Synchronisation Distribuée
Diffusion fiable

Laurent PHILIPPE

Master 2 Informatique
Ingénierie des Systèmes et Logiciels

2021/2022

UNIVERSITÉ DE FRANCHE-COMTÉ

femto-st SCIENCES & TECHNOLOGIES

La diffusion

Définition

Envoi d'un message depuis un processus à plusieurs processus.

- Diffusion générale : tous les processus d'un réseau (broadcats)
- Diffusion de groupe : sous-groupe des processus (multicast)

Intérêts

- Informer/Interroger un ensemble de processus
- Tous les processus reçoivent le même message

Exemples

- Lamport
- Calcul distribué : calcul de convergence
- Gestion de consensus

La diffusion fiable ou synchronisée

Problème

- Diffusions concurrentes par plusieurs processus
- Ordre de réception quelconque des messages
- Interprétation différente de l'évolution de l'état du système

Principe

- Respecter certaines propriétés
- Ordre de réception absolu impossible
- Ordre de réception relatif entre les processus

Rappel

Hypothèses de travail

- Les processus sont identifiés par un ID
- Temps de communication finis mais non prévisibles
- Exécution d'un processus est une suite d'évènements
- Évènements internes ou externes
- Pas de panne des processus (machines)
- Messages arrivent intacts
- Messages ne sont pas dupliqués

Sommaire

Diffusion fiable

Propriété, définition

Pour être fiable, une diffusion doit satisfaire 3 critères :

- **validité** : lorsqu'un processus diffuse, tous les membres du groupe reçoivent ;
- **accord** : si un processus reçoit, alors tous les autres membres du groupe reçoivent ;
- **intégrité** : chaque message n'arrive qu'une et une seule fois.

Diffusion fiable

Différents types de diffusions fiables

- FIFO : les messages sont délivrés dans le même ordre que l'ordre d'envoi, protocole FBCAST
- Causale : les messages sont délivrés dans un ordre compatible avec le respect de la causalité, protocole CBCAST ;
- Atomique : les messages sont tous délivrés dans le même ordre, protocole ABCAST.

Diffusion fiable

Mise en œuvre

Dans le contexte des protocoles de diffusion fiable

- Le système n'a aucun contrôle sur le temps de communication des messages, ni sur la date d'arrivée ou de réception des messages
- Faire la distinction entre *réception* d'un message et *délivrance* de ce message
- Différer la délivrance d'un message au processus destinataire afin de se conformer aux définitions précédentes
- Ne favorise pas les performances du système, mais garantit le bon fonctionnement d'une application distribuée.

Les protocoles de diffusion fiable

Notations

- **reçoit**$_p(m)$: réception du message m par le processus P ;
- **delivre**$_p(m)$: livraison du message m au processus P

Propriété

Dans tous les cas et de manière évidente, la réception précède toujours la délivrance du message. On note :

$$\textbf{reçoit}_p(m) \preceq \textbf{delivre}_p(m)$$

Sommaire

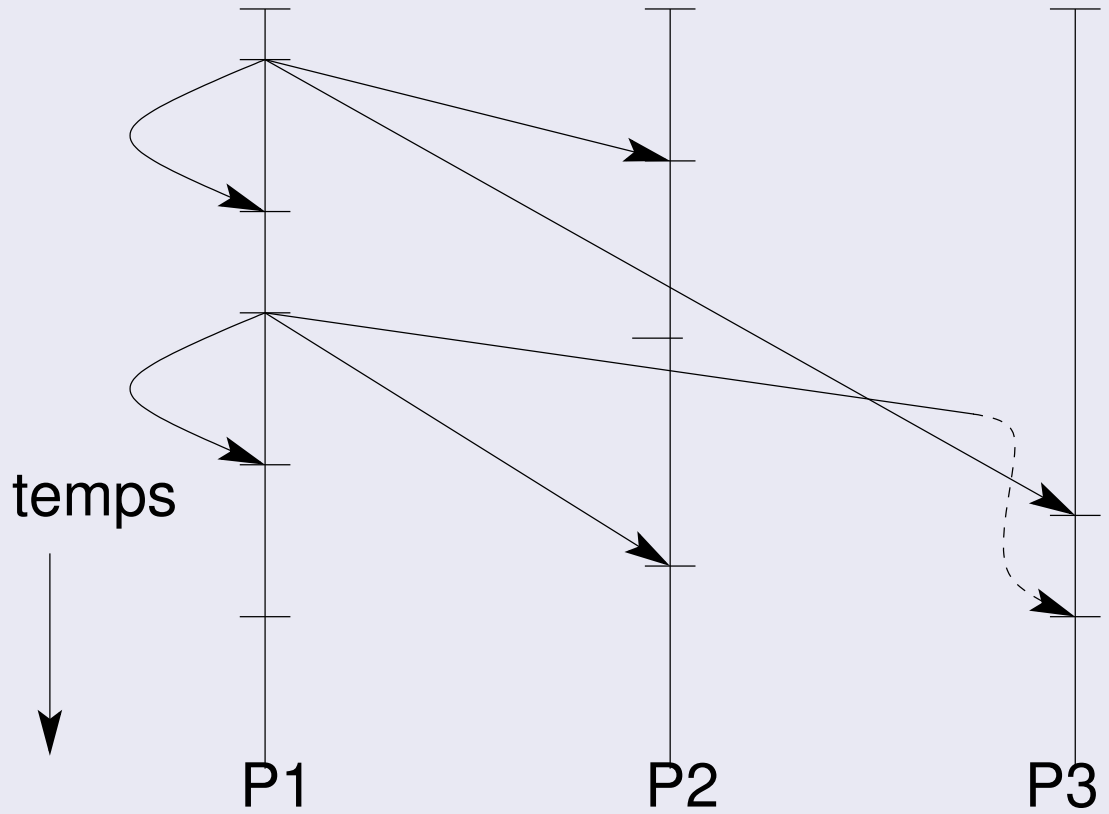
Ordre FIFO : FBCAST

Définition

Si un processus diffuse un message m_1 avant de diffuser un message m_2, alors aucun processus correct ne délivre m_2 à moins qu'il n'ait déjà délivré m_1.
Les messages envoyés par un processus P sont délivrés dans l'ordre d'émission par tout processus Q correct :
Soient les messages m_1, m_2 et $P \in G \mid$
$\text{send}_p(m_1) \preceq \text{send}_p(m_2) \Longrightarrow \forall Q \in G : \text{delivre}_q(m_1) \preceq \text{delivre}_q(m_2)$

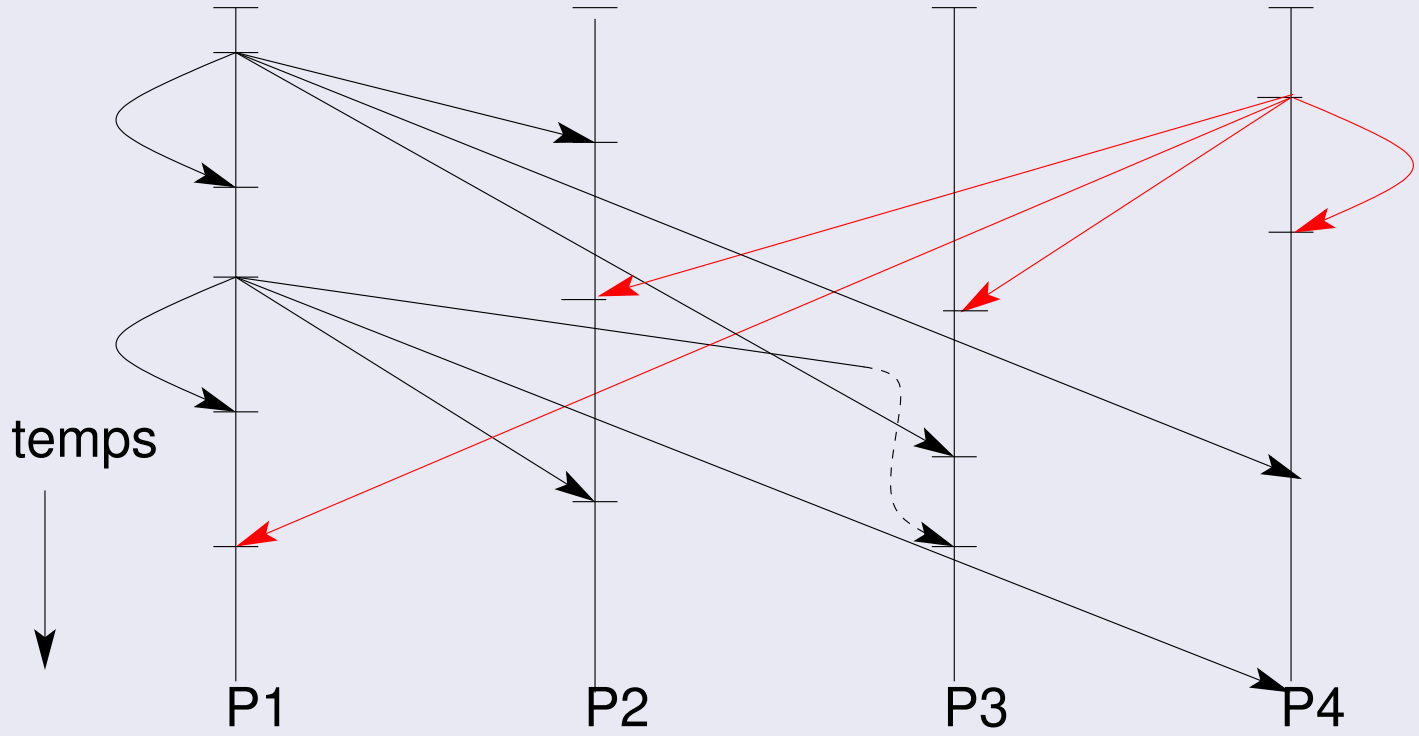
Ordre FIFO : FBCAST

Exécution correcte d'une diffusion fiable de type FIFO



Ordre FIFO : FBCAST

Exécution correcte d'une diffusion fiable de type FIFO



Protocole FBCAST

Protocole

- Le message m est accompagné du numéro de séquence d'émission $\#seq(m)$ et du numéro du processus émetteur $sender(m)$
- Sur le processus émetteur, les numéros de séquence d'émission constituent une suite continue (sans trou)
- Alors chaque processus P connaît le prochain numéro de séquence $next(Q)$ du message à délivrer en provenance de chaque processus Q
- Grâce à ces informations, le processus P récepteur d'un message m est capable de délivrer ou non le message, et s'il le délivre, il essaie de délivrer tous les messages en provenance de Q qui n'auraient pas encore été délivrés

Protocole FBCAST

Protocole

- Initialement, les numéros de séquence des messages en provenance de tous les autres processus ont la valeur 1, sur tous les processus
- À la réception, sur P d'un message m en provenance de Q, m est stocké parmi les messages attendant d'être délivrés ;
- Tant qu'il existe un message m en provenance de Q tel que le prochain numéro de séquence du message $next(Q)$ est égal à $\#seq(m)$, alors le message m est délivré, supprimé de l'ensemble des messages à délivrer et le prochain numéro de séquence du message à délivrer en provenance de Q ($next(Q)$) est incrémenté.

Protocole FBCAST

Exercices

- Ecrire l'algorithme sous forme de règles
- On dipose d'une file de messages en local *file*, avec les fonctions suivantes :
  - $put(message, emetteur, numseq)$
  - $message = get(emeteur, numseq)$

Sommaire

Relation de causalité (ou précédence)

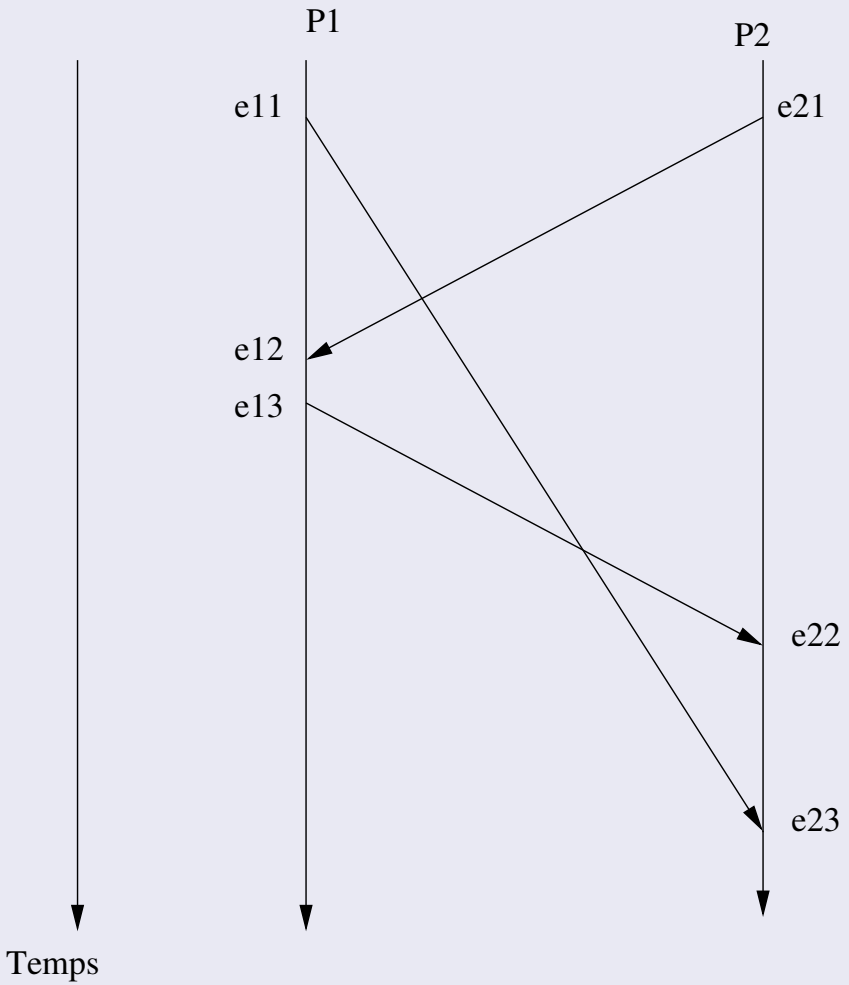
Définition/Rappel

- La relation de causalité est notée $\preceq$: $e_1 \preceq e_2$, e_1 cause de e_2
- Par exemple : l'émission d'un message précède la réception de celui-ci ou un évènement local en précède un autre
- Relation d'ordre stricte (irréflèxive, anti-symétrique et transitive) mais partielle : on ne peut pas toujours mettre en relation deux évènements quelconques.

Chemin causal

- Suite d'événements ayant une relation de causalité
- Si $e_i \preceq e_j$, alors il existe un chemin causal entre e_i et e_j
- Deux événements concurrents ne sont pas liés par une dépendance causale et il n'y a pas de chemin qui les relie

Exemple de relation de causalité



- $e_{11} \preceq e_{23}$
- $e_{13} \preceq e_{22}$
- $e_{21} \preceq e_{12}$
- Les événements suivants sont concurrents : $e_{11} \parallel e_{21}$

Estampilles vs. Relation de causalité

Estampilles

- Estampille définit un ordre total : tous les événements peuvent être ordonnés les uns par rapport aux autres
- Estampilles sont cohérentes avec la dépendance causale : si $e_1 \preceq e_2$ alors $H(e_1) < H(e_2)$
- En cas de concurrence entre évènements, l'ordre arbitraire entre estampilles égales (numéro de processus) est sans risque pour la causalité

Estampilles vs. Relation de causalité

Relation de causalité

- Dans certains cas, l'ordre total est inutile : on a besoin de la causalité entre évènements, savoir si un évènement e_1 dépend d'un évènement e_2.
- Mais les estampilles effacent la notion de dépendance causale : si $H(e_1) < H(e_2)$ on ne sait rien sur le lien causal entre les évènements qui peut exister ou non.
- Les estampilles ne sont pas *denses* : si $H(e_i) < H(e_j)$, on ne peut pas savoir s'il existe e_k tel que $e_i \preceq e_k$ et/ou $e_k \preceq e_j$.

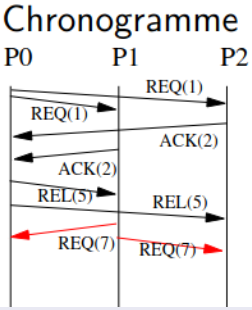
Estampilles vs. Relation de causalité

Exécution Lamport

P_1 demande à entrer en SC

Var	h	F_H	F_M
P_0	5	5,2,2	REL, ACK, ACK
P_1	7	5,7,0	REL, REQ, REL
P_2	6	5,0,0	REL, REL, REL

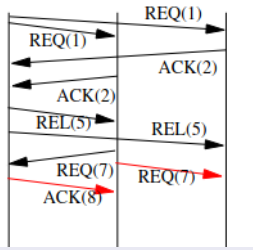
Chronogramme :



P_0 reçoit REQ(7) de P_1

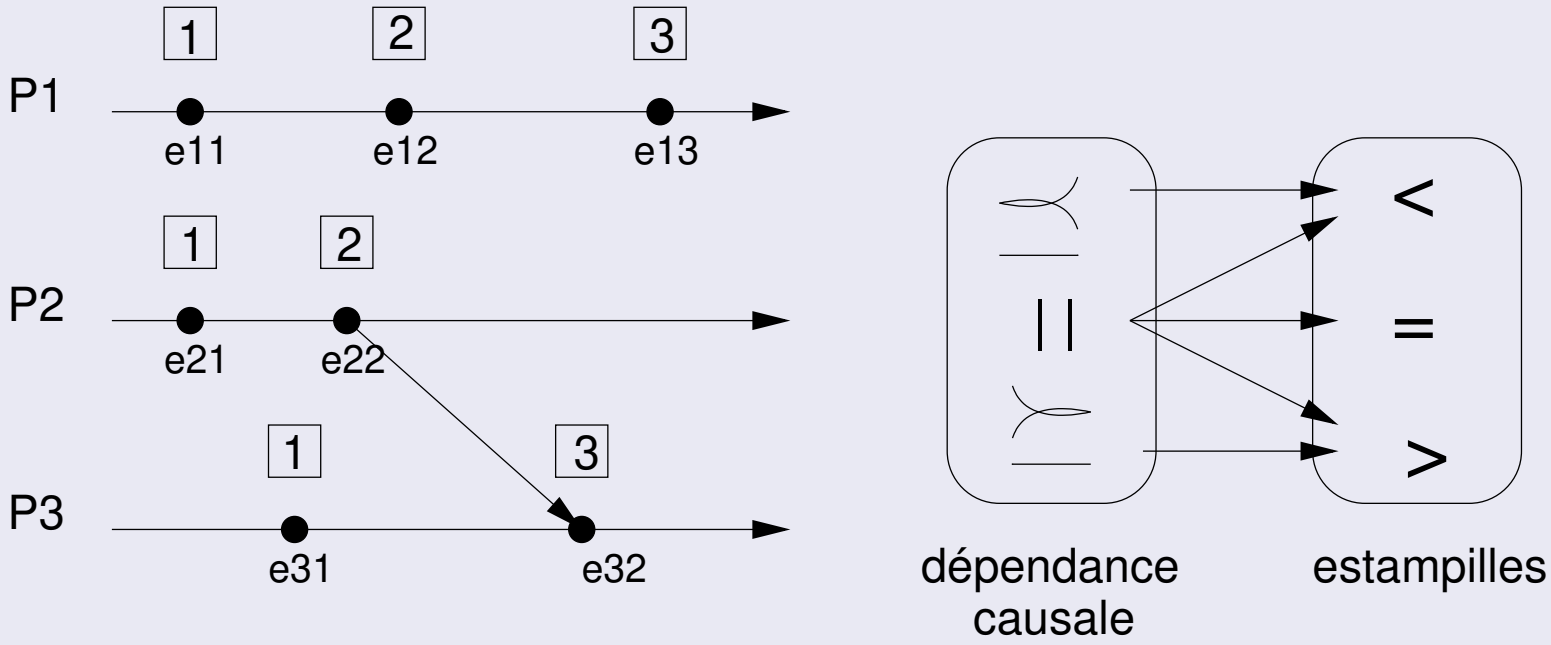
Var	h	F_H	F_M
P_0	8	5,7,2	REL, REQ, ACK
P_1	7	5,7,0	REL, REQ, REL
P_2	6	5,0,0	REL, REL, REL

Chronogramme :



Existe-t-il un évènement sur P_0 entre 5 et 8 ?

Interaction indépendance causale $\leftrightarrow$ estampille



Les historiques

Causalité

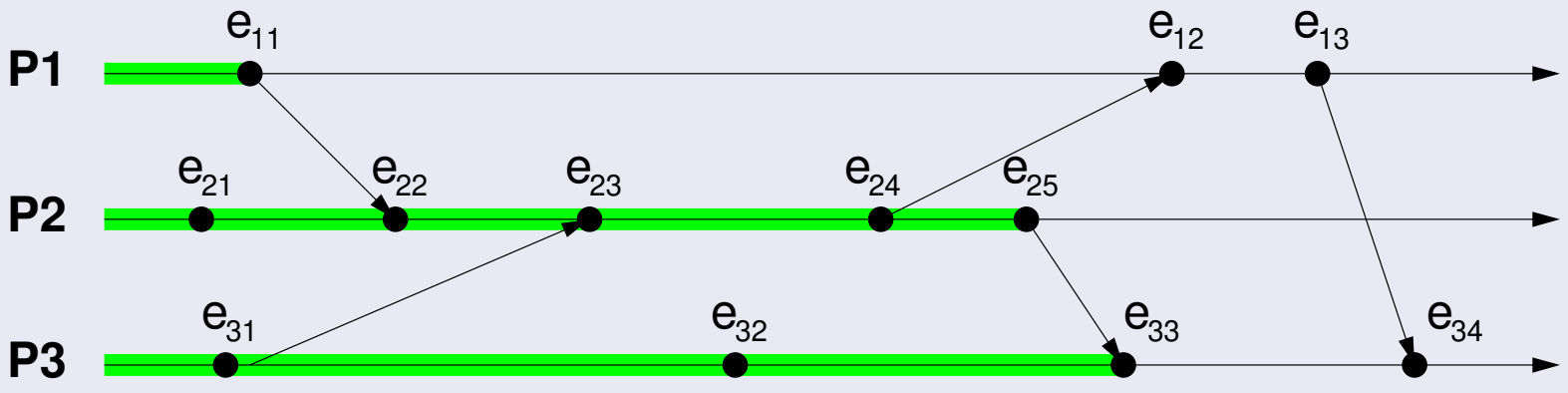
- Utilisés par protocoles basés sur la causalité
- Pour savoir si deux évènements ont une dépendence causale, il faut conserver l'historique.

Définition

- On définit par $hist(e)$ l'historique d'un événement e
- $hist(e)$ = ensemble des événements appartenant à tous les chemins causals
- $hist(e) = \{e\} \cup \{e' \mid e' \preceq e\}$

Les historiques

Exécution d'événements sur 3 processus



$$hist(e_{33}) = \{e_{11}, e_{21}, e_{22}, e_{23}, e_{24}, e_{25}, e_{31}, e_{32}, e_{33}\}$$

Les historiques

- Historique d'un événement e est utilisé pour la datation de e
- Examen de son passé permet de déterminer ses dépendances causales
- Dépendance de deux événements e et e' :

$$\begin{cases} hist(e) & = & \{e\} \cup \{e' \mid e' \preceq e\} \\ e \preceq e' & \Leftrightarrow & e \in hist(e') \\ e \parallel e' & \Leftrightarrow & (e \notin hist(e')) \wedge (e' \notin hist(e)) \end{cases}$$

- Inconvénient de l'historique : sa taille croît sans cesse

Projection de l'historique

Définition

- Projection de l'historique $hist(e)$ d'un événement e sur le processeur P_i est l'ensemble :

$$hist_i(e) = \{e' \in hist(e) \mid e' \in P_i\}$$

- Sur la figure :

$$hist(e_{33}) = \{e_{11}, e_{21}, e_{22}, e_{23}, e_{24}, e_{25}, e_{31}, e_{32}, e_{33}\}$$

Les projections de $hist(e_{33})$, respectivement sur les processeurs P_1, P_2 et P_3, sont :

- $hist_1(e_{33}) = \{e_{11}\}$
- $hist_2(e_{33}) = \{e_{21}, e_{22}, e_{23}, e_{24}, e_{25}\}$
- $hist_3(e_{33}) = \{e_{31}, e_{32}, e_{33}\}$

Projection de l'historique

Propriété

Si $e_{i,k}$ événement de projection de l'historique de e sur P_i, alors tout événement antérieur à $e_{i,k}$ sur P_i est dans l'historique de e :

$$\exists k \mid e_{i,k} \in hist_i(e) \Longrightarrow \forall j < k : e_{i,j} \in hist(e)$$

Réduction de la représentation de l'historique

- Soit l'événement $e_{i,k}$, le plus récent des événements de l'historique de e sur P_i
- D'après la propriété, $\forall e_{i,k-j} \mid k > j > 0, e_{i,k-j} \in hist_i(e)$
- Il suffit donc de ne conserver que l'entier k pour caractériser la projection de l'historique de e sur P_i.
- Horloges vectorielles : un événement par processus

Représentation d'un historique par un vecteur

- $hist(e) = \bigcup_i hist_i(e)$: un vecteur $V(e)$, avec une valeur par processus, suffit pour représenter $hist(e)$
- Pour un système de n processus, le vecteur est défini de la manière suivante :

$$\forall i (1 \leq i \leq n) : V(e)[i] = k, \text{ tel que :}$$

$$e_{i,k} \in hist_i(e) \text{ et } e_{i,k+1} \notin hist_i(e)$$

- $V(e)[i]$: nombre d'événements de P_i connus de e

Exemple

Exemple précédent

Dans l'exemple précédent on a :

- $V(e_{33})[1] = 1$
- $V(e_{33})[2] = 5$
- $V(e_{33})[3] = 3$

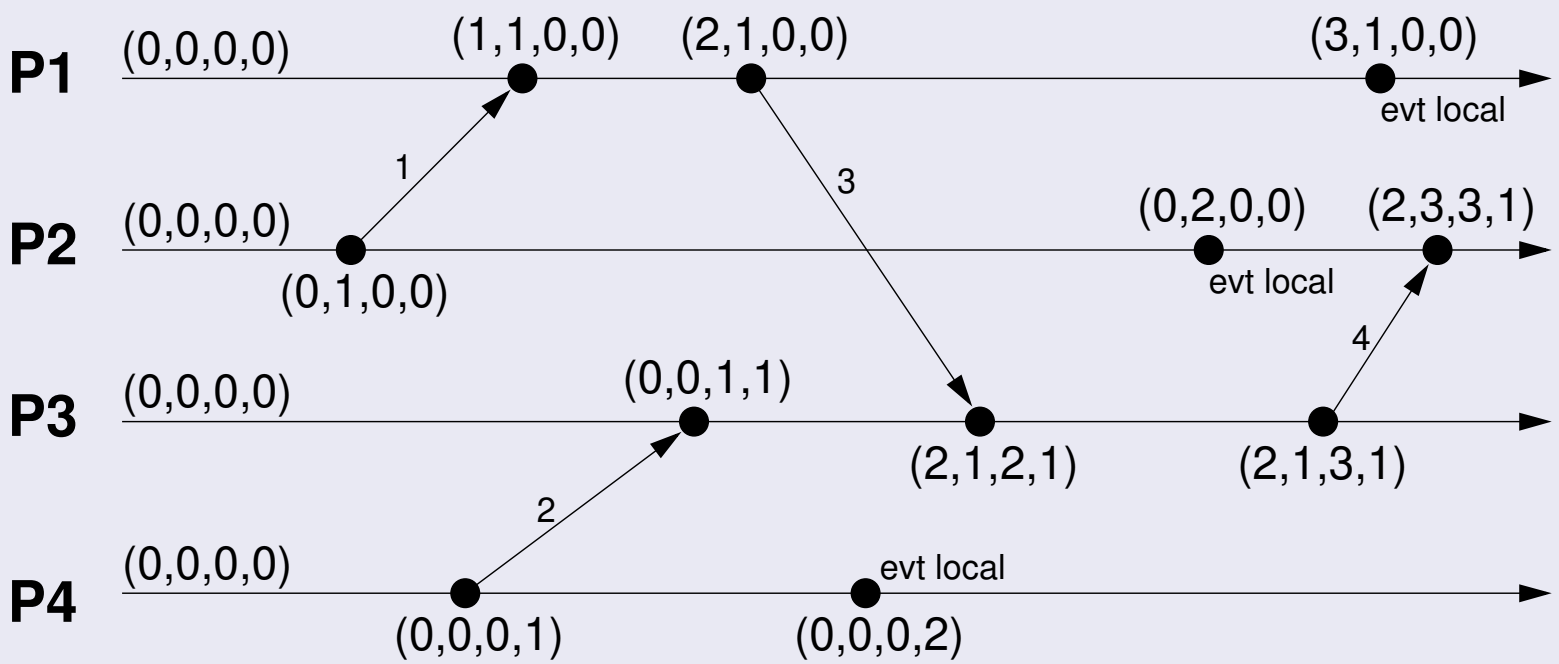
Les horloges vectorielles

Définition

- Un vecteur V_i de taille n est associé à chaque processus P_i
- Initialement $V_i = (0, \ldots, 0)$
- A chaque événement local à P_i, $V_i[i] \leftarrow V_i[i] + 1$
- A chaque émission, le message m porte l'estampille V_e (V_e est l'horloge vectorielle de l'émetteur P_e)
- A la réception de (m, V_e) par un processus P_i :
  - $V_i[i] \leftarrow V_i[i] + 1$
  - $V_i[j] \leftarrow max(V_i[j], V_e[j])$ pour $j = 1, \ldots, n, j \neq i$

Exemple d'exécution

Séquence d'événements sur 4 processeurs



Les horloges vectorielles

Propriété : ordre partiel

Soit deux horloges vectorielles V et V', il existe un ordre *partiel* entre ces 2 horloges :

$$V \leq V' \text{ si et seulement si } \forall i(1 \leq i \leq n) : V[i] \leq V'[i]$$

$$V < V' \text{ si et seulement si } (V \leq V') \wedge (V \neq V')$$

$$V \| V' \text{ si et seulement si } \neg(V \leq V') \wedge \neg(V' \leq V)$$

Les horloges vectorielles

Propriété : denses

- Soient e_i un événement du processeur P_i et e_j un événement du processeur P_j.
- Si $V(e_i)[k] < V(e_j)[k]$ alors il existe e_k tel que $\neg(e_k \preceq e_i) \wedge (e_k \preceq e_j)$.

Signification

Il existe forcément un événement e_k qui a permis l'incrémentation de la composante k de l'horloge sur le processeur P_k. Cet événement a eu lieu avant l'événement e_j sans qu'on puisse dire s'il s'est produit avant ou après l'événement e_i. La seule chose que l'on puisse affirmer, c'est que e_k n'est pas la cause de e_i.

Sommaire

Ordre causal : CBCAST

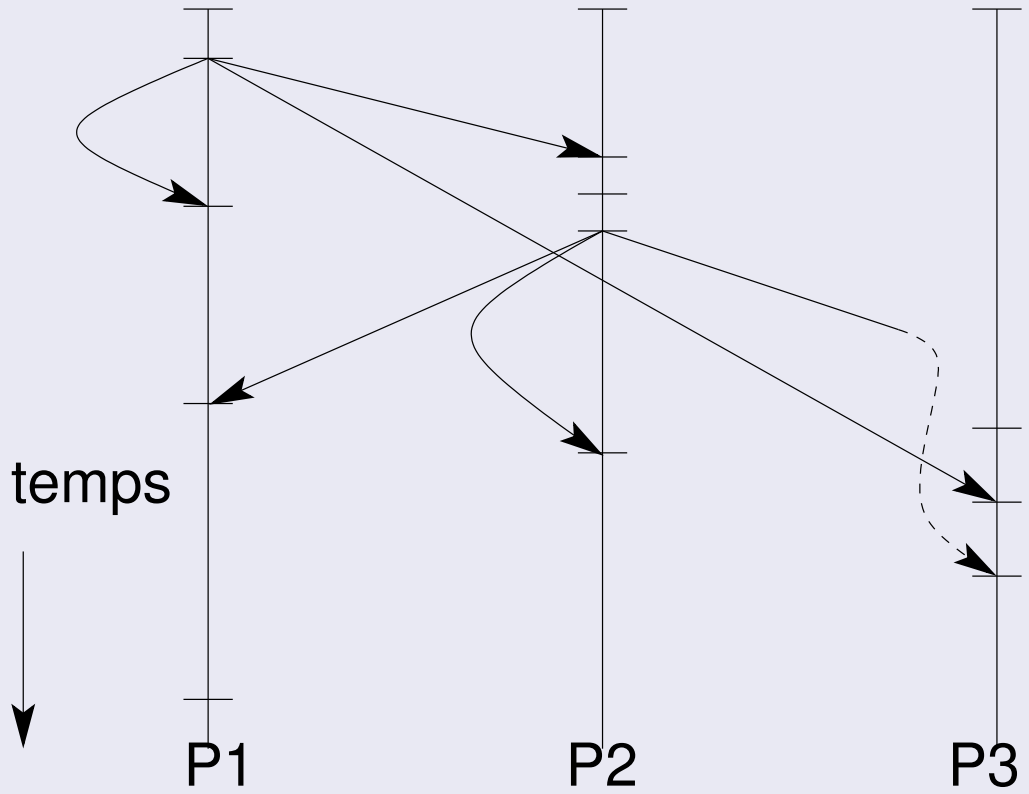
Définition

Si le message m est la cause du message m' (m' est envoyé après la délivrance de m sur le processus émetteur de m') alors tous les processus délivrent le message m' après le message m. Ceci peut s'écrire de la manière suivante :

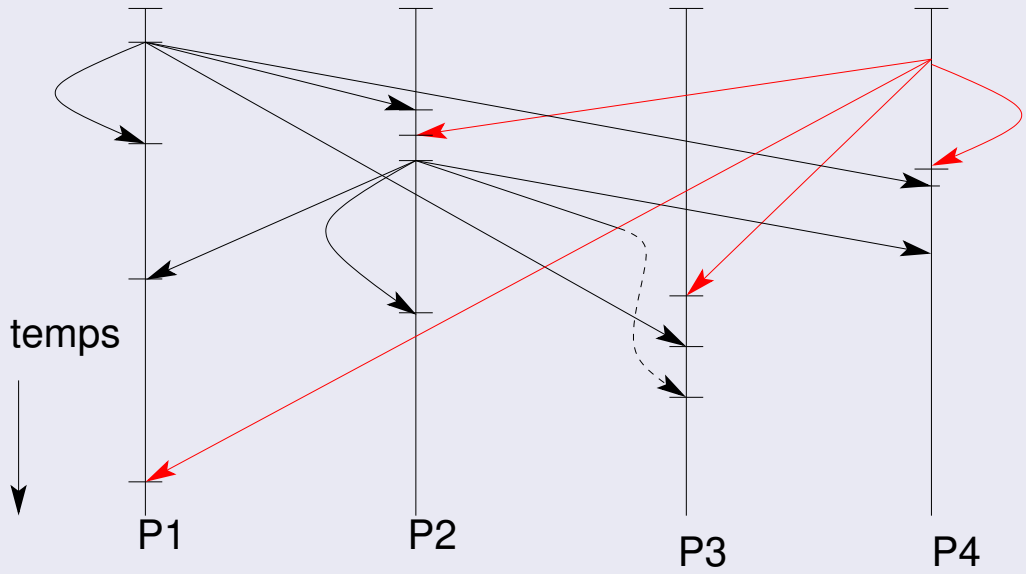
$$\text{Si } m \preceq m' \Longrightarrow \forall P \in G \text{ alors } \text{delivre}_p(m) \preceq \text{delivre}_p(m')$$

L'ordre de livraison des messages respecte la relation de causalité entre m et m'.

Ordre causal : CBCAST



Ordre causal : CBCAST



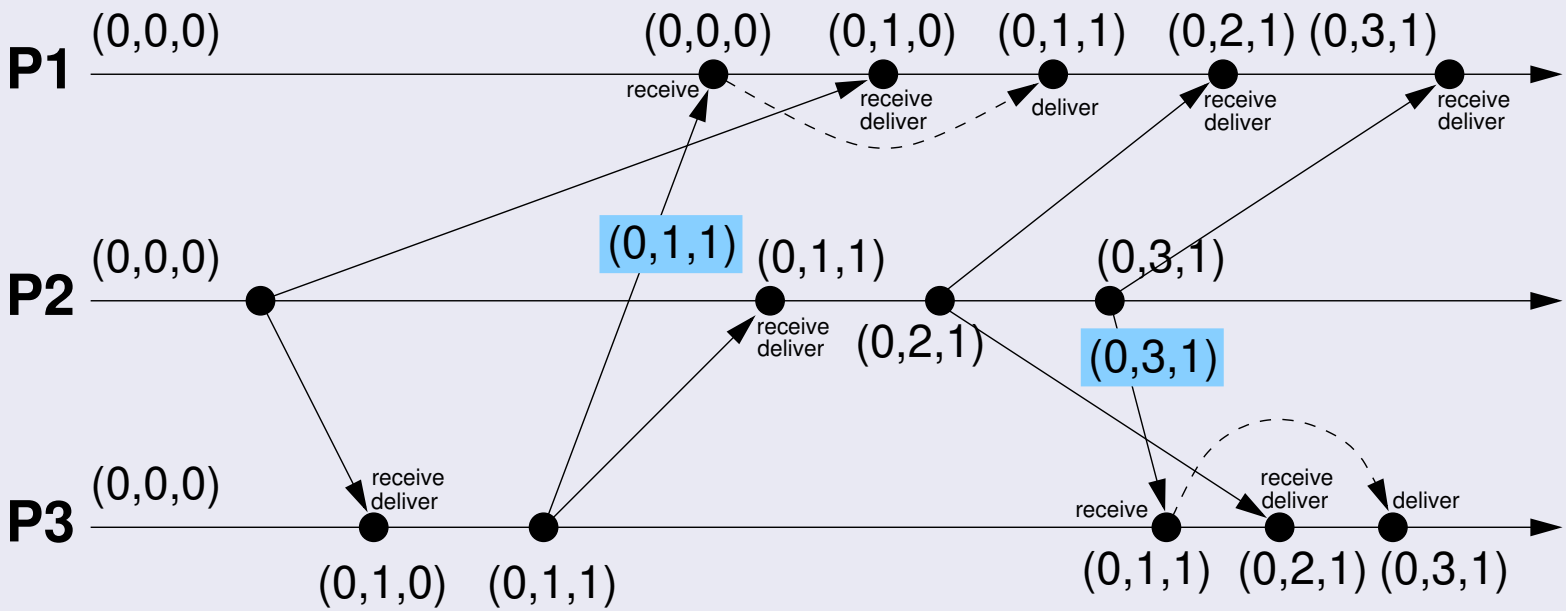
Le protocole CBCAST

CBCAST utilise l'historique :

1. Avant d'envoyer m, le processus P_i incrémente $V_i[i]$
2. Le processus P_i envoie le message m estampillé avec V_i
3. À la réception d'un message m estampillé par V_m en provenance de P_j, le processus $P_i \neq P_j$ diffère sa livraison jusqu'à ce que les conditions suivantes soient réalisées :
   - Si $i \neq j$, $V_i[j] = V_m[j] - 1$: le message qui arrive de i porte le numéro attendu et donc aucun message de j n'a été perdu.
   - Si $i = j$: le processus reçoit son propre message, la condition est $V_i[i] = V_m[i]$
   - $\forall k \in [1, n], k \neq j : V_i[k] \geq V_m[k]$, tous les messages précédents ont été reçus
4. Après remise de m : $V_i \leftarrow max(V_m, V_j)$

Le protocole CBCAST

Déroulement de diffusions fiables de type CBCAST



Protocole CBCAST

Exercices

- Dérouler l'algorithme pour le cas où :
  1. on a trois processus,
  2. le processus p_0 diffuse, la diffusion se termine
  3. le processus p_1 diffuse,
  4. le processus p_2 diffuse avant d'avoir reçu le message de p_1
  5. p_0 diffuse après avoir reçu le message de p_1,
  6. p_2 reçoit le message de p_0 avant celui de p_1
- Écrire l'algorithme sous forme de règles, en supposant qu'on dispose d'une file de messages en local (*put*, *get*)

Sommaire

Ordre atomique ou total : ABCAST

Définition

La relation d'ordre est étendue aux processus concurrents. En effet, si on a un groupe g, tous les processus p de ce groupe délivrent les messages dans le même ordre, quelque soit l'ordre l'émission des messages, d'où l'expression :

$\exists m, m'$ et $p \in g \mid$
$\text{delivre}_p(m) \preceq \text{delivre}_p(m') \Rightarrow \forall q \in g : \text{delivre}_q(m) \preceq \text{delivre}_q(m')$

Ordre atomique ou total : ABCAST

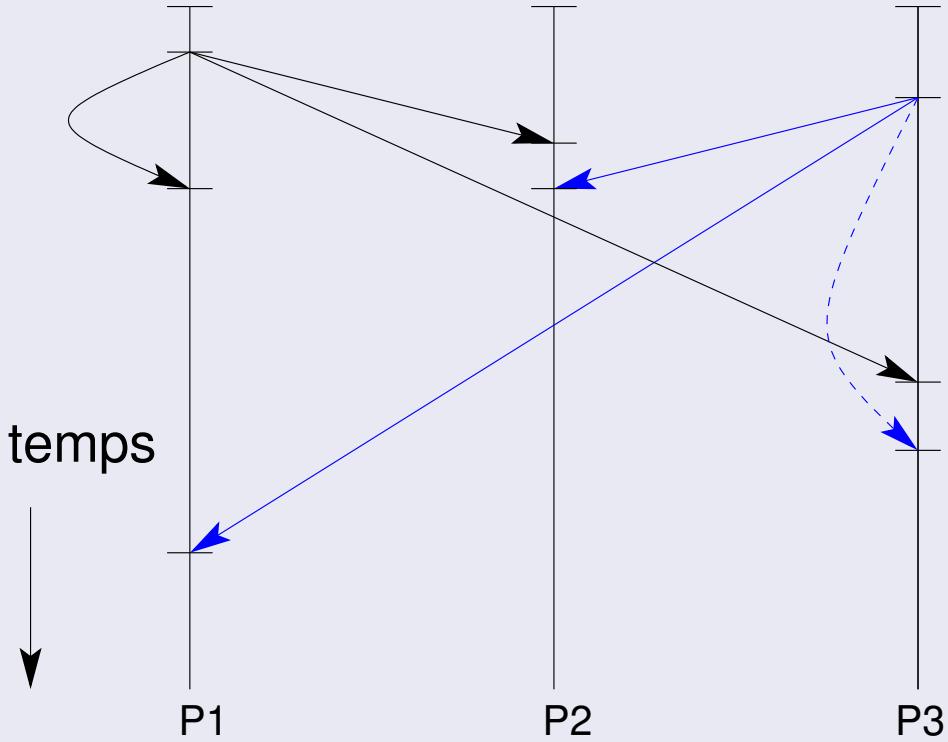


Figure – Exemple d'exécution correcte d'une diffusion fiable atomique

Ordre atomique ou total : ABCAST

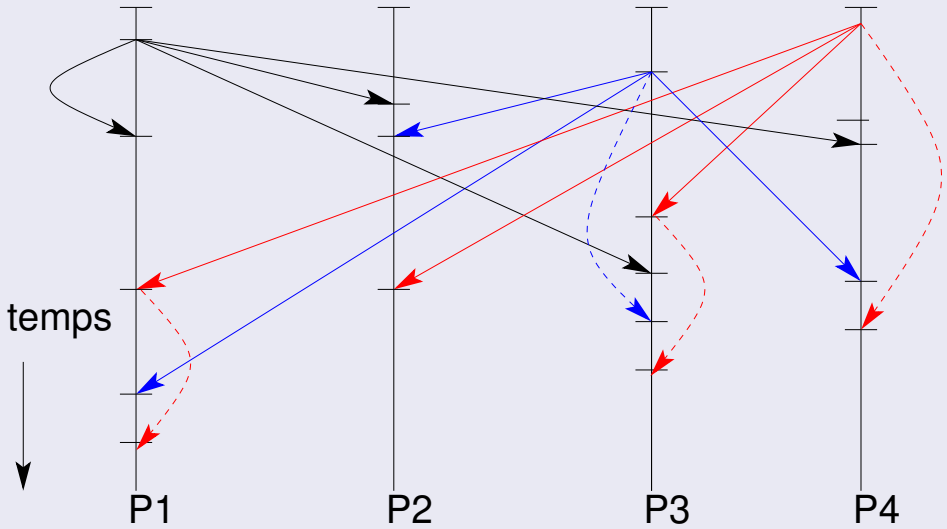


Figure – Exemple d'exécution correcte d'une diffusion fiable atomique

Le protocole ABCAST

Principe

- Pour respecter l'ordre total (ou atomique), on utilise un protocole de validation à deux phases.
- Le processus émetteur envoie d'abord le message auquel
- Les récepteurs associent une estampille provisoire qu'ils proposent à l'émetteur,
- Le processus émetteur valide ensuite la valeur de l'estampille définitive et la communique aux processus destinataires afin que le message puisse être délivré dans l'ordre des estampilles définitives.
- Protocole coûteux prend $3n$ messages.

Le protocole ABCAST

Protocole - 1ère Phase

- Soit g un groupe de n processus P_i avec $i = 1 \ldots n$.
- Lors d'une diffusion le processus P_i émetteur joint à la diffusion du message l'estampille provisoire : $\langle$`date d'émission.numéro de l'émetteur`$\rangle$ et l'envoie au membres du groupe
- A la réception, chaque destinataire P_j, y compris l'émetteur :
  - incrémente sa valeur locale d'estampille de 1,
  - met sa propre estampille au message reçu : $\langle$`estampille.numéro de processus`$\rangle$
  - marque le message en attente : *Pending*

Le protocole ABCAST

Protocole - 2nde Phase

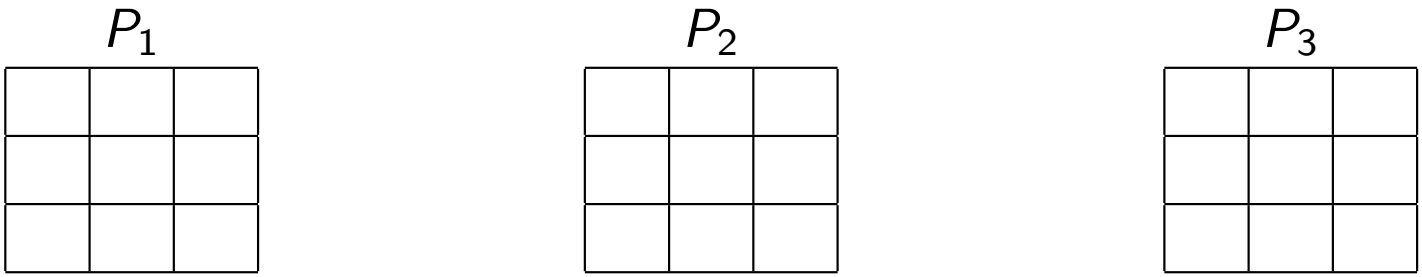
1. Chaque destinataire P_j renvoie l'estampille attribuée à l'émetteur P_i du message ;
2. Quand l'émetteur P_i a reçu toutes les réponses, il choisit la plus grande estampille $\langle$`date d'émission.numéro de l'émetteur`$\rangle$ comme estampille définitive puis envoie une validation avec cette estampille à tous les membres du groupe ;
3. A la réception de l'estampille définitive, le message est marqué validé : *Ready* ;
4. Les messages validés sont alors délivrés aux applications *dans l'ordre de leurs estampilles*.

Le protocole ABCAST

Exemple

- Diffusions fiables de type ABCAST au sein d'un groupe de trois processus
- Soit les processus P_1, P_2 et P_3 diffusant chacun respectivement les messages m_1, m_2 et m_3.
- L'état courant des estampilles de ces processus est 16 pour P_1, 14 pour P_2, 12 pour P_3 ;
- P_1 reçoit les messages dans l'ordre m_1, m_3 et m_2
- P_2 reçoit les messages dans l'ordre m_2, m_1 et m_3
- P_3 reçoit les messages dans l'ordre m_3, m_2 et m_1

Exemple ABCAST



Émission des messages m_1, m_2, m_3
A la réception des messages sur les processus, ils sont mis en attente et le protocole leur associe des estampilles provisoires.

Exemple ABCAST

P_1		
m_1		
17.1		
P		

P_2		
m_2		
15.2		
P		

P_3		
m_3		
13.3		
P		

Réception des messages m_1 en P_1, m_2 en P_2, m_3 en P_3

Exemple ABCAST

P_1

m_1	m_3	
17.1	18.1	
P	P	

P_2

m_2	m_1	
15.2	16.2	
P	P	

P_3

m_3	m_2	
13.3	14.3	
P	P	

Réception des messages m_1 en P_1, m_2 en P_2, m_3 en P_3
Réception des messages m_3 en P_1, m_1 en P_2, m_2 en P_3

Exemple ABCAST

P_1

m_1	m_3	m_2
17.1	18.1	19.1
P	P	P

P_2

m_2	m_1	m_3
15.2	**16.2**	17.2
P	P	P

P_3

m_3	m_2	m_1
13.3	14.3	**15.3**
P	P	P

Réception des messages m_1 en P_1, m_2 en P_2, m_3 en P_3
Réception des messages m_3 en P_1, m_1 en P_2, m_2 en P_3
Réception des messages m_2 en P_1, m_3 en P_2, m_1 en P_3

Exemple ABCAST

P_1

m_1	m_3	m_2
17.1	18.1	19.1
P	P	P

P_2

m_2	m_1	m_3
15.2	**16.2**	17.2
P	P	P

P_3

m_3	m_2	m_1
13.3	14.3	**15.3**
P	P	P

Réception des messages m_1 en P_1, m_2 en P_2, m_3 en P_3
Réception des messages m_3 en P_1, m_1 en P_2, m_2 en P_3
Réception des messages m_2 en P_1, m_3 en P_2, m_1 en P_3
A chaque réception, les processus envoient un message vers le processus émetteur avec l'estampille provisoire du message.

Exemple ABCAST

P_1

m_1	m_3	m_2
17.1	18.1	19.1
R	P	P

P_2

m_2	m_1	m_3
15.2	**17.1**	17.2
P	R	P

P_3

m_3	m_2	m_1
13.3	14.3	**17.1**
P	P	R

Le processus P_1 choisit la plus grande estampille ($e = \max\{17.1, 16.2, 15.3\} = 17.1$) comme définitive et la diffuse. A sa réception les messages sont notés prêts (R) à être délivrés.
Localement, ils sont réordonnés en fonction des nouvelles estampilles pour être délivrés dès qu'ils auront la plus petite estampille locale.

Le message m_1 est délivré sur le processus P_1.

Exemple ABCAST

P_1

m_3	m_2	
18.1	**19.1**	
P	P	

P_2

m_2	m_1	m_3
15.2	17.1	17.2
P	R	P

P_3

m_3	m_2	m_1
13.3	14.3	17.1
P	P	R

L'estampille définitive pour m_2 est calculée (19.1) et est diffusée à tous les processus.

Exemple ABCAST

P_1

m_3	m_2	
18.1	**19.1**	
P	R	

P_2

m_1	m_3	m_2
17.1	17.2	**19.1**
R	P	R

P_3

m_3	m_1	m_2
13.3	17.1	**19.1**
P	R	R

Le message m_1 est délivré sur le processus P_2

Exemple ABCAST

P_1

m_3	m_2	
18.1	19.1	
P	R	

P_2

m_3	m_2	
17.2	19.1	
P	R	

P_3

m_3	m_1	m_2
13.3	17.1	19.1
P	R	R

L'estampille définitive pour m_3 est calculée (18.1) et est diffusée à tous les processus.

Exemple ABCAST

P_1

m_3	m_2	
18.1	19.1	
R	R	

P_2

m_3	m_2	
18.1	19.1	
R	R	

P_3

m_1	m_3	m_2
17.1	**18.1**	19.1
R	R	R

Les message m_3 puis m_2 peuvent être délivrés sur les processus P_1 et P_2
Les message m_1, m_3 puis m_2 peuvent être délivrés sur le processus P_3

Protocole ABCAST

Exercices

- Dérouler l'algorithme pour le cas où :
  1. On a quatre processus,
  2. L'état initial est 8 pour P_0, 10 pour P_1, 6 pour P_2 et 7 pour P_3
  3. Le processus P_1 diffuse un message qui est traité par l'ensemble des membres du groupe,
  4. Les processus P_0 et P_3 diffusent ensuite un message en même temps,
  5. le processus P_2 diffuse un message alors que les deux messages précédents ont été reçus mais non validés (fin de première phase).
- Écrire l'algorithme sous forme de règles.