

Chapitre 5

Gestion de la mémoire

① Monoprogrammation

② Multiprogrammation

- Avec partitions fixes
- Avec partitions variables
 - ▶ Algorithmes de sélection de partition

③ La mémoire virtuelle

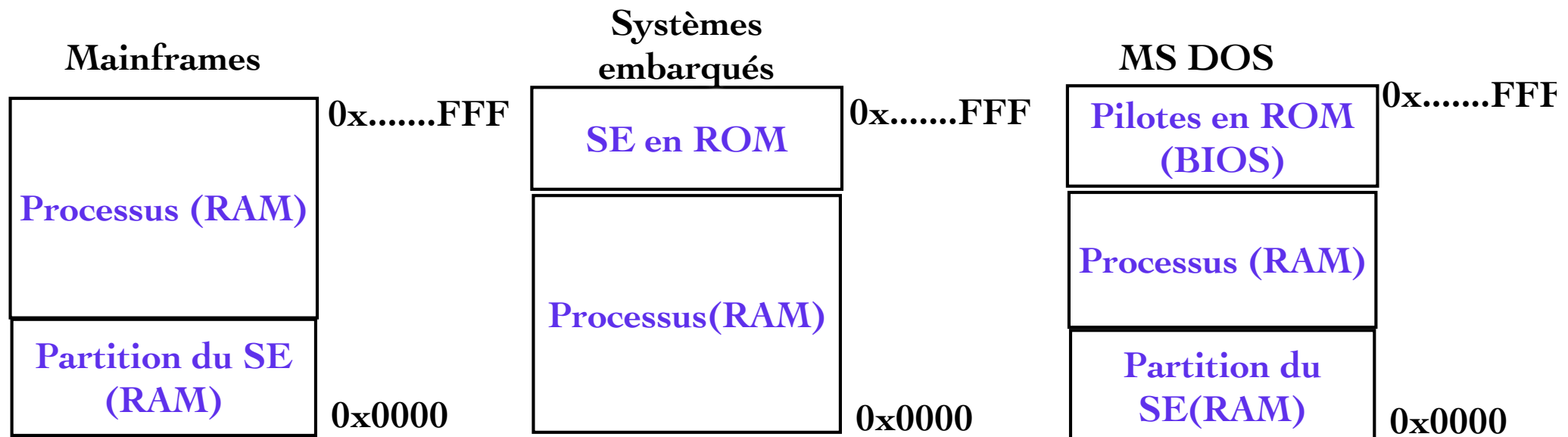
- Pagination
- Segmentation

Introduction

- ▶ **En multiprogrammation:** le SE charge plusieurs processus en RAM
- ▶ **Gestion de la Mémoire (GE):** est du ressort du système de gestion de la mémoire ou gestionnaire de la mémoire qui est parti intégrante du noyau du SE
- ▶ **Fonctionnalités de GE**
 - ▶ Connaître les parties libres de la RAM
 - ▶ Allouer la RAM au processus en évitant autant que possible le gaspillage
 - ▶ Récupérer la mémoire libérée après la fin d'un processus
 - ▶ Offrir aux processus une mémoire virtuelle supérieure à celle de la RAM au moyen des techniques de va-et-vient et de pagination

Monoprogrammation

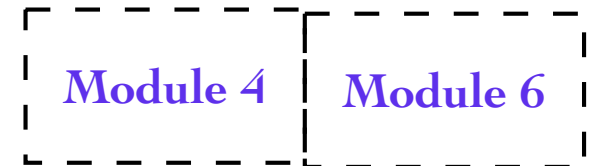
- Pour un système monoprocesseur et **monotâche**
 - Le SE se trouve au niveau des adresses basses de la RAM
 - Pour les système embarqués (consoles de jeux, téléphones mobiles ect.), le SE est dans la ROM



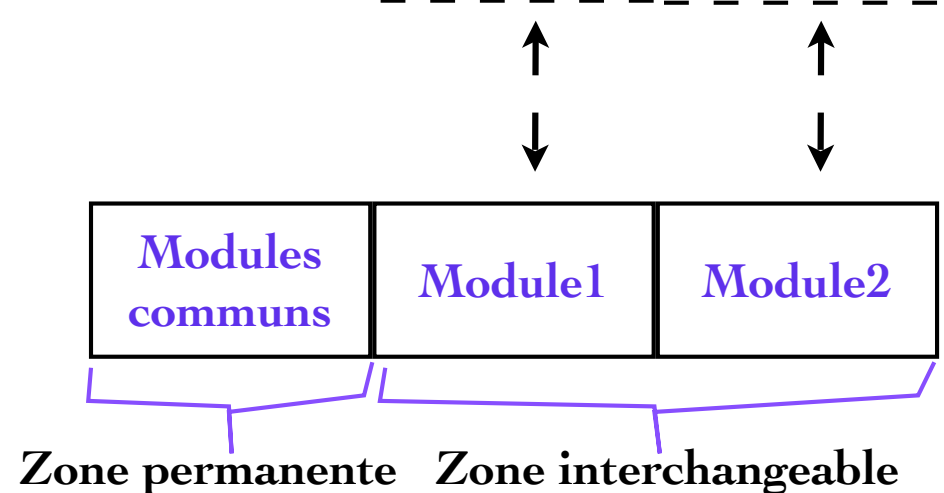
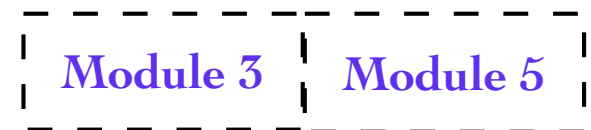
Monoprogrammation

- ▶ Un seul processus en RAM
- ▶ Taille du processus $>$ RAM
- ▶ Utilisation des segments de recouvrements (overlay)
 - ▶ Le programme est composé d'une zone permanente
 - ▶ Un ensemble d'overlays interchangeables correspondant le plus souvent aux grandes fonctionnalités du programme

Overlay 1

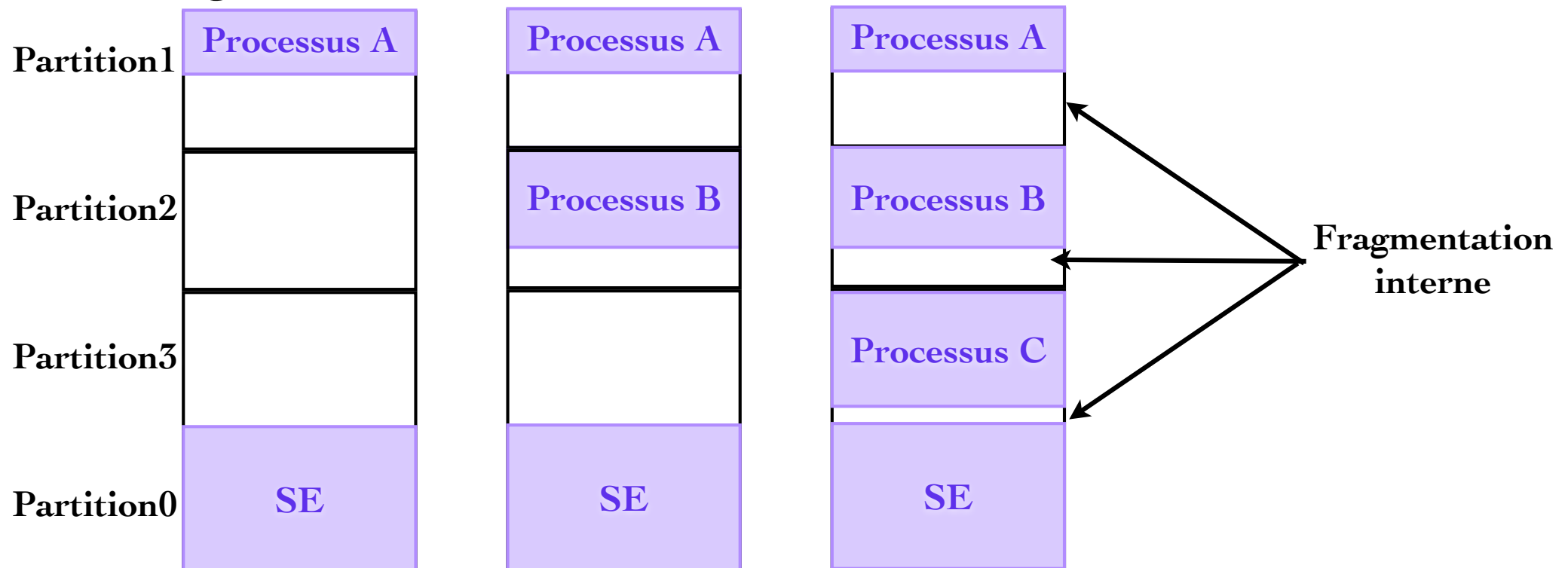


Overlay 2



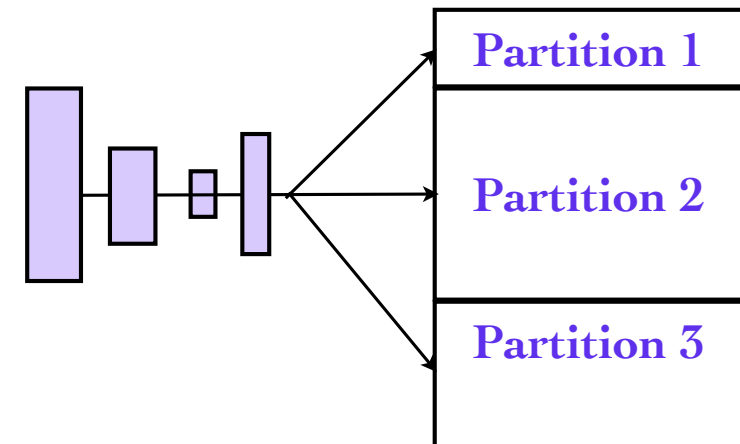
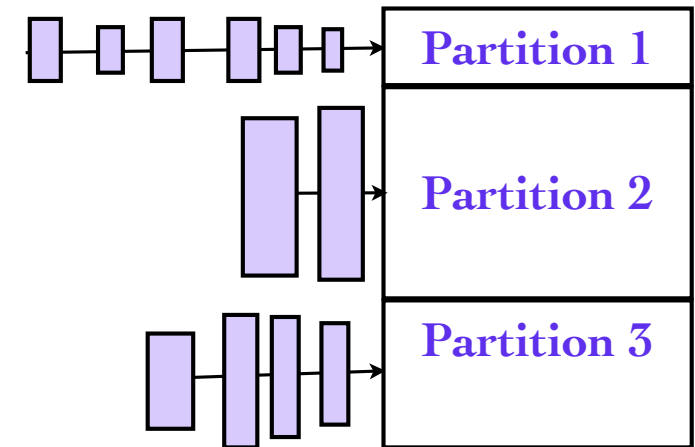
Multiprogrammation avec partitions fixes (1)

- ▶ La RAM divisée en partitions de **taille identique**
- ▶ L'espace non utilisé à la fin d'une partition est perdu:
fragmentation interne



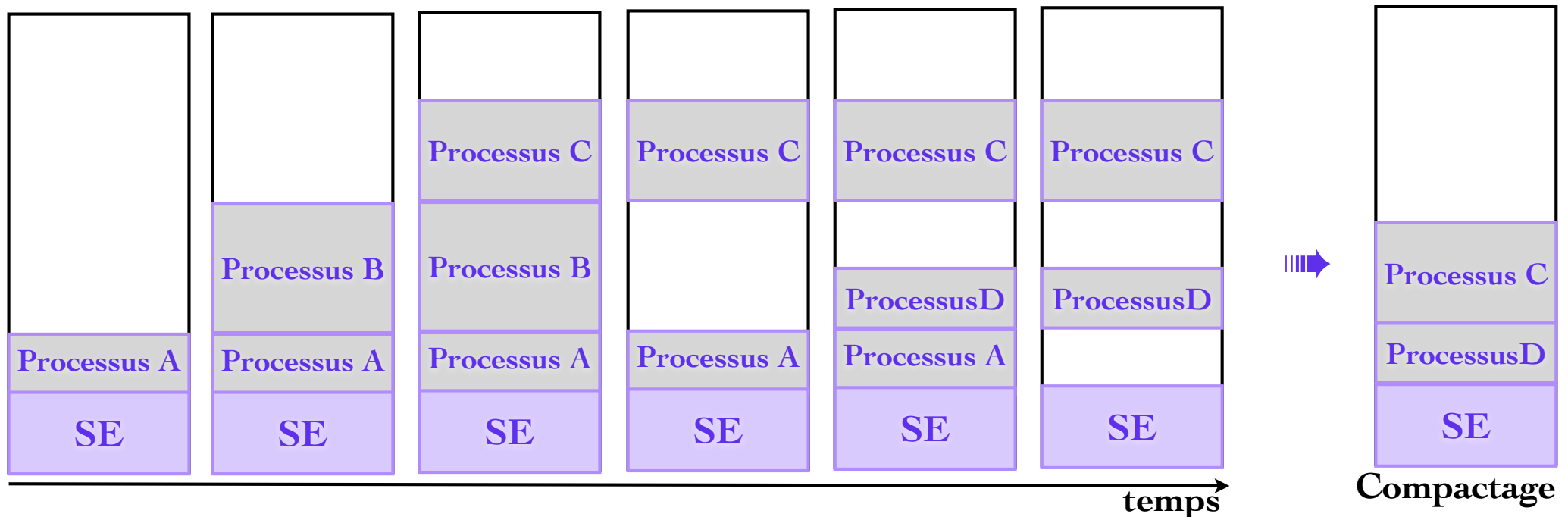
Multiprogrammation avec partitions fixes (2)

- Pour réduire la **fragmentation interne**, La RAM est divisée en partitions de **tailles inégales**
- **File d'attente pour chaque partition**
 - Un nouveau processus est placé dans la file d'attente de la plus petite partition pouvant le contenir
 - Problème possible: grande partition vide, petite partition pleine
- **File unique,**
 - Dès qu'une partition se libère, on y place le plus grand processus de la file d'attente qui peut y tenir
 - Désavantage les petites tâches
 - D'autres algorithmes cherchent la première tâche pouvant se tenir dans la partition



Multiprogrammation avec partitions variables

- ▶ Le SE alloue une zone mémoire à un processus selon son besoin
- ▶ Problème possible: **fragmentation externe**
- ▶ **Compactage** pour rassembler les zones libérées avec l'espace non alloué: surcharge due aux déplacements des processus
- ▶ Gérer la mémoire libre par des **mécanismes** et **algorithmes**



Mécanismes et algorithmes de gestion des zones libres

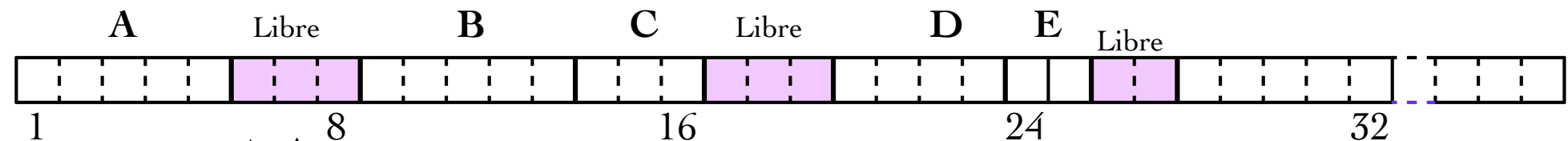
- ▶ Lorsque plusieurs zones mémoires sont libres, le SE doit:
 - ▶ Utiliser des mécanismes pour identifier les zones libres
 - ▶ Gestion par tables de bits
 - ▶ Gestion par listes chaînées
 - ▶ Gestion par subdivision
 - ▶ Utiliser un algorithme pour sélectionner la zone mémoire dans laquelle le processus sera chargé
 - ▶ Première zone libre (First Fit)
 - ▶ Zone libre suivant (Next Fit)
 - ▶ Meilleure ajustement (Best Fit)
 - ▶ Plus grand résidu (Worst Fit)

Mécanismes et algorithmes de gestion des zones libres

► Gestion par tables de bits

► Division de la mémoire en unités

► On fait correspondre chaque unité dans une table de bits



1	1	1	1	1	0	0	0
1	1	1	1	1	1	1	1
0	0	0	1	1	1	1	1
1	0	0	1	1	1	1	1

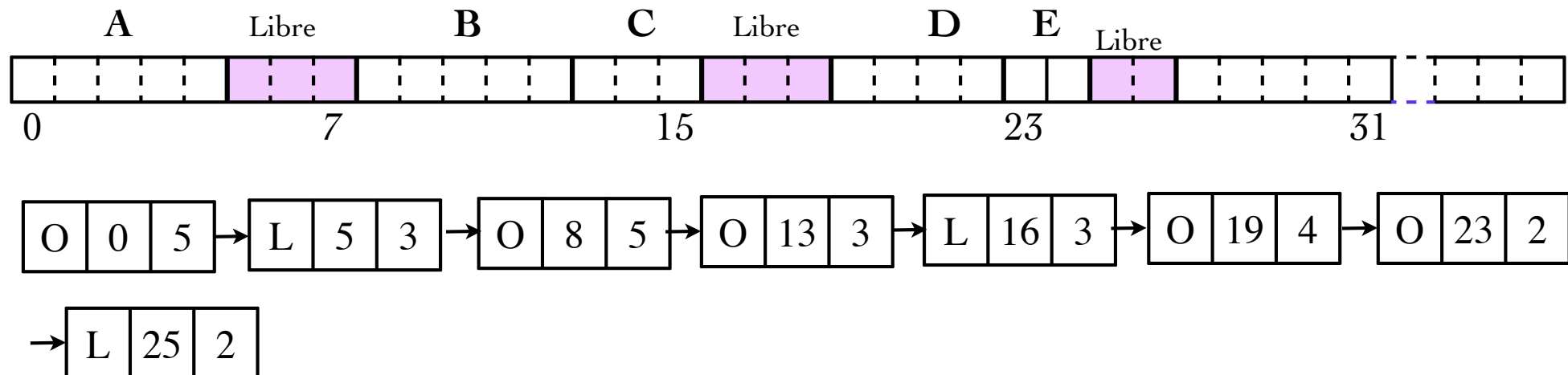
Table de bits

► Table de bits: 0=libre, 1=occupée

► Pour allouer k unités contigües selon l'algorithme (FF, BF, WF, NF), le SE doit parcourir la table de bits à la recherche de k zéros consécutifs.

Mécanismes et algorithmes de gestion des zones libres

- ▶ Gestion par listes chaînées
- ▶ Chaque segment est représenté par:
 - ▶ Bit d'état occupé (O), Libre (L),
 - ▶ Son adresse de début et sa taille,



Mécanismes et algorithmes de gestion des zones libres

► Gestion par subdivision

- La subdivision (buddy system ou allocation des copains) propose un compromis entre l'allocation de taille fixe et celle de taille variable.
- Le SE ne manipule que des blocs de mémoire dont la taille est égale à une puissance de 2.
- Initialement, il existe un seul blocs dont la taille est égale à celle de la mémoire
- Un processus reçoit une unité de mémoire qui est la plus petite puissance de 2 supérieure à sa taille

État initial	1024			
Alloue A=70	A	128	256	512
Alloue B=35	A	B	64	256
Alloue C=200	A	B	64	C
Libère A	128	B	64	C
Alloue D=60	128	B	D	C
Libère B	128	64	D	C
Libère D	256		C	
Libère C	1024			

Algorithmes de sélection des partitions

- ▶ **Algorithme de la première zone libre (First Fit)**
 - ▶ Le processus est alloué à la première zone mémoire libre rencontrée et suffisamment grand pour le contenir.
 - ▶ Favorise les zones en début de mémoire
- ▶ **Algorithme de la zone libre suivante (Next Fit)**
 - ▶ Cette algorithme se veut équitable en commençant la recherche de la zone à allouer partir de la zone dernièrement allouée et non en début de mémoire
 - ▶ Aucune zone libre n'est favorisée

Algorithmes de sélection des partitions

- ▶ **Algorithme du meilleur ajustement (Best Fit)**
 - ▶ Le processus est alloué à la plus petite zone libre suffisamment grand pour le contenir.
 - ▶ Nécessité de parcourir intégralement les zones libres de la mémoire
 - ▶ Etablit une correspondance presque parfaite entre la zone mémoire et la taille du processus
 - ▶ La partie restante de la zone allouée est trop petite, quasiment inutile.
- ▶ **Algorithme du plus grand résidu (Worst Fit)**
 - ▶ Evite la création de zones petites et inutiles en choisissant la plus grande zone restante
 - ▶ Nécessité de parcourir intégralement les zones libres de la mémoire

Pagination

- ▶ La pagination est l'allocation de zones de mémoire non contigües pour un même processus
- ▶ La mémoire physique est découpée en zones de **taille fixe: cadres de page**
- ▶ Les processus divisés en blocs: **pages** (de même taille que les cadres)
- ▶ La taille d'une page est une puissance de 2
- ▶ **Adresse logique:** numéro de page || déplacement dans la page
- ▶ **Table de pages:** liaison entre numéro de page et le cadre de page (une table par processus)
 - ▶ Le SE stocke la table de pages ou de segments en mémoire
 - ▶ Sur certains systèmes, un registre de gestion de mémoire pointe vers la table de page ou de segment du processus élu

Pagination

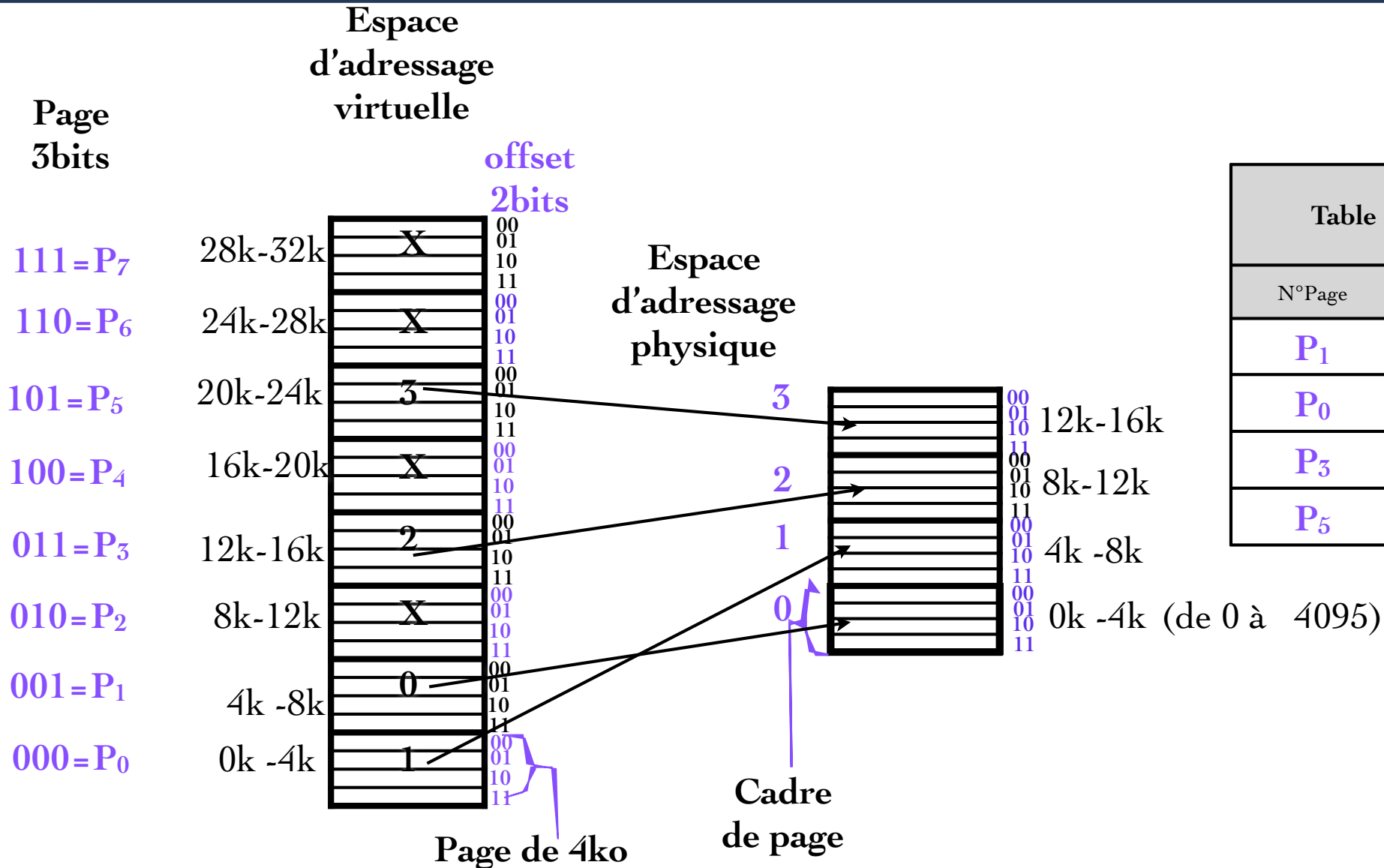
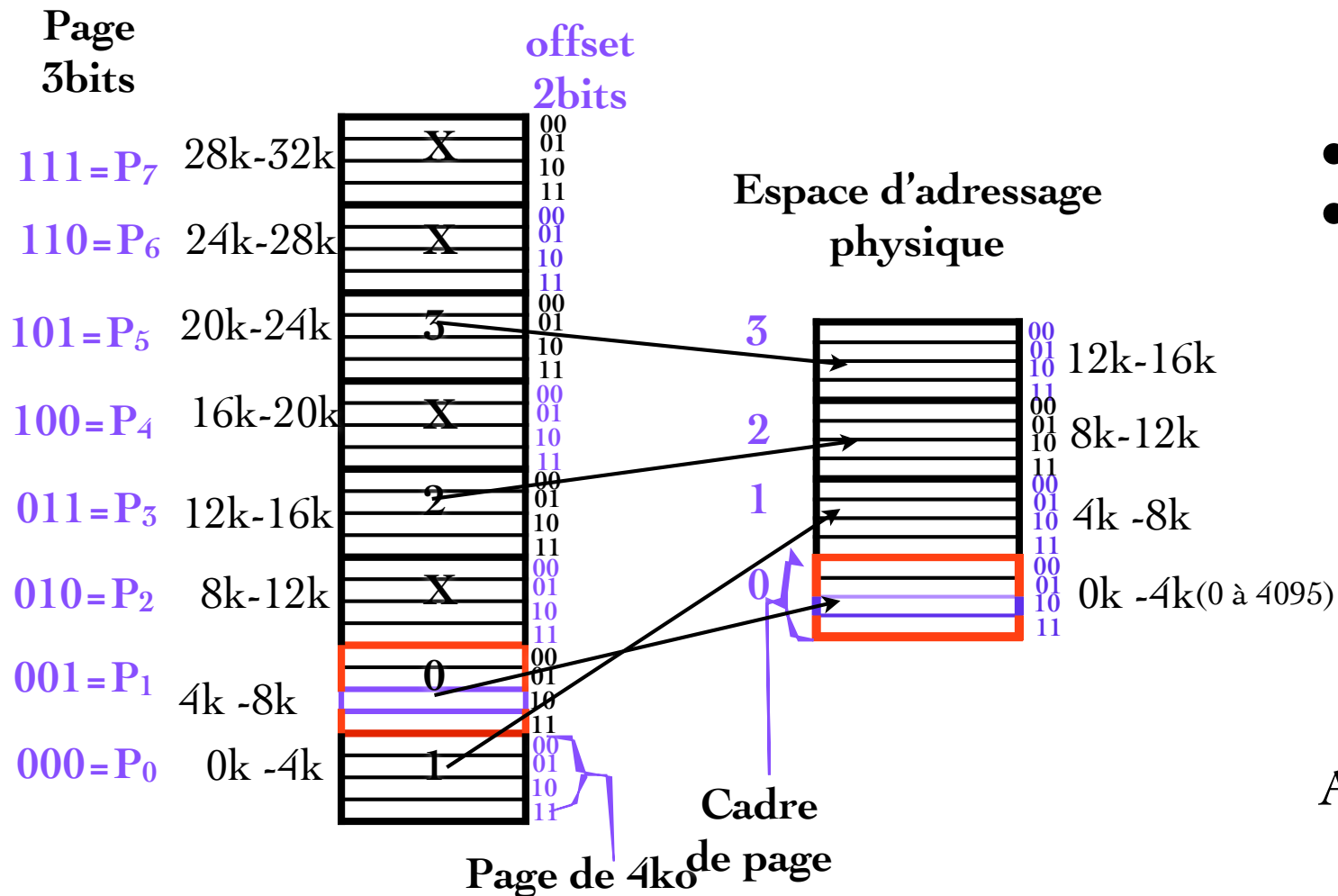


Table de Page	
N°Page	N°Cadre Page
P ₁	0
P ₀	1
P ₃	2
P ₅	3

Pagination: Exemple

NB: L'adressage est faite par Ko au lieu d'octet
Par octet, on aurait besoin de 12 bits pour coder l'offset (4096)



Adresse logique

- Numéro page=001
- Déplacement=10

↓ MMU

Adresse physique:

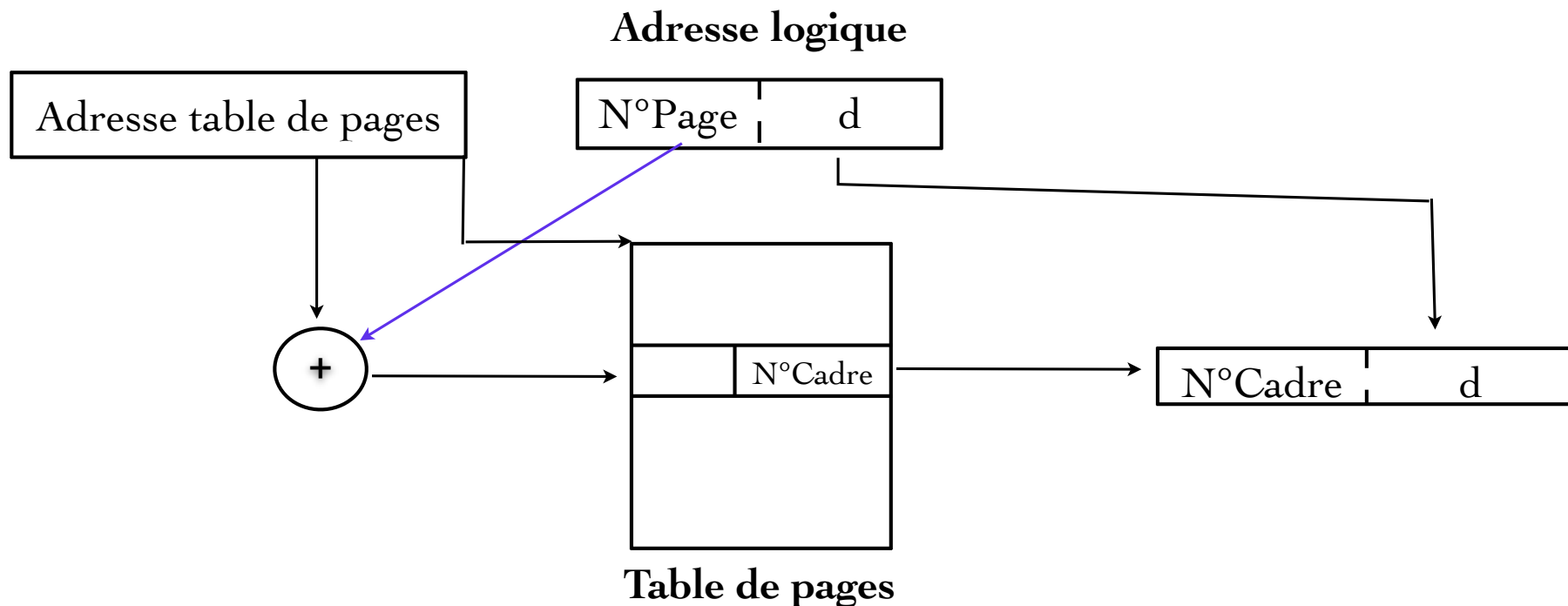
- Numéro cadre=00
- Déplacement=10

↓

AL(00110) = AP(0010)

Pagination

- Pour chaque processus, une **table de pages** stocke le numéro du cadre de page alloué à chaque page
- L'adresse physique est générée en concaténant le déplacement de la page au numéro du cadre de page extrait de la table de pages

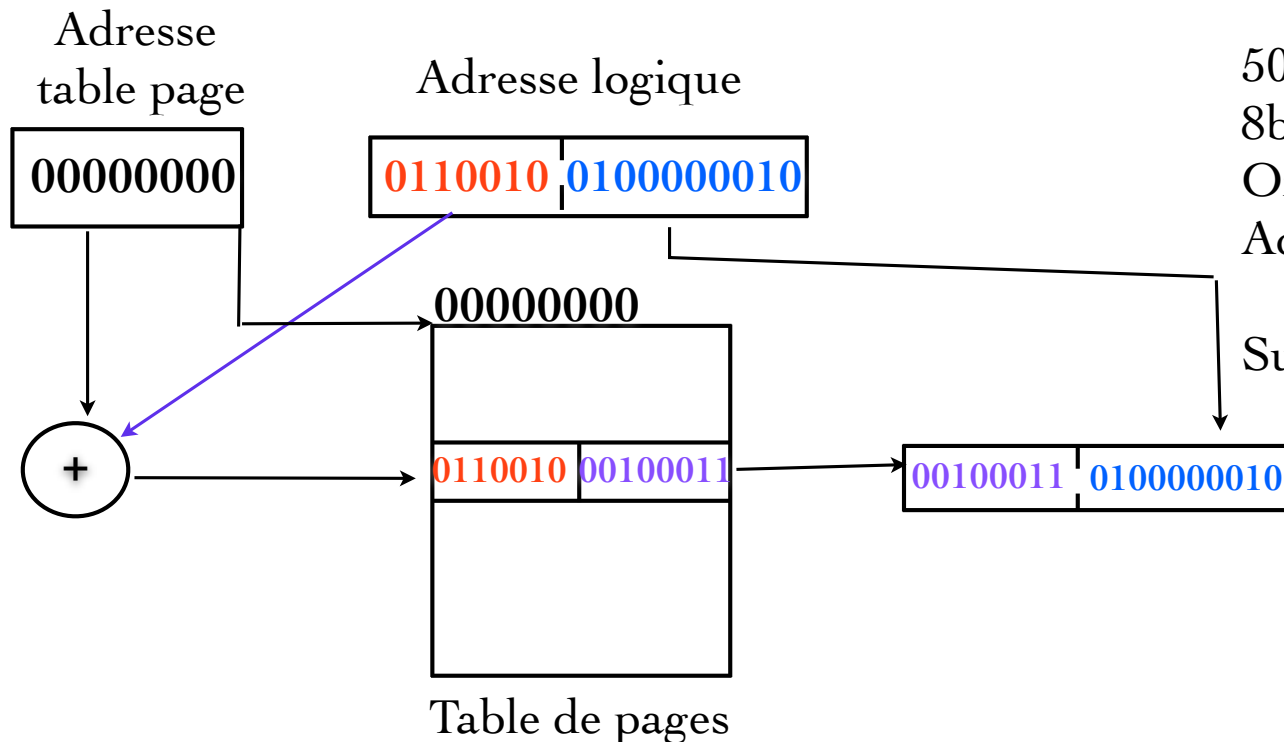


Exemple (1)

- ▶ Soit un système de 2^{18} octets de mémoire physique, 128 pages d'espace d'adressage logique et une taille de page 2^{10} octets
- ▶ Une variable globale est décalée de 258 de la 50ème page logique chargée dans le 35ème cadre
- Nombre de bits pour une adresse logique est 17bits
 - ▶ 7bits pour le numéro de page ($128\text{pages}=2^7$)
 - ▶ 10 bits pour l'offset
- Nombre de bits pour une adresse physique est 18bits
 - ▶ Taille d'un cadre de page=taille de page, Donc **10 bits** pour l'offset
 - ▶ Chaque entrée de la table de page contient $2^{18} \div 2^{10} = \mathbf{8\text{bits}}$ pour spécifier le numéro de cadre de page
 - ▶ Le processus a 128 pages, seules les 128 premières entrées de la table de page sont définies comme valides

Exemple (2)

- Soit un système de 2^{18} octets de mémoire physique, 128 pages d'espace d'adressage logique et une taille de page 2^{10} octets
- Une variable globale est décalée de 258 de la 50^{ème} page logique (chargée dans 35^{ème} cadre)



Adresse logique

50^{ème} page (sur 7bits) = 0110010

Offset (sur 10bits) = 258 = 0100000010

Adresse physique = 01100100100000010

Adresse Physique

50^{ème} page = 35^{ème} cadre (sur 8bits) = 00100011

Offset (sur 10bits) = 258 = 0100000010

Adresse physique = 001000110100000010

Adresse table de page

Supposé être 00000000

Défaut de pages

- ▶ Le défaut de pages est l'accès à une page non présente en mémoire physique
- ▶ **Dans ce cas**
 - ▶ Le SE choisit un cadre de page
 - ▶ Écrit si nécessaire son contenu en mémoire virtuelle
 - ▶ Charge en mémoire la page qui faisait défaut
 - ▶ Il faut un algorithme pour le choix de la page à décharger

Algorithmes de remplacement des pages

► FIFO

- La page la plus ancienne sera supprimée en premier
- Problème possible: elle peut être la plus utilisée
- Le FIFO est rarement utilisé pour le remplacement de pages

Algorithmes de remplacement des pages

- ▶ **LRU: (Least Recently Used)**
 - ▶ La page la moins récemment utilisée sera déchargée
 - ▶ Evidemment celles les plus utilisées durant les dernières instructions seront certainement les plus demandées dans le futur

Algorithmes de remplacement des pages

- ▶ **Non récemment utilisée: NRU (Not Recently Used)**
- ▶ L'algorithme choisit au hasard une page de classe basse
 - ▶ Chaque page peut être classée
 - ▶ Classe 0: non lue, non écrite
 - ▶ Classe 1: non lue, écrite
 - ▶ Classe 2: lue, non écrite
 - ▶ Classe 3: lue, écrite

Algorithmes de remplacement des pages

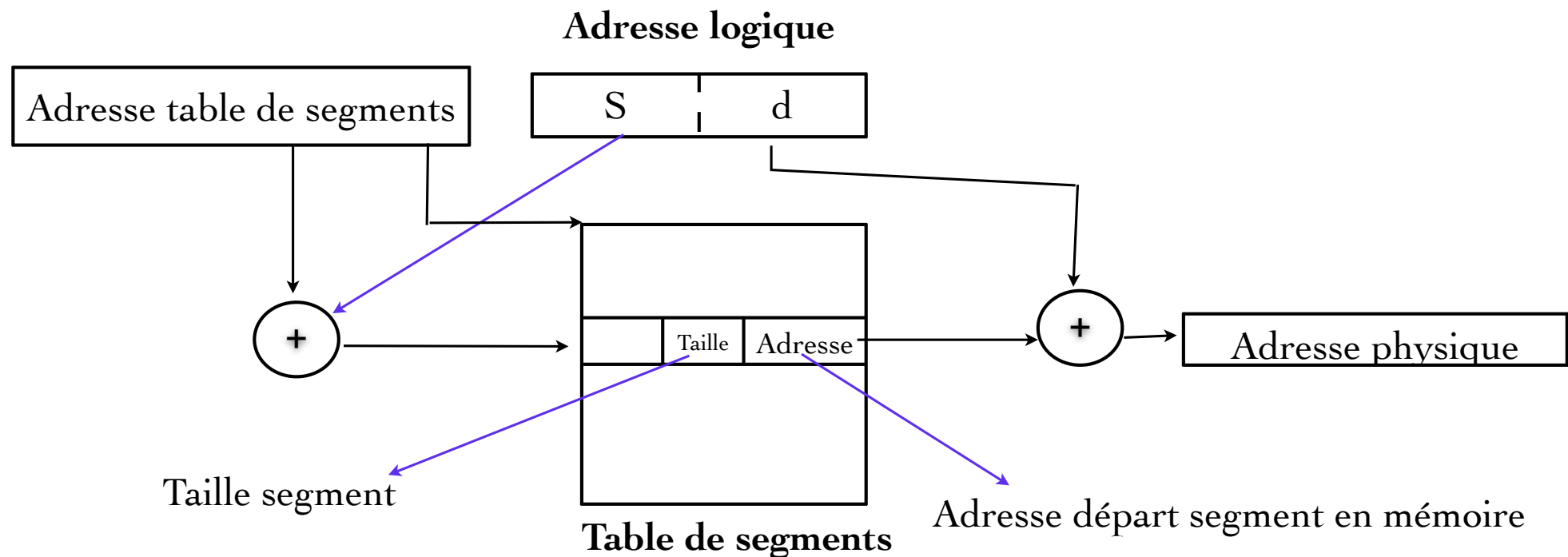
- ▶ **LRU: (Least Frequently Used)**
 - ▶ La page la moins fréquemment utilisée sera déchargée
 - ▶ Associe un compteur à chaque page
 - ▶ A chaque interruption, on ajoute le compteur
 - ▶ Celle avec le plus petit compteur est remplacée

Segmentation

- ▶ La segmentation divise le programme en blocs plus petits appelés **segments** et alloués indépendamment à la mémoire.
- ▶ Contrairement à la pagination, les segments sont de **taille variable**
- ▶ Le SE conserve une **liste de zones de mémoire libres** à allouer aux segments
- ▶ **Adresse logique**: numéro de segment + déplacement dans le segment
- ▶ **Table de segments** contient:
 - ▶ N° segment comme indexe
 - ▶ L'adresse de départ du segment en mémoire RAM
 - ▶ La taille du segment
- ▶ **Taille des segments**: puissance de 2

Segmentation

- Pour chaque processus, une **table de segments** stocke l'index du segment (numéro), son adresse départ, sa taille
- L'adresse physique est générée en ajoutant le déplacement à l'adresse de départ du segment extrait de la table de segments

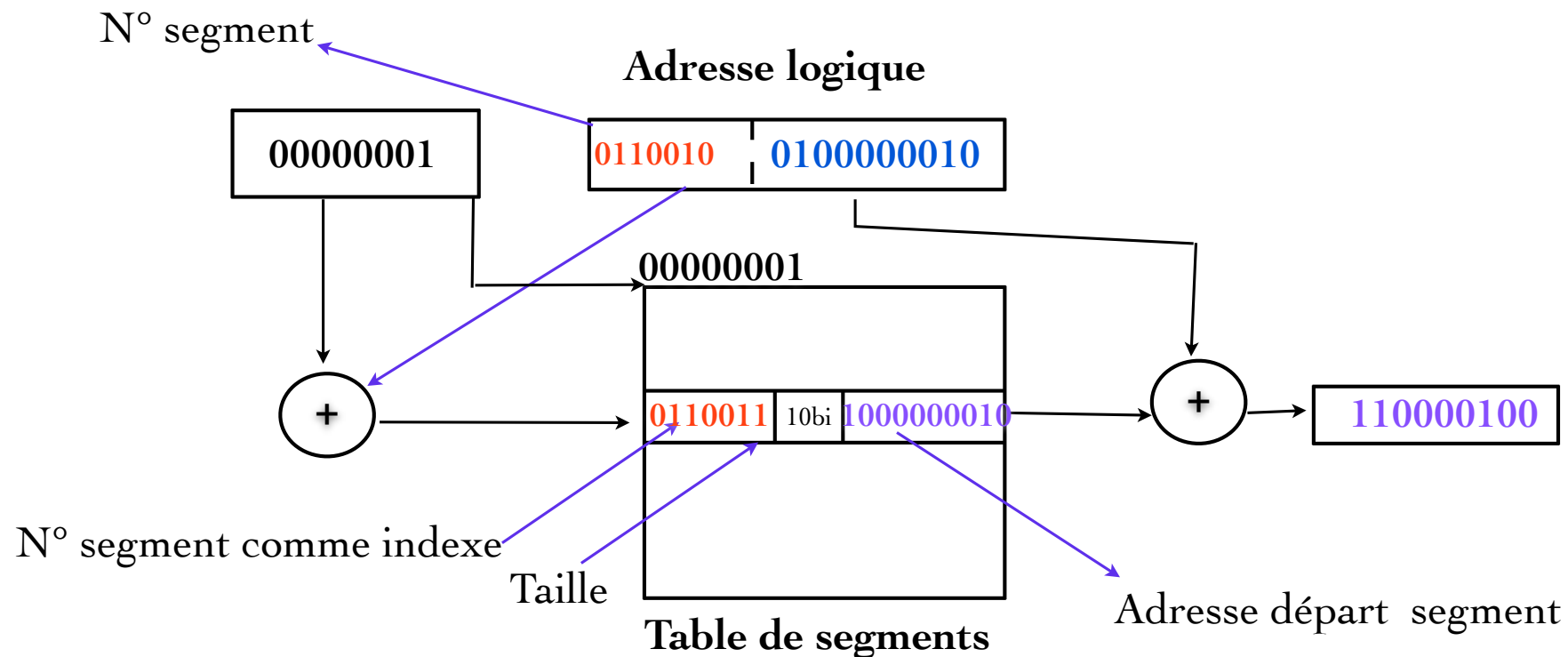


Exemple (1)

- ▶ Soit un système de segmentation simple, 2^{18} octets de mémoire physique, 128 segments d'espace d'adressage logique
- ▶ Une variable globale X est décalée de 258 du 50ème segment dont la taille est 2^{10} octets et adresse de départ=514
- ▶ La taille de segment maximale est de 2^{10} octets
- Nombre de bits pour une adresse logique est 17bits
 - ▶ 7bits pour le numéro de segment (128 segments= 2^7)
 - ▶ 10 bits pour l'offset au sein de ce segment, peut être nbits pour un autre
- Chaque entrée de la table de segment contient une adresse de départ d'un segment et sa taille
 - ▶ Si l'adresse départ segment= 514=1000000010
 - ▶ Taille= 2^{10} , décalage=258=0100000010
 - ▶ Adresse table de segment supposé être 00000001

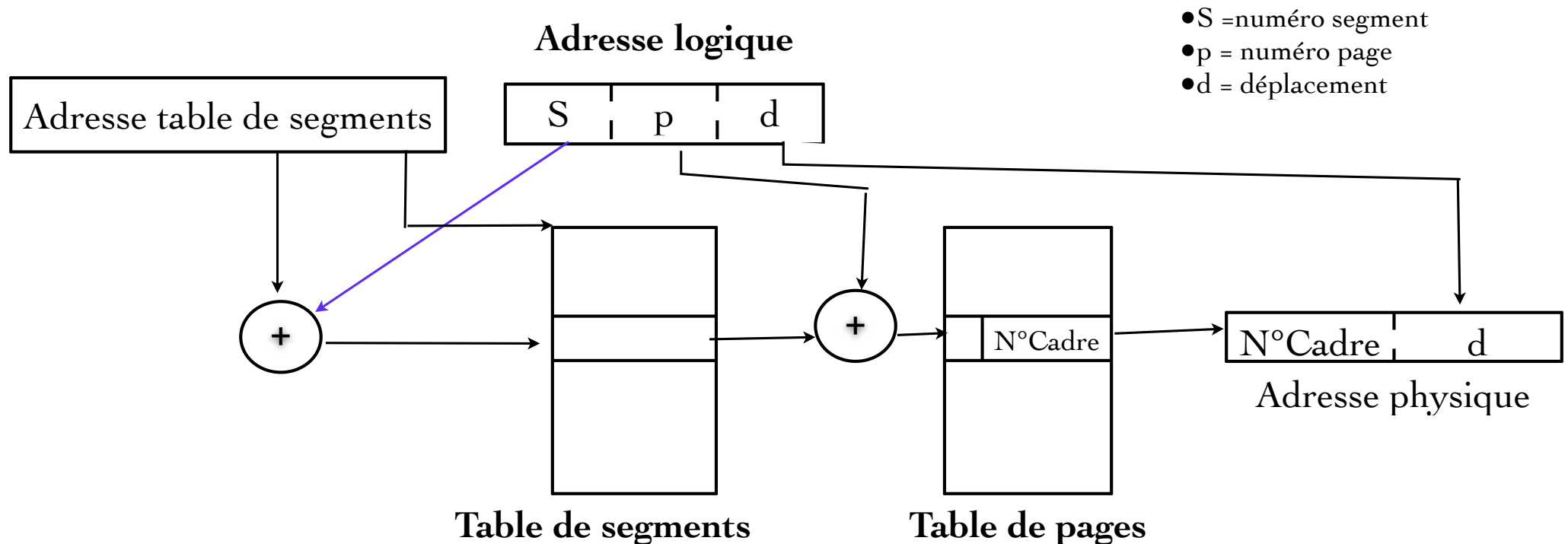
Segmentation

- Soit un système de segmentation simple, 2^{18} octets de mémoire physique, 128 segments d'espace d'adressage logique
- Une variable globale X est décalée de 258 dans le 50^{ème} segment dont la taille est 2^{10} octets et adresse de départ=514



Segmentation avec pagination

- Pour chaque processus, une **table de segments** stocke l'index du segment (numéro), son adresse départ, sa taille
- L'adresse physique est générée en ajoutant le déplacement au numéro de cadre de page extrait de la table de pages.



Segmentation avec pagination

- Soit un système de segmentation avec pagination a 2^{14} octets de mémoire physique, 8 segments de taille maximum 2^{14} octets. Chaque segment est paginé (page de taille 256 octets)
- Une variable globale X est décalée de 50 dans la 20^{ème} page (chargée dans 15^{ème} cadre) du segment 5 dont la taille est 2^{14} .
- L'adressage est faite par octets

NB: C'est la table de pages qui est présent en mémoire et non la table de segment, c'est pourquoi l'adresse physique dépend uniquement du **numéro de cadre** concaténé avec **le déplacement** alors que l'adresse logique est fonction du **numéro de segment**, **numéro de page**, du **déplacement**

