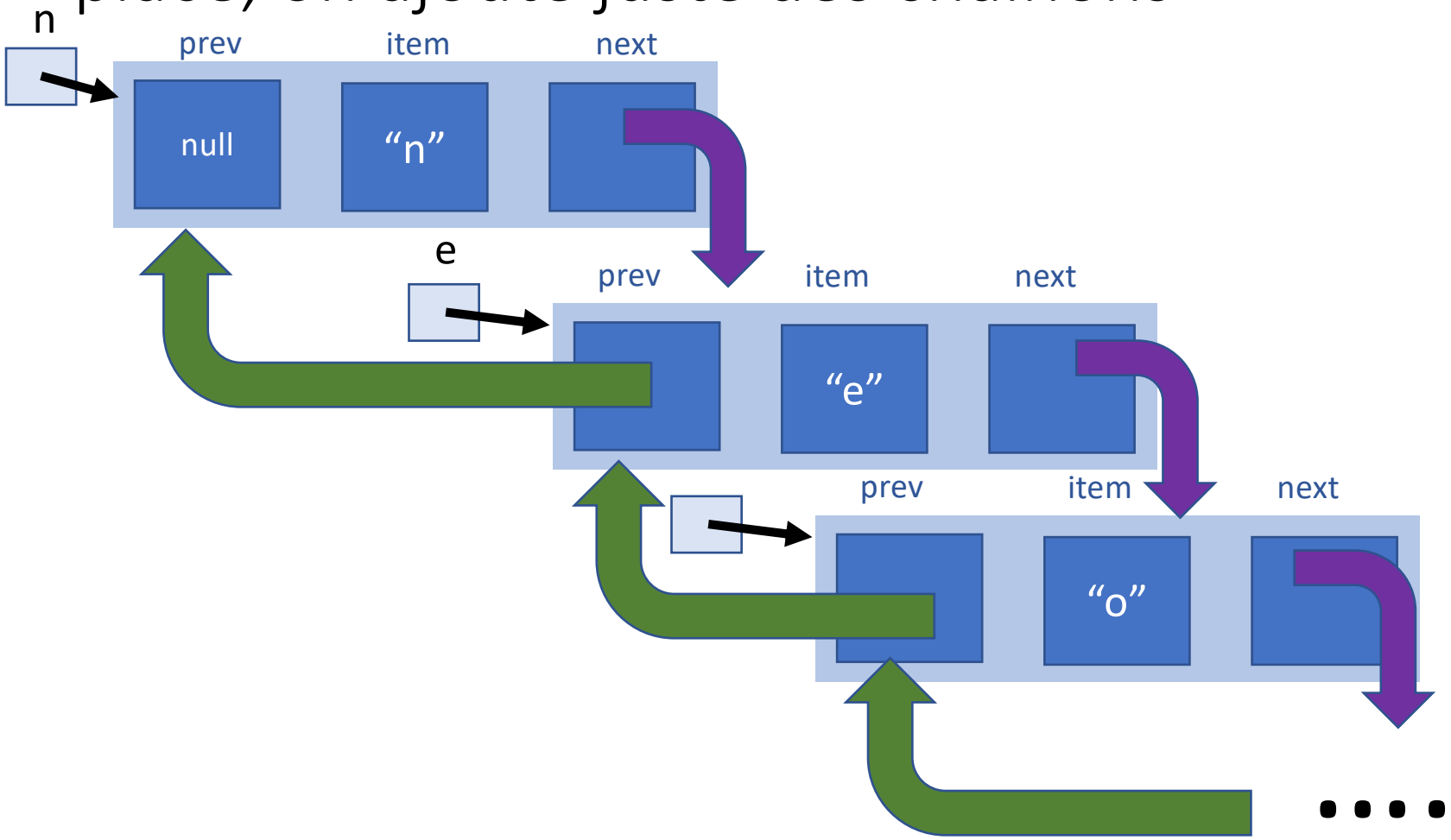
Insérer au milieu : ajouter “o” entre “e” et “z”



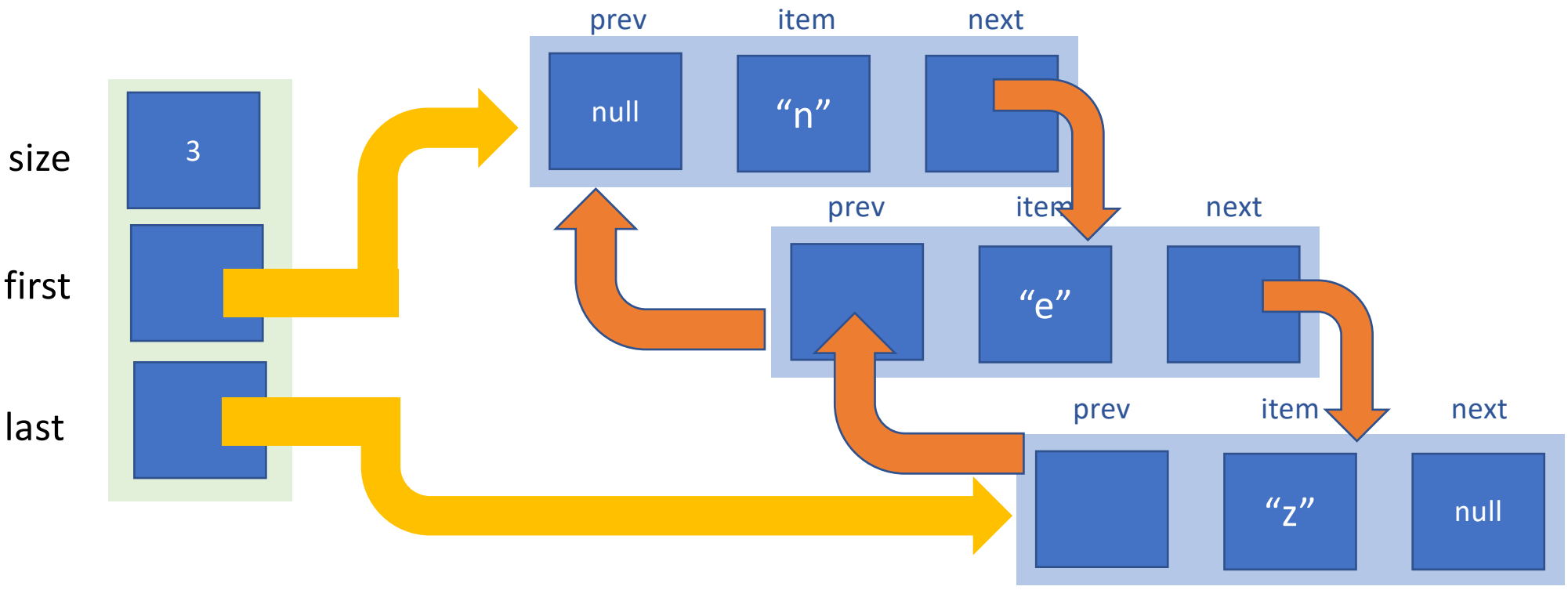
On l’intègre entre e et z (code à faire en exercice ….)

On ne décale rien, on déplace des références seulement !

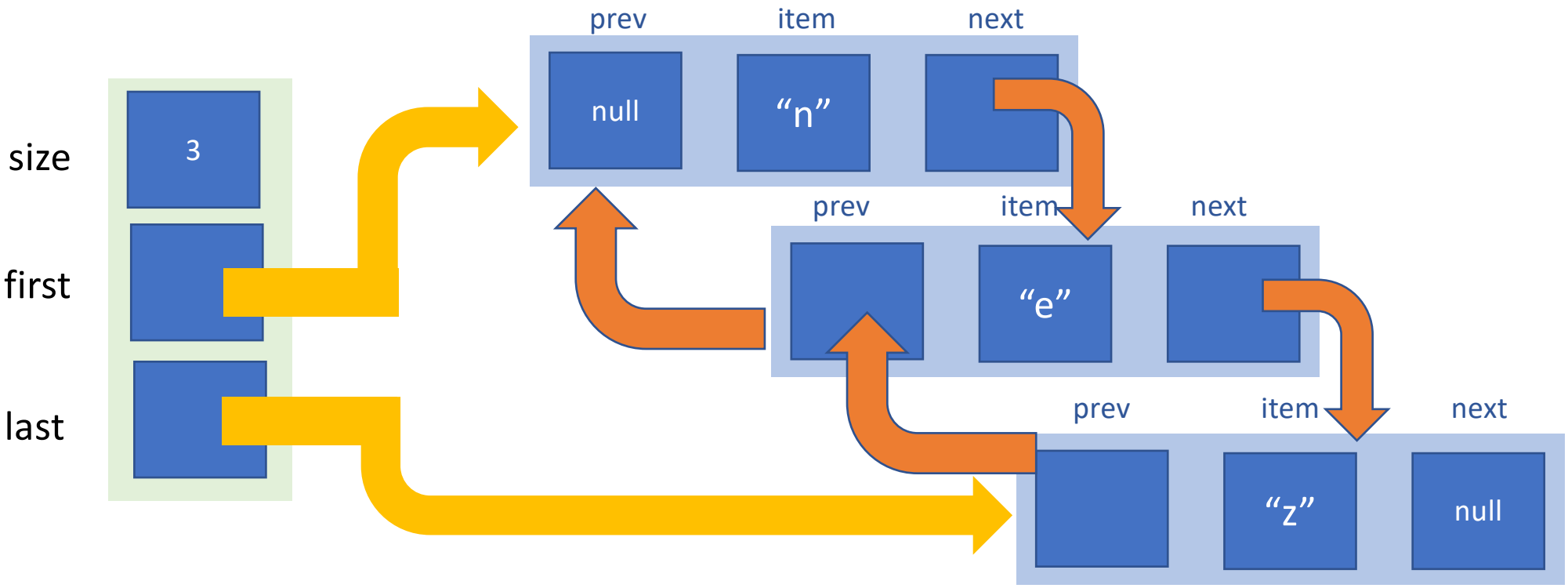
Si on veut compléter à la fin : pas de problème de place, on ajoute juste des chaînons



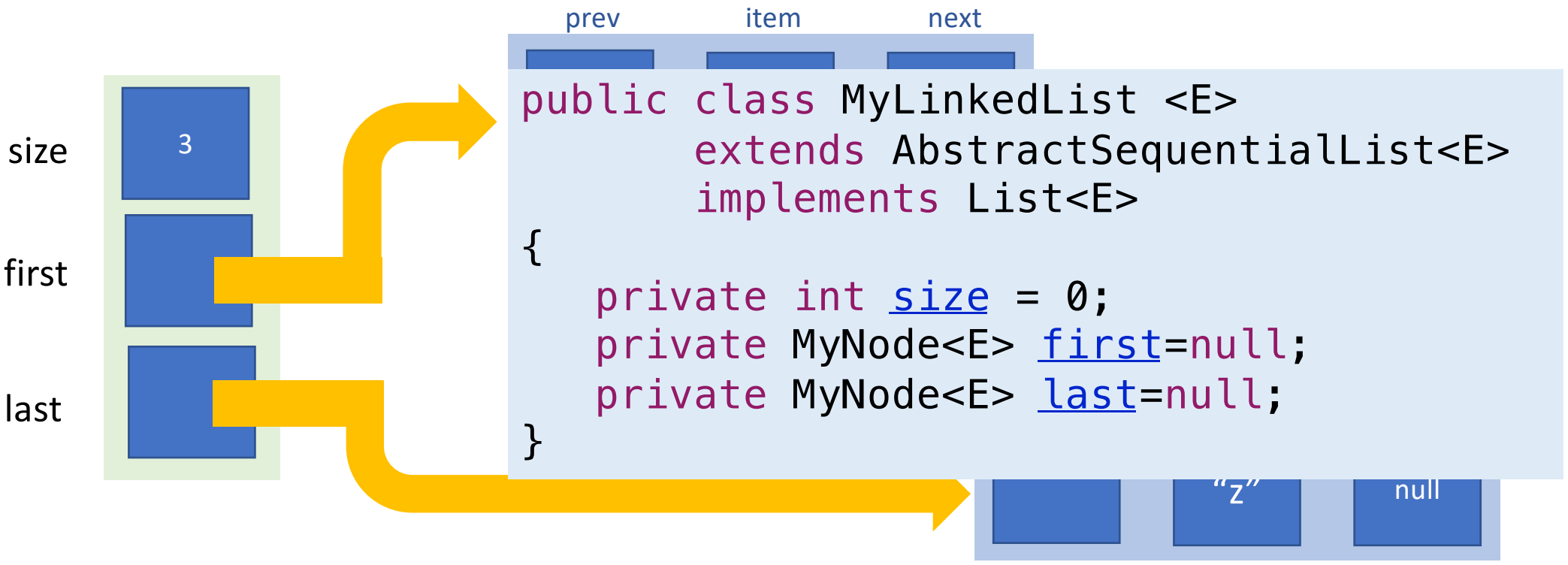
Et maintenant on référence le tout dans un objet dont la classe est MyLinkedList



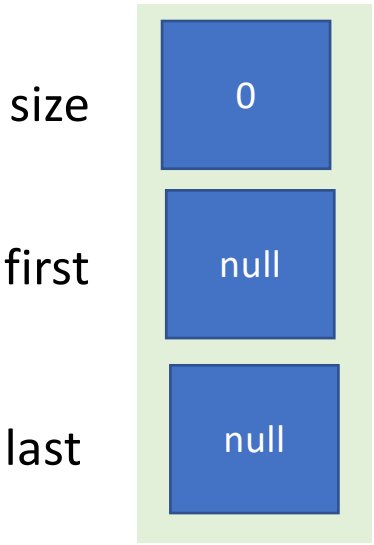
Cette liste aura les opérations de **List** comme les **ArrayList**



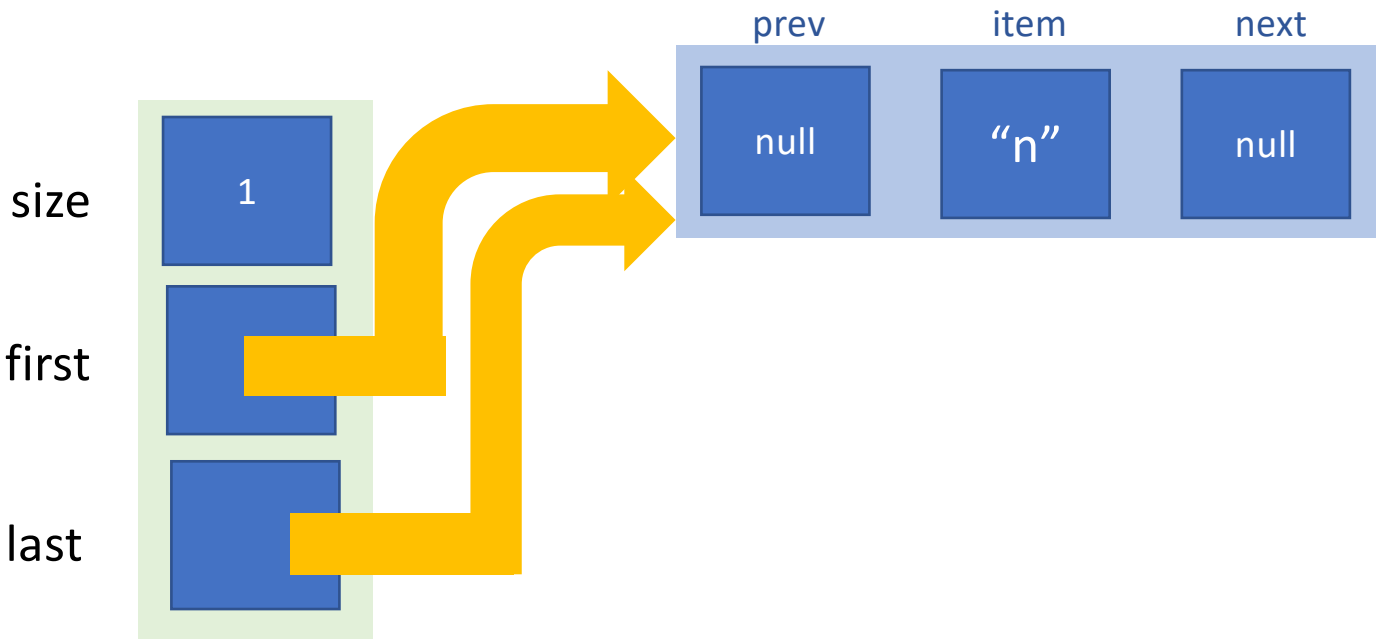
Structure de MyLinkedList



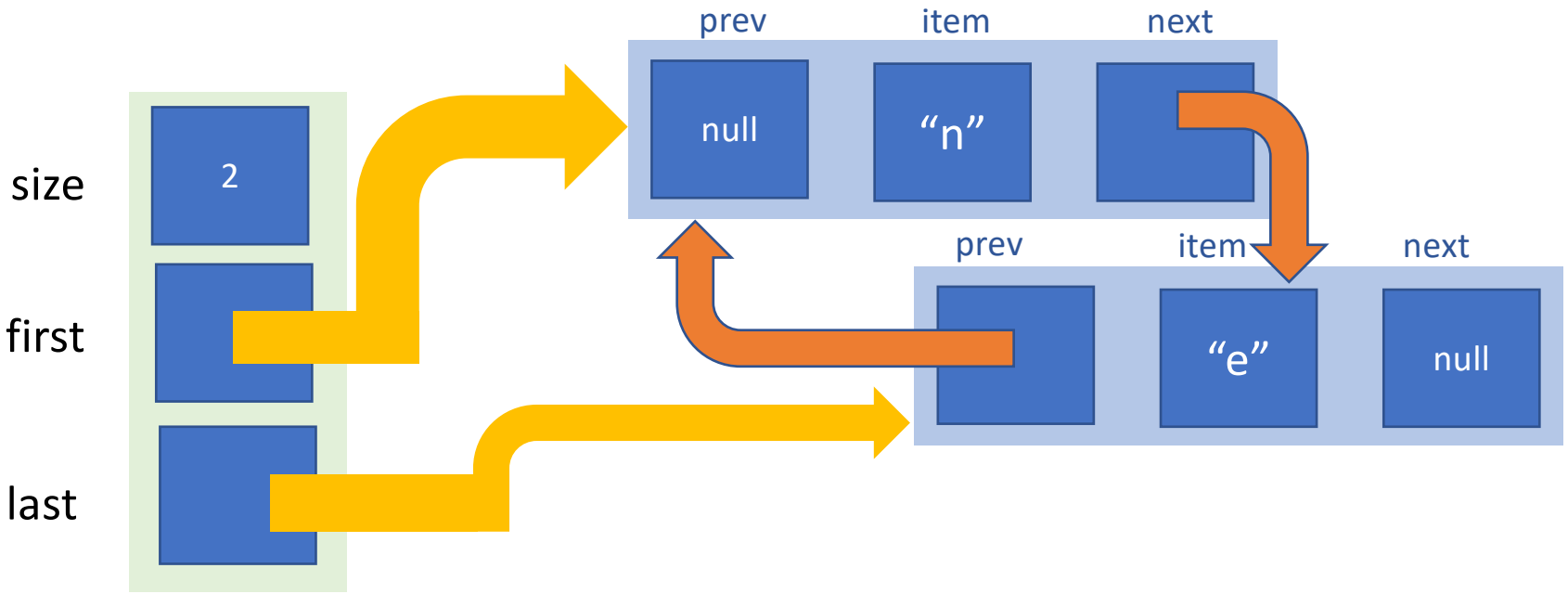
Une MyLinkedList vide



On ajoute un élément dans une liste vide



On ajoute un élément à la fin d'une liste



Le reste à découvrir en programmant quelques fonctions !

Structure chaînée : une structure souple et flexible

- En synthèse :
  - C’est facile de remplir au début, à la fin et au milieu en déplaçant des liens
  - C’est également facilement de supprimer
  - Ce n’est jamais plein

- Mais :
  - La place occupée est plus importante que pour un tableau : c’est celle correspondant aux valeurs + deux emplacements supplémentaires pour les références vers les cellules précédente et suivante
  - L’accès à un élément à partir d’un indice entier est plus long, il faut parcourir la structure à partir du début et avancer autant de fois que l’indice l’indique
  - On doit mémoriser dans une variable à part le nombre d’éléments insérés

Listes chaînées : une structure souple et extensible

- Le procédé que nous avons vu est celui utilisé dans les **LinkedList**

- Les algorithmes sont cachés quand on utilise une **LinkedList**, et les méthodes réalisent les traitements nécessaires :
  - Création des nœuds, déplacement des références

- Et si vous voulez aller plus loin, on peut lire le code source de la LinkedList par exemple ici :

http://www.docjar.com/html/api/java/util/LinkedList.java.html