Systèmes d'exploitations

Matthieu Lemerre
CEA LIST

Année 2021-2022

Récapitulif: cours précédents et ce cours

Dans les cours précédent, nous avons vu:

- Comment est organisée la mémoire à l'intérieur d'un processus
- Comment on peut créer plusieurs fils d'exécutions indépendants (threads)

Récapitulif: cours précédents et ce cours

Dans les cours précédent, nous avons vu:

- Comment est organisée la mémoire à l'intérieur d'un processus
- Comment on peut créer plusieurs fils d'exécutions indépendants (threads)

Dans ce cours, nous allons voir:

- Comment isoler/confiner/protéger les threads
  - Comment garantir que des fils d'exécutions indépendants restent indépendants
  - Comment passer du multi-thread au multi-processus
- Comment est organisée la mémoire dans un système à plusieurs processus
  - Allocation mémoire efficace entre processus

Récapitulif: cours précédents et ce cours

Dans les cours précédent, nous avons vu:

- Comment est organisée la mémoire à l'intérieur d'un processus
- Comment on peut créer plusieurs fils d'exécutions indépendants (threads)

Dans ce cours, nous allons voir:

- Comment isoler/confiner/protéger les threads
  - Comment garantir que des fils d'exécutions indépendants restent indépendants
  - Comment passer du multi-thread au multi-processus
- Comment est organisée la mémoire dans un système à plusieurs processus
  - Allocation mémoire efficace entre processus

Applications de ce cours:

- Sécurisation de systèmes embarqués
- Comprendre le fonctionnement et le coût des principaux appels systèmes UNIX/Windows (fork,mmap,...)

Sommaire

1. Cours 5: Protection et organisation de la mémoire
   - Le confinement mémoire
   - Mémoire virtuelle et pagination
   - Conclusion

Question

Exemple

Vous êtes un étudiant qui écrit un programme C. Votre maîtrise des pointeurs n'est pas parfaite et votre programme écrit à des endroits aléatoires en mémoire. Que peut-il se passer?

- 1. Vous corrompez la mémoire de votre processus

Question

Exemple

Vous êtes un étudiant qui écrit un programme C. Votre maîtrise des pointeurs n'est pas parfaite et votre programme écrit à des endroits aléatoires en mémoire. Que peut-il se passer?

- 1. Vous corrompez la mémoire de votre processus
- 2. Vous corrompez la mémoire d'un autre processus

Question

Exemple

Vous êtes un étudiant qui écrit un programme C. Votre maîtrise des pointeurs n'est pas parfaite et votre programme écrit à des endroits aléatoires en mémoire. Que peut-il se passer?

- 1. Vous corrompez la mémoire de votre processus
- 2. Vous corrompez la mémoire d'un autre processus
- 3. Tout l'OS plante et il faut redémarrer

Question

Exemple

Vous êtes un étudiant qui écrit un programme C. Votre maîtrise des pointeurs n'est pas parfaite et votre programme écrit à des endroits aléatoires en mémoire. Que peut-il se passer?

- 0. Le processus s'arrête en signalant une erreur
- 1. Vous corrompez la mémoire de votre processus
- 2. ~~Vous corrompez la mémoire d'un autre processus~~
- 3. ~~Tout l'OS plante et il faut redémarrer~~

Définition (*Confinement* et *isolation*)

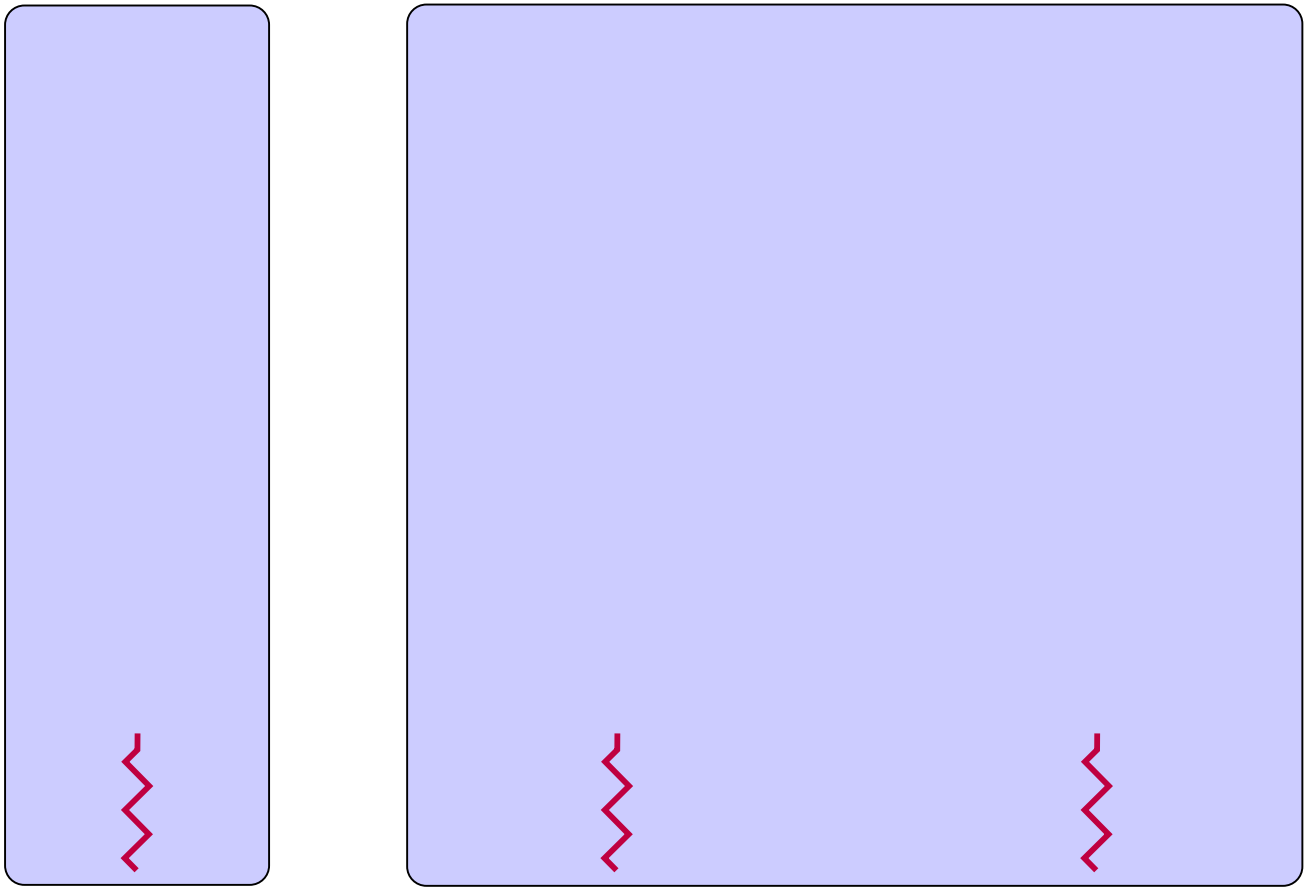
Assurer que l'exécution d'un code ne puisse pas nuire au reste du système.

Définition (Confinement *mémoire*)

Assurer que l'exécution d'un code ne peut pas accéder (lire et écrire) aux zones mémoires du reste du système.

Confinement et espace d'adressage

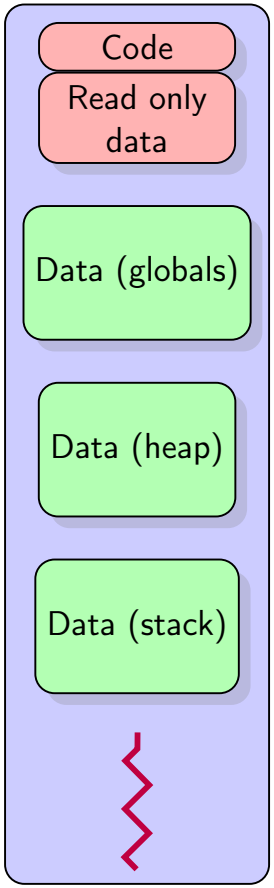
Confinement et espace d'adressage



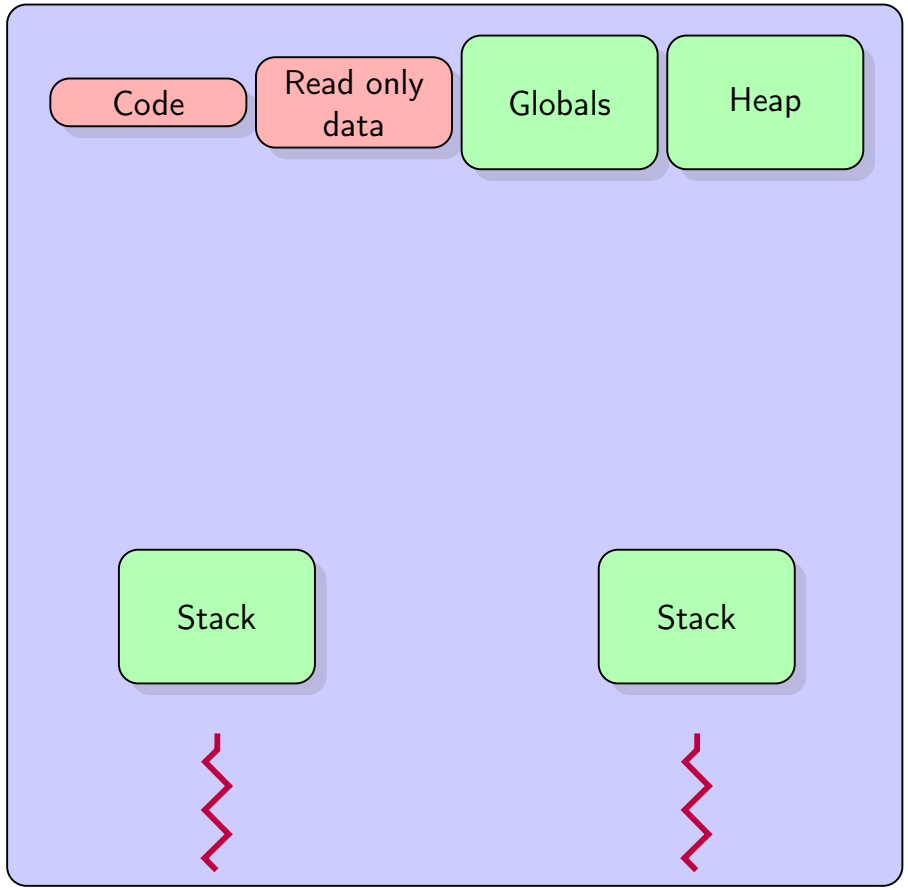
Processus mono-threadé

Processus multi-threadé

Confinement et espace d'adressage

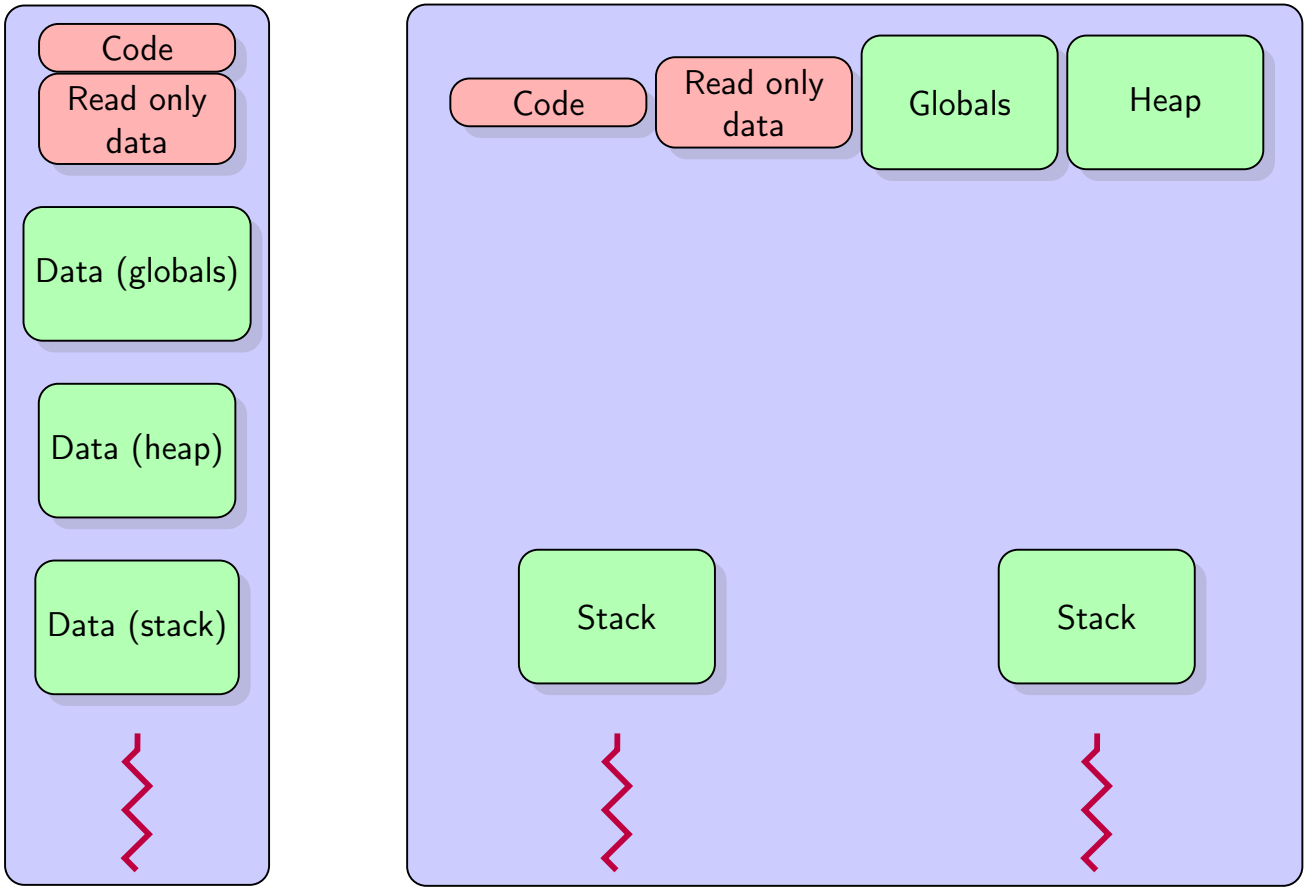


Processus mono-threadé



Processus multi-threadé

Confinement et espace d'adressage



Processus mono-threadé Processus multi-threadé

- Espace d'adressage = ensemble des adresses accessibles par des threads (boites bleues)
- Le confinement mémoire consiste à isoler différents threads d'exécution dans des espaces d'adressages séparés.

Pourquoi confiner?

Pourquoi confiner?

- Améliorer la sûreté/disponibilité
  - Les autres tâches continuent de tourner sans être impactées
  - La détection d'une erreur permet de redémarrer la tâche fautive
- Faciliter la mise au point
  - Détecter les erreurs au plus tôt, au plus près
- Améliorer la sécurité
  - Limiter la prise de contrôle d'un attaquant à un seul composant
  - Empêcher la lecture de secrets (mot de passes, clés cryptos)

Pourquoi confiner?

- Améliorer la sûreté/disponibilité
  - Les autres tâches continuent de tourner sans être impactées
  - La détection d'une erreur permet de redémarrer la tâche fautive
- Faciliter la mise au point
  - Détecter les erreurs au plus tôt, au plus près
- Améliorer la sécurité
  - Limiter la prise de contrôle d'un attaquant à un seul composant
  - Empêcher la lecture de secrets (mot de passes, clés cryptos)

- Limite: pas de protection à l'intérieur de l'enceinte de confinement

Comment confiner?

Comment confiner?

- Séparation physique
  - Les programmes s'exécutent sur des machines séparées.
- Solution logicielle
  - Le logiciel garanti intrinsèquement ne jamais faire d'erreur mémoire
- Solution matérielle
  - Mécanisme matériel pour empêcher l'accès à la mémoire des autres tâches/des autres machines virtuelles.

Confinement par séparation physique

S'assurer de l'absence d'interférence entre programmes en assurant une séparation physique:

- Machines séparées
- Processeurs et mémoires séparées sur une même carte
- Coeurs d'exécution séparés sur une même puce

– Coûte cher (plusieurs unités de calcul)

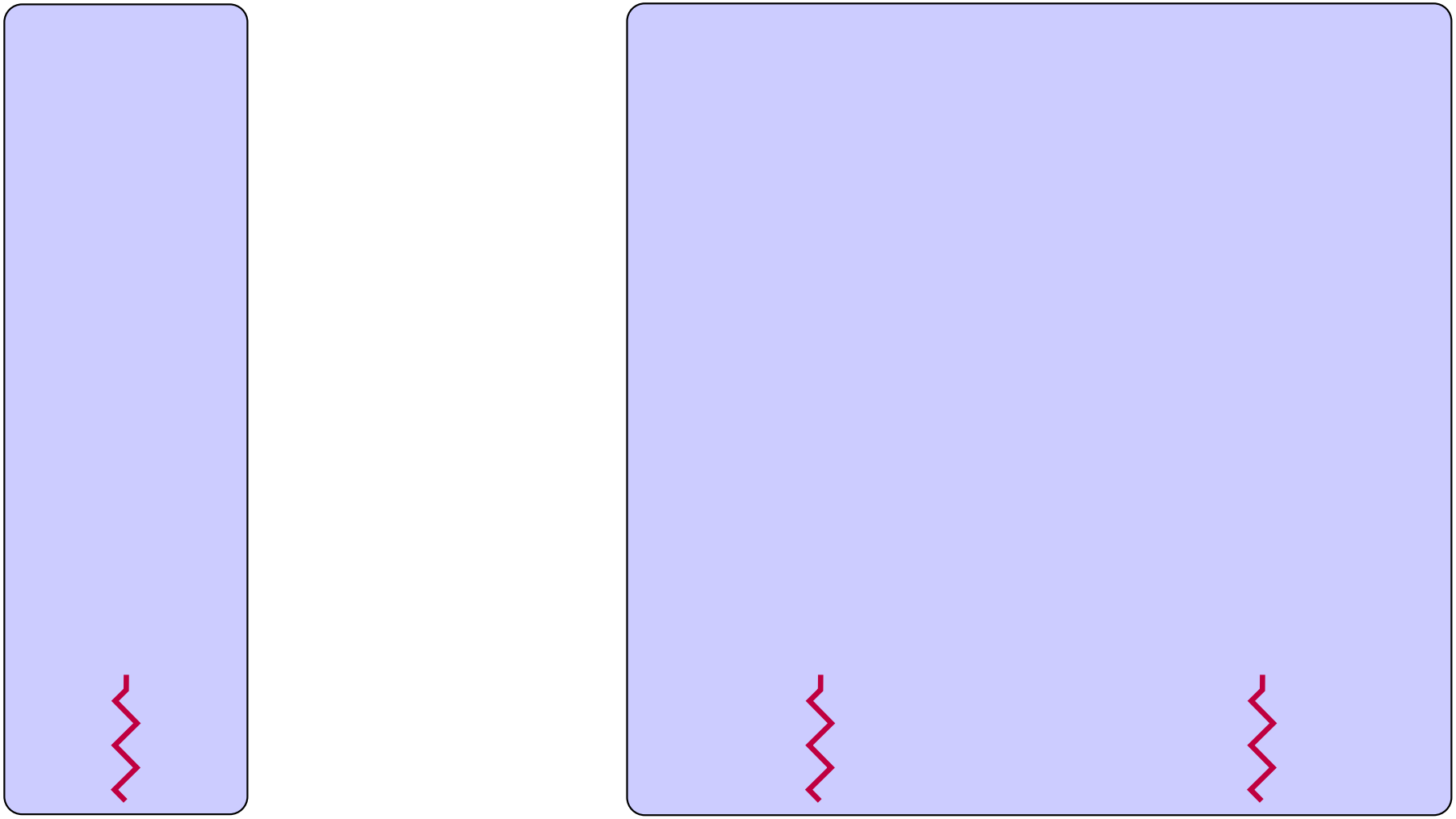
– Pas mémoire partagée entre tâches

\+ Plus haut niveau de confinement possible

⚠ N'empêche pas toute attaque:

- Réseau
- Clé USB
- Microphone...

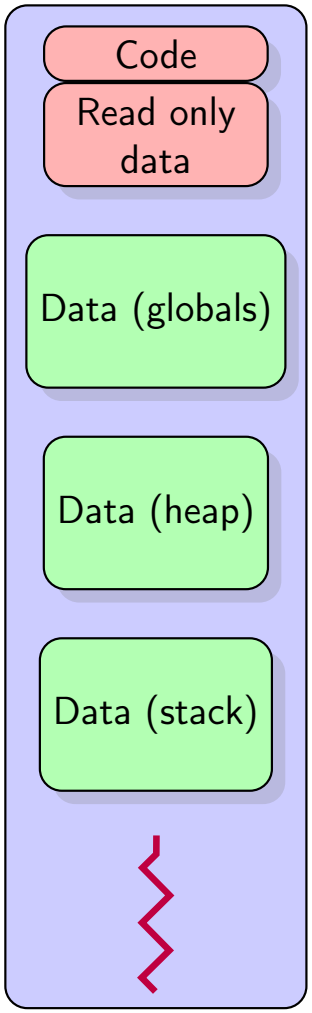
Confinement mémoire au sein d'une seule machine



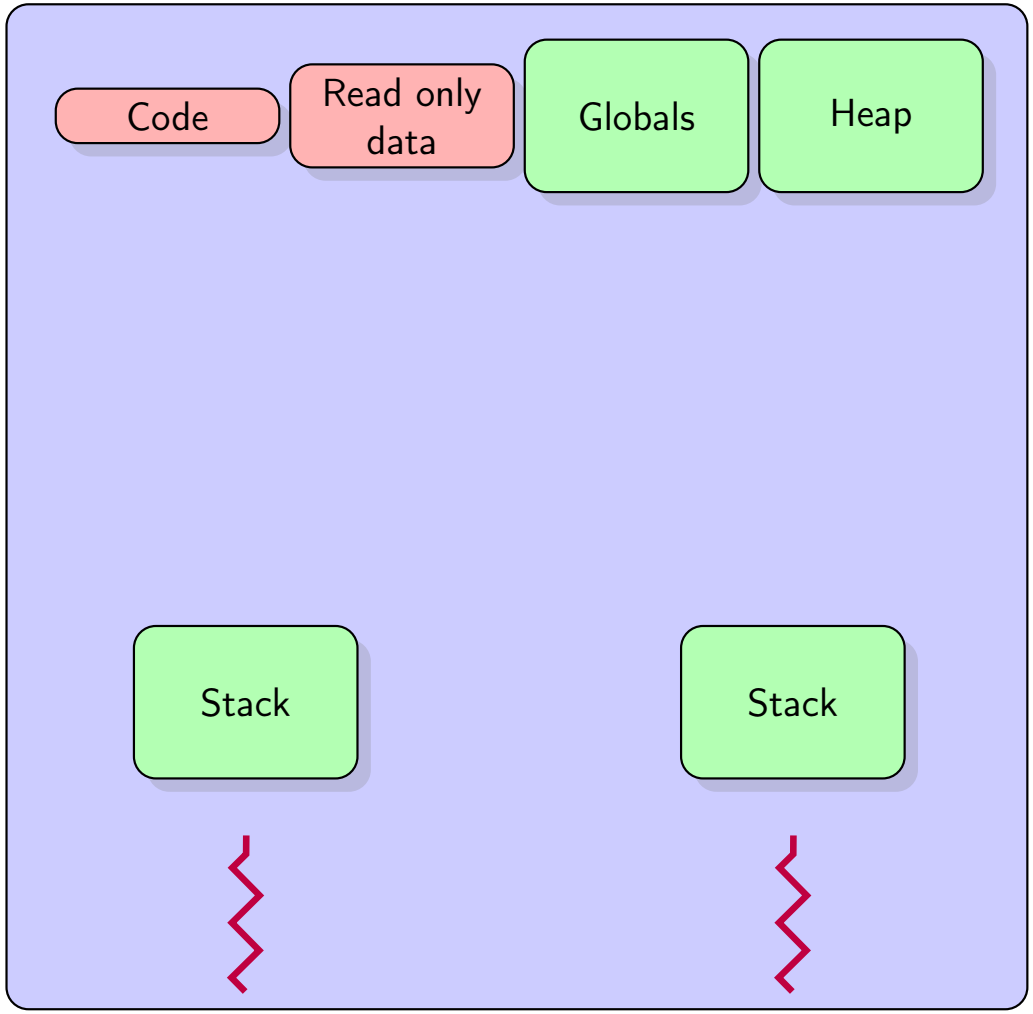
Process A
(mono-thread)

Process B
(multi-thread)

Confinement mémoire au sein d'une seule machine

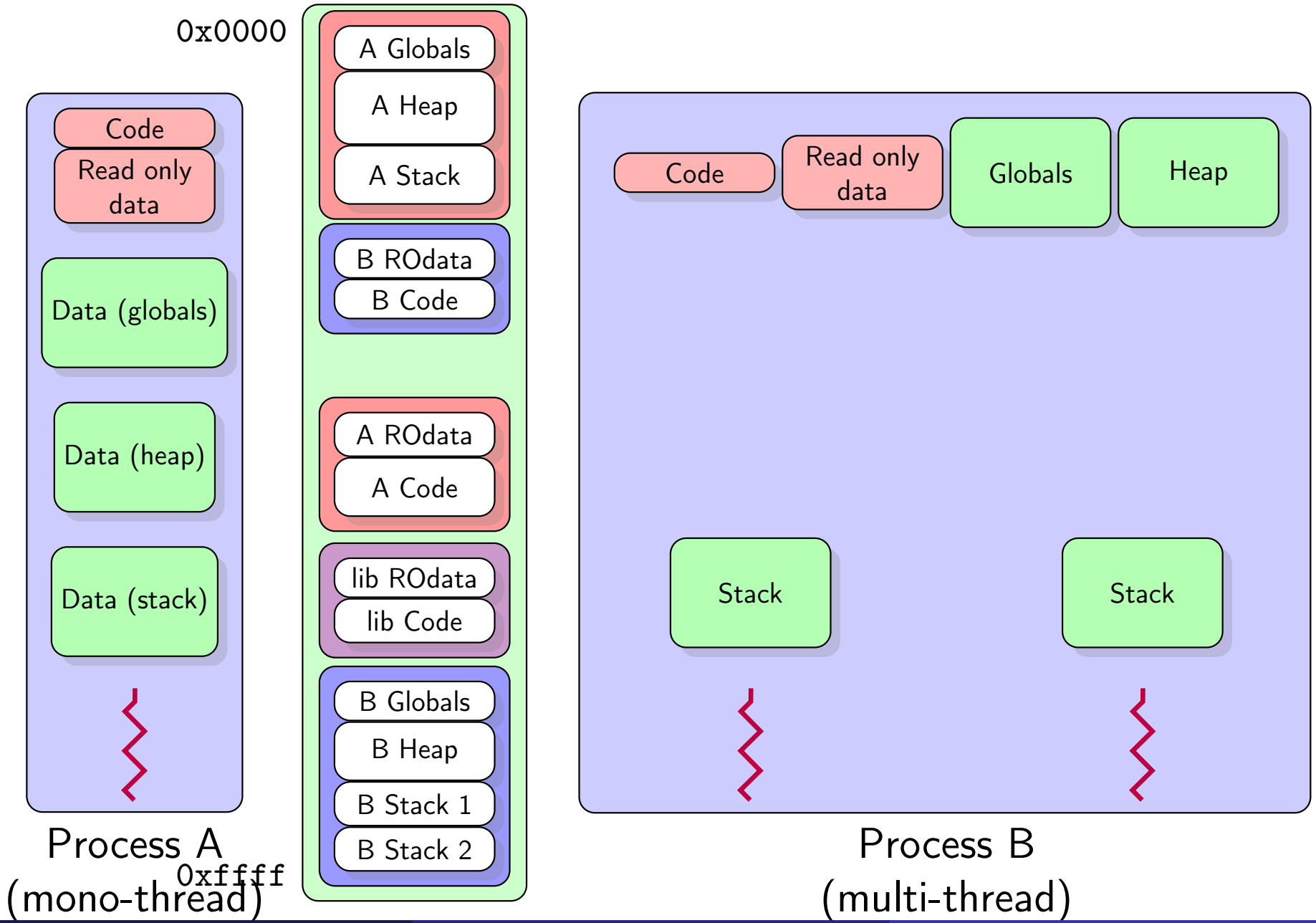


Process A
(mono-thread)

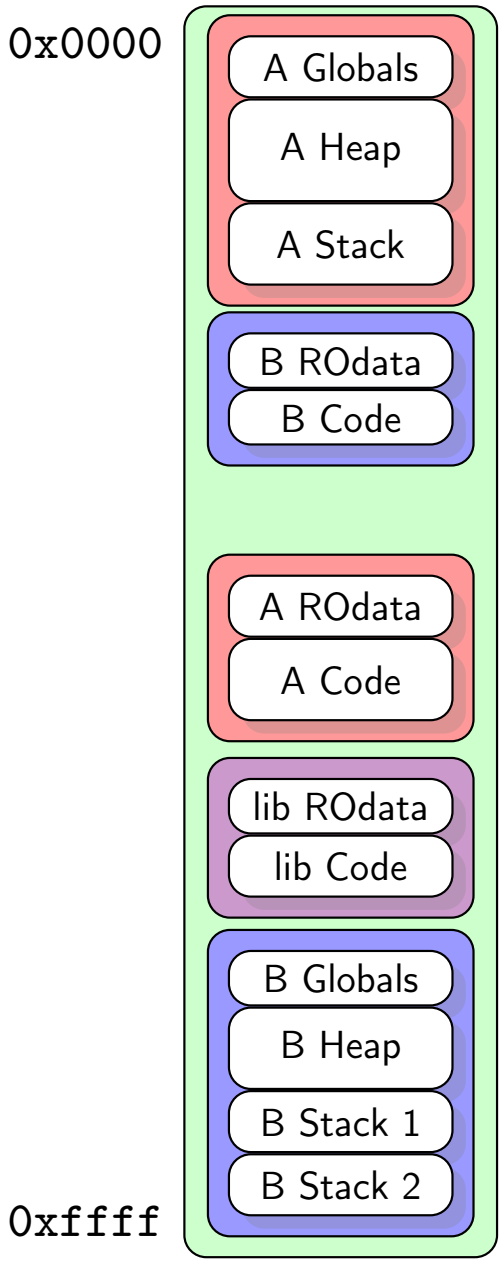


Process B
(multi-thread)

Confinement mémoire au sein d'une seule machine



Confinement mémoire au sein d'une seule machine



Confinement par logiciel

S'assurer, sans support matériel, que le programme exécuté ne vas pas accéder à des zones en dehors de son espace d'adressage.

- Techniques dynamiques (vérification pendant l'exécution):
  - Interprétation (Python, Bash, Valgrind) ou just-in-time compilation (Java, C#)
  - Instrumentation (E-ACSL,KCC)

Confinement par logiciel

S'assurer, sans support matériel, que le programme exécuté ne vas pas accéder à des zones en dehors de son espace d'adressage.

- Techniques dynamiques (vérification pendant l'exécution):
  - Interprétation (Python, Bash, Valgrind) ou just-in-time compilation (Java, C#)
  - Instrumentation (E-ACSL,KCC)
- Techniques statiques (vérification avant ou à la compilation)
  - Langages type-safe (ADA,Ocaml,Rust,Java,C#)
  - Vérification et preuve de programme (Frama-C, Astrée, Polyspace)

Confinement par logiciel

S'assurer, sans support matériel, que le programme exécuté ne vas pas accéder à des zones en dehors de son espace d'adressage.

- Techniques dynamiques (vérification pendant l'exécution):
  - Interprétation (Python, Bash, Valgrind) ou just-in-time compilation (Java, C#)
  - Instrumentation (E-ACSL,KCC)
- Techniques statiques (vérification avant ou à la compilation)
  - Langages type-safe (ADA,Ocaml,Rust,Java,C#)
  - Vérification et preuve de programme (Frama-C, Astrée, Polyspace)

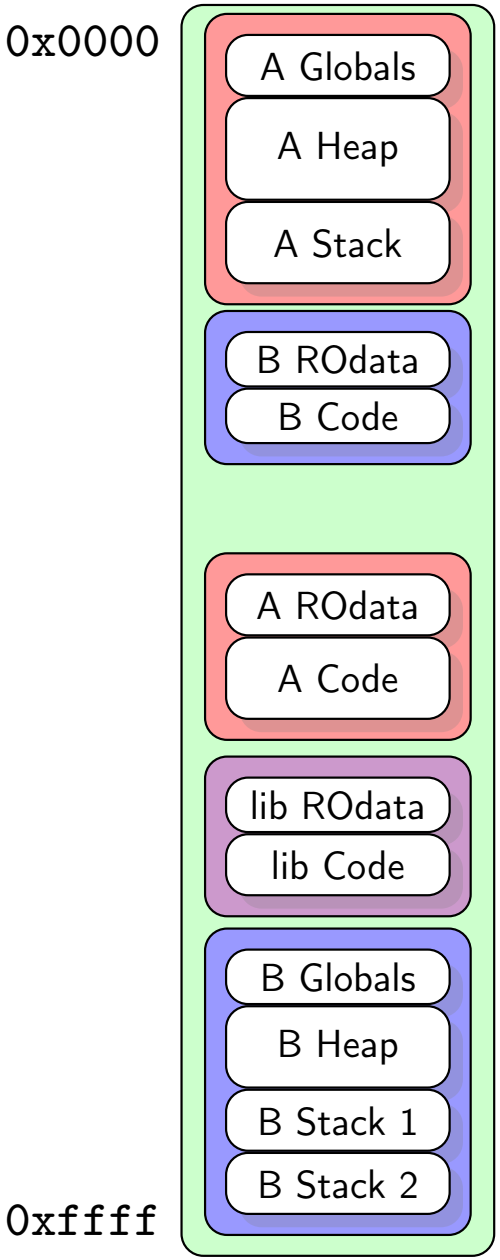
\+ Ne demande pas de modification au processeur
  - Mais tous en sont équipés, sauf les micro-controlleurs

\−
  - Surcoût à l'exécution et/ou
  - Effort de re-développement ou vérification important et/ou
  - Ne fonctionne pas sur du code legacy

\− Ne prémuni pas contre les erreurs physiques (bitflip)

\− Quelle garantie que la technique n'a pas de faille et est bien appliquée?

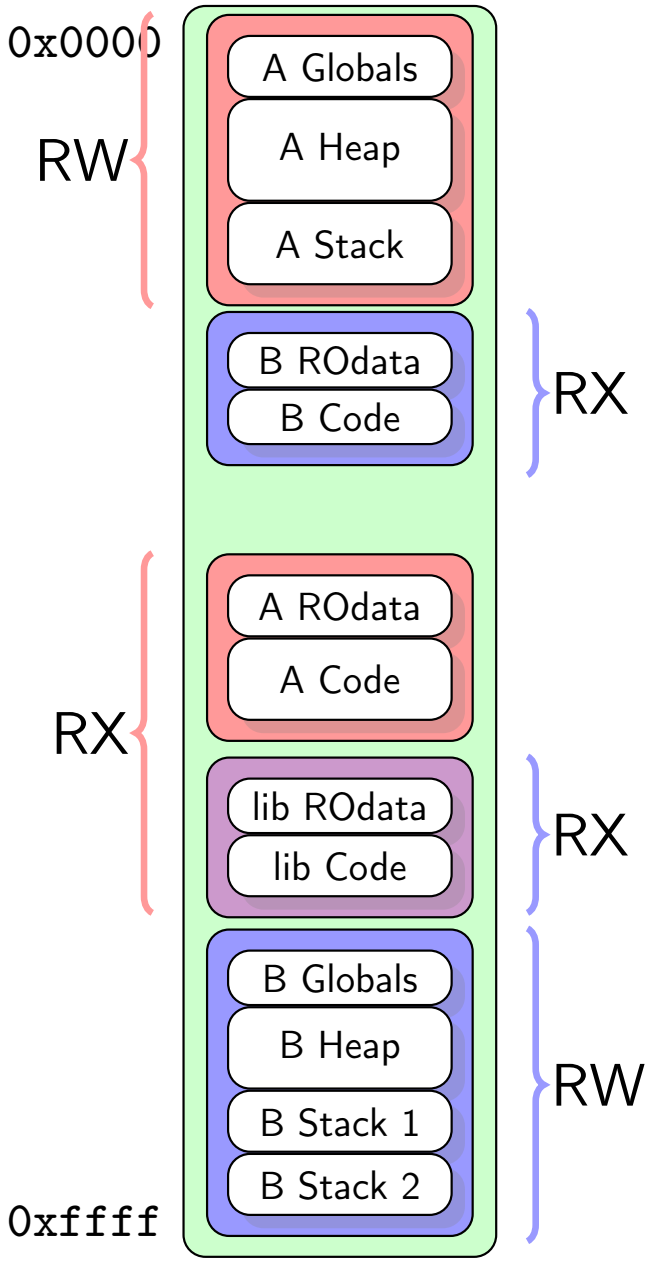
Confinement par matériel: définition d'espaces d'adressages



Définition (Confinement matériel)

- Mécanisme matériel s'assurant que l'exécution d'un thread ne puisse accéder qu'à un ensemble d'adresses pré-enregistré.
- Cet ensemble d'adresse s'appelle un *espace d'adressage*.
- Un *processus* est définit par un thread et un espace d'adressage.

Confinement par matériel: définition d'espaces d'adressages



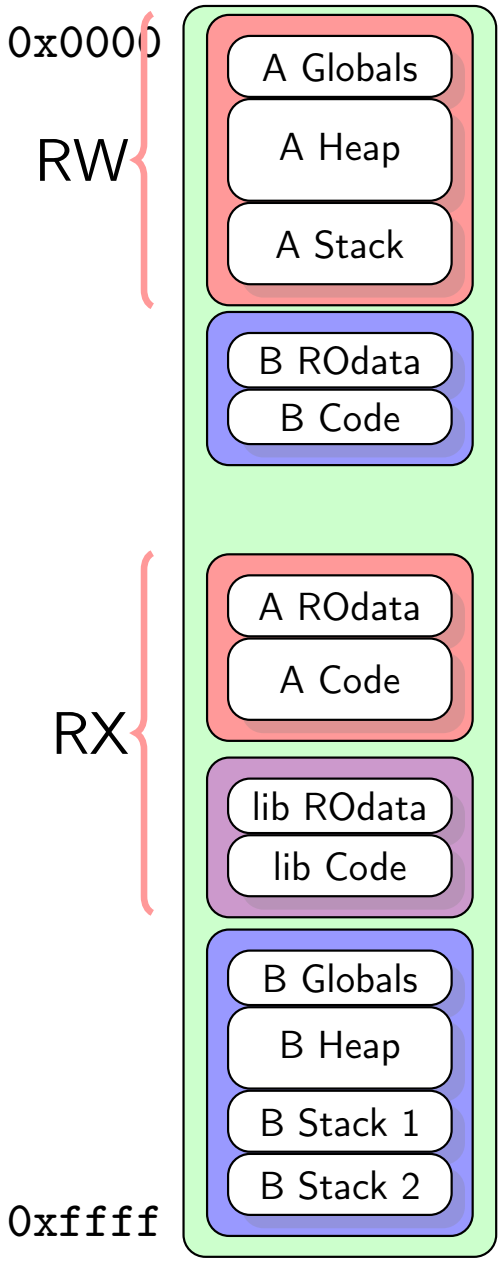
Définition (Confinement matériel)

- Mécanisme matériel s'assurant que l'exécution d'un thread ne puisse accéder qu'à un ensemble d'adresses pré-enregistré.
- Cet ensemble d'adresse s'appelle un *espace d'adressage*.
- Un *processus* est définit par un thread et un espace d'adressage.

Exemple (Deux processus isolés par MPU)

- A peut accéder à deux segments
- B peut accéder à trois segments

Confinement par matériel: définition d'espaces d'adressages



Définition (Confinement matériel)

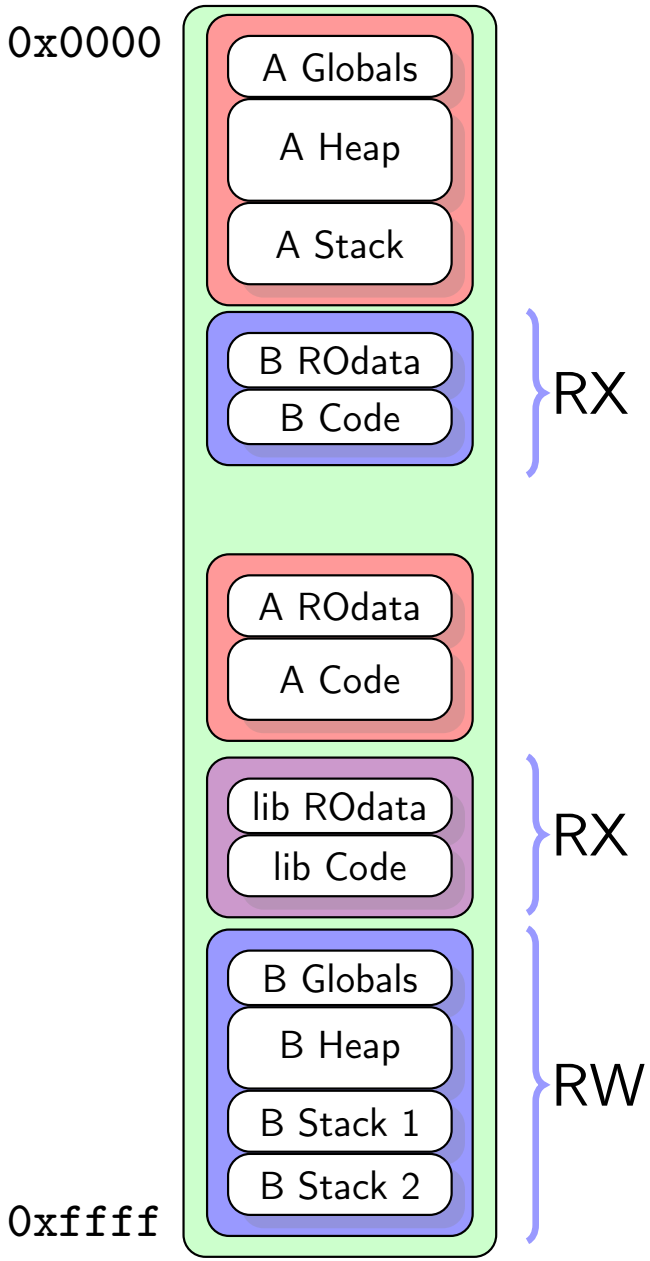
- Mécanisme matériel s'assurant que l'exécution d'un thread ne puisse accéder qu'à un ensemble d'adresses pré-enregistré.
- Cet ensemble d'adresse s'appelle un *espace d'adressage*.
- Un *processus* est définit par un thread et un espace d'adressage.

Exemple (Deux processus isolés par MPU)

- A peut accéder à deux segments
- B peut accéder à trois segments

Adresses accessibles quand A s'exécute

Confinement par matériel: définition d'espaces d'adressages



Définition (Confinement matériel)

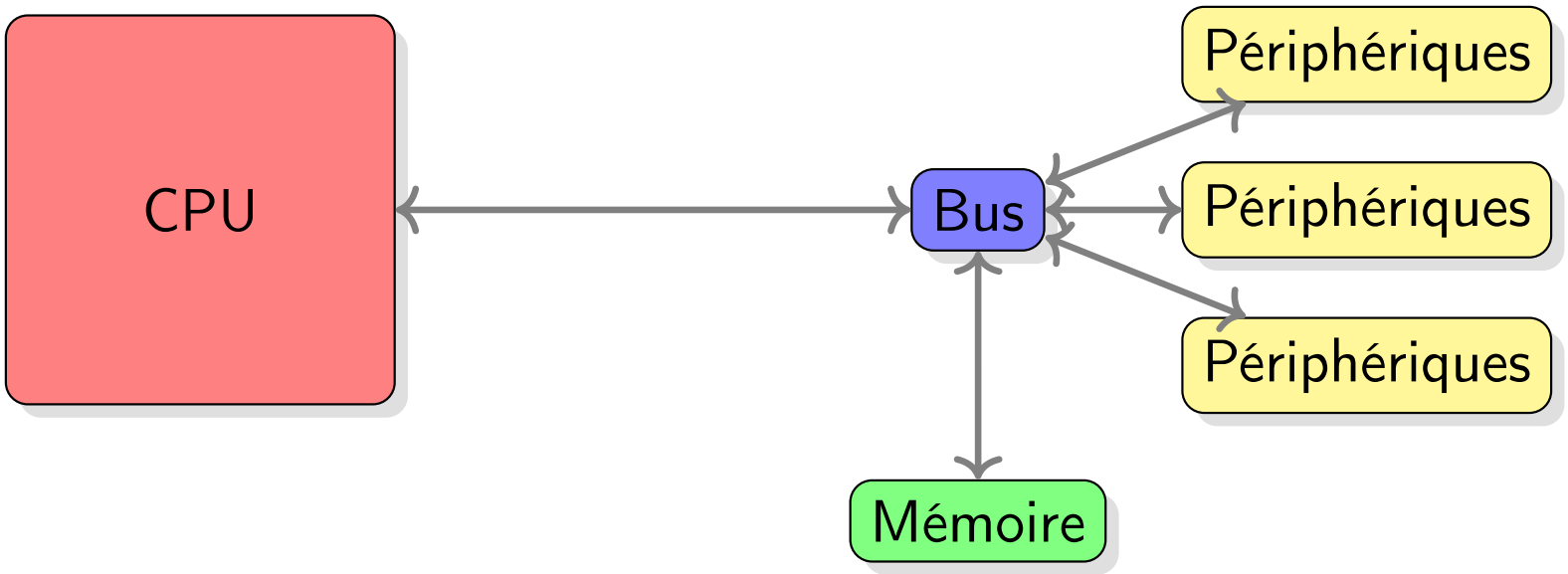
- Mécanisme matériel s'assurant que l'exécution d'un thread ne puisse accéder qu'à un ensemble d'adresses pré-enregistré.
- Cet ensemble d'adresse s'appelle un *espace d'adressage*.
- Un *processus* est définit par un thread et un espace d'adressage.

Exemple (Deux processus isolés par MPU)

- A peut accéder à deux segments
- B peut accéder à trois segments

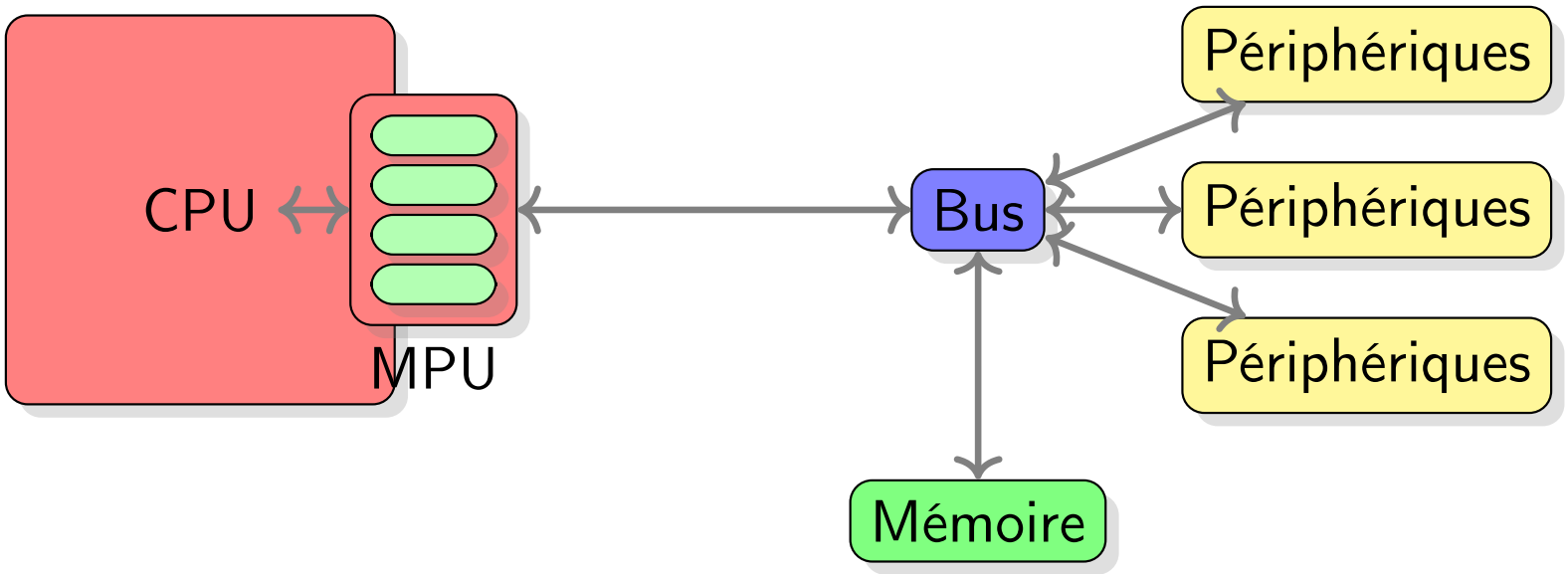
Adresses accessibles quand B s'exécute

Confinement par matériel: fonctionnement d'une MPU



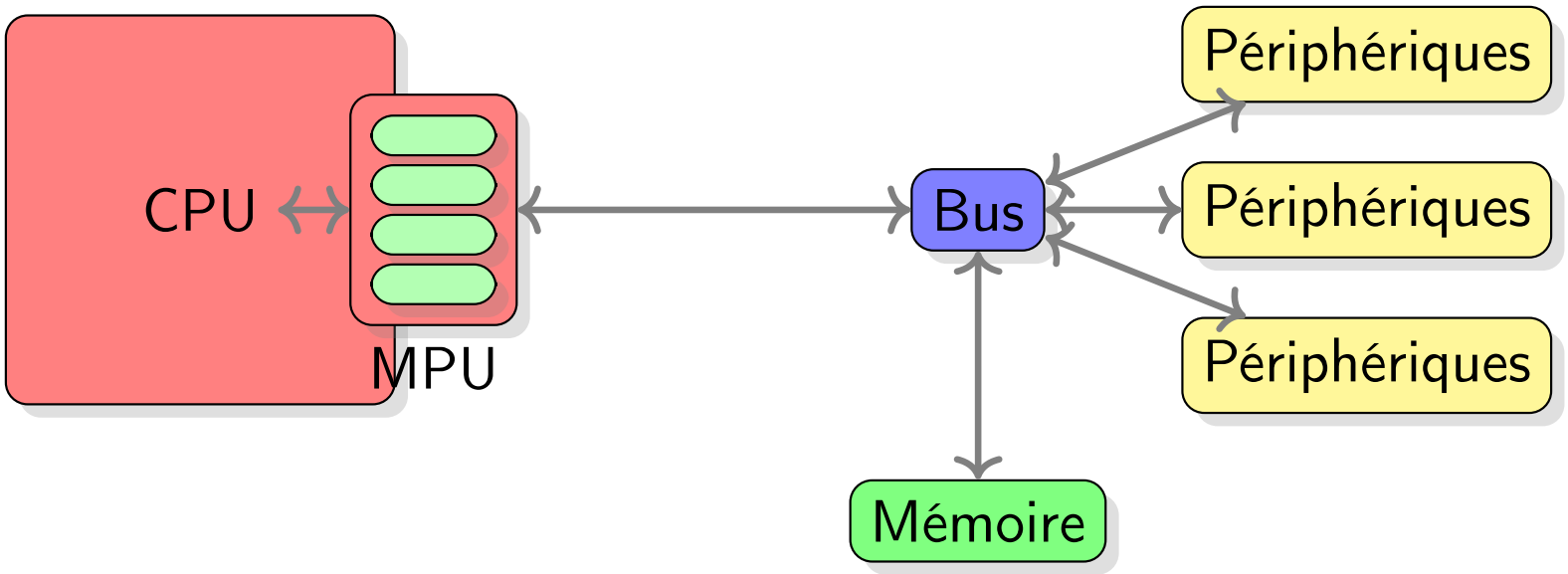
Comment le matériel vérifie si les adresses accédées sont dans l'espace d'adressage autorisé?

Confinement par matériel: fonctionnement d'une MPU



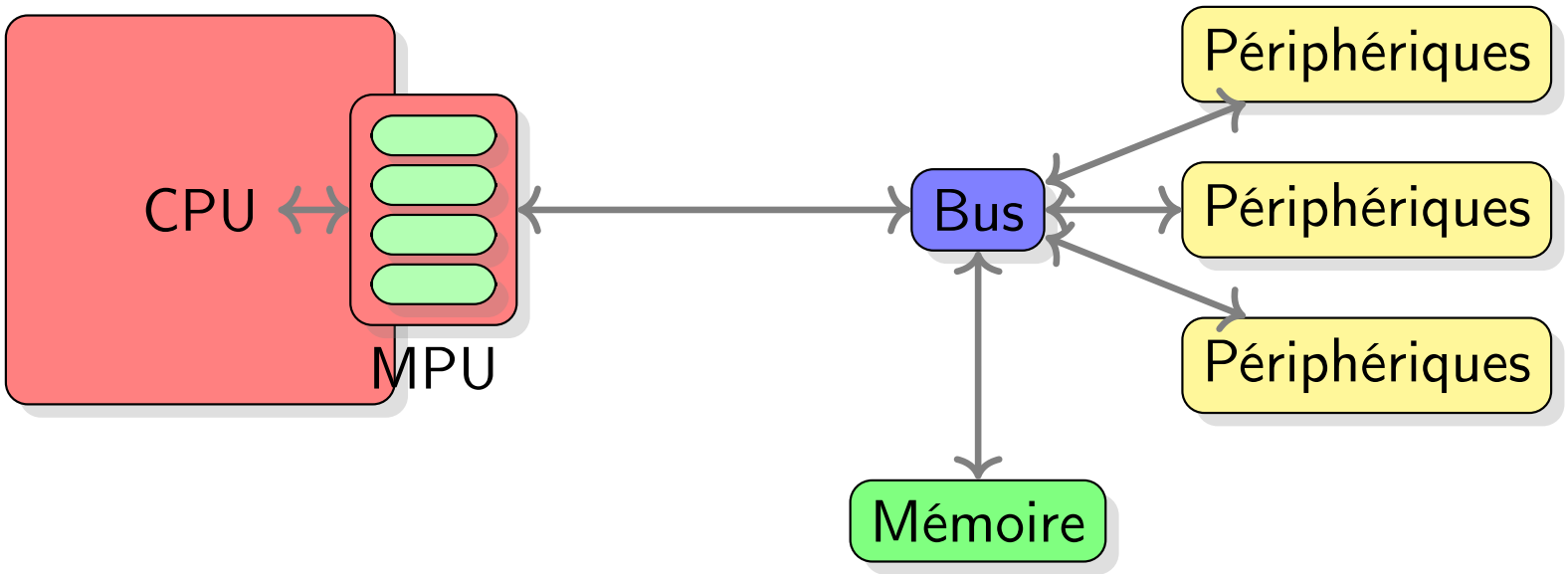
- Memory Protection Unit:
  - S'intercalle entre le processeur et le bus
  - Est paramétrée par un nombre fixe de registres spéciaux:
    - Adresse de début, Adresse de fin, Permission
  - Tout accès mémoire en dehors de ces plages provoque une interruption.

Confinement par matériel: fonctionnement d'une MPU



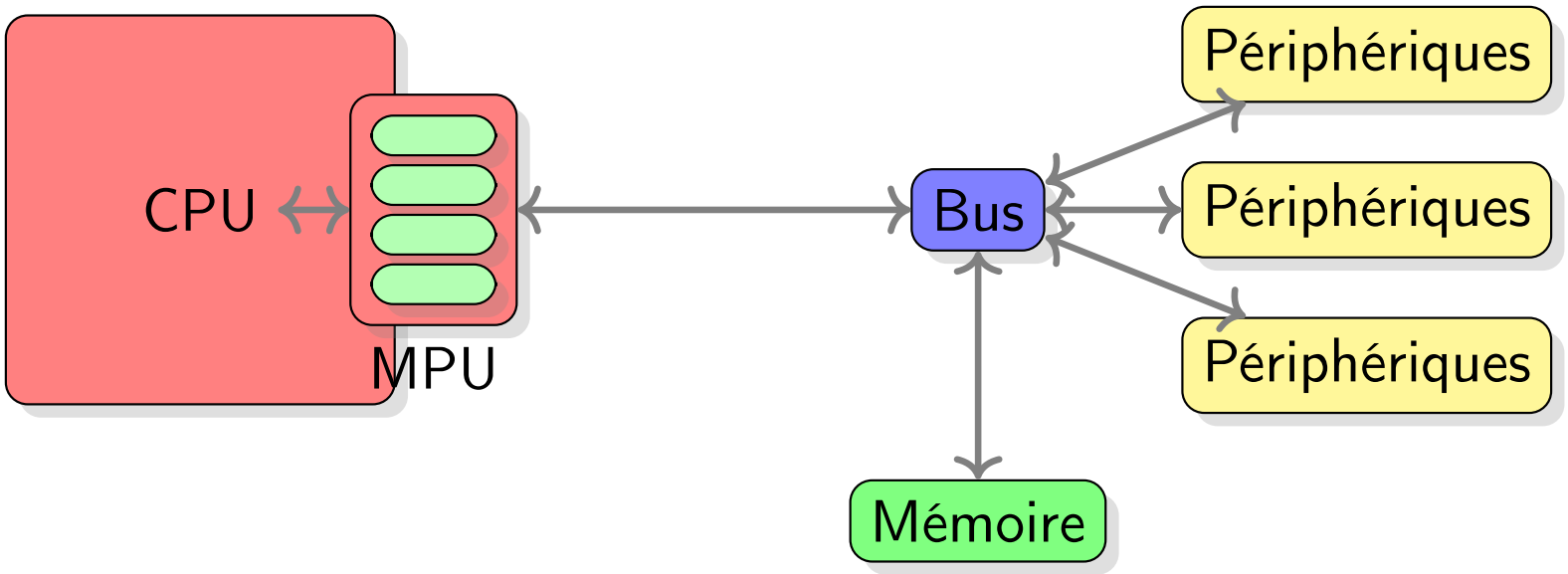
- Memory Protection Unit:
  - S'intercalle entre le processeur et le bus
  - Est paramétrée par un nombre fixe de registres spéciaux:
    - Adresse de début, Adresse de fin, Permission
  - Tout accès mémoire en dehors de ces plages provoque une interruption.
- Il faut modifier le paramétrage de la MPU quand on change de thread.

Confinement par matériel: fonctionnement d'une MPU



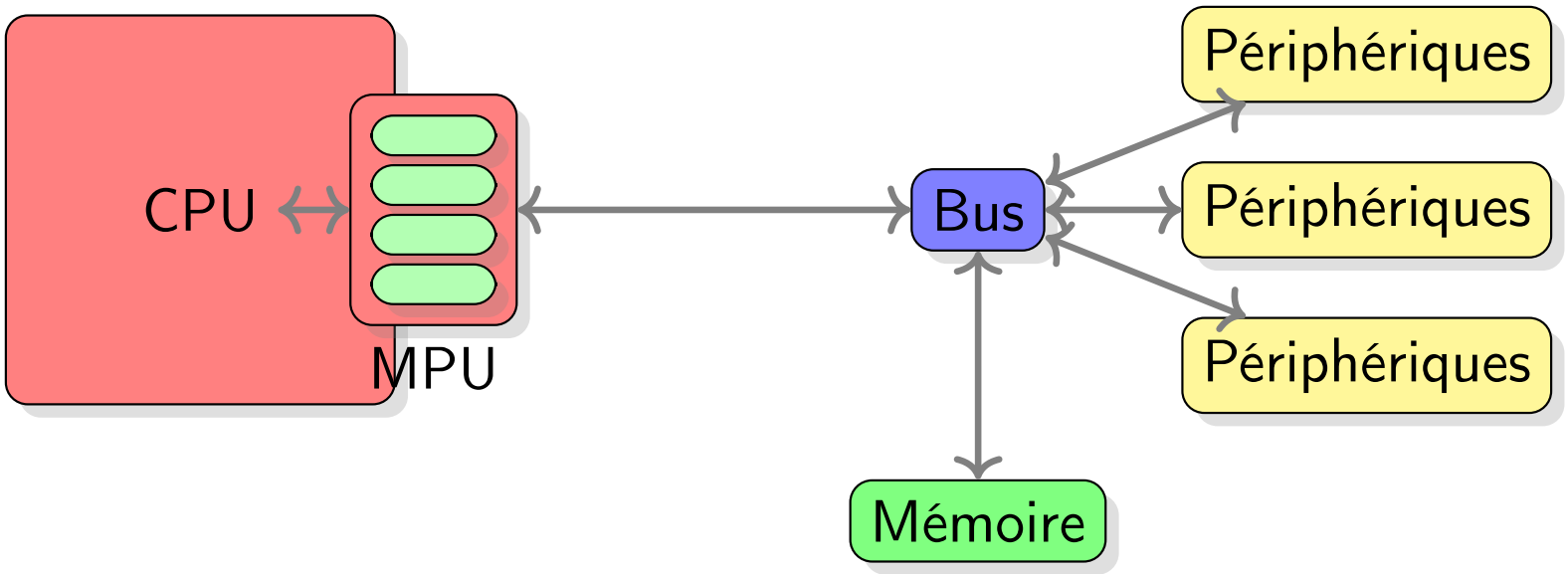
- Comment un process peut sortir du confinement?

Confinement par matériel: fonctionnement d'une MPU



- Comment un process peut sortir du confinement?
- Il lui suffit de modifier la configuration de la MPU!

Confinement par matériel: fonctionnement d'une MPU



- Comment un process peut sortir du confinement?
- Il lui suffit de modifier la configuration de la MPU!
- ➜ L'écriture des registres MPU doit être reservé à du code privilégié (de confiance)
- Ce code, comprenant le changement de thread et le changement d'espace d'adressage, s'appelle le noyau.

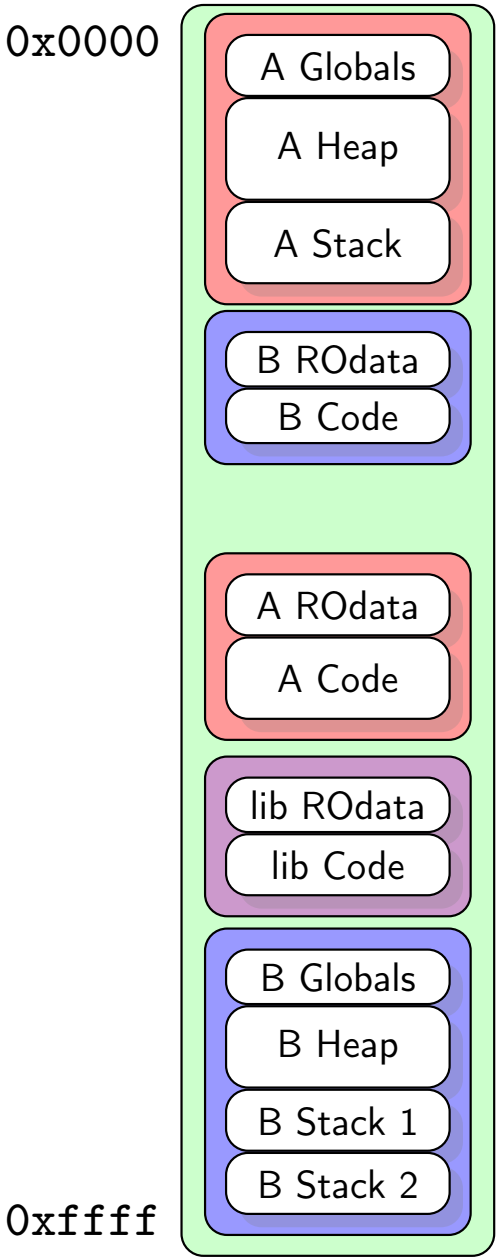
Sommaire

1. Cours 5: Protection et organisation de la mémoire
   - Le confinement mémoire
   - Mémoire virtuelle et pagination
   - Conclusion

Plan

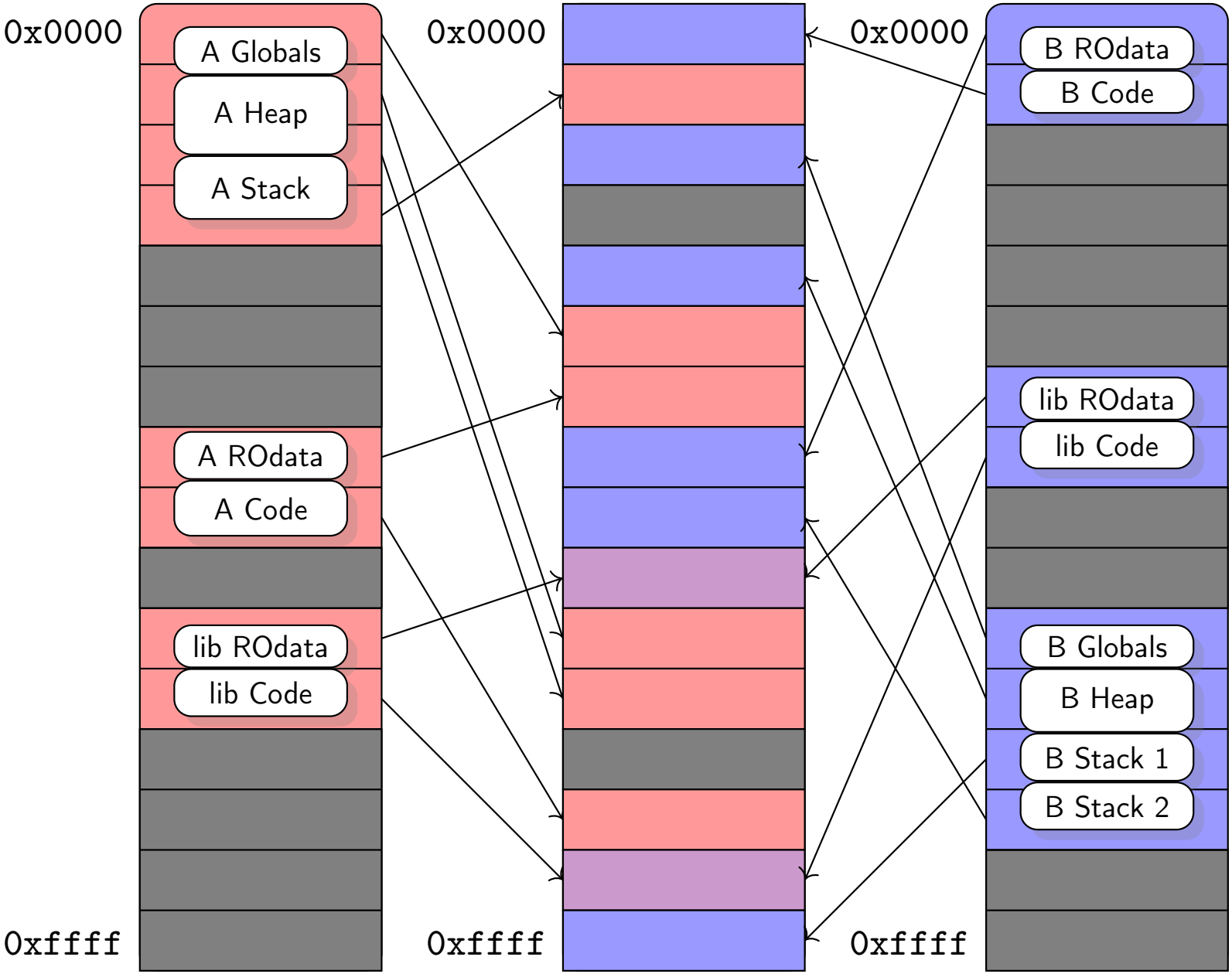
1. Cours 5: Protection et organisation de la mémoire
   - Le confinement mémoire
   - Mémoire virtuelle et pagination
   - Conclusion

Fragmentation externe

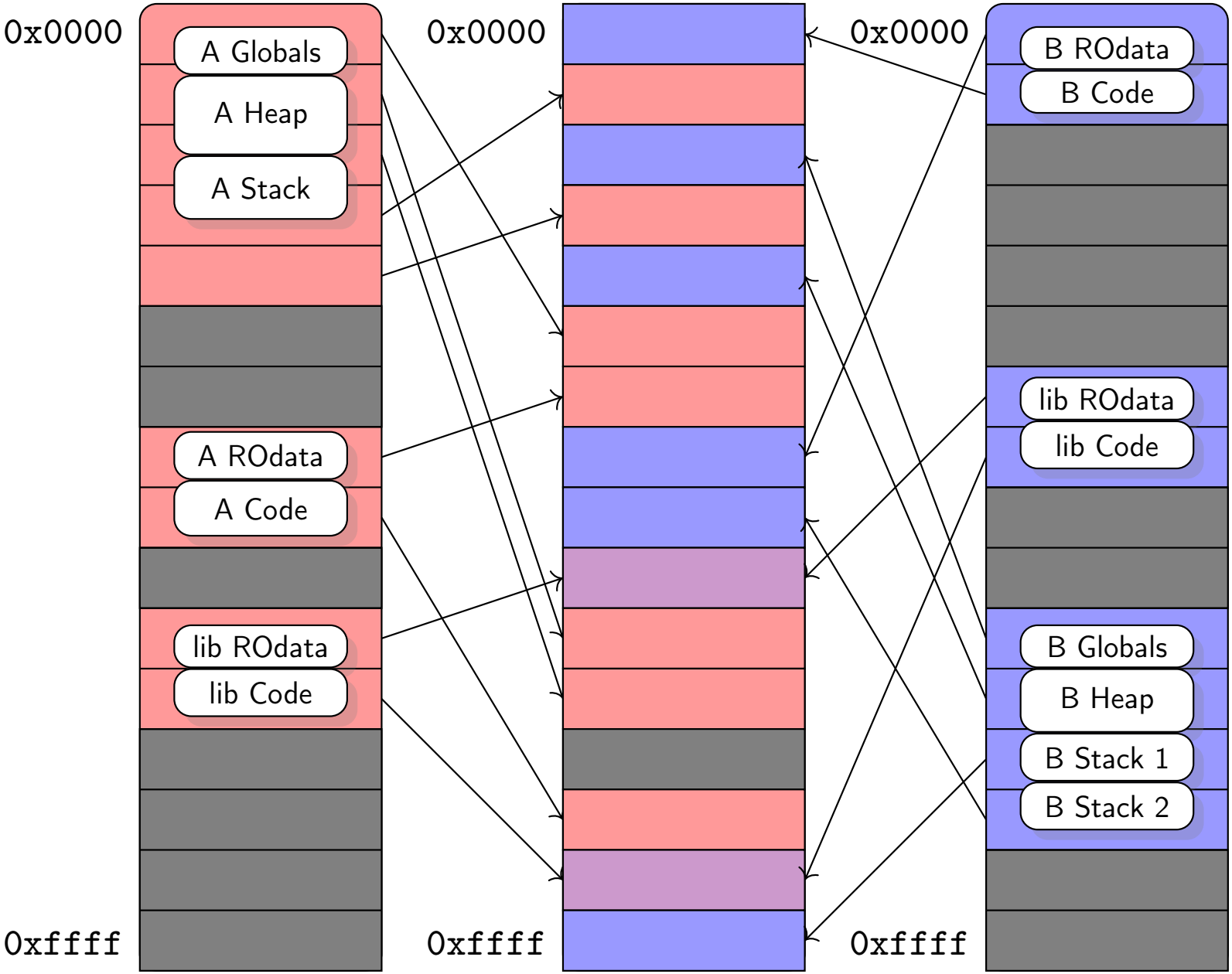


- *A* souhaite faire grossir sa pile, mais ne peut pas:
  - Il y a de la mémoire disponible mais elle est mal placée (*fragmentation*).
- En général: l'organisation mémoire d'un processus dépend de celle des autres.
  - Peut convenir pour un système embarqué (un seul logiciel fixé pour l'ordinateur)
  - Gênant pour serveurs, ordinateurs de bureaux (exécution de programmes pas connus à l'avance).

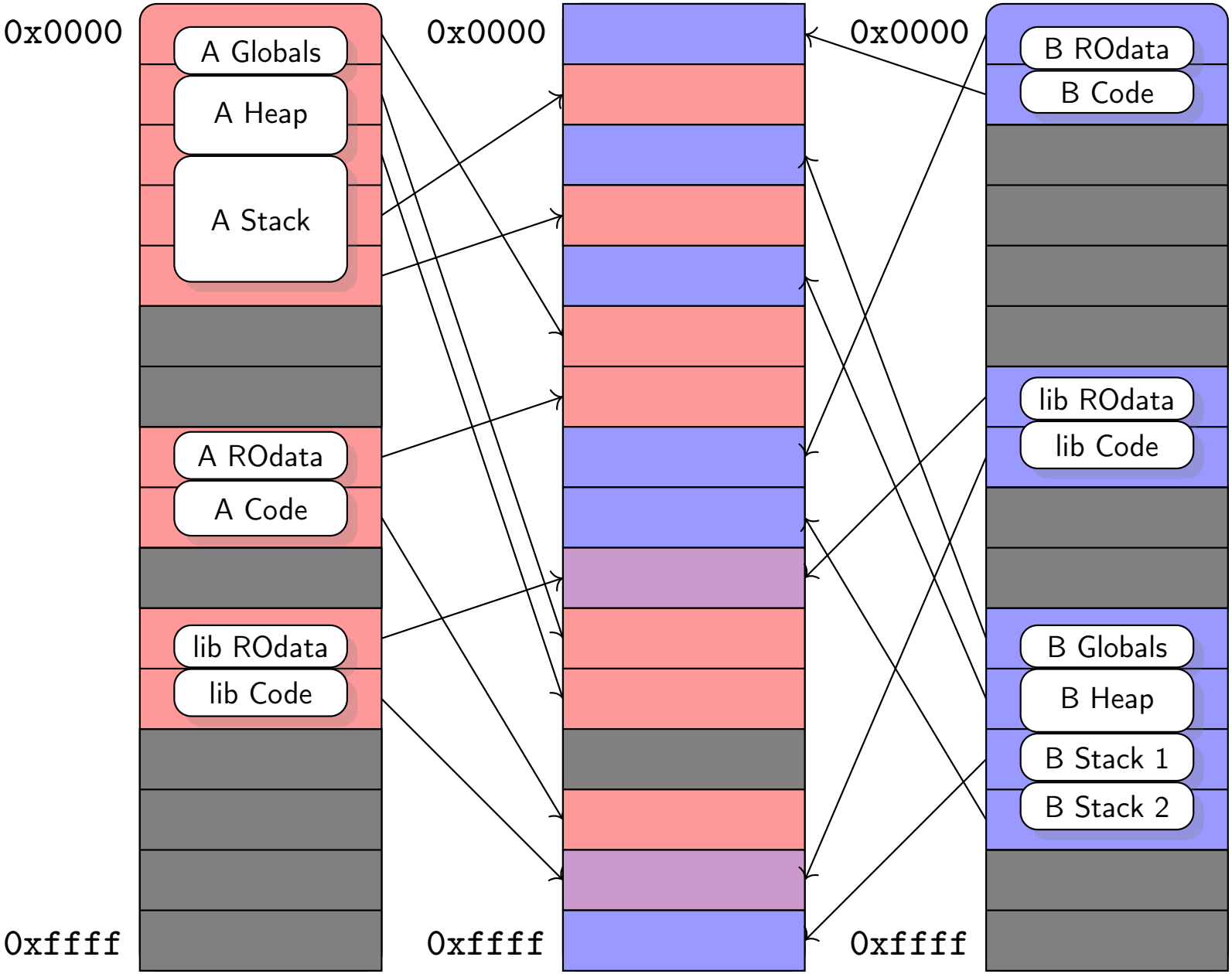
Pagination et mémoire virtuelle



Pagination et mémoire virtuelle

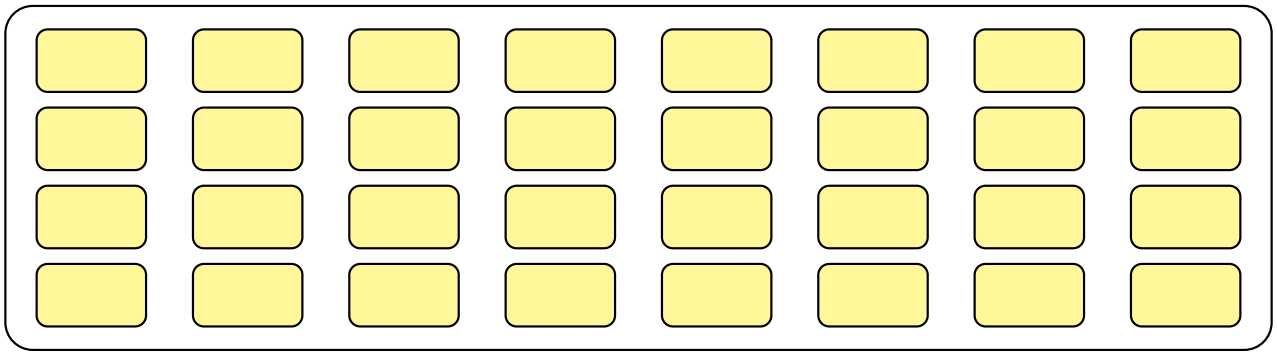


Pagination et mémoire virtuelle



Fonctionnement de la pagination: division en pages

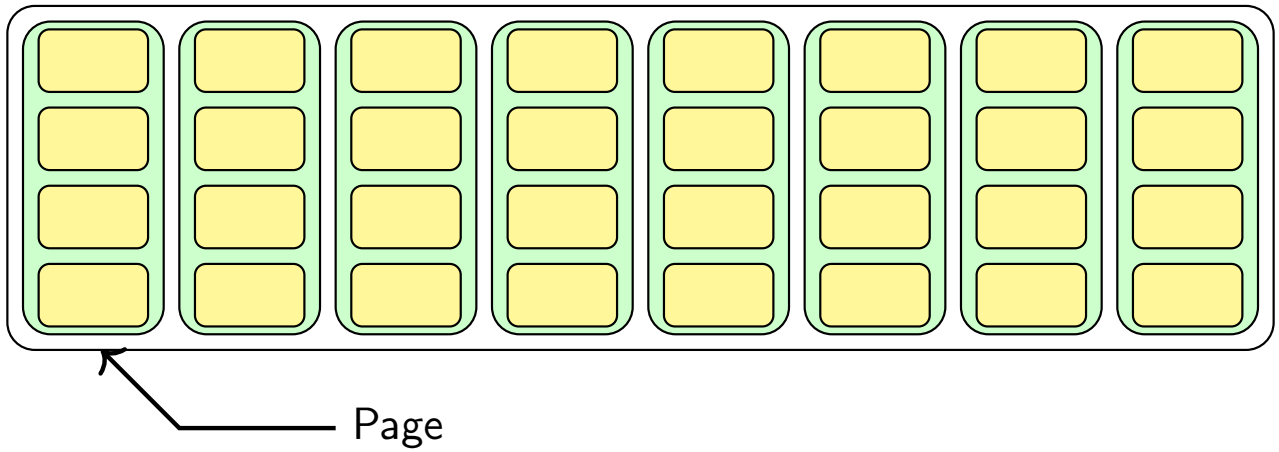
1. Division de la mémoire en zones de taille fixe: les pages



- Grande taille de pages: perte de mémoire quand toute la page n'est pas utilisée (fragmentation interne)
- Petite taille de page: plus de pages temps et mémoire nécessaire pour les gérer plus importante

Fonctionnement de la pagination: division en pages

1. Division de la mémoire en zones de taille fixe: les pages

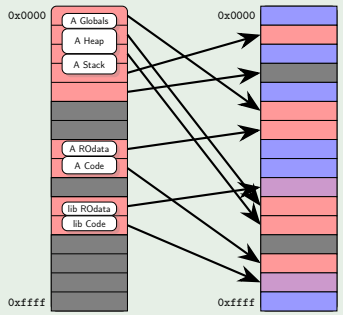


- Grande taille de pages: perte de mémoire quand toute la page n'est pas utilisée (fragmentation interne)
- Petite taille de page: plus de pages temps et mémoire nécessaire pour les gérer plus importante
- Compromis le plus couramment utilisé: 4ko

Fonctionnement de la pagination: translation d'adresse

Exemple (Translation de `load 0x12345678` avec pages de taille 4ko)

- 0x12345678 = 0x12345 * 4096 + 0x678
- Adresse virtuelle:
  - PT[0x12345] * 4096 + 0x678
  - 0xabcdef678



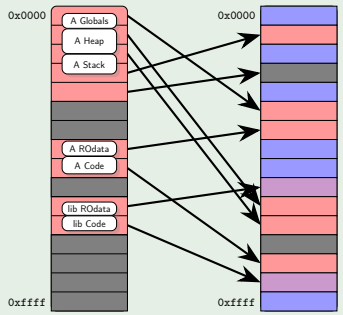
Translation, par la ▸ MMU, des *adresses virtuelles* en *adresses physiques*

- Décomposition des adresses virtuelles en un *numéro de page* (bits de poids fort) et un *déplacement* (bits de poids faible)
- Utilisation d'un tableau associatif PT pour faire correspondre numéro de page à la page physique
- Recombinaison pour obtenir l'adresse physique.

Fonctionnement de la pagination: translation d'adresse

Exemple (Translation de `load 0x12345678` avec pages de taille 4ko)

- 0x12345678 = 0x12345 * 4096 + 0x678
- Adresse virtuelle:
  - PT[0x12345] * 4096 + 0x678
  - 0xabcdef678



Translation, par la ▸ MMU, des *adresses virtuelles* en *adresses physiques*

- Décomposition des adresses virtuelles en un *numéro de page* (bits de poids fort) et un *déplacement* (bits de poids faible)
- Utilisation d'un tableau associatif PT pour faire correspondre numéro de page à la page physique
- Recombinaison pour obtenir l'adresse physique.

Notons:

- Il y a un tableau associatif différent par processus
- Ce tableau définit l'espace d'adressage du processus
- Ce tableau associatif s'appelle la *table des pages* (page table)
- La translation est accélérée par l'usage d'un cache: le TLB

Le défaut de page

Que faire si la translation d'adresse échoue :

- Il n'y a pas de page correspondant à une adresse virtuelle dans la table des pages (page non *mappée*)
- Ou il y a une page mais accessible avec des droits différents (e.g. pas de droit d'écriture ou d'exécution)

Le défaut de page

Que faire si la translation d'adresse échoue :

- Il n'y a pas de page correspondant à une adresse virtuelle dans la table des pages (page non *mappée*)
- Ou il y a une page mais accessible avec des droits différents (e.g. pas de droit d'écriture ou d'exécution)

Une interruption est générée: c'est le *défaut de page*

- Prévient le noyau d'une erreur de chargement à une adresse

Comment le noyau doit traiter le défaut de page?

Le défaut de page

Que faire si la translation d'adresse échoue :

- Il n'y a pas de page correspondant à une adresse virtuelle dans la table des pages (page non *mappée*)
- Ou il y a une page mais accessible avec des droits différents (e.g. pas de droit d'écriture ou d'exécution)

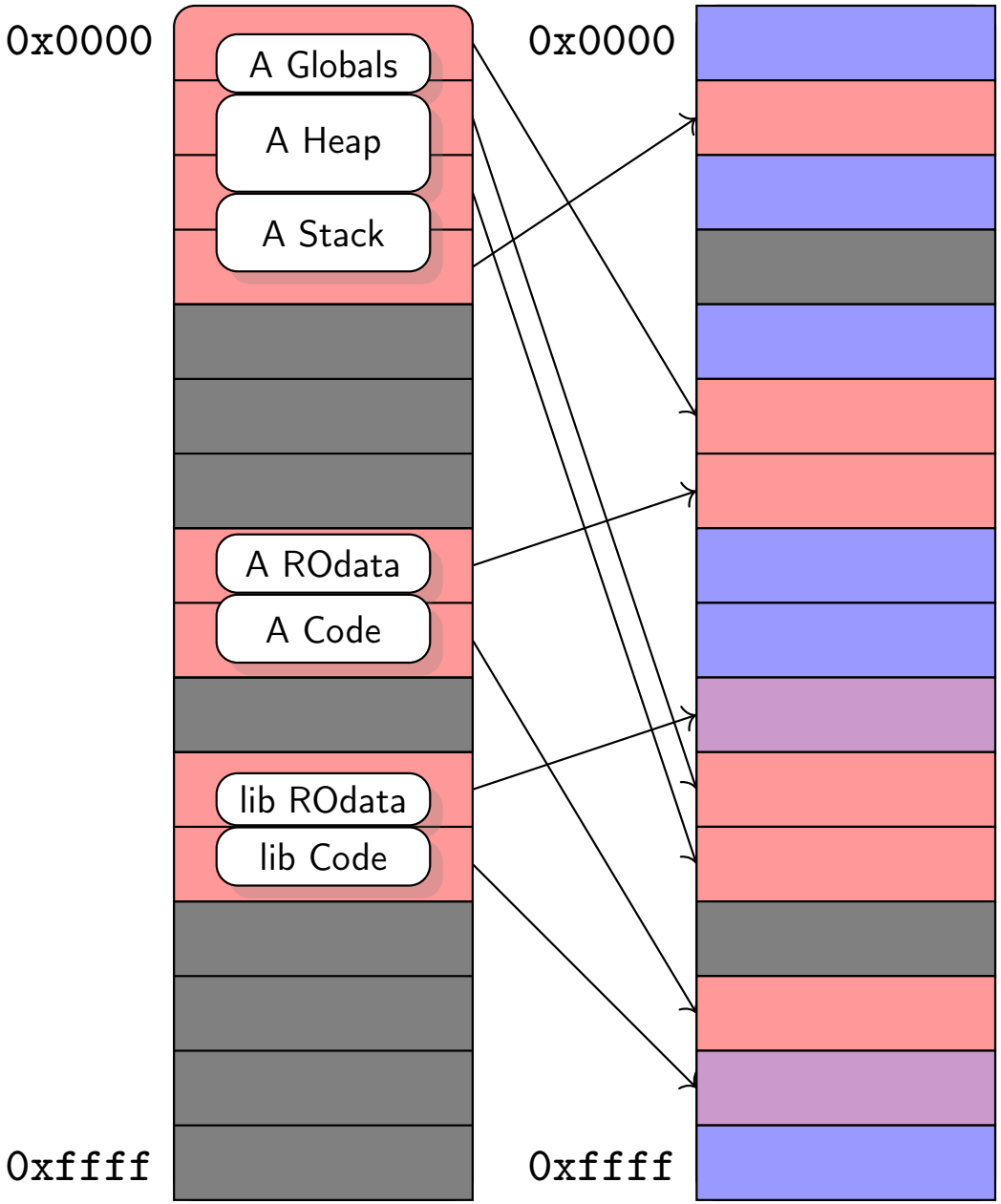
Une interruption est générée: c'est le *défaut de page*

- Prévient le noyau d'une erreur de chargement à une adresse

Comment le noyau doit traiter le défaut de page?

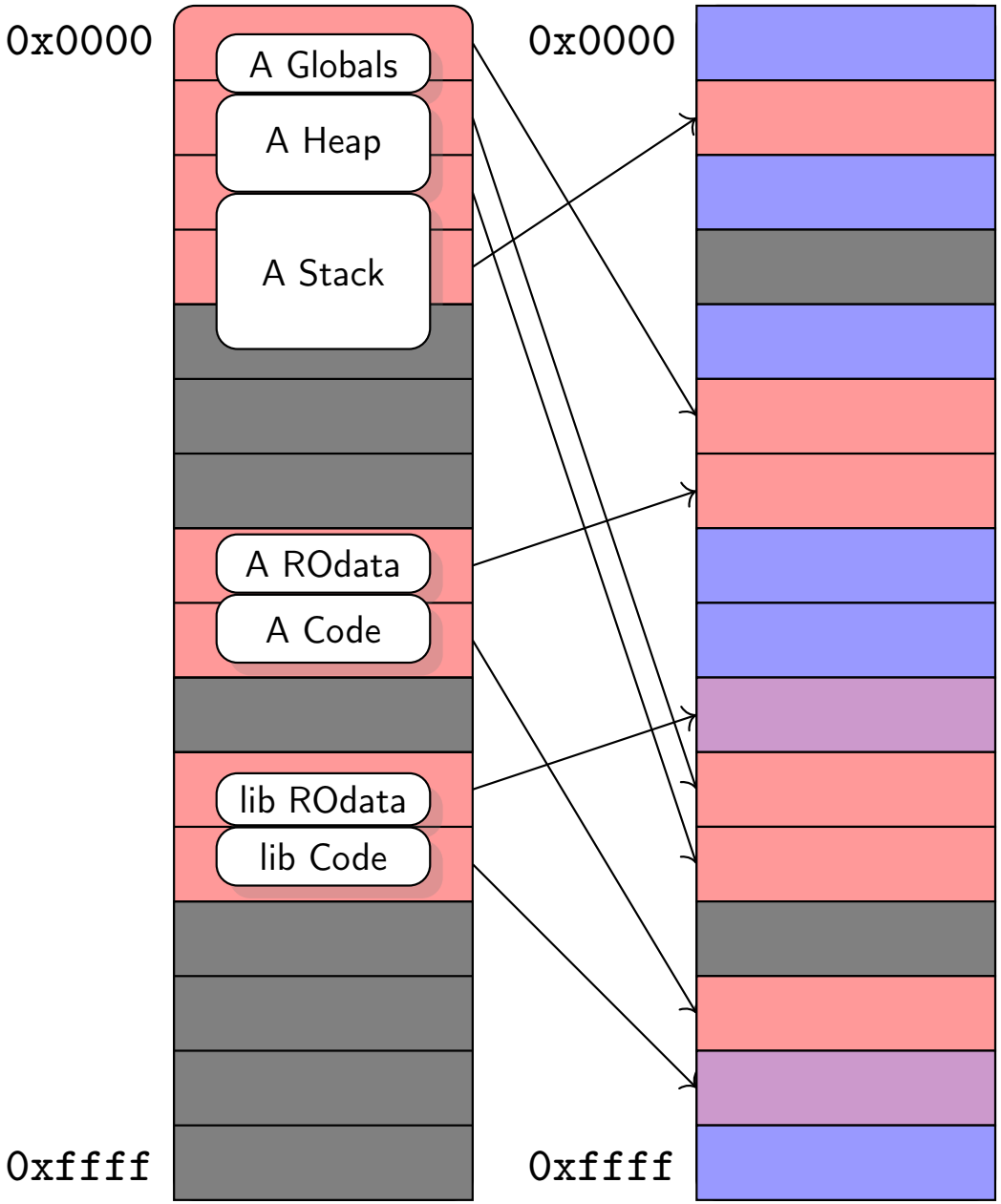
- Quand la mémoire virtuelle est utilisée seulement pour la protection mémoire: tuer ou redémarrer la tâche
- D'autres utilisations sont possibles!

Allocation mémoire à la demande



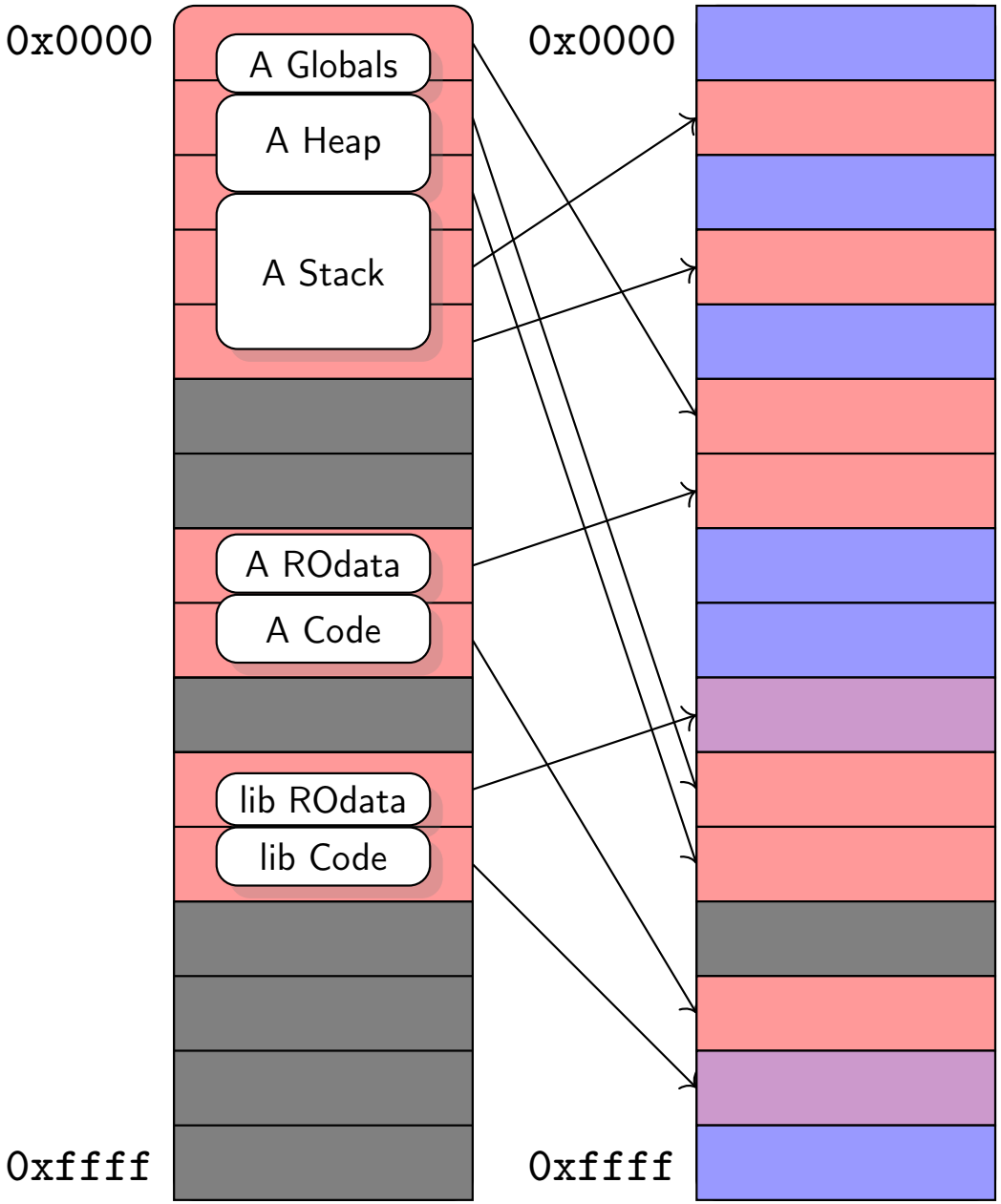
- Prog: Donne-moi de la mémoire
- Noyau: (syscall) OK
- ...

Allocation mémoire à la demande



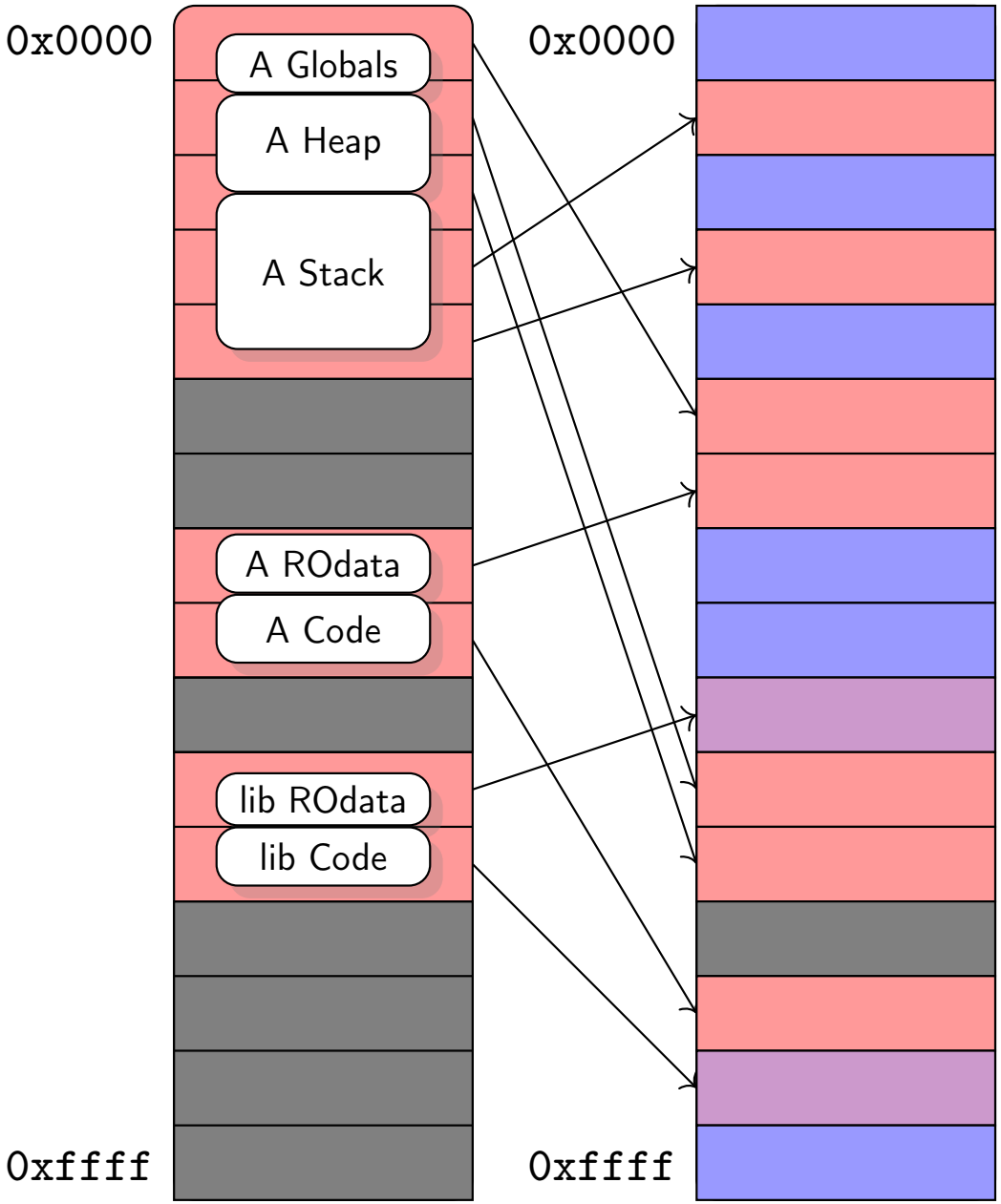
- Prog: Donne-moi de la mémoire
- Noyau: (syscall) OK
- ...
- Prog: j'accède à ma nouvelle mémoire

Allocation mémoire à la demande



- Prog: Donne-moi de la mémoire
- Noyau: (syscall) OK
- ...
- Prog: j'accède à ma nouvelle mémoire
- Noyau: (défaut de page) donne vraiment la mémoire

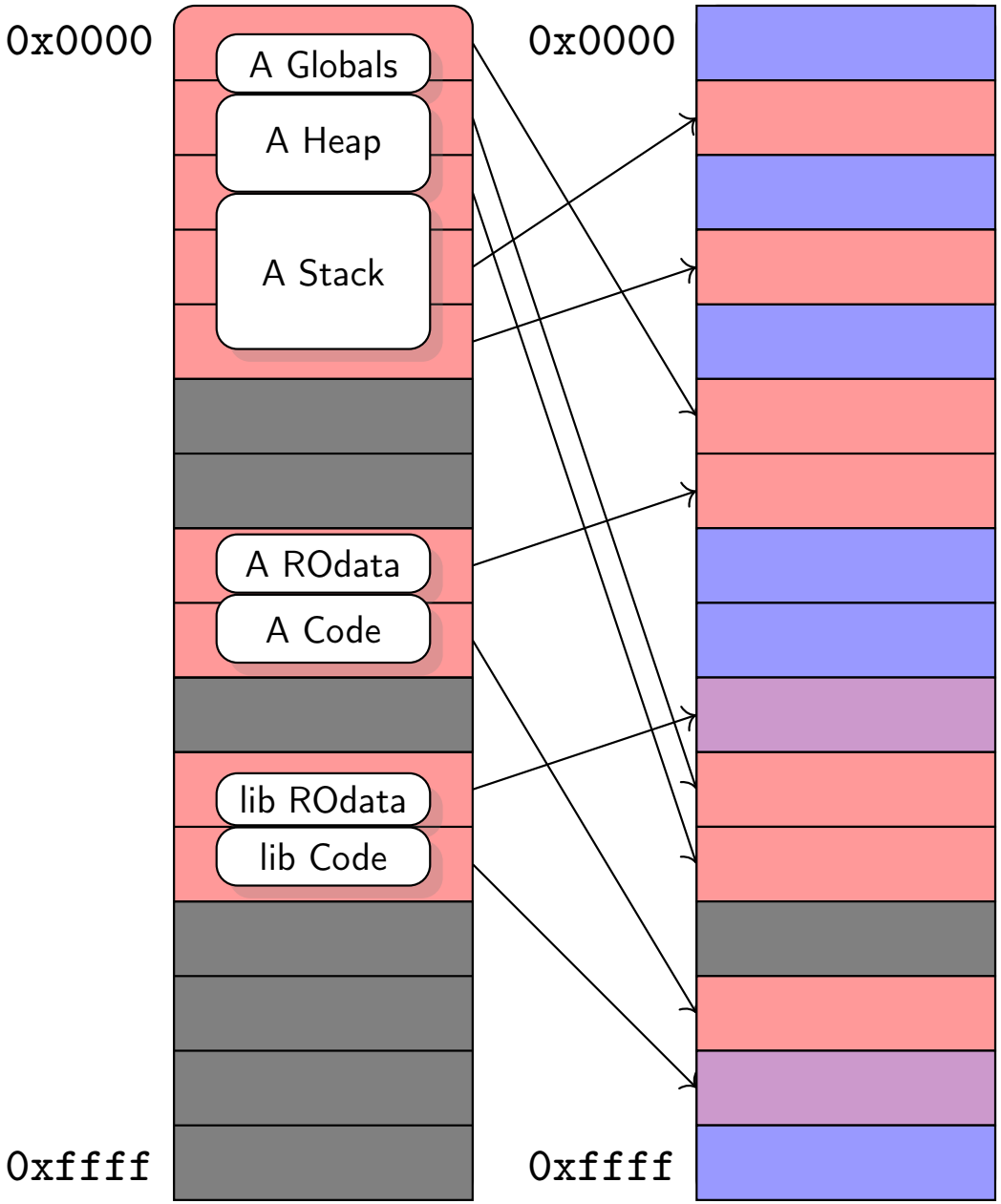
Allocation mémoire à la demande



- Prog: Donne-moi de la mémoire
- Noyau: (syscall) OK
- ...
- Prog: j'accède à ma nouvelle mémoire
- Noyau: (défaut de page) donne vraiment la mémoire

Intérêt

Allocation mémoire à la demande



- Prog: Donne-moi de la mémoire
- Noyau: (syscall) OK
- ...
- Prog: j'accède à ma nouvelle mémoire
- Noyau: (défaut de page) donne vraiment la mémoire

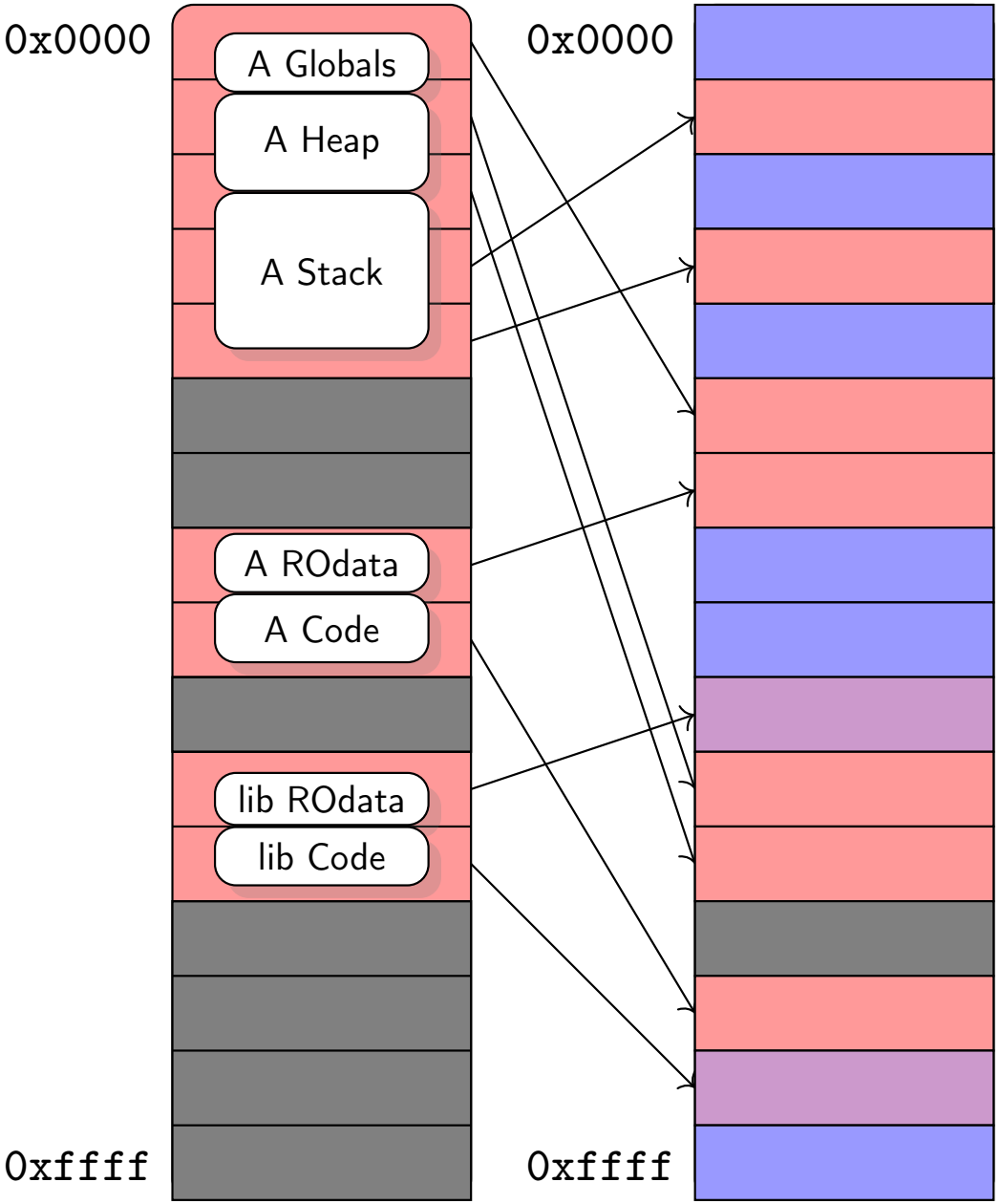
Intérêt

Ne dépense de la mémoire que lorsque/si le programme en a besoin

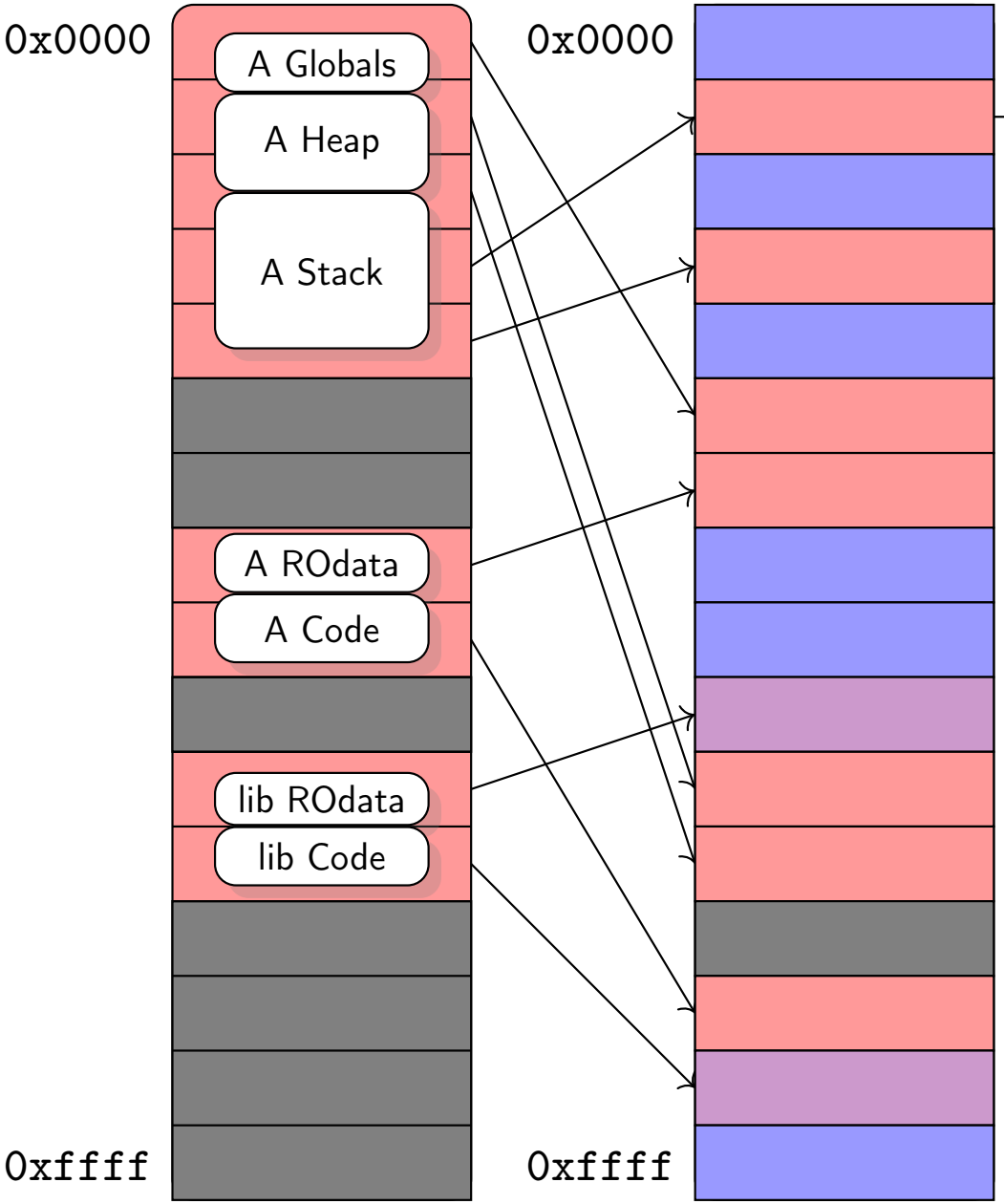
Man malloc

```
...
By default, Linux follows an optimistic memory allocation
 strategy.This means that when malloc() returns non-NULL
there is no guarantee that the memory really is available.
...
```

Swapping: la mémoire vue comme un cache



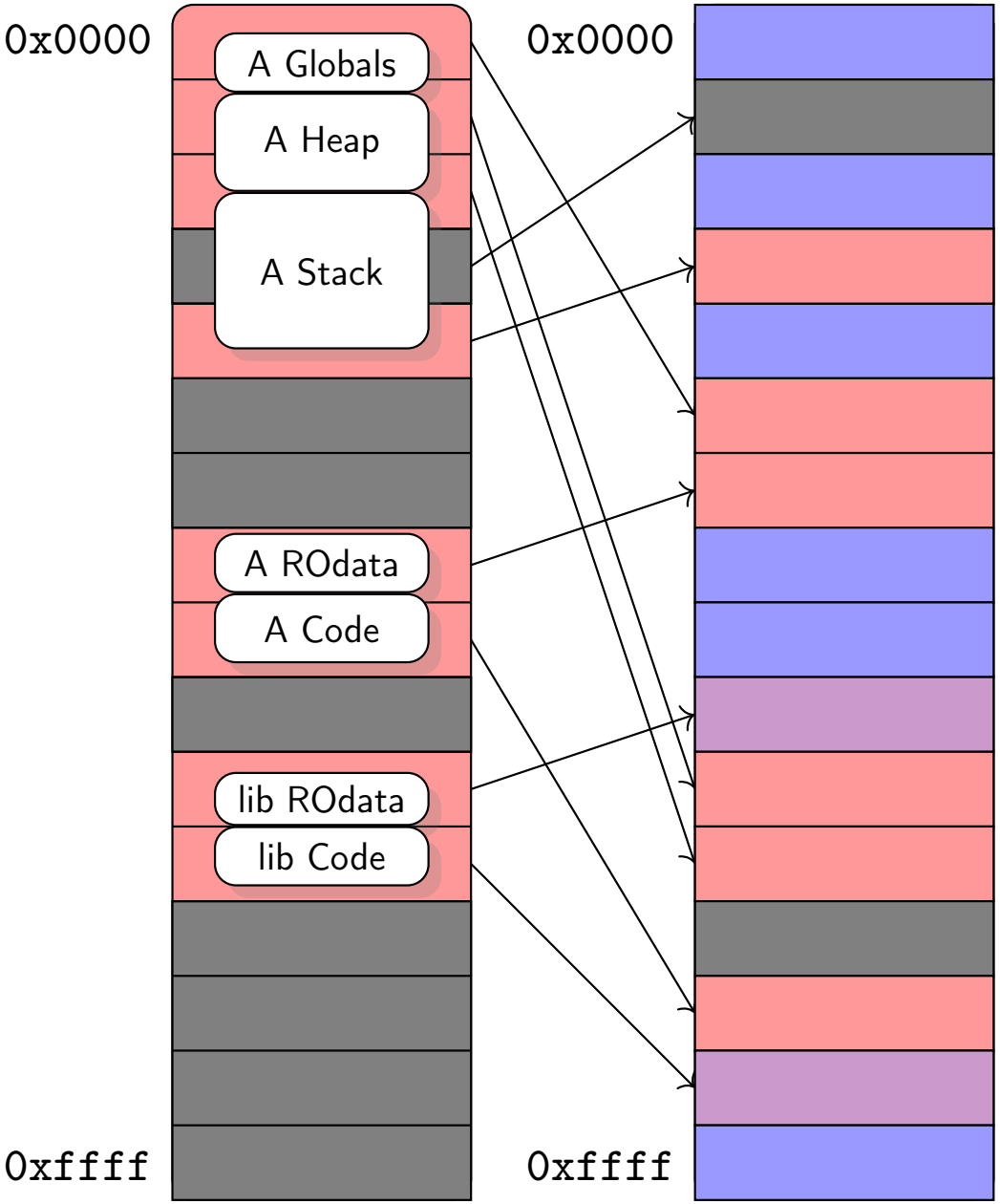
Swapping: la mémoire vue comme un cache



→ disque dur

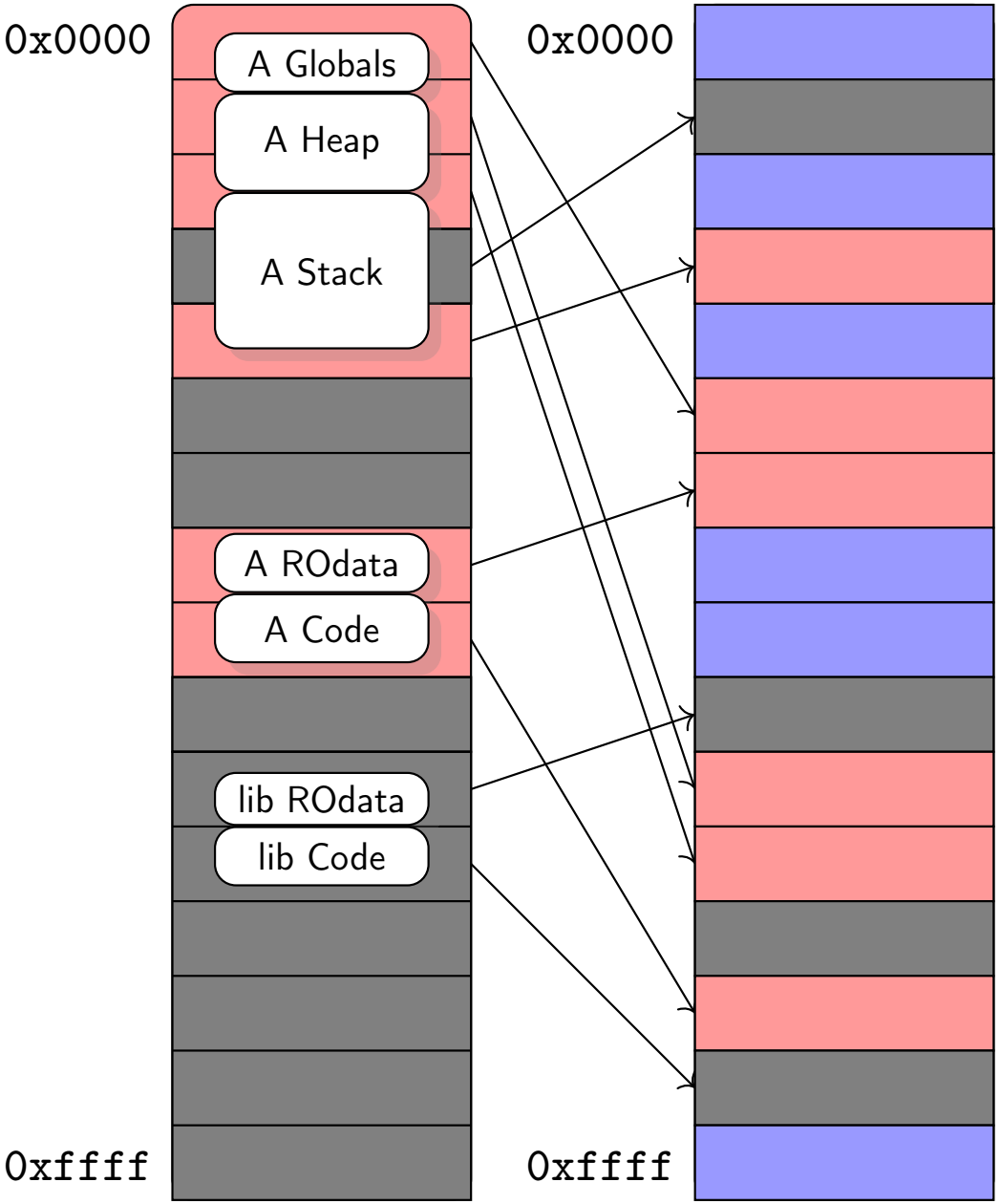
- Copie de pages mémoires vers le disque dur

Swapping: la mémoire vue comme un cache



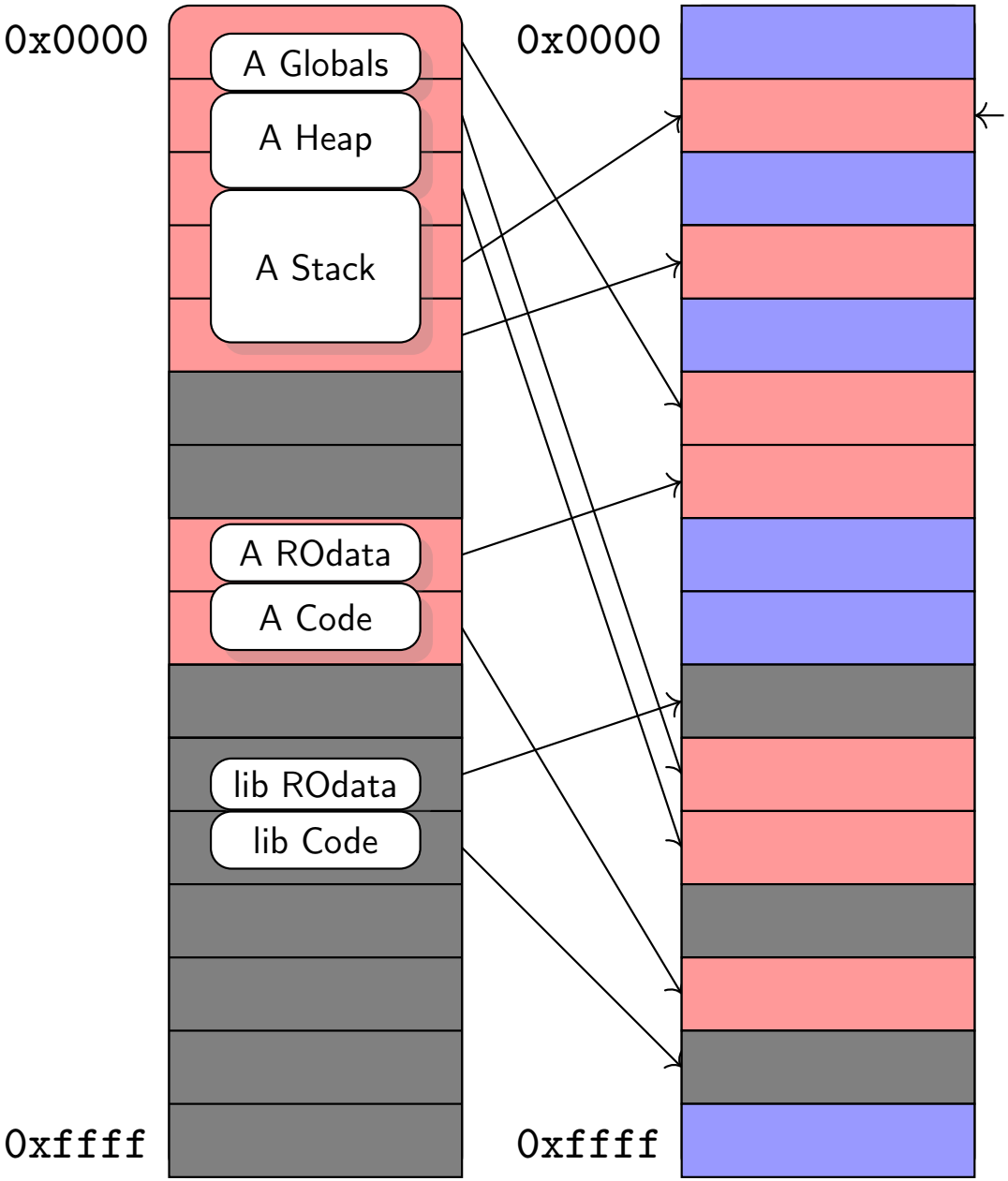
- Copie de pages mémoires vers le disque dur
- Libération des pages copiées sur disque dur

Swapping: la mémoire vue comme un cache



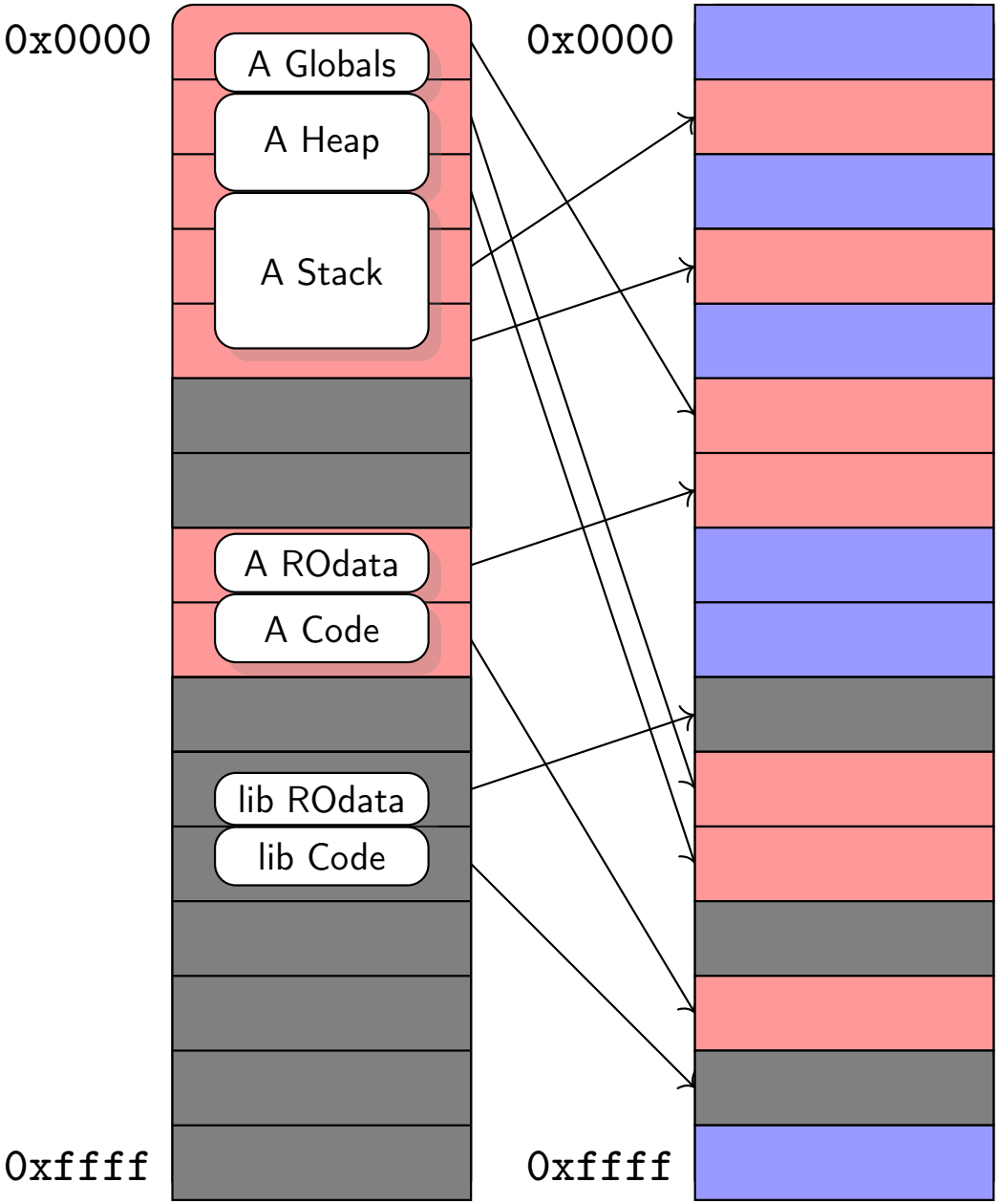
- Copie de pages mémoires vers le disque dur
- Libération des pages copiées sur disque dur
- Pas toujours besoin de copier vers le disque:
  - Code et constantes des librairies (déjà dans un fichier)
  - Pages non modifiée depuis la dernière copie sur le disque

Swapping: la mémoire vue comme un cache



- Copie de pages mémoires vers le disque dur
- Libération des pages copiées sur disque dur
- Pas toujours besoin de copier vers le disque:
  - Code et constantes des librairies (déjà dans un fichier)
  - Pages non modifiée depuis la dernière copie sur le disque
- Copie depuis le disque dur lors des défauts de page

Swapping: la mémoire vue comme un cache



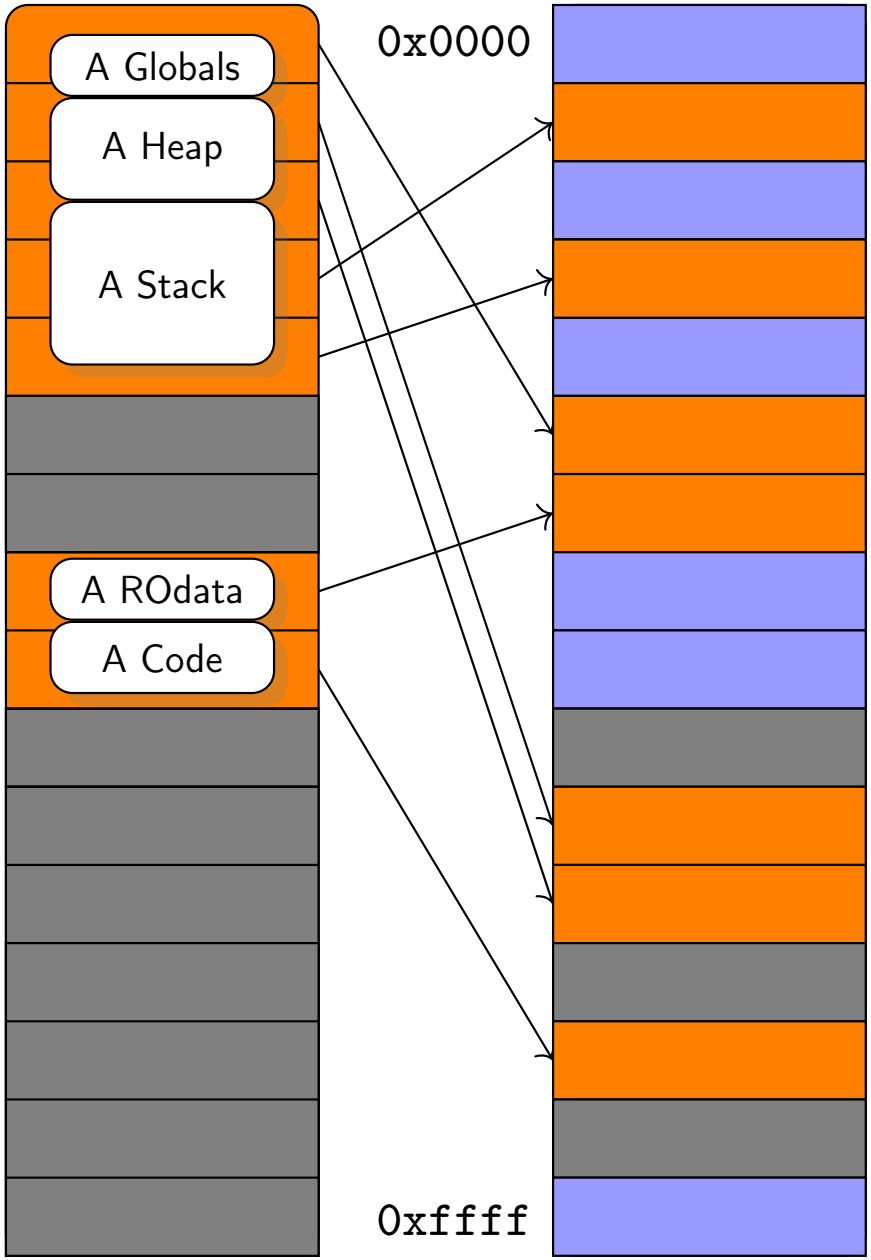
- `mmap`: appel système UNIX permettant de projeter un fichier en mémoire.
- Fonctionne sur le même principe (récupération automatique, sur le disque, de la partie du fichier, lors des défauts de page).

Exemple: top

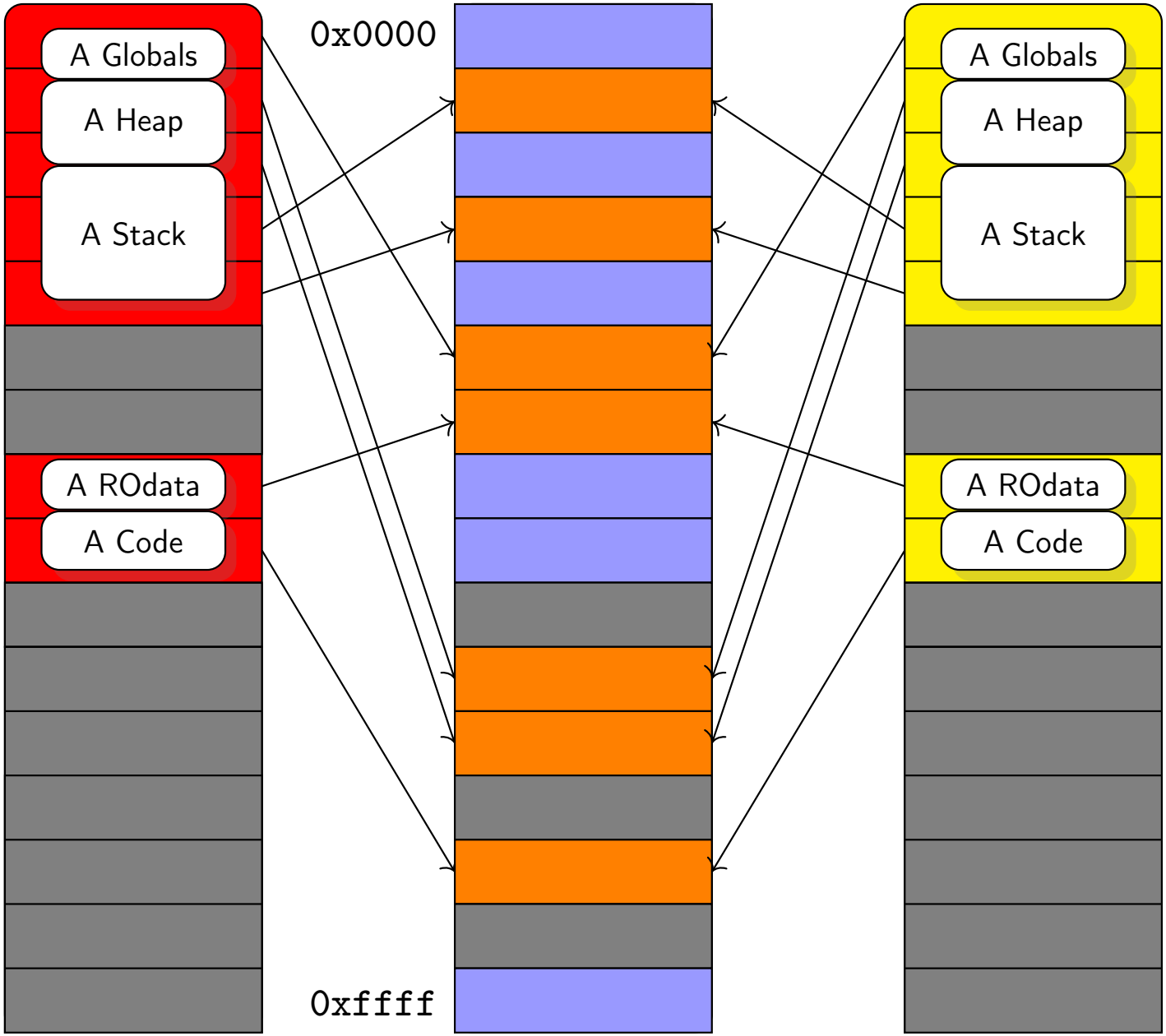
```
top - 22:27:29 up 1 day, 11:50, 12 users,  load average: 0.41, 0.45, 0.42
Tasks: 212 total,   1 running, 159 sleeping,   0 stopped,   0 zombie
%Cpu(s):  4.0 us,  2.3 sy,  0.0 ni, 93.3 id,  0.1 wa,  0.0 hi,  0.3 si,  0.
KiB Mem : 20250460 total,   733424 free,  2996680 used, 16520356 buff/cache
KiB Swap: 14325756 total, 14325756 free,        0 used. 17440368 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
12919 matthieu  20   0 3121324 739968 143900 S   7.3  3.7  76:09.07 firefox
 1247 matthieu  20   0  430156 107116  95656 S   3.6  0.5  13:35.63 Xorg
12970 matthieu  20   0 2105116 420108 103360 S   2.3  2.1  29:43.22 Web Con
```

`fork()` avec copy on write

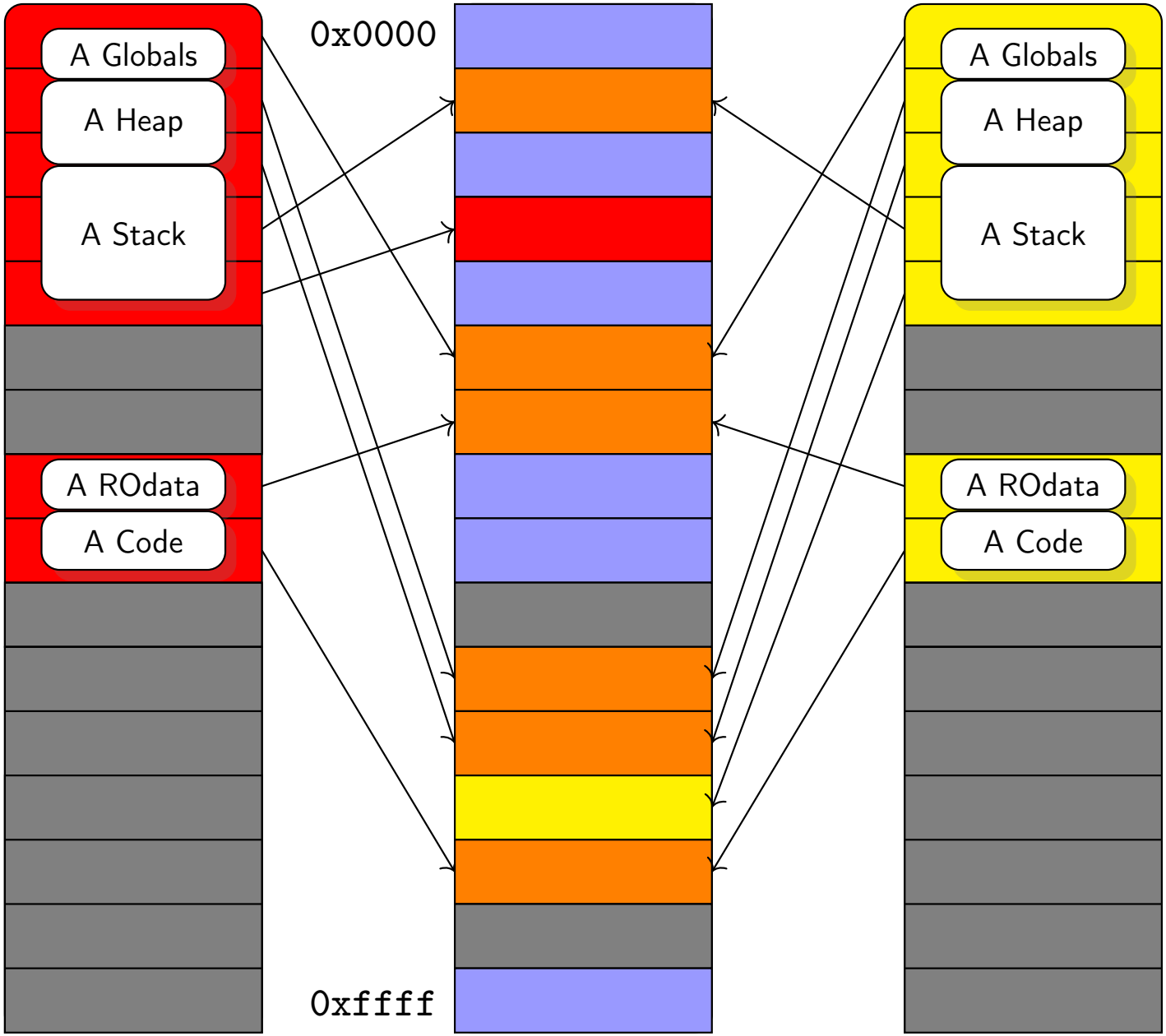


`fork()` avec copy on write



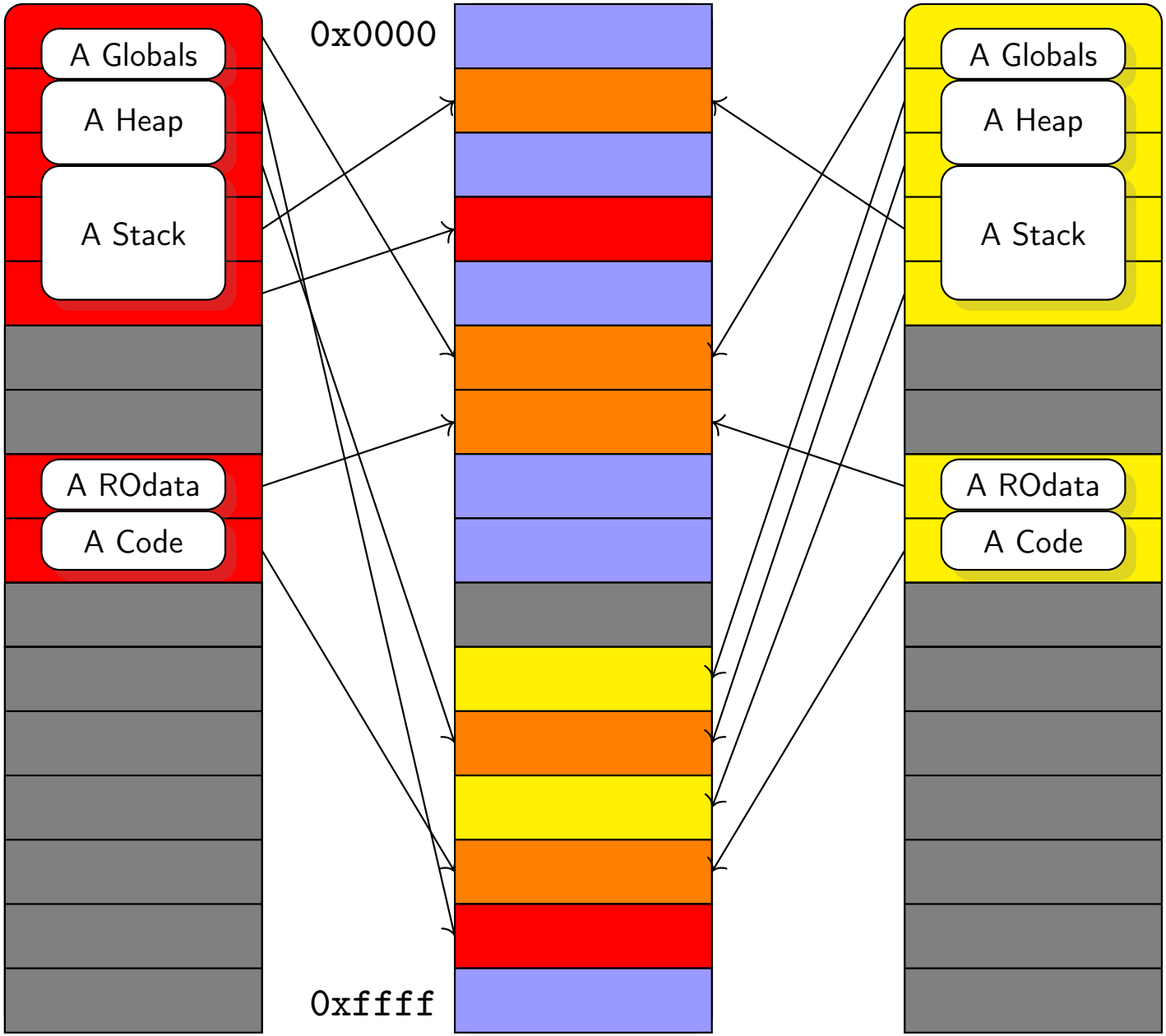
- `fork()` duplique un processus
- Initialement, toute la mémoire est partagée entre les deux copies

`fork()` avec copy on write



- `fork()` duplique un processus
- Initialement, toute la mémoire est partagée entre les deux copies
- Les pages sont dupliquées à la première modification (accès aux pages partagées en lecture seule)

`fork()` avec copy on write



- `fork()` duplique un processus
- Initialement, toute la mémoire est partagée entre les deux copies
- Les pages sont dupliquées à la première modification (accès aux pages partagées en lecture seule)

Sommaire

1. Cours 5: Protection et organisation de la mémoire
   - Le confinement mémoire
   - Mémoire virtuelle et pagination
   - Conclusion

Conclusion

- Confinement mémoire
  - Protège un processus des erreurs mémoires commises par les autres threads
  - Par séparation physique, protection logicielle, ou protection matérielle (le plus courant)
  - La MPU ou la MMU, programmé par le noyau, permet de réaliser cette protection.
- Organisation de la mémoire
  - La MMU permet également la virtualisation (ou translation) d'adresses
  - Évite la fragmentation externe
  - Permet d'implanter des fonctionnalités utiles (extension de mémoire avec swap, `fork` efficace, etc.)