

Allocation de la mémoire II

Problème

La mémoire se fragmente en petit blocs libres, les gros processus ne peuvent entrer.

Problème

La mémoire se fragmente en petit blocs libres, les gros processus ne peuvent entrer.

Source du problème :

les blocs sont de tailles variables

Problème

La mémoire se fragmente en petit blocs libres, les gros processus ne peuvent entrer.

Source du problème :

les blocs sont de tailles variables

Deuxième solution

Découper les processus en blocs de tailles fixes avec multiples translations :

La pagination

Pagination

Principe

Mémoire



Processus



Principe

- Découper la mémoire en **cadres** de taille fixe

Mémoire



Processus



Principe

- Découper la mémoire en **cadres** de taille fixe
- Découper le processus en **pages** de taille fixe

Mémoire



Processus



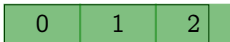
Principe

- Découper la mémoire en **cadres** de taille fixe
- Découper le processus en **pages** de taille fixe

Mémoire

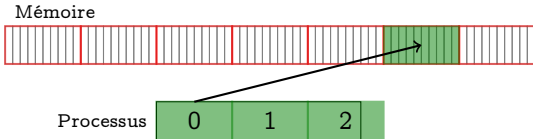


Processus



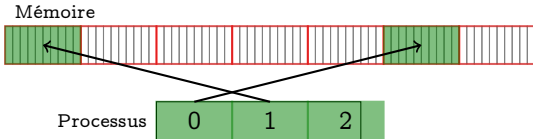
Principe

- Découper la mémoire en **cadres** de taille fixe
- Découper le processus en **pages** de taille fixe
- Placer les **pages** dans les **cadres**



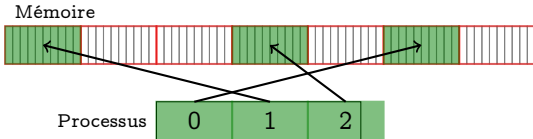
Principe

- Découper la mémoire en **cadres** de taille fixe
- Découper le processus en **pages** de taille fixe
- Placer les **pages** dans les **cadres**



Principe

- Découper la mémoire en **cadres** de taille fixe
- Découper le processus en **pages** de taille fixe
- Placer les **pages** dans les **cadres**



Blocs de taille fixe

- ✓ Pas de fragmentation !
- ✓ Découpage très simple
- ✓ Rien à faire à la compilation

Blocs de taille fixe

- ✓ Pas de fragmentation !
- ✓ Découpage très simple
- ✓ Rien à faire à la compilation

Adaptation

Redimensionnement des processus sans réallocation.

Blocs de taille fixe

- ✓ Pas de fragmentation !
- ✓ Découpage très simple
- ✓ Rien à faire à la compilation

Adaptation

Redimensionnement des processus sans réallocation.

Mais...

Perte de mémoire potentielle dans le dernier bloc
fragmentation interne

Quelle taille de cadre ?

Une puissance de deux :

- ✓ traduction d'adresses simple et efficace

Quelle taille de cadre ?

Une puissance de deux :

- ✓ traduction d'adresses simple et efficace

Exemple : @logique 16 bits

64 pages (6 bits) de 1ko maximum (10 bits)

Quelle taille de cadre ?

Une puissance de deux :

✓ traduction d'adresses simple et efficace

Exemple : @logique 16 bits

64 pages (6 bits) de 1ko maximum (10 bits)

0xA13E = 1010 0001 0011 1110

Quelle taille de cadre ?

Une puissance de deux :

✓ traduction d'adresses simple et efficace

Exemple : @logique 16 bits

64 pages (6 bits) de 1ko maximum (10 bits)

0xA13E = 1010 0001 0011 1110

Quelle taille de cadre ?

Une puissance de deux :


✓ traduction d'adresses simple et efficace

Exemple : @logique 16 bits

64 pages (6 bits) de 1ko maximum (10 bits)

0xA13E = 1010 0001 0011 1110

Page 40 101000



Quelle taille de cadre ?

Une puissance de deux :

✓ traduction d'adresses simple et efficace

Exemple : @logique 16 bits

64 pages (6 bits) de 1ko maximum (10 bits)

0xA13E = 1010 0001 0011 1110

Page 40 101000

Décalage 318 01 0011 1110

Quelle taille de cadre ?

Une puissance de deux :

✓ traduction d'adresses simple et efficace

Exemple : @logique 16 bits

64 pages (6 bits) de 1ko maximum (10 bits)

0xA13E = 1010 0001 0011 1110

Page 40 101000

Décalage 318 01 0011 1110

319^{ème} octet de la 41^{ème} page

Nombre de cadres

Adressage en binaire: 2^n cadres

Nombre de cadres

Adressage en binaire: 2^n cadres

Taille des pages/cadres

Adressage en binaire: 2^m octets

→ Taille de la RAM: 2^{n+m}

Nombre de cadres

Adressage en binaire: 2^n cadres

Taille des pages/cadres

Adressage en binaire: 2^m octets

→ Taille de la RAM: 2^{n+m}

Fragmentation interne

Les processus font rarement $k \times 2^m$ octets

→ En moyenne 1/2 cadre perdu par processus

En mémoire physique

Une table par processus: mémorise le cadre où est stocké chaque page du processus.

En mémoire physique

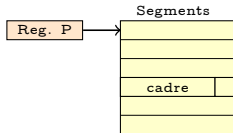
Une table par processus: mémorise le cadre où est stocké chaque page du processus.

Segments	
cadre	

Cadre: Correspondance

En mémoire physique

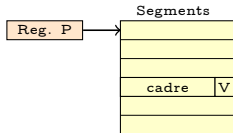
Une table par processus: mémorise le cadre où est stocké chaque page du processus.



Cadre: Correspondance

En mémoire physique

Une table par processus: mémorise le cadre où est stocké chaque page du processus.

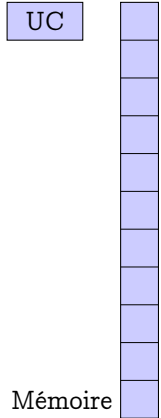


Cadre: Correspondance
V: validité

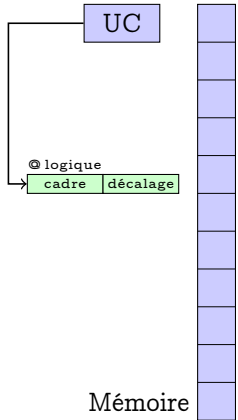
Couverture

La table couvre tout l'espace logique du processus: toutes les pages ne sont pas forcément valides.

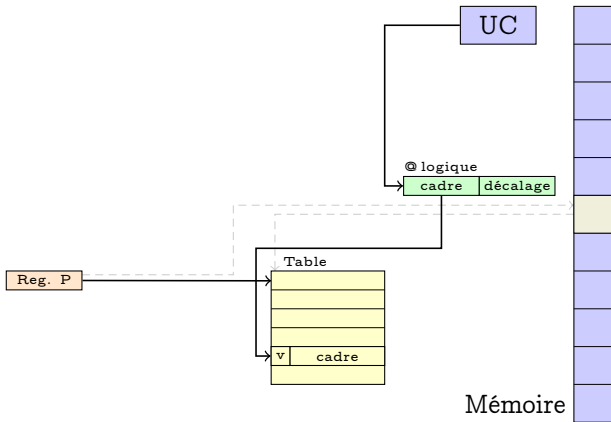
Pagination



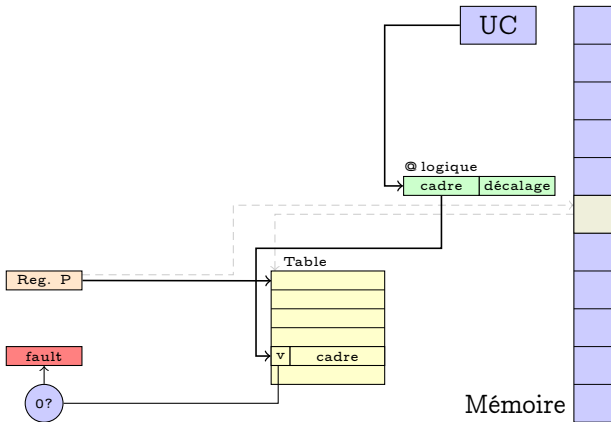
Pagination



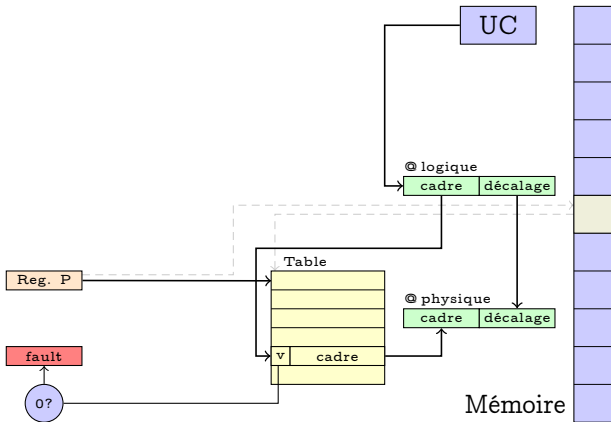
Pagination



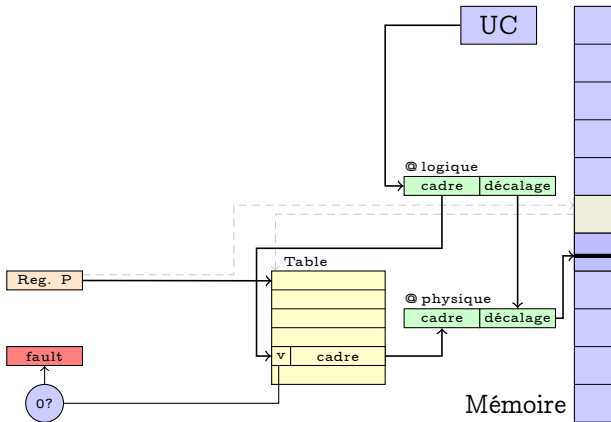
Pagination



Pagination



Pagination

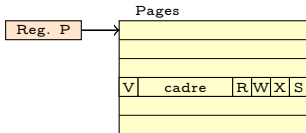


Gestion des droits

Dans la table des pages, indiquer les accès autorisés :
read, write, execute, supervisor...

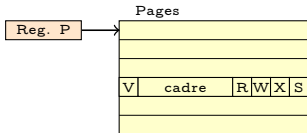
Gestion des droits

Dans la table des pages, indiquer les accès autorisés :
read, write, execute, supervisor...



Gestion des droits

Dans la table des pages, indiquer les accès autorisés : *read, write, execute, supervisor...*



En pratique

Configurés par l'OS, vérifiés à chaque accès par la MMU. En cas d'erreur, interruption...

Grand espace d'adressage (32bits ou plus)

Grand espace d'adressage (32bits ou plus)

- X Trop de pages ($n=20$, $m=12$)
 - grande table (2.5Mo par processus)

Grand espace d'adressage (32bits ou plus)

- X Trop de pages ($n=20$, $m=12$)
 - grande table (2.5Mo par processus)
- X Pages trop grosses ($n=10$, $m=12$)
 - fragmentation interne (2Mo par processus)

Grand espace d'adressage (32bits ou plus)

- X Trop de pages ($n=20$, $m=12$)
 - grande table (2.5Mo par processus)
- X Pages trop grosses ($n=10$, $m=12$)
 - fragmentation interne (2Mo par processus)

Paginer la table des pages

- Petites pages
- Plusieurs niveaux de tables de pages

Pagination Hierarchique

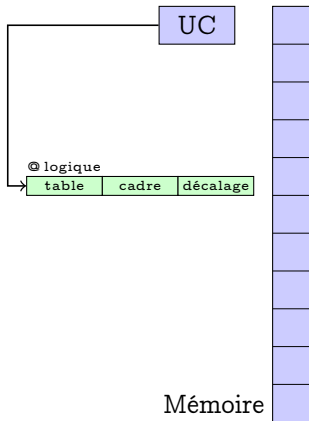
Pagination à deux niveaux

UC

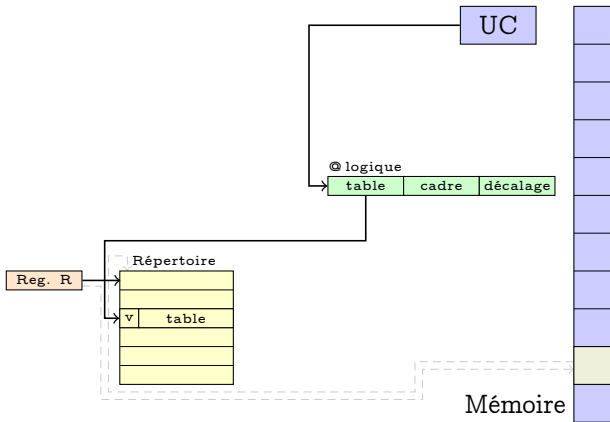


Mémoire

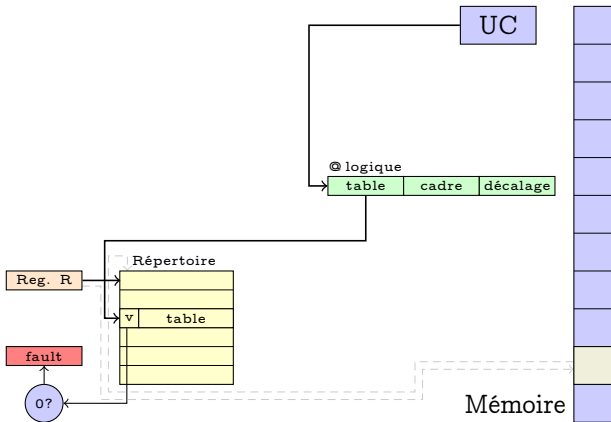
Pagination à deux niveaux



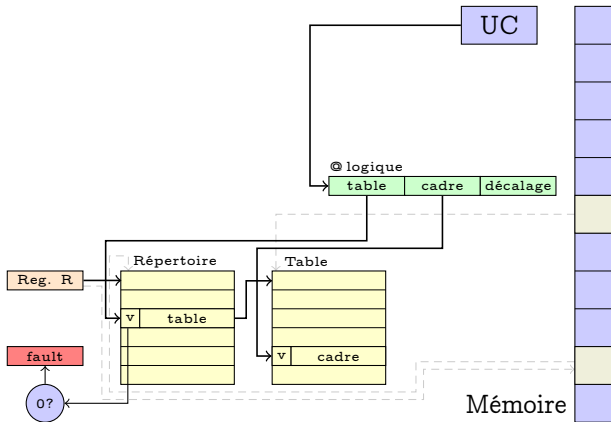
Pagination à deux niveaux



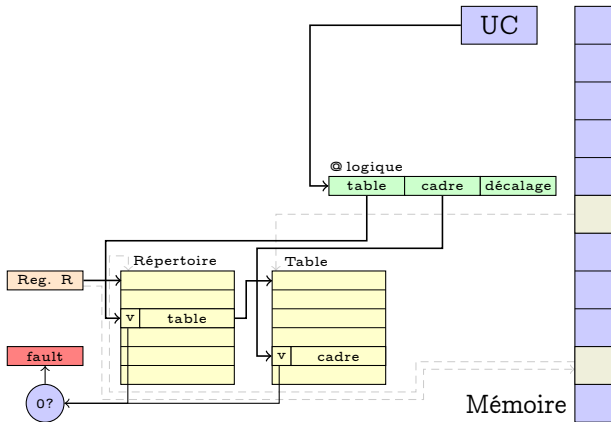
Pagination à deux niveaux



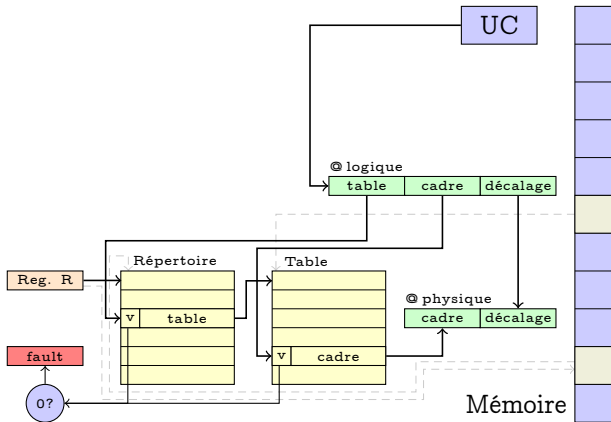
Pagination à deux niveaux



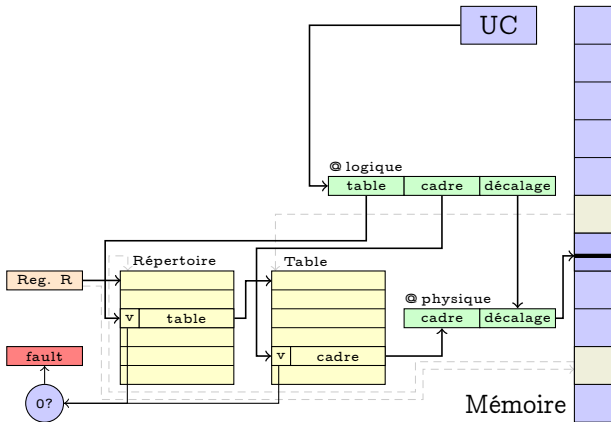
Pagination à deux niveaux



Pagination à deux niveaux



Pagination à deux niveaux



Système 32bits

- Adresses sur 32bits et pages de 4ko ($m = 12$)

Système 32bits

- Adresses sur 32bits et pages de 4ko ($m = 12$)

Pagination à 1 niveau

Table de pages = 2^n lignes de n bits ($n = 20$)

Total = $2^{20} \times 20 \div 8 = 2.5\text{Mo}$ par processus

Système 32bits

- Adresses sur 32bits et pages de 4ko ($m = 12$)

Pagination à 1 niveau

Table de pages = 2^n lignes de n bits ($n = 20$)

Total = $2^{20} \times 20 \div 8 = 2.5\text{Mo}$ par processus

Pagination à 2 niveaux

$n_1 = 10, n_2 = 10, m = 12$

- Répertoire = 2^{10} lignes de 32bits = 4Ko
- 1 table de pages = 2^{10} lignes de 20bits = 2.5Ko

6.5Ko à 2.5Mo par processus<*3->

Mémoire virtuelle

Problème

- Plusieurs centaines de processus sur le système
- Certains sont très gros (plusieurs Go)

Somme des tailles des processus $>$ Capacité RAM

Problème

- Plusieurs centaines de processus sur le système
- Certains sont très gros (plusieurs Go)

Somme des tailles des processus $>$ Capacité RAM

Données inutilisées

- Code de gestion d'erreurs
- Gros tableau : pas tout en même temps
- Bibliothèque : très variable

Ne charger que ce qui est utile !

Mémoire virtuelle

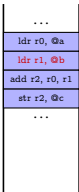
Chaque processus peut adresser plus d'espace qu'il n'y a effectivement de mémoire physique.

Avantages

- Plus de processus en parallèle
- Moins de soucis de gestion de la mémoire

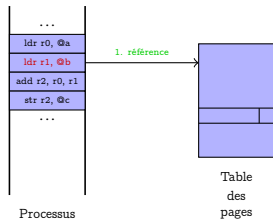
Pagination à la demande

- Table des pages $V = 0$ page sur le disque
- L'OS charge les pages manquantes si nécessaire

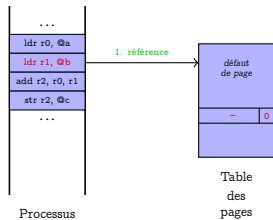


Processus

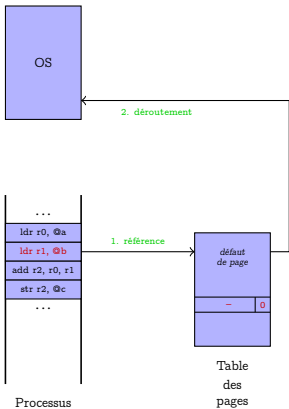
Pagination à la demande



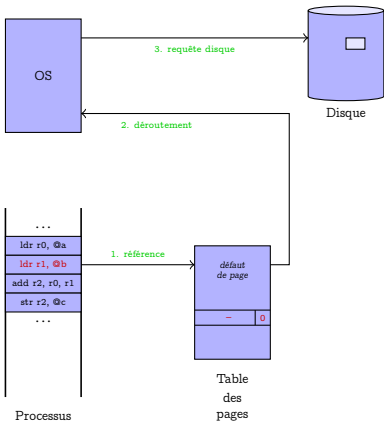
Pagination à la demande



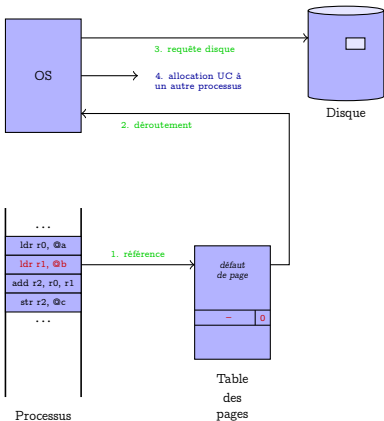
Pagination à la demande



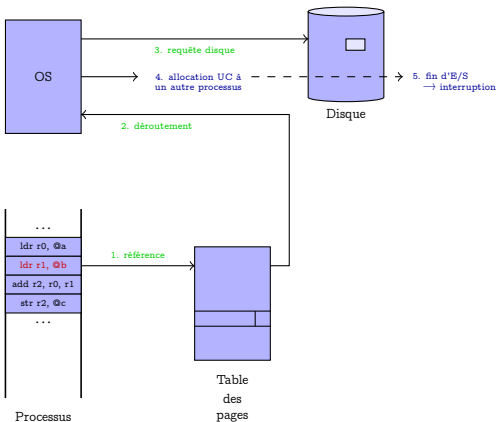
Pagination à la demande



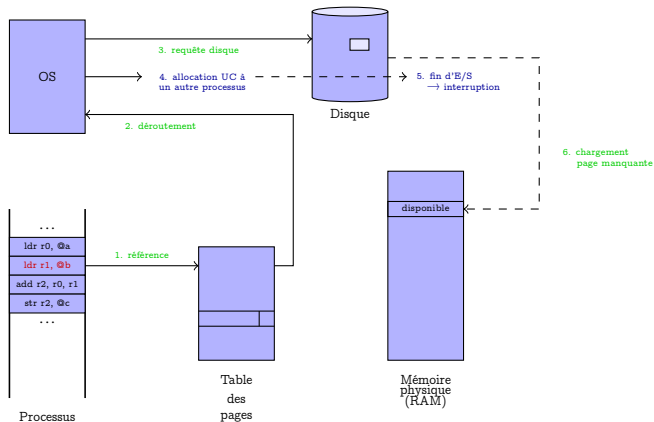
Pagination à la demande



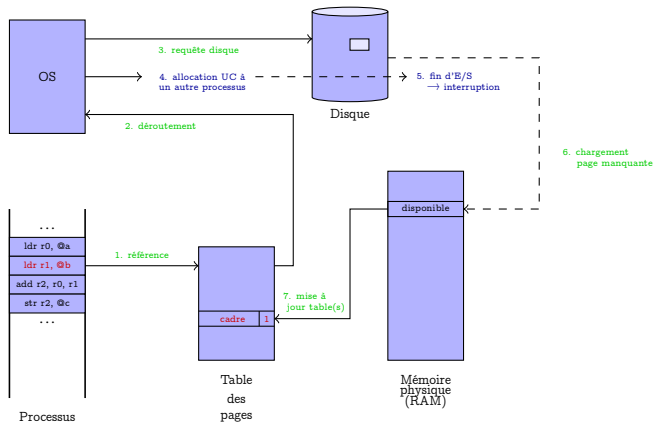
Pagination à la demande



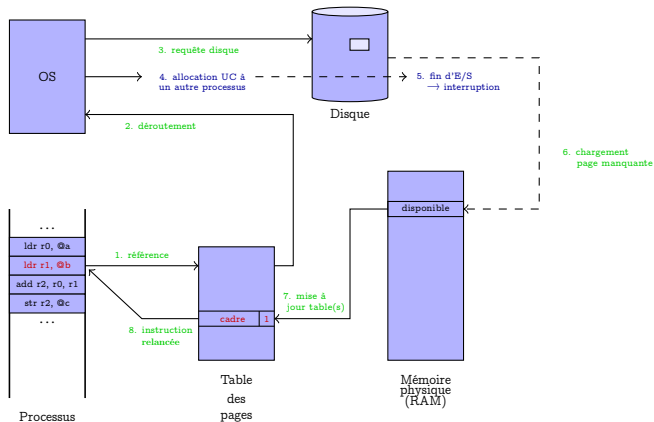
Pagination à la demande



Pagination à la demande



Pagination à la demande



Coût des défauts de pages

- p = probabilité de défaut de page
- M = temps d'accès à la mémoire
- D = temps de traitement du défaut

$$\text{Temps d'accès} = (1-p) \times M + p \times D = M + p(D-M)$$

Coût des défauts de pages

- p = probabilité de défaut de page
- M = temps d'accès à la mémoire
- D = temps de traitement du défaut

$$\text{Temps d'accès} = (1-p) \times M + p \times D = M + p(D-M)$$

Propriété

Le temps d'accès à la mémoire est, en moyenne, proportionnel à la probabilité de défaut de page.

Example

32bits

- Cadres/Pages de 4Ko ($m = 12$)
- Pagination à 2 niveaux ($n_1 = n_2 = 10$)

64bits

- Adresses virtuelles sur 48bits
- Cadre/Pages de 4Ko ($m = 12$)
- Pagination à 4 niveaux ($n_{1,...,4} = 9$)
- Adresses physiques sur 52bits

Plus de détails au prochain cours et en TD...