Systèmes d'exploitation - Pagination

Pierre Gançarski

BUT Informatique - S31

ATTENTION
Ces transparents ne sont qu'un guide du cours : de nombreuses explications et illustrations manquent.

De nombreuses précisions seront données au tableau et à l'oral pendant le cours.

Université de Strasbourg

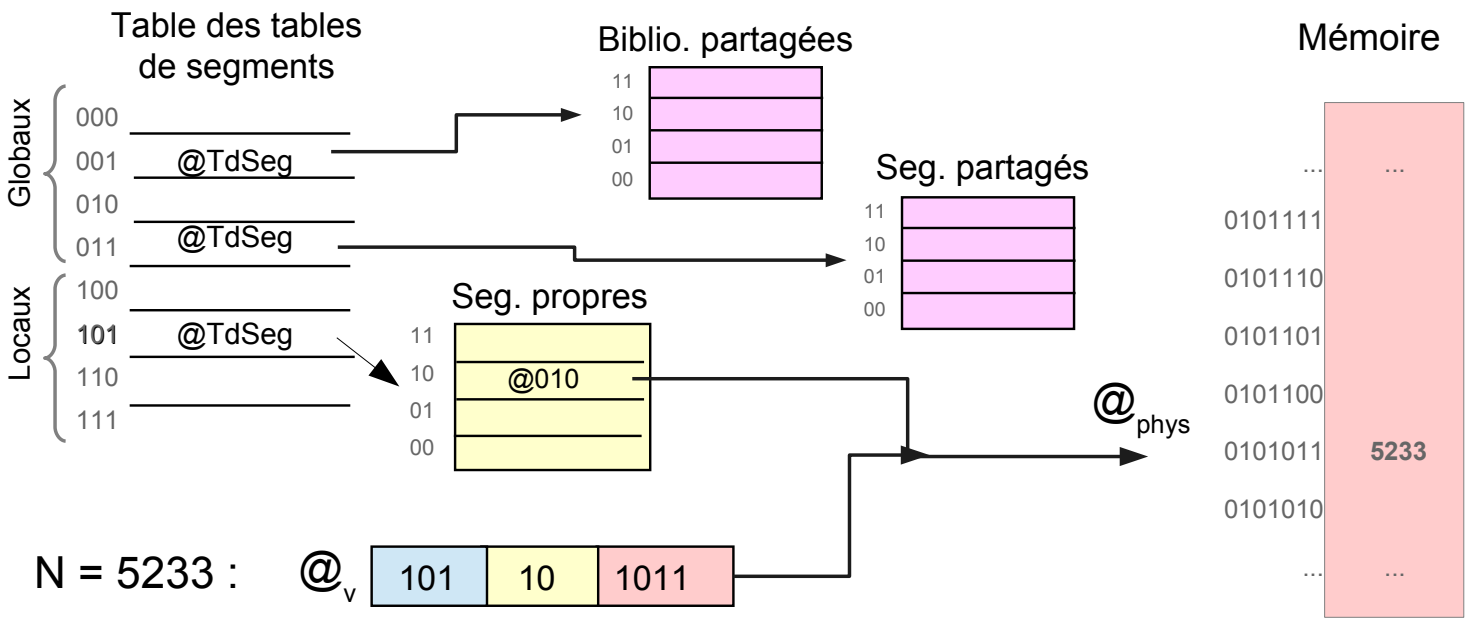
IUT Robert Schuman
UNIVERSITÉ DE STRASBOURG

Plan

La segmentation (Rappel)

Cas d'Unix

- Les tables de segments sont référencées dans une table "système" : chaque processus dispose d'entrées dans cette table
- Les tables donnent les adresses de relocation
- Tab. de seg. globaux –> partage de segments : bibliothèques, mémoires partagées
- Tab. de seg. locaux –> espace d'adressage propre aux processus



La segmentation

Pb1 : Fragmentation externe

- Les segment sont swappés en bloc
- Les segments ne sont pas de taille fixe, et peuvent croître en cours d'exécution (augm. de la limite).

$\Rightarrow$ Phénomène de **fragmentation externe** : création de nombreuses petites zones de mémoire libre pas assez grandes pour stocker un segment.

- Compactage : très coûteux.

Limites physiques et mémoire virtuelle

Pb2 : Taille maximale des processus

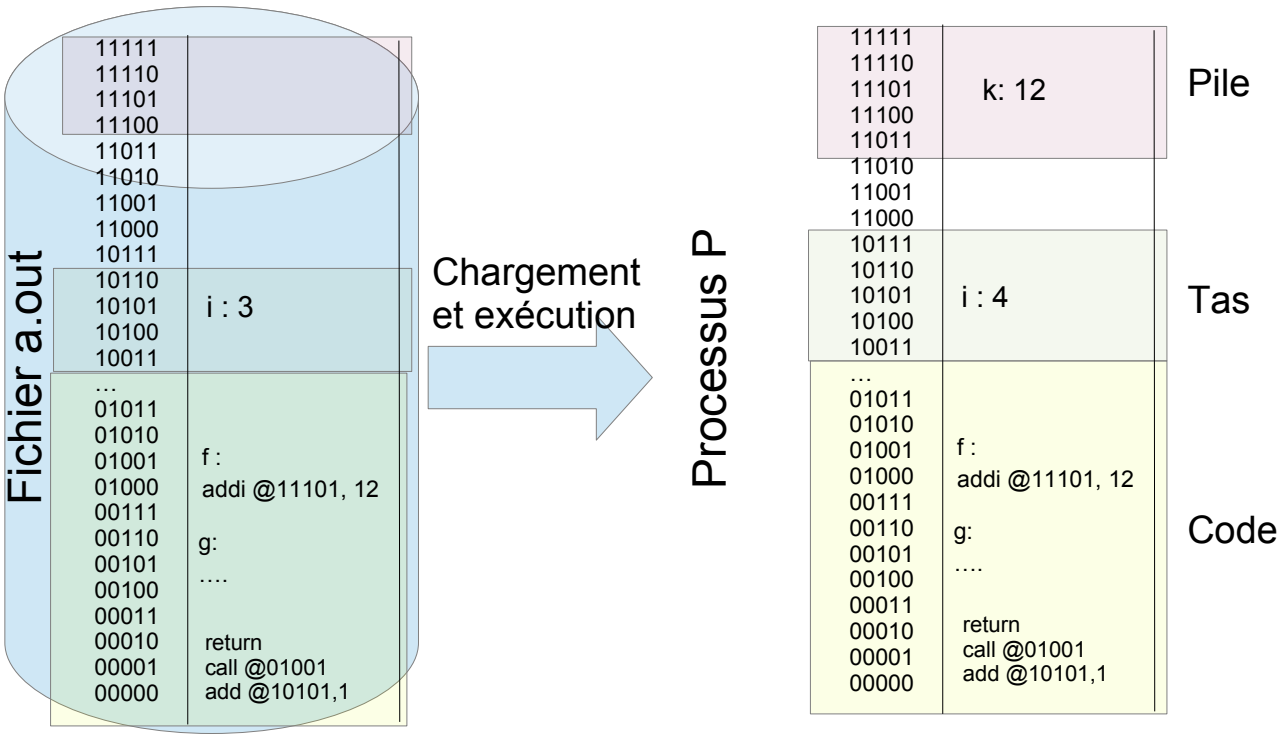
- L'adresse virtuel d'un processus est souvent plus grand que la taille réelle de la mémoire :
  - Adresse virtuelle sur 64 bits :
    $2^{64} octets = 2^4 \times 2^{60} = 16 Eo \approx 16 \times 10^{18}$ octets
  - Adresse réelle (physique) sur 36 bits :
    $2^{36} octets = 2^6 \times 2^{30}$ octets soit 64Go
- Comment autoriser des processus dont l'espace d'adressage est plus grand que la quantité de mémoire physique effectivement présente ?

$\rightarrow$ Les systèmes de mémoire virtuelle s'appuient généralement sur la technique de la *pagination*.

Structure d'un processus

Rappel : Exécution d'un processus

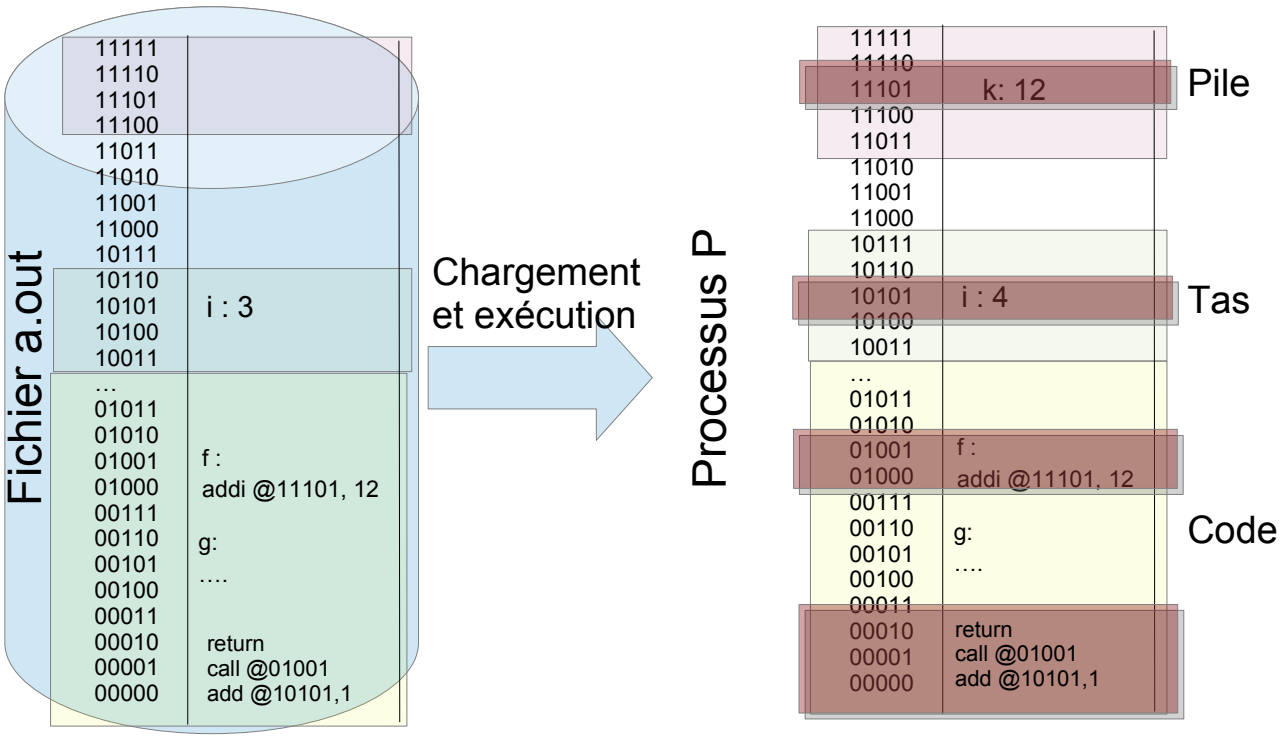
- Exécution d'un processus
  - Le fichier exécutable (a.out) est chargée en mémoire
  - les instructions se déroulent une à une
- Chaque processus travaille sur son propre espace d'adressage, indépendant des autres.



Exécution d'un processus

On remarque

- que peu de mémoire a été réellement utilisé
- que l'espace d'adressage entre 11000 et 11010 n'a pas servi
- que la partie du code correspondant à g() n'a pas été utilisée



Exécution d'un processus

Deux questions

- Comment pouvoir charger que les parties réellement utilisées d'un processus ?
- Comment pouvoir charger plusieurs processus (ou parties de processus) différents en mémoire ?

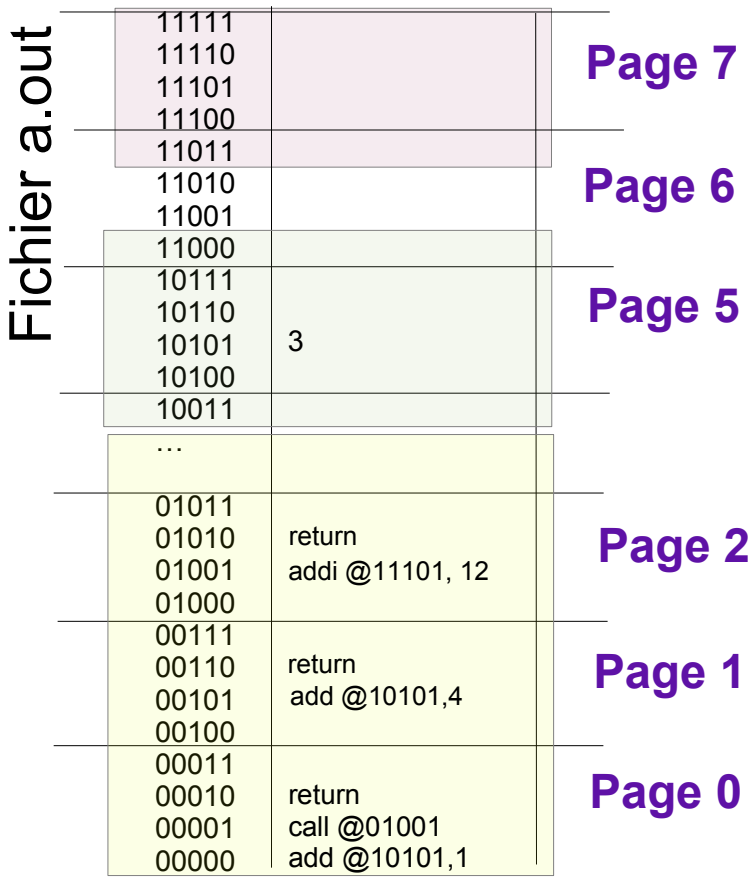
$\rightsquigarrow$ Pagination

Plan

Page

Principe

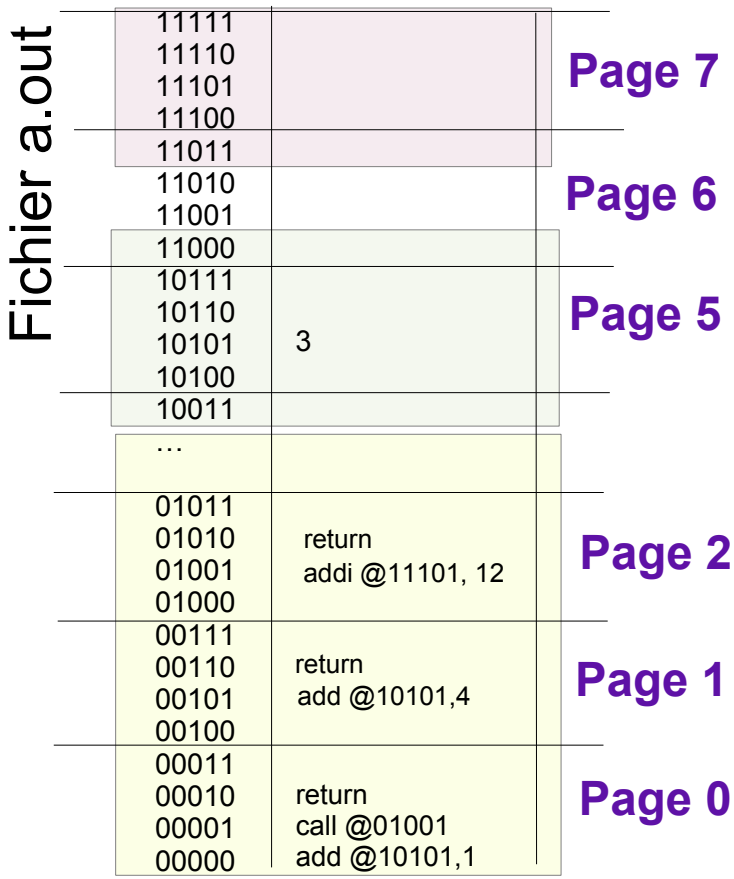
- Les processus sont virtuellement découpés en **pages**
  - Unités logiques de taille fixe (ex : 4 octets)

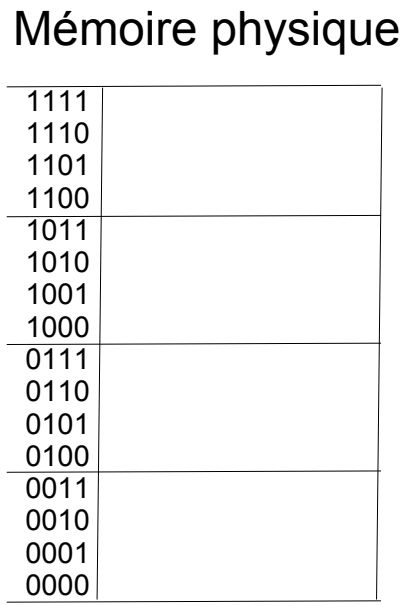


Pagination

Déroulement de l'exemple

- Lors du lancement du processus, le compteur ordinal est mis à 0



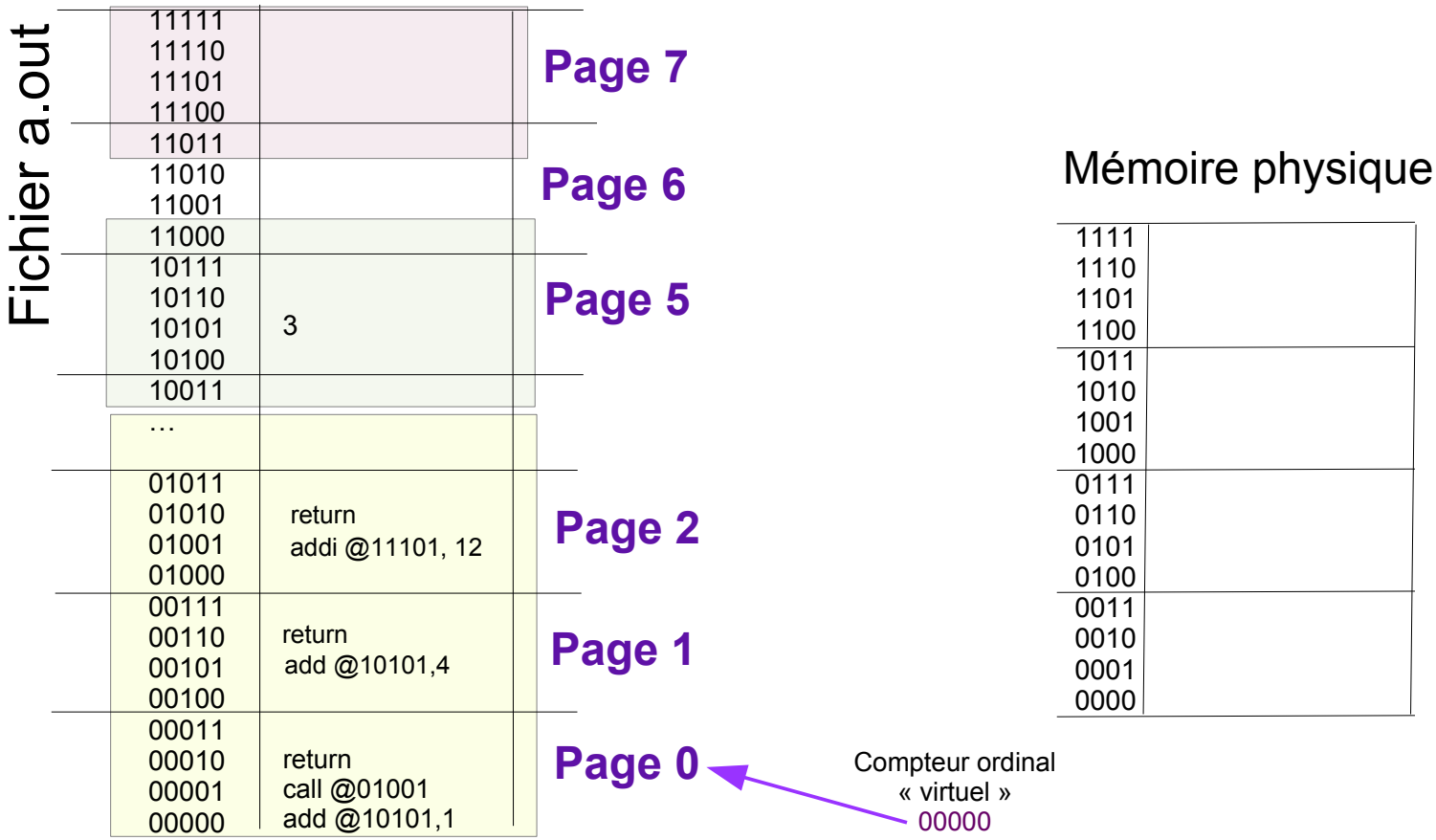


Compteur ordinal
« virtuel »
00000

Pagination

Déroulement de l'exemple

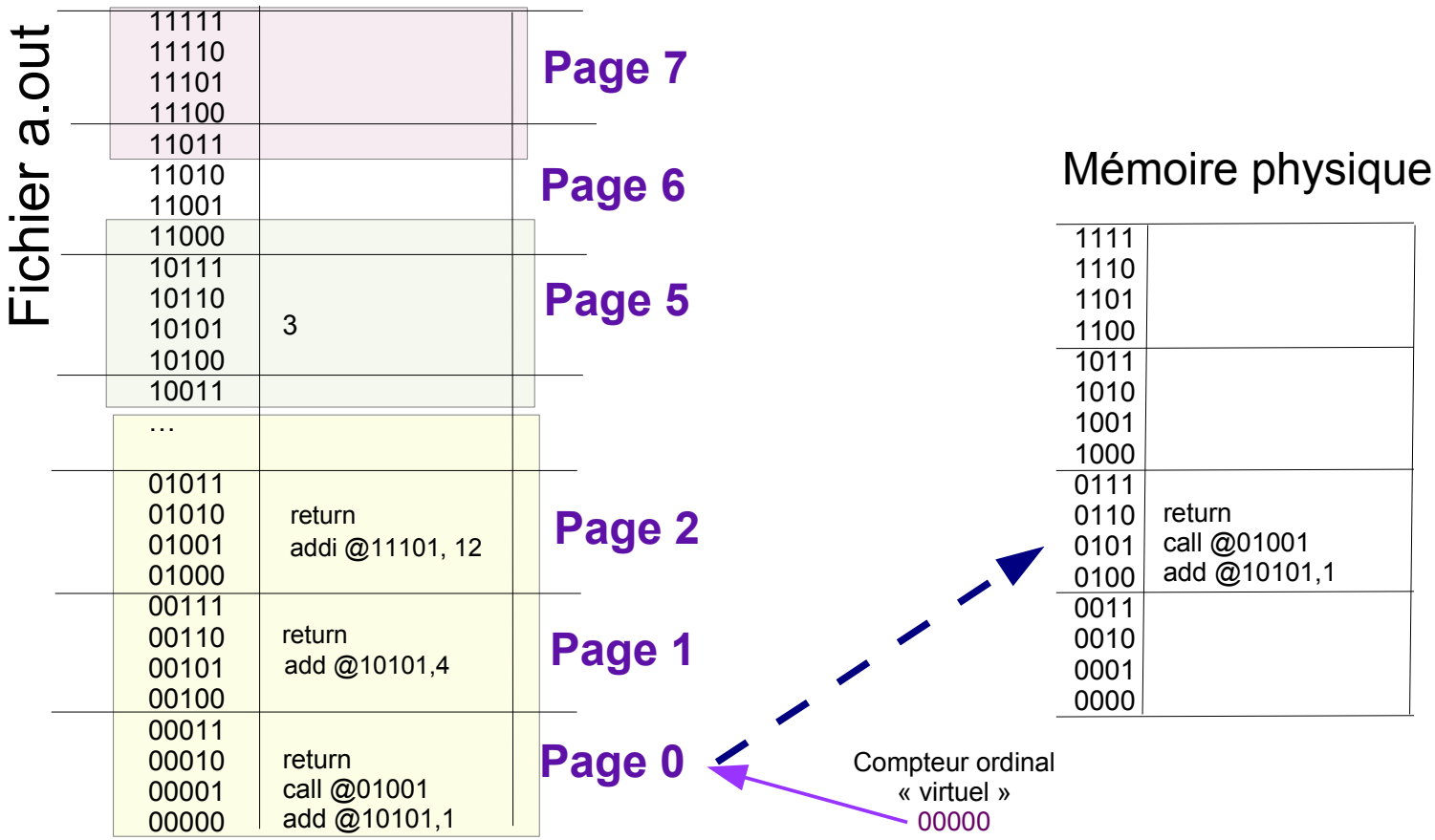
- Calcul du numéro de page contenant l'instuction 0



Pagination

Déroulement de l'exemple

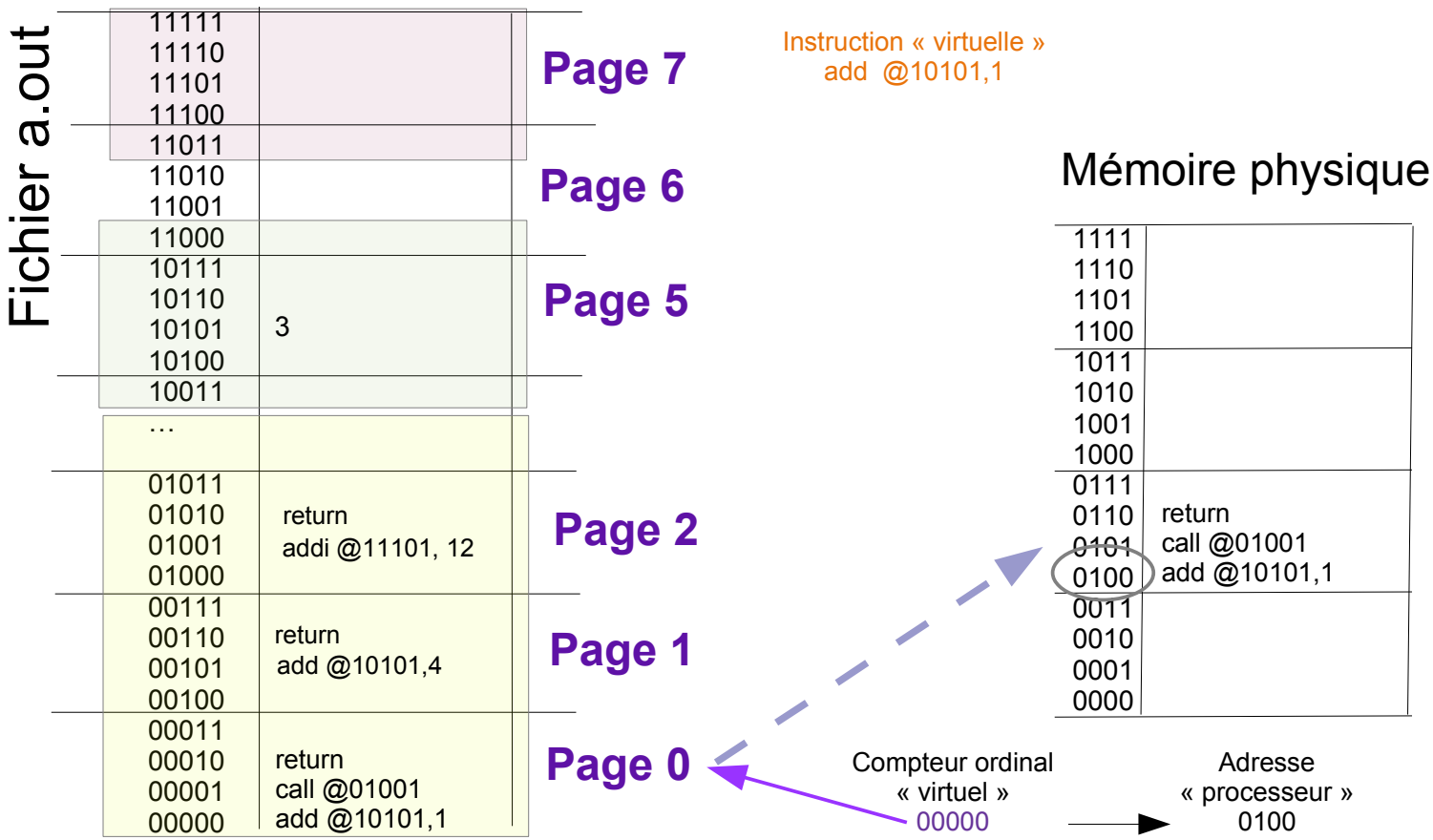
- Chargement de la page de code contenant le `main` dans le cadre #1



Pagination

Déroulement de l'exemple

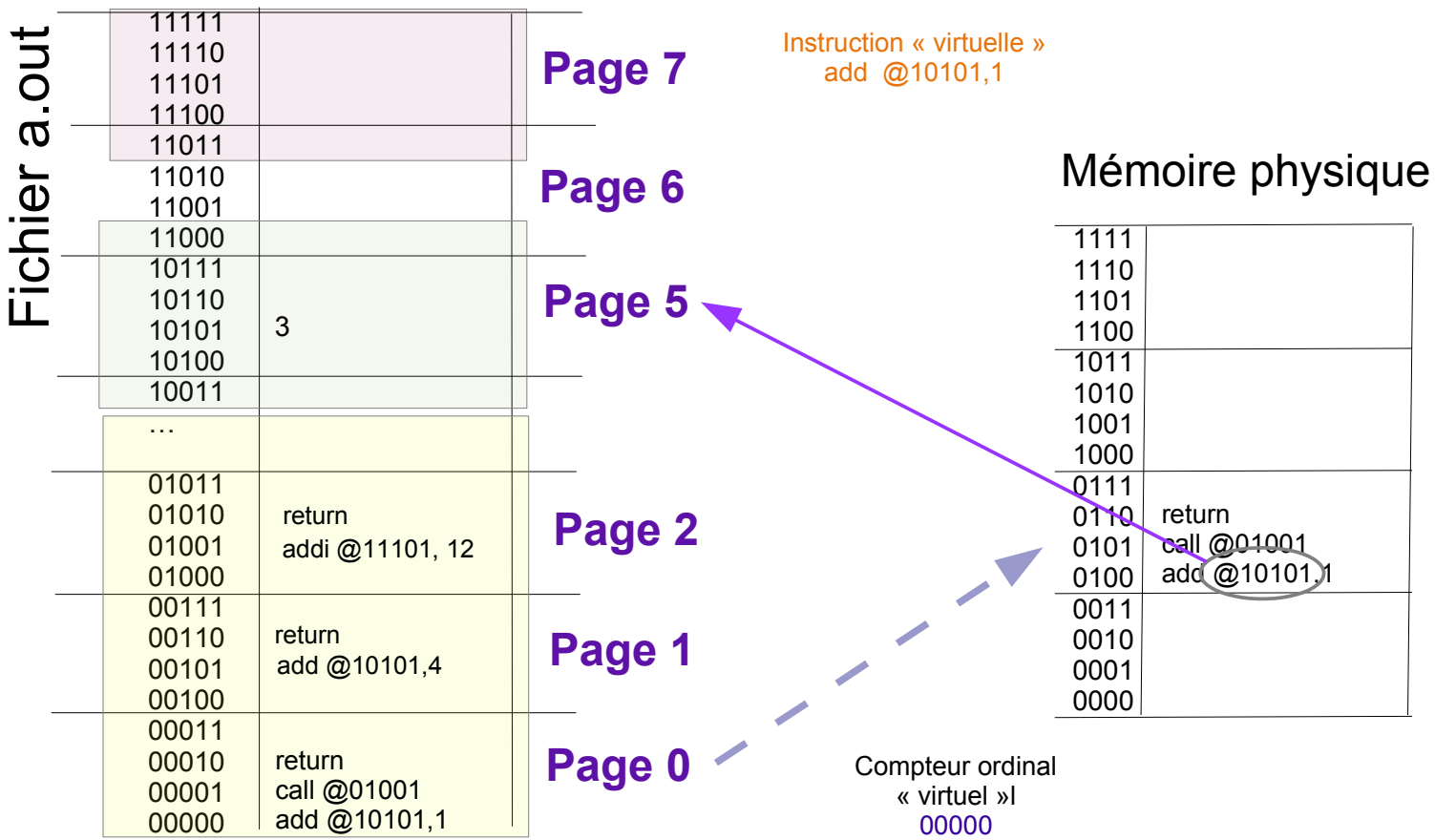
- Il y a maintenant une adresse "physique" pour l'instruction : le processus peut la charger



Pagination

Déroulement de l'exemple

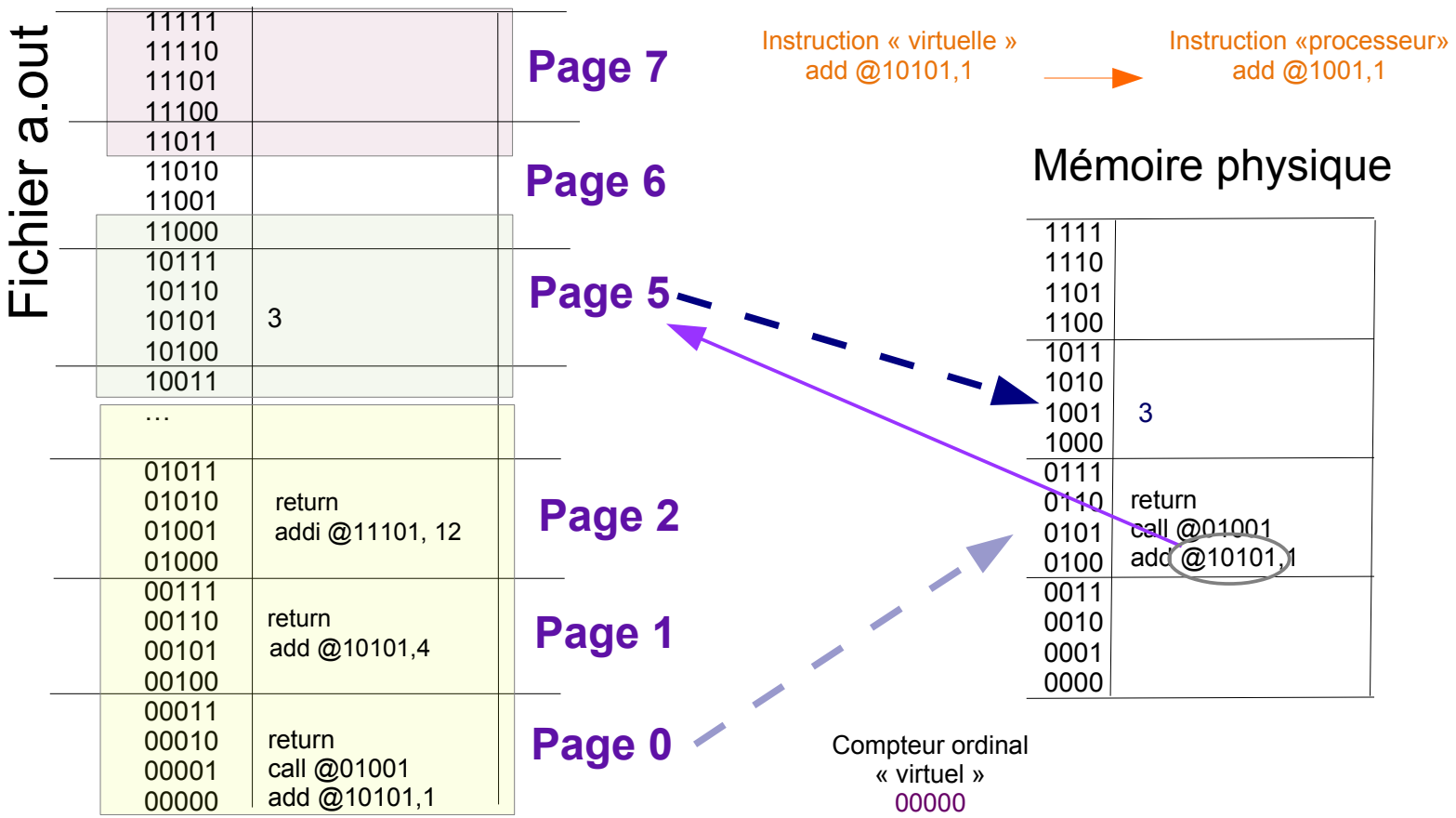
- Pour exécuter l'instruction, il faut charger la page correspondante à l'adresse @10101. Le calcul donne 5 : cette page n'est pas chargée dans un cadre → **Défaut de page**



Pagination

Déroulement de l'exemple

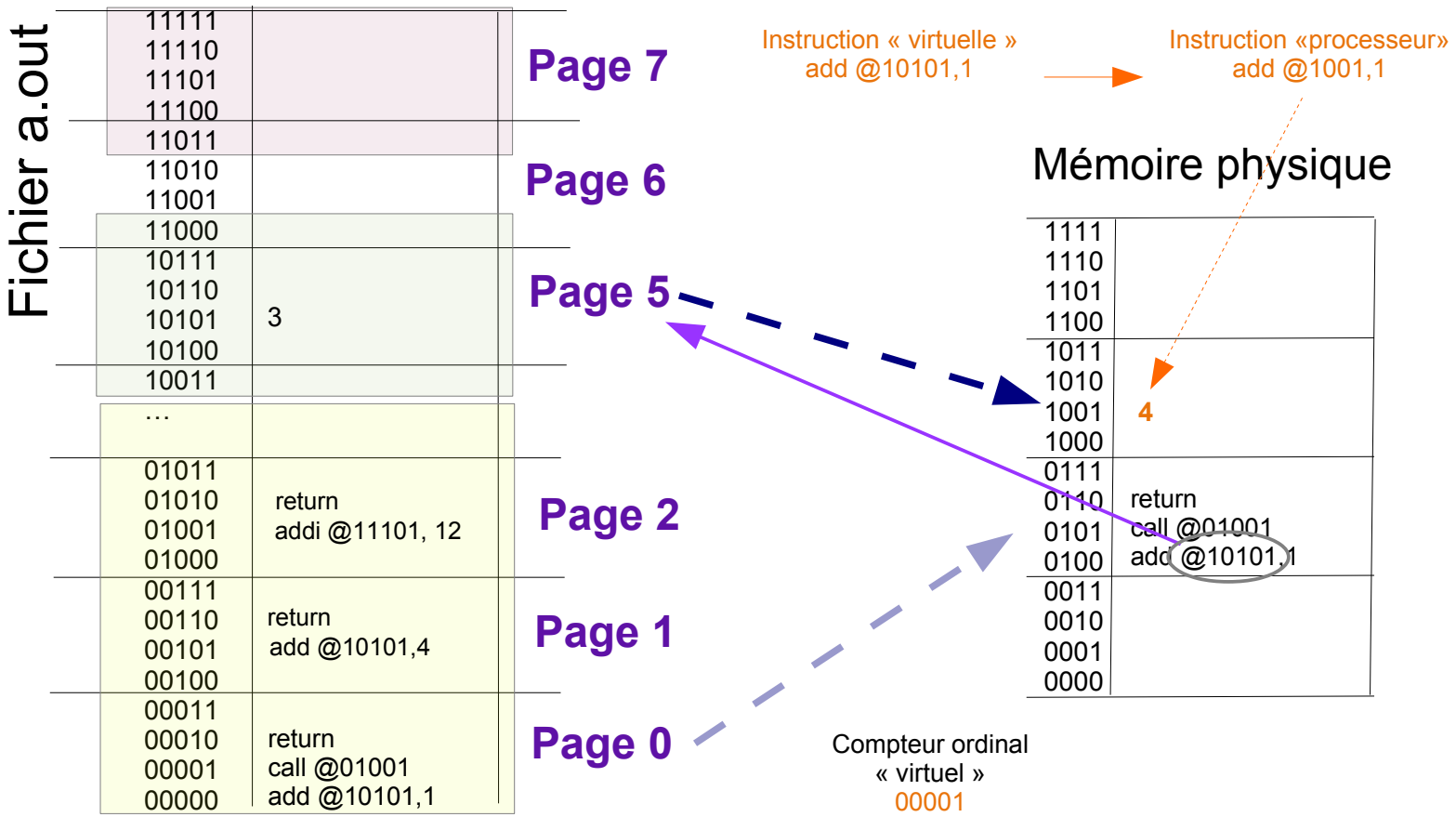
- La "vraie" instruction est add @1001,1



Pagination

Déroulement de l'exemple

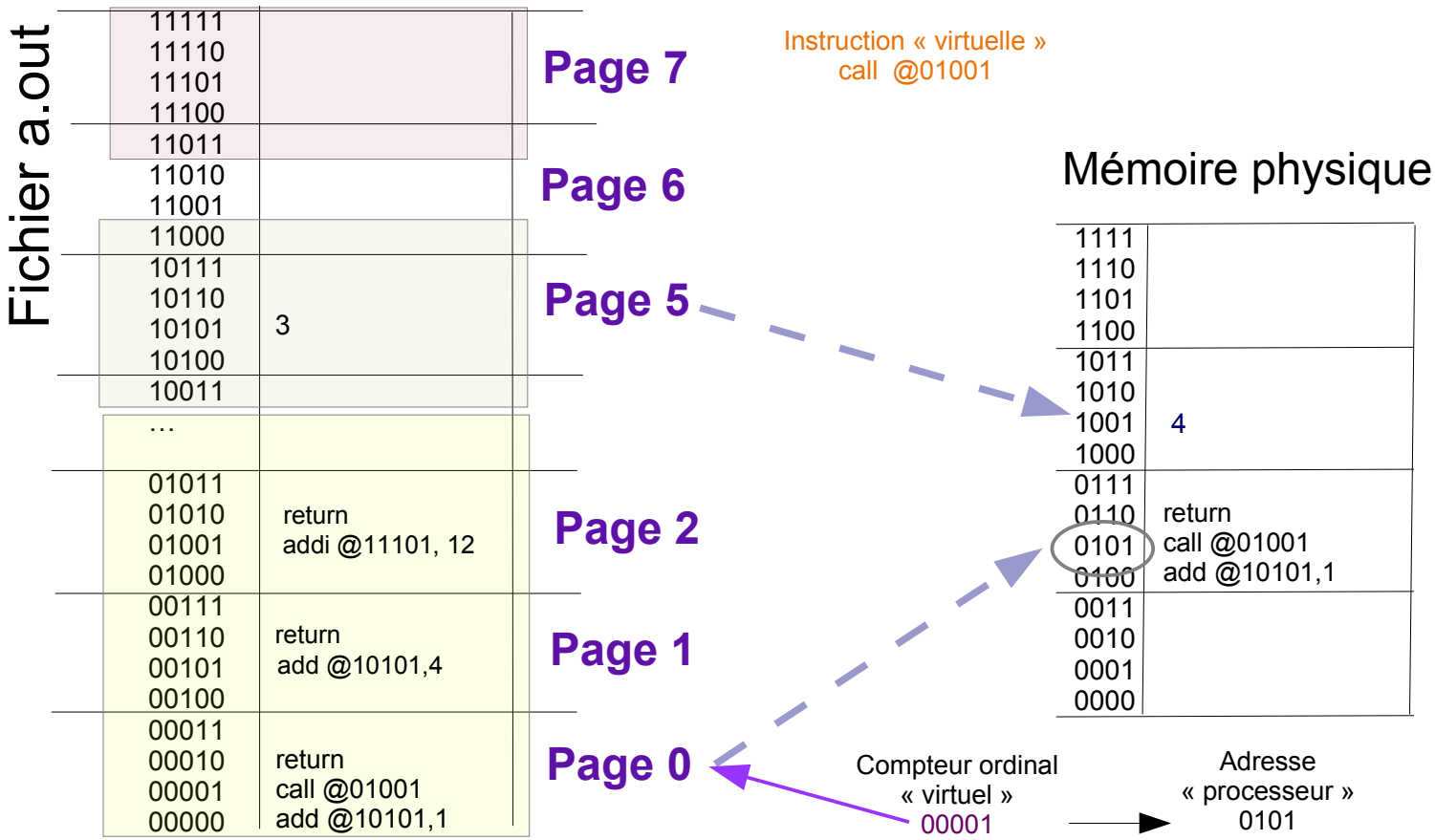
- La "vraie" instruction est exécutée par le processeur → La valeur de `i` est modifiée
- Le compteur ordinal est mis à jour (incrémenté)



Pagination

Déroulement de l'exemple

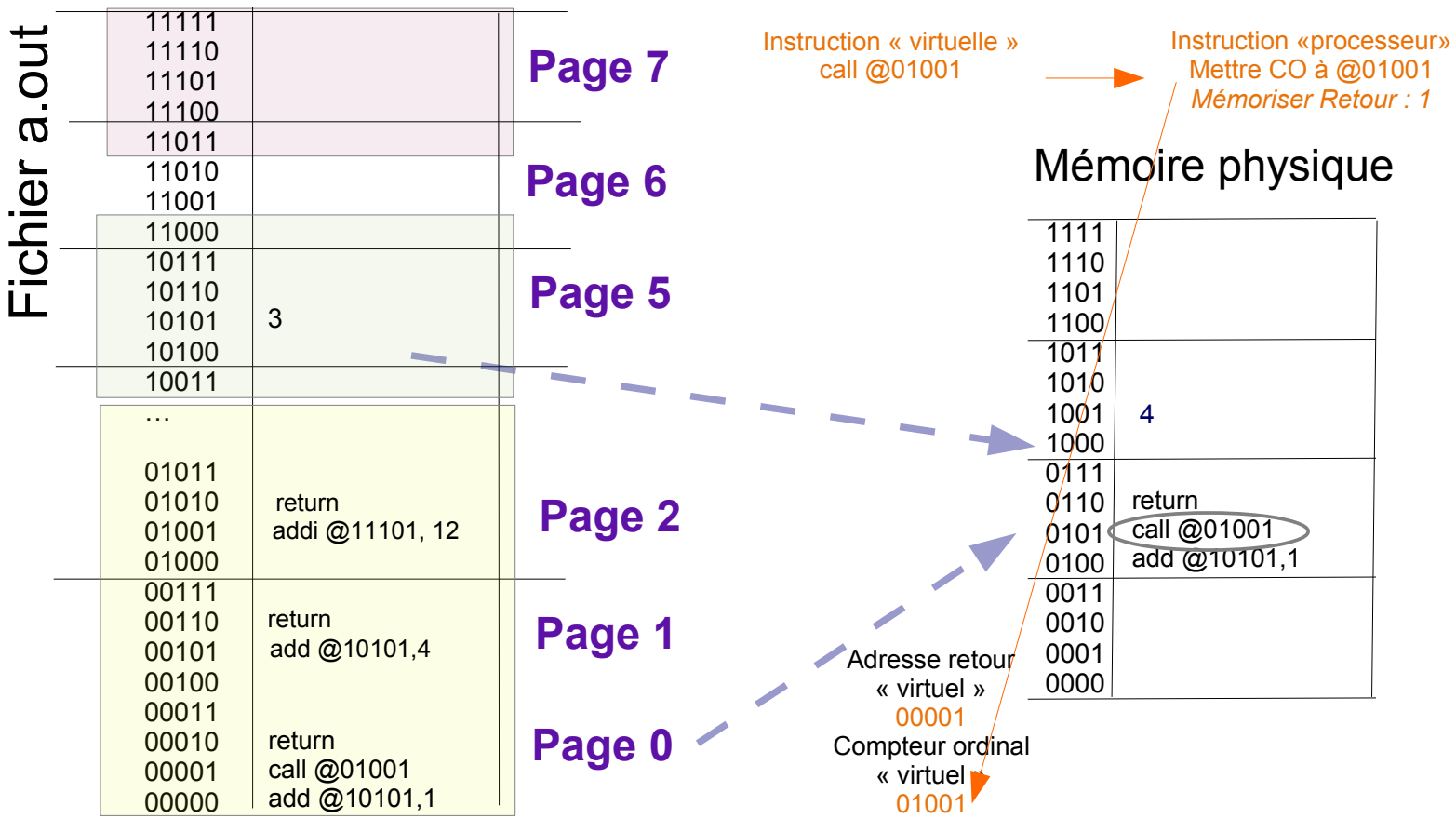
- Calcul de la page contenant l'instruction #1 → Page déjà présente



Pagination

Déroulement de l'exemple

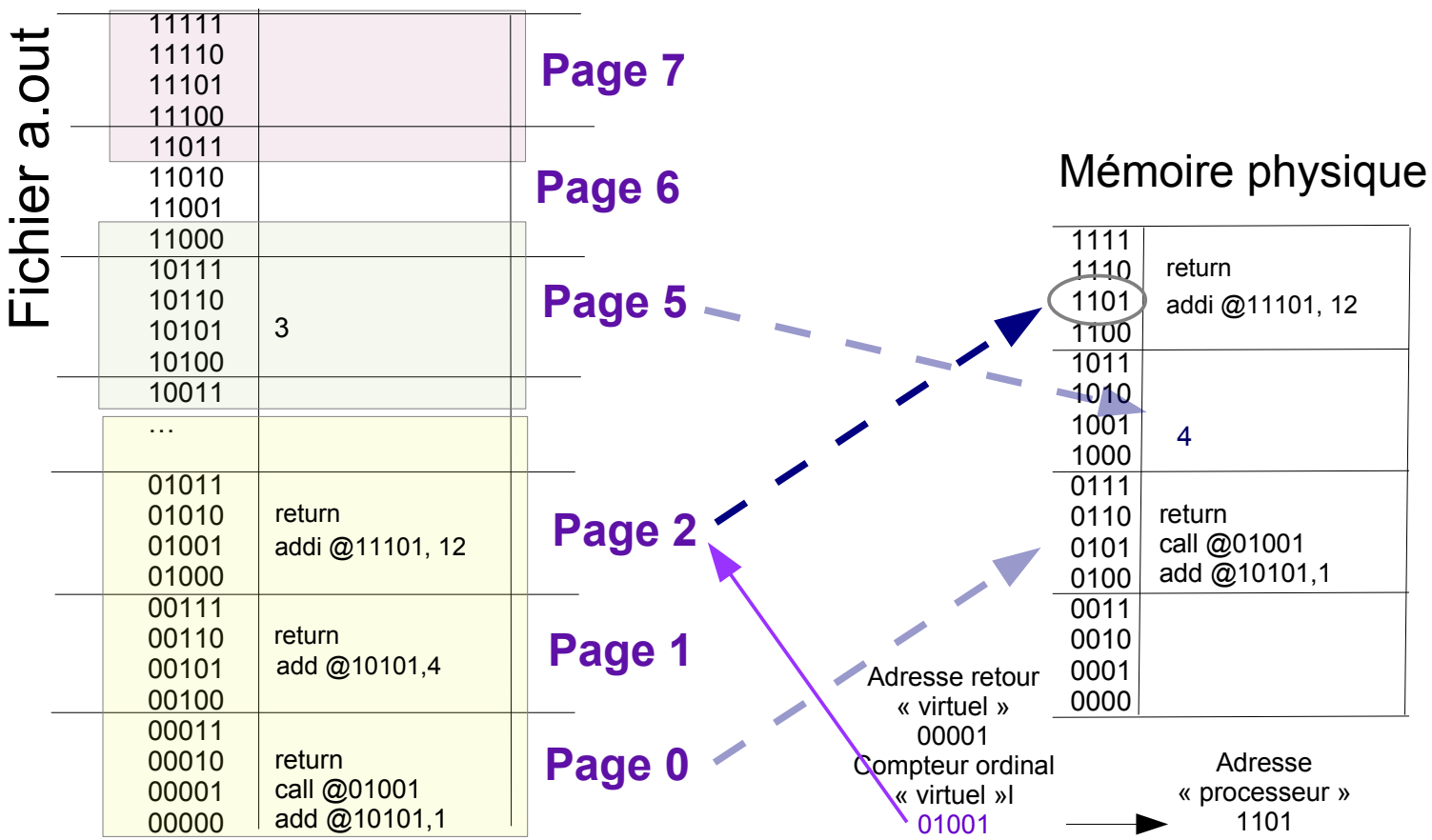
- Exécution de l'instruction : mémorisation de l'adresse de retour et mise à jour du CO



Pagination

Déroulement de l'exemple

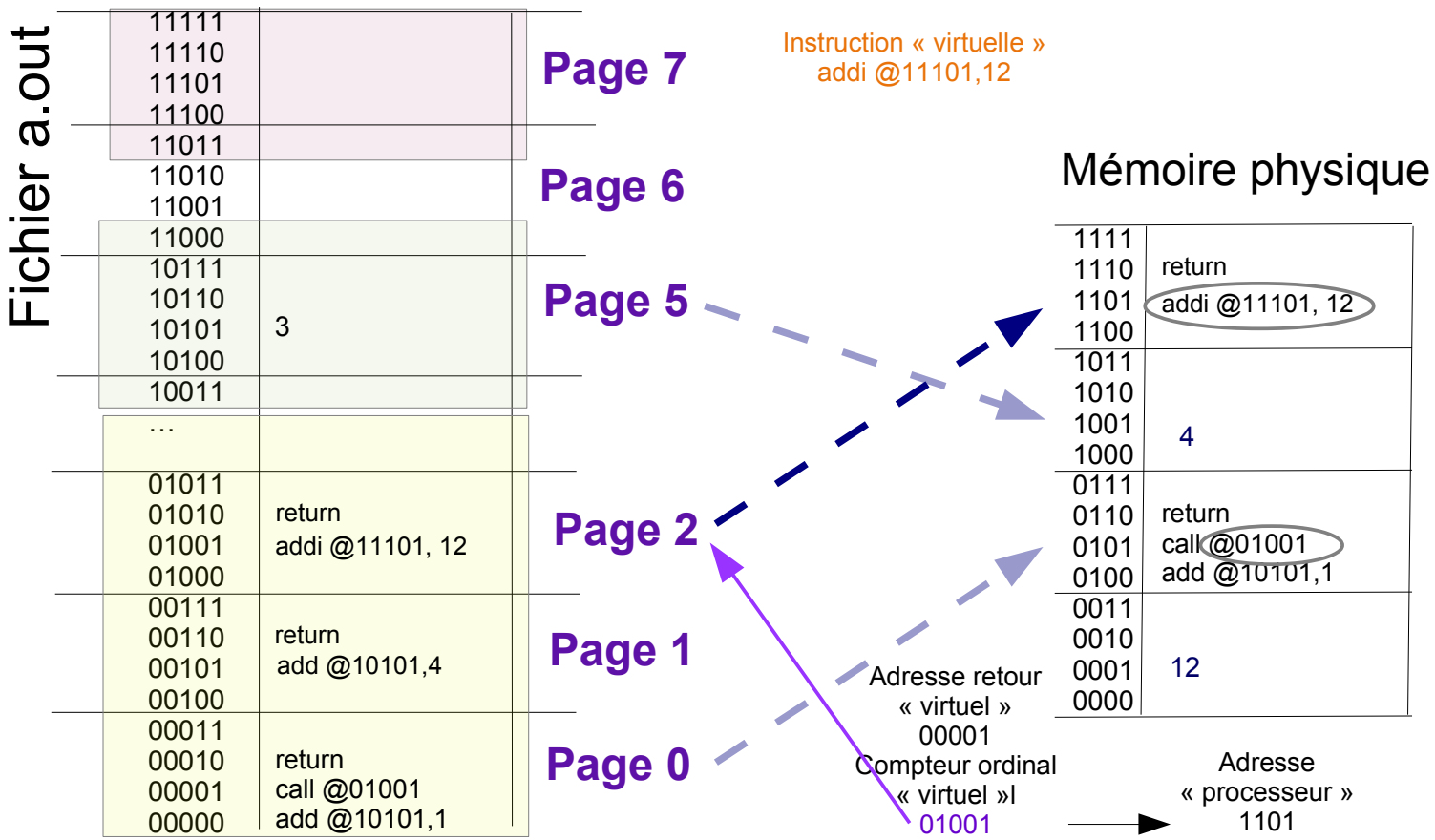
- Calcul de la page contenant l'instruction #01001 → **Défaut de page**
  → Chargement de la page #2



Pagination

Déroulement de l'exemple

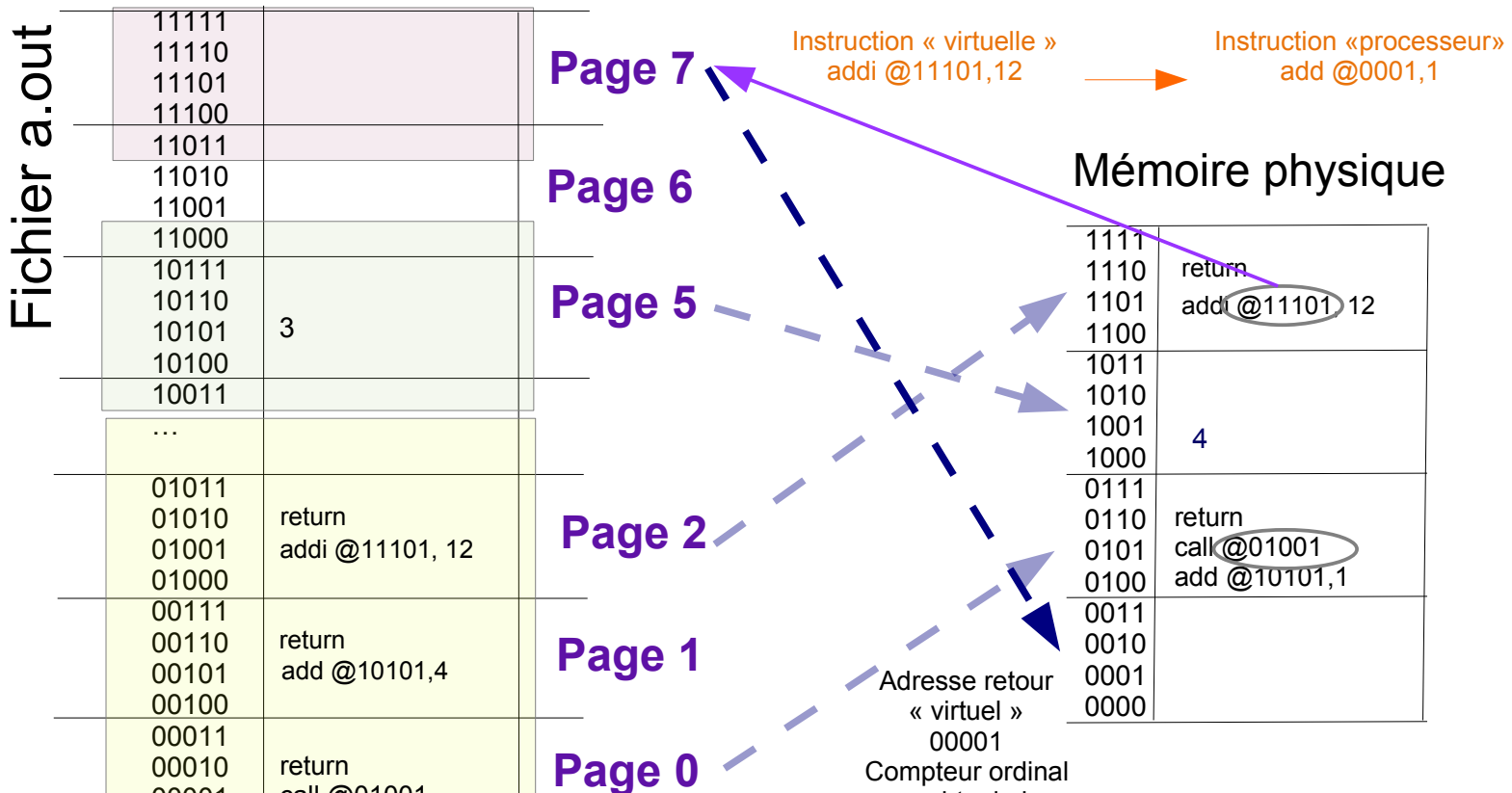
- L'instruction à exécuter est `addi @11101,12`



Pagination

Déroulement de l'exemple

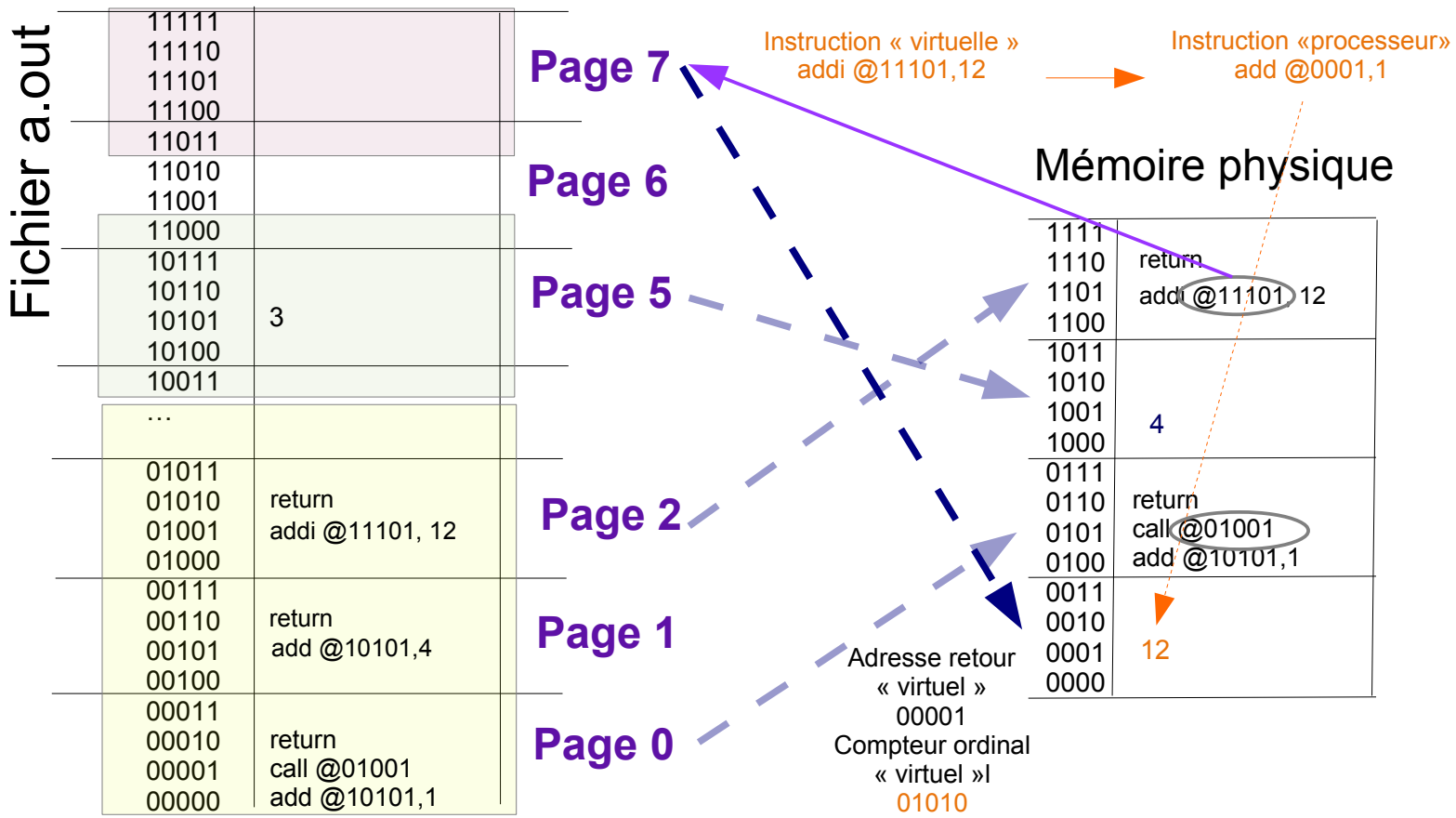
- Pour exécuter l'instruction, il faut charger la page correspondante à l'adresse `@11101` → **Défaut de page**
  → Chargement de la page #7
- La "vraie" instruction est `add @1001,1`



Pagination

Déroulement de l'exemple

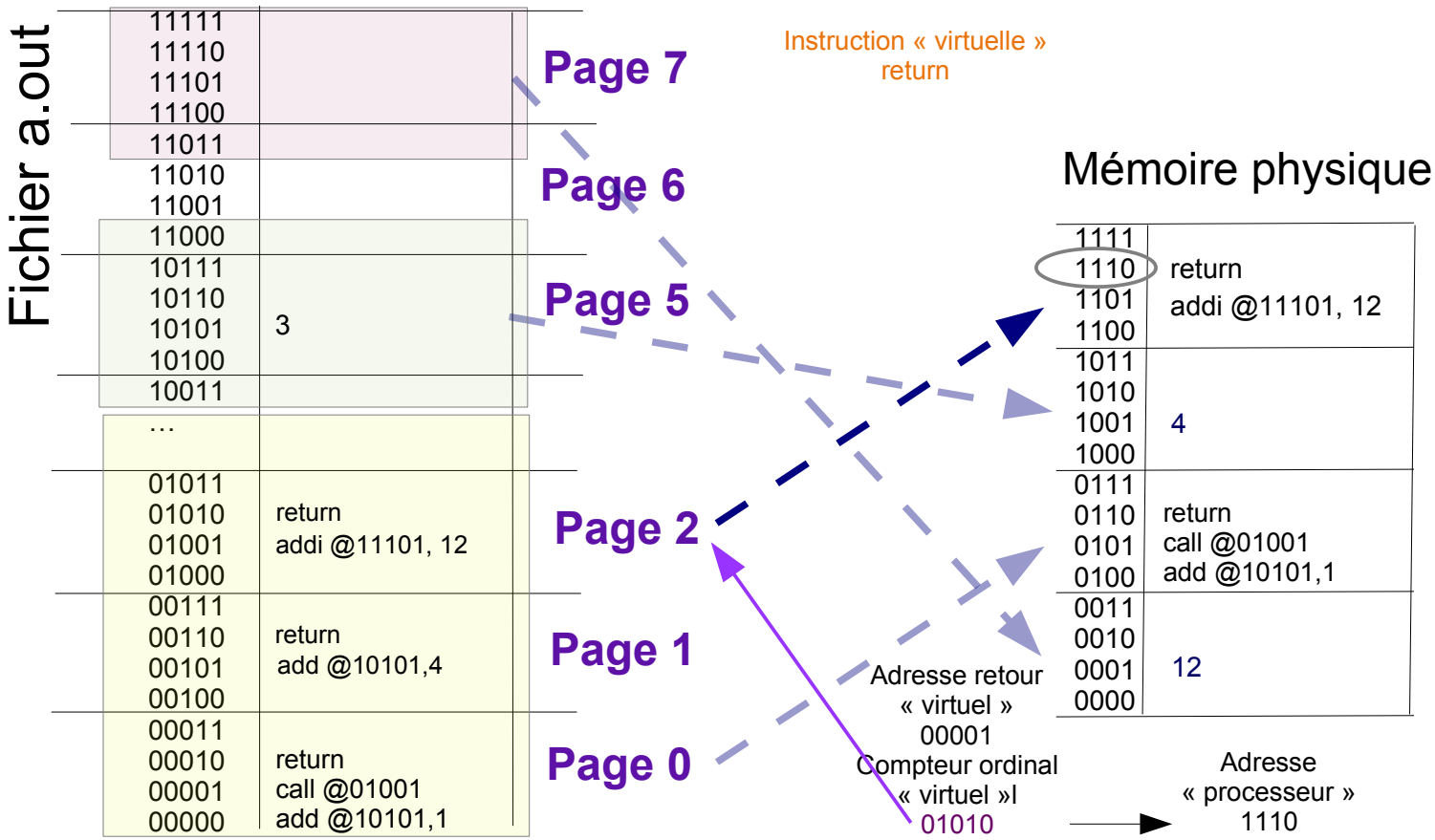
- Exécution de la vraie instruction
- Mise à jour de CO



Pagination

Déroulement de l'exemple

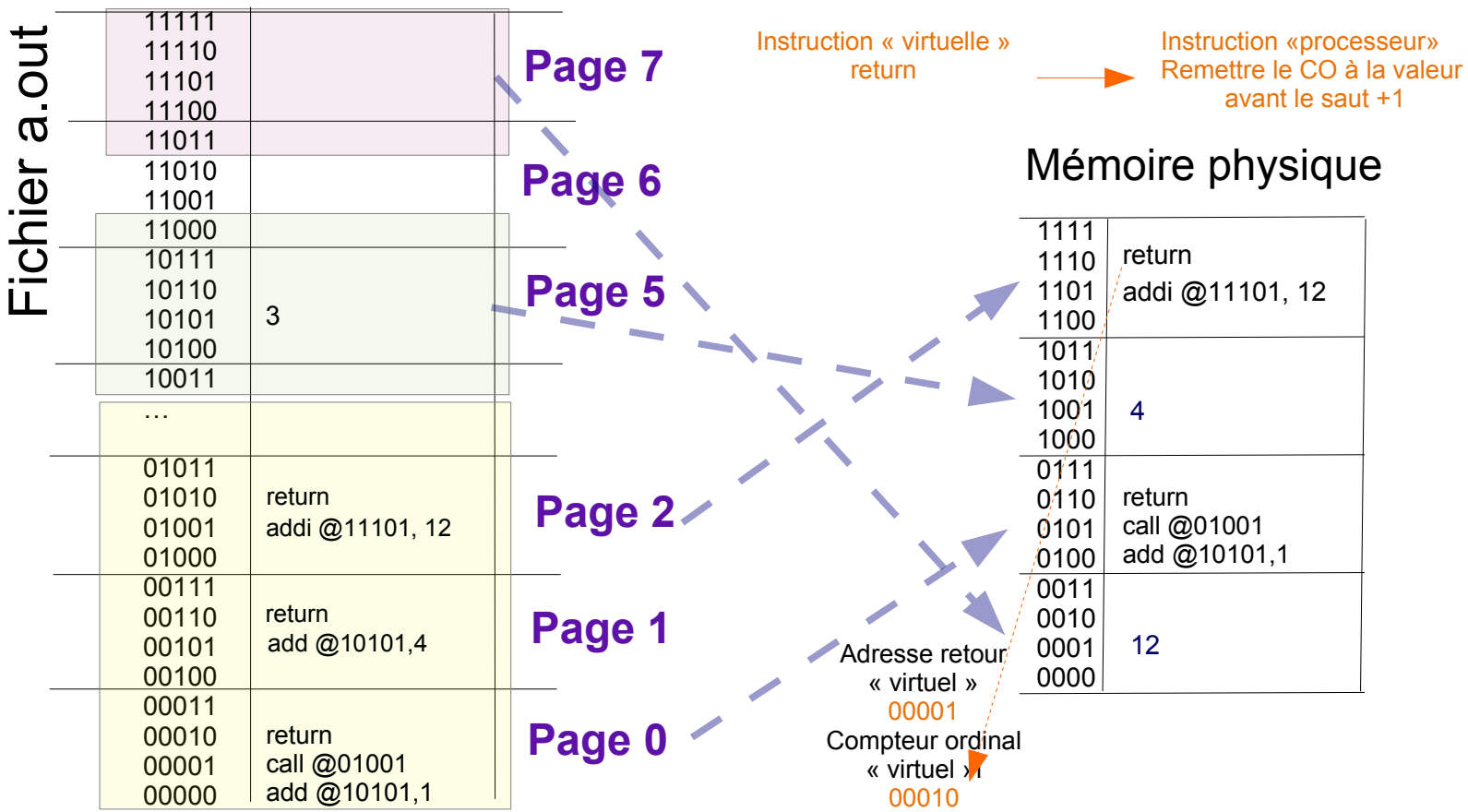
- Calcul de la page contenant l'instruction #01010 → Déjà chargée



Pagination

Déroulement de l'exemple

- L'instruction à exécuter est `return`



Pagination

MMU : Memory Managment Unit

- Chaque processus manipule uniquement des adresses *virtuelles* (ou logiques)
- L'unité de **gestion de mémoire** (MMU) intégrée au processeur :
  - Traduit à la volée les adresses virtuelles en adresses physiques
  - Gére les pages : charge/supprime les pages dans les cadres, ...

Table des pages

- Les pages sont chargées à la volée : le cadre qui est affecté à une page peut être "quelconque" et être changé en cours d'exécution

$\Rightarrow$ Comment est mémorisée la relation entre les pages et les cadres ?

$\Rightarrow$ Utilisation d'une **table des pages** (par processus) qui fait permet de la correspondance

Table des pages

Table des pages

- A chaque page P_i correspond **l'entrée** i dans la table qui donne l'adresse du cadre : exemple, la case 2, correspondant à la page P_2 contient 11 $\rightarrow$ la page P_2 est chargée dans le cadre 3
- La table des pages est nécessairement dans la MMU

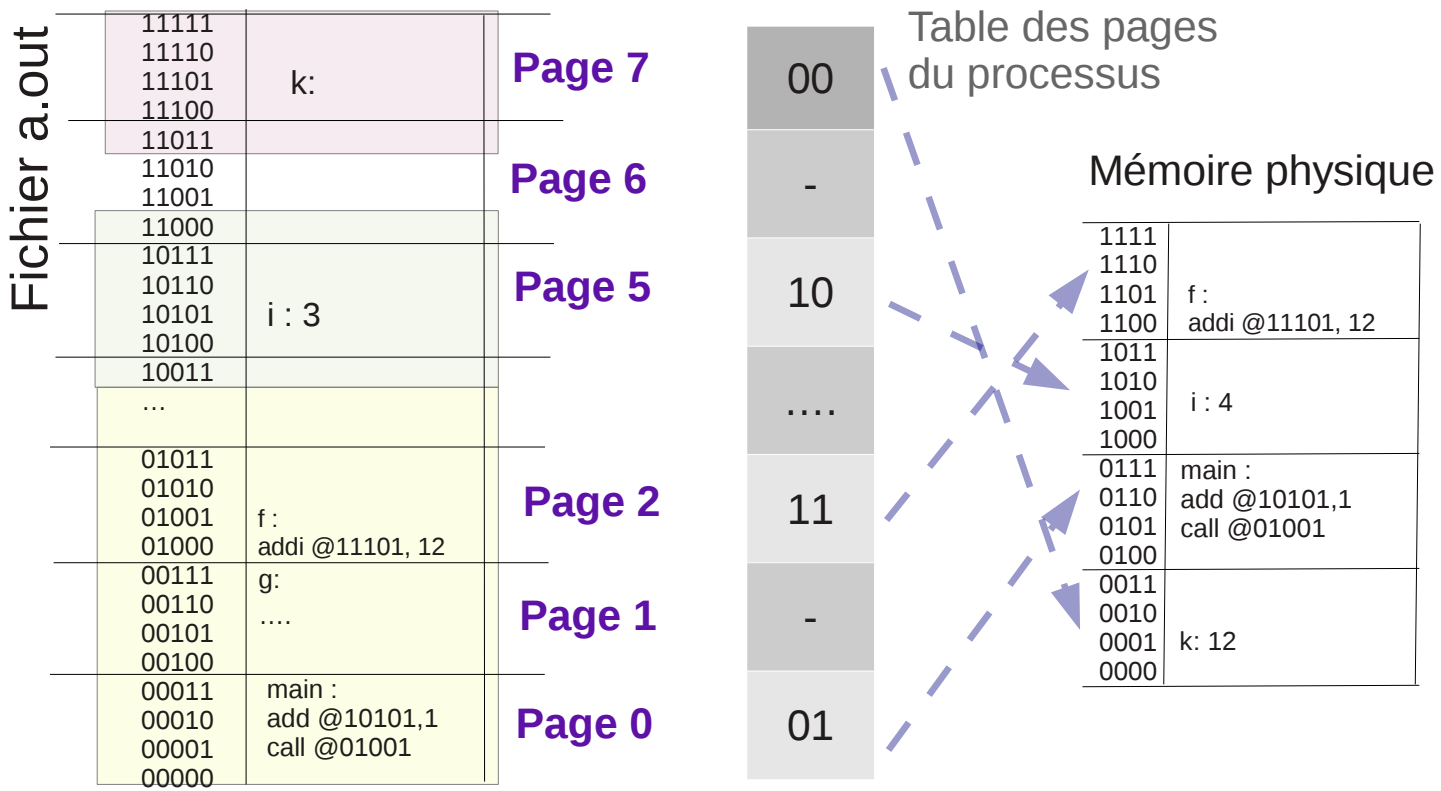
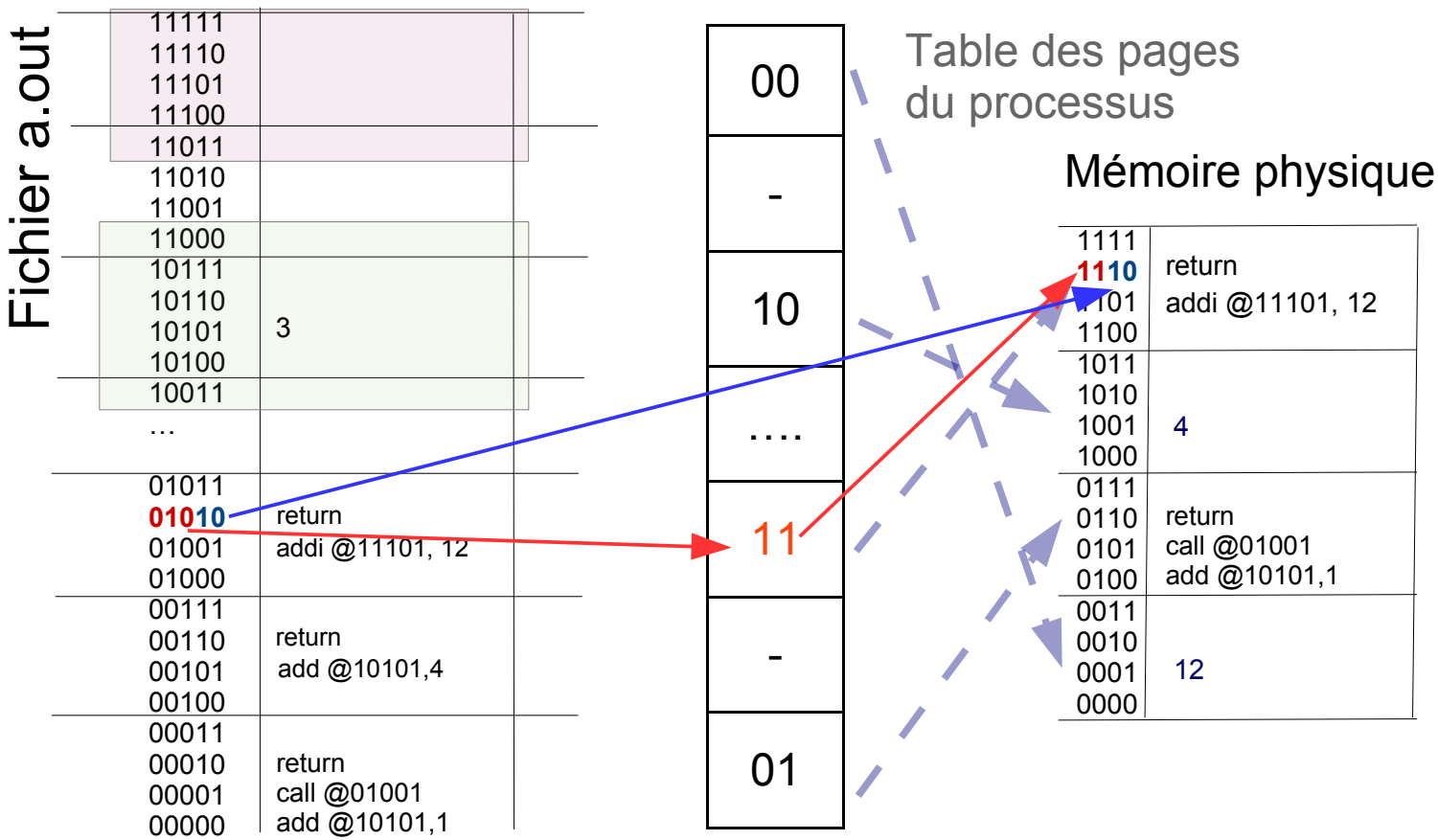


Table des pages

Table des pages

- La traduction d'adresse consiste à décoder chaque accès mémoire : @adresse → < #**page** , dépl. > → <#**cadre** , déplacement >



Adresse logique ↔ adresse physique

Traduction : processus complet

- Le **processeur** veut exécuter l'instruction `add` $@_v$`0100011,42`

→ Lors du décodage de l'instruction, il doit traduire l'adresse

→ Pour cela la MMU transforme l'adresse virtuelle $@_v0100011$:

- Elle décompose l'adresse en deux parties :

$$\underbrace{0\ 1\ 0}_{\text{No de page}}\quad\underbrace{0\ 0\ 1\ 1}_{\text{Déplacement dans la page}}$$

- Elle regarde dans la table des pages à l'entrée correspondant à la page 010 → elle trouve le n° du cadre correspondant (par ex. 11)
- L'adresse physique associée à l'adresse virtuelle est donc :

$$\underbrace{1\ 1}_{\text{No de cadre}}\quad\underbrace{0\ 0\ 1\ 1}_{\text{Déplacement dans le cadre}}$$

- La MMU remplace l'adresse $@_v0100011$ par l'adresse phys. $@_p110011$

⇒ Le processeur exécute en fait l'instruction `add` $@_p$`110011,42`

Plan

Gestion des pages

Défaut de page

- Lors de la traduction d'une adresse, la MMU teste si la page est déjà chargée en mémoire
  - Si oui, l'adresse physique est calculée.
  - Si non : **défaut de page**
    $\rightarrow$, déroutement vers le système d'exploitation :
    - La MMU émet un interruption : déroutement vers le système d'exploitation
    - Le SE détermine la page à charger en fonction de l'adresse virtuelle ayant causé le défaut de page
    - Recherche d'un cadre de page disponible, ou à défaut remplacement d'un cadre existant.
    - Décrémentation du compteur ordinal du processus afin de réexécuter l'instruction qui a généré le défaut de page.

Gestion de la pagination par le SE

Remplacement d'un cadre

- Lorsque le SE a besoin de placer une page dans un cadre et qu'il n'y a plus de cadre libre : il va libérer un cadre
- Utilisation d'un algorithme de remplacement de page : donne le "meilleur" cadre à vider
- Il existe plusieurs algorithme (: FIFO, LRU...) qui permettent de choisir au mieux le cadre à libérer en fonction de l'utilisation ou non de ce cadre, du fait que les données dans le cadre ont été modifiées ou non ...

Processus paginé

Structure d'un processus

- Un processus en cours d'exécution est donc formé
  - du fichier compilé
  - d'un ensemble de pages chargées en mémoire physique
  - d'un ensemble de pages stockées en zone swap sur un disque
  - d'une table des pages
    - chargée dans la mémoire de la MMU si le processus est actif
    - sauvegardée en MC ou dans la zone swap sinon

  $\rightarrow$ elle sera chargée/déchargée lors de la commutation de processus

Linux et la pagination

Exemple : Linux

- Connaître la taille d'une page : `getconf PAGESIZE`
- Résultat de `vm_stat` sur un Mac :

```
Mach Virtual Memory Statistics:
Page size of 4096 bytes
Pages free:                                4157.
Pages active:                           1521068.
Pages inactive:                         1182976.
...
"Translation faults":                16442337642.
...
Swapins:                                1422626.
Swapouts:                               1643960.
```

Linux et la pagination

Exemple : Linux

- Résultat de `pagestuff -a TextEdit` sur un Mac :

```
...
File Page 0 contains contents of section (__TEXT, __text)
File Page 1 contains contents of section (__TEXT, __text)
...
File Page 18 contains contents of section(__TEXT, __text)
File Page 19 contains contents of section(__TEXT, __text)
File Page 19 contains ... of section (__TEXT, __const)
File Page 19 contains ... of section (__TEXT, __cstring)
...
File Page 27  contains ... of section (__DATA_CONST,
    __const)
File Page 27 contains ... of section (__DATA_CONST,
    __cfstring)
....
File Page 36 contains symbol table for non-global
    symbols
File Page 36 contains symbol table for defined global
    symbols
...
```

Allocation de pages

Pages partagées

- Exemple sous UNIX : après un `fork()`, le père et le fils ont leur propre table des pages et partagent les mêmes données.
  $\rightarrow$ Les tables de pages du père et du fils pointent sur les mêmes cadres physiques, en accès lecture seule.

  Mais dès qu'un accès en écriture intervient sur un cadre partagé, le système fait une copie physique de la page en cause : le père et le fils auront une copie distincte de la page.

Mémoire cache et page

Mémoire cache et taille des pages

- Pour optimiser les temps d'accès sur entrées/sorties, on utilise très souvent des mémoires cache

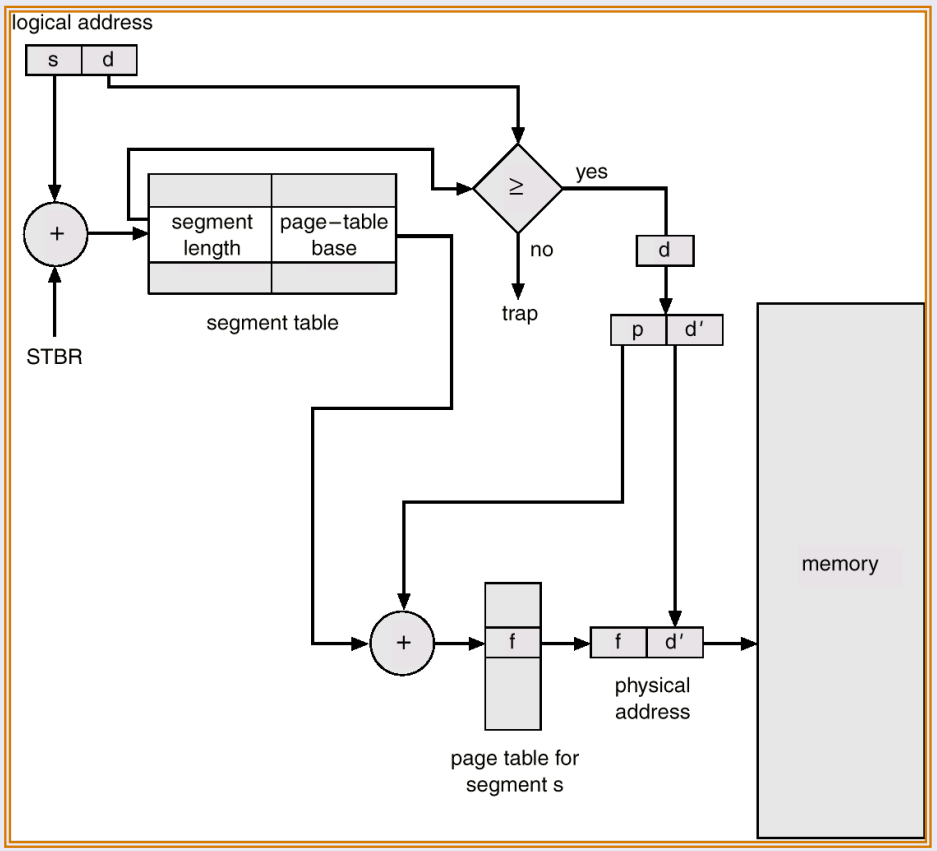
→ il est préférable d'avoir une taille de blocs cohérente avec la taille des pages : idéalement, un bloc contient exactement une page (ou un multiple)

Plan

La segmentation paginée

Segment et pagination

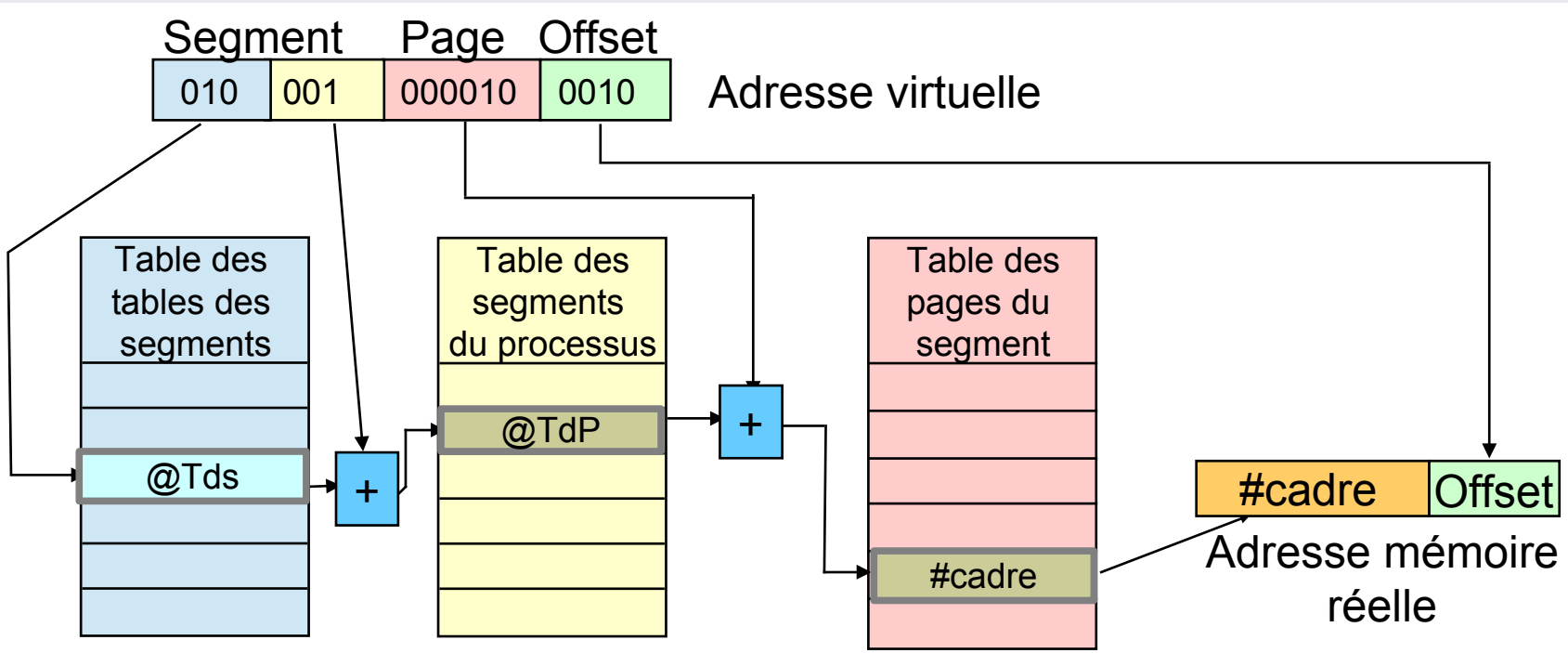
- Principe : on associe une table des pages à chaque segment.
- Chaque segment dispose d'un segment dédié à sa table des pages



La segmentation paginée

Segment et pagination Unix

- Les adresse sont décomposées en 4 parties



La segmentation paginée

Segment et pagination Unix

- Les adresse sont décomposées en 4 parties
- Schéma global