Système d'Exploitation

Sous-système de Gestion Mémoire

Université de Tours
Faculté des Sciences et Techniques
Antenne Universitaire de Blois

Licence Sciences et Technologies
Mention : Informatique
2ème Année

Mohamed TAGHELIT
taghelit@univ-tours.fr

Plan

1. Sous-système de Gestion de Mémoire
2. Partitionnement fixe et dynamique
3. Pagination
4. Segmentation
5. Principes de la Mémoire Virtuelle
6. Mémoire Virtuelle Paginée
7. Mémoire Virtuelle Segmentée
8. Algorithmes de Remplacement de Pages

Exigences de la Gestion Mémoire

- Délocalisation
  - Les programmeurs ne savent pas où le programme sera placé en mémoire (possibilité de swap)
  - Nécessité de convertir les références mémoires en adresses physiques
- Protection
  - Pas d'interférences entre processus (pas de références dans l'espace d'adressage des autres processus sans autorisation)
  - Impossibilité de vérifier les adresses absolues lors de la compilation (doit-être faite lors de l'exécution)
- Partage
  - Permettre à plusieurs processus de partager des portions de mémoire (code, segments partagés)
- Organisation Logique
  - Organisations mémoire (linéaire) et programme (modules) différentes
  - Écriture et compilation des modules séparées (références résolue lors de l'exécution), protection et partage des modules → segmentation
- Organisation Physique
  - Hiérarchie mémoire (centrale (volatile) et secondaire (non volatile))
  - Les flux ne peuvent être de la responsabilité des programmeurs → responsabilité du gestionnaire mémoire

Partitionnement

- Partitionnement fixe
- Partitionnement dynamique
- Pagination
- Segmentation

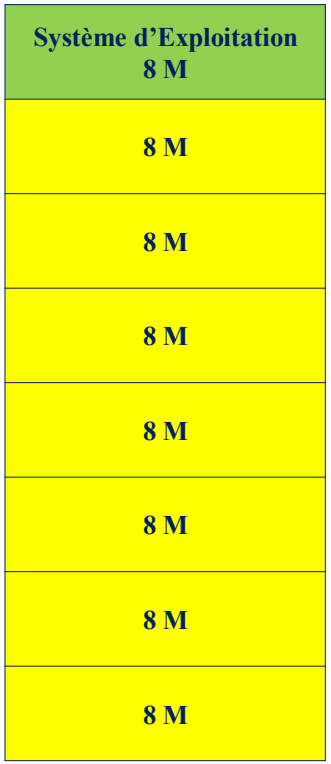
Partitionnement Fixe

Tailles identiques

- Partitions de taille fixe
  - Taille processus ≤ taille partition
  - Placement trivial
  - Chargement du processus dans une partition libre
  - Possibilité de swapper un processus

- Inconvénients
  - Taille processus > taille partition (overlays)
  - Inefficacité de l'utilisation mémoire (fragmentation interne)
  - Limitation du nombre de processus actifs

Mémoire Physique

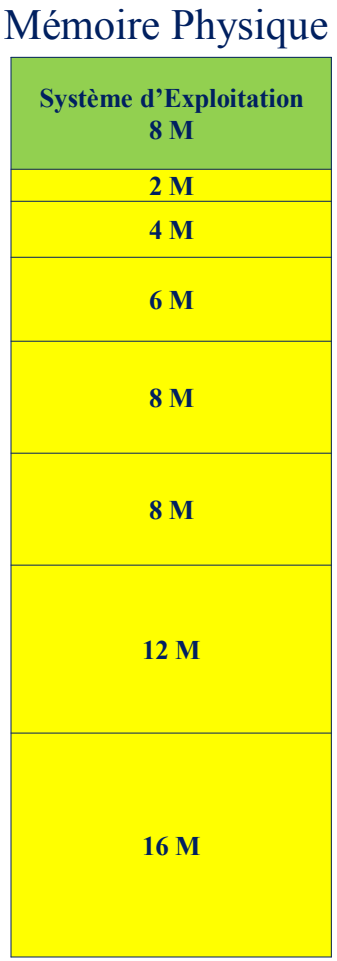


Partitionnement Fixe

Tailles Différentes

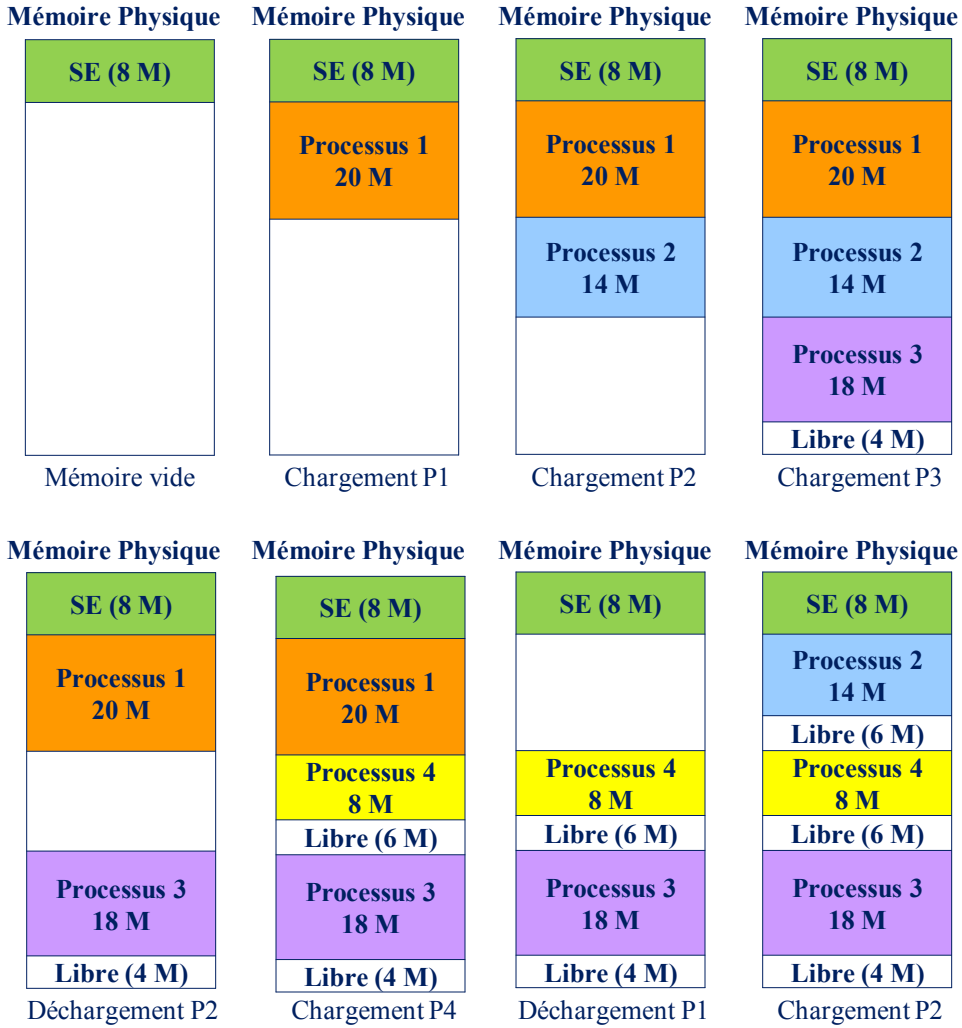
- Résoud en partie les problèmes du partitionnement précédent
  - Processus jusqu'à 16 M sans overlay
  - Placement des petits processus dans les petites partitions (réduction de la fragmentation interne)

- Inconvénients
  - Placement moins trivial (files individuelles ou une commune)
  - Limitation du nombre de processus actifs
  - L'utilisation mémoire reste inefficace (dans les 2 cas)



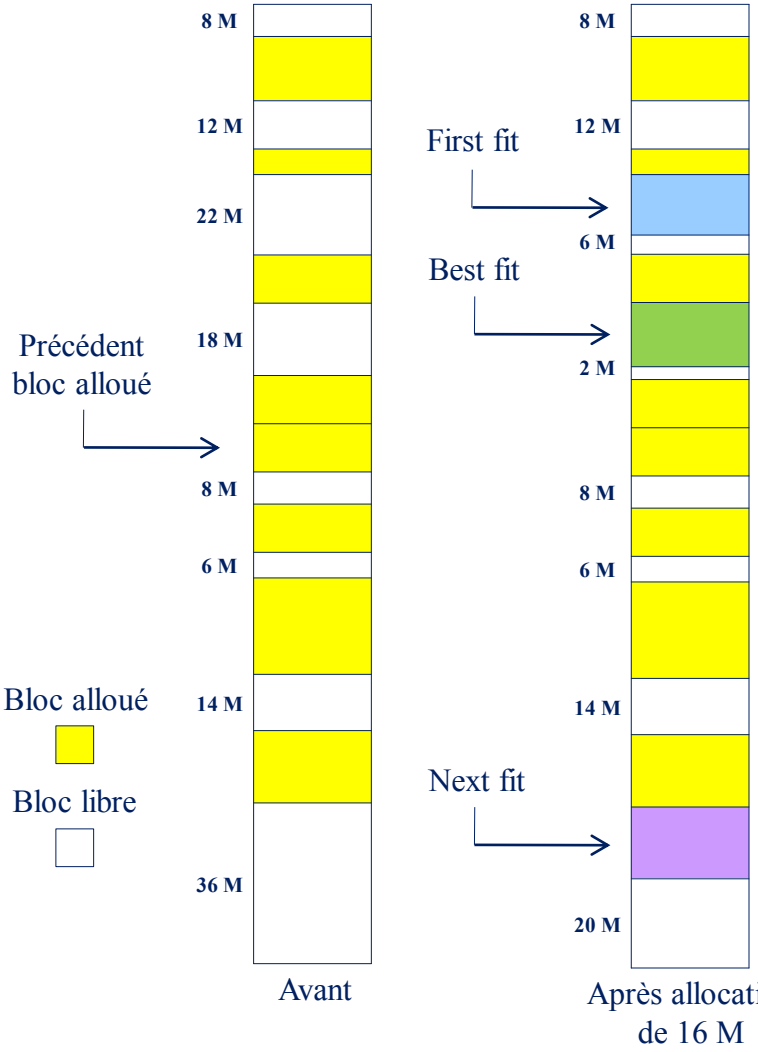
Partitionnement Dynamique

- Partitions en nombre et en tailles variables
  - Allouer au processus la quantité mémoire requise exactement
  - Placement des petits processus dans les petites partitions (réduction de la fragmentation interne)
- Inconvénients
  - Complique l'allocation et la libération
  - Fragmentation de la mémoire non utilisée (fragmentation externe)
  - Nécessite un compactage (consommation de temps et perte de temps CPU)
- Exemple (RAM de 64 M)
  - Processus 1, 20 M
  - Processus 2, 14 M
  - Processus 3, 18 M
  - Processus 4, 8 M



Algorithmes de Placement

- Best-fit
  - Choisir la partition la plus proche en taille de la demande
  - La plus petite partition produira la plus petite fragmentation
  - Compactage fréquent
- First-fit
  - Scruter la mémoire à partir du début et choisir la 1ère partition libre pouvant contenir le processus
  - Rapide
- Next-fit
  - Scruter la mémoire à partir de l'emplacement du dernier placement
  - Place le plus souvent vers la fin de la mémoire
  - Les grandes partitions sont scindées en partitions plus petites
  - Compactage nécessaire pour obtenir une grande partition à la fin de la mémoire



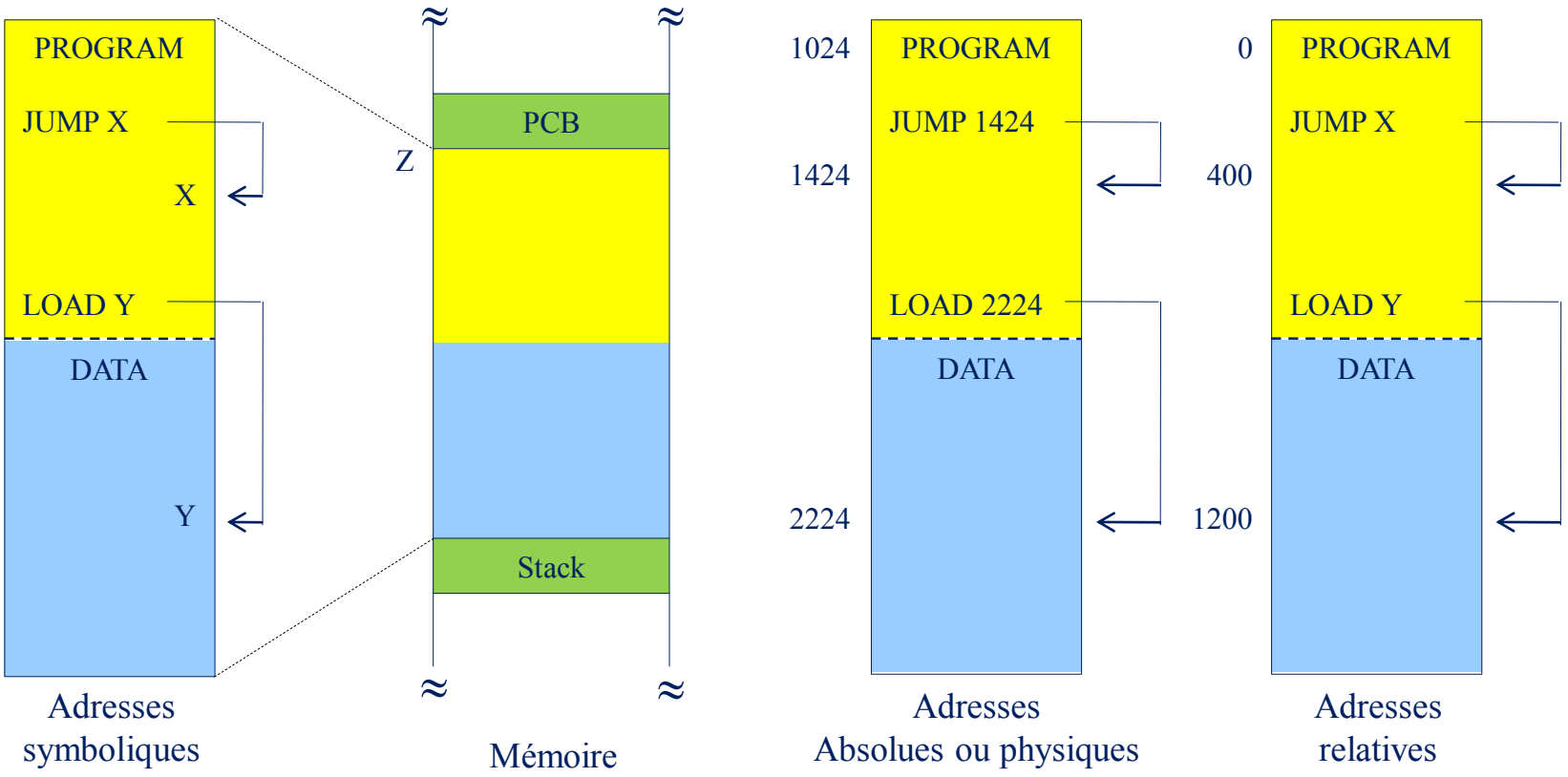
Système “Buddy”

- La totalité de la mémoire est considérée comme une seule partition de taille 2^U
- Si processus de taille S tel que $2^{U-1} < S <= 2^U$ Alors allocation de la totalité (2^U)
  Sinon scinder la partition en 2 partitions de taille égale et recommencer

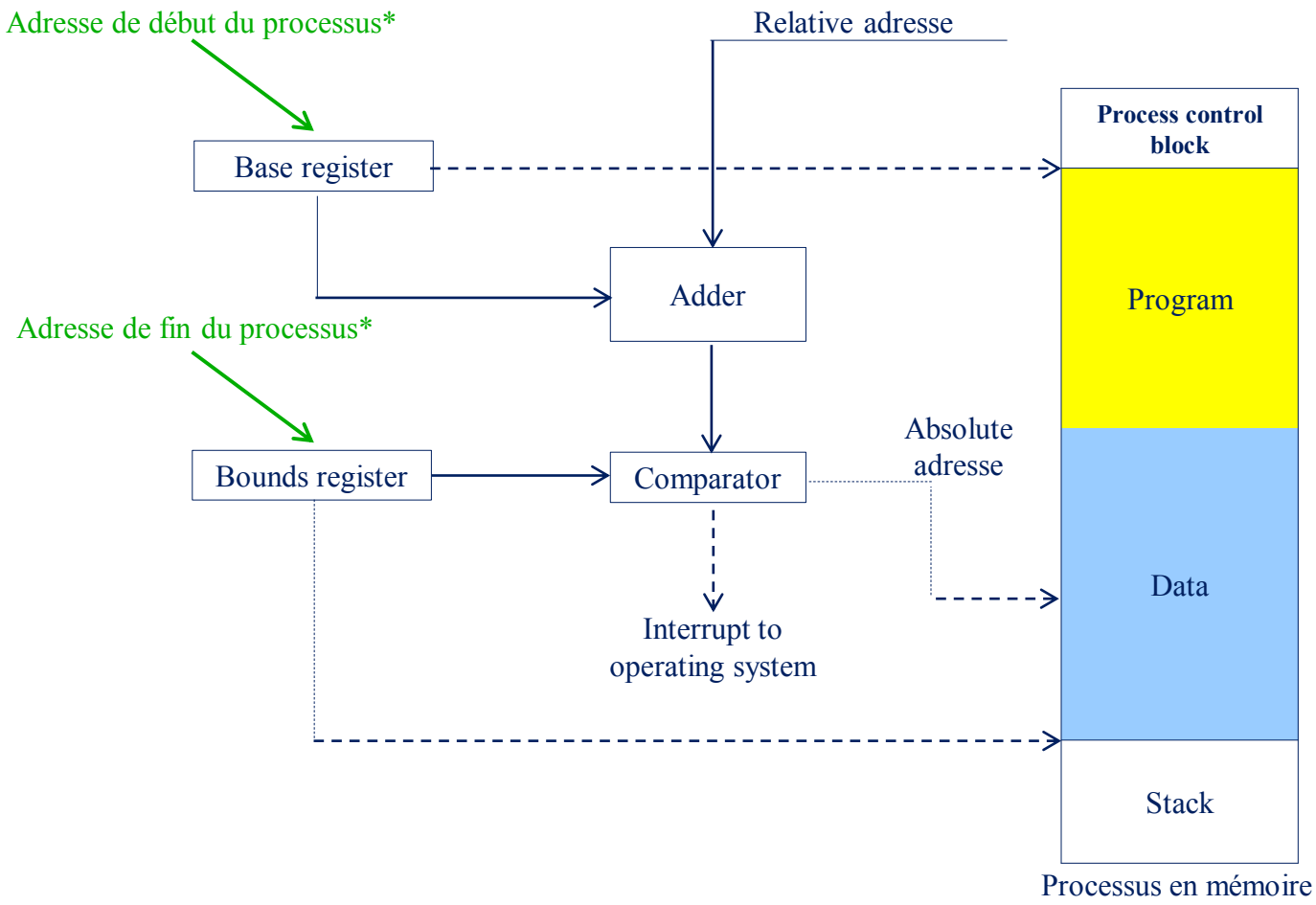
Bloc de 1 M	1 M					
Requête de 100 K	A = 128 K	128 K	256 M	512 M		
Requête de 240 K	A = 128 K	128 K	B = 256 K	512 M		
Requête de 64 K	A = 128 K	C=64K	64 K	B = 256 K	512 M	
Requête de 256 K	A = 128 K	C=64K	64 K	B = 256 K	D = 256 K	256 K
Libération de B	A = 128 K	C=64K	64 K	256 K	D = 256 K	256 K
Libération de A	128 K	C=64K	64 K	256 K	D = 256 K	256 K
Requête de 75 K	E = 128 K	C=64K	64 K	256 K	D = 256 K	256 K
Libération de C	E = 128 K	128 K	256 K	D = 256 K	256 K	
Libération de E	512 K			D = 256 K	256 K	
Libération de D	1 M					

Adressage

- Le swap et le compactage impliquent des emplacements différents en mémoire des processus lors de leur vie.



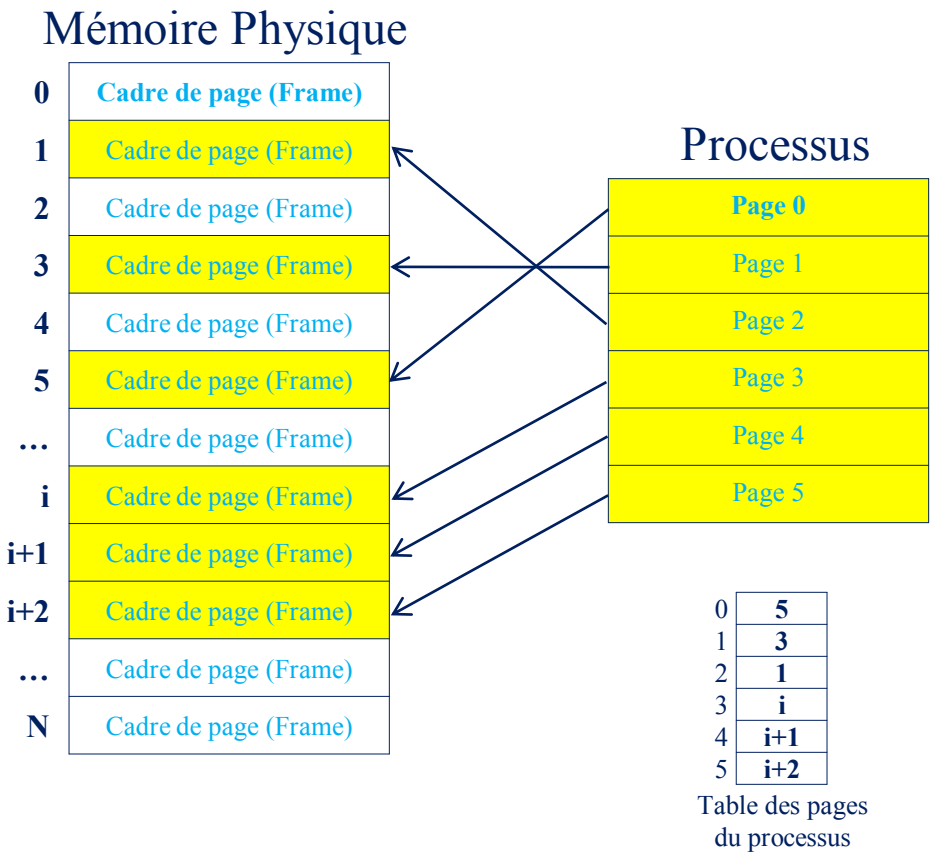
Translation d'adresses



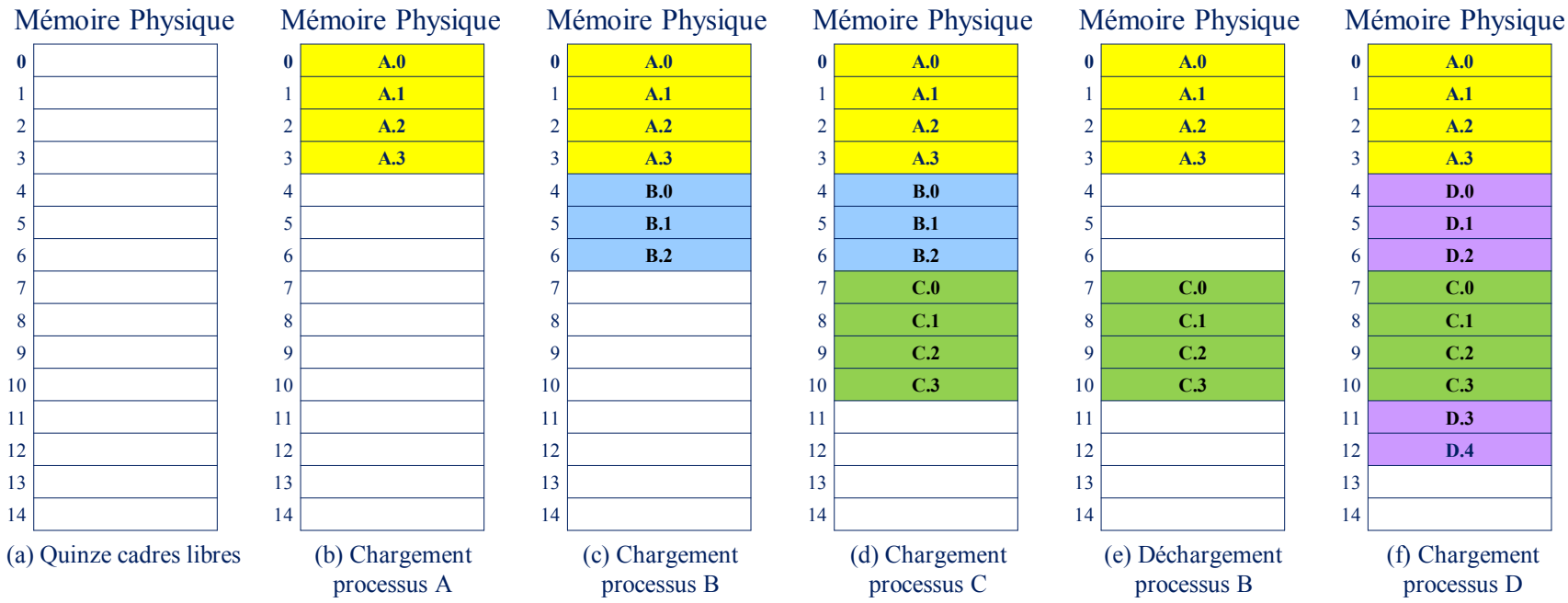
*Valeurs établies lors du chargement (ou swap en mémoire centrale) du processus

Pagination - Notion de Base

- Mémoire découpée en morceaux de petite taille fixe (cadres de page = *frames*)
- Processus découpés en morceaux de même taille (pages)
- Processus chargé en mémoire
  - Toutes les pages du processus sont chargées dans des cadres de pages disponibles (non nécessairement contigus)
  - Une table des pages du processus est établie
- Espace perdu en mémoire
  - Partie de la dernière page du processus (fragmentation interne)



Pagination - Exemple



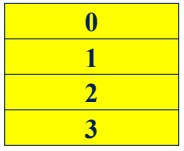


Table des pages du processus A

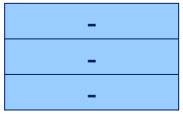


Table des pages du processus B

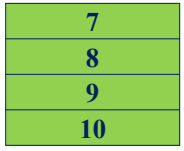


Table des pages du processus C

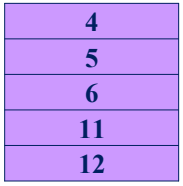
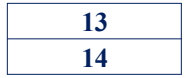


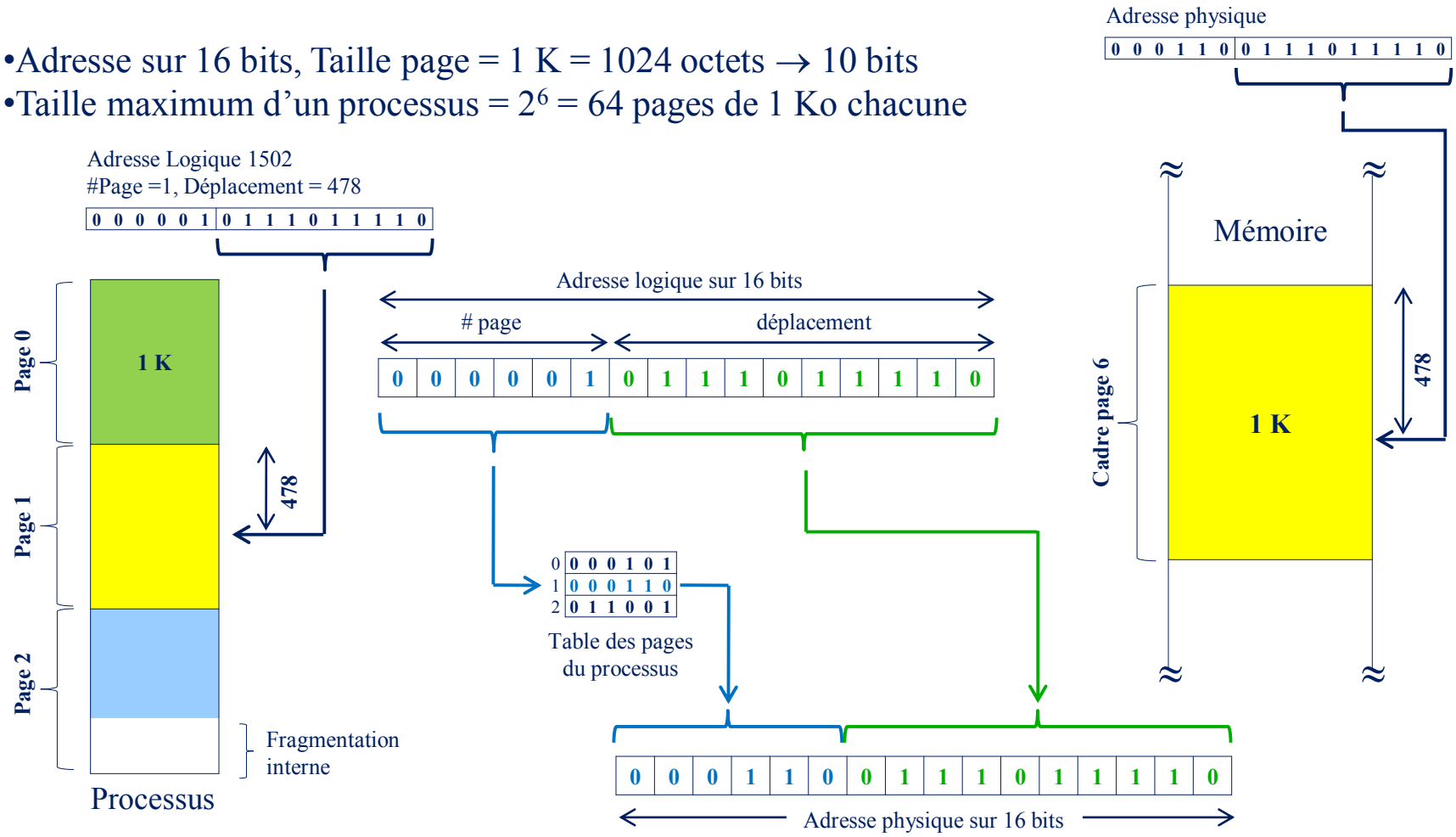
Table des pages du processus D



Liste des cadres libres

Pagination – Translation d'Adresse

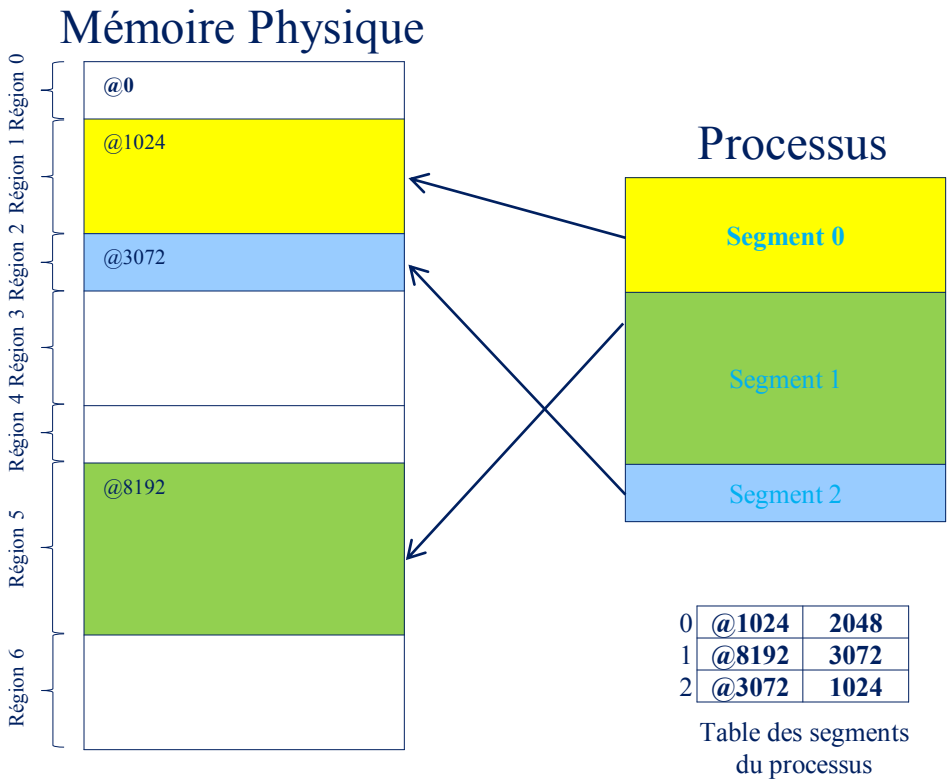
- Adresse sur 16 bits, Taille page = 1 K = 1024 octets → 10 bits
- Taille maximum d'un processus = 2^6 = 64 pages de 1 Ko chacune



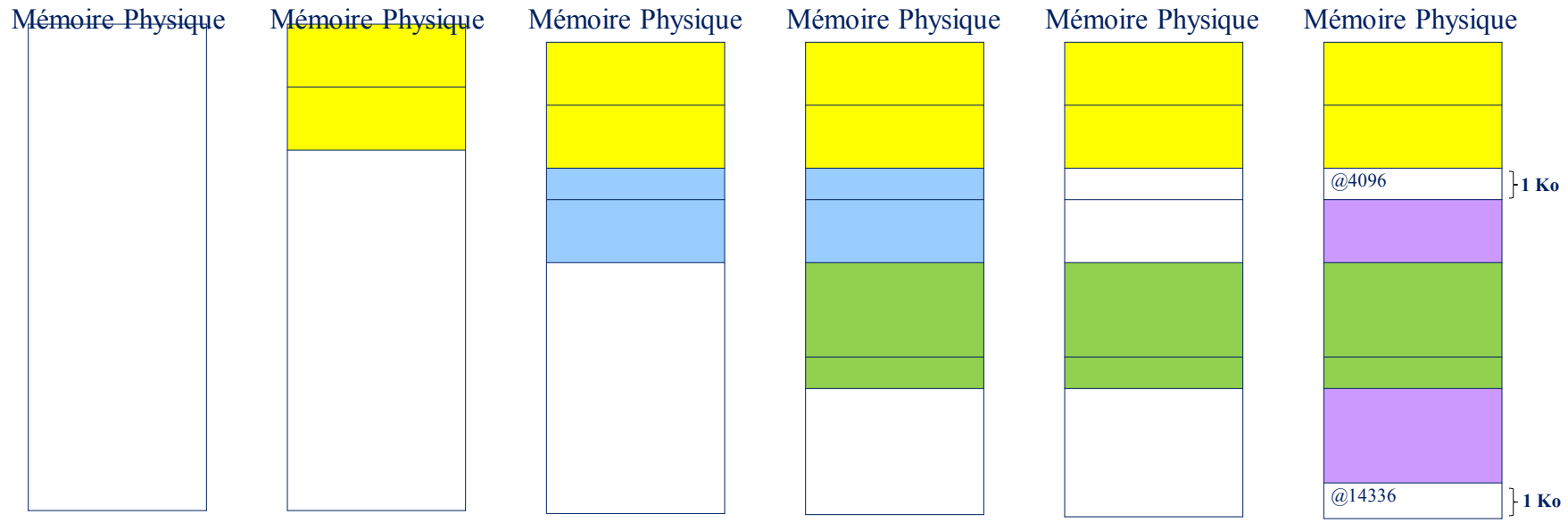
Taille page = 2^n ⇒ Adresse relative ≡ Adresse Logique (#page + déplacement)

Segmentation - Notion de Base

- Processus découpés en morceaux de tailles différentes (segments)
- Processus chargé en mémoire
  - Tous les segments du processus sont chargées
  - Les segments ne sont pas nécessairement contigus en mémoire
  - Une table des segments du processus est établie
- Espace perdu en mémoire
  - Parties ne pouvant contenir aucun segment d'un processus (fragmentation externe)
  - Plus les segments des processus sont petits plus la fragmentation est faible



Segmentation - Exemple



(a) Mémoire libre

(b) Chargement processus A

(c) Chargement processus B

(d) Chargement processus C

(e) Déchargement processus B

(e) Chargement processus D

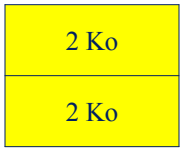


Table des segments du Processus A

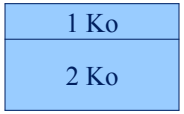


Table des segments du Processus B

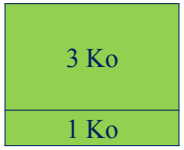


Table des segments du Processus C

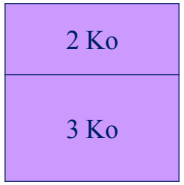


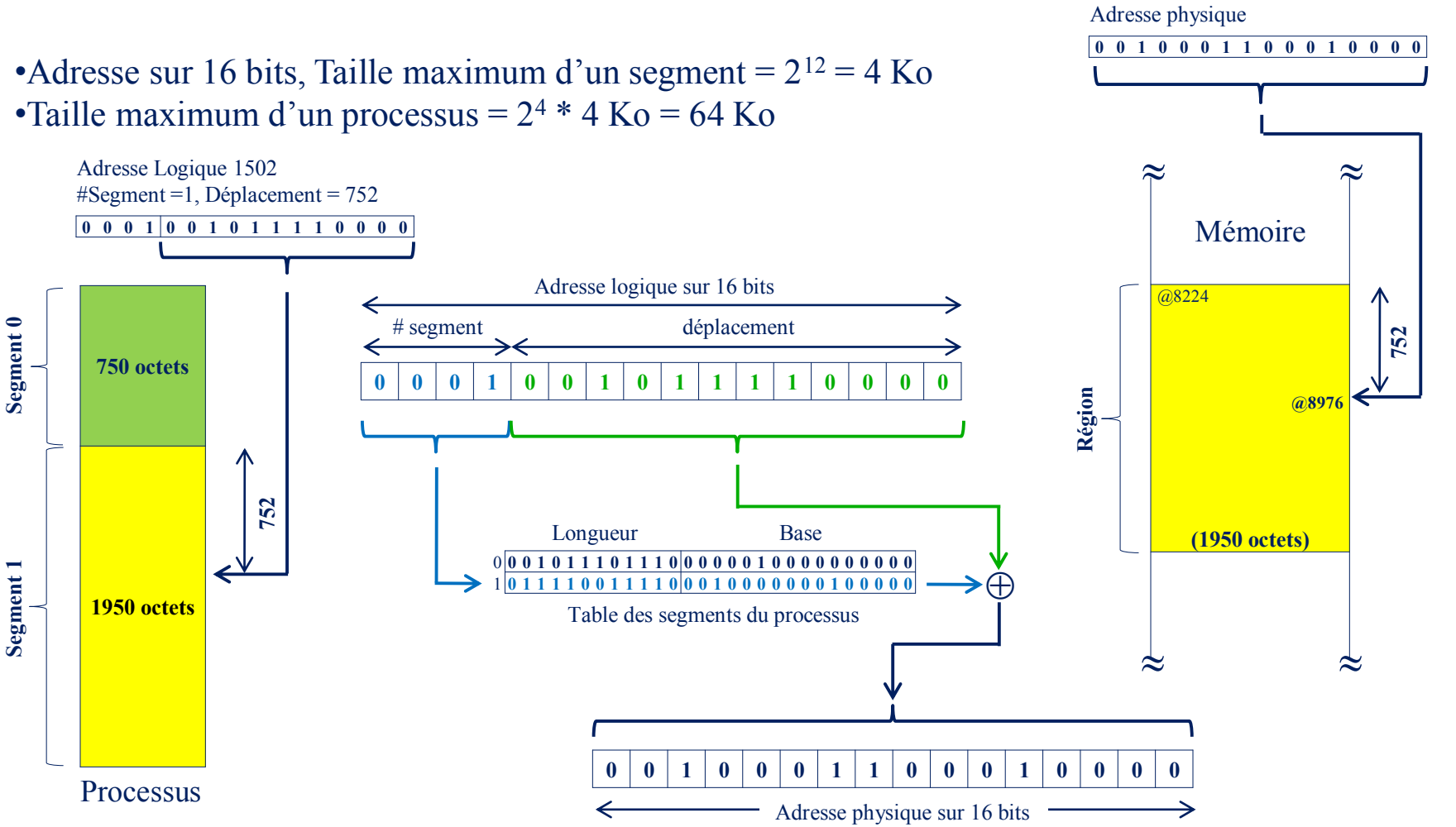
Table des segments du Processus D

@4096	1024
@14336	1024

Liste des régions libres

Segmentation – Translation d'Adresse

- Adresse sur 16 bits, Taille maximum d'un segment = 2^{12} = 4 Ko
- Taille maximum d'un processus = 2^4 * 4 Ko = 64 Ko



Si déplacement > longueur du segment ⇒ Débordement

Principes de la Mémoire Virtuelle

- Les références mémoire sont dynamiquement translatées en adresses physique en cours d'exécution
  - les processus sont swapés de façon à occuper différentes régions de la mémoire
- Un processus peut être scindé en plusieurs parties qui ne nécessitent pas d'être placées de manière contigüe en mémoire
- Il n'est pas nécessaire que toutes les parties d'un processus soient chargées en mémoire lors de son exécution
  - Le système charge en mémoire que quelques parties du processus
  - Une interruption est générée lorsqu'une adresse référence une partie non présente en mémoire
  - Le système bloque le processus en attente du chargement
  - La partie du processus contenant l'adresse logique est chargée en mémoire
    - Le système émet une requête d'E/S disque
    - Un autre processus est élu pendant l'opération d'E/S
    - Une interruption est générée lorsque l'E/S disque est achevée et le système fait passer le processus concerné dans l'état prêt

Avantages/Inconvénients de la MV

Avantages

- Un plus grand nombre de processus peut être maintenu en mémoire
  - Une partie de chaque processus est chargée plutôt que la totalité → plus de processus en mémoire (favorise/accroit la multiprogrammation)
  - A tout instant, il y a plus de chance d'avoir un processus prêt → utilisation efficace de la CPU
- Un processus peut être plus grand que la totalité de la mémoire centrale
- Décharge les programmeurs des contraintes mémoire

Inconvénients

- Risque de swapper en mémoire secondaire une partie du processus juste avant qu'elle soit nécessaire (*thrashing condition*)
- Le processeur risque de passer plus de temps à swapper les (parties de) processus qu'à exécuter les instructions des utilisateurs

Principe de localité

- Les références de programmes et de données d'un processus sont confinées sur un petit espace
- Cela suggère qu'un petit nombre de parties d'un processus sera nécessaire durant une courte durée de temps
- Il est possible de prédire quelles parties du processus seront nécessaires dans un futur proche (*avoids trashing*)
- La mémoire virtuelle peut être pratique et efficace si :
  - Support matériel pour la pagination et/ou segmentation
  - Support par le SE des mouvements de pages et/ou segments entre mémoire secondaire et mémoire principale

Pagination

- Chaque processus a sa propre table de pages
- Chaque entrée de la table de pages contient le numéro de cadre de page correspondant en mémoire centrale
- Toutes les parties d'un processus ne sont pas nécessairement chargées en mémoire lors de son exécution
  - Il est nécessaire de savoir si la page est en mémoire centrale ou pas (bit P)
  - Il est nécessaire de savoir si le contenu de la page a été modifié depuis son chargement (bit M)
  - Il est nécessaire de savoir si la protection et le partage sont gérés au niveau page

Adresse virtuelle

Entrée de la table de pages

Pagination - Translation d'Adresse

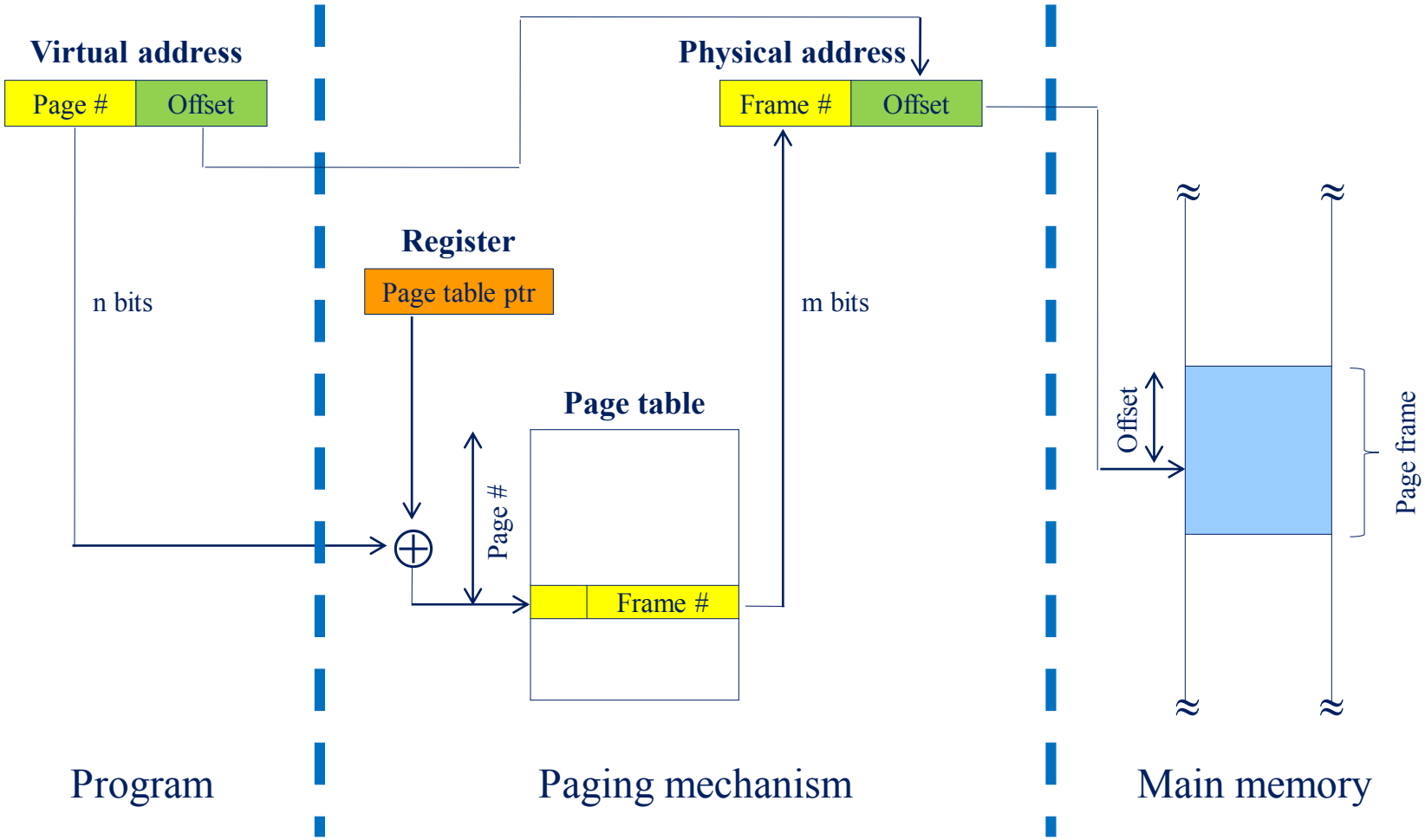


Table des Pages

- Vaste espace d'adresses logiques (2^{32} à 2^{64})
- Table de pages elle-même très volumineuse

Taille une entrée = 4 octets → Taille table de pages = 4 Mo d'espace d'adresses physiques/processus

- Les tables de pages sont elles-mêmes stockées en mémoire virtuelle
- Quand un processus s'exécute, une partie seulement de sa table de pages est en mémoire principale

Pagination à Deux Niveaux

Translation d'Adresse

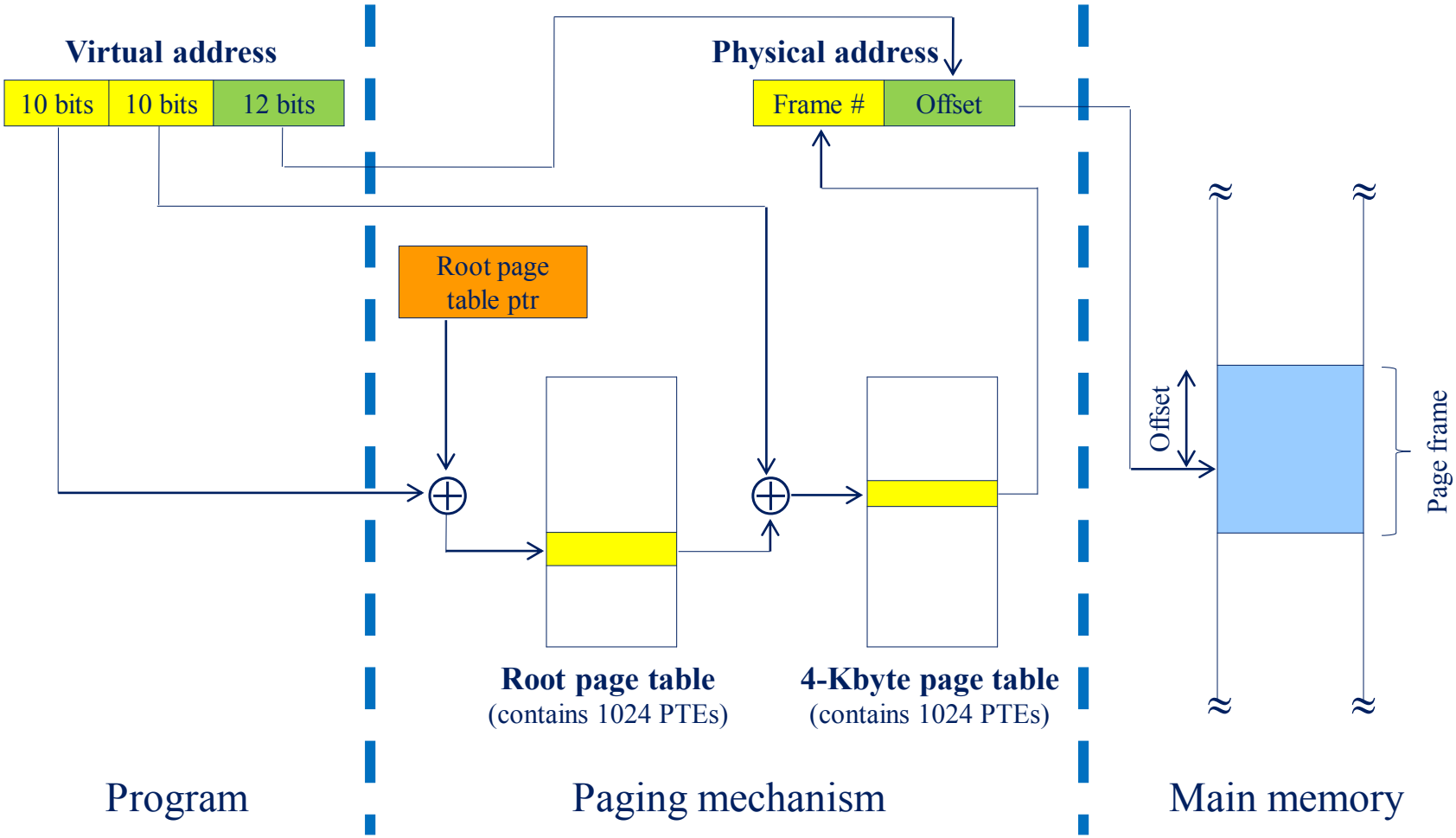


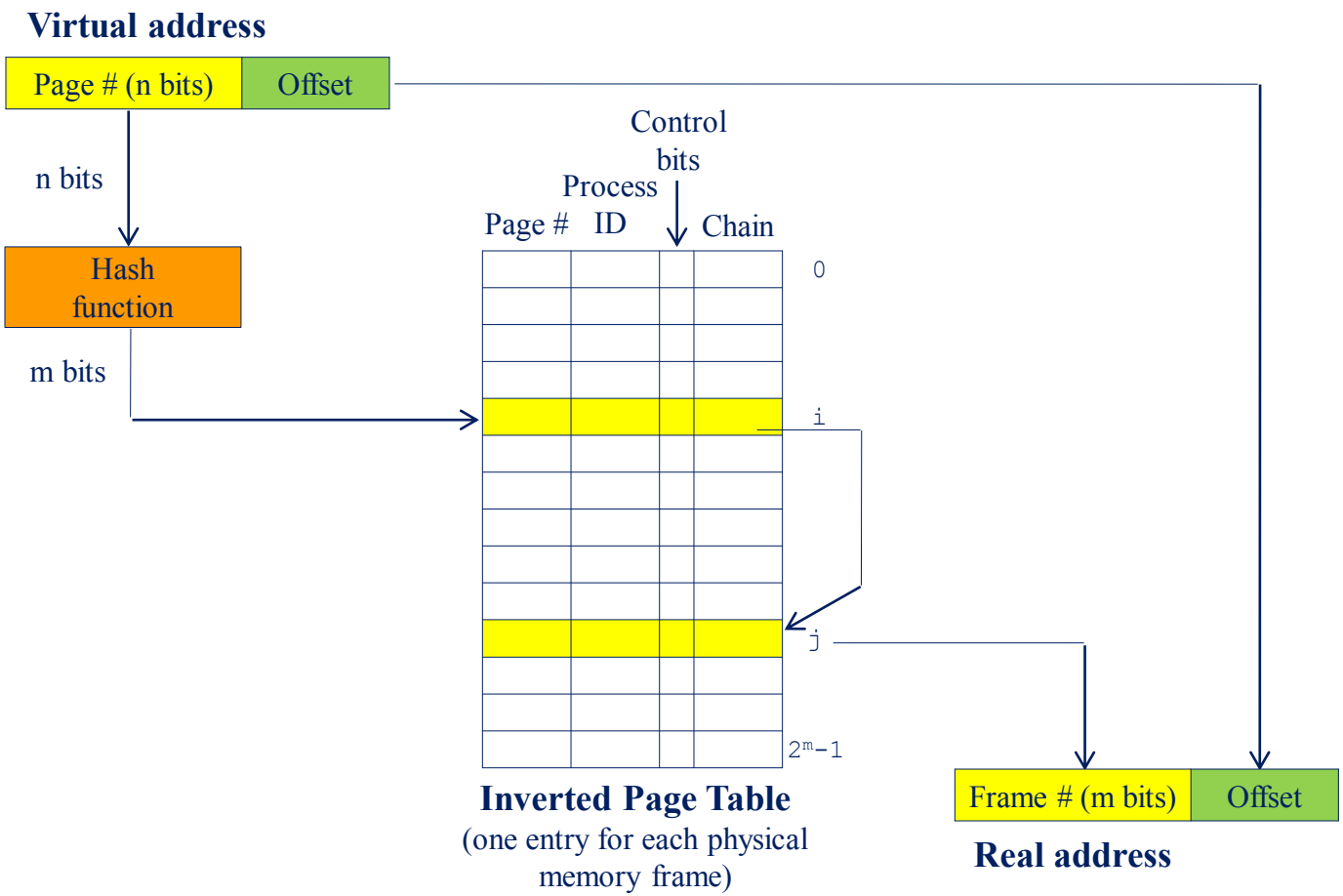
Table des Pages Inversée

- Une seule table où chaque entrée correspond à une page physique
- Le numéro de page d'une adresse virtuelle est soumis à une fonction (ou table) de hachage pour produire un index de la table inversée
- La taille de la table est fixe quel que soit le nombre de processus ou le nombre de pages virtuelles supportées

Entrée d'une table de pages inversée

- Chaînage en cas de collisions

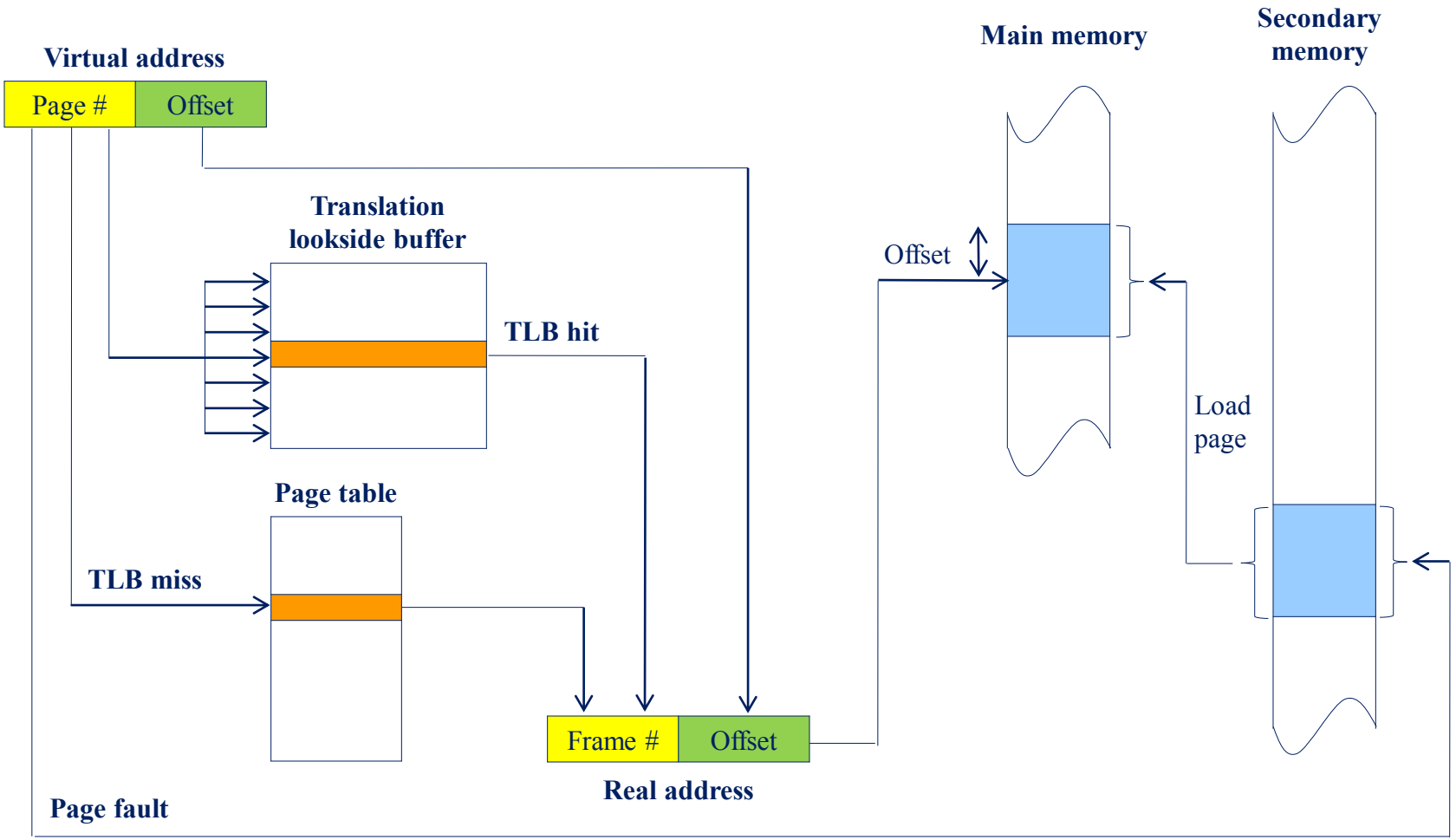
Table de Pages Inversée



Translation Look-aside Buffer (TLB)

- Chaque référence mémoire virtuelle (d'une donnée, par exemple) implique deux accès en mémoire physique
  - L'un pour l'entrée de la table de pages
  - L'autre pour la donnée
- L'accès mémoire est ralenti d'un facteur 2
- Utilisation d'un cache à grande vitesse pour y remédier
  - Appelé Translation Look-aside Buffer (TLB)
  - Petite mémoire associative très rapide
  - Contient les entrées de table de pages les plus récemment utilisées
- Principe
  - Pour une adresse virtuelle donnée, le processeur examine la TLB
  - Si l'entrée de la table de pages est présente, le cadre de page est disponible et l'adresse réelle est construite
  - Si l'entrée de la table de pages n'est pas présente, le numéro de page est utilisé pour indexer la table de pages du processus
  - Si la page n'est pas en mémoire principale, alors générer un défaut de page
  - La TLB est mise à jour pour inclure la nouvelle entrée de table de pages

Translation Look-aside Buffer

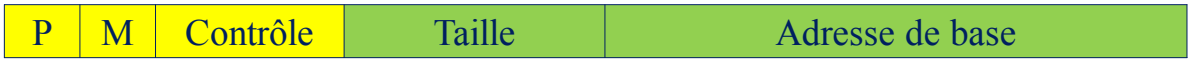


Segmentation

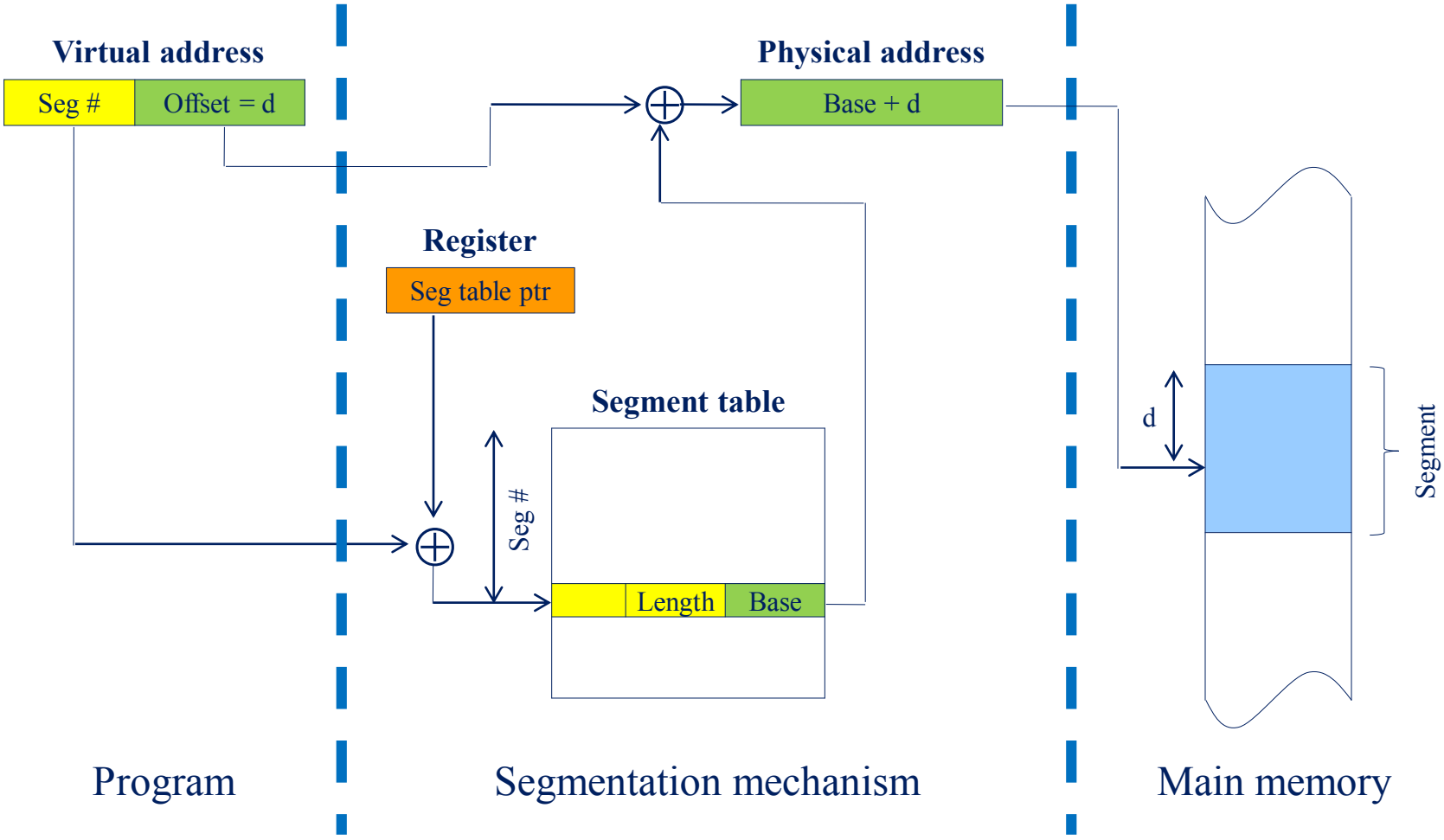
- Les segments peuvent être de tailles différentes et dynamiques
- Gestion facilité des structures de données dynamiques
- Adaptée au partage de données entre processus
- Adaptée à la protection
- Une table de segments par processus
- Les segments d'un processus ne sont pas nécessairement tous chargés
  - Il est nécessaire de savoir si le segment est en mémoire centrale ou pas (bit P)
  - Il est nécessaire de savoir si le contenu du segment a été modifié depuis son chargement (bit M)
  - Il est nécessaire de savoir si la protection et le partage sont gérés au niveau segment

Adresse virtuelle

Entrée de la table de segments



Segmentation - Translation d'Adresse



Combinaison Segmentation et Pagination

- La pagination est transparente au programmeur
- La segmentation est visible pour le programmeur
- Chaque segment est découpé en pages de taille fixe

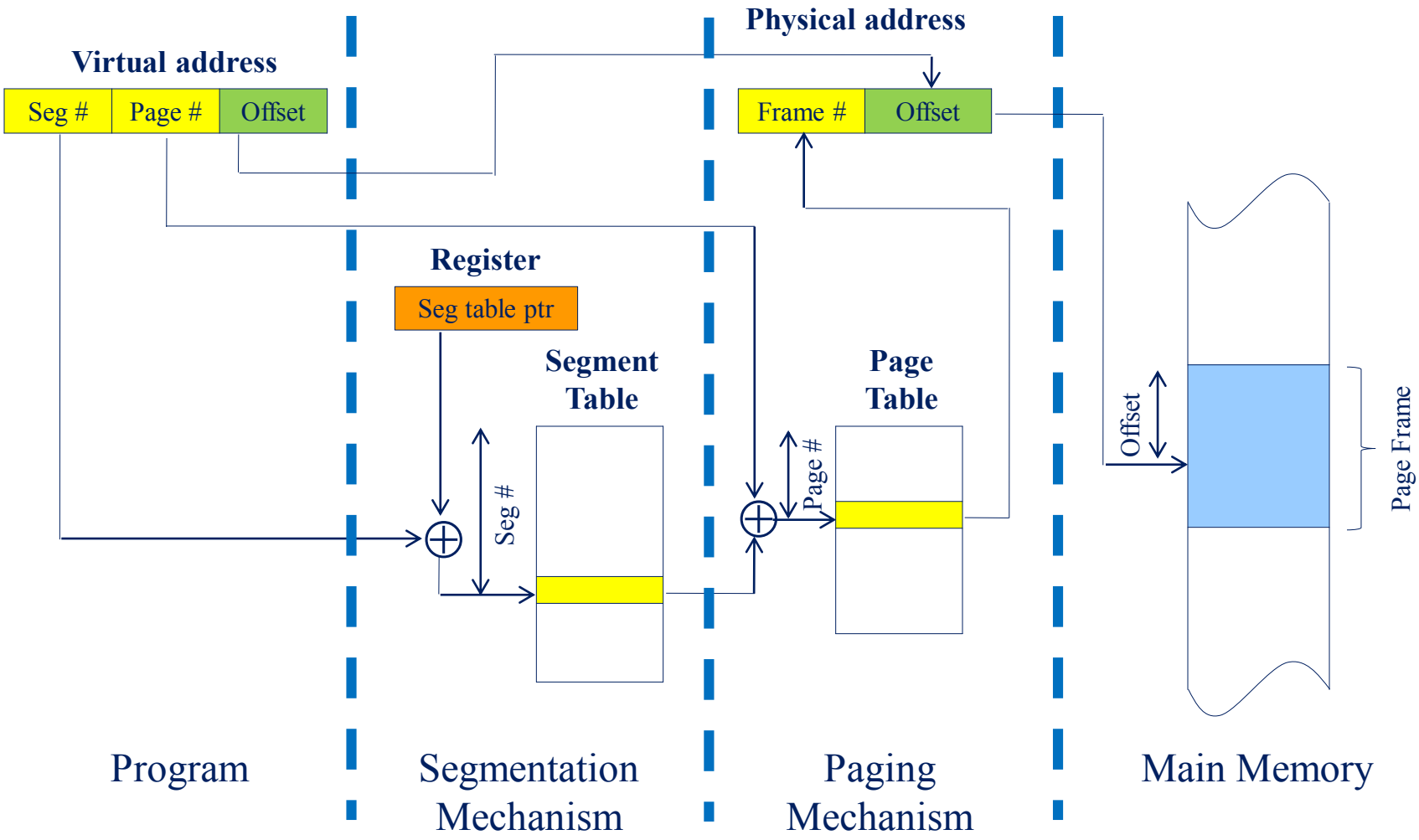
Adresse virtuelle

Entrée de la table de segments

Entrée de la table de pages

Segmentation et Pagination

Translation d'Adresse



Algorithmes de Gestion Mémoire

- Minimiser le taux de défauts de pages
  - Défaut de page → Surcharge (quelle page remplacée, E/S des pages, commutation de processus)
- Minimiser la probabilité, durant l'exécution d'un processus, de référencer une page absente
- Politiques mises en œuvre :
  - Politique de chargement : détermine **quand** une page doit être chargée en mémoire principale
    - A la demande : une page n'est chargée que si une référence à un emplacement de cette page est faite
    - Anticipation : des pages, autres que celles concernées par un défaut de page, sont chargées
  - Politique de placement : détermine **où**, en mémoire principale, (une portion) un processus doit résider
  - Politique de remplacement : détermine **quelle** page, en mémoire principale, doit être sélectionnée pour être remplacée quand une nouvelle page doit être chargée
    - Combien de cadres de pages allouer à chaque processus actif (*Fixed/Variable Allocation*)
    - L'ensemble des pages concernées par le remplacement est-il limité aux seules pages du processus ayant provoquer un défaut de page ou à l'ensemble des cadres de pages (*Local/Global Replacement*)
    - Quelle page, parmi l'ensemble des pages concernées, sera sélectionnée pour remplacement

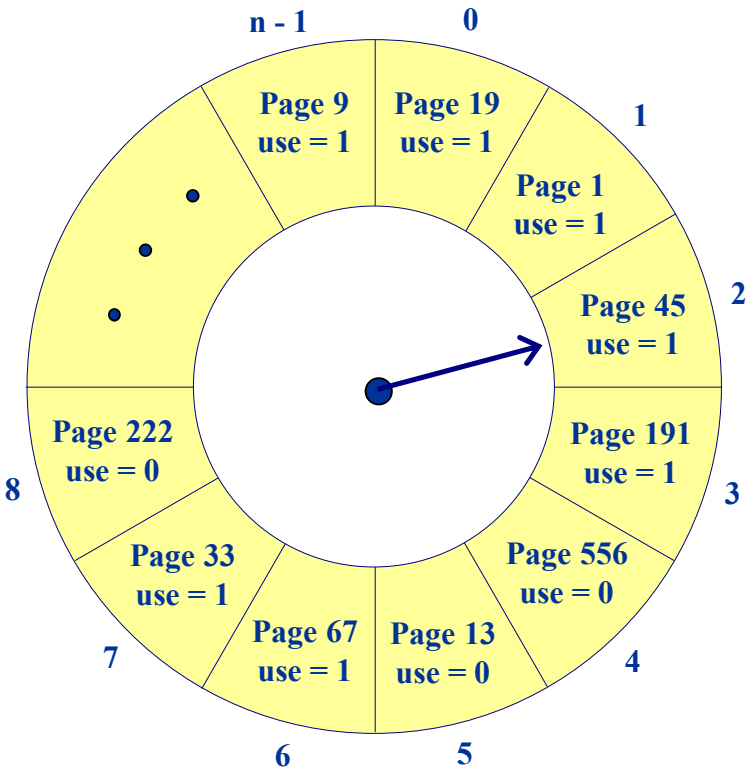
Stratégies de la Politique de Remplacement

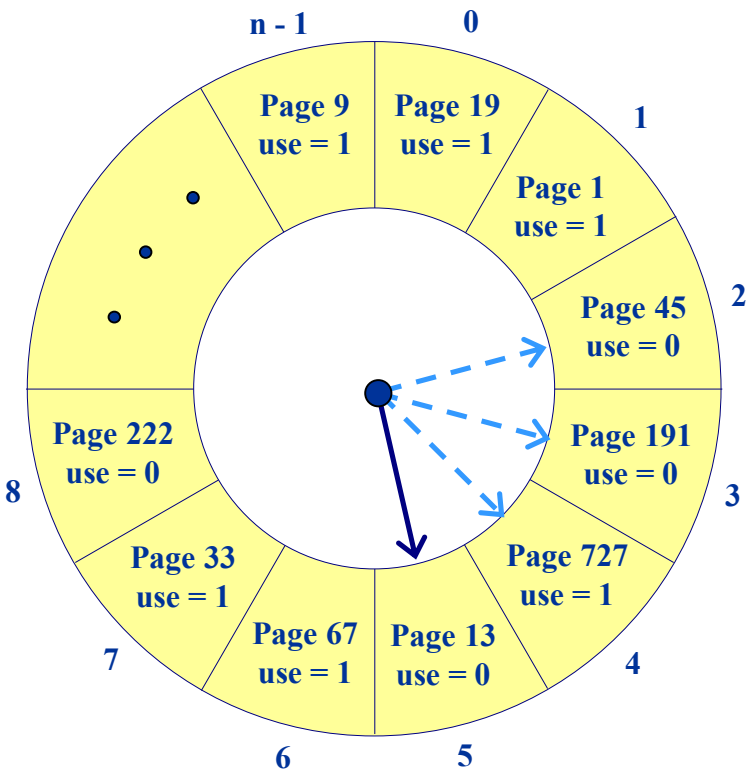
	Local Replacement	Global Replacement
Fixed Allocation	• Number of frames to process is fixed • Page to be replaced is chosen from among the frames allocated to that process.	• Not possible.
Variable Allocation	• The number of frames allocated to a process may be changed from time to time, to maintain the working set of the process. • Page to replaced is chosen from among the frames allocated to that process.	• Page to be replaced is chosen from all available frames in main memory; this causes the size of the resident set of processes to vary.

Algorithmes de Remplacement

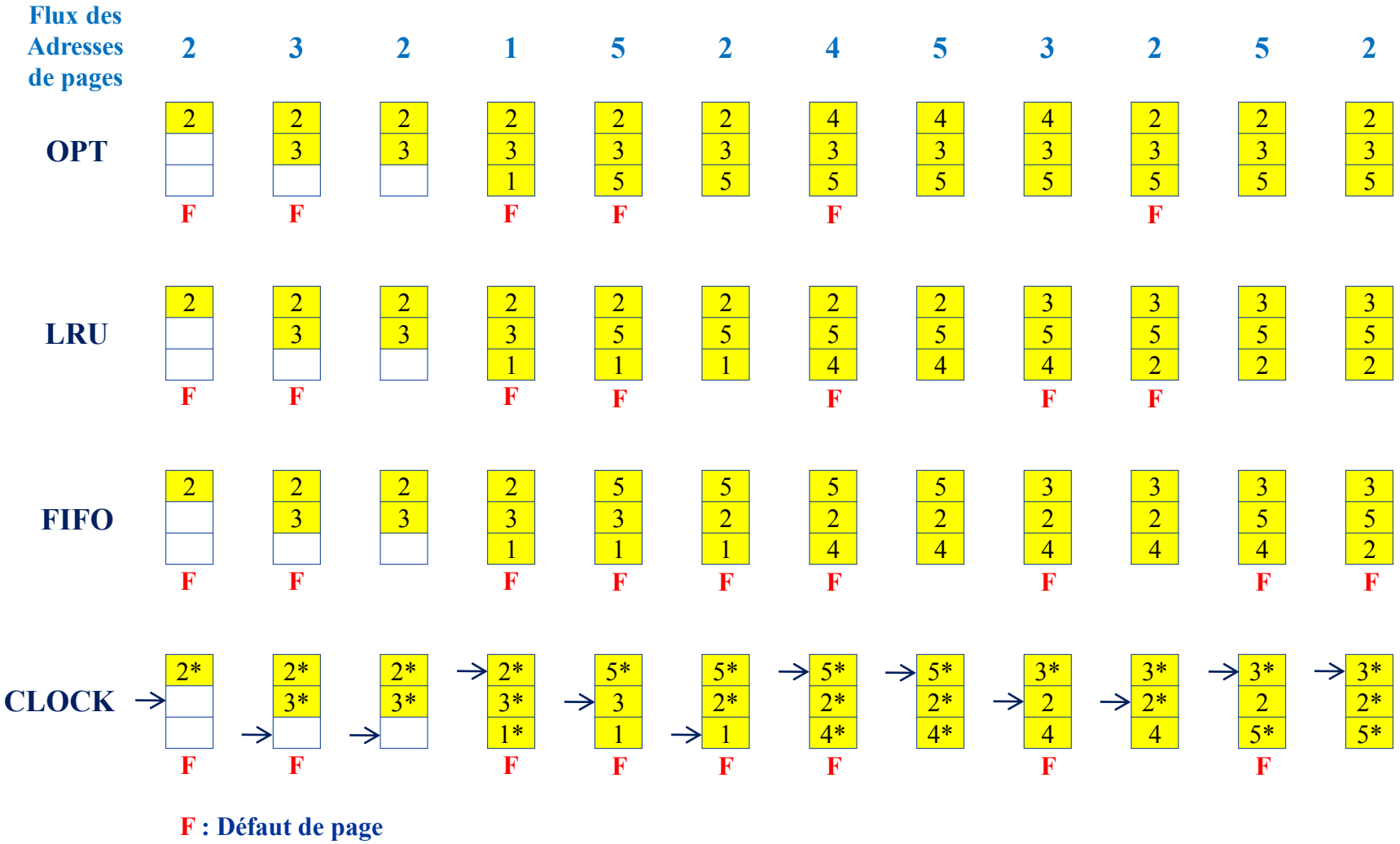
- Optimal
  - Sélectionne la page qui sera le plus tardivement référencée
  - Politique engendrant le moins de défauts de pages (référence de comparaisons)
- Least Recently Used (LRU)
  - Remplace la page en mémoire la plus anciennement référencée
  - Implémentation difficile : surcharge (datage à chaque référence, pile des pages référencées)
  -
- First-In-First-Out (FIFO)
  - Considère les cadres de pages alloués à un processus comme un tampon circulaire
  - Simple mais pas très efficace
- Clock
  - Association d'un bit (d'utilisation) à chaque cadre de page
  - Considère les cadres de pages comme un tampon circulaire

Exemple de la politique Clock



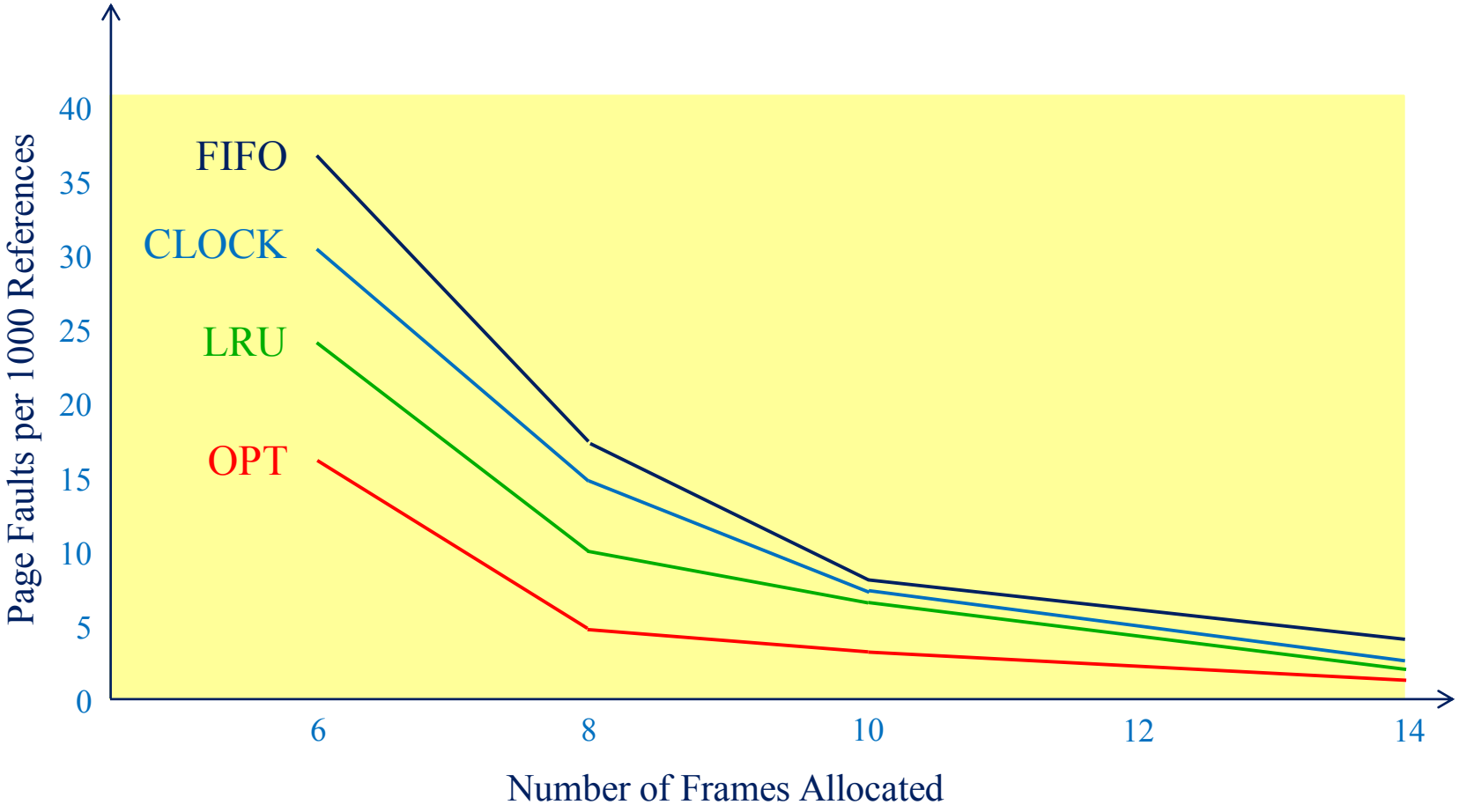


Comportement des Algorithmes



Comparaison des Algorithmes

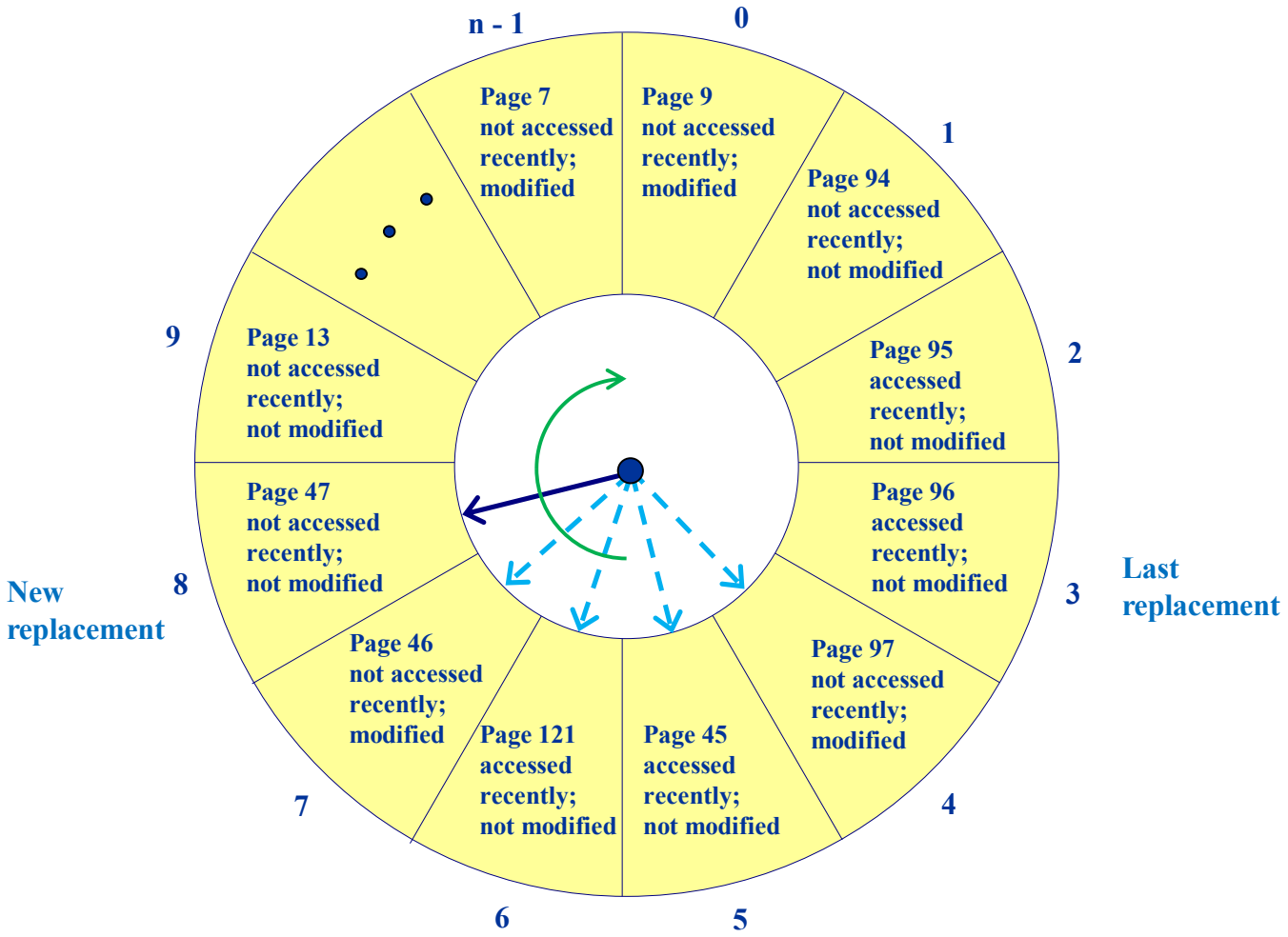
Cas : Fixed Allocation and Local Replacement



Amélioration de la Politique Clock

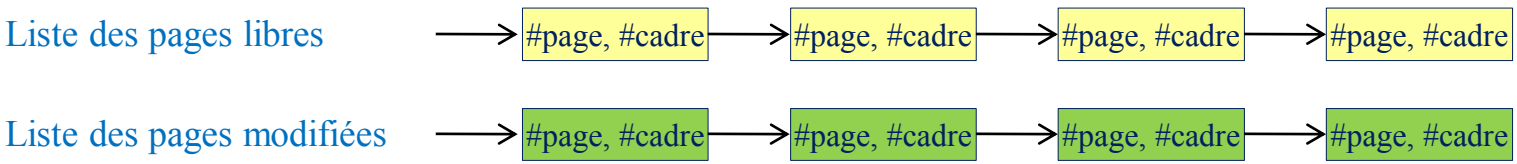
- Augmenter le nombre de bits utilisés
  - La plupart des systèmes supportant la pagination utilisent un bit de modification (en plus du bit d'utilisation)
  - Chaque cadre de page fait partie de l'une des catégories suivantes :
    - Pas accédé récemment, pas modifié (u = 0; m = 0)
    - Accédé récemment, pas modifié (u = 1; m = 0)
    - Pas accédé récemment, modifié (u = 0; m = 1)
    - Accédé récemment, modifié (u = 1; m = 1)
  - Principe de l'algorithme
    1. Parcourir le tampon des cadres à partir de la position courante de la flèche. Durant ce parcourt, n'opérer aucun changement au bit d'utilisation. Le premier cadre rencontré avec (u = 0; m = 0) est sélectionné pour remplacement.
    2. Si l'étape 1 échoue, parcourir à nouveau le tampon et rechercher le cadre avec (u = 0; m = 1). Le premier cadre rencontré, de cette catégorie, est sélectionné pour remplacement. Durant ce parcourt, mettre à 0 le bit d'utilisation de chaque cadre rencontré.
    3. Si l'étape 2 échoue, le pointeur aura rejoint sa position initiale et tous les cadres du tampons ont leur bit d'utilisation à 0. Répéter l'étape 1 et, si nécessaire, l'étape 2. Cette fois-ci, un cadre sera trouvé pour remplacement.
- Amélioration : gain de temps
  - privilégier les cadres qui n'ont pas été modifiés (pas d'écriture)

Exemple de Remplacement



Page Buffering

- Amélioration des performances de la pagination
  - Utilisation d'une politique simple de remplacement de pages (telle que FIFO),
  - La page à remplacer n'est pas perdue mais mise dans une des deux listes :

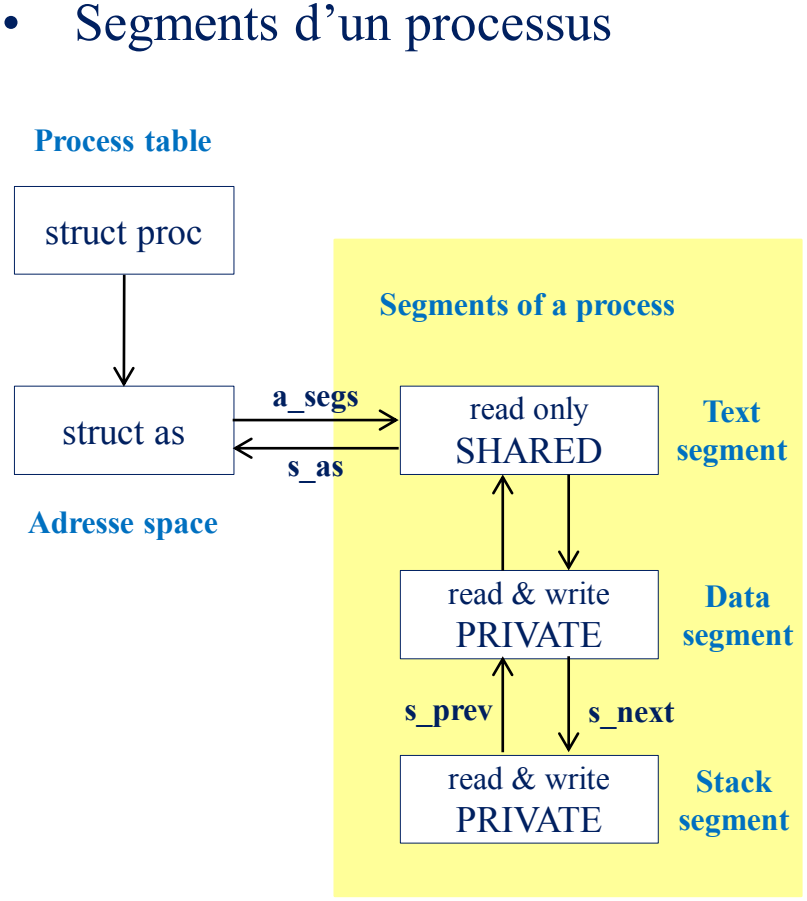


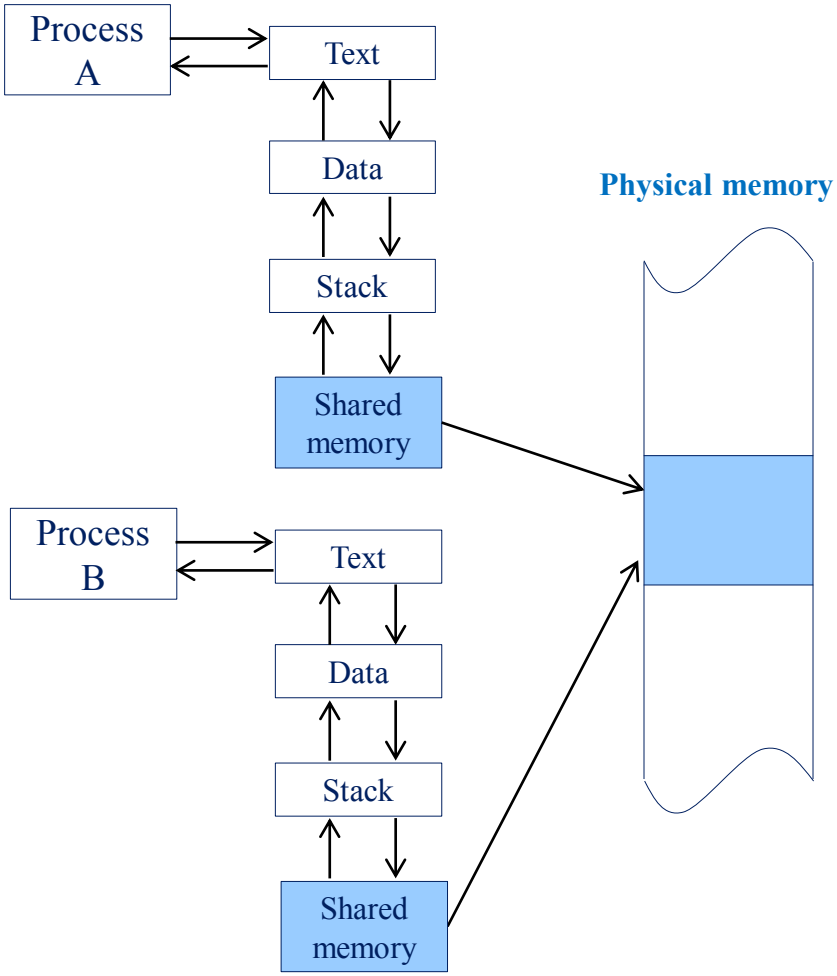
  - Les pages ne sont pas physiquement déplacées de la mémoire
    - seule l'entrée dans la table des pages est retirée pour être mise dans l'une des deux listes
    - Si le processus référence à nouveau cette page, elle est ajoutée à moindre coût

- Le système essaye de maintenir, à tout instant, un minimum de pages libres
- La liste des pages modifiées est organisée de façon que les pages puissent être écrites par groupes, limitant ainsi le nombre d'accès disque

Gestion Mémoire Unix

- Segments d'un processus





Gestion Mémoire Unix

Sous SVR4 (et Solaris), deux gestionnaires mémoire cohabitent

- Allocateur mémoire pour le noyau
  - Durant son exécution, le noyau alloue/libère fréquemment de petites quantités mémoire (structures proc, zombie, vnode, descripteurs de fichiers, …), bien inférieures à la taille d'une page
  - Le nombre de demandes de blocks d'une taille particulière varie lentement dans le temps
  - Utilisation d'une politique modifiée du système buddy (*lazy buddy system*), moins coûteuse que les politique best-fit ou first-fit

    → repousser le regroupement des blocks jusqu'à ce que cela devienne nécessaire

- Système de pagination
  - Allocation des cadres de pages aux processus
  - Allocation des cadres de pages pour les tampons disques

Gestion Mémoire Unix

Structures de données utilisées par le système de pagination du SVR4

- Page table : une table de pages par processus, avec une entrée pour chaque page en mémoire virtuelle pour ce processus

Page table entry

Page frame number	Age	Copy on write	Modify	Reference	Valid	Protect

- Disk block table : une entrée de cette table est associée à chaque page d'un processus; elle décrit la copie disque de cette page virtuelle

Disk block descriptor

Swap device number	Device block number	Type of storage

- Page frame data table : décrit chaque cadre de la mémoire physique et est indexée par le numéro de cadre

Page frame data table entry

Page state	Reference count	Logical device	Block number	Pfdata pointer

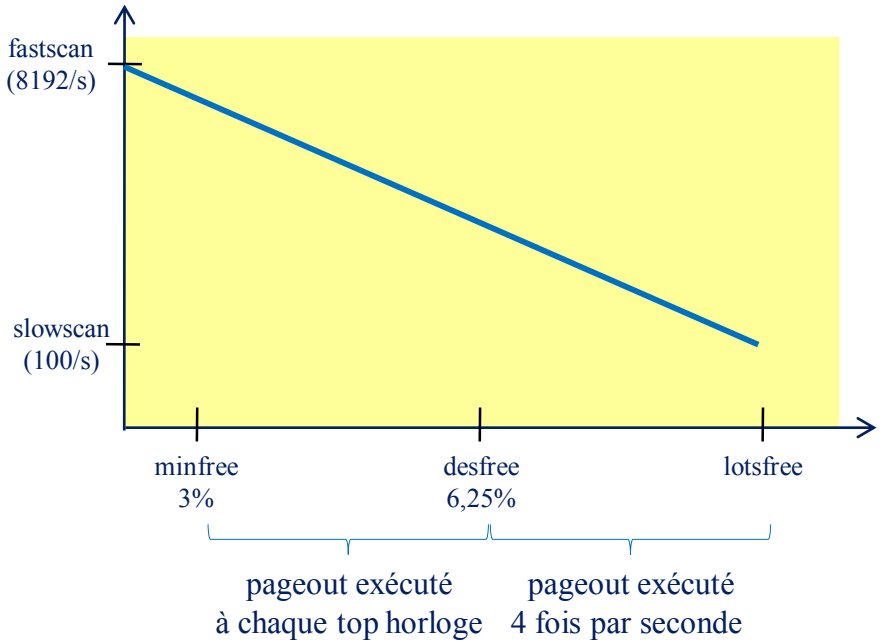
- Swap-use table : une table par unité de swap, avec une entrée pour chaque page de l'unité

Swap_use table entry

Reference count	Page/storage unit number

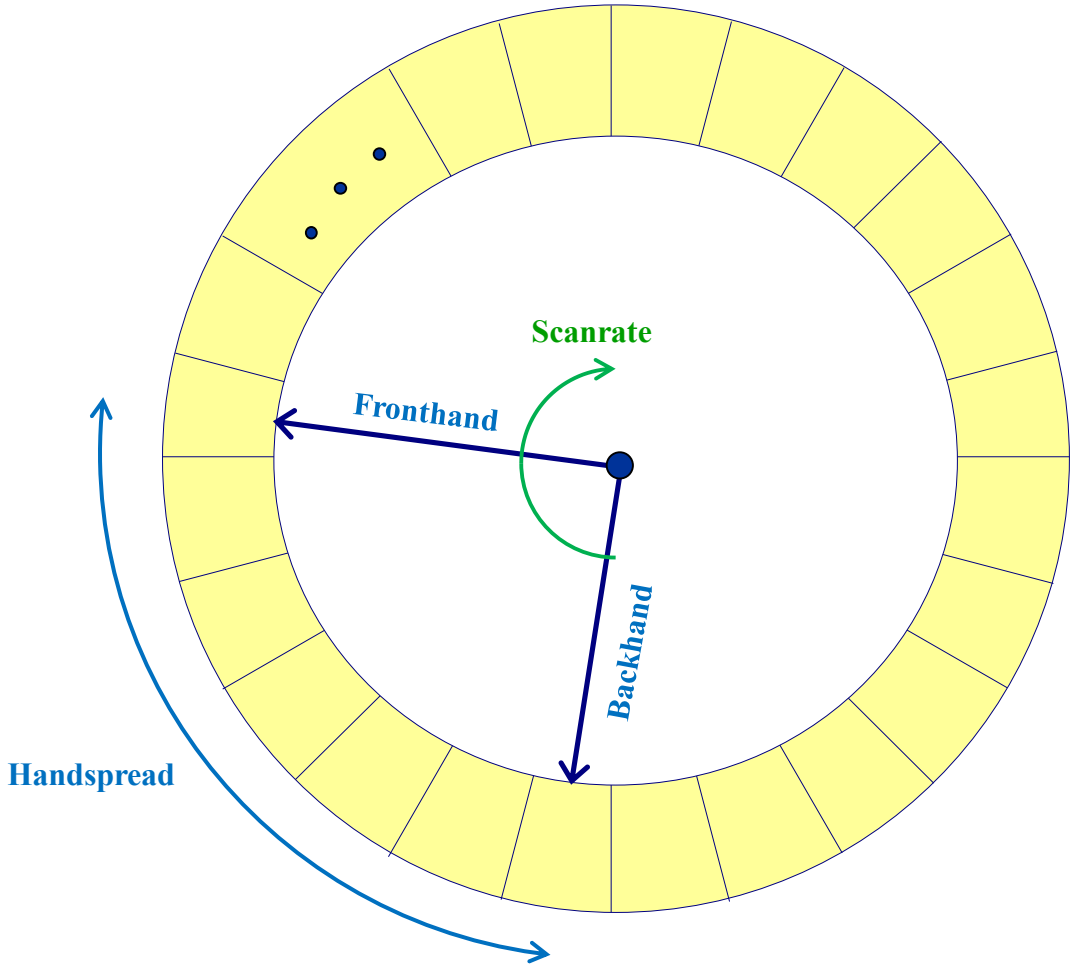
Gestion Mémoire Unix

- Remplacement de pages
  - La table *page frame data table* est utilisée pour le remplacement des pages
  - Tous les cadres disponibles sont chaînés pour constituer la liste des cadres libres, pour le chargement de pages lorsque nécessaire
  - Lorsque le nombre de cadres libres est inférieur à un certain seuil (*lotsfree*), le noyau (*pageout*) "vol" des pages pour compenser



Gestion Mémoire Unix

- Algorithme de remplacement Not Recently Used (NRU)



Bibliographie

- Operating Systems, Internals and Design Principles,
  Willam Stallings - Printice Hall

- The magic garden explained, The internals of Unix System V Release 4
  Berny Goodheart & James Cox - Prentice Hall

- Principes des Systèmes d'Exploitation,
  A. Silberschatz, P.B. Galvin, G. Gagne - Vuibert