Allocation de la mémoire

Principe général

Multi-programmation

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

Principe général

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

Allocation exclusive

Principe

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

Allocation exclusive

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

Allocation exclusive

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

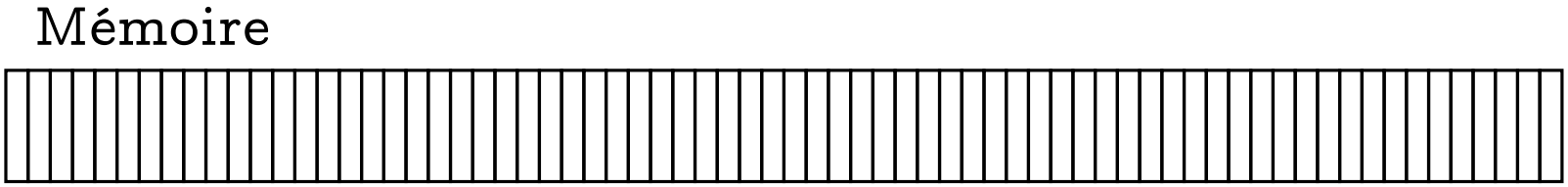
X Très lent !

Allocation contigüe

Principe

Multi-programmation et temps partagé

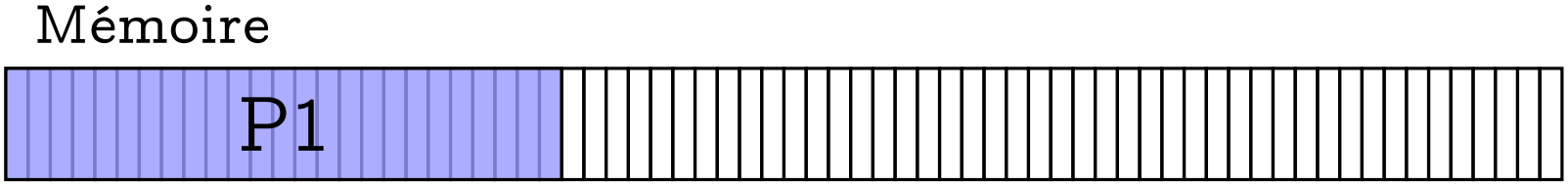
- Les processus n'utilisent pas toute la mémoire
- Placer tous les processus en mémoire



Principe

Multi-programmation et temps partagé

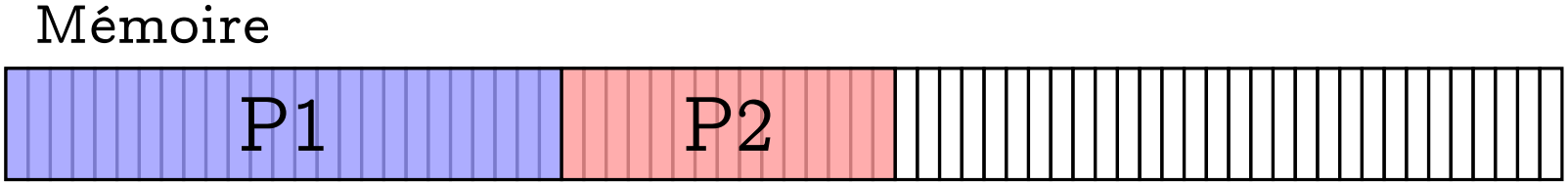
- Les processus n'utilisent pas toute la mémoire
- Placer tous les processus en mémoire



Principe

Multi-programmation et temps partagé

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



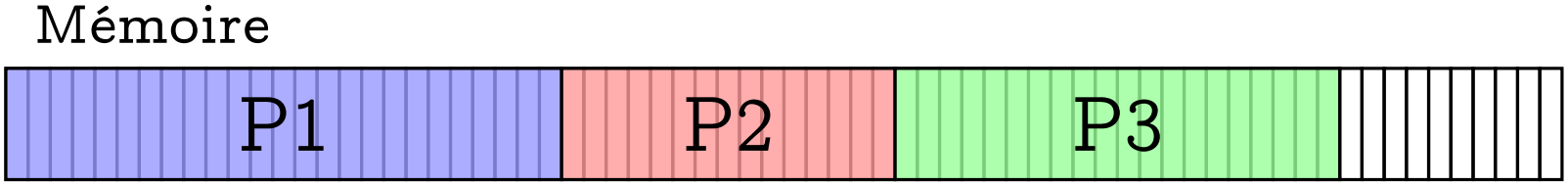
Problème

- 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



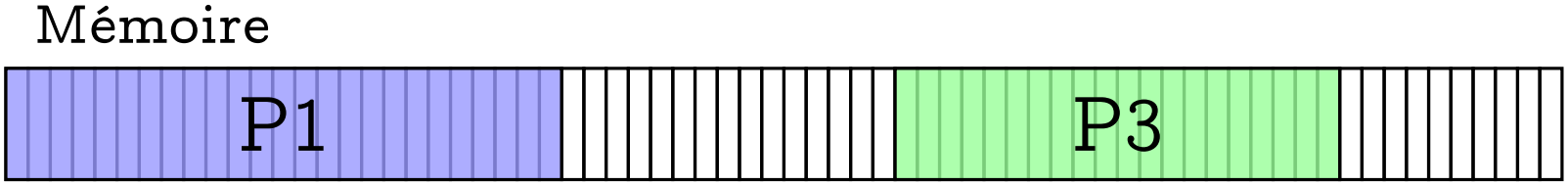
Problème

- 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



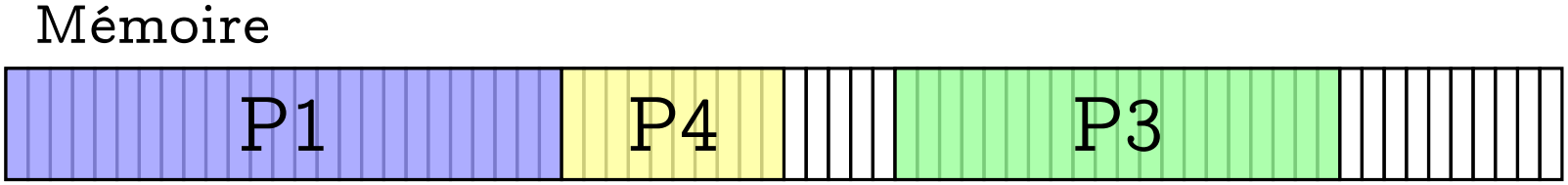
Problème

- 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



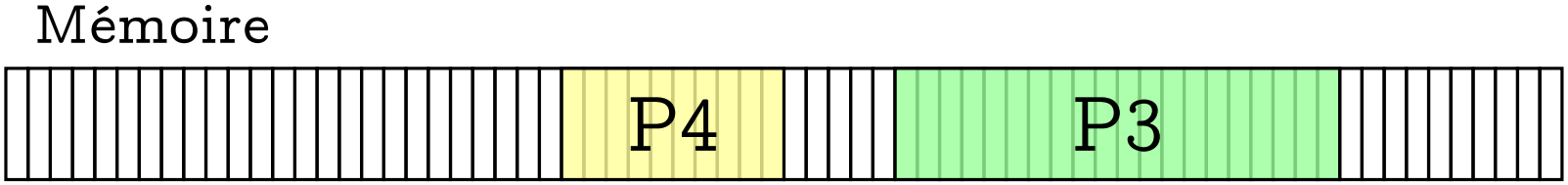
Problème

- 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



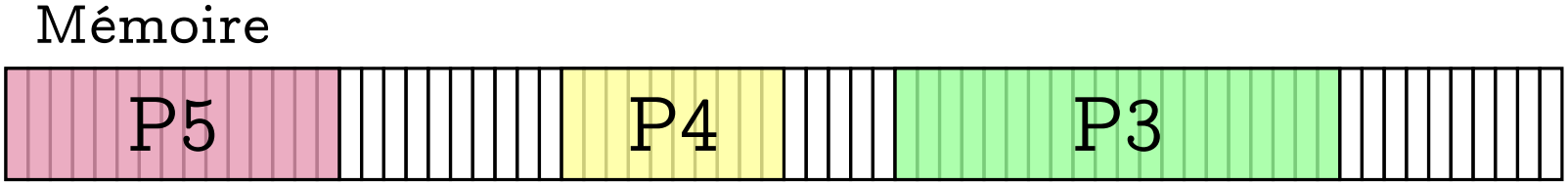
Problème

- Où placer les processus ?

Principe

Multi-programmation et temps partagé

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



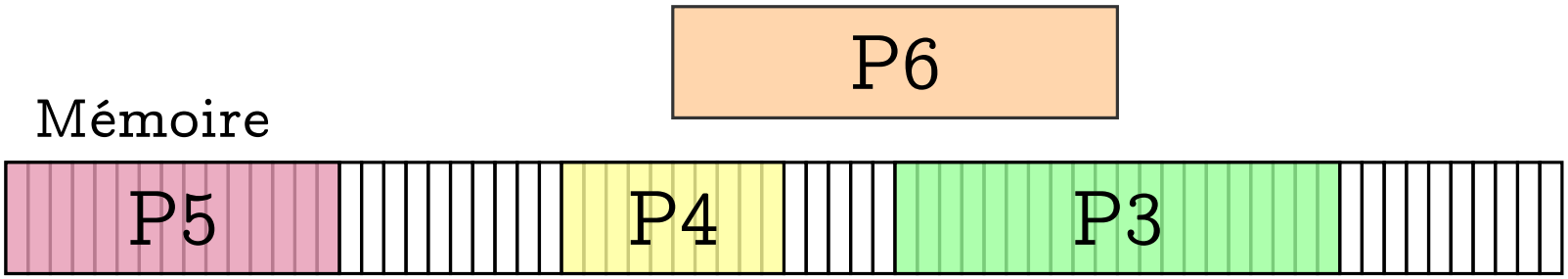
Problème

- Où placer les processus ?

Principe

Multi-programmation et temps partagé

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



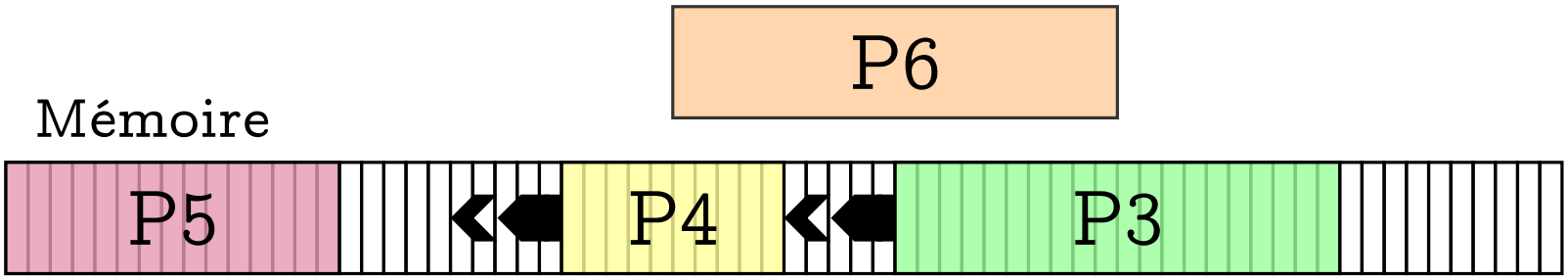
Problème

- 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



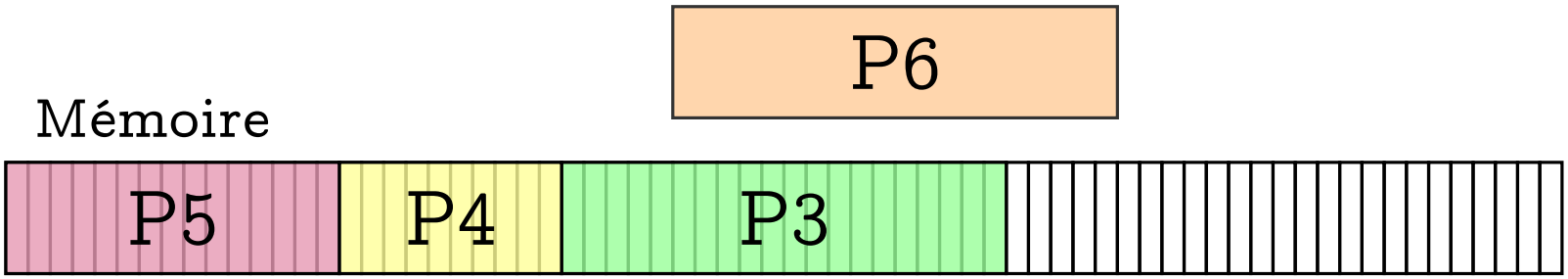
Problème

- 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



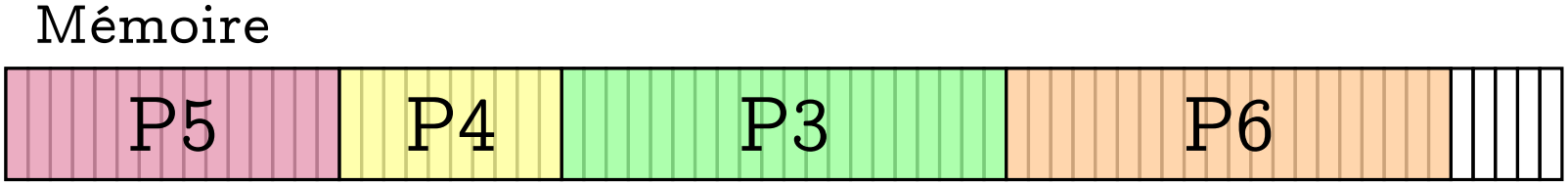
Problème

- 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



Problème

- Trous trop petits : il faut défragmenter

Où placer les processus ?

Stratégie *first-fit*

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

Où placer les processus ?

Stratégie *first-fit*

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

Stratégie *best-fit*

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

Où placer les processus ?

Stratégie *first-fit*

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

Stratégie *best-fit*

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

Stratégie *worse-fit*

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

Addresses de base variable

Problème:

Quelles sont les adresses des variables ?

Addresses de base variable

Problème:

Quelles sont les adresses des variables ?

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

Addresses de base variable

Problème:

Quelles sont les adresses des variables ?

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

@ symboliques :

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

Addresses de base variable

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
```

Addresses de base variable

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 ?

Édition de liens

Par le compilateur

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

Édition de liens

Par le compilateur

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

Par le système d'exploitation

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

Édition de liens

Par le compilateur

✓ Traduction efficace des adresses

X 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

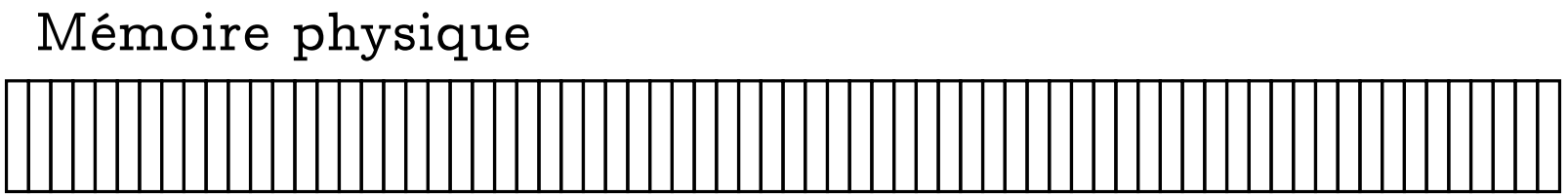
✓ Adresse de base dynamique

X Édition de liens très coûteuse

Adressage logique de la mémoire

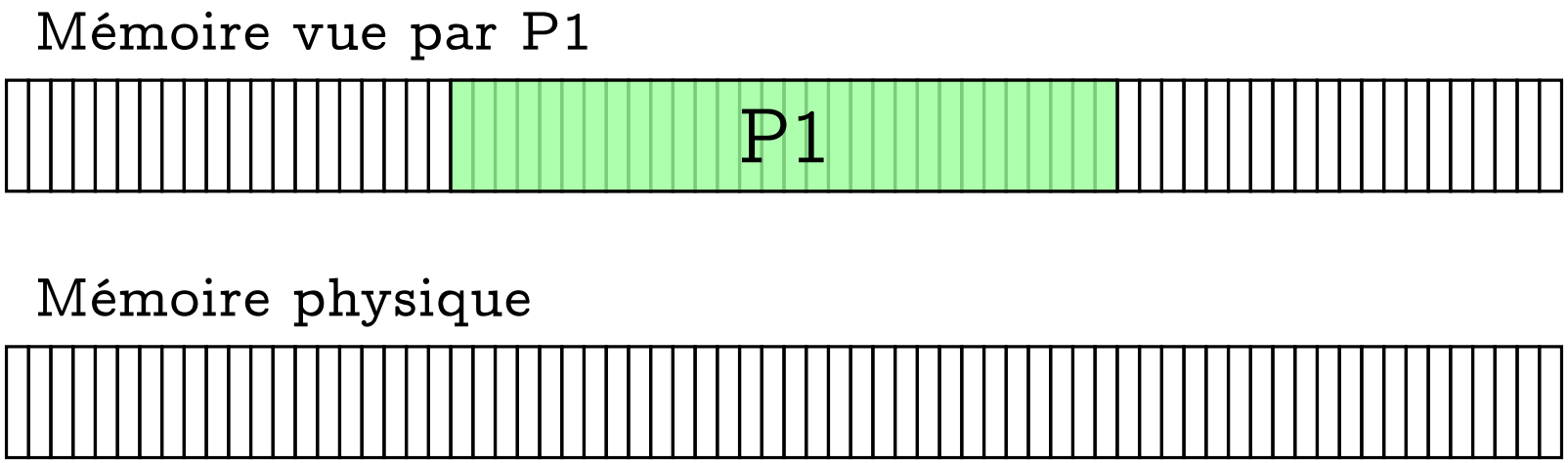
Principe

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



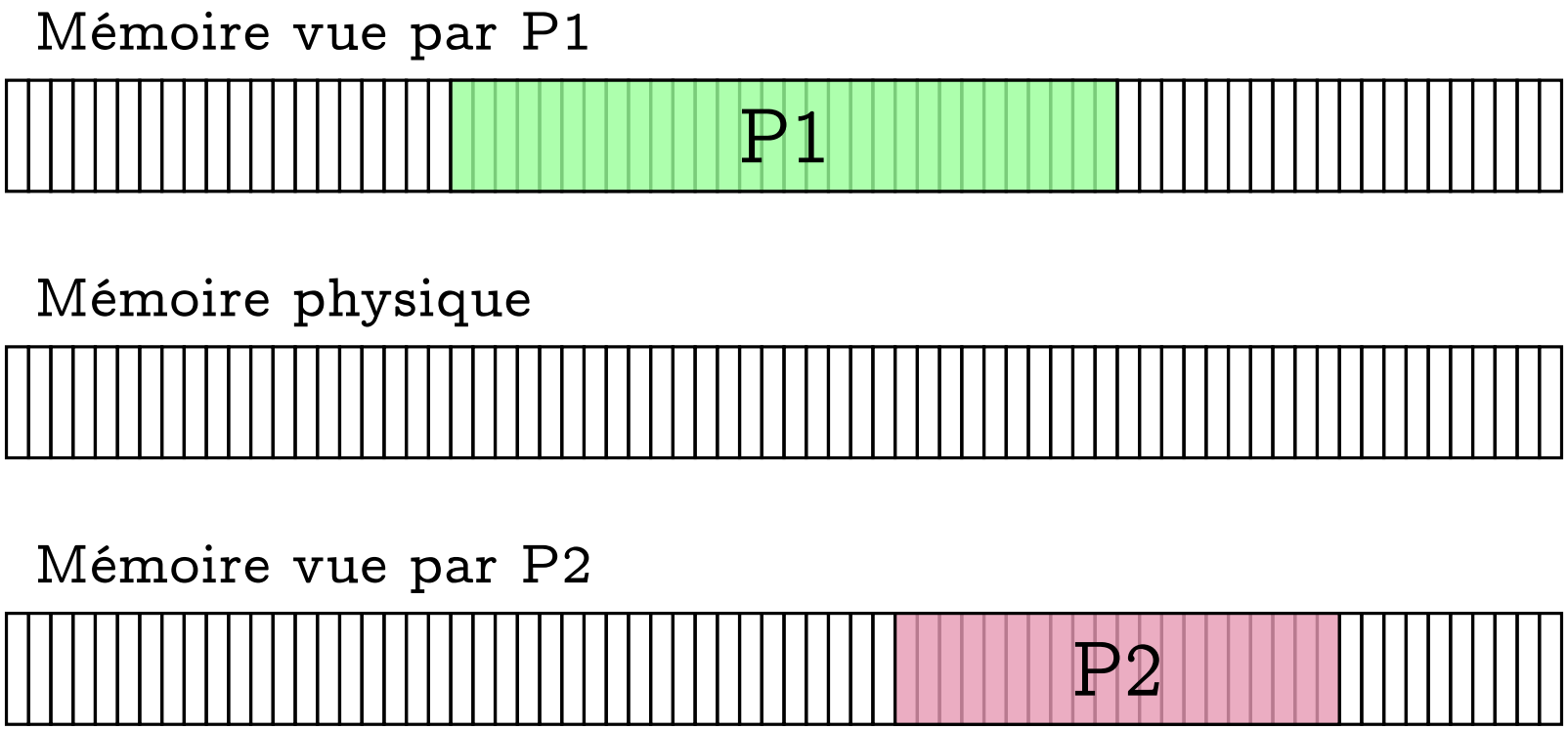
Principe

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



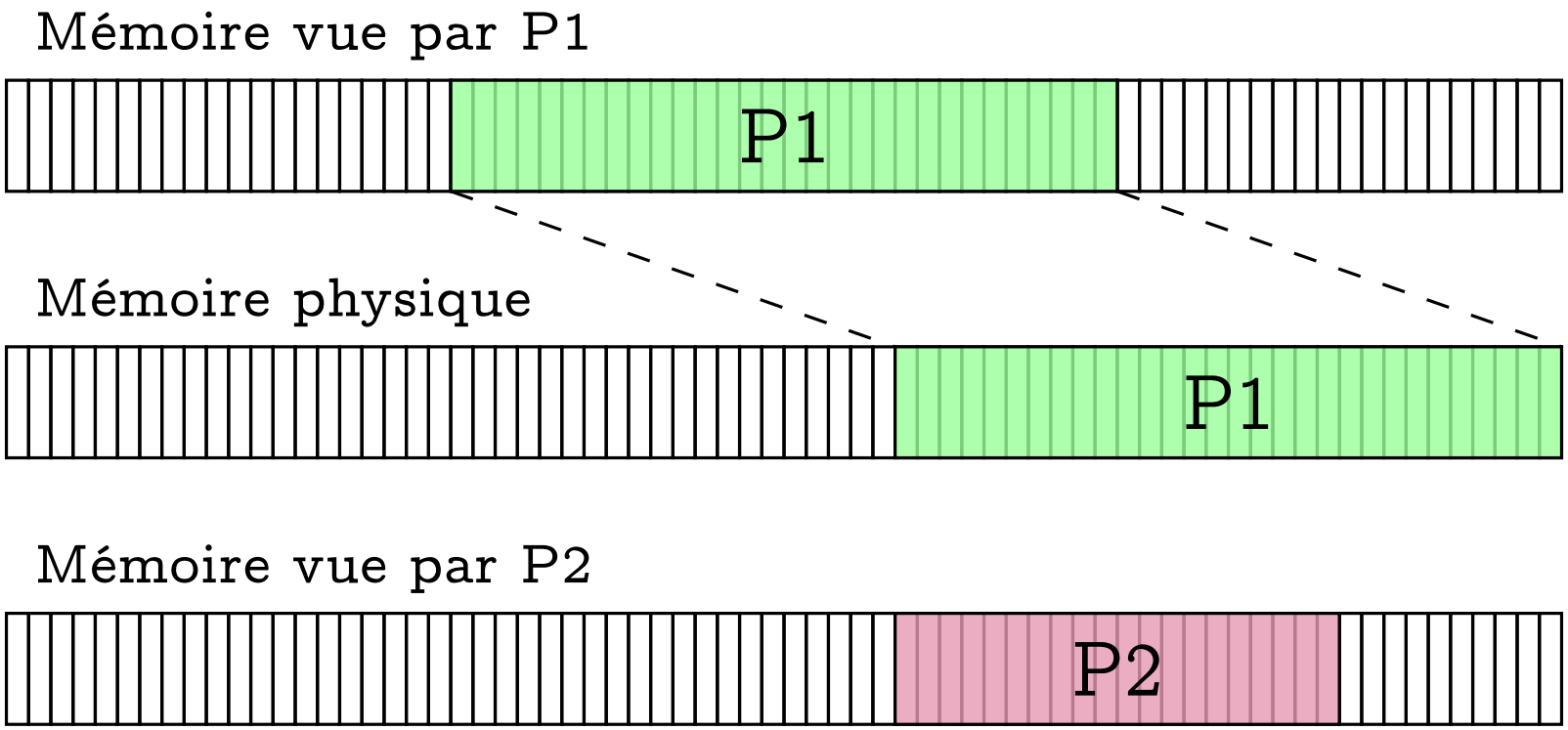
Principe

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



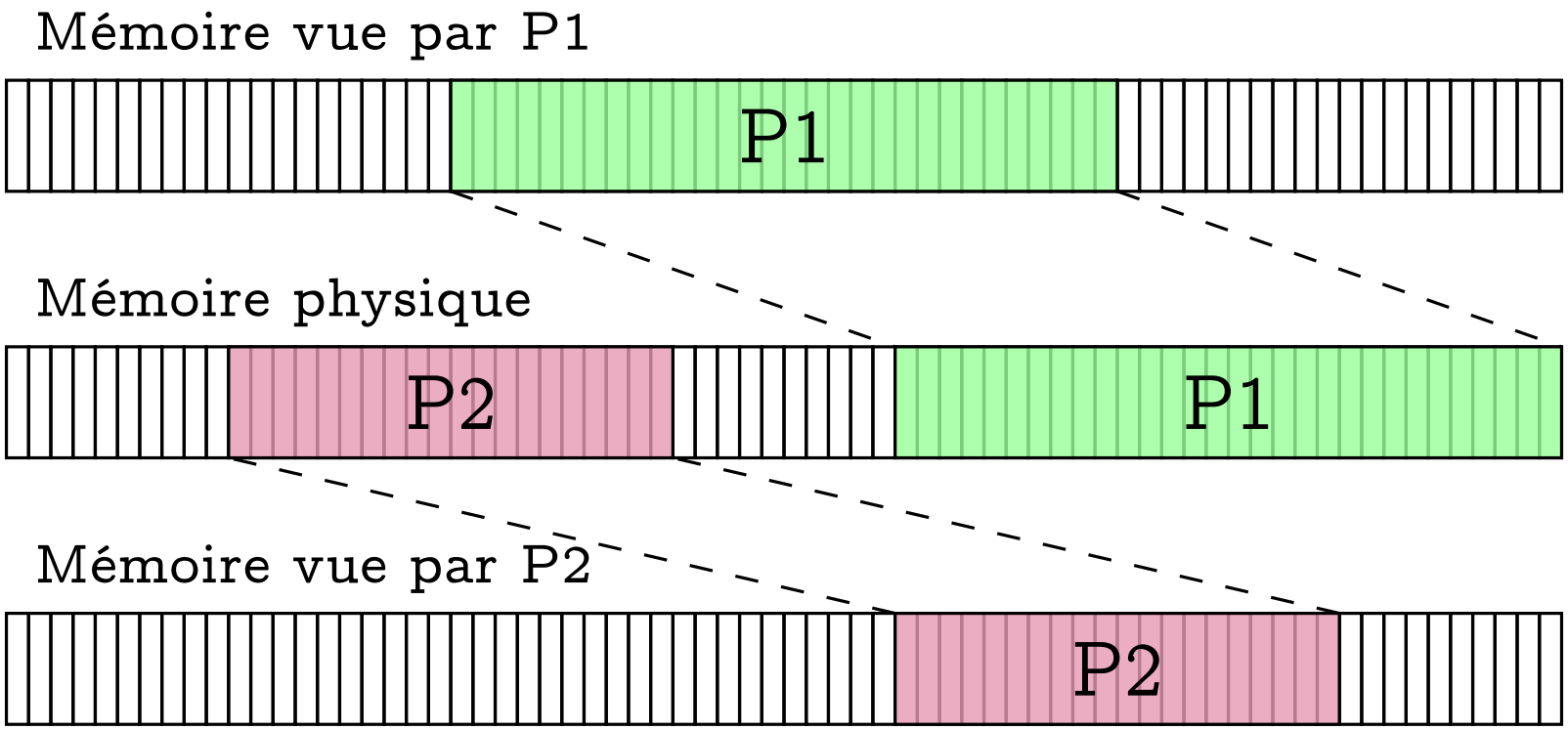
Principe

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



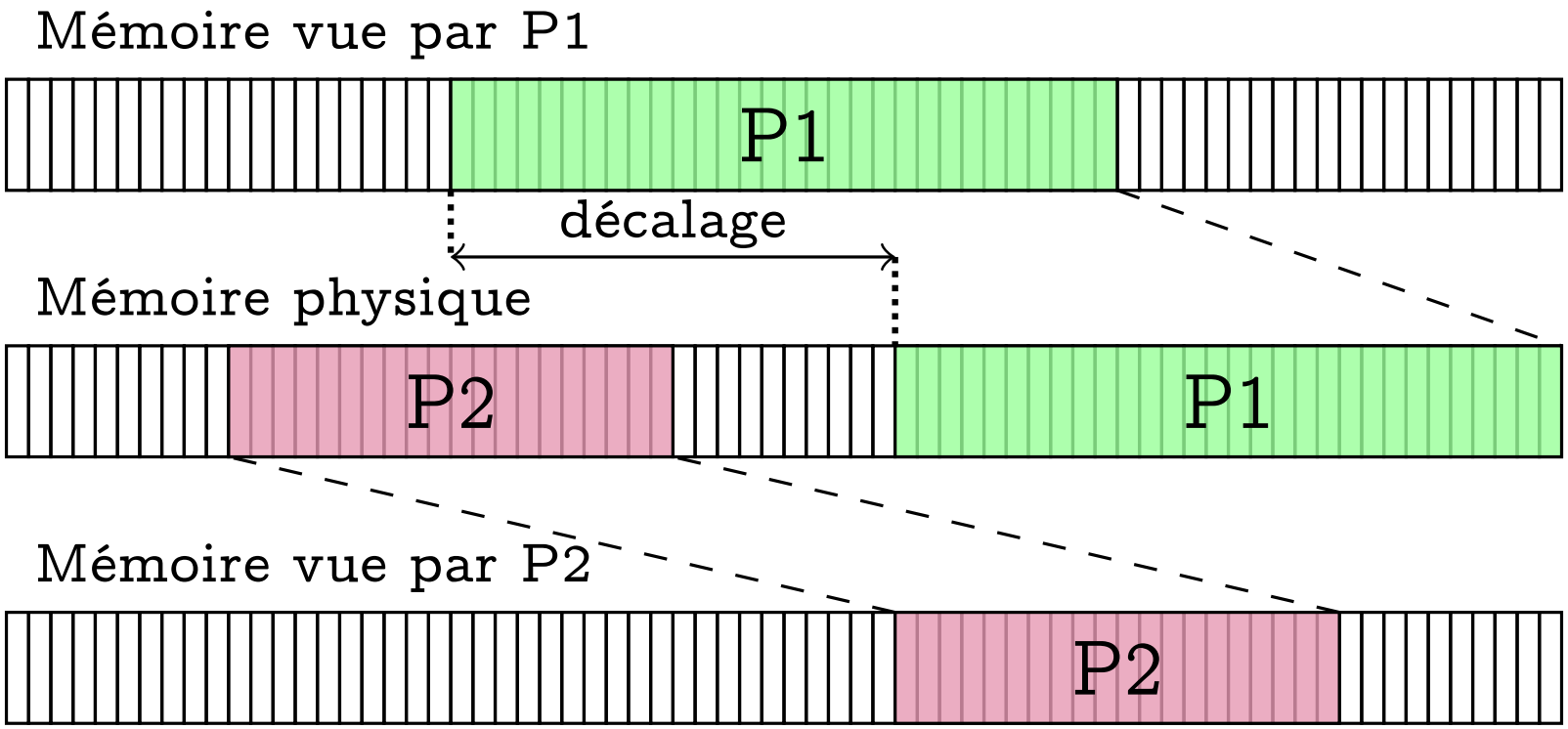
Principe

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

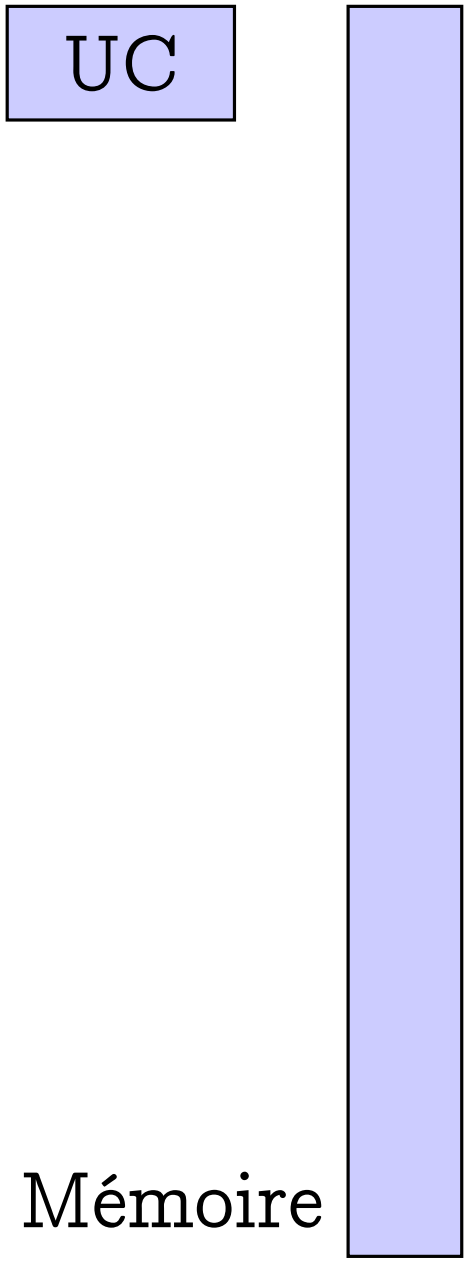
Chaque processus à son propre décalage qui est ajouté à chaque adresse utilisée.

Implémentation

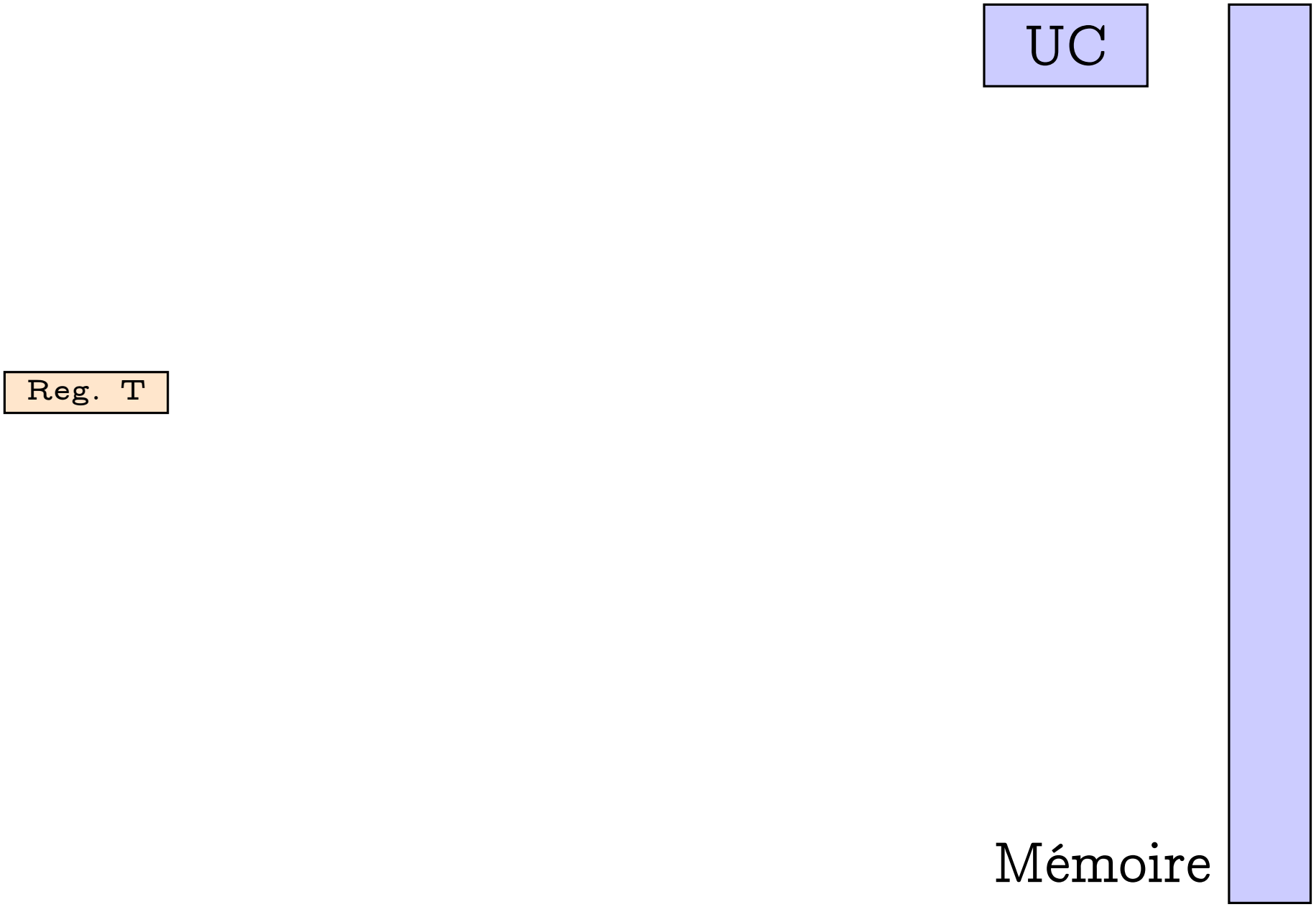
La traduction est réalisée par la MMU (*Memory Managment Unit*)

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

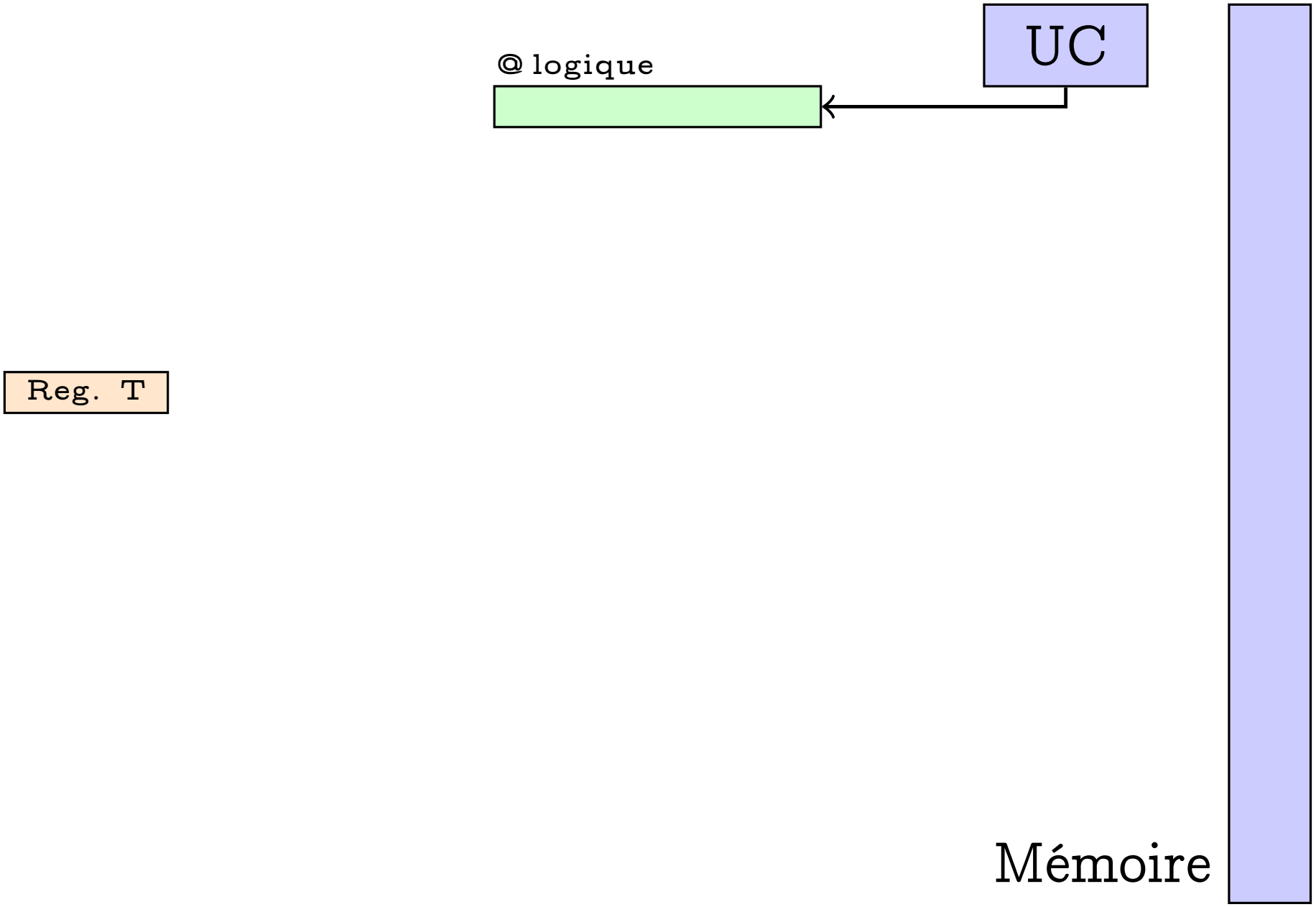
Translation



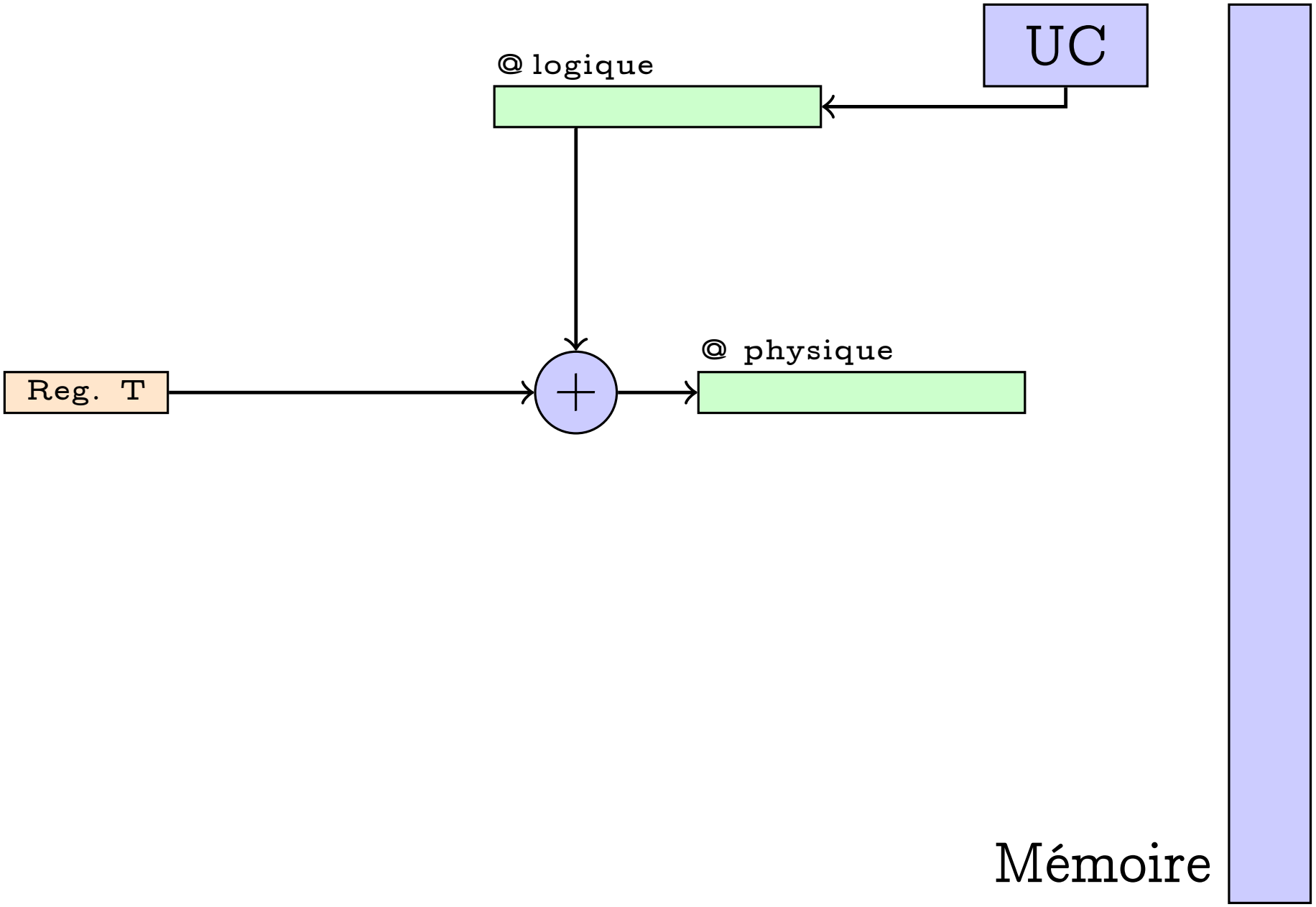
Translation



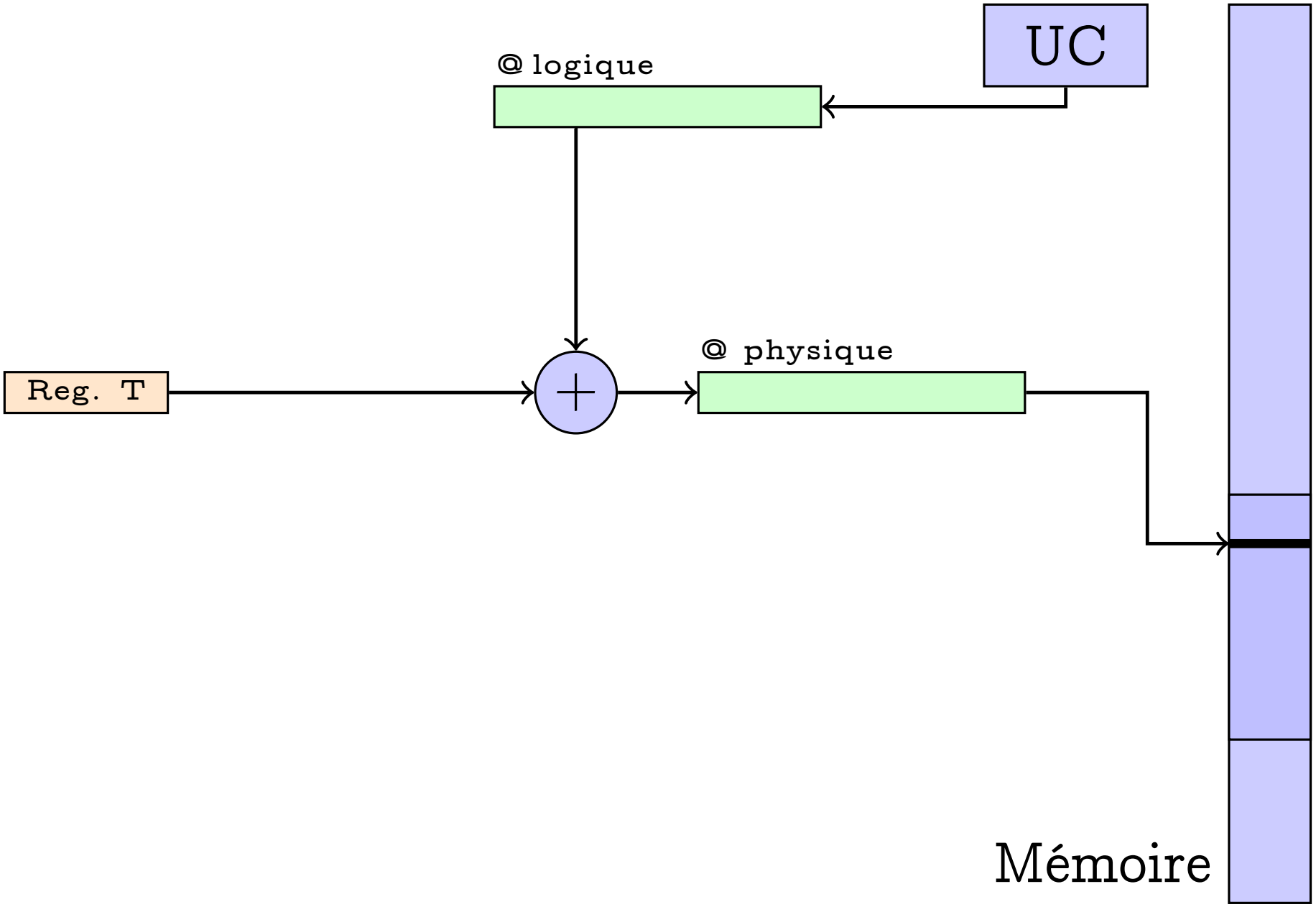
Translation



Translation



Translation



Fragmentation

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

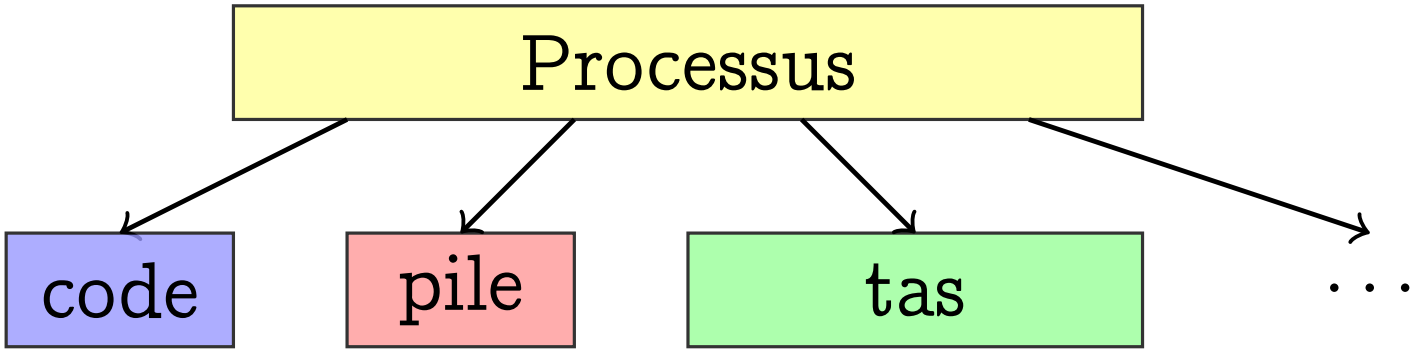
Segmentation

Découpage sémantique

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

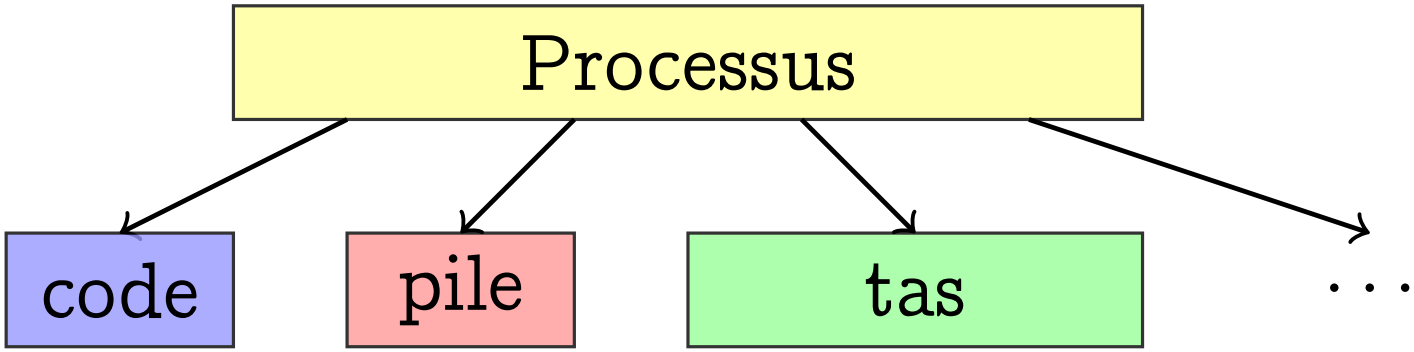
Découpage sémantique

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



Découpage sémantique

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



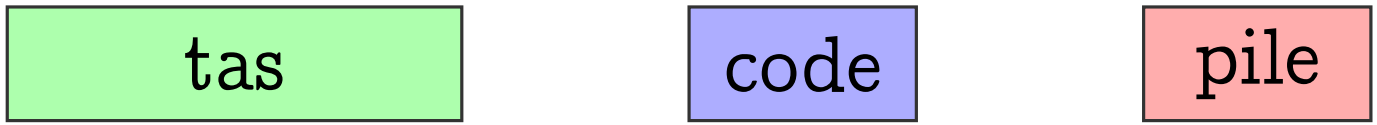
Segments:

Chaque segment peut être placé indépendamment en mémoire.

Adressage logique

Les segments sont placés en mémoire logique

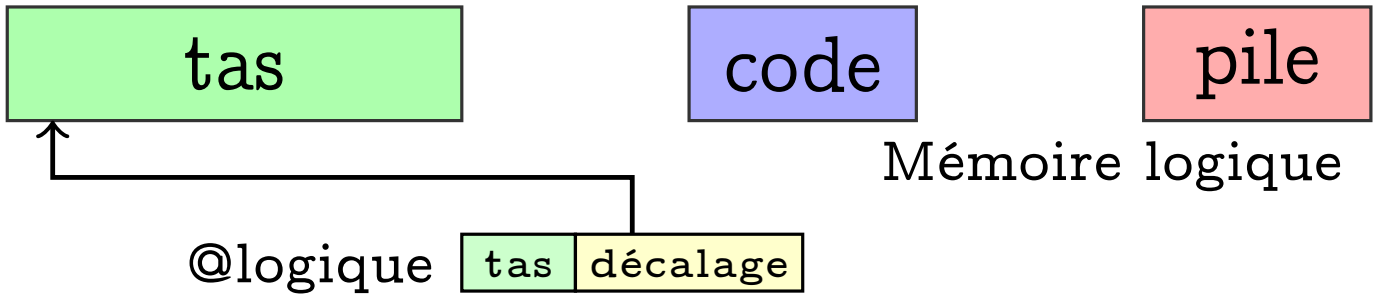
L'adresse logique se compose d'un sélecteur de segment et d'un décalage dans ce segment.



Mémoire logique

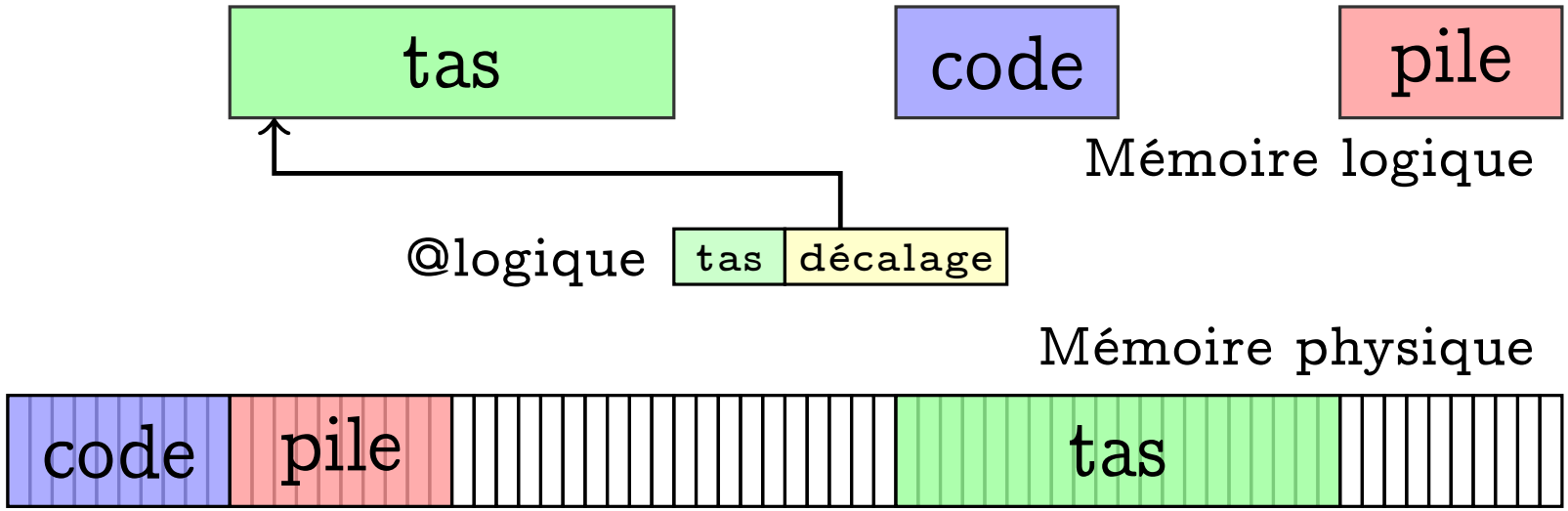
Les segments sont placés en mémoire logique

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



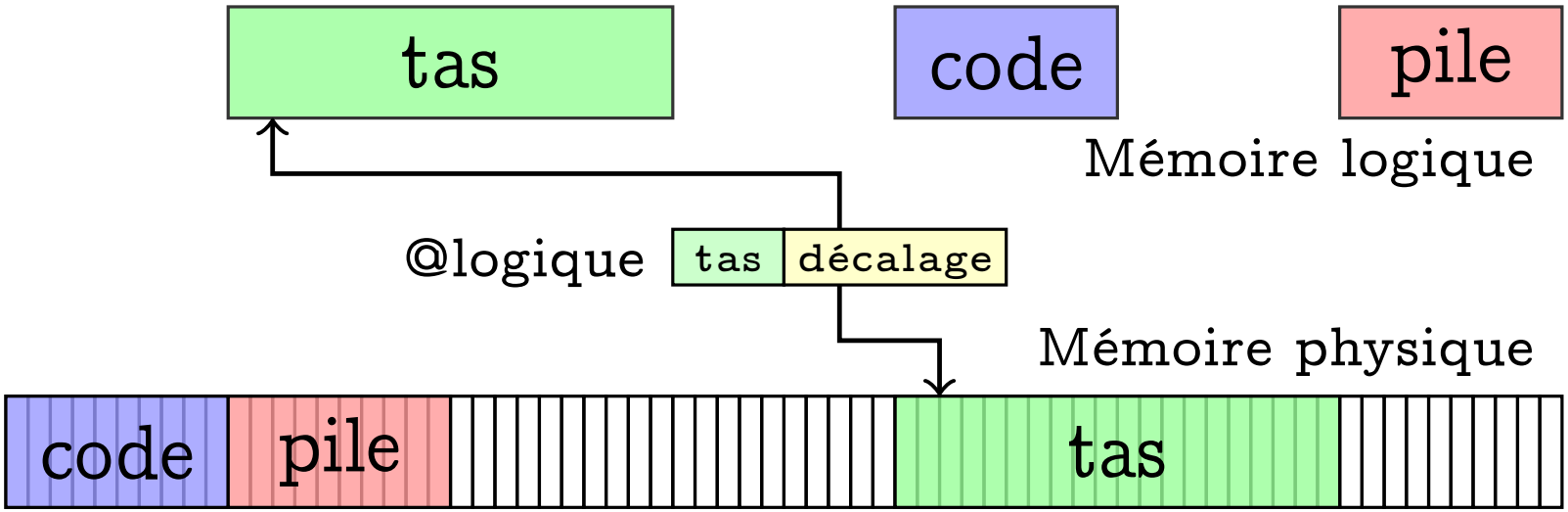
Les segments sont placés en mémoire logique

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



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)

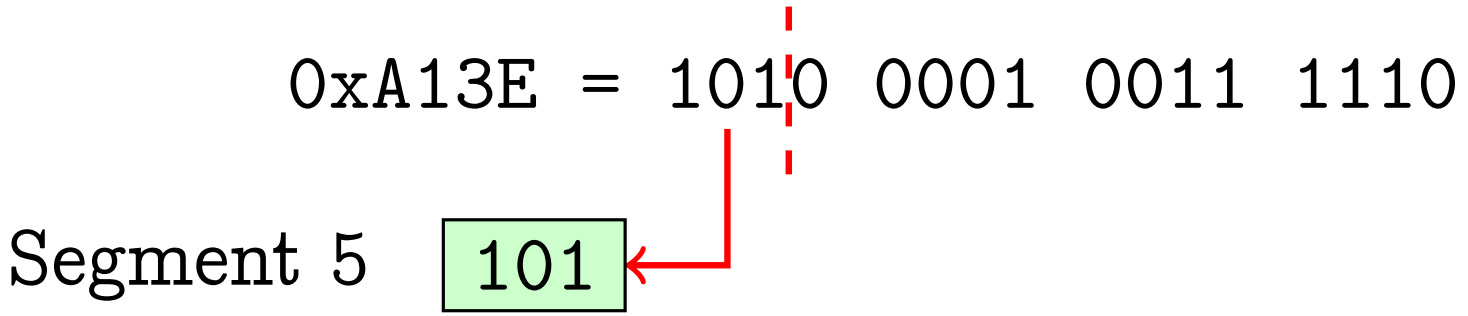
`0xA13E = 1010 0001 0011 1110`

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)

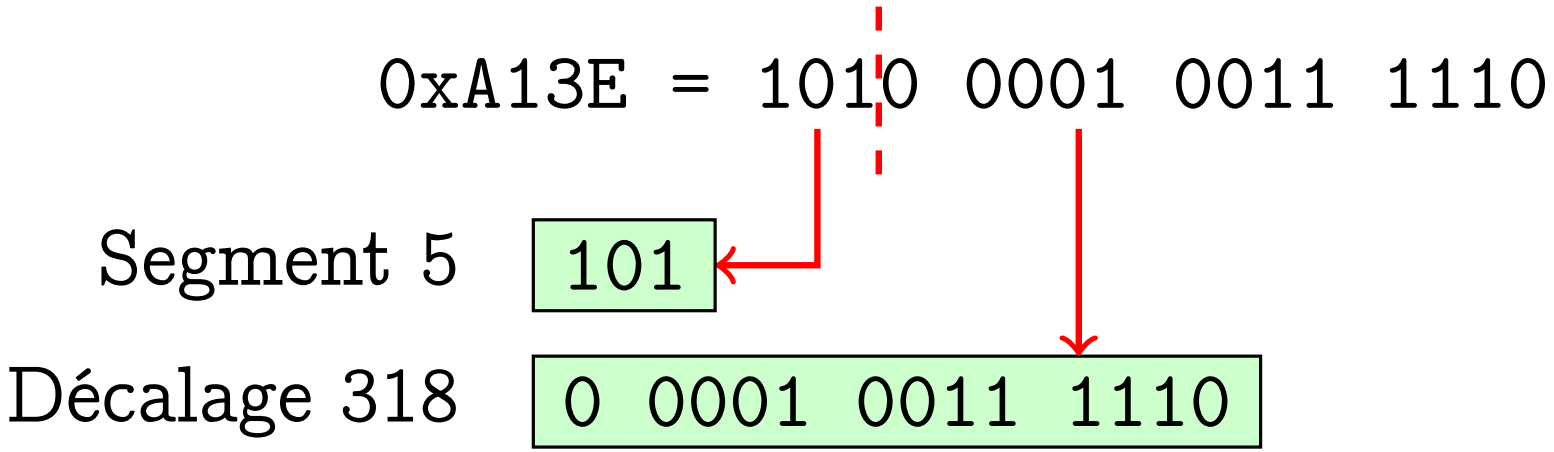


Découpage

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)

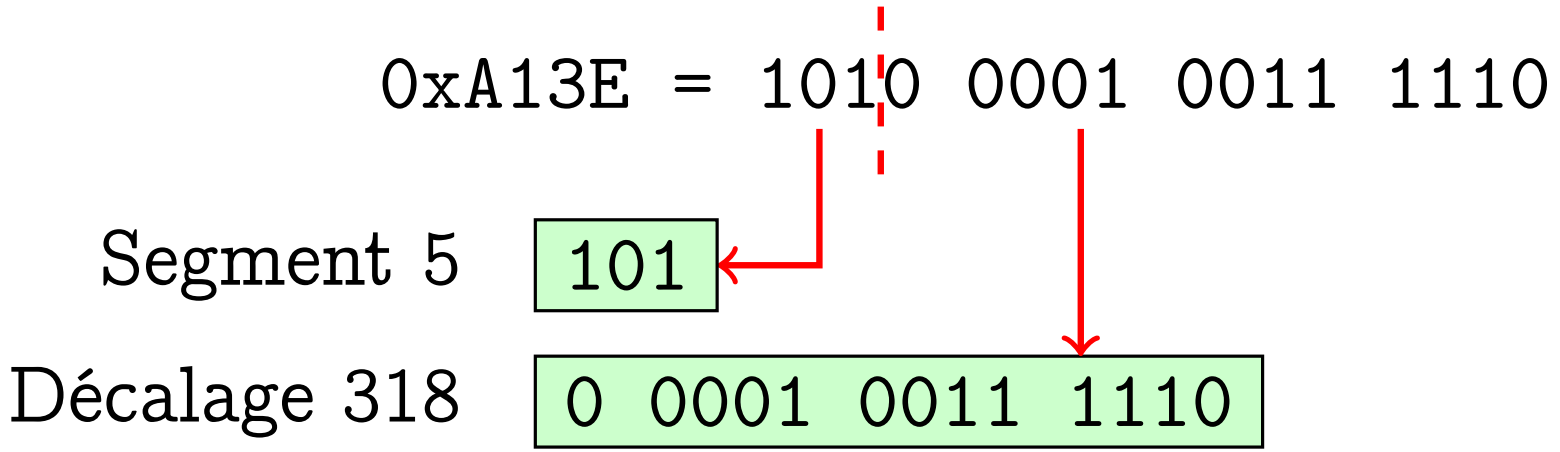


Découpage

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)



319ème octet du 5ème segment

Table des segments

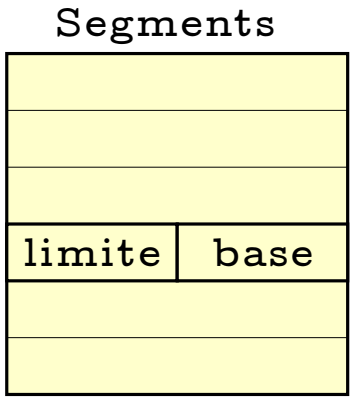
En mémoire physique

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

Table des segments

En mémoire physique

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



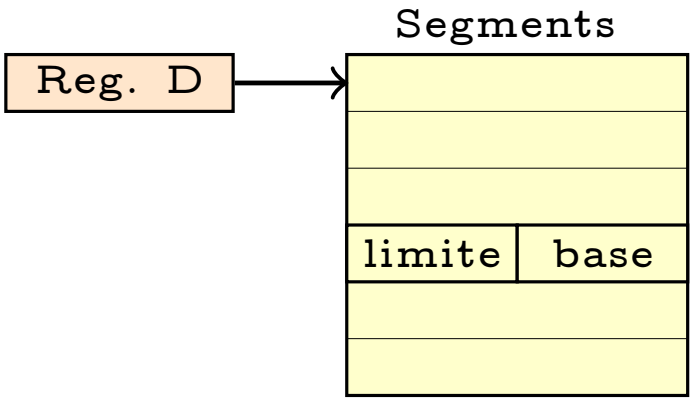
Limite : taille

Base : translation

Table des segments

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

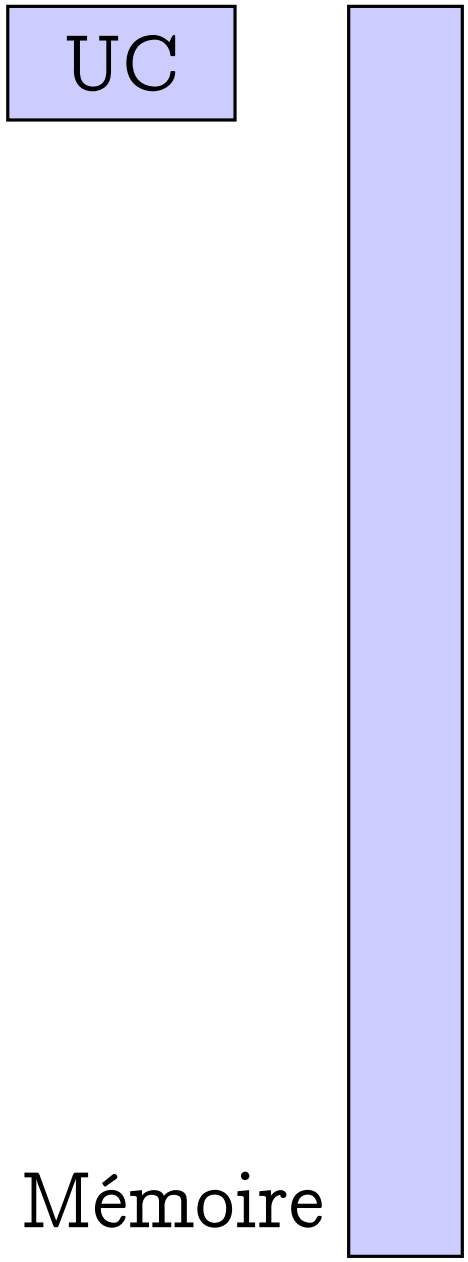
En cas d'erreur

La MMU provoque un SEGFAULT :

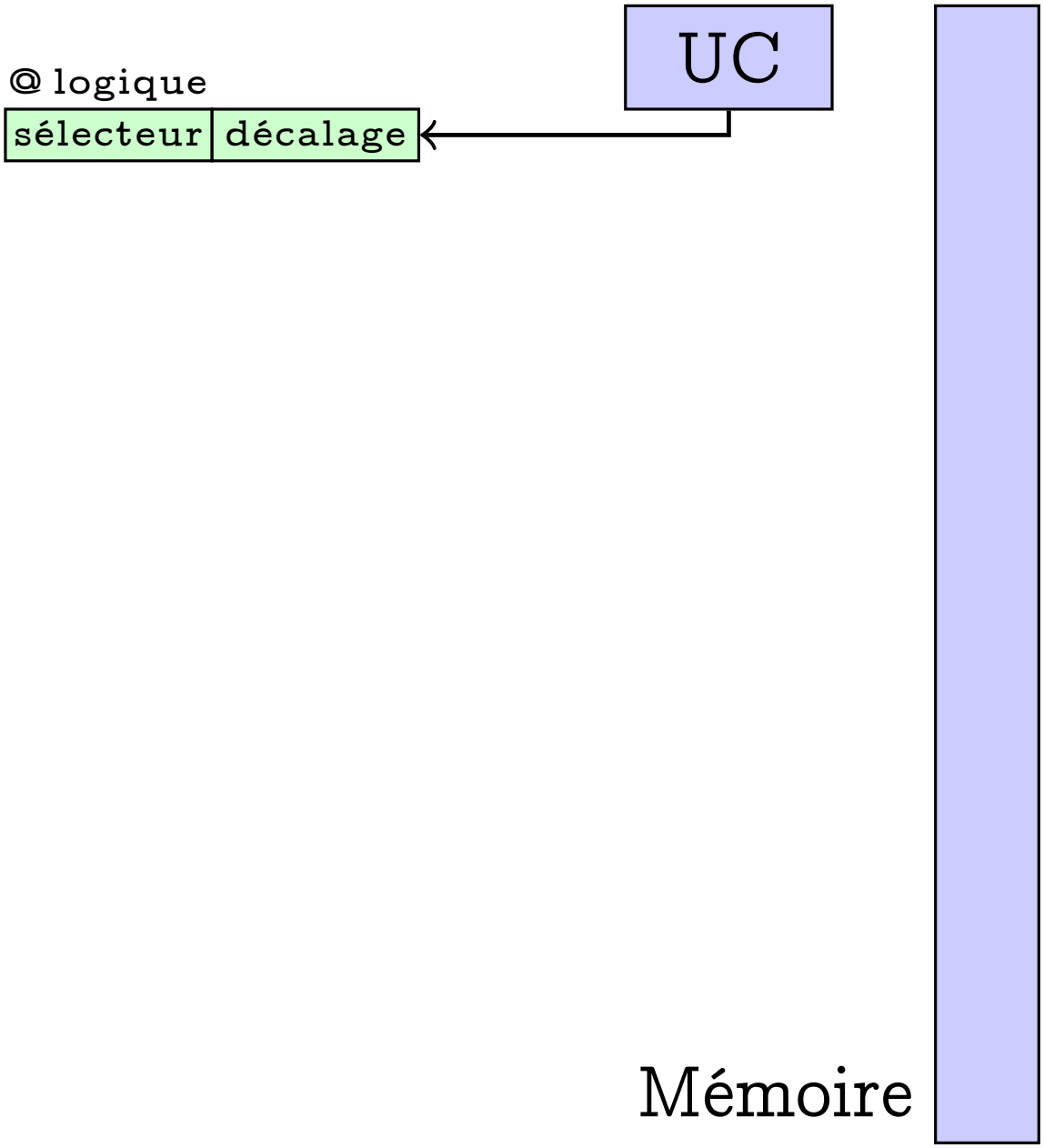
Segmentation Fault

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

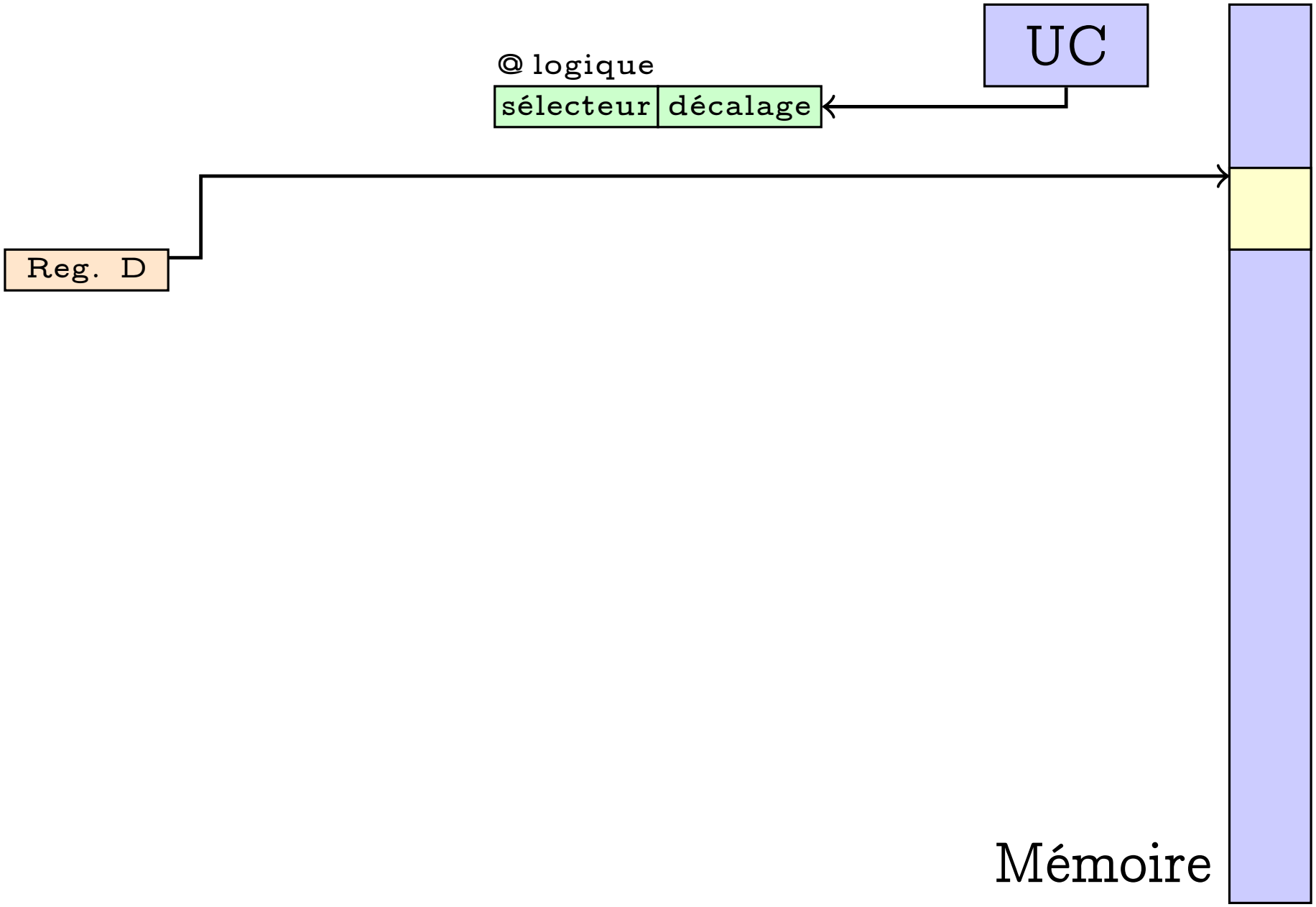
Segmentation



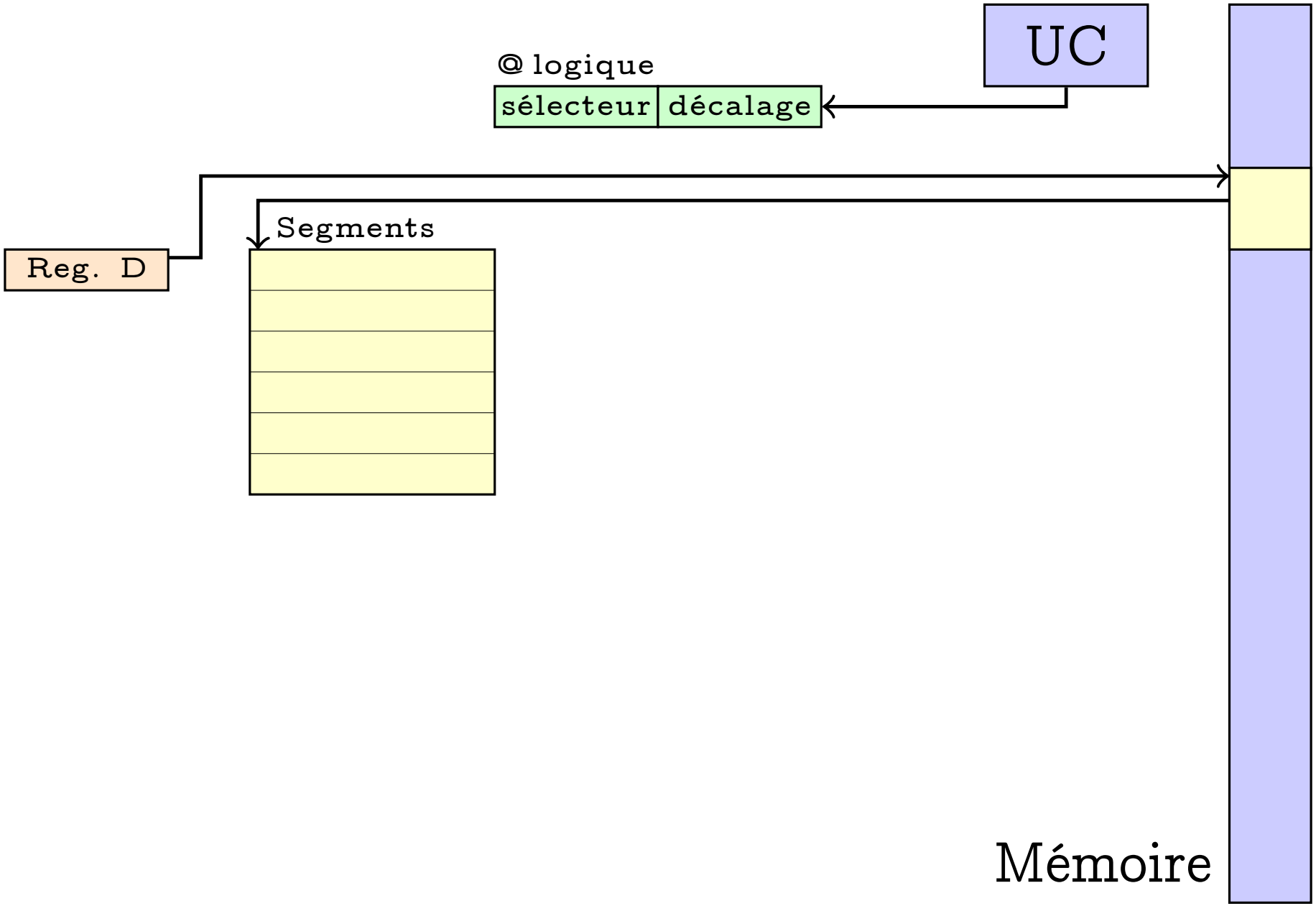
Segmentation



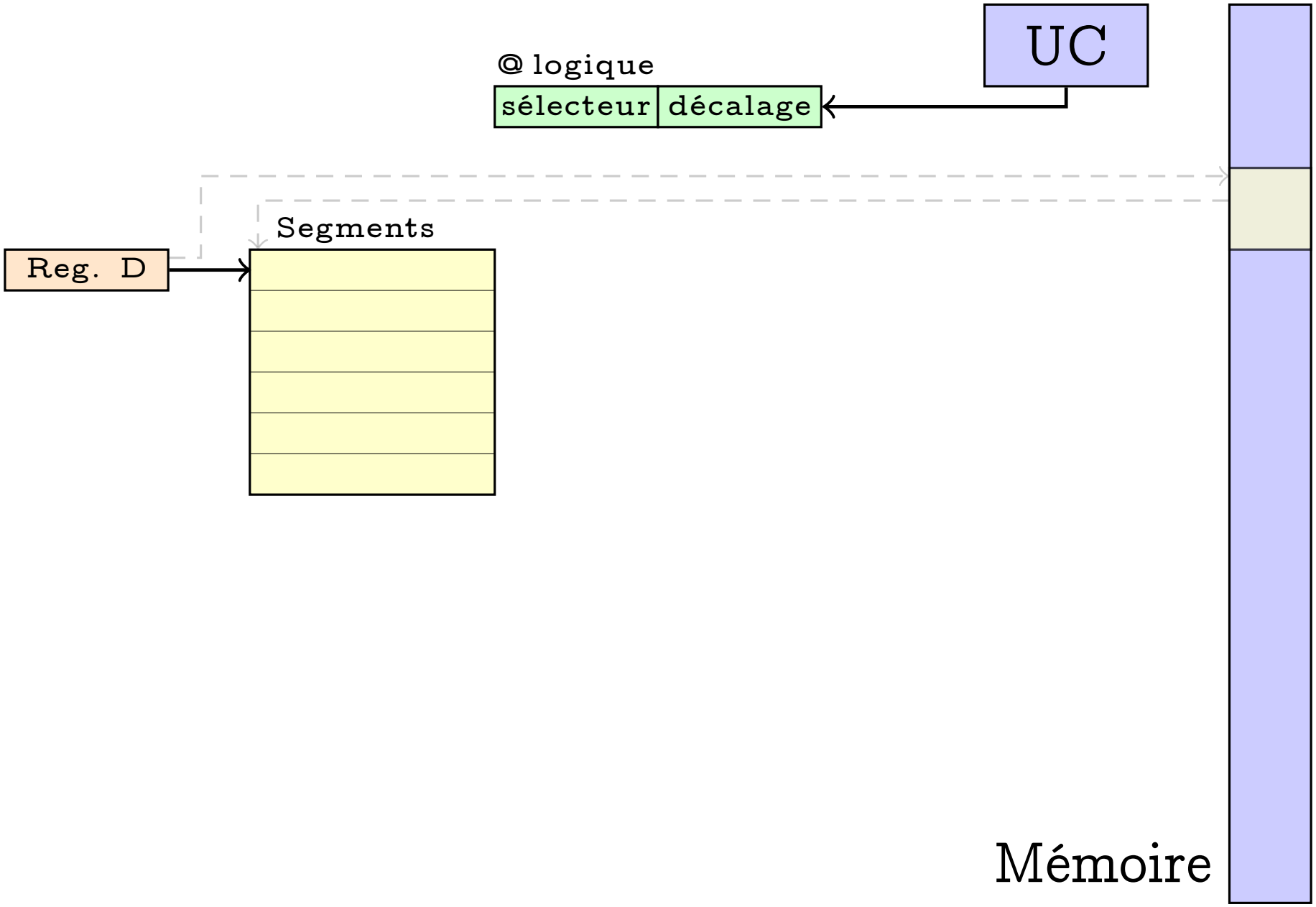
Segmentation



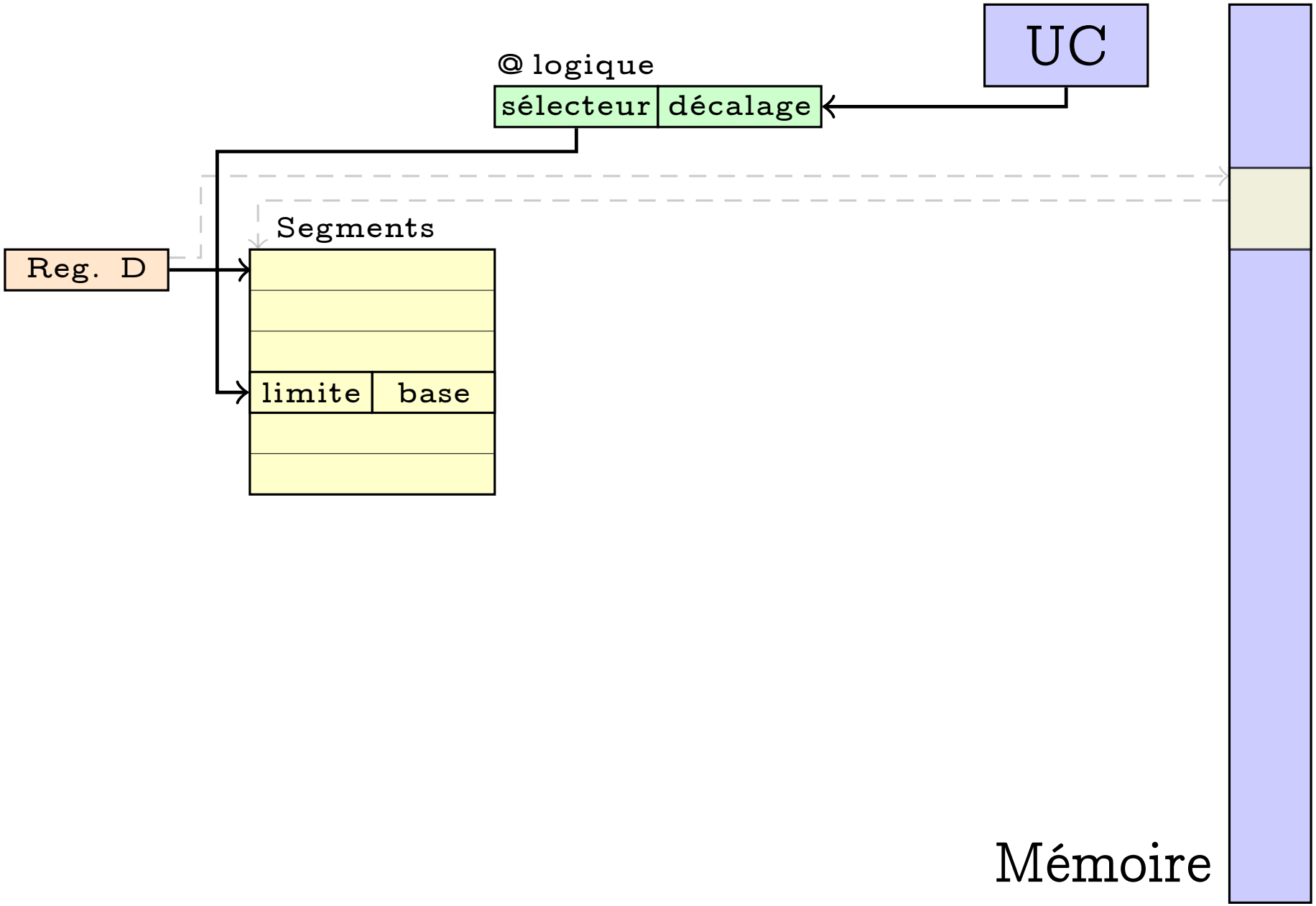
Segmentation



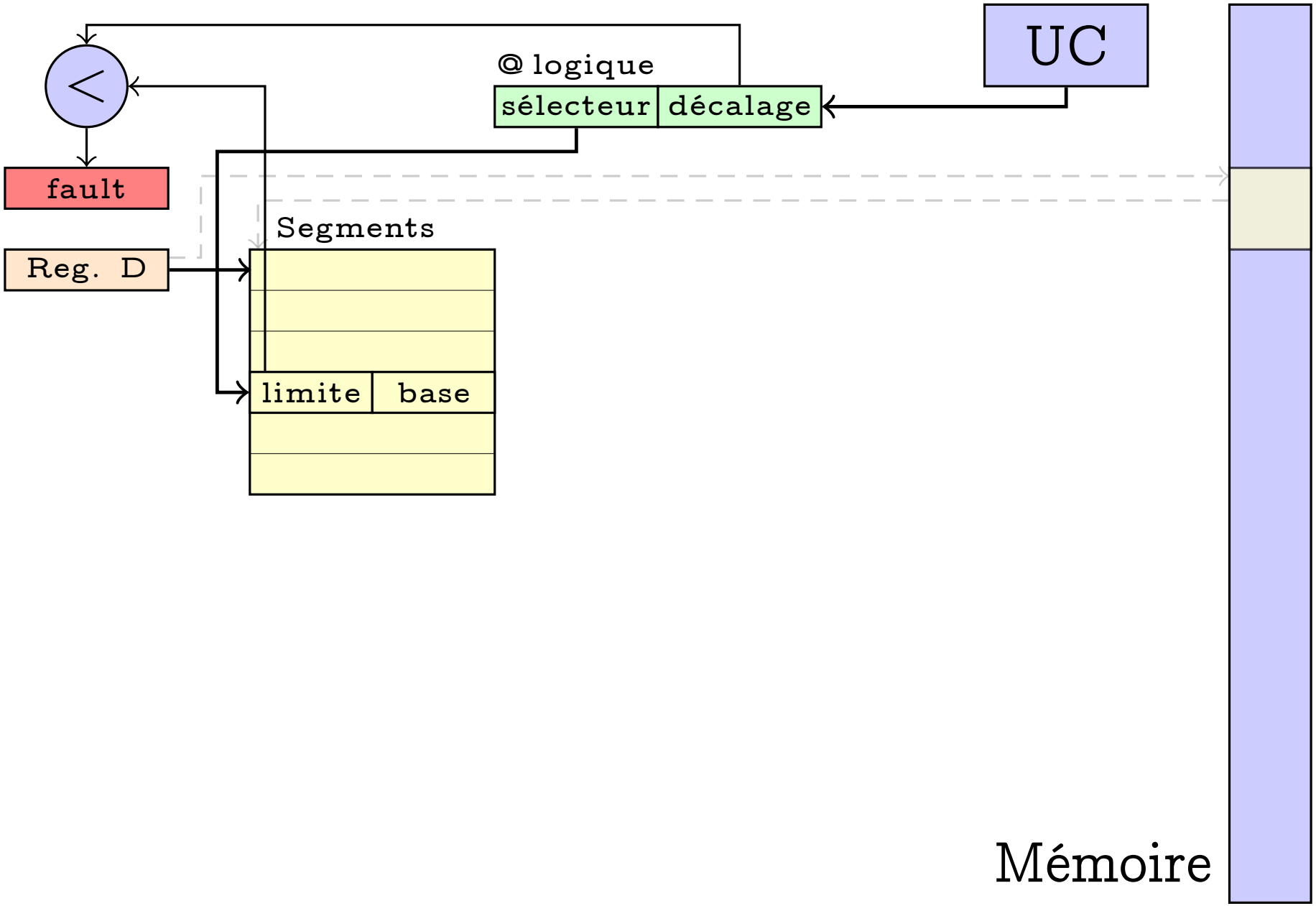
Segmentation



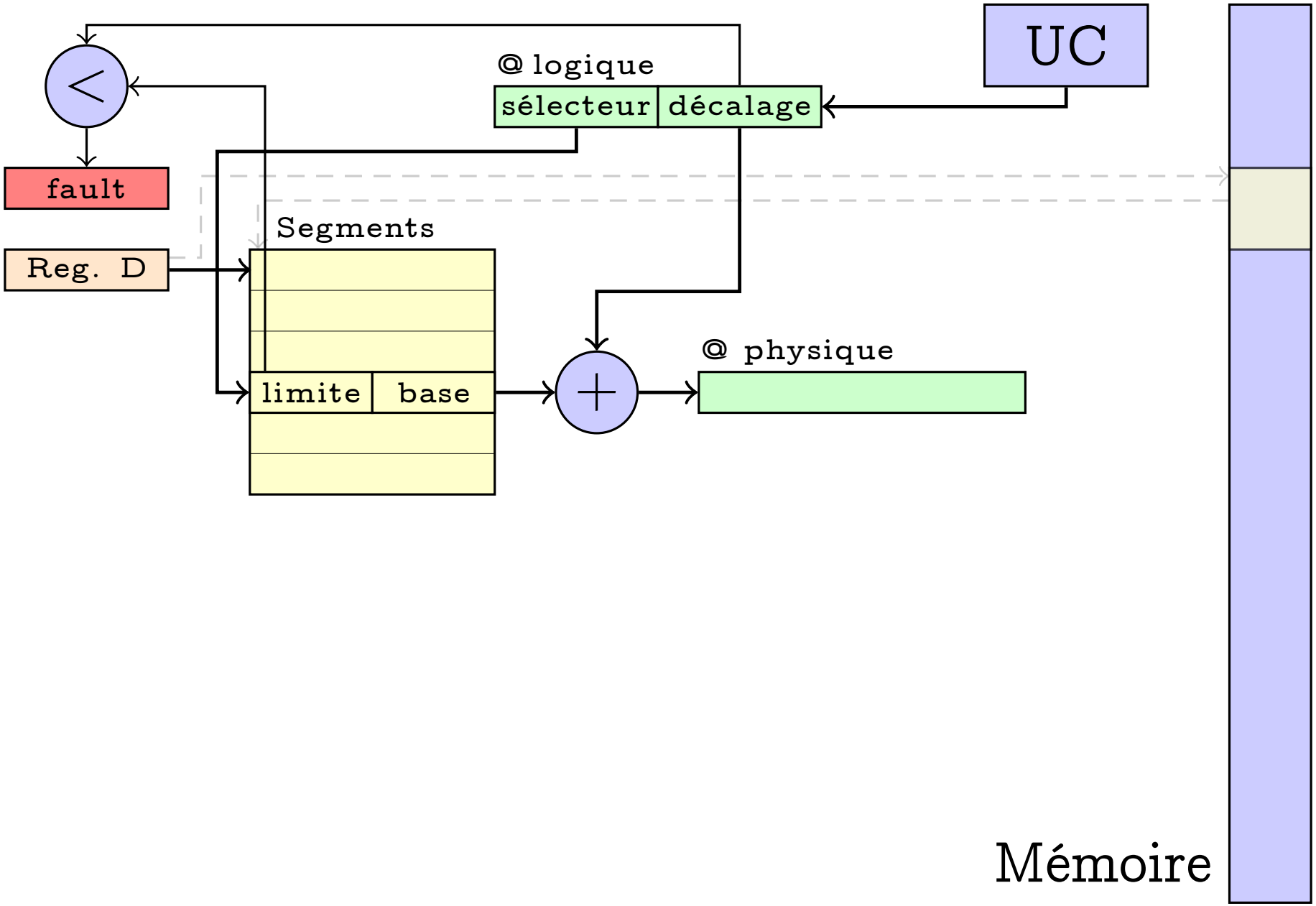
Segmentation



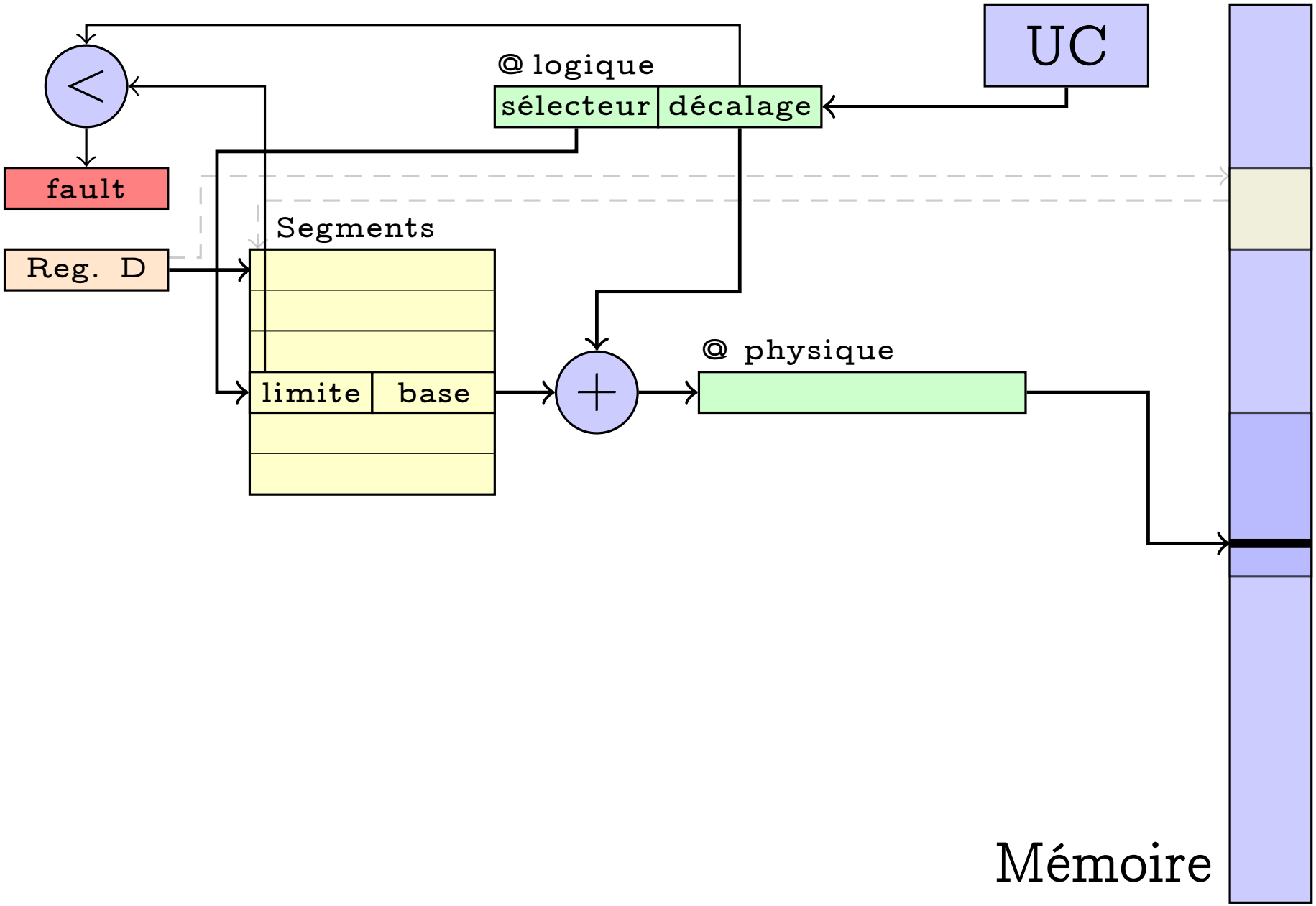
Segmentation



Segmentation



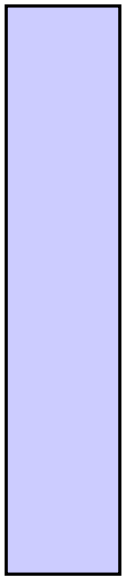
Segmentation



Avantages

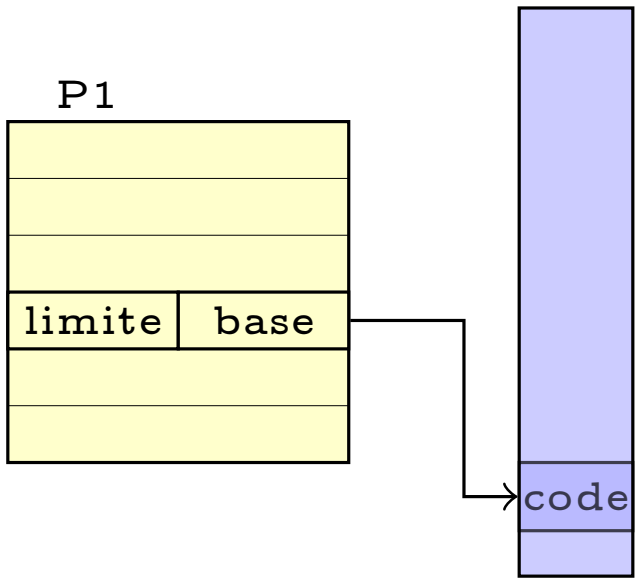
Segments partagés

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



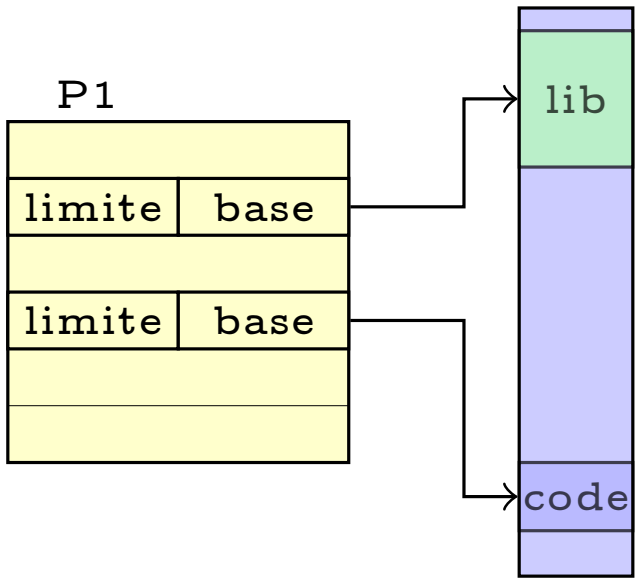
Segments partagés

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



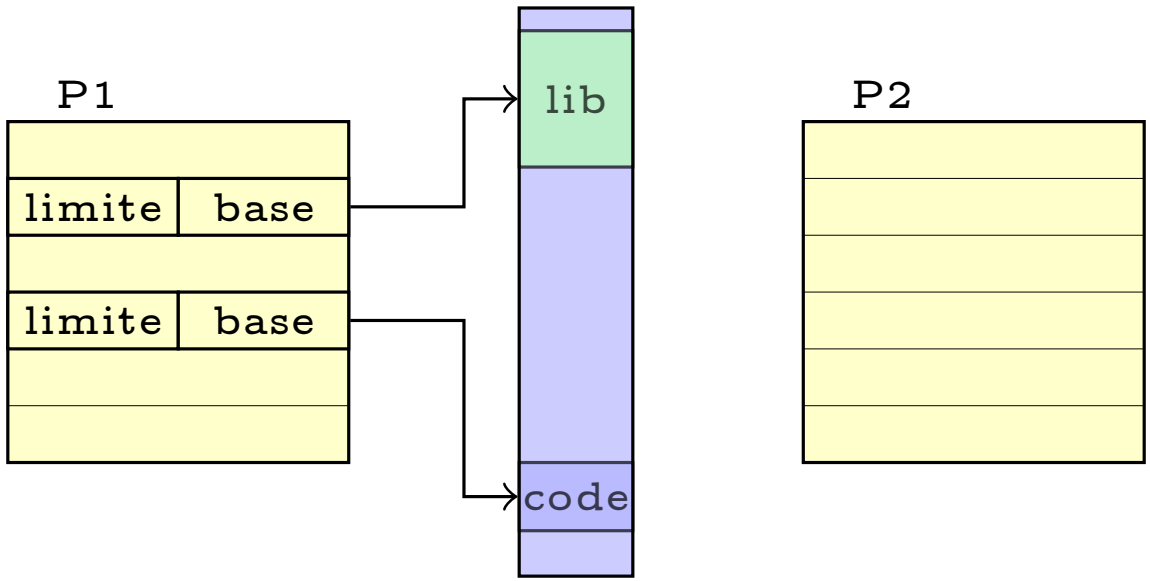
Segments partagés

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



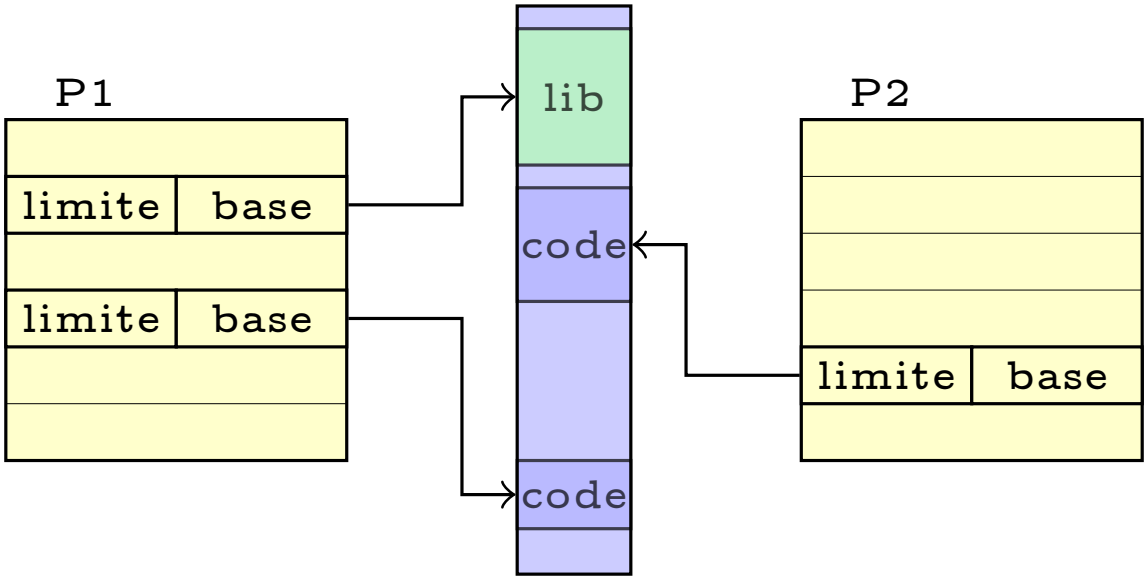
Segments partagés

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



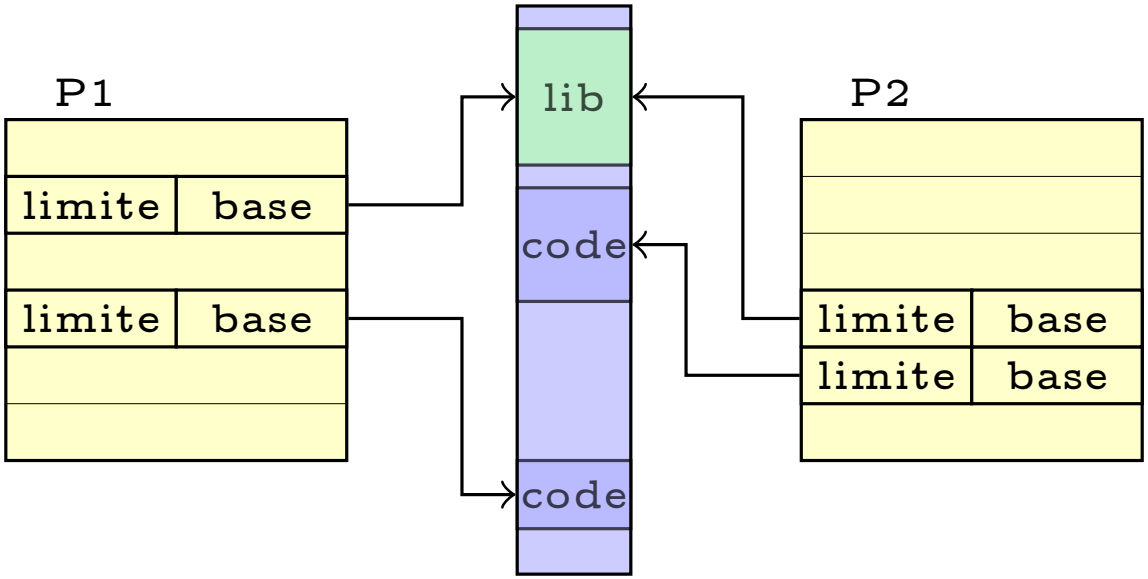
Segments partagés

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



Segments partagés

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

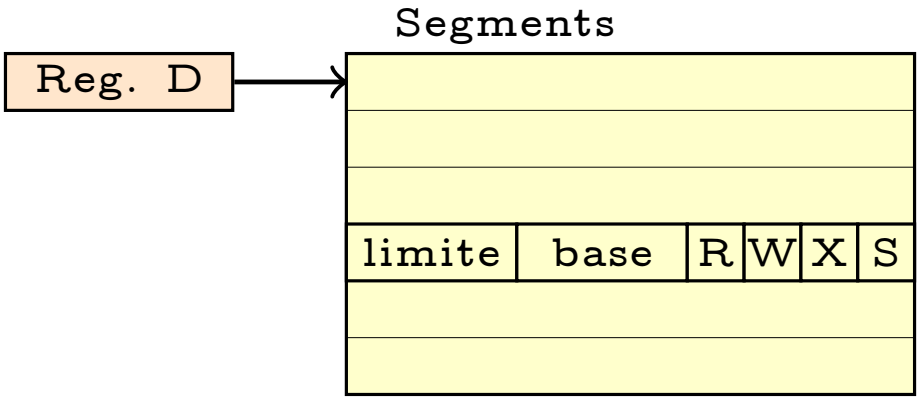


Gestion des droits

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

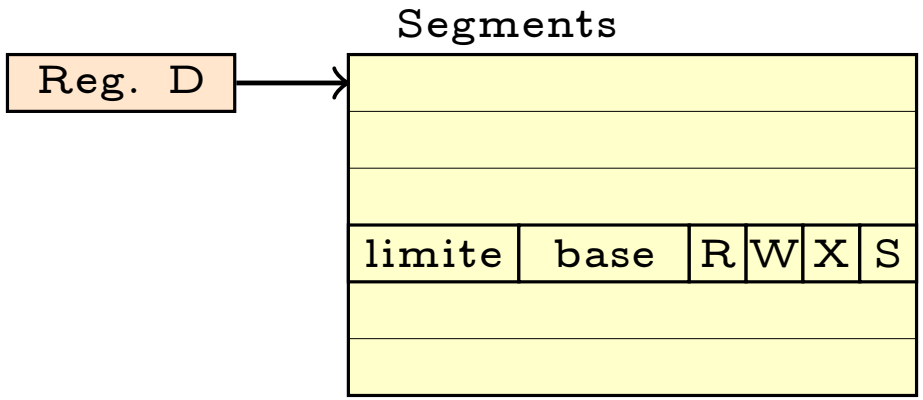
Gestion des droits

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...

Avantages

- ✓ Moins de fragmentation
- ✓ Rapide (géré par la MMU)
- ✓ Partage simple
- ✓ Protection efficace

Inconvénient

Block de tailles variables donc

toujours de la fragmentation.