Programmation Concurrente

Notion de Processus

Polytech/INFO 4, 2022-2023

Fabienne Boyer
UFR IM2AG, LIG, Université Grenoble Alpes
Fabienne.Boyer@imag.fr

POLYTECH GRENOBLE

UNIVERSITÉ Grenoble Alpes

L'exécution concurrente, omniprésente..

- **Tous les jours, vous utilisez votre téléphone mobile sans vous questionner :**
  - Vous pouvez téléphoner et recevoir des sms en parallèle
  - Vous pouvez déclencher un appel téléphonique depuis un sms
  - Votre alarme peut se déclencher pendant que vous êtes en train d'écouter de la musique ou regarder une vidéo
  - …
- **Toutes ces applications agissent de manière collaborative**
- **Cette collaboration repose sur des mécanismes dédiés fournis par le système d'exploitation**

L'exécution concurrente, omniprésente..

- **Tous les jours, vous utilisez votre browser pour glaner des informations sur le web:**
  - Vous êtes potentiellement des dizaines / centaines / milliers à accéder à certains sites simultanément
  - Mais vous n'avez pas le sentiment d'attendre...

- **Les serveurs Web sont capables de traiter un grand nombres de requêtes en parallèle**
- **Cette parallélisation repose aussi sur des mécanismes dédiés fournis par le système d'exploitation**

L'exécution concurrente, omniprésente..

- **Dans les exemples pré-cités, plusieurs *flôts d'exécution* sont en cours sur une machine**
  - Ces flôts sont concurrents
  - Ils sont aussi appelés tâches, processus, threads, ..
  - Ils mettent en oeuvre une fonction applicative
  - Ils peuvent partager des données ou se coordonner
- **Pour que tout se passe bien, certaines contraintes doivent être respectées**
  - Les données partagées doivent rester cohérentes
  - Les flôts doivent avoir un temps d'exécution et de réponse corrects
- **Tout cela repose sur des mécanismes fournis par le système sous-jacent**

Objectifs du cours

- **Maîtriser les bases de la programmation concurrente**
  - Connaître les mécanismes (concepts et outils) fournis par le système
  - Savoir les utiliser avec rigueur et efficacité
- **Contexte technique**
  - C / Java
- **Portée du cours**
  - Concurrence locale (flôts d'exécution co-localisés sur une machine)

Organisation de l'enseignement

- **Equipe pédagogique**
  - Fabienne Boyer (Fabienne.Boyer@imag.fr, Equipe Erods / LIG)
  - Olivier Gruber (Olivier.Gruber@imag.fr, Equipe Erods / LIG)
- **Fonctionnement**
  - 11 sessions de 3H

    Chaque session comprends

    - un cours à base de transparents

    - un travail encadré à faire (TD/TP)
  - 1 TP en java à faire en dehors des séances (programmation concurrente multi-threadée)
  - 1 examen

Sessions

1. Notion de Processus
2. Notion de Threads
3. Notion d'Exclusion mutuelle
4. Outil de type Moniteurs
5. Outil de type Sémaphores
6. Solutions directes à base de moniteurs
7. Solutions FIFO et à Priorités à base de moniteurs
8. *Projet de programmation concurrente (lancement)*
9. Gestion des Interblocages
10. Notion d'Exécuteurs
11. Wrap-up et soutenances

Bibliographie

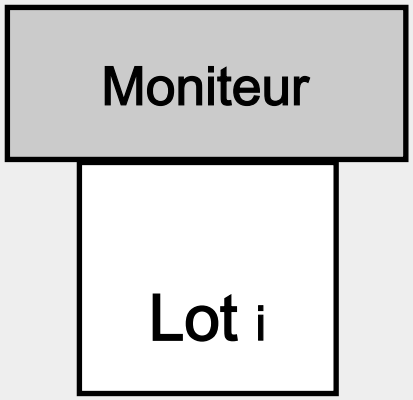
- **Systèmes d'exploitation, fournisseur des concepts et outils de bas niveau pour la programmation concurrente :**
  - Silberschatz, Galvin and Gagne, Operating System Concepts, Addison-Wesley, 2nd edition, 2014
  - A. Tanenbaum, Modern Operating Systems, dec 2015
- **Programmation multi-threadée (Java context)**
  - Java Concurrency in Practice

Modèles d'exécution concurrente

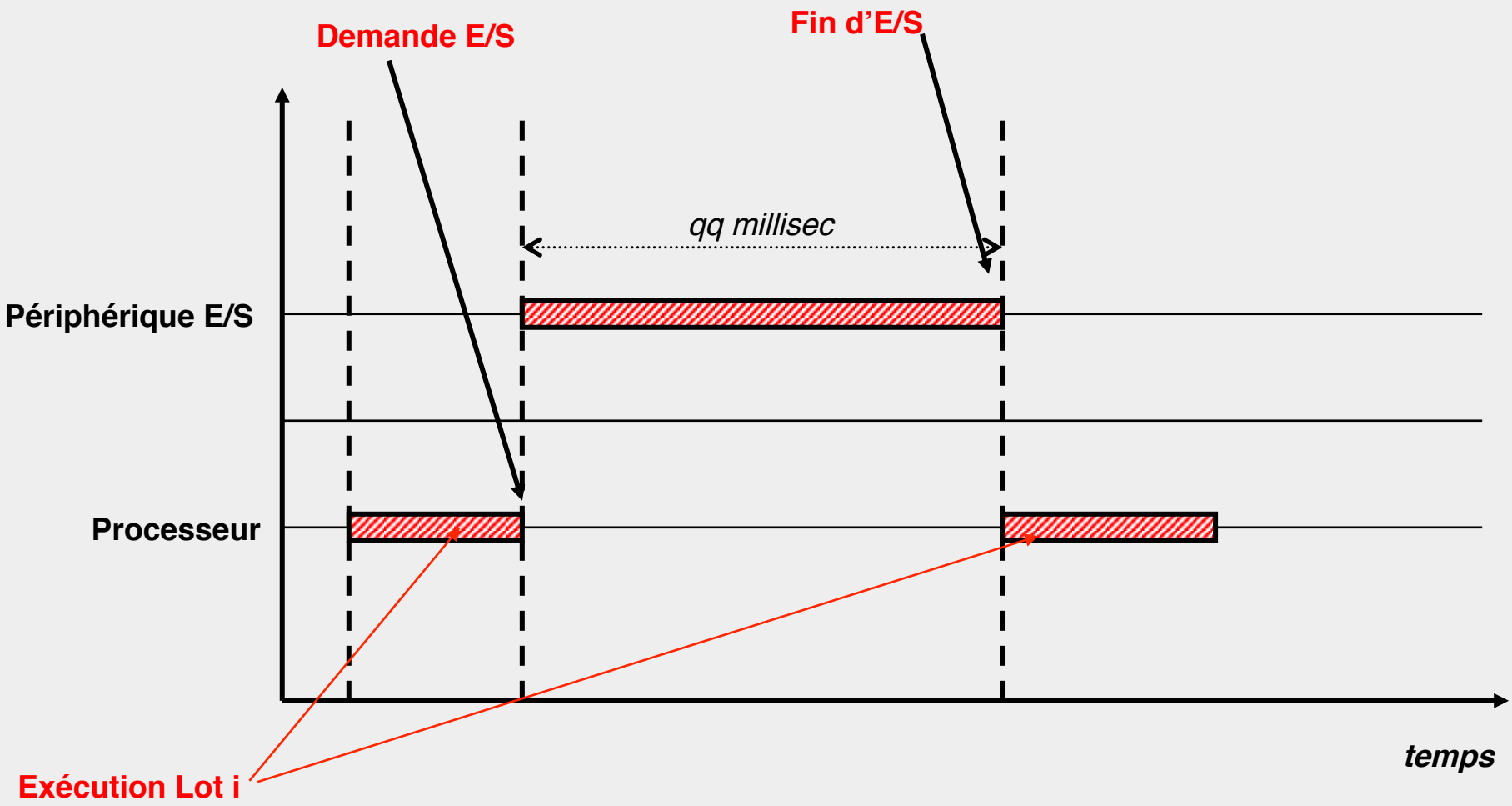
- **3 modèles de base fournis par les systèmes d'exploitation (hypothèse mono-processeur)**
  - Mono-programmation
  - Multi-programmation
  - Temps-partagé

Premières générations de systèmes: mono-programmés

- **Un programme à la fois..**
- **Ordinateurs "mainframes"**
  - Traitement par lots (cartes perforées)
  - L'enchainement des lots est décrit par une carte perforée spéciale (carte de controle)
  - Le SE se limite à un moniteur résident qui enchaine les lots
- **Inconvénients**
  - Lent
  - Peu interactif
  - Pas vraiment de concurrence

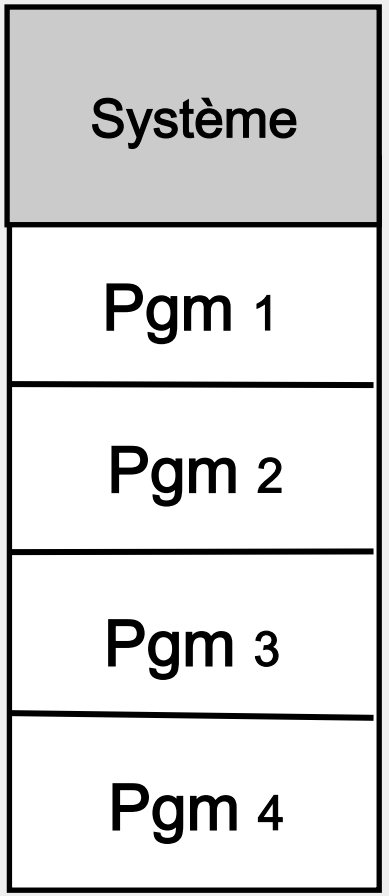


Mono-programmation

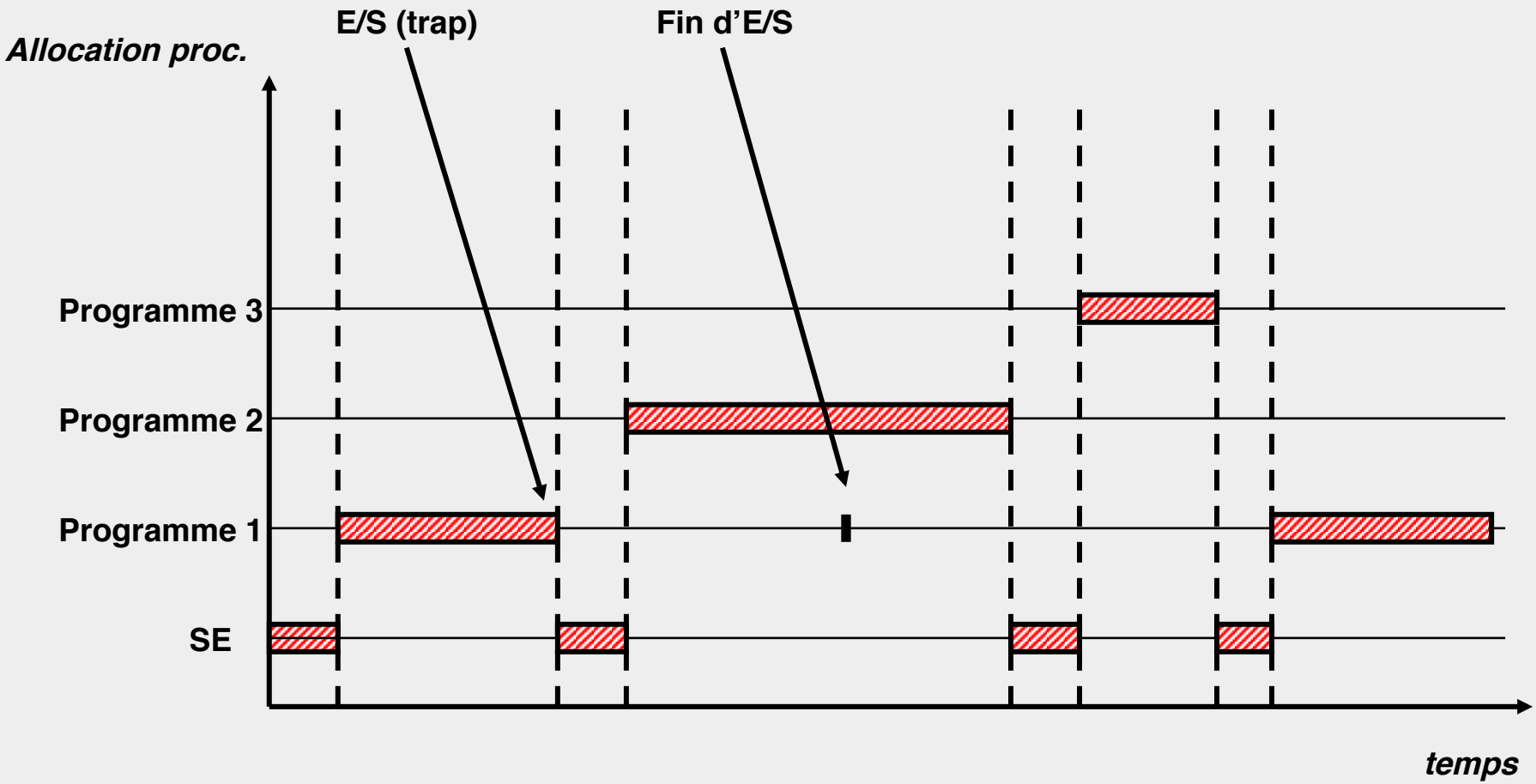


Multi-programmation (1960/1970)

- **Systèmes multi-programmés**
  - Plusieurs programmes en mémoire simultanément
  - Multiplexage du processeur entre les programmes
  - Perte du processeur lors des E/S
- **Avantages**
  - Meilleure utilisation de l'UC
  - On gagne sur les temps d'exécution globaux
- **Inconvénients**
  - Complexité
  - Taille de mémoire grande
  - Partage et protection des ressources



Multi-programmation



Multi-programmation et protection

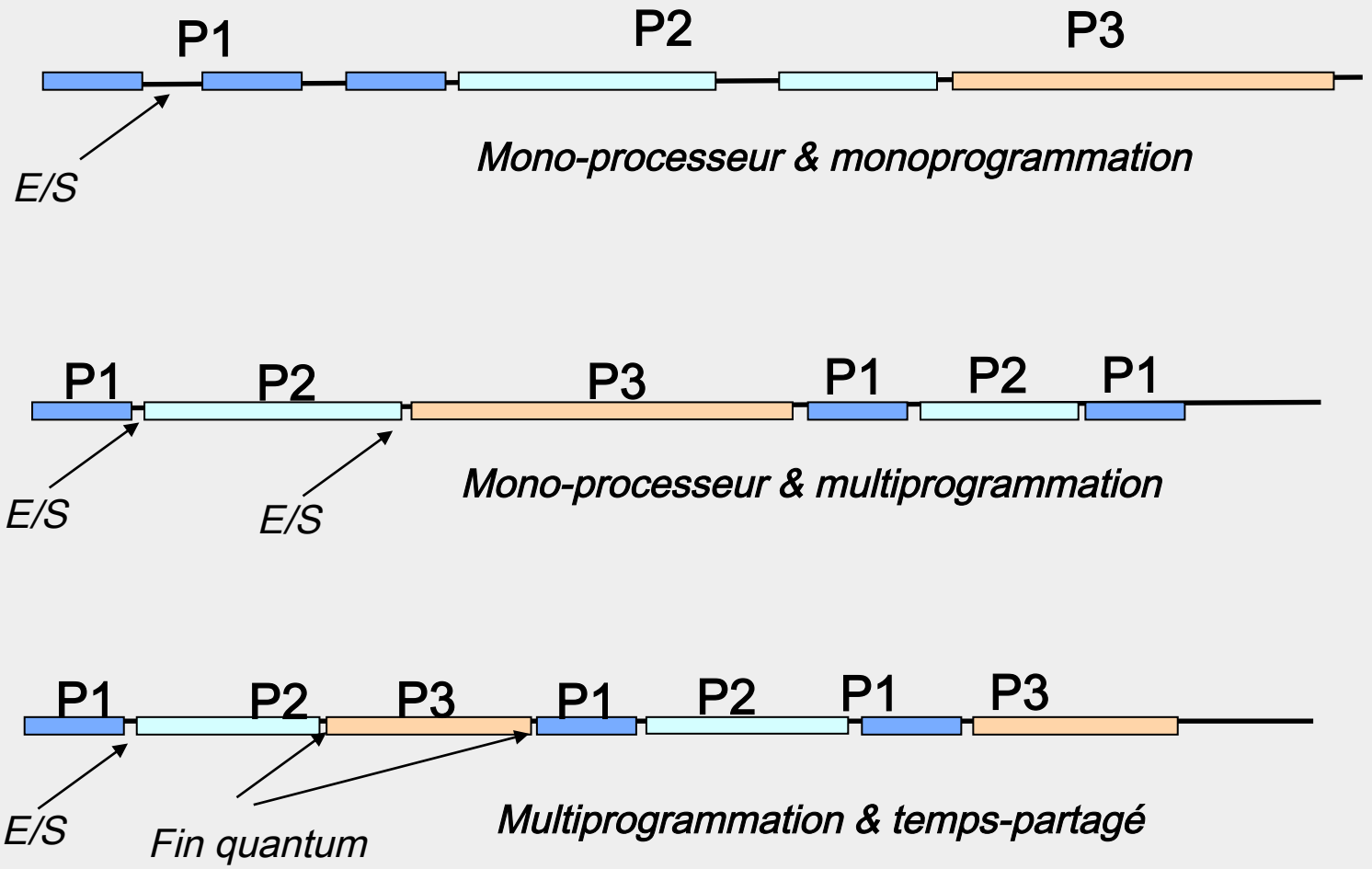
- Eviter qu'un programme en cours d'exécution puisse lire / écrire dans la zone mémoire affectée à un autre programme
- Eviter qu'un programme en cours d'exécution puisse manipuler la zone réservée au système autrement que par les appels système
- Eviter qu'un programme en cours d'exécution puisse lire / écrire dans les buffers d'E/S d'un autre programme

➔ La gestion des ressources (mémoire, E/S, …) devient une tâche complexe pour le système

Temps-partagé (1970)

- **Systèmes à temps partagé**
  - Partage du temps processeur entre les programmes en cours d'exécution (quantum de temps)
  - Processus en mémoire ou "swappé" sur disque
    - Plus grand nombre de programmes en cours
    - Une mémoire plus grande pour chaque programme en cours
- **Avantages**
  - Temps de réponse corrects pour programmes courts, même en présence de programmes longs et non interactifs
- **Inconvénients**
  - Complexité
  - L'utilisation du processeur peut être moins bonne

Systèmes mono/multiprogrammés et temps partagé



Notion de processus

- **Concept fourni par le système pour exécuter un programme**
  - On dit qu'un processus correspond à un programme en cours d'exécution
  - Un processus est lancé à la demande du système ou d'un autre processus
  - Il est créé par un processus père (sauf processus initial)
  - Il est identifié de manière unique (*pid*)
  - Il peut être administré (démarré, suspendu, stoppé, ..)
- **Regroupe deux unités**
  - Flôt d'exécution : exécute la suite d'instruction qui compose le programme
  - Espace d'adressage : mémoire manipulable par les instructions exécutées
- **Propriétés d'isolation**
  - Isolation des fautes : une faute dans un processus ne peut affecter un autre
  - Isolation mémoire : un processus ne peut accéder à l'espace d'adressage d'un autre

Echange de données entre processus

- **Au moment de la création**
  - Héritage de l'état du père
- **Via des messages (stream)**
  - Tubes et queues de message
- **Via des fichiers ou via le réseau**

- *Via des segments de mémoire partagée*
  - *Portion partagée de l'espace d'adressage*
  - *Utilisé dans la programmation de niveau système*

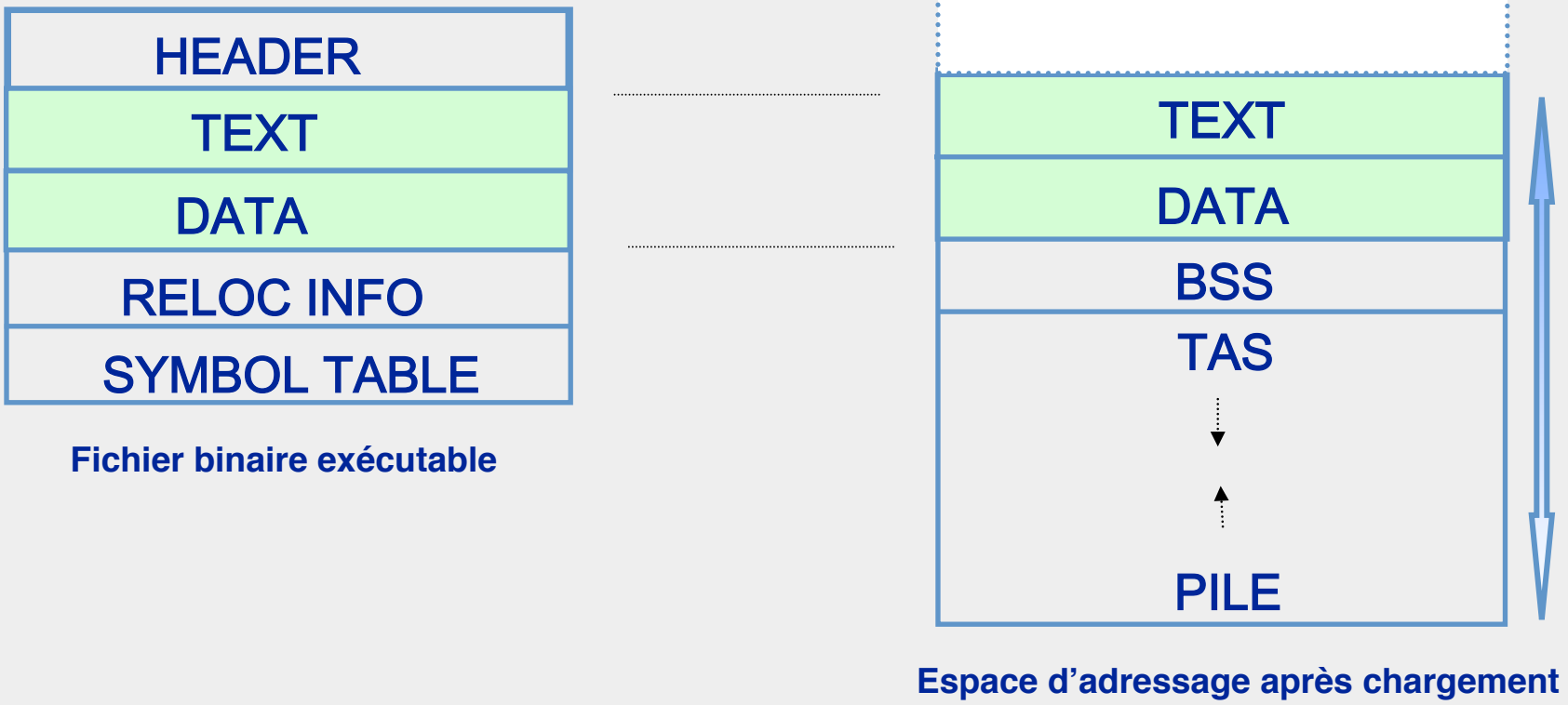
(rappel) Exécution d'un programme

- **Lancement du programme via le shell (cas général)**

  ./mypgm → le shell demande au système de créer un processus pour exécuter le programme mypgm

- **Le système effectue un ensemble d'actions**
  - Allocation d'un espace d'adressage
  - Résolution des liens vers les bibliothèques dynamiques
  - Placement des arguments du *main* sur la pile
  - Initialisation des registres (*considérer que PC pointe sur la fonction main*)
  - Lancement de l'exécution
    - Exécution de la fonction main(..) puis de la fonction exit()

(rappel) Espace d'adressage d'un processus



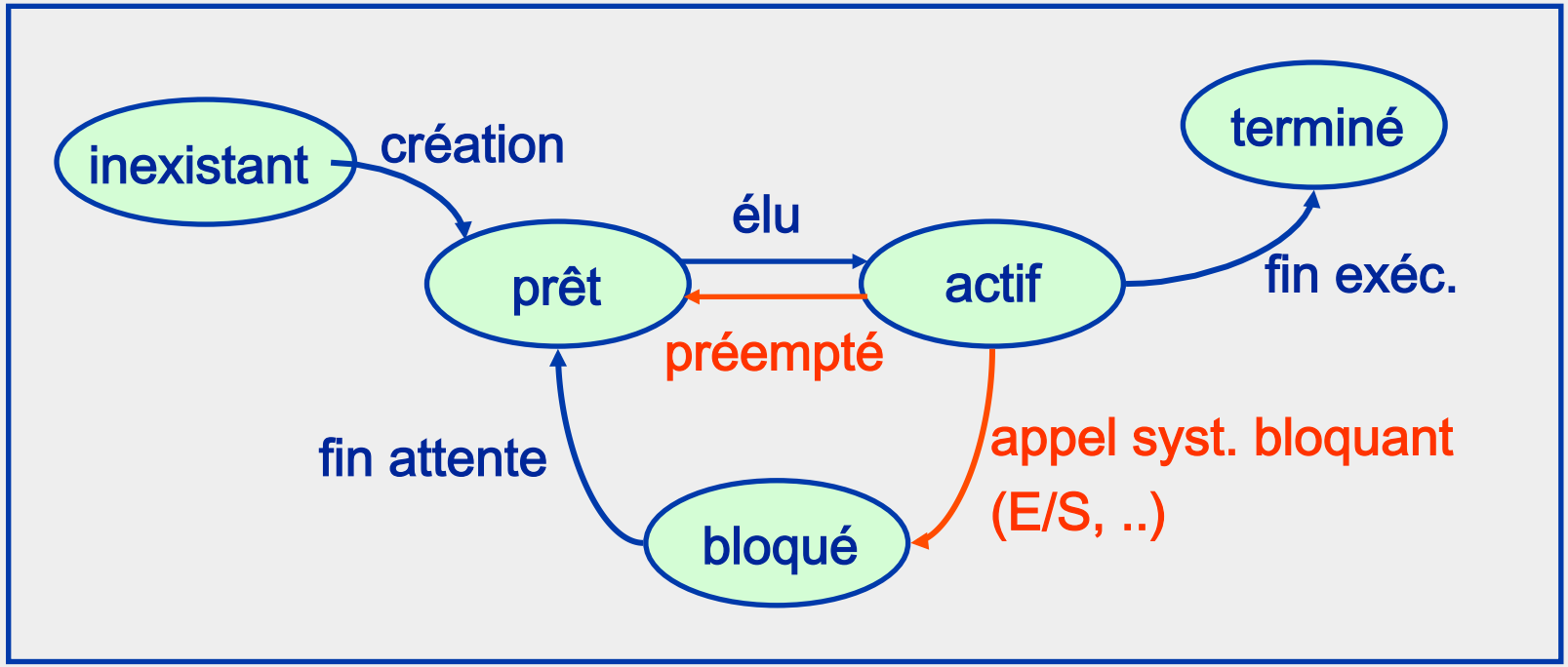
Fichier binaire exécutable

Espace d'adressage après chargement

Modes d'exécution d'un processus

- **Mode utilisateur**
  - Accès réduit à l'espace d'adressage propre au processus
  - Instructions limitées
- **Mode superviseur**
  - Toutes les instructions autorisées et toute la mémoire accessible
  - Superviseur # SuperUser (shell)
- **Passage du mode utilisateur au mode superviseur**
  - Interruptions
  - Déroutements
  - Appels au superviseur

Cycle de vie



remarque : certains systèmes définissent des états supplémentaires (suspendu, …)

Gestion des processus par le Système

- **Les processus sont indexés dans des files**
  - File des prêts (Ready queue)
  - File des bloqués sur E/S (Device queues)
  - File des bloqués sur conditions de synchronisation (Blocked queue)
  - ...
- **Un seul processus actif (hypothèse mono-coeur)**

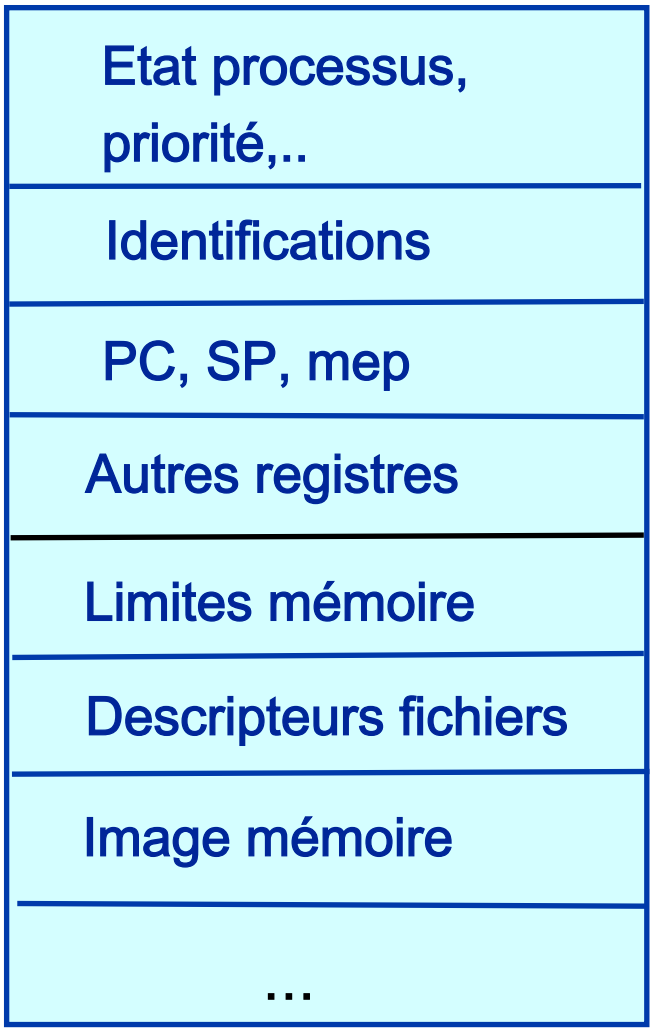
→ **Le système fait migrer les processus entre les files**

→ **Il prend soin de conserver le contexte d'exécution pour chaque processus commuté**

Structure de contrôle des processus

PCB (Process Control Block): informations permettant de gérer un processus

Table des Processus:
PCB [MAX-PROCESSUS]



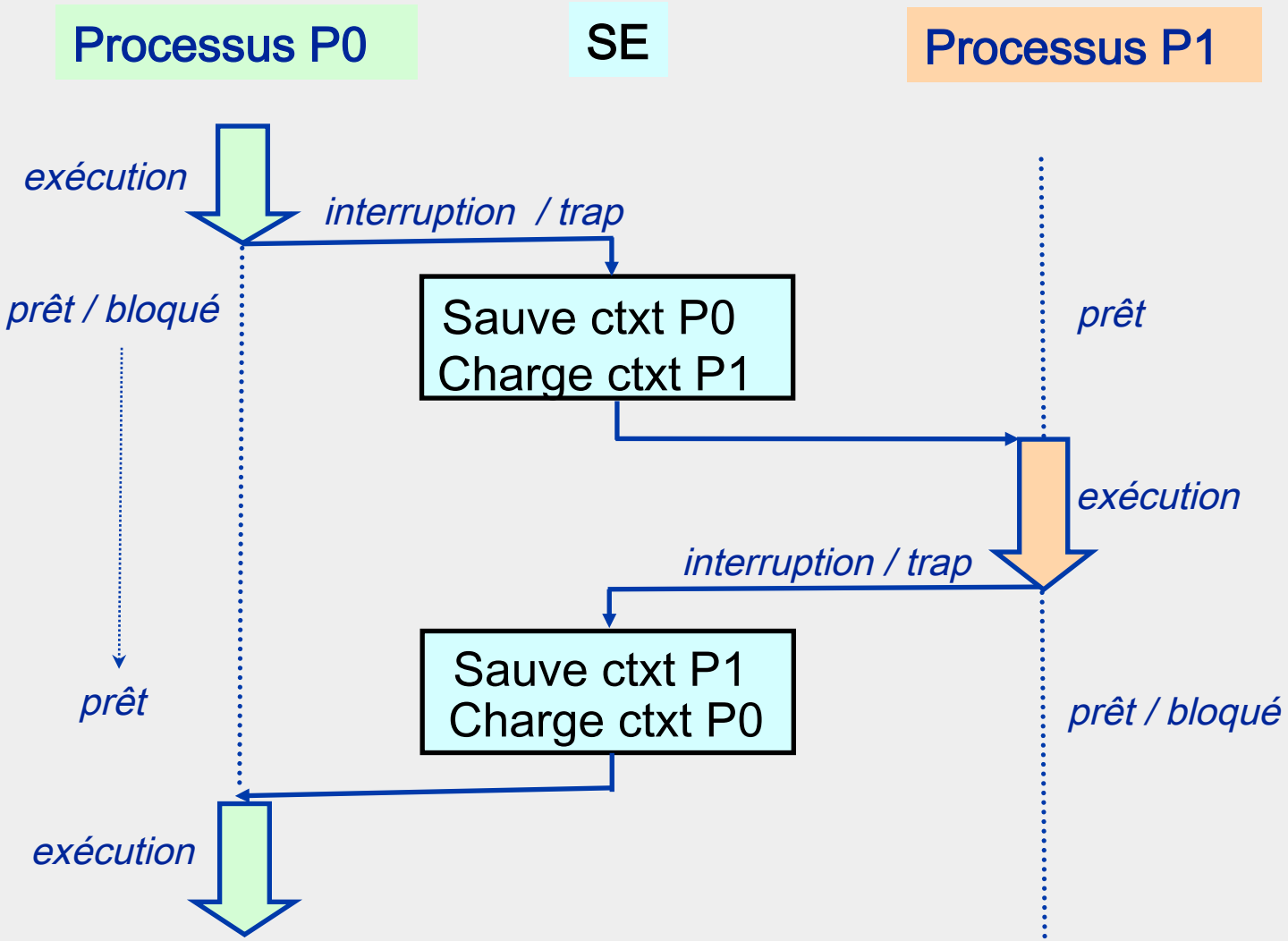
Contexte d'exécution

- **Etat courant du processus**
  - Etat de l'espace d'adressage
    - Code du programme exécuté
    - Données du programme exécuté
  - Etat des ressources utilisées par le processus
    - Registres (PC, SP, etc.,)
    - Liste des fichiers ouverts (descripteurs)
    - Variables d'environnement
    - …

→ **Sauvegardé lorsque le processus est commuté**

→ **Restauré lorsque le processus reprend la main**

Commutation de processus



Opérations de base sur les processus

- **"API processus"**
  - Création / destruction
  - Activation / suspension
  - Suspension momentanée (sleep)
  - Attente de la terminaison d'un fils
  - …
  - *+ Commutation (opération de bas niveau)*

Création d'un processus

- **Allocation PID**
- **Allocation image mémoire (clone de l'image mémoire du père)**
- **Allocation + initialisation PCB**
- **Si préemption du père pour le fils**
  - Arrête le père et sauvegarde son contexte dans son PCB
  - Etat du père ← prêt
  - Ajoute le père dans la File des prêts
  - Etat du fils ← actif
  - Donne le processeur au fils (commutation)
- **Sinon**
  - Etat du fils ← prêt
  - Ajoute le fils dans la File des prêts
- **Renvoie le PID du processus créé**

Exercices

- **Exercice**
  - Décrire les étapes principales pour les opérations suivantes
    - Destruction
    - Activation / Suspension
    - Suspension momentanée (sleep)
- **Objet de l'exercice**
  - Compréhension de la notion de processus
  - Compréhension du cycle de vie associé aux processus

Allocation du processeur au processus

- **L'ordonnanceur (scheduler) est la partie du système qui gère l'allocation du processeur**

- **Critères**
  - Équitable entre les processus
  - Efficace (usage optimal du temps processeur)
    - Ex: si les processus passent leur temps à commuter, l'usage du temps processeur n'est pas bon
  - Minimisant les temps de réponse des processus
    - Ex: si les processus ne commutent jamais, les temps de réponse seront mauvais
  - Maximisant le rendement (nombre de processus qui progressent par unité de temps)

Leviers

- **Choix du quantum**
  - Petit ou grand
- **Gestion de priorités**
  - En général, processus systèmes >> processus utilisateurs interactifs >> processus utilisateurs peu interactifs

Algorithmes classiques d'ordonnancement

- **Multiprogrammation**
  - FIFO / FCFS
  - PCTE / SJF (Plus Court Temps Exécution/Shortest Job First)

- **Multiprogrammation + temps partagé**
  - PCTE / SJF préemptif
  - Tourniquet / Round Robin
  - CFS (linux)
  - ..

First-Come, First-Served (FIFO)

Processus	Temps d'exécution
P_1	24
P_2	3
P_3	3

- Supposons que les processus arrivent dans l'ordre : P_1 , P_2 , P_3

- Temps de réponse de P_1 = 24; P_2 = 27; P_3 = 30
- Temps moyen : (24 + 27 + 30)/3 = 27

Extrait et traduit de Sylberschatz

First-Come, First-Served

Supposons que les processus arrivent dans l'ordre : P_2 , P_3 , P_1 .

- **Temps de réponse de $P_1 = 30$; $P_2 = 3$, $P_3 = 6$**
- **Temps moyen : (30 + 3 + 6)/3 = 13**
- **Meilleur que le cas précédent**

➔ ***Traiter les processus courts avant les longs***

Extrait et traduit de Sylberschatz

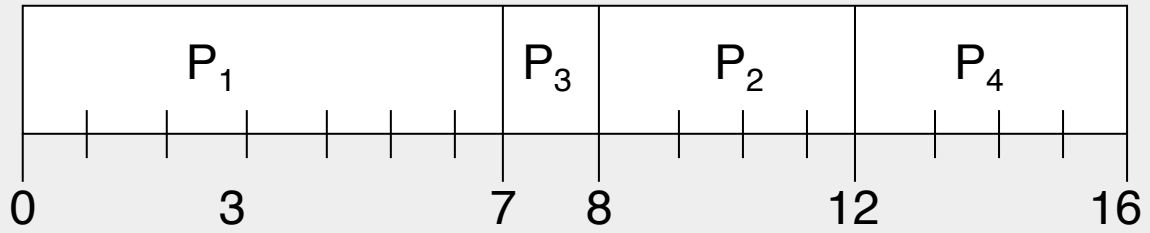
Shortest-Job-First (SJF)

- **Associe à chaque processus la durée de son exécution**
- **Exécute en priorité le processus dont la durée est la plus courte**
- **Deux possibilités**
  - Non préemptif
  - Préemptif (algorithme Shortest-Remaining-Time-First (SRTF))

Non-Preemptive SJF

Process	Arrivée	Temps d'exéc.
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (non-preemptif)

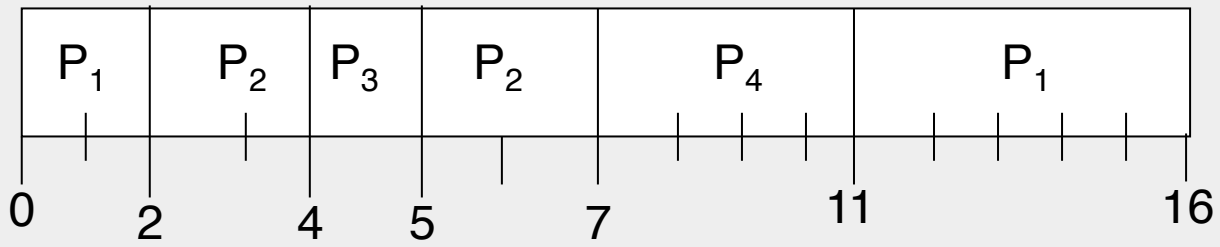


- Temps de réponse moyen = (7 + 8 + 12 + 16)/4 = 10,75

SJF Préemptif

Processus	Arrivée	Temps d'exec.
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (preemptif)



- Temps de réponse moyen = (16 + 7 + 5 + 11)/4 = 8,25

Round Robin

- **Principes**
  - Les processus prêts sont dans une file de type FIFO
  - Lors d'une commutation, le processus le suivant dans la file est élu (*getFirst*())
  - Le processus préempté s'insère en fin de file (*putLast*())
- **Le choix du quantum est l'un des points critiques du Round Robin**

Round Robin avec priorités statiques

- **Principes**
  - Les processus ont des priorités
  - Une priorité est associée à un nombre de quantum (1, 2, 4, ..)
  - Plus la priorité est forte, plus le quantum est petit
  - Le processus le plus prioritaire est élu
- **Evaluation**
  - Bon compromis efficacité / rendement
  - Temps de réponse bons (travaux interactifs prioritaires)
  - Famine possible (les processus de faible priorité peuvent ne jamais s'exécuter)
    - *Aging Solution* ≡ augmenter la priorité d'un processus avec son âge

    → Système à priorités dynamiques

Round Robin avec priorités dynamiques

- **Principes**
  - Déterminer la nature d'un processus (interactif, calcul)
  - Monter la priorité d'un processus s'il est interactif ou âgé

- **Exemple de mise en oeuvre**
  - Soit P devant effectuer des calculs pendant 100 quanta
  - Le quantum de P est 1 quanta initialement (forte priorité)
  - Au bout de 1 quanta, P est commuté
  - Le quantum de P passe à 2 quanta (baisse de priorité)
  - Au bout de 2 quanta, P est commuté
  - Le quantum de P passe à 4 quanta (baisse de priorité)
  - Etc

  P se verra successivement assigner les quantums suivants: 1, 2, 4, 8, 16, 32, 64

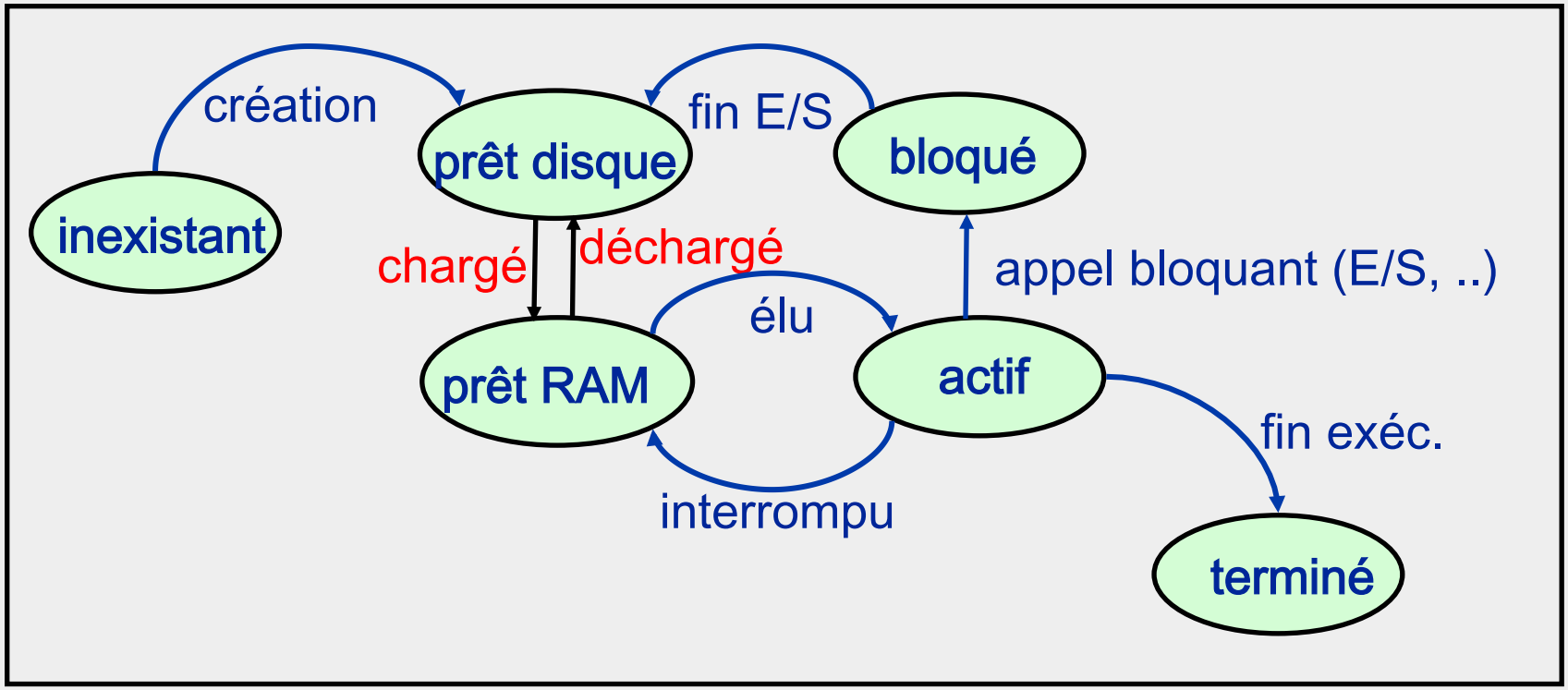
  Il n'épuisera pas son dernier quantum

  Il aura été commuté 7 fois alors qu'avec un round robin statique, il aurait été commuté 100 fois

Algorithmes d'ordonnancement à plusieurs niveaux

- **Ensemble des processus prêts trop important pour tenir en mémoire centrale**
  - Certains sont déchargés sur disque, ce qui rend leur activation plus longue
  - Le processus élu est toujours pris parmi ceux chargés en mémoire
- **En parallèle, on utilise un deuxième algorithme d'ordonnancement pour gérer les déplacement des processus prêts entre le disque et la mémoire centrale**

Cycle de vie avec deux niveaux d'ordonnancement



RAM = Random Access Memory

Notion de Processus – points clés

- **Programme en cours d'exécution**
  - Commuté / multi-programmation & temps partagé
  - Sauvegarde/Restoration de contexte (registres, mep, fichiers ouverts, variables d'environnement, ..) lors des commutations

- **Tout lancement de programme donne lieu à la création d'un processus**
  - `ex: > gcc .. , > java .., > xterm, > firefox, …`

- **L'usage de processus permet d'obtenir**
  - Du (pseudo) parallélisme
  - De l'isolation mémoire et fautes

Notion de Processus – points clés

- **Un processus peut faire des appels système (*fork()*) pour créer des processus fils**
  - Ces processus effectuent des traitements parallèles en étant isolés (fautes, mémoire)
  - Si l'on veut faire des traitements parallèles sans isolation, alors on utilise les threads

Processus vs Containers

- **Les processus peuvent être déployés dans des containers**
  - Typiquement, une application A (potentiellement multi-processus) déployée dans un container
- **Les containers fournissent un niveau d'isolation supplémentaire**
  - Isolation du système de fichier
  - Permet à l'application A de disposer de son propre environnement et librairies (/etc, /val/lib, etc)

Usage des processus : exemple du Shell

- **Un processus qui crée d'autres processus pour exécuter des commandes (sauf pour les commandes *built-in)***
  - *Besoin d'isoler les fautes*
- **Les variables d'environnement du shell sont initialisées via des fichiers de configuration**
  - Fichiers typiques : .login, .logout, .cshrc ou .bashrc, ..
  - Variables typiques : HOME, PATH, TERM, PWD, CWD, ..
- **Les variables d'environnement peuvent être modifiées et d'autres variables peuvent être créées pour le shell courant**
  - *setenv VAR value* (VAR=value en bash)
  - *Ou bien modifier le fichier de configuration et le « resourcer » pour le prendre en compte dans le shell courant (e.g., source ~/.cshrc ou . ~/.cshrc)*
- **Les variables sont héritées par un shell fils**
  - Mais elles sont « écrasées » par la relecture des fichiers de configuration
  - Sauf pour les variables exportées (visibles par les shells secondaires)
  - *Ex dans bash : export VAR*

Exemple du shell

xterm 1

[initialisation automatique des variables d'envt à partir du fichier .bashrc]
➢**env**
USERNAME=paul
PATH=.:/usr/local/bin:/usr/bin:/sbin:/bin
PWD=/home/paul
➢**PATH=$PATH:/usr/lib/jvm/jdk-8/bin**
➢**CLASSPATH=.**
➢**env**
USERNAME=paul
PATH=.:/usr/local/bin:/usr/bin:/sbin:/bin:/usr/lib/jvm/jdk-8/bin
PWD=/home/paul
CLASSPATH=.
>xterm &

.bashrc

USERNAME=paul
PATH=.:/usr/local/bin:……
PWD=/home/paul
…..

xterm 2

[initialisation automatique des variables d'envt à partir du fichier .bashrc]
➢**env**
USERNAME=paul
PATH=.:/usr/local/bin:/usr/bin:/sbin:/bin
PWD=/home/paul

CLASSPATH et PATH ne sont pas hérités dans le shell fils
Il aurait fallu les exporter dans le père pour ce faire