Inverser une liste

Inverser une liste chaînée

Voici l'algorithme que nous allons expliciter :

```
function reverse(ll) {
  let current = ll.head;
  if (!current || !current.next) {
    return;
  } else {
    let prev = null;
    while (current) {
      const next = current.next;
      current.next = prev;
      prev = current;
      current = next;
    }
    ll.head = prev;
  }
}
```

Copier

Nous commençons par déclaré une variable `current` qui contient l'élément courant.

Au début, nous y plaçons la référence de la tête de la liste.

Nous vérifions que la liste n'est pas vide et qu'elle n'a pas un seul élément : dans ces cas nous n'avons rien à faire et nous pouvons sortir de la fonction.

Si la liste a au moins deux éléments, nous déclarons une variable `prev` qui va contenir l'élément précédent à chaque élément.

Nous l'initialisons à `null` car la tête de liste, qui est pour le moment l'élément courant, n'a pas d'élément précédent.

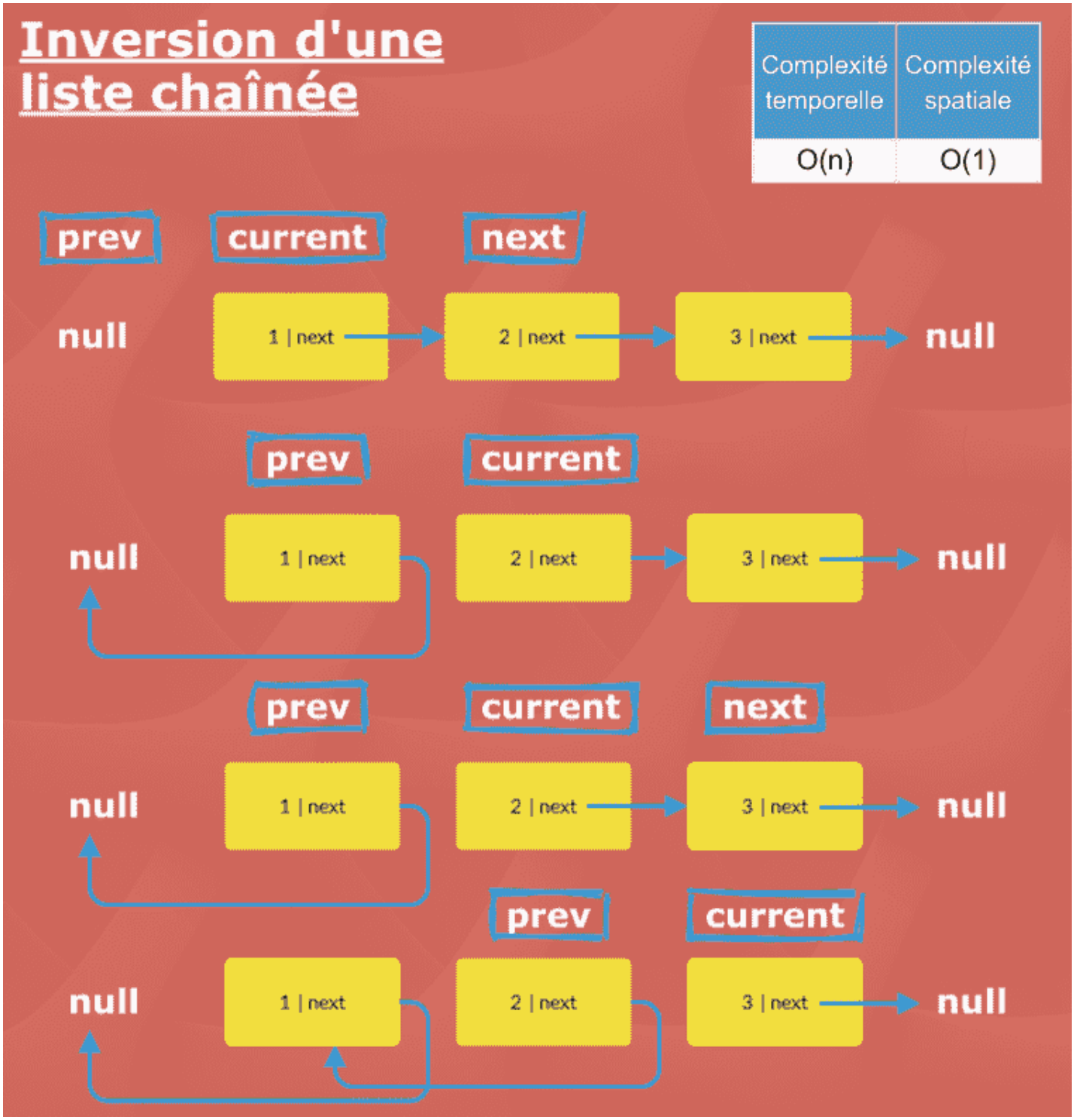
Ensuite, nous itérons sur la liste en utilisant la propriété `next` comme d'habitude. La différence est qu'ici nous avons besoin à chaque fois des références de l'élément précédent, de l'élément courant et de l'élément suivant pour procéder à l'inversion.

Nous avons déclarons donc à chaque itération une variable `next` qui contient d'abord l'élément suivant de l'élément courant (première ligne de notre schéma).

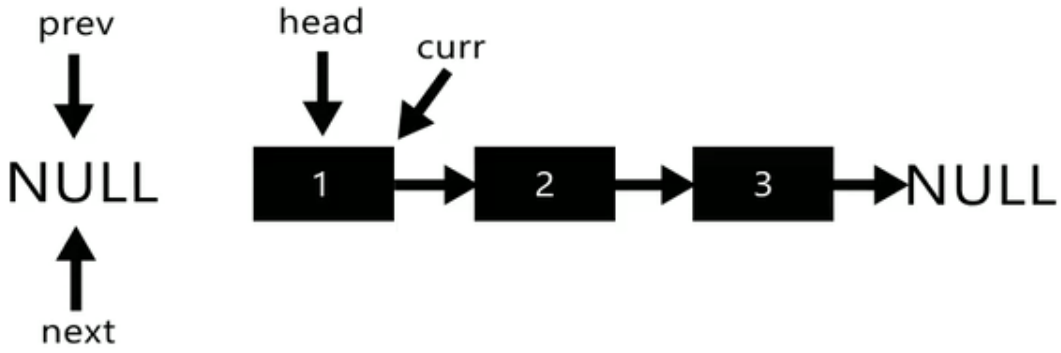
Vient ensuite l'inversion à proprement parlé : **nous réassignons la propriété `next` de l'élément courant en lui donnant la référence de l'élément précédent (deuxième ligne du schéma).**

Ensuite, nous assignons l'élément courant à la variable `prev` et nous assignons l'élément suivant à la variable `current` pour la prochaine itération.

Une fois que nous avons finis totalement l'inversion, nous sortons de la boucle et nous modifions la propriété `head` de la liste chaînée : la nouvelle tête devenant le dernier élément de la liste.



Voici une petite animation (le code est en `C` mais la logique est exactement la même) :




```
while (current != NULL)
     {
          next  = current->next;
          current->next = prev;
          prev = current;
          current = next;
     }
     *head_ref = prev;
```