Systèmes répartis

Causalité et datation

Gérard Padiou

Département Informatique et Mathématiques appliquées
ENSEEIHT

Octobre 2012

plan

Plan

Problème de base et difficultés

Comment dater les événements d'une exécution répartie ?

- Pas d'horloge globale ;
- Tous les événements ne sont pas causalement liés ;
- Datation cohérente avec la relation causale :

$$\forall e, e' : e \prec e' \Rightarrow d_e < d_{e'}$$

Exemple

- Idée : $\rightarrow$ Utiliser les horloges de chaque site
- Risque : Datation incohérente de l'événement de réception d'un message : la date de réception précède la date d'émission.
- Possible si l'horloge du site de réception est en retard (suffisamment) sur celle du site de l'émetteur.

Difficultés

- Cohérence avec la causalité ;
- Datation définissant un ordre total ;

Solutions

Nécessite une synchronisation des horloges locales :

$$\texttt{invariant} \quad Max(h_i \ : \ i = 1, N) - Min(h_i \ : \ i = 1, N) < \epsilon$$

Protocole de synchronisation d'horloges possible mais dans des contextes réseaux assurant une certaine qualité de service ou par l'usage d'un signal diffusé (horloge atomique).

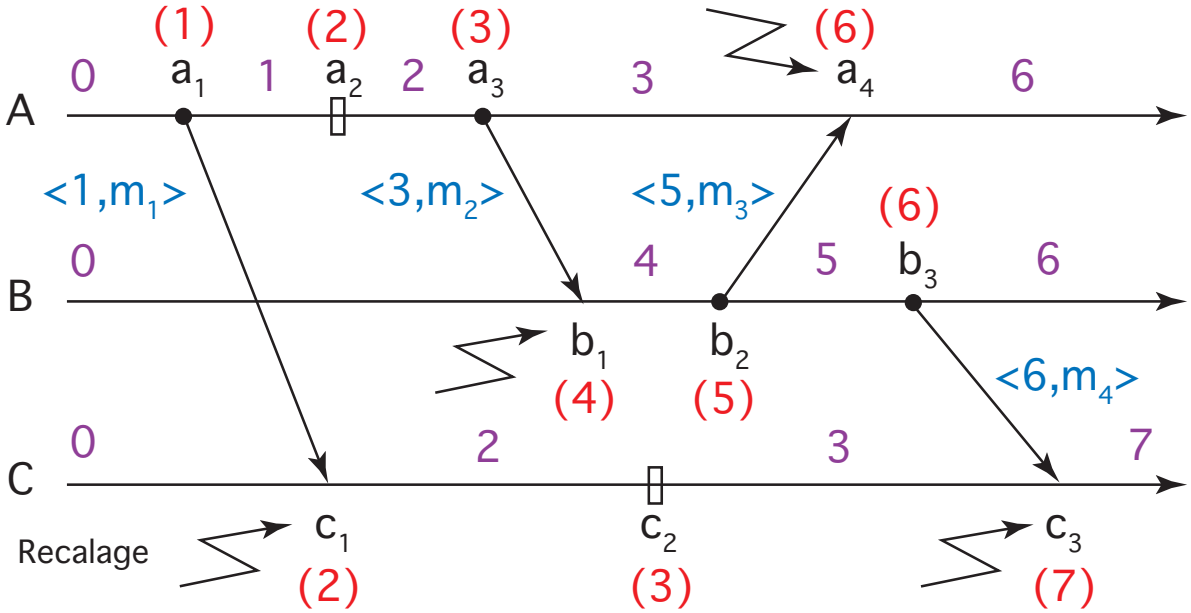
Temps logique

Principes de base

- Compter les événements au lieu du temps réel ;
- Associer à chaque site une horloge logique locale ;
- Surcharger les messages avec leur date d'émission ;
- Recaler si nécessaire l'horloge locale d'un site lors de chaque réception de message ;
- Avantage : vision plus abstraite d'un calcul réparti.

Une tentative...

- Un compteur « horloge » sur chaque site ;
- Surcharge des messages et recalage de compteur.



☺ $e \prec e' \Rightarrow d(e) < d(e')$;

☺ Mêmes dates $\Rightarrow$ pas causalement liés ;

☹ mais $d(e) < d(e') \not\Rightarrow e \prec e'$ (ex : $d(c_2) < d(b_3) \not\Rightarrow c_2 \prec b_3$)

Horloge de Lamport

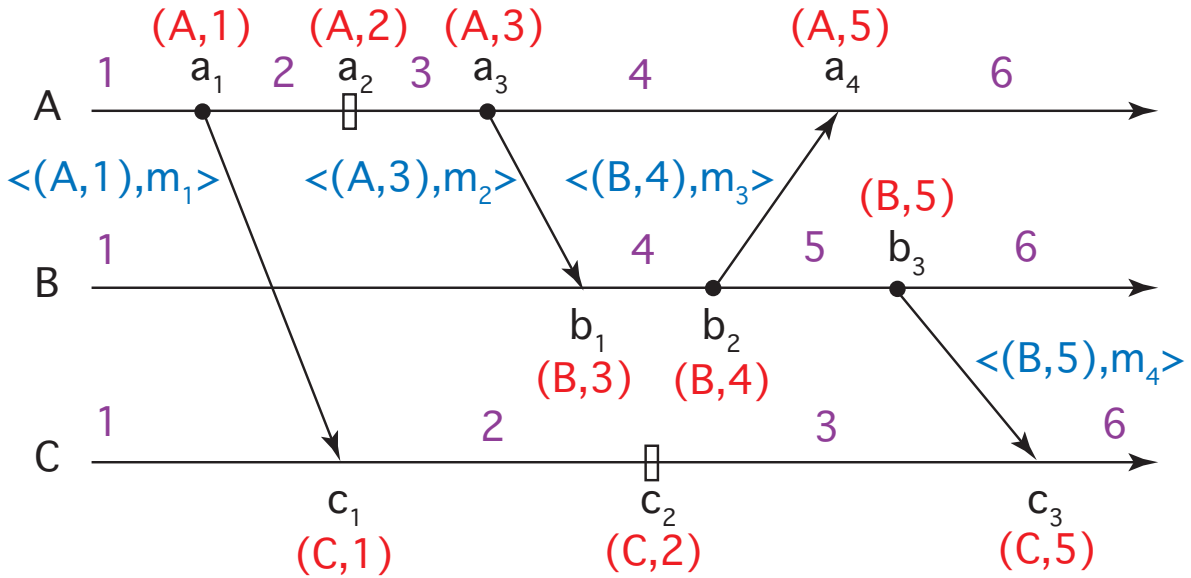
Propriétés

- Introduit un ordre total entre événements ;
  $\rightarrow$ Dates distinctes pour tout couple d'événements
- Date = (numéro de site, compteur local)
  $\Rightarrow$ ordre total sur les sites (par leur numéro)

```
public class Date {
   protected int s ; protected int cpt ;
   static boolean Prec ( Date d1, Date d2 ) {
    return (d1.cpt < d2.cpt)
         ||( (d1.cpt == d2.cpt) && (d1.s < d2.s)) ;
   }
}
```

Actions associées aux événements

Type d'événement sur un site s	Action sur le site s
Événement interne sur s	$H_s.Top()$
Émission sur s de m	$dm = H_s.Top()$; envoi de $< dm, m >$
Réception sur s de $< dm, m >$	$H_s.Recaler(dm)$



Horloge de Fidge-Mattern

Propriétés

- datation isomorphe à l'ordre causal ;
- utilisation de vecteurs de dimension égale au nombre de sites ;
- coût plus élevé : surcharge des messages par un vecteur.

Expression des relations entre dates

$$\forall D, D' \ : \ D \leq D' \ \equiv \ \forall i \ : \ D[i] \leq D'[i]$$

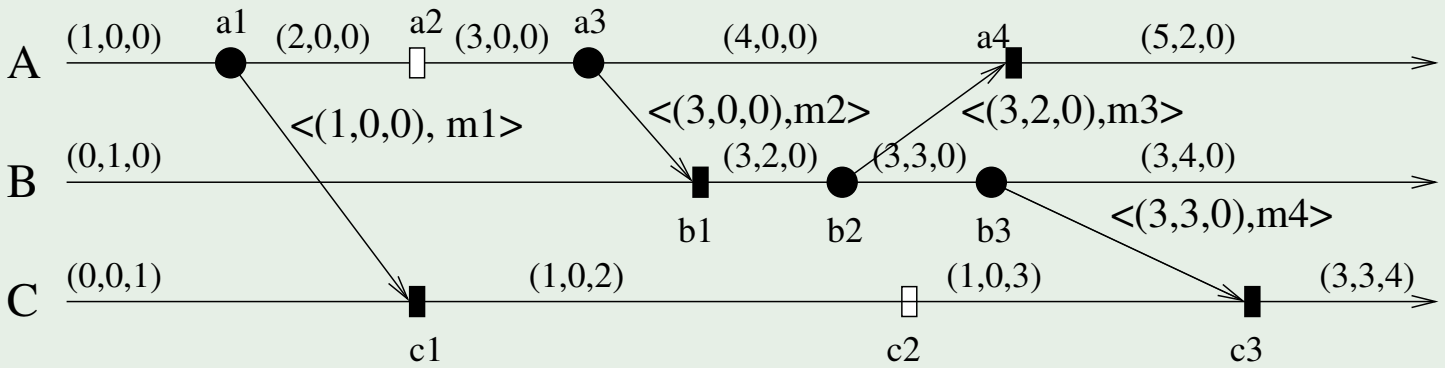
$$\forall D, D' \ : \ D < D' \ \equiv \ D \leq D' \wedge \exists k \ : \ D[k] < D'[k]$$

$$D \ || \ D' \ \equiv \ \neg(D < D') \wedge \neg(D' < D)$$

Actions associées aux événements

Type d'événement sur un site s	Action sur le site s
Événement interne sur s	$H_s.Top()$
Émission sur s de m	$dvm = H_s.Top()$; envoi de $< dvm, m >$
Réception sur s de $< dvm, m >$	$H_s.Recaler(dvm)$

Exemple



Plan

Définition d'un protocole (causalement) ordonné

Objectif

Garantir la cohérence des réceptions sur un même site par rapport à leur causalité éventuelle en émission.

$\Rightarrow$ réordonner les messages reçus sur un site

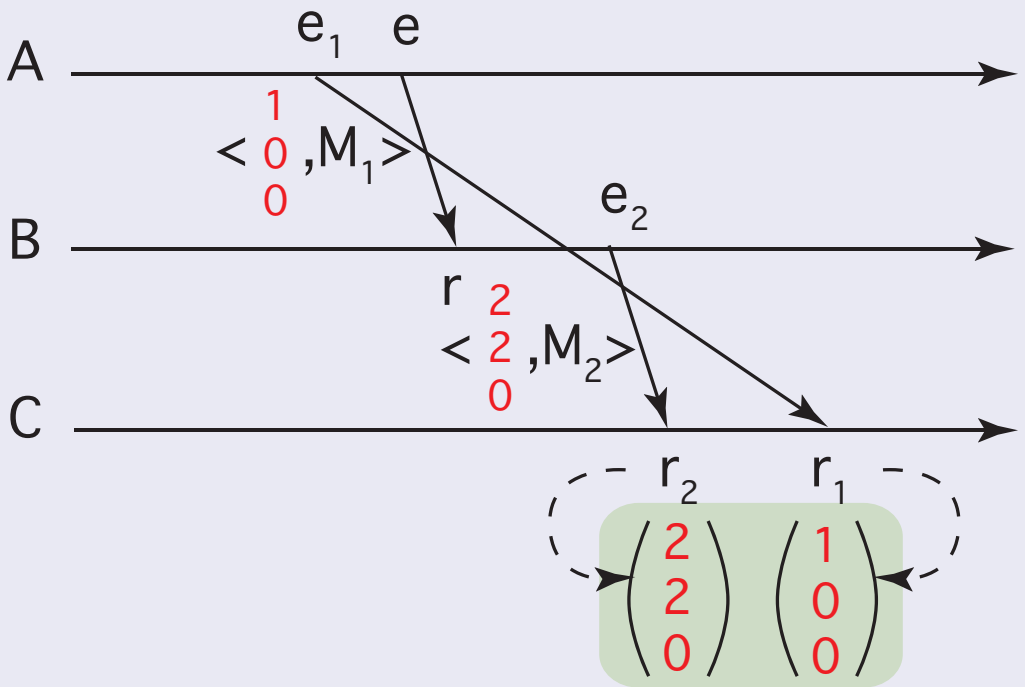
- Trois événements au lieu de deux par message :
  - l'émission e,
  - la réception r,
  - la délivrance d
  - Causalité : $e \prec r \prec d$
- S'exprime par la propriété :

$$\forall s \ : \ (\forall m, m' \ : \ r_s \wedge r'_s \ :: \ e \prec e' \ \Rightarrow d \prec d'))$$

Un essai, mais . . .

Datation causale

Permet la détection des anomalies en réception MAIS pas leur prévention.



Principes de l'algorithme

Notion d'histoire causale d'un message $H_c(m)$

L'histoire causale $H_c(m)$d'un message m est l'ensemble des messages qui ont leurs événements d'émission précédant causalement l'émission de m :

$$H_c(m) = \{m' : e' \prec e\}$$

Critère de délivrance d'un message

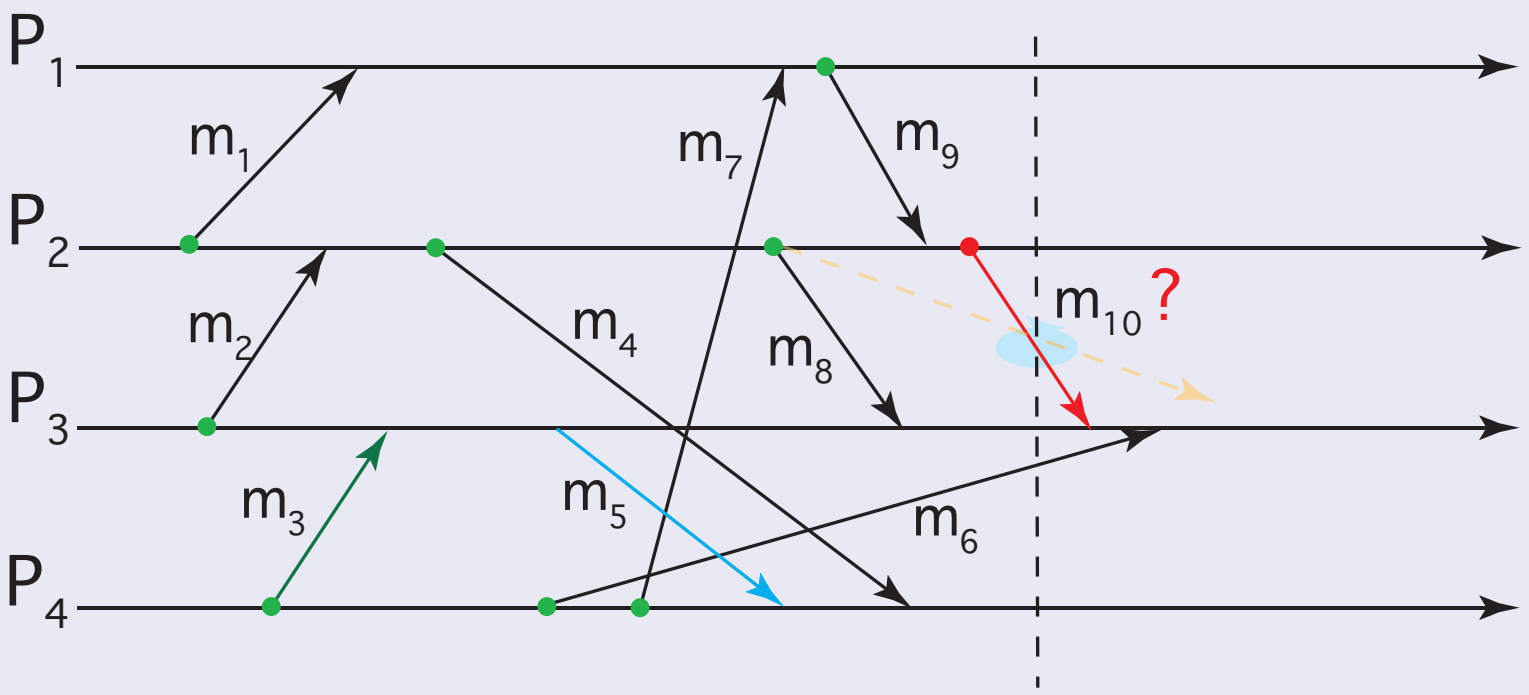
Un message m est délivré sur un site s ssi tous les messages de son histoire causale ayant aussi s comme site de destination ont été déjà délivrés :

$$\forall s : \forall m, m' \in H_c(m) : r_s \wedge r'_s \Rightarrow d_{s'} \prec d_s$$

Principes de l'algorithme

Exemple

.



Solution : Approche par compteurs (Michel Raynal)

Structures de données

☞ Représenter l'histoire causale de chaque message.

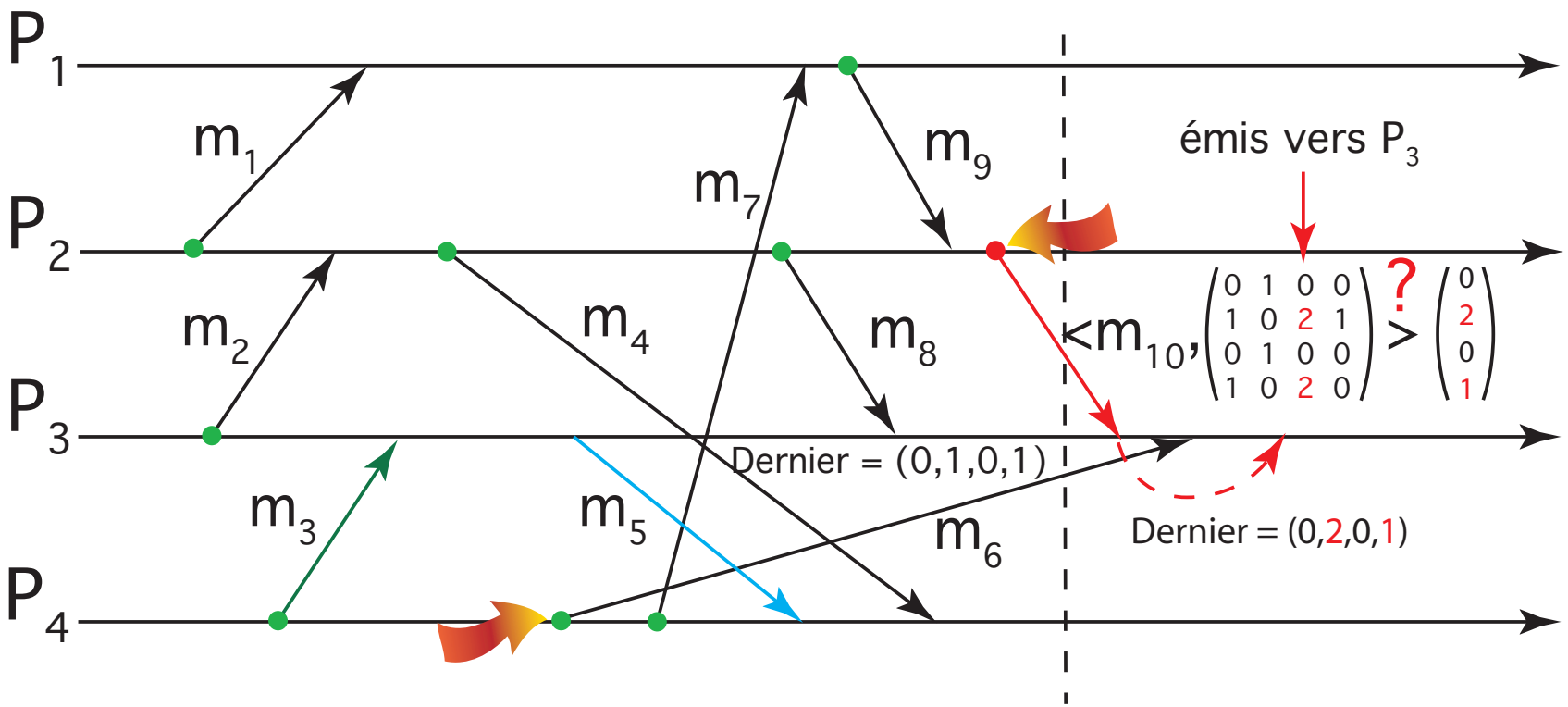
Chaque site gère :

- *MP* : une matrice de précédence causale
  $MP[i, j]$ = nombre de messages émis de S_i vers S_j
- *Dernier* : un vecteur de compteurs des messages reçus de chaque site
  $Dernier[i]$ = nombre de messages reçus du site S_i
- Tout message est surchargée par une copie de la matrice *MP* du site émetteur.

Actions associées aux événements

Type d'événement sur un site s	Action sur le site s
Émission sur s de M vers s'	$MP_s[s, s']++$; envoi de $< MP_s, M >$
Réception sur s' de $< MP, M >$ issu de s	$MP_{s'} = max(MP, MP_{s'})$ $Dernier[s]++$
Délivrance sur s' de $< MP, M >$ issu de s	Délivrable(M) $\equiv$ $Dernier[s] == MP[s, s']$ $\forall i \neq s : Dernier[i] \geq MP[i, s']$

Contrôle des délivrances

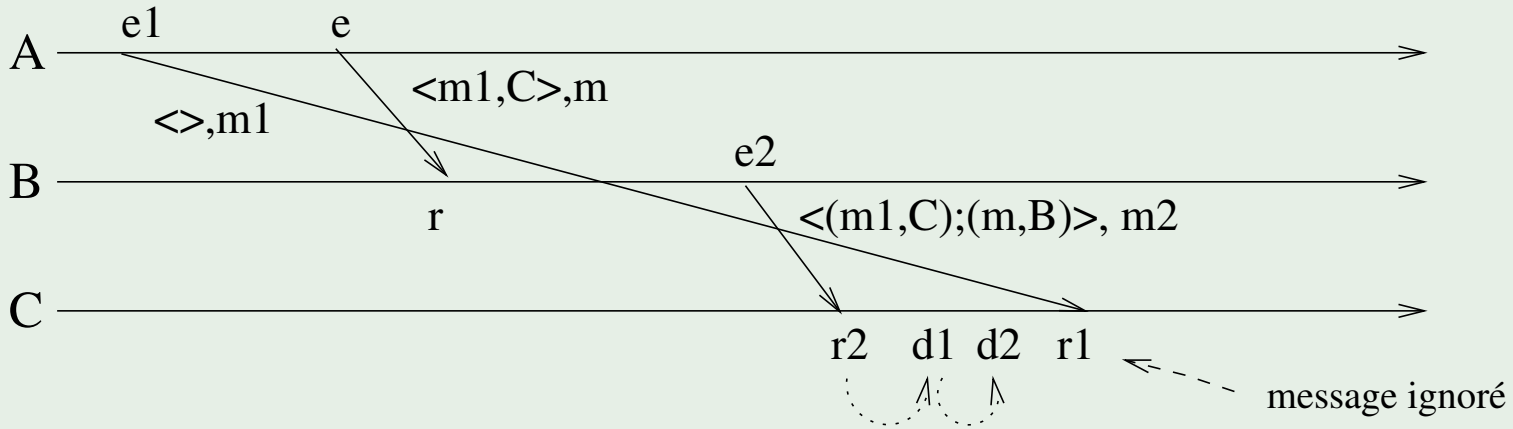


Approche par surcharge (piggybacking)

Principe

- Surcharger chaque message avec l'histoire des messages qui le précèdent causalement
- Dans le contexte du courrier électronique : Approche similaire du « réexpédier » avec copie de ce que l'on a reçu

Exemple



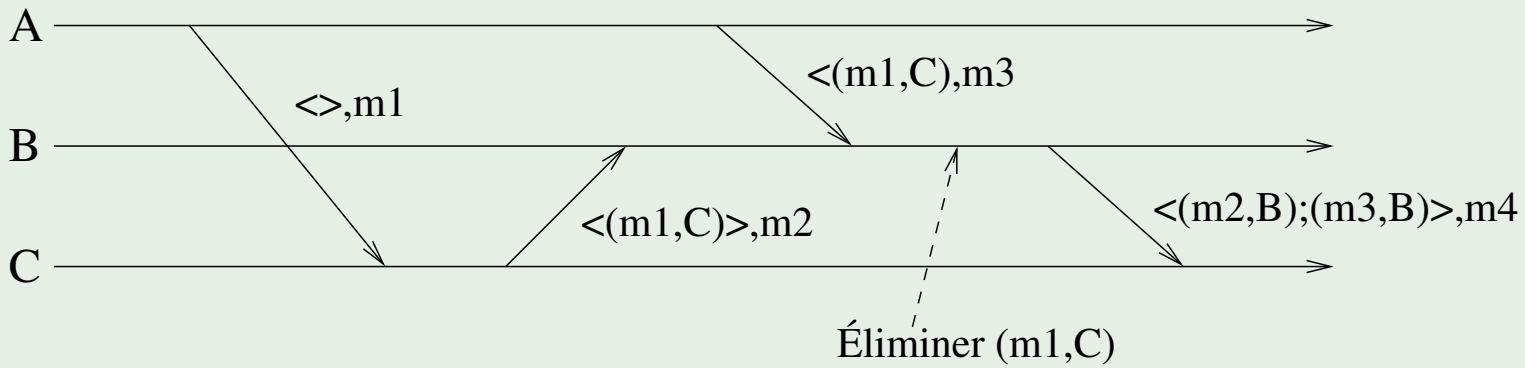
Approche par surcharge (piggybacking)

Mise en œuvre

- ☺ Simple et apport d'une certaine tolérance aux pertes de messages par redondance ;
- ☹ Messages de + en + longs

  ⇒ Quand, Comment réduire les histoires ?

Exemple

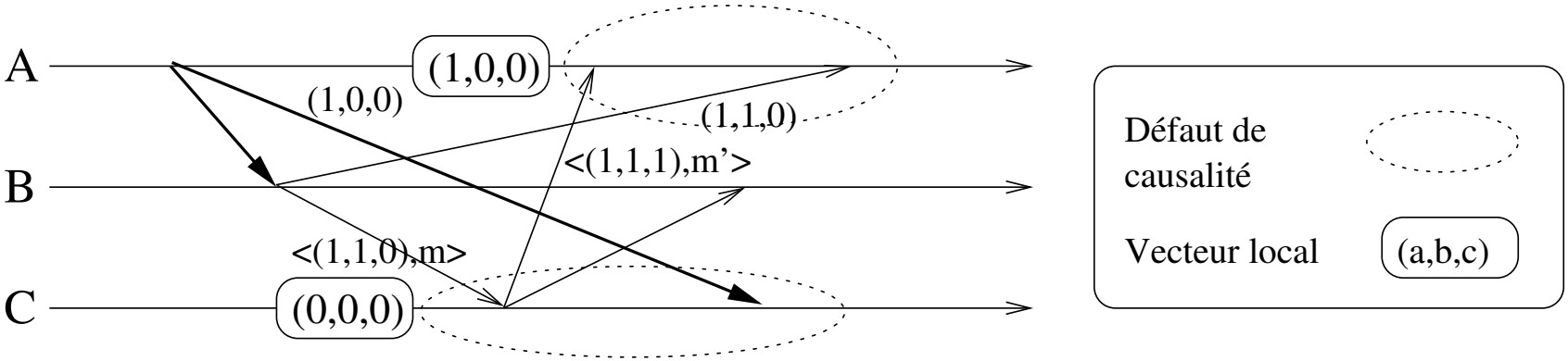


Une chance. . .

La diffusion ordonnée est plus simple !

Approche par matrice causale

- Il suffit de gérer un vecteur d'émission au lieu d'une matrice.
- Toutes les lignes sont identiques : un processus envoie le même nombre de messages à tous.



Plan

Tolérance aux fautes

Travaux de Kenneth P. Birman (Cornell University)

Objectif : modèle d'exécution tolérant aux fautes

Principes

- Groupes reconfigurables de processus ;
- Service assuré par un groupe de serveurs ;
- Diffusion inter et intra-groupe ;
- Pile de micro-protocoles ;
- Synchronisme virtuel ;
- Boîtes à outils : Isis, Horus, Ensemble ;
- Monde Java : Javagroups (http://www.jgroups.org)

La notion de groupe de processus

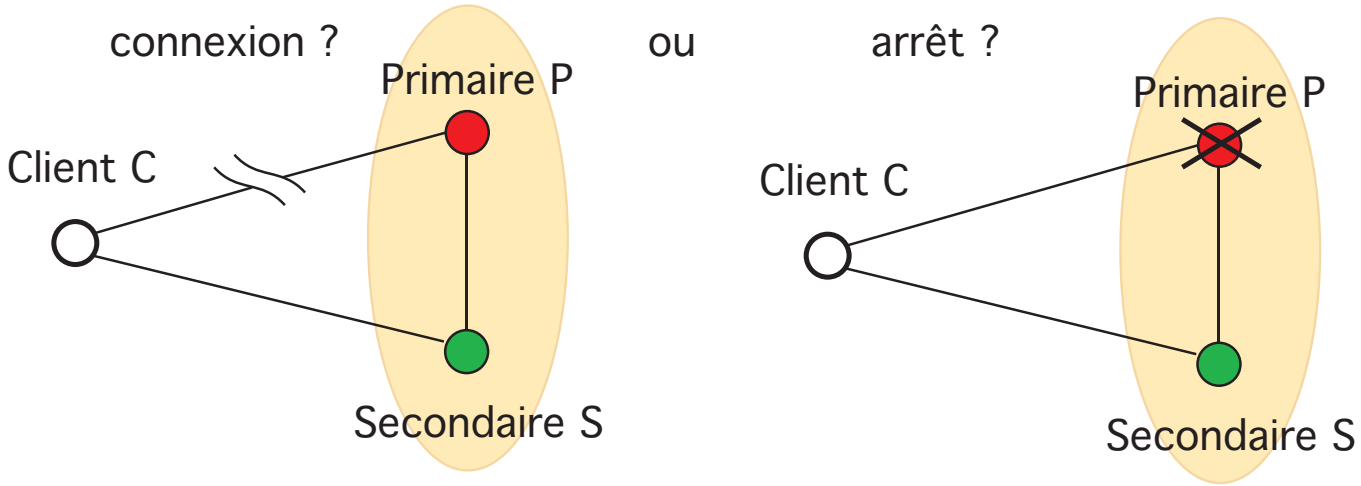
Diffusion inter-groupes

- Les clients ne connaissent que le nom du groupe ;
- Les membres ne communiquent pas ;
- Diffusion atomique et ordonnée des messages.

Diffusion intra-groupe

- Les membres du groupe coopèrent ;
- Liste des membres connue de chacun ;
- Cohérence : Même vision du groupe.

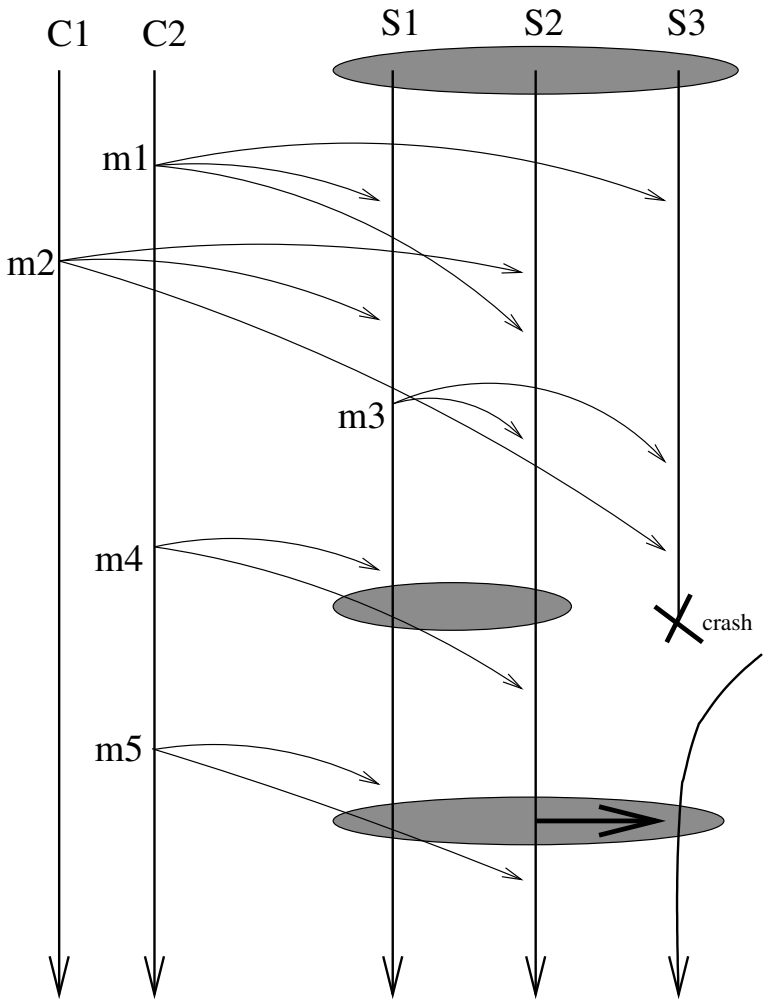
Problèmes clés



- Diffusion cohérente vers le groupe :
  - Ordonnancement des messages : fifo, causal, total ;
  - Traitement atomique des défaillances.
- Composition du groupe : ajout/retrait d'un membre, historique ;
- Synchronisation des traitements des membres.

Diffusion anarchique

- Ordre causal ?
- Ordre total ?
- Atomicité ?
- Arrêt/Reprise ?



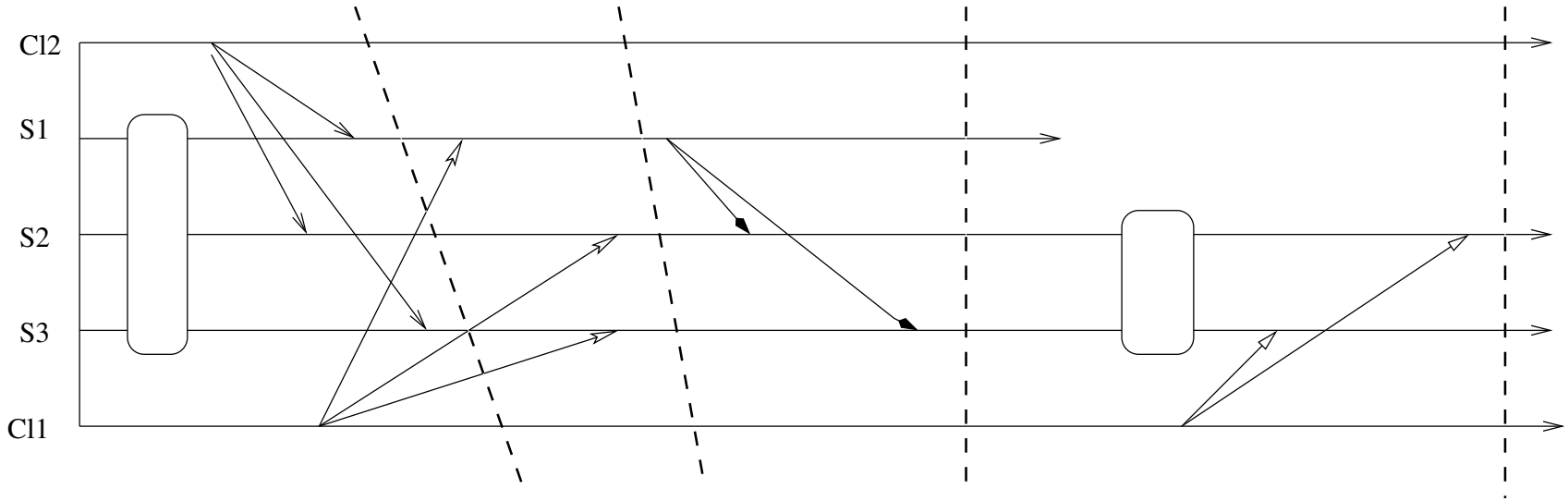
Synchronisme fort

Propriétés

- Communications fiables ;
- Diffusion atomique, i.e. de temps logique nul ;
- Atomicité du passage du groupe à la liste de membres ;
- Messages délivrés à tous les destinataires dans le même ordre ;
- Cet ordre est compatible avec la causalité entre messages ;
- Atomicité du transfert d'état lors de l'insertion d'un processus ;
- Fautes atomiques vis-à-vis des diffusions, et ordonnées identiquement sur tous les sites.

Synchronisme virtuel

Objectif : s'approcher du synchronisme fort

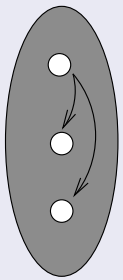


Fonctionnalités d'Isis

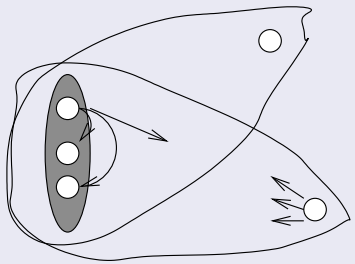
Gestion des groupes

- Créer/Détruire un groupe ;
- Se joindre à un groupe (en tant que membre) ;
- Quitter un groupe (dont on est membre) ;
- Se connecter à un groupe (en tant que client) ;
- Se déconnecter d'un groupe.

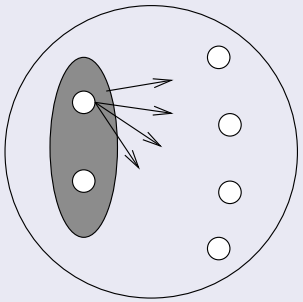
Différents types de groupe



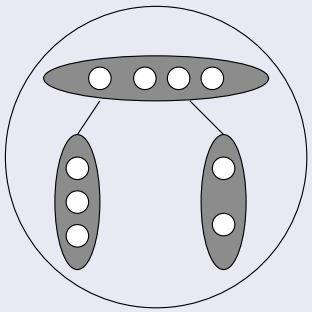
Groupe de pairs



Groupe client–serveur



groupe de diffusion



groupe hiérarchique

Fonctionnalités d'Isis

Types de diffusion

- `fbcast` : canal *fifo* fiable ;
- `cbcast` : diffusion (causalement) ordonnée ;
- `abcast` : diffusion totalement ordonnée atomique ;
- `gbcast` : diffusion utilisée pour la gestion des groupes.
  (Cohérence avec les diffusions ordonnées atomiques)

Mise en œuvre d'Isis

Structure d'une application (cliente et/ou serveur)

1. initialiser les librairies Isis et se connecter à Isis ;
2. déclarer les tâches et les points d'entrées ;
3. déclarer les gestionnaires de signaux et d'entrées-sorties ;
4. initialiser l'état de l'application ;
5. se joindre à des groupes et/ou devenir client ;
6. indiquer à Isis qu'elle est prête à recevoir les messages.

Mise en œuvre d'Isis

Comportement d'un client

```
void main() {
  /* connexion au service Isis */
  isis_init();
  serveur=pg_lookup("/services/exemple");
  pg_client(serveur);
  /* émission de requête */
  bcast(serveur,NumOp,<argument>,...,0);
}
```

Mise en œuvre d'Isis

Comportement d'un serveur (déclarations)

```
void Handler_1(message *m1) {
  /* accepter le message */
  msg_get(m1,"<format>",var1,...); ...
}
void Handler_p(message *mp) { ...};
void send_state() {
  /* envoi de l'état du serveur */
  xfer_out("<format>",x,...);
}
void rcv_state(message *m) {
  /* mise à jour de l'état du serveur */
}
```

Mise en œuvre d'Isis

Comportement d'un serveur (suite)

```
void main() {
  isis_init(0); /* connexion au service */
  /* connexion des handlers */
  isis_entry(OP1,Handler_1,"OP1");
  ...
  /* insertion du serveur dans son groupe */
  pg_join("/services/exemple",PG_XFER,
                send_state, rcv_state,0);
  isis_mainloop(0); /* attente d'un appel */
}
```