GESTION DES PROCESSUS

SE

Madame Khaoula ElBedoui-Maktouf

2ème année Ingénieur Informatique

Plan

I. Modèle

II. Mise en œuvre

III. Ordonnancement

IV. Etude de Cas

Modèle

1. Notion d'un processus

❖ Processus

- **Processus (Fr)** **Process (En)**
- **Processeur (Fr)** **Processor (En)**

Modèle

1. Notion d'un processus

❖ **Processus**

- **Processus (Fr)** **Process (En)**
- **Processeur (Fr)** **Processor (En)**

C'est un concept de base pour illustrer l'exécution concurrente des tâches dans le cas de la multiprogrammation

Modèle

1. Notion d'un processus

- **Processus**
  - **Programme en cours d'exécution**

Modèle

1. Notion d'un processus

❖ Processus

– **Programme séquentiel en cours d'exécution**

Modèle

1. Notion d'un processus

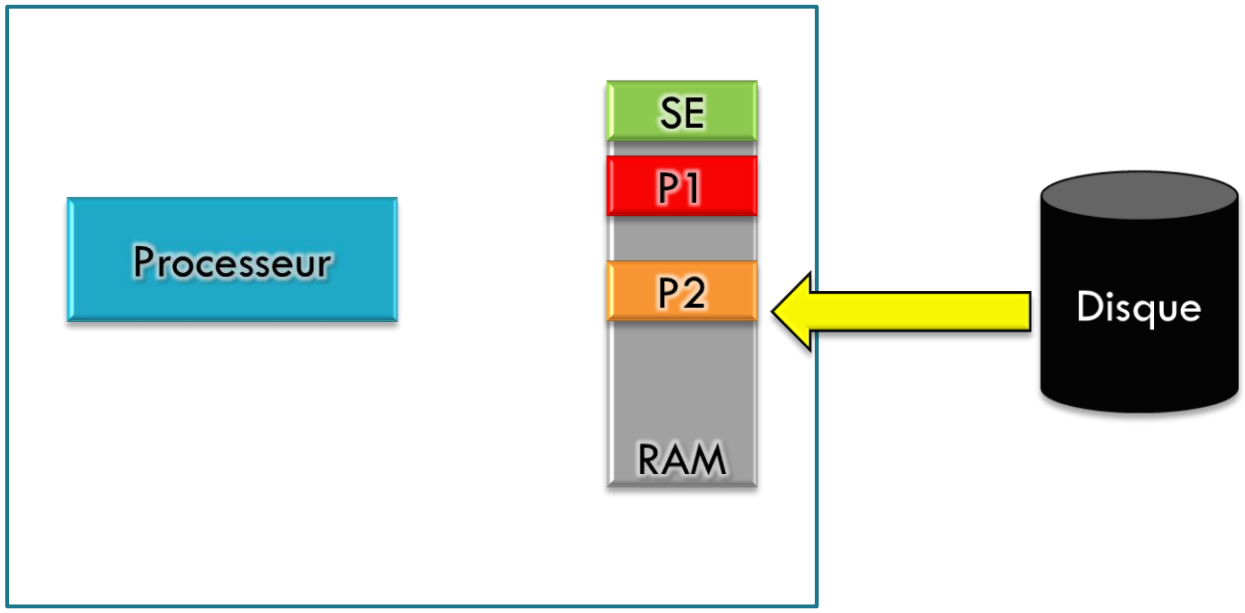
❖ Processus

– **Programme séquentiel en cours d'exécution**

- **Le programme est statique Alors que Le processus est dynamique**
- **L'exécution d'un même programme plusieurs fois engendre des processus différents**

Modèle

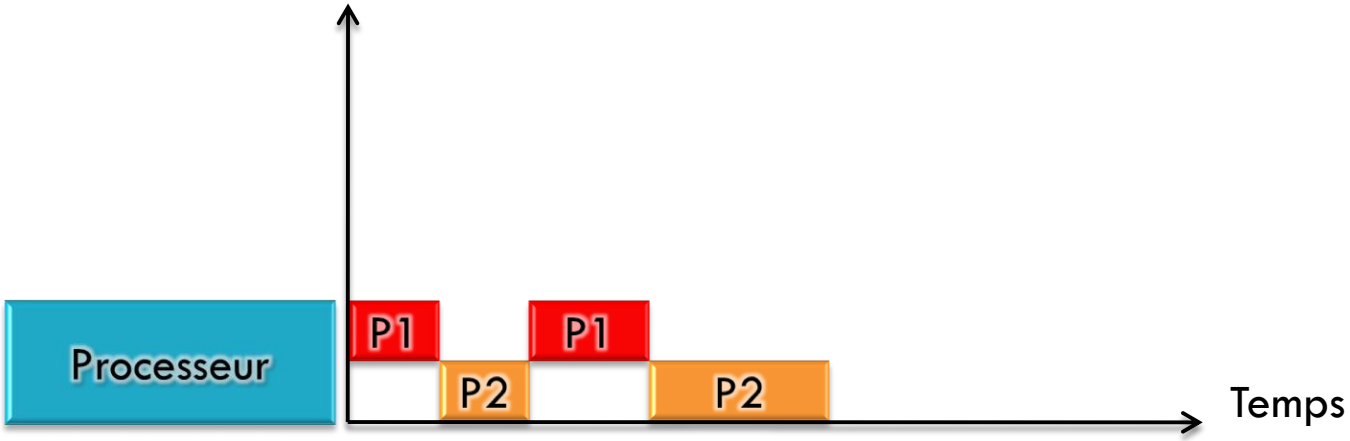
1. Notion d'un processus



Pour pouvoir s'exécuter sur le processeur, le processus doit être chargé en RAM

Modèle

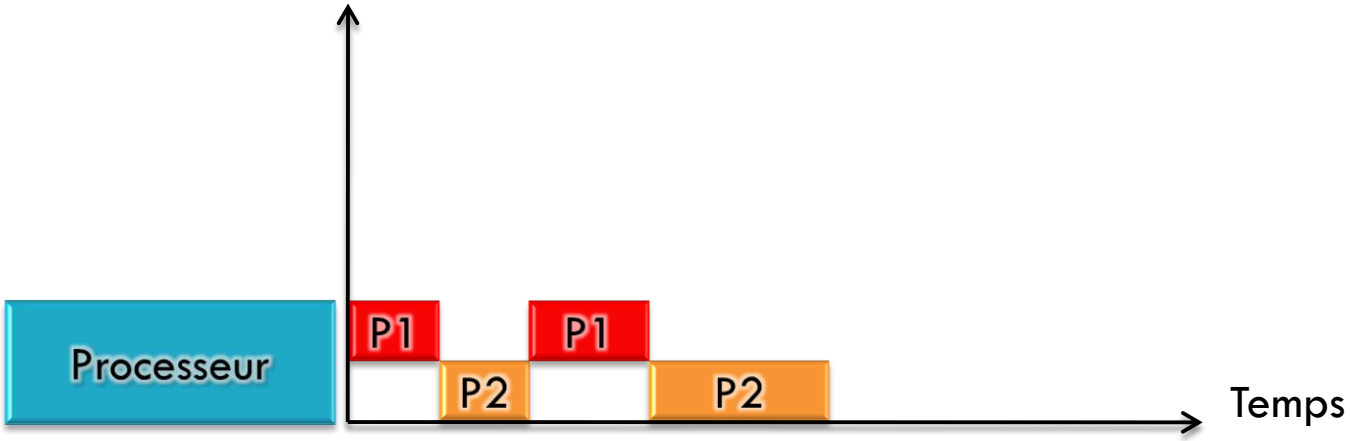
1. Notion d'un processus



Sur un système monoprocesseur, les processus partagent le temps processeurs et évoluent tous au cours du temps

Modèle

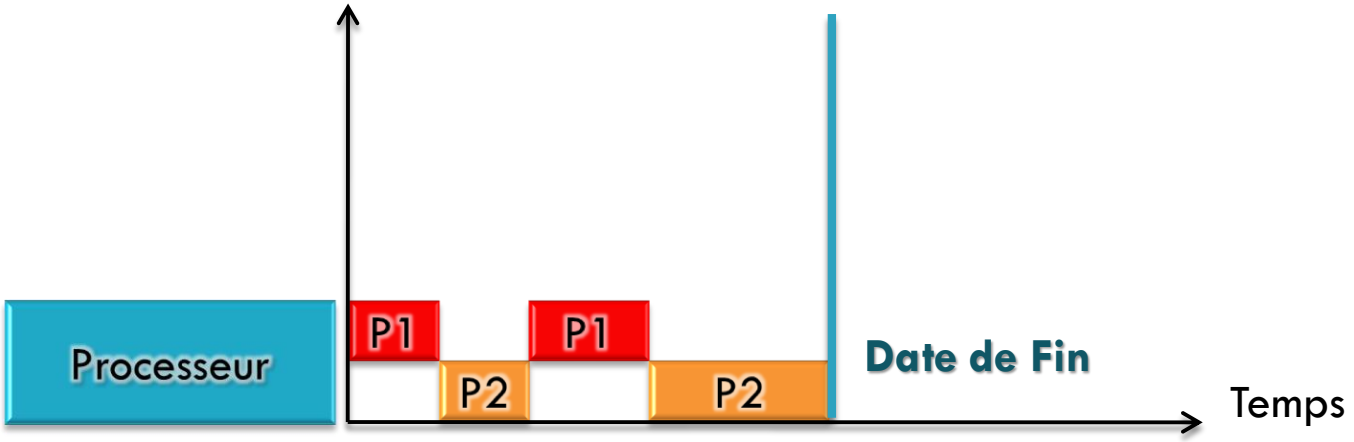
1. Notion d'un processus



- À un instant donné un seul processus tourne
- Sur un intervalle de temps important tous les processus ont fait leurs exécutions

Modèle

1. Notion d'un processus

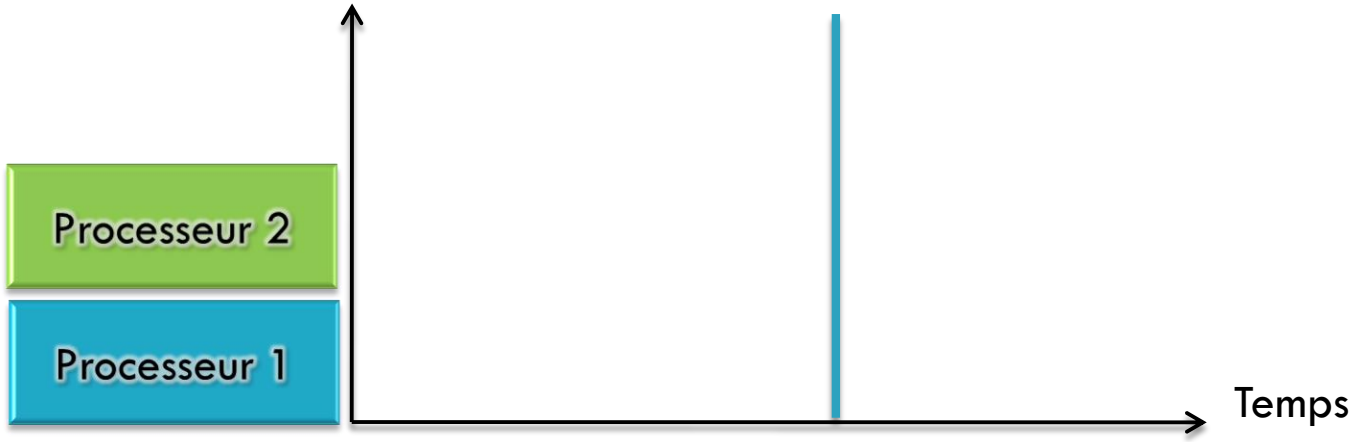


- À un instant donné un seul processus tourne
- Sur un intervalle de temps important tous les processus ont fait leurs exécutions

C'est le **pseudo-parallélisme**

Modèle

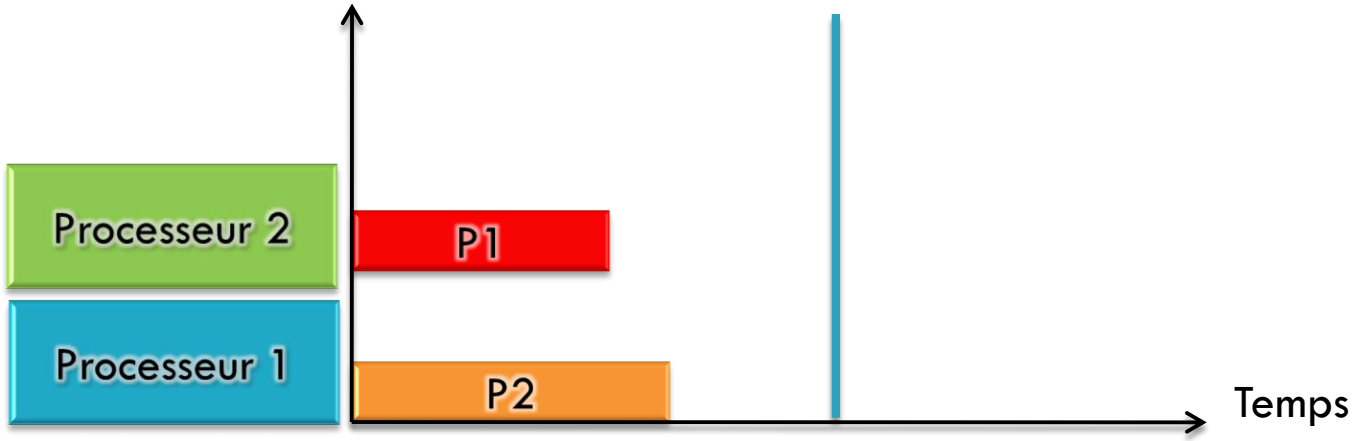
1. Notion d'un processus



Sur un système multiprocesseurs

Modèle

1. Notion d'un processus

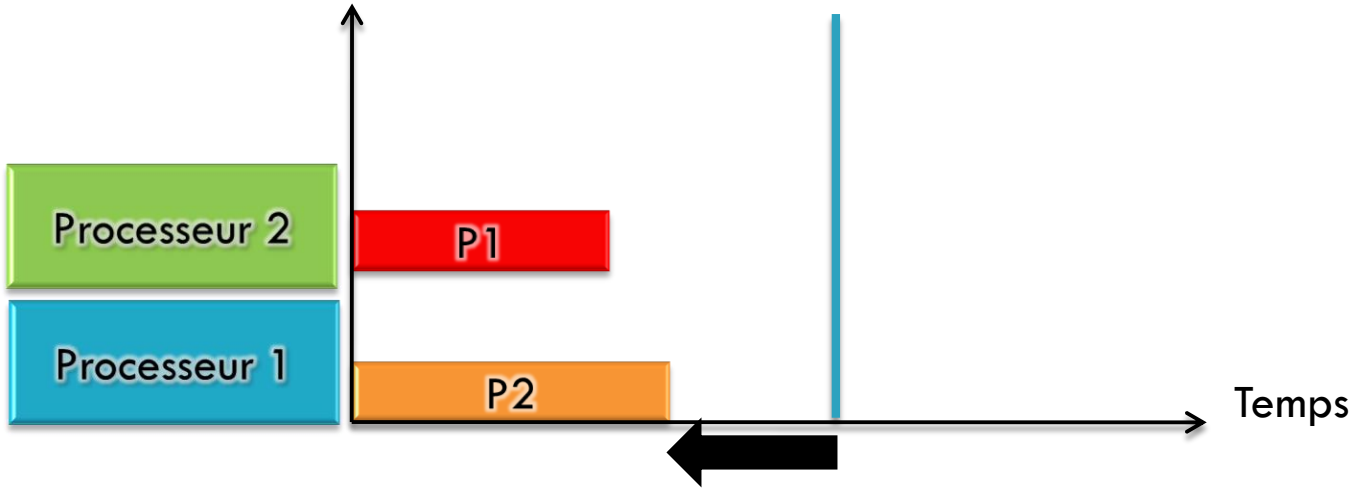


Sur un système multiprocesseurs

C'est le parallélisme

Modèle

1. Notion d'un processus



Minimisation de la date de fin

Sur un système multiprocesseurs

C'est le parallélisme

Modèle

2. Relations entre processus

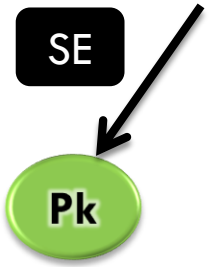
La création d'un processus nécessite un appel système

Modèle

2. Relations entre processus

La création d'un processus nécessite un appel système

(Nécessité de contrôle du SE)

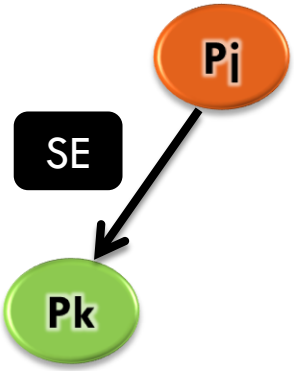


Modèle

2. Relations entre processus

La création d'un processus nécessite un appel système

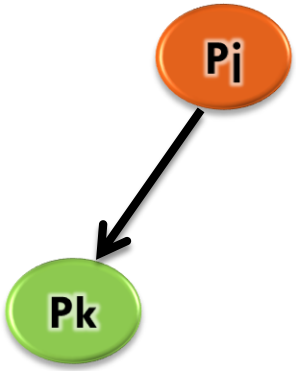
Cet appel est issu d'un autre processus



Modèle

2. Relations entre processus

La création d'un processus nécessite un appel système



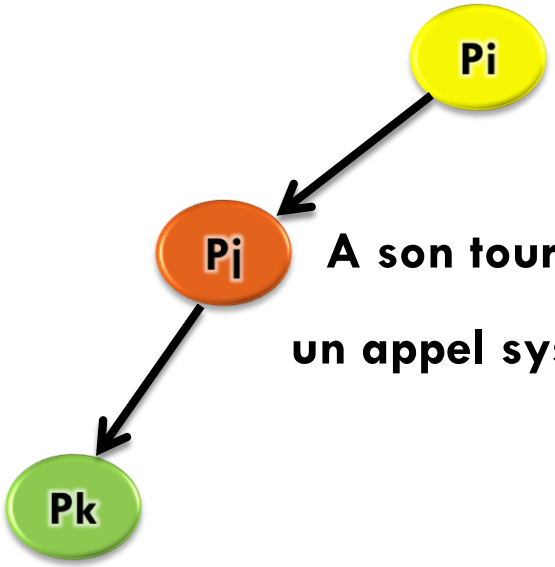
Le processus créateur s'appelle Père (Parent)

Le processus crée s'appelle Fils

Modèle

2. Relations entre processus

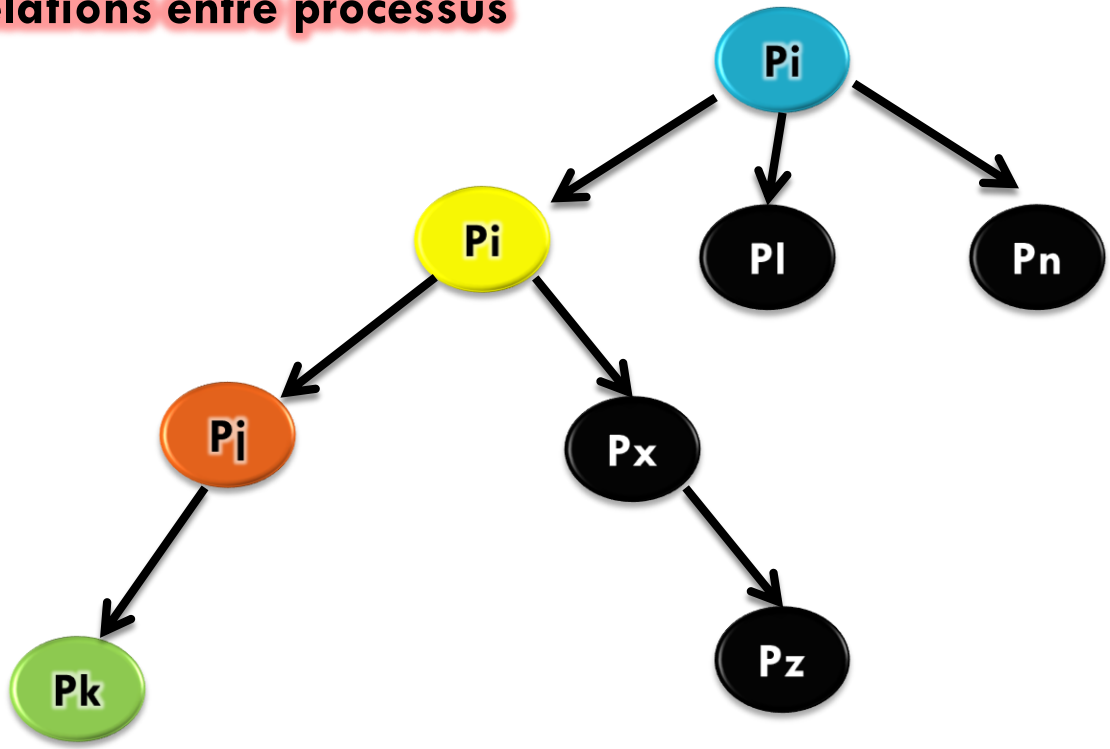
La création d'un processus nécessite un appel système



A son tour, le processus père a été crée suite à un appel système émis par un autre processus parent

Modèle

2. Relations entre processus



Ainsi tous les processus d'un même système sont reliés entre eux par une Arborescence de processus

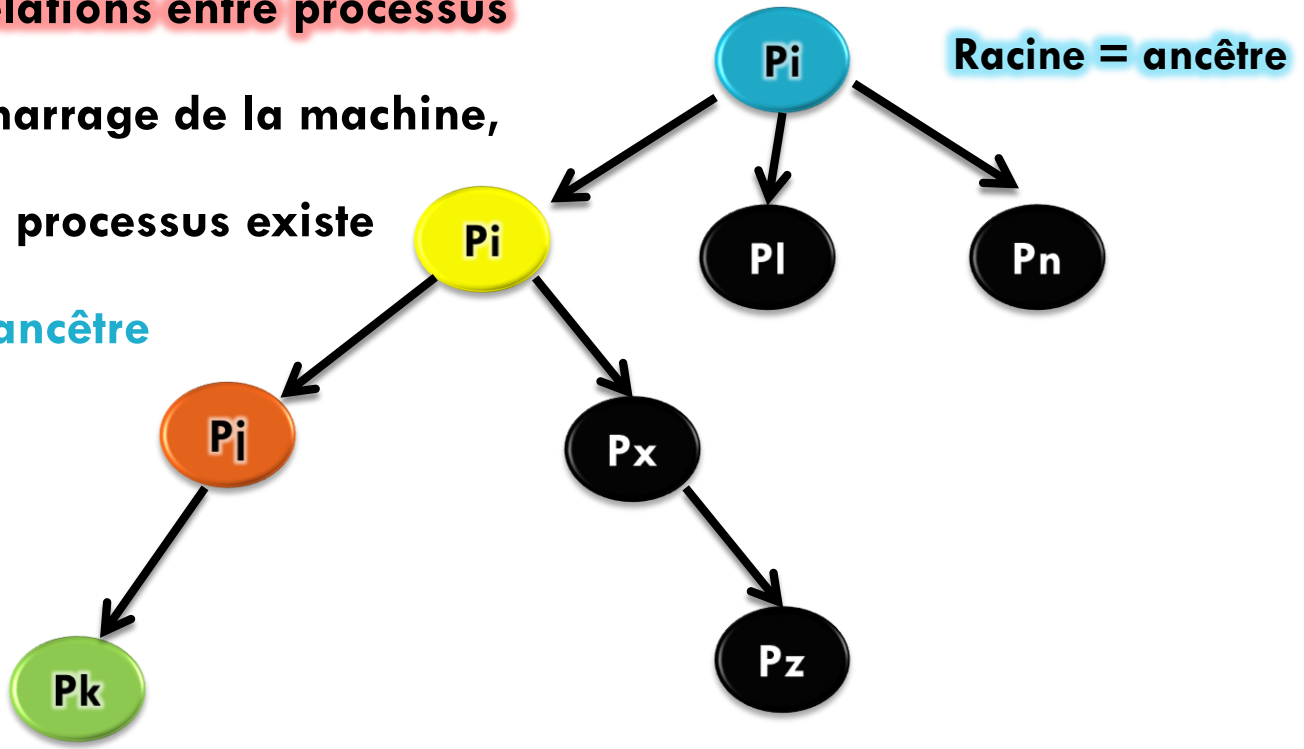
Modèle

2. Relations entre processus

Au démarrage de la machine,

un seul processus existe

C'est l'ancêtre

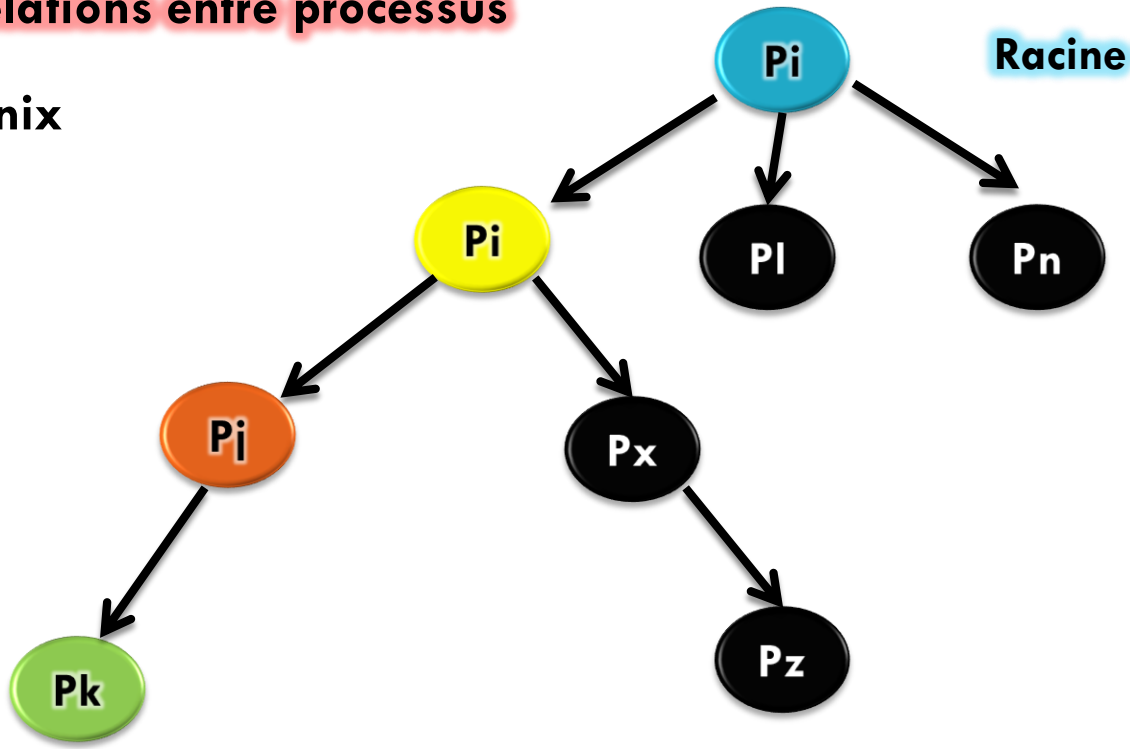


Arborescence de processus

Modèle

2. Relations entre processus

Sous Unix



La commande shell **pstree** permet d'afficher cette arborescence de processus

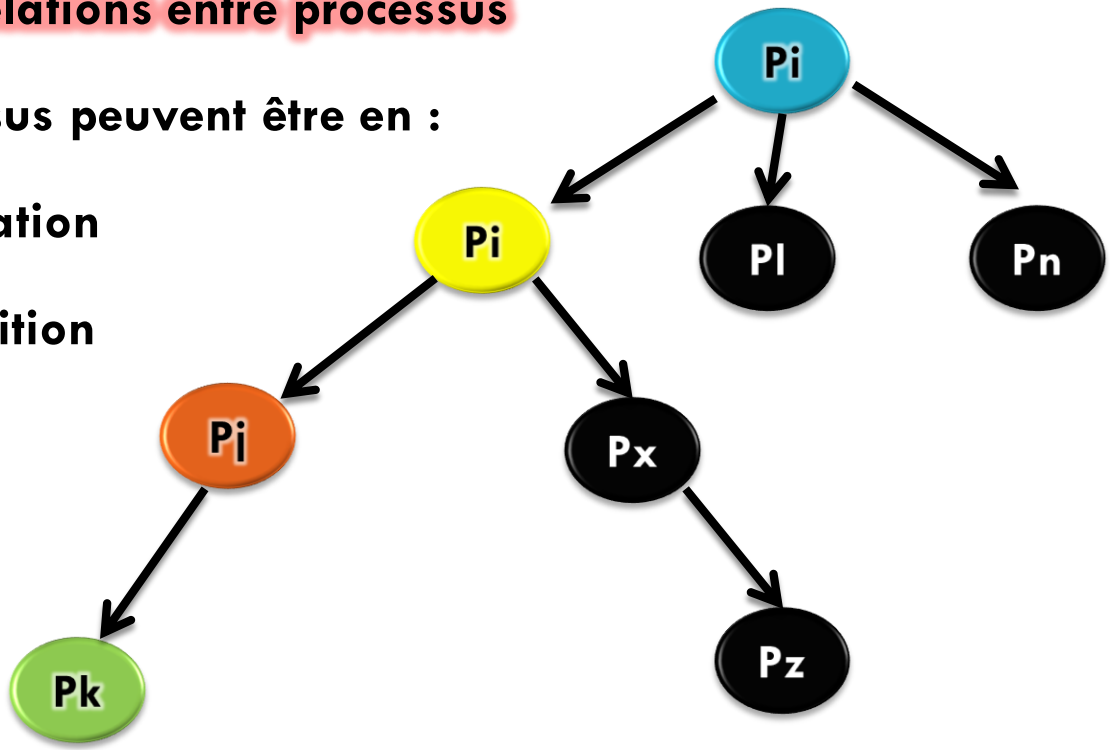
Modèle

2. Relations entre processus

Les processus peuvent être en :

a) Coopération

b) Compétition



Modèle

2. Relations entre processus

Les processus peuvent être en :

a) **Coopération : chacun d'eux réalise une partie du travail**

b) **Compétition : c.a.d en conflit ou en concurrence sur des ressources partagées du système (processeur, mémoire, Fichiers, …)**

Modèle

3. États d'un processus

Durant sa vie, le processus passe par plusieurs états :

Modèle

3. États d'un processus

Durant sa vie, le processus passe par plusieurs états :

- **Actif** : le processus tourne sur le processeur
- **Prêt** : le processus n'attend que la libération du processeur
- **Bloqué** : le processus attend un événement autre que la libération du processeur

Modèle

3. États d'un processus

Durant sa vie, le processus passe par plusieurs états :

- **Actif : Processus OK – Processeur OK**
- **Prêt : Processus OK – Processeur Non**
- **Bloqué : Processus Non – Processeur ∀**

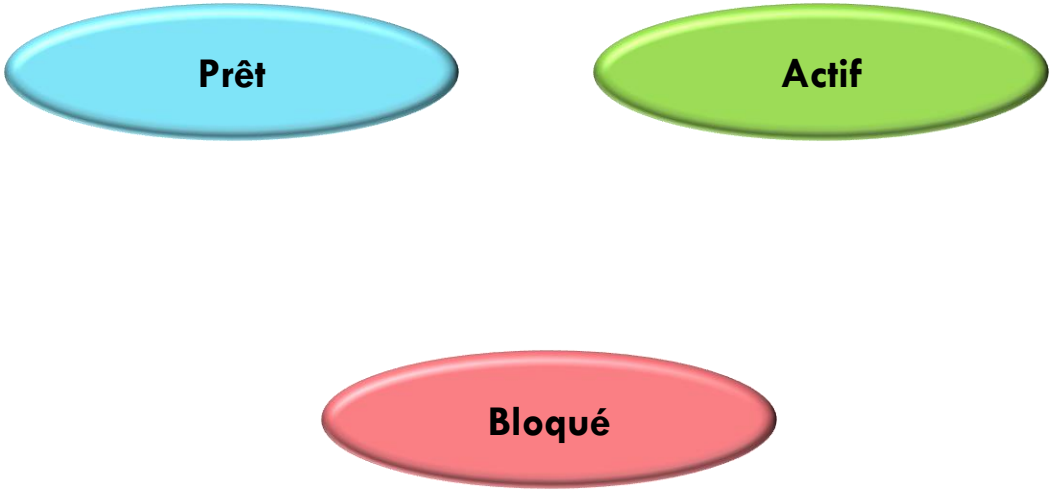
Remarque

Le processus bloqué ne peut pas s'exécuter même si le processeur est libre

Modèle

3. États d'un processus

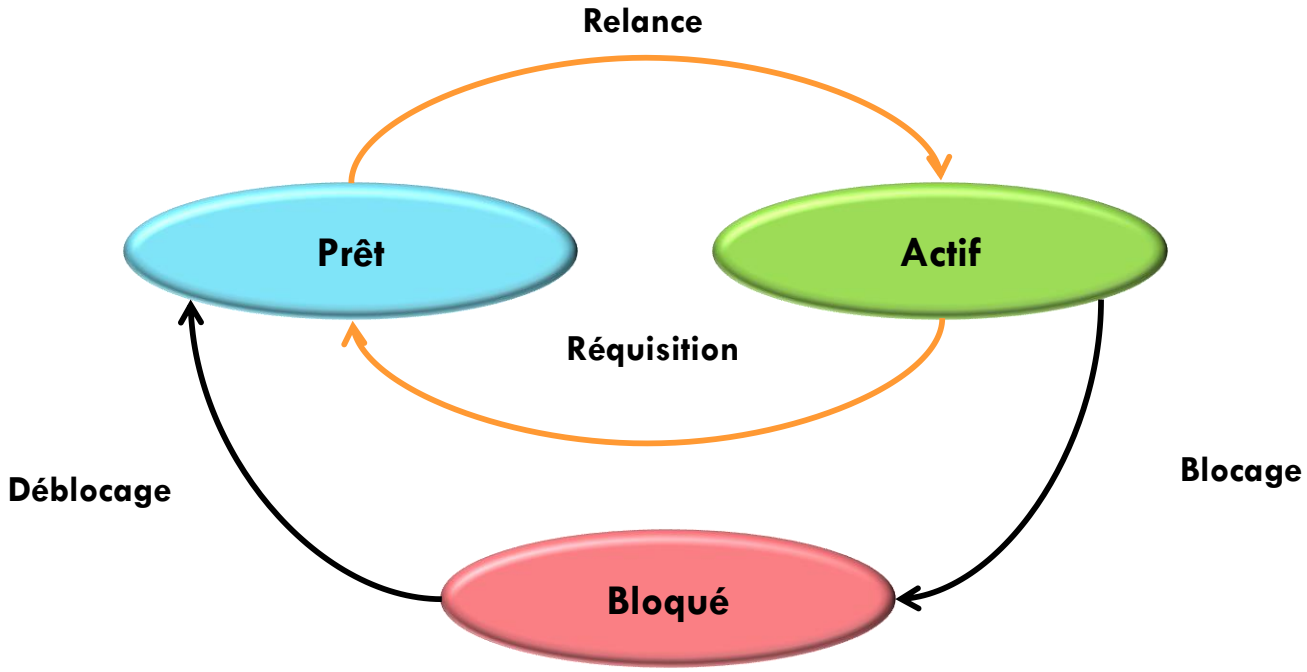
Nous avons donc trois états



Modèle

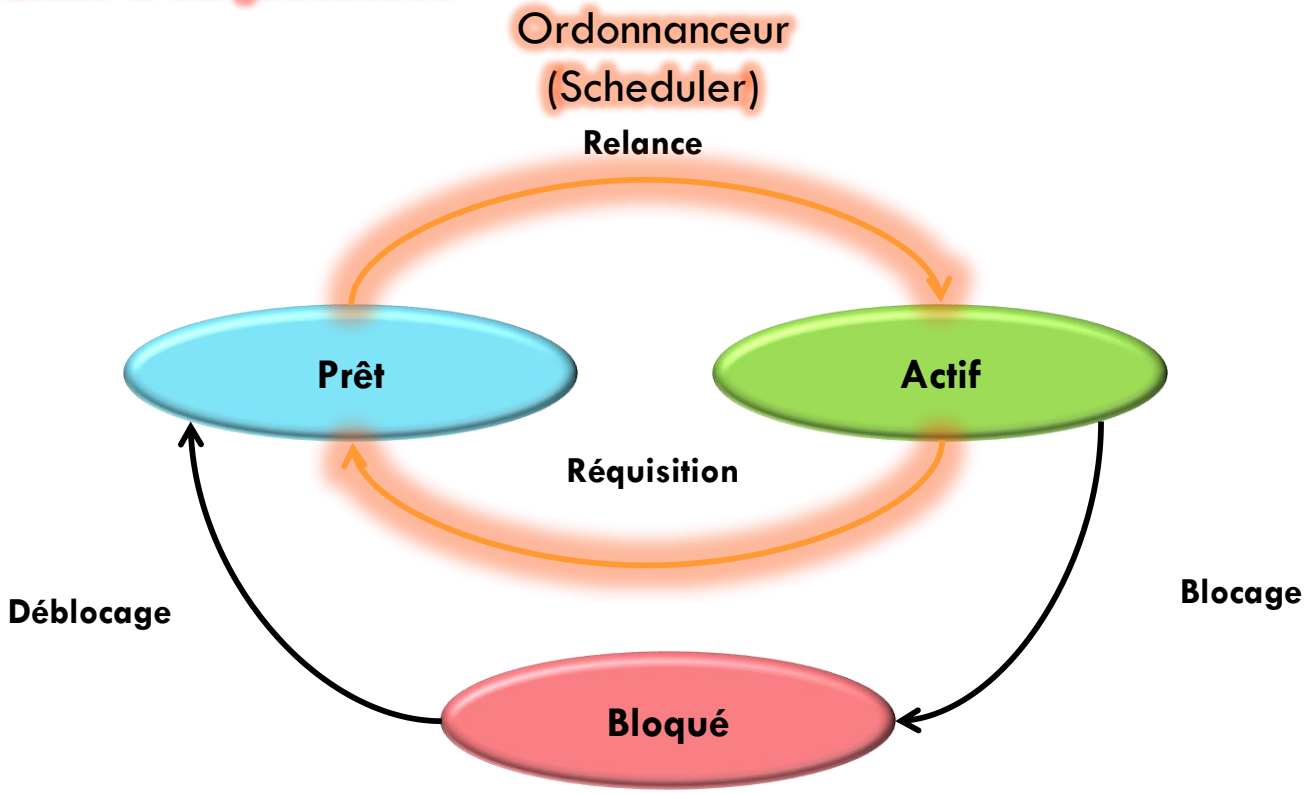
3. États d'un processus

et quatre transitions possibles entre ces états



Modèle

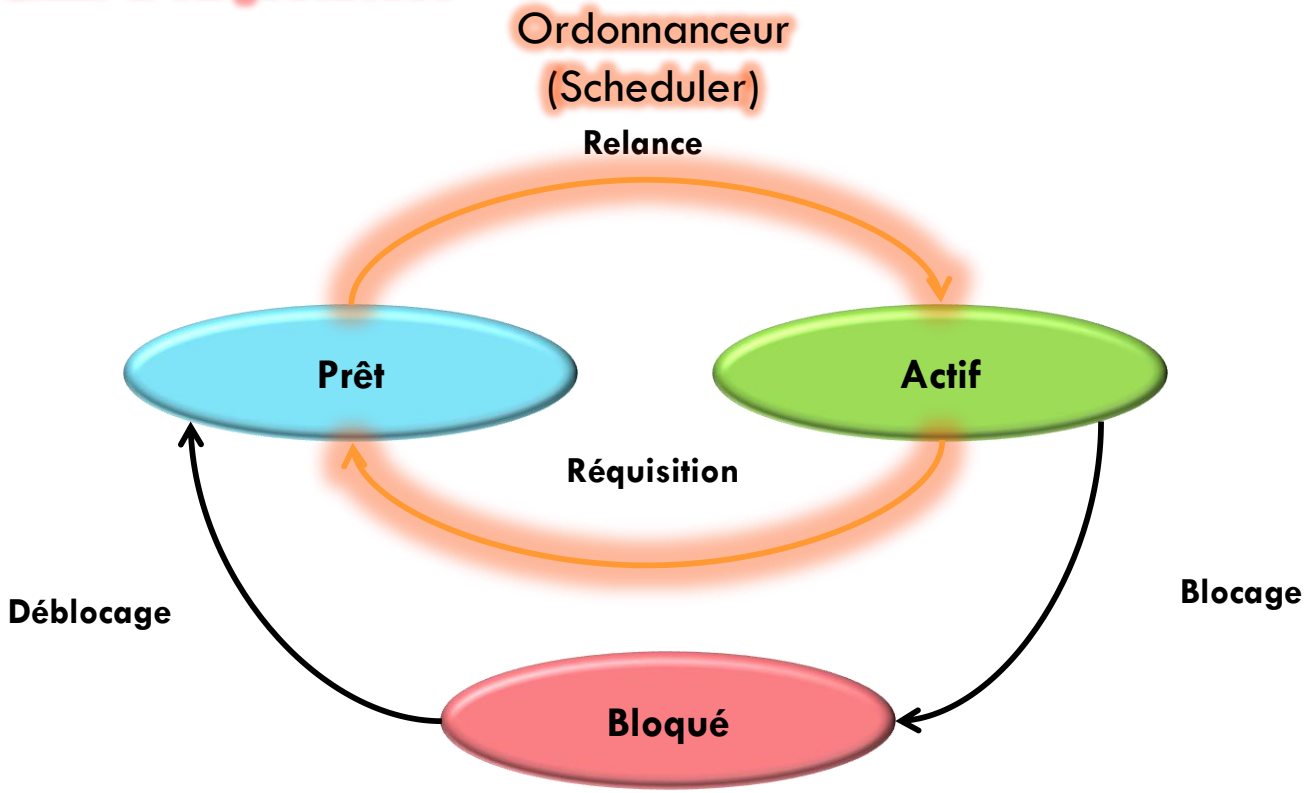
3. États d'un processus



L'ordonnanceur est un module de SE qui assure l'ordonnancement des processus

Modèle

3. États d'un processus



L'ordonnanceur arbitre le passage des processus sur le processeur

Mise en œuvre

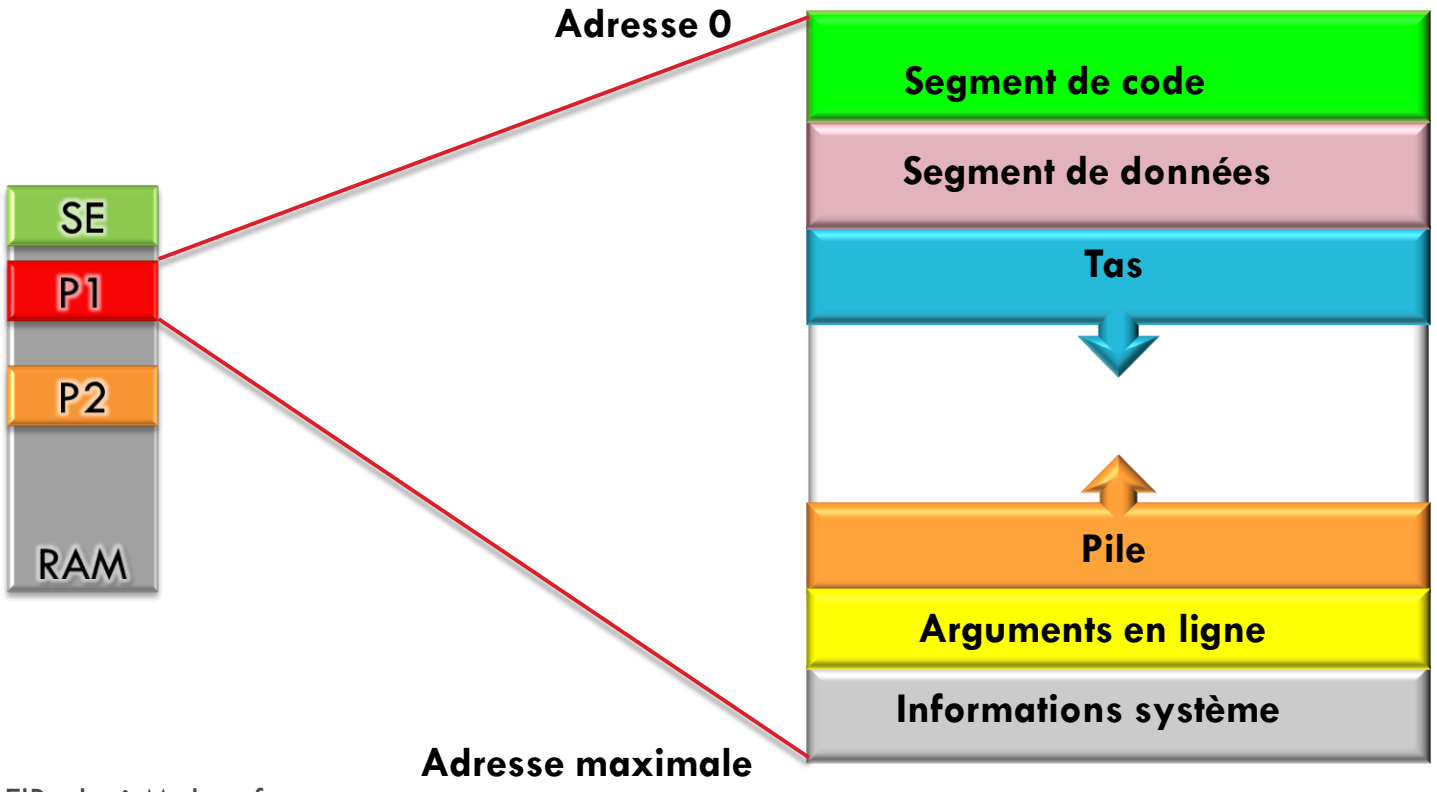
1. Processus

Le processus dispose de son propre espace d'adressage

Mise en œuvre

1. Processus

Le processus dispose de son propre espace d'adressage



Mise en œuvre

1. Processus

Le processus dispose de son propre espace d'adressage

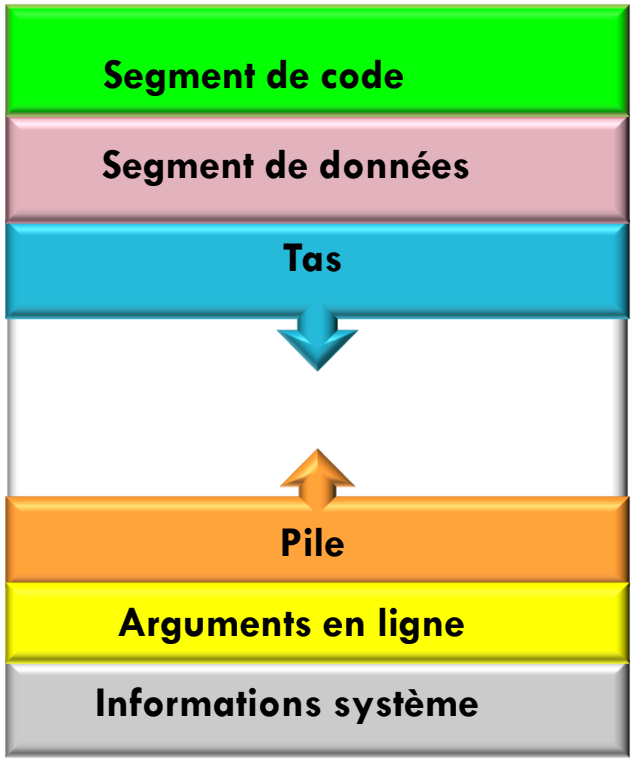
Adresse 0

Les instructions

Les données

Allocations dynamiques (new, malloc)

Appel aux fonctions



Adresse maximale

Mise en œuvre

1. Processus

Le processus est géré via une structure de donnée dite

Mise en œuvre

1. Processus

Le processus est géré via une structure de donnée dite

PCB (Process Control Bloc) qui est une structure de données

Mise en œuvre

1. Processus

Le processus est géré via une structure de donnée dite

PCB (Process Control Bloc) qui est une structure de données

Le PCB contient les informations nécessaires pour la relance d'un processus

Mise en œuvre

1. Processus

Le processus est géré via une structure de donnée dite

PCB (Process Control Bloc) qui est une structure de données

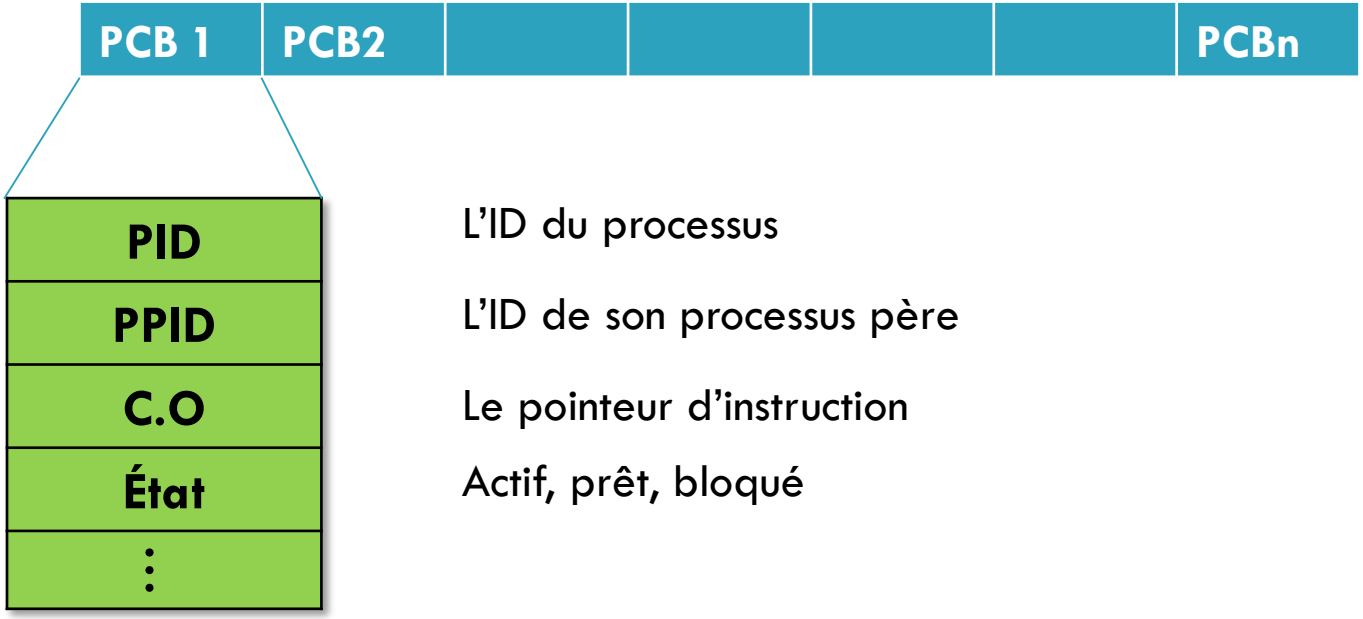
Le PCB contient les informations nécessaires pour la relance d'un processus

Les PCB sont rangés dans une table (dite table des processus) qui se trouve dans l'espace mémoire du système

Mise en œuvre

1. Processus

Table de processus



L'ID du processus

L'ID de son processus père

Le pointeur d'instruction

Actif, prêt, bloqué

Mise en œuvre

2. Commutation de contexte

Le passage d'un processus à autre engendre un changement de contexte qui consomme du temps induit par l'opération atomique :

- **Sauvegarde du contexte actuel**
- **Chargement du nouveau contexte**

Mise en œuvre

2. Commutation de contexte

Le passage d'un processus à autre engendre un changement de contexte qui consomme du temps induit par l'opération atomique :

- **Sauvegarde du contexte actuel PCB du processus actif**
- **Chargement du nouveau contexte PCB d'un autre processus**

Mise en œuvre

2. Commutation de contexte

Le passage d'un processus à autre engendre un changement de contexte qui consomme du temps induit par l'opération atomique :

- **Sauvegarde du contexte actuel**
- **Chargement du nouveau contexte**

La commutation de contexte peut être provoquée par :

a) **Interruption**

b) **Déroutement**

Mise en œuvre

2. Commutation de contexte

a) Interruption (it)

Signal émis par un dispositif externe au processeur.

Le traitement d'une interruption se fait en se référant à un vecteur d'interruptions et en tenant compte de sa priorité

Mise en œuvre

2. Commutation de contexte

a) Interruption (it)

Signal émis par un dispositif externe au processeur.

Le traitement d'une interruption se fait en se référant à un vecteur d'interruptions et en tenant compte de sa priorité

- **Indexé par un numéro de périphérique**
- **Indique pour chaque it l'adresse de la fonction de traitement correspondante**

it	@ Routine
0	@ fonction 0
1	@ fonction 1
2	@ fonction 2
…	…

Mise en œuvre

2. Commutation de contexte

a) Interruption (it)

Étapes de traitement d'une it :

a) **Sauvegarde du processus courant**

b) **Passage en mode noyau**

c) **Identification de l'it**

d) **Chargement de la routine correspondante**

e) **Relance de processus interrompu**

it	@ Routine
0	@ fonction 0
1	@ fonction 1
2	@ fonction 2
…	…

Mise en œuvre

2. Commutation de contexte

a) Interruption (it)

Sous Unix, on distingue les interruptions suivantes :

it	Cause
0	Horloge
1	Disque
2	Terminaux
3	Périphériques
4	Logiciels (exp. trap)
5	Autres

Mise en œuvre

2. Commutation de contexte

b) Déroutement (dt)

Engendré par le processus actif et ce suite à une erreur d'exécution :

- **Débordement d'un tableau**
- **Instruction incorrecte (exp. division par zéro)**
- **Violation d'une protection**
- **…**

Mise en œuvre

2. Commutation de contexte

b) Déroutement (dt)

Engendré par le processus actif et ce suite à une erreur d'exécution :

- **Débordement d'un tableau**
- **Instruction incorrecte (exp. division par zéro)**
- **Violation d'une protection**
- **…**

Le traitement d'un déroutement ne peut pas être différé (pas de mise en attente possible

Mise en œuvre

2. Commutation de contexte

b) Déroutement (dt)

Engendré par le processus actif et ce suite à une erreur d'exécution :

- **Débordement d'un tableau**
- **Instruction incorrecte (exp. division par zéro)**
- **Violation d'une protection**
- **…**

Le traitement d'un déroutement ne peut pas être différé (pas de mise en attente possible

Le traitement d'un déroutement est toujours plus prioritaire que celui d'une interruption

Mise en œuvre

2. Commutation de contexte

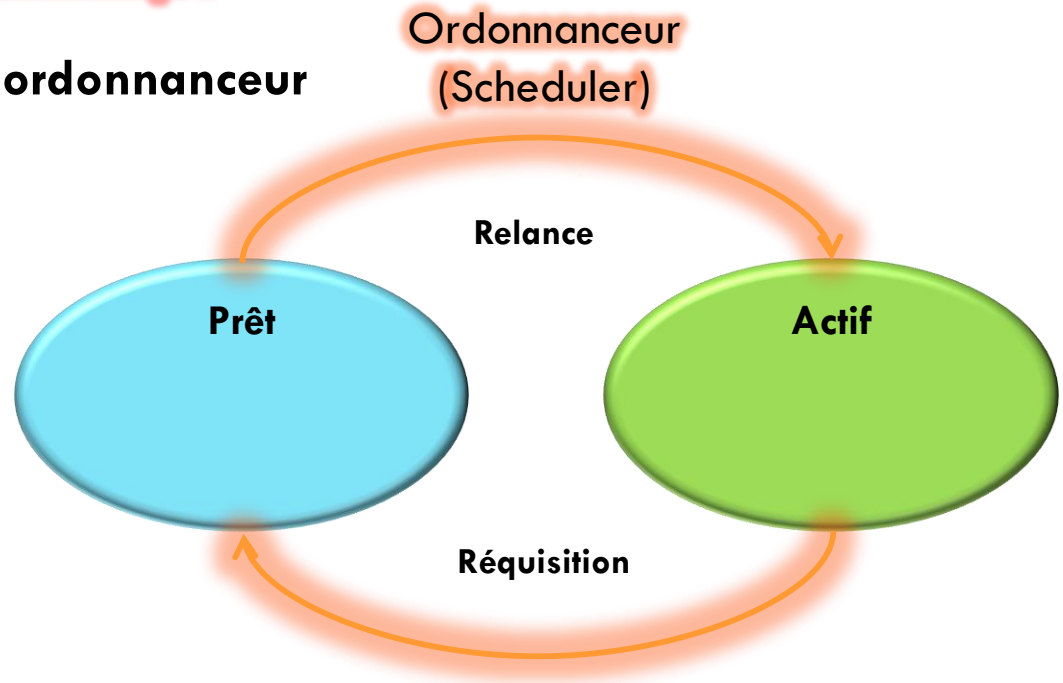
Interruption (it) Vs Déroutement (dt)

it	dt
Externe au processeur	Interne au processeur
Attente possible	Traitement immédiat
Traitée sur n'importe quel processeur (cas multiprocesseurs)	Traité par le processeur concerné
Niveaux de priorité	Pas de priorité

Ordonnancement

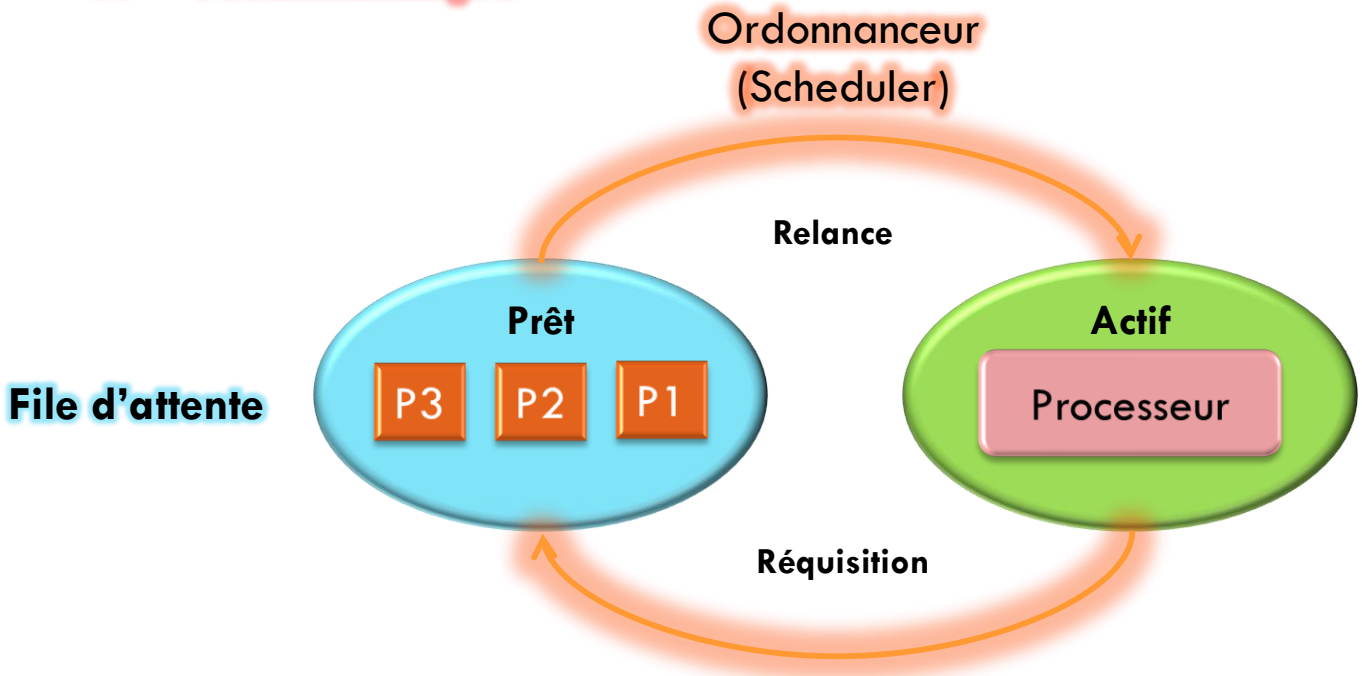
1. Problématique

Revenons à l'ordonnanceur



Ordonnancement

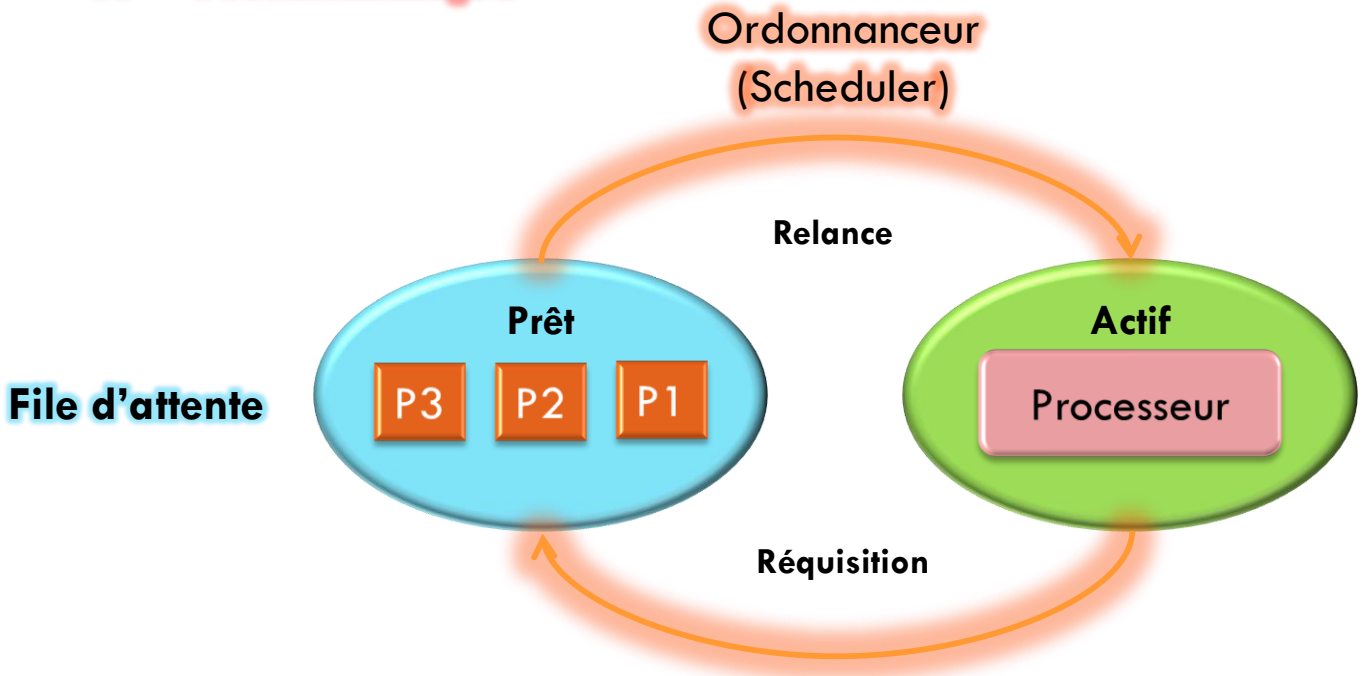
1. Problématique



La gestion de la concurrence des processus prêts pour l'obtention de processeur est effectuée par l'ordonnanceur

Ordonnancement

1. Problématique



Pour faire le choix, l'ordonnanceur se base sur un algorithme d'ordonnancement

Ordonnancement

1. Problématique

L'algorithme d'ordonnancement doit assuré :

a. **Équité : chaque processus doit avoir sa part du temps processeur**

b. **Efficacité : utiliser le temps processeur 100%**

c. **Rendement : augmenter le nombre de travaux effectués par UT**

Ordonnancement

2. Types

Il existe deux type d'algorithme d'ordonnancement :

- **Ordonnancement non préemptif (sans réquisition)**
  - **Un processus est exécuté sans interruption**

- **Ordonnancement préemptif (avec réquisition)**
  - **L'ordonnanceur peut interrompre l'exécution d'un processus pour allouer le processeur à un autre processus**

Ordonnancement

2. Types

Il existe deux type d'algorithme d'ordonnancement :

- **Ordonnancement non préemptif (sans réquisition)**
  - **FCFS (First Come First Served)**
  - **SJF (Shortest Job First)**
- **Ordonnancement préemptif (avec réquisition)**
  - **RR (Round Robin)**
  - **Priorité**

Ordonnancement

2. Types

- **Paramètres de performance**

- **Temps de Réponse d'un processus = date de sa fin – date de son arrivée**
- **Temps de Réponse Moyen (TRM) = Σ TR des processus/Nombre de processus**
- **Temps d'Attente d'un processus = TR du processus – sa durée**
- **Temps d'Attente Moyen (TAM) = Σ TA processus/ Nombre de processus**
- **Nombre de commutation de contexte = nbre changement de contexte**

Ordonnancement

2. Types

- **Paramètres de performance**

- **Temps de Réponse d'un processus = date de sa fin – date de son arrivée**
- **Temps de Réponse Moyen (TRM) = Σ TR des processus/Nombre de processus**
- **Temps d'Attente d'un processus = TR du processus – sa durée**
- **Temps d'Attente Moyen (TAM) = Σ TA processus/ Nombre de processus**
- **Nombre de commutation de contexte = nbre changement de contexte**

Ordonnancement

2. Types

- **Paramètres de performance**

- **Temps de Réponse d'un processus = date de sa fin – date de son arrivée**
- **Temps de Réponse Moyen (TRM) = Σ TR des processus/Nombre de processus**
- **Temps d'Attente d'un processus = TR du processus – sa durée**
- **Temps d'Attente Moyen (TAM) = (Σ TR - Σ durée)/ Nombre de processus**
- **Nombre de commutation de contexte = nbre changement de contexte**

Ordonnancement

3. Algorithmes

- **FCFS**

Le passage des processus sur le processeur est réalisé selon leur ordre d'arrivée c.à.d. chronologique (la file d'attente est gérée selon le FIFO)

Ordonnancement

3. Algorithmes

- **FCFS**

Le passage des processus sur le processeur est réalisé selon leur ordre d'arrivée c.à.d. chronologique (la file d'attente est gérée selon le FIFO)

Avantages

- ✓ **Simplicité**
- ✓ **Équité**

Inconvénients

- × **Pas d'optimisation**

Ordonnancement

3. Algorithmes

- **FCFS**

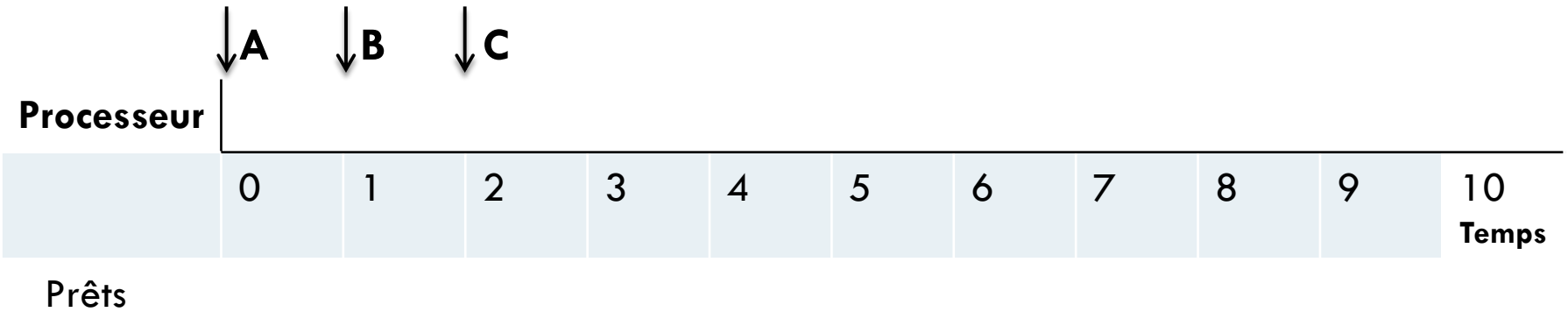
Processus	Durée	Date d'arrivée
A	5	0
B	3	1
C	2	2

Ordonnancement

3. Algorithmes

- **FCFS**

Processus	Durée	Date d'arrivée
A	5	0
B	3	1
C	2	2

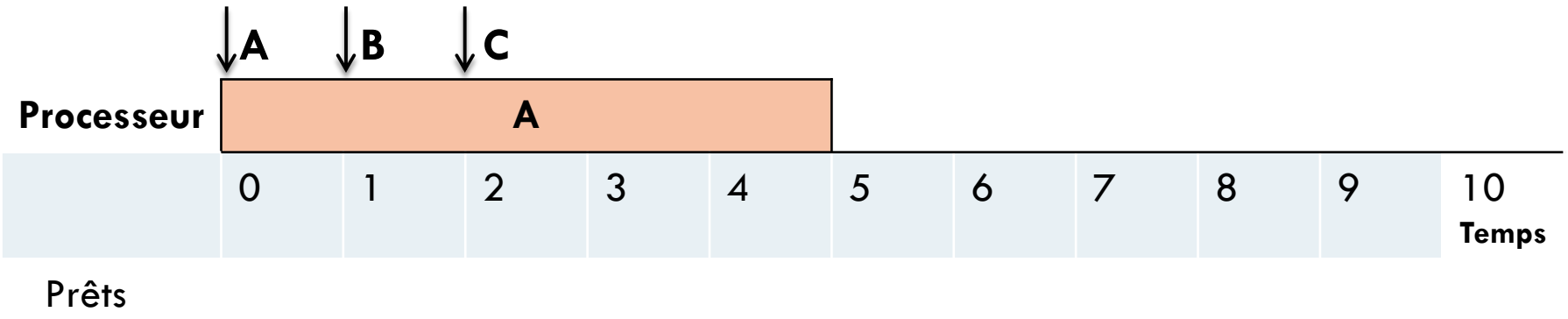


Ordonnancement

3. Algorithmes

- **FCFS**

Processus	Durée	Date d'arrivée
A	5	0
B	3	1
C	2	2



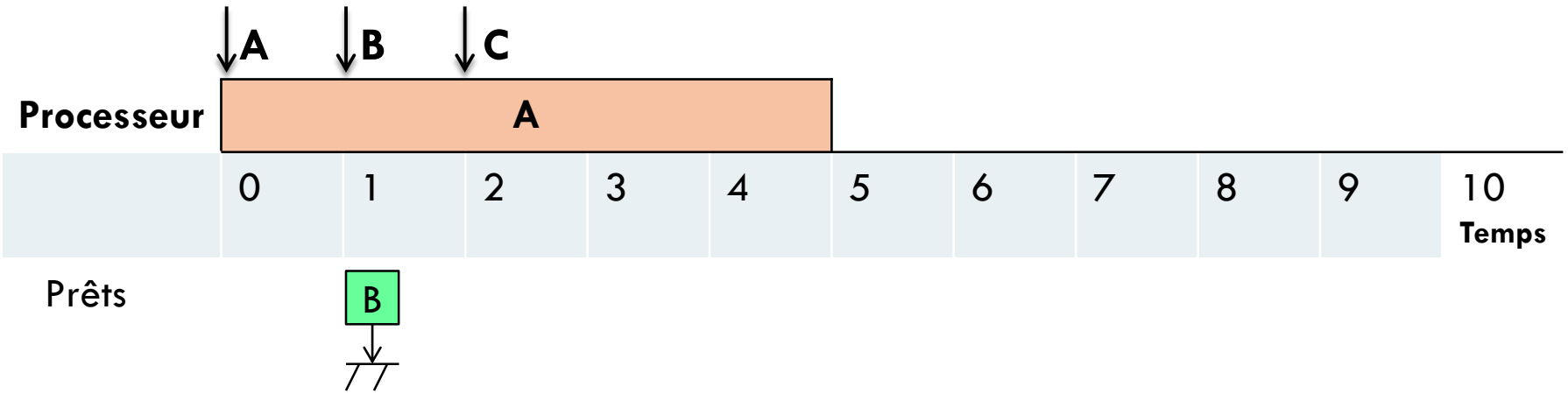
Prêts

Ordonnancement

3. Algorithmes

- **FCFS**

Processus	Durée	Date d'arrivée
A	5	0
B	3	1
C	2	2

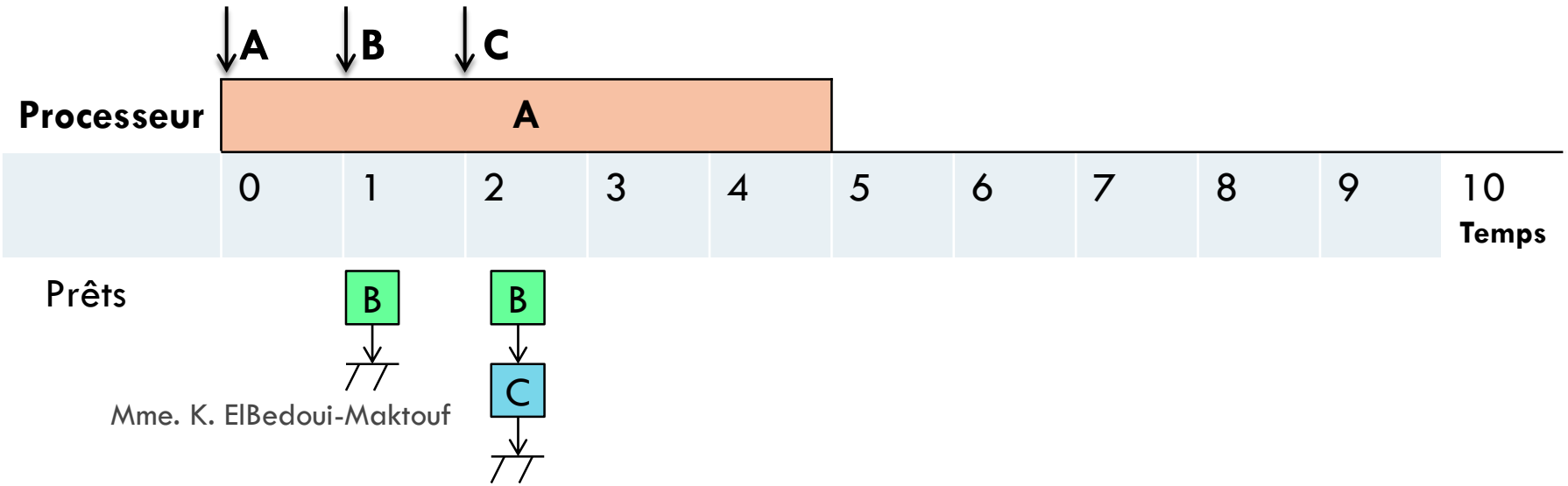


Ordonnancement

3. Algorithmes

- **FCFS**

Processus	Durée	Date d'arrivée
A	5	0
B	3	1
C	2	2

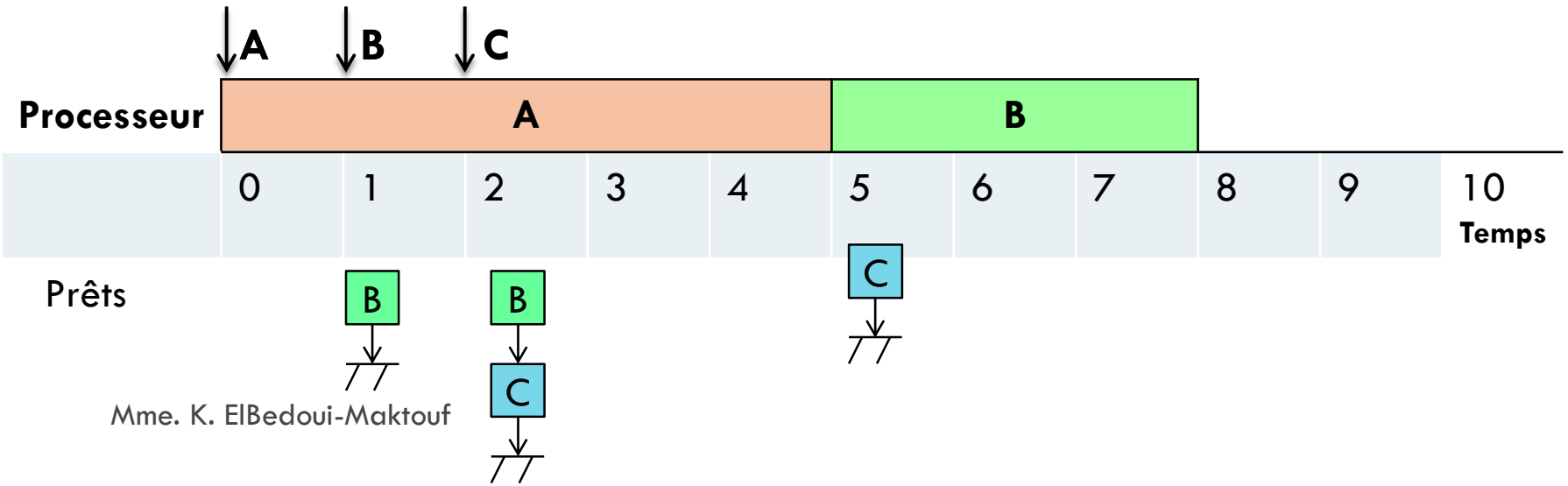


Ordonnancement

3. Algorithmes

- **FCFS**

Processus	Durée	Date d'arrivée
A	5	0
B	3	1
C	2	2

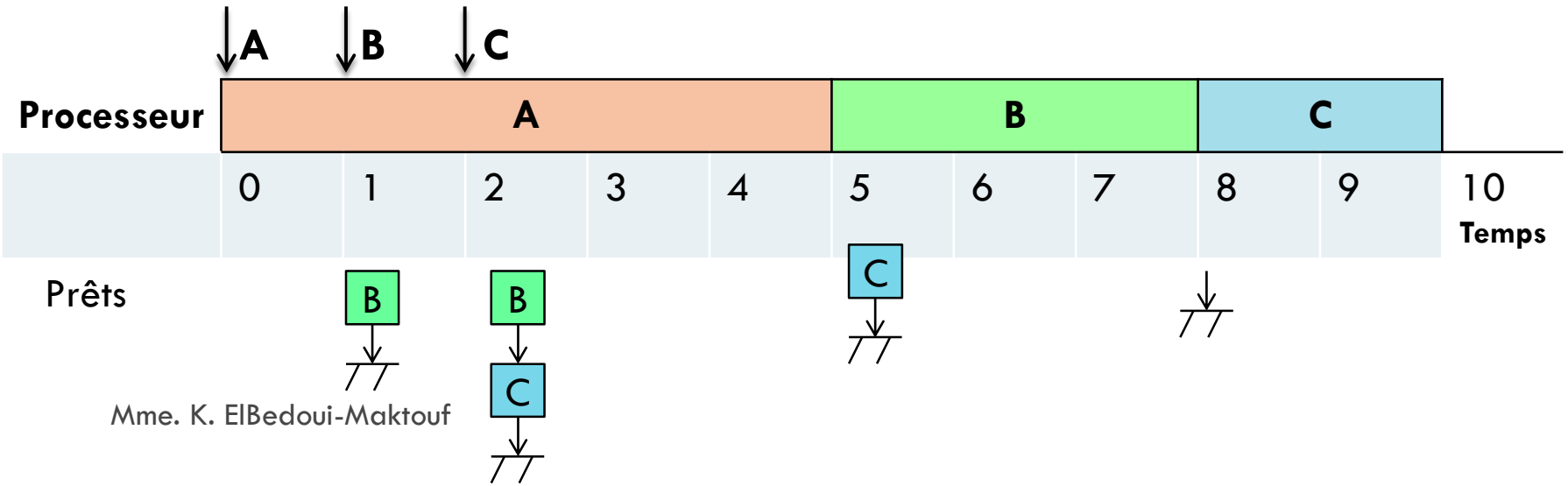


Ordonnancement

3. Algorithmes

- **FCFS**

Processus	Durée	Date d'arrivée
A	5	0
B	3	1
C	2	2



Ordonnancement

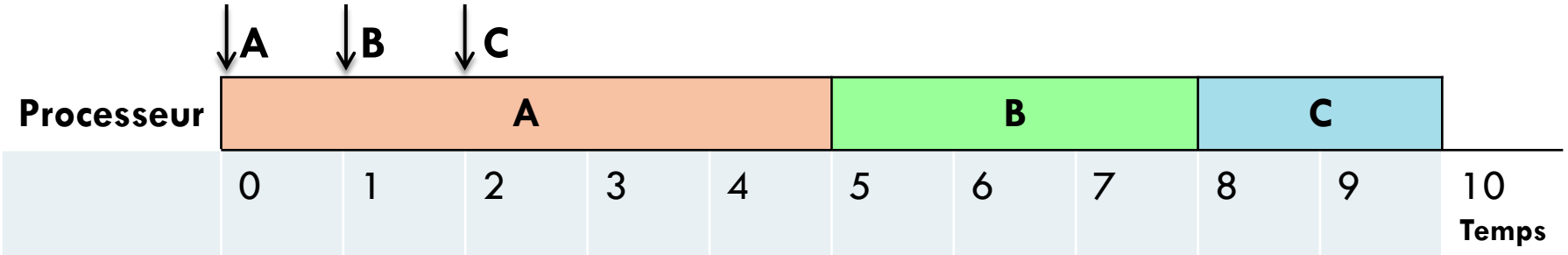
3. Algorithmes

- **FCFS**

TRM = ((5-0)+(8-1)+(10-2)) /3 =20/3

TAM = ((0-0)+(5-1)+(8-2)) /3 ou =20-10 /3 =10/3

NbCC = 2



Ordonnancement

3. Algorithmes

- **SJF (Shortest Job First)**

  Le passage des processus sur le processeur est réalisé selon la plus petite durée (suppose la connaissance en avance des durées)

Ordonnancement

3. Algorithmes

- **SJF (Shortest Job First)**

Le passage des processus sur le processeur est réalisé selon la plus petite durée (suppose la connaissance en avance des durées)

Avantages

- ✓ **TRM réduit**

Inconvénients

- ✗ **Difficile à mettre en œuvre à cause de la difficulté de connaître en avance la durée de l'exécution**
- ✗ **Risque de famine pour les processus de longue durée**

Ordonnancement

3. Algorithmes

- **SJF**

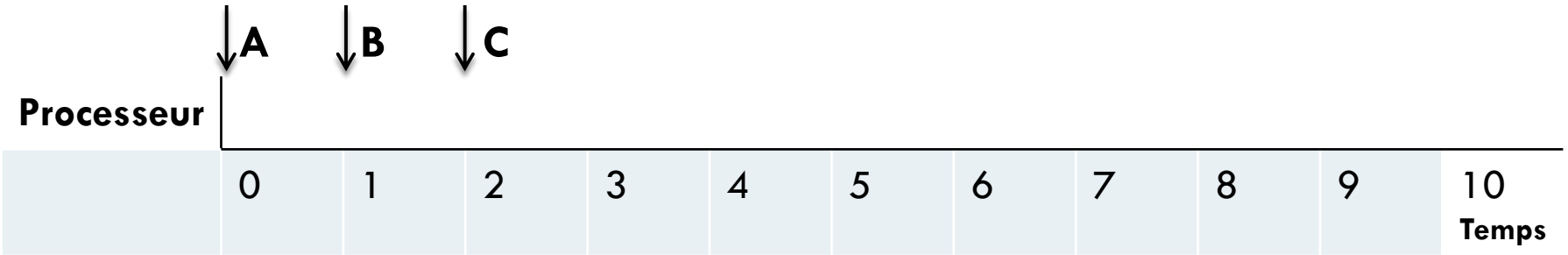
Processus	Durée	Date d'arrivée
A	5	0
B	3	1
C	2	2

Ordonnancement

3. Algorithmes

- **SJF**

Processus	Durée	Date d'arrivée
A	5	0
B	3	1
C	2	2

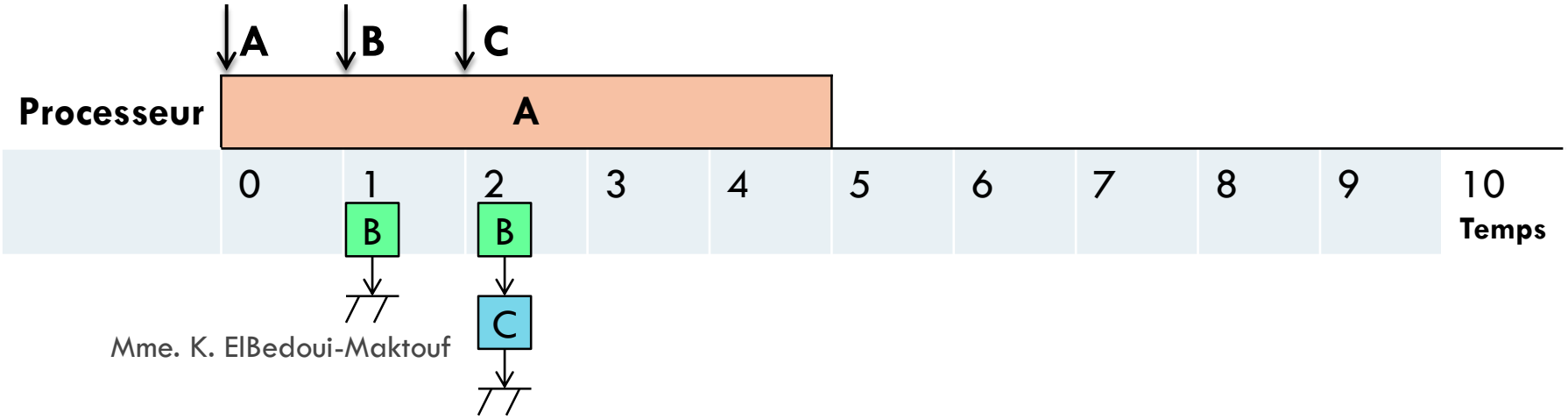


Ordonnancement

3. Algorithmes

- **SJF**

Processus	Durée	Date d'arrivée
A	5	0
B	3	1
C	2	2

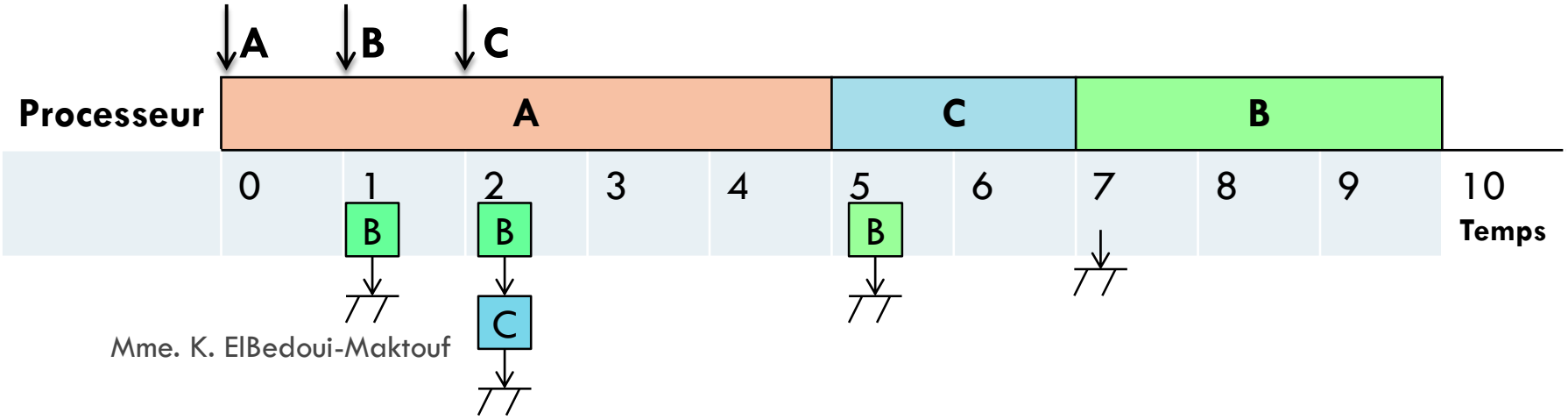


Ordonnancement

3. Algorithmes

- **SJF**

Processus	Durée	Date d'arrivée
A	5	0
B	3	1
C	2	2



Ordonnancement

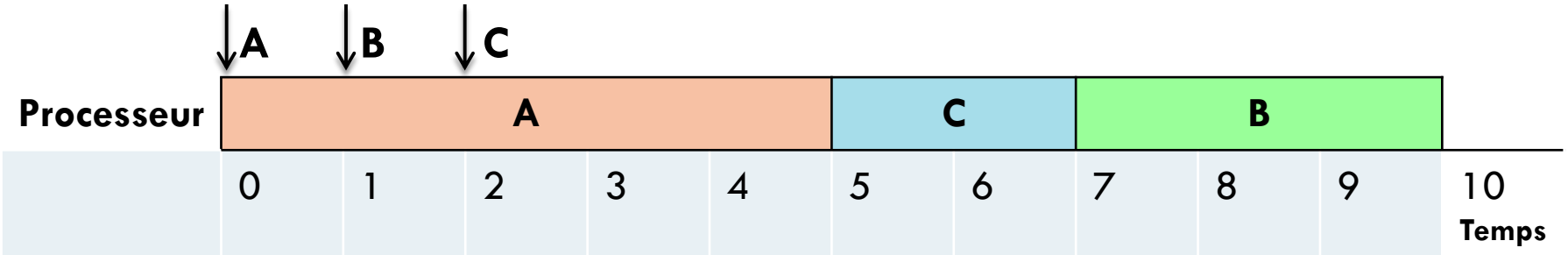
3. Algorithmes

- **SJF**

TRM = ((5-0)+(10-1)+(7-2)) /3 =19/3

TAM = ((0-0)+(7-1)+(5-2)) /3 ou =19-10 /3 = 9/3

NbCC = 2



Ordonnancement

3. Algorithmes

- **RR (Round Robin)**

  Chaque processus a un laps du temps fixe dit quantum durant lequel il bénéficie du processeur

Ordonnancement

3. Algorithmes

- **RR (Round Robin)**

Chaque processus a un laps du temps fixe dit quantum durant lequel il bénéficie du processeur

Avantages

- ✓ **Simplicité**
- ✓ **Équité**

Inconvénients **Problème de choix du quantum**

- ✗ **Trop petit** => **trop de commutations**
- ✗ **Trop grand** => **augmente le TR (≈ FCFS)**

Ordonnancement

3. Algorithmes

- **RR (Round Robin)**

Processus	Durée	Date d'arrivée
A	5	0
B	3	1
C	2	2

La valeur du quantum est supposée

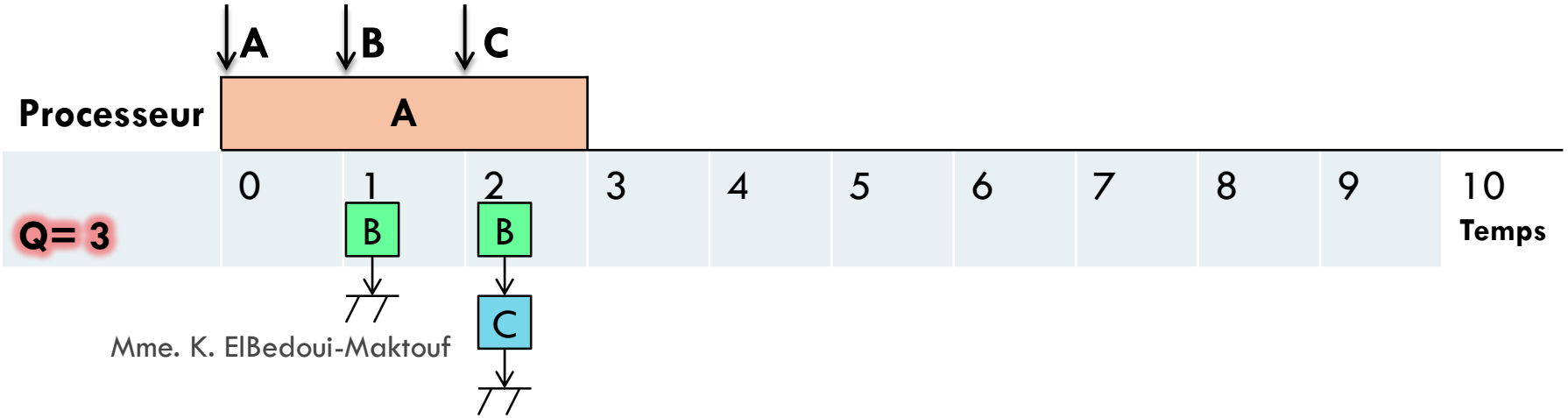
Q= 3

Ordonnancement

3. Algorithmes

- **RR (Round Robin)**

Processus	Durée	Date d'arrivée
A	5	0
B	3	1
C	2	2

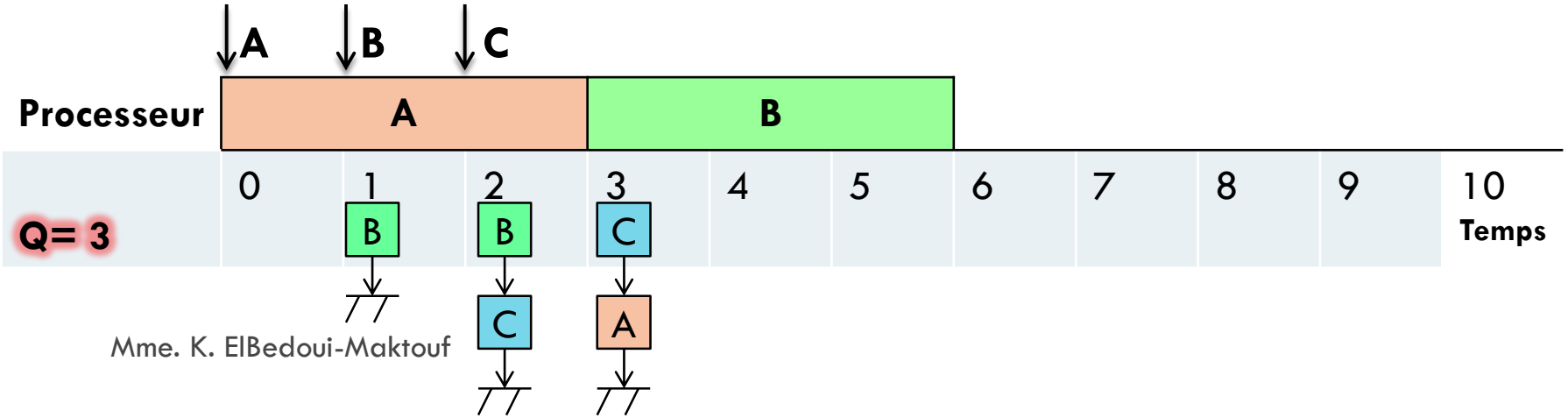


Ordonnancement

3. Algorithmes

- **RR (Round Robin)**

Processus	Durée	Date d'arrivée
A	5	0
B	3	1
C	2	2

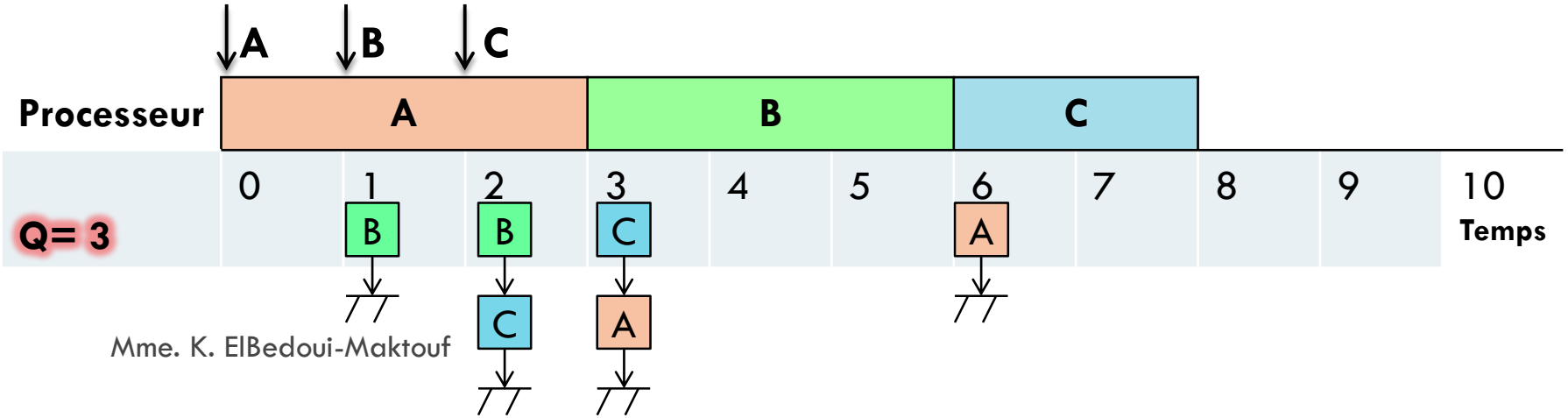


Ordonnancement

3. Algorithmes

- **RR (Round Robin)**

Processus	Durée	Date d'arrivée
A	5	0
B	3	1
C	2	2

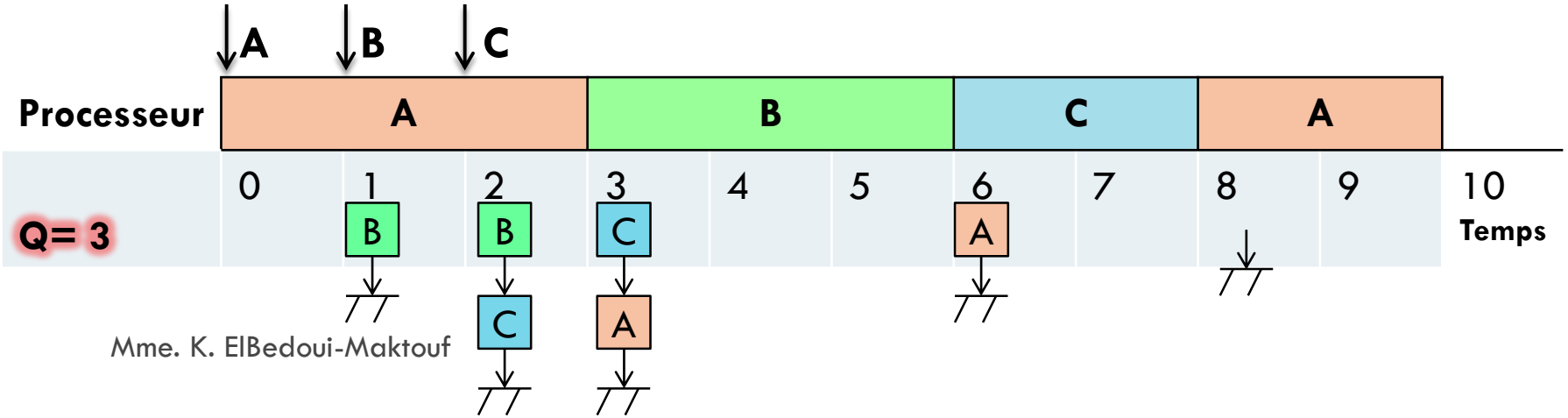


Ordonnancement

3. Algorithmes

- **RR (Round Robin)**

Processus	Durée	Date d'arrivée
A	5	0
B	3	1
C	2	2



Ordonnancement

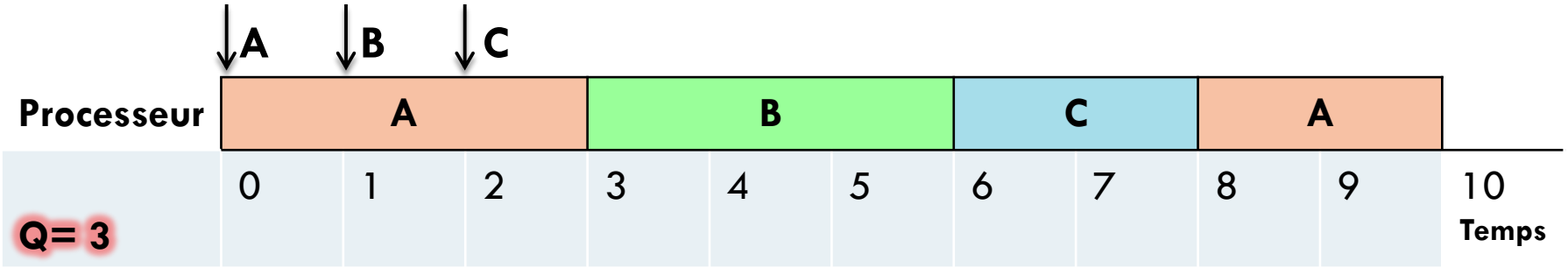
3. Algorithmes

- **RR (Round Robin)**

TRM = ((10-0)+(6-1)+(8-2)) /3 =21/3

TAM = ((0-0)+(8-3)+(3-1)+(6-2)) /3 ou 21-10 /3 =11/3

NbCC = 3



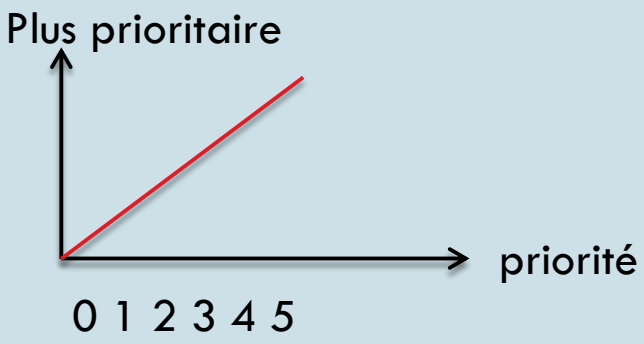
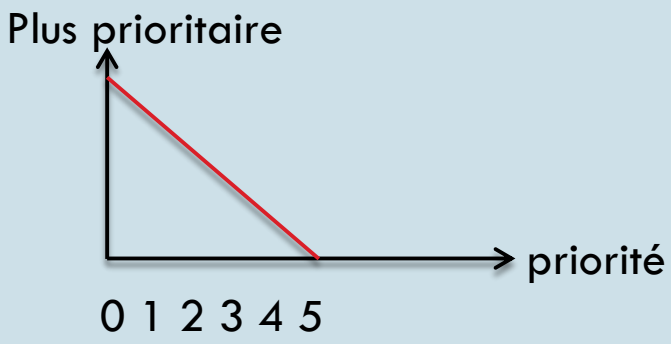
Ordonnancement

3. Algorithmes

- **Priorité**

Chaque processus a une priorité et l'ordonnanceur lance à chaque fois le processus le plus prioritaire

Système de priorité :

Croissant	Décroissant
	

Ordonnancement

3. Algorithmes

- **Priorité**

Chaque processus a une priorité et l'ordonnanceur lance à chaque fois le processus le plus prioritaire

- **Statique: les priorités sont fixes au cours du temps**

- **Dynamique: les priorités changent au cours du temps**

(sous Linux: commande nice et renice)

Ordonnancement

3. Algorithmes

- **Priorité**

Chaque processus a une priorité et l'ordonnanceur lance à chaque fois le processus le plus prioritaire

Avantages

- ✓ **Statique: mise en œuvre simple**
- ✓ **Dynamique: minimise la famine**

Inconvénients

- × **Statique: risque de famine**
- × **Dynamique: mise en œuvre complexe**

Ordonnancement

3. Algorithmes

- **Priorité**

Statique

Processus	Durée	Date d'arrivée	Priorité ↗
A	5	0	1
B	3	1	3
C	2	2	2

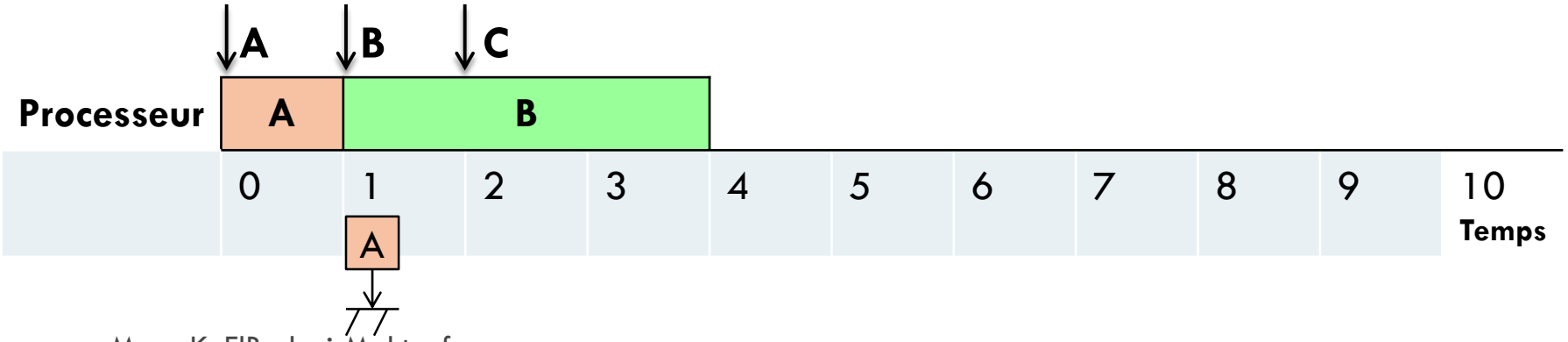
Ordonnancement

3. Algorithmes

- **Priorité**

Statique

Processus	Durée	Date d'arrivée	Priorité ↗
A	5	0	1
B	3	1	3
C	2	2	2



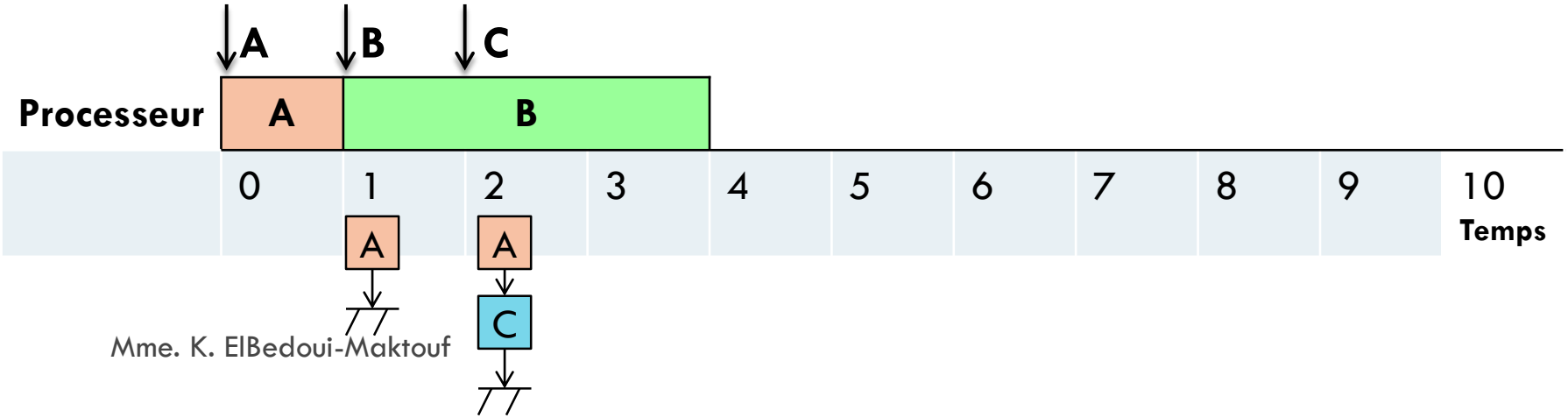
Ordonnancement

3. Algorithmes

- **Priorité**

Statique

Processus	Durée	Date d'arrivée	Priorité ↗
A	5	0	1
B	3	1	3
C	2	2	2



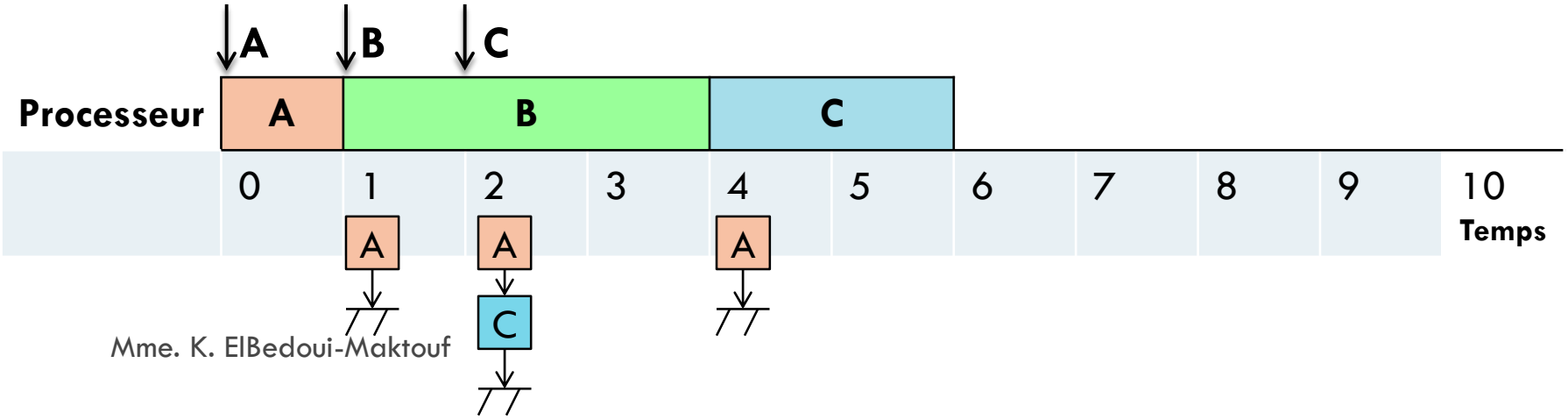
Ordonnancement

3. Algorithmes

- **Priorité**

Statique

Processus	Durée	Date d'arrivée	Priorité ↗
A	5	0	1
B	3	1	3
C	2	2	2



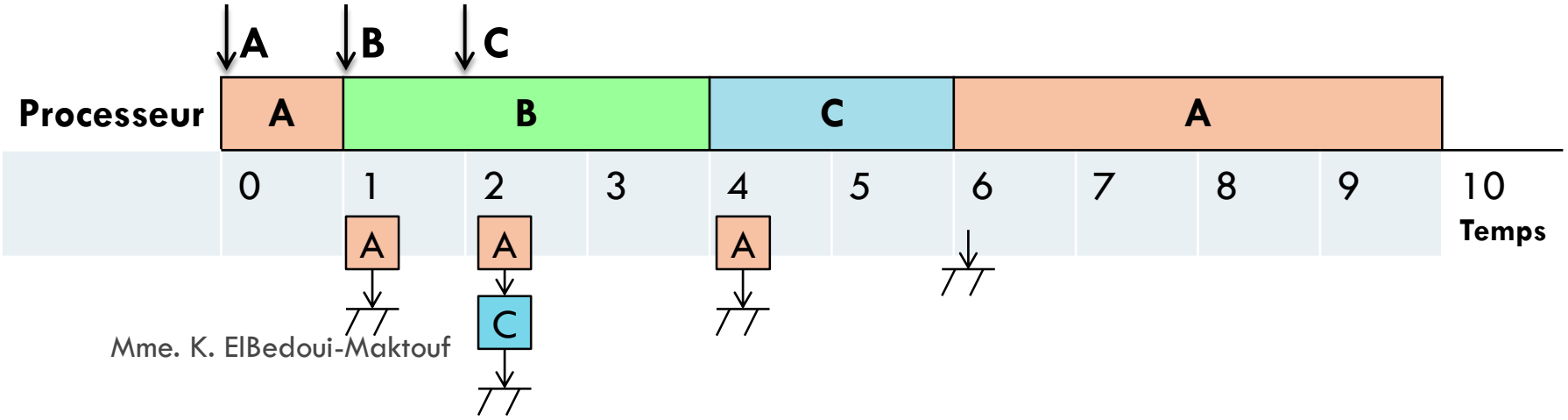
Ordonnancement

3. Algorithmes

- **Priorité**

Statique

Processus	Durée	Date d'arrivée	Priorité ↗
A	5	0	1
B	3	1	3
C	2	2	2



Ordonnancement

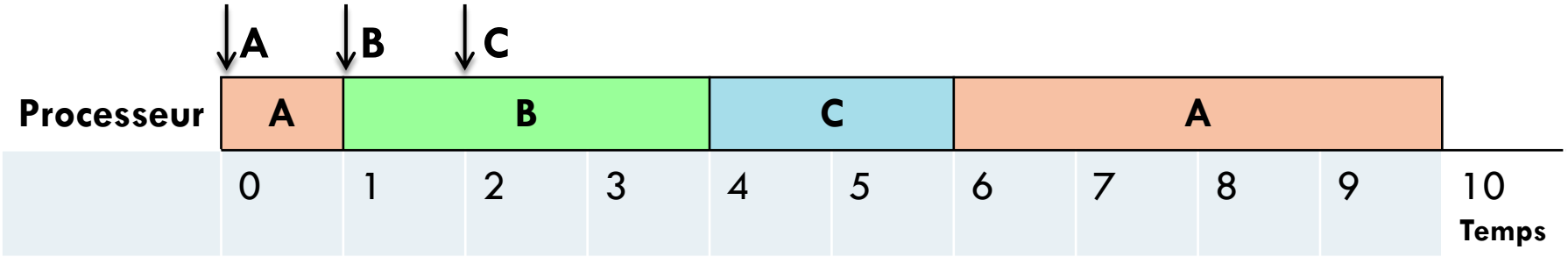
3. Algorithmes

- **Priorité**

TRM = ((10-0)+(4-1)+(6-2)) /3 =17/3

TAM = ((0-0)+(6-1) +(1-1)+(4-2)) /3 ou =17-10 /3 = 7/3

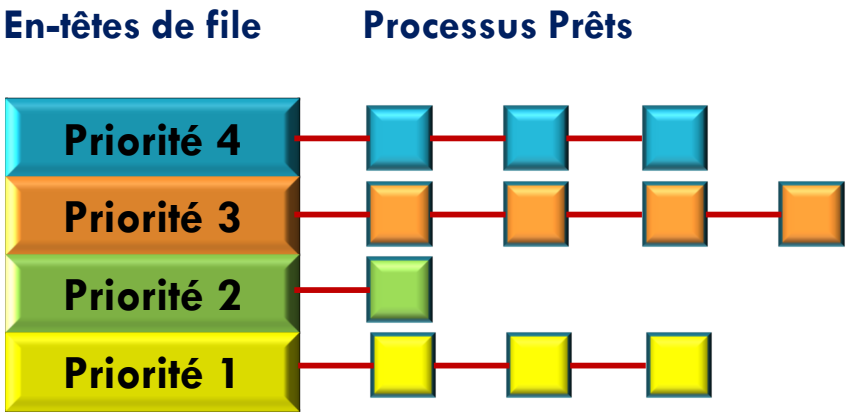
NbCC = 3



Ordonnancement

4. Multi-niveaux

- Les processus sont servis selon la priorité
- Les processus d'une même priorité sont servis selon FCFS ou RR



Etude de cas

1. Cas d'Unix

- Deux niveaux: haut et bas niveaux

- **Ordonnanceur de haut niveau** (Swapper): se charge des transferts des processus entre la mémoire centrale et le disque

Un processus bloqué ne reste pas en mémoire

Etude de cas

1. Cas d'Unix

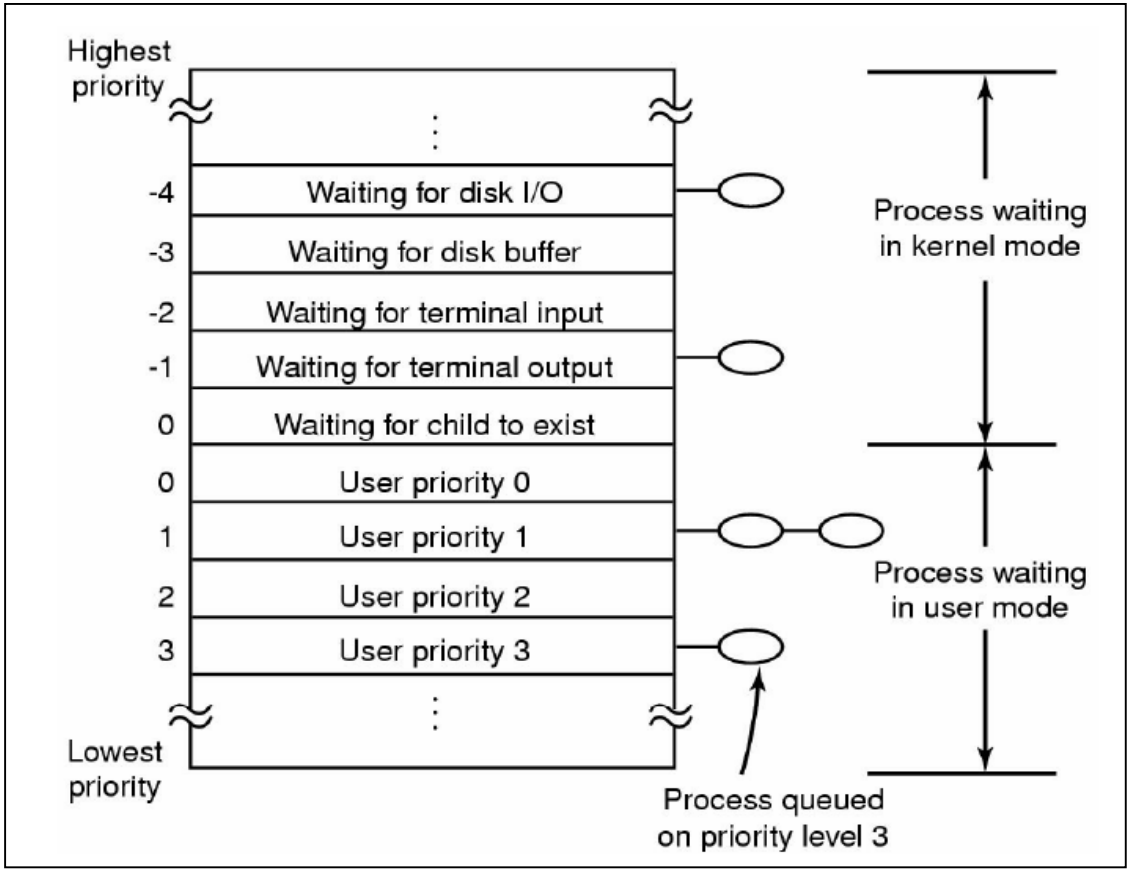
- Deux niveaux: haut et bas niveaux
- **Ordonnanceur de bas niveau** (Scheduler): se charge de choisir un processus parmi ceux qui résident en mémoire (prêts)
  - Ordonnancement par niveau de priorités
  - Ordonnancement RR pour la même priorité (Q =100ms)
  - Les propriétés des processus en mode utilisateur ≥ 0
  - Les propriétés des processus en mode noyau < 0
  - Les priorités sont dynamiquement calculées

Priorité = CPU_usage + nice + base

Etude de cas

1. Cas d'Unix

Processus en mode noyau sont plus prioritaires que les processus en mode utilisateur



Etude de cas

2. Cas de Linux

- Trois classes de processus
  - Processus FIFO SCHED_FIFO : prioritaires non préemptibles
  - Processus Tourniquet SCHED_RR: prioritaires interruptibles
  - Processus Temps partagé OTHER : les moins prioritaires, s'exécutent uniquement si aucun processus des classes 1 et 2 n'est prêt

- Chaque processus créé fait partie d'une des trois classes et a une priorité comprise entre 1 et 40 (20 par défaut)
- Les valeurs élevées correspondent à des priorités élevées

Etude de cas

2. Cas de Linux

- Chaque processus de la deuxième et troisième classe a un quantum exprimé en *jiffies*

- *Jiffy*= Top horloge = 10ms

- Quantum d'un processus = valeur de sa priorité * *jiffy*

- **Exemple:**
  - Processus avec priorité initiale 30
  - Quantum de ce processus = 30* *jiffy* = 30 * 10 = 300 ms

Etude de cas

2. Cas de Linux

- **Algorithme d'ordonnancement**

Calcul de Note :

Si processus ∈ {SCHED_FIFO, SCHED_RR} alors

Note = 1000 + Priorité

Sinon (processus = OTHER)

Si Quantum > 0 alors

Note = Quantum + Priorité

Sinon (Quantum = 0)

Note = 0

Choix du processus dont la valeur de Note est la plus élevée

Etude de cas

2. Cas de Linux

❖ Algorithme d'ordonnancement

Toutes les 10ms (*Jiffy*)

Quantum = Quantum -1

Les quanta sont réajustés lorsque tous les processus sont bloqués ou leurs quanta sont nuls :

Quantum = Quantum / 2 + Priorité

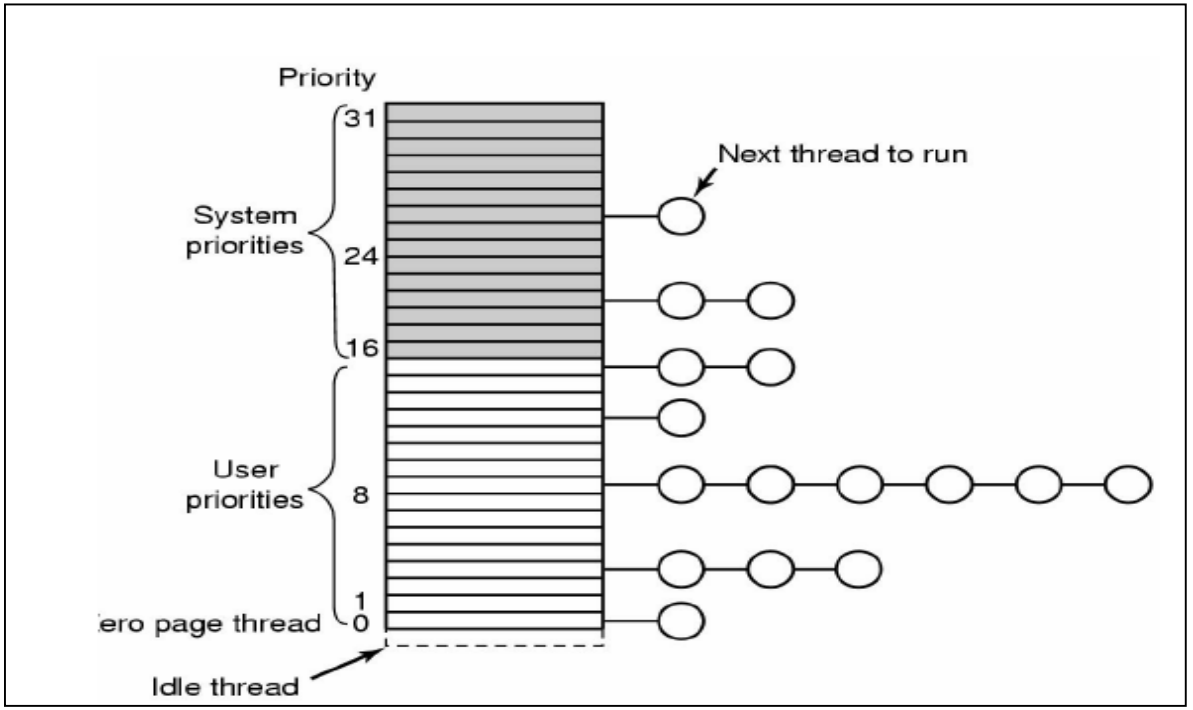
Retirer le processus du CPU si:

- **Son quantum atteint 0**
- **Il se bloque en attente d'un évènement**
- **Un processus de plus haute Note bloqué auparavant redevient prêts**

Etude de cas

3. Cas de Windows XP

Préemptif et à priorité avec files multiples (32 niveaux de priorité). Les priorités sont dynamiques.



Etude de cas

3. Cas de Windows XP

Un processus exécute le code de l'ordonnanceur dans 3 cas:

- **Il se bloque**
- **Il libère une ressource**
- **Son quantum a expiré**

Remarque

Le code de l'ordonnanceur est aussi appelé à la fin d'une E/S.

Etude de cas

3. Cas de Windows XP

Algorithme d'ordonnancement:

- **L'ordonnanceur choisit le processus le plus prioritaire en tête de file et lui alloue le processeur pendant au maximum un quantum.**
- **Quand un processus est sélectionné, son exécution peut être interrompue (avant la fin du quantum) si un autre processus de priorité plus élevée devient prêt. Le processus préempté est remis en tête de la file associée à son niveau de priorité.**
  - **Si un processus est suspendu suite à sa consommation de son quantum, alors sa priorité peut être diminuée.**
  - **S'il est interrompu pour une requête d'E/S, son quantum est décrémenté par contre sa priorité augmente (+1 pour un disque, +6 pour un clavier, +8 pour la carte son).**

Etude de cas

3. Cas de Windows XP

- **Si un processus utilisateur à l'état prêt reste trop longtemps en attente du processeur (depuis 4 secondes environ), il se voit attribuer une priorité 15 et un quantum de 4 unités (pour éviter le problème de famine). C'est le *balance set manager* qui se charge de la recherche de tels processus toutes les secondes.**

- **À la fin du quantum, le priorité du processus décrémente jusqu'à reprendre sa priorité de base originale.**

FIN

Madame Khaoula ElBedoui-Maktouf

2ème année Ingénieur Informatique