Structures de données

Structures des données dans ce cours…

- Les Tableaux (complément)
- Les Listes chaînées
- Les Arbres
- Les graphes

Objectif de ce cours

- Découvrir les différentes structures de données
- Apprendre à les utiliser :
  - Comment les définir sur machine (implémenter) ?
  - Comment les manipuler ? (leur ajouter des éléments, les diminuer, …)
- Déterminer le domaine d'applications de chaque structure
- Apprendre à choisir la version de structure appropriée au cas traité

Compétences visée dans ce cours

- Programmer au sein d'un groupe
- Apprendre à décomposer un problème
- Apprendre séparer méthode et besogne
- Apprendre à exprimer la méthode choisie
- Apprendre à présenter son travail

Méthodologie de programmation

- Décrire de manière précise le but du Projet
- Etablir des schémas et dessins qui illustrent les principales transitions envisagées
- Produire une première écriture de l'algorithme dans une situation générale et non limite du problème
- Améliorer par « une succession de brouillons » cette première écriture en précisant à chaque fois de nouveaux points dans l algorithme (Zoom sur chaque partie de l'algorithme)
- Traiter les cas limites
- Produire une écriture globale englobant le cas général et les cas limites

Méthodologie de programmation

- Relire l'algorithme et réviser ses différentes parties
- Chercher à optimiser l'algorithme
- Traduire l'algorithme en langage
- Programmer
- Tester et valider

Méthode de production d'un programme

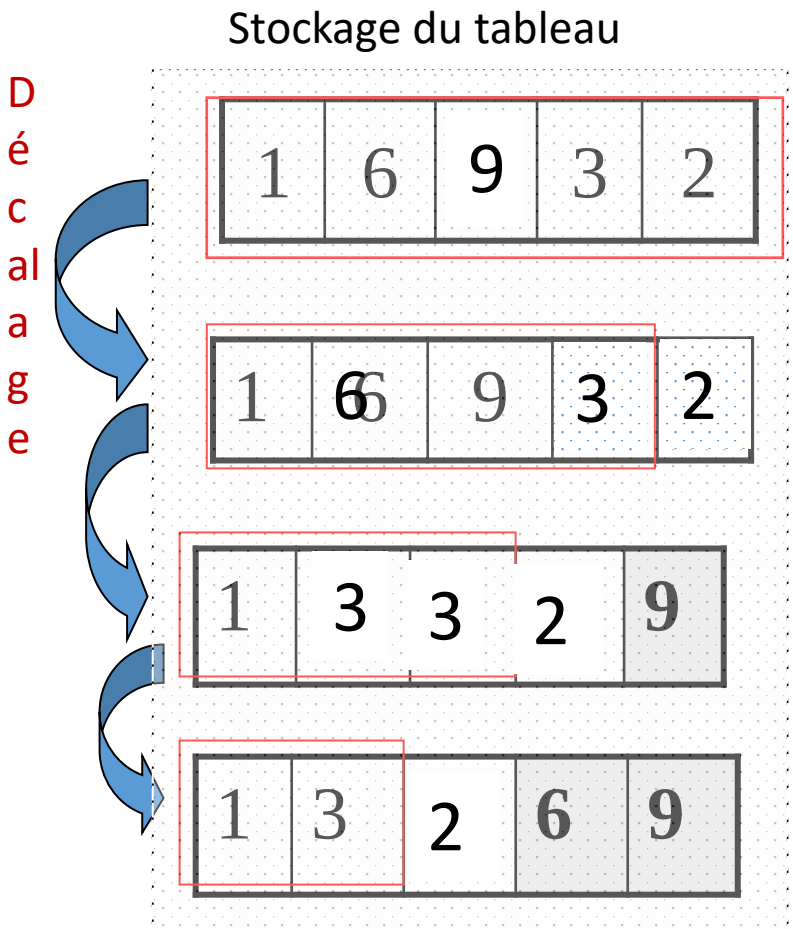
- Exemple : un tri

- Un algorithme de tri sert à ordonner les éléments d'une structure (alphabétique, croissant, décroissant…)
- Plus précisément; il faut:
  - Réorganiser l'ensemble en re-disposant ses éléments pour les mettre dans l'ordre voulu

Maintenant: **Structure** ⟶ **Tableau**

Méthode de production d'un programme

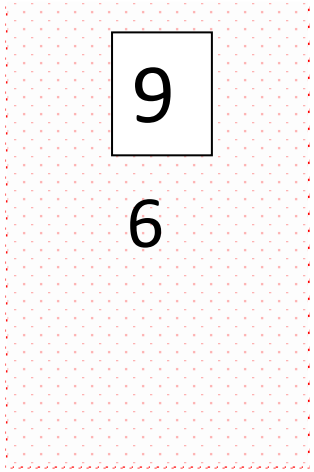
- Exemple : un tri



Trouver le Max ?

Trouver le Max ?

Mémoire de travail



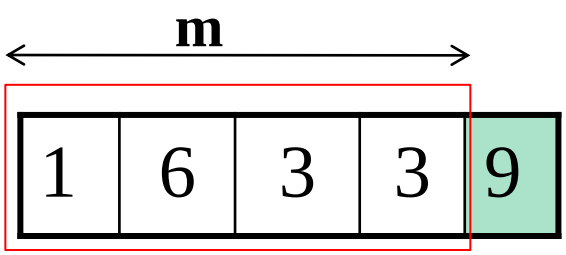
Méthode de production d'un programme

- Exemple : un tri

- **Pour chaque sous tableau** ⟶ **Boucle sur m de n à 2**
- **Déterminer le max & le stoker** ⟶ **À détailler davantage**
- **Décaler (pour libérer la place du max)** **Boucle et affectations**
- **Insérer le max** **Tab[m] <- max**

Méthode de production d'un programme

- Exemple : un tri (nouvelle écriture)



Déterminer le max :

```
Max <- Tab[1]
 k <- 1
Pour j= 2,m
   Si (Tab[j]>Max)
   Max <-Tab[j]
   k <- j
   Fin Si
Fin pour j
```

- Pour m=n,2
- Déterminer le max
- *(supposons que* max = Tab[k])
- Mem<-Tab[k]
- Pour j=k+1, m
- Tab[j-1] <-Tab[j]
- Fin pour k
- Tab[m] <- Mem
- Fin pour m

Méthode de production d'un programme

- Exemple : un tri (Finalisation de l'écriture)

- Regrouper toutes les parties écrites séparément
- Effectuer une dernière lecture
- Vérifier que l'algorithme traite convenablement les cas limites
- Optimiser si possible

Méthode de production d'un programme

- Exemple : un tri (optimisation et amélioration)

- Apres avoir formuler une version complète de l'algorithme :
- … ici on peut remplacer les décalages par une permutation entre :
  - La case où se trouve le max relatif
  - La dernière case du sous tableau

Complexité algorithmique

- Étudier la Complexité d'un algorithme consiste à :
  - Déterminer approximativement le nombre d'opérations (arithmétique, logique, affectation,…) qu'il lui faut pour s'exécuter dans le pire des cas.

- DE CE FAIT
  - Complexité est directement liée à Temps d'exécution
  - Complexité est fonction de : n (taille du tableau)
  - Complexité est un $O(n)$, $O(n^2)$ …etc.

Structures évoluantes

Notion de Structure et notion d'ensemble

- Structure vs Ensemble

  Une structure, *comme par exemple « les personnes qui sont connectées et consultent ce cours »* diffère de « l'ensemble » par ce que :

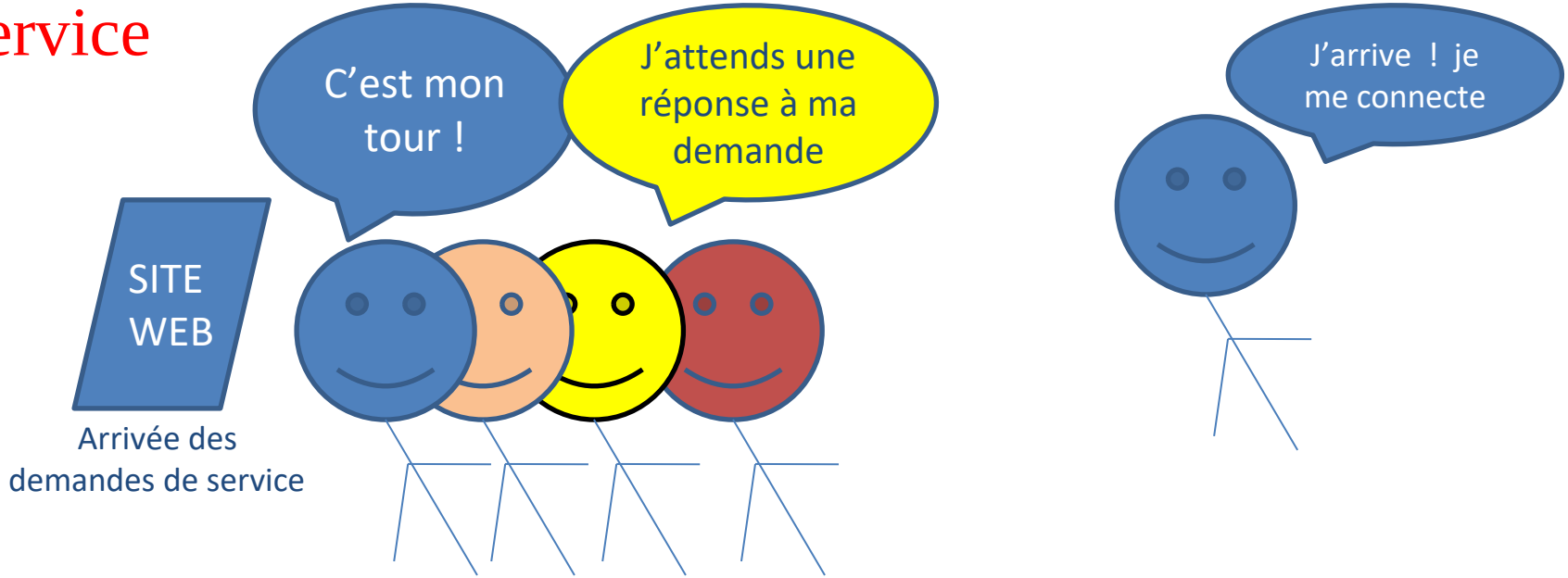
  – Elle peut augmenter ou diminuer, voire changer d'éléments, tout en préservant son identité.

  – L'évolution de la structure peut même être régie par des lois établies, ou alors être aléatoire.

Commentaire : Exemple les listes chainées, les arbres, les graphes évoluent au cours du temps, une liste chainée L même lorsqu'elle change d'éléments ne change pas de nom ni de pointeur TETE.

File

- Liste des utilisateurs d'un site WEB qui attendent chacun un service



- Le serveur ne peut pas traiter à la fois toutes les demandes soumises. Il les stocke pour les traiter ensuite une dans l'ordre de leur arrivée.
  - A chaque fois, il doit traiter la plus ancienne parmi celles qui attendent.

Commentaire : Lorsqu'il y a beaucoup de demandes de service en attente, on doit savoir par qui commencer. La réglementation de la priorité doit être implémentée dans le logiciel qui prend en charge le traitement des services.

File

- La file d'attente des demandes de retrait aux guichets automatiques d'une banque

  Parmi les présents, on doit servir le premier à être arrivé.

- Une file est une structure dont la loi d'évolution respecte la règle FIFO (First In First Out)
  - La diminution de la structure doit se faire exactement par le plus ancien de ses éléments.
  - Nous devons mémoriser l'ancienneté de chaque élément présent.
- Le lien entre les éléments est de type suivant-précédent.

Les applications qui se basent sur FIFO son très nombreuses en pratique. Elle illustre le principe de priorité par l'ancienneté. Mis à part les cas limites, les point d'entrée et de sorties doivent être différents.

File

- Réalisation d'une file

Évolutivité → **Structure dynamique**

Lien (succession et précédence) → **Chainage un à un**

Donc: Utilisation d'une liste chainée.

Chaque élément de la file sera représenté par un élément de la liste chaînée !

On peut réaliser la file avec un tableau (variable statique) à condition de connaitre au préalable le nombre max d'élément qu'elle peut avoir (lorsque ce nombre existe!)

Pile

- Pile de programmes en exécution

La Pile de programmes qui, pour leur exécution, font appel à d'autres programmes *: parmi les présents, on doit servir le dernier à être arrivé.*

```
Programme A1
Début
    .
    Exécuter
A2
    .
    .
Fin
```

```
Programme A2
Début
   .
   Exécuter A3
   .
   .
fin
```

```
Programme A3
Début
   .
   Exécuter A4
   .
   .
fin
```

```
Programme A4
Début
   .
   .
   .
fin
```

- *Pour exécuter A1 on doit d'abord exécuter A2*
- *Pour exécuter A2 on doit d'abord exécuter A3*
- *Pour exécuter A3 on doit d'abord exécuter A4*

On exécute toujours le dernier à avoir rejoint la structure !

Pile

- Pile de programmes en exécution

- Définition : Une Pile est une structure dont la loi d'évolution respecte la règle LIFO (Last In First Out) :
  - La diminution de la structure doit se faire exactement par le plus récent de ses éléments.
  - Nous devons mémoriser l'ancienneté de chaque élément.
  - Le lien entre les éléments est de type : suivant-précédent

Pile

- Réalisation d'une pile

Évolutivité → **Structure dynamique**

Lien (succession et précédence) → **Chainage un à un**

Donc: Utilisation d'une liste chainée.

Chaque élément de la pile sera représenté par un élément de la liste chaînée !

Listes chainées

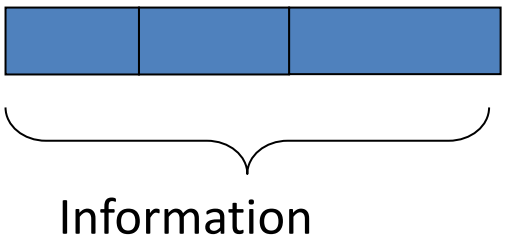
Listes chainées

- Composition de l’élément de la L.C.
- Construction de la liste chainée.
- Ajouter en tête de la L.C.

Exemples de parties information

Liste d'étudiants: on retient le **nom**, le **prénom** et **l'adresse**

La partie Information est un **enregistrement** composé comme suit:



Information de type composé :

{

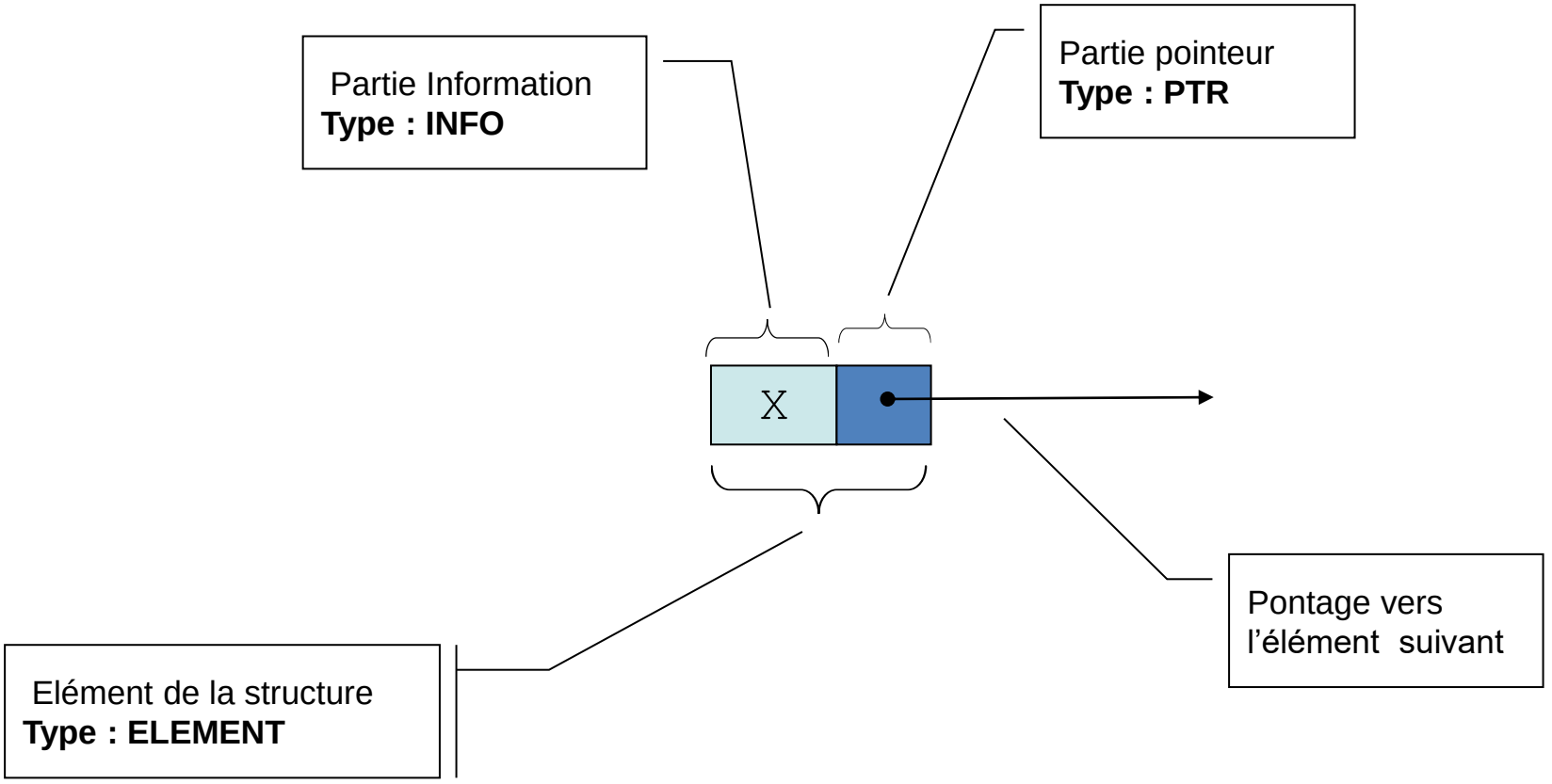
nom de type chaine de 20 caractères

prénom de type chaine de 20 caractères

adresse de type chaine de 80 caractères

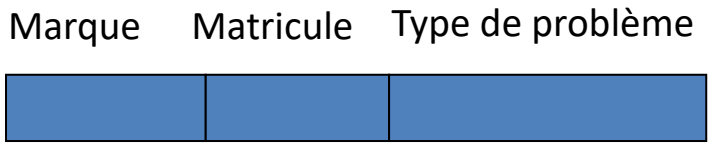
}

Composition de l'élément de la structure chainée



- Dans un garage on considère la Liste de **véhicules** réparés
  - chaque véhicule est représenté par:
    - sa **marque**,
      - son **matricule**
        - le **type de problème** qu'il a

- la partie information est un enregistrement composé comme suit



Exemples:

MEGANE ,2137 ب 15 ,vidange

Mercedes , 9870 أ 12 , freins

Information de type composé :

{

Marque de type chaine de 20 caractères

Matricule de type chaine de 15 caractères

Type_de_prob de type chaine de 60

}

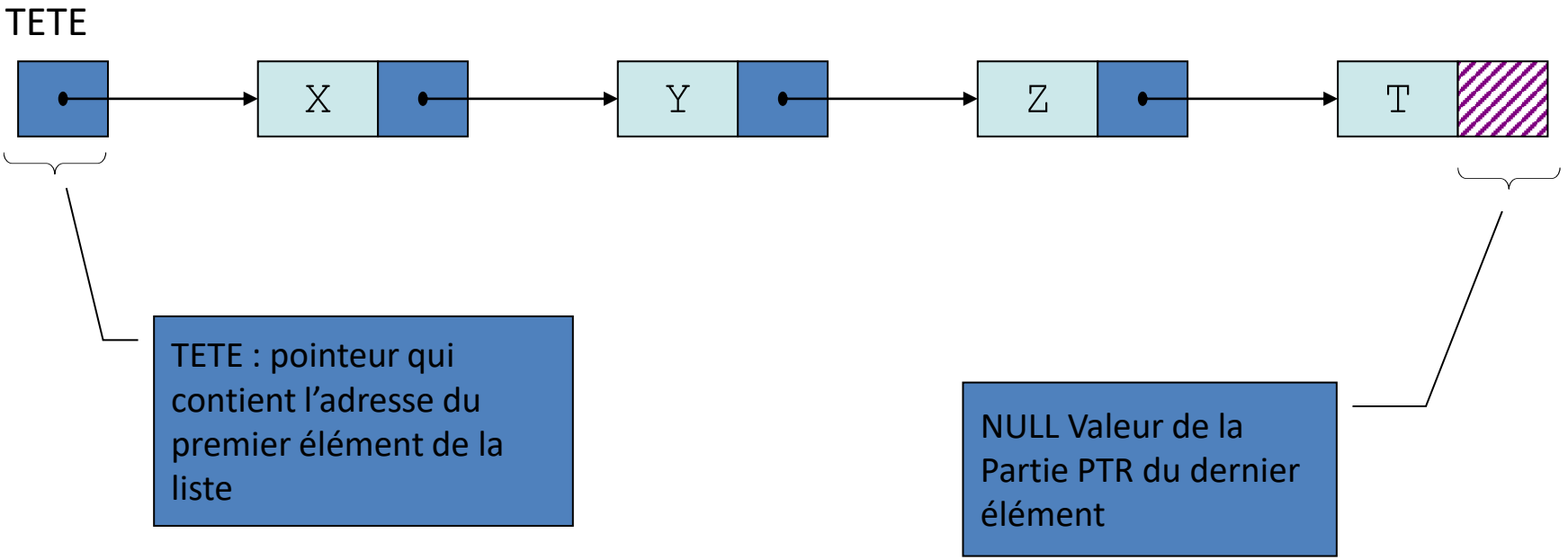
Nombres entiers obtenus lors d’un tirage:

La partie Information est constituée d’un seul champs de type entier

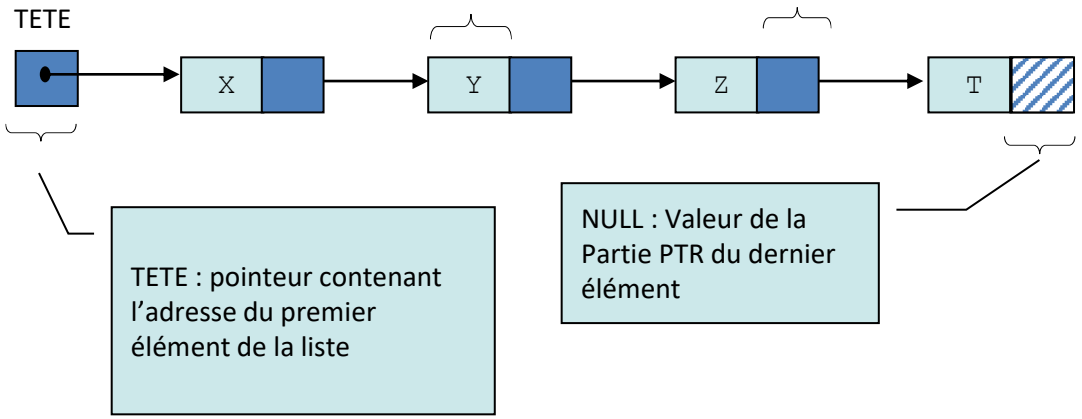
Exemple : 6,4,9,33,…

Information est de type entier

Construction de la liste chainée



Pour définir la liste chainée, il faut:



Si **X**, puis **Y**,**Z** et, enfin **T** sont les éléments de la liste

- Préciser la structure composée «ELEMENT»
- Etablir le lien de succession entre les éléments
- Préciser TETE (adresse du 1er élément)
- Caractériser le dernier élément (NULL)

INFO :type permettant d'informer les éléments de la structure

PTR : de type pointeur sur ELEMENT

Mise en œuvre /déclaration de la liste chaînée

Mise en œuvre algorithmique

ELEMENT de type composé:

{

INFO de type Information

PTR de type pointeur sur ELEMENT

}

TETE de type pointeur sur ELEMENT

Déclaration (langage C)

1er Cas simplifié: int information;

```
struc ELEMENT {
Int                    INFO;
Struct ELEMENT   * PTR  ;
}
Struct ELEMENT * TETE;
```

Mise en œuvre/déclaration de la LC

Mise en œuvre algorithmique

ELEMENT de type composé:

{

INFO de type Information

PTR de type pointeur sur ELEMENT

}

TETE de type pointeur sur ELEMENT

Déclaration (langage C)

2éme Cas simplifié avec **typedef** et int information;

```
Typedef struct ELEMENT  {
Int                INFO;
Struct ELEMENT * NEXT  ;} ELEMENT;

ELEMENT * TETE;
```

Mise en œuvre/déclaration de la LC

Mise en œuvre algorithmique

- ELEMENT de type composé:

{

INFO de type Information

PTR de type pointeur sur ELEMENT

}

- TETE de type pointeur sur ELEMENT

Déclaration (langage C)

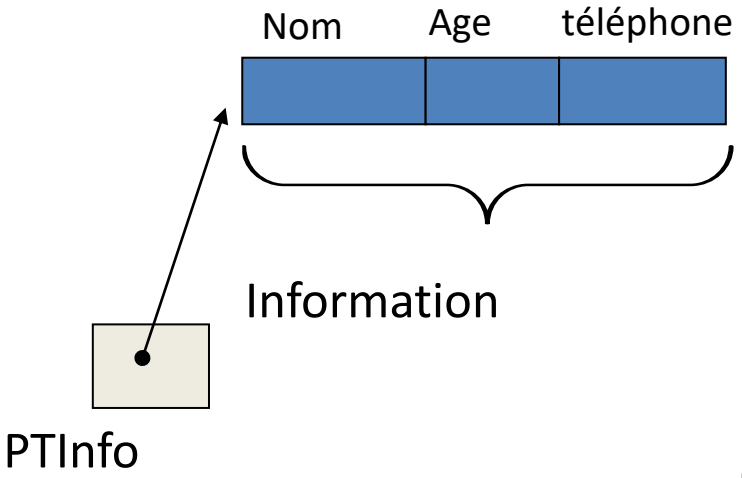
3éme Cas avec typedef et information qcq;

```
Typedef {.....} Information;
Typedef struc ELEMENT  {
Information       INFO;
Struct ELEMENT * PTR  ;} ELEMENT;
```

```
ELEMENT * TETE;
```

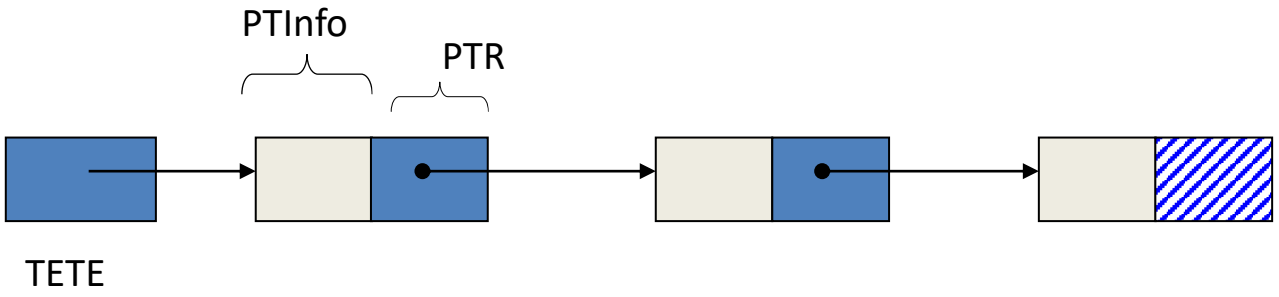
Liste chainée référencée

On utilise un pointage de la partie information :



PTInfo est un type qui pointe uniquement sur la partie Information

On peut alors définir la L.C. comme suit :

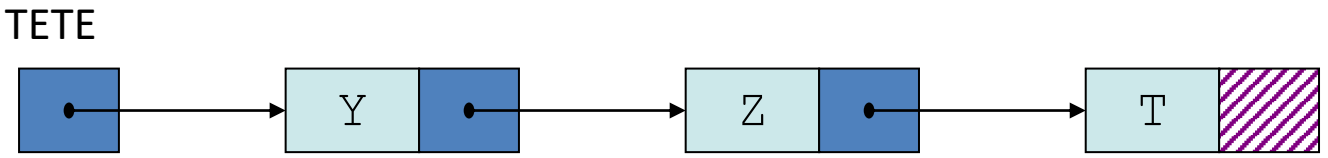


Ajouter un élément à la L.C.

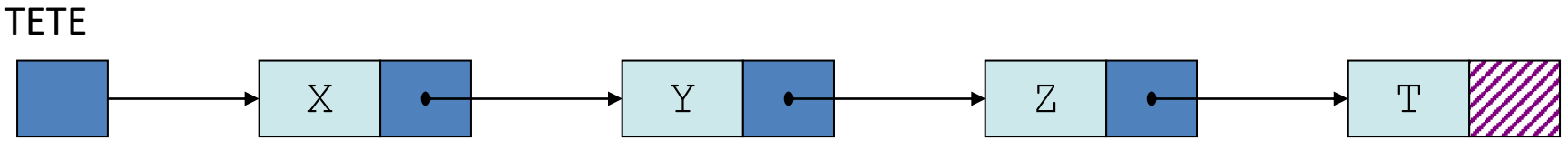
- Augmenter la structure d'un nouvel élément a plus d'une variante :
  - Ajouter en tête de la L.C.
  - Insérer au milieu de la structure
  - Ajouter en queue

Ajouter en tête de la L.C.
1- ‘cas fréquemment rencontré’

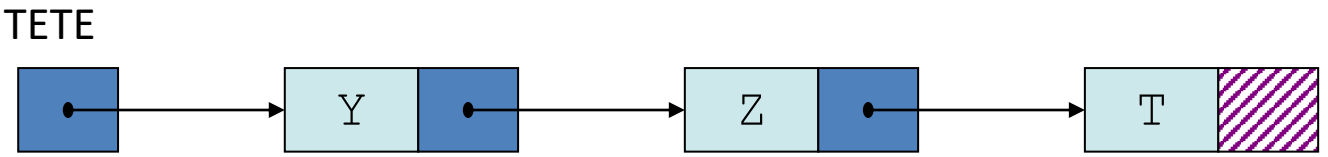
Etat avant insertion



Etat après insertion

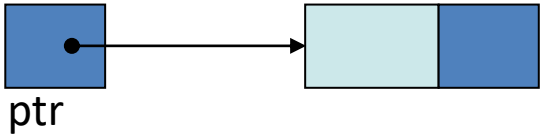


Évolution Quantitative

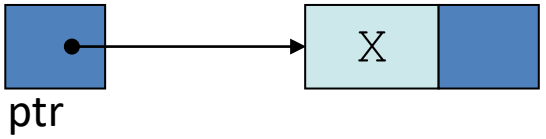


- 1 élément en plus,
- il contient la valeur X

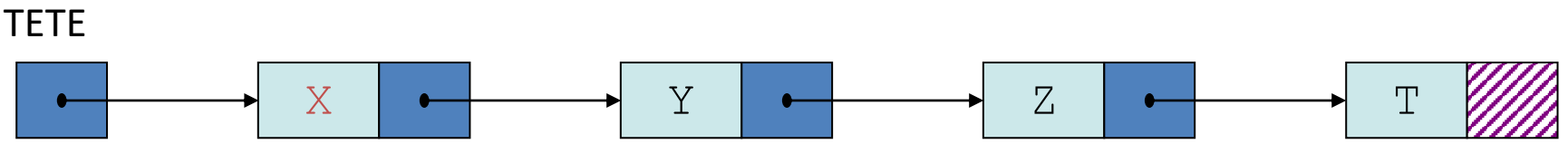
Il faut demander de la mémoire pour cet élément



ptr ← Obtenir (ELEMENT)

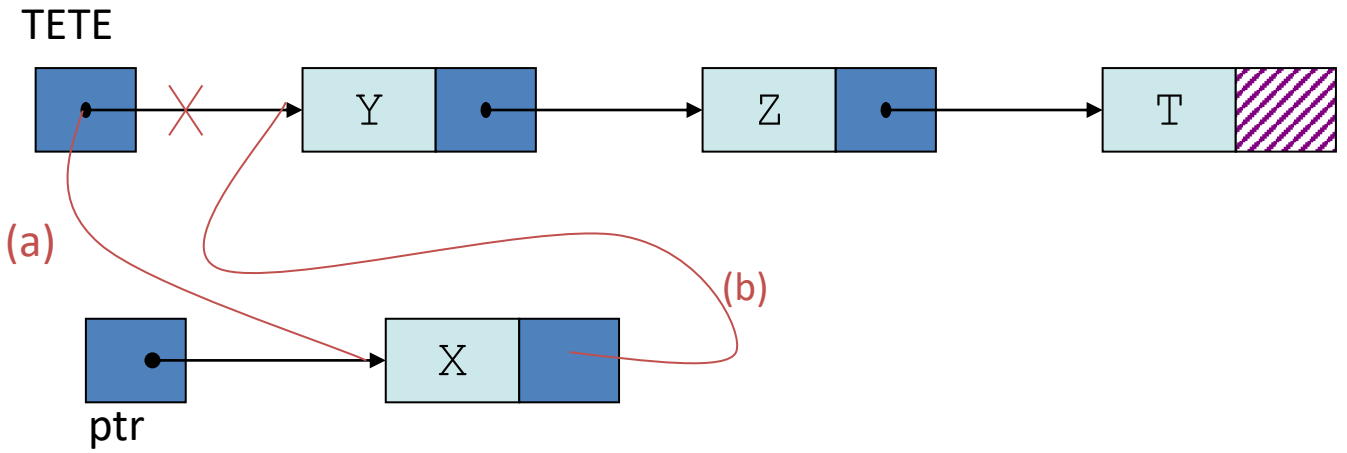


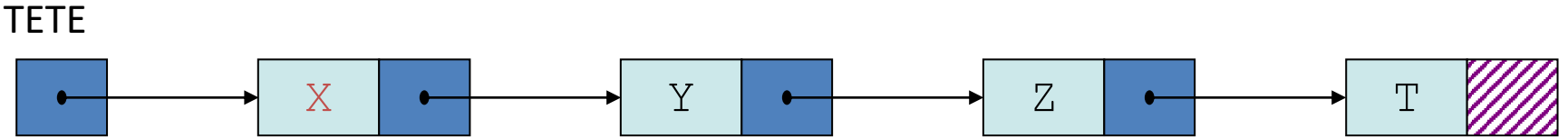
ptr → INFO ← X



Evolution qualitative (le chaînage)

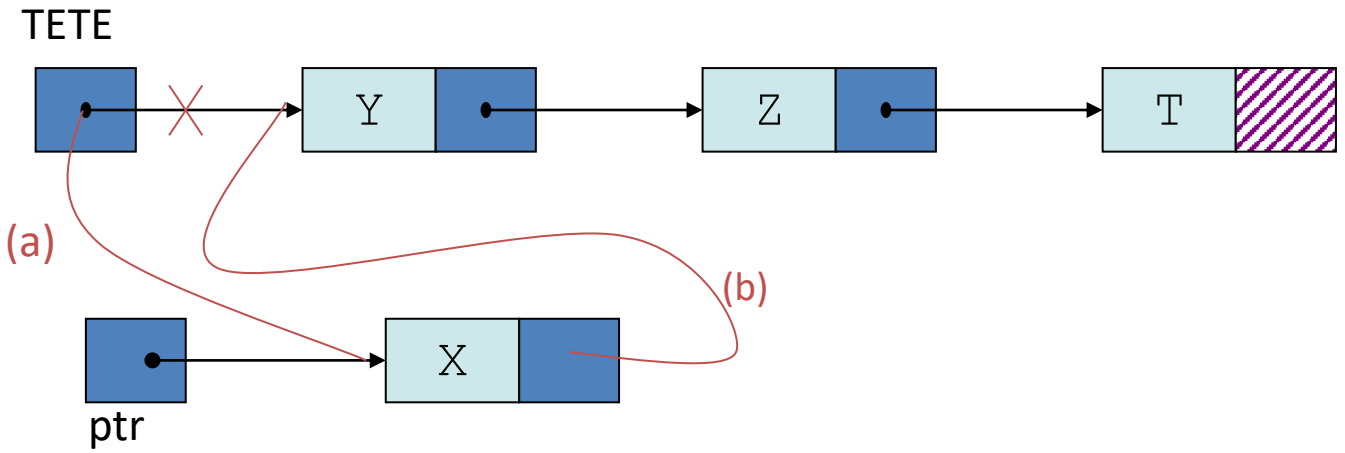
1-partant de la structure initiale, construire la nouvelle





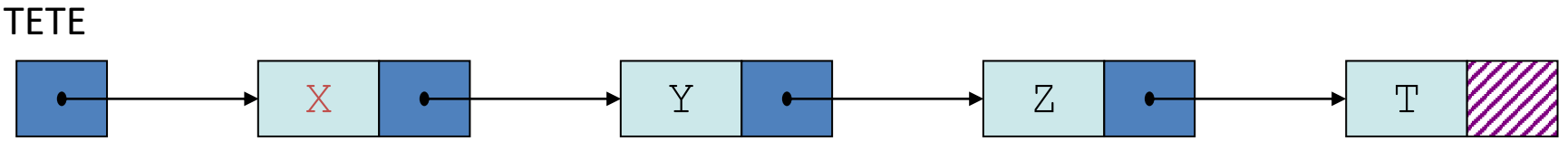
Evolution qualitative (le chaînage)

2- écriture algorithmique de chaque opération



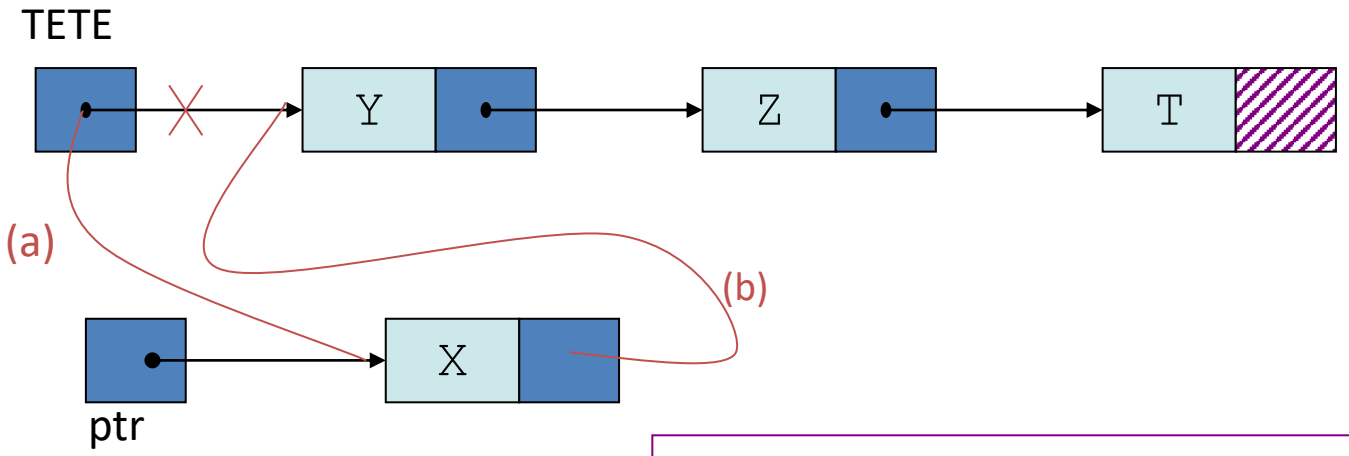
(a) ⟶ TETE← ptr ;

(b) ⟶ ptr→ PTR ← TETE ;



Evolution qualitative (le chaînage)

3-détermination de l'ordre des opérations de chaînage

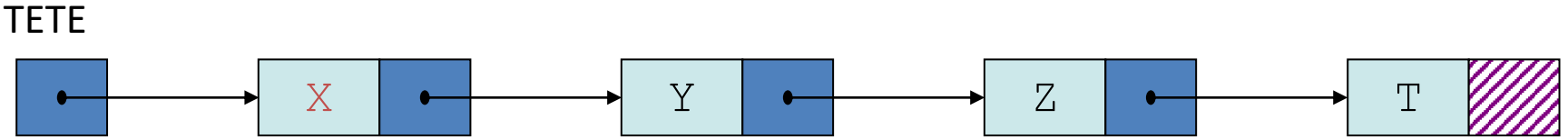


(a) ⟶ TETE← ptr ;

(b) ⟶ ptr→ PTR ← TETE ;

(b) doit se faire avant (a)…Sinon, on perd la cellule contenant Y …

… et tous le reste !



Remarques importantes concernant les structures chaînées

- d.m.g. on peut tjrs retarder les opérations qui affectent la structure, et commencer par celles qui ont lieu extra-structure
- On a intérêt à préserver (marquer par un pointeur de travail) l'adresse de la première cellule concernée par un changement!

Rédaction de l'algorithme:

Début

Variables de travail:

ptr de type pointeur sur ELEMENT

Étude quantitative:

ptr ← Obtenir (ELEMENT) ;

ptr → INFO ← X ;

Étude qualitative:

ptr → PTR ← TETE ;

TETE ← ptr ;

fin

Traduction en L.P.

Début

ptr de type pointeur sur ELEMENT

ptr ← Obtenir (ELEMENT) ;

ptr → INFO ← X ;

Ptr → PTR ← TETE ;

TETE ← ptr ;

fin

{

ELEMENT * ptr;

ptr =(ELEMENT *)malloc(sizeof(ELEMENT));

ptr->INFO = 'X';

ptr->PTR = TETE;

TETE = ptr ;

}

Etude des Cas Limites (C.L.)

- Énumération des cas limites.
- Traitement des C.L. un à un.
- Intégration des C.L. dans l’algorithme général

Énumération des cas limites

- Liste chainée initialement vide
- Liste chainé contenant initialement un seul élément (dans le doute)
- Plus de mémoires libres : exception

Liste chainée initialement vide

Etat avant insertion

TETE

Question :

L'algorithme du cas général est–t-il encore valable pour ce C.L. ?

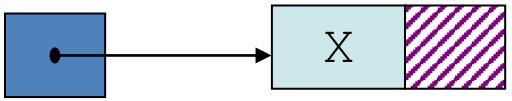
Réponse :

OUI !

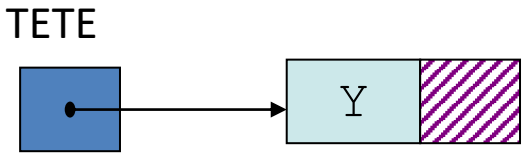
On n'a pas besoin d'écrire un algorithme particulier pour ce C.L.

Etat après insertion

TETE



Liste chainée contenant initialement un seul élément



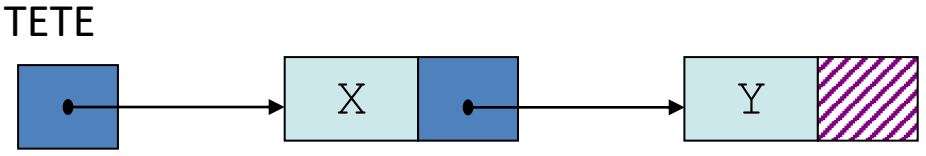
Question :

L'algorithme du cas général est–t-il encore valable pour ce C.L. ?

Réponse :

OUI !

On n'a pas besoin d'écrire un algorithme particulier pour ce C.L.



Intégration des C.L. dans l'algorithme général

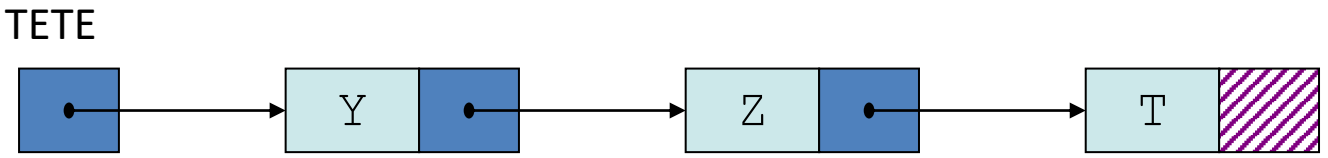
- L'algorithme général écrit pour traiter le cas « le plus fréquent » prend en compte tous les C.L. :

- Il constitue donc l'algorithme global.

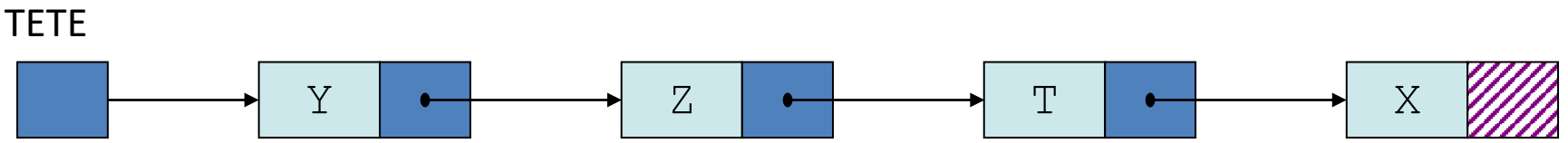
- Reste à traiter les exceptions.

Ajouter en queue de la L.C.
1- ‘cas fréquemment rencontré’

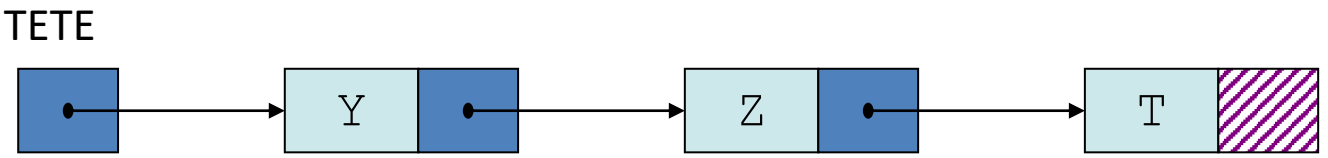
Etat avant insertion



Etat après insertion

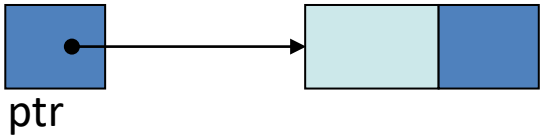


Évolution Quantitative

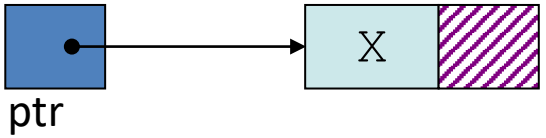


- 1 élément en plus,
- sa partie INFO contient X
- sa partie PTR contient NULL

Il faut demander de la mémoire pour cet élément

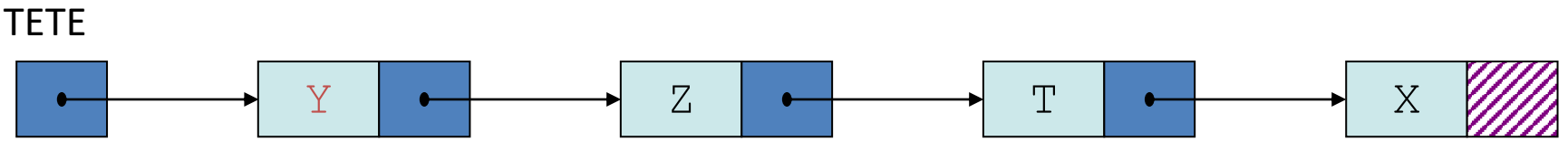


ptr ← Obtenir (ELEMENT)



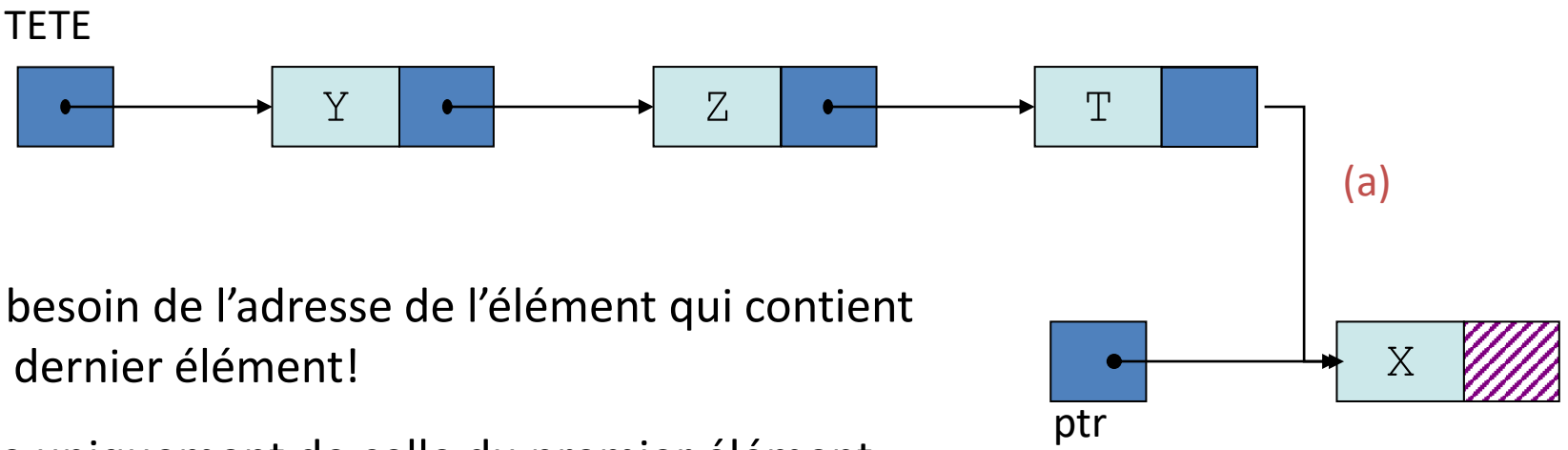
ptr → INFO ← X

ptr → PTR ← NULL



Evolution qualitative (le chaînage)

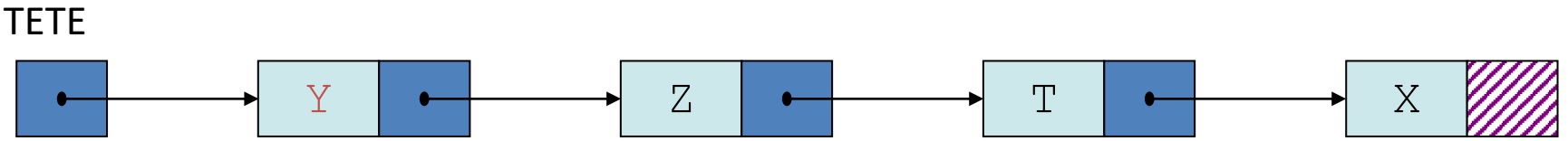
1-partant de la structure initiale, construire la nouvelle



Pour écrire (a) on a besoin de l'adresse de l'élément qui contient T c.à.d. l'adresse du dernier élément!

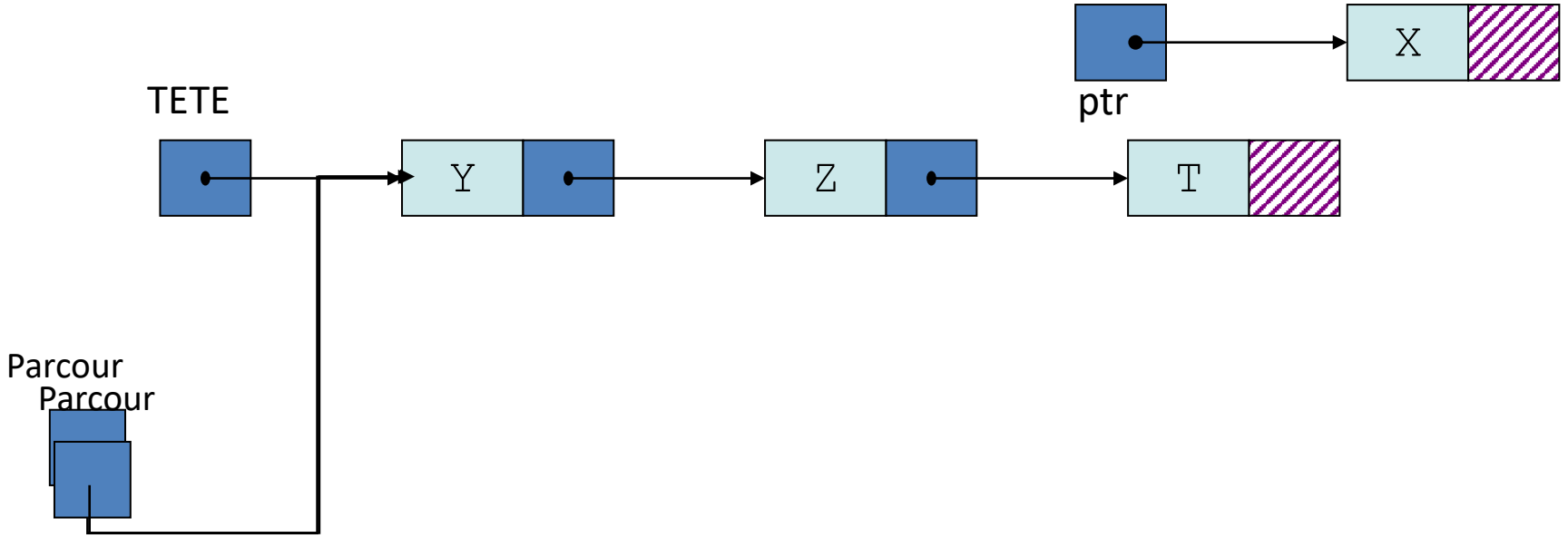
Pour cela on dispose uniquement de celle du premier élément qui est dans TETE

Il faut donc parcourir la liste du début jusqu'à obtention de l'adresse du dernier élément

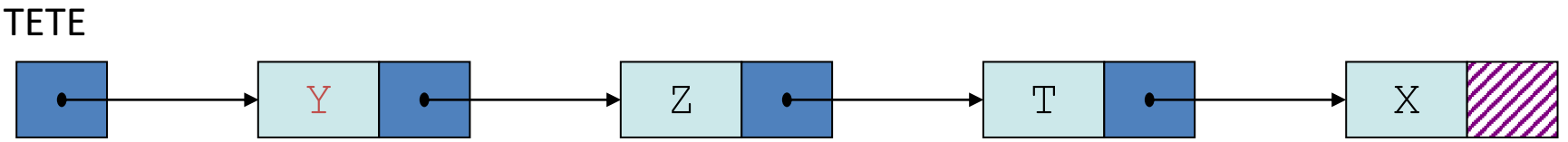


Evolution qualitative (le chaînage)

2- Déterminer l'adresse du dernier élément

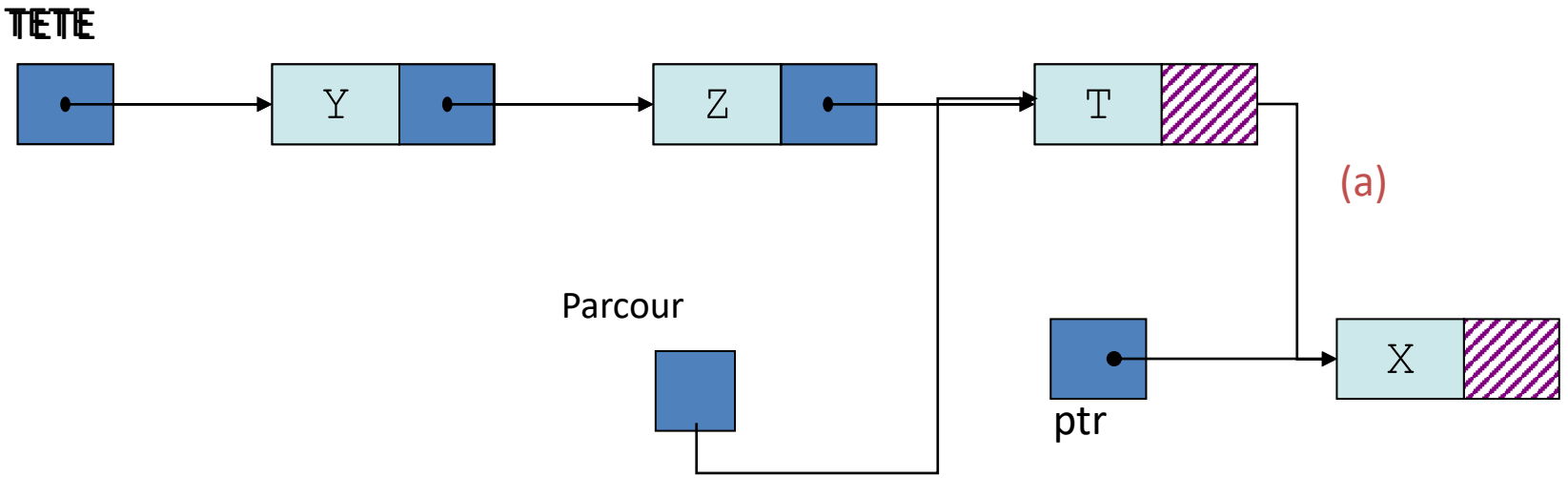


Parcour <- TETE ;
Faire
{Parcour<- Parcour ->PTR;}
Jusqu'à :
(Parcour->PTR= NULL)

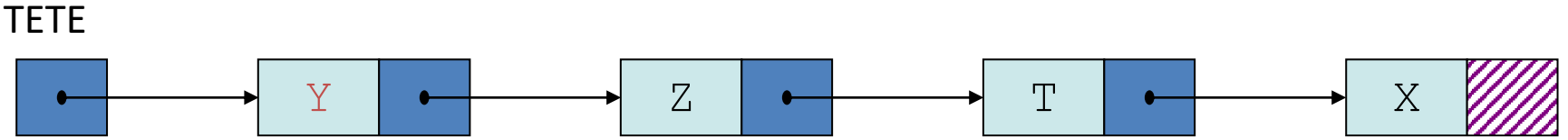


Evolution qualitative (le chaînage)

2- écriture algorithmique de chaque opération



(a) ⟶ Parcour → PTR ← ptr ;



Récapitulation étude qualitative

Parcour ← TETE ;

Faire

{Parcour ← Parcour → PTR;}

Jusqu'à :

(Parcour → PTR= NULL)

Parcour → PTR ← ptr

Rédaction de l'algorithme:

Début

Variables de travail:

Ptr, Parcour de type pointeur sur ELEMENT

Étude quantitative:

ptr ← Obtenir (ELEMENT) ;

ptr → INFO ← X ; ptr → PTR ← NULL;

Étude qualitative:

Parcour ← TETE ;
tq (Parcour → PTR!= NULL)
{Parcour ← Parcour → PTR;}
Parcour → PTR ← ptr ;

fin

Etude des Cas Limites (C.L.)

- Énumération des cas limites.
- Traitement des C.L. un à un.
- Intégration des C.L. dans l’algorithme général

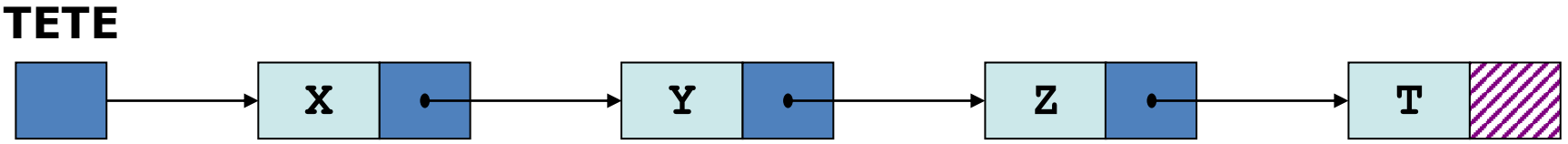
Soustraire un élément à la L.C.

- Diminuer structure d'un nouvel élément a plus d'une variante :
  - Soustraire en tête de la L.C.
  - Soustraire au milieu de la structure
  - Soustraire l'élément en queue

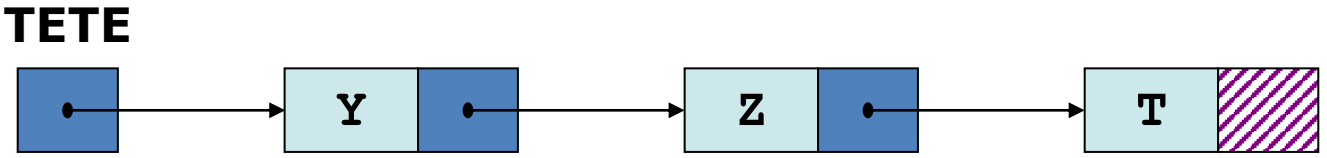
Soustraire un élément en tête

1- ‘cas fréquemment rencontré’

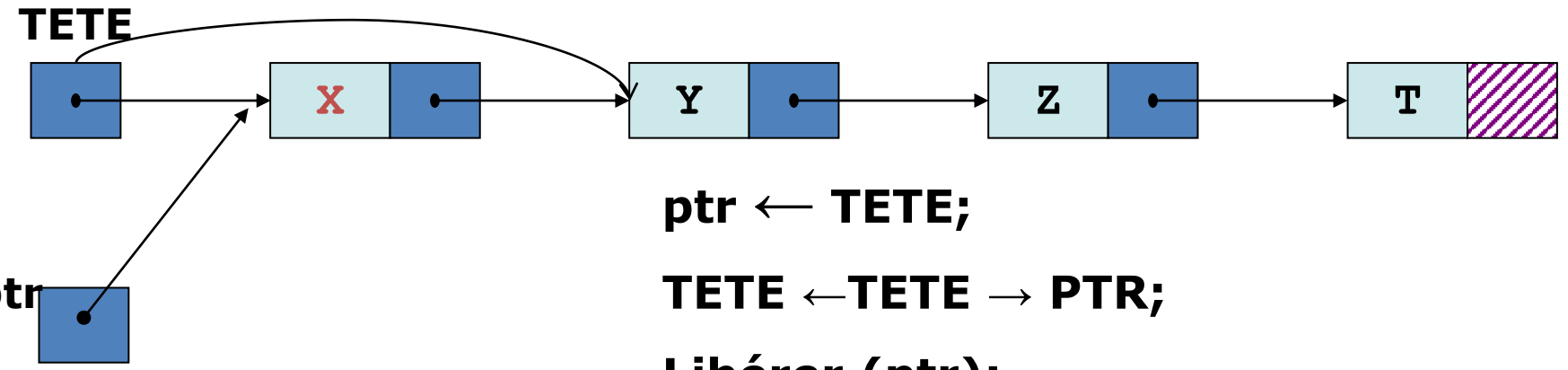
Etat avant suppression



Etat après suppression



Évolution Quantitative/qualitative



ptr ⟵ TETE;

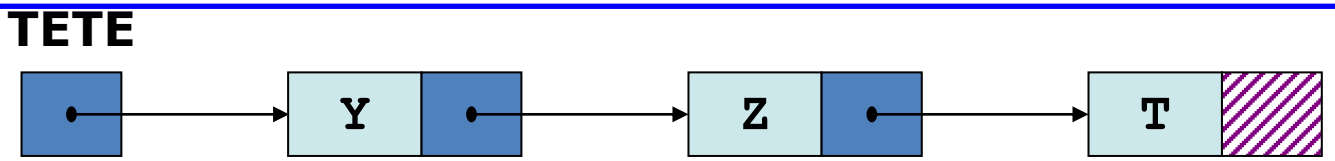
TETE ⟵TETE ⟶ PTR;

Libérer (ptr);

- **1 élément en moins,**
- **il contient l'adresse du suivant Il faut :**

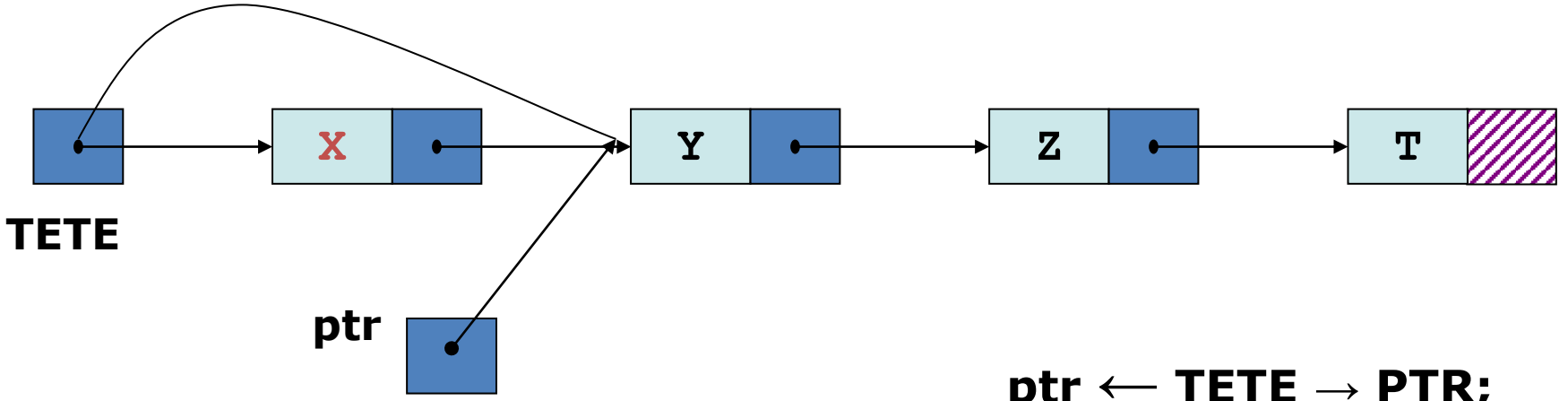
Restituer la mémoire qu'il occupe au SYSTEME .

préserver l'adresse du suivant et la placer dans TETE.



Deuxième possibilité

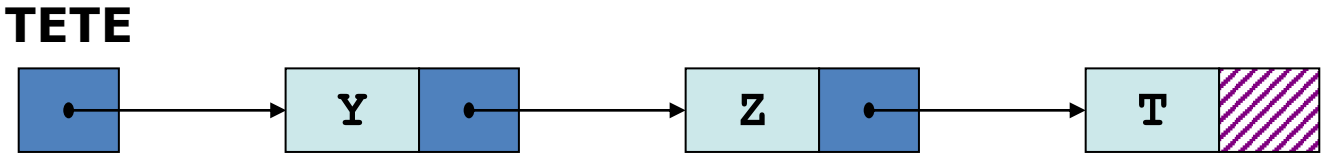
1-partant de la structure initiale, construire la nouvelle



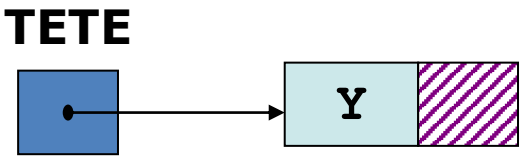
ptr ← TETE → PTR;

Libérer (TETE);

TETE ← ptr ;



Liste chainé contenant initialement un seul élément



Libérer (TETE) ;

TETE ← NULL ;

Intégration des C.L. dans l'algorithme général

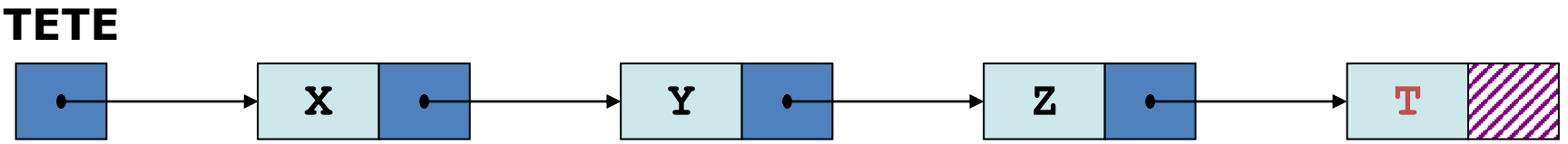
- L'algorithme général écrit pour traiter le cas « le plus fréquent » prend en compte tous les C.L. :

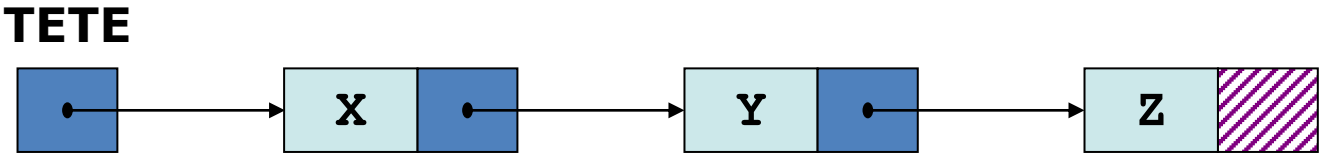
- Il constitue donc l'algorithme global.

- Reste à traiter les exceptions.

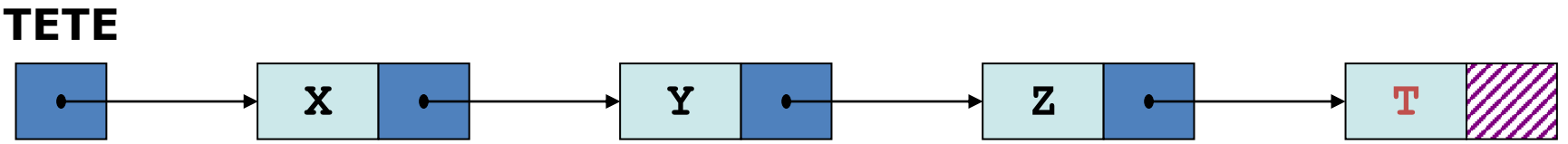
Soustraire un élément en queue

1- ‘cas fréquemment rencontré’





Évolution Quantitative/qualitative



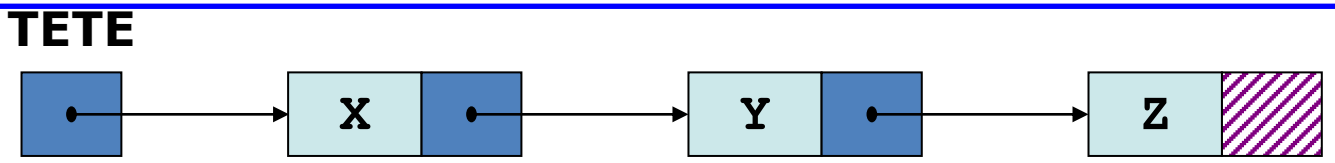
- **1 élément en moins,**
- **il contient la marque NULL de fin de L.C.**

Il faut :

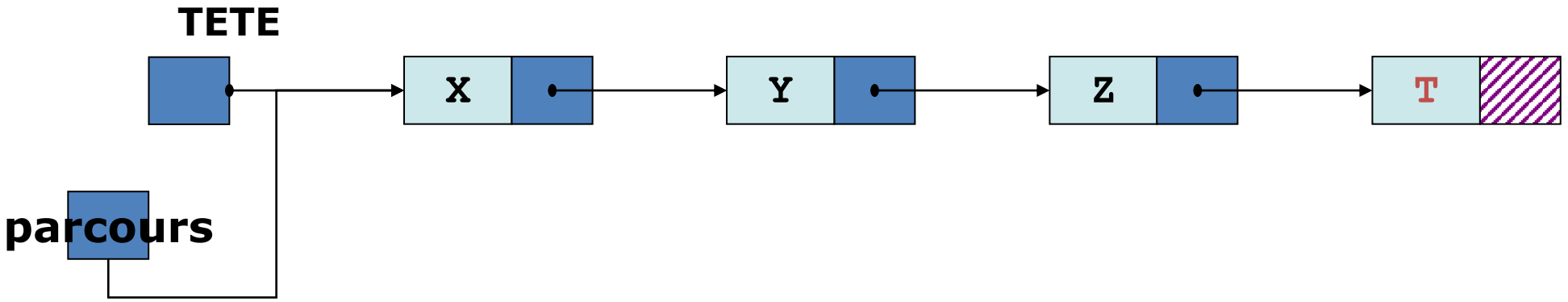
Restituer la mémoire qu'il occupe au SYSTEME

marquer la fin de la liste en mettant NULL dans :

l'avant dernier élément !



Évolution Quantitative/qualitative



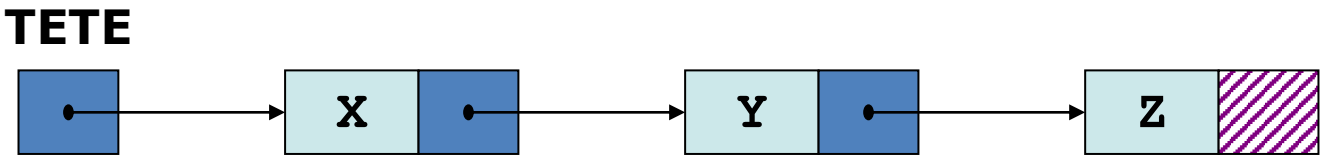
Parcours <- TETE

Tant que (Parcours ->PTR ->PTR#NULL)

Parcours <- Parcours ->PTR

Libérer(Parcours ->PTR)

Parcours ->PTR = NULL



Représentation des piles et files par listes chainées

File

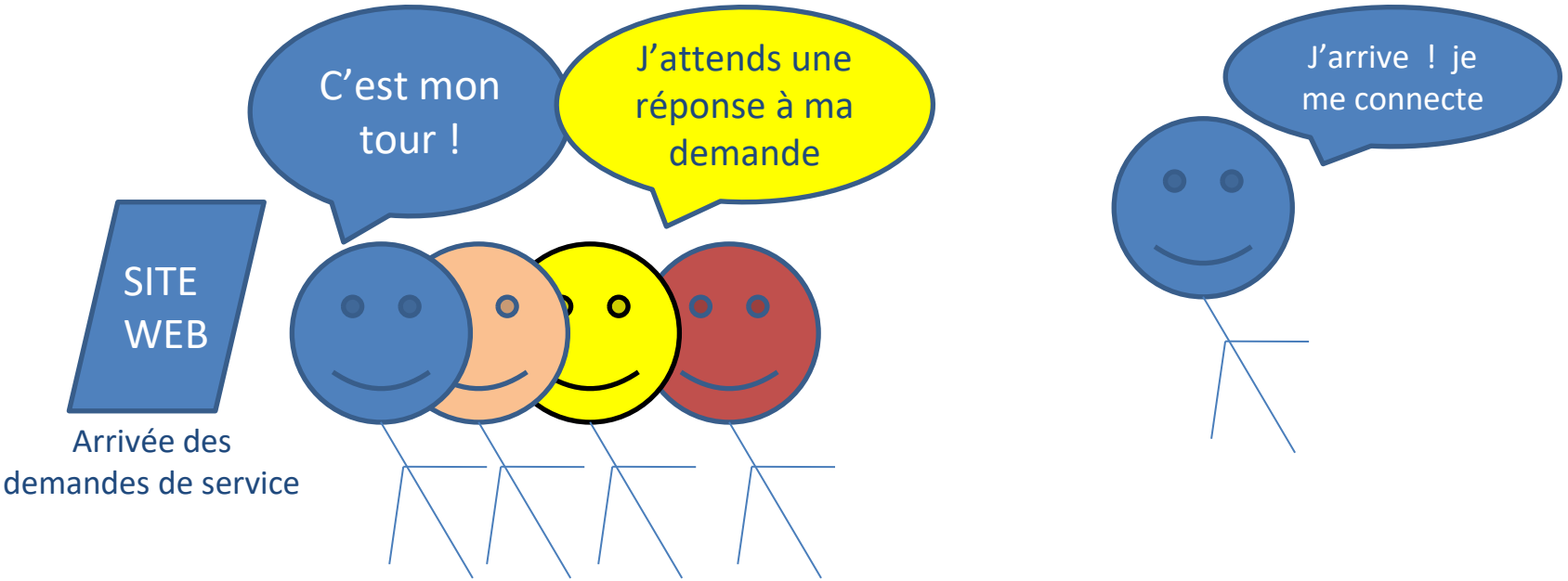
- Réalisation d'une file à l'aide d'une liste chainée

- Chaque élément de la file sera représenté par un élément de la liste chaînée.
- Nous avons deux possibilités :
  - 1ère représentation: l'élément pointe sur le suivant
  - 2ème représentation: l'élément pointe sur le précédent

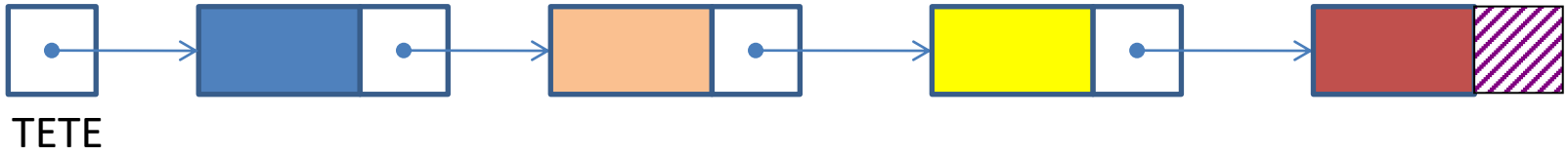
Deux situation se dessinent : TETE pt d'entrée et QUEUE pt de sortie, ou bien TETE pt de sortie QUEUE pt d'entrée

1ère Représentation de la file

- Chaque élément pointe sur le suivant



- Chaque élément pointe sur le suivant :

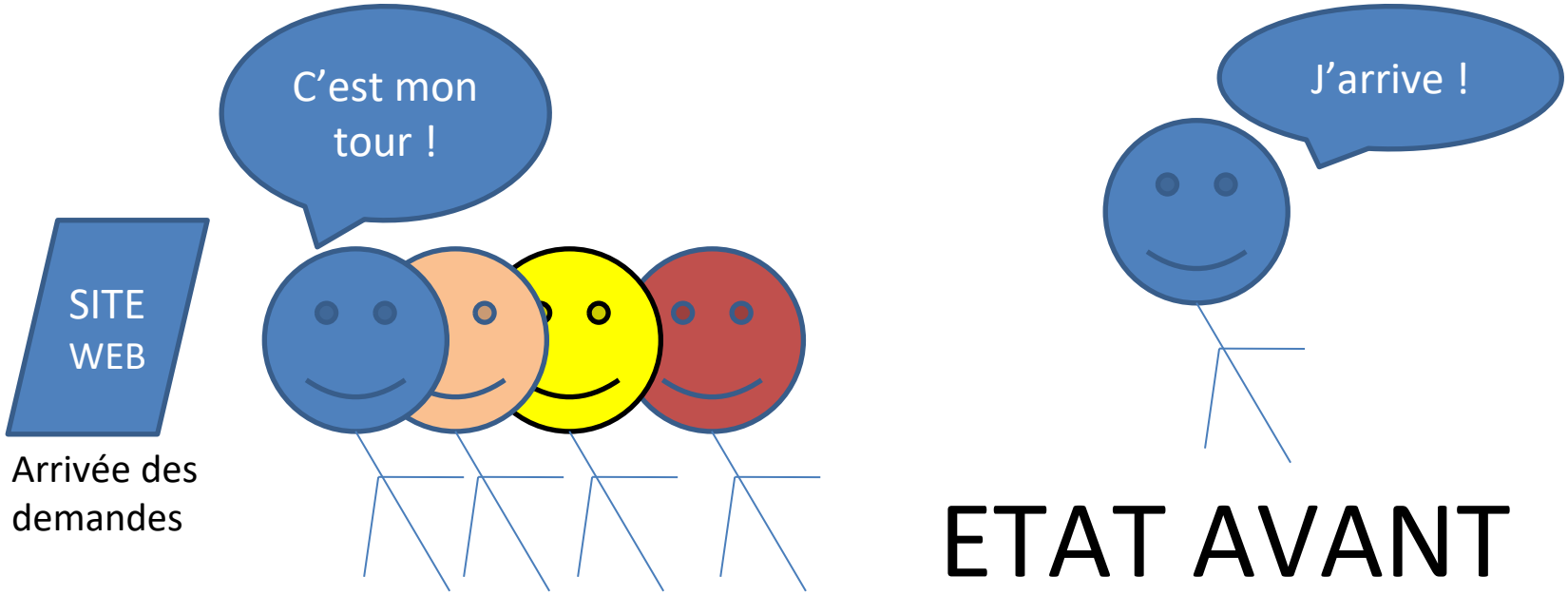


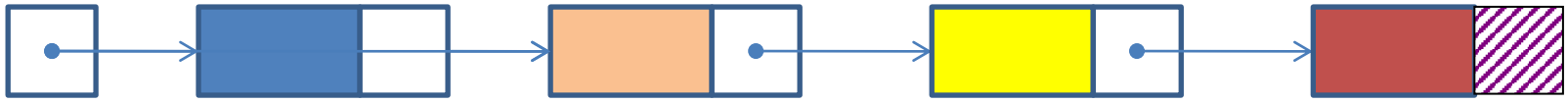
- TETE pointe sur le premier élément de la file !

La partie pointeur de l'élément le plus ancien pointe sur son suivant et ainsi de suite.
Le premier élément de la file correspond ici au premier élément de la liste chainée.

1ère Représentation de la file

- Evolution FIFO: Diminution

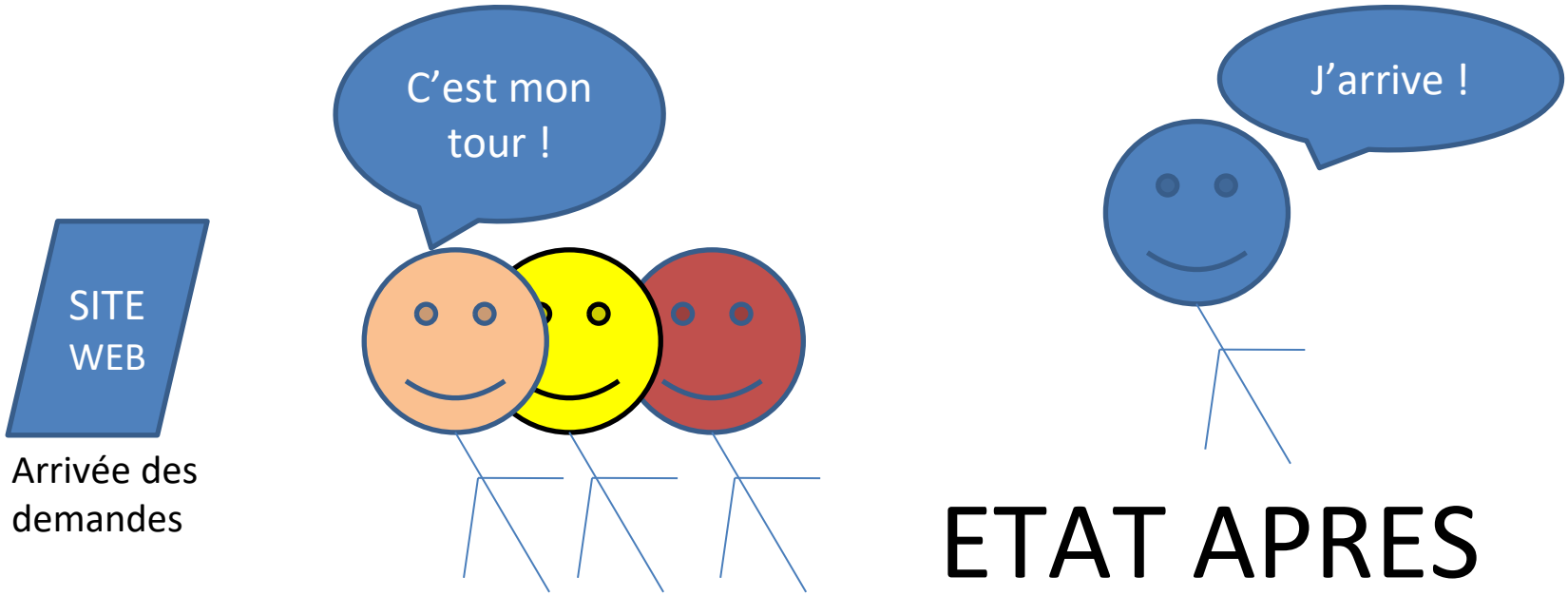


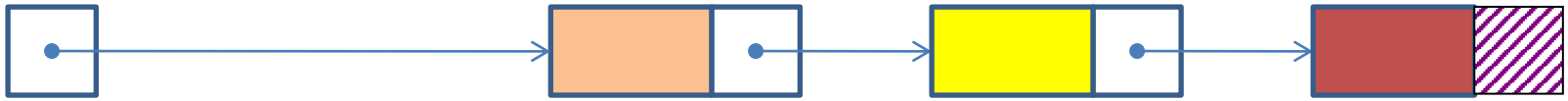


?

1ère Représentation de la file

- Evolution FIFO: Diminution



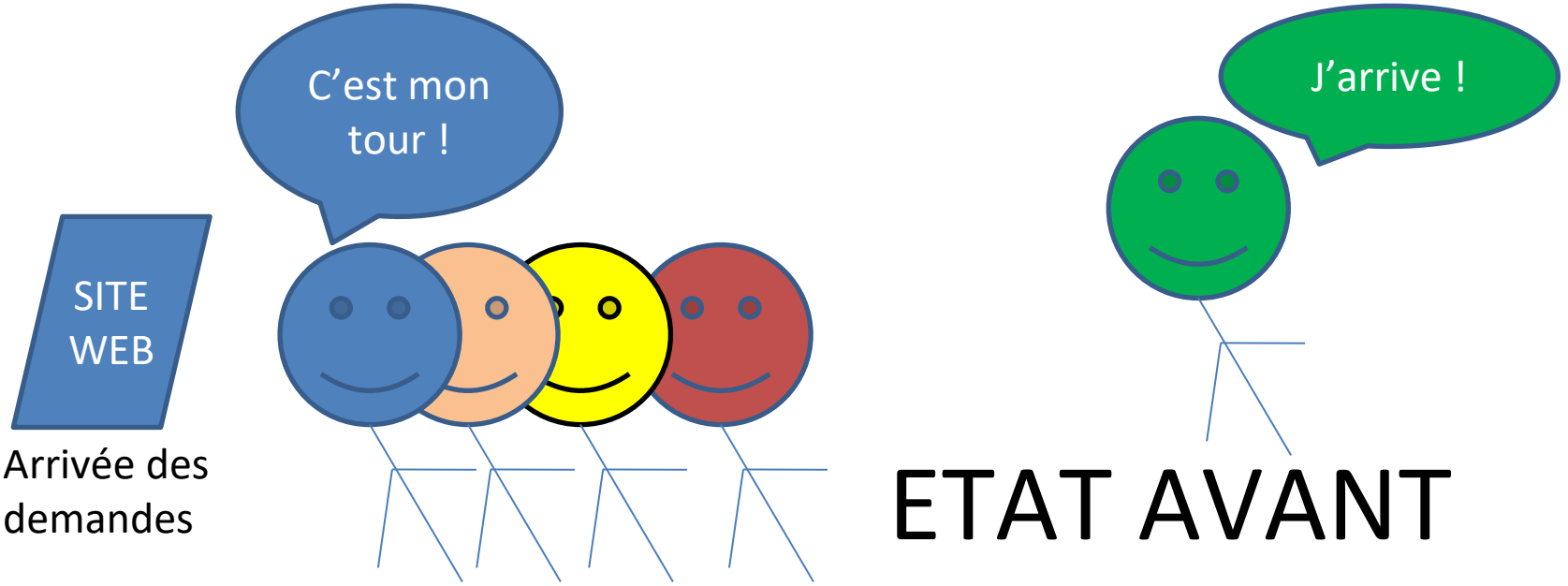


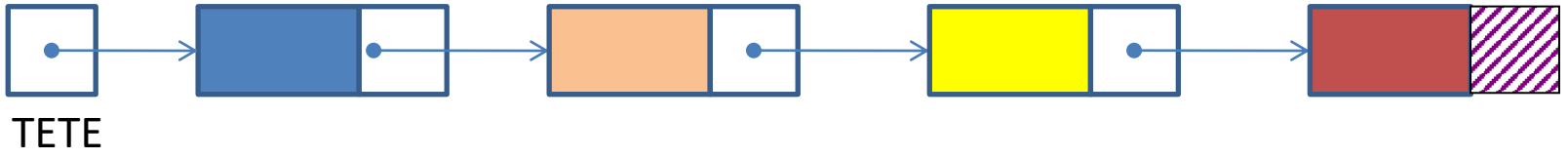
Diminution selon FIFO

suppression en tête

1ère Représentation de la file

- Evolution FIFO: Augmentation

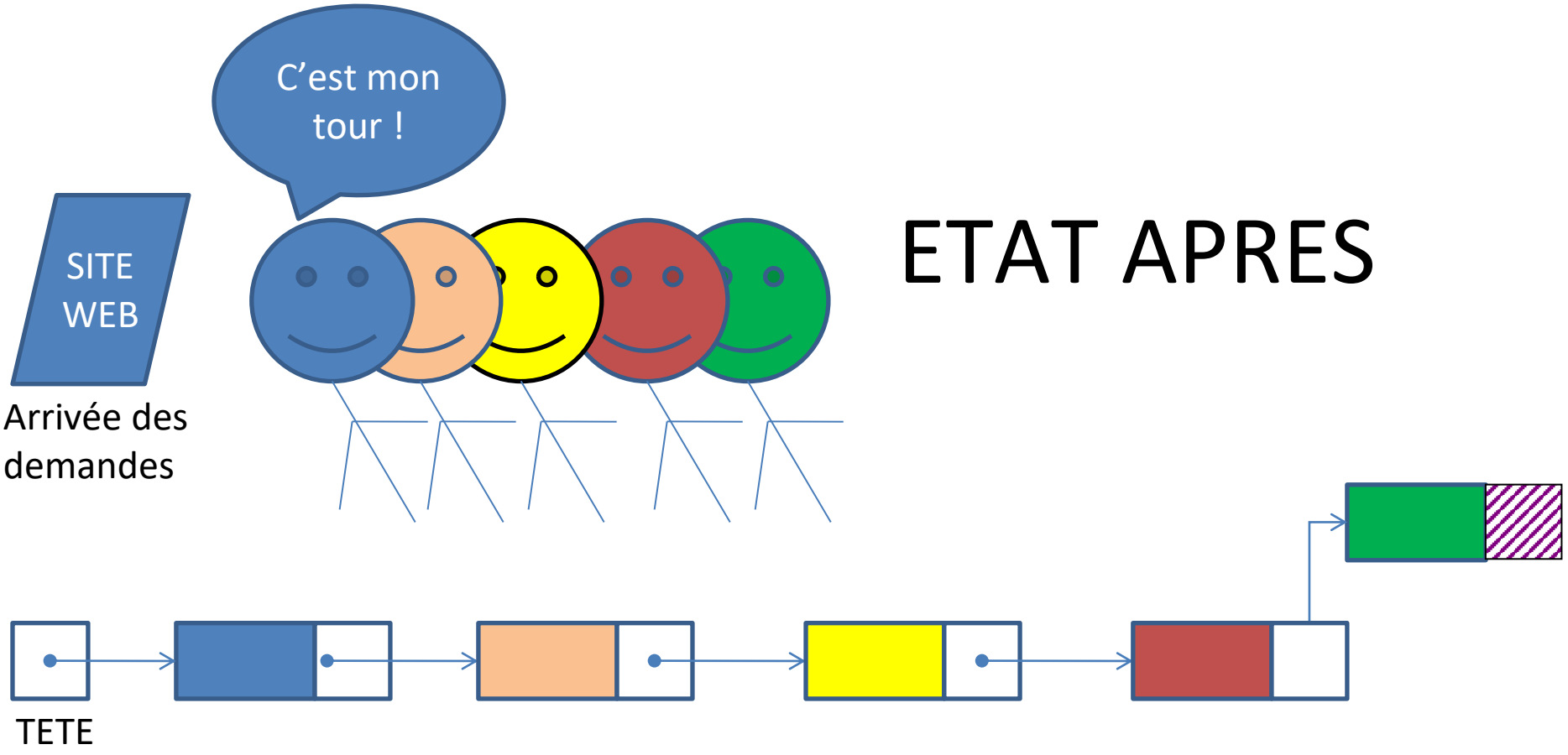




Augmentation selon FIFO  ?

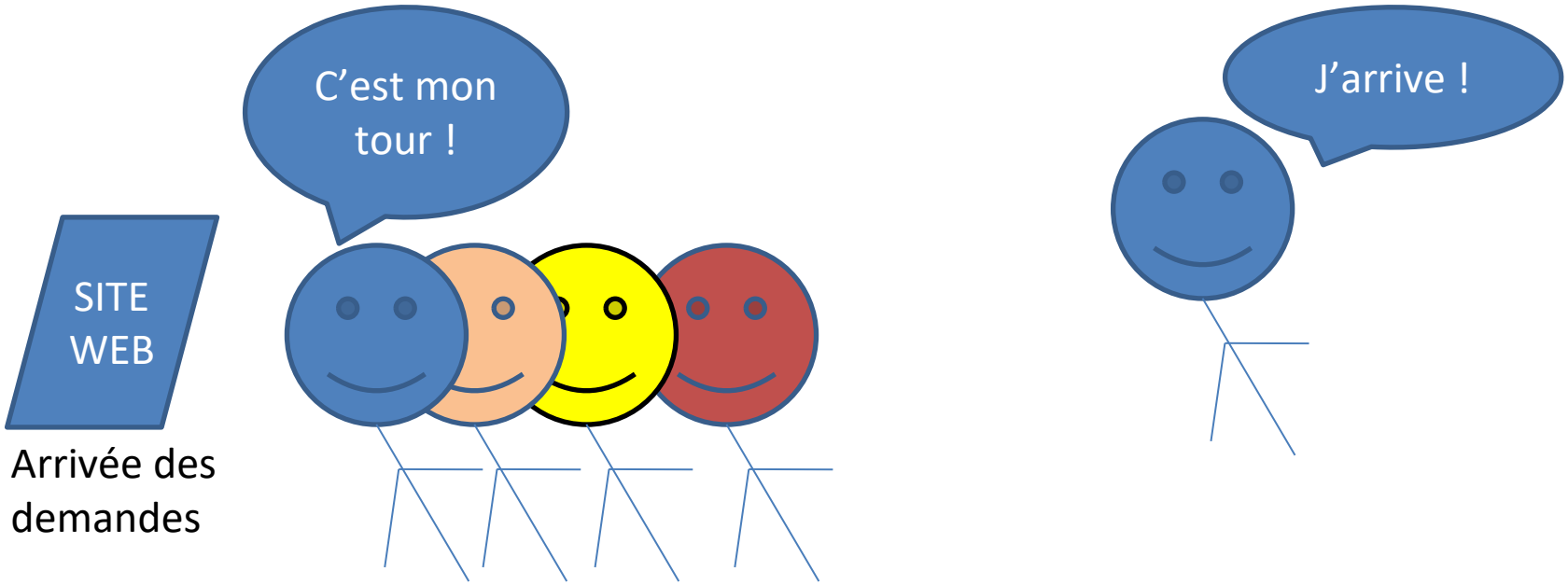
1ère Représentation de la file

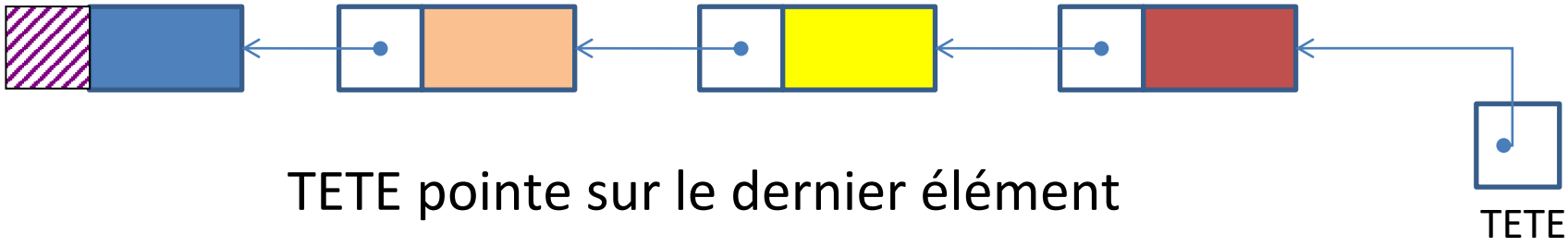
- Evolution FIFO: Augmentation



2ème Représentation de la file

- L'élément pointe sur le précédent

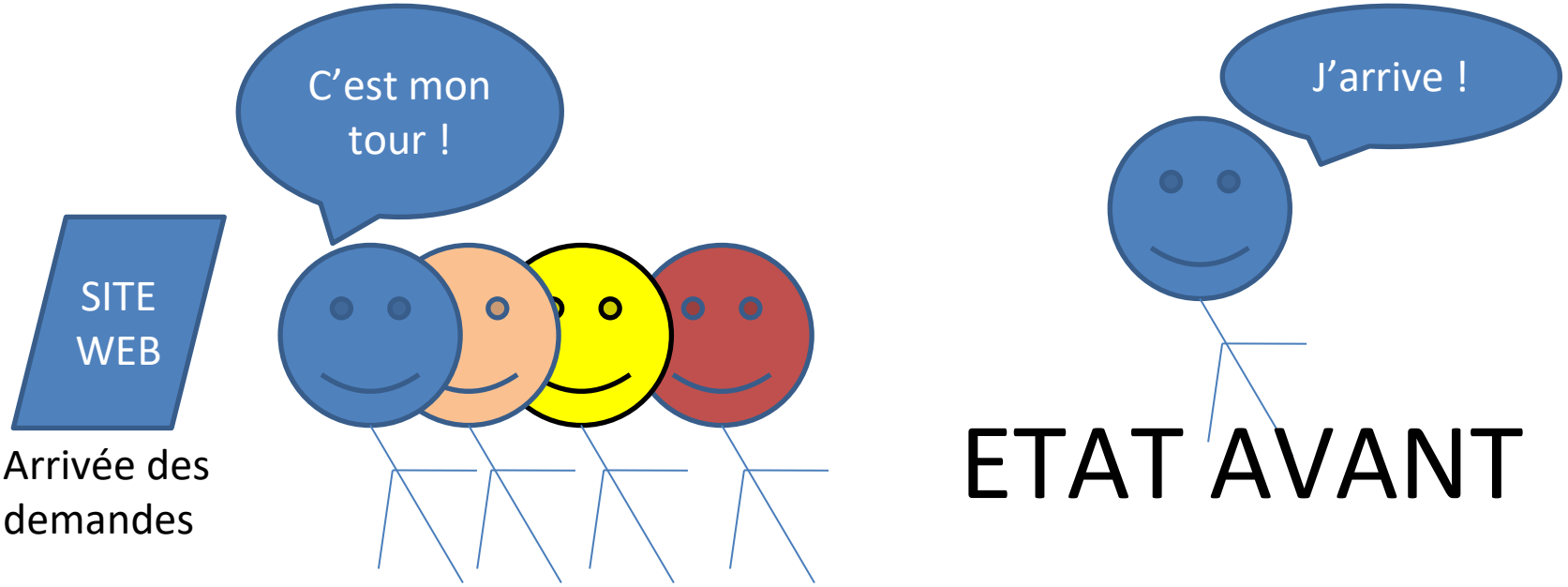


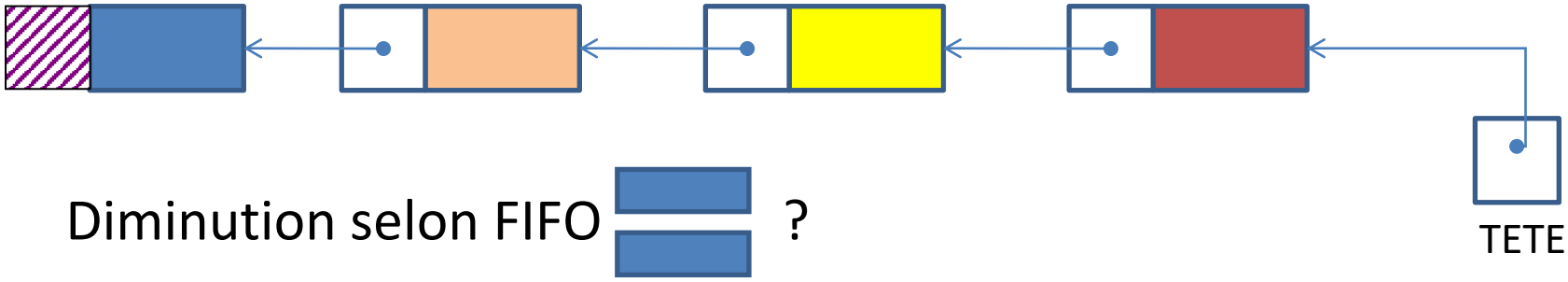


TETE pointe sur le dernier élément

2ème Représentation de la file

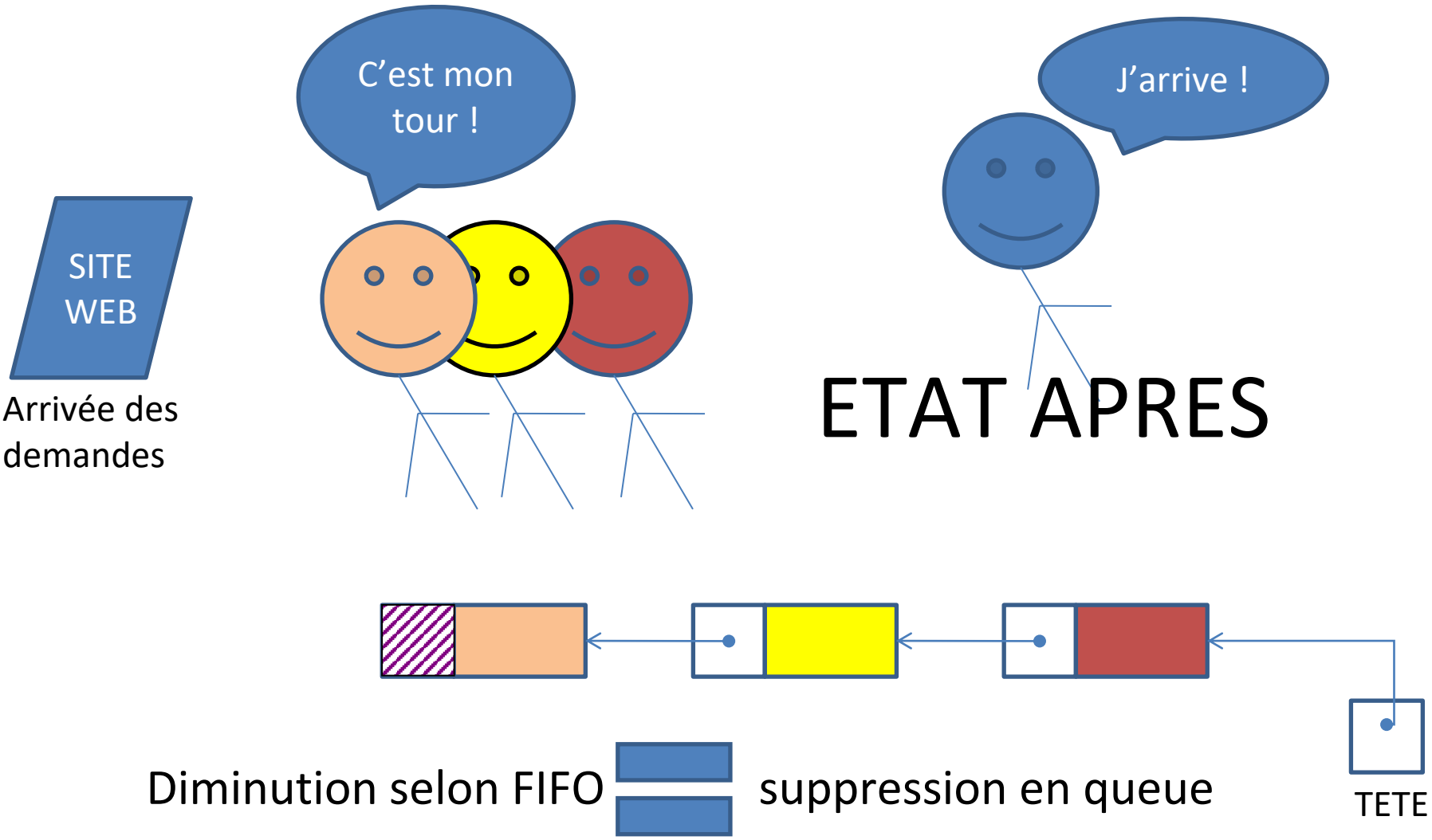
- Evolution de la FIFO: Diminution





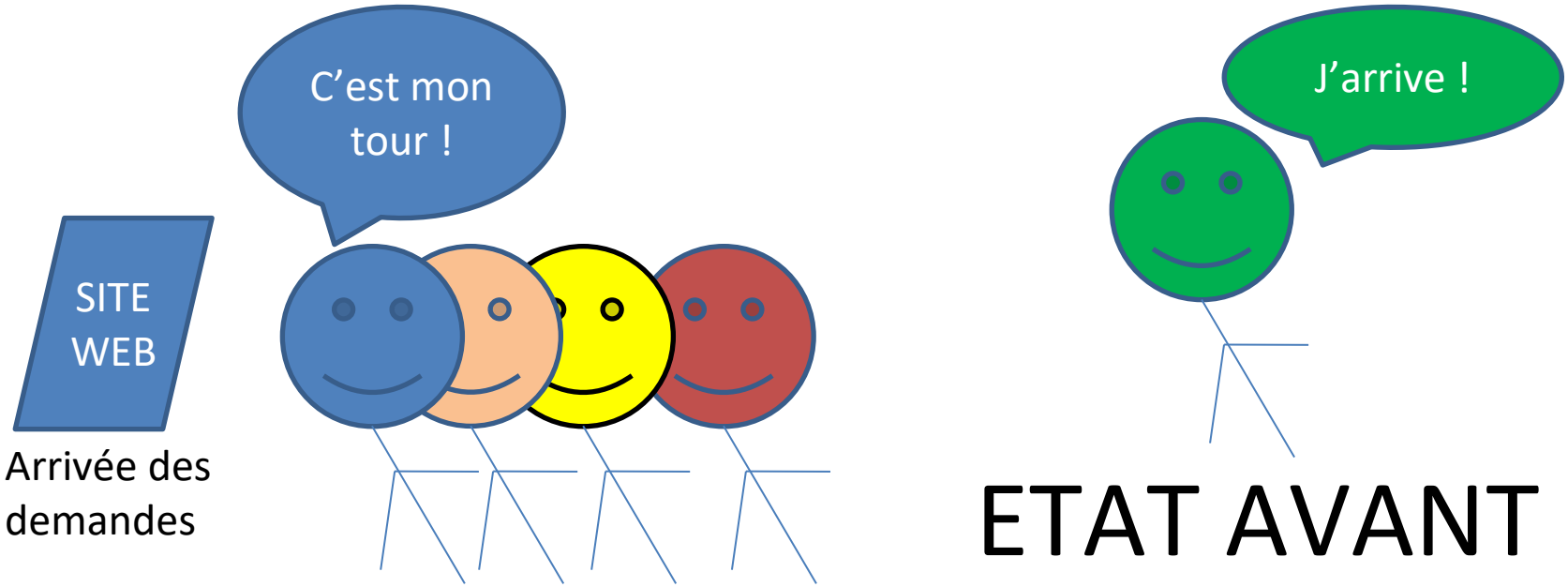
2ème Représentation de la file

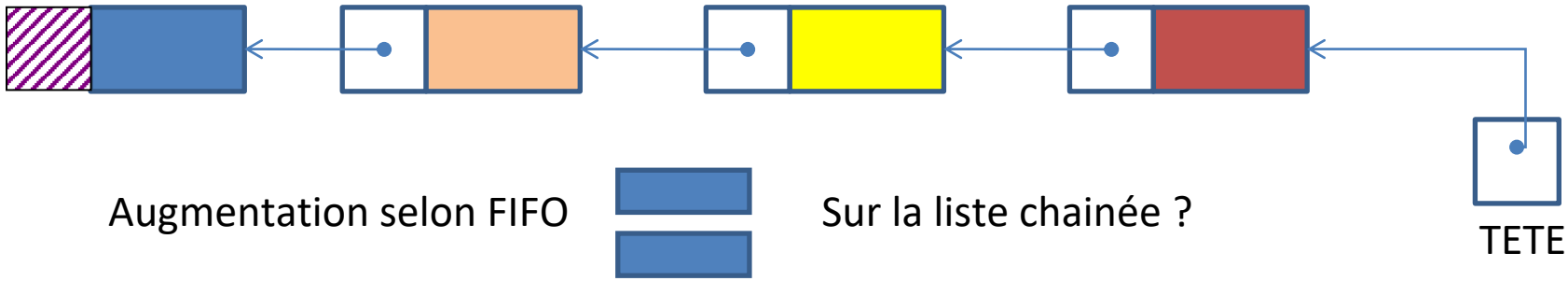
- Evolution de la FIFO: Diminution



2ème Représentation de la file

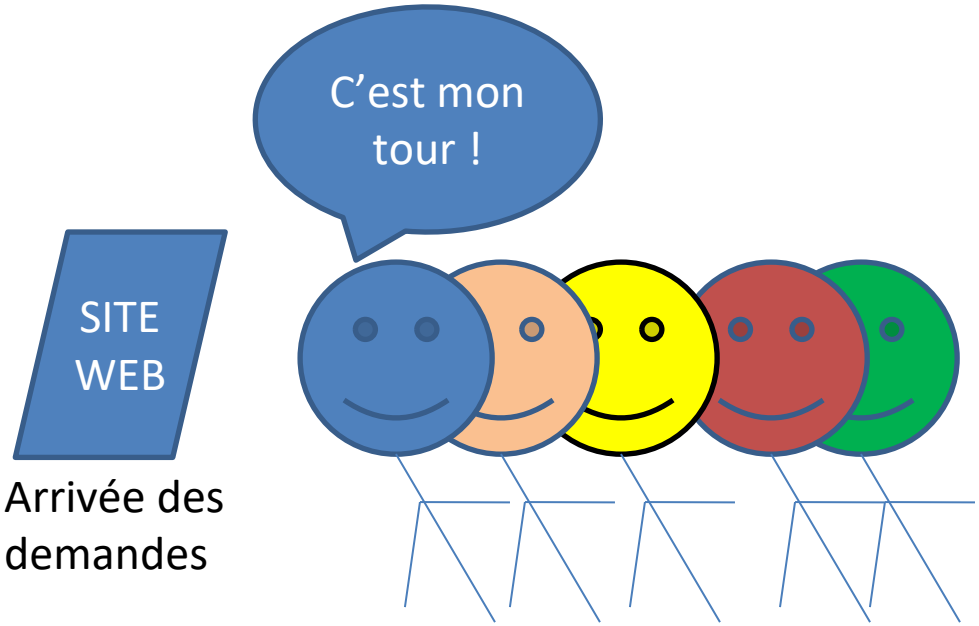
- Evolution de la FIFO: Augmentation



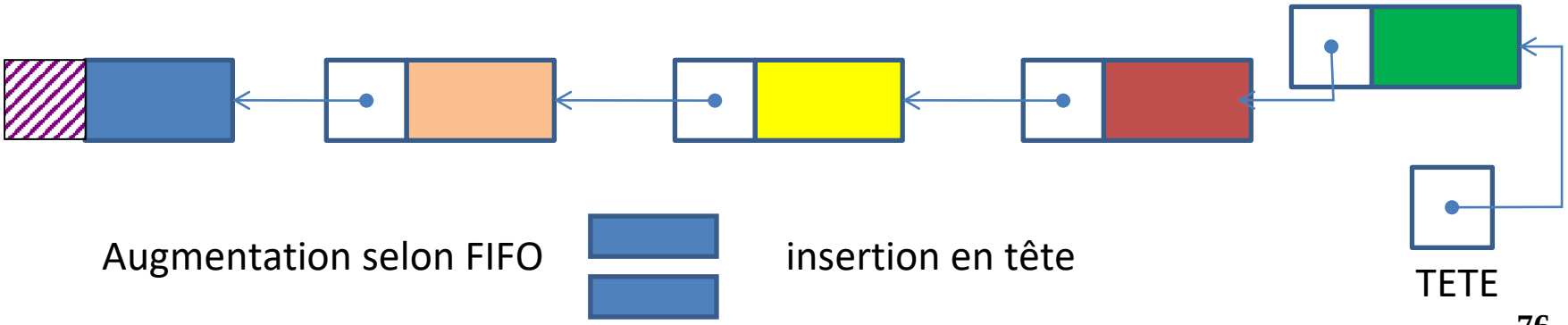


2ème Représentation de la file

- Evolution de la FIFO: Augmentation



ETAT APRES



File

- Récapitulation et Comparaison

- Pointage sur le suivant

- TETE pointe sur le premier élément
- Diminution = suppression en tête

  3 opérations (pas de parcours)
- Augmentation = insertion en queue

  n +3 : En utilisant un pointage TETE

  3 avec 2 pointages TETE et QUEUE

- Pointage sur le précédent

- TETE pointe sur le dernier élément
- Diminution = suppression en queue

  n +3 pour tout type de pointage
- Augmentation = insertion en tête

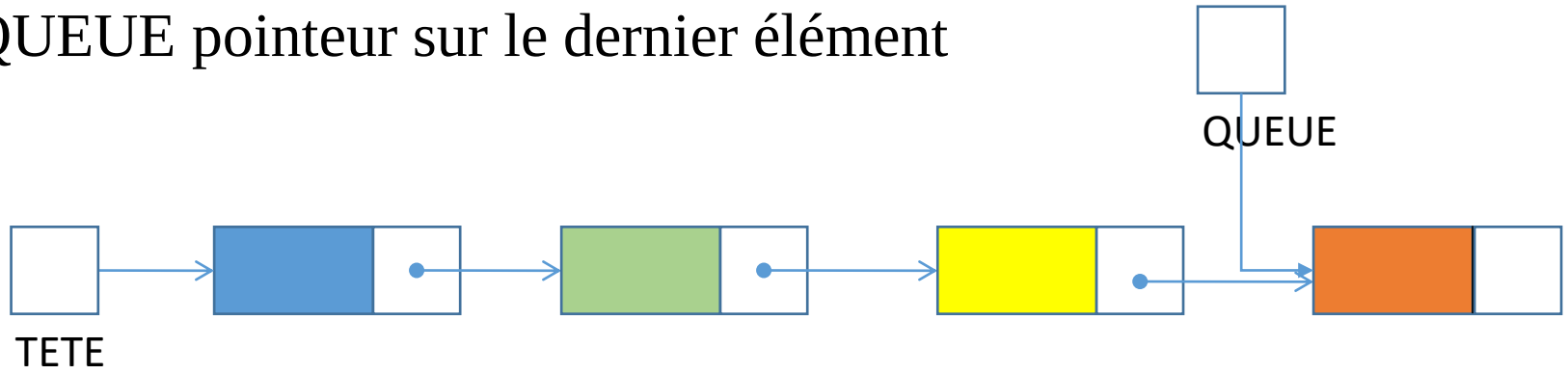
  3 opérations (pas de parcours!)

On retient : pointage sur suivant & utilisation de TETE & QUEUE

(car on peut réaliser la file sans faire de parcours)

Autres possibilités de réalisation

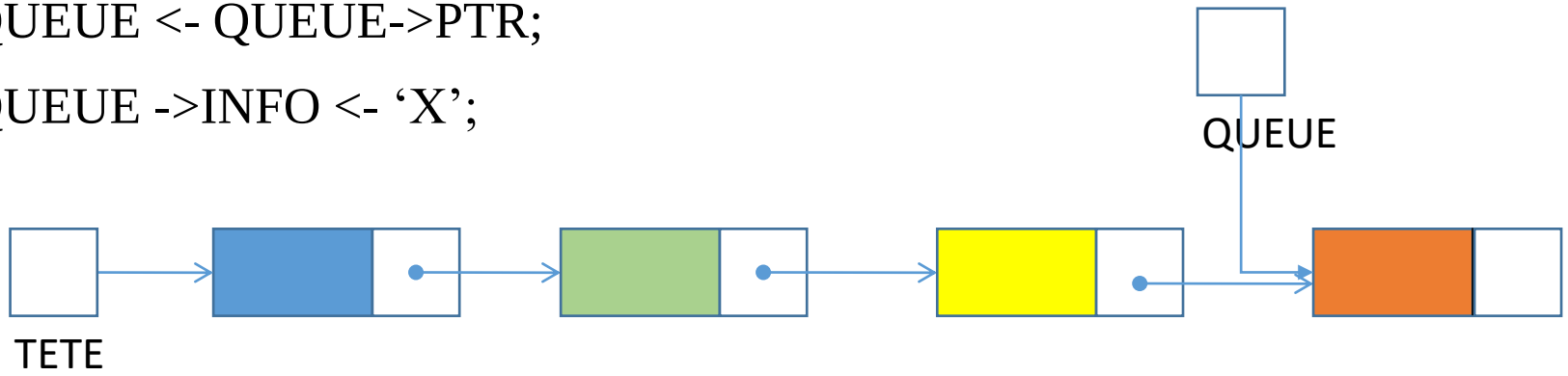
- Liste avec TETE et QUEUE
- C'est une liste chaînée pour laquelle nous utilisons deux pointages:
  - TETE pointeur sur le premier élément
  - QUEUE pointeur sur le dernier élément



- Remarques :
- Nous n'avons plus besoin de mettre NULL dans le dernier élément
- Il faut mettre à jour QUEUE lors d'un ajout ou une suppression en queue …

Autres possibilités de réalisation

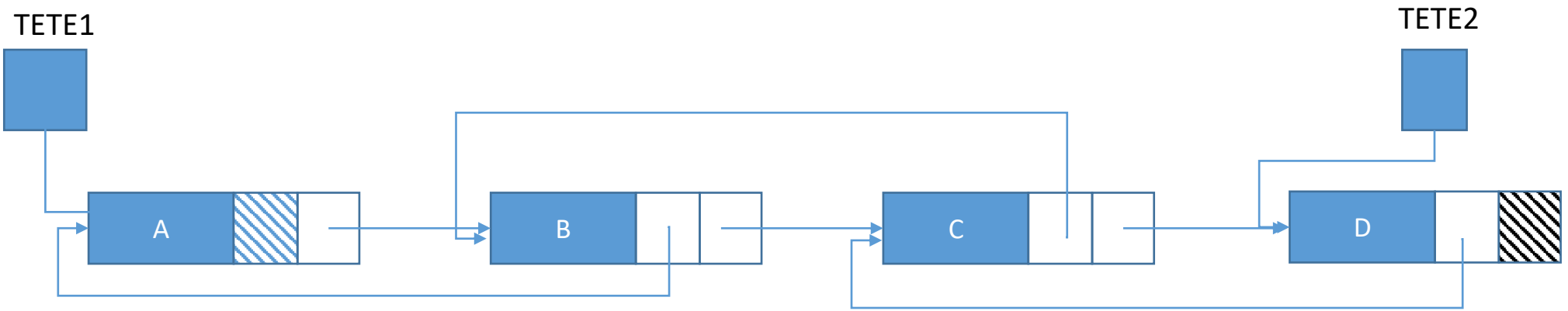
- Liste avec TETE et QUEUE
- L'insertion en queue n'exige pas de parcours:
  - QUEUE -> PTR <- obtenir (élément);
  - QUEUE <- QUEUE->PTR;
  - QUEUE ->INFO <- 'X';



- La suppression exige toujours le parcours

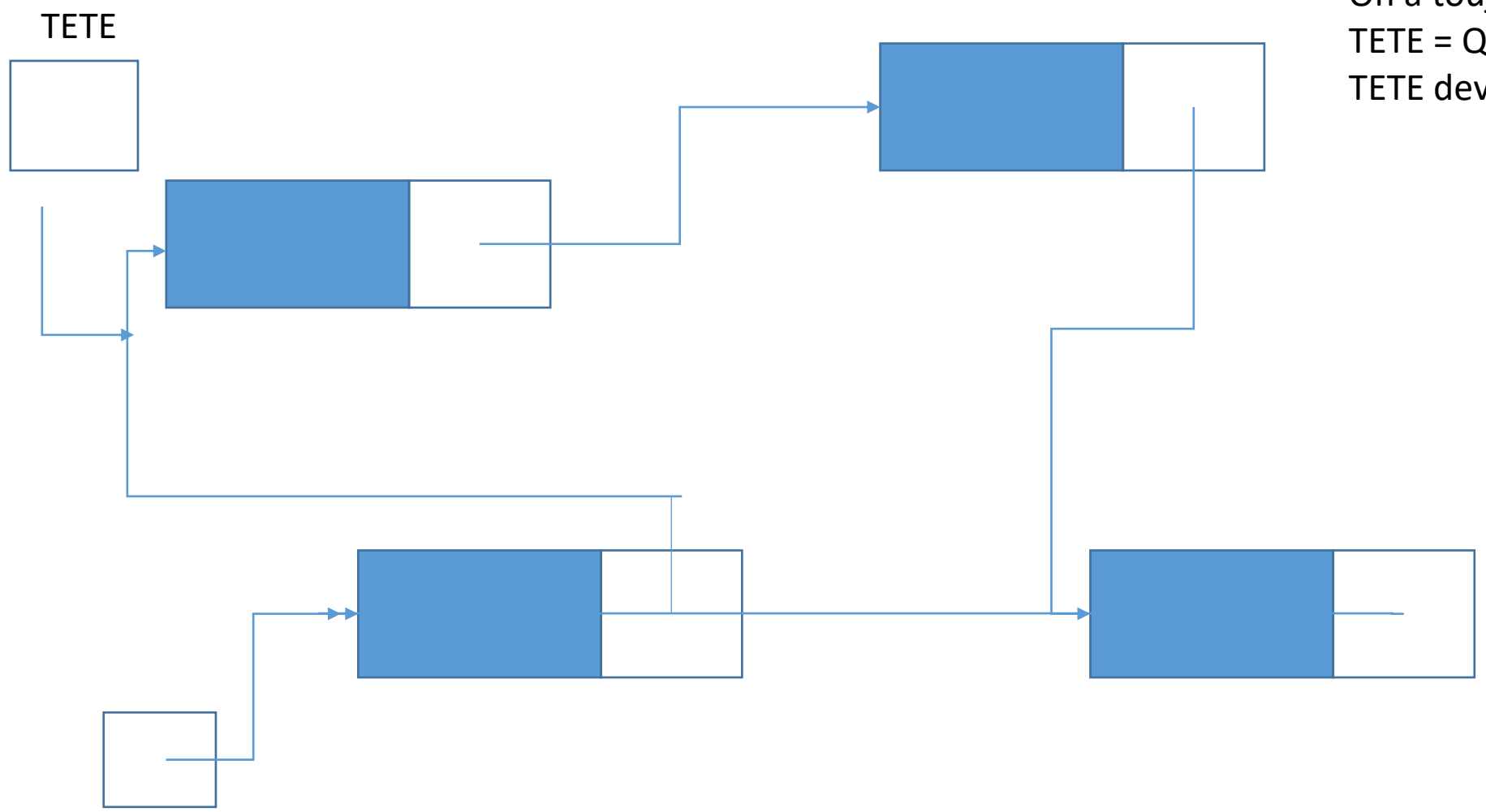
Autres possibilités de réalisation

- Liste chainée double en ascenseur



Autres possibilités de réalisation

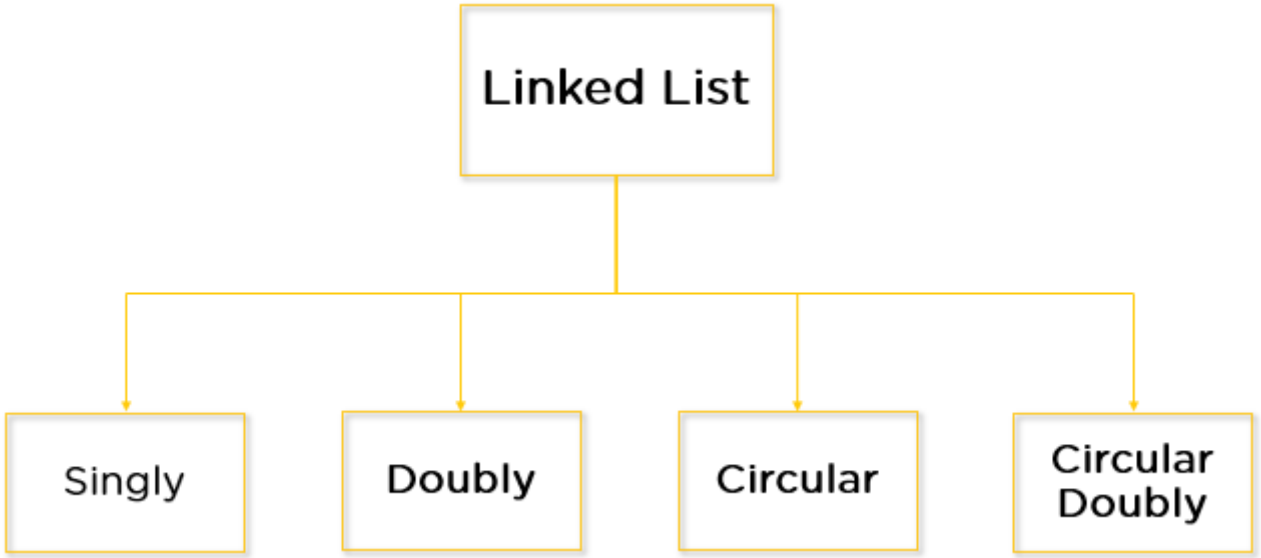
- Liste chainée circulaire



On a toujours :
TETE = QUEUE ->PTR
TETE devient inutile

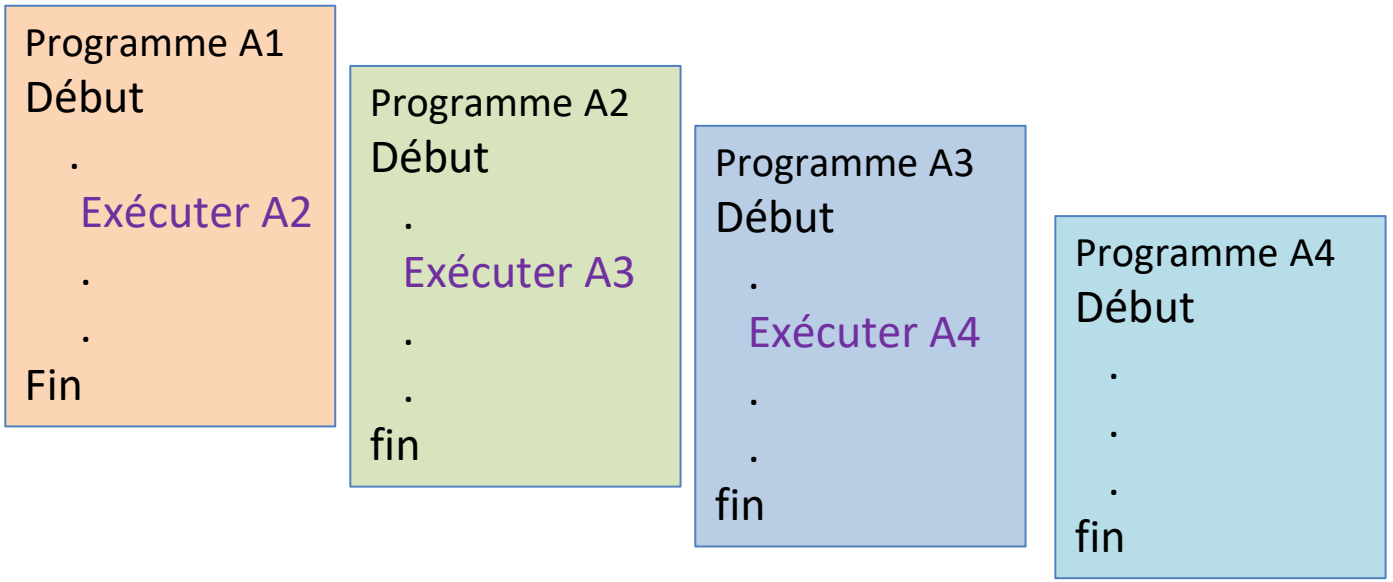
Autres possibilités de réalisation

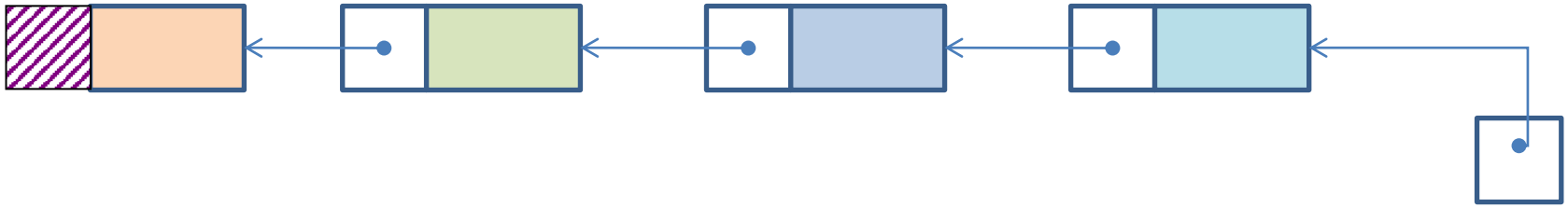
- Récapitulation et Comparaison



Pile

- Réalisation d'une Pile à l'aide d'une liste chainée

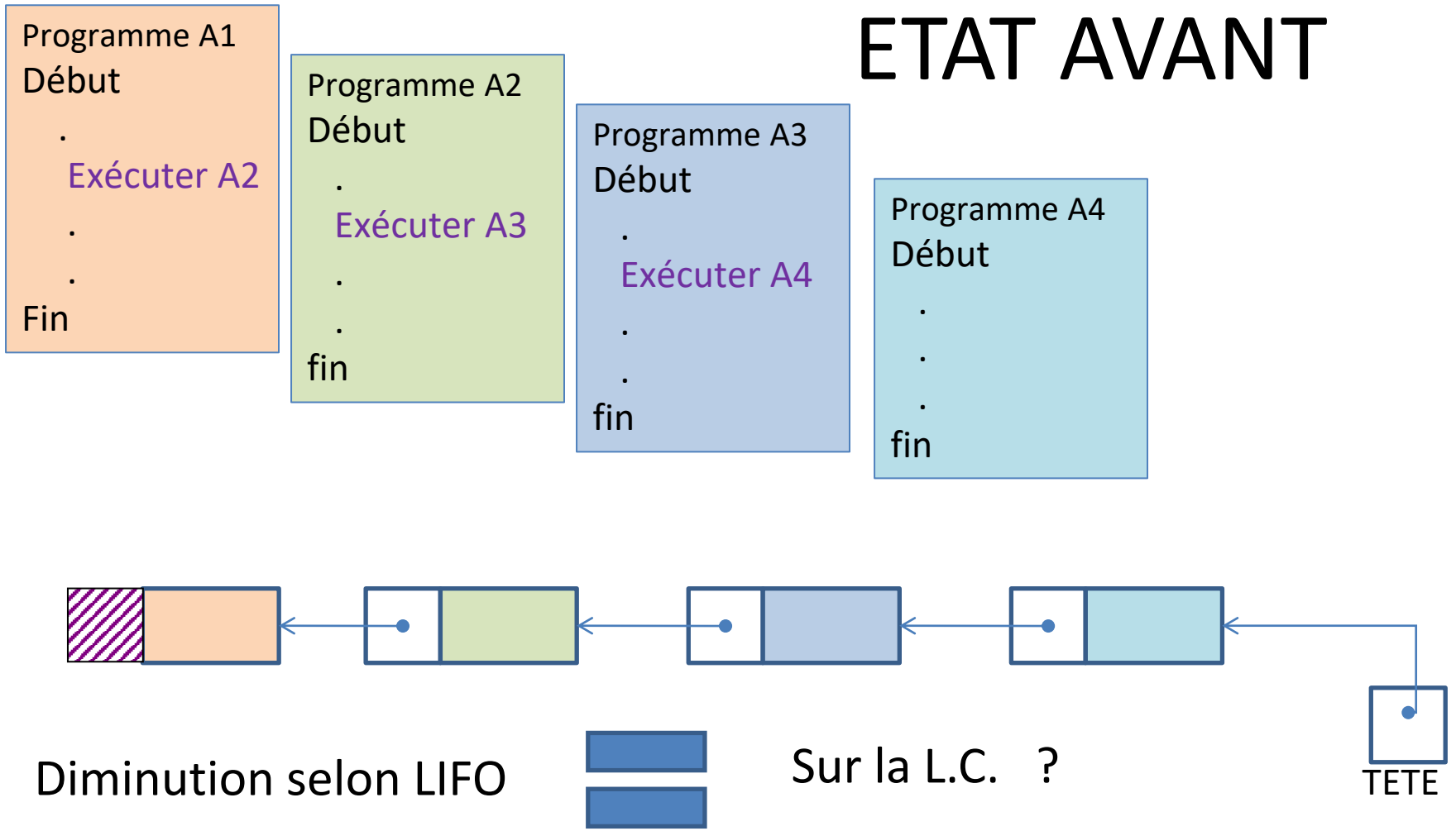




Avec pointage sur précédent :

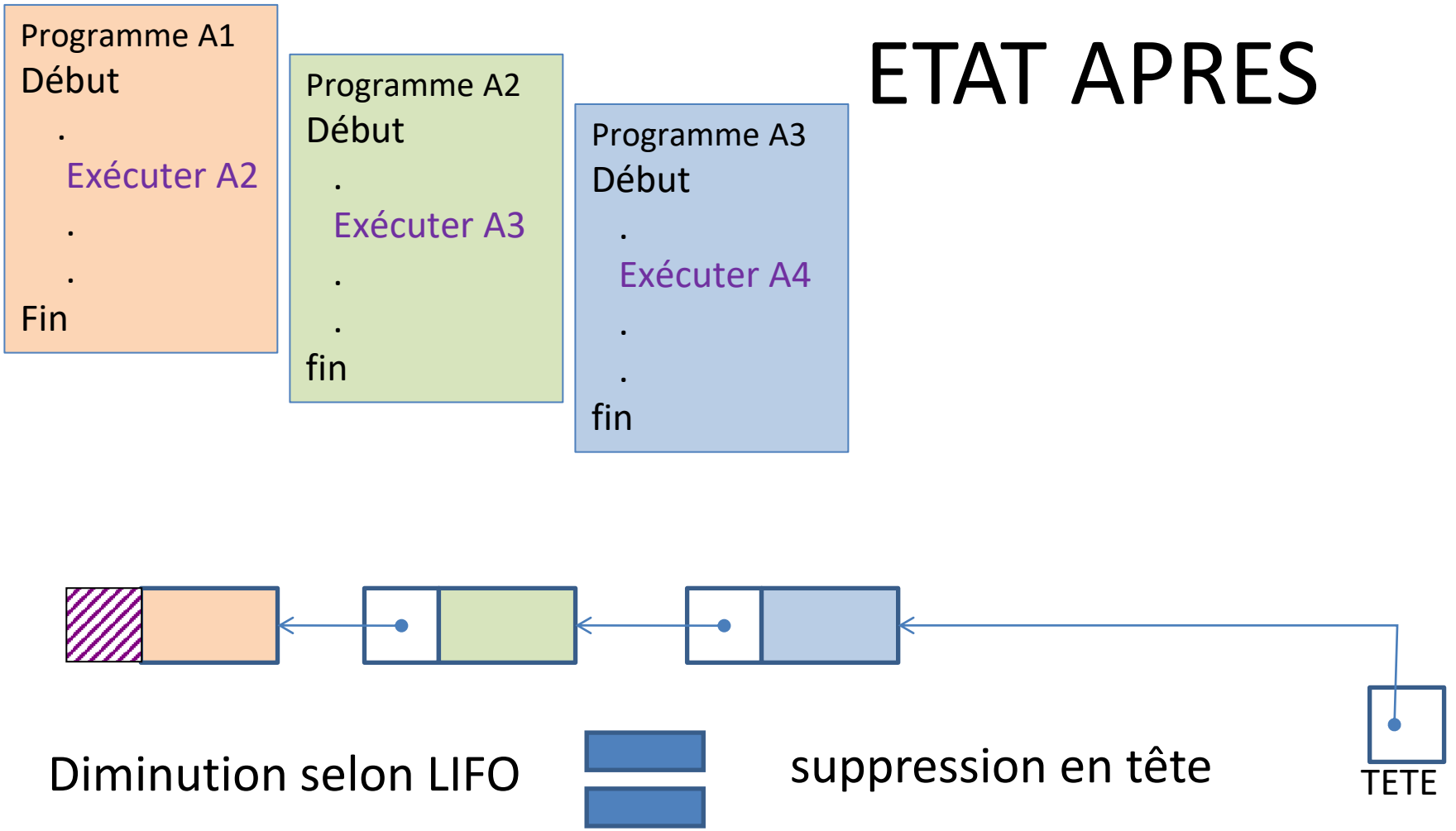
Pile

- Evolution LIFO: Diminution



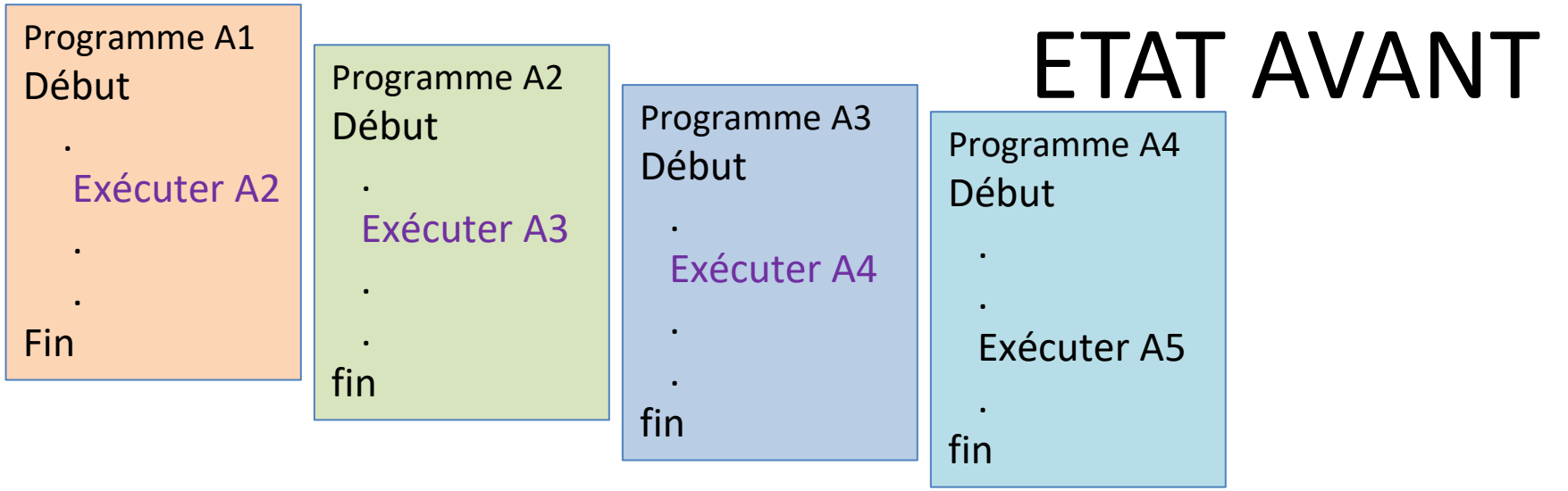
Pile

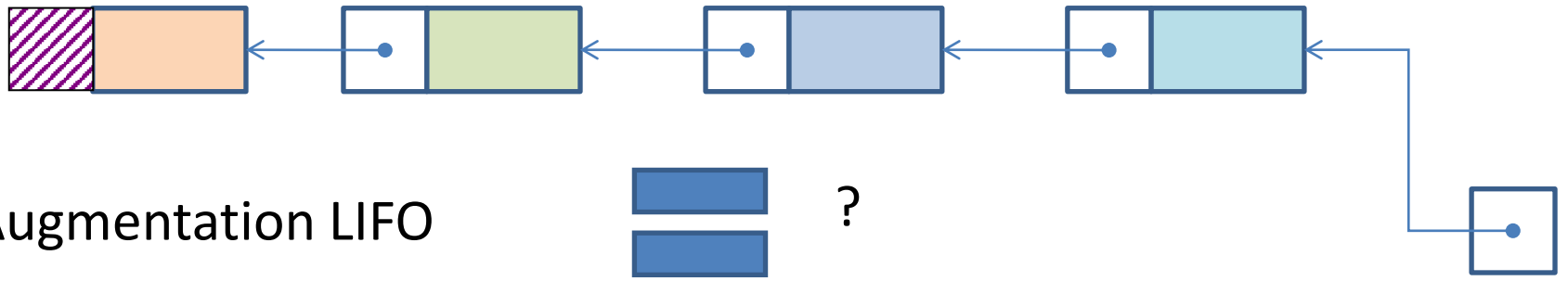
- Evolution LIFO: Diminution



Pile

- Evolution LIFO: Augmentation



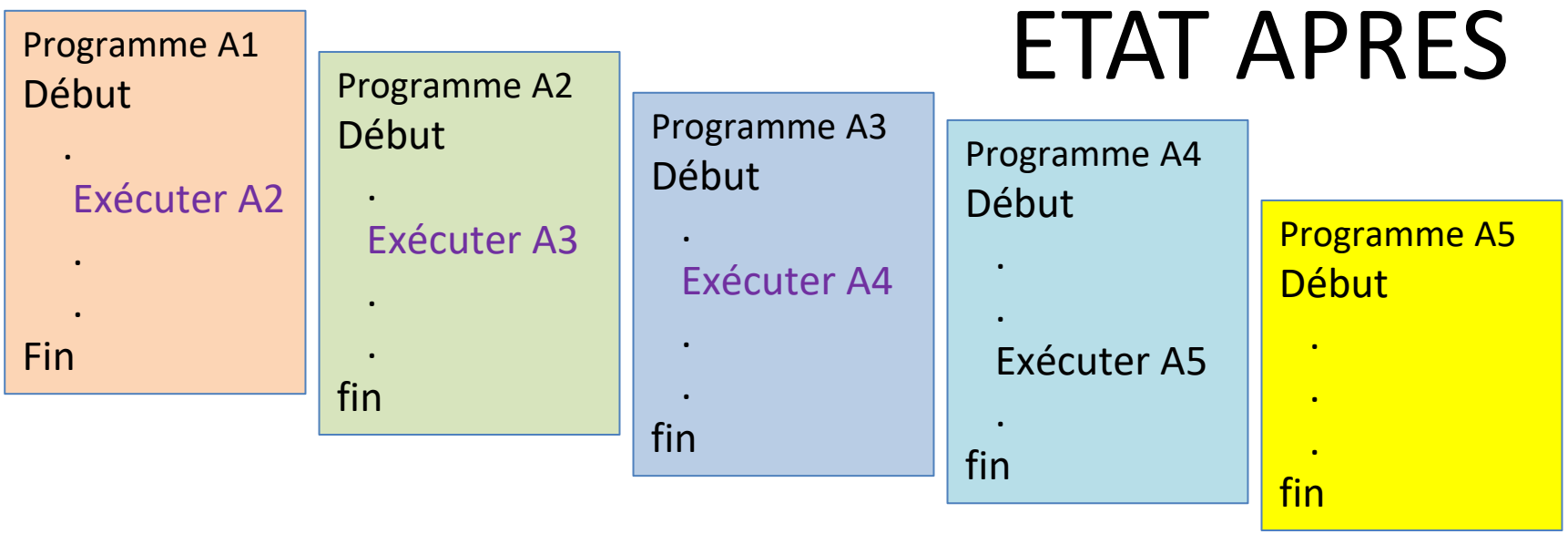


Augmentation LIFO

?

Pile

- Evolution LIFO: Augmentation



Augmentation LIFO

Insertion en tête

TETE

Pile

- Récapitulation et Comparaison

- Pointage sur le suivant

- TETE pointe sur le premier élément
- Diminution = suppression en queue

  n+3 opérations (car : parcours)
- Augmentation = insertion en queue

  n +3 : En utilisant un pointage TETE

  3 avec 2 pointages TETE et QUEUE

- Pointage sur le précédent

- TETE pointe sur le dernier élément
- Diminution = suppression en tête

  3 opérations
- Augmentation = insertion en tête

  3 opérations (pas de parcours!)

On retient : pointage sur le précédent

(car on peut réaliser la pile sans faire de parcours)

Pile et file par représentation statique

Représentation

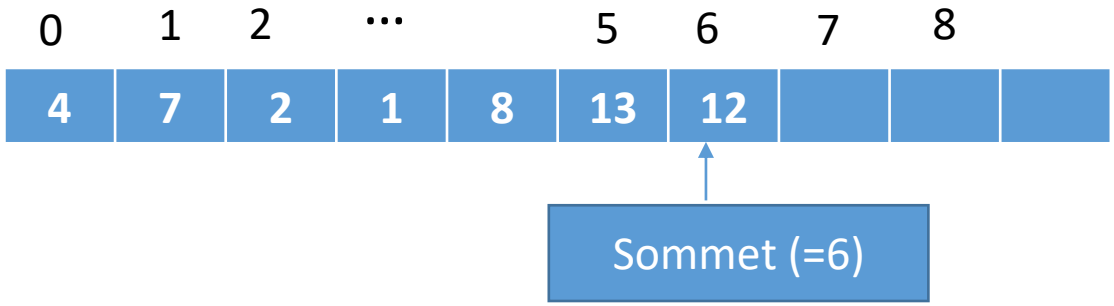
- File et Pile

- Nous cherchons à représenter la pile et la file à l'aide d'une structure de tableau
- Nous supposons que le nombre maximal d'éléments qu'elle peut contenir est connu (= taille du tableau)

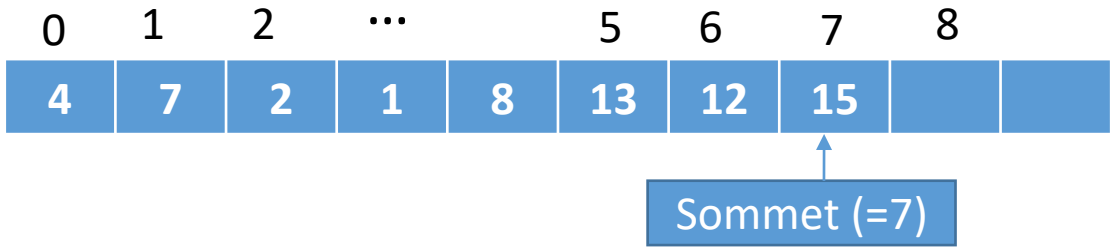
Pile

- Pour une pile les ajouts et les suppressions s'effectuent au même endroit.

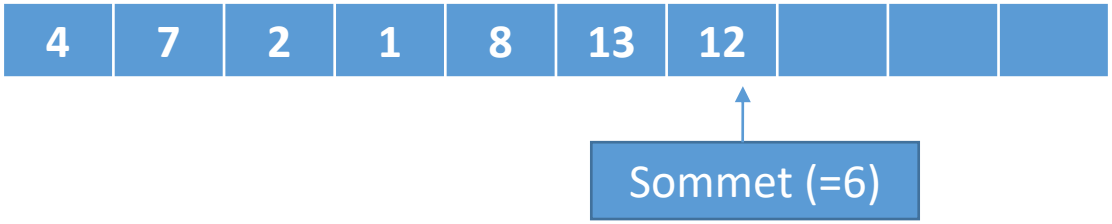
On définit Sommet qui indiquera l'endroit du sommet de la pile



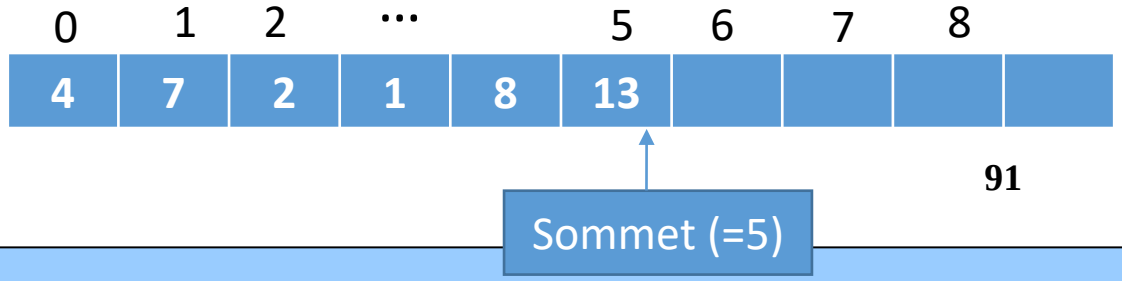
Dans ce cas: après Empiler (15)



Dépiler donne :



Encore Dépiler :



Pile

- Programmation de pile sur tableau

- La programmation est simple et immédiate
  - On prend Sommet de type entier
  - Le sommet contiendra toujours l'indice du sommet de la pile.
  - On ne dépassera jamais la capacité du tableau du fait de l'hypothèse sur la taille maximale de la pile !

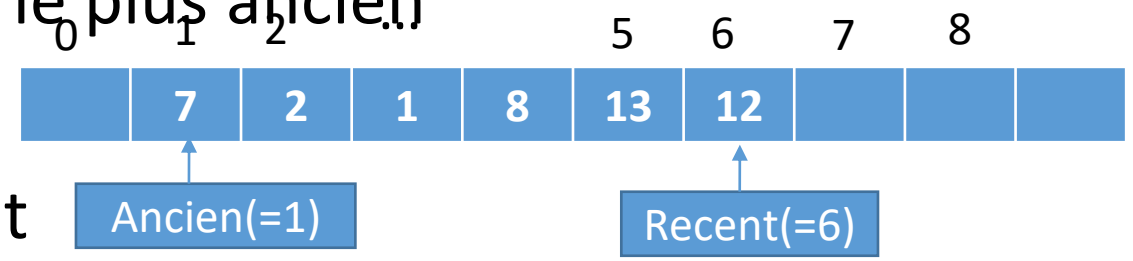
File

- Nous cherchons ici une représentation sous forme de tableau pour le cas d'une file
- Nous supposons que le nombre maximal d'élément que la file peut contenir est connu (= taille du tableau)
- Pour une file, sous savons que les ajouts et les suppressions s'effectuent à des endroits différents
- Nous devons donc utiliser deux variables; comme par exemple :
  - Récent pour indiquer l'emplacement de l'élément le plus récent
  - Ancien pour indiquer l'endroit de l 'élément le plus ancien

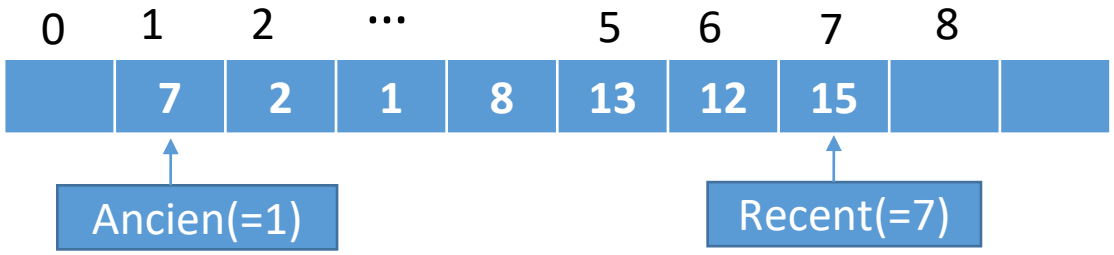
File

Ancien = l'endroit de l'élément le plus ancien

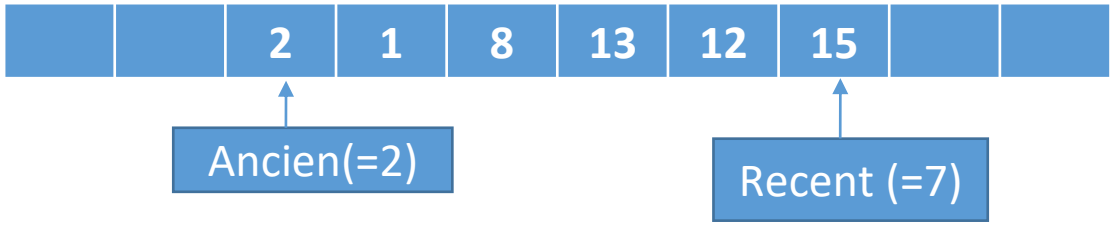
Recent = élément le plus récent



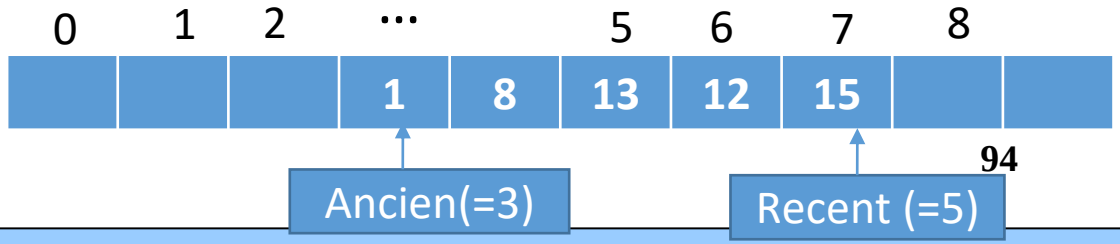
Dans ce cas; Apres Enfiler (15)



Défiler donne :



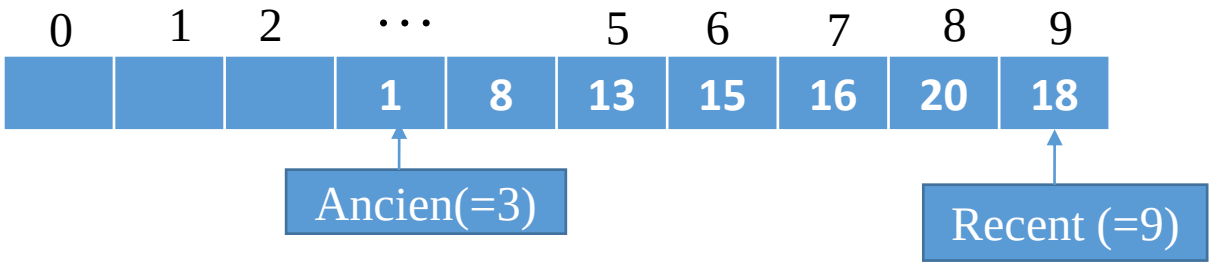
Encore Défiler :



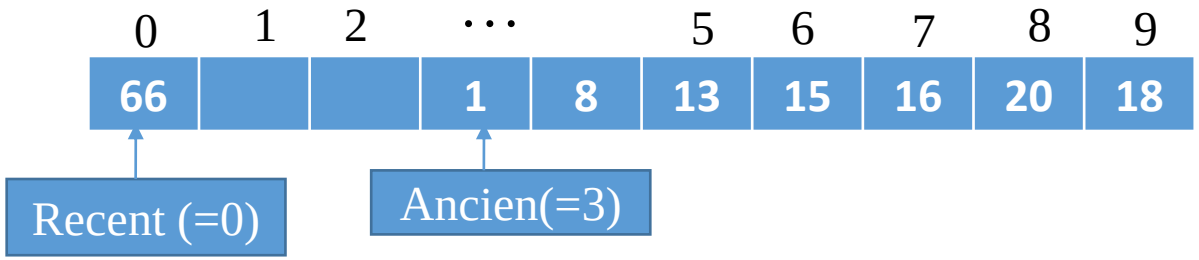
File

- Traitement des débordements

- Contrairement au cas d'un pile les indicateurs Recent et Ancien peuvent dépasser la dernière case du tableau :



- Que se passe t-il lorsqu'on atteint la dernière case du tableau i.e. lors de la (n+1)ème insertion ?



File

- Mise à jour de Recent & Ancien

- Remarque : lors de la mise en file il faut :
  - incrémenter Recent (si Recent < n-1)
  - Mettre Recent = 0 sinon

- Remarque : lors de La suppression il faut :
  - incrémenter Ancien (si Ancien < n-1)
  - Mettre Ancien = 0 sinon

File

- Fonction d'indice de MAJ de Recent & ancien

- En fait la mise à jour de Recent se fera toujours comme suit :
  - Recent <- (Recent+ 1) mod n

- De même pour le mise à jour de Ancien sera :
  - Ancien <- (Ancien +1)mod n

File

- Teste de validité

- Il faut traiter les cas limites suivants:
  - Enfilement dans un tableau plein
  - Défilement dans un tableau contenant une seul élément
  - Défilement dans un tableau ne contenant pas d’ élément

File

- Autre Solutions : après chaque insertion ou suppression réarranger le tableau en fixant la tête de file dans la première case du tableau

- Enfiler 22

- Défiler

Remarque 1 : Dans les deux cas d ’insertion et de suppression , Recent est seul à varier

Remarque 2 : le réarrangement, en cas de défiler, est de complexité algorithmique O(n)

Example d'implementation

Linked List

- On line examples

- Basic implementation of linked-list in C
  - https://www.techiedelight.com/linked-list-implementation-part-1/
- Other languages
  - https://www.techiedelight.com/linked-list-implementation-part-2/