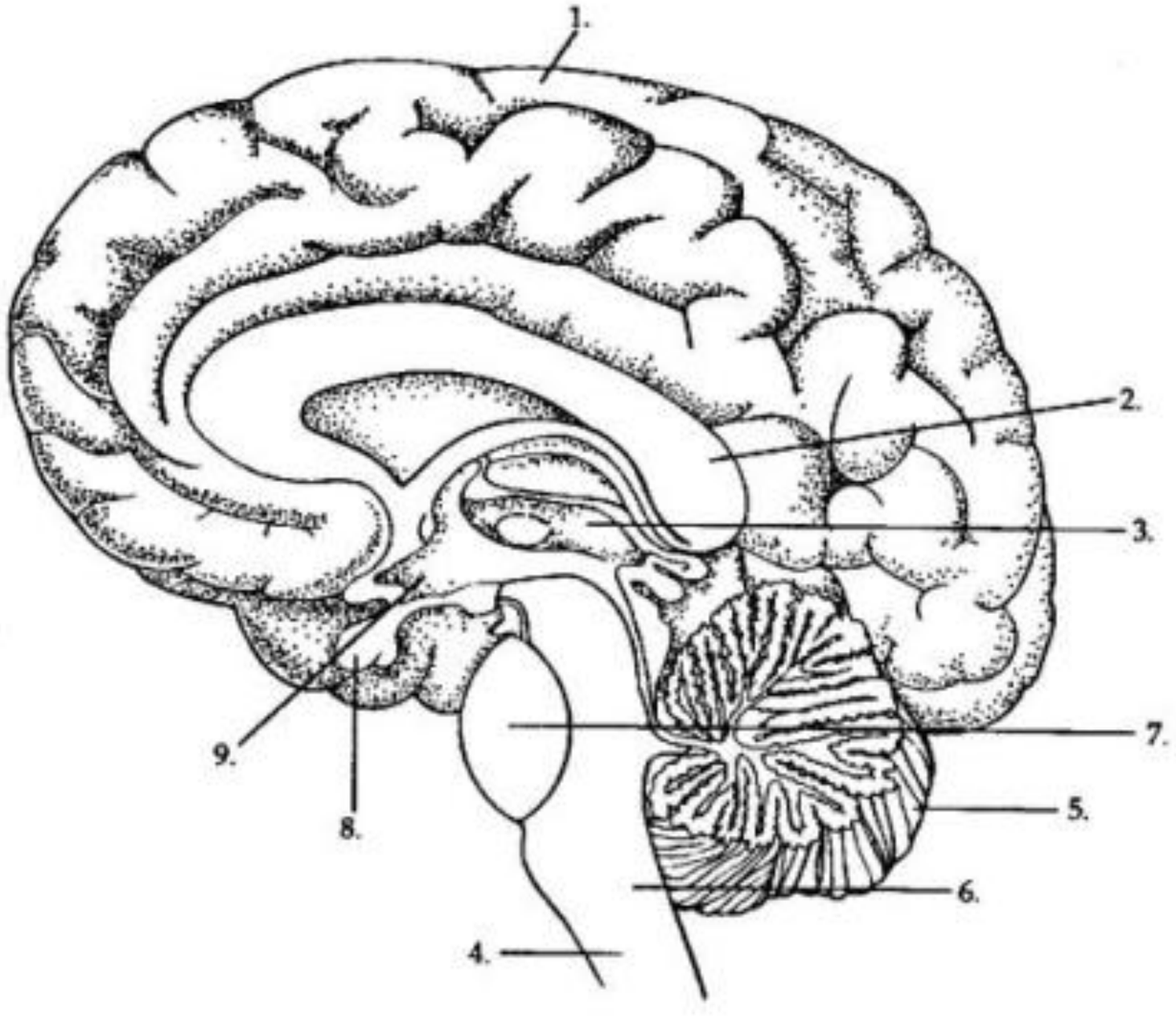
Gestion de la mémoire

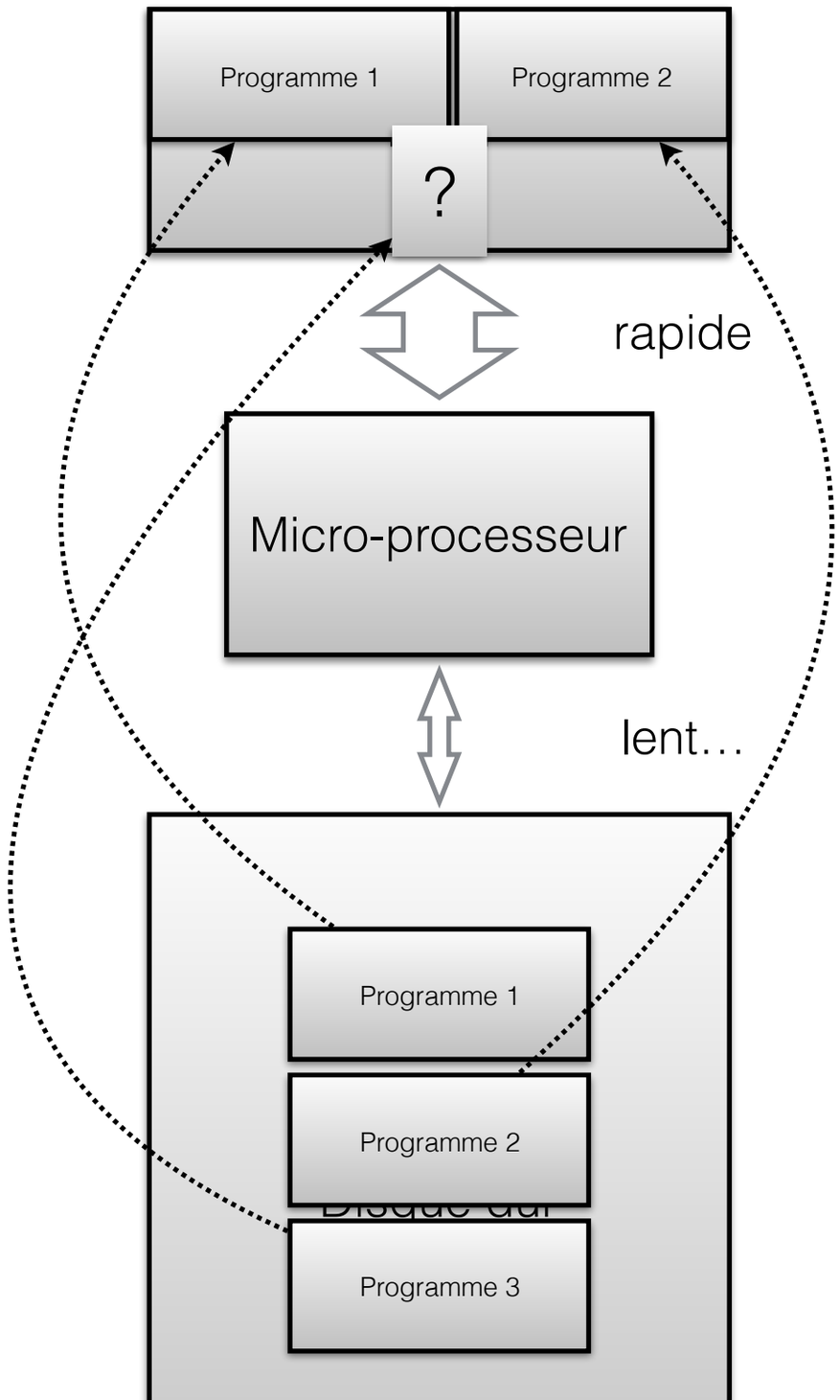


GIF-1001 Ordinateurs: Structure et Applications, Hiver 2019

Jean-François Lalonde

Gestion mémoire: objectif

- Un ordinateur possède *plusieurs* programmes
- Où ces programmes sont-ils situés?
  - Sur le disque dur
- Peut-on les exécuter s'ils sont sur le disque dur?
  - Non, le disque dur est un périphérique de stockage *lent*
- Donc, il nous faut les transférer dans la mémoire principale (RAM)
- L'objectif de la gestion mémoire est de *partager la mémoire RAM entre les divers programmes*



Gestion mémoire

- Objectif principal:
  - partager la mémoire RAM entre les divers programmes
- Objectifs secondaires:
  - Utilisation simple pour un programme
  - Maximiser l'utilisation de la mémoire disponible

Aujourd’hui

- Deux mécanismes d’allocation mémoire:
  - allocation mémoire contigüe
  - allocation mémoire paginée

Allocation mémoire contigüe

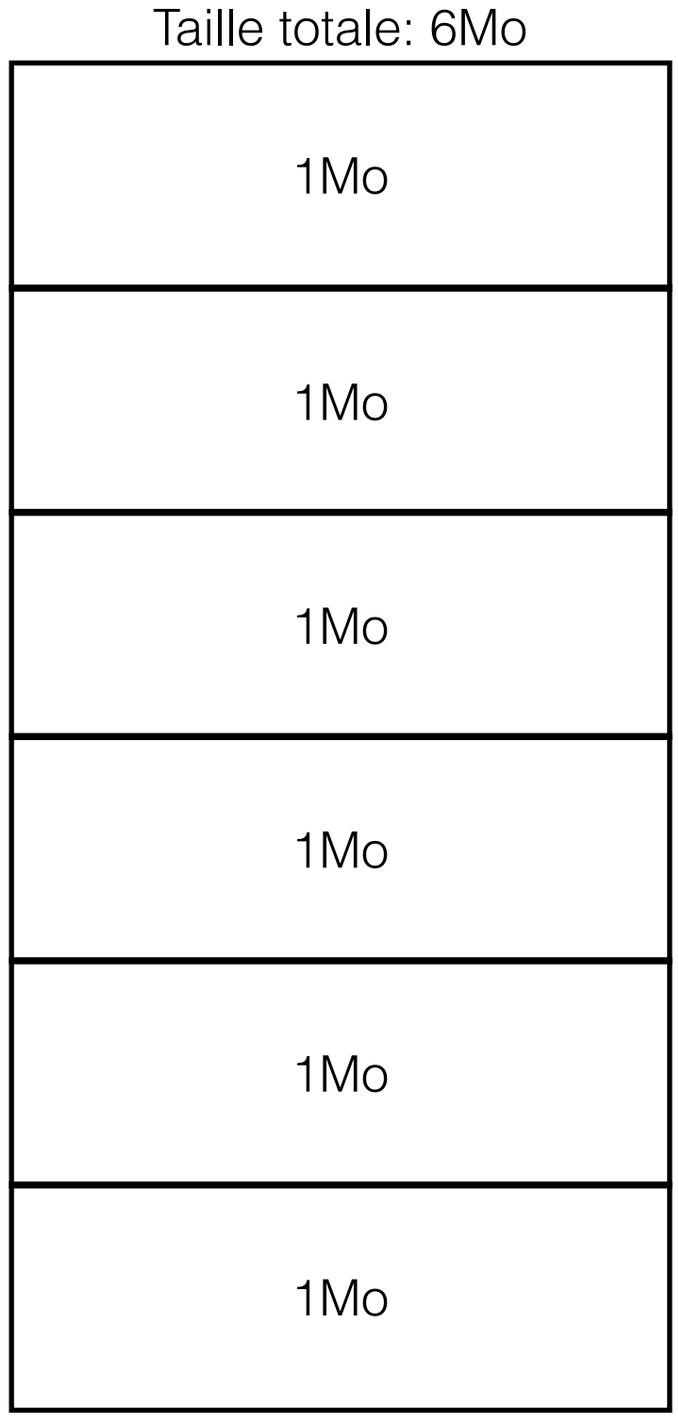
Allocation mémoire contigüe

- On réserve un «bloc» (ou «fragment») de mémoire contigu.
- Les blocs de mémoire peuvent avoir une taille:
  - **fixe**: la taille des blocs est prédéterminé, et est la même pour tous les processus.
  - **variable**: chaque processus se voit allouer un bloc de taille différente.

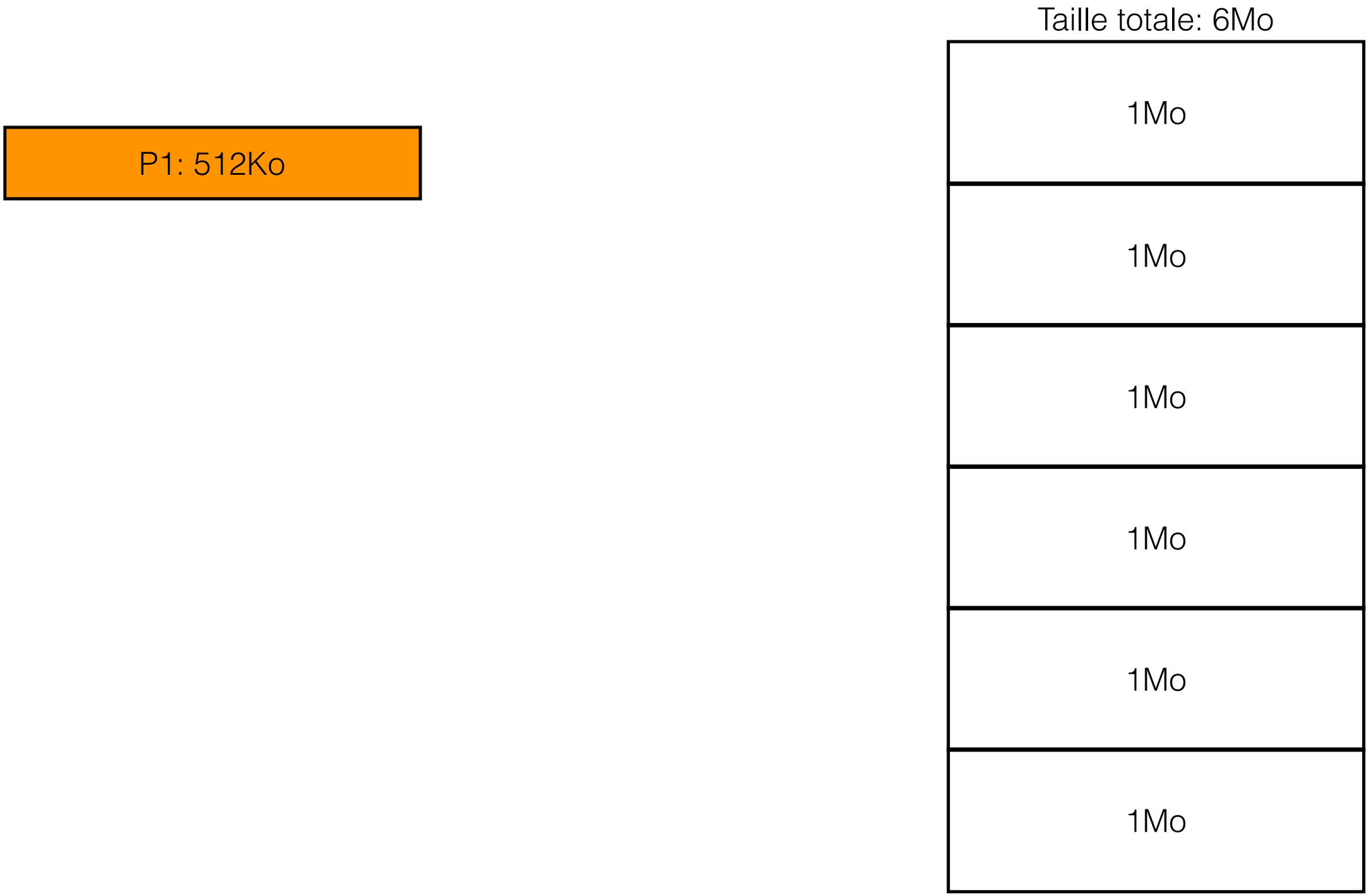
Allocation mémoire contigüe, taille **fixe**

- On divise la mémoire en blocs de taille **fixe**, par exemple 1Mo

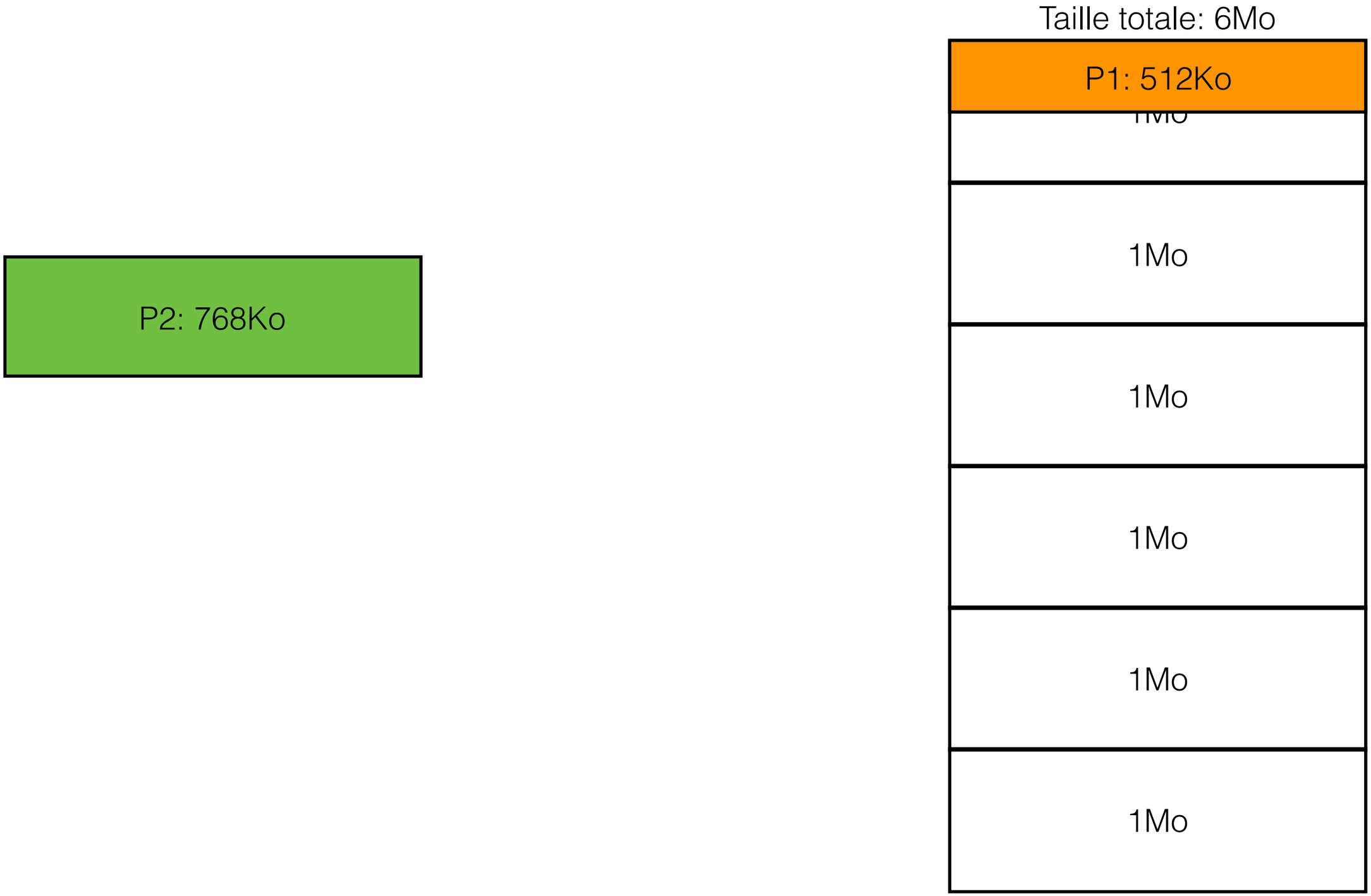
- Chaque processus est placé dans un bloc libre



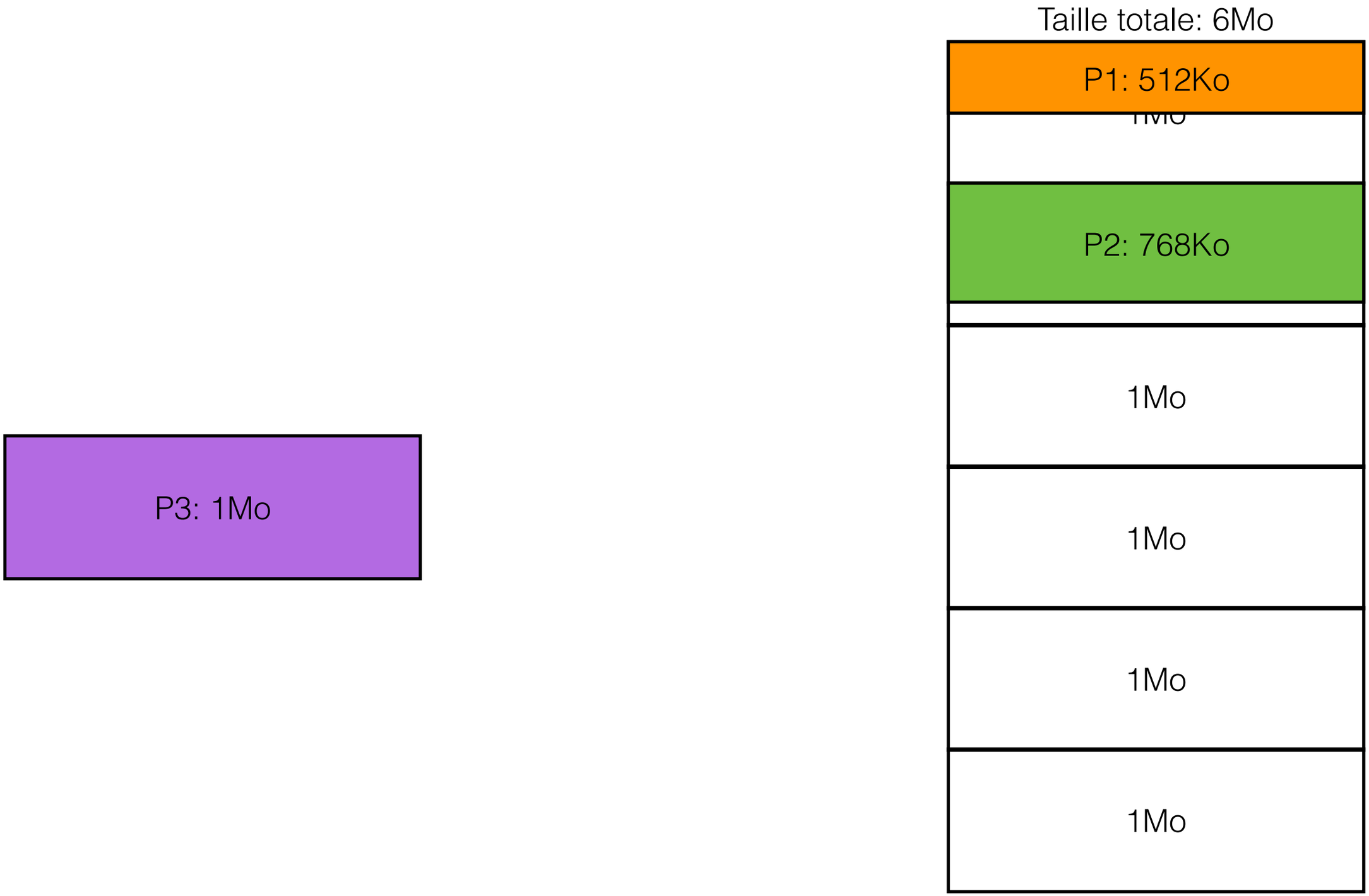
Allocation mémoire contigüe, taille **fixe**



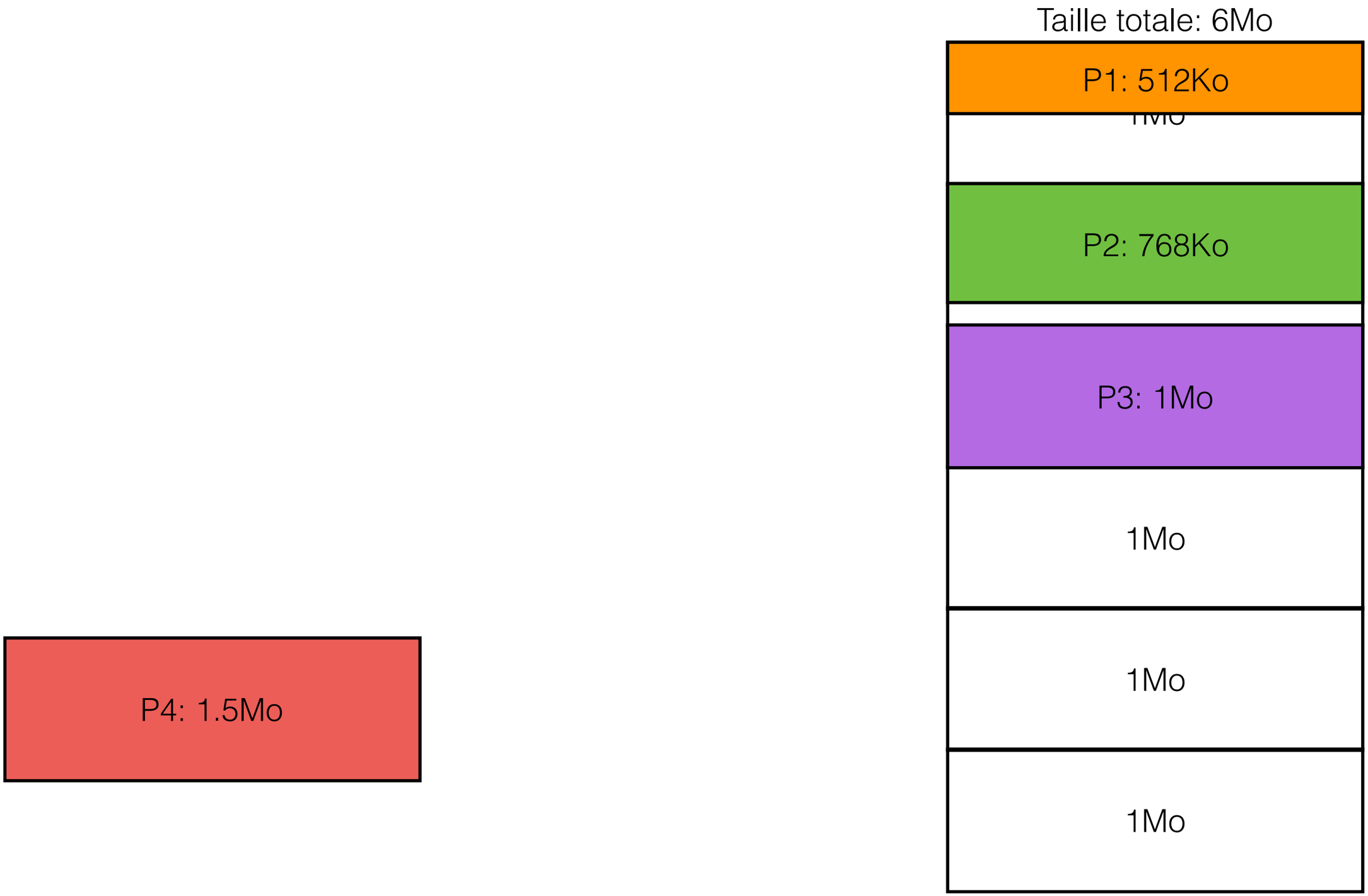
Allocation mémoire contigüe, taille **fixe**



Allocation mémoire contigüe, taille **fixe**



Allocation mémoire contigüe, taille **fixe**

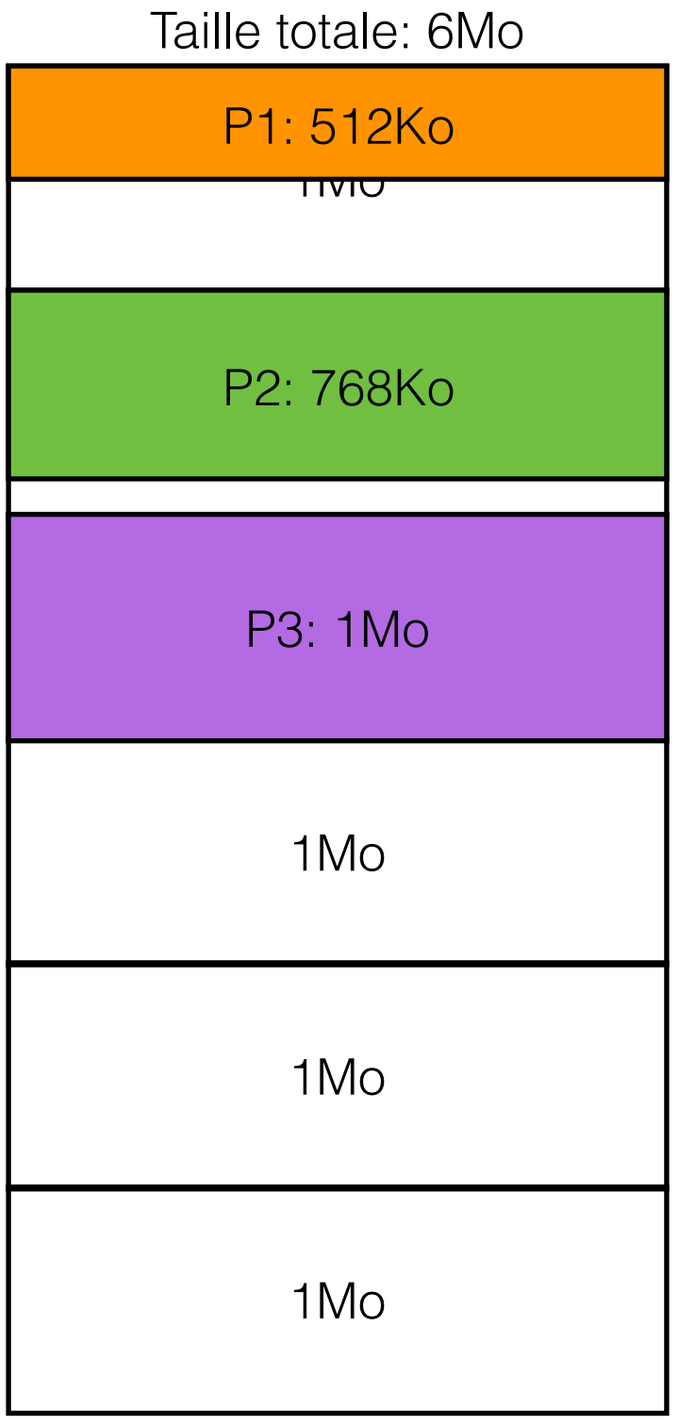


Allocation mémoire contigüe, taille **fixe**

- Quels sont les problèmes avec l'allocation contigüe avec partitions de taille fixe?
  - Que faire si le processus nécessite plus de mémoire qu'un bloc?
  - Beaucoup d'espace mémoire perdu: **fragmentation**!

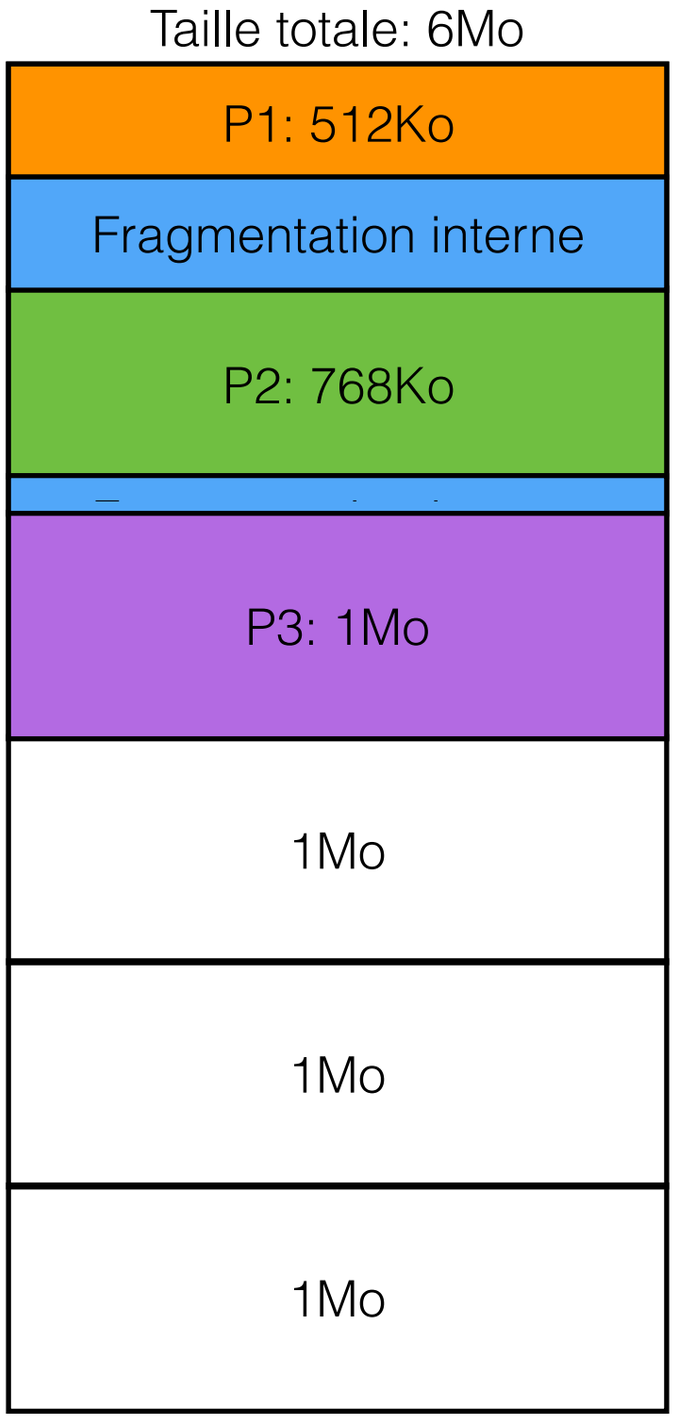
P4: 1.5Mo

Ne peut être admis en mémoire!



Fragmentation

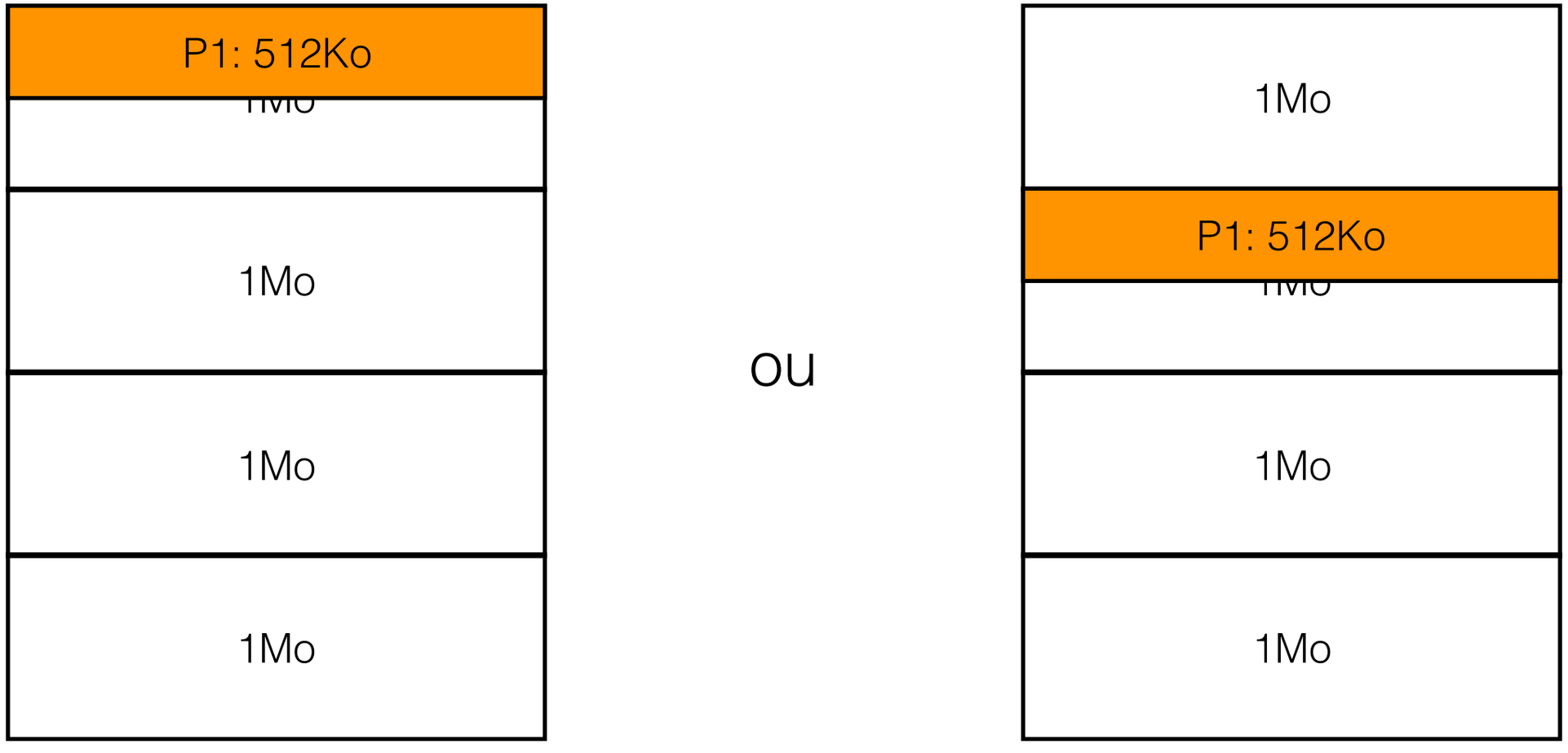
- Il en existe 2 types:
  - Fragmentation **interne**: espace perdu **à l'intérieur** une partition
  - Fragmentation **externe**: espace perdu **à l'extérieur** d'une partition
- Avec des partitions de taille **fixe**, seule la fragmentation **interne** est possible
  - aucun espace à l'extérieur d'une partition n'est perdu—il peut toujours être alloué à un autre processus peu importe son emplacement



Adresses **virtuelles** et adresses **physiques**

- Est-ce qu'un processus doit absolument savoir à quel endroit il est placé en mémoire?

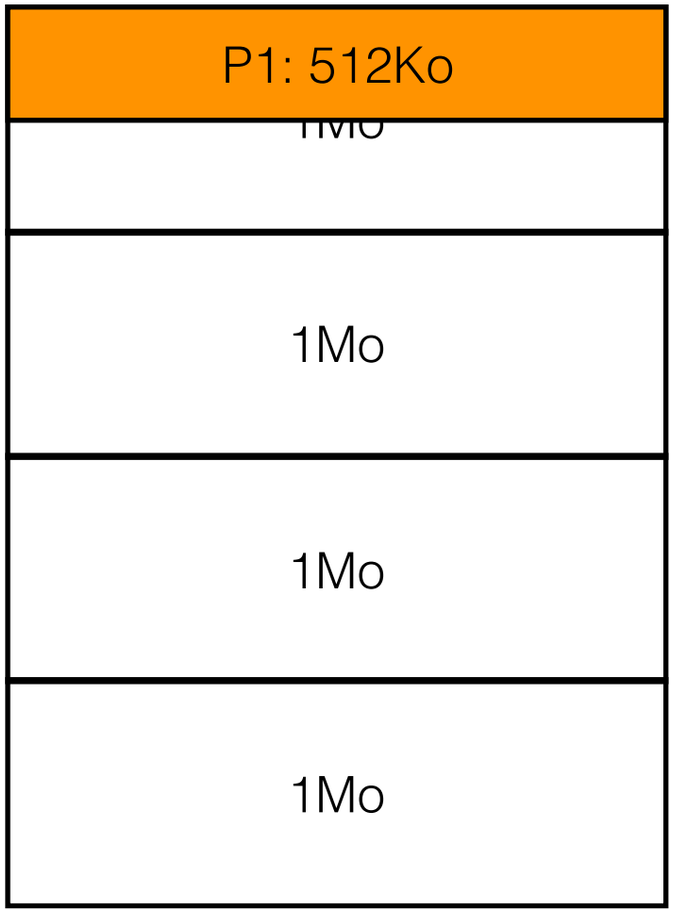
- Pour P1, est-ce que d'être placé dans le premier ou le second bloc fait une différence?



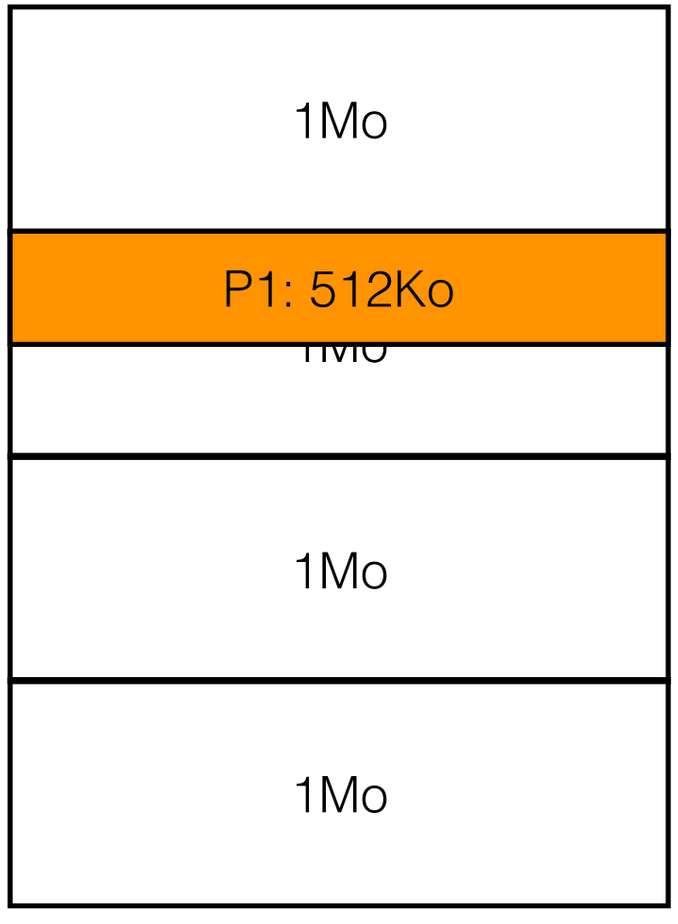
Adresses **virtuelles** et adresses **physiques**

- Aucune différence!

- Tout ce que le processus doit savoir, c'est comment accéder à chacune des adresses dans son bloc de mémoire.



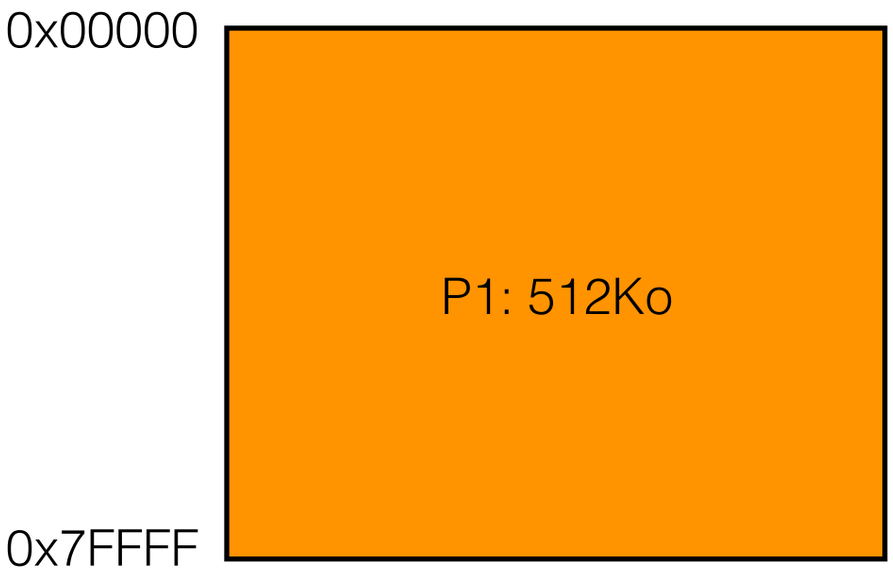
ou



Adresses **virtuelles** et adresses **physiques**

Adresses **virtuelles**

du point de vue du **processus**



Adresses **physiques**

du point de vue de la **mémoire**

Adresses **virtuelles** et adresses **physiques**

- Un processus utilise des adresses **virtuelles**
- La mémoire utilise des adresses **physiques**
- Il faut donc «traduire» entre les deux: c'est le travail du «**Memory Management Unit**» (MMU)

Allocation contigüe fixe: MMU

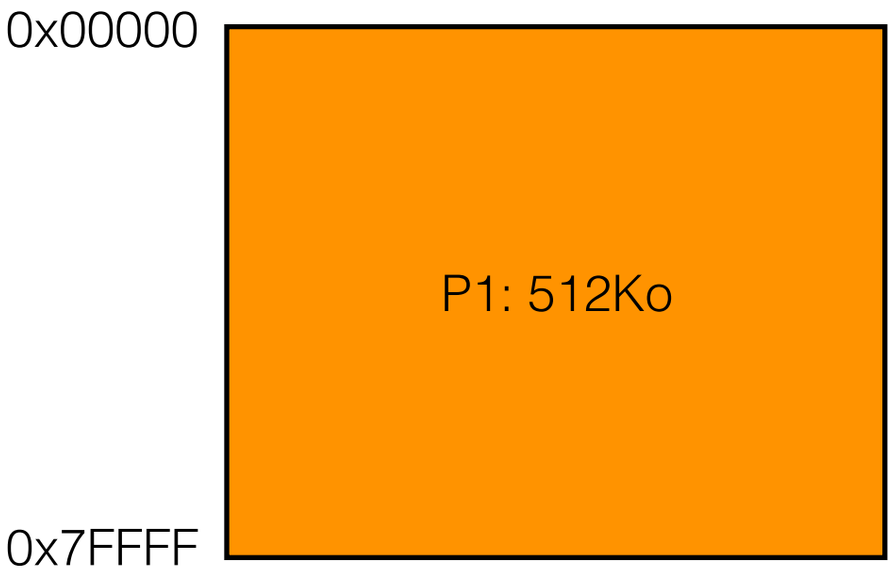
- En allocation contigüe avec partitions de taille fixe, le **MMU** effectue le calcul suivant pour traduire une adresse **virtuelle** en adresse **physique**:

$a_{partition}$ est la première adresse de la partition attribuée au processus

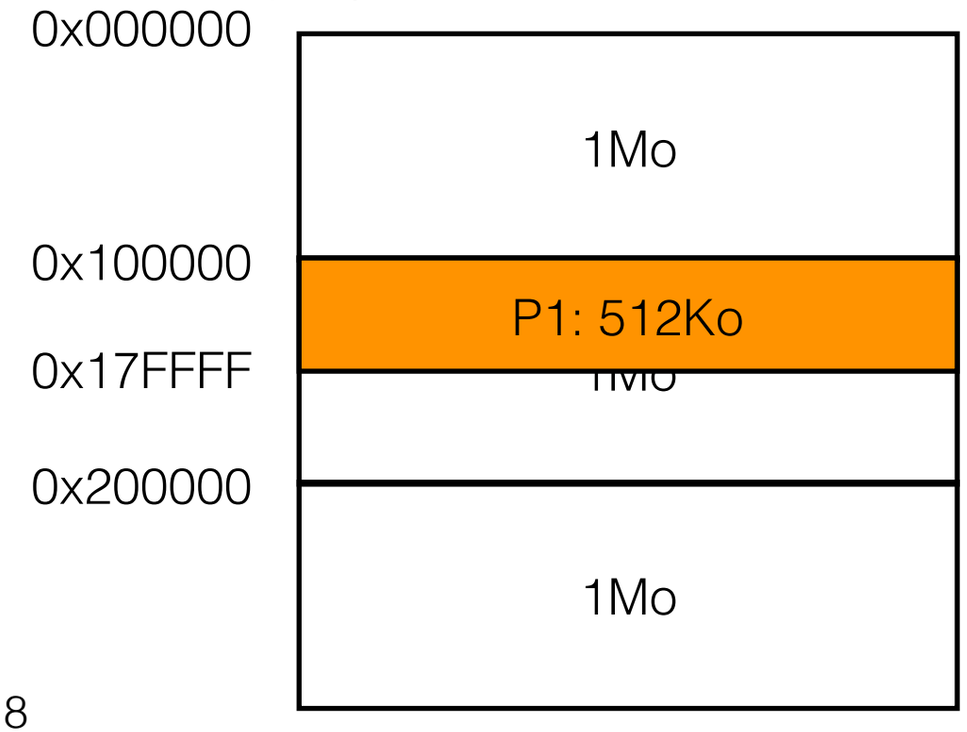
$$a_{physique} = a_{virtuelle} + a_{partition}$$

- Par exemple, l'adresse virtuelle 0x00AB3 correspond à l'adresse physique 0x100AB3 dans l'exemple ci-bas.

Adresses **virtuelles**
(du point de vue du **processus**)

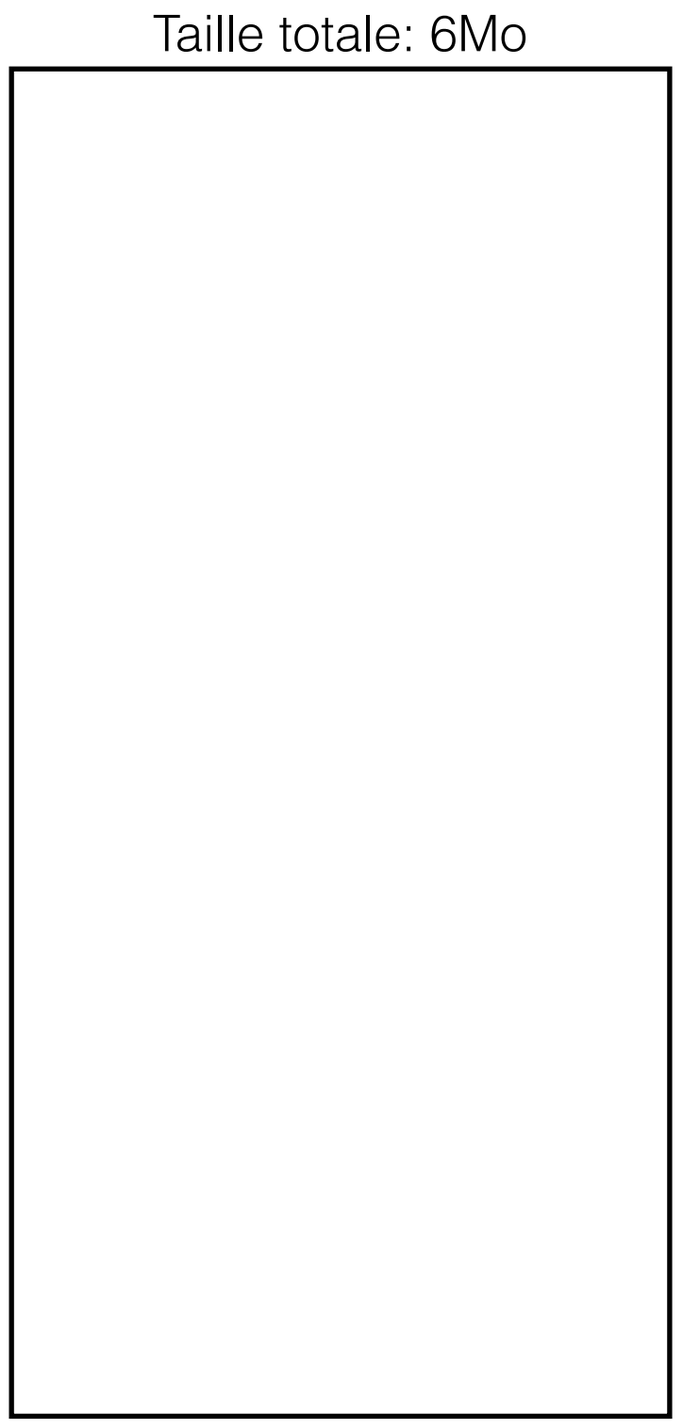


Adresses **physiques**
(du point de vue de la **mémoire**)

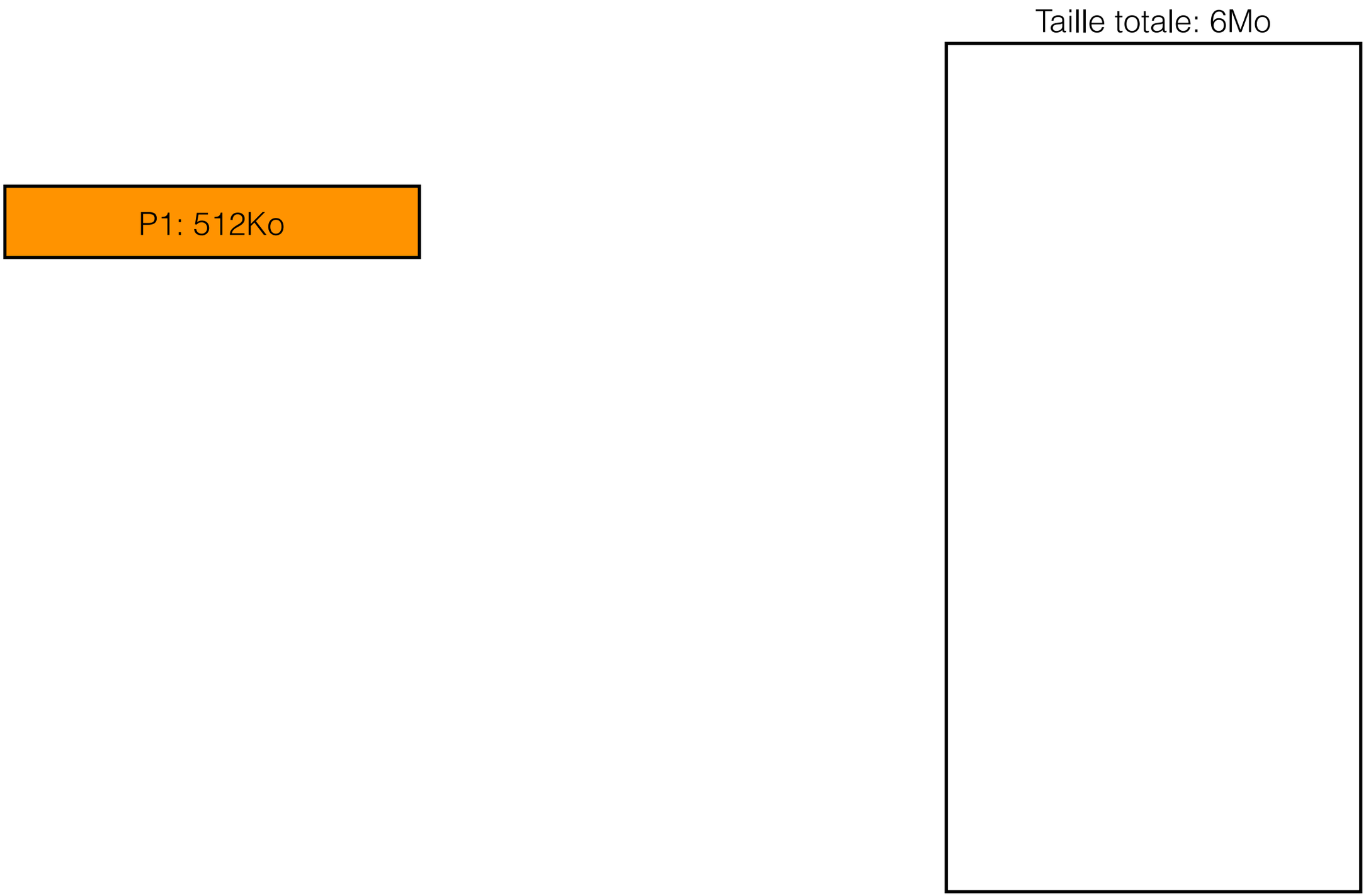


Allocation mémoire contigüe, taille **variable**

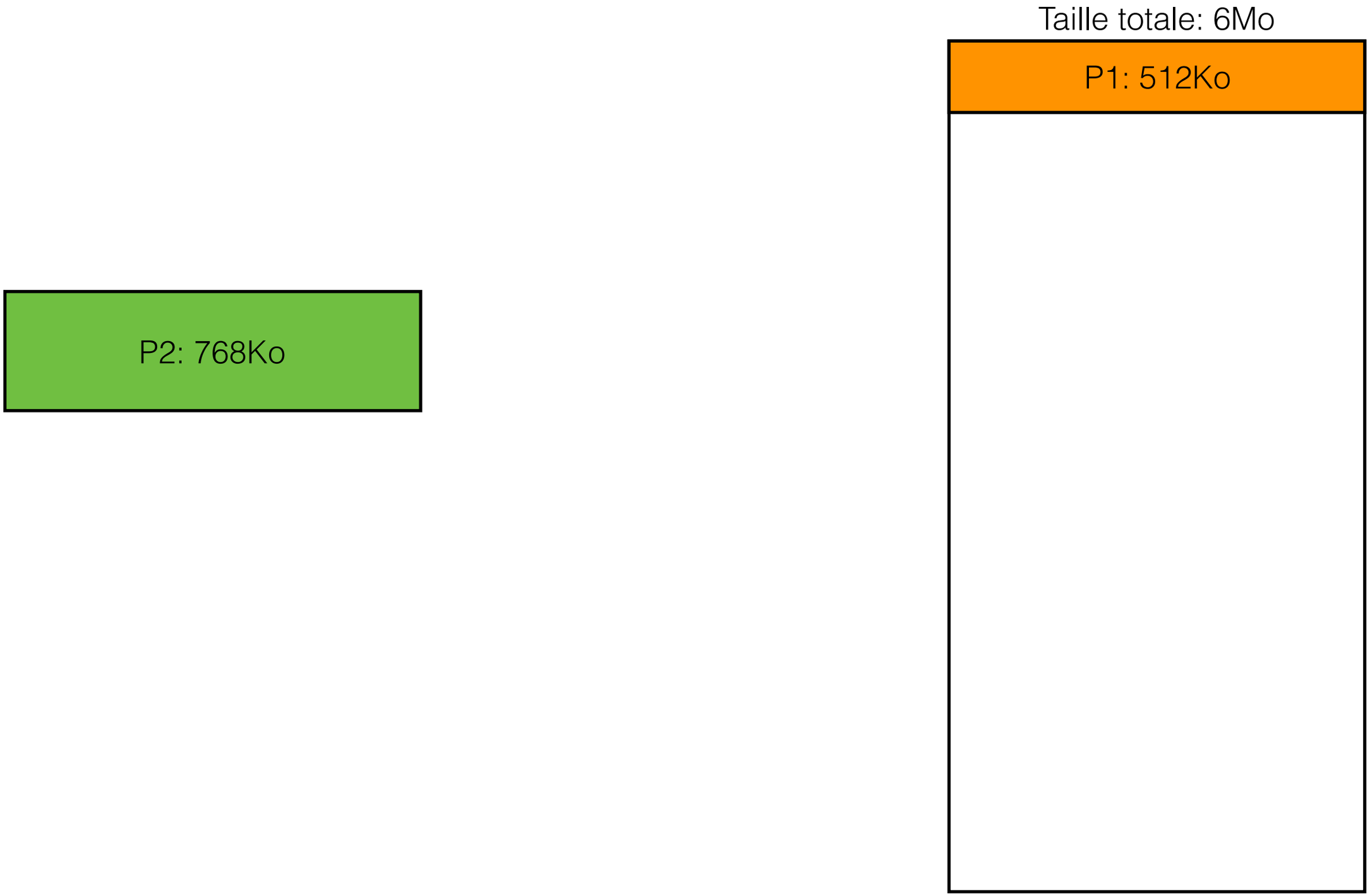
- On crée un bloc de la bonne taille pour chaque processus.
- Le bloc peut parfois être créé à différents endroits: il faudra déterminer la bonne stratégie à employer!



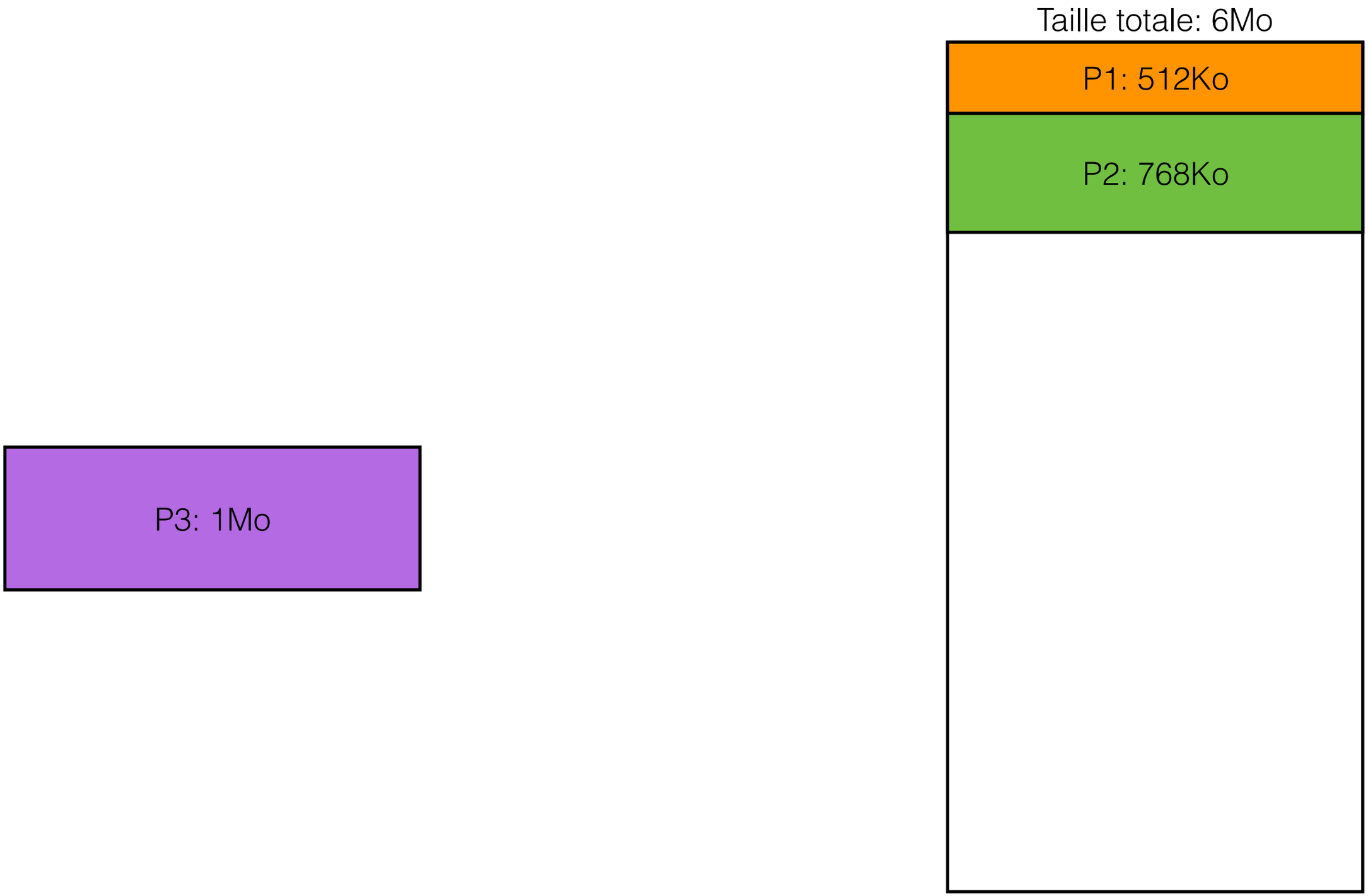
Allocation mémoire contigüe, taille **variable**



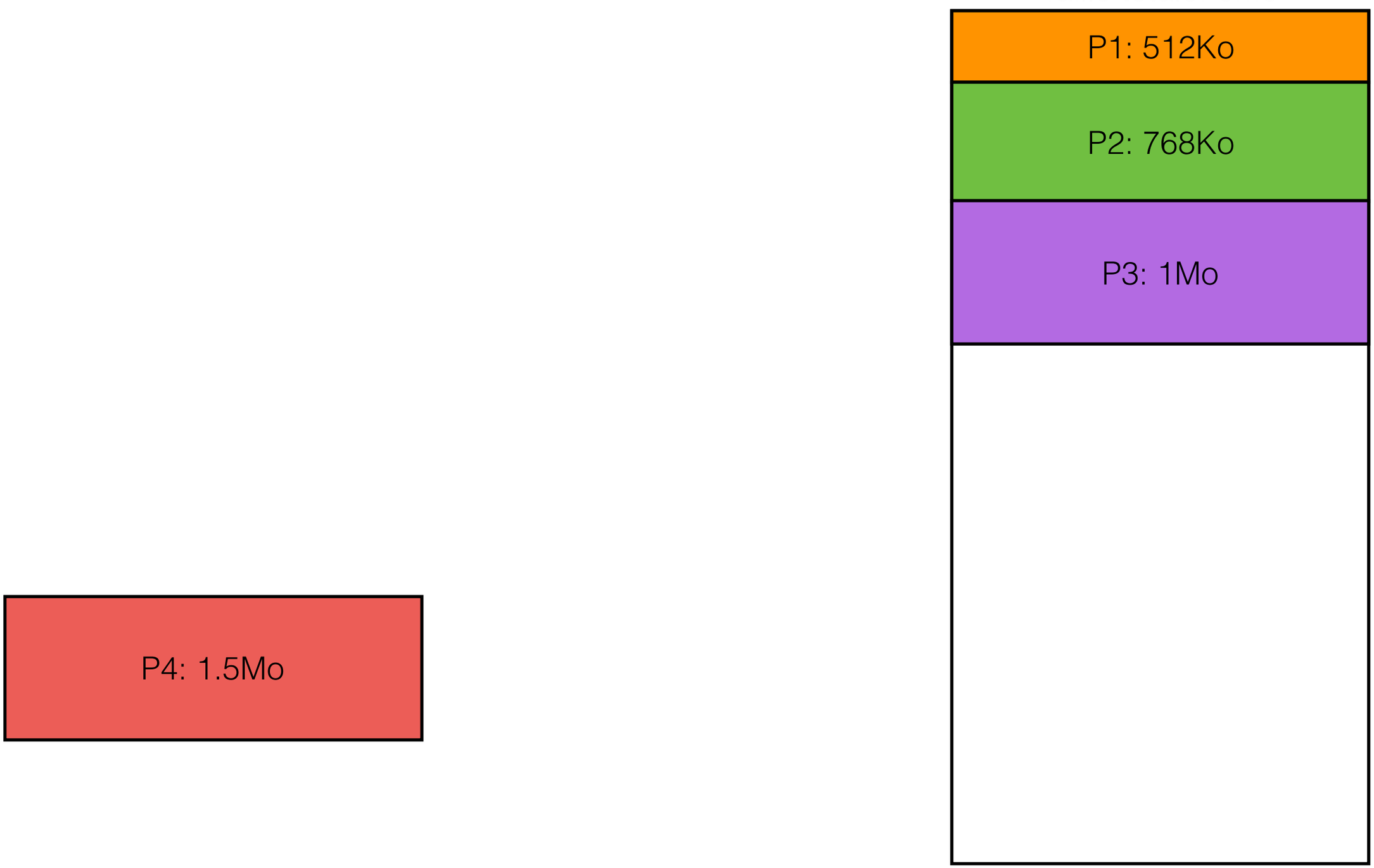
Allocation mémoire contigüe, taille **variable**



Allocation mémoire contigüe, taille **variable**

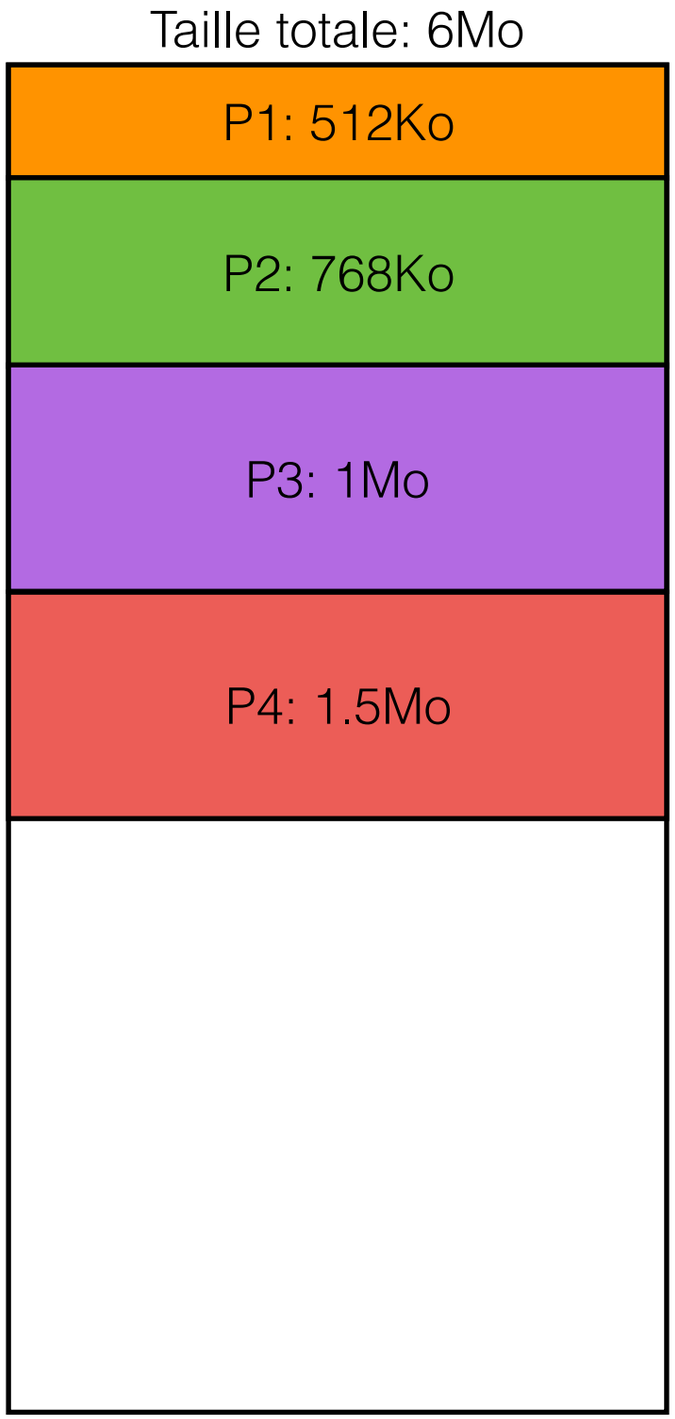


Allocation mémoire contigüe, taille **variable**



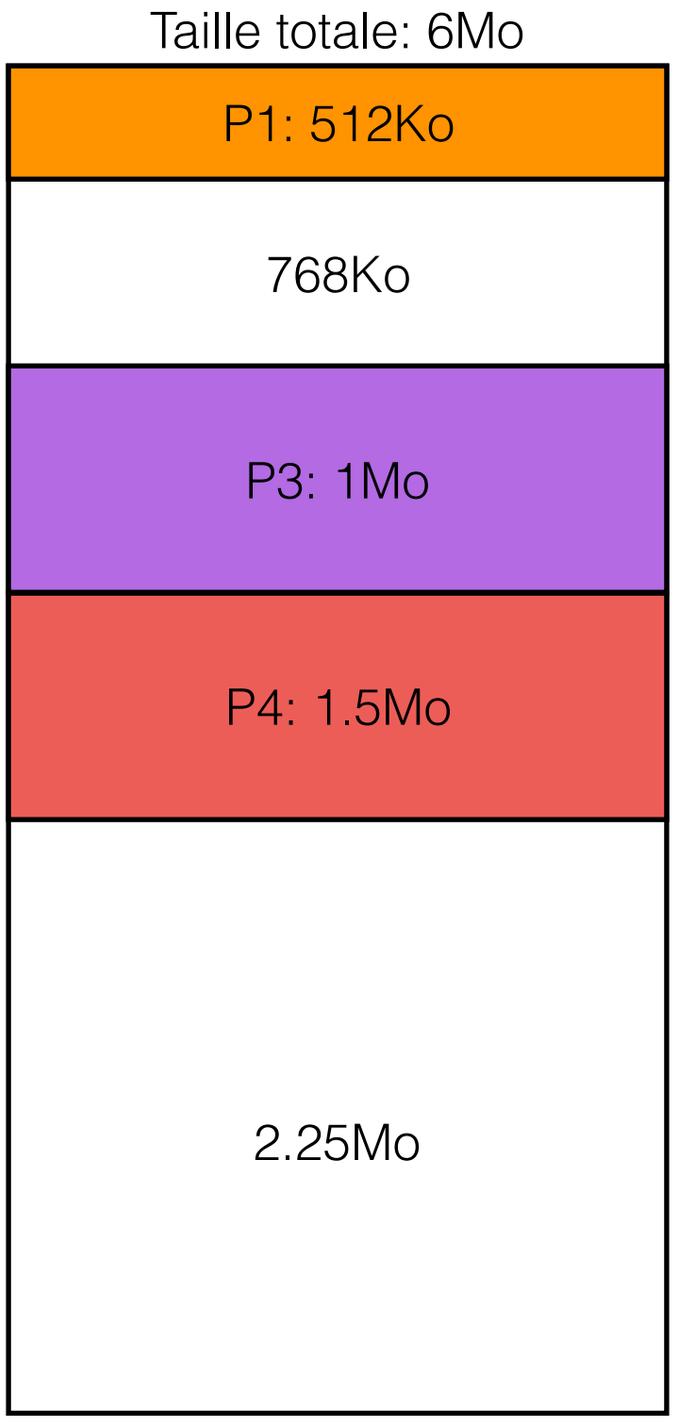
Allocation mémoire contigüe, taille **variable**

Qu’arrive-t-il si un processus (ex: P2) se termine?



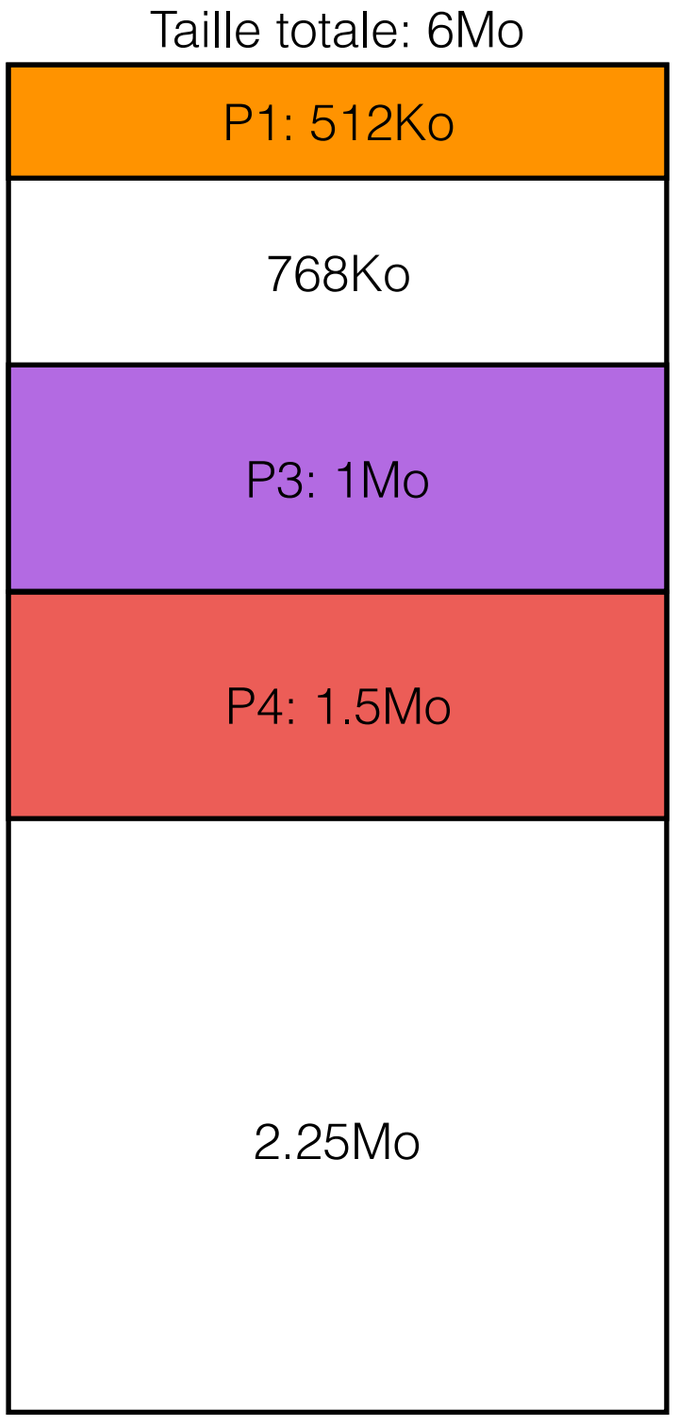
Allocation mémoire contigüe, taille **variable**

Cela crée un trou!



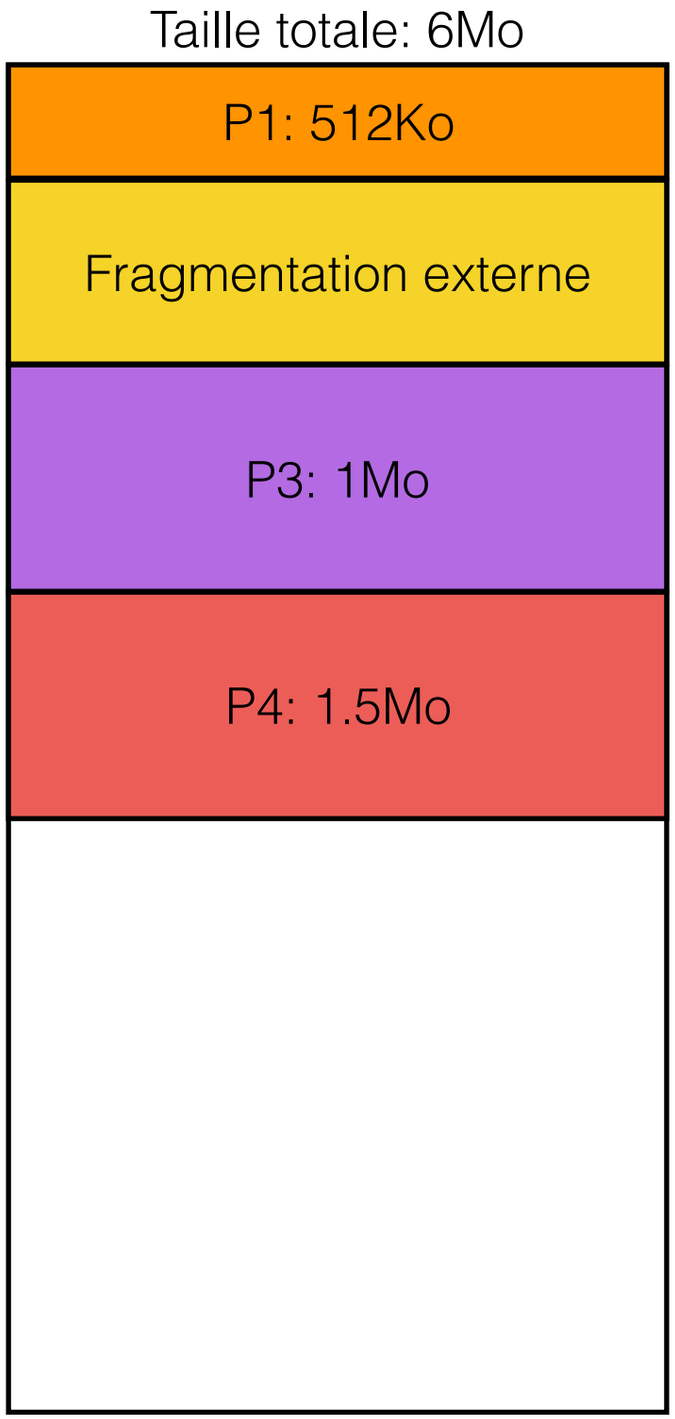
Allocation contigüe, taille fixe

- Quels sont les problèmes avec l'allocation contigüe avec partitions de taille fixe?
- Lorsqu'un processus se termine, il peut laisser des «trous».



Fragmentation

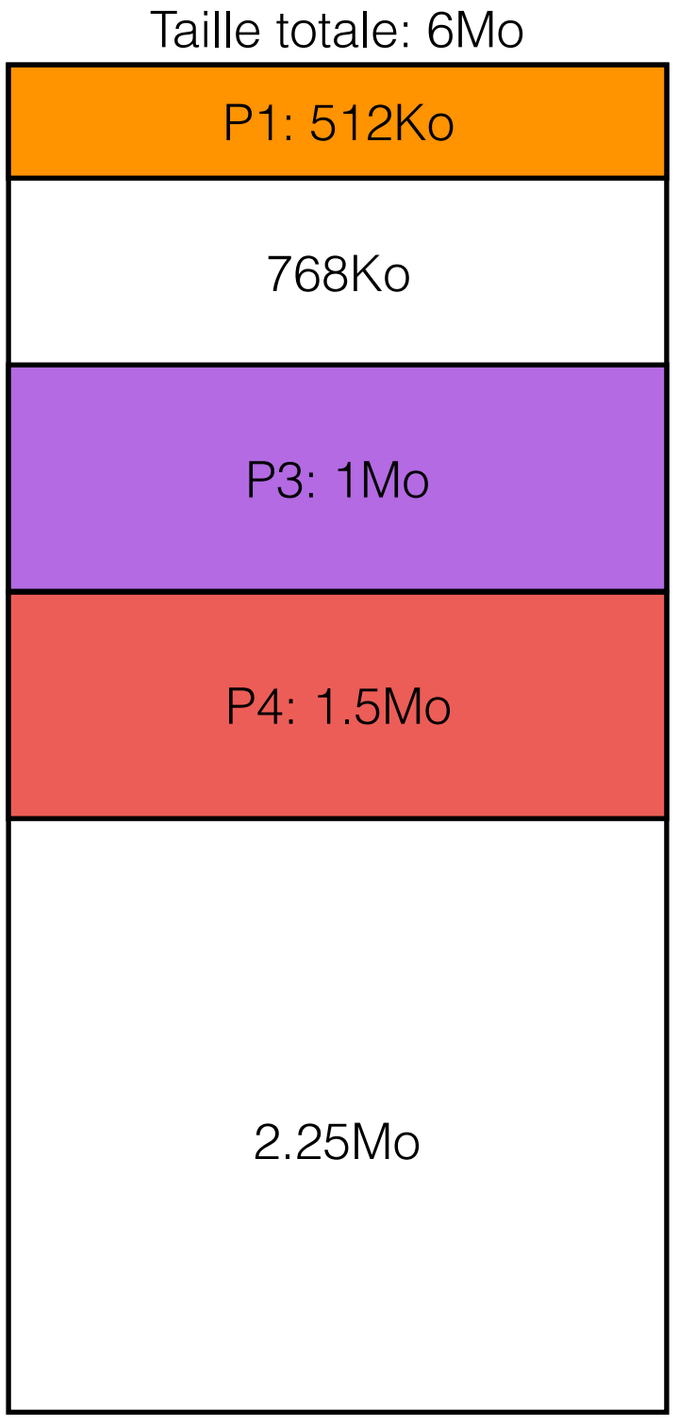
- Il en existe 2 types:
  - Fragmentation **interne**: espace perdu **à l'intérieur** une partition
  - Fragmentation **externe**: espace perdu **à l'extérieur** d'une partition

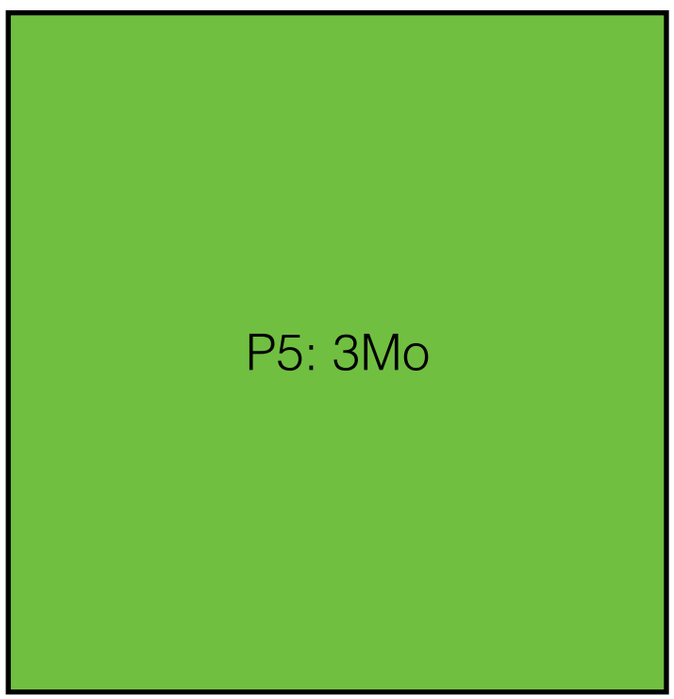


- Avec des partitions de taille **variable**, seule la fragmentation **externe** est possible
  - aucun espace à l'intérieur d'une partition n'est perdu car chaque partition est créée en fonction de la taille de son processus

Allocation contigüe, taille variable

- Quels sont les problèmes avec l'allocation contigüe avec partitions de taille variable?
  - Lorsqu'un processus se termine, il peut laisser des «trous».
  - Que faire si aucun «trou» n'est assez grand pour un processus?





Ne peut être admis en mémoire!
(même si l'espace total libre est suffisant: 2.25Mo+768Ko = 3Mo)

Algorithmes d'allocation de mémoire

- Il existe plusieurs algorithmes afin de déterminer l'emplacement d'un programme en mémoire. Le but de tous ces algorithmes est de maximiser l'espace mémoire occupé.
  - **Première allocation (« First Fit »)**: Le programme est mis dans le premier bloc de mémoire suffisamment grand à partir du début de la mémoire.
    - Souvent utilisé malgré sa simplicité apparente!
  - **Prochaine allocation (« Next Fit »)**: Le programme est mis dans le premier bloc de mémoire suffisamment grand à partir du dernier bloc alloué.
    - Crée souvent un peu plus de fragmentation que « First Fit »
- **Meilleure allocation (« Best Fit »)**: Le programme est mis dans le bloc de mémoire le plus petit dont la taille est suffisamment grande pour l'espace requis.
  - Semble meilleur, mais demande beaucoup de temps de calcul!
- **Pire allocation (« Worse Fit »)**: Le programme est mis dans le bloc de mémoire le plus grand.

Exercice mémoire contigüe #1

- Effectuez les étapes suivantes pour un système ayant une mémoire de 16Mo:
  - Les processus suivants sont alloués en mémoire (dans l'ordre)
    - P1, 3Mo
    - P2, 5Mo
    - P3, 2Mo
    - P4, 2Mo
- P1 et P3 se terminent
- À quel endroit en mémoire le processus (P5, 2Mo) sera-t-il alloué si l'on emploie chacun des algorithmes d'allocation mémoire

Exercice mémoire contigüe #1

- Le processus P5 sera placé à des endroits différents, en fonction de l'algorithme utilisé:
  - Première allocation: 0—2
  - Meilleure allocation: 8—10
  - Prochaine (ou pire) allocation: 12—14

Exercice mémoire contigüe #2

- Un système possédant une mémoire de 16Mo doit allouer les processus suivants:

Processus	Taille	Temps de création	Durée
P1	3Mo	0	3
P2	2Mo	1	3
P3	4Mo	2	3
P4	2Mo	3	3
P5	3Mo	4	3

- Indiquez le contenu de la mémoire après l'allocation du processus P5

Exercice mémoire contigüe #2

Processus	Taille	Temps de création	Durée
P1	3Mo	0	3
P2	2Mo	1	3
P3	4Mo	2	3
P4	2Mo	3	3
P5	3Mo	4	3

Allocation contigüe: MMU

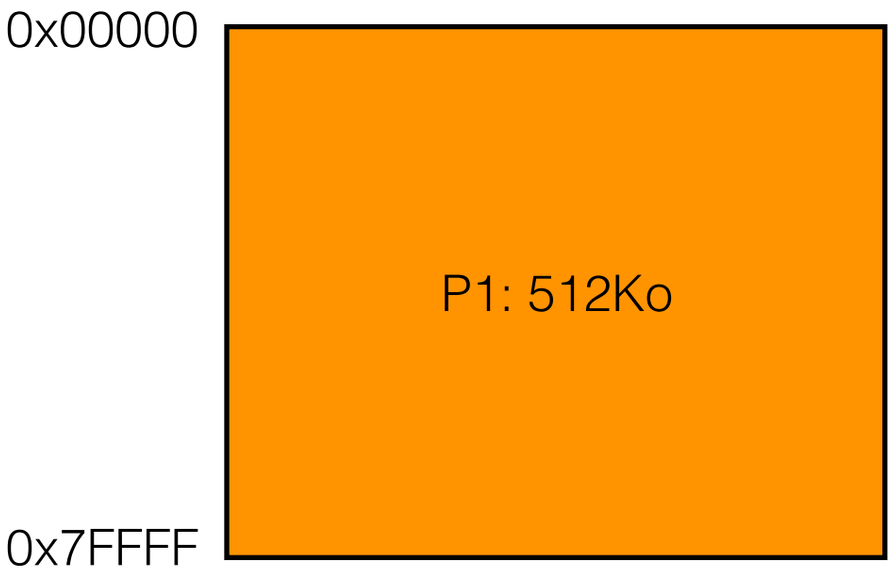
- En allocation contigüe, le **MMU** effectue le calcul suivant pour traduire une adresse **virtuelle** en adresse **physique**:

$a_{partition}$ est la première adresse de la partition attribuée au processus

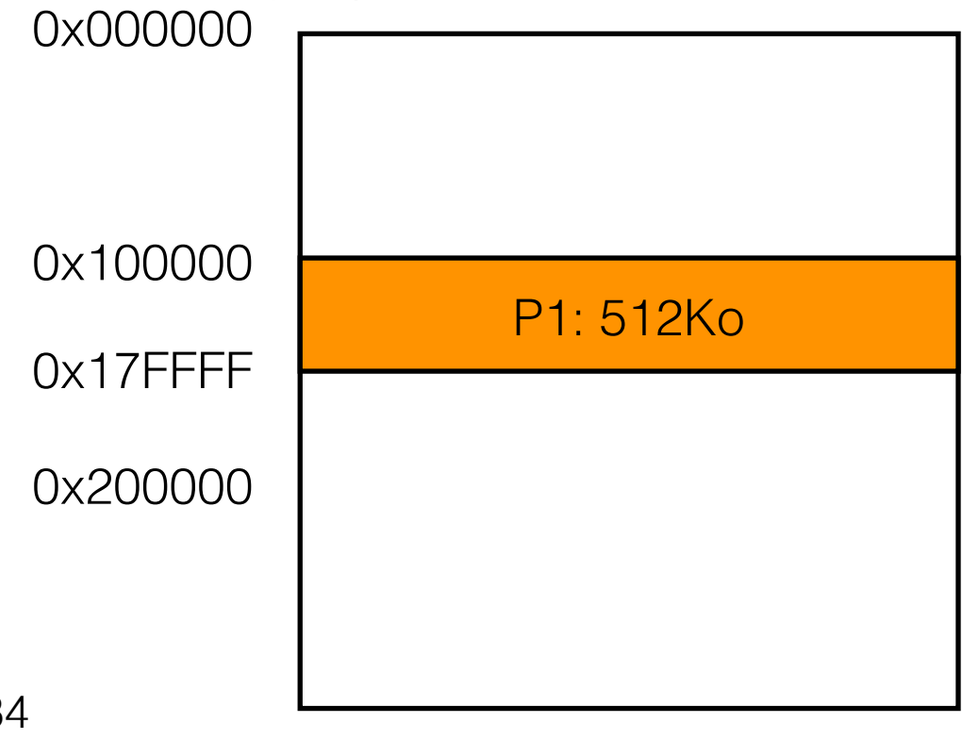
$$a_{physique} = a_{virtuelle} + a_{partition}$$

- Par exemple, l'adresse virtuelle 0x00AB3 correspond à l'adresse physique 0x100AB3 dans l'exemple ci-bas.

Adresses **virtuelles**
(du point de vue du **processus**)



Adresses **physiques**
(du point de vue de la **mémoire**)



Exercice mémoire virtuelle #1

- Dans un système en allocation contigüe avec partitions à taille variable, un processus P1 occupe les adresses physiques 0x2000 à 0x4FFF.
    - Aux «yeux» de P1, quelle est sa plage d'adresses virtuelles disponible?
    - Quelle est la taille totale de mémoire virtuelle disponible pour ce processus?
    - Quelle est l'adresse physique correspondant aux adresses virtuelles:
        - 0x0010?
        - 0x1234?

Exercice mémoire virtuelle #1

- Dans un système en allocation contigüe avec partitions à taille variable, un processus P1 occupe les adresses mémoire 0x2000 à 0x4FFF.
  - Aux «yeux» de P1, quelle est sa plage d'adresses (virtuelles) disponible?
    - 0x0000 à 0x2FFF
  - Quelle est la taille totale de mémoire virtuelle disponible pour ce processus?
    - 12Ko
- Quelle est l'adresse physique correspondant aux adresses virtuelles:
  - 0x0010?
    - 0x2010
  - 0x1234?
    - 0x3234

Récapitulation: allocation contigüe

1. Un nouveau programme est copié dans un emplacement disponible en mémoire, dans un bloc de mémoire contigu.

2. On peut créer des partitions de taille:

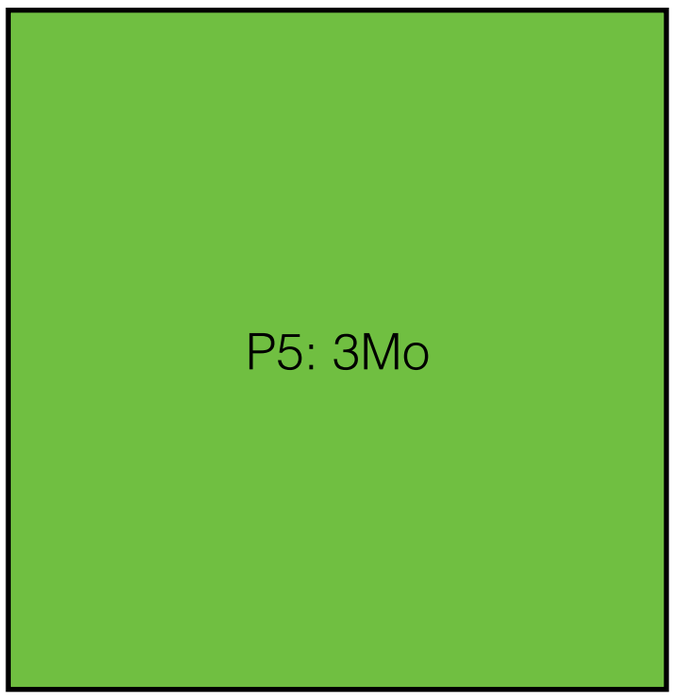
    - **fixe**: la première partition disponible est choisie quand un nouveau processus doit être alloué
    - **variable**: on doit déterminer où créer la partition, nécessite le choix d'un algorithme d'allocation mémoire plus compliqué

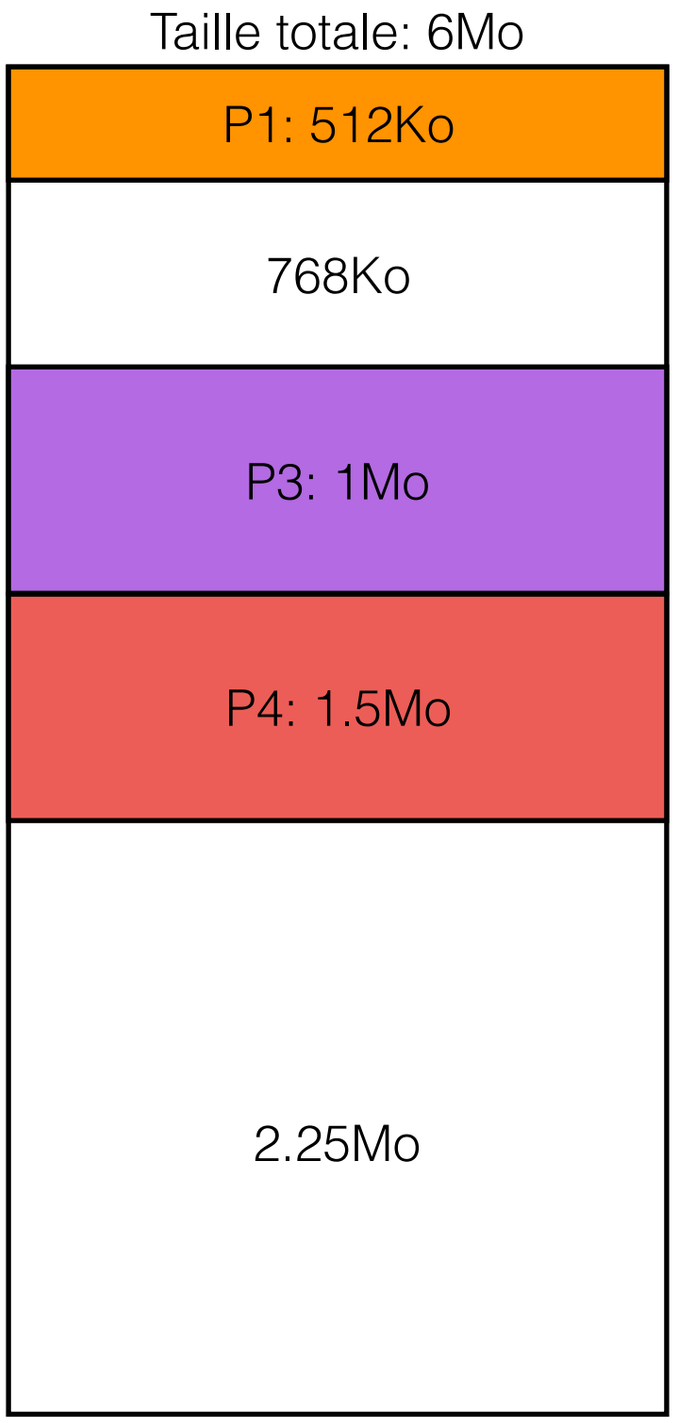
3. Le programme utilise des adresses «virtuelles»

4. Le **MMU** traduit les adresse **virtuelles** en adresses **physiques**

Question

Que faire avec P5?

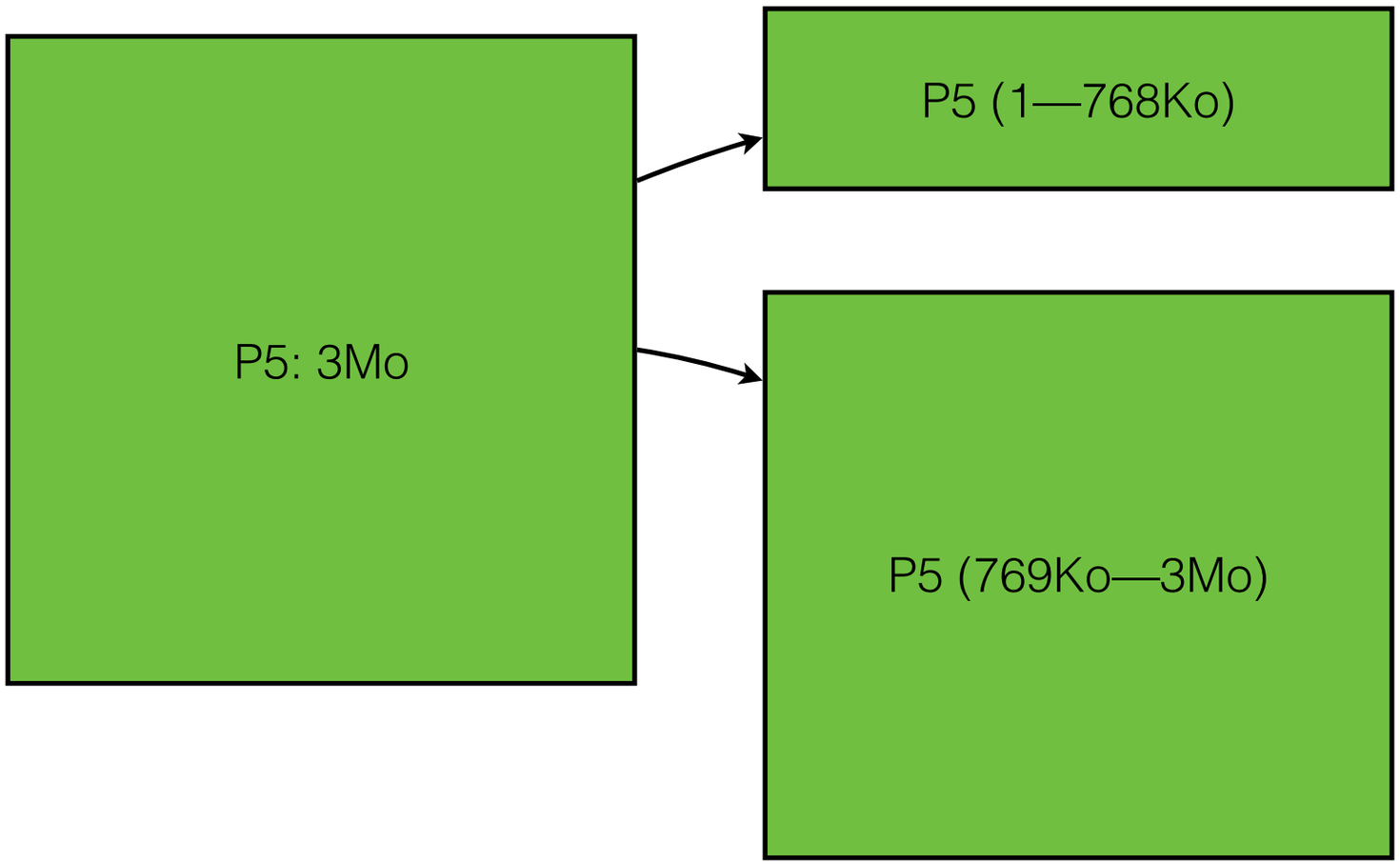


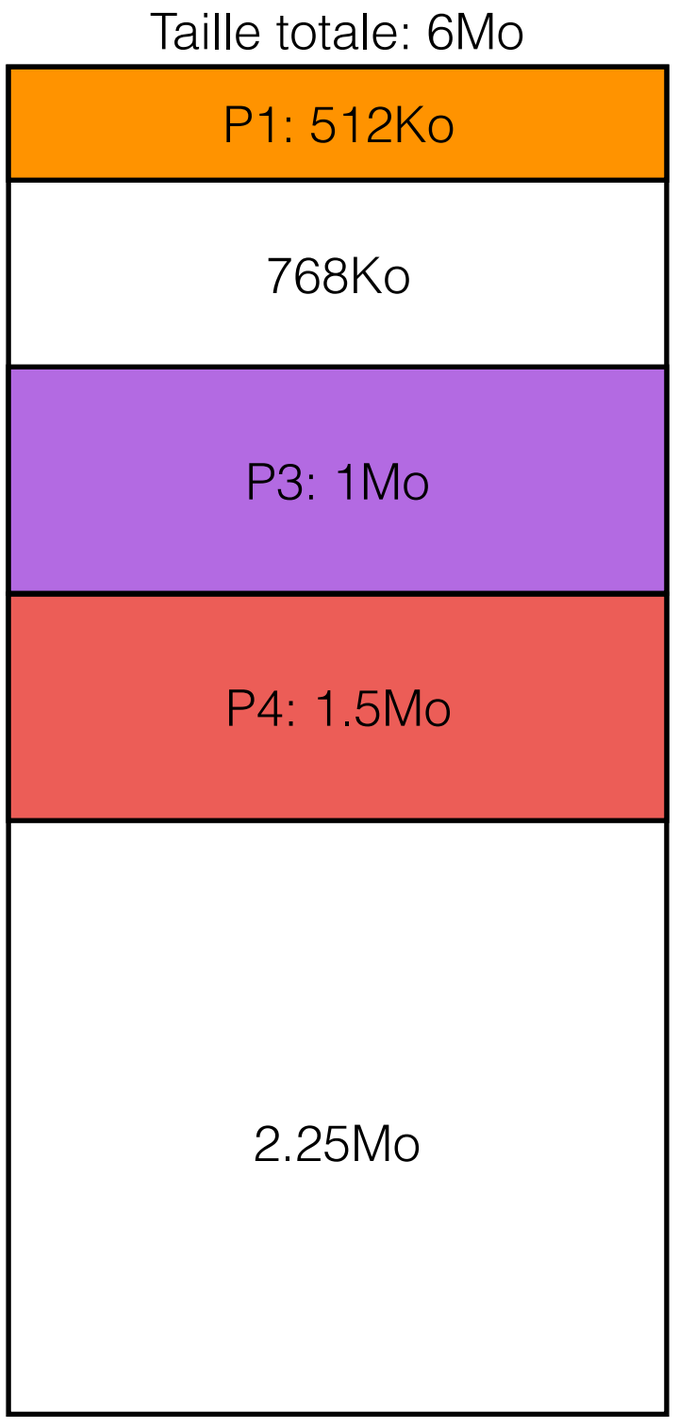


Question

Que faire avec P5?

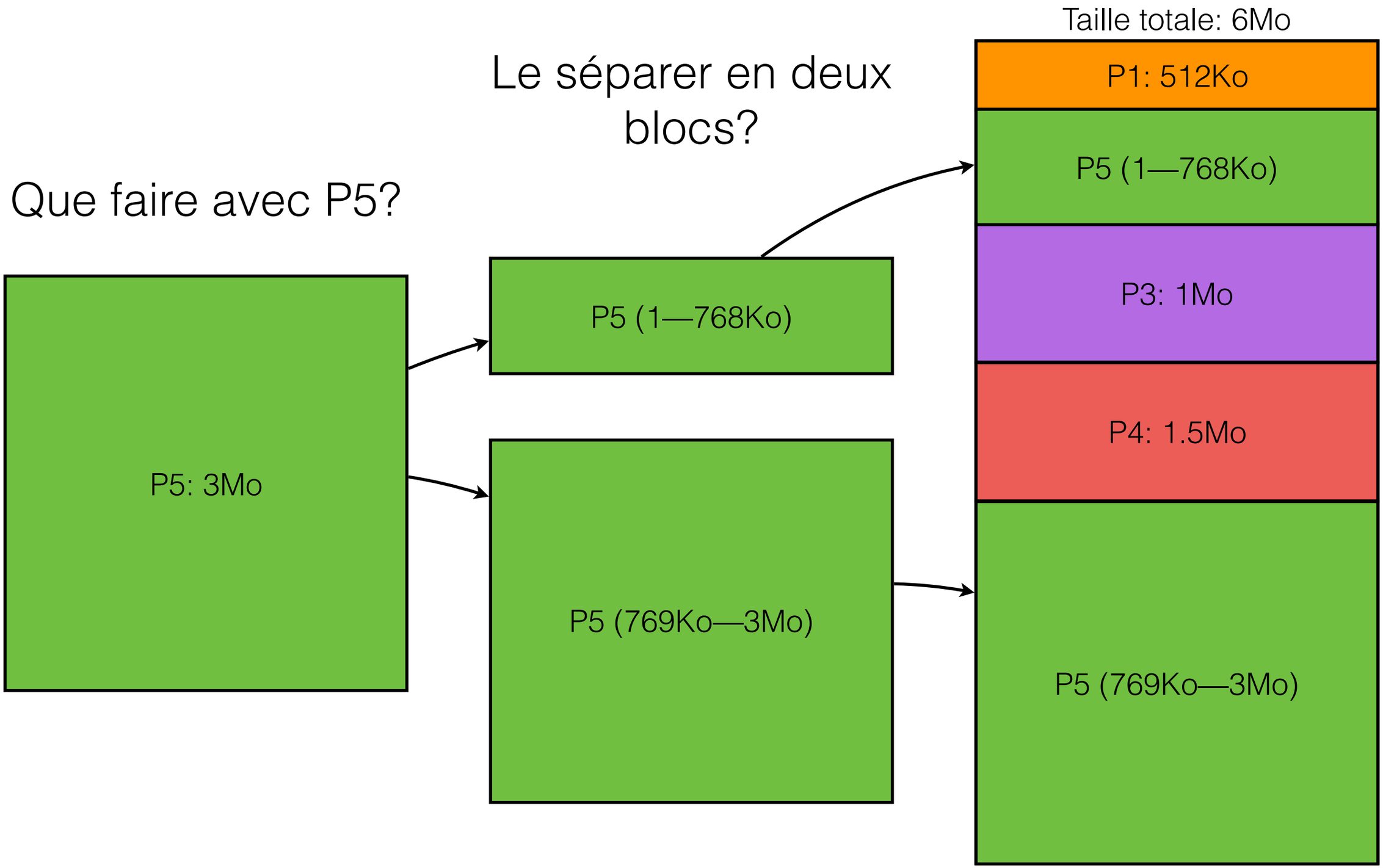
Le séparer en deux blocs?





Question

Placer chacun des blocs dans les espaces libres?



Question

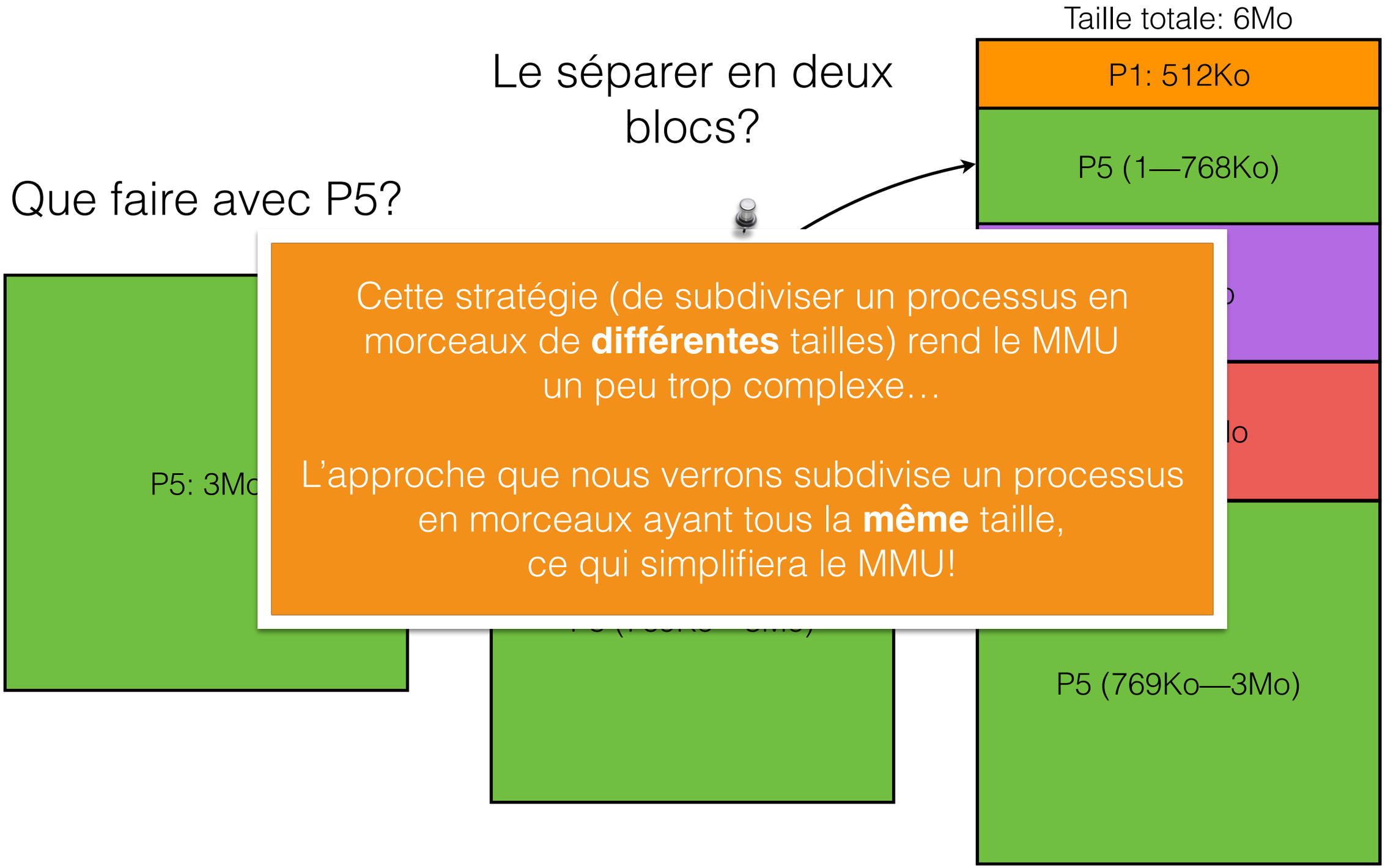
Placer chacun des blocs dans les espaces libres?

Le séparer en deux blocs?

Que faire avec P5?

Cette stratégie (de subdiviser un processus en morceaux de **différentes** tailles) rend le MMU un peu trop complexe…

L'approche que nous verrons subdivise un processus en morceaux ayant tous la **même** taille, ce qui simplifiera le MMU!

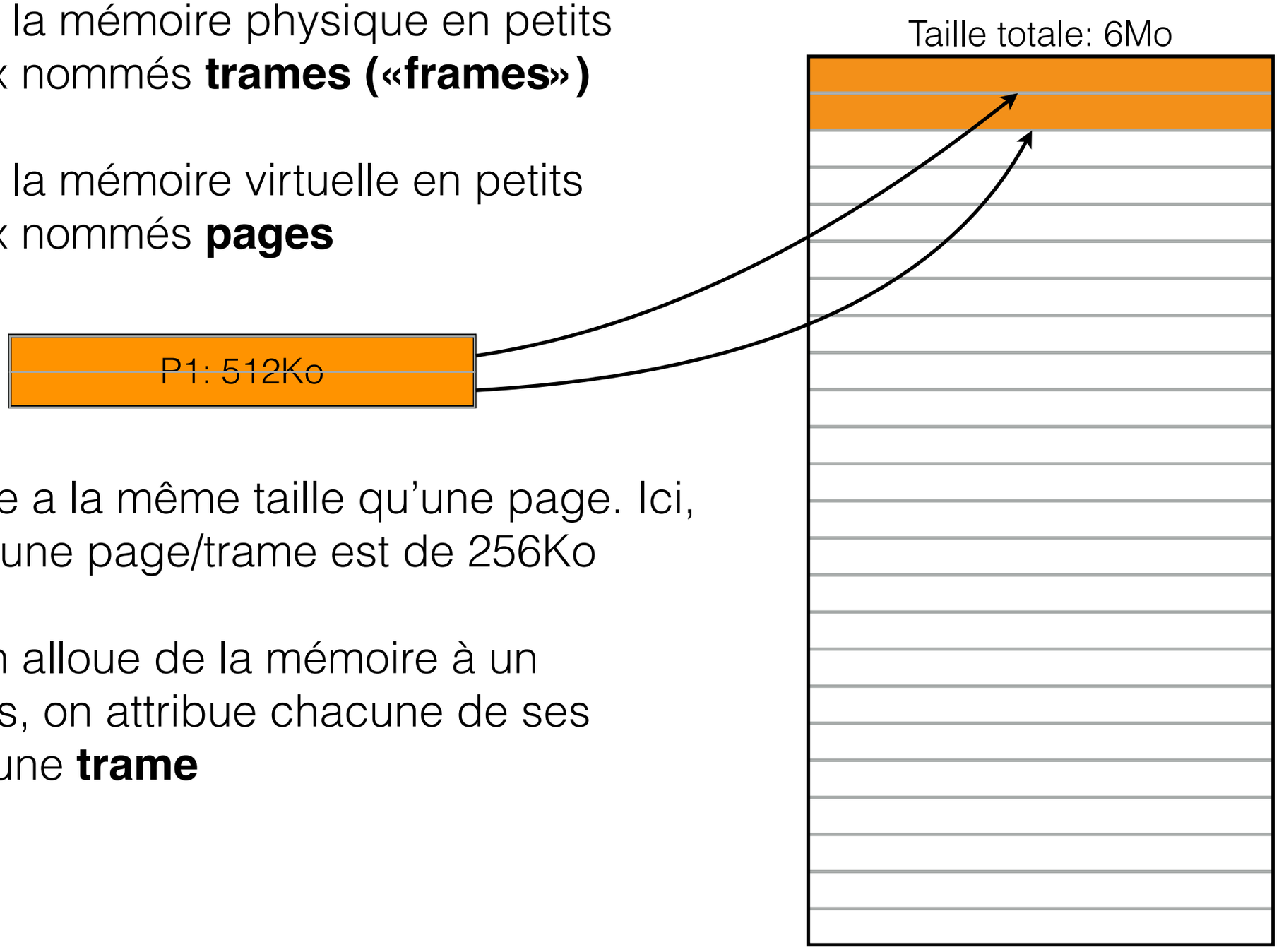


Allocation mémoire paginée

Allocation mémoire paginée

- On divise la mémoire physique en petits morceaux nommés **trames («frames»)**

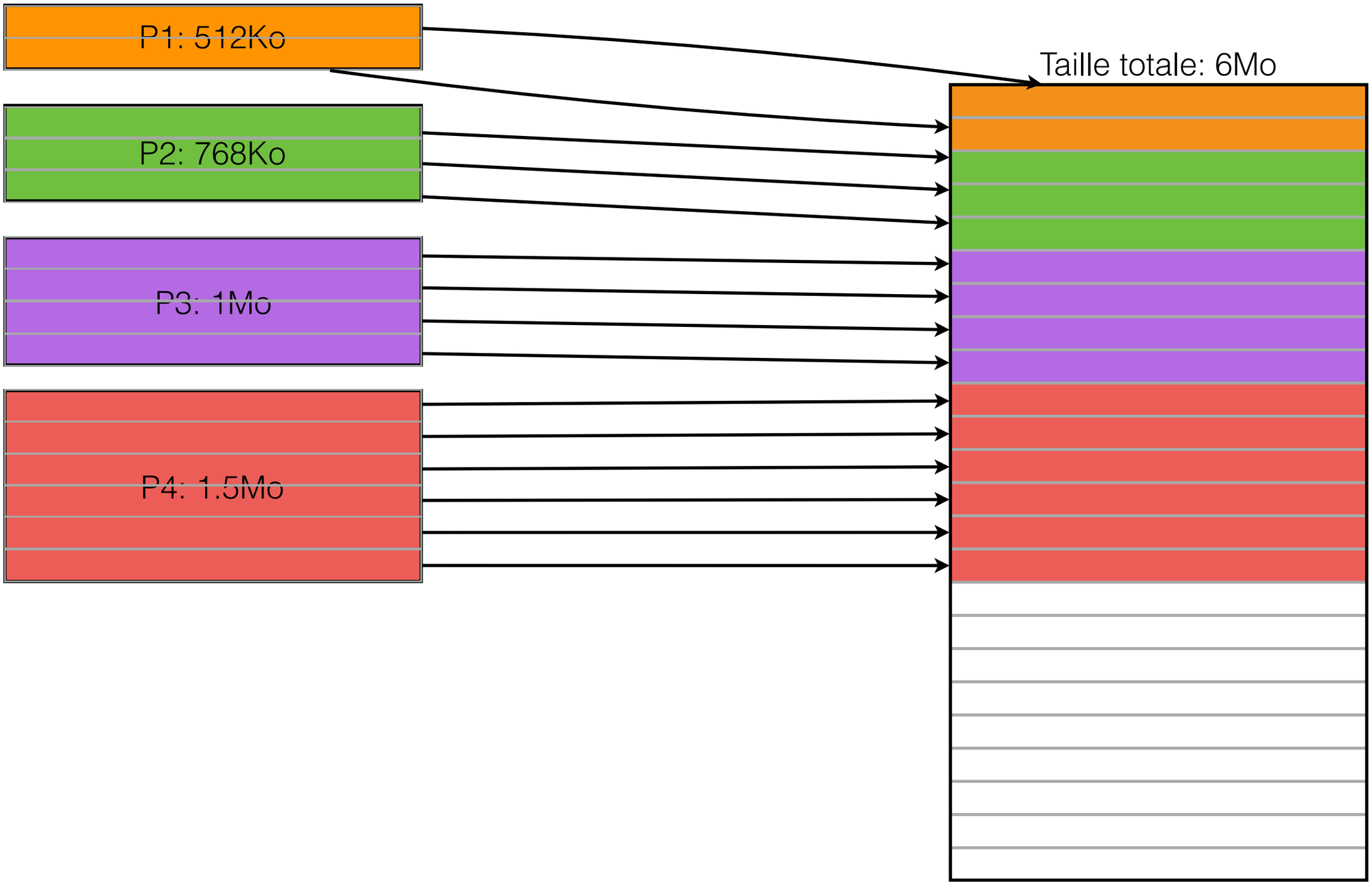
- On divise la mémoire virtuelle en petits morceaux nommés **pages**



- Une trame a la même taille qu'une page. Ici, la taille d'une page/trame est de 256Ko

- Lorsqu'on alloue de la mémoire à un processus, on attribue chacune de ses **pages** à une **trame**

Allocation mémoire paginée



Allocation mémoire paginée

P1: 512Ko

P3: 1Mo

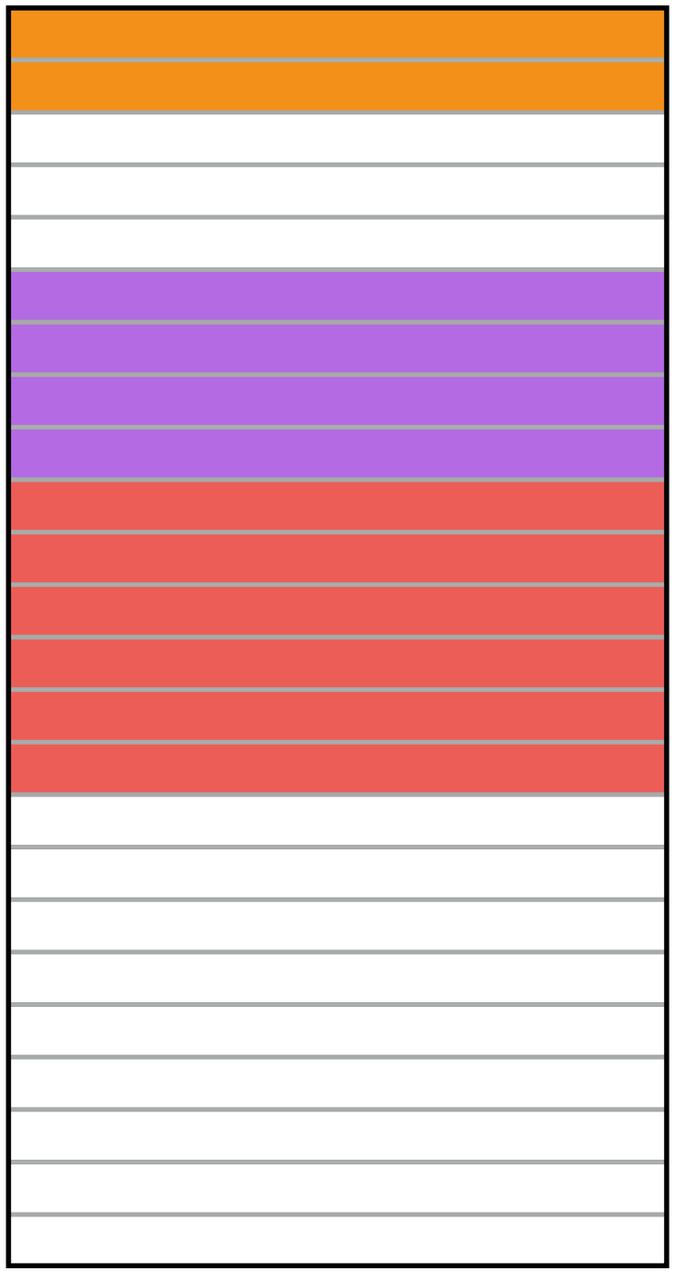
P4: 1.5Mo

P5: 3Mo

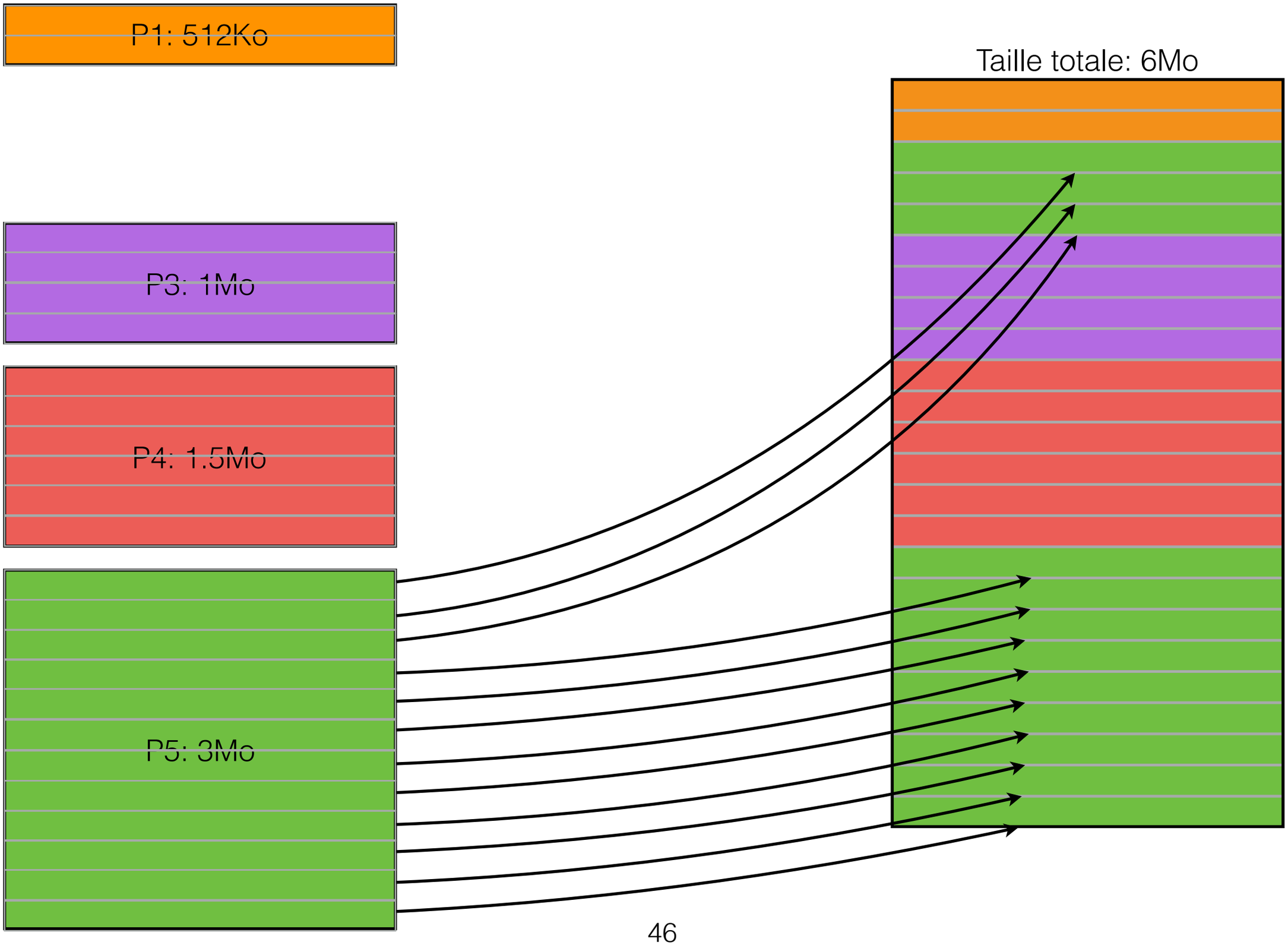
P2 se termine

P5 est admis

Taille totale: 6Mo



Allocation mémoire paginée



Allocation mémoire paginée: **MMU**

- Dans le cas de l'allocation mémoire paginée, le MMU est plus complexe que pour l'allocation contigüe
    - On ne peut plus simplement additionner la première adresse!
- Pour traduire l'adresse, il faut effectuer trois étapes:

1. Déterminer la **page** de l'adresse **virtuelle**;
2. Trouver la **trame** (dans la mémoire **physique**) correspondant à cette page;
3. Remplacer le numéro de page par le numéro de trame.

1. Déterminer la **page** de l'adresse **virtuelle**

- Une adresse virtuelle est divisée en deux:
  - le numéro de page (MSB)
  - la position dans la page (LSB)
- Le nombre de bits utilisé pour ces deux parties dépend de la **taille d'une page**
- Prenons, par exemple, des pages de 4Ko. Combien de bits avons-nous besoin pour encoder la position de chaque octet dans la page?
  - il y a 4K octets dans une page, donc $4K = 4 \times 2^{10}$
  - $\log_2(4 \times 2^{10}) = \log_2(2^2 \times 2^{10}) = \log_2(2^{12}) = 12$ bits

Quelle position dans quelle page réfère cette adresse?

0x 0 0 0 3 D 7 A 1

	Numéro de page					Position		
0x	0	0	0	3	D	7	A	1

2. Trouver la **trame** correspondant à cette **page**

- Comment fait-on pour savoir
  - l'endroit où une page est stockée?
  - la correspondance entre une page et une trame?
- On utilise la **table des pages**, une structure de données dans le MMU qui conserve la correspondance entre chaque:
  - **page** de la mémoire **virtuelle**
  - **trame** de la mémoire **physique**

La table des pages

- La **table des pages** est une structure de données dans le MMU qui conserve la correspondance entre chaque:
  - **page** de la mémoire **virtuelle**
  - **trame** de la mémoire **physique**

En pratique, on ne stocke pas le numéro de page.

Le numéro de **ligne** dans la table correspond au numéro de **page**.

# page	# trame
⋮	⋮
0x3B	0x3A2
0x3C	0x123
0x3D	0x2F3
0x3E	0x5A0
0x3F	0x2F0
0x40	0x10F
⋮	⋮

3. Remplacer le # de **page** par le # de **trame**

- Grâce à la table des pages, on sait que la page 0x3D correspond à la trame 0x2F3

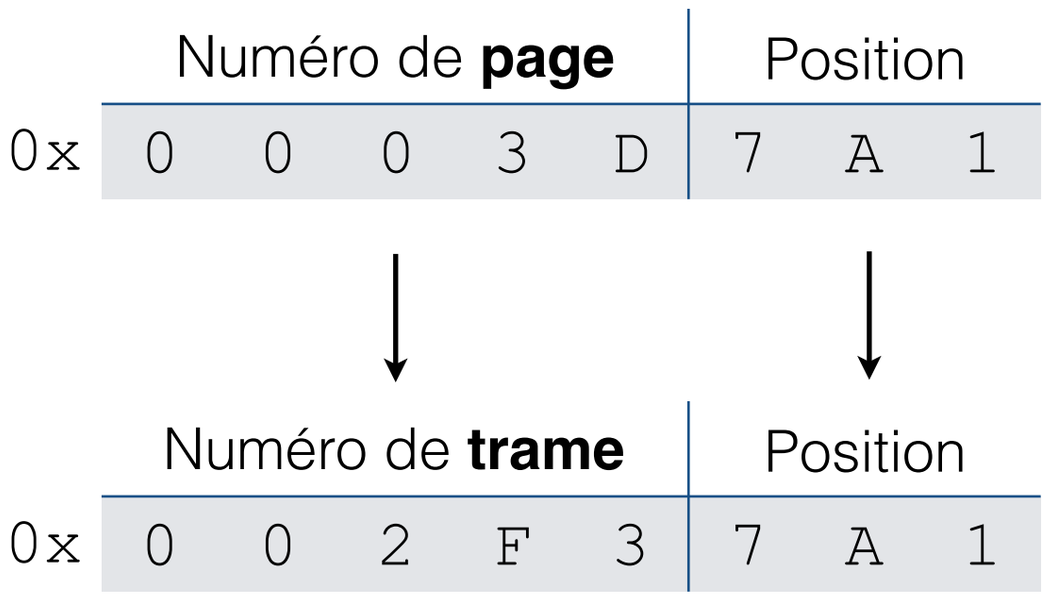
- Il suffit de remplacer le numéro de page par le numéro de trame pour obtenir l'adresse **physique**

Adresse virtuelle

0x 0 0 0 3 D 7 A 1

Adresse physique

0x 0 0 2 F 3 7 A 1



Taille de la table des pages

- La table des pages est définie par:
  - le nombre de lignes: il y a une ligne par page, donc c'est équivalent au **nombre de pages**
  - l'information stockée dans chaque ligne: le numéro de trame correspondant à chaque page.
    - notez qu'il n'est pas nécessaire de stocker le numéro de page, nous n'avons qu'à lire la ligne correspondant à la page.
      - par exemple, la page 0x2D (45) est à la ligne 45 dans la table
    - combien de bits sont nécessaires pour représenter le numéro de trame?
      - cela dépend du nombre total de trames dans le système!

Taille de la table des pages

- Un système possède les caractéristiques suivantes:
  - une mémoire *physique* de 32Mo
  - une mémoire *virtuelle* de 64Mo
  - la taille d'une page est de 4Ko
- Quelle est la taille de la table des pages?

Taille de la table des pages

- Un système possède les caractéristiques suivantes:
  - une mémoire *physique* de 32Mo
  - une mémoire *virtuelle* de 64Mo
  - la taille d'une page est de 4Ko

Rappel
pages = mémoire **virtuelle**
trames = mémoire **physique**

- Quelle est la taille de la table des pages?
  - Déterminer le nombre de *pages*

$$64\text{Mo} / 4\text{Ko} = 64 \times 2^{20} / 4 \times 2^{10} = 2^{26} / 2^{12} = 2^{14} \text{ pages}$$

  - Déterminer le nombre de *trames*

$$32\text{Mo} / 4\text{Ko} = 32 \times 2^{20} / 4 \times 2^{10} = 2^{25} / 2^{12} = 2^{13} \text{ trames}$$

  - Déterminer le nombre de bits nécessaires pour stocker le # de trame

$$\log_2(2^{13}) = 13 \text{ bits}$$

  - Calculer la taille totale (#pages x #bits)

$$2^{14} \times 13 = 2^4 \times 13 \times 2^{10} = 208 \times 2^{10} = 208\text{Kbits} = 26\text{Ko}$$

FAQ

Est-ce qu'un processus doit être entièrement en mémoire?

Non!
Seules les pages nécessaires peuvent être chargées en mémoire.

FAQ

Laquelle est plus grosse:
la mémoire **virtuelle** ou la mémoire **physique**?

Ou est-ce que les deux doivent avoir la même taille?

La mémoire **virtuelle** peut être beaucoup plus grosse
que la mémoire **physique**!

Conditions

- 2 conditions pour qu'un programme puisse s'exécuter:
  - l'instruction (ou la donnée) nécessaire doit être en mémoire RAM
  - la table de pages pour ce programme doit contenir une entrée qui traduit l'adresse (virtuelle) du programme vers l'adresse (physique) en RAM

Table des pages inversée

- Une table des pages indique:
  - le numéro de trame correspondant à chaque page.
- Il peut aussi être utile de savoir l'inverse:
  - le numéro de page correspondant à chaque trame.
- C'est la **table des pages inversée**.
- La taille d'une page de table inversée est sa taille est proportionnelle à la taille de la mémoire **physique**.

Illustration de la mémoire virtuelle

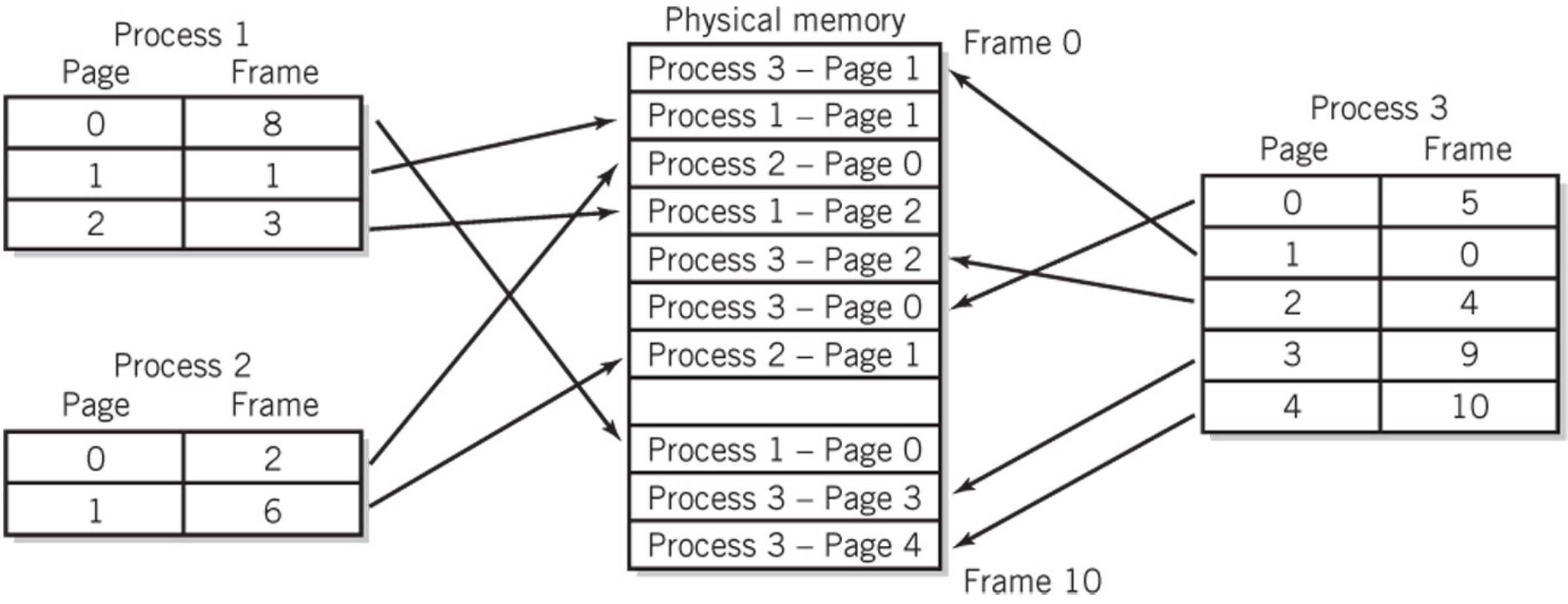


Table de pages inversées

Englander, ch. 18

Table des pages et table des pages inversée

- Caractéristiques du système:
  - Pages de 4Ko
  - Mémoire physique (RAM) de 16Ko
  - Mémoire virtuelle de 32Ko
- Effectuez la séquence d'opérations suivantes en mettant à jour les tables à droite, et indiquez les adresses physiques correspondantes
  - Charger valeur (e.g. LDR) à l'adresse 0x0A38
  - Écrire une valeur (e.g. STR) à l'adresse 0x3B97
  - Charger la valeur (e.g. LDR) à l'adresse 0x2928
  - Brancher (e.g. B) à l'adresse 0x7BF0

Table des pages

# page	# trame
0	1
1	2
2	
3	0
4	
5	
6	
7	

Table des pages inversée

# trame	# page
0	page 3
1	page 0
2	page 1
3	

Table des pages et table des pages inversée

- Caractéristiques du système:
  - Pages de 4Ko
  - Mémoire physique (RAM) de 16Ko
  - Mémoire virtuelle de 32Ko
- Effectuez la séquence d'opérations suivantes en mettant à jour les tables à droite, et indiquez les adresses physiques correspondantes
  - Charger valeur (e.g. LDR) à l'adresse 0x0A38
    - l'adresse physique est 0x1A38
  - Écrire une valeur (e.g. STR) à l'adresse 0x3B97
  - Charger la valeur (e.g. LDR) à l'adresse 0x2928
  - Brancher (e.g. B) à l'adresse 0x7BF0

Table des pages

# page	# trame
0	**1**
1	2
2	
3	0
4	
5	
6	
7	

Table des pages inversée

# trame	# page
0	page 3
1	page 0
2	page 1
3	

Table des pages et table des pages inversée

- Caractéristiques du système:
  - Pages de 4Ko
  - Mémoire physique (RAM) de 16Ko
  - Mémoire virtuelle de 32Ko
- Effectuez la séquence d'opérations suivantes en mettant à jour les tables à droite, et indiquez les adresses physiques correspondantes
  - Charger valeur (e.g. LDR) à l'adresse 0x0A38
  - Écrire une valeur (e.g. STR) à l'adresse 0x3B97
    - l'adresse physique est 0x0B97
  - Charger la valeur (e.g. LDR) à l'adresse 0x2928
  - Brancher (e.g. B) à l'adresse 0x7BF0

Table des pages

# page	# trame
0	1
1	2
2	
3	**0**
4	
5	
6	
7	

Table des pages inversée

# trame	# page
0	page 3
1	page 0
2	page 1
3	

Table des pages et table des pages inversée

- Caractéristiques du système:
  - Pages de 4Ko
  - Mémoire physique (RAM) de 16Ko
  - Mémoire virtuelle de 32Ko
- Effectuez la séquence d'opérations suivantes en mettant à jour les tables à droite, et indiquez les adresses physiques correspondantes
  - Charger valeur (e.g. LDR) à l'adresse 0x0A38
  - Écrire une valeur (e.g. STR) à l'adresse 0x3B97
  - Charger la valeur (e.g. LDR) à l'adresse 0x2928
    - Faute de page!
  - Brancher (e.g. B) à l'adresse 0x7BF0

Table des pages

# page	# trame
0	1
1	2
2	**?**
3	0
4	
5	
6	
7	

Table des pages inversée

# trame	# page
0	page 3
1	page 0
2	page 1
3	

Allocation et désallocation des pages

- Les pages allouées à un programme peuvent se retrouver séparées dans la mémoire. L'allocation de pages est très simple: il suffit de maintenir une liste des pages libres et de retirer une page libre pour l'allouer.

- Il est possible que plusieurs programmes utilisent une même page s'ils ont du code en commun.

- Désallouer des pages est simple: il suffit de mettre une page dans les pages libres.

- Qu'arrive-t-il si un programme a besoin d'une page qui n'existe pas?

Fautes de page

- Une **faute de page** survient lorsque le processus requiert une page qui n'est pas en mémoire.

- S'il y a de la place dans la table des pages:

  1. chercher la page sur le disque dur;

  2. la copier en mémoire RAM (physique);

  3. mettre à jour la table des pages.

- Sinon

  - il faut remplacer une autre page par la page requise.

    - Laquelle? Habituellement, celle qui a été utilisée le moins récemment.

Table des pages et table des pages inversée

- Caractéristiques du système:
  - Pages de 4Ko
  - Mémoire physique (RAM) de 16Ko
  - Mémoire virtuelle de 32Ko
- Effectuez la séquence d'opérations suivantes en mettant à jour les tables à droite, et indiquez les adresses physiques correspondantes
  - Charger valeur (e.g. LDR) à l'adresse 0x0A38
  - Écrire une valeur (e.g. STR) à l'adresse 0x3B97
  - Charger la valeur (e.g. LDR) à l'adresse 0x2928
    - L'adresse physique est 0x3928
  - Brancher (e.g. B) à l'adresse 0x7BF0

Table des pages

# page	# trame
0	1
1	2
2	**3**
3	0
4	
5	
6	
7	

Table des pages inversée

# trame	# page
0	page 3
1	page 0
2	page 1
3	**page 2**

Table des pages et table des pages inversée

- Caractéristiques du système:
  - Pages de 4Ko
  - Mémoire physique (RAM) de 16Ko
  - Mémoire virtuelle de 32Ko
- Effectuez la séquence d'opérations suivantes en mettant à jour les tables à droite, et indiquez les adresses physiques correspondantes
  - Charger valeur (e.g. LDR) à l'adresse 0x0A38
  - Écrire une valeur (e.g. STR) à l'adresse 0x3B97
  - Charger la valeur (e.g. LDR) à l'adresse 0x2928
  - Brancher (e.g. B) à l'adresse 0x7BF0
    - Faute de page! Quelle trame utiliser? Que faire avec le contenu de cette trame?

Table des pages

# page	# trame
0	1
1	2
2	3
3	0
4	
5	
6	
7	**?**

Table des pages inversée

# trame	# page
0	page 3
1	page 0
2	page 1
3	page 2

Table des pages et table des pages inversée

- Caractéristiques du système:
  - Pages de 4Ko
  - Mémoire physique (RAM) de 16Ko
  - Mémoire virtuelle de 32Ko
- Effectuez la séquence d'opérations suivantes en mettant à jour les tables à droite, et indiquez les adresses physiques correspondantes
  - Charger valeur (e.g. LDR) à l'adresse 0x0A38
  - Écrire une valeur (e.g. STR) à l'adresse 0x3B97
  - Charger la valeur (e.g. LDR) à l'adresse 0x2928
  - Brancher (e.g. B) à l'adresse 0x7BF0
    - L'adresse physique est 0x2BF0

Table des pages

# page	# trame
0	1
1	~~**2**~~
2	3
3	0
4	
5	
6	
7	**2**

Table des pages inversée

# trame	# page
0	page 3
1	page 0
2	**page 7**
3	page 2

Table des pages et table des pages inversée

- Tables finales:

Table des pages

# page	# trame
0	1
1	
2	3
3	0
4	
5	
6	
7	2

Table des pages inversée

# trame	# page
0	page 3
1	page 0
2	page 7
3	page 2

Récapitulation: allocation paginée

- La mémoire:
  - **virtuelle** est divisée en **pages**;
  - **physique** est divisée en **trames**;
- Les pages et les trames ont la même taille;
- La **table des pages** stocke la correspondance entre une page et une trame;
- La traduction d'adresse implique 3 étapes:
  1. Déterminer la page de l'adresse virtuelle;
  2. Trouver la trame (dans la mémoire physique) correspondant à cette page;
  3. Remplacer le numéro de page par le numéro de trame.
- La **table des pages inversée** stocke la correspondance entre une trame et une page;
- Lorsqu'une page n'est pas chargée en mémoire, il y a **faute de page**.

Exercice 1: Exemple de calculs reliés aux tables de pages

- Supposons une mémoire de 16 Mo et un système d'exploitation supportant des pages de 4Ko pour des programmes ayant 64 Mo maximum. Supposons que 4 bits par page de programme sont utilisés pour donner des informations supplémentaires

- Supposons l'extrait de la table de page pour le programme X suivant:

#page virtuelle	#frame de la mémoire	Bits d'information
…	…	…
0x14	B	En mémoire, modifiée
0x13	4	En mémoire
0x12	C	En mémoire, modifiée
0x11	2	En mémoire
…	…	…

Q1: Quelle est la taille de la table de page?

- Q2: À quelle adresse de la mémoire retrouverons-nous l'instruction du programme à l'adresse virtuelle 0x143AB?

Solution de l'exercice 1

- Q1
  - La taille de la table = nombre d'entrée * taille d'une entrée
  - Le nombre d'entrée dans la table de page sera le nombre de page virtuelle: nombre d'entrée = taille de prog./taille des pages = 16k
  - La taille d'une entrée est le nombre de bits qu'il faut pour d'écrire le numéro de page de mémoire + les 4 bits d'information. Le nombre de bits requis pour décrire le numéro de page de mémoire est log2(nombre de page de mémoire). Donc:
  - Taille d'une entrée = log2(16Mo/4ko) bits + 4 bits = 16 bits
    - R1: La taille de la table = 16k * 2 = 32k
- Q2
  - Pour trouver l'adresse physique, il faut trouver le numéro de page virtuelle et le remplacer par le numéro de page physique selon la table de page
  - Le numéro de page virtuelle est l'adresse virtuelle divisée par la taille d'une page. Le reste de la division est la position dans la page (offset)
  - 0x143AB/0x01000 (4k = 0x01000) = 0x14 reste 0x3AB
  - Selon la table de pages, la page 0x14 du programme est placé à la page 0xB de la mémoire.
  - R2: L'adresse virtuelle 0x143AB se retrouve à l'adresse physique 0xB3AB