UNIVERSITÉ DE STRASBOURG

cnrs

Systèmes Distribués

Introduction, Horloges & Cohérence

Pascal Mérindol (CM)
Benoit Sonntag (CM + TP)
Antoine Gallais (TD)

merindol@unistra.fr
http://www-r2.u-strasbg.fr/~merindol/

Plan & Organisation

- **La pratique : RPC, RMI, CORBA, SOAP => en TP (+ Projet avec B. Sonntag)**
- **Introduction aux Systèmes Distribués**
- **La «théorie» (en CM/TD) :**
  - Horloges & Diffusion/Partage
  - Exclusion Mutuelle & Inter-Blocages
  - Ordonnancement
- **Divers :** tolérance aux pannes, intro. sécurité, consensus, etc.

- **Pascal Mérindol CM, Antoine Gallais TD, Benoit Sonntag (CM/)TP :**
  - 50% CC2, 25% CC1, 25% Projet.
  - 9*2h TP, 6*2h TD, 10*2h CM.

Liens utiles (& refs du cours)

- **https://robinet.u-strasbg.fr/enseignement/sd + moodle**
- **http://iut-info.unistra.fr/~gancars/enseignement/index_enseignement.htm**
- **http://www.pps.jussieu.fr/~rifflet/enseignements/AlgoProgSysRep/**
- **http://www.informatics.sussex.ac.uk/courses/dist-sys/node1.html**
- **http://code.google.com/intl/fr-FR/edu/parallel/**

- G. F. Coulouris, J. Dollimore, T. Kindberg. ``Distributed Systems -- Concepts and Design". Ed. Addison-Wesley, 4th Edition. 2005.

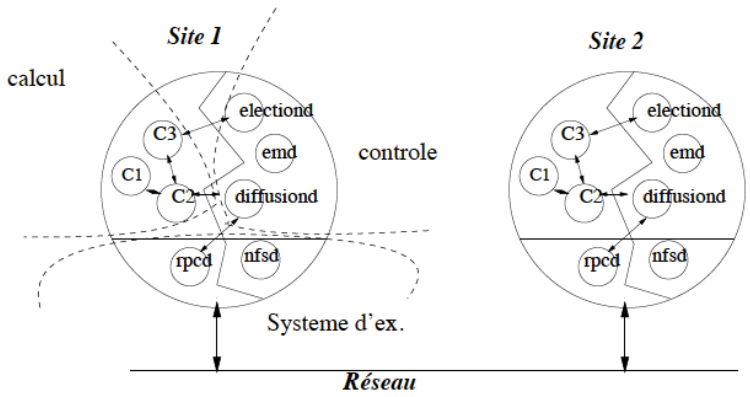
Définition(s)

- Assurer la coopération d'un ensemble de processus dans un environnement distribué
- Services rendus aux applications réparties (multi-sites)
  - communications inter-processus
  - partage des ressources physiques sous jacentes
- Mécanismes nécessaires :
  - transfert, partage et reconnaissance d'informations,
  - contrôle de cohérence, diffusion, synchronisation, ordonnancement,
  - contrôle global du système, élection, exclusion mutuelle, reprise sur panne,
  - transactions sécurisées, administration, etc ...
- Objet du cours : les systèmes faiblement couplés

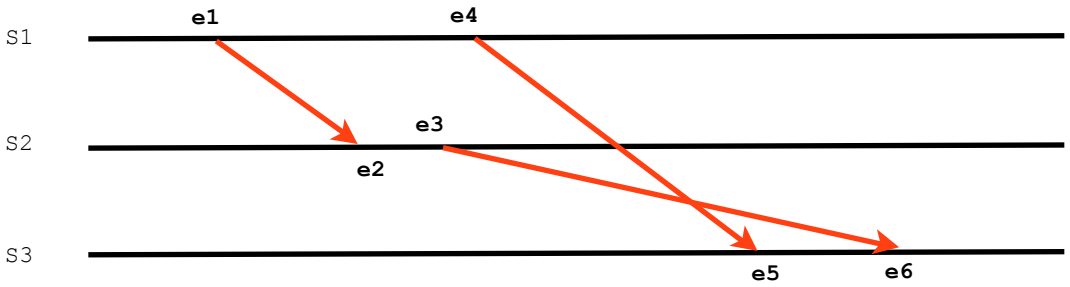
Principes de base

- Pas d'état global
  - les processus sont «égo-centrés» : pas de connaissance ni d'ordre strict global !
- Les données sont distribuées
  - duplication : cohérence ?
  - partitionnement multi-site : où retrouver et comment désigner l'info ?
- Pas de contrôle global
  - pas de hiérarchie type processus maitre...(robustesse du système)
  - ...ou élection pour des taches spécifiques (reprise sur panne)
- Comment «s'en sortir» dans un tel environnement ?

Briques de base



- Protocole
  - Comportements et règles inter-processus
- Processus / sites
  - dit de calcul (liés à l'application)
  - dit de contrôle (inter-agissent avec le système d'exploitation)
- Liaisons logiques : *le graphe de communication*
  - quelle structure ? anneau, étoile, arbre, ...
  - quelle quantité de messages ? combien de pertes tolérées ?
  - quel comportement ? FIFO..?

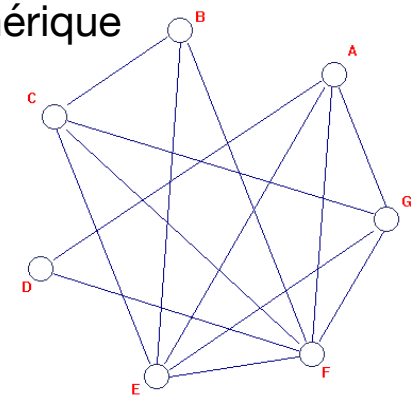


- FIFO ne veut pas dire que e4 est antérieur à e3 vu de S3

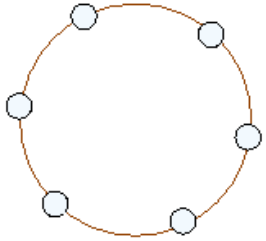
Aspects réseau

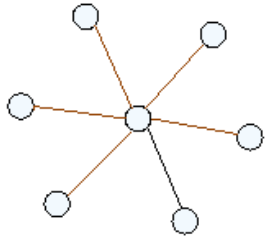
Graphe & topologies

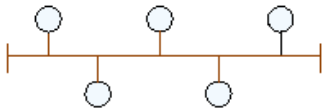
modèle générique

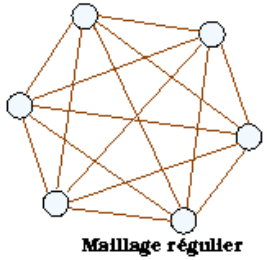


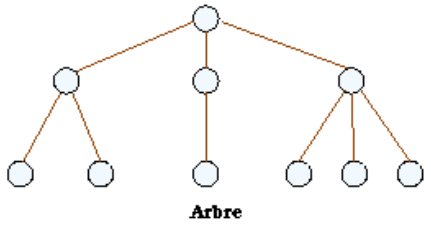
modèles spécifiques











Robustesse aux pertes

Soit *p* la probabilité de perte d'un msg et *n* le nb de msg nécessaire à un protocole.

La probabilité *P(X=k)* de perte de *k* msg est alors de $C_n^k p^k (1-p)^{n-k}$

La probabilité que le protocole aboutisse est de $P(X=0) = (1-p)^n$

ex : n=1000 messages et $p=10^{-3}$ alors ~2/3 d'aboutir du premier coup seulement...

Combien de tentatives *t* pour assurer au moins une probabilité *x* de réussite (*1-x* : échec) ?

La probabilité d'échec après *t* tentatives est de $(1-P(X=0))^t$ donc $(1-P(X=0))^t < 1-x$

ex : pour x=0.99 (et situation identique à l'ex. précédent) on a t=10

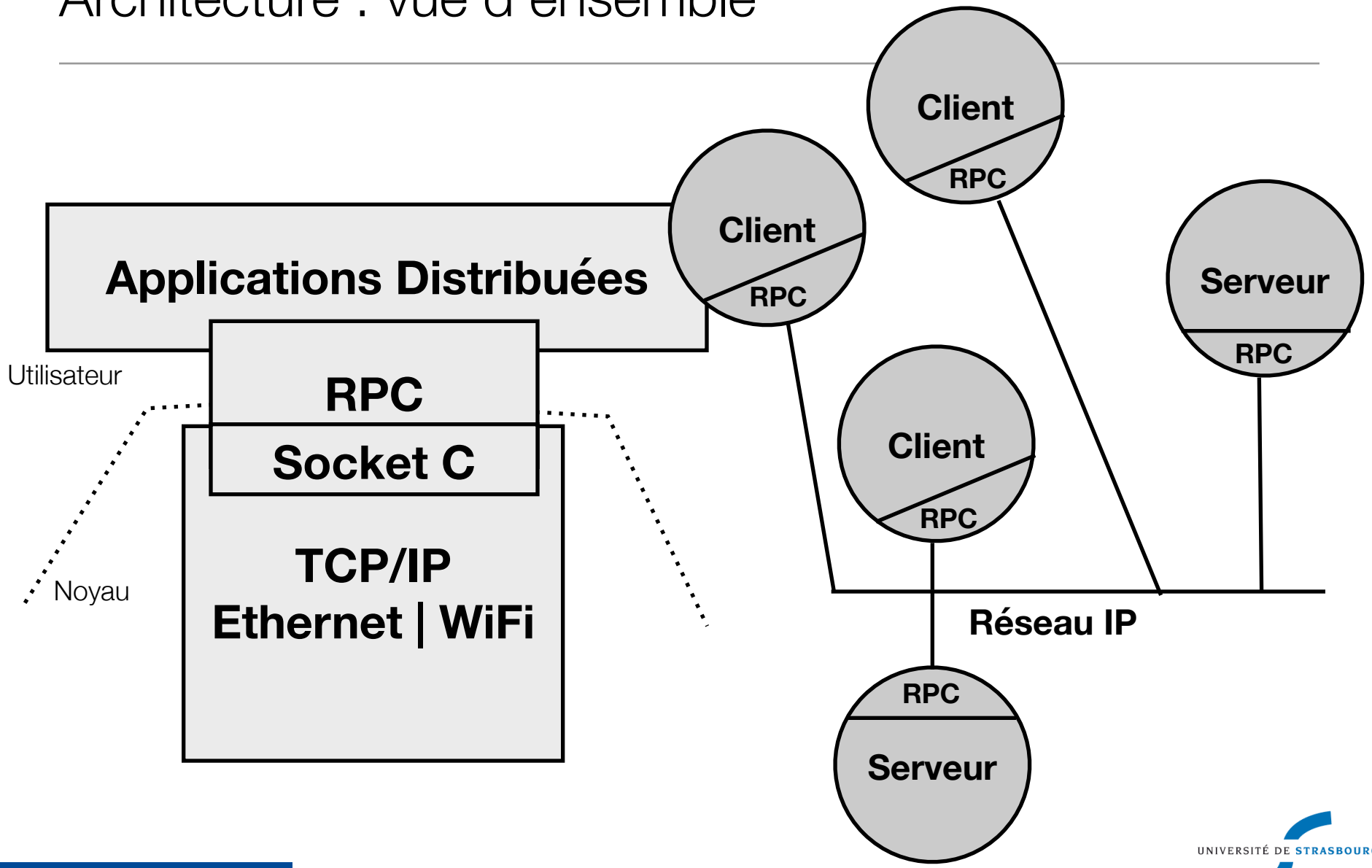
Les protocoles

- Calcul diffusant

  ➡ le droit d'émettre initialement détenu par une racine est diffusé, souvent via un arbre diffusant : messages diffusés de père en fils et calcul de «bas en haut»

- Jeton circulant
  - privilège circulant autour d'un anneau logique ou physique
- Estampillage
  - horloge logique
  - horloge vectorielle & causalité

Architecture : vue d'ensemble

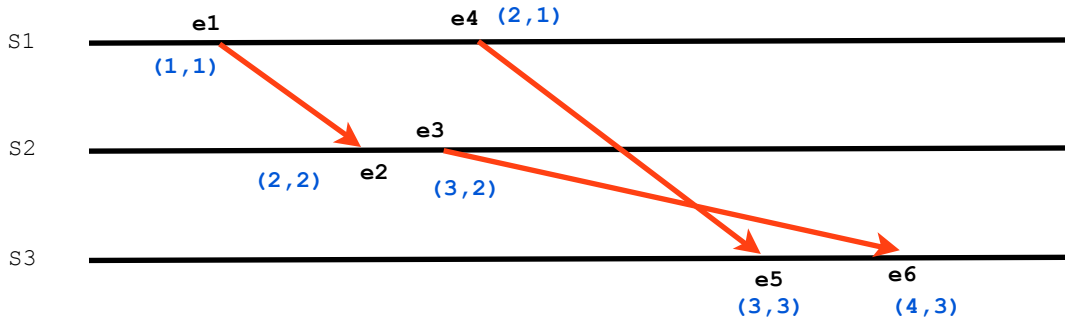


Plan : contenu

- **Partage des données & Diffusion**
- **(Exclusion mutuelle) & Inter-blocages**
- **Ordonnancement**
- **Tolérance aux pannes & Sécurité**
- **Consensus**

Horloge logique

- Objectif : Définir un ordre global strict
- Principe (Lamport, 1978)
  - horloge locale à chaque site i : H_i
  - ni envoi, ni réception : incrémenter l'horloge locale, $H_i = H_i + 1$
  - envoi sur i : incrémenter l'horloge locale + estampiller le msg (H_i, i)
  - réception d'un msg (H_j, j) sur i : $H_i = \max(H_i, H_j) + 1$



e4 => e3,
e4 dépend de e3 ?

- Et la causalité / indépendance des processus ?

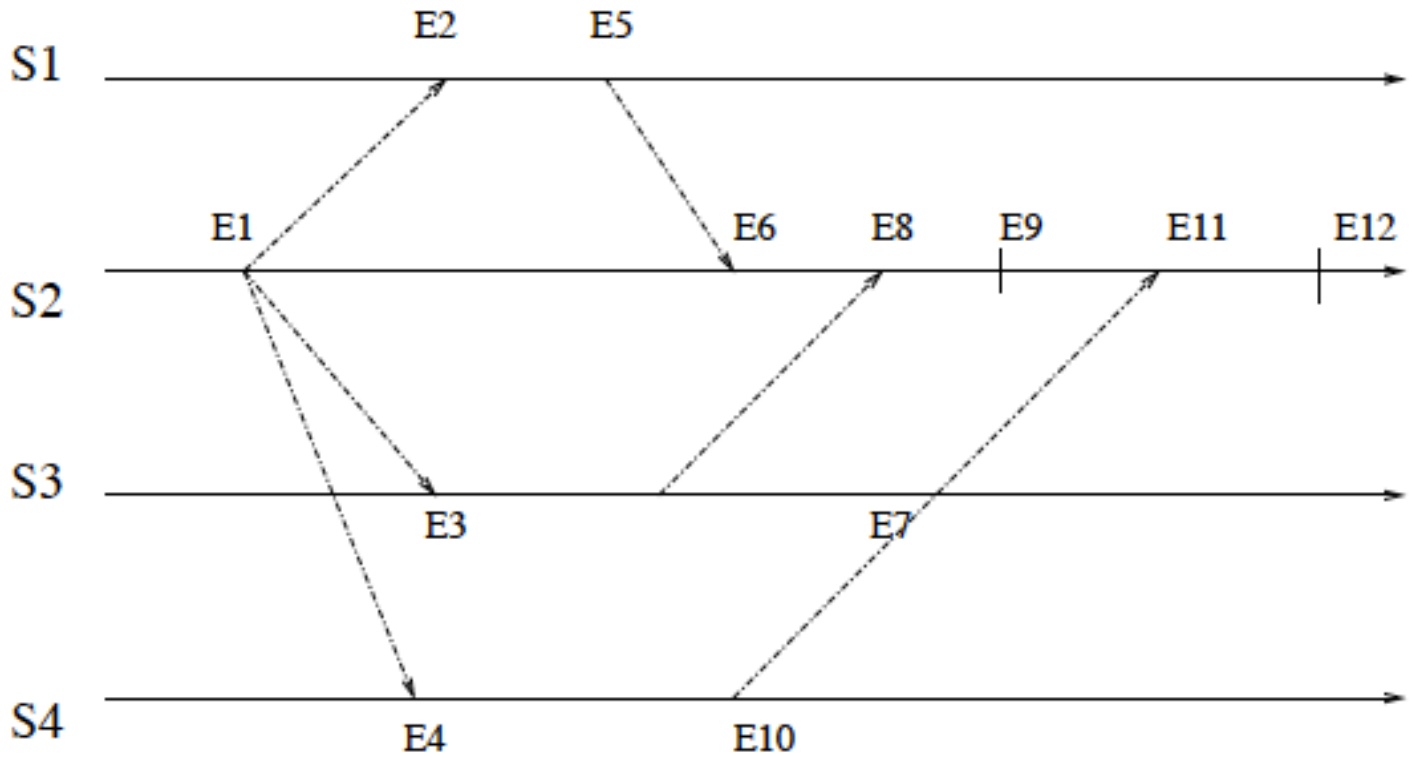
Horloge vectorielle

- Principe (avec n sites) :
  - chaque site i dispose d'un vecteur $V_i = [1,...,n]$, initialement à $[0,...,0]$
  - $V_i[i] = V_i[i] + 1$ à chaque événement
  - envoi sur i : le msg est estampillé par le vecteur courant de i
  - réception d'un msg estampillé V_j : $V_i[k] = max(V_i[k], V_j[k])$ pour $k = 1...n$
  - Chaque événement est estampillé avec le vecteur courant
    - Causalité entre deux évènements i et j ssi ∀k :

$$V_i[k] <= V_j[k] \text{ ou } V_i[k] >= V_j[k]$$

    - indépendance sinon !
      - Notion de «précédence causale»

A vous de jouer



- Donnez les estampilles issues d'un système avec horloge logique
- Donnez les vecteurs issus d'un système avec horloge vectorielle/causale
- Donnez le graphe de «précédence»

Données distribuées

- Désignation / Indexation / Recherche des infos
- Cohérence / Consistance des infos dupliquées

Désignation / Nommage

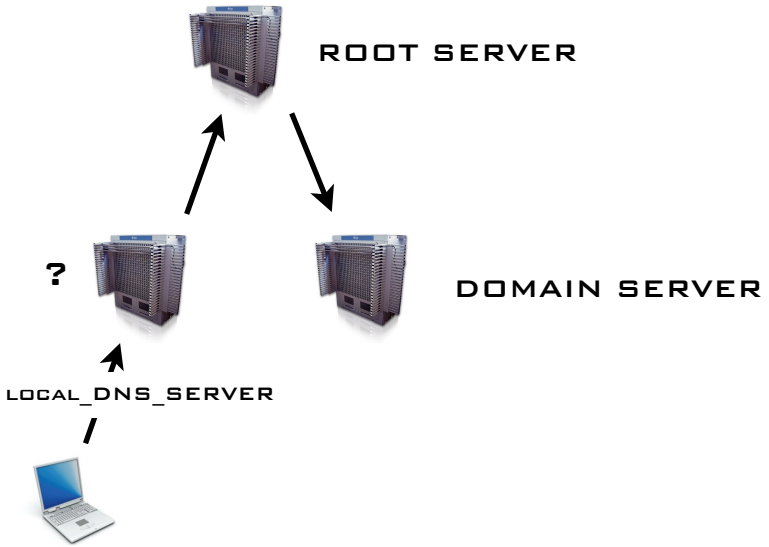
- Nommage d'un objet/fichier
  - nom symbolique (couche haute) / nom interne (couche basse)
  - Le nommage est le «mapping» de l'un vers l'autre
  - unicité d'un nom ! (+ penser à sa recherche et à sa robustesse face aux contrefaçons)
  - dans un contexte réparti, comment assurer l'unicité du nom ?

- Migration
  - définir le niveau de mobilité
  - peu mobile => liens de «poursuite» ?
  - très mobile => centralisation ?
    => diffusion ?

Désignation / Recherche

Mapping et recherche de noms

- **nom symbolique <= ? => nom interne**
  - manuellement ? :(((
  - via des fichiers de configuration ? :((
  - via des serveurs de désignation type NIS
  - via des serveurs de noms type DNS

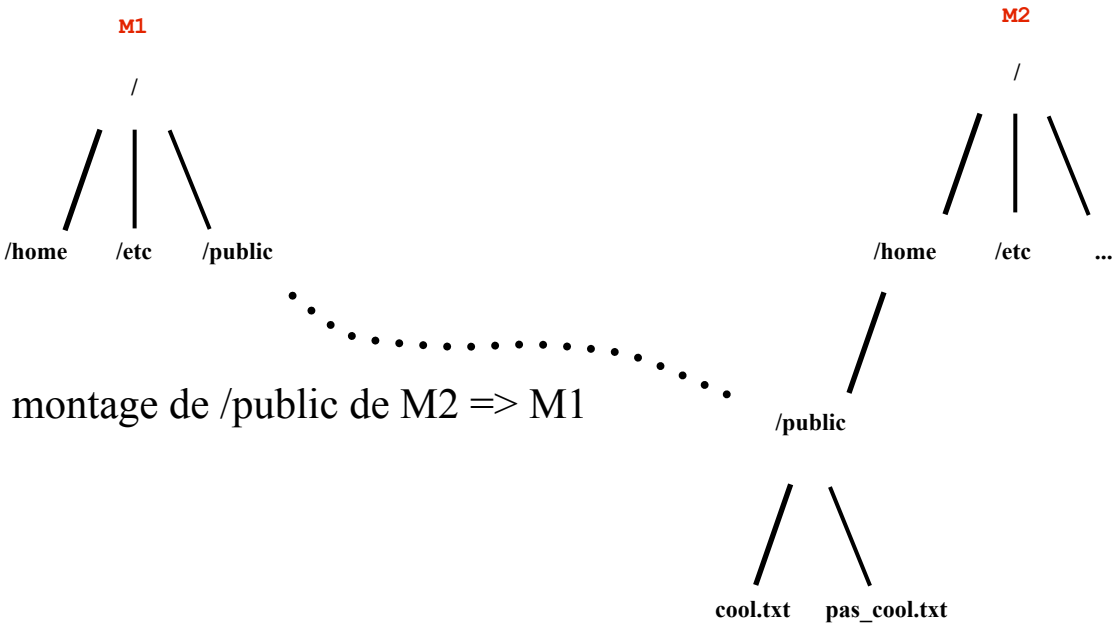


`dig +trace`

```
pascal@robinet: /home/pascal
pascal@robinet:~$  dig +trace wikipedia.org

; <<>> DiG 9.7.1-P2 <<>> +trace wikipedia.org
;; global options: +cmd
.                       424770  IN      NS      a.root-servers.net.
.                       424770  IN      NS      b.root-servers.net.
.                       424770  IN      NS      d.root-servers.net.
.                       424770  IN      NS      m.root-servers.net.
.                       424770  IN      NS      j.root-servers.net.
.                       424770  IN      NS      l.root-servers.net.
.                       424770  IN      NS      k.root-servers.net.
.                       424770  IN      NS      i.root-servers.net.
.                       424770  IN      NS      e.root-servers.net.
.                       424770  IN      NS      f.root-servers.net.
.                       424770  IN      NS      c.root-servers.net.
.                       424770  IN      NS      h.root-servers.net.
.                       424770  IN      NS      g.root-servers.net.
;; Received 488 bytes from 130.79.200.1#53(130.79.200.1) in 1 ms

org.                    172800  IN      NS      d0.org.afilias-nst.org.
org.                    172800  IN      NS      b0.org.afilias-nst.org.
org.                    172800  IN      NS      b2.org.afilias-nst.org.
org.                    172800  IN      NS      a0.org.afilias-nst.info.
org.                    172800  IN      NS      a2.org.afilias-nst.info.
org.                    172800  IN      NS      c0.org.afilias-nst.info.
;; Received 433 bytes from 2001:dc3::35#53(m.root-servers.net) in 8 ms

wikipedia.org.          86400   IN      NS      ns0.wikimedia.org.
wikipedia.org.          86400   IN      NS      ns1.wikimedia.org.
wikipedia.org.          86400   IN      NS      ns2.wikimedia.org.
;; Received 143 bytes from 2001:500:48::1#53(b2.org.afilias-nst.org) in 8 ms

wikipedia.org.          3600    IN      A       208.80.152.2
;; Received 47 bytes from 91.198.174.4#53(ns2.wikimedia.org) in 29 ms
```

Désignation symbolique

- Principes avec les systèmes de fichiers
  - unification de multiple arborescences
  - montage logique (NFS) vs. super racine virtuelle (nom associé à la machine distante)



L'exemple NFS

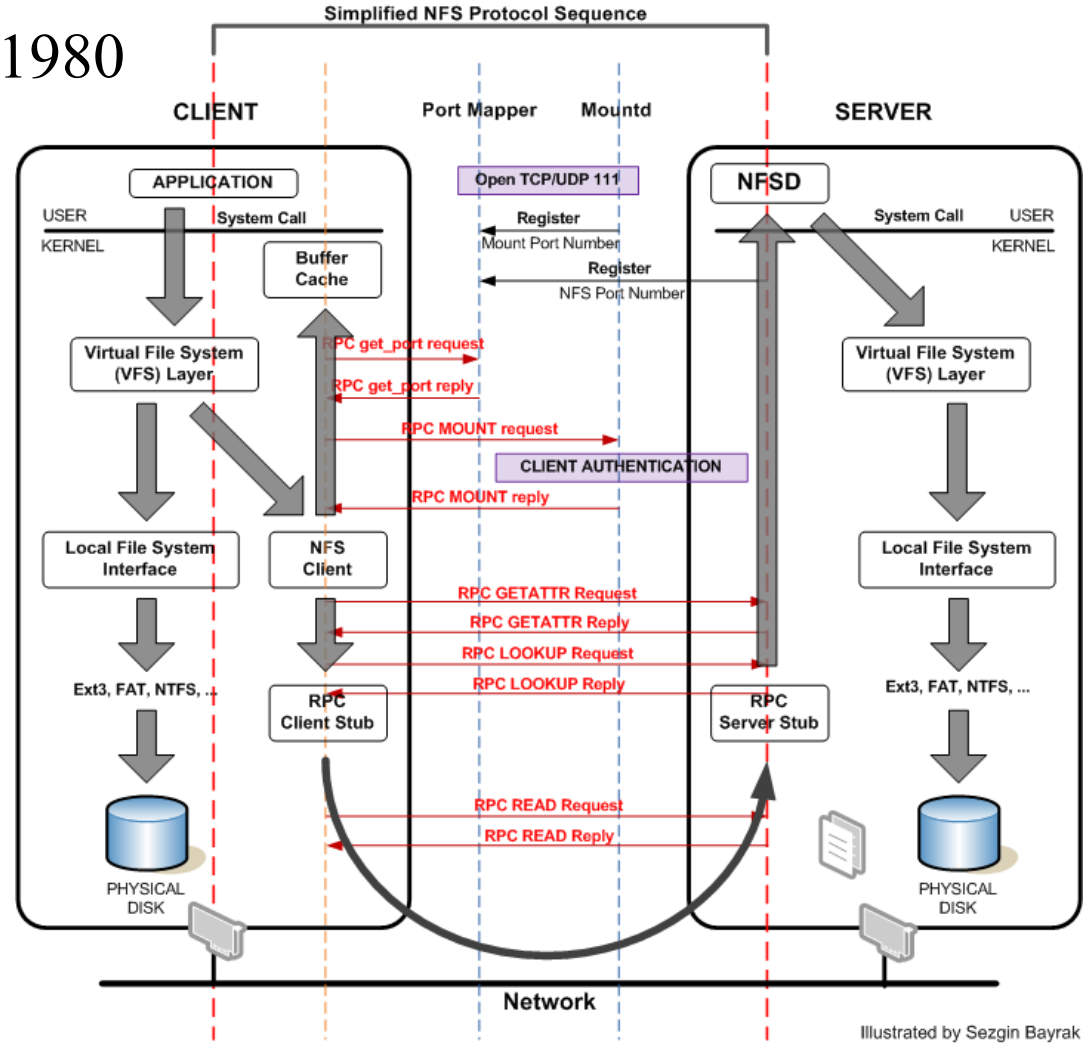
- Network File System, 1980

✓Système de désignation transparent pour les utilisateurs :)

✓Vitual i-node : nº machine + nº syst-de-fichiers + inode

✓Montage : `mount(remote_host, remote_directory, local_directory)`

✓Mapping du type : **<@IP,port,file>**



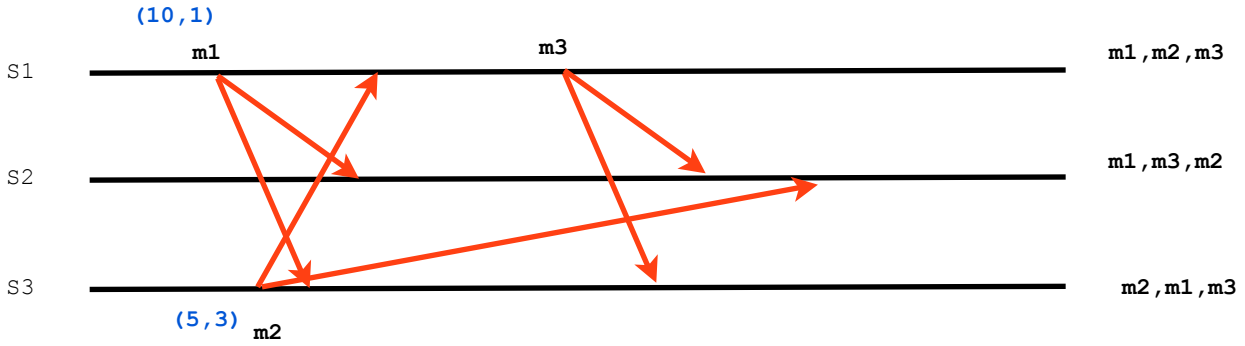
Illustrated by Sezgin Bayrak

NFS, quelques caractéristiques

- Optimisation avec cache
  - réduction du temps d'accès via des caches serveurs et clients (pages récentes)
  - validité d'un cache client : rafraichissement il y a moins de *t=3* sec ? (ou bien si la dernière date de mise à jour est synchro entre le serveur et le client)
  - Solution acceptable en pratique malgré les risques d'incohérence
- Serveur sans états
  - Les clients font le boulot ... => bonne tolérance aux pannes :) mais qqs inconvénients aussi
- Droits d'accès
  - basé sur l'**uid** mais sans unification globale (NIS ?)
- NFS est plus souple qu'un système avec état tel que AFS

Cohérence

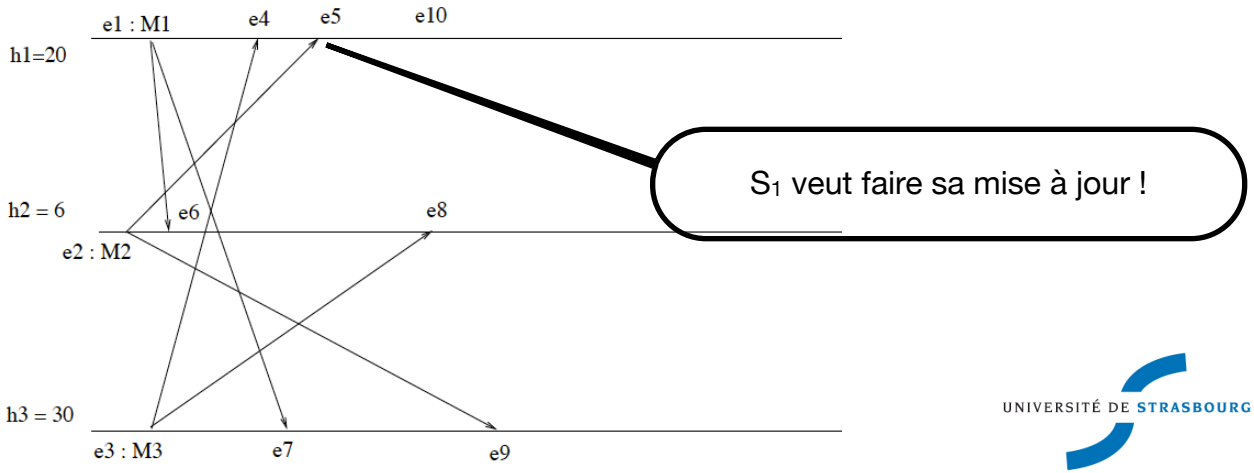
- Comment gérer la consistance des données dupliquées
  - copies explicites ou dans des caches
- Trois niveaux de cohérences pour les mises à jour
  - faible : dans un tps fini sans vérification de l'ordre (opérations «commutatives» !)
  - forte : vérification de l'ordre globale strict (on «retarde» la délivrance des messages)
  - causale : vérification de la causalité (au sens ordre, pas précédence)



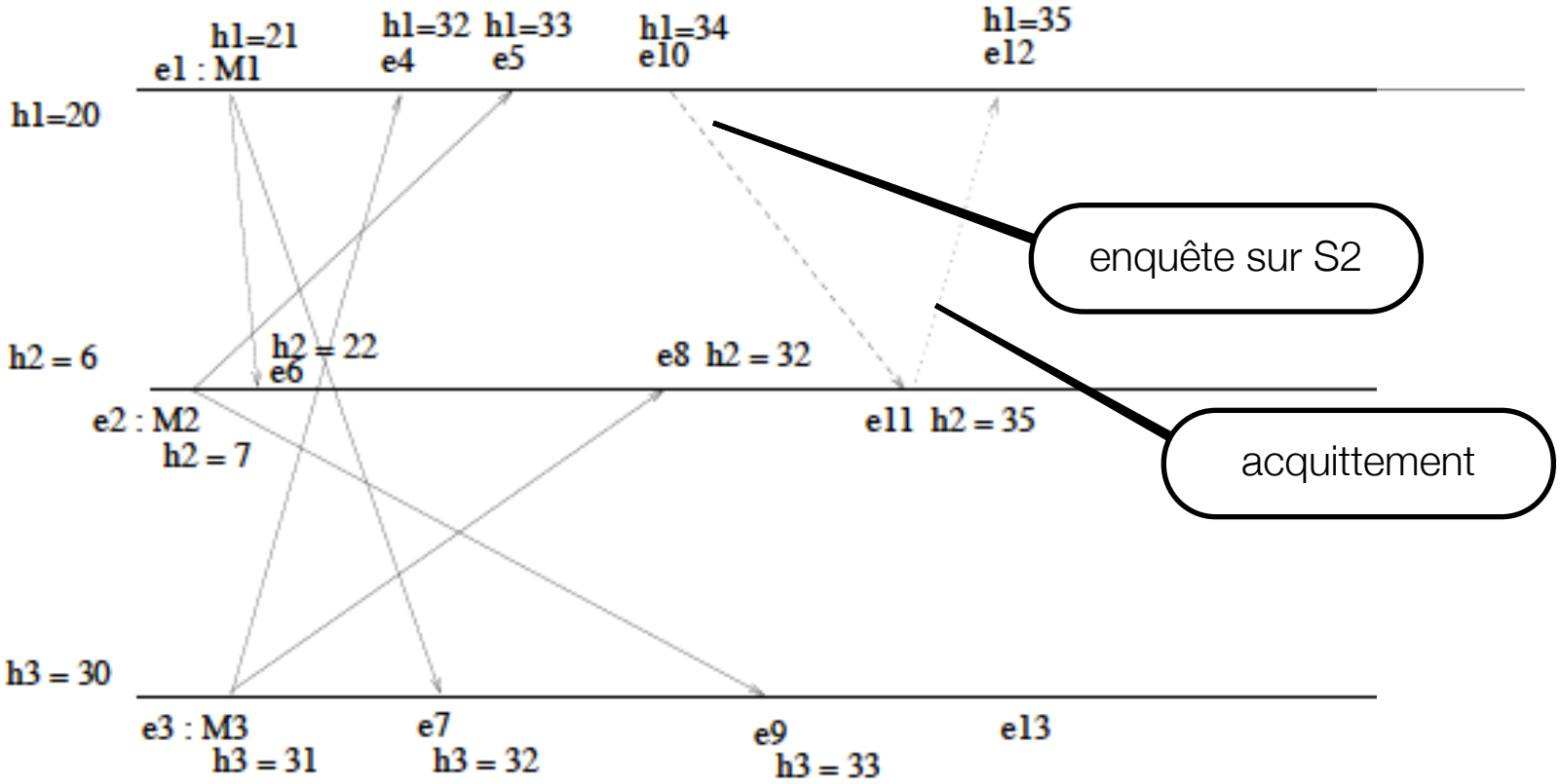
- Quels ordres pour une cohérence forte et causale ici ?

Maintenir la Cohérence !

- Exclusion mutuelle ?
  - centralisé, avec jeton, par autorisation ?
  - techniques assez lourdes et pas toujours nécessaires...(indépendance des modifs ?)
- ou Maintien de la cohérence «à la carte»
  - la mise à jour se fait selon les besoins de chaque site
  - les evts sont datés pour respecter l'historique des m.à.j antérieures



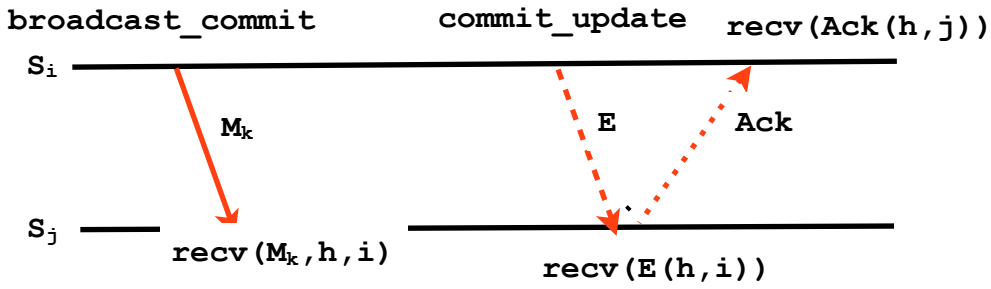
Maintenir la Cohérence !



- Principe algorithmique
  - chaque site dispose d'un tableau *F[j]* de piles contenant les msgs de chaque site *j*
  - diffusion et datage pour alerter des modifs
  - parcours des piles de message dans *F* et exécution des modifs dans l'ordre
  - quand une liste est vide, il faut s'assurer qu'il n'y a pas de modifs antérieures en cours...

Un premier algorithme (suppose FIFO)

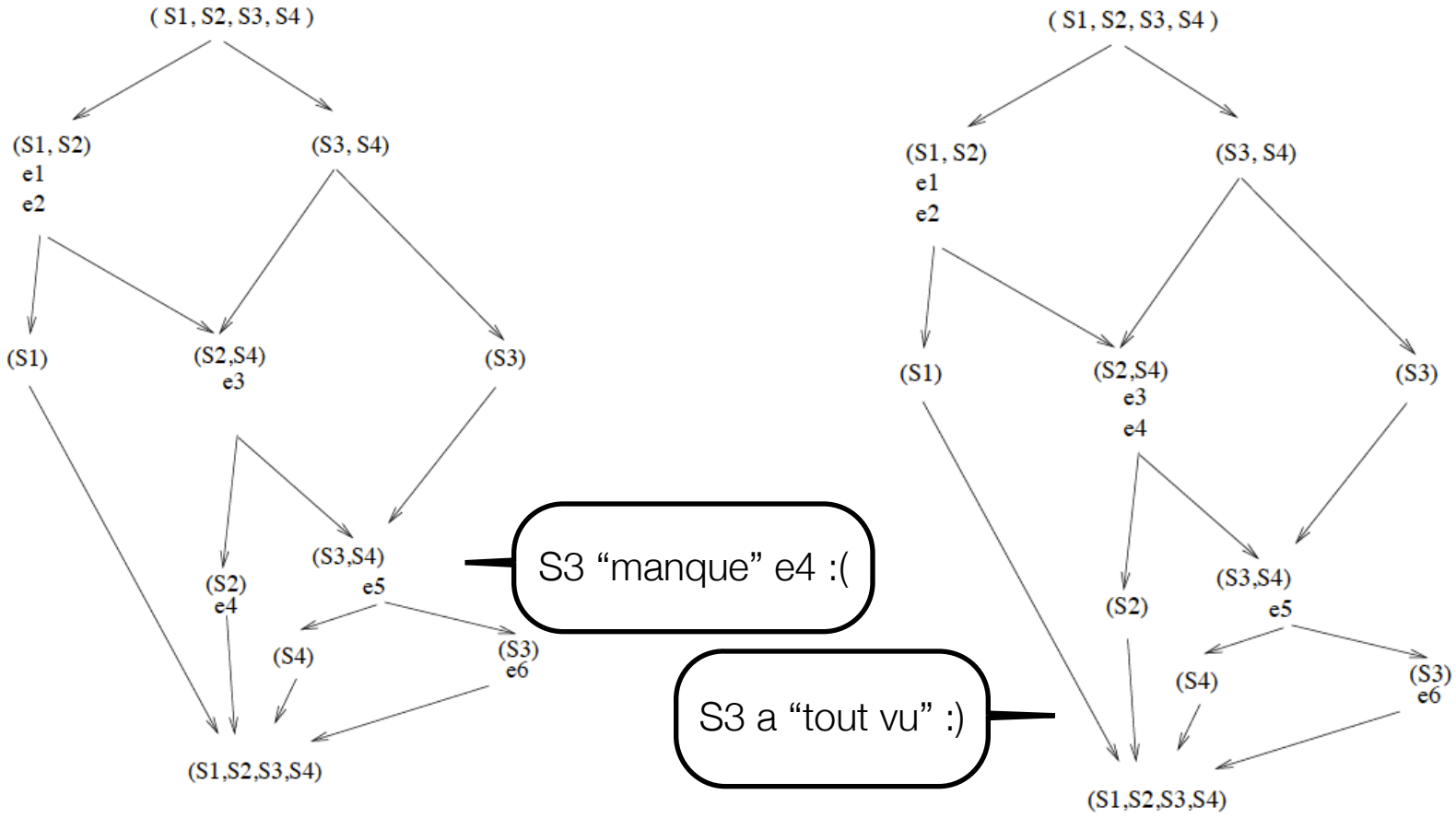
```
broadcast_commit(Mk){ /* on Si */
Hi++;
d(Mk)= Hk = Hi; /* d() ~ date */
broadcast (Mk,Hk,i);
F[i] ← Mk(Hk, i);  /* push Mk in F[i] */ }

recv(M,h,i){ / * on Sj */
Hj = max(h,Hj)+1;
F[i] ← M(h,i);  }

commit_update(Mk(Hk,i)){ / * on Si */
do{
check_list(Mk(Hk,i)); /* check updates to commit before Mk */
for all j ∈ V={j | F[j] == ∅} { /* empty sub-list */
send E(Hi,i) to Sj; /* E = Examine*/ }
}
while(V ≠ ∅) /* wait for a ack (or any msg) of each Sj ∈ V */
commit Mk;            }

check_list(Mk,Hk,i){ / * locally on Si */
do{
W = {all j | d(b(F[j])) < (Hk,i)};  /* all updates to commit before */
If (Type(b(F[j])) == ``commit'') {commit F[j];} /* b() ~ bottom */
pop(F[j]);}
while(W ≠ ∅)       }

recv(E(h,i)){ / * Examine msg received on Sj */
Hj = max(h,Hj)+1;
send Ack(Hj,j) to Si;  }
```




```
recv(Ack(h,j)){ / * on Si */
Hi = max(h,Hi)+1;
F[j] ← Ack(h, j);}
```

Détection de l'incohérence : incohérence mutuelle

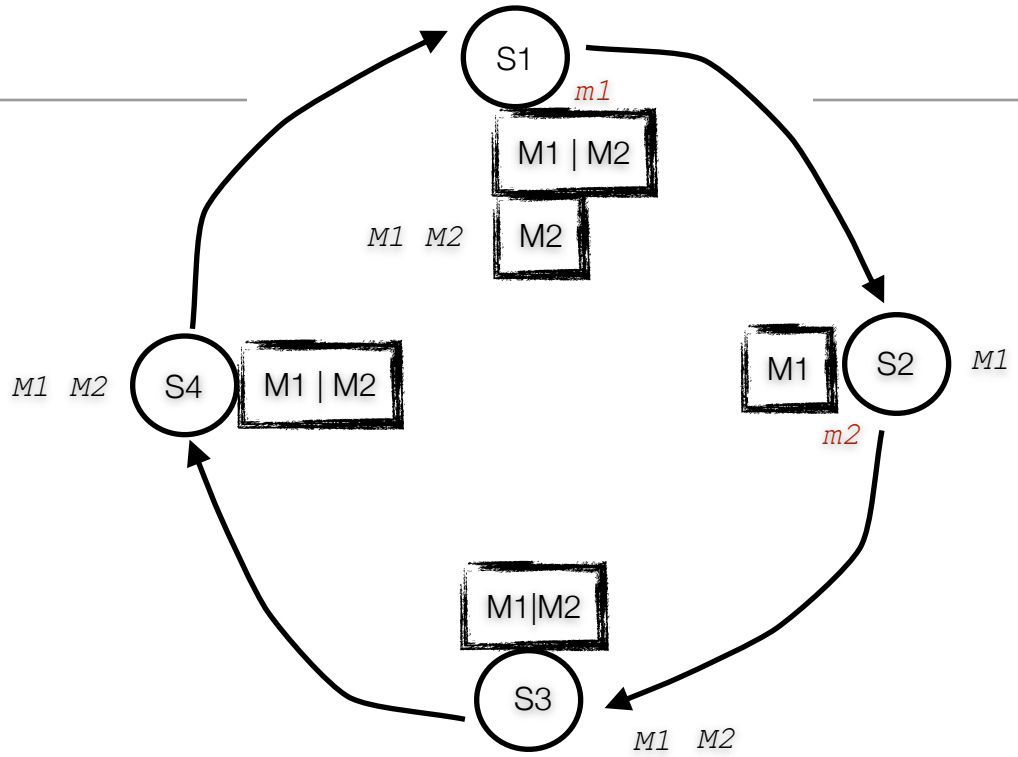
- Partitionnement : sous-réseaux n'assurant plus qu'une cohérence locale
- Comment assurer la cohérence globale après re-connexion ?



Diffusion

- Assurer la fiabilité de la diffusion
  - pas de perte de message
  - tout les sites voient la même chose
  - mais pas d'ordre supposé...
  - on parle de diffusion atomique si :
    - ✓l'ordre de délivrance (≠réception) respecte l'*ordre causal*
    - ✓ou s'il est identique sur chq site : *uniformité*
- Plusieurs modes
  - Anneau logique uni-directionnel
  - Avec serveur central
  - Avec estampille
  - Avec respect de l'ordre causal

L'anneau uni-directionnel



- A la réception d'un jeton sur un site j
  - traitement des msgs dans l'ordre donné par le jeton
  - j supprime les msg marqués j (en tête de jeton)
  - ajoute ses nvx msgs en queue de jeton (marqués j)
  - il transmet au site suivant dans l'anneau

Serveur diffusant

- Tout (ou presque) passe par le serveur !
  - un site désirant émettre informe le serveur
  - celui-ci numérote et assure la diffusion
  - utilisation de fenêtre d'anticipation

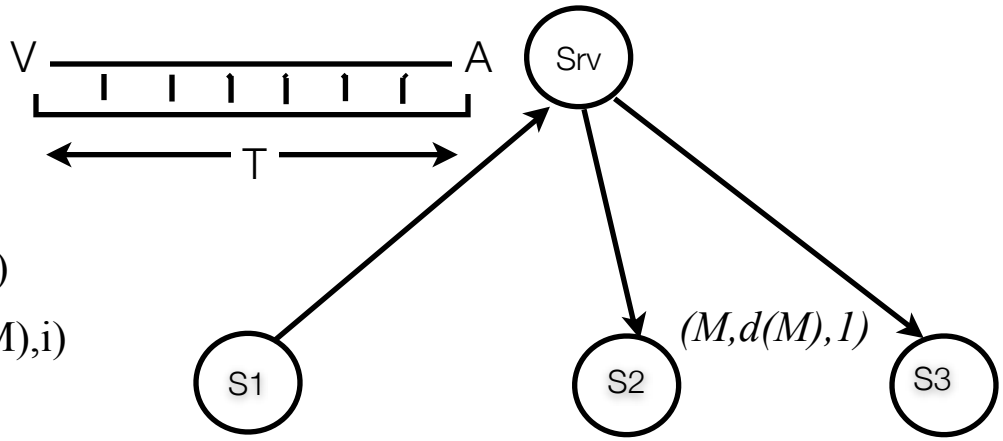
✓N : le nb de sites
✓T : le nb de msgs stockables
✓Buff[T] : le tampon en diffusion
✓A : le numéro du dernier msg reçu à diffuser
✓V : le dernier msg acquittés par tous les destinataires
✓ACK[N][T] : tableau d'acquittements

✓Ri : numéro du dernier msg reçu et acquitté sur le site i

- Coté serveur :

A la réception d'un msg M de S_i :

- Si tampon non plein (A -V < T)
  - ✓ d(M)=A+1 et diffuse (M,d(M),i)
  - ✓ A++
  - ✓ Buff[d(M)%T] = M
  - ✓ ACK[d(M)%T][x] = 0, ∀x ∈ [1,N]
- Sinon prévient émetteur :(=> impossible de diffuser pour le moment !



Serveur diffusant

Coté serveur (suite) :

✓A l'envoi d'un msg M, le serveur arme un timer, à son expiration il renvoie M à l'ensemble des sites n'ayant pas acquitté i.e ACK[d(M)%T][x] = 0 (+ ré-armement)

✓A réception d'une demande de ré-émission pour un site donné, il renvoie le msg concerné

✓A réception d'un ack (d(M),k) : ACK[d(M)%T][k] = 1, si ∀x, ACK[d(M)%T][x] =1, alors il déstocke M et si V = d(M) -1, alors V++ tant que ∄x | ACK[(d(M)%T)++][x] = 0

Coté client :

Sur S_k à la réception d'un message (M,d(M),i) :

✓si d(M) > R_k + 1 : il stocke le msg et re-demande tous les msg datés x tel que d>x> R_k + 1 non stockés

✓si d(M) < R_k +1 : il émet l'ack (d(M),k)

✓si d(M) = R_k +1 : il émet l'ack (d(M),k); «consomme» M et "décale jusqu'au prochain trou" R_k ++

Diffusion avec estampille

Avec l'hypothèse canaux FIFO

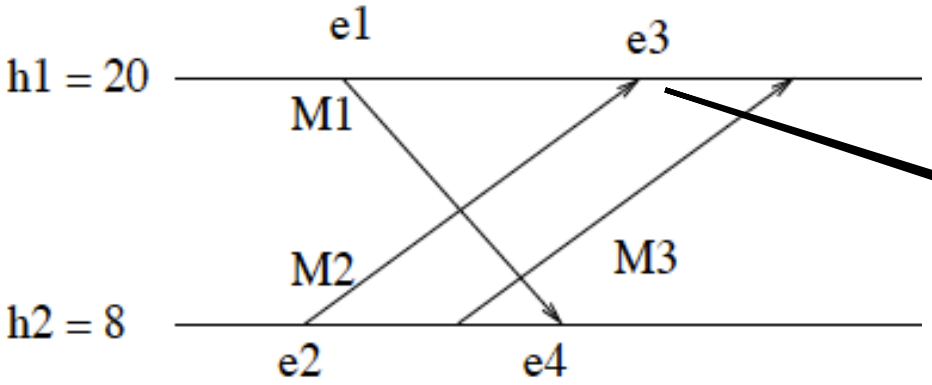
- ✓ «relativement» simple :)
- ✓ utilisation d'acquittements datés (et diffusés) + recalage d'horloge
- ✓ temps d'attente élevé :(

Sans l'hypothèse FIFO

- ✓ plus complexe !
- ✓ **ATOMIC BROADCAST (ABCAST) :**
  - ➡ basé sur les dates de validation, i.e, les acquittements de l'ensemble des sites
  - ➡ coût message plus élevé
- ✓ **CAUSAL BROADCAST (CBCAST) :**
  - ➡ l'ordre de délivrance est identique à l'ordre causal d'émission

Diffusion avec estampille

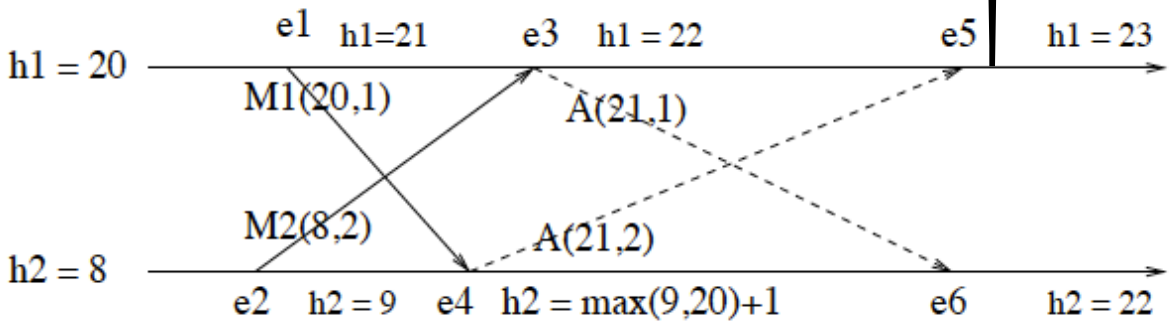
- Avec estampillage simple ?



S1 ne peut pas traiter M1 en e3

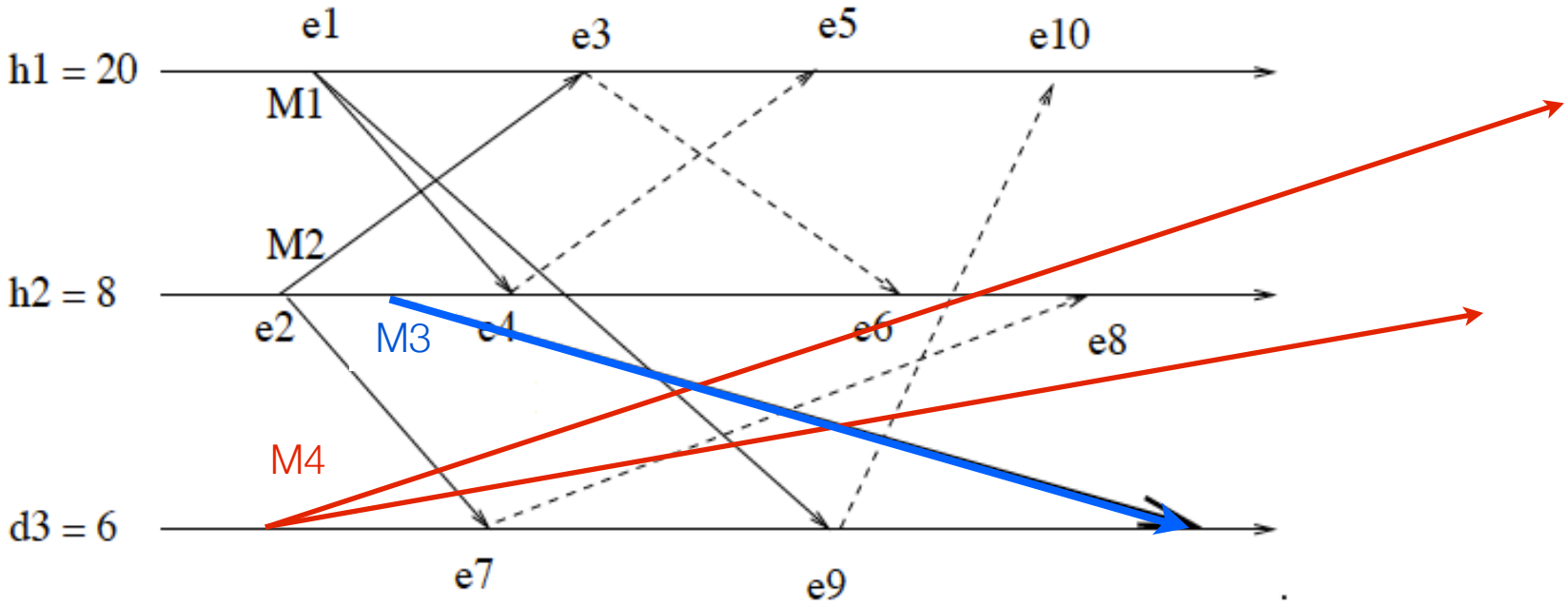
S1 peut traiter M2|M1en e5

- Avec acquittements ?



Diffusion avec estampille

- Avec trois sites (ou +) ?



✓ S1 et S2 doivent attendre les acks de S3

✓ S3 ne peut pas traiter M1 en e9 (M3!)

➡ Acquittements diffusés ?

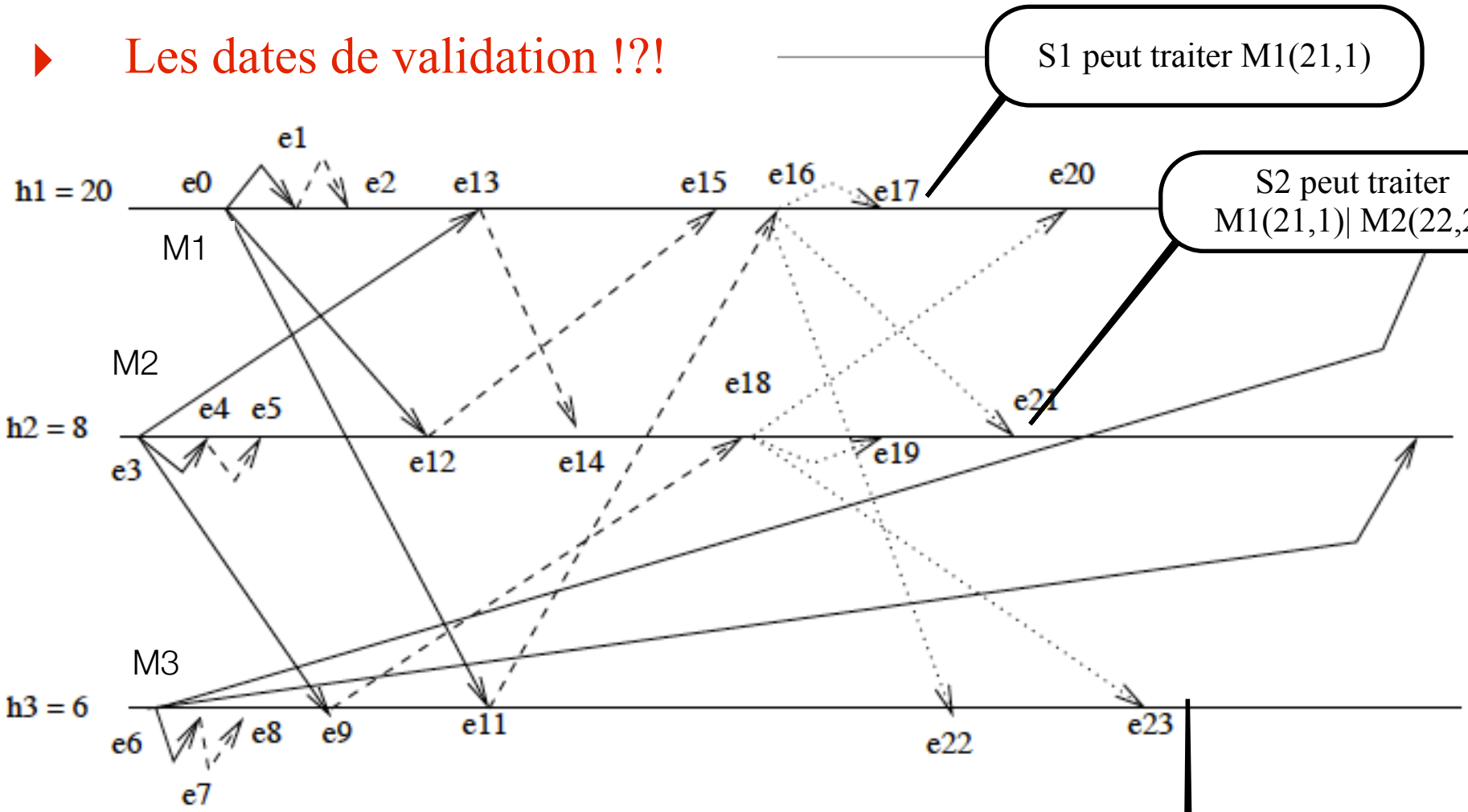
✓ et sans FIFO (avec des déséquencements, M4) ?

- Les dates d'émissions...

...c'est pas le bon plan !

ABCAST

- Les dates de validation !?!



ABCAST, pseudo-code

- Un site S_i diffuse un msg $M_{k,i}$ en le numérotant par son horloge locale H_i ($k = H_i$).
Puis H_i++; S_i maintient un tableau $A_{k,i}(1,...,n)$: acquittement reçus pour $M_{k,i}$.

- Un site S_j qui reçoit $M_{k,i}$, le place dans une file d'attente locale AM_j.
Il renvoie à l'émetteur un acquittement $R_{k,j}$ estampillé par la date de réception (H_j, j) puis H_j++;

- Un site S_i recevant un ack $R_{k,j}$ estampillé ($H_{k,j}$,j), met $A_{k,i}[j]$ à VRAI et mémorise ($H_{k,j}$,j).
Lorsque l'émetteur a reçu tous les $A_{k,j}$, il valide le message $M_{k,i}$ en diffusant un message $V_{k,i}$ estampillé par $E_{k,i} = max\{H_{k,j}, j\}$.

- Un site S_j qui reçoit un message de validation $V_{k,i}$ d'estampille $E_{k,i}$ (pour le msg $M_{k,i}$), date le message $M_{k,i}$ avec l'estampille $E_{k,i}$. Il effectue $H_j = \max(E_{k,i}, H_j)+1$, et déplace $M_{k,i}$ dans la liste des messages utilisables : UM_j

Reliable Communication in the Presence of Failures

KENNETH P. BIRMAN and THOMAS A. JOSEPH
Cornell University

uniformité

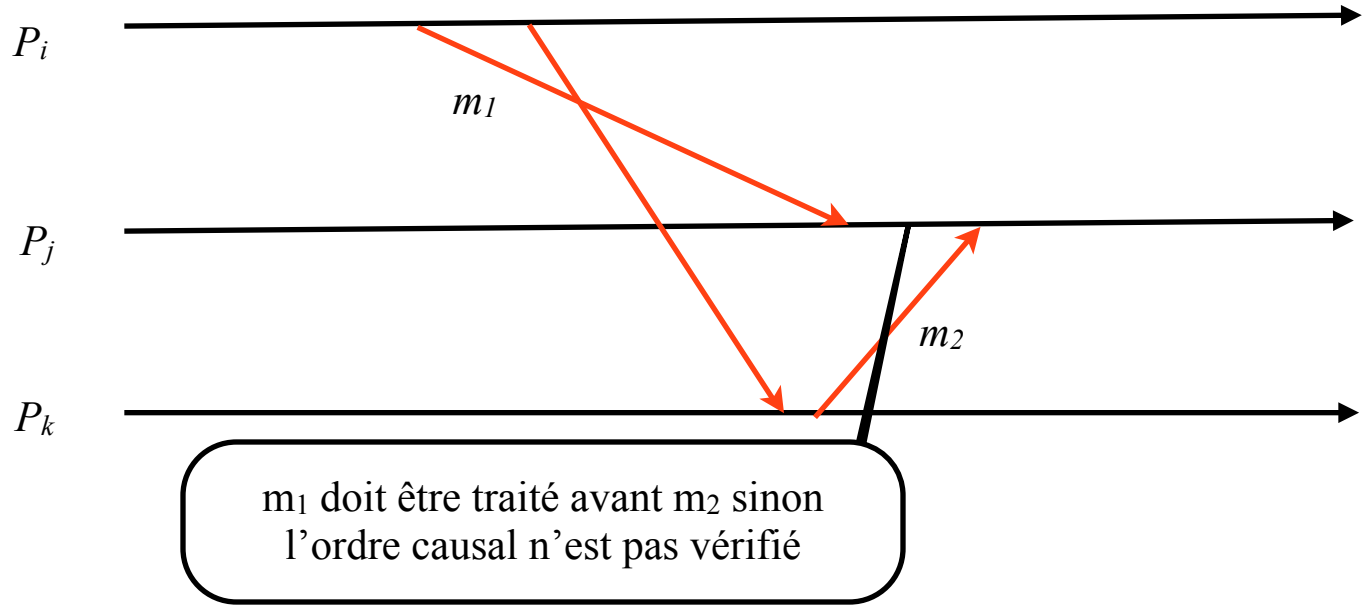
- Pour chaque msg m utilisable de date de validation v, un site j attend de connaitre la date de validation de tous les messages qu'il a acquittés avec une date inférieure à v avant validation de m.
- Finalement il traite dans l'ordre de leur date de validation les messages UM_j

Ordre causal

Ordre causal ≠ précédence causal (horloge vectorielle)

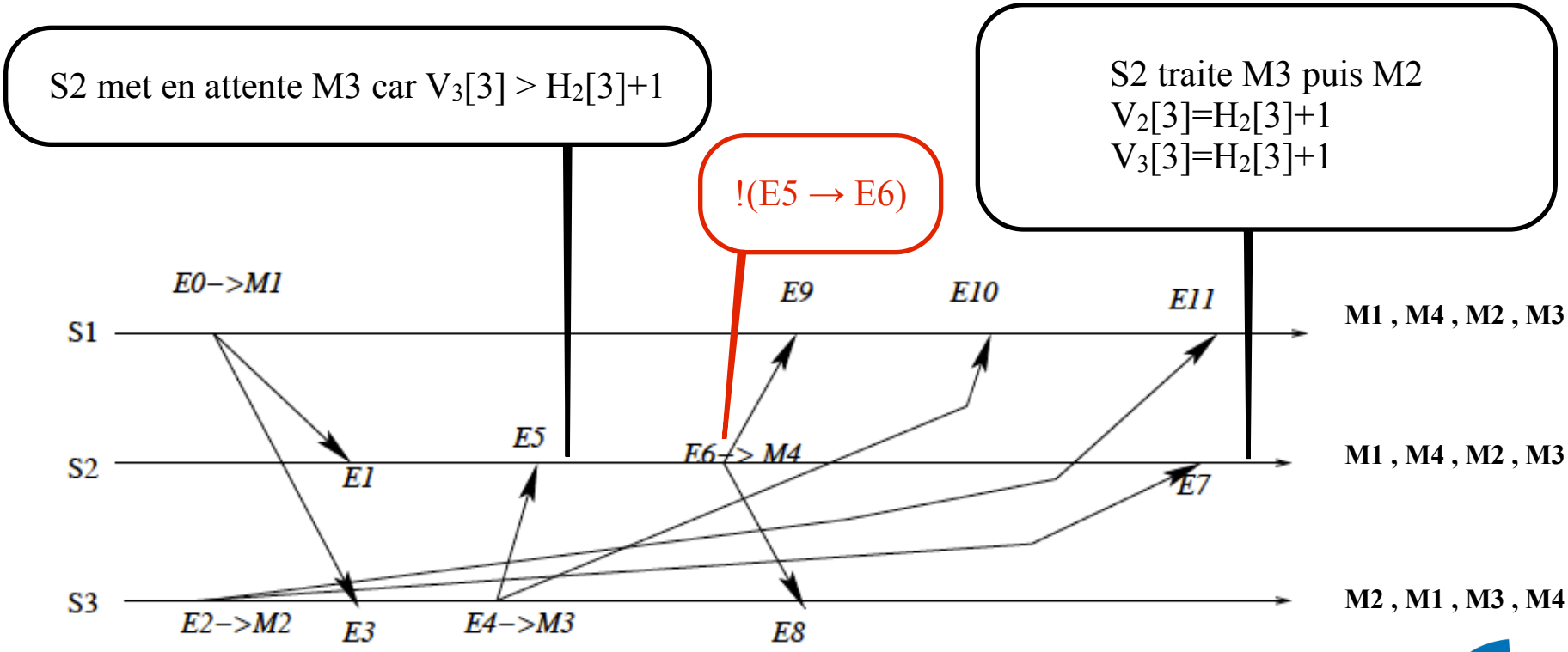
- ✓ $\forall P_i, P_j, P_k \ \forall m_1$ *émis sur* $C_{i \to j}$, $\forall m_2$ *émis sur* $C_{k \to j}$:

  $emission_i(m_1) \rightarrow emission_k(m_2) \Rightarrow reception_j(m_1) \rightarrow reception_j(m_2)$
- ✓ Ordre causal => FIFO mais la réciproque n'est pas vrai !



CBCAST

- Avec horloge vectorielle
- Les messages sont diffusés avec le vecteur courant
- Traitement local direct et mise en attente des msgs si besoin



CBCAST, pseudo-code

- Chaque site S_i utilise une horloge vectorielle $H_i[N]$ (N sites).
- Chaque envoi de message est daté avec cette horloge :

retrouver la relation de précédence causale entre les messages est alors possible (propriété des horloges vectorielles).

✓ avant de diffuser un message m, S_i traite m puis incrémente son horloge vectorielle $H_i[i]$ + + et estampille m par $V_m = H_i$

✓ à réception d'un msg m diffusé par S_i estampillé V_m, le récepteur S_j le met en attente avant délivrance jusqu'à ce que :

Ordre causal : Réception ≠ délivrance !

➡ $V_m[i] = H_j[i] + 1$

➡ $\forall\ k \neq i,\ V_m[k] \leq H_j[k]$

✓ après traitement du msg m envoyé par S_i, S_j incrémente son horloge vectorielle : $H_j[i]$ + +

Ordre causal ?

- Avec horloge matricielle
- Sans diffusion il est en effet nécessaire de connaitre «l'activité» de tous les sites