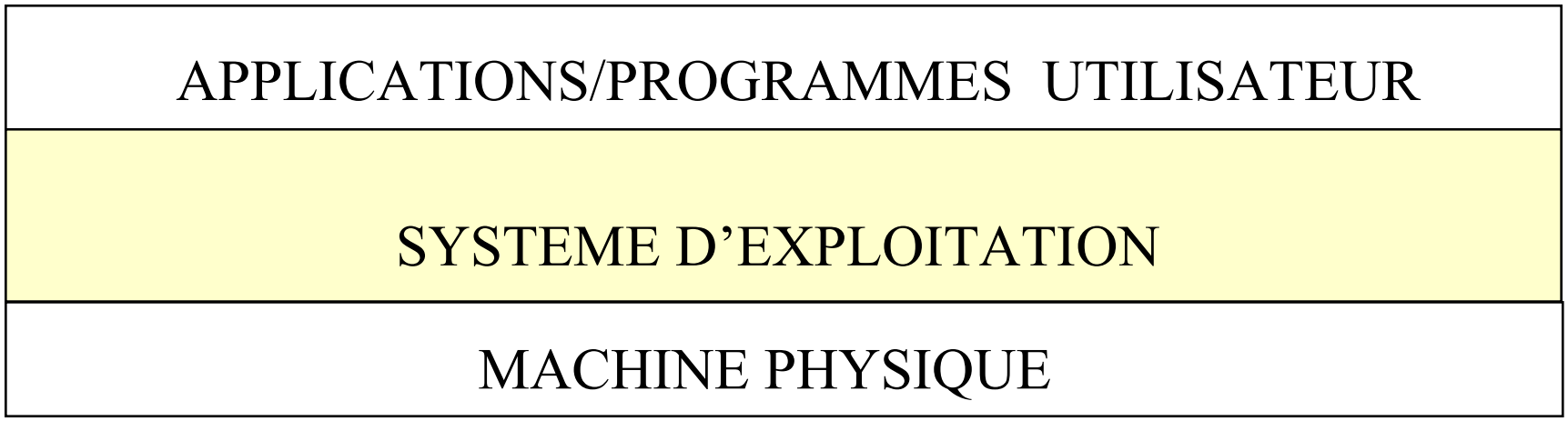
PDF-XChange Viewer
Click to buy NOW!
www.docu-track.com

Introduction
aux systèmes d'exploitation

DEFINITION D'UN SYSTEME D'EXPLOITATION

PDF-XChange Viewer Click to buy NOW! www.docu-track.com

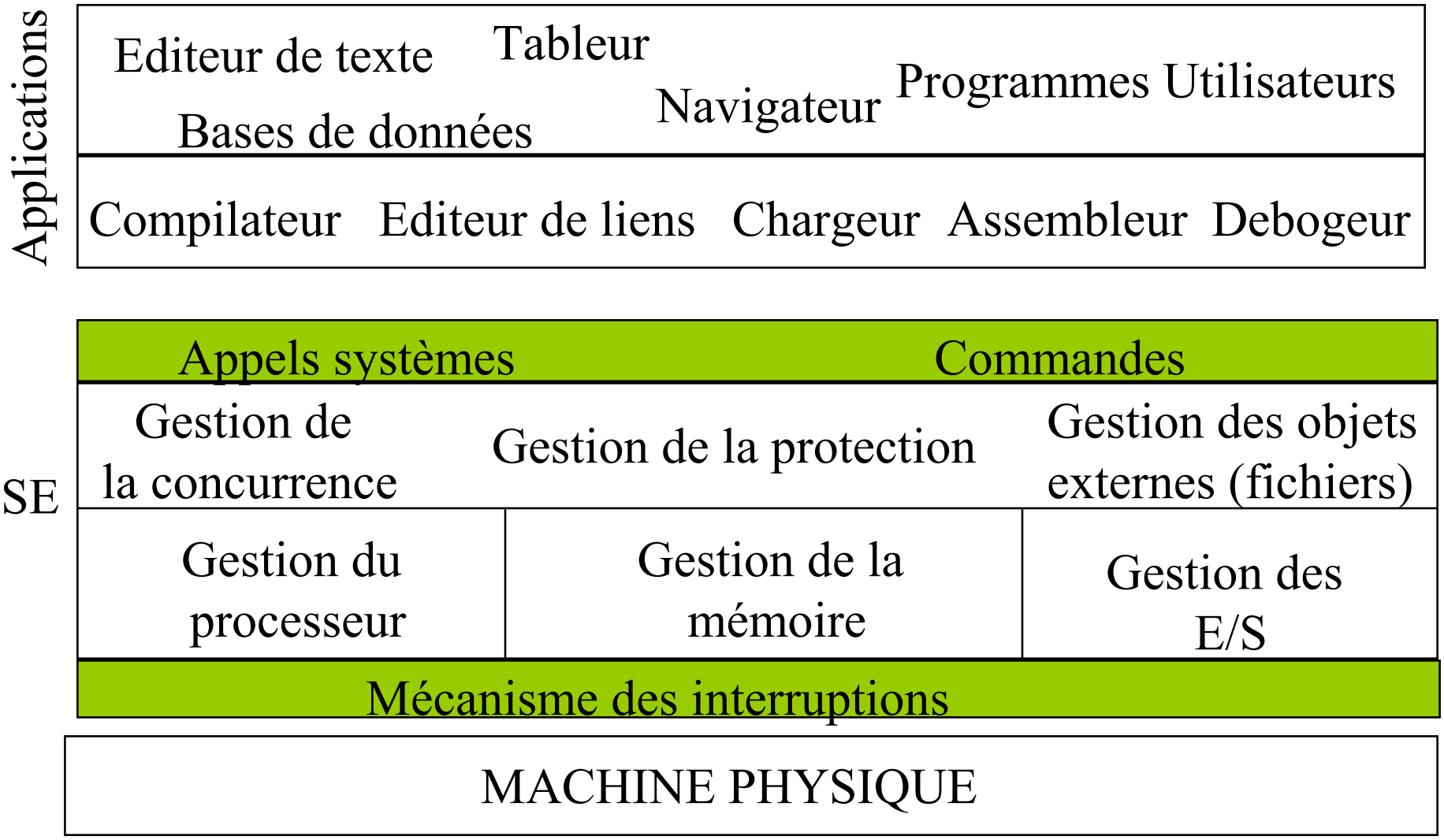


Ensemble de programmes qui réalisent l'interface entre le matériel de l'ordinateur et les utilisateurs. Il a deux objectifs principaux :

- construction au dessus du matériel d'une machine virtuelle plus facile d'emploi et plus conviviale
- prise en charge de la gestion de plus en plus complexe des ressources et partage de celle-ci

Comme son nom le suggère, le SE a en charge l'exploitation de la machine pour en faciliter l'accès, le partage et pour l'optimiser

FONCTIONS D'UN SYSTEME D'EXPLOITATION



PDF-XChange Viewer
Click to buy NOW!
www.docu-track.com

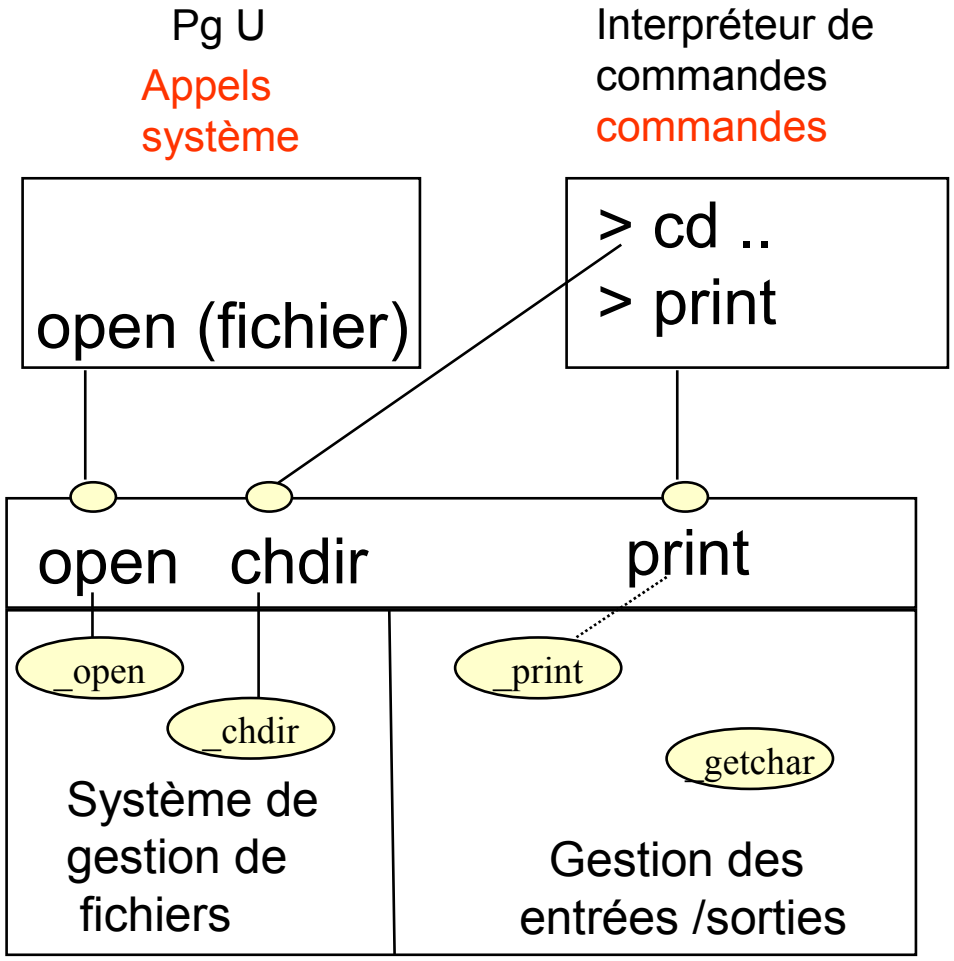
Notions de base

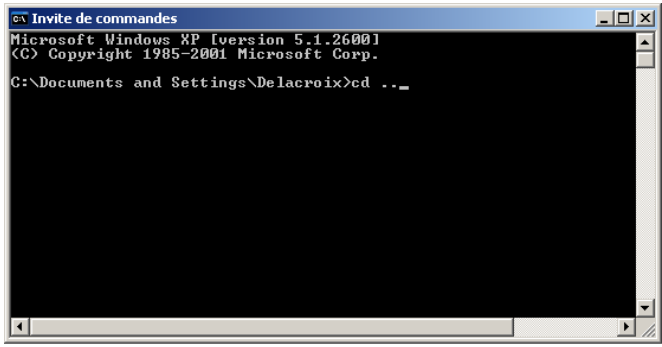
- Modes d'exécutions
- Interruptions logicielles et matérielles
- Chargement du système

Appels système et commandes

PDF-XChange Viewer Click to buy NOW! www.docu-track.com

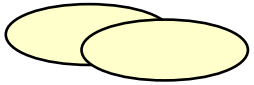
- Les fonctionnalités du système d'exploitation sont accessibles par le biais des **commandes** ou des **appels système**





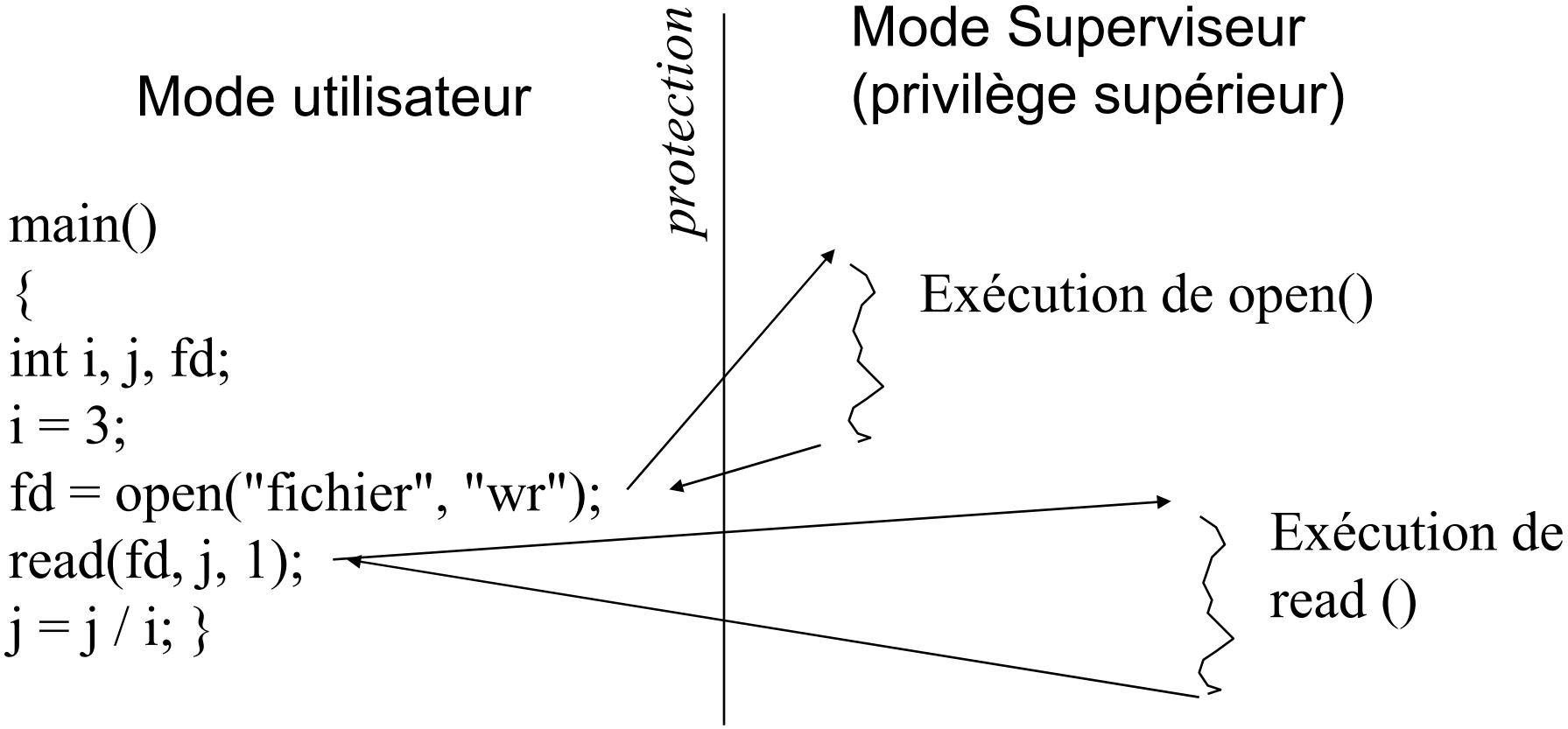
Interface d'appel

SE : ensemble de fonctions



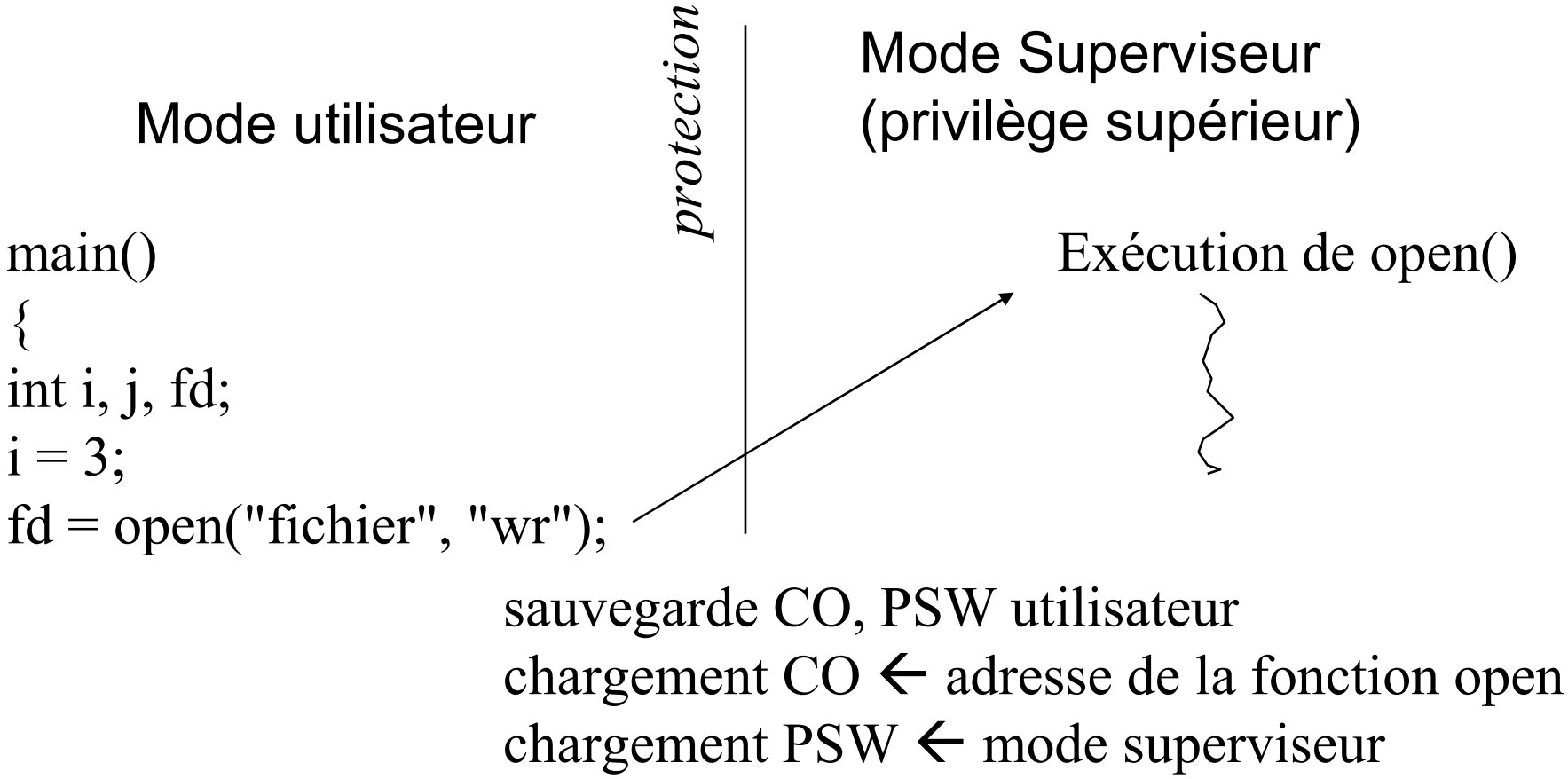
Modes d'exécutions

- Lors de l'appel a une fonction du système, le programme utilisateur passe d'un **mode d'exécution dit utilisateur** à un **mode d'exécution dit superviseur**.



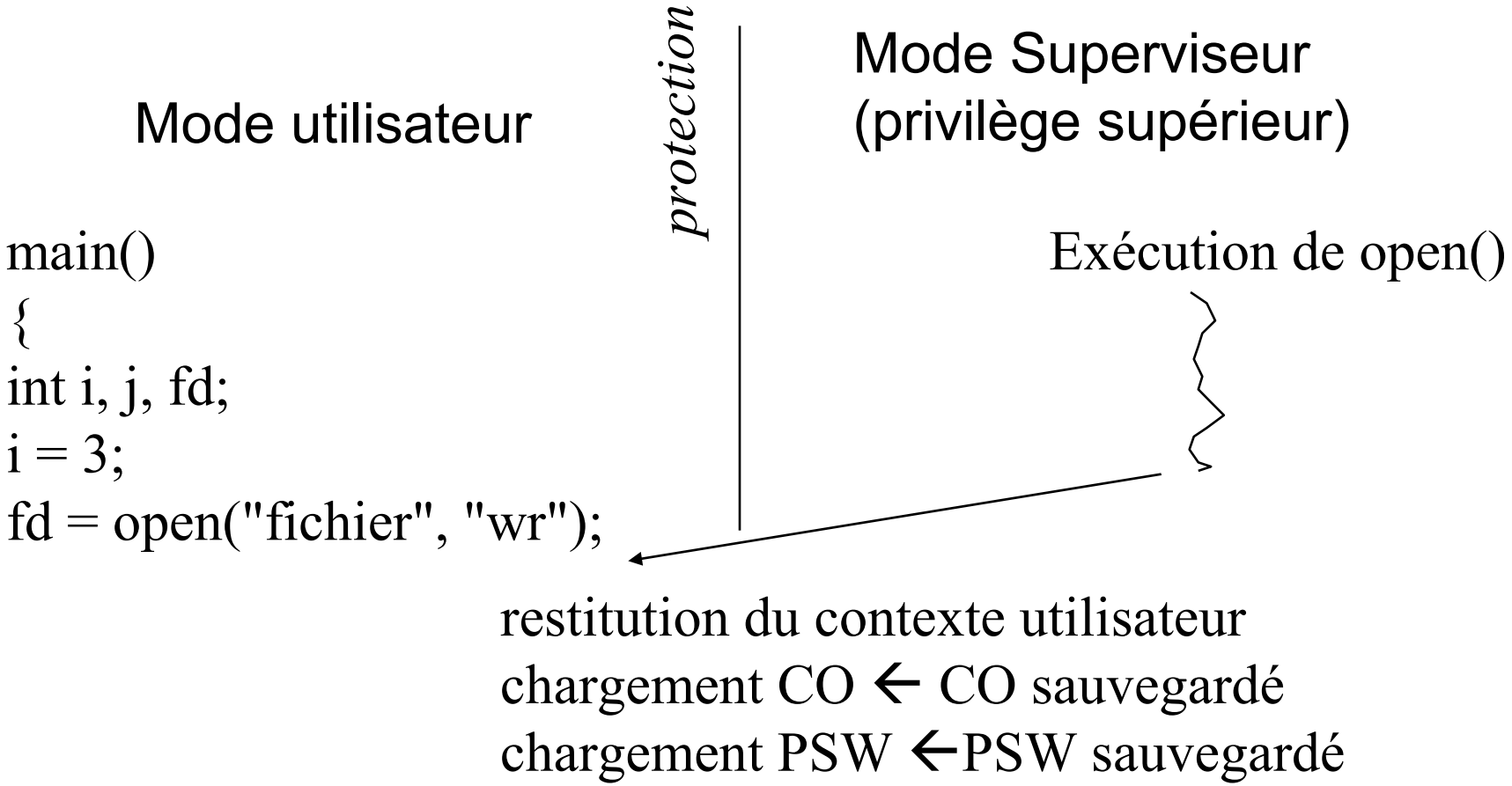
Modes d'exécutions

- Le passage du mode utilisateur au mode superviseur s'accompagne d'opérations de **commutation de contexte : sauvegarde de contexte utilisateur**

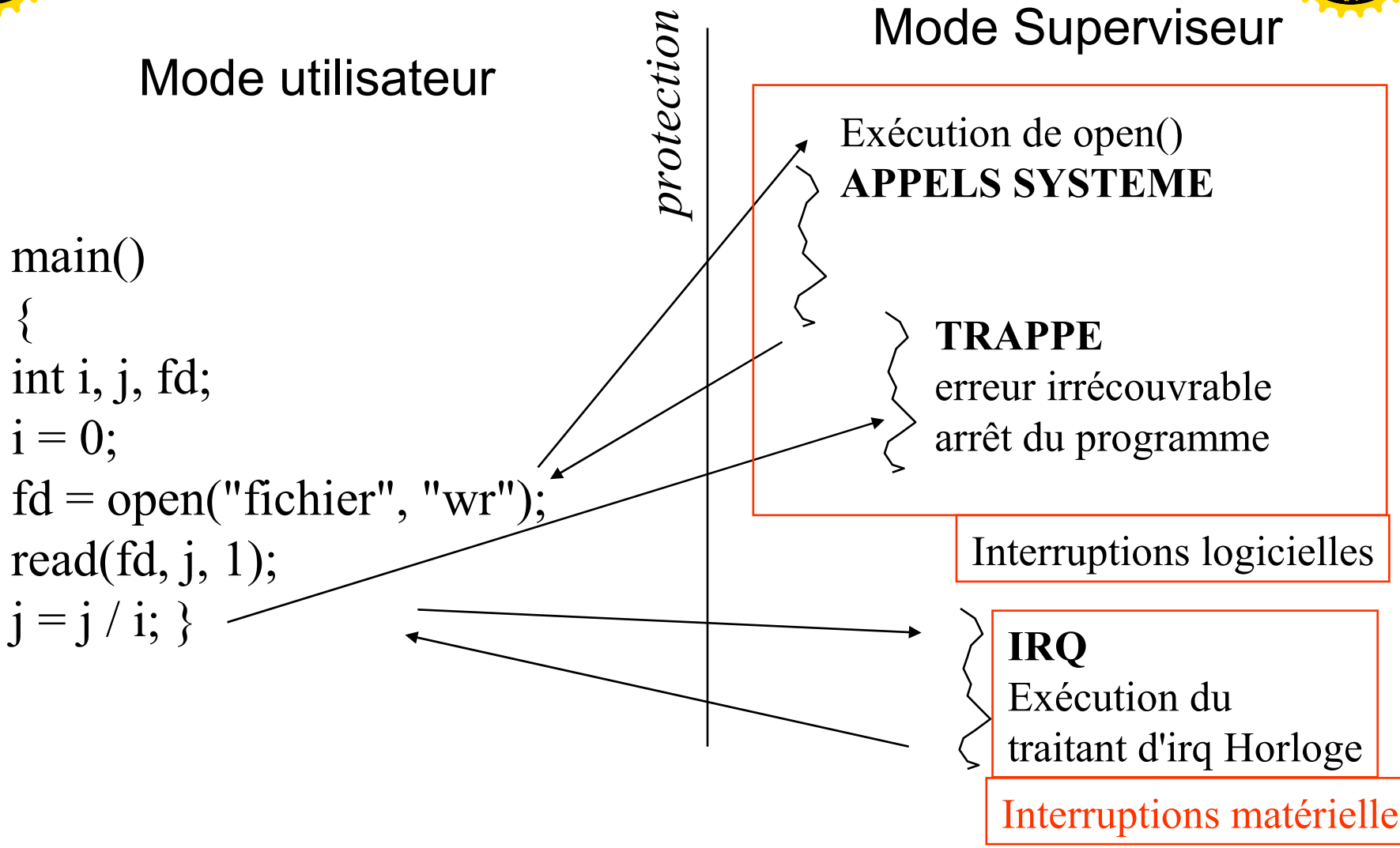


Modes d'exécutions

- Le passage du mode superviseur au mode utilisateur s'accompagne d'opérations de **commutation de contexte : restitution de contexte utilisateur**



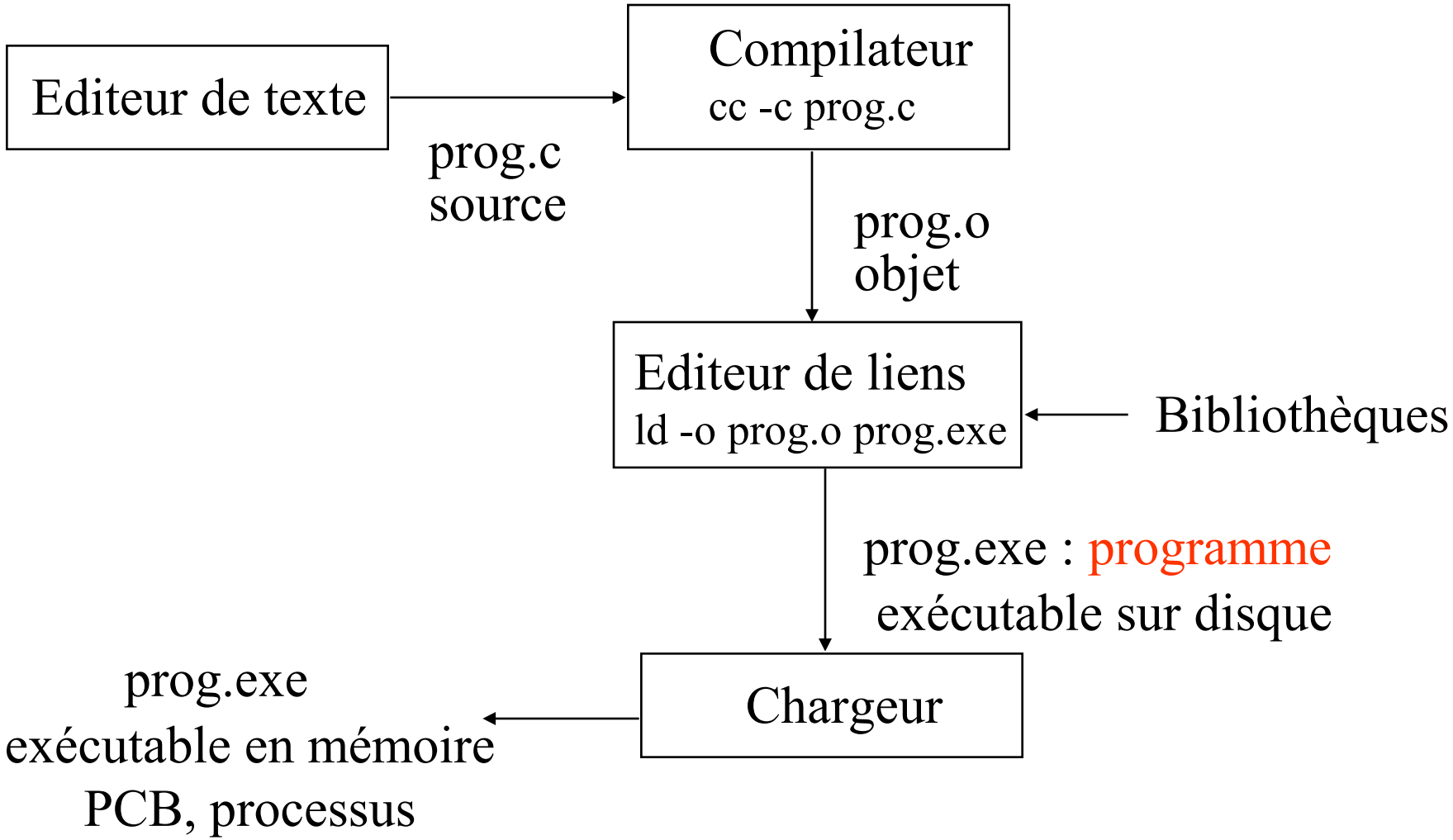
Passage en mode superviseur d'un programme utilisateur



PDF-XChange Viewer
Click to buy NOW!
www.docu-track.com

Processus
Ordonnancement

Du programme au processus



Notion de processus

- Définitions

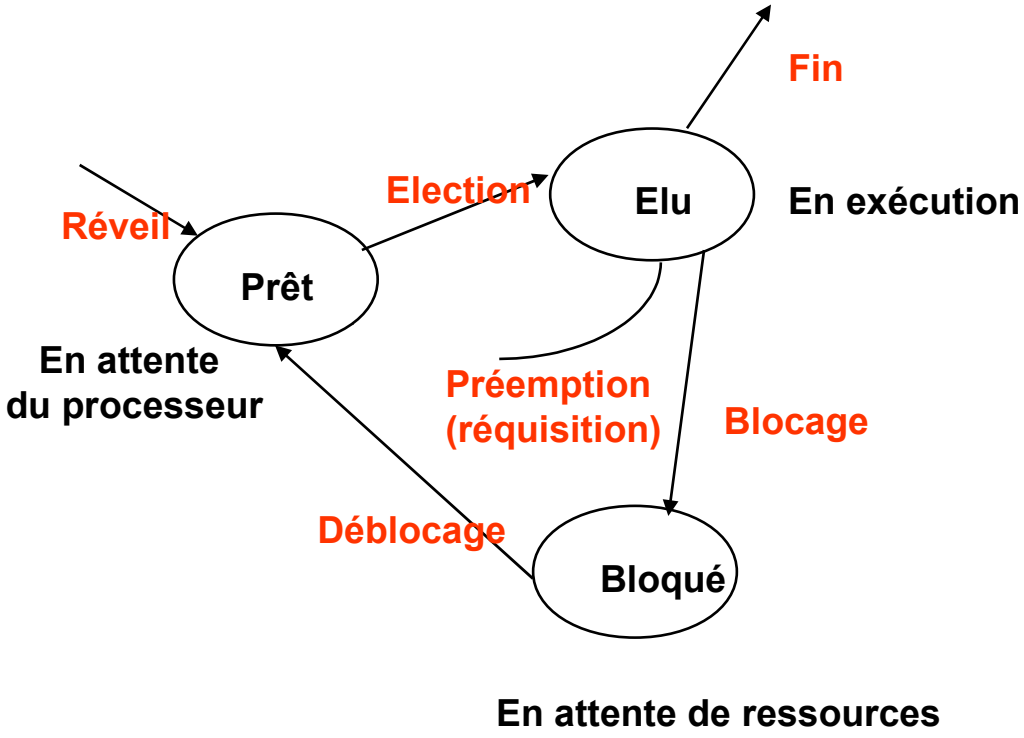
  – Un processus est un programme en cours d ’exécution auquel est associé un environnement processeur (CO, PSW, RSP, registres généraux) et un environnement mémoire appelés contexte du processus.

  – Un processus est l’instance dynamique d ’un programme et incarne le fil d ’exécution de celui-ci dans un espace d'adressage protégé (objets propres : ensemble des instructions et données accessibles)

bloc de contrôle de processus PCB

identificateur processus
état du processus
compteur instructions
contexte pour reprise (registres et pointeurs, piles,..)
pointeurs sur file d' attente et priorité(ordonnancement)
informations mémoire (limites et tables pages/segments
informations de comptabilisation et sur les E/S, périphériques alloués, fichiers ouverts,..

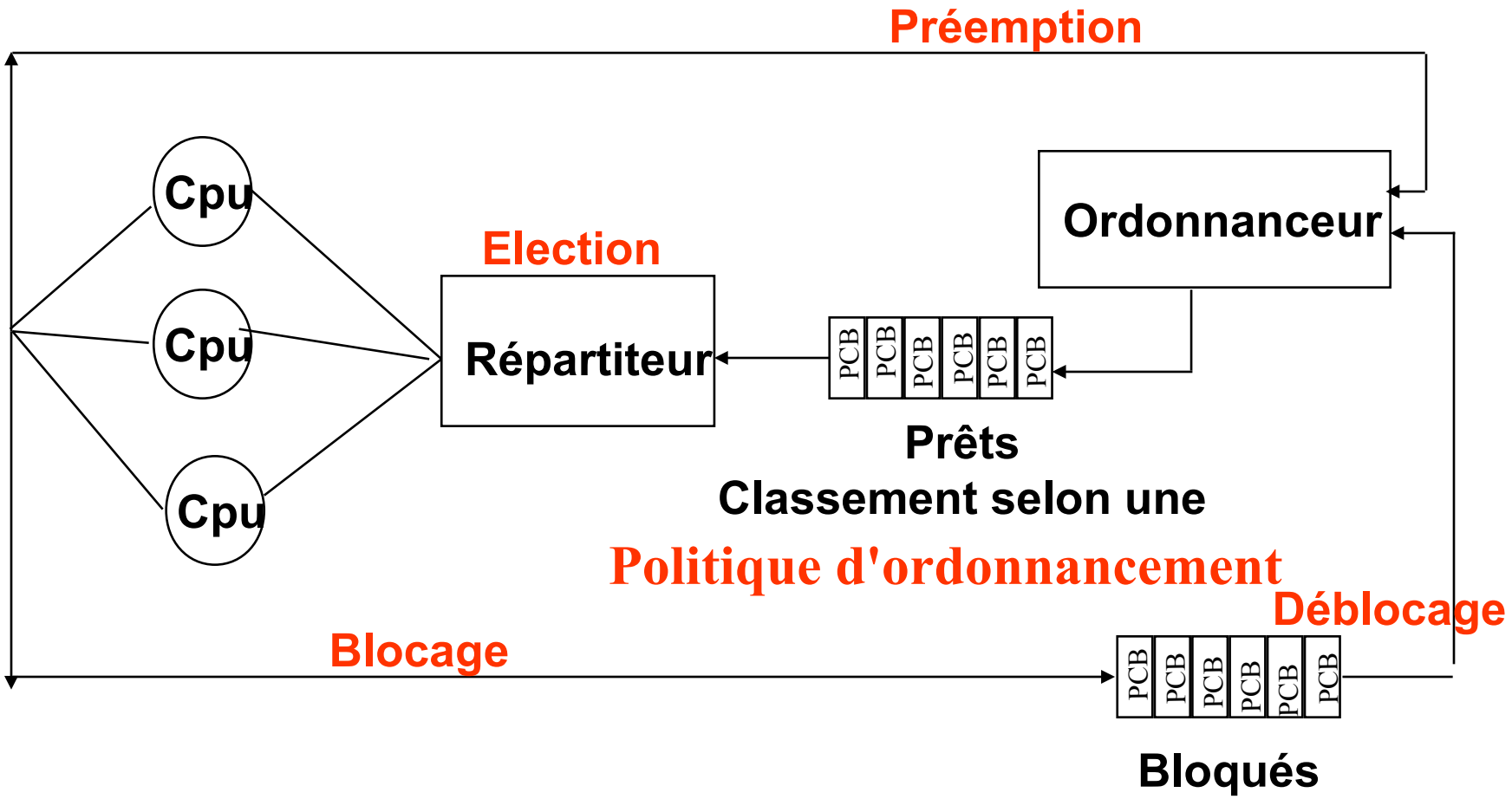
Bloc de contrôle de processus ou PCB



PDF-XChange Viewer
Click to buy NOW!
www.docu-track.com

Ordonnancement dans un système multiprocessus

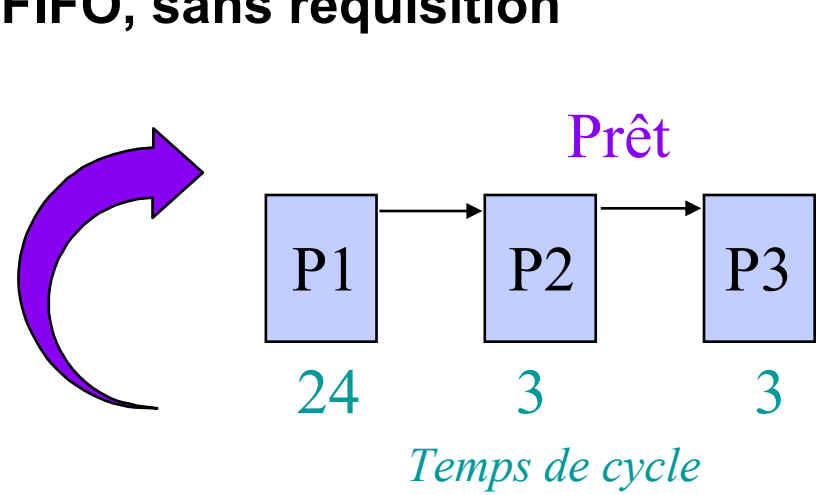
Système multiprocessus
Ordonnanceur et répartiteur

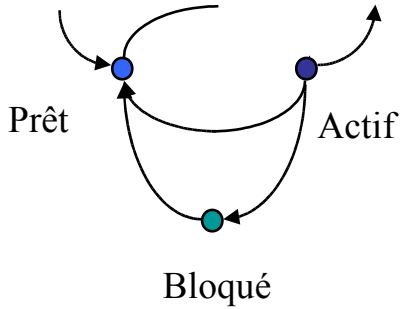


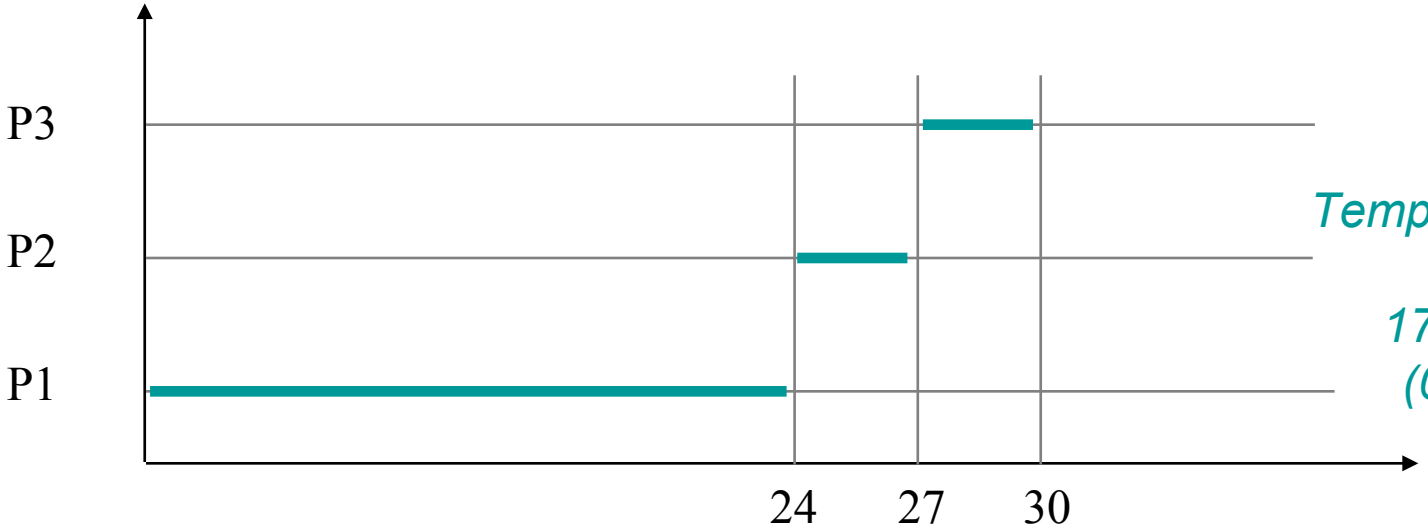
Algorithme : Premier Arrivé Premier Servi

PDF-XChange Viewer Click to buy NOW! www.docu-track.com

- **FIFO, sans réquisition**



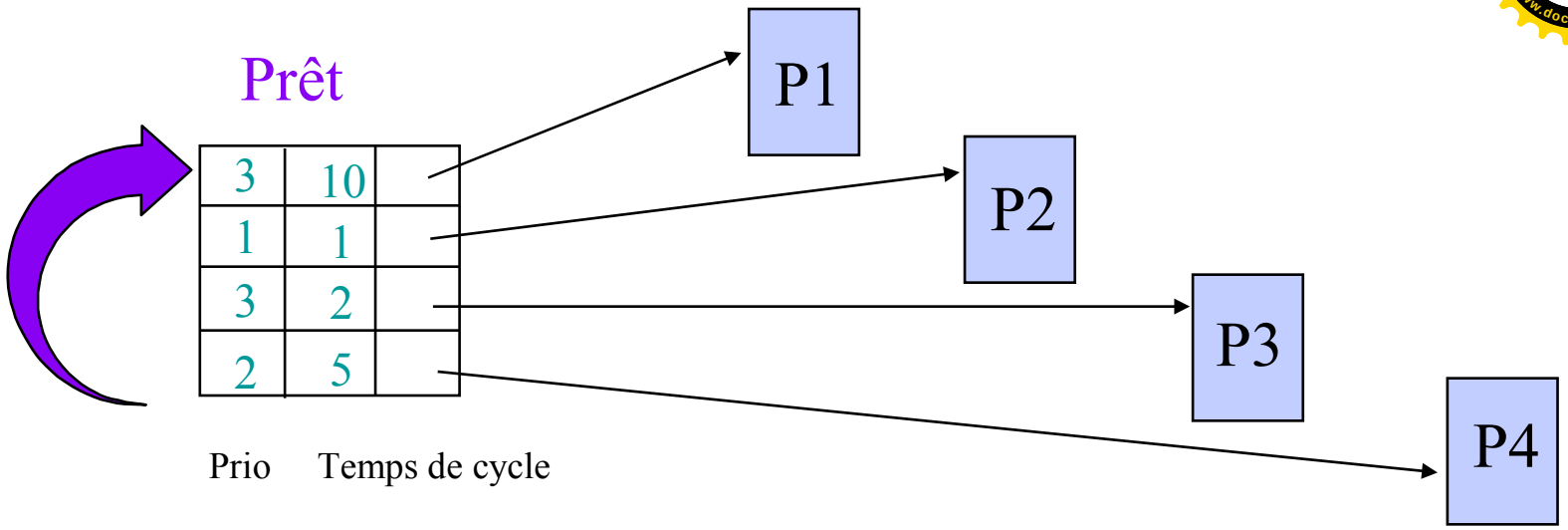




Temps moyen d'attente

17 millisecondes
(0 + 24 + 27)/3

Algorithme : avec priorités



↳ Priorité : le plus petite valeur correspond à la plus forte priorité

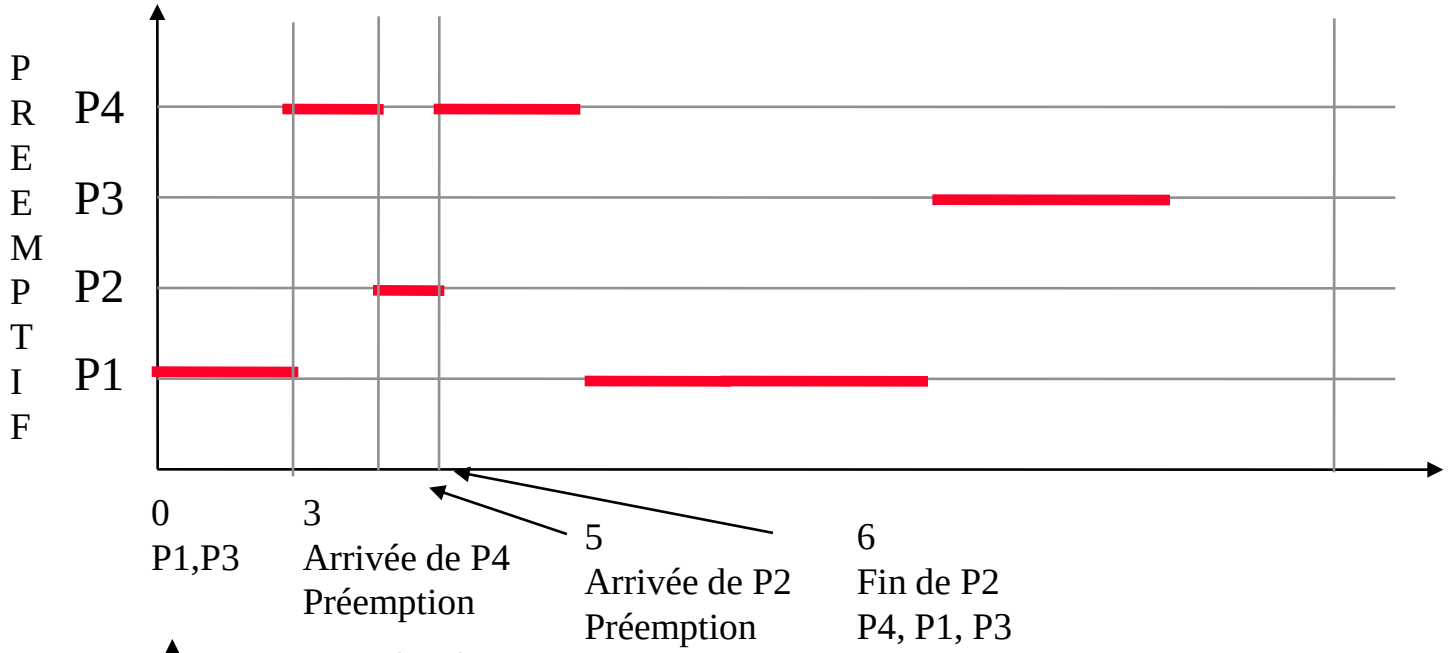
Temps moyen d'attente

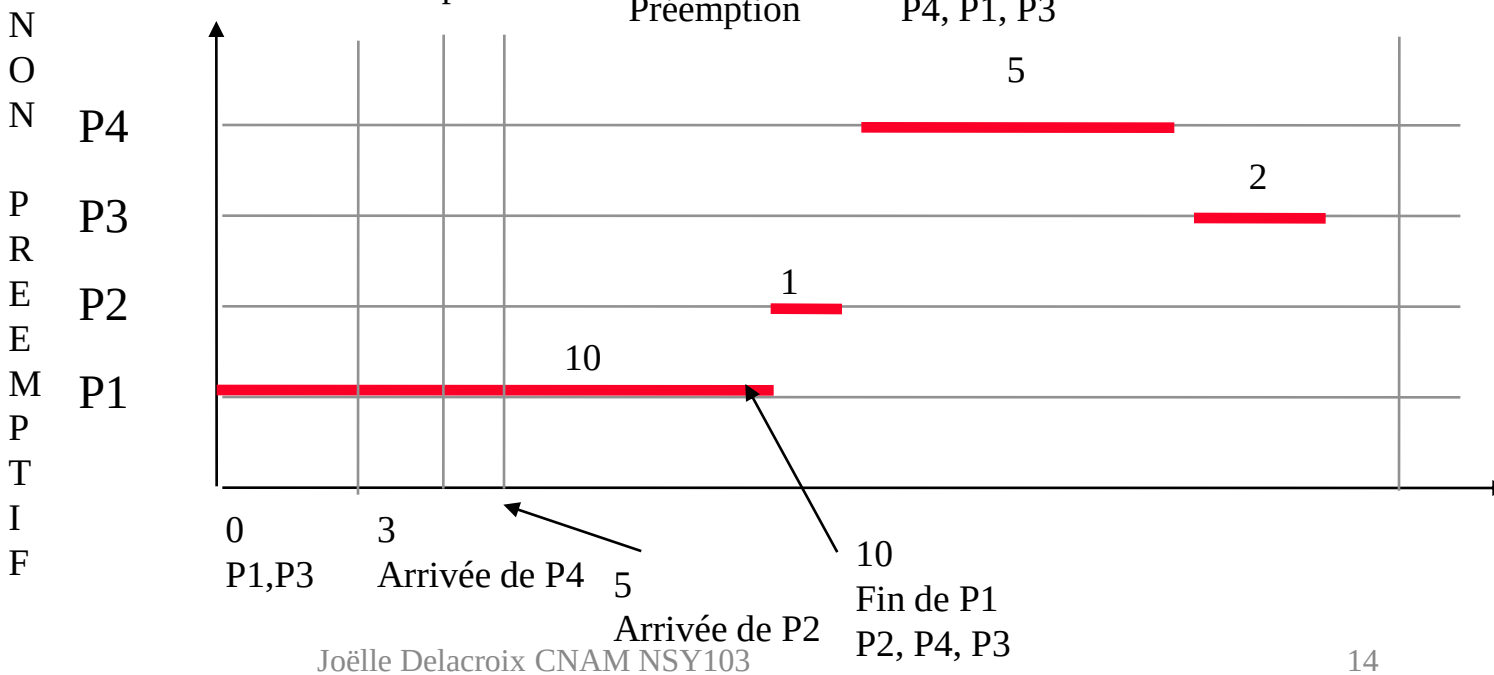
8,2 millisecondes

Politiques d'ordonnancement par priorité

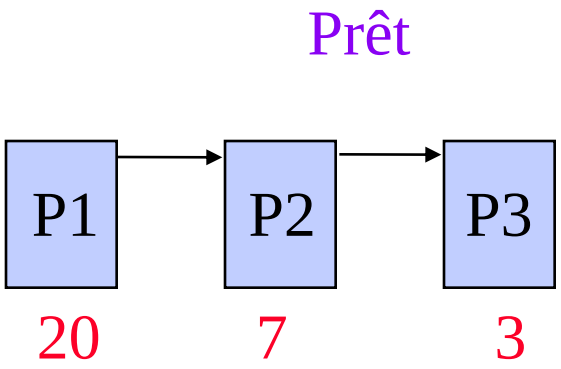
Prêt

Date de soumission	Tps de cycle	Prio	
0	3	10	P1
5	1	1	P2
0	4	2	P3
3	2	5	P4

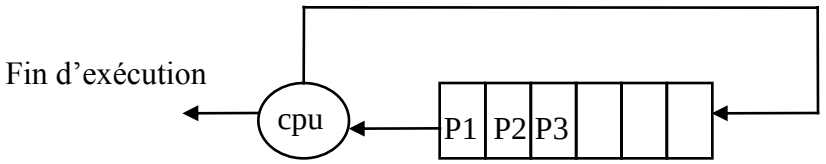


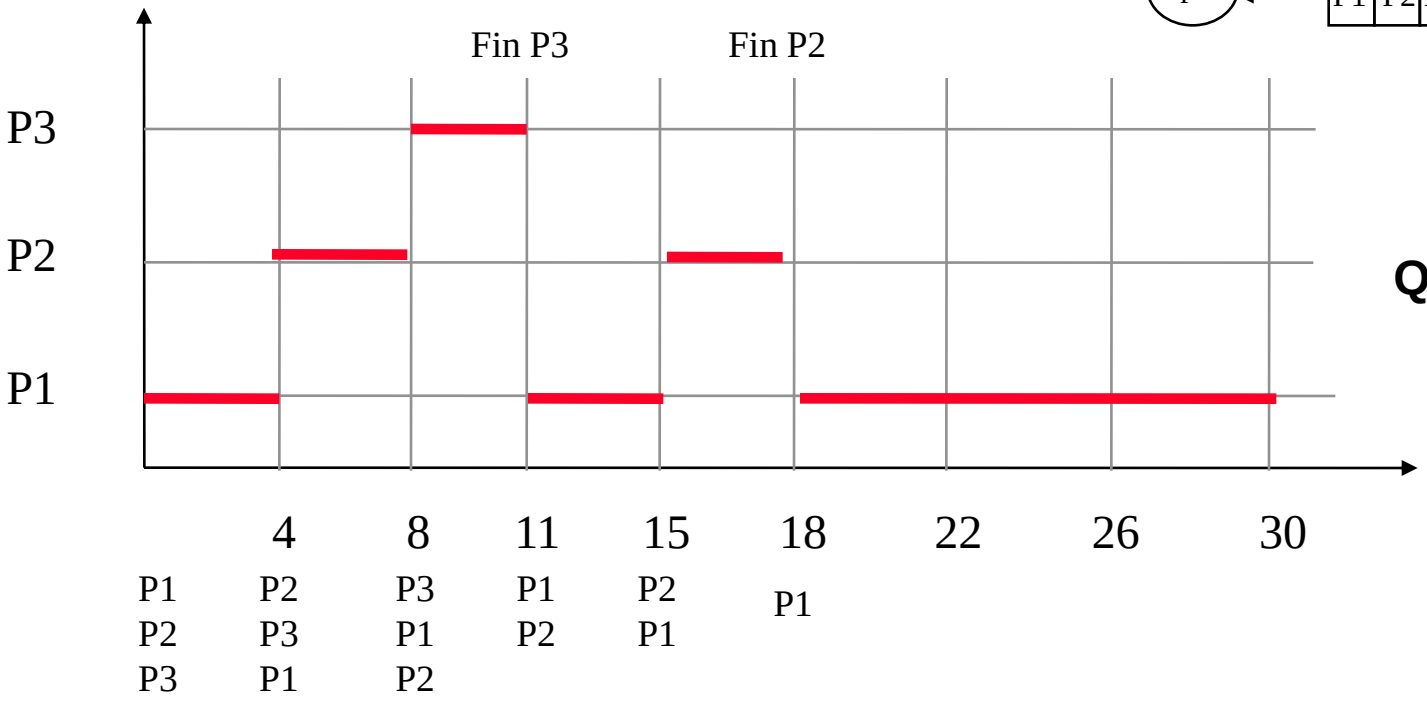


Algorithme : tourniquet



Un processus élu s'exécute au plus durant un quantum; à la fin du quantum, préemption et réinsertion en fin de file d'attente des processus prêts

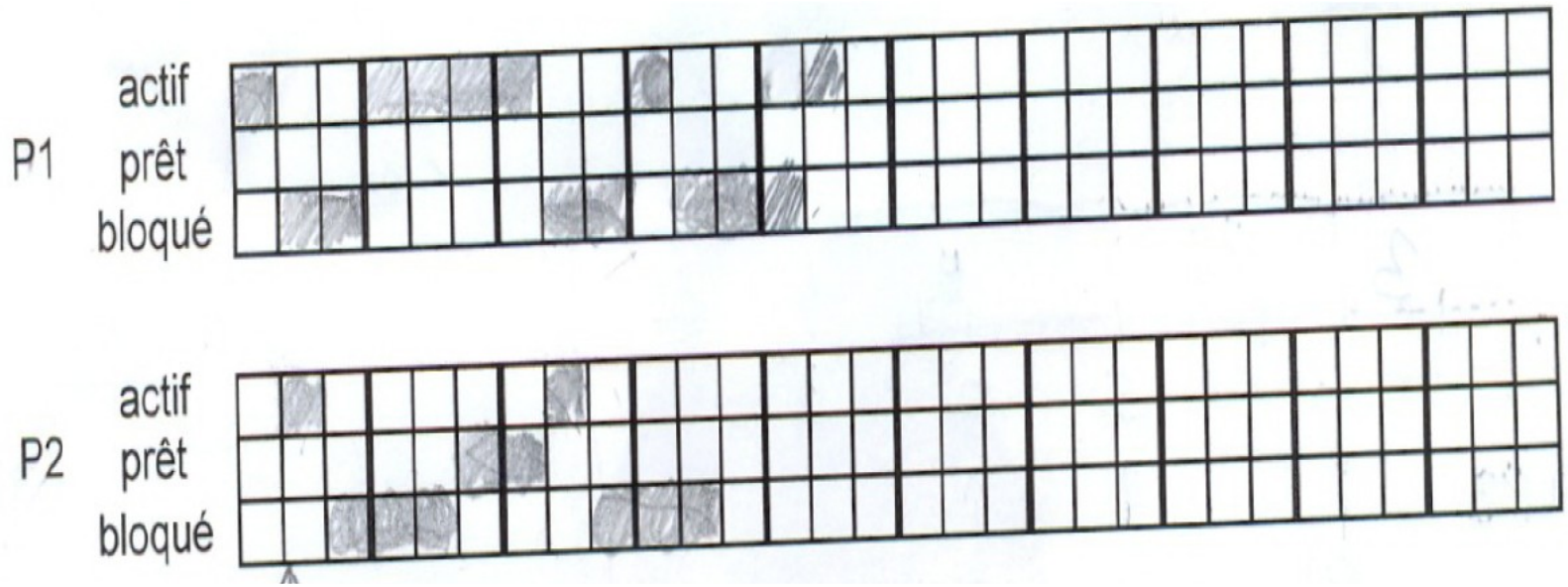




Quantum = 4

Ordonnancement

- On considère les deux processus suivants, P1 et P2 qui effectuent du calcul et des entrées-sorties avec un disque. Les entrées-sorties consistent à lire ou écrire un bloc du disque (B1 ou B2) ce qui demande 20 ms. Lors de la fin d'une entrée-sortie pour un processus, celui-ci est mis en bout de la file des processus prêts.
- Processus P1
  - calcul 10 ms
  - lecture B1
  - calcul 40 ms
  - écriture B1
  - calcul 10 ms
  - lecture B2
  - calcul 10 ms
- Processus P2
  - calcul 10 ms
  - lecture B1
  - calcul 10 ms
  - écriture B1
- Le processus P1 est lancé au temps 0, et **le processus P2 est lancé 10 ms après**. Donner le chronogramme correspondant en colorant les cases du diagramme suivant.



PDF-XChange Viewer
Click to buy NOW!
www.docu-track.com

2. Processus Unix / Linux

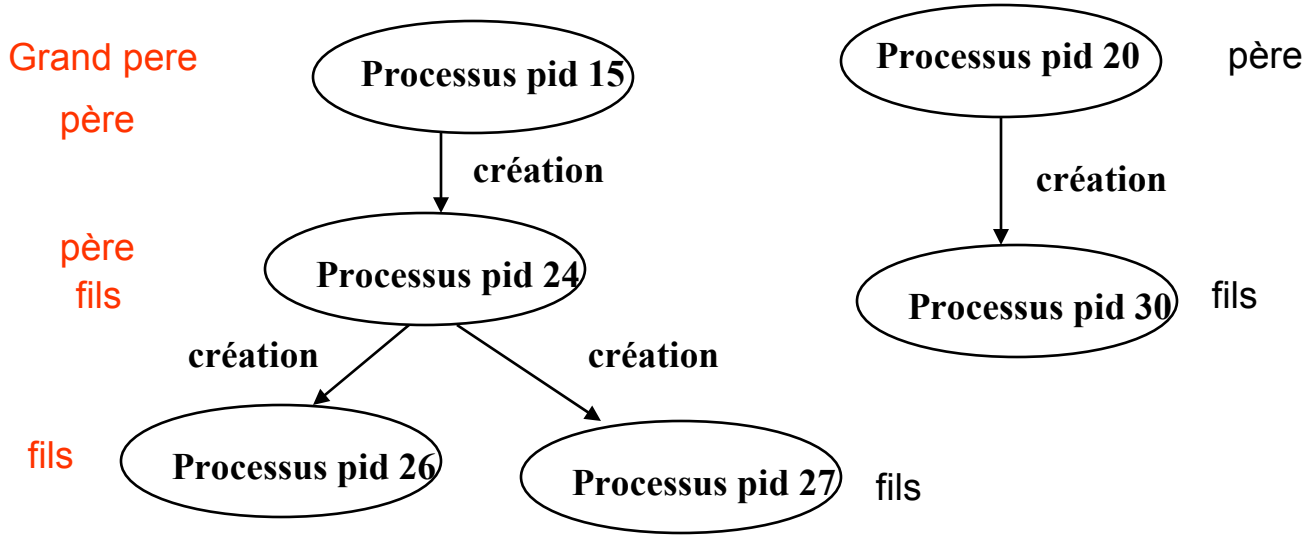
Le processus est identifié par un pid

Processus Unix / Linux

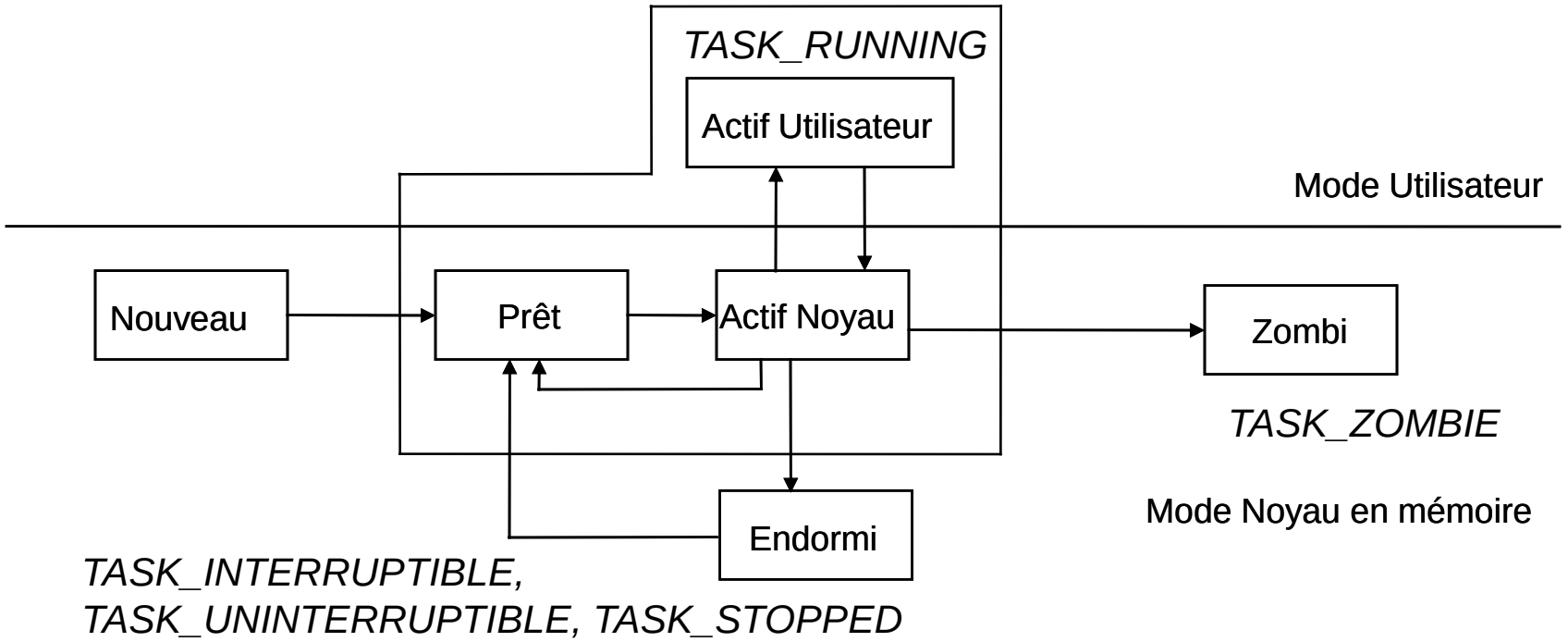
PDF-XChange Viewer Click to buy NOW! www.docu-track.com

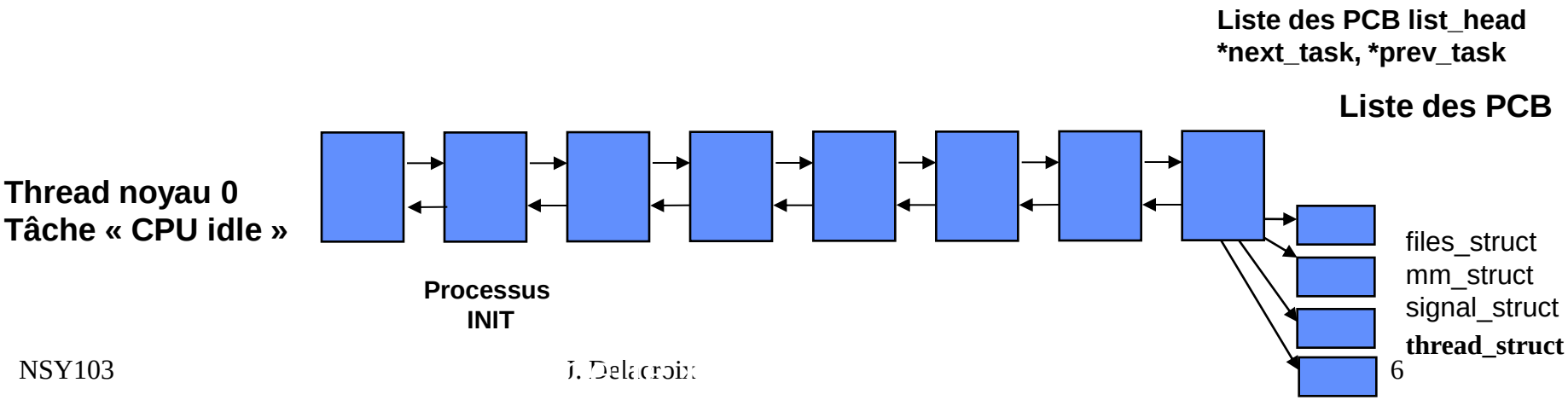
- Tout processus Linux peut créer un autre processus Linux

  - Arborescence de processus avec un rapport père - fils entre processus créateur et processus crée



Processus Linux





Primitive de création de processus

PDF-XChange Viewer Click to buy NOW! www.docu-track.com

- Primitive fork

```
#include <unistd.h>
pid_t fork(void)
```

- La primitive fork() permet la création dynamique d ’un nouveau processus qui s ’exécute de manière concurrente avec le processus qui l ’a créé.
- Tout processus Unix/Linux hormis le processus 0 est crée à l ’aide de cette primitive.
- Le processus créateur (le père) par un appel à la primitive fork() crée un processus fils qui est une copie exacte de lui-même (code et données)

Primitive de création de processus

PDF-XChange Viewer Click to buy NOW! www.docu-track.com

MODE UTILISATEUR

MODE SYSTEME

PROCESSUS PID 12222

```
Main ()
{
pid_t ret ;
int i, j;

for(i=0; i<8; i++)
        i = i + j;

ret = fork();

 12224


}
```

PROCESSUS PID 12224

```
Main ()
{
pid_t ret ;
int i, j;

for(i=0; i<8; i++)
        i = i + j;

ret = fork();
```

0

Exécution de l ’appel système fork

Si les ressources noyau sont disponibles

retourner le pid du processus crée à son père

et 0 au processus fils

Primitive de création de processus

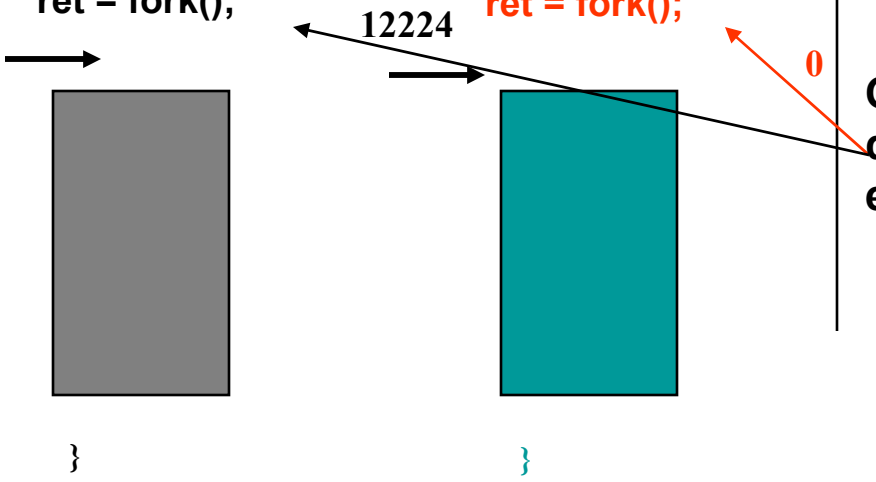
PDF-XChange Viewer Click to buy NOW! www.docu-track.com

PROCESSUS
PID 12222

```
Main ()
{
pid_t ret ;
int i, j;

for(i=0; i<8; i++)
        i = i + j;

ret = fork();
```

PROCESSUS
PID 12224

```
Main ()
{
pid_t ret ;
int i, j;

for(i=0; i<8; i++)
        i = i + j;

ret = fork();
```



- **Chaque processus père et fils reprend son exécution après le fork()**

- **Le code et les données étant strictement identiques, il est nécessaire de disposer d 'un mécanisme pour différencier le comportement des deux processus après le fork()**

On utilise pour cela le code retour du fork() qui est différent chez le fils (toujours 0) et le père (pid du fils crée)

Primitive de création de processus

PROCESSUS
PID 12222

```
Main ()
{
pid_t ret ;
int i, j;

for(i=0; i<8; i++)
          i = i + j;

ret= fork();

if (ret == 0)
   printf(" je suis le fils ")
else
{
     printf ("je suis le père");
     printf ("pid de mon fils %d", ret)
 }
}
```

Pid du fils : 12224
getpid : 12222
getppid :shell

PROCESSUS
PID 12224

```
Main ()
{
pid_t ret ;
int i, j;

for(i=0; i<8; i++)
          i = i + j;

ret= fork();

if (ret == 0)
  printf(" je suis le fils ")
else
{
     printf ("je suis le père");
     printf ("pid de mon fils, %d" , ret)
 }
}
```

getpid : 12224
getppid :12222

Synchronisation père / fils

PDF-XChange Viewer Click to buy NOW! www.docu-track.com

- Primitive exit()

```
#include <stdlib.h>
void exit (int valeur);
pid_t wait (int *status);
```

  - un appel à la primitive exit() provoque la terminaison du processus effectuant l'appel avec un code retour *valeur*. (Par défaut, le franchissement de la dernière } d'un programme C tient lieu d'exit)

  - un processus qui se termine passe dans l'état Zombie et reste dans cet état tant que son père n'a pas pris en compte sa terminaison

  - Le processus père "récupère" la terminaison de ses fils par un appel à la primitive wait ()

Primitive de création de processus

PROCESSUS
PID 12222

```
Main ()
{
pid_t ret ;
int i, j;

for(i=0; i<8; i++)
          i = i + j;
ret= fork();
if (ret == 0)
{
        printf(" je suis le fils ");
        exit(); }
else
{
     printf ("je suis le père");
     printf ("pid de mon fils, %d" , ret)
     wait();
 }

}
```

PROCESSUS
PID 12224

```
Main ()
{
pid_t ret;
int i, j;

for(i=0; i<8; i++)
          i = i + j;
ret= fork();

if (ret== 0)
{
        printf(" je suis le fils ");
        exit; }
else
{
     printf ("je suis le père");
     printf ("pid de mon fils, %d" ,ret);
     wait();
 }

}
```

Primitives de recouvrement (execl)

#include <sys/types.h>
#include <sys/wait.h>
int execl(const char *ref, const char *arg, …, NULL)

PROCESSUS

Chemin absolu du code exécutable

arguments

```
Main ()
{
pid_t pid ;
int i, j;

for(i=0; i<8; i++)
        i = i + j;

pid= fork();

if (pid == 0)
{
    printf(" je suis le fils ");
    execl("/home/calcul","calcul","3","4", NULL);
        executable      paramètres
}
else
{
    printf ("je suis le père");
    printf ("pid de mon fils, %d" , pid);
    wait();
 }

}
```

Création d'un processus fils
par duplication du code et données du père

Le processus fils recouvre le code et les données hérités du père par ceux du programme calcul.
Le père transmet des données de son environnement vers son fils par les paramètres de l'exec

Le père attend son fils

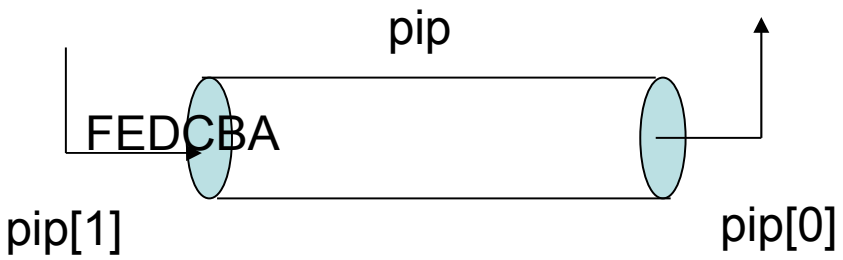
PDF-XChange Viewer
Click to buy NOW!
www.docu-track.com

3. Communication entre processus

Outils de communication

- **Communication entre processus d'une même machine**
  - **Les tubes anonymes**
  - **Les files de messages**
  - **Les segments de mémoires partagées**
- **Communication entre processus distants**
  - **Les sockets**

Exemple tubes anonymes

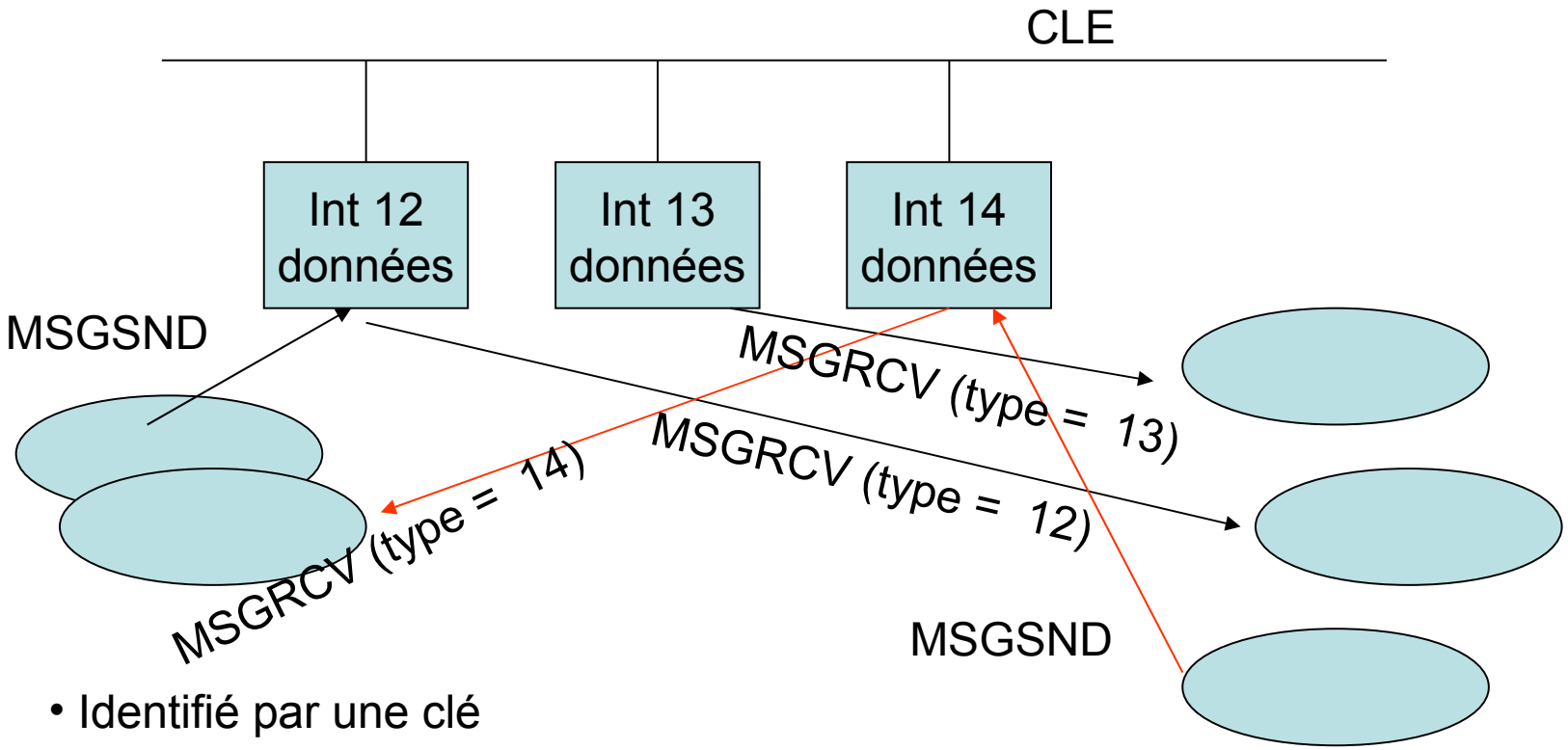



```
main ()  {
        int pip[2];
        pid_t retour;
        char chaine[7];
pipe(pip);   -- création de tube
pid = fork();   -- création fils
if (retour == 0)
        { -- le fils ecrit dans le tube
        close (pip[0]); -- pas de lecture sur le tube
        write (pip[1], "bonjour", 7);
        close (pip[1]);
        }
else
        { -- le pere lit le tube
        close (pip[1]); -- pas d'ecriture sur le tube
        read(pip[0], chaine, 7);
        close (pip[0]);
        }
```

- Identifié et accessible par les descripteurs en lecture et écriture (création, héritage)
- Entre processus de même famille
- Pas de conservation des structures
- Unidirectionnel, FIFO, lecture destructive

- Erreur : écriture dans un tube sans lecteurs

Exemple Files de messages



- Identifié par une clé
- Entre processus quelconque connaissant le clé
- Bidirectionnel
- Conservation des structures
- Multiplexage

Exemple Files de messages

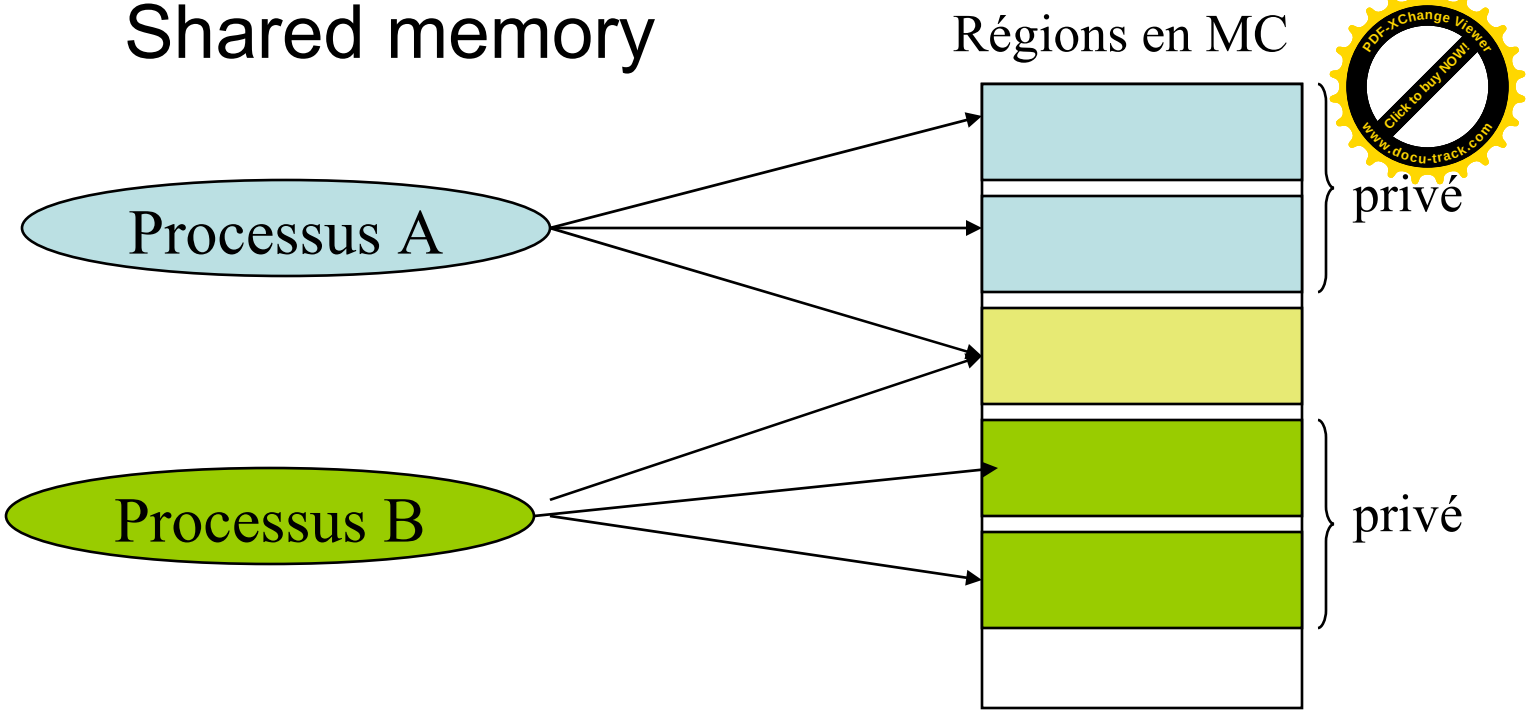
PROCESSUS A

```
#include  <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#define cle 17
struct msgbuf_exo {long mtype;
                   char mtext[20]; }
struct msgbuf_exo msgp;
main ()  {
int msq id ; /* identifiant de la MSQ */

/* allocation de la msq */
msqid = msgget (cle, IPC_CREAT |
     IPC_EXCL | 0666);

/* ecriture message dans la msq */
msgp.mtype  =12;   on associe un type
     au message

strcpy(msgp.mtext, "ceci est un
     message");

msgsnd (msqid, &msgp,
     strlen(msgp.mtext), 0);
}
```

PROCESSUS B

```
#include  <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#define cle 17
struct msgbuf_exo {long type;
                   char mtext[20]; }
struct msgbuf_exo msgp;
main ()  {
int msqid ; /* identifiant de la MSQ */

/* recuperation de la msq */
msqid = msgget (cle, 0);

/* lecture message dans la msq de type
     12*/
msgrcv (msqid, &msgp, 19, 12, 0);

/* destruction de la msq */
msgctl (msqid, IPC_RMID, NULL);
}
```

Shared memory



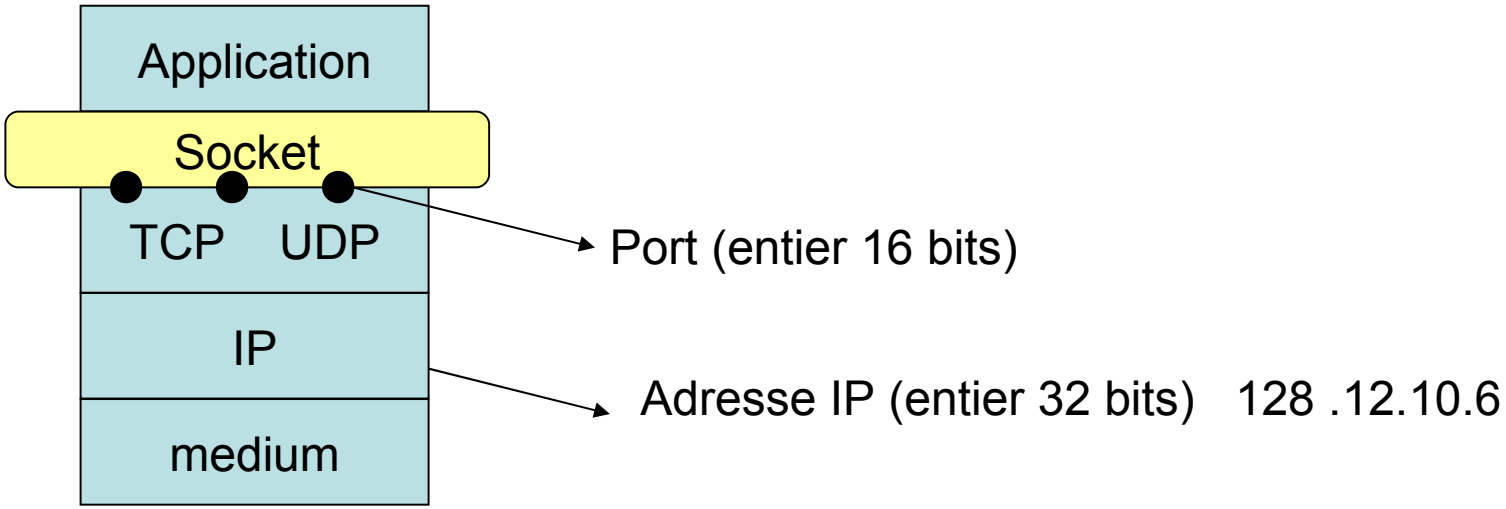
- Région de mémoire pouvant être partagée entre plusieurs processus
- Un processus doit attacher le région à son espace d'adressage avant de pouvoir l'utiliser
- Outil IPC repéré par une clé unique
- L'accès aux données présentes dans la région peut requérir une synchronisation (outil sémaphore)

```
/**********************************/
/* processus créateur du segment et
civation          */
/**********************************/
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

#define CLE         256

main()
{
int shmid;
char *mem ;

/* création du segment de mémoire
partagée avec la clé CLE */
shmid=shmget((key_t)CLE,1000,0750 |
IPC_CREAT | IPC_EXCL);

/* attachement */
mem=shmat(shmid,NULL,0);

/* écriture dans le segment */
strcpy(mem,"voici une écriture dans le
segment");
exit(0);
}
```



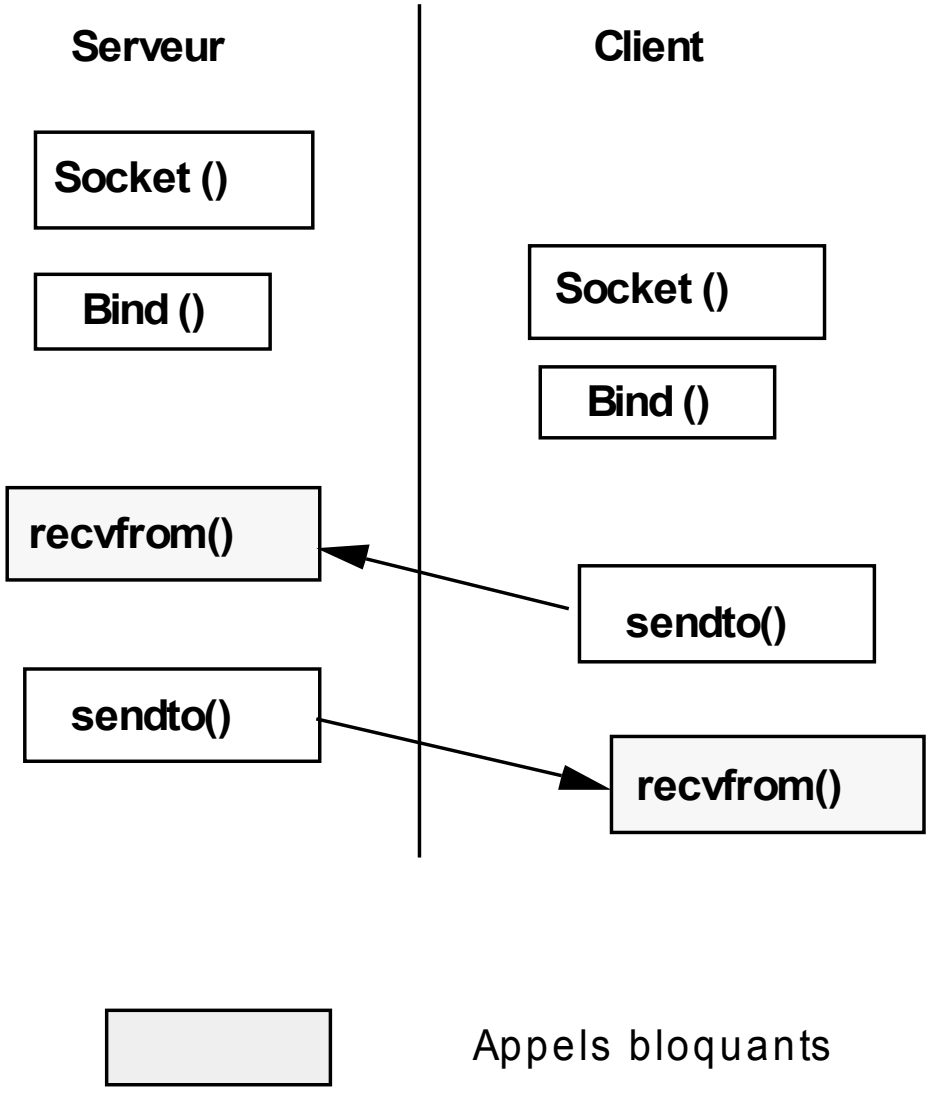
```
/**********************************/
/* processus destructeur du segment et
lecteur          */
/**********************************/
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

#define CLE         256

main()
{
/* récupération du segment de mémoire */
shmid=shmget((key_t)CLE,0,0);

/* attachement */
mem=shmat(shmid,NULL,0);

/* lecture dans le segment */
printf("lu: %s\n",mem);

/* détachement du processus */
shmdt(mem);

/* destruction du segment */
shmctl(shmid,IPC_RMID,NULL);

exit(0)
}
```

Les sockets



- Interface de programmation
- Point de communication réseau : adresse d'application (adresse IP, n°port)
- Associée à un protocole et un mode de communication
  - Datagramme (UDP) (AF_INET, SOCK_DGRAM)
  - Connecté (TCP) (AF_INET, SOCK_STREAM)

L'interface socket : Communication en mode datagramme

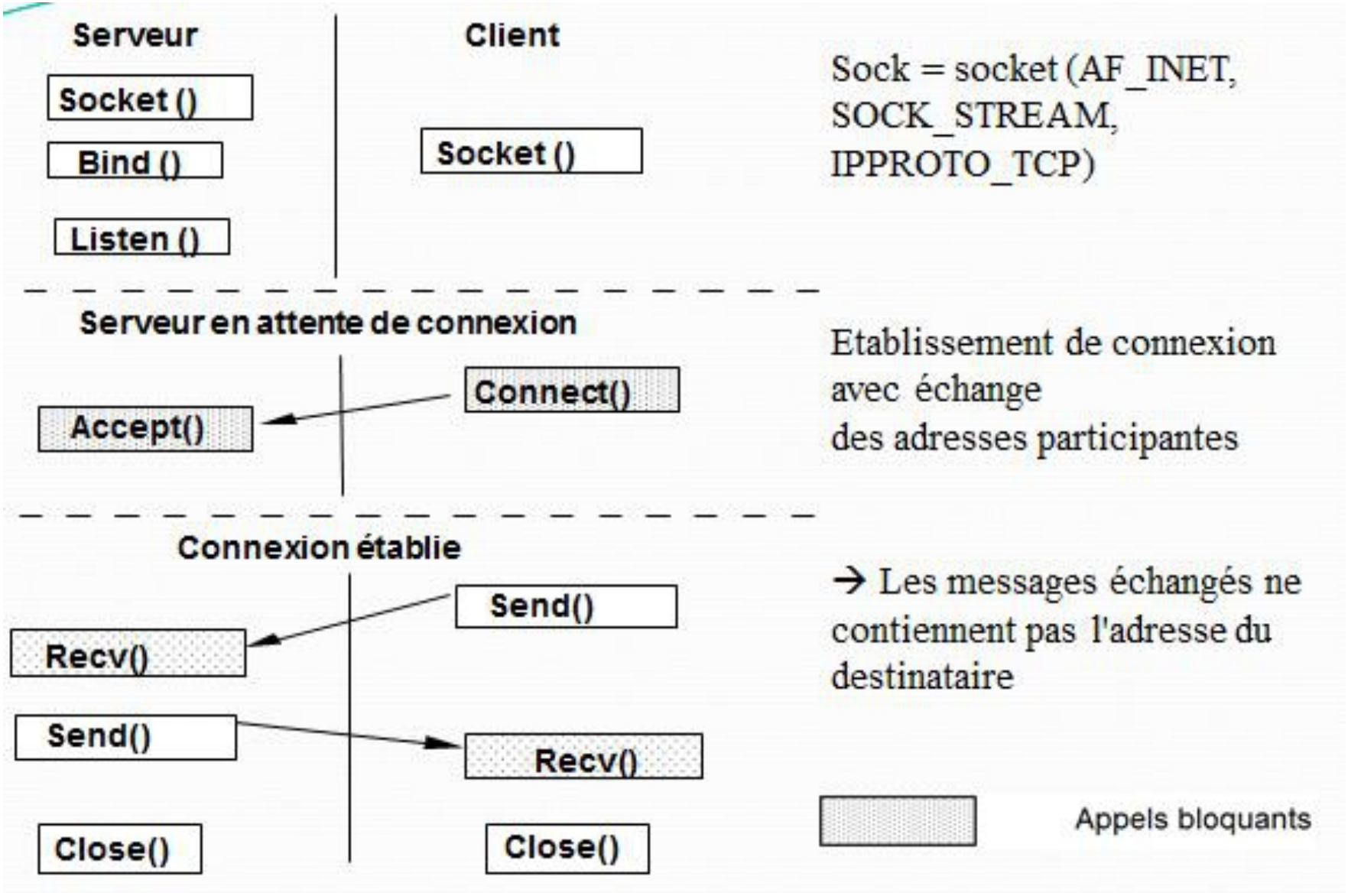


Sock = socket (AF_INET, SOCK_DGRAM, IPPROTO_UDP)

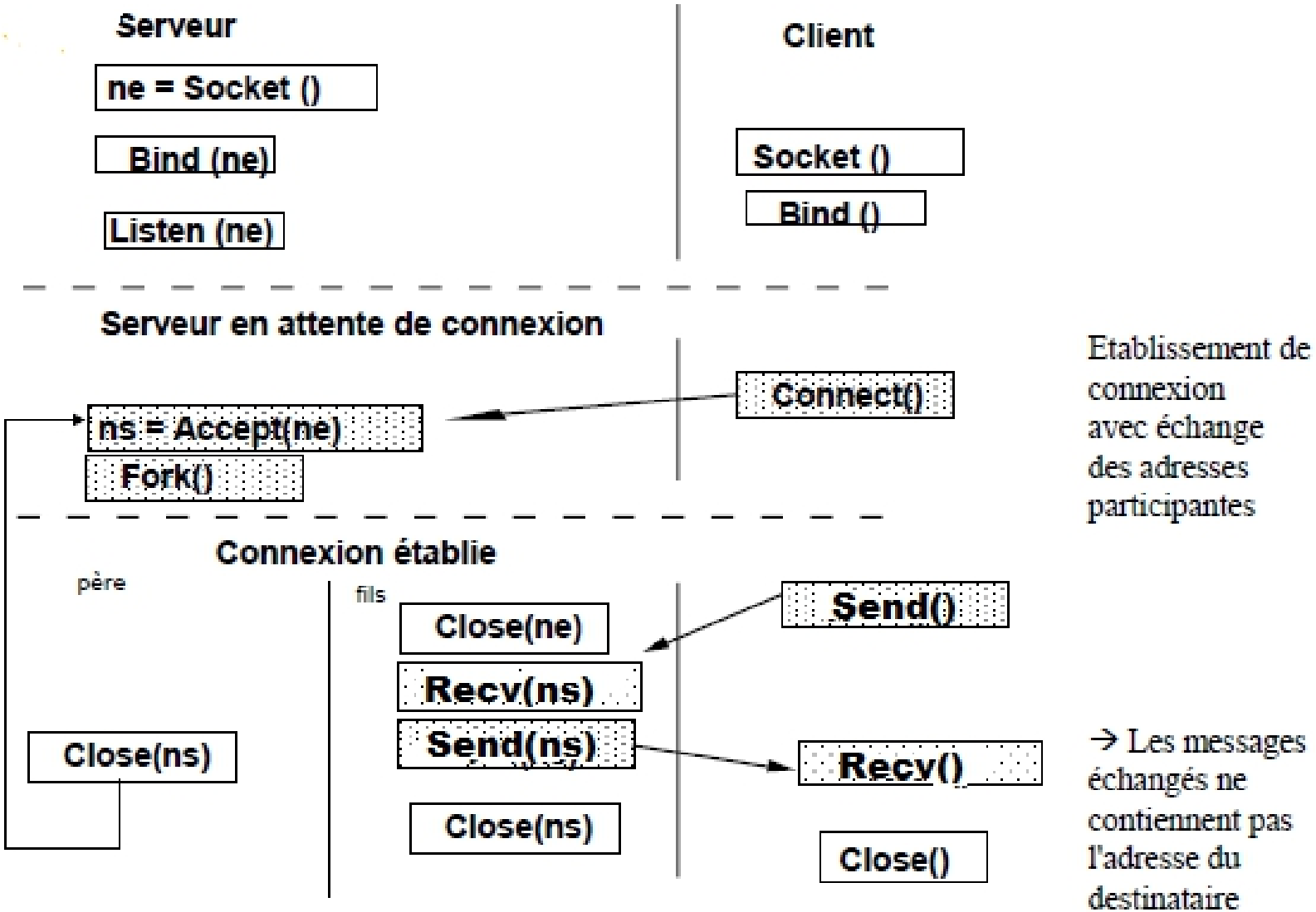
Pas d'établissement de connexion

→ Chaque message échangé contient l'adresse du destinataire

L'interface socket : Communication en mode connecté



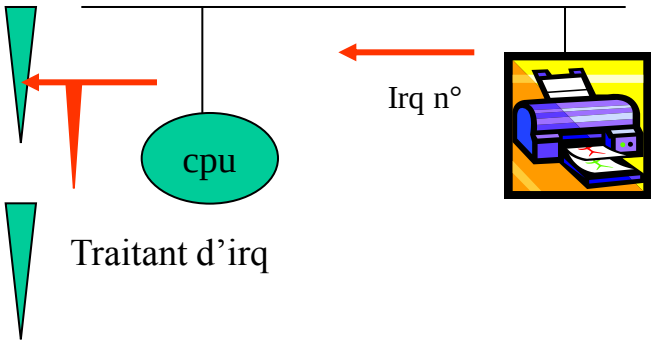
L'interface socket : Communication en mode connecté

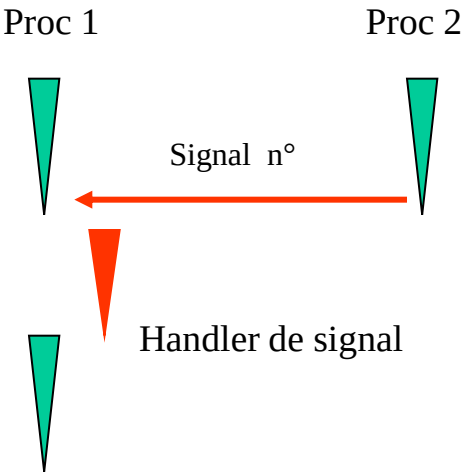


Outils de communication entre processus
Les signaux

Signaux : définition

- Le signal est une interruption logicielle délivrée à un processus.
  - Il informe les processus de l'occurrence d'événements asynchrones et permet de faire exécuter à un processus une action relative à ces événements.
  - Il ne transporte pas de données





- 64 signaux identifiés par un numéro et un nom (SIG*X*)
  - 1 à 31 : signaux classiques
  - 32 à 63 : signaux temps réel

Traitement associé à un signal

- A tout signal est associé un traitement par défaut (ignorer, terminer le processus avec ou sans core, stopper le processus) qui est exécuté lors de la prise en compte du signal par le processus;

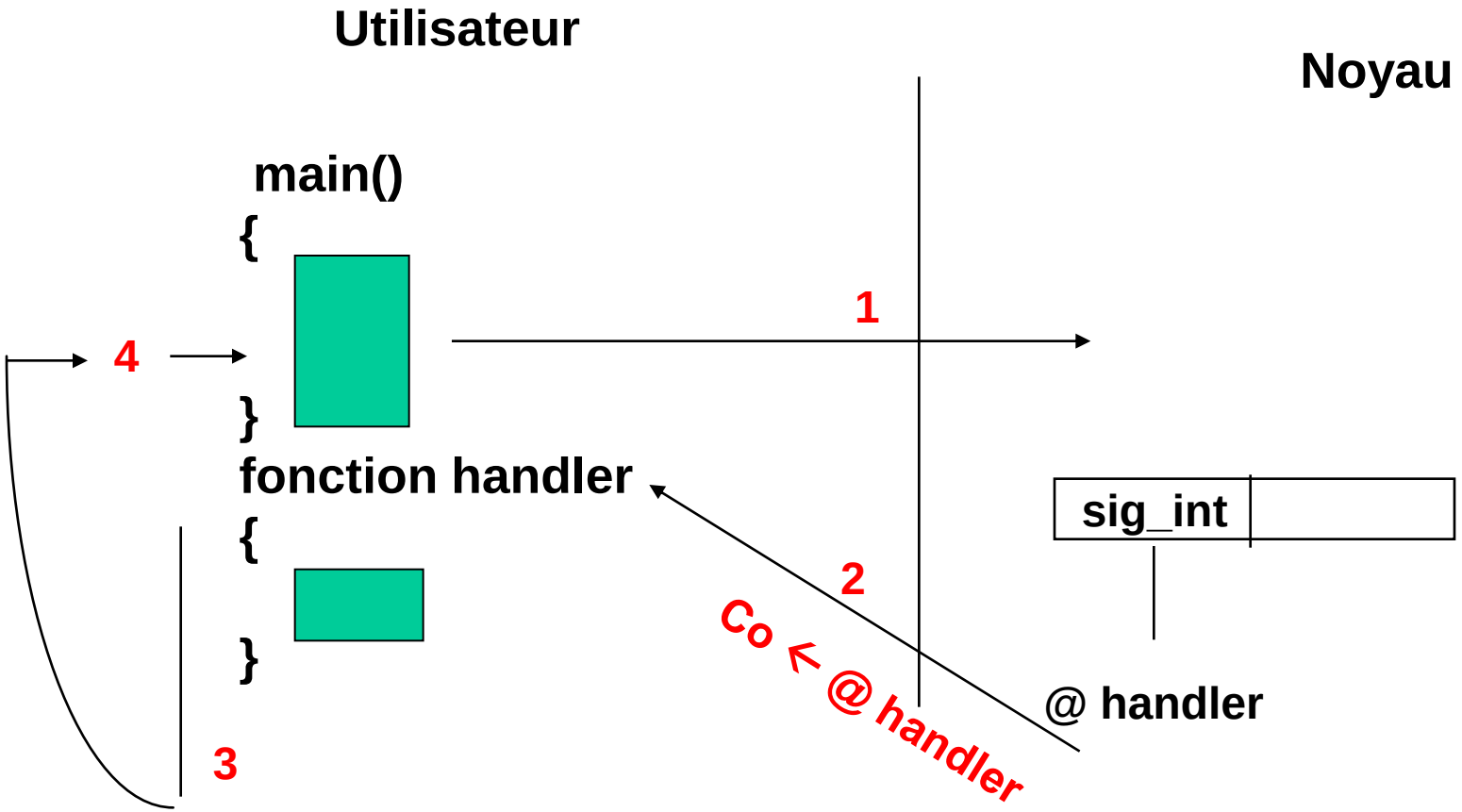
- Tout processus peut installer pour chaque type de signal (hormis SIGKILL), un nouveau traitement appelé handler
  - handler SIG_IGN : pour ignorer le signal (sauf mort du fils pour Linux)
  - handler fonction utilisateur pour capter le signal
    - ↳ fonction signal et sigaction

Les fonctions liées aux signaux

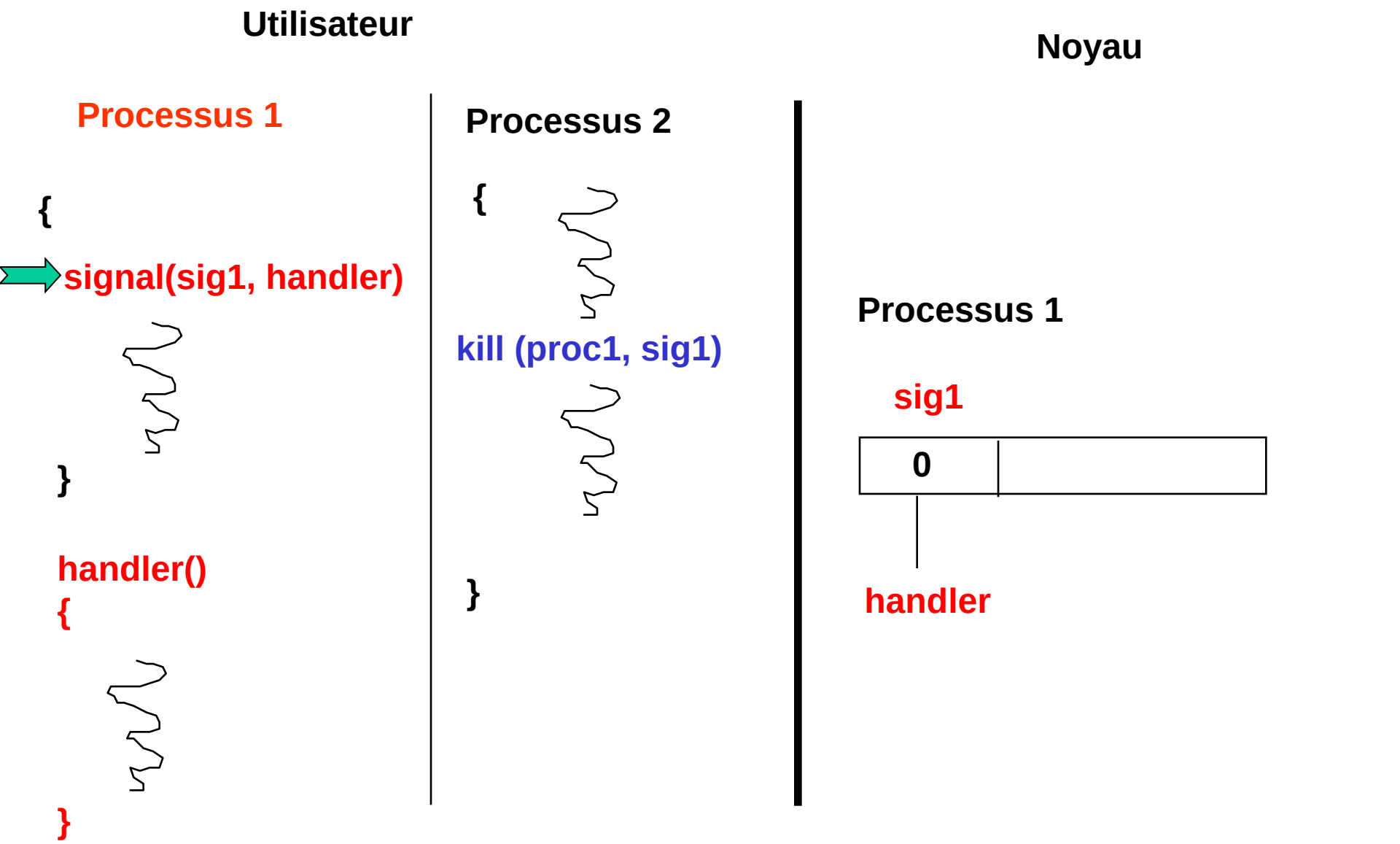
- Envoyer un signal à un processus
  - int kill (pid_t pid, int sig) kill (12563, SIGKILL)
  - kill -n°signal pid kill -9 12563
- Associer un handler à un signal
  - signal(int sig, fonction) signal(SIGINT, p_hand)
  - sigaction(int sig, struct sigaction action, NULL)
- Armer une temporisation
  - int alarm (int seconds) alarm(10)
  - au bout de *seconds* unités de temps, le signal SIGALRM est envoyé au processus
- Attendre un signal
  - int pause();

Exécution d'un handler de signal défini par l'utilisateur

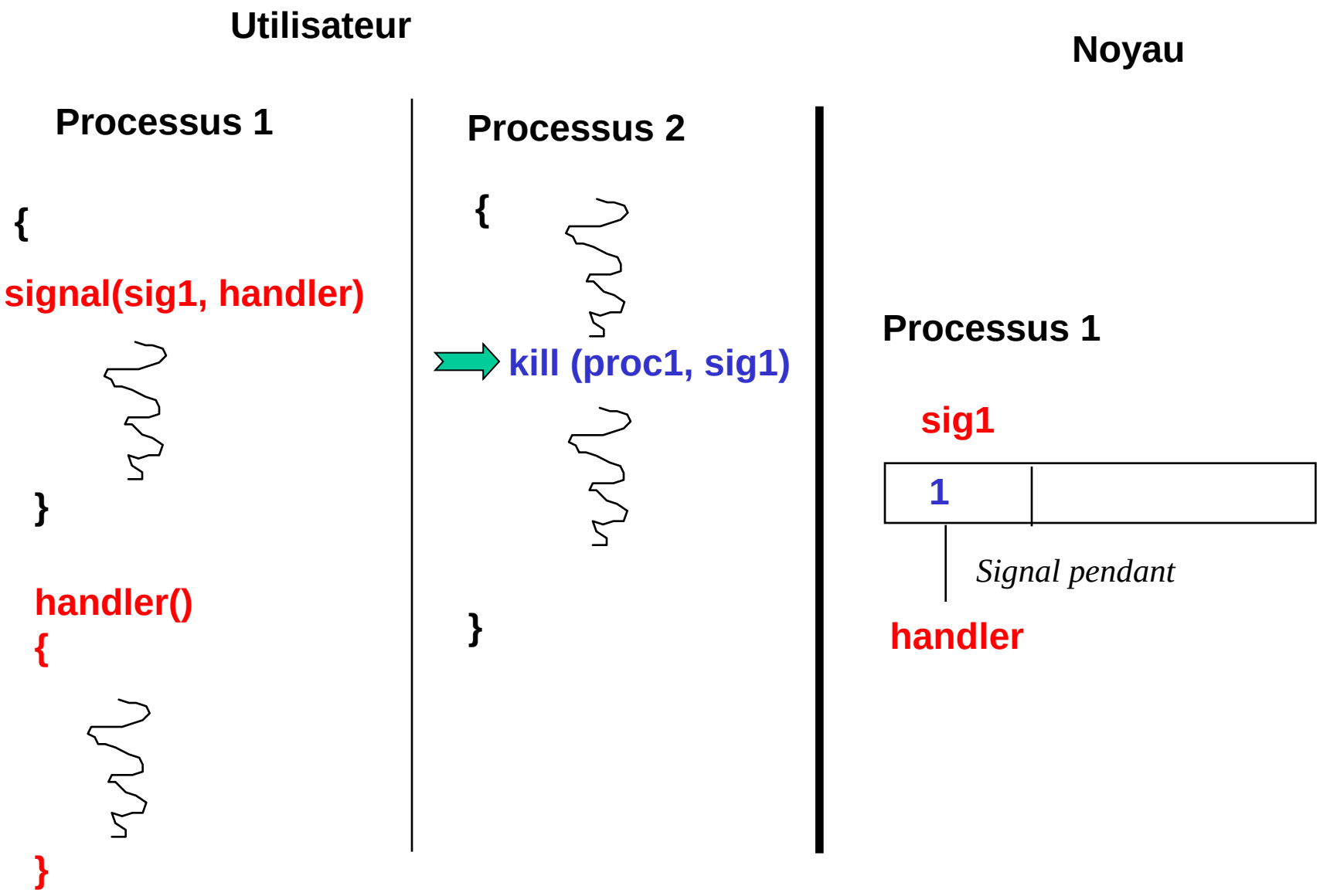
- Traitement des signaux



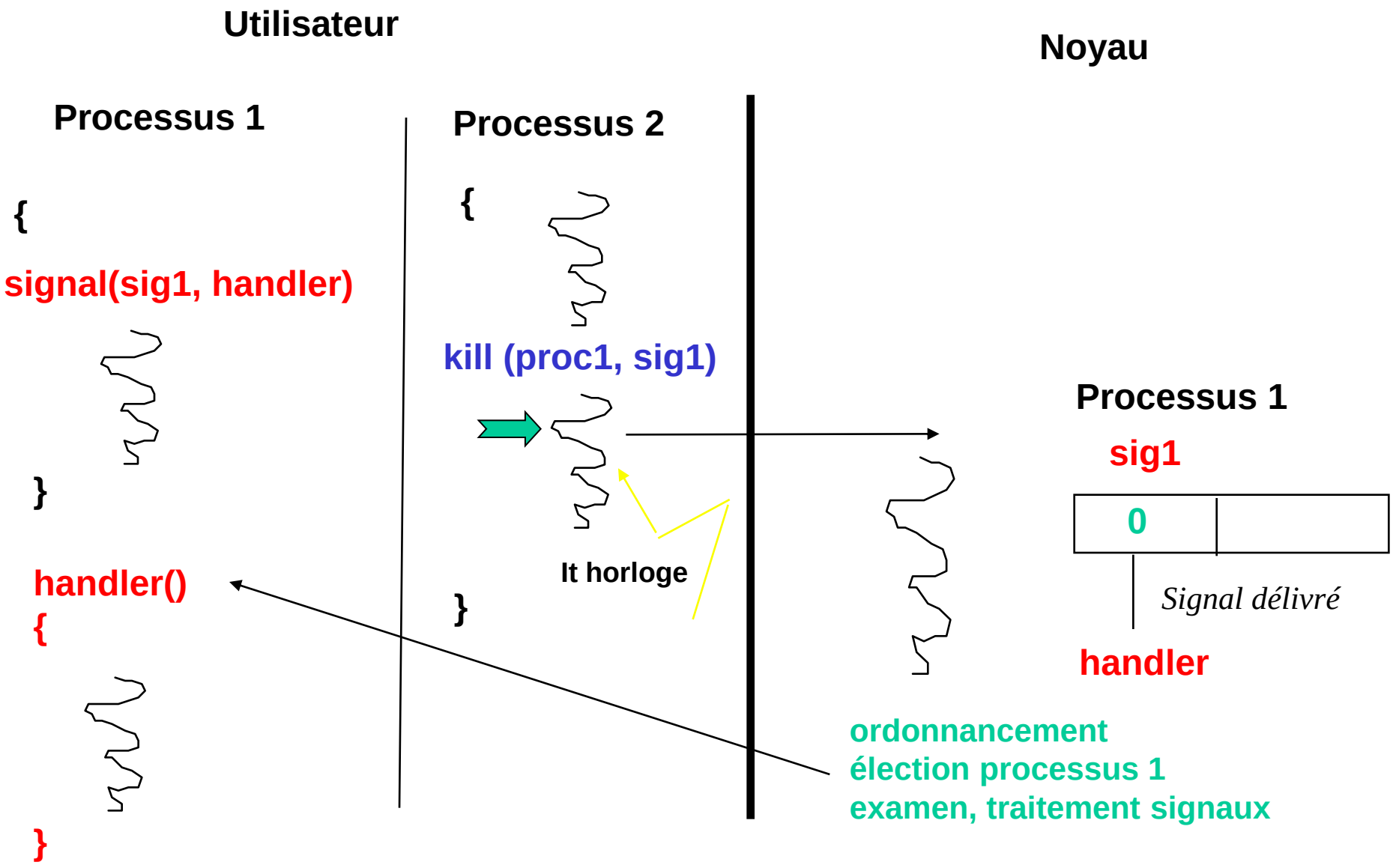
Signaux : synthèse



Signaux : synthèse



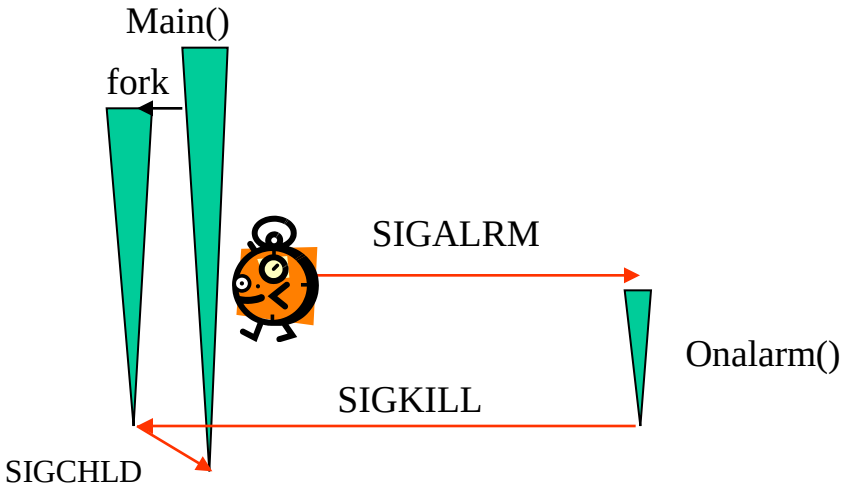
Signaux : synthèse



Signaux et interruptions

- Signaux
  - Le processus P2 envoie un signal au processus P1 (signal pendant chez P1)
  - Plus tard, le processus P1 est élu. Il quitte le mode noyau. Il exécute le handler du signal en mode utilisateur (signal délivré à P1).

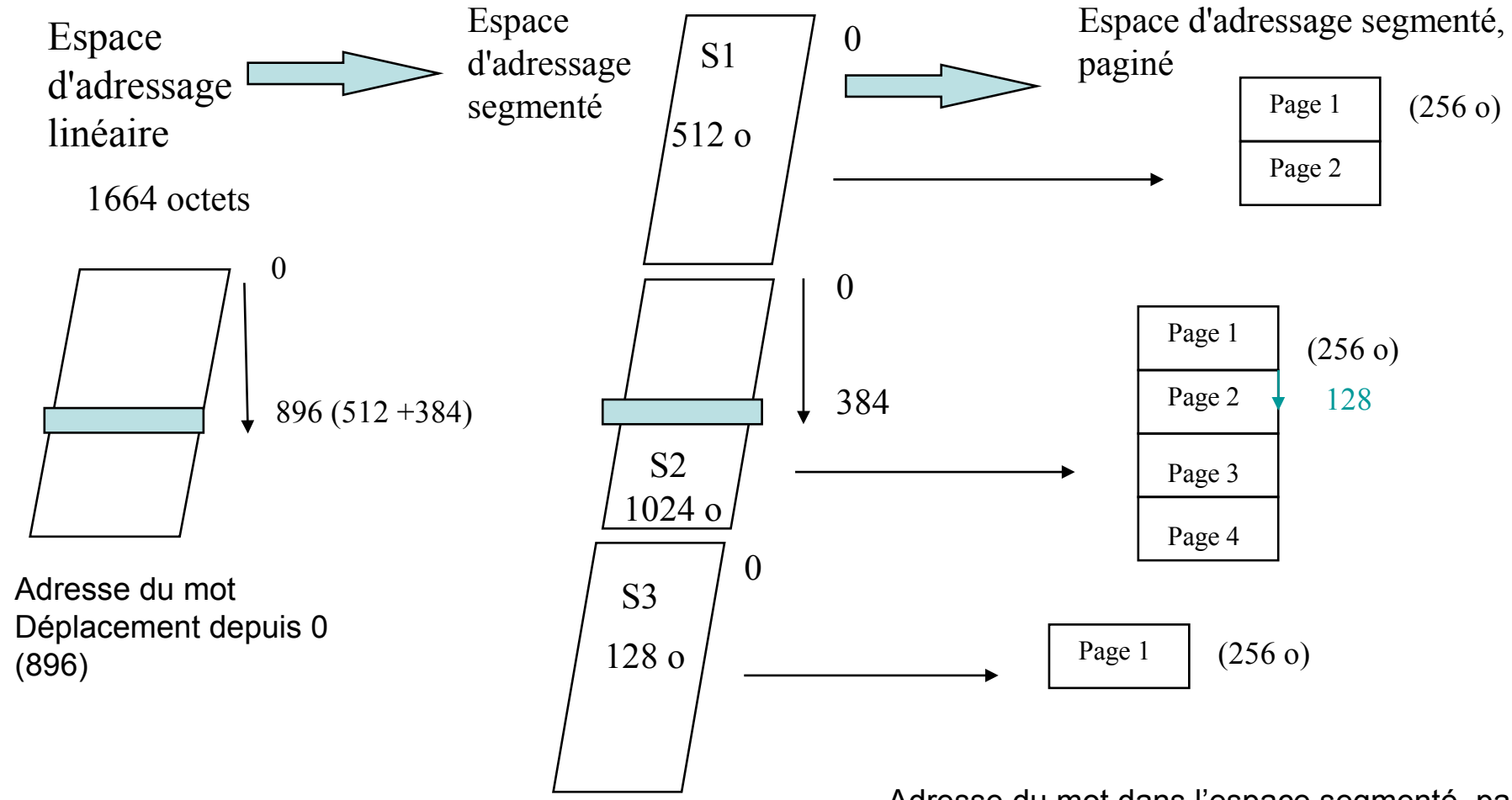
- Interruptions
  - Le dispositif matériel X envoie une interruption lors de l'exécution du processus P1.
  - Immédiatement, le processus P1 est dérouté en mode noyau pour exécuter le handler « routine » de l'interruption.



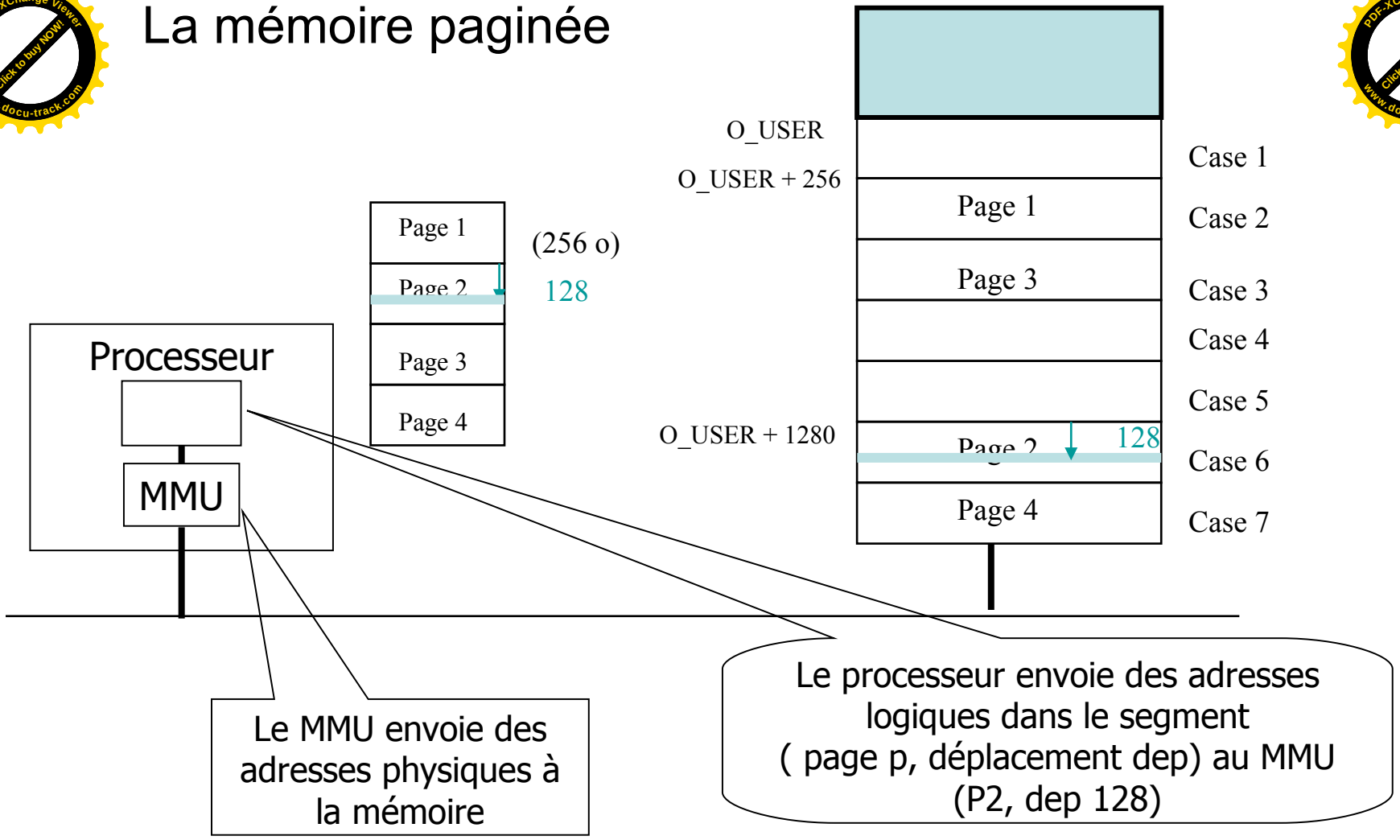
PDF-XChange Viewer
Click to buy NOW!
www.docu-track.com

4. Gestion de la mémoire centrale

Espace d'adressage segmenté et paginé
adresse logique segmentée paginée
(n°segment S, n°page du segment, déplacement dep dans la page)



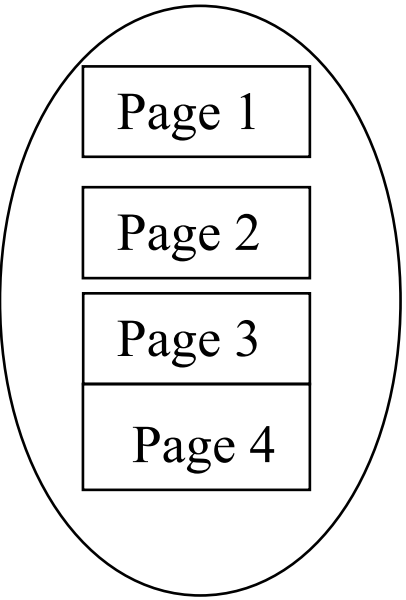
La mémoire paginée



☞ Il faut convertir l'adresse paginée en son équivalent adresse physique
Adresse physique = **adresse implantation case contenant la page adr (O_User + 1280)** + deplacement dep (128)

➢ Table des pages

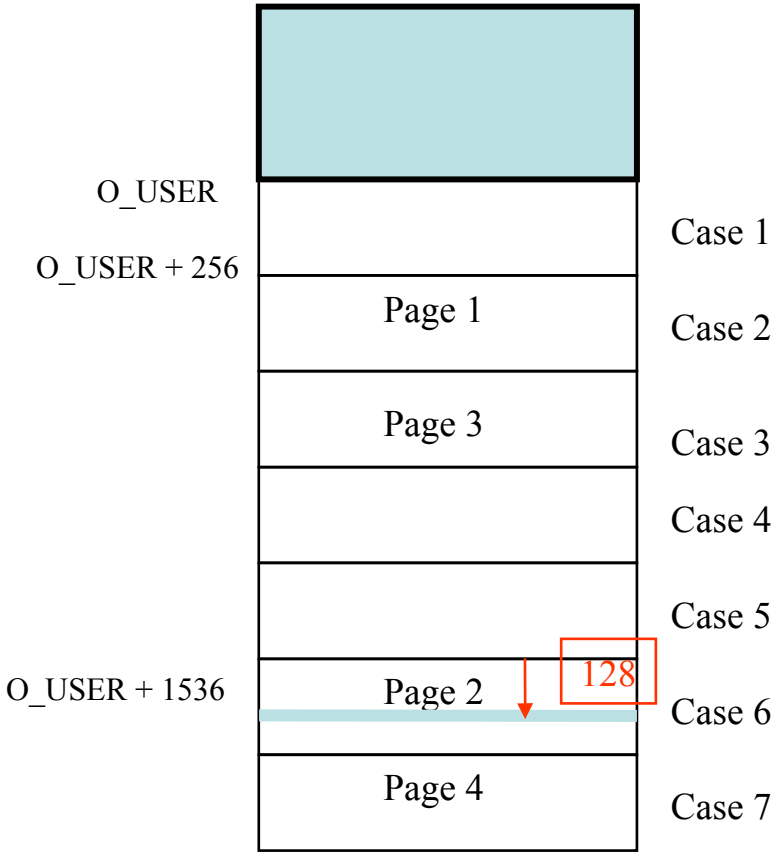
La mémoire paginée



Espace d'adressage du segment S2

Table des pages du segment

Numéro page	adresse case
1	Adr C2 (O_User + 256)
2	Adr C6 (O_User + 1280)
3	Adr C3 (O_User + 512)
4	Adr C7 (O_User + 1536)



La mémoire segmentée paginée

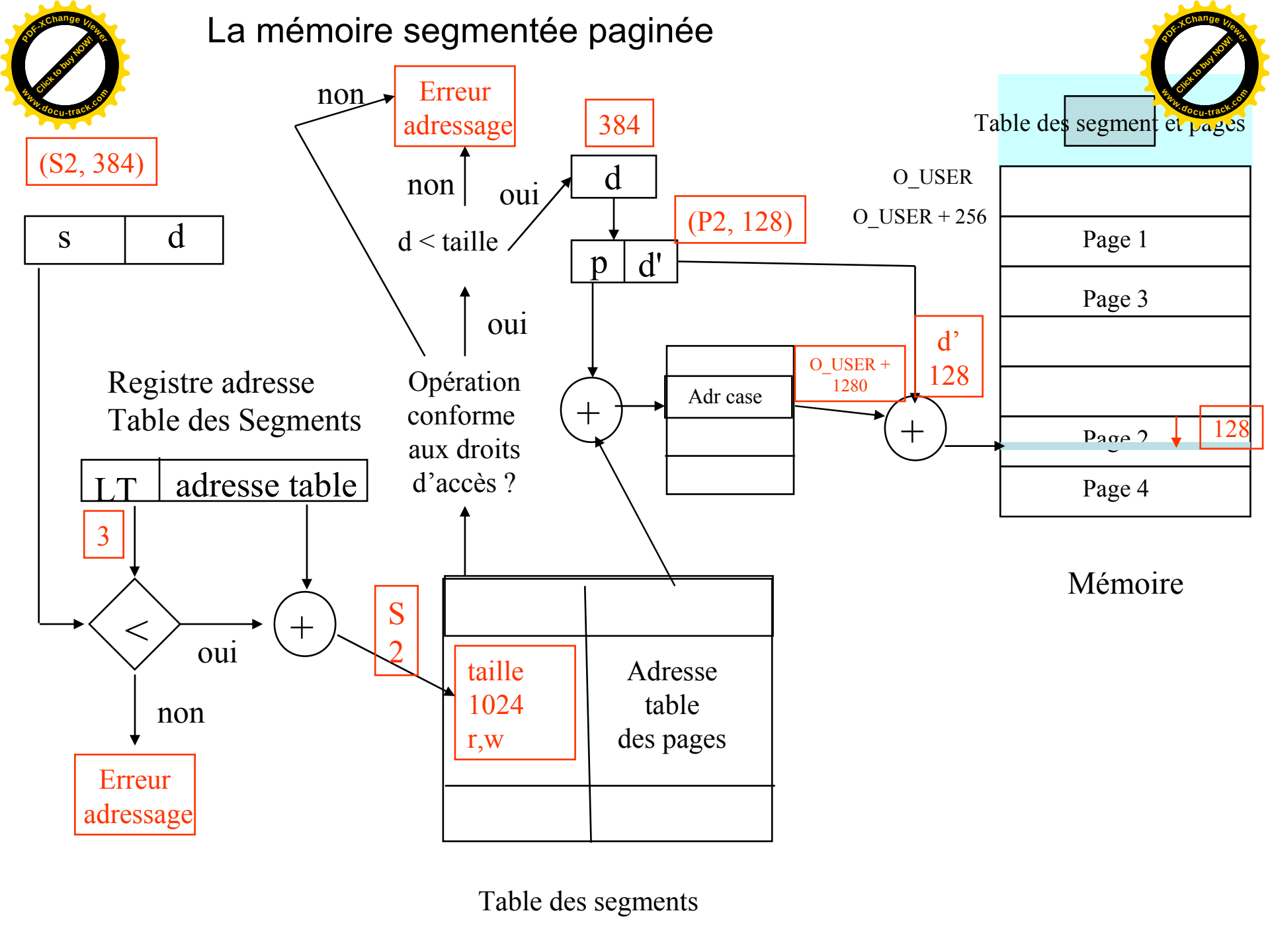


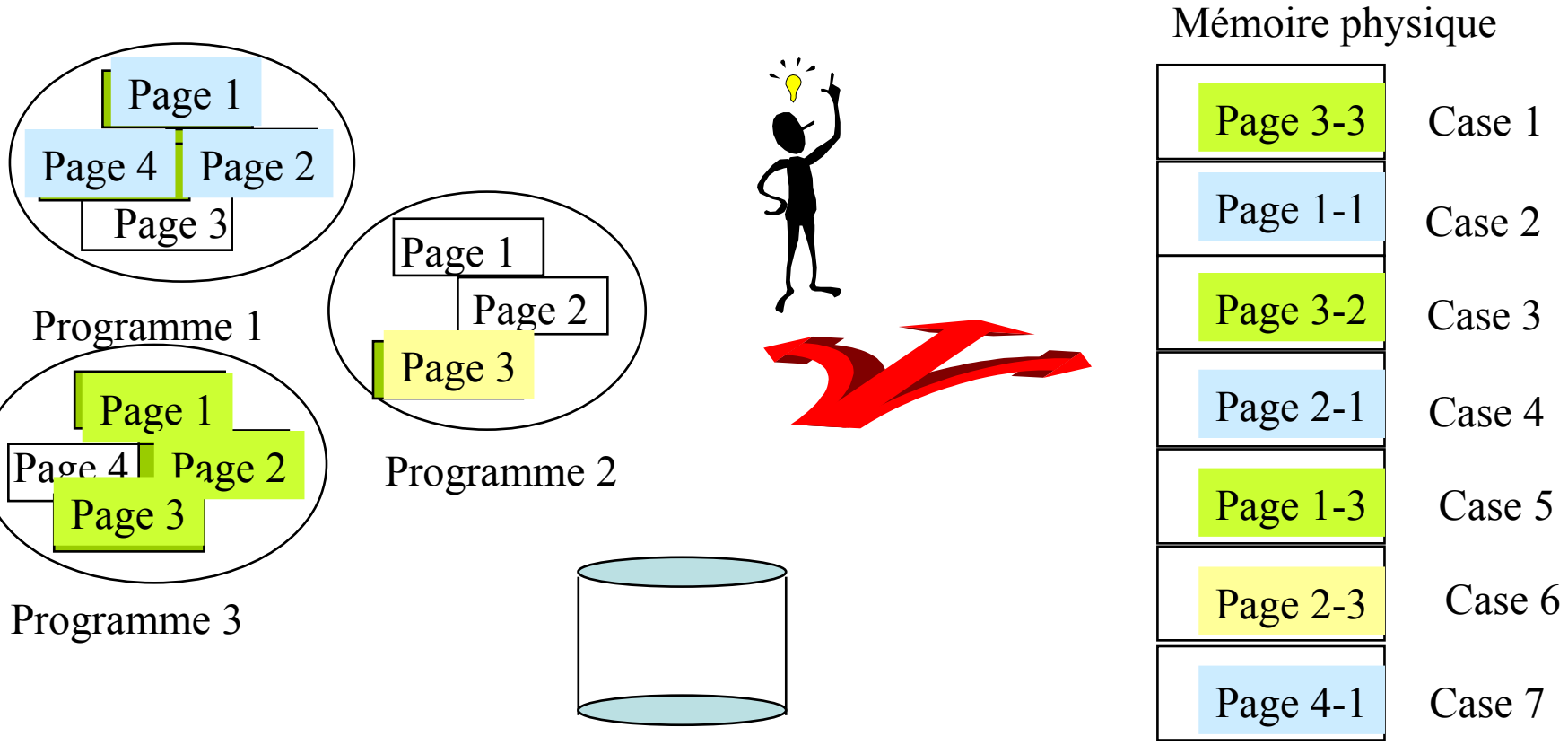
Table des segments

Mémoire virtuelle

PDF-XChange Viewer Click to buy NOW! www.docu-track.com

- La capacité de la mémoire centrale est trop petite pour charger l'ensemble des pages des programmes utilisateurs.

  ☞ Ne charger que les pages utiles à un instant (principes de localité).



Bit de validation

V	2
V	4
I	-
V	7

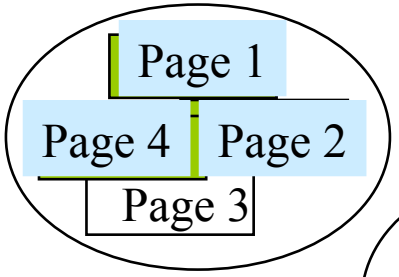
Processus 1

I	-
I	-
V	3

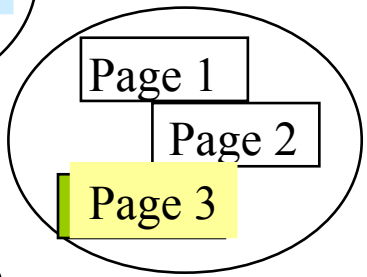
Processus 2

V	5
V	6
V	1
I	-

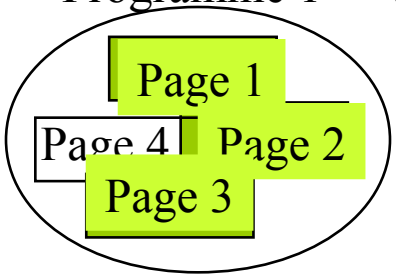
Processus 3



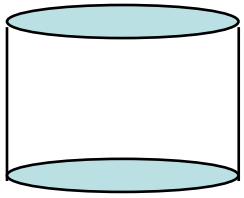
Programme 1



Programme 2



Programme 3



Mémoire physique

Page 3-3	Case 1
Page 1-1	Case 2
Page 3-2	Case 3
Page 2-1	Case 4
Page 1-3	Case 5
Page 2-3	Case 6
Page 4-1	Case 7

Bit de validation et défaut de page

V	2
V	4
I	-
V	7

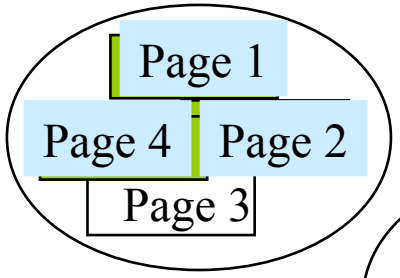
Processus 1

I	-
I	-
V	3

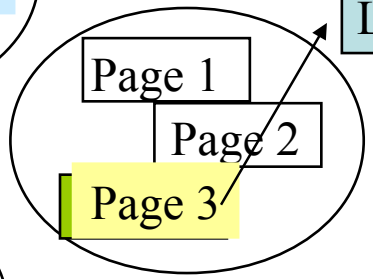
Processus 2

V	5
V	6
V	1
I	-

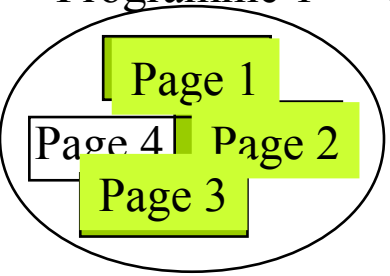
Processus 3



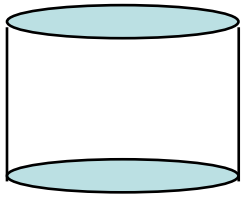
Programme 1



Programme 2



Programme 3



Load D R (P2, d)

Processus 2 : accès à la page 2

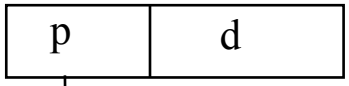
DEFAUT DE PAGE

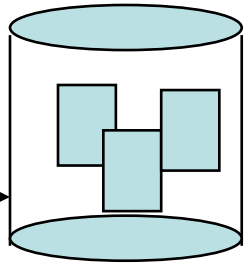
Mémoire physique

Page 3-3	Case 1
Page 1-1	Case 2
Page 3-2	Case 3
Page 2-1	Case 4
Page 1-3	Case 5
Page 2-3	Case 6
Page 4-1	Case 7

Défaut de page

Adresse logique





1. Déroutement
E/S disque

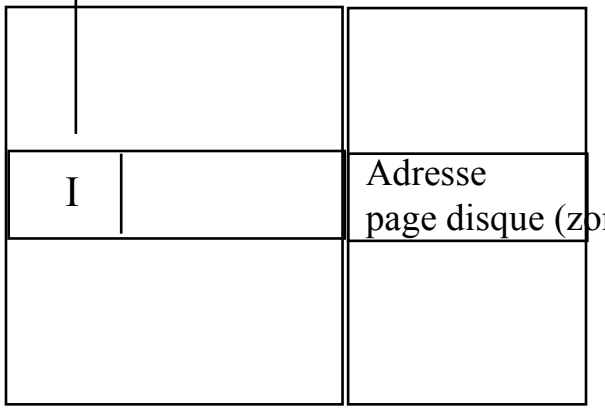
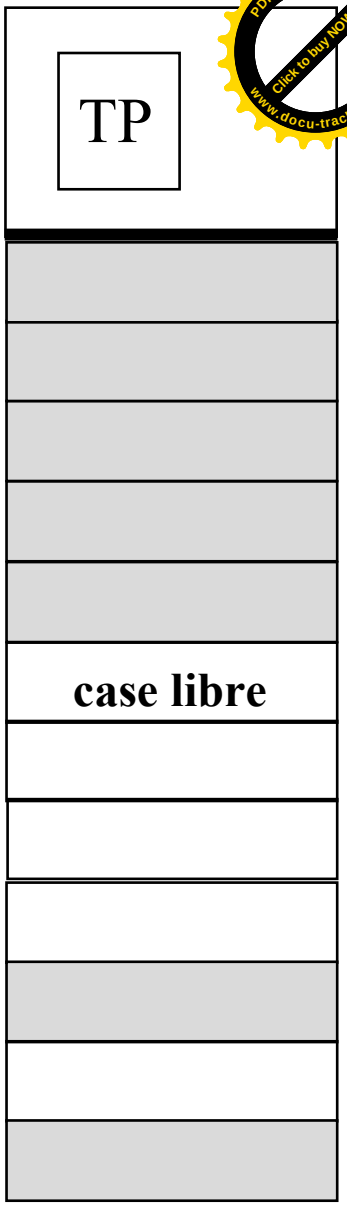


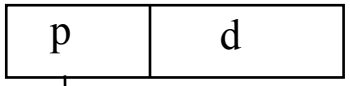
Table des pages Table du disque

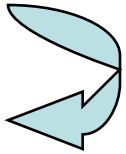


occupée

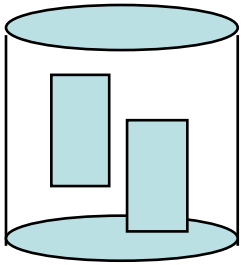
Défaut de page

Adresse logique

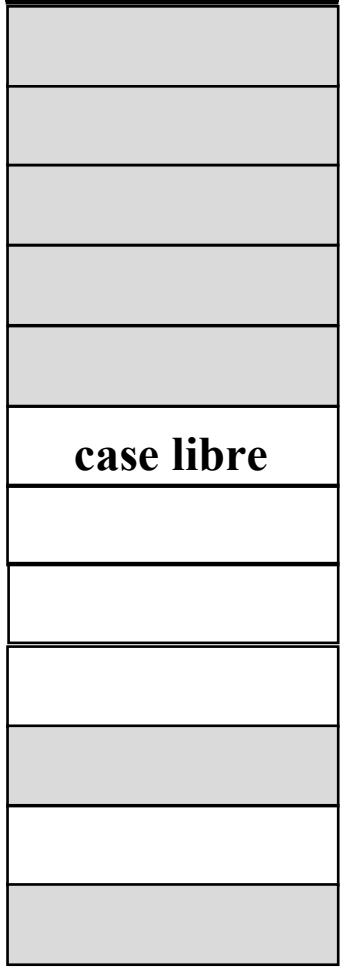




4. Reprise instruction



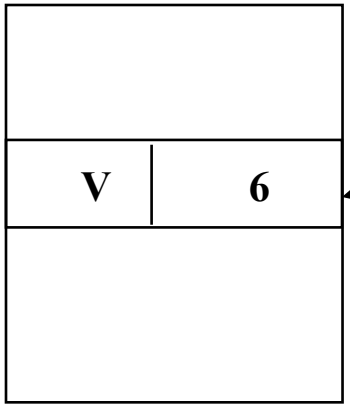
TP



6

2 Chargement de la page

p



3. Mise à jour table des pages

Table des pages

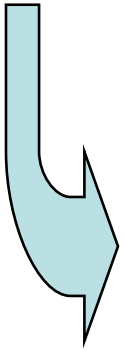
occupée

Chargement de page

PDF-XChange Viewer Click to buy NOW! www.docu-track.com

- Lors d'un défaut de page, la page manquante est chargée dans une case libre

☞ la totalité des cases de la mémoire centrale peuvent être occupées



• le système d'exploitation utilise un algorithme pour choisir une case à libérer

- FIFO (First In, First out) : la page la plus anciennement chargée est libérer
- LRU (Least Recently Used) : la page la moins récemment accédée est libérée

On considère trois processus PA, PB et PC qui disposent d'un espace d'adressage segmenté et paginé.
Le processus PA dispose d'un espace d'adressage composé de 2 segments S1A et S2A, comportant respectivement 4 pages et 2 pages.
Le processus PB dispose d'un espace d'adressage composé de 3 segments S1B, S2B et S3B, comportant respectivement 3 pages, 2 pages et 2 pages.
Le processus PC dispose d'un espace d'adressage composé de 1 segment S1C, comportant respectivement 5 pages.
La mémoire centrale est composée de 20 cases numérotées de 1 à 20. Chaque case a une capacité de 1024 octets. Lors d'un défaut de pages, la page manquante est chargée **dans la case libre de plus grand numéro.**
A l'instant t, l'allocation des espaces d'adressage est la suivante :
Pour le processus PA, seules les pages P1, P3 et P4 du segment S1A sont chargées en mémoire centrale respectivement dans les cases 10, 12 et 7 ; seule la page P2 du segment S2A est chargée en mémoire centrale dans la case 5 ;
Pour le processus PB, seules les pages P1 et P2 du segment S1B sont chargées en mémoire centrale respectivement dans les cases 6 et 4 ; seule la page P1 du segment S2B est chargée en mémoire centrale dans la case 20 ; aucune page du segment S3B n'est en mémoire centrale.
Pour le processus PC, seules les pages P1, P2 et P5 sont chargées en mémoire centrale respectivement dans les cases 13, 14 et 8.

Question 1

Représentez sur un schéma les structures de données (tables des segments, tables des pages et mémoire centrale) correspondant à l'allocation décrite.

Question 2

Le processus PA accède à l'adresse linéaire 1 804 dans son espace d'adressage. Donnez l'adresse logique (virtuelle) puis l'adresse physique correspondante.
Le processus PB accède à l'adresse linéaire 5 512 dans son espace d'adressage Donnez l'adresse logique (virtuelle) puis l'adresse physique correspondante.

FIFO : la plus anciennement chargée

référence	0	1	4	2	0	1	3	0	4
défauts	*	*	*	*	*	*	*		*
	0	0	0	**2**	2	2	**3**	3	3
		1	1	1	**0**	0	0	0	**4**
			4	4	4	**1**	1	1	1

LRU : la moins récemment accédée

référence	0	1	4	2	0	1	3	0	4
défauts	*	*	*	*	*	*	*		*
	0	0	0	**2**	2	2	**3**	3	3
		1	1	1	**0**	0	0	0	0
			4	4	4	**1**	1	1	**4**

PDF-XChange Viewer
Click to buy NOW!
www.docu-track.com

6. Système de gestion de fichiers

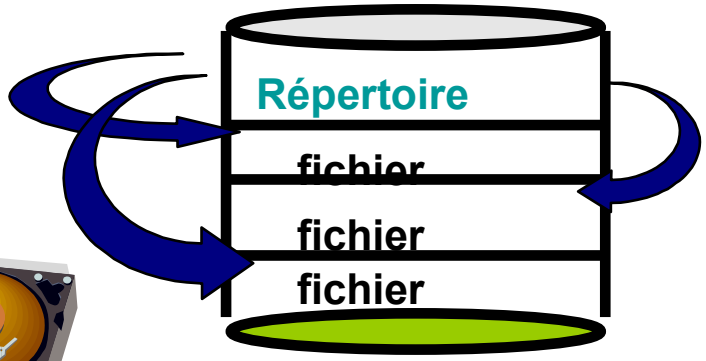
(SGF Linux)

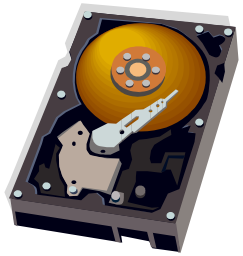
Système de gestion de fichiers

Données

enregistrement

Fichier logique organisé selon un mode





Fichier physique
Descripteur +
Ensemble de blocs
alloués
selon une méthode
données

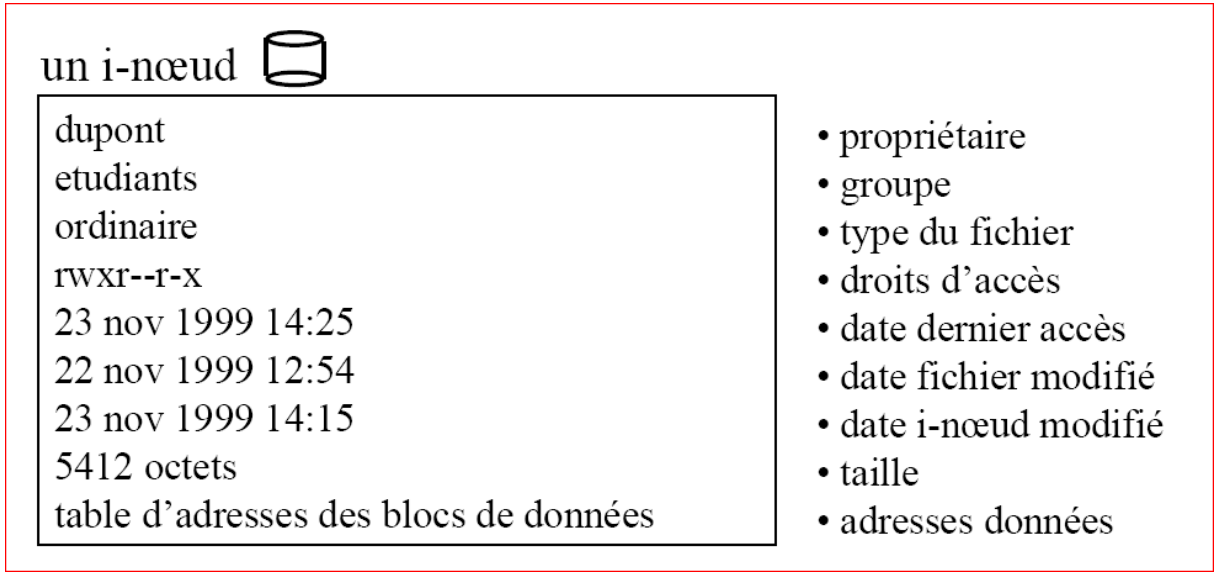
Fichier Linux

- Identifié par un nom, sans structure logique (suite d'octets)
- La méthode d 'allocation mise en œuvre est de type allocation indexée.
- Un fichier Linux est composé d'un descripteur appelé « inode » et de blocs physiques, qui sont soit des blocs d'index, sont des blocs de données. Les blocs de données sont alloués au fur et à mesure de l'extension du fichier.
- Un bloc est identifié par un numéro codé sur 4 octets. La taille d'un bloc est un multiple de la taille d'un secteur (512 octets)

Fichier Linux : inode

- Structure stockée sur le disque, allouée à la création du fichier et répérée par un numéro

- Contient les attributs du fichier :
  - Nom
  - Type : fichiers normaux, répertoires, *périphériques, tubes nommés , sockets*
  - Droits d'accès
  - Heures diverses
  - Taille du fichier en octets
  - Table des adresses des blocs de données



Fichier Linux : structure

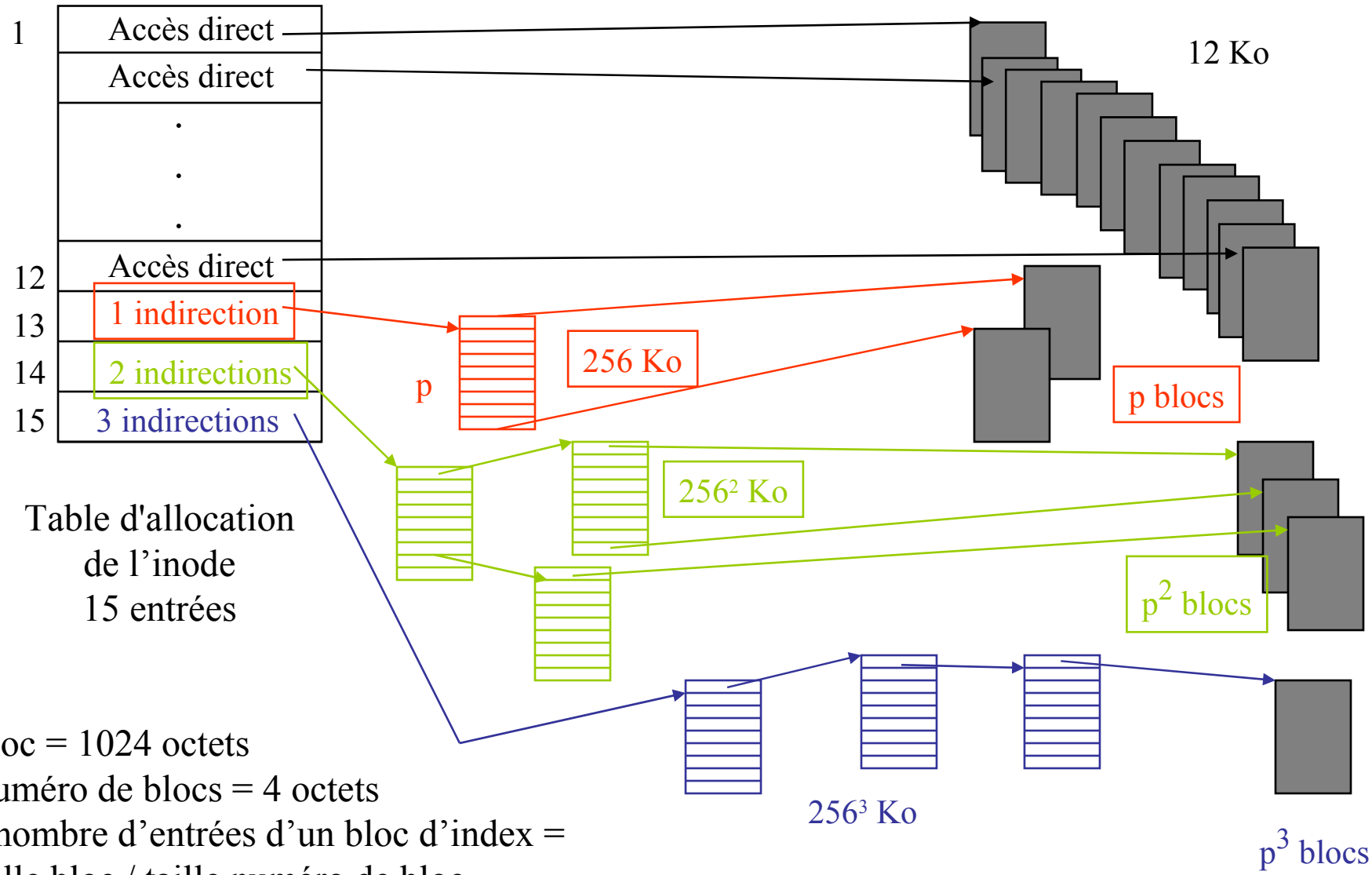
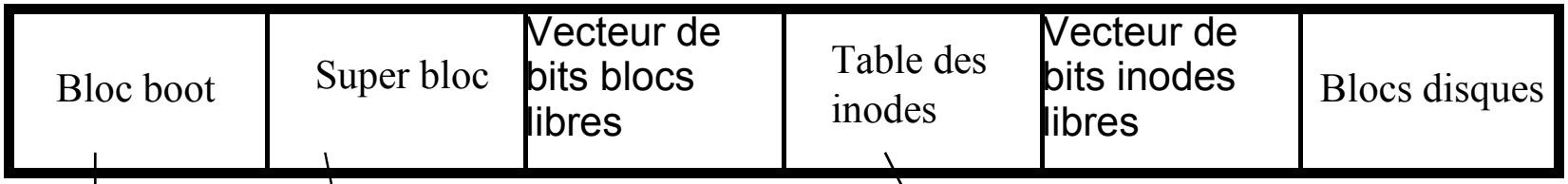


Table d'allocation
de l'inode
15 entrées

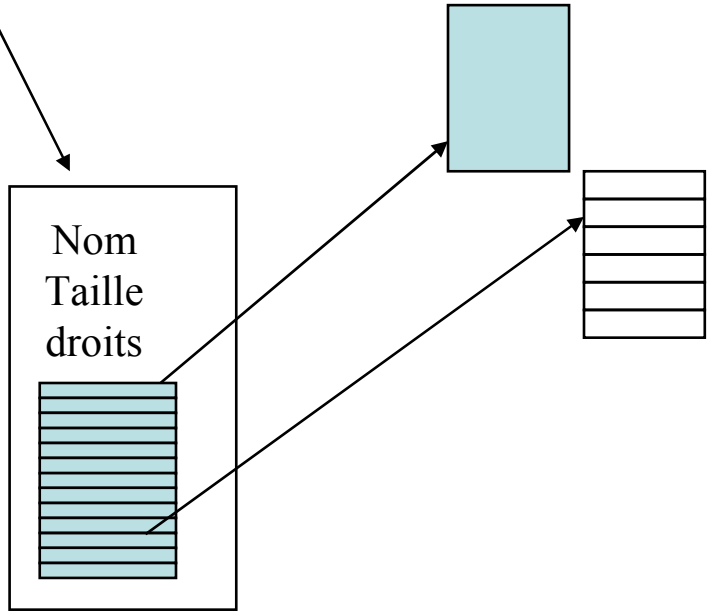
Bloc = 1024 octets
Numéro de blocs = 4 octets
p nombre d'entrées d'un bloc d'index =
taille bloc / taille numéro de bloc

Partition Linux : structure



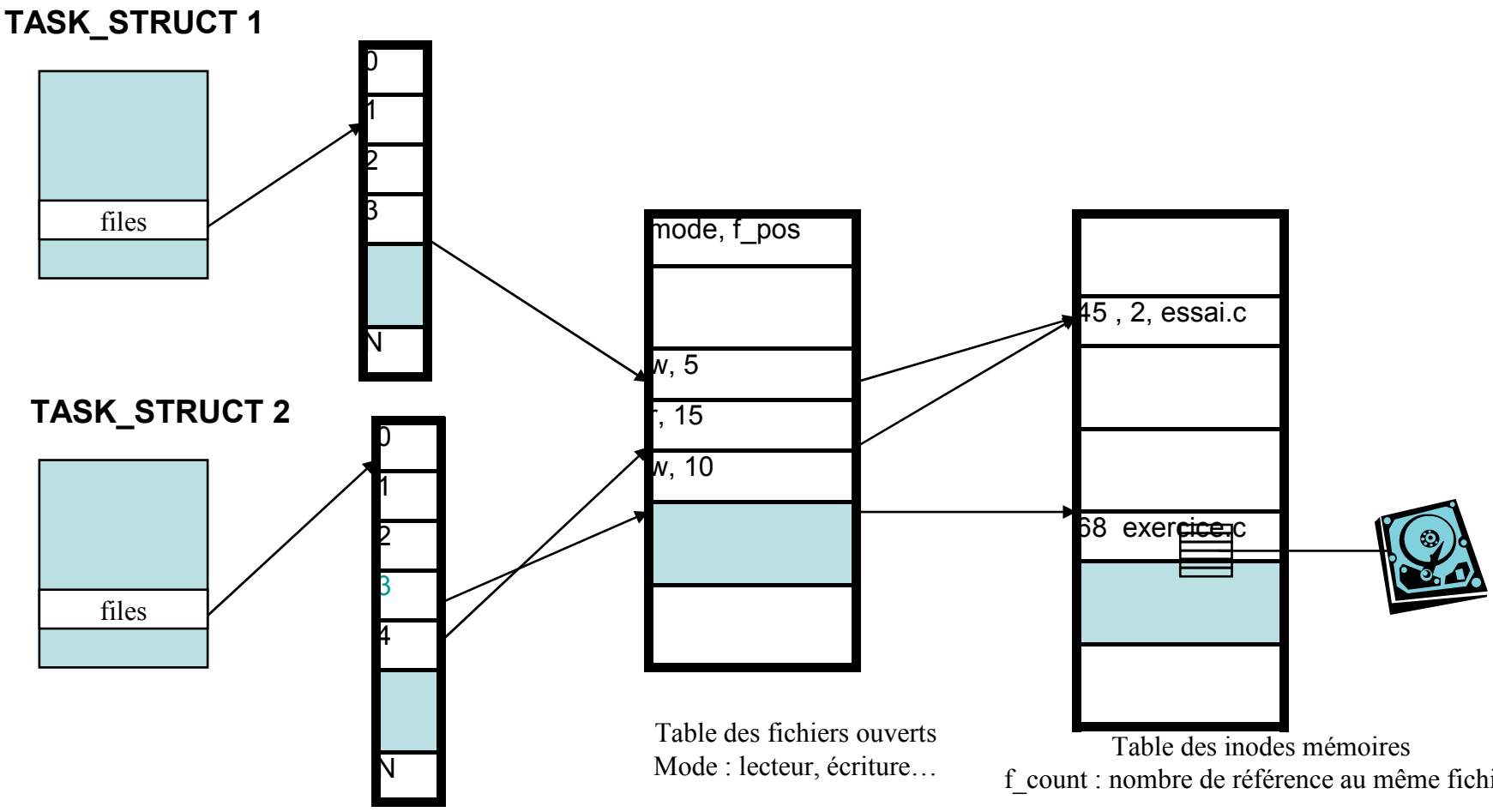
1er secteur
contient l'amorçage du SE
et description de la partition

Nom de la partition
Taille de la partition
Taille d'un bloc
Nombre d'inodes
Nombre d'inodes libres
Nombre de blocs libres



inode

Structure de gestion des fichiers dans un processus



f_pos : pointeur de fichier ; octet courant

fp = open (« /home/delacroix/exercices.c » , « w »)
fp = 3

17	.	34	..	68	exercice .c

Lecture d'un fichier : le buffer cache
while(read (fd, &un_eleve, sizeof(un_eleve)) > 0);

- Le système maintient une liste de tampons mémoire qui joue le rôle de cache pour les blocs du disque et permet de réduire les entrées/sorties.

- La taille d'un tampon est égale à la taille d'un bloc disque. Il est identifié par un numéro de bloc physique et numéro de périphérique.

- Lorsque le système doit lire un bloc depuis le disque :

  – Il cherche d'abord si le bloc est déjà présent dans la liste des tampons mémoire
  – Si non, il prend un tampon libre et copie le bloc disque dans le tampon.
  – Si tous les tampons sont occupés, il libère un tampon en choisissant le moins récemment accédé.

PDF-XChange Viewer Click to buy NOW! www.docu-track.com

On considère un système Linux/Unix. Chaque utilisateur dispose d'un compte et d'un répertoire de travail qui constitue le répertoire dans lequel il peut stocker ses fichiers.
L'utilisateur delacroi qui appartient au groupe des enseignants en informatique ensinf exécute la commande ls –l qui permet d'afficher l'ensemble des fichiers de son répertoire.

```
lmi20: # ls -l
-rwxrw-r--   1 delacroi   ensinf  71680 Mar 25 19:28 exemple3
-rw-rw-r--   1 delacroi   ensinf 591 Mar 25 19:24 exemple3.txt
-rw-r--r--   1 delacroi   ensinf  590 Mar 25 19:24 exemple3.txt~
drwxr-xr-x   2 delacroi   ensinf  4096 Apr  5 19:27 exercices
```

A- Expliciter quels sont les droits associés au fichier exemple3 pour les différents utilisateurs de la machine.

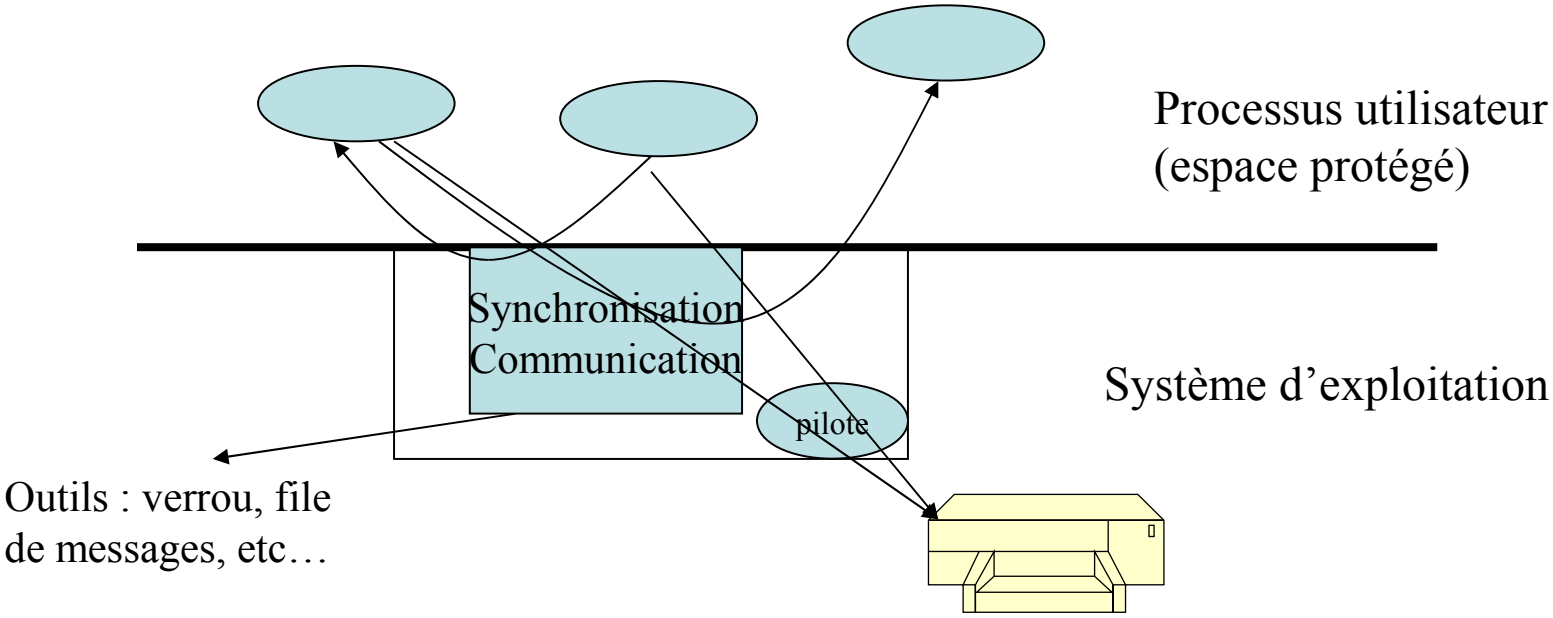
B- L'utilisateur delacroi exécute la commande suivante : lmi20: # chmod a+w exemple3
Que se passe-t-il ? Quels sont les droits associés au fichier exemple3.

C- Le fichier exemple3 a une taille de 72704 octets. Les blocs de données du système de gestion de fichiers ont une taille de 512 octets. Un numéro de bloc occupe 4 octets.

- Combien de blocs de données comporte le fichier ? (142)
- Quel taille d'index ? Combien de blocs d'index sont nécessaires ? (128 – 3)
- Combien d'accès disque sont nécessaires pour lire de façon séquentielle ce fichier, sachant que le système de gestion de fichiers maintient un cache des blocs disque les plus récemment lus ? 12 + 1 + 128 + 2 + 2

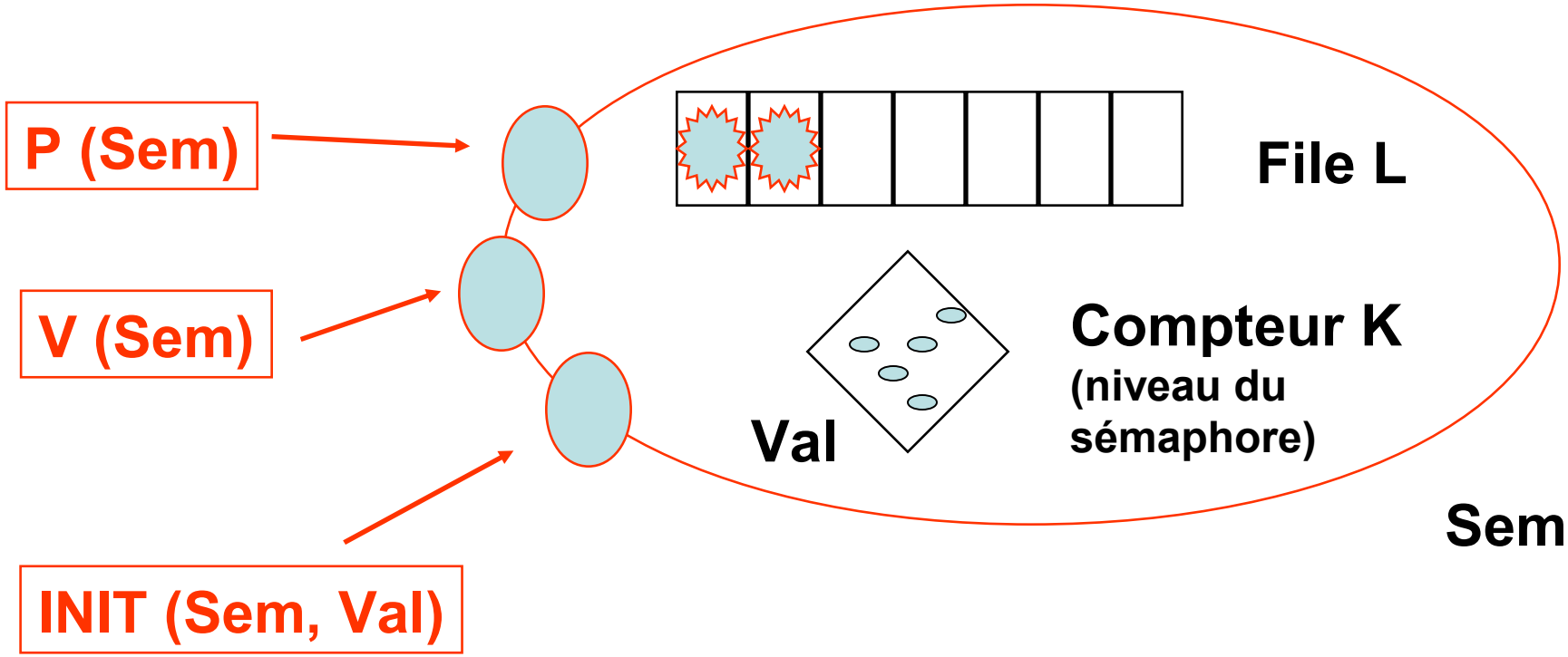
Synchronisation entre processus

Principes : Exclusion mutuelle et interblocage
Communication entre processus



Les sémaphores

- Structure

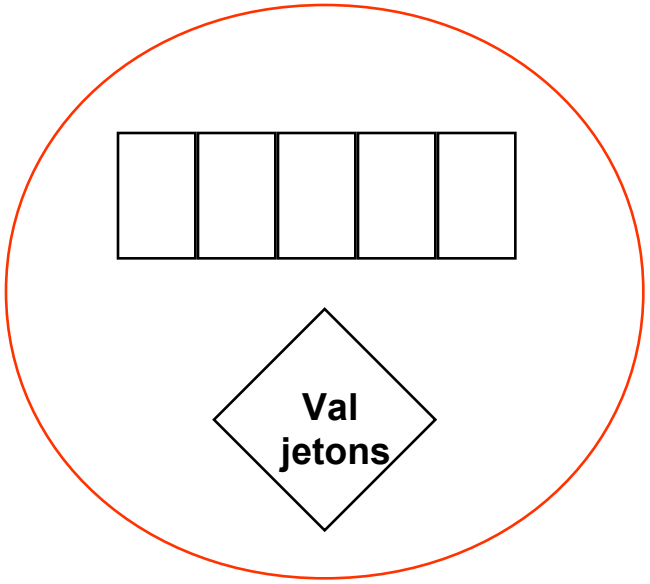


Opérations indivisibles

Les sémaphores

- Opération Init (Sem, Val)

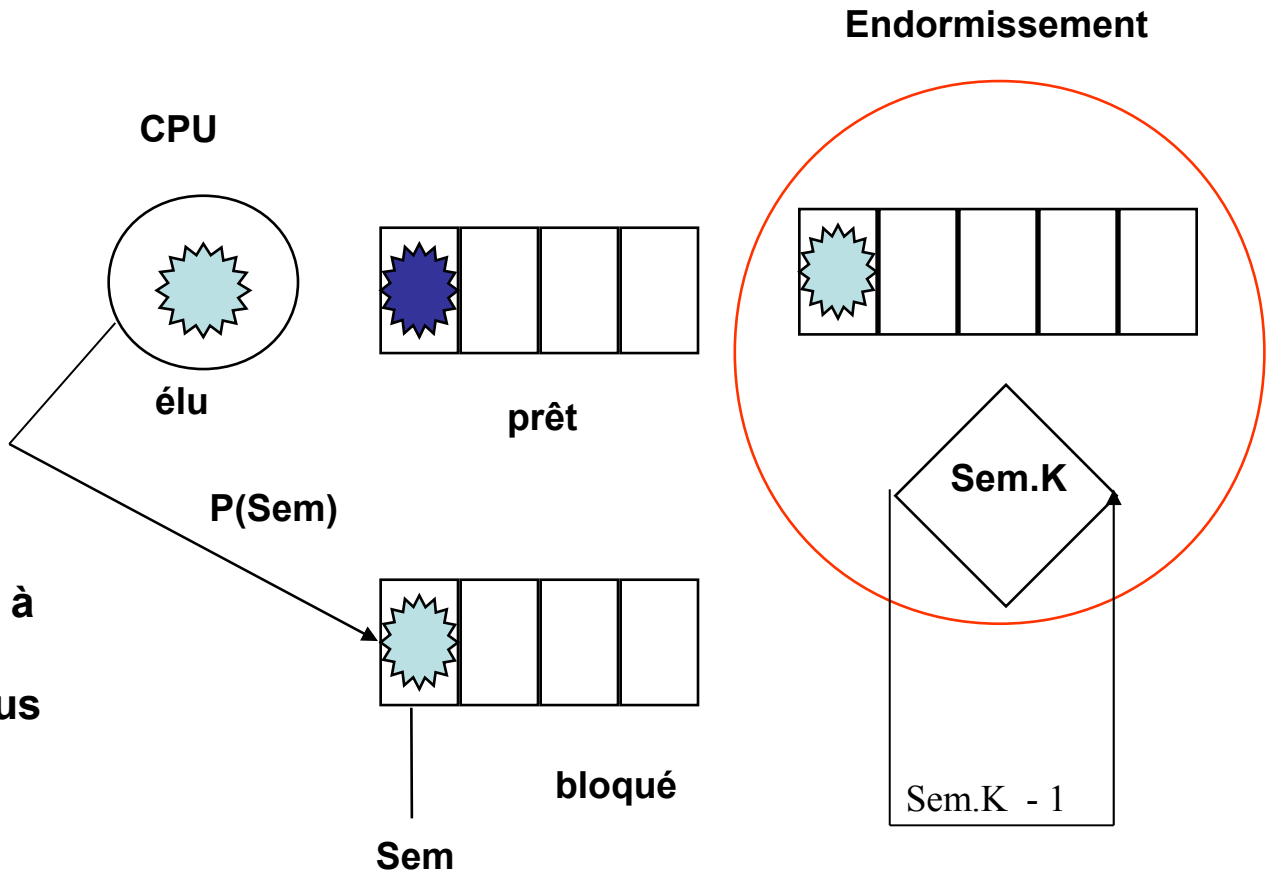
```
Init (Sem, Val)

début
        masquer_it
        Sem. K := Val jetons;
        Sem. L := ∅
        demasquer_it
fin
```



Les sémaphores

- Opération P (Sem)

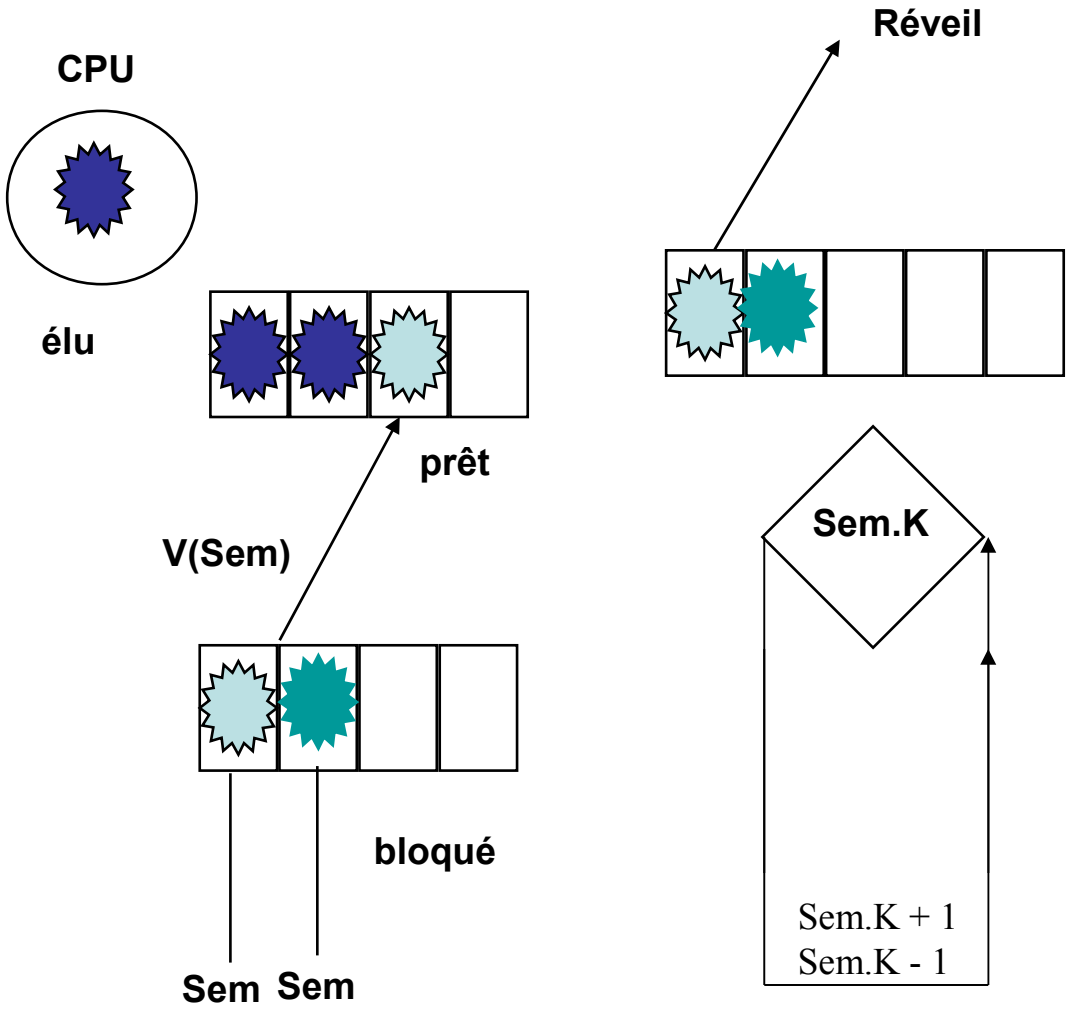
```
P  (Sem)

Début
Masquer_IT
Si Sem.K > 0
alors
Sem.K := Sem.K - 1;
sinon
        ajouter ce processus à
        Sem.L
        endormir ce processus
fsi
Démasquer_IT
fin
```



Les sémaphores

- Opération V (Sem)

```
V (Sem)

début
Sem.K := Sem.K + 1;
Si il y a des processus endormis
sur Sem
alors
        sortir un processus  de
        Sem.L
        Sem.K := Sem.K - 1;
        réveiller ce processus
fsi
fin
```



Section critique avec sémaphore

PDF-XChange Viewer Click to buy NOW! www.docu-track.com

1 seul processus en section critique → Sémaphore Mutex initialisé à 1

P (Mutex) **Entrée section_critique**

Section Critique
Utilisation ressource critique
à un seul point d'accès

V (Mutex) **Sortie section_critique**

Mutex

K = 1

Réservation :

P(Mutex)

Si nb_place > 0

alors

Réserver une place

nb_place = nb_place - 1

fsi

V(Mutex)

Client 1

Demande Réservation

P(Mutex) *K > 0 → P passant*

Nb_Place > 0 = 1

Nb_Place = Nb_Place - 1

V(Mutex) *→ V réveille Client2*

Client 2

Demande Réservation

P(Mutex) *→ K = 0 P bloquant*

Nb_Place non accessible

Nb_Place = 0

Interblocage

PDF-XChange Viewer Click to buy NOW! www.docu-track.com

- **Interblocage**

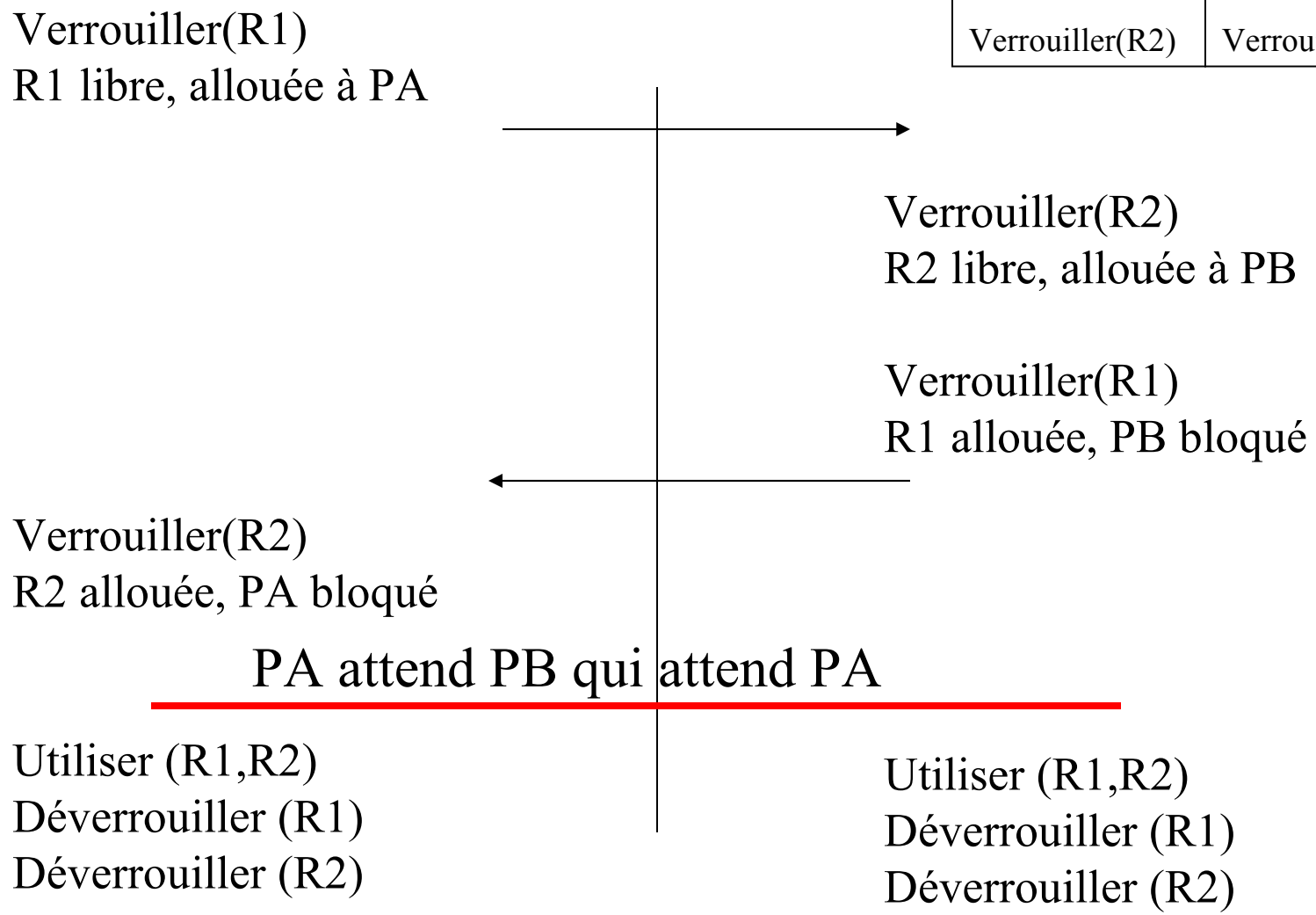
Ensemble de n processus attendant chacun une ressource déjà possédée que par un autre processus de l'ensemble

R1 et R2 à un seul point d'accès
R1, R2 libre

Processus PA	Processus PB
Verrouiller(R1)	Verrouiller(R2)
Verrouiller(R2)	Verrouiller(R1)
Utiliser (R1,R2)	Utiliser (R1,R2)
Déverrouiller (R1)	Déverrouiller (R1)
Déverrouiller (R2)	Déverrouiller (R2)

Interblocage

Processus PA	Processus PB
Verrouiller(R1)	Verrouiller(R2)
Verrouiller(R2)	Verrouiller(R1)



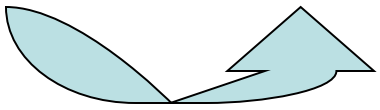
☞ **Aucun processus ne peut poursuivre son exécution**

Politiques de prévention

- **Imposer un ordre total sur l'allocation des ressources : tout processus doit demander l'accès aux ressources selon un ordre préétabli :**

- **Par exemple : R1 avant R2**

Verrouiller(R1)	Verrouiller(R2)
Verrouiller(R2)	Verrouiller(R1)
Utiliser (R1,R2)	Utiliser (R1,R2)
Déverrouiller (R1)	Déverrouiller (R1)
Déverrouiller (R2)	Déverrouiller (R2)



Verrouiller(R1)
Verrouiller(R2)

Utiliser (R1,R2)

Déverrouiller (R1)
Déverrouiller (R2)

Exercice 1

On considère quatre processus P1, P2, P3 et P4 dont les caractéristiques sont les suivantes :

	Temps d'exécution	priorité (plus petite valeur = plus grande priorité)
P1	8 unités	3
P2	6 unités	4
P3	12 unités	1
P4	4 unités	2

Question 1

Les quatre processus sont présents à l'instant t = 0 dans la file des processus prêts dans l'ordre donné par la liste (P1 est la tête de liste). L'ordonnancement est en FIFO. Donnez l'ordre d'exécution des processus et le temps de réponse de chaque processus.

Question 2

Les quatre processus sont présents à l'instant t = 0 dans la file des processus prêts dans l'ordre donné par les priorités. L'ordonnancement est par priorité. Donnez l'ordre d'exécution des processus et le temps de réponse de chaque processus.

Question 3

Les quatre processus sont présents à l'instant t = 0 dans la file des processus prêts dans l'ordre donné par la liste (P1 est la tête de liste). L'ordonnancement est en tourniquet avec un quantum Q égal à 4 unités. Donnez l'ordre d'exécution des processus et le temps de réponse de chaque processus.

Exercice 2

On considère quatre processus P1, P2, P3, P4 dont les caractéristiques sont les suivantes :

	Date d'arrivée	Temps d'exécution	priorité (plus petite valeur = plus grande priorité)
P1	0	9 unités	2
P2	3	5 unités	1
P3	4	7 unités	3
P4	10	4 unités	1

Question 1

Représentez l'exécution des processus en supposant un ordonnancement par priorités **non** préemptives. Donnez le temps de réponse de chaque processus.

Question 2

Représentez l'exécution des processus en supposant un ordonnancement par priorités préemptives. Donnez le temps de réponse de chaque processus.