

# Algorithmique Répartie

[http ://homepages.laas.fr/francois/POLYS](http://homepages.laas.fr/francois/POLYS)

**François Vernadat**  
**INSA-DGEI - LAAS-CNRS**

**vernadat@laas.fr**



## Définitions

#1 “Un système distribué est un système qui s'exécute sur un ensemble de machines sans mémoire partagée, mais que pourtant l'utilisateur voit comme une seule et unique machine.” – A. Tanenbaum

#2 “Un système réparti est un système qui vous empêche de travailler quand une machine dont vous n'avez jamais entendu parler tombe en panne” – L. Lamport (ndlc : *exemple NFS (Network file system)*)

#3 “Un système réparti est un système (informatique) dans lequel les ressources (calcul, stockage, utilisateurs) et le **contrôle** ne sont pas centralisés” .

#4 “ Ensemble d'agents sans mémoire commune *coopérant* via un système de communication asynchrone “

=> les agents ont des capacités de traitement (processeurs), de stockage (mémoire), de communication avec le monde extérieur (capteurs, actionneurs)

4 / 152

## Introduction (suite)

### Pourquoi répartir

- Besoin de communication et de partage d'informations (système géographiquement réparti)
- Partage de ressources (programmes, données, services)
- Besoin de systèmes à haute disponibilité
- possibilité d'évoluer, critère économique, ...

### Exemples type :

Réseaux (ordinateurs, capteurs/actionneurs), WWW, NFS, Peer to peer, Contrôle aérien, Systèmes bancaires, ...

5 / 152

## Introduction (suite)

### Remarques

Systèmes souvent dynamiques :

Nombre d'agents et/ou Topologie du graphe de communication

Pas de Temps global

Systèmes vs Algorithmes (← terminaison)

### Avantages Escomptés :

Exploitation du //

↑ Puissance de Calcul

↑ Meilleure utilisation des ressources

↑ Fiabilité (redondance)

### Inconvénient :

COMPLEXITÉ

6 / 152

## Problème de la connaissance mutuelle

### Caractéristiques

- ① Agents sans mémoire commune  
Connaissance sur les autres ← Communication
- ② Communication Asynchrone
  - communication +/- fiable,
  - délai arbitraire (fini mais non borné)

### Conséquences

- ① Connaissance d'un Agent sur les autres est toujours sujette à caution (informations possiblement périmées)
- ② Pas de connaissance a priori de l'état global de son environnement (sa reconstruction est possible mais coûteuse)  
⇒ **Coopération difficile**

### Paradoxe de la connaissance dans un contexte asynchrone

Pour coopérer, les agents doivent avoir une connaissance commune

Pour obtenir cette connaissance commune, ils doivent communiquer

Toute communication "asynchrone" affaiblit la connaissance commune

7 / 152

## Nécessité d'un contrôle

Evolution du Système  $\mapsto$  évolution en // des  $\neq$  agents  
PB : Interdépendances (entre agents, sur des ressources, sur des données) interdisent certaines évolutions

## Exemples

- Exclusion mutuelle, Lecteurs/Ecrivains  
variable  $v$  partagée par deux agents ne peut être écrite et lue simultanément
- Producteur/Consommateur  
 $NPP - NPC \leq \text{Stockage\_Consommateur}$   
Pb du choix distant  
Illustré en TP Systèmes Concurrents (semestre suivant)

8 / 152

# Système distribué/réparti

## Éléments du Système : Agents, Données, Réseau, Contrôle

Agents *répartis* géographiquement

Données *distribuées* (*dupliquées & réparties*) sur les agents

Contrôle lui-aussi distribué entre les agents (à suivre)

### ex #1 : Duplication des données

	$S_1$	$S_2$	$\dots$	$S_n$
$n$ copies de $D$	$D_1$	$D_2$	$\dots$	$D_n$

$\Uparrow$  Accès + facile, Tolérance aux pannes

$\Downarrow$  (Donnée Variable) Assurer la cohérence des copies multiples

### ex # 2 : Répartition des données

	$S_1$	$S_2$	$\dots$	$S_n$
$E = \bigcup_{i \in I} e_i$	$e_1$	$e_2$	$\dots$	$e_n$

$\Uparrow$  Meilleure répartition, Confidentialité

$\Downarrow$  Accès à l'info (reconstitution)

Cohérence relative des données  $\sum_{i \in I} e_i = \text{Cste}$

9 / 152

# Contrôle centralisé Vs distribué

## Contrôle Centralisé

Un site particulier (défini statiquement) joue le rôle central d'arbitre :

- L'arbitre prend toutes les décisions.
- Il a généralement besoin de connaissances disséminées dans le système.

Avantages/Inconvénients

- (+) Très Simple (modulo le **pb** de connaissance)
  - (-) // disparaît, possible goulot d'étranglement, panne de l'arbitre **fatale**
- ⇒ Les avantages escomptés de la distribution disparaissent

## Contrôle Distribué

Pas de chef/arbitre statiquement défini

- Agents égaux en droit et en devoir

Avantages/Inconvénients

- (+) Pannes possibles (⇒ fonctionnement dégradé), // "maximum"
  - (-) Complexe
- ⇒ Les avantages escomptés de la distribution persistent

## Influence de la topologie :

Certaines topologies facilitent la répartition du contrôle (anneaux, arbres, ...)

⇒ Algorithmes répartis associés pour se ramener à ces topologies

10 / 152

# Propriétés attendues dans un Système Réparti

## Classification de Leslie Lamport

### ❶ Propriété de sûreté (safety)

Enoncé type : "*Rien de mauvais ne peut arriver*"

ex : jamais deux agents écrivant simultanément la même ressource

ex : deux philosophes adjacents ne peuvent manger simultanément

### ❷ Propriété de vivacité (liveness)

Enoncé type : "*Quelque chose de bon finira par arriver*"

ex : un agent en attente pour écrire finira par écrire

ex : absence de famine pour les philosophes

**Propriété :** Toute propriété d'un système réparti peut être exprimée comme une combinaison de propriétés de sûreté et de vivacité.

## Leslie Lamport – Prix Dijkstra 2000, Prix Turing 2013

Chercheur américain spécialiste de l'algorithmique répartie

Safety/Liveness, Causalité & Horloges de Lamport (cf chapitre suivant)

Algorithmes répartis, Temporal logic of actions (TLA), Latex (!), ...

## Caractéristiques

### Générales

- Simplicité  
structure de données sur chaque site, messages, ...
- Nombre de messages  
voir chapitres Mutex et gestion des données distribuées
- Taille des messages
- Résistance aux Pannes,
- Autostabilisant (se dit d'un système/algorithme qui après une défaillance revient de lui-même à un fonctionnement correct)

### Spécifiques

- Exclusion mutuelle : Temps minimum entre deux C.S consécutives
- Diffusion : Temps de propagation
- ...

12 / 152

## Rapide panorama du cours

### Quelques problèmes classiques

- Election (Tirage au sort),
- Exclusion mutuelle,
- Gestion des données distribuées,
- Détection de la terminaison

### Quelques Outils Génériques

- Temps causal,
- Phases,
- Vagues,
- Consensus, Quorums

### Topologies d'intérêt

- Arbres couvrants (diffusion, terminaison, mutex)
- Anneaux (mutex, équité)

13 / 152

# Introduction

## Motivation

Absence de temps global/physique dans un système réparti (SR)  
Temps Logique  $\mapsto$  Permettre de reconstituer une notion de temps -  
et l'ordre associé entre les différents événements - dans un SR

## Temps Logique/Causal

Temps logique calculé localement  
 $\rightarrow$  Ordonnancement local des événements  
 $\rightarrow$  Etablir des propriétés entre ceux-ci  
 $\rightarrow$  Simplifier le Contrôle  
  
 $\rightarrow$  Permettre de représenter graphiquement l'exécution **normalisée**  
d'un SR via des chronogrammes respectant la causalité/parallélisme  
(ordonnée = ref du site / abscisse = horloge de Lamport)

15 / 152

# Causalité

## Modèle d'exécution d'un SR

- Système composé de  $n$  sites reliés par des canaux *fiables* (i.e. sans perte) mais avec délai d'acheminement arbitraire
- Exécution d'un système : Ensemble d'événements où
- Événement : émission, réception, evt interne

## Relation de Causalité (potentielle)

Relation régie par 2 contraintes "physiques"

- $C_1$  : Les événements qui se déroulent sur un site sont totalement ordonnés (même si cet ordre est arbitraire)
- $C_2$  : Pour un message  $M$ , l'événement associé à son émission précède l'événement associé à sa réception

16 / 152

# Ordre Causal $\rightsquigarrow$

## Définition de $\rightsquigarrow$

$A \rightsquigarrow B =_{\text{def}}$  "A est possiblement une cause de B" ssi

- ①  $\exists$  un message  $M$  tel que :
  - A corresponde à l'émission de  $M$  et
  - B corresponde à sa réception
- ② A et B ont eu lieu sur le même site et A avant B
- ③  $\exists C : A \rightsquigarrow C$  et  $C \rightsquigarrow B$  (transitivité)

## Propriétés de $\rightsquigarrow$

- Relation d' **ordre strict** (i.e., transitif, irréflexif, antisymétrique) **partiel**

Par ex pour les vecteurs,  $\leq$  est un ordre partiel.

$$\text{Ainsi : } \begin{vmatrix} 0 \\ 1 \end{vmatrix} \not\leq \begin{vmatrix} 1 \\ 0 \end{vmatrix}$$

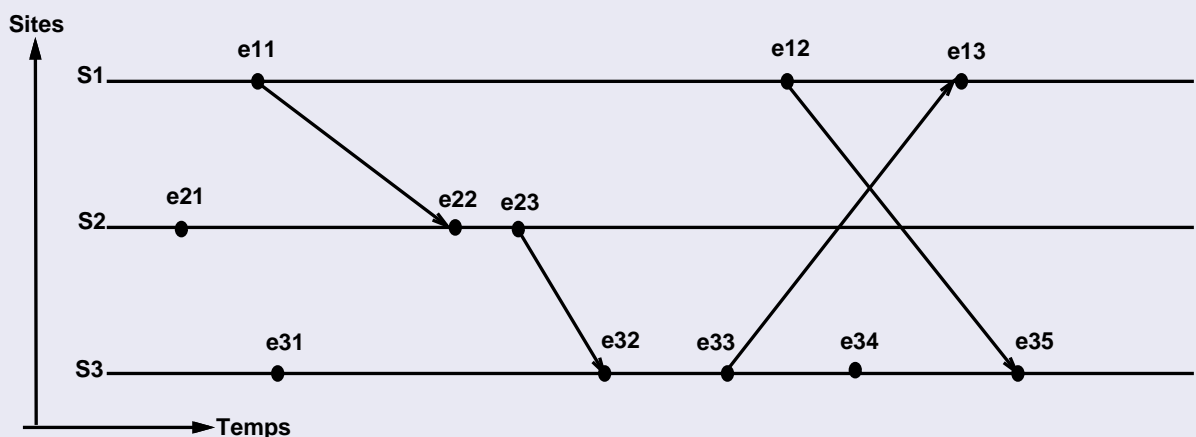
## Indépendance causale (notée $\wr$ )

- $A \wr B$  ssi  $\neg(A \rightsquigarrow B) \wedge \neg(B \rightsquigarrow A)$
- $\wr$  va permettre de rendre compte du parallélisme

17 / 152

# Ordre Causal : Chronogrammes

## Chronogramme



3 Sites (Agents) : S1, S2, S3

3 événements internes : e21, e31, e34

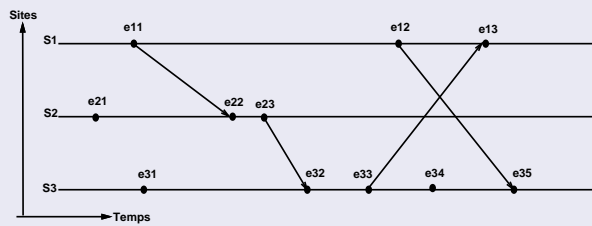
4 messages : e11-e12, e23-e32, e33-e13, e12-e35

18 / 152



# Ordre Causal : Chronogrammes (suite)

## Causalité



$e11 \rightsquigarrow e32$  car  $\left\{ \begin{array}{l} e11 \rightsquigarrow e22 \text{ (1)} \\ e22 \rightsquigarrow e23 \text{ (2)} \\ e23 \rightsquigarrow e32 \text{ (1)} \end{array} \right. + \text{transitivité (3)}$

$e12 \not\rightsquigarrow e34$  car  $e34 \not\rightsquigarrow e12$  et  $e12 \not\rightsquigarrow e34$

- Chemin causal : suite d'événements directement contigus pour  $\rightsquigarrow$   
Exemples :  $(e11, e12, e13)$ ,  $(e11, e22, e23, e33, e13)$ ,  $(e11, e22, e23, e33, e34, e35)$ , etc  
nb : les preuves seront basées sur des récurrences sur la longueur des chemins causaux.
- Indépendance causale  $\neq$  Absence de Cause Commune  
Ainsi  $e12 \not\rightsquigarrow e34$  et  $(e11 \rightsquigarrow e34 \text{ et } e11 \rightsquigarrow e12)$

19 / 152

# Horloges Logiques de Lamport (1978)

## Ordre de Lamport $\prec$

$\prec$  : Ordre calculé de façon répartie par chaque site au fur et à mesure de l'exécution du système

$\prec$  est un ordre **total** "cohérent" avec l'ordre causal (qui lui est **partiel**)

cohérence : Si  $A \rightsquigarrow B$  Alors  $A \prec B$

## Horloges de Lamport

Chaque site ( $i$ ) dispose d'une horloge logique  $H_i$  (initialisée à 0)

Tout message envoyé est estampillé par la valeur de l'horloge locale

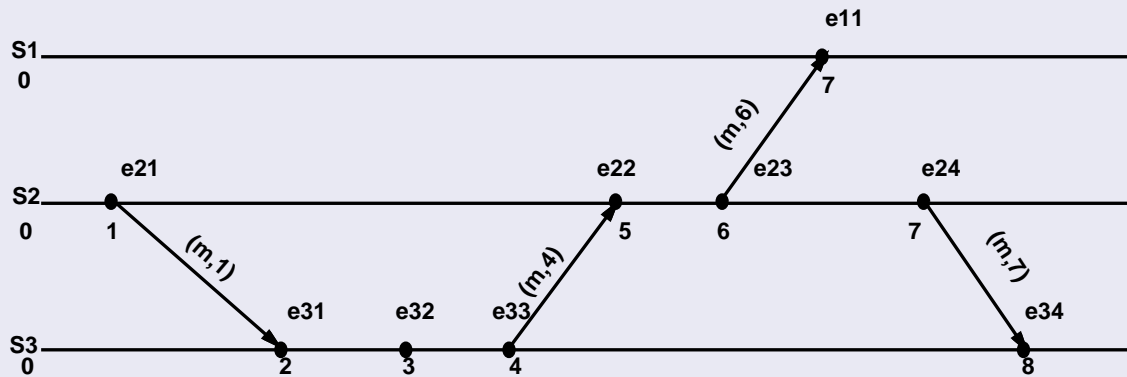
## Evolution des Horloges

- $R_1$  : Entre 2 événements locaux, un site incrémente son horloge locale de 1  
 $H_i := H_i + 1$
- $R_2$  : A la réception d'un message estampillé par  $k$ , le site  $i$  recale son horloge ainsi :  $H_i := \text{Max}(H_i, k) + 1$

20 / 152

# Horloges de Lamport

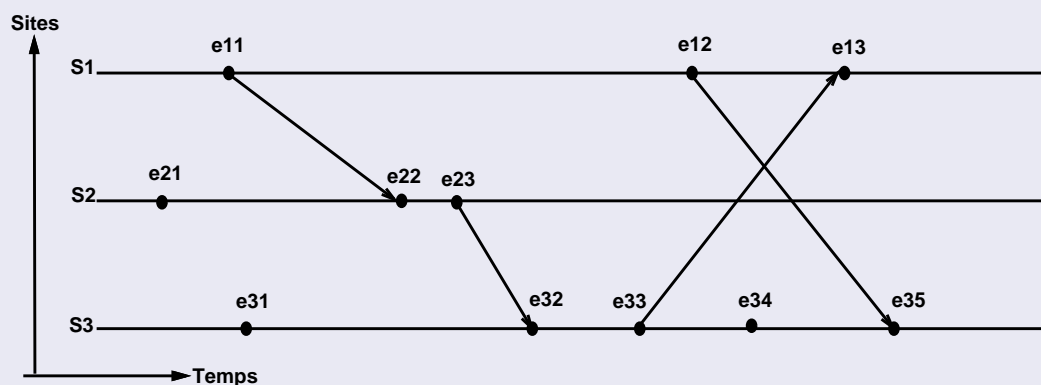
## Exemple



21 / 152

# Horloges de Lamport

## Exercice



- 1 Donnez les horloges de Lamport des événements représentés ci-dessus
- 2 e11 et e31 sont-ils causalement ordonnés ?
- 3 Idem pour e13 et e34 ?
- 4 Quel rapport avec leurs estampilles ?

22 / 152

# Ordre de Lamport

## Définition de $\prec$

Estampillage de Lamport  $\mathcal{E} : Evt \mapsto \mathbb{N}$  Estampillage de Lamport  
 $A \mapsto \mathcal{E}(A)$

$\prec (\subset Evt \times Evt) =_{def} A \prec B \text{ ssi } \mathcal{E}(A) < \mathcal{E}(B)$

## Propriété de cohérence

Si  $A \rightsquigarrow B$  Alors  $A \prec B$  preuve par récurrence sur " $| \rightsquigarrow |$ "

## Corollaire

$\mathcal{E}(A) = \mathcal{E}(B) \Rightarrow A \wr B$   
(car  $\mathcal{E}(A) \geq \mathcal{E}(B) \Rightarrow \neg(A \rightsquigarrow B)$ )

## En résumé

$\prec$  est un ordre total (large) cohérent avec la causalité  
 $\mapsto$  ordre total **strict** (en incorporant l'identité des sites)  
Ordre total strict dynamiquement calculé  $\mapsto$  Résolution distribuée de conflits

23 / 152

# Application à la résolution distribuée de conflits

## Algorithme (schéma) d'exclusion mutuelle à base de permissions (bloquantes)

- Pour entrer en section critique (SC), un site demande la permissions des autres sites.
- Il entre en section critique lorsqu'il a obtenu toutes les permissions. Il libère tous les sites à sa sortie de section critique.
- Un site *oisif* accorde sa permission et se bloque en attente d'un message de libération

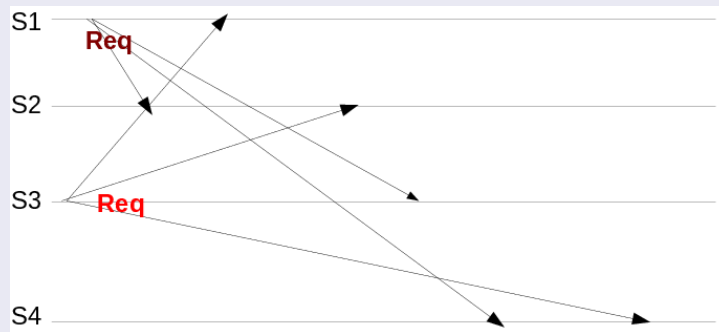
## Propriétés à garantir

- Respect de l'exclusion mutuelle : un processus, au plus, présent en section critique (SC) (sûreté)
- Un processus en attente de SC, l'obtient en temps fini (vivacité)

24 / 152

# Application à la résolution distribuée de conflits (suite)

## Quid des requêtes concurrentes ?



S3 et S1 sont en conflits pour l'accès à la CS

⇒ "Interblocage" car le protocole ne prévoit rien

S1 attend l'autorisation de S3 tandis que S3 attend l'autorisation de S1

## Interblocage

L'interblocage se produit lorsque deux (ou plus) processus concurrents s'attendent mutuellement : ici S1 et S3

nb : Un interblocage conduit à un blocage mais tout blocage ne procède pas d'un interblocage.

25 / 152

# Application à la résolution distribuée de conflits (suite)

## Résolution de l'interblocage par des priorités

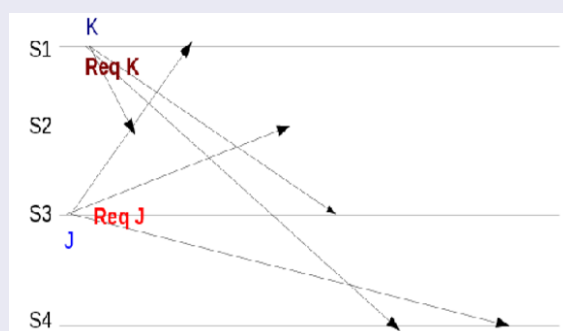
Priorité **statique** :

- + Utiliser le nom des sites pour établir une priorité entre ceux-ci
- + Signer les messages émis

Résolution : S1 est plus prioritaire que S3. Il considère la requête de S3 comme sa permission. S3 sait que sa requête est moins prioritaire que celle de S1 ; il se bloque jusqu'à la réception du message de libération.

Ca fonctionne mais ... mécanisme **inégalitaire** !

Priorité **dynamique** (à l'aide de compteurs ou d'horloges de Lamport)  
les requêtes sont estampillées par la valeur de l'Horloge de Lamport  
ses valeurs sont utilisées pour décréter les priorités des requêtes



26 / 152

# Horloges de Lamport (fin)

## Horloges de Lamport / Compteurs

- + Solution générique et répartie pour résoudre les conflits  
*Alternative par "Tirage au sort" vue plus tard*
- Comment borner les compteurs/horloges ?  
(Ricart & Aggrawala, Lamport, ....)

## Lamport Variante

Incrémentation des horloges ( $inc > 0$ )

+ $d$  dans  $R_1$  où  $d$  durée de l'événement associé

+ $c$  dans  $R_2$  où  $c$  durée de la communication

$\mapsto h \approx$  temps nécessaire à la réalisation de l'événement associé.

## Bilan provisoire (1979)

$\rightsquigarrow$ , la relation de causalité est un ordre partiel

$\prec$ , l'ordre de Lamport, est total !

Propriété de cohérence :  $A \rightsquigarrow B \Rightarrow A \prec B$

Quid de  $\Leftarrow$  ?

27 / 152

# Horloges vectorielles (Fidge & Mattern (88/89))

## Principe

Chaque site est doté d'un vecteur d'horloges  $Vh_i[1 \dots n]$

Tout message est estampillé par le vecteur d'horloges de l'émetteur

**Interprétation** :  $Vh_i(j) \approx$  "Connaissance" du site  $i$  sur le "comportement" du site  $j$

## Evolutions des Horloges

$R_1$  : Avant tout événement,  $S_i$  incrémente la composante de son horloge  
 $Vh_i(i) := Vh_i(i) + 1$

$R_2$  : A la réception d'un message estampillé par  $VH$ , le récepteur ( $i$ ) recalcule son vecteur d'horloges ainsi :  $\forall j \in [1..n] : Vh_i(j) := \text{Max}(Vh_i(j), VH(j))$

## Ordre de Fidge & Mattern (FM) $\prec$

Estampillage de FM :  $\gamma : \text{Evt} \mapsto \mathbb{N} \quad A \mapsto \gamma(A)$

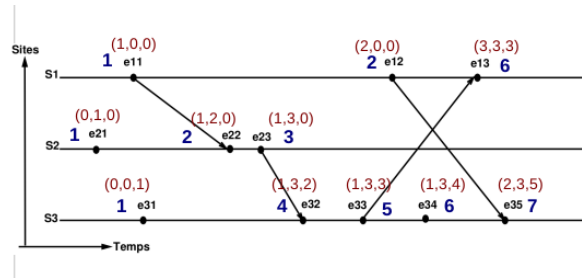
$< (\subset \text{Evt} \times \text{Evt}) =_{\text{def}} A < B \text{ ssi } \gamma(A) < \gamma(B)$  ( $<$  est un ordre partiel)

**Propriété Adéquation** :  $A \rightsquigarrow B \text{ ssi } A < B$  par récurrence sur " $| \rightsquigarrow |$ "

Corollaire :  $\gamma(A) \# \gamma(B) \Leftrightarrow A \not\prec B$

28 / 152

# Horloges vectorielles FM : Exemple



## Rappel

$e_{11} \rightsquigarrow e_{32}$  car  $\left\{ \begin{array}{l} e_{11} \rightsquigarrow e_{22} \text{ (1)} \\ e_{22} \rightsquigarrow e_{23} \text{ (2)} \\ e_{23} \rightsquigarrow e_{32} \text{ (1)} \end{array} \right. + \text{transitivité (3)}$   
 $e_{12} \not\rightsquigarrow e_{34}$  car  $e_{34} \not\rightsquigarrow e_{12}$  et  $e_{12} \not\rightsquigarrow e_{34}$

## Exercice : Que peut-on dire des couples d'événements suivants ?

$e_{21} \& e_{11}$

$e_{21} \& e_{32}$

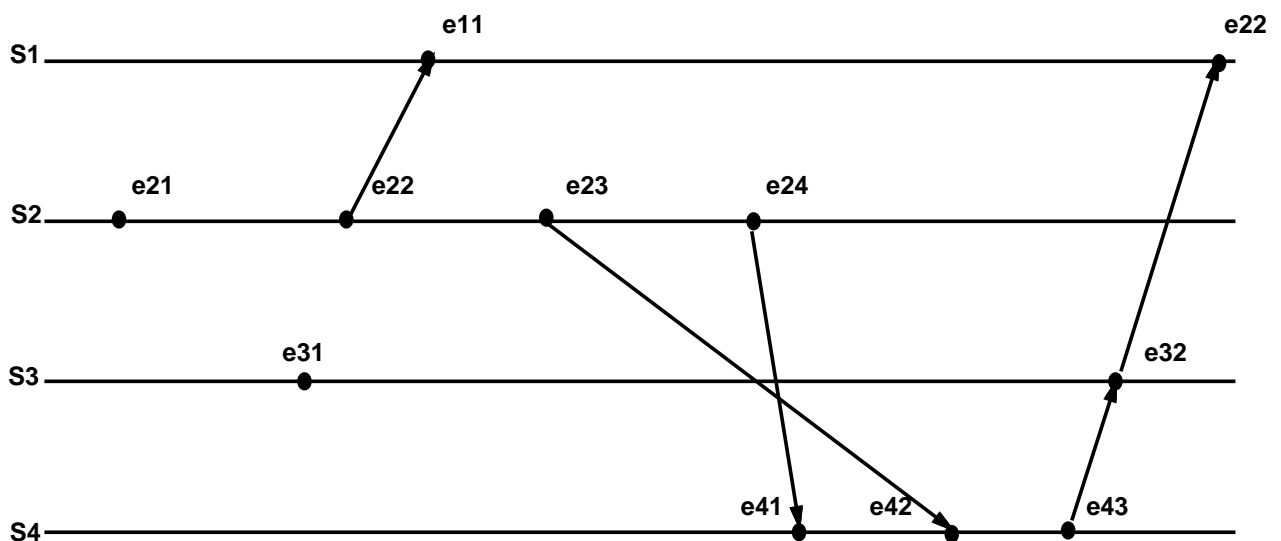
$e_{22} \& e_{32}$

$e_{11} \& e_{32}$   $e_{11} \& e_{23}$

29 / 152

# Comparatif O.G, Lamport F&M (exercice)

où **O.G** est un Observateur Global qui voit/note tout



30 / 152

## O.G, Lamport F&M : Renseigner le tableau ci-dessous

	e21	e31	e22	e11	e23	e24	e41	e42	e43	e32	e22
L											
F & M											
O.G											

### Moralité

F&M offre la meilleure observation

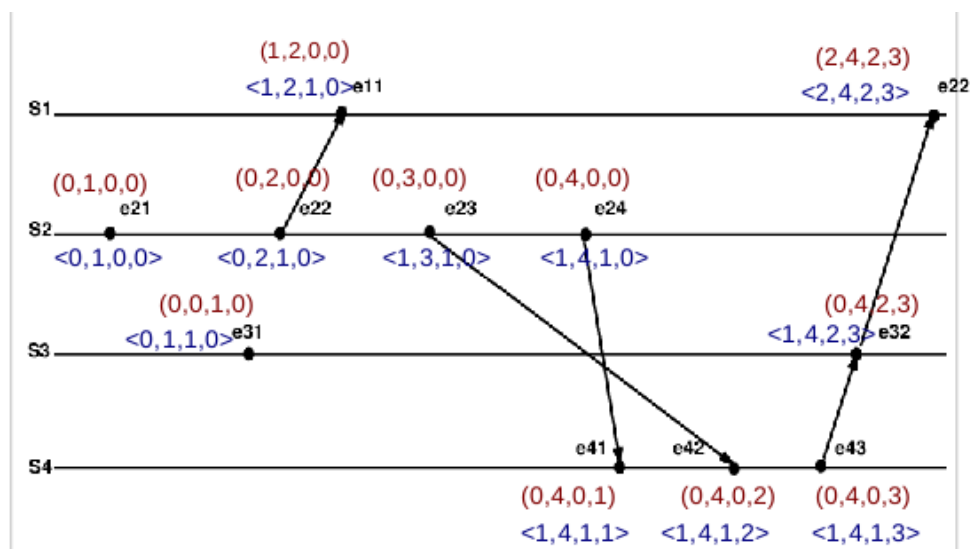
Paradoxalement, une vision centrale n'aide pas!!!!

**Application de F&M au debugging réparti**

31 / 152

## Comparatif O.G, Lamport F&M (solution)

où **O.G** est un Observateur Global qui voit/note tout



32 / 152

# Autre application de F&M : Diffusion causale

## Ordre Causal : Propriété caractérisant la diffusion des messages

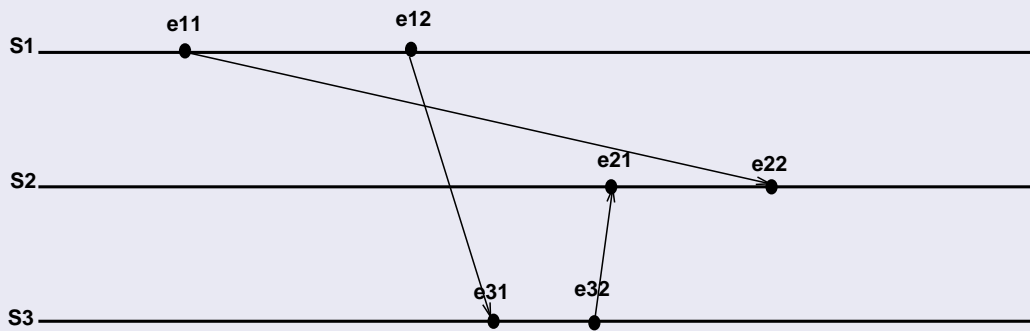
Medium  $\models$  Ordre Causal ssi

$\forall P_i, P_j, P_k, \forall m_1$  émis sur  $c_{ij}, \forall m_2$  émis sur  $c_{kj}$  :

$emission_i(m_1) \prec emission_k(m_2) \Rightarrow reception_j(m_1) \prec reception_j(m_2)$

Prop : Ordre Causal  $\Rightarrow$  Ordre Fifo (réciproque fausse)

## Exemple de violation de l'ordre causal : e22 devrait arriver avant e21



## Algos de "diffusion causale" basés sur F&M

Solution : e21 est mis en attente et ne sera délivré qu'après e22

Applications : Jeux distribués, Applications militaires

33 / 152

# Synchronisation par phases

## Définitions de Base

**Synchronisation** : Ens de Règles (mécanismes) permettant de contrôler l'évolution d'un S.R

**Phases** : Concept générique permettant de résoudre le contrôle réparti d'une certaine classe de calculs répartis.

## Classe de calculs considérés :

$P_1 \quad P_2 \quad \dots \quad P_n \quad N \text{ sites}$   
 $d_1 \quad d_2 \quad \dots \quad d_n \quad N \text{ données}$   
Calcul de  $R_i = F(d_1, d_2, \dots, d_n)$

- $R_i = R_j \quad \forall (i, j) \rightsquigarrow$  PB Election (Réunion, Négociation)
- $R_i \neq R_j \rightsquigarrow$  Tables de Routage (optimaux)  
Arbres de Recouvrement de poids minimaux

## Algorithme $\mapsto$ Schéma d'algorithme

Gallager 1983 : Calcul de tables de routages dans les réseaux

Konig 1988 : Schéma général d'algorithme à "Phases"

35 / 152



# Algorithmes par phases

## Concept de Phases

- Tous les sites ont le même comportement (symétrie " $\rightsquigarrow$ " Distribué)
- Phase :
  - a) Envoyer un message à chacun de ses voisins
  - b) Recevoir un message de chacun d'eux
  - c) Calcul local (en fonction de la connaissance acquise)

## Algorithme par Phases

$\left\{ \begin{array}{ll} \text{I]} & \text{Initialisation} \\ \text{II]} & \text{Phases}^* \approx \text{"Itération Répartie"} \\ \text{III]} & \text{Terminaison} \end{array} \right.$

## Remarques

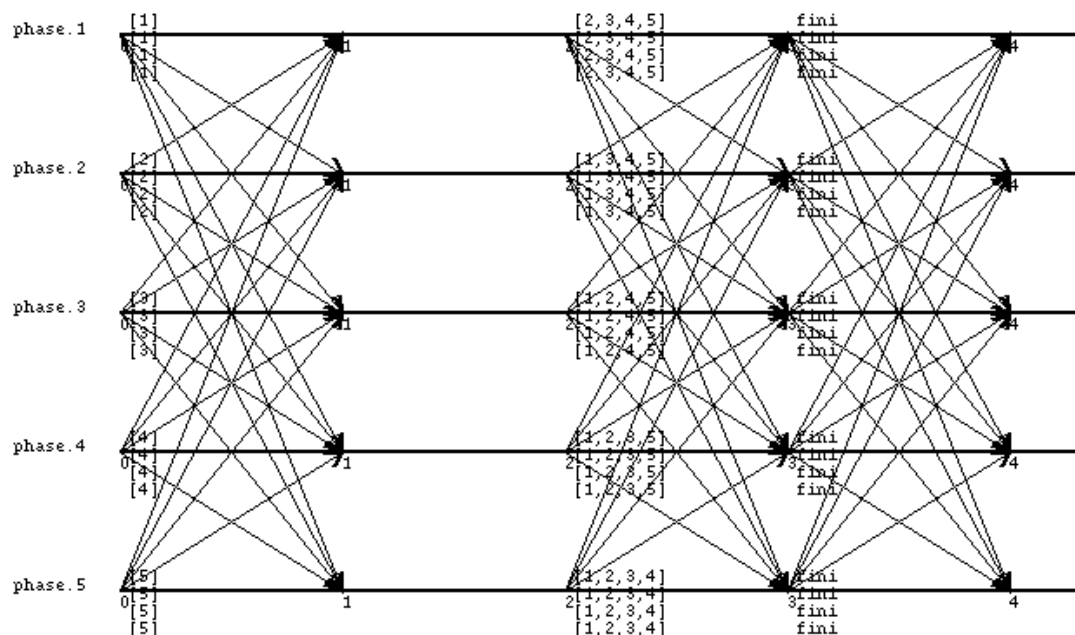
*Exécution phasée  $\rightsquigarrow$  Couplage) très fort entre les  $\neq$  sites :*

*$S_i$  peut commencer la phase  $k$  avant  $S_{i+1}$ , mais  $S_i$  ne terminera pas la phase  $k$  sans que  $S_{i+1}$  ne l'ait commencée.*

**Résoudre** la détection répartie de la **Terminaison**

36 / 152

## Chronogrammes types d'Algorithmes par phases : Graphe Complet

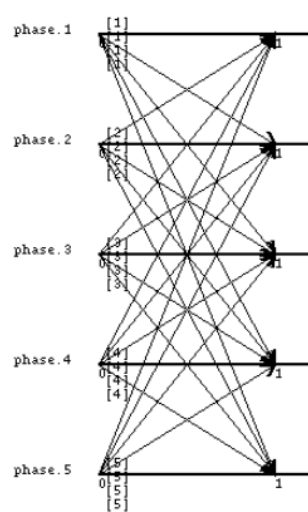


37 / 152

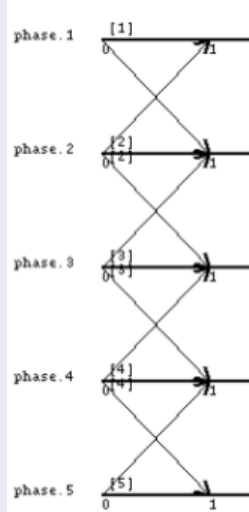
# Chronogrammes d'algorithmes phasés et Topologie

## exo : Retrouver la topologie à partir d'un chronogramme

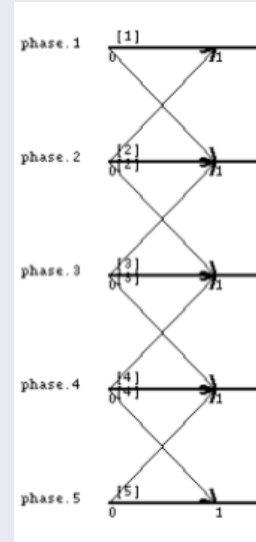
### Graphe Complet



?



### A compléter pour obtenir un anneau



38 / 152

## Calcul par phases des Tables de Routages Optimaux (Gallager 1983)

### Hypothèses générales

- Communication fiable
- Sites interconnectés par des canaux (bi-directionnels)
- Chaque site connaît "ses" canaux (les ports le reliant à l'extérieur)

### Table de Routage (locale) :

$$Tro : Canaux \mapsto 2^{Sites}$$

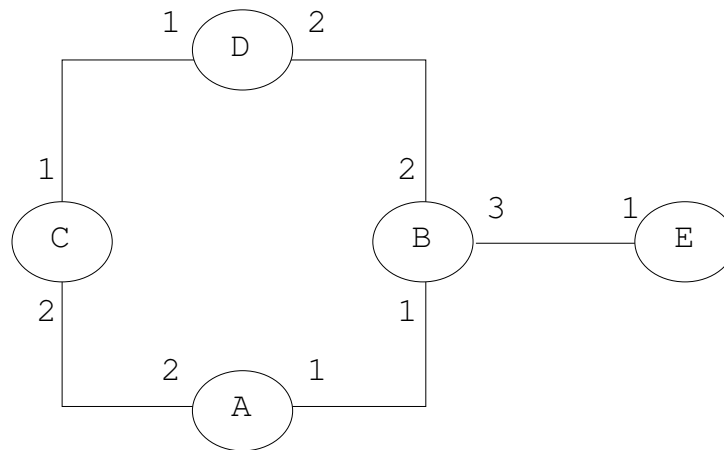
$$Tro(can) = S \text{ où } S \subset Sites$$

### Utilisation

Un site  $S_i$  recevant un message  $(Mesg, Dest)$  avec  $dest \neq S_i$  ré-expédie le message  $(Mesg, Dest)$  sur un canal  $c$  tel que  $Dest \in Tro(c)$  vérifiant : Pour un site  $A, B \in Tro(c)$  ssi  $c$  est un canal à emprunter pour joindre  $B$  à partir de  $A$  en suivant le + court chemin.

39 / 152

## Exemple : Topologie et de Tables de routages associée



A	Sites
1	{B, E, D}
2	{C, <u>D</u> }

C	Sites
1	{B, E, D}
2	{A, <u>B</u> , <u>E</u> }

E	Sites
1	{A, B, C, D}

B	Sites
1	{A, C}
2	{ <u>C</u> , D}
3	{E}

D	Sites
1	{A, C}
2	{ <u>A</u> , B, E}

40 / 152

## Algorithme de Calcul des Tables de routages (préliminaires)

Notations : Pour un site  $S_i$

$canaux_i$  : canaux du site,  $routage_i$  : table du site

$ph_i$  : compteur de phases

$inf_i, new_i$  : Ensemble d'identité de sites

Informations apprises  $new_i$  / connues  $inf_i$

Hypothèse simplificatrice : Chaque site connaît le diamètre du graphe

nb : On *triche* pour simplifier la 1ere version, on lèvera ensuite cette hypothèse.

### Rappels sur les Graphes

- **Distance** :  $Sites \times Sites \mapsto \mathbb{N}$   
distance = longueur du + court chemin les reliant  
(+ court = nombre minimum d'arêtes)
- **Excentricité** :  $Sites \mapsto \mathbb{N}$   
Distance maximum entre le site et les autres
- **Diamètre d'un Réseau** :  
Maximum des Excentricités

41 / 152

# Algorithme de Calcul des Tables de routages

Init :

$ph_i := 0; new_i := \{i\}; inf_i := \{i\};$

Phases\*

Tant que  $ph_i < diametre$  faire - - Simplification pour la terminaison

$ph_i := ph_i + 1;$

$\forall c \in canaux_i : \text{envoyer } new_i \text{ sur } c$

$new_i := \emptyset$

$\forall c \in canaux_i$

{ recevoir  $m$  sur  $c$

$Y = m - (inf_i \cup new_i)$

$routing_i(c) := routing_i(c) \cup Y$

$new_i := new_i \cup Y$  }

$inf_i := inf_i \cup new_i$

fin\_tant\_que

Term :  $\emptyset$

42 / 152

## Exemple d'exécution sur la topologie suivante

$A.1 \leftrightarrow 1.B.2 \leftrightarrow 1.C$

Phase 1 sur B

$inf = \{B\}, new = \{B\}$

**Emissions**

! 1 {B} & ! 2 {B}

$new \leftarrow \emptyset$

**Réceptions**

? 1 {A} ( $Y = \{A\}$ )

$routing(1) := \emptyset \cup \{A\}$

$new \leftarrow \emptyset \cup \{A\}$

? 2 {C} ( $Y = \{C\}$ )

$routing(2) := \emptyset \cup \{C\}$

$new \leftarrow \{A\} \cup \{C\}$

$inf = \{B, A, C\}, new = \{A, C\}$

Phase 1 sur A

(Pour C s(A/C))

$inf = \{A\}, new = \{A\}$

**Emissions**

! 1 {A}

$new \leftarrow \emptyset$

**Réceptions**

? 1 {B} ( $Y = \{B\}$ )

$routing(1) := \emptyset \cup$

{B}

$new \leftarrow \emptyset \cup \{B\}$

$inf = \{A, B\}$

$new = \{B\}$

43 / 152

## Exemple d'exécution (suite)

$A.1 \leftrightarrow 1.B.2 \leftrightarrow 1.C$

### Phase 2 sur B

#### Emissions

! 1 {A,C} & ! 2 {A,C}  
 $new \leftarrow \emptyset$

#### Réceptions

? 1 {B} ( $Y = \emptyset$ )  
? 2 {B} ( $Y = \emptyset$ )  
 $inf = \{B, A, C\}, new = \emptyset$

### Phase 2 sur A (analogue pour C)

#### Emissions

! 1 {B}  
 $new \leftarrow \emptyset$

#### Réceptions

? 1 {A,C} ( $Y = \{C\}$ )  
     $routing(1) := \{B\} \cup \{C\}$   
 $inf = \{B, A, C\}, new = \{C\}$

Remarques : Pour cette topologie, Diametre = 2

- B est en position centrale (il est d'excentricité minimale 1)
- B possède toute l'info à l'issue de la 1ère phase
- La seconde phase permet aux sites plus excentrés (A et C) d'obtenir à leur tour cette information

44 / 152

## Terminaison d'un algorithme à phases

### Cas général (sans tricherie)

Pas besoin de connaître le diamètre du graphe

- A l'issue de la  $k^{ieme}$  phase, le site  $i$  a tout appris des sites au plus distants de  $k$
- Soit  $new_k$  la valeur de  $new$  à l'issue de la  $k^{ieme}$  phase  
Si  $new_k = \emptyset$  alors  $\forall p \geq k \quad new_p = \emptyset$
- L'algorithme est terminé pour le site  $i$   
la première fois que  $new$  passe à  $\emptyset$
- Pour un site  $i$  tq nb phases = Excentricite( $i$ )  
le site  $i$  sait tout mais il ne le sait pas  
pour nb phases = Excentricite( $i$ ) + 1 alors  $new = \emptyset$   
le site  $i$  apprend qu'il sait tout

### Petit pb à régler

Les sites n'ont pas en général la même excentricité,  
Ils ne terminent donc pas en même temps.

⇒ un site ne peut arrêter dès qu'il a fini (sinon interblocage)

Il va donc faire une phase en plus pour les autres

45 / 152

## Schéma Général des Algos à Phase (1/2)

### Données

Comme précédemment avec en plus

- *fini* : message envoyé – par un site sachant qu'il a terminé – aux sites qui sont plus excentrés que lui
- *canaux\_fermes<sub>i</sub>* : ensemble des sites d'excentricité moindre

### Init :

```
phi := 0;  
newi := {(di, i)};  
infi := {(di, i)};  
canaux_fermesi := ∅
```

### Term :

```
R = canauxi \ canaux_fermesi  
∀ r ∈ R : envoyer fini sur r  
∀ r ∈ R : recevoir m sur r
```

46 / 152

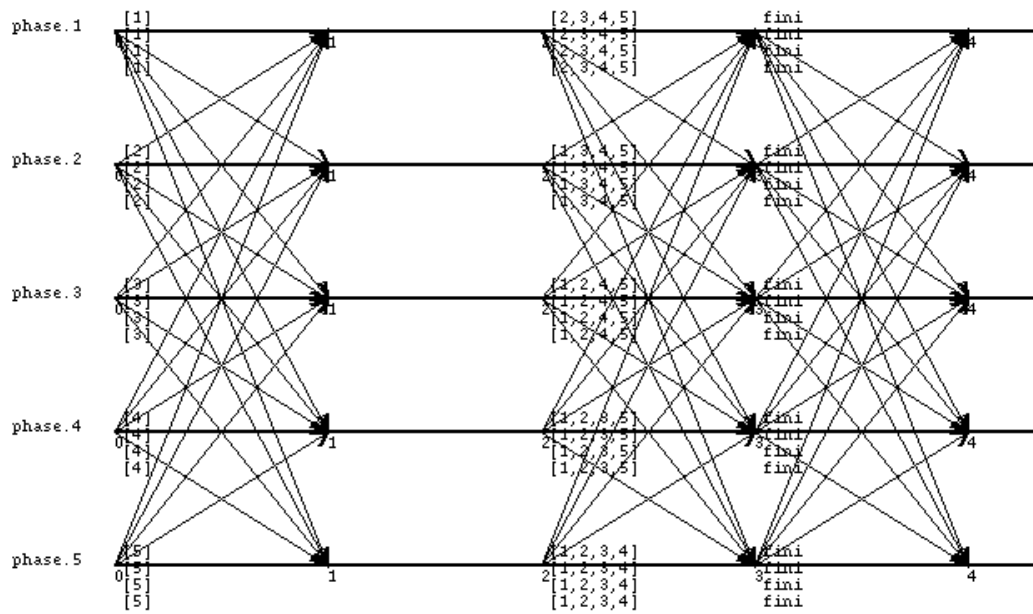
## Schéma Général des Algos à Phase (2/2)

### Phases\*

```
Tant que newi ≠ ∅ faire  
  phi := phi + 1;  
  ∀ c ∈ canauxi : envoyer newi sur c  
  newi := ∅  
  ∀ c ∈ canauxi  
    { recevoir m sur c  
      Si m = fini alors  
        canaux_fermesi := canaux_fermesi ∪ {c}  
      Sinon newi := newi ∪ (m \ infi)  
      "Calcul dépendant de l'algorithme spécifique"  
    }  
  infi := infi ∪ newi  
fin_tant_que
```

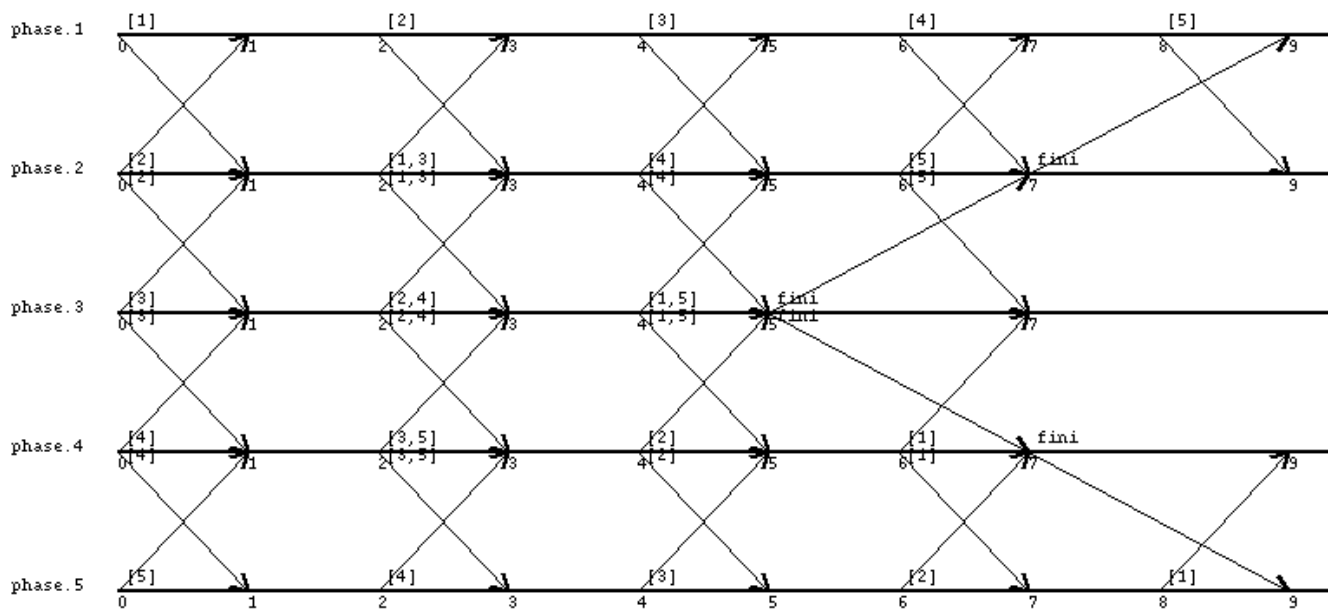
47 / 152

# Calcul phasé de table de routages sur une clique



48 / 152

# Calcul phasé de table de routages sur un bus



49 / 152

# Synchronisation par Vagues

Second schéma général d'algorithmes

Phases  $\approx$  Itération répartie

Vagues  $\approx$  Récursion répartie

## Plan

- 1 Exemple Introductif :  
Construction répartie d'une arborescence couvrante à partir d'un réseau connexe
- 2 Schéma général d'un Algorithme par vagues
- 3 Application à la détection de la terminaison distribuée

## Hypothèses Générales :

- Canaux bidirectionnels fiables,
- Site distingué (racine), un site ne connaît que ses voisins

51 / 152

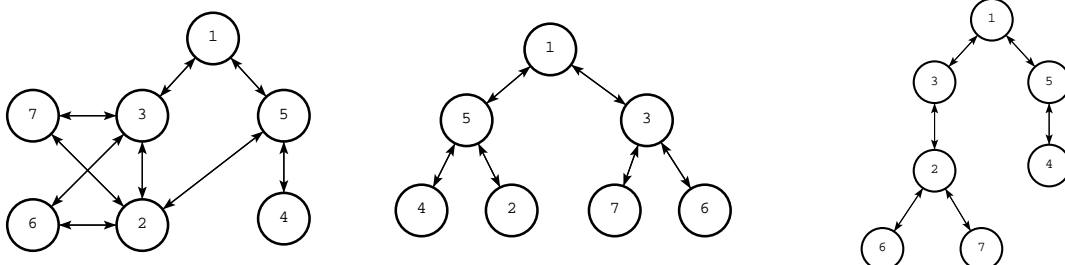
## Arbres couvrants (1/2)

### Arborescence couvrante (spanning tree)

Partant d'un graphe connexe, on construit **un** arbre

- contenant tous les sommets du graphe
- les arcs de l'arbre sont des arcs du graphe
- en général plusieurs arbres couvrants pour un même graphe

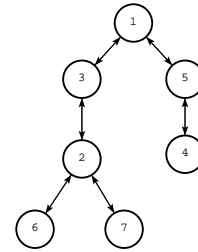
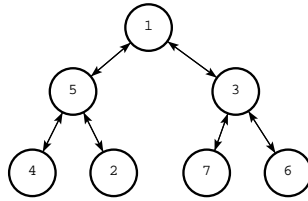
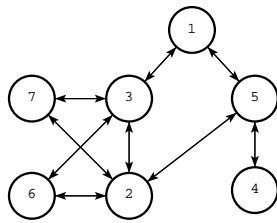
*Un graphe et deux de ses possibles arbres couvrants*



52 / 152



## Arbres couvrants (2/2)



### Intérêt de cette topologie

- ❶ Diffusion d'un message à tous les sites du réseau
  - $n - 1$  messages (Optimal)
  - Temps de propagation en  $2 \times p$   
( $p = \text{Log } n$  pour les arbres équilibrés)
- ❷ Exclusion Mutuelle (Quorums en  $\text{Log } n$ )  
(cf arbres dynamiques Algo de Naimi-Trehel)
- ❸ Détection répartie de la Terminaison

53 / 152

## Construction d'arbre couvrant : Principes (1/2)

### Racine

Seul processus initialement actif

Elle envoie un message *aller* à chacun de ses voisins (fils) et attend un message *retour* de chacun de ceux-ci

(vu de la racine, on a une **phase** de calcul)

### Vagues

Un site recevant un message  $\text{aller}(k)$  est atteint par la  $k^{\text{ième}}$  vague,

Un site renvoyant un message  $\text{retour}(k)$  termine la  $k^{\text{ième}}$  vague

### Construction de l'arbre

≡ Succession de phases synchronisées par la racine

A l'issue de la Phase  $n \mapsto$  arborescence des + courts chemins d'ordre  $n$

54 / 152

## Construction d'arbre couvrant : Principes (2/2)

### Notations :

$R$  la racine,

$P_i$  un site quelconque

$d_i$  la distance entre  $R$  et  $P_i$

### Invariants associés à l'algorithme

Pour tout site  $P_i$  et tout entier  $k$  ( $n^0$  de vague)

- $d_i > k \rightarrow P_i$  n'est pas atteint par la vague  $n^0 k$
- $d_i = k \rightarrow P_i$  va apprendre qu'il est dans l'arborescence en recevant message *aller*( $k$ ), Il connaîtra aussi son père (expéditeur du message) et sa profondeur ( $k + 1$ ).
- $d_i < k \rightarrow P_i$  connaît l'ensemble de ses fils dans l'arborescence

55 / 152

## Construction d'arbre couvrant : Données

### Messages Utilisés :

- *aller*( $k$ ) où  $k$  est le  $n^0$  de vague
- *retour*(*resp*) où *resp*  $\in \{Continuer, Termine, Deja\_marque\}$

### Connaissance Initiale :

- *voisins*(*\_lv*) : Voisins d'un site dans le réseau
- *privilege* : détenu uniquement par la racine
- *non\_marque*

### Données d'un Site :

- *vague*(*\_nv*) & *prof*(*\_prof*)
- *marque* : vrai si le site est dans l'arborescence – *pere*(*\_p*) : identité du père du site
- *fils*(*\_status*, *\_fils*) : ensemble des fils d'un site (*\_status*  $\in \{Prov, Def\}$ )
- *explore*(*\_nature*, *\_liste*) où *\_nature*  $\in \{wait\_fils, wait\_pere, termine\}$  et *\_liste* sous-ensemble des fils du site

56 / 152

## Construction d'arbre couvrant : algorithme (1/2)

Réception par le site  $P_i$  d'un message  $aller(k)$  – émis par  $P_j$

Site  $P_i$ , non-marqué

Il entre dans l'arborescence

Il apprend sa profondeur ( $k + 1$ ), son père ( $P_j$ )

Soit  $Fils = voisins - \{P_j\}$ .

Si  $Fils = \emptyset$  alors  $P_i$  renvoie à  $P_j$  *retour(termine)*

Sinon  $P_i$  renvoie à  $P_j$  *retour(Continuer)*

$F$  est l'ensemble **provisoire** des fils de  $P_i$

Site  $P_i$ , marqué, dont le père est  $\neq P_j$

$P_i$  renvoie à  $P_j$  *retour(deja\_marque)*

Site  $P_i$ , marqué, dont le père est  $P_j$

$P_i$  le propage à chacun de ses fils et passe en attente

57 / 152

## Construction d'arbre couvrant : algorithme (2/2)

Bilan d'une Vague

Site attend (*explore(wait\_fils, \_w)*) une réponse

*retour(resp)* de chacun des éléments de  $_w$

Il reçoit ces réponses et les traite  $\mapsto \langle \_cont, \_term, \_dm \rangle$

Si  $\_cont = \emptyset$  c'est fini pour lui.

Si c'est la racine, l'algo est terminé

Sinon, il renvoie à son père *retour(termine)*

Sinon

Si c'est la racine, il renvoie à  $\_cont$  *aller(\_Nv)*

Sinon, il renvoie à son père *retour(continuer)*

Gestion de ses Fils (*fils(\_status, \_f)*)

S'il connaissait déjà exactement ses fils ( $\_status = def$ )

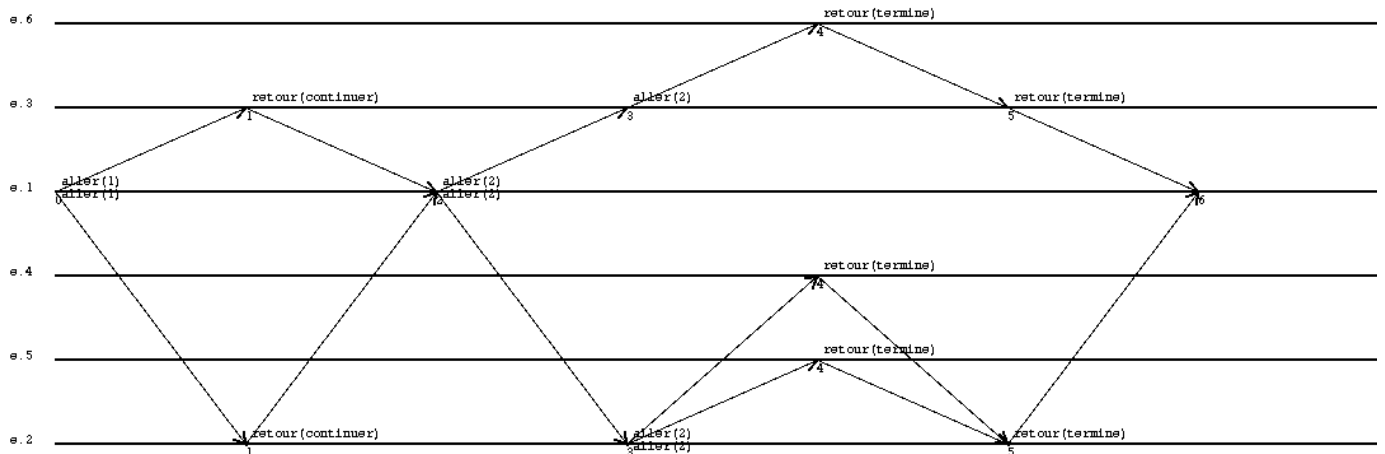
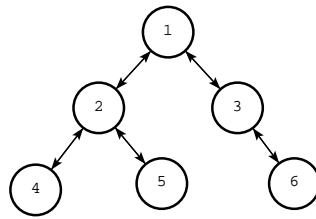
Alors pas de chgt

Sinon ( $\_status = prov$ )

$fils \leftarrow \_term + \_cont$  et  $\_status \leftarrow def$

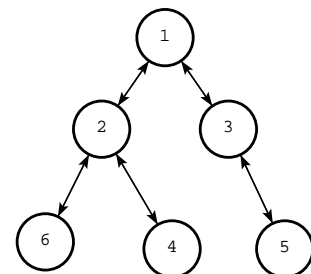
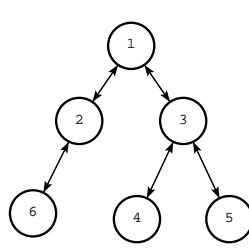
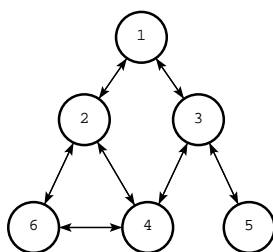
58 / 152

# Exemples d'exécution : Le graphe initial est déjà un arbre !!



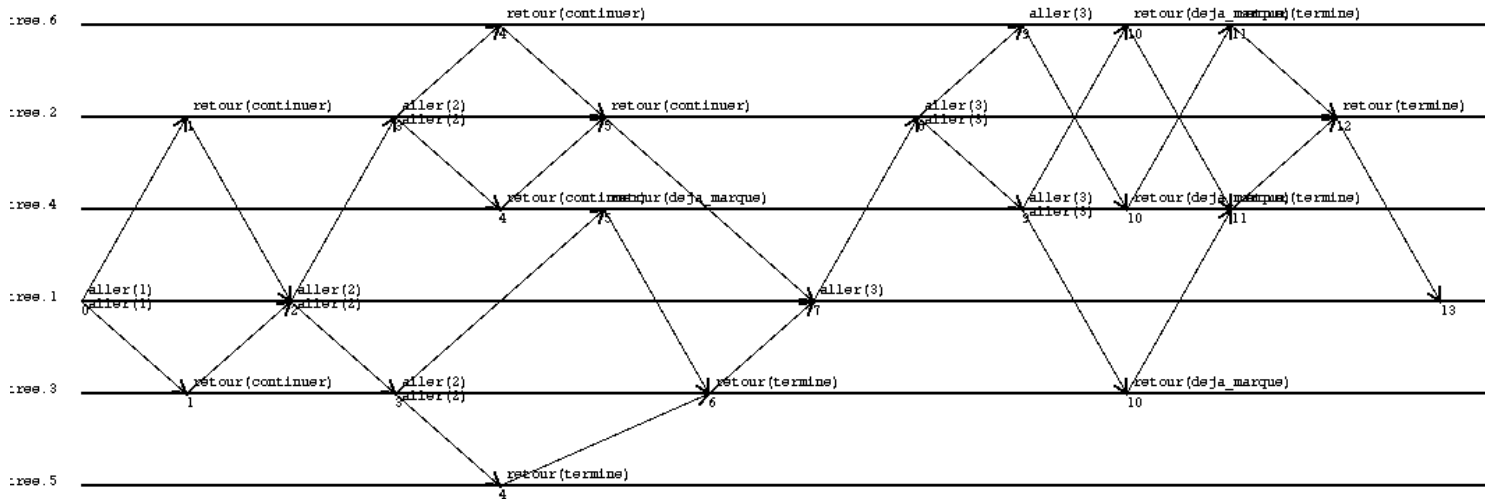
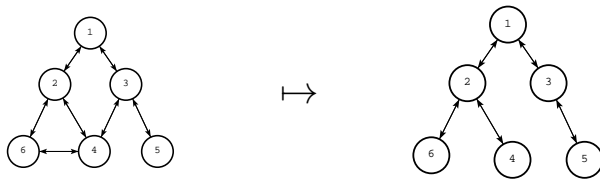
59 / 152

# Exemples d'exécution : Un graphe et 2 AC possibles (1/3)



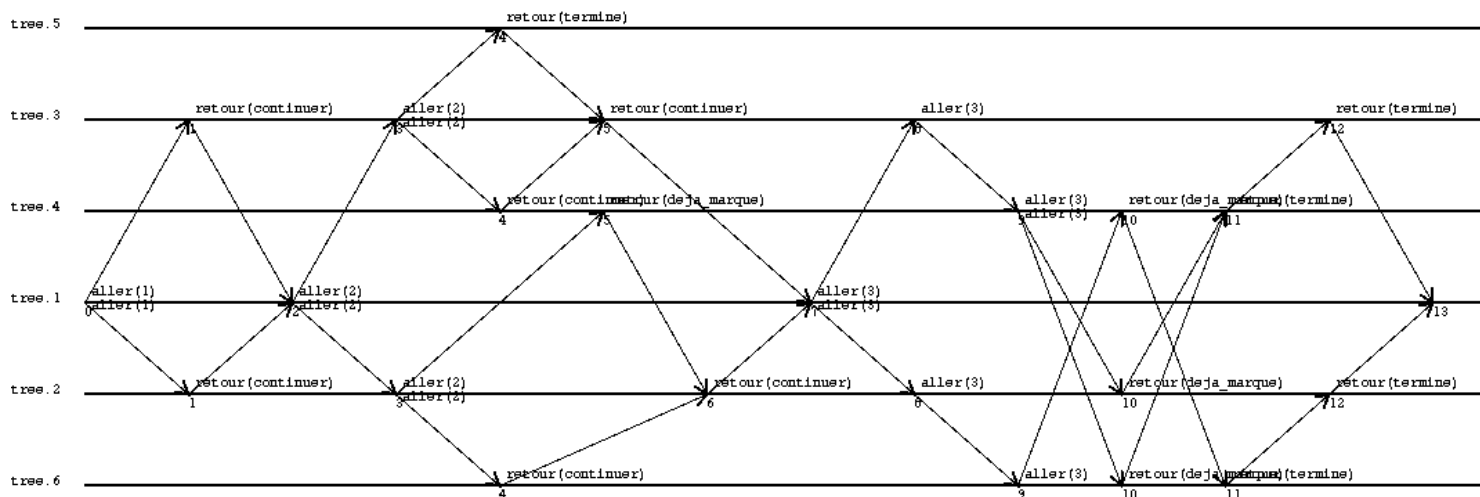
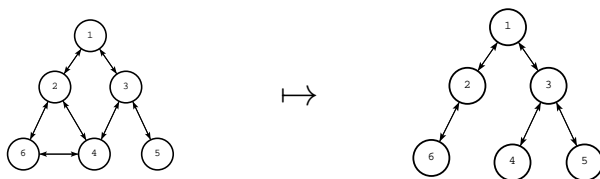
60 / 152

## Exemples d'exécution : une première solution/exécution (2/3)



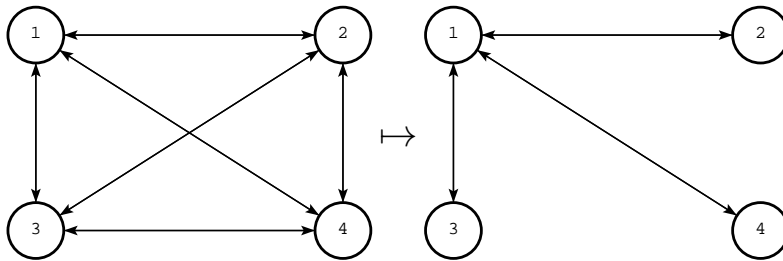
61 / 152

## Exemples d'exécution : une seconde solution/exécution (3/3)



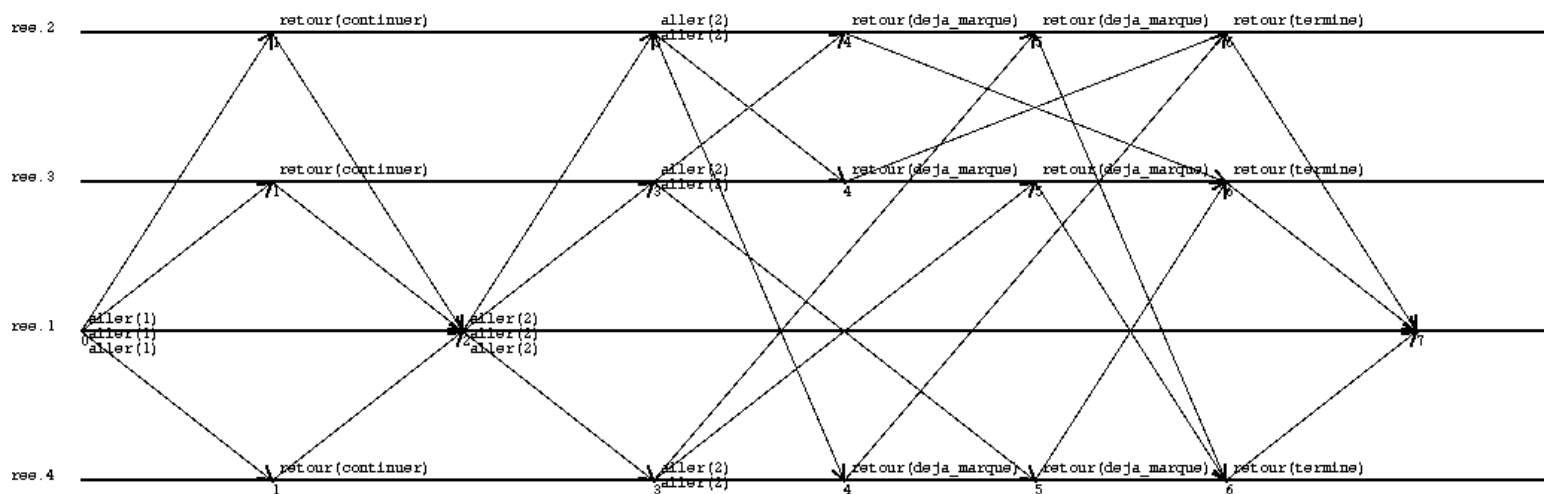
62 / 152

## Exemples d'exécution : Le graphe est complet (1/2)



63 / 152

## Exemples d'exécution : graphe complet (2/2)



64 / 152

# Schémas généraux d'algorithmes à vagues : Arborescence

## Site Quelconque $P_i$

```
j := 0 ;  
Repeter  
  attendre aller(diffuse)  
  recu := diffuse  
   $\forall f \in \text{Fils} : \text{envoyer aller(diffuse)} \text{ à } f$   
  res :=  $\emptyset$   
   $\forall f \in \text{Fils} :$   
    attendre retour(collecte) de f  
    res := res  $\cup$  collecte  
  j := j + 1  
  collecte := collecte  $\cup D$   
  envoyer retour(collecte) à pere  
Jusqu'à Condition de terminaison
```

65 / 152

# Schémas généraux d'algorithmes à vagues : Anneaux

## Initiateur de la Vague : $P_\alpha$

```
diffuse := valeur à diffuser  
recu := diffuse ; j := 1 ;  
envoyer jeton(diffuse,  $\emptyset$ ) à succ  
Repeter  
  attendre jeton(diffuse, collecte)  
  diffuse := F(collecte)  
  recu := diffuse ; j := j + 1 ;  
  envoyer à succ  
    jeton(diffuse, collecte)  
Jusqu'à Condition de terminaison
```

## SITE QUELCONQUE $P_i$

```
j := 0 ;  
Repeter  
  attendre jeton(diffuse, collecte)  
  recu := diffuse ; j := j + 1 ;  
  collecte := collecte  $\cup D$   
  envoyer à succ  
    jeton(diffuse, collecte)  
Jusqu'à Condition de terminaison
```

## Symétrique

```
Jeton( $\langle D_1, C_1 \rangle, \dots \langle D_n, C_n \rangle$ )
```

66 / 152

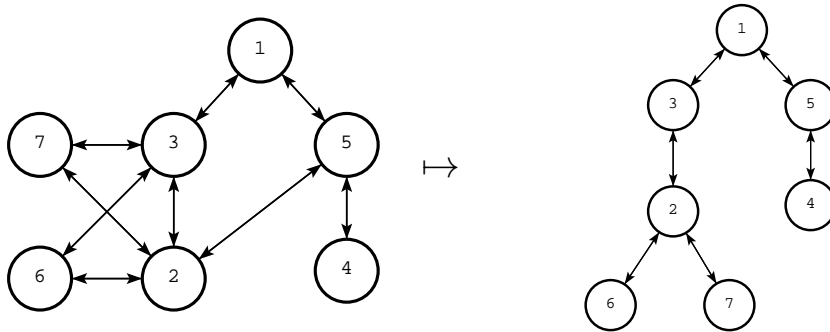
# "TD" : Calcul des tables de routages optimaux dans une arborescence couvrante

## Hypothèses

Communication bi-directionnelle fiable

Réseau connexe

Arborescence couvrante de ce réseau

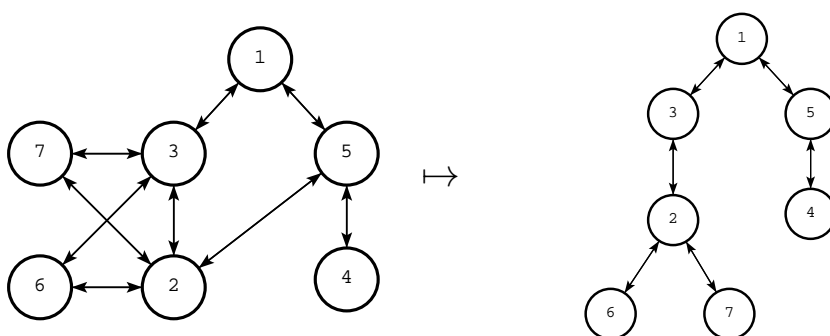


## Enoncé

Proposer un algorithme distribué permettant de construire pour chacun des sites les tables de routage optimaux associées à cette arborescence.

67 / 152

# "TD" : Calcul des tables de routages optimaux dans une arborescence couvrante : résultat attendu



Site	1	Site	2	Site	3	Site	4	Site	5	Site	6	Site	7
1	1	1	3	1	1	1	5	1	1	1	2	1	2
2	3	2	2	2	2	5	5	2	1	2	2	2	2
3	3	3	3	3	3	5	5	3	1	2	2	3	2
4	5	3	1	4	4	4	4	4	4	2	2	4	2
5	5	3	1	5	5	5	5	5	5	2	2	5	2
6	3	6	2	6	5	6	5	6	1	6	6	6	2
7	3	7	2	7	5	7	5	7	1	7	2	7	7

68 / 152



## "TD" : Calcul des tables de routages optimaux dans une arborescence couvrante : données (1/2)

### Représentation des tables de routage

$TR$ , ensemble de couples  $(x, y) \in Sites \times Sites$

avec la signification suivante :

Pour un site  $S$ ,  $(x, y) \in TR_S$  ssi le site  $S$  pour envoyer un message au site  $x$  doit l'envoyer au site  $y$ .

### Initialisation des tables :

Sur chaque site  $S$ , ayant pour père  $PS$  et possédant un ensemble de fils  $FS$

debut

$TR \leftarrow \emptyset$

Pour chaque  $e \in FS \cup \{S, PS\}$  :

$TR \leftarrow TR \cup \{(e, e)\}$

fin

### Convention

La racine est son propre père.

69 / 152

## "TD" : Calcul des tables de routages optimaux dans une arborescence couvrante : données (2/2)

### Connaissance initiale d'un site :

Chaque site connaît son père et ses fils directs dans l'arborescence.  
i.e la connaissance obtenue à l'issue de la construction de l'A.C

### Quatre états : *wait\_pere*, *wait\_fils*, *wait\_fin* et *termine*.

Les sites sont initialement dans l'état *wait\_pere*.

### Messages utilisés : (messages signés par leur Exp)

*debut\_calcul* déclenche le calcul des tables.

Il est envoyé initialement par la racine à chacun de ses fils et sera propagé vers les feuilles de l'arbre.

*Fils(ensemble\_de\_fils)* est envoyé par un noeud à son père pour lui indiquer l'ensemble de ses fils (au sens large)

*Sites(ensemble\_de\_sites)* est envoyé par la racine à tous ses fils pour leur indiquer l'ensemble de tous les sites de l'arborescence.

Ce message sera propagé dans tout l'arbre.

70 / 152

## "TD" : Calcul des tables de routages .... :

### Schéma Général : Flux

#### Principe

Les tables de routage peuvent être construites en une vague et demi :  
"1 vague pour que la racine obtienne toute la connaissance  
+ une demi-vague pour que le reste des sites ait aussi cette connaissance."

#### Flux : Racine $\mapsto$ Feuilles

- La racine, dans l'état *wait\_pere*, lance la première vague en envoyant le message *debut\_calcul* à chacun de ses fils, elle se place en attente de ses fils *wait\_fils*.
- Une Feuille, dans l'état *wait\_pere*, recevant *debut\_calcul* renvoie à son père le message *Fils*( $\emptyset$ ) et passe dans l'état *wait\_fin*.
- Un noeud intermédiaire, dans l'état *wait\_pere*, recevant *debut\_calcul* le propage à chacun de ses fils et passe en *wait\_fils*

71 / 152

## "TD" : Calcul des tables de routages .... :

### Schéma Général : Reflux (retour vers la racine)

#### Noeud Intermédiaire :

Un noeud *S*, dans l'état *wait\_fils*, ayant *FS* pour fils directs attend un message du type *Fils*(*ensemble\_de\_fils*) de chacun de ses fils.

A la réception, il exécute le calcul suivant :

$F \leftarrow \{S\} \cup FS$

Pour chaque fils  $f \in FS$  :

    attendre *Fils*(*ensemble\_de\_fils*) de *f*

$F \leftarrow F \cup \text{ensemble\_de\_fils}$

    Pour chaque  $e \in \text{ensemble\_de\_fils}$ :

$TR \leftarrow TR \cup \{(e, f)\}$

*fin*

Envoyer *Fils*(*F*) à son pere

Passer dans l'état *wait\_fin*

#### Racine :

Comportement identique : Par contre, au lieu d'envoyer à son père le message *Fils*(*F*), elle renvoie le message *Sites*(*F*) à chacun de ses Fils et passe dans l'état *termine*.

72 / 152

## "TD" : Calcul des tables de routages .... : Schéma Général : Fin de la demi-vague

**Comportement d'un site  $S$ , dans l'état *wait\_fin* à la réception du message  $Sites(ensemble\_de\_sites)$  venant de son père  $PS$ .**

### Noeud intermédiaire

Noeud  $S$  ayant  $PS$  pour père,

A sa réception, il exécute le calcul suivant :

Pour chaque  $s \in ensemble\_de\_sites$ :

Si  $(s, x) \notin TR$  Alors  $TR \leftarrow TR \cup \{(s, PS)\}$

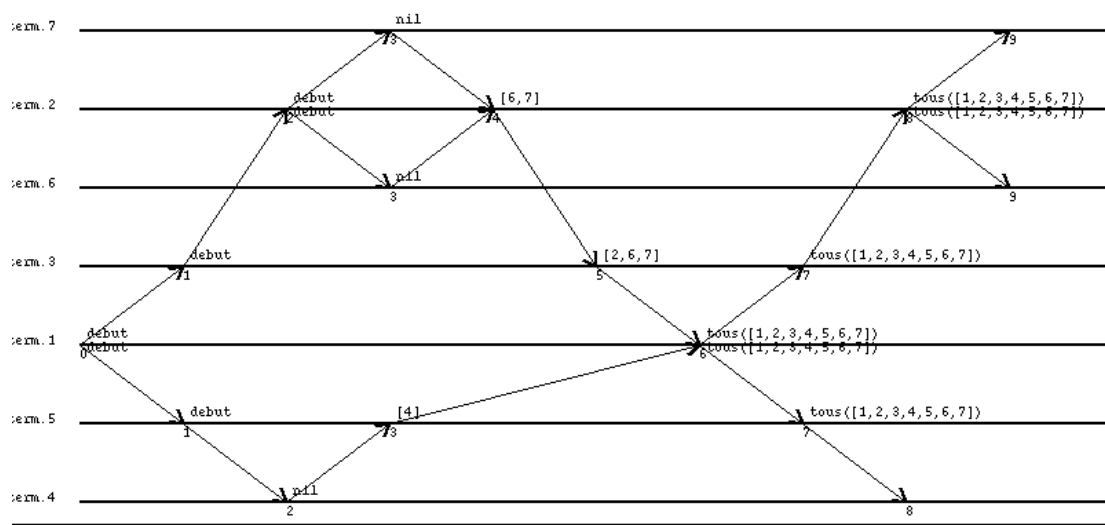
Il envoie ensuite le message  $Sites(ensemble\_de\_sites)$  et passe dans l'état *termine*.

### Feuille

Même traitement qu'un noeud intermédiaire, par contre passage direct dans l'état *termine*.

73 / 152

## "TD" : Calcul des tables de routages .... : Schéma Général : Exécution = 1 phase et demi



74 / 152

# "TD" : Calcul des tables de routages .... :

## Schéma Général : Exécution coté sites

### Etat des sites à l'issue de la première vague

```
term.1{[TR{[(1, 1), (2, 3), (3, 3), (4, 5), (5, 5), (6, 3), (7, 3)]}, fils{[3, 5]}, pere{1}, status{wait}}
term.2{[TR{[(2, 2), (6, 6), (7, 7)]}, fils{[6, 7]}, pere{3}, status{wait}}
term.3{[TR{[(2, 2), (3, 3), (6, 2), (7, 2)]}, fils{[2]}, pere{1}, status{wait}}
term.4{[TR{[(4, 4)]}, fils{nil}, pere{5}, status{wait}}
term.5{[TR{[(4, 4), (5, 5)]}, fils{[4]}, pere{1}, status{wait}}
term.6{[TR{[(6, 6)]}, fils{nil}, pere{2}, status{wait}}
term.7{[TR{[(7, 7)]}, fils{nil}, pere{2}, status{wait}}}
```

### Etat des sites à l'issue de l'algorithme

```
term.1{[TR{[(1, 1), (2, 3), (3, 3), (4, 5), (5, 5), (6, 3), (7, 3)]}, fils{[3, 5]}, pere{1}, status{termine}}
term.2{[TR{[(1, 3), (2, 2), (3, 3), (4, 3), (5, 3), (6, 6), (7, 7)]}, fils{[6, 7]}, pere{3}, status{termine}}
term.3{[TR{[(1, 1), (2, 2), (3, 3), (4, 1), (5, 1), (6, 2), (7, 2)]}, fils{[2]}, pere{1}, status{termine}}
term.4{[TR{[(1, 5), (2, 5), (3, 5), (4, 4), (5, 5), (6, 5), (7, 5)]}, fils{nil}, pere{5}, status{termine}}
term.5{[TR{[(1, 1), (2, 1), (3, 1), (4, 4), (5, 5), (6, 1), (7, 1)]}, fils{[4]}, pere{1}, status{termine}}
term.6{[TR{[(1, 2), (2, 2), (3, 2), (4, 2), (5, 2), (6, 6), (7, 2)]}, fils{nil}, pere{2}, status{termine}}
term.7{[TR{[(1, 2), (2, 2), (3, 2), (4, 2), (5, 2), (6, 2), (7, 7)]}, fils{nil}, pere{2}, status{termine}}}
```

75 / 152

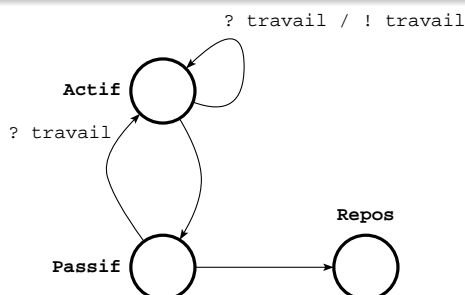
## Terminaison Répartie : description du problème

### Comportement des sites/agents régit par 3 états

Passif : *L'agent attend qu'on lui confie du travail*

Actif : *L'agent effectue le travail qui lui a été confié*

Repos : *L'agent a cessé le travail*



"Boucle sur Actif" :  
un agent actif peut se  
décharger sur ses voisins

### Problème de la terminaison

*Comment un agent peut-il détecter que l'on ne lui  
confiera plus de travail et qu'il peut se mettre au repos ?*

77 / 152

## Terminaison Répartie : difficulté du problème

### Problème de la terminaison (suite)

*Comment un agent peut-il détecter qu'il a terminé ...*

Pb rencontré dans tous les **calculs** !

par ex lors du calcul des tables de routage

nb : Terminaison  $\equiv$  ramasse-miettes

### A trouver une condition de terminaison ...

**Nécessaire et Suffisante** et **Locale**

### Recherche d'une C.N.S

**C1 : Tous les agents sont passifs**

Nécessaire oui

Suffisant ? **Non** !

**C2 : Aucune requête de travail n'est en transit**

$CNS = C1 \wedge C2$

C.N.S **globale** : état des agents et des voies de communication !!

78 / 152

## Terminaison Répartie : $\neq$ cas de figure

### Bonne terminaison

*Un algorithme sera dit bien terminé si l'on atteint **inévitablement** un état global du système où :*

- *Tous les agents sont au repos*
- *Il n'y a pas de requête de travail en transit*

### Terminaison non détectée

*Le système atteint un état où tous les agents sont passifs alors qu'il n'y a pas de requête en transit*

$\approx$  Interblocage

### Détection erronée de la terminaison

*Le système atteint un état où tous les agents sont au repos alors qu'il y a encore des requêtes en transit*

$\approx$  Blocage

79 / 152

# Terminaison Répartie : Algorithme de détection par vagues

## Principe général

*Superposer* à l'application – dont on veut assurer la bonne détection de la terminaison – un algorithme de détection de la Terminaison

**Pb** : Veiller à éviter les interférences entre les deux algorithmes.

## Solution par "Vagues"

- ❶ Construire une arborescence couvrante
- ❷ L'application utilisera la topologie de l'AC
- ❸ Lorsque la racine est passive, elle lance une vague de détection

Hypothèses habituelles : Canaux fiables & **fifo**

80 / 152

# Terminaison Répartie : Algorithme (1/4)

## Etats d' un agent

**Passif** : site ne fait rien

**Actif** : site travaille

**Wait** : site informé qu'une vague de détection de la terminaison est en cours ; il attend le résultat

**Terminé** : site sachant que la terminaison est détectée

## Flux : Messages Racine $\mapsto$ Feuilles

**Fini ?** : pour initier/propager une vague de détection de la terminaison

**Terminé** : pour signifier que la terminaison a été détectée

## Reflux : Feuilles $\mapsto$ Racine

**Pas\_Fini** : pour signifier la non terminaison

**Ok\_Fini** : pour signifier la terminaison "locale"

81 / 152

## Terminaison Répartie : Algorithme (2/4)

### Lancement d'une vague

La racine **passive** lance une vague de détection : elle envoie le message *Fini ?* à chacun de ses fils et passe dans l'état *wait*.

### Terminaison d'un site

Un site passif recevant le message *termine* de son père sait que la terminaison est atteinte, il en informe ses fils et passe à l'état **termine**

### Réception d'une vague

Un site reçoit le message **Fini ?** de son père  
Si il est passif alors il le propage à ses fils  
et passe en attente de ses fils  
Sinon il reste actif et  
renvoie le message *pas\_fini* à son père

82 / 152

## Terminaison Répartie : Algorithme (3/4)

### Bilan d'une vague

Un site ayant propagé une vague de terminaison reçoit les réponses de chacun de ses fils

$Mesg \in \text{Messages\_Application} \cup \{\text{ok\_fini}, \text{pas\_fini}\}$

Il les traite  $\mapsto \langle \text{work}, \text{passif}, \text{actif} \rangle$

**Si**  $\text{work} = \emptyset$  &  $\text{actif} = \emptyset$

**Alors**  
Succès

Si c'est la racine  
alors l'application est terminée,  
elle renvoie le message **termine** à chacun de ses fils  
et termine elle-même  
Sinon il renvoie le message **ok\_fini** à son père et attend

**Sinon**  
Echec

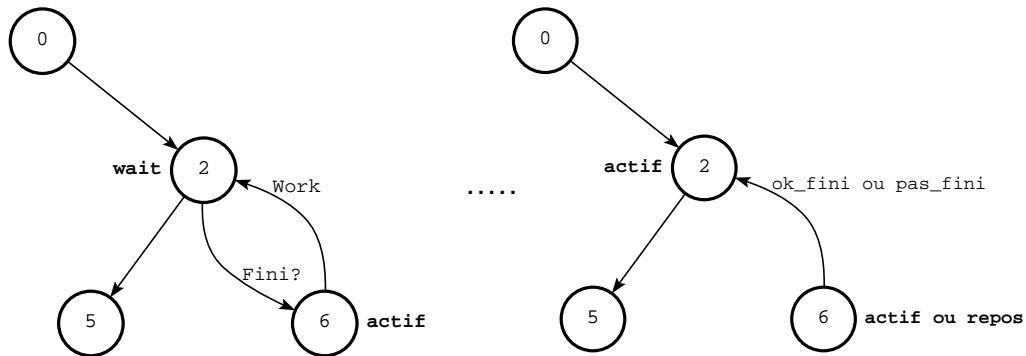
Le site repasse en **actif** ou **passif**  
suivant que **work** est vide ou non  
Si c'est la racine alors **rien**  
Sinon le site renvoie le message *pas\_fini* à son père

83 / 152

## Terminaison Répartie : Algorithme (4/4)

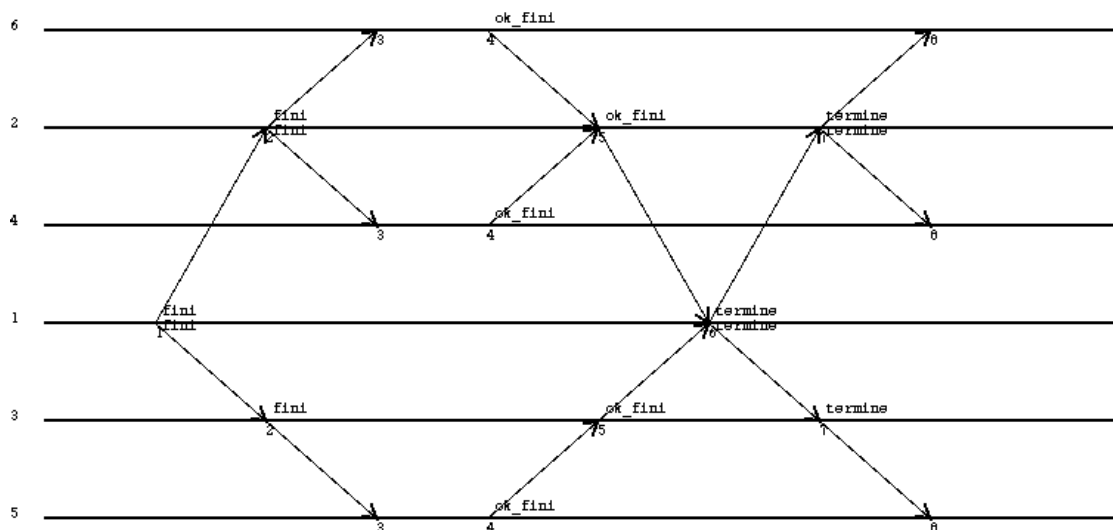
## Interférences entre la terminaison et l'application

Croisement entre une requête de travail et une vague de détection  
Le site 2 ayant lancé une vague reçoit un message de l' application de la part de son fils 6, il recevra **ensuite** la réponse de celui-ci quant à sa propre terminaison ; A ce moment la, le site 2 n' est plus dans l'état wait, le message de 6 doit donc être ignoré.



84 / 152

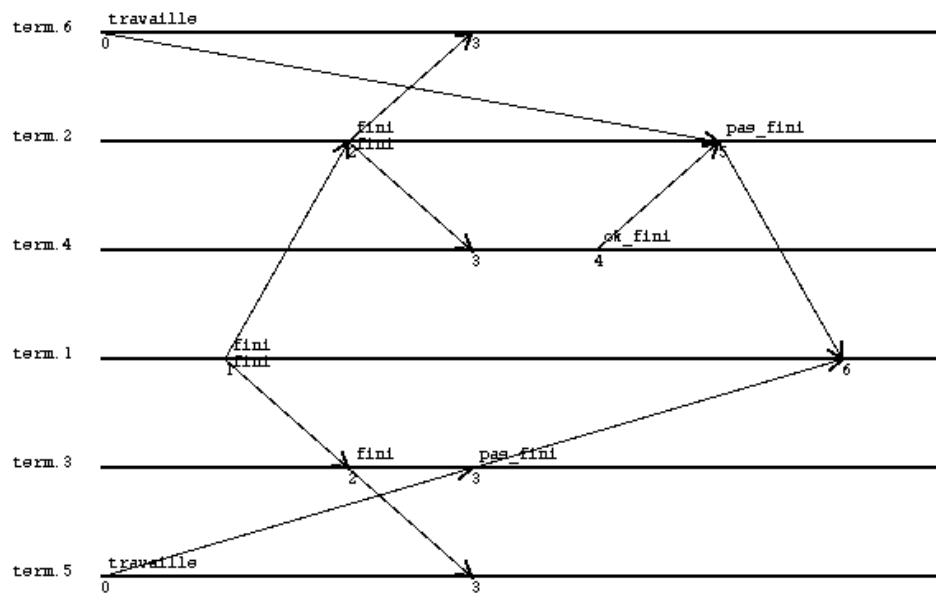
# Terminaison Répartie : Succès



85 / 152

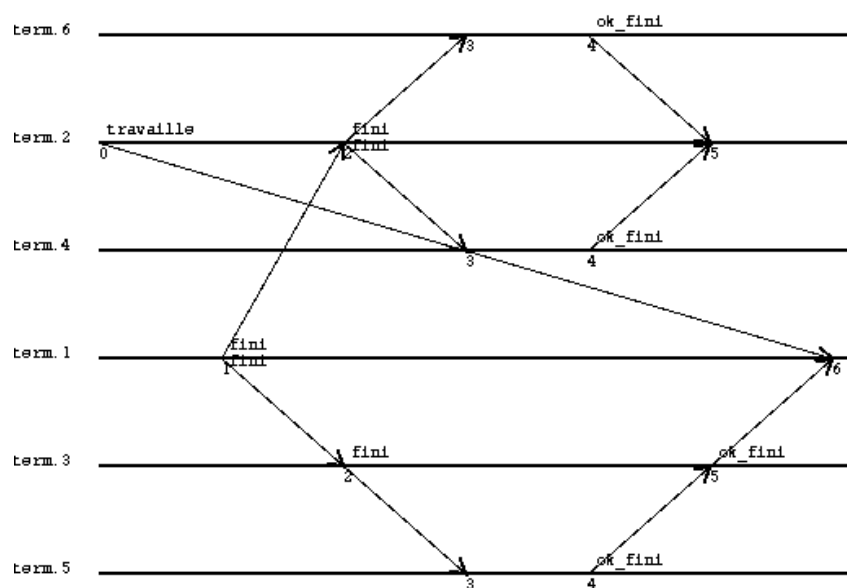


## Terminaison Répartie : Echech (1/2)



86 / 152

## Terminaison Répartie : Echech (2/2)



87 / 152

# Tirage au Sort Réparti

## ↪ Mécanisme Distribué pour résoudre des Conflicts

**Conflicts sont résolus**

de manière équitable

avec un résultat **imprédictible**

## ↪ façon Symétrique de casser la Symétrie

**Alternative aux**

Priorités statiques

Messages Datés (Horloges)

89 / 152

# Tirage au Sort Réparti

## Algorithmes Symétriques

- *chaque site exécute le même code*
- *aucun site ne bénéficie d'un privilège connu statiquement*
- *chaque site est impliqué également dans le résultat final*

## Symétrie d'un tirage au sort ?

Degré de symétrie évalué en termes de Probabilités

## Propriétés attendues du tirage au sort

- 1 Probabilité pour un site de gagner soit **indépendante** de ses **choix** de son **identificateur**
- 2 Résultat imprédictible

90 / 152

# Tirage au Sort Réparti : 3 Algorithmes

## Algo #1

- 1 Election d'un vainqueur entre les  $n$  Agents

## Algo #2

Sélection d'un vainquer parmi  $p$  compétiteurs dans une configuration globale de  $n$  Agents.

- 1 Détermination des Compétiteurs
- 2 Election d'un vainqueur entre les compétiteurs

## Algo #3

Obtention d'un Ordre total entre  $p$  compétiteurs dans une configuration globale de  $n$  Agents.

- 1 Détermination des Compétiteurs
- 2 Obtention d'un Ordre total entre les compétiteurs

Pour les 3 algos, chaque site doit connaître le résultat.

91 / 152

## Plan

- Hypothèses
- Schéma Général
- Propriétés Attendues
- 2 Algorithmes de tirage au sort
- Ordre Total
- Conclusion

92 / 152

## Communication

- $M_1$  : Chaque site peut communiquer avec les autres
- $M_2$  : Délais de transmission sont finis
- $M_3$  : Médium de transmission est fiable

## Sites

- $A_1$  : Chaque site connaît tous ses partenaires  
Soit  $Agents = \{a_1, a_2, \dots, a_n\}$
- $A_2$  : Chaque site est identifié par un identificateur  
Soit  $Code$  la bijection de  $\{a_1, a_2, \dots, a_n\}$  sur  $\{1, \dots, n\}$
- $A_3$  : Chaque site connaît un ensemble de valeurs noté  $Choix$

93 / 152

# Schéma général des 3 algorithmes

## Communication **réduite** à une phase

Chaque site

- choisit **aléatoirement** une valeur  $\in Choix$
- l'envoie à chacun de ses partenaires
- attend leurs réponses

**Complexité** :  $O(n^2)$  messages

## A l'issue de la phase

Un **même** ensemble (**EV**) des  $n$  valeurs échangées  
est **connu** par **tous** les sites

## Calcul Local

Election	< – >	Vainqueur :	$Choix^n \mapsto Competiteurs$ $EV \mapsto Vainqueur(EV)$
Ordre	< – >	bijection	$B_{EV} : Competiteurs \mapsto \{1, \dots, c\}$ $B_{EV} \mapsto <_{B_{EV}} \text{ Ordre Canonique}$

94 / 152

# Propriétés des algorithmes

## Qualitatives

**Terminaison** de l'algorithme dans un état **consistant**

- Un site et un seul gagne, les autres perdent
- Le vainqueur est un compétiteur

Tous les sites connaissent le vainqueur

## Probabilistes

- Equiprobabilité **faible** de gagner pour chacun des sites :  
*Chaque site à la même chance de gagner*  
En posant  $P(I)$  : probabilité de gain pour le site  $I$   
(P1)  $\forall I, J \in \text{Agents} : P(I) = P(J)$
- Equiprobabilité **forte** de gagner pour chacun des sites :  
*Chaque site est traité de la même façon : Ses chances de gain sont indépendantes : de son identificateur ou de la valeur choisie*  
En posant  $P(I, v)$  : probabilité que  $i$  gagne avec la valeur  $v$   
(P2)  $\forall I, J \in \text{Agents}, \forall v, w \in \text{Choix} : P(I, v) = P(J, w)$

95 / 152

## Une solution avec deux sites

### Données

$\text{Agents} = \{A, B\}$   
 $\text{Choix} = \{v1, v2\}$

### Détermination du vainqueur

$\text{Vainqueur} : \text{Choix}^2 \mapsto \text{Agents}$

$\text{Vainqueur}((x, y)) = \begin{cases} A & \text{iff } x = y \\ B & \text{otherwise} \end{cases}$

### Distribution des valeurs

$\{v1, v2\}^2$	Vainqueur
(v1,v1)	A
(v1,v2)	B
(v2,v1)	B
(v2,v2)	A

### Propriétés d'équiprobabilité satisfaites

$P(A) = P(B) = 1 / 2$

$P(A, v1) = P(A, v2) = P(B, v1) = P(B, v2) = 1 / 4$

96 / 152

# Algo #1 : Tirage au sort d'un vainqueur entre $n$ compétiteurs

## Données

On se donne arbitrairement une Bijection *Code*

$Code : Agents \mapsto \{1, \dots, n\}$  (*Compétiteurs* = *Agents*) *Choix* =  $\{1, \dots, n\}$

## Détermination du vainqueur

Calcul annexe

Soit  $w : \{1, \dots, n\}^n \mapsto \{1, \dots, n\}$ , l'application

$$w((a_1, \dots, a_n)) = 1 + \left[ \left( \sum_{j=1}^{j=n} a_j \right) \bmod (n) \right]$$

Calcul principal

Soit  $Vainqueur : \{1, \dots, n\}^n \mapsto Agents$

$Vainqueur((a_1, \dots, a_n)) = A$  ssi  $Code(A) = w((a_1, \dots, a_n))$

97 / 152

# Algo #1 : Cas de deux sites

## Choix de la bijection *Code*

$Agents = \{A, B\}$

et on pose **arbitrairement**  $Code(A) = 1$   $Code(B) = 2$

## Résultats

$\{1, 2\}^2$	$w$	Vainqueur
(1,1)	1	A
(1,2)	2	B
(2,1)	2	B
(2,2)	1	A

## Probabilités

$P(A) = P(B) = 1 / 2$

$P(A,1) = P(A,2) = P(B,1) = P(B,2) = 1 / 4$

Ces probabilités sont bien indépendantes du choix de la bijection *Code*

98 / 152

## Algo #1 : Éléments de preuve

Propriété d'Equiprobabilité forte :  $P(i, v) = P(j, w)$

$Card(Choix) = Card(Agents) = n$

Soit  $A_n^n$ , l'ensemble de tous les arrangements, on a  $Card(A_n^n) = n^n$

$C(i, v)$  : l'ensemble des arrangements où  $i$  choisit  $v$  et gagne.

$P(i, v) = Card(C(i, v)) / Card(A_n^n)$

$Card(C(i, v)) = Card(A_n^{n-2}) = n^{n-2}$  **A montrer !**

$\Rightarrow P(i, v) = n^{n-2} / n^n = 1/n^2$  (indépendant de  $i$  et de  $v$ )

$Card(C(i, v)) = n^{n-2}$

Lemme : Soit  $(a_1, \dots, a_{n-1})$ ,  $n-1$  valeurs  $\in A_n^{n-1}$ ,  $\forall A \in Agents$ ,

$\exists$  un unique  $a_n \in \{1, \dots, n\} : Vainqueur((a_1, \dots, a_{n-1}, a_n)) = A$

$$\text{preuve : } a_n = [\text{Code}(A) - (1 + \sum_{k=1}^{n-1} a_k)] \bmod (n)$$

Finalement,  $\forall i \in Agents, \forall v \in \{1, \dots, n\}, \forall (a_1, \dots, a_{n-2}) \in A_n^{n-2}$

$\exists$  a unique  $a_n \in \{1, \dots, n\} : Vainqueur((a_1, \dots, a_{n-2}, v, a_n)) = i$

99 / 152

## Algo #2 : Tirage au sort parmi $c$ entre $n$ sites (1/2)

Solution triviale :  $2 * n^2$  messages

– On fait une première phase pour connaître les compétiteurs

– On renumérote les compétiteurs et on applique l'algo #1

Défi : résoudre algo #2 au même coût qu'algo #1

Inchangé

Pour  $Card(Agents) = n$ , on dispose de Code bijection de  $Agents \mapsto [1, n]$

1<sup>ères</sup> Modifications

$Choix = \{0\} \cup \{1, \dots, n!\}$  ( **$n!$  discuté plus tard**)

Ajout de la valeur 0 pour les non compétiteurs

Détermination des Compétiteurs

On note toujours EV, l'ensemble des valeurs échangées

$Competiteurs = \{A \in Agents : v_A \neq 0\}$

On note  $c = Card(Competiteurs)$

100 / 152

## Algo #2 : Tirage au sort parmi $c$ entre $n$ sites (2/2)

### Détermination du vainqueur

$Vainqueur : \{1, \dots, n!\}^n \mapsto \text{Competiteurs}$

$$w((a_1, \dots, a_n)) = 1 + \left[ \left( \sum_{j=1}^n a_j \right) \bmod (c) \right]$$

$Vainqueur((a_1, \dots, a_n)) = C$  iff **Ecode(C)** =  $w((a_1, \dots, a_n))$

Où **Ecode** :  $\text{Competiteurs} \mapsto \{1, \dots, c\}$

$Ecode(C) = Code(C) - \text{Card}(\{P \in \text{Passif} : Code(P) < Code(C)\})$

Exemple avec 5 sites dont 3 compétiteurs seulement

$EV = \langle 0, 2, 3, 0, 1 \rangle$  &  $w = 1 + (6 \bmod 3) = 1$

Agents	A1	A2	A3	A4	A5
Code	1	2	3	4	5
Values	0	2	3	0	1
Compétiteur	non	oui	oui	non	oui
Ecode	/	1	2	/	3
Vainqueur	/	oui	/	/	/

101 / 152

## Algo #2 : Éléments de preuve

### Propriété d'Equiprobabilité forte

$\text{Competiteurs} = \{c_1, \dots, c_c\}$      $\text{Card}(\text{Competiteurs}) = c$

On note  $P(\text{Competiteurs}, I, V)$ , la probabilité pour que  $I$  gagne avec la valeur  $V \in \{1, \dots, n!\}$

$P(\text{Competiteurs}, I, V) = 1/(n! \times c)$

### Choix de $n!$

On va compter le nombre d'éléments de  $C_{\uparrow \langle \text{Competiteurs} \rangle}$  de deux façons

(1)  $\text{Card}(C_{\uparrow \langle \text{Competiteurs} \rangle}) = c \times \text{Card}(C_{\uparrow \langle \text{Competiteurs} \rangle}(c_i))$   
(A cause de la Pté d'équiprobabilité faible)

(2)  $\text{Card}(C_{\uparrow \langle \text{Competiteurs} \rangle}) = B^c$     ( $B = \text{Card}(\text{Choix})$ )

(1) & (2)  $\Rightarrow B^c = c \times \text{Card}(C_{\uparrow \langle \text{Competiteurs} \rangle}(c_i))$   
ie  $B^c$  doit être un multiple de  $c$  pour chaque  $c \in \{1, \dots, n\}$   
d'où  $\text{Card}(B) = n!$  (ou plus généralement  $\text{Ppcm}(1, \dots, n)$ )

102 / 152



## Algo #3 : Obtention d'un Ordre Total entre les compétiteurs (1/2)

### Solution triviale

Appliquer 1 fois Algo #2 puis  $c$  fois Algo #1 où  $c$  est le nombre de compétiteurs  
Coût factoriel

### Solution retenue

#### Itérer l'élection avec les mêmes messages échangés

Le premier site dans l'ordre est le site vainqueur (au sens de algo #2)

Après cela, les autres compétiteurs le considèrent comme passif.

Un nouveau vainqueur est déterminé avec les valeurs restantes pour obtenir le second dans l'ordre, etc ...

Coût - en terme de communication - inchangé wrt Algo #2

103 / 152

## Algo #3 : Ordre Total entre les compétiteurs (2/2)

### Principe sur un exemple ....

Agents	A1	A2	A3	A4	A5
Valeurs	0	2	3	0	3
$f_2(\text{Valeurs})$	0	2	3	0	0
$f_3(\text{Valeurs})$	0	2	0	0	0

Ordre associé :  $A5 < A3 < A2$

B	
1	A5
2	A3
3	A2

### ... ou plus formellement

$$B(1) = S_i \text{ ssi } w(v_1, v_2, \dots, v_n) = \text{Encode}(S_i)$$

$$B(k) = S_i \text{ ssi } w(f_k(v_1), f_k(v_2), \dots, f_k(v_n)) = \text{Encode}(S_i)$$

où les  $f_k$  ( $2 \leq k \leq c$ ) sont définis comme suit :

$$f_k(v_i) = 0 \text{ ssi } v_i = 0 \text{ ou } \exists p < k \text{ tel que } B(p) = S_i$$

$$f_k(v_i) = v_i \text{ sinon}$$

### Propriétés

$B$  est bijective

On définit  $<_b$  par  $X <_b Y$  ssi  $b(X) < b(Y)$

$<_b$  est un ordre total strict sur *Competiteurs*

104 / 152

# Gestion des Données Dupliquées

## Données dupliquées

Sites	$S_1$	$S_2$	$\dots S_n$
Donnée $D$	$D_1$	$D_2$	$\dots D_n$

## Objectif

Améliorer les performances en lecture (//)  
la résistance aux défaillances

Prix à payer : Assurer la cohérence mutuelle des copies

## Paradigme des lecteurs/écrivains – 2 Opérations : *lire(d)*, *ecrire(d)*

$R_1$  : Toute exécution de *ecrire(d)* exclut toute autre opération

$R_2$  : Les exécutions de *lire(d)* peuvent être simultanées

$R_3$  : La valeur rendue par *lire(d)* est la dernière valeur produite par *ecrire(d)*

A suivre : Différents algorithmes  
permettant de limiter la complexité (message)

106 / 152

## Algo #1 "Write all, Read one"

### Principe de base

- Tout site lit sa copie de façon indépendante
- Pour écrire un site doit :
  - (a) demander leur permission à **tous** les autres.
  - (b) leur communiquer ensuite la nouvelle valeur.

### Propriétés

$R_1, R_2$  &  $R_3$  sont trivialement satisfaites.

**Interblocage possible** ( $\leftarrow$  Horloges de Lamport, Tirage au sort, ...)

### Caractéristiques

Intéressant si

- (1) Lecture/Ecriture est grand
- (2) Peu de défaillances

**défaillance d'un site :**

- au minimum interdit toute écriture
- dans le pire des cas, peut être fatale au système

107 / 152

## Algo #2 : Algorithmes par Vote (consensus) (1/2)

### Principe de base

- Un site pour lire doit obtenir la permission de  $R$  sites
- Un site pour écrire doit obtenir la permission de  $W$  sites.  
Il les libère après l'écriture

### Conditions sur $R$ et $W$ pour assurer $R_1$ et $R_2$

$C_1 : R + W > N$  &  $C_2 : 2 \times W > N$   
où  $N$  est le nombre total de sites

### Pour assurer $R_3$ : n° de versions

A chaque copie  $D_i$  de  $D$  est associée un n° de version

$S_i : D_i \mapsto NV_i(D_i)$

La copie "courante" doit être affectée par le plus grand n°

NB : Il peut y avoir plusieurs copies ayant le même n°

**Invariant** :  $NV_i(D_i) = NV_j(D_j) \Rightarrow D_i = D_j$

108 / 152

## Algo #2 : Algorithme (1/2)

### Principe général

**Lecture** : Un site demande la permission et "leur" valeur à  $R$  sites  
Il prend en compte la valeur associée au plus grand n°

**Ecriture** : Même chose que pour lecture mais à  $W$  sites  
Il produit la nouvelle valeur  $D'$  et son n°  $N'$   
Il communique aux  $W$  sites  $(D', N')$

### Données

Etat d'un agent = *idle* | *frozen* | *wait*

Variable :  $\langle NV, D \rangle$

Messages : *reqLecture*, *reqEcriture*,  $\langle NV, D \rangle$ , *Lib*( $D, NV$ )

### Autorisation d'un site

Site idle reçoit une requête (lecture/écriture)

Il renvoie  $\langle NV, D \rangle$  et devient frozen

109 / 152

## Algo #2 : Algorithmme (2/2)

### Libération d'un site

Réception de  $Lib(D', N')$

$D \leftarrow D'; NV \leftarrow N'$ . Le site passe de frozen à idle

### Lire(D) : Sur un site $S_i$ , choisir un ensemble $L$ de $R$ sites

$\forall s \in L$  : envoyer à  $s$  requete\_lecture

Attendre les  $R$  réponses  $\langle NV_l, D_l \rangle$

Soit  $nv_{max} = \text{Max}\{NV_l : l \in L\}$  et  $D'$  la donnée associée

$D \leftarrow D'; NV \leftarrow nv_{max}$

$\forall s \in L$  : envoyer  $Lib(NV, D)$  à  $s$

### Ecrire(D) : Sur un site $S_i$ , choisir un ensemble $E$ de $W$ sites

$\forall s \in E$  : envoyer à  $s$  requete\_ecriture

Attendre les  $W$  réponses  $\langle NV_l, D_l \rangle$

Soit  $nv_{max} = \text{Max}\{NV_l : l \in L\}$  et  $D'$  la donnée associée

$D \leftarrow F(D'); NV \leftarrow nv_{max} + 1$

$\forall s \in W$  : envoyer  $Lib(D, NV)$  à  $s$

110 / 152

## Algo #2 : Exemple d'Exécution

"Quorums" fixes : on fixe ici arbitrairement les ensembles  $Q_R$  &  $Q_W$

$S = \{A, B, C, D\}, N = 4$

On choisit :  $W = 3, R = 2$

Sites	A	B	C	D
$Q_R$	{A,B}	{B,C}	{C,D}	{D,A}
$Q_W$	{A,B,C}	{B,C,D}	{C,D,A}	{D,A,B}

nb : Chaque site appartient à ses quorums de lecture et d'écriture

$\mapsto$  un message économisé

### Etat Initial

	A	B	C	D
$D$	d	d	d	d
$n^o$	0	0	0	0

111 / 152

## Algo #2 : Exemple d'exécution (suite)

### A veut écrire

$\forall s \in Q_w(A)$  : envoyer à  $s$  requete.lecture  
 Attendre les réponses  
 $Resp = \{(B, 0, d), (C, 0, d)\} \cup \{(A, 0, d)\}$   
 $Max(n^o) := 0; Vc(D) := d$   
 $d' := F(d); nv := Max(n^o) + 1$   
 $\forall s \in Q_w(A)$  : envoyer à  $s$  Lib( $D, d', nv$ )

... Après réception de Lib  
par les sites  $\in Q_w(A)$  :

	A	B	C	D
$D$	$d'$	$d'$	$d'$	$d$
$n^o$	1	1	1	0

### D veut lire

$Q_R(D) = \{A, D\}$   
 $D$  s'adresse à  $A$  qui  
 possède la valeur  
 courante .../...

112 / 152

## Algo #2 : Exemples de chronogrammes d'exécution

### Exemple avec 6 sites

On choisit  $R = 3$  &  $W = 4$   
 nb : chaque site appartient  
 implicitement à ses quorums

### Quorums

Site	$Q_R$	$Q_W$
1	$\{2,3\}$	$\{2,3,4\}$
2	$\{1,3\}$	$\{1,3,4\}$
3	$\{1,2\}$	$\{1,2,4\}$
4	$\{5,6\}$	$\{3,5,6\}$
5	$\{4,6\}$	$\{3,4,6\}$
6	$\{4,5\}$	$\{3,4,5\}$

### Chrono 1

2 lectures en //  
 1 Lecture + 1 Ecriture  
 demandées en //  
 Lecture **puis** Ecriture

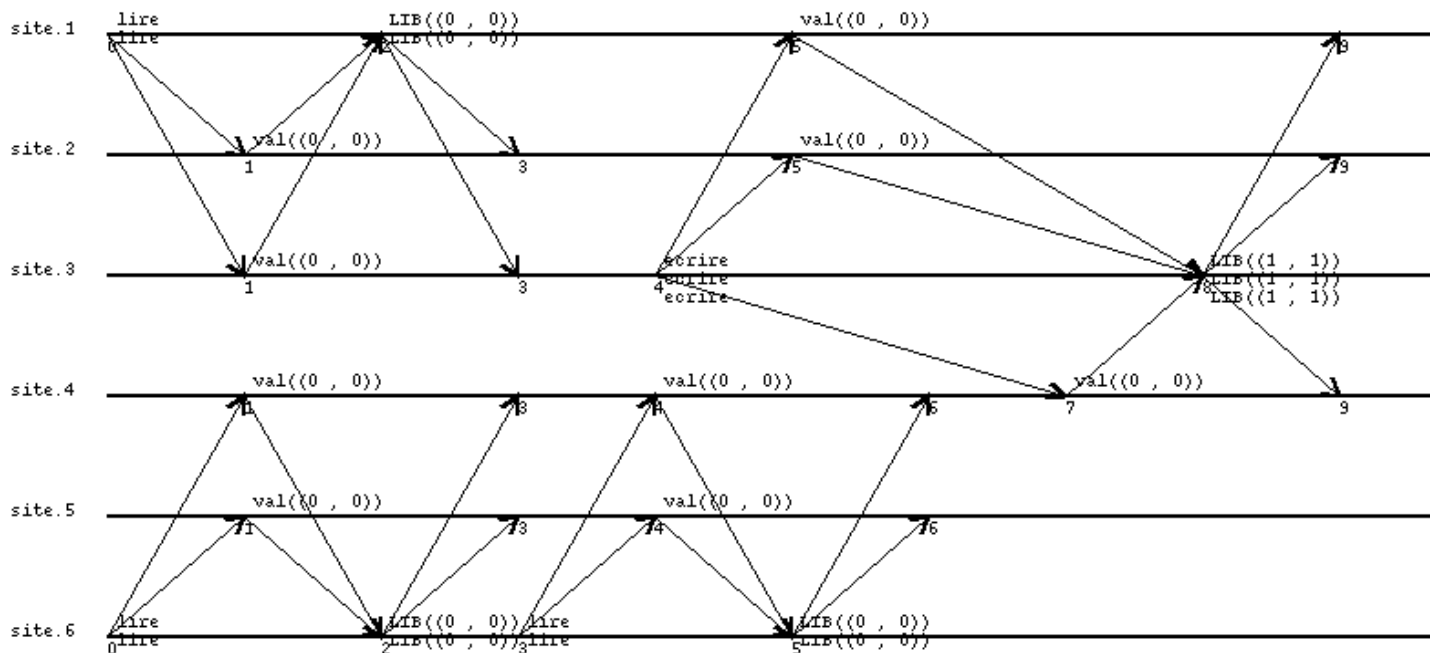
### Chrono 2

2 lectures en //  
 1 Lecture + 1 Ecriture  
 demandées en //  
 Ecriture **puis** Lecture

113 / 152

## Algo #2 : Exécution Chrono #1

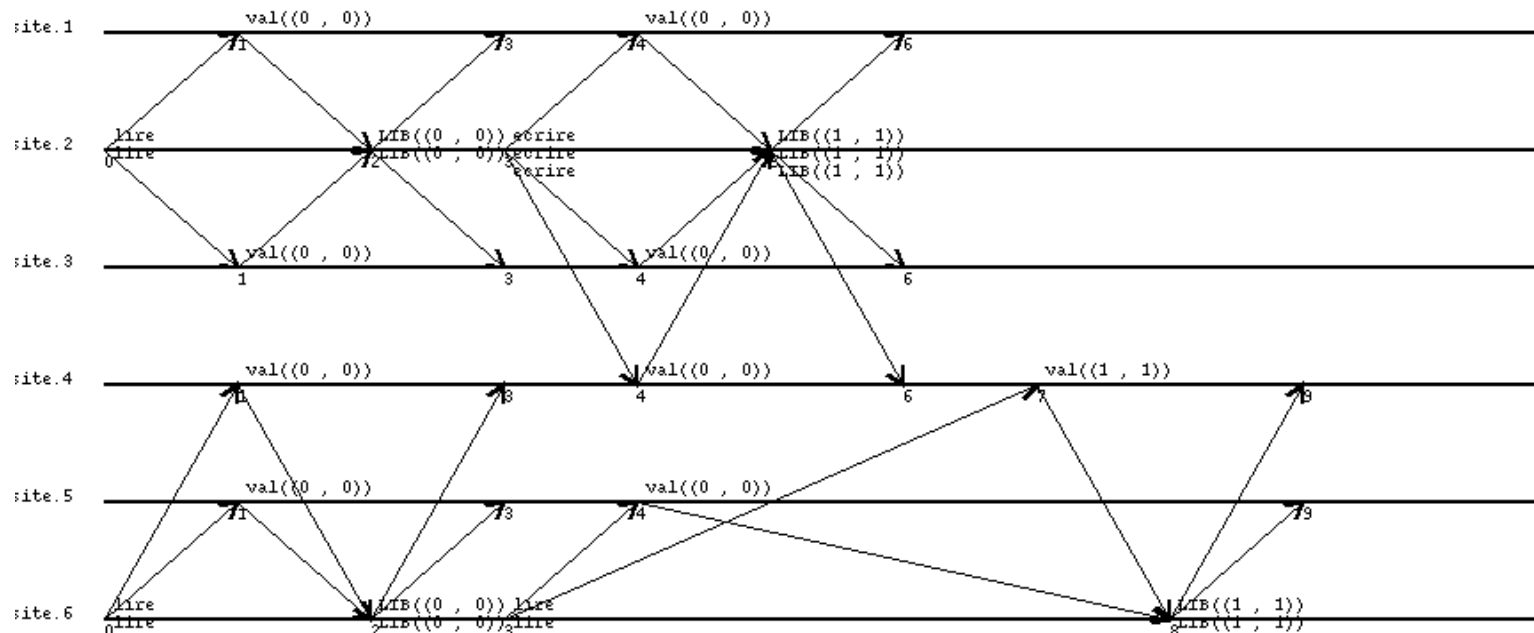
(2 lectures en //) (demandes // Lecture et Ecriture) (Lecture **puis** Ecriture)



114 / 152

## Algo #2 : Exécution Chrono #1

(2 lectures en //) (demandes // Lecture et Ecriture) (Ecriture **puis** Lecture)



115 / 152

## Algo #2 : Schéma de preuve

### I Accès Simultanés : $R_1$ & $R_2$

$$\left. \begin{array}{l} C_1 : R + W > N \\ C_2 : 2 \times W > N \end{array} \right\} \Rightarrow R_1 \wedge R_2$$

### II Cohérence Mutuelle des copies : $R_3$

**lemme** : Si  $R \subset S : |R| = r$  et  $W \subset S : |W| = w$

Alors  $R + W > n \Rightarrow R \cap W \neq \emptyset$

**preuve** :

- Soit  $W'$  le dernier quorum d'écriture  
Tout site  $s \in W'$  possède la version courante et le n° correct
  - Pour lire, un site s'adresse **nécessairement** à un site ayant appartenu au dernier quorum d'écriture.  
→ Lecture de la valeur courante
- Idem pour l'écriture

116 / 152

## Algo #3 : Généralisation Algo #2 via des Quorums

### Vers les quorums

Soient  $Q_W : S \mapsto 2^S$        $Q_R : S \mapsto 2^S$   
 $s \mapsto Q_W(s)$        $s \mapsto Q_R(s)$

vérifiant  $\forall s, t \in S \left\{ \begin{array}{l} Q_R(s) \cap Q_W(t) \neq \emptyset \\ Q_W(s) \cap Q_W(t) \neq \emptyset \end{array} \right.$

### Algo #3

On remplace dans algo #2 :

- $R$  sites par  $Q_R(s)$
- $W$  sites par  $Q_W(s)$

preuve : identique à celle d'algo #2

### Aperçu des Quorums

- $Q_R$  et  $Q_W$  sont des **quorums sur  $S$**   
Quorum :  $Q \subset 2^S$  tq  $\forall e, f \in Q : (e \not\subset f) \text{ et } (f \not\subset e)$
- $Q_W$  est une **côterie**  $\forall e, f \in Q_W : e \cap f \neq \emptyset$
- $Q_R$  et  $Q_W$  sont **complémentaires** :  $\forall e \in Q_W, \forall f \in Q_R : e \cap f = \emptyset$

117 / 152

# Quorums à Grille ( $\sqrt{n}$ ) – Maekawa 85

Exemple de quorums complémentaires dont l'un est une coterie

$$Q_R(A) = Q_R(B) = \{A, B\}$$

$$Q_R(C) = Q_R(D) = \{C, D\}$$

$$Q_R(E) = \{E\}$$

$$Q_W(A) = \{A, B, C, E\}$$

$$Q_W(B) = \{A, B, D, E\}$$

$$Q_W(E) = \{A, C, E\}$$

$$Q_W(C) = \{A, C, D, E\}$$

$$Q_W(D) = \{B, C, D, E\}$$

Propriété des grilles

Définition de  $Q_W$  &  $Q_R$

$$\text{Site } s \mapsto \begin{cases} Q_R(s) = \text{ligne}(s) \\ Q_W(s) = \text{ligne}(s) \cup \text{colonne}(s) \end{cases}$$

propriété :  $\forall s, t : \text{ligne}(s) \cap \text{colonne}(t) \neq \emptyset$

corollaires :  $\forall s, t :$

$$Q_R(s) \cap Q_W(t) \neq \emptyset$$

(complémentarité de  $Q_R$  et  $Q_W$ )

$$Q_W(s) \cap Q_W(t) \neq \emptyset \quad (Q_W \text{ est une coterie})$$

Grille

A	B
C	D
E	E

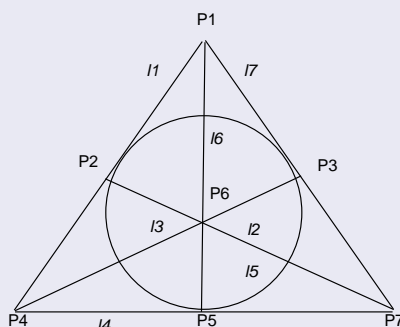
118 / 152

## Quorums (suite)

Retrouver les protocoles associés aux grilles représentées

A	B	C	D	E
A	B	C	D	E

Coterie engendrée par un plan projectif



Coterie associée

$$Q_1 \quad \{1, 2, 4\}$$

$$Q_2 \quad \{2, 6, 7\}$$

$$Q_3 \quad \{3, 6, 4\}$$

$$Q_4 \quad \{4, 5, 7\}$$

$$Q_5 \quad \{5, 2, 3\}$$

$$Q_6 \quad \{6, 1, 5\}$$

$$Q_7 \quad \{7, 1, 3\}$$

119 / 152



# Fragmentation & Duplication

## Cas général

1 Donnée  $D$  et  $n$  sites de stockages  $S_1, S_2, \dots, S_n$

- **Partition** de  $D = D_1 \oplus D_2 \dots \oplus D_c$  (  $c$  fragments de  $D$ )
- Chaque fragment est **dupliqué** sur certains sites

## Matrice de Distribution

- $c$  : nombre de fragments &  $n$  : nombre de sites
- $D[1 \dots c, 1 \dots n] : \{0, 1\}$
- $D(i, j) = 1$  ssi le fragment  $i$  est stocké sur le site  $j$

## Difficultés

Celles de la duplication

+ reconstitution de la donnée avec des fragments **cohérents**

NB : La matrice de distribution est une donnée globale "inconnue"

121 / 152

# Fragmentation & Duplication Symétrique

## Conditions symétriques

- Fragmentation :  $|D| = c \times |D_i| \quad \forall i \in \{1, \dots, c\}$
- Duplication :  $RD_1$  &  $RD_2$ 
  - $RD_1$  : Chaque fragment est stocké sur  $s$  sites  
(**distribution égale pour chaque fragment**)
  - $RD_2$  : Chaque site stocke le même nombre de fragments  
(**charge équilibrée entre les différents sites**)

## Conséquences de la symétrie

- $RD_1$  &  $RD_2 \Leftrightarrow MD_1$  &  $MD_2$
- - $MD_1 : \forall j : \sum_{i \in \{1, \dots, c\}} D(i, j) = m$   
**Tout site stocke  $m$  fragments**
  - - $MD_2 : \forall i : \sum_{j \in \{1, \dots, n\}} D(i, j) = s$   
**Tout fragment est stocké sur  $s$  sites**

122 / 152

# Exemples de Fragmentation/Duplication symétrique

## Rappel

$s$  = nombre de sites de stockage pour un fragment  
 $m$  = nombre total de fragments par site

### 1er Exemple : 3 sites & 3 fragments

	$S_1$	$S_2$	$S_3$
$D_1$	0	1	1
$D_2$	1	0	1
$D_3$	1	1	0

→  $s = m = 2$

### Sd Exemple : 6 sites & 4 fragments

	$S_1$	$S_2$	$S_3$	$S_4$	$S_5$	$S_6$
$D_1$	1	0	1	0	1	0
$D_2$	1	0	0	1	0	1
$D_3$	0	1	1	0	0	1
$D_4$	0	1	0	1	1	0

→  $s = 3$  &  $m = 2$

## Remarques

Si  $c = s = m = n$  Alors chaque site stocke tous les fragments (inutile)  
 Si  $c = n$  et  $s = m = 1$  Alors Fragmentation (pure)

123 / 152

## Fragmentation & Duplication Symétrique (suite)

### Contraintes sur $(n, c, m, s)$ liées à la Symétrie

→ Compter ( $NF$ ) le nombre total de fragments dans le système  
 $c$  fragments, chaque fragment est sur  $s$  sites →  $NF = c \times s$   
 $n$  sites et chaque site stocke  $m$  fragments →  $NF = n \times m$   
 → **Symétrie possible** ssi  $c \times s = n \times m$

### Retour sur les exemples précédents

1er :  $(2 \times 3 = 3 \times 2)$  & Sd :  $(4 \times 3 = 6 \times 2)$

### 3ème Exemple : 3 sites & 4 fragments

Trouver (les plus petits)  $m$  et  $s$   
 tels que  $4 \times s = 3 \times m$   
 →  $s = 3$  et  $m = 4$

**Duplication Pure**

	$S_1$	$S_2$	$S_3$
$D_1$	1	1	1
$D_2$	1	1	1
$D_3$	1	1	1
$D_4$	1	1	1

124 / 152

# Intérêt de la Symétrie : Facteur de Reconstitution

## Egalitaire

Chaque site participe de façon égale au stockage

Chaque fragment à le même nombre de "stockeurs"

## Facteur de Reconstitution (FR)

**Définition :** FR correspond au nombre **minimal** de sites qu'il faut interroger pour reconstituer dans le **pire des cas** la donnée initiale

Symétrie **permet** une expression **analytique** de  $FR$  :  $FR = (n - s) + 1$

## Preuve

①  $FR > n - s$  (par l'absurde)

Fragment  $f_1$ ,  $Stock(f_1)$  les  $s$  sites de stockage de  $f_1$

$U = Sites \setminus Stock(f_1)$

On a  $|U| = n - s$  et  $U$  ne permet pas d'obtenir  $f_1$

②  $FR = (n - s) + 1$

**lemme :** Si  $A \subset E$  et  $B \subset E$  Alors  $|A| + |B| > |E| \Rightarrow A \cap B \neq \emptyset$

Soit  $U : |U| = (n - s) + 1$  on a  $U \cap Stock(f) \neq \emptyset \quad \forall \text{ fragment } f$

125 / 152

# Protocole Associé : Aggrawal et El Abbadi (1990)

## Lire(x)

- 1) Obtenir les fragments et les numéros de versions de  $R$  sites
- 2) Soit  $vmax$  le plus grand numéro de version
- 3) Parmi les  $R$  sites du quorum, extraire les fragments  
issus de  $FR$  d'entre-eux **estampillés par  $vmax$**
- 4) Reconstituer  $x$  et 5) Libérer le quorum

## Ecrire(x)

- 1) Obtenir les fragments et les numéros de versions de  $W$  sites
- 2) Soit  $vmax$  le plus grand numéro de version
- 3) Parmi les  $W$  sites du quorum, extraire les fragments  
issus de  $FR$  d'entre-eux **estampillés par  $vmax$**
- 4) Reconstituer  $x$ , calculez  $Nx$  ( $NV = vmax + 1$ )
- 5) Fragmenter  $Nx$  et envoyer à tout site du quorum "son" fragment et le  $NV$

## Conditions sur $W$ et $R$ :

$C_1 : FR \leq R \leq n \ \& \ FR \leq W \leq n$

$C_2 : W + W > n$

$C_3 : n + FR \leq R + W \leq 2n$

126 / 152

## Exemple d'exécution (1/3)

### Configuration

6 sites, 4 fragments – Chaque fragment est stocké sur 3 sites ( $s=3$ )

$FR = (6 - 3) + 1 = 4$ , On prend  $R = 5$ ,  $W = 5$

	$S_1$	$S_2$	$S_3$	$S_4$	$S_5$	$S_6$
$D_1$	$d_1$ 0		$d_1$ 0		$d_1$ 0	
$D_2$	$d_2$ 0			$d_2$ 0		$d_2$ 0
$D_3$		$d_3$ 0	$d_3$ 0			$d_3$ 0
$D_4$		$d_4$ 0		$d_4$ 0	$d_4$ 0	

### Ecriture de $S_2$ : Interrogation de $Q_W(S_2) = S_1, \dots, S_5$

$S_2?$	$(S_1, (d_1, 0), (d_2, 0))$	$Vmax = 0$
	$(S_2, (d_3, 0), (d_4, 0))$	$D = d_1 \oplus d_2 \oplus d_3 \oplus d_4$
	$(S_3, (d_1, 0), (d_3, 0))$	$D' = F(D)$
	$(S_4, (d_2, 0), (d_4, 0))$	$D' = d'_1 \oplus d'_2 \oplus d'_3 \oplus d'_4$
	$(S_5, (d_1, 0), (d_4, 0))$	$NV = 1$

127 / 152

## Exemple d'exécution (2/3)

### Ecriture de $S_2$ (suite) : libération de $Q_W(S_2)$

$S_2!$	$S_1 : ((d'_1, 1), (d'_2, 1))$
	$S_2 : ((d'_3, 1), (d'_4, 1))$
	$S_3 : ((d'_1, 1), (d'_3, 1))$
	$S_4 : ((d'_2, 1), (d'_4, 1))$
	$S_5 : ((d'_1, 1), (d'_4, 1))$

### Mise à jour après libération du quorum

	$S_1$	$S_2$	$S_3$	$S_4$	$S_5$	$S_6$
$D_1$	$d'_1$ 1		$d'_1$ 1		$d'_1$ 1	
$D_2$	$d'_2$ 1			$d'_2$ 1		$d_2$ 0
$D_3$		$d'_3$ 1	$d'_3$ 1			$d_3$ 0
$D_4$		$d'_4$ 1		$d'_4$ 1	$d'_4$ 1	

128 / 152

## Exemple d'exécution (3/3)

Lecture par  $S_6$  ( $S_2 \dots S_6$ )

$$\begin{array}{l}
 S_6? \quad (S_2, (d'_3, 1), (d'_4, 1)) \\
 \quad (S_3, (d'_1, 1), (d'_3, 1)) \\
 \quad (S_4, (d'_2, 1), (d'_4, 1)) \\
 \quad (S_5, (d'_1, 1), (d'_4, 1)) \\
 \quad (S_6, (d_2, 0), (d_3, 0))
 \end{array}
 \mapsto
 \begin{array}{l}
 V_{max} = 1 \\
 D' = d'_1 \oplus d'_2 \oplus d'_3 \oplus d'_4 \\
 NV = 1
 \end{array}$$

Après lecture par  $S_6$

	$S_1$	$S_2$	$S_3$	$S_4$	$S_5$	$S_6$
$D_1$	$d'_1 \ 1$		$d'_1 \ 1$		$d'_1 \ 1$	
$D_2$	$d'_2 \ 1$			$d'_2 \ 1$		$d'_2 \ 1$
$D_3$		$d'_3 \ 1$	$d'_3 \ 1$			$d'_3 \ 1$
$D_4$		$d'_4 \ 1$		$d'_4 \ 1$	$d'_4 \ 1$	

129 / 152

## Schéma de preuve

Rappel des conditions sur  $W$  et  $R$  :

$$\begin{array}{l}
 C_1 : FR \leq R \leq n \ \& \ FR \leq W \leq n \\
 C_2 : W + W > n \\
 C_3 : n + FR \leq R + W \leq 2n
 \end{array}$$

preuve

- $C_1 \mapsto$  Tout quorum de lecture (d'écriture) est **suffisamment grand** pour récupérer **chaque fragment** de la donnée  
(!! pas nécessairement le plus récent)
- $C_2 \mapsto$  **Pas d'écritures simultanées**
- $C_{3-a} \mapsto W + R > n$  **Lecture et Ecriture sont exclusives**
- $C_{3-b} \mapsto W + R \geq n + FR \Rightarrow |W \cap R| \geq FR$   
Tout quorum de lecture **intersecte** tout quorum d'écriture sur au moins  $FR$  sites
- $C_1$  et  $C_{3-b} \mapsto$  **Tout quorum** (lecture ou écriture) permet de récupérer l'**exemplaire le plus récent** de **chaque** fragment de la donnée

130 / 152

# Exclusion Mutuelle Répartie : taxinomie des algorithmes (M. Raynal 1988)

## Rappel : Problèmes à résoudre (cf L. Lamport)

Safety : Au plus une CS en cours

Liveness : Tout agent en attente de CS y accèdera en un temps fini.

## Taxinomie des algorithmes d'Exclusion Mutuelle

- ① Algorithmes à base de Jetons (Le Lann 1977)
- ② Algorithmes à base de Permissions (Lamport 1978)

## Remarques sur l'exclusion mutuelle

- Problématique déjà rencontrée dans les algos de gestion des données dupliquées et illustration des horloges de Lamport.  
Algos déjà vus sont à base de permission  
≠ variantes en vue de minimiser le nombre de messages
- Un des pbs les plus étudiés de l'algorithmique répartie.  
Pas forcément le moins intéressant.

132 / 152

# Algorithmes à base de jeton

## Principe : Droit d'accès en SC est matérialisé par un "Jeton"

- Unicité du Jeton  $\Leftrightarrow$  Exclusion Mutuelle (safety)
  - Accès Equitable au Jeton  $\Leftrightarrow$  Absence de Famine (liveness)
    - ① Mouvement Perpétuel (Jeton bouge spontanément ex : anneaux)
    - ② Mouvement à la demande (Jeton ne bouge que s'il est demandé)
- Focus sur Naimi-Trehel 87 Arbre Reconfigurable dynamiquement

## Algo hybride hors classe :

### Exclusion Mutuelle Centralisée

Jeton (token-asking)

Permission (1 site possède toutes les suffrages)

133 / 152

# Anneau à Jeton Circulant (Token Ring)

## Le pour et le contre

### Pour :

- Simplicité  
(simplicité "relative" cf tp Systèmes concurrents semestre suivant)
- Taille des messages (0)
- Nombre de Messages (0 ?)
- Panne d'un site "facilement" récupérable

### Contre :

- Communication Permanente  
(liée à la circulation du jeton)
- Tous les sites participent même ceux inactifs  
(intérêt de Naimi-Trehel vu un peu plus loin)
- Temps minimum entre deux C.S consécutives  
**indépendant** de l'activité  
**dépend** des positions respectives des sites

134 / 152

## Algorithmes à Base de Permissions : rappels

### Principe général (safety)

- Site : oisif, attente, section critique
  - Un site oisif envoie une requête (**signée**) "aux autres" et passe en attente
  - Un site oisif recevant une requête répond *ok* et reste oisif
  - **Verdict** : Si toutes les réponses sont *ok* — > entrée en SC  
Sinon **Conflit et Inter-blocage**

### Résolution des conflits (liveness)

nb : solution **simple** si tous les sites en conflit se **connaissent**

– Déterminer un ordre sur l'ensemble des sites en conflit

- Statique (basé sur l'identité des sites)
- Dynamique (basé sur l'exécution du système)  
cf compteurs de requêtes, Horloges de Lamport
- Aléatoire (cf tirage au sort)

– Résolution basée sur l'ordre obtenu : 1<sup>er</sup> dans l'ordre passe en CS ; lorsqu'il sort, il prévient le second dans l'ordre .... etc

⇒ Résolution **Inter-blocage + Famine**

135 / 152

# Algorithmes à Base de Permissions : Panorama

## Algorithmes

**Classiques** : Lamport 78, Ricart & Aggrawala 81

Evolutions : Votes (85), Quorums (85.....)

↓ Nombre de Messages

↑ Résolution Inter-blocage & Famine

(Maekawa 85) Complexité  $c \times \sqrt{n}$  messages où  $3 \leq c \leq 5$   
& Horloges non bornées

## Qualités

- Nombre de Messages :  $c \times \sqrt{n}$  à  $2 \times n$   
Taille des messages : bornée ou non bornée  
Simplicité : dépend de la résolution des conflits
- Temps minimum entre deux C.S consécutives :
  - dépend du nombre de messages et de la résolution les conflits
  - indépendant de la position dans la topologie

136 / 152

## Algorithme de Trehel-Naimi (1987) : Taxinomie : Algo à base de demande de Jeton

### Principe Général

- **Arborescence dynamique** des sites
- La racine possède le **privilège** (jeton)
- La position de racine est **temporaire**
- Chaque site connaît son prédécesseur dans l'arbre (*dernier*)
- Une demande d'un site  $S_i$  va transiter (via la relation *dernier*) jusqu'à la **racine** de l'arbre
- L'arbre **évolue** au fur et à mesure des demandes d'entrée en CS ;
- Les sites inactifs se retrouvent progressivement en position de feuille

### Intérêt de Naimi-Trehel

Résolution **Conjointe** : Exclusion Mutuelle + Famine

Complexité *moyenne* en  $\log(n)$

137 / 152



# Description de l'algorithme de Trehel-Naimi

## Contexte d'un site

### Etat d'un site :

Application : *oisif*, *attente*, *CS* (mutuellement exclusifs)

Gestion de l'Arbre : *privilege*, *dernier*(*\_autre*)

(*\_autre* = *nil* si le site est une racine)

**Communication** : Chaque site a deux files de communication : *req* et *ack*

Sur *req* réception des requêtes (requêtes "*signées*")

Sur *ack* réception de l'autorisation

## Conditions Initiales : Sites = ( $S_i$ ) $i \in [1..n]$

Tous les sites sont dans l'état *oisif*

$S_1$  est la racine initiale

$S_i \neq S_1 \rightarrow \text{dernier}(S_1)$  est vrai, privilège est faux

Pour  $S_1$ , privilège est vrai,  $\text{dernier}(\text{nil})$  est vrai

$\mapsto$  **Configuration d'étoile centrée en  $S_1$**

138 / 152

# Description de l'algorithme de Trehel-Naimi : Comportement d'un site (1/2)

## Emission d'une Requête

Un site *oisif*

n'étant pas la racine ( $\text{dernier}(\text{@ref})$ )

envoie sa requête à son père ( $\text{@ref}$ )

passé en état d'attente et positionne  $\text{dernier}$  à *nil*.

– *Il se considère ainsi comme la prochaine racine !*

## Entrée en C.S

Un site en attente recevant un message ok entre en CS

## Sortie de C.S

Un site en CS quitte la CS retourne à l'état *oisif*

Il positionne  $\text{privilege}$  à vrai

139 / 152

## Description de l'algorithme de Trehel-Naimi : Comportement d'un site (2/2)

### Transit d'une requête via un noeud intermédiaire

Un site oisif ayant pour père @ref (dernier(@ref))  
recevant la requête du site  $i$ , transmet cette requête à son père (@ref)  
il reste oisif mais positionne dernier à  $i$   
– le site de transit "sait" que le site demandeur deviendra la racine ;  
il change de père au profit de la prochaine racine

### Réception d'une requête par la racine

La racine (dernier(nil)), oisive  
recevant la requête du site  $i$   
lui renvoie le message ok  
reste oisif et positionne dernier à  $i$  (change de père)

### Entrée en C.S de la racine

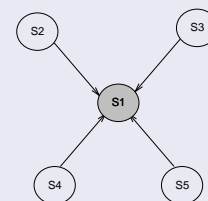
La racine (dernier(nil)) oisive  
"n'étant pas soumise à une requête"  
entre en CS

140 / 152

## Description de l'algorithme de Trehel-Naimi : Exemples d'exécution (1/3)

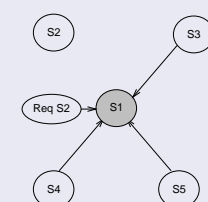
### Etat initial

	$S_1$	$S_2$	$S_3$	$S_4$	$S_5$
<b>Dernier</b>	nil	$S_1$	$S_1$	$S_1$	$S_1$
<b>Privilege</b>	V	F	F	F	F
<b>Etat</b>	oisif	oisif	oisif	oisif	oisif
<b>Req</b>	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
<b>Ack</b>	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$



### $S_2$ a demandé à entrer en CS

	$S_1$	$S_2$
<b>Dernier</b>	nil	nil
<b>Privilege</b>	V	F
<b>Etat</b>	Oisif	Attente
<b>Req</b>	[2]	$\emptyset$
<b>Ack</b>	$\emptyset$	$\emptyset$

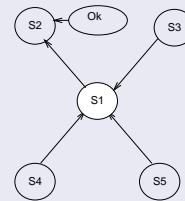


141 / 152

## Description de l'algorithme de Trehel-Naimi : Exemples d'exécution (2/3)

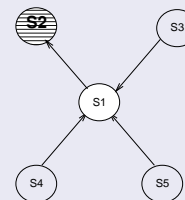
### Réception par $S_1$ & Envoi d'un ack

	$S_1$	$S_2$
Dernier	$S_2$	nil
Privilege	F	F
Etat	Oisif	Attente
Req	$\emptyset$	$\emptyset$
Ack	$\emptyset$	[ok]



### Réception du ack et passage en CS

	$S_1$	$S_2$
Dernier	$S_2$	nil
Privilege	F	F
Etat	Oisif	CS
Req	$\emptyset$	$\emptyset$
Ack	$\emptyset$	$\emptyset$

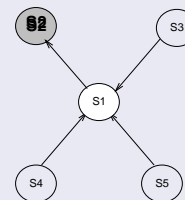


142 / 152

## Description de l'algorithme de Trehel-Naimi : Exemples d'exécution (3/3)

### Retour à Oisif

	$S_1$	$S_2$
Dernier	$S_2$	nil
Privilege	F	V
Etat	Oisif	Oisif
Req	$\emptyset$	$\emptyset$
Ack	$\emptyset$	$\emptyset$



### Quid de la complexité ?

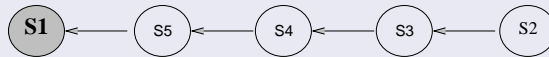
**$\log(n)$  en moyenne /  $n$  dans le pire des cas**  
voir exemple illustratif à la suite

**Variante** Arbre "Equilibré" Helary-Raynal (94)  $\log(n)$   
"AVL" distribué : la topologie est nettement moins dynamique !

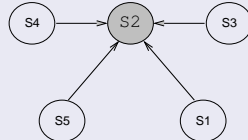
143 / 152

## Modification de la topologie : Illustration du pire cas (1/2)

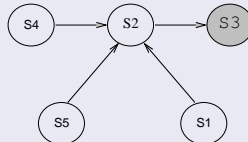
On choisit initialement le pire cas



Requête de  $S_2$  et Accès en CS



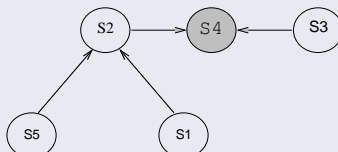
Requête de  $S_3$  et Accès en CS



144 / 152

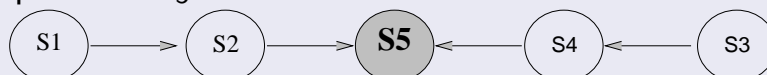
## Modification de la topologie : Illustration du pire cas (2/2)

Requête de  $S_4$  et Accès en CS



Retour au pire cas ...

Après une requête de  $S_5$



Après une requête de  $S_1$



Après une requête de  $S_2$



145 / 152

# Exclusion Mutuelle : bilan comparatif

## Complexité

$n$	Ricart & Agrawala $2 \times (n - 1)$	Maekawa $3\sqrt{n}$	Trehel $\text{Log}(n)$
100	198	30	5,2
1000	1998	93	7,5
10000	19998	300	9.8

## Vivacité (absence de Famine)

	Ricart & Agrawala	Maekawa	Trehel
	Compteurs (bornés)	Horloges (Lamport) Non-Bornées	Intégré

146 / 152

## Rapide bilan

### Quelques Outils Génériques

- Temps causal,
- Phases,
- Vagues,
- Consensus, Quorums, [Systèmes de votes](#), ...

### Quelques problèmes classiques

- Election (Tirage au sort),
- Exclusion mutuelle,
- Gestion des Données Distribuées,
- Calcul d'arbre couvrant
- Calcul de tables de routages
- Détection de la Terminaison
- [Diffusion causale](#)
- [Etat Global Réparti](#), ...

148 / 152

*Computer Networks* A.S TANNENBAUM – Prentice Hall

*Elements of Distributed Algorithms :  
Modeling and analysis with Petri Nets*  
W. REISIG – Springer

*Concurrent Programming : Algorithms, Principles and Foundations*  
M. RAYNAL – Springer

*Distributed Algorithms for Message-Passing Systems*  
M. RAYNAL – Springer

*Communication et le temps dans les réseaux et les systèmes répartis*  
*Synchronisation et état global dans les systèmes répartis*  
*Gestion des données réparties : problèmes et protocoles*  
M. RAYNAL – Collection E.D.F, Eyrolles - 1992

149 / 152

## Table des Matières

❶	Introduction .....	2
❶	Terminologie, Exemples	
❷	Le Contrôle : centralisé Vs réparti	
❸	Propriétés attendues dans un Système Réparti	
❹	Qualité d'un algorithme réparti	
❺	Panorama du cours	13
❷	Temps Causal (L. Lamport) .....	13
❶	Motivation	
❷	Causalité	
❸	Horloges de Lamport	
	Application à la résolution distribuée de conflits	
❹	Hologes vectorielles (Fidge & Mattern (88/89)	
	Application de F&M	
❸	Synchronisation par Phases .....	33
❶	Algorithmes à phases	
❷	Calcul phasé de tables de routages optimaux	
❸	Algorithme de Calcul des Tables de routages	
❹	Schéma Général	

150 / 152

# Table des Matières

④	Synchronisation par Vagues.....	49
①	Construction répartie d'arbres couvrants	
②	Algorithme de construction d'arbre couvrant	
③	Exemples d'Exécution	
④	Schémas généraux d'algorithmes à vagues	
⑤	Exercice : Calcul des tables de routages optimaux dans une arborescence couvrante	
⑤	Terminaison Répartie.....	75
①	Description du problème	
②	Principe de la solution par "Vagues"	
③	Schéma d'algorithme	
④	Exécutions	
⑥	Tirage au Sort Réparti .....	87
①	Introduction	
②	Algo #1 : Tirage au sort d'un vainqueur entre $n$ compétiteurs	
③	Algo #2 : Tirage au sort parmi $c$ entre $n$ sites	
④	Algo #3 : Obtention d'un ordre total entre les compétiteurs	

151 / 152

# Table des Matières

⑦	Gestion des Données Dupliquées.....	104
①	Paradigme des lecteurs/écrivains	
②	Protocole de Base : "Write all, Read one"	
③	Algorithmes par Vote (consensus)	
④	Généralisation par des Quorums	
⑧	Fragmentation & Duplication.....	119
①	Fragmentation & Duplication Symétrique	
②	Protocole Associé : Aggrawal et El Abbaddi (1990)	
⑨	Exclusion Mutuelle Répartie : taxinomie des algorithmes .....	130
①	Algorithme à base de Jeton	
②	Anneau à Jeton Circulant (Token Ring)	
③	Algorithmes à Base de Permissions	
④	Algorithme de Trehel-Naimi (1987) : Jeton à la demande	
⑩	Epilogue .....	146
①	Conclusion	
②	Bibliographie	
③	Table des Matières	

152 / 152