

Systemes d'exploitation

Interaction de processus

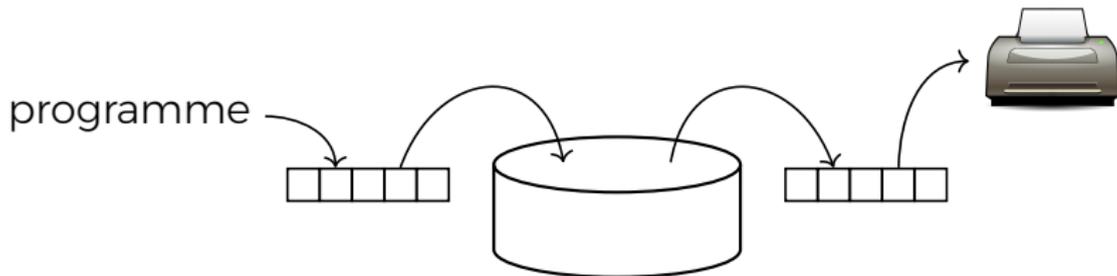
Sergiu Ivanov

`sergiu.ivanov@univ-evry.fr`

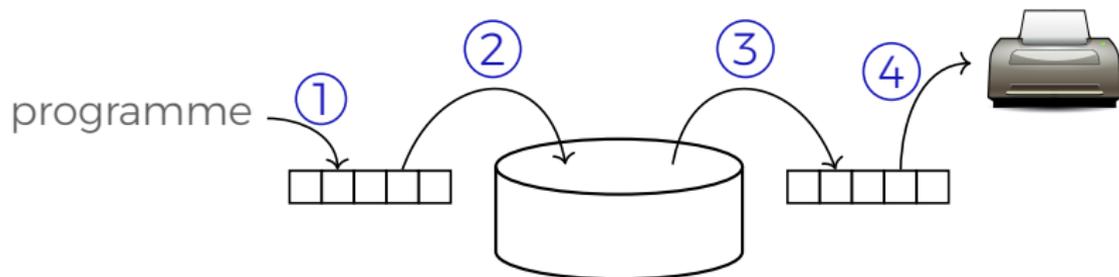
`http://lacl.fr/~sivanov/`

Comment les processus interagissent-ils ?

Combien y a-t-il de processus ?



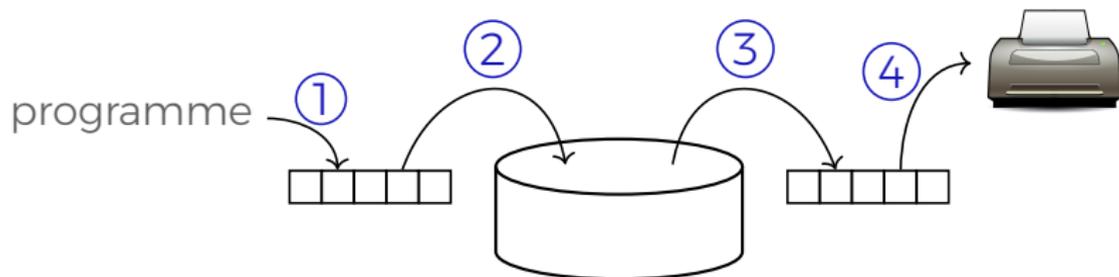
Processus concurrents



Processus **autonomes** \implies peuvent tourner **en parallèle**.

Sont-ils **indépendants** ?

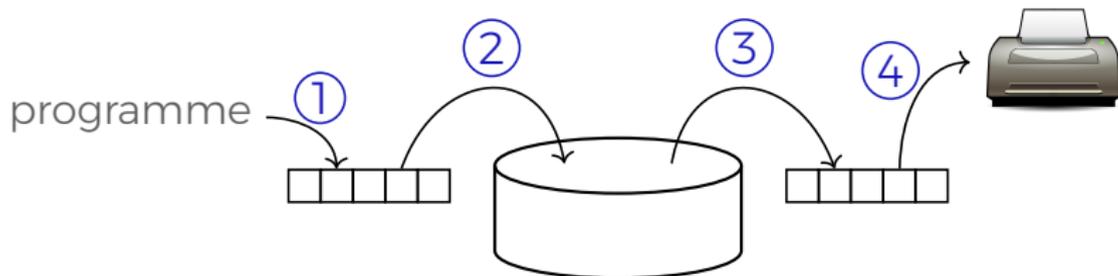
Processus concurrents



Processus **autonomes** \implies peuvent tourner **en parallèle**.

Sont-ils **indépendants** ?

Processus concurrents



Processus **autonomes** \implies peuvent tourner **en parallèle**.

Sont-ils **indépendants** ?

Ressources partagées \implies **Non**.

Processus **concurrents** = processus autonomes qui partagent des ressources

Outline

1. Relations entre processus
2. Problèmes associés à la communication
3. Déterminisme et parallélisme maximal
 - Caractère déterminé et la condition de Bernstein
 - Parallélisme maximal
4. Blocages
5. Définition des états du système
6. Définition du blocage
7. Prévention des blocages
8. Détection des blocages et reprise
9. Évitement des blocages

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

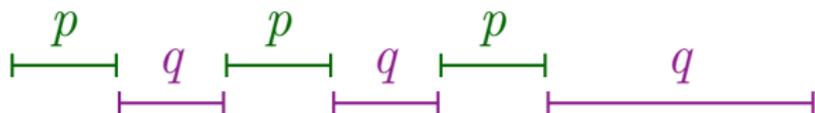


Schéma 3 : exécution réellement parallèle



Pourquoi du pseudo-parallélisme ?

Exécution d'un ensemble de processus

Schéma 1 : exécution séquentielle



Schéma 2 : exécution pseudo-parallèle

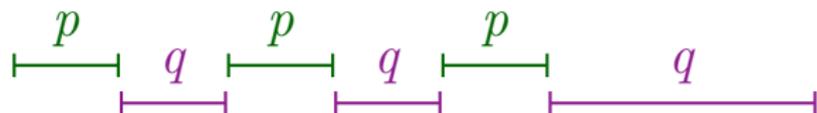


Schéma 3 : exécution réellement parallèle



Pourquoi du pseudo-parallélisme ?

Exécution d'un ensemble de processus

Schéma 1 : exécution séquentielle



Schéma 2 : exécution pseudo-parallèle

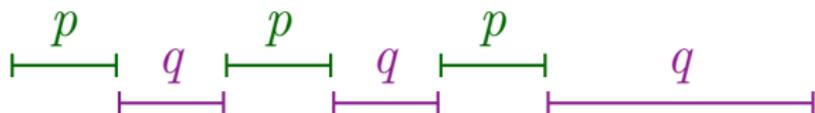


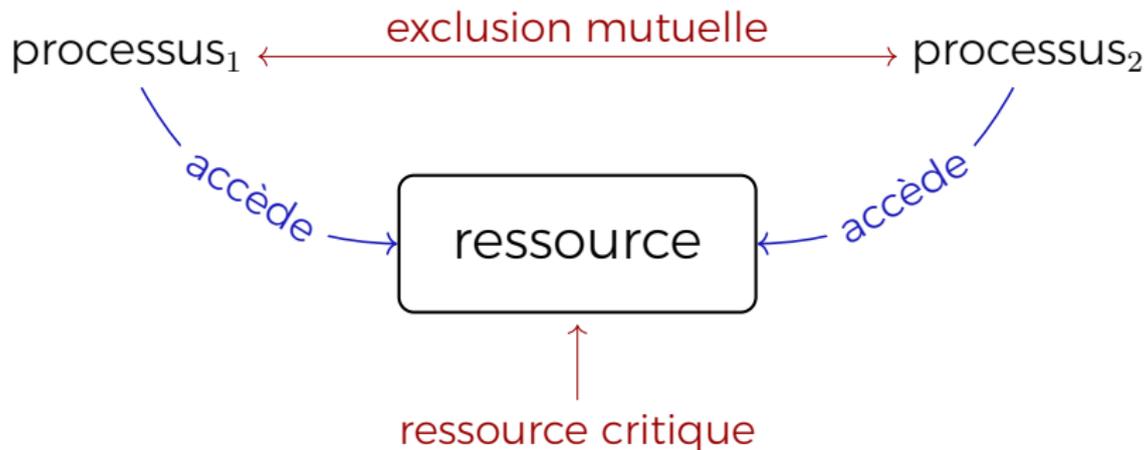
Schéma 3 : exécution réellement parallèle



Pourquoi du pseudo-parallélisme ?

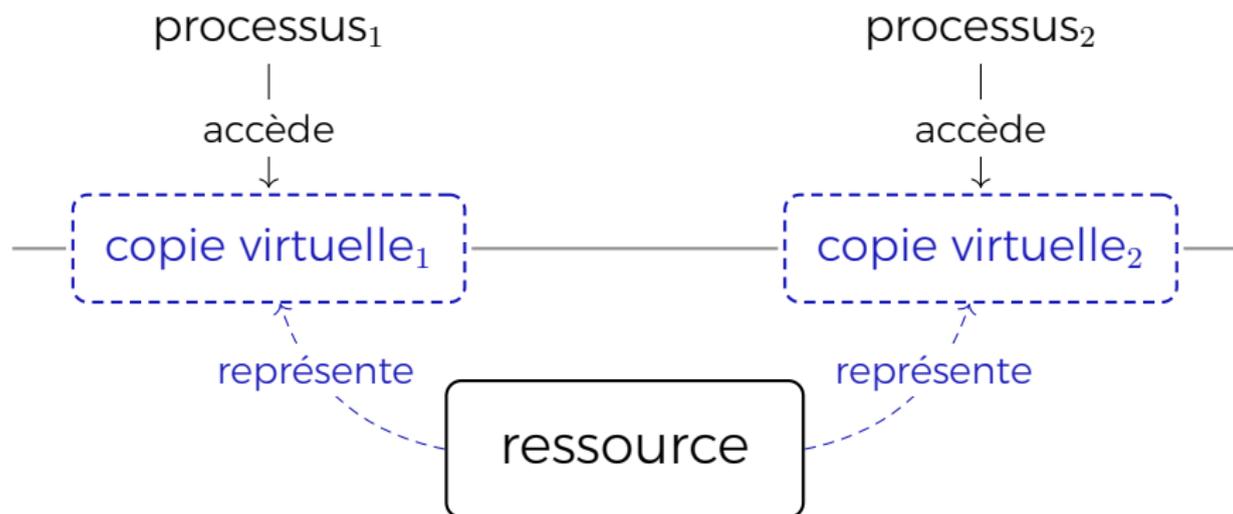
- ▶ nb. processeurs < nb. tâches

Compétition entre processus



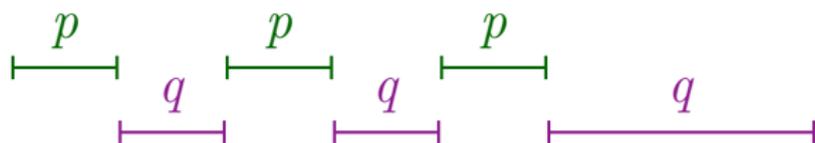
Coopération : ressources virtuelles

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



Les processus n'ont pas à gérer la compétition.

Exécution réelle :



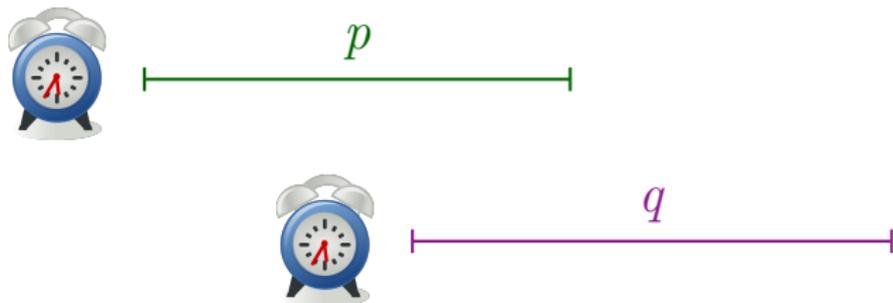
Ce que voient les processus : et l'utilisateur



Schéma logique/virtuel

Schéma logique \implies temps logique

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



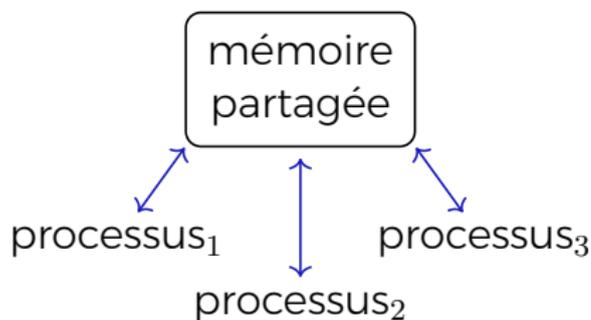
p et q ont chacun un temps à part = temps logique

Outline

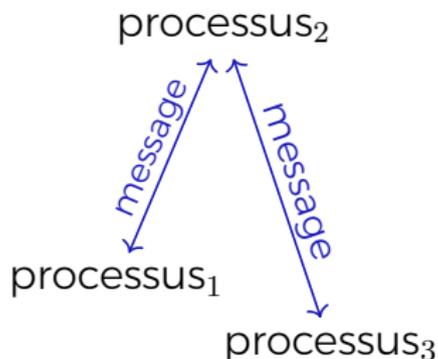
1. Relations entre processus
2. Problèmes associés à la communication
3. Déterminisme et parallélisme maximal
 - Caractère déterminé et la condition de Bernstein
 - Parallélisme maximal
4. Blocages
5. Définition des états du système
6. Définition du blocage
7. Prévention des blocages
8. Détection des blocages et reprise
9. Évitement des blocages

Communication entre processus

Système centralisé



Système réparti



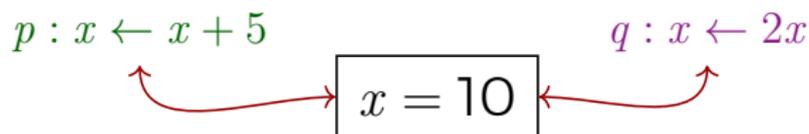
Mauvaise
communication



- ▶ Mauvais **fonctionnement**
- ▶ Mauvaise utilisation des **ressources**

Problèmes associés à la communication

1. Indéterminisme



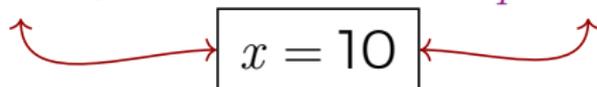
- ▶ exécution p , puis $q : x = 30$
- ▶ exécution q , puis $p : x = 25$

Problèmes associés à la communication

1. Indéterminisme

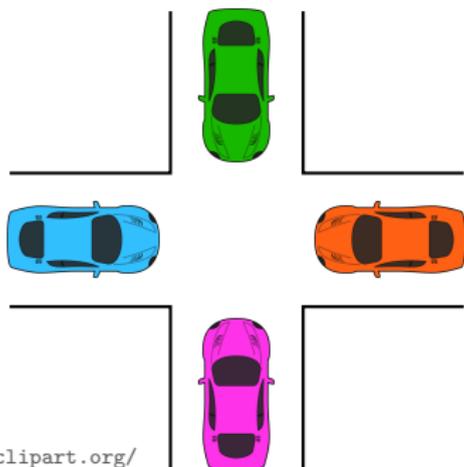
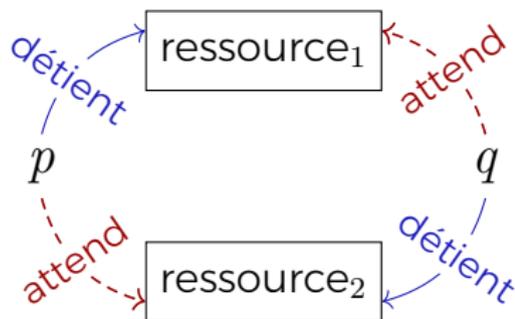
$p : x \leftarrow x + 5$

$q : x \leftarrow 2x$



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

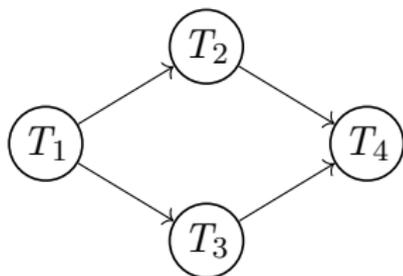
2. Blocages



<https://openclipart.org/>

L'état du système

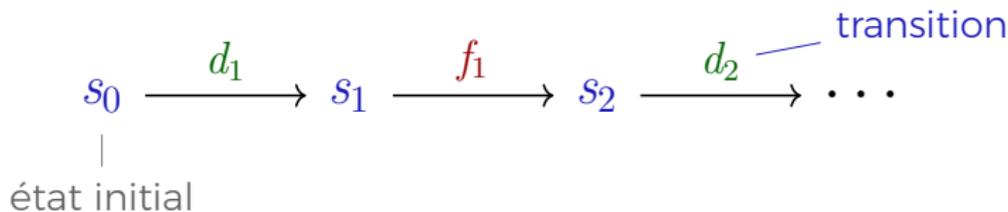
Système de tâches : $S = (E, <)$,



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

||
agit sur
⇓

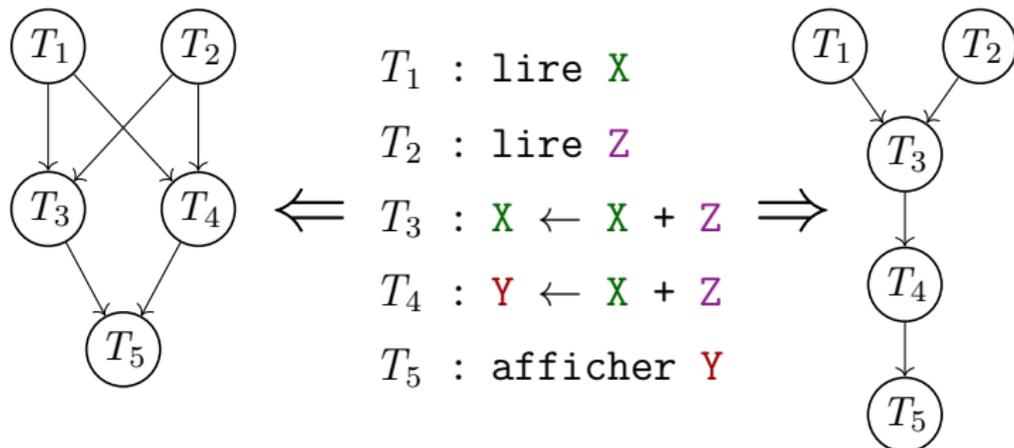
L'état du système = les informations, les ressources, etc.
associés au système



Outline

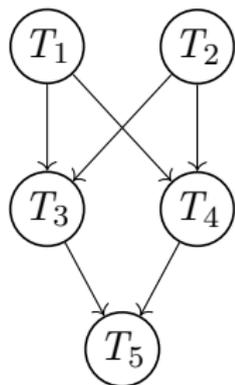
1. Relations entre processus
2. Problèmes associés à la communication
3. **Déterminisme et parallélisme maximal**
 - Caractère déterminé et la condition de Bernstein
 - Parallélisme maximal
4. Blocages
5. Définition des états du système
6. Définition du blocage
7. Prévention des blocages
8. Détection des blocages et reprise
9. Évitement des blocages

Décomposition en processus parallèles



Décomposition en processus parallèles

Système **indéterminé**



T_1 : lire X

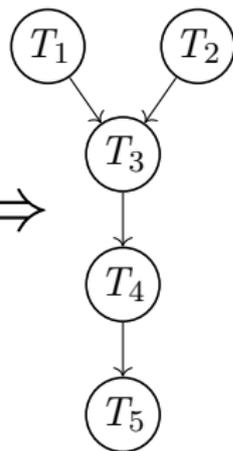
T_2 : lire Z

T_3 : $X \leftarrow X + Z$

T_4 : $Y \leftarrow X + Z$

T_5 : afficher Y

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.

mémoire

cellule

$$M = (C_1, C_2, \dots, C_n)$$

$$s_k = (C_1(k), C_2(k), \dots, C_n(k))$$

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: $w = d_1f_1 d_2f_2 d_3f_3 d_4f_4 d_5f_5$

	X	Y	Z
T_1 : lire X	0	0	0
T_2 : lire Z	0	0	0
T_3 : $X \leftarrow X + Z$	3	0	0
T_4 : $Y \leftarrow X + Z$	3	0	0
T_5 : afficher Y	3	0	7
	.	.	.

Suites des valeurs écrites dans les variables:

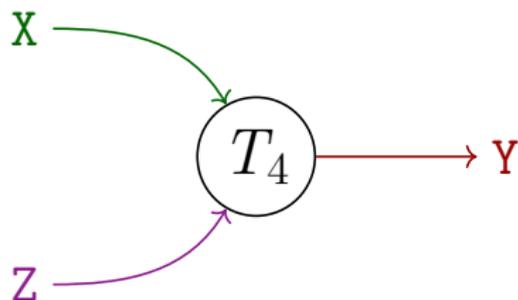
$V(X, w)$	0	3	10
$V(Y, w)$	0	17	
$V(Z, w)$	0	7	

Domaines de tâches

	Domaine de <u>L</u> ecture	Domaine d' <u>E</u> criture
T_1 : lire X	$L_1 = \emptyset$	$E_1 = \{X\}$
T_2 : lire Z	$L_2 = \emptyset$	$E_2 = \{Z\}$
T_3 : $X \leftarrow X + Z$	$L_3 = \{X, Z\}$	$E_3 = \{X\}$
T_4 : $Y \leftarrow X + Z$	$L_4 = \{X, Z\}$	$E_4 = \{Y\}$
T_5 : afficher Y	$L_5 = \{Y\}$	$E_5 = \emptyset$

Interprétation de tâches

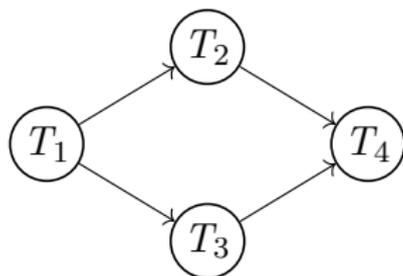
$$T_4 : \mathbf{Y} \leftarrow \mathbf{X} + \mathbf{Z} \quad L_4 = \{\mathbf{X}, \mathbf{Z}\} \quad E_4 = \{\mathbf{Y}\}$$



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

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

1. Relations entre processus
2. Problèmes associés à la communication
3. **Déterminisme et parallélisme maximal**
Caractère déterminé et la condition de Bernstein
Parallélisme maximal
4. Blocages
5. Définition des états du système
6. Définition du blocage
7. Prévention des blocages
8. Détection des blocages et reprise
9. Évitement des blocages

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 X , les mêmes suites de valeurs sont écrites dans X lors de l'exécution de w et de w' :

$$V(X, w) = V(X, w')$$

Le déterminisme et les conditions de Bernstein

Soit S un système de tâches.

- ▶ si toutes $T_1, T_2 \in S$ sont non-interférentes
 $\implies S$ – déterminé.
- ▶ si S – déterminé et toute tâche de S a le domaine d'écriture $\neq \emptyset$
 \implies toutes $T_1, T_2 \in S$ sont non-interférentes

Domaine d'écriture vide \implies interférences cachées possibles

Outline

1. Relations entre processus
2. Problèmes associés à la communication
3. **Déterminisme et parallélisme maximal**
Caractère déterminé et la condition de Bernstein
Parallélisme maximal
4. Blocages
5. Définition des états du système
6. Définition du blocage
7. Prévention des blocages
8. Détection des blocages et reprise
9. Évitement des blocages

Parallélisme maximal

Comment **paralléliser au maximum** un système ?

- ▶ construire le système de tâches **équivalent** avec le moins de contraintes de précedence

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 X , S_1 écrit la même suite de valeurs dans X que S_2

Parallélisme maximal : définition

Un système de tâches S est de **parallélisme maximal** si la **suppression** de tout arc $(T_1) \longrightarrow (T_2)$ 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 :

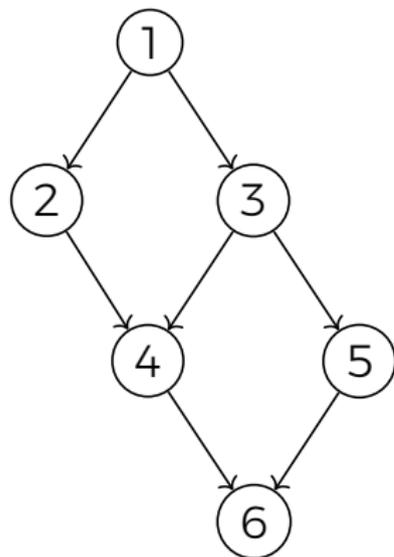
1. prendre toute paire de tâches T_1 et T_2 connectées par un chemin dans S

2. ajouter l'arc $(T_1) \longrightarrow (T_2)$ à S_{max} si

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

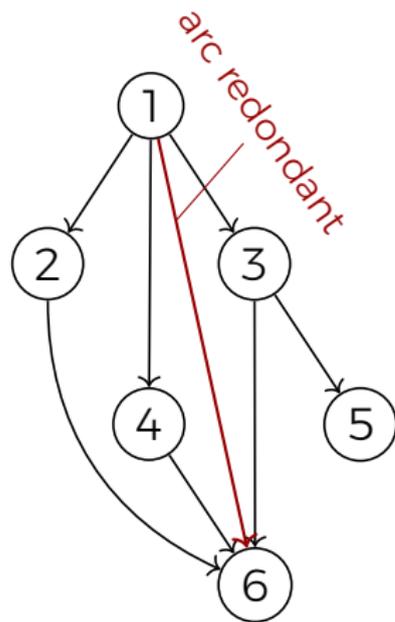
À la fin, éliminer les arcs redondants.

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

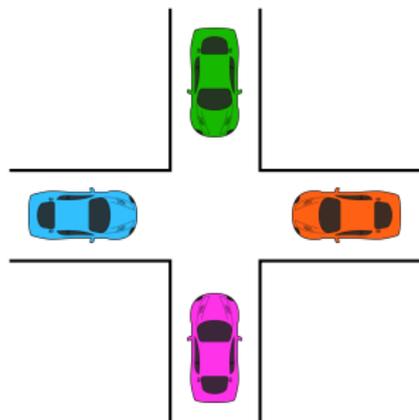
Outline

1. Relations entre processus
2. Problèmes associés à la communication
3. Déterminisme et parallélisme maximal
 - Caractère déterminé et la condition de Bernstein
 - Parallélisme maximal
4. Blocages
5. Définition des états du système
6. Définition du blocage
7. Prévention des blocages
8. Détection des blocages et reprise
9. Évitement des blocages

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**



1. Rattrapage des blocages

- ▶ laisser le blocage se produire, puis reprendre et redistribuer certaines ressources

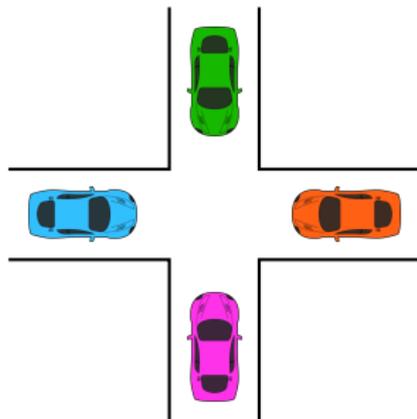
2. Prévention des blocages

- ▶ 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

1. Rattrapage : faire reculer l'une des voitures
2. Prévention : installer des panneaux  ou des  panneaux
3. Évitement : agent de police réglant la circulation



Outline

1. Relations entre processus
2. Problèmes associés à la communication
3. Déterminisme et parallélisme maximal
 - Caractère déterminé et la condition de Bernstein
 - Parallélisme maximal
4. Blocages
5. Définition des états du système
6. Définition du blocage
7. Prévention des blocages
8. Détection des blocages et reprise
9. Évitement des blocages

L'état du système après le pas k 1/2

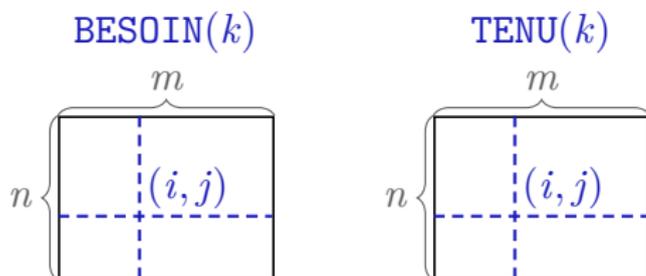
Système de n processus : P_1, P_2, \dots, P_n

Type de ressource : R_1 R_2 \dots R_m

Quantité disponible : N_1 N_2 \dots N_m

► capacité du système

État après le pas k :



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

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

L'état du système après le pas k 2/2

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

Quantité disponible : N_1 N_2 \dots N_m

$DISPO_j(k)$ = nombre de ressources R_j libres

- ▶ non détenues par personne

$DISPO_j(k) = N_j -$ instances détenues

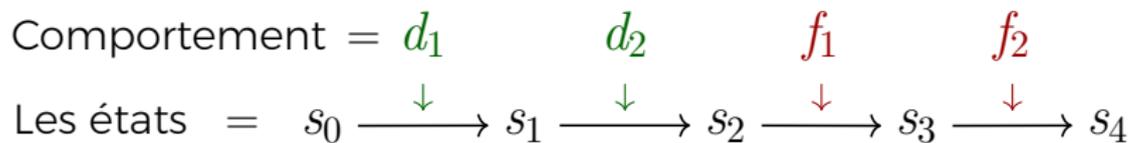
$$\sum_{i=1}^n \boxed{\text{TENU}_{i,j}(k)}$$

$DISPO(k) = (DISPO_1(k), \dots, DISPO_m(k))$

Outline

1. Relations entre processus
2. Problèmes associés à la communication
3. Déterminisme et parallélisme maximal
 - Caractère déterminé et la condition de Bernstein
 - Parallélisme maximal
4. Blocages
5. Définition des états du système
6. Définition du blocage
7. Prévention des blocages
8. Détection des blocages et reprise
9. Évitement des blocages

Définition du blocage ^{1/2}



s_4 — blocage si :

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

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

$$(1, 2) \leq (2, 3), \text{ mais } (1, 2) \not\leq (3, 1)$$

Définition du blocage ^{2/2}

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

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

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

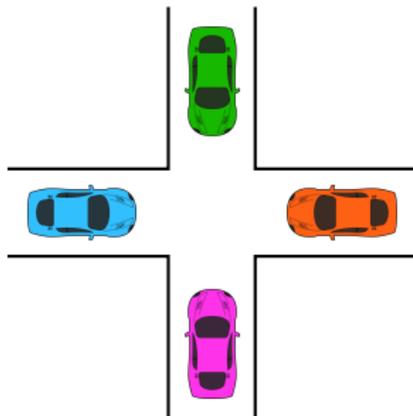
Définition du blocage ^{2/2}

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

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

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 $\frac{1}{n}$ 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

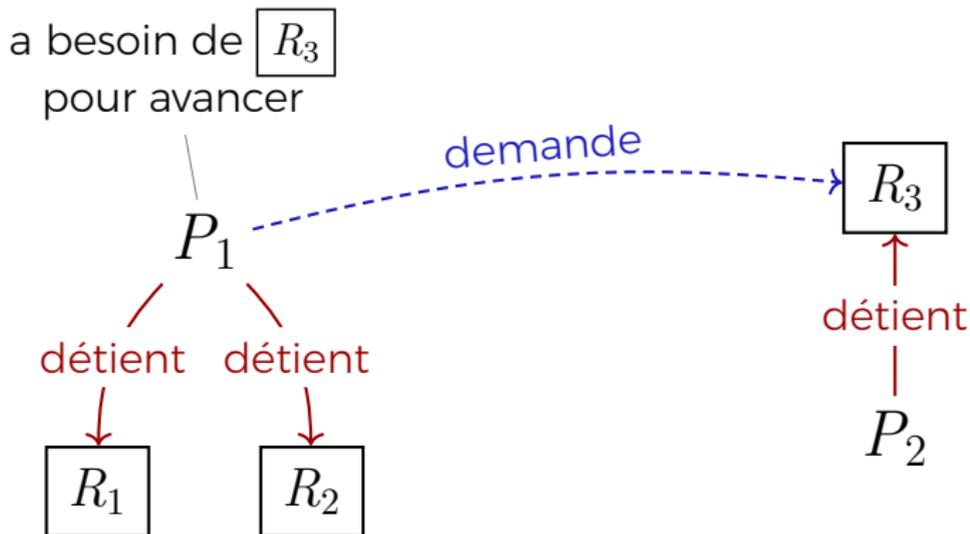
1. Relations entre processus
2. Problèmes associés à la communication
3. Déterminisme et parallélisme maximal
 - Caractère déterminé et la condition de Bernstein
 - Parallélisme maximal
4. Blocages
5. Définition des états du système
6. Définition du blocage
7. Prévention des blocages
8. Détection des blocages et reprise
9. Évitement des blocages

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.

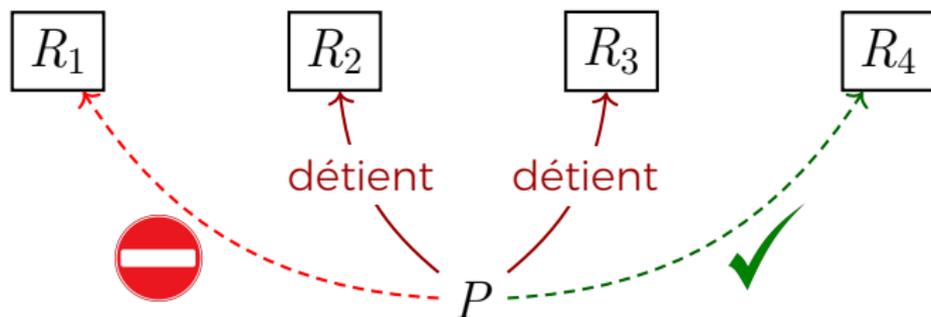
Empêcher les processus de « tenir et attendre »

- ▶ 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 \implies ralentissements

Empêcher l'attente circulaire

Utilisation ordonnée des ressources :



P_1 peut demander R_4 parce que $4 > 3$ et $4 > 2$.

P_1 ne peut pas demander 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

1. Relations entre processus
2. Problèmes associés à la communication
3. Déterminisme et parallélisme maximal
 - Caractère déterminé et la condition de Bernstein
 - Parallélisme maximal
4. Blocages
5. Définition des états du système
6. Définition du blocage
7. Prévention des blocages
8. Détection des blocages et reprise
9. Évitement des blocages

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 :

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

- ▶ 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

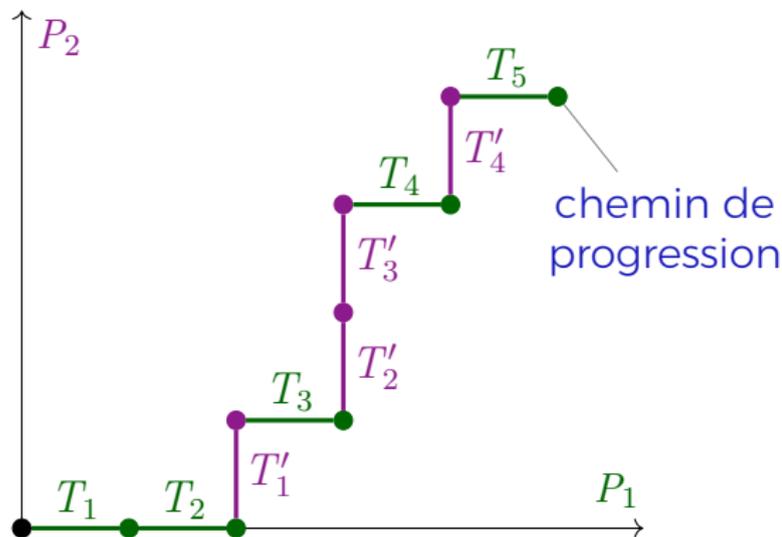
1. Relations entre processus
2. Problèmes associés à la communication
3. Déterminisme et parallélisme maximal
 - Caractère déterminé et la condition de Bernstein
 - Parallélisme maximal
4. Blocages
5. Définition des états du système
6. Définition du blocage
7. Prévention des blocages
8. Détection des blocages et reprise
9. Évitement des blocages

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 : $T_1 T_2 T'_1 T_3 T'_2 T'_3 T_4 T'_4 T_5$

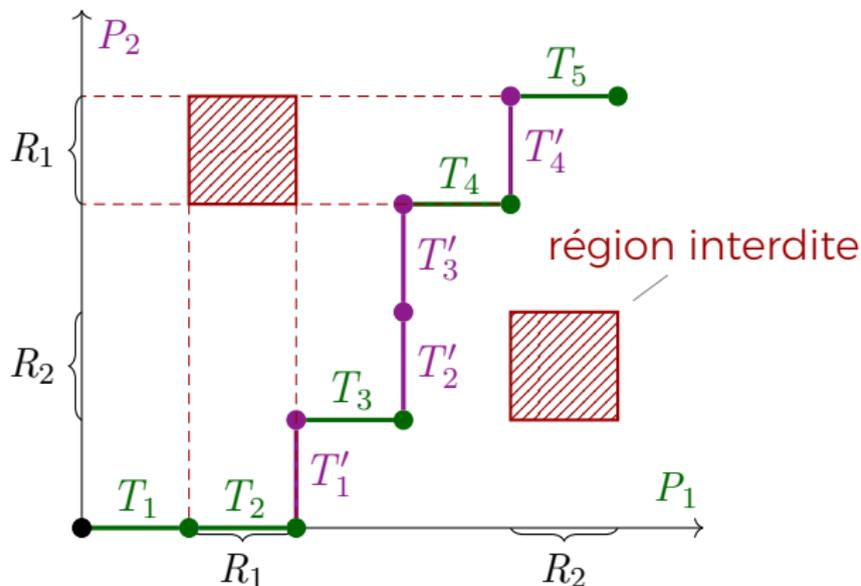


Régions interdites

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

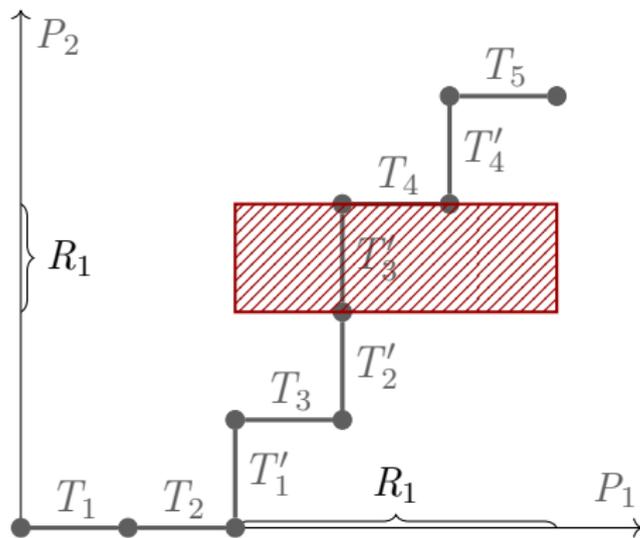


Régions dangereuses

Supposons les besoins en ressources suivants:

▶ $T_3, T_4, T_5 : R_1$

▶ $T'_3 : R_1$

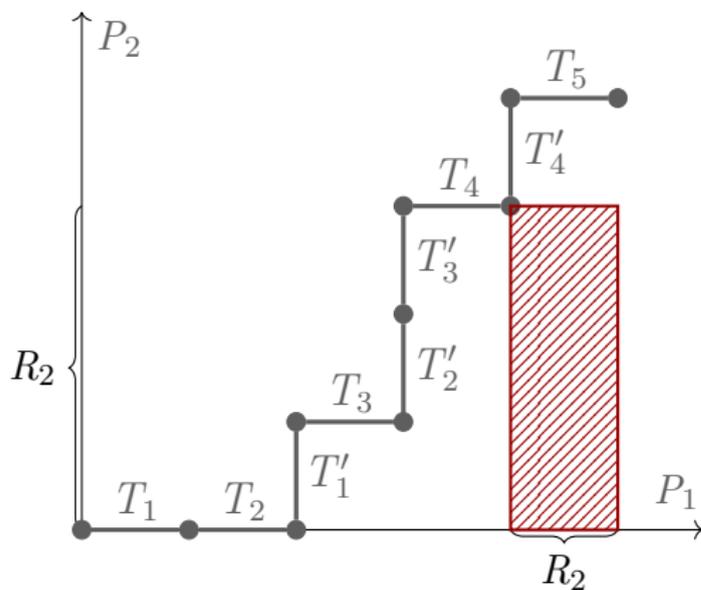


Régions dangereuses

Supposons les besoins en ressources suivants:

► $T_5 : R_2$

► $T'_1, T'_2, T'_3 : R_2$



Régions dangereuses

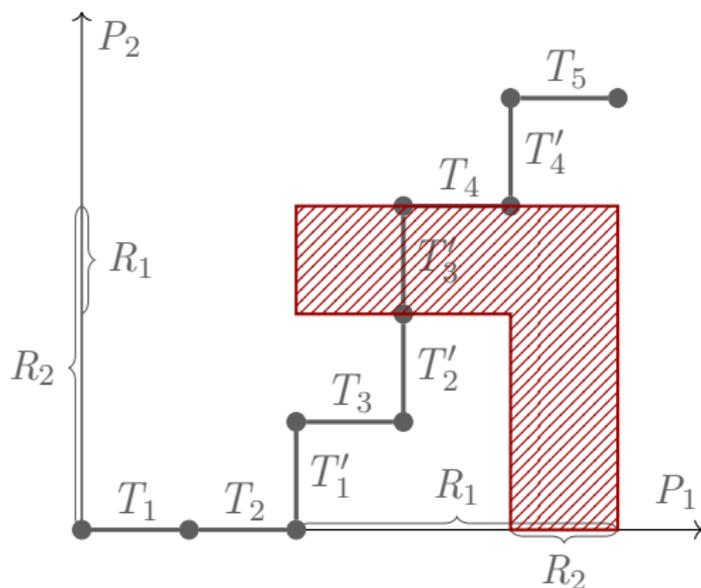
Supposons les besoins en ressources suivants:

▶ $T_3, T_4, T_5 : R_1$

▶ $T_5 : R_2$

▶ $T'_3 : R_1$

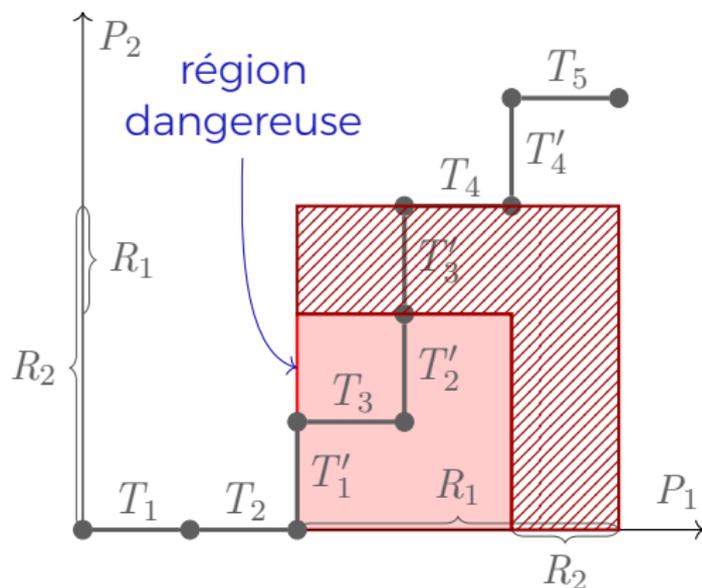
▶ $T'_1, T'_2, T'_3 : R_2$



Régions dangereuses

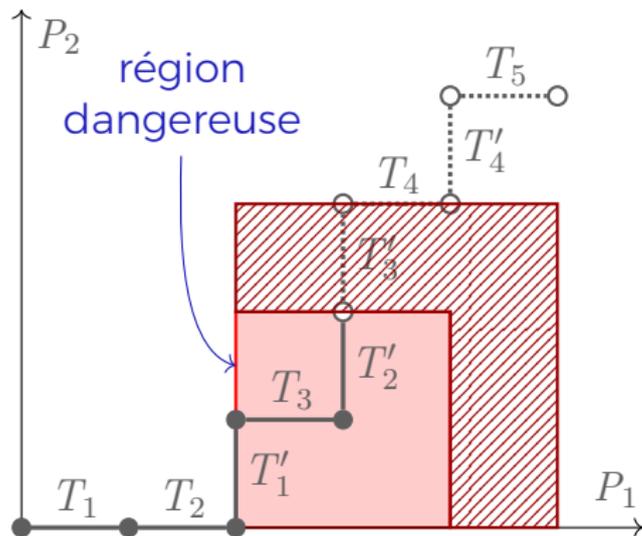
Supposons les besoins en ressources suivants:

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



États sûrs

Chemin dans une région dangereuse \Rightarrow 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 \implies **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 :

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

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** :

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

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

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

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

Si t_{k+1} n'est pas un blocage $\implies s_{k+1}$ est sûr
 \implies on peut dire oui à P