

# Systèmes et algorithmes répartis

## Problèmes génériques

Philippe Quéinnec, Gérard Padiou

ENSEEIHT  
Département Sciences du Numérique

30 septembre 2021



# Plan



## 1 Exclusion mutuelle

- Le problème
- Jeton circulant
- Algorithme de Ricart-Agrawala
- Algorithme à base d'arbitres

## 2 Détection de la terminaison

- Le problème
- Terminaison sur un anneau
- Algorithme des quatre compteurs
- Algorithme des crédits

## 3 Détection de l'interblocage

- Le problème
- Caractérisation de l'interblocage
- Algorithme de Chandy, Misra, Haas

## 4 La diffusion fiable



# Plan

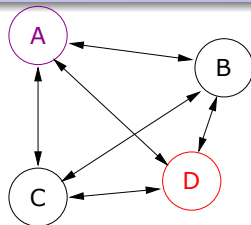
- 1 Exclusion mutuelle
  - Le problème
  - Jeton circulant
  - Algorithme de Ricart-Agrawala
  - Algorithme à base d'arbitres
- 2 Détection de la terminaison
  - Le problème
  - Terminaison sur un anneau
  - Algorithme des quatre compteurs
  - Algorithme des crédits
- 3 Détection de l'interblocage
  - Le problème
  - Caractérisation de l'interblocage
  - Algorithme de Chandy, Misra, Haas
- 4 La diffusion fiable



## Spécification du problème

```
process P(int i) :  
    ...; Entrer(i); <SC>; Sortir(i); ...
```

### Diagramme d'état



D est en exclusion  
A est candidat

- Sûreté : Un processus **au plus** en exclusion  
 $\forall i, j :: P_i.\text{exclusion} \wedge P_j.\text{exclusion} \Rightarrow i = j$
- Vivacité faible : pas d'interblocage (certains candidats finissent par entrer)
- Vivacité forte : Tout candidat finit par entrer
- Protocole : Tout processus en exclusion finit par sortir

[ Précis 4.1 pp.63–65 ]

# Élection vs exclusion mutuelle

## Problèmes similaires. . .

Isoler un processus parmi tous : introduire une **asymétrie**

## . . . mais bien différents

- Élection d'un **quelconque** des processus mais exclusion mutuelle parmi les **candidats**
- L'élection est définitive mais l'exclusion mutuelle se termine et se transmet  $\Rightarrow$  évolution **dynamique**



# Algorithme à base de jeton circulant

## Algorithme basé sur le contrôle d'un objet circulant

- Anneau logique (indépendant de la structure du réseau physique) : chaque site a un successeur
- Jeton circulant :
  - un site non demandeur transmet le jeton à son successeur
  - un site demandeur attend le jeton pour obtenir l'exclusion mutuelle
  - un site qui sort d'exclusion mutuelle transmet le jeton à son successeur

## Propriétés

- Sûreté : unicité du jeton
- Vivacité : intégrité de l'anneau (existence et circulation du jeton)

# Algorithme à base de jeton circulant

```
Process P(i : 0..N-1) {  
  type Etat = {hors,candidat,exclusion}  
  Etat EC ← hors;  
  bool jeton ← (i = 0); // jeton initialement sur site 0  
  on (EC = hors) : // hors → candidat  
    EC ← candidat;  
  on (EC = candidat ∧ jeton) : // candidat → exclusion  
    EC ← exclusion;  
  on (EC = exclusion) : // exclusion → hors  
    EC ← hors;  
    send MsgJeton to  $P_{i \oplus 1}$   
    jeton ← false;  
  on reception MsgJeton : // réception jeton  
    jeton ← true;  
  on jeton ∧ EC = hors : // transmission jeton  
    send MsgJeton to  $P_{i \oplus 1}$   
    jeton ← false;  
}
```

## Algorithmes à base de permission

Un processus candidat doit demander à d'autres processus la **permission** d'entrer en exclusion

- À tout  $P_i$  on associe un ensemble  $D_i$  contenant les processus à contacter
- Correction :  $\forall i \neq j : j \in D_i \vee i \in D_j$
- Deux types de permissions :
  - **individuelles** : un processus donne son autorisation selon son propre état  $\Rightarrow$  n'engage que lui
  - **d'arbitre** : les processus s'échangent des permissions préexistantes en nombre fixé  $\Rightarrow$  engage tous les processus qui dépendent de lui
- **Objectif** : Minimiser les ensembles  $D_i$

1. *An Optimal Algorithm for Mutual Exclusion in Computer Networks*, Glenn Ricart and Ashok K. Agrawala. Communications of the ACM, January 1981.



## Permissions individuelles : Ricart et Agrawala



### Hypothèses

- Chaque processus connaît l'identité des  $N$  processus
- Réseau de communication fiable et maillé (non FIFO)
- Pas de défaillance de processus

### Solution

- Utilisation de permissions individuelles avec :  
 $\forall i : D_i = \text{Tous} - \{i\}$
- Ordonnancement des requêtes par datation

### Messages

- Message de requête : le site  $i$  demande l'autorisation à  $j$
- Message d'autorisation : le site  $j$  donne son autorisation à  $i$
- Pas de message de refus : le refus est temporaire



# Algorithme de Ricart et Agrawala

## Principes

- Requêtes d'entrée totalement ordonnées :  
⇒ Utilisation d'horloges de Lamport
- Chaque processus  $P_i$  candidat ou en exclusion connaît la date de sa requête courante  $Date(R_i)$
- Un candidat entre en exclusion s'il a obtenu les permissions de tous les autres  
⇒ il possède alors la requête la plus ancienne
- Revient à vérifier que la requête  $R_i$  d'un processus  $P_i$  est la plus vieille requête des processus candidats ou en exclusion :  
$$\forall k : P_k.Etat \neq hors \Rightarrow Date(R_i) \leq Date(R_k)$$



# Algorithme de Ricart-Agrawala

```
Process P(i : 0..N-1) {  
  type Etat = {hors,candidat,exclusion}; Etat EC ← hors;  
  Date hloc ← new Date(0,i); // horloge locale  
  Date dr; // date de la requête de ce site  
  Set<int> Att ← ∅; // sites lui ayant demandé l'autorisation  
  Set<int> D; // sites dont i attend l'autorisation  
  on (EC = hors) : // hors → candidat  
    EC ← candidat;  
    D ← 0..N-1 \ {i};  
    dr ← hloc.Top();  
    for k ∈ D : send Request(i,dr) to Pk;  
  on reception Request(p, d) :  
    hloc.Recaler(d);  
    if (EC ≠ hors ∧ dr < d) then Att ← Att ∪ {p};  
    else send Perm(i) to Pp;  
  on (EC = candidat) ∧ reception Perm(p) : // candidat → excl  
    D ← D \ {p};  
    if (D = ∅) then EC ← exclusion;  
  on (EC = exclusion) : // exclusion → hors  
    for k ∈ Att : send Perm(i) to Pk;  
    Att ← ∅; EC ← hors;
```

# Permissions d'arbitres



## Principe

Obtenir la permission de tous les arbitres contactés.

Condition nécessaire :  $\exists$  arbitre commun :  $\forall i, j : D_i \cap D_j \neq \emptyset$

## Exemple

<i>i prend pour arbitre :</i>	1	2	3	4	5
1		•	•		
2			•	•	
3		•		•	
4		•		•	
5	•	•			

Note : 1 et 5 ne sont pas arbitres



# Quorums

## Définition

Un système de quorum est un ensemble d'ensembles, tel que tout couple d'ensembles a une intersection non vide :

$$\mathcal{Q} = \{Q_1, \dots, Q_n\}, \forall i, j : Q_i \cap Q_j \neq \emptyset$$

Un quorum  $Q_i$  est responsable pour l'ensemble total.

Idéalement :

- Effort identique : Tous les quorums ont la même taille :  
 $\forall i : |Q_i| = K$
- Responsabilité identique : Tous les sites appartiennent au même nombre de quorums :  $\forall i : |\{j : i \in Q_j\}| = D$
- Minimalité :  $K = D =$  le plus petit possible (théorie :  $\lceil \sqrt{n} \rceil$ )

## Permissions d'arbitres

Construction facile d'un système de quorum quasi optimal :

- Construire une matrice arbitraire, éventuellement en dupliquant certains sites, p.e. pour 14 sites :

1	2	3	4
5	6	7	8
9	10	11	12
13	14	1	2

- Tout processus utilise les arbitres de sa colonne et de sa ligne  
 $D_6 = \{2, 5, 7, 8, 10, 14\}$
- Tout processus utilise au plus  $2\lceil\sqrt{n}\rceil - 1$  arbitres
- Tout arbitre appartient à au plus  $2\lceil\sqrt{n}\rceil - 1$  ensembles

---

1. A  $\sqrt{n}$  Algorithm for Mutual Exclusion in Decentralized Systems, Mamoru Maekawa. ACM Transactions on Computer Systems, May 1985.



## Permissions d'arbitres

ATTENTION : pas de miracle...

### Modèle parallèle processus $\leftrightarrow$ ressources critiques

- Un processus doit collecter des jetons  $\equiv$  ressources critiques
- Problème : **risque d'interblocage**
  - Solution préventive : ordonner les jetons et les demander en respectant cet ordre
  - Solution dynamique :
    - ordonner les requêtes (approche transactionnelle) et appliquer l'algorithme « *wound-wait* » ou « *wait-die* »
    - nécessite un ordre total sur les requêtes :  
 $\Rightarrow$  usage d'horloges de Lamport
    - Risque de livelock



# Plan

- 1 Exclusion mutuelle
  - Le problème
  - Jeton circulant
  - Algorithme de Ricart-Agrawala
  - Algorithme à base d'arbitres
- 2 **Détection de la terminaison**
  - Le problème
  - Terminaison sur un anneau
  - Algorithme des quatre compteurs
  - Algorithme des crédits
- 3 Détection de l'interblocage
  - Le problème
  - Caractérisation de l'interblocage
  - Algorithme de Chandy, Misra, Haas
- 4 La diffusion fiable





# Terminaison : détecter une propriété stable



## Spécification

- Propriété **stable** à détecter :

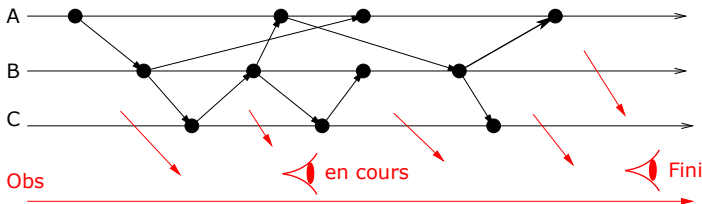
Tous les processus sont passifs **ET** pas de message en transit.

- Sûreté : Pas de **fausse détection** :

$$Term \Rightarrow (\forall i :: P_i.\text{passif} \wedge \text{EnTransit} = \emptyset)$$

- Vivacité : La terminaison **fini par** être détectée :

$$(\forall i :: P_i.\text{passif} \wedge \text{EnTransit} = \emptyset) \leadsto Term$$



[ Précis 4.2 pp.66–69 ]



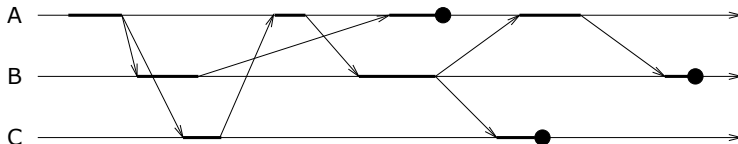
# Calcul diffusant



## Définition d'un calcul diffusant

- Un processus initial émet un ou plusieurs messages
- Puis, tous les processus adoptent le même comportement :

```
loop { /* un pas de calcul */  
    recevoir( $m$ ) ;  
    traiter  $m$  ;  
    envoyer 0 à  $N - 1$  messages ;  
}
```



⇒ Les phases actives peuvent être vues comme atomiques



## Terminaison sur un anneau (Misra, 1983)



### Principe

Les sites sont organisés en anneau (communication FIFO depuis le précédent / vers le suivant mais à destinataire arbitraire).  
Parcourir l'anneau et vérifier que tous les sites sont passifs.

### Difficulté

Un message émis avant le passage du visiteur sur le site émetteur peut être reçu après le passage du visiteur sur le site récepteur et réactiver un site trouvé passif

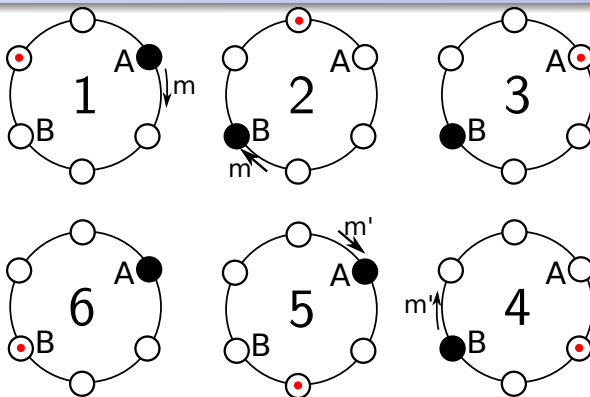
⇒ faire **deux** tours en vérifiant qu'aucun site n'a changé d'état entre temps

---

1. *Detecting Termination of Distributed Computation Using Markers*, Jayadev Misra. 2nd ACM Symposium on Principles of Distributed Computing, 1983.



## Terminaison – Misra



- tous trouvés passifs, mais calcul pas terminé !
  - sites A et B trouvés passifs, mais ils ont été actifs entre temps
- ⇒ faire **deux** tours en vérifiant qu'aucun site n'a changé d'état entre temps



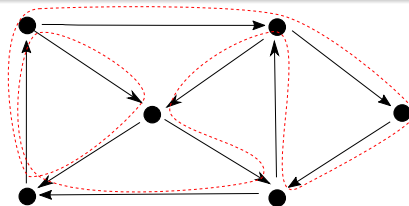
[illegible]

## Terminaison avec un anneau logique



### Graphe de communication arbitraire

- Circuit logique contenant **tous les arcs** (éventuellement plusieurs fois)
- Communication **FIFO** : le jeton ne peut pas dépasser un message antérieur sur le même arc
- Terminaison comme précédemment, avec  $N = \text{longueur du circuit}$

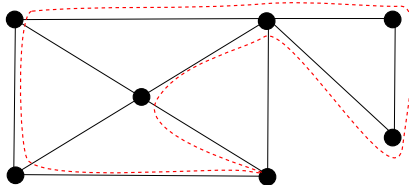


## Terminaison avec un anneau logique



### Graphe de communication arbitraire – avec la causalité

- Circuit logique contenant tous les sites (mais pas nécessairement tous les arcs)
- Communication **causale** : le jeton en transit depuis un site  $s$  ne peut pas arriver sur  $s'$  **avant** les messages émis avant sa visite sur  $s$  et envoyés directement de  $s$  vers  $s'$
- Terminaison comme précédemment, avec  $N =$  longueur du circuit

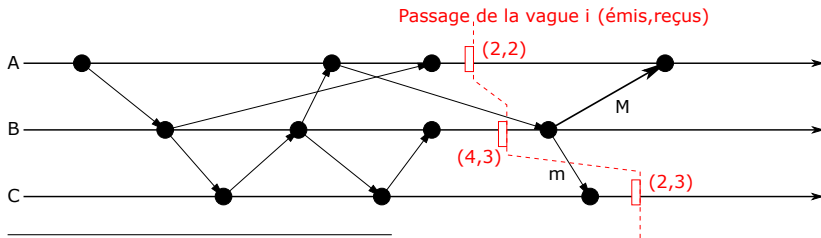


# Algorithme des quatre compteurs



## Principe

- Terminaison  $\triangleq Emis(t) = Reçus(t)$  **mais** impossible à évaluer
- Approche : Compteurs **locaux** des messages émis et reçus
- Mécanisme de **vague** pour collecter les valeurs des compteurs
- La vague  $i$  collecte  $R_i \triangleq \sum r_i$  et  $E_i \triangleq \sum e_i$

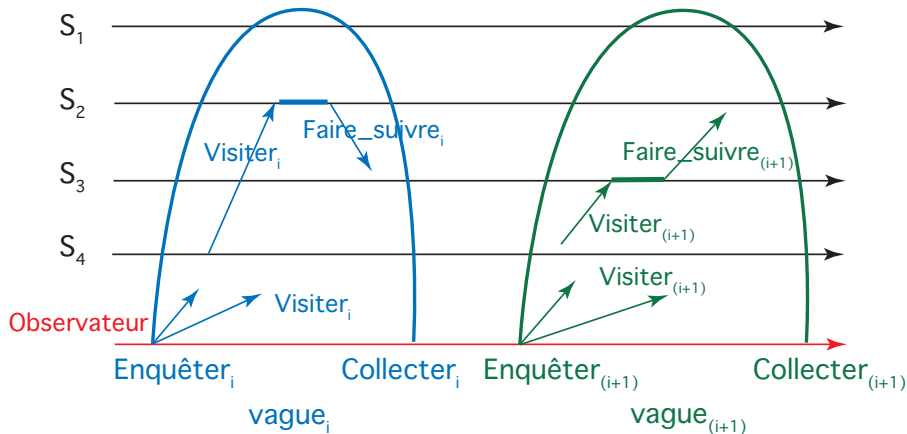


1. *Algorithms for Distributed Termination Detection*, Friedemann Mattern. Distributed Computing, 1987.





## Vague : itération répartie



*Handwritten signature*

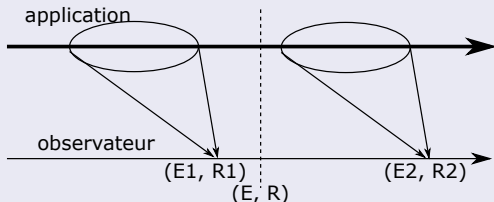


# Algorithme des quatre compteurs – preuve

**Vivacité** : si le calcul est terminé, alors la terminaison est détectée

Calcul terminé  $\Rightarrow$  il existe une date à partir de laquelle les compteurs de messages émis/reçus par site ne changent plus  $\Rightarrow$  deux vagues successives trouveront  $E_1 = R_1 = E_2 = R_2$ .

**Sûreté** : si la terminaison est annoncée, alors le calcul est terminé



- |     |                       |                             |
|-----|-----------------------|-----------------------------|
| (1) | $E \geq R$            | <i>calcul réparti</i>       |
| (2) | $E_1 \leq E \leq E_2$ | <i>compteurs croissants</i> |
| (3) | $R_1 \leq R \leq R_2$ | <i>idem</i>                 |
| (4) | $E_2 = R_1$           | <i>détection annoncée</i>   |
| (5) | $E \leq R$            | <i>d'après 2,3,4</i>        |
|     | $E = R$               | <i>d'après 1,5</i>          |

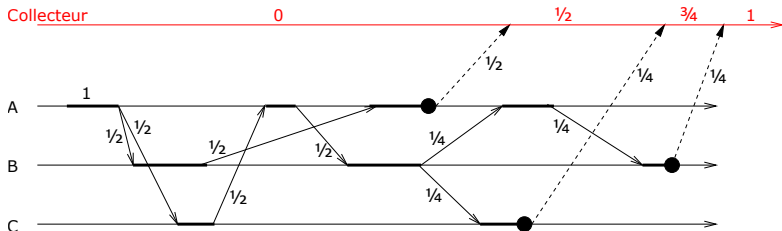
$(E, R)$  valeurs réelles  
(et inconnues) des  
compteurs entre deux  
vagues successives.

# Algorithme des crédits (Mattern)

Un peu oublié mais pourtant simple et performant...

## Principe

- Le processus initial possède un crédit de 1
- Le crédit courant est **partagé** entre les messages émis
- Un processus **rend** son crédit s'il n'envoie pas de message
- Terminaison lorsque la **somme** collectée égale 1



*Handwritten signature*

# Plan

- 1 Exclusion mutuelle
  - Le problème
  - Jeton circulant
  - Algorithme de Ricart-Agrawala
  - Algorithme à base d'arbitres
- 2 Détection de la terminaison
  - Le problème
  - Terminaison sur un anneau
  - Algorithme des quatre compteurs
  - Algorithme des crédits
- 3 **Détection de l'interblocage**
  - Le problème
  - Caractérisation de l'interblocage
  - Algorithme de Chandy, Misra, Haas
- 4 La diffusion fiable



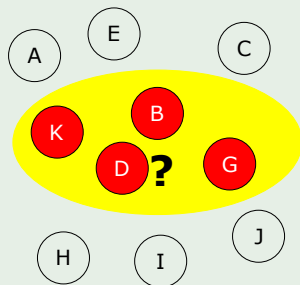
## Spécification du problème

### Détection d'une propriété stable

- Interblocage dû aux communications : attente de la réception d'un message
- Un processus est-il définitivement bloqué ?
- Sûreté : pas de fausse détection
- Vivacité : un processus bloqué finit par le savoir

Note : définition identique en centralisé, résolutions différentes (absence d'état global)

### Exemple



D s'interroge ?  
K, B, D, G en interblocage

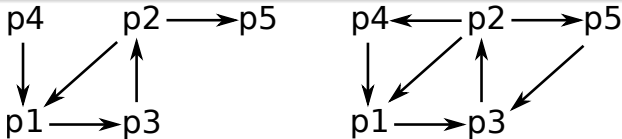
[ Précis 4.3 pp.72–75 ]

## Graphe d'attente



### Graphe d'attente

Graphe dont les nœuds sont les processus, et un arc  $p_i \rightarrow p_j$  si  $p_i$  est bloqué / en attente de  $p_j$



### Modèles de communication

- Modèle ET : un processus est bloqué tant qu'il n'a pas reçu un message depuis **tous** ceux qu'il attend.
- **Modèle OU** : un processus est bloqué tant qu'il n'a pas reçu un message depuis **l'un** de ceux qu'il attend.



## Caractérisation de l'état d'interblocage

### Modèle ET

Existence d'un cycle dans le graphe d'attente

### Modèle OU

Existence d'une composante fortement connexe terminale dans le graphe d'attente

### Composante fortement connexe terminale (CFCT)

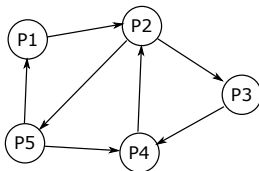
Un sous-graphe  $G'$  d'un graphe  $G = \{S, A\}$  est une CFCT (*knot*) ssi il existe un chemin entre tout couple de sommets de  $G'$  **et** si tout sommet de  $G'$  a ses successeurs dans  $G'$  :

$$\forall s, s' \in G' : \exists s \xrightarrow{*} s' \wedge \forall s \in G' : \text{succ}(s) \neq \emptyset \wedge \text{succ}(s) \subset G'$$

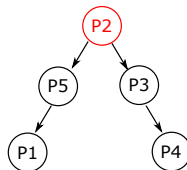


## Algorithme : calcul diffusant et arbre de contrôle

- Phase 1 : construction d'un arbre de recouvrement des sites bloqués (message d'enquête)
- Phase 2 : un site répond lorsqu'il est bloqué et que tous ses successeurs ont répondu, ou qu'il a déjà été visité (dans ce cas, cycle ou jonction avec une enquête en cours)
- Si le site initiateur obtient une réponse de tous ses successeurs, il y a interblocage

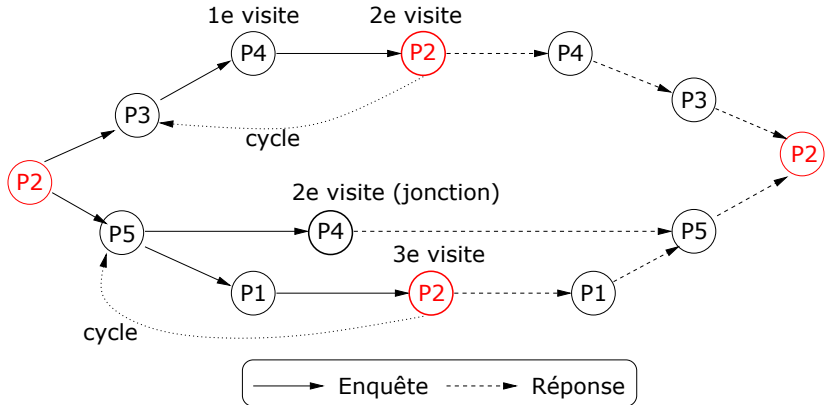


Arbre construit



1. *Distributed Deadlock Detection*, K. Mani Chandy, Jayadev Misra and Laura Haas. ACM Transactions on Computer Systems, May 1983.





## Propriétés

- Terminaison de la construction de l'arbre  $\Rightarrow P_2$  est interbloqué
- Pas de terminaison : Il faudra recommencer...

nt

# Plan

- 1 Exclusion mutuelle
  - Le problème
  - Jeton circulant
  - Algorithme de Ricart-Agrawala
  - Algorithme à base d'arbitres
- 2 Détection de la terminaison
  - Le problème
  - Terminaison sur un anneau
  - Algorithme des quatre compteurs
  - Algorithme des crédits
- 3 Détection de l'interblocage
  - Le problème
  - Caractérisation de l'interblocage
  - Algorithme de Chandy, Misra, Haas
- 4 La diffusion fiable

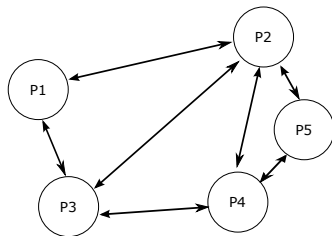


## La diffusion fiable

Envoyer un message à un ensemble de destinataires, tels que tous les processus corrects le délivrent, ou aucun.

### Hypothèses

- Réseau point-à-point fiable (tout message finit par arriver, intact, s'il existe un lien entre l'émetteur et le destinataire)
- Réseau connexe, pas nécessairement complet
- **Défaillance d'arrêt** : un processus peut s'arrêter définitivement, à tout moment



## Réalisation par inondation

### Diffuser(m), sur p

```
-- p = émetteur, m = message  
∀ s ∈ voisins(p) ∪ {p} faire  
    envoyer(⟨p,m⟩) à s  
fin pour
```

### Réception(⟨p,m⟩), sur q

```
si q n'a pas déjà délivré m alors  
    si p ≠ q alors -- propagation  
        ∀ s ∈ voisins(q) faire  
            envoyer(⟨p,m⟩) à s  
        fin pour  
    fin si  
    délivrer(m)  
fin si
```

## Propriétés

- Diffusion fiable *uniforme* : tous les processus (corrects ou ultérieurement défaillants) délivrent le message, ou aucun.
- Tout processus qui délivre un message l'a au préalable envoyé à ses voisins. Pour qu'un processus ne reçoive pas un message, il faudrait donc qu'aucun processus ne le lui ait envoyé, et donc aucun n'a pu le délivrer.
- Nombre de messages nécessaires = nombre de liens de communication ( $\times 2$ )
- Le protocole tolère des arrêts de processus, tant que le graphe reste connexe.
- Si le graphe cesse d'être connexe  $\Rightarrow$  partitions. La propriété de fiabilité devient « tous les destinataires d'une même partition le délivrent, ou aucun ».



# Conclusion

Quelques problèmes standards :

- Prise de cliché (chapitre II)
- Élection (chapitre II)
- Exclusion mutuelle
- Interblocage
- Terminaison d'un calcul réparti
- Diffusion fiable

