

Consensus et détection de fautes

Pierre Sens

Pierre.Sens@lip6.fr

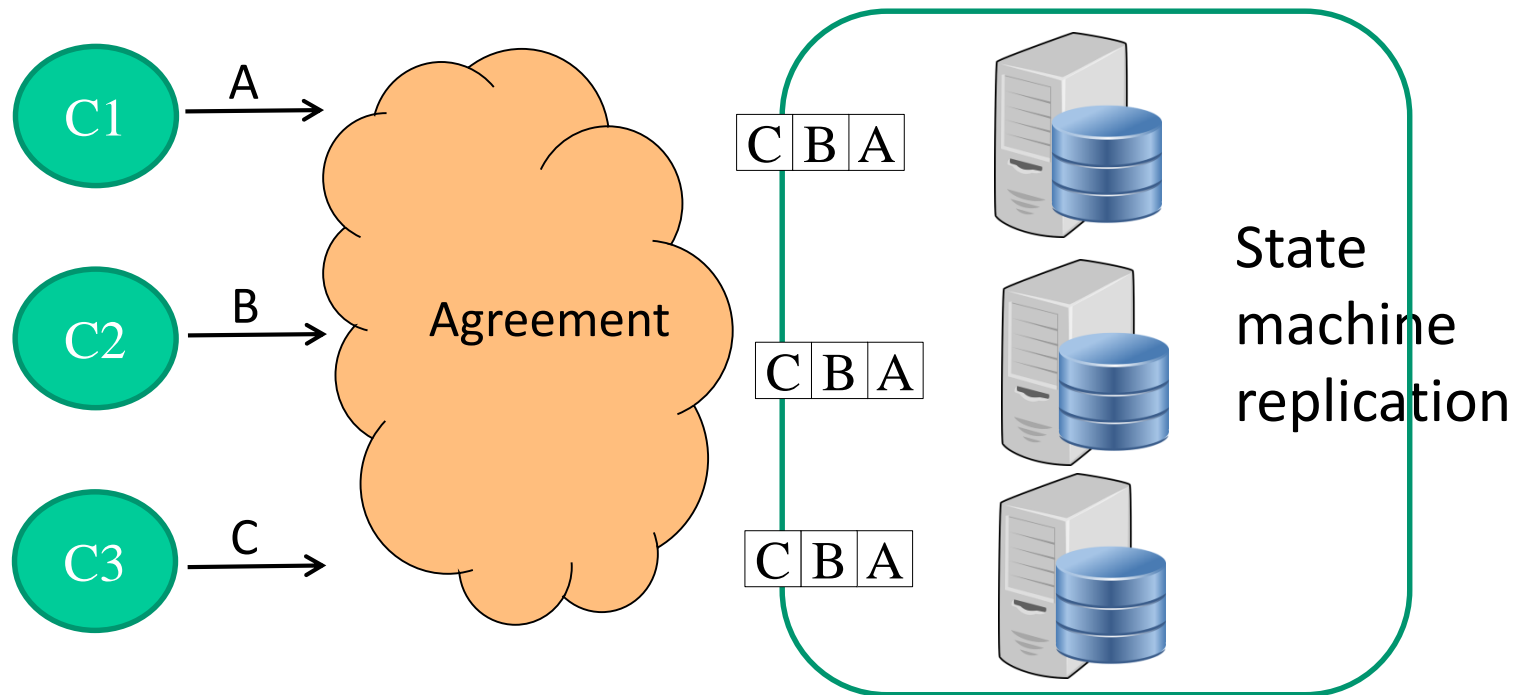
Algorithmes répartis tolérants les fautes

- Construction d'applications fiables
 - Problème complexe
 - Agencement de primitives fiables
- Définition de primitives
 - Consensus, diffusion atomique, gestion de groupe, ...

Le consensus = dénominateur commun

Problème d'accord

- Abstraction fondamentale pour construire des services fiables



Accord sur l'ordre des opérations

Spécification du consensus (1)

- Permettre l'accord entre processus

Initialement

1 valeur initiale par processus

Finalement

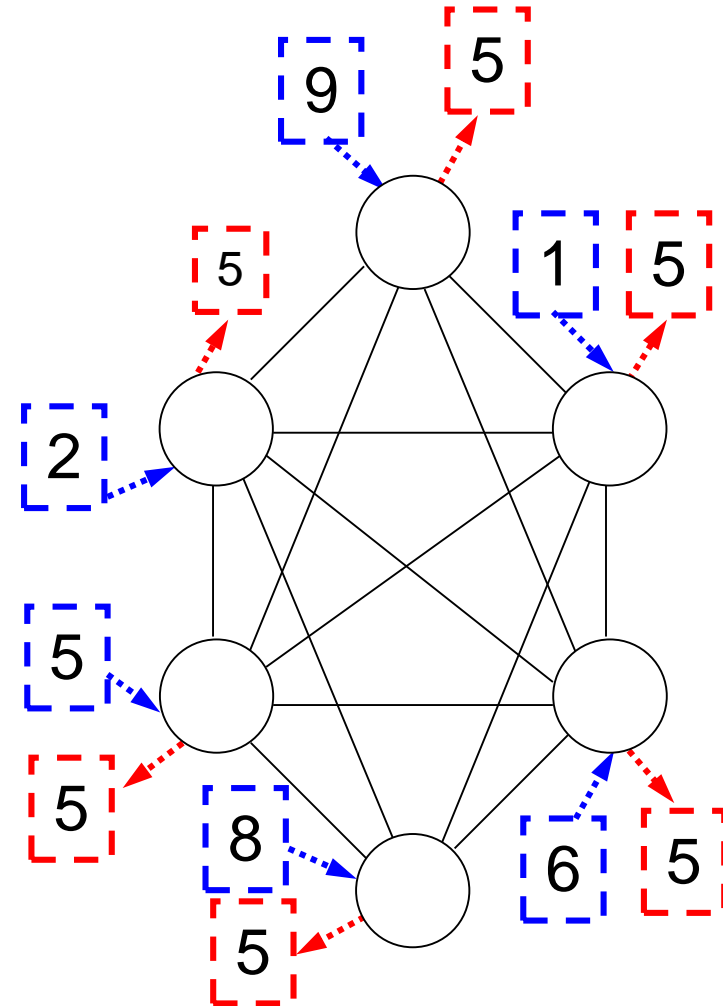
Tous les processus corrects
décident une même valeur

Validité: si un processus décide v alors v est une valeur proposée

Terminaison : tous les processus corrects décident finalement

Cohérence (agreement) : deux processus corrects ne peuvent décider différemment

[Intégrité : un processus doit décider au plus une fois]



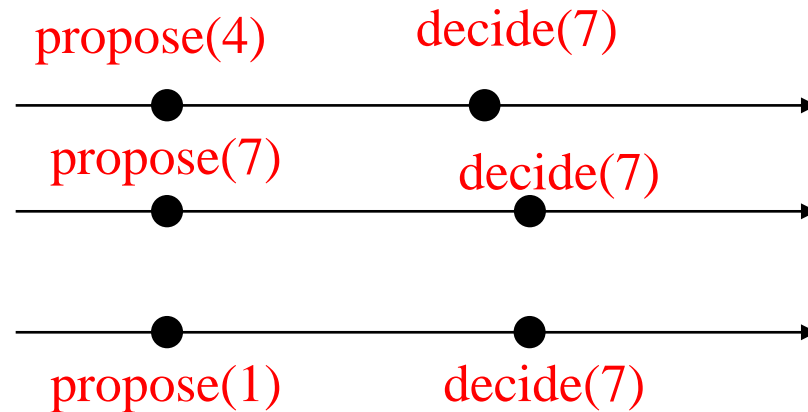
Spécification du consensus (2)

Consensus uniforme :

- **Validité**: si un processus décide v alors v est une valeur proposée
- **Terminaison** : tous les processus corrects décident finalement
- **Cohérence uniforme** (uniform agreement) : deux processus ~~corrects~~ ne peuvent décider différemment

Spécification du consensus (3)

- Deux primitives:
 - **propose(v)**: le processus appelant propose une valeur initiale v
 - **decide(v)**: le processus appelant décide v



Définition et modèle (1) : Processus

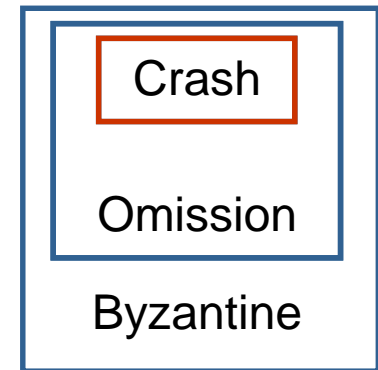
2 types de processus:

- **correct** : ne défaille pas pendant toute la durée de l'exécution
- **fautif** : pas correct
- Interconnexion :
 - $\Pi = \{p_1, p_2, \dots, p_N\}$ – N processus communiquent par passage de messages
 - Graphe complet

Définition et modèle (2) : Types de fautes

Processus :

- **Franche** (*crash*) : le processus fautif n'émet plus ni ne reçoit de message de façon *permanente* = silence sur défaillance - *fail-silent*, variante *fail-stop* (faute visible)
- **Omission** : Transitoire
- **Temporaire** : Trop tôt ou trop tard
- **Byzantin** : malveillance



Canaux :

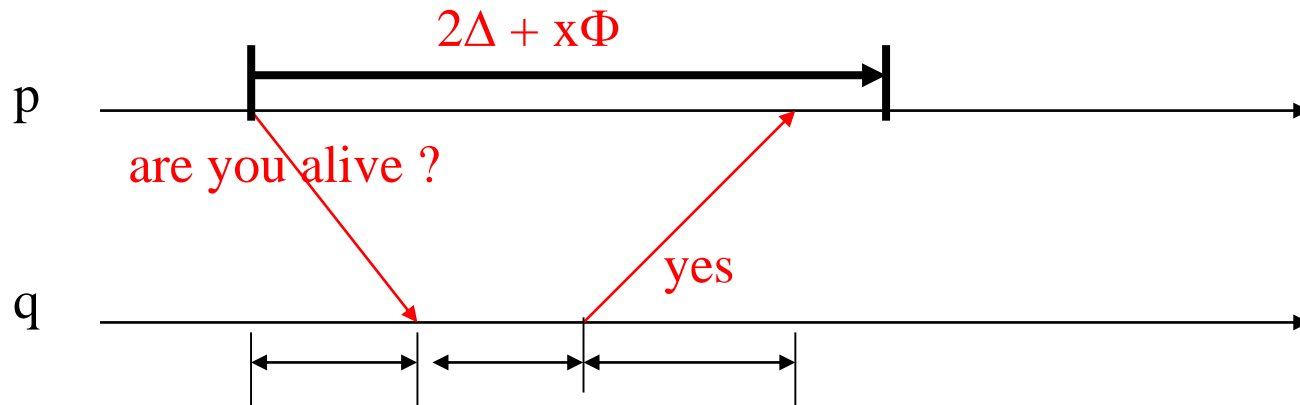
- **Fiable** (*reliable*): si p exécute `send(m)` vers q et q est correct, alors q recevra m
- **Quasi-fiable** (*quasi-reliable*): si p exécute `send(m)` vers q et p et q sont corrects, alors q recevra m
- **Equitable** (*fair-lossy*) : si un processus correct envoie un message m à q une infinité de fois, alors q recevra m

Définition et modèle (3) : Modèles temporels

- Hypothèse sur les vitesses de transmission et de traitement des messages
- Modèle synchrone :
 - **Borne Δ sur le temps de transmission** : Si un processus p envoie un message vers q à l'instant t , alors q reçoit le message avant $t+\Delta$.
 - **Borne Φ sur la vitesse relative des processus** : Si le processus le plus rapide prend x unités de temps pour un traitement, alors le processus le plus lent ne peut pas prendre plus $x\Phi$ temps pour faire le même traitement

Modèle temporel : système synchrone

Permet une détection parfaite



Modèle temporel : système asynchrone

- Pas de borne sur les délais de transmission
- Pas de borne sur les vitesses relatives des processus

=> Impossible de distinguer entre un processus lent et un processus « crashé »

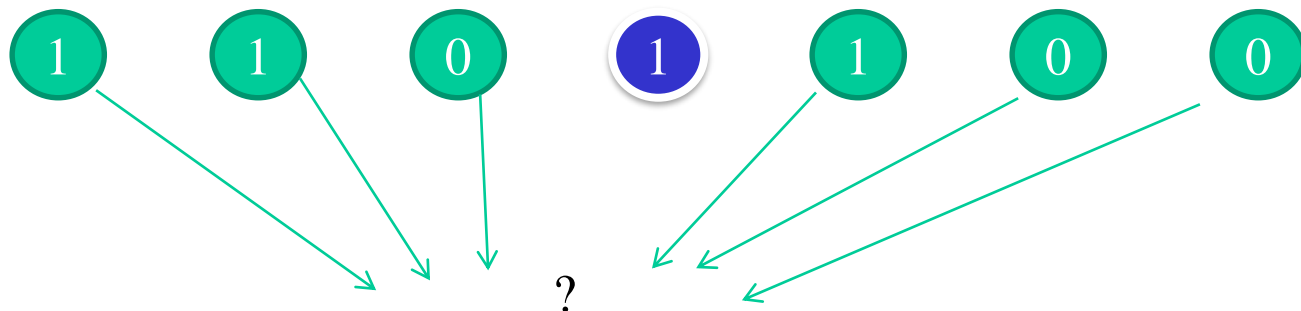
Modèles temporel : systèmes partiellement synchrones

(Dwork, Lynch, Stockmeier, 1988):

- Modèles intermédiaires entre synchrone et asynchrone (32 modèles)
- Bornes Δ et Φ du modèle synchrone :
 1. Existent mais sont **inconnues**, ou
 2. Sont connues mais **ont lieu à partir d'un temps T** appelé GST : **global stabilization time**
- Avant GST, le système est instable (pas de bornes)
- Après GST, le système est stable (bornes)
- GST est inconnu

Impossibilité de Fischer, Lynch et Paterson [FLP 85]

- Impossible de résoudre le consensus de façon déterministe
 - Asynchrone
 - Réseau fiable
 - 1 seul crash
- Idée :
 - Impossible de différencier un processus défaillant d'un processus lent
 - La décision peut dépendre d'un seul vote



Contourner FLP 85

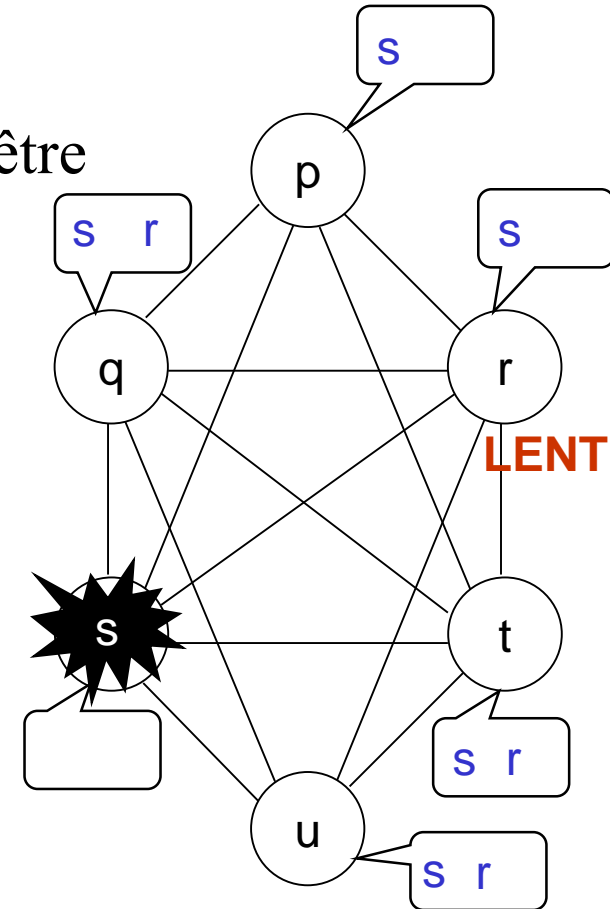
- **Changer le problème**
 - k -agreement [Cha90]
 - Plusieurs valeurs peuvent être décidées
- **Systèmes partiellement synchrones** [DDS87]
 - Les bornes sont non connues, valables uniquement à partir d'un moment
 - Variantes : alternance de bonne et mauvaise périodes
 - Borne restreinte à certains nœuds :
 - 1 bi-source (ultime) : Il existe (ultimement) une borne sur les liens entrants et sortants de la source
 - 1 source (ultime) : Il existe (ultimement) une borne sur les liens sortants
 - Algorithmique dépendante du système
- **Consensus « imparfait »**
 - Consensus probabiliste [BO83] : Des processus peuvent ne pas terminer
 - Paxos [Lamport 89] : Hypothèse très faible, terminaison non assurée
- **Les détecteurs de défaillances non fiables** [CHT96]
 - Algorithmique en asynchrone (indépendante du système)
 - Hypothèses plus facilement utilisables

Détecteur de défaillances non fiables [CHT 96]

- Introduit en 1991
- Oracle local sur chaque nœud
- Fournit une liste des processus suspectés d'être défectueux
- Informations non fiables
 - Possibilité de fausses suspicions

Complétude : un processus défectueux doit être détecté comme défectueux

Justesse : un processus correct ne doit pas être considéré comme défectueux



Qualités des détecteurs

- Complétude (completeness)
 - **forte** : Il existe un instant à partir duquel tout processus défaillant est suspecté par *tous* les processus corrects
 - **Faible** : Il existe un instant à partir duquel tout processus défaillant est suspecté par *un* processus corrects
- Justesse (accuracy) :
 - **Forte** : aucun processus correct n'est suspecté
 - **Faible** : il existe au moins un processus correct qui n'est jamais suspecté
 - **Finalement forte** : *il existe un instant à partir duquel* tout processus correct n'est plus suspecté par aucun processus correct
 - **Finalement faible** : *il existe un instant à partir duquel* au moins un processus correct n'est suspecté par aucun processus correct

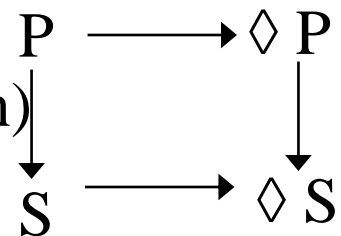
Classes de détecteurs

Hypothèses : pannes franches, communication fiable, réseau asynchrone

	Justesse			
	Forte	Faible	Finalement forte	Finalement faible
Complétude forte	P	S	$\diamond P$	$\diamond S$
Complétude faible	Q	W	$\diamond Q$	$\diamond W$

- Complétudes forte et faible sont équivalentes
(on peut construire une complétude forte à partir d'une faible)
 \Rightarrow 4 classes : P, S, $\diamond P$, $\diamond S$

- Force des détecteurs (\longrightarrow = implication)



Les complétudes forte et faible sont équivalentes

1. La complétude faible est incluse dans la forte
2. Construction de la complétude forte à partir de la faible

|| Task 1: repeat forever

{p queries its local failure detector module \mathcal{D}_p }

$suspects_p \leftarrow \mathcal{D}_p$

send $(p, suspects_p)$ to all

|| Task 2: when receive $(q, suspects_q)$ for some q

$output_p \leftarrow (output_p \cup suspects_q) - \{q\}$

→ Uniquement 4 classes de FD : P, S, $\diamond P$, $\diamond S$

Détecteur et consensus

- Consensus résoluble avec $\diamond S$
- $\diamond S$ le plus faible détecteur pour résoudre le consensus (minimalité) avec **une majorité de processus corrects**
- FLP \Rightarrow Impossible à implémenter en asynchrone

Leader ultime

- Détecteur de défaillances Ω

Ω : un détecteur de défaillances dont la sortie est un unique processus supposé être correct

q est la sortie de Ω à l'instant t :

p fait confiance à q à l'instant t

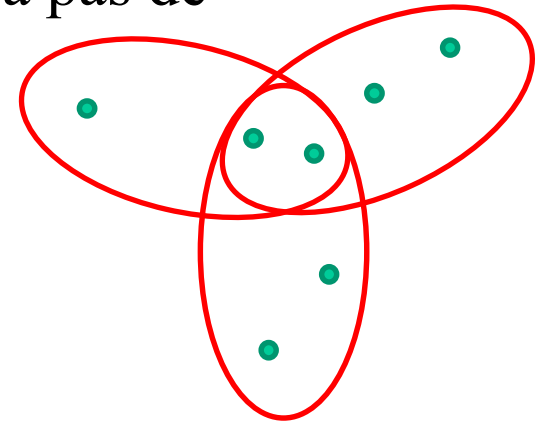
Ω assure :

un jour tous les processus corrects feront confiance au *même* processus *correct*.

Ω et $\langle \rangle S$ équivalent

Détecteur minimum avec $n-1$ fautes

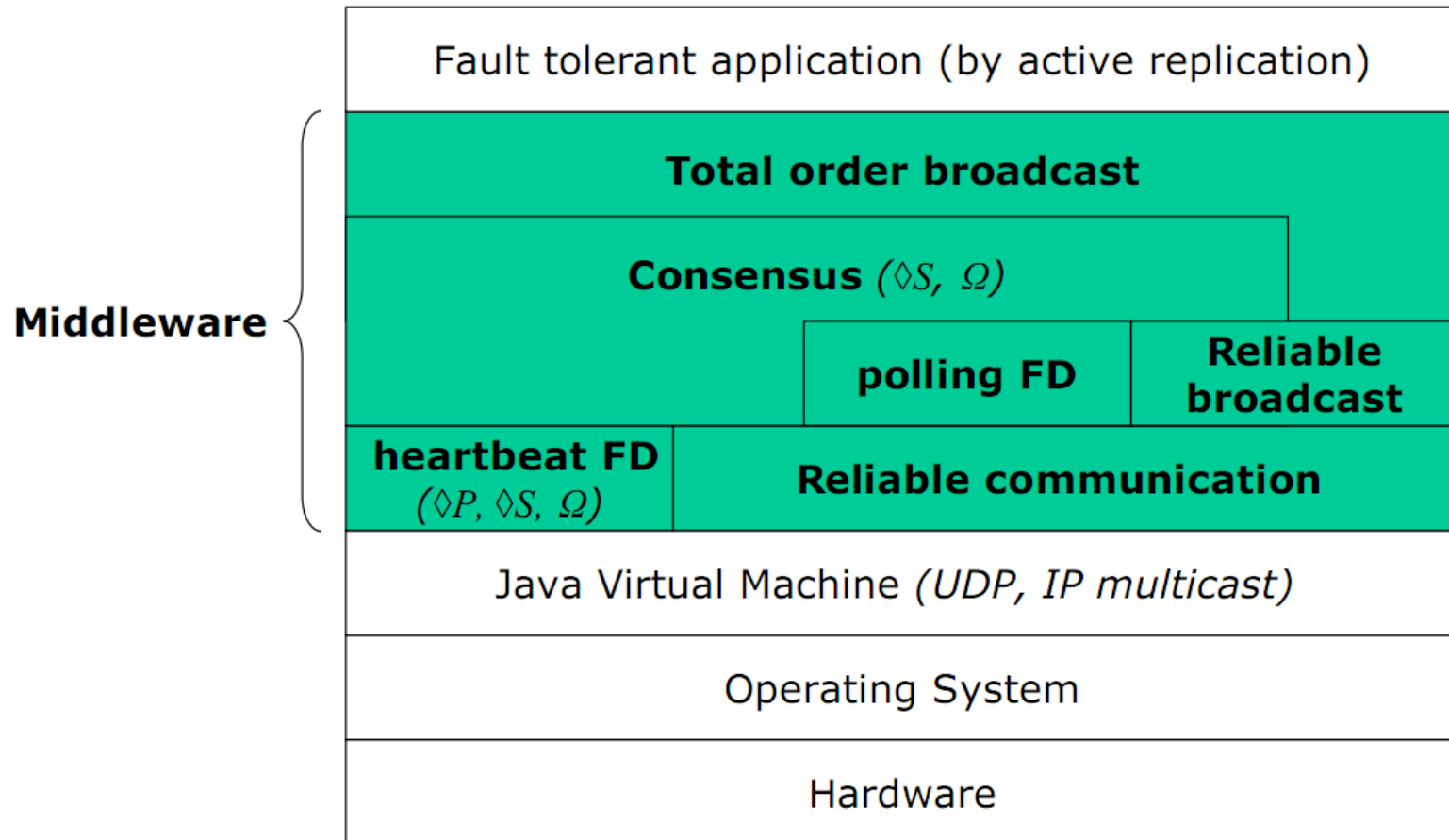
- Détecteur de fautes Σ
 - Propose une liste de processus corrects
 - *Intersection* : Il y a toujours une intersection non vide dans les listes proposées par Σ
 - *Complétude ultime* : ultimement, il n'y a pas de processus fautif dans les listes
- (Ω, Σ) le plus faible détecteur pour réaliser le consensus avec $n-1$ fautes



FD minimum

Problems Models	Consensus	k-set agreement	set agreement	Eventual consistency
Shared memory	Ω [LH94]	k-anti- Ω [GK09]	anti- Ω [Z10]	
Message passing	(Ω, Σ) [DFG10]	?	\mathcal{L} [DFGT08]	Ω [DKGPS15]

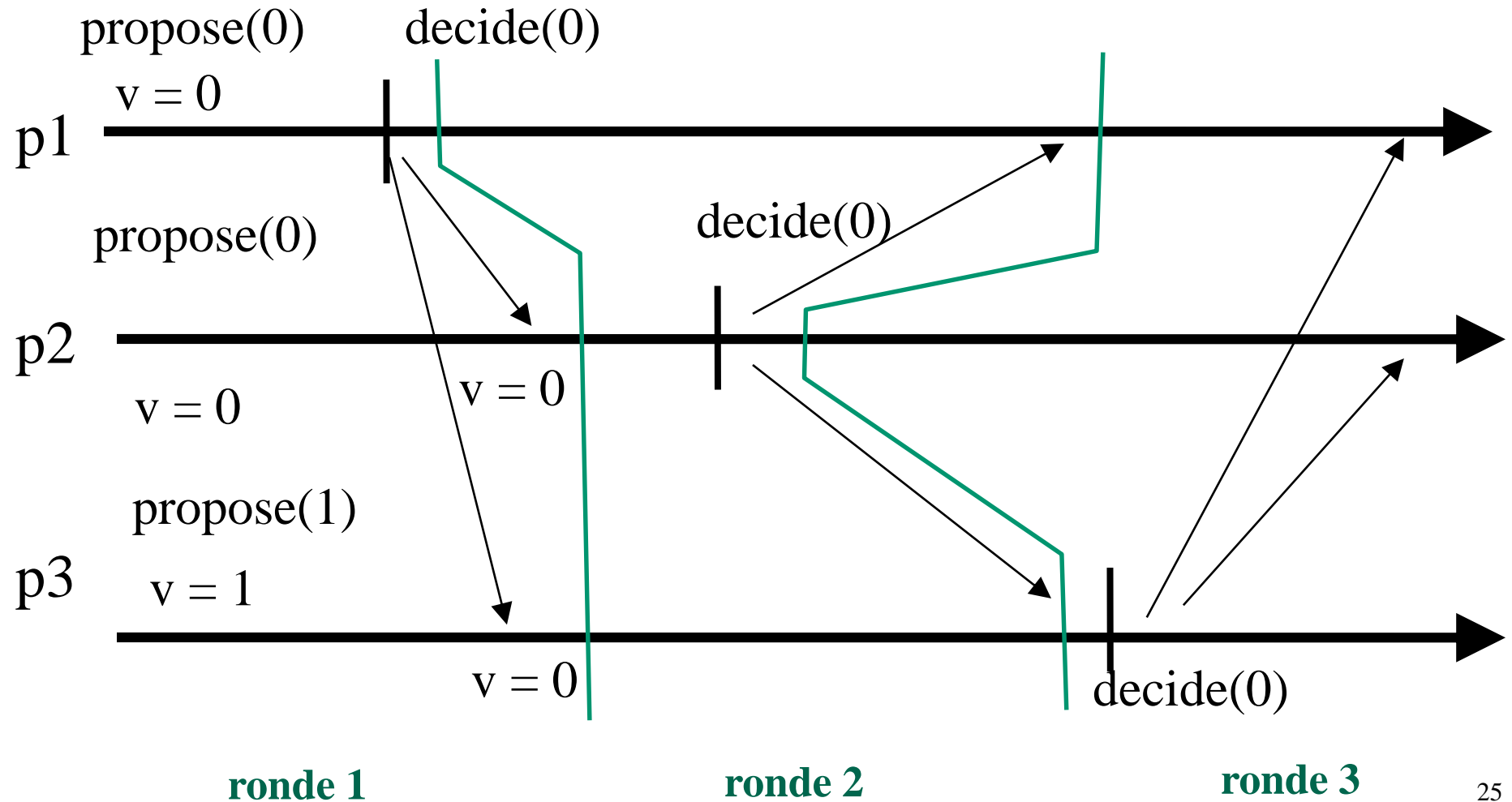
Fault-tolerant Architecture



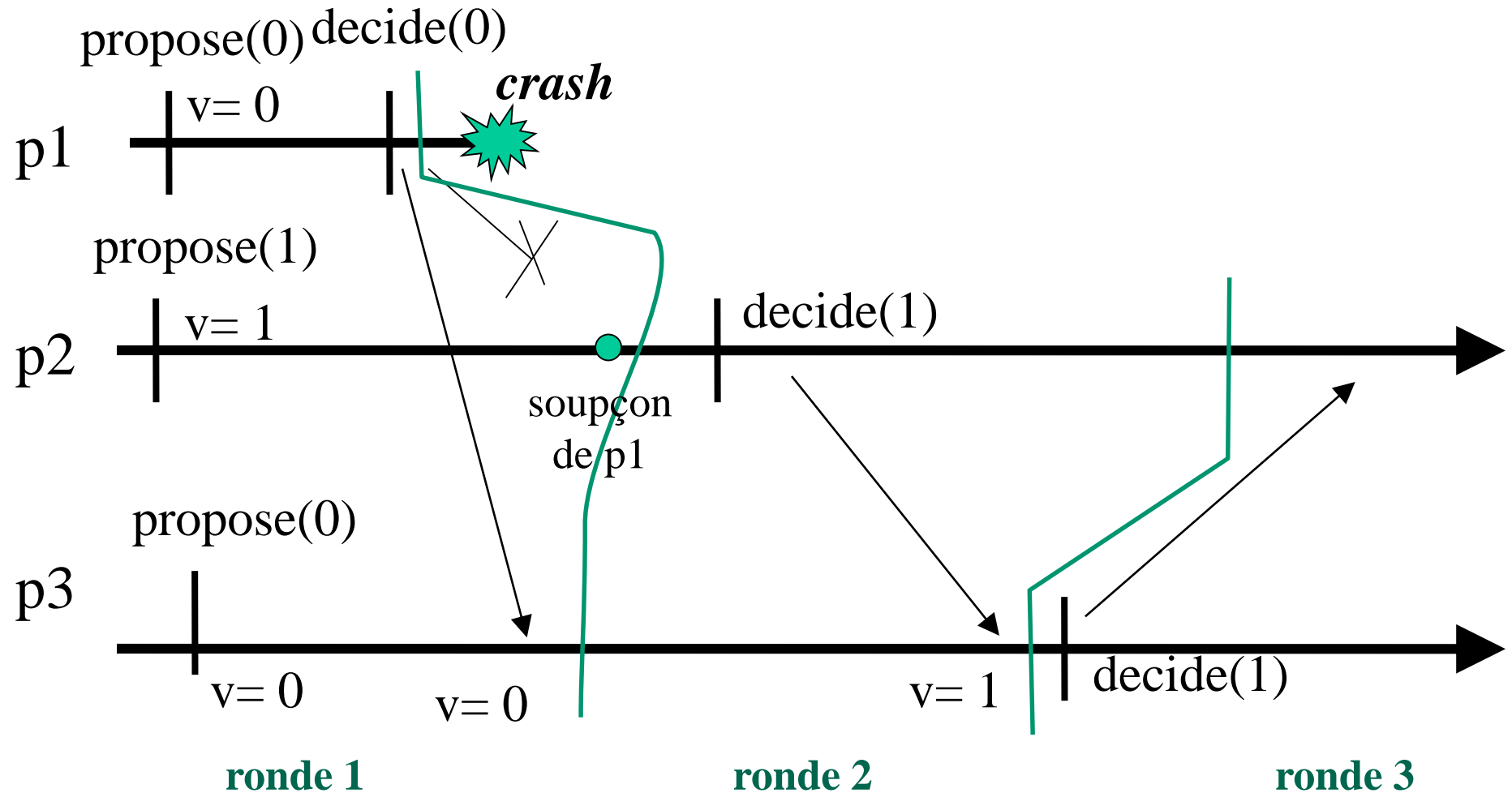
Consensus avec P – Algorithme 1

- Fondé sur des rondes et un leader
- Les processus exécutent des rondes de manière incrémentale (1 à n)
- Dans chaque ronde : le processus dont l'id correspond au numéro de ronde est le leader ($\text{id leader} = \text{id ronde} \% N$)
- Le leader choisit sa valeur courante, la décide et la diffuse à tous.
- Les “non” leader ($\text{id node} \neq \text{id ronde} \% N$) attendent :
 - (a) la réception du message du leader pour choisir sa valeur
 - (b) la suspicion du leader
- En n rondes tous les processus ont décidé (tous ont été leaders)

Algorithme 1 : Exemple



Algorithme 1 : Exemple (2)



Autres consensus sur P

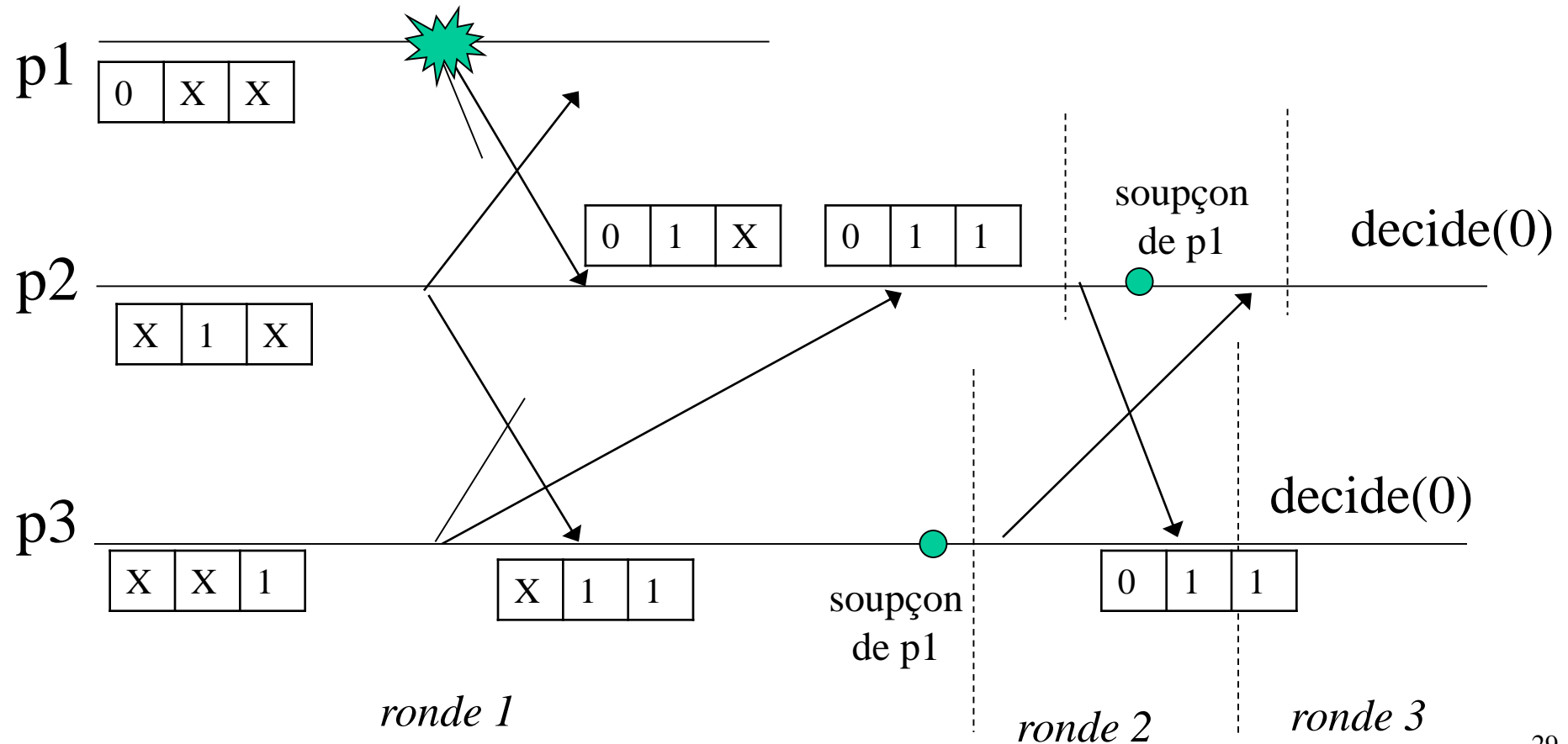
- Attendre les N rondes pour décider [CT 91, DLS 88]
- Attendre $f+1$ rondes pour décider : Algorithme 2

Consensus avec P : Algorithme 2

- f = nombre maximum de fautes tolérées
- Chaque processus P_i maintient un vecteur V_i pour stocker les valeurs proposées
- $f+1$ rondes :
 - Chaque processus P_i diffuse V_i de façon incrémentale
 - Attendre la réception des vecteurs de tous les processus non suspectés
- Après $f+1$ rondes (tours) :
 - P_i choisit et décide **la première valeur non vide** de son vecteur

Algorithme 2 : Exemple

$f = 1$



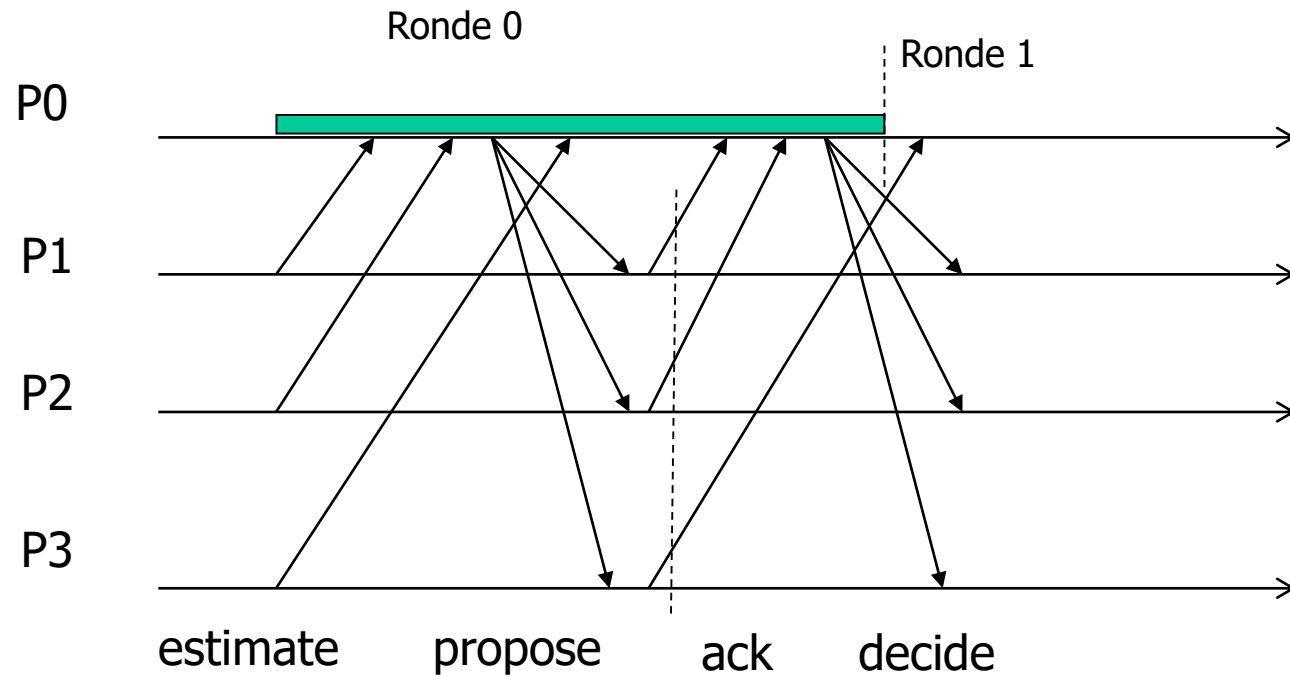
Consensus avec $\langle \rangle_S$

- Algorithme du coordinateur tournant [CHT 96]
- $f < n/2$ crashes
- Processus numérotés $1, 2, \dots, n$
- Exécution de *rondes asynchrones*
- Ronde r , coordinateur = processus $(r \bmod n) + 1$
- Le coordinateur c :
 - Impose sa valeur v
 - v est choisie si c n'est pas suspecté

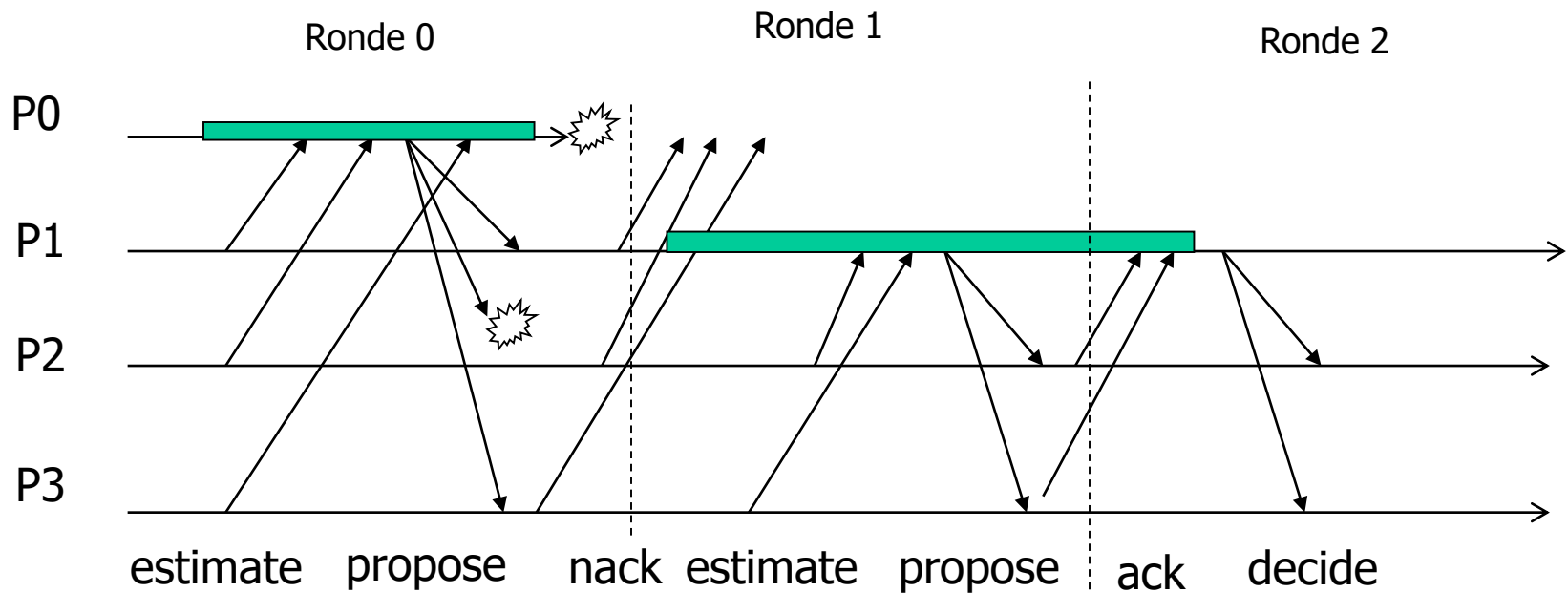
Consensus avec $\langle \rangle S$ (2)

- 4 phases par ronde
- **Phase 1** : chaque processus envoie au coordinateur sa valeur courante estampillée par la ronde de sa dernière mise à jour.
- **Phase 2** : le coordinateur réunit une majorité de valeurs, valeur estimée = une valeur parmi les plus à jour, diffusion de la valeur estimée
- **Phase 3** : Pour chaque processus correct :
 - Réception de la valeur estimée : renvoyer ack au coordinateur, mise à jour de la valeur courante
 - Soupçon du coordinateur : renvoyer nack
- **Phase 4** :
 - Coordinateur reçoit une majorité de réponses (ack-nack) :
si majorité de ack
valeur finale = valeur estimé + diffusion fiable valeur
A la réception tous les processus décident la valeur reçue
 - Si pas de majorité : changement de ronde

Coordinateur tournant : Exemple

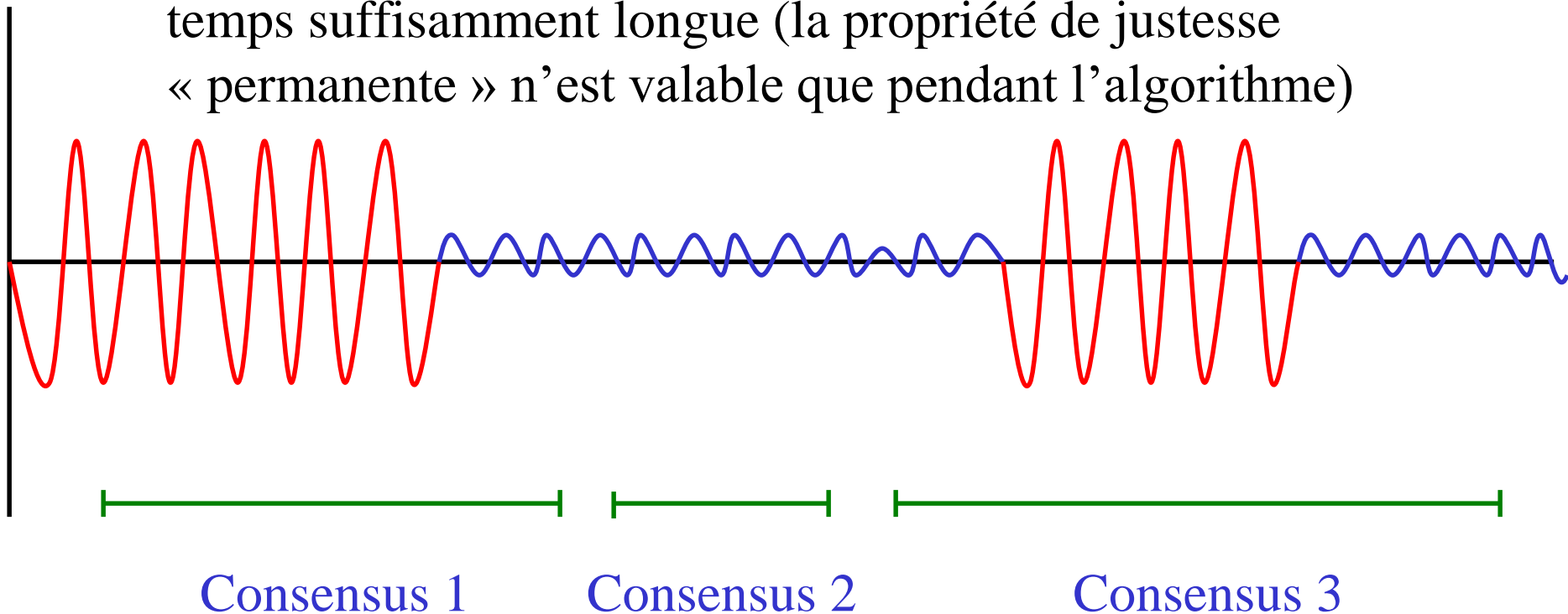


Coordinateur tournant : Exemple (2)



Comportement en cas d'instabilité

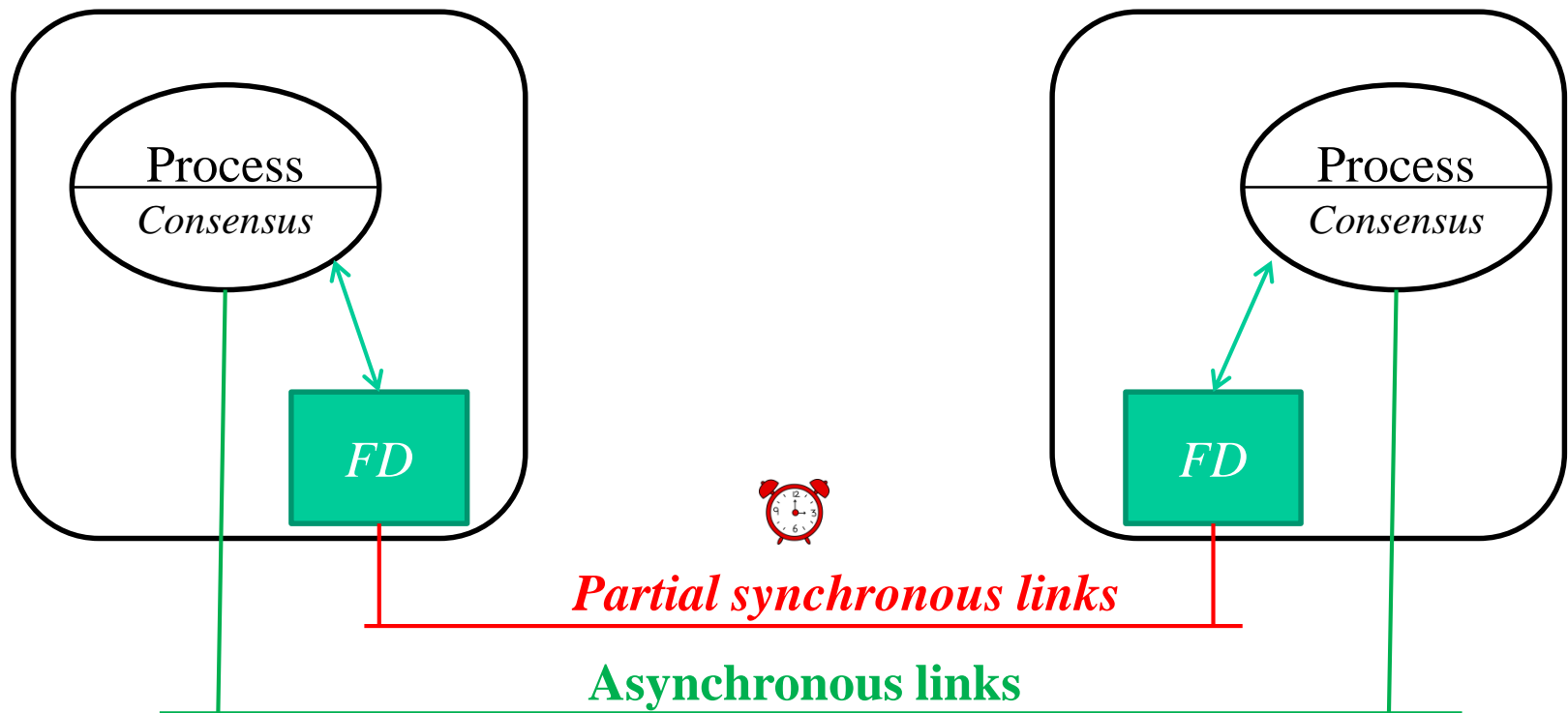
- Il suffit que le détecteur se stabilise pendant une période de temps suffisamment longue (la propriété de justesse « permanente » n'est valable que pendant l'algorithme)



Mise en œuvre des détecteurs de fautes

- Métriques
- Modèles temporels
- Implémentation
- Exemple : Passage à l'échelle

Implementation des FDs



Hypothèses temporelles (1)

- Implémentations reposant sur des temporisateurs :
 - Systèmes partiellement synchrones

Pour $\langle \rangle P$ (à terme, plus d'erreur)

- Il existe un temps (GST : Global stabilization time) où il y a une borne inconnue sur les délais de transmission et de traitement des messages (Modèle M3 [CHT 96])

⇒ Permet d'implémenter $\langle \rangle P$

Idée : à chaque erreur on augmente son temporisateur

⇒ Il existe un moment (après GST) où on ne fera plus d'erreur (le temporisateur a atteint la borne inconnue)

Eventual Perfect FD ($\Diamond P$)

Hypothèse : Le système devient synchrone après un temps GST (inconnu)

Initialisation: $\text{suspected}_i = \{\}; \forall j \neq i \in \{1, \dots, n\} \Delta_{i,j} = \Delta_0$

Task 1: repeat every Δ

 send HEARTBEAT to all – $\{p_i\}$

Task 2 : when did not receive HEARTBEAT during last $\Delta_{i,j}$ from p_j

$\text{suspected}_i = \text{suspected}_i \cup \{p_j\}$

Task 3: when received HEARTBEAT from p_j and $p_j \in \text{suspected}_i$

$\text{suspected}_i = \text{suspected}_i - \{p_j\}$

$\Delta_{i,j} = \Delta_{i,j} + 1$

Hypothèses temporelles (2)

Pour $\langle \rangle$ S et Ω (à terme plus d'erreur sur 1 processus) :

hypothèse réduite à un ensemble de canaux ultimement synchrones
(lien ultimement ponctuel $\langle \rangle$ -timely)

Définition: p est une \diamond_j -source: au moins j liens sortant de p sont ultimement ponctuels

Attention: la borne n n'est pas connue

Ω peut être implémenté si au il y a au moins une \diamond_f -source correcte
(f = nombre de défaillants)

Ω Algorithm

Suppose une \diamond **f**-source: $\exists p$ that has at least f output *eventually timely links*

on initialization:

- 1 $\forall q \neq p : \text{Timeout}[q] \leftarrow \eta + 1$
- 2 $\forall q : \text{counter}[q] \leftarrow 0, \text{suspect}[q] \leftarrow \emptyset$
- 3 $\forall q \neq p : \text{reset timer}(q) \text{ to } \text{Timeout}[q]$
- 4 **start tasks 0, 1 and 2**

task 0:

5 **repeat forever**

6 $\text{leader} \leftarrow \ell$ such that $(\text{counter}[\ell], \ell) = \min\{(\text{counter}[q], q) : q \in \Pi\}$

task 1:

7 **repeat forever**

8 send (ALIVE, *counter*) to all processes except p every η time

task 2:

9 **upon** receive (ALIVE, c) from q **do**

10 **for each** $r \in \Pi$ **do** $\text{counter}[r] \leftarrow \max\{\text{counter}[r], c[r]\}$

11 reset *timer*(q) to $\text{Timeout}[q]$

12 **upon** expiration of *timer*(q) **do**

13 $\text{Timeout}[q] \leftarrow \text{Timeout}[q] + 1$

14 send (SUSPECT, q) to all

15 reset *timer*(q) to $\text{Timeout}[q]$

16 **upon** receive (SUSPECT, q) from r **do**

17 $\text{suspect}[q] \leftarrow \text{suspect}[q] \cup \{r\}$

18 **if** $|\text{suspect}[q]| \geq n - f$ **then**

19 $\text{suspect}[q] \leftarrow \emptyset$

20 $\text{counter}[q] \leftarrow \text{counter}[q] + 1$

Idée : un vecteur
(counter) qui compte
le nombre de
suspensions

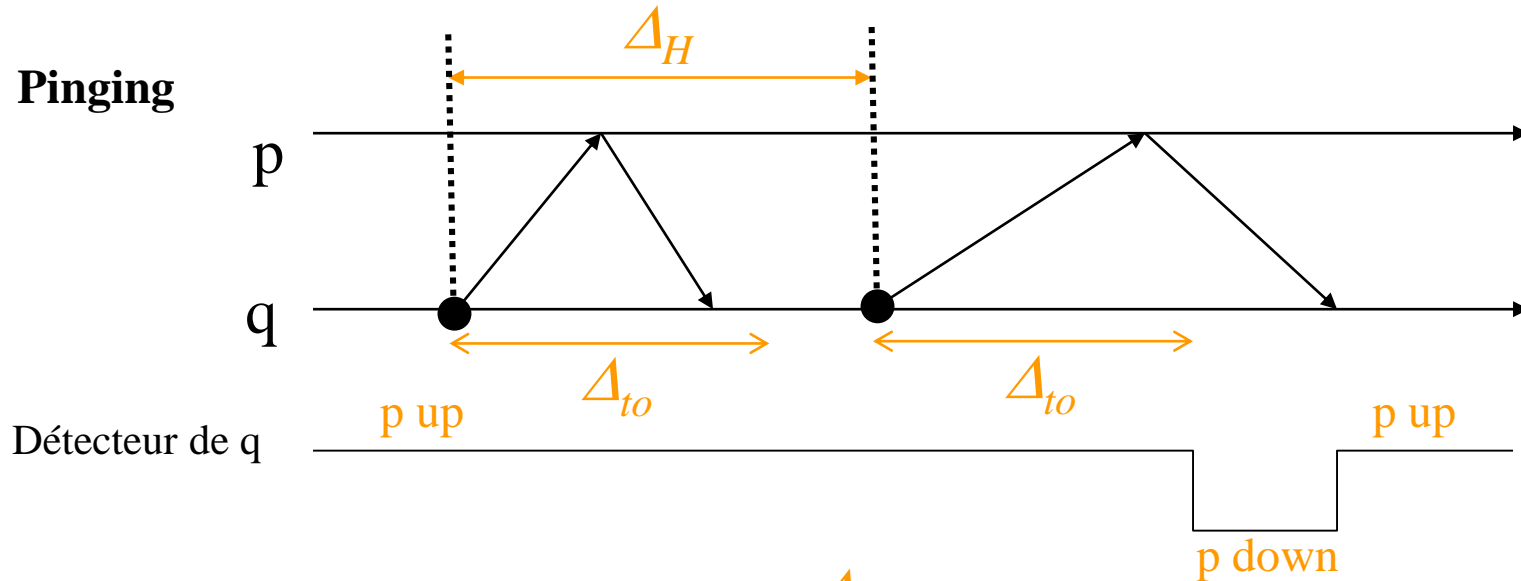
compteur de q
augmenté
uniquement si
suspecté par au
moins $n-f$
processus

Hypothèses temporelles (3)

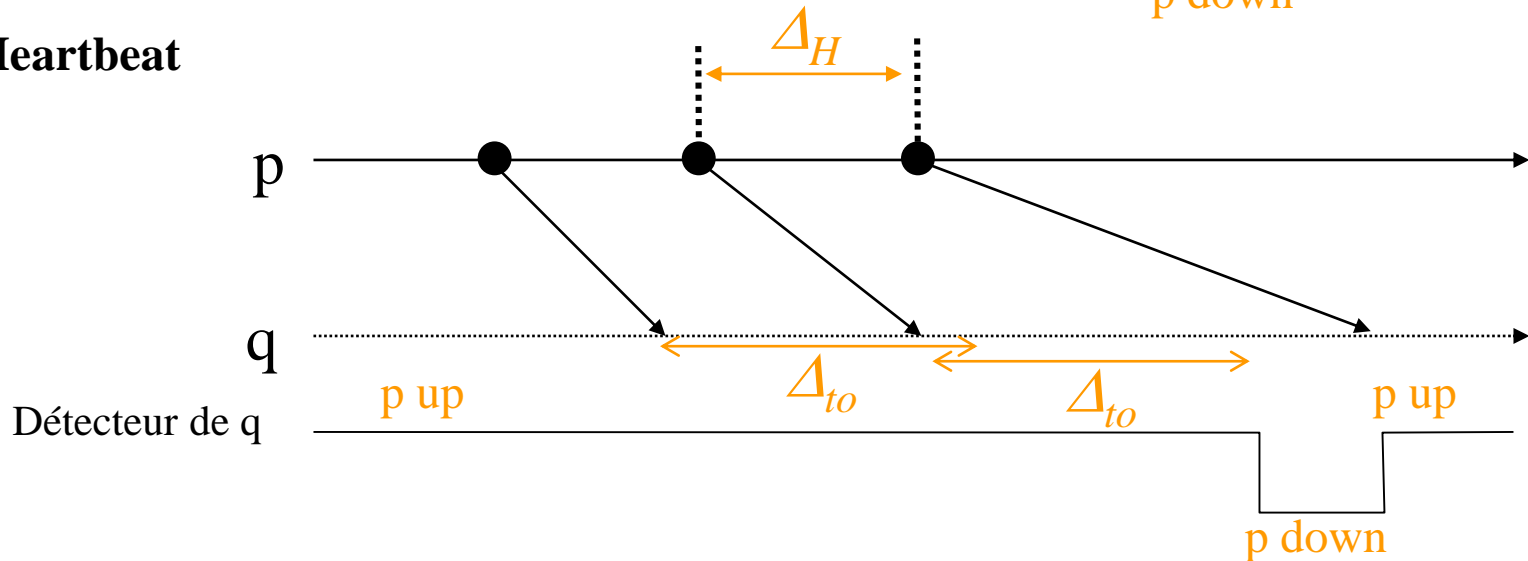
- Implémentations asynchrones (sans temporisateur):
 - Basées sur query-response (attendre un nombre fini de réponses – $n - f$)
 - Connaissance a priori du nombre de processus défaillants (f)
 - Hypothèse relative sur des canaux de communication (canaux plus rapides que d'autres)

Techniques d'implémentation

- **Pinging**

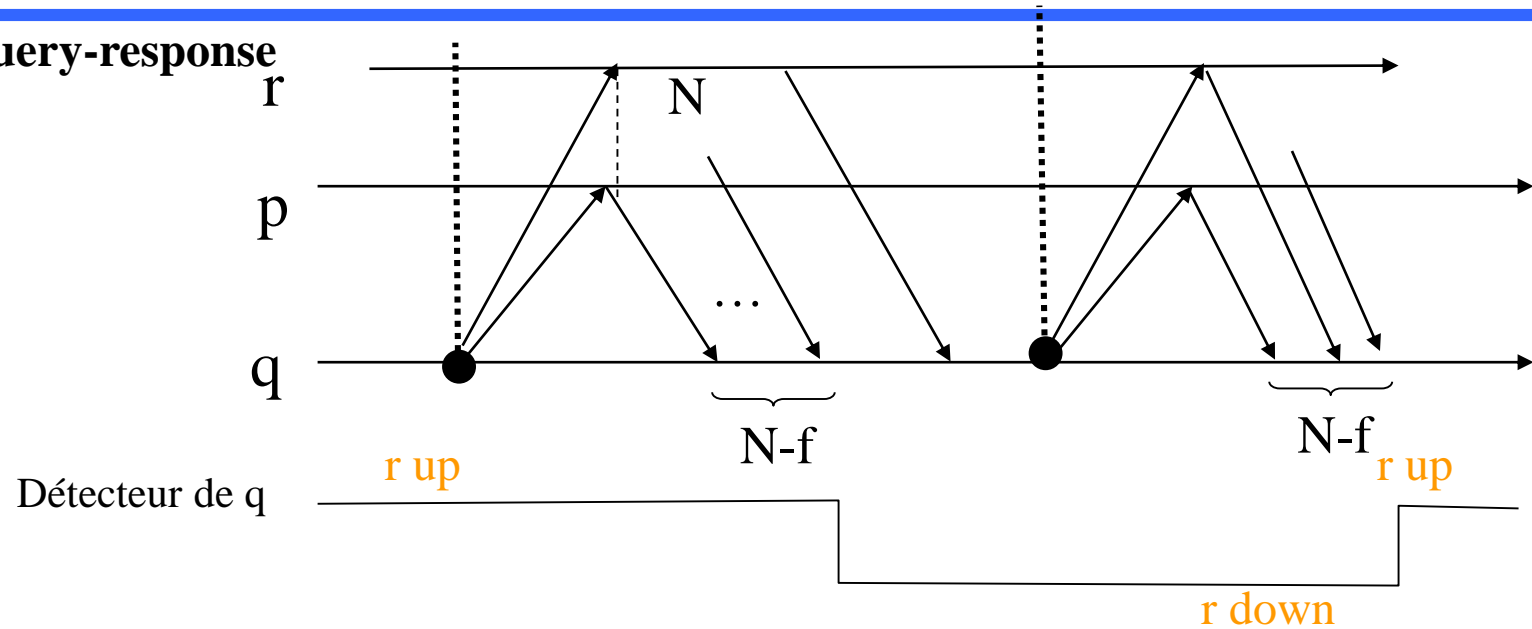


- **Heartbeat**



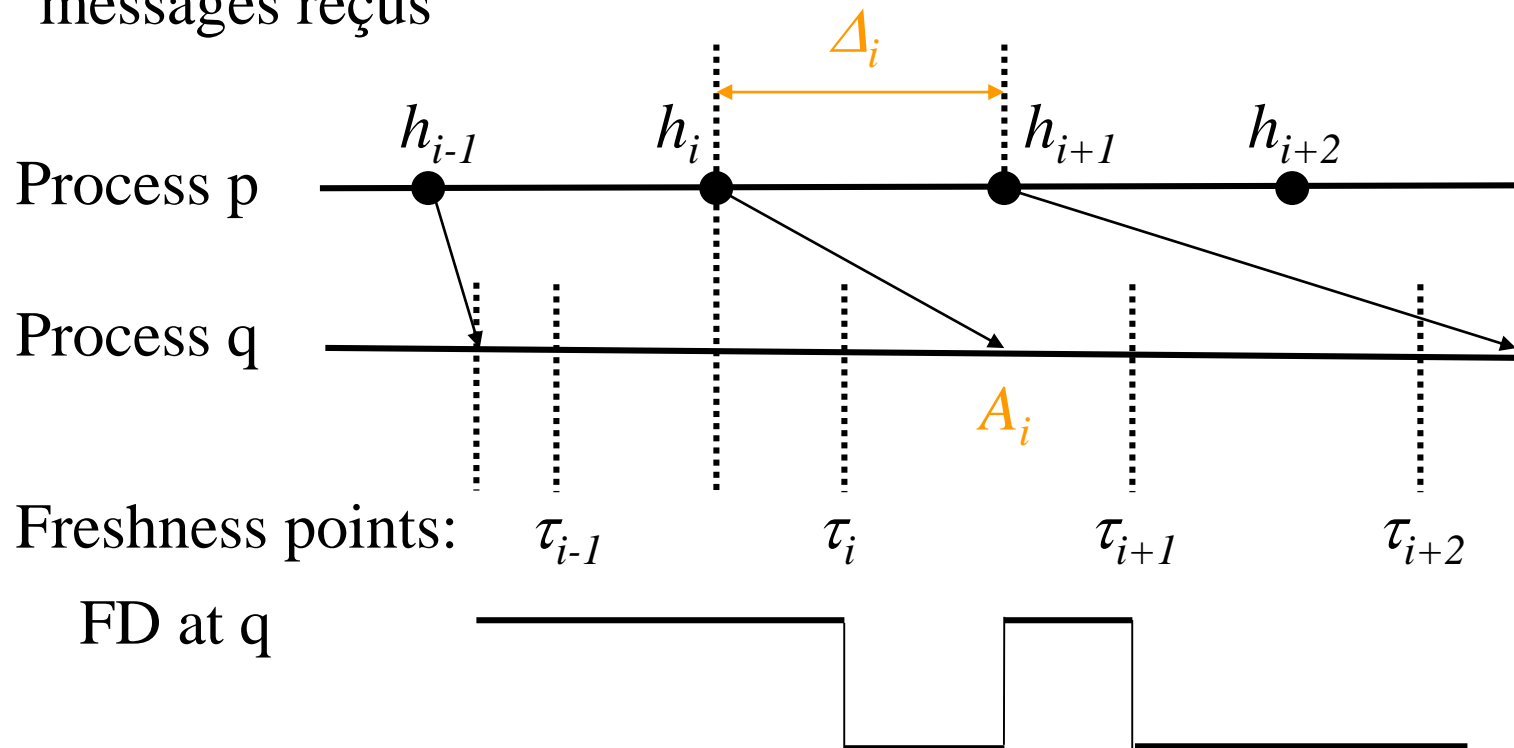
Implementations (2)

- Query-response



Estimation de Chen et al.

Idée : Définition de la date de réception du prochain heartbeat (freshness point) en fonction des N derniers messages reçus



Estimation du temporisateur

- Prochain temporisateur :
 - $Timeout(\tau_{k+1}) = \text{date}(EA_{k+1}) + \text{marge de sécurité}(\alpha_{k+1})$

- Date : Chen's estimation

$$EA_{(k+1)} \approx \frac{1}{n} \left(\sum_{i=k-n}^k A_i - \Delta_i \cdot i \right) + (k+1) \cdot \Delta_i$$

- Marge : Fixe (Chen et al.)

Dynamique (Bertier et al. – Basé sur RTT)

$$error_{(k)} = A_k - EA_{(k)} - delay_{(k)}$$

$$delay_{(k+1)} = delay_{(k)} + \gamma \cdot error_{(k)}$$

$$var_{(k+1)} = var_{(k)} + \gamma \cdot (|error_{(k)}| - var_{(k)})$$

$$\alpha_{(k+1)} = \beta \cdot delay_{(k+1)} + \phi \cdot var_{(k+1)}$$

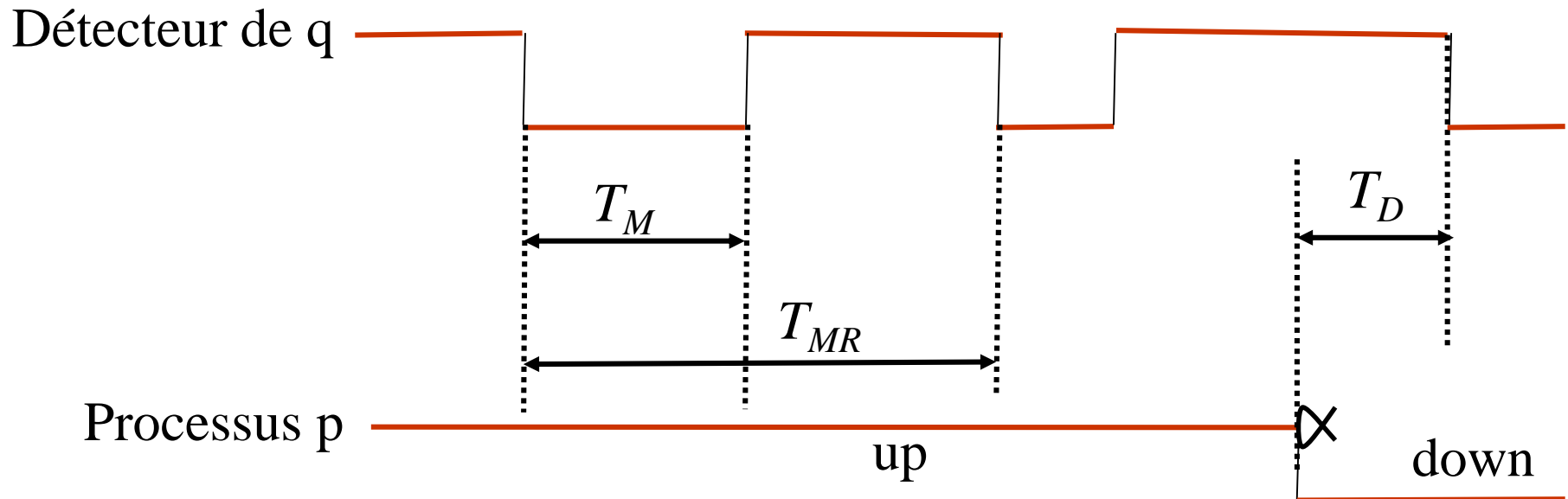
Métriques : Qualité de détection (QoD)

- Complétude

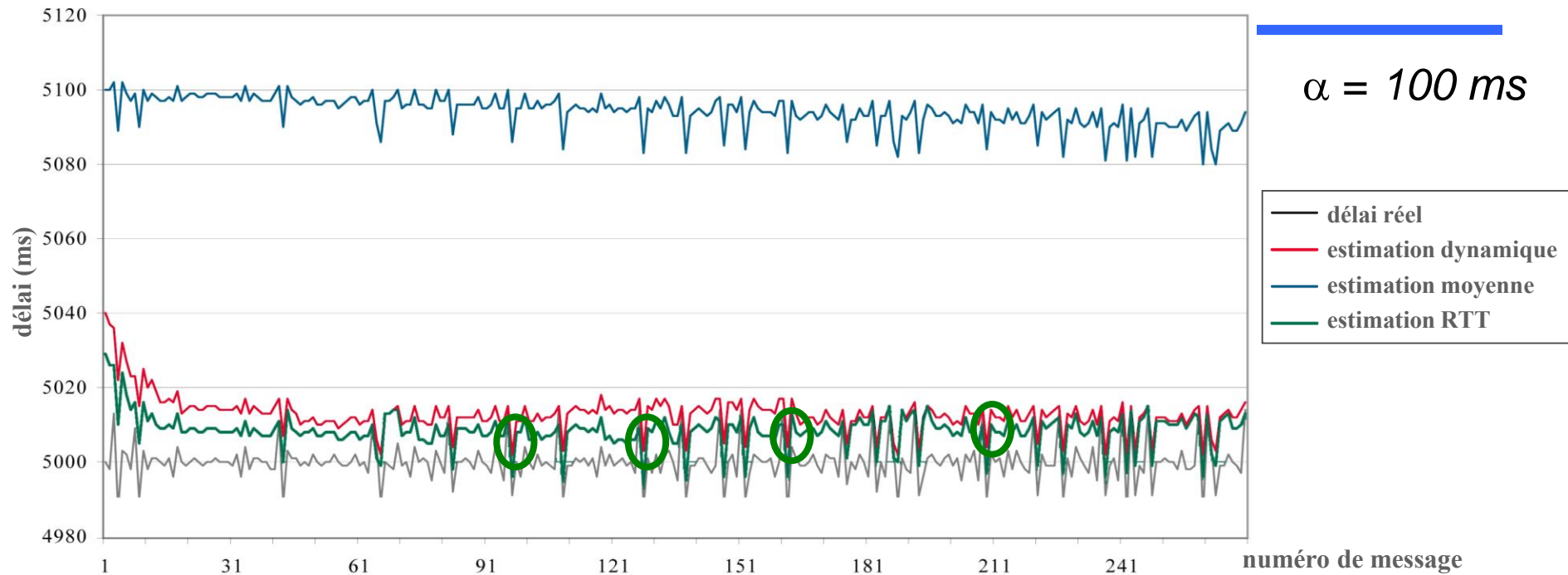
- Temps de détection (T_D)

- Justesse

- Temps entre deux erreurs (T_{MR})
- Durée des erreurs (T_M)



Performances



	Estimation RTT	Estimation de Chen	Estimation dynamique
Nombre de fausses détections	4	0	0
Temps de détection moyen (ms)	5011,9	5089,9	5016,6

Comparaison

Durée : 38 heures

Utilisation normale du laboratoire

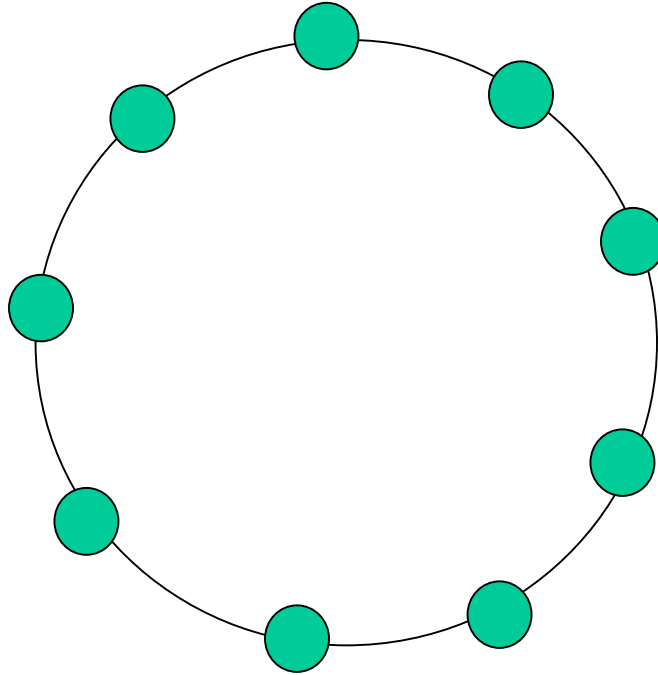
	Historic + RTT	RTT	Historic
False detection	24	54	29
Mistake duration (ms)	31,6	25,23	36,61
Detection time (ms)	5131,7	5081,79	5672,53

Passage à l'échelle des FD

Peu d'implementation à grande échelle

- Structure en anneau [WLL07] – PDC
- Approche probabiliste [GCG01] – PODC
- Organisation hierarchique [BMS03] – DSN
- **Hypercube pour HPC** [BBGHR16] – Supercomputing

Structure en anneau



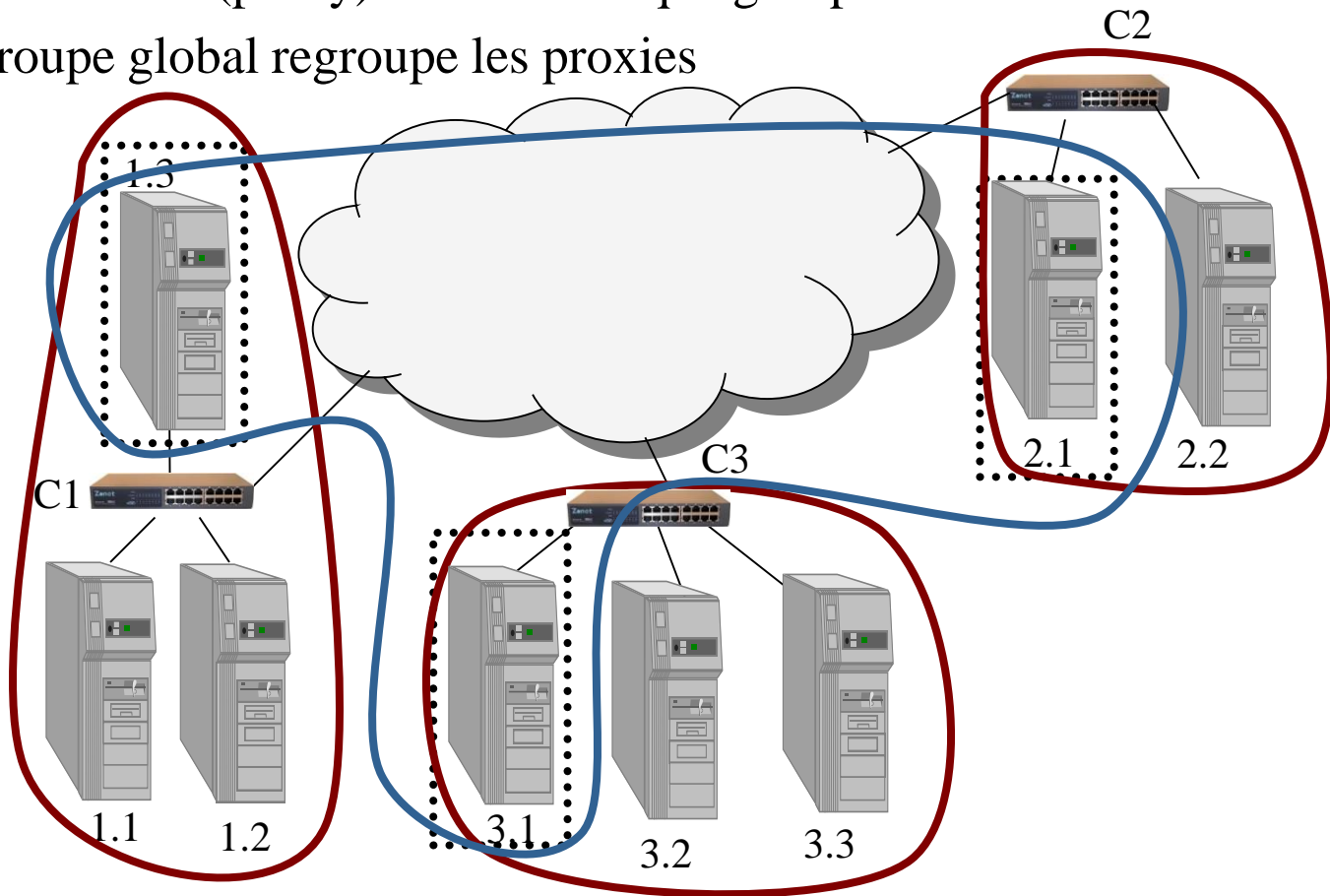
- Chaque noeud envoie un heartbeat à son successeur
- Faible coût en messages
- Temps de détection longs (propagation des information dans l'anneau)

Approche probabiliste

- A chaque ronde, chaque nœud choisit aléatoirement un nœud distant à surveiller
- Assure une complétude et justesse probabiliste
- Passe à l'échelle
- Difficulté pour dimensionner les temporisateurs

Organisation hiérarchique

- Cluster => groupe de détection local
- Un mandataire (proxy) élu dans chaque groupe local
- Un groupe global regroupe les proxies



Failure detection and propagation in HPC

George Bosilca¹, Aurélien Bouteiller¹, Amina Guermouche¹,
Thomas Hérault¹, Yves Robert^{1,2}, Pierre Sens³, and Jack
Dongarra^{1,4}

University Tennessee Knoxville¹

ENS Lyon, France²

LIP6, Inria Paris, France³

University of Manchester, UK⁴

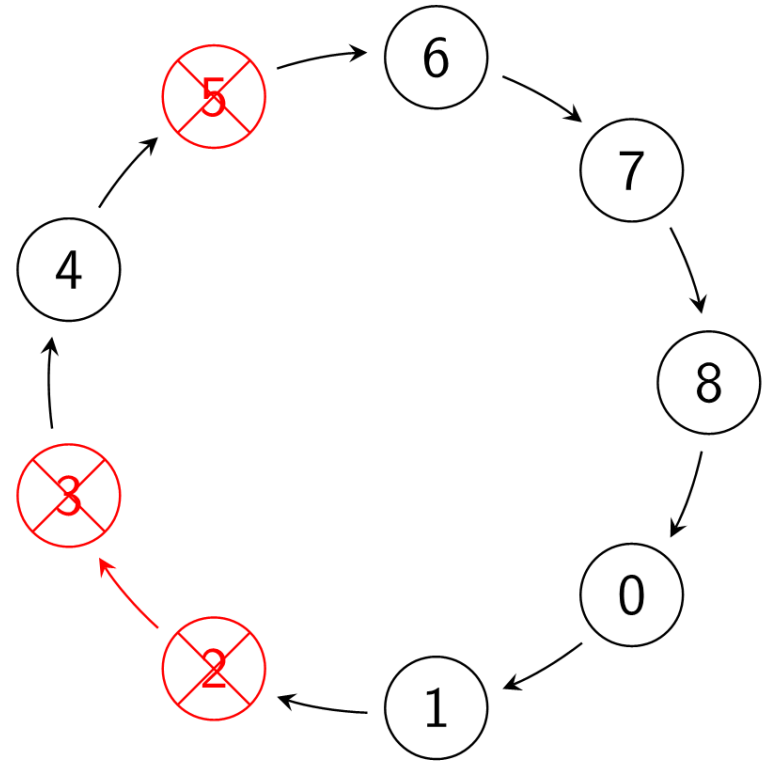
SC'16 – November, 2016

Resilient applications in HPC context

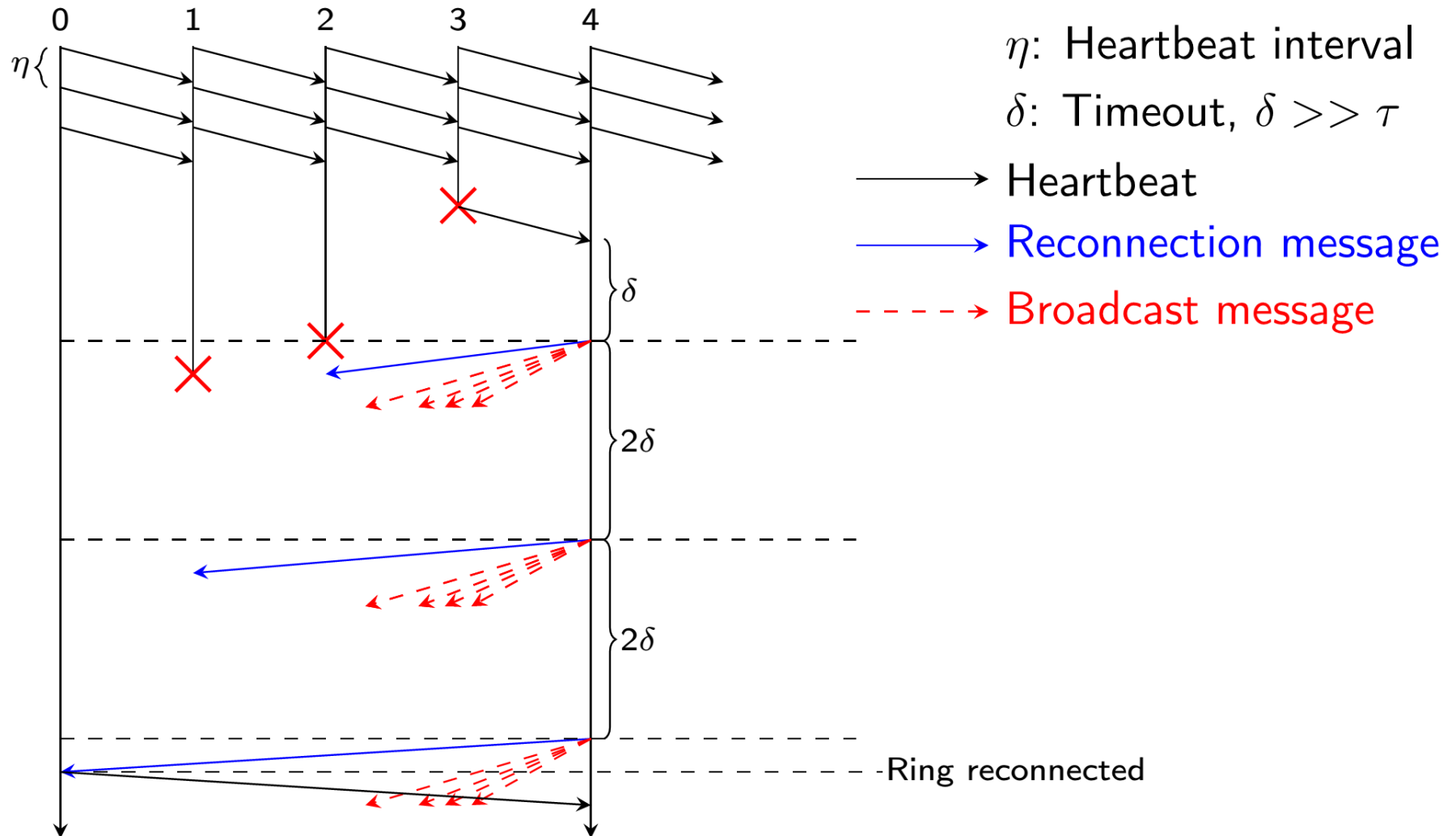
- Applications continue execution after crash of several nodes (nodes eventually occurs)
- Need rapid and global knowledge of group members
- 3 key features:
 1. Rapid: failure detection
 2. Global: failure knowledge propagation
 3. Resilience mechanism should have minimal impact

Failure detection

- Processes arranged as a ring
- Periodic heartbeats from a node to its successor
- **Maintain ring of live nodes**
 - Reconnect ring after a failure
 - Inform all processes

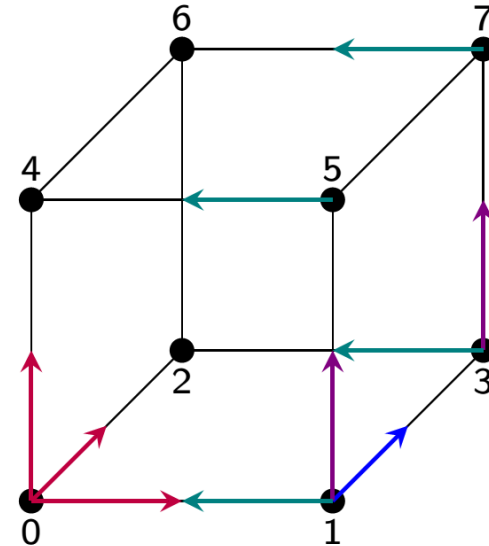


Reconnecting the ring



Broadcast algorithm

- Hypercube Broadcast Algorithm [1]
- Disjoint paths to deliver multiple broadcast message copies
- Completes if $f \leq \log(n) - 1$
(f : number of failures, n : number of live processes)



Receiver Node	Path via		
	Node1	Node2	Node4
1	0	0-2-3	0-4-5
2	0-1-3	0	0-4-6
3	0-1	0-2	0-4-5-7
4	0-1-5	0-2-6	0
5	0-1	0-2-6-7	0-4
6	0-1-3-7	0-2	0-4
7	0-1-3	0-2-6	0-4-5

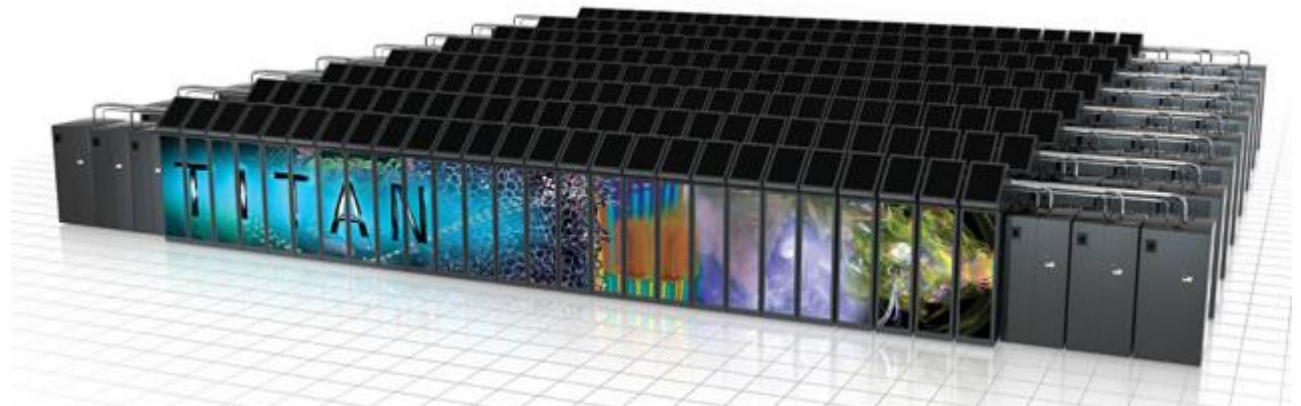
[1] P. Ramanathan and Kang G. Shin, 'Reliable Broadcast Algorithm', IEEE Trans. Computers, 1988

Implementation in MPI

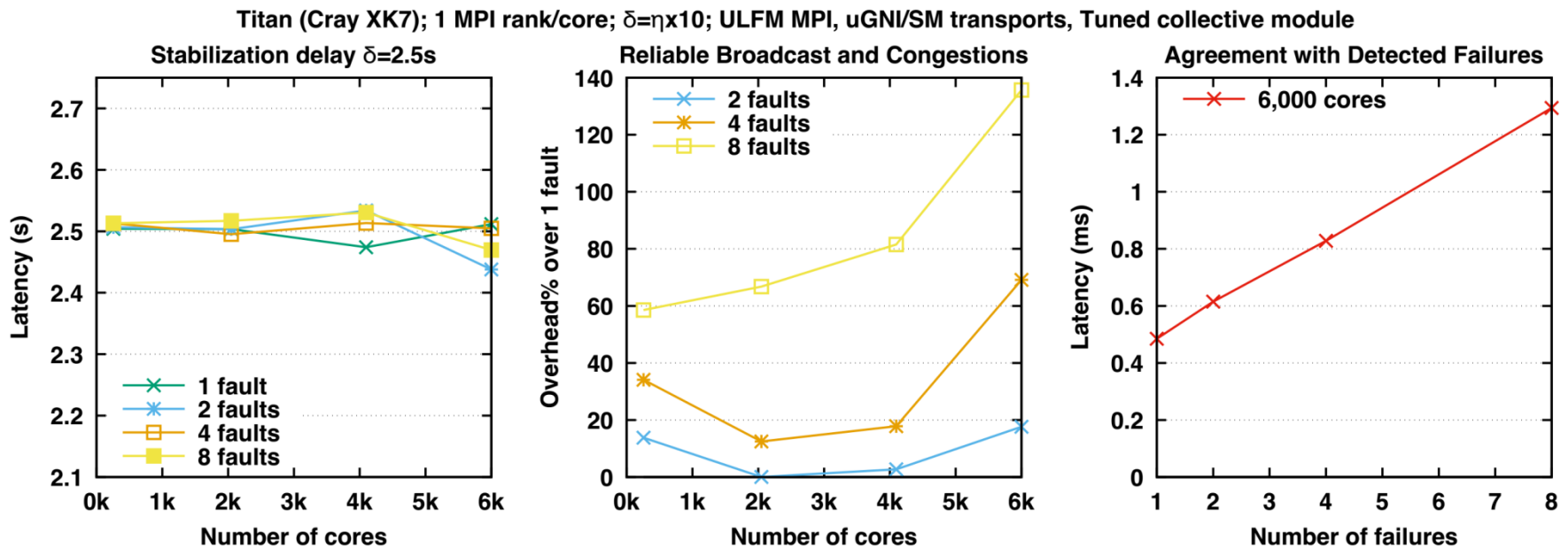
- Implementation in ULFM / Open MPI
 - Observation ring and propagation topology implemented in Byte Transport Layer (BTL)
 - MPI internal thread independently from application communications
 - RDMA put channel to directly raise a flag at receiver memory
- No allocated memory, no message wait queue

Experimental setup

- Titan ORNL Supercomputer
- 16-core AMD Opteron processors
- Cray Gemini interconnect
- ULFM / OpenMPI 2.x
- Up to 6, 000 cores
- Average of 30 times



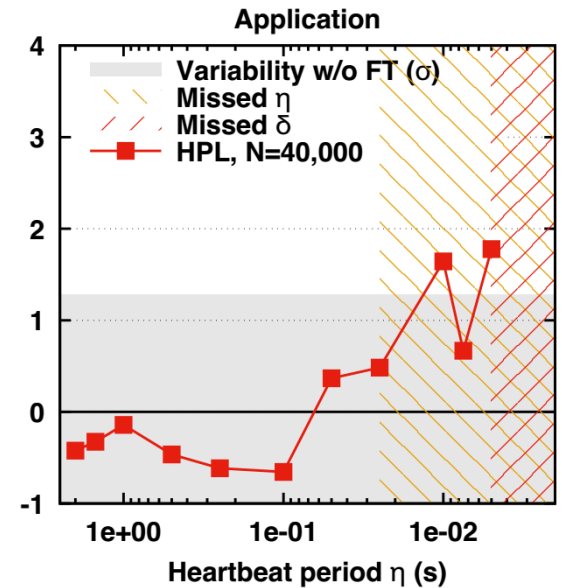
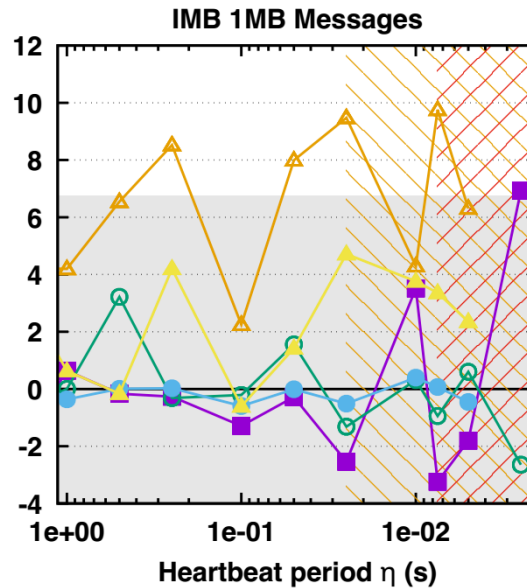
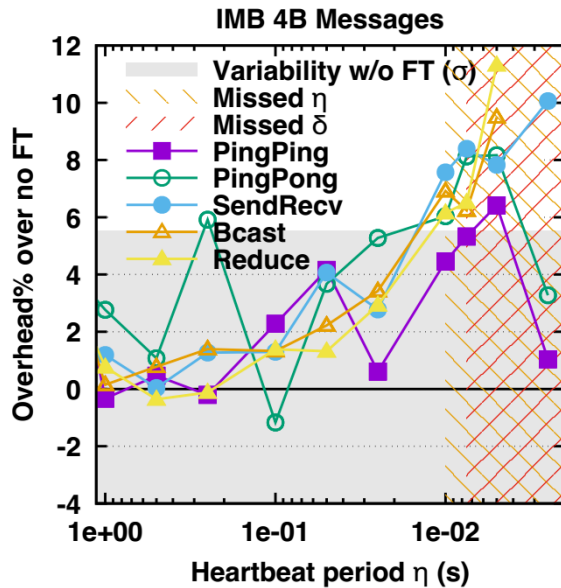
Detection and propagation delay



Noise

- $\delta = 10 \times \eta$

Titan (Cray XK7); 256 MPI ranks on 256 cores; $\delta = \eta \times 10$; ULFM MPI, uGNI/SM transports, Tuned collective module



Conclusions

- Points forts des détecteurs non fiables :
 - Détection de défaillances comme abstraction : permet de s'abstraire de synchronisme
 - Détection rapide des fautes (aspect non fiable)
 - Permet de contourner FLP
- Points faibles des détecteurs :
 - Hypothèse de fiabilité des canaux \Rightarrow il existe des détecteurs supposant des canaux « équitable devant les fautes » (fair lossy channel :)
 - Propriété perpétuelle de justesse peu réaliste \Rightarrow perpétuité restreinte à la durée de l'algorithme
 - Modèle de fautes simple (crash) : extension vers fautes omission

Références

- [BSM 03] M. Bertier, O. Marin, and P. Sens. Performance analysis of a hierarchical failure detector. In Proceedings of the International Conference on Dependable Systems and Networks, June 2003.
- [BBGHR16] G. Bosilca, A. Bouteiller, A. Guermouche, T. Hérault, Y. Robert, P. Sens, J. Dongara. Failure Detection and Propagation in HPC systems. SC 2016 - The International Conference for High Performance Computing, Networking, Storage and Analysis, Salt Lake City, United States, Nov 2016.
- [Cha90] S. Chaudhuri. Agreement is harder than consensus : set consensus problems in totally asynchronous systems. In Proceedings of the ninth annual ACM symposium on Principles of distributed computing, pages 311--324. ACM Press, 1990.
- [CHT 96] T. D. Chandra, V. Hadzilacos, and S. Toueg. The weakest failure detector for solving consensus. Journal of the ACM, 43(4) :685722, 1996.
- [CTA 00] W. Chen, S. Toueg, M. K. Aguilera. On the quality of service of failure detectors. In Proc. of First Conference on Dependable Systems and Networks, June 2000.
- [DFG10] C. Delporte-Gallet, H. Fauconnier, R. Guerraoui: Tight failure detection bounds on atomic object implementations. J. ACM 57(4): 22:1-22:32 (2010)
- [DKGPS15] Swan Dubois, Rachid Guerraoui, Petr Kuznetsov, Franck Petit, Pierre Sens: The Weakest Failure Detector for Eventual Consistency. PODC 2015: 375-384
- [DLS88] Dwork, C., Lynch, N., and Stockmeyer, L. Consensus in the presence of partial synchrony. Journal of the ACM 35, 2 (Apr.), 288–323, 1988
- [DFGT08] C. Delporte-Gallet, H. Fauconnier, R. Guerraoui, A. Tielmann: The Weakest Failure Detector for Message Passing Set-Agreement. DISC 2008: 109-120
- [FLP 85] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. Journal of the ACM, 32(2) :374--382, Apr 1985.

Références

- [GCG01] I. Gupta, T. D. Chandra, and G. S. Goldszmidt. On scalable and efficient distributed failure detectors. In Proceedings of the twentieth annual ACM symposium on Principles of distributed computing, pages 170-179. ACM Press, 2001.
- [LAF99] M. Larrea, S. Arévalo, and A. Fernandez. Efficient algorithms to implement unreliable failure detectors in partially synchronous systems. In Proceedings of the 13th International Symposium on Distributed Computing, pages 344-358. Springer-Verlag, 1999.
- [LFA00] M. Larrea, A. Fernández, and S. Arévalo. Optimal implementation of the weakest failure detector for solving consensus. In Proceedings of the 19th Annual ACM Symposium on Principles of Distributed Computing (PODC 2000), pages 334-344, NY, July 16-19 2000. ACM Press.
- [GK09] E. Gafni, P. Kuznetsov: The weakest failure detector for solving k-set agreement. PODC 2009: 83-91
- [LH94] W-K. Lo, Vassos Hadzilacos: Using Failure Detectors to Solve Consensus in Asynchronous Shared-Memory Systems (Extended Abstract). WDAG 1994: 280-295
- [MMR03] A. Mostefaoui, E. Mourgaya, M. Raynal, "Asynchronous Implementation of Failure Detectors", Proc. Int'l IEEE Conf. Dependable Systems and Networks (DSN '03), pp. 351-360, 2003.
- [WLL07] J. Wieland, M. Larrea, A. Lafuente, An evaluation of ring-based algorithms for the Eventually Perfect failure detector class. PDP 2007: 163-170
- [Z10] P. Zielinski: Anti-Omega: the weakest failure detector for set agreement. Distributed Computing 22(5-6): 335-348 (2010)