

Les Listes Chaînées en C++

Limites des tableaux

- ▶ **Taille Fixe :**
La taille d'un tableau est définie à sa déclaration et ne peut pas être modifiée.
- ▶ **Opérations Coûteuses :**
L'insertion et la suppression d'éléments nécessitent de déplacer les éléments suivants, ce qui est inefficace.
- ▶ **Allocation Contiguë :**
Les tableaux requièrent une allocation de mémoire contiguë, problématique en cas de mémoire fragmentée.
- ▶ **Réallocation Coûteuse :**
Agrandir un tableau implique de copier tous les éléments dans un nouveau tableau, ce qui prend du temps et des ressources.

Solution: Structure dynamique : taille variable pendant l'exécution

C'est quoi une liste chaînée

Définition

- une liste linéaire = une suite d'éléments de même type repérés par leur emplacement,
- elle comporte n élément où n peut varier au cours de l'exécution,
- chaque élément est à une place donnée. Les places sont ordonnées et repérées par leur rang,
- la première place est appelé tête de la liste et la dernière "queue" de la liste
- on suppose qu'il y a une fonction successeur (succ) qui donne l'élément suivant pour chaque élément de la liste
- on va distinguer deux types de listes : accès direct sur un élément de rang i ou indirect de façon séquentielle à partir de la tête (la fonction succ)

Types de listes chaînées

- ▶ Liste simplement chaînée
- ▶ Liste doublement chaînée (prev + next)
- ▶ Liste simplement chaînée circulaire
- ▶ Liste doublement chaînée circulaire

Liste simplement chaînée

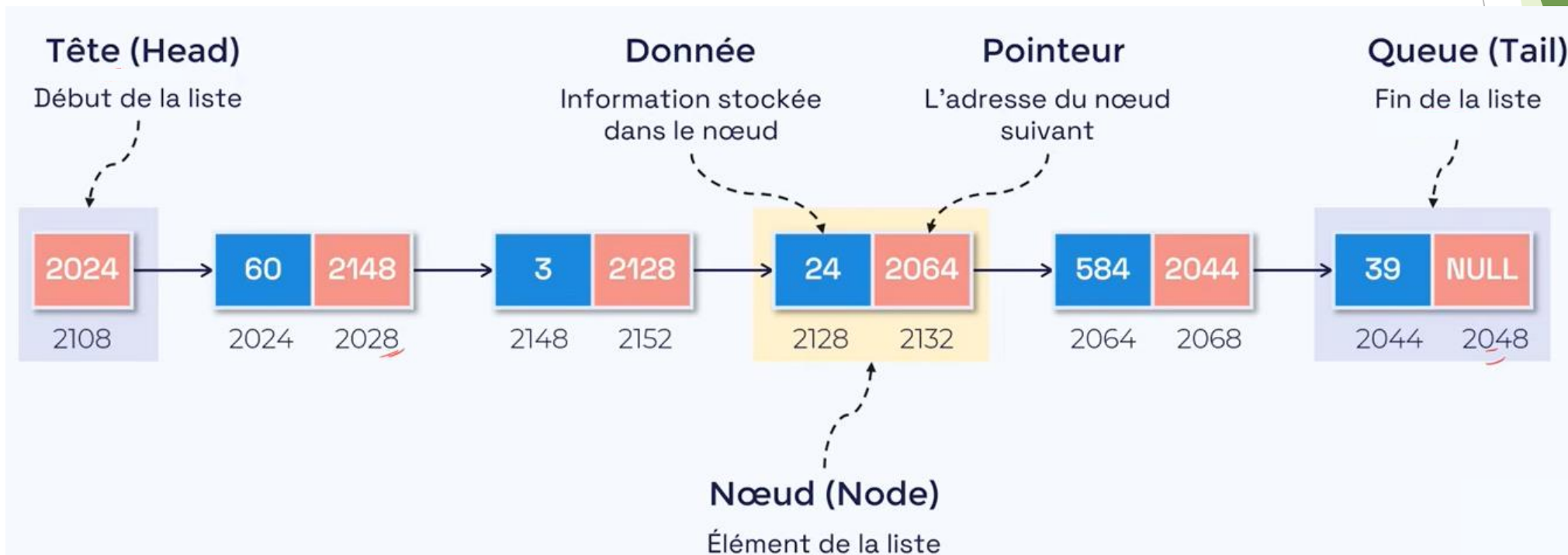
C'est quoi une liste chaînée ?

- ▶ Une **liste chaînée** est comme une chaîne composée de maillons.
Chaque maillon = un **nœud**.
- ▶ Un **nœud** contient :
 - ▶ une **valeur**
 - ▶ un **pointeur** vers le nœud suivant

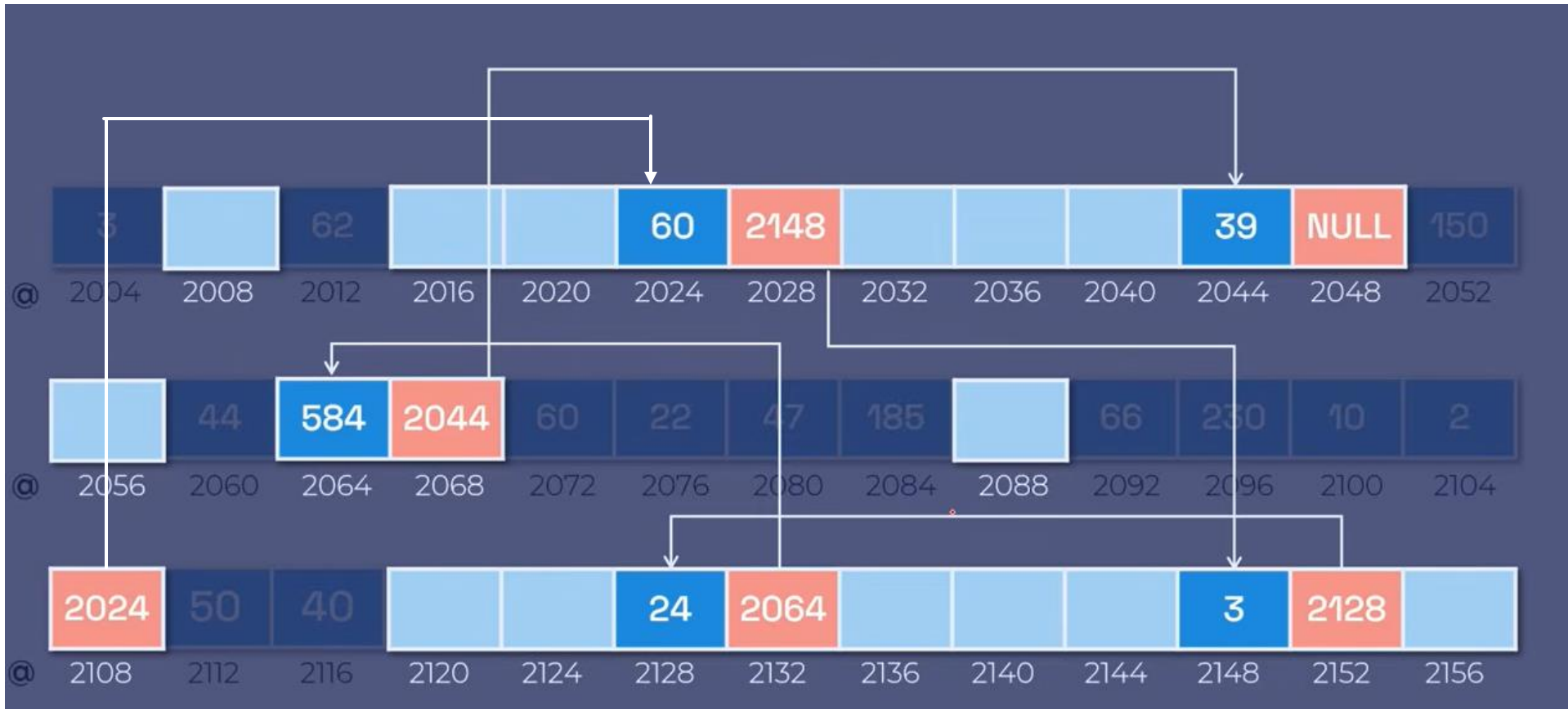
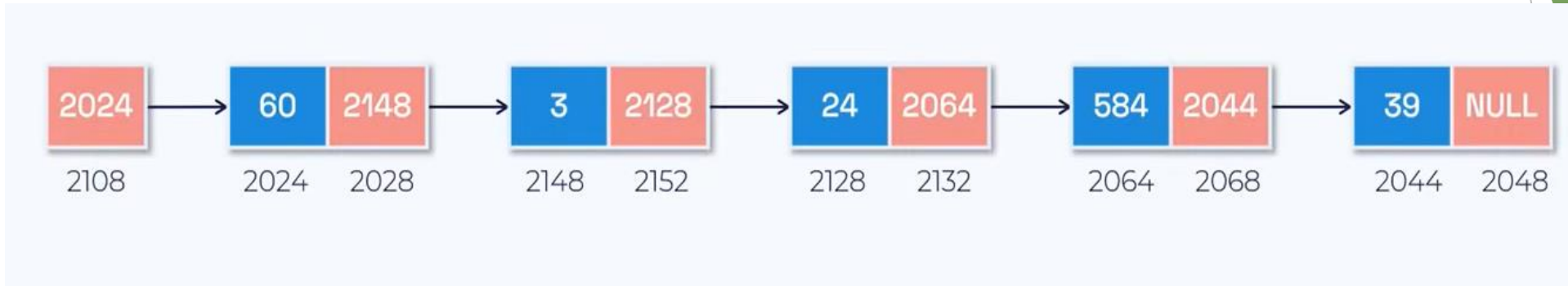
Liste simplement chaînée : Structure d'un nœud en C++

```
struct Node {  
    int data;           // La valeur  
    Node* next;         // L'adresse du nœud suivant  
};
```

La liste chaînée simple (Exemple)



La liste chaînée simple (Exemple)

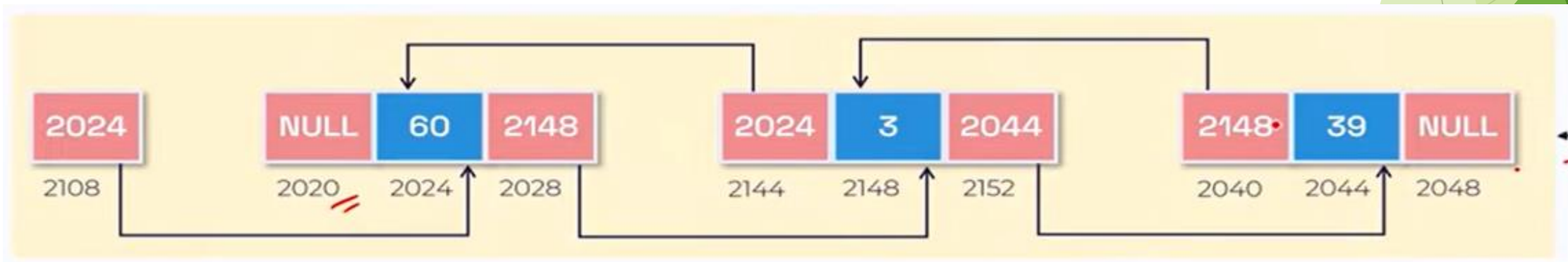


Liste doublement chaînée

Une **liste doublement chaînée** (ou *Doubly Linked List* en anglais) est une structure de données **dynamique** composée de **nœuds**, où chaque nœud contient :

1. Une **valeur ou donnée** (*data*)
2. Un **pointeur vers le nœud suivant** (*next*)
3. Un **pointeur vers le nœud précédent** (*prev*)

Contrairement à une liste chaînée simple, chaque nœud sait **qui est avant lui et après lui**, ce qui permet de parcourir la liste **dans les deux sens** (avant → arrière ou arrière → avant).



Liste doublement chaînée: Structure d'un nœud en C++

```
struct Noeud {  
    int valeur;           // La valeur stockée dans le noeud  
    Noeud* suivant;       // Pointeur vers le noeud suivant  
    Noeud* precedent;     // Pointeur vers le noeud précédent  
};
```

Liste simplement chaînée circulaire

- ▶ **Définition d'une Liste Simplement Chaînée Circulaire (LSCC)**
- ▶ Une **liste simplement chaînée circulaire** est une variante de la **liste chaînée simple** où :
 - ▶ Chaque élément (noeud) contient une **valeur** et un **pointeur vers le suivant**.
 - ▶ Le **dernier noeud ne pointe pas vers NULL**, mais **retourne vers le premier noeud** (la tête).
 - ▶ Cela forme une **boucle circulaire**

Liste simplement chaînée circulaire

Différence avec une liste simplement chaînée normale

Caractéristique	Liste Chaînée Simple	Liste Chaînée Circulaire
Pointeur du dernier noeud	<code>nullptr</code>	pointe vers le premier
Début et fin	Commence à <code>head</code> , finit à <code>nullptr</code>	Commence à <code>head</code> , jamais vraiment de fin (boucle infinie possible)
Utilité	Parcours simple	Parcours circulaire, applications comme file circulaire, jeux, etc.

Liste simplement chaînée circulaire



Liste doublement chaînée circulaire

- ▶ C'est une liste dont chaque élément possède trois choses :
 - ▶ Une valeur
 - ▶ Un pointeur vers l'élément suivant
 - ▶ Un pointeur vers l'élément précédent
- ▶ Comme une liste doublement chaînée normale, MAIS...
 - 👉 Le dernier élément **ne pointe pas vers NULL**.
 - 👉 Il pointe vers le **premier**.
 - 👉 Et le premier pointe vers le **dernier**.
- ▶ Donc :
 - ✓ il n'y a **pas de début ou de fin absolue**
 - ✓ on peut tourner dans la liste à l'infini
 - ✓ on peut aller dans les deux sens

Liste doublement chaînée circulaire

```
struct Noeud {  
    int valeur;  
    Noeud* suivant; // Pointeur vers le noeud suivant  
    Noeud* precedent; // Pointeur vers le noeud précédent  
};
```

Circularité :

- dernier->suivant = premier
- premier->precedent = dernier.