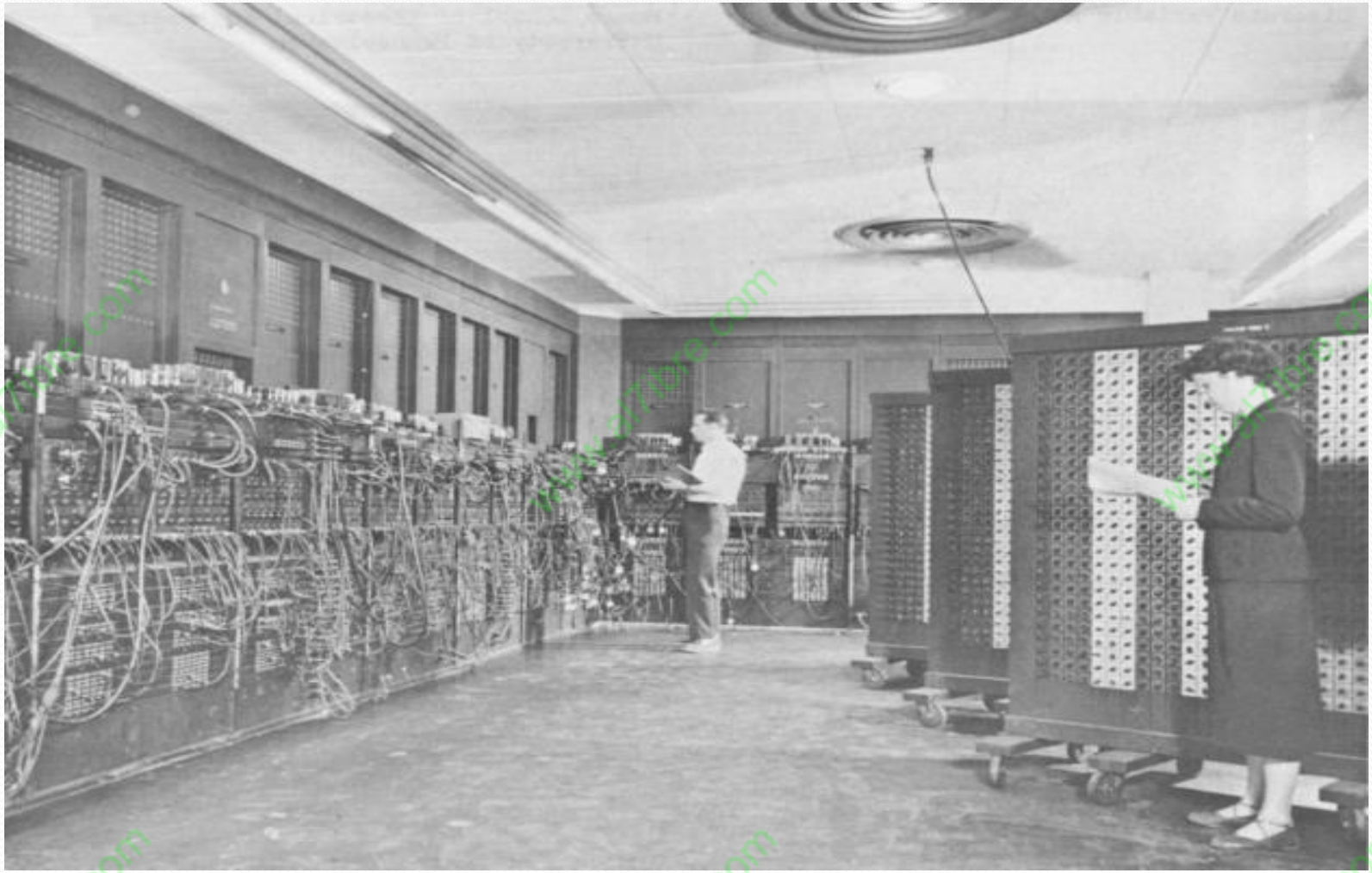
Introduction
Historique et évolution des ordinateurs

Pr. Omar Megzari
Département d'Informatique
FSR

megzari@fsr.ac.ma

www.al7ibre.com

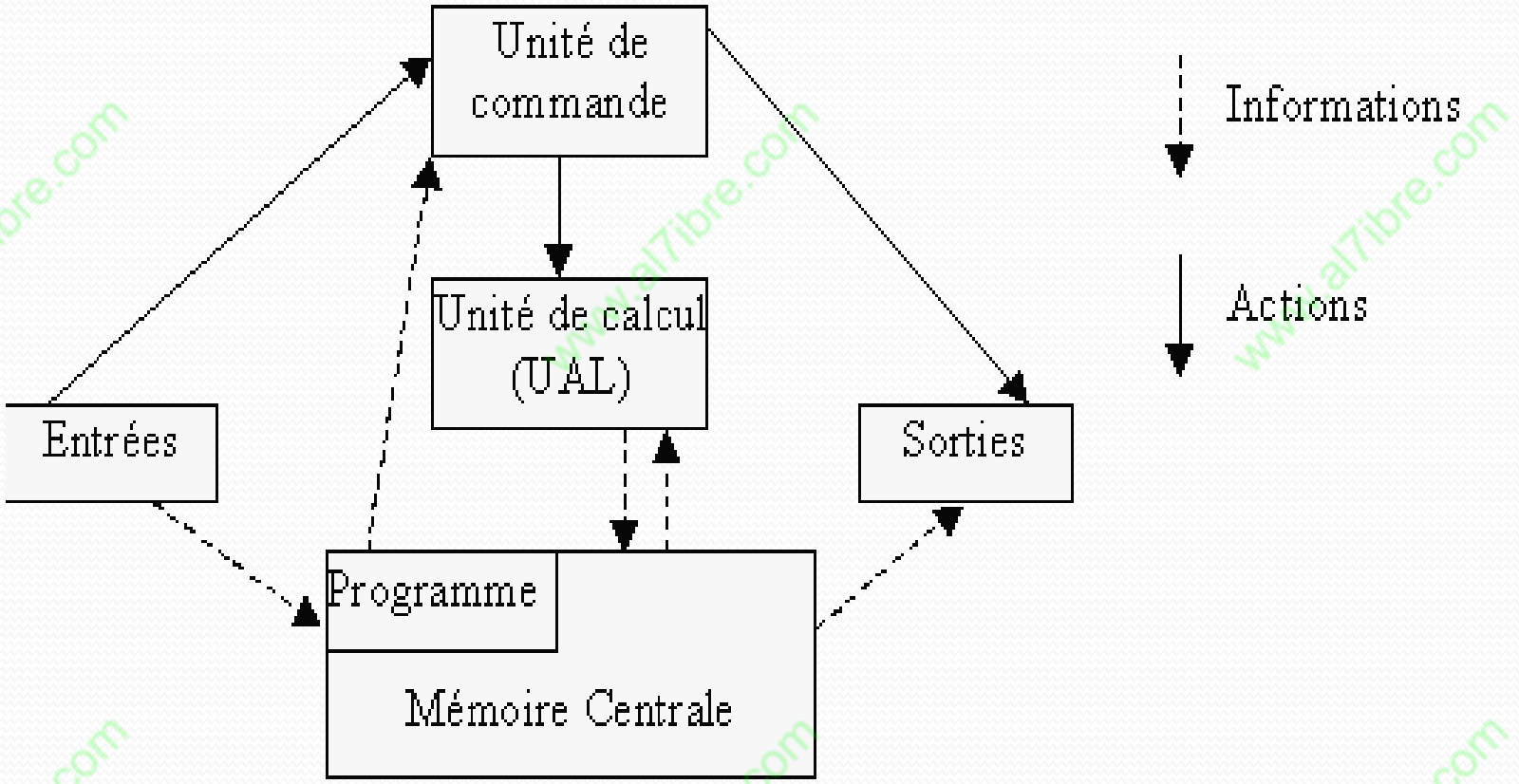
ENIAC pesait 30 tonnes



www.al7ibre.com

Schéma de la machine de Von Newman

UAL = unité arithmétique et logique



www.al7ibre.com

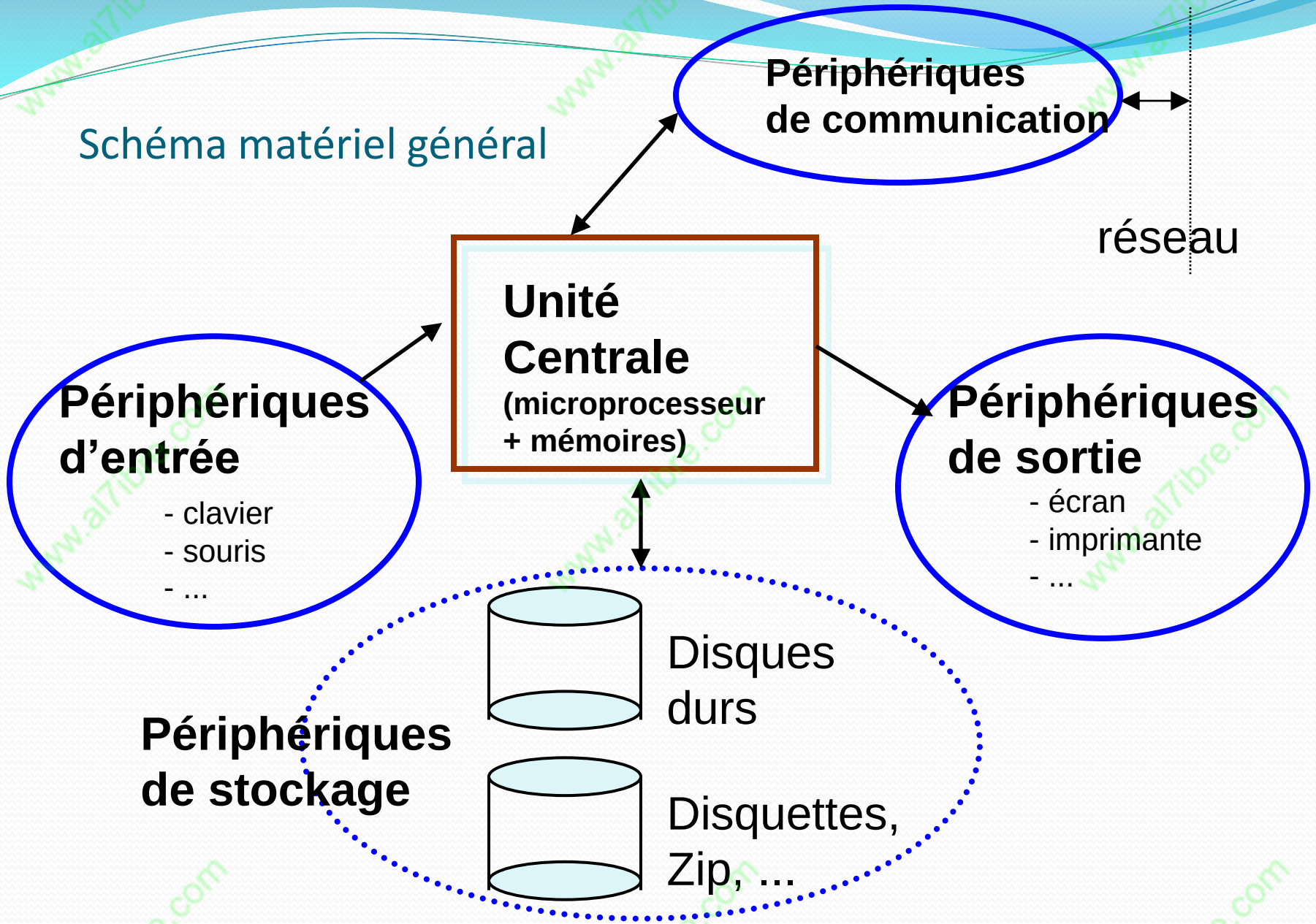
Machine de Von Newman

Ces dispositifs permettent la mise en oeuvre des fonctions de base d'un ordinateur :

-le stockage de données,
-le traitement des données,
-le mouvement des données et
-le contrôle des périphériques.

www.al7ibre.com

Schéma matériel général



www.al7ibre.com

L’unité centrale

- Le (micro)processeur ou CPU : Central Processing Unit
  - Unité arithmétique et logique (UAL) et Unité de commande
- Il exécute les programmes :
  - un programme est une suite d’instructions

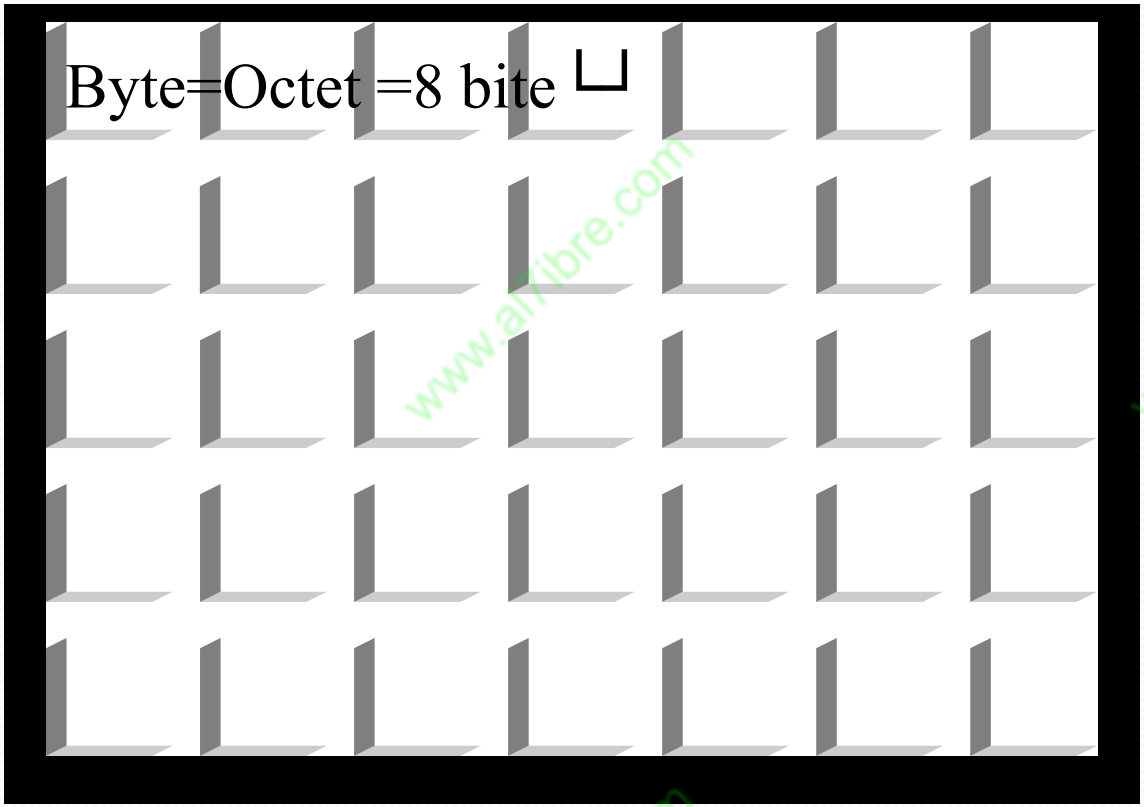
www.al7ibre.com

Mémoire vive : RAM

- RAM (Random Access Memory)
  - Permet de stocker des informations lorsqu'elle est alimentée électriquement
  - Lecture / Écriture
  - Mémoire volatile : contient des programmes et des données en cours d'utilisation
  - Capacité variable selon les ordinateurs

www.al7ibre.com

Mémoire vive : RAM



www.al7ibre.com

Mémoire morte : ROM

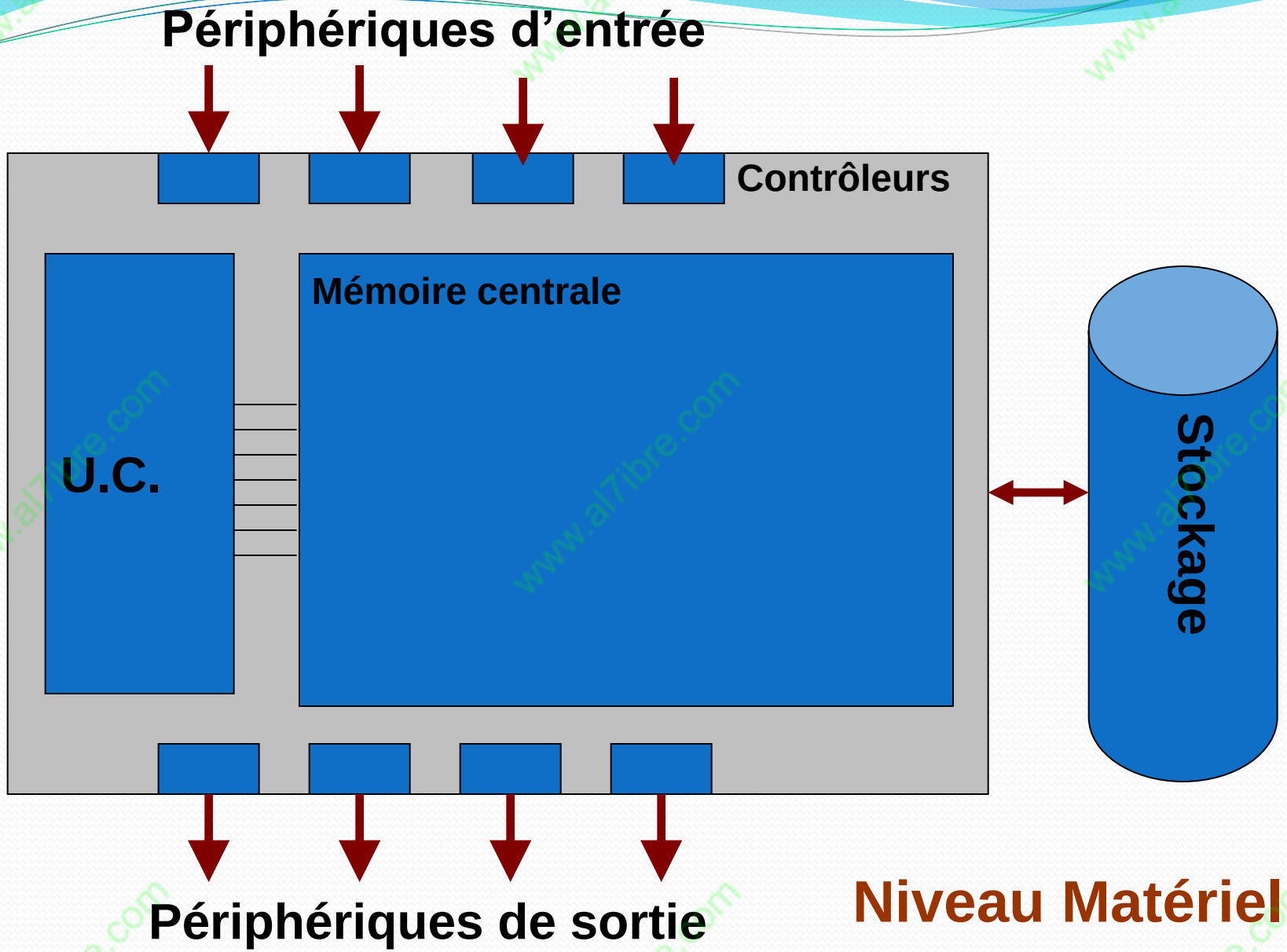
- ROM (Read Only Memory)
  - En lecture seule
  - Mémoire permanente
  - Contient les programmes de base au démarrage de l'ordinateur BIOS (Basic Input Output System)
  - Permet l'initialisation de l'ordinateur, initialisation de périphériques, lancement du système d'exploitation…

www.al7ibre.com

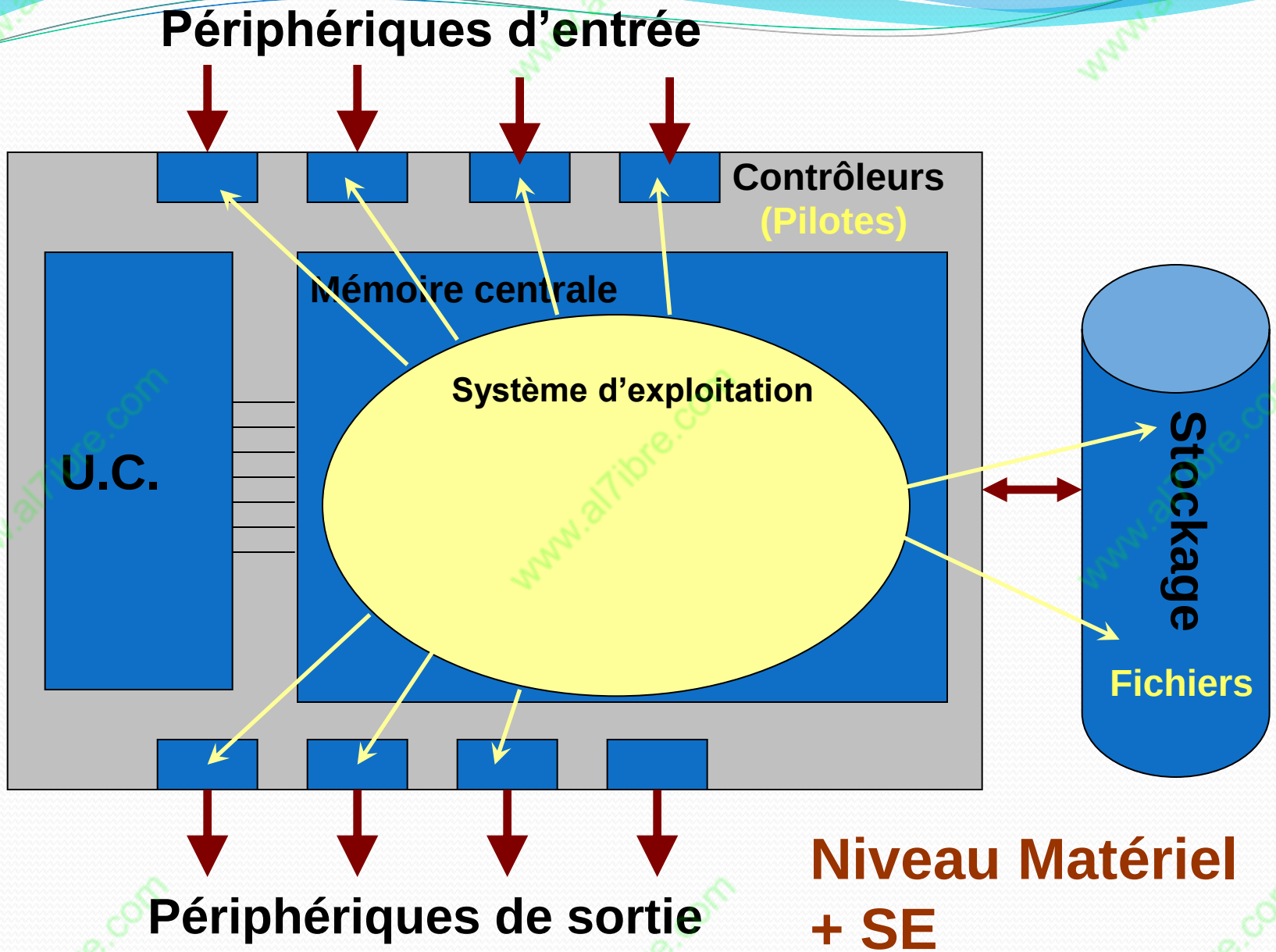
Les périphériques

- Les périphériques de stockage
- Les périphériques d'entrée
- Les périphériques de sortie
- Les périphériques de communication

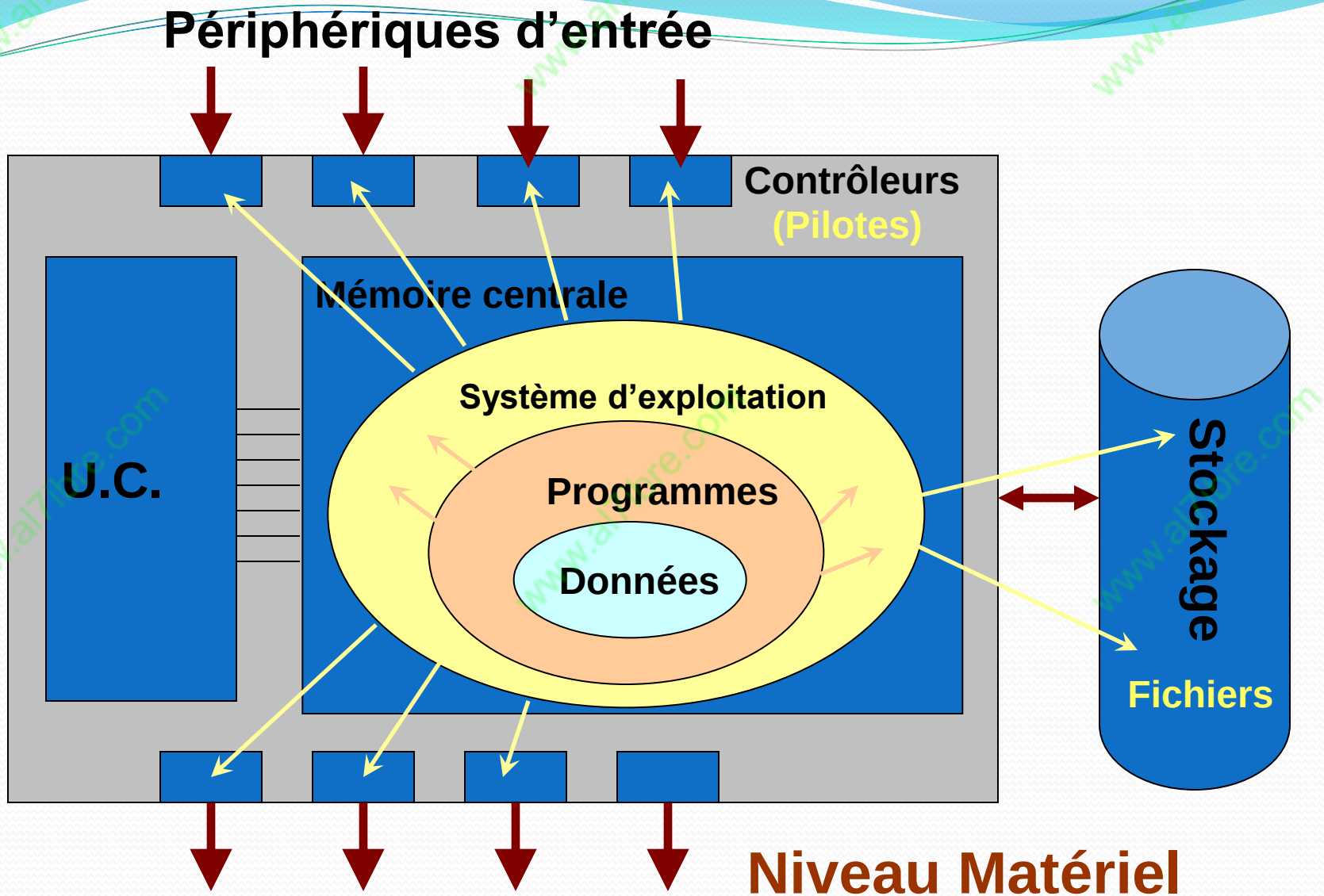
www.al7ibre.com



Niveau Matériel



Niveau Matériel + SE



Niveau Matériel
+ SE + Programmes

Périphériques d’entrée

- Permettent d’envoyer des informations à l’Unité Centrale

Périphériques de sortie

- Permettent d ’envoyer les résultats à l’extérieur de l’Unité Centrale
  - Écrans
    - taille (en pouce), résolution…
  - Imprimantes
    - matricielles, jet d ’encre, laser
  - Enceintes

www.al7ibre.com

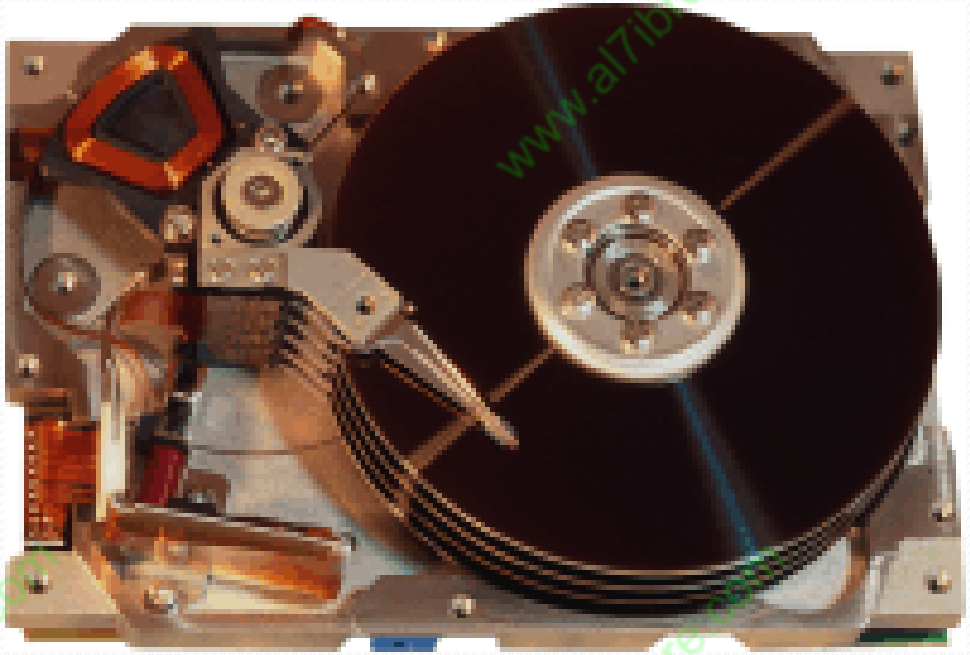
Les périphériques de stockage

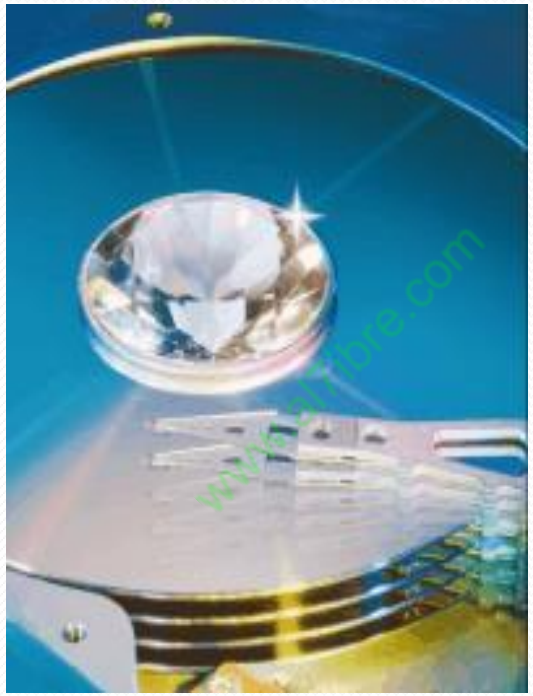
- CD-ROM
- DVD
- Disque dur > 320 Go
- Différence entre RAM et supports de stockage
- USB(***Universal Serial Bus)***

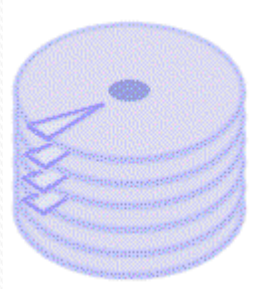
www.al7ibre.com

Les périphériques de stockage

- Capacité en Go actuellement
- Plusieurs têtes de lectures





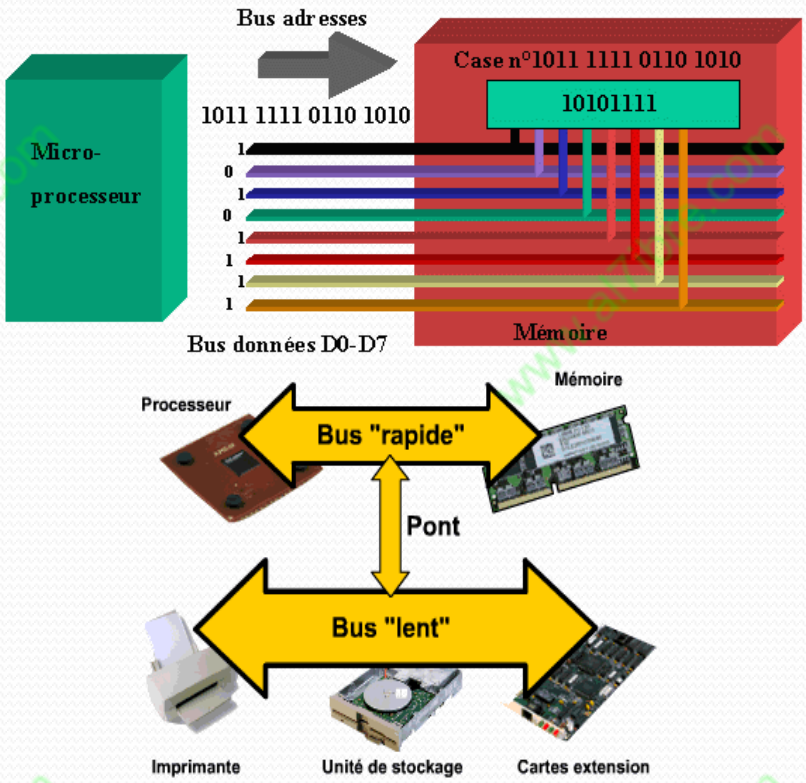


www.al7ibre.com

Les BUS

Permettent le transfert des données entre les composants de l'ordinateur

Différentes technologies → plus ou moins grande capacité de transfert



Système d'exploitation (SE)

- Fournit l'interface usager/machine:
  - Masque les détails du matériel aux applications
  - Le SE doit donc traiter ces détails
- Contrôle l'exécution des applications
  - Le fait en reprenant périodiquement le contrôle de l'UCT
  - Dit à l'UCT quand exécuter tel programme
- Il doit optimiser l'utilisation des ressources pour maximiser la performance du système

www.al7ibre.com

Quelques mots sur les systèmes d'exploitation

- Définition

  Un système d'exploitation (SE; en anglais: OS = operating system) est un ensemble de programmes de gestion du système qui permet de gérer les éléments fondamentaux de l'ordinateur:

Le matériel - les logiciels - la mémoire - les données – les réseaux.

- Logiciel très important…

  tout programme s'exécute sur un SE

Fonctions d'un système d'exploitation

- Gestion de la mémoire
- Gestion des systèmes de fichiers
- Gestion des processus
- Mécanismes de synchronisation
- Gestion des périphériques
- Gestion du réseau
- Gestion de la sécurité.

www.al7ibre.com

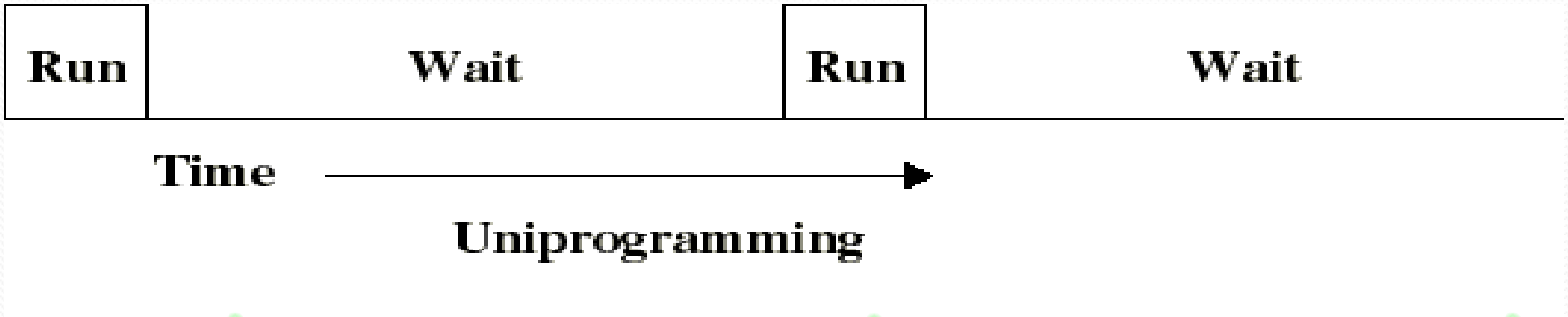
Ressources et leur gestion

- Ressources:
  - physiques: mémoire, unités E/S, UCT…
  - Logiques = virtuelles: fichiers et bases de données partagés, canaux de communication logiques, virtuels…
  - les ressources logiques sont bâties par le logiciel sur les ressources physiques
- Allocation de ressources: gestion de ressources, leur affectation aux usagers qui les demandent, suivant certains critères

www.al7ibre.com

Traitement par lots multiprogrammé

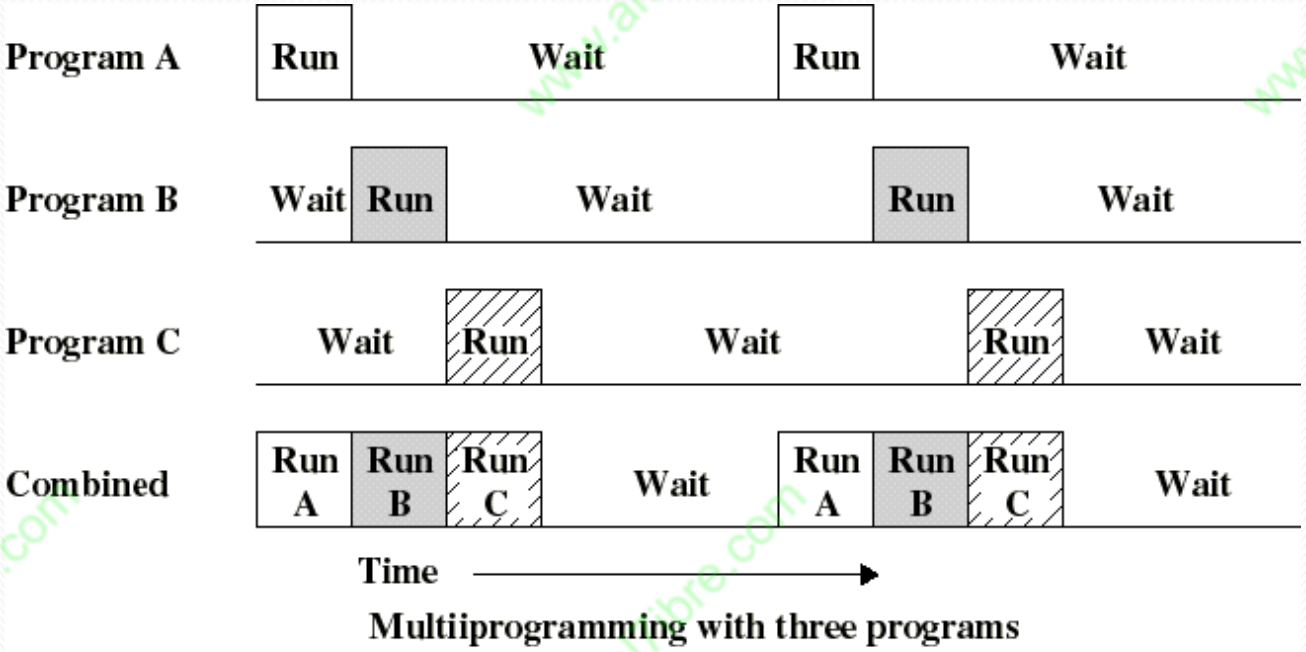
- Les opérations E/S sont extrêmement lentes (comparé aux autres instructions)
- Même avec peu d'E/S, un programme passe la majorité de son temps à attendre
- Donc: pauvre utilisation de l'UCT lorsqu'un seul programme usager se trouve en mémoire



www.al7ibre.com

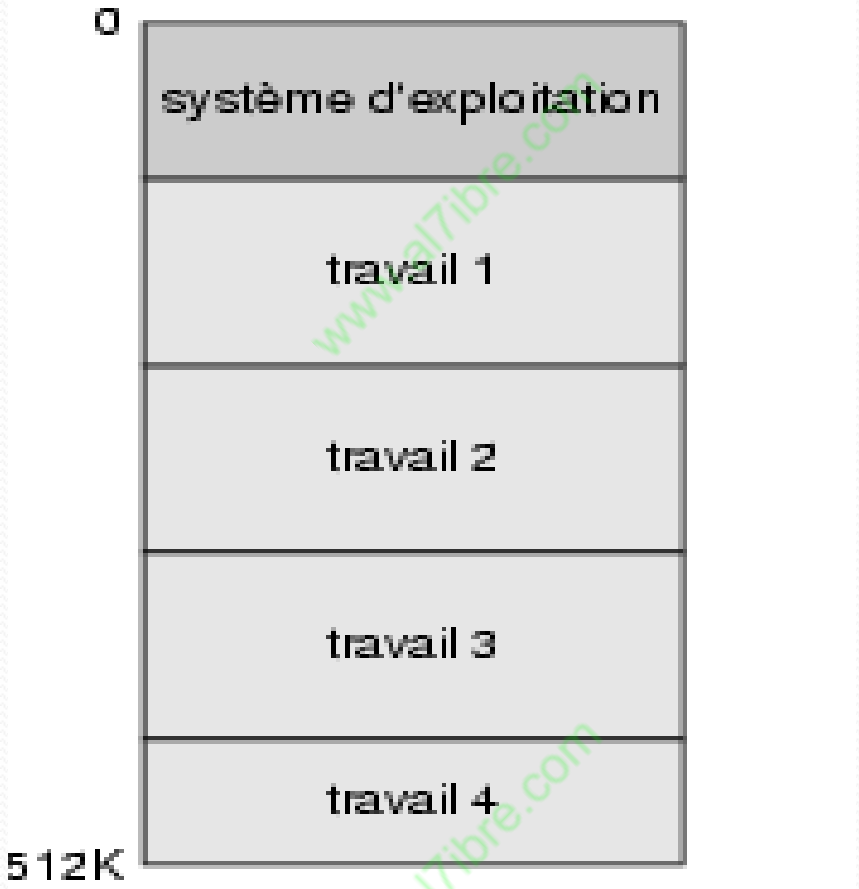
Traitement par lots multiprogrammé

- Si la mémoire peut contenir plusieurs programmes, l'UCT peut exécuter un autre programme lorsqu'un autre attend une E/S
- C'est de la multiprogrammation



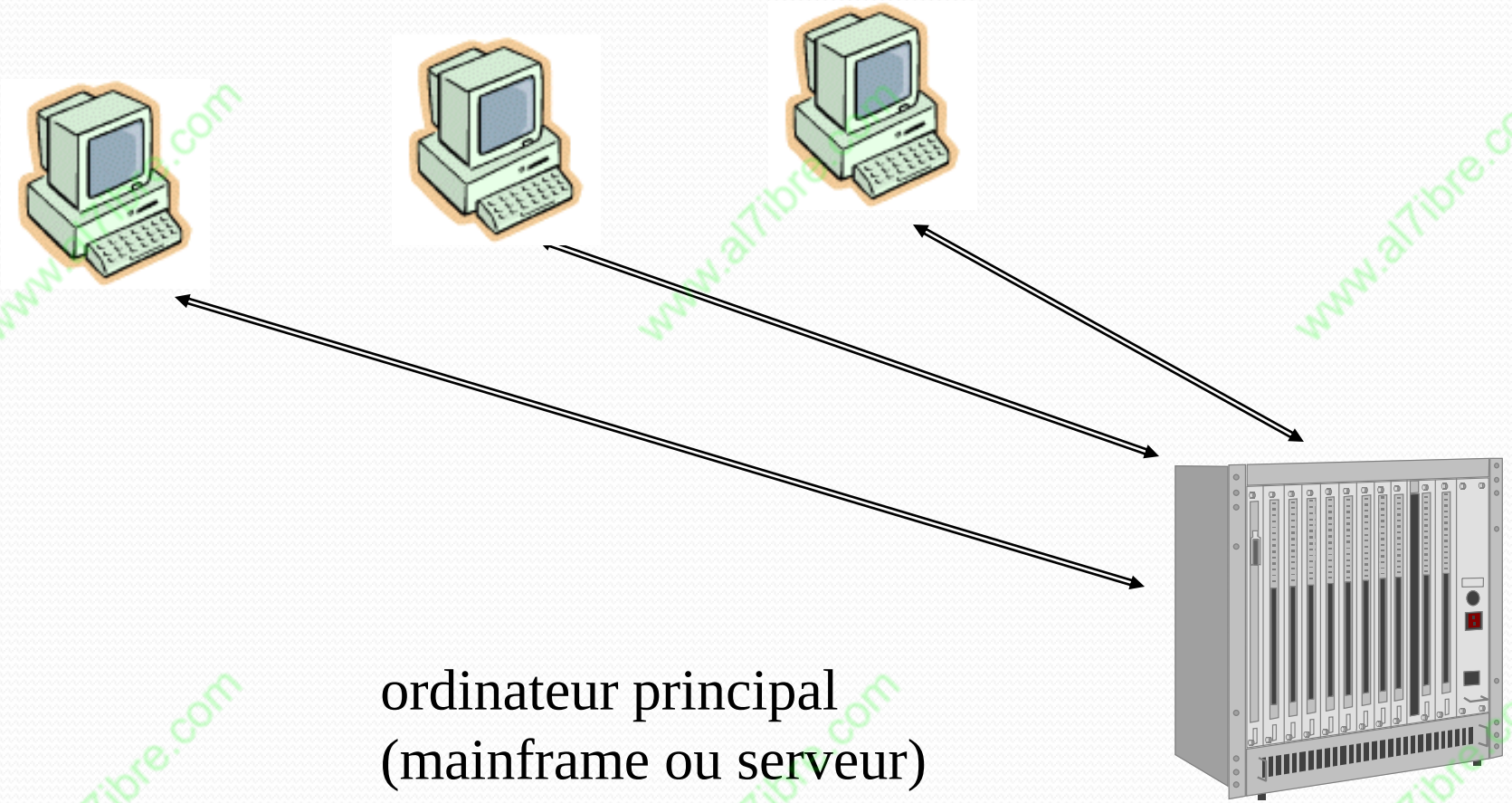
Multiiprogramming with three programs

Plusieurs programmes en mémoire pour la multiprogrammation



www.al7ibre.com

Terminaux
‘intelligents’ (PCs)’



ordinateur principal
(mainframe ou serveur)

Systèmes à temps partagé (TSS)

- Le traitement par lots multiprogrammé ne supporte pas l'interaction avec les usagers
  - **excellente utilisation des ressources mais frustration des usagers!**
- TSS permet à la multiprogrammation de desservir plusieurs usagers simultanément
- Le temps d'UCT est partagé par plusieurs usagers
- Les usagers accèdent simultanément et interactivement au système à l'aide de terminaux

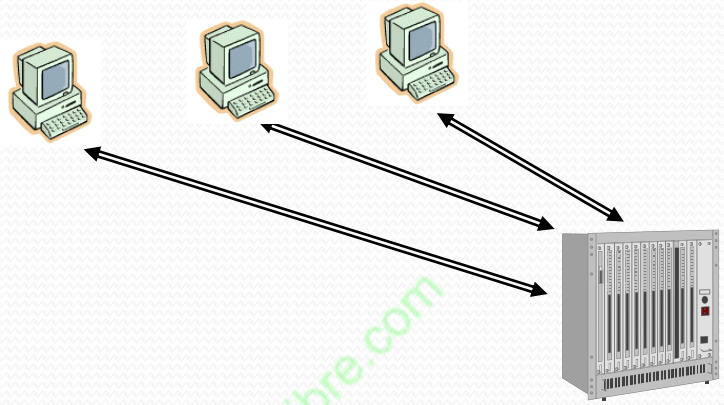
Systèmes à temps partagé (TSS)

- Le temps de réponse humain est lent: supposons qu'un usager nécessite, en moyenne, 2 sec du processeur par minute d'utilisation
- Environ 30 usagers peuvent donc utiliser le système sans délais notable du temps de réaction de l'ordinateur
- Les fonctionnalités du SE dont on a besoin sont les mêmes que pour les systèmes multiprogrammés, plus
  - la communication avec usagers
  - le concept de *mémoire virtuelle* pour faciliter la gestion de mémoire
  - traitement central des données des usagers (partagées ou non)

Retour aux concepts de TSS

- Plusieurs PC (clients) peuvent être desservis par un ordinateur plus puissant (serveur) pour des services qui sont trop complexes pour eux (clients/serveurs, bases de données, etc)

- Les grands serveurs utilisent beaucoup des concepts développés pour les systèmes TSS



www.al7ibre.com

Types de systèmes d'exploitation

- Ordinateur central (Mainframe)
  - Grande capacité d'E/S à cause du nombre d'usagers
  - Plus populaires avec l'augmentation de la vitesse des réseaux
  - Axé sur traitement de plusieurs jobs à la fois
    - Lot (batch) – jobs de routine comme la production d'un rapport
    - Transaction – faire des réservations
    - Partage de temps – Usagers qui accèdent une base de données

www.al7ibre.com

Types de systèmes d'exploitation

- Serveur
  - Permet le partage des ressources matériel et logiciel
  - Serveurs d'impressions, de fichiers, Web
- Multiprocesseur
  - Normalement une variation d'un SE pour serveur
  - Permets à plusieurs processeurs à travailler ensemble
    - Plusieurs processeurs sur la même carte

www.al7ibre.com

Types de systèmes d'exploitation

- SE pour ordinateur personnel
  - Donne une interface à un simple usager
  - Windows, Linux, Macintosh
- Système d'exploitation temps-réel
  - Unique parce que les programmes ont des contraintes temporelles (deadlines): temps réel dur (avion)
- SE embarqués
  - Similaire au SE temps-réel
  - Assistant numérique personnel (PDAs), Contrôleur de tableau de bord automobile, Gameboy
  - Ont des préoccupations que les autres SE n'ont pas: encombrement, puissance, mémoire.

www.al7ibre.com

Types de systèmes d'exploitation

- Les SE Smart Card
  - Similaire à embarqués
  - Opère sur les cartes de la grosseur d'une carte de crédit avec un processeur
  - Contraintes sévères de mémoire et de puissance de calcul
- Systèmes d'exploitation répartis:
  - Le SE exécute à travers un ensemble de machines qui sont reliées par un réseau

www.al7ibre.com

Principes des systèmes d'exploitations

Processus

www.al7ibre.com

Processus

- Le processus est ***LE*** concept central dans les systèmes d'exploitation
- Un processus est une abstraction d'un programme en exécution
  - Auquel on a donné des ressources
- Un certain nombre de processus (2 ou plus) s'exécutant en même temps forment un système multitâche, multithread, ou multiprogrammé.
- Est-ce que ces programmes s'exécute vraiment en parallèle?

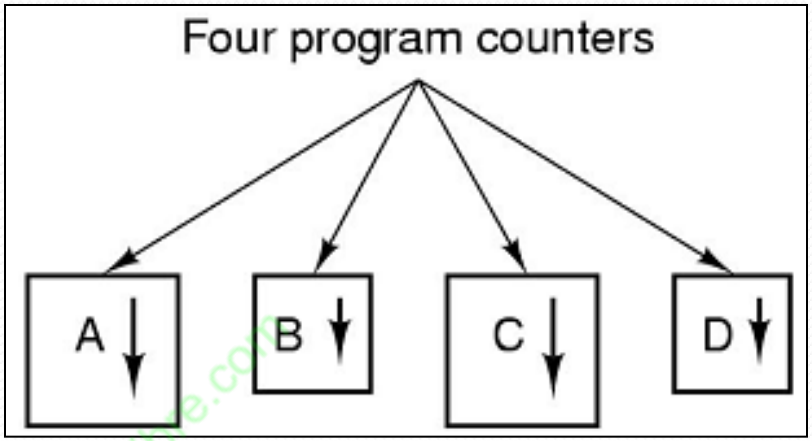
www.al7ibre.com

Processus

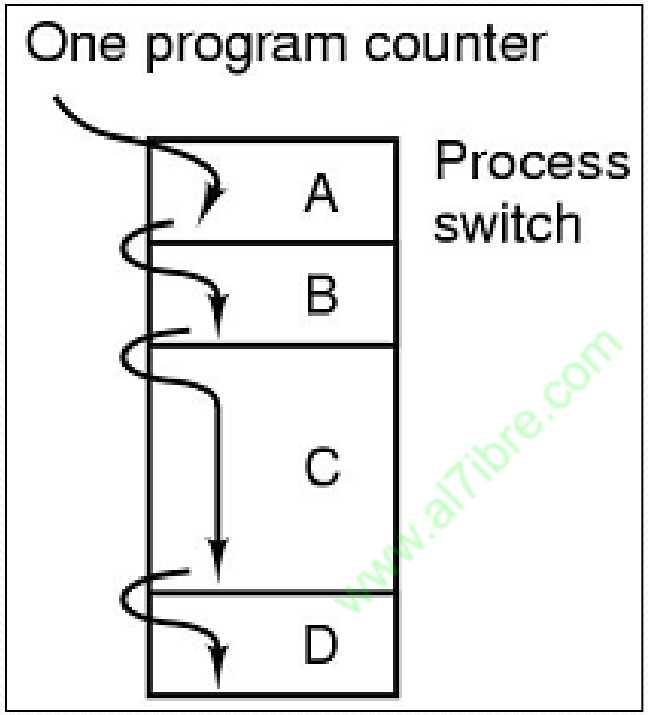
- Le model des processus est tout simplement l'idée que tout logiciel qui est exécutable est organisé en un nombre de processus séquentiels incluant le SE.
- **Pseudo-parallélisme** est parfois utilisé pour référer à des processus multiples s'exécutant sur un seul processeur
- Ceci diffère du parallélisme sur un système à multiprocesseurs

www.al7ibre.com

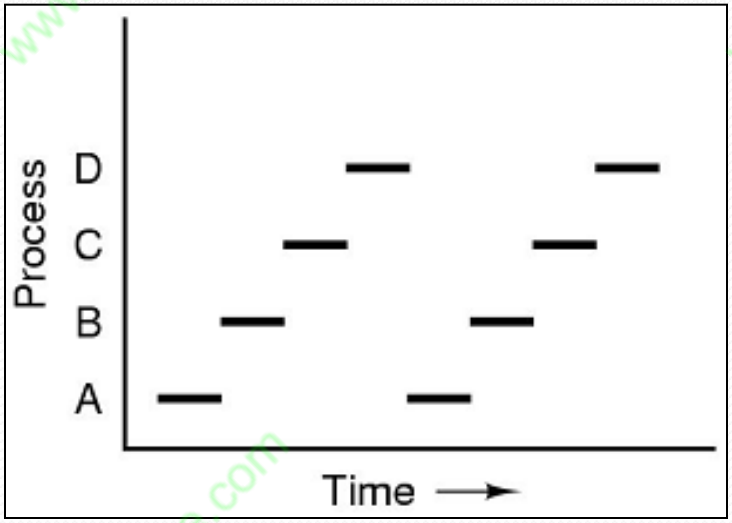
Processus



Conceptuel



Écoulement du prog



Temps d'exécution des processus

www.al7ibre.com

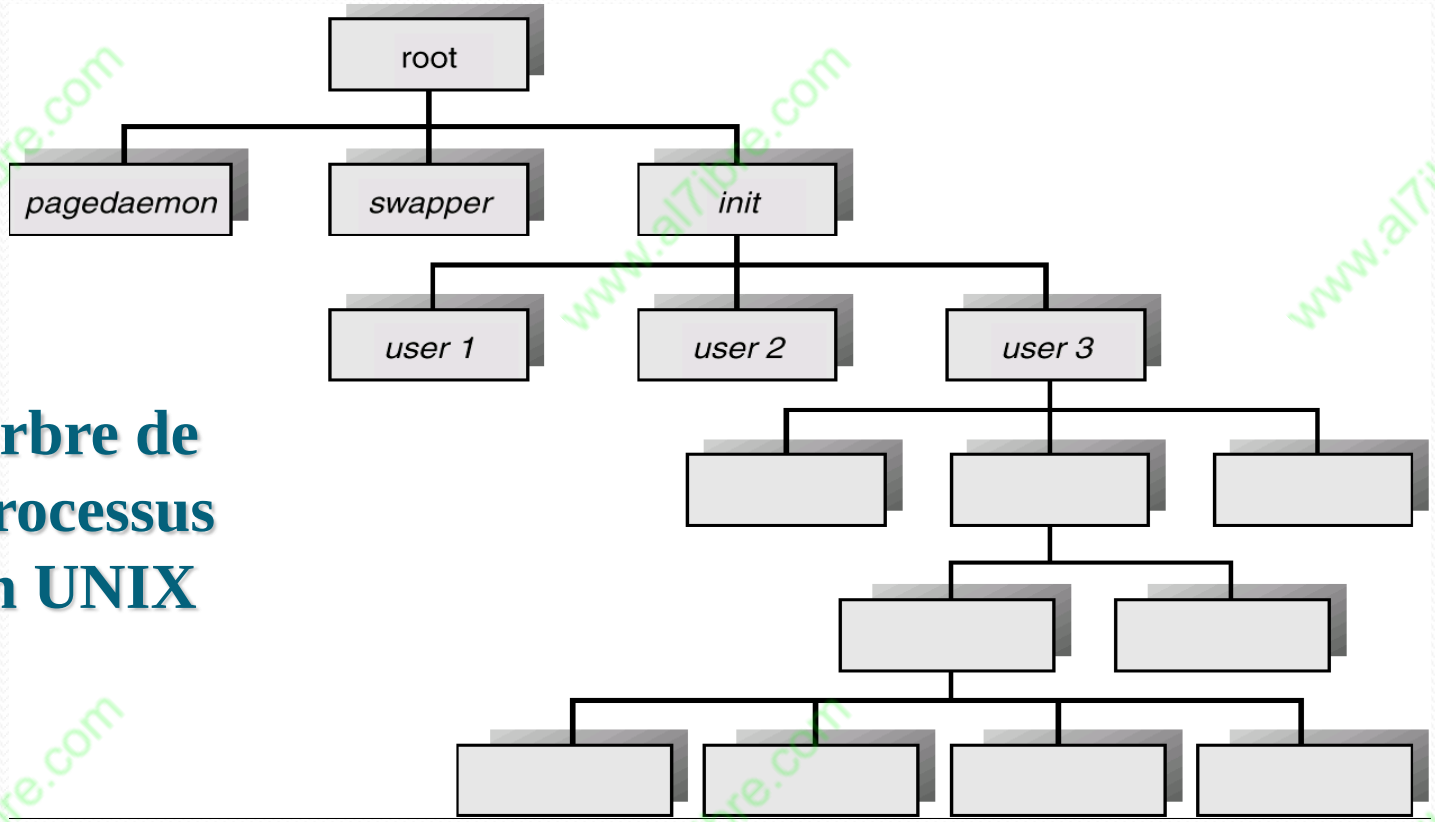
Création de Processus

- Les systèmes communs ont besoin de créer des nouveaux processus durant l'opération.
  - Initialisation du système
  - Création de processus appelé par un processus en exécution
  - Demande d'un utilisateur pour créer un processus
  - Initialisation d'un job batch

www.al7ibre.com

Création de processus

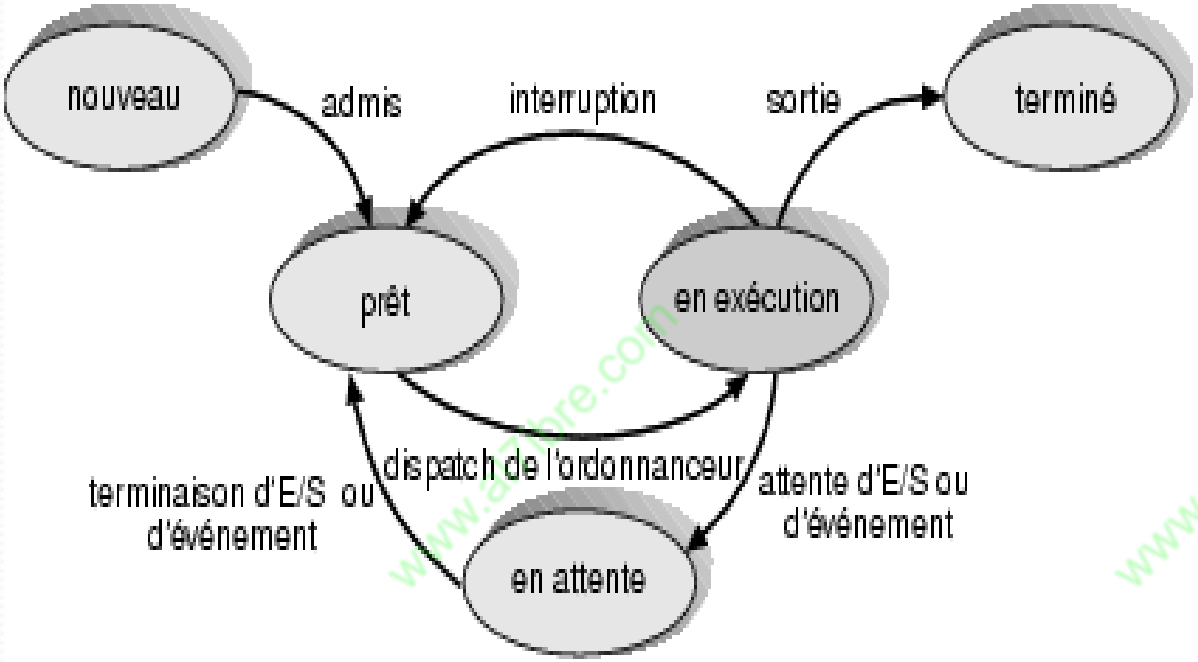
- Les processus peuvent créer d'autres processus, formant une hiérarchie (instruction fork ou semblables)



Arbre de processus en UNIX

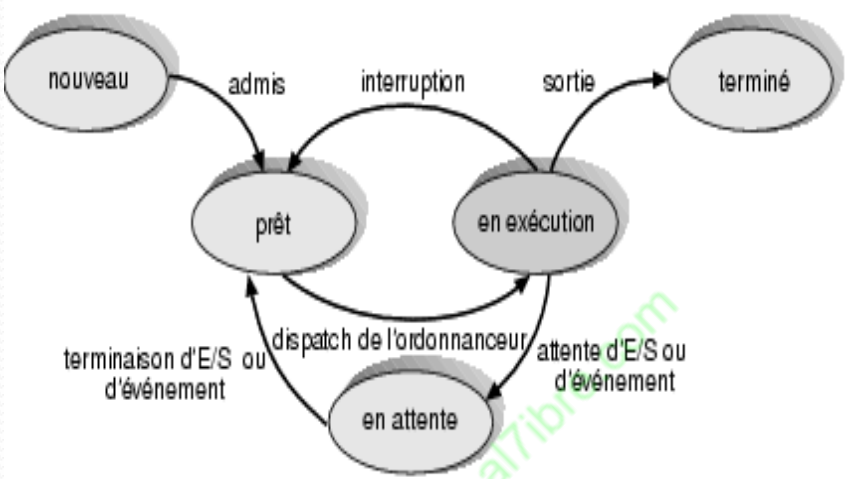
www.al7ibre.com

États des processus



- Quand le processus est en exécution 'running' il utilise le CPU pour faire son travail
- Quand le processus est prêt 'ready', il voudrait s'exécuter mais le CPU est alloué à un autre.

États des processus



- En état attente ‘blocked’, le processus ne peut s’exécuter parce qu’il attend après une condition (entrée/sortie, expiration du chrono, attente d’un autre processus etc…)
- Des fois, des commandes sont appelées pour entrer dans l’état bloqué (block, pause,wait).
- Parfois le système cause la transition automatiquement à l’état prêt ‘ready’ (ordonnancement)

Implémentation des Processus

- En multiprogrammation, un processus s'exécute sur l'UCT de façon intermittente
- Chaque fois qu'un processus reprend l'UCT (transition prêt → exécution) il doit la reprendre dans la même situation où il l'a laissée (même contenu de registres UCT, etc.)
- Donc au moment où un processus sort de l'état exécution il est nécessaire de sauvegarder ses informations essentielles, qu'il faudra récupérer quand il retourne à cet état

www.al7ibre.com

Implémentation des Processus

- Comment le SE implémente le model des processus?
  - La table des processus, un tableau ou structures avec une entrée par proces
  - Information typique:
    - État des Processus
    - Compteur ordinal
    - Pointeur de pile
    - Allocation de la mémoire
    - État des fichiers ouverts
    - Information de gestion/ordonnancement
    - Et encore plus!

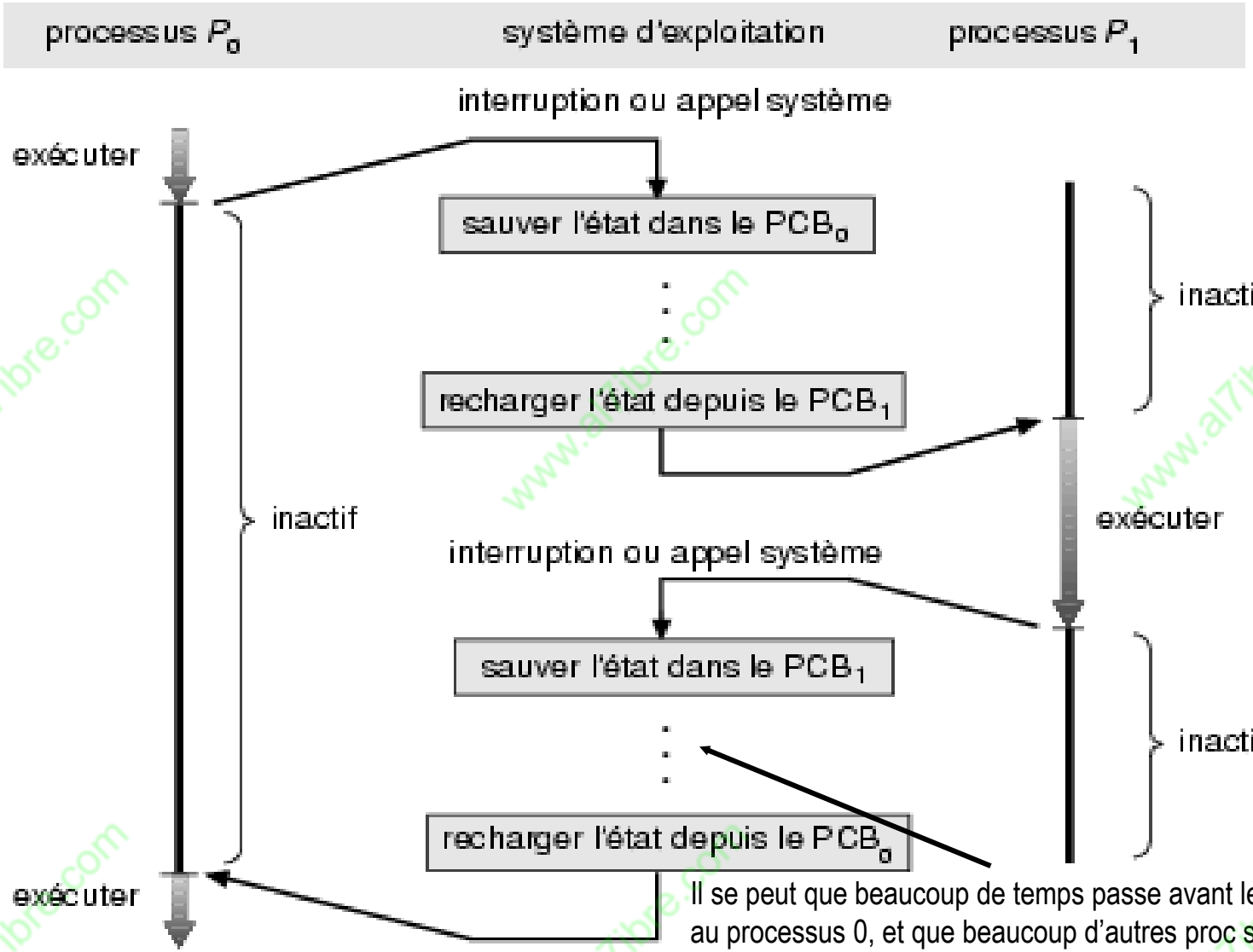
PCB = Process Control Block

Commutation de processus

- Aussi appelé
  - commutation de contexte
  - Changement de contexte
  - context switching
- Quand l'UCT passe de l'exécution d'un processus 0 à l'exécution d'un proc 1, il faut
  - mettre à jour le PCB de **0**
  - sauvegarder le PCB de **0**
  - reprendre le PCB de **1**, qui avait été sauvegardé avant
  - remettre les registres d'UCT, compteur d 'instructions etc. dans la même situation qui est décrite dans le PCB de **1**

www.al7ibre.com

Commutation de processeur (context switching)



Il se peut que beaucoup de temps passe avant le retour au processus 0, et que beaucoup d'autres proc soient exécutés entre temps

Commutation de processus

Comme l'ordinateur n'a, la plupart du temps, qu'un processeur, il résout ce problème grâce à un pseudo-parallélisme

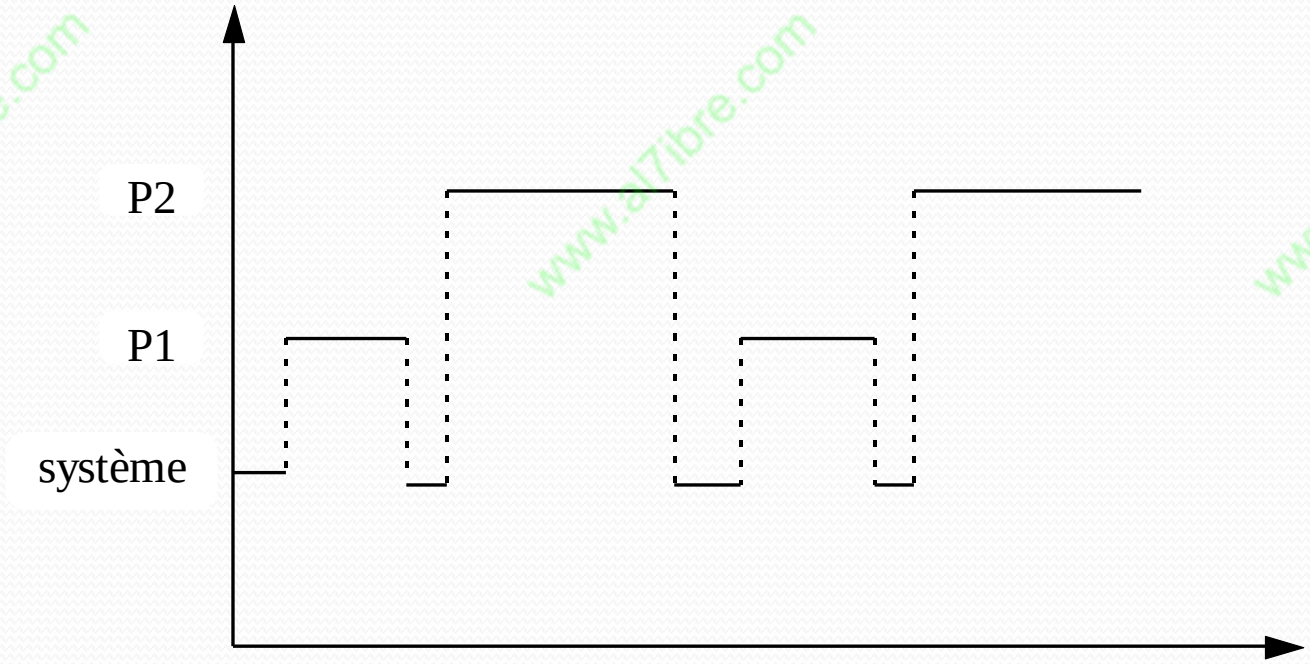
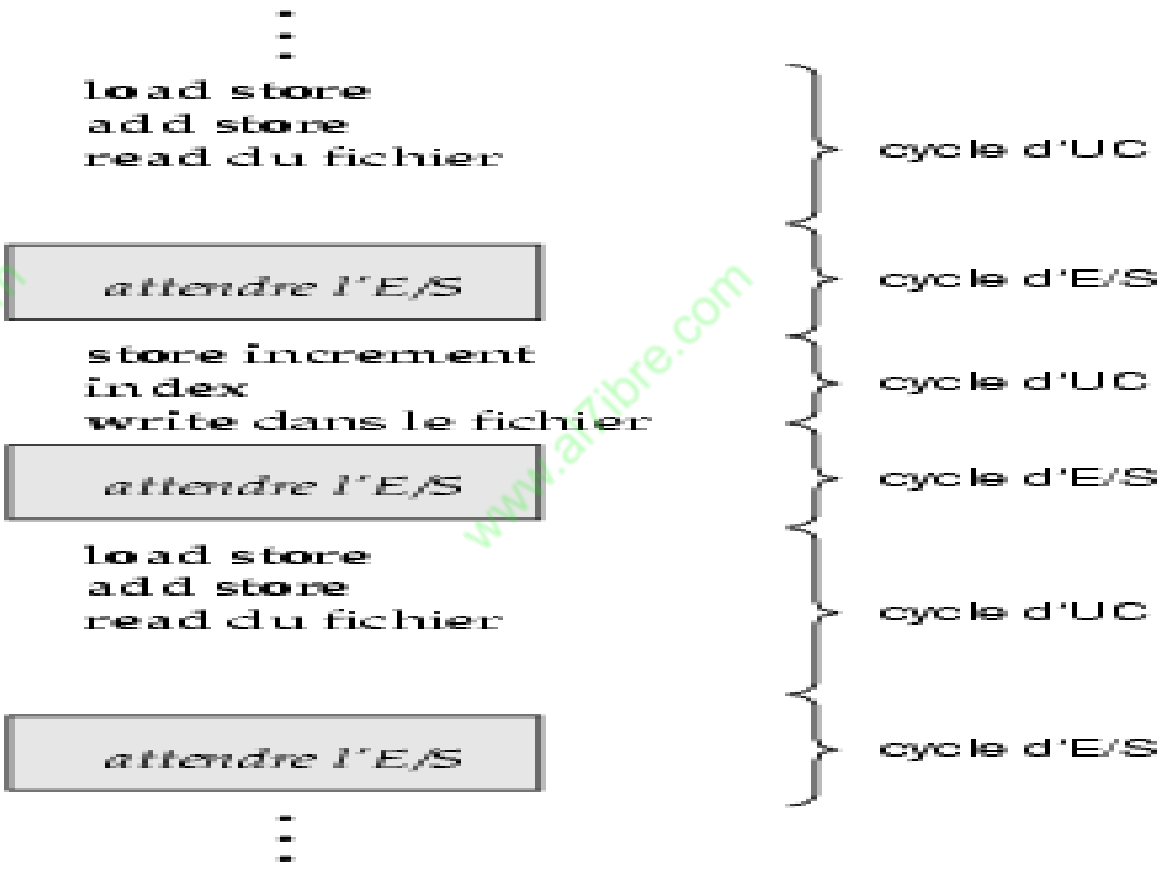


Figure 1 Le multi-tâche

www.al7ibre.com

Les cycles d'un processus



- Cycles (bursts) d'UCT et E/S: l'exécution d'un processus consiste de séquences d'exécution sur UCT et d'attentes E/S

Notion d'ordonnancement

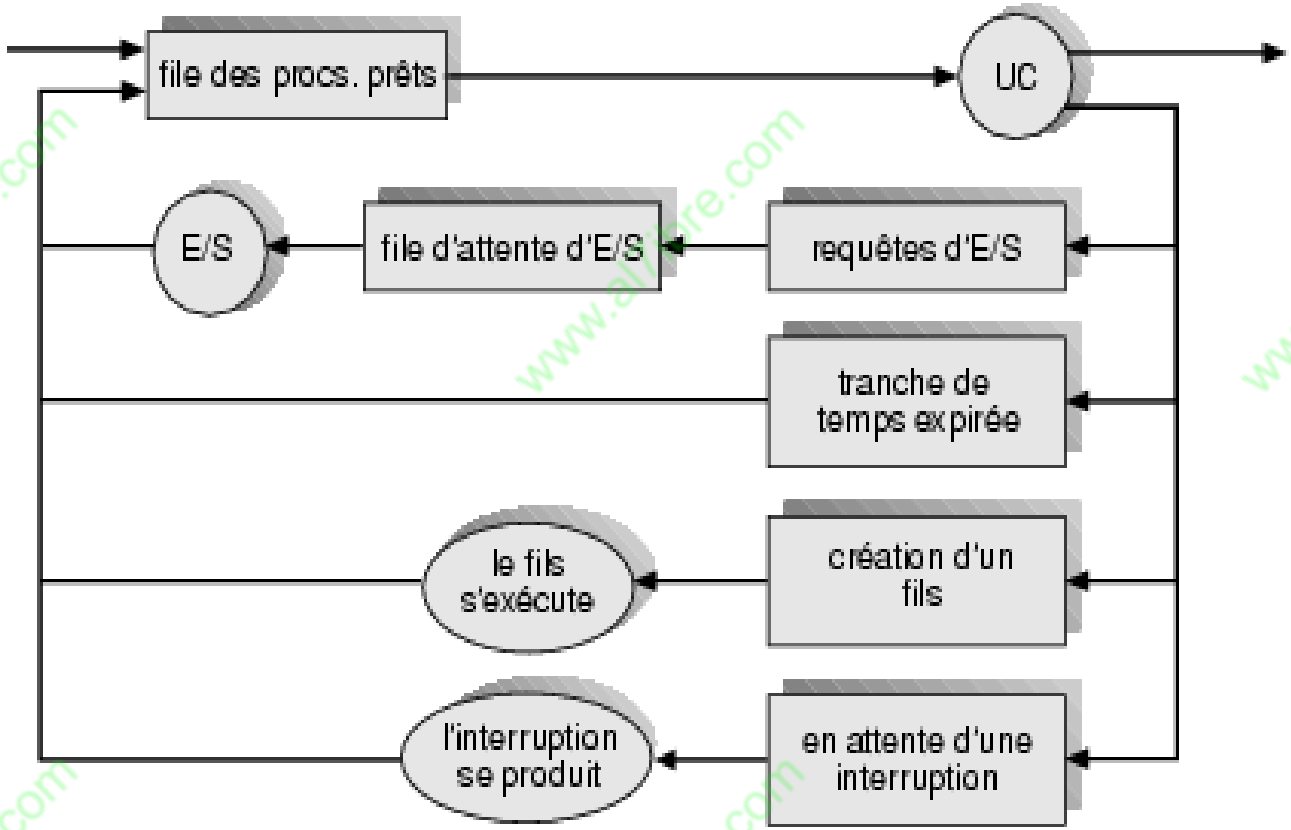
- L'ordonnancement sur les systèmes d'exploitation initiaux était facile: exécute le prochain programme sur le ruban magnétique
- Les ordinateurs personnels ont changé cet environnement parce ce que nous donnons la priorité au processus qui interagit avec l'utilisateur
- Avec plusieurs processus qui sont en compétition pour les précieux cycles du CPU, une décision doit être faite pour savoir quel processus va s'exécuter
- L'**ordonnanceur** est la partie du système d'exploitation qui fait ce choix, basé sur un **algorithme d'ordonnancement**
- L'ordonnanceur peut faire une très grande différence dans la performance qui est perçu par l'utilisateur

www.al7ibre.com

Ordonnanceurs (schedulers)

- Trois types d`ordonnanceurs :
  - À court terme: sélectionne quel processus doit exécuter la transition prêt → exécution. Il est exécuté très souvent (millisecondes)
    - doit être très efficace
  - À long terme: sélectionne quels processus peuvent exécuter la transition nouveau → prêt. Il doit être exécuté beaucoup plus rarement
    - Il contrôle le niveau de multiprogrammation
  - À moyen terme: le manque de ressources peut parfois forcer le SE à *suspendre* des processus, il sélectionne donc quels processus sortir temporairement de la mémoire pour palier au manque de celle-ci
    - ils ne seront plus en concurrence avec les autres pour les ressources
    - ils seront repris plus tard quand les ressources deviendront disponibles
    - Ces processus sont enlevés de mémoire centrale et mis en mémoire secondaire, pour être repris plus tard
    - 'swap out', 'swap in' , va-et-vient

Ordonnanceur court terme



Ordonnancement des Processus

- La multiprogrammation est conçue pour obtenir une utilisation maximale des ressources, surtout l'UCT
- L'ordonnanceur UCT est la partie du SE qui décide quel processus dans la file ready/prêt obtient l'UCT quand elle devient libre
- L'ordonnanceur doit viser à une utilisation optimale de l'UCT

www.al7ibre.com

Buts des algorithmes d'ordonnancement

- Il y a normalement plusieurs processus dans la file d'attente des processus prêt
- Quand l'UCT devient disponible, lequel choisir?
- L'idée générale est d'effectuer le choix dans l'intérêt de l'efficacité d'utilisation de la machine
- Mais cette dernière peut être jugée selon différents critères…

www.al7ibre.com

Buts des algorithmes d'ordonnancement

- Utilisation UCT: pourcentage d'utilisation
- Débit = Throughput: nombre de processus qui complètent dans l 'unité de temps
- Temps de rotation = turnaround: le temps pris par le proc de son arrivée à sa terminaison.
- Temps d'attente: attente dans la file prêt (somme de tout le temps passé en file prêt)
- Temps de réponse (pour les systèmes interactifs): le temps entre une demande et la réponse

www.al7ibre.com

Critères d'ordonnancement

- Utilisation UCT:
  - à maximiser
- Débit (Throughput):
  - à maximiser
- Temps de rotation (turnaround):
  - à minimiser
- Temps d'attente:
  - à minimiser
- Temps de réponse
  - à minimiser

www.al7ibre.com

Files d’attente

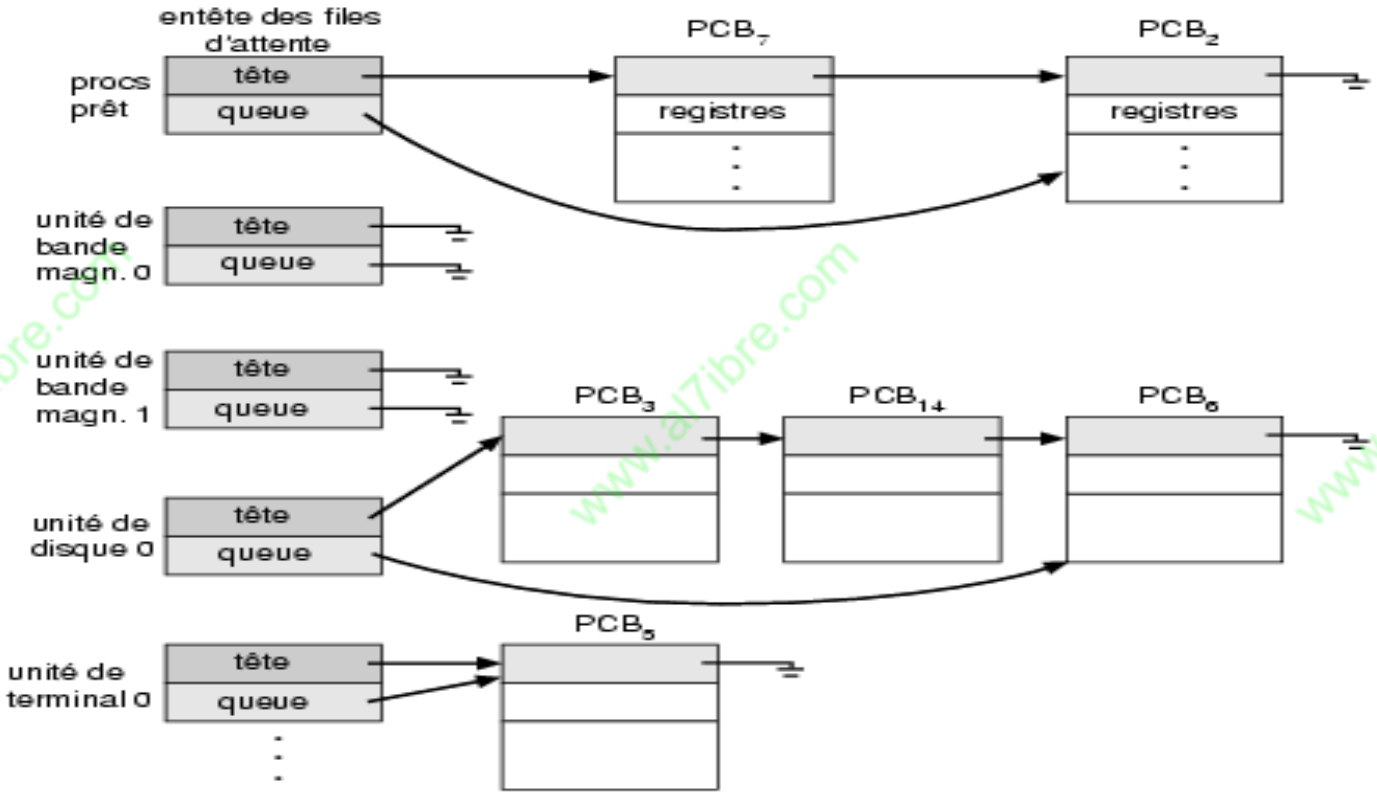
- Les ressources d’ordinateur sont souvent limitées par rapport aux processus qui en demandent
- Chaque ressource a sa propre file de processus en attente
- En changeant d’état, les processus se déplacent d ’une file à l’autre
  - File prêt: les processus en état prêt=ready
  - Files associés à chaque unité E/S
  - etc.

www.al7ibre.com

Files d'attente

Ce sont les PCBs qui sont dans les files d'attente

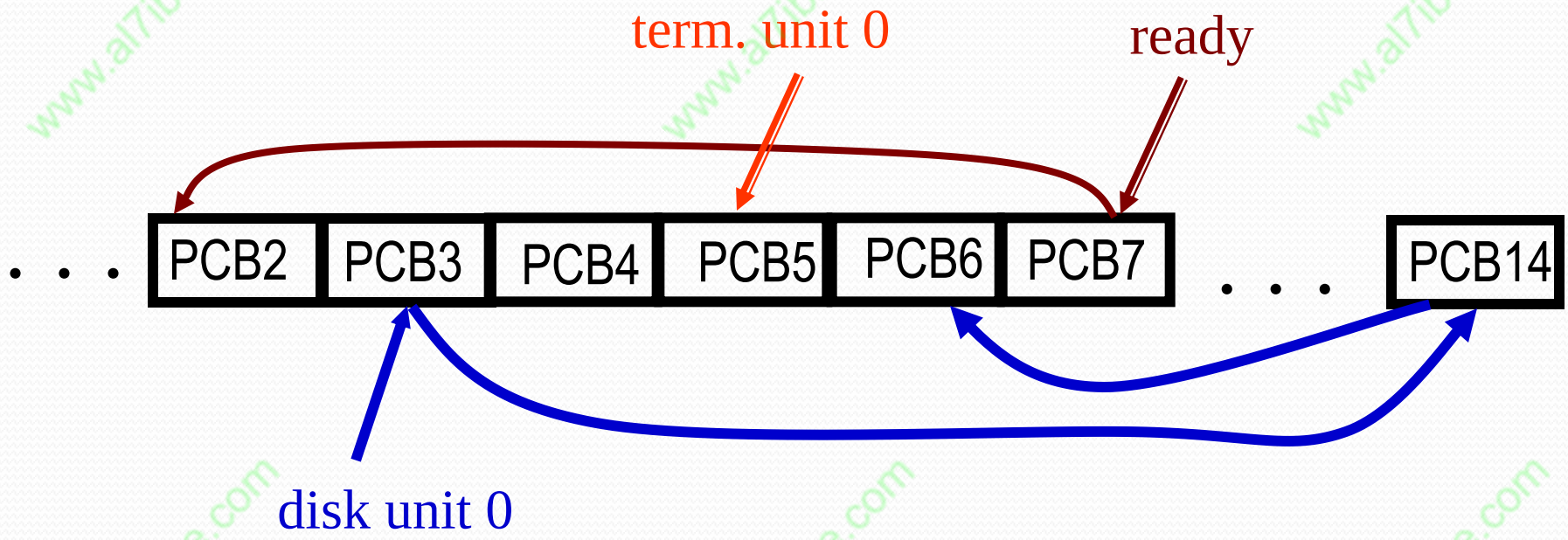
File des prêts



Nous ferons l'hypothèse que le premier processus dans une file est celui qui utilise la ressource: ici, processus 7 s'exécute, processus 3 utilise disque 0, etc.

Les PCBs

- Les PCBs ne sont pas déplacés en mémoire pour être mis dans les différentes files:
  ce sont les pointeurs qui changent.



Catégories d'algorithmes d'ordonnancement

Différent algorithmes sont utilisés sur différents systèmes.

Trois catégories majeurs pour algorithmes :

- Systèmes de lots
  - Séries de programmes qui attendent pour exécution
- Systèmes interactifs
  - Utilisateurs aux terminaux qui attendent pour leurs réponses
- Systèmes en temps réel
  - Exécutent généralement dans un environnement où les processus coopèrent pour finir une tâche

www.al7ibre.com

Ordonnancement des systèmes par lots

- Premier-arrivé premier-servis(PAPS)

  First-Come First-Served(FCFS)

  - L'algorithme le plus simple
  - Non-préemptif (sans réquisition)
  - Une seule file de processus prêts
  - Chacun a le CPU et s'exécute jusqu'à ce qu'il bloque
  - Avantages:
    - Facile à comprendre et implémenter
    - impartial

www.al7ibre.com

Premier-arrivé premier-servis(PAPS) – First-Come First-Served(FCFS)

- Non-préemptif
- Applicable pour les jobs de grandeur connue

Exemple:

Processus	Temps de cycle
P1	24
P2	3
P3	3

Si les processus arrivent au temps 0 dans l'ordre: P1 , P2 , P3 Le diagramme Gantt est:

Temps d'attente pour P1= 0; P2= 24; P3= 27

Temps attente moyen: (0 + 24 + 27)/3 = 17

Tenir compte du temps d'arrivée!

- Dans le cas où les processus arrivent à des moments différents, il faut soustraire le temps d'arrivée
  - P0 arrive à temps 0
  - P1 arrive à temps 2
  - P3 arrive à temps 5

Temps d'attente pour P1= 0; P2= 24-2=22; P3= 27-5=22

Temps attente moyen: (0 + 22 + 22)/3 = 14,6 < 17

www.al7ibre.com

Premier-arrivé premier-servis(PAPS) – First-Come First-Served(FCFS)

Si les mêmes processus arrivent à l'instant 0 mais dans l'ordre

$$P_2 , P_3 , P_1 .$$

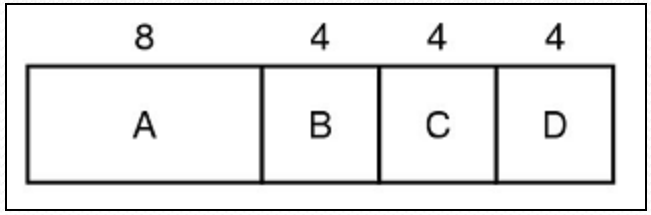
Le diagramme de Gantt est:

- Temps d'attente pour $P_1 = 6$ $P_2 = 0$ $P_3 = 3$
- Temps moyen d'attente: $(6 + 0 + 3)/3 = 3$
- Beaucoup mieux!
- Donc pour cette technique, le temps d'attente moyen peut varier grandement

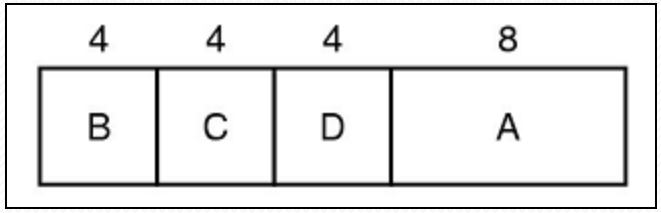
Ordonnancement des systèmes de lots

Avec l'exemple précédent et le suivant on voie, qu'en exécutant les plus court en premier on obtient un meilleur temps d'attente ou de réponse moyen d'où:

- L'algorithme du plus petit job en premier - Shortest Job First
  - Non-préemptif
  - Applicable pour les jobs de grandeur connue, ie: différent types de réclamations dans une compagnie d'assurance
  - Les processus mit dans cet ordre produisent le plus petit temps de réponse (généralement)



Temps de réponse moyen
= [8+(8+4)+(8+4+4)+(8+4+4+4)]/4
= 14



Temps de réponse moyen
= [4+(4+4)+(4+4+4)+(4+4+4+8)]/4
= 11

Shortest Job First (SJF)

- Notez que le système est optimale quand tout les travaux sont disponibles en même temps
  - Si un long job est en exécution quand un nombre de petits travaux arrivent, le délais restant du job long est ajouté à tous les nouveaux travaux
- Est-ce qu'il y a une solution à ce problème?
- Le temps du premier processus a plus d'impacte sur le temps de réponse de tous les autres d'où:

SJF avec préemption (réquisition)= SRTF

www.al7ibre.com

Shortest Remaining Time First (SRTF)

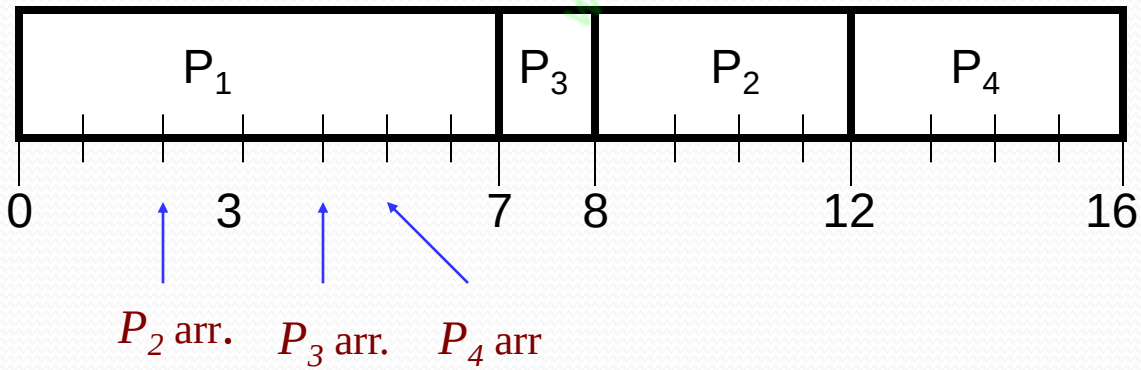
- Le plus petit temps restant est le Prochain -Shortest Remaining Time First (ou Next)
  - Similaire à SJF, mais préemptif
  - Si un job qui arrive a besoin de moins de temps pour compléter que le job en court, on commence à exécuter le nouveau job
  - Donne un bon service au nouveaux jobs qui sont courts

www.al7ibre.com

Example de SJF

Processus	Arrivée	Cycle
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

- **SJF (sans préemption)**



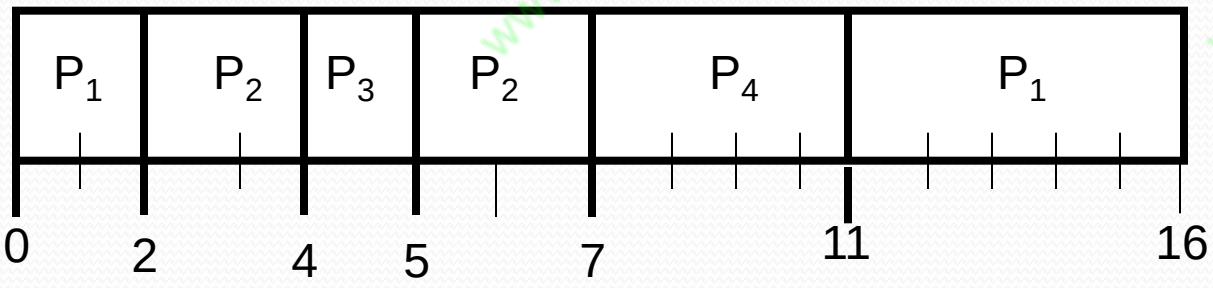
- **Temps d'attente moyen = (0 + 6 + 3 + 7)/4 = 4**

www.al7ibre.com

Exemple de SRTF=SJF avec préemption

Processus	Arrivée	Cycle
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

- **SJF (préemptive, avec réquisition)**



P_2 arr. P_3 arr. P_4 arr

- **Temps moyen d`attente = (9 + 1 + 0 +2)/4 = 3**
  - **P1 attend de 2 à 11, P2 de 4 à 5, P4 de 5 à 7**

Le plus court d'abord SJF: critique

- Difficulté d'estimer la longueur à l'avance
- Les processus longs souffriront de *famine* lorsqu'il y a un apport constant de processus courts
- La préemption est nécessaire pour environnements à temps partagé
- Un processus long peut monopoliser l'UCT s'il est le 1er à entrer dans le système et il ne fait pas d'E/S
- Il y a assignation implicite de priorités: préférences aux travaux plus courts

www.al7ibre.com

Ordonnancement des systèmes interactifs

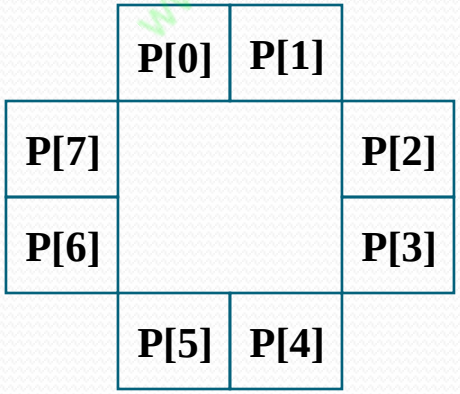
- Quelqu'un quelque part est assit à un clavier et entre de l'information
- On se rappelle de nos objectifs:
  - Temps de réponse rapide
  - Proportionnalité (Les choses que les usagers *pensent* qui devraient prendre peu de temps; le devraient)
- Tout les algorithmes pour les systèmes interactifs peuvent être utilisés dans les systèmes de lots

www.al7ibre.com

Tourniquet = Round-Robin (RR)

- Chaque processus se voit alloué une tranche de temps appelé quantum de temps (p.ex. 10-100 millisecondes.) pour s'exécuter (*tranche de temps*)
- S'il s'exécute pour un quantum entier sans autres interruptions, il est interrompu par la minuterie et l 'UCT est alloué à un autre processus
- Le processus interrompu redevient prêt (à la fin de la file)
- Méthode préemptive

La file prêt est un cercle (dont RR)

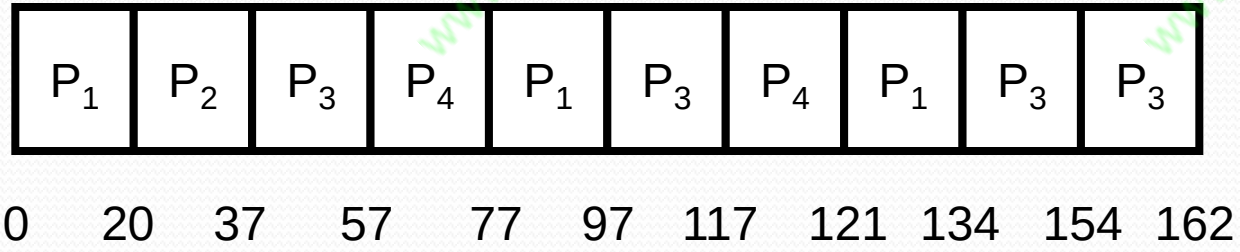


- Seul décision d'implémentation :

combien de temps est assigné pour un quantum?

Exemple: Tourniquet Quantum = 20

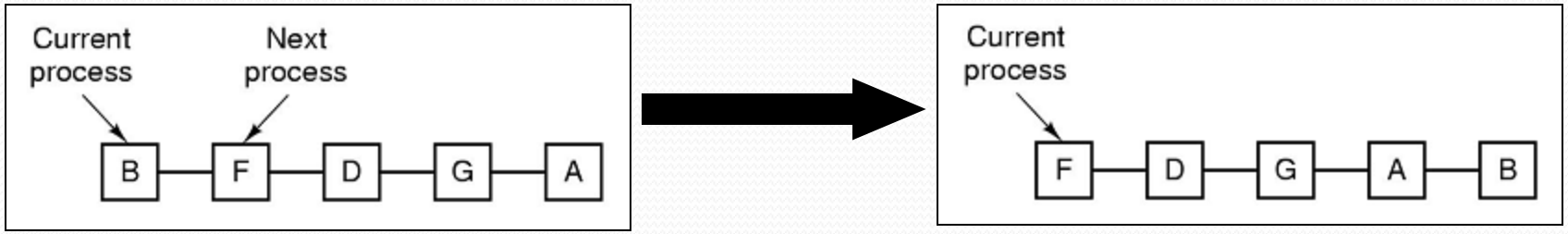
Processus	Cycle
P_1	53
P_2	17
P_3	68
P_4	24



- Normalement,
  - temps de rotation (turnaround) plus élevé que SJF
  - mais temps attente moyen meilleur – contrôlez!

www.al7ibre.com

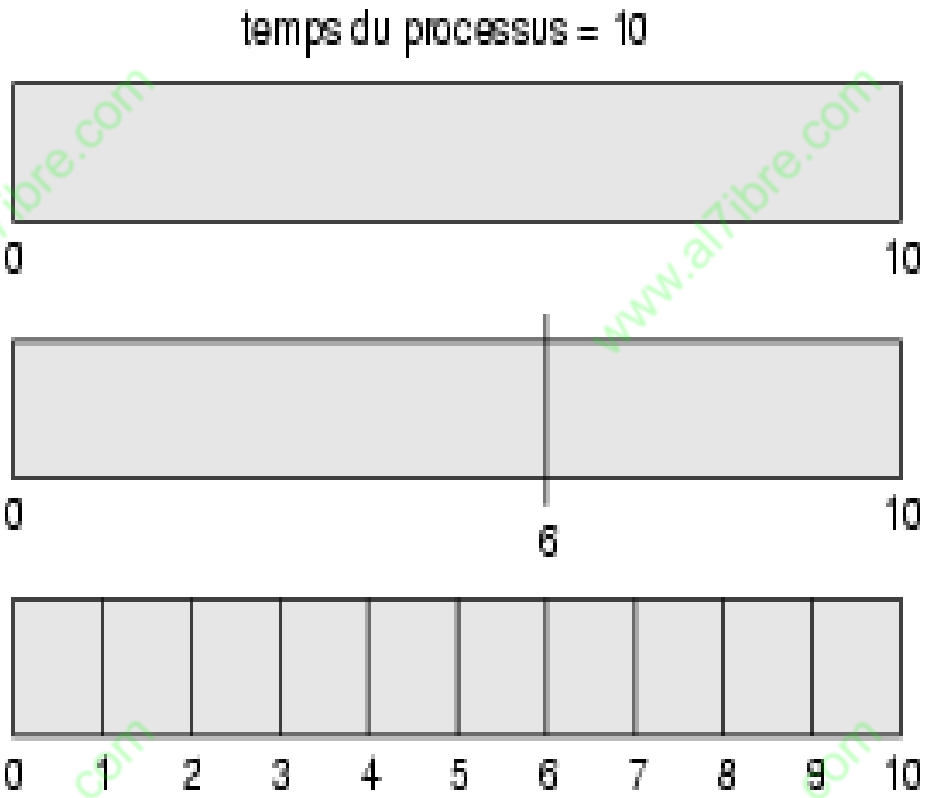
Ordonnancement des systèmes interactifs



- Ordonnancement de type tourniquet
  - Combien de temps dans un quantum?
    - Le changement de processus prend du temps: sauvegarde/charge le PCB
    - Pour un changement de contexte qui prend 1 ms combien de temps devrait être un quantum? Considérez l'overhead:

Longueur du quantum	1ms	4ms	10ms	20ms	50ms
Overhead	50%	20%	9.1%	4.8%	2%

Un petit quantum augmente les commutations de contexte (l'overhead)



tranches de temps	changement de contexte
12	0
6	1
1	9

Ordonnancement des systèmes interactifs

- Ordonnancement par priorité
  - Basé sur les utilisateurs: Si nous partagions un mainframe alors l'ordre des priorités pourrait être: chercheurs, étudiants,…
  - Basé sur les processus: même sur un PC avec utilisateur unique il va y avoir des processus multiples. Qu'est-ce qui est plus prioritaire: le mouvement de la souris ou envoyer les courriels?
- Comment est-ce que l'on assigne les priorités?
  - Statiquement: chaque processus qui est créé par un processus à priorité 'X' se voit assigné la priorité 'X'
  - Dynamiquement: Réagit pour donner au processus interactifs une priorité accrue
    - Met la priorité à $1/f$ où f est la fraction du dernier quantum qu'un processus a utilisé … les commandes courtes font qu'un processus a une plus haute priorité

www.al7ibre.com

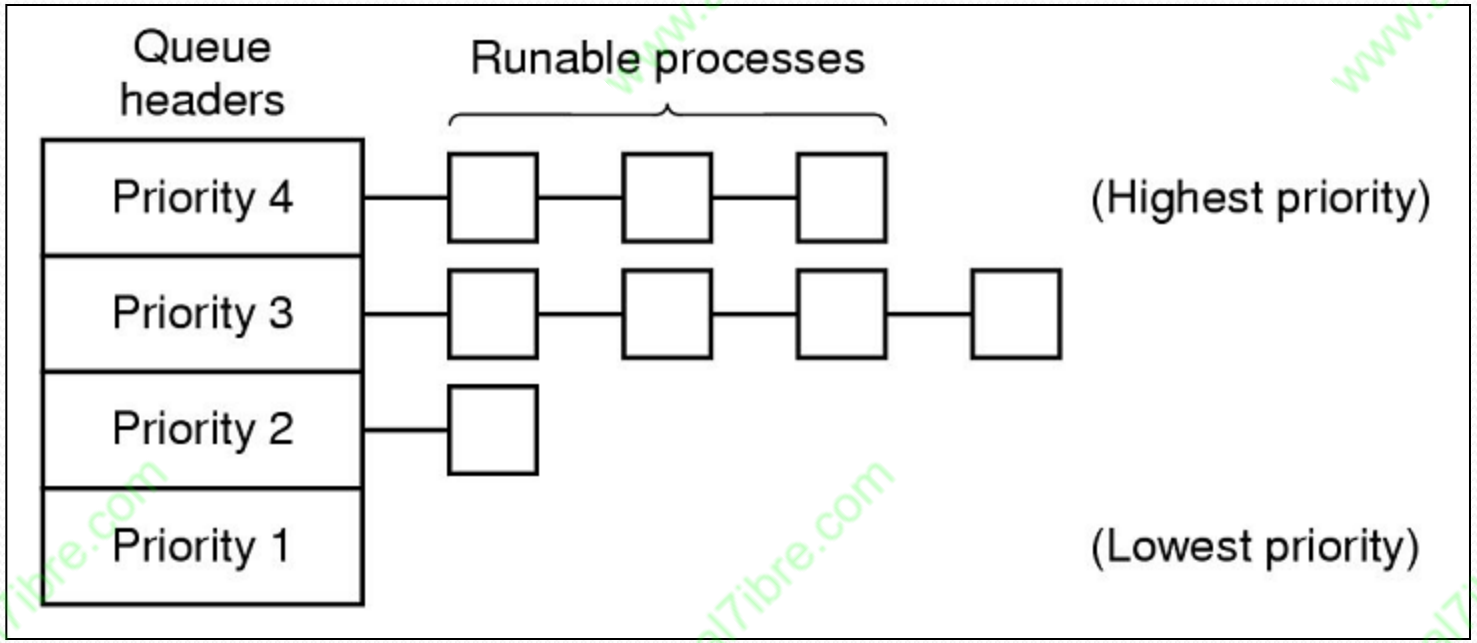
Ordonnancement des systèmes interactifs

- Ordonnancement par priorité: combien de temps on va laisser les processus s'exécuter? Quelques choix:
  - À jamais jusqu'à ce que un processus de plus haute priorité arrive
  - Pour un quantum, potentiellement plus de quantums donnés aux processus à plus hautes priorités
  - Diminue la priorité d'un processus chaque tic d'horloge jusqu'à ce que sa priorité soit plus basse qu'un autre processus (ie: on avance vers les processus de plus basse priorité)

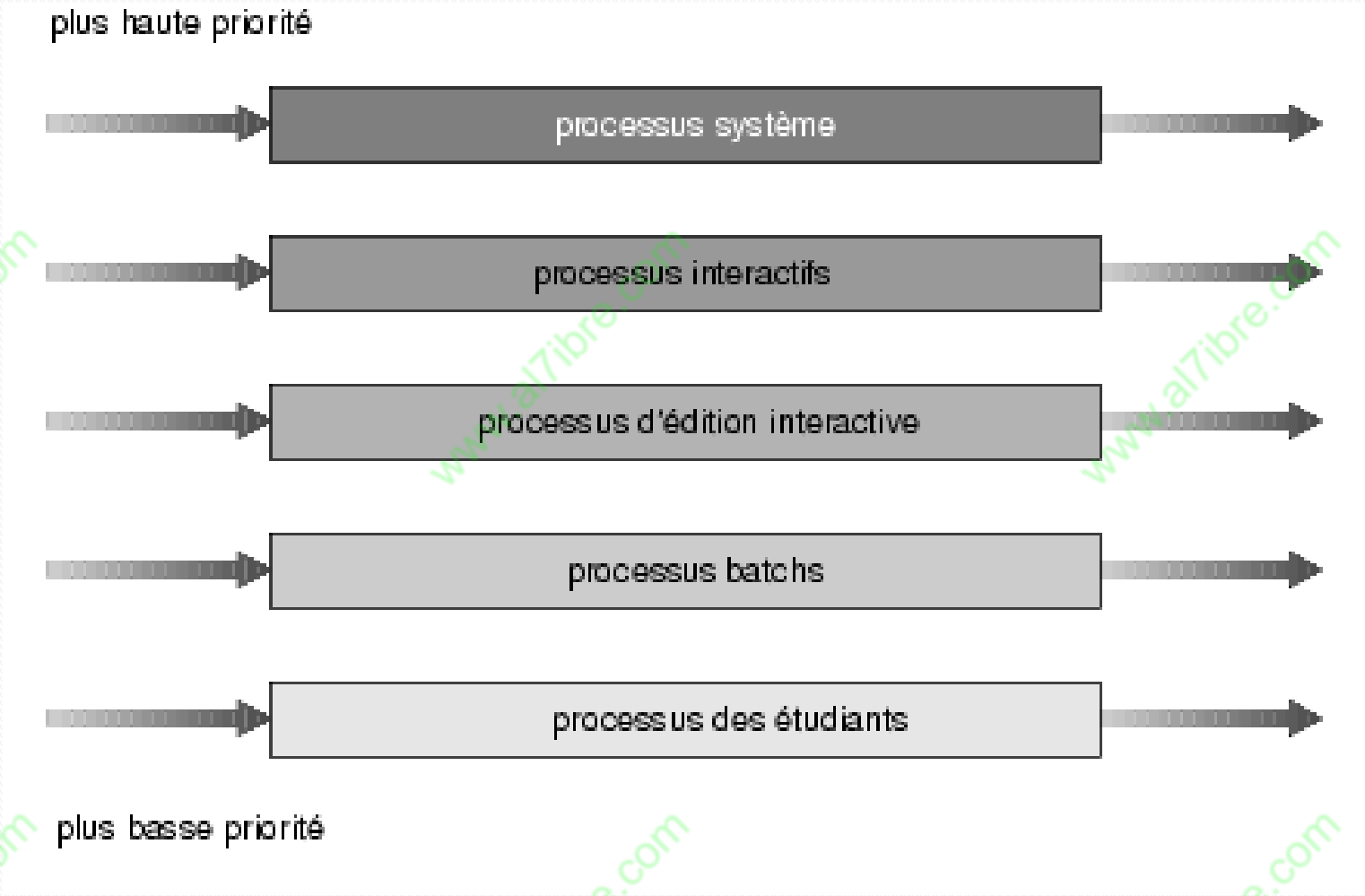
www.al7ibre.com

Ordonnancement des systèmes interactifs

- Ordonnancement par priorité: hybride avec Round-Robin
  - Avoir Round-Robin dans chaque groupe de priorité et exécute seulement les processus dans le groupe le plus haut
    - Problème: Les processus avec les priorités les plus basses peuvent souffrire de famine…



Ordonnancement avec files multiples



www.al7ibre.com

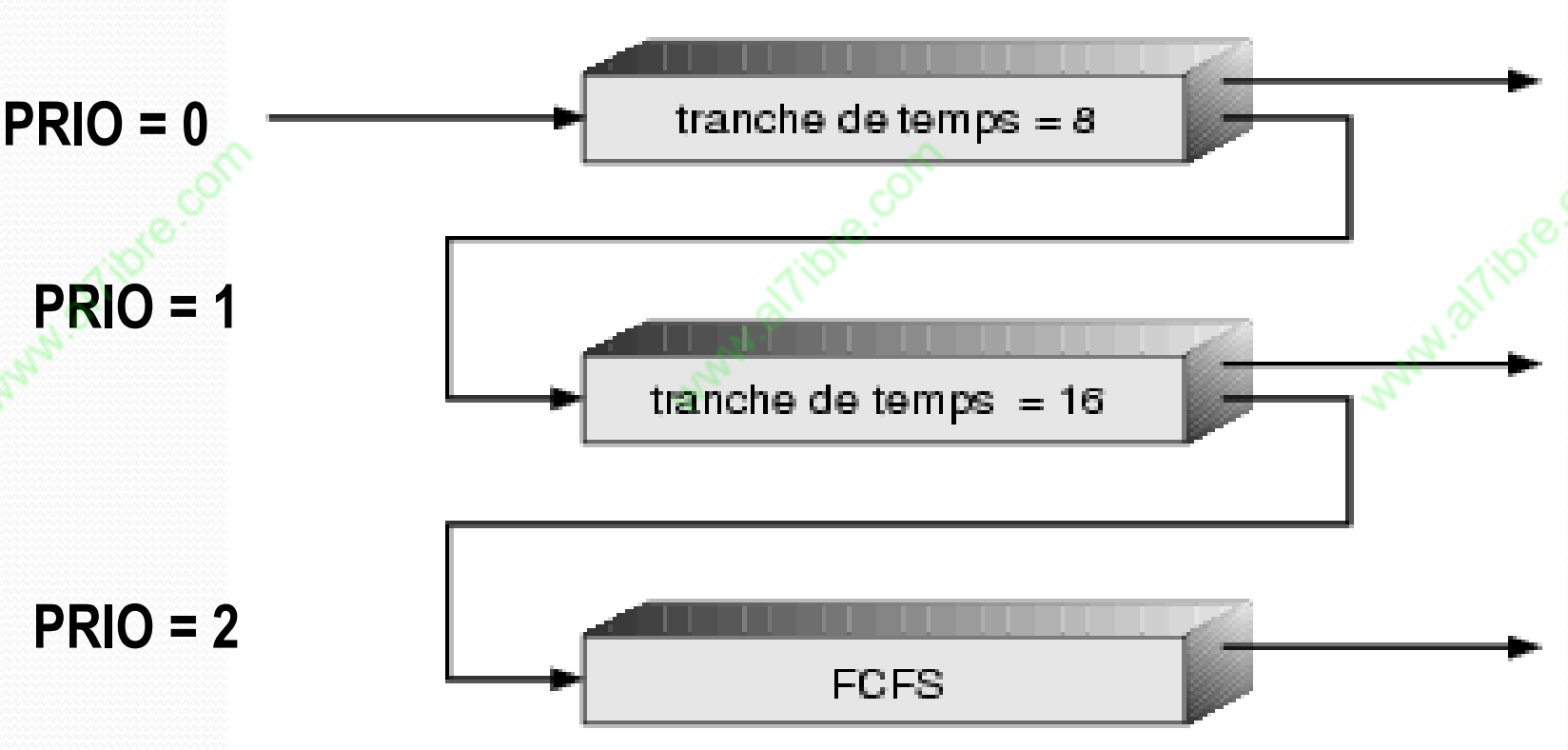
Files multiples et à retour

- Un processus peut passer d'une file à l'autre, par exemple quand il a passé trop de temps dans une file

- À déterminer:
  - nombre de files
  - algorithmes d'ordonnancement pour chaque file
  - algorithmes pour décider quand un processus doit passer d'une file à l'autre
  - algorithme pour déterminer, pour un processus qui devient prêt, sur quelle file il doit être mis

www.al7ibre.com

Files multiples et à retour



Exemple de files multiples à retour

- Trois files:
  - Q0: tourniquet, quantum 8 msecs
  - Q1: tourniquet, quantum 16 msecs
  - Q2: FCFS
- Ordonnancement:
  - Un nouveau processus entre dans Q0, il reçoit 8 msecs d 'UCT
  - S 'il ne finit pas dans les 8 msecs, il est mis dans Q1, il reçoit 16 msecs additionnels
  - S 'il ne finit pas encore, il est interrompu et mis dans Q2
  - Si plus tard il commence à demander des quantums plus petits, il pourrait retourner à Q0 ou Q1

www.al7ibre.com

En pratique…

- Les méthodes que nous avons vu sont toutes utilisées en pratique (sauf le plus court en premier *pur* qui est impossible)
- Les SE sophistiqués fournissent au gérant du système une librairie de méthodes, qu'il peut choisir et combiner au besoin après avoir observé le comportement du système
- Pour chaque méthode, plusieurs paramètres sont disponibles: par exemple la durée du quantum.

www.al7ibre.com

Principes des systèmes d'exploitation

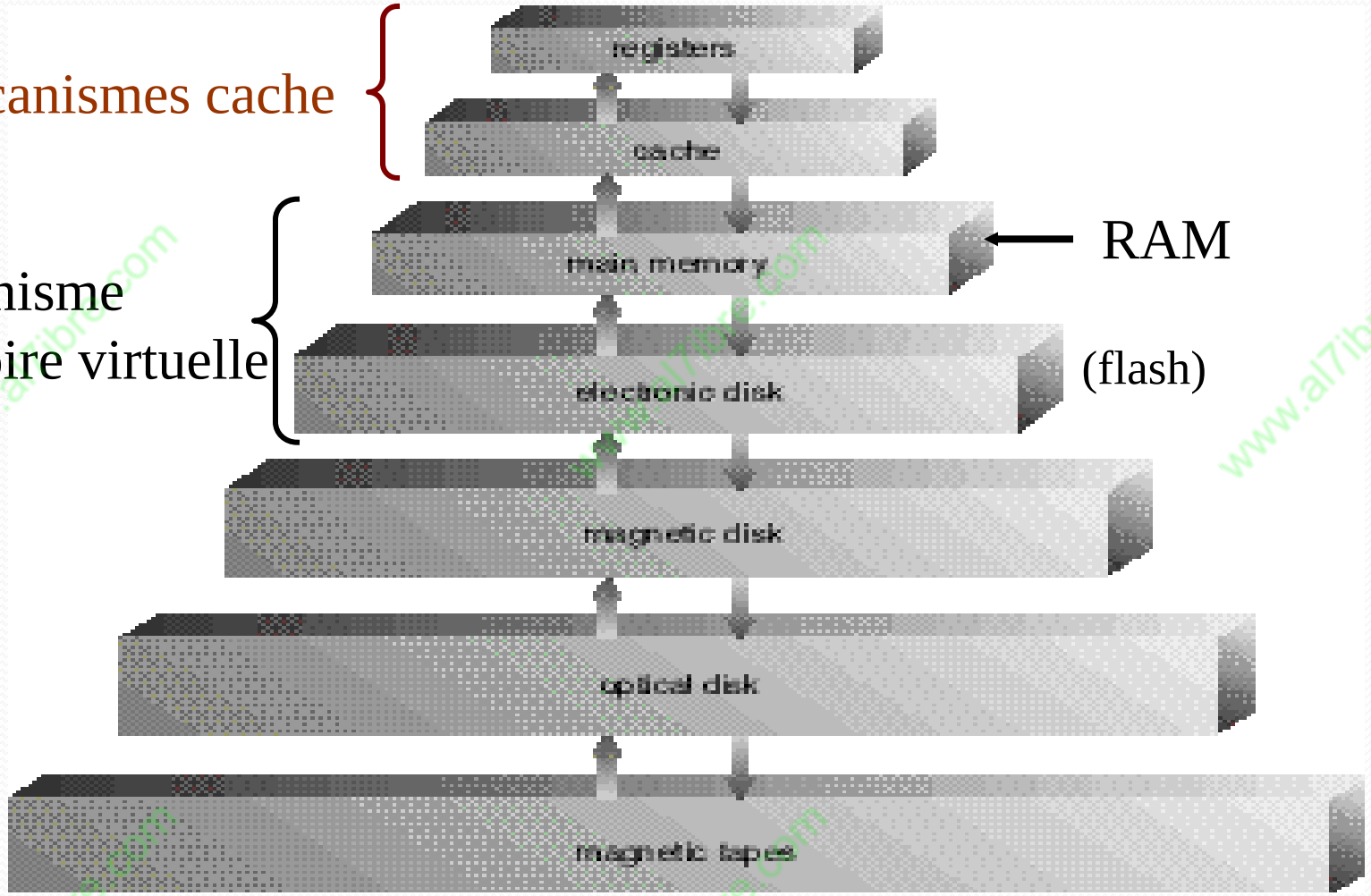
Gestion de la mémoire de base

www.al7ibre.com

La mémoire: concept de hiérarchie de mémoire

Mécanismes cache

Mécanisme mémoire virtuelle



www.al7ibre.com

Gestion de mémoire: objectifs

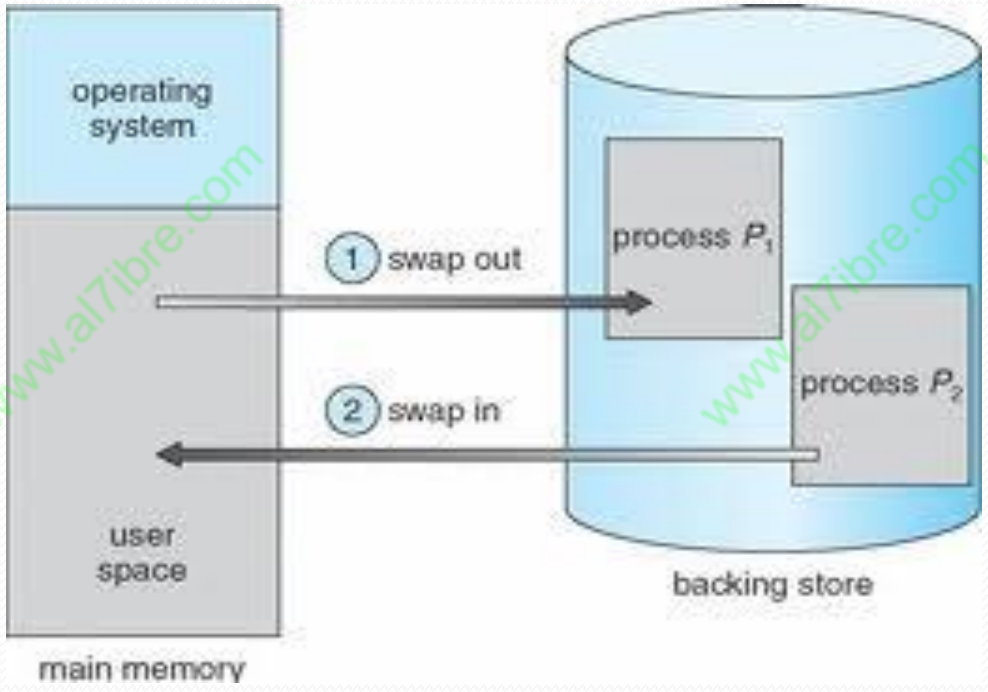
- Optimisation de l'utilisation de la mémoire principale = RAM
- Les plus grand nombre possible de processus actifs doit y être gardé, de façon à optimiser le fonctionnement du système en multiprogrammation
  - garder le système le plus occupé possible, surtout l'UCT
  - s'adapter aux besoins de mémoire de l'usager
    - allocation dynamique au besoin

www.al7ibre.com

Gestion de la mémoire

Qu'est-ce que les SE font quand ils exécutent des programmes qui demandent de la mémoire qui excède leurs capacités?

- Une partie du programme ou la totalité est permuté au disque (swapped)
- On a besoin de quelque chose pour gérer le mouvement des programmes entre le(s) disque(s) et la mémoire



La partie du système d'exploitation qui fait cela s'appel le **gestionnaire de mémoire.**

Gestion de la mémoire

- Trouver de la mémoire libre pour un module de chargement:
  - contiguë ou
  - non contiguë
- En premier on considère deux systèmes simples:
  - Monoprogrammation
  - Partitions fixe
- En grande part, utile pour les systèmes par lots (batch)
- Quand les processus sont chargés en mémoire ils s'exécutent jusqu'à la terminaison

www.al7ibre.com

Multiprogrammation

Comment organiser la mémoire de manière à faire cohabiter efficacement plusieurs processus tout en assurant la protection des processus ?

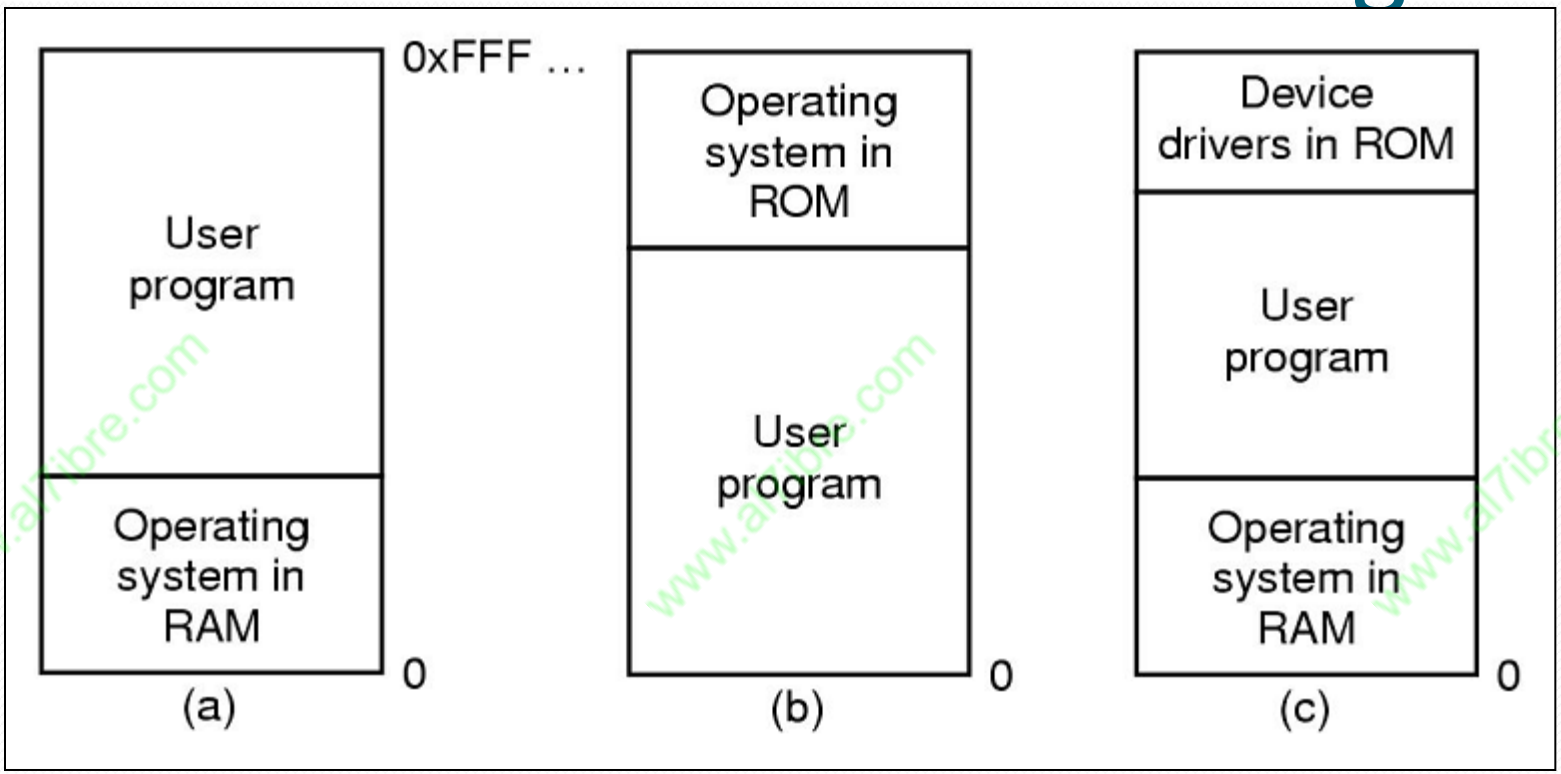
- Multiprogrammation sans va-et-vient. Ex : moniteur MS DOS
  - Un processus chargé en mémoire y séjournera jusqu'à ce qu'il se termine
- Multiprogrammation avec va-et-vient : réutilisation de l'espace mémoire
  - Un processus peut être déplacé temporairement sur le disque (mémoire de réserve : swap area ou backing store) pour permettre le chargement et donc l'exécution d'autres processus. Le processus déplacé sur le disque sera ultérieurement rechargé en mémoire pour lui permettre de poursuivre son exécution.

Gestion de la mémoire contiguë

- Monoprogrammation sans permutation ou pagination
  - L'arrangement le plus simple possible
  - Seulement un programme va s'exécuter à la fois
  - Le SE copie le programme du disque en mémoire et l'exécute. Quand c'est fini, le SE est prêt à accepter une nouvelle commande de l'utilisateur
  - Le nouveau programme écrase le dernier programme en mémoire
  - Trois configurations

www.al7ibre.com

Gestion de la mémoire contiguë

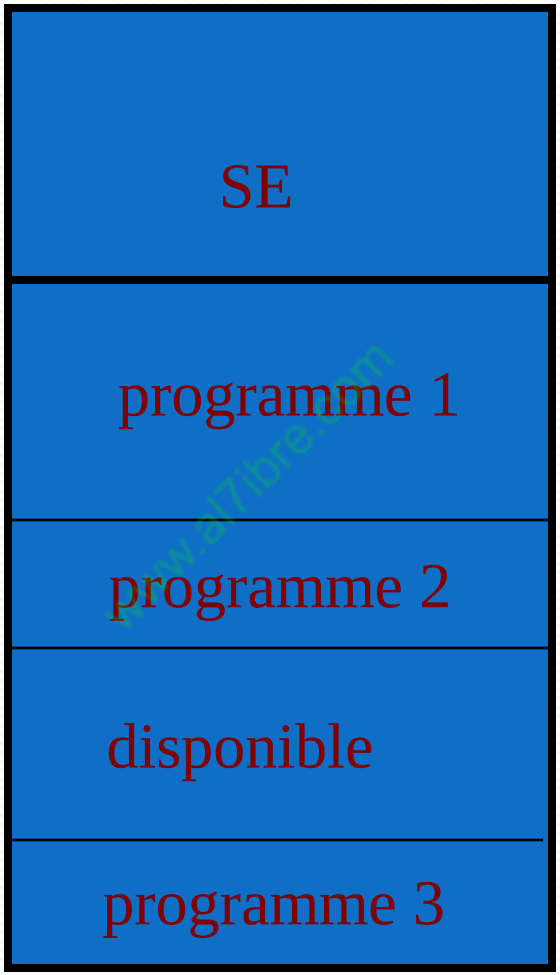


a) Rarement utilisé de nos jours

b) Utilisé dans les baladeurs MP3, les ordinateurs de poche (Palm)

c) Était le modèle initiale pour les PCs, ex: DOS

Affectation contiguë de mémoire



Nous avons ici 4 partitions pour des programmes - chacun est lu dans une seule zone de mémoire

www.al7ibre.com

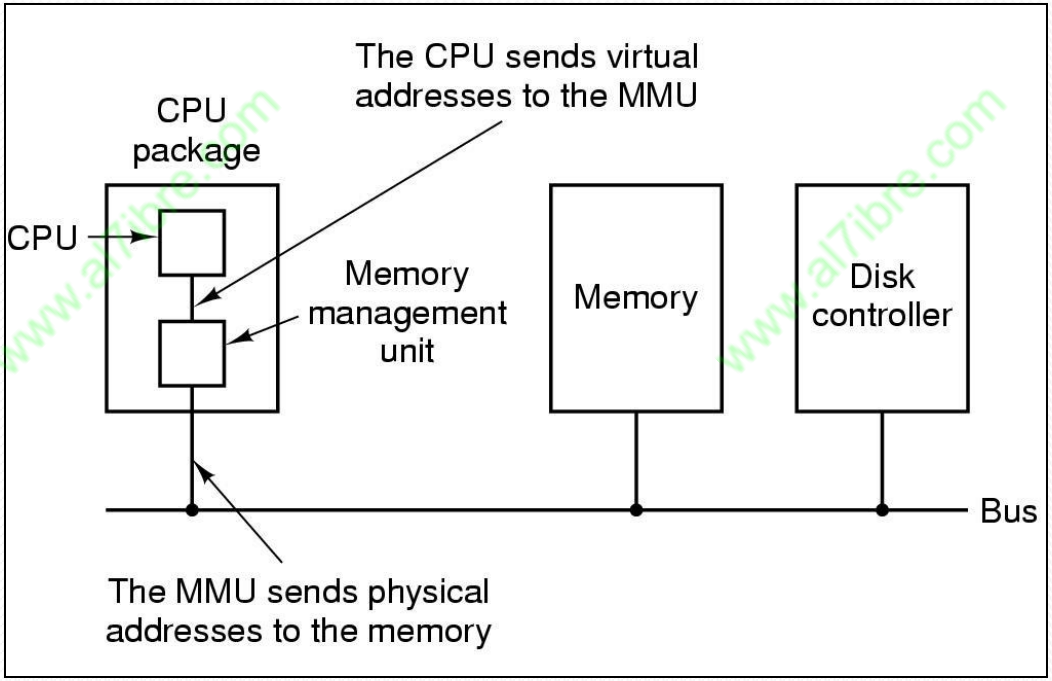
Adresses mémoires

- En général on compile tous les programmes avec des adresses *logiques* qui sont **relatives** à l'adresse de base: zéro. Les adresses *physiques* sont calculées durant l'exécution du programme
  - La conversion des adresses est fait par une pièce de matériel que l'on appel Unité de gestion de la mémoire (Memory Management Unit) (MMU)
  - Le MMU prend les adresses logiques du processus et les transforment en adresses physique dans le RAM

www.al7ibre.com

Arrière-plan

- Les adresses qui sont données au MMU sont connues comme des **adresses virtuelles** ou **adresses logiques**
- Le MMU produit une **adresse physique**
  - L'adresse physique est dans le RAM



Définition des adresses

- Une adresse *logique* est une adresse d'un emplacement dans le programme
  - par rapport au programme lui-même seulement
  - indépendante de la position du programme en mémoire physique
- Une adresse *physiques* est une adresse réelle de la RAM

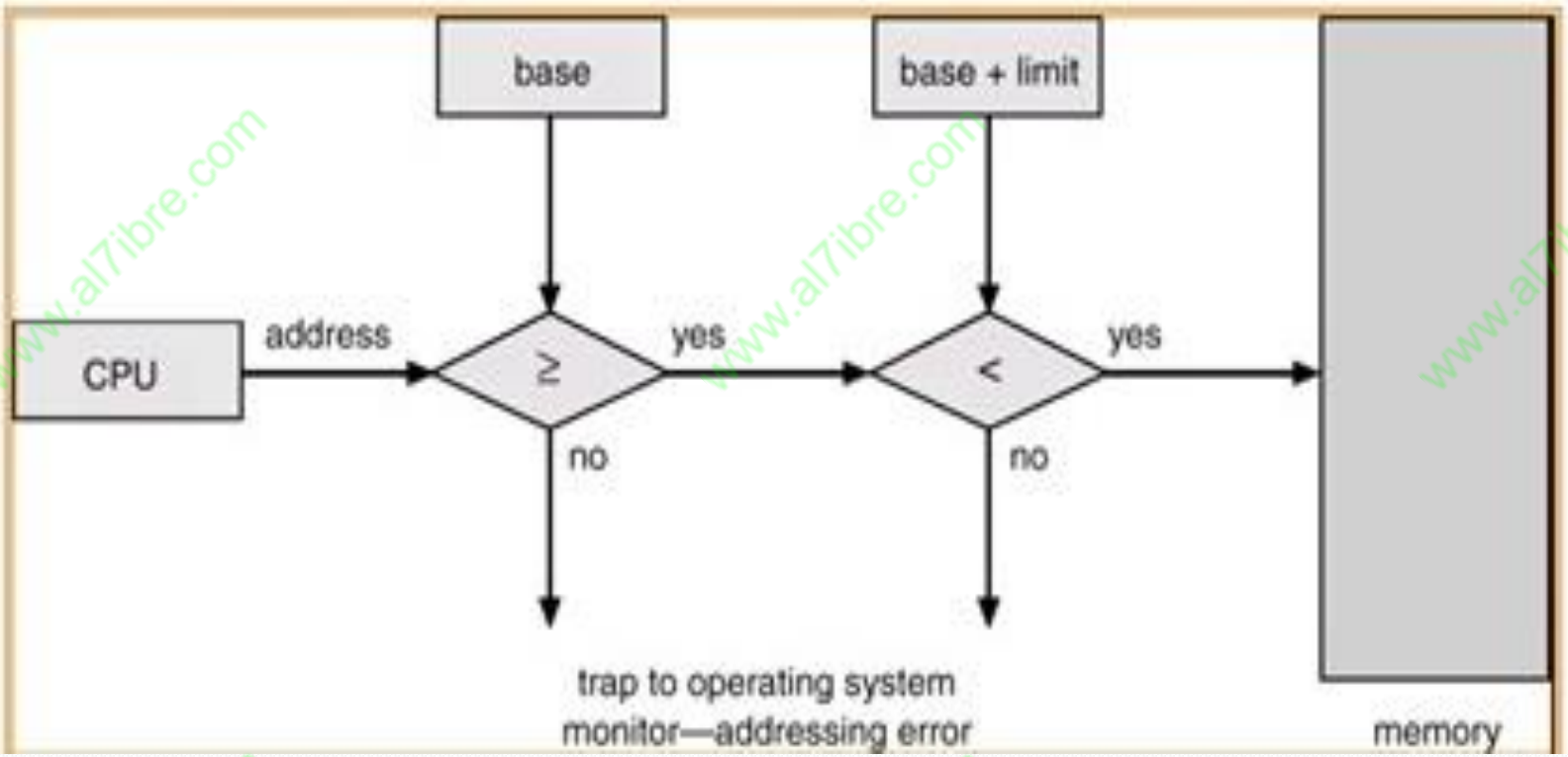
  Les adresses *physiques* sont calculées durant l'exécution du programme

www.al7ibre.com

Traduction adresses logiques →physiques

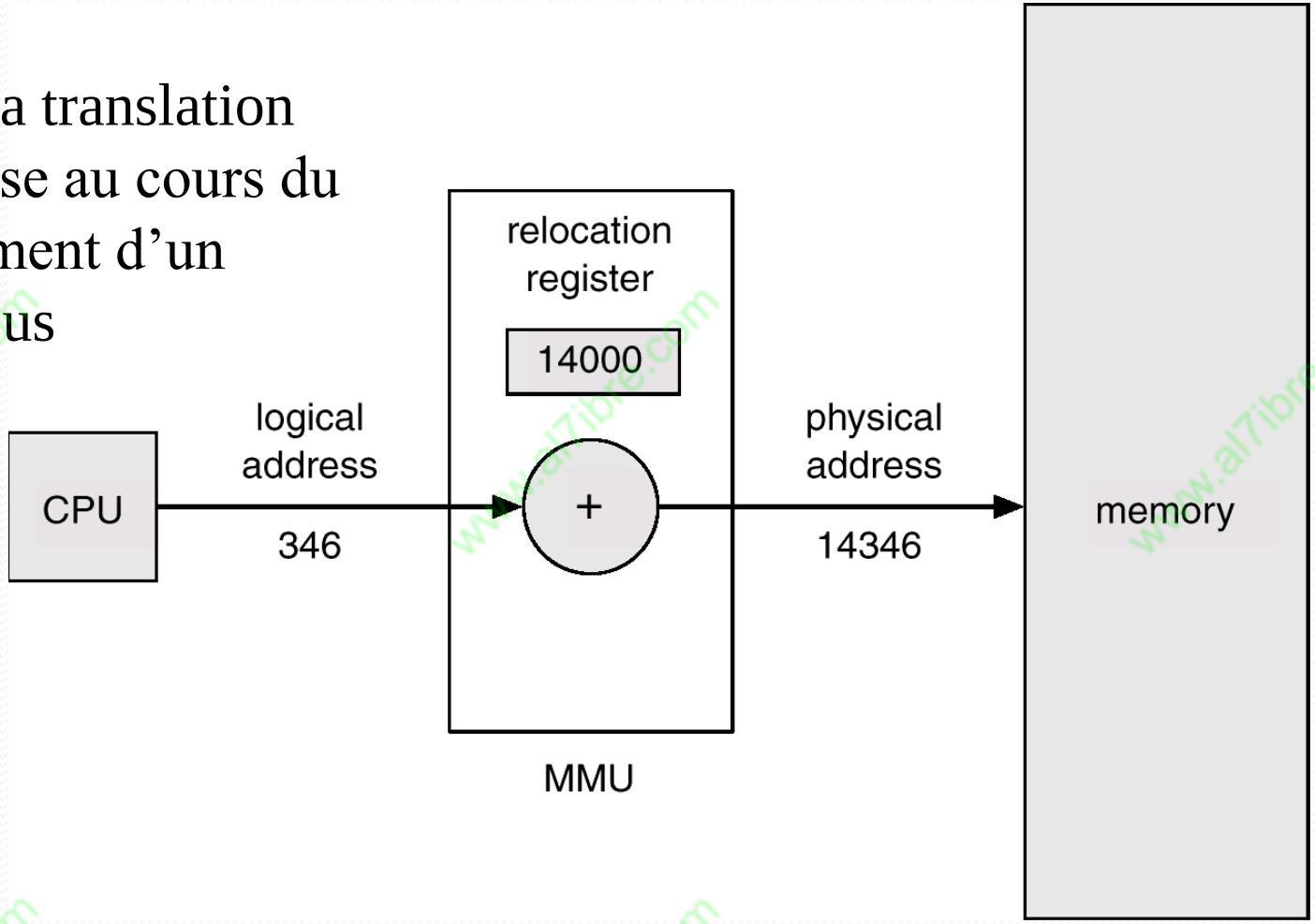
- Les registres matériels permettent de délimiter le domaine des processus;
- Le matériel compare les adresses émises par le processus aux registres de base et de barrière ;
- Registre de base contient l'adresse de base du processus en mémoire qui permet de décrire la zone d'adressage d'un programme, la plus petite adresse légale
- Registre barrière contient une adresse limite qui peut être comparée à toute adresse d'instruction ou de données manipulées par un programme. (taille de la plage accessible)

Traduction adresses logiques →physiques



Traduction adresses logiques →physiques

Eviter la translation d'adresse au cours du chargement d'un processus



Liaison (Binding) d'adresses logiques et physiques

- La liaison des adresses logiques aux adresses physiques peut être effectuée à des moments différents:
  - Compilation: quand l'adresse physique est connue au moment de la compilation (rare)
    - p.ex. parties du SE
  - Chargement: quand l'adresse physique où le progr est chargé est connue, les adresses logiques peuvent être traduites (rare aujourd'hui)
  - Exécution: normalement, les adresses physiques ne sont connues qu'au moment de l 'exécution
    - p.ex. allocation dynamique

www.al7ibre.com

Chargement et liaison dynamique

- Un processus s'exécutant peut avoir besoin de différents modules du programme en différents moments
- Le chargement statique peut donc être inefficace
- Il est mieux de charger les modules sur demande = dynamique
  - dll, dynamically linked libraries

www.al7ibre.com

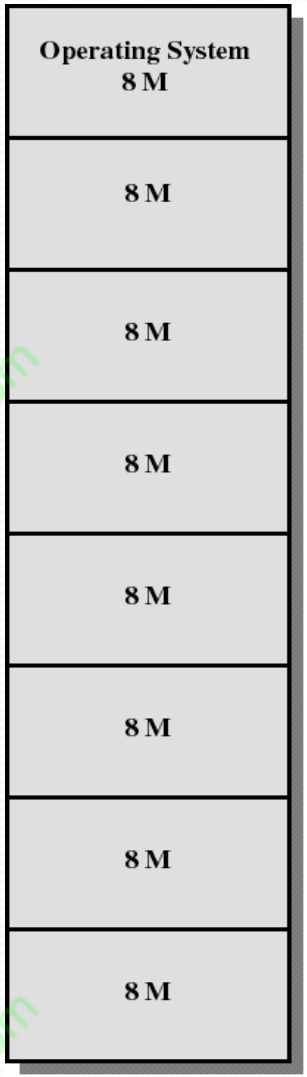
Gestion de la mémoire contiguë

- Multiprogrammation avec partitions fixes
  - Pour exploiter les bénéfices de la multiprogrammation nous avons besoin d'avoir plus d'un programme en mémoire à la fois
  - Solution simple (pour les systèmes par lots): diviser la mémoire en *n* partitions et de mettre le prochain programme qui arrive dans la plus petite partition qui peut la contenir

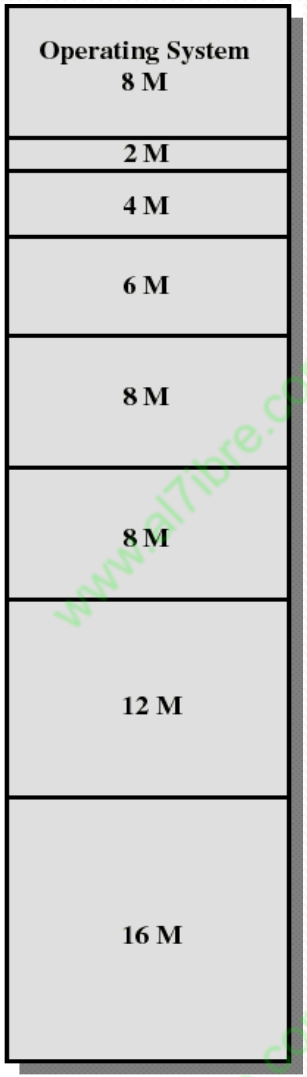
www.al7ibre.com

Partitions fixes

- Première organisation de l'allocation contiguë
- Mémoire principale subdivisée en régions distinctes: partitions
- Les partitions sont soit de même taille ou de tailles inégales
- N'importe quel programme peut être affecté à une partition qui soit suffisamment grande



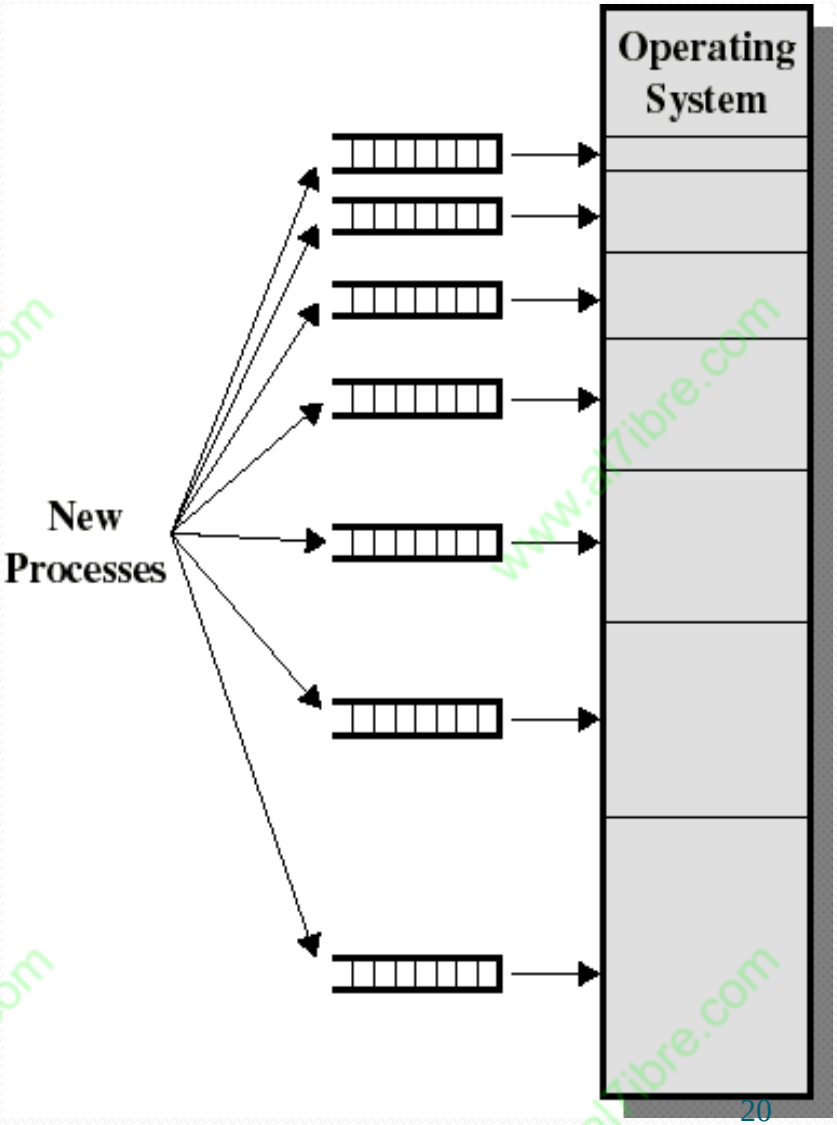
Equal-size partitions



Unequal-size partitions

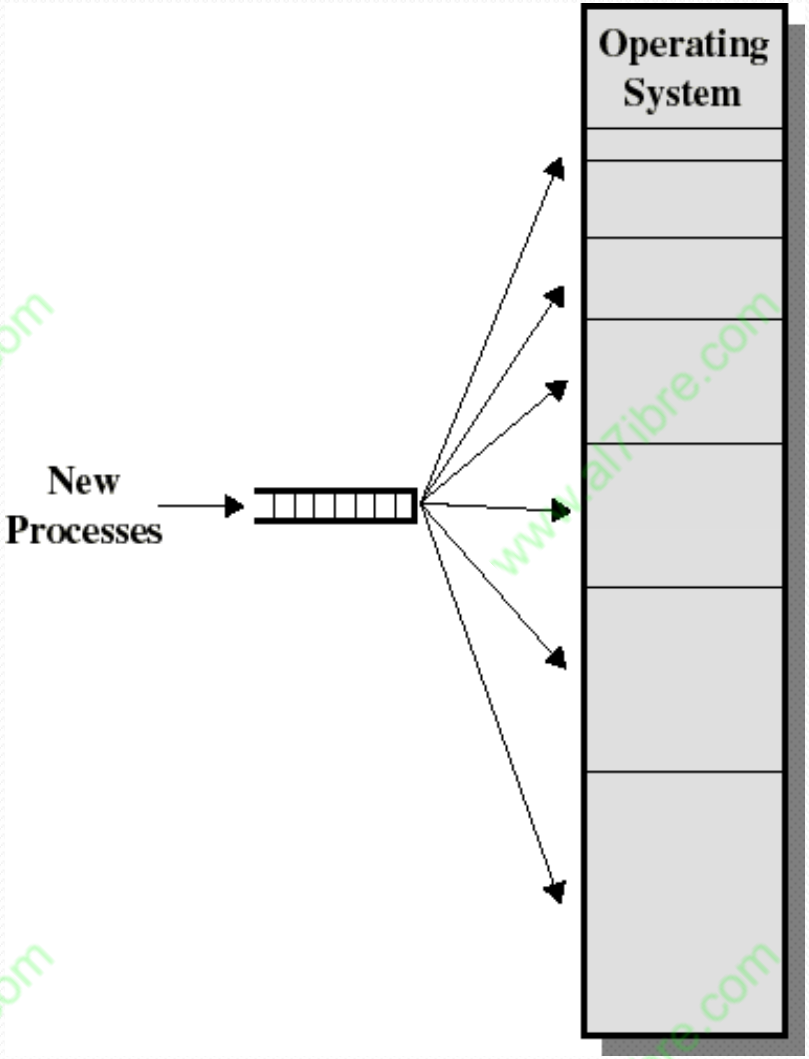
Algorithme de placement pour partitions fixes

- **Partitions de tailles inégales: utilisation de plusieurs queues**
  - assigner chaque processus à la partition de la plus petite taille pouvant le contenir
  - Une file par taille de partition
  - tente de minimiser la fragmentation interne
  - Problème: certaines files seront vides s'il n'y a pas de processus de cette taille :fragmentation externe



Algorithme de placement pour partitions fixes

- **Partitions de tailles inégales: utilisation d'une seule file**
  - On choisit la plus petite partition libre pouvant contenir le prochain processus
  - le niveau de multiprogrammation augmente au profit de la fragmentation interne



www.al7ibre.com

Partitions fixes

- Simple, mais...
- Inefficacité de l'utilisation de la mémoire: tout programme, si petit soit-il, doit occuper une partition entière. Il y a fragmentation interne.
- Les partitions à tailles inégales atténue ces problèmes mais ils y demeurent...

www.al7ibre.com

Partitions dynamiques ou variables

- Partitions en nombre et tailles variables
- Chaque processus se voit alloué exactement la taille de mémoire requise
- Probablement des trous inutilisables se formeront dans la mémoire: c'est la fragmentation externe

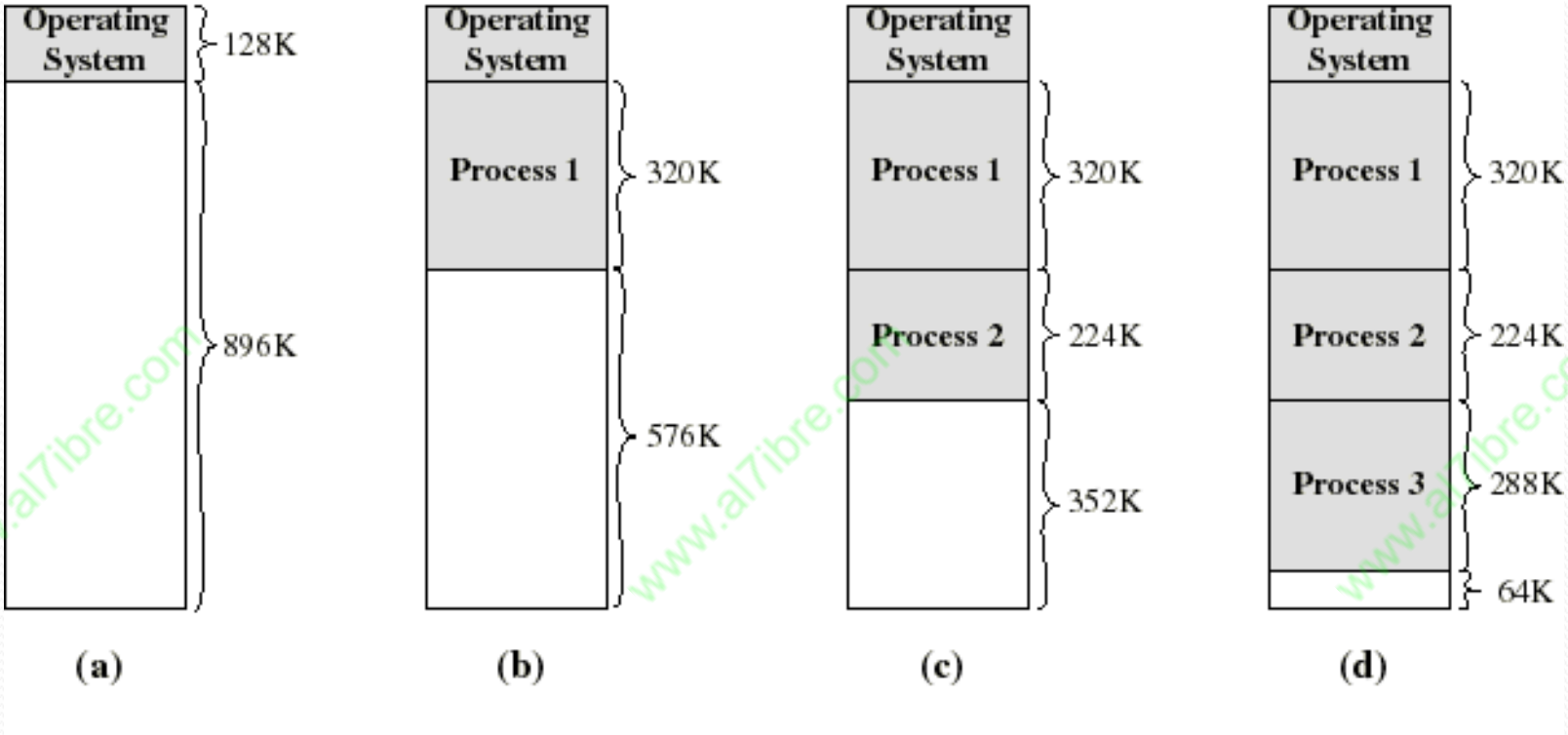
www.al7ibre.com

Permutation

- Les systèmes antérieurs étaient plus simples parce que quand les programmes étaient chargés en mémoire, ils étaient laissés là jusqu'à leur terminaison
- Quand nous n'avons pas assez de mémoire principale pour garder touts les processus actifs en mémoire, nous devons les permuter entre la mémoire principale et le disque…
- La permutation d'un processus consiste à amener un processus du disque à la mémoire dans son entièreté. Le processus est exécuté pour un temps et remis sur le disque

www.al7ibre.com

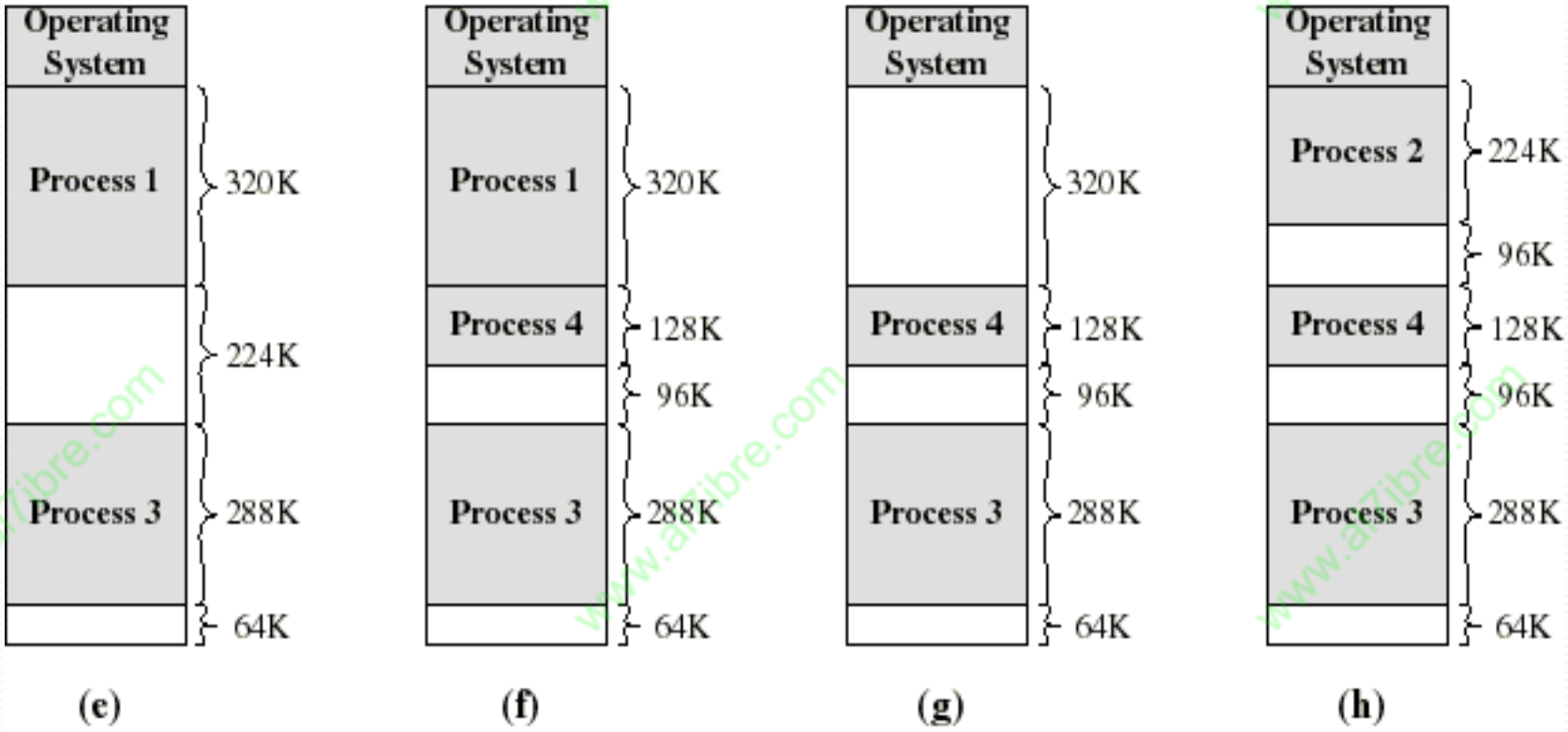
Partitions dynamiques: Permutation



- **(d) Il y a un trou de 64K après avoir chargé 3 processus: pas assez d'espace pour autre processus**
- **Si tous les processus se bloquent (p.ex. attente d'un événement), P2 peut être permuté et P4=128K peut être chargé.**

Swapped out

Partitions dynamiques: Permutation



- **(e-f) P2 est suspendu, P4 est chargé. Un trou de 224-128=96K est créé (*fragmentation externe*)**
- **(g-h) P1 se termine ou il est suspendu, P2 est chargé à sa place: produisant un autre trou de 320-224=96K...**
- **Nous avons 3 trous petits et probablement inutiles. 96+96+64=256K de fragmentation externe**
- **COMPRESSION pour en faire un seul trou de 256K**

Permutation ou va-et-vient

- La différence entre ce système et les partitions fixes est que le nombre, la location et la grandeur des partitions varient dynamiquement
  - Avantages:
    - Une solution bien plus flexible
    - Une meilleur utilisation de la mémoire
  - Désavantages:
    - Plus compliqué à implémenter
    - Il peut y avoir des “trous” laissés dans la mémoire, qui peuvent être compactés pour corriger le problème

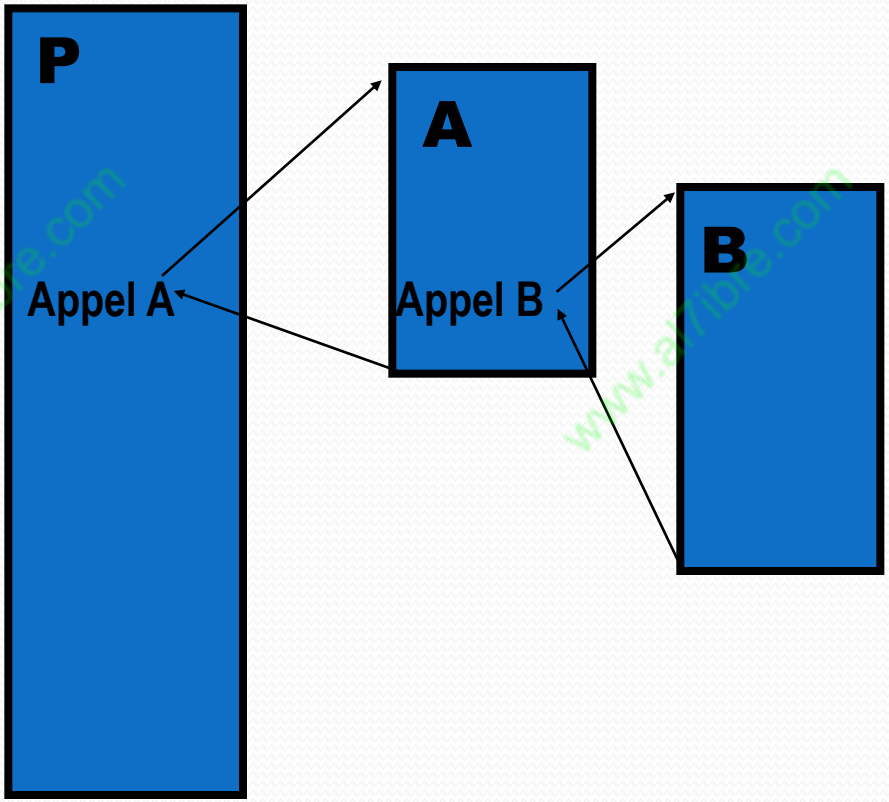
www.al7ibre.com

Permutation – Combien de mémoire

- Combien de mémoire devrait-on assigner à un processus quand il est permuté dans la mémoire?
  - Si une grandeur de données fixe peut être déterminé alors cette grandeur exacte est allouée
  - Cependant, si un processus a une pile et/ou un tas (heap), alors nous devons lui allouer de l'espace pour grandir pour empêcher d'avoir à déplacer le processus continuellement dans la mémoire

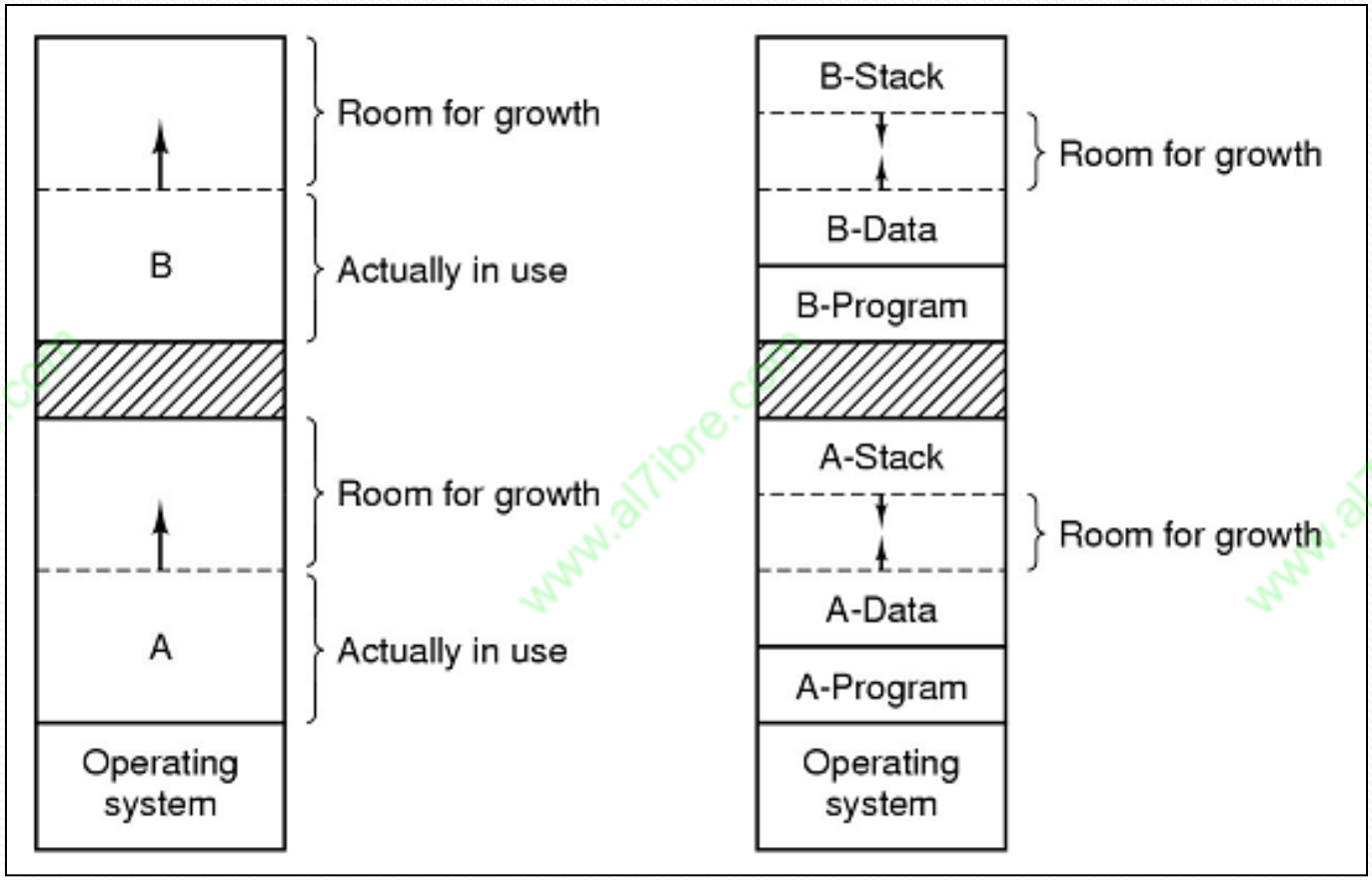
www.al7ibre.com

La Pile d'un processus



www.al7ibre.com

Permutation – Combien de mémoire



Segment de données

Segment de données et pile

Permutation - Gestion

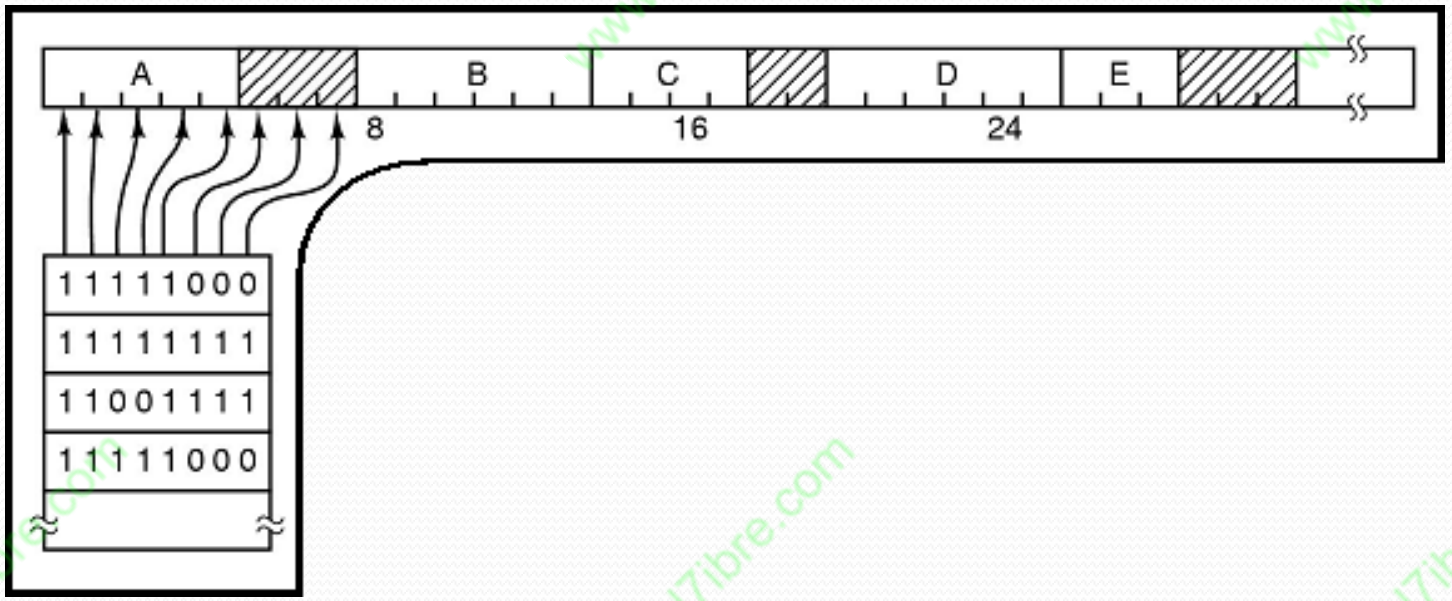
- Avant d'implanter une technique de gestion de la mémoire centrale par va-et-vient, il est nécessaire de connaître son état : les zones libres et occupées; de disposer d'une stratégie d'allocation et enfin de procédures de libération. Les techniques que nous allons décrire servent de base au va-et-vient; on les met aussi en œuvre dans le cas de la multiprogrammation simple où plusieurs processus sont chargés en mémoire et conservés jusqu'à la fin de leur exécution.

- Deux méthodes:
  - Tableaux de bits et
  - listes chaînées

www.al7ibre.com

Permutation - Gestion

- Gestion de la mémoire avec tableaux de bits:
  - Divise la mémoire en unités d'allocation tel que 4 octets ou plusieurs kilooctets
  - Utilise un tableau de bits avec des 1 pour désigner les unités alloués et des 0 pour désigner les unités libres
  - De quelle grandeur sont nos unités d'allocation?



Permutation - Gestion

- Gestion de la mémoire avec tableaux de bits:
  - La plus petite est l'unité d'allocation, le plus grand sera le tableau de bits correspondant
  - Cependant, même avec des unités de 4 octets (32 bits) on perd seulement 1/33 de la mémoire
  - Les unités qui sont larges nous font perdre la fin de la dernière unité, ex: pour un unité de 64Ko, si nous avons un programme qui a 65Ko, alors nous perdons 63 kilooctets!

www.al7ibre.com

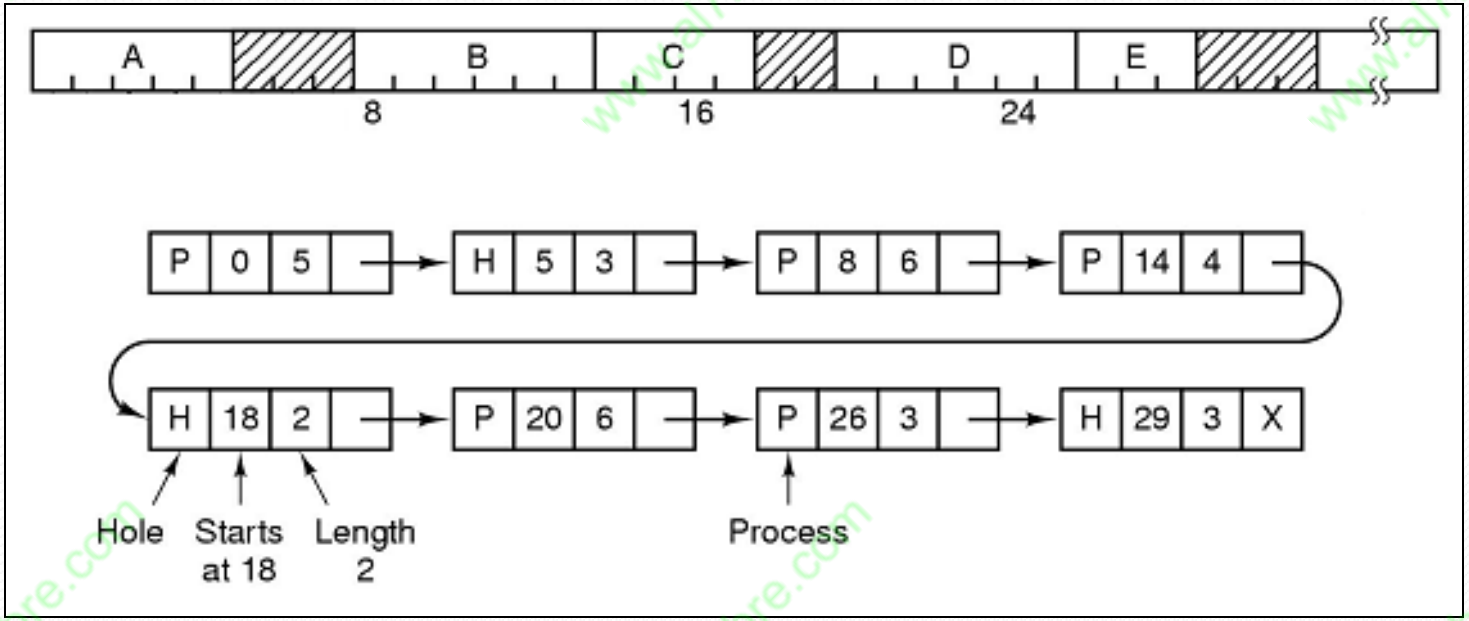
Permutation - Gestion

- Gestion de la mémoire avec tableaux de bits:
  - Avantages:
    - Facile à implémenter
    - Le tableau de bits est de grandeur fixe, peu importe combien de programme sont en mémoire
  - Désavantage
    - Peut prendre du temps pour chercher dans le tableau pour trouver une série de 0 consécutifs pour placer un programme

www.al7ibre.com

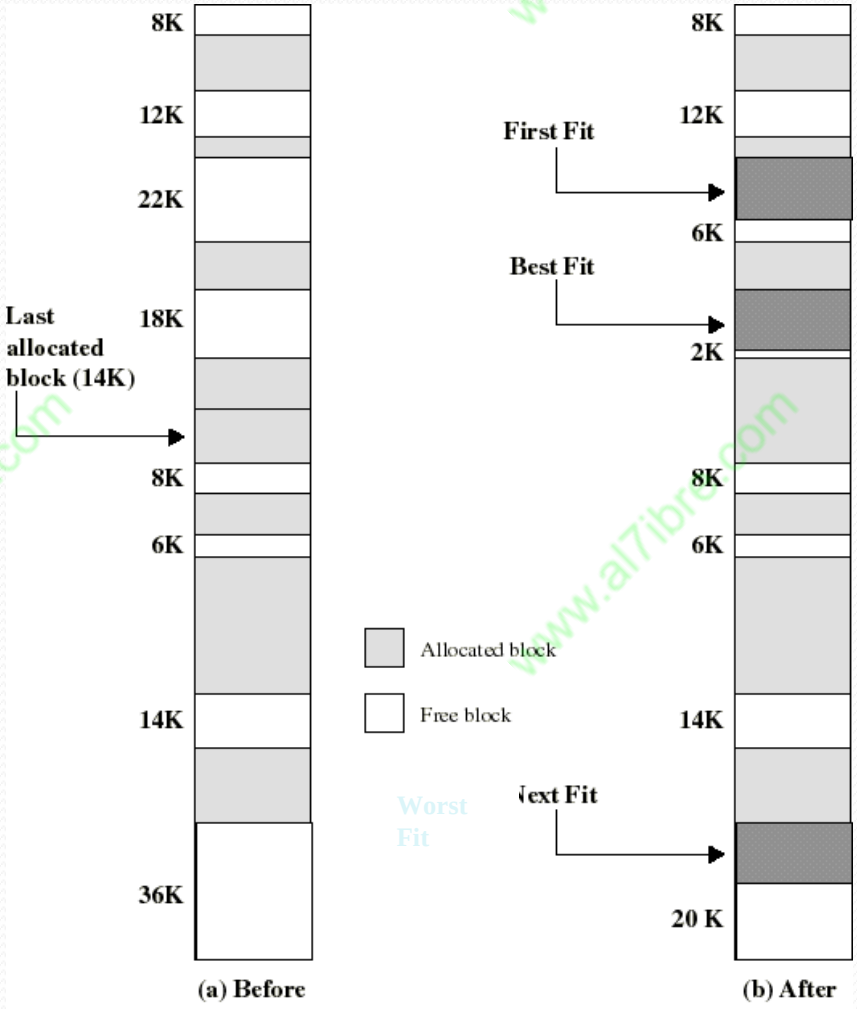
Permutation - Gestion

- Gestion de la mémoire avec des listes chaînées:
  - Utilise une liste chaînée pour repérer les blocs libres et les blocs occupés
  - Avantage: moins de recherche à faire



Algorithmes de Placement

- "Best-fit": choisir l'emplacement dont la taille est la plus proche
- "First-fit": choisir le 1er emplacement à partir du début
- "Worst-fit": choisir l'emplacement dont la taille est la plus loin
- Prochain trou (Next fit) – amélioration First-fit. Programme inséré à partir de là dernière insertion
  - Donne une petite augmentation de performance sur First-fit (simulation Bays)
- Placement rapide (Quick fit) – On garde les trou tel que 4KB, 8KB, etc dans une liste séparée pour une localisation facile des trous
  - Bon à l'allocation, mais lent pour la dé-allocation parce que plusieurs listes doivent être réconciliés



Example Memory Configuration Before and After Allocation of 16 Kbyte Block

Algorithmes de placement: commentaires

- **Quel est le meilleur?**
  - critère principal: diminuer la probabilité de situations où un processus ne peut pas être servi, même s'il y a assez de mémoire...
- **La simulation montre qu'il ne vaut pas la peine d'utiliser les algorithmes les plus complexes... donc first fit**
- **"Best-fit": cherche le plus petit bloc possible: l'espace restant est le plus petit possible**
  - la mémoire se remplit de trous trop petits pour contenir un programme
- **"Worst-fit": les allocations se feront souvent à la fin de la mémoire**

www.al7ibre.com

Permutation - Gestion

- Note finale sur les listes: on pourrait garder des listes séparées pour les trous et les processus
  - Accélère la recherche de trous!
  - Permet d'ordonner les trous par grandeur pour un allocation encore plus rapide!!
  - Plus compliqué pour la dé-allocation parce que la mémoire qui devient un trou doit être placée dans le bon espace sur l'autre liste (en ordre)…

www.al7ibre.com

Fragmentation: mémoire non utilisée

- Un problème majeur dans l'affectation contiguë:
  - Il y a assez d'espace pour exécuter un programme, mais il est fragmenté de façon non contiguë
    - externe: l'espace inutilisé est entre partitions
    - interne: l'espace inutilisé est dans les partitions

www.al7ibre.com

Compaction

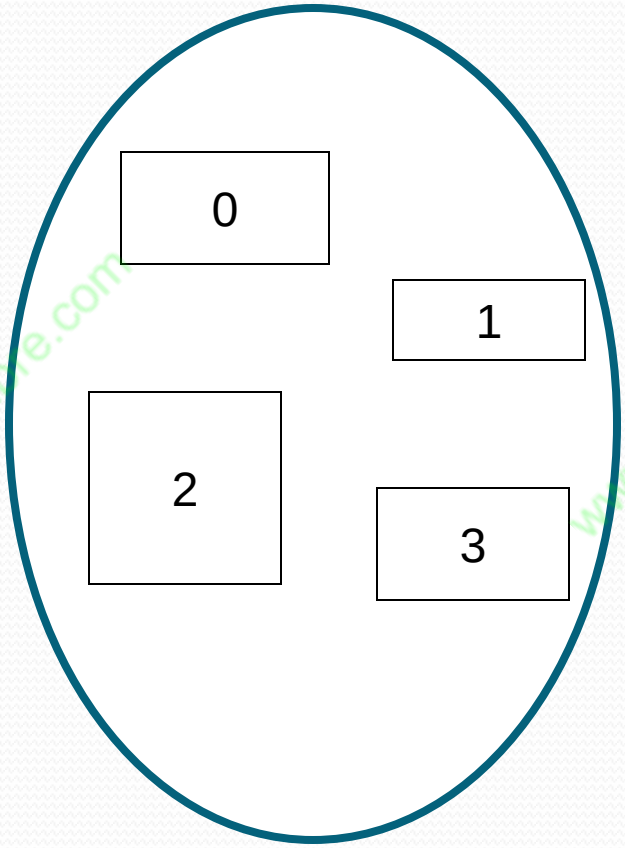
- Une solution pour la fragmentation externe
- Les programmes sont déplacés en mémoire de façon à réduire à 1 seul grand trou plusieurs petits trous disponibles
- Effectuée quand un programme qui demande d'être exécuté ne trouve pas une partition assez grande, mais sa taille est plus petite que la fragmentation externe existante
- Désavantages:
  - temps de transfert programmes
  - besoin de rétablir tous les liens entre adresses de différents programmes

www.al7ibre.com

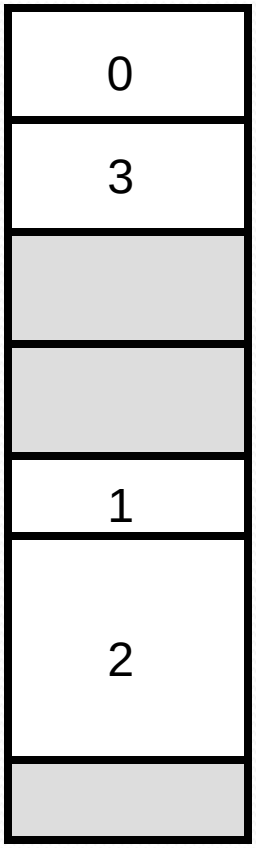
Allocation non contiguë

- Afin de réduire le besoin de compression, le prochain pas est d`utiliser l'allocation non contiguë
  - diviser un programme en morceaux et permettre l`allocation séparée de chaque morceau
  - les morceaux sont beaucoup plus petits que le programme entier et donc permettent une utilisation plus efficace de la mémoire
    - les petits trous peuvent être utilisés plus facilement
- Il y a deux techniques de base pour faire ceci
  - la segmentation utilise des parties de programme qui ont une valeur logique (des modules)
  - la pagination utilise des parties de programme arbitraires (morcellement du programmes en pages de longueur fixe).
  - Combinaison des deux techniques

Affectation non contiguë de mémoire



espace usager



mémoire physique

www.al7ibre.com

Segmentation

- La **segmentation** est une division de mémoire en segments, chacun commençant à une adresse de base dans la mémoire physique. Chaque processus peut avoir plusieurs segments
  - On doit maintenant spécifier un numéro de segment et un offset pour accéder à la mémoire
  - Référé en tant que mémoire à deux dimensions

www.al7ibre.com

Détails

- L'adresse logique consiste d 'une paire:

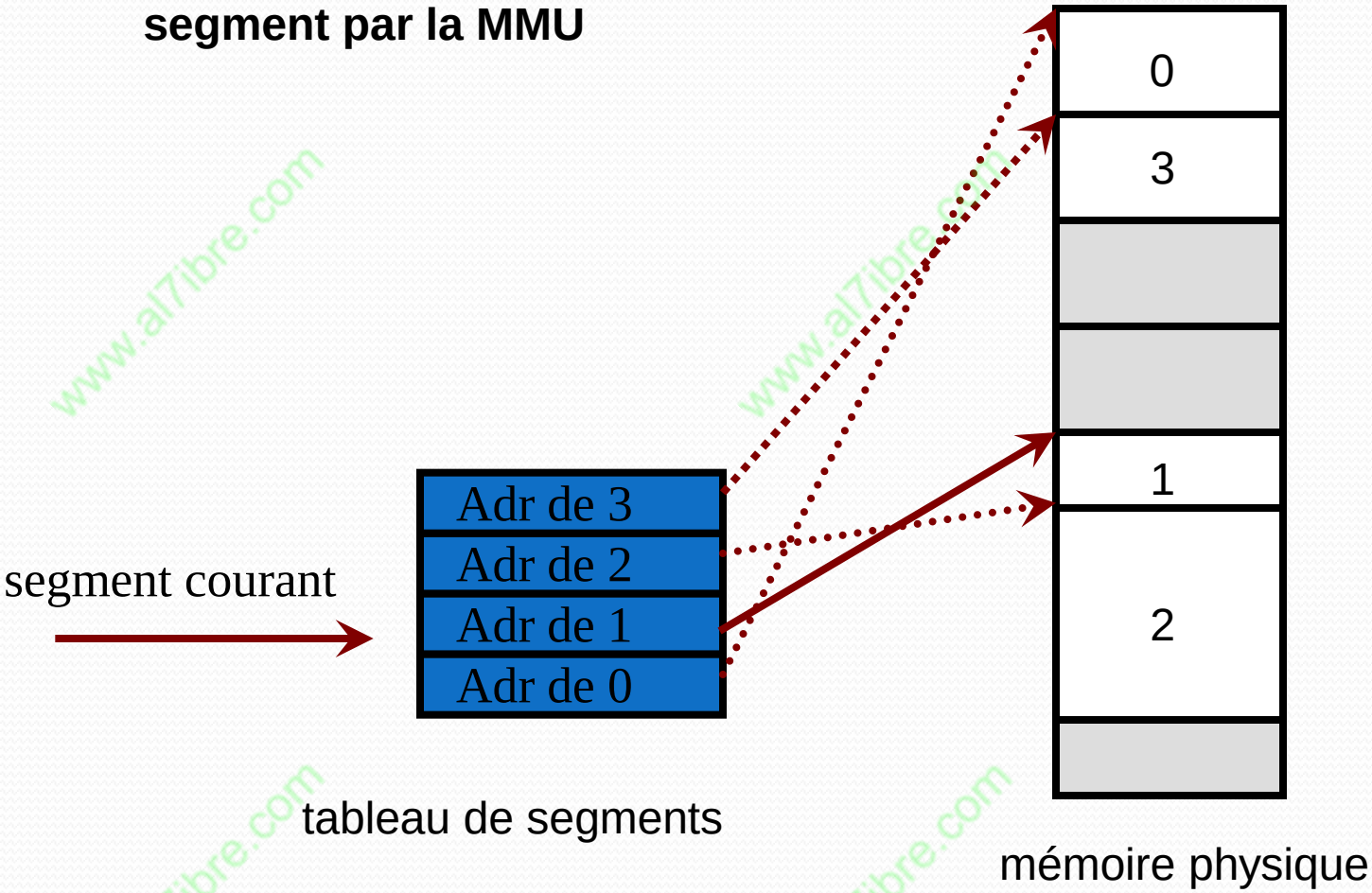
  <No de segm, décalage>

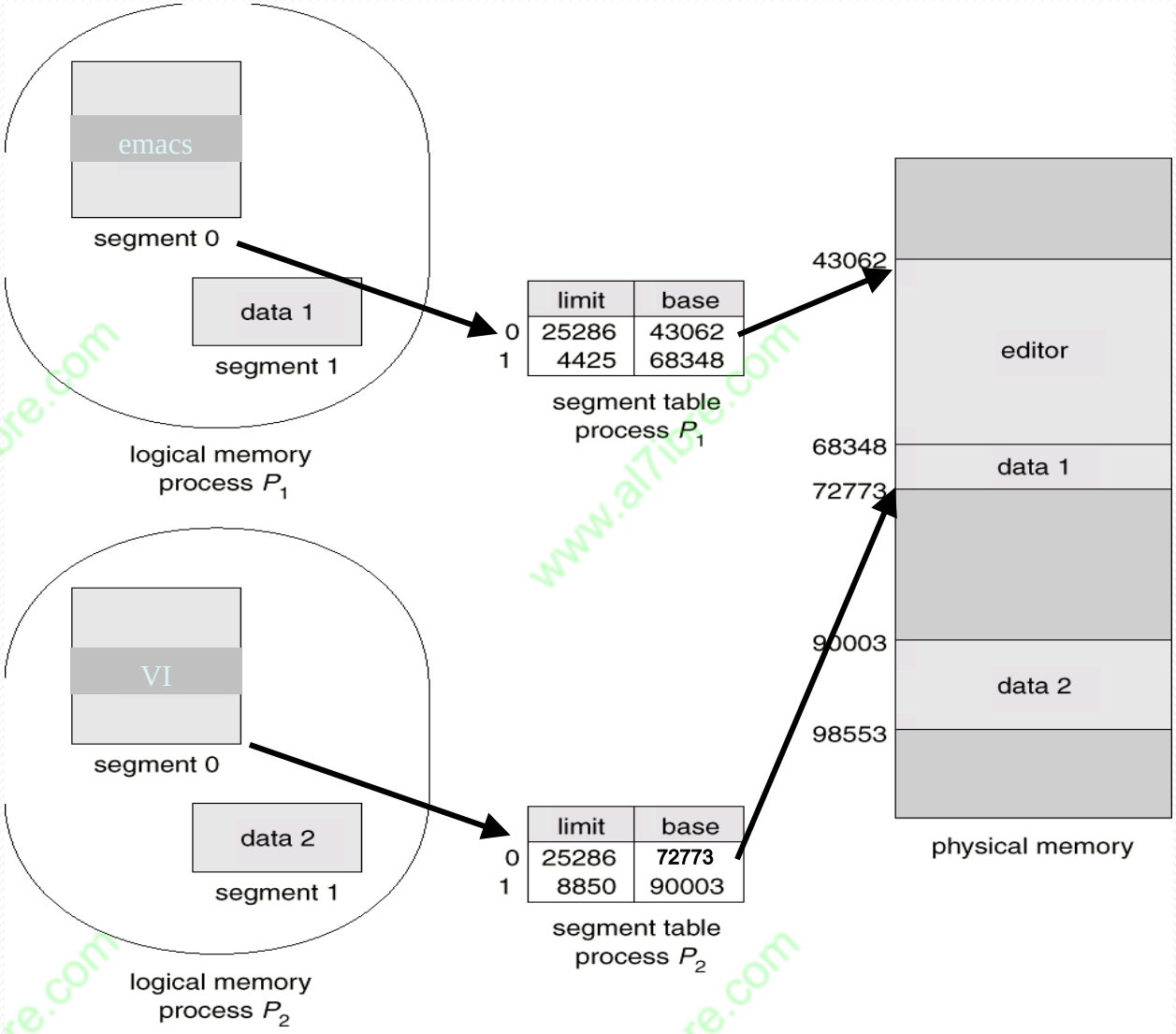
  où décalage est l 'adresse *dans* le segment
- le tableau des segments contient: descripteurs de segments
  - adresse de base
  - longueur du segment
  - Infos de protection
- Dans le PBC du processus il y aura un pointeur à l'adresse en mémoire du tableau des segments
- Il y aura aussi là dedans le nombre de segments dans le processus
- Au moment de la commutation de contexte, ces infos seront chargées dans les registres appropriés d'UCT

www.al7ibre.com

Mécanisme pour la segmentation

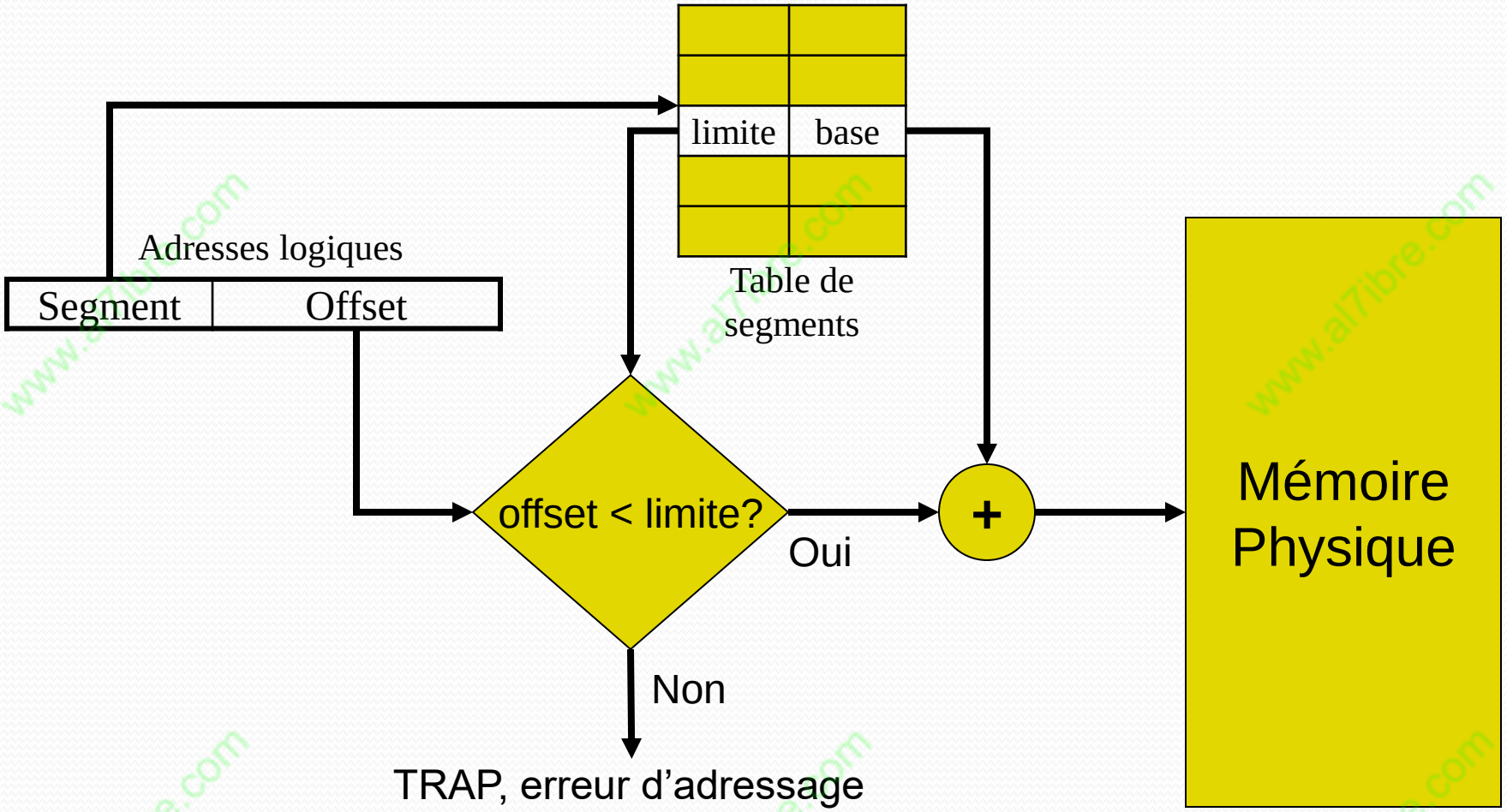
- **Un tableau contient l'adresse de début de tous les segments dans un processus**
- **Chaque adresse dans un segment est ajoutée à l 'adresse de début du segment par la MMU**




emacs
segment 0
data 1
segment 1
logical memory
process P_1
limit
base
0
25286
43062
1
4425
68348
segment table
process P_1
43062
editor
68348
data 1
72773
90003
data 2
98553
physical memory
VI
segment 0
data 2
segment 1
logical memory
process P_2
limit
base
0
25286
72773
1
8850
90003
segment table
process P_2

Segmentation

- Comment est-ce que les adresses sont traduites?

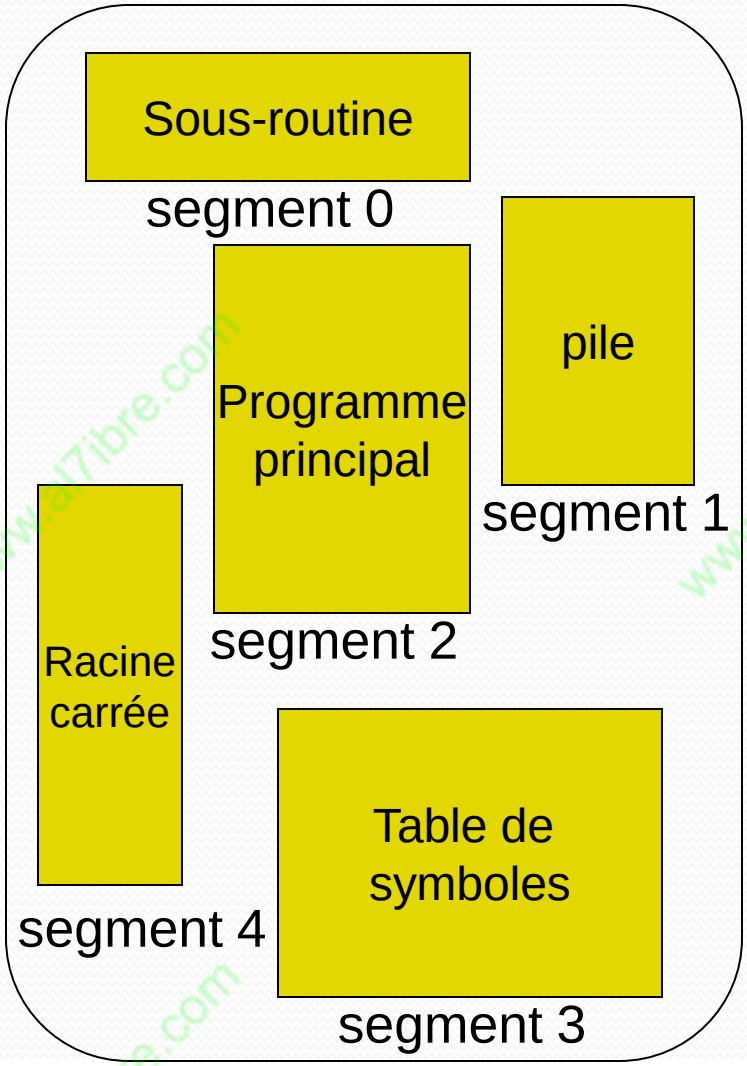


www.al7ibre.com

Segmentation et protection

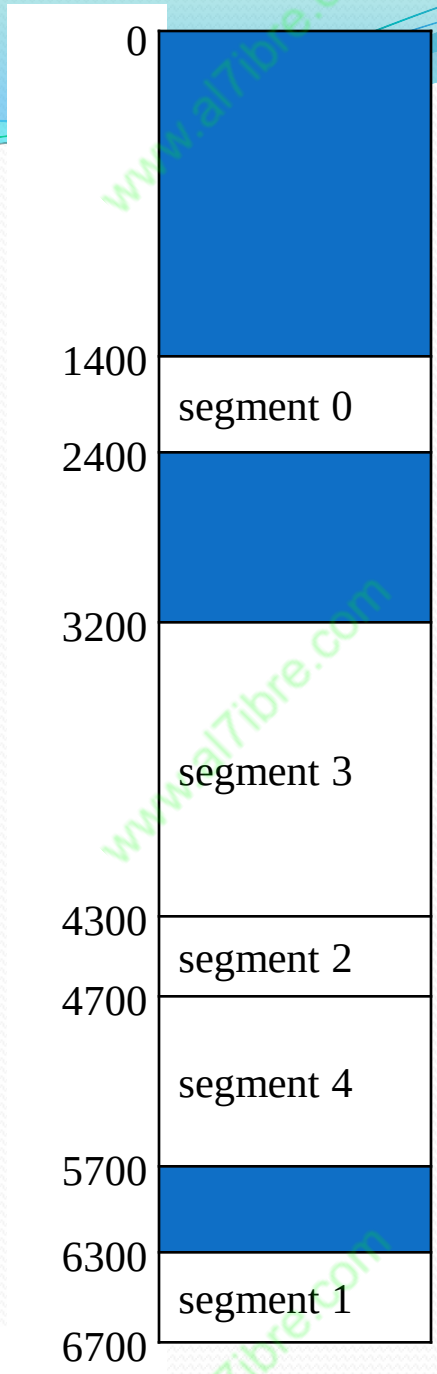
- Chaque entrée dans la table des segments peut contenir des infos de protection:
  - longueur du segment
  - privilèges de l`usager sur le segment: lecture, écriture, exécution
    - Si au moment du calcul de l'adresse on trouve que l'usager n'a pas droit d'accès→interruption
    - ces infos peuvent donc varier d'un usager à autre, par rapport au même segment!

Segmentation



Espace d'adresses logiques

	limite	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700



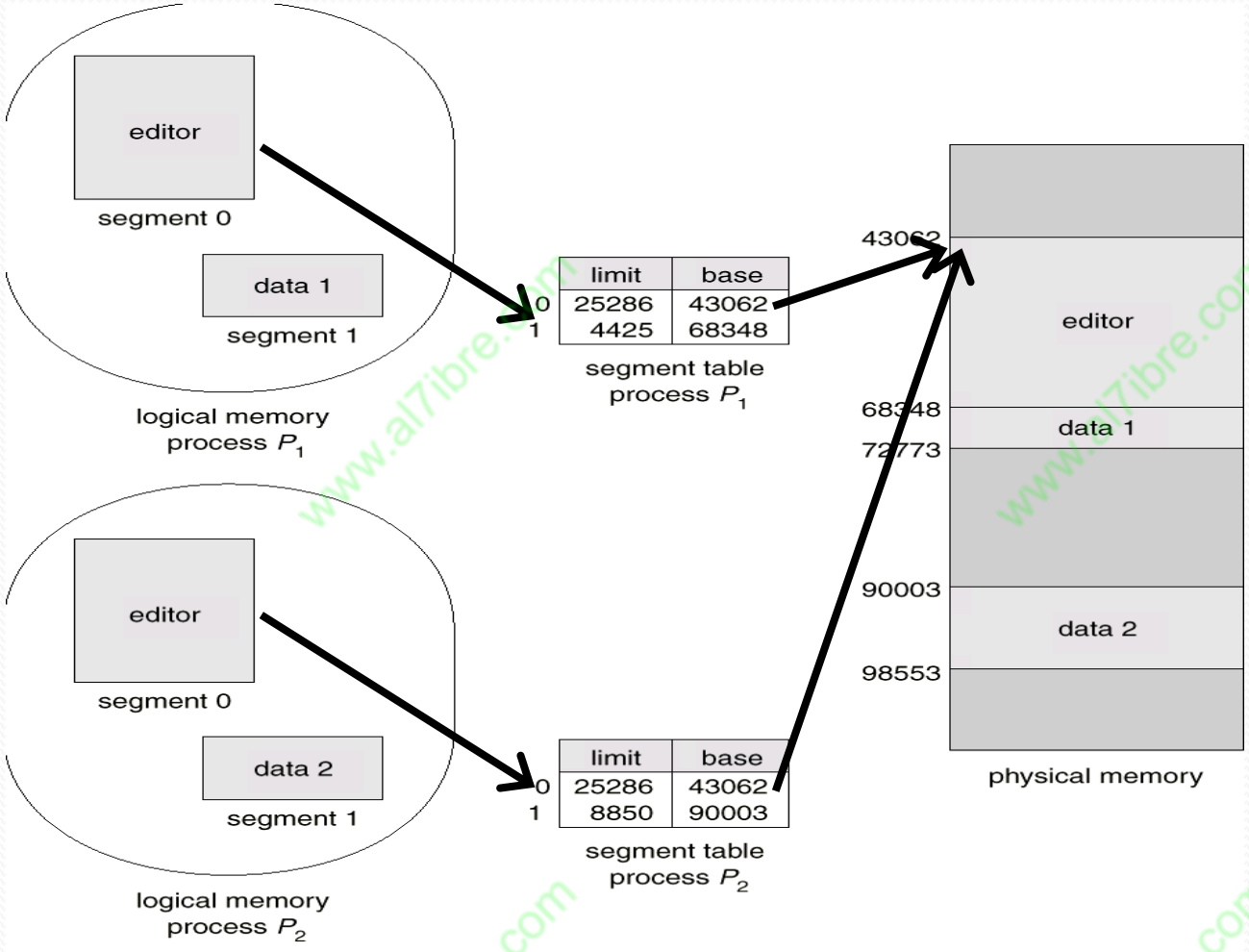
Mémoire physique

Segmentation

- Avantages:
  - Il peut être possible d'agrandir ou de les rapetisser
  - On peut donner à chaque segment sa propre information de protection … Ceci est beaucoup plus facile que d'essayer de protéger chaque page en mémoire
  - Lier les programmes (linking) est une tâche triviale
  - Le code peut être partagé entre les processus plus facilement. On charge le segment de code seulement une fois. Les copies du même programme accède le même segment

www.al7ibre.com

Partage de segments: le segment 0 est partagé



P.ex: DLL utilisé par plus usagers

Évaluation de la segmentation simple

- **Avantages: l'unité d'allocation de mémoire (segment) est**
  - plus petite que le programme entier
  - une entité logique connue par le programmeur
  - les segments peuvent changer de place en mémoire
- **Désavantage: le problème des partitions dynamiques:**
  - La fragmentation externe n'est pas éliminée:
    - trous en mémoire, compression?

www.al7ibre.com

Segmentation

- Désavantages:
  - Tout comme les systèmes de permutation, la *fragmentation* peut gaspiller de la mémoire.
  - Les segments peuvent être trop larges pour entrer dans la mémoire physique
- Nous connaissons déjà une solution pour adresser les programmes qui sont plus grands que la mémoire physique?

www.al7ibre.com

Mémoire virtuelle

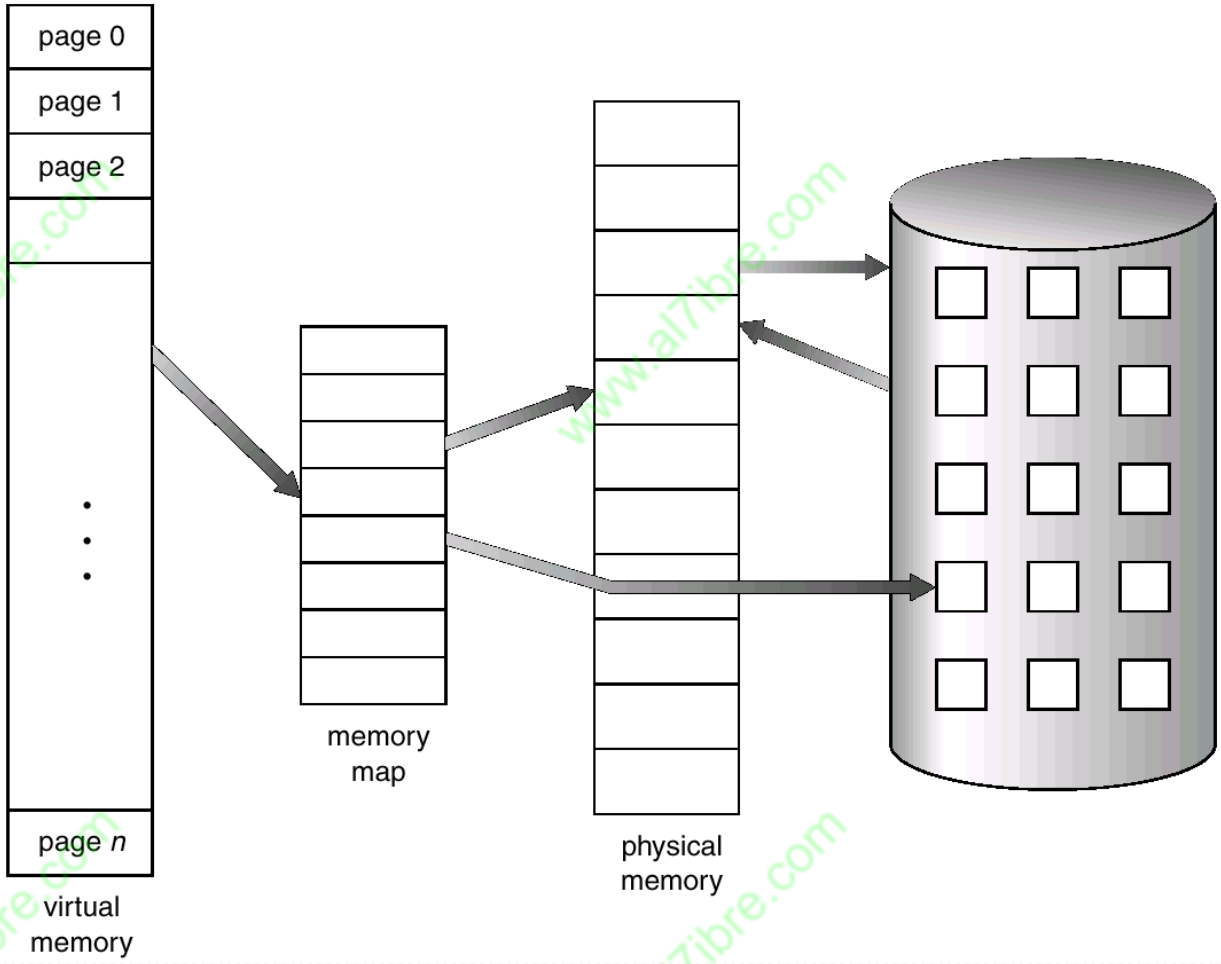
- La permutation de processus (entiers) nous a permit d'avoir plus d'un programme en mémoire en même temps, mais nous n'avons pas adressé le problème d'un programme trop large pour la mémoire

- La solution qui a fait surface s'appel la ***mémoire virtuelle***. L'idée principale est de garder une partie du programme en mémoire et une partie sur le disque
  - Maintenant un programme de 16MO peut tourner sur un système qui a seulement 4MO de RAM!
  - Et en plus on peut avoir une quantité de processus qui tournent en même temps et que la mémoire RAM ne peut contenir

www.al7ibre.com

Mémoire virtuelle:

résultat d'un mécanisme qui combine la mémoire principale et les mémoires secondaires



www.al7ibre.com

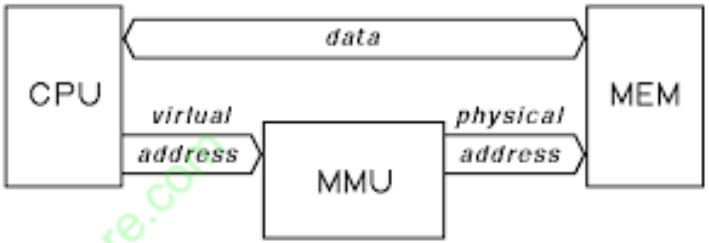
Pagination

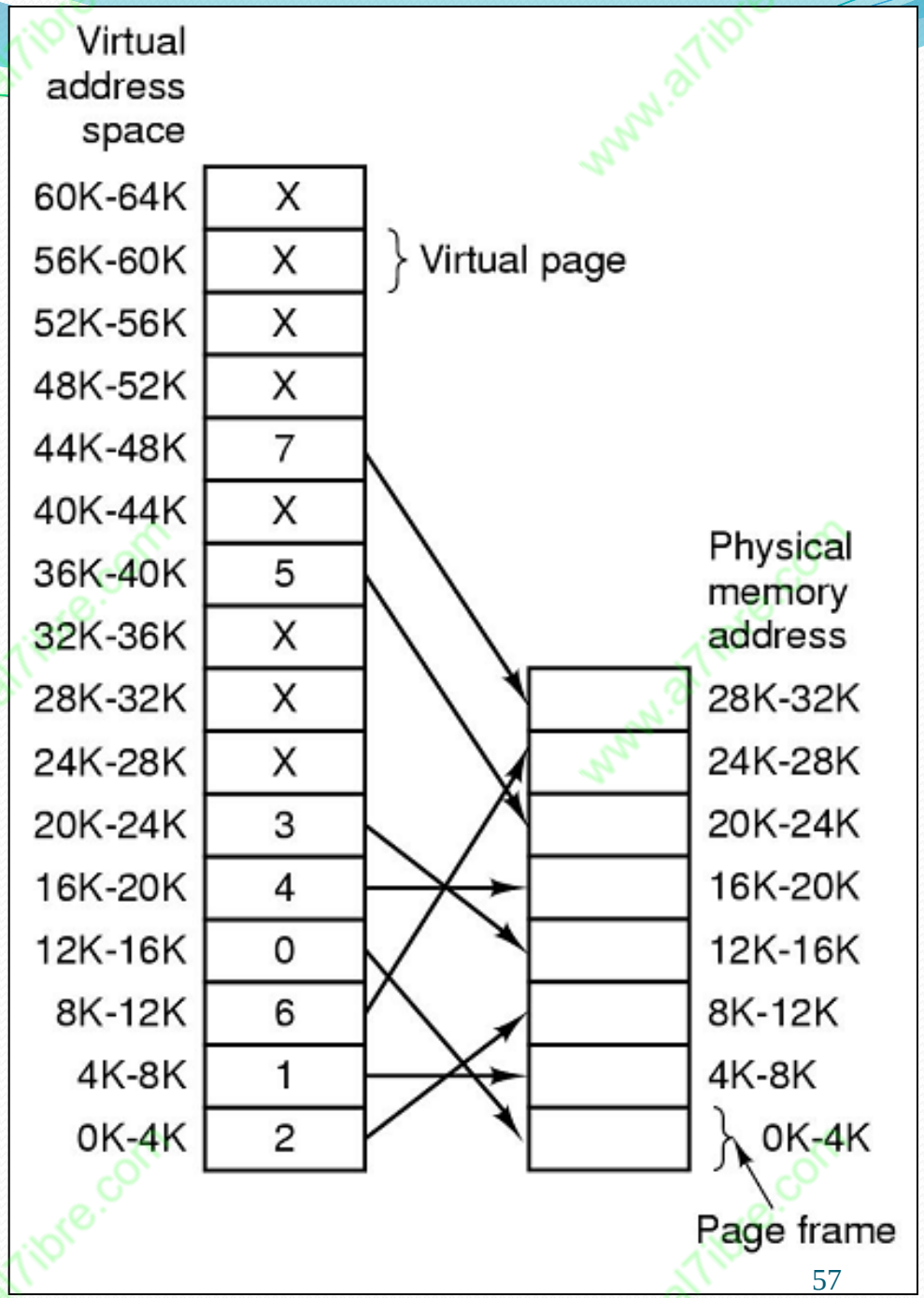
- Les processus croient qu'ils peuvent accéder la mémoire entière qui leur est donnée, basé sur le nombre de lignes d'adresses sur le matériel
- Pour être capable de donner des "sections" de mémoire qui peuvent être déplacées entre le disque et la mémoire physique nous divisons l'espace d'adresses virtuelle en morceaux que l'on appel **pages**
  - Le morceau de mémoire physique qui correspond à une page est un **cadre de page** (**Page Frame**) ou tout simplement cadre

www.al7ibre.com

Pagination

- Cet ordinateur en particulier a seulement 32KO de RAM
- Le MMU mappe les adresses virtuelles utilisées par le CPU en adresses physiques qui sont mises sur le bus





Pagination

- Manques (un X) veulent dire que la page n'est pas en mémoire et le MMU Trap au SE pour que la page soit chargée
  - Ceci est un **défaut de page**
  - Quand cela ce produit le SE charge la nouvelle page dans un cadre de page (possiblement en expulsant une page qui est en utilisation) et met à jour le MMU avec la nouvelle information
    - La place où cette information est stockée s'appelle **Table de pages**

www.al7ibre.com

Tables de pages

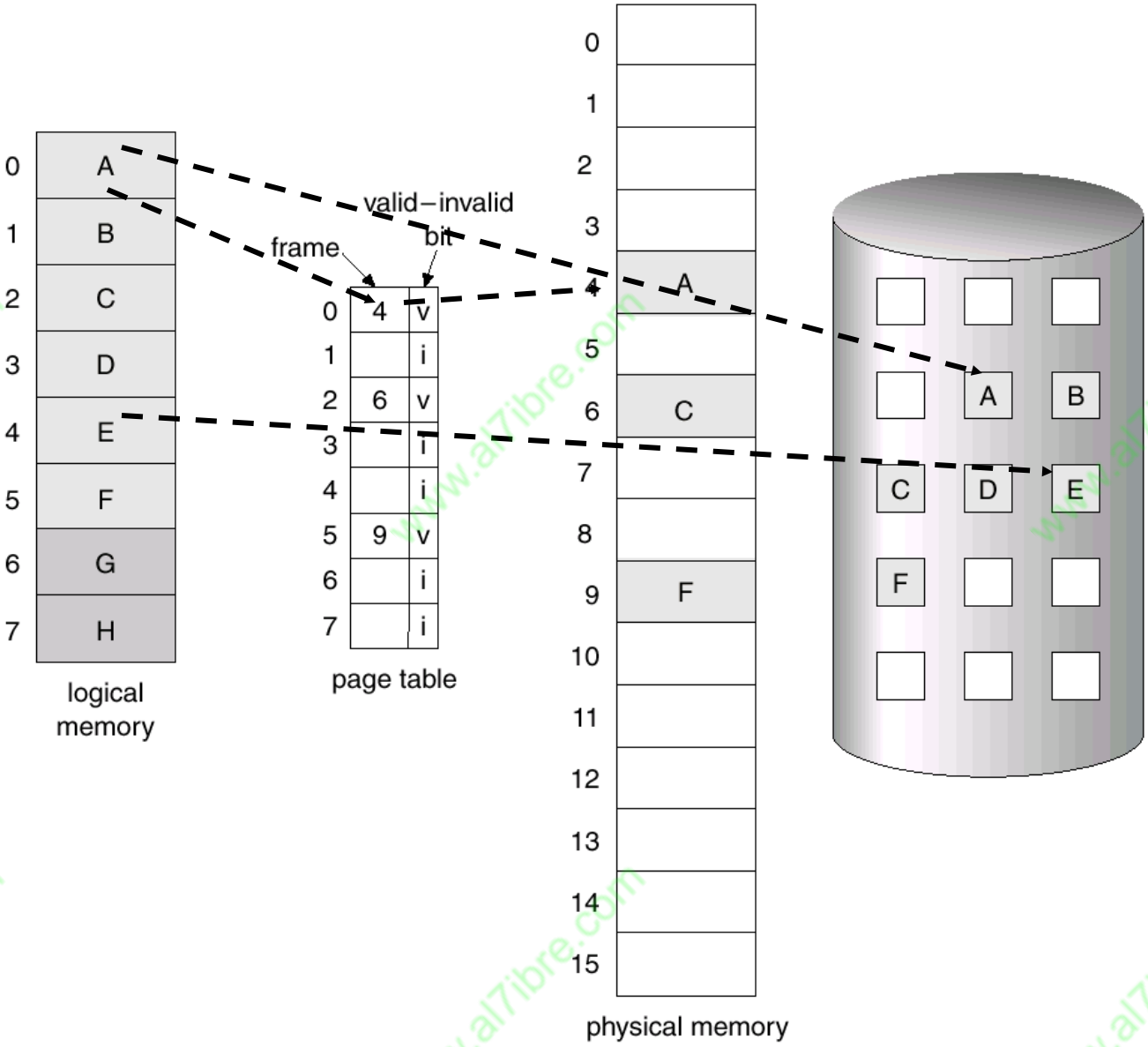
- La table de pages stock un nombre d'entrées, une pour chaque page en mémoire virtuelle, indiquant si la page est dans la mémoire physique (notez qu' une table de pages est requise pour chaque processus!)

www.al7ibre.com

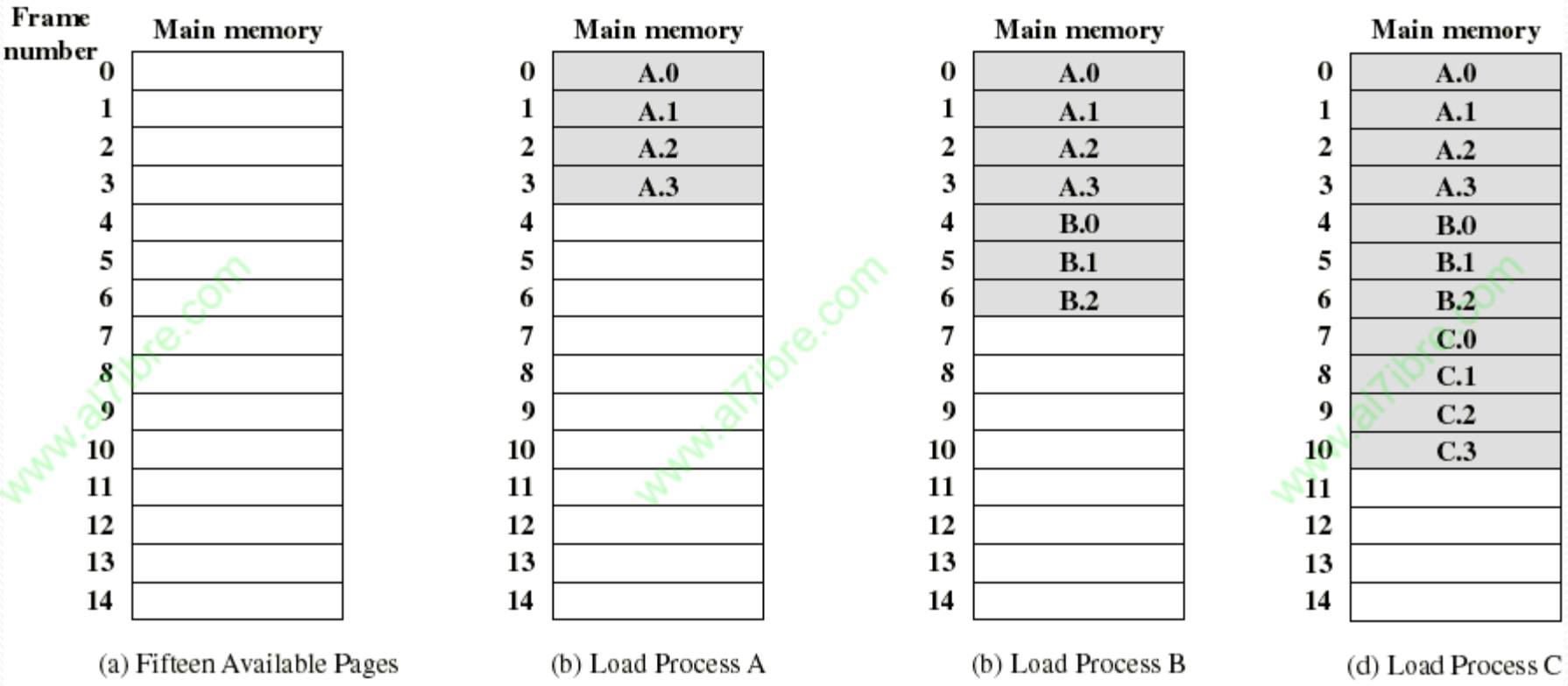
Pages

Page A en RAM **et** sur disque

Page E **seulement** sur disque



Exemple de chargement de processus



- **Supposons que le processus B se termine ou est suspendu**

Exemple de chargement de processus

- **Nous pouvons maintenant transférer en mémoire un processus D, qui demande 5 cadres**
  - bien qu`il n'y ait pas 5 cadres contigus disponibles
- **La fragmentation externe est limitée au cas que le nombre de pages disponibles n'est pas suffisant pour exécuter un programme en attente**
- **Seule la dernière page d'un processus peut souffrir de fragmentation interne**

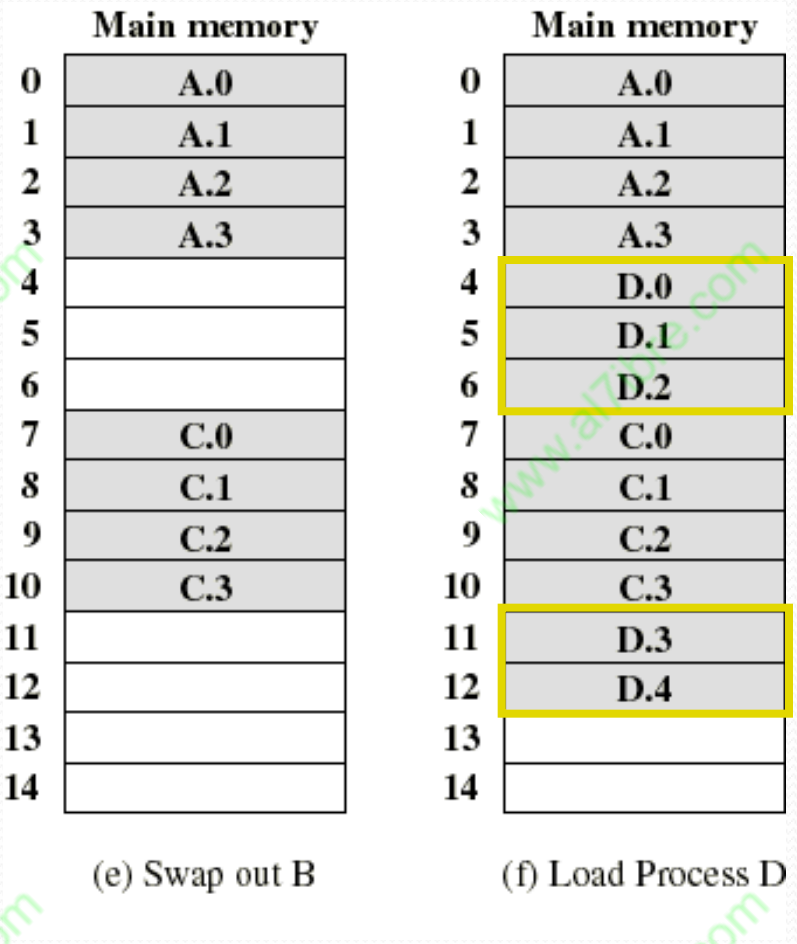


Table de pages

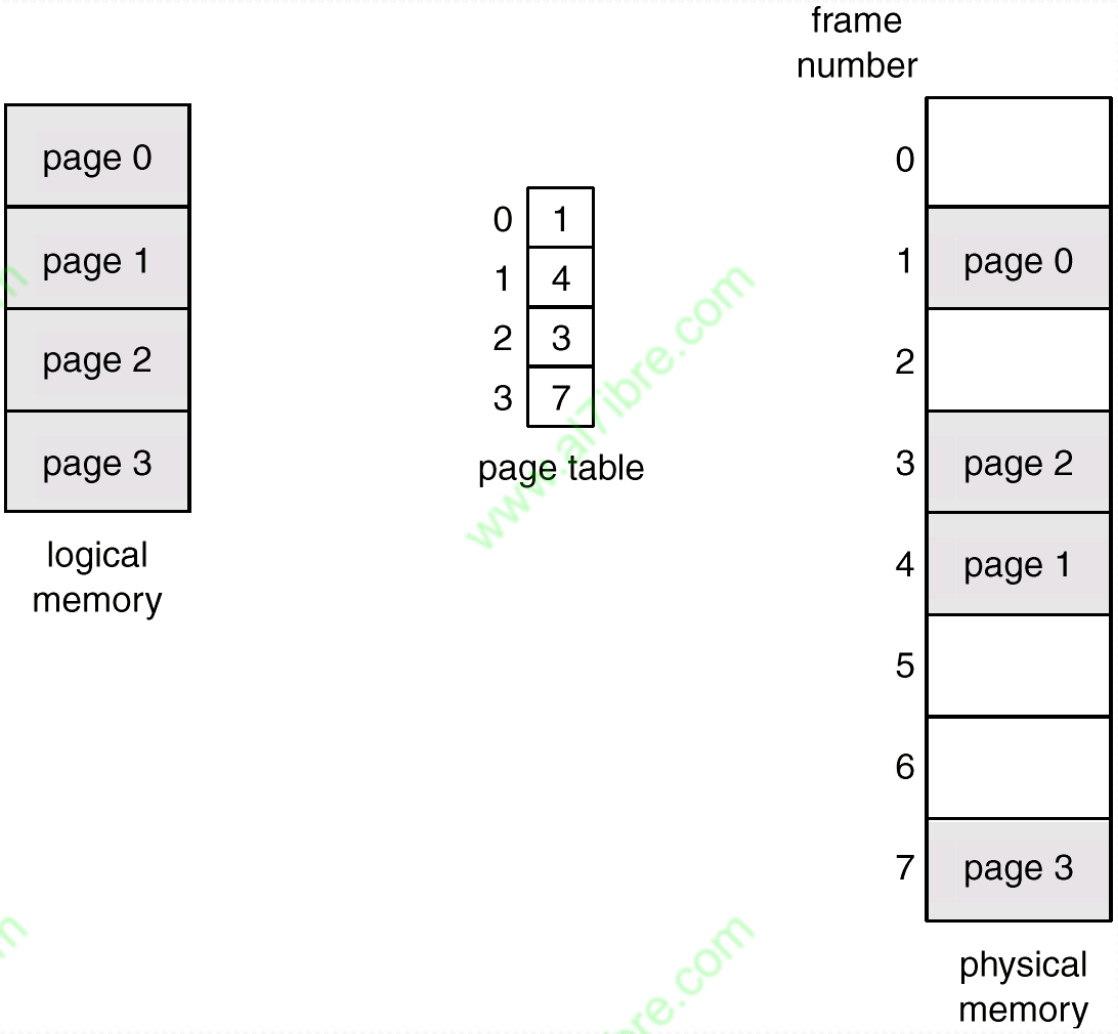
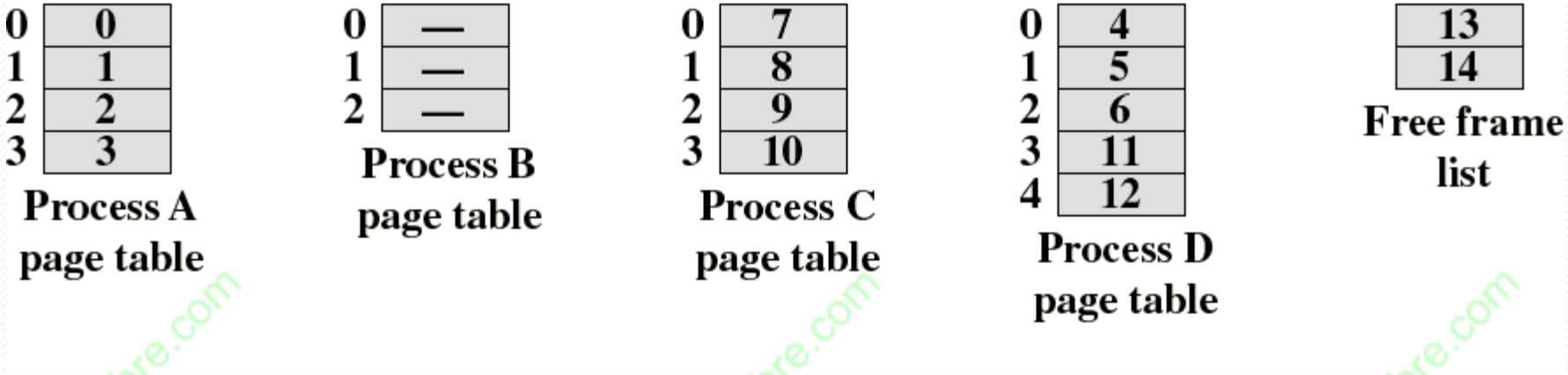


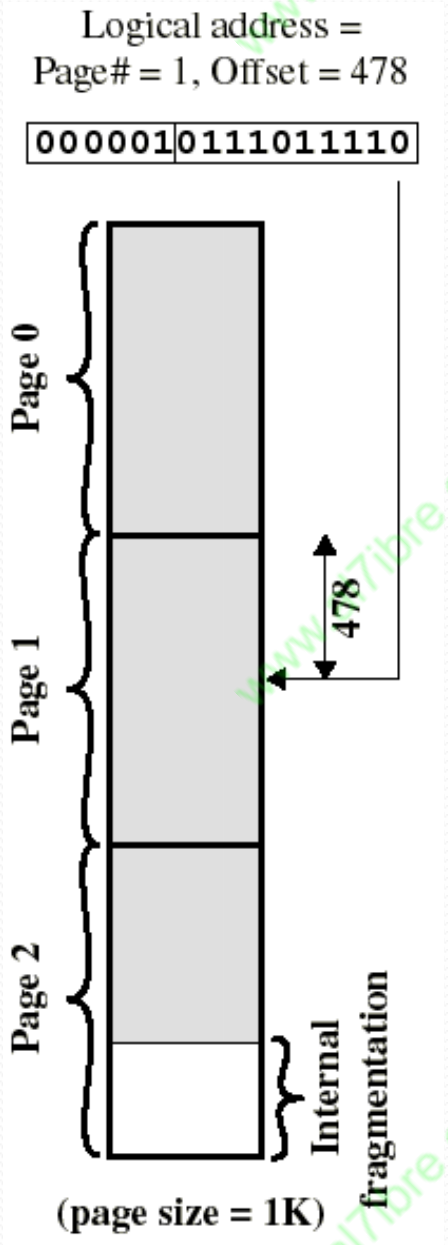
Table de pages



- **Le SE doit maintenir une table de pages pour chaque processus**
- **Chaque entrée d'une table de pages contient le numéro de cadre où la page correspondante est physiquement localisée**
- **Une table de pages est indexée par le numéro de la page afin d'obtenir le numéro du cadre**
- **Une liste de cadres disponibles est également maintenue (free frame list)**

Adresse logique (pagination)

- **L'adresse logique est facilement traduite en adresse physique car la taille des pages est une puissance de 2**
- **L'adresse logique (n,d) est traduite en adresse physique (k,d) en utilisant n comme index sur la table des pages et en le remplaçant par l'adresse k trouvée**
  - d ne change pas

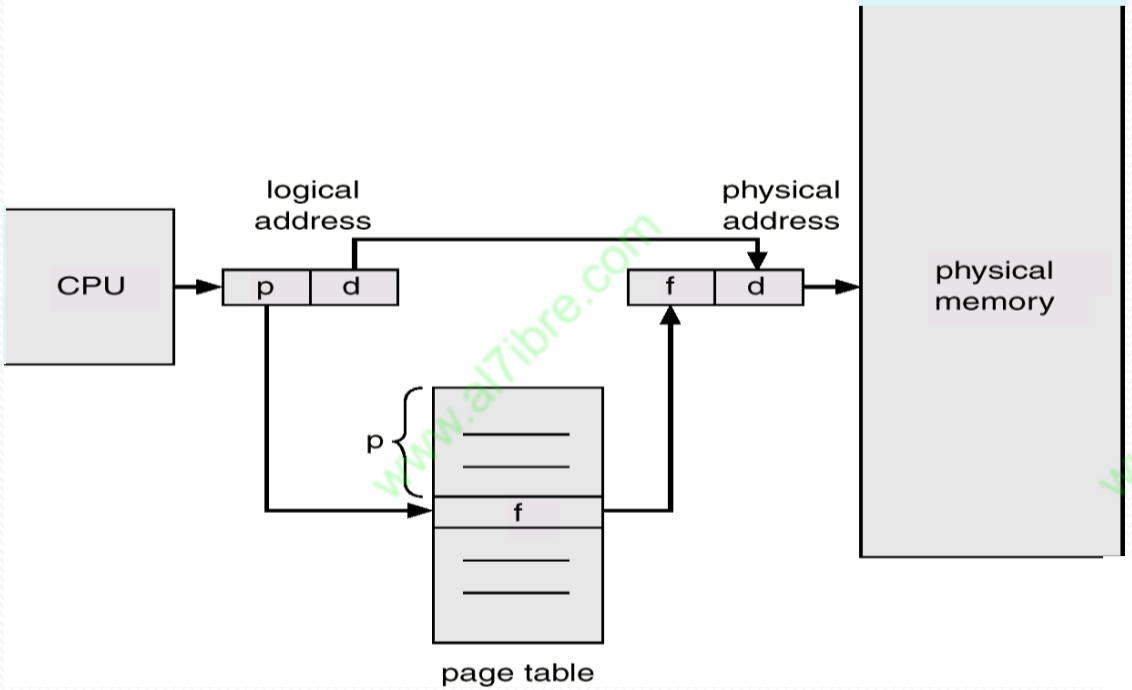


Adresse logique

- **Donc les pages sont invisibles au programmeur, compilateur ou assembleur (seule les adresses relatives sont employées)**
- **La traduction d'adresses au moment d'exécution est facilement réalisable par le matériel:**
  - l'adresse logique (n,d) est traduite en une adresse physique (k,d) en indexant la table de pages et en annexant le même décalage d au numéro du cadre k
- **Un programme peut être exécuté sur différents matériels employant dimensions de pages différentes**

www.al7ibre.com

Mécanisme: matériel



Traduction d'adresses: pagination
nous *ajoutons* le décalage à l'adresse de la page.

Tables de Pages

- Quelles sont les entrées dans la table de pages?
  - **Bit de présence** (pour que l'on puisse savoir si on va sur le disque ou non…aussi appelé bit de validité)
  - **Bit de modification (Dirty bit)** (pour que l'on puisse savoir si la page a été modifiée et doit être réécrite sur le disque durant la pagination.)
  - **Bit d'utilisation?** (nous aide à décider quelle page nous allons permuter… nous allons en voir plus sur cela avec les algorithmes de remplacement de pages)
    - Bit est mise à jour par une lecture ou une écriture
  - **Bits de protection** – Peut spécifier qu'est-ce que l'utilisateur peut faire avec la page ie: lire / écrire / exécuter

www.al7ibre.com

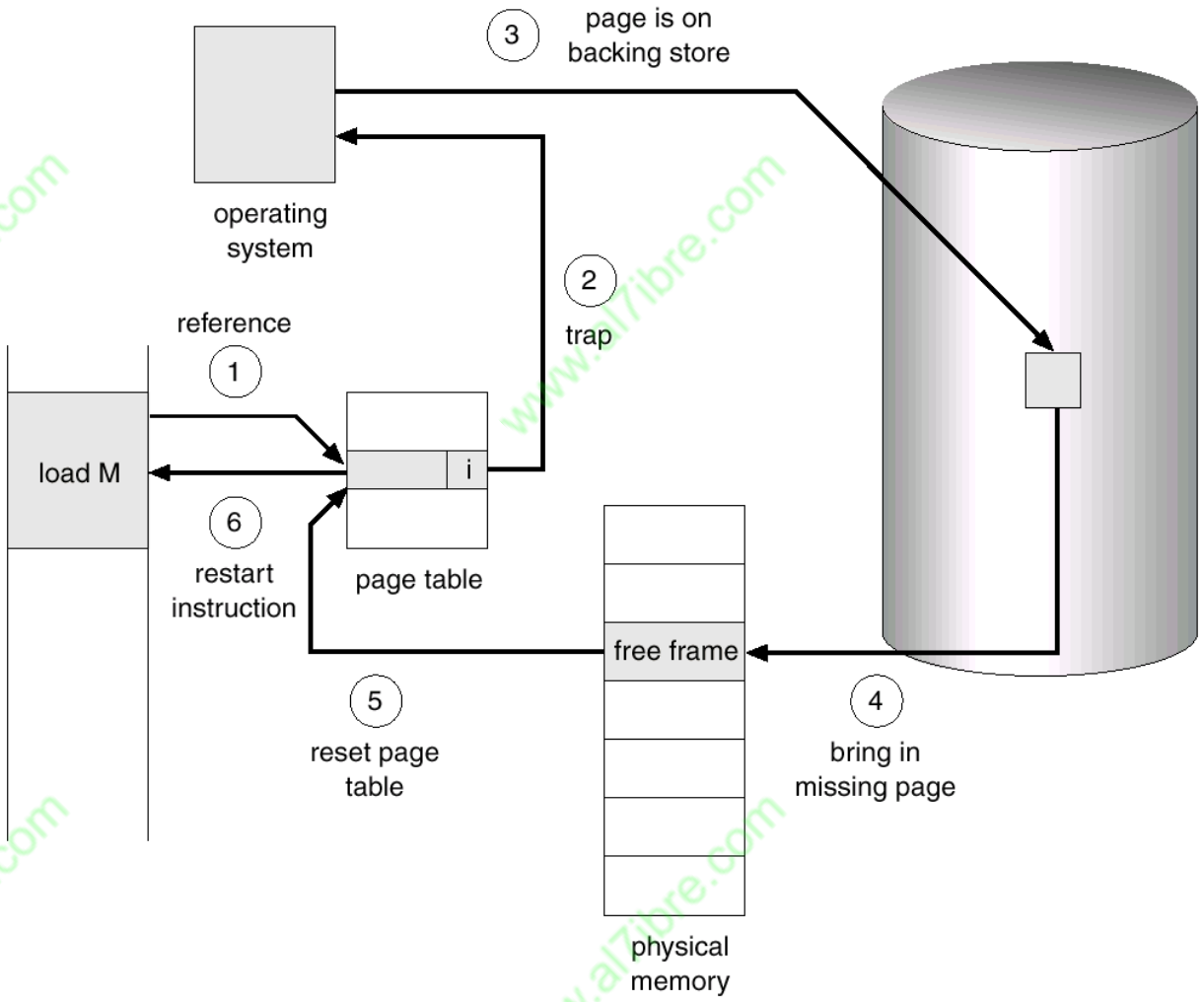
Tables de Pages

- Où ailleurs pourrait-on stocker l'information de la table pour le processus courant?
  - Dans la mémoire principale
    - Plus lent que les registres
  - Sur le disque
    - Vraiment trop lent pour nos besoins, mais on peut tricher un peut
  - Une combinaison
    - Une mixture de registres, mémoire, et disque peut être utilisée dans un système (pagination à niveaux multiples).

www.al7ibre.com

Exécution d'une défaut de page:

va-et-vient plus en détail

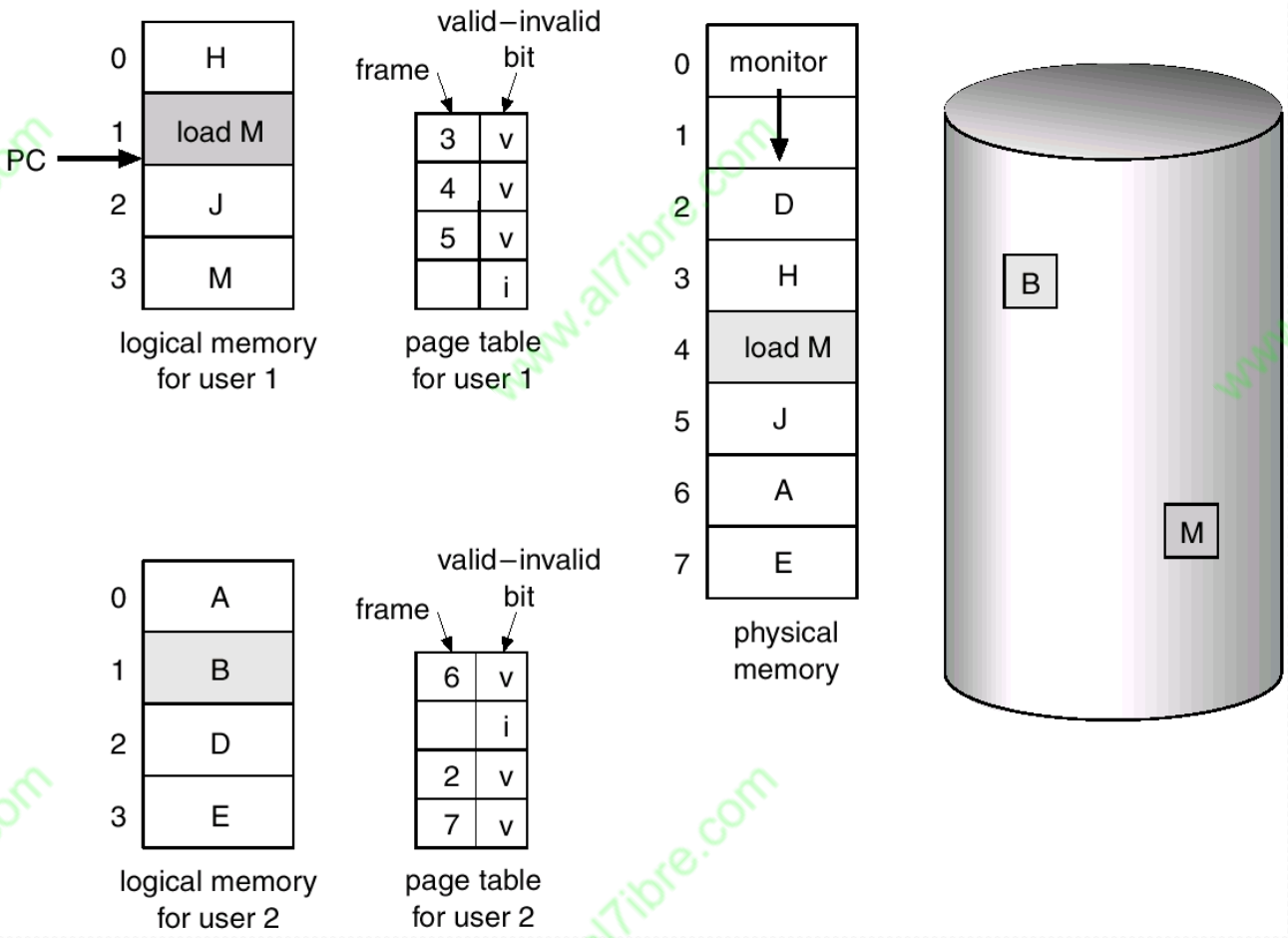


Séquence d'événements pour défaut de page

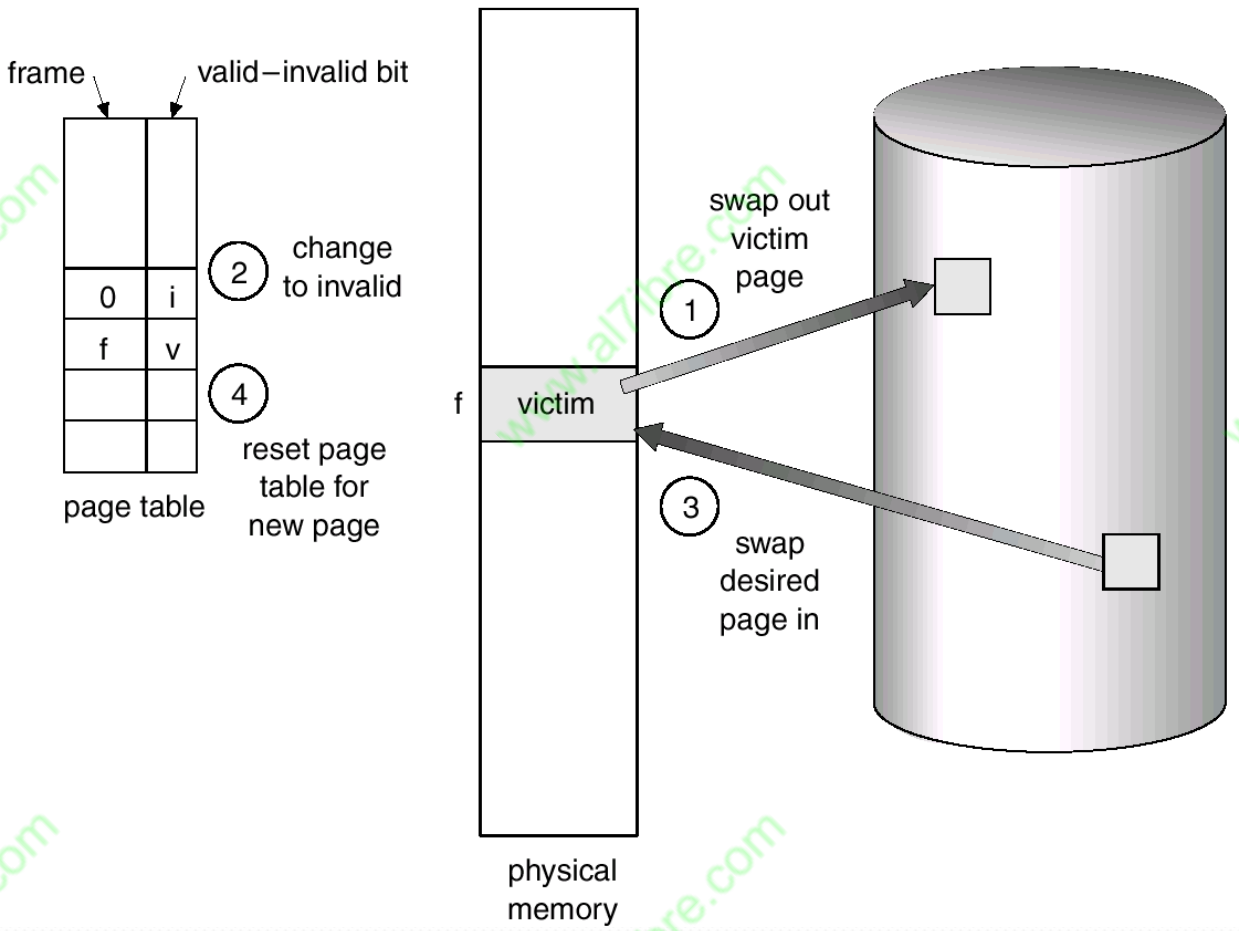
- Trappe au SE: page demandée pas en RAM
- Sauvegarder le PCB
- Un autre processus peut maintenant avoir l'UCT
- SE trouve la page sur disque
- lit la page du disque dans un cadre de mémoire libre (supposons qu`il y en un!)
  - exécuter les opérations disque nécessaires pour lire la page
- L'unité disque a complété le transfert et interrompt l'UCT
  - sauvegarder le PCB du processus s'exécutant
- SE met à jour le contenu du tableau des pages du processus qui a causé le défaut de page
- Ce processus devient prêt=ready
- la page désirée étant en mémoire, il pourra maintenant continuer

www.al7ibre.com

Quand la RAM est pleine et que nous avons besoin d`une page qui ne se trouve pas en RAM



La page victime...



Algorithmes de replacement de pages

- On se rappel: la pagination suit un ensemble de règles:
  - Elle permet à un programme d'être chargé en mémoire une page à la fois
  - Il y a une table qui identifie quelle page est chargée dans quel cadre
  - Quand une page est demandée et qu'elle n'est pas en mémoire physique, un défaut de page se produit
    - Le SE doit maintenant charger la page dans la mémoire. Si il n'y a pas de cadre libre, une page doit être évincée de la mémoire. Quelle page est choisit pour être évincée?

Algorithmes de replacement de pages

- Il existe un ensemble d'algorithmes qui peuvent être utilisés pour choisir quelle page va être la meilleure candidate pour l'éviction
- Considérations:
  - Les pages qui ont été modifiées doivent être écrites sur le disque avant l'éviction
  - L'utilisation de l'information d'état tel que les bits d'utilisations (used) et de modifications (dirty) vont être utiles pour prendre cette décision
  - Ces algorithmes sont applicables à d'autres domaines de recherche: caches, serveurs Web, etc…

www.al7ibre.com

Critères d'évaluation des algorithmes

- Les algorithmes de choix de pages à remplacer doivent être conçus de façon à minimiser le taux de défaut de pages à long terme
- Mais il ne peuvent pas impliquer des temps de système excessifs, p.ex. mise à jour de tableaux en mémoire pour chaque accès de mémoire

www.al7ibre.com

Exemple pour évaluation des algorithmes

- Nous allons expliquer et évaluer les algorithmes en utilisant la chaîne de référence pages suivante :

2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2

- Attention: les séquences d'utilisation pages ne sont pas aléatoires...
- L'évaluation sera faite sur la base de cet exemple, évidemment pas suffisant pour en tirer des conclusions générales

www.al7ibre.com

Algorithmes de replacement de pages

- L'algorithme de remplacement de pages optimal
  - Un algorithme théorique qui représente la décision absolue, sans aucun doute, le meilleur choix pour évincer une page
  - On évince la page qui serait la dernière à être utilisée, basé sur les pages en mémoire à l'instant de la décision

www.al7ibre.com

Algorithmes de replacement de pages

- L'algorithme de remplacement de pages optimal
  - **Est-ce que nous pouvons implémenter cet algorithme?**
    - Absolument pas. Si nous pouvions déterminer quand chaque page est requise dans le futur basé sur l'état courant, l'algorithme serait facile à implémenter.
  - **Cependant, si un programme est utilisé pour un ensemble particulier d'E/S, il est possible de tracer quelles pages sont requises et dans quel ordre**
    - Cet enregistrement peut être utilisé pour des tests de performance pour comparer nos algorithmes réalisables à ceux l'algorithme optimal

www.al7ibre.com

Algorithmes de replacement de pages

- Premier arrivé, premier sortie PAPS (FIFO)
  - Facile à implémenter. Garde une liste de toutes les page en mémoire en ordre quelles sont arrivées
  - Sur un défaut de page, la page la plus vielle est enlevée et une nouvelle page est ajoutée à la fin de la liste
  - Avantage: Très facile à implémenter
  - Désavantage: Aucune façon de déterminer si la page qui est enlevée est en très utilisée ou pas. L'âge peut être une indication mais n'est pas nécessairement la meilleure indication de l'utilisation d'une page

www.al7ibre.com

Algorithmes de replacement de pages

- Algorithme de deuxième chance
  - Cet algorithme est une modification du PAPS pour le rendre possiblement plus raisonnable
  - Avant d'évincer la page la plus vielle, on vérifie la bit d'utilisation
    - Si la page est en utilisation, même si elle est vielle, elle se voit donner une deuxième chance et son entrée est déplacée vers la fin de la liste, ce qui a pour effet de la rendre comme une nouvelle page. Son bit d'utilisation est remis à zéro à ce moment
  - La recherche continue de cette façon jusqu'à ce qu'une veille page qui n'a pas été utilisé est trouvé

www.al7ibre.com

Algorithmes de replacement de pages

- Algorithme de remplacement de pages de l'horloge
  - L'algorithme de la deuxième chance peut être lent parce qu'il déplace constamment les pages dans la liste chaînée pour garder les pages dans le bon ordre.
  - Une meilleur approche est de garder les entrées de pages dans une liste circulaire (on peut penser à une sorte d'horloge). Une main pointe à la page la plus vielle.
  - Cette main n'est rien de plus qu'un pointeur à une entrée dans la liste de pages en mémoire

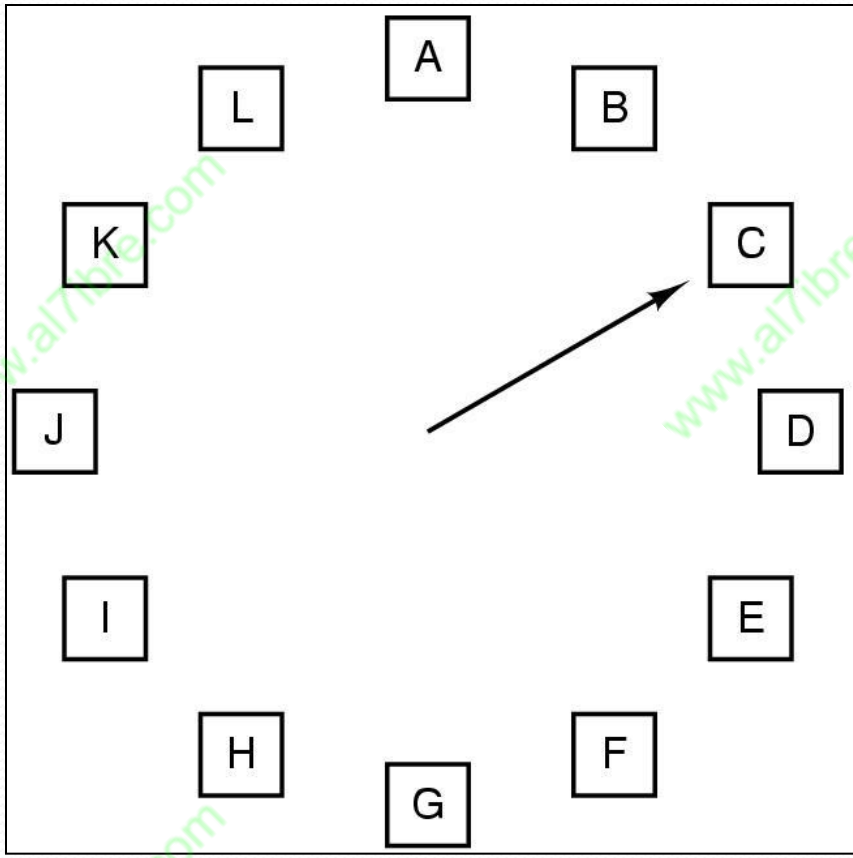
www.al7ibre.com

Algorithmes de replacement de pages

- L'algorithme de l'horloge
  - Quand un défaut de page est détecté, la page qui est pointée par la 'main' est inspectée. Si elle n'a pas été utilisée, elle est évincée et la main avance à la prochaine position
  - Si la page est en utilisation, le bit d'utilisation est remis à zéro et la main est avancé à la prochaine position pour faire une autre vérification
  - Ceci continue jusqu'à ce qu'une page soit trouvée pour être évincée

www.al7ibre.com

Algorithmes de replacement de pages



When a page fault occurs, the page the hand is pointing to is inspected. The action taken depends on the R bit:

R = 0: Evict the page

R = 1: Clear R and advance hand

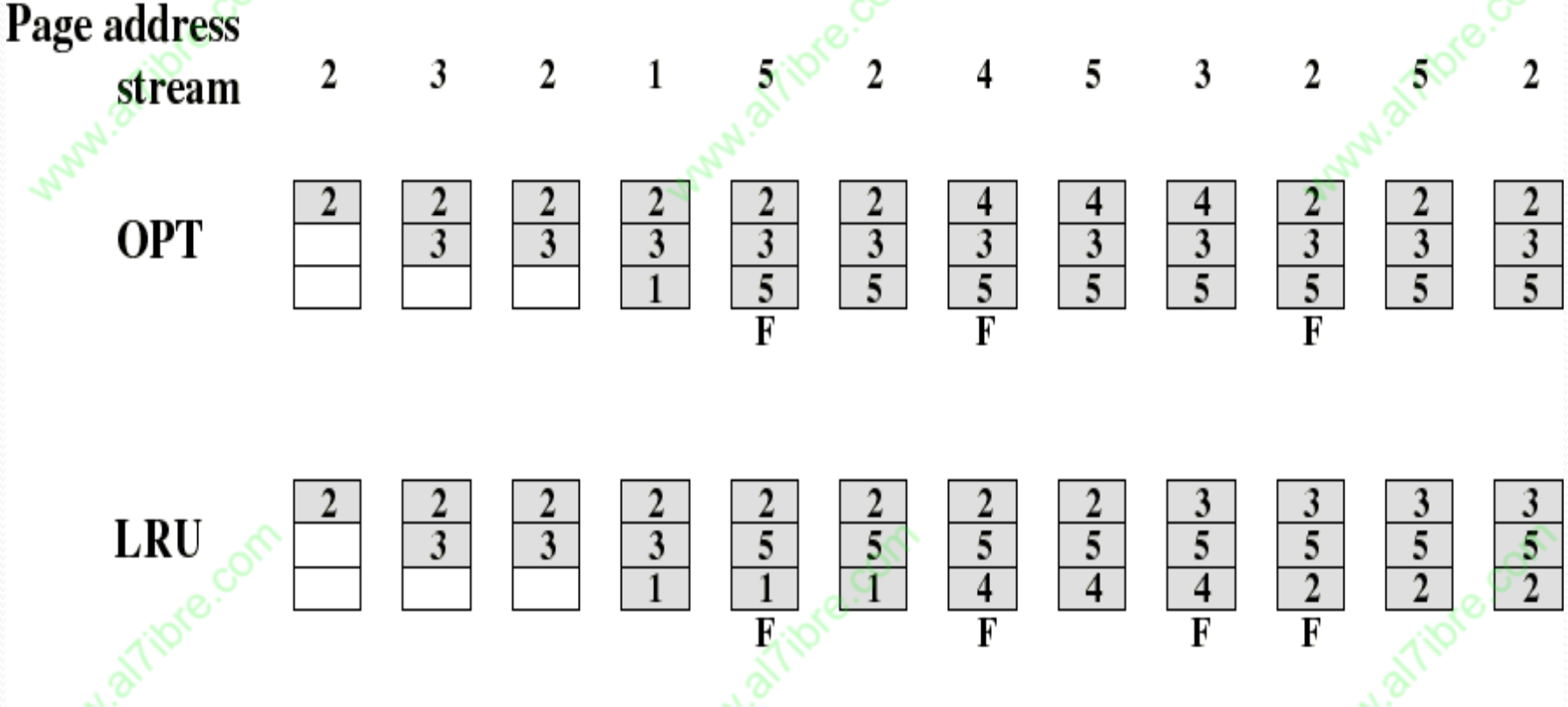
Algorithmes de replacement de pages

- L'algorithme de remplacement de la page la moins récemment utilisée (MRU) (LRU)
  - Une bonne approximation de l'algorithme optimal est qu'une page qui a grandement utilisée les dernières instructions va probablement être grandement utilisée dans les quelques prochaines instructions (et le contraire est vrai)
  - Donc, quand un défaut de page ce produit, on évince la page qui n'a pas été utilisé pour le plus long temps
    - Comment est-ce que cela diffère de PAPS?

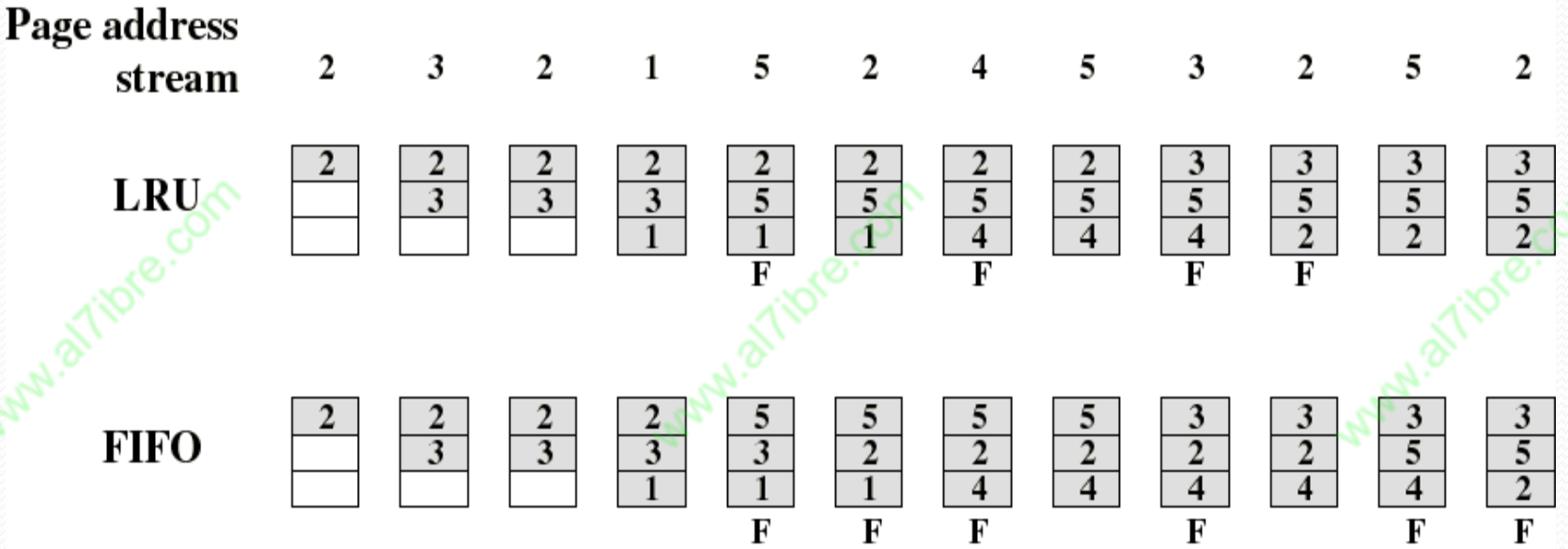
www.al7ibre.com

Comparaison OPT-LRU

- Exemple: Un processus de 5 pages s'ìl n'y a que 3 pages physiques disponibles.
- Dans cet exemple, OPT occasionne 3+3 défauts, LRU 3+4.



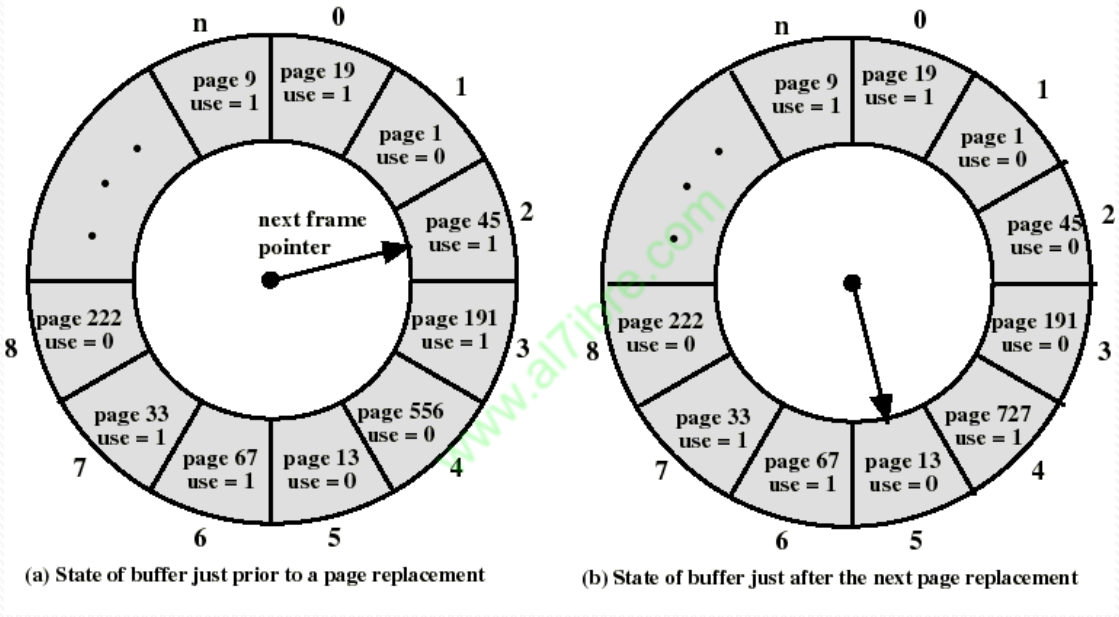
Comparaison de FIFO avec LRU



- Contrairement à FIFO, LRU reconnaît que les pages 2 and 5 sont utilisées fréquemment
- La performance de FIFO est moins bonne:

  dans ce cas, LRU = 3+4, FIFO = 3+6

Algorithme de l'horloge



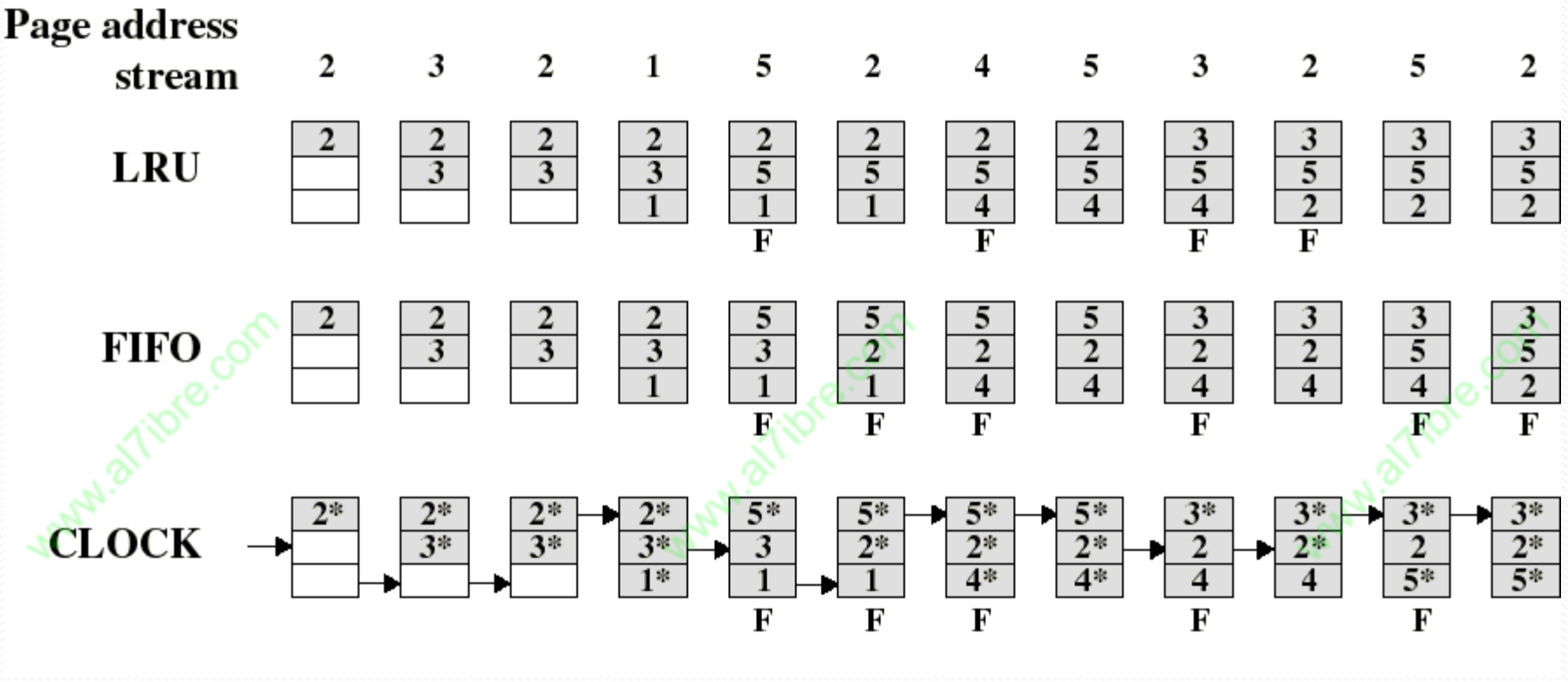
(a) State of buffer just prior to a page replacement

(b) State of buffer just after the next page replacement

La page 727 est chargée dans le cadre 4.

La prochaine victime est 5, puis 8.

Comparaison: Horloge, FIFO et LRU



- Astérisque indique que le bit utilisé est 1
- L'horloge protège du remplacement les pages fréquemment utilisées en mettant à 1 le bit "utilisé" à chaque référence
- LRU = 3+4, FIFO = 3+6, Horloge = 3+5

L’histoire jusqu’à maintenant…

- La solution de la mémoire virtuelle jusqu’à maintenant est la pagination
  - La pagination est un modèle de gestion de la mémoire qui est “plat” ce qui veut dire que les programmeurs voient les adresses qui commencent à 0 jusqu’à une adresse maximum
  - Pour certaines applications, il pourrait être utile de permettre différents espaces de mémoire à l’intérieur d’un seul processus...
    - Par exemple, dans un programme il pourrait y avoir des espaces de mémoire distincts pour le texte du programme, la pile et le tas

www.al7ibre.com

Segmentation avec pagination!

- Pour obtenir le meilleur des deux mondes, les segments peuvent être paginés
  - Élimine le problème de la fragmentation
  - Permet à des segments larges à être partiellement en mémoire
- Requis:
  - Chaque processus a besoin d'une table de segment
    - Cette table même peut être segmentée et paginée!
  - Chaque entrée dans la table de segment pointe à la table de pages pour ce segment
    - Tout comme avant, ceci peut être une table de pages multiniveaux

www.al7ibre.com

Chapitre 4

1. Systèmes d'entrée/sortie
2. Systèmes de fichiers
3. Structure de mémoire de masse (disques)

www.al7ibre.com

1. Systèmes d'entrée/sortie

Concepts importants :

- Matériel E/S
- Communication entre UCT et contrôleurs périphériques
- DMA
- Pilotes et contrôleurs de périphériques
- Sous-système du noyau pour E/S
  - Tamponnage, cache, spoule

www.al7ibre.com

Catégories de périphériques d'E/S

- Les périphériques d'E/S viennent en deux types généraux:
  - Périphériques par blocs
  - Périphériques par caractères
- Les périphériques par blocs stockent les données en blocs de taille fixe, chacun possédant sa propre adresse
  - Les disques sont la représentation la plus courante des périphériques de blocs
  - Parce que chaque bloc est adressable, chaque bloc peut être indépendamment lu/écrit des autres blocs

www.al7ibre.com

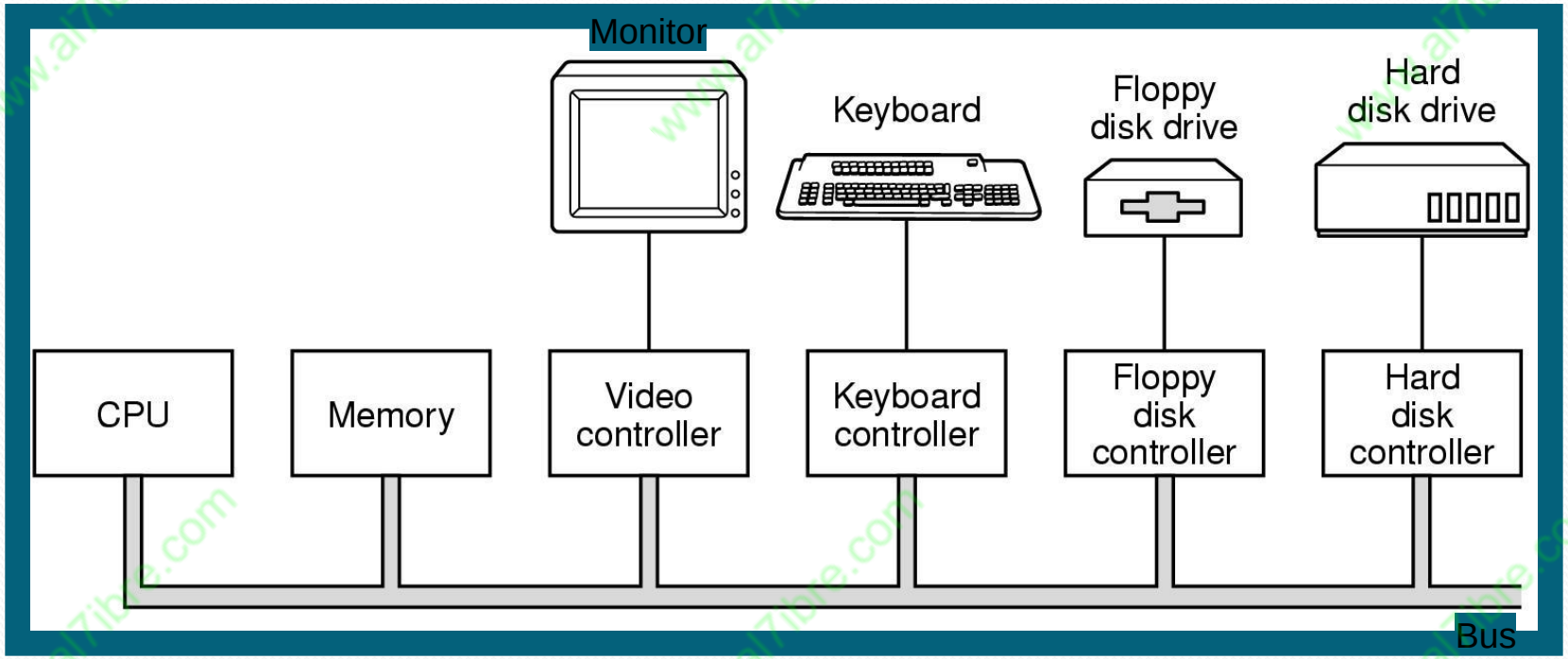
Catégories de périphériques d'E/S

- Les périphériques par caractères acceptent et fournissent des flots de caractères sans aucune structure
  - Non adressable
  - Aucune opération de recherche (seek)
  - Exemples: souris, imprimante, interfaces de réseau, modems,…
- Certains périphériques chevauchent les frontières:
  - les bandes magnétiques pour sauvegarder entreposent des blocs de données de disques, mais l'accès est séquentiel
- Certains périphériques ne font pas dans les modèles:
  - Écran: n'ont pas de blocs ou de flots, mais ont de la mémoire mappée

www.al7ibre.com

Contrôleurs de périphériques

- On se rappel: Les périphériques d'E/S ont typiquement une composante mécanique et une composante électronique
  - La partie électronique est le contrôleur



Contrôleurs de périphériques

- Sur un PC, le contrôleur de périphérique est habituellement sur un circuit imprimé
  - Il peut être intégré sur la carte mère
- Le job du contrôleur est de convertir un flot de série de bits en octets ou en blocs d'octets et de faire les conversions et corrections
  - En fin de compte tous les périphériques traitent des bits. C'est le contrôleur qui groupe ou dégroupe ces bits

www.al7ibre.com

Le logiciel d'E/S ont des couches

- Pilotes de périphériques
  - Chaque périphérique d'E/S attaché à l'ordinateur requiert du code spécifique pour faire l'interface entre le matériel et le SE. Ce code s'appel pilote de périphérique
    - Ceci est parce que au niveau du matériel, les périphériques sont radicalement différents les uns des autres
    - Parfois un pilote va prendre soins d'une classe de périphériques qui sont proche ex.: un nombre de souris
  - Les pilotes de périphériques sont normalement produit par le manufacturier du périphérique pour les SEs populaires

www.al7ibre.com

Le logiciel d'E/S ont des couches

- Que font les pilotes de périphériques?
  - Ils acceptent les commandes abstraites de lecture/écriture de la couche supérieure
  - Fonctions assorties:
    - Initialise le périphérique
    - Gère la puissance – Arrête un disque de tourner, ferme un écran, ferme une caméra, etc.

www.al7ibre.com

Le logiciel d'E/S ont des couches

- Qu'est-ce qu'un pilote fait sur une lecture/écriture?
  - Vérifie les paramètres d'entrée & retourne les erreurs
  - Converti les commandes ***abstraites*** (lit du secteur) en commandes ***physiques*** (tête, traque, secteur, et cylindre)
  - Met les demandes dans une queue si le périphérique est occupé
  - Amène le périphérique en état de fonctionnement si requis
  - Contrôle le périphérique en envoyant des commandes par les registres de contrôle

www.al7ibre.com

Le logiciel d'E/S ont des couches

- Qu'est-ce que un pilote fait sur une lecture/écriture?
  - Quand une demande est envoyée, une des deux solutions possibles peut arriver:
    - Le pilote doit attendre pour la demande se termine, donc le pilote ***bloque***. Il va se réveiller plus tard,
    - Le résultat est instantané (ex.: écriture dans l'espace de mémoire de l'écran) donc le travail ***continue*** jusqu'à ce que l'E/S soit terminé

www.al7ibre.com

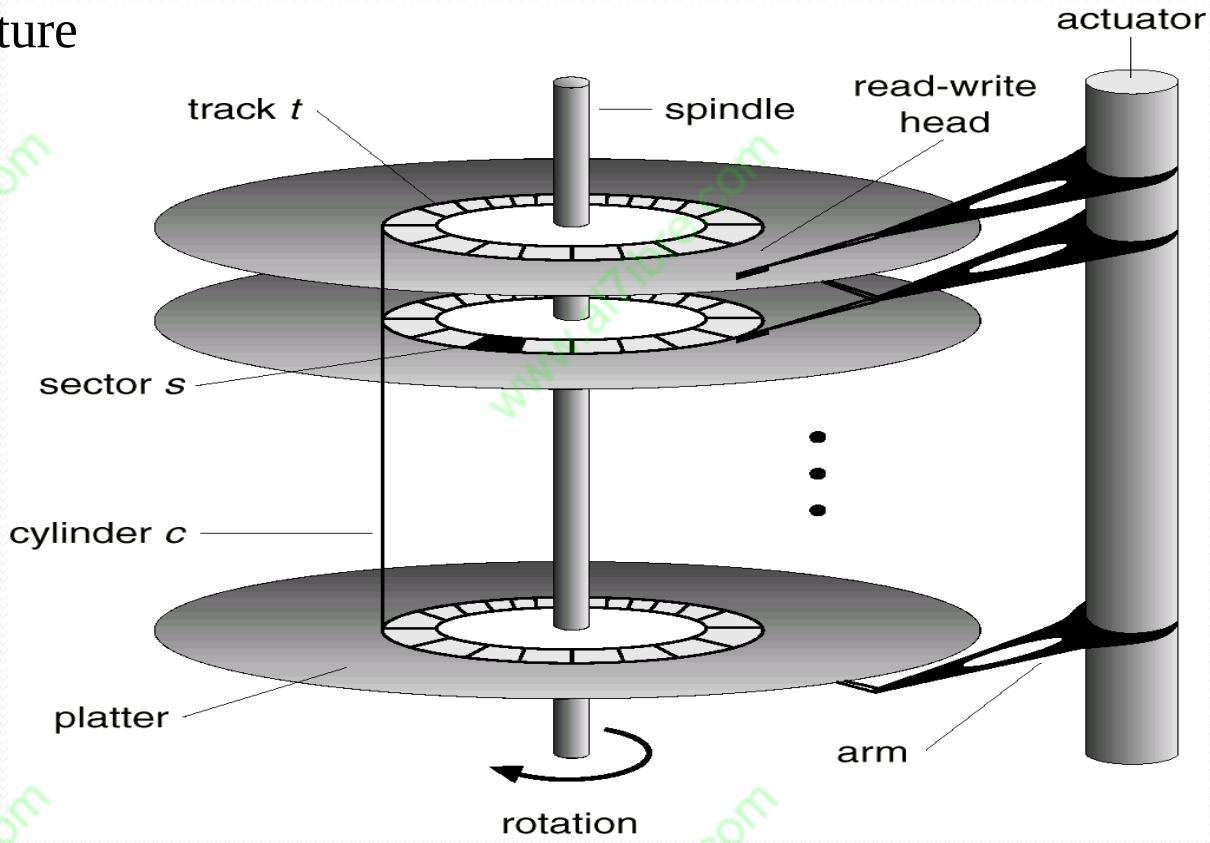
Structure de mémoire de masse (disques magnétiques)

- Plats rigides couverts de matériaux d ’enregistrement magnétique
  - surface du disque divisée en pistes (tracks) qui sont divisées en secteurs
  - le contrôleur disque détermine l` interaction logique entre l ’unité et l ’ordinateur

www.al7ibre.com

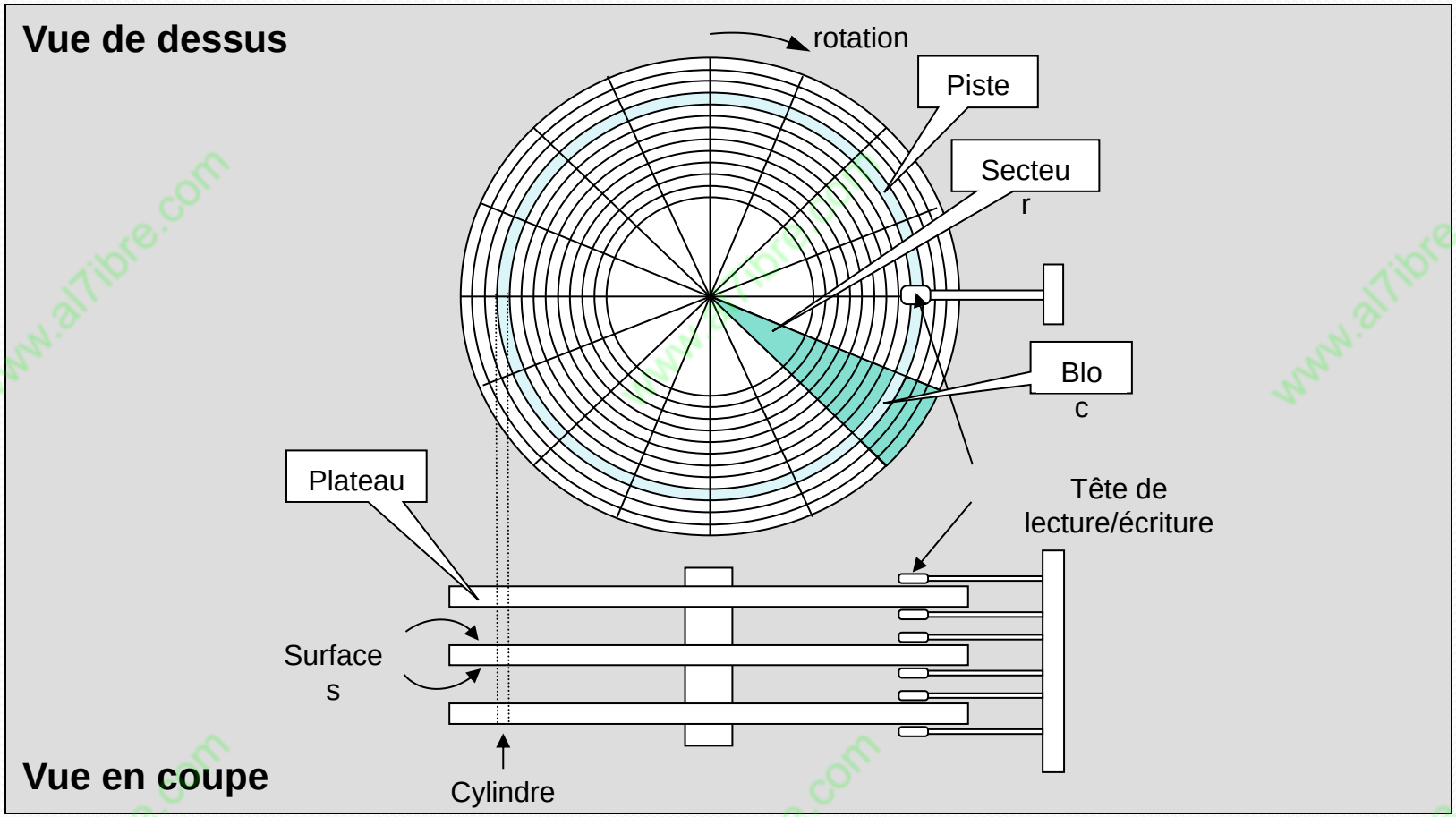
Nomenclature

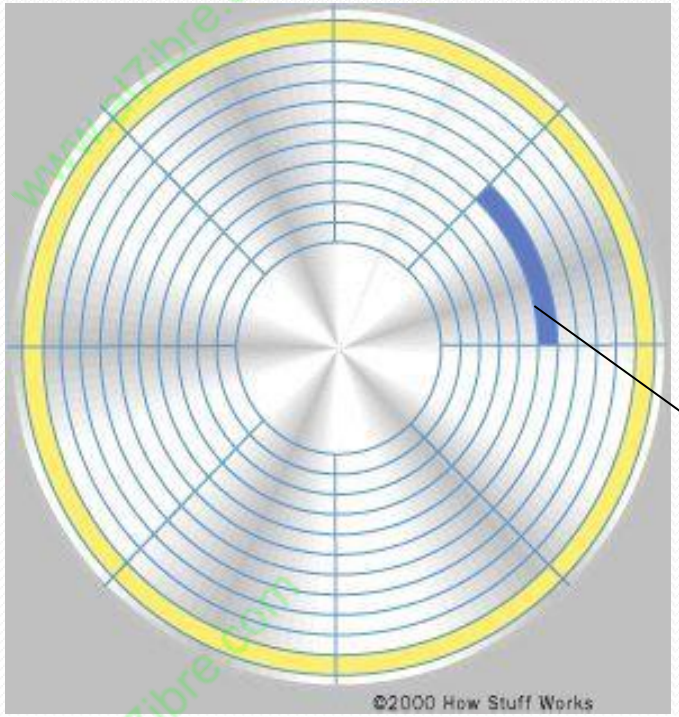
cylindre: l'ensemble de pistes qui se trouvent dans la même position du bras de lecture/écriture



www.al7ibre.com

Vue schématique d'un disque dur





Cylindres – Secteurs - Clusters

Cylindre: un tour de disque

Secteur: Un subdivision d'un cylindre (512 Kilooctect)

Cluster: Un groupement de secteurs

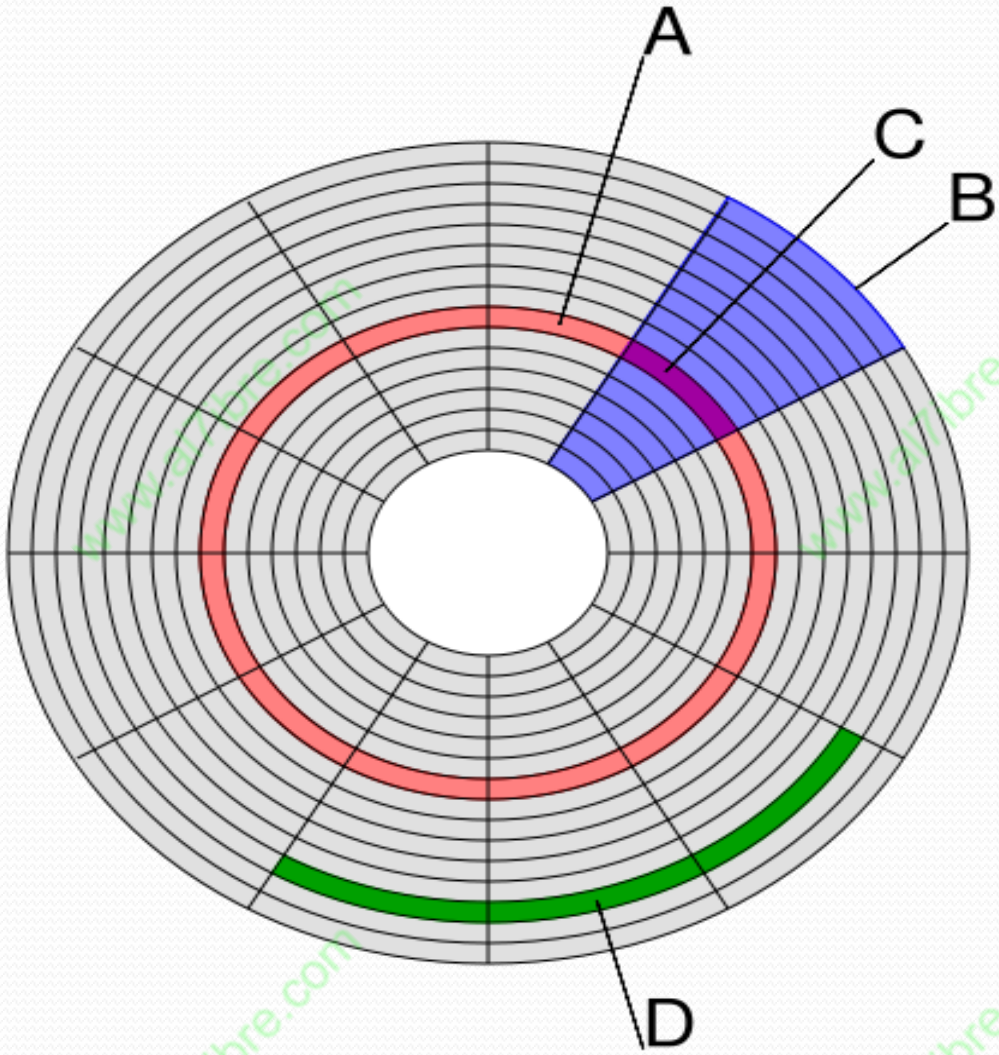
Low Level Format = Division d'un disque en secteurs

Adresse	Les Données	CRC
Cylindre tête Secteur	Structure, programme ou données	Correction d'erreur
Low level format	Application ou système	A l'écriture

Support physique de codage de l'information

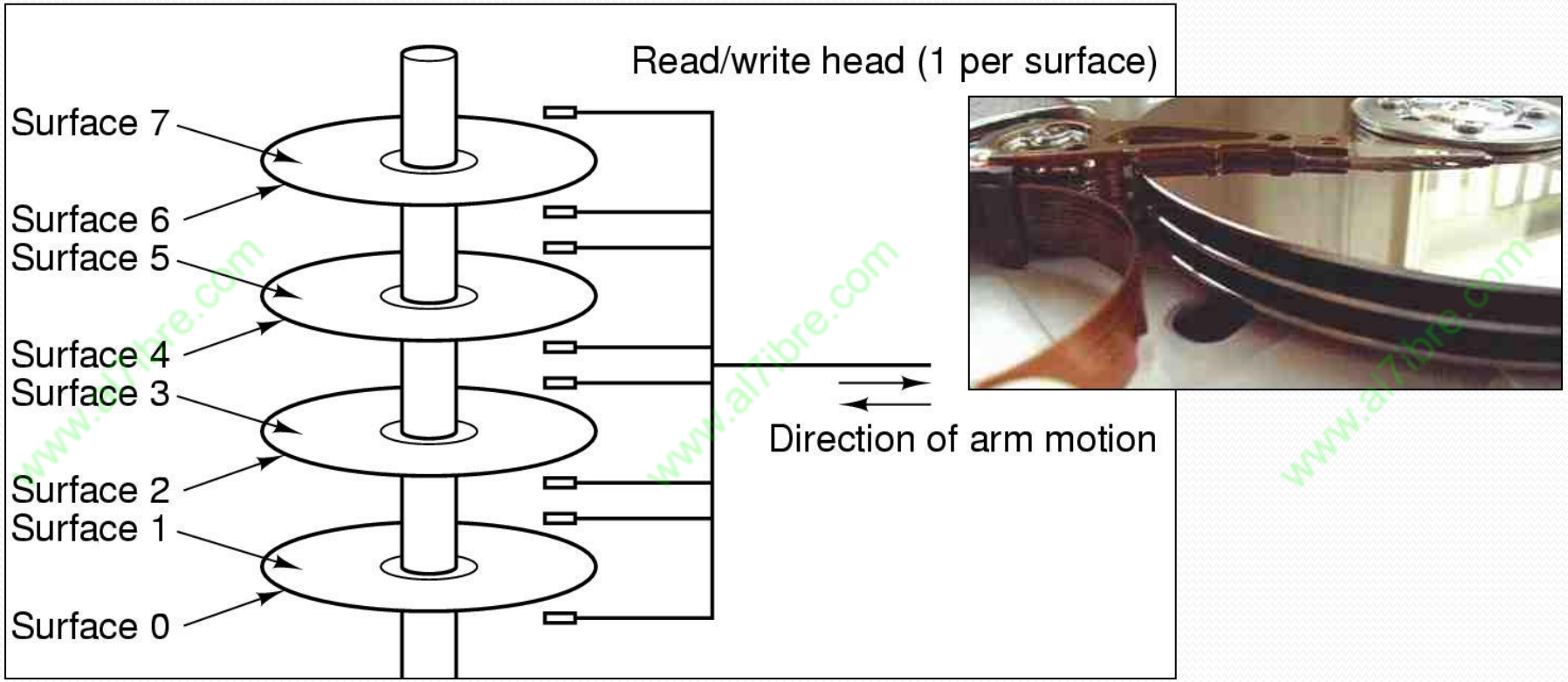
- Disque dur

(A) Piste
(B) Secteur géométrique
(C) secteur d'une piste
(D) cluster



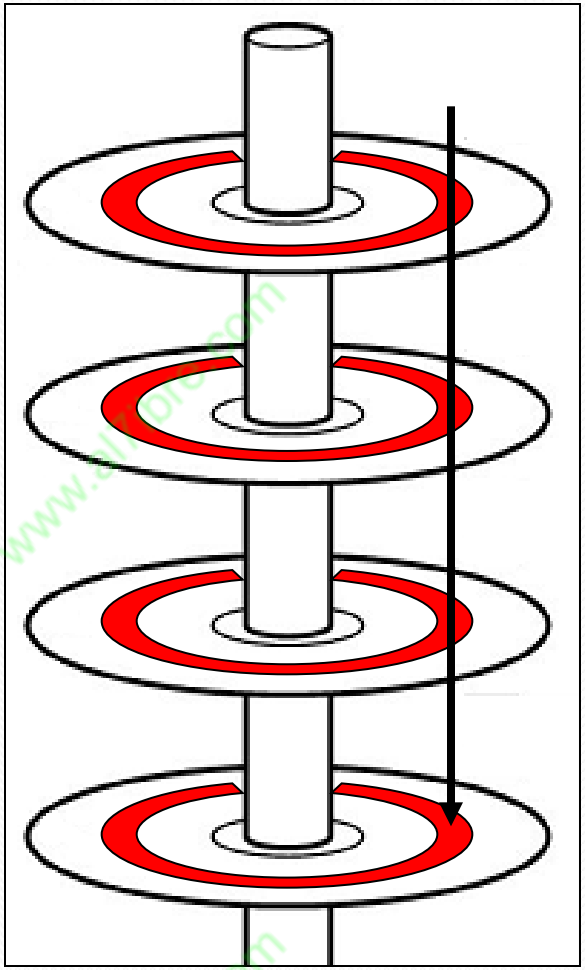
www.al7ibre.com

Revue des disques magnétiques



- Les disques sont organisés en cylindres, pistes et secteurs

Revue des disques magnétiques



- Toutes les pistes pour une position donnée du bras forment un cylindre.
  - Donc le nombre de cylindre est égale au nombre de piste par côté de plateau
  - La location sur un disque est spécifié par (cylindre, tête, secteur) mais en erreur par: (cylindre, piste, secteur)

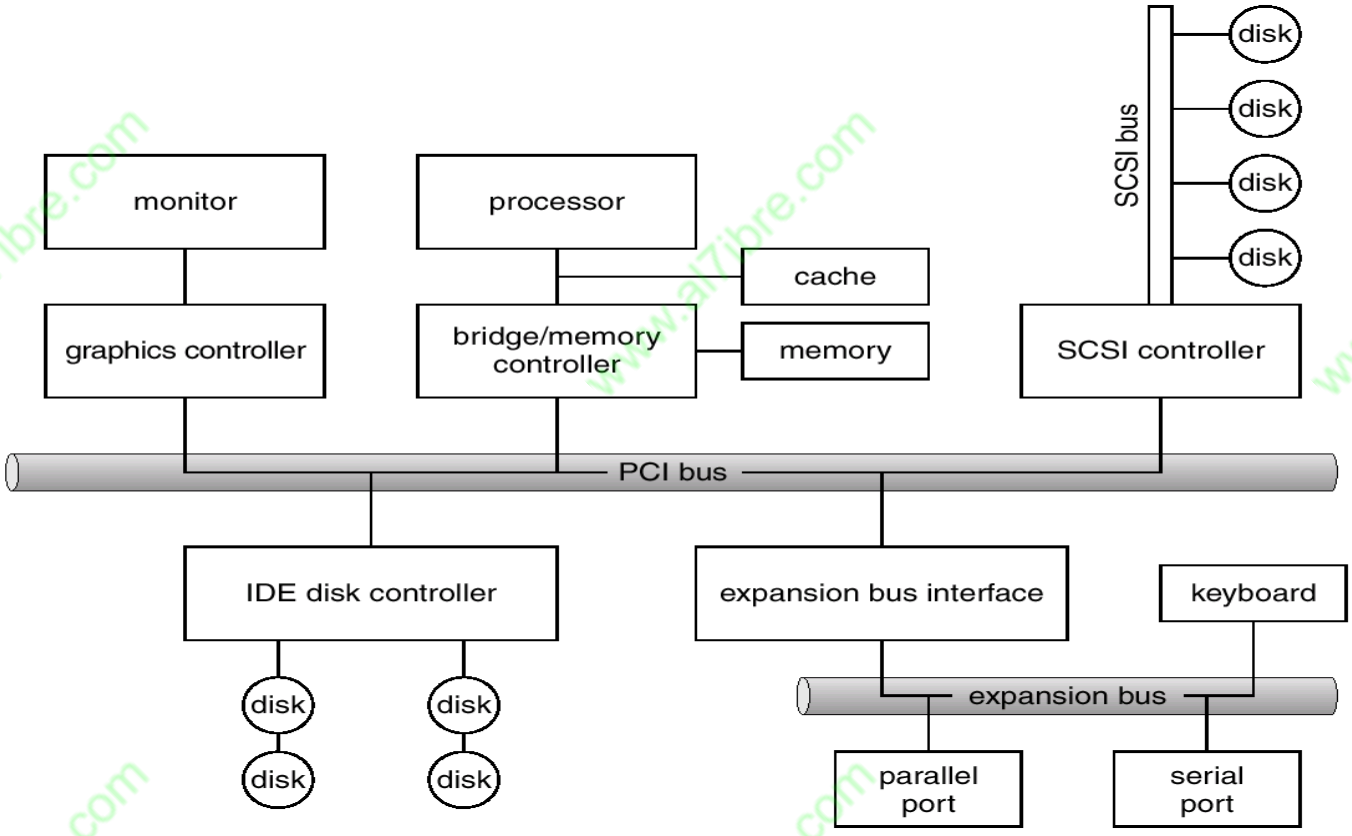
www.al7ibre.com

Sous-système E/S du noyau

- Fonctionnalités:
  - Mise en tampon
  - Mise en cache
  - Mise en attente et réservation de périphérique
  - Gestion des erreurs

www.al7ibre.com

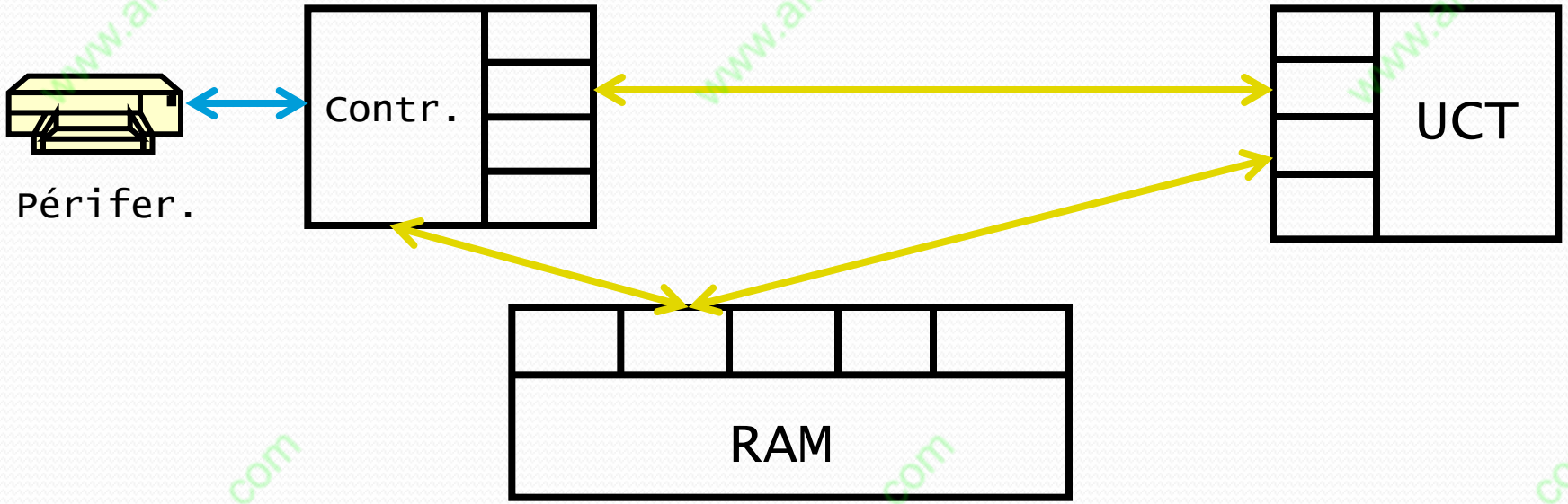
Structure typique de bus PC



PCI: Peripheral Component Interconnect

Communication entre UCT et contrôleurs périphériques

- Deux techniques de base:
  - UCT et contrôleurs communiquent directement par des registres
  - UCT et contrôleurs communiquent par des zones de mémoire centrale
  - Combinaisons de ces deux techniques



www.al7ibre.com

Accès direct en mémoire (DMA)

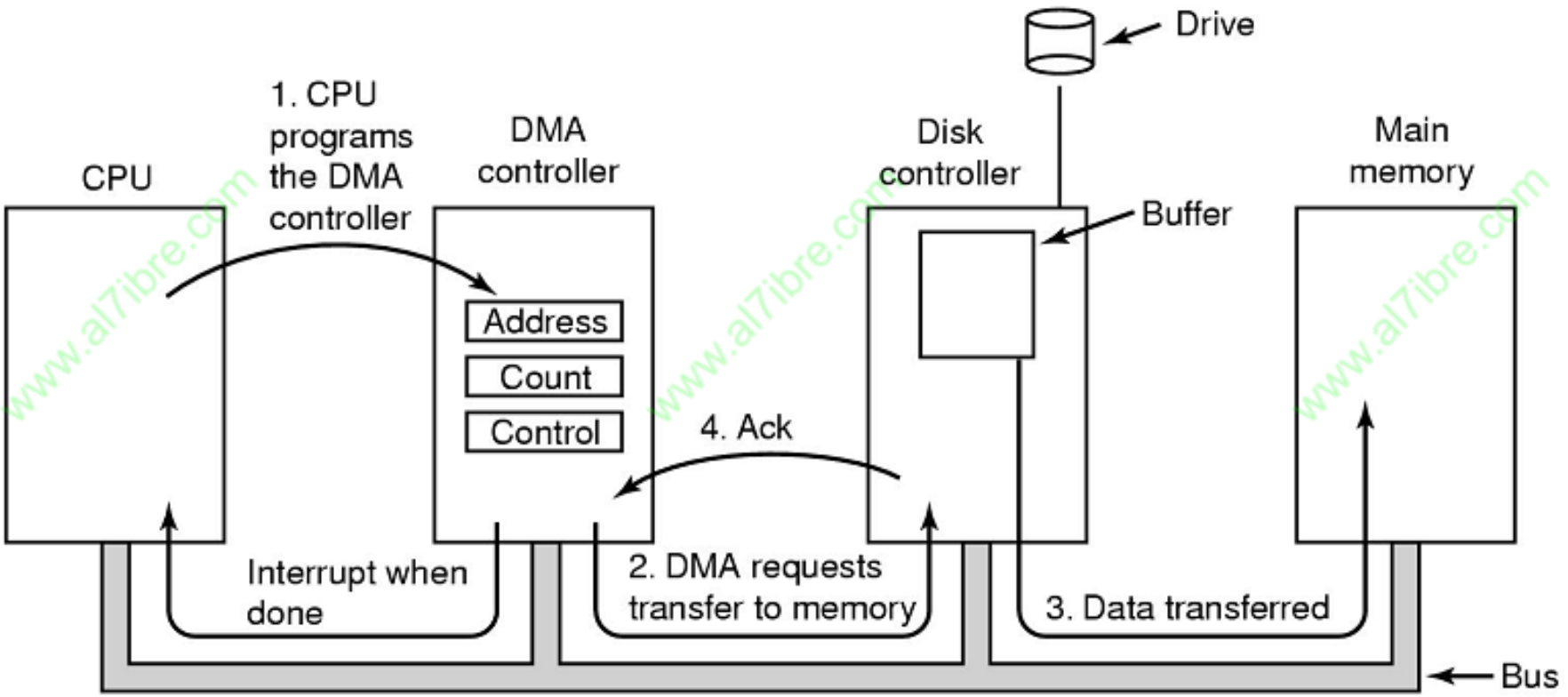
- Dans les systèmes sans DMA, l'UCT est impliquée dans le transfert de chaque octet
- DMA est utile pour exclure l'implication de l'UCT surtout pour des E/S volumineuses
- Demande un contrôleur spécial a accès direct à la mémoire centrale

www.al7ibre.com

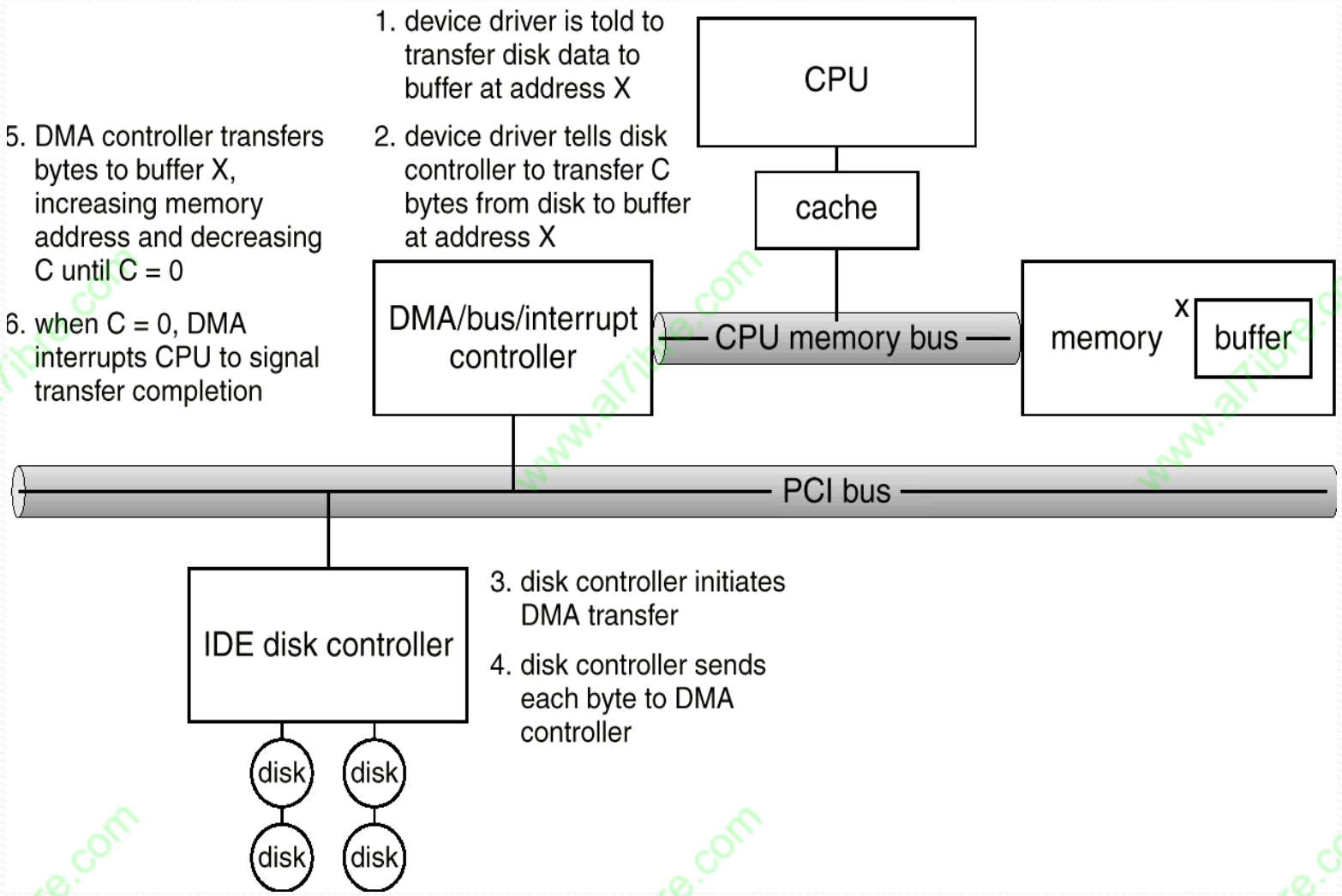
Accès directe à la mémoire (DMA)

- DMA est utilisé pour libérer le CPU d'avoir à déplacer des octets du périphérique vers la mémoire
  - Cela demande une autre pièce de matériel appelé un contrôleur DMA
  - Le SE/CPU charge les registres du contrôleur DMA avec l'information nécessaire pour l'instruire de quel périphérique prendre/passer les données, où les mettre en mémoire et combien d'octets doivent être écrit/lu

www.al7ibre.com


Drive
1. CPU programs the DMA controller
DMA controller
Disk controller
Main memory
CPU
Buffer
Address
Count
Control
4. Ack
Interrupt when done
2. DMA requests transfer to memory
3. Data transferred
Bus

DMA: six étapes



DMA: six étapes

- 1- CPU demande au pilote du périphérique (disque) (software) de transférer les données du disque au buffer à l'adresse x
- 2 - Le pilote du disque demande au contrôleur du disque (hardware) de transférer c octets du disque vers le buffer à l'adresse x
- 3 - Le contrôleur du disque initie le transfert DMA
- 4 - Le contrôleur du disque envoie chaque octet au contrôleur du DMA
- 5 - Le contrôleur DMA transfert les octets au buffer x en augmentant l'adresse x et décrémentant le compteur c
- 6 - Lorsque c=0 DMA envoie une interruption pour signaler la fin du transfert

www.al7ibre.com

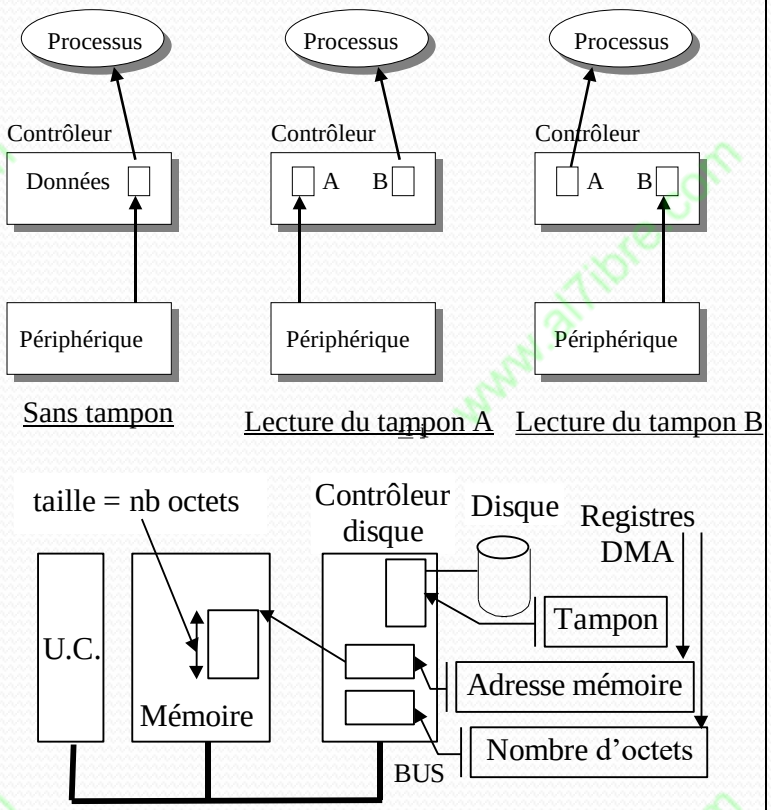
Tampons de disques

- Les disques ont besoin de tampons pour deux raisons principales:
  - Tamponner les données qui arrive plus vite que l'on peut les transférer au système d'exploitation et vice-versa
  - Lecture avancé de données qui n'ont pas encore étés demandées, mais qu'il le peuvent sous peu (données qui suivent la demande précédente)

www.al7ibre.com

Mise en tampon

- Principes.
  - Simultanéité des opérations d'entrées et de sorties avec les opérations de calcul.
  - Le contrôleur de périphérique inclue plusieurs registres de données.
  - Pendant que l'UCT accède à un registre, le contôleur peut accéder à un autre registre.



27

Mise en tampon

- Double tamponnage:
  - P.ex. en sortie: un processus écrit le prochain enregistrement sur un tampon en mémoire tant que l'enregistrement précédent est en train d'être écrit
  - Permet superposition traitement E/S

www.al7ibre.com

Mise en cache

- Quelques éléments couramment utilisés d'une mémoire secondaire sont gardés en mémoire centrale
- Donc quand un processus exécute une E/S, celle-ci pourrait ne pas être une E/S réelle:
  - Elle pourrait être un transfert en mémoire, une simple mise à jour d'un pointeur, etc.

www.al7ibre.com

Traitement des erreurs

- Il y a deux classes d'erreurs dans cette couche:
  - ***Erreurs de programmation*** – le processus de l'utilisateur demande l'impossible tel que d'écrire à une souris, lire d'une imprimante, ou accéder à un fichier qui n'a pas été ouvert
  - ***Erreurs d'E/S*** – une tentative a été faite pour écrire au disque mais l'opération a échoué au niveau physique. Si le pilote ne peut pas traiter le problème (par exemple en essayant d'écrire encore), il est passé à la couche supérieure
- Cette couche est responsable pour **collationner** les erreurs qui peuvent se produire et de les **rapporter** à l'utilisateur d'une façon consistante quand cela est requis

Gestion des erreurs

- Exemples d'erreurs à être traités par le SE:
  - Erreurs de lecture/écriture, protection, périph non-disponible
- Les erreurs retournent un code 'raison'
- Traitement différent dans les différents cas…

www.al7ibre.com

Gestion de requêtes E/S

- P. ex. lecture d'un fichier de disque
  - Déterminer où se trouve le fichier
  - Traduire le nom du fichier en nom de périphérique et location dans périphérique
  - Lire physiquement le fichier dans le tampon
  - Rendre les données disponibles au processus
  - Retourner au processus

www.al7ibre.com

2- Systèmes de fichiers

- Systèmes fichiers
- Méthodes d'accès
- Méthodes d'allocation
- Gestion de l'espace libre

www.al7ibre.com

Que c’est qu’un fichier

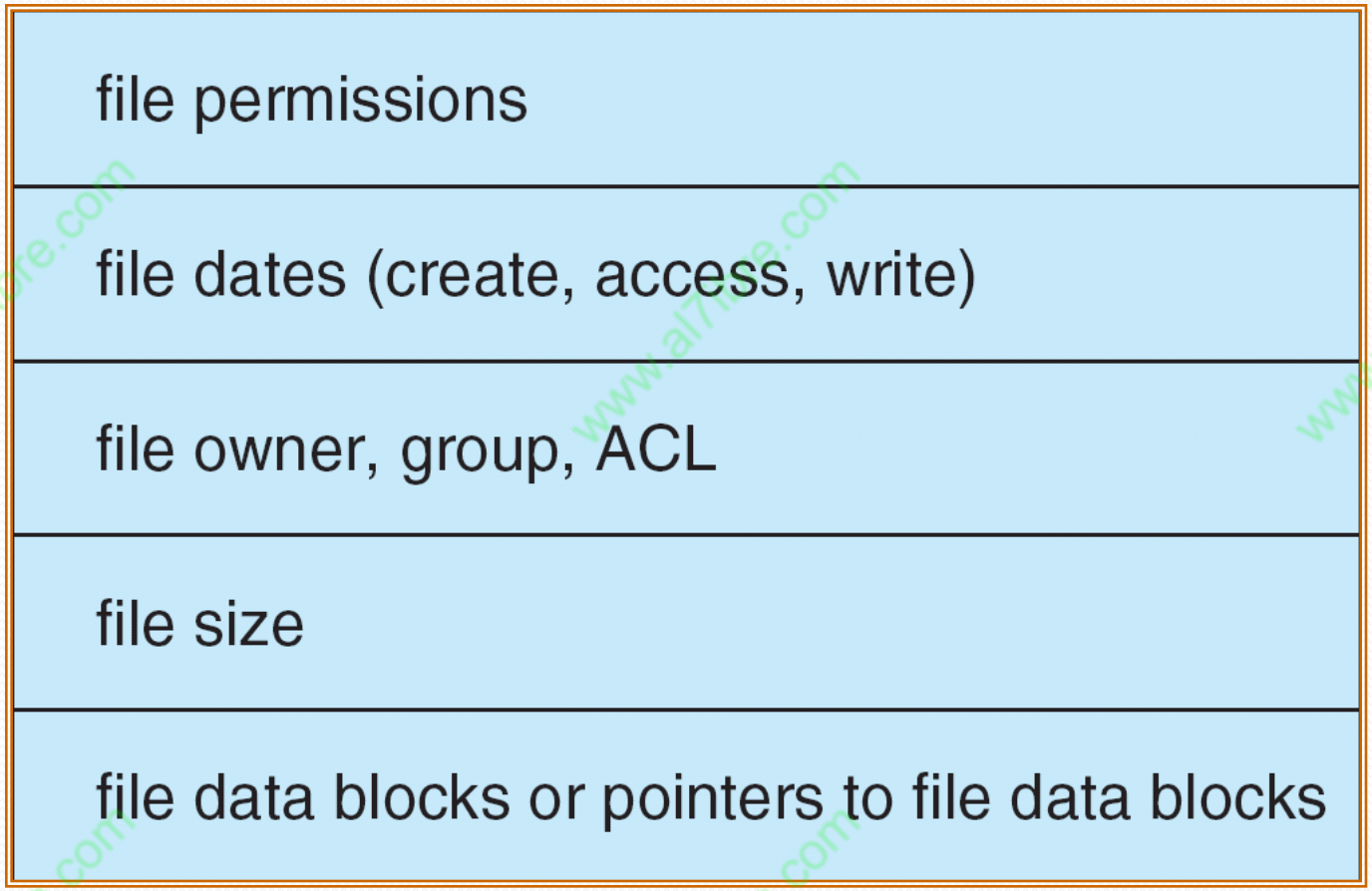
- Collection nommée d’informations apparentées, enregistrée sur un stockage secondaire
  - Nature permanente
- Les données qui se trouvent sur un stockage secondaires doivent être dans un fichier
- Différents types:
  - Données (binaire, numérique, caractères….)
  - Programmes

www.al7ibre.com

Attributs d'un fichier

- Constituent les propriétés du fichiers et sont stockés dans un fichier spécial appelé répertoire (directory). Exemples d'attributs:
  - Nom:
    - pour permet aux personnes d'accéder au fichier
  - Identificateur:
    - Un nombre permettant au SE d'identifier le fichier
  - Type:
    - Ex: binaire, ou texte; lorsque le SE supporte cela
  - Position:
    - Indique le disque et l'adresse du fichier sur disque
  - Taille:
    - En bytes ou en blocs
  - Protection:
    - Détermine qui peut écrire, lire, exécuter…
  - Date:
    - pour la dernière modification, ou dernière utilisation
  - Autres…

Un “File Control Block” typique



www.al7ibre.com

Opérations sur les fichiers: de base

- Création
- Écriture
  - Pointeur d'écriture qui donne la position d'écriture
- Lecture
  - Pointeur de lecture
- Positionnement dans un fichier (temps de recherche)
- Suppression d'un fichier
  - Libération d'espace
- Troncature: remise de la taille à zéro tout en conservant les attributs

www.al7ibre.com

Autres opérations

- Ajout d'infos
- Rénommage
- Copie
  - peut être faite par rénommage: deux noms pour un seul fichier
- Ouverture d'un fichier: le fichier devient associé à un processus qui en garde les attributs, position, etc.
- Fermeture
- Ouverture et fermeture peuvent être explicites (ops *open*, *close*)
- ou implicites

www.al7ibre.com

Méthodes d’allocation

www.al7ibre.com

Structure physique des fichiers

- La mémoire secondaire est subdivisée en blocs et chaque opération d'E /S s'effectue en unités de blocs
  - Les blocs ruban sont de longueur variable, mais les blocs disque sont de longueur fixe
  - Sur disque, un bloc est constitué d'un multiple de secteurs contiguës (ex: 1, 2, ou 4)
    - la taille d'un secteur est habituellement 512 bytes
- Il faut donc insérer les enregistrements dans les blocs et les extraire par la suite
  - Simple lorsque chaque octet est un enregistrement par lui-même
  - Plus complexe lorsque les enregistrements possèdent une structure (ex: « main-frame IBM »)
- Les fichiers sont alloués en unité de blocs. Le dernier bloc est donc rarement rempli de données
  - Fragmentation interne

www.al7ibre.com

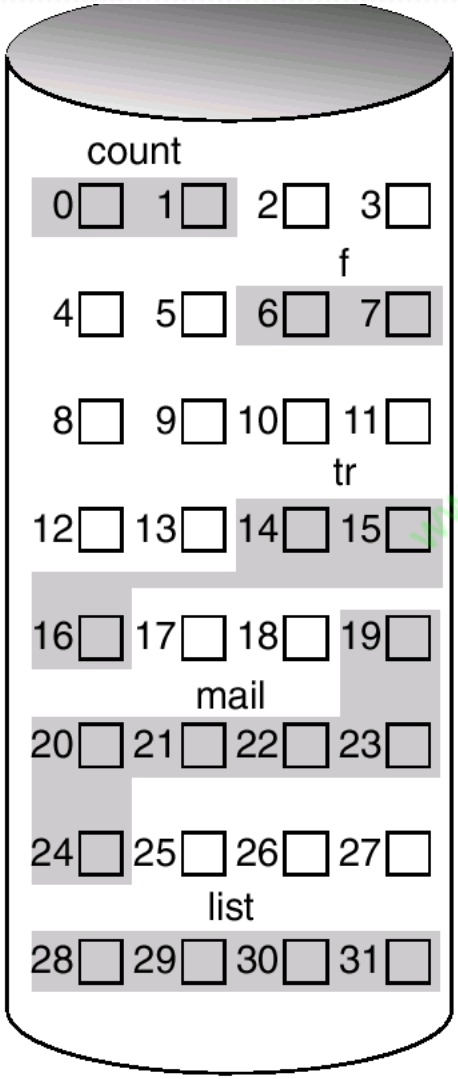
Trois méthodes d’allocation de fichiers

- Allocation contiguë
- Allocation enchaînée
- Allocation indexée

www.al7ibre.com

Allocation contiguë sur disque

répertoire



directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Allocation contiguë

- Chaque fichier occupe un ensemble de blocs contigu sur disque
- Simple: nous n'avons besoin que d'adresses de début et longueur
- Supporte tant l'accès séquentiel, que l'accès direct
- Moins pratique pour les autres méthodes

www.al7ibre.com

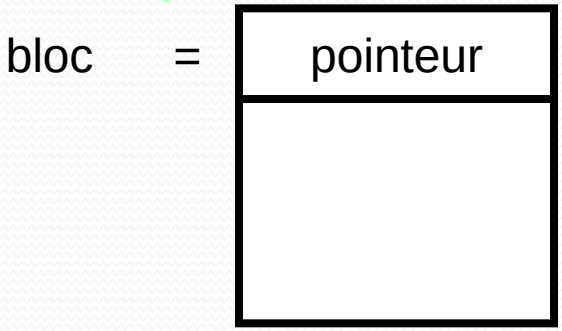
Allocation contiguë

- Application des problèmes et méthodes vus dans le chapitre de l'alloc de mémoire contiguë
- Les fichiers ne peuvent pas grandir
- Impossible d'ajouter au milieu
- Exécution périodique d'une compression (compaction) pour récupérer l'espace libre

www.al7ibre.com

Allocation enchaînée

- Le répertoire contient l ’adresse du premier et dernier bloc, possibl. le nombre de blocs
- Utilisé par MS-DOS et OS2.
- Chaque bloc contient un pointeur à l’adresse du prochain bloc:



www.al7ibre.com

Allocation enchaînée

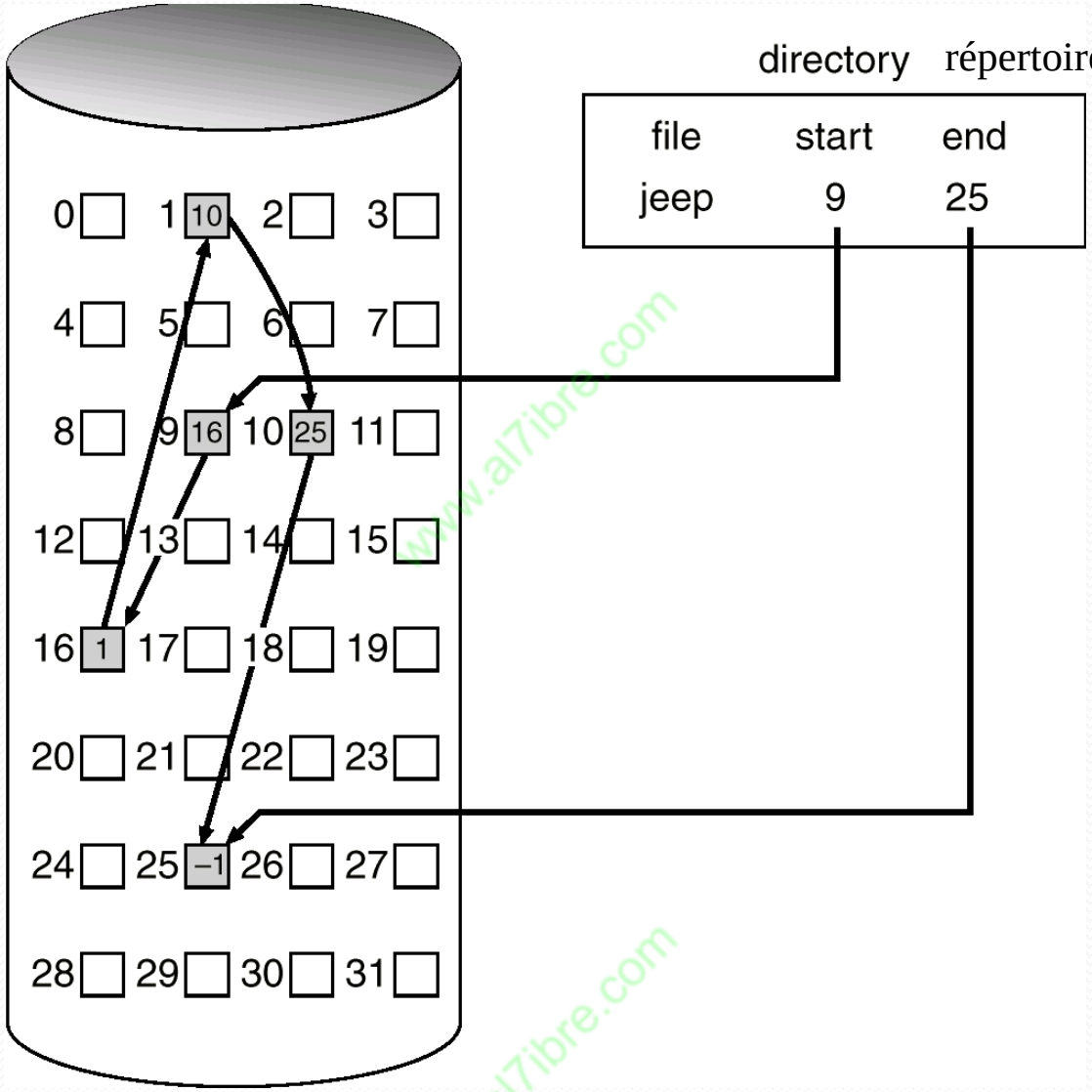
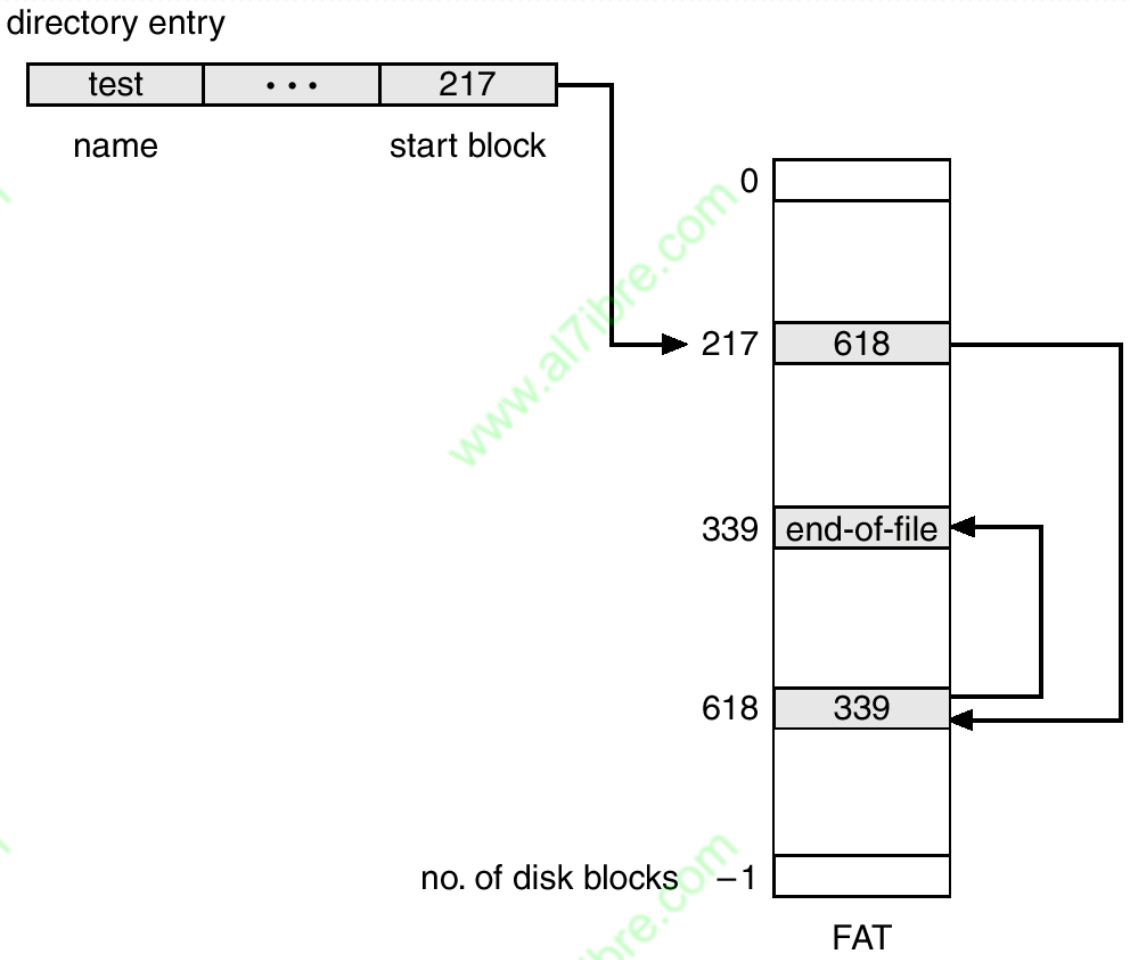


Tableau d'allocation de fichiers (FAT)



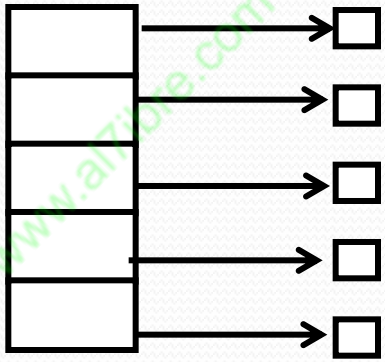
Avantages - désavantages

- Pas de fragmentation externe - allocation de mémoire simple, pas besoin de compression
- L ’accès à l ’intérieur d ’un fichier ne peut être que séquentiel
  - Pas façon de trouver directement le 4ème enregistrement…
- L ’intégrité des pointeurs est essentielle
- Les pointeurs gaspillent un peu d ’espace

www.al7ibre.com

Allocation indexée: semblable à la pagination

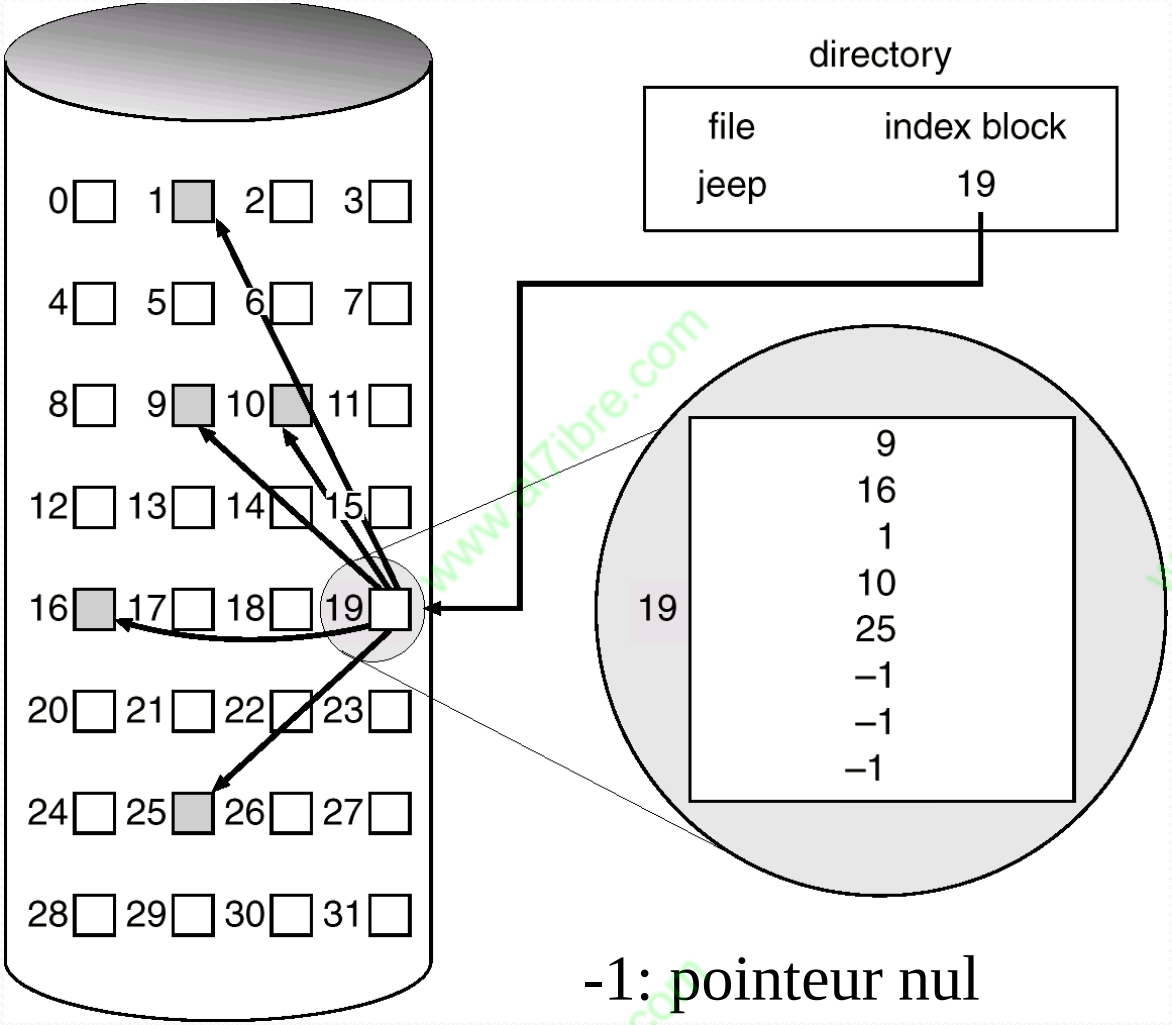
- Tous les pointeurs sont regroupés dans un tableau (index block)



index table

www.al7ibre.com

Allocation indexée



Allocation indexée

- À la création d ’un fichier, tous les pointeurs dans le tableau sont *nil (-1)*
- Chaque fois qu’un nouveau bloc doit être alloué, on trouve de l ’espace disponible et on ajoute un pointeur avec son adresse

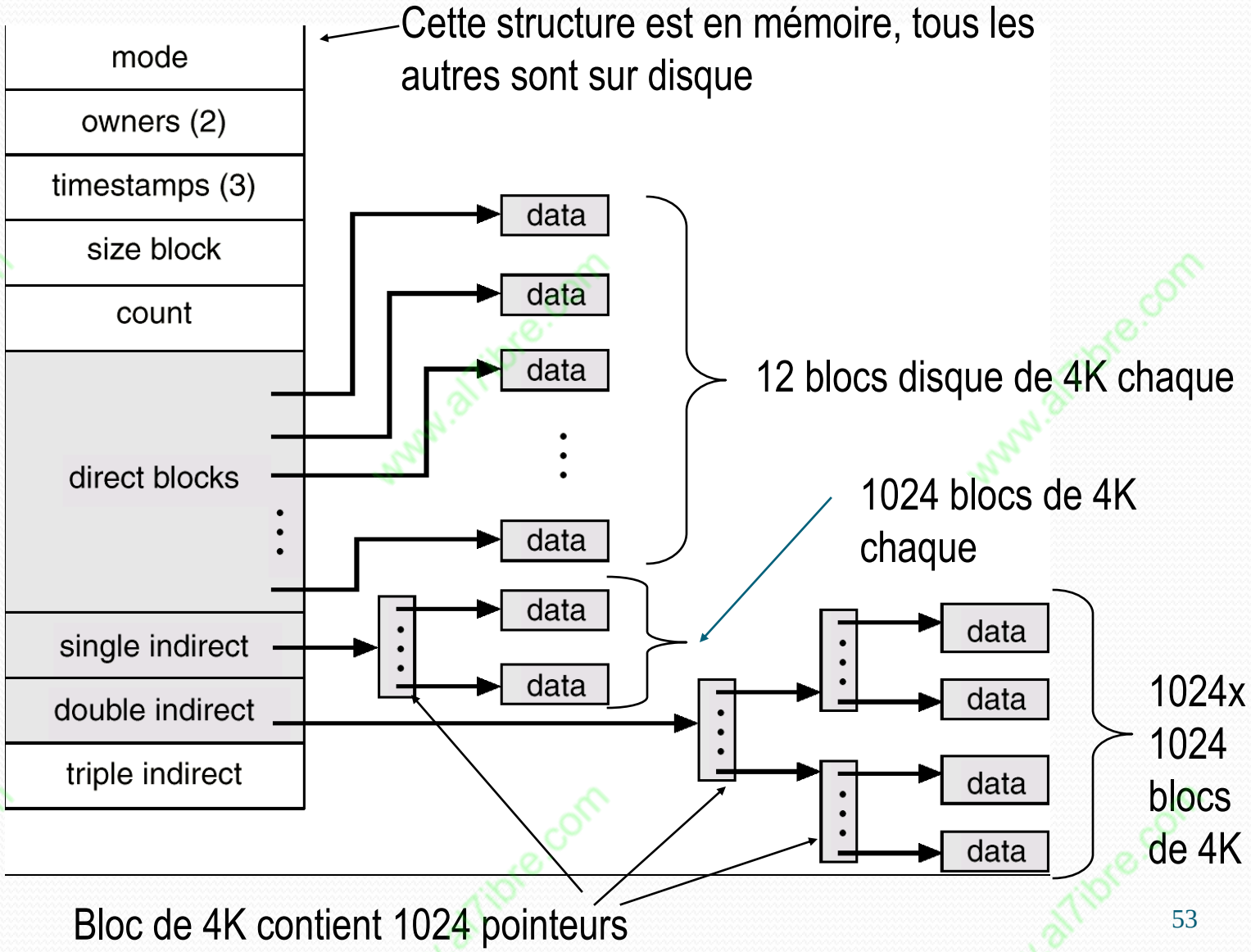
www.al7ibre.com

Allocation indexée

- Pas de fragmentation externe, mais les index prennent de l'espace
- Permet accès direct (aléatoire)
- Taille de fichiers limitée par la taille de l'index block
  - Mais nous pouvons avoir plusieurs niveaux d'index: Unix
- Index block peut utiliser beaucoup de mém.

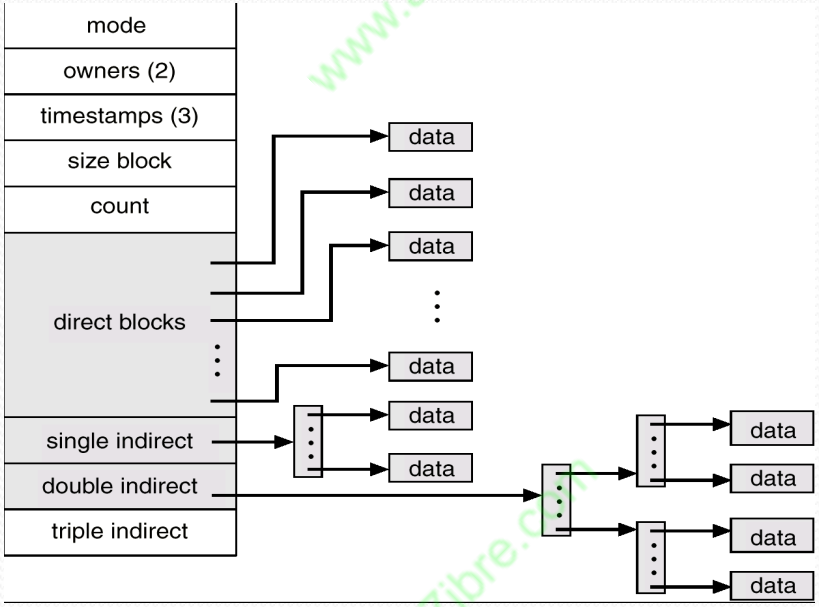
www.al7ibre.com

UNIX BSD: indexé à niveaux



UNIX BSD

- Les premiers blocs d'un fichier sont accessibles directement
- Si le fichier contient des blocs additionnels, les premiers sont accessibles à travers un niveau d'indices
- Les suivants sont accessibles à travers 2 niveaux d'indices, etc.
- Donc le plus loin du début un enregistrement se trouve, le plus indirect est son accès
- Permet accès rapide à petits fichiers, et au début de tous les fich.
- Permet l'accès à des grands fichier avec un petit répertoire en mémoire



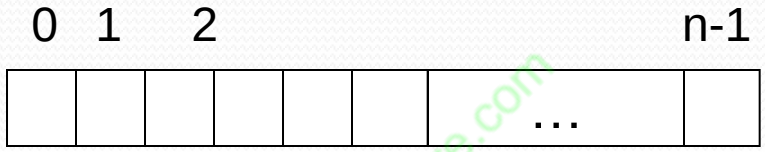
Gestion de l’espace libre

www.al7ibre.com

Gestion d'espace libre

Solution 1: vecteur de bits (*n* blocs)

(solution Macintosh, Windows 2000)



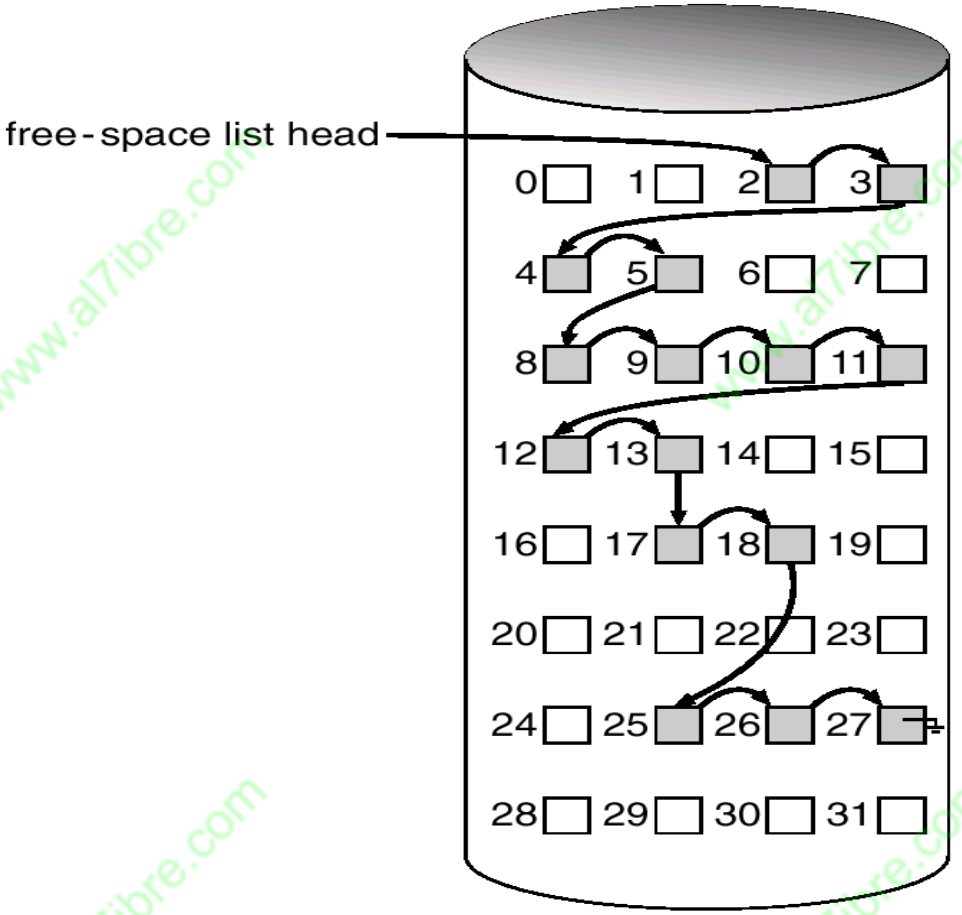
$$bit[i] = \begin{cases} 0 \Rightarrow block[i] \text{ libre} \\ 1 \Rightarrow block[i] \text{ occupé} \end{cases}$$

- Exemple d'un vecteur de bits où les blocs 3, 4, 5, 9, 10, 15, 16 sont occupés:
  - 0001110001100011…
- L'adresse du premier bloc libre peut être trouvée par un simple calcul

Gestion d'espace libre

Solution 2: Liste liée de mémoire libre (MS-DOS, Windows 9x)

Tous les blocs de mémoire libre sont liés ensemble par des pointeurs



Comparaison

- Bitmap:
  - si la bitmap de toute la mémoire secondaire est gardée en mémoire principale, la méthode est rapide mais demande de l'espace de mémoire principale
  - si les bitmaps sont gardées en mémoire secondaire, temps de lecture de mémoire secondaire…
    - Elles pourraient être paginées, p.ex.
- Liste liée
  - Pour trouver plusieurs blocs de mémoire libre, plus. accès de disque pourraient être demandés
  - Pour augmenter l'efficacité, nous pouvons garder en mémoire centrale l 'adresse du 1er bloc libre

www.al7ibre.com

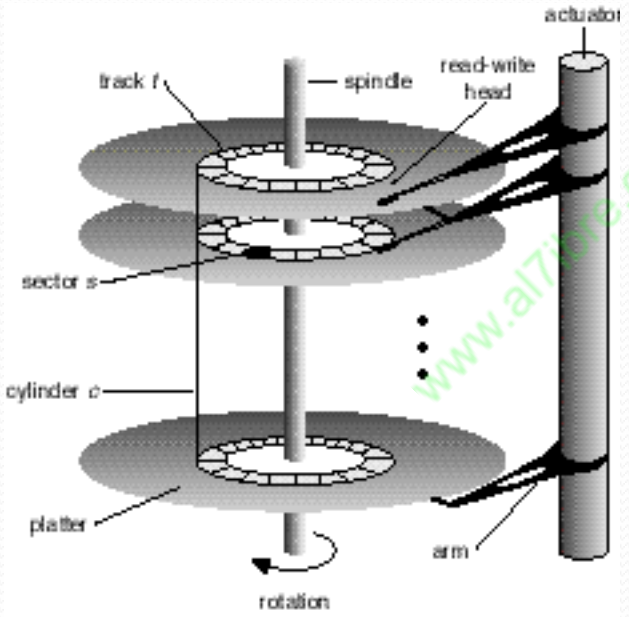
Ordonnancement disques

- Problème: utilisation optimale du matériel
- Réduction du temps total de lecture disque
  - étant donné une file de requêtes de lecture disque, dans quel ordre les exécuter?

www.al7ibre.com

Paramètres à prendre en considération

- Temps de positionnement (seek time):
  - le temps pris par l`unité disque pour se positionner sur le cylindre désiré
- Temps de latence de rotation
  - le temps pris par l 'unité de disque qui est sur le bon cylindre pour se positionner sur le secteur désirée
- Temps de lecture
  - temps nécessaire pour lire la piste
- Le temps de positionnement est normalement le plus important, donc il est celui que nous chercherons à minimiser



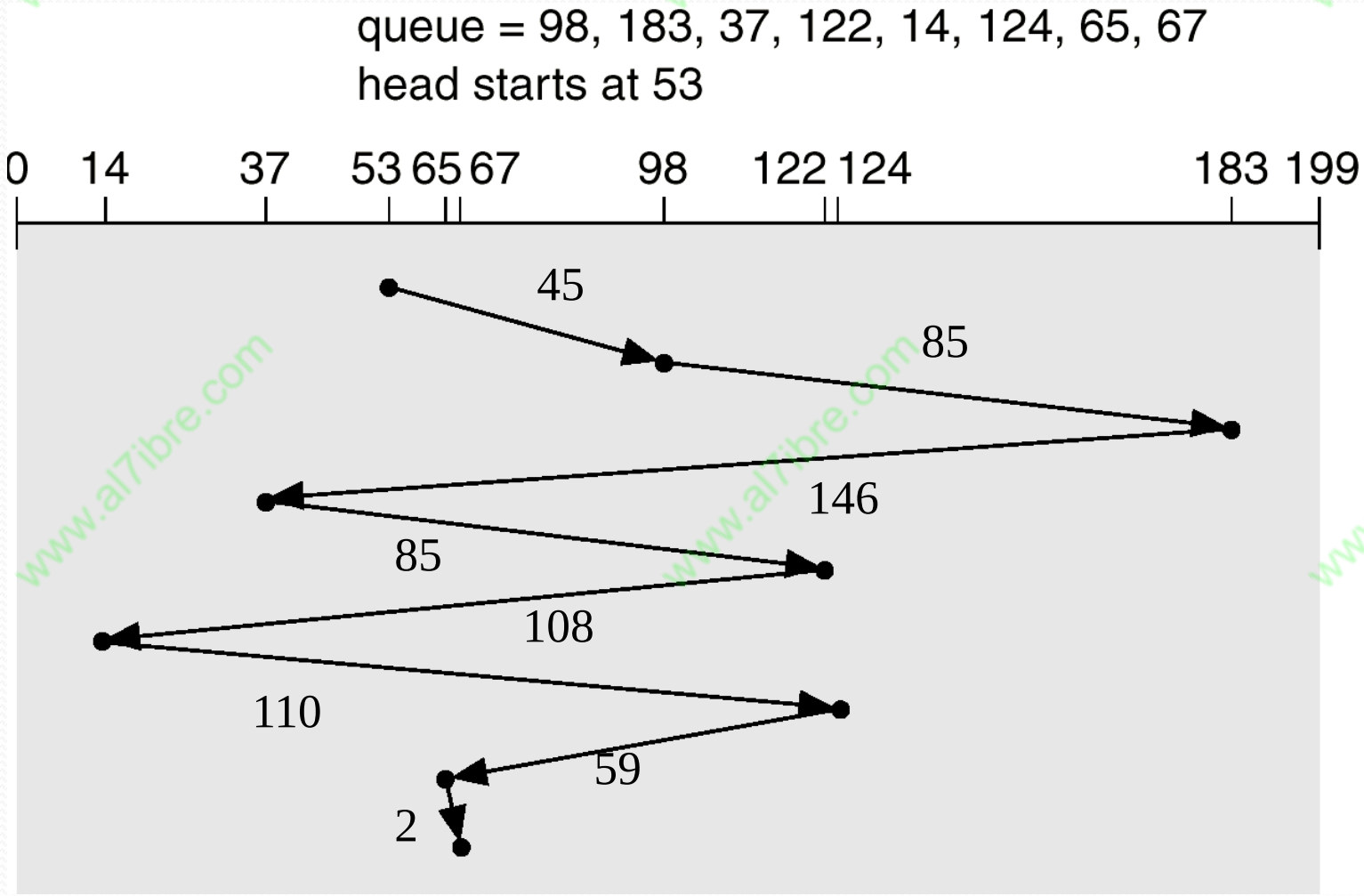
File d'attente disque

- Dans un système multiprogrammé avec mémoire virtuelle, il y aura normalement une file d'attente pour l 'unité disque
- Dans quel ordre choisir les requêtes d 'opérations disques de façon à minimiser les temps de recherche totaux
- Nous étudierons différents méthodes par rapport à une file d 'attente arbitraire:

98, 183, 37, 122, 14, 124, 65, 67

- Chaque chiffre est un numéro séquentiel de cylindre
- Il faut aussi prendre en considération le cylindre de départ: 53
- Dans quel ordre exécuter les requêtes de lecture de façon à minimiser les temps totaux de positionnement cylindre
- Hypothèse simpliste: un déplacement d`1 cylindre coûte 1 unité de temps

Premier entré, premier sorti: FIFO



axe de rotation

Mouvement total: 640 cylindres = (98-53) + (183-98)+...

En moyenne: 640/8 = 80

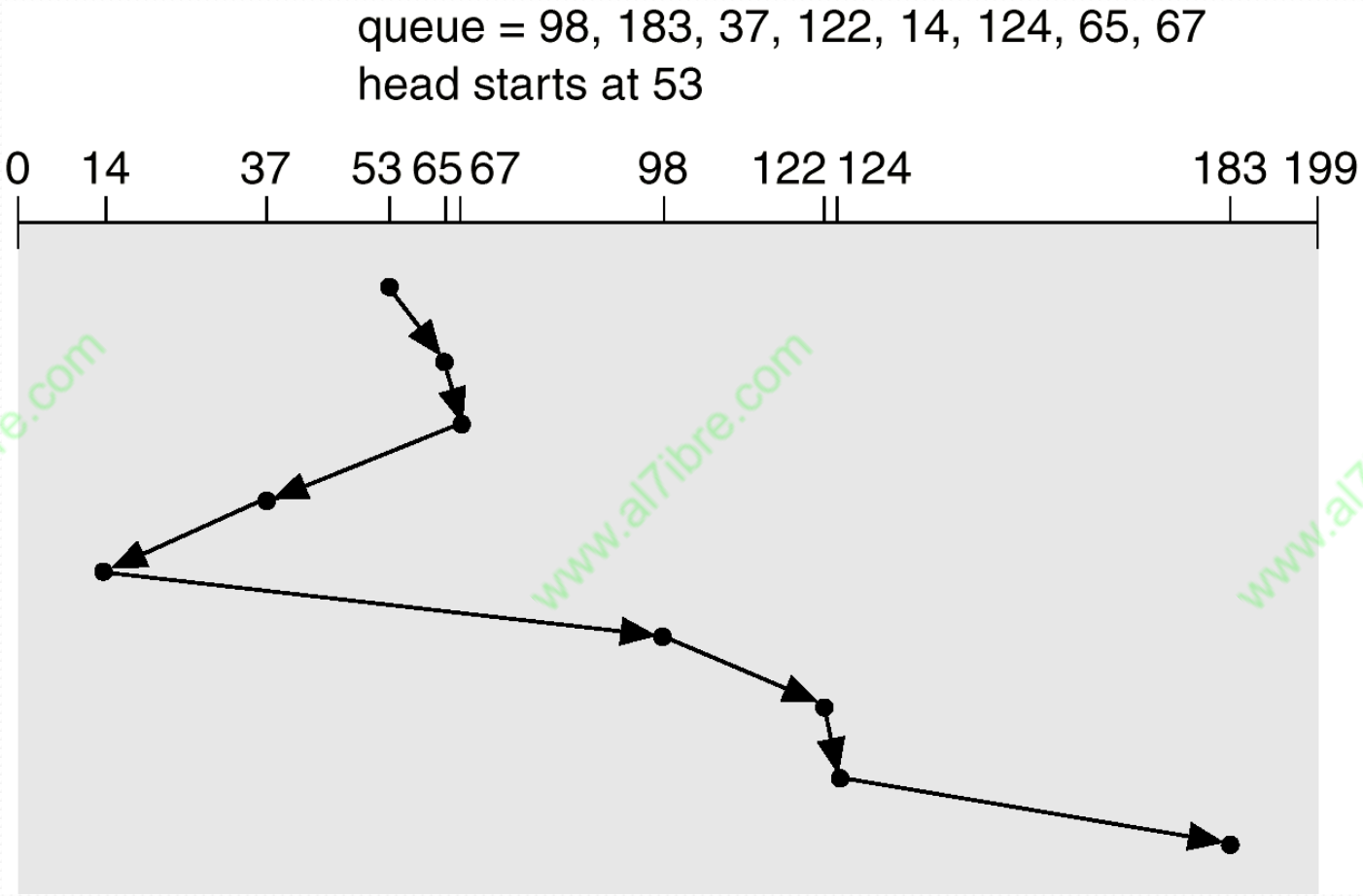
SSTF: Shortest Seek Time First

Plus Court Temps de Recherche (positionnement) d'abord (PCTR ou PCTP)

- À chaque moment, choisir la requête avec le temps de recherche le plus court à partir du cylindre courant
- Clairement meilleur que le précédent
- Mais pas nécessairement optimal! (v. manuel)
- Peut causer famine

www.al7ibre.com

SSTF: Plus court servi



Mouvement total: 183 cylindres (680 pour le précédent)

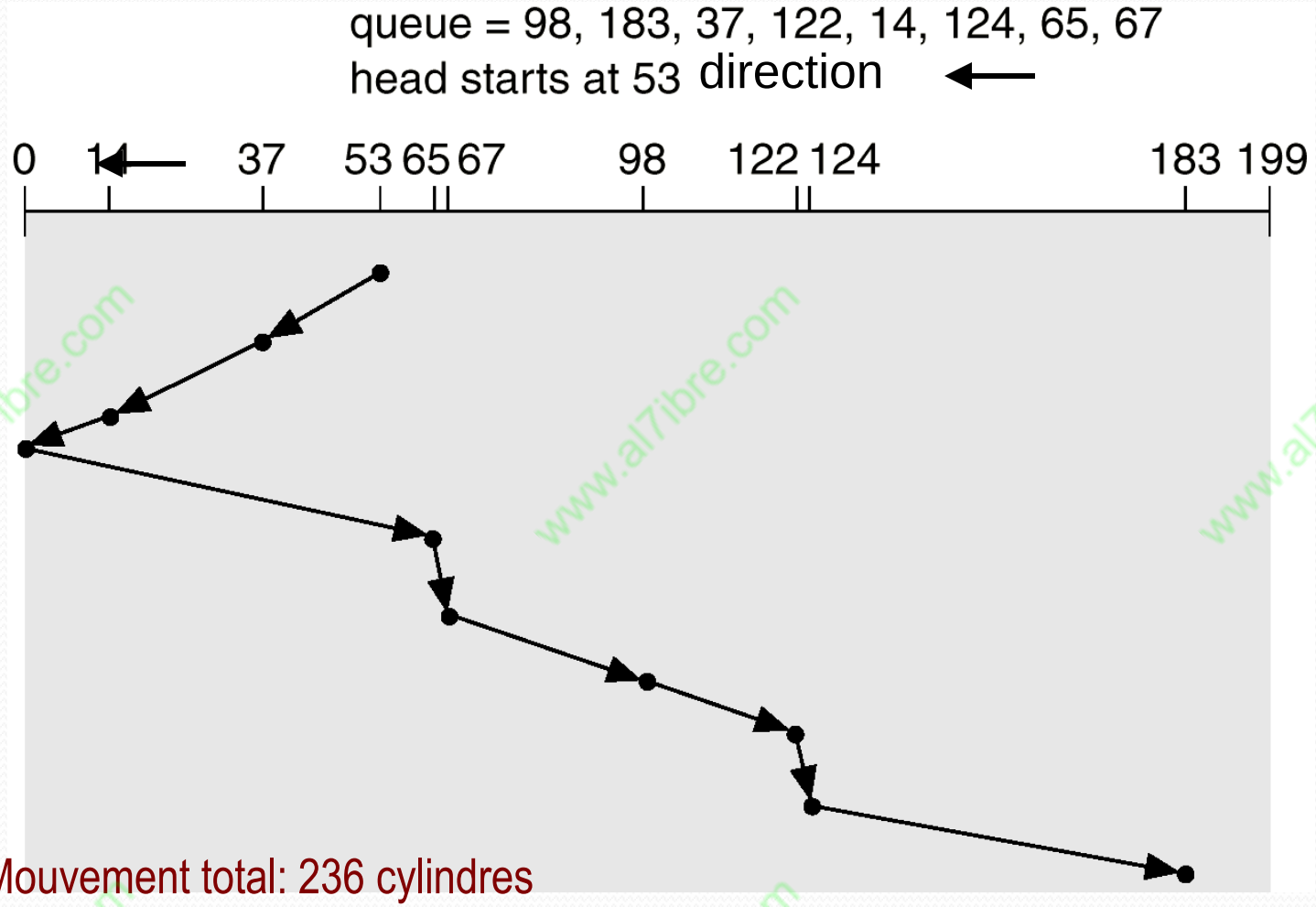
En moyenne: 183/8 = 22.8 (80 pour le précédent)

SCAN: l'algorithme du bus

- La tête balaye le disque dans une direction, puis dans la direction opposée, etc., en desservant les requêtes quand il passe sur le cylindre désiré
  - Pas de famine

www.al7ibre.com

SCAN: le bus



Mouvement total: 236 cylindres

En moyenne: 236/8= 29.5 comme pour SSTF)

Problèmes du SCAN

- Peu de travail à faire après le renversement de direction
- Les requêtes seront plus denses à l'autre extrémité
- Arrive inutilement jusqu'à 0

www.al7ibre.com

Look: l’algorithme de l’ascenseur

- La tête balaye le disque dans une direction, puis dans la direction opposée, etc., en desservant les requêtes quand il passe sur le cylindre désiré mais ne va pas jusqu'au bout du disque, elle rebrousse chemin lorsqu’il n’y a plus de piste à servir dans ce sens.

www.al7ibre.com

C-SCAN et C-LOOK

C-SCAN

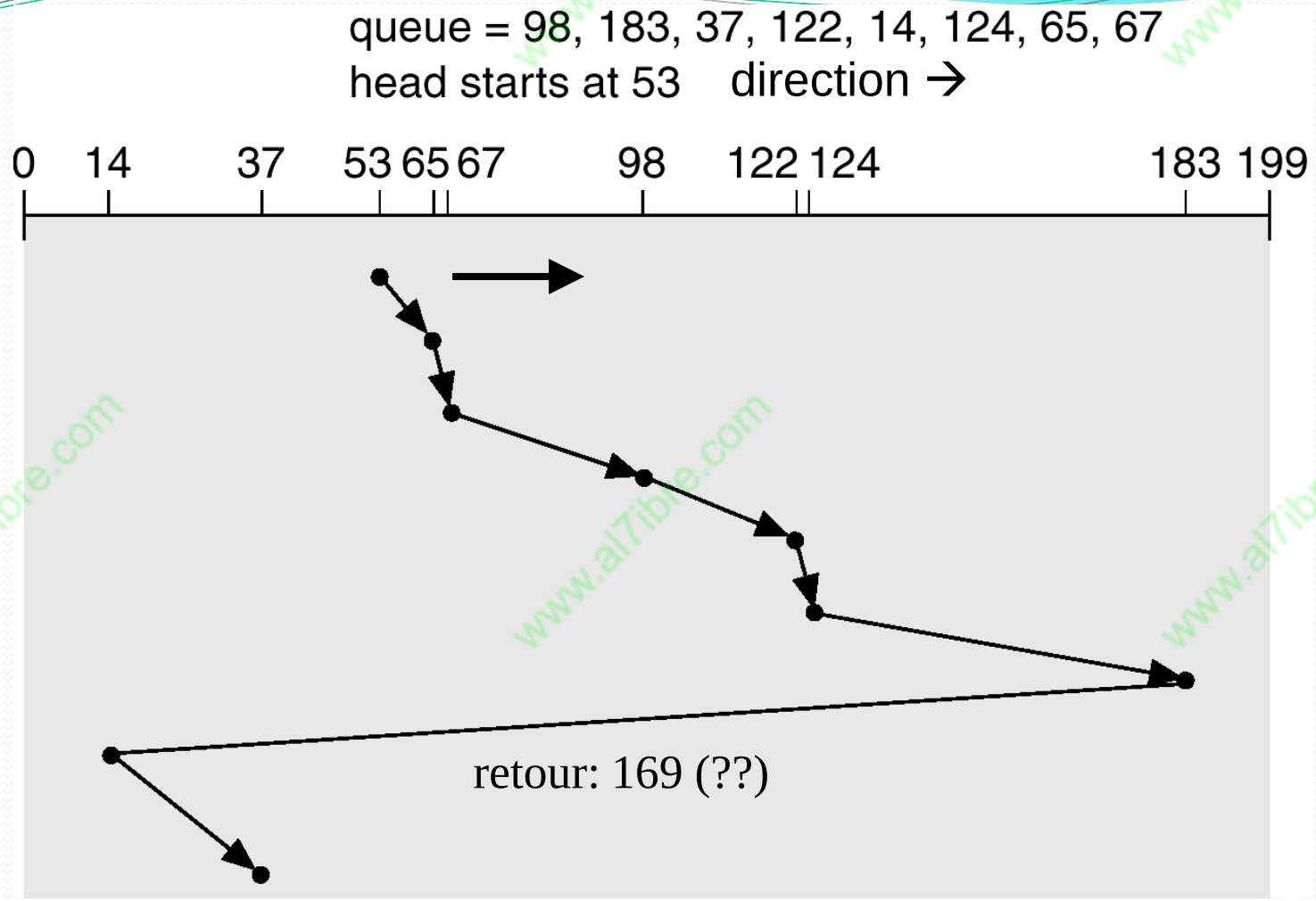
- Retour rapide au début (cylindre 0) du disque au lieu de renverser la direction
- Hypothèse: le mécanisme de retour est beaucoup plus rapide que le temps de visiter les cylindres

C-LOOK

- La même idée, mais au lieu de retourner au cylindre 0, retourner au premier cylindre qui a une requête

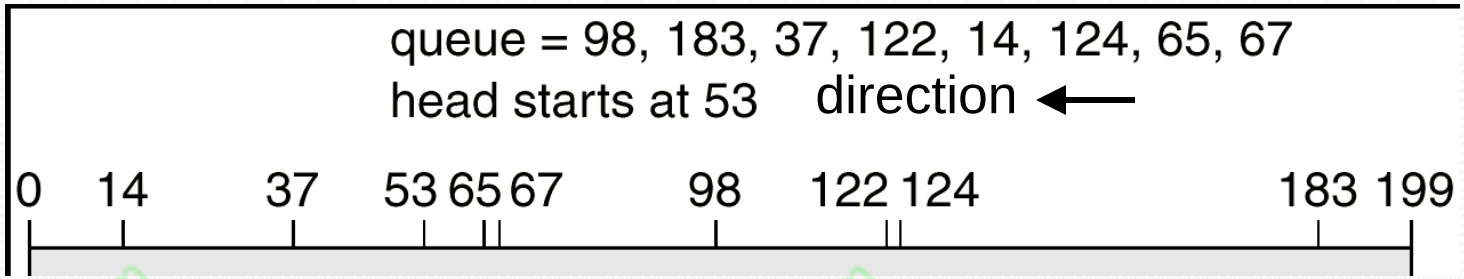
www.al7ibre.com

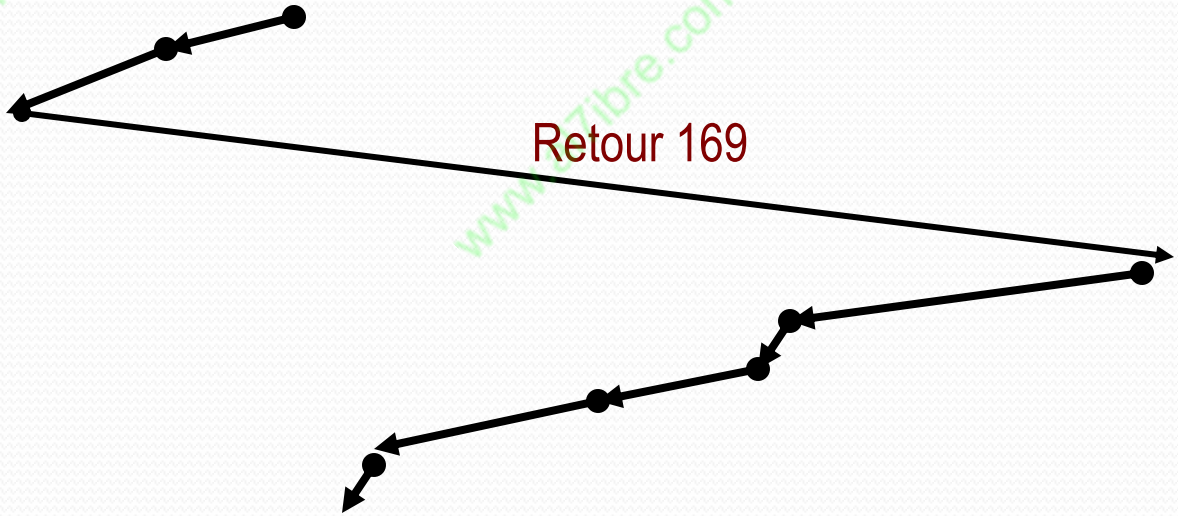
C-LOOK



153 sans considérer le retour (19.1 en moyenne) (26 pour SCAN)
MAIS 322 avec retour (40.25 en moyenne)
Normalement le retour sera rapide donc le coût réel sera entre les deux

C-LOOK avec direction initiale opposée





Résultats très semblables:
157 sans considérer le retour, 326 avec le retour

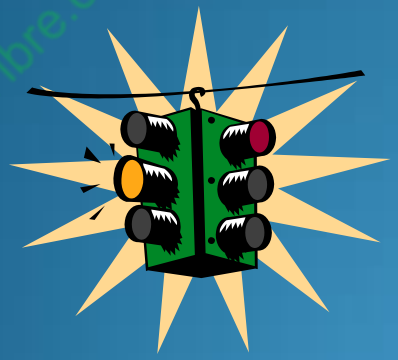
Comparaison

- Si la file souvent ne contient que très peu d'éléments, l'algorithme du 'premier servi ' devrait être préféré (simplicité)
- Sinon, SSTF ou SCAN ou C-SCAN?
- En pratique, il faut prendre en considération:
  - Les temps réels de déplacement et retour au début
  - L`organisation des fichiers et des répertoires
    - Les répertoires sont sur disque aussi…
  - La longueur moyenne de la file
  - Le débit d 'arrivée des requêtes

www.al7ibre.com

Synchronisation de Processus

Chapitre 5



www.al7ibre.com

Synchronisation de Processus

1. **Conditions de Concurrence**
2. **Sections Critiques**
3. **Exclusion Mutuelle**
4. **Sommeil & Activation**
5. **Sémaphores**
6. **Mutex**
7. **Moniteurs**

www.al7ibre.com

Problèmes avec concurrence = parallélisme

- Les processus concurrents doivent parfois partager données (fichiers ou mémoire commune) et ressources
  - On parle donc de tâches *coopératives*
- Si l'accès n'est pas contrôlé, le résultat de l'exécution du programme pourra dépendre de l'ordre d'entrelacement de l'exécution des instructions (*non-déterminisme*).
- Un programme pourra donner des résultats différents et parfois indésirables

www.al7ibre.com

Un exemple

- Deux processus exécutent cette même procédure et partagent la même base de données
- Ils peuvent être interrompus n'importe où
- Le résultat de l'exécution concurrente de P1 et P2 dépend de l'ordre de leur *entrelacement*

M. X demande une réservation d'avion

Base de données dit que fauteuil A est disponible

Fauteuil A est assigné à X et marqué occupé

www.al7ibre.com

Vue globale d'une exécution possible

P1

P2

Interruption ou retard

M. Leblanc demande une réservation d'avion

M. Guy demande une réservation d'avion

Base de données dit que fauteuil 30A est disponible

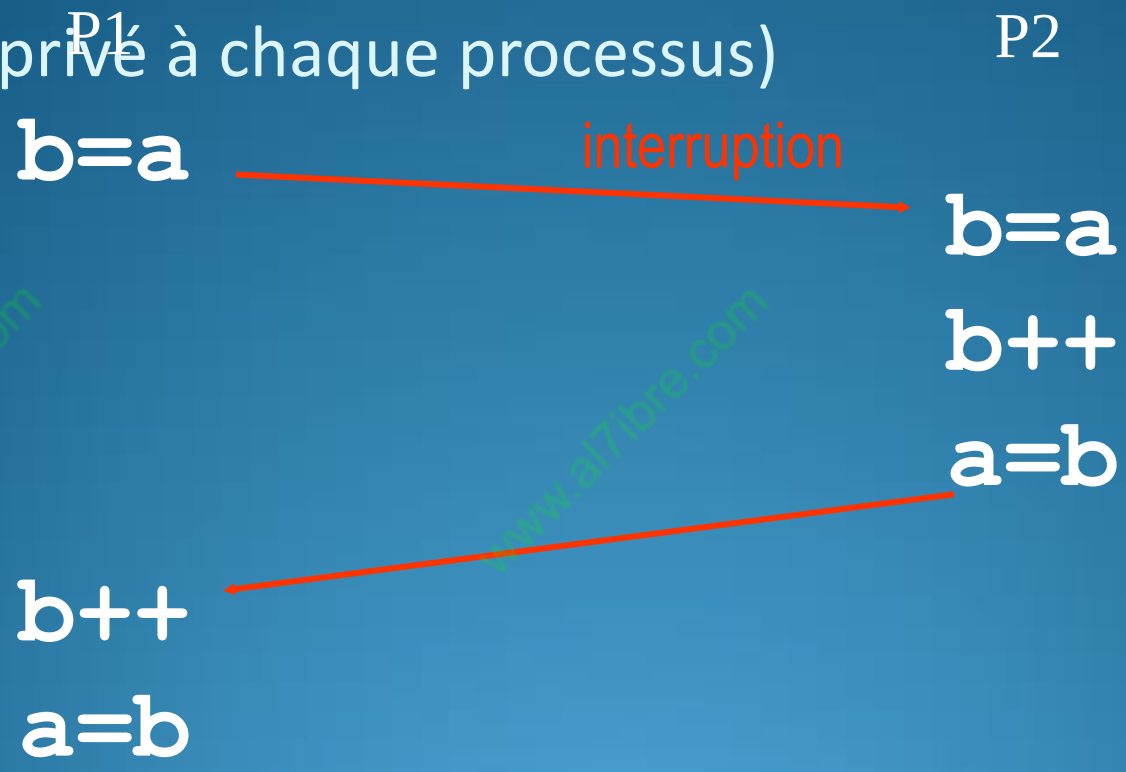
Base de données dit que fauteuil 30A est disponible

Fauteuil 30A est assigné à Leblanc et marqué occupé

Fauteuil 30A est assigné à Guy et marqué occupé

www.al7ibre.com

Deux opérations en parallèle sur une var a partagée (b est privé à chaque processus)



Supposons que a soit 0 au début

P1 travaille sur le vieux a donc le résultat final sera a=1.

Serait a=2 si les deux tâches sont exécutées l'une après l'autre

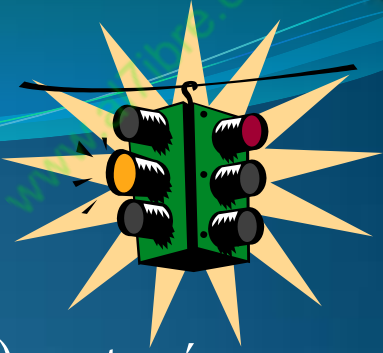
Si a était sauvegardé quand P1 est interrompu, il ne pourrait pas être partagé avec P2 (il y aurait deux a tandis que nous en voulons une seule)

Section Critique

- Partie d'un programme dont l'exécution ne doit pas *entrelacer* avec autres programmes
- Une fois qu'un tâche y entre, il faut lui permettre de terminer cette section sans permettre à autres tâches de jouer sur les mêmes données

www.al7ibre.com

Le problème de la section critique



- Lorsqu'un processus manipule une donnée (ou ressource) partagée, nous disons qu'il se trouve dans une section critique (SC) (associée à cette donnée)
- Le problème de la section critique est de trouver un algorithme d`*exclusion mutuelle* de processus dans l`exécution de leur SCs afin que le résultat de leurs actions *ne dépendent pas* de *l'ordre d'entrelacement* de leur exécution (avec un ou plusieurs processeurs)
- L'exécution des sections critiques doit être mutuellement exclusive: à tout instant, un seul processus peut exécuter une SC pour une var donnée (même lorsqu'il y a plusieurs processeurs)
- Ceci peut être obtenu en plaçant des instructions spéciales dans les sections d`entrée et sortie
- Pour simplifier, dorénavant nous faisons l'hypothèse qu'il n'y a q'une seule SC dans un programme.

www.al7ibre.com

Structure du programme

- Chaque processus doit donc demander une permission avant d'entrer dans une section critique (SC)
- La section de code qui effectue cette requête est la section d'entrée
- La section critique est normalement suivie d'une section de sortie
- Le code qui reste est la section restante (SR): non-critique

```
repeat
   section d'entrée
    section critique
   section de sortie
    section restante
forever
```

www.al7ibre.com

Application

M. X demande une réservation d'avion

Section d'entrée

Base de données dit que fauteuil A est disponible

Fauteuil A est assigné à X et marqué occupé

Section de sortie

Section critique

Critères nécessaires pour solutions valides

- Exclusion Mutuelle
  - À tout instant, au plus un processus peut être dans une section critique (SC) pour une variable donnée
- Non interférence:
  - Si un processus s'arrête dans sa section restante, ceci ne devrait pas affecter les autres processus
- Mais on fait l'hypothèse qu'un processus qui entre dans une section critique, en sortira.

www.al7ibre.com

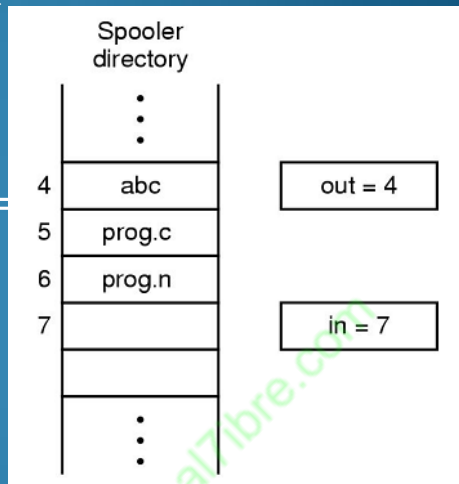
Critères nécessaires pour solutions valides

- Progrès:
  - absence d`interblocage (Chap 6)
  - si un processus demande d`entrer dans une section critique à un moment où aucun autre processus en fait requête, il devrait être en mesure d'y entrer
- Absence de famine: aucun processus éternellement empêché d'atteindre sa SC

- Difficile à obtenir, nous verrons…

Conditions de Concurrence

- Conditions de concurrence (*race conditions*): situation où 2 processus ou plus effectuent des lectures et des écritures conflictuelles.
- Exemple du Spouler d'impression
  - Un processus qui veut imprimer un fichier, entre son nom dans un **répertoire de spoule**
  - Le processus **démon d'impression** regarde périodiquement s'il y a des fichiers à imprimer. Il a 2 variables:
    - **in**: pointe vers la prochaine entrée libre.
    - **out**: pointe vers le prochain fichier à imprimer
  - *in* = 7, *out* = 4
  - A et B deux processus qui veulent imprimer un fichier
  - A >> lire *in*, *next_free_slot* = 7
  - Interruption: la CPU bascule vers le processus B
  - B >> lire *in*, *next_free_slot* = 7, entrée7 = fichierB, *in* =
  - A >> entrée7 = fichierA, *in* = 8
  - Problème: le fichierB ne sera pas imprimé



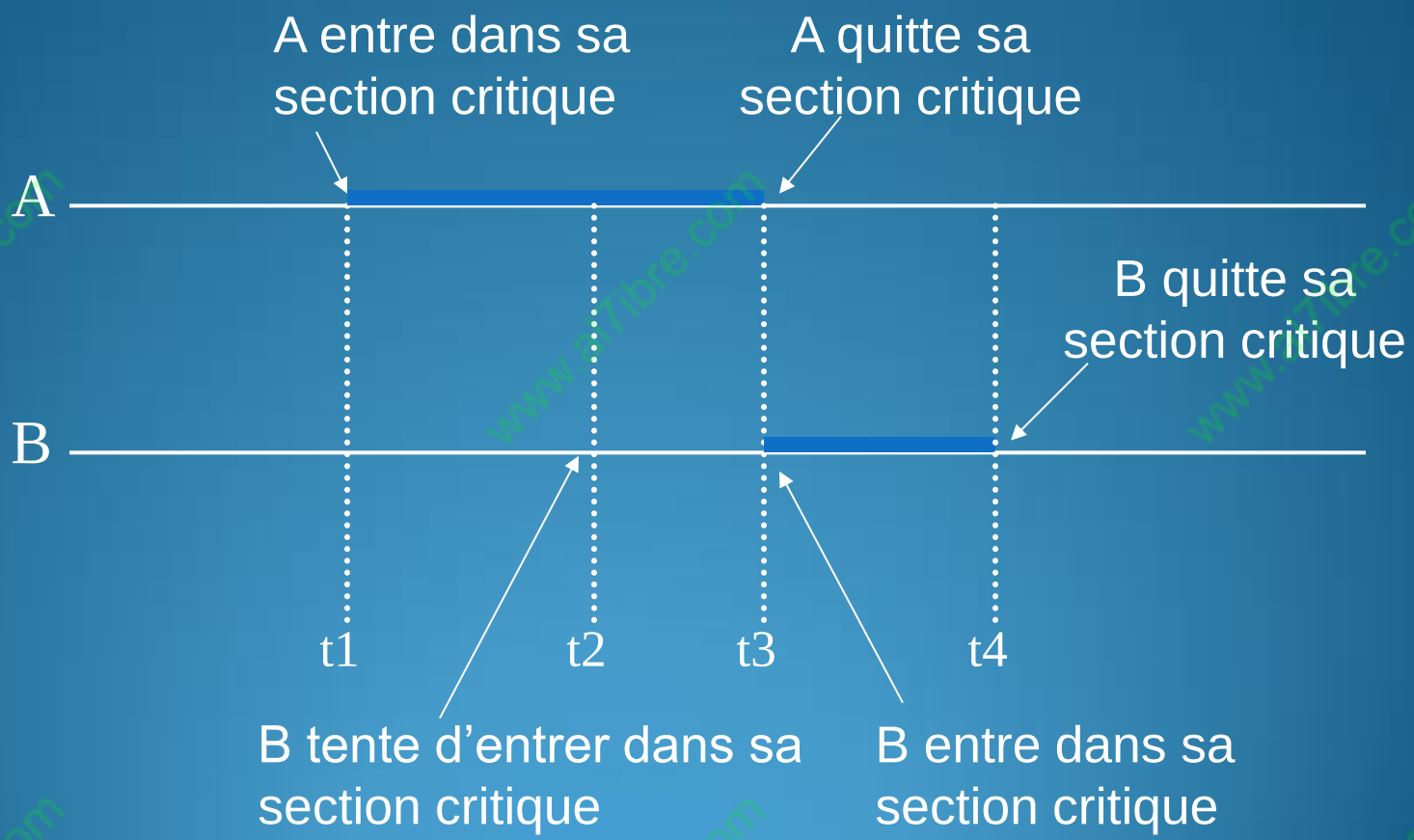
… Conditions de Concurrence

- Comment éviter les conditions de concurrence?
- Solution: Interdire que plusieurs processus lisent et écrivent des données partagées simultanément.
- Exclusion Mutuelle: permet d'assurer que si un processus utilise une variable ou fichier partagés, les autres processus seront exclus de la même activité

www.al7ibre.com

Les Sections Critiques

les Sections Critiques, méthode d'exclusion mutuelle



L'Exclusion Mutuelle avec Attente Active (*busy waiting*)

- Désactivation des interruptions
  - Après son entrée dans une SC, un processus désactive les interruptions, puis les réactive
  - Il empêche ainsi l'horloge d'envoyer des interruptions et le processeur de basculer
  - Il est imprudent de permettre à des processus user de désactiver les interruptions
- Variables de verrou *(lock)*
  - Avant d'entrer en SC, tester la valeur de verrou, si verrou = 0, verrou ← 1, entrer en SC
  - Défaillance: 2 processus peuvent entrer simultanément dans leurs sections critiques comme le spouler d'impression
- Alternance Stricte
  - la variable *turn* porte le numéro du processus dont c'est le tour d'entrer en SC. Chaque processus inspecte la valeur de la variable, avant d'entrer en SC.
  - Inconvénient: consomme bcp de temps CPU

… Exclusion Mutuelle avec Attente Active

(*busy waiting*)

- … Alternance Stricte

```
while (TRUE) {
    while (turn != 0);
    critical_region();
    turn = 1;
    non_critical_region();
}
```

```
while (TRUE) {
    while (turn != 1);
    critical_region();
    turn = 0;
    non_critical_region();
}
```

- Les attentes actives sont performantes dans le cas où elles sont brèves. En effet, il y' a risque d'attente
  - P0 quitte la CS, turn = 1
  - P1 termine sa CS, turn = 0
  - Les 2 processus sont en section non critique
  - P0 exécute sa boucle, quitte la SC et turn = 1
  - Les 2 processus sont en section non critique
  - P0 quoiqu'il a terminé, il ne peut pas entrer en SC, il est bloqué

… Sommeil & Activation

- Problème de blocage:
  - Le consommateur note que le tampon est vide
  - Interruption: arrêt du consommateur sans qu'il parte en sommeil
  - Le producteur insère un jeton, incrémente le décompte, appelle *wakeup* pour réveiller le consommateur
  - Le signal *wakeup* est perdu, car le consommateur n'est pas en sommeil
  - Le consommateur reprend, pour lui le tampon est vide, il dort
  - Le producteur remplit le tampon et dort
- Solution: ajouter un *bit d'attente d'éveil*.
  - Quand un *wakeup* est envoyé à un processus le bit est à 1;
  - le consommateur teste le bit, s'il est à 1, il le remet à 0 et reste en éveil
  - Cette solution est + difficile à généraliser en cas de + sieurs processus.

Une leçon à retenir…

- À fin que des processus avec des variables partagées puissent réussir, il est nécessaire que tous les processus impliqués utilisent le même algorithme de coordination
  - Un protocole commun

www.al7ibre.com

Critique des solutions par logiciel

- Difficiles à programmer! Et à comprendre!
  - Les solutions que nous verrons dorénavant sont toutes basées sur l'existence d'instructions spécialisées, qui facilitent le travail.

- Les processus qui requièrent l'entrée dans leur SC sont occupés à attendre (busy waiting); consommant ainsi du temps de processeur
  - Pour de longues sections critiques, il serait préférable de bloquer les processus qui doivent attendre…

www.al7ibre.com

Solutions matérielles: désactivation des interruptions

- Sur un uniprocesseur: exclusion mutuelle est préservée mais l'efficacité se détériore: lorsque dans SC il est impossible d'entrelacer l'exécution avec d'autres processus dans une SR
- Perte d'interruptions
- Sur un multiprocesseur: exclusion mutuelle n'est pas préservée
- Une solution qui n'est généralement pas acceptable

```
Process Pi:
repeat
   inhiber interrupt
    section critique
   rétablir interrupt
    section restante
forever
```

Solutions basées sur des instructions fournies par le SE (appels du système)

- Les solutions vues jusqu'à présent sont difficiles à programmer et conduisent à du mauvais code.
- On voudrait aussi qu`il soit plus facile d'éviter des erreurs communes, comme interblocages, famine, etc.
  - Besoin d'instruction à plus haut niveau
- Les méthodes que nous verrons dorénavant utilisent des instructions puissantes, qui sont implantées par des appels au SE (system calls)

www.al7ibre.com

Sémaphores

- Un sémaphore S est un entier qui, sauf pour l'Initialisation, est accessible seulement par ces 2 opérations atomiques et mutuellement exclusives:
  - wait(S)
  - signal(S)
- Il est partagé entre tous les procs qui s`intéressent à la même section critique
- Les sémaphores seront présentés en deux étapes:
  - sémaphores qui sont occupés à attendre (busy waiting)
  - sémaphores qui utilisent des files d 'attente
- On fait distinction aussi entre sémaphores compteurs et sémaphores binaires, mais ce derniers sont moins puissants.

www.al7ibre.com

Sémaphores occupés à attendre

(busy waiting)

- La façon la plus simple d'implanter les sémaphores.
- Utiles pour des situations où l'attente est brève, ou il y a beaucoup d'UCTs
- S est un entier initialisé à une valeur positive, de façon que un premier processus puisse entrer dans la SC
- Quand S>0, jusqu'à n processus peuvent entrer
- Quand S<=0, il faut attendre S+1 signals (d'autres processus) pour entrer

```
wait(S):
while S<=0 {};
 S--;
```

Attend si no. de processus qui peuvent entrer = 0 ou négatif

```
signal(S):
 S++;
```

Augmente de 1 le no des processus qui peuvent entrer

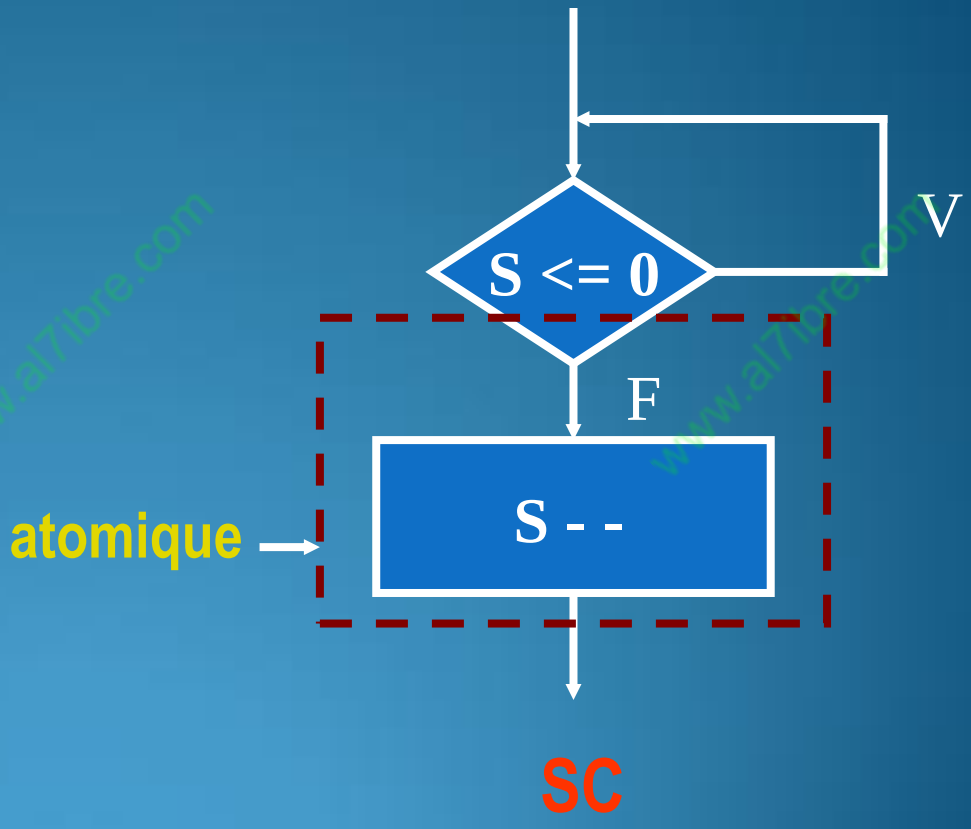
Atomicité

Wait: La séquence test-décrément est atomique, mais pas la boucle!

Signal est atomique.

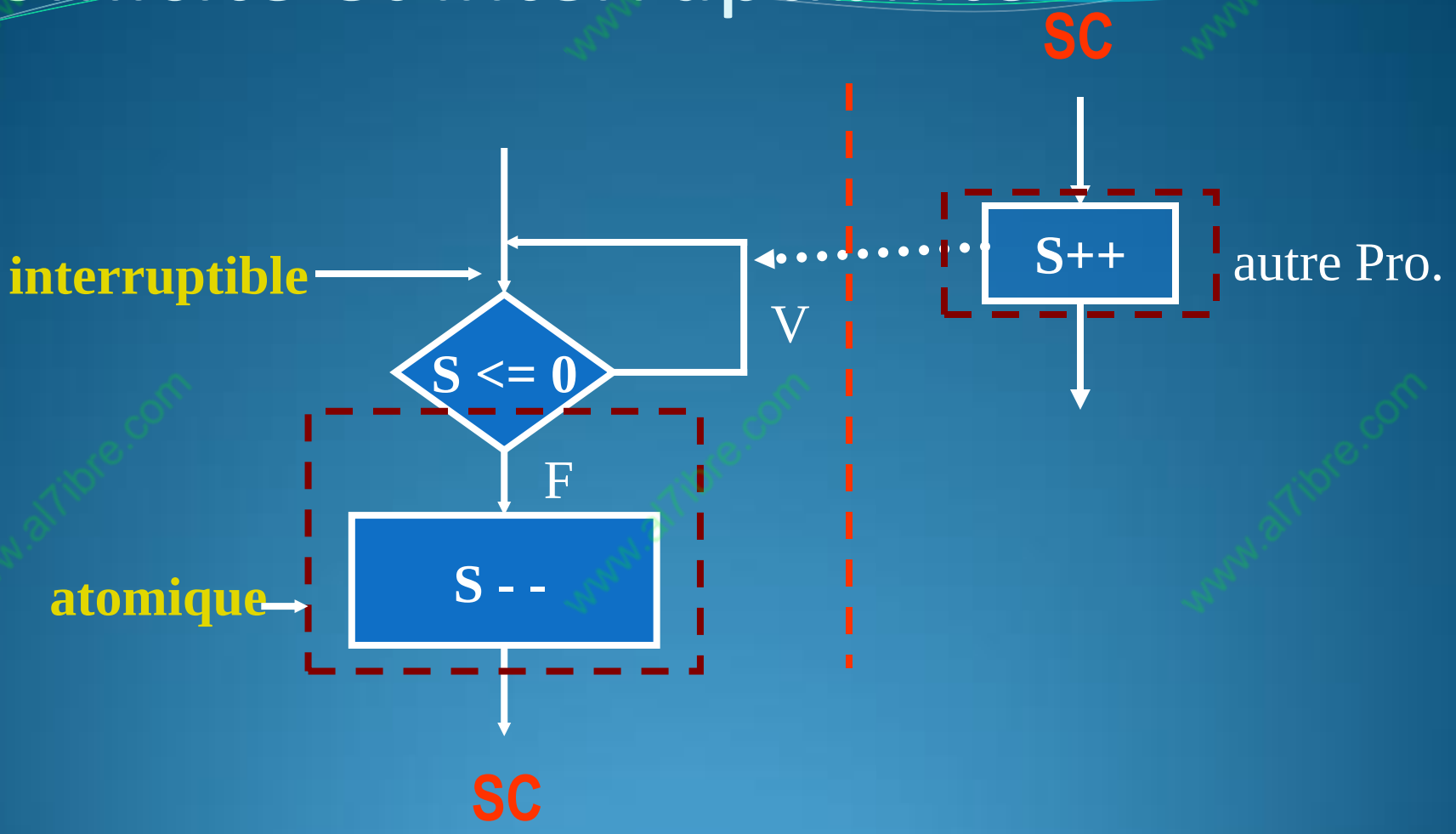
Rappel: les sections atomiques ne peuvent pas être exécutées simultanément par différent processus

(ceci peut être obtenu un utilisant un des mécanismes précédents)



www.al7ibre.com

Atomicité et interruptibilité



La boucle n'est pas atomique pour permettre à un autre processus d'interrompre l'attente sortant de la SC

Utilisation des sémaphores pour sections critiques

- Pour n processus
- Initialiser S à 1
- Alors 1 seul processus peut être dans sa SC
- Pour permettre à k processus d'exécuter SC, initialiser S à k

```
processus Ti:
repeat
   wait(S);
    SC
   signal(S);
    SR
forever
```

www.al7ibre.com

Initialise S à >=1

```
processus T1:
    repeat
    wait(S);
        SC
    signal(S);
        SR
    forever
```

```
processus T2:
repeat
  wait(S);
    SC
  signal(S);
    SR
forever
```

Semaphores: vue globale

Peut être facilement généralisé à plus. processus

Utilisation des sémaphores pour synchronisation de processus

- On a 2 processus : T1 et T2
- Énoncé S1 dans T1 doit être exécuté avant énoncé S2 dans T2
- Définissons un sémaphore S
- Initialiser S à 0

- Synchronisation correcte lorsque T1 contient:

```
S1;
signal(S);
```

- et que T2 contient:

```
wait(S);
S2;
```

www.al7ibre.com

Interblocage et famine avec les sémaphores

- Famine: un processus peut n'arriver jamais à exécuter car il ne teste jamais le sémaphore au bon moment
- Interblocage: Supposons S et Q initialisés à 1

T0 T1

wait(S)

wait(Q)

wait(Q) wait(S)

Sémaphores: observations

```
wait(S):
while S<=0 {};
 S--;
```

- Quand S >= 0:
  - Le nombre de processus qui peuvent exécuter wait(S) sans devenir bloqués = S
    - S processus peuvent entrer dans la SC
    - noter puissance par rapport à mécanismes déjà vus
    - dans les solutions où S peut être >1il faudra avoir un 2ème sém. pour les faire entrer un à la fois (excl. mutuelle)
- Quand S devient > 1, le processus qui entre le premier dans la SC est le premier à tester S (choix aléatoire)
  - ceci ne sera plus vrai dans la solution suivante
- Quand S < 0: le nombre de processus qui attendent sur S est = |S|

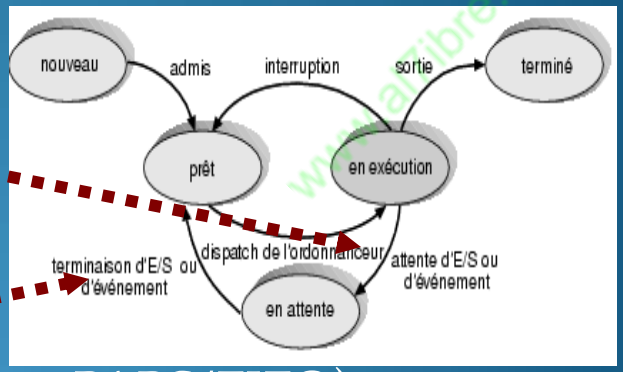
www.al7ibre.com

Comment éviter l'attente occupée et le choix aléatoire dans les sémaphores

- Quand un processus doit attendre qu'un sémaphore devienne plus grand que 0, il est mis dans une file d'attente de processus qui attendent sur le même sémaphore.
- Les files peuvent être PAPS (FIFO), avec priorités, etc. Le SE contrôle l`ordre dans lequel les processus entrent dans leur SC.
- *wait* et *signal* sont des appels au SE comme les appels à des opérations d'E/S.
- Il y a une file d'attente pour chaque sémaphore comme il y a une file d'attente pour chaque unité d'E/S.

Sémaphores sans attente occupée

- Un sémaphore S devient une structure de données:
  - Une valeur
  - Une liste d'attente L
- Un processus devant attendre un sémaphore S, est bloqué et ajouté la file d'attente S.L du sémaphore (v. état bloqué = attente chap 4).



- signal(S) enlève (selon une politique juste, ex: PAPS/FIFO) un processus de S.L et le place sur la liste des processus prêts/ready.

Implementation

(les boîtes réprésentent des séquences non-interruptibles)

```
wait(S):  S.value --;
          if S.value < 0   {        // SC occupée
                add this processus to S.L;
                block               // processus mis en état attente (wait)
                }
```

```
signal(S): S.value ++;
           if S.value ≤ 0 {         // des processus attendent
                remove a process P from S.L;
                wakeup(P)           // processus choisi devient prêt
                }
```

S.value doit être initialisé à une valeur non-négative (dépendant de l'application, v. exemples)

Figure montrant la relation entre le contenu de la file et la valeur de S

Quand S < 0: le nombre de processus qui attendent sur S est = |S|

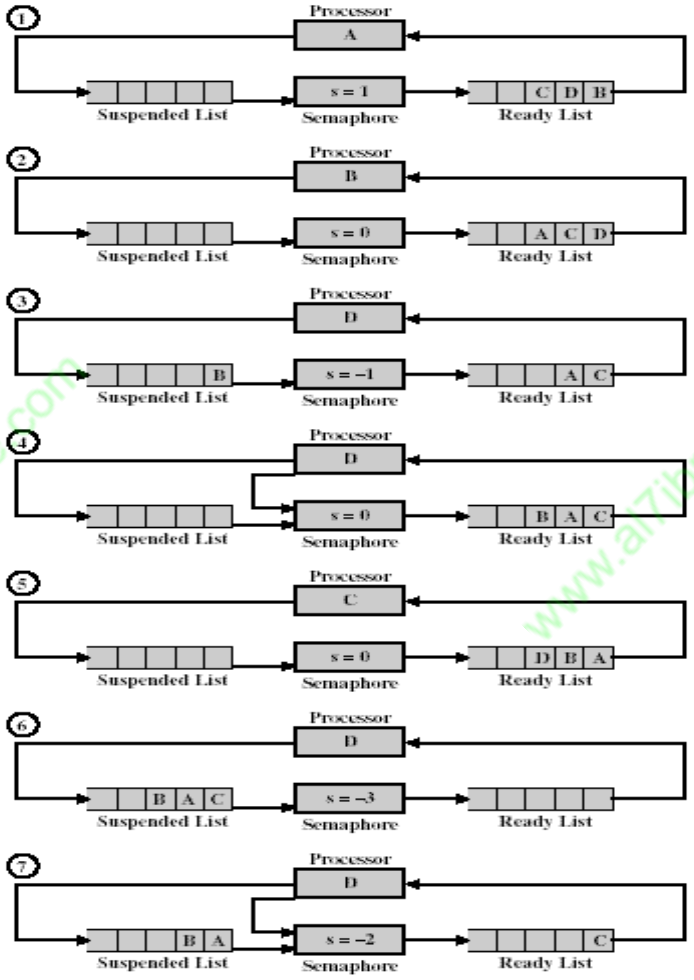


Figure 5.8 Example of Semaphore Mechanism

Wait et signal contiennent elles mêmes des SC!

- Les opérations *wait* et *signal* doivent être exécutées atomiquement (un seul thr. à la fois)
- Dans un système avec 1 seule UCT, ceci peut être obtenu en inhibant les interruptions quand un processus exécute ces opérations
- L'attente occupée dans ce cas ne sera pas trop onéreuse car *wait* et *signal* sont brefs

www.al7ibre.com

Problèmes classiques de synchronisation

- Tampon borné (producteur-consommateur)
- Écrivains - Lecteurs
- Les philosophes mangeant

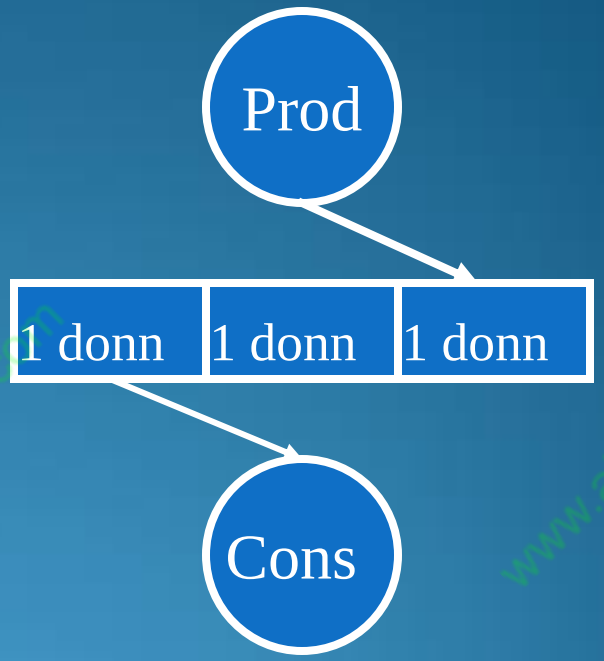
www.al7ibre.com

Le pb du producteur - consommateur

- **Un problème classique dans l ’étude des processus communicants**
  - un processus *producteur* produit des données (p.ex.des enregistrements d ’un fichier) pour un processus *consommateur*

www.al7ibre.com

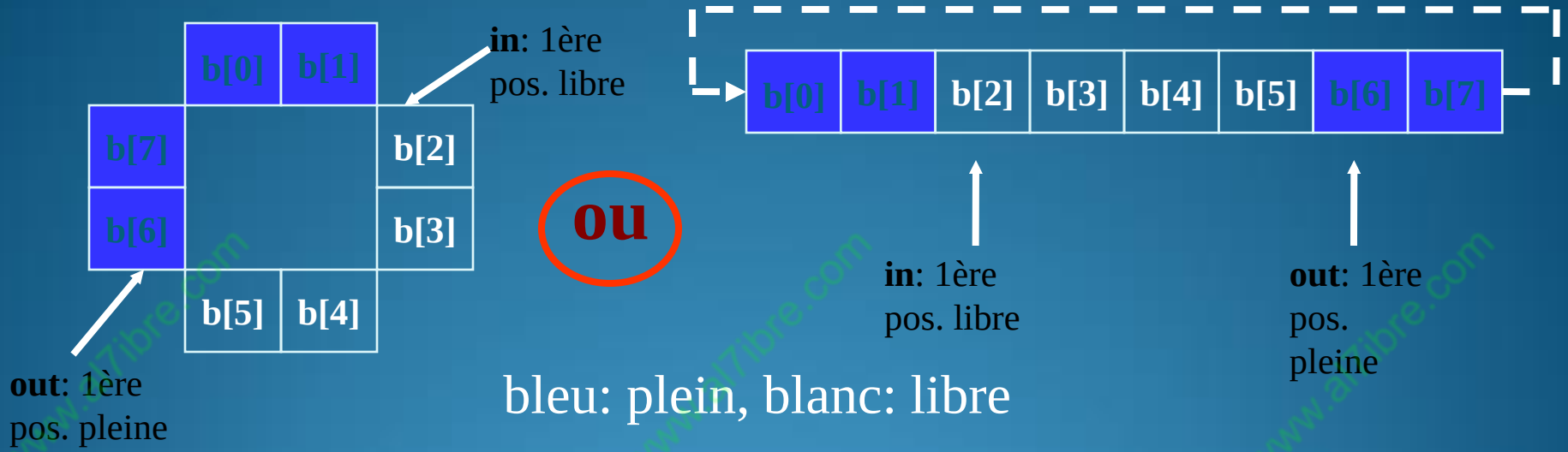
Tampons de communication



Si le tampon est de longueur 1, le producteur et consommateur doivent forcement aller à la même vitesse

Des tampons de longueur plus grandes permettent une certaine indépendance. P.ex. à droite le consommateur a été plus lent

Le tampon borné (bounded buffer)
une structure de données fondamentale dans les SE



bleu: plein, blanc: libre

Le tampon borné se trouve dans la mémoire partagée entre consommateur et usager

Pb de sync entre processus pour le tampon borné

- Étant donné que le prod et le consommateur sont des processus indépendants, des problèmes pourraient se produire en permettant accès simultané au tampon
- Les sémaphores peuvent résoudre ce problème

www.al7ibre.com

Sémaphores: rappel.

- **Soit S un sémaphore sur une SC**
  - il est associé à une file d ’attente
  - S positif: S processus peuvent entrer dans SC
  - S zéro: aucun processus ne peut entrer, aucun processus en attente
  - S négatif: |S| processus dans file d ’attente
- **Wait(S): S - -**
  - si après S >= 0, processus peut entrer dans SC
  - si S < 0, processus est mis dans file d ’attente
- **Signal(S): S++**
  - si après S<= 0, il y avait des processus en attente, et un processus est réveillé
- **Indivisibilité = atomicité de ces ops**

www.al7ibre.com

Solution avec sémaphores

- **Un sémaphore S pour exclusion mutuelle sur l'accès au tampon**
  - Les sémaphores suivants ne font pas l'EM
- **Un sémaphore N pour synchroniser producteur et consommateur sur le nombre d'éléments consommables dans le tampon**
- **Un sémaphore E pour synchroniser producteur et consommateur sur le nombre d'espaces libres**

www.al7ibre.com

Solution de P/C: tampon circulaire fini de dimension k

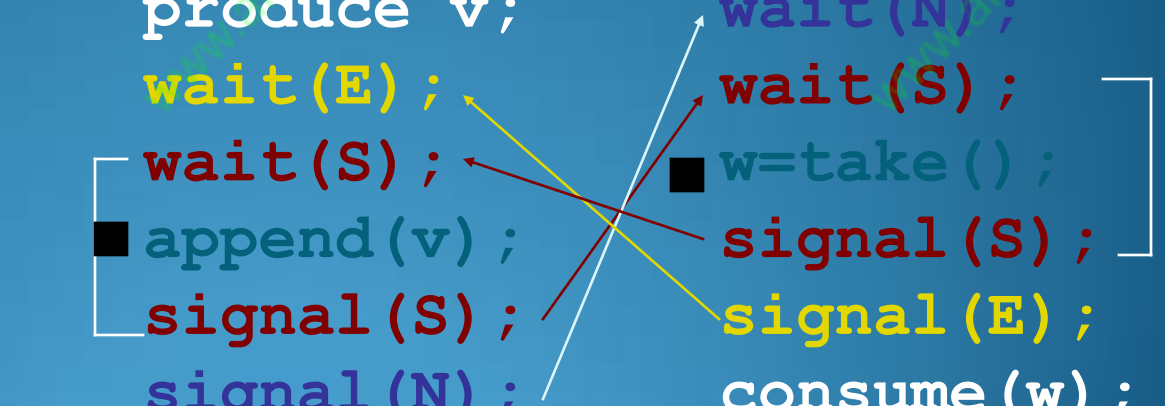
```
Initialization: S.count=1; //excl. mut.
                N.count=0; //esp. pleins
                E.count=k; //esp. vides
```

```
append(v):
  b[in]=v;
  In ++ mod k;
```

```
take():
  w=b[out];
  Out ++ mod k;
  return w;
```

```
Producer:
repeat
   produce v;
   wait(E);
   wait(S);
   append(v);
   signal(S);
   signal(N);
forever
```

```
Consumer:
repeat
   wait(N);
   wait(S);
   w=take();
   signal(S);
   signal(E);
   consume(w);
forever
```



- Sections critiques

Points importants à étudier

- **dégâts possibles en interchangeant les instructions sur les sémaphores**
  - ou en changeant leur initialisation
- **Généralisation au cas de plus. prods et cons**

www.al7ibre.com

Problème des lecteurs - rédacteurs

- Plusieurs processus peuvent accéder à une base de données
  - Pour y lire ou pour y écrire
- Les rédacteurs doivent être synchronisés entre eux et par rapport aux lecteurs
  - il faut empêcher à un processus de lire pendant l'écriture
  - il faut empêcher à deux rédacteurs d 'écrire simultanément
- Les lecteurs peuvent y accéder simultanément

www.al7ibre.com

Une solution (n'exclut pas la famine)

- Variable readcount: nombre de processus lisant la base de données
- Sémaphore mutex: protège la SC où readcount est mis à jour
- Sémaphore wrt: exclusion mutuelle entre rédacteurs et lecteurs
- Les rédacteurs doivent attendre sur wrt
  - les uns pour les autres
  - et aussi la fin de toutes les lectures
- Les lecteurs doivent
  - attendre sur wrt quand il y a des rédacteurs qui écrivent
  - bloquer les rédacteurs sur wrt quand il y a des lecteurs qui lisent
  - redémarrer les rédacteurs quand personne ne lit

www.al7ibre.com

Les données et les rédacteurs

Données: deux sémaphores et une variable

```
mutex, wrt: semaphore (init. 1);
readcount : integer (init. 0);
```

Rédacteur

```
wait(wrt);
        . . .
    // écriture
        . . .
signal(wrt);
```

Les lecteurs

```
wait(mutex);
        readcount ++ ;
        if readcount == 1 then wait(wrt);
signal(mutex);
```

Le premier lecteur d 'un groupe pourrait devoir attendre sur wrt, il doit aussi bloquer les rédacteurs. Quand il sera entré, les suivants pourront entrer librement

```
//SC: lecture
```

```
wait(mutex);
        readcount -- ;
        if readcount == 0 then signal(wrt);
signal(mutex):
```

Le dernier lecteur sortant doit permettre l`accès aux rédacteurs

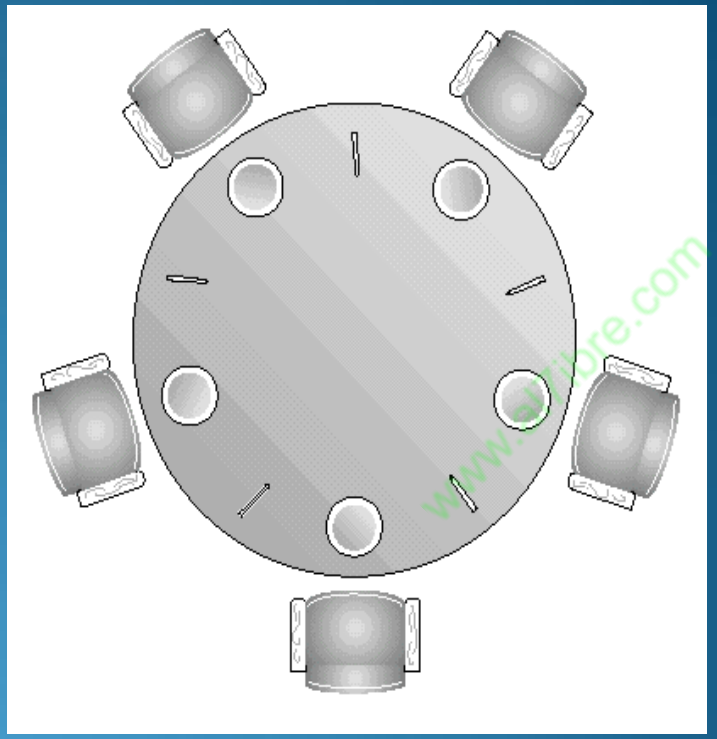
Observations

- Le 1er lecteur qui entre dans la SC bloque les rédacteurs (wait (wrt)), le dernier les remet en marche (signal (wrt))
- Si 1 rédacteur est dans la SC, 1 lecteur attend sur wrt, les autres sur mutex
- un signal(wrt) peut faire exécuter un lecteur ou un rédacteur

www.al7ibre.com

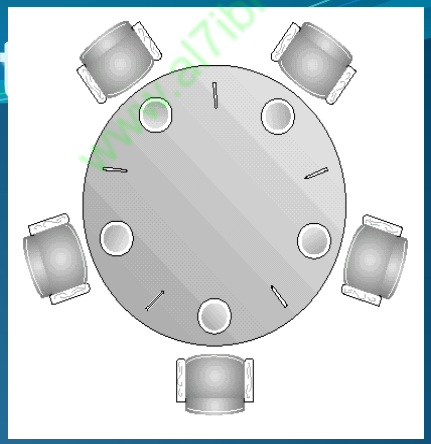
Le problème des philosophes mangeant

- 5 philosophes qui mangent et pensent
- Pour manger il faut 2 fourchettes, droite et gauche
- On en a seulement 5!
- Un problème classique de synchronisation
- Illustre la difficulté d'allouer ressources aux processus tout en évitant interblocage et famine



www.al7ibre.com

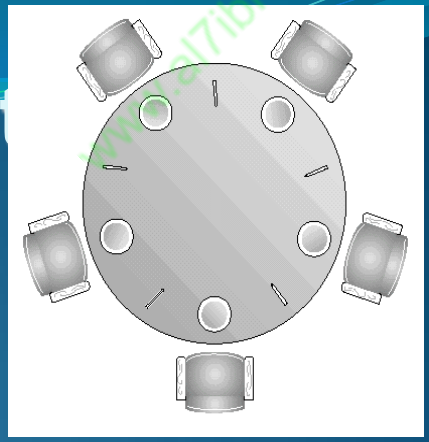
Le problème des philosophes mangeant



- **Un processus par philosophe**
- **Un sémaphore par fourchette:**
  - fork: array[0..4] of semaphores
  - Initialisation: fork[i] =1 for i:=0..4
- **Première tentative:**
  - interblocage si chacun débute en prenant sa fourchette gauche!
    - **`Wait(fork[i])`**

```
processus Pi:
repeat
 think;
 wait(fork[i]);
 wait(fork[i+1 mod 5]);
 eat;
 signal(fork[i+1 mod 5]);
 signal(fork[i]);
forever
```

Le problème des philosophes mangeant



- **Une solution: admettre seulement 4 philosophes à la fois qui peuvent tenter de manger**
- **Il y aura touj. au moins 1 philosophe qui pourra manger**
  - même si tous prennent 1 fourchette
- **Ajout d'un sémaphore T qui limite à 4 le nombre de philosophes "assis à la table"**
  - initial. de T à 4
- **N'empêche pas famine!**

```
processus Pi:
repeat
  think;
  wait(T);
  wait(fork[i]);
  wait(fork[i+1 mod 5]);
  eat;
  signal(fork[i+1 mod 5]);
  signal(fork[i]);
  signal(T);
forever
```

Avantage des sémaphores
(par rapport aux solutions précédentes)

- Une seule variable partagée par section critique
- deux seules opérations: wait, signal
- contrôle plus localisé (que avec les précéds)
- extension facile au cas de plus. processus
- possibilité de faire entrer plus. processus à la fois dans une section critique
- gestion de files d`attente par le SE: famine évitée si le SE est équitable (p.ex. files FIFO)

www.al7ibre.com

Problème avec sémaphores: difficulté de programmation

- **wait et signal sont dispersés parmi plusieurs processus, mais ils doivent se correspondre**
  - V. programme du tampon borné
- **Utilisation doit être correcte dans tous les processus**
- **Un seul "mauvais" processus peut faire échouer toute une collection de processus (p.ex. oublie de faire signal)**
- **Considérez le cas d`un processus qui a des waits et signals dans des boucles et des tests...**

www.al7ibre.com