UNIVERSITE DE BEJAIA
FACULTE DES SCIENCES EXACTES
DEPARTEMENT D'INFORMATIQUE
1ème année MASTER

Année universitaire 2012/2013

MODULE DE SYSTEMES DISTRIBUES

Plan

Mme Z.TAHAKOURT
Maitre assistante
Université de Béjaia

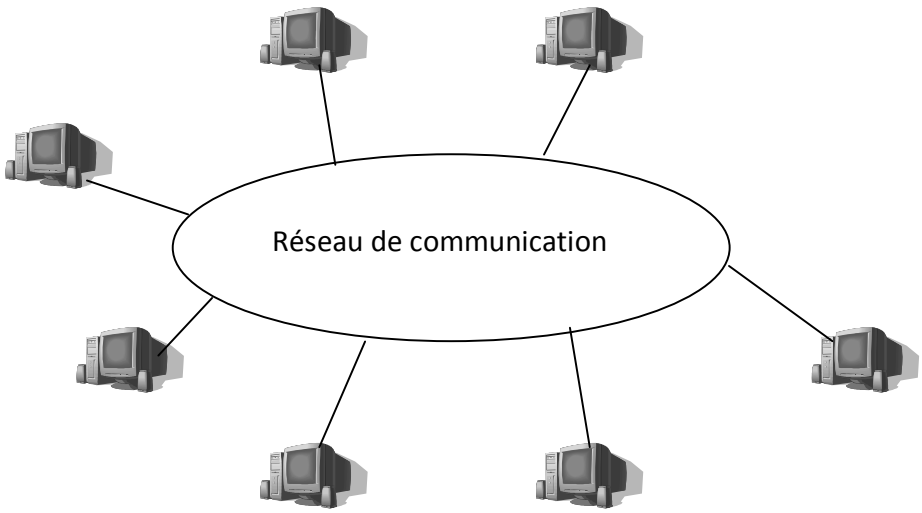
Chapitre I : INTRODUCTION AUX SYSTEMES DISTRIBUES

Plan du Chapitre

1. **Introduction**
2. **Système distribué Versus système centralisé**
3. **Caractéristiques des Systèmes distribués**
4. **Motivations des Systèmes distribués**
5. **La sécurité des Systèmes distribués**
6. **Avantages et inconvénients des Systèmes distribués**
7. **L'algorithmique répartie**
8. **Les problèmes de la distribution**

1. Introduction

Un système informatique distribué (réparti)= ensemble d'ordinateurs connectés par un réseau de communication.



Exemples de systèmes distribués (SD) :

- Serveur de fichiers : les fichiers se trouvent physiquement sur un serveur, mais virtuellement accessibles à partir de n'importe quelle machine, seulement si le réseau ou le serveur se plante plus d'accès aux fichiers.
- Web : un serveur web auquel se connectent des navigateurs web (clients).
- SD pour calcul scientifiques : plusieurs machines hétérogènes connectées par un réseau local ou internet (grille de calcul).le principe est le suivant : un ou plusieurs serveur distribue des calculs aux machines clientes, un client exécute son calcul puis renvoie les résultats au serveur. Avantage : utilisation maximum des ressources de calcul, seulement si le réseau ou le serveur plante le système s'arrête.

2. Système distribué Versus système centralisé

Système centralisé :

- Tout est localisé sur la même machine et accessible par programme
- Système logiciel s'exécutant sur une seule machine
- Accéder localement aux ressources nécessaires (données, code, périphériques, mémoire,…)

Système distribué :

- Ensemble d'ordinateurs standard indépendants connectés en réseau et communiquent via se réseau par envoi de messages
- Cet ensemble apparait du point de vu de l'utilisateur comme une unique entité

3. Caractéristiques des Systèmes distribués

Il n'existe pas de définition rigoureuse de ce qu'un système distribué. On peut essayer de dégager quelques propriétés qui caractérisent les systèmes qui sont universellement considérés comme distribués. Un système est considéré comme distribué s'il présente les caractéristiques suivantes :

- Le système est constitué d'un ensemble d'éléments de traitement (processeurs) reliés par des organes de communication qui leur permettent d'échanger de l'information.
- Un élément d'un SD peut tomber en panne sans affecter nécessairement le fonctionnement de l'ensemble.
- Absence de mémoire commune ; ce qui implique l'impossibilité de capter instantanément l'état global du système au moyen partagées.
- Absence d'horloge physique commune.
- Variabilité des délais de transmission de messages ce qui implique que deux éléments d'un SD peuvent avoir des informations différentes sur un troisième élément, et voir des événements s'y produire dans des ordres différents.

4. Motivations des Systèmes distribués

Les principales motivations (objectifs) derrière le développement d'un SD sont :

- Faire communiquer des applications existantes, tout en respectant l'autonomie locales de chacune d'elles ; la principale fonction du système est dans ce cas la communication d'information.
- Utiliser et partager des ressources existantes, comme par exemple :

1. un système de fichier afin d'utiliser des fichiers à partir de n'importe quelle machine.

2. Partage d'imprimante entre plusieurs machines.

3. Accès à une base de données partagée.

- Réduire le temps d'exécution d'une grosse application par l'exécution parallèle de plusieurs modules sur des processeurs différents (clustering), Comipar exemple :

    1. Le calcul scientifique distribué sur un ensemble de machine.

- Augmenter la disponibilité d'un système en assurant la redondance du matériel, des données ou des traitements, comme par exemple :

    1. existence de plusieurs imprimantes sur le réseau qui permet d'imprimer sur n'importe laquelle

    2. existence de plusieurs copies de données par réplication des données dans les bases de données distribuées.

- Garantir la transparence, c'est-à-dire donner l'illusion que l'ensemble des machines se comportent exactement comme un système monoprocesseur :

    1. Accès : les utilisateurs ne doivent pas pouvoir savoir où sont placées les ressources

    2. Concurrence : des accès simultanés aux ressources doivent pouvoir se faire sans interférence entre eux.

    3. Migration : les processus et données mobiles doivent être sans modification du nom et du chemin d'accès.

    4. Réplication : des utilisateurs accèdent à des copies multiples des données sans le savoir.

    5. Parallélisme : les processus devraient pouvoir s'exécuter en parallèle sans que l'utilisateur le sache.

5. La sécurité des Systèmes distribués

La nature des SD fait qu'ils sont souvent sujets à des attaques :

- Les communications à travers le réseau peuvent être interceptées.

- On ne connait pas toujours bien un élément distant avec qui on communique

Des solutions à ces problèmes :

- Connexion sécurisée par authentification avec les éléments distants.
- Cryptage des messages circulant sur le réseau.

6. Avantages et inconvénients des Systèmes distribués

6.1 Avantages

- Partage de données : de multiples utilisateurs peuvent accéder à une base de données partagées.
- Partage de périphériques : de multiples utilisateurs peuvent partager des ressources chères telles que des imprimantes en couleurs par exemple.
- Communication : facilite la communication interpersonnelle avec le courrier électronique par exemple.
- Rapidité d'exécution : un SD peut avoir une puissance de calcul global plus importante qu'un gros ordinateur.
- Fiabilité : le système peut continuer à fonctionner même si une machine tombe en panne.

6.2 Inconvénients

- Logiciels : peu de logiciels existants actuellement pour les SD.
- Réseau : le réseau peut être saturé ou provoquer d'autres problèmes.
- Sécurité : les données confidentielles peuvent être piratées.

7. L'algorithmique répartie

L'algorithmique répartie ou calcul réparti (distribué) est un calcul parallèle réalisé par un ensemble de processus qui utilisent plusieurs processeurs (un ou plusieurs processus par processeurs). La communication entre ces processus peut être réalisée par :

1. Mémoire partagée, ou
2. Echange de message.

Un calcul réparti se fait sur un système réparti.

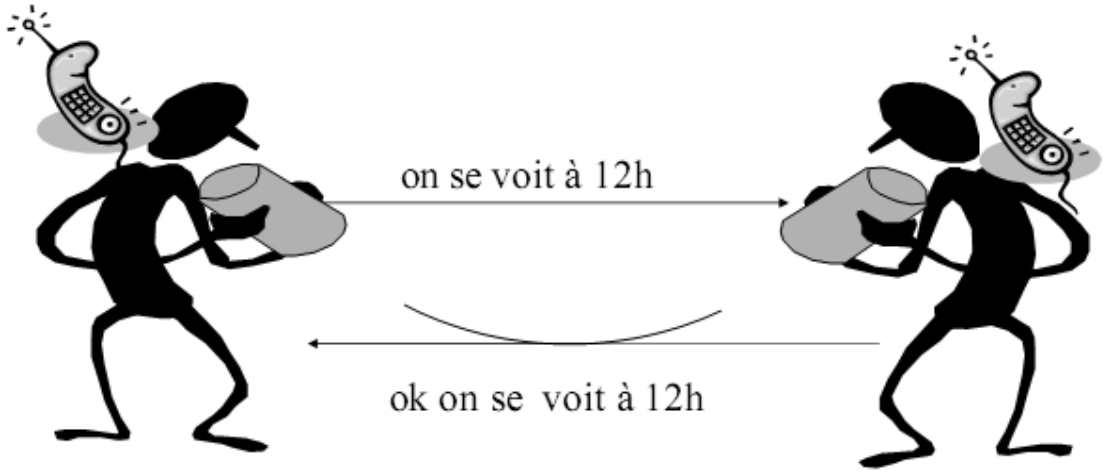
7.1 Modèles temporels d'un système distribué

Il existe deux modèles temporels de SD :

a) Synchrone : il existe des bornes sur les vitesses des processus, sur les délais de transmission et sur la dérive des horloges.
b) Asynchrone : aucune borne.

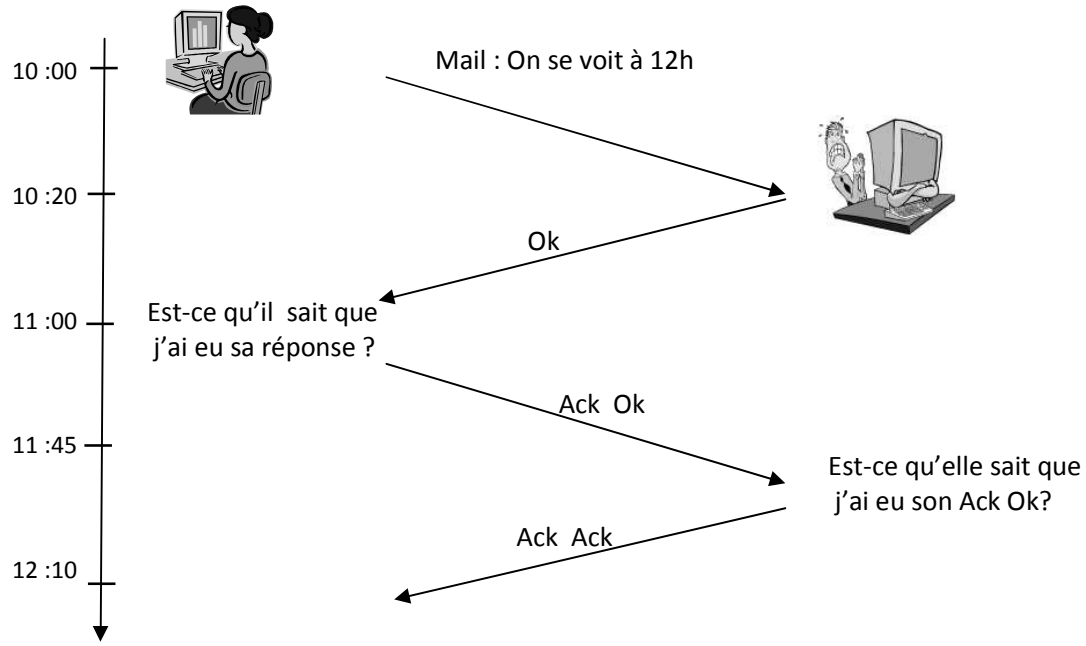
Modèles de communication : Synchrone /asynchrone

- Communication Synchrone



Même notion de temps, transmission instantanée, généralement bornée.

- Communication Asynchrone



Nota : Pour notre cours on s'intéressera au deuxième modèle, à savoir « asynchrone ».

7.2 Avantages et inconvénients d'un système distribué asynchrone

a) Avantages :

1. Meilleures tolérances aux défaillances → décentralisation des décisions)
2. Meilleure efficacité → autonomie relatives des processus

Ce qui implique un couplage faible, donc moins de risques de blocages mutuels.

c) **Inconvénients :** Contrôle des processus plus difficile, causée par :

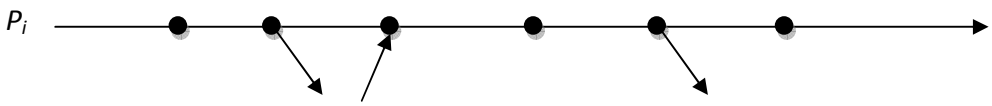
1. L'asynchronisme : évaluations instantanées impossibles (expressions réparties sur plusieurs processus)
2. Des défaillances : maintien de la cohérence du calcul difficile
3. Des deux (asynchronisme + défaillances) : comment distinguer entre un processus lent et processus crashé ou arrêté.

7.3 Calcul réparti asynchrone fiable

On considère dans ce qui suit le modèle du système suivant :

- Ensemble de processus p1, p2, …, pn.
- Séquences d'événements locaux par processus
  - Histoire de P_i : $[e_{i,1} ; e_{i,2} ; \dots ; e_{i,x} ; \dots[$
  - $h_{i,x}=[e_{i,1} ; e_{i,2} ; \dots ; e_{i,x}]$
  - $h_{i,x+1}= h_{i,x}.[e_{i,x+1}]$
- trois types d'événements :
  - interne (mise à jour de l'état local du processus)
  - Emission de message *m* sur un processus p_i vers un processus p_j.
    - Evénement *émission* de m vers p_j.
    - Action : *envoyer (m, p_j)*
    - Effet : *dépose* m dans le canal $c_{i,j}$
  - Réception de message *m* sur un processus P_j.
    - Evénement *réception* de m depuis p_i.
    - Action : *délivrer (m)*
    - Effet : *retire* m du canal $c_{i,j}$: met à jour les variables de P_j.

$e_{i,1}$ $e_{i,2}$ $e_{i,3}$ $e_{i,4}$ ... $e_{i,x-1}$ $e_{i,x}$



8. Les problèmes de la distribution

De ce qui précède on peut mettre en évidence les principaux problèmes du traitement distribué.

- Désignation des éléments ou des entités (machines, processus, fichiers,…) :

  C'est-à-dire la connaissance des éléments formant le système, en les identifiant en les localisant. Le système de désignation des entités est l'ensemble des conventions, procédures et structures de données qui permettent de donner un nom unique à chaque entité et d'utiliser ce nom pour y accéder.

- Synchronisation des activités et ordre des événements :

  Une propriété des SD est l'impossibilité de déterminer un état global instantané du système. Sur un site donné, les informations sur le reste du système ne peuvent être que partielles ou périmées. En dépit de ces incertitudes, il faut néanmoins pouvoir synchroniser des activités qui se déroulent sur des sites différents (Chapitre II).

- Gestion de la cohérence de l'information distribuée :

  Le problème de la gestion d'informations distribuées est lié au précédent. Il faut pouvoir assurer la cohérence mutuelle d'informations distribuées en l'absence d'état global. Ce problème se pose notamment pour la gestion cohérente de copies multiples d'une même information que des impératifs de disponibilité et d'efficacité amènent à conserver (Chapitre III).

- Tolérance aux pannes :

  L'objectif visé est d'assurer le fonctionnement d'un système éventuellement dans un mode dégradé, en présence de panne de ses éléments (processeurs, mémoires, système de communication) (Chapitre IV).

Chapitre II : ORDONNANCEMENT DES EVENEMENTS DANS UN SYSTEME DISTRIBUE

Plan du Chapitre*ODU*

1. Introduction

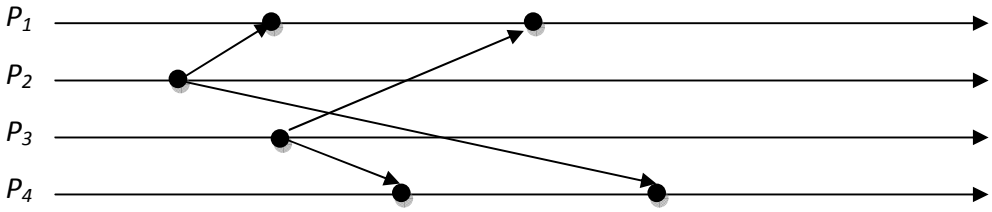
Un système distribué est composé de n sites (processus) interconnectés par les canaux de communication.

On appelle *événement* un changement local d'état sur un site ou bien l'envoi ou la réception de messages.

Les messages échangés dans un SD sont essentiellement destinés à la coordination entre les tâches, c-à-d à leur synchronisation. Le problème majeur de cette synchronisation est que les transmissions ne sont pas immédiates ; les délais de transmission sont en général beaucoup plus lents que le temps séparant deux instructions d'un même processus. Les conséquences sont :

1. A un instant donné, on ne peut connaitre à partir d'un site donné, qu'une approximation de l'état de l'autre site.

2. Deux événements observables quelconques, d'un système peuvent être perçus dans un ordre différent d'un site à l'autre, comme l'illustre le schéma ci-dessous.



3. Il faut souvent que les différents sites puissent s'accorder sur l'ordre chronologique. Or dans un SD, on ne peut pas utiliser une heure unique, car rien ne permet d'assurer que les horloges sont synchrones.

2. Précédence causale

L'ordonnancement des événements dans un SD repose sur la définition d'une relation globale de précédence. Soit a et b deux événements. On dit que *a précède directement b* si et seulement si l'une des conditions suivantes est vrai :

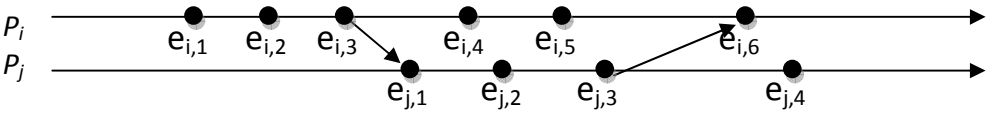
1. *a* et *b* arrivent sur le même site, et *a* est antérieur à *b* sur ce site ;
2. *a* est l'envoi d'un message *m* depuis un site, et *b* la réception de ce message *m* sur un autre site.

La relation *précède* (également notée →) est la fermeture transitive de la relation précède directement. Cette relation est un *ordre partiel*. Cette définition traduit le principe de causalité, appliqué globalement au système : la seule interaction possible entre deux sites est l'échange de messages. Les liens de causalité entre événements se produisent sur des sites différents utilisent nécessairement l'échange de messages entre ces sites. C'est pourquoi la relation → est aussi appelée relation de précédence causale (il s'agit plus exactement d'une dépendance potentielle : pour que *a* puisse être cause de b, il est nécessaire que *a*→*b*).

Deux événements a et b sont dits concurrents (ou causalement indépendants) si aucun des deux ne précède causalement l'autre, et ne peut donc influer sur lui. On écrit :

$a \,//\, b \Leftrightarrow \neg(a \rightarrow b)$ et $\neg(b \rightarrow a)$

Exemple1



On a : $e_{i,1} \rightarrow e_{i,2} \rightarrow e_{i,3} \rightarrow e_{i,4} \rightarrow e_{i,5} \rightarrow e_{i,6}$

$e_{j,1} \rightarrow e_{j,2} \rightarrow e_{j,3} \rightarrow e_{j,4}$

$e_{i,2} \rightarrow e_{i,3} \rightarrow e_{j,1} \rightarrow e_{j,2} \rightarrow e_{j,3} \rightarrow e_{i,6}$

Ces séquences de dépendance causales sont des relation d'ordre partielles, car elles ne peuvent pas comprendre tous les événements du système.

2.1 Histoire d'un événement

Il est appelé aussi *cône de causalité*, correspond à l'ensemble des événements qui le précédent causalement, y compris lui-même.

Exemple précédent :

histoire ($e_{j,3}$)=cône de causalité($e_{j,3}$)=$h_{j,3}$=[$e_{i,1}$; $e_{i,2}$;$e_{i,3}$; $e_{j,1}$; $e_{j,2}$; $e_{j,3}$]

2.2 Propriétés

Soient a et b deux événements, alors :

- $a \rightarrow b \Leftrightarrow a \in$ histoire(b)

- a // b $\Leftrightarrow$ a $\notin$ histoire(b) et b $\notin$ histoire(a)

Objectif : trouver des mécanismes qui permettent d'associer des dates aux événements concernés tel que : si a → b alors la date associée à *b* doit être relativement à un temps logique global après la date associée à *a*.

3. Les horloges logiques

3.1 Horloges linéaires de Lamport

Une première méthode pour ordonner les événements dans un SD repose sur l'emploi d'horloges logiques linéaire et d'estampilles, appelées horloges de *Lamport* (1978). Ce mécanisme réalise une datation des événements qui a les propriétés suivantes :

- l'ordre des dates est compatible avec la relation de précédence causale,
- la datation d'un événement sur un site ne met en œuvre que les informations locales au site.

On définit sur chaque site S_i un compteur H_i à valeur entières initialisées à 0, appelé horloge logique, qui sert à dater les événements sur ce site. Lorsqu'un événement se produit sur un site S_i, la valeur de H_i est incrémentée de 1 et la date de l'événement *a*, notée $H_i(a)$ est par définition la nouvelle valeur de H_i. Pour garantir le respect de la précédence causale, tout message *m* émis par un site S_i porte une estampille $E(m)$ qui est sa date d'émission. Un site S_j qui reçoit un message *m* exécute l'instruction suivante :

$$H_j := Max[\, H_j, E(m) \,] + 1$$

Rappel de mathématiques :

C) Ordre total, ordre partiel

> Soit R une relation d'ordre sur E. On dit que R définit un ordre total sur E lorsque deux éléments de E sont toujours comparables pour R, c'est-à-dire : $\forall x \in E, \forall y \in E, (xRy \text{ ou } yRx)$.
>
> Dans le cas contraire, on parle d'ordre partiel.

La relation ainsi définie, n'est pas un ordre total : en effet des événements causalement indépendants, arrivant sur des sites différents, peuvent avoir la même date. Pour obtenir un ordre total, il suffit de définir un ordre arbitraire entre les sites. Si *a* et *b* sont des événements arrivant respectivement sur les sites S_i et S_j, on peut définir comme suit une relation d'ordre total, notée $\Rightarrow$:

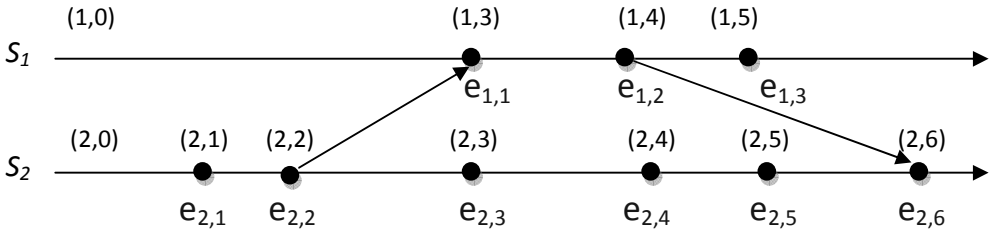
$$(a \Rightarrow b) \Leftrightarrow H_i(a) < H_j(b) \text{ ou } (H_i(a) = H_j(b) \text{ et } i<j).$$

Autrement dit, si

Un événement est maintenant daté par le couple (Numéro de site, estampille).

Localement (si i = j), H_i donne l'ordre des événements du processus. Les 2 horloges de 2 processus différents permettent de déterminer l'ordonnancement des événements des 2 processus. Si égalité de la valeur de l'horloge, le numéro du processus est utilisé pour les ordonner.

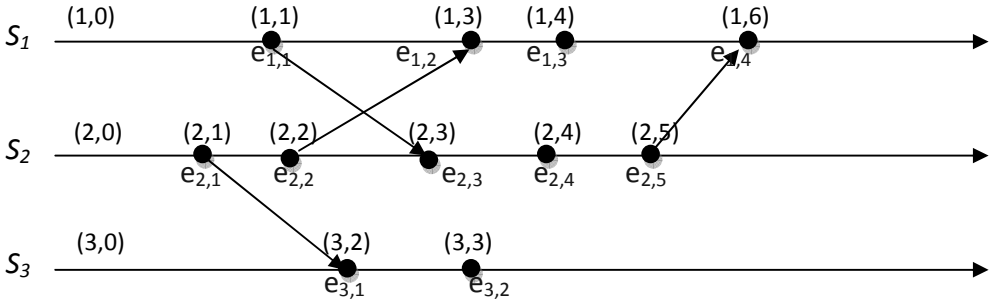
Exemple1



Sur cet exemple l'ordre total obtenu est:

$e_{2,1} \Rightarrow e_{2,2} \Rightarrow e_{1,1} \Rightarrow e_{2,3} \Rightarrow e_{1,2} \Rightarrow e_{2,4} \Rightarrow e_{1,3} \Rightarrow e_{2,5} \Rightarrow e_{2,6}$

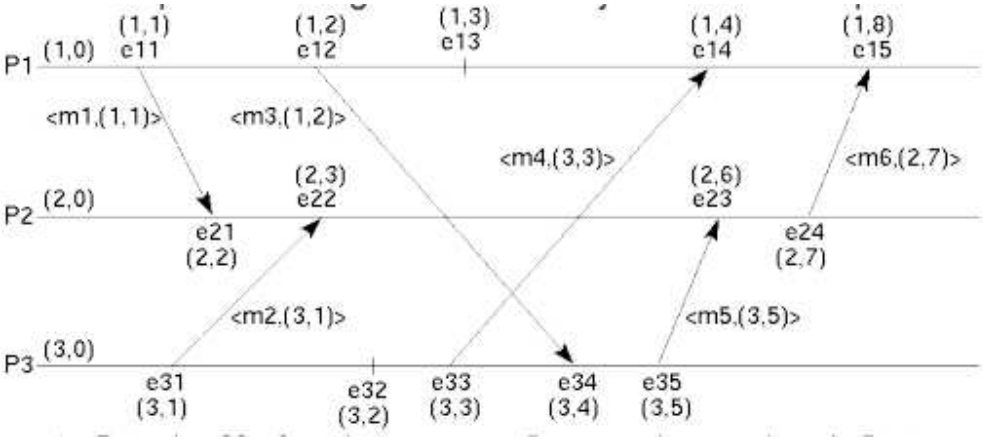
Exemple2



Question: ordonner tous les événements de ce système.

$e_{1,1} \Rightarrow e_{2,1} \Rightarrow e_{2,2} \Rightarrow e_{3,1} \Rightarrow e_{1,2} \Rightarrow e_{2,3} \Rightarrow e_{3,2} \Rightarrow e_{1,3} \Rightarrow e_{2,4} \Rightarrow e_{2,5} \Rightarrow e_{1,4}$

Exemple3



Question: ordonner tous les événements de ce système.

$e_{1,1} \Rightarrow e_{3,1} \Rightarrow e_{1,2} \Rightarrow e_{2,1} \Rightarrow e_{3,2} \Rightarrow e_{1,3} \Rightarrow e_{2,2} \Rightarrow e_{3,3} \Rightarrow e_{1,4} \Rightarrow e_{3,4} \Rightarrow e_{3,5} \Rightarrow e_{2,3} \Rightarrow e_{2,4} \Rightarrow e_{1,5}$

Propriétés :

1. si un événement est daté par h, alors h-1 représente le nombre d'événements qui le précédent.
2. Si $a \rightarrow b \Rightarrow H(a)<H(b)$, mais si $H(a)< h(b) \neg\Rightarrow a \rightarrow b$

L'ordonnancement des événements par estampilles est une technique d'usage universel dans les SD. Les principales classes d'utilisation de ce mécanisme sont :

- les algorithmes qui mettent en œuvre une file d'attente virtuelle répartie. Une telle file d'attente possède une représentation partielle ou totale sur chaque site concerné par l'algorithme, et l'ordre global des éléments de la file est défini par la relation $\Rightarrow$. Ces algorithmes comprennent notamment l'exclusion mutuelle (qu'on verra à la fin du chapitre), la mise à jour cohérente de copies multiples et la diffusion cohérentes.
- La détermination de l'événement le plus récent dans un ensemble d'événements.
- La génération de noms uniques, que ce soit pour la construction de noms internes globaux ou pour des raisons de protection

3.2 Horloges vectorielles de Mattern

La relation d'ordre $\Rightarrow$ définie par les estampilles ne suffit pas pour établir une relation de causalité entre deux événements. La relation $a \Rightarrow b$ implique uniquement que b ne précède pas causalement a ; il est possible que a précède causalement b ou bien que a et b sont indépendants. L'ordre total introduit par les estampilles « efface » artificiellement la précédence causale.

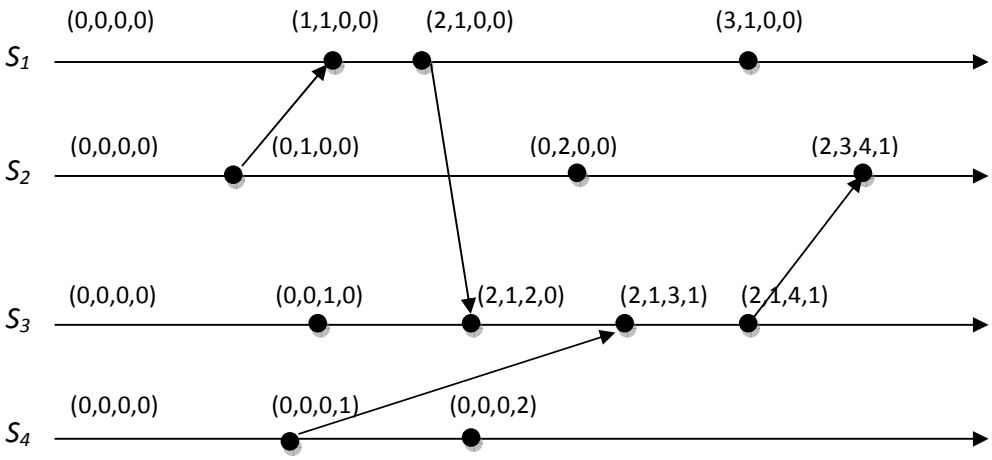
Il est parfois utiles de pouvoir déterminer s'il ya ou non dépendance causale entre deux événements, par exemple pour la recherche des causes potentielles d'une erreur.

Le mécanisme des *horloges vectorielles* a été introduit, par *Mattern*, pour caractériser la dépendance causale.

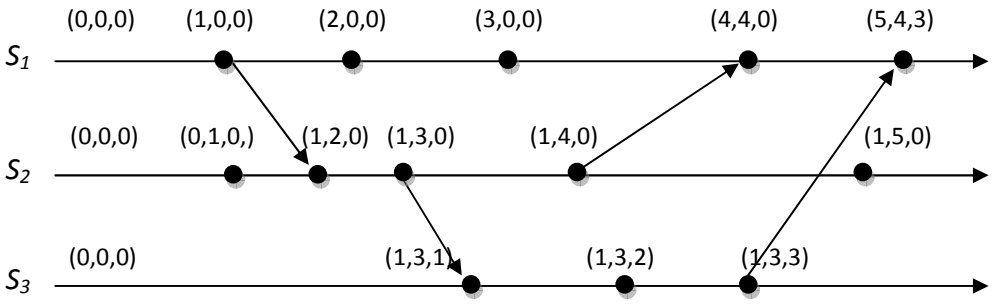
Soit n le nombre de sites. Sur chaque site S_i, i =1,…, n.
On définit une horloge vectorielle comme un vecteur V_i [1..n] initialisé à 0.
Lorsqu'un événement se produit sur le site S_i , on exécute V_i [i] := V_i [i]+1
Chaque message m porte comme estampille V_m l'horloge vectorielle V_i du site émetteur.
A la réception d'un message (m, V_m) le site récepteur, soit S_i , exécute :

V_i [j] :=Max [V_i [j], V_m [j]] pour j=1…, n.

Exemple1



Exemple2



Appelons passé d'un événement *a* l'ensemble constitué de *a* lui-même et des événements qui précédent causalement *a*. Si un événement *a* est daté par le vecteur V_a alors : $V_a[j]$ = nombre d'événements du passé de a sur le site S_j.

Définissons une relation d'ordre partiel entre horloges vectorielles :

- $\forall$ V,W : V $\leq$ W $\Leftrightarrow \forall$j : V[i] $\leq$ W[i]
- $\forall$ V,W : V < W $\Leftrightarrow$ (V $\leq$ W) ET ($\exists$ j : V[j] < W[j])
- V // W $\Leftrightarrow \neg$ (V < W) et $\neg$ (W < V)

La propriété essentielle des horloges vectorielles s'exprime comme suit :
Soit deux événements a et b datés respectivement par les vecteurs V_a et V_b. Alors :
$a \rightarrow b \Leftrightarrow V_a < V_b$
a et *b* causalement indépendant $\Leftrightarrow V_a // V_b$

3.3 Les Horloges matricielles

Objectif des HM :

- Obtenir, pour un site, des informations sur les horloges, sur les messages échangés et sur les événements internes des autres sites entre eux.
- Connaître, pour un site :
  1. tous les événements internes et externes (donc, les horloges) que les sites connaissent les uns des autres.
  2. les messages émis entre tous les sites, avec vérification de l'ordre causal.
  3. les messages effectivement délivrés.

3.3.1 Délivrance de message

La délivrance d'un message est l'opération consistant à rendre accessible le message aux applications cliente.

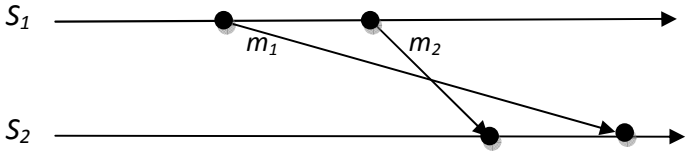
3.3.2 Propriétés

Deux propriétés à respecter :

- ✓ **Ordre de délivrance FIFO: sur le même site**

  Si deux messages sont envoyés successivement depuis un même site S_i vers un même destinataire S_j, le premier sera délivré sur le site S_j avant le second, c'est-à-dire que,
  Send (m_1,j) → *send*(m_2,j) *alors: receive*(m_1) → *receive*(m_2)

Exemple1



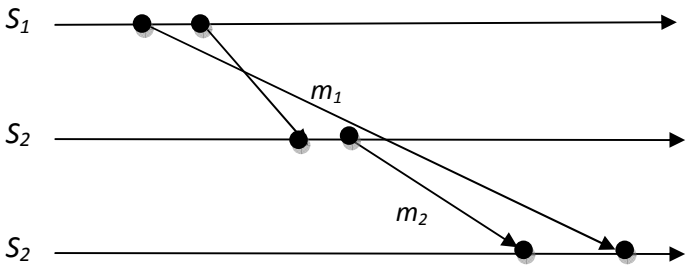
Ici la propriété de l'ordre de délivrance FIFO n'est pas respectée.

- ✓ **Ordre de délivrance causale: extension à plusieurs sites**

  Un message ne peut être délivré que si tous les messages qui lui sont causalement antérieurs ont été délivrés.

  Si l'envoi du message m_1 par le site S_i à destination du site S_k précède causalement l'envoi du message m_2 par le site S_j à destination du site S_k alors, le message m_1 sera délivré avant le message m_2 sur le site S_k.
  Send (m_1,k) → *send*(m_2,k) *alors: receive*(m_1) → *receive*(m_2)

Exemple2



Là aussi la propriété de l'ordre de délivrance causale n'est pas respectée.

3.3.3 Principe des horloges matricielles

Cet autre mécanisme de datation des événements va permettre de détecter l'arrivée de messages dont la délivrance violerait le respect de la causalité et permettra donc d'en différer la délivrance.

Avec ce système de datation, dans un système de *n* sites, les horloges d'un site S_i et les estampilles des événements (et des messages échangés entre les sites) sont des matrices carrées d'ordre *n*.

Signification de la matrice :

Nous noterons :

1. HM_i *l'horloge matricielle du site Si,*
2. EM_m *l'estampille matricielle du message m et,*
3. EM_e *l'estampille matricielle d'un événement e.*
4. $HM_i[j,j]$ est le nombre d'événements locaux de S_j (pour $j=i$: événements locaux au site S_i)
5. La ligne *i* ($\forall$ *k* $HM_i[i,k]$) est le nombre de messages émis par Si vers les autres sites S_k.
6. Les autres lignes j ($\forall$ *k* $HM_i[j,k]$) sont les nombres de messages que le site S_i sait que les sites S_j ont émis vers les sites S_k.

En d'autres mots : sur le site S_i, la matrice HM_i va mémoriser:

- le nombre de messages que le site S_i a envoyés aux différents autres sites : cette information est fournie par la $i^{ème}$ ligne de la matrice. L'élément diagonal compte les événements sur le site S_i;
- pour chacun des autres sites S_j, le nombre de messages émis par ce site dont le site S_i a connaissance (c'est-à-dire dont le site S_i sait qu'ils ont été envoyés). Ainsi sur le site S_i, la valeur $HM_i[j,k]$ donne le nombre de messages en provenance du site S_j délivrables sur le site S_k dont le site S_i a connaissance (c'est-à-dire dont l'envoi est causalement antérieur à tout ce qui arrivera dorénavant sur S_i).

Un message envoyé par un site sera estampillé en utilisant l'horloge matricielle du site émetteur. La modification et la synchronisation des horloges matricielles des différents sites sont réalisées de la manière suivante :

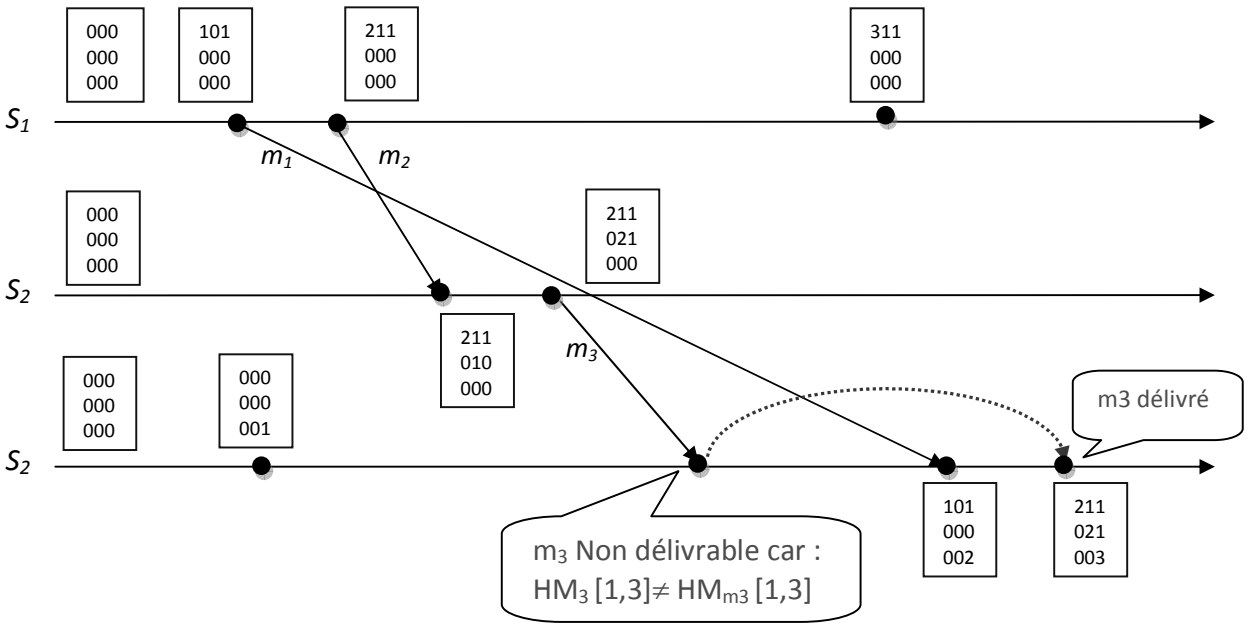
- Evénement interne se produit sur le site S_i, $HM_i[i,i]$ ++;
- Envoi d'un message *m* par le site S_i vers le site S_j :
  - $HM_i[i,i]$ ++ ;
  - $HM_i[i,j]$ ++ ;
  - $EM_m := HM_i$.
- Réception d'un message *m* (EM_m) sur le site S_i depuis le site S_j :

  On doit d'abord contrôler le respect de la délivrance FIFO et la délivrance causale, c'est-à-dire vérifier que :
  1. $HM_m[j,i] = HM_i[j,i]+1$ (ordre FIFO sur le canal (j,i))
  2. pour tout $k \neq i$ et j, $HM_m[k,i] \leq HM_i[k,i]$ (tous les messages des sites différents de S_j ont été reçus)

  On peut alors délivrer le message et mettre à jour les horloges comme suit :
  - $HM_i[i,i]$ ++ ;
  - $HM_i[j,i]$ ++ ;
  - $\forall k \neq i$ et j , et pour tout $l \neq i$: $HM_i[k,l] := \text{Max}[HM_i[k,l], EM_m[k,l]]$;

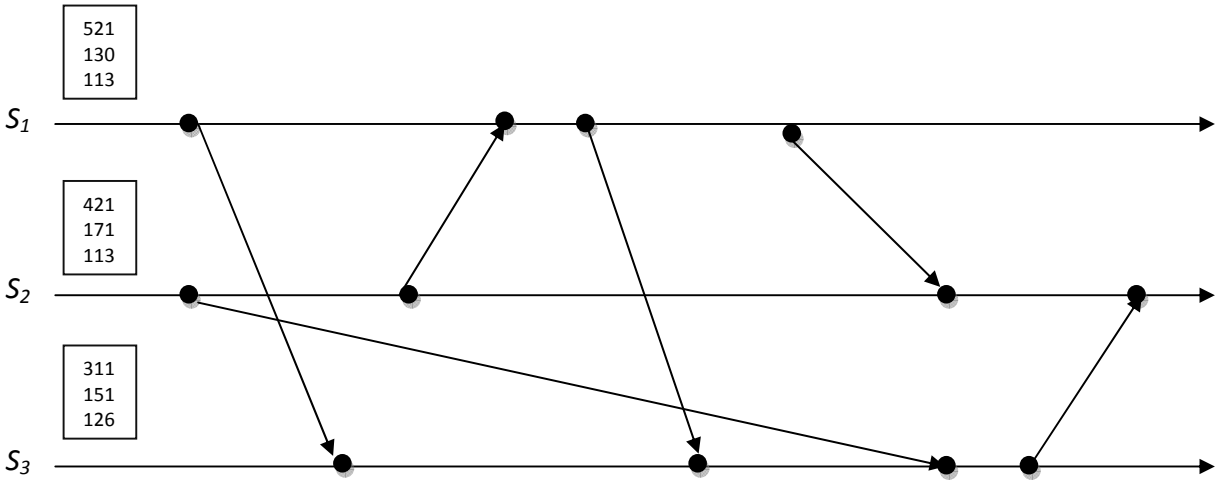
Exemple1

Considérons le calcul réparti suivant, Dans le cas du non respect de la causalité on retarde la délivrance.



Exemple2

Compléter le chronogramme suivant en datant les événements à l'aide des horloges matricielles. Dans le cas où la causalité n'est pas respectée, retarder la délivrance.



4. Exclusion mutuelle

Il s'agit ici d'étudier le contrôle de n processus entrant en compétition pour se partager une même ressource ; D'un point de vue informatique, ce problème est fréquent. Citons-en quelques exemples:

- suite d'opérations dans un fichier ou une base de données;
- accès à une ressource telle qu'une imprimante;

la section critique peut être utilisée par un seul processus ; quand un processus l'utilise, on dit qu'il est en *section critique*. On suppose que l'utilisation de la ressource toujours une durée finie. L'utilisation de la ressource par un processus suivra toujours le plan suivant

- protocole d'acquisition de la ressource (prologue)
- utilisation de la ressource en section critique
- protocole de libération de la ressource (épilogue)

Un processus P' entamant son processus d'acquisition alors que la section critique est utilisée par un processus P devra attendre (rester dans la phase d'acquisition) au moins jusqu'à ce que P ait terminé son protocole de libération.

Voyons les embûches à éviter lorsqu'on met au point des protocoles d'exclusion mutuelle.

Imaginons ces exemples de protocoles d'acquisition :

1. si un processus découvre qu'il est en compétition avec d'autres pour l'acquisition, il laisse passer les autres : c'est l'exemple le plus simpliste qu'on puisse imaginer de ce qui conduit à une situation dite d'*interblocage* ; tous les processus entrant en compétition sont bloqués et la ressource reste éternellement non utilisée. *L'interblocage* dans le cas de l'exclusion mutuelle peut s'énoncer ainsi : interblocage = il existe au moins un processus qui désire accéder à la section critique et aucun processus ne peut l'atteindre.
2. Considérons maintenant un système dans lequel opèrent trois processus P1, P2, P3 qui utilisent une même ressource R. Si le protocole est tel que P1 et P2 puissent alterner l'utilisation de la section critique, ou bien même que P1 puisse l'utiliser de façon répétitive, de telle sorte que P3 n'accédera jamais à la section critique, on dit qu'il y a *Famine*. Si un protocole rend impossible la famine, on dit aussi qu'il assure l'équité. On peut énoncer : équité = non famine = tout processus demandant la section critique l'atteint en un temps fini. La qualité d'un algorithme d'assurer l'équité s'appelle aussi la *vivacité.* L'équité entraîne le non interblocage.

Remarque : Dans un contrôle distribué basé sur l'échange de messages, pour juger si une demande d'accès à la section critique est antérieure à une autre, on se réfère à des horloges logiques et non à l'unique échelle du temps (il est impossible à un contrôle distribué de connaître l'ordre réel d'une séquence d'événements ayant lieu sur différents sites car il n'y a pas d'horloge commune).

La qualité minimum requise par un protocole d'exclusion mutuelle est, en plus de réaliser l'exclusion, d'éviter l'interblocage.

Plusieurs algorithmes d'exclusion mutuelle ont été proposés dans la littérature nous en étudierons quelques uns ; ceux qui sont basés sur les horloges de Lamport.

4.1 Algorithme de Lamport 1978

Cet algorithme, proposé en 1978, vise à satisfaire les demandes des différents sites dans l'ordre où elles sont formulées. Cela suppose évidemment l'existence d'un ordre dans les requêtes: des horloges logiques linéaires sont utilisées à cet effet.

4.1.1 Eléments de base de l'algorithme

- Cet algorithme suppose que les canaux de communication entre les différents sites respectent la propriété. FIFO.
- Chacun des composants du système utilise une horloge linéaire qu'il synchronise lors de la réception de messages en provenance des autres composants du système.
- Trois types de messages (estampillés lors de leur envoi) sont utilisés et chacun des messages sera systématiquement envoyé à tous les autres participants :
    1. REQUETE (REQ): un tel message est envoyé lorsqu'un site veut entrer en section critique;
    2. ACQUITTEMENT (ACQ) : un tel message est envoyé par un site qui reçoit un message du type précédent ;
    3. LIBERATION (LIB) : un tel message est envoyé par un site lorsqu'il sort de section critique.
- Chaque site gère une file d'attente dans laquelle il place, dans l'ordre induit par la valeur de leurs estampilles, toutes les requêtes pour entrer en section critique (y compris les siennes) et leurs estampilles. En fait, la file des requêtes sera répliquée sur chaque site, si bien que chaque site peut décider de la possibilité d'entrer en section critique sur la base des informations qu'il possède.
- Le traitement complet (entrée et sortie) d'une phase de section critique (SC) requiert, pour un système de n sites, 3*(N-1) messages (N-1 messages de chacun des types).

4.1.2 Algorithme

Pour chaque site S_i on a :

LAMPORT 3(n-1)messages

HL_j : horloge de S_i
EL_i : estampille de S_i
$FILE_i$: file d'attente de S_i

1ère étape :
- Si le site S_i veut entrer en SC, il place sa requête dans sa file d'attente $FILE_i$ et envoie un message de type *REQUETE* (REQ, S_i, EL_i), à tous les autres sites.
- Entrer en SC lorsqu'il aura effectivement reçu un accord qui est un message de type ACQUITTEMENT (ACQ, S_i, EL_i) de tous les autres sites et que sa que sa demande est la plus ancienne.

2ème étape :
- Si le site S_j reçoit un message de type *REQUETE* en provenance du site S_i, il met à jour son horloge HL_j (selon la méthode habituelle à partir des valeurs de HL_j et de EL_m), il place la requête dans sa file d'attente $FILE_j$, et envoie à S_i un message de type (*ACQ* , S_i, EL_i);

3ème étape :
- Lorsque le site S_i sort de SC, il envoie à tous les autres sites un message de type *LIBERATION* (*LIB* , S_i, EL_i)et retire sa requête de $FILE_i$;
- si le *site* S_j reçoit du site S_i un message de type *LIBERATION*, il met à jour son horloge et enlève de $FILE_j$ le message de type *REQUETE de* S_i.

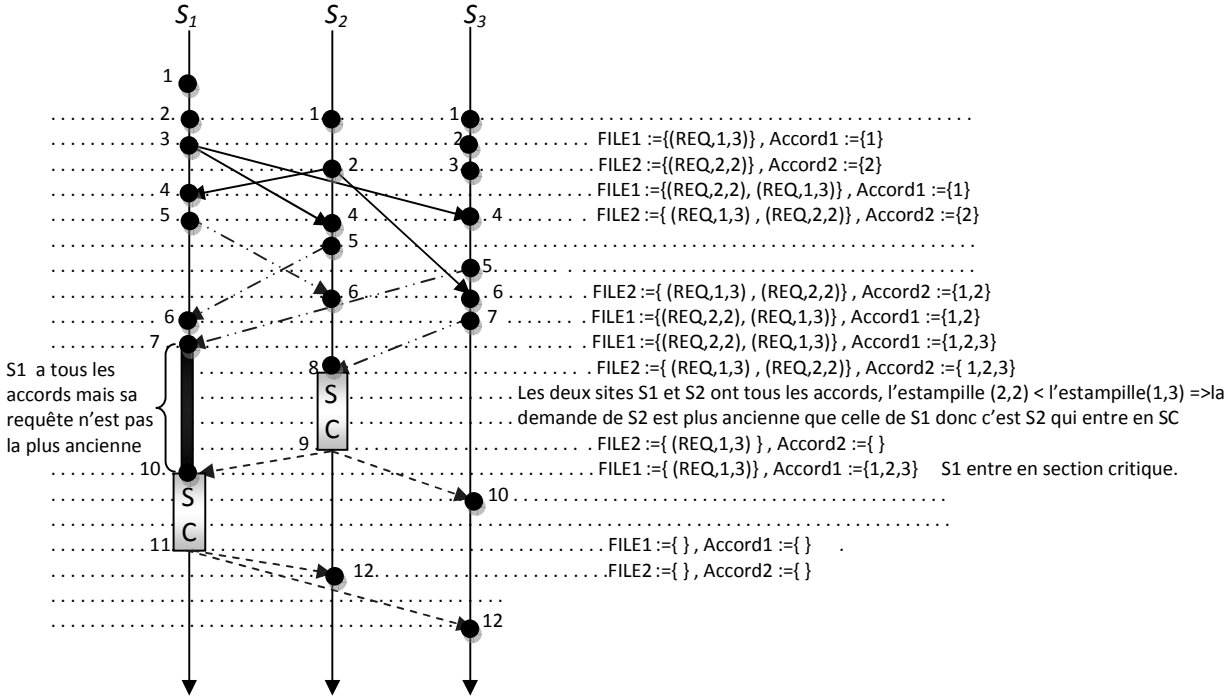
Exemple

Dans cet exemple impliquant trois sites, les sites S1 et S2 veulent entrer en section critique alors que leurs horloges linéaire ont respectivement 3 et 2 comme valeur. Les messages envoyés par S1 à S2 et S3 sont donc estampillés 3.1. Les messages envoyés par S2 à S1 et S3 sont quant à eux estampillés 2.2.

Dans la figure ci-dessous :

- les envois de messages de type REQUETE correspondent aux flèches ⟶
- les envois de messages de type ACQUITTEMENT correspondent aux flèches ⇢
- les envois de messages de type LIBERATION correspondent aux flèches ⇢

On y a matérialisé les files des requêtes et les ensembles de sites dont la réponse est arrivée pour les sites demandeurs (on n'a pas fait apparaitre ces informations pour le site S3 car ce site n'a pas demandé à entrer en section critique.
On peut observer que sur le site S1, au temps 7.1, les réponses des deux autres sites ont été reçues mais la demande de S1 n'est pas la première dans sa file des requêtes: il ne peut donc rentrer immédiatement en section critique et doit attendre de pouvoir retirer la demande qui se trouve devant la sienne.



4.2 Algorithme de Ricart-Agrawala 1981

Dans l'algorithme de Lamport, un participant qui reçoit une demande d'un autre site, y répond immédiatement. Chaque site décide lui-même s'il peut entrer en section critique en fonction des informations dont il a connaissance (ensemble des demandes en cours). Cette approche impose à un processus sortant de section critique, d'envoyer un message spécifique à tous les autres.

L'approche adoptée dans l'algorithme de Ricart-Agrawala est sensiblement différente.
Si pour entrer en section critique, un site doit toujours obtenir l'accord de tous les autres (et de lui-même), un site ne donnera son accord que si la demande correspondante n'est pas incompatible avec une demande qu'il aurait lui-même formulée préalablement, c'est-à-dire avec une estampille plus petite.
Ce sont ces demandes qu'il ne peut satisfaire immédiatement qu'un site enregistrera dans une file d'attente : à sa sortie de section critique, un site enverra les réponses qu'il a différées.
On fait ainsi l'économie des messages de libération : le nombre de total de messages pour réaliser une section critique est de 2*(N-1) (N étant le nombre de sites).

4.2.1 Eléments de base de l'algorithme

- Chacun des composants du système utilise une horloge linéaire qu'il synchronise lors de la réception de messages.

- Deux types de messages (estampillés lors de leur envoi) sont utilisés et chacun des messages sera systématiquement envoyé à tous les autres participants :
    1. REQUETE : un tel message est envoyé lorsqu'un site veut entrer en section critique ;
    2. ACQUITTEMENT : un tel message est envoyé (soit immédiatement à la réception d'un message de type REQUETE, soit ultérieurement à la sortie de section critique du site).
- Chaque site gère une file d'attente dans laquelle il place toutes les requêtes qu'il ne peut satisfaire immédiatement.

4.2.2 Algorithme

Le programme exécuté sur chaque site S_i est de la forme :

RICARD-AGRAWALA 2(n-1)messages

HL_j : horloge de S_i
EL_i : estampille de S_i
$FILE_i$: file d'attente de S_i

1ère étape : - Demande d'entrée d'un site S_i en SC. Diffusion de (REQ, S_i, EL_i) à tous les sites, S_i compris.
- Entrée en SC quand tous les sites ont donné leur accord, c-à-d un message ACQ.

2ème étape: - Réception par S_i d'une demande d'entrée en SC faite par S_j. Trois cas peuvent se présenter :

- S_i n'est ni en SC ni candidat pour y entrer : il renvoie (ACQ, S_i, EL_i).
- S_i est lui-même candidat pour rentrer en SC :
    - Si La demande de S_j est antérieure à sa demande: S_i renvoie un (ACQ, S_i, EL_i)
    - Autrement : Mise en attente par S_i de la demande de S_j dans sa file $FILE_i$.
- S_i est en section critique : Mise en attente par S_i de la demande de S_j dans sa file $FILE_i$.

3ème étape : - Libération de la SC : On envoie (ACQ, EL_i , S_i) à toutes les demandes mises en attentes par S_i, puis les retire de $FILE_i$.

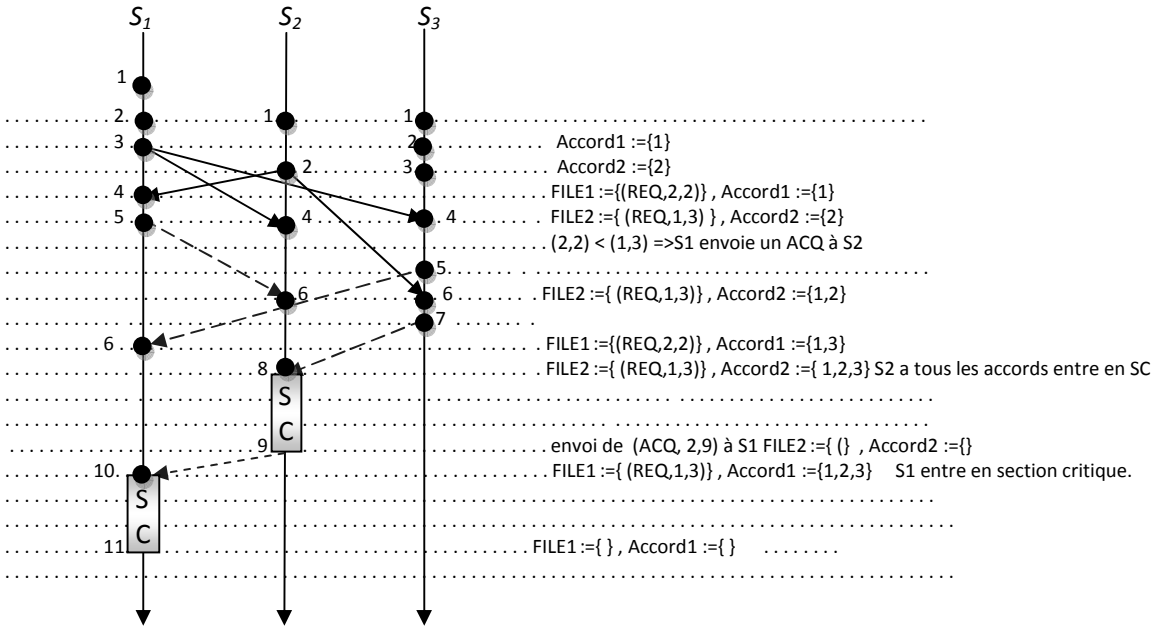
Exemple

Nous appliquons l'algorithme Ricart-Agrawala au système de 3 sites utilisé pour l'algorithme de Lamport,

Sur la figure suivante :

- les envois de messages de type REQUETE correspondent aux flèches ⟶
- les envois de messages de type ACQUITTEMENT correspondent aux flèches – – →

On peut observer sur l'exemple suivant que la réponse à la demande du site S_1 par le site S_2 est différée parce que ce site est dans l'attente d'entrer en SC et sa demande est antérieure à cette demande.



Chapitre III : COHERENCE DE L'INFORMATION DISTRIBUEE

1. Introduction

Les problèmes de cohérence d'information dans un système réparti peuvent schématiquement s'exprimer sous la forme générale suivante : étant donné la représentation répartie d'une information, fournir le moyen d'accéder à cette information comme si elle était centralisée, c-à-d représentée sur un site unique et manipulée par des processus s'exécutant sur ce site. Cette formulation recouvre plusieurs cas comme par exemple :

- *Définition et représentation de l'état global d'un système réparti* : il est impossible, à partir d'un site donné, d'accéder à un état global instantané, car les informations obtenues à partir des sites distants sont périmées au moment où elles sont reçues, et que leur ordre de réception n'est pas nécessairement compatible avec l'ordre causal. On souhaite néanmoins pouvoir accéder à un passé du système, qui soit cohérent au sens de la compatibilité avec l'ordre causal. Ce point est présenté en section 2.
- *Gestion d'informations en copies multiples :* deux raisons principales conduisent à conserver des exemplaires multiples d'une même information.
    - *La tolérance aux fautes :* pour augmenter la probabilité de retrouver un exemplaire de l'information en cas de défaillance.
    - *La recherche d'efficacité :* si une information distante est fréquemment consultée sur un site, il peut être utile d'en avoir une copie locale ; c'est le principe du *cache*.

  La gestion de la cohérence de copies multiples peut être vue comme une application des *transactions*. Néanmoins, elle peut aussi être traitée directement, en utilisant la diffusion : une modification sur une copie quelconque est diffusée à toutes les autres, et la cohérence causale est assurée si la diffusion est fiable et respecte l'ordre causal. La diffusion fiable est étudiée en section 2.

2. Etat global d'un système réparti

Obtenir une vision instantanée d'un système réparti, consistant en la collection des états des différents sites le composant (typiquement une image mémoire de chacun des sites) est difficile à obtenir sans figer chacun des systèmes.

L'absence de mémoire commune et le caractère aléatoire des délais d'acheminement des messages échangés entre les sites rendent impossible le calcul d'un état global du système dans un système réparti.

Un exemple des difficultés rencontrées lors de la définition de l'état global d'un système réparti, est fourni par la sauvegarde et de la restauration de l'état d'un système. Pour permettre à un système de reprendre son fonctionnement après une défaillance totale ou partielle, on sauvegarde périodiquement l'état de chaque processus ; le dernier état sauvegardé est restauré lors de la reprise. Dans un système réparti, il faut que l'ensemble des états partiel qui servent de base à une reprise constitue un état global cohérent.

2.1 Quelques définitions

2.1.1 Etat local d'un site S_i

C'est l'état initial et la suite des événements sur ce site depuis l'instant initial. On note EL_i .

2.1.2 Etat d'une voie de communication entre deux sites S_i et S_j

C'est l'ensemble des messages émis par S_i et non encore reçus par S_j. On note EC_{ij} .

2.1.3 Etat global d'un système distribué

Il est constitué d'un état local de chacun des sites et d'un état de chacun des canaux de communication. On note EG={ pour tout i, j ; $\cup EL_i$, $\cup EC_{ij}$ }.

Si on représente l'évolution des états des sites par des lignes parallèles, et les transmissions de messages par des flèches joignant des points sur des lignes différentes, on peut représenter un état global du système par une coupure.

2.1.4 Coupure

Une coupure C est une ligne joignant les points correspondant aux photographies instantanées c_i des états locaux. $C = < c_1 , c_2 ,..., c_n >$, où $c_i = EL_i$

Pour l'exemple 1 ci-dessous, on a :

$C=< c_1 , c_2 , c_3 >$ où $c_1=\{ e_1^1\}$, $c_2=\{ e_2^1 , e_2^2 , e_2^3 , e_2^4 \}$ et $c_3 =\{ e_3^1\}$

$D=< d_1 , d_2 , d_3 >$ où $d_1=\{ e_1^1\}$, $d_2=\{ e_2^1 , e_2^2 , e_2^3, e_2^4 \}$ et $d_3 =\{ e_3^1, e_3^2, e_3^3\}$

2.1.5 Etat global associé à une coupure

Si le système est composé de n processus, l'état global associé à une coupure est défini par l'ensemble des états contenu dans la coupure et l'état des canaux de communication.

Pour l'exemple 1, on a :

l'état global associé à C est : $\{e_1^1, e_2^1 , e_2^2 , e_2^3 , e_2^4 , e_3^1, EC_{12}=\{m3\} , EC_{13}=\{\} , EC_{21}=\{\} , EC_{23}=\{m4\}, EC_{31}=\{\} , EC_{32}=\{\} \}$

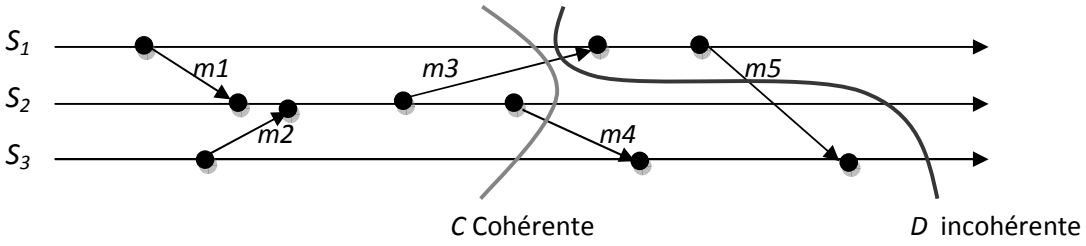
2.1.6 Coupure cohérente

D'après le principe de causalité, une coupure correspond à un état global cohérent ssi aucune flèche n'a son origine à droite de la coupure et son extrémité à gauche (un message ne peut être transmis depuis l'avenir vers le passé). De manière plus rigoureuse, une coupure C est cohérente ssi

$\forall a, b \in E , b \in C$ ET $a \rightarrow b \Rightarrow a \in C$

Cohérence=respect de la causalité c-à-d un message ne peut pas venir du futur.

Exemple1



La coupure *D* est incohérente : la réception de *m5* est dans la coupure mais pas son émission.
La coupure *C* est cohérente :

2.2 But de la recherche d'états globaux

- Trouver des états cohérents à partir desquels on peut reprendre un calcul distribué, en cas de plantage.
- Faciliter le debugguing et la mise au point d'applications distribuées.

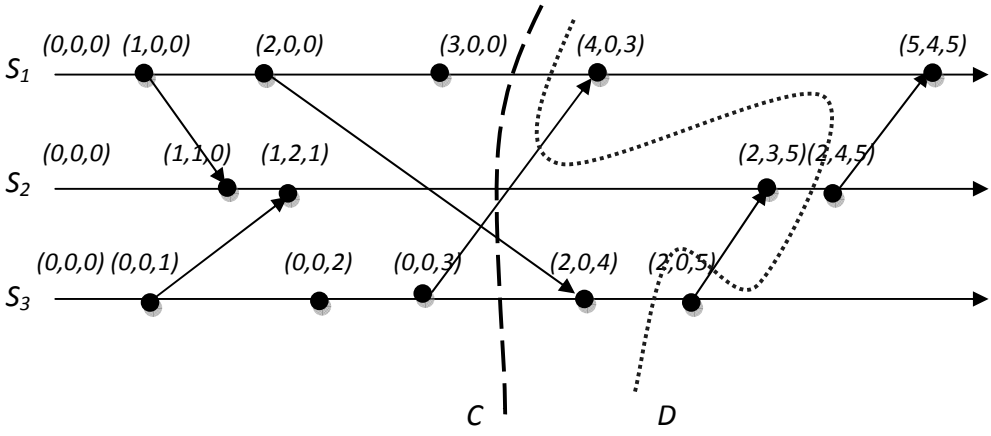
2.3 Horloges vectorielles et coupures

Lorsque la vérification de cohérence de coupure n'est pas réalisable à travers un schéma, il est possible d'utiliser les horloges vectorielles.
On peut dater les coupures au moyen des horloges vectorielles.
Soit une coupure *C* définie par les états locaux $c_1, c_2, \ldots, c_n$ respectivement sur n sites.
Soit V (c_i) la datation vectorielle de l'événement le plus récent de c_i sur le site S_i. Alors on définit la date de la coupure *C* par :

$Vc=Max[V(c_1), V(c_2), \ldots, V(c_n)]$, c'est à dire: $\forall i : Vc[i]=Max[V(c_1)[i], V(c_2)[i], \ldots, V(c_n)[i]]$,

Alors la coupure C est cohérente si et seulement si : $Vc=(V(c_1)[1], V(c_2)[2], \ldots, V(c_n)[n])$

Exemple



Question : les coupures *C* et *D* sont-elles cohérentes ?

Coupure *C*

La coupure *C* est : <c_1, c_2, c_3>, où c_1 ={ e_1^1, e_1^2,e_1^3}, c_2 ={e_2^1 , e_2^2} et c_3 ={e_3^1 , e_3^2 , e_3^3 } correspondant respectivement aux dates : [(3,0,0), (1,2,1), (0,0,3)]

VC =[Max(3,1,0), Max(0,2,0), Max(0,1,3)]=[3,2,3]

C'est une coupure cohérente car : Vc[1]=V(e_1^3)[1]=3

Vc [2]=V(e_2^2)[2]=2

Vc [3]=V(e_3^3)[3]=3

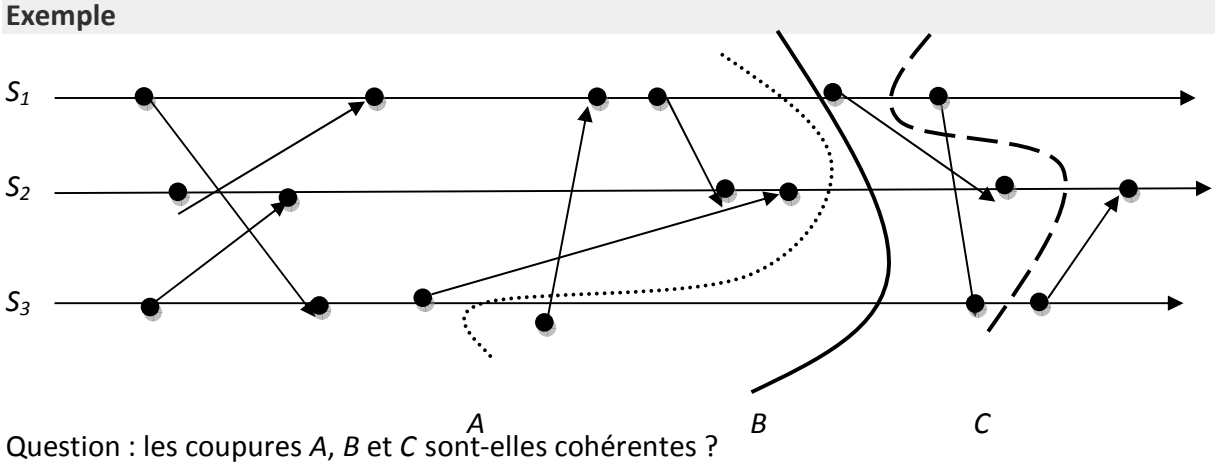
Coupure *D*

La coupure *D* est : <d_1, d_2, d_3>, où d_1 ={ e_1^1, e_1^2,e_1^3}, d_2 ={e_2^1 , e_2^2, e_2^3} et d_3 ={e_3^1 , e_3^2 , e_3^3 , e_3^4 } correspondant respectivement aux dates : [(3,0,0), (2,3,5), (2,0,4)]

VD =[Max(3,2,2), Max(0,3,0), Max(0,5,4)]=[3,3,5]

On a VD[3]= 5 et V(e_3^4)[3]=4 donc VD[3] $\neq$ V(e_3^4)[3] ce qui veut dire que cette coupure est incohérente.

Exemple



Question : les coupures *A*, *B* et *C* sont-elles cohérentes ?

2.4 Détermination d'un état cohérent, Algorithme de Chandy et Lamport (1985)

Nous décrivons maintenant le principe de l'algorithme de *Chandy* et *Lamport* pour l'enregistrement de l'état global d'un système réparti. L'enregistrement peut être déclenché par un processus quelconque, et éventuellement par plusieurs processus à la fois. Les système est constitué d'un ensemble de processus reliés par des canaux de communication, sur lesquels les messages sont reçus dans l'ordre de leur émission (propriété FIFO). L'algorithme repose sur l'utilisation d'une information particulière appelée *marqueur*, qui est envoyée sur chaque canal et permet de séparer les messages envoyés avant et après l'enregistrement de l'état local par le processus émetteur. L'algorithme s'arrête quand chaque processus a reçu un marqueur sur chacun de ses canaux. L'état global calculé est cohérent, il définit une coupure cohérente. L'algorithme fonctionne comme suit :

2.4.1 Contraintes sur le système

- Canaux de communication unidirectionnels et FIFO.
- Fiabilité : pas de plantage ou de perte de messages.

CHANDY-LAMPORT

1ère Etape : un processus qui déclenche l'enregistrement de l'état global, enregistre son état, puis envoie un marqueur sur chacun de ses canaux de sortie, avant tout autre envoi.

2ème Etape : un processus Pi qui reçoit un marqueur sur un canal c exécute le programme suivant :

If etat(Pi) non encore enregistré **then**

- Enregistrement de l'état de Pi:

- envoyer un marqueur sur tous les canaux de sortie ;

- Etat(c) :=∅;

- Enregistrement de l'état du canal c.

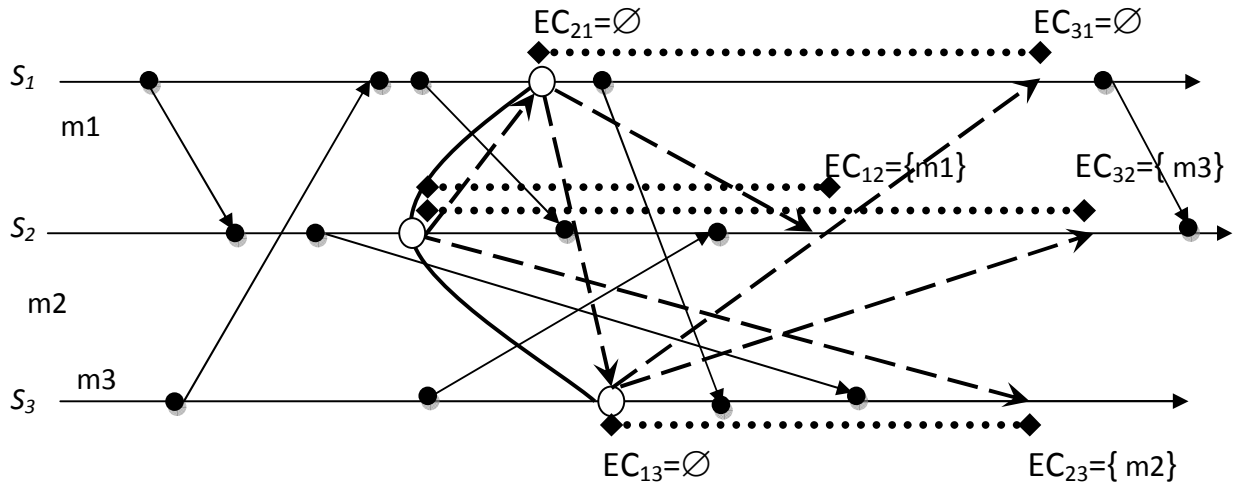
Else
- enregistrement de l'état du canal c = etat(c)= suite des messages reçus sur c entre l'enregistrement de l'état de Pi et la réception du marqueur.

Une fois enregistrés tous les états des processus et des canaux, un algorithme de terminaison doit être utilisé pour collecter l'ensemble de l'information, qui constitue l'état global. Il faut noter que l'état global enregistré ne correspond pas nécessairement à un état effectivement atteint par le système. Néanmoins, cet état est cohérent au sens où il est un état intermédiaire

dans une séquence d'événements qui est une permutation d'une séquence réelle avec des états initiaux et finaux identiques.

Exemple

On considère le système à 3 sites décrit ci-dessous. Le site S2 lance l'algorithme de Chandy-Lamport pour l'enregistrement d'un état global cohérent. Dérouler l'algorithme.



: Enregistrement du canal

: Marqueur

: Sauvegarde de l'état du site

L'état global ainsi défini dans cette exemple correspond à : EG={ $e_{1,1}$, $e_{1,2}$, $e_{1,3}$, $e_{2,1}$, $e_{2,2}$, $e_{3,1}$, $e_{3,2}$, $EC_{21}=\emptyset$, $EC_{31}=\emptyset$, $EC_{12}=\{m1\}$, $EC_{32}=\{\}m3$, $EC_{13}=\emptyset$, $EC_{23}=\{m2\}$}, c'est un état cohérente.

3. Diffusion fiable

Etant donné un groupe de n sites reliés par un réseau de communication, la *diffusion* d'un message est l'envoi de ce message à destination de chacun de ces sites par un émetteur pouvant ou non appartenir au groupe. En pratique, on demande en général que la diffusion soit *fiable*, c-à-d tous les destinataires du message corrects, qui ne sont pas en panne ou fautifs, doivent le recevoir, ou bien en cas de panne non récupérable de l'émetteur pendant la diffusion, aucun ne doit le recevoir. Un protocole de diffusion fiable doit résister aux pannes du système de communication et des sites. Il existe deux types de diffusion : diffusion générale ou *Broadcast* et diffusion de groupe ou *multicast*.

Diffusion générale (Broadcast) : les destinataires sont tous les sites du système. L'émetteur est également destinataire.

Diffusion de groupe (multicast) : les destinataires sont les membres d'un ou plusieurs groupe(s) spécifiés. Tous les sites du système. L'émetteur peut ne pas appartenir au(x) groupe(s) destinataire(s).

La diffusion est utilisée dans plusieurs applications comme :

- Programmation d'applications réparties : par exemple dans certaines applications, on cherche à accélérer la recherche d'une information en faisant exécuter la recherche sur plusieurs processus en parallèle. Un processus qui a trouvé l'information doit alors prévenir les autres.
- Gestion d'informations en copies multiples : la diffusion sert à mettre à jour toutes les copies.
- Observation de mesures : si l'ordre causal est préservé, il devient possible à un observateur unique d'avoir une vue globale cohérente de l'ensemble du système.

Il existe de nombreuses formes de diffusions selon les propriétés requises. Au minimum on demande la propriété de fiabilité (tout ou rien), déjà vue, qui garantit une forme de connaissance partagée (propriété d'accord).

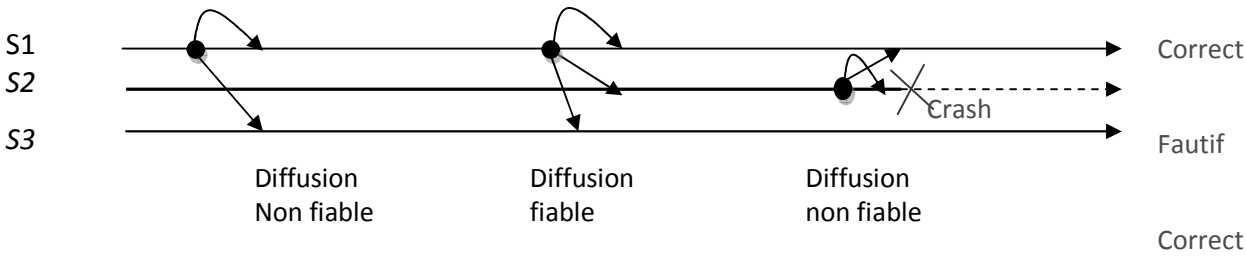
3.1 Propriétés indépendantes de l'ordre d'émission

Cette catégorie de propriétés concerne uniquement les destinataires.

3.1.1 Diffusion fiable : un message est délivré à tous ses destinataires ou à aucun.

Exemple

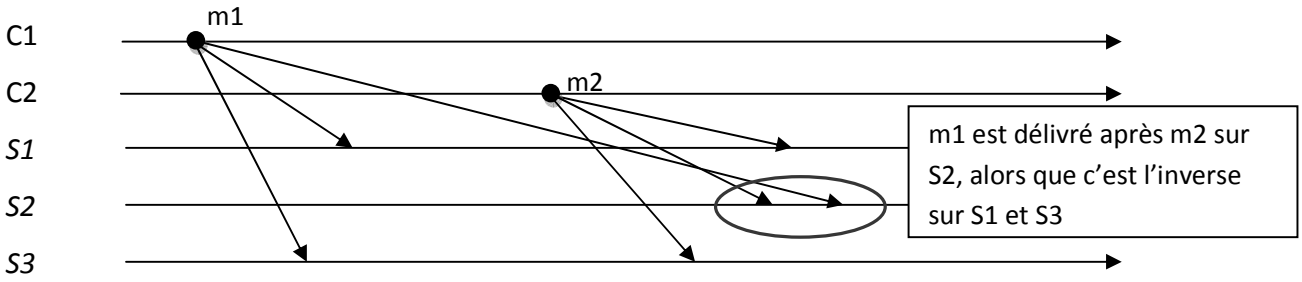
On considère le système à 3 sites décrit ci-dessous. Trois cas de diffusion sont présentés.



3.1.1 Diffusion atomique ou totalement ordonnée : la diffusion est fiable et les messages sont délivrés dans le même ordre à tous leurs destinataires.

Exemple

On considère le système à 5 sites décrit ci-dessous. nous sommes dans le cas d'une diffusion de groupe.



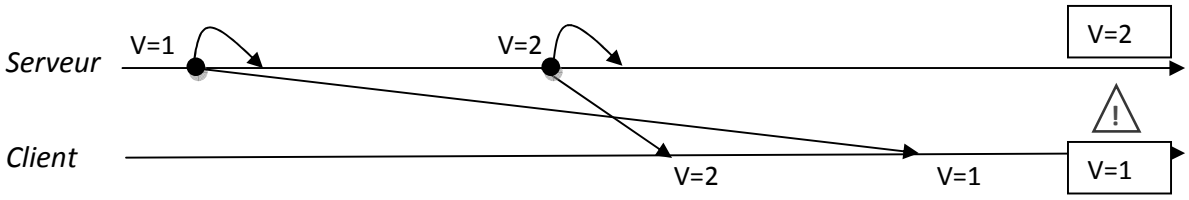
3.2 Propriétés liées à l'ordre d'émission

La propriété de fiabilité ne spécifie rien sur l'ordre de réception des messages par les destinataires, ni sur la relation entre l'ordre d'émission et l'ordre de réception. On peut spécifier les propriétés suivantes :

3.2.1 Diffusion FIFO : deux messages issus du même émetteur sont délivrés à tous récepteur dans leur ordre d'émission. C'est-à-dire que, si un membre diffuse m1 puis m2, alors tout membre correct doit délivrer m1 avant m2.

Exemple

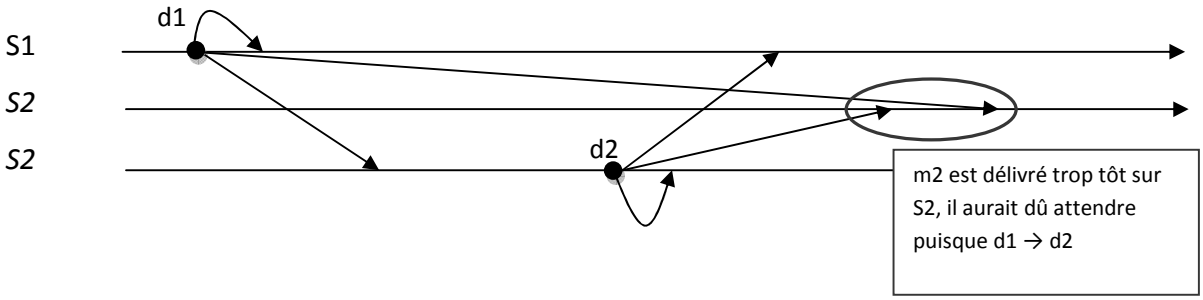
On considère le système à 2 sites décrit ci-dessous. Le site Serveur lance la mise à jour de V.



3.2.2 Diffusion Causale : pour tout récepteur, l'ordre de réception de deux messages respecte leur ordre causal d'émission (implique FIFO). C'est-à-dire que, si la diffusion de m1 précède causalement la diffusion de m2, alors tout processus correct doit délivrer m1 avant m2.

Exemple

On considère le système à 3 sites décrit ci-dessous.



3.2 Types de diffusion fiable

Les deux classes de propriétés précédemment présentées sont indépendantes, on peut avoir plusieurs combinaisons possibles comme :

Diffusion FIFO = Diffusion fiable + Ordre FIFO

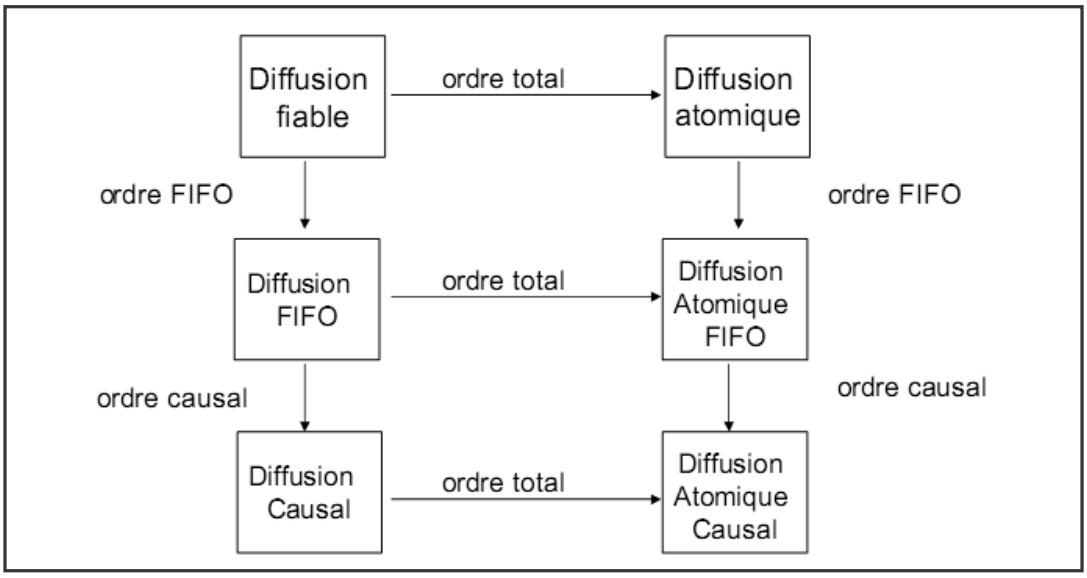
Diffusion Causale (CBCAST : *Causal broadcast*) = Diffusion fiable +Ordre Causal

Diffusion Atomique (ABCAST: *atomic broadcast*) = Diffusion fiable + Ordre Total

Diffusion Atomique FIFO = Diffusion FIFO+Ordre Total

Diffusion Atomique Causale = Diffusion Causale+Ordre Total

3.3



Protocoles de diffusion fiable

Plusieurs protocoles de diffusion fiable ont été proposés en littérature, nous en étudierons trois : le premier utilise un séquenceur qui garantit la diffusion FIFO, le second protocole garantit la diffusion causale et le troisième la diffusion atomique.

3.3.1 Protocole utilisant un serveur (diffusion FIFO)

Ce protocole est écrit par *Kaashock* en 1989. Il utilise un serveur de séquencement appelé *séquenceur*, sur un site particulier, pour assurer la mise en ordre uniforme des messages. Ce protocole garantit la diffusion FIFO mais pas la diffusion causale.

Pour diffuser un message, l'émetteur l'envoie au séquenceur. Celui-ci lui attribue un numéro d'ordre et le diffuse (on suppose que le message est destiné à tous les sites du système et non à un groupe). Le séquenceur conserve une copie des messages qu'il a diffusé, avec leur numéro d'ordre, dans un tampon « historique ». si tous les messages jusqu'à un numéro d'ordre k ont été acquittés par tous les destinataires, ils peuvent être éliminés du tampon.

Chaque destinataire accepte les messages dans l'ordre de leurs numéros et conserve le numéro *s* du dernier message reçu. Si un message arrive avec un numéro supérieur à *s+1* le récepteur le met en attente et demande au séquenceur de lui envoyer une copie du ou des messages manquants. Le séquenceur retrouve ces messages dans son tampon historique.

Si le tampon devient plein, le séquenceur entame une phase de resynchronisation dans laquelle il s'assure que tous les messages présents dans le tampon ont bien été reçus par tous les destinataires.il renvoie si nécessaire les messages qui ont été perdus, puis il vide le tampon.

3.3.2 CBCAST : Protocole respectant l'ordre causal (diffusion causale)

Le protocole CBCAST (pour Causal BroadCAST) est un protocole de diffusion qui respecte la dépendance causale. Il a été réalisé par *Birman* en 1990, en utilisant les horloges vectorielles. Son principe est le suivant.

PROTOCOLE CBCAST

Chaque site Sk du système maintient un vecteur VSk à n composants (n: nombre de sites). Le composant Vk[i] (i=1,…,n) indique le numéro associé au dernier message reçu en provenance du site i.

1ère Etape : avant de diffuser un message m, un site Si exécute VSi[i] := VSi[i]+ 1, et associe au message m l'estampille Vm=VSi

2ème Etape : à la réception d'un message m estampillé par Vm par un site Sj depuis un site Si, m est mémorisé. Ensuite, Sj teste si le message m satisfait les conditions (a) et (b) ci-dessous, pour être délivré ou s'il doit attendre pour respecter l'ordre causal :

a. Vm[i] = VSj[i]+1
b. $\forall$ k$\neq$i , Vm[k]$\leq$ VSj[k] :

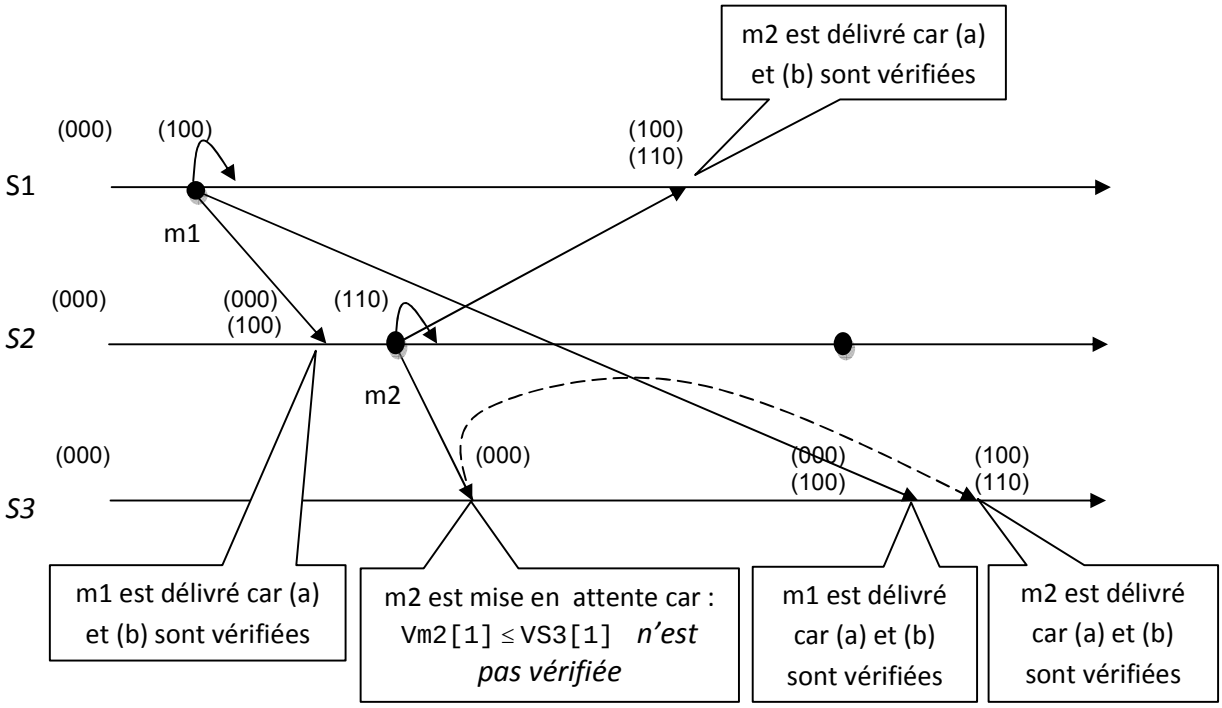
3ème Etape : après la délivrance de m, l'horloge locale VSj de Sj est mise à jour : VSj[k] :=Max[VSj[k], Vm] pour k :=1,…,n.

La condition (a) garantit que le récepteur Sj na raté aucun message en provenance du site Si (ceci est requis car : deux messages en provenance d'un même émetteur vers un même récepteur sont toujours causalement liés, c'est la propriété FIFO).
La condition (b) garantit que l'émetteur Si n'a reçu aucun message que le site Sj n'a pas encore reçu (ceci est requis pour être sûr que le message de l'émetteur Si n'est pas causalement lié à un message que le site Sj n'a pas encore reçu).

Exemple

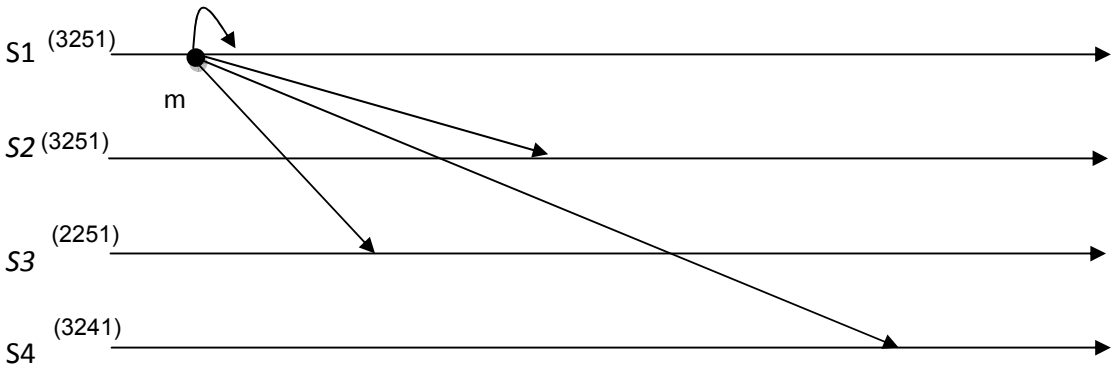
On considère le système à 3 sites décrit ci-dessous. Le site S1 diffuse un message m1, . le site S2 diffuse un message m2.

Les valeurs courantes des horloges de chaque site avant et après délivrance sont respectivement en haut et en bas.



Exemple

On considère le système à 4 sites décrit ci-dessous. le site S1 diffuse un message m. En considérant les valeurs des horloges locales à chaque site, et en appliquant l'algorithme CBCAST, on aura : S1, S2 qui délivre le message et S3, S4 le mettent en attente.



3.3.3 ABCAST : Protocole utilisant estampilles (diffusion Atomique)

Le protocole ABCAST (pour Atomique BroadCAST) écrit par Birman en 1987, assure un ordre de réception uniforme par ordonnancement global des messages. Cet ordonnancement est réalisé au moyen d'estampilles. Le principe est le suivant :

Un site qui reçoit un message à destination d'un processus de ce site, le met dans une file d'attente locale au site et le marque comme «en attente ». il renvoie à l'émetteur un accusé de réception estampillé par la date de réception du message. Lorsque l'émetteur a reçu tout les accusés de réception pour un message, il affecte à ce message une estampille définitive qui est la plus grande des différentes estampilles des accusés de réception du message. Cette estampille est rediffusée à tous les sites destinataires. Chaque destinataire affecte définitivement cette estampille au message correspondant en attente, marque celui-ci comme « prêt » et réordonne la file dans l'ordre croissant des estampilles. Si le message de tête est dans l'état « prêt », il est sorti de la file et délivré à ses destinataires sur le site.

Tolérance aux pannes dans les systèmes distribués

Alain BUI
Professeur
Département de Mathématiques et Informatique
alain.bui@univ-reims.fr

Introduction

- Nombre croissant de composants dans un système => Probabilité plus grande que des composants tombent en panne pendant l'exécution de l'algorithme
- Causes diverses
  - Erreurs de conception, Accidents, Malveillances etc.
- Objectif: éviter de relancer un algorithme après chaque panne => concevoir des algorithmes capables de fonctionner malgré des pannes.

- Système Distribué: panne => système est affecté partiellement et improbable qu'il le soit en totalité
- Idée de solution: les sites corrects prennent en charge les tâches des sites défaillants
- Conséquence: perte de performances mais pas de fonctionnement erroné

Définition

- erreurs => défaillances => fautes
- Un composant est défaillant s'il ne répond plus à sa spécification (composant en SD = lien ou site)
- Une faute ou panne désigne une défaillance temporaire ou définitive d'un ou plusieurs composants du système

Spécifications

- Spécifications pour les sites
  - Si un site n'a pas atteint un état final, il finira par exécuter une autre étape de l'algorithme
- Spécifications pour les liens de communications
  - Un site j reçoit un message d'un site i au plus une fois et seulement si i a précédemment envoyé le message à j
  - Si i a envoyé un message à j et j exécute infiniment des étapes de l'algorithme alors j finira par recevoir le message de i.

Algorithmes Robustes

- Robuste : Garantir la correction du comportement global du système vis à vis des spécifications de l'algorithme
  - Spécifications définies en terme d'invariants qui doivent être constamment vérifiés
  - Aucun dysfonctionnement n'est toléré pour le système
  - Algos robustes masquent les fautes
  - Approche dite *pessimiste*

Algorithmes Robustes : exemple comportemental

- Exclusion Mutuelle
  - Propriétés de sûreté et de vivacité **toujours** vérifiées
  - Par exemple, on ne se retrouvera jamais avec une configuration où 2 sites sont en même temps en SC
- Élection
  - Un et un seul site sera élu
  - Par exemple, à aucun moment il existe une configuration où simultanément plusieurs sites décident qu'ils sont élus.

Algorithmes auto-stabilisants

- **Finir** par garantir la correction du comportement global du système vis à vis des spécifications de l'algorithme
  - Système tolère certaines périodes de dysfonctionnement
  - Algorithmes ne masquent pas les fautes
  - Approche dite *optimiste*

Algorithmes Auto-stabilisants: exemple comportemental

- Exclusion mutuelle
  - Propriété de sûreté non vérifié pendant un intervalle de temps
  - Deux sites peuvent se retrouver en SC
  - MAIS au bout d'un moment le système retrouve un comportement correct (l'algorithme retrouve de lui même un état valide)

Classification des fautes

- Des critères
  - Origine de la faute
    - Type de composant : ex. lignes ou sites
  - Cause de la faute: bénignes ou malignes
    - Défaillances temporaires ou définitives : si le composant fonctionne il fonctionne correctement

    ex. ligne transmet le msg ou non / site traite le msg ou non

    - Défaillances byzantines : comportement arbitraire du composant défaillant. Si le composant fonctionne, il ne fonctionne par correctement à l'insu des autres composants.

    ex. site répond « blanc » à certains sites et « noir » à d'autres.

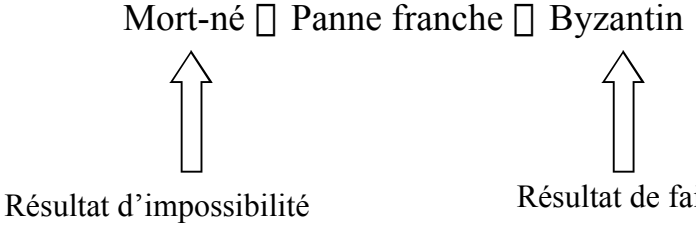
Classification (suite)

- Durée de la faute
  - Définitive
  - Temporaire
- Détectabilité de la faute
  - Détectable localement. Réparation par le site lui-même.
  - Non détectable localement. Réparation nécessite échange de messages.

- Différentes classification selon ces critères, en voici une …

Une hiérarchie

- Sites
  - Site mort-né : site n'exécute aucune instruction de son algo.
  - Site en panne franche : site fonctionne correctement jusqu'à l'apparition de la panne et cesse totalement de fonctionner.
  - Site byzantin : comportement arbitraire.



Typologie

- Panne franche
  - Composant fonctionne correctement puis panne et cesse immédiatement de fonctionner = panne permanente
    - Panne franche de site
    - Coupure d'une ligne => changement de topologie du réseau

- Panne transitoire
  - Comportement erroné des composants pendant une certaine période. Comportement correct ensuite.
    - On peut distinguer si la panne n'apparaît qu'une fois ou plus ou moins périodiquement
    - Corruption mémoire
    - Annulation d'une transaction
    - Perte de messages sur une ligne
- Panne byzantine
  - Toute panne engendrant une comportement s'écartant des spécifications

Robustes vs Auto-stabilisants

	Fautes transitoires	Fautes définitives	Masquant
AS	OUI	NON	NON
Robuste	NON	OUI	OUI

- 2 approches complémentaires
- Choix dépend du problème à résoudre