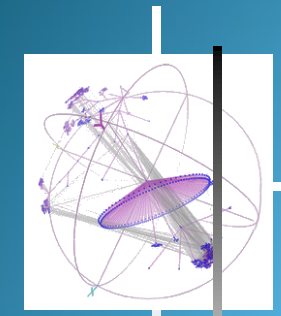
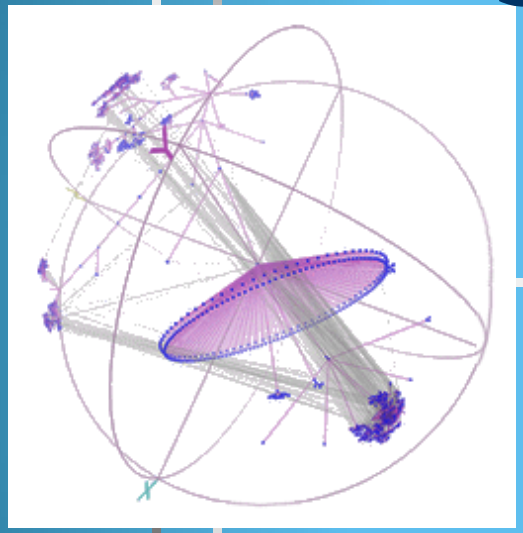
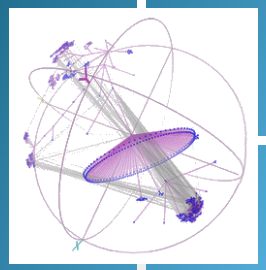




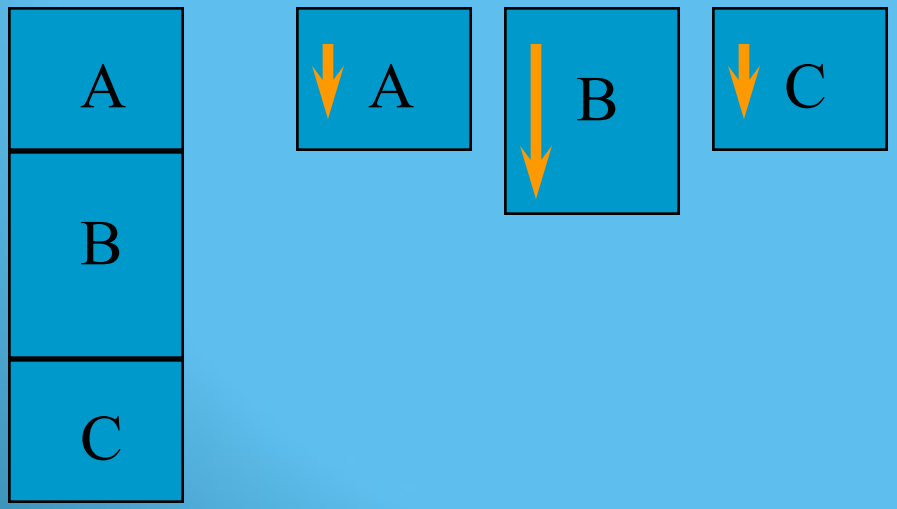
Systèmes d'exploitation
Chapitre II

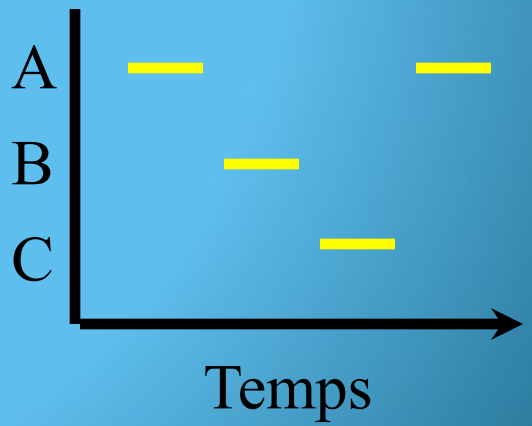
Gestion des processus

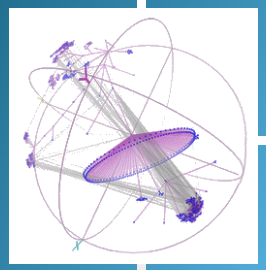


Processus

- “Un programme qui s’exécute”
- Les ordinateurs autorisent maintenant plusieurs processus simultanément (pseudo parallélisme)





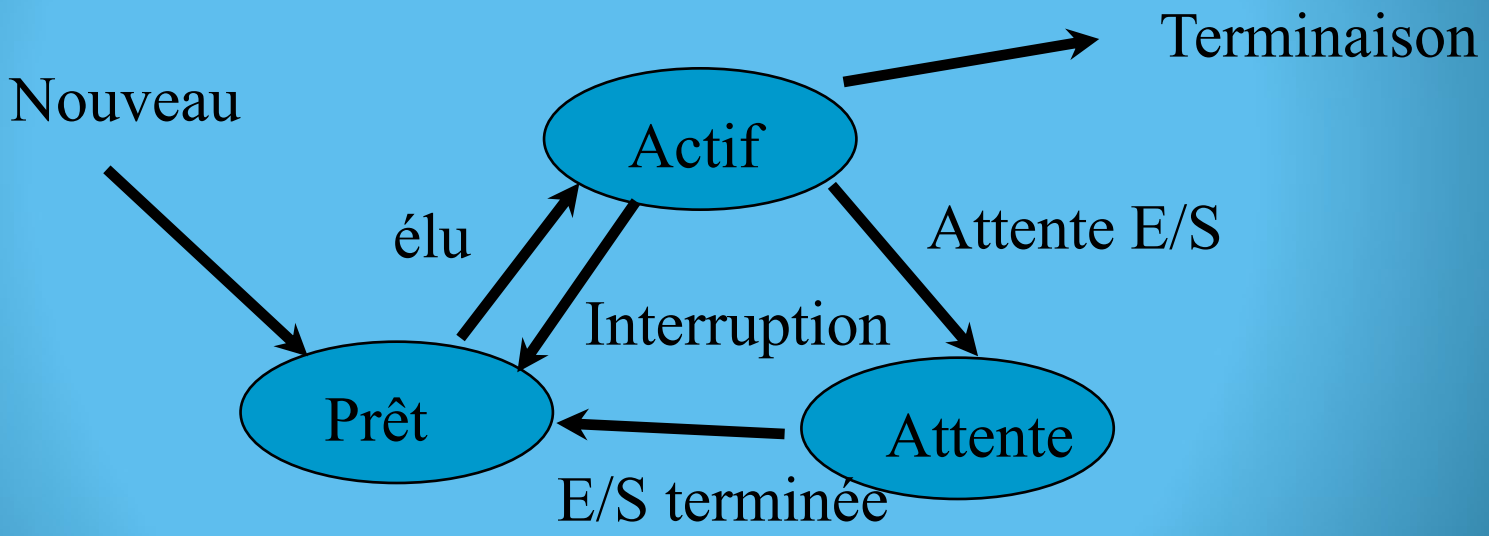


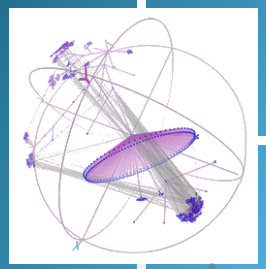
État de processus

- Au fur et a mesure qu'un processus s'exécute, il change d'état
  - nouveau: le processus vient d'être créé
  - exécutant-running: le processus est en train d'être exécuté par l 'UCT
  - attente-waiting: le processus est en train d'attendre un événement (p.ex. la fin d 'une opération d'E/S)
  - prêt-ready: le processus est en attente d'être exécuté par l 'UCT
  - terminé: fin d'exécution

Etats des processus

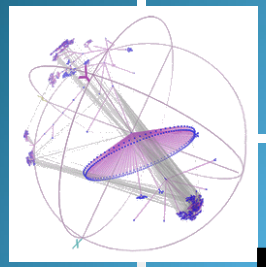
- Changement d'état des processus





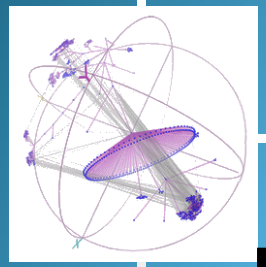
États Nouveau, Terminé

- **Nouveau**
  - Le SE a créé le processus
    - a construit un identificateur pour le processus
    - a construit les tableaux pour gérer le processus(PCB)
  - mais ne s'est pas encore engagé à exécuter le processus (pas encore admis)
    - pas encore alloué des ressources
  - La file des nouveaux travaux est souvent appelée spoule travaux (job spooler)
- **Terminé:**
  - Le processus n'est plus exécutable, mais ses données sont encore requises par le SE (comptabilité, etc.)



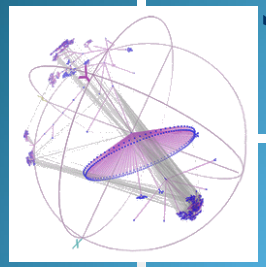
Transitions entre processus

- **Prêt → Exécution**
  - Lorsque **l'ordonnanceur** UCT choisit un processus pour exécution
- **Exécution → Prêt**
  - Résultat d'une interruption causée par un événement indépendant du processus
    - ☞ Il faut traiter cette interruption, donc le processus courant perd l'UCT
      - Cas important: le processus à épuisé son intervalle de temps (quantum)



Transitions entre processus

- **Exécution → Attente**
  - Lorsqu'un processus fait requête d'un service du SE que le SE ne peut offrir immédiatement (interruption causée par le processus lui-même)
    - un accès à une ressource pas encore disponible
    - initie une E/S: doit attendre le résultat
    - a besoin de la réponse d'un autre processus
- **Attente → Prêt**
  - lorsque l'événement attendu se produit

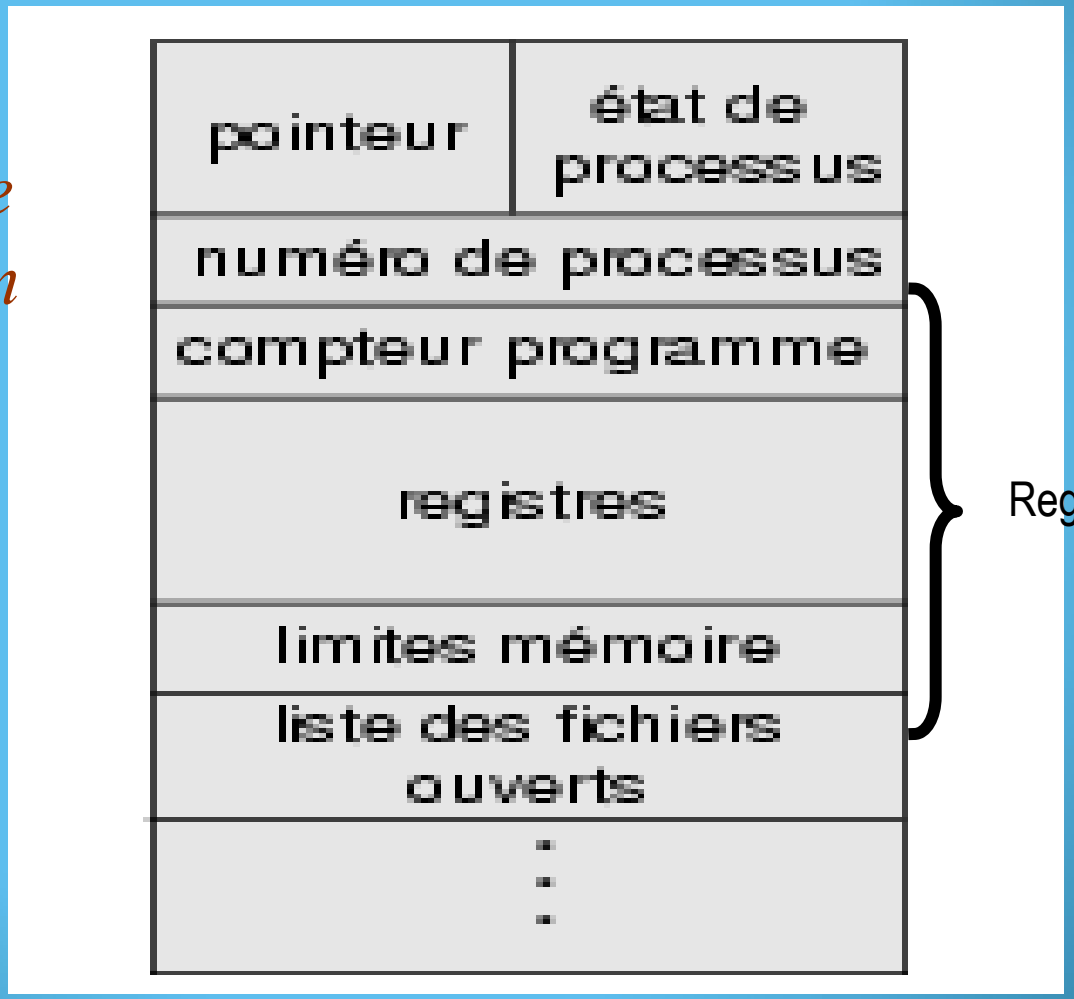


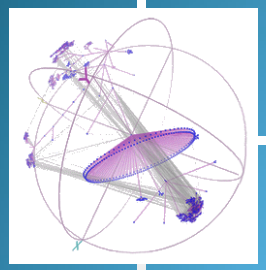
Sauvegarde d'informations du processus

- En multiprogrammation, un processus s'exécute sur l'UCT de façon intermittente
- Chaque fois qu'un processus reprend l'UCT (transition prêt $\rightarrow$ exécution) il doit la reprendre dans la même situation où il l'a laissée (même contenu des registres UCT, etc.)
- Donc au moment où un processus sort de l'état exécution il est nécessaire de sauvegarder ses informations essentielles, qu'il faudra récupérer quand il retourne à cet état

PCB = Process Control Block

Représente la situation actuelle d'un processus, pour le reprendre plus tard

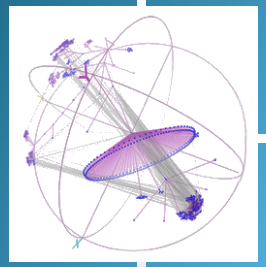




Process Control Block

Le PCB contient entre autres:

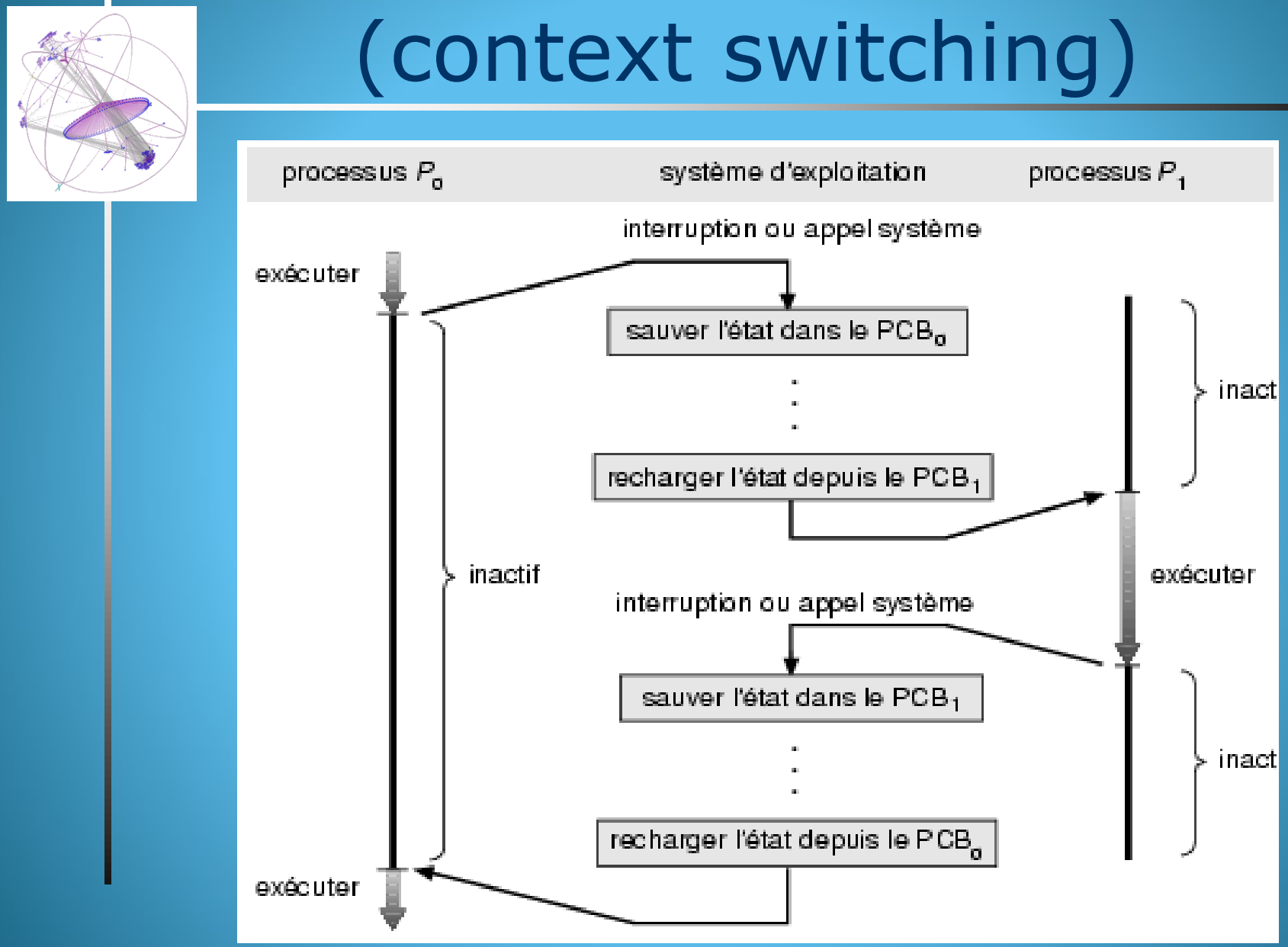
- pointeur: les PCBs sont rangés dans des listes enchaînées (à voir)
- état de processus: ready, running, waiting…
- compteur programme: le processus doit reprendre à l ’instruction suivante
- autres registres UCT
- bornes de mémoire
- fichiers qu’il a ouvert
- etc.,

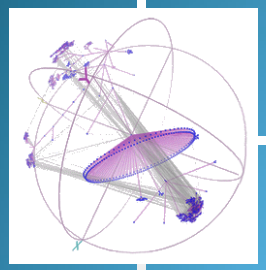


Commutation de processus

- appelé aussi commutation de contexte ou context switching
- Quand l'UCT passe de l'exécution d'un processus 0 à l'exécution d'un proc 1, il faut
  - mettre à jour le PCB de **0**
  - sauvegarder le PCB de **0**
  - reprendre le PCB de **1**, qui avait été sauvegardé avant
  - remettre les registres d 'UCT, compteur d 'instructions etc. dans la même situation qui est décrite dans le PCB de **1**

Commutation de processeur (context switching)



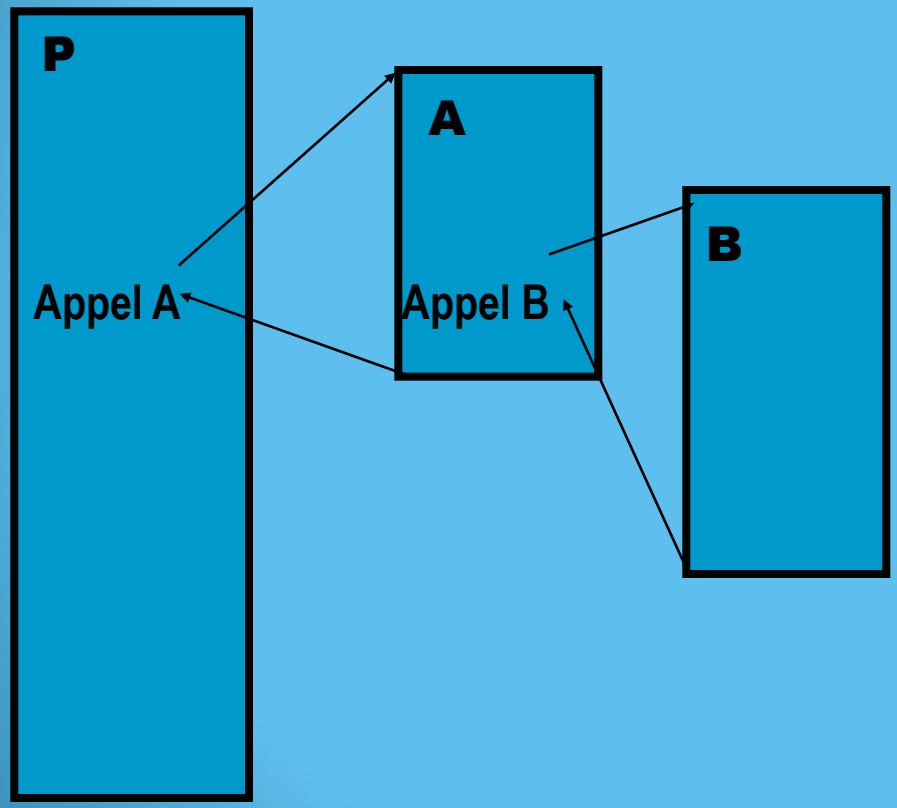


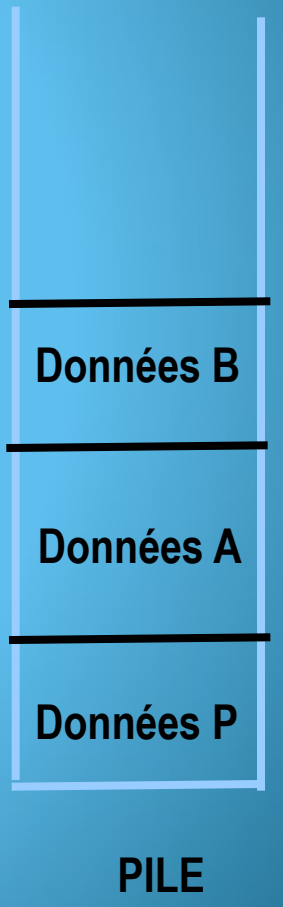
La pile d'un processus

Il faut aussi à sauvegarder entre autre: la pile d'un processus

- Quand un processus fait appel à une procédure, à une méthode, etc., il est nécessaire de mettre dans une pile l'adresse à laquelle le processus doit retourner après avoir terminé cette procédure, méthode, etc.
- Aussi on met dans cette pile les variables locales de la procédure qu'on quitte, les paramètres, etc., pour les retrouver au retour
- Donc il y a normalement une pile d'adresses de retour après interruption et une pile d'adresses de retour après appel de procédure
  - Ces deux piles fonctionnent de façon semblable, mais sont normalement séparées
- Les informations relatives à ces piles (base, pointeur…) doivent aussi être sauvegardées au moment de la commutation de contexte

La Pile d'un processus





Commutation de processus

Comme l'ordinateur n'a, la plupart du temps, qu'un processeur, il résout ce problème grâce à un pseudo-parallélisme

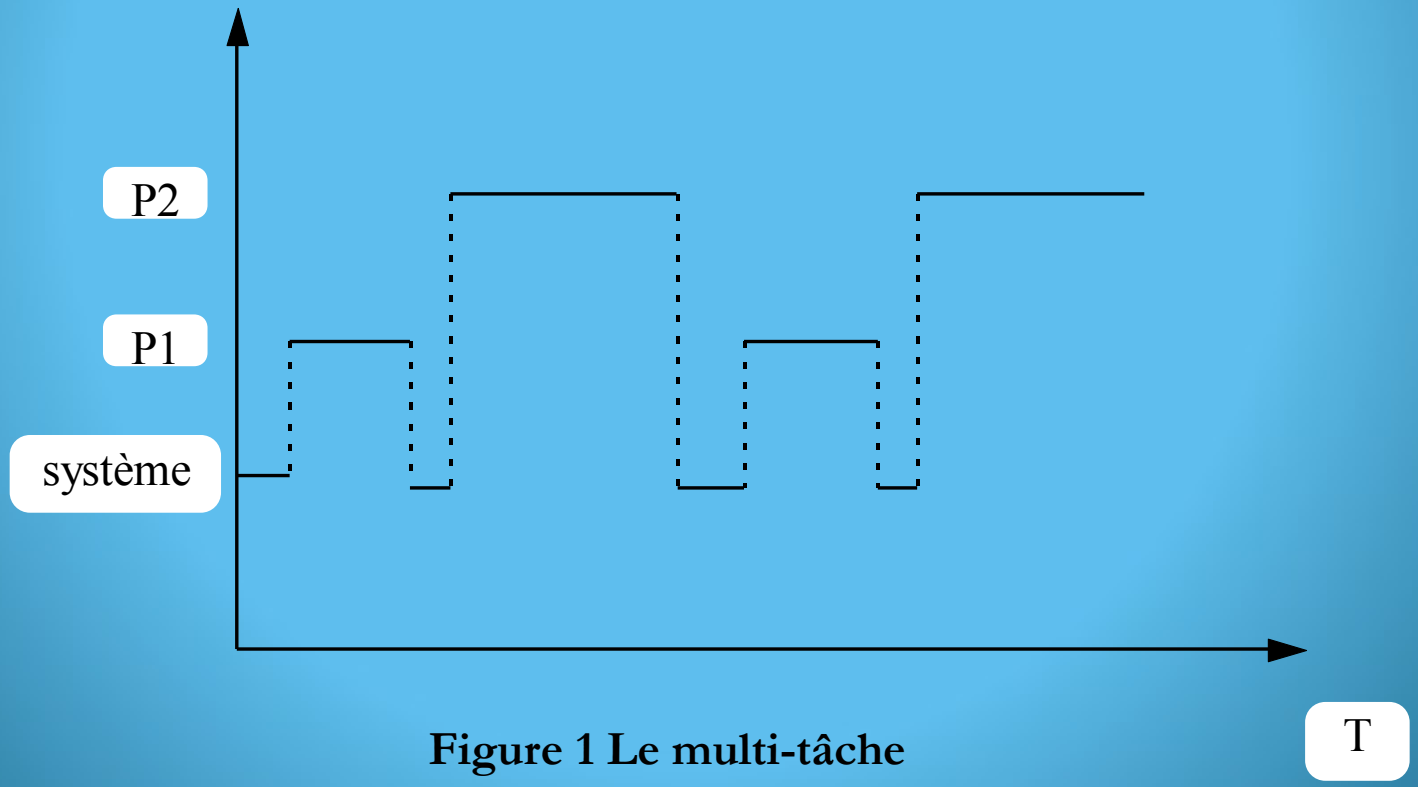
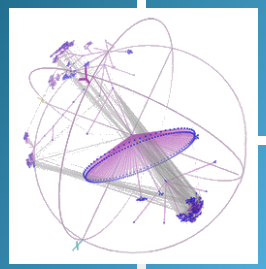


Figure 1 Le multi-tâche

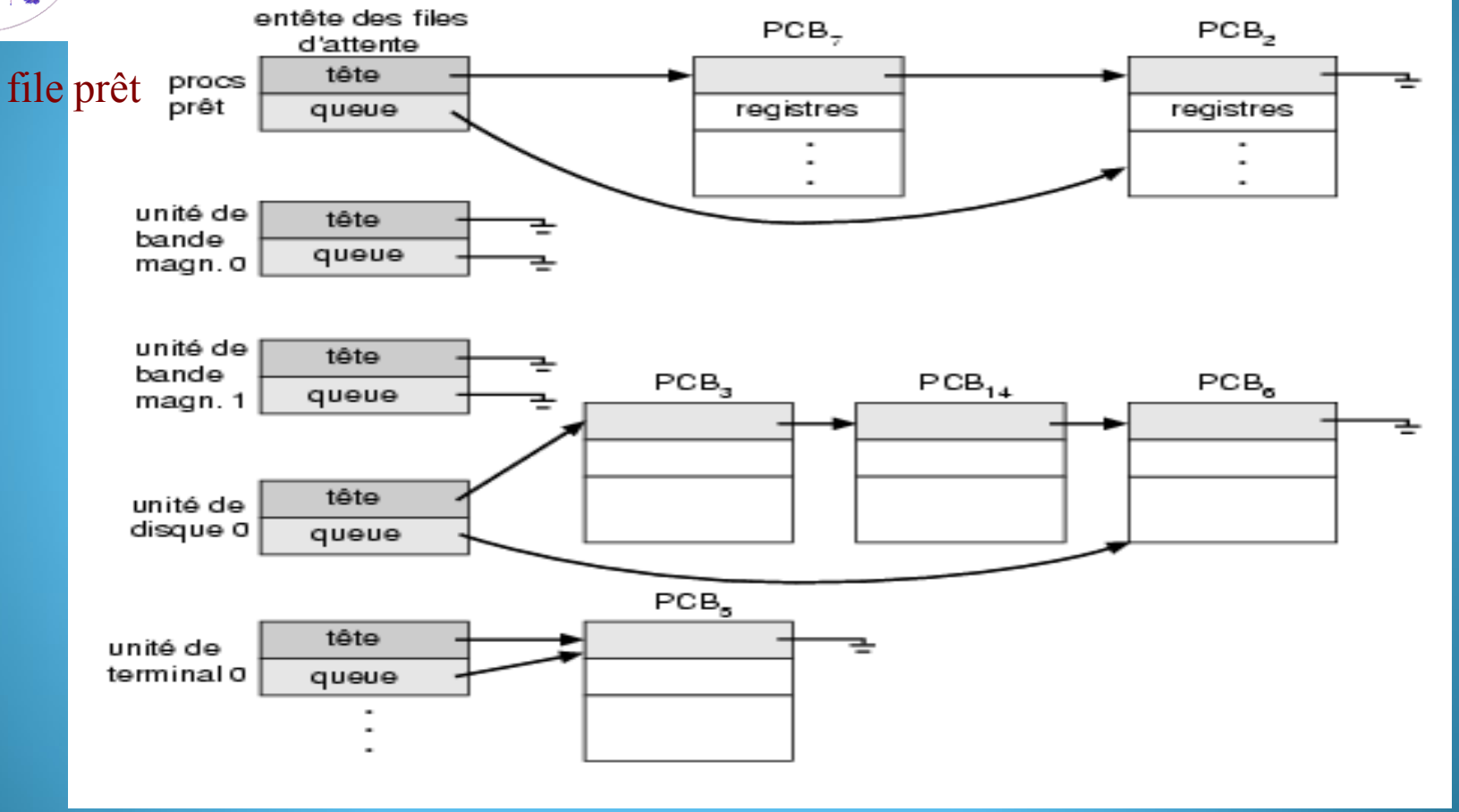


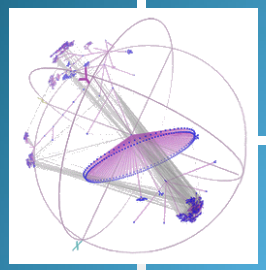
Files d'attente

- Les ressources d'ordinateur sont souvent limitées par rapport aux processus qui en demandent
- Chaque ressource a sa propre file de processus en attente
- En changeant d'état, les processus se déplacent d 'une file à l`autre
  - File prêt: les processus en état prêt=ready
  - Files associés à chaque unité E/S
  - etc.

Files d'attente

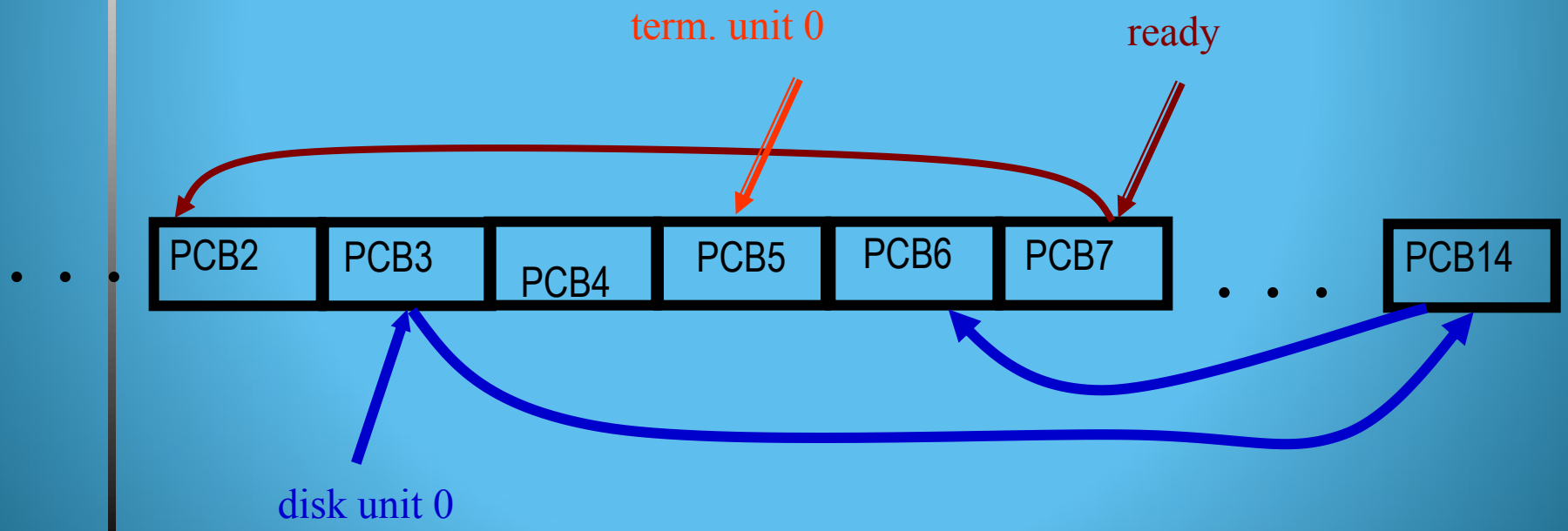
Ce sont les PCBs qui sont dans les files d'attente

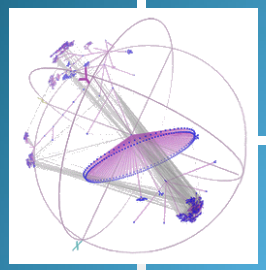




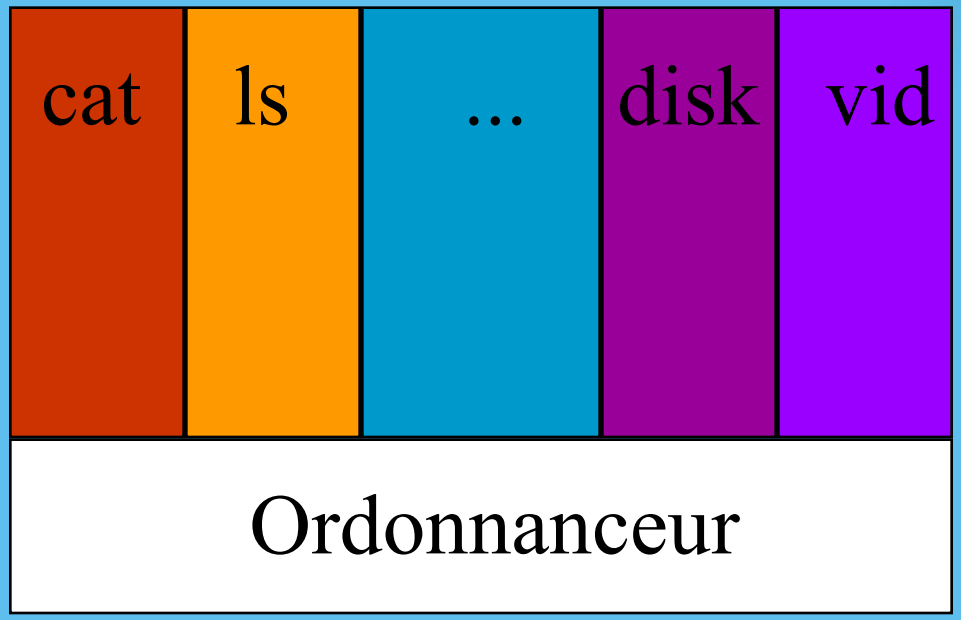
Les PCBs

Les PCBs ne sont pas déplacés en mémoire pour être mis dans les différentes files:
ce sont les pointeurs qui changent.

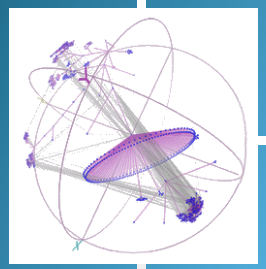




Ordonnanceur



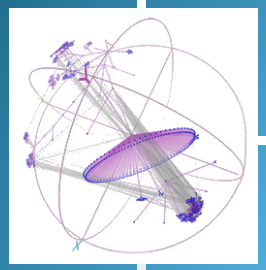
- Tous les services sont des processus



Ordonnanceurs (schedulers)

Programmes qui gèrent l'utilisation de ressources de l'ordinateur

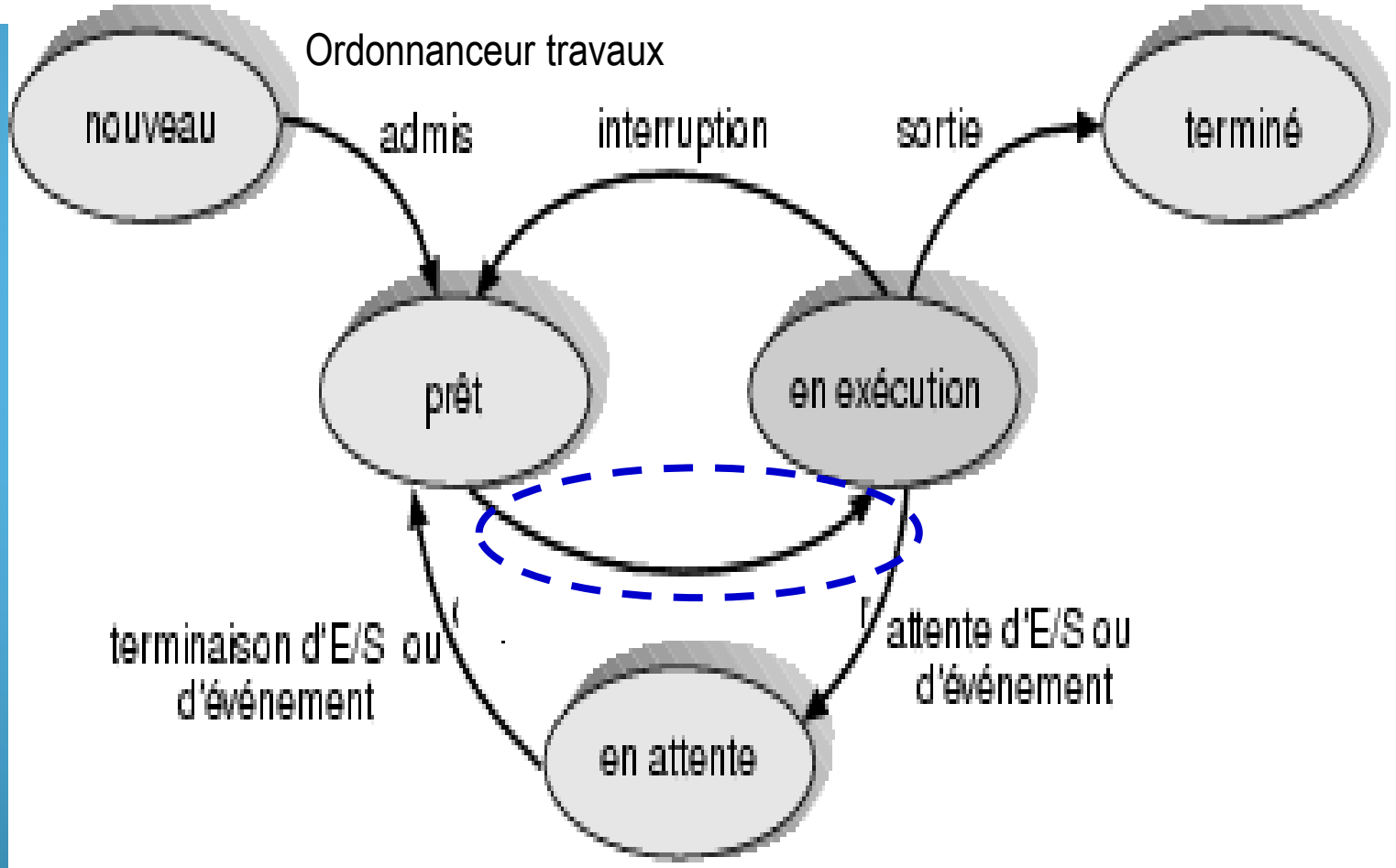
- Trois types d'ordonnanceurs :
  - À court terme = ordonnanceur processus: sélectionne quel processus doit exécuter la transition prêt → exécution
  - À long terme = ordonnanceur travaux: sélectionne quels processus peuvent exécuter la transition nouveau → prêt (événement *admitted*) (de spoule travaux à file prêt)
  - À moyen terme: répond au manque de mémoire



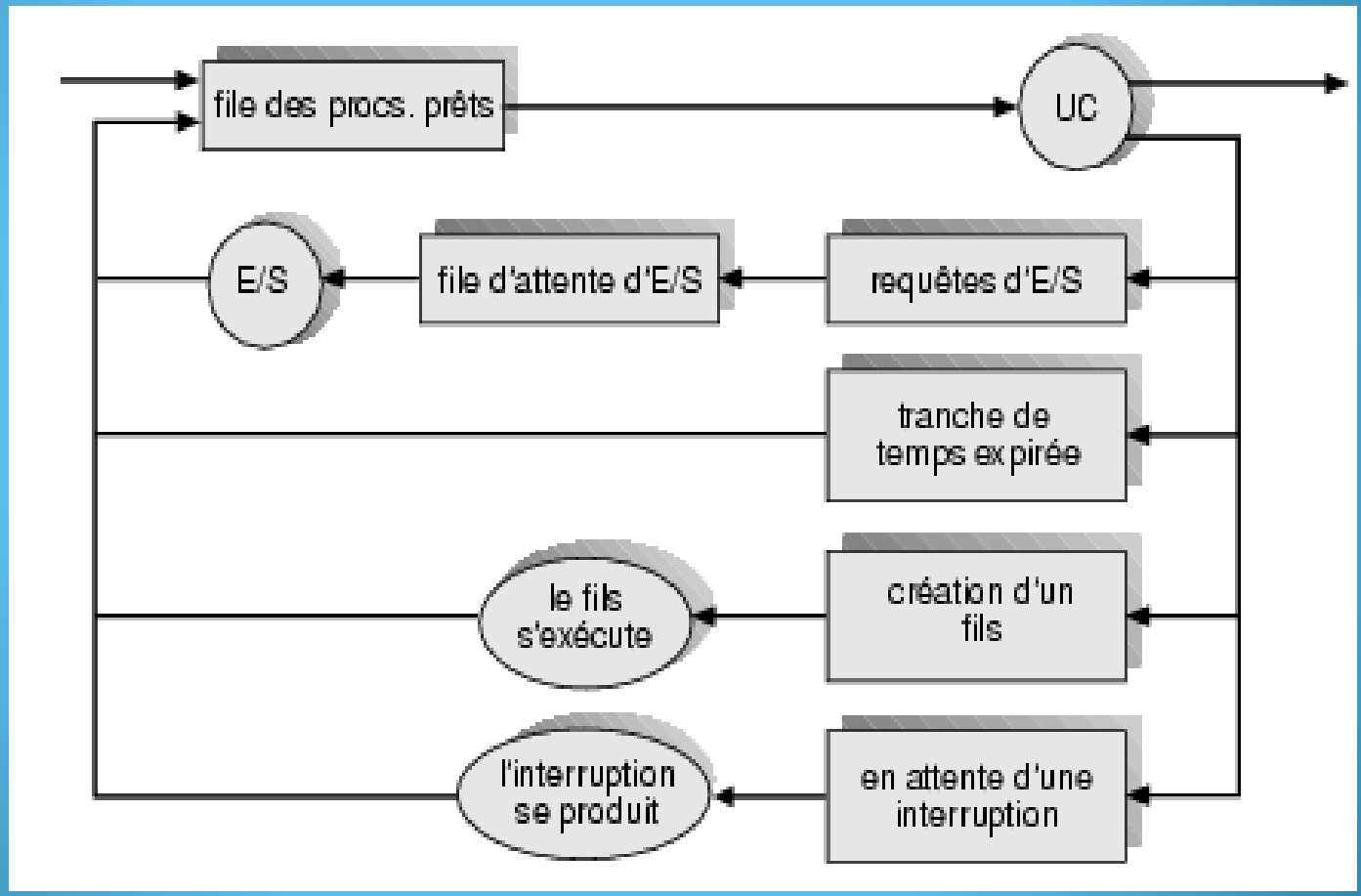
Ordonnanceurs

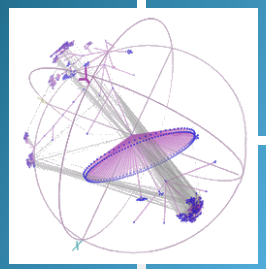
- L'ordonnanceur à court terme est exécuté très souvent (millisecondes), Il faut donc que ça aille très vite
  - typiquement de 1 à 1000 microsecondes
- L'ordonnanceur à long terme doit être exécuté beaucoup plus rarement: il contrôle le niveau de multiprogrammation
  - doit établir une équilibre entre les travaux liés à l'UCT et ceux liés à l'E/S de sorte à ce que les ressources de l'ordinateur soient bien utilisées

Ordonnanceur travaux = long terme
Ordonnanceur processus = court terme



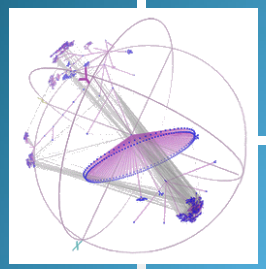
Ordonnancement de processus (court terme)





Ordonnanceur à moyen terme

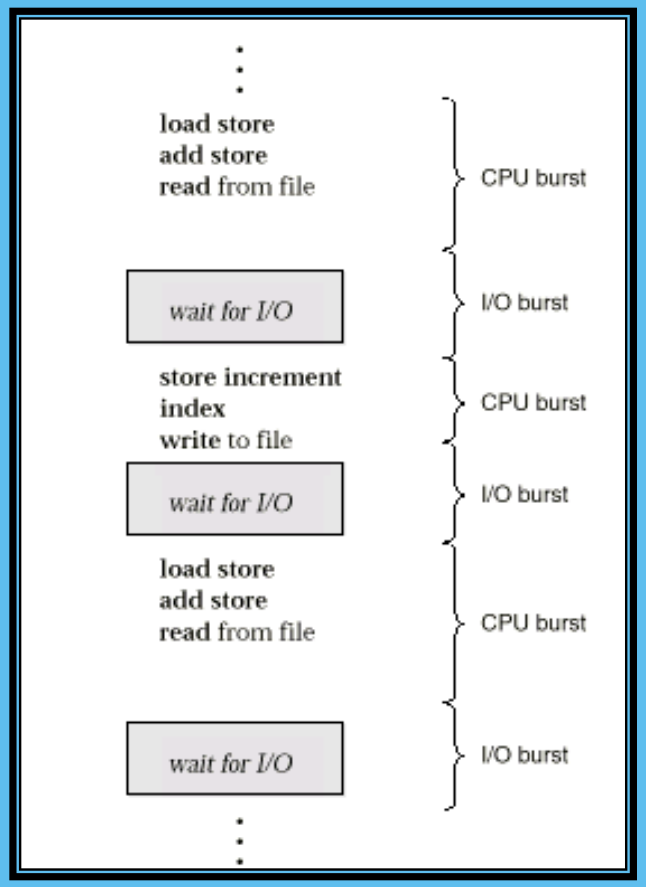
- Le manque de ressources peut parfois forcer le SE à *suspendre* des processus
  - ils seront plus en concurrence avec les autres pour des ressources
  - ils seront repris plus tard quand les ressources deviendront disponibles
- Ces processus sont enlevés de mémoire centrale et mis en mémoire secondaire, pour être repris plus tard
  - 'swap out', 'swap in' , va-et-vient



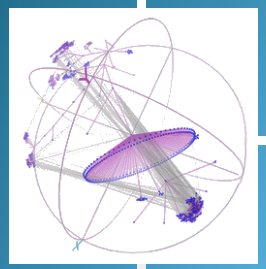
Concepts de base

- La multiprogrammation est conçue pour obtenir une utilisation maximale des ressources, surtout l'UCT
- L'ordonnanceur UCT est la partie du SE qui décide quel processus dans la file ready/prêt obtient l 'UCT quand elle devient libre
  - Un programme principalement E/S (interactif) a généralement de court pics d'activité
  - Un programme de calcul au contraire peut avoir des pics d'activité très longs.

Alternance CPU E/S



Cycles (bursts) d'UCT et E/S: l'exécution d'un processus consiste de séquences d'exécution sur UCT et d'attentes E/S



Quand invoquer l'ordonnanceur

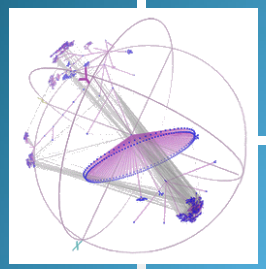
- Choisir un processus parmi ceux qui sont prêts et lui donner les ressources CPU.

- L'ordonnancement a lieu quand un processus :

1. Se termine.
2. Passe de l'état "actif" à "attente".
3. Passe de l'état "actif" à "prêt".
4. Passe de l'état "attente" à "prêt".

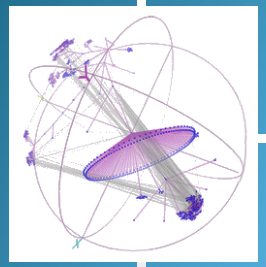
Un nouveau processus doit être choisi pour 1 et 4

Pour 2 et 3 : mode préemptif ou non préemptif.



Critères d'ordonnancement

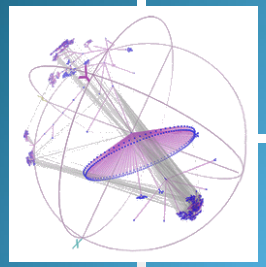
- Il y a normalement plusieurs processus dans la file des prêt
- Quand l'UCT devient disponible, lequel choisir?
- L'idée générale est d'effectuer le choix dans l'intérêt de l'efficacité d'utilisation de la machine
- Mais cette dernière peut être jugée selon différents critères…



Critères de qualité

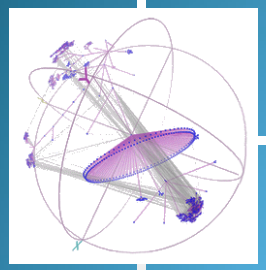
- Utilisation UCT: pourcentage d'utilisation
- Débit = Throughput: nombre de processus terminés dans une unité de temps
- Temps de rotation, temps de virement temps d'exécution, turnaround: le temps pris par le processus de son arrivée à sa terminaison.
  - inclus : temps passé dans tous les états, à faire des E/S...
- Temps d'attente: temps passé dans l'état "prêt" (somme de tout les temps passés dans la file des prêt)
- Temps de réponse : temps passé entre la création et le premier passage actif
  - Important pour l'interactivité

Mesures très liées.



Critères d'optimisation

- Exemples :
  - Maximiser l'utilisation et le débit
  - Minimiser le temps d'exécution, d'attente et de réponse
- Approches classiques :
  - En général : optimiser les valeurs moyennes.
  - Parfois : optimiser les valeurs min-max :
    - minimiser le temps maximal de réponse

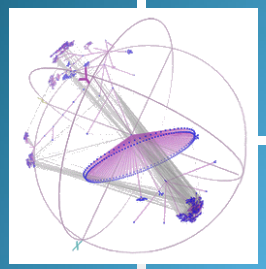


Ordonnancement PAPS (FIFO)

Premier-arrivé Premier-servi (First-in First-out)

- Les processus prêt sont stockés dans une FIFO et servis par ordre d'arrivée
- L'ordonnancement PAPS est non préemptif
  - Mauvais partage du temps

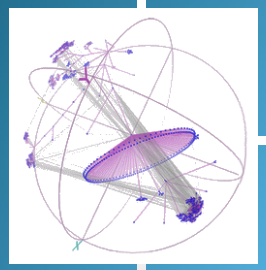
Processus	Temps d'exécution
P_1	24
P_2	3
P_3	3



Un exemple pour PAPS

- Ordre d'arrivée : P_1 , P_2 , P_3
- Diagramme de Gantt :

- Temps d'attente : $P_1 = 0$; $P_2 = 24$; $P_3 = 27$
- Temps moyen d'attente : (0+24+27)/3=17

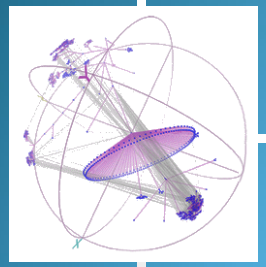


Un autre exemple

- Ordre d'arrivée : P_2 , P_3 , P_1
- Diagramme de Gantt :

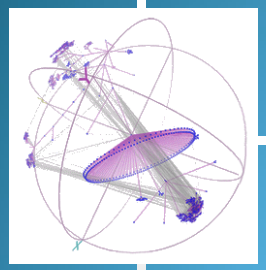
- Temps d'attente : $P_1 = 6$; $P_2 = 0$, $P_3 = 3$
- Temps moyen d'attente : (6+0+3)/3 = 3

Bien meilleur que dans le cas précédent.



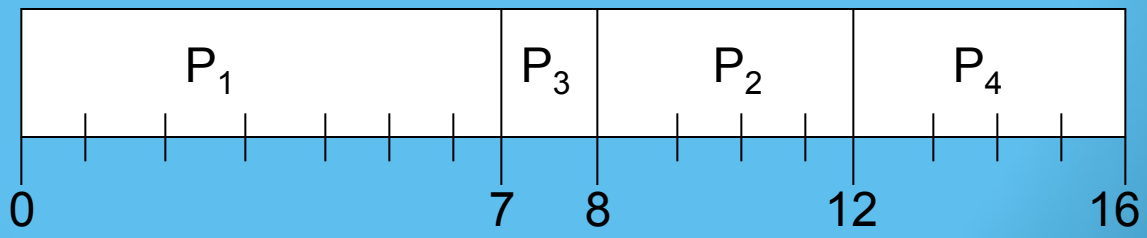
Plus court temps d'exécution (PCTE) Shortest job first (SJF)

- Le CPU est attribué au processus qui a le plus petit temps d'exécution (en utilisant PAPS en cas d'égalité)
- Deux approches :
  - Non préemptif (PCTE, SJF) : quand le CPU est accordé, il ne change pas jusqu'à la fin de son utilisation.
  - Préemptif (PCTER, SRTF) : si un nouveau processus arrive avec un temps d'exécution plus court que ce qui reste au processus courant il prend sa place
- PCTER : optimal pour le temps d'attente moyen.



Exemple pour PCTE

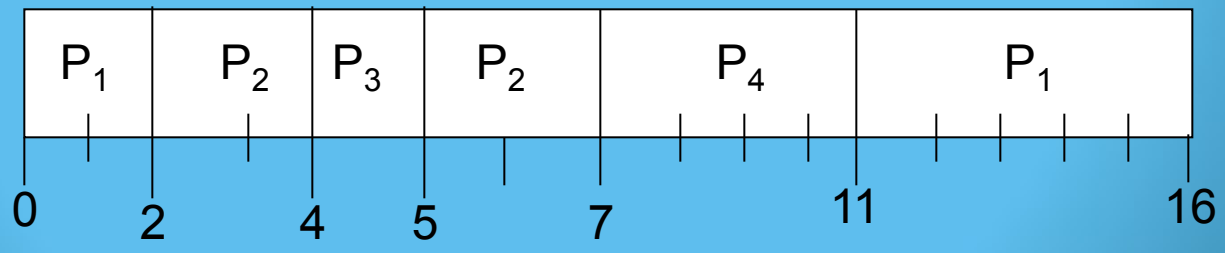
Processus	Arrivée	Durée
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4



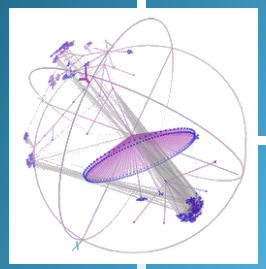
- Temps moyen d'attente = (0+6+3+7)/4=4

Exemple pour PCTER

Processus	Arrivée	Durée
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4



- Temps moyen d'attente = (9+1+0+2)/4=3



Prédire la durée d'utilisation

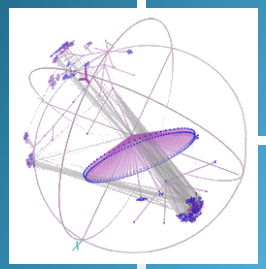
- Utilisation d'une moyenne des durées précédentes

1. t_n = durée du $n^{ième}$ pic de CPU
2. τ_{n+1} = durée prédite du prochain pic
3. $\alpha, 0 \le \alpha \le 1$

Alors :

$$\tau_{n+1} = \alpha\, t_n + (1-\alpha)\tau_n.$$

t_n : histoire la plus récente, τ_n : histoire passée



Moyennage exponentiel

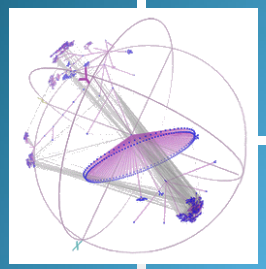
α=0: $\tau_{n+1}=\tau_n$: le passé récent ne compte pas.

α=1: $\tau_{n+1}=t_n$: seul le passé récent compte.

- En général, α =1/2
- τ_0 est une constante (moyenne du système)
- La formule peut être développée en :

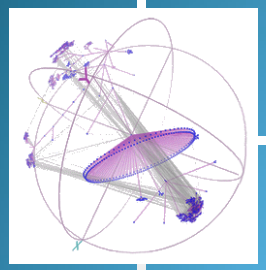
$$\tau_{n+1} = \alpha\, t_n + (1-\alpha)\,\alpha\, t_{n-1} + \ldots + (1-\alpha)^j\,\alpha\, t_{n-j} + \ldots + (1-\alpha)^{n+1}\,\tau_0$$

- Comme α et (1 - α) sont plus petits que 1, les termes ont de moins en moins de poids.



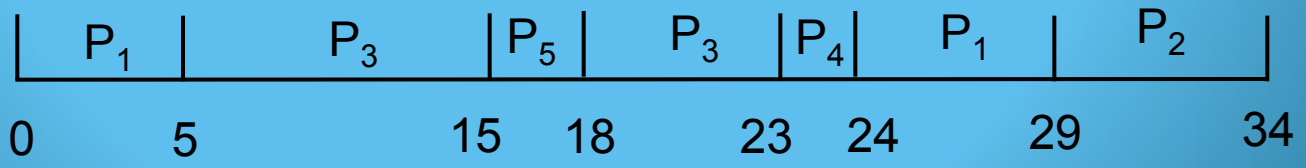
Ordonnancer avec priorités

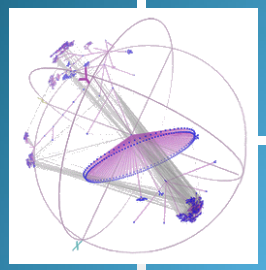
- Une priorité (entière) est associée à chaque processus
- Le CPU est donné au processus le plus prioritaire (avec PAPS si besoin)
  - Préemptif : si un nouveau processus plus prioritaire arrive, lui donner la main
  - Non préemptif : le processus courant termine son cycle
- PCTE est un algorithme à priorité.
- Risque de famine : augmenter la priorité avec l'âge: technique de vieillissement.



Priorité + préemption

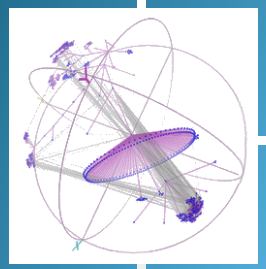
Processus	Durée	Arrivée	Priorité
P_1	10	0	3
P_2	5	2	7
P_3	15	5	2
P_4	1	10	2
P_5	*3*	*15*	*1*





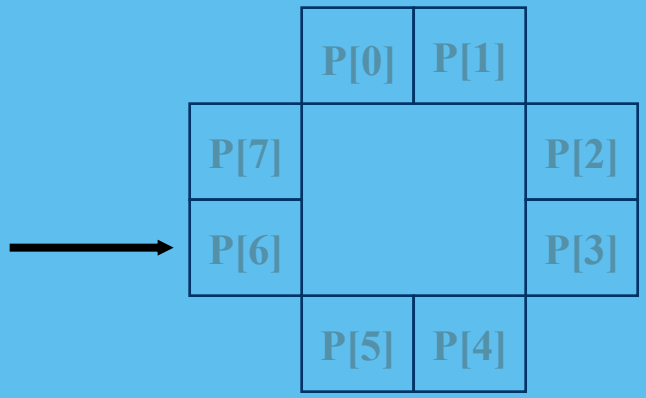
Tourniquet (Round Robin)

- Initialement pour les systèmes à temps partagé.
- Chaque processus obtient un peu de temps CPU (un *quantum*), typiquement de 10-100 millisecondes puis est préempté et devient "prêt".
  - Similaire à PAPS avec préemption
- S'il y a *n* processus prêt avec un quantum *q*, alors le temps de réponse est au maximum (*n*-1)*q*

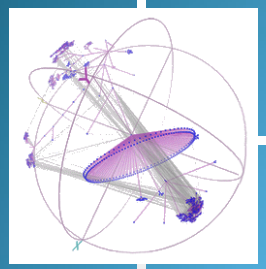


Tourniquet (Round Robin)

- S'il s'exécute pour un quantum entier sans autres interruptions, il est interrompu par la minuterie et l'UCT est donnée à un autre processus
- Le processus interrompu redevient prêt (à la fin de la file)

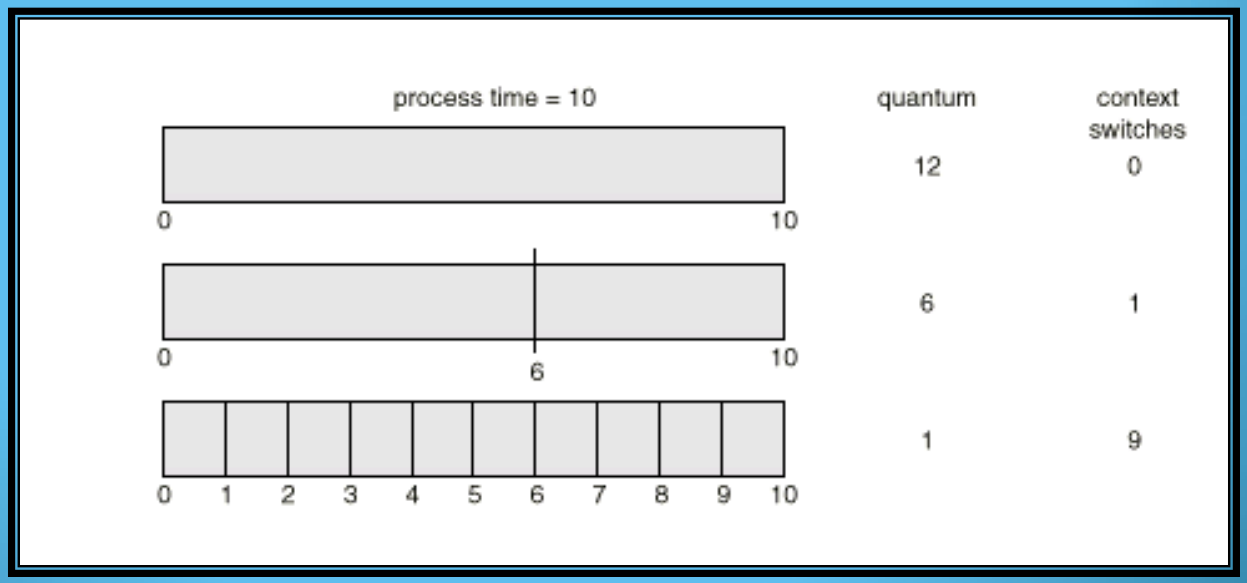


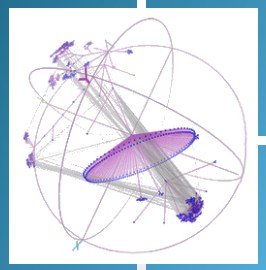
La file prêt est un cercle (dont RR)



Changements de contexte

- La performance dépend de q
  - *q* grand : similaire à PAPS
  - Si q est petit, le temps de réponse diminue mais il y a plus de changements de contexte. q doit être grand par rapport à la durée d'un changement de contexte

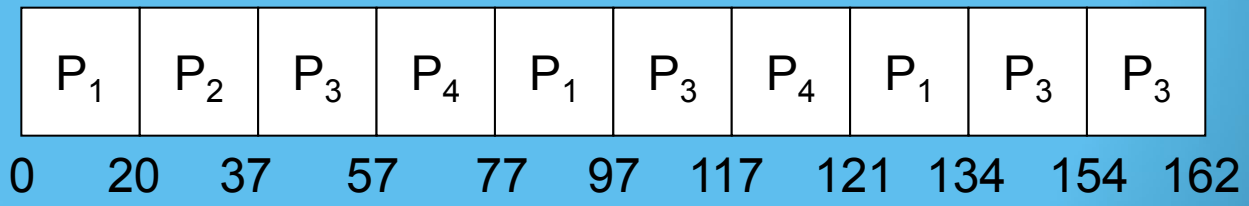




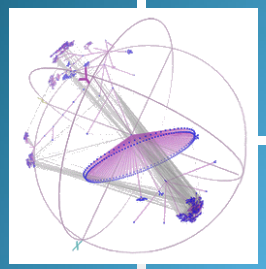
Tourniquet avec q = 20

Processus	Durée
P_1	53
P_2	17
P_3	68
P_4	24

- Le diagramme de Gantt est :

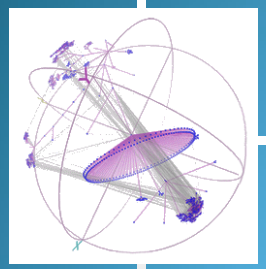


- Typiquement temps d'exécution supérieur à PCTE mais meilleur temps de réponse.

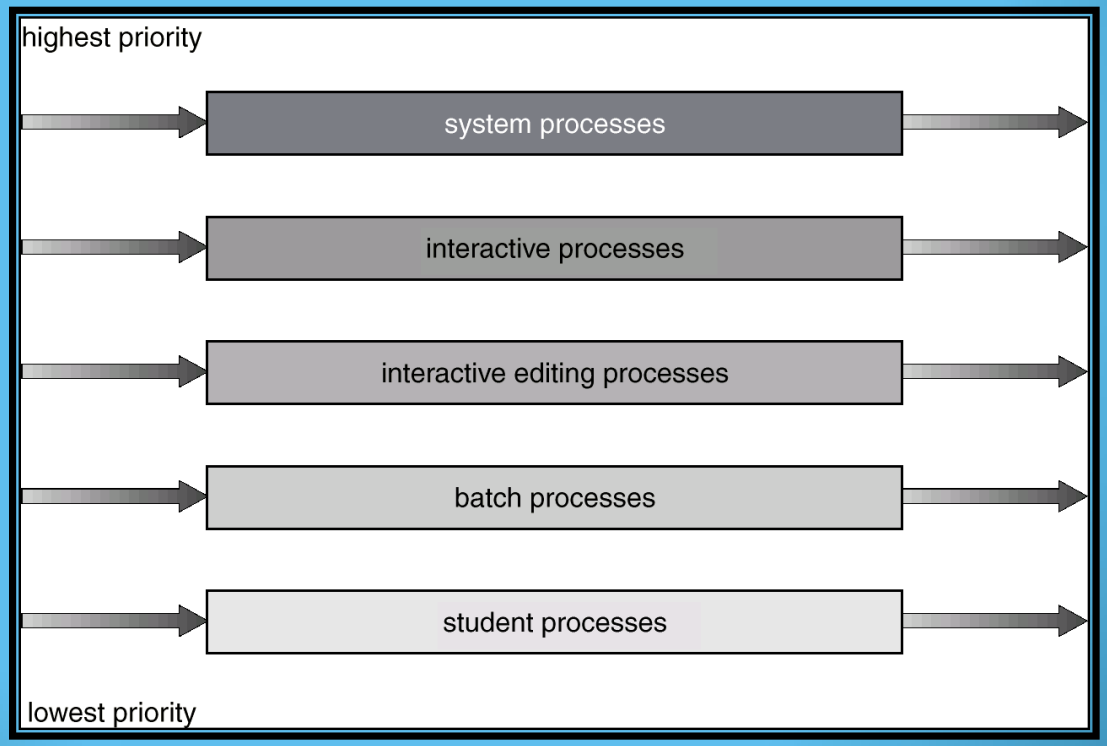


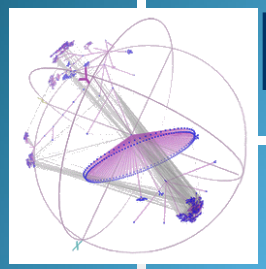
Ordonnancement multi-files

- La file pour les “prêts” est séparée en plusieurs files
  - distinguer différents types de processus.
- Un algorithme pour chaque file
- L’ordonnancement obligatoire entre les files
  - Une file prioritaire avec préemption : risque de famine.
  - Partage du temps : chaque file obtient une fraction du temps qu’elle gère comme elle le souhaite



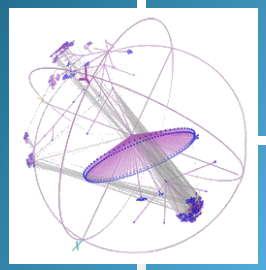
Exemple d'ordonnancement





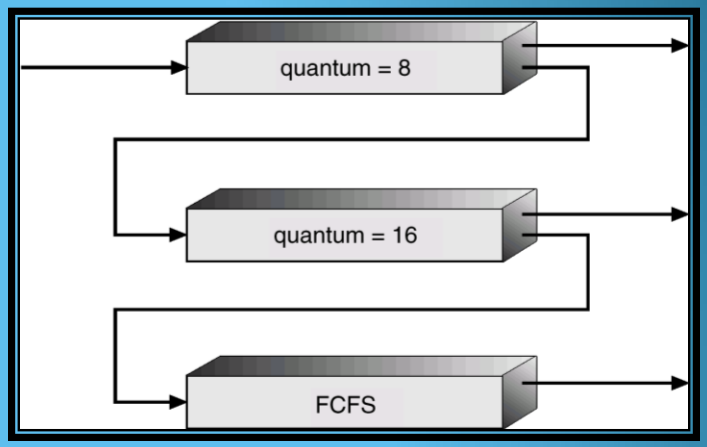
Plusieurs files avec feedback

- Un processus peut changer de file
  - Gestion de l'âge par exemple.
- Ce type d'ordonnanceur est défini par :
  - nombre de files,
  - ordonnanceur pour chaque file
  - règles pour changer un processus de file (vers le haut ou vers le bas)
  - règles pour décider de la file initiale d'un processus
- Algorithme d'ordonnancement le plus complexe et le plus général

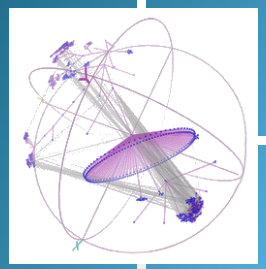


Exemple

- Trois files :
  - Q_0 – tourniquet, q=8 ms
  - Q_1 – tourniquet, q=16 ms
  - Q_2 – PAPS



- Ordonnancement
  - Arrivée dans Q_0 avec une règle PAPS, puis tourniquet de 8 secondes. S'il n'a pas terminé en 8 millisecondes il passe dans Q_1.
  - Dans Q_1, la règle est toujours PAPS et le processus obtient 16 millisecondes. S'il n'a toujours pas terminé, il est préempté et passe dans Q_2.
  - Règles inter files...



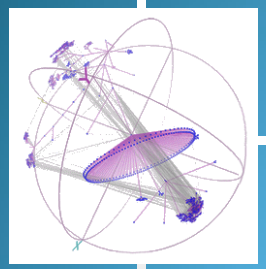
Évaluation des algorithmes

- Comment choisir un algorithme ?

1. Choix des critères (utilisation du CPU maximale avec temps de réponse < 1s)
2. Évaluer/comparer les algorithmes

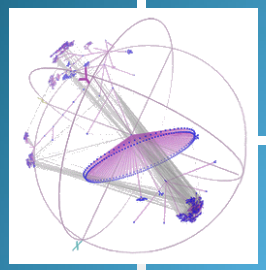
- Méthodes d'évaluation

1. Modélisation déterministe (scénario)
2. Modélisation des files d'attente
3. Simulation (scénario aléatoire ou rejoué)
4. Implémentation dans le SE (très coûteux)



En pratique...

- Les méthodes que nous avons vu sont toutes utilisées en pratique (sauf plus court servi *pur* qui est impossible)
- Les SE sophistiqués fournissent au gérant du système une librairie de méthodes, qu'il peut choisir et combiner au besoin après avoir observé le comportement du système
- Pour chaque méthode, plusieurs paramètres sont disponibles: p.ex. durée du quantum, etc.



Ordonnancement Linux

- Ordonnancement temps partagé
  - Seuls les processus en mode utilisateur et non en mode superviseur peuvent être préemptés
  - priorité avec des crédits (pour celui qui en a le plus)
  - Quand le quantum expire, perte de 1 crédit (vieillissement)
  - Un processus sans crédit est suspendu
  - Si tous les processus prêt sont suspendus, tous les processus en regagnent
  - Les processus en arrière plan on une priorité plus faible et reçoivent donc moins de crédits.

  $\Rightarrow$ Processus interactifs accumulent plus de crédits