Systèmes d'exploitation

ENSISA 1A

Maxime Devanne

maxime.devanne@uha.fr

Bureau 3.37

1A IR

1er semestre

UNIVERSITÉ HAUTE-ALSACE

ensisa

Systèmes d'exploitation

ENSISA 1A

Chapitre 3

Mémoire

Objectifs

Objectifs pédagogiques du chapitre

- Connaître les **caractéristiques** et **l'organisation** de la mémoire
- Comprendre les différentes méthodes **de gestion de la mémoire**
  - Va-et-vient
  - Mémoire virtuelle
- Connaître les **caractéristiques** de la mémoire virtuelle et appréhender son **fonctionnement**
  - **La pagination**
- Comprendre l'utilité de la **segmentation** de la mémoire

Objectifs

Objectifs pédagogiques du chapitre

- Connaître les **caractéristiques** et **l'organisation** de la mémoire
- Comprendre les différentes méthodes **de gestion de la mémoire**
  - Va-et-vient
  - Mémoire virtuelle
- Connaître les **caractéristiques** de la mémoire virtuelle et appréhender son **fonctionnement**
  - **La pagination**
- Comprendre l'utilité de la **segmentation** de la mémoire

- **Introduction**
- La gestion de la mémoire
- La mémoire virtuelle
- Les algorithmes de remplacement des pages
- La segmentation
- Conclusion

La mémoire : une ressource importante

- Evolution de la mémoire
  - 1962 : IBM 7094
    - 32K mots (36 bits/mot) ≈ 160 KO

  - Aujourd'hui : entre 4GO et 64 GO de RAM
  - Soit environ 200 000 fois plus

  - La taille des programmes augmente aussi vite que celle des mémoires

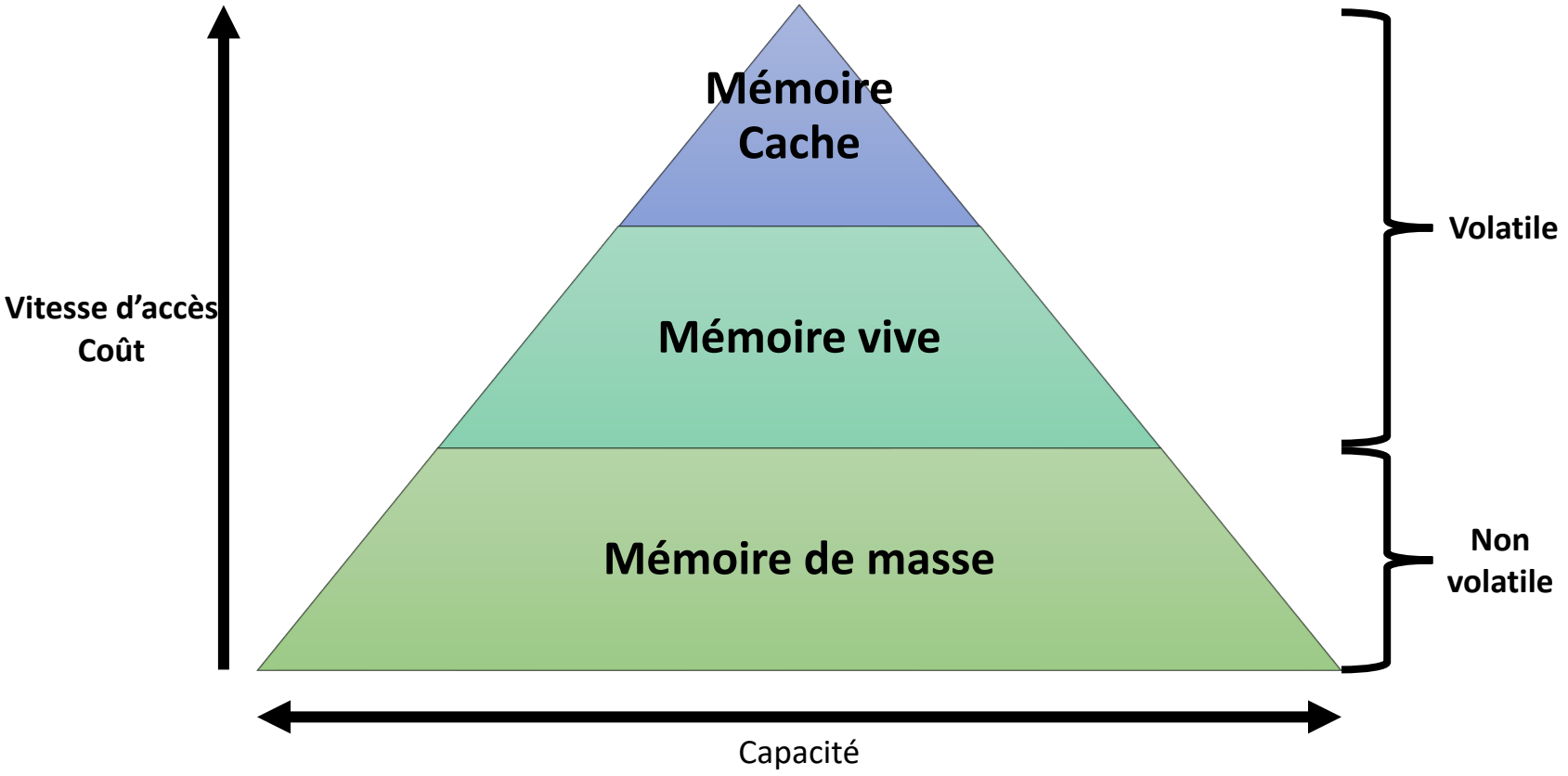
La mémoire : une ressource importante

- Le souhait de tous les programmeurs
  - Mémoire infinie
  - Mémoire rapide d'accès
  - Mémoire non volatile
    - Qui ne perd pas son contenu lorsque le courant est coupé
  - Peu coûteuse

  - La technologie actuelle ne le permet pas
  - La plupart des ordinateurs disposent d'une **hiérarchisation de la mémoire**

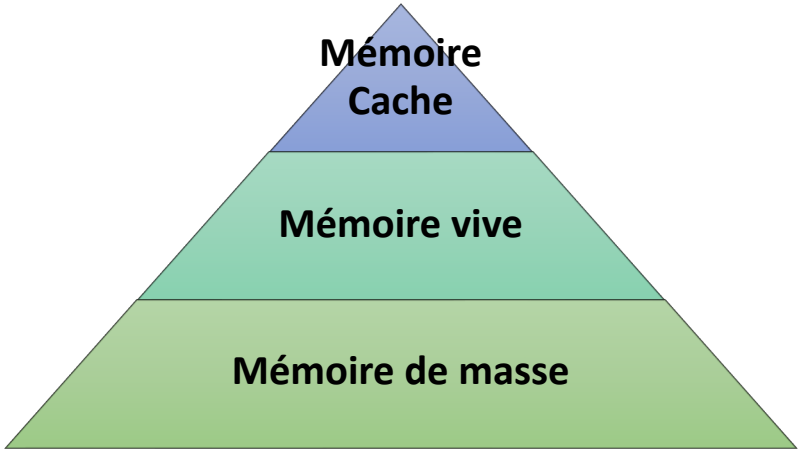
La mémoire : une ressource importante

- Hiérarchisation de la mémoire



La mémoire : une ressource importante

- Hiérarchisation de la mémoire
  - Le SE a pour rôle de coordonner comment ces différentes mémoires sont utilisées



→ Gestion réalisée dans le matériel

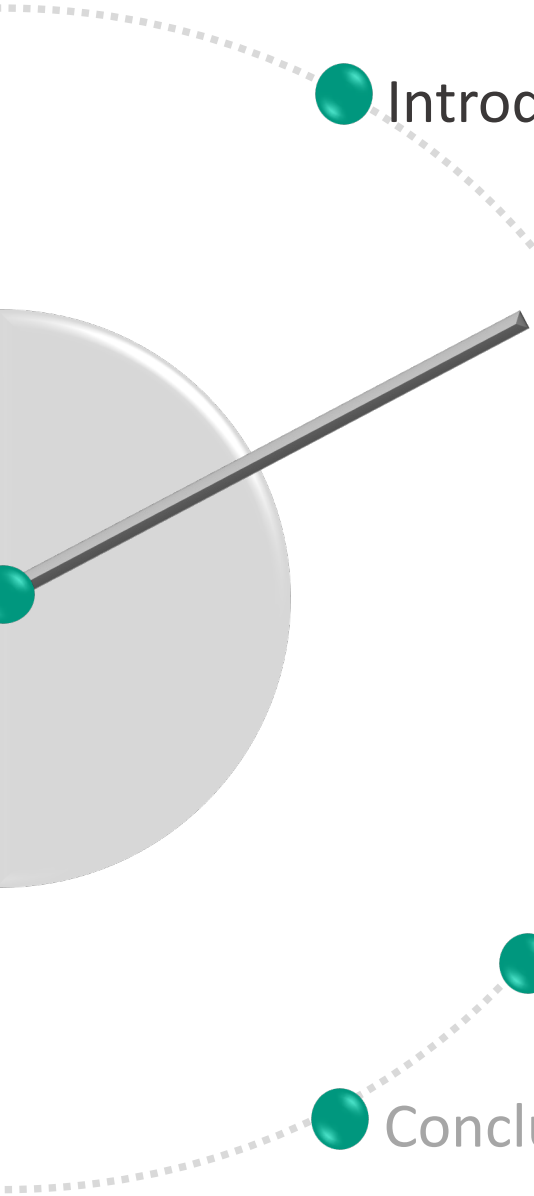
→ **Gestion de la mémoire principale : RAM (Random Access Memory)**

→ Systèmes de fichiers : chapitre suivant

La mémoire : une ressource importante

- La mémoire principale (RAM)
  - Elle permet au SE et aux processus de stocker des informations

  - Le **gestionnaire de mémoire** (Management Memory Unit – MMU)
    - Garder une trace de la partie de la mémoire qui est en cours d'utilisation et de celle qui ne l'est pas
    - Allouer la mémoire aux processus qui en ont besoin
    - Libérer la mémoire quand les processus ont terminé leur travail



- Introduction
- **La gestion de la mémoire**
- La mémoire virtuelle
- Les algorithmes de remplacement des pages
- La segmentation
- Conclusion

La gestion de la mémoire

Défis

- La gestion de la mémoire pose des problèmes :
  - Où mettre le noyau (permanent) ?
  - Où mettre les processus (dynamiques) ?
  - Comment empêcher un processus de modifier librement la mémoire du noyau ou d'un autre processus ?
  - Que faire lorsque la mémoire est pleine ?

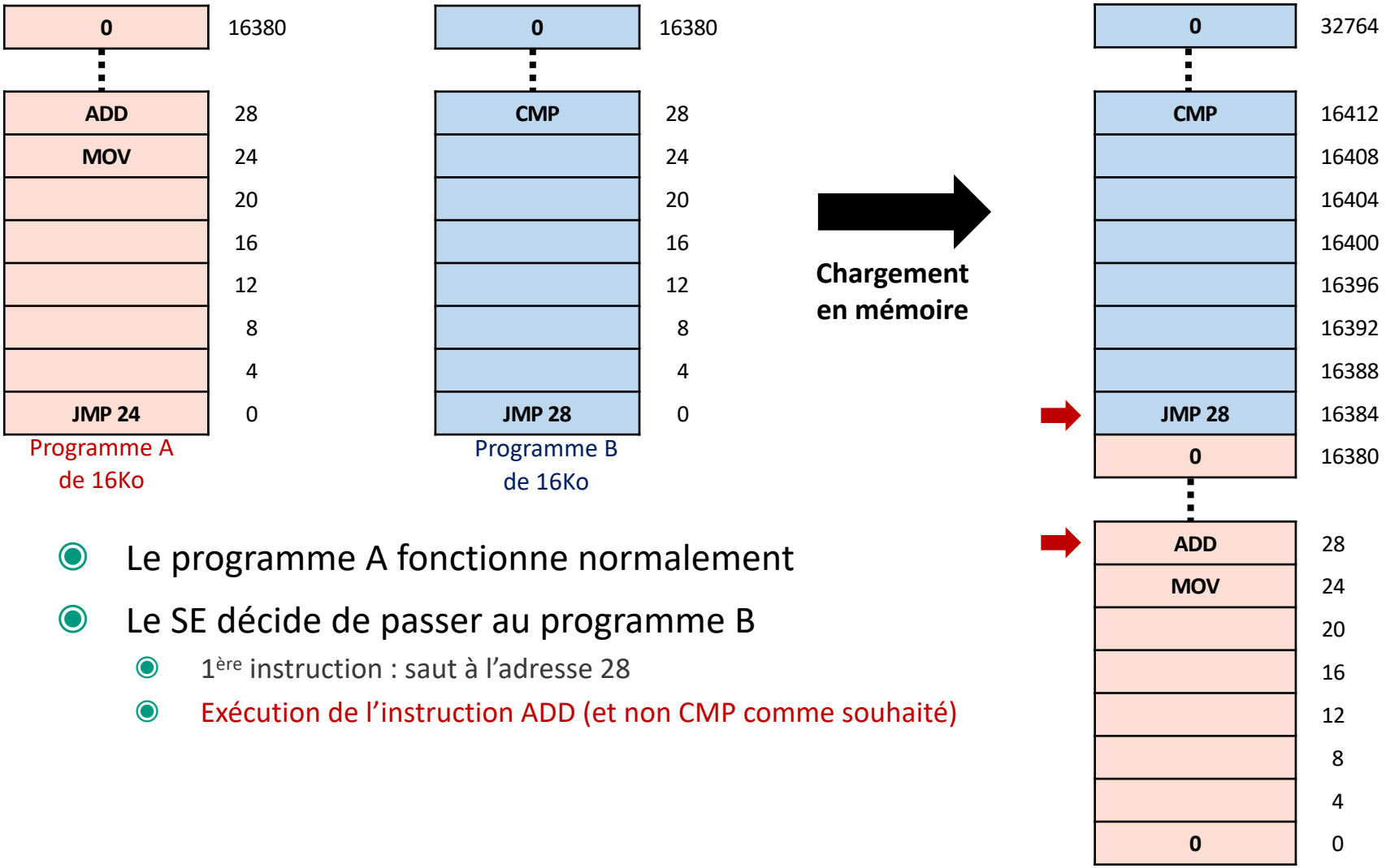
La gestion de la mémoire

Sans abstraction

- Les ordinateurs d'avant 1980 ne disposaient pas d'abstraction de la mémoire
  - Chaque programme voyait simplement la mémoire physique
  - Inconvénients :
    - Risque de plantage général
      - Les programmes de l'utilisateurs ont accès à chaque octet de la mémoire
      - Risque d'accès aux mêmes octets…
    - Il n'était alors pas possible de faire de la multiprogrammation
      - Ou alors copier l'intégralité de la mémoire sur le disque avant de charger et exécuter le programme suivant
      - Exécuter plusieurs programmes est très coûteux

La gestion de la mémoire

Sans abstraction



- Le programme A fonctionne normalement
- Le SE décide de passer au programme B
  - 1ère instruction : saut à l'adresse 28
  - Exécution de l'instruction ADD (et non CMP comme souhaité)

Abstraction de la mémoire

- Notion d'espace d'adressage
  - L'abstraction de la mémoire est un mécanisme permettant d'éviter de manipuler les adresses physiques de la mémoire

```
byte[] table = new byte[32]; // allocation de 32 octets
```

  - Les adresses utilisées par les processus sont des adresses virtuelles qui sont converties en adresses physiques à chaque accès mémoire

```
System.out.println(table[10]); // ou est table[10] en memoire ?
```

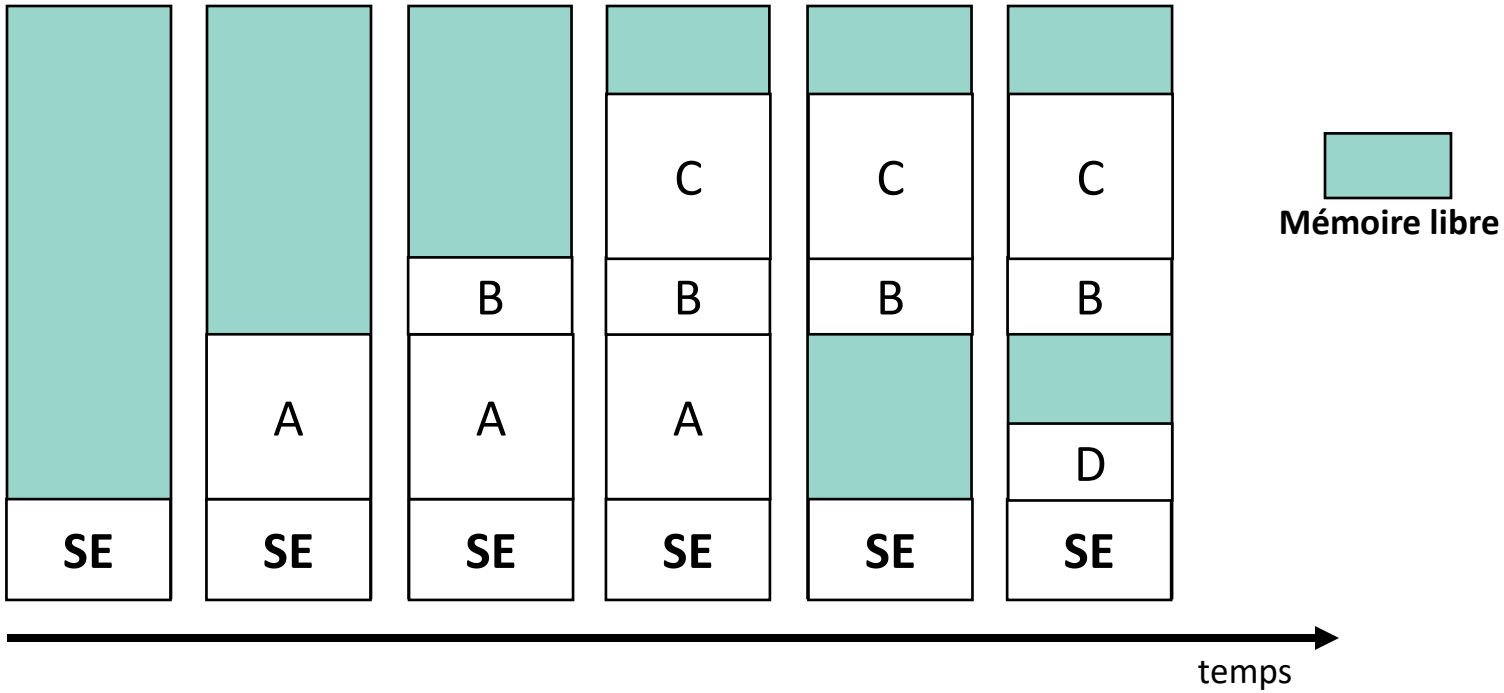
  - Afin d'avoir un temps de conversion des adresses correcte
    - Cette solution repose sur une coopération entre le système et le matériel
      - Le système définit la fonction de conversion
      - Le matériel (MMU) transforme les adresses virtuelles en adresses physiques
  - Les adresses virtuelles d'un processus forment son espace d'adressage

Approches de gestion de la mémoire

- Deux approches de gestion de la mémoire :
  - Va-et-vient :
    - Consiste à considérer chaque processus dans son intégralité
      - Exécution puis placement sur le disque
  - Mémoire virtuelle :
    - Permet aux programmes de s'exécuter même quand ils sont partiellement en mémoire principale

Stratégie va-et-vient

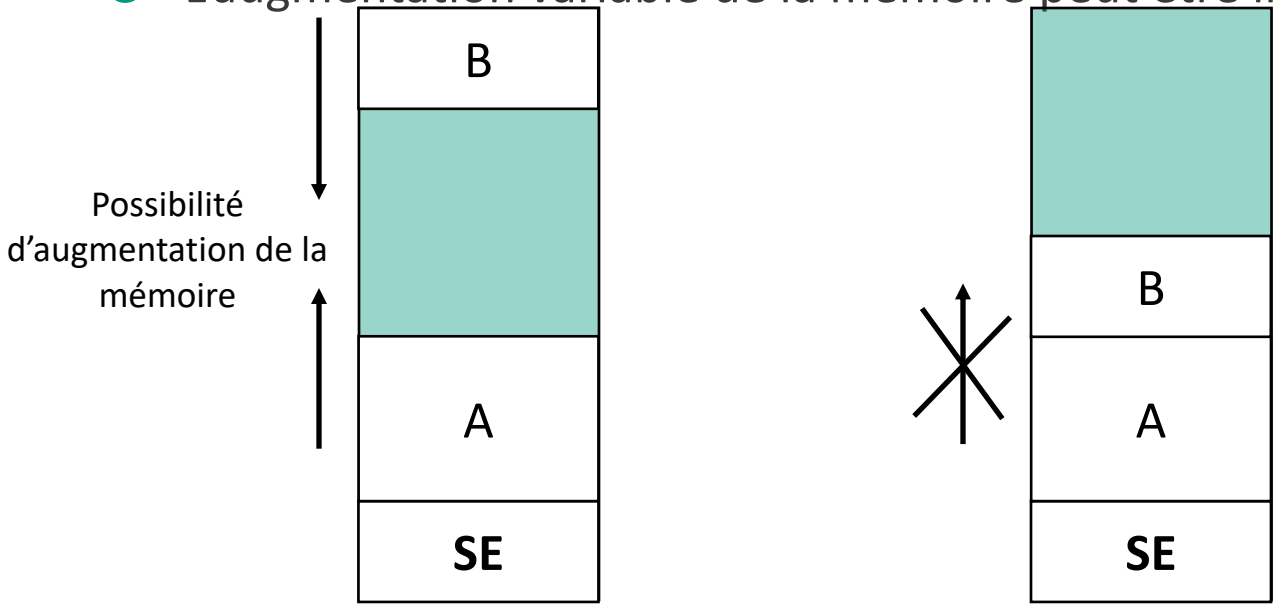
- Va-et-vient
  - Définit au chargement un bloc d'espace mémoire contigüe :



  - Lorsque le processus est terminé, il est stocké sur le disque

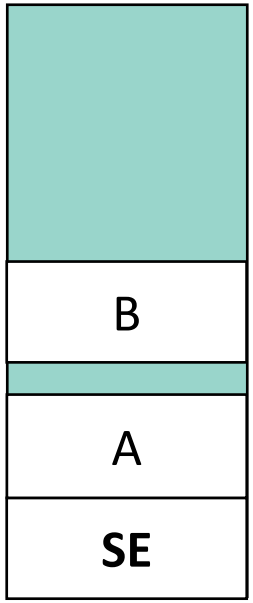
Stratégie va-et-vient

- Compactage
  - L'opération va-et-vient crée de multiples trous dans la mémoire
    - Fragmentation externe
  - Le compactage (défragmentation) consiste à tout recombiner en une seule zone en déplaçant les processus
  - L'augmentation variable de la mémoire peut être impossible



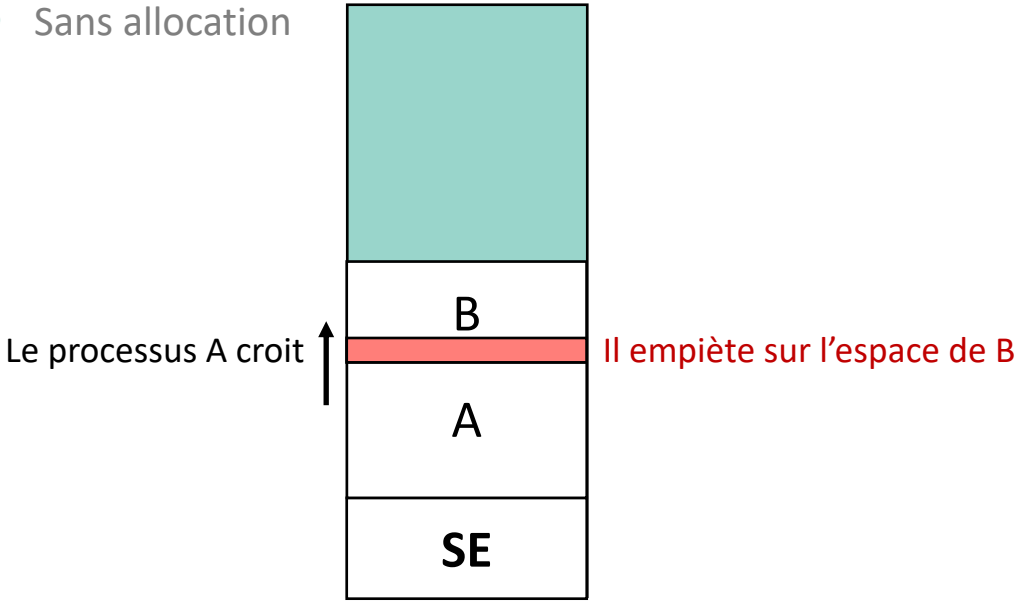
Stratégie va-et-vient

- Accroissement de la mémoire
  - Il est généralement bon de prévoir que la plupart des processus s'agrandiront lors de leur exécution
  - Allouer de la mémoire supplémentaire à chaque fois qu'un processus est chargé
    - Cela évite de devoir déplacer le processus lors de son exécution
      - Sans allocation



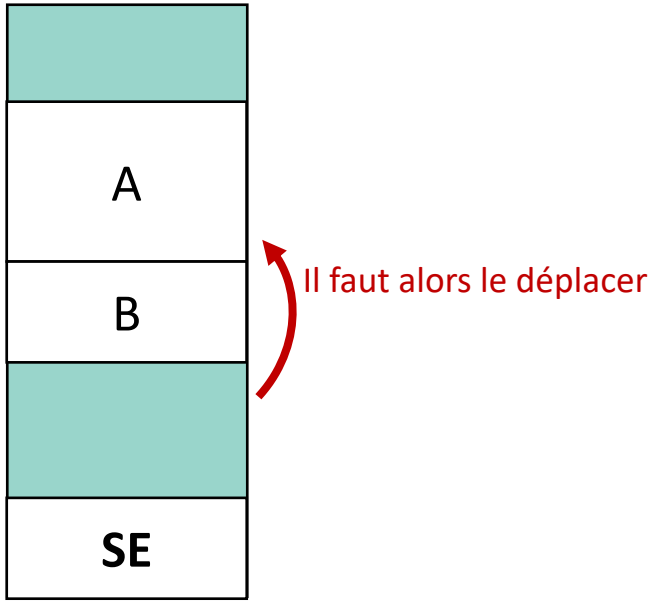
Stratégie va-et-vient

- Accroissement de la mémoire
  - Il est généralement bon de prévoir que la plupart des processus s'agrandiront lors de leur exécution
  - Allouer de la mémoire supplémentaire à chaque fois qu'un processus est chargé
    - Cela évite de devoir déplacer le processus lors de son exécution
      - Sans allocation



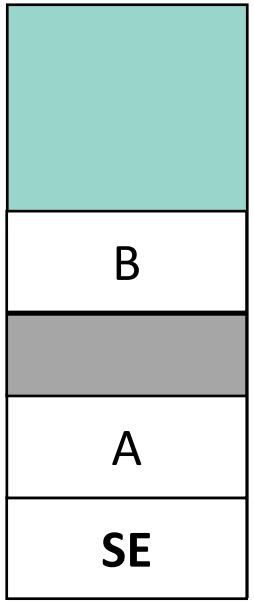
Stratégie va-et-vient

- Accroissement de la mémoire
  - Il est généralement bon de prévoir que la plupart des processus s'agrandiront lors de leur exécution
  - Allouer de la mémoire supplémentaire à chaque fois qu'un processus est chargé
    - Cela évite de devoir déplacer le processus lors de son exécution
      - Sans allocation



Stratégie va-et-vient

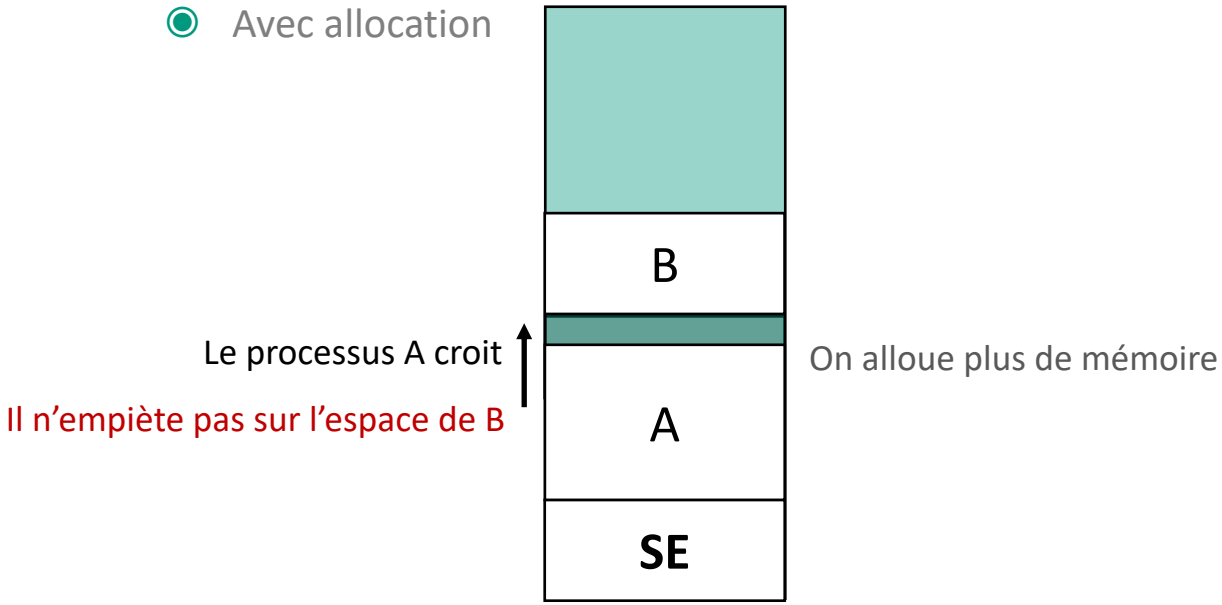
- Accroissement de la mémoire
  - Il est généralement bon de prévoir que la plupart des processus s'agrandiront lors de leur exécution
  - Allouer de la mémoire supplémentaire à chaque fois qu'un processus est chargé
    - Cela évite de devoir déplacer le processus lors de son exécution
      - Avec allocation



On alloue plus de mémoire

Stratégie va-et-vient

- Accroissement de la mémoire
  - Il est généralement bon de prévoir que la plupart des processus s'agrandiront lors de leur exécution
  - Allouer de la mémoire supplémentaire à chaque fois qu'un processus est chargé
    - Cela évite de devoir déplacer le processus lors de son exécution
      - Avec allocation

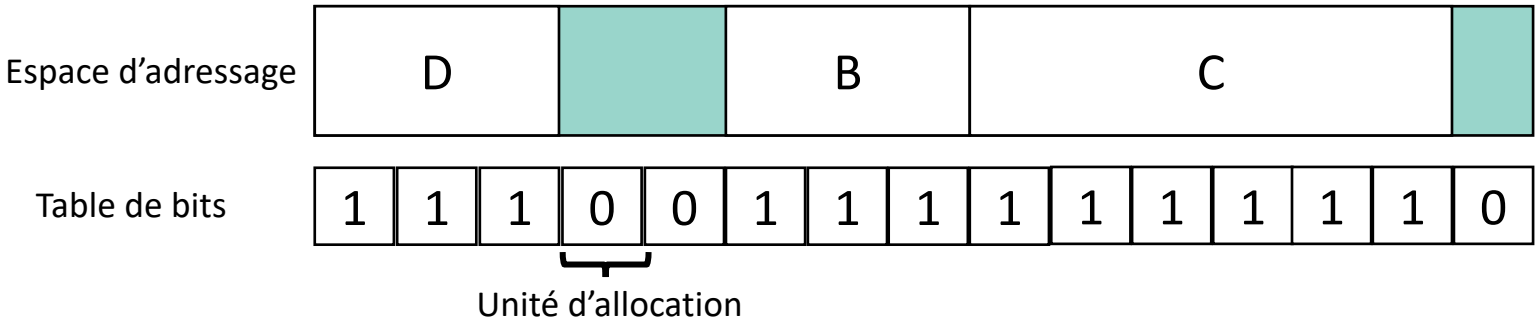


Gestion de la mémoire libre

- Comment savoir où placer les processus ?
- Gestion de l'attribution dynamique de la mémoire
  - Le SE doit garder une trace de l'utilisation de la mémoire
    - Partie de la mémoire libre/disponible
    - Partie de la mémoire utilisée/non disponible
  - Deux méthodes :
    - La table de bits
    - La liste chainée

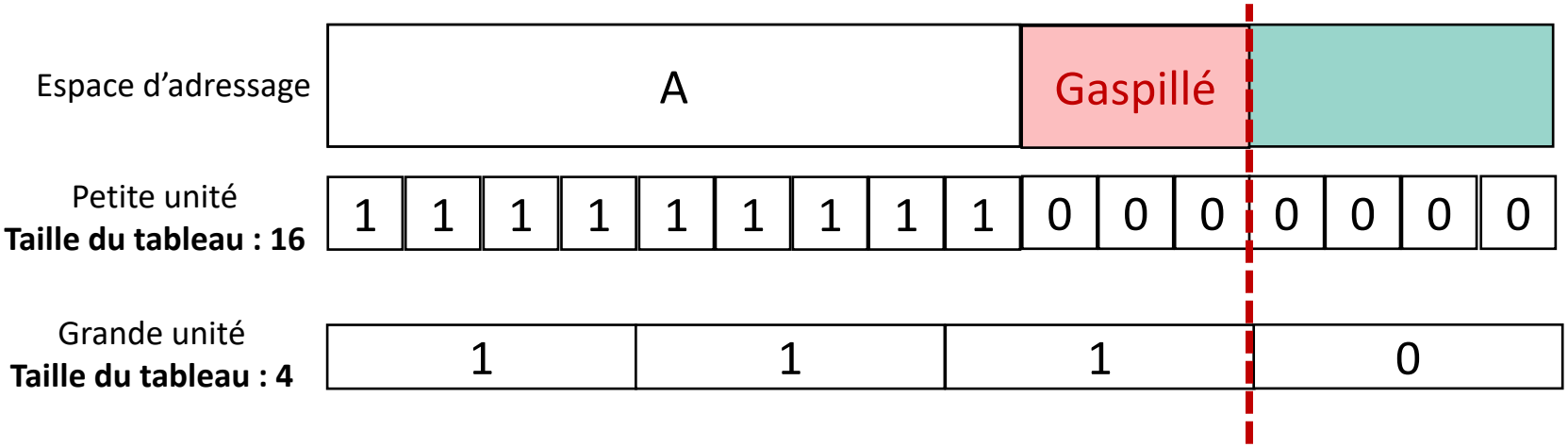
Gestion de la mémoire libre

- Utilisation de table de bits
  - La mémoire est répartie en unités d'allocations
    - Taille variable selon les gestionnaires
      - De quelques octets à plusieurs kilo octets
  - Chaque unité d'allocation correspond à un bit du tableau
    - 0 si le bloc mémoire est disponible
    - 1 si le bloc mémoire est utilisé



Gestion de la mémoire libre

- Utilisation de table de bits
  - La taille de l'unité d'allocation est un élément fondamental
    - Unité d'allocation petite
      - Taille du tableau de bits **importante**
      - Moins de risque de gaspillage
    - Unité d'allocation grande
      - Taille du tableau de bits **petite**
      - Mais risque de gaspillage dans la dernière unité allouée

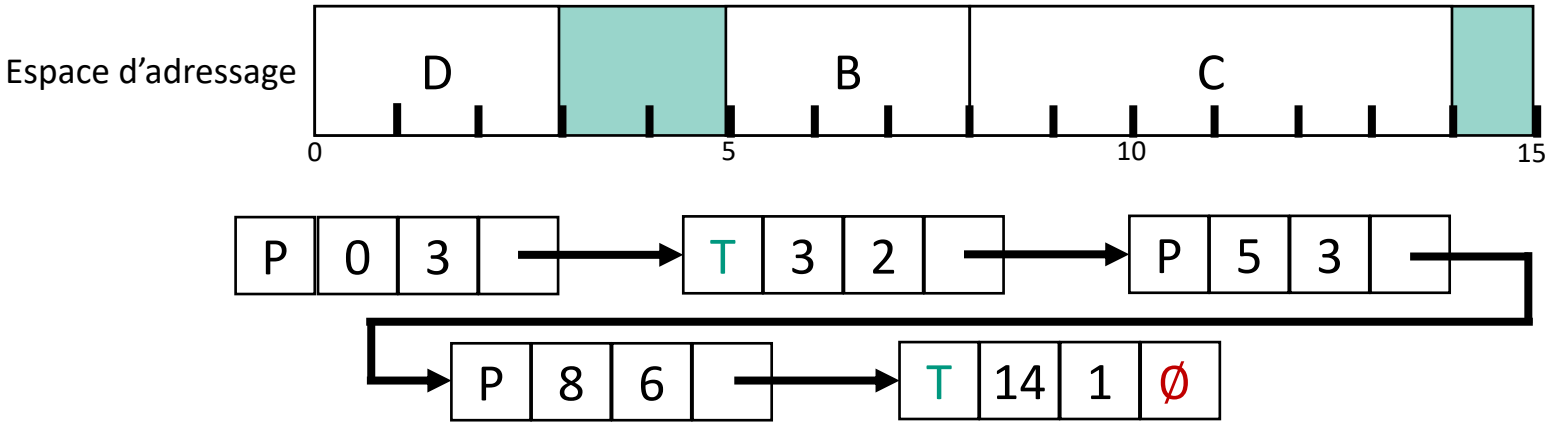


Gestion de la mémoire libre

- Utilisation de table de bits
  - Avantage
    - Un moyen simple de garder une trace des « mots » mémoire dans une quantité fixe de mémoire
    - La taille de la table ne dépend que de la mémoire et de l'unité d'allocation
  - Inconvénient
    - Lorsqu'un processus de **k unités** doit être chargé en mémoire
    - Le gestionnaire de mémoire doit parcourir le tableau pour trouver un espace libre d'au moins **k unités**
      - **k** bits à 0 consécutivement
    - **Opération lente**

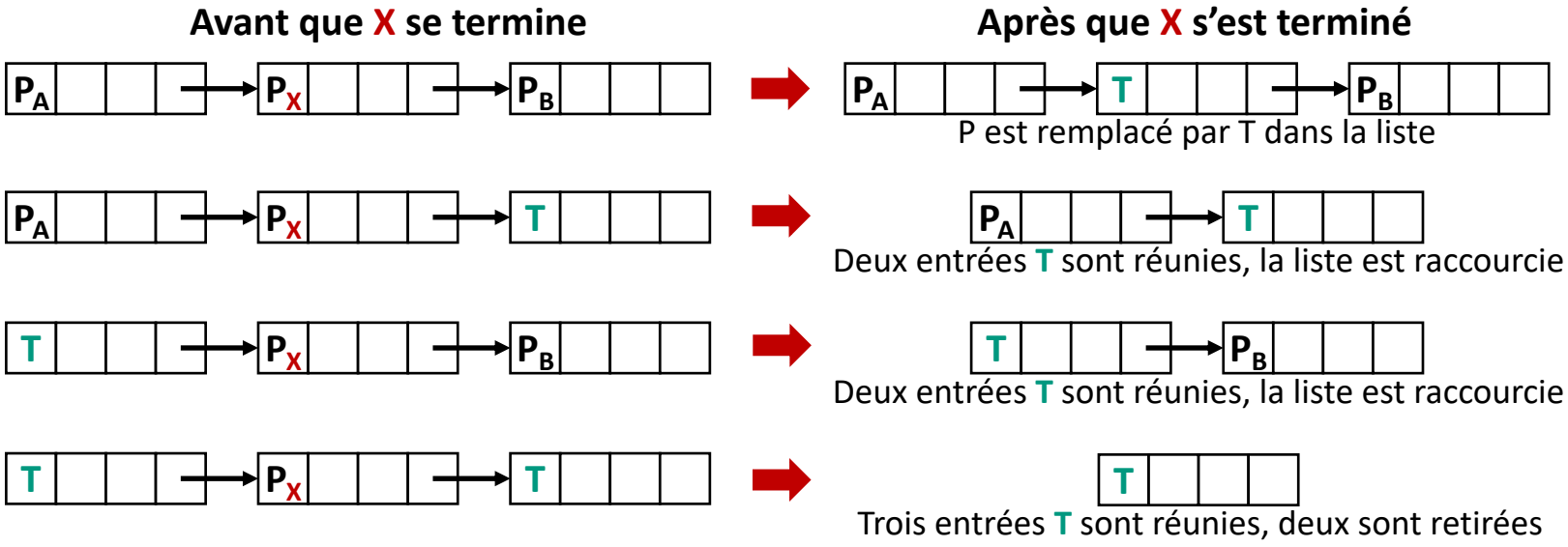
Gestion de la mémoire libre

- Utilisation d'une liste chainée
  - Liste chainée des segments de mémoires alloués et libres
    - Un segment est :
      - Soit un processus (P)
      - Soit un trou entre deux processus (T)
    - Chaque entrée dans la liste indique
      - Le type de segment
      - L'adresse à laquelle il débute et sa longueur
      - Spécifie un pointeur vers le segment suivant



Gestion de la mémoire libre

- Utilisation d'une liste chainée
  - La liste est la plupart du temps triée par adresse
    - Pratique lorsqu'un processus se termine
    - La mise à jour de la liste est directe
  - Cas possible lorsqu'un processus **X** qui se termine a deux voisins
    - Tous les cas sauf le début et la fin de la liste

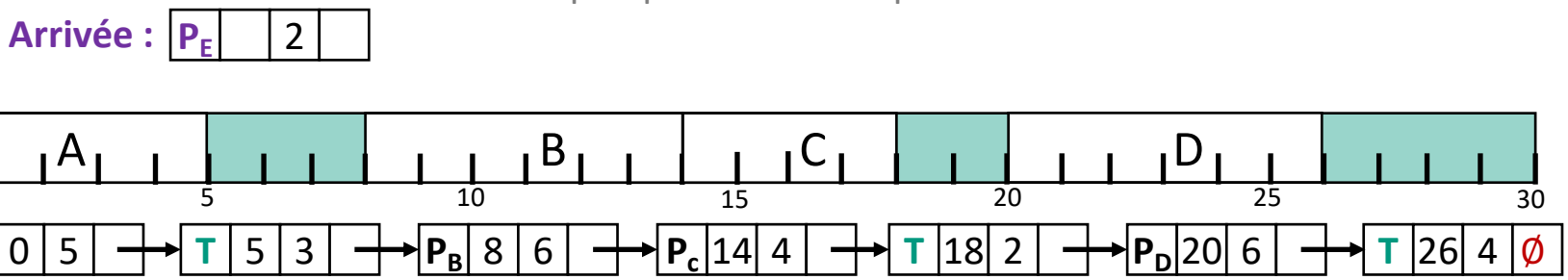


Placement d'un processus en mémoire

- Comment gérer un nouveau processus arrivant ?
  - Où placer le processus en mémoire ?
  - Quel espace mémoire allouer à ce processus ?
- L'utilisation des listes chainées facilite la gestion
  - Tri par adresse
- Il existe plusieurs algorithmes
  - Première zone libre (first fit)
  - Zone libre suivante (next fit)
  - Meilleur ajustement (best fit)
  - Plus grand résidu (worst fit)
  - Placement rapide (quick fit)

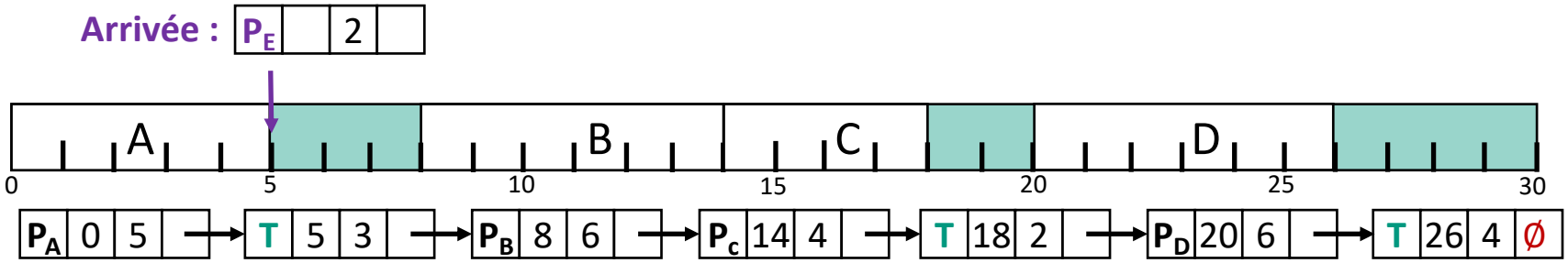
Placement d'un processus en mémoire

- Algorithme de la première zone libre (first fit)
  - Le gestionnaire de mémoire parcourt la liste des segments jusqu'à trouver un trou qui soit assez grand
  - Le trou est ensuite divisé en deux parties
    - Un segment destiné au processus
    - Un segment pour la mémoire non utilisée
      - Sauf cas très peu probable où le processus et le trou ont la même taille



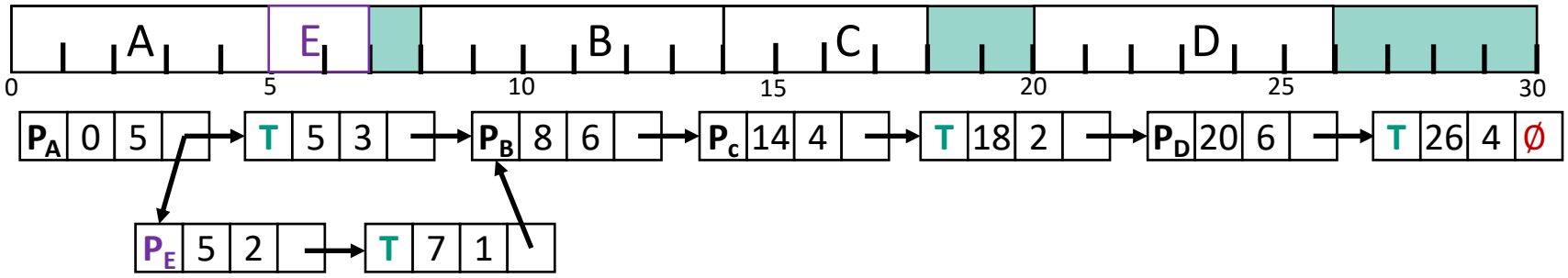
Placement d'un processus en mémoire

- Algorithme de la première zone libre (first fit)
  - Le gestionnaire de mémoire parcourt la liste des segments jusqu'à trouver un trou qui soit assez grand
  - Le trou est ensuite divisé en deux parties
    - Un segment destiné au processus
    - Un segment pour la mémoire non utilisée
      - Sauf cas très peu probable où le processus et le trou ont la même taille



Placement d'un processus en mémoire

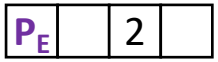
- Algorithme de la première zone libre (first fit)
  - Le gestionnaire de mémoire parcourt la liste des segments jusqu'à trouver un trou qui soit assez grand
  - Le trou est ensuite divisé en deux parties
    - Un segment destiné au processus
    - Un segment pour la mémoire non utilisée
      - Sauf cas très peu probable où le processus et le trou ont la même taille

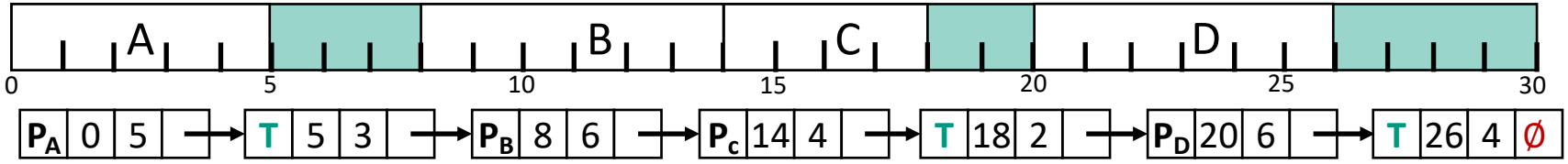


  - Algorithme rapide car limite la recherche

Placement d'un processus en mémoire

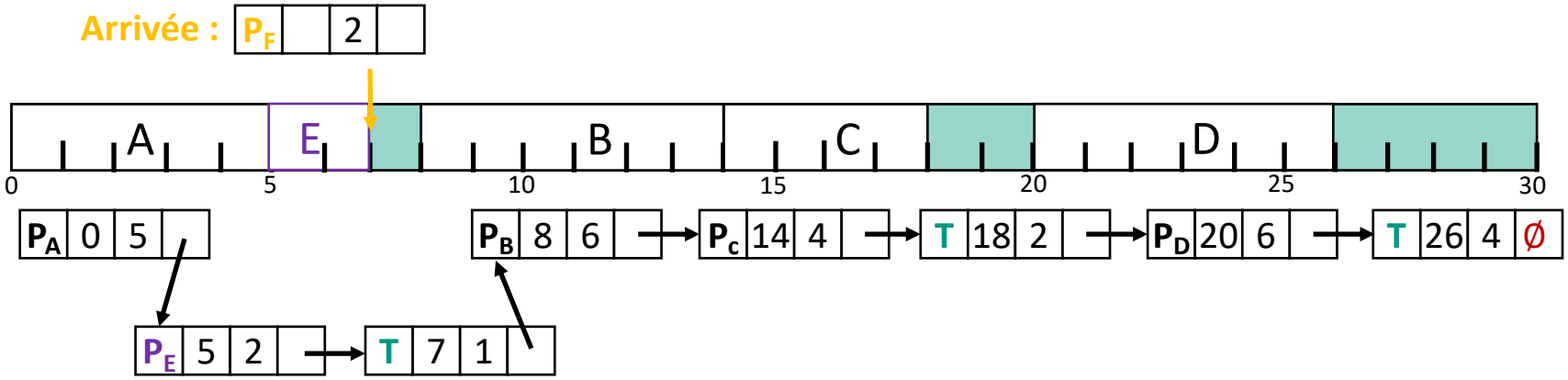
- Algorithme de la zone libre suivante (next fit)
  - Variante de la première zone libre
  - Mémorise également la position **i** de l'espace libre trouvé
  - Permet de recommencer la recherche suivante à cette position **i**
    - Fonctionne mieux si le processus suivant nécessite au moins autant d'espace mémoire que le précédent

Arrivée : 



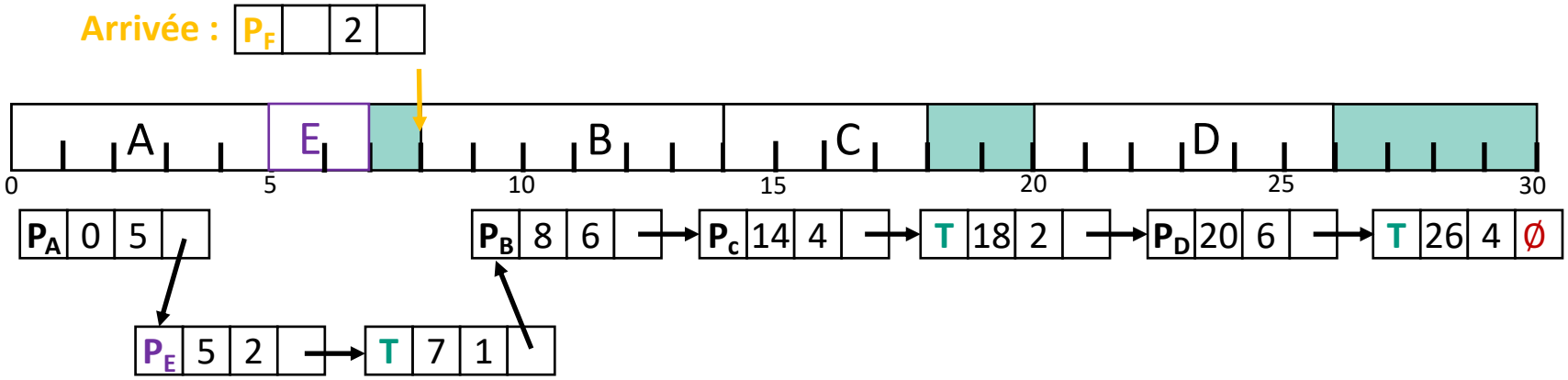
Placement d'un processus en mémoire

- Algorithme de la zone libre suivante (next fit)
  - Variante de la première zone libre
  - Mémorise également la position **i** de l'espace libre trouvé
  - Permet de recommencer la recherche suivante à cette position **i**
    - Fonctionne mieux si le processus suivant nécessite au moins autant d'espace mémoire que le précédent



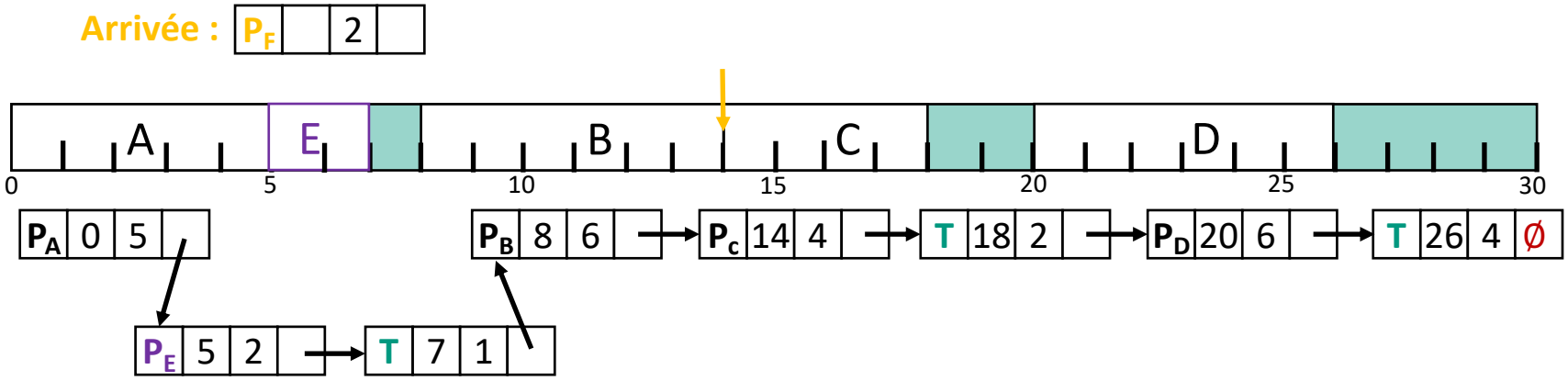
Placement d'un processus en mémoire

- Algorithme de la zone libre suivante (next fit)
  - Variante de la première zone libre
  - Mémorise également la position **i** de l'espace libre trouvé
  - Permet de recommencer la recherche suivante à cette position **i**
    - Fonctionne mieux si le processus suivant nécessite au moins autant d'espace mémoire que le précédent



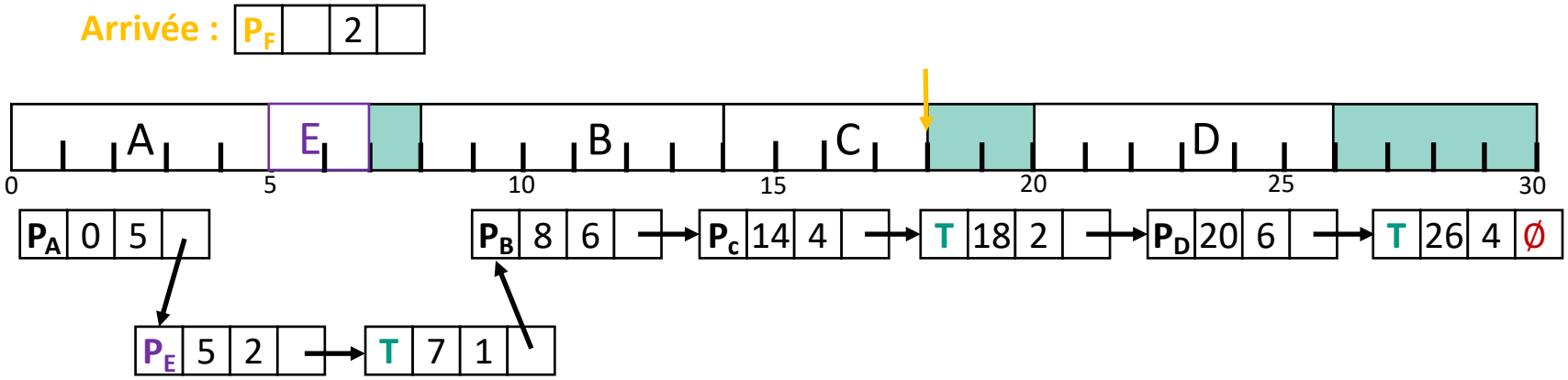
Placement d'un processus en mémoire

- Algorithme de la zone libre suivante (next fit)
  - Variante de la première zone libre
  - Mémorise également la position **i** de l'espace libre trouvé
  - Permet de recommencer la recherche suivante à cette position **i**
    - Fonctionne mieux si le processus suivant nécessite au moins autant d'espace mémoire que le précédent



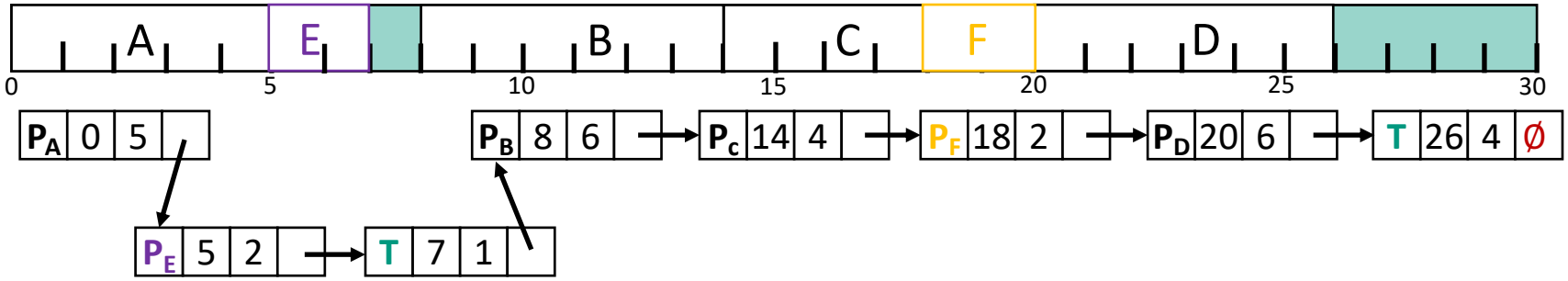
Placement d'un processus en mémoire

- Algorithme de la zone libre suivante (next fit)
  - Variante de la première zone libre
  - Mémorise également la position **i** de l'espace libre trouvé
  - Permet de recommencer la recherche suivante à cette position **i**
    - Fonctionne mieux si le processus suivant nécessite au moins autant d'espace mémoire que le précédent



Placement d'un processus en mémoire

- Algorithme de la zone libre suivante (next fit)
  - Variante de la première zone libre
  - Mémorise également la position **i** de l'espace libre trouvé
  - Permet de recommencer la recherche suivante à cette position **i**
    - Fonctionne mieux si le processus suivant nécessite au moins autant d'espace mémoire que le précédent

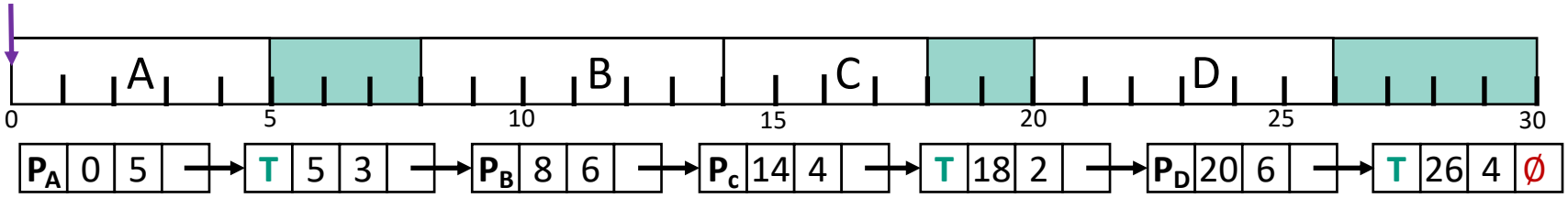


  - Des études ont montré que les performances sont légèrement meilleures

Placement d'un processus en mémoire

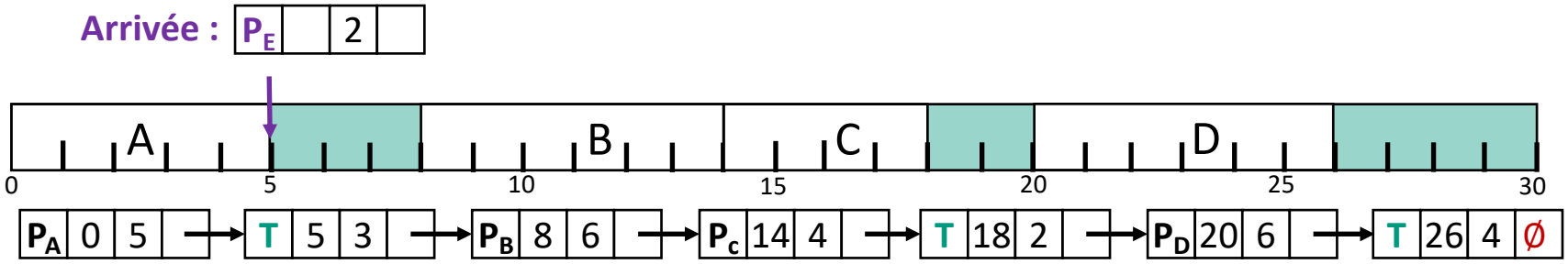
- Algorithme du meilleur ajustement (best fit)
  - Effectue une recherche dans toute la liste
  - Choisit le plus petit trou adéquat
    - Plutôt que de « casser » un gros trou

Arrivée : | P_E | | 2 | |



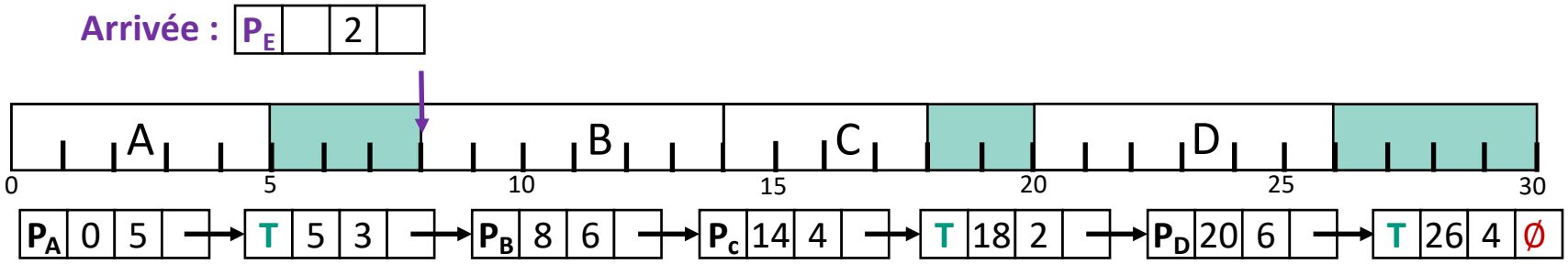
Placement d'un processus en mémoire

- Algorithme du meilleur ajustement (best fit)
  - Effectue une recherche dans toute la liste
  - Choisit le plus petit trou adéquat
    - Plutôt que de « casser » un gros trou



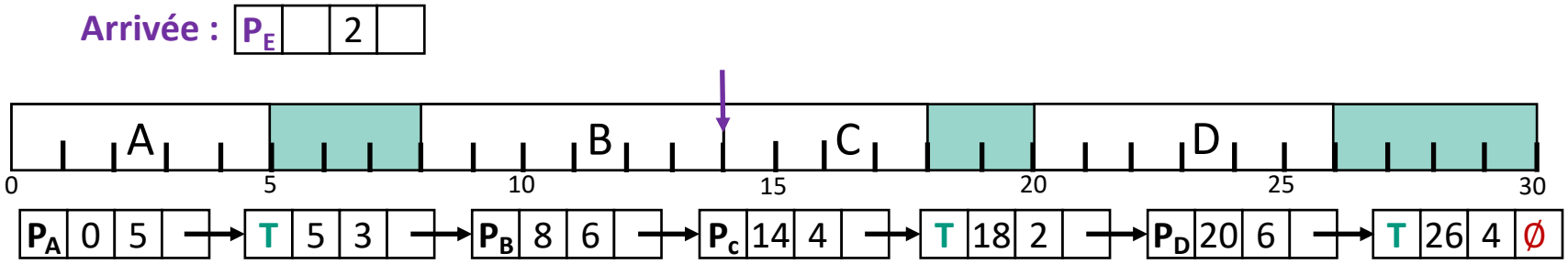
Placement d'un processus en mémoire

- Algorithme du meilleur ajustement (best fit)
  - Effectue une recherche dans toute la liste
  - Choisit le plus petit trou adéquat
    - Plutôt que de « casser » un gros trou



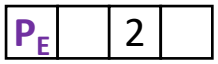
Placement d'un processus en mémoire

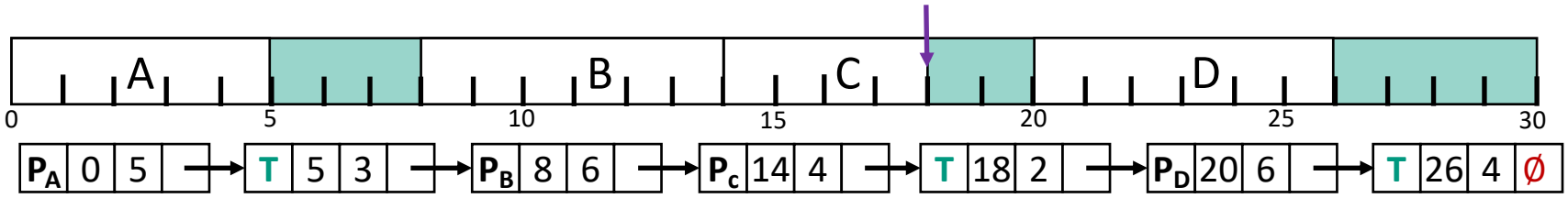
- Algorithme du meilleur ajustement (best fit)
  - Effectue une recherche dans toute la liste
  - Choisit le plus petit trou adéquat
    - Plutôt que de « casser » un gros trou



Placement d'un processus en mémoire

- Algorithme du meilleur ajustement (best fit)
  - Effectue une recherche dans toute la liste
  - Choisit le plus petit trou adéquat
    - Plutôt que de « casser » un gros trou

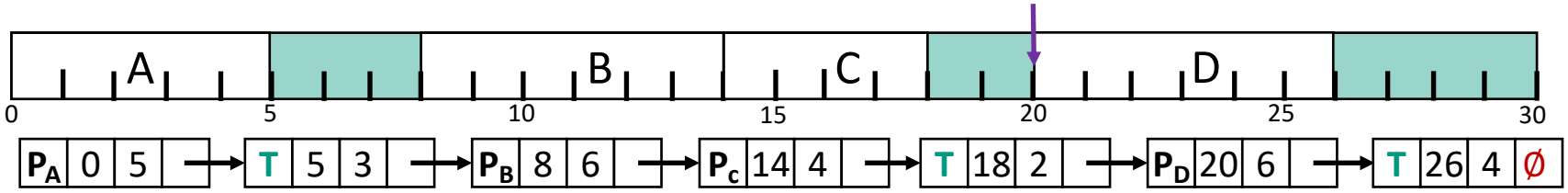
Arrivée : 



Placement d'un processus en mémoire

- Algorithme du meilleur ajustement (best fit)
  - Effectue une recherche dans toute la liste
  - Choisit le plus petit trou adéquat
    - Plutôt que de « casser » un gros trou

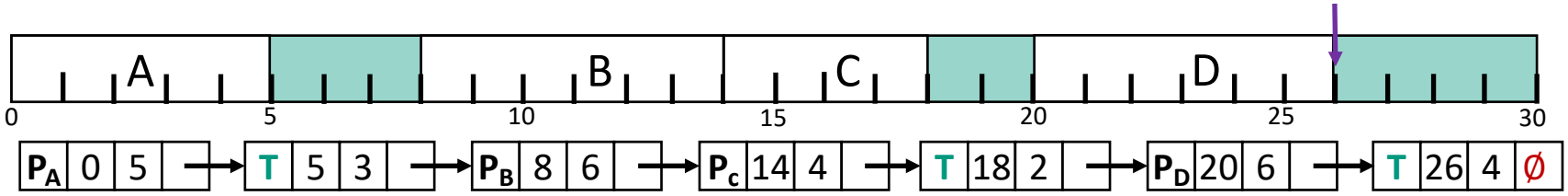
Arrivée : P_E | | 2 |



Placement d'un processus en mémoire

- Algorithme du meilleur ajustement (best fit)
  - Effectue une recherche dans toute la liste
  - Choisit le plus petit trou adéquat
    - Plutôt que de « casser » un gros trou

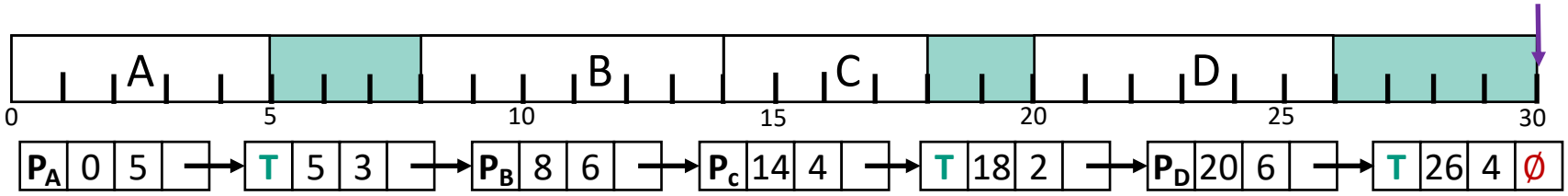
Arrivée : | P_E | | 2 | |



Placement d'un processus en mémoire

- Algorithme du meilleur ajustement (best fit)
  - Effectue une recherche dans toute la liste
  - Choisit le plus petit trou adéquat
    - Plutôt que de « casser » un gros trou

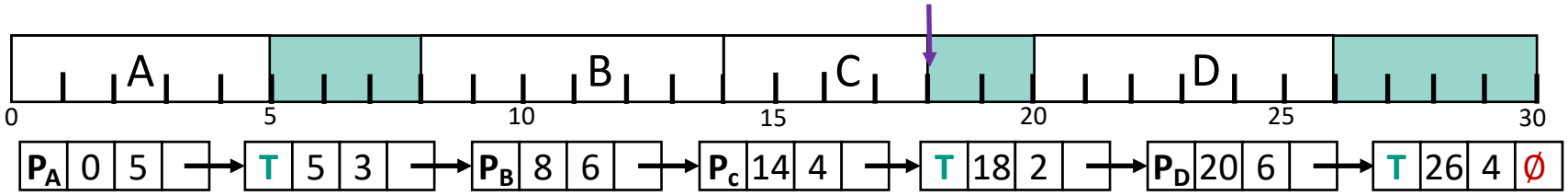
Arrivée : | P_E | | 2 | |



Placement d'un processus en mémoire

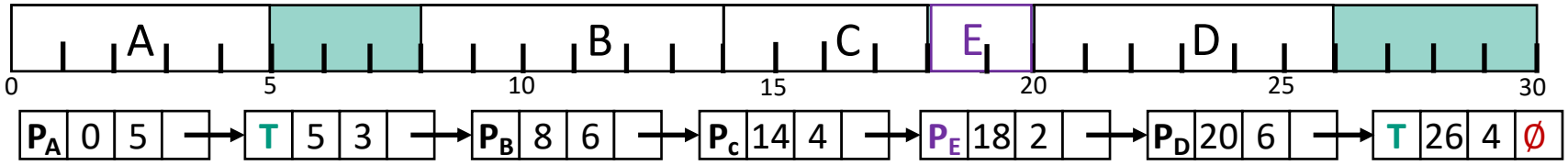
- Algorithme du meilleur ajustement (best fit)
  - Effectue une recherche dans toute la liste
  - Choisit le plus petit trou adéquat
    - Plutôt que de « casser » un gros trou

Arrivée : | P_E | | 2 | |



Placement d'un processus en mémoire

- Algorithme du meilleur ajustement (best fit)
  - Effectue une recherche dans toute la liste
  - Choisit le plus petit trou adéquat
    - Plutôt que de « casser » un gros trou

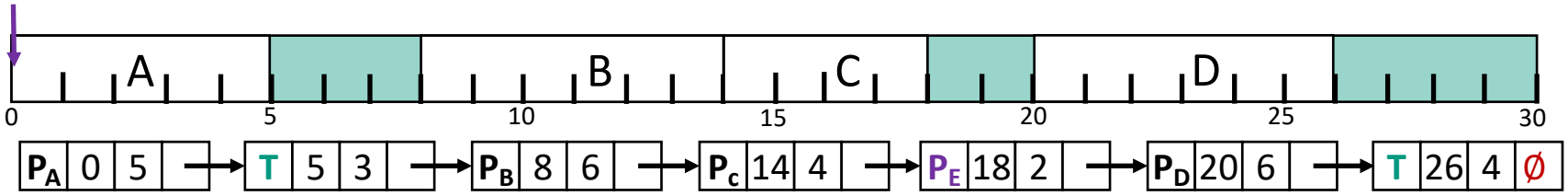


- L'algorithme est plus lent que la première zone libre
  - Recherche sur toute la liste
- L'algorithme peut aussi perde de la mémoire !
  - Il laisse des trous minuscules inutiles et inutilisables

Placement d'un processus en mémoire

- Algorithme du plus grand résidu (worst fit)
  - Pensé pour régler le problème des trous minuscules
  - Choisir le trou qui laisserait un plus grand résidu d'espace libre
    - Le plus grand trou

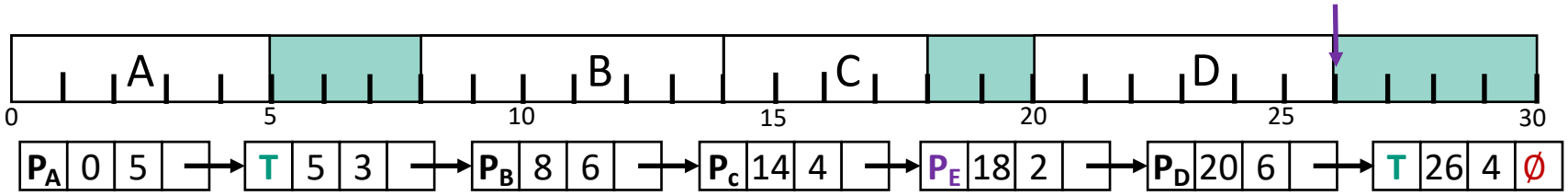
Arrivée : P_E | | 2 |



Placement d'un processus en mémoire

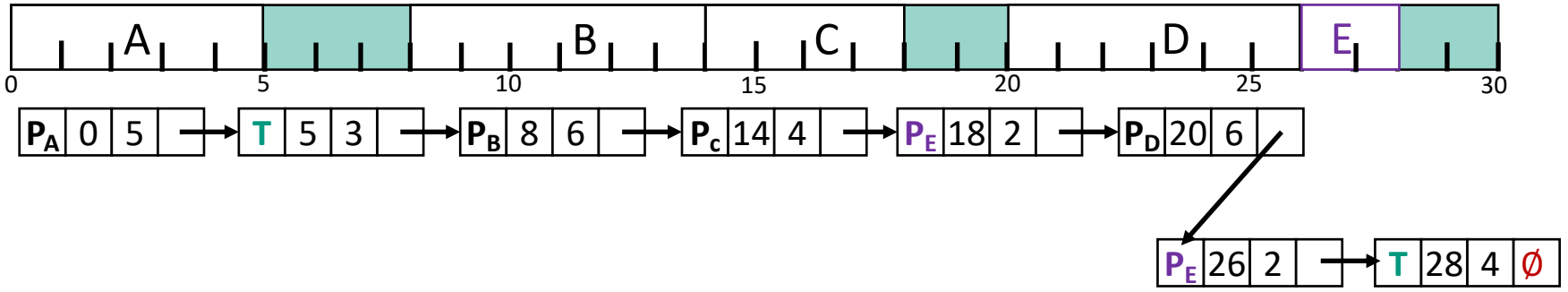
- Algorithme du plus grand résidu (worst fit)
  - Pensé pour régler le problème des trous minuscules
  - Choisir le trou qui laisserait un plus grand résidu d'espace libre
    - Le plus grand trou

Arrivée : | P_E | | 2 | |



Placement d'un processus en mémoire

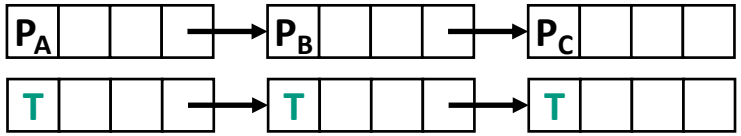
- Algorithme du plus grand résidu (worst fit)
  - Pensé pour régler le problème des trous minuscules
  - Choisir le trou qui laisserait un plus grand résidu d'espace libre
    - Le plus grand trou



- Des études ont montré qu'il n'est en pratique pas plus performant

Placement d'un processus en mémoire

- Comment augmenter la rapidité des algorithmes ?
  - Etablir des listes séparées pour les processus et les trous
  - Les algorithmes ne parcourent que la liste des trous
    - Plus rapide



  - La liste des trous doit être triée dans l'ordre croissant
    - Permet de s'arrêter dès qu'on trouve un trou qui convient
    - First fit et Best fit sont alors équivalents
- L'algorithme du placement rapide (quick fit)
  - Gère des listes de trous séparés pour les tailles les plus communes

- Inconvénient de ces algorithmes
  - Couteux lorsqu'un processus se termine
    - La recherche de ses voisins pour fusion n'est pas évidente

Exemple

- Soit l'espace mémoire suivant
  - Tailles en Ko

Espace libre

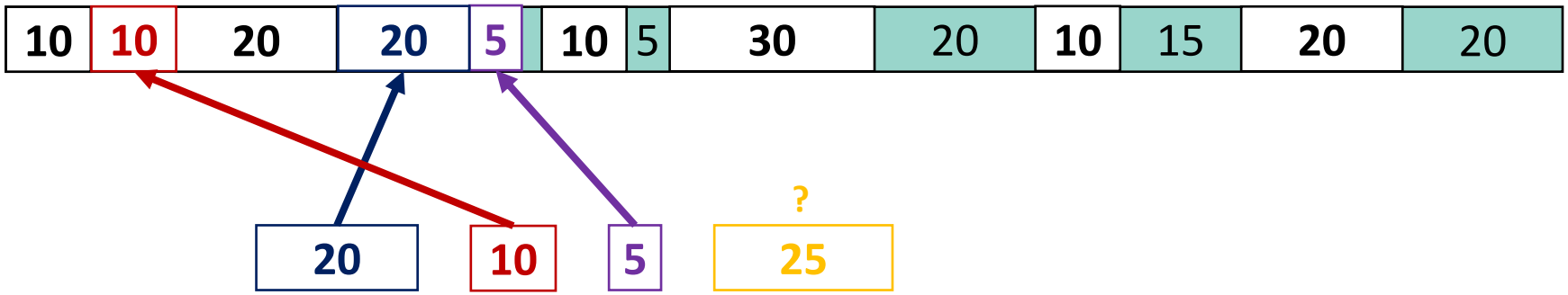
- Des requêtes d'allocation mémoire arrivent dans l'ordre suivant :

  20 **10** **5** **25**

- A quelles adresses sont alloués les blocs avec « First fit » ?
- A quelles adresses sont alloués les blocs avec « Next fit » ?
- A quelles adresses sont alloués les blocs avec « Best fit » ?
- A quelles adresses sont alloués les blocs avec « Worst fit » ?

Solution

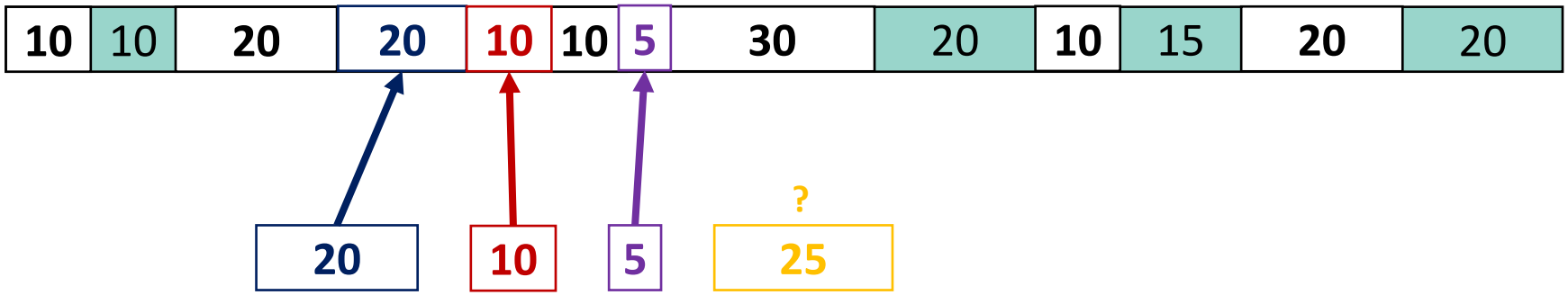
- Soit l'espace mémoire suivant
  - Tailles en Ko



- A quelles adresses sont alloués les blocs avec **« First fit » ?**

Solution

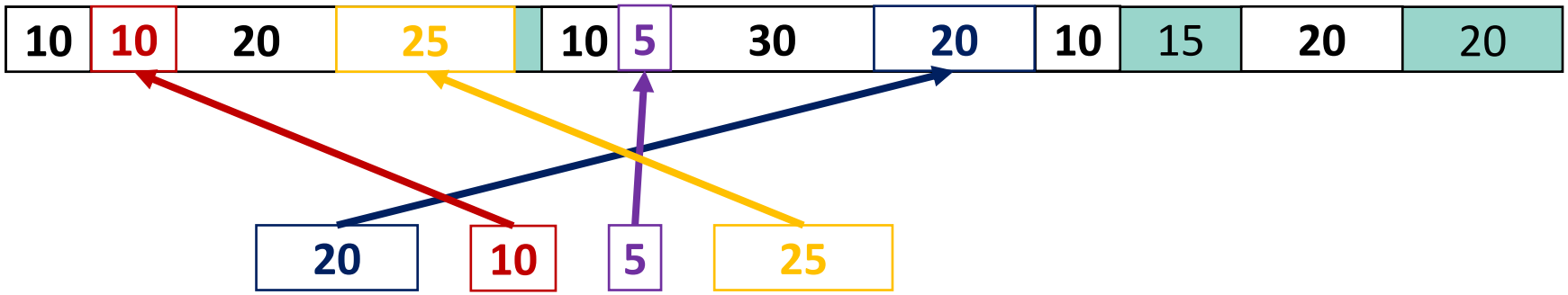
- Soit l'espace mémoire suivant
  - Tailles en Ko



- A quelles adresses sont alloués les blocs avec **« Next fit »** ?

Solution

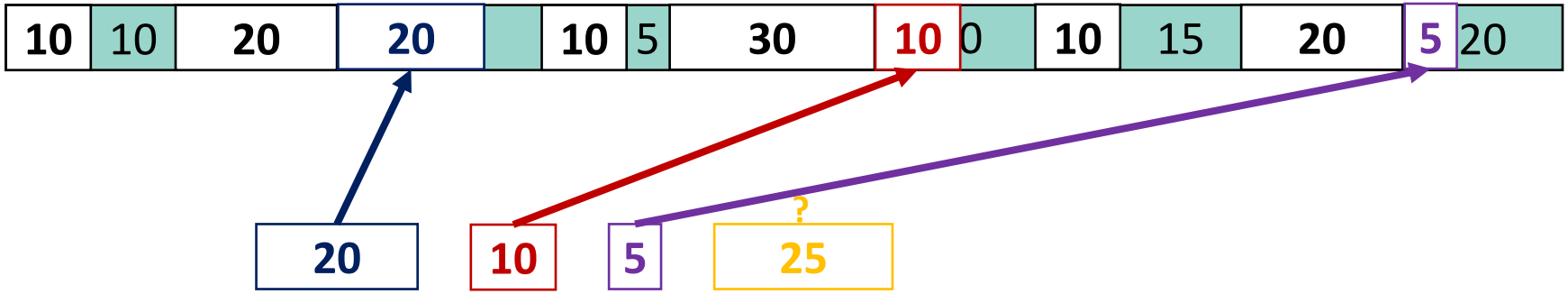
- Soit l'espace mémoire suivant
  - Tailles en Ko



- A quelles adresses sont alloués les blocs avec « **Best fit** » ?

Solution

- Soit l'espace mémoire suivant
  - Tailles en Ko



- A quelles adresses sont alloués les blocs avec « **Worst fit** » ?

Conclusion

- La stratégie va-et-vient
  - Charge un processus en mémoire
  - Copie le processus terminé sur le disque

  - Inconvénient
    - Fragmentation de la mémoire
    - Le volume à transférer peut être très couteux
      - Ex: 10 secondes pour transférer un programme de 1 Go

  - Idée :
    - Les programmes sont divisés en parties appelés **segment de recouvrement**
      - Exécutés les uns après les autres
      - Découpage par les utilisateurs

    - Automatisation : **la mémoire virtuelle**

Conclusion

Et maintenant ?

- **On passe aux exercices !**
  - **Disponibles sur Moodle**

Conclusion

Et maintenant ?

- **La suite à la prochaine séance…**
- Le quizz et le sondage seront proposés à l'issue du chapitre

UNIVERSITÉ HAUTE-ALSACE

ensisa

Systèmes d'exploitation

ENSISA 1A

Chapitre 3

Mémoire