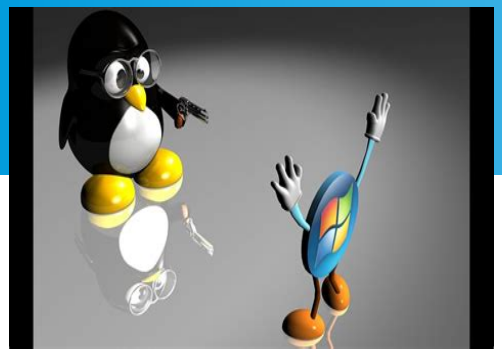
Notion de processus

Attention, il ne faut pas confondre le code source du programme et un processus, qui, lui, correspond à l'exécution de ce programme par un ordinateur.

Si une recette de cuisine correspond au code source du programme, le cuisinier en train de préparer cette recette dans sa cuisine correspond à un processus.



États d'un processus

Tous les systèmes d'exploitation "modernes" (Linux, Windows, macOS, Android, iOS...) sont capables de gérer l'exécution de plusieurs processus « en même temps ».

"En même temps", signifie plutôt "chacun son tour".

Pour gérer ce "chacun son tour", les systèmes d'exploitation attribuent des "états" au processus.

États d'un processus

Voici les différents états :

- Élu : lorsqu'un processus est en train de s'exécuter (qu'il utilise le microprocesseur), on dit que le processus est dans l'état "élu".
- Bloqué : Un processus qui se trouve dans l'état élu peut demander à accéder à une ressource pas forcément disponible instantanément (par exemple lire une donnée sur le disque dur). Le processus ne peut pas poursuivre son exécution. En attendant de recevoir la ressource, il passe de l'état "élu" à l'état "bloqué".

États d'un processus

- Prêt : Lorsque le processus finit par obtenir la ressource attendue, celui-ci peut potentiellement reprendre son exécution. Mais un seul processus peut se trouver dans un état "élu" (le microprocesseur ne peut "s'occuper" que d'un seul processus à la fois). Il ne peut donc repasser à l'état « élu » : il passe donc à l'état "prêt" (il est en attente de la reprise de son exécution).

États d'un processus

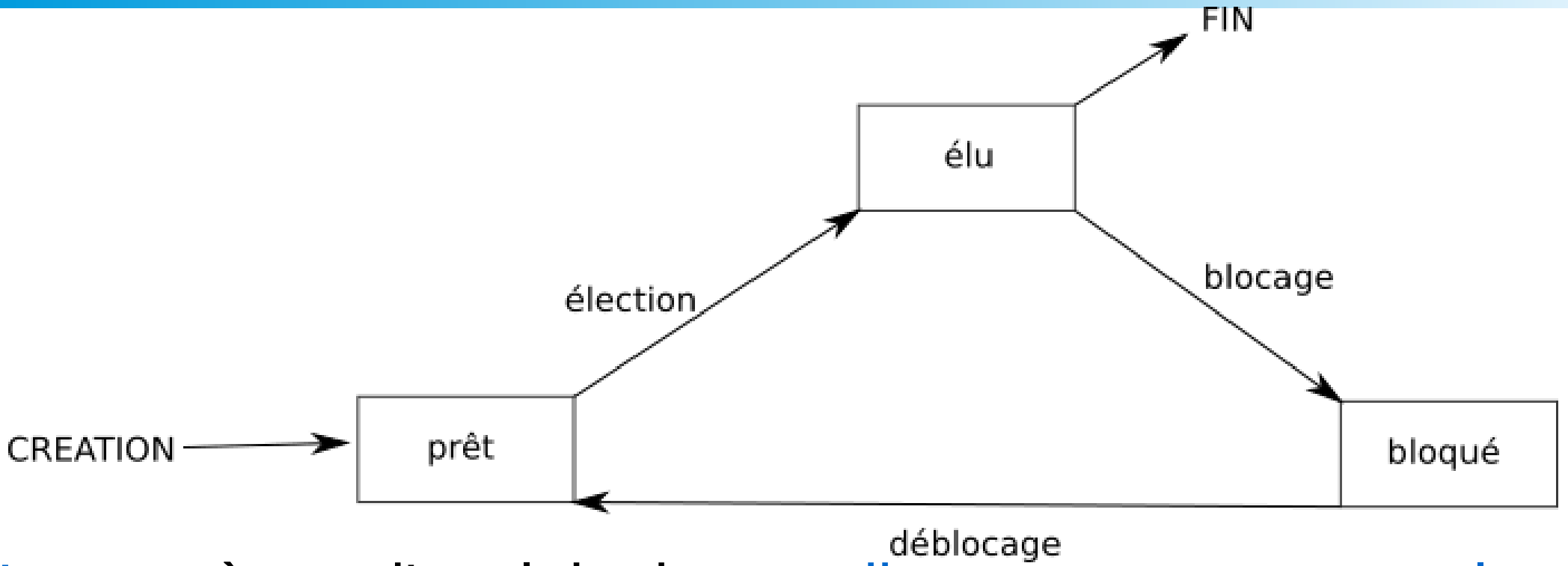
Vocabulaire :

- Le passage de l'état "prêt" vers l'état "élu" constitue l'opération "d'élection".
- Le passage de l'état élu vers l'état bloqué est l'opération de "blocage".

Création d'un processus : un processus est toujours créé dans l'état "prêt".

Terminaison d'un processus : pour se terminer, un processus doit obligatoirement se trouver dans l'état "élu".

Résumé



Le système d'exploitation attribue aux processus leur état "élu", "bloqué" ou "prêt". On dit que le système gère l'ordonnancement des processus (tel processus sera prioritaire sur tel autre...)

La place des « ressources »

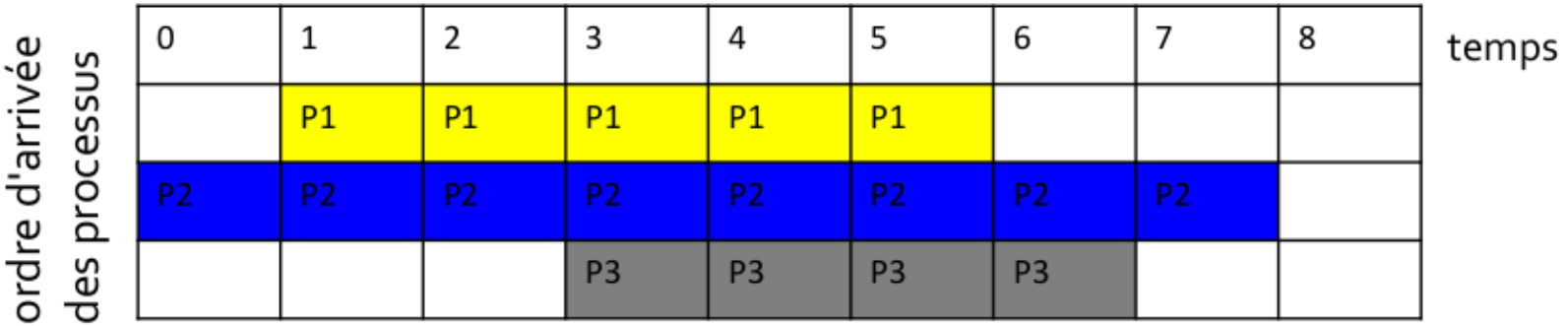
Un processus qui utilise une ressource doit la "libérer" une fois qu'il a fini de l'utiliser afin de la rendre disponible pour les autres processus.

Pour libérer une ressource, un processus doit obligatoirement être dans un état "élu".

Ordonnancement

Pour illustrer les différents algorithmes d'ordonnancement de tâches nous considérerons la situation simplifiée de trois processus P1, P2 et P3 **de même priorité** dont le moment d'arrivée est donné par l'échelle de temps et de durée indiquée par le découpage en cycles ou quantum de temps pleins (nombre entier de carrés) :

Processus	Date d'arrivée	Temps d'exécution
P1	T1	5
P2	To	8
P3	T3	4



L'algorithme FCFS (First Comme First Serve : premier entré, premier sorti).
Cet algorithme utilise une file (FIFO). Les processus sont traités par ordre d'arrivé sans tri particulier.

Exemple :

Processus	Date d'arrivée	Temps d'exécution
P1	T1	5
P2	To	8
P3	T3	4

Dans ce cas, c'est P2 qui arrive en premier et qui est exécuté entièrement puis viendra ensuite le tour de P1 qui est arrivé en second puis P3

Temps	To	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16
Processus	P2	P2	P2	P2	P2	P2	P2	P2	P1	P1	P1	P1	P1	P3	P3	P3	P3

Processus	Date d'arrivée	Temps d'exécution
P1	To	5
P2	To	8
P3	To	4

Dans le cas ou les processus arrivent en même temps dans la file, s'il n'y a pas de consigne de priorité alors l'ordre d'attente correspond au numéro d'attribution du processus.

Par exemple dans le tableau, P1, P2 et P3 ont une date d'arrivée identique alors l'ordre se fait en fonction du numéro (PID) donc, P1 avant P2 avant P3.
L'exécution se déroule ensuite pour chaque processus durant son temps d'exécution.

L'algorithme SJF (Shortest Job First : plus court d'abord).

Cet algorithme consiste à sélectionner le processus ayant le moins de temps d'exécution (le temps d'utilisation du CPU le plus faible).

Si un nouveau processus arrive dont sa durée d'utilisation du processeur est inférieure au temps restant d'exécution du processus en cours, alors ce nouveau processus obtient le contrôle du processeur

Exemple avec l'algorithme SJF

Processus	Date d'arrivée	Temps d'exécution
P1	T1	5
P2	To	8
P3	T3	4

On voit que P2 arrive en To, il est le seul processus dans le système , il passe à l'état élu et utilise la ressource processeur.

Ensuite arrive P1 en T1 pour une durée de 5UT puis P3 en T3 pour une durée de 4UT. Par conséquent, pendant que P2 s'exécute sur le processeur, nous avons en même temps dans la file d'attente P1 et P3. P3 ayant moins d'unité de temps que P1, alors il y a un tri et nous avons P3 puis P1 dans la file d'attente.

Lorsque P2 aura fini c'est donc P3 qui est éligible avant P1.

Temps	To	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16
Processus	P2	P2	P2	P2	P2	P2	P2	P2	P3	P3	P3	P3	P1	P1	P1	P1	P1

il existe une autre version de cet algorithme (**SRTF : Shortest Remaining Time First**) qui est plus complexe. Si un nouveau processus arrive dont sa durée d'utilisation du processeur est inférieure au temps restant d'exécution du processus en cours, alors ce nouveau processus obtient le contrôle du processeur.

Processus	Date d'arrivée	Temps d'exécution
P1	T1	5
P2	To	8
P3	T3	4

En reprenant l'exemple ci-dessus et avec et algorithme : P2 commencerait mais serait interrompu par P1. En effet, P1 arrive en T1 avec 5 UT. Comme cela fait une unité de temps que P2 est en train d'être exécuté, il lui reste 7 UT à faire.

Or 5 UT < 7UT, il y a donc un changement de contexte entre P2 et P1, et donc P1 prend la place de P2 dans le processeur.

P3 arrive en T3 avec 4 UT et nous avons un calcul à faire à savoir, combien reste-t-il d'UT pour P1 quand P3 arrive. Cela fait 2UT que P1 s'exécute, donc il reste 3UT à P1, et comme 3UT < 4UT alors P1 reste en cours d'exécution et P3 rentre en file d'attente. Par conséquent, pendant que P1 s'exécute (et termine son exécution, car il a une UT plus petite et qu'il n'y a plus de nouveau processus en approche) sur le processeur, nous avons en même temps dans la file d'attente P2 et P3. P3 ayant moins d'unité de temps que P2, 4 UT < 7 UT, alors il y a un tri et nous avons P3 puis P2 dans la file d'attente.
Lorsque P1 aura fini c'est donc P3 qui est éligible avant P2.

Temps	To	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16
Processus	P2	P1	P1	P1	P1	P1	P3	P3	P3	P3	P2	P2	P2	P2	P2	P2	P2

L'algorithme RR (Round-Robin : Le tourniquet).

Cet algorithme consiste à allouer à tour de rôle et à chaque processus un quota de temps processeur. Ainsi un processus en cours d'exécution peut être interrompu si le quota de temps alloué est écoulé. Dans ce cas, le processus est remis dans la queue des processus de l'état prêt.

Exemple :

Processus	Date d'arrivée	Temps d'exécution	Quota
P1	To	5	2
P2	To	8	2
P3	To	4	2

P1, P2 et P3 arrivent en To. Selon les identifiants des processus, nous avons P1 avant P2 avant P3. Le quantum de temps est de 2 UT, donc tous les 2UT il y aura un changement de contexte.

La répartition des processus à l'état élu se fera selon le tableau ci-dessous en respectant les quota et les temps d'exécution de chaque processus.
On remarque qu'à T12, il ne reste plus qu'1 UT à P1 et 4UT à P2 (P3 ayant terminé). Comme les temps sont décomptés à chaque T, le 1 UT qui restait à P1 est alloué à P2. Puis P2 étant le seul processus restant, il prend les 2UT de quota pour les tours suivants.

Temps	To	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16
Processus	P1	P1	P2	P2	P3	P3	P1	P1	P2	P2	P3	P3	P1	P2	P2	P2	P2

Avec les algorithmes précédents chacun possèdent des avantages et des inconvénients.
Le dernier algorithme par exemple abouti à des temps de séjour des processus dans l'état prêt plus long qu'avec les autres algorithmes.
En revanche, cet algorithme permet que tous les processus s'exécutent alors qu'avec les autres algorithmes, il se peut que certains processus n'obtiennent jamais la ou les ressources dont il ont besoin pour être exécutés (on parle de famine)

Pour finir, nous avons fait l'hypothèse que les processus avaient un ordre en fonction de leur identification (PID). Mais il se peut qu'un **processus soit prioritaire** par rapport aux autres processus, ainsi il intégrera les files d'attente différemment. Il existe **des algorithmes de planification** avec priorité permettant de gérer ce type de processus que nous ne verrons pas dans ce cours.

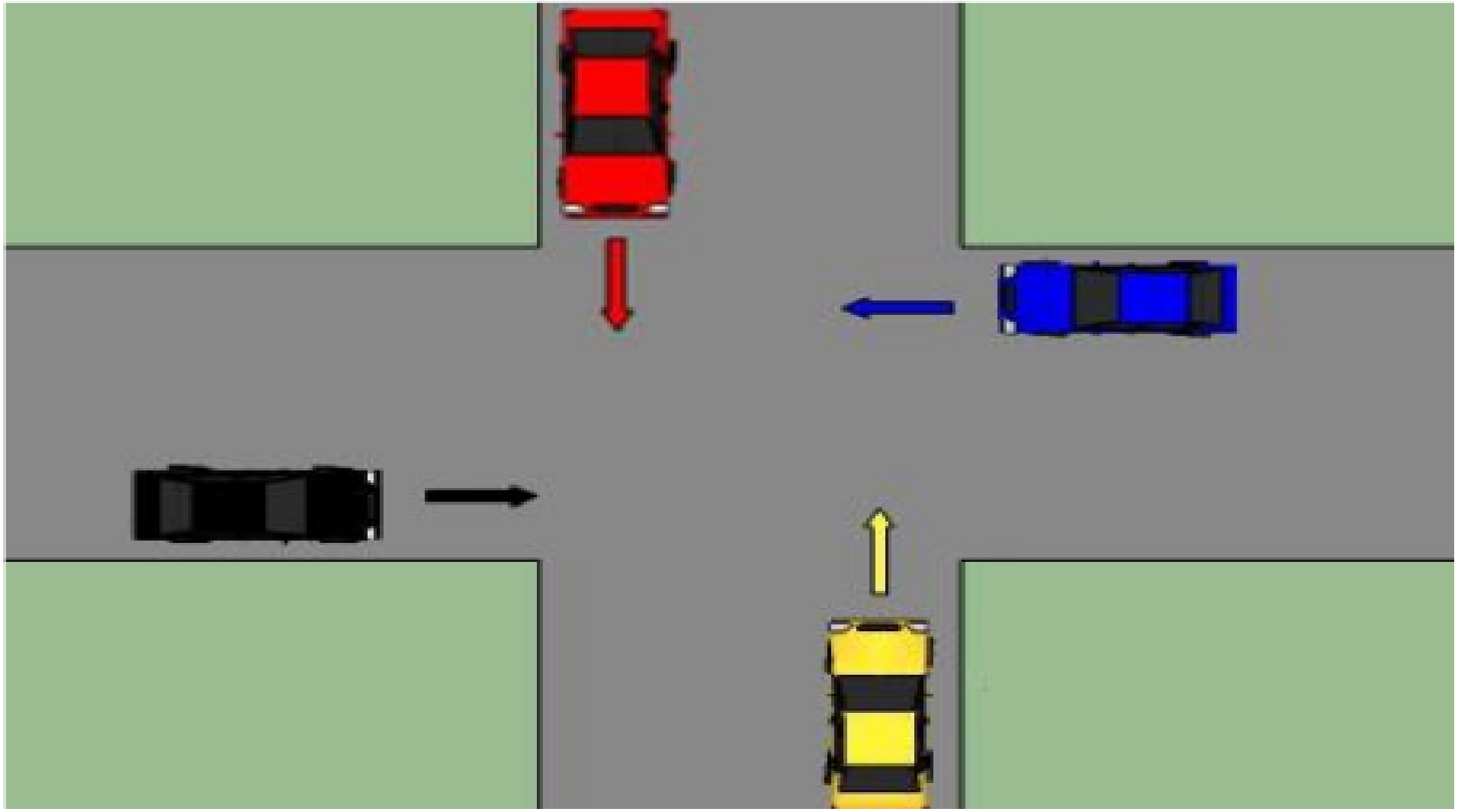
Interblocage (deadlock)

Deux processus P1 et P2, deux ressources : R1 et R2.

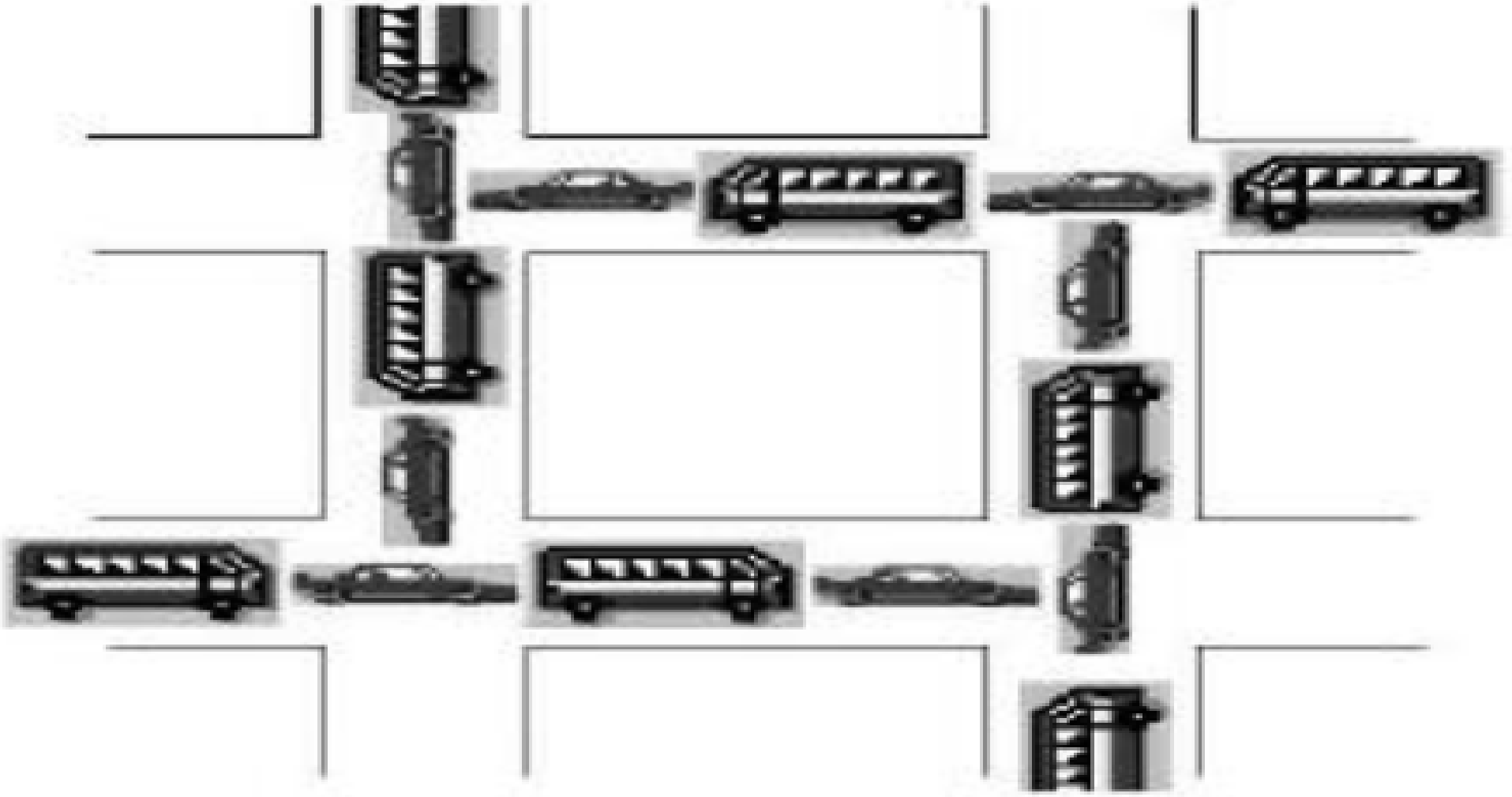
- P1 actif,
  - il demande R1, se bloque
  - et passe à l'état « prêt » quand il obtient R1
- Plus tard, P2 devient actif,
  - il demande R2, se bloque
  - et passe donc à l'état « prêt » quand il obtient R2
- Plus tard, P1 redevient actif et demande R2

Identifier le problème.

Interblocages – vie courante



Interblocages – vie courante



Dessiner l'interblocage

On peut modéliser la distribution des ressources (on dit l'allocation) pour les processus par un **graphe**.
On indique que les nœuds (les sommets) représentent les processus (cercle sur le schéma (diapositive suivante)) ou les ressources (en rectangle)
Les arcs orientés d'une **ressource vers un processus** signifient que la **ressource est allouée au processus**.
Les arcs orientés d'un **processus vers une ressource** signifient que le **processus est bloqué en attente de la ressource**.

Supposons trois processus A, B et C qui utilisent trois ressources R, S et T

A	B	C
Demande R Demande S Libère R Libère S	Demande S Demande T Libère S Libère T	Demande T Demande R Libère T Libère R

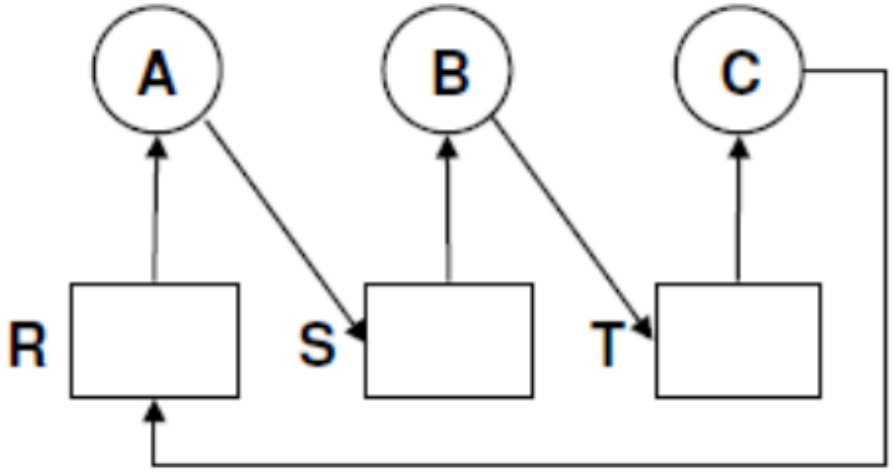
Dessiner l'interblocage

Supposons trois processus A, B et C qui utilisent trois ressources R, S et T

A	B	C
Demande R Demande S Libère R Libère S	Demande S Demande T Libère S Libère T	Demande T Demande R Libère T Libère R

Si chaque processus sont exécutés de façon séquentielle : A suivi de B suivi de C, il n' y a pas interblocage.
En revanche si l'ordonnanceur gère l'exécution des processus de la manière suivante :
A demande R et l'obtient
B demande S et l'obtient
C demande T et l'obtient
A demande S
B demande T
C demande R
Il y a interblocage.

Dessiner l'interblocage



Qu'observe-t-on ?

On observe un **cycle**.

D'un point de vue algorithmique, si on construit le graphe des ressources pour les différents processus, la **présence d'un cycle dans ce graphe indiquera une situation d'interblocage.**

Nous verrons dans la partie algorithmique le moyen de détecter des cycles dans les graphes.

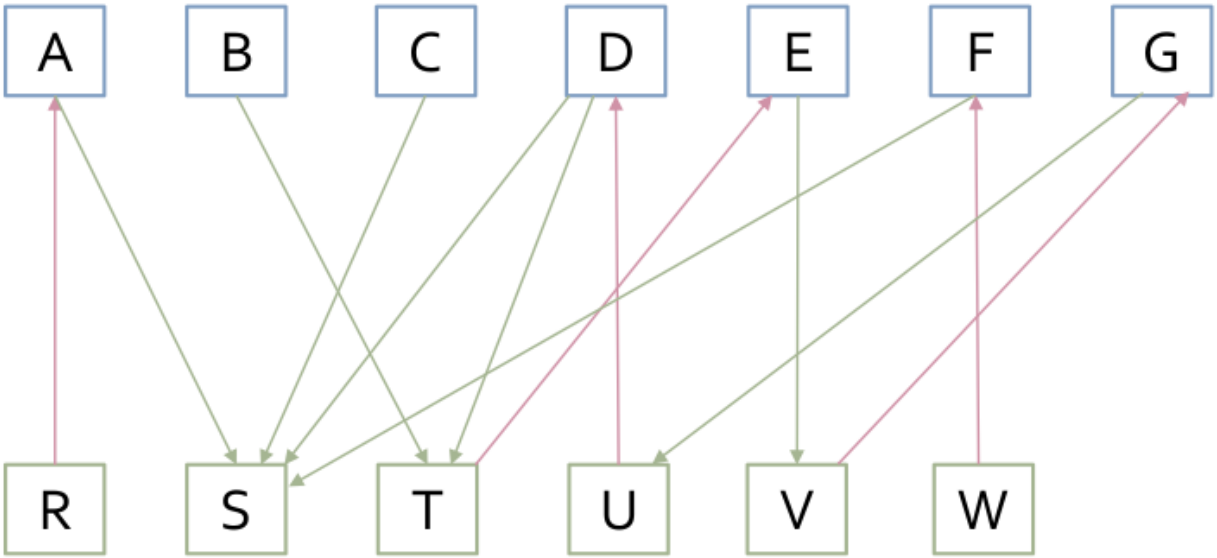
Exercice

Considérons un système ayant sept processus, A à G, et six ressources R à W. L'attribution des ressources est la suivante :

. A détient R et demande S ;
. B demande T ;
. C demande S ;
. D détient U et demande S et T ;
. E détient T et demande V;
. F détient W et demande S ;
. G détient V et demande U.

Construire le graphe d'allocation des ressources ? Y a-t-il un interblocage ?
Si oui, quels sont les processus concernés ?
Utiliser la schématisation précédente.

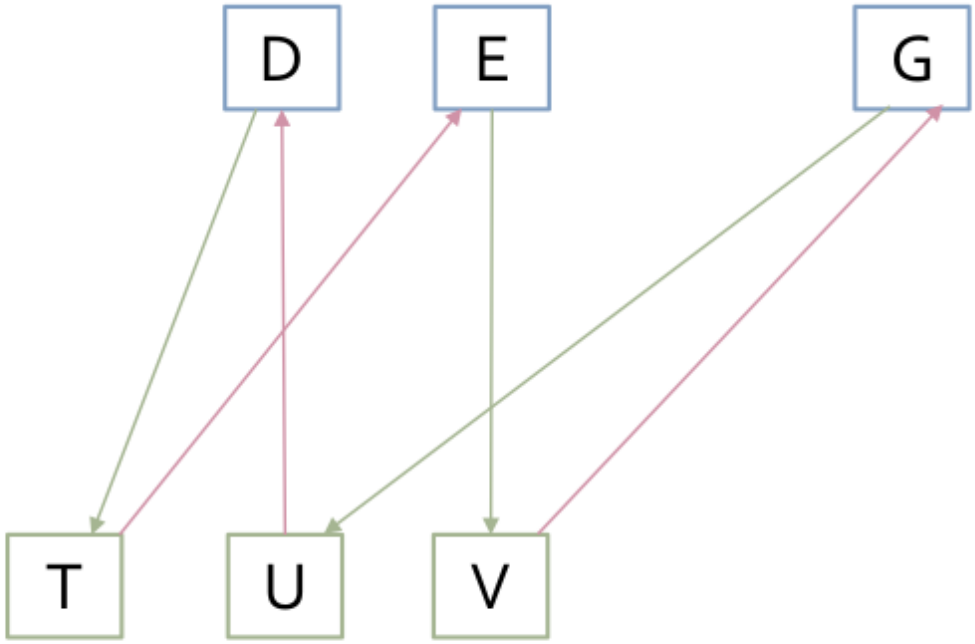
Correction - graphe



Si il y a un interblocage, **on doit trouver un cycle dans ce graphe**.
Je peux donc éliminer tous les processus qui soit :

- demandent une ressource mais n'en possèdent pas (que des flèches sortantes).
- possèdent une ressource mais n'en demandent pas (que des flèches entrantes).
- possèdent une ressource qui n'est pas demandé par un autre processus.
- demandent une ressource qu'aucun processus ne possède

Simplification graphe



Il reste alors ces processus.
Il forme bien un cycle, si vous n'en êtes pas convaincu, regardez la diapositive suivante.

Simplifications - suite