

# La mémoire virtuelle

- [Définition](#)
  - [Les problèmes de l'allocation mémoire](#)
  - [Correspondance adresses \*virtuelles\* - adresses \*physiques\*](#)
  - [Principes et mécanismes de base de la pagination](#)
  - [La mémoire virtuelle linéaire](#)
  - [Le défaut de page](#)
  - [Le choix d'une victime - remplacement](#)
  - [Le préchargement - La localité](#)
  - [Pagination à deux niveaux](#)
  - [Structure d'un programme](#)
  - [Avantages / Inconvénients de la pagination](#)
  - [La mémoire virtuelle segmentée](#)
  - [Les segments](#)
  - [Problèmes](#)
  - [Le partage de l'information en mémoire virtuelle linéaire](#)
  - [Le partage de l'information en mémoire segmentée](#)
- 

## Définition

Mémoire **virtuelle** = **support** de l'ensemble des informations **potentiellement** accessibles.



Ensemble des emplacements dont l'adresse **peut être** engendrée par le processeur.

≠

Mémoire **physique** = Ensemble des emplacements RAM physiquement présents dans l'ordinateur.

## Pourquoi une mémoire virtuelle ?

Mémoire **physique** coûteuse.

Mémoire **secondaire** (disques, mémoire étendue, ...) peu coûteuse.

Programmes gourmands en mémoire et qui ne "tiennent pas" toujours en RAM.

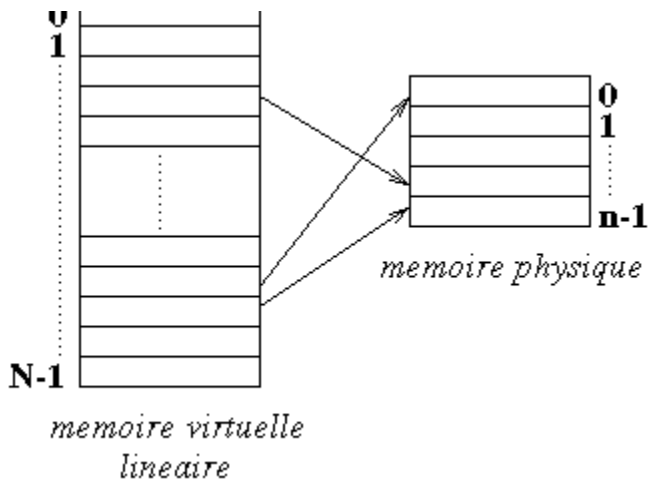


Utiliser la mémoire secondaire "comme" mémoire RAM.

## Idée générale

Il s'agit de conserver en mémoire une "partie" des programmes en cours d'exécution. Si un programme  $A$  veut s'exécuter alors qu'il n'y a plus de place en mémoire, un "bout" d'un autre programme est "viré" en mémoire secondaire et remplacé par un "bout" de  $A$ .

Donc, un programme est découpé en bouts que l'on nomme **pages**, de taille fixe. La mémoire physique est elle aussi découpée en pages, de même taille, ainsi que la mémoire secondaire.



## Les problèmes de l'allocation mémoire

- correspondance entre adresses **virtuelles** et adresses **physiques** ;
- gestion de la mémoire physique ;

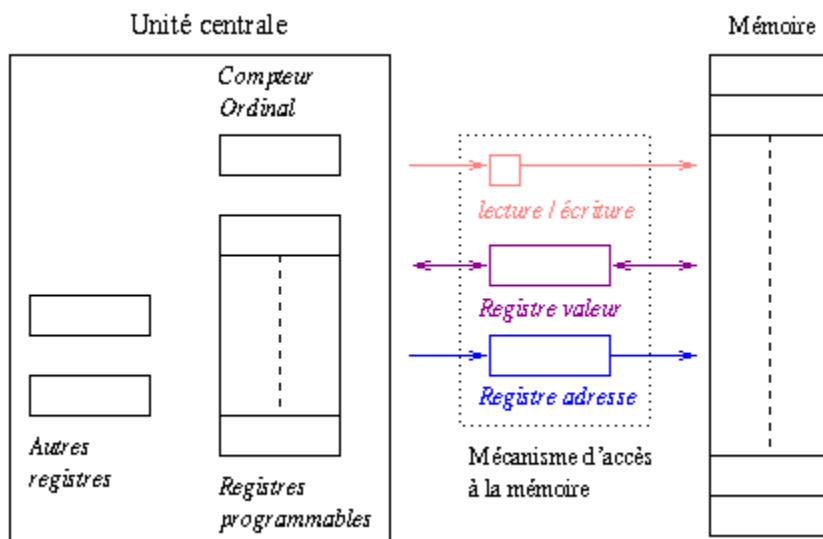
Et si multi-processus :

- partage de l'information ;
- protection mutuelle.

## Correspondance adresses virtuelles - adresses physiques

On utilise une fonction topographique qui associe à une adresse virtuelle, une adresse réelle.

Voici l'architecture classique d'accès à la mémoire :

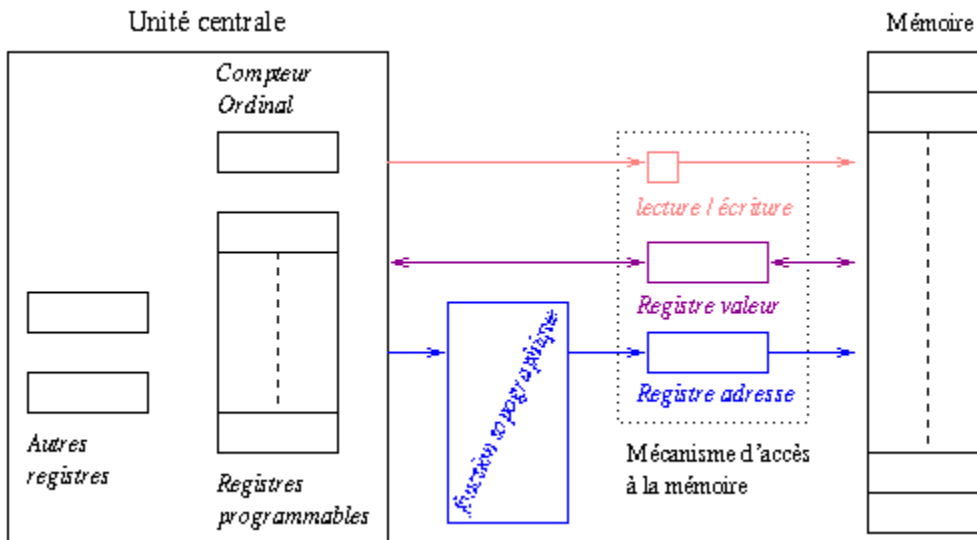


Voici la fonction topographique :

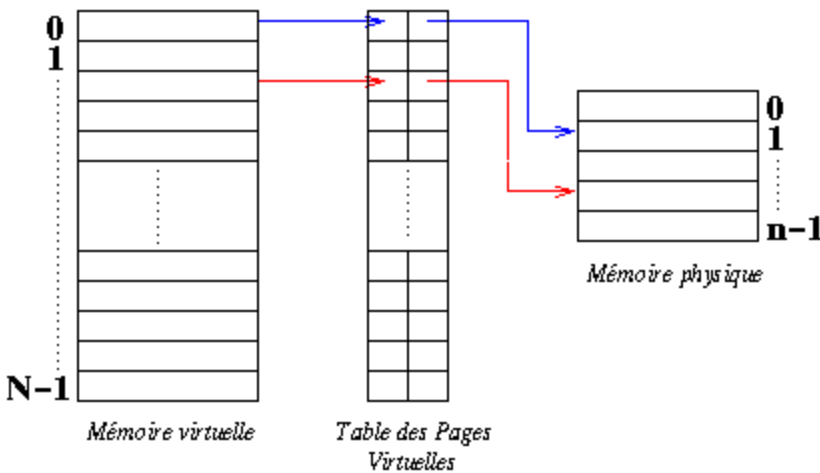
```

Fonction topo(x : adresse_virtuelle) : adresse_reelle;
début
topo := f(x);
fin
  
```

Et voici l'architecture avec réimplantation dynamique :



La mémoire virtuelle, avec sa table des pages, est **une** implémentation possible de la fonction topographique.



## La table des pages virtuelles

Présent	Modifié	Protection	Num page physique
...	...	...	...
oui	non	rx	18
...	...	...	...

- Présent : page virtuelle présente en mémoire physique ?
- Modifié : page modifiée ?
- Protection : droits d'accès.
- Num page physique : page physique correspondante.

La table des pages virtuelles est une implémentation particulière d'une **fonction de pagination** (il en existe d'autres).

En général, c'est un bit dans le PSW (Program Status Word) qui indique si l'on utilise ou non la mémoire virtuelle.

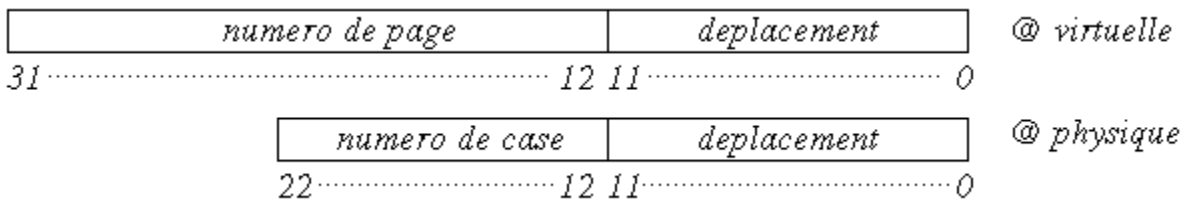
# Principes et mécanismes de base de la pagination

Soit un processeur avec un bus d'adresse sur 32 bits, il peut adresser  $2^{32}$  octets, soit 4 Go. L'ordinateur a une mémoire physique de 8 Mo.

$L$  = taille de la page ou de la case, par exemple 4096 octets, soit  $2^{12}$ .

$N$  = nombre de pages de la mémoire virtuelle, par exemple 1 Méga de pages, soit  $2^{20}$ .

$n$  = nombre de cases de la mémoire physique, par exemple 2048 cases, soit  $2^{11}$ .



## La mémoire virtuelle linéaire

Pourquoi mémoire virtuelle **linéaire** ?

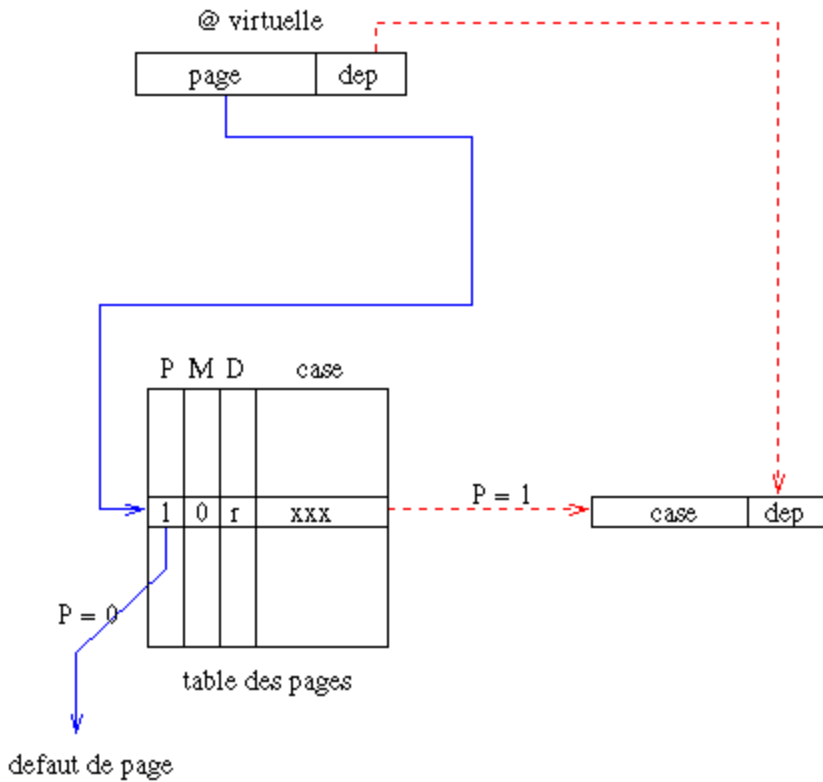
⇨ L'adresse du 1er octet de la page  $n$  = l'adresse du dernier octet de la page  $(n-1) + 1$ .

Avantage : organisation identique à celle d'une mémoire physique.

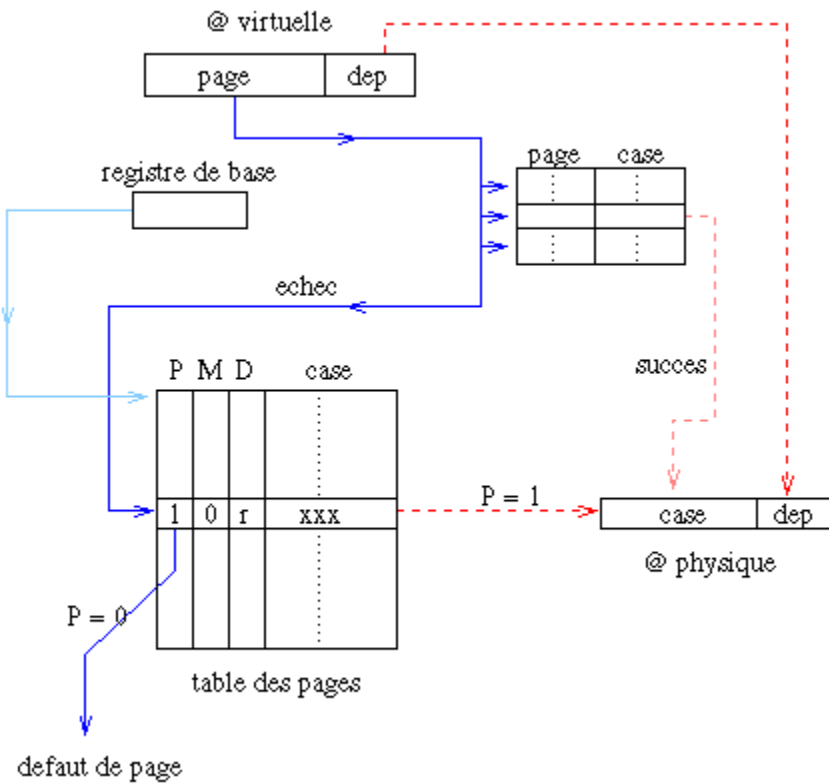
### Adresse Virtuelle ⇨ Adresse Physique

Le calcul de l'adresse réelle à partir de l'adresse virtuelle se réalise ainsi :

- le numéro de page virtuelle donne l'entrée de la TPV dans laquelle se trouve le numéro de page physique ;
- le déplacement est le même (les pages physiques et virtuelles ont la même taille) ;
- si la page virtuelle n'est pas présente en mémoire physique, alors il se produit un *défaut de page*.



Pour accélérer le processus, on utilise des *mémoires associatives* qui recensent les dernières pages utilisées :



## Le défaut de page

L'adresse virtuelle référence une page qui n'est pas présente en mémoire physique. Le mécanisme d'adressage génère un **défaut de page**.

Si la mémoire physique est pleine :

- virer de la mémoire physique une page (**remplacement**) :
  - choisir une page "victime",
  - si elle a été modifiée, la réécrire sur disque,
  - modifier les indicateurs de présence en TPV ;

Puis, dans tous les cas :

- charger la page référencée en mémoire physique (**placement**) ;
- modifier les indicateurs de présence en TPV.

## Le choix d'une victime - remplacement

De nombreux algorithmes :

- **FIFO** - First In First Out : ordre chronologique de chargement ;
- **LRU** - Least Recently Used : ordre chronologique d'utilisation ;
- **FINUFO** - First In Not Used, First Out (algorithme de l'horloge ou Clock) : approximation du LRU ;
- **LFU** - Least Frequently Used ;
- **Random** : au hasard ;
- **MIN** : algorithme optimal.

Performances : MIN, LRU, FINUFO, [FIFO, Random].

## Le préchargement - La localité

Optimisation du système : tenir compte de la **localité** en **préchargeant** des pages **avant** d'en avoir besoin.

**Localité** : à un instant donné, les références observées dans un passé récent sont (en général) une bonne estimation des prochaines références.

En moyenne, 75% des références intéressent moins de 20% des pages. C'est la **non-uniformité**.

⇒ on va essayer d' **anticiper** la demande.

## Le problème de la taille de la TPV

Pour être utilisée, la TPV doit être placée en mémoire physique.

Par exemple, si on a  $2^{20}$  pages virtuelles, la TPV aura une taille d'à peu près  $2^{20} * \text{taille d'une entrée} = 10 \text{ Mo}$  si une entrée tient sur 10 octets.

C'est à dire plus que la taille de la mémoire physique !!!!

Solution : ⇒ on va paginer la TPV.

## Pagination à deux niveaux

La mémoire virtuelle est divisée en **Hyperpages** qui sont elles-mêmes divisées en **pages**.

Une adresse virtuelle = numéro d'hyperpage ; numéro de page ; déplacement.

Attention ! L'accès à la mémoire est plus lent d'une indirection (utilisation de mémoires associatives).

La mémoire est toujours **linéaire**.

## Structure d'un programme

Un programme destiné à être chargé en mémoire virtuelle se décompose ainsi :

- En-tête : < nom, taille (en nombre de pages) >
- Table de couplage :
  - < page\_virtuelle, nombre\_de\_pages, texte > (texte est en général une référence à un secteur disque qui contient le code)
  - < page\_virtuelle', nombre\_de\_pages', texte' >
  - ...
- Table des points d'entrées :
  - < Entrée, adresse\_exec >
  - < Entrée, adresse\_exec >
  - ...

**Problème** : le code d'un programme est "absolu", c'est à dire qu'on ne peut pas le déplacer dans la mémoire virtuelle (il a une adresse fixe).

**Solution** : posséder une fonction de couplage dynamique et un vecteur de translation. Mais cela coûte cher en performance.

Pour éviter les conflits d'implantation en mémoire virtuelle et la fragmentation, on associe une mémoire virtuelle à chaque utilisateur.

## Avantages / Inconvénients de la pagination

Avantages :

- Meilleure utilisation de la mémoire physique (programmes implantés par fragments, dans des pages *non-consécutives*).
- Possibilité de ne charger des pages que lorsqu'elles sont référencées (chargement à la demande).
- Indépendance de l'espace virtuel et de la mémoire physique (mémoire virtuelle généralement plus grande).
- Possibilité de ne vider sur disque que des pages modifiées.
- Possibilité de recouvrement dynamique (couplage).

Inconvénients :

- Fragmentation interne (toutes les pages ne sont pas remplies).
- Impossibilité de lier deux (ouo plusieurs) procédures liées aux mêmes adresses dans l'espace virtuel.

## Mémoire virtuelle segmentée

La mémoire virtuelle = ensemble de segments.

Un segment = suite d'emplacements **consécutifs**.

Adresse virtuelle = numéro de segment ; déplacement.

En général, les segments correspondent à un découpage **logique** d'un programme (segment de pile, de

données, de code, ...).

L'algorithme de remplacement remplace un (ou plusieurs) segments entiers par le segment référencé.

Évidemment, il existe une table des segments.

## Les segments

<b>nom du segment</b>
<b>taille</b>
<b>droits</b>
<b>contenu</b>
<b>options</b>
<b>debut du segment</b> →

**Options** : le segment doit-il rester le plus longtemps possible en mémoire physique ?

**Contenu** : donne des informations à l'algorithme de remplacement. Si contenu = données modifiées → écriture sur disque (cher). Si code → rien à faire.

## Problèmes

Un segment = des zone de mémoire **contigües**, d'où une gestion **difficile** de la mémoire (utilisation de "ramasse-miettes" - garbage collector).

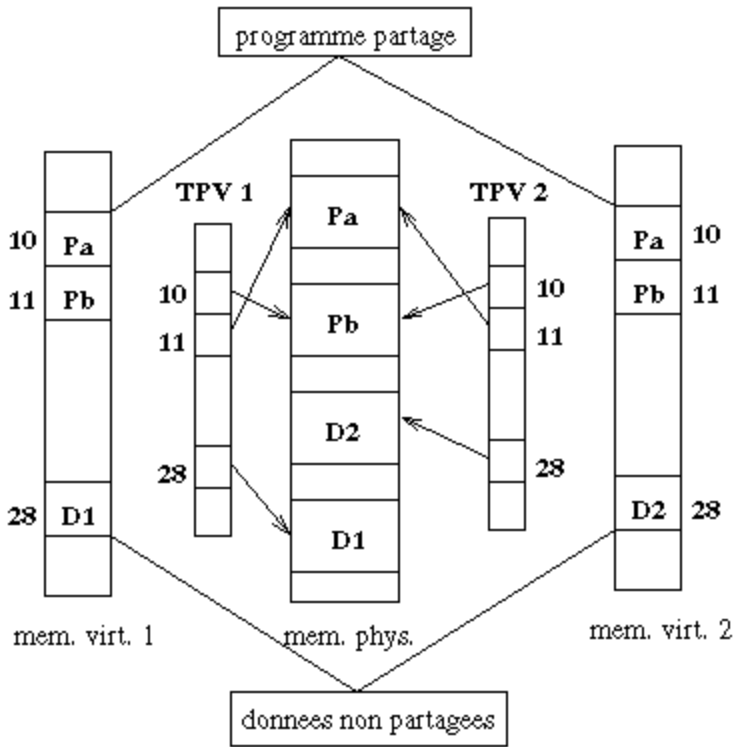
Idée : Paginer les segments !

On a une TPV par segment.

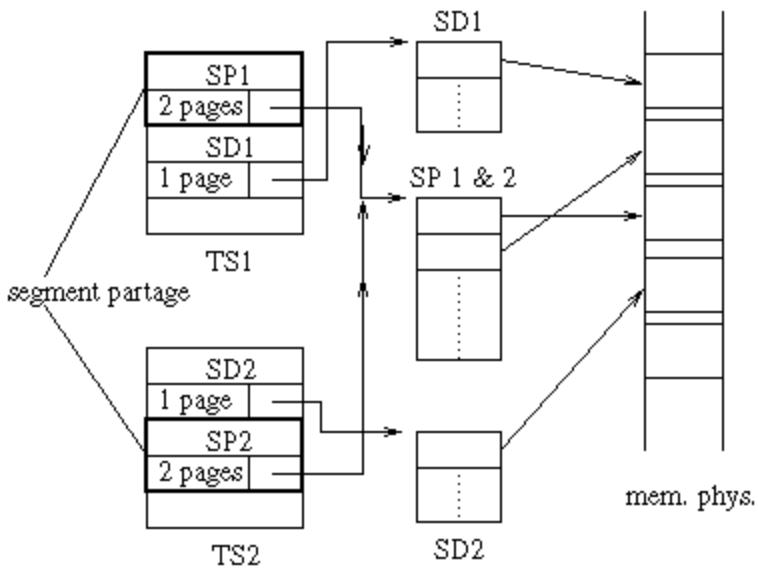
Une adresse segmentée = numéro de segment + déplacement dans le segment.

Un déplacement dans le segment = numéro de page virtuelle + déplacement dans la page.

## Le partage de l'information en mémoire virtuelle linéaire



## Le partage de l'information en mémoire segmentée



[Retour au sommaire.](#)