4. Interblocages

F. Boyer, UJF- Laboratoire Lig
Fabienne.Boyer@imag.fr

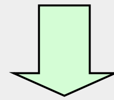
- Le problème
- Exemples
- Caractérisation des interblocages
- Traitement des interblocages
  - Prévention
  - Détection et guérison
  - Evitement

Ce cours a été conçu à partir de…

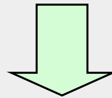
- **Cours de E. Berthelot**
  - http://www.iie.cnam.fr/%7EBerthelot/
- **Cours de A. Sylberschatz**
  - www.sciences.univ-nantes.fr/info/perso/permanents/attiogbe/SYSTEME/CoursSysteme.html
- **Cours de A. Griffaut**
  - http://dept-info.labri.fr/~griffault/Enseignement/SE/Cours
- **Cours de H. Bouzourfi, D. Donsez**
  - http://www-adele.imag.fr/~donsez/cours/#se

Le problème

Processus concurrents et synchronisés



Partage de ressources



Interblocages possibles

→ Problème posé au concepteur de système et au programmeur d'applications concurrentes

Exemple d'interblocage dans le Producteur / Consommateur

Processus Producteurs et consommateurs en exclusivité totale:

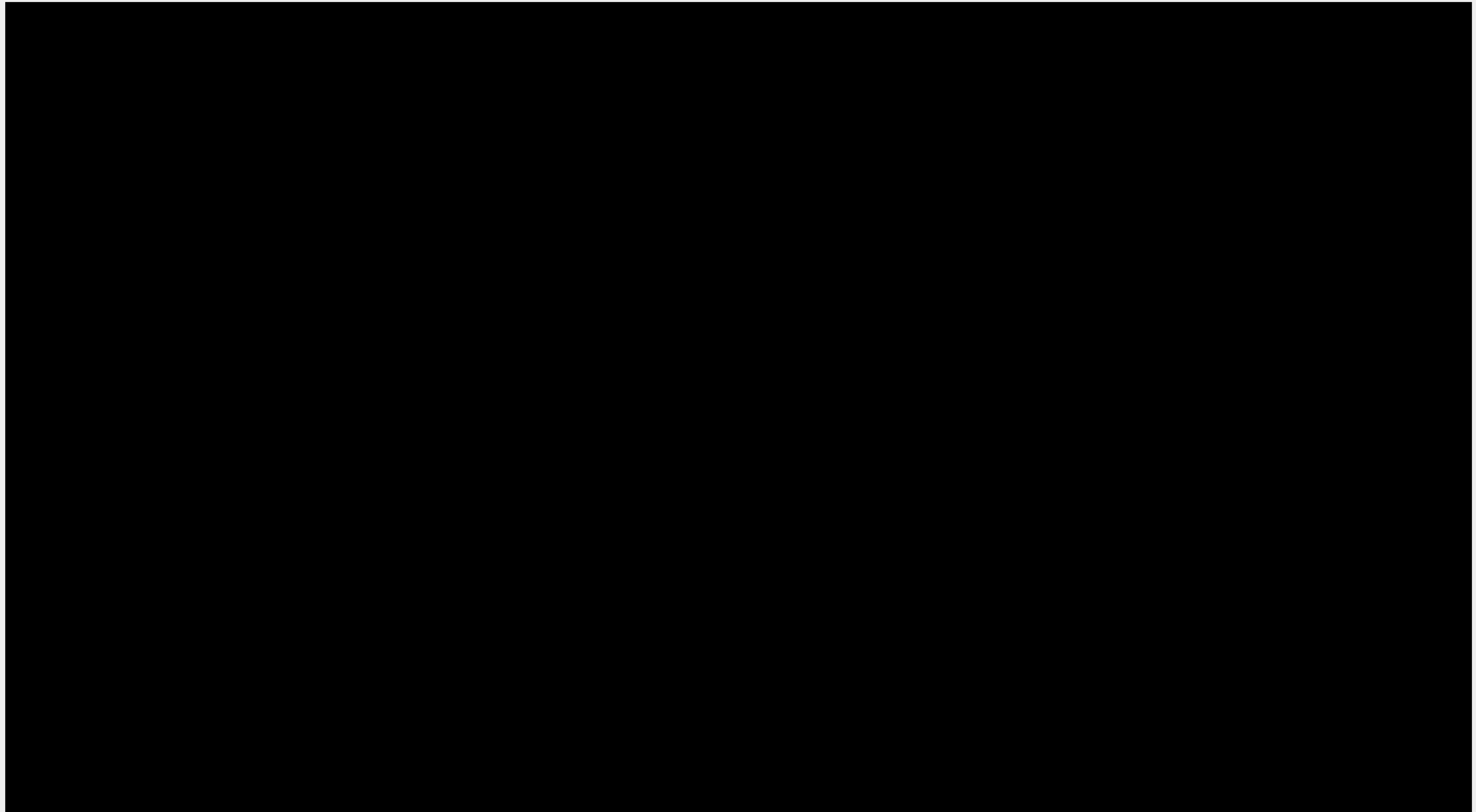
```
produce(Msg msg) {
        mutex.P();
        notFull.P();
        buffer[in++]=msg;
        notEmpty.V();
        mutex.V();
}
```

```
Msg consume() {
        mutex.P();
        notEmpty.P();
        msg = buffer[out++];
        notFull.V();
        mutex.V();
}
```

Exemple d'interblocage dans l'application bancaire

- **Si le grain de partage est la base :**
  - Toute la base est verrouillée pour une opération bancaire
  - Innefficace
- **Si le grain de partage est le compte :**
  - Un sémaphore par compte
  - Attention aux interblocages

Exemple de l'application bancaire



Caractérisation des interblocages

- Ensemble de ressources
  → partageables ou non
  → réquisitionnables ou non

- Chaque ressource est dans l'état :
  - libre
  - verrouillée en mode exclusif
  - verrouillée en mode partagé

- Chaque processus utilise une ressource comme suit :
  - demande (si conflit → attente)
  - utilisation
  - libèration

Caractérisation des interblocages

Etat / *Demande*	Libre	Verrou partagé	Verrou exclusif
Partagé	oui	oui	conflit
Exclusif	oui	conflit	conflit

Caractérisation des interblocages

- **4 conditions nécessaires pour l'apparition d'un interblocage**

  1) ***Exclusion mutuelle:*** **ressources non partageables, ou pouvant générer un conflit**

  2) ***Tenir et attendre***: **chaque processus possédant au moins une ressource est en conflit pour une autre ressource possédée par un autre processus.**

Caractérisation des interblocages

3) Pas de préemption: on ne peut pas forcer un processus à libérer une ressource qu'il possède déjà.

4) Cycle dans le graphe des attentes: il existe un ensemble de processus $\{P_0, P_1, \ldots, P_n\}$ tel que

P_0 est en conflit pour une ressource possédée par P_1,

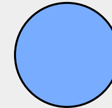
P_1 " " " P_2,

P_{n-1} " " " P_n,

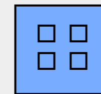
et P_n " " " P_0.

Graphe des attentes

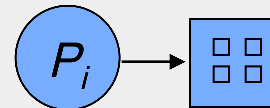
- **Ensemble des Processus**
  ***P* = {*P*₁, *P*₂, …, *P*ₙ},**



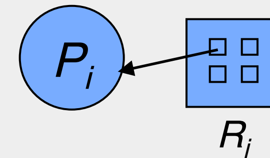
- **Ensemble des Ressources**
  $R = \{R_1, R_2, \ldots, R_m\}$*



- P_i **demande** R_j
  $P_i \rightarrow R_j$

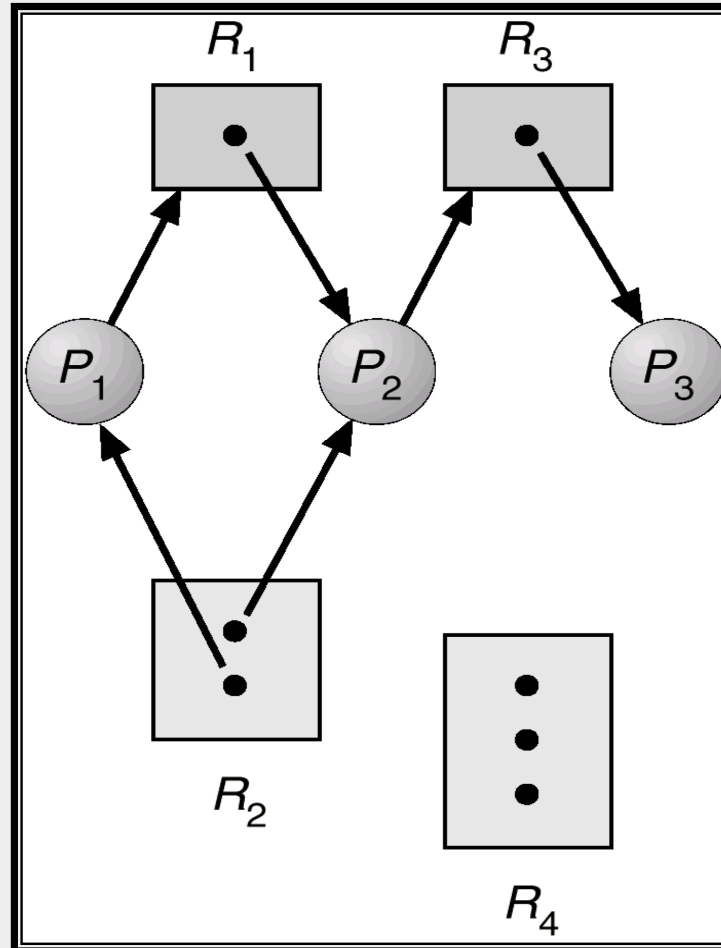


- P_i **détient** R_j
  $R_j \rightarrow P_i$

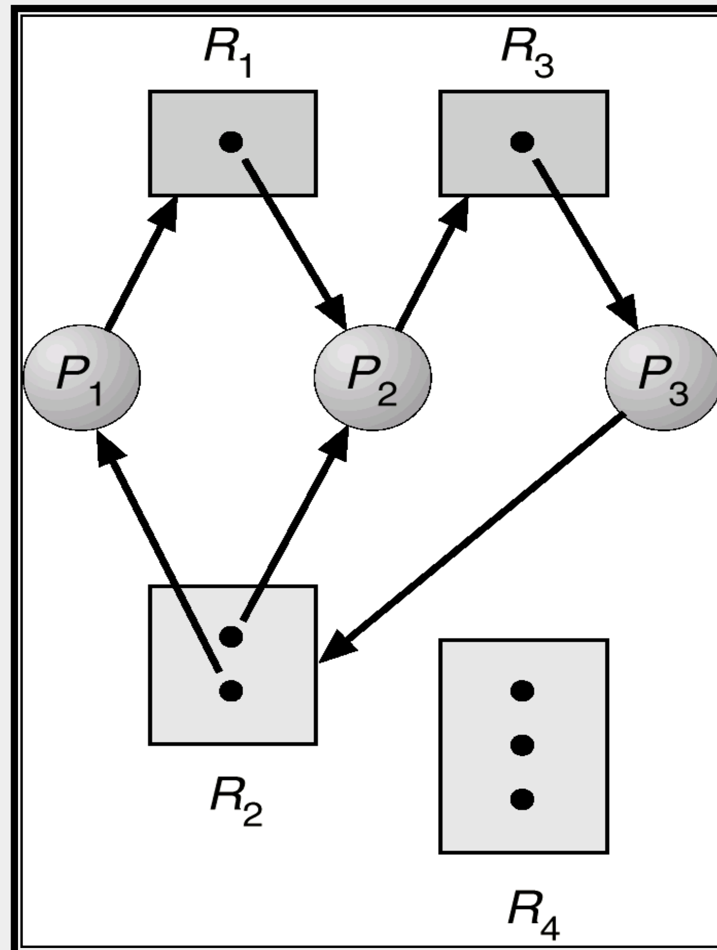


**** Ressources multiples autorisées***

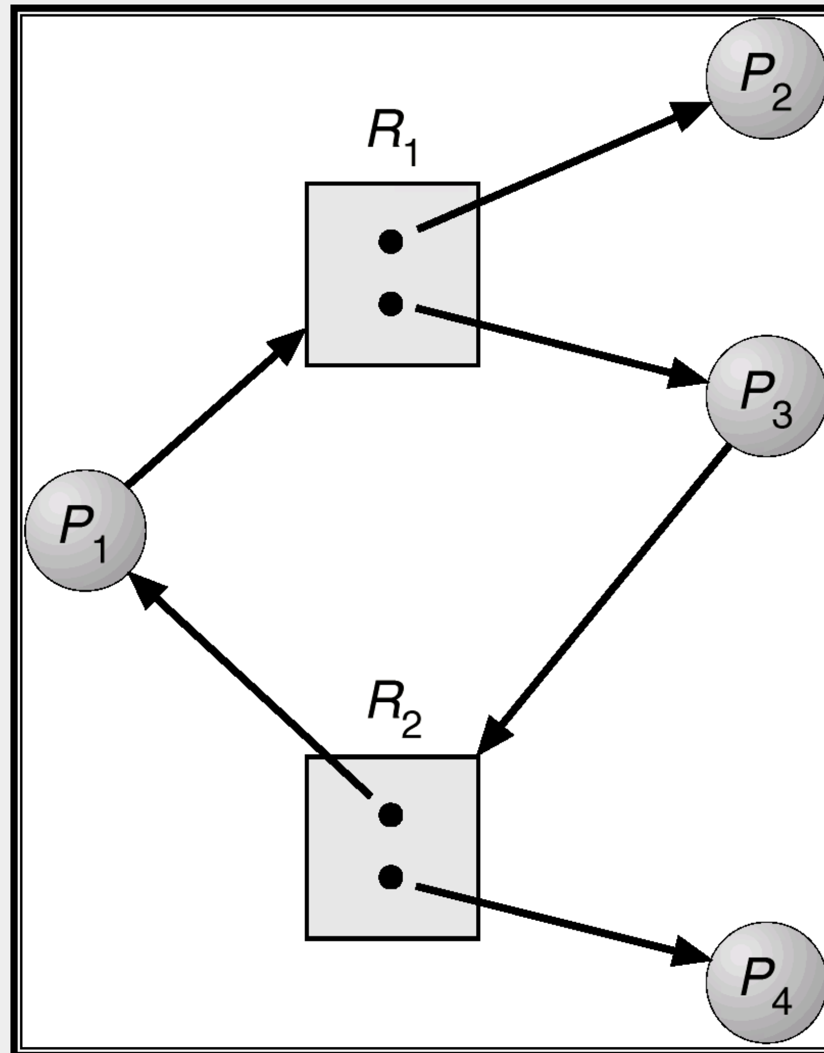
Exemple de graphe des attentes



Exemple de graphe des attentes avec interblocage



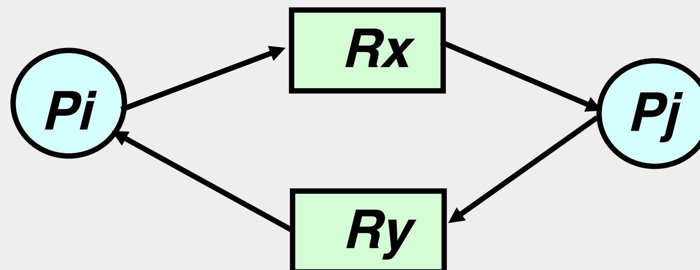
Exemple de graphe des attentes avec cycle et sans interblocage



Caractérisation des interblocages

- Pas de cycle dans le graphe des attentes ⇒ pas d'interblocage

- Cycle dans le graphe des attentes ⇒ interblocage possible



Traitement des interblocages

- **Prévention**
  - Contraintes sur le modèle de programmation des processus.
- **Evitement**
  - Analyse dynamique (à chaque demande de ressource) de l'état global d'allocation des ressources pour garantir une exécution sans interblocage.
- **Détection – guérison**
  - Analyse dynamique et périodique de l'état d'allocation des ressources. Si détection d'interblocage → guérison.

Prévention des interblocages

- Assurer qu'au moins l'une des quatres conditions de l'interblocage soit fausse.

Prévention des interblocages

- **Condition 1 : ressources non partageables ou pouvant générer un conflit**
  - Transformation de la ressource pour qu'elle tende à être partageable sans conflit
  - Modèle de programmation asynchrone
    - Exemple d'une imprimante
    - Spool (buffer de requêtes désynchronisées) géré par un processus démon
    - Remarque : le buffer reste une ressource pouvant générer un conflit.

Prévention des interblocages

- **Schématiquement:**
  - Resource X
  - Modèle synchrone
    - get(X) <operation O on X> free(X)
  - Modèle asynchrone
    - X.registerOperation(O, callback)
    - X.registerOperation(O, callback, data)

Prévention des interblocages

- **Condition 2 : tenir et attendre**
  - **Acquisition globale des ressources (Conservative Two Phase Locking)**
  - **Chaque processus demande simultanément toutes les ressources dont il a besoin (tout ou rien)**

    → Utilisation réduite des ressources

    → Fragmentation possible de l'exécution du processus pour garder moins lontemps les ressources (libération de toutes les ressources avant d'effectuer une nouvelle demande)

Exemple du compte bancaire / Tenir et attendre

```
Transfert(account from, account to, amount) {
  mutex.P();
  while (!(from.isFree()) || !(to.isFree())) {
    nbWaiting++; mutex.V(); waiting.P(); mutex.P();
  }
  from.setBusy(); to.setBusy();
  mutex.V();

  from.bal -= amount;
  to.bal += amount;

  mutex.P();
  from.setFree(); to.setFree();
  // wake up all the waiting processes
  for ( ;nbWaiting>0; nbWaiting--){
      waiting.V();
  }
  mutex.V();
}
```

```
interface account {
    public boolean isFree();
    public void setFree();
    public void setBusy();
    …
}
```

Allocation globale des ressources

- Technique très utilisée dans les bases de données
- Conservative 2-phase locking
  - All locks acquired simultaneously
- Basic 2-phase locking
  - Expanding phase (*locks may be acquired but none can be released)*
  - Shrinking phase (*locks can be released but no new lock can be acquired)*
- Conservative 2-PL moins performant mais garantit absence d'interblocages, basic 2PL garantit seulement la sérialisabilité de l'exécution

Allocation globale des ressources

- Conservative 2-PL moins performant mais garantit absence d'interblocages

- Basic 2PL garantit seulement la sérialisabilité de l'exécution

```
from.P();
from.bal -= amount;
to.P();
to.bal += amount;
from.V();
to.V();
               …
```

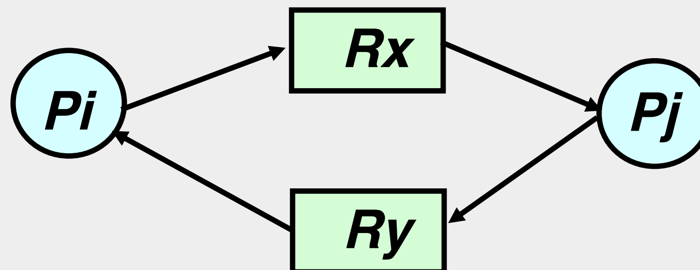
Prévention des interblocages

Condition 3 : préemption

- Si un processus P qui possède des ressources (Ra) demande d'autres ressources (Rb) qui ne peuvent pas lui être allouées, alors il libère toutes les ressources qu'il possède déjà
  - Les ressources préemptées sont ajoutées à la liste des ressources en attente pour le processus P.
  - Le processus P sera redémarré seulement lorsqu'il pourra obtenir toutes les ressources demandées (Ra et Rb).

Prévention des interblocages

- **Condition 4 : cycle dans le graphe des attentes**
  - **On impose un ordre total sur les ressources.**
  - **Chaque processus demande des ressources dans un ordre croissant.**



Cycle impossible si ordre d'allocation = Rx puis Ry (x < y)

© F. Boyer, UGA

Détection - guérison

- Autorise le système à entrer dans une situation d'interblocage

- Algorithme de détection périodique (thread *démon*)

```
while (true) {
        < détecter interblocage >
        < traiter interblocage >
        < attendre délai >
}
```

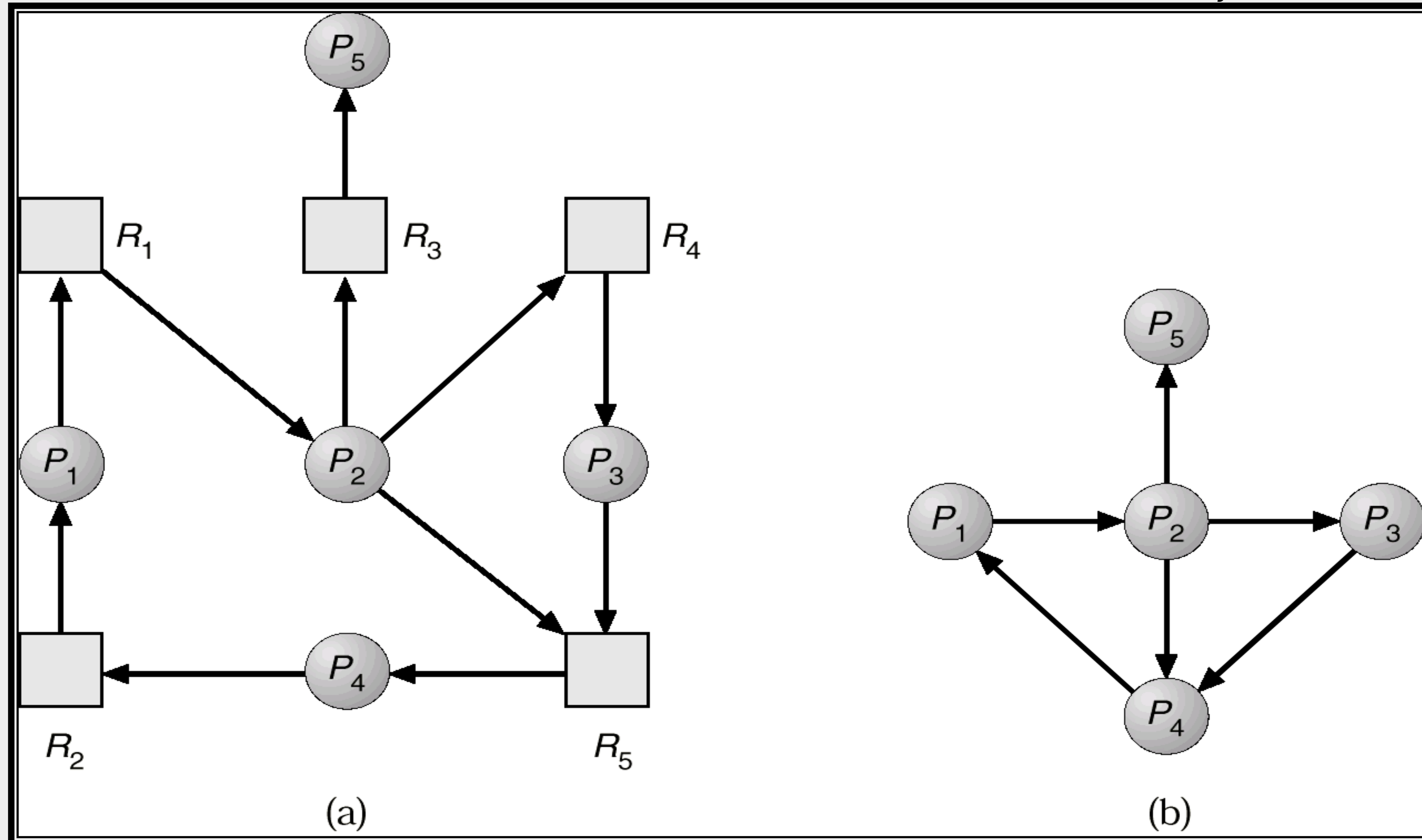
Détection dans le cas de ressources simples

- **Maintien d'un graphe *wait-for***
  - ***Noeuds = processus***
  - **$P_i \rightarrow P_j$ si P_i attend une ressource possédée par P_j.**

- **Exécution périodique d'un algorithme de recherche d'un cycle dans le graphe wait-for**

Graphe d' allocation de ressources / graphe wait-for

Sylberschatez



Graphe des Attentes **Graphe des Allocations**

Détection dans le cas de ressources multiples

- ***Available:*** **un vecteur de longueur *m* indique le nombre de ressources libres de chaque type.**

- ***Allocation:*** **une matrice *n* x *m* definit le nombre de ressources de chaque type allouées à chaque processus.**

- ***Request:*** **une matrice *n* x *m* indique les demandes en cours de chaque processus. Si *Request [i,j]= k*, alors le processus P_i demande *k* instances supplémentaires de la ressource de type R_j.**

m : nombre de types de ressources

n : nombre de processus de l'application

Algorithme de détection

1. Soient *Work* et *Finish, deux* vecteurs initialisés:

   Work = *Available*

   for i = 1,2, …, *n*,

   if $Allocation_i \neq 0$,

   then *Finish*[i] = *false*;

   else *Finish*[i] = *true;*

2. Trouver un processus *i* tq :

   Finish[*i*] == *false and* $Request_i \leq Work$

   If no such i exists, go to step 4.

Algorithme de détection (2)

3. $Work = Work + Allocation_i$
   $Finish[i] = true$
   go to step 2.

4. If $Finish[i]$ == false, for some i, $1 \leq i \leq n$, then the system is in deadlock state.

Exemple de détection

- 5 processus P_0 .. P_4
- 3 types de ressources
  A (7 instances), *B* (2 instances), et *C* (6 instances)
- T_0:

	Allocation	*Request*	*Work*
	A B C	*A B C*	*A B C*
P_0	0 1 0	0 0 0	0 0 0
P_1	2 0 0	2 0 2	
P_2	3 0 3	0 0 0	
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

- Séquence <P_0, P_2, P_3, P_1, P_4> résulte en
  Finish[*i*] = true pour tout *i*.

Exemple de détection (suite)

- P_2 demande une instance supplémentaire de type C

	Allocation	*Request*	*Work*
	A B C	*A B C*	*A B C*
P_0	0 1 0	0 0 0	0 0 0
P_1	2 0 0	2 0 1	
P_2	3 0 3	0 0 1	
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

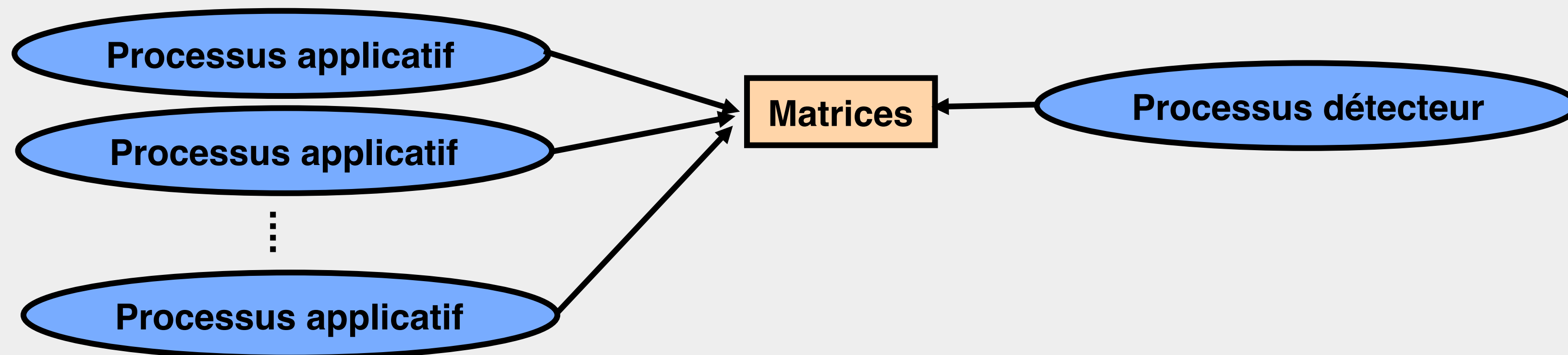
- Etat du système
  - Interblocage des processus P_1, P_2, P_3, and P_4.

Maintien des informations pour la détection

- Matrice mise à jour / demandes et libérations

- Processus dédié qui analyse les matrices régulièrement

  → Coût en CPU et en mémoire



Guérison

- **Aborter (stopper l'exécution)**
  - tous les processus impliqués dans l'interblocage
  - un processus à la fois, juqu'à ce que l'interblocage soit éliminé.
- **Rollback (revenir en arrière dans l'execution du programme en annulant les actions réalisées)**
  - Utiliser un journal des opérations (journal après / journal avant)

Choix du processus à aborter

Quel ordre pour aborter les processus ?

- **Critères :**
  - Le processus en cours d'exécution
  - Priorité du processus
  - Temps d'exécution effectué
  - Ressources utilisées par le processus
  - Ressources manquantes pour le processus
  - Processus interactif ou batch?

- **Famine possible**

Guérison

Aborter un processus :

- Libération des ressources
- Perte du contexte d'exécution
- Reprise de l'exécution à son point de départ ou à un point de reprise (rollback)

Problème des opérations « permanentes »

Enregistrements sur fichiers / BD

E/S, Impressions

…

→ Utilisation d'un journal

→ Certaines opérations ne peuvent pas être défaites

Evitement des interblocages

- **Chaque processus annonce le nombre maximum de ressources de chaque type qu'il peut utiliser**
- **A chaque demande de ressource, un *algorithme d'évitement* examine l'état d'allocation des ressources pour assurer des conditions d'exécution sans interblocage (état sûr)**
- **L'état d'allocation des ressources est défini par les ressources libres/allouées et par les demandes futures des processus**

Evitement des interblocages : notion d'état sûr

Etat sûr : il existe une séquence sûre d'exécution des processus

Séquence sûre : séquence d'exécution des processus qui leur permet à tous de terminer, même si chaque processus demande le maximum de son annonce

Si une application est dans un état sûr, elle a la garantie de pouvoir se terminer sans IB.

Si elle n'est pas dans un état sûr, un IB est possible.

Evitement des interblocages : notion d'état sûr (2)

Séquence <P_1, P_2, …, P_n> *sûre* : pour chaque P_i, les ressources que Pi peut encore demander peuvent être satisfaite par les ressources libres + les ressources utilisées par les processus P_j, ($j<i$)

- Si les ressources demandées par P_i ne sont pas libres, alors P_i attend que tous les processus P_j soient terminés.
- Quand P_j termine, P_i peut obtenir les ressources, s'exécuter, libérer les ressources et terminer.
- Quand P_i termine, P_{i+1} peut obtenir les ressources demandées, etc.

Exemple d’évitement : algorithme du banquier

- Ressources multiples
- Chaque processus doit annoncer le nombre maximum d’instances de chaque type de ressource qu’il pourra demander
- Quand un processus demande une ressource, il peut être mis en attente
- Quand un processus possède toutes les ressources dont il a besoin, il termine son exécution dans un temps fini

Structures de données

Available*:* vecteur de longueur *m*.

Si available [*j*] = *k*, il y a *k* instances de R_j disponibles à l'instant courant.

Max*:* matrice nxm.

Si *Max* [*i,j*] = *k*, alors P_i peut demander au plus k instances de R_j.

Allocation*:* matrice nxm.

Si Allocation[*i,j*] = *k* alors P_i possède *k* instances de R_j.

Need*:* matrice nxm.

Si *Need*[*i,j*] = *k*, alors P_i peut demander *k* instances supplémentaires de R_j.

Need [i,j] = Max[i,j] – Allocation [i,j]

Algorithme de sureté

1. Soient Work et Finish, deux vecteurs initialisés:

   Work = Available

   Finish [i] = faux (i=1..n)

2. Trouver un processus i tel que :

   Finish [i] = faux et $Need_i \leq Work$

   Si echec, aller en 4.

3. Work = Work + $Allocation_i$
   Finish[i] = vrai
   aller en 2.

4. Si Finish [i] == vrai pour tout i,

   alors le système est dans un état sûr.

Algorithme d'allocation pour un processus

P_i demande k instances de la ressource de type R_j :

1. Si $Request_i \leq Need_i$ aller en 2
   Sinon, erreur (Pi excède sa demande max)

2. Si $Request_i \leq Available$, aller en 3.
   Sinon P_i attend (ressources non disponibles)

3. Essaie d'allouer les ressources à Pi en modifiant l'état comme suit :

   $Available = Available - Request_i$;
   $Allocation_i = Allocation_i + Request_i$;
   $Need_i = Need_i - Request_i$;

   - Si état sûr $\Rightarrow$ ressources allouées à P_i.
   - Sinon $\Rightarrow$ P_i attend, et l'ancien état (Available, $Allocation_i$ et $Need_i$) est restauré

Exemple : P1 demande (1, 2 ,1)

- 5 processus $P_{0\ldots}P_4$;
- 3 ressources de type *A* (10 instances), *B* (5instances, and *C* (7 instances).
- T_0:

	Allocation	*Max*	*Available*	*Need*
	A B C	*A B C*	*A B C*	*A B C*
P_0	0 1 0	7 5 5	3 3 2	7 4 5
P_1	2 0 0	3 2 2		1 2 2
P_2	3 0 2	9 0 2		6 0 0
P_3	2 1 1	2 2 2		0 1 1
P_4	0 0 2	4 3 3		4 3 1

→ Le système est dans un état sûr car la séquence < P_1, P_3, P_4, P_0, P_2> satisfait le critère de sureté.

Exemple P_4 demande (3,3,0)

	Allocation	Need	Available
	A B C	A B C	A B C
P_0	0 1 0	7 4 5	2 3 0
P_1	3 0 2	0 2 0	
P_2	3 0 1	6 0 1	
P_3	2 1 1	0 1 0	
P_4	0 0 2	4 3 1	

→ Attente de P4

Exemple : P_0 demande (0,2,0)

	Allocation	Need	Available
	A B C	A B C	A B C
P_0	0 1 0	7 4 5	2 3 0
P_1	3 0 2	0 2 0	
P_2	3 0 1	6 0 1	
P_3	2 1 1	0 1 0	
P_4	0 0 2	4 3 1	

- Si on alloue 2 ressources B à P0 : état non sûr

Remarques sur l'algorithme du banquier

- Pessimiste : suppose que tous les processus demanderont la totalité de leur annonce

- Il peut donc conduire à retarder des allocations qui ne conduiront pas à des interblocages si certains processus ne demandent pas tout ce qu'ils ont annoncé

- Il peut conduire à un écroulement du nombre de processus en exécution si le nombre de ressources disponibles devient faible, si les annonces sont voisines du nombre total de ressources du système et si les requêtes des processus approchent de leur maximum

En cas de ressources en exemplaire unique

- Il existe une simplification de l'algorithme du banquier qui est fondée sur une recherche de circuit dans le graphe des allocations et demandes
- Une requête ne doit être satisfaite que si elle ne conduit pas à l'apparition d'un circuit
- L'inconvénient principal est que chaque processus doit annoncer à son début le maximum de ce qu'il pourra utiliser
- Difficile lorsque c'est la consultation des premières ressources qui indique les prochaines ressources à accéder, comme c'est le cas dans une base de données
- Déclarer que le processus va utiliser la totalité des ressources conduit à une disparition du parallélisme

Conclusions sur les IB

- Pas de solution idéale
- Méthode choisie dépend du contexte
  - Prévention : implique de connaître les demandes des processus + met en œuvre des algorithmes complexes
  - Détection-guérison : implique une perturbation raisonnable de l'application
- Attitude à adopter
  - Programmation « sûre »
  - Validation par tests / outils spécifiques