Programmation Système

Département Informatique

IUT2 de Grenoble

BUT2 - Ressource 3.05

Organisation du cours

1. Système d'exploitation
2. Processus
3. Partage des ressources
4. Système de Gestion de Fichiers
5. Entrées/Sorties

3. Partage des ressources

3.1. Partage du CPU

- Etats d'un processus
- Ordonnancement de processus

3.2. Partage de la mémoire centrale

- Pagination
- Mémoire virtuelle

RAM partagée entre processus (Rappel)

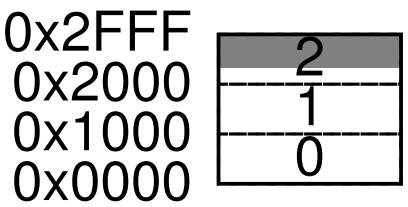
Deux objectifs

- chaque processus définit **ses adresses de façon indépendante** *(chacun peut utiliser la même adresse mémoire pour pointer une donnée différente)*
- taille des espaces mémoires (4 Gio ou 4 Exio par proc.)
  $\gg$
  taille de l'espace physique (16 Gio)

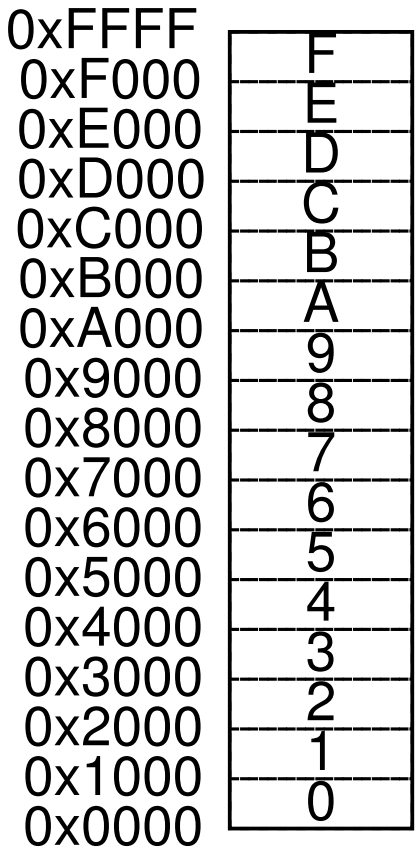
Deux solutions complémentaires

- pagination
- mémoire virtuelle

Pagination : Principe (Rappel)



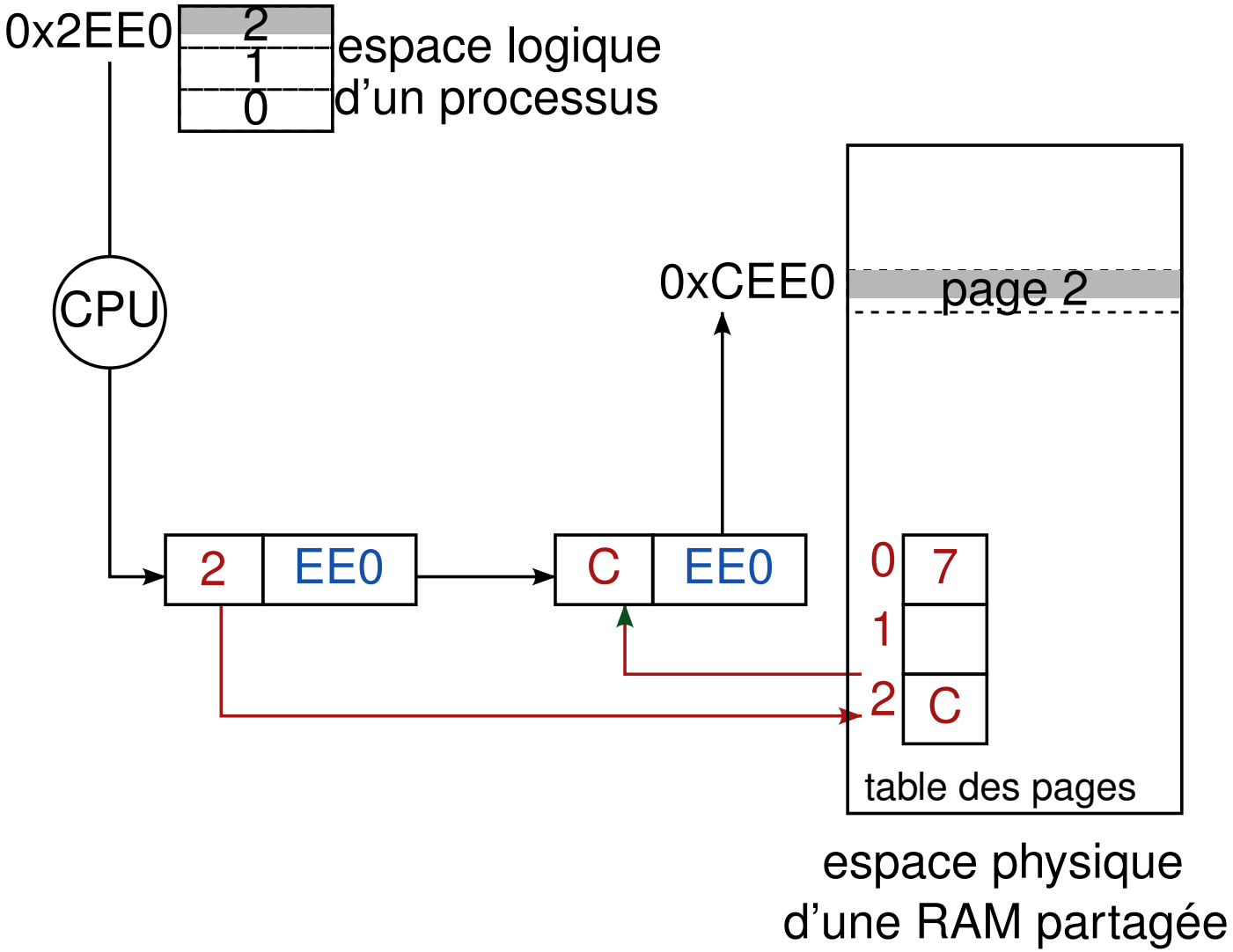
espace logique
d'un processus
de 12 Kio = 3 pages



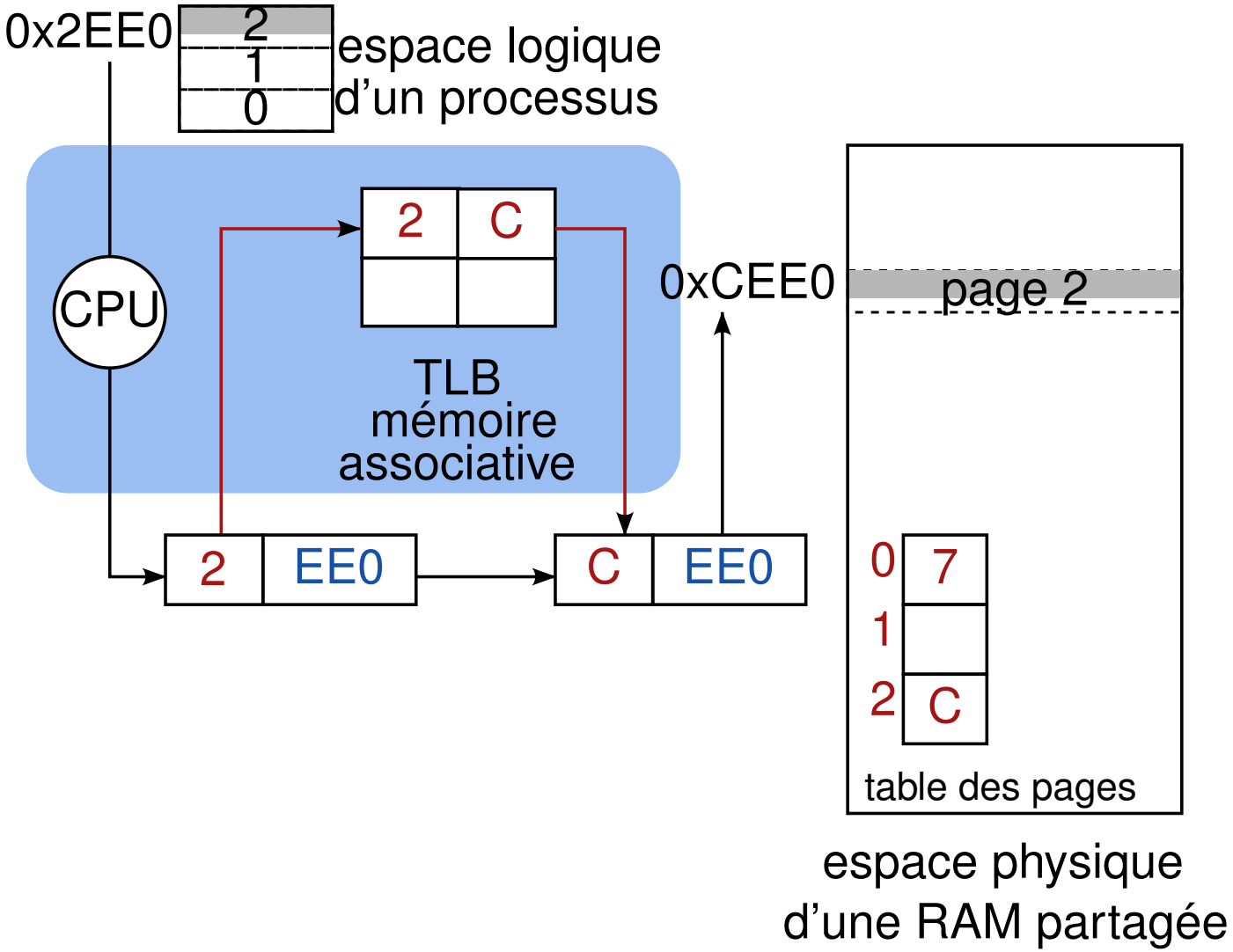
espace physique
d'une RAM partagée
de 64 Kio = 16 pages

taille d'une page = taille d'une case = 4 Kio = 0x1000 octets

Pagination : Traduction par table des pages (Rappel)



Pagination : Traduction par TLB (Rappel)



Mémoire virtuelle

Principe

ne charger une page en RAM que si elle est utilisée

Implications

- espace logique d'un processus $\gg$ RAM
- table des pages : + information de présence
- table des pages peut elle-même être paginée : n'en créer que les parties utiles

Mémoire virtuelle

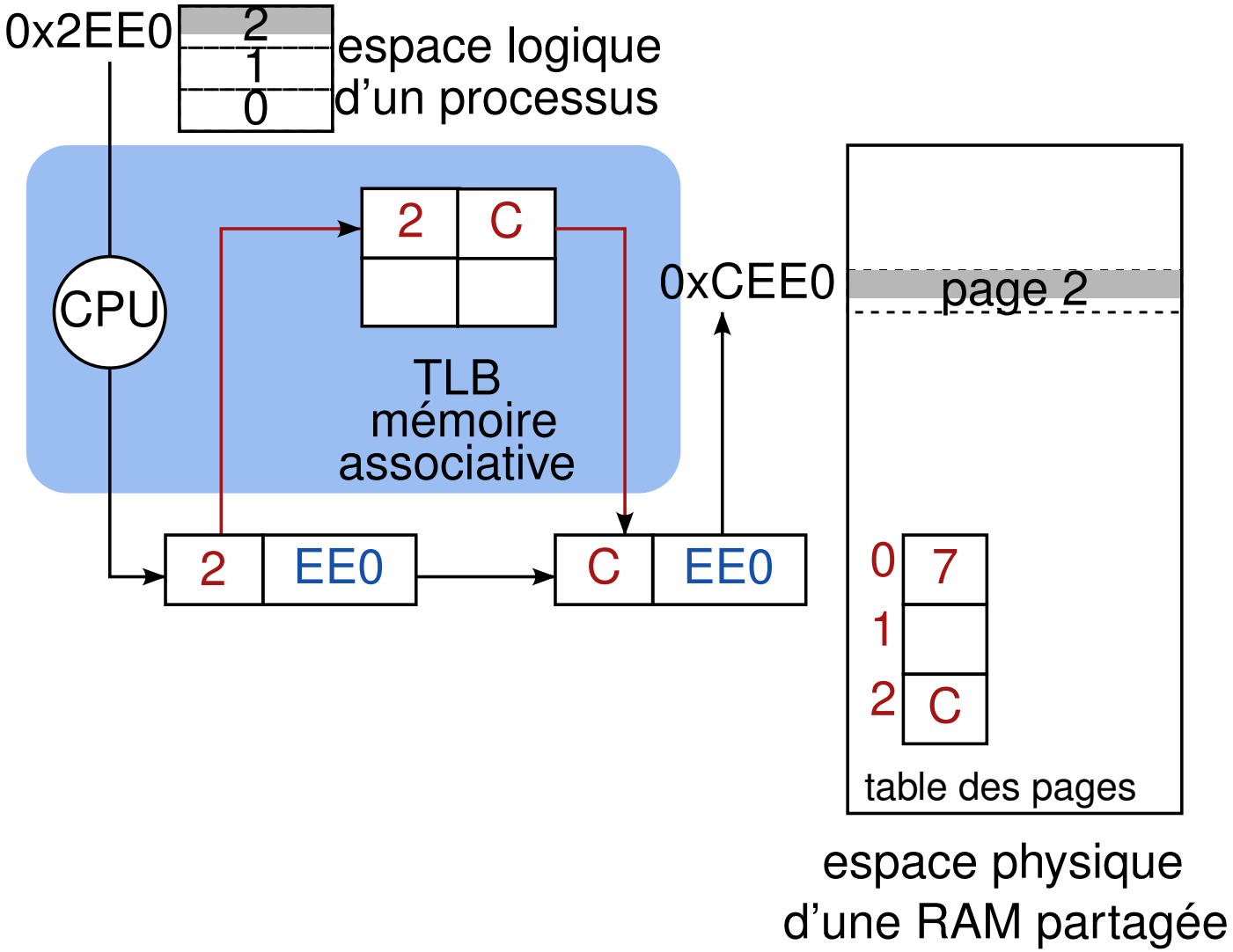
Principe

ne charger une page en RAM que si elle est utilisée

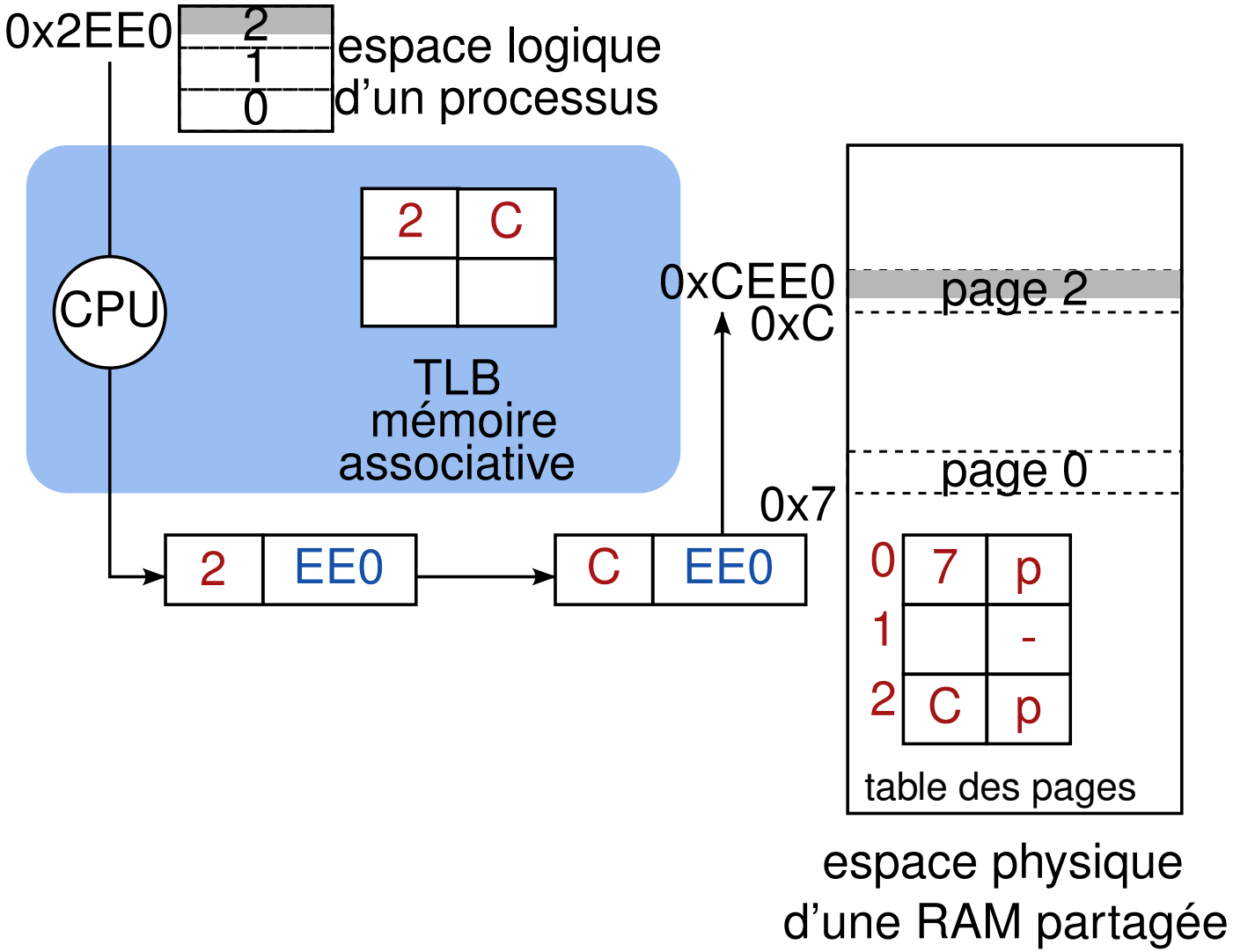
Implications

- espace logique d'un processus $\gg$ RAM
- table des pages : + information de présence
- table des pages peut elle-même être paginée : n'en créer que les parties utiles

Pagination et mémoire virtuelle : Bit de présence



Pagination et mémoire virtuelle : Bit de présence



Mémoire virtuelle

Principe

ne charger une page en RAM que si elle est utilisée

Implications

- espace logique d'un processus $\gg$ RAM
- table des pages : + information de présence
- table des pages peut elle-même être paginée : n'en créer que les parties utiles

Mémoire virtuelle

Principe

ne charger une page en RAM que si elle est utilisée

Implications

- espace logique d'un processus $\gg$ RAM
- table des pages : + information de présence
- table des pages peut elle-même être paginée : n'en créer que les parties utiles

Défaut de page

Définition

demander une page qui n'est pas chargée en RAM

Résolution

- commutation de processus (gestion de ddp)
- trouver une case physique libre
- mettre à jour table des pages + TLB
- commut. de proc. et réexecution de l'instr. ayant mené au ddp

Majeur/Mineur

- s'il nécessite une E/S (disque) pour être résolu : **défaut majeur** ;
- sinon : **défaut mineur**

Défaut de page

Définition

demander une page qui n'est pas chargée en RAM

Résolution

- commutation de processus (gestion de ddp)
- trouver une case physique libre
- mettre à jour table des pages + TLB
- commut. de proc. et réexecution de l'instr. ayant mené au ddp

Majeur/Mineur

- s'il nécessite une E/S (disque) pour être résolu : **défaut majeur** ;
- sinon : **défaut mineur**

Défaut de page : dirty bit

TD

Mise en réserve (swap)

Si RAM pleine, le *kernel swap daemon* retire des pages :

- page non modifiée => case vidée
- page modifiée (dirty) => page déplacée en mem. sec. : swap

Quand rencontre-t-on un DDP **majeur** ?

Si on demande une page

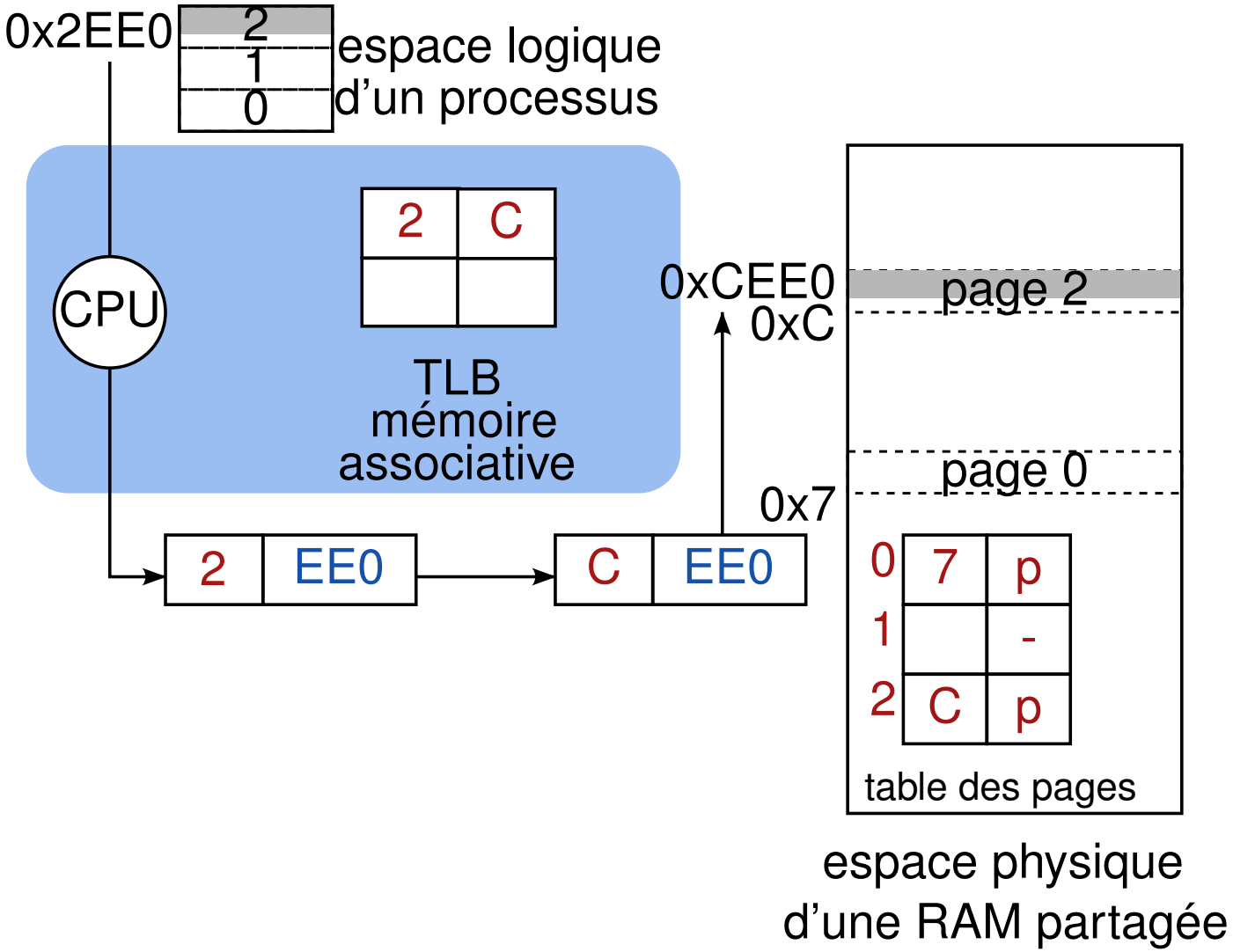
- **absente et dirty (à rechercher depuis le swap)**
- appartenant à un "segment *mappé* depuis un fichier"

Quand rencontre-t-on un DDP **mineur** ?

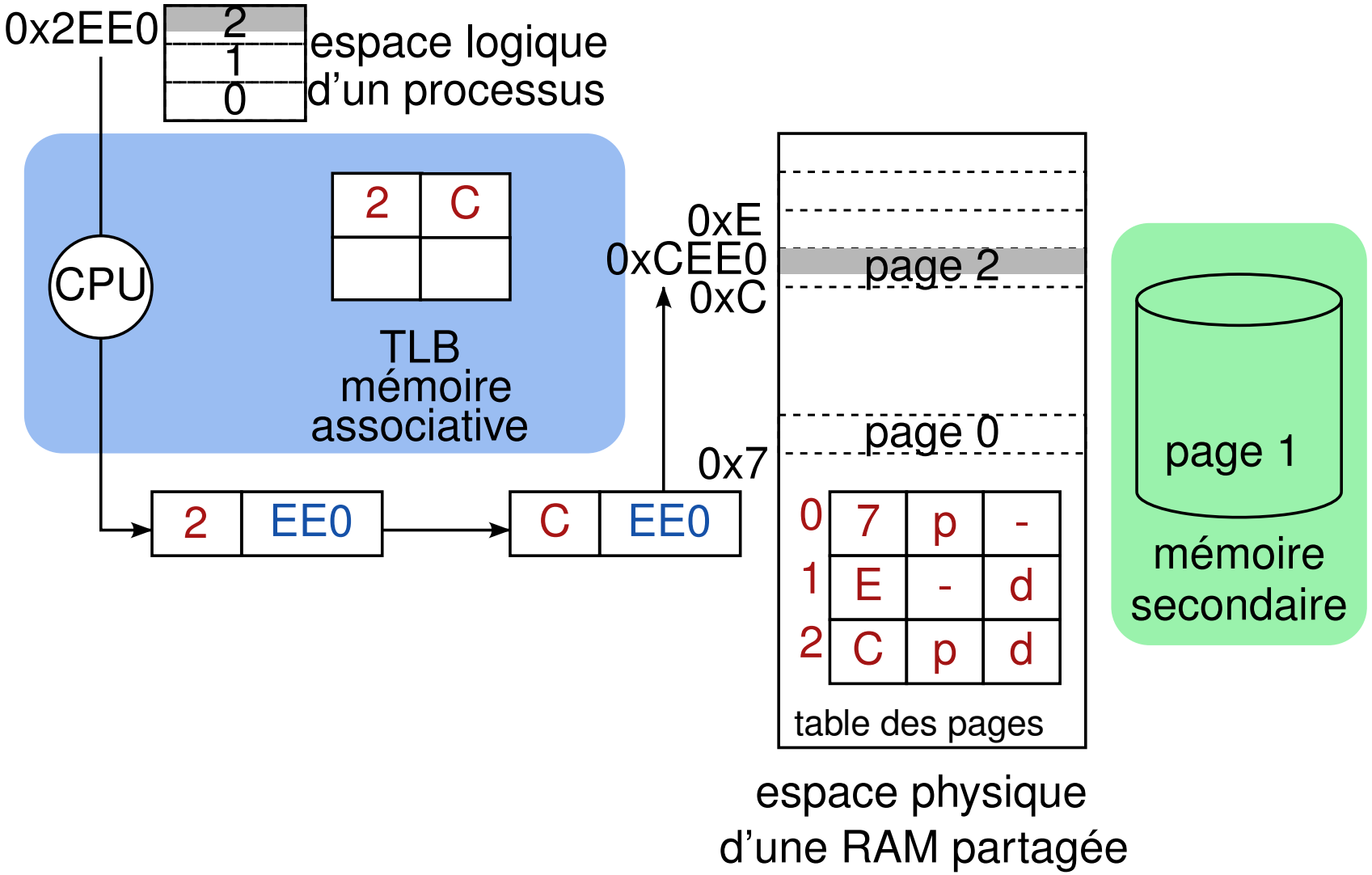
Si on demande une page **absente et non dirty**.

Dirty bit pour chaque page, positionné à vrai s'il y a eu écriture.

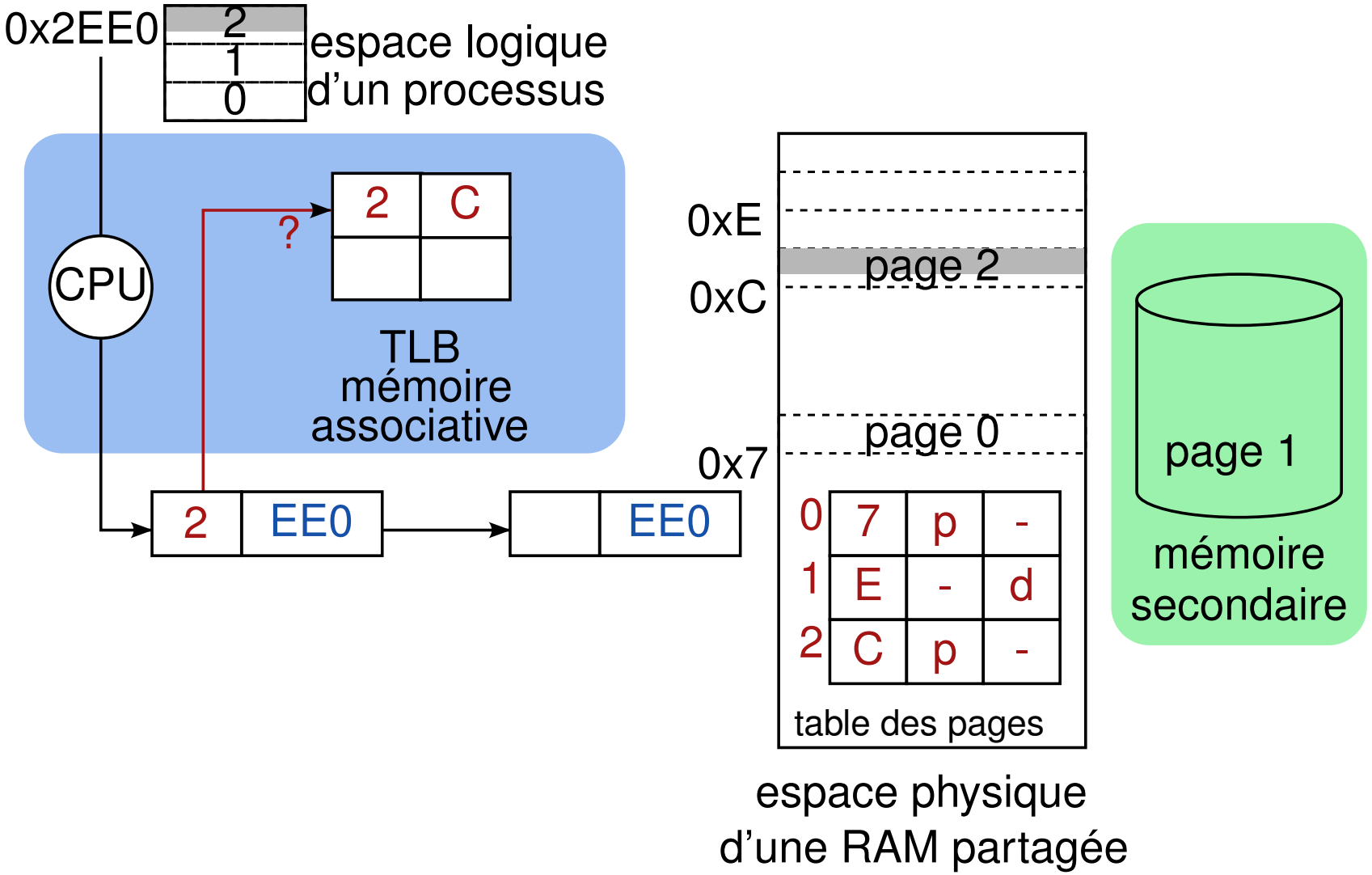
Défaut de page : dirty bit



Défaut de page : dirty bit

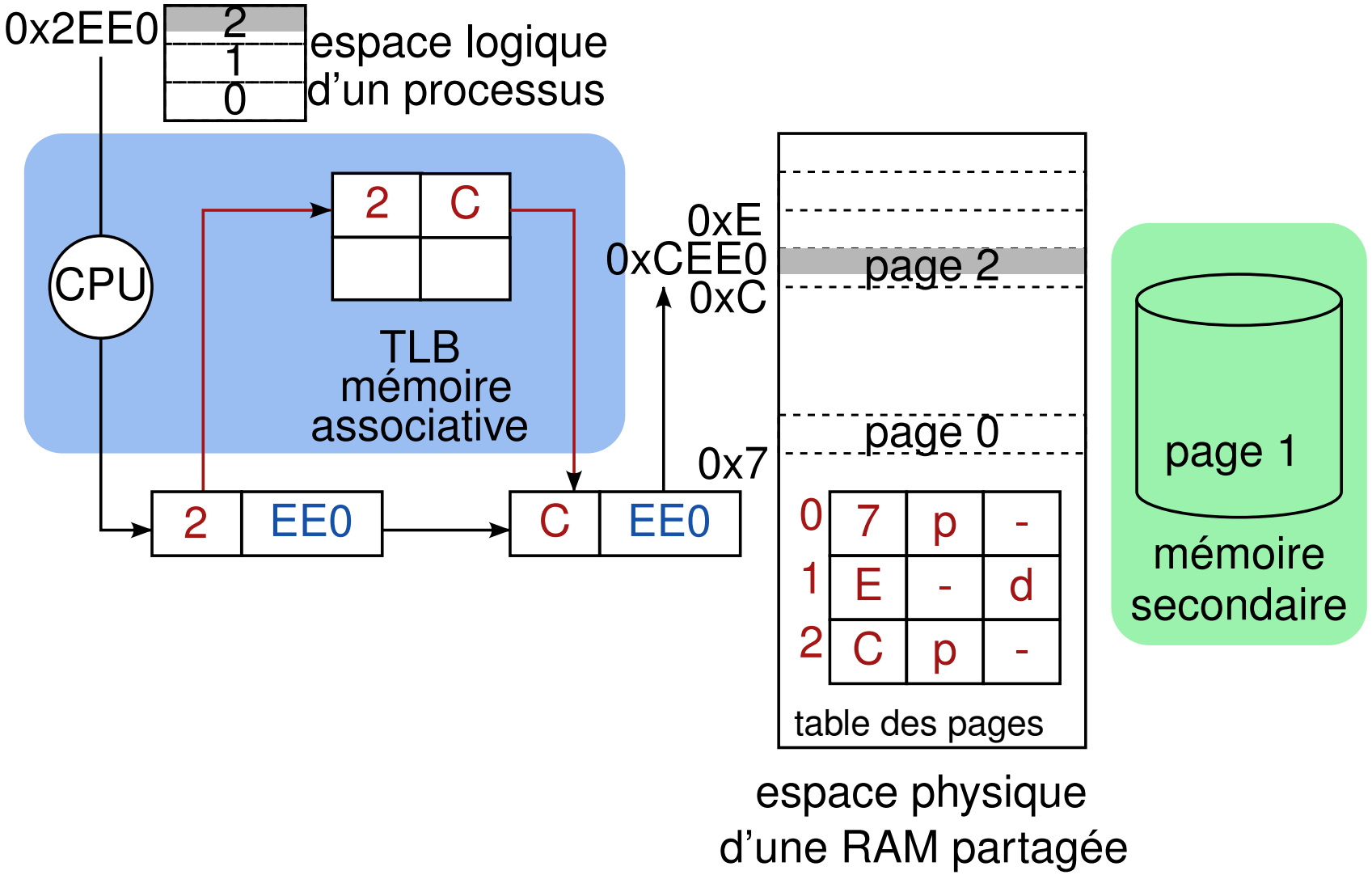


Cas 1 : pas de défaut de page, 1 accès mémoire

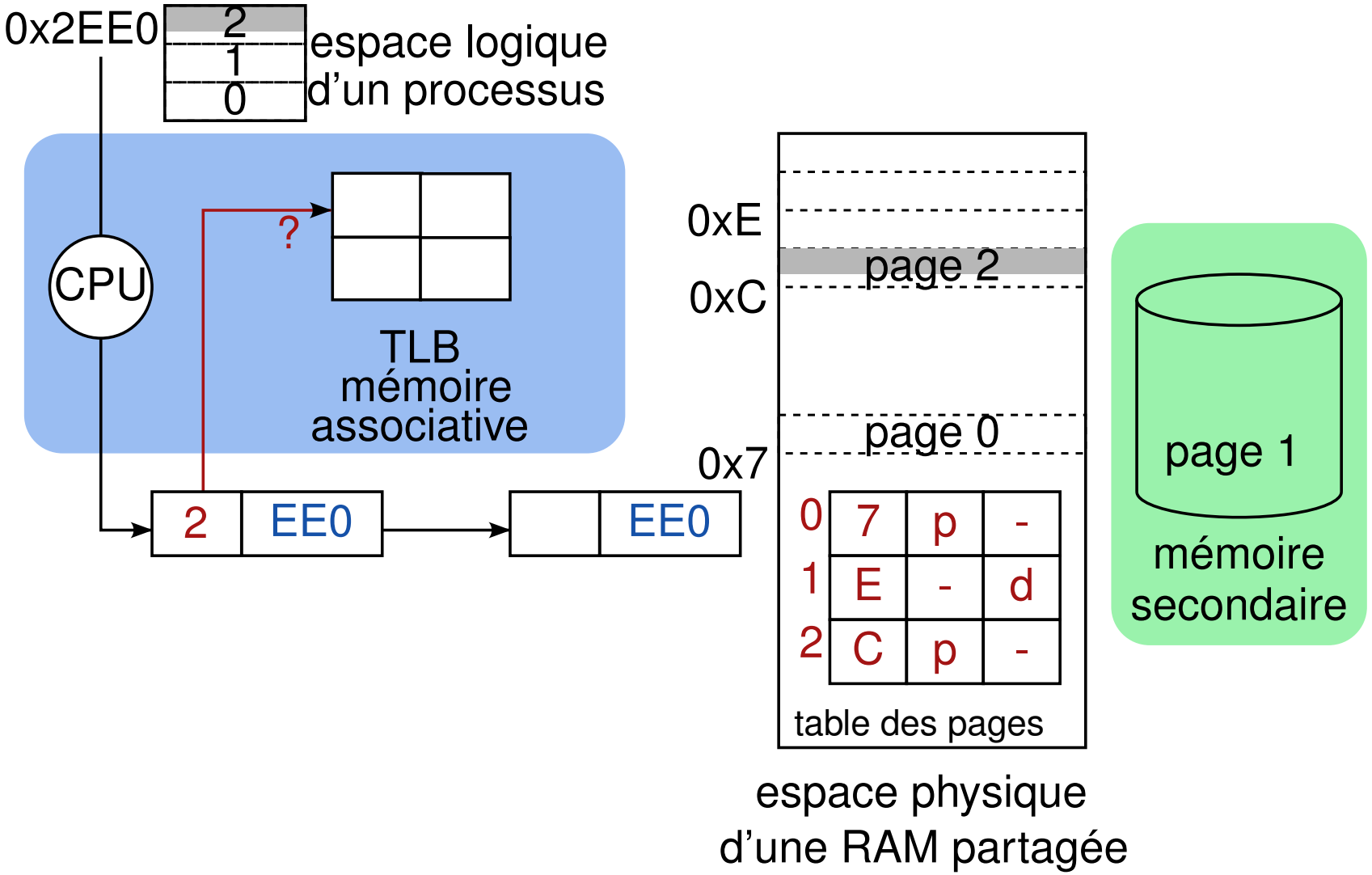


Cas 1 : pas de défaut de page, 1 accès mémoire

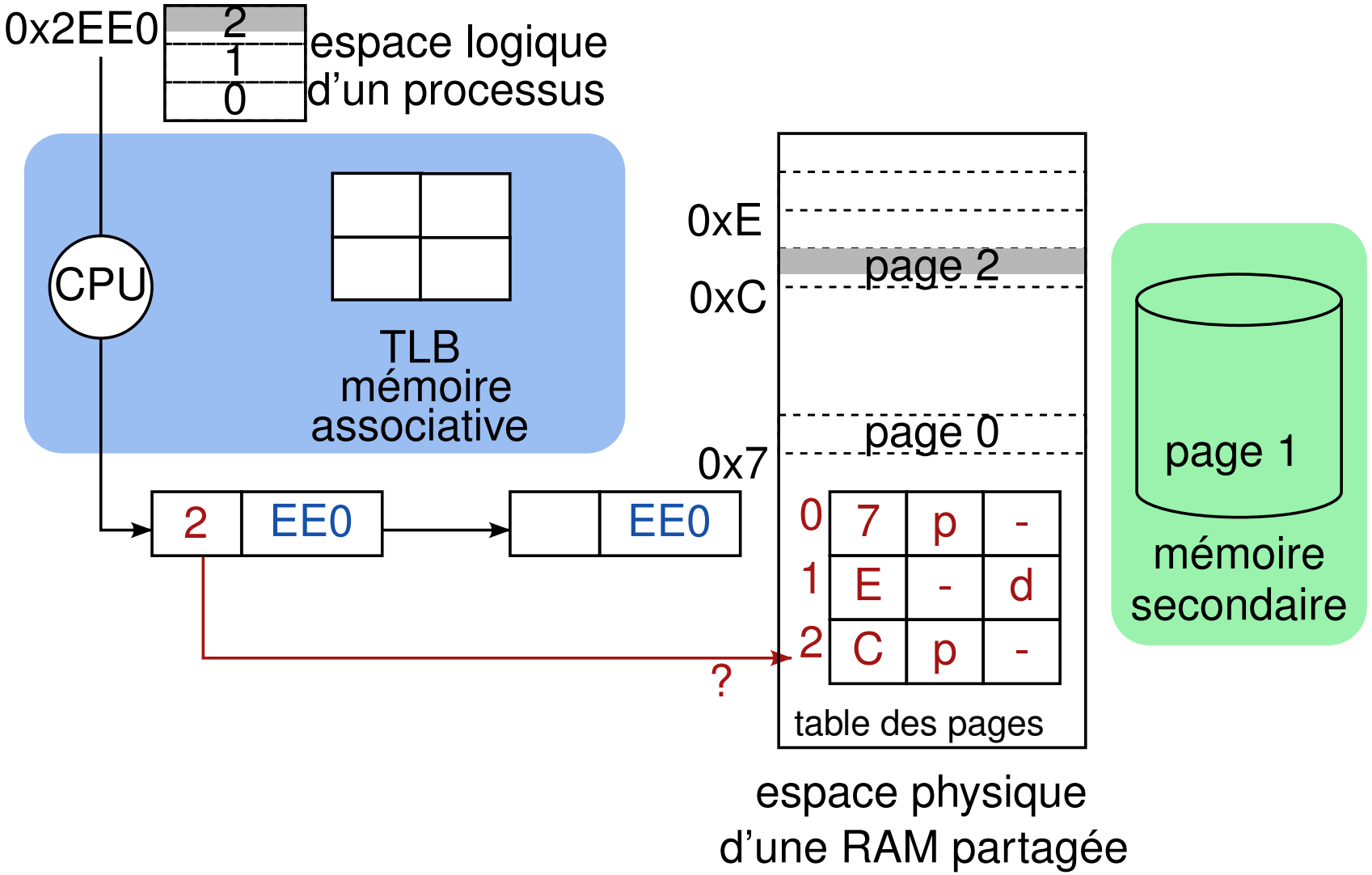
TD



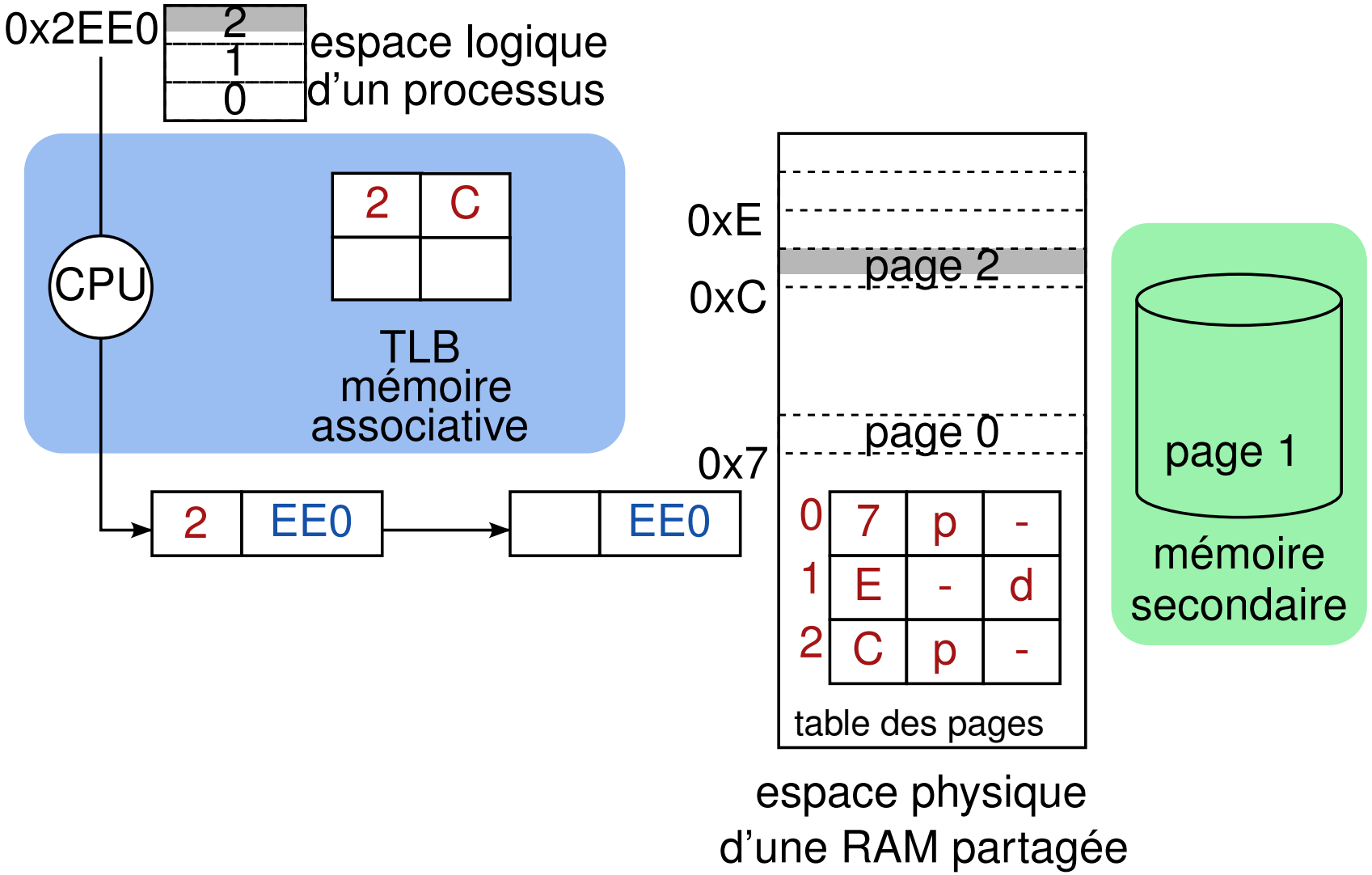
Cas 2 : pas de défaut de page, 2 accès mémoire



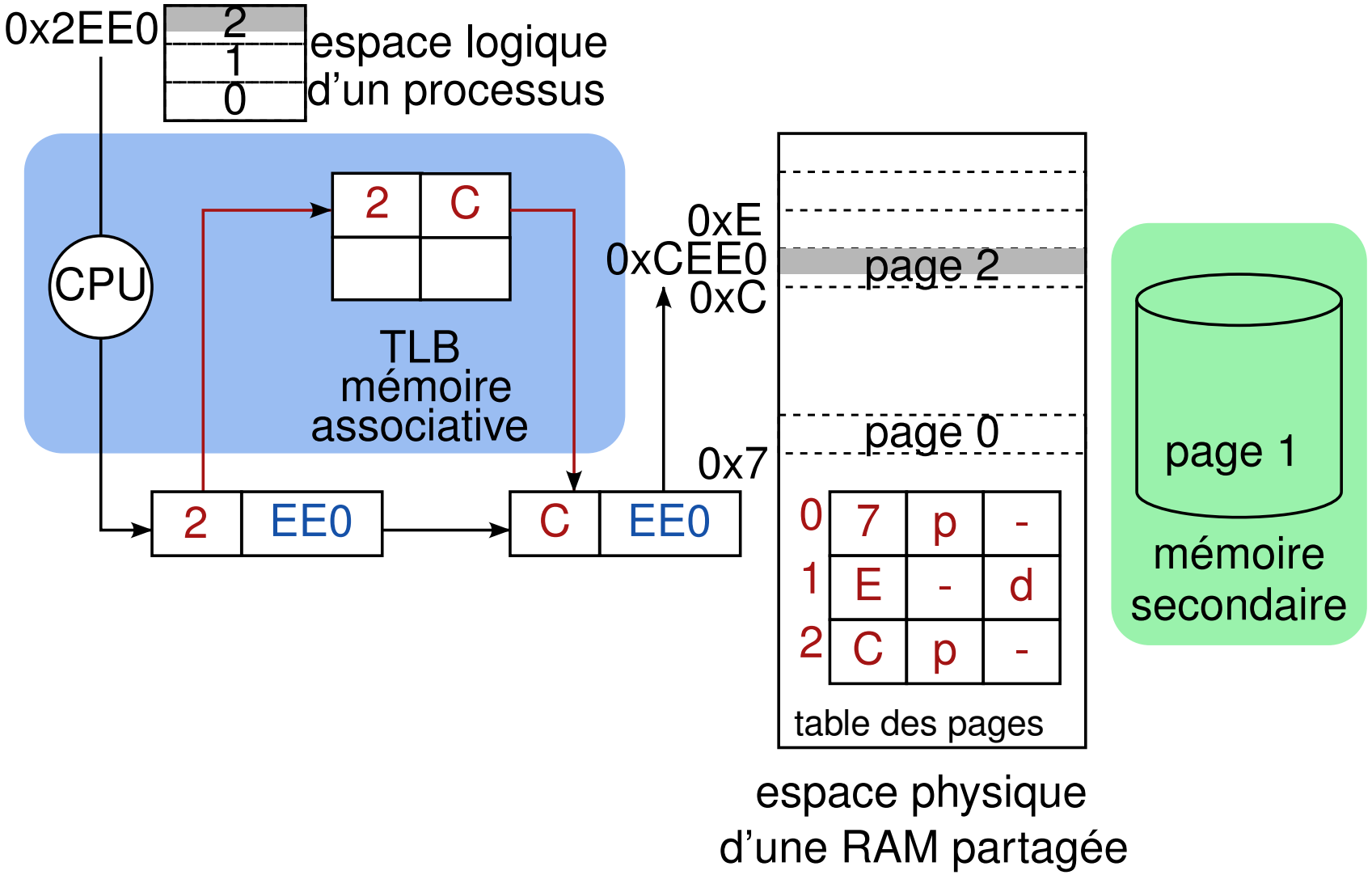
Cas 2 : pas de défaut de page, 2 accès mémoire



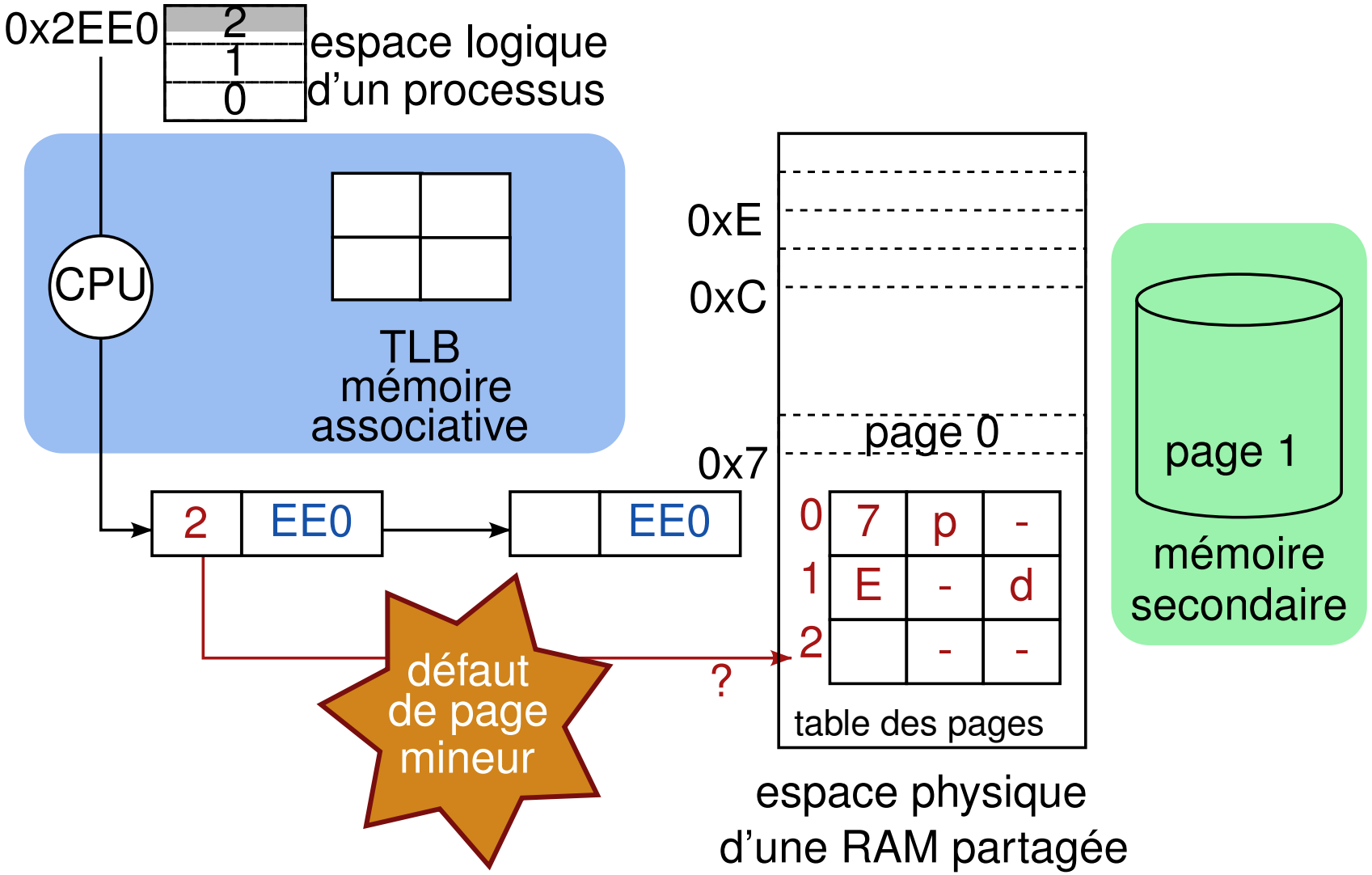
Cas 2 : pas de défaut de page, 2 accès mémoire



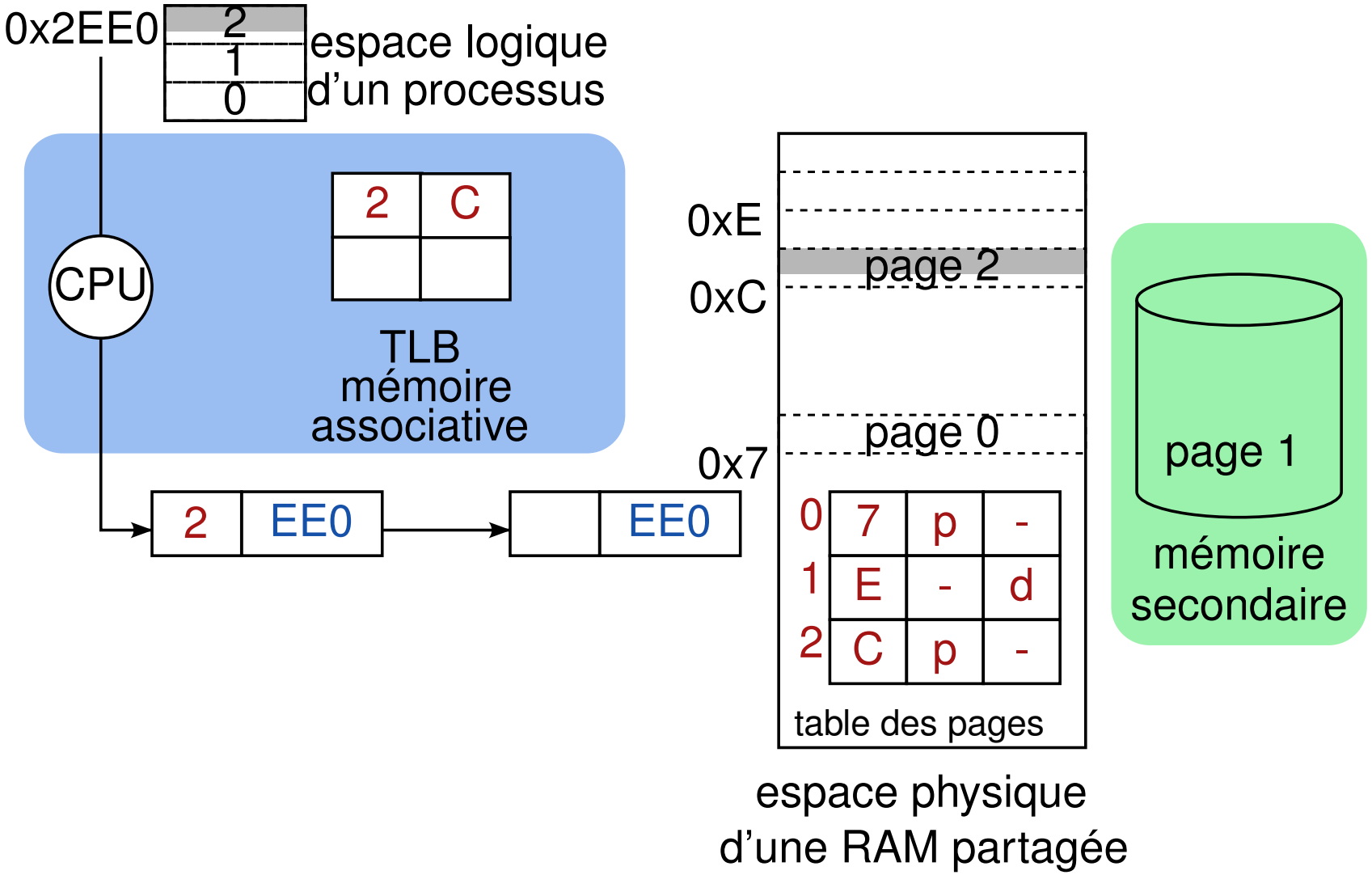
Cas 2 : pas de défaut de page, 2 accès mémoire



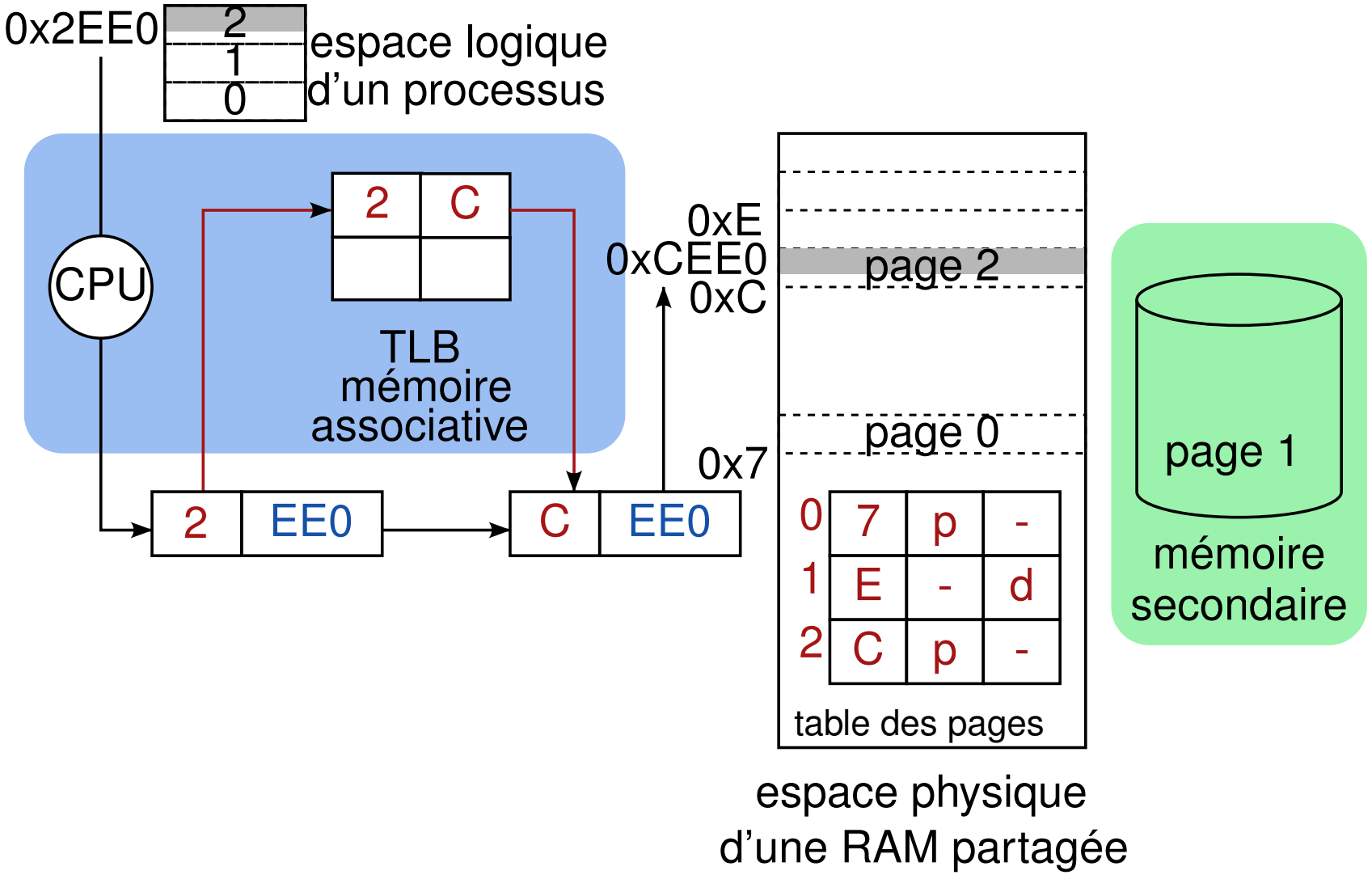
Cas 3 : `malloc()` ⇒ défaut de page mineur



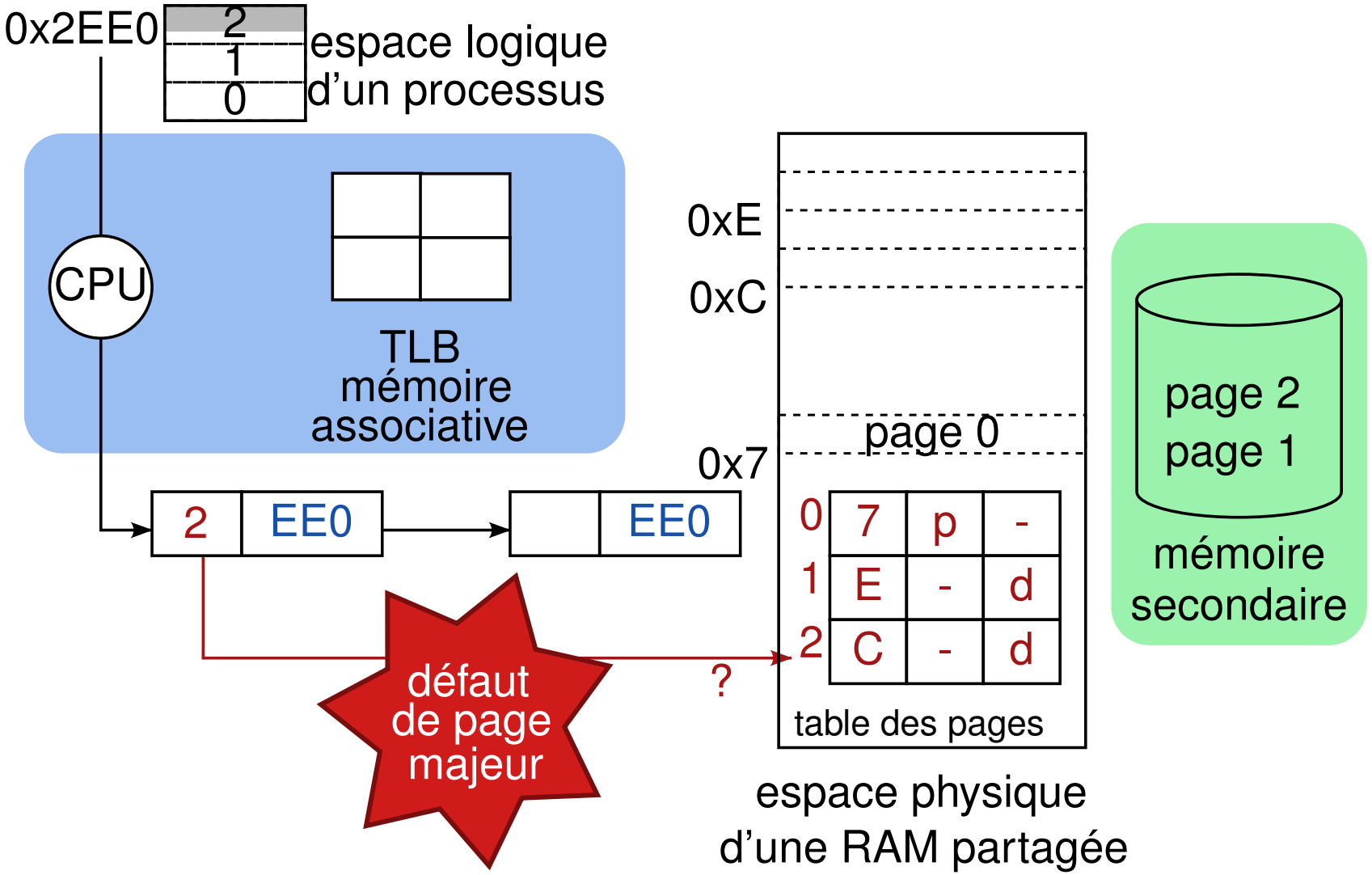
Cas 3 : `malloc()` ⇒ défaut de page mineur



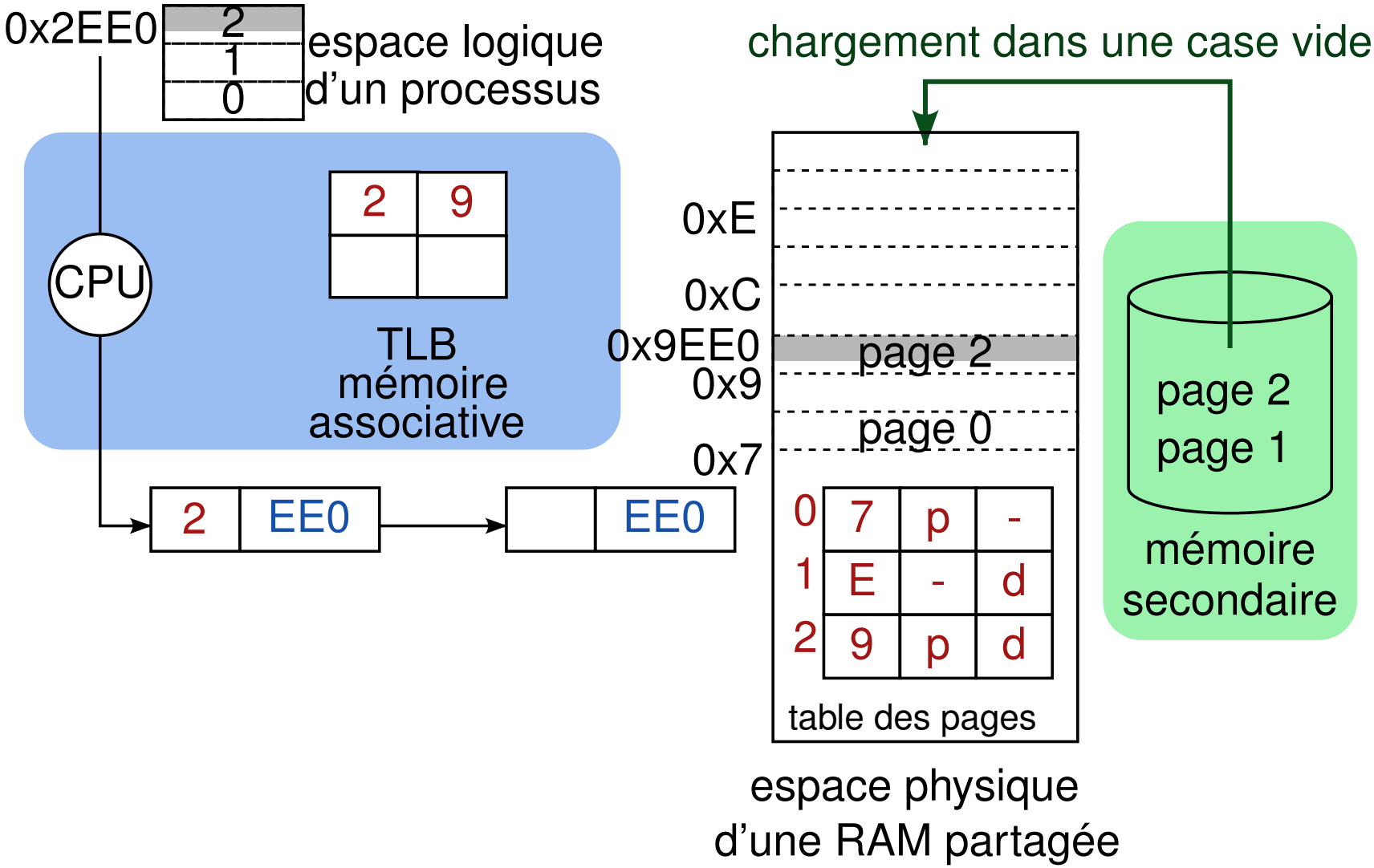
Cas 3 : `malloc()` ⇒ défaut de page mineur



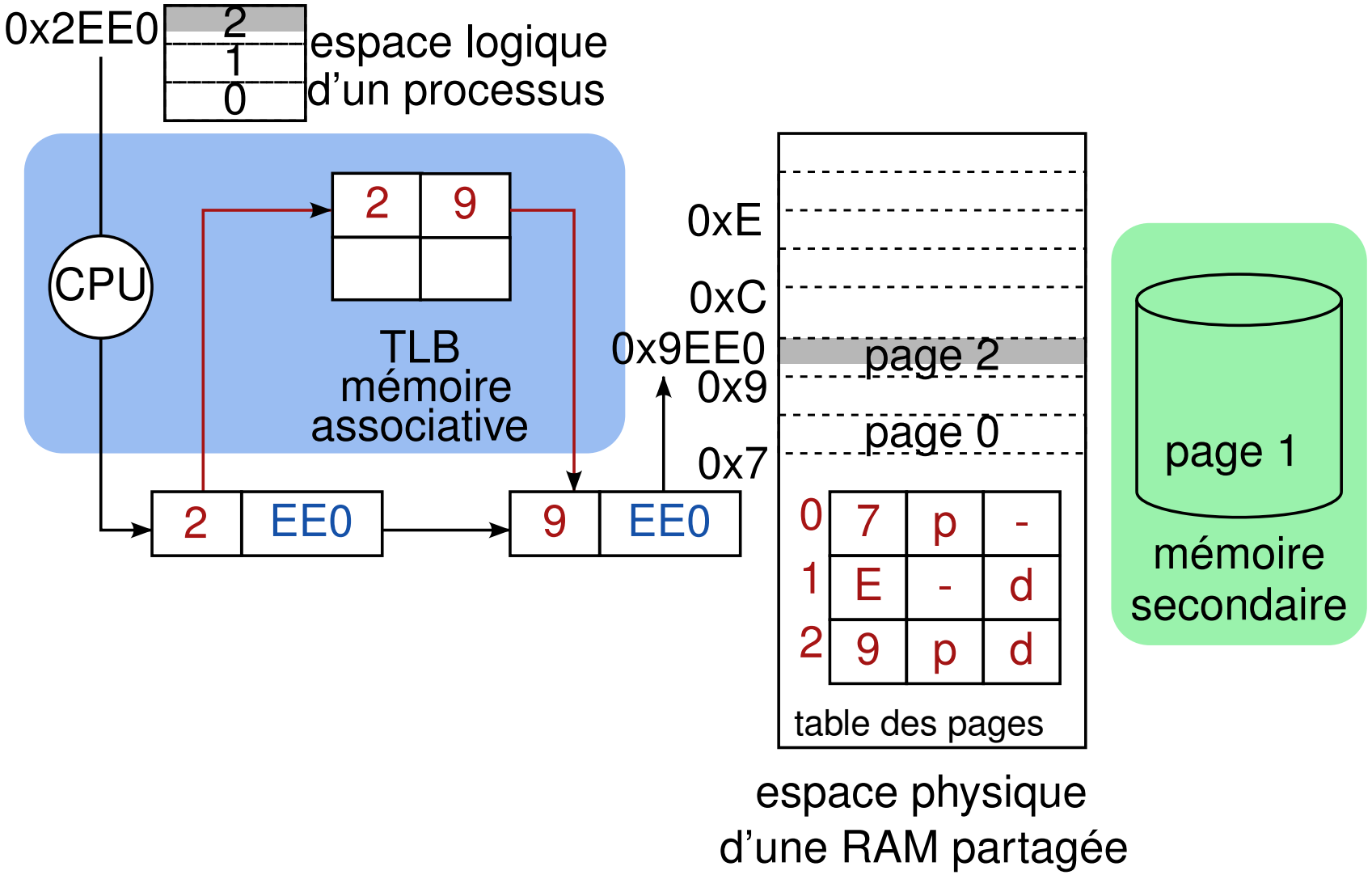
Cas 4 : page absente et dirty ⇒ défaut de page majeur



Cas 4 : page absente et dirty ⇒ défaut de page majeur



Cas 4 : page absente et dirty ⇒ défaut de page majeur



Défaut de page / défaut de cache

Mémoire cache (rappel)

- copie partielle de la RAM
- contient les dernières pages utilisées
- défaut de cache = donnée/page qu'il faut chercher depuis la RAM
  (à décliner sur les différents niveaux de cache L1, L2, L3. . .)

Pour le programmeur : minimiser . . .

- les risques de ddp majeurs (accès à la mém. secondaire)
- les risques de défaut de cache (utilisation contre-productive)
- la taille de la **chaîne de référence** du prog. (en 1ère approx.)

Chaîne de référence d'un programme

suite des numéros de pages utilisées au cours de son exécution

Défaut de page / défaut de cache

Mémoire cache (rappel)

- copie partielle de la RAM
- contient les dernières pages utilisées
- défaut de cache = donnée/page qu'il faut chercher depuis la RAM
  (à décliner sur les différents niveaux de cache L1, L2, L3...)

Pour le programmeur : minimiser ...

- les risques de ddp majeurs (accès à la mém. secondaire)
- les risques de défaut de cache (utilisation contre-productive)
- la taille de la **chaîne de référence** du prog. (en 1ère approx.)

Chaîne de référence d'un programme

suite des numéros de pages utilisées au cours de son exécution

Défaut de page / défaut de cache

Mémoire cache (rappel)

- copie partielle de la RAM
- contient les dernières pages utilisées
- défaut de cache = donnée/page qu'il faut chercher depuis la RAM
  (à décliner sur les différents niveaux de cache L1, L2, L3...)

Pour le programmeur : minimiser ...

- les risques de ddp majeurs (accès à la mém. secondaire)
- les risques de défaut de cache (utilisation contre-productive)
- la taille de la **chaîne de référence** du prog. (en 1ère approx.)

Chaîne de référence d'un programme

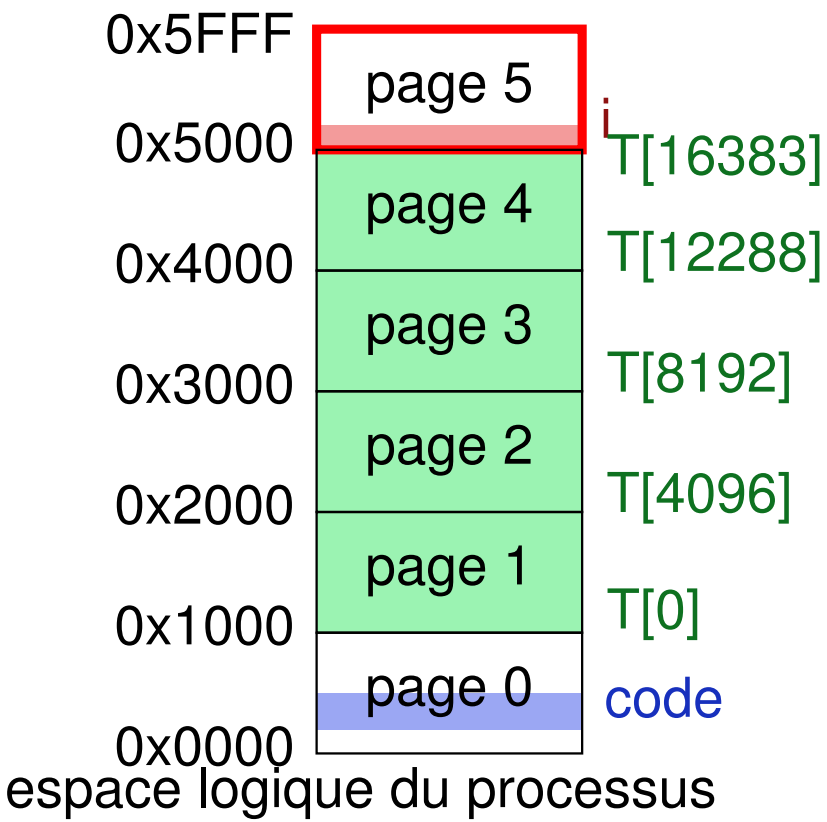
suite des numéros de pages utilisées au cours de son exécution

Chaîne de référence

Fichier prog.c

```
#define N 16384
char T[N];
int i;

void main (void)
{
   for (i=0;i<N;i++)
      T[i]='*';
}
```



Chaîne de référence de ce programme (ignorons le code en page 0)

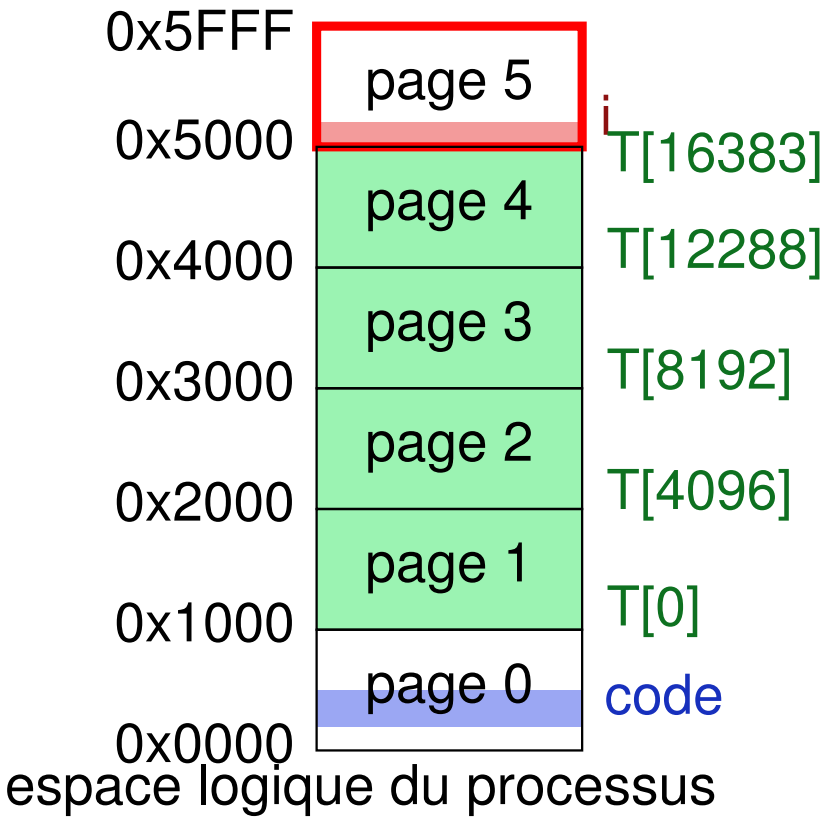
$\omega = 5(515)^{4096}(525)^{4096}(535)^{4096}(545)^{4096}5$ i=0

Chaîne de référence

Fichier prog.c

```
#define N 16384
char T[N];
int i;

void main (void)
{
   for (i=0;i<N;i++)
      T[i]='*';
}
```



Chaîne de référence de ce programme (ignorons le code en page 0)

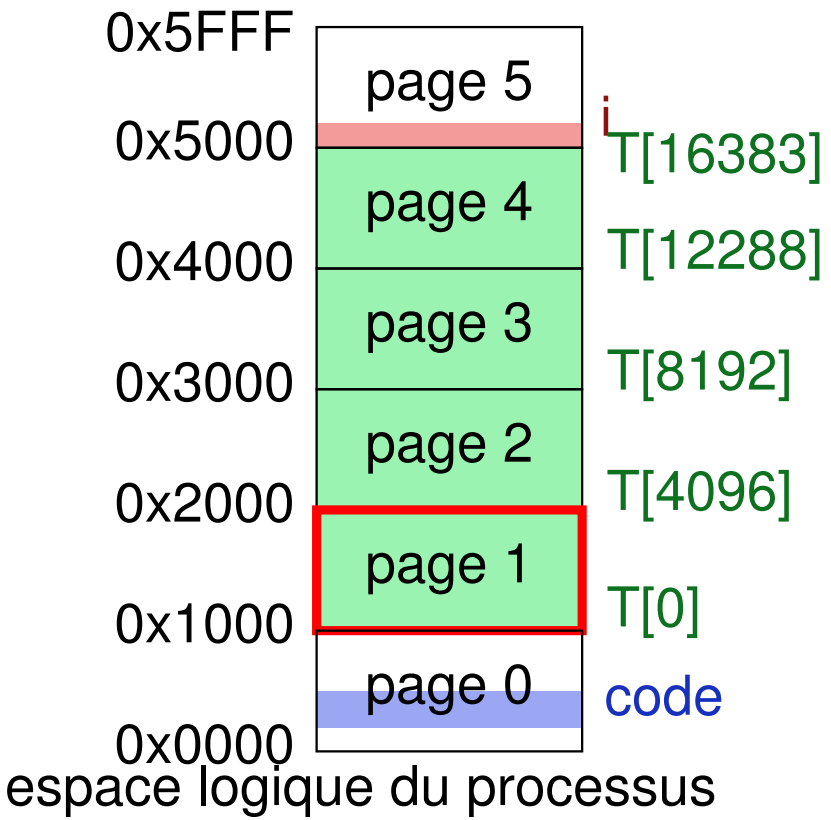
$\omega = 5(515)^{4096}(525)^{4096}(535)^{4096}(545)^{4096}5$ i=0

Chaîne de référence

Fichier prog.c

```
#define N 16384
char T[N];
int i;

void main (void)
{
   for (i=0;i<N;i++)
      T[i]='*';
}
```



Chaîne de référence de ce programme (ignorons le code en page 0)

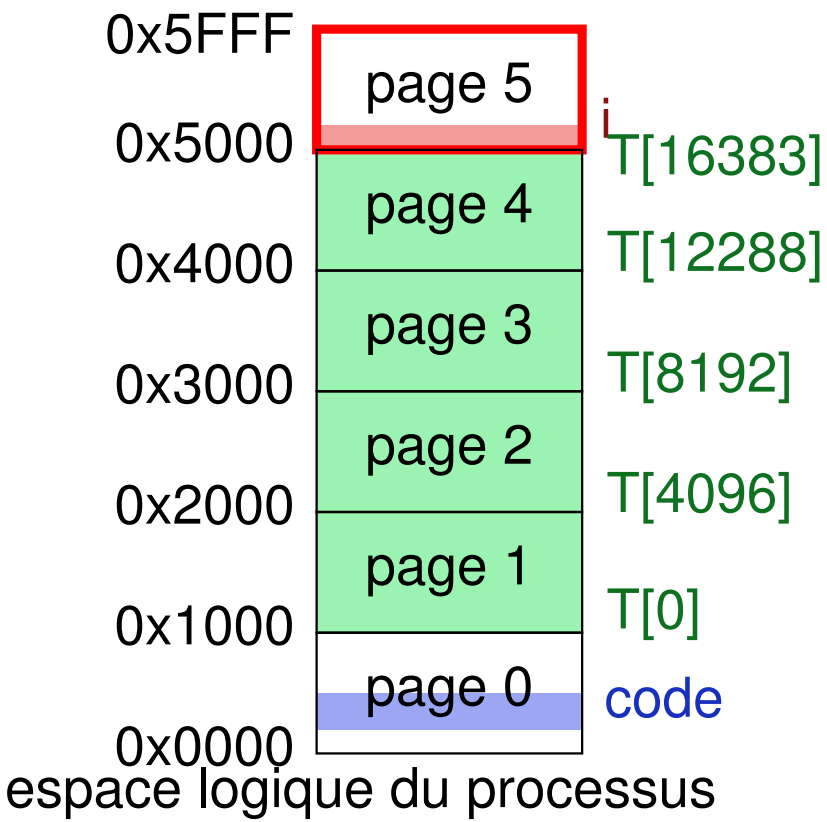
$\omega = 5(515)^{4096}(525)^{4096}(535)^{4096}(545)^{4096}5$ i=0

Chaîne de référence

Fichier prog.c

```
#define N 16384
char T[N];
int i;

void main (void)
{
   for (i=0;i<N;i++)
      T[i]='*';
}
```



Chaîne de référence de ce programme (ignorons le code en page 0)

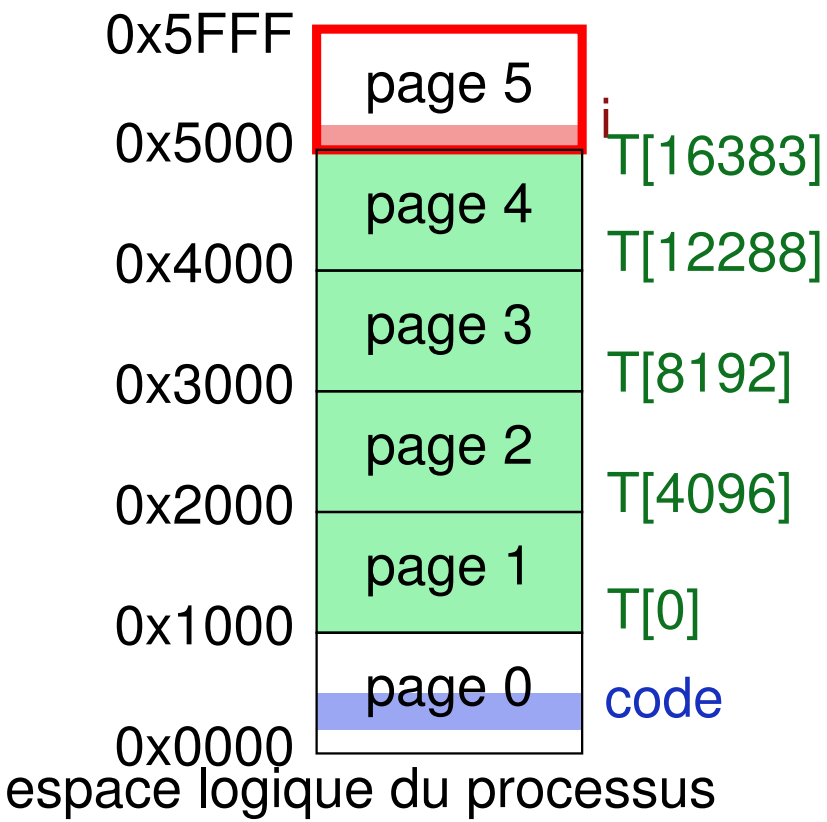
$\omega = 5(515)^{4096}(525)^{4096}(535)^{4096}(545)^{4096}5$ i=1

Chaîne de référence

Fichier prog.c

```
#define N 16384
char T[N];
int i;

void main (void)
{
   for (i=0;i<N;i++)
      T[i]='*';
}
```



Chaîne de référence de ce programme (ignorons le code en page 0)

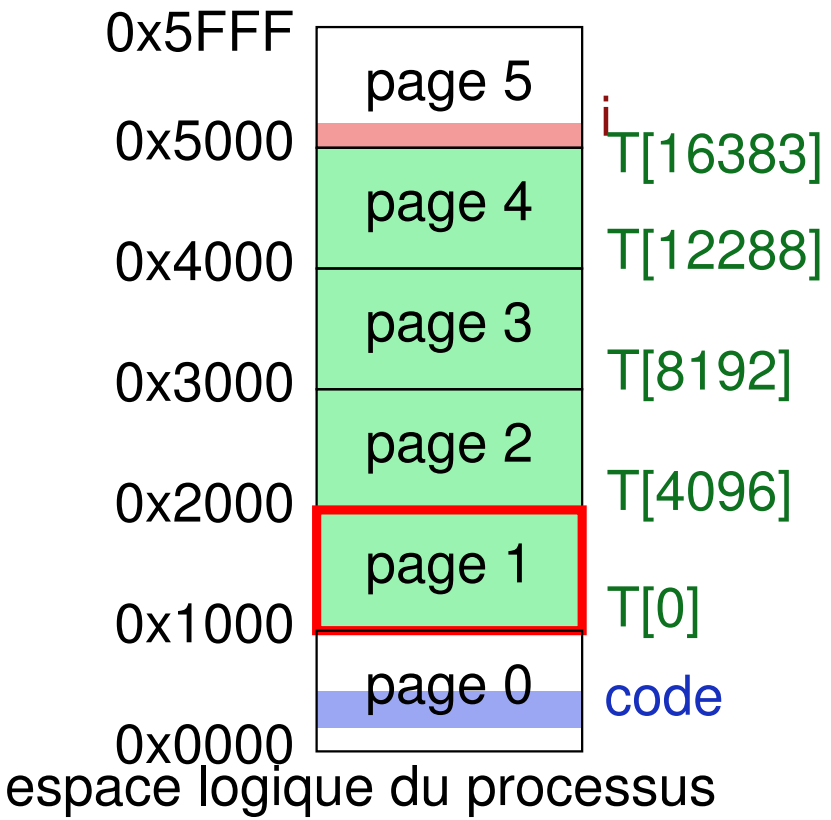
$\omega = 5(515)^{4096}(525)^{4096}(535)^{4096}(545)^{4096}5$ i=1

Chaîne de référence

Fichier prog.c

```
#define N 16384
char T[N];
int i;

void main (void)
{
   for (i=0;i<N;i++)
      T[i]='*';
}
```



Chaîne de référence de ce programme (ignorons le code en page 0)

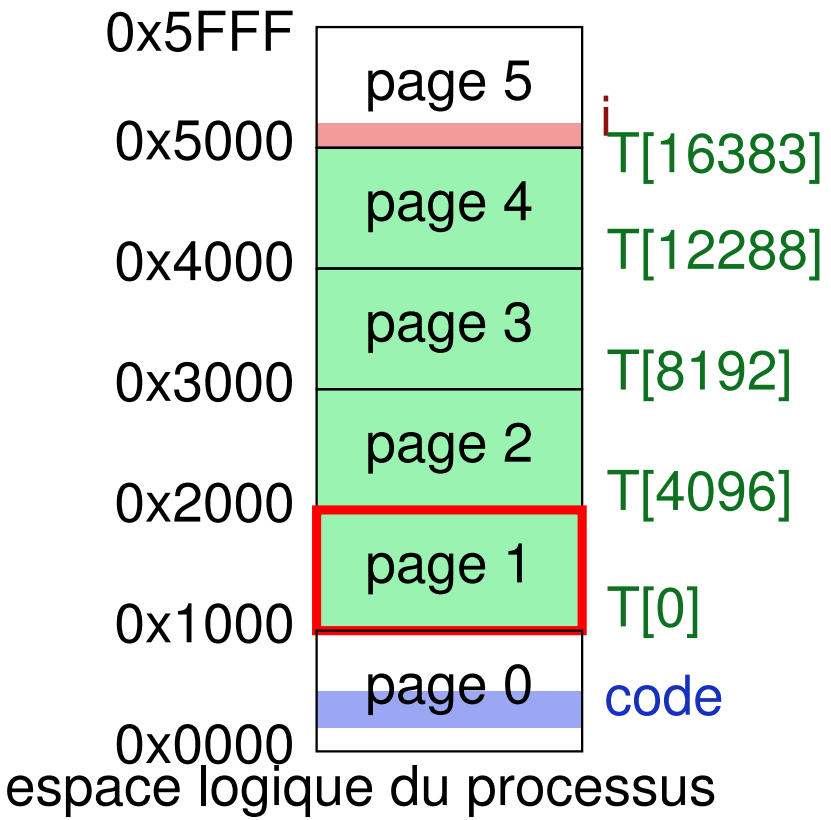
$\omega = 5(515)^{4096}(525)^{4096}(535)^{4096}(545)^{4096}5$ i=1

Chaîne de référence

Fichier prog.c

```
#define N 16384
char T[N];
int i;

void main (void)
{
   for (i=0;i<N;i++)
      T[i]='*';
}
```



Chaîne de référence de ce programme (ignorons le code en page 0)

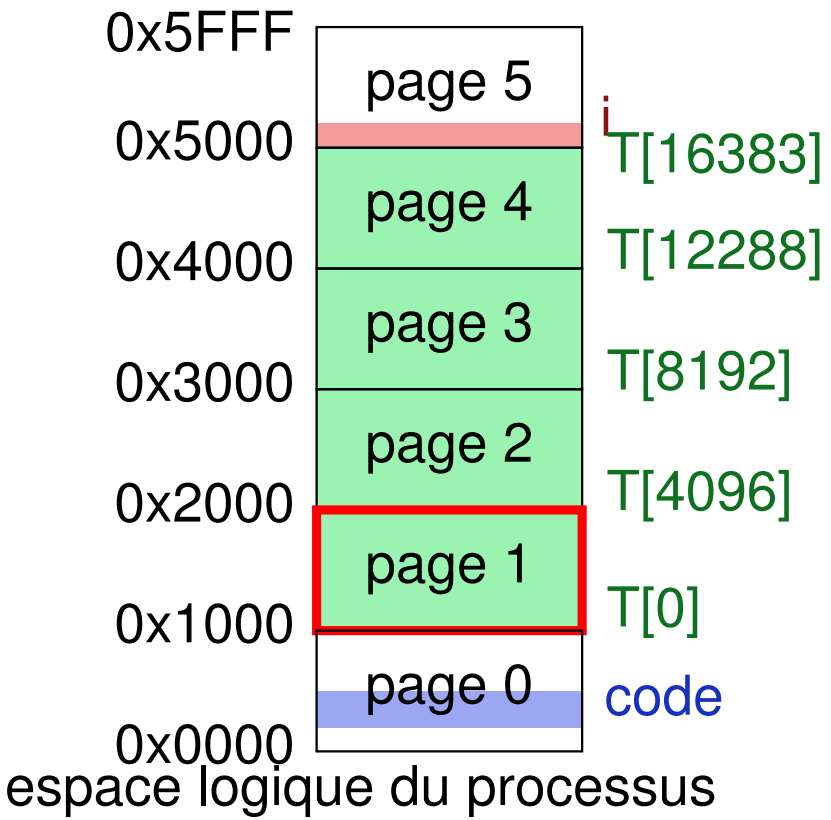
$\omega = 5(515)^{4096}(525)^{4096}(535)^{4096}(545)^{4096}5$ i=2

Chaîne de référence

Fichier prog.c

```
#define N 16384
char T[N];
int i;

void main (void)
{
   for (i=0;i<N;i++)
      T[i]='*';
}
```



Chaîne de référence de ce programme (ignorons le code en page 0)

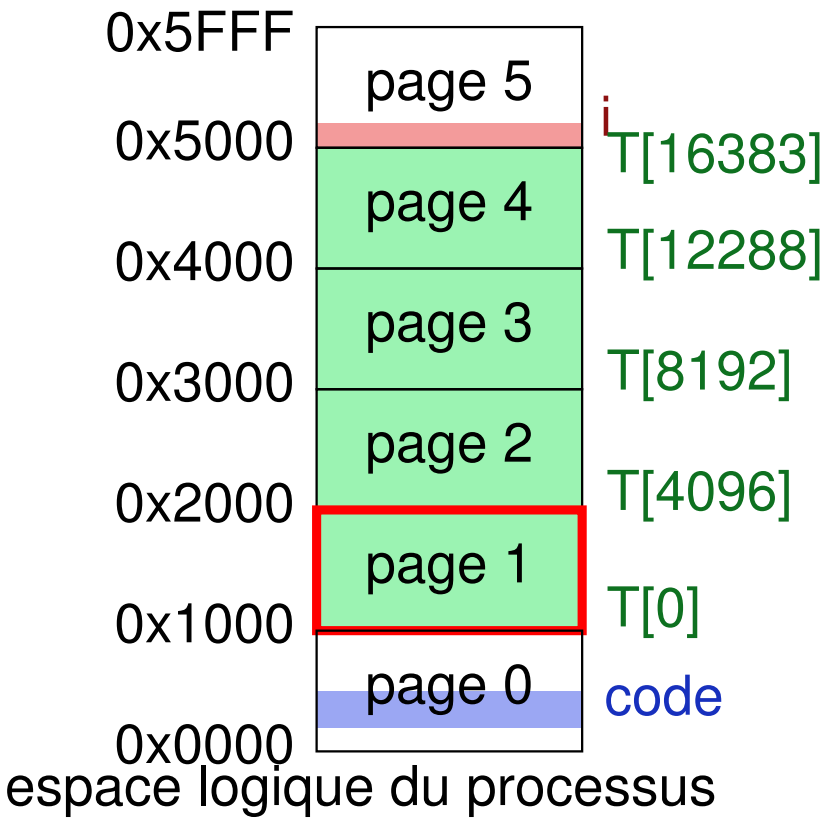
$\omega = 5(515)^{4096}(525)^{4096}(535)^{4096}(545)^{4096}5$ i=3

Chaîne de référence

Fichier prog.c

```
#define N 16384
char T[N];
int i;

void main (void)
{
   for (i=0;i<N;i++)
      T[i]='*';
}
```



Chaîne de référence de ce programme (ignorons le code en page 0)

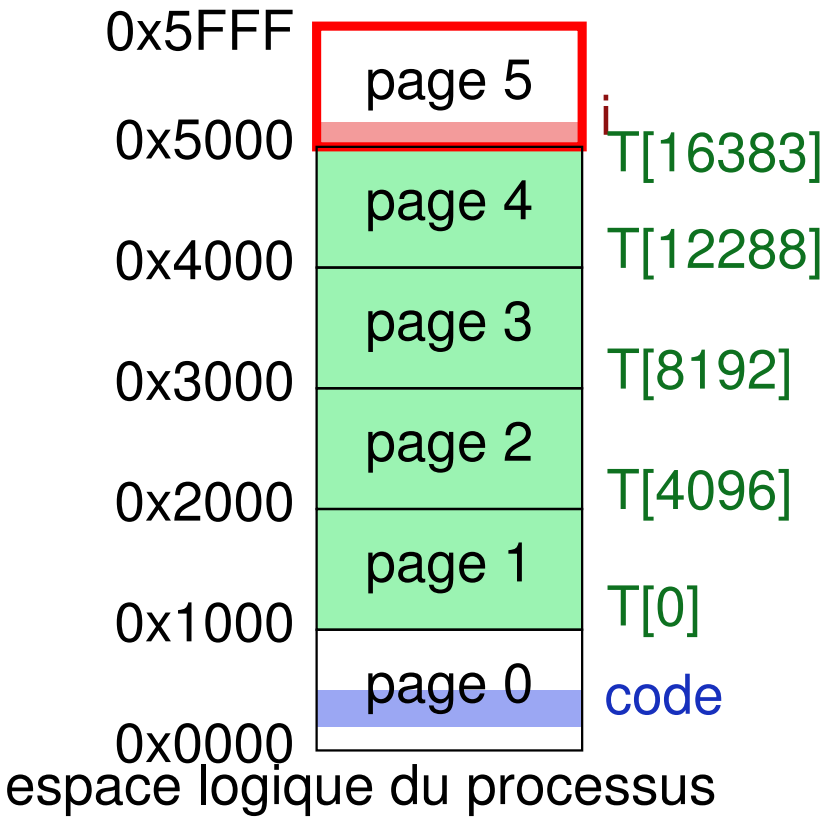
$\omega = 5(515)^{4096}(525)^{4096}(535)^{4096}(545)^{4096}5$ i=4095

Chaîne de référence

Fichier prog.c

```
#define N 16384
char T[N];
int i;

void main (void)
{
   for (i=0;i<N;i++)
      T[i]='*';
}
```



Chaîne de référence de ce programme (ignorons le code en page 0)

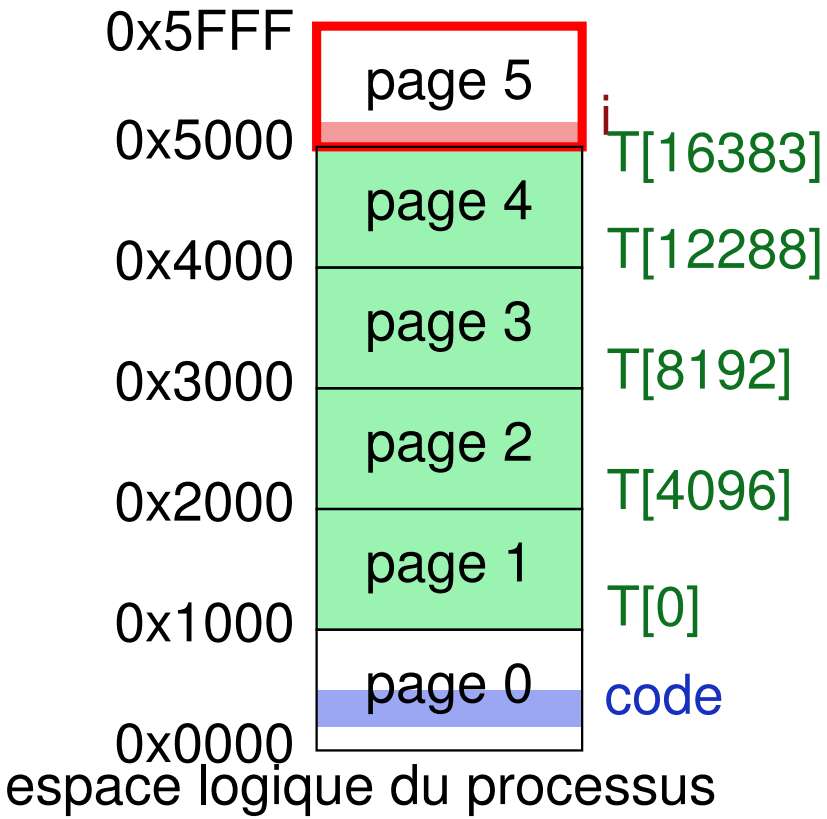
$\omega = 5(515)^{4096}(525)^{4096}(535)^{4096}(545)^{4096}5$ i= 4096

Chaîne de référence

Fichier prog.c

```
#define N 16384
char T[N];
int i;

void main (void)
{
   for (i=0;i<N;i++)
      T[i]='*';
}
```



Chaîne de référence de ce programme (ignorons le code en page 0)

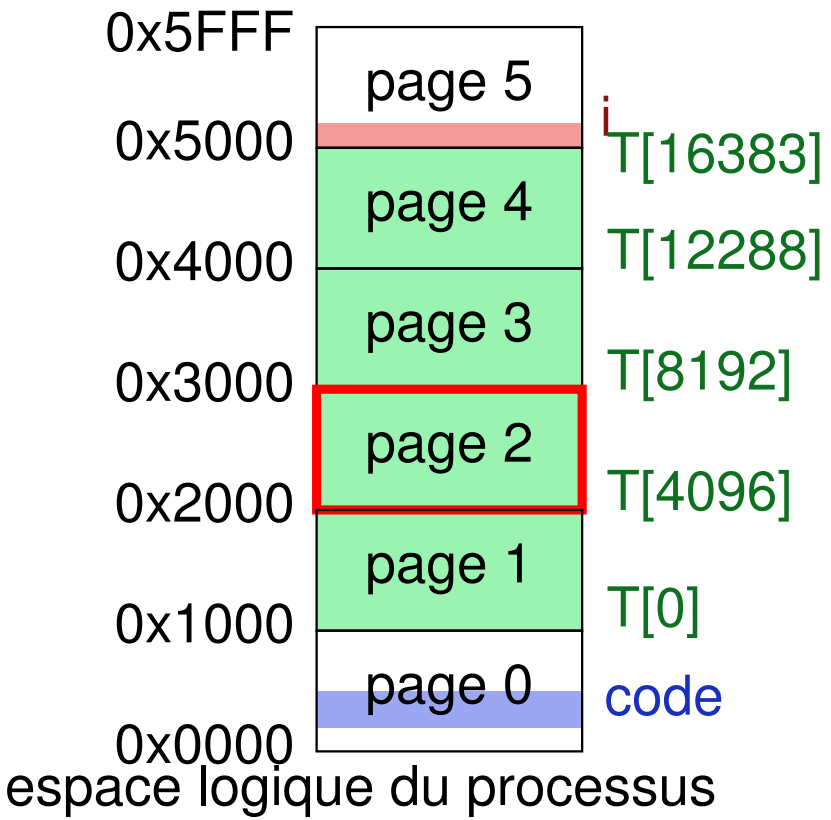
$\omega = 5(515)^{4096}(525)^{4096}(535)^{4096}(545)^{4096}5$ i= 4096

Chaîne de référence

Fichier prog.c

```
#define N 16384
char T[N];
int i;

void main (void)
{
   for (i=0;i<N;i++)
      T[i]='*';
}
```



Chaîne de référence de ce programme (ignorons le code en page 0)

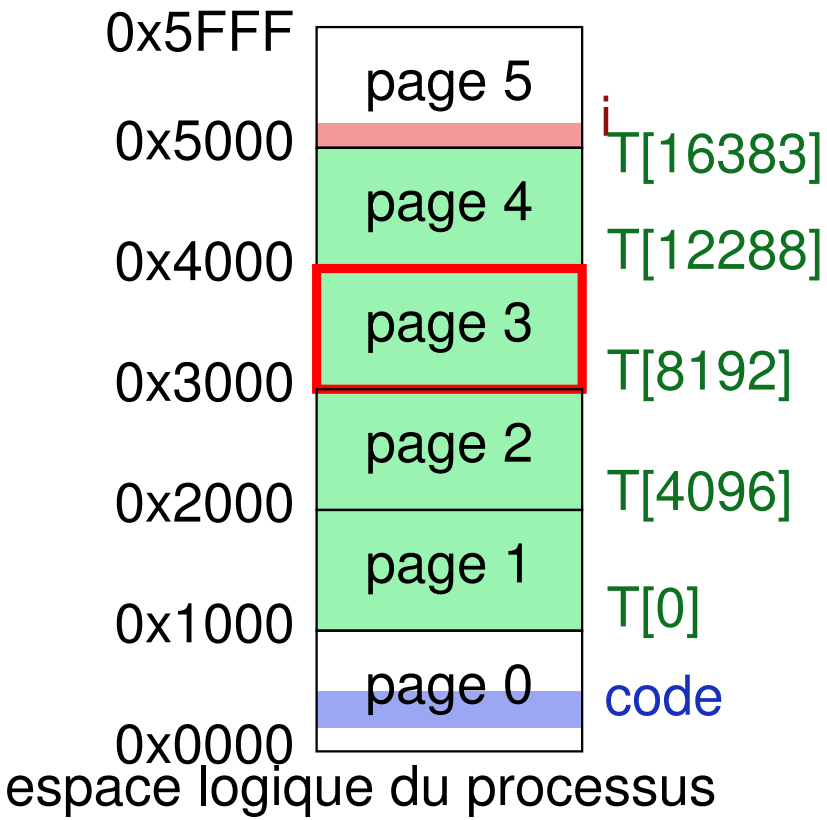
$\omega = 5(515)^{4096}(525)^{4096}(535)^{4096}(545)^{4096}5$ i= 4096

Chaîne de référence

Fichier prog.c

```
#define N 16384
char T[N];
int i;

void main (void)
{
   for (i=0;i<N;i++)
      T[i]='*';
}
```



Chaîne de référence de ce programme (ignorons le code en page 0)

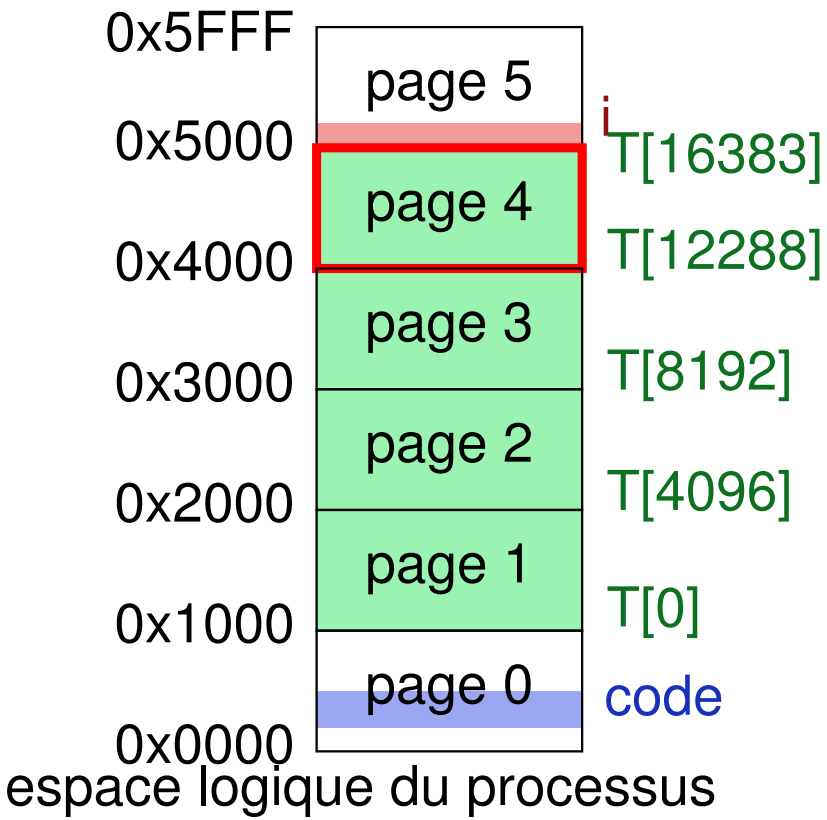
$\omega = 5(515)^{4096}(525)^{4096}(535)^{4096}(545)^{4096}5$ i= 8192

Chaîne de référence

Fichier prog.c

```
#define N 16384
char T[N];
int i;

void main (void)
{
   for (i=0;i<N;i++)
      T[i]='*';
}
```



Chaîne de référence de ce programme (ignorons le code en page 0)

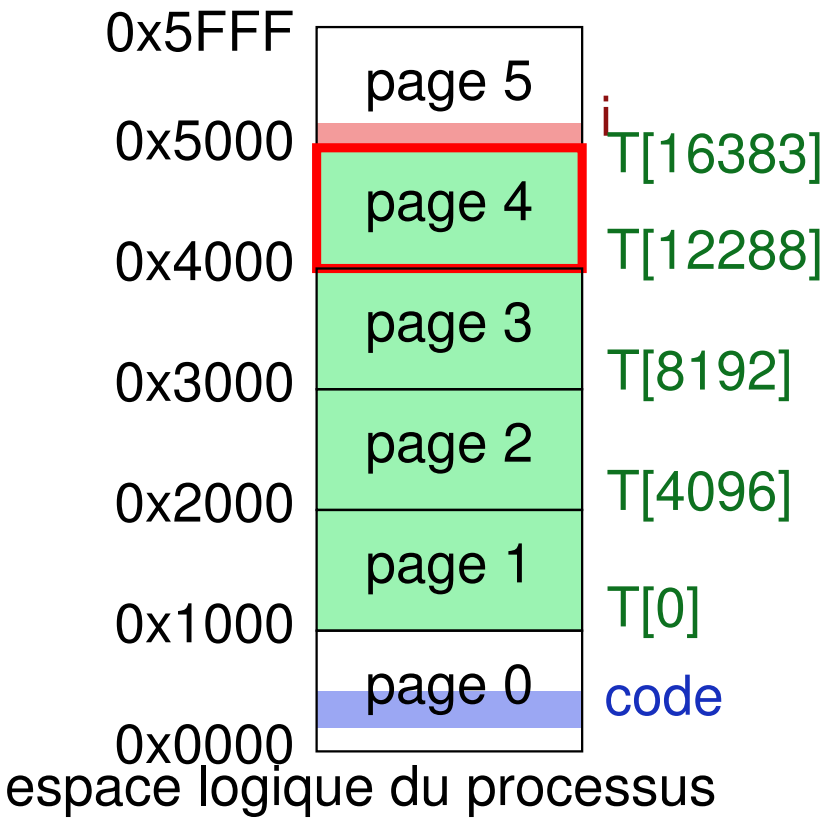
$\omega = 5(515)^{4096}(525)^{4096}(535)^{4096}(545)^{4096}5$ i= 12288

Chaîne de référence

Fichier prog.c

```
#define N 16384
char T[N];
int i;

void main (void)
{
   for (i=0;i<N;i++)
      T[i]='*';
}
```



Chaîne de référence de ce programme (ignorons le code en page 0)

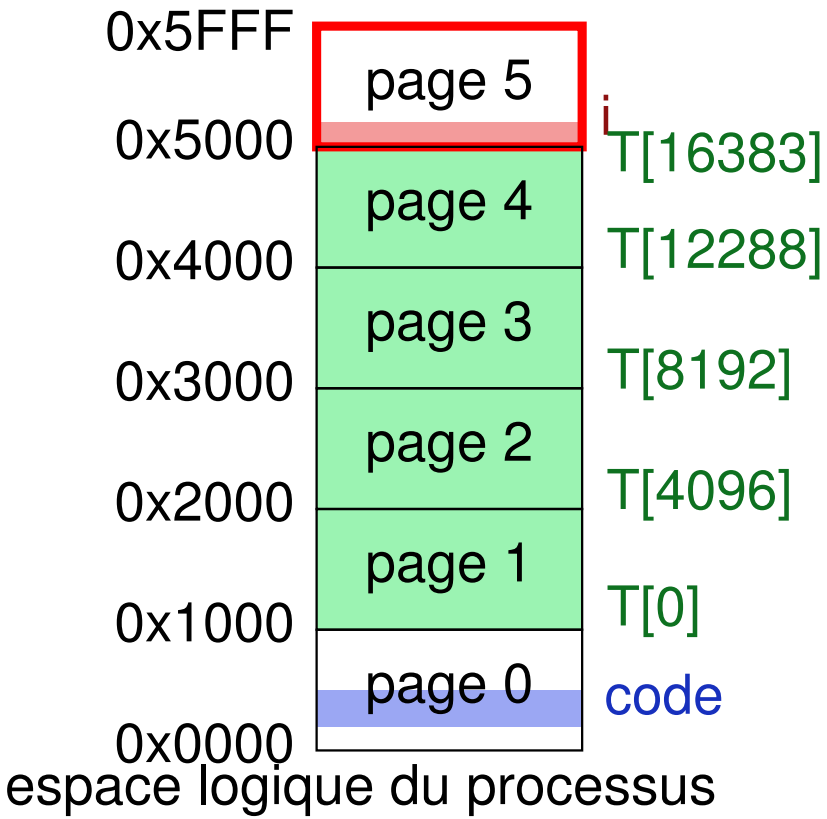
$\omega = 5(515)^{4096}(525)^{4096}(535)^{4096}(545)^{4096}5$ i= 16383

Chaîne de référence

Fichier prog.c

```
#define N 16384
char T[N];
int i;

void main (void)
{
   for (i=0;i<N;i++)
      T[i]='*';
}
```



Chaîne de référence de ce programme (ignorons le code en page 0)

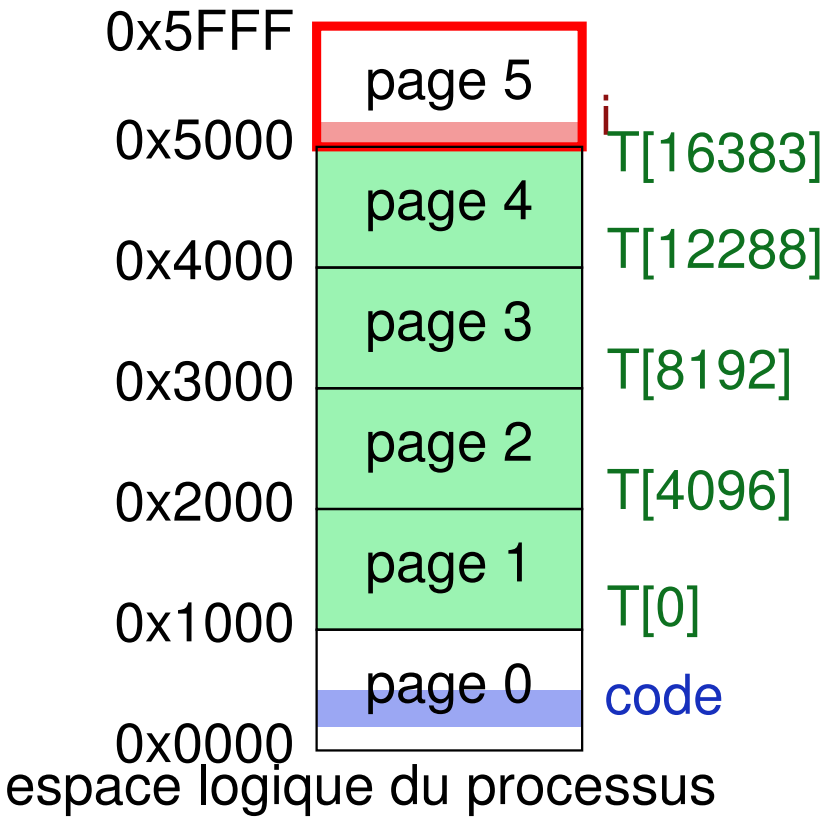
$\omega = 5(515)^{4096}(525)^{4096}(535)^{4096}(545)^{4096}5$ i= 16384

Chaîne de référence

Fichier prog.c

```
#define N 16384
char T[N];
int i;

void main (void)
{
   for (i=0;i<N;i++)
      T[i]='*';
}
```



Chaîne de référence de ce programme (ignorons le code en page 0)

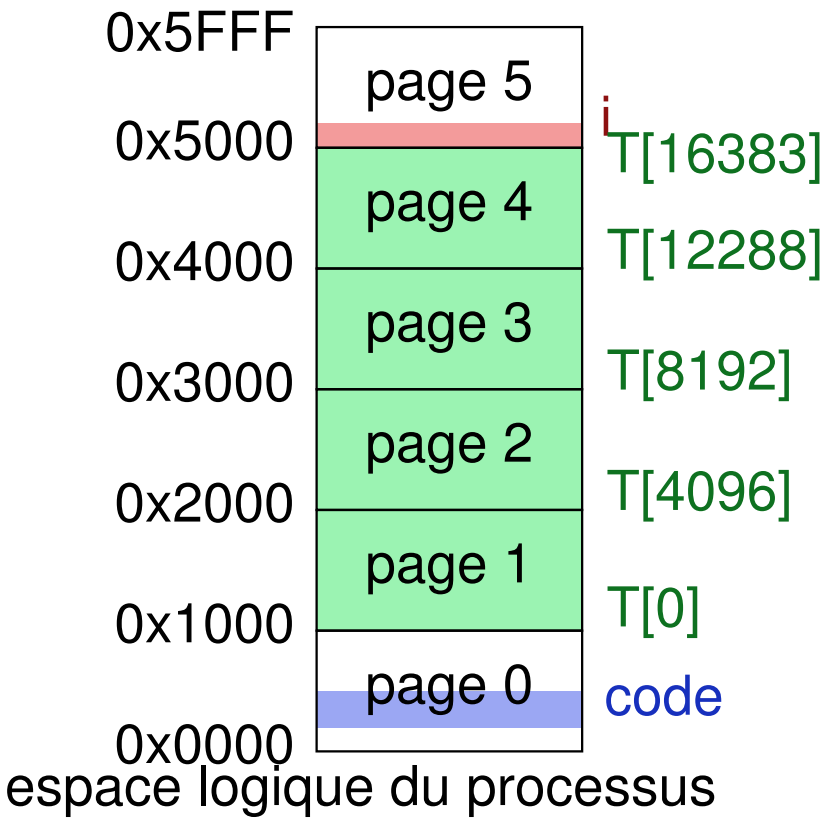
$\omega = 5(515)^{4096}(525)^{4096}(535)^{4096}(545)^{4096}5$ i= 16384

Chaîne de référence

Fichier prog.c

```
#define N 16384
char T[N];
int i;

void main (void)
{
   for (i=0;i<N;i++)
      T[i]='*';
}
```



Chaîne de référence de ce programme (ignorons le code en page 0)

$\omega = 5(15)^{4096}(25)^{4096}(35)^{4096}(45)^{4096}$

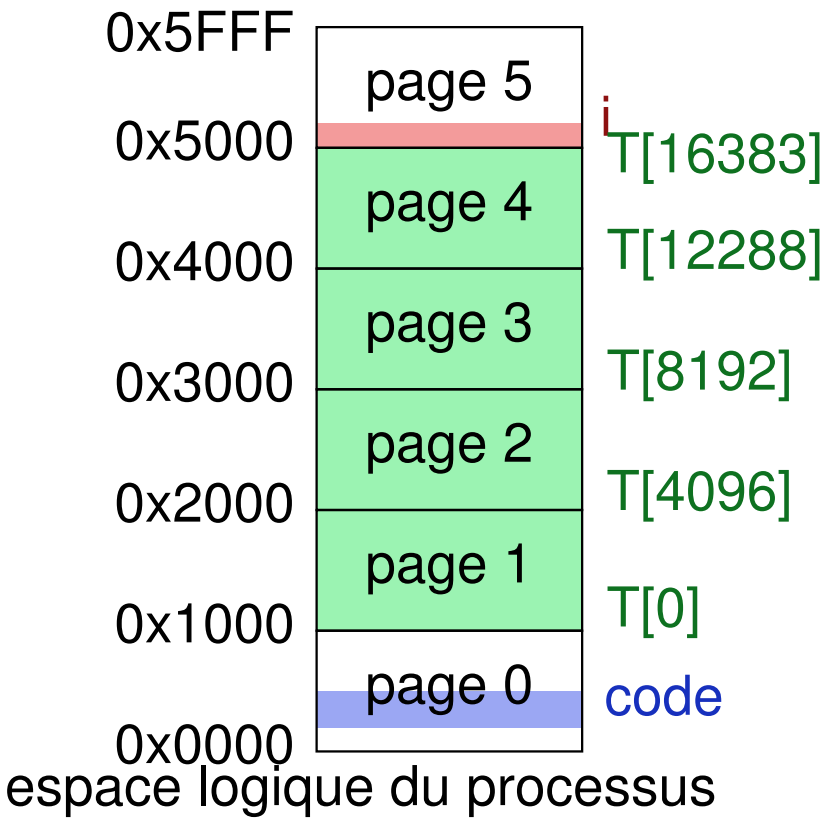
Chaîne de référence

TD

Fichier prog.c

```
#define N 16384
char T[N];
int i;

void main (void)
{
   for (i=0;i<N;i++)
     T[i]='*';
}
```



Chaîne de référence de ce programme (ignorons la page de code 0)

$\omega = 5(15)^{4096}(25)^{4096}(35)^{4096}(45)^{4096}$: 32 769 ddp si 1 case libre

: 5 ddp si 5 cases libres

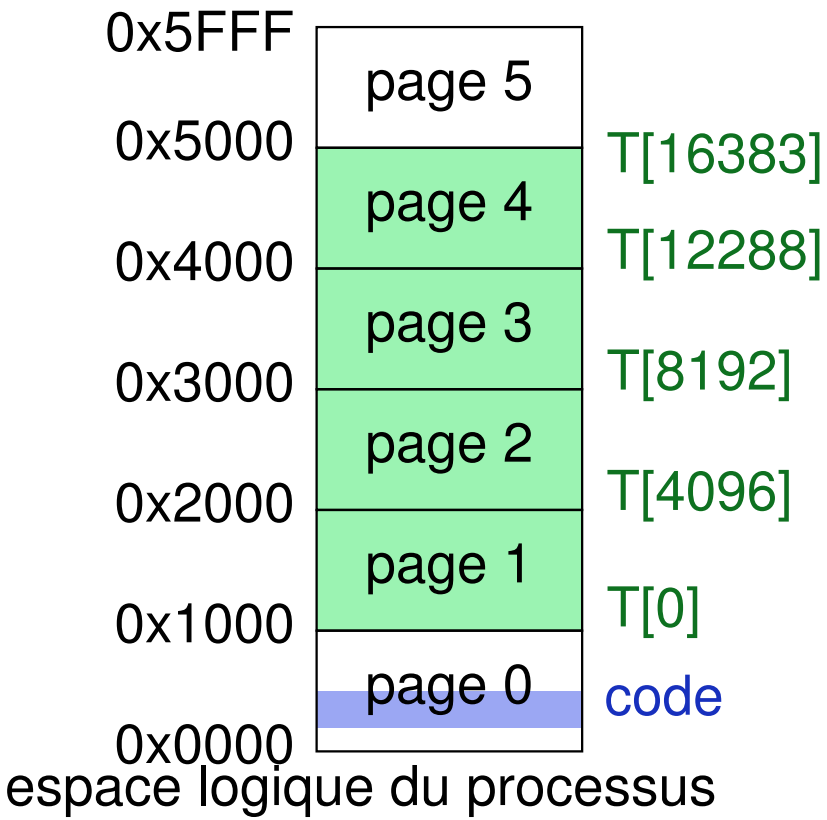
Chaîne de référence

TD

Fichier prog.c

```
#define N 16384
char T[N];
register int i;

void main (void)
{
   for (i=0;i<N;i++)
      T[i]='*';
}
```



Chaîne de référence de ce programme (ignorons la page de code 0)

$\omega = (1)^{4096}(2)^{4096}(3)^{4096}(4)^{4096} = 1234$: 4 ddp $\forall$ nb cases libres

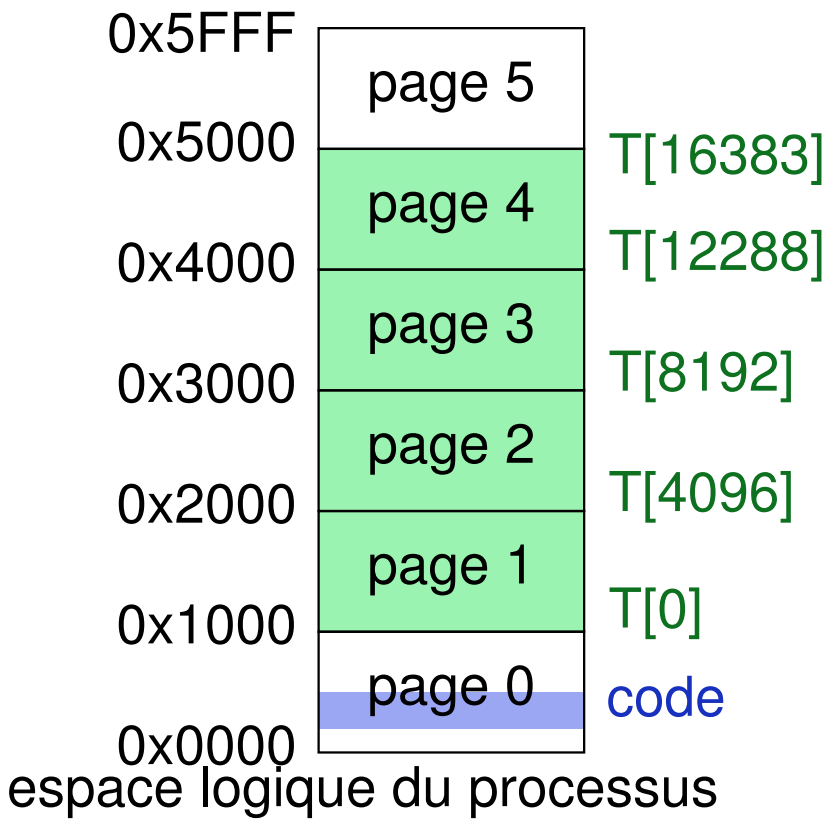
Chaîne de référence

TD

Fichier prog.c

```
#define N 16384
char T[N];
register int i;

void main (void)
{
   for (i=0;i<4096;i++)
      for (j=0;j<4;j++)
         T[i+(j*4096)]='*';
}
```



Chaîne de référence de ce programme (ignorons la page de code 0)

$\omega = (1234)^{4096}$: 16 384 ddp si 1 case libre

: 4 ddp si 4 cases libres

Organisation du cours

1. Système d'exploitation
2. Processus
3. Partage des ressources
4. Système de Gestion de Fichiers
5. Entrées/Sorties

4. Système de Gestion de Fichiers

4.1. Révisions [R1.04]

- Mémoire secondaire
- Arborescence de fichiers et répertoires
- Attributs d'un fichier/répertoire (métadonnées)

4.2. SGF ext4

- Structures de données : i-nœud et datablocks
- Protection

4.3. Accès aux fichiers par les processus

4.4. Une arborescence mais des supports et SGF variés

4. Système de Gestion de Fichiers

4.1. Révisions [R1.04]

- Mémoire secondaire
- Arborescence de fichiers et répertoires
- Attributs d'un fichier/répertoire (métadonnées)

4.2. SGF ext4

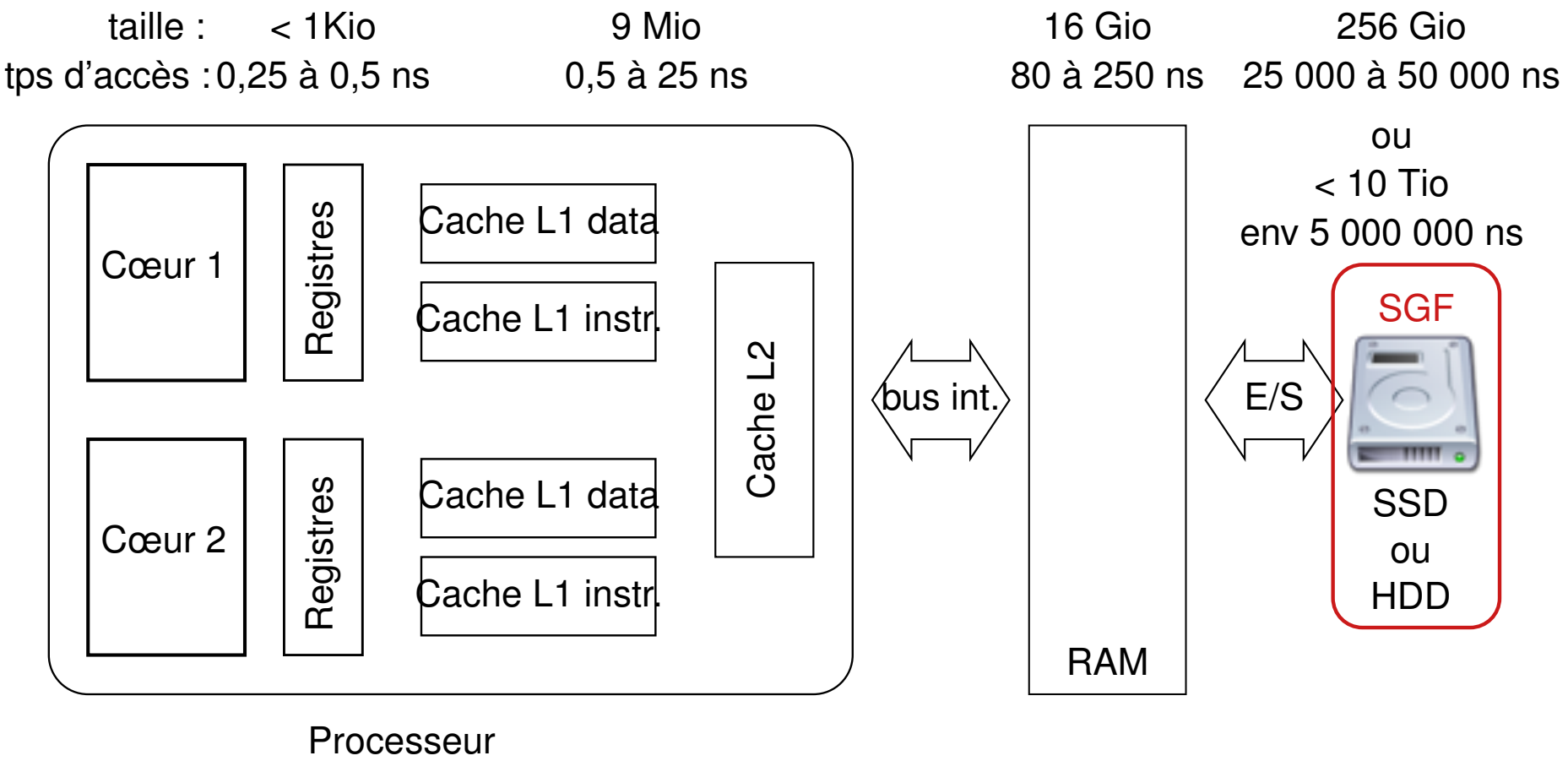
- Structures de données : i-nœud et datablocks
- Protection

4.3. Accès aux fichiers par les processus

4.4. Une arborescence mais des supports et SGF variés

Composants - Mémoires (Rappel)

Exemple de hiérarchie



Mémoire secondaire : périphérique de stockage

Types de périphériques

- HDD : disque durs magnétiques,
  grosse capacité
  mais lents (dépl. mécaniques)
- SSD : mémoire flash,
  plus rapide (grille de cellules)
  mais plus petite capacité
  et nb. écritures limité sur chaque cellule

Types d'interface

- SATA : pour HDD et anciens SSD
- NVMe : pour nouveaux SSD plus rapides
- USB : pour les périphériques externes (HDD ou SSD)

Partition

Un périphérique de stockage est généralement découpé en partitions

Partition *bootable*

- section de boot *(boot block)* : pour démarrer l'OS présent sur la partition *(chargé en RAM par le bootloader)*
- le reste : attributs et données des fichiers et répertoires

Fichiers et répertoires

- **Fichiers** : suite d'octets représentant un texte (selon un certain encodage), un programme exécutable, des données binaires, ...
- **Répertoires ou dossiers** (directory) : contiennent des fichiers et/ou des répertoires $\Rightarrow$ un arbre

Partition

Un périphérique de stockage est généralement découpé en partitions

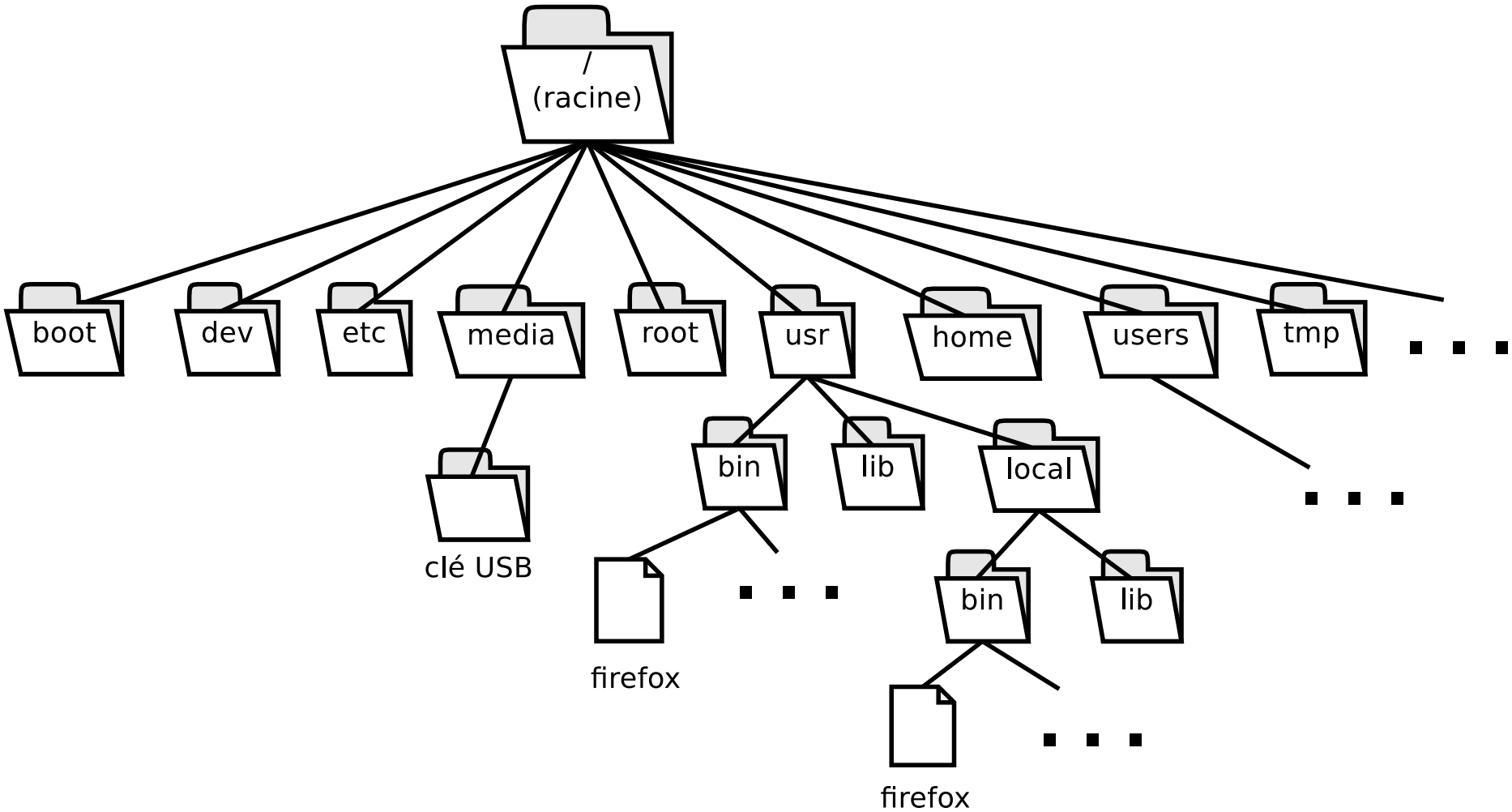
Partition *bootable*

- section de boot *(boot block)* : pour démarrer l'OS présent sur la partition *(chargé en RAM par le bootloader)*
- le reste : attributs et données des fichiers et répertoires

Fichiers et répertoires

- **Fichiers** : suite d'octets représentant un texte (selon un certain encodage), un programme exécutable, des données binaires, ...
- **Répertoires ou dossiers** (directory) : contiennent des fichiers et/ou des répertoires $\Rightarrow$ un arbre

Arborescence Linux standard



Standard : FHS *(Filesystem Hierarchy Standard)*

Attributs d'un fichier (métadonnées)

Informations sur le fichier

- Nom
- Type (fichiers, dossier, ...)
- Taille
- Date de dernière modification
- Propriétaire (UID)
- Groupe (GID)
- Permissions / droits d'accès
- ...

Définir un SGF

Pour chaque fichier, gérer

- les attributs de taille fixe : **i-nœud**
- et les données de taille variable : **extents de datablocks**

Enregistrer la structure arborescente des fichiers et répertoires

définition du contenu d'un **datablock de répertoire**

Proposer des solutions de protection

- superblock
- journal
- checksums

4. Système de Gestion de Fichiers

4.1. Révisions [R1.04]

- Mémoire secondaire
- Arborescence de fichiers et répertoires
- Attributs d'un fichier/répertoire (métadonnées)

4.2. SGF ext4

- Structures de données : i-nœud et datablocks
- Protection

4.3. Accès aux fichiers par les processus

4.4. Une arborescence mais des supports et SGF variés

Des SGF dédiés à des SE

Linux

- ext2 *Second Extended File system* (1993)
- ext3 : Linux 2.4.15 (2001) . . .
- ext4 : Linux 2.6.28 (2008) . . .
- mais aussi BTRFS, XFS, OpenZFS . . .

Windows OS

- FAT (12, 16, 32) *File Allocation Table* : DOS, Windows 3.x, 95, 98
- NTFS *New Technology File System* : NT, 2000, XP, Vista, 7, 8, 10
- ReFS *Resilient File System* : Windows Server 2012, 2016, . . .

Mac OS

- HFS *Hierarchical File System*
- HFS+ : Mac OS 8.1 (1998)
- APFS : Mac OS 10.13 (2017)

Des SGF dédiés à des SE

Linux

- ext2 *Second Extended File system* (1993)
- ext3 : Linux 2.4.15 (2001) . . .
- ext4 : Linux 2.6.28 (2008) . . .
- mais aussi BTRFS, XFS, OpenZFS . . .

Windows OS

- FAT (12, 16, 32) *File Allocation Table* : DOS, Windows 3.x, 95, 98
- NTFS *New Technology File System* : NT, 2000, XP, Vista, 7, 8, 10
- ReFS *Resilient File System* : Windows Server 2012, 2016, . . .

Mac OS

- HFS *Hierarchical File System*
- HFS+ : Mac OS 8.1 (1998)
- APFS : Mac OS 10.13 (2017)

Des SGF dédiés à des SE

Linux

- ext2 *Second Extended File system* (1993)
- ext3 : Linux 2.4.15 (2001) . . .
- ext4 : Linux 2.6.28 (2008) . . .
- mais aussi BTRFS, XFS, OpenZFS . . .

Windows OS

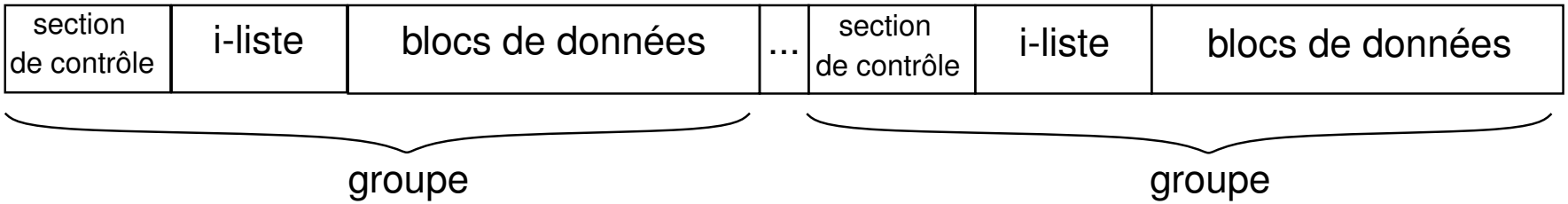
- FAT (12, 16, 32) *File Allocation Table* : DOS, Windows 3.x, 95, 98
- NTFS *New Technology File System* : NT, 2000, XP, Vista, 7, 8, 10
- ReFS *Resilient File System* : Windows Server 2012, 2016, . . .

Mac OS

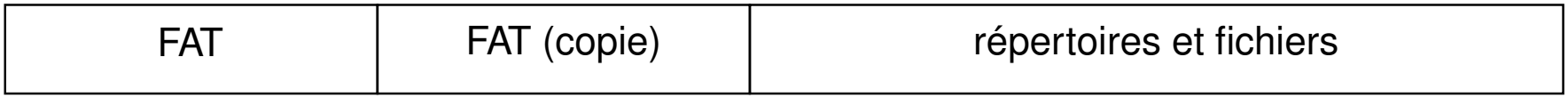
- HFS *Hierarchical File System*
- HFS+ : Mac OS 8.1 (1998)
- APFS : Mac OS 10.13 (2017)

Organisation physique d'un SGF dans une partition

- contenu pour extfs =

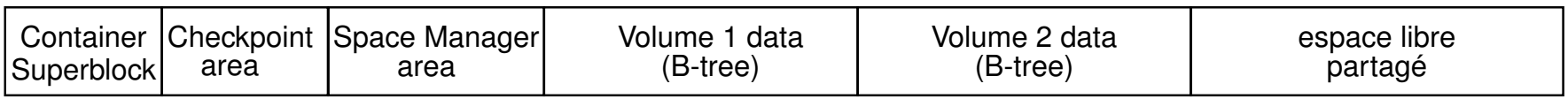


- contenu pour FAT =



- contenu pour NTFS =

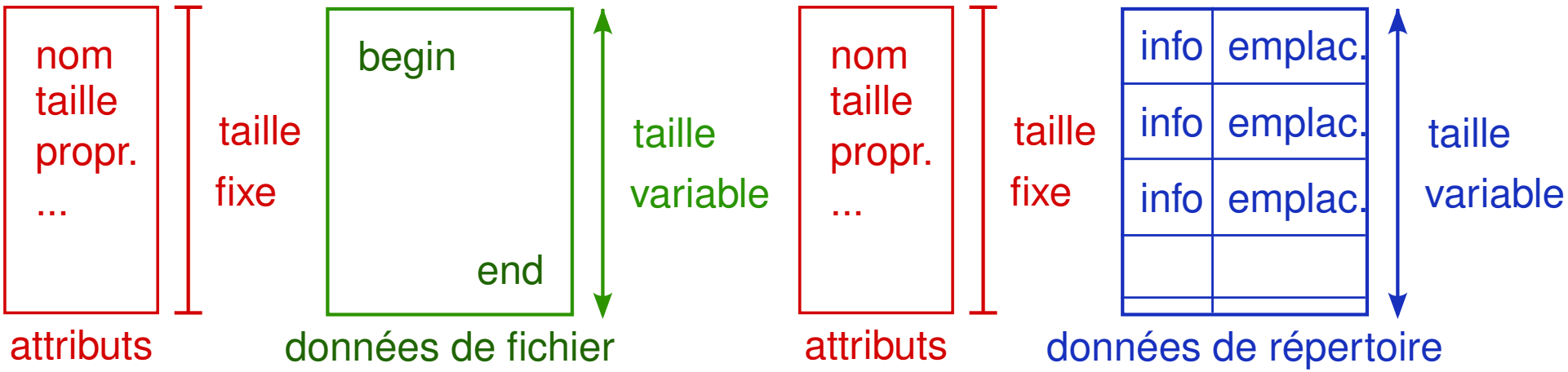
- contenu pour APFS =



Organiser et stocker l'information

Gérer

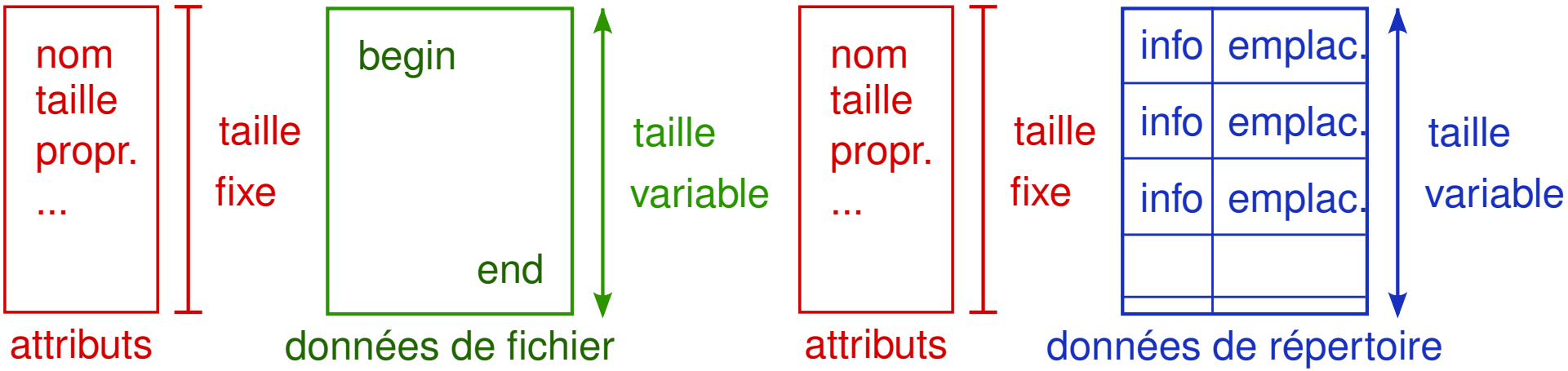
- une taille fixe pour les attributs
- une taille variable pour les données
- une arborescence



Organiser et stocker l'information

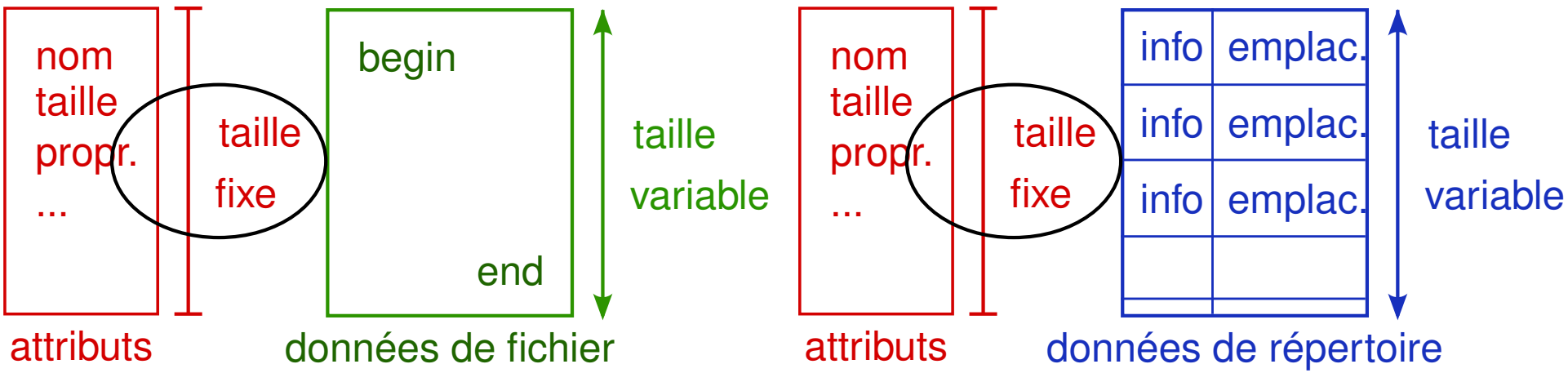
Gérer

- une taille fixe pour les attributs : 1 i-nœud
- une taille variable pour les données : des *datablocks*
- une arborescence : le contenu de *datablocks* de répertoires



ext2/3/4fs : un i-nœud et des datablocks

Le i-nœud : une structure de taille fixe pour ranger les attributs



ext2/3/4fs : un i-nœud et des datablocks

TD

méta-données (attributs)

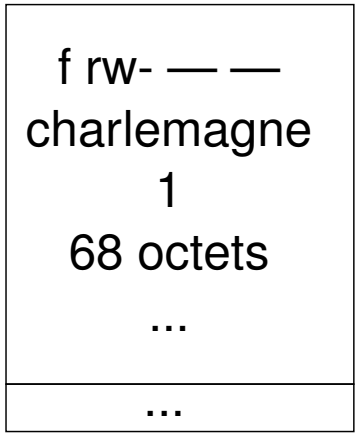
mode : nature + droits

propriétaire et groupe

nombre de liens

taille

dates : dernier accès, chgt i-nœud, modif.contenu



i-nœud du fichier /home/charlemagne/salut.c

---

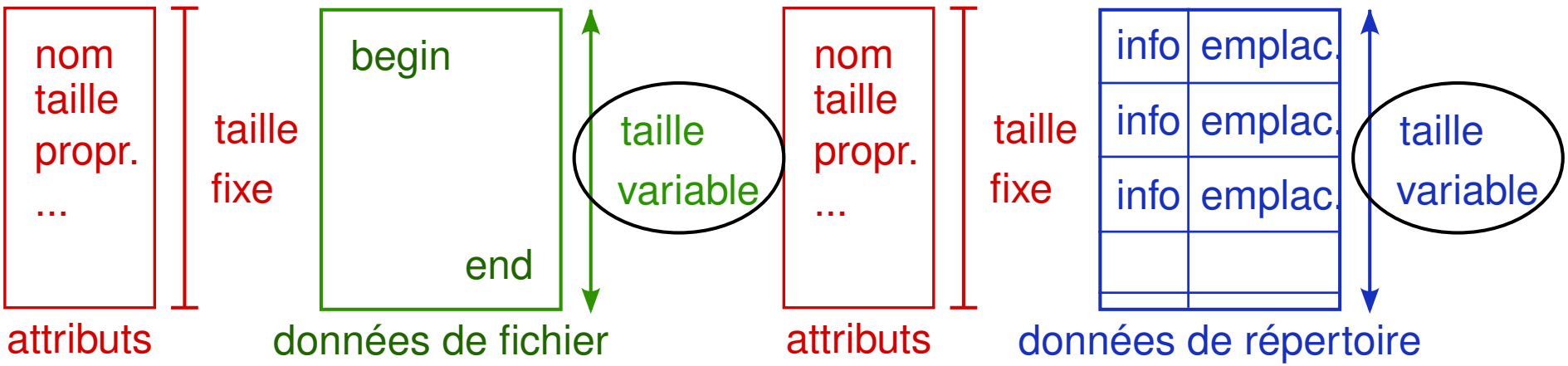
données

```
#include <stdio.h>
int main()
{
  printf("salut\ n") ;
  return 0 ;
}
```

contenu de /home/charlemagne/salut.c

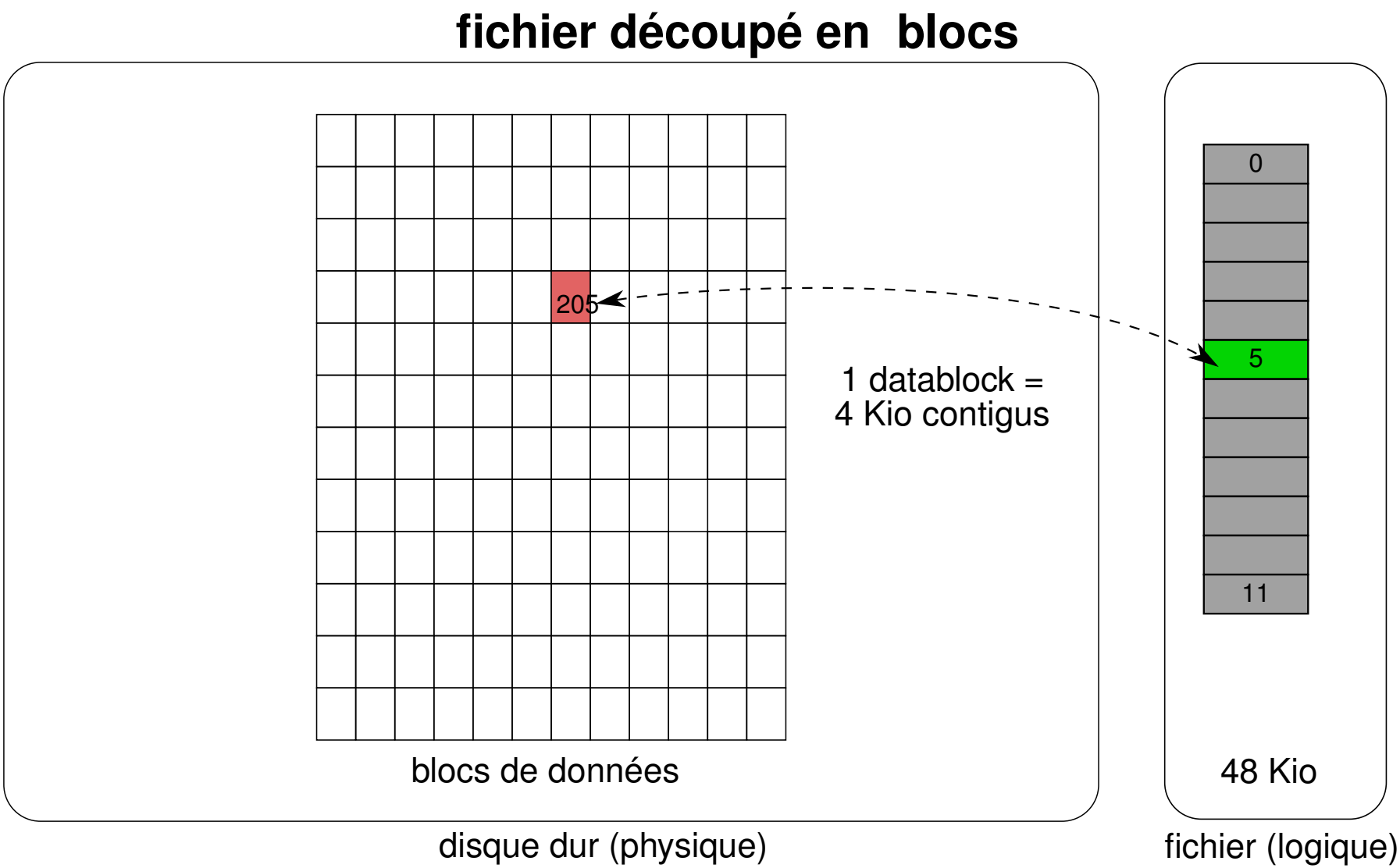
ext4fs : arbre d'extents de datablocks

Gérer la taille variable des données



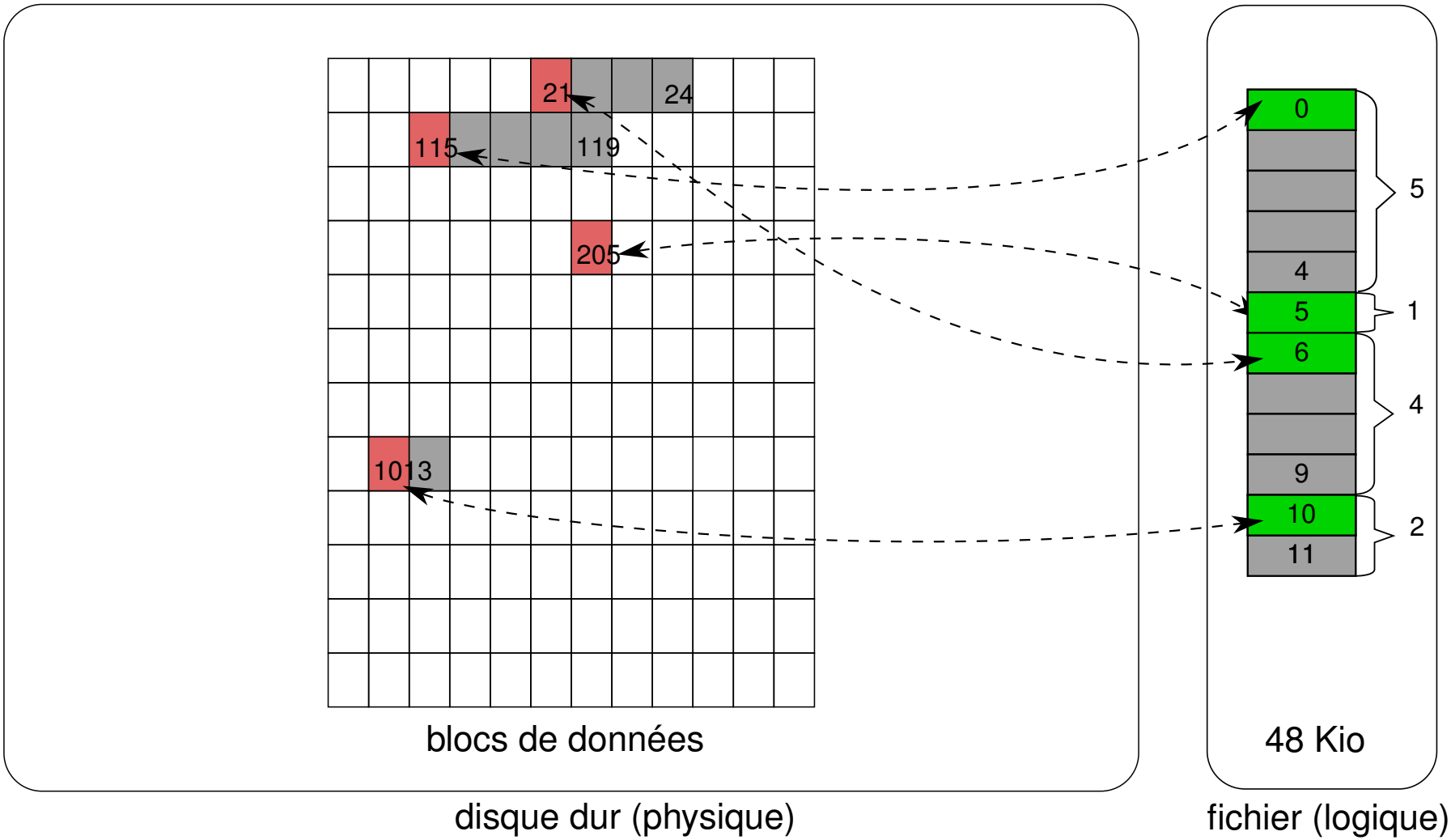
avec un ensemble non nécessairement contigu de *datablocks* (chaque *datablock* = 4Kio contigus).

ext4fs : arbre d'extents de datablocks



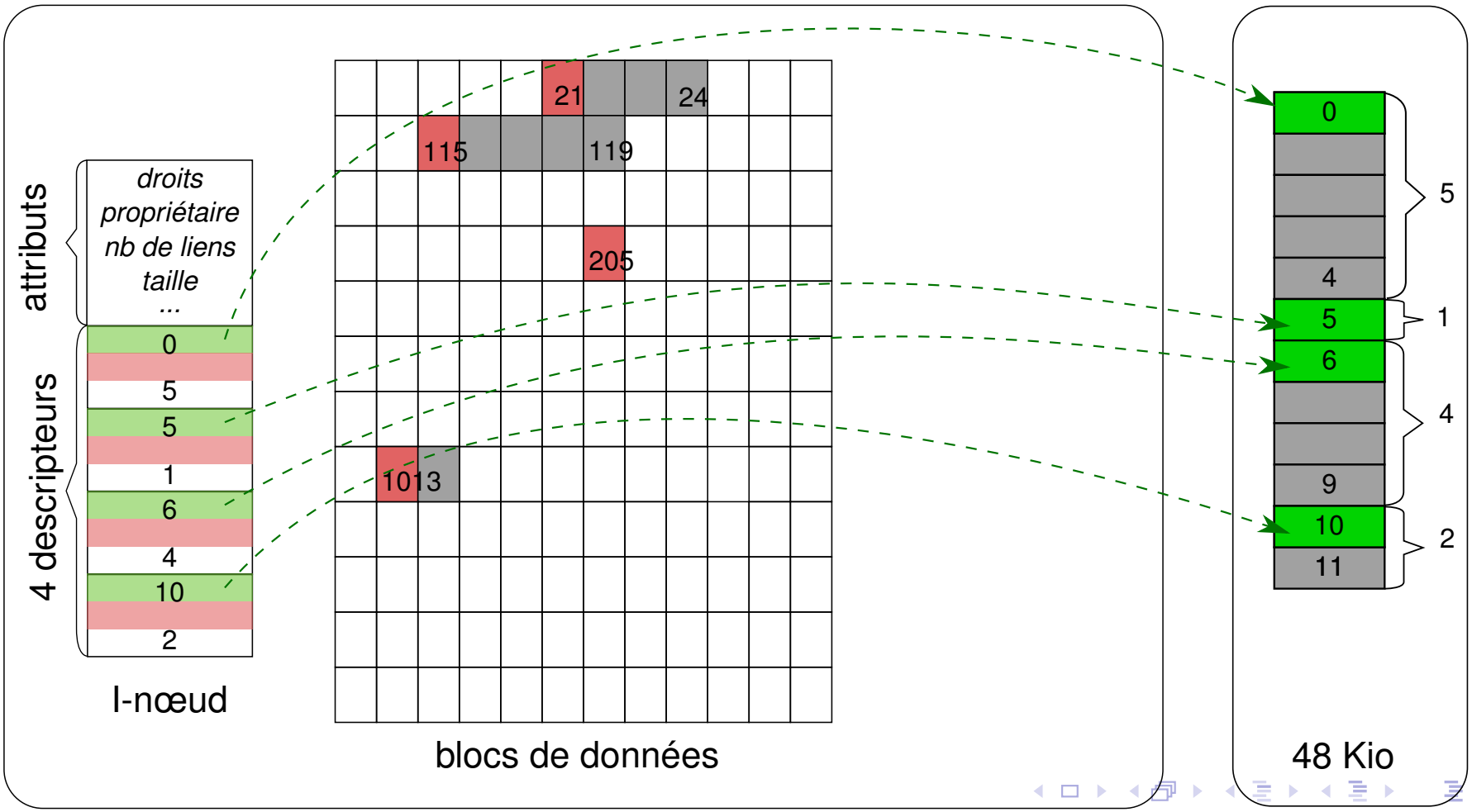
ext4fs : arbre d'extents de datablocks

fichier découpé en *paquets* de blocs contigus = *extents*



ext4fs : arbre d'extents de datablocks

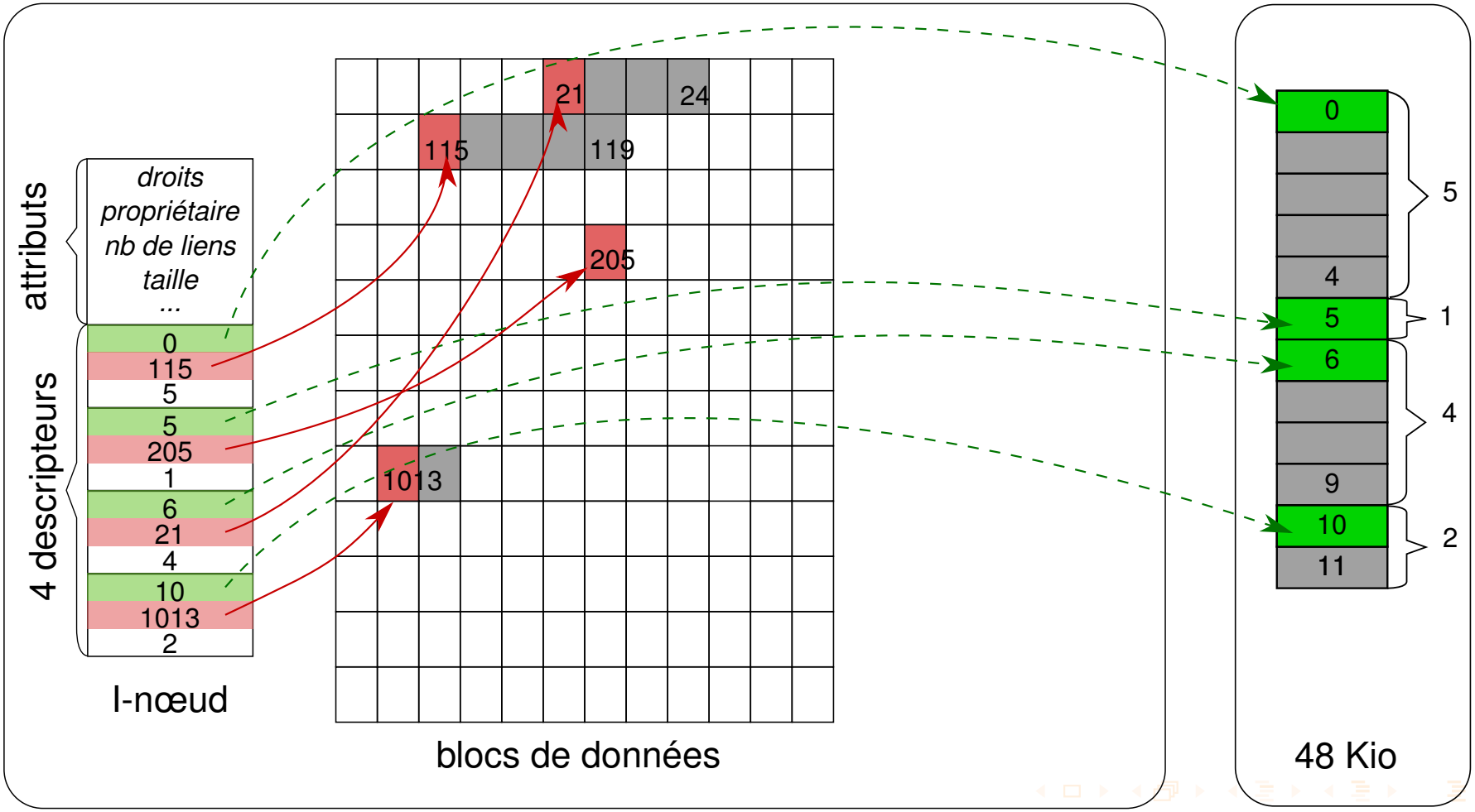
fichier découpé en *paquets* de blocs contigus = *extents*
***descripteurs d'extents* dans le i-nœud**



ext4fs : arbre d'extents de datablocks

TD

fichier découpé en *paquets* de blocs contigus = *extents*
***descripteurs d'extents* dans le i-nœud**



ext4fs : arbre d'extents de datablocks

Fichier creux

Si plusieurs descripteurs d'extents :

- les blocs d'un fichier peuvent ne pas être contigus sur le disque
- certains blocs du fichier peuvent ne pas exister sur le disque : **fichier creux** (ex : image d'une Machine Virtuelle [S1.03, S2.03])

descripteur d'extent (12 octets)

- numéro bloc dans fichier : 4 octets
- numéro (index) bloc sur disque : 6 octets
- nombre **blocs consécutifs** : 2 octets

Taille maximale d'un extent

16 bits : 15 bits pour le nombre de blocs + 1 bit pour l'initialisation
$\Rightarrow$ taille $\leq (2^{15} = 32768)$ blocs

ext4fs : arbre d'extents de datablocks

Fichier creux

Si plusieurs descripteurs d'extents :

- les blocs d'un fichier peuvent ne pas être contigus sur le disque
- certains blocs du fichier peuvent ne pas exister sur le disque : **fichier creux** (ex : image d'une Machine Virtuelle [S1.03, S2.03])

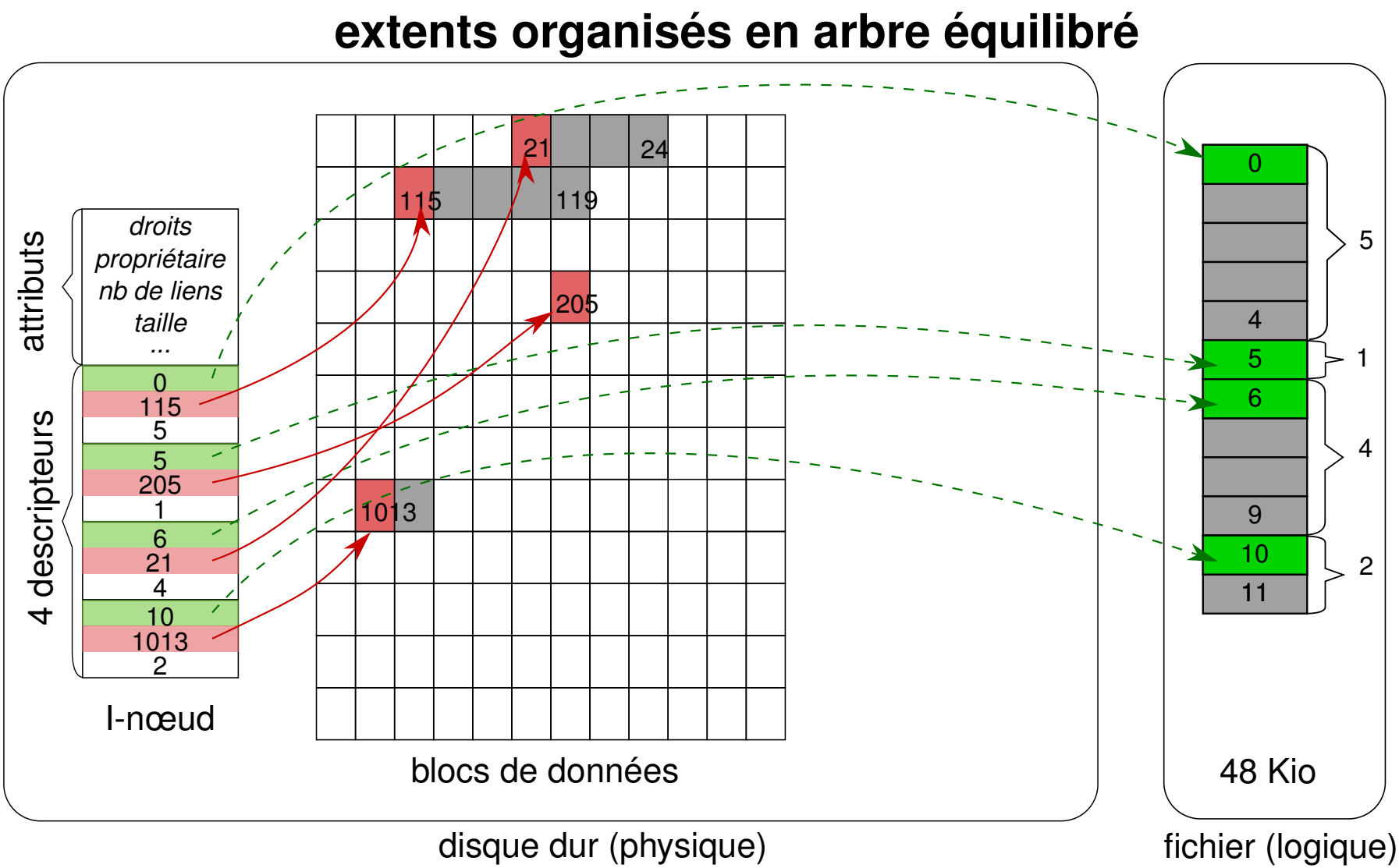
descripteur d'extent (12 octets)

- numéro bloc dans fichier : 4 octets
- numéro (index) bloc sur disque : 6 octets
- nombre **blocs consécutifs** : 2 octets

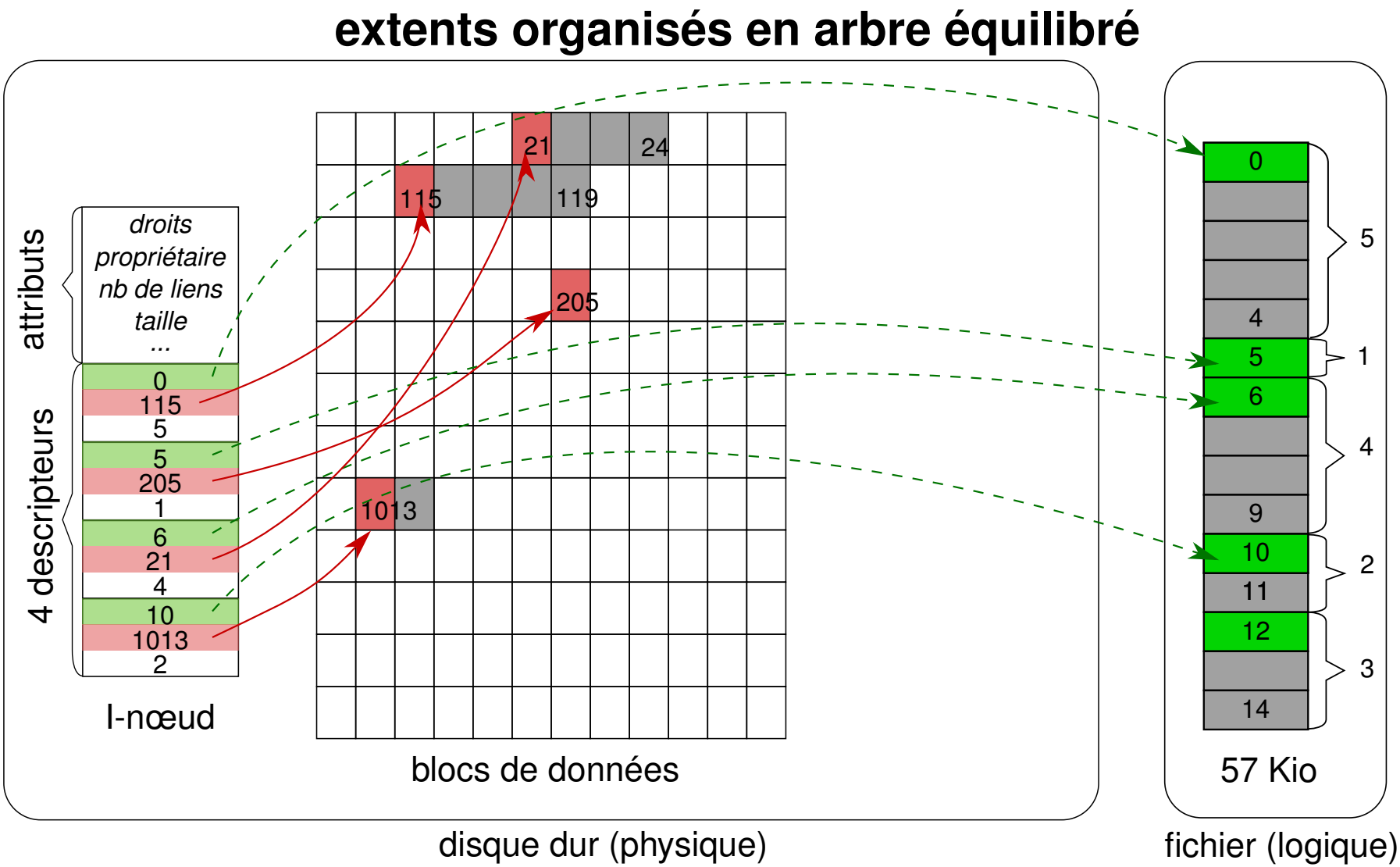
Taille maximale d'un extent

16 bits : 15 bits pour le nombre de blocs + 1 bit pour l'initialisation
$\Rightarrow$ taille $\leq (2^{15} = 32768)$ blocs

ext4fs : arbre d'extents de datablocks

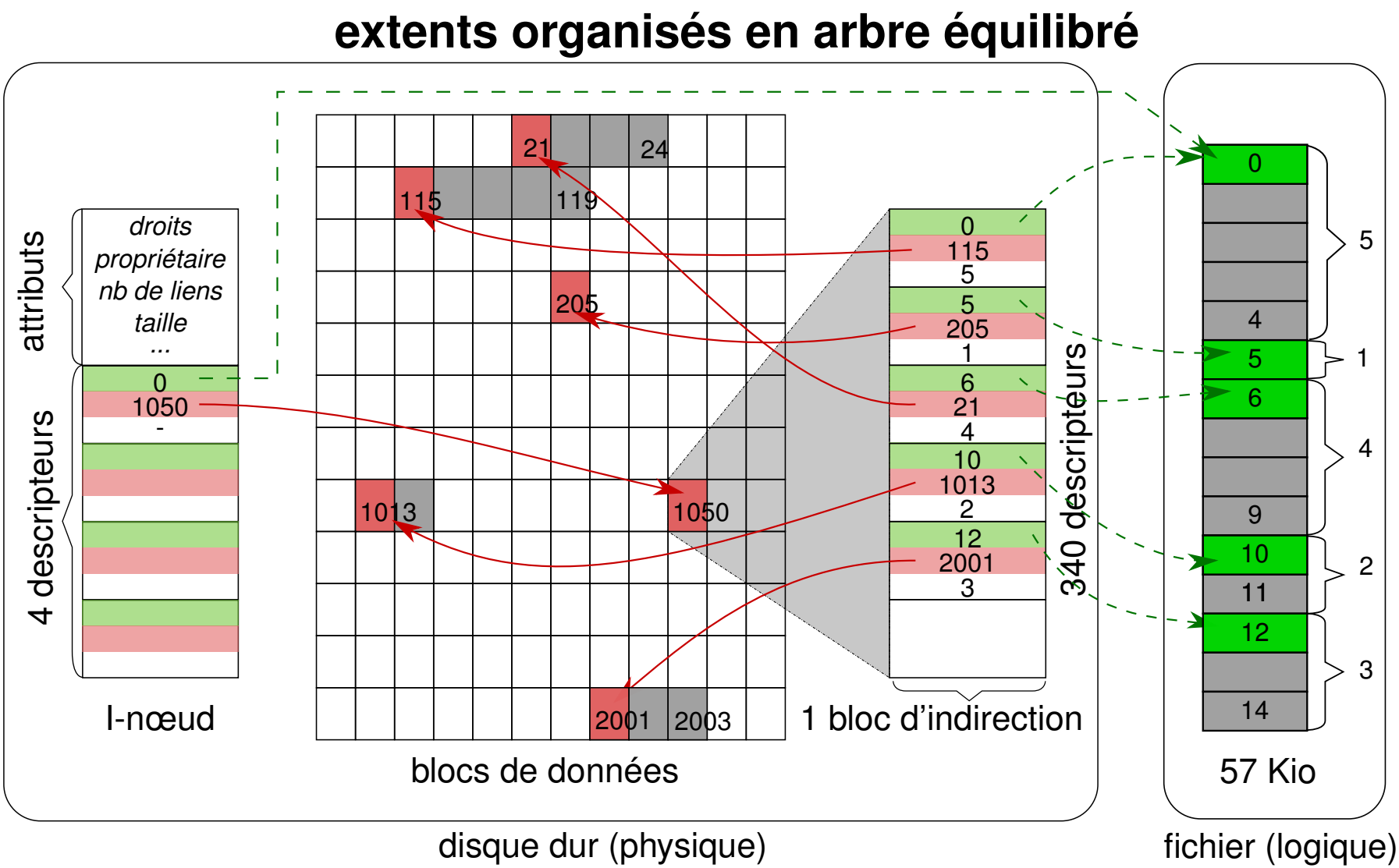


ext4fs : arbre d'extents de datablocks



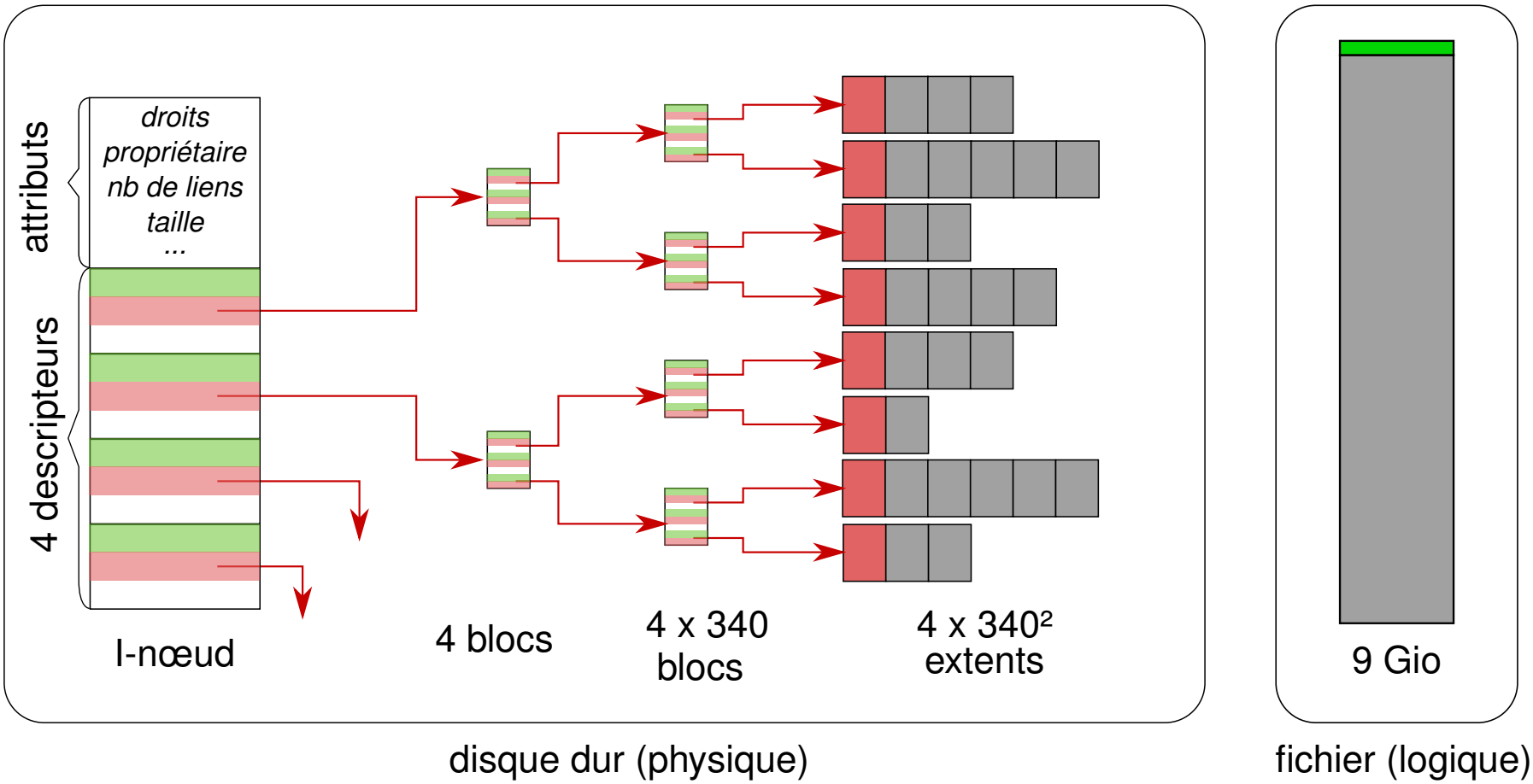
ext4fs : arbre d'extents de datablocks

TD

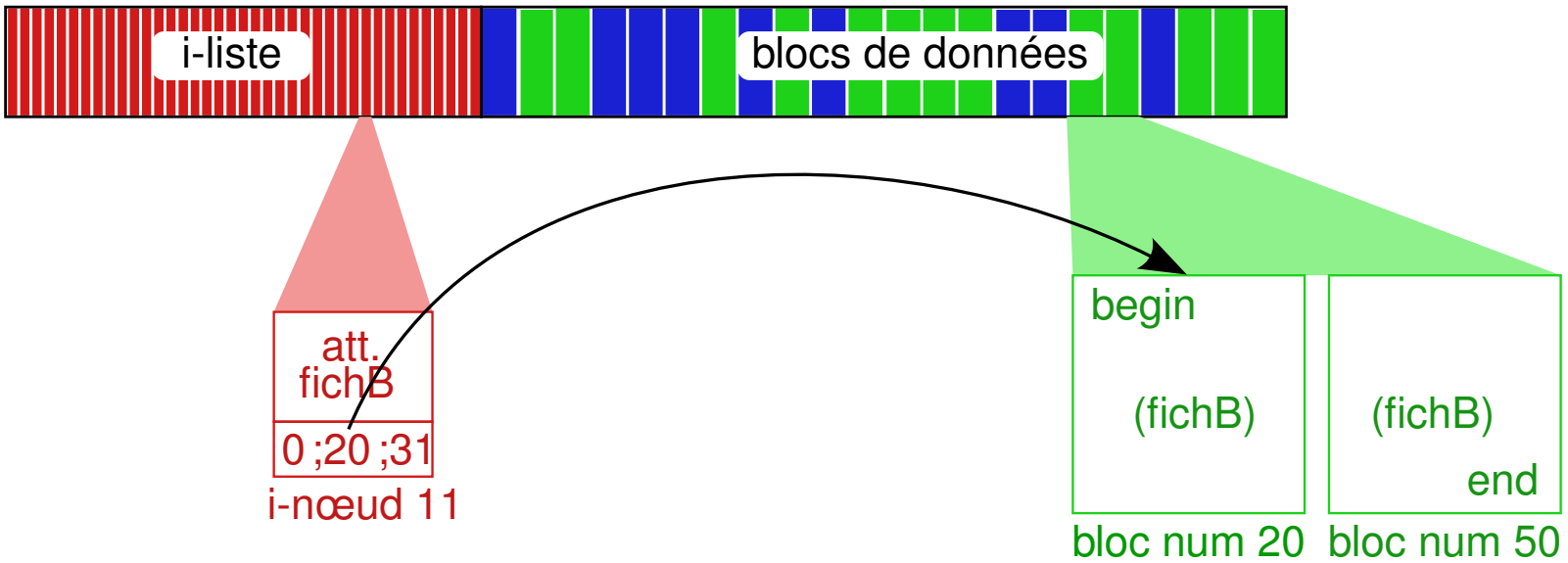


ext4fs : arbre d'extents de datablocks

extents organisés en arbre équilibré



ext2/3/4fs : un i-nœud et des datablocks



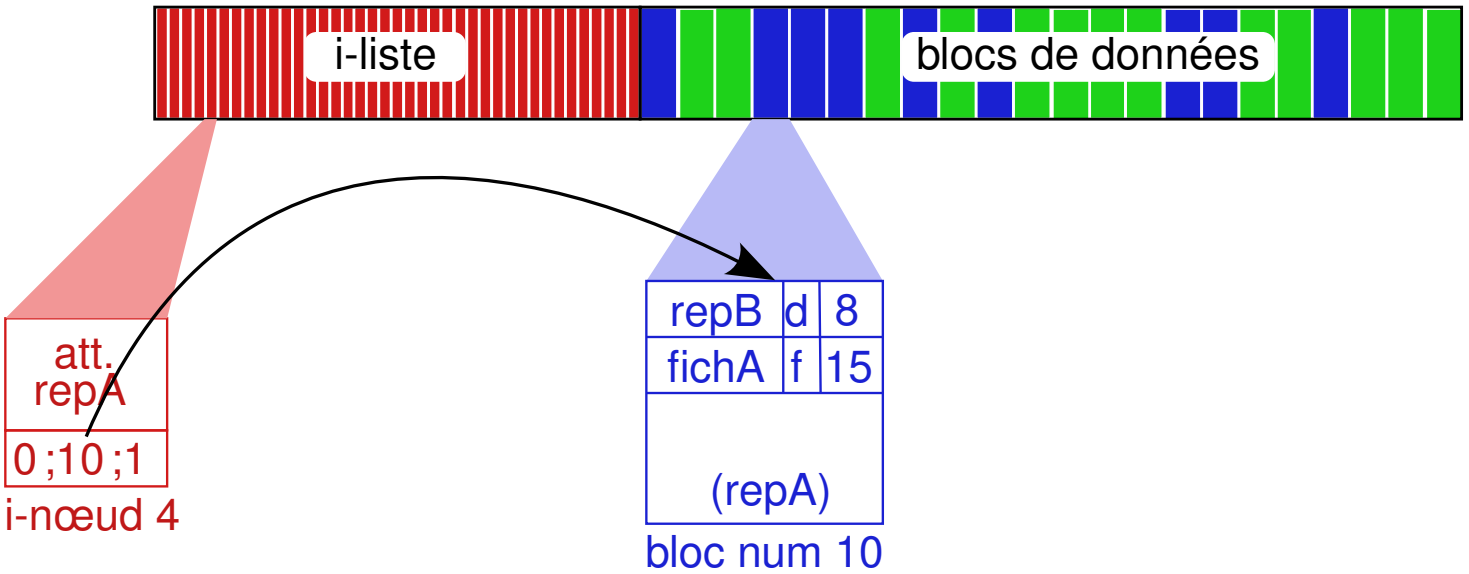
Fichier

- 1 i-nœud
- 0, 1 ou plusieurs blocs de données de 4Kio

Organisation séparée

- i-nœuds regroupés dans une i-liste
- blocs de données regroupés à part

ext2/3/4fs : un i-nœud et des datablocks



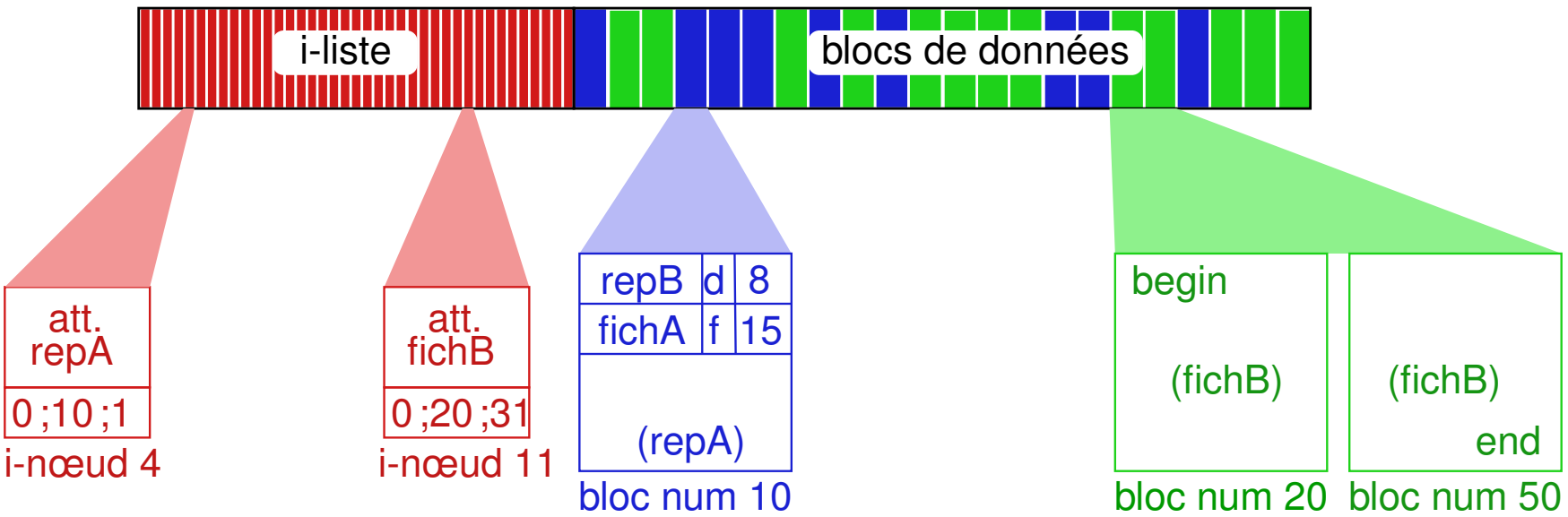
Répertoire

- 1 i-nœud
- 1 ou plusieurs blocs de données de 4Kio

Organisation séparée

- i-nœuds regroupés dans une i-liste
- blocs de données regroupés à part

ext2/3/4fs : un i-nœud et des datablocks



Fichier ou répertoire

- 1 i-nœud
- 0, 1 ou plusieurs blocs de données de 4Kio

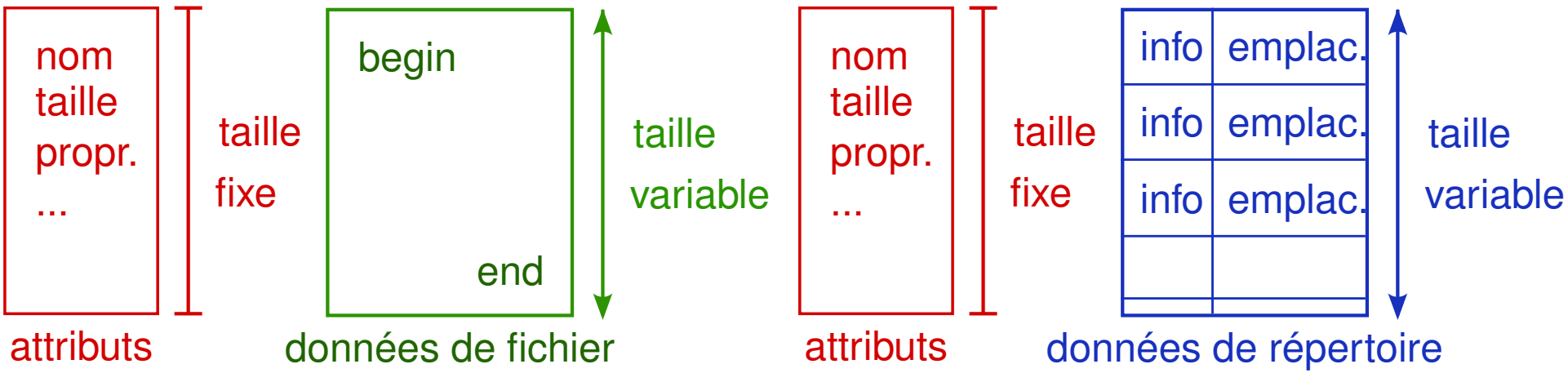
Organisation séparée

- i-nœuds regroupés dans une i-liste
- blocs de données regroupés à part

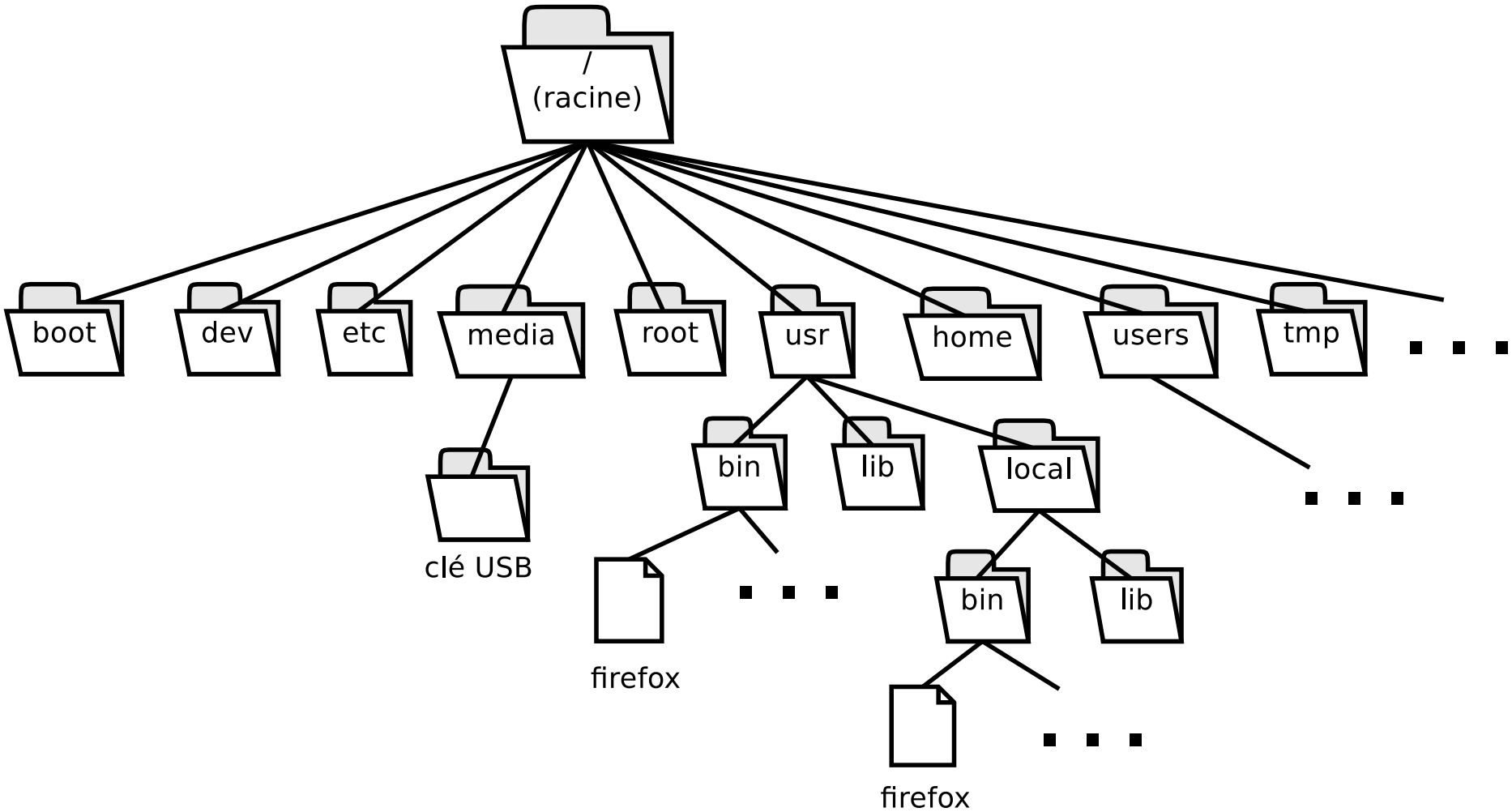
Organiser et stocker l'information

Gérer

- une taille fixe pour les attributs : 1 i-nœud
- une taille variable pour les données : des *datablocks*
- une arborescence : le contenu de *datablocks* de répertoires

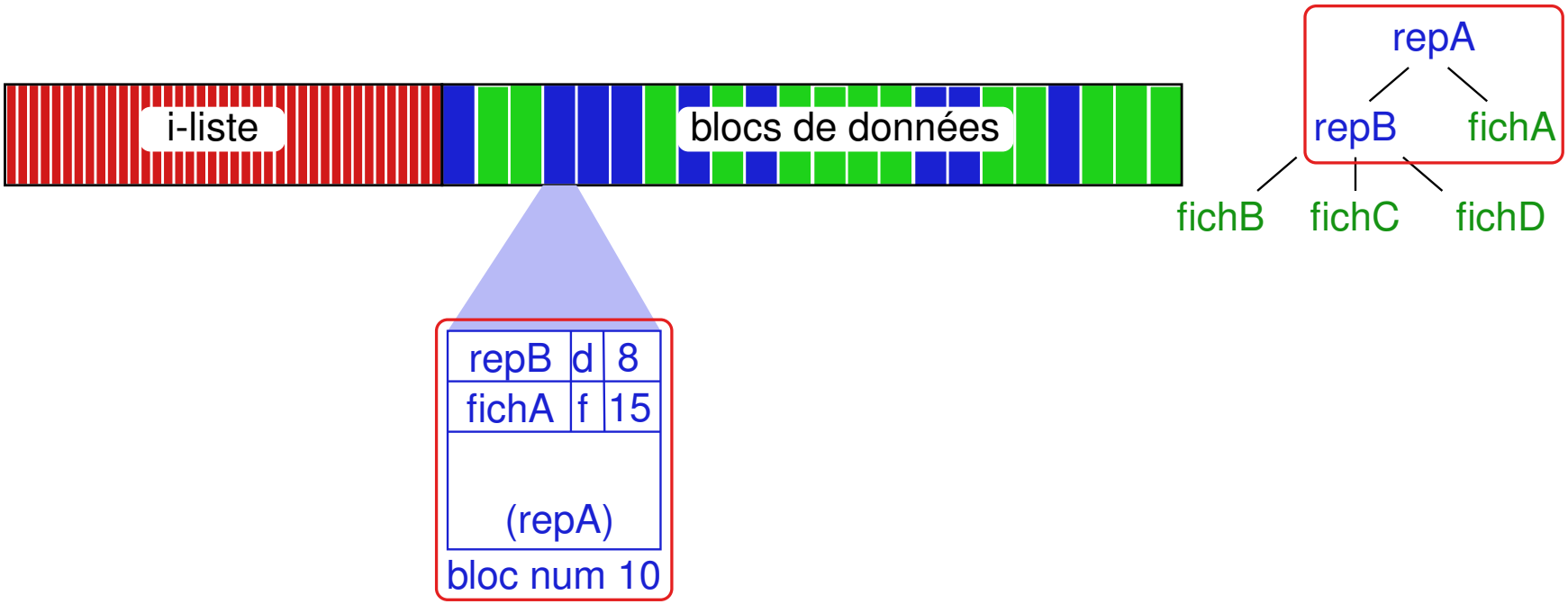


Arborescence Linux standard



Standard : FHS *(Filesystem Hierarchy Standard)*

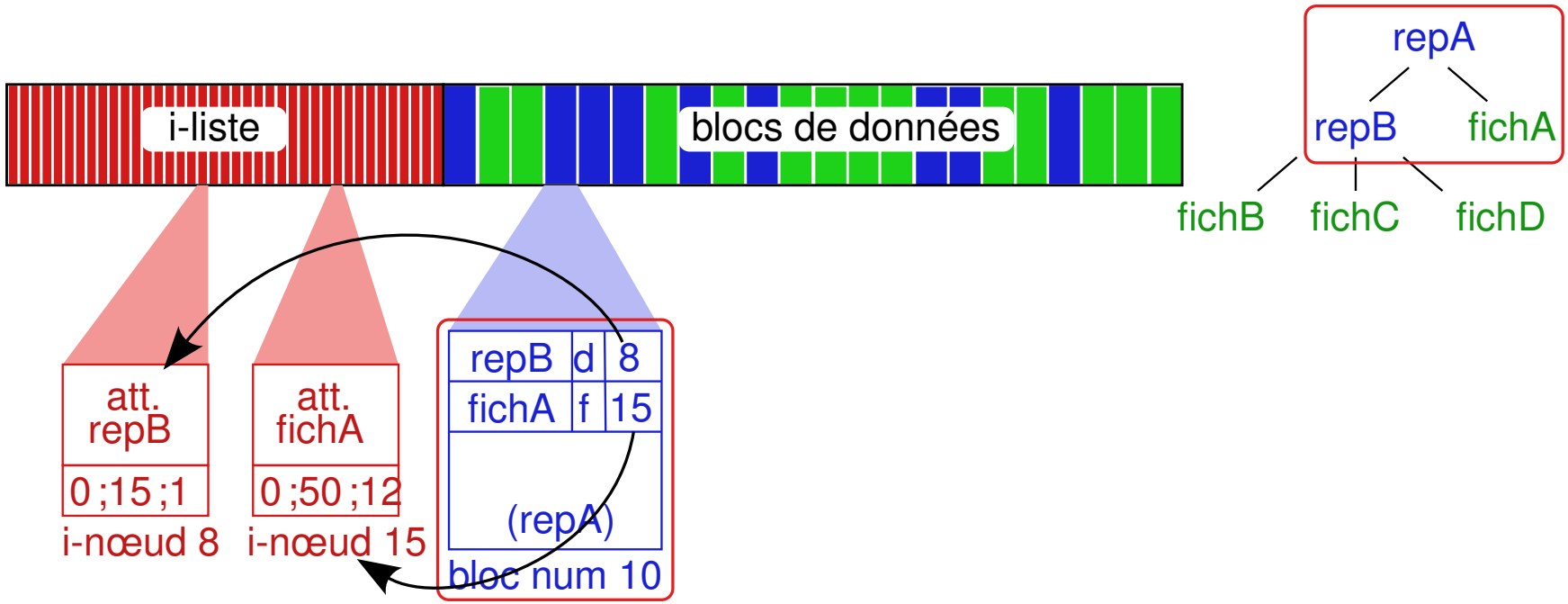
ext2/3/4fs : un i-nœud et des datablocks



Données d'un répertoire

- suite d'entrées, une entrée par fichier/répertoire contenu
- chaque entrée :
  nom et type du fichier/répertoire + numéro du i-nœud

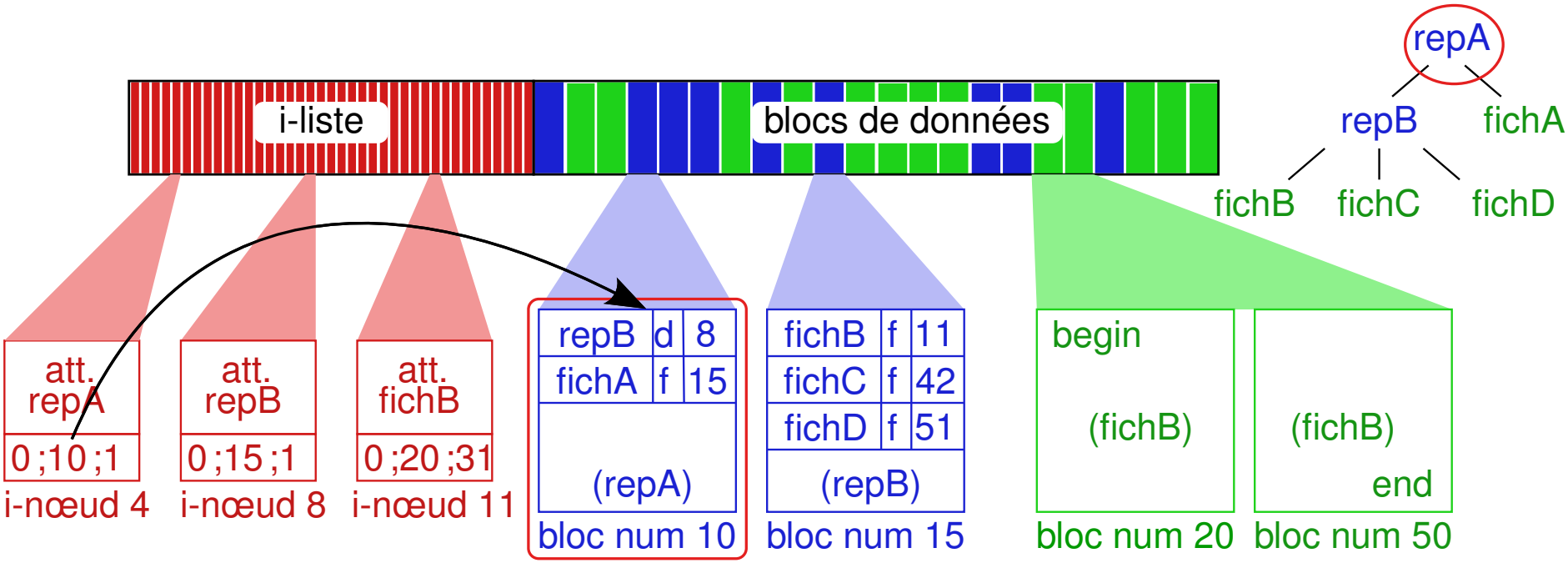
ext2/3/4fs : un i-nœud et des datablocks



Données d'un répertoire

- suite d'entrées, une entrée par fichier/répertoire contenu
- chaque entrée :
  nom et type du fichier/répertoire + numéro du i-nœud

ext2/3/4fs : un i-nœud et des datablocks



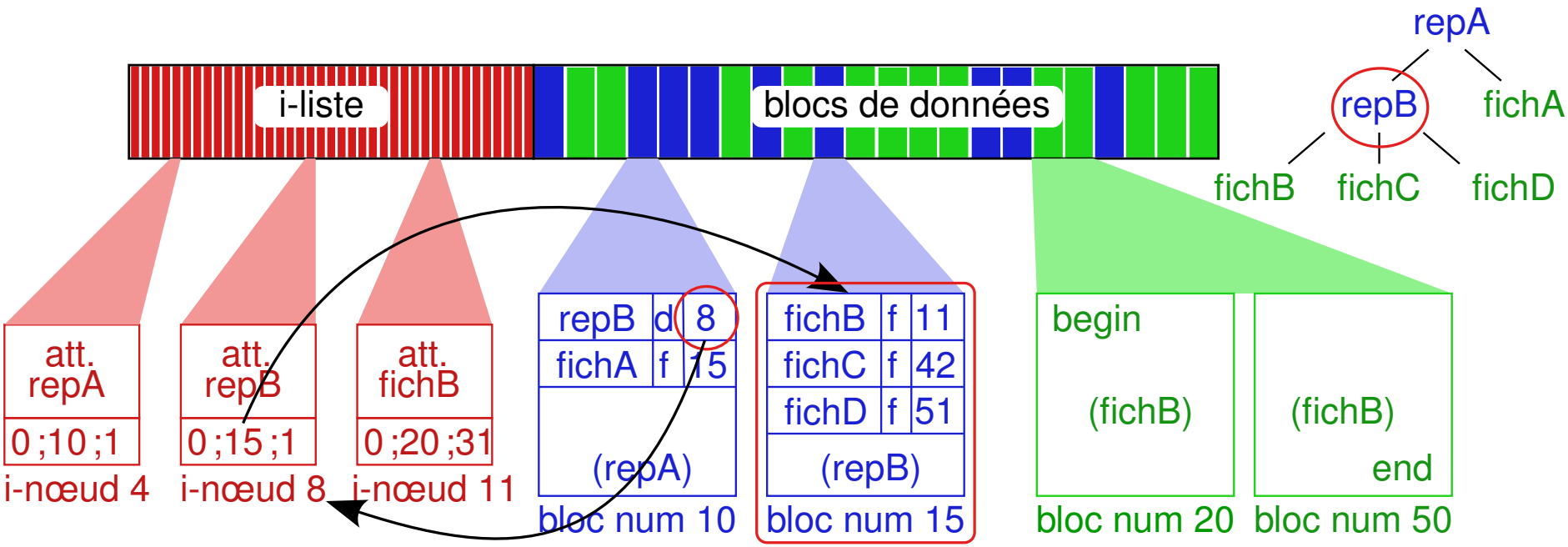

```
cat repA/repB/fichB
```

Données d'un répertoire

- suite d'entrées, une entrée par fichier/répertoire contenu
- chaque entrée :
  nom et type du fichier/répertoire + numéro du i-nœud

ext2/3/4fs : un i-nœud et des datablocks

TD

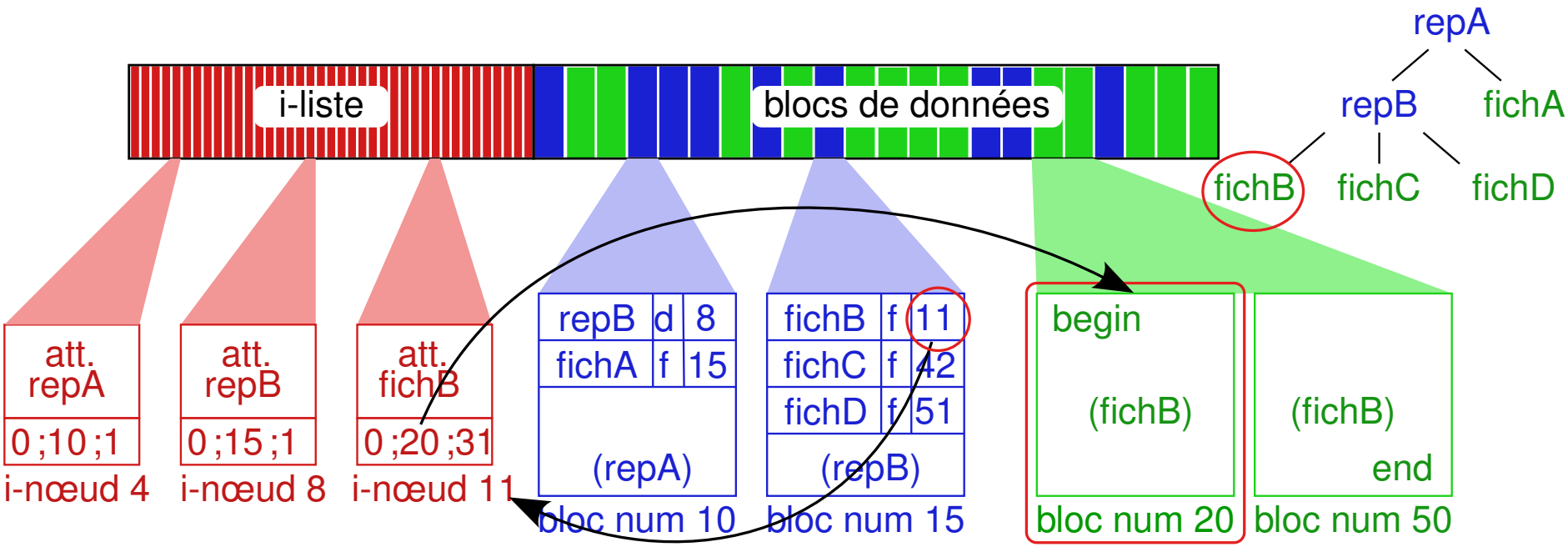



```
cat repA/repB/fichB
```

Données d'un répertoire

- suite d'entrées, une entrée par fichier/répertoire contenu
- chaque entrée :
  nom et type du fichier/répertoire + numéro du i-nœud

ext2/3/4fs : un i-nœud et des datablocks




```
cat repA/repB/fichB
```

Données d'un répertoire

- suite d'entrées, une entrée par fichier/répertoire contenu
- chaque entrée :
  nom et type du fichier/répertoire + numéro du i-nœud

4. Système de Gestion de Fichiers

4.1. Révisions [R1.04]

- Mémoire secondaire
- Arborescence de fichiers et répertoires
- Attributs d'un fichier/répertoire (métadonnées)

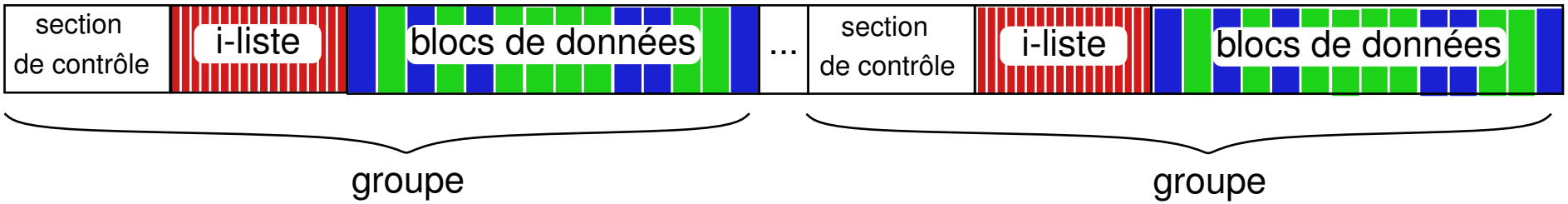
4.2. SGF ext4

- Structures de données : i-nœud et datablocks
- Protection

4.3. Accès aux fichiers par les processus

4.4. Une arborescence mais des supports et SGF variés

ext2/3/4fs : groupe = sect. de contr. + i-liste + blocs



Section de contrôle ⊃ Superblock

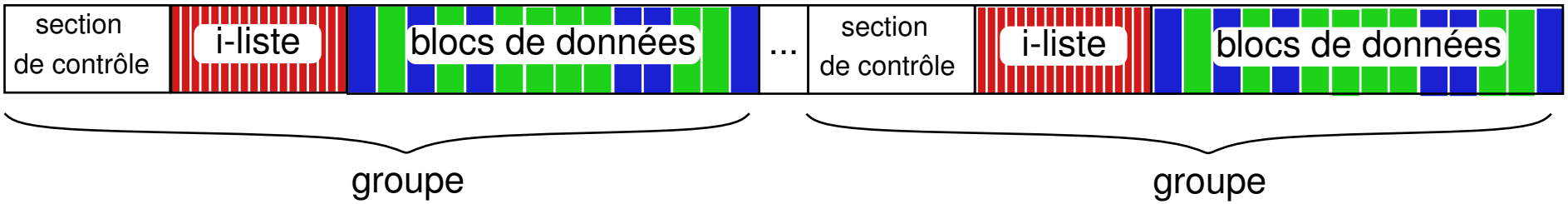
données essentielles pour la gestion et l'utilisation du SGF

- nombre et localisation des i-nœuds libres
- nombre et localisation des blocs libres
- etc.

Intérêt d'avoir plusieurs groupes

- + fiable : Superblock dupliqué
- + performant (HDD) : distance i-nœuds-datablocks réduite
- `flex_bg` (ext4) : conserve les copies de superblock mais réunit les i-listes et datablocks de plusieurs groupes (gros fichiers)

ext2/3/4fs : groupe = sect. de contr. + i-liste + blocs



Section de contrôle ⊃ Superblock

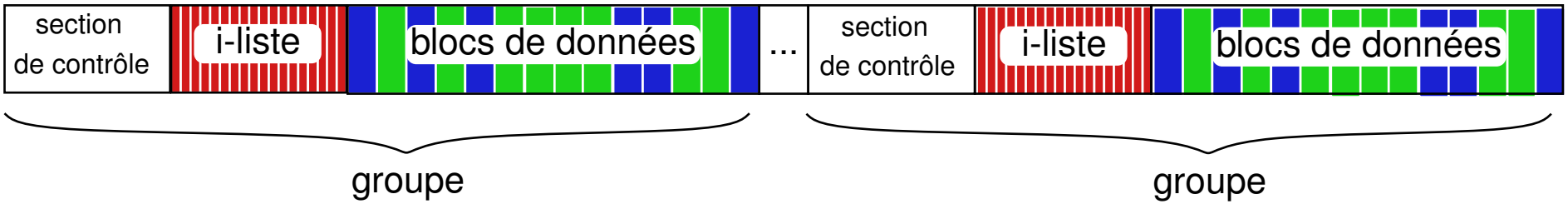
données essentielles pour la gestion et l'utilisation du SGF

- nombre et localisation des i-nœuds libres
- nombre et localisation des blocs libres
- etc.

Intérêt d'avoir plusieurs groupes

- + fiable : Superblock dupliqué
- + performant (HDD) : distance i-nœuds-datablocks réduite
- `flex_bg` (ext4) : conserve les copies de superblock mais réunit les i-listes et datablocks de plusieurs groupes (gros fichiers)

ext2/3/4fs : groupe = sect. de contr. + i-liste + blocs



Section de contrôle ⊃ Superblock

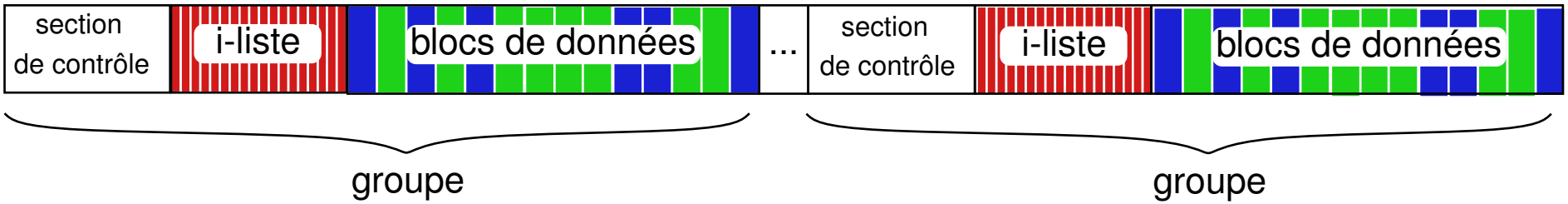
données essentielles pour la gestion et l'utilisation du SGF

- nombre et localisation des i-nœuds libres
- nombre et localisation des blocs libres
- etc.

Intérêt d'avoir plusieurs groupes

- + fiable : Superblock dupliqué
- + performant (HDD) : distance i-nœuds-datablocks réduite
- `flex_bg` (ext4) : conserve les copies de superblock mais réunit les i-listes et datablocks de plusieurs groupes (gros fichiers)

ext2/3/4fs : groupe = sect. de contr. + i-liste + blocs



Section de contrôle ⊃ Superblock

données essentielles pour la gestion et l'utilisation du SGF

- nombre et localisation des i-nœuds libres
- nombre et localisation des blocs libres
- etc.

Intérêt d'avoir plusieurs groupes

- + fiable : Superblock dupliqué
- + performant (HDD) : distance i-nœuds-datablocks réduite
- `flex_bg` (ext4) : conserve les copies de superblock mais réunit les i-listes et datablocks de plusieurs groupes (gros fichiers)

ext3/4fs : journal

Problème possible

- chaque action sur un fichier/répertoire se décompose en plusieurs étapes (gestion des i-nœuds, extents, datablocks, à rechercher).
- **si l'action est interrompue (arrêt intempestif, panne, ...)**
  $\Rightarrow$ **état non-cohérent**.

Solution

- écrire immédiatement dans un journal les modifications demandées pour une action = ***transaction***.
  (écriture séquentielle, rapide, dans un fichier spécial).
- retranscrire ces modifications en tâche de fond.
- supprimer la transaction du journal une fois l'action accomplie.
- en cas de panne : à la reprise, recommencer toute l'action si la transaction est complète dans le journal.

ext4fs (2012) : checksums

Définition

- petit code numérique calculé à partir d'un ensemble de données plus important
- permet de vérifier avec une haute probabilité l'intégrité de cet ensemble
- utile après transmission ou copie (Ex R2.05 : FCS d'une trame ethernet ou checksum dans les en-têtes UDP/TCP)

Dans ext4fs (depuis 2012)

- les structures de extfs (i-nœud, datablock, extent, superblock. . .)
- dans le journal

Dans btrfs

- également pour les données