

Mémoire 1/2

Thomas Lavergne
lavergne@lisn.fr

Multi-programmation

Plusieurs processus souhaitent occuper la mémoire
en même temps.

Multi-programmation

Plusieurs processus souhaitent occuper la mémoire *en même temps*.

Objectifs

- Partager la mémoire sans conflits
- Protéger les données de chaque processus
- Conserver des performances acceptables

Principe

Un seul processus présent en mémoire à la fois, les autres attendent sur le disque.

Principe

Un seul processus présent en mémoire à la fois, les autres attendent sur le disque.

Ordonnancement

- Sauvegarder le processus courant sur le disque
- Restaurer le nouveau depuis le disque

Principe

Un seul processus présent en mémoire à la fois, les autres attendent sur le disque.

Ordonnancement

- Sauvegarder le processus courant sur le disque
- Restaurer le nouveau depuis le disque

Bilan

- ✓ Partage et protège parfaitement
- ✗ Très lent !

Allocation contigüe

Principe

Multi-programmation et temps partagé

- Les processus n'utilisent pas toute la mémoire
- Placer **tous** les processus en mémoire

Mémoire



Principe

Multi-programmation et temps partagé

- Les processus n'utilisent pas toute la mémoire
- Placer **tous** les processus en mémoire

Mémoire

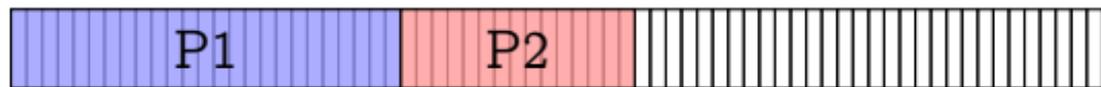


Principe

Multi-programmation et temps partagé

- Les processus n'utilisent pas toute la mémoire
- Placer **tous** les processus en mémoire

Mémoire



Problème

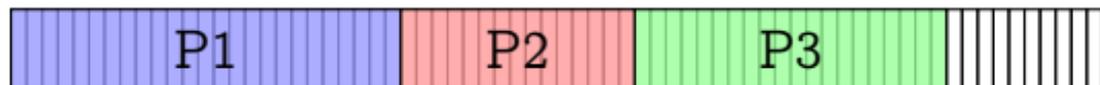
L'adresse de base des processus n'est pas fixe

Principe

Multi-programmation et temps partagé

- Les processus n'utilisent pas toute la mémoire
- Placer **tous** les processus en mémoire

Mémoire



Problème

L'adresse de base des processus n'est pas fixe

Principe

Multi-programmation et temps partagé

- Les processus n'utilisent pas toute la mémoire
- Placer **tous** les processus en mémoire

Mémoire



Problème

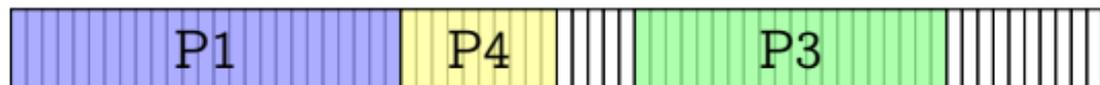
Des trous apparaissent : c'est la fragmentation

Principe

Multi-programmation et temps partagé

- Les processus n'utilisent pas toute la mémoire
- Placer **tous** les processus en mémoire

Mémoire



Problème

Des trous apparaissent : c'est la fragmentation

Principe

Multi-programmation et temps partagé

- Les processus n'utilisent pas toute la mémoire
- Placer **tous** les processus en mémoire

Mémoire



Problème

Où placer les processus ?

Principe

Multi-programmation et temps partagé

- Les processus n'utilisent pas toute la mémoire
- Placer **tous** les processus en mémoire

Mémoire



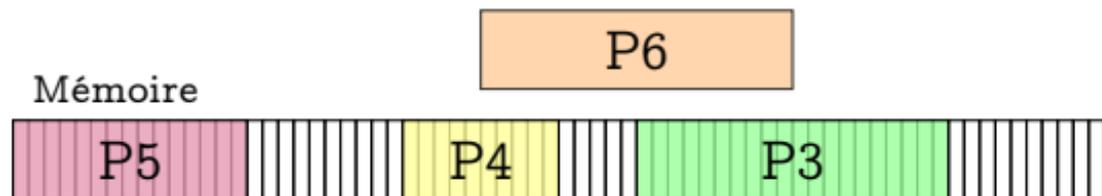
Problème

Où placer les processus ?

Principe

Multi-programmation et temps partagé

- Les processus n'utilisent pas toute la mémoire
- Placer **tous** les processus en mémoire



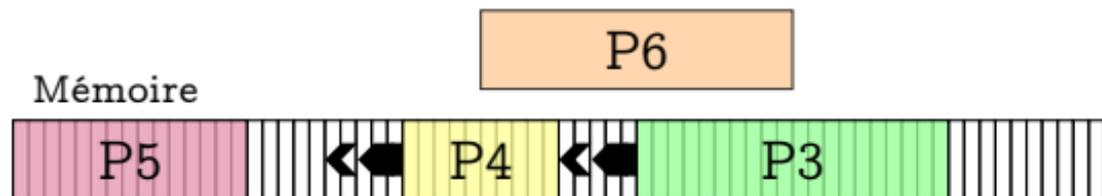
Problème

Trous trop petits : il faut défragmenter

Principe

Multi-programmation et temps partagé

- Les processus n'utilisent pas toute la mémoire
- Placer **tous** les processus en mémoire



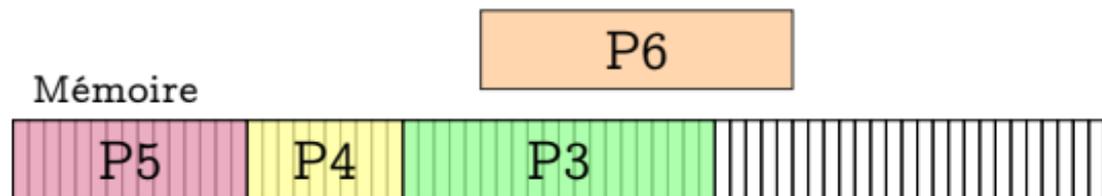
Problème

Trous trop petits : il faut défragmenter

Principe

Multi-programmation et temps partagé

- Les processus n'utilisent pas toute la mémoire
- Placer **tous** les processus en mémoire



Problème

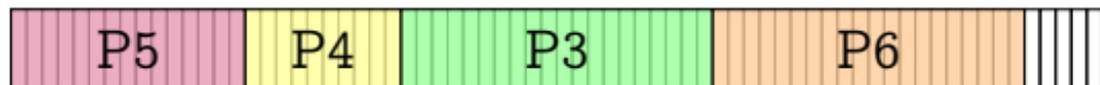
Trous trop petits : il faut défragmenter

Principe

Multi-programmation et temps partagé

- Les processus n'utilisent pas toute la mémoire
- Placer **tous** les processus en mémoire

Mémoire



Problème

Trous trop petits : il faut défragmenter

Stratégie *first-fit*

✓ Très simple

✗ Pas terrible dans tous les cas...

Stratégie *first-fit*

- ✓ Très simple
- ✗ Pas terrible dans tous les cas...

Stratégie *best-fit*

- ✓ Conserve de gros blocs pour de gros processus
- ✗ Fragmente rapidement la mémoire

Où placer les processus ?

Stratégie *first-fit*

- ✓ Très simple
- ✗ Pas terrible dans tous les cas...

Stratégie *best-fit*

- ✓ Conserve de gros blocs pour de gros processus
- ✗ Fragmente rapidement la mémoire

Stratégie *worse-fit*

- ✓ Fragmente plus lentement la mémoire
- ✗ Les gros blocs disparaissent rapidement

Problème:

Quelles sont les adresses des variables ?

Problème:

Quelles sont les adresses des variables ?

```
int a = 3;  
a = a + 1;
```

Problème:

Quelles sont les adresses des variables ?

```
int a = 3;  
a = a + 1;
```

@ symboliques :

```
load a, r1  
add #1, r1  
store r1, a
```

Problème:

Quelles sont les adresses des variables ?

```
int a = 3;  
a = a + 1;
```

@ symboliques :

```
load a, r1  
add #1, r1  
store r1, a
```

@ mémoire :

```
load 2B10, r1  
add #1, r1  
store r1, 2B10
```

Problème:

Quelles sont les adresses des variables ?

```
int a = 3;  
a = a + 1;
```

@ symboliques :

```
load a, r1  
add #1, r1  
store r1, a
```

@ mémoire :

```
load 2B10, r1  
add #1, r1  
store r1, 2B10
```

Édition de liens

Qui doit s'en charger ? Le compilateur ou l'OS ?

Par le compilateur

- ✓ Traduction efficace des adresses
- ✗ Adresse de base fixée à la compilation

Par le compilateur

- ✓ Traduction efficace des adresses
- ✗ Adresse de base fixée à la compilation

Par le système d'exploitation

- ✓ Adresse de base dynamique
- ✗ Édition de liens très coûteuse

Par le compilateur

- ✓ Traduction efficace des adresses
- ✗ Adresse de base fixée à la compilation

Combiner les deux

- ✓ Compilateur : adresse de base fixe
- ✓ Système : décalage dynamique des adresses
(avec l'aide de la MMU)

Par le système d'exploitation

- ✓ Adresse de base dynamique
- ✗ Édition de liens très coûteuse

Principe

Chaque processus reçoit son propre espace de *mémoire logique*.

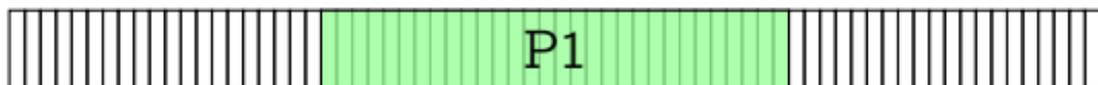
Mémoire physique



Principe

Chaque processus reçoit son propre espace de *mémoire logique*.

Mémoire vue par P1



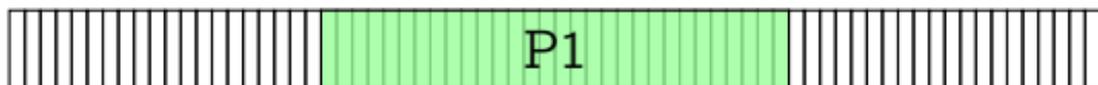
Mémoire physique



Principe

Chaque processus reçoit son propre espace de *mémoire logique*.

Mémoire vue par P1



Mémoire physique

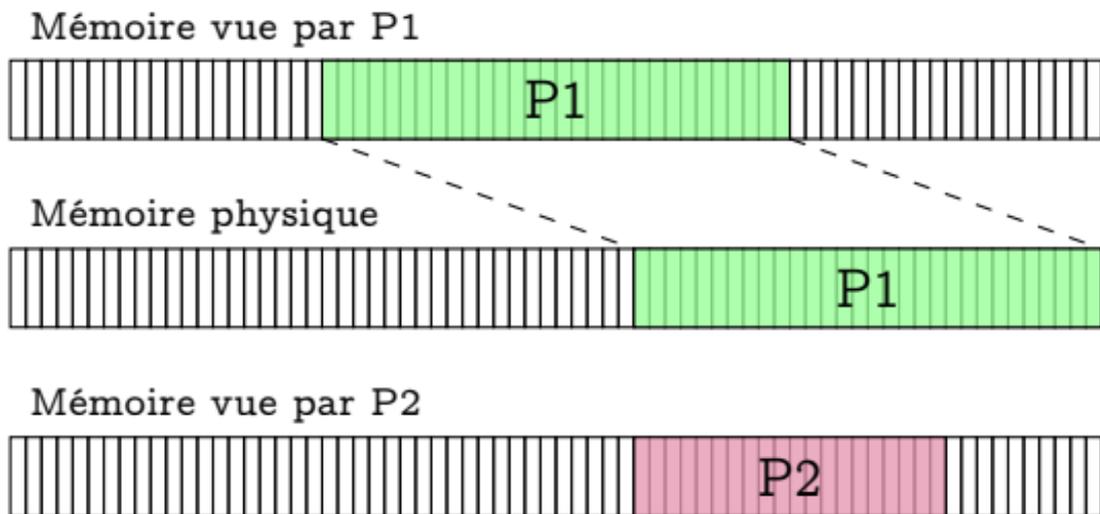


Mémoire vue par P2



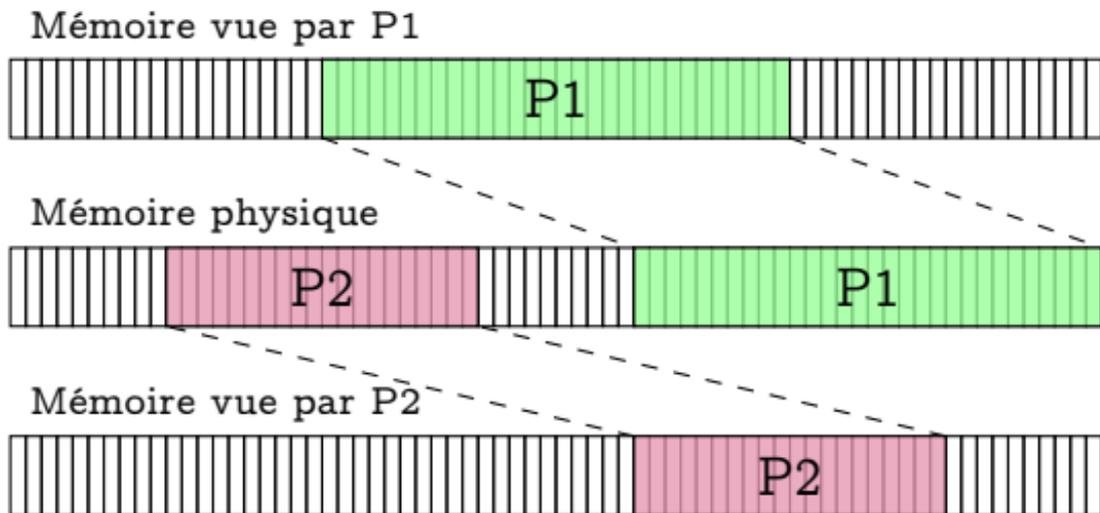
Principe

Chaque processus reçoit son propre espace de *mémoire logique*.



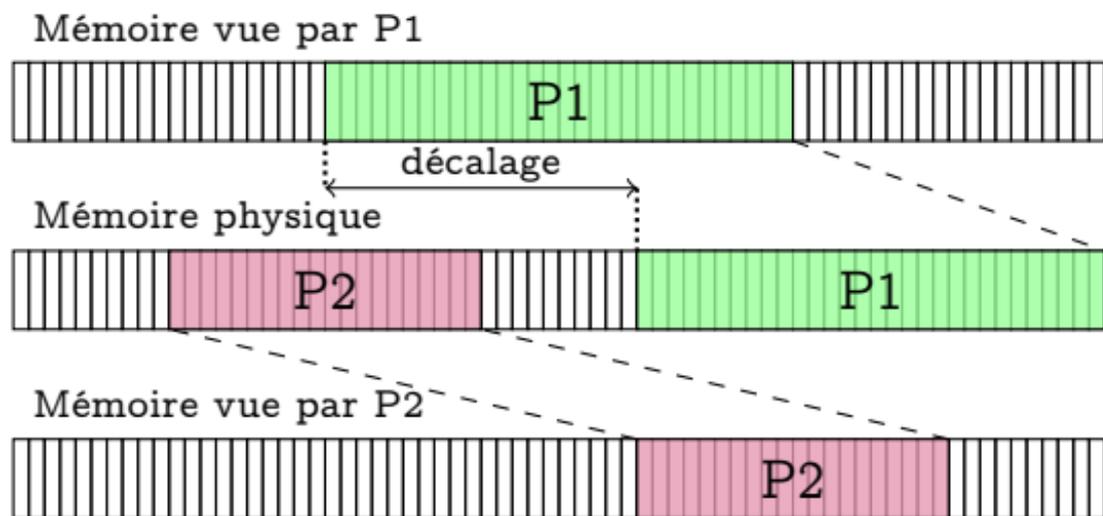
Principe

Chaque processus reçoit son propre espace de *mémoire logique*.



Principe

Chaque processus reçoit son propre espace de *mémoire logique*.



Principe

Chaque processus à son propre **décalage** qui est ajouté à chaque adresse utilisée.

Principe

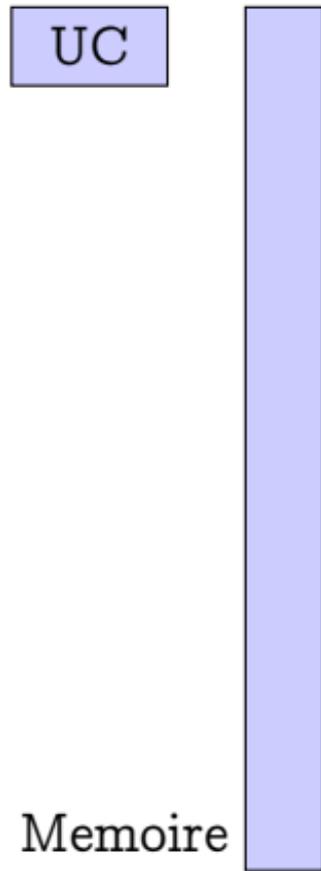
Chaque processus à son propre **décalage** qui est ajouté à chaque adresse utilisée.

Implémentation

La traduction est réalisée par la MMU (*Memory Management Unit*)

- La MMU est configurée par le système
- Les processus utilisent des *adresses logiques*
- La MMU les traduit en *adresses physiques*

UC



Memoire

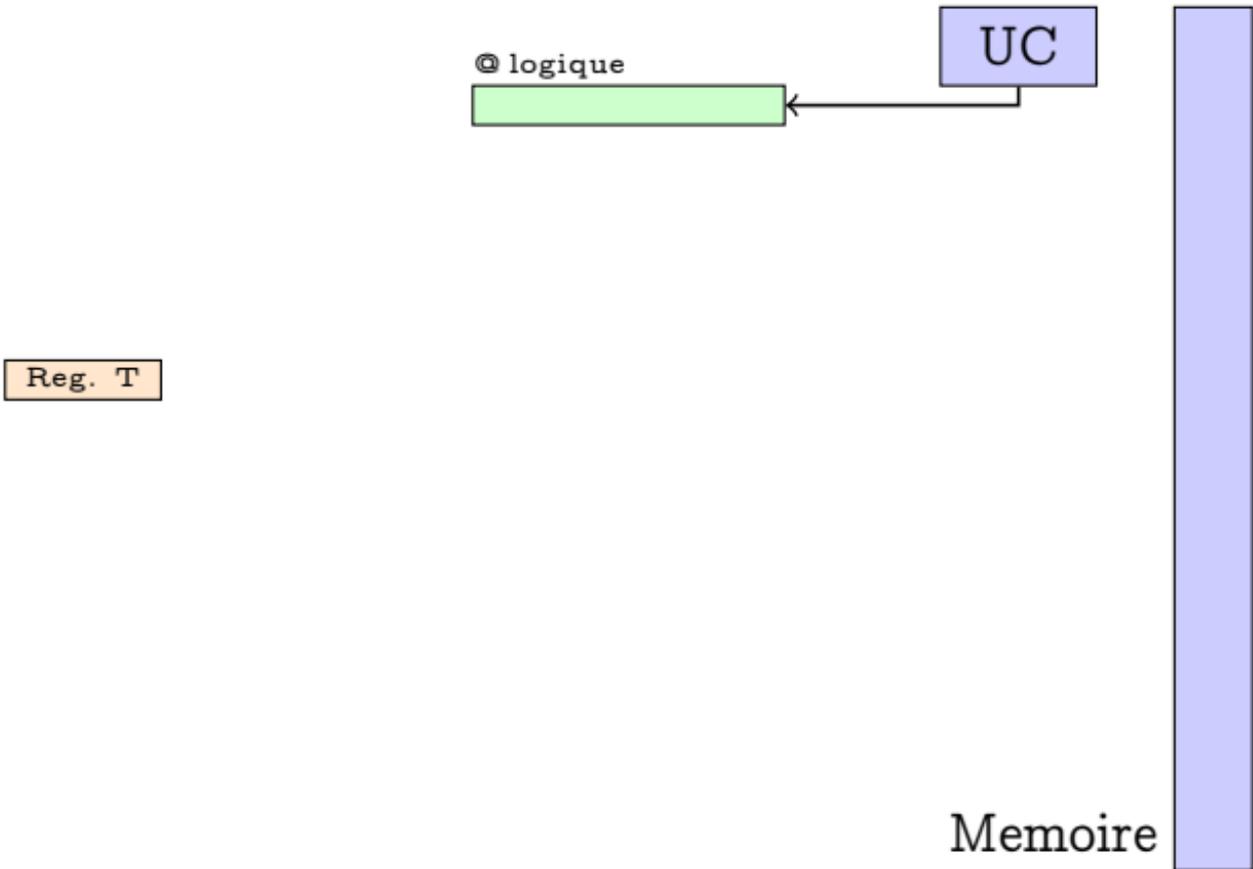
Translation

Reg. T

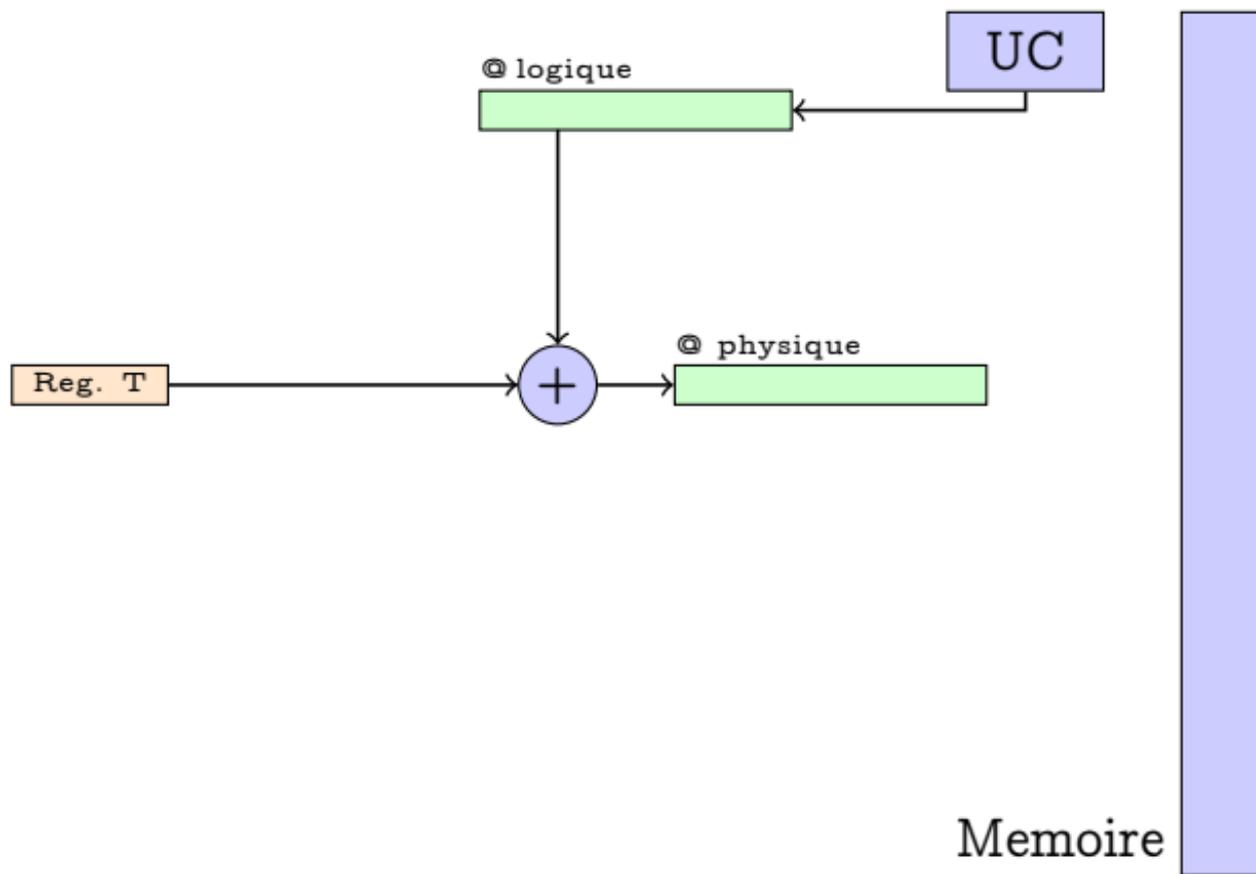
UC

Memoire

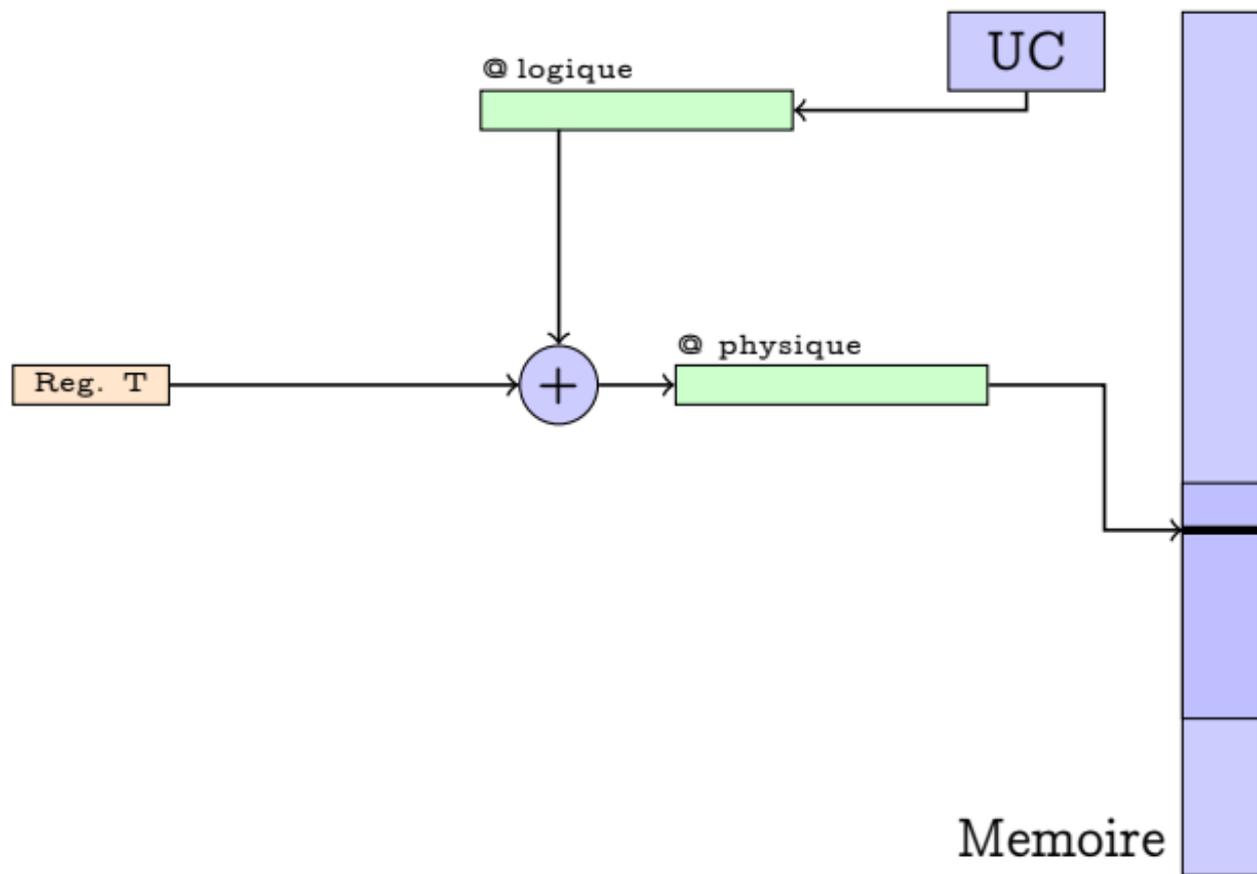
Translation



Translation



Translation



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 processus sont de gros blocs

Problème

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

Source du problème :

les processus sont de gros blocs

Première solution

Découper les processus en plus petits blocs avec multiples translations :

La segmentation

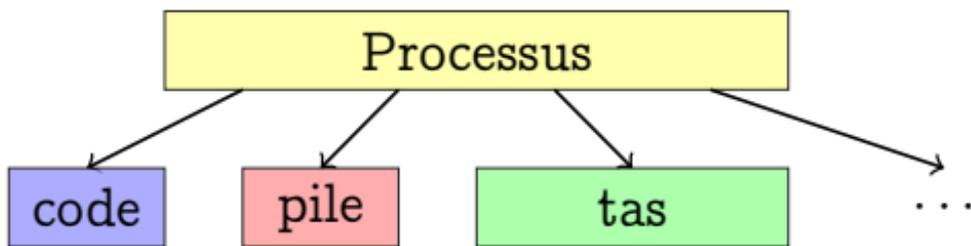
Segmentation

Découpage sémantique

Segments correspondant aux différents types de données : code, pile, tas, ...

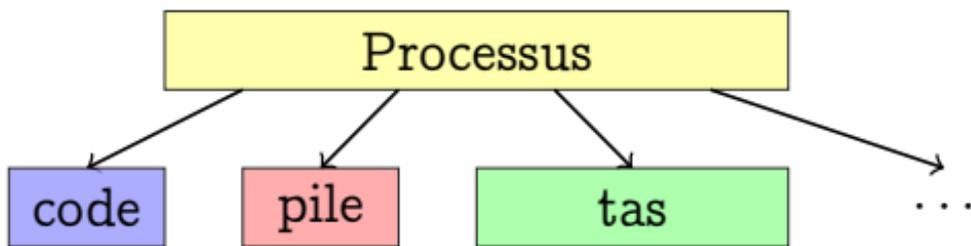
Découpage sémantique

Segments correspondant aux différents types de données : code, pile, tas, ...



Découpage sémantique

Segments correspondant aux différents types de données : code, pile, tas, ...



Segments:

Chaque segment peut être placé indépendamment en mémoire.

Adressage logique

Les segments sont placés en mémoire logique
L'adresse logique se compose d'un sélecteur de segment et d'un décalage dans ce segment.

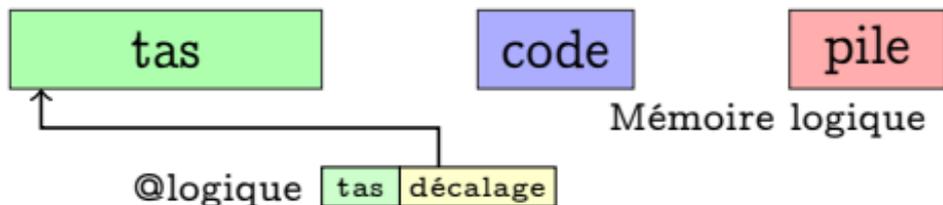
tas

code

pile

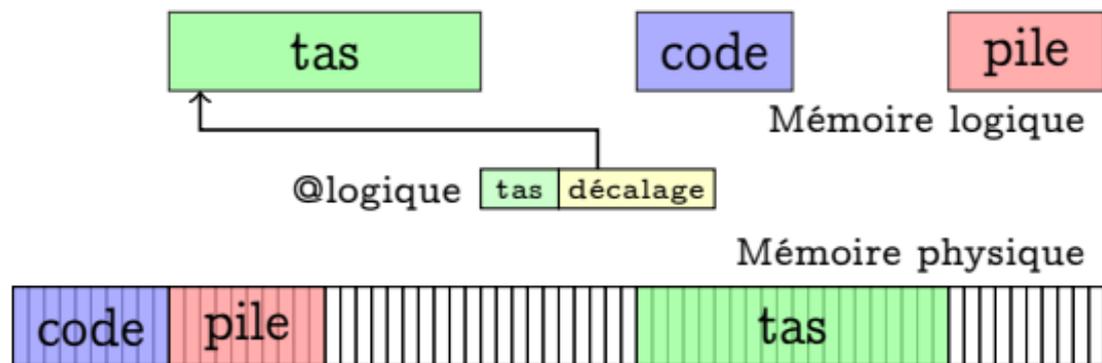
Mémoire logique

Les segments sont placés en mémoire logique
L'adresse logique se compose d'un sélecteur de segment et d'un décalage dans ce segment.



Adressage logique

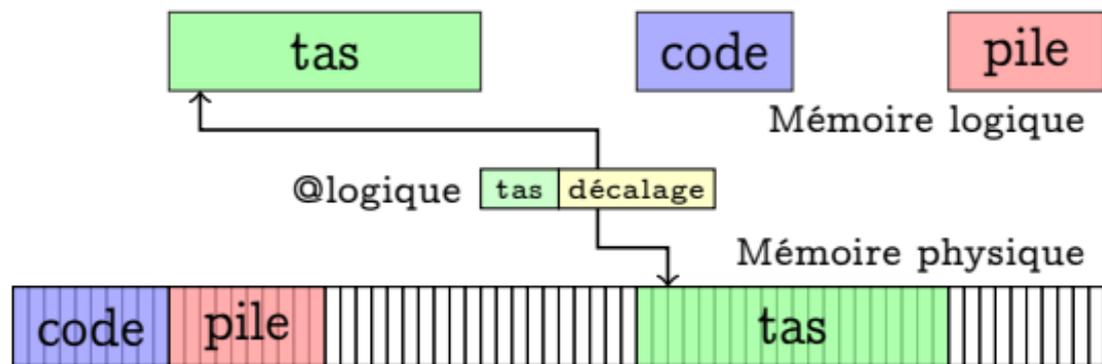
Les segments sont placés en mémoire logique
L'adresse logique se compose d'un sélecteur de segment et d'un décalage dans ce segment.



Adressage logique

Les segments sont placés en mémoire logique

L'adresse logique se compose d'un sélecteur de segment et d'un décalage dans ce segment.



Traduction d'adresse

Appliquer une translation en fonction du segment, au passage vérifier les limites.

Découpage

Le découpage se fait sur la représentation binaire. n bits permettent d'indexer 2^n valeurs.

Découpage

Le découpage se fait sur la représentation binaire. n bits permettent d'indexer 2^n valeurs.

Exemple : @logique 16 bits

8 segments (3 bits) de 8ko maximum (13 bits)

Découpage

Le découpage se fait sur la représentation binaire. n bits permettent d'indexer 2^n valeurs.

Exemple : @logique 16 bits

8 segments (3 bits) de 8ko maximum (13 bits)

0xA13E = 1010 0001 0011 1110

Découpage

Le découpage se fait sur la représentation binaire. n bits permettent d'indexer 2^n valeurs.

Exemple : @logique 16 bits

8 segments (3 bits) de 8ko maximum (13 bits)

0xA13E = 1010 0001 0011 1110

Découpage

Le découpage se fait sur la représentation binaire. n bits permettent d'indexer 2^n valeurs.

Exemple : @logique 16 bits

8 segments (3 bits) de 8ko maximum (13 bits)

0xA13E = 1010 0001 0011 1110

Segment 5 101

Découpage

Le découpage se fait sur la représentation binaire. n bits permettent d'indexer 2^n valeurs.

Exemple : @logique 16 bits

8 segments (3 bits) de 8ko maximum (13 bits)

0xA13E = 1010 0001 0011 1110

Segment 5 101

Décalage 318 0 0001 0011 1110

Découpage

Le découpage se fait sur la représentation binaire. n bits permettent d'indexer 2^n valeurs.

Exemple : @logique 16 bits

8 segments (3 bits) de 8ko maximum (13 bits)

0xA13E = 1010 0001 0011 1110

Segment 5 101

Décalage 318 0 0001 0011 1110

319^{ème} octet du 5^{ème} segment

En mémoire physique

Une table par processus : mémorise la translation et la taille de chaque segments du processus.

En mémoire physique

Une table par processus : mémorise la translation et la taille de chaque segments du processus.

Segments

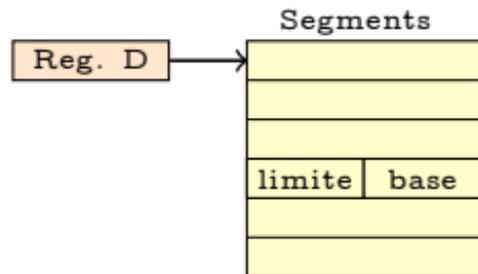
| | |
|--------|------|
| | |
| | |
| | |
| limite | base |
| | |
| | |

Limite : taille

Base : translation

En mémoire physique

Une table par processus : mémorise la translation et la taille de chaque segments du processus.



Limite : taille

Base : translation

Configuration de la MMU

Un registre permet d'indiquer où trouver la table en mémoire physique pour le processus courant.

MMU: Pendant la translation

- Vérification de la validité du segment
- Vérification que l'accès est dans les limites

MMU: Pendant la translation

- Vérification de la validité du segment
- Vérification que l'accès est dans les limites

En cas d'erreur

La MMU provoque un SEGFALT :

Segmentation Fault

- Déclenchement d'une interruption
- Le système reprend la main pour la gérer
- Terminaison du processus fautif

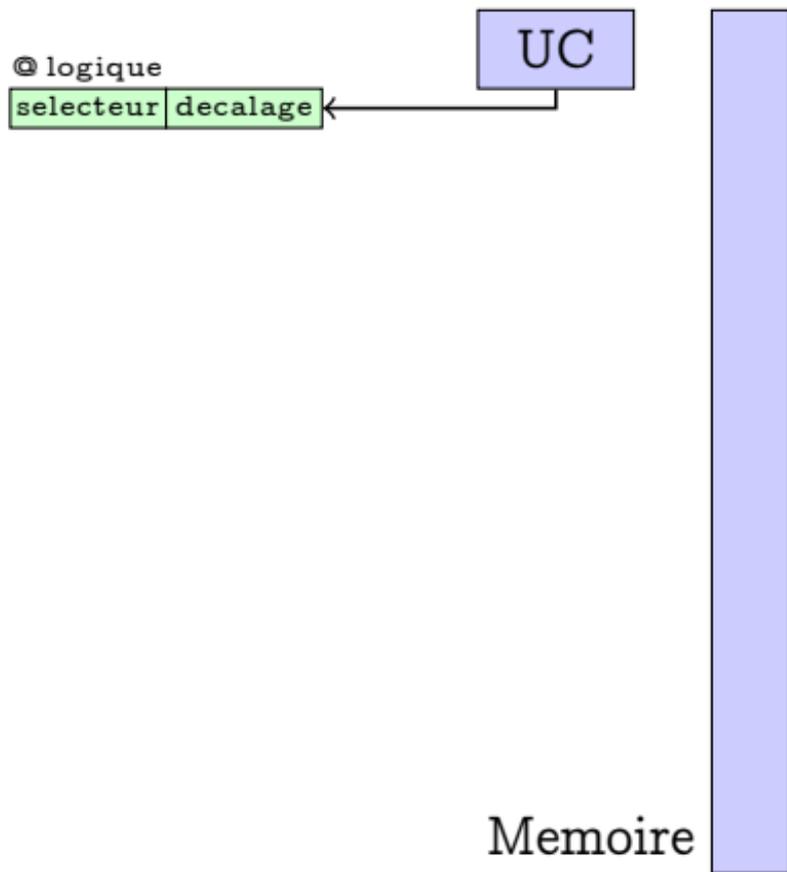
Segmentation

UC

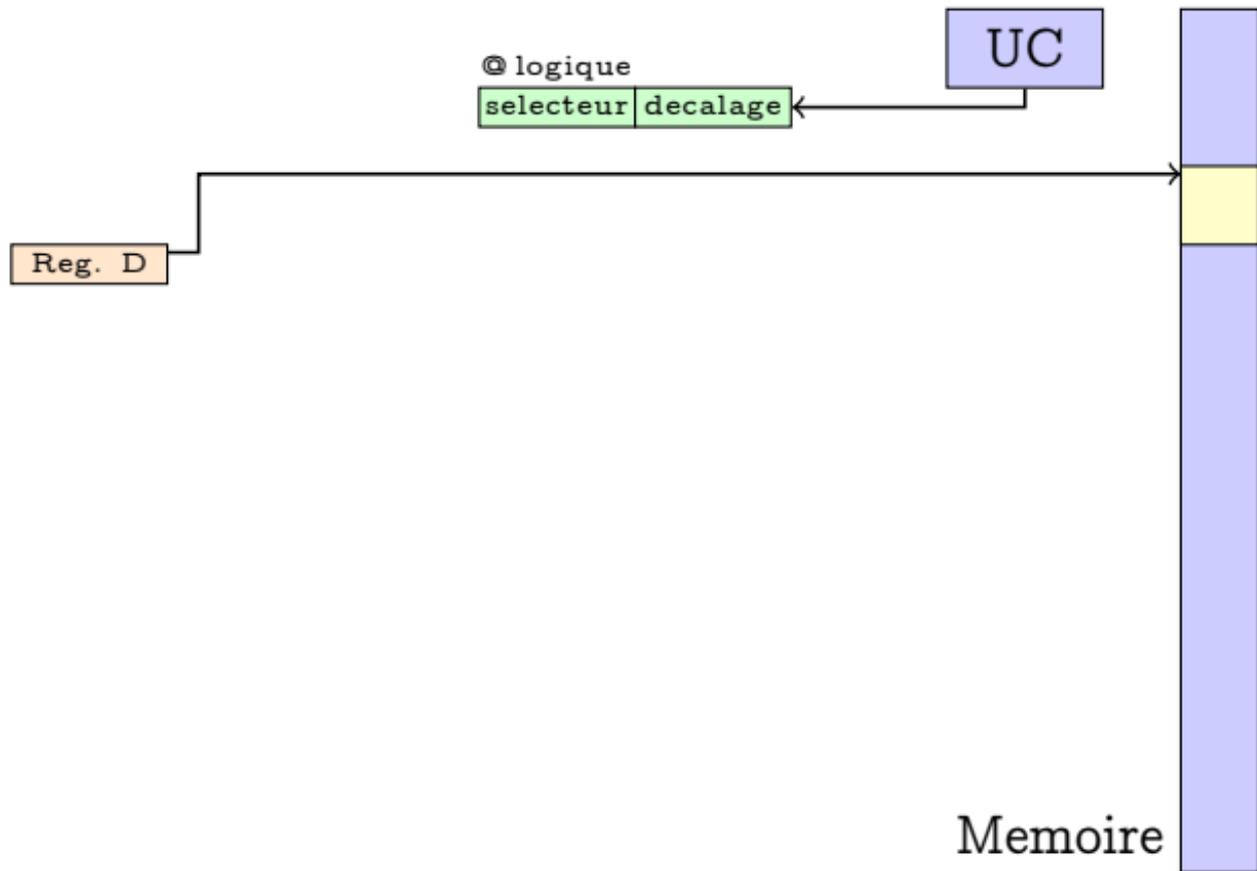


Memoire

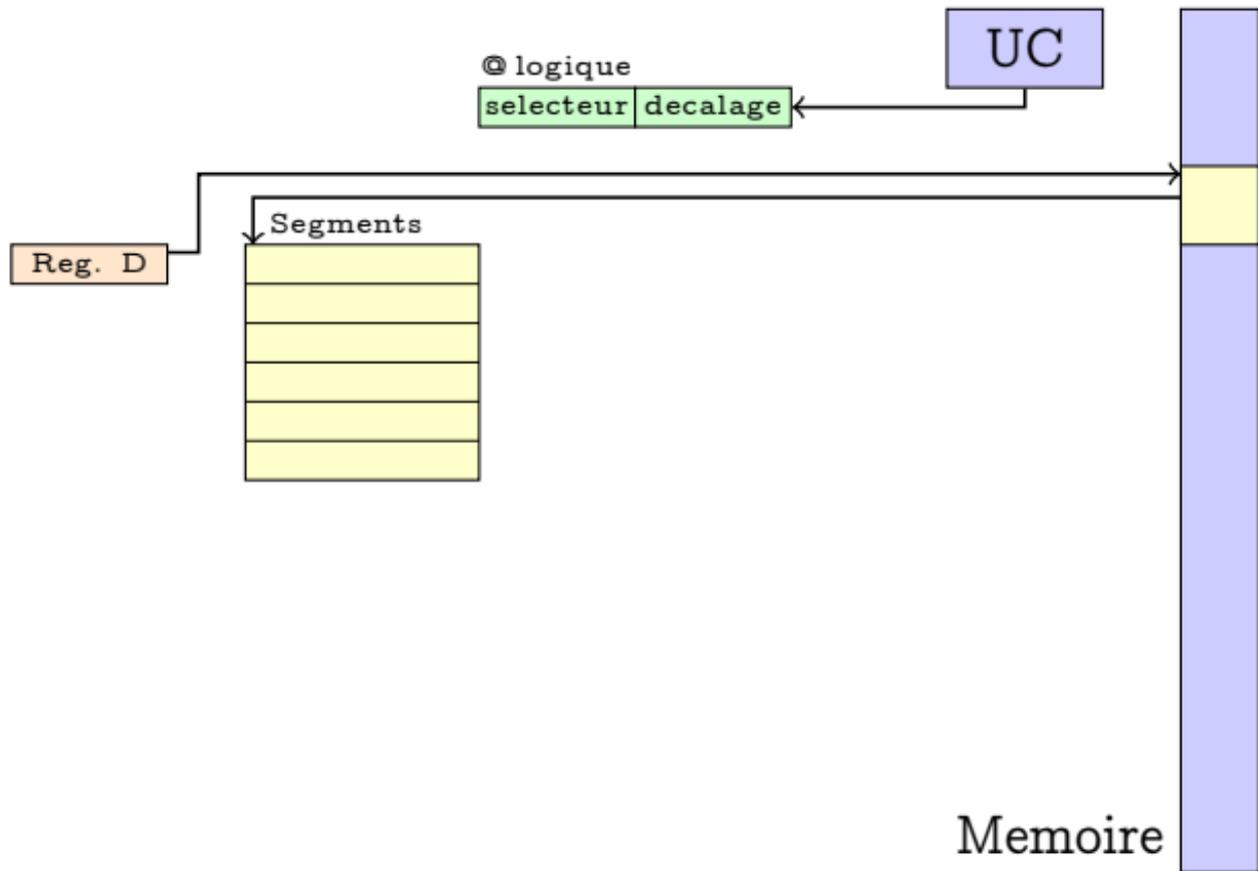
Segmentation



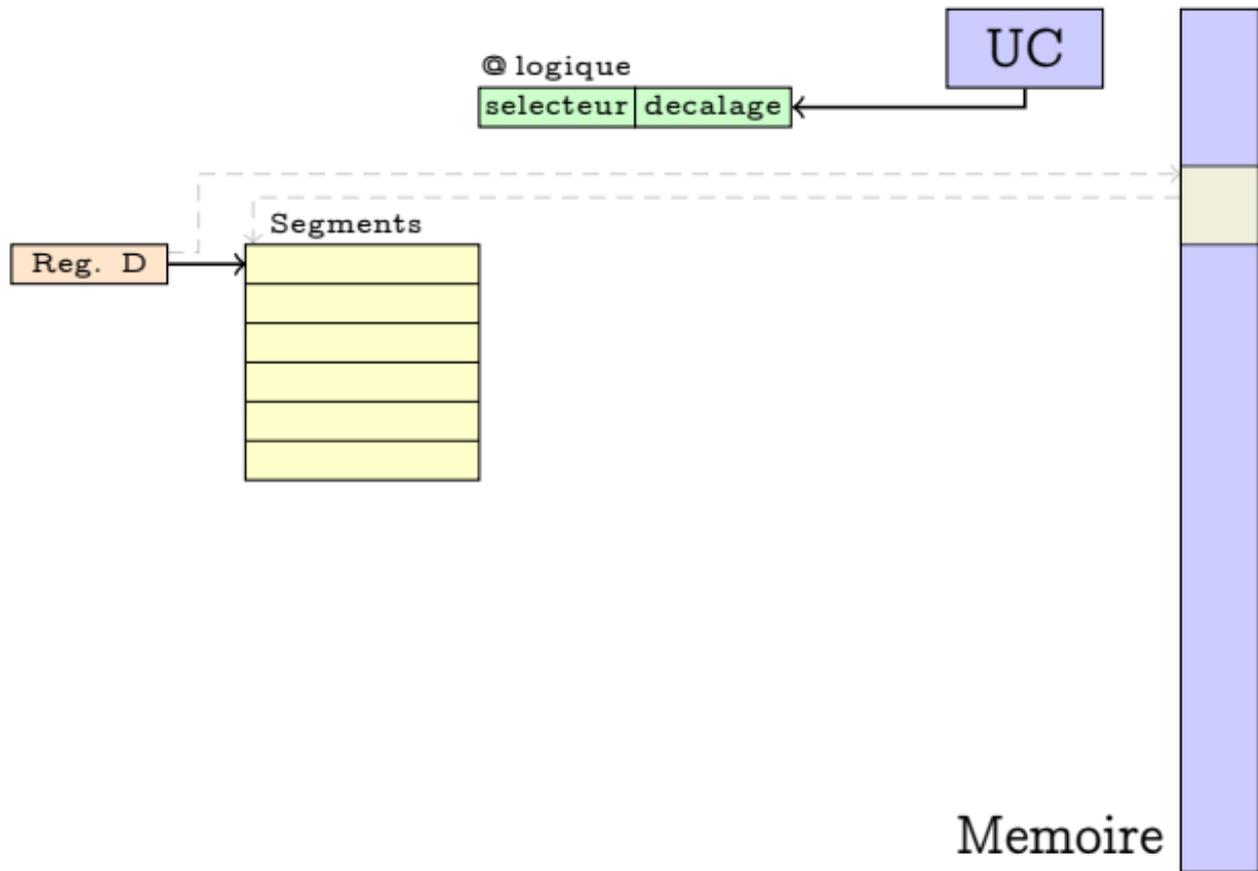
Segmentation



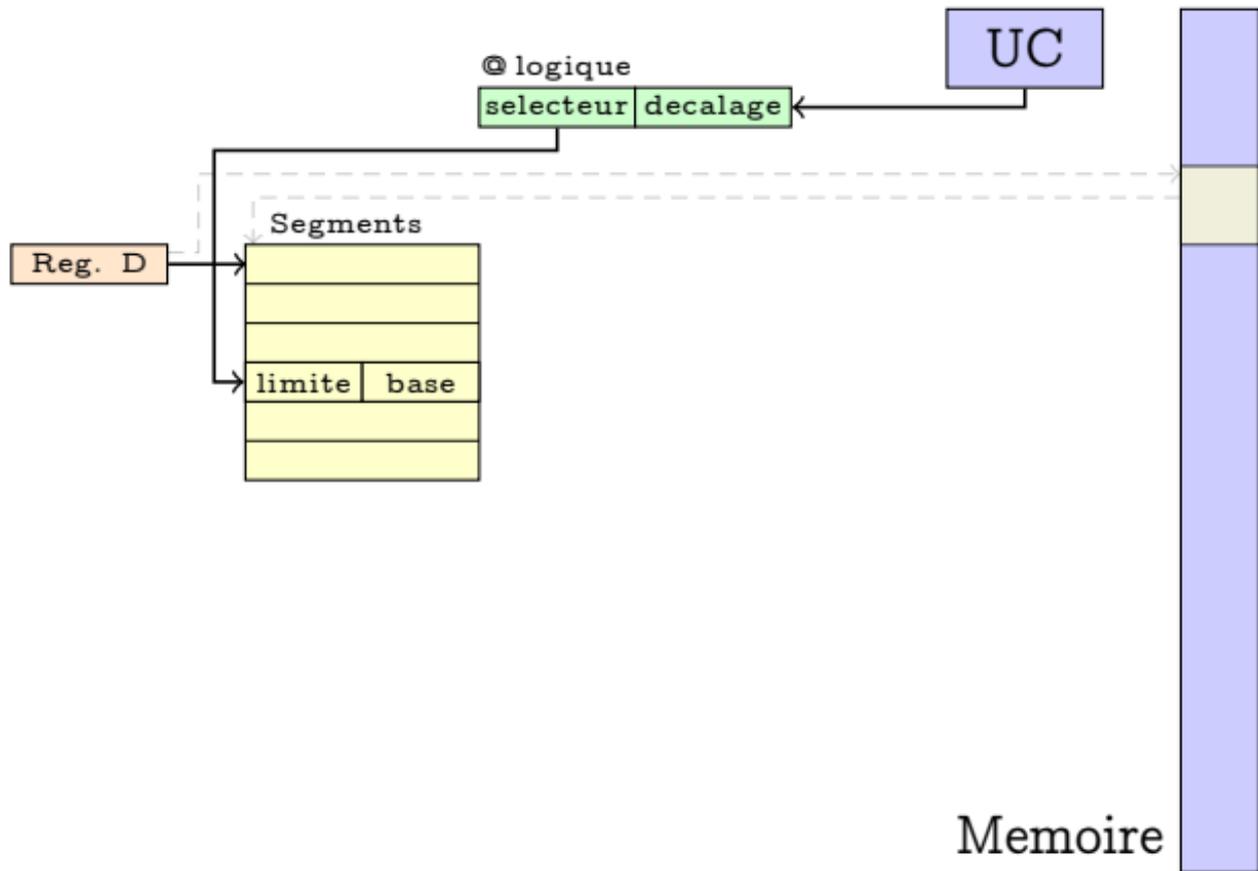
Segmentation



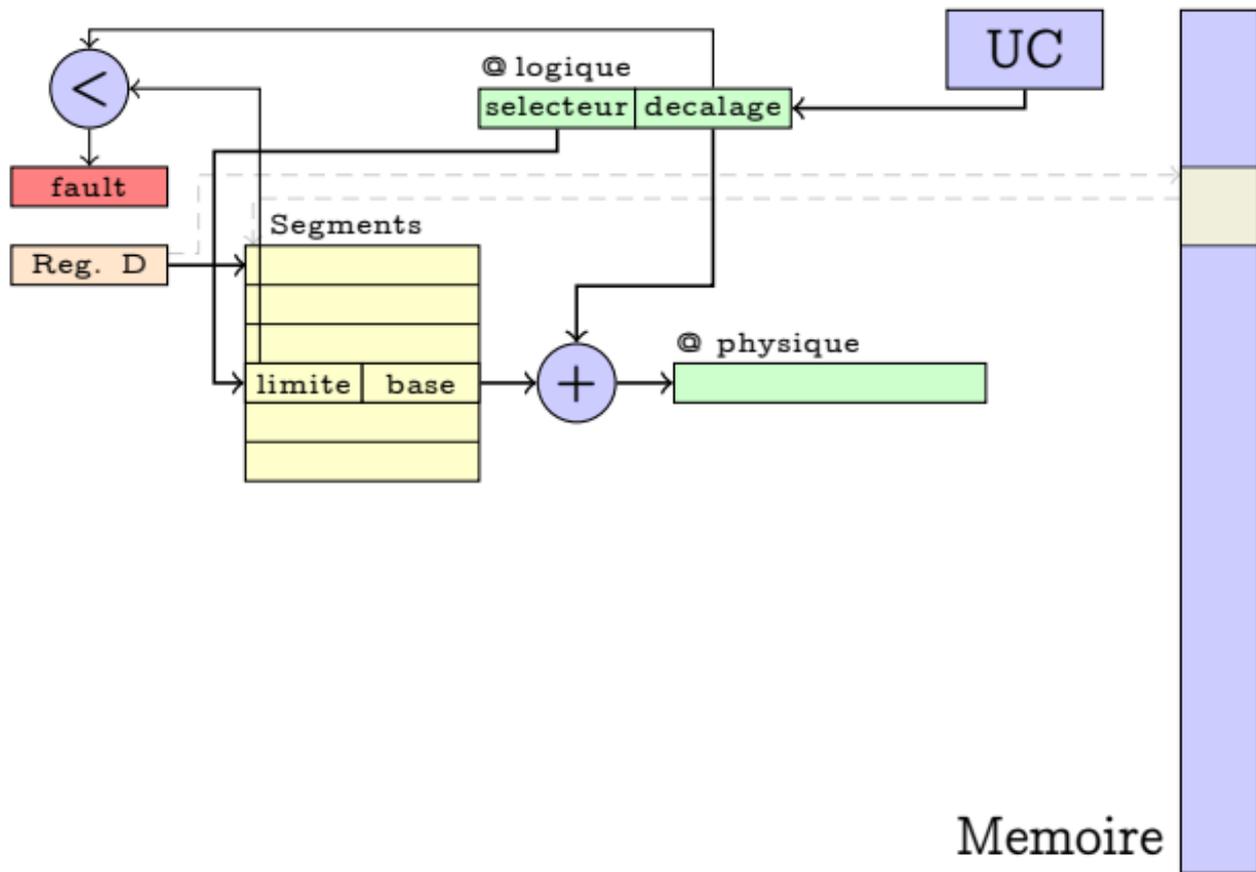
Segmentation



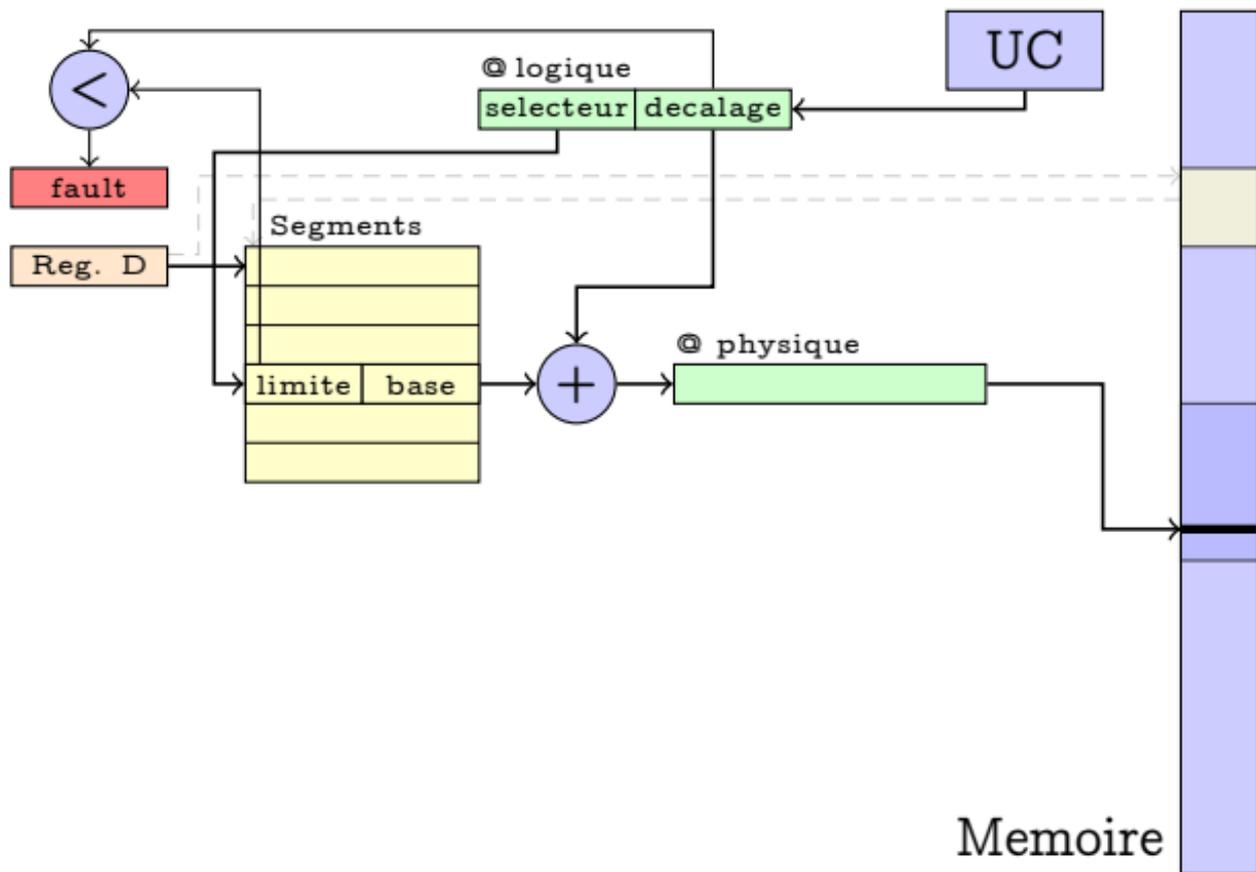
Segmentation



Segmentation



Segmentation



Avantages

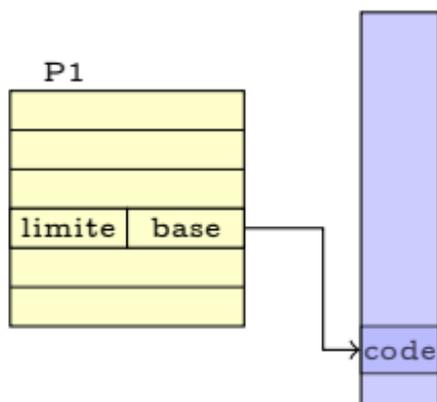
Segments partagés

Un même segment peut-être utilisé par plusieurs processus. Partage de code, de données...



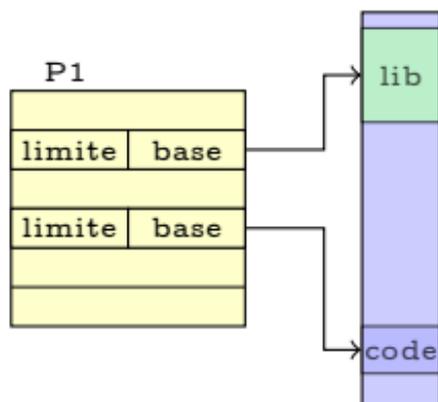
Segments partagés

Un même segment peut-être utilisé par plusieurs processus. Partage de code, de données...



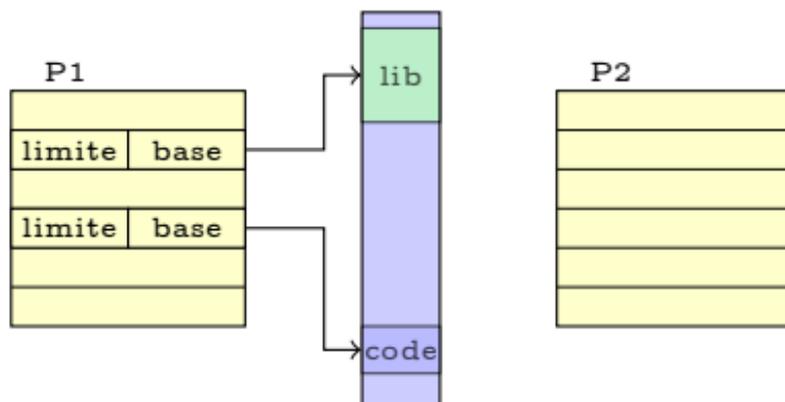
Segments partagés

Un même segment peut-être utilisé par plusieurs processus. Partage de code, de données...



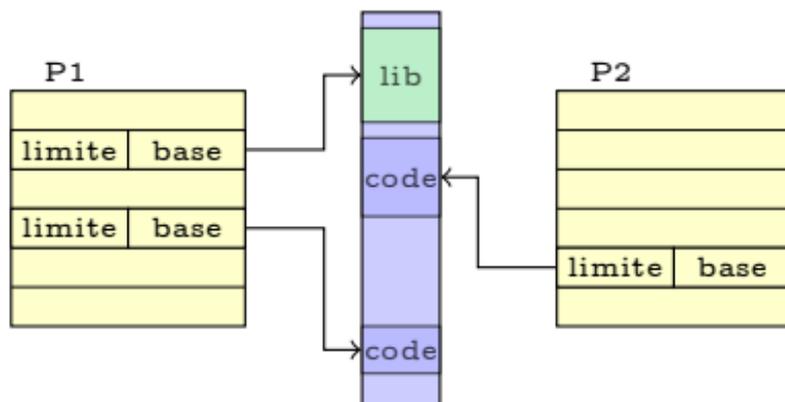
Segments partagés

Un même segment peut-être utilisé par plusieurs processus. Partage de code, de données...



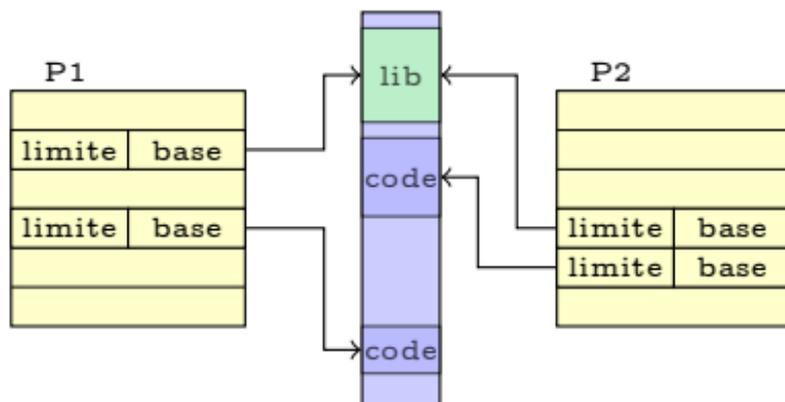
Segments partagés

Un même segment peut-être utilisé par plusieurs processus. Partage de code, de données...



Segments partagés

Un même segment peut-être utilisé par plusieurs processus. Partage de code, de données...

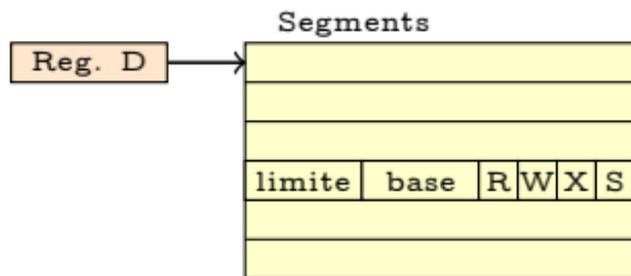


Gestion des droits

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

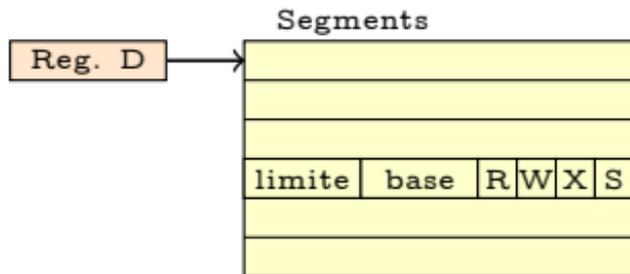
Gestion des droits

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



Gestion des droits

Dans la table des descripteurs, 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...

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
- Placer les **pages** dans les **cadres**

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**

Mémoire



Processus



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}

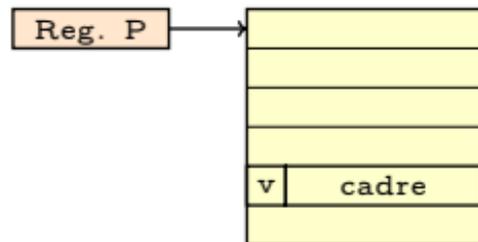
Fragmentation interne

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

→ En moyenne 1/2 cadre perdu par processus

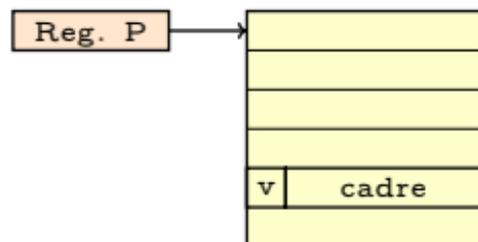
En mémoire physique

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



En mémoire physique

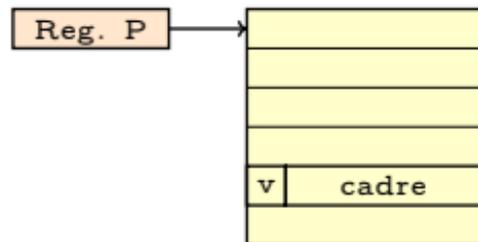
Une table par processus : mémorise le cadre où est stocké chaque page du processus.



Cadre : Correspondance

En mémoire physique

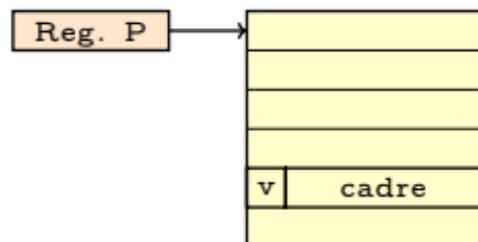
Une table par processus : mémorise le cadre où est stocké chaque page du processus.



Cadre : Correspondance
V : validité

En mémoire physique

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



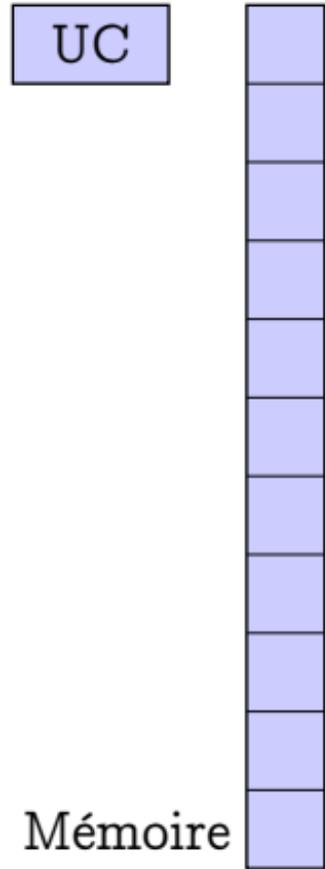
Cadre : Correspondance
V : validité

Couverture

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

Pagination

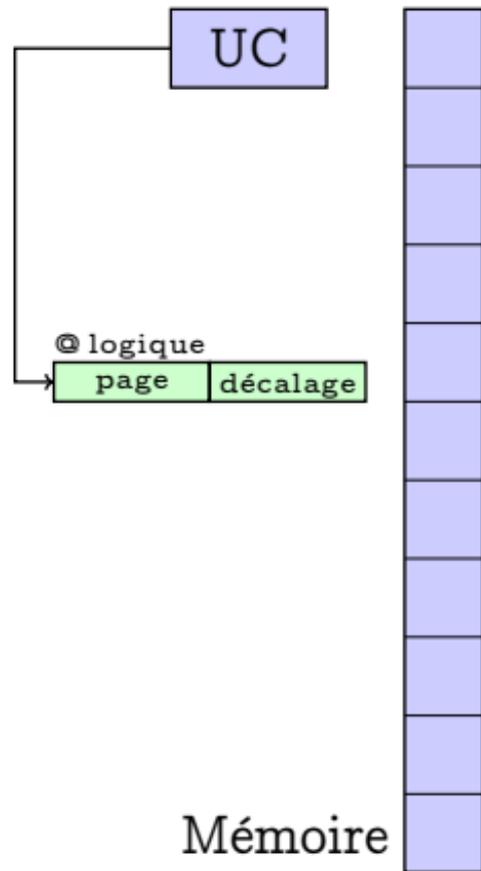
UC



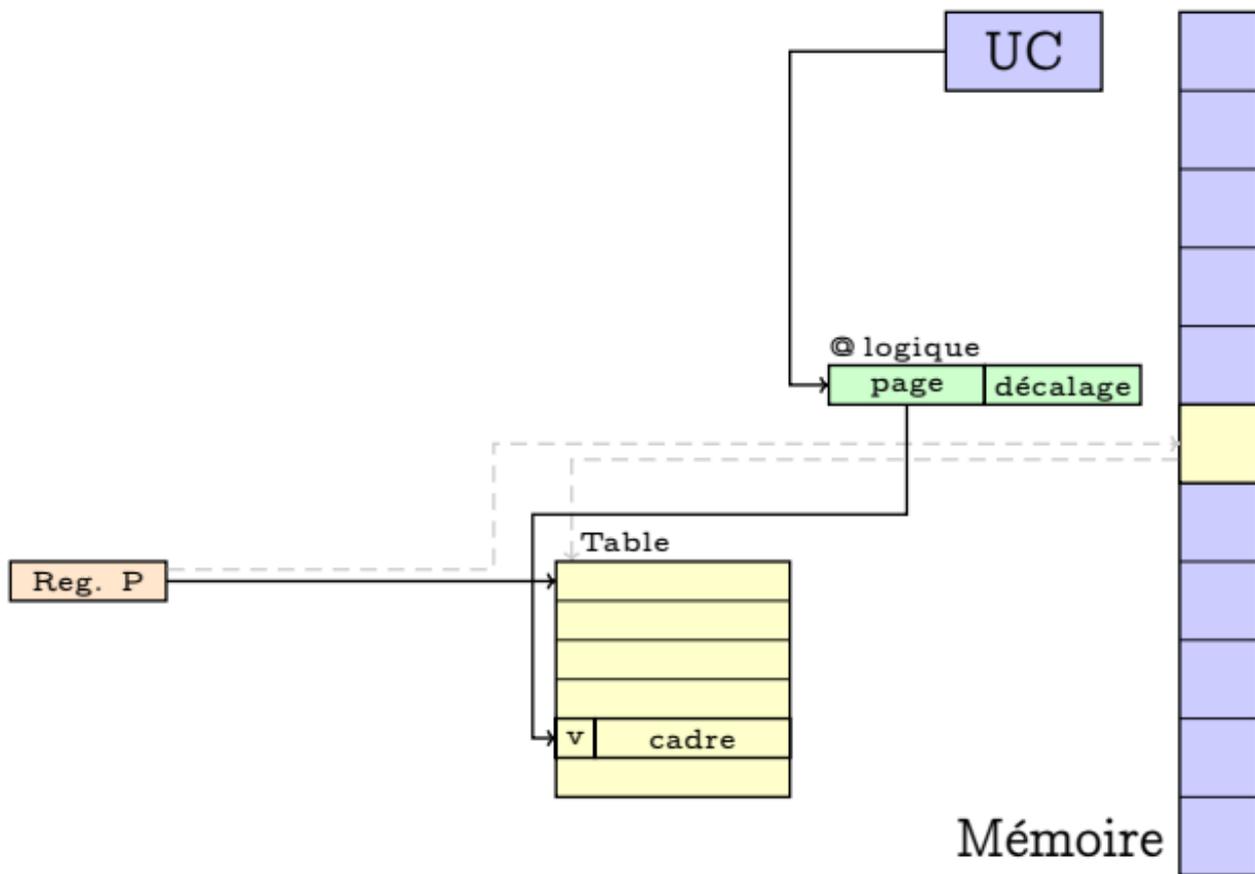
The diagram shows a vertical stack of 12 light blue rectangular cells, representing memory pages. To the left of the top cell is a light blue box containing the text 'UC'. To the left of the bottom cell is the text 'Mémoire'.

Mémoire

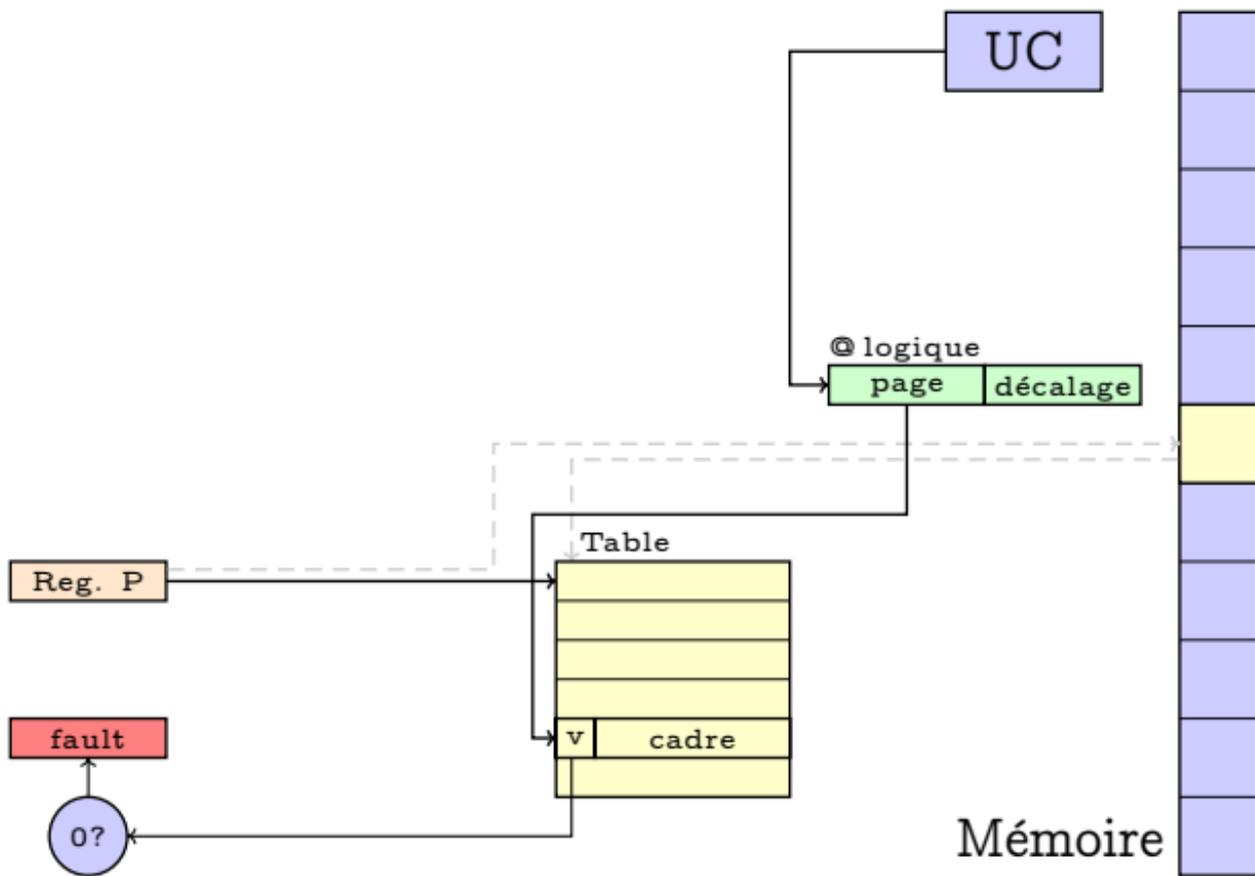
Pagination



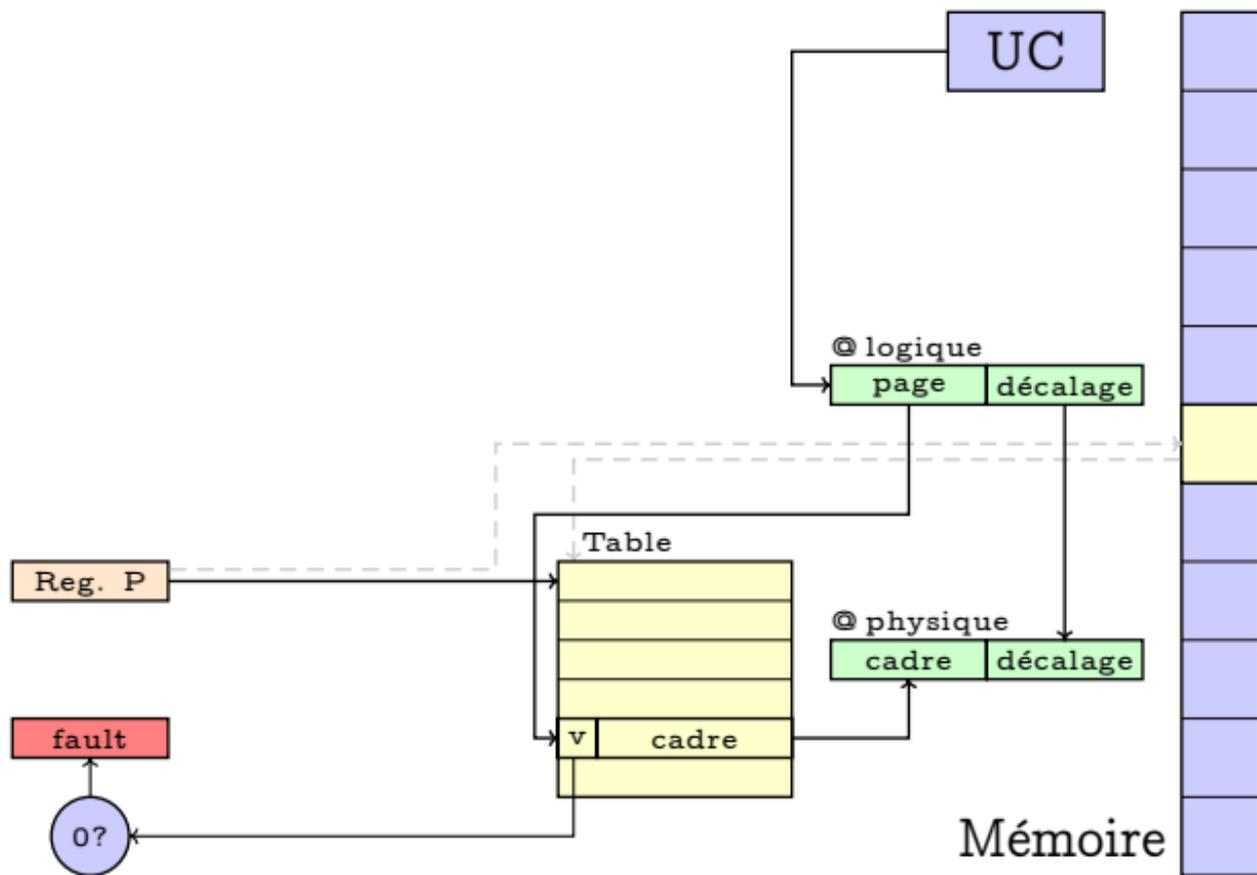
Pagination



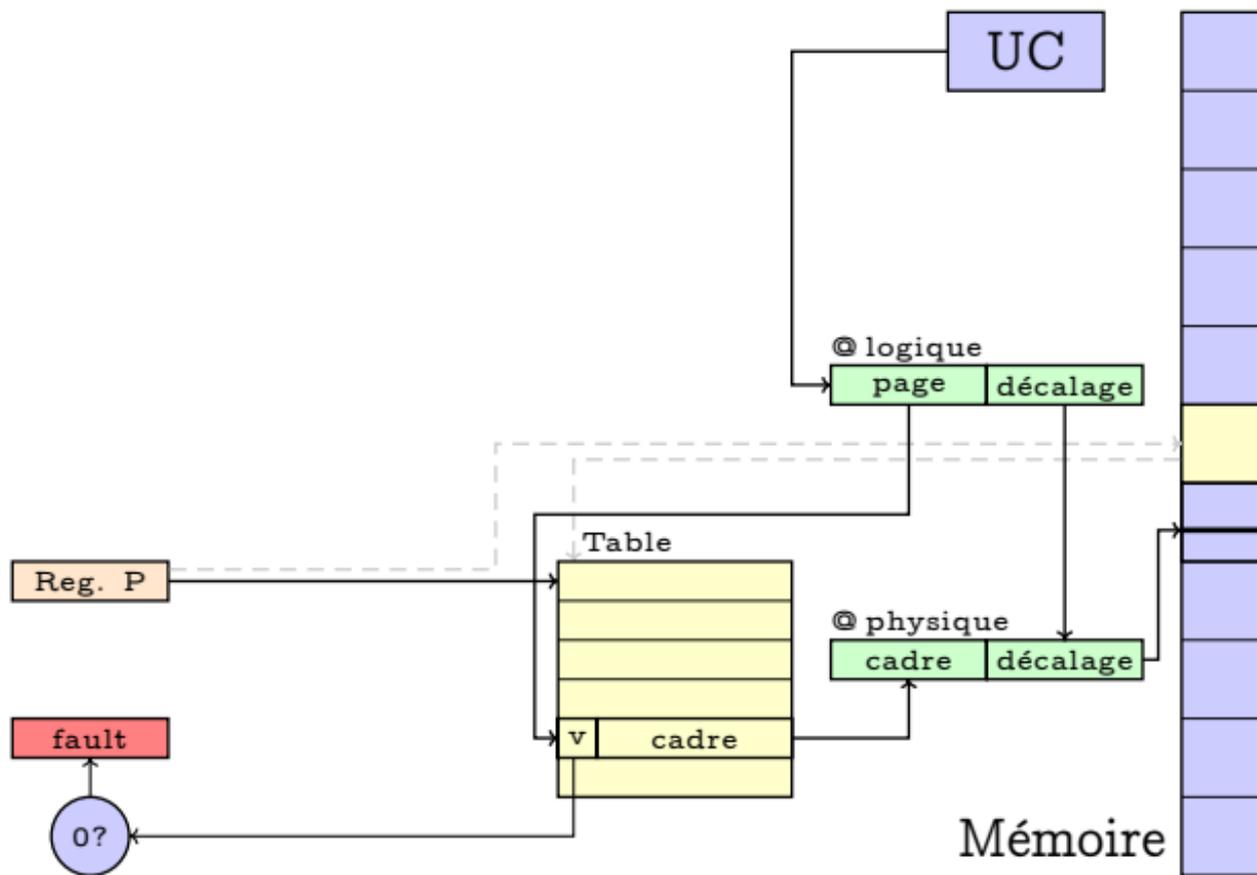
Pagination



Pagination



Pagination

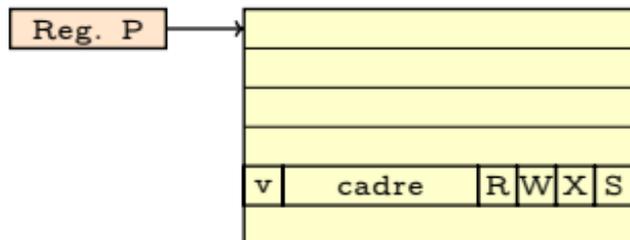


Gestion des droits

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

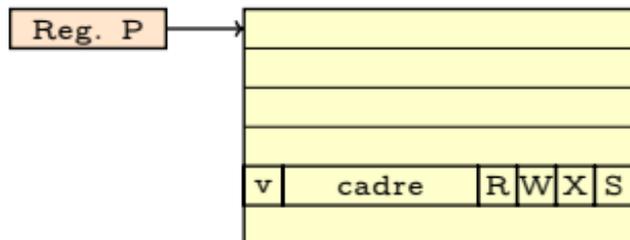
Gestion des droits

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 ($n=20$, $m=12$)
 - o grande table (4Mo par processus)

Grand espace d'adressage (32bits ou plus)

- X Trop de pages ($n=20$, $m=12$)
 - o grande table (4Mo par processus)
- X Pages trop grosses ($n=10$, $m=22$)
 - o fragmentation interne (2Mo par processus)

Grand espace d'adressage (32bits ou plus)

- X Trop de pages ($n=20, m=12$)
 - o grande table (4Mo par processus)
- X Pages trop grosses ($n=10, m=22$)
 - o fragmentation interne (2Mo par processus)

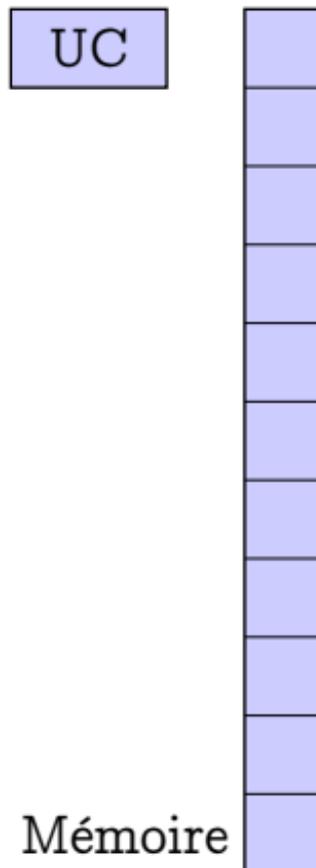
Paginer la table des pages

- Petites pages
- Plusieurs niveaux de tables de pages

Pagination Hiérarchique

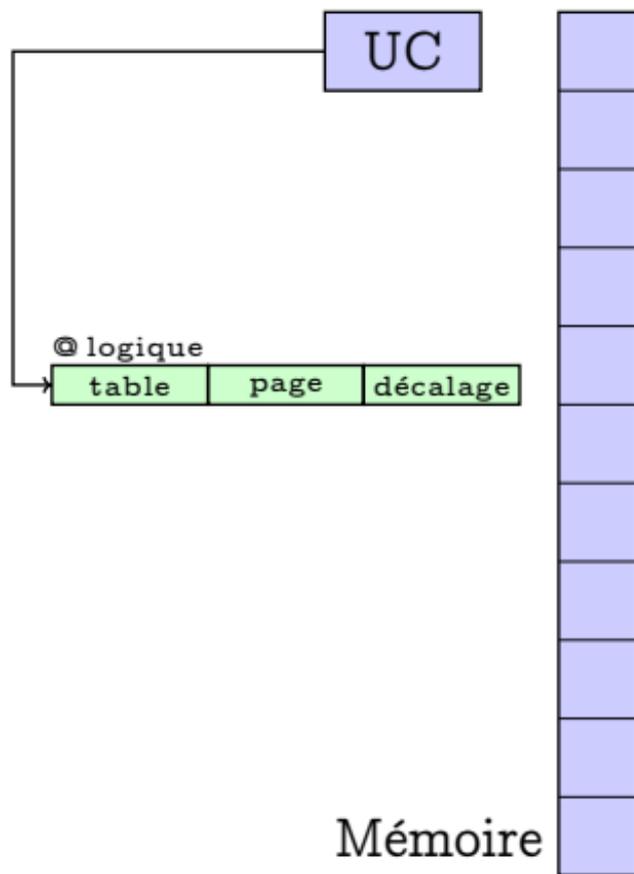
Pagination à deux niveaux

UC

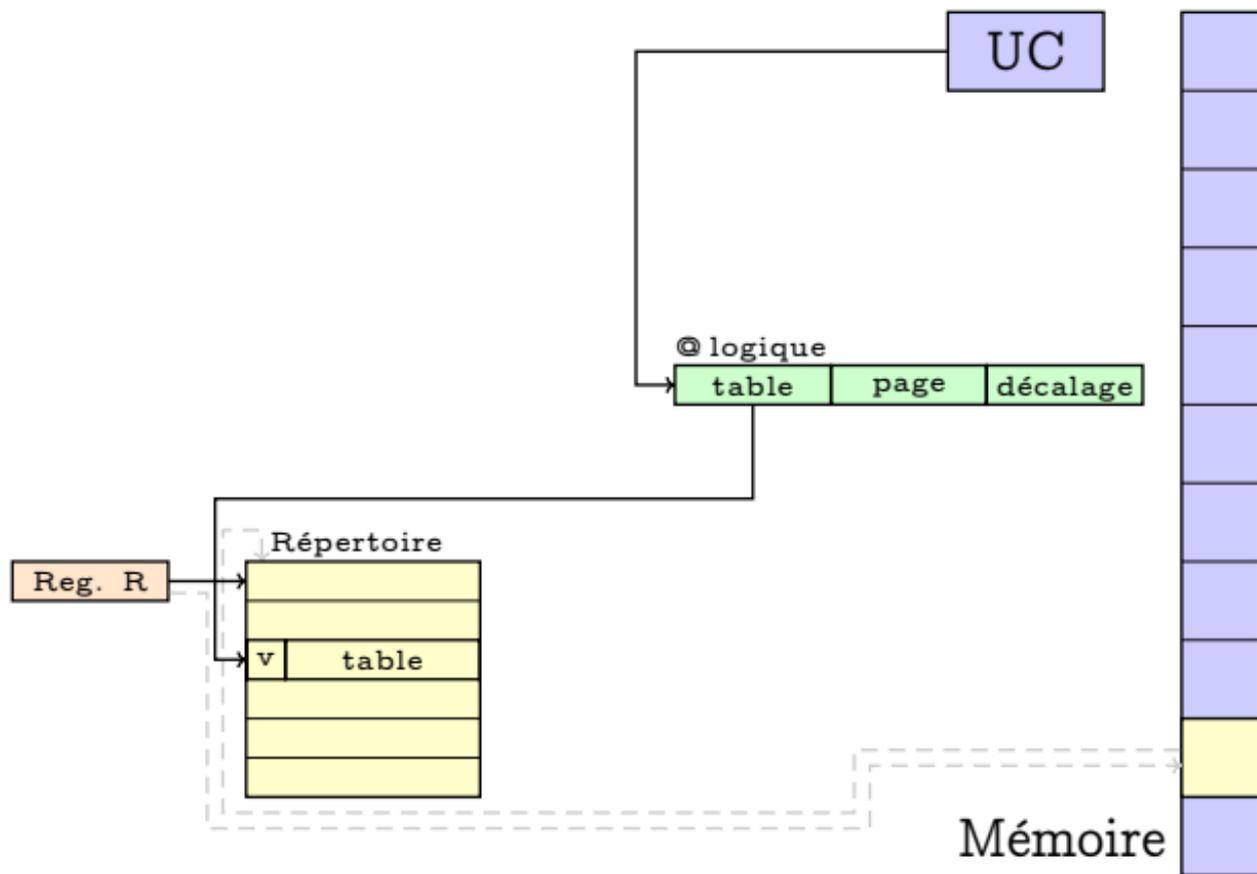


Mémoire

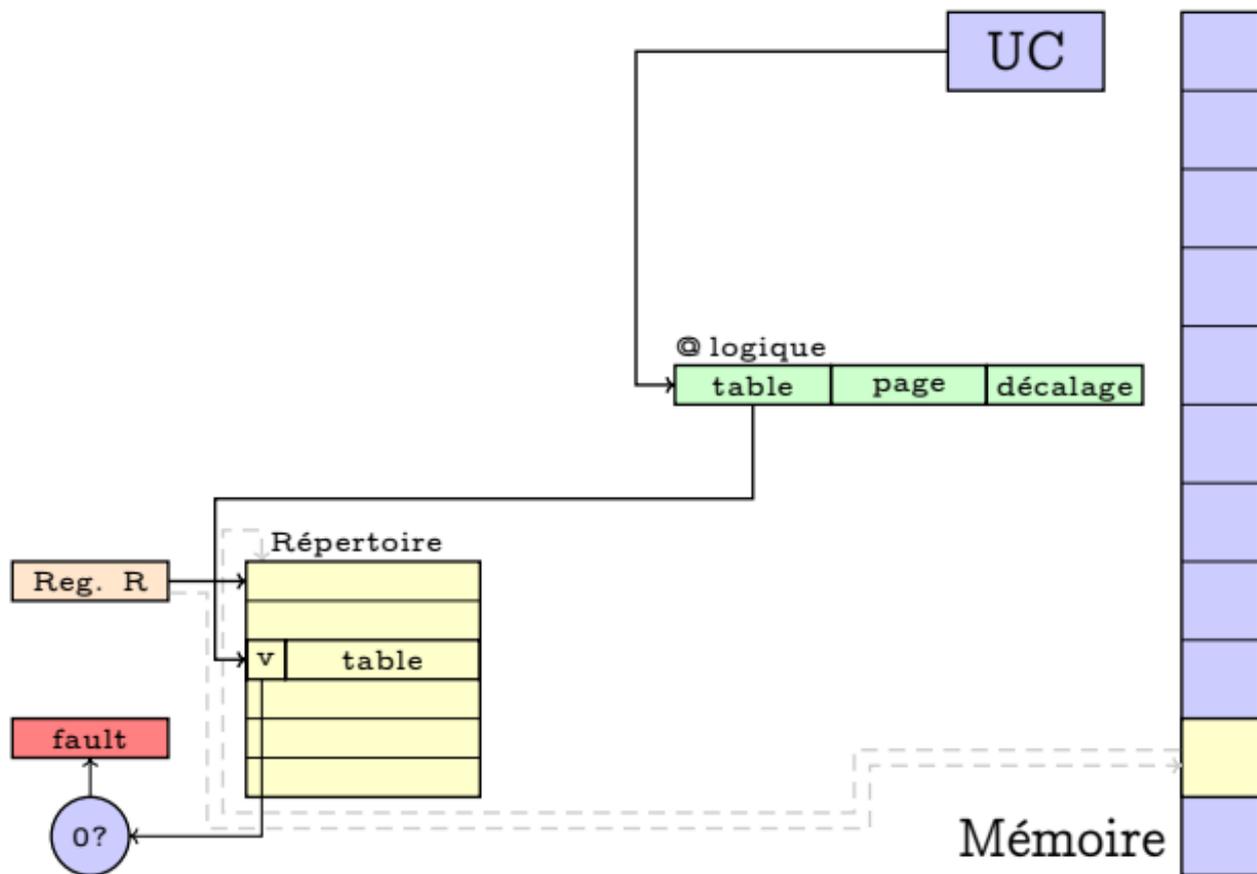
Pagination à deux niveaux



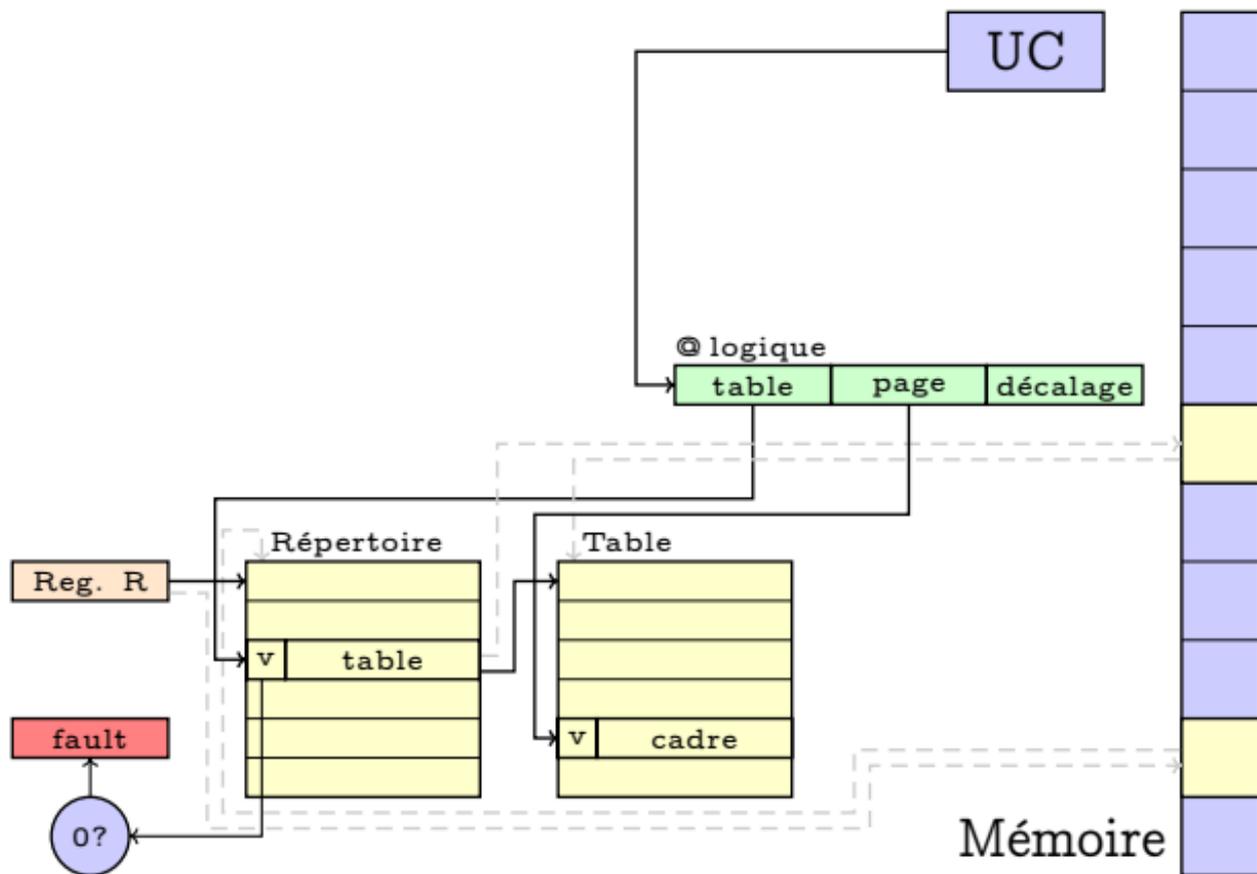
Pagination à deux niveaux



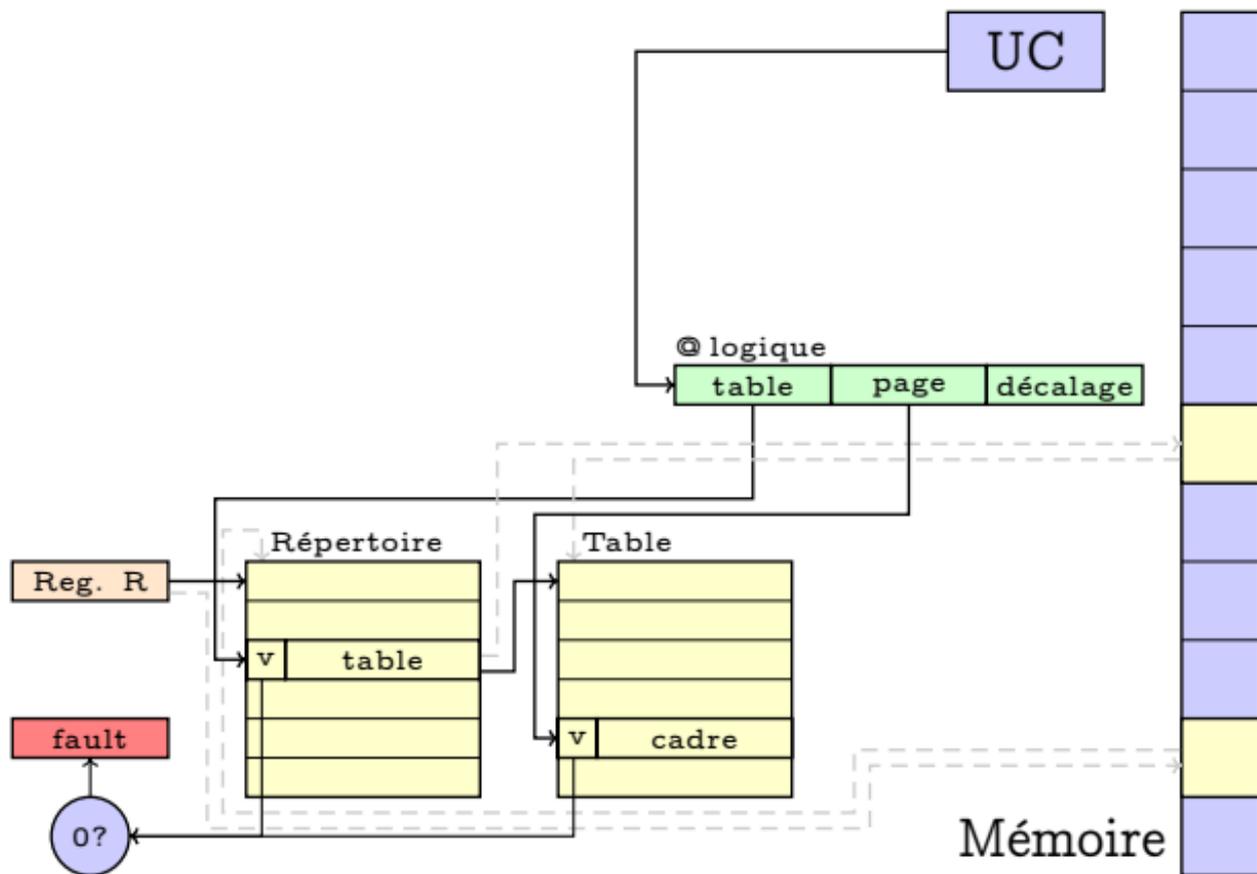
Pagination à deux niveaux



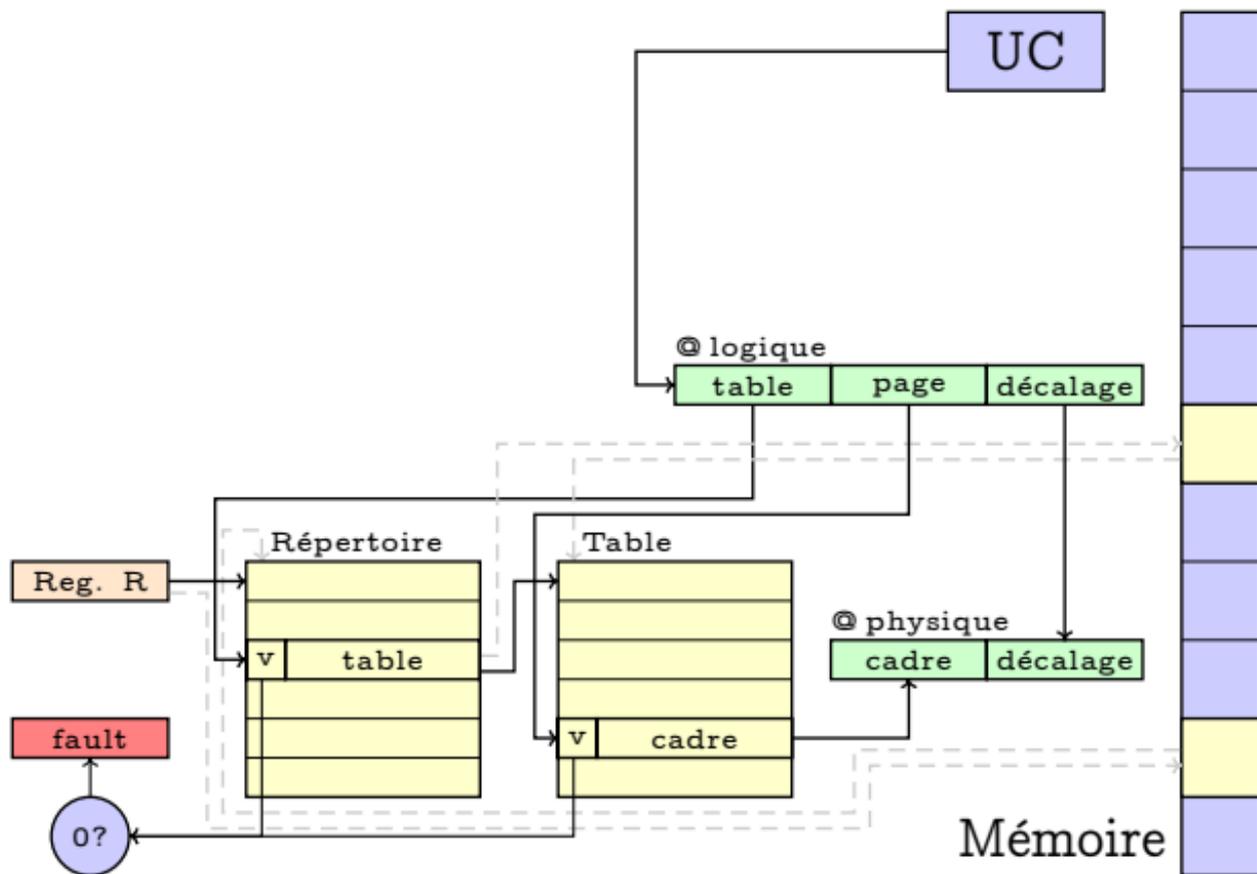
Pagination à deux niveaux



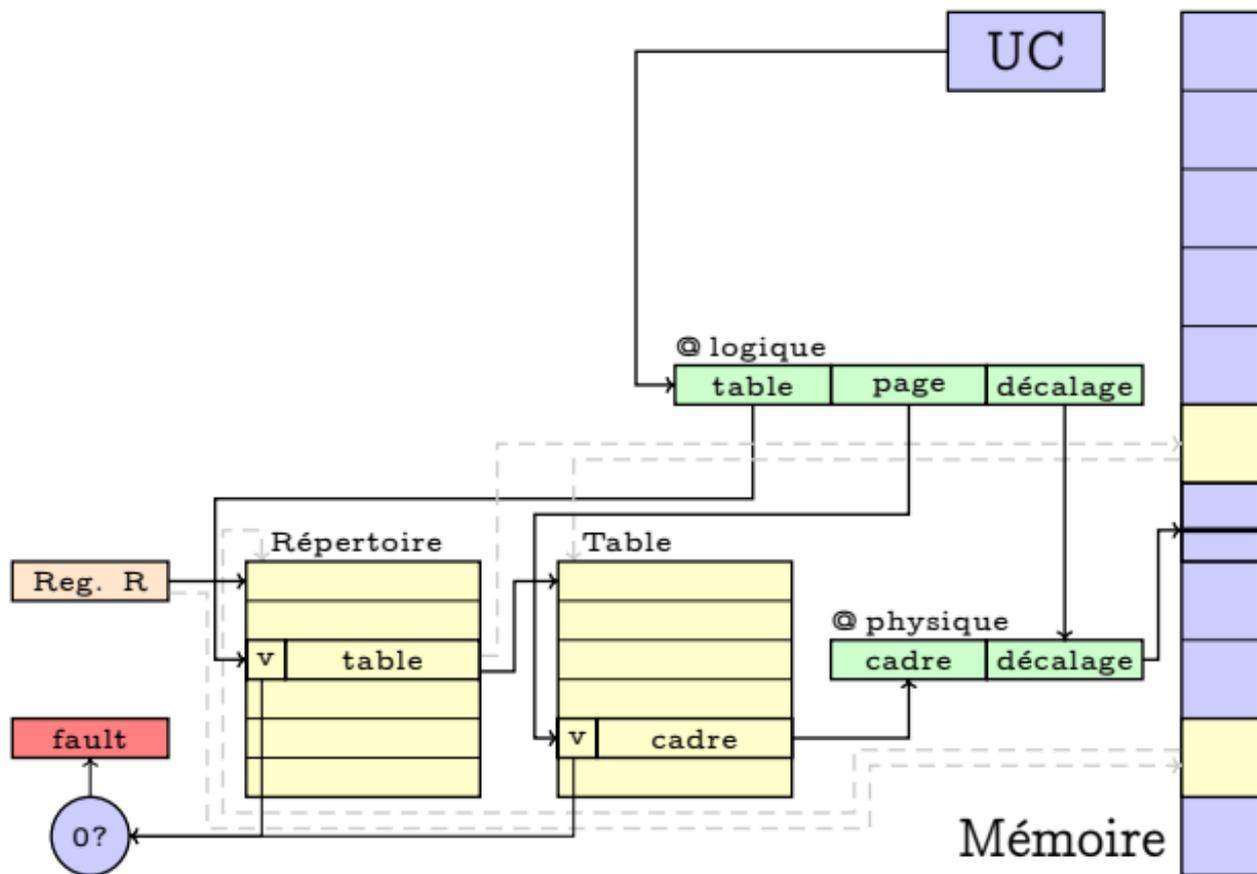
Pagination à deux niveaux



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 = $2^{20} \times 4o = 4Mo$ 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 = $2^{20} \times 4o = 4Mo$ par processus

Pagination à 2 niveaux

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

- Répertoire = 2^{10} lignes de $4o = 4Ko$
- 1 table de pages = 2^{10} lignes de $4o = 4Ko$

8Ko à 4Mo par processus