LISTES CHAINÉES

T.D.A : Définition

Un *type de données abstrait* est composé d'un ensemble d'objets, similaires dans la forme et dans le comportement, et d'un ensemble d'opérations sur ces objets.

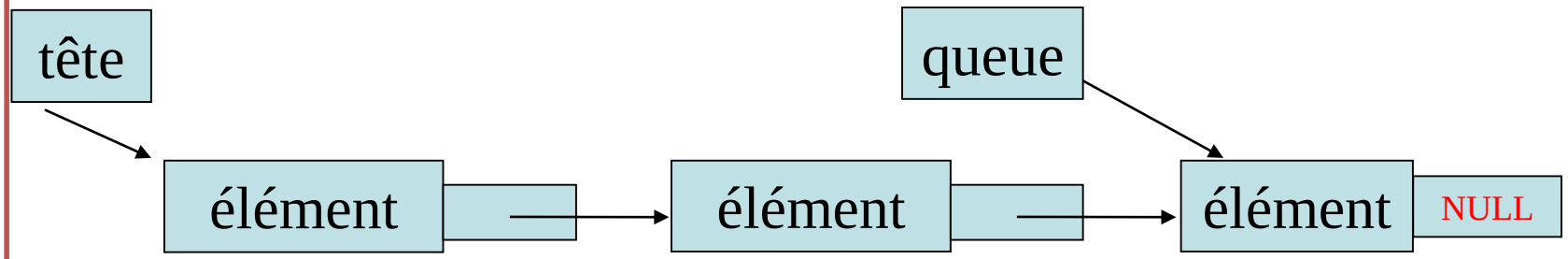
L'implémentation d'un T.D.A. ne suit pas de schéma préétabli. Il dépend des objets manipulés et des opérations disponibles pour leur manipulation.

T.D.A : Avantages

- prise en compte de types complexes.
- séparation des services et du codage.
  - L'utilisateur d'un TDA n'a pas besoin de connaître les détails du codage.
- écriture de programmes modulaires.

Les listes chaînées

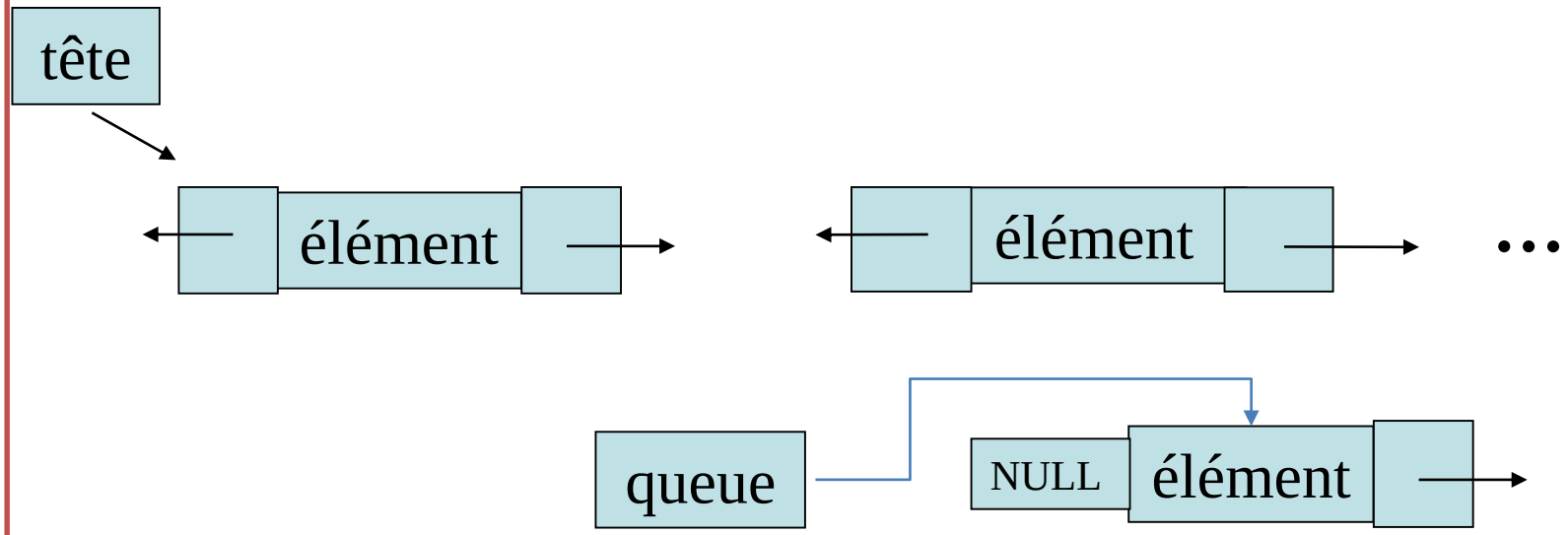
Définition : Une liste chaînée est composée d'une suite finie de cellules (ou couples) formées d'un élément et de l'adresse (ou référence) vers l'élément suivant.



- Le 1er élément est sa tête
- Le dernier est sa queue
- Le pointeur dernier élément a une valeur sentinelle (NULL)

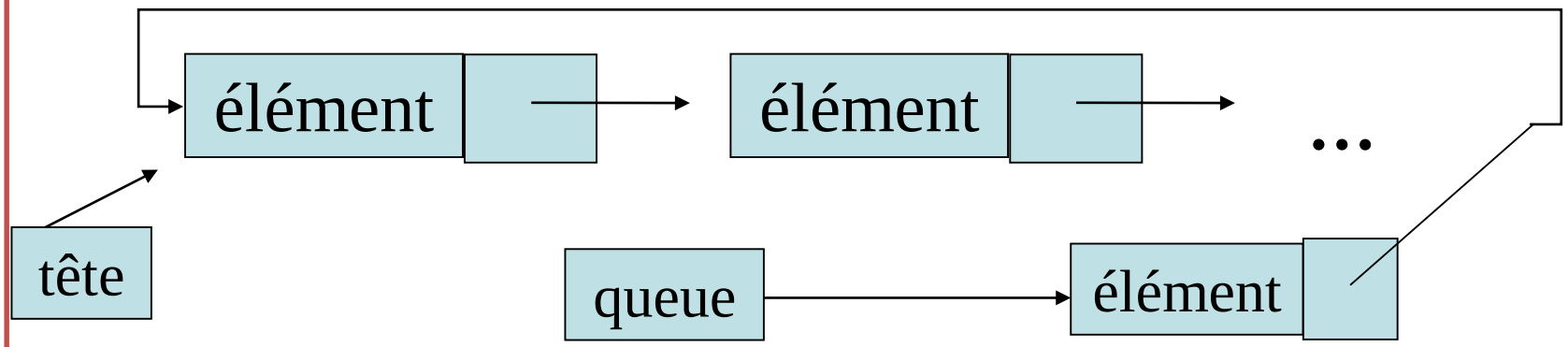
Les listes doublement chaînées

Les cellules d'une liste doublement chaînée admettent aussi un pointeur vers le précédent

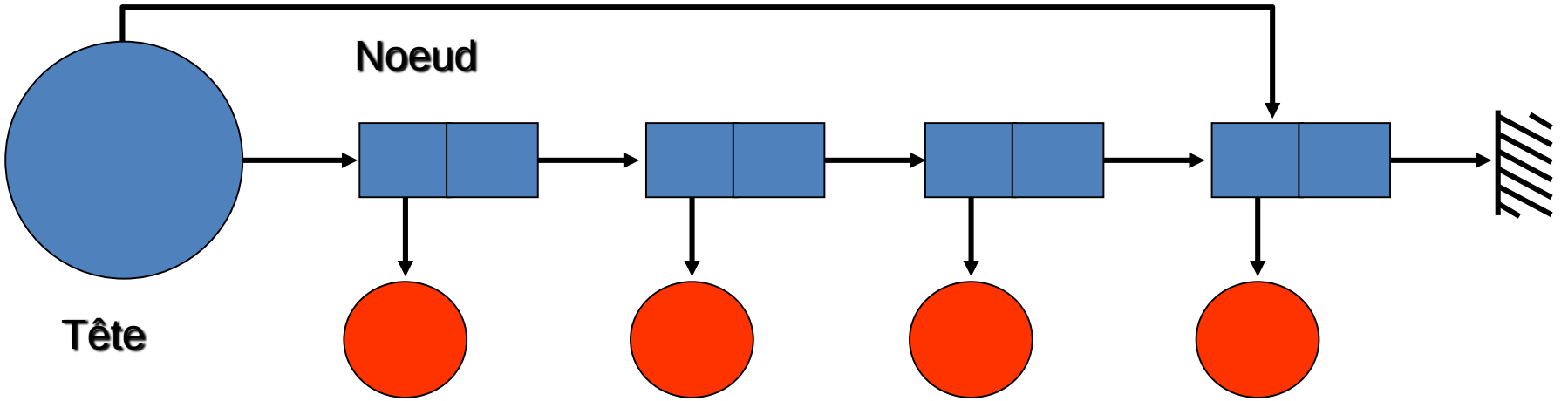


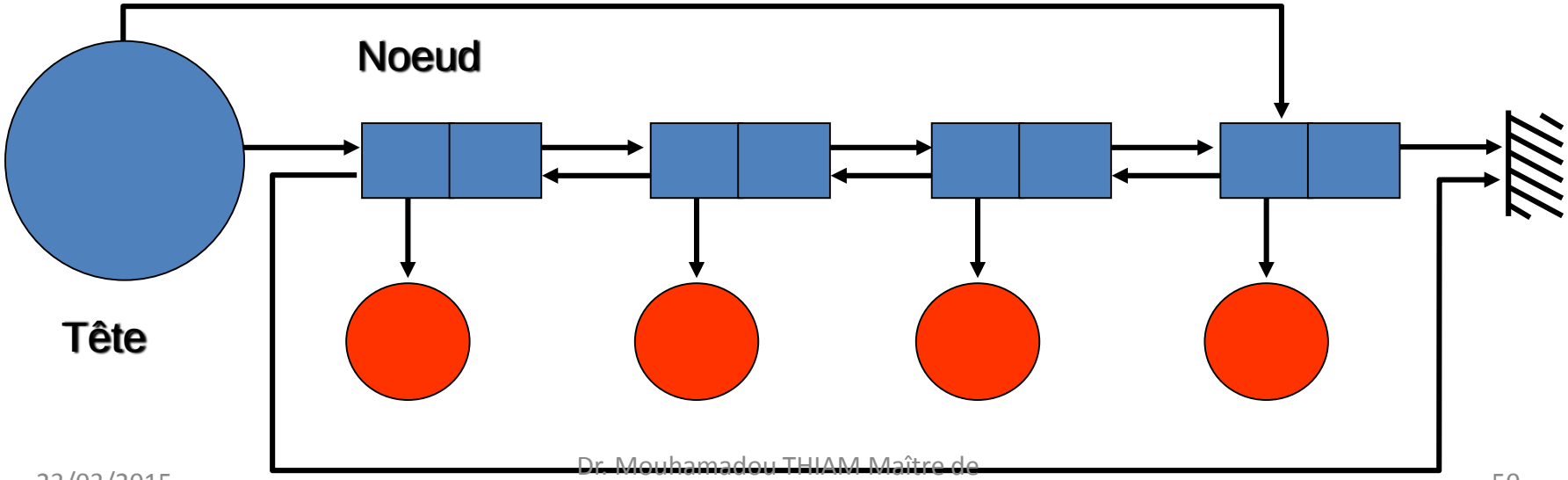
Les listes chaînées circulaires

Définition : Une liste chaînée circulaire admet la même structure qu'une liste classique mais le champs « suivant » de la dernière cellule contient l'adresse de la première cellule.

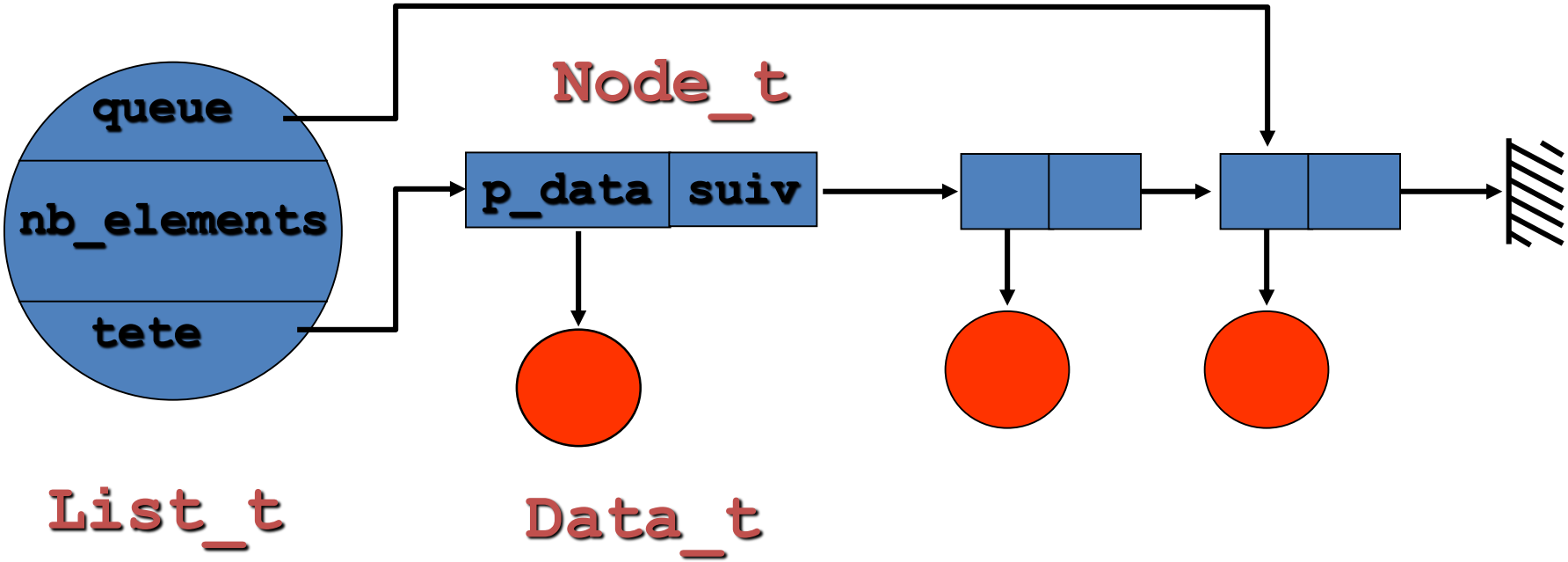


Structures





Structures



Spécifications

- Créer une liste vide
- Créer une liste avec le 1er élément
- Ajouter un élément (début / fin / milieu)
- Concaténer 2 listes
- Retirer un élément (début / fin / milieu)
- Supprimer un élément particulier
- Détruire une liste
- Trier les éléments d'une liste
- Permuter 2 éléments

Liste chaînée : Header

```
typedef struct List_t {
    struct Node_t* tete;
    struct Node_t* queue;
    int nb_elements_;
} List_t;
```

```
typedef struct Node_t {
    struct Data_t* p_data_;
    struct Node_t* suiv;
} Node_t;
```

```
typedef struct Data_t {
    ...
} Data_t;
```

Liste chaînée : Header

```
• List_t* list_create( void );

• int list_insert_item(
    List_t* list, Data_t* item
  );

• int list_append_item(
     List_t* list, Data_t* item
  );

• int list_insert_item_before(
    List_t* list,
    Data_t* to_insert,
     Data* list_item
  );
```

Liste chaînée : Header

- `int list_destroy( List_t* list );`
- `int list_empty( List_t* list );`
- `Data_t* list_remove_head( List_t* list );`
- `Data_t* list_remove_tail( List_t* list );`
- ```
  int list_remove_item(
    List_t* list
    Data_t* item
  );
  ```
- `int list_sort( List_t* list );`
  ```

Liste chaînée : Utilisation

Avant d'aller plus loin, vérifions si nos spécifications sont suffisantes...

Pour cela, nous allons écrire un programme qui utilise les fonctions du fichier **list.h**, sans nous préoccuper de la façon dont elles sont implantées.

But du programme: construire et trier une liste d'entiers par ordre croissant.

Liste simplement chaînée

- Définir le T.D.A d'un nœud

```
struct nombre{
    int val;
    struct nombre * suiv;
}* Node_t, NBR;
```

Création de la liste

- Définir le T.D.A de la liste

```
typedef struct List_t {
        Node_t tete;
        Node_t queue;
        int nb_elements;
} List_t, *L;
```

Création d'un nœud

```
Node_t createNode (int n){
    Node_t s = (Node_t)malloc(sizeof(NBR));
    s→val = n;
    s→suiv = NULL;
    return s;
}
```

Création de la liste vide

```
void createListe (L * suite){
        *suite→tete = NULL;
        *suite→queue = NULL;
        *suite→nb_elements = 0;
}
```

Création de la liste

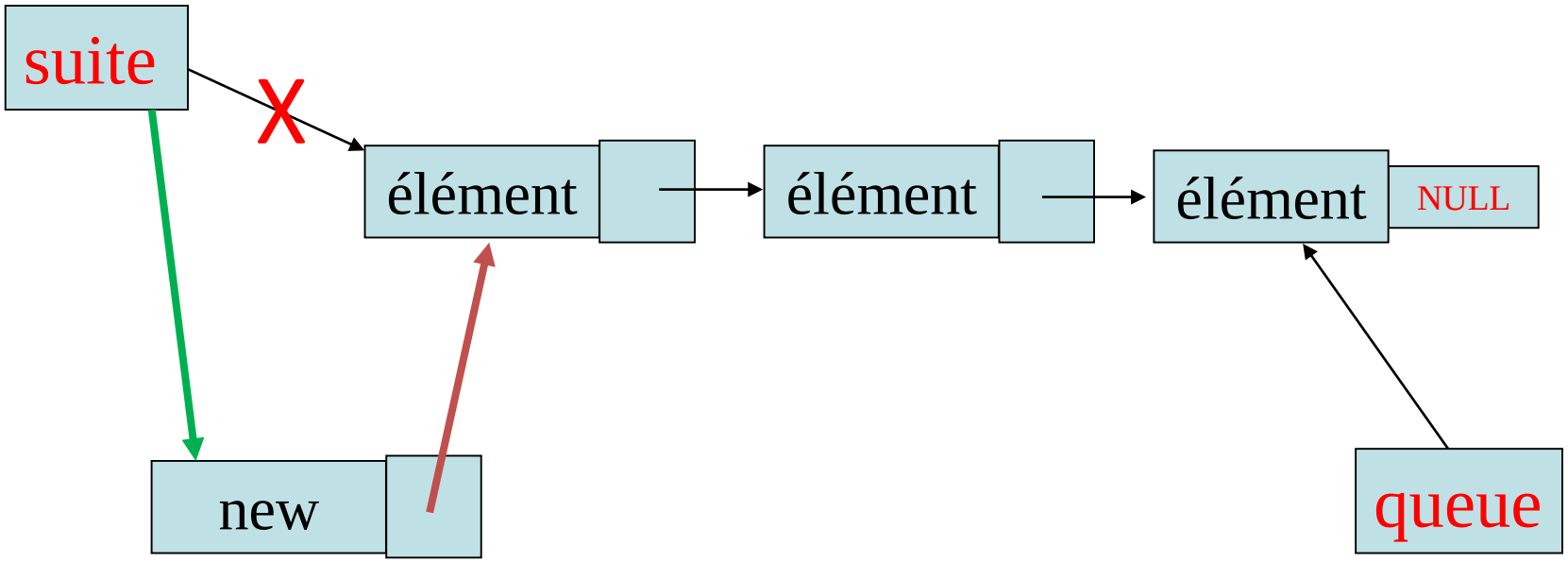
```
Node_t createNode (int n){
Node_t s = (Node_t)malloc(sizeof(NBR));
        s→val = n;
        s→suiv = NULL;
        return s;
      }
```

```
void create (int n, L * suite){
      *suite→tete = createNode(n);
      *suite→queue = *suite→tete ;
      *suite→nb_elements = 1;
}
```

Affichage de la liste

```
void show (L suite){
      Node_t p = suite→tete;
     while (p!=NULL){
         printf("%d ", p→val);
         p = p→suiv;
    }
}
```

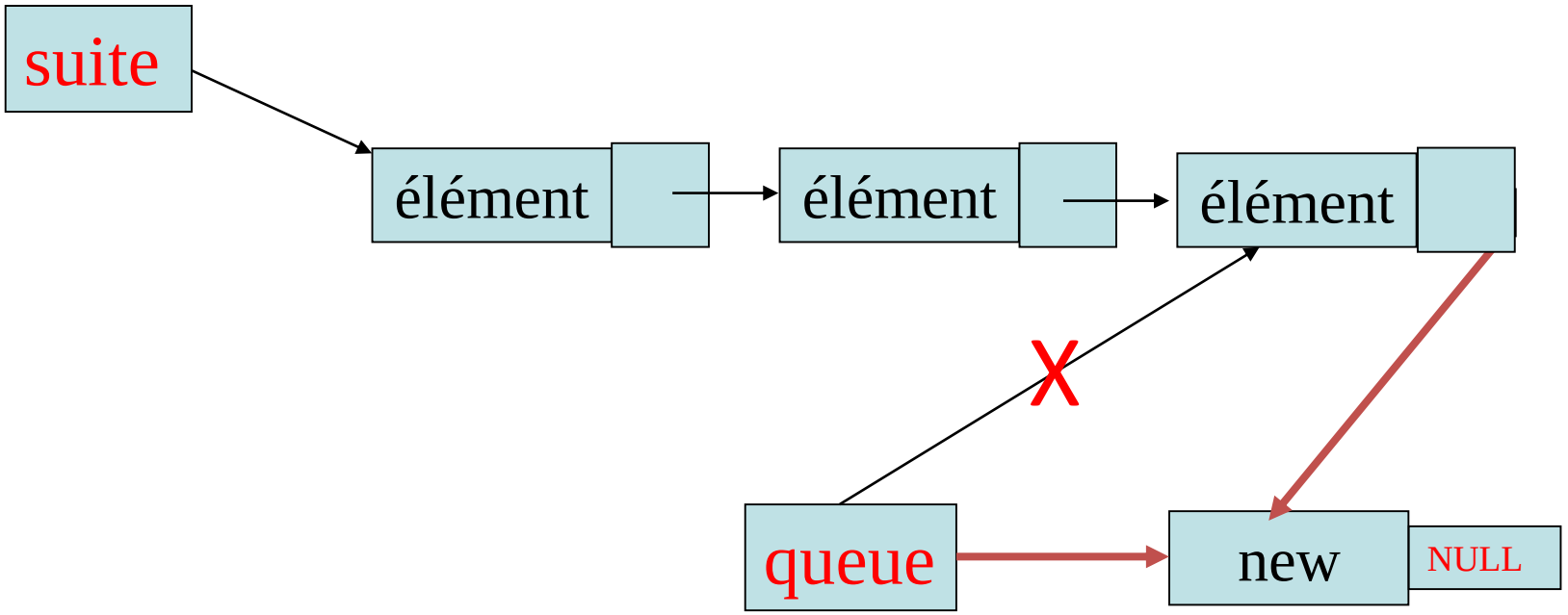
Ajouter en tête de liste



Ajouter en tête de liste

```
void ajoutDeb (L * suite, int n){
    Node_t s = createNode (n);
    s→suiv = *suite→tete;
    *suite→tete = s;
    *suite→nb_elements += 1;
}
```

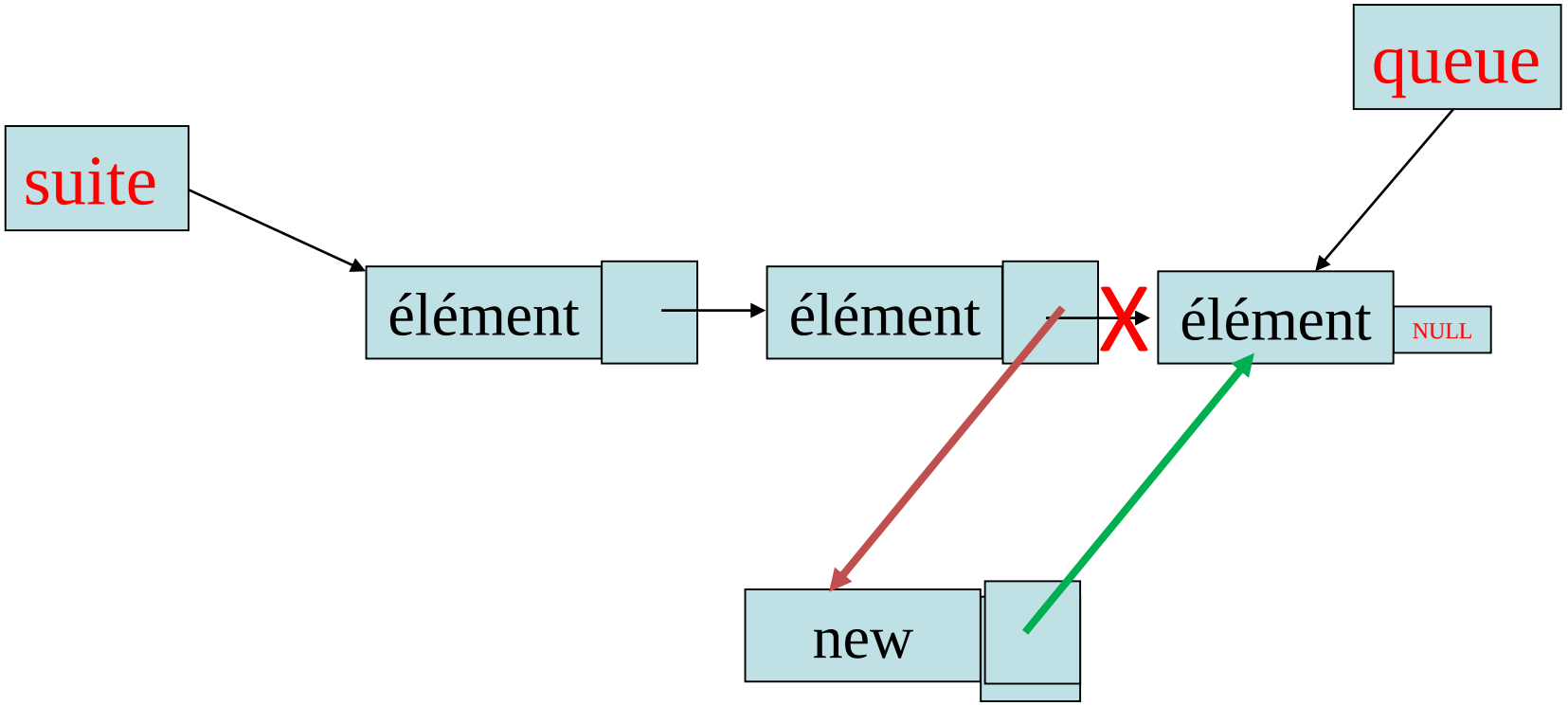
Ajouter en queue de liste



Ajouter en queue de liste

```
void ajoutFin (L* suite, int n){
        Node_t   s = createNode (n),          //nouvel élément
                 p=*suite→tete;               //pointeur de parcours
      if (p == NULL){
                 *suite→tete=*suite→queue = s;
                 nb_elements = 1;
      }
      else{
                 while (p→suiv != NULL)
                          p = p→suiv;
                 p→suiv = s;
                 *suite→queue = s;
                 nb_elements +=1;
      }
}
```

Ajouter dans la liste



Ajouter dans la liste

```
void ajoutDans (Node_t * suite, int n, int m){
 Node_t   s = createNode (n), //nouvel élément
                p=*suite→tete;       //pointeur de parcours
   while (p != NULL && p→val != m)
                p = p→suiv;
   s → suiv = p→suiv
   p→suiv = s;
   if (s→suiv == NULL) *suite→queue = s;
}
```