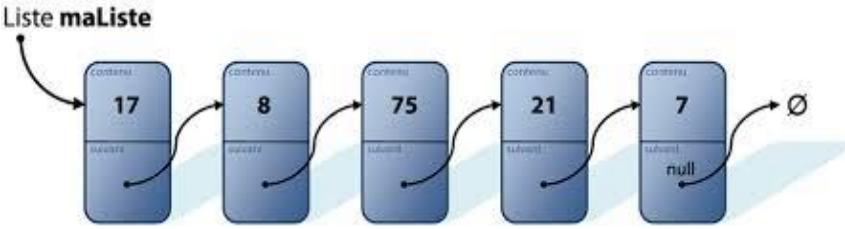
PART 4 : LES LISTES CHAÎNÉES



1. Tableaux de structures
2. Pointeurs dans une structure
3. Listes chaînées - Définition
4. Listes chaînées - Nœuds
5. Listes chaînées - Tête
6. Listes chaînées - Déclaration
7. Listes chaînées - Créer un noeud
8. Listes chaînées - Ajouter un nœud
9. Traverser - Chercher dans une LC
10. Insertion d'un nœud dans une LC

Tableaux de structures

- Nous avons créé des structures de type définit à l'aide de la syntaxe : *typedef struct {…} NOM_DU_TYPE;*

- Nous pouvons avoir des tableaux de structures:

  NOM_DU_TYPE nomDuTableau[X];

- Par exemple:

```
typedef struct
  {
     char prenom[15];
     char surnom[25];
     unsigned long numeroDeCollege;
     float moyenne;
  } ETUDIANT; //Nom du type
ETUDIANT etudiants[20]; //tableau d'étudiants
```

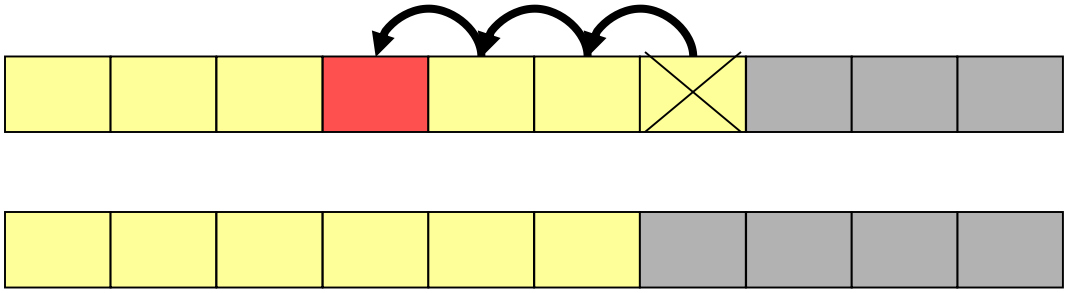
Tableaux de structures

- Les tableaux de structures sont des structures qui sont puissantes pour stocker des quantités **connues** de données complexes qui ne *changent pas souvent mais consultées souvent*

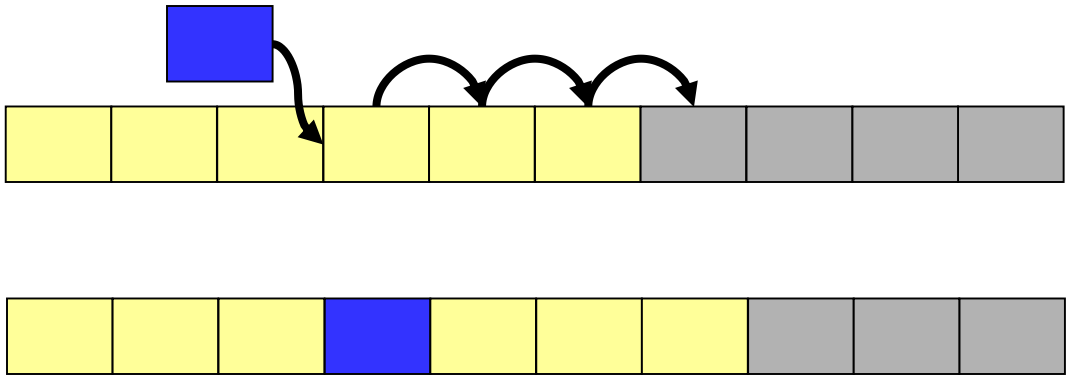
- Cependant les tableaux de structures ont des limitations
  - Vous devez savoir la quantité des éléments pour votre tableau avant le temps de compilation **=> pas flexible**
  - Ou utiliser des tableaux qui sont larges pour contenir un maximum d'éléments **=> perte d'espace excessive**
  - Si vous avez à effacer des éléments, vous devez décider si vous laissez un trou ou si vous compactez le tableau
  - Si vous voulez insérer un élément dans un tableau dans un ordre quelconque vous devez déplacer des éléments **=> pas vraiment efficace pour les tableaux qui sont larges**.

Tableaux de structures

- Effacer un élément dans un tableau:



- Insérer un élément dans un tableau:



Pointeurs dans une structure

- Pour résoudre le manque d'efficacité de l'utilisation des tableaux pour entreposer des structures, nous faisons appel aux pointeurs

- Chaque structure que nous définissons peut contenir des champs qui sont de différents types; ceux ci inclus des variables de types pointeurs

Pointeurs dans une structure

- Par exemple, si on veut stocker le nom d'un partenaire de labo dans la structure ETUDIANT, on peut ajouter deux nouveaux champs:

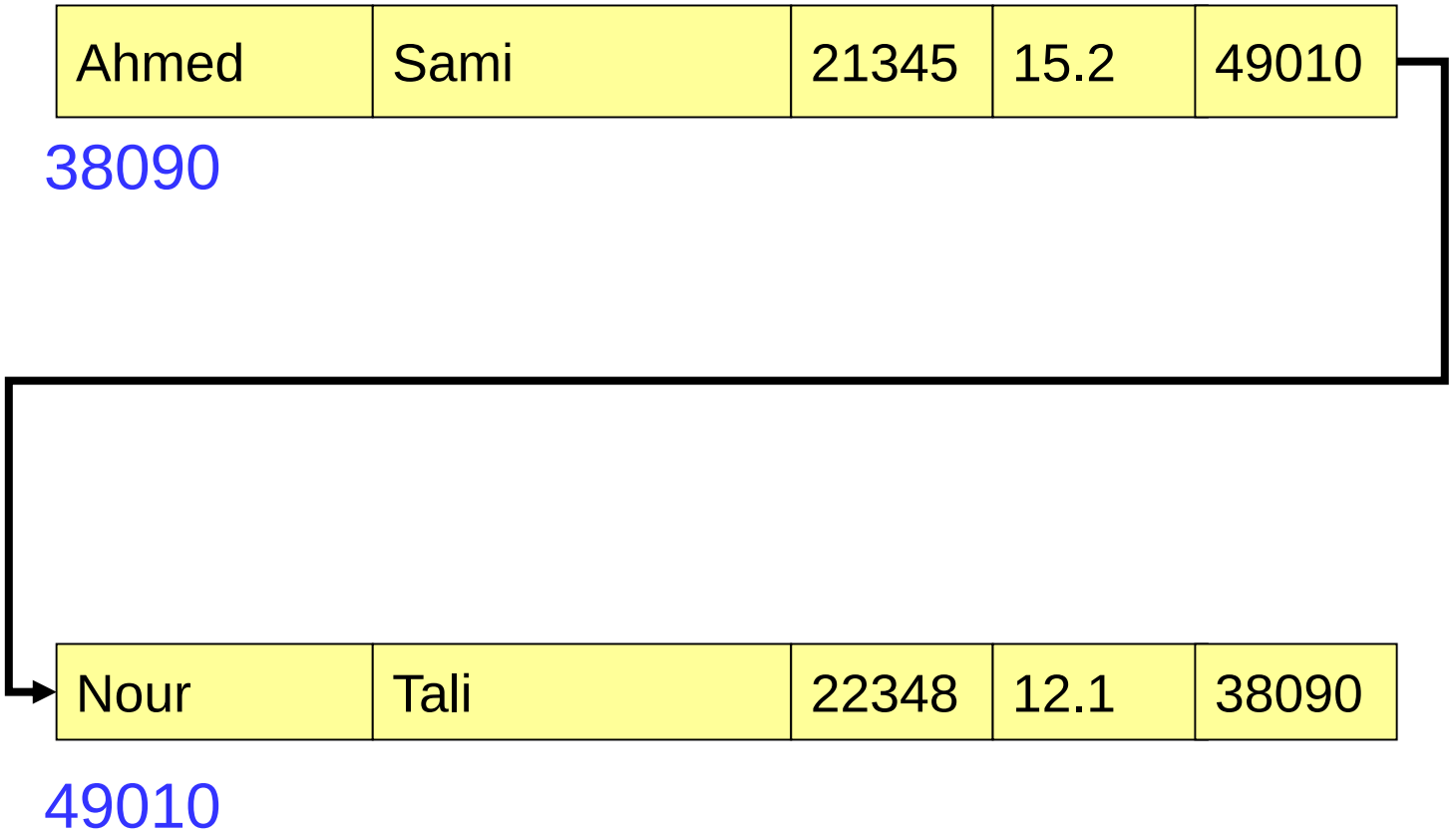
```
typedef struct
  {
     char prenom[15];
     char surnom[25];
     unsigned long numeroDeCollege;
     float moyenne;
     char prenomLabPart[15];
     char surnomLabPart[25];
  } ETUDIANT; //nom du type
```

Pointeurs dans une structure

- Le dernier exemple requiert 40 octets de plus pour chaque structure.
- Une façon plus efficace d'avoir la même information serait de prendre un pointeur qui pointe au partenaire de labo. (4 octets au lieu de 40!):

```
typedef struct TAG_ETUDIANT
  {
     char prenom[15];
     char surnom[25];
     unsigned long numeroDeCollege;
     float moyenne;
     struct TAG_ETUDIANT* partDeLabo;
  } ETUDIANT; //nom du type
```

Pointeurs dans une structure



Listes chaînées - Définition

- Nous pouvons donc pointer sur une autre variable de type structure …et ***lier les deux structures ensemble.***
- Nous pouvons alors organiser nos données avec un autre genre de structure de données
- Au lieu de se servir des tableaux de structures, nous pourrions utiliser des pointeurs pour ***lier*** toutes les autres structures de *même type* ensemble
- Cette nouvelle structure est une ***liste chaînée***
- Les éléments dans une liste chaînée s'appellent des ***noeuds***

Les listes chaînées - Définition

- Permet de stocker un ensemble de données de même type.

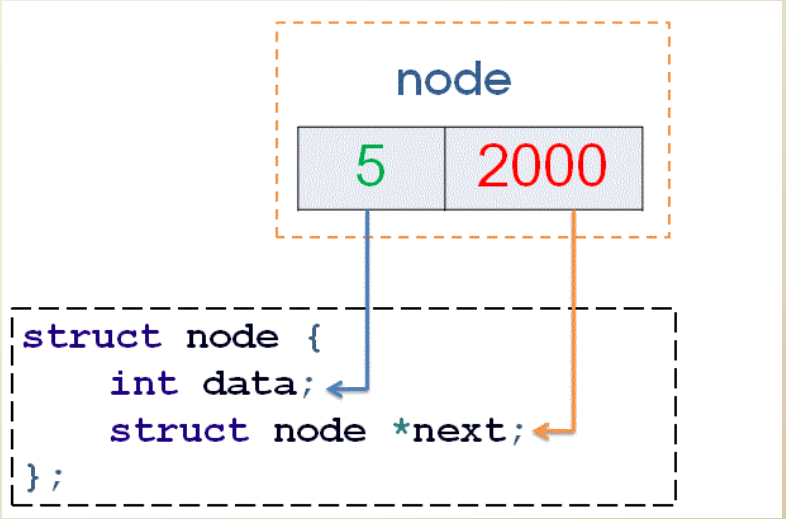
- Avantage sur les tableaux:
  - facilitation d'étendre la taille
  - insertion sans avoir à décaler les éléments un par un

- Les listes chaînées utilisent des structures dont **l'un des membres est un pointeur sur une structures de même type** servant ainsi de chaines entres les éléments de la liste

Listes chaînées - Nœuds

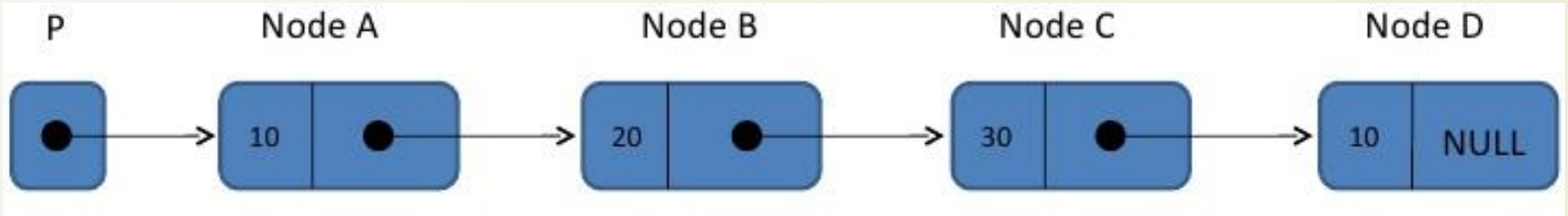
- Chaque nœud dans une liste chaînée contient deux composantes majeurs:
  - Les ***données*** => information de l'étudiant
  - Le ***lien*** => un pointeur au prochain nœud dans la liste

- Une liste chaînée est donc une chaîne de structures ordonnées et du même type

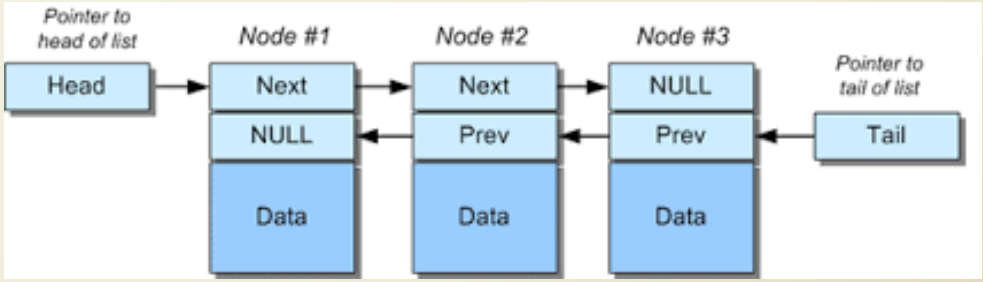


Les listes chaînées - Types

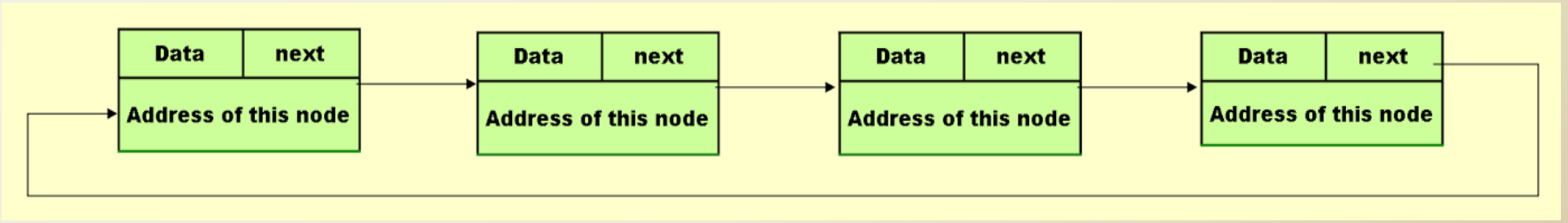
- **Listes simplement chainées:**



- **Listes doublement chainées:**



- **Listes circulaires:**



Listes chaînées - Tête

- Les *listes chaînées sont des structures dynamiques* qui grandissent et rapetissent à nos besoins.

- Nous pouvons ainsi utiliser les fonctions **`malloc`** et **`free`** pour grandir et rapetisser nos chaînes de structures.

- Comme nous l'avons vue dans le cours sur l'allocation de la mémoire dynamique, les blocs de mémoire n'ont pas de nom symbolique (comme les variables) qui leurs sont attachés.

- Nous devons donc déclarer un pointeur qui "tiens" la ***tête*** (début) de la liste chaînée.

Listes chaînées - Déclaration

- On peut déclarer un nœud de structure pour ETUDIANT de cette façon:

```
typedef struct TAG_NOEUD_ETUDIANT
  {
    char prenom[15];
    char surnom[25];
    unsigned long numeroDeCollege;
    float moyenne;
    struct TAG_NOEUD_ETUDIANT* pProchainNoeud;
  } NOEUD_ETUDIANT; //nom du type
```

Listes chaînées - Créer un noeud

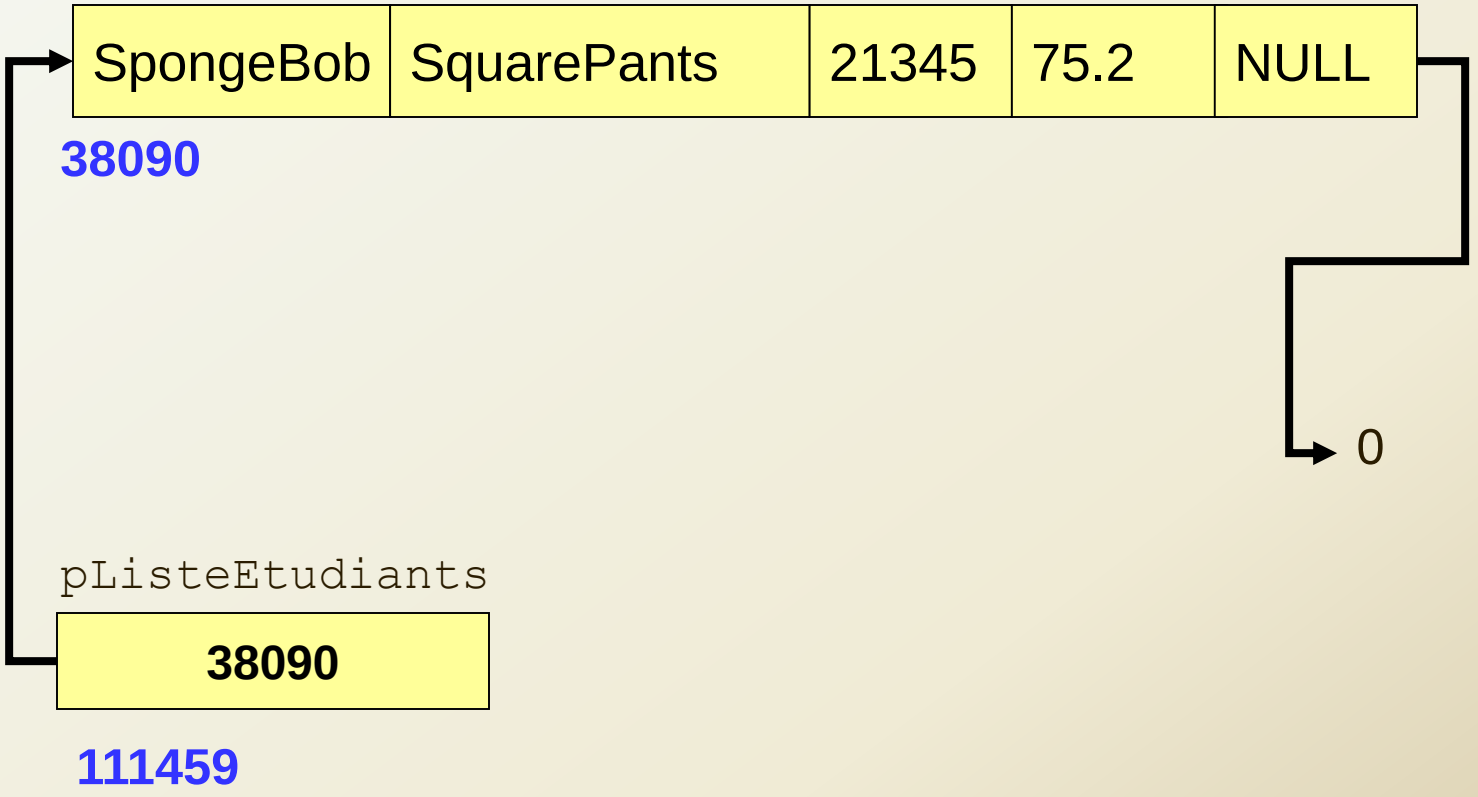
- La ***tête*** de la liste chaînée est un pointeur de type `NOEUD_ETUDIANT`:

```
NOEUD_ETUDIANT *pListeEtudiants = NULL;
```

- On peut maintenant ajouter un premier étudiant dans la liste:

```
pListeEtudiants =
   (NOEUD_ETUDIANT*)malloc(sizeof(NOEUD_ETUDIANT));
strcpy(pListeEtudiants->prenom,"SpongeBob");
strcpy(pListeEtudiants->surnom,"SquarePants");
pListeEtudiants->numeroDeCollege = 21345;
pListeEtudiants->moyenne = 74.2;
pListeEtudiants->pProchainNoeud = NULL;//le prochain nœud existe pas
```

Listes chaînées - Créer un noeud

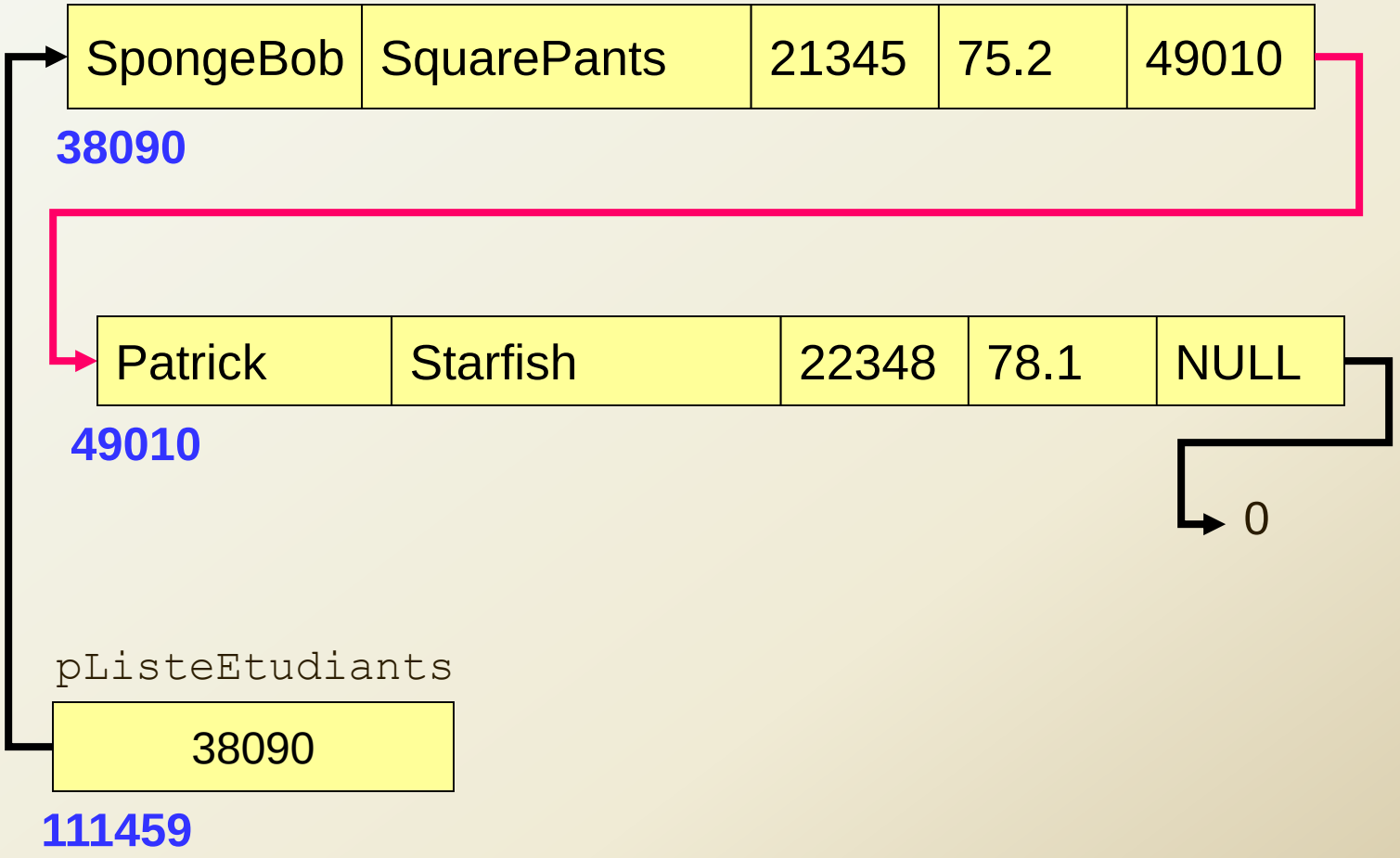


Listes chaînées - Ajouter un noeud

- on peut ensuite créer un nouvel étudiant dans la liste après SpongeBob:

```
NOEUD_ETUDIANT* pNouvelEtudiant = NULL; //dans la sect déclaration
   …
pNouvelEtudiant = (NOEUD_ETUDIANT*)malloc(sizeof(NOEUD_ETUDIANT));

strcpy(pNouvelEtudiant->prenom,"Patrick");
strcpy(pNouvelEtudiant->surnom,"Starfish");
pNouvelEtudiant->numeroDeCollege = 22348;
pNouvelEtudiant->moyenne = 78.1;
pNouvelEtudiant->pProchainNoeud = NULL;

//Maintenant on lie le premier nœud au deuxième
//Fleche rouge sur la prochaine diapositive
pListeEtudiants->pProchainNoeud = pNouvelEtudiant;
```

Listes chaînées – Ajouter un noeud



Traverser une LC - Chercher dans une LC

1. Déclarer un pointeur de type NŒUD_ETUDIANT:
   NOEUD_ETUDIANT* pTraverse = NULL;

2. Mettre pTraverse à la tête de la liste : pTraverse = pListeEtudiants;

3. Vérifier si le pointeur est NULL

4. Traiter le nœud courant

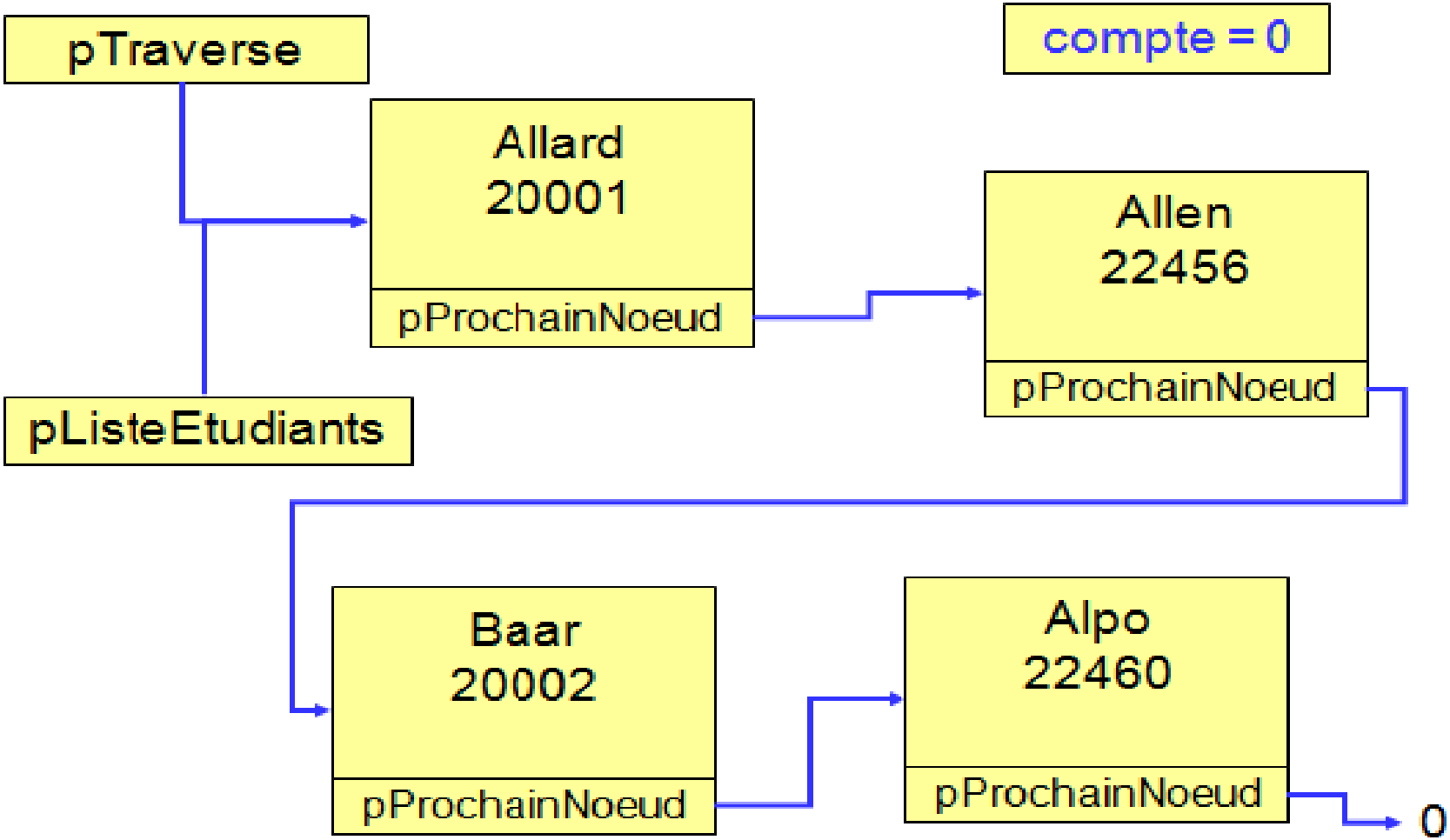
5. Mettre le pointeur sur le prochain nœud :
   pTraverse = pTraverse >pProchainNoeud;

6. Retourner à l'étape 3 jusqu'à fin

Traverser une LC - Chercher dans une LC

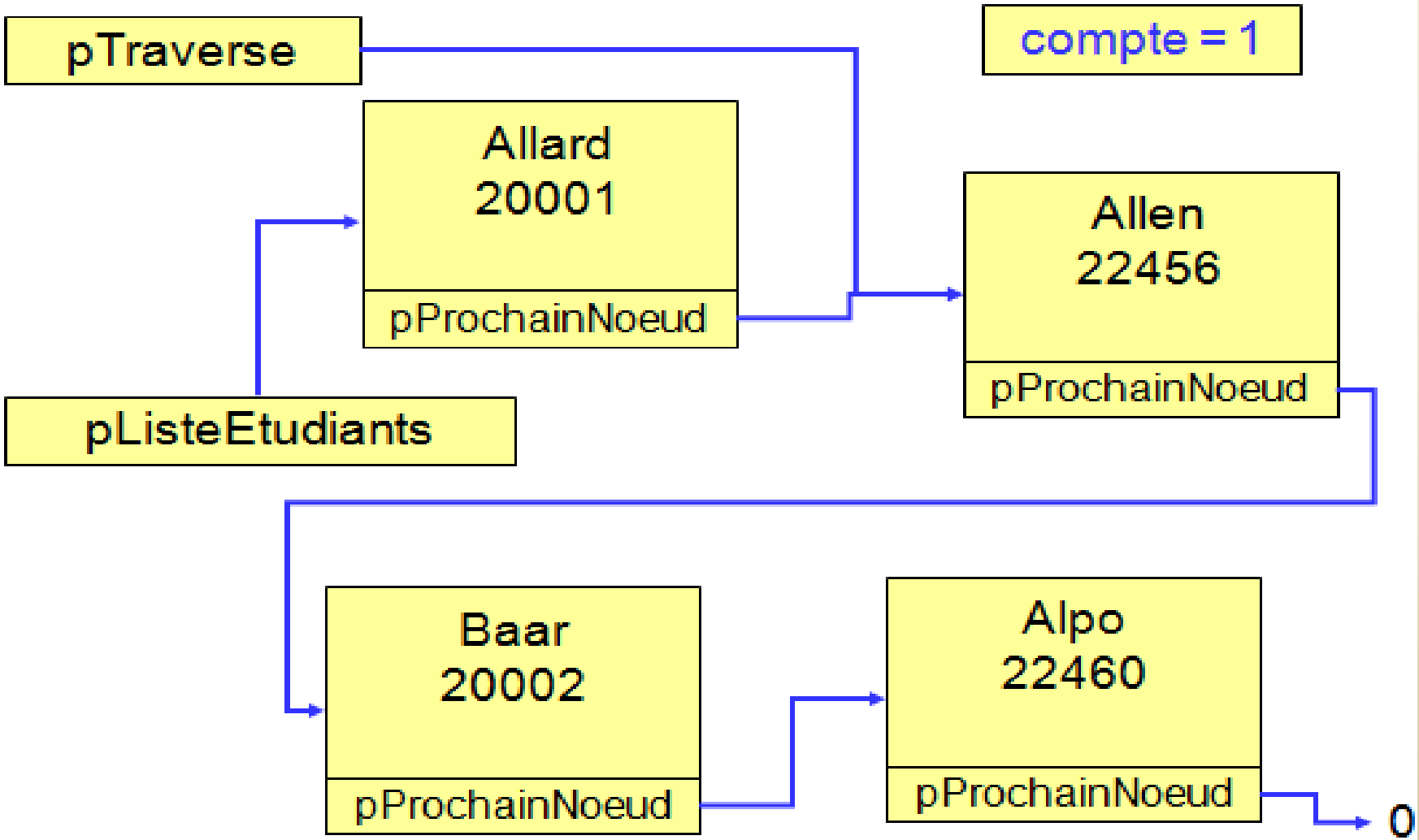
```
…
int compte = 0;
NOEUD_ETUDIANT* pTraverse = pListeEtudiants;
while (pTraverse != NULL)
{
   compte++;  //traite le noeud courrant
   pTraverse = pTraverse->pProchainNoeud;
}  //fin while pTraverse
…
```

Traverser une LC - Chercher dans une LC



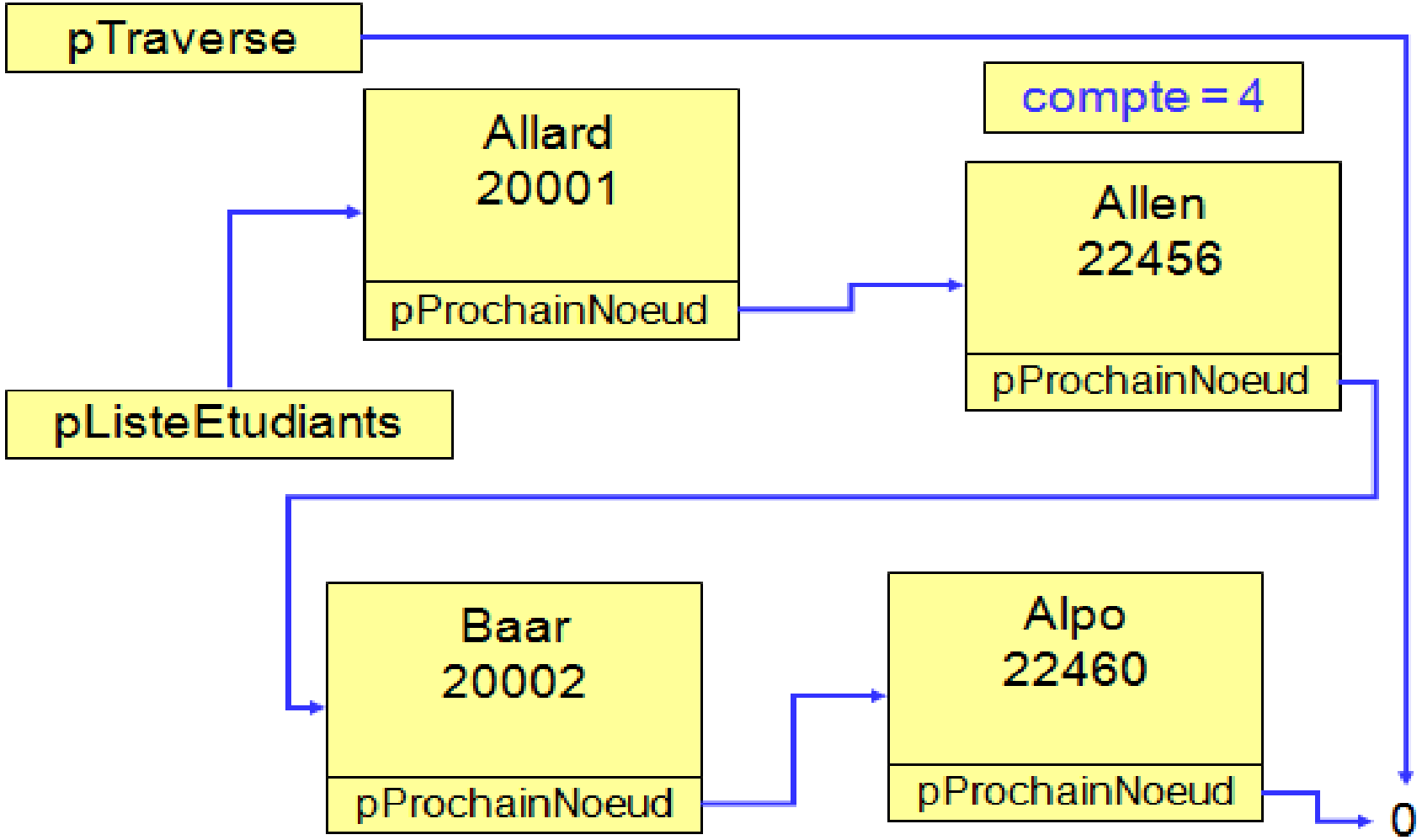
pTraverse = pListeEtudiants;

Traverser une LC - Chercher dans une LC



pTraverse = pTraverse->pProchainNoeud;

Traverser une LC - Chercher dans une LC




```
while (pTraverse->pProchainNoeud != NULL){...}
```

Insertion d'un nœud à la tête d'une LC

1. Déclarer un nœud temporaire (pTemp) et demander de la mémoire avec `malloc`
   - On vérifie s'il y a assez de mémoire
2. Mettre tous les champs du nœud pointé par pTemp à leurs valeurs appropriées
3. Pour insérer un nœud au début de la liste:
   a. À la tête de la liste:
      i. Pointer le nouveau nœud au premier nœud en utilisant le pointeur de tête:

         pTemp->pProchainNoeud = pListeEtudiants;

      ii. Pointez le pointeur de tête au nouveau nœud (pTemp)

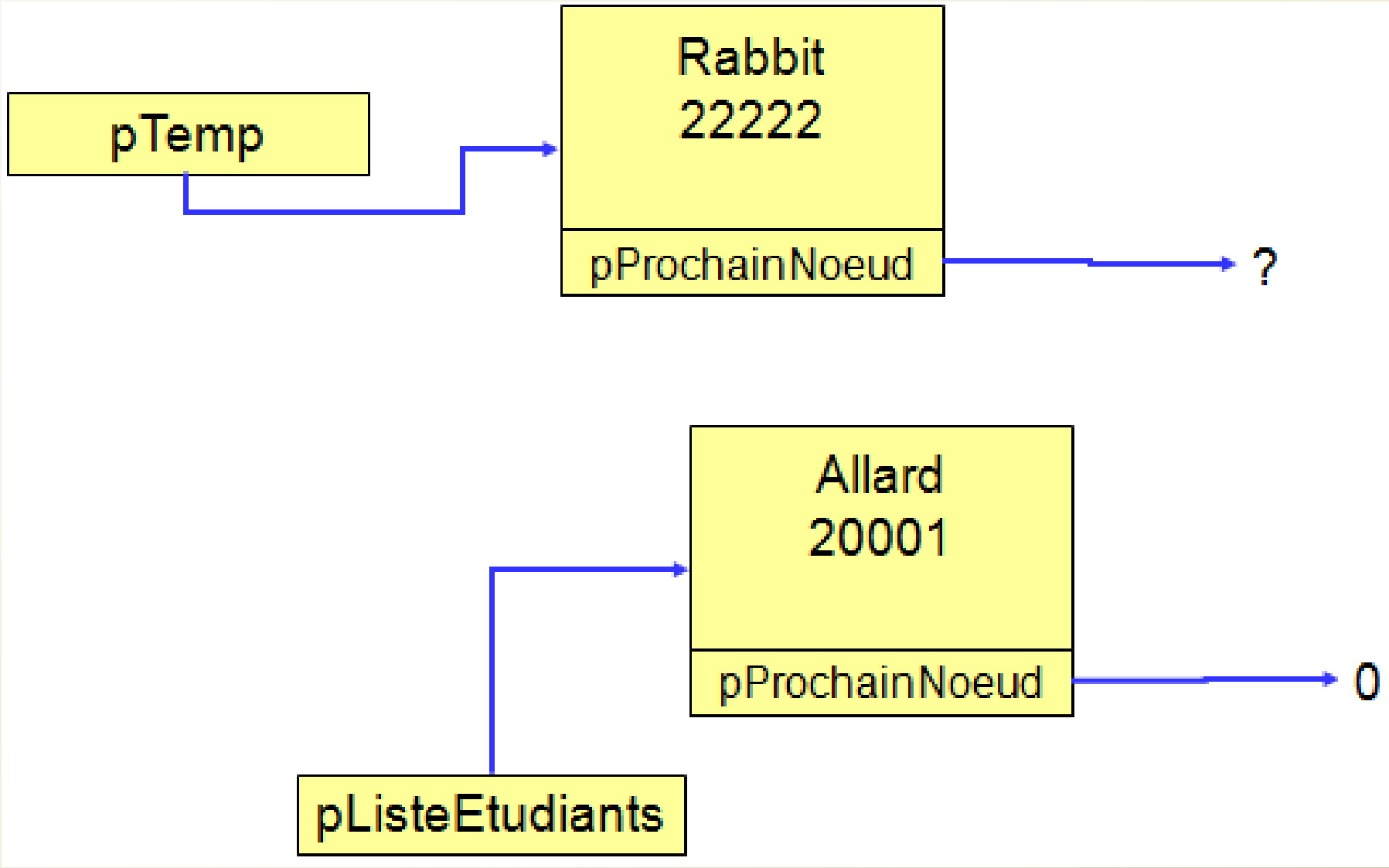
         pListeEtudiants = pTemp;

Insertion d'un nœud à la tête d'une LC

```
//1. créez un nouveau noeud

NOEUD_ETUDIANT* pTemp = NULL;
pTemp = (NOEUD_ETUDIANT *)malloc(sizeof(NOEUD_ETUDIANT));

//2. initialisez le nouveau noeud

strcpy(pTemp->prenom,"Jack");
strcpy(pTemp->surnom,"Rabbit");
pTemp->numeroDeCollege = 22222;
pTemp->moyenne = 99.9;

//3.a Insérez le nœud à la tête de la liste

pTemp->pProchainNoeud = pListeEtudiants;
pListeEtudiants = pTemp;
```

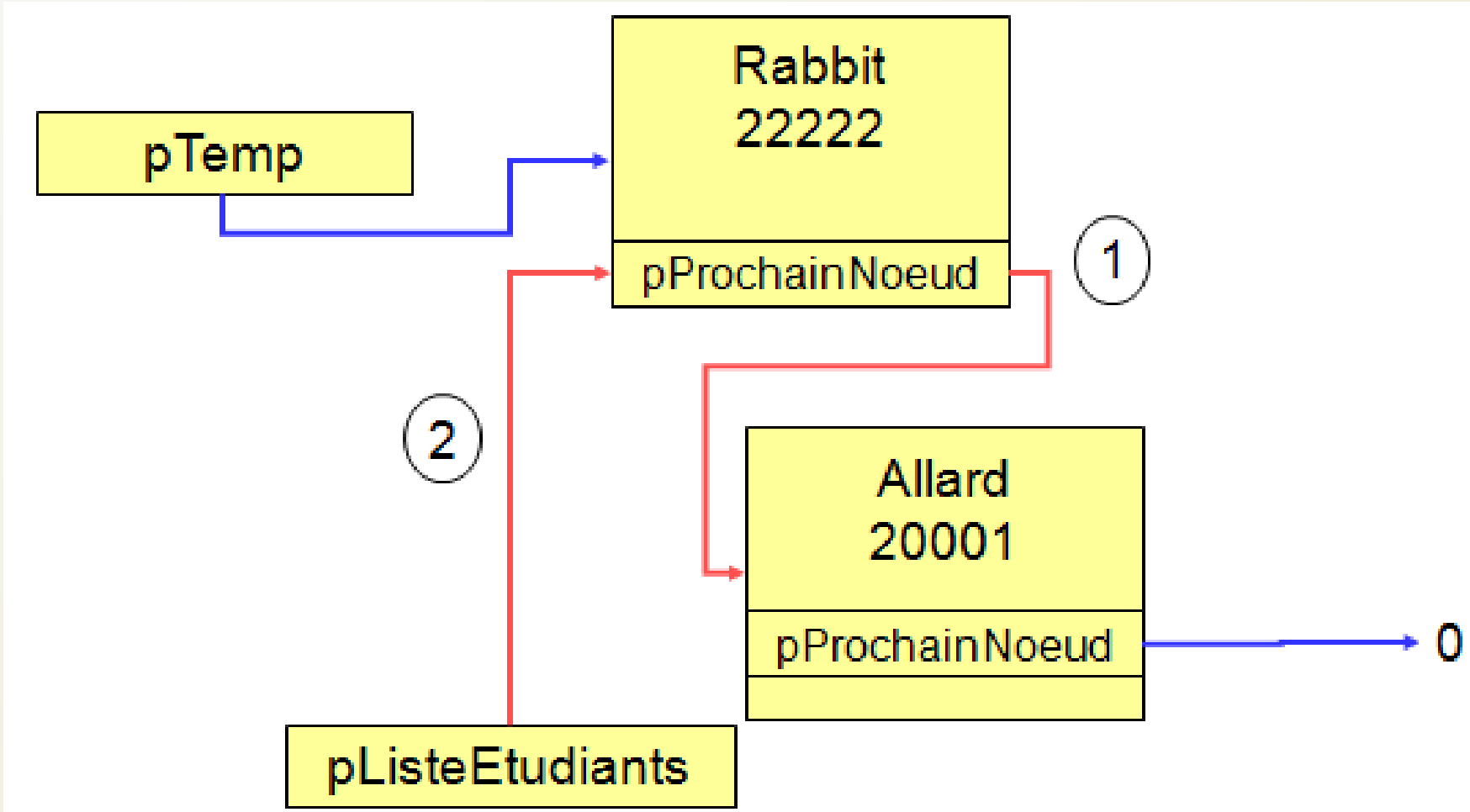
Insertion d'un nœud à la tête d'une LC

Étapes 1,2 – déclare et initialise le nœud



Insertion d'un nœud à la tête d'une LC

Étape 3a – déclare et initialise le nœud



1. pTemp->pProchainNoeud = pListeEtudiants;
2. pListeEtudiants = pTemp;

Insertion d'un nœud après la tête de la LC

1. Créer un nouveau nœud
2. Initialiser le nouveau nœud
3. Pour insérer dans la LC:
   b. N'importe où d'autre que la tête dans la LC
      i. Traversez la liste en utilisant pTraverse (ici nous assumons que nous voulons insérer *après* pTraverse)
      ii. Pointez le nouveau nœud ***où*** pTraverse pointe (indirection)

```
pTemp->pProchainNoeud = pTraverse->pProchainNoeud;
```

      i. Pointez pTraverse au nouveau nœud

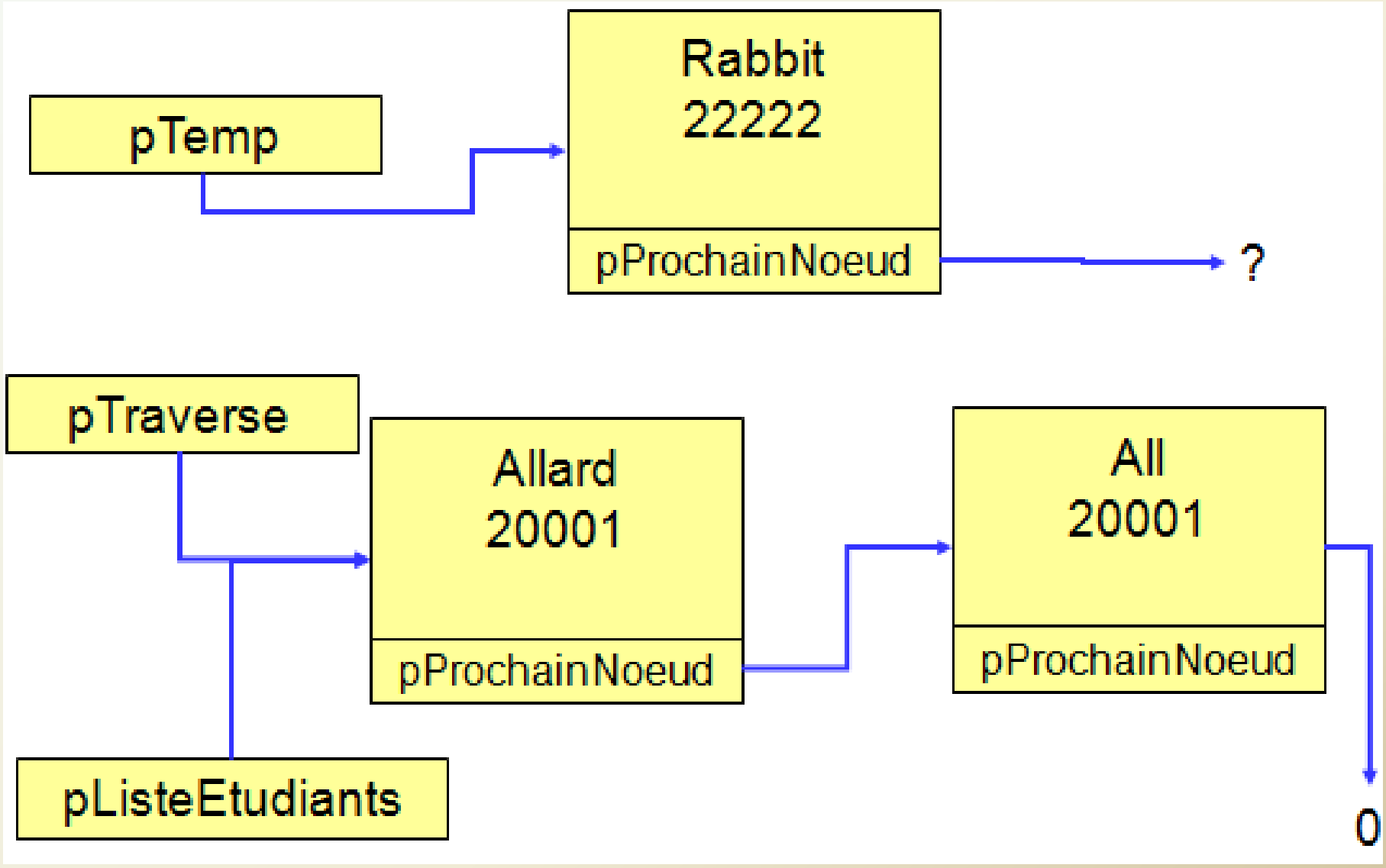
```
pTravers->pProchainNoeud = pTemp;
```

Insertion d'un nœud après la tête de la LC

```
//1. créez un nouveau noeud
NOEUD_ETUDIANT* pTemp = NULL;
pTemp = (NOEUD_ETUDIANT *)malloc(sizeof(NOEUD_ETUDIANT));

//2. initialisez le nouveau nœud
strcpy(pTemp->prenom,"Jack");
strcpy(pTemp->surnom,"Rabbit");
pTemp->numeroDeCollege = 22222;
pTemp->moyenne = 99.9;

//3.b. insérez le nœud n'importe où d'autre dans la LC
pTemp->pProchainNoeud = pTraverse->pProchainNoeud;
pTraverse->pProchainNoeud = pTemp;
```

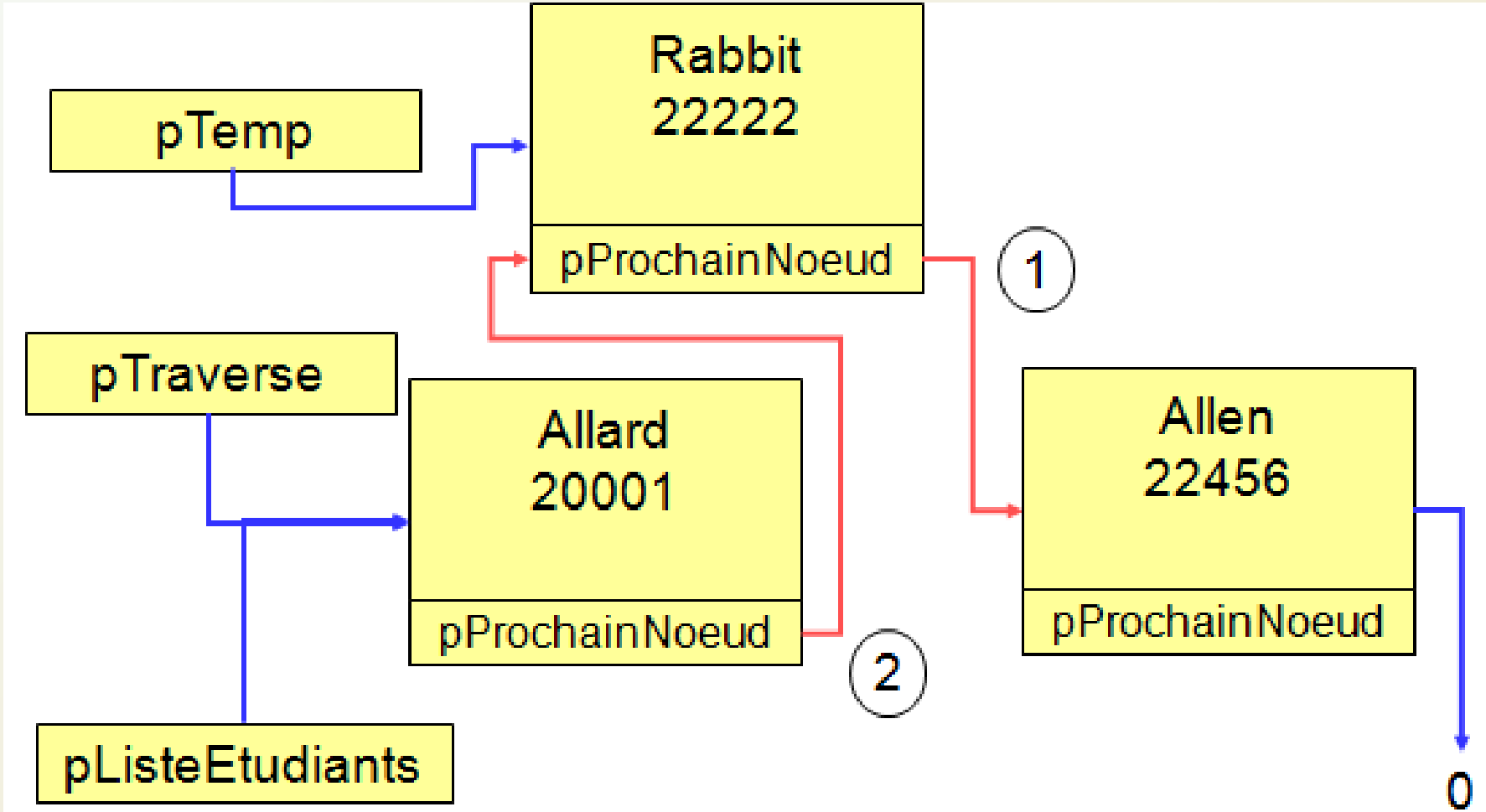
Insertion d'un nœud après la tête de la LC

Étapes 1,2 – Déclare et initialise le nouveau nœud



Insertion d'un nœud après la tête de la LC

Étape 3b – Déclare et initialise le nouveau nœud



1. pTemp->pProchainNoeud = pTravers->pProchainNoeud;
2. pTravers->pProchainNoeud = pTemp;

Effacer un nœud à la tête de la LC

1. Déclarer un nouveau pointeur appelé pPred qui suit pTraverse mais un nœud en arrière

2. Vérifiez si la liste est vide et imprime une erreur si c'est le cas

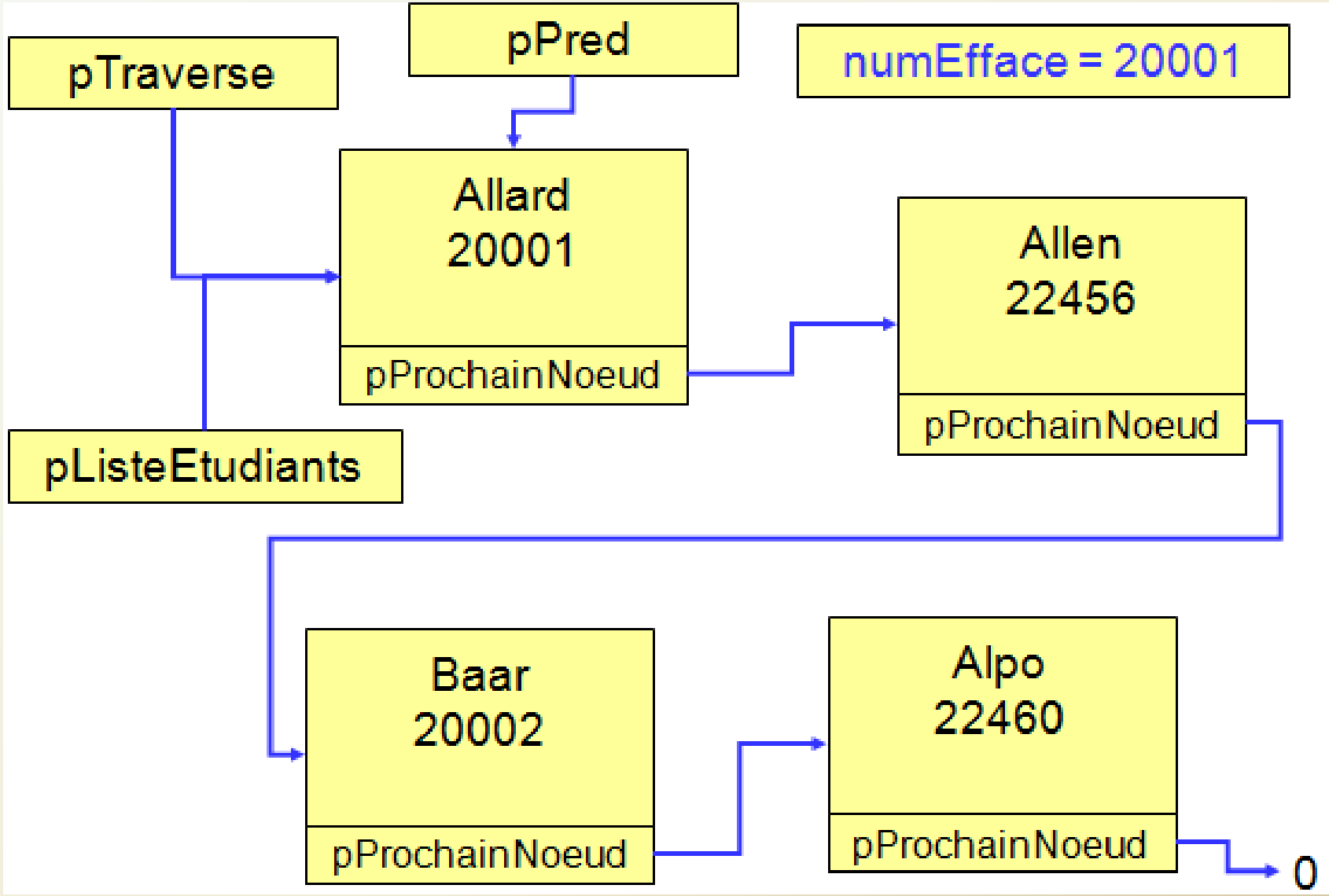
3. Vérifiez si le nœud à être effacé est le premier (la tête)
   a. nous mettons la tête au prochain nœud dans la liste.
      pListeEtudiants = pListeEtudiants->pProchainNoeud;
   b. Libèrez la mémoire

Effacer un nœud à la tête de la LC

```
…//Condition spéciale si le noeud est la tête
unsigned long numEfface = 0;

//1. Déclarer deux pointeurs
NOEUD_ETUDIANT *pTraverse = pListeEtudiants;
NOEUD_ETUDIANT *pPred = pListeEtudiants;

//2. vérifier si la liste est vide
if (pListeEtudiants == NULL){…} //imprimer erreur et sortir

//3a. Efface la tête de la liste parce que c'est le bon nœud
if (pListeEtudiants->numeroDeCollege == numEfface)
{
    pListeEtudiants = pListeEtudiants->pProchainNoeud;
    free(pTraverse);
}
else
    pTraverse = pTraverse->pProchainNoeud;
    …      //pTraverse est maintenant en avant de pPred
```

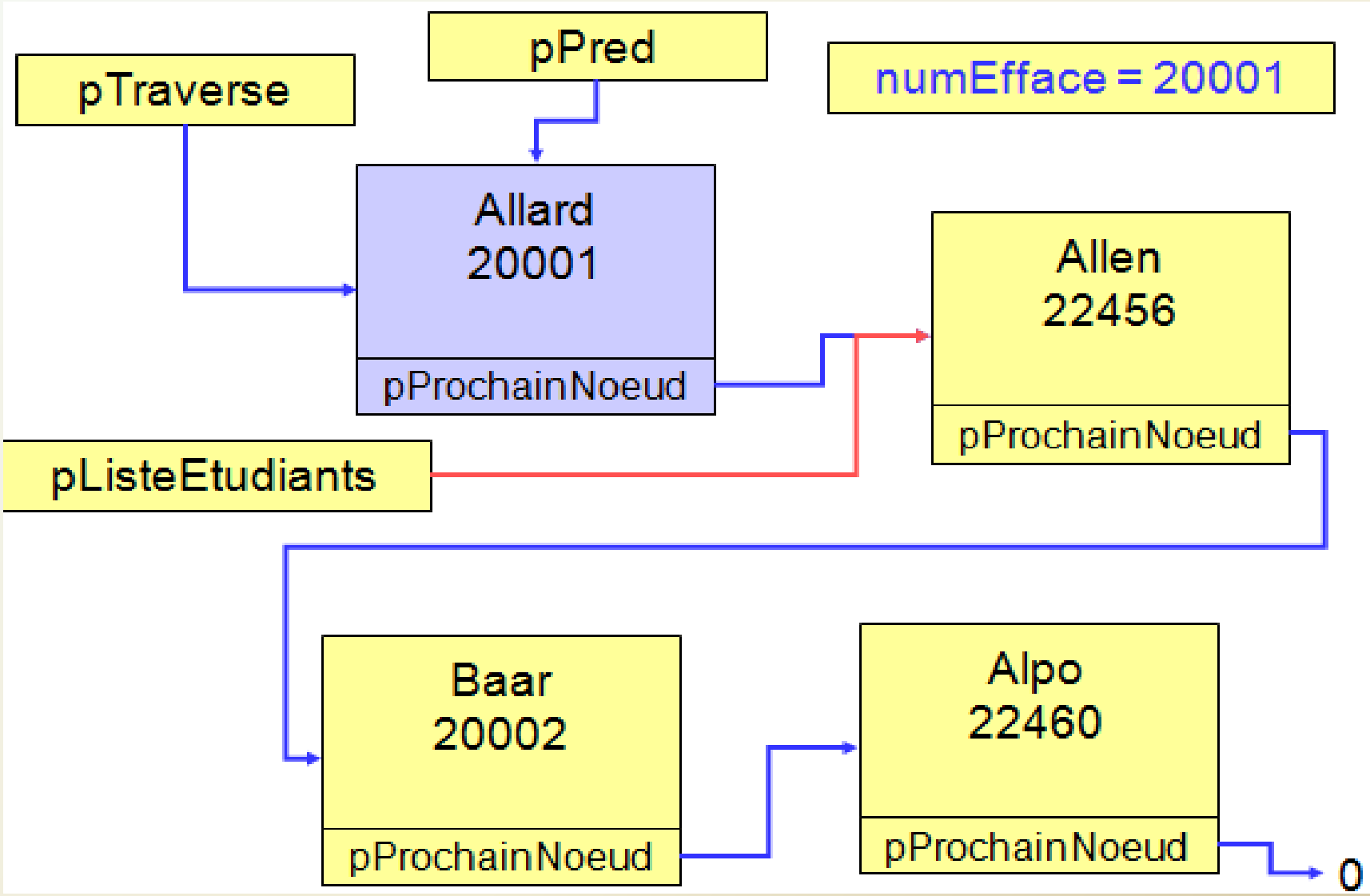
Effacer un nœud à la tête de la LC

Étapes 1,2 – trouve le nœud à effacer



Effacer un nœud à la tête de la LC

Étapes 3a,b – efface le nœud et libère la mémoire



Effacer un nœud après la tête de la LC

3. Voir si le nœud de tête doit être effacé

4. On fait une recherche pour le nœud à être effacé en testant si la condition numeroDeCollege est verifiée
   a. Avancer les deux pointeurs (pTraverse et pPred)

5. Efface le nœud qui suit pPred
   a. régler le pointeur de prochain nœud de pPred comme suit:

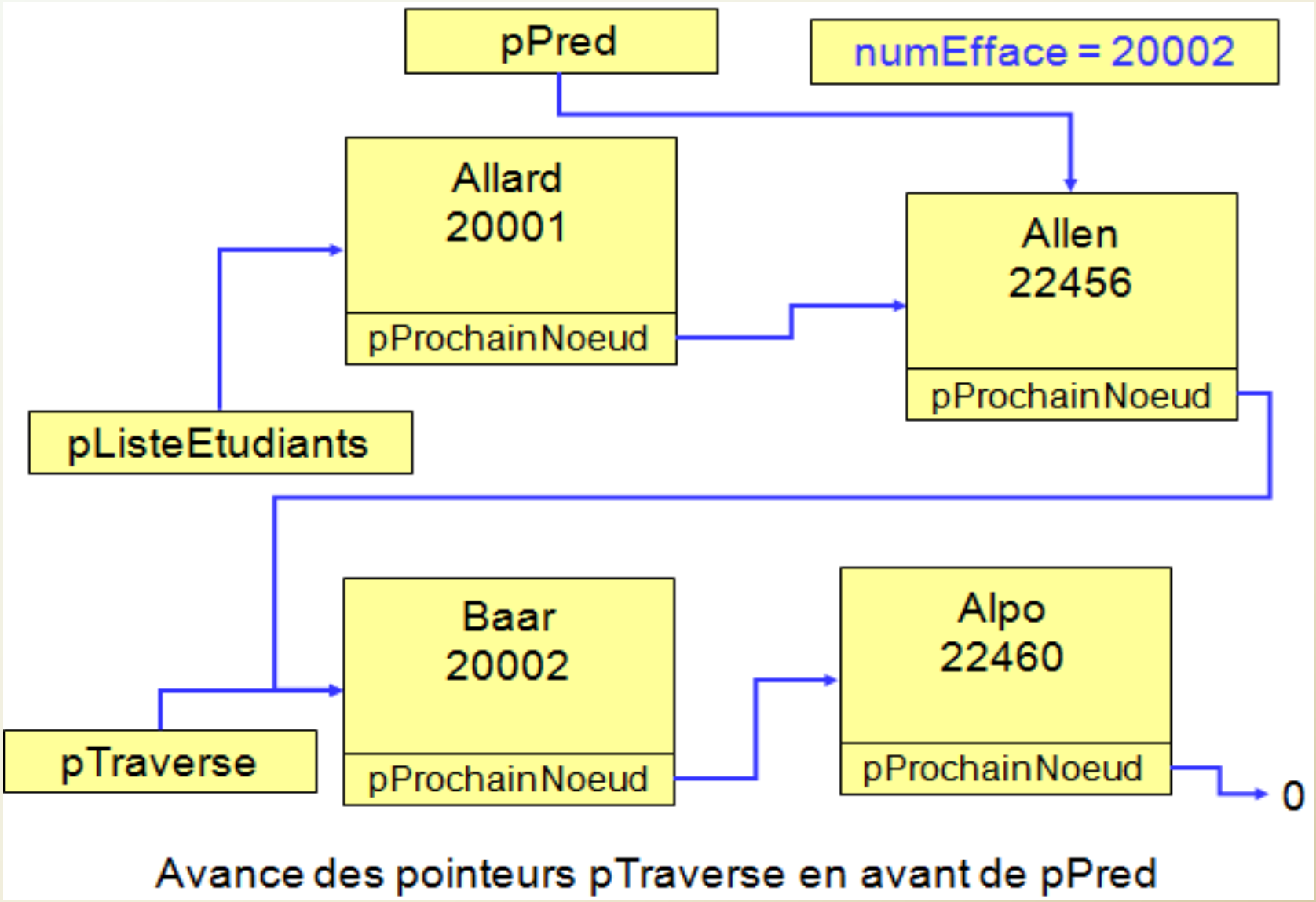
      pPred->pProchainNoeud = pTraverse->pProchainNoeud;

   b. Libère la mémoire où pTraverse pointe

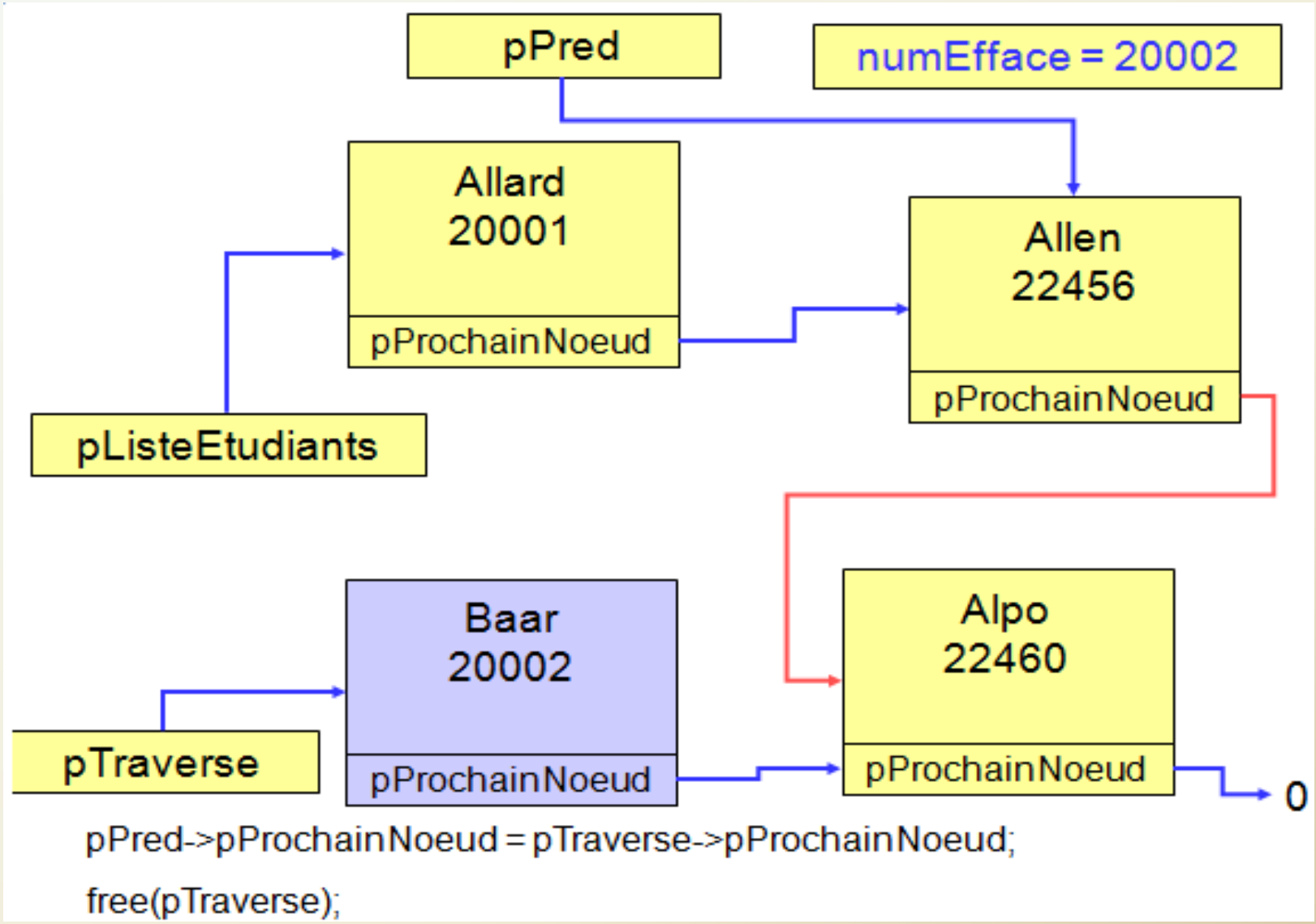
Effacer un nœud dans une LC - général

```
…//étapes 1 - 3b diapos précédents
//4. trouve le nœud à effacer
while (pTraverse->numeroDeCollege != numEfface)
{
    pPred = pTraverse;   //4a. Avance les pointeurs
    pTraverse = pTraverse->pProchainNoeud;
    if (pTraverse == NULL) {…}    //check si fin de la LC
          //si on arrive à la fin sans trouver le nœud
          //il faut donner une erreur
}
//5. Efface le nœud et libère la mémoire
pPred->pProchainNoeud = pTraverse->pProchainNoeud;
free(pTraverse);
…
```

Effacer un nœud dans une LC - général



Effacer un nœud dans une LC - général



Effacer un nœud dans une LC - général

- Vous pouvez aussi vous servir de l'indirection pour tenir le prédécesseur du nœud que vous voulez effacer
- Quand vous vous servez de ce genre d'indirection faites attention à ce que vous libérez (free)!

```
while ((pTraverse->pProchainNoeud)->numeroDeCollege != numEfface)
{
    pTraverse = pTraverse->pProchainNoeud;
    if (pTraverse->pProchainNoeud == NULL) {…} //imprime une erreur et …
}
pTemp = pTraverse->pProchainNoeud;
pTraverse->pProchainNoeud = (pTraverse->pProchainNoeud)-
    >pProchainNoeud;
free(pTemp); //Cette ligne est importante
…
```