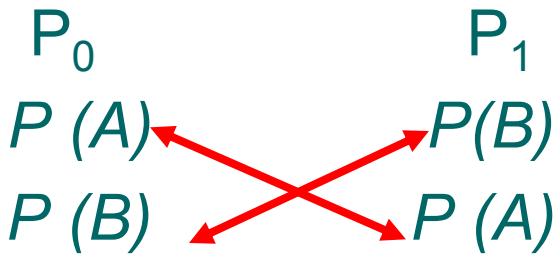
Interblocage = impasse (Deadlock)

Interblocages: concepts importants

- Caractérisation: les 4 conditions
- Graphes allocation ressources
- Séquences de terminaison
- États sûrs et no-sûrs
- Prévenir les interblocages
- Éviter les interblocages
- Détecter les interblocages
- Récupérer un interblocage

Exemple

- **Sémaphores**



- **Scénario d'interblocage:**
  - initialisation de A et B à 1
  - P0 exécute P(A), A=0
  - P1 exécute P(B), B=0
  - P0 et P1 ne peuvent pas aller plus loin
    - **Qu'arrive au lieu si P0 exécute entièrement avant P1?**

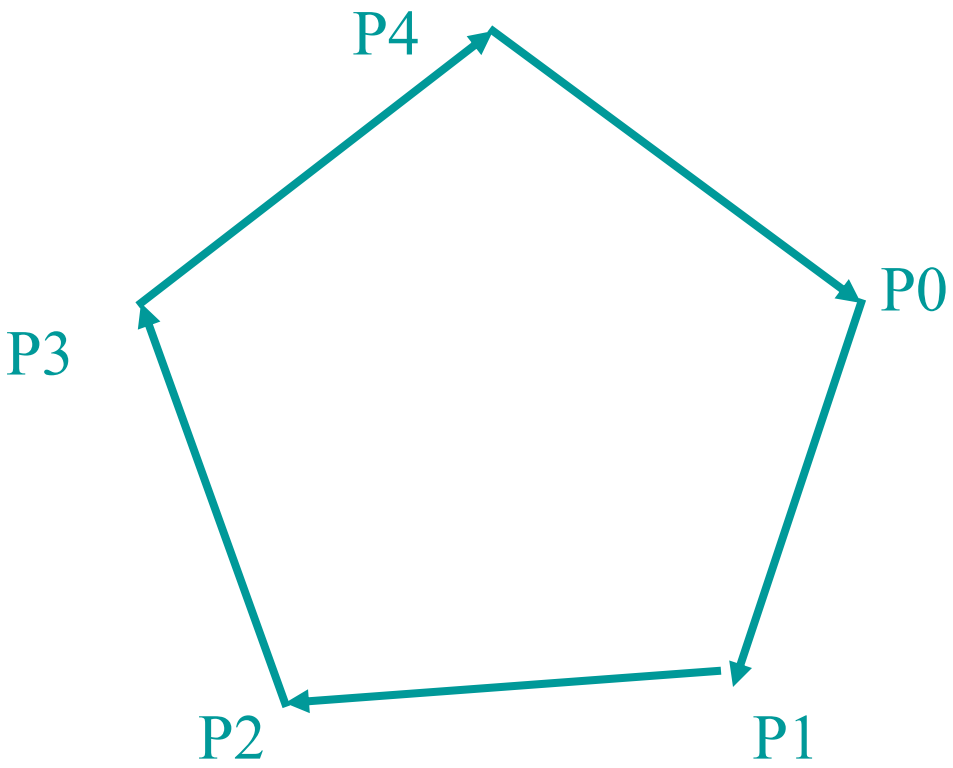
Définition (Tanenbaum)

- **Un ensemble de processus est en *interblocage* si chaque processus attend un événement que seul un autre processus de l'ensemble peut provoquer**
- **L'événement est une *libération de ressource***
  - Prenant ce mot dans le sens le plus vaste: ressource peut être un signal, un message,un sémaphore, etc.

Caractérisation d'interblocage

- **L'interblocage demande la présence simultanée de 4 conditions (conditions nécessaires)**
  - **Exclusion mutuelle**: le système a des ressources non partageables (1 seul proc à la fois peut s'en servir)
    - Ex.: UCT, zone de mémoire, périphérique, mais aussi sémaphores, moniteurs, sections critiques
  - **Saisie et attente** (hold and wait): un processus a saisi une ressource non partageable et attend d' autresressources détenues par les autres pour compléter sa tâche
  - **Pas de préemption**: un processus qui a saisi une ressource non partageable la garde jusqu'à ce qu'il aura complété sa tâche
  - **Attente circulaire**: il y a un cycle de processus tel que chaque processus pour compléter doit utiliser une ressource non partageable qui est utilisée par le suivant, et que le suivant gardera jusqu`à sa terminaison
    - **En présence des 3 premières conditions, une attente circulaire est un interblocage**
    - **Les 3 premières conditions n'impliquent pas nécessairement interblocage, car l'attente circulaire pourrait ne pas se réaliser**

Attente circulaire - aucun ne lâche - aucun processus ne peut terminer donc interblocage



Pour terminer, chaque processus doit saisir une ressource que le prochain ne lâchera pas → interblocage

Exercice

- Considérez un système dans lequel chaque processus n'a besoin que d'une seule ressource pendant toute son existence

- L'interblocage, est-il possible?

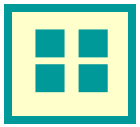
Graphes d'allocation ressources

- **Un ensemble de sommets V et d'arêtes E**
- **V est partitionné dans:**
  - $P = \{P_1, P_2, \ldots, P_n\}$, l'ensemble qui consiste de tous les processus dans le système
  - $R = \{R_1, R_2, \ldots, R_m\}$, l'ensemble des ressources dans le système
- **arête requête – arête dirigée $P_i \rightarrow R_k$**
- **arête affectation – arête dirigée $R_i \rightarrow P_k$**

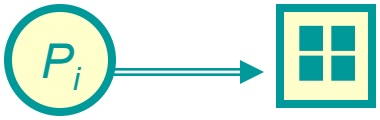
Graphe d'allocation ressources

- **Processus**

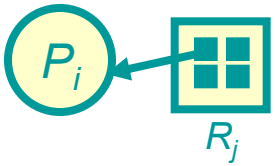
- **Ressource dont il y a 4 exemplaires (instances)**



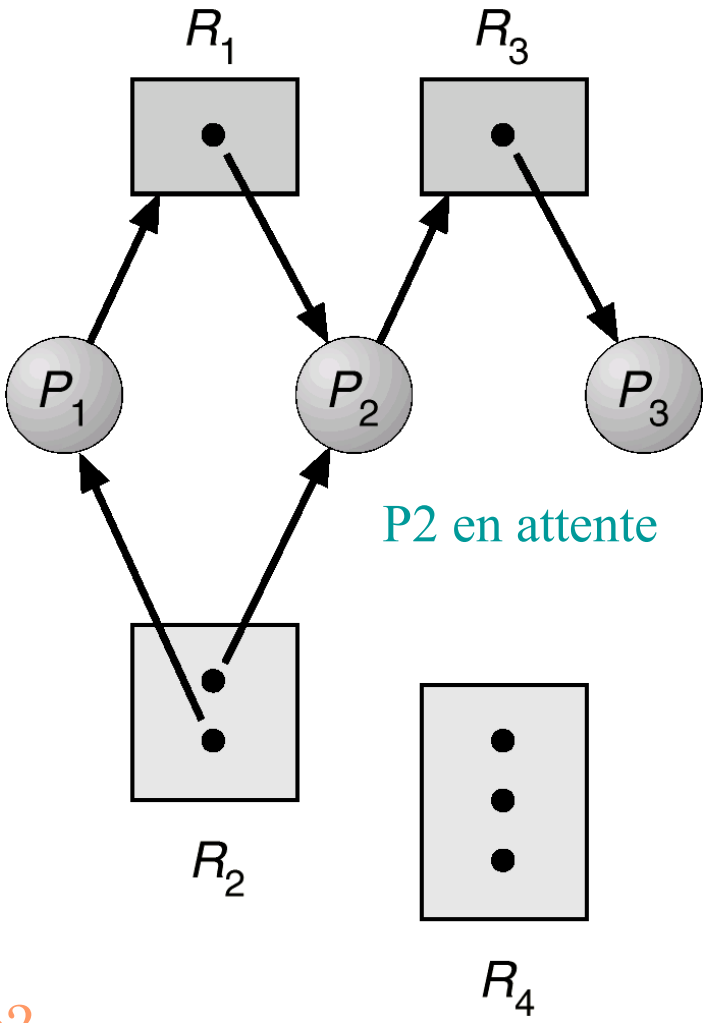
- **Pi attend un exemplaire de Ri, dont il y en a 4**



- **Pj a saisi (et utilise) un exemplaire de Rj**



Exemple de graphe allocation ressources



P1 en attente

P3 pas en attente

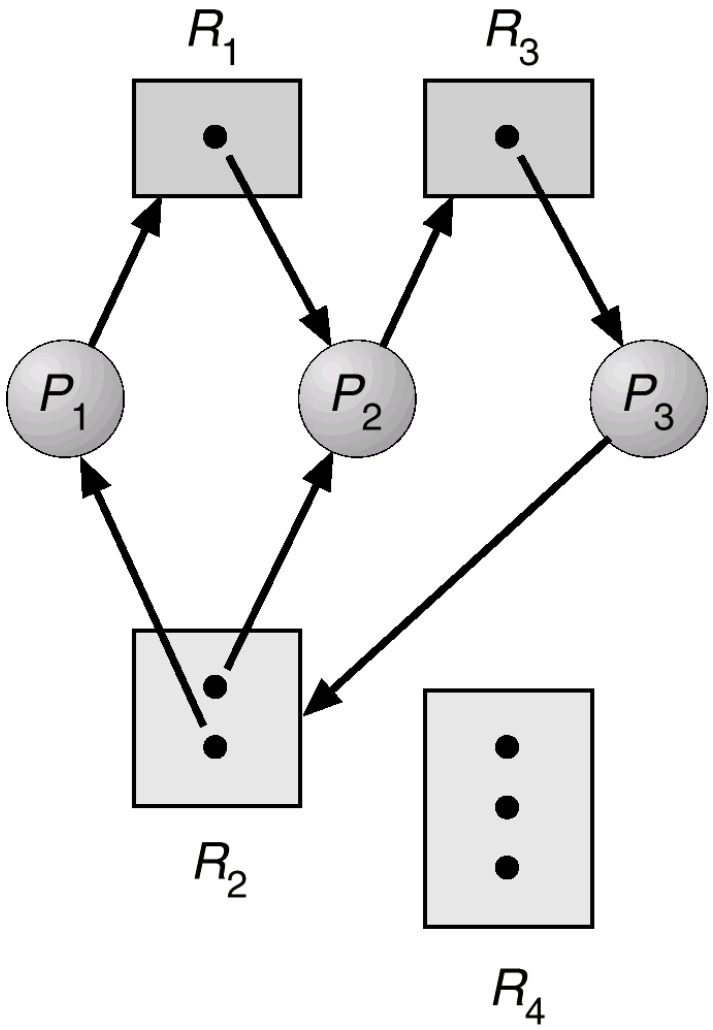
P2 en attente

Y-a-t-il interblocage?

Utilisation de ces graphes

- **Nous supposons l'existence des 3 premières conditions**
  - Excl. Mutuelle, saisie et attente, pas de préemption
- **Pour montrer qu'il n'y a pas d'interblocage, nous devons montrer qu'il n'y a pas de cycle, car il y a un processus qui peut terminer sans attendre aucun autre, et puis les autres de suite**
- **<P_3, P_2, P_1> est un ordre de terminaison de processus: tous peuvent terminer dans cet ordre**

Graphe allocation ressources avec interblocage



Cycles:

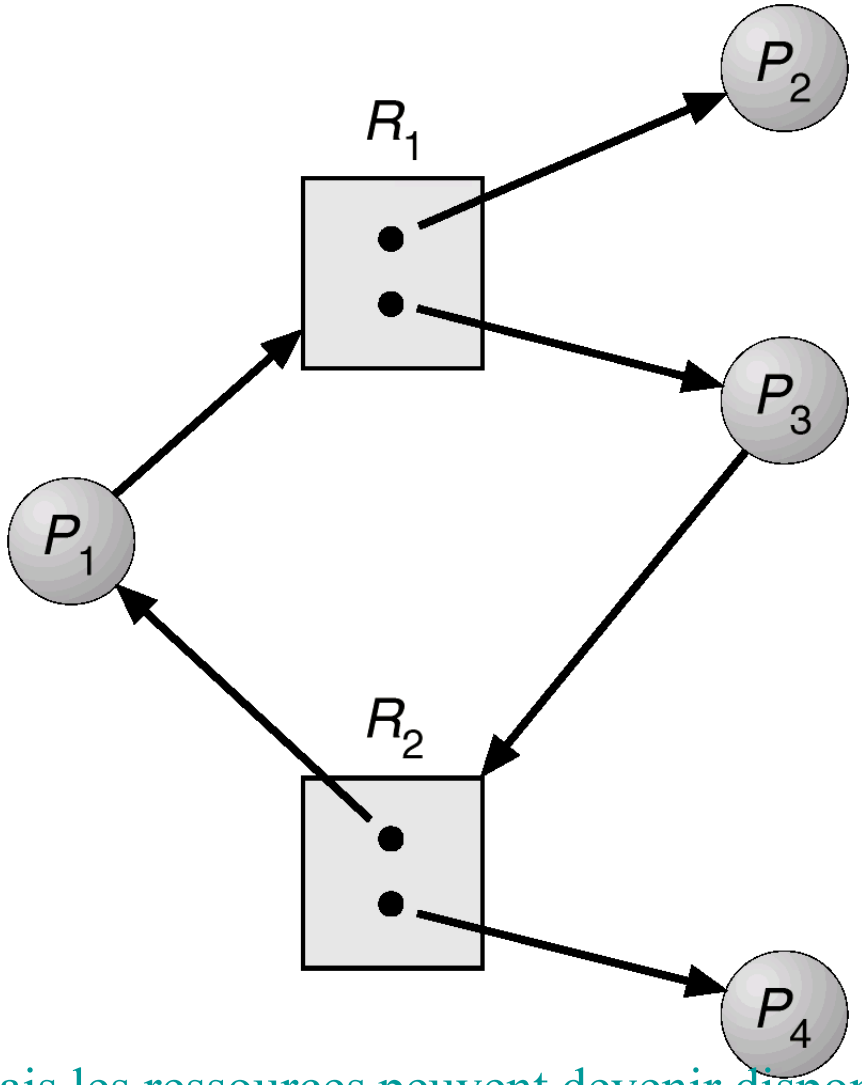
P1 → R1 → P2 → R3 → P3 → R2 → P1

P2 → R3 → P3 → R2 → P2

aucun proc ne peut terminer

aucune possibilité d'en sortir

Graphe allocation ressources avec cycle, mais pas d' interblocage (pourquoi?)



Attente circulaire, mais les ressources peuvent devenir disponibles

Constatations

- **Les cycles dans le graphe alloc ressources ne signalent pas nécessairement une attente circulaire**
- **S ’il n`y a pas de cycles dans le graphe, aucun interblocage**
- **S ’il y a de cycles:**
  - Si seulement une ressource par type, interblocage
    - (pourquoi?!)
  - Si plusieurs ressources par type, possibilité d’interblocage
    - Il faut se poser la question: y-a-t-il un processus qui peut terminer et si oui, quels autres processus peuvent terminer en conséquence?

Hypothèse de terminaison

- **Un proc qui a toutes les ressources dont il a besoin, il s'en sert pour un temps fini, puis il les libère**
- **Nous disons que le processus termine, mais il pourrait aussi continuer, n'importe, l'important est qu'il laisse la ressource**
- **Il n'y a pas d'interblocage si tous les processus peuvent terminer, un à la fois**
  - Laisser leurs ressources

Méthodes pour traitement interblocage

- **Prévenir: concevoir le système de façon qu'un interblocage soit impossible (difficile)**
- **Éviter: les interblocages sont possibles, mais sont évités (avoidance)**
- **Détecter et récupérer: Permettre les interblocages, en récupérer**
- **Ignorer le problème, qui donc doit être résolu par le gérant ou l'usager**
  - malheureusement, méthode d 'utilisation générale!

Prévention d'interblocage: prévenir au moins une des 4 conditions nécessaires

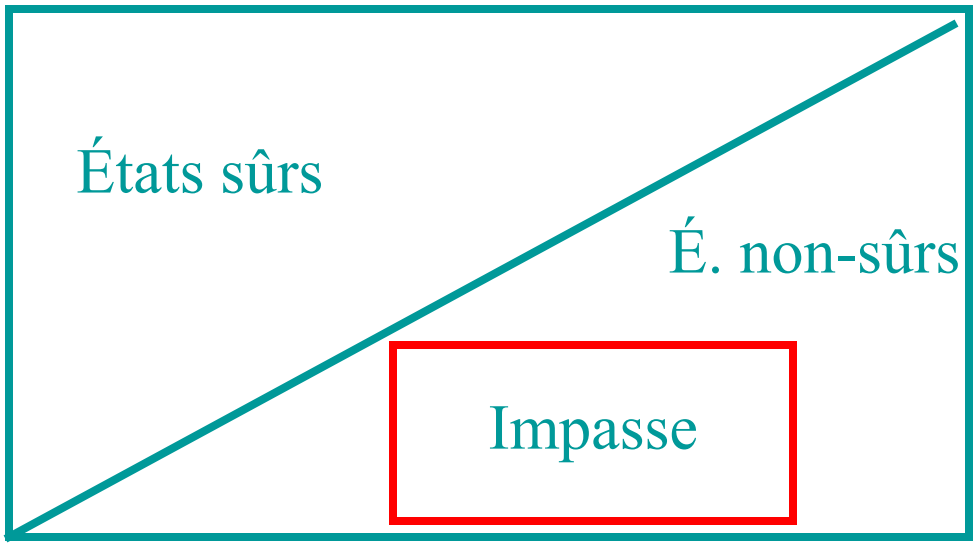
- Exclusion mutuelle: **réduire le plus possible l'utilisation des ressources partagées et Sections Critiques**
- Saisie et attente **(hold and wait): un processus qui demande des nouvelles ressources ne devrait pas en retenir des autres (les demander toutes ensemble)**
- Pas de préemption: **si un processus qui demande d'autres ressources ne peut pas les avoir, il doit être suspendu, ses ressources doivent êtres rendues disponibles**
- Attente circulaire: **imposer un ordre partiel sur les ressources, un processus doit demander les ressources dans cet ordre.**

Éviter les interblocages (deadlock avoidance)

- Chaque processus doit déclarer le nombre max. de ressources dont il prévoit avoir besoin
- L'algorithme examine toutes les séquences d'exécution possibles pour voir si une attente circulaire est possible

État sûr (safe state)

- **Un état est sûr si le système peut terminer sans interblocages**
- **Ne pas allouer une ressource à un processus si l'état qui en résulte n'est pas sûr**



État sûr

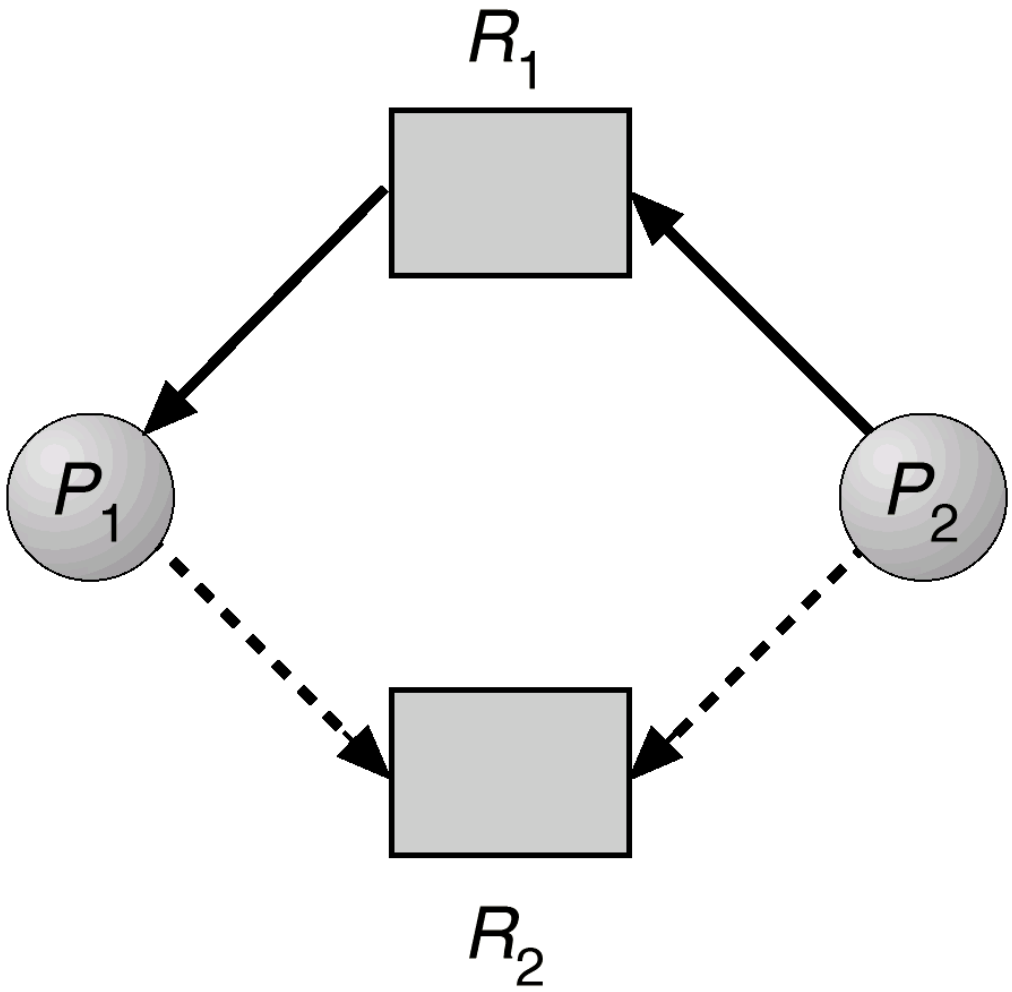
- **Une séquence de proc <P_1, P_2, …, P_n> est sûre si pour chaque Pi, les ressources que Pi peut encore demander peuvent être satisfaites par les ressources couramment disponibles + ressources utilisées par *les Pj qui les précèdent*.**
  - Quand Pi aboutit, Pi+1 peut obtenir les ressources dont il a besoin, terminer, donc
- **<P_1, P_2, …, P_n> est un ordre de terminaison de processus: tous peuvent se terminer dans cet ordre**

Algorithme d'allocation de ressources

- **Il faut maintenant prendre en considération:**
  - les requêtes possibles dans le futur (chaque processus doit déclarer ça)

- **Arête demande P_i - - > R_j indique que le processus P_i peut demander la ressource R_j (ligne à tirets)**

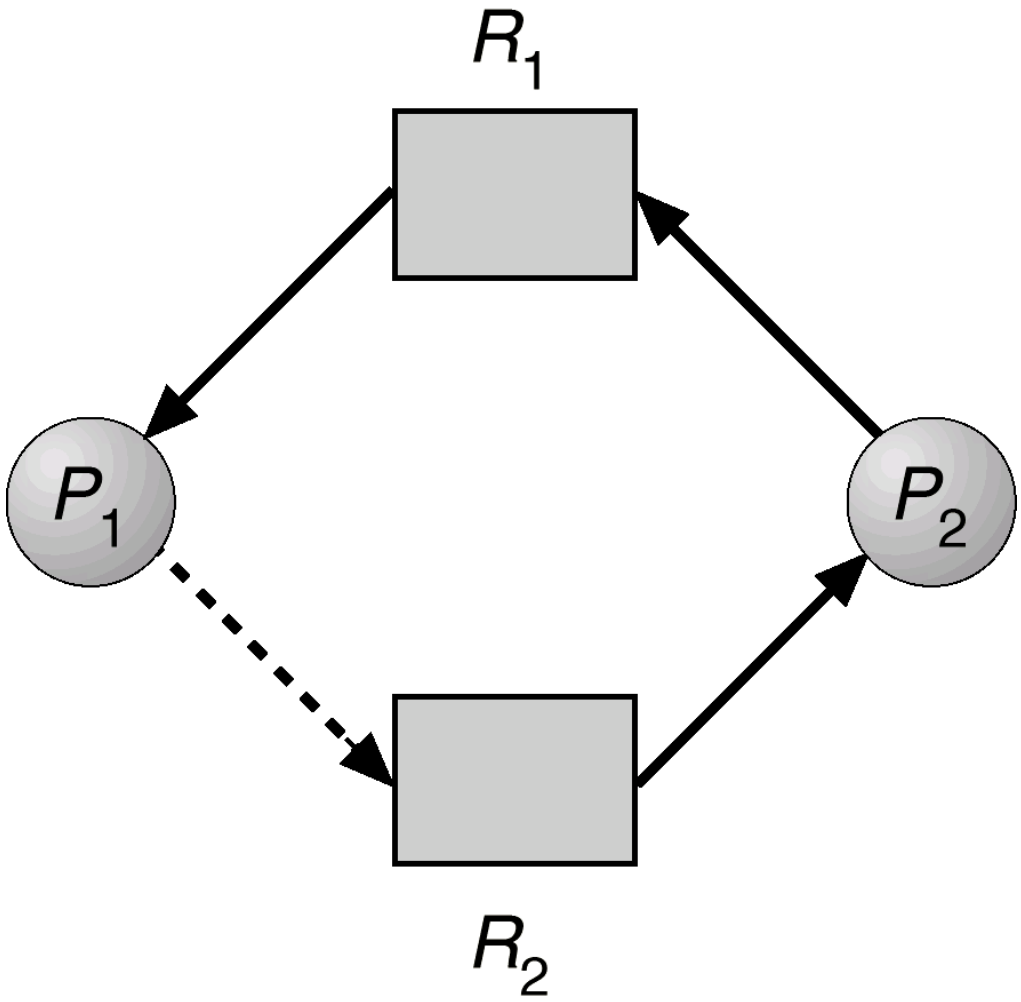
Graphe d`allocation ressources



Ligne continue: requête courante;

tirets: requête possible dans le futur

Un état pas sûr



Si P2 demande R2, ce dernier ne peut pas lui être donné, car ceci peut causer un cycle dans le graphe: P1 req R2,

Refus d'allouer une ressource: l'algorithme du banquier

- Les processus sont comme des clients qui désirent emprunter de l'argent (ressources) à la banque...
- Un banquier ne devrait pas prêter de l'argent s'il ne peut pas satisfaire les besoins de tous ses clients
- En tout temps l'état du système est défini par les valeurs de R(i), C(j,i) pour tout type i et processus j, et par d'autres valeurs de vecteurs et matrices.

L'algorithme du banquier

- **Nous devons aussi connaître la quantité allouée A(j,i) de ressources de type i au processus j**
- **La quantité totale de ressource de type i disponible est donnée par: $V(i) = R(i) - \Sigma_k A(k,i)$**
- **N(j,i) est la quantité de ressources i requise (restant) par le processus j pour terminer sa tâche: N(j,i) = C(j,i) - A(j,i)**
- **Pour décider si la requête doit être accordée, l'algorithme du banquier teste si cette allocation conduira le système dans un état prudent (safe state):**
  - accorder la requête si l'état est prudent
  - sinon refuser la requête

L’algorithme du banquier

- **Un état est prudent ssi il existe une séquence {P1..Pn} où chaque Pi est alloué toutes les ressources dont il a besoin pour terminer**
  - ie: nous pouvons toujours exécuter tous les processus jusqu’à terminaison à partir d’un état prudent

L'algorithme du banquier

- **Q(j,i) est la quantité ressource i demandé par le processus j lors d'une requête.**
- **Pour déterminer si cette requête doit être accordée, nous utilisons l'algorithme du banquier:**
  - Si Q(j,i) <= N(j,i) pour tout i alors continuer. Sinon rapporter l'erreur (montant réclamé dépassé).
  - Si Q(j,i) <= V(i) pour tout i alors continuer. Sinon attendre (ressource pas encore disponible)
  - Prétendre que la requête est accordée et déterminer le nouvel état:

L'algorithme du banquier

- V(i) = V(i) - Q(j,i) pour tout i
- A(j,i) = A(j,i) + Q(j,i) pour tout i
- N(j,i) = N(j,i) - Q(j,i) pour tout i

- Si l'état résultant est prudent, alors accorder la requête. Sinon le processus j doit attendre pour sa requête Q(j,i); rétablir l'état précédent.

- **L'algorithme de prudence est la partie qui détermine si un état est prudent**
- **Initialisation:**
  - tous les processus sont "non terminés"
  - le vecteur de travail contient d'abord les ressources disponibles: W(i) = V(i) ; pour tout i;

L'algorithme du banquier

- **REPEAT: Choisir un processus j "non terminé" tel que N(j,i) <= W(i) pour tout i.**
  - Si un tel j n'existe pas, goto EXIT
  - Sinon: "terminer" ce processus et récupérer toutes ses ressources: W(i) = W(i) + A(j,i) pour tout i. Ensuite goto REPEAT
- **EXIT: Si tous les processus ont "terminés" alors cet état est prudent. Sinon il est imprudent.**

Algorithme du banquier: exemple

- **Nous avons 3 types de ressources avec les quantités suivantes:**
  - R(1) = 9, R(2) = 3, R(3) = 6
- **et 4 processus avec l'état initial:**

	Réclamées			Allouées			disponibles		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	3	2	2	1	0	0	1	1	2
P2	6	1	3	5	1	1			
P3	3	1	4	2	1	1			
P4	4	2	2	0	0	2			

  - **Supposer que la requête de P2 est Q = (1,0,1). Doit-elle être accordée?**

Algorithme du banquier: exemple

- **L'état résultant serait:**

	Réclamées			Allouées			disponibles		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	3	2	2	1	0	0	0	1	1
P2	6	1	3	6	1	2			
P3	3	1	4	2	1	1			
P4	4	2	2	0	0	2			

- **Cet état est prudent avec la séquence {P2, P1, P3, P4}. Après P2, nous avons W = (6,2,3) ce qui permet à tous les autres processus de terminer. La requête est donc accordée**

Algorithme du banquier: exemple

- Cependant si, de l'état initial, P1 requiert Q = (1,0,1). L'état résultant serait:

	Réclamées			Allouées			disponibles		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	3	2	2	2	0	1	0	1	1
P2	6	1	3	5	1	1			
P3	3	1	4	2	1	1			
P4	4	2	2	0	0	2			

- Cet état n'est pas prudent car, pour terminer, chaque processus nécessite une unité de R1. La requête est refusée: P1 est bloqué.

Détection d ’interblocage

- On permet au système d’entrer en un état d’interblocage
- L’interblocage est détecté
- On récupère de l’interblocage

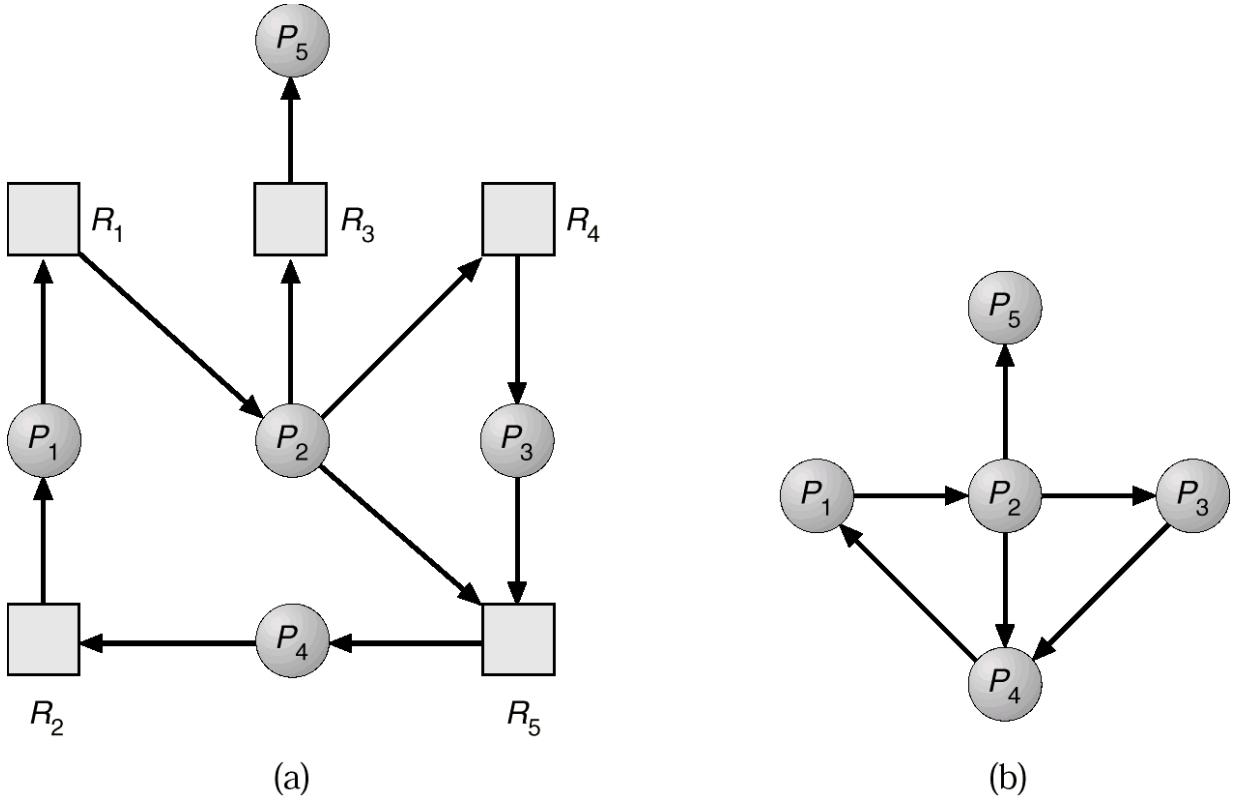
Différence entre attente et interblocage

- **Il est difficile de détecter s'il y a effectivement un interblocage dans le système**
- **Nous pourrions voir qu'un certain nombre de processus est en attente de ressources**
  - ceci est normal!
- **Pour savoir qu'il y a interblocage, il faut savoir qu'aucun processus dans un groupe n'a de chance de recevoir la ressource**
  - car il y a attente circulaire!
- **Ceci implique une analyse additionnelle, que peu de SE se prennent la peine de faire...**

Méthode de détection d'interblocage dans le cas d'une ressource par type

- **Essentiellement, la méthode déjà décrite**
  - Construire un graphe d'allocation ressources et voir s'il y a une manière dont tous les proc peuvent terminer
- **Dans le cas d'une ressource par type, l`algorithme cherche des cycles dans le graphe**
- **Plus difficile dans le cas de plusieurs ressources par type**

Graphe allocation ressources et graphe d 'attente
(cas d'1 ressource par type)



Un algorithme de détection d'interblocage

- **Utilisez les matrices et vecteurs précédents pour l'allocation des ressources**
- **Marquer chaque processus non impliqué dans un interblocage. Initialement tous les processus sont sans marque (possiblement impliqués). Alors effectuer:**
  - Marquer chaque processus j pour lequel: A(j,i) = 0 pour tout i. (puisqu'ils ne peuvent pas être impliqués)
  - Initialiser le vecteur de travail: W(i) = V(i) pour tout i
  - REPEAT: Choisir un processus j non marqué tel que Q(j,i) <= W(i) pour tout i. Arrêter si un tel j n'existe pas.
  - Si un tel j existe: marquer le processus j et faire W(i) = W(i) + A(j,i) pour tout i. Goto REPEAT
  - À la fin: chaque processus non marqué est impliqué dans un interblocage.

Détection d'impasse: commentaires

- Processus j n'est pas impliqué dans une impasse lorsque Q(j,i) <= W(i) pour tout i.
- Nous sommes alors optimistes et assumons que le processus j ne demandera pas plus de ressources pour terminer sa tâche
- Il libérera alors toutes ses ressources. Alors: W(i) = W(i) + A(j,i) pour tout i
- Si cette supposition est incorrecte, l'impasse pourrait survenir plus tard
- Cette impasse sera détectée la prochaine fois que l'algorithme de détection sera invoqué

Détection d'impasse: exemple

	Réclamées					Allouées					disponibles				
	R1	R2	R3	R4	R5	R1	R2	R3	R4	R5	R1	R2	R3	R4	R5
P1	0	1	0	0	1	1	0	1	1	0	0	0	0	0	1
P2	0	0	1	0	1	1	1	0	0	0					
P3	0	0	0	0	1	0	0	0	1	0					
P4	1	0	1	0	1	0	0	0	0	0					

- Marquer P4 car il n'a pas de ressources alloués
- Faire W = (0,0,0,0,1)
- La requête de P3 <= W. Alors marquer P3 et faire W = W + (0,0,0,1,0) = (0,0,0,1,1)
- L'algorithme termine. P1 et P2 sont impliqués dans une impasse

Récupérer d'interblocages

- **Terminer tous les processus dans l'interblocage**
- **Terminer un processus à la fois, espérant d'éliminer le cycle d'interblocages**
- **Dans quel ordre: différents critères:**
  - priorité
  - besoin de ressources: passé, futur
  - combien de temps il a exécuté, de combien de temps il a encore besoin
  - etc.

Récupération: préemption de ressources

- **Minimiser le coût de sélectionner la victime**
- **Rollback: retourner à un état sûr**
  - besoin d`établir régulièrement et garder des ‘points de reprise’ sortes de photos de l ’état courant du processus
- **Famine possible si un processus est toujours sélectionné**

Combinaison d'approches

- **Combiner les différentes approches, si possible, en considération des contraintes pratiques**
  - prévenir
  - éviter
  - détecter
- **utiliser les techniques les plus appropriées pour chaque classe de ressource**