Systèmes d'exploitation

Interaction de processus

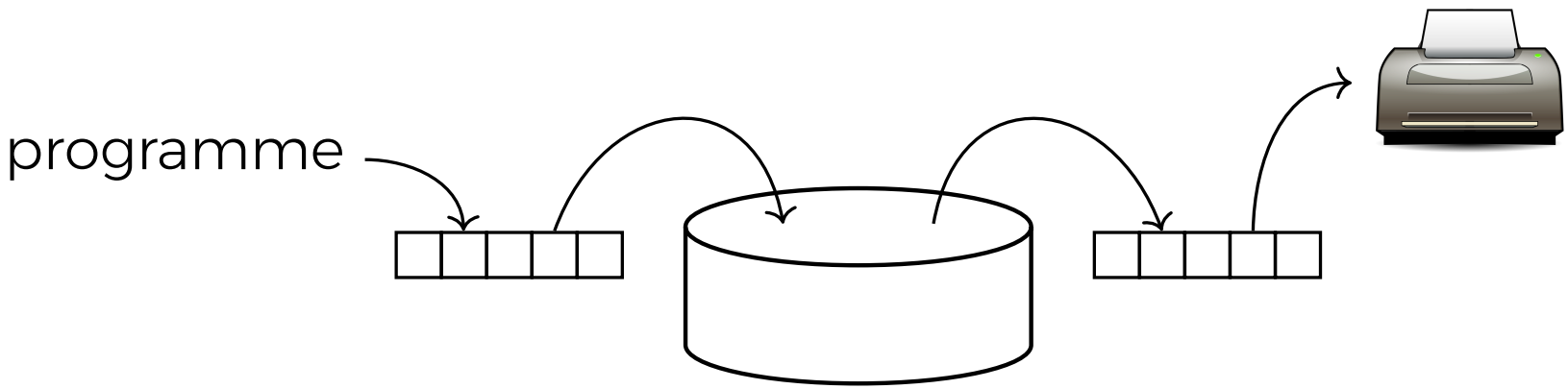
Sergiu Ivanov

sergiu.ivanov@univ-evry.fr

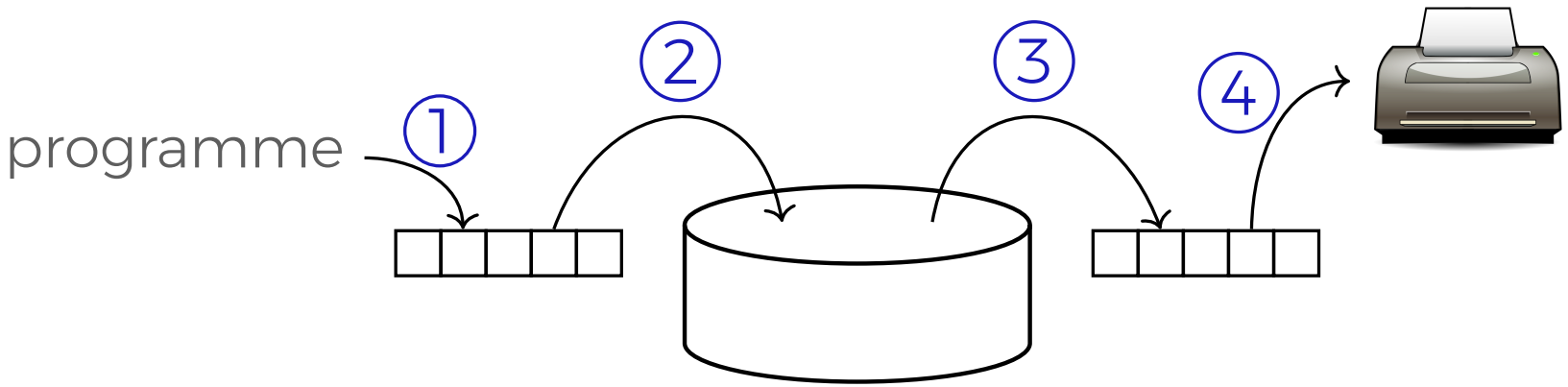
https://www.ibisc.univ-evry.fr/~sivanov/fr/os-ueve.html

Comment les processus interagissent-ils ?

Combien y a-t-il de processus ?



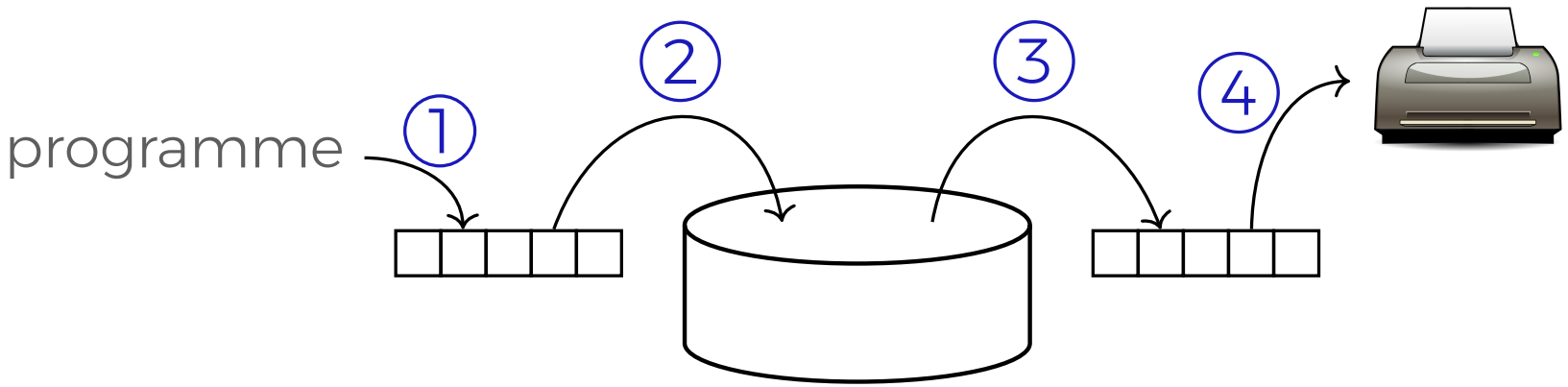
Processus concurrents



Processus autonomes $\Longrightarrow$ peuvent tourner en parallèlle.

Sont-ils indépendants ?

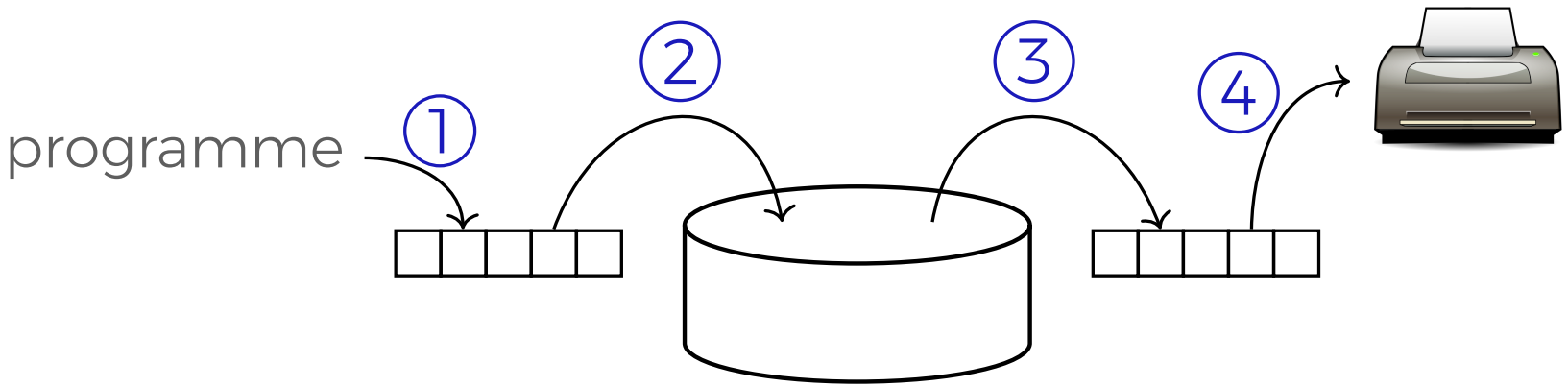
Processus concurrents



Processus autonomes $\Longrightarrow$ peuvent tourner en parallèlle.

Sont-ils indépendants ?

Processus concurrents



Processus autonomes $\Longrightarrow$ peuvent tourner en parallèlle.

Sont-ils indépendants ?

Ressources partagées $\Longrightarrow$ Non.

Processus concurrents $=$ processus autonomes qui partagent des ressources

Outline

Traces temporelles et contextes

Trace temporelle = la suite des débuts et des fins des actions résultant de l'exécution d'un processus.

Contexte = ensemble d'informations auxquelles un processus peut accéder.

- contexte commun = la partie du contexte d'un processus qui appartient au contexte d'un autre processus
- contexte privé = contexte − contexte commun

Exécution d'un ensemble de processus

Schéma 1 : exécution séquentielle

Schéma 2 : exécution pseudo-parallèle

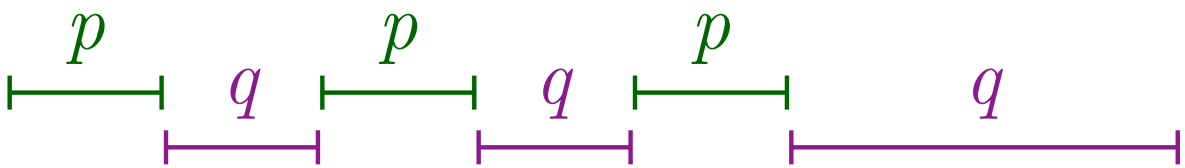
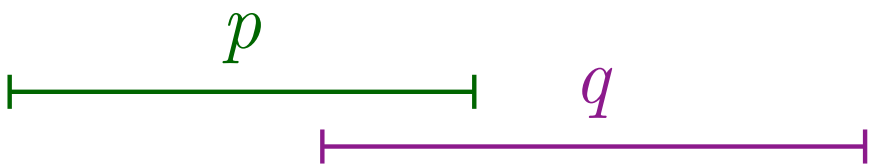


Schéma 3 : exécution réellement parallèle



Pourquoi du pseudo-parallèlisme ?

Exécution d'un ensemble de processus

Schéma 1 : exécution séquentielle

Schéma 2 : exécution pseudo-parallèle

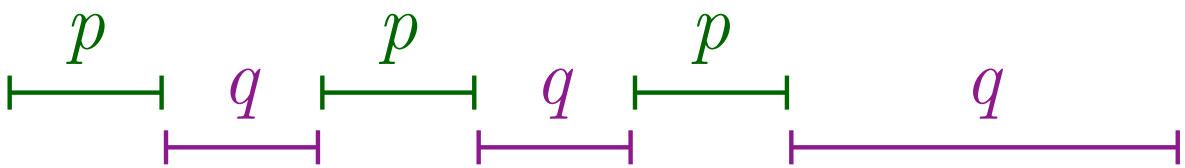
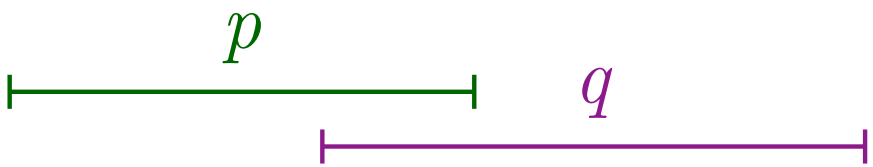


Schéma 3 : exécution réellement parallèle



Pourquoi du pseudo-parallèlisme ?

Exécution d'un ensemble de processus

Schéma 1 : exécution séquentielle

Schéma 2 : exécution pseudo-parallèle

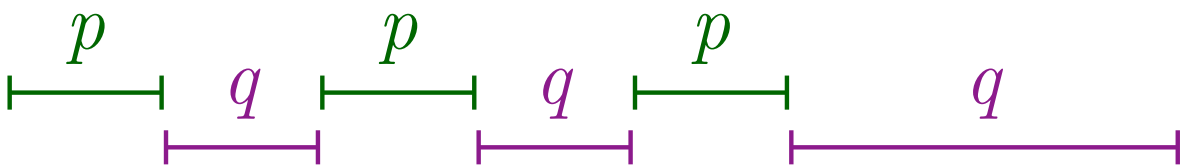
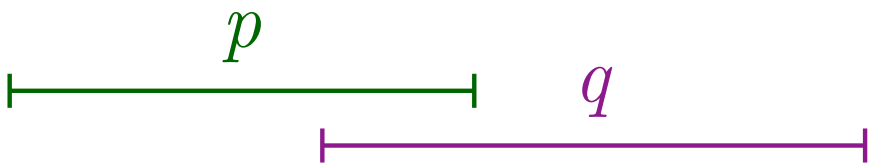


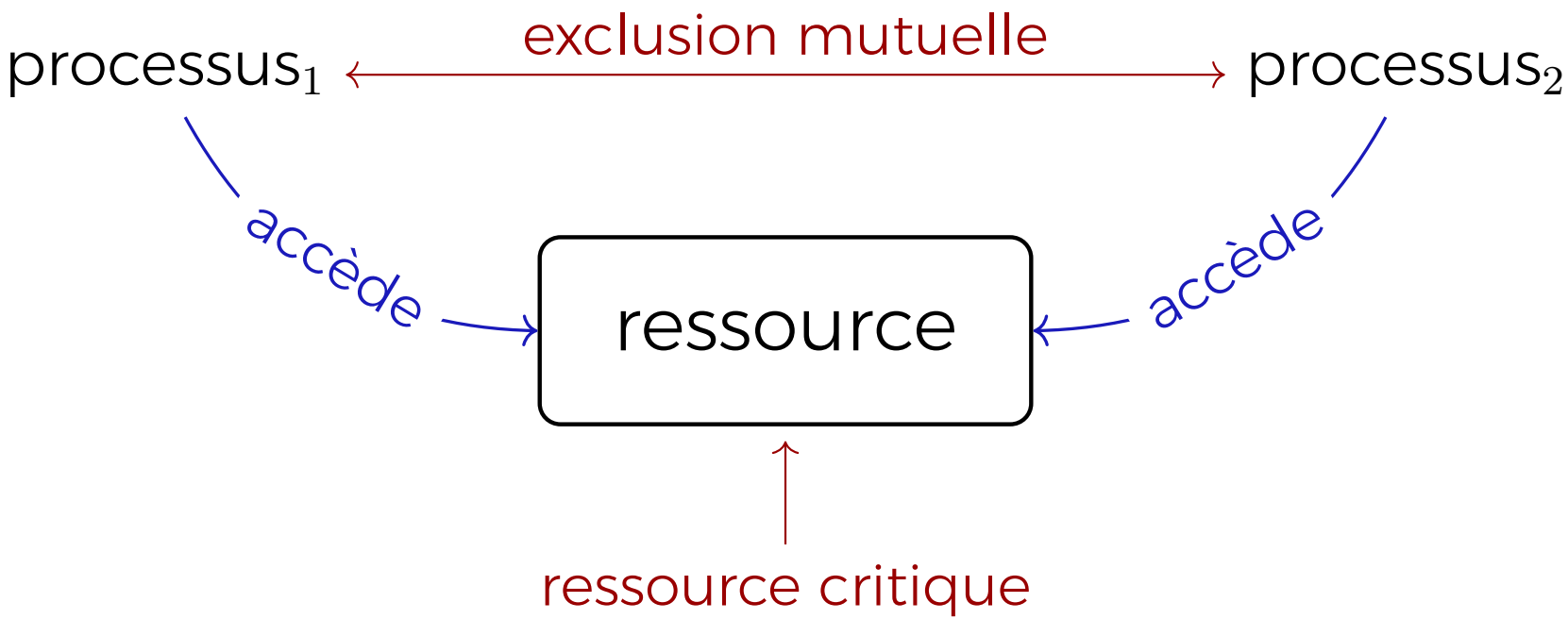
Schéma 3 : exécution réellement parallèle



Pourquoi du pseudo-parallèlisme ?

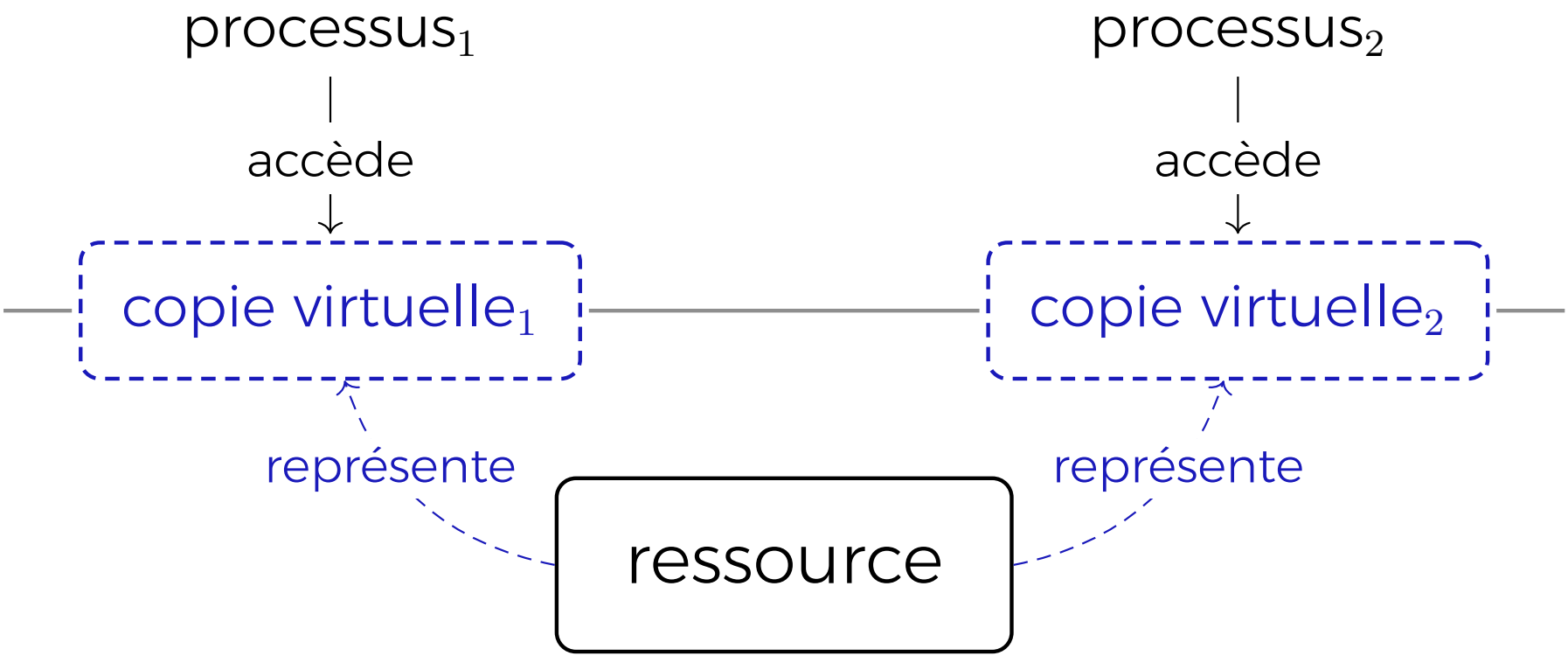
- nb. processeurs $<$ nb. tâches

Compétition entre processus



Coopération : ressources virtuelles

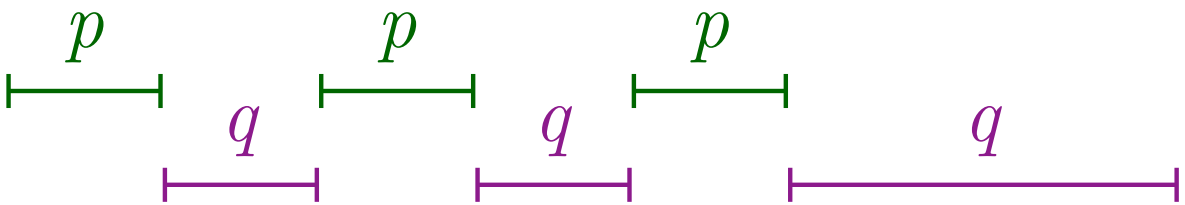
Mission du SE : gérer la compétition entre les processus en leur proposant des ressources virtuelles.



Les processus n'ont pas à gérer la compétition.

Schéma logique processeur virtuel

Exécution réelle :



Ce que voient les processus : et l'utilisateur

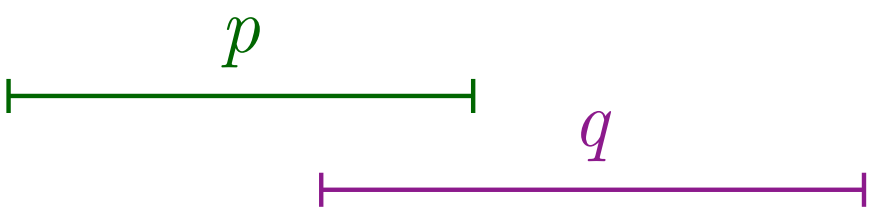
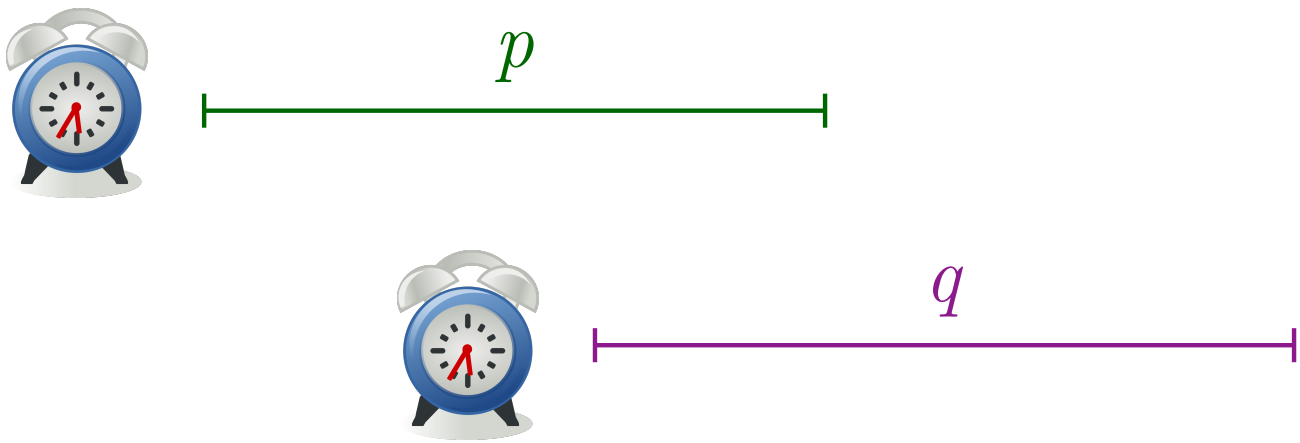


Schéma logique/virtuel

Schéma logique $\Longrightarrow$ temps logique

Avec un processeur virtualisé, aucun ordre relatif d'évènements de p et q n'est garanti.

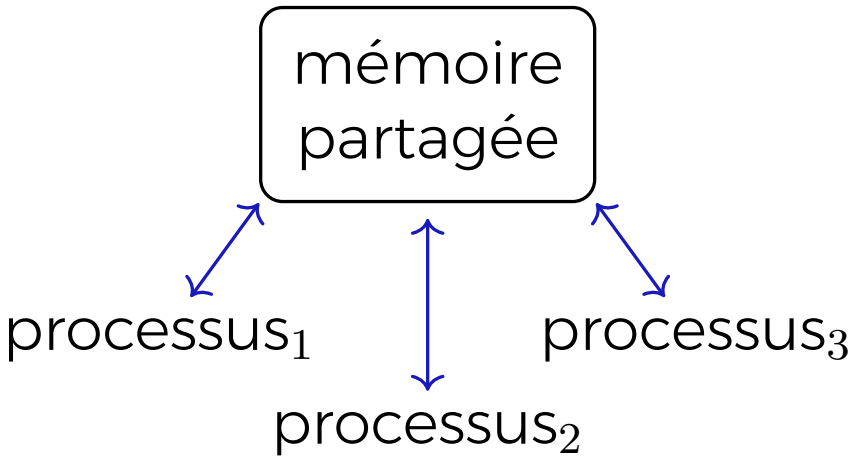


p et q ont chacun un temps à part $=$ temps logique

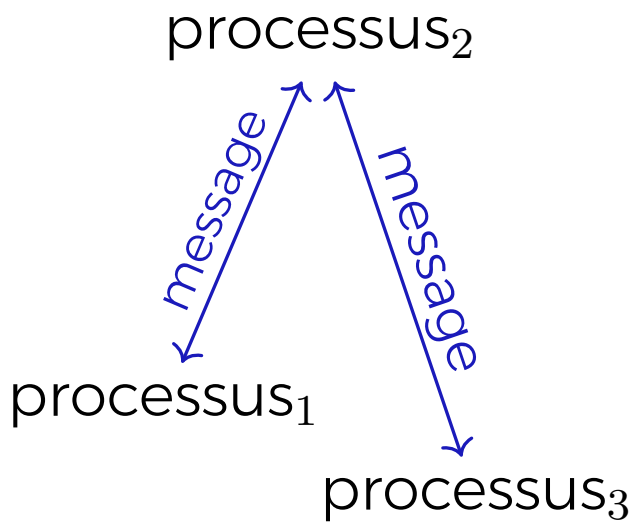
Outline

Communication entre processus

Système centralisé



Système réparti

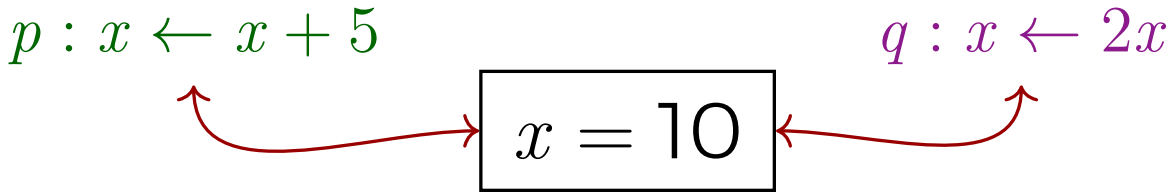


Mauvaise communication $\Longrightarrow$

- Mauvais fonctionnement
- Mauvaise utilisation des ressources

Problèmes associés à la communication

1. Indéterminisme

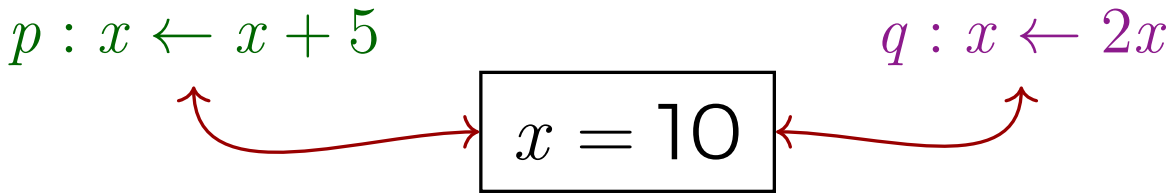


- exécution p, puis q : $x = 30$
- exécution q, puis p : $x = 25$

https://openclipart.org/

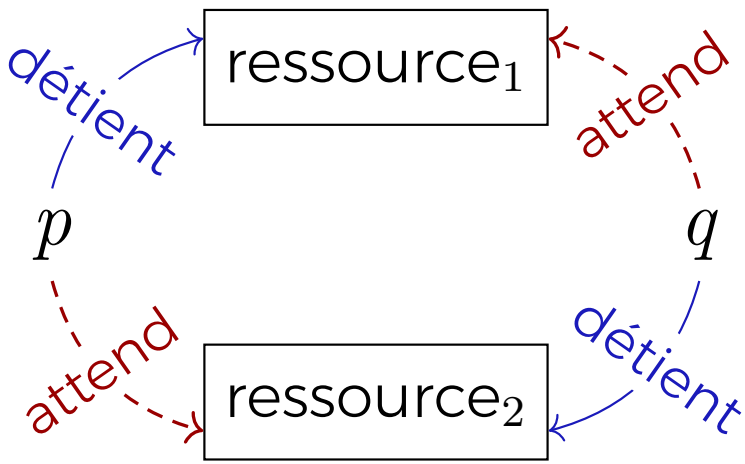
Problèmes associés à la communication

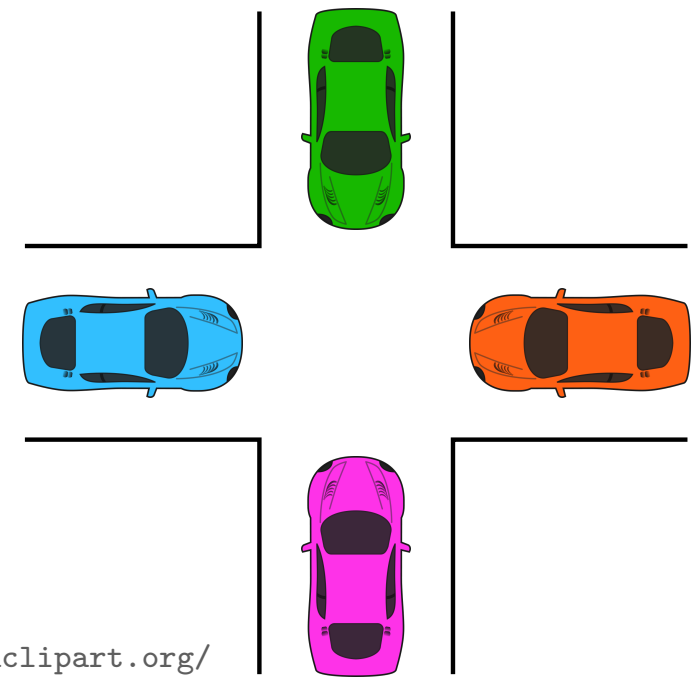
1. Indéterminisme



- exécution p, puis q : $x = 30$
- exécution q, puis p : $x = 25$

2. Blocages

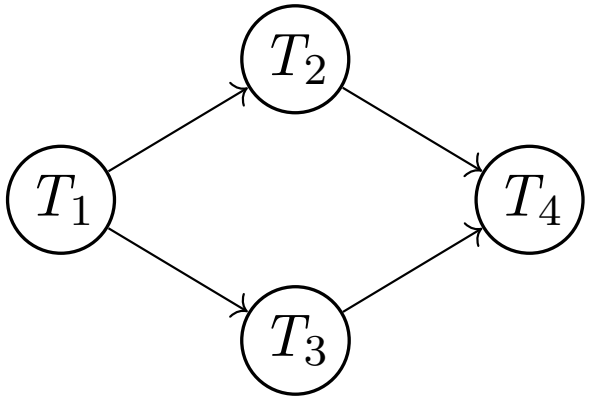




https://openclipart.org/

L'état du système

Système de tâches : $S = (E, <)$,



Un comportement : $w = d_1 f_1 \; d_2 d_3 \; f_2 f_3 \; d_4 f_4$

$\Downarrow\!\Downarrow$
agit sur
$\Downarrow\!\Downarrow$

L'état du système = les informations, les ressources, etc. associés au système

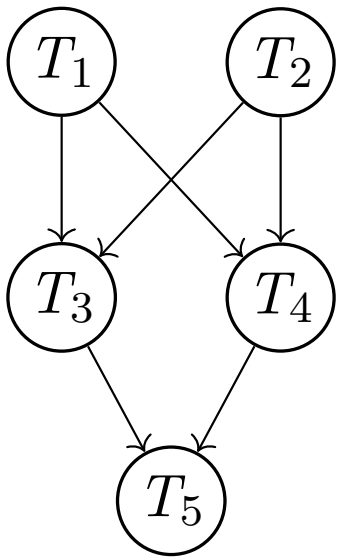
$$s_0 \xrightarrow{d_1} s_1 \xrightarrow{f_1} s_2 \xrightarrow{d_2} \cdots$$

transition

état initial

Outline

Décomposition en processus parallèles



$\Longleftarrow$

T_1 : lire X

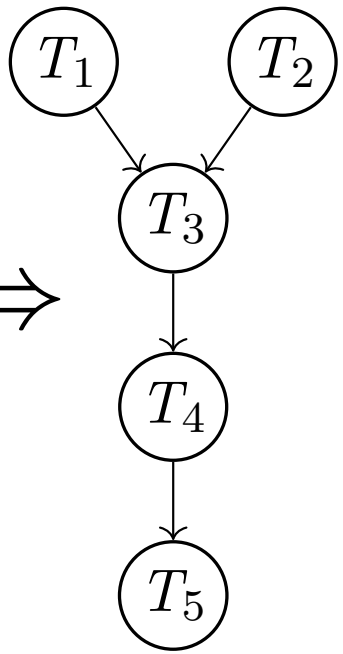
T_2 : lire Z

T_3 : X $\leftarrow$ X + Z

T_4 : Y $\leftarrow$ X + Z

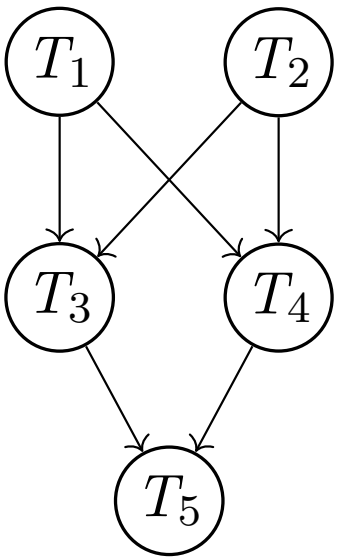
T_5 : afficher Y

$\Longrightarrow$



Décomposition en processus parallèles

Système indéterminé



$\Leftarrow$

T_1 : `lire X`

T_2 : `lire Z`

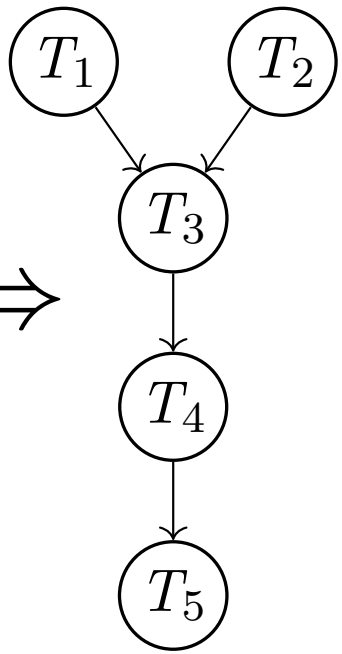
T_3 : `X` $\leftarrow$ `X + Z`

T_4 : `Y` $\leftarrow$ `X + Z`

T_5 : `afficher Y`

$\Rightarrow$

Système déterminé



décomposition équivalente

Comment décomposer ?

1. Calculer les changements d'état.
   - les valeurs écrites dans la mémoire, sur les périphériques, etc.
2. Trouver les conditions pour que le système soit déterminé.
3. Construire la décomposition parallèle.

Définition de l'état formelle

mémoire cellule

$$M = \left(C_1, \quad C_2, \quad \ldots, \quad C_n \right)$$

$$s_k = \left(C_1(k), \quad C_2(k), \quad \ldots, \quad C_n(k) \right)$$

état à l'instant k contenu de la cellule n à l'instant k

L'évolution de l'état se représente souvent sous forme de tableau.

Évolution de l'état

Comportement séquentiel: $w = d_1 f_1\ d_2 f_2\ d_3 f_3\ d_4 f_4\ d_5 f_5$

T_1 : `lire X`

T_2 : `lire Z`

T_3 : `X` $\leftarrow$ `X + Z`

T_4 : `Y` $\leftarrow$ `X + Z`

T_5 : `afficher Y`

	X	Y	Z
	0	0	0
d_1	0	0	0
f_1	3	0	0
d_2	3	0	0
f_2	3	0	7
d_3	.	.	.

Suites des valeurs écrites dans les variables:

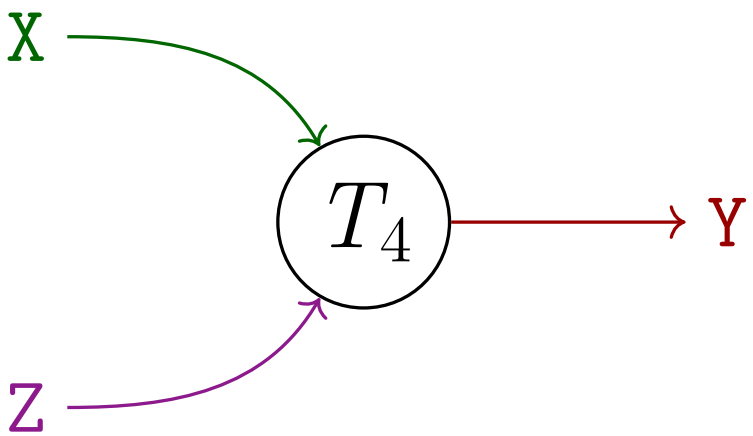
$V(\mathtt{X}, w)$	0	3	10
$V(\mathtt{Y}, w)$	0	17	
$V(\mathtt{Z}, w)$	0	7	

Domaines de tâches

	Domaine de Lecture	Domaine d'Écriture
T_1 : `lire X`	$L_1 = \varnothing$	$E_1 = \{\texttt{X}\}$
T_2 : `lire Z`	$L_2 = \varnothing$	$E_2 = \{\texttt{Z}\}$
T_3 : `X` $\leftarrow$ `X + Z`	$L_3 = \{\texttt{X}, \texttt{Z}\}$	$E_3 = \{\texttt{X}\}$
T_4 : `Y` $\leftarrow$ `X + Z`	$L_4 = \{\texttt{X}, \texttt{Z}\}$	$E_4 = \{\texttt{Y}\}$
T_5 : `afficher Y`	$L_5 = \{\texttt{Y}\}$	$E_5 = \varnothing$

Interprétation de tâches

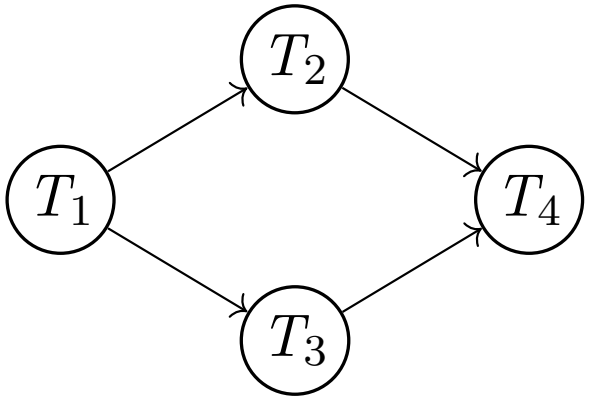
$$T_4 \ : \ \texttt{Y} \leftarrow \texttt{X} + \texttt{Z} \qquad L_4 = \{\texttt{X}, \texttt{Z}\} \qquad E_4 = \{\texttt{Y}\}$$



$F_{T_4}(X, Z) = (X + Z)$ ← la fonction qui interprète T_4

Système de tâches : structure complète

- le graphe de précédence



- les domaines de lecture L_i et d'écriture E_i de chaque tâche T_i
- l'interprétation F_{T_i} de chaque tâche T_i

Outline

Déterminisme d'un système de tâches

Un système est déterminé si, pour toute paire de comportements w et w', et pour toute cellule de mémoire $\mathtt{X}$, les mêmes suites de valeurs sont écrites dans $\mathtt{X}$ lors de l'exécution de w et de w':

$$V(\mathtt{X}, w) = V(\mathtt{X}, w')$$

L'interférence et les conditions de Bernstein

Deux tâches T_1 et T_2 sont non-interférentes si l'une des conditions 1 ou 2 est vraie :

1. $L_1 \cap E_2 = \varnothing$ et $L_2 \cap E_1 = \varnothing$ et $E_1 \cap E_2 = \varnothing$

   $L_1 \cap E_2 = \varnothing$: T_1 ne lit pas ce que T_2 écrit

   $L_2 \cap E_1 = \varnothing$: T_2 ne lit pas ce que T_1 écrit

   $E_1 \cap E_2 = \varnothing$: T_1 et T_2 n'écrivent pas dans les mêmes cellules

2. 

Le déterminisme et les conditions de Bernstein

Soit S un système de tâches.

- si toutes $T_1, T_2 \in S$ sont non-interférentes

  $\Longrightarrow$ S – déterminé.

- si S – déterminé et toute tâche de S a le domaine d'écriture $\neq \varnothing$

  $\Longrightarrow$ toutes $T_1, T_2 \in S$ sont non-interférentes

---

Domaine d'écriture vide $\Longrightarrow$ interférences cachées possibles

Outline

Parallélisme maximal

Comment paralléliser au maximum un système ?

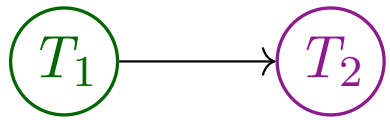
- construire le système de tâches équivalent avec le moins de contraintes de précédence

Équivalence de systèmes de tâches

Deux systèmes S_1 et S_2 sont équivalents si

- ils sont déterminés
- ils utilisent les « mêmes » cellules mémoire
  - on peut retrouver les cellules « équivalentes »
- pour toute cellule $\mathtt{X}$, S_1 écrit la même suite de valeurs dans $\mathtt{X}$ que S_2

Parallélisme maximal : définition

Un système de tâches S est de parallélisme maximal si la suppression de tout arc 

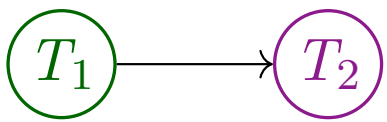
dans son graphe entraîne l'interférence de T_1 et T_2.

Tous les arcs sont nécessaires pour le fonctionnement correct.

Construction du parallélisme maximal

Soit S un système de tâches.

Construire le système de tâches S_{max} équivalent de parallélisme maximal de façon suivante :

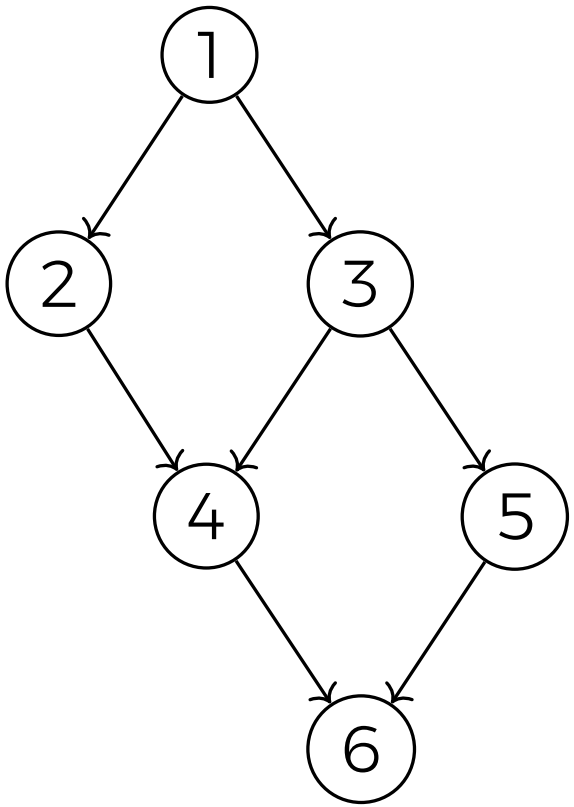
1. prendre toute paire de tâches T_1 et T_2 connectées par un chemin dans S
2. ajouter l'arc 

à S_{max} si

$$L_1 \cap E_2 \neq \varnothing \quad \text{ou} \quad L_2 \cap E_1 \neq \varnothing \quad \text{ou} \quad E_1 \cap E_2 \neq \varnothing$$

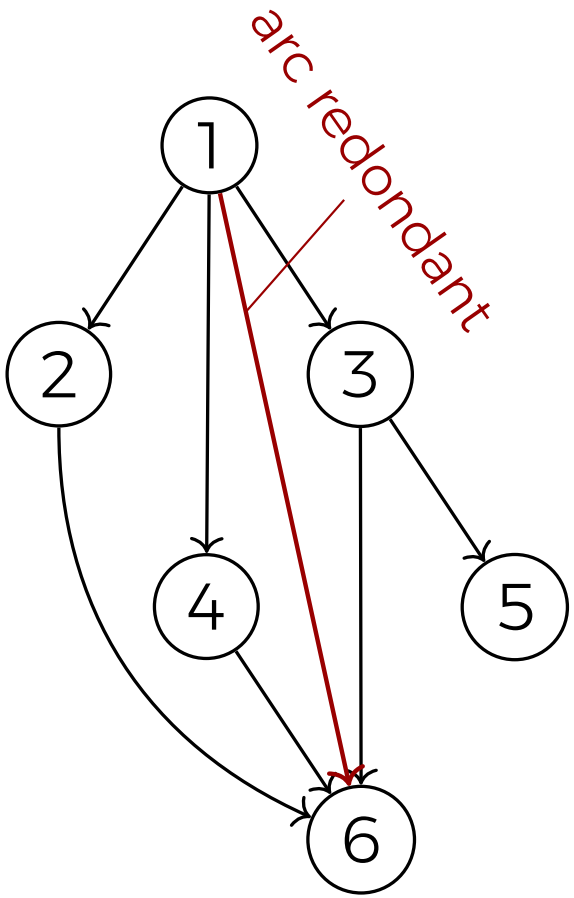
À la fin, éliminer les arcs redondants.

Parallélisme maximal : exemple



cellule	appartient à L_i	appartient à E_i
M_1	L_1, L_6	E_2
M_2	L_6	E_4
M_3	L_2, L_3	
M_4	L_2, L_3, L_4	E_1, E_6
M_5	L_5	E_3, E_5

Parallélisme maximal : solution



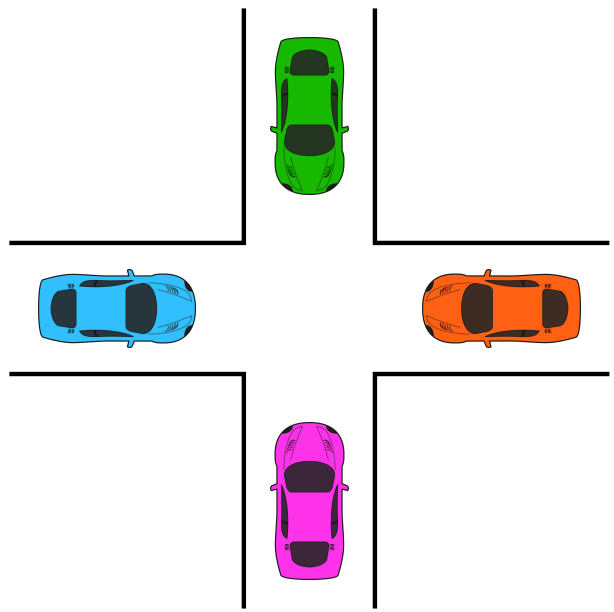
cellule	appartient à L_i	appartient à E_i
M_1	L_1, L_6	E_2
M_2	L_6	E_4
M_3	L_2, L_3	
M_4	L_2, L_3, L_4	E_1, E_6
M_5	L_5	E_3, E_5

Outline

Blocages

Caractéristiques d'un blocage entre quelques processus :

- les ressources ne sont pas partagées
- les ressources ne peuvent pas être réquisitionnées
- les processus sont en attente circulaire



https://openclipart.org/

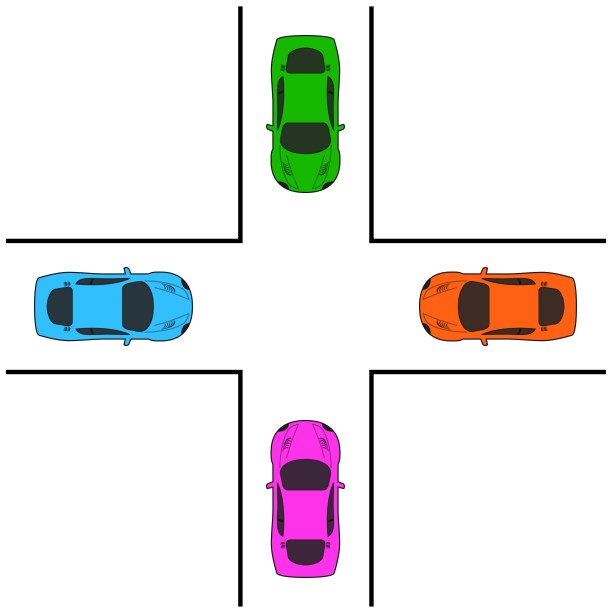
Méthodes de résolution des blocages $^{1}/_{2}$

1. Rattrapage des blocages
   - laisser le blocage se produire, puis reprendre et redistribuer certaines ressources
2. Prévention des blocages
   - s'assurer que les conditions de blocage ne seront jamais vraies
   - vérification statique
3. Évitement des blocages
   - contrôler les processus pas à pas
   - vérifier à chaque pas qu'il existe une possibilité d'échapper au blocage

Méthodes de résolution des blocages $^{2}/_{2}$

1. Rattrapage : faire reculer l'une des voitures
2. Prévention : installer des panneaux 

ou des 
3. Évitement : agent de police réglant la circulation



https://openclipart.org/

Outline

L'état du système après le pas k $^{1}/_{2}$

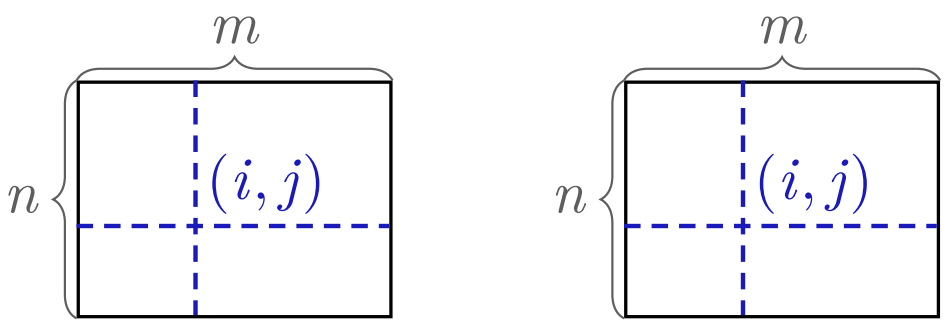
Système de n processus : $P_1, P_2, \ldots, P_n$

Type de ressource : $\boxed{R_1}$ $\boxed{R_2}$ $\cdots$ $\boxed{R_m}$

Quantité disponible : N_1 N_2 $\cdots$ N_m

- capacité du système

État après le pas k : $\texttt{BESOIN}(k)$ $\texttt{TENU}(k)$



$\texttt{BESOIN}_{i,j}(k)$ = nb. de ressources R_j demandées par P_i

$\texttt{TENU}_{i,j}(k)$ = nb. de ressources R_j détenues par P_i

L'état du système après le pas k

Type de ressource :	R_1	R_2	$\cdots$	R_m
Quantité disponible :	N_1	N_2	$\cdots$	N_m

$\texttt{DISPO}_j(k)$ = nombre de ressources R_j libres

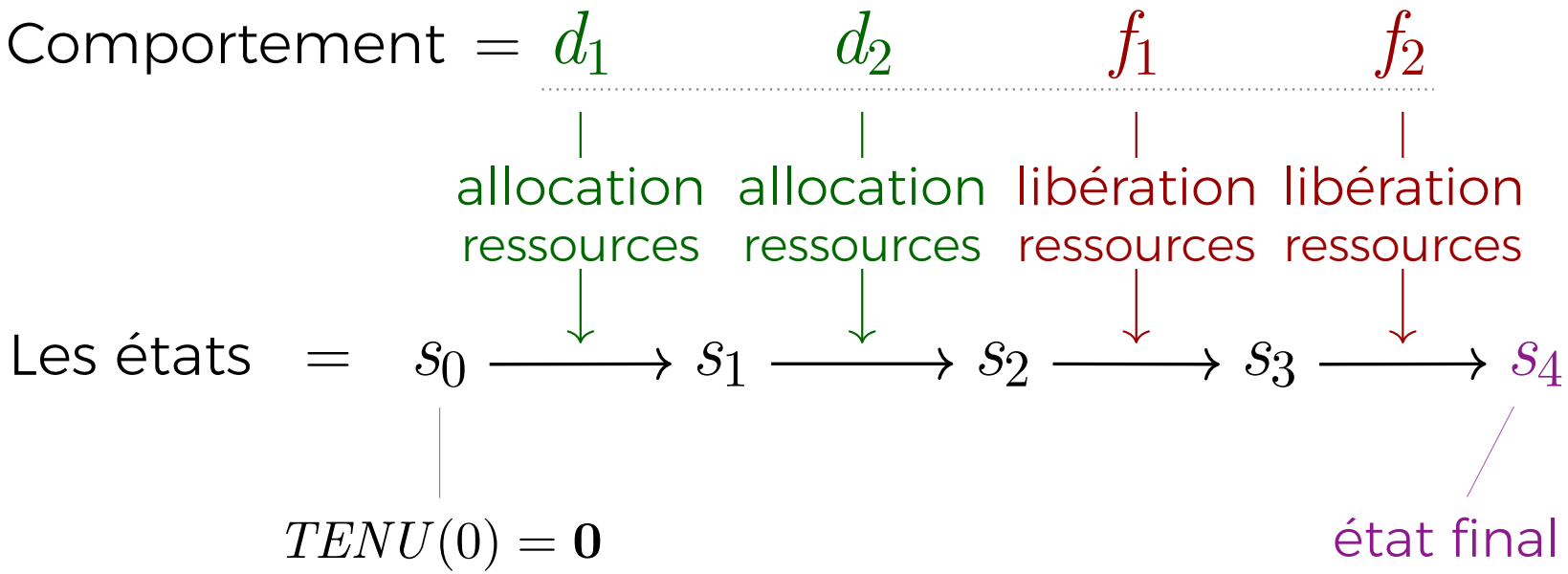
- non détenues par personne

$\texttt{DISPO}_j(k) = N_j -$ instances détenues

$$\sum_{i=1}^{n} \texttt{TENU}_{i,j}(k)$$

$$\texttt{DISPO}(k) = (\texttt{DISPO}_1(k), \ldots, \texttt{DISPO}_m(k))$$

L'état $=$ f(comportement)



Les états forment une chaîne d'états.

Cette chaîne est terminée si $\texttt{BESOIN}(4) + \texttt{TENU}(4) = \mathbf{0}$.

Outline

Définition du blocage $^{1}/_{2}$

$$\text{Comportement} = \overset{d_1}{\downarrow} \quad \overset{d_2}{\downarrow} \quad \overset{f_1}{\downarrow} \quad \overset{f_2}{\downarrow}$$

$$\text{Les états} = s_0 \xrightarrow{\downarrow} s_1 \xrightarrow{\downarrow} s_2 \xrightarrow{\downarrow} s_3 \xrightarrow{\downarrow} s_4$$

s_4 — blocage si :

- il existe un ensemble de processus $D = \{P_1, \ldots, P_n\}$
- pour tout $P_i \in D$

$$\texttt{BESOIN}_i(4) \not\leq \texttt{DISPO}(4) + \sum_{k \notin D} \texttt{TENU}_k(4)$$

$(1, 2) \leq (2, 3)$, mais $(1, 2) \not\leq (3, 1)$

Définition du blocage $^{2}/_{2}$

$\exists D = \{P_1, \ldots, P_n\}$, tel que $\forall P_i \in D$:

$$\texttt{BESOIN}_i(4) \leq \texttt{DISPO}(4) + \sum_{k \notin D} \texttt{TENU}_k(4)$$

ressources dont les processus de D ont besoin $\leq$ ressources disponibles $+$ ressources qui seront relâchées par d'autres

https://openclipart.org/

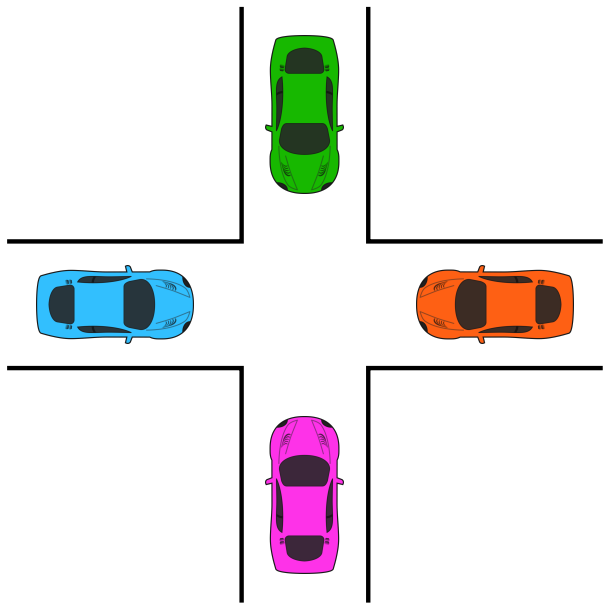
Définition du blocage $^{2}/_{2}$

$\exists D = \{P_1, \ldots, P_n\}$, tel que $\forall P_i \in D$:

$$\texttt{BESOIN}_i(4) \not\leq \texttt{DISPO}(4) + \sum_{k \notin D} \texttt{TENU}_k(4)$$

ressources dont les processus de D ont besoin $>$ ressources disponibles $+$ ressources qui seront relâchées par d'autres

Blocage : Les processus dans D n'ont pas assez de ressources pour avancer, même s'ils attendent



https://openclipart.org/

Blocages : remarques

- État bloqué $\overset{1}{\underline{\qquad\qquad}}\overset{n}{}$ ensembles D
- Aucune transition possible à partir d'un état $s_k \implies s_k$ — blocage
- Tous les processus ne sont pas forcément bloqués dans un état bloqué.
- Blocage $\implies$ attente circulaire

- Accès à des ressources en exclusion mutuelle $\implies$ blocages
- Blocage $\implies$ non-réquisition des ressources

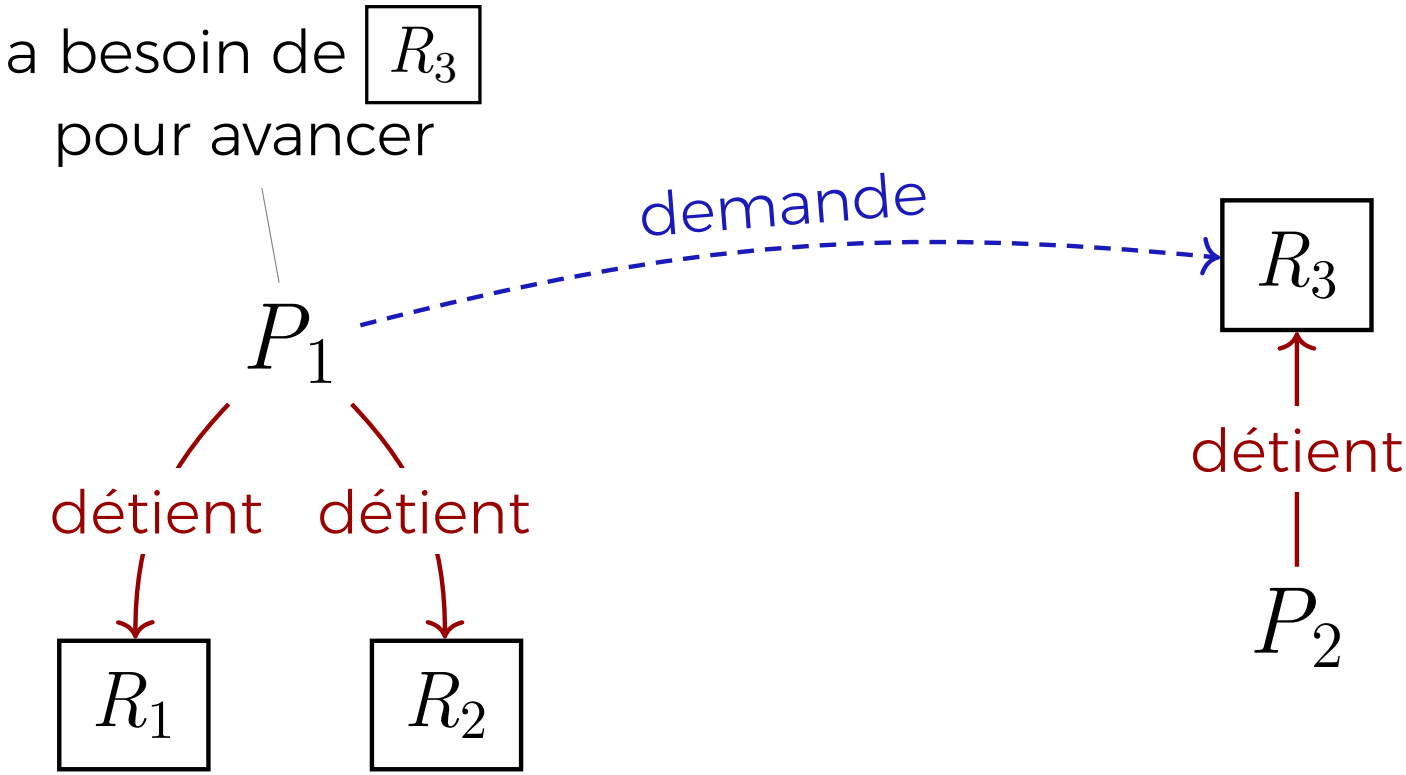
Outline

Prévention des blocages = contrôler l'accès aux ressources pour faire en sorte que les blocages ne se produisent jamais.

Supprimer les phénomènes suivants :

- la non-réquisition des ressources
- « tenir et attendre »
- l'attente circulaire

Réquisition des ressources



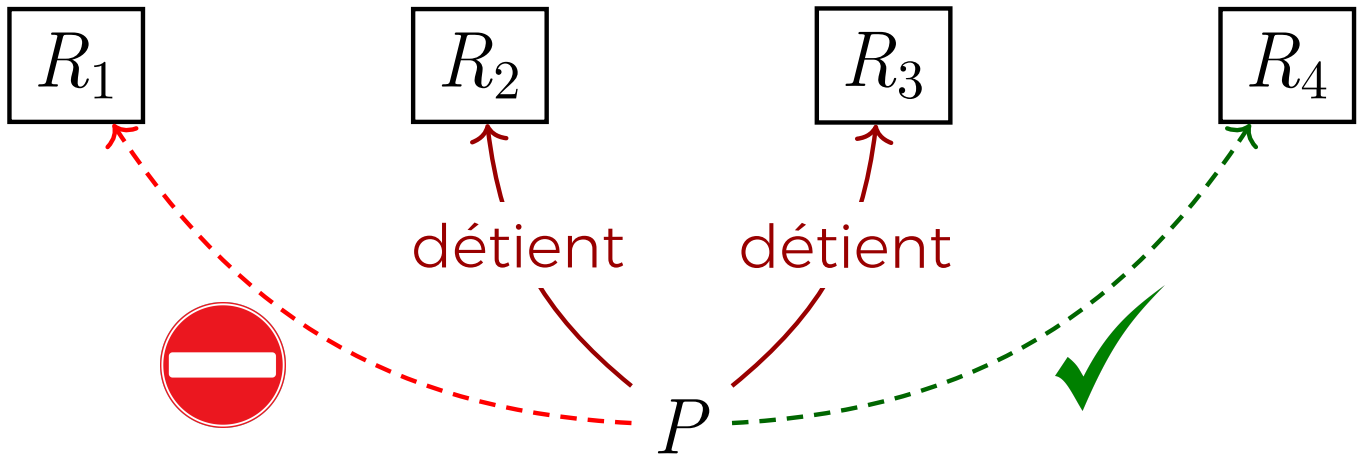
Solution : Réquisitionner R_3 pour laisser d'autres processus avancer.

Empêcher les processus de « tenir et attendre »

- Demander toutes les ressources au lancement
  - \+ aucune demande et détention
  - — détention de ressources inutilisées
- Libérer toutes les ressources avant chaque demande
  - \+ moins contraignant
  - — réallocation de la même ressources $\Longrightarrow$ ralentissements

Empêcher l'attente circulaire

Utilisation ordonnée des ressources :



P_1 peut demander $\boxed{R_4}$ parce que $4 > 3$ et $4 > 2$.

P_1 ne peut pas demander $\boxed{R_1}$ parce que $1 < 2$.

P peut demander uniquement les ressources qui ont un numéro plus grand que les numéros des ressources que P détient déjà.

Utilisation ordonnée : remarques

- Pour demander plusieurs unités de la même ressource, faire une seule demande.
  - interdit de redemander des ressources du même type
- La numérotation des ressources est importante.
  - pour refléter l'utilisation habituelle

Outline

Détection des blocages

Un état s_n — blocage ?

Réponse : Tester s'il existe un ou plusieurs processus P_i qui n'ont pas assez de ressources pour avancer, même s'ils attendent :

$$\texttt{BESOIN}_i(n) \not\leq \texttt{DISPO}(n) + \sum_{k \notin D} \texttt{TENU}_k(n)$$

- tester assez souvent pour améliorer la détection
- ne pas tester trop souvent pour ne pas ralentir

Reprise après un blocage

Reprise = réquisition des ressources de certains processus bloqués

---

Problèmes :

- quels processus ?
- qu'en faire après la reprise ?

Reprise : à qui réquisitionner ?

- Retirer les ressources de tous les processus bloqués.
- Retirer progressivement, jusqu'à disparition du blocage
  - surcoût considérable
- Choisir les processus selon un système de critères :
  - priorité, ancienneté, temps restant, ressources détenues, ressources nécessaires pour terminer, etc.

Famine = les ressources du même processus sont systématiquement réquisitionnées

Un processus en famine peut ne jamais terminer.

Une solution : Augmenter sa priorité à chaque réquisition.

Que faire après la réquisition ?

Des ressources ont été retirées à des processus bloqués.

Que faire de ces processus ?

- Les détruire.
  - \+ simplicité
  - — perte de résultats
  - — les processus détruits recommencent leur exécution depuis le début

- Faire un retour en arrière partiel rollback
  - \+ souplesse
  - — stockage des points de sauvegarde

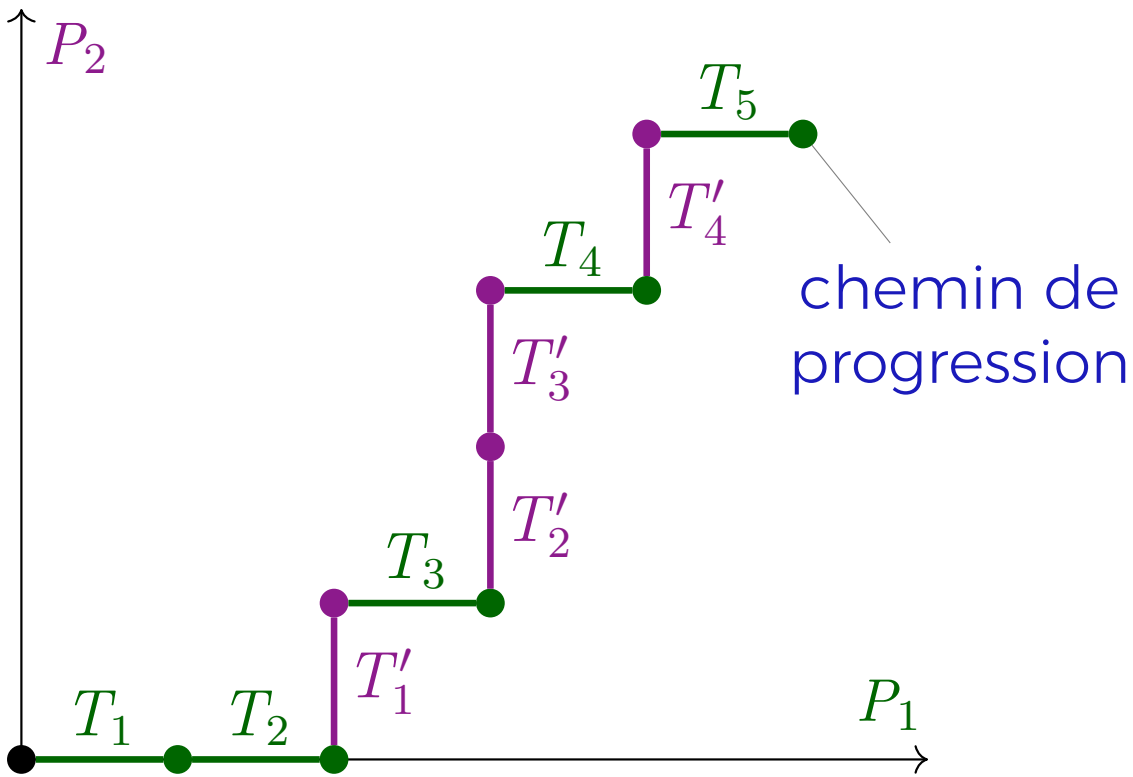
Outline

Chemins de progression

Soient deux processus :

$$P_1 = T_1\,T_2\,T_3\,T_4\,T_5 \quad \text{et} \quad P_2 = T'_1\,T'_2\,T'_3\,T'_4$$

Soit l'évolution : $T_1\,T_2\,T'_1\,T_3\,T'_2\,T'_3\,T_4\,T'_4\,T_5$

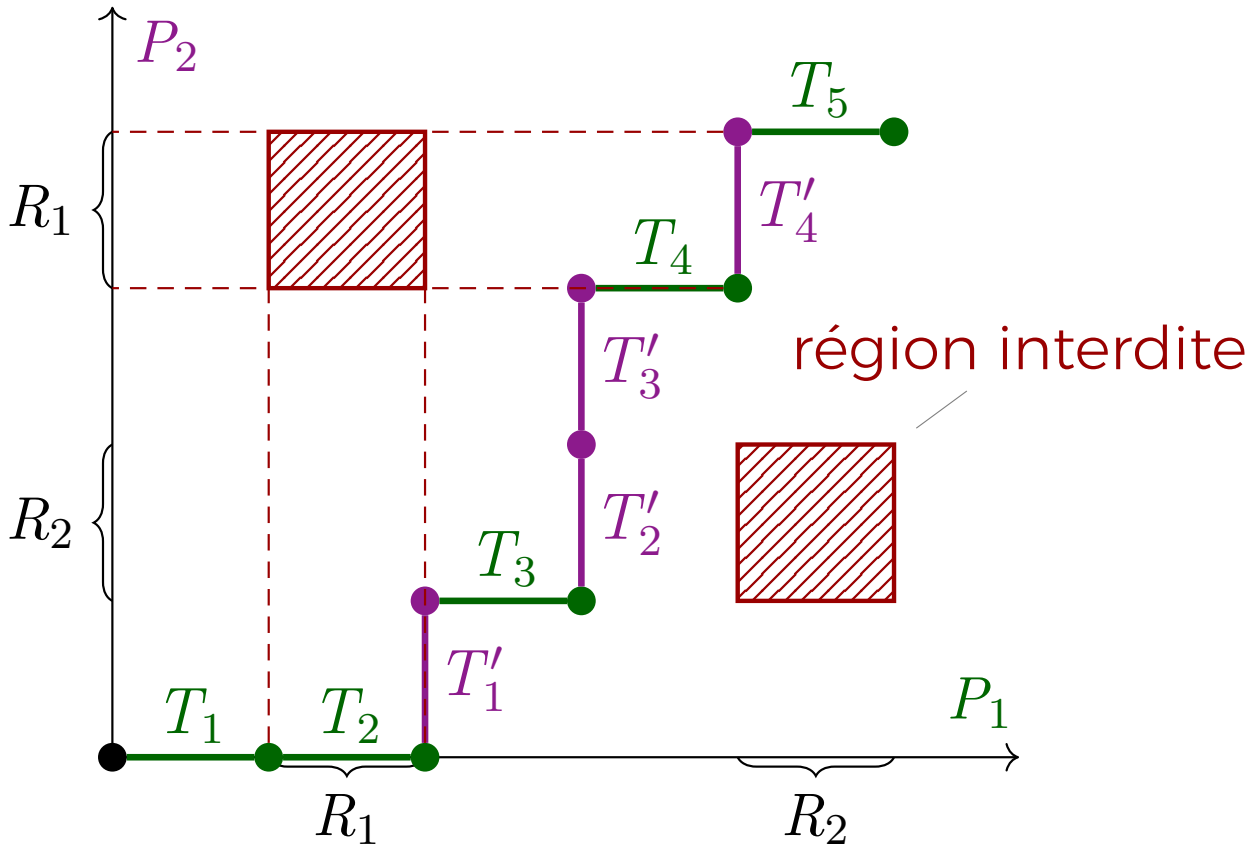


Régions interdites

Soient $P_1 = T_1\,T_2\,T_3\,T_4\,T_5$ et $P_2 = T'_1\,T'_2\,T'_3\,T'_4$.

Supposons les besoins en ressources suivants:

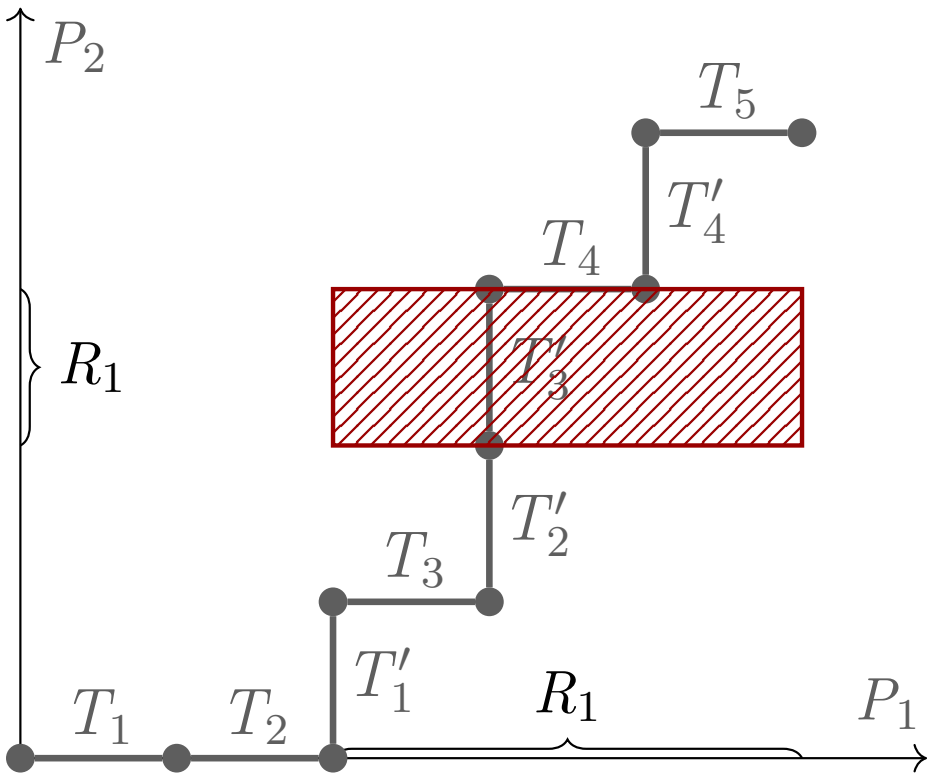
- T_2 a besoin de R_1
- T_5 a besoin de R_2
- T'_2 a besoin de R_2
- T'_4 a besoin de R_1



Régions dangereuses

Supposons les besoins en ressources suivants:

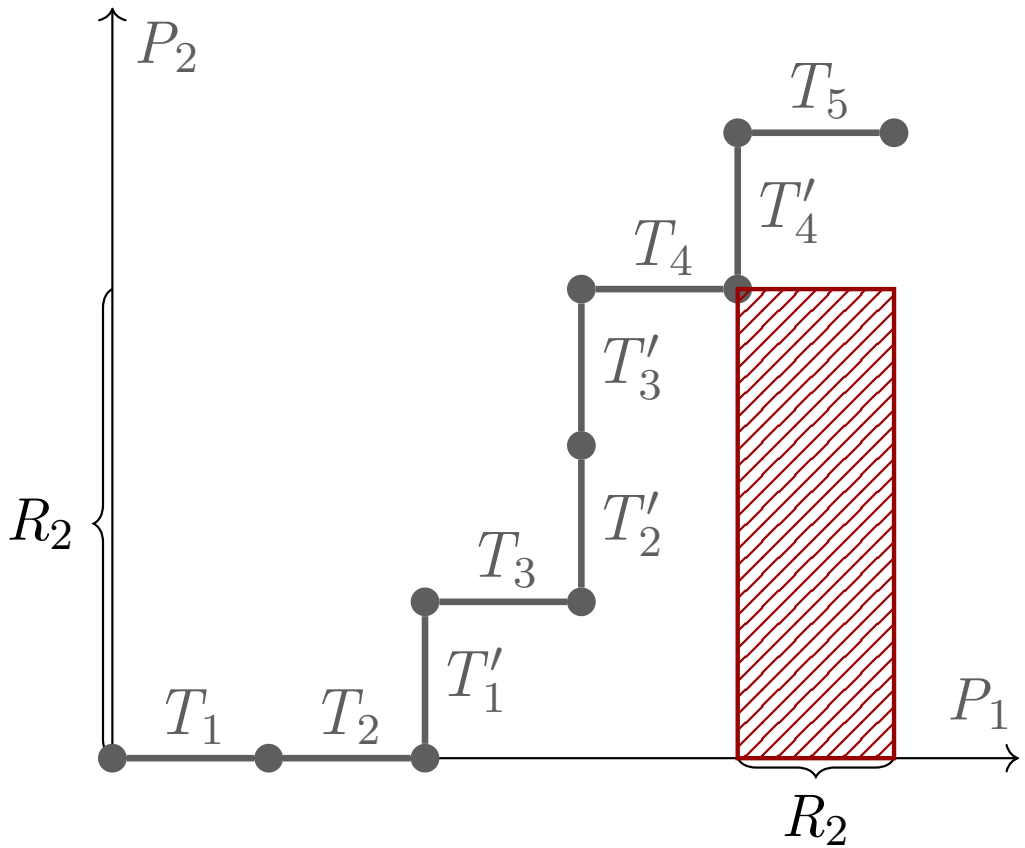
- $T_3, T_4, T_5 : R_1$
- $T'_3 : R_1$



Régions dangereuses

Supposons les besoins en ressources suivants:

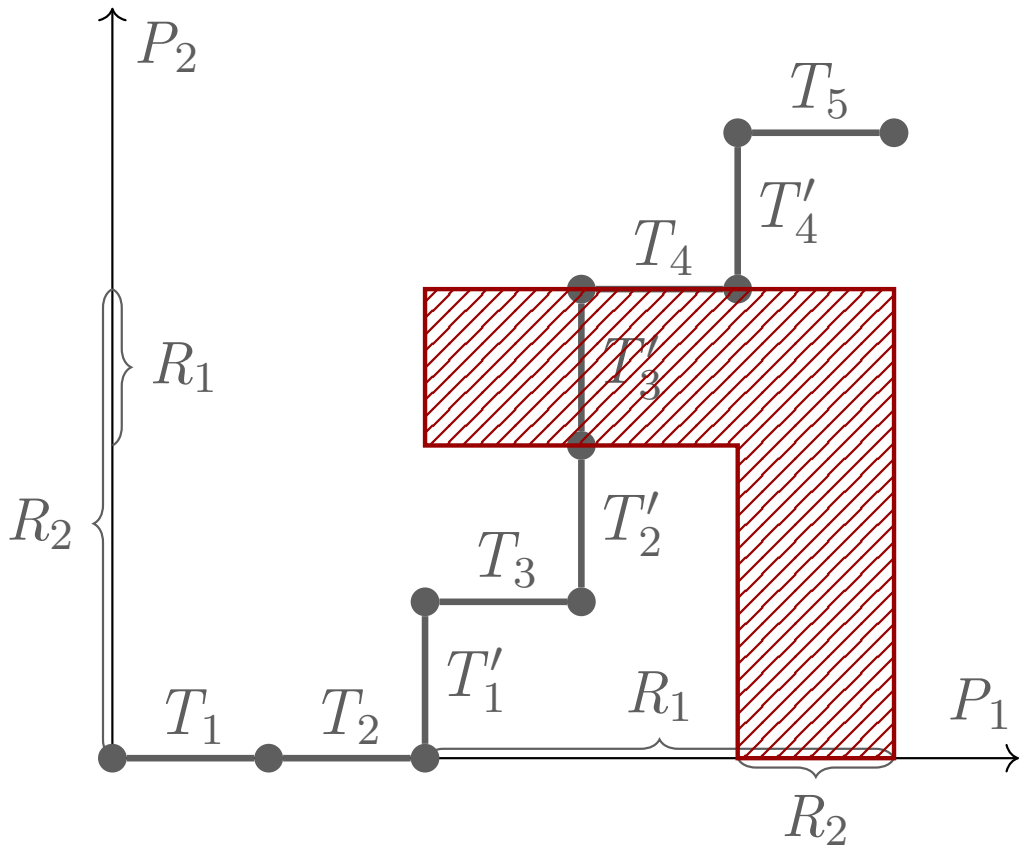
- $T_5 : R_2$
- $T'_1, T'_2, T'_3 : R_2$



Régions dangereuses

Supposons les besoins en ressources suivants:

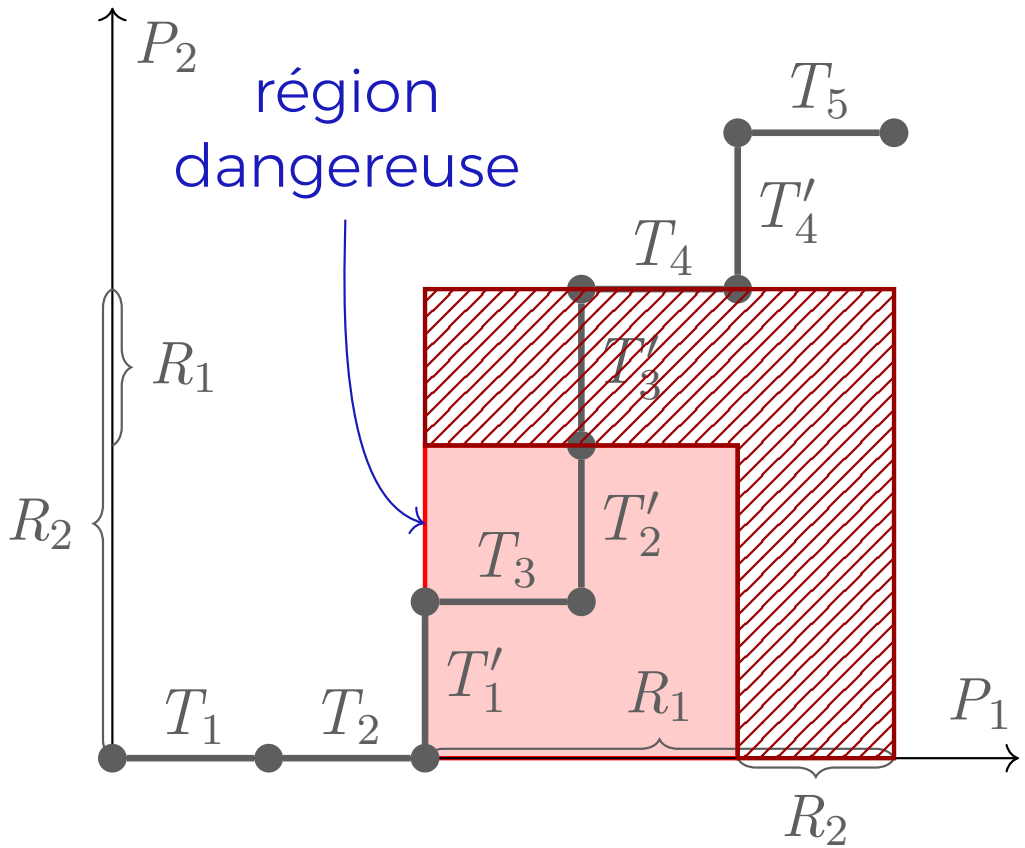
- $T_3, T_4, T_5 : R_1$
- $T_5 : R_2$
- $T'_3 : R_1$
- $T'_1, T'_2, T'_3 : R_2$



Régions dangereuses

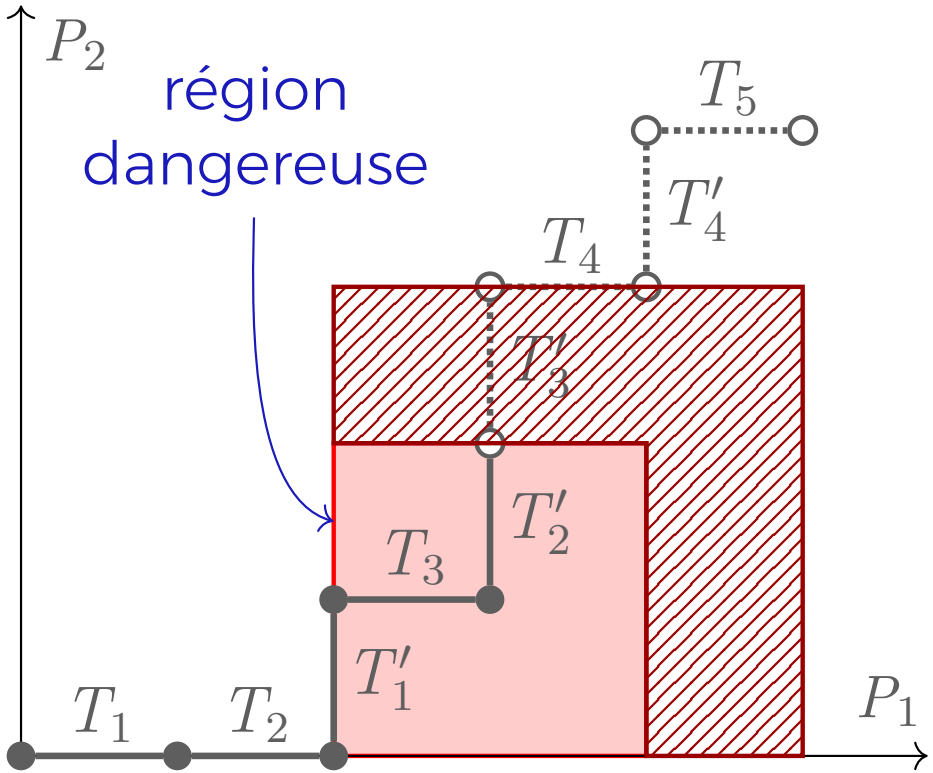
Supposons les besoins en ressources suivants:

- $T_3, T_4, T_5 : R_1$
- $T_5 : R_2$
- $T'_3 : R_1$
- $T'_1, T'_2, T'_3 : R_2$



États sûrs

Chemin dans une région dangeureuse $\Longrightarrow$ blocage



État sûr $=$ état qui est à l'extérieur de toute région dangereuse

- le blocage peut être évité

Évitement des blocages

Stratégie : Empêcher le système de rentrer dans des états non-sûrs.

- S'assurer que chaque nouvel état est sûr
  - algorithmes énumératifs $\Longrightarrow$ surcoût
  - nécessite la connaissance complète de tous les futurs besoins en ressources

- Fixer le quota maximal de demandes, par processus

  Pour chaque processus P_i, à tout pas d'exécution n :

$$\texttt{BESOIN}_i(n) + \texttt{TENU}_i(n) \leq \texttt{MAX}_i$$

Algorithme du banquier — critère suffisant

Soit s_k un état sûr et P qui demande des ressources.

Algorithme du banquier :

1. Supposer qu'on dise oui : construire s_{k+1}
2. Construire l'état t_{k+1} dans lequel chaque processus P_i demande son quota entier :

$$\texttt{BESOIN}'_i(k+1) \leftarrow \texttt{MAX}_i - \texttt{TENU}_i(k+1),$$

si $\texttt{TENU}_i(k+1) + \texttt{BESOIN}_i(k+1) > 0$

3. Tester si t_{k+1} est un blocage :

$$\texttt{BESOIN}'_i(k+1) \not\leq \texttt{DISPO}(k+1) + \sum_{m \notin D} \texttt{TENU}_m(k+1)$$

Si t_{k+1} n'est pas un blocage $\implies$ s_{k+1} est sûr

$\implies$ on peut dire oui à P