

---

# ARA

## Algorithmique Répartie avancée

### Master 2 - SAR

Luciana Arantes

---

## Planning

### ■ 7 cours de 3 heures

- 25/09 : Protocole de Diffusion : L. Arantes
- 02/10 : Détecteur de Défaillance : P. Sens
- 09/10 : Checkpointing : O. Marin
- 16/10 : Algorithmes Auto-stabilisants : F. Petit
- 23/10 : suite : F. Petit
- 30/10: Consensus Paxos : P. Sens
- 8/11 : Objets Partagés : L. Arantes

### ■ Evaluation

- Devoir (30%) + Examen (70%)

---

# Rappels

## Modèles de fautes et modèles temporels

24/09/2012

ARA: Introduction et Protocole de Diffusion

3

---

## Modèles de fautes

### ■ Origines des fautes

- fautes logicielles (de conception ou de programmation)
  - quasi-déterministes, même si parfois conditions déclenchantes rares
  - très difficiles à traiter à l'exécution : augmenter la couverture des tests
- fautes matérielles (ou plus généralement système)
  - non déterministes, transitoires
  - *corrigées* par point de reprise ou *masquées* par réplication
- piratage
  - affecte durablement un sous-ensemble de machines
  - masqué par réplication

### ■ Composants impactés

- Processus, processeurs, canaux de communication

24/09/2012

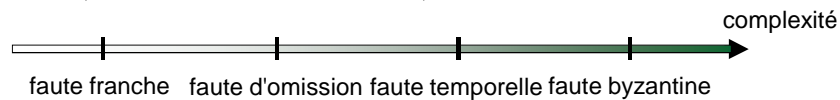
ARA: Introduction et Protocole de Diffusion

4

# Modèles de fautes

## ■ Classification des fautes

- *faute franche* : arrêt définitif du composant, qui ne répond ou ne transmet plus
- *faute d'omission* : un résultat ou un message n'est transitoirement pas délivré
- *faute temporelle* : un résultat ou un message est délivré trop tard ou trop tôt
- *faute byzantine* : inclut tous les types de fautes, y compris le fait de délivrer un résultat ou un message erroné (intentionnellement ou non)



24/09/2012

ARA: Introduction et Protocole de Diffusion

5

# Modèles temporels

## ■ Constat

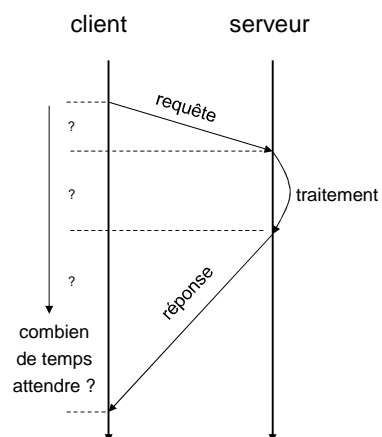
- vitesses processus différentes
- délais de transmission variables

## ■ Problème

- ne pas attendre un résultat qui ne viendra pas (suite à une faute)
- combien de temps attendre avant de reprendre ou déclarer l'échec ?

## ■ Démarche

- élaborer des modèles temporels dont on puisse tirer des propriétés



24/09/2012

ARA: Introduction et Protocole de Diffusion

6

## Modèles temporels (2)

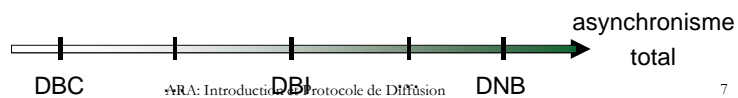
Modèle temporel = hypothèses sur :

- délais de transmission des messages
- écart entre les vitesses des processus

système synchrone : Modèle Délais/écarts Bornés Connus (DBC)  
- permet la détection parfaite de faute

système partiellement synchrone : Modèle Délais/écarts Bornés Inconnus (DBI)

système asynchrone : Modèle Délais/écarts Non Bornés (DNB)



24/09/2012

ARA: Introduction et Protocole de Diffusion

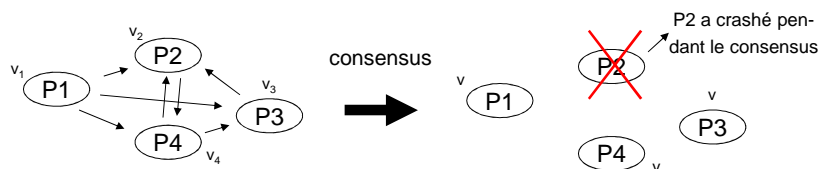
7

## Modèles temporels (3)

Résultat fondamental :

Fischer, Lynch et Paterson 85 : le problème du consensus ne peut être résolu de façon déterministe dans un système asynchrone en présence de ne serait-ce qu'une faute franche.

Problème du consensus :  $N$  processus se concertent pour décider d'une valeur commune, chaque processus proposant sa valeur initiale  $v_i$ .



Spécification formelle du consensus :

- terminaison : tout processus correct finit par décider
- accord : deux processus ne peuvent décider différemment
- intégrité : un processus décide au plus une fois
- validité : si  $v$  est la valeur décidée, alors  $v$  est une des  $v_i$

24/09/2012

ARA: Introduction et Protocole de Diffusion

8

# Notre modèle

---

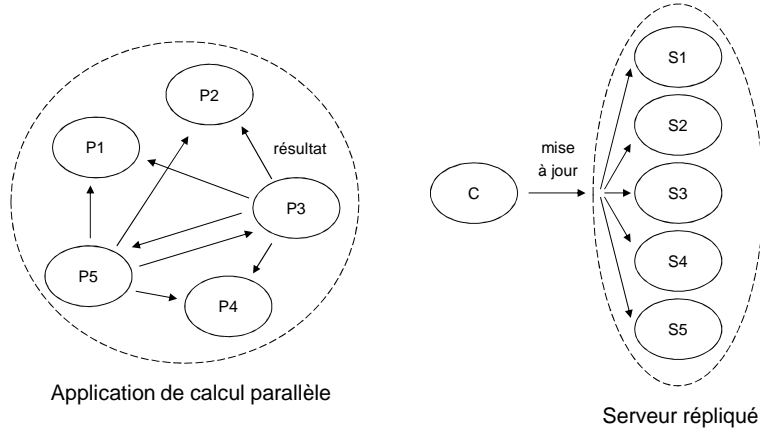
- **Ensemble de processus séquentiels indépendants**
  - > Chaque processus n'exécute qu'une seule action à la fois
- **Communication par échange de messages**
  - > Aucune mémoire partagée
  - > Les entrées des processus sont les messages reçus, les sorties sont les messages émis
- **Système asynchrone (souvent considéré):**
  - > Asynchronisme des communications
    - Aucune hypothèse sur les temps d'acheminement des messages (Pas de borne supérieur)
  - > Asynchronisme des traitements
    - Aucune hypothèse temporelle sur l'évolution des processus
- **Pas d'horloge commune**

---

# Protocoles de Diffusion

# Motivation (1)

Dans certaines situations, les processus d'un système réparti (ou un sous-ensemble de ces processus) doivent être **adressés** comme **un tout**.

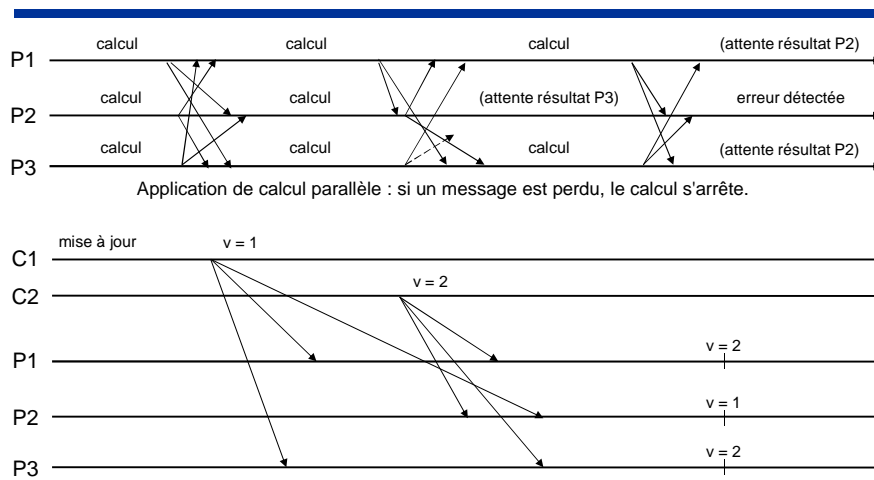


24/09/2012

ARA: Introduction et Protocole de Diffusion

11

# Motivation (2)



Application de calcul parallèle : si un message est perdu, le calcul s'arrête.  
 Serveur répliqué : si les messages ne sont pas reçus dans le même ordre sur tous les processus, ils sont incohérents.

24/09/2012

ARA: Introduction et Protocole de Diffusion

12

## Diffusion : Définition

---

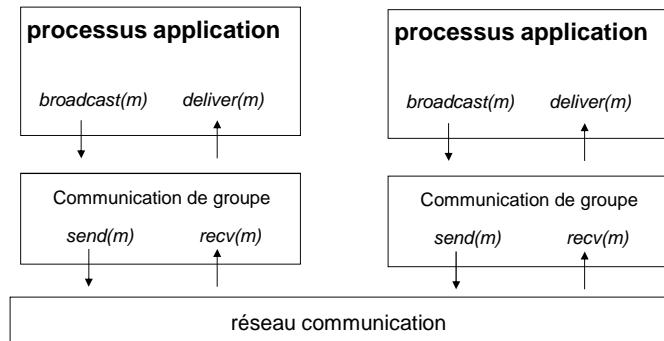
- **Un processus émetteur envoie un message à un groupe de processus.**
  - **Groupe** : ensemble de processus (les membres du groupe) auxquels on s'adresse par des diffusions, et non par des envois point à point.

## Diffusion : primitives

---

- **Primitives de diffusion utilisées par le processus  $p$  :**
  - ***broadcast* ( $m$ )** : le processus  $p$  diffuse le message  $m$  au groupe.
  - ***deliver* ( $m$ )** : le message  $m$  est délivré au processus  $p$ .
- **La diffusion est réalisée au dessus d'un système de communication existant.**

# Architecture



24/09/2012

ARA: Introduction et Protocole de Diffusion

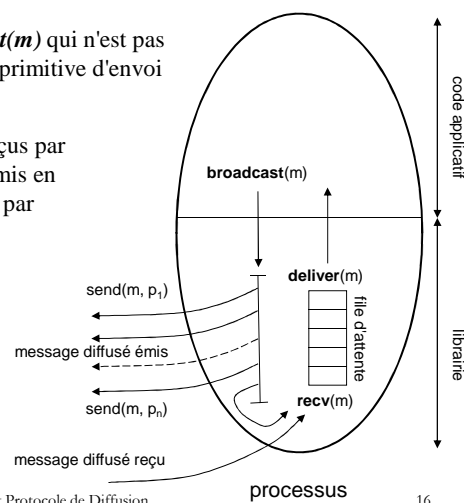
15

# Modèle d'implémentation

- Les messages sont diffusés par  $broadcast(m)$  qui n'est pas atomique en pratique : elle s'appuie sur la primitive d'envoi point à point  $send(m, p)$ .

- Les messages ne sont pas directement reçus par l'application : ils sont reçus par  $recv(m)$ , mis en attente, traités puis délivrés à l'application par  $deliver(m)$ .

- **Objectif** : implémenter des primitives  $broadcastX()$  et  $deliverX()$  qui garantissent la propriété X.



24/09/2012

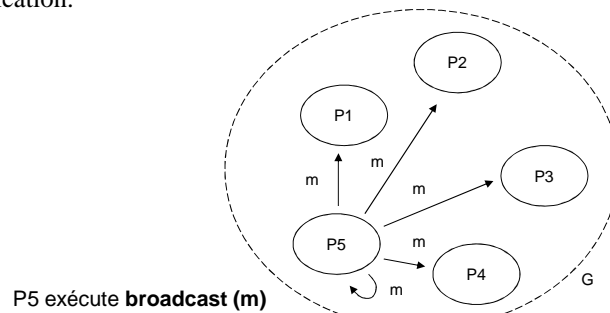
ARA: Introduction et Protocole de Diffusion

16



## Diffusion: primitives (2)

- Le message envoyé à chaque processus est le même, mais le nombre et l'identité des destinataires est masqué à l'émetteur, qui les désigne par leur groupe d'appartenance. On assure ainsi la transparence de réplication.



24/09/2012

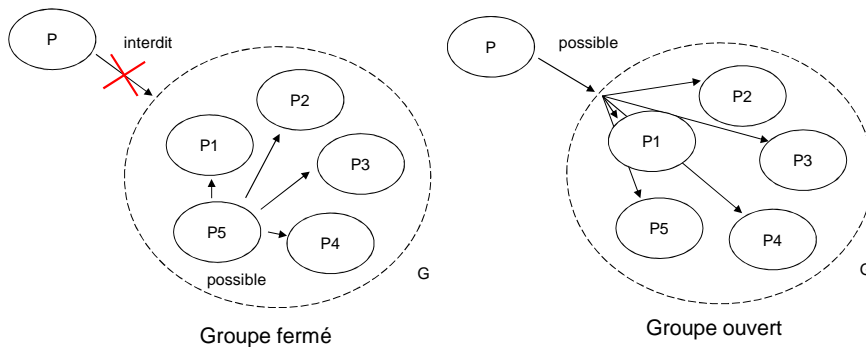
ARA: Introduction et Protocole de Diffusion

17

## Groupe (1)

Un *groupe* peut être :

- fermé** : *broadcast(m)* ne peut être appelé que par un membre du groupe
- ouvert** : *broadcast(m)* peut être appelé par un processus extérieur au groupe



24/09/2012

ARA: Introduction et Protocole de Diffusion

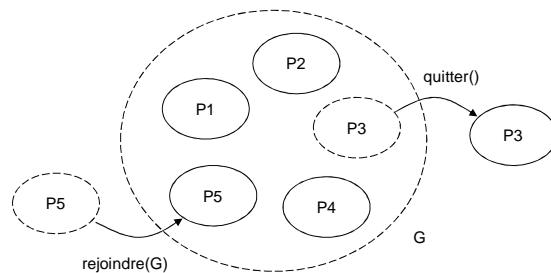
18

## Groupe (2)

---

Un groupe peut être :

- **statique** : la liste des membres du groupe est fixe et déterminée au départ
- **dynamique** : les processus peuvent rejoindre ou quitter le groupe volontairement par l'intermédiaire d'un service de gestion de groupe



Groupe dynamique

24/09/2012

ARA: Introduction et Protocole de Diffusion

19

## Problèmes

---

- Les processus peuvent tomber en panne, notamment au milieu d'un envoi multiple de message
- L'ordre de réception des messages sur les différents processus destinataires n'est pas garanti (entrelacement dû aux latences réseau variables)

### ■ Problèmes à résoudre

- assurer des propriétés de diffusion :
  - garantie de **remise** des messages
  - garantie d'**ordonnement** des messages reçus

24/09/2012

ARA: Introduction et Protocole de Diffusion

20

# Communications et Processus

---

## ■ Communications

- Point à point
- Tout processus peut communiquer avec tout les autres
- Canaux fiables : si un processus  $p$  correct envoie un message  $m$  à processus correct  $q$ , alors  $q$  finit par le recevoir ("eventually receives").

## ■ Processus

- Susceptibles de subir de pannes franches. Suite à une panne franche, un processus s'arrête définitivement : on ne considère pas qu'il puisse éventuellement redémarrer.

➔ Un processus qui ne tombe pas en panne sur toute une exécution donnée est dit **correct**, sinon il est dit **fautif**.

# Propriétés des diffusions (1)

---

## ■ Garantie de remise

- Diffusion Best-effort (Best-effort Broadcast)
- Diffusion Fiable (Reliable Broadcast)
- Diffusion Fiable Uniforme (Uniform Reliable Broadcast).

## ■ Garantie d'ordonnement

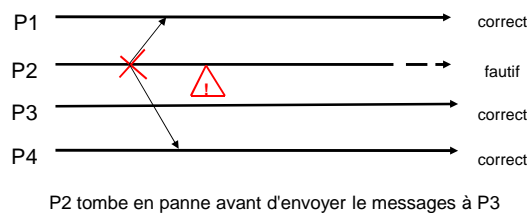
- les messages sont délivrés dans un ordre :
  - FIFO
  - Causal
  - Total

■ Les garanties de remise et d'ordre sont orthogonales

# 1. Garantie de Remise: Best Effort

## ■ Diffusion Best-effort

- Garantie la délivrance d'un message à tous les processus corrects si l'émetteur est correct.
- **Problème** : pas de garantie de remise si l'émetteur tombe en panne



24/09/2012

ARA: Introduction et Protocole de Diffusion

23

# Diffusion Best-Effort

## ■ Spécification

- **Validité** : si  $p_1$  et  $p_2$  sont corrects alors un message  $m$  diffuser par  $p_1$  finit par être délivré par  $p_2$ .
- **Intégrité**: un message  $m$  est délivré au plus une fois et seulement s'il a été diffusé par un processus.

## ■ Algorithme

Processus  $P$  :

**BestEffort\_broadcast(m)**

. envoyer  $m$  à tous les processus  $y$  compris  $p$  /\* groupe fermé \*/

upon **rcv(m)** :

**BestEffort\_deliver(m)** /\* délivrer le message \*/

24/09/2012

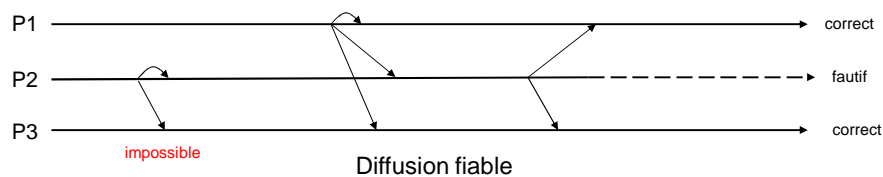
ARA: Introduction et Protocole de Diffusion

24

## 2. Garantie de Remise : Fiable

### ■ Diffusion Fiable (Reliable Broadcast)

- si l'émetteur du message  $m$  est **correct**, alors **tous** les destinataires **corrects** délivrent le message  $m$ .
- si l'émetteur du message  $m$  est *fautif*, tous ou aucun processus corrects délivrent le message  $m$ .



24/09/2012

ARA: Introduction et Protocole de Diffusion

25

## Diffusion Fiable – spécification (1)

### ■ Spécification

- **Validité** : si un processus **correct** diffuse le message  $m$ , alors tous les processus **corrects** délivrent  $m$
- **Accord** : si un processus **correct** délivre le message  $m$ , alors tous les membres **corrects** délivrent  $m$
- **Intégrité**: Un message  $m$  est délivré au plus une fois à tout processus **correct**, et seulement s'il a été diffusé par un processus.

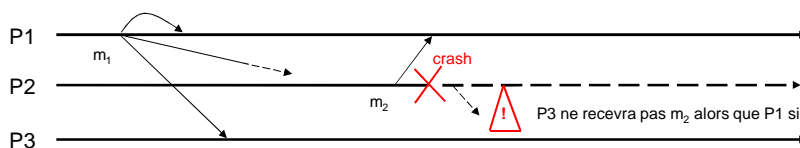
24/09/2012

ARA: Introduction et Protocole de Diffusion

26

## Diffusion Fiable : principe

Si un processus correct délivre le message diffusé  $m$ , alors tout processus correct délivre aussi  $m$



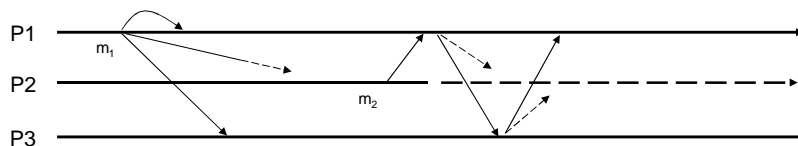
24/09/2012

ARA: Introduction et Protocole de Diffusion

27

## Diffusion Fiable : principe

Implémentation *possible* : sur réception d'un message diffusé par un autre processus, chaque processus rediffuse ce message avant de le délivrer.



24/09/2012

ARA: Introduction et Protocole de Diffusion

28

## Diffusion Fiable : algorithme

---

Chaque message  $m$  est estampillé de façon unique avec :

- $sender(m)$  : l'identité de l'émetteur
- $seq\#(m)$  : numéro de séquence

Processus  $P$  :

**Variable locale :**

$rec = \emptyset$ ;

**Real\_broadcast ( $m$ )**

estampiller  $m$  avec  $sender(m)$  et  $seq\#(m)$ ;

envoyer  $m$  à tous les processus  $y$  compris  $p$

**upon recv( $m$ ) do**

if  $m \notin rec$  then

$rec \cup = \{ m \}$

  if  $sender(m) \neq p$  then

    envoyer  $m$  à tous les processus sauf  $p$

**Real\_deliver( $m$ )** /\* délivrer le message \*/

24/09/2012

ARA: Introduction et Protocole de Diffusion

29

## Diffusion Fiable : discussion

---

### ■ Avantages :

- la fiabilité ne repose pas sur la détection de la panne de l'émetteur
- l'algorithme est donc valable dans tout modèle temporel

### ■ Inconvénients :

- l'algorithme est très inefficace : il génère  $n(n-1)$  envois par diffusion
- ce qui le rend inutilisable en pratique

### ■ Remarques :

- l'algorithme ne garantit aucun ordre de remise

24/09/2012

ARA: Introduction et Protocole de Diffusion

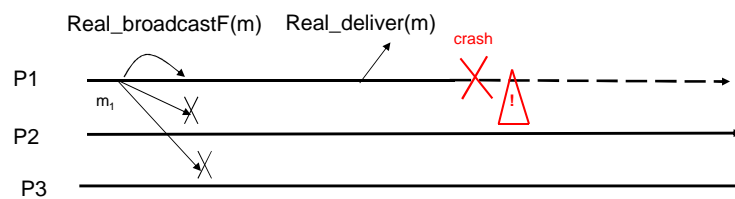
30

## Diffusion Fiable

### ■ Problème :

- aucune garantie de délivrance est offerte pour les processus fautifs

### ■ Exemple :



- $P_1$  délivre  $m$  et après il crash ;  $P_2$  et  $P_3$  ne reçoivent pas  $m$
- $P_1$  avant sa défaillance peut exécuter des actions irréversibles comme conséquence de la délivrance de  $m$

24/09/2012

ARA: Introduction et Protocole de Diffusion

31

## 3: Garantie de Remise – fiable uniforme

### ■ Diffusion Fiable Uniforme (Uniform Reliable Broadcast)

- Si un message  $m$  est délivré par un processus (**fautif** ou **correct**), alors tout processus **correct** finit aussi par délivrer  $m$ .

24/09/2012

ARA: Introduction et Protocole de Diffusion

32



## Diffusion Fiable Uniforme

---

- **Propriété d'uniformité**

- Une propriété (accord, intégrité) est dite **uniforme** si elle s'applique à tous les processus : **corrects** et **fautifs**.

- **Diffusion Fiable Uniforme**

- **Validité** : si un processus correct diffuse le message  $m$ , alors tous les processus corrects délivrent  $m$
- **Accord uniforme** : si un processus (**correct** ou **fautif**) délivre le message  $m$ , alors tous les membres corrects délivrent  $m$ .
- **Intégrité uniforme**: Un message  $m$  est délivré au plus une fois à tout processus (**correct** ou **fautif**), et seulement s'il a été diffusé par un processus.

24/09/2012

ARA: Introduction et Protocole de Diffusion

33

## Diffusion Fiable Temporisée

---

- **Diffusion fiable temporisée = diffusion fiable + borne**

- Système de communication synchrone
- **Borne** : il existe une constante  $\Delta$  telle que si un message  $m$  est diffusé à l'instant  $t$ , alors aucun processus correct ne délivre  $m$  après le temps  $t + \Delta$ .

24/09/2012

ARA: Introduction et Protocole de Diffusion

34

## Garantie d'ordre (1)

- **Ordre Total**
  - Les messages sont délivrés dans le même ordre à tous leurs destinataires.
- **Ordre FIFO**
  - si un membre diffuse  $m_1$  puis  $m_2$ , alors tout membre correct qui délivre  $m_2$  délivre  $m_1$  avant  $m_2$ .
- **Ordre Causal**
  - si  $broadcast(m_1)$  précède causalement  $broadcast(m_2)$ , alors tout processus correct qui délivre  $m_2$ , délivre  $m_1$  avant  $m_2$ .

Observations :

- La propriété d'ordre total est indépendante de l'ordre d'émission
- Les propriétés d'ordre FIFO et Causal sont liées à l'ordre d'émission

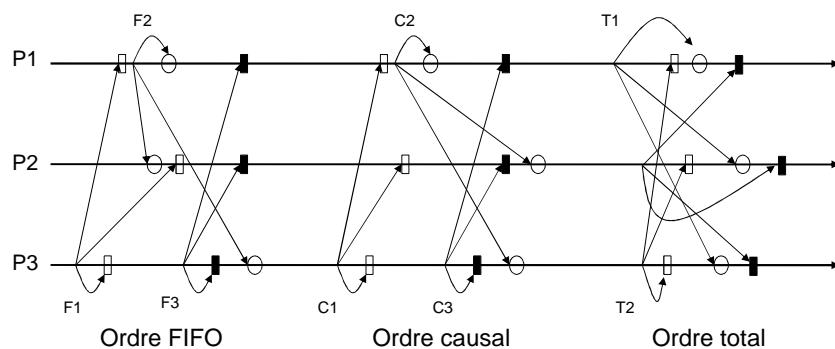
24/09/2012

ARA: Introduction et Protocole de Diffusion

35

## Garantie d'ordre (2)

### ■ Exemple



24/09/2012

ARA: Introduction et Protocole de Diffusion

36

## Garantie d'ordre (3)

### ■ Remarques :

- une diffusion causale est nécessairement FIFO (la diffusion causale peut être vue comme une généralisation de l'ordre FIFO à tous les processus du groupe)
- L'ordre FIFO et l'ordre causal ne sont que des ordres partiels : ils n'imposent aucune contrainte sur l'ordre de délivrance des messages diffusés concurremment
- l'ordre total n'a pas de lien avec l'ordre FIFO et l'ordre causal : il est à la fois plus fort (ordre total des messages délivrés) et plus faible (aucun lien entre l'ordre de diffusion et l'ordre de délivrance)

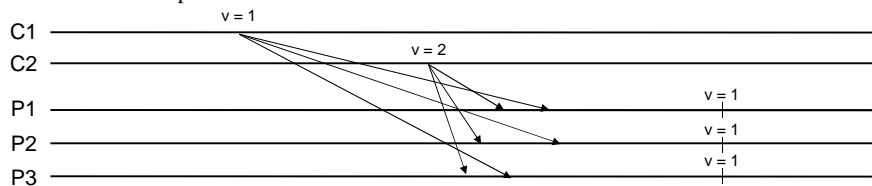
24/09/2012

ARA: Introduction et Protocole de Diffusion

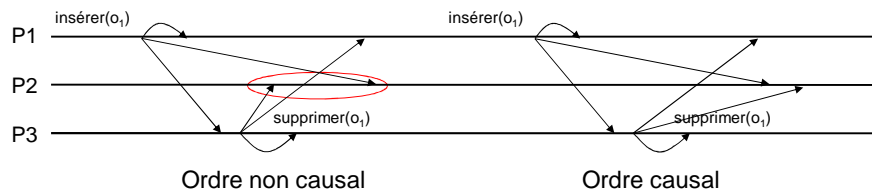
37

## Garantie d'ordre - Exemple utilisation (4)

**Ordre total** : permet de maintenir la cohérence des répliques d'un serveur en présence d'écrivains multiples.



**Ordre causal** : permet de préserver à faible coût l'enchaînement d'opérations logiquement liées entre elles.



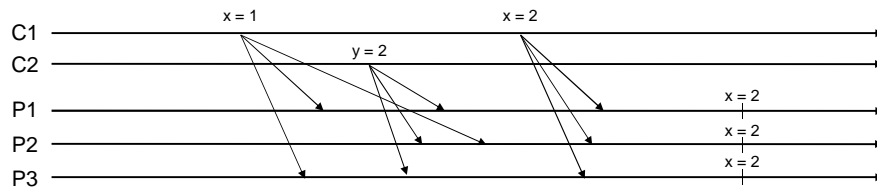
24/09/2012

ARA: Introduction et Protocole de Diffusion

38

## Garantie d'ordre - Exemple utilisation (5)

**Ordre FIFO** : permet de maintenir la cohérence des répliques d'un serveur en présence d'un écrivain unique.

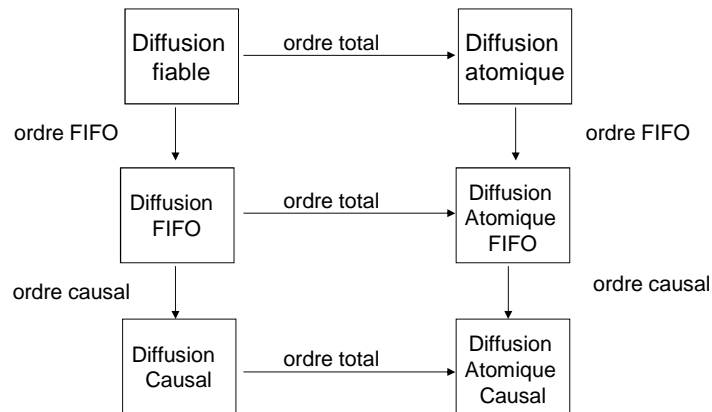


Les trois garanties d'ordre FIFO, causal et total sont plus ou moins coûteuses à implémenter : choisir celle juste nécessaire à l'application visée.

## Types de Diffusion Fiable (1)

- Diffusion FIFO = Diffusion fiable + Ordre FIFO
- Diffusion Causal (CBCAST) = Diffusion fiable + Ordre Causal
- Diffusion Atomique (ABCAST) = Diffusion fiable + Ordre Total
- Diffusion Atomique FIFO = Diffusion FIFO + Ordre Total
- Diffusion Atomique Causal = Diffusion Causal + Ordre Total

## Types de Diffusion Fiable (2)



Relation entre les primitives de diffusion [Hadzilacos & Toueg]

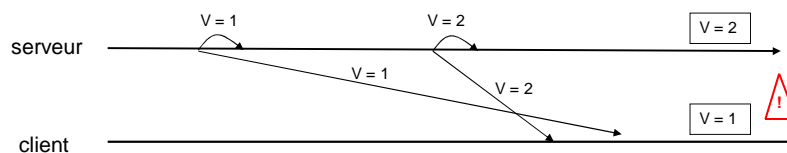
24/09/2012

ARA: Introduction et Protocole de Diffusion

41

## Diffusion FIFO - motivation

- Dans la diffusion fiable il n'y a aucune spécification sur l'ordre de délivrance des messages.



24/09/2012

ARA: Introduction et Protocole de Diffusion

42

# Diffusion FIFO

---

- **Diffusion FIFO = diffusion fiable + ordre FIFO**
  - **Ordre FIFO** : si un membre diffuse  $m_1$  puis  $m_2$ , alors tout membre correct qui délivre  $m_2$  délivre  $m_1$  avant  $m_2$ .
  - Ayant un algorithme de diffusion fiable, il est possible de le transformer dans un algorithme de diffusion FIFO

24/09/2012

ARA: Introduction et Protocole de Diffusion

43

## Diffusion FIFO – algorithme (1)

---

Processus  $p$  :

**Variable locale :**

pendMsg =  $\emptyset$ ; /\* message pas encore délivré \*/  
next [ N ] = 1 pour tous processus; /\*seq# du prochain message de  $q$  que  $p$  doit délivrer \*/

**FIFO\_broadcast (m)**

Real\_broadcast(m); /\*m estampillé avec seq# \*/

**upon Real\_deliver(m) do**

s = sender (m);

pendMsg  $\cup$  = { m }

**while** ( $\exists m' \in$  pendMsg : sender( $m'$ ) = s and seq#( $m'$ ) = next[s]) **do**

**FIFO\_delivrer(m)** /\* délivrer le message \*/

next[s]++;

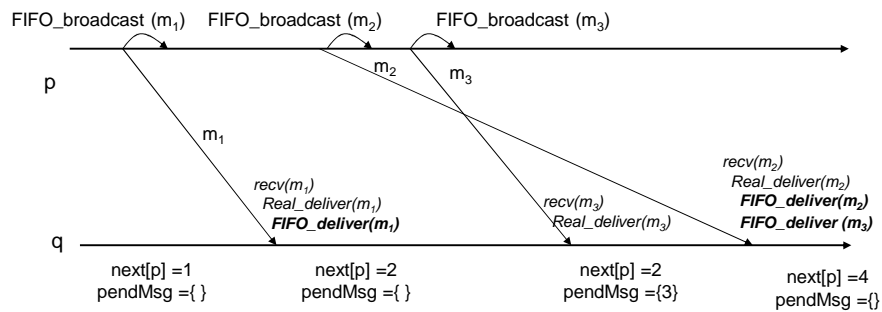
pendMsg -= { m };

24/09/2012

ARA: Introduction et Protocole de Diffusion

44

## Diffusion FIFO – algorithme (2)



24/09/2012

ARA: Introduction et Protocole de Diffusion

45

## Diffusion Causal - CBCAST

- **Diffusion Causal = diffusion fiable + ordre Causal**
  - Objectif : délivrer les messages dans l'ordre causal de leur diffusion.
  - **Ordre causal** : si  $broadcast(m_1)$  précède causalement  $broadcast(m_2)$ , alors tout processus correct qui délivre  $m_2$ , délivre  $m_1$  avant  $m_2$ .
    - $broadcast_p(m_1) \rightarrow broadcast_q(m_2) \Leftrightarrow deliver_p(m_1) \rightarrow deliver_q(m_2)$
  - Causal Order  $\rightarrow$  FIFO order
  - Fifo Order  $\not\rightarrow$  Causal Order

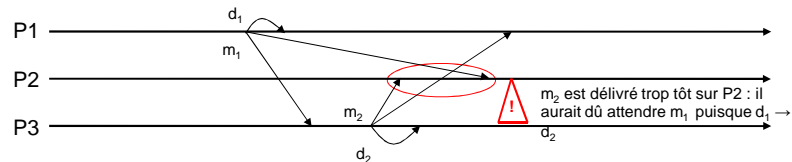
24/09/2012

ARA: Introduction et Protocole de Diffusion

46

# Diffusion Causal

$\text{broadcast}_p(m_1) \rightarrow \text{broadcast}_q(m_2) \Leftrightarrow \text{deliver}_p(m_1) \rightarrow \text{deliver}_q(m_2)$



- Un algorithme de diffusion FIFO peut être transformé dans un algorithme de diffusion causal :
  - transporter avec chaque message diffusé l'historique des messages qui le précèdent causalement.

24/09/2012

ARA: Introduction et Protocole de Diffusion

47

# Diffusion Causal – algorithme

Processus  $p$  :

**Variable locale :**

seqMsg = vide; /\* sequence de messages que  $p$  a délivré depuis sa diffusion précédente \*/  
 delv =  $\emptyset$ ; /\* messages délivrés \*/

**Causal\_broadcast (m)**

FIFO\_broadcast(seqMsg  $\Theta$  m); /\* diffuser tous les messages délivrés depuis la diffusion précédente + m \*/

seqMsg = vide;

**upon FIFO\_deliver( $m_1, m_2, \dots, m_n$ ) do**

**for**  $i=1..n$  **do**

**if**  $m_i \notin \text{delv}$  **then**

**Causal\_delivrer( $m_i$ )** /\* délivrer le message \*/

delv  $\cup = \{ m_i \}$

seqMsg  $\Theta = m_i$  /\*ajouter  $m_i$  à la fin de la seqMsg \*/

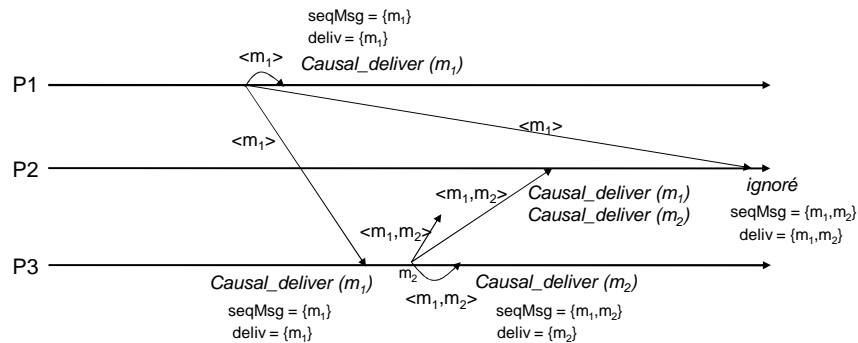
24/09/2012

ARA: Introduction et Protocole de Diffusion

48



## Diffusion Causal – algorithme



- **Avantage :**  
La délivrance d'un message n'est pas ajournée en attente d'une condition
- **Inconvénient**
  - Taille des messages

24/09/2012

ARA: Introduction et Protocole de Diffusion

49

## Diffusion Causal – algorithme avec horloges vectorielles (sans garantie de remise)

➤ Historique de messages peut être représenté au moyen d'une d'horloge vectorielle

Processus **P** :

HV[k]<sub>m</sub> venant de P<sub>j</sub> représente :

- k = j : le nombre de messages diffusés par P<sub>j</sub>
- k ≠ j : le nombre de diffusions de P<sub>k</sub> délivrées par P<sub>j</sub> avant diffusion de m.

**Variables locales :**

HV[N] = {0, 0, ..., 0}

FA = ∅

**Causal\_Broadcast(m)**

HV[i] += 1

estampiller m avec HV;

envoyer m à tous les processus y compris p

Isis - Birman 91

24/09/2012

ARA: Introduction et Protocole de Diffusion

50

## Diffusion Causal – algorithme avec horloges vectorielles (sans garantie de remise)

**Upon recv(m, HV[]<sub>m</sub>) :**

s = sender (m);

FA.queue(< m, HV[]<sub>m</sub> >)

**delay** delivery of m **until**

(1) HV[s]<sub>m</sub> = HV[s]<sub>p</sub> + 1 **and** (2) HV[k]<sub>m</sub> ≤ HV[k]<sub>p</sub> pour tout k; k ≠ s

// D'autres réceptions se produisent pendant l'attente. On attend d'avoir délivré :

// 1- toutes les diffusions précédentes effectuées par s,

// 2- toutes les diffusions délivrées par s avant la diffusion de m

FA.dequeue(< m, HV[]<sub>m</sub> >)

deliver(m);

HV[s]<sub>p</sub> +=1;

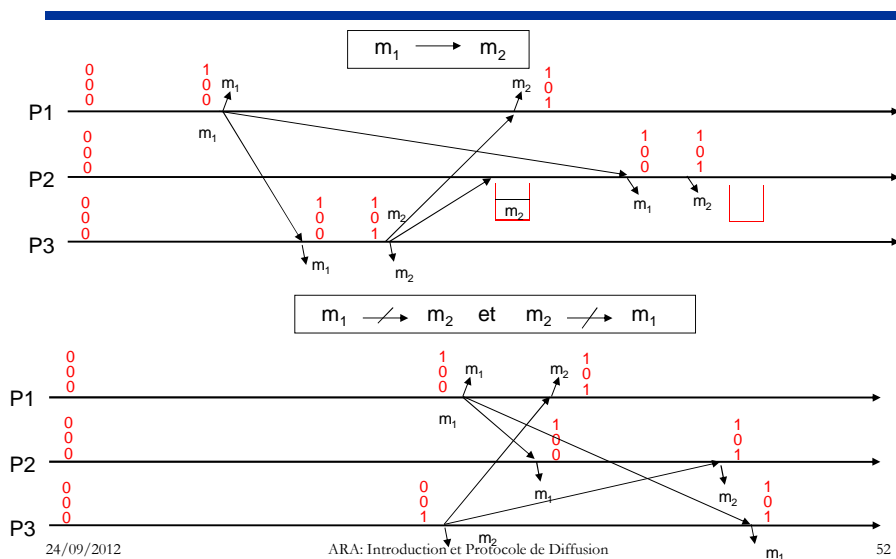
- (1) : assure que p a délivré tous les messages provenant de s qui précèdent m
- (2) : assure que p a délivré tous les messages délivrés par s avant que celui-ci envoie m

24/09/2012

ARA: Introduction et Protocole de Diffusion

51

## Diffusion Causal – algorithme avec horloges vectorielles - Exemple



24/09/2012

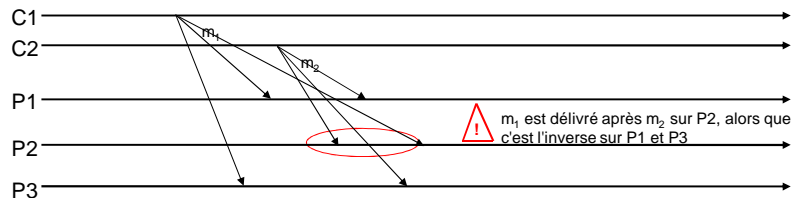
ARA: Introduction et Protocole de Diffusion

52

# Diffusion Atomique - ABCAST

## ■ Diffusion atomique = diffusion fiable + ordre total

- Tous les processus corrects délivrent le même ensemble de messages dans le même ordre.
- **Ordre Total** : si les processus corrects  $p$  et  $q$  délivrent tous les deux les messages  $m$  et  $m'$ , alors  $p$  délivre  $m$  avant  $m'$  seulement si  $q$  délivre  $m$  avant  $m'$ .
- Exemple d'une diffusion **pas** atomique



24/09/2012

ARA: Introduction et Protocole de Diffusion

53

# Diffusion Atomique - ABCAST

- **Résultat fondamental** : Dans un système asynchrone avec pannes franches, la diffusion atomique est équivalent au consensus.

Consensus impossible dans un système asynchrone avec pannes franches  $\Leftrightarrow$  Diffusion atomique impossible dans un système asynchrone avec pannes franches

- Si on dispose d'un algorithme de diffusion atomique, on sait réaliser le consensus
  - Chaque processus diffuse atomiquement sa valeur proposée à tous les processus
  - Tous les processus reçoivent le même ensemble de valeurs dans les même ordre
  - Ils décident la première valeur
- Si on dispose d'un algorithme de consensus, on sait réaliser la diffusion atomique

Diffusion Atomique  $\Leftrightarrow$  Consensus

Chandra & Toueg 1996

24/09/2012

ARA: Introduction et Protocole de Diffusion

54

## Diffusion Atomique - ABCAST

---

### ■ Remarques :

- ABCAST n'est pas réalisable dans un système asynchrone si on suppose l'existence de fautes (d'après FLP).
- ABCAST est réalisable (n nodes):
  - Avec un détecteur de pannes de classe P ou S en tolérant n-1 pannes
  - Avec détecteur de pannes de classe  $\diamond$  S en tolérant n/2 - 1 pannes
  - Avec un protocole de diffusion fiable temporisée en utilisant des hypothèse de synchronisme.

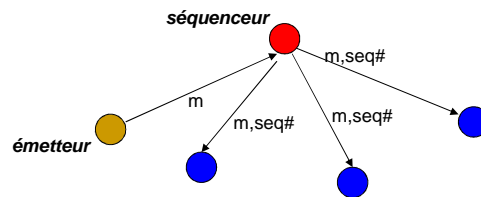
## Diffusion Atomique - algorithmes

---

- **Un protocole ABCAST doit garantir l'ordre de remise de messages et tolérer les défaillances**
- **L'ordre d'un protocole ABCAST peut être assuré par :**
  - Un ou plusieurs séquenceurs
    - séquenceur fixe
    - séquenceur mobile
  - Les émetteurs
    - À base de privilège
  - Les récepteurs
    - Accord des récepteurs
- Remarques: les algorithmes présentés à la suite ne traitent pas les pannes

## Diffusion totalement ordonnée : Séquenceur fixe

---



24/09/2012

ARA: Introduction et Protocole de Diffusion

57

## Diffusion totalement ordonnée : Séquenceur fixe

---

### ■ Principe :

- Un processus, le séquenceur, est choisi parmi tous les processus
  - Responsable de l'ordonnement des messages
- Émetteur envoie le message  $m$  au séquenceur
  - Séquenceur attribue un numéro de séquence  $seq\#$  à  $m$
  - Séquenceur envoie le message à tous les processus.

24/09/2012

ARA: Introduction et Protocole de Diffusion

58

# Séquenceur fixe - algorithme

Processus *P* :

**Variables locales :**

nextdelv = 1;  
pend = ∅;

**Emetteur :**

**OT\_broadcast (m)**  
send m au séquenceur;

**Séquenceur :**

**intit :**  
seq#=1;  
**upon revc(m) do**  
send (m,seq#) à to processus  
seq#++;

**Destinateur :**

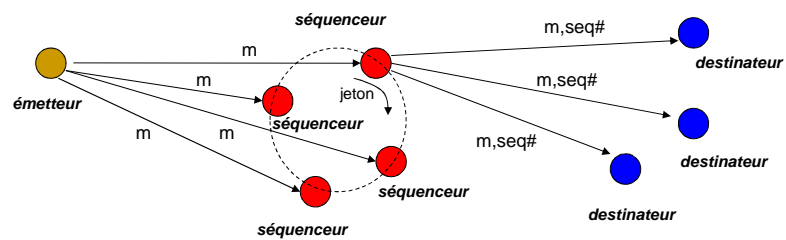
**upon revc(m) do**  
pend ∪= {m}  
**while** (∃ (m',seq#) ∈ pend : seq#'=nextdelv) **do**  
**OT\_deliver (m')**  
nextdelv++;  
pend -= {m'}

24/09/2012

ARA: Introduction et Protocole de Diffusion

59

# Diffusion totalement ordonnée : Séquenceur mobile



24/09/2012

ARA: Introduction et Protocole de Diffusion

60

## Diffusion totalement ordonnée : Séquenceur mobile

### ■ Principe

- Un groupe de processus agissent successivement comme séquenceur
- Un message est envoyé à tous les séquenceurs.
- Un jeton circule entre les séquenceurs, contenant :
  - un numéro de séquence
  - Liste de messages déjà séquencés
- Lors de la réception du jeton, un séquenceur :
  - attribue un numéro de séquence à tous les messages pas encore séquencés et envoie ces messages aux destinataires
  - Ajoute les messages envoyés dans la liste du jeton

### ■ Avantages

- répartition de charge

### ■ Inconvénients

- Taille jeton
- coût circulation du jeton

24/09/2012

ARA: Introduction et Protocole de Diffusion

61

## Séquenceur mobile - algorithme

### Variables locales :

```
nextdelv = 1;  
pend = ∅;
```

### Emetteur :

```
OT_broadcast (m)  
send m à tous les séquenceurs;
```

### Destinateur :

```
upon revc(m) do  
  pend ∪= {m}  
  while (∃ (m',seq#) e pend : seq#=nextdelv) do  
    OT_deliver (m')  
    nextdelv++;  
    pend -= {m'}
```

### Séquenceur :

```
intit :  
  rec = ∅;  
  if (p= si)  
    token.seq# = 1  
    token.liste = ∅;  
upon revc(m) do  
  rec ∪= {m}  
upon revc(token) do  
  for each m' in rec \ token.liste do  
    send (m',token.seq#) à tous les  
      destinataires  
    token.seq#++;  
    token.liste ∪= {m}  
  send (token) au prochain séquenceur
```

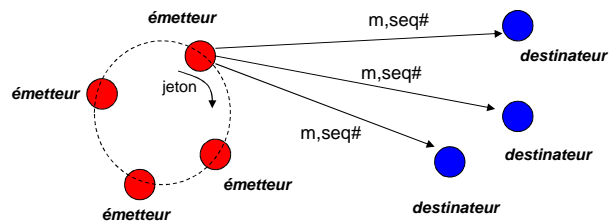
24/09/2012

ARA: Introduction et Protocole de Diffusion

62

## Diffusion totalement ordonnée : à base de priorité

---



24/09/2012

ARA: Introduction et Protocole de Diffusion

63

## Diffusion totalement ordonnée : à base de priorité

---

### ■ Principe

- Un jeton donne le droit d'émettre
- Jeton circule entre les émetteurs contenant le numéro de séquence du prochain message à envoyer.
- Lorsqu'un émetteur veut diffuser un message, il doit attendre avoir le jeton
  - attribue un numéro de séquence aux messages à diffuser
  - envoie le jeton aux prochains émetteurs

### ■ Inconvénients

- Nécessaire de connaître les émetteurs (pas adéquat pour de groupe ouvert)
- Pas très équitable : un processus peut garder le jeton et diffuser un nombre important de messages en empêchant les autres de le faire

24/09/2012

ARA: Introduction et Protocole de Diffusion

64



## Diffusion totalement ordonnée : à base de priorité

---

### Variables locales :

```
nextdelv = 1;
pend = ∅;
send_pend = ∅;
```

### Destinateur :

```
upon revc(m) do
  pend ∪= { m }
  while (∃ (m',seq#) ∈ pend : seq#'=nextdelv) do
    OT_deliver (m')
    nextdelv++;
    pend -=(m')
```

### Emetteur :

```
intit :
  send_pend = ∅;
  if (p=s1)
    token.seq# = 1
```

### procedure OT\_broadcast (m)

```
send_pend ∪= {m}
```

### upon revc(token) do

```
for each m' in send_pend do
  send (m',token.seq#) à tous les
  destinateurs
  token.seq#++;
  send_pend ∪= ∅;
  send (token) au prochain émetteur
```

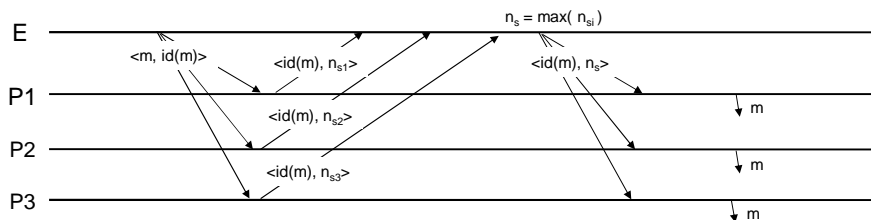
## Diffusion totalement ordonnée : accord récepteurs

---

### ■ Principe

- Les processus se concertent pour attribuer un numéro de séquence à chaque message. Chaque diffusion nécessite deux phases :
  - diffusion du message et collecte des propositions de numérotation
  - choix d'un numéro définitif et diffusion du numéro choisi

## Accord récepteurs



Les numéros proposés sont *<date logique réception, identité récepteur>* pour assurer un ordre total. Chaque processus maintient une file d'attente des messages en attente de numérotation définitive, triée de façon croissante sur les numéros.

24/09/2012

ARA: Introduction et Protocole de Diffusion

67

## Accord récepteurs : algorithm

- *E* diffuse le message *m* au groupe :
  - sur réception de *m*,  $P_j$  attribue à *m* son numéro de réception provisoire, le marque *non délivrable*, et l'insère dans sa file d'attente
  - puis  $P_j$  renvoie à *E* le numéro provisoire de *m* comme proposition de numéro définitif
  - quand *E* a reçu tous les numéros proposés, il choisit le plus grand comme numéro définitif et le rediffuse
  - sur réception du numéro définitif,  $P_j$  réordonne *m* dans sa file et le marque délivrable
  - puis  $P_j$  délivre tous les messages marqués *délivrable* situés en tête de la file d'attente

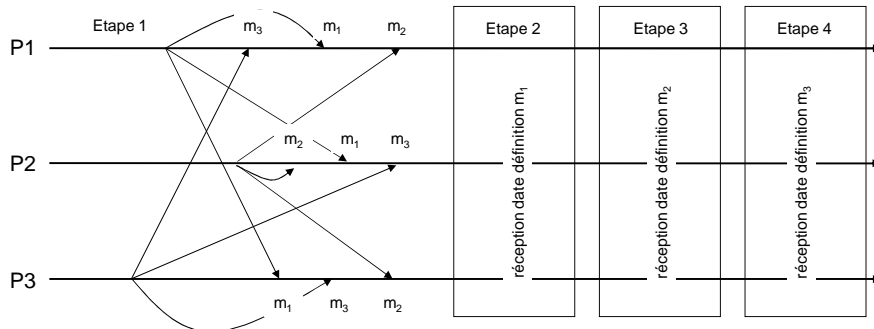
Birman - Joseph 87

24/09/2012

ARA: Introduction et Protocole de Diffusion

68

## Accord récepteurs : exemple



P1, P2 et P3 diffusent simultanément les trois messages m<sub>1</sub>, m<sub>2</sub> et m<sub>3</sub> (seuls les messages de l'étape 1 sont représentés).

Note : il s'agit d'un **exemple** d'exécution ; la date définitive d'un message n'arrive **pas nécessairement** dans le même laps de temps sur tous les processus, ni dans le même ordre pour les différents messages.

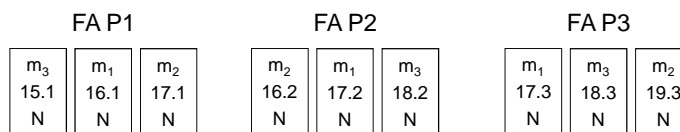
24/09/2012

ARA: Introduction et Protocole de Diffusion

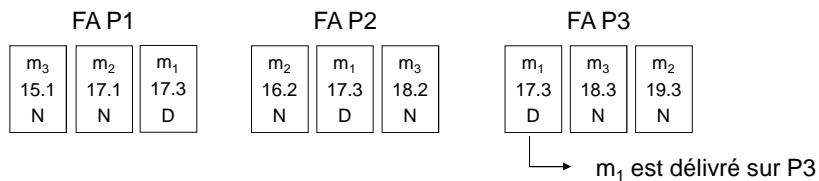
69

## Accord récepteurs : exemple (cont.)

Etape 1 : réception des messages et proposition de numérotation



Etape 2 : réception de la date de définitive de m<sub>1</sub> : 17.3



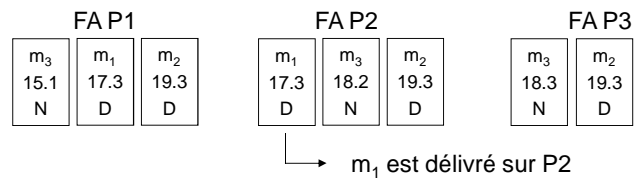
24/09/2012

ARA: Introduction et Protocole de Diffusion

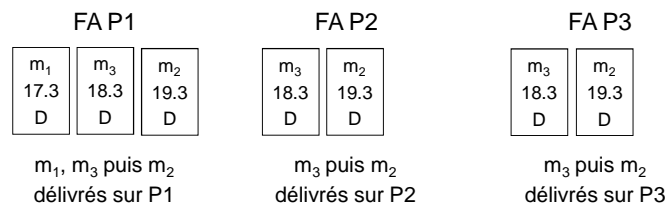
70

## Accord récepteurs : exemple (cont.)

Etape 3 : réception de la date de définitive de  $m_2$  : 19.3



Etape 4 : réception de la date de définitive de  $m_3$  : 18.3



24/09/2012

ARA: Introduction et Protocole de Diffusion

71

## Diffusion totalement ordonnée tolérance aux fautes

### ■ Quelques mécanismes :

- Détecteurs de défaillance
- Redondance
  - Exemple : séquenceur
- Stabilité des messages
  - Un message est *k-stable* s'il a été reçu par  $k$  processus.
    - $f$  défaillances : un messages  $(f+1)$ -stable a été reçu par au moins 1 processus correct. Sa délivrance peut être garantie.
- Pertes de messages
  - Numérotation des messages.

24/09/2012

ARA: Introduction et Protocole de Diffusion

72

## Bibliographie

---

- X. Défago and A. Schiper and P. Urban Total order broadcast and multicast algorithms: Taxonomy and survey, *ACM Comput. Surv.*, 36(4):372—421.
- K. Birman, T. Joseph. Reliable communication in presence of failures. *ACM Transactions on Computer Systems*, Vol. 5, No. 1, Feb. 1987
- K. Birman and R. Cooper. The ISIS Project: Real Experience with a Fault Tolerant Programming System. *Operating Systems Review*, Apr. 1991, pages 103-107.
- K. Birman, A. Schiper and P. Stephenson. Lightweight Causal and Atomic Group Multicast. *ACM Transactions on Computer Systems*, Aug. 1991, (3):272-314.
- R. Guerraoui, L. Rodrigues. *Reliable Distributed Programming*, Springer, 2006
- V. Hadzilacos and S. Toueg. A Modular Approach to Fault-tolerant Broadcasts and Related Problems. *Technical Report TR94-1425*. Cornell University.
- T. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems, *Journal of the ACM*, Vol. 43. No. 2, 1996, pages 225-267.

---

## Epidemic Broadcast

*Diffusion Epidémique*

# Epidemic Broadcast

---

- **The broadcast algorithms that we have seen till now are not scalable**
  - They consider a set of processes known by all processes from the beginning.
- **Epidemic algorithms are effective solution for disseminating in large scale and dynamic systems.**
  - They do not provide deterministic broadcast guarantees but just make probabilistic claims about such guarantees.
- **An epidemic broadcast uses a randomized approach where all the participants in the protocol should collaborate in the same manner to disseminate information.**

06/11/07

ARA: Consensus - Broadcast

75

# Epidemic Broadcast

---

- When a process  $p$  wishes to send a broadcast message, it selects  $k$  processes at random and sends the message to them
  - $k$  is a typical configuration parameter called *fanout*.
- Upon receiving a message from  $p$  for the first time, a process  $q$  repeats the same procedure of  $p$ 's :  $q$  selects  $k$  gossip targets processes and forwards the message to them.
  - If a node receives the message twice, it simply discards the message
    - Each process needs to keep track of which messages it has already seen and delivered. The size of this buffer is also a scalable constraints.
- The step consisting of receiving a message and forwarding it is called a *round*.
  - An epidemic algorithm usually performs a *maximum number of rounds*  $r$  for each message.

06/11/07

ARA: Consensus - Broadcast

76

## Epidemic broadcast

---

- **Epidemic broadcast can only be applied to applications that do not require full reliability.**
  - The cost of full reliability is usually not acceptable in large scale systems.
  - However, it is possible to build scalable randomized epidemic algorithms which provide good reliability guarantees.
  - It exhibit a very stable behavior even in the presence of failures.

06/11/07

ARA: Consensus - Broadcast

77

## Epidemic Broadcast

---

- **Parameters associated with the configuration of gossip protocols :**
  - **Fanout ( $k$ ):** number of nodes that are selected as gossip targets by a node for each message that is received by the first time.
    - Tradeoff associated between desired reliability level and redundancy level of the protocol.
  - **Maximum rounds ( $r$ ):** maximum number of times a given gossip message is retransmitted by nodes.
    - Each message carries a *round value*, which is increased each time the message is retransmitted.
    - **Modes :**
      - *Unlimited mode*: the parameter maximum round is undefined
      - *Limited mode* : the parameter maximum round is defined with a value greater than 0.
    - Higher value: higher reliability as well as message redundancy.

06/11/07

ARA: Consensus - Broadcast

78

# Epidemic Broadcast

---

## ■ Probabilistic Broadcast

### ➤ Properties

- *Probabilistic validity* : There is a given probability such that for any two correct processes  $p_i$  and  $p_j$ , every message broadcast by  $p_i$  is eventually delivered by  $p_j$  with this probability.
- *No duplication* : No message is delivered more than once by a process
- *No creation* : If a message  $m$  is delivered by some process  $p_j$ , then  $m$  was previously broadcast by some process  $p_i$ .

06/11/07

ARA: Consensus - Broadcast

79

# Epidemic Broadcast

---

## ■ Strategies

- *Eager push approach* : Nodes send message to selected nodes as soon as they receive them for the first time
- *Pull approach* : Periodically, nodes query random selected nodes for information about recently received messages. When they receive information about a message they did not receive yet, they explicitly request the message to their neighbors.
- *Lazy push approach* : When a node receives a message for the first time, it gossips only the message identifier. If a node receives a identifier of a message it has not received, it makes an explicitly pull request.
- *Hybrid approach* : First phase uses a push gossip to disseminate a message in best-effort manner. A second phase of pull gossip is used to recover messages not received in the first phase.

06/11/07

ARA: Consensus - Broadcast

80



# Eager Push Epidemic Broadcast

---

## Algorithm

```

Init :
  delivered =  $\emptyset$ 

Epid_broadcast (m)
  gossip(self, m, maxrounds);

upon recv (pi, <src,m, r>)
  if ( $m \notin$  delivered)
    delivered = delivered  $\cup$  {m}
    Epid_deliver(src,m)
  if (r > 0)
    gossip(self, m, maxrounds - 1);

Function chose-targets (ntargets)
  targets =  $\emptyset$ 
  while ( | targets| < ntargets ) do
    candidate = random ( $\Pi$ )
    if (candidate  $\in$  targets) and (candidate !=
    self)
      targets = targets  $\cup$  {candidate};
  return targets

procedure gossip (src,msg,round)
  for i  $\in$  chose-targets(fanout) do send (i, msg, round)
  
```

06/11/07

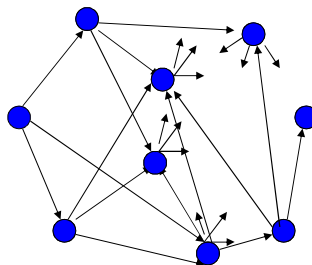
ARA: Consensus - Broadcast

81

# Eager Push Epidemic Broadcast

---

## Execution example



Fanout = 3 ; Maxround = 3

06/11/07

ARA: Consensus - Broadcast

82

## Epidemic Broadcast

---

- **Ideally, one would like to have each participant to select gossip targets at random from the entire system, as shown in the previous example.**
  - Realistic if it is deployed within a moderate sized cluster.
  - Such approach is not scalable :
    - High memory cost to maintain full membership information.
    - High cost of ensuring the update of such information.
- **Solution:**
  - Gossip-based (epidemic) broadcast protocols rely on *partial view*, instead of full membership information.

06/11/07

ARA: Consensus - Broadcast

83

## Epidemic Broadcast : Partial view

---

- **Partial view**
  - A process just knows a small subset of the entire system membership, from which it can select nodes to whom relay gossip messages
  - The membership protocol establishes *neighboring* association among nodes.
    - It must maintain the partial view at each node in face of dynamic changes in the system membership.
      - Joining of new nodes, crashes of nodes, etc.
  - A partial view must be a tradeoff between *scalability* against *reliability*
    - Small views scale better, while large views reduce the probability that processes become isolated or that network partitions occur.
  - **Overlay**
    - Partial views of all nodes of the system define a graph

06/11/07

ARA: Consensus - Broadcast

84

## Epidemic Broadcast : Partial view

---

- **Partial View Properties : related to the graph properties of the overlay defined by the partial view of all nodes**
  - *Connectivity* : the overlay should be connected : there should be at least one path from each node to all other nodes.
  - *Degree Distribution* : number of edges of the node.
    - *In-degree* of node  $n$  : number of nodes that have  $n$  in their partial view. It provides a measure of *reachability*.
    - *Out-degree* of node  $n$  : number of nodes in  $n$ 's view: measure of the importance of that node to maintain the overlay.
  - *Average Path Length* : the average of all shortest paths between all pair of nodes in the overlay.
  - *Accuracy* of node  $n$ : number of neighbors of  $n$  that have not failed divided by the total number of neighbors of  $n$ .

06/11/07

ARA: Consensus - Broadcast

85

## Epidemic Broadcast : Partial view

---

- **Strategies to maintain partial view**
  - *Reactive strategy* : a partial view only changes in response to some external event such as a joining of a node, a crash of a node, etc.
  - *Cyclic strategy* : A partial view is update every  $\Delta T$  units of time, as a result of some periodic process that usually involves the exchange of information with one or more neighbors.
  - *Mixing strategy* : the partial view membership is included in the epidemic broadcast protocol
    - Whenever a process forwards a message, it also includes in it a set of processes it knows. Process that receives this message can update its own list of known processes.
    - It does not introduce extra communication to maintain membership.

06/11/07

ARA: Consensus - Broadcast

86

## Epidemic Broadcast : Partial view

---

### ■ Example : CYCLON

- Cyclic strategy : exchanging of view periodically among neighbors (*shuffling operation*), at a fixed period  $\Delta T$ .
- A node keeps in cache pointers to its neighbors
  - Each pointer to a neighbor has a predictable lifetime
  - Field *age* : express the age of the pointer in  $\Delta T$  intervals since the moment it was created.

06/11/07

ARA: Consensus - Broadcast

87

## Epidemic Broadcast : Partial view

---

### ■ Example : CYCLON (cont.)

- Shuffling by node  $p$  :
  1. Increase by one the *age* of all its neighbors
  2. Select neighbor  $q$  with the highest age among all neighbors and  $l$  other random neighbors.
  3. Send the  $l$  random neighbors to  $q$
  4. Upon receiving from  $q$  a subset of  $l$  of  $q$ 's neighbors :
    - discard those neighbors already in  $p$ 's cache
    - update  $p$ 's cache to include all remaining entries by firstly using empty caches slots (if any), and secondly replacing entries.

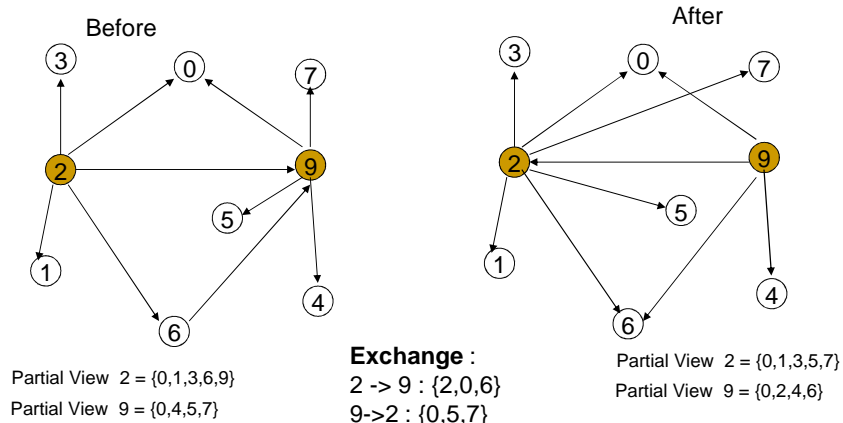
06/11/07

ARA: Consensus - Broadcast

88

## Epidemic Broadcast : Partial view

### Cyclon : example execution



06/11/07

ARA: Consensus - Broadcast

89

## Gossip protocol in ad hoc Networks

- **An ad hoc network is a multi-hop wireless network with no fixed infrastructure**
  - Node broadcasts a message which is received by all nodes within one hop (neighbors)
- **Gossiping protocol Gossip(p)[HHL06]**
  - A source node sends the message  $m$  with probability 1.
  - Upon reception of  $m$ 
    - first time,
      - it broadcasts  $m$  with probability  $p$
      - it discards  $m$  with probability  $1-p$
    - Otherwise it discards  $m$

24/09/2012

ARA: Introduction et Protocole de Diffusion

90

## Gossip protocol in ad hoc Networks

---

- **If the source has few neighbors, chance that none of them will gossip and the algorithm dies.**
  - Solution : Gossip ( $p,k$ )
    - Gossip with probability 1 for the  $k$  hops before continuing to gossip with probability  $p$ .
      - Gossip (1,1) is equivalent to flooding.
      - Gossip ( $p,0$ ) : even the source gossips with probability  $p$ .

## Bibliographie

---

- R. Guerraoui, L. Rodrigues. *Reliable Distributed Programming*, Springer, 2006
- O. Babaoglu, S. Toueg. *Understanding Non-blocking Atomic Commitment*. Distributed Systems. ACM Press (S. Mullender Ed.), 1993.
- Michel Raynal. Revisiting the Non-Blocking Atomic Commitment Problem. Technical Report IRISA 1997.
- J. C. A. Leitão. *Gossip-based broadcast protocols*. Master thesis's. 2007.
- P.Eugster, R.Guerraoui,A. Kermarrec and L. Massoulié. *From Epidemics to Distributed Computing*. IEEE Computer, 37,pages 60-67.

## Bibliographie

---

- K. Birman and T. Joseph . *Exploiting virtual synchrony in distributed systems*. Proceedings of the eleventh ACM Symposium on Operating systems principles, pages 123-138, 1987.
- S. Voulgaris, D. Gavidia, M. Stten. *Cyclon : Inexpensive membership management for unstructured p2p overlays*. Journal of Network and System Management. Vol 13, pages 197-217, 2005.
- Z.J.Haas, J. Halpern, L. Li. *Gossip-Based Ad Hoc Routing*. IEEE Transactions on Network, Vol. 14, N. 13, pages 479-491, 2006