NFP 136 - Système d'exploitation

Amélie Lambert

2015-2016

Plan du cours

1. Système d'exploitation : rôle et services
2. Les appels système et les commandes
3. La chaîne de production de programme
4. Les processus
5. Ordonnancement du processeur

Bibliographie et sources :

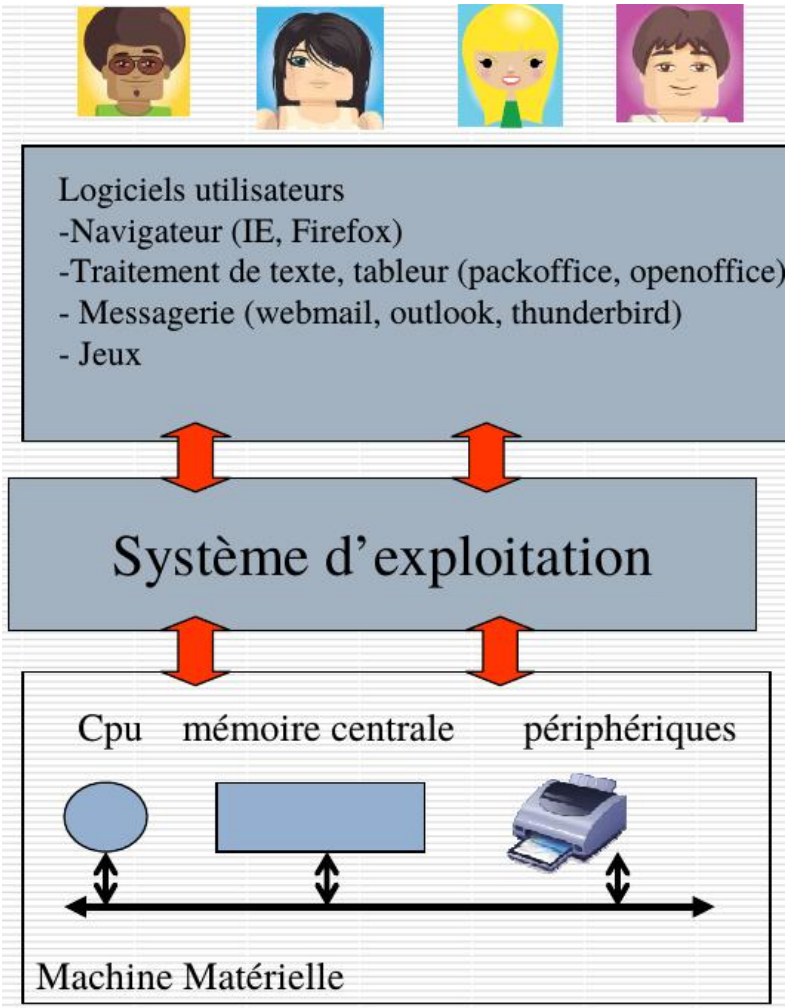
- DELACROIX J. Cours CNAM NFA003, NSY103
- CAZES A. DELACROIX J. Architecture des machines et des Systèmes informatiques Dunod 2003

1. Système d'exploitation : rôle et services

Les différents niveaux de la machine informatique

Une machine informatique est composée de trois couches :

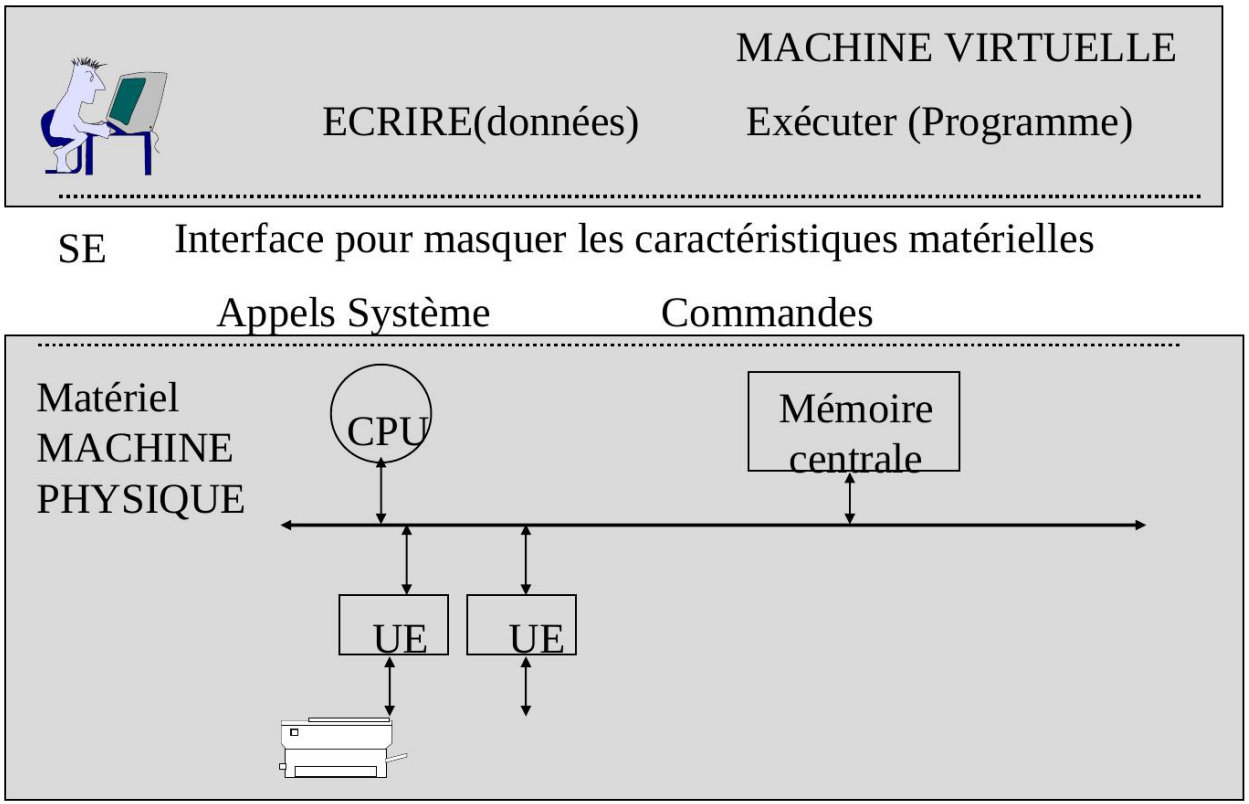
- Le matériel **hardware** (la machine physique : processeur, mémoire centrale, périphériques, l'ensemble communiquant par un bus)
- Le **logiciel de système d'exploitation** (ensemble de programmes qui se place à l'interface entre le matériel et les logiciels applicatifs)
- Les logiciels des utilisateurs **software** (programmes qui permettent à l'utilisateur de réaliser des taches sur la machine).



Qu'est ce qu'un système d'exploitation

Le système d'exploitation est un ensemble de programmes qui réalisent l'interface entre la machine physique et les utilisateurs.

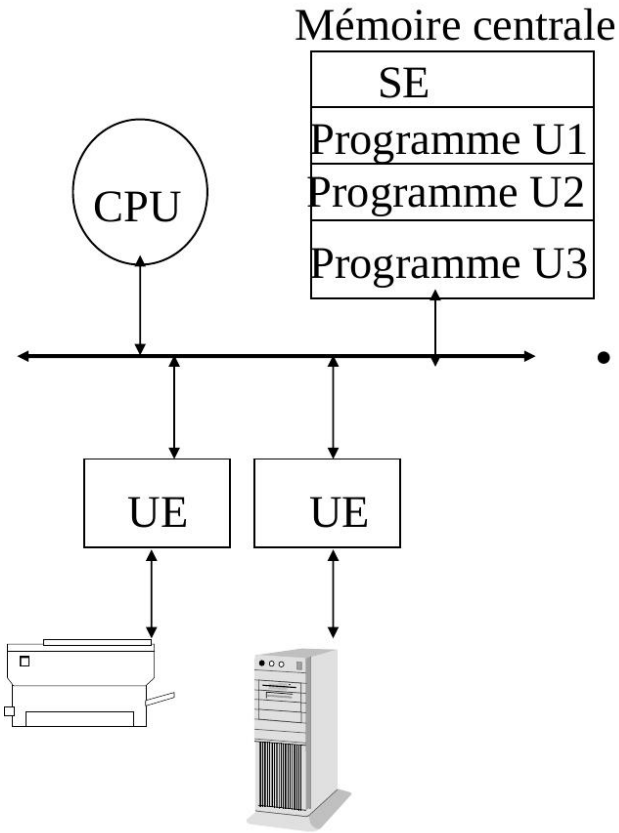
C'est une machine virtuelle plus facile d'emploi qui est construite au dessus du matériel.



Le rôle d'un système d'exploitation

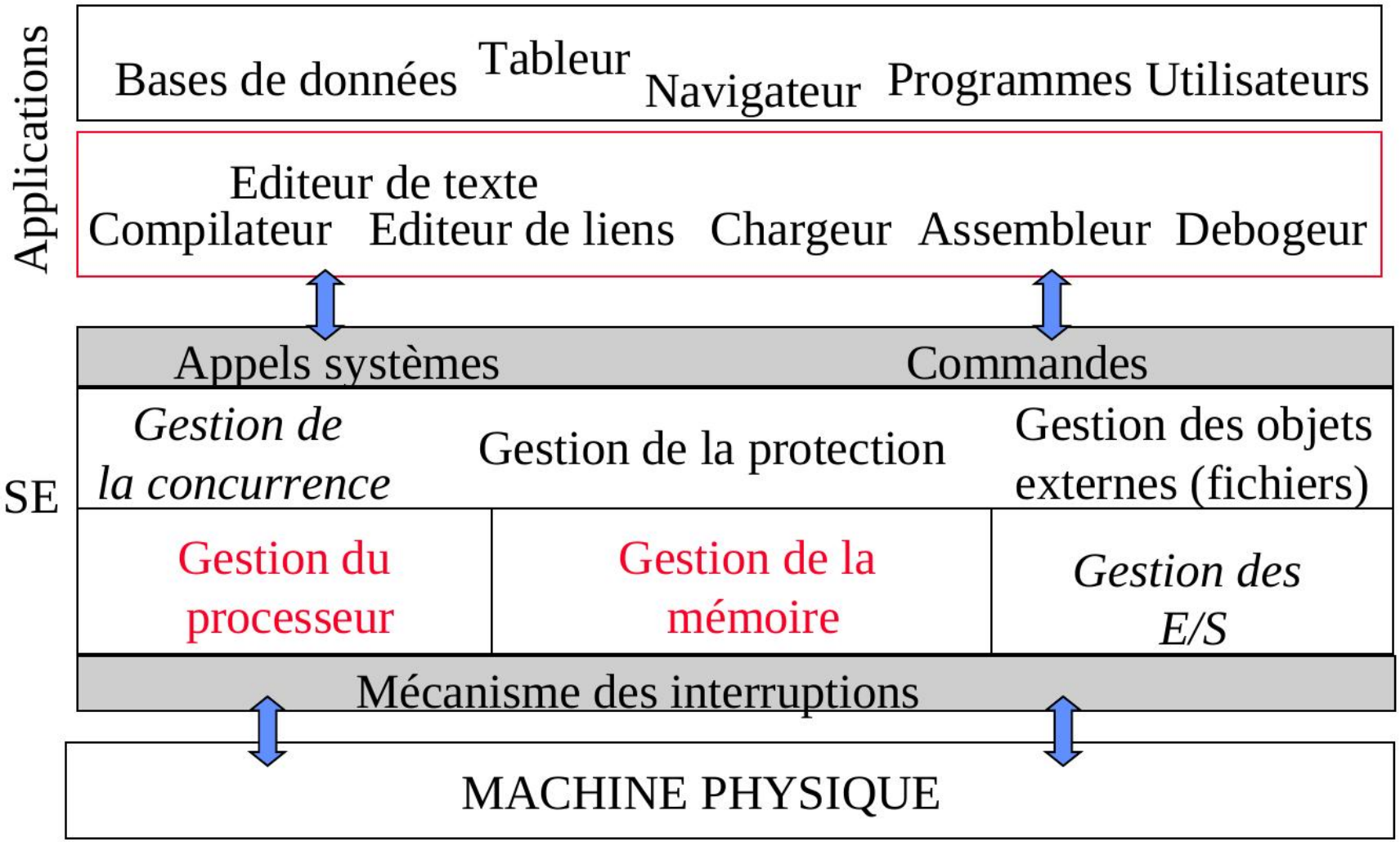
Son rôle est d'exploiter la machine physique pour en faciliter l'accès, le partage et pour l'optimiser.

Un de ses principaux objectif est la prise en charge de la gestion et du partage des ressources.



- le processeur : qui s'exécute ?
- la mémoire centrale :
  - où charger un programme ?
  - Comment protéger l'accès entre programmes ?
- Gestion des périphériques ?

Les fonctions d'un système d'exploitation

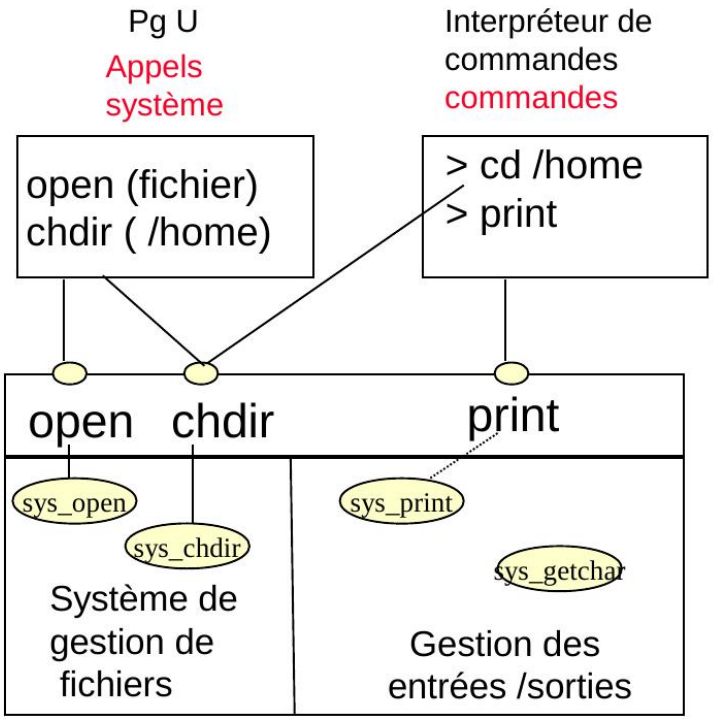


2. Les appels système et les commandes

Les appels systèmes et les commandes

Les fonctionnalités du système d'exploitation sont accessibles par le biais de fonctions prédéfinies : les commandes ou les appels système.

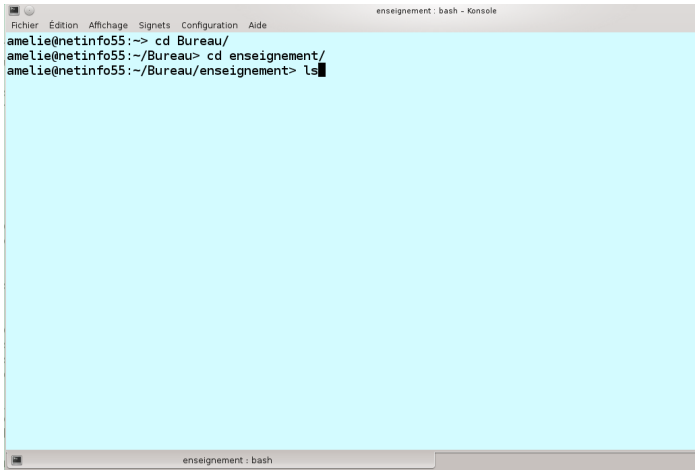
- Les commandes sont prises en compte par un interpréteur de commande, qui invoque la fonction système correspondante.
- Les appels système sont des fonctions des langages de programmation, accessibles via des bibliothèques.

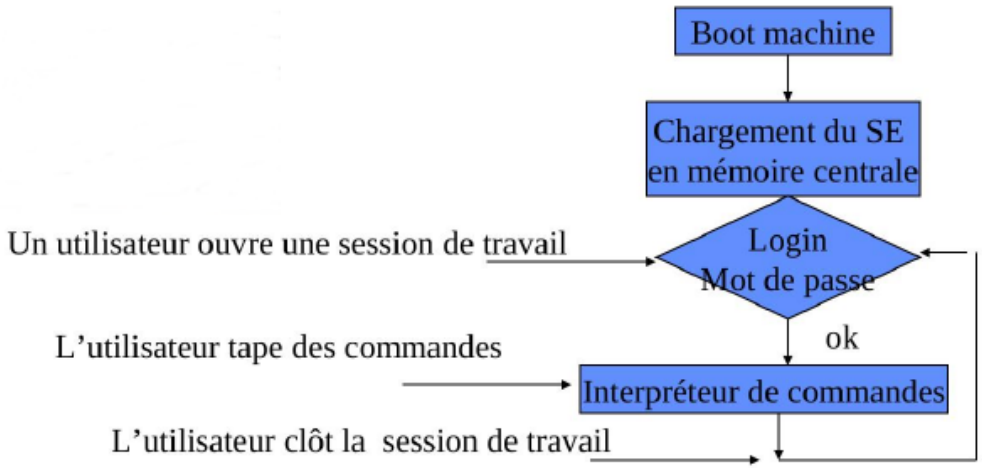


Les appels système ou les commandes peuvent référencer les mêmes fonction système.

Interpréteur de commandes

Un interpréteur de commandes est un programme du système d'exploitation. Son rôle est de traiter les commandes tapées au clavier par l'utilisateur.






```
TANT QUE l'utilisateur ne ferme pas la session FAIRE
 Emettre un signe d'invite
 Lire la ligne courante
 Exécuter la commande indiquée sur cette ligne
FIN
```

Modes d'exécutions

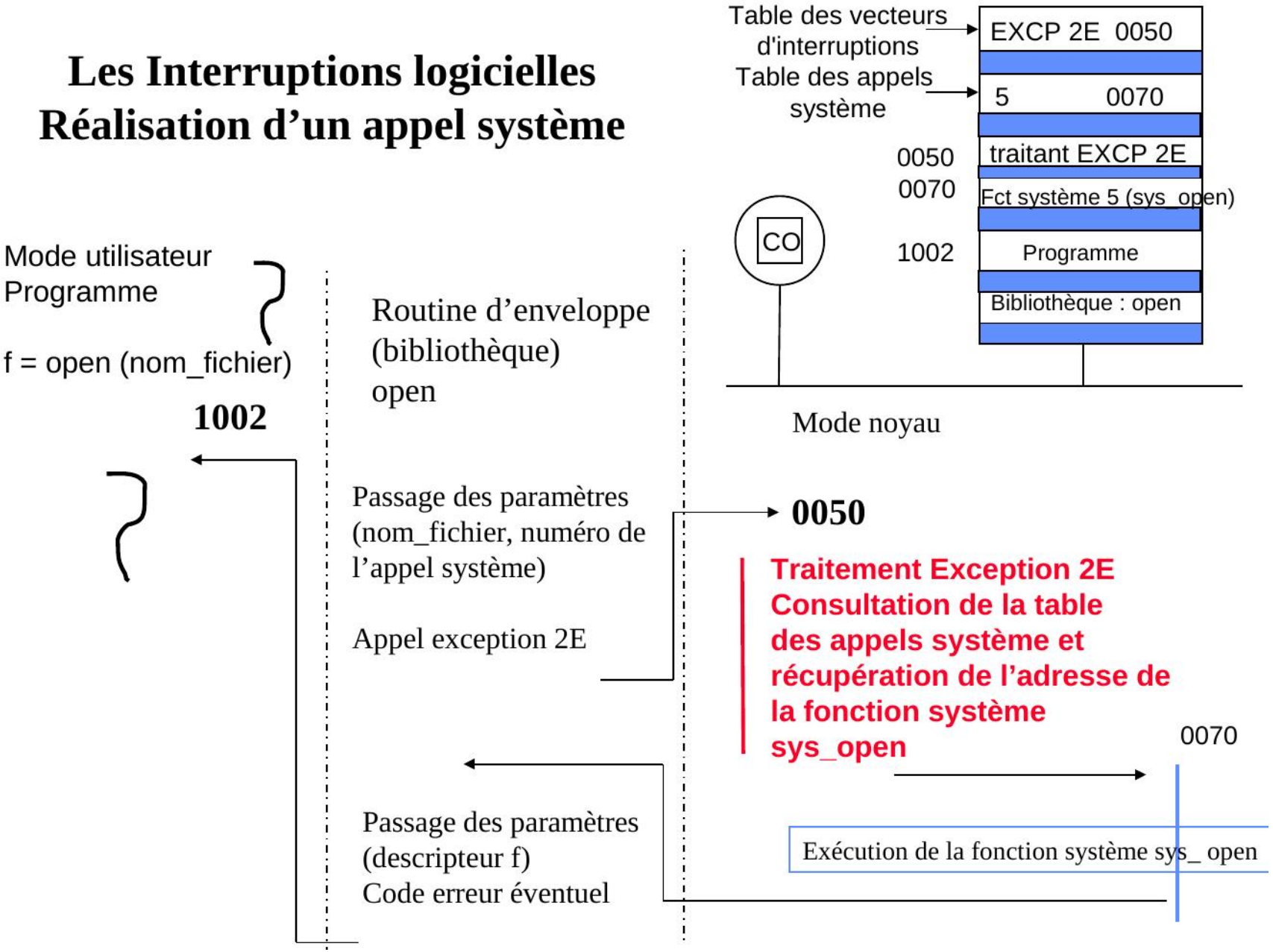
Il existe deux modes d'exécutions :

- **Le mode utilisateur** dans lequel s'exécutent tous les programmes utilisateurs.
- **Le mode superviseur** dans lequel s'exécutent les programmes plus sensibles du système d'exploitation, notamment les appels systèmes ou les interruptions.

Réalisation d'un appel système

- Les fonctions du système sont exécutées en mode superviseur.
- Le basculement du mode utilisateur au mode superviseur s'effectue par la levée de l'exception particulière
- Chaque fonction du système est connue au niveau du système d'exploitation, par un numéro (exemple open, 5). Une table des appels système associe à chaque numéro de fonction, son adresse en mémoire centrale.

Réalisation d'un appel système : Exemple

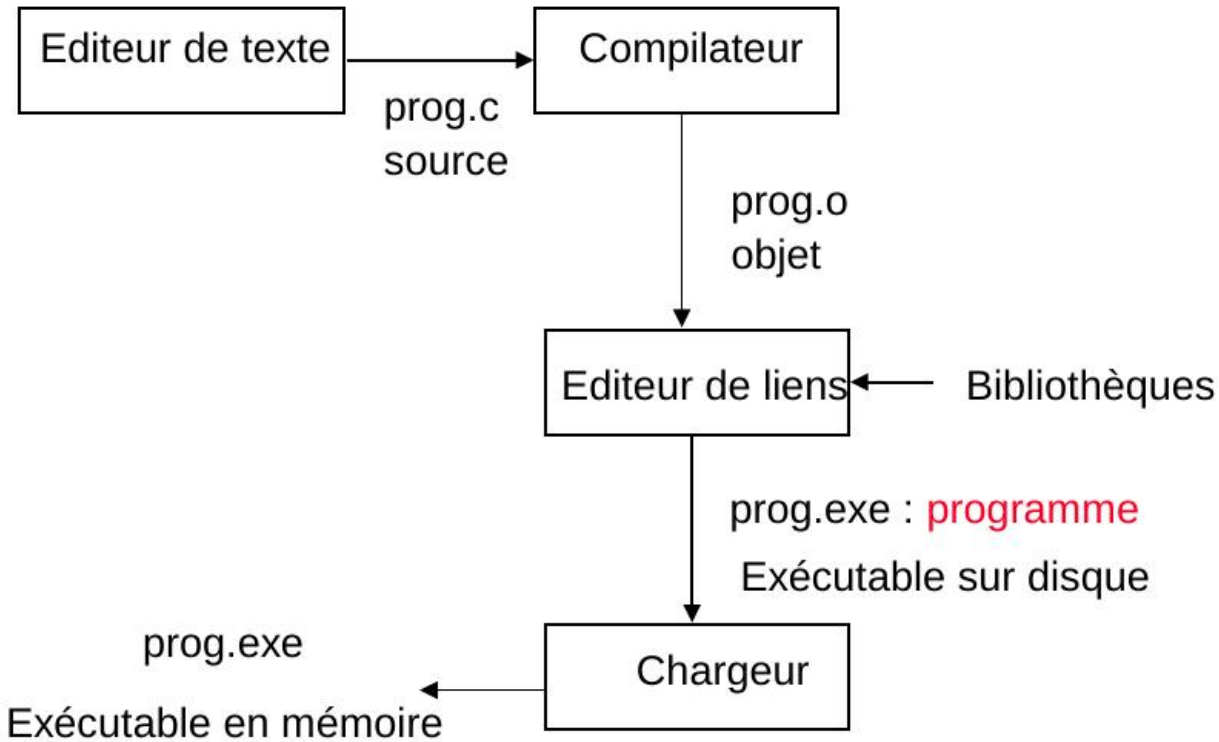


3. La chaîne de production de programmes

La chaîne de production de programmes

Cette chaîne désigne l'ensemble des étapes nécessaires à la construction d'un programme exécutable à partir d'un programme dit source :

- Compilation
- Edition des liens
- Chargement



Rôle d'un ordinateur

1. J'ai un problème à résoudre : Calculer le périmètre d'un rectangle de longueur a et de largeur b.

2. J'écris un algorithme :
   `Perimetre := 2a + 2b`

3. Grâce à un langage de programmation, je code la solution pour la faire exécuter par l'ordinateur $\Longrightarrow$ Programme constitué d'instruction :

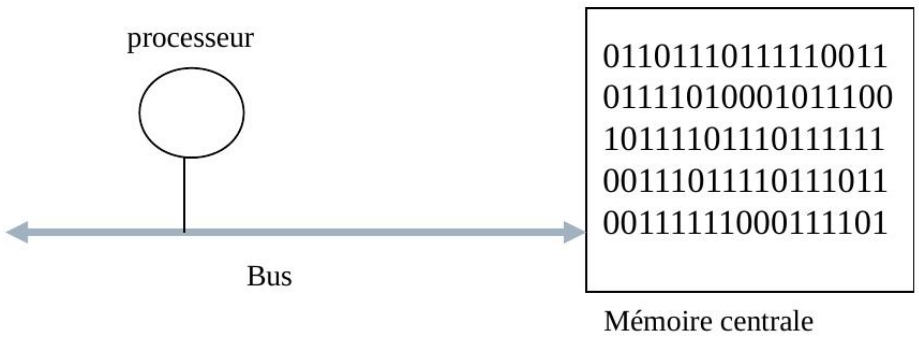
```
public static double perimetre (double a, double b){
  double perimetre;
  perimetre = 2*a +2*b;
  return perimetre;
}
```

Le codage d'un problème

```
public static double perimetre (double a, double b){
   double perimetre;
   perimetre = 2*a +2*b;
   return perimetre;
}
```

Programme en langage de haut niveau : instructions de haut niveau

Compilateur



Programme à exécuter : instructions machine et valeurs en binaire

Structure d'un langage de haut niveau (1/2)

Un langage de haut niveau s'appuie sur :

- **Un alphabet :** symboles élémentaires disponibles (caractères, chiffres, ponctuations),
- **Des identificateurs :** groupe de symboles de l'alphabet (ex : `AB12`),
- **Des déclarations ou instructions :** séquences d'identificateurs et de symboles formés selon la syntaxe du langage (ex : `A1 = 3;`).

Un programme est une suite de déclarations et d'instructions du langage qui respectent la syntaxe du langage.

Structure d'un langage de haut niveau (2/2)

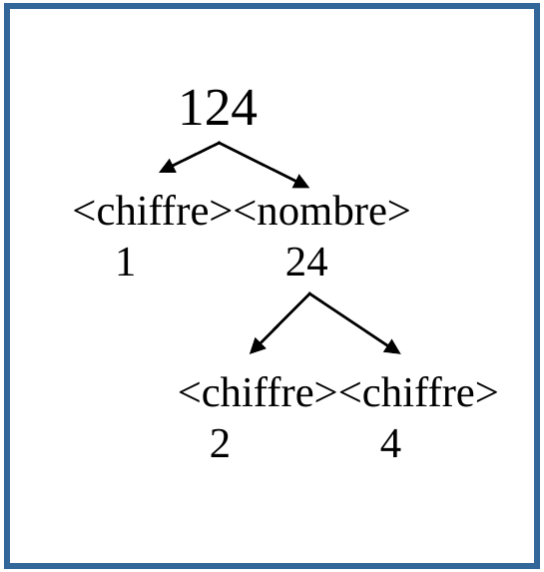
Il faut exprimer la syntaxe du langage. On utilise pour cela la notation de BACKUS-NAUR (BNF)

< objet du langage >::=<objet du langage> | symbole

| représente une alternative
<> entoure les objets du langage

Exemple :

<nombre>::=<chiffre> |<chiffre><nombre>
<chiffre>::= 0|1|2|3|4|5|6|7|8|9



Exemple

```
PROGRAM Z
INT A
INT B
DEBUT
A:=5
B:=A/2
FIN
```

<programme>::=PROGRAM <identificateur><corps de programme>
<corps de programme>::=<suite de declarations> DEBUT <suite d'affectations> FIN
<suite de declarations>::=<declaration>
|<declaration><suite de declarations>
<declaration>::=INT <identificateur>
<suite d'affectations>::=<affectation>
|<affectation><suite d'affectations>
<affectation>::=<identificateur >:=<terme>
|<terme><operateur><terme>
<terme>::=<entier> |<identificateur>
<operateur>::= +|−|∗|/
<identificateur>::=<lettre> |<lettre><chiffre>
<entier>::=<chiffre> |<chiffre><entier>
<lettre>::= $A|B|C|D|E....|X|Y|Z$
<chiffre>::= 0|1|2|3|4...|9

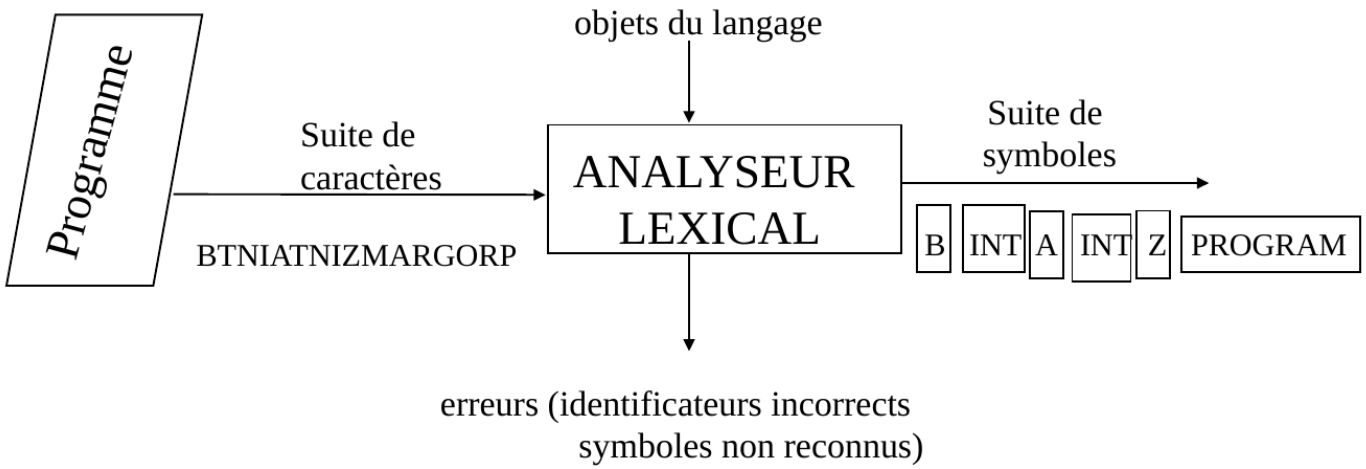
Le rôle du compilateur

- Un **compilateur** traduit un **programme source** écrit en langage de haut niveau en un **programme objet** en langage de bas niveau.
- Le compilateur est lui-même un programme important et volumineux.
- Le travail du compilateur se divise en plusieurs phases :
  - **analyse lexicale** (recherche des mots-clés)
  - **analyse syntaxique** (vérification de la syntaxe)
  - **analyse sémantique** (vérification de la sémantique)
  - **génération du code objet**

Analyse lexicale

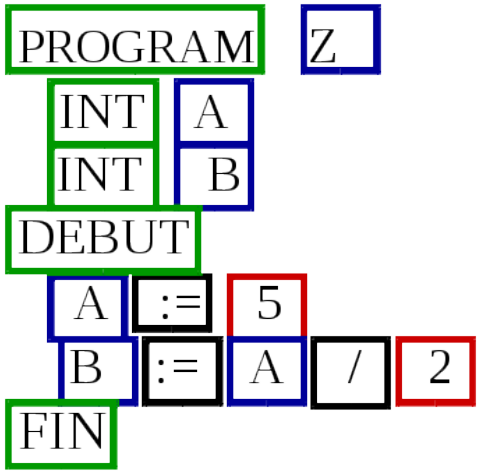
Rôle de l'analyse lexicale :

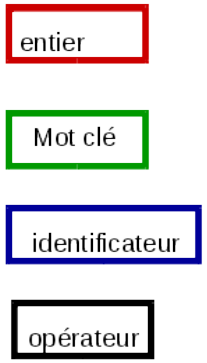
- reconnaître dans la suite de caractères que constitue un programme les objets du langage
- éliminer le "superflu" (espaces, commentaires)



Analyse lexicale : Exemple

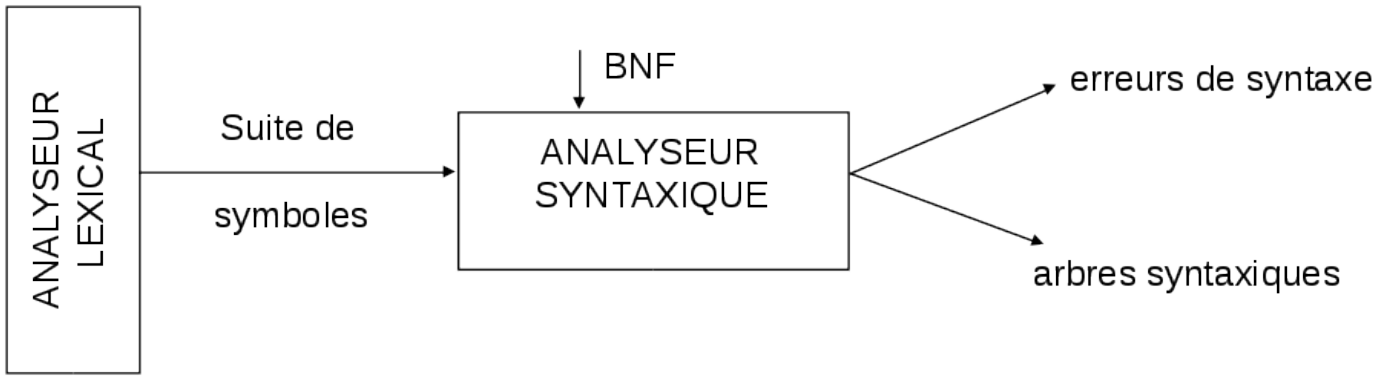
```
PROGRAM Z
INT A
INT B
DEBUT
A:=5
B:=A/2
FIN
```



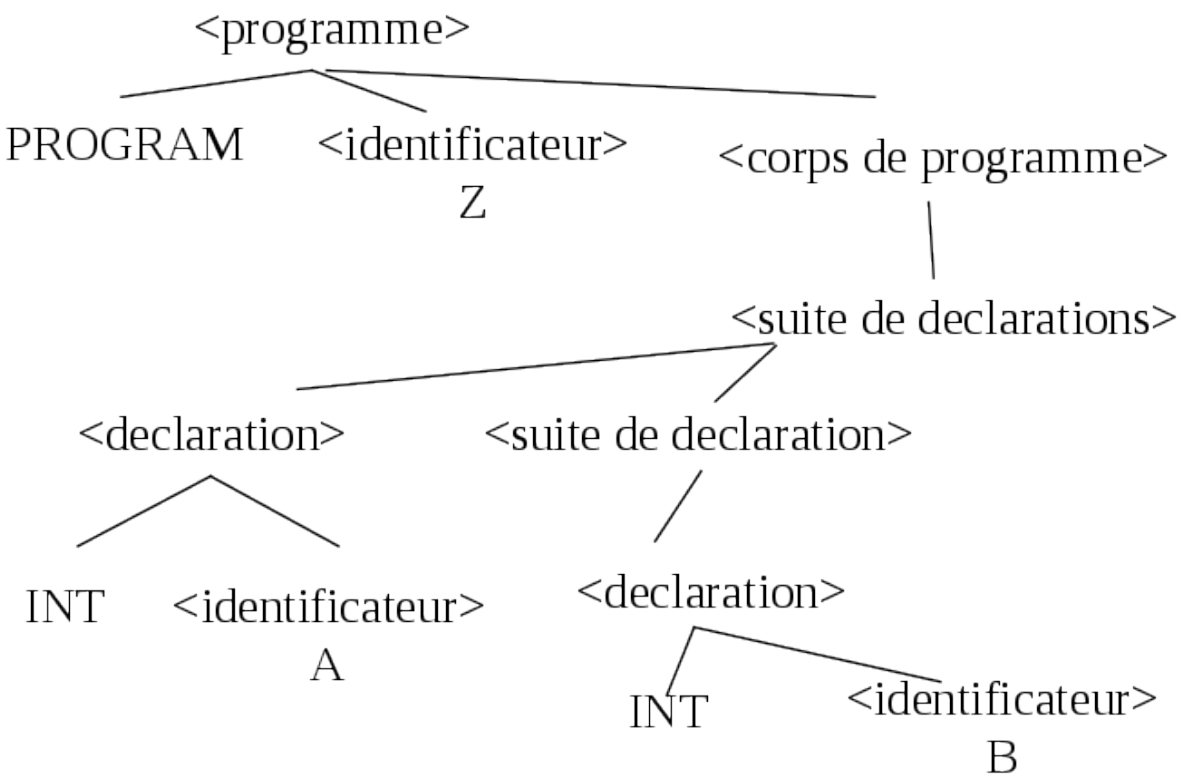


Analyse syntaxique

Rôle de l'analyse syntaxique : reconnaître si la suite de symboles issue de l'analyse lexicale respecte la syntaxe du langage

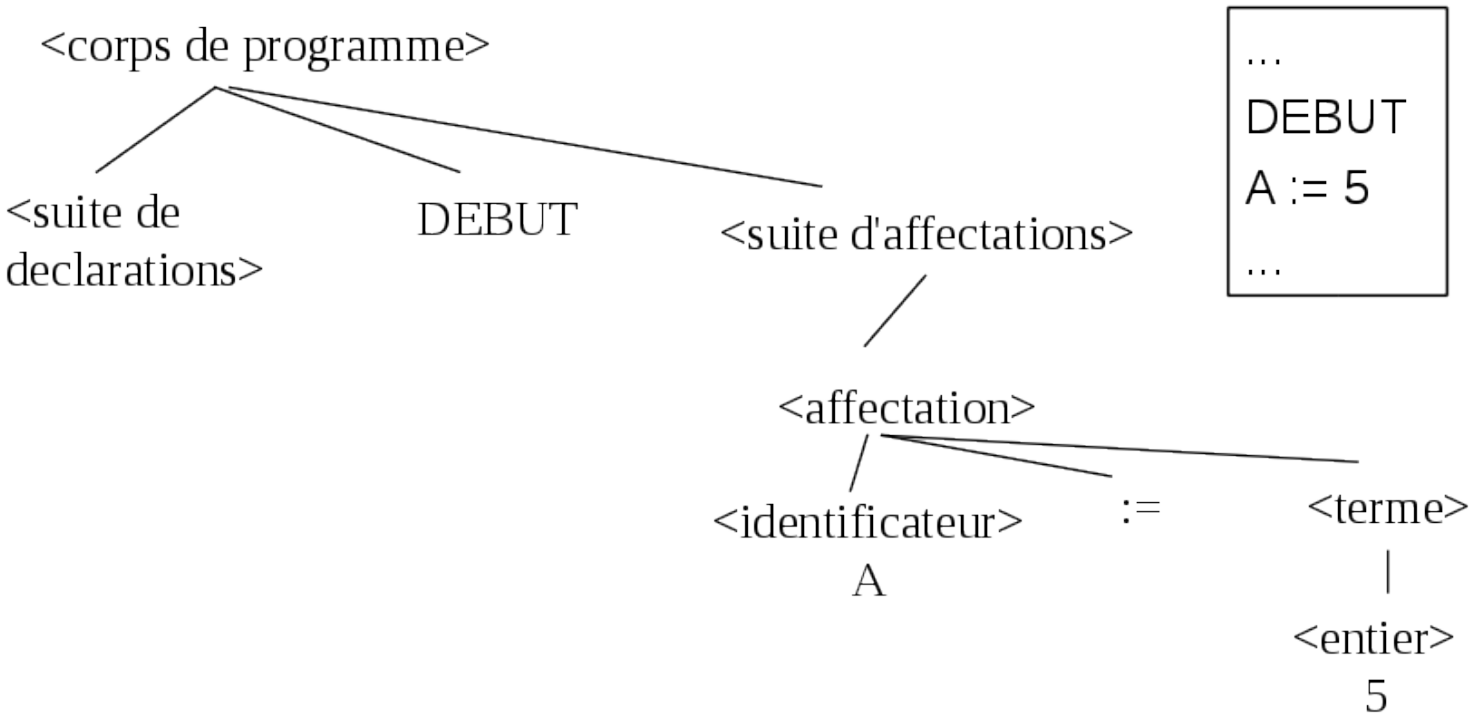


Analyse syntaxique : Exemple (1/3)

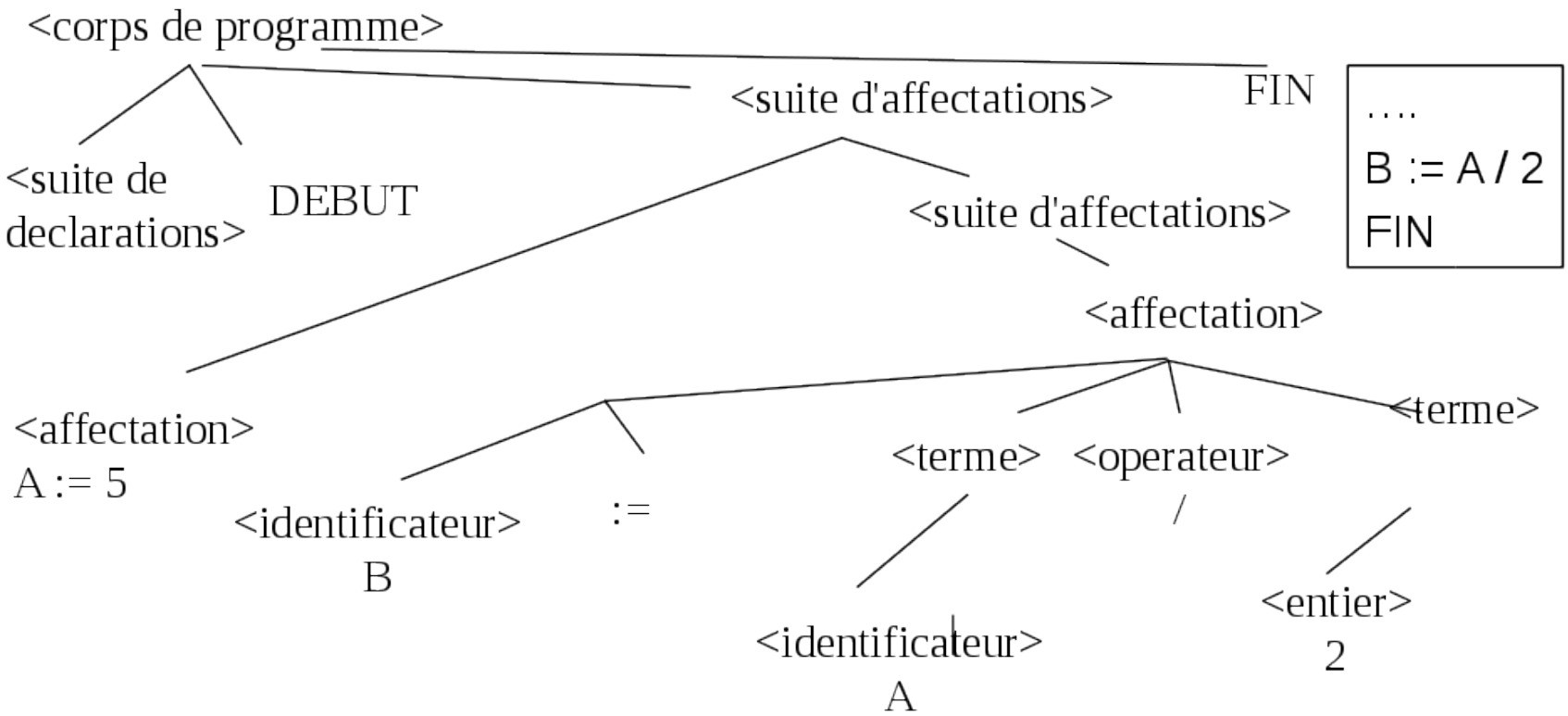



```
PROGRAM Z
    INT A
    INT B
    .....
```

Analyse syntaxique : Exemple (2/3)



Analyse syntaxique : Exemple (3/3)



<corps de programme> ::= <suite de declarations> DEBUT <suite d'affectations> FIN

<suite d'affectations> ::= <affectation> | <affectation><suite d'affectations>

<affectation> ::= <identificateur > := <terme> | <terme> <operateur><terme>

<terme> ::= <entier> | <identificateur>

<operateur> ::= + | - | * | /

Analyse Sémantique

Rôle de l'analyse sémantique : trouver le sens et la signification des différentes phrases du langage

- quels sont les objets manipulés et leurs propriétés ? (type, durée de vie, taille, adresse,...)
- contrôler la cohérence dans l'utilisation des objets. (erreur de type, absence de déclarations, déclarations multiples, déclarations inutiles, expressions incohérentes,...)

Analyse Sémantique : Exemple

```
float i;
for (i=1;i<=5;i++)
   tab[i]=3;
```

l'indice de parcours `i` du tableau est de type réel.

```
int a;
int b;
a = 5;
b = a/2;
```

Le résultat de type réel affecté à un entier.

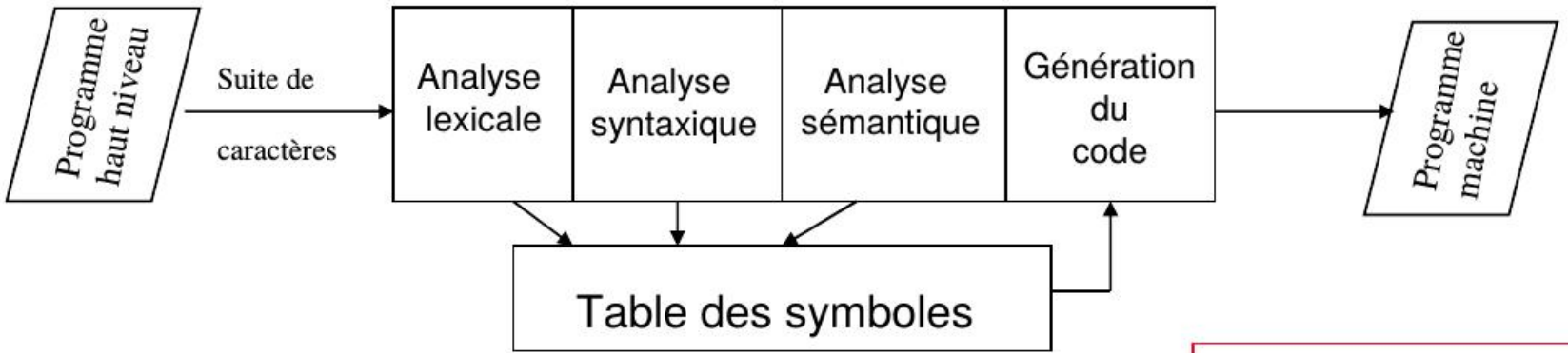
Table des symboles

- Le compilateur manipule une table des symboles qui contient toutes les informations sur les propriétés des objets du programme

- La table est construite durant les 3 phases d'analyse lexicale, analyse syntaxique et analyse sémantique.

nom	type	taille	adresse
A	entier	4 octets	$(0)_H$
B	entier	4 octets	$(4)_H$

Génération du code



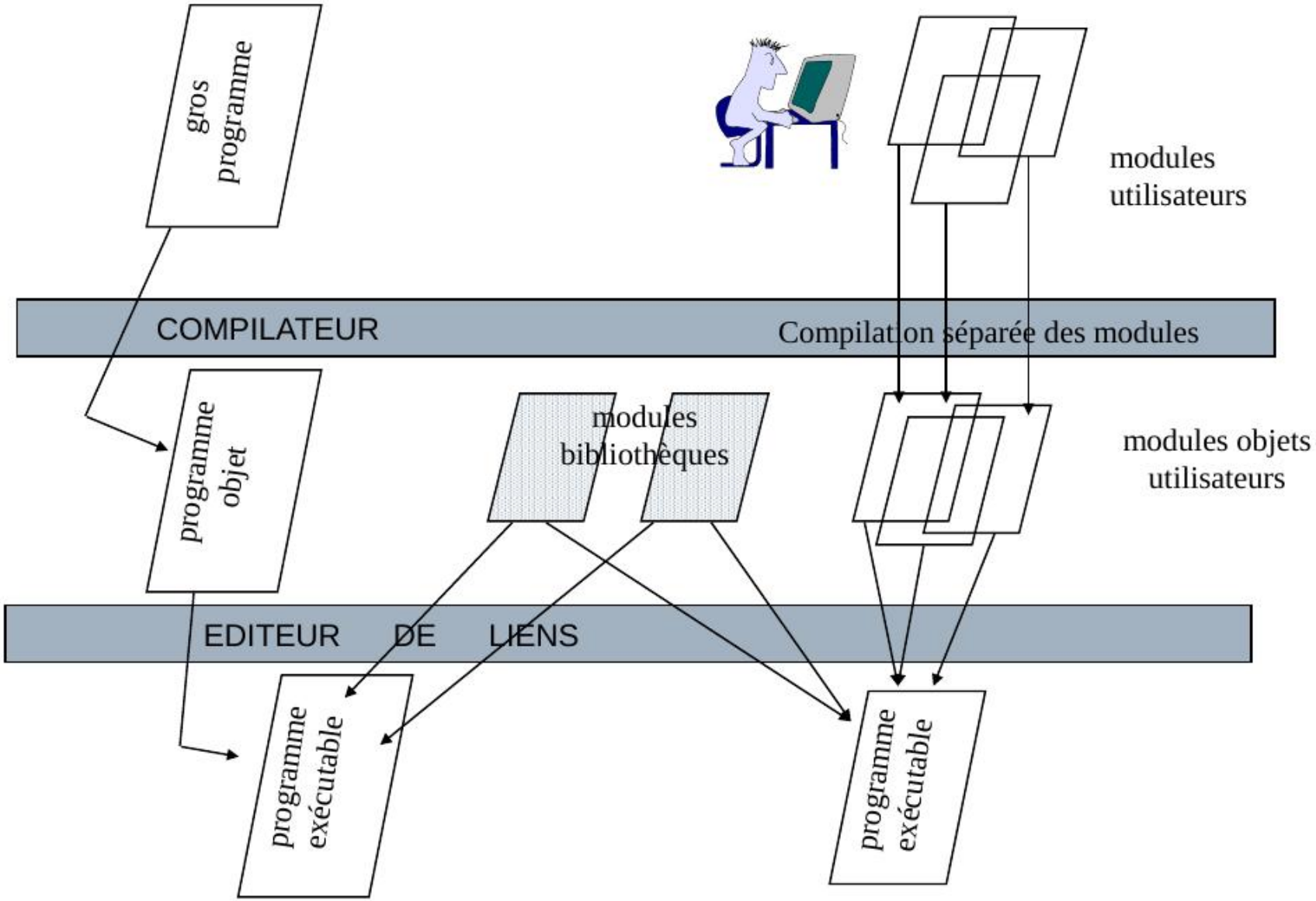
- La génération de code est l'étape ultime de la compilation.
- Elle consiste à produire le code machine équivalent du code en langage haut niveau. Ce code machine est qualifié de relogeable car les adresses dans ce code sont calculées à partir de 0

$(0)_H$
$(4)_H$
$(8)_H$ 00000 000 0001 $(4)_H$
$(C)_H$ 00000 000 0001 $(4)_H$
$(10)_H$ 00001 001 0001 $(0)_H$

$(14)_H$ 00000 000 0001 $(2)_H$
$(18)_H$ 00001 001 0001 $(4)_H$
$(1C)_H$ 00000 000 0001 $(6)_H$
$(20)_H$ 00001 000 0001 $(8)_H$

Edition de liens

Le développement d'un gros programme



Rôle de l'éditeur de liens (1/4)

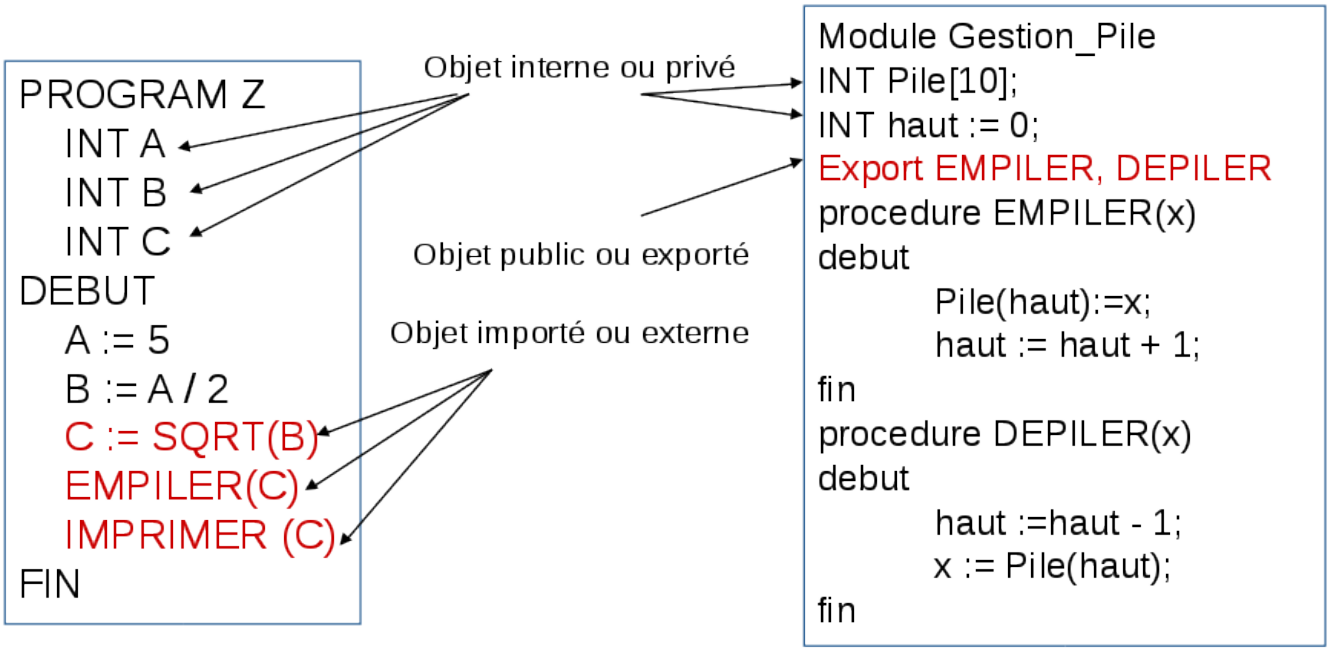
Un éditeur de liens est un logiciel qui permet de combiner plusieurs modules objet obtenus par compilation séparée pour construire un seul programme exécutable

- modules objets utilisateur
- modules objets prédéfinis dans des bibliothèques (fonctions interfaces des appels systèmes, fonctions mathématiques, fonctions graphiques, ...)

Rôle de l'éditeur de liens (2/4)

Un module comprend trois catégories d'objets

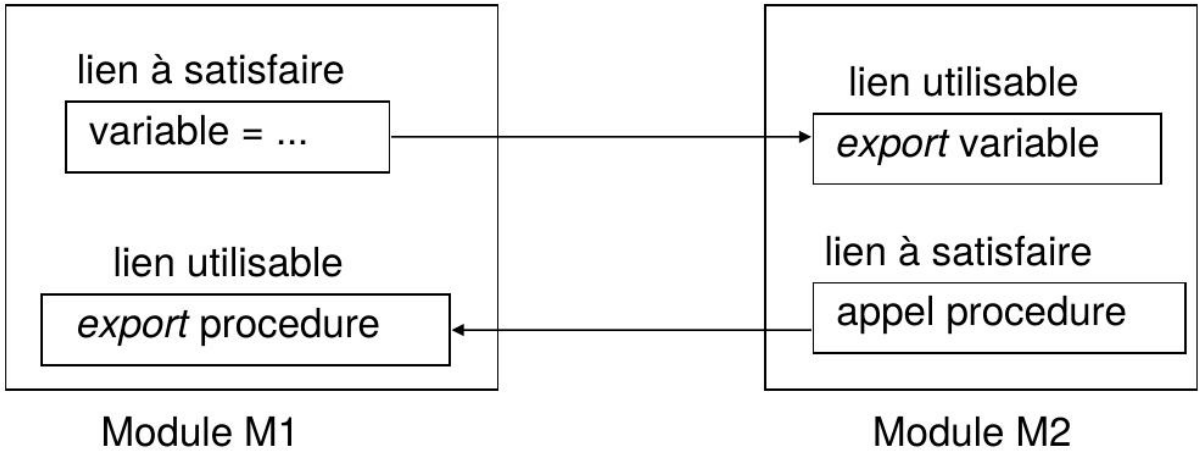
- **objet interne au module :** inaccessible de l'extérieur.
- **objet exporté ou public :** interne au module mais accessible de l'extérieur.
- **objet importé ou externe :** n'appartenant pas au module, mais utilisé par le module.



Rôle de l'éditeur de liens (3/4)

Le compilateur recense dans chaque module les objets internes, exportés ou importés. Pour chaque objet rencontré, selon sa catégorie :

- si l'objet est interne, il associe une adresse à l'objet dans la table des symboles.
- si l'objet est exporté, il lui associe une adresse à l'objet dans la table des symboles et publie cette adresse sous forme d'un lien utilisable.
- si l'objet est importé, il ne connaît pas l'adresse à l'objet. Il demande à obtenir cette adresse sous forme d'un lien à satisfaire.



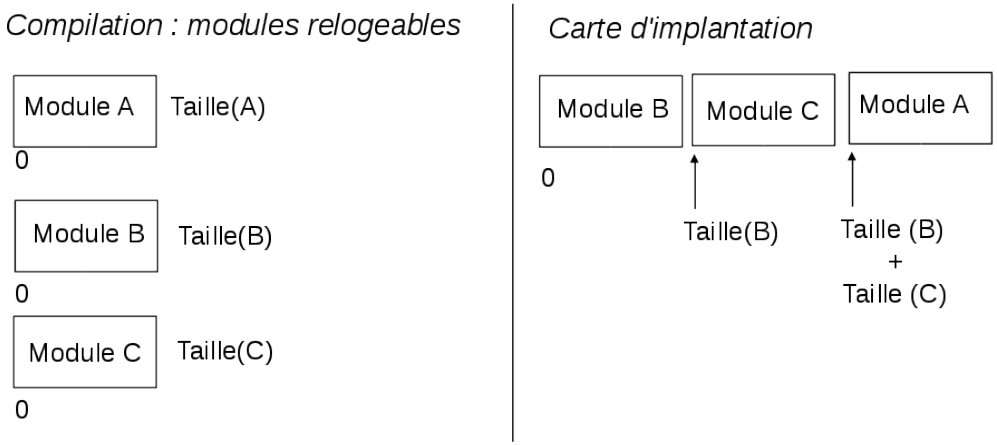
Rôle de l'éditeur de liens (3/4)

L'éditeur de liens doit construire le programme exécutable final à partir des modules objet entrant dans sa composition. Il procède en trois étapes :

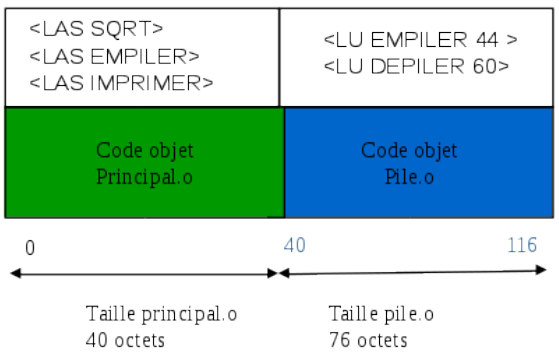
1. Construction de la carte d'implantation du programme
2. Construction de la table des liens utilisables
3. Construction du programme exécutable final

Construction de la carte d'implantation du programme

Détermination des adresses d'implantation de chaque module utilisateur du programme, placés les uns derrière les autres



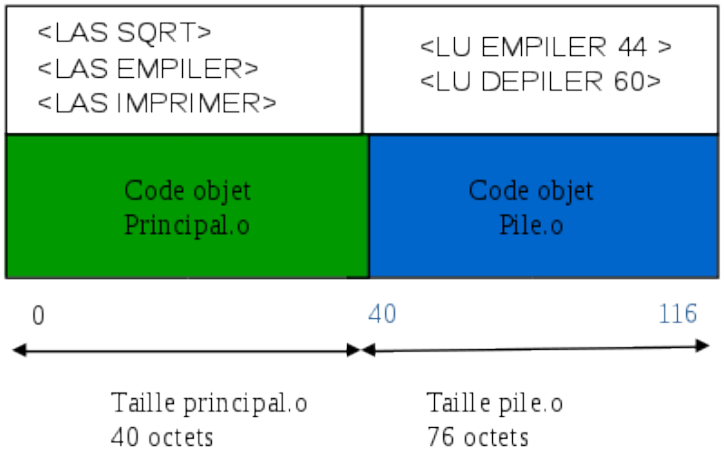
Exemple :



Construction de la table des liens utilisables

```
Pour chaque lien dans chaque module faire
  Si (lien n'est pas dans la table) alors
    créer une entrée pour lien
    Si (lien est un LU) alors
      Associer à lien son adresse selon la carte
    FinSi
    Si (lien est un LAS) alors
      Associer à lien adresse indefinie
    FinSi
  Sinon
    Si (lien est un LU) et (lien est en adresse indéfinie) alors
      associer à nom de lien son adresse selon la carte
    FinSi
  FinSi
FinPour
```

Construction de la table des liens utilisables : exemple



Nom de lien	Adresse /carte
SQRT	? Bib math
EMPILER	? 44 + 40
IMPRIMER	? Bib E/S
DEPILER	60 + 40

Construction de l'exécutable final

L'éditeur de liens construit le code exécutable :

1. Il remplace les occurrences des objets figurant dans les LAS par leur adresse dans la table des liens
2. Il translate les adresses des objets dans les modules de la valeur de l'adresse d'implantation du module dans la carte.

4. Les processus

Notion de processus

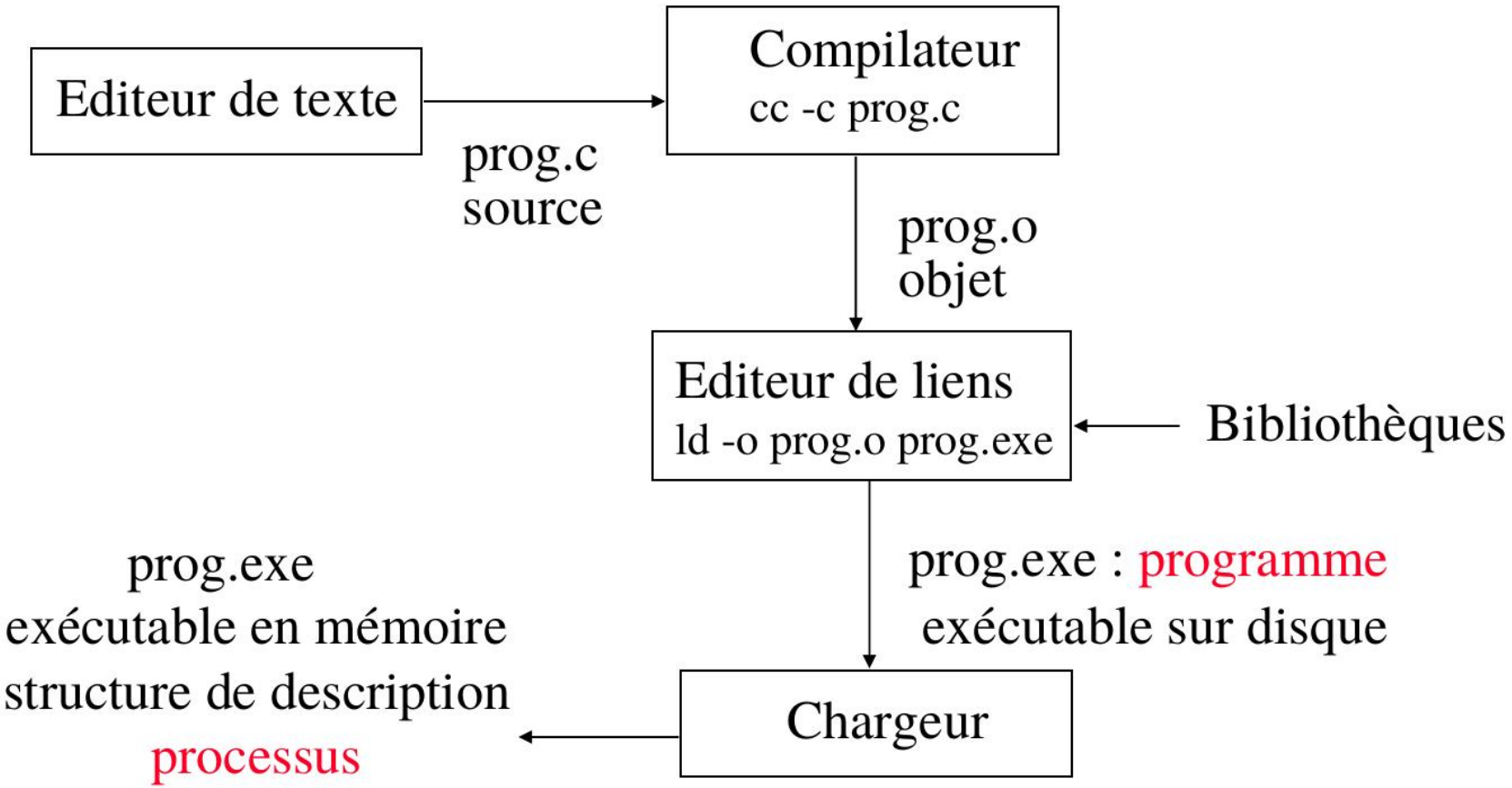
Définitions d'un processus :

- 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.

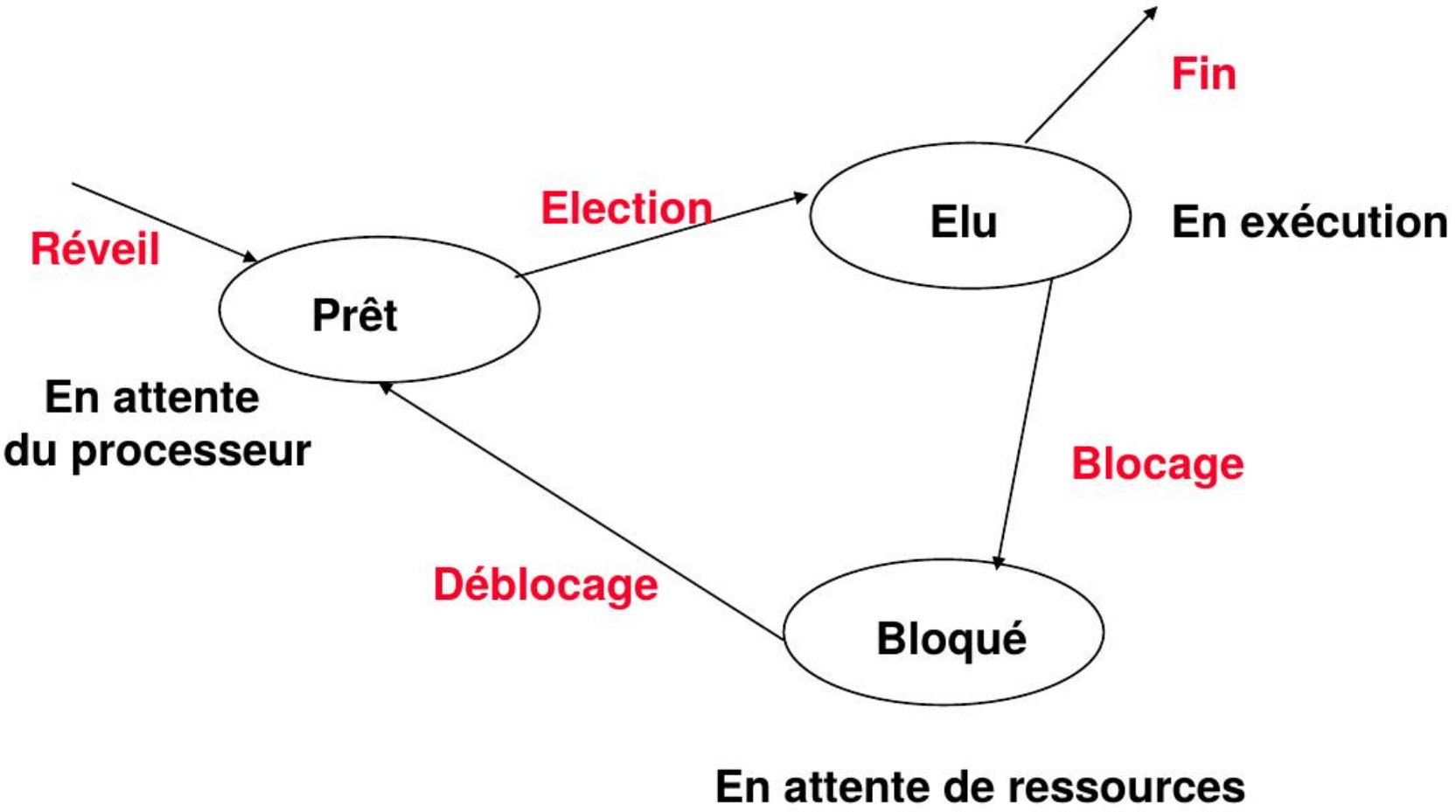
- Un processus est l'instance dynamique d 'un programme et incarne le fil d 'exécution de celui-ci dans un espace d'adressage protégé (objets propres : ensemble des instructions et données accessibles)

- Un programme réentrant est un programme pour lequel il peut exister plusieurs processus en même temps.

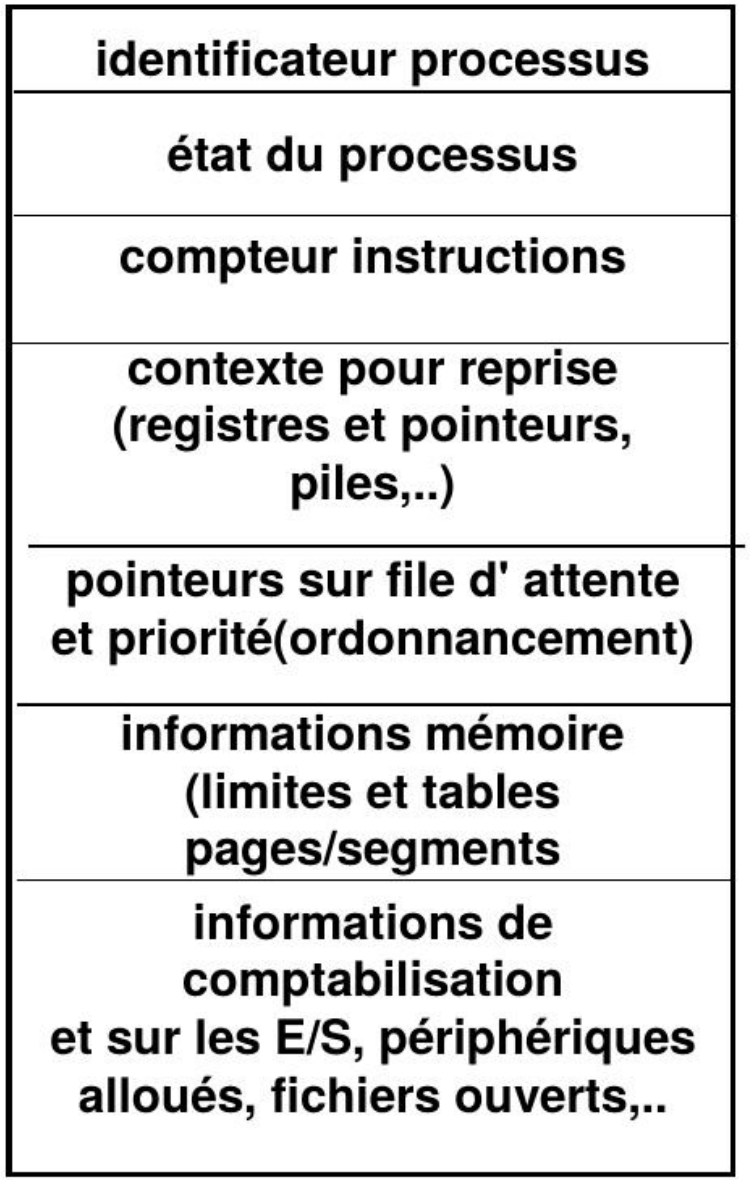
Du programme au processus



Système multiprocessus - Etats des processus



Bloc de contrôle du processus

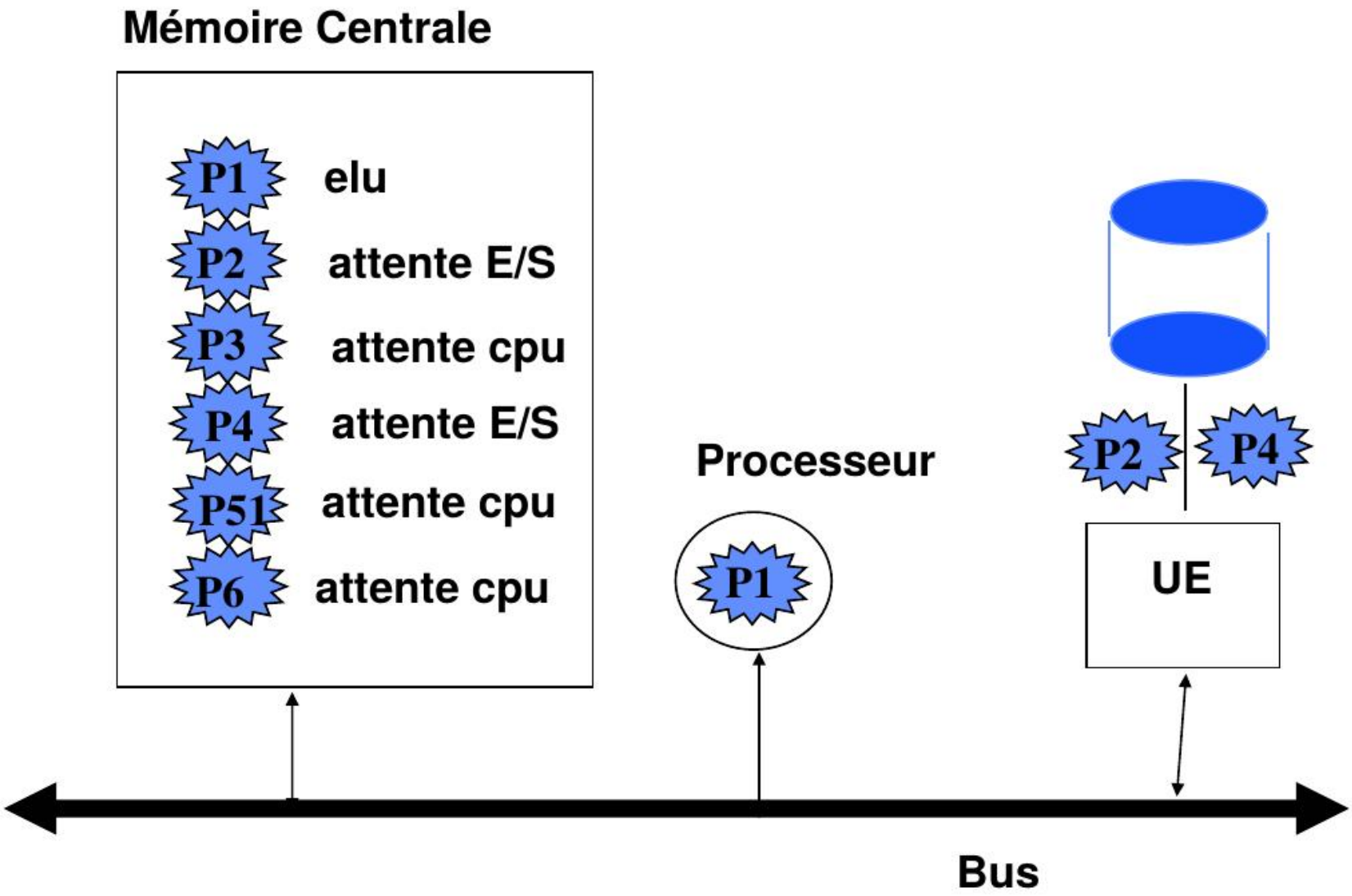


Les processus

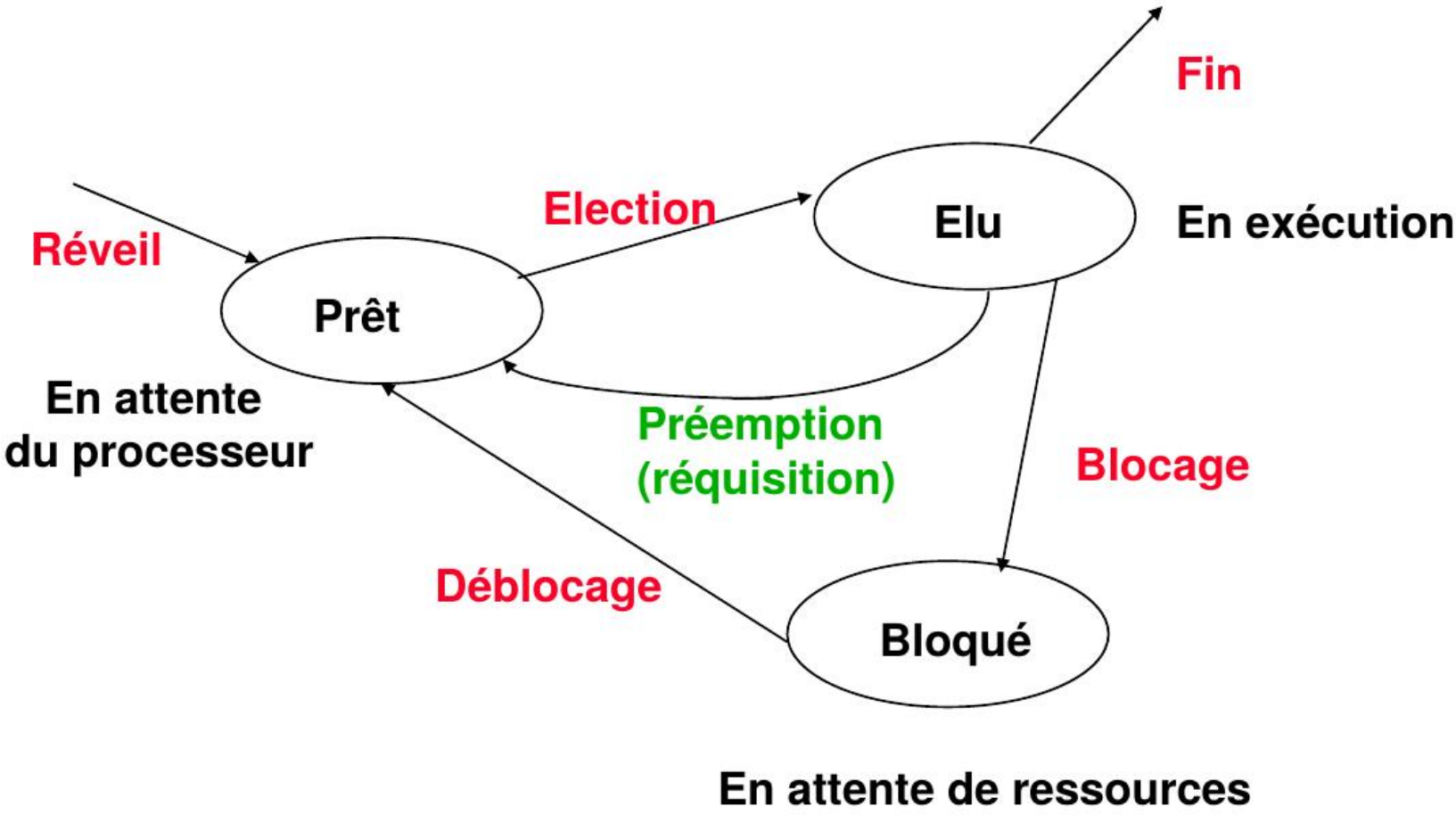
- **Opérations sur les processus :** Création, Terminaison normale ou erreur, Suspension, réveil.
- **Coopération entre processus :** Processus concurrents : indépendants ou coopérants. partage d'information (exemple fichier partagé) et synchronisation (division d'un programme en plusieurs tâches concurrentes)
- **Communication entre processus :** partage de mémoire et messages

5. Ordonnancement des processus

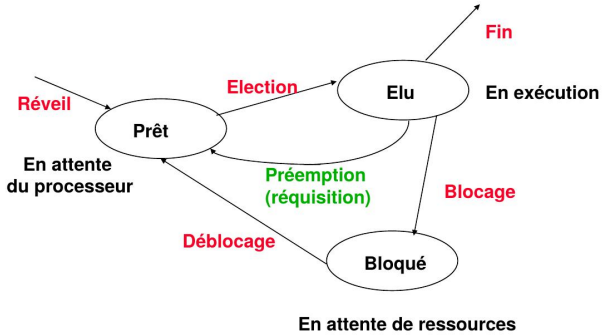
Système multiprocessus



Etats des processus (1/2)

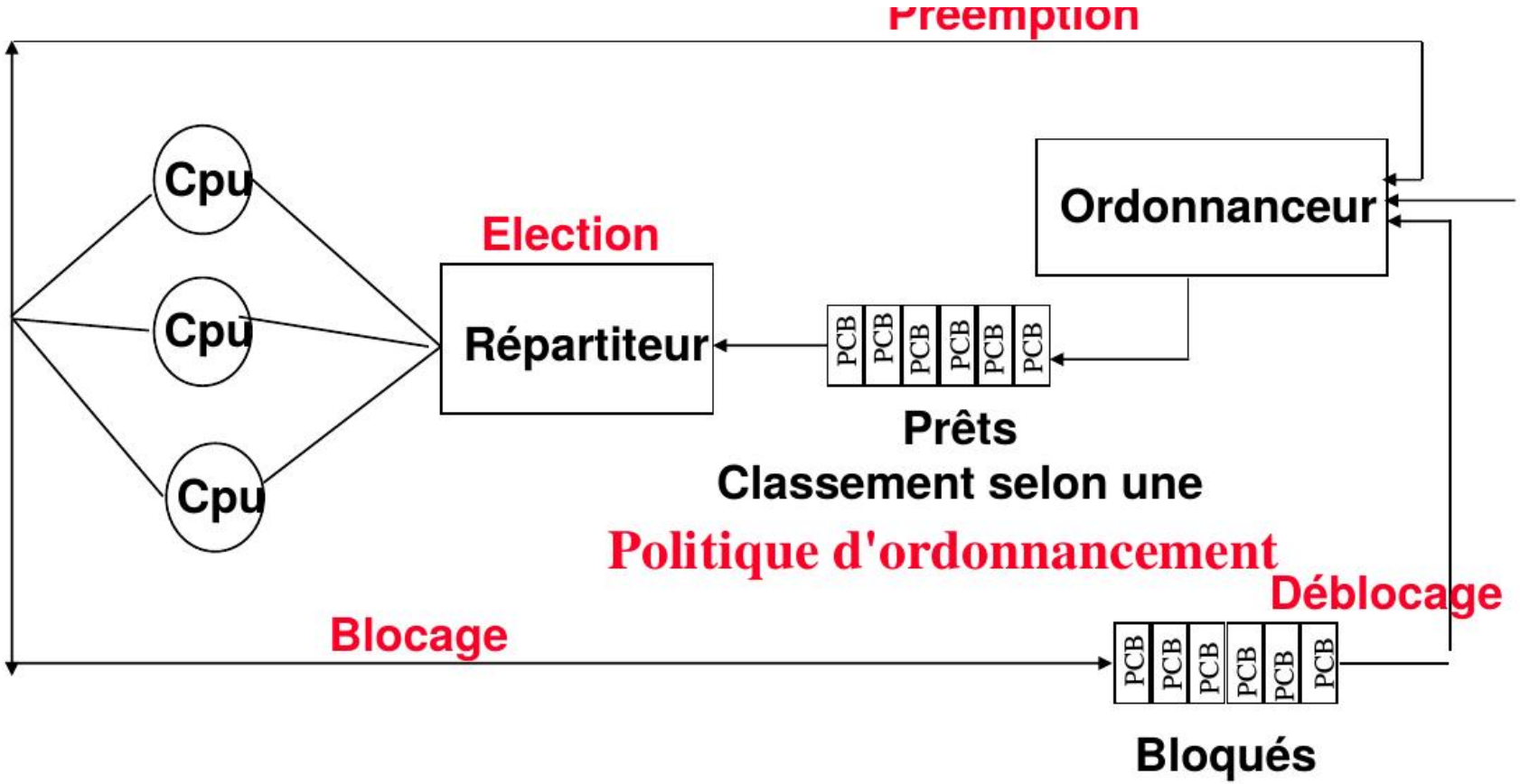


Etats des processus (2/2)



- **Election :** allocation du processeur
- **Préemption :** réquisition du processeur :
  - ordonnancement non préemptif : un processus élu le demeure sauf s 'il se bloque de lui-même
  - ordonnancement préemptif : un processus élu peut perdre le processeur s'il se bloque de lui-même (état bloqué) ou si le processeur est réquisitionné pour un autre processus (état prêt)

Système multiprocessus - Ordonnanceur et répartiteur



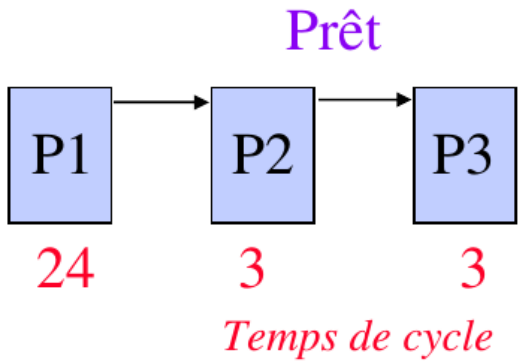
Politiques d'ordonnancement

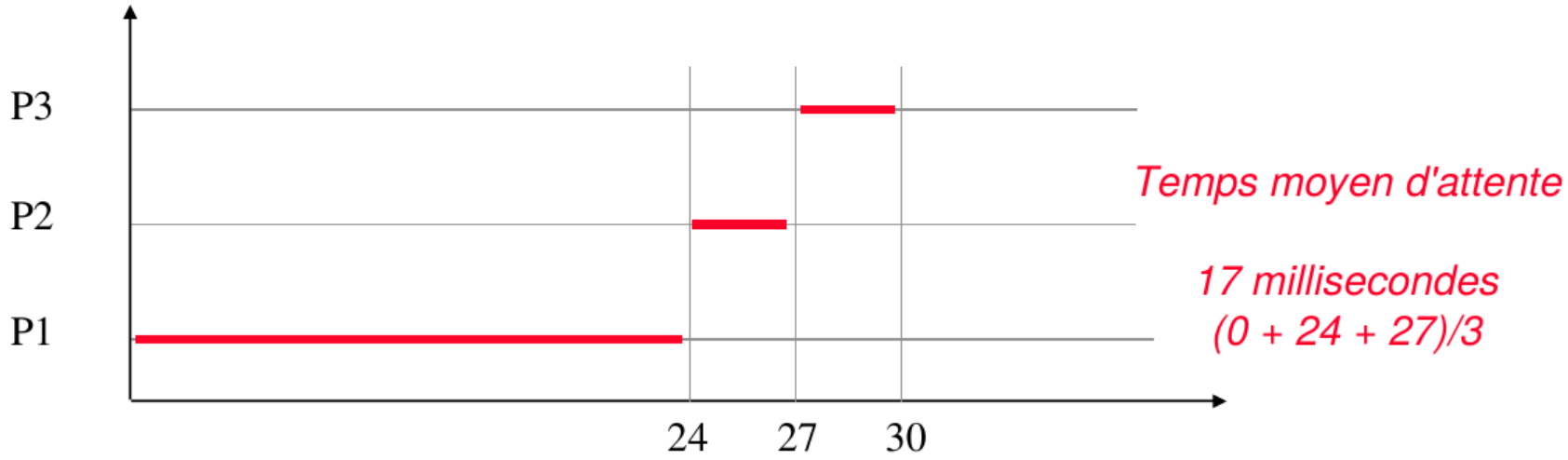
- Premier arrivé, premier servi (FIFO)
- Par priorités constantes : chaque processus reçoit une priorité, le processus de plus forte priorité est élu, algorithme avec ou sans préemption.
- Par tourniquet (round robin) : Un processus élu s'exécute au plus durant un quantum (intervalle de temps) ; à la fin du quantum, préemption et réinsertion en fin de file d'attente des processus prêts.

Ordonnancement FIFO

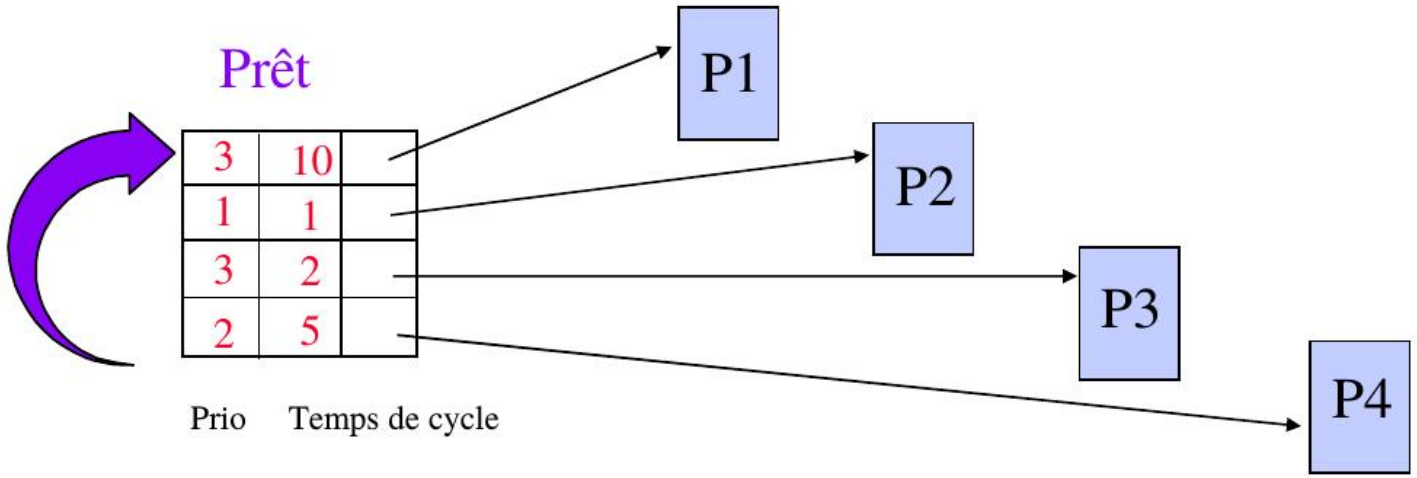
- **FIFO, sans réquisition**

Temps d'attente =
date de début d'exécution – date de soumission
Temps de réponse =
date de fin d'exécution – date de soumission

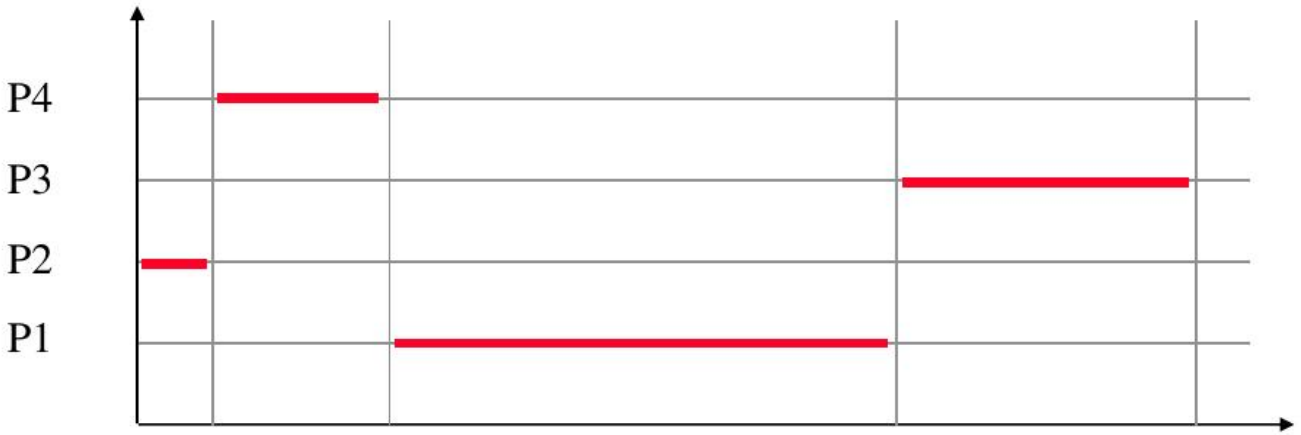




Ordonnancement par priorités constantes (sans préemption)



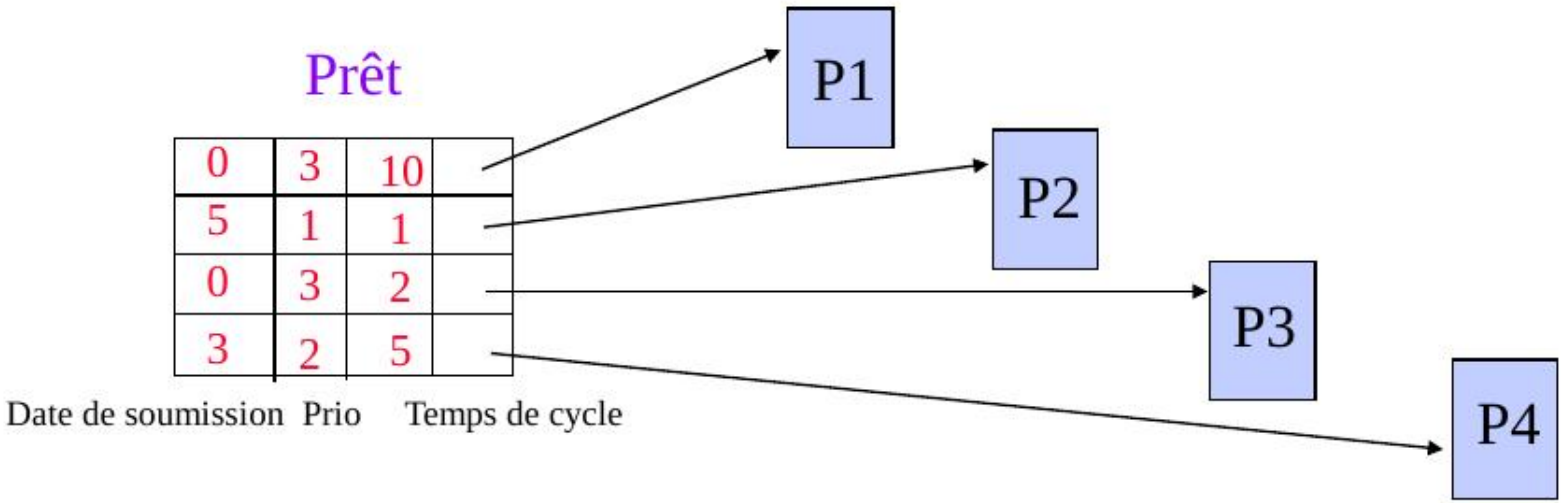
↳ Priorité : le plus petite valeur correspond à la plus forte priorité



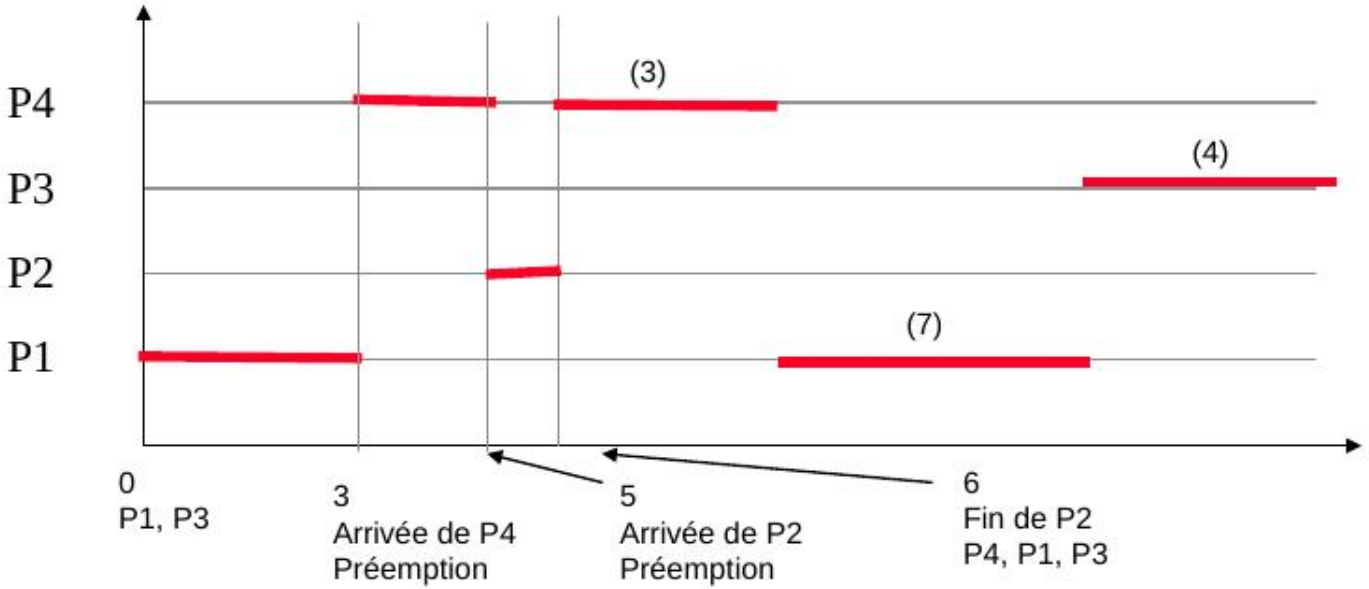
Temps moyen d'attente

8,2 millisecondes

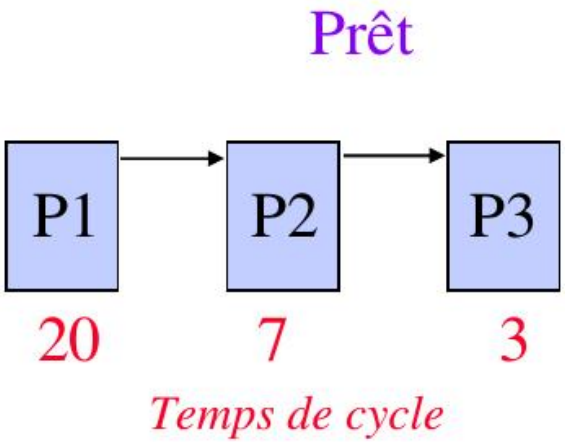
Ordonnancement par priorités constantes (avec préemption)

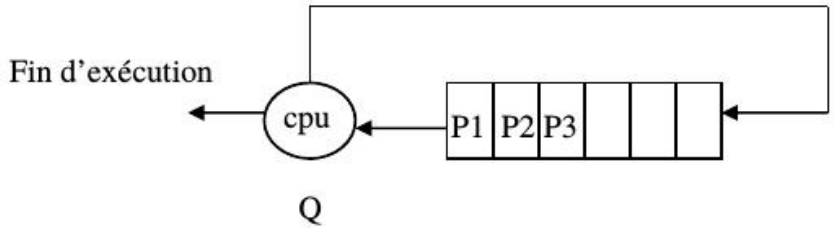


↳ Priorité : le plus petite valeur correspond à la plus forte priorité



Ordonnancement par tourniquet (round robin)





Quantum = 4

