
Consensus (2)

Paxos

Pierre.Sens@lip6.fr

Master 2 – Informatique / SAR
ARA

Références

- **Paxos Made Simple**

Leslie Lamport

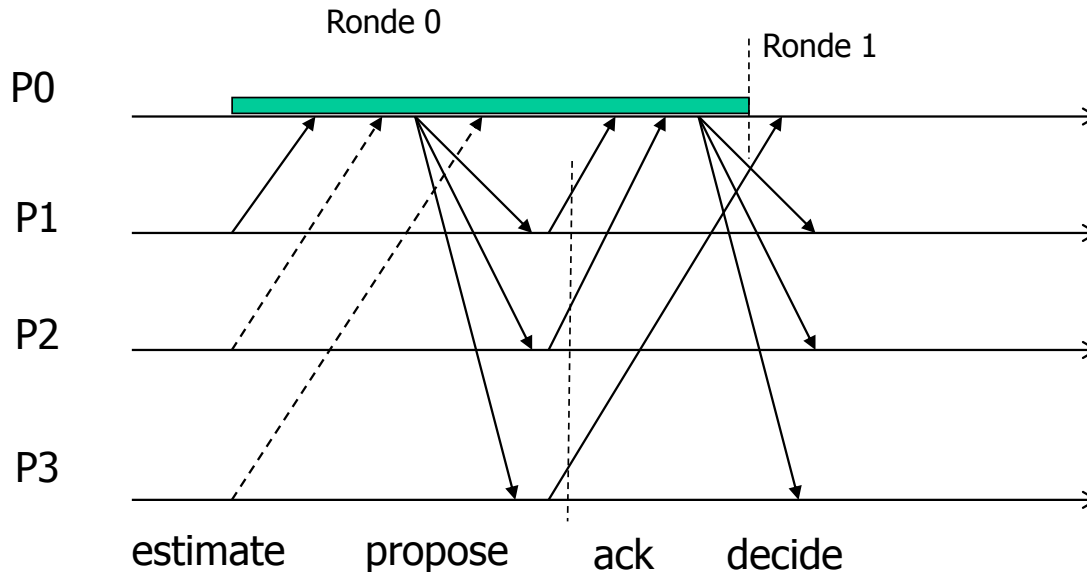
ACM SIGACT News (Distributed Computing Column) 32, 4 18-25(121), Décembre 2001

- Historique :

- "Part-Time Parliament" [Lamport 88,98,01]
- Protocole inspiré du fonction du Parlement sur l'Ile de Paxos antique : le Parlement a fonctionné, malgré l'absence régulière des législateurs et l'oubli des informations de leur messenger.

Limite de l'approche à coordinateur tournant (CHT96)

- Nécessite des canaux fiable (peu réaliste)
- Impact de la perte de message : Blocage



Si on n'arrive pas à collecter une majorité → bloqué dans une ronde.

Au delà du coordinateur tournant

- Eviter d'être bloqué dans une ronde
 - Changer de ronde sur l'expiration d'un temporisateur
 - Changer de ronde si d'autres nœuds ont changé
- Maintenir le principe du leader pour connaître les valeurs courantes (majorité)

=> Algorithme de Paxos

Basé sur Ω

Paxos : Les hypothèses



- Communication
 - Asynchrone
 - Pas d'altération de messages
 - **Possibilité de pertes**
- Processus
 - Nombre fixe
 - Fautes **franches** avec possibilité de reprise (crash-recovery). Chaque processus possède un état persistant

Principes de Paxos (Παξος)

- Repose sur un leader (utilisation d'un détecteur Ω)
 - Le leader démarre un nouveau “**ballot**” (i.e.,ronde, vue, scrutin)
 - Cherche à joindre une majorité d'agents
- Les agents rejoignent toujours les “ballots” les plus récents (ignore les “ballots” anciens)
- Deux phases :
 - 1) Collecter les résultats des scrutins (ballot) précédents de la part d'une majorité d'agent
 - 2) Puis proposer une nouvelle valeur, et obtenir une majorité pour l'approuver
- L'algorithme s'arrête si il existe un leader unique pendant les 2 tours d'échanges avec une majorité de d'agents
- Remarques :
 - Il peut y avoir plusieurs leader concurrents
 - Les numéros de ballot permettent de distinguer les valeurs proposées par les différents leader



Paxos : Structure – Les numéros de ballots

- Chaque valeur est associée à **un numéro de ballot**
- Numéro de ballots avec un ordre total
- Pairs $\langle \text{num}, \text{process id} \rangle$
- $\langle n_1, p_1 \rangle > \langle n_2, p_2 \rangle$
 - Si $n_1 > n_2$  n^0 de ronde / saut in
 - ou $n_1 = n_2$ et $p_1 > p_2$  ordre total sur tous les processus
- Le leader courant p choisit localement un numéro unique croissant :
 - Si le dernier ballot connu est $\langle n, q \rangle$
alors p choisit $\langle n+1, p \rangle \Rightarrow$ le couple est unique.

Paxos : Structure – Les variables locales

- Val_i *doit être la même pour tout le monde à terme.*
Valeur courante
- $BallotNum_i$, initialement $\langle 0, 0 \rangle$
Numéro du dernier ballot auquel p_i a pris part (phase 1)
- $AcceptNum_i$, initialement $\langle 0, 0 \rangle$
Numéro du ballot associé à la dernière valeur acceptée par p_i (phase 2)
- $AcceptVal_i$, initialement \perp
Dernière valeur acceptée (phase 2)

Phase 1 : Préparation (Prepare)

- Objectif : demander à joindre le tour (ballot) courant et collecter les informations des décisions passées
- Périodiquement sur p_i (jusqu'à que la décision soit prise) :

Si leader = p_i **alors**

BallotNum $_i$ = \langle BallotNum $_i$.num+1, p_i \rangle

send (“prepare”, BallotNum $_i$) à tous

- Réception sur p_j (“prepare”, bal) de p_i :

Si bal \geq BallotNum $_j$ **alors**

BallotNum $_j$ \leftarrow bal

send (“ack”, bal, AcceptNum $_j$, AcceptVal $_j$) à p_i

output
de Ω

↑
moment d'acceptation
↑
valeur précédente

Phase 2 : Acceptation

Réception (“ack”, BallotNum, b , val) sur p_i de $n-f$ processus
(une majorité)

Si toutes les $vals = \perp$ **alors** $Val_i = \text{initial value}$

sinon $Val_i = \text{la valeur } val \text{ associé au plus grand } b$

send (“accept”, BallotNum, Val_i) à tous */* proposition */*

Réception sur p_j (“accept”, b , v)

Si $b \geq \text{BallotNum}_j$ **alors**


*la valeur associée au
ou si j'en ai pas une.*

$\text{AcceptNum}_j \leftarrow b$; $\text{AcceptVal}_j \leftarrow v$ */* Acceptation */*

send (“accept”, b , v) à tous (*uniquement la première fois*)

Paxos : Décision

Réception (“accept”, b, v) de $n-f$ processus (majorité)

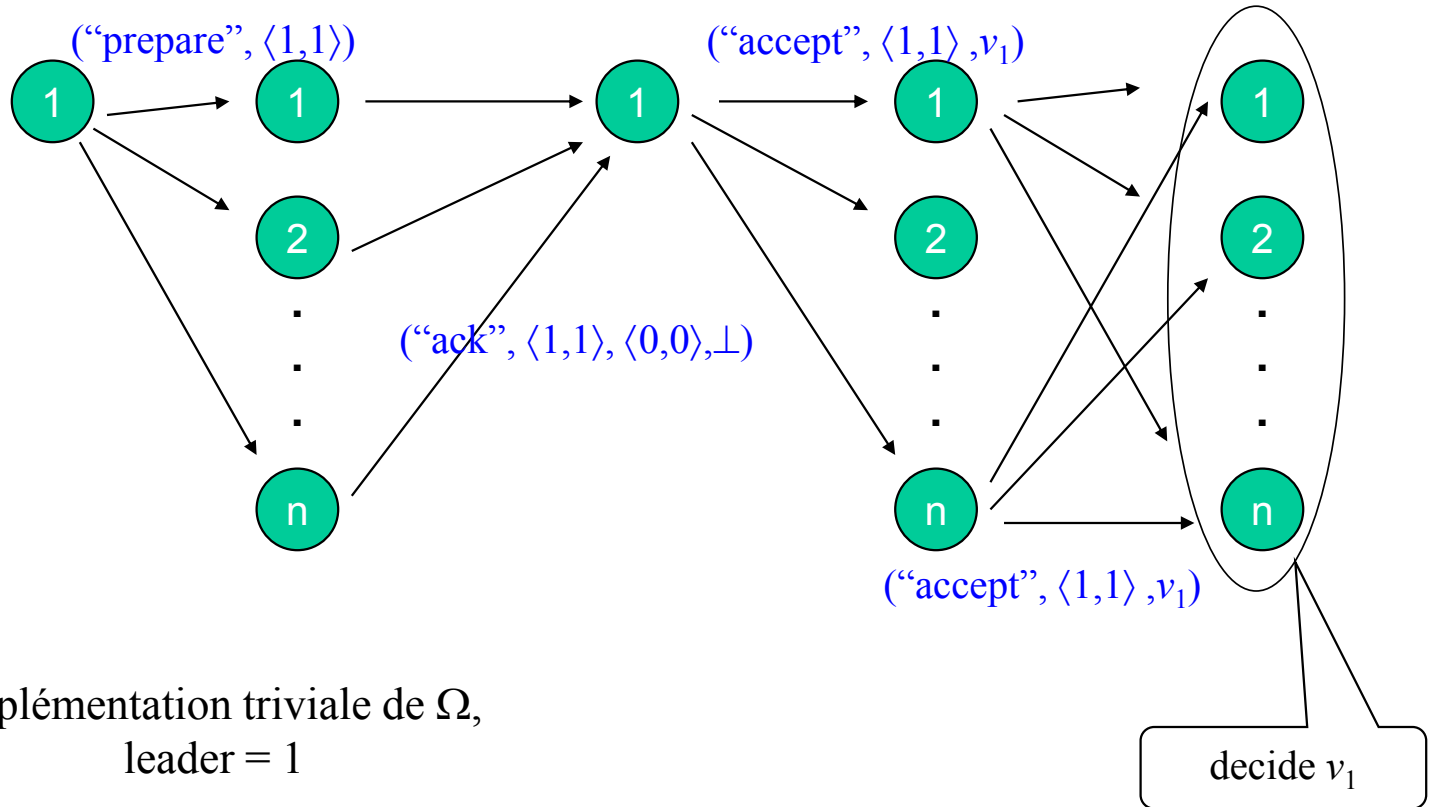
decider v  Si la valeur a été acceptée par une majorité de processus, même si il y a un problème, la valeur va perdurer.
périodiquement send (“decide”, v) à tous

Réception (“decide”, v)

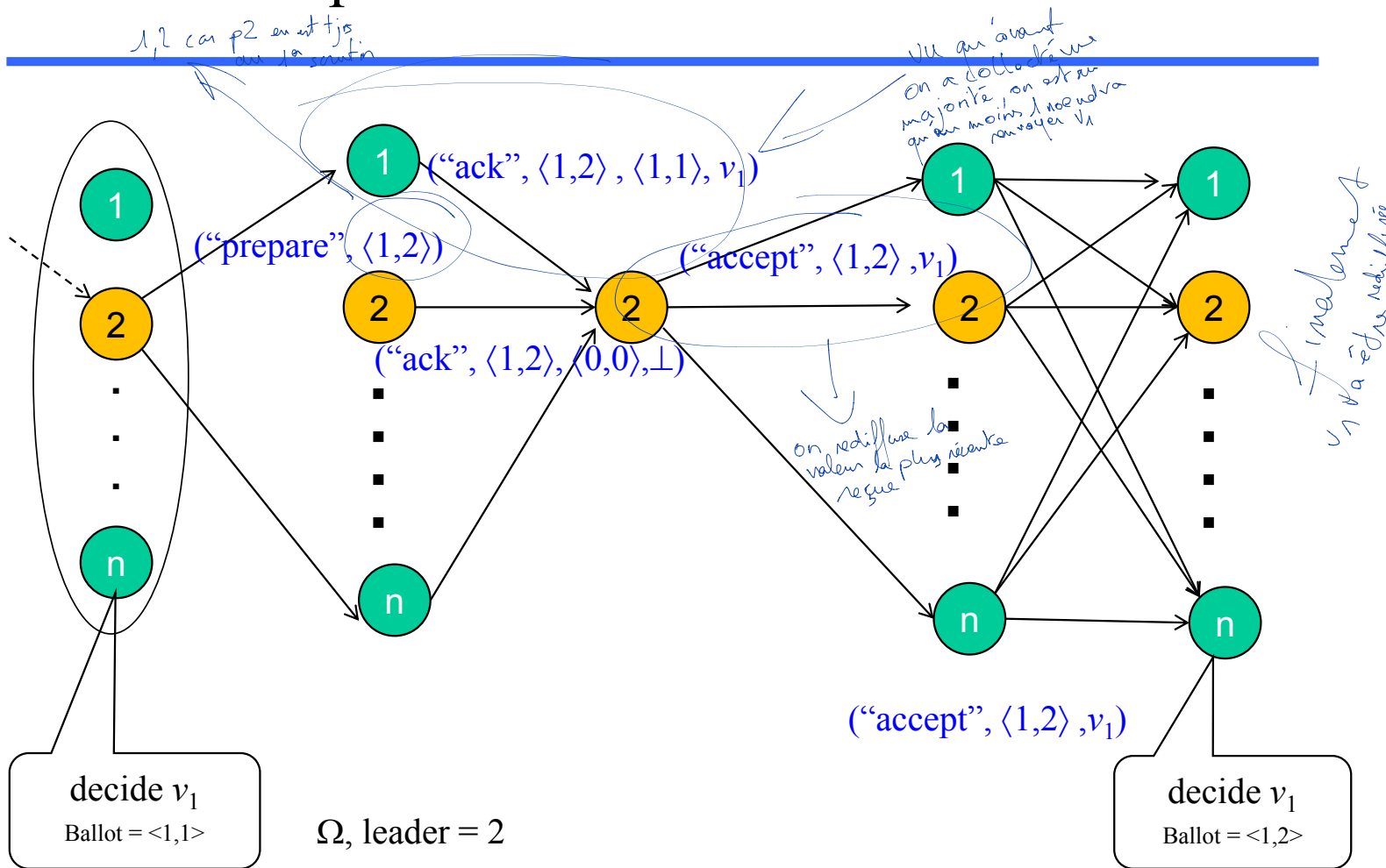
decider v

Tolérer les pertes de messages : si une valeur tarde
=> changement de ballot (de rondes)

Exemple d'exécution sans faute



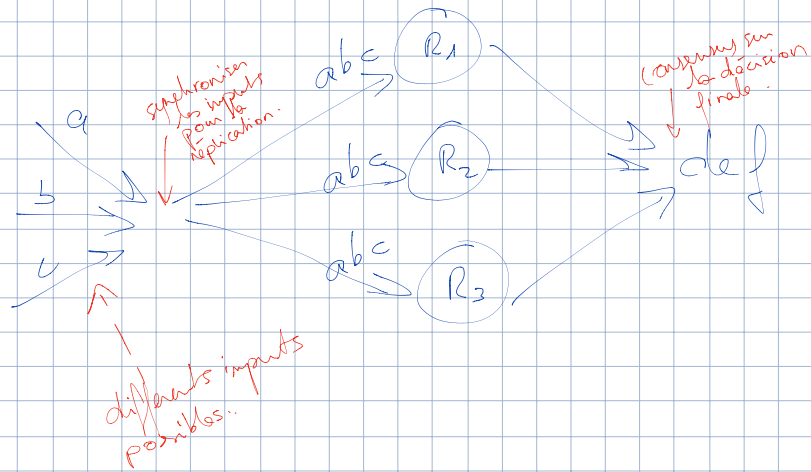
Exemple d'exécution avec deux leaders



Caractéristiques clés de Paxos

- Hypothèse faible (asynchrone, perte de message, crash-recovery)
- Intérêt pratique
- Optimisations possibles
 - Le processus 1 peut directement proposer sa valeur
- Terminaison non assurée (FLP toujours valable) sauf si les canaux sont ultimement fiables

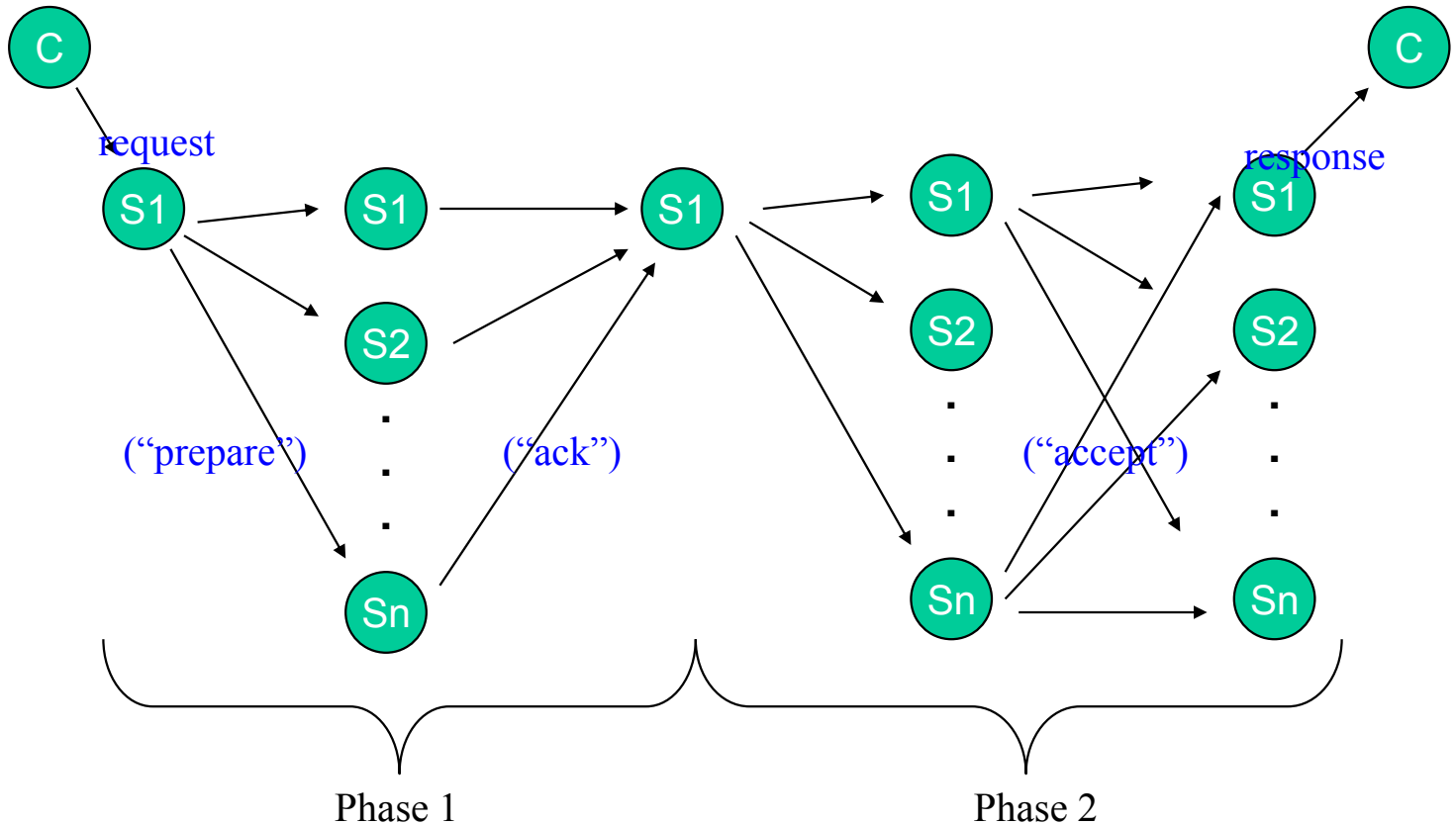
↳ pour être sûr que l'algo converge il faut passer à un moment par des canaux synchrones (FLP)



Utilisation de Paxos pour une machine à états répliquée

- SMR : State-Machine Replication
- Données dupliquées sur n serveurs
- Des clients génèrent des opérations
- Les opérations doivent être effectuées par **tous** les serveurs corrects dans le **même ordre**
 - Accord sur la séquence d'opérations
 - Equivalent à la diffusion atomique = diffusion fiable + totalement ordonnée

Paxos pour SMR



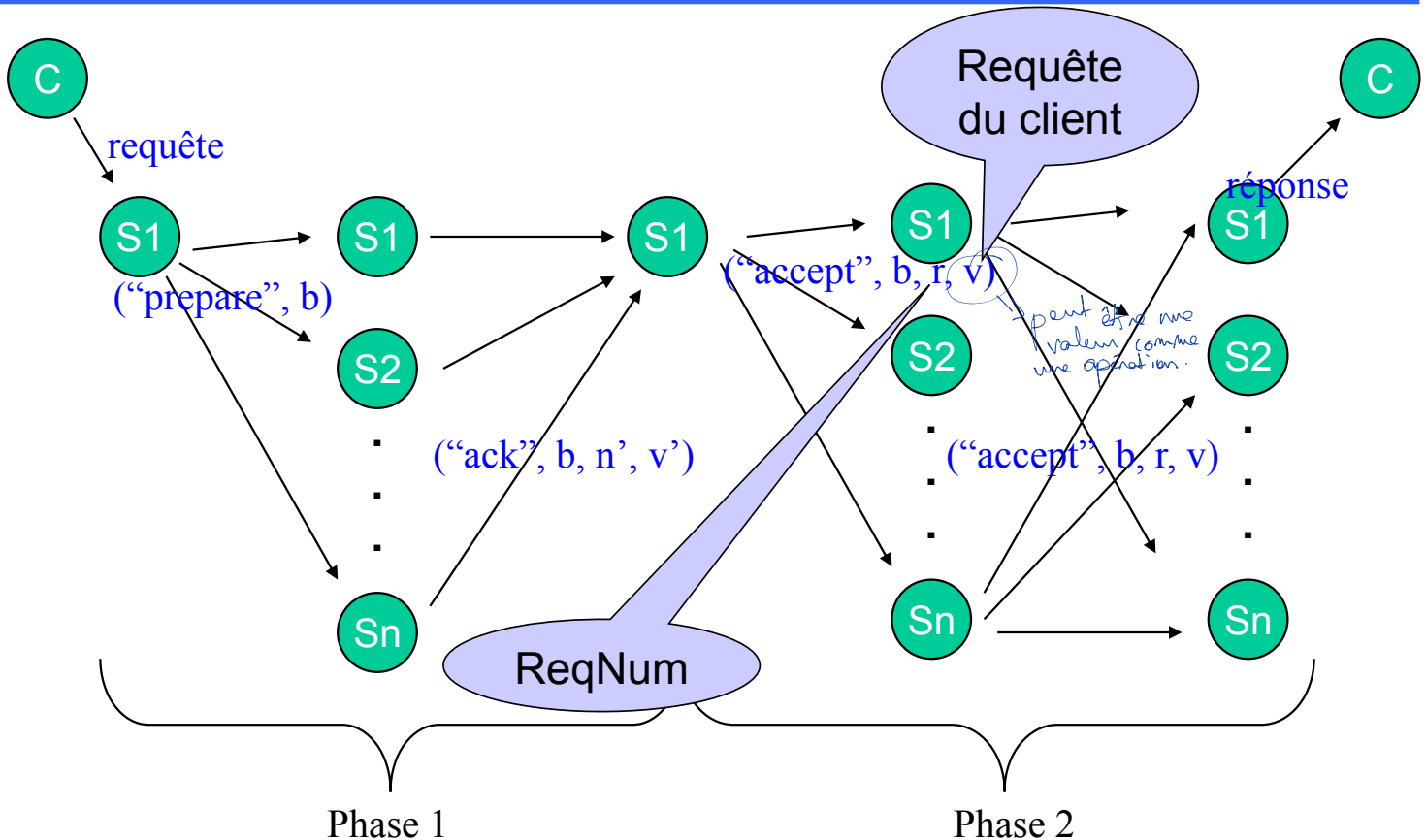
Adaptation de Paxos

- Plusieurs instances de consensus en parallèle (1 par requête) => Ajout d'un numéro de requête ReqNum par client
- Transformation des variables locales en tableaux (non bornés)
 - AcceptNum[r], AcceptVal[r], $r = 1, 2, \dots$
- Ajout du numéro de requêtes dans les messages accept
- Ordre des operations sur la machine à états
 - AcceptVal[1], puis AcceptVal[2], etc.
 - Après leur consensus respectif une réponse est renvoyée au client (uniquement par le leader)

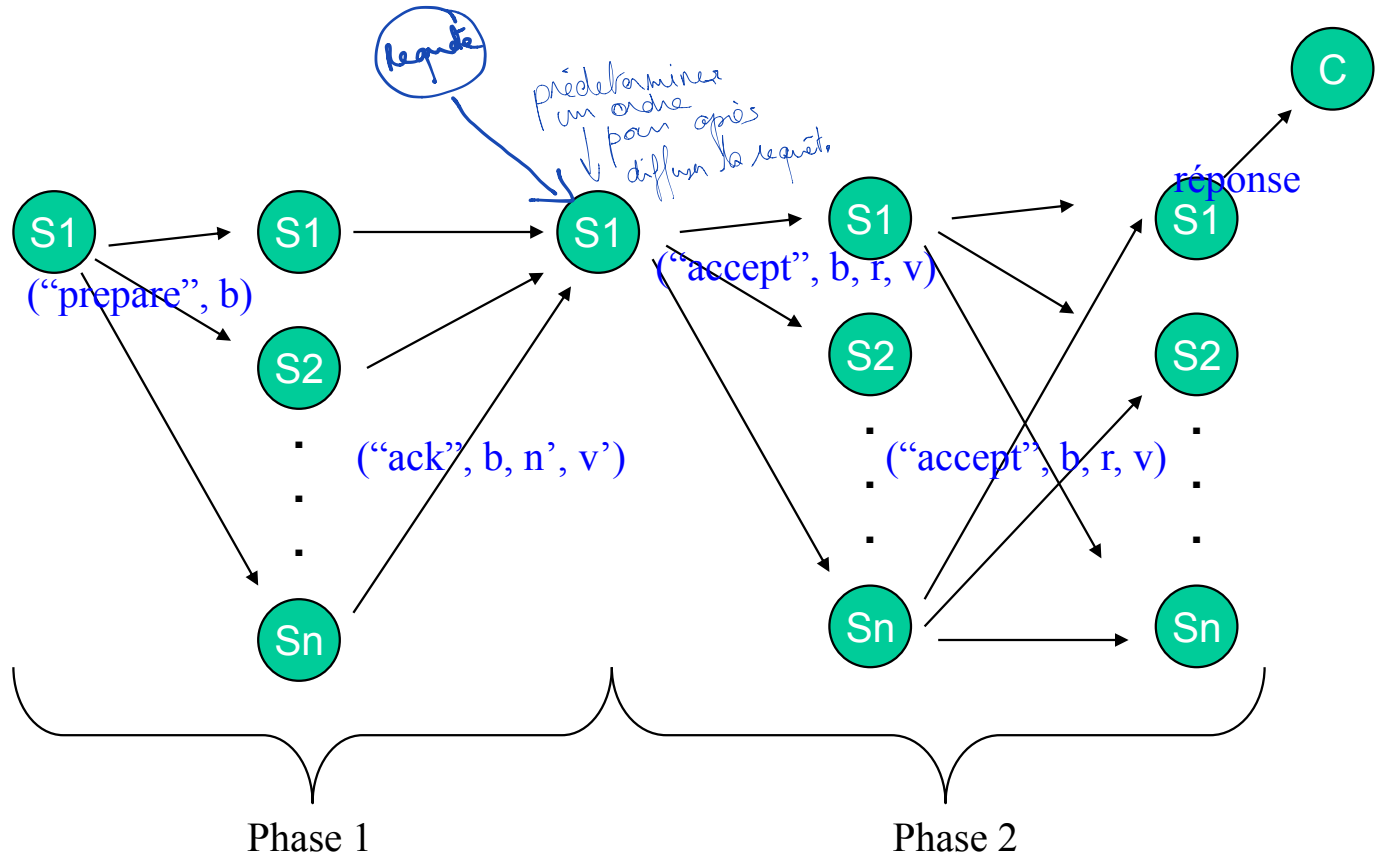
Amélioration: On peut limiter les consensus de Paxos aux opérations

non commutatives et employer des consensus (mais coûteux pour des op. commutatives (additions par ex)).

Paxos – SMR : Exécution sans faute



Optimisation : Phase 1 en amont



SMR basé sur Paxos

Réception (“request”, v) du client

Si (je ne suis pas le leader) **alors** transférer la requête au leader
sinon

/ proposer v avec un nouveau numéro de requête */*

ReqNum \leftarrow ReqNum + 1;

send (“accept”, BallotNum, ReqNum, v) à tous

Réception (“accept”, b, r, v)

/ accepter la proposition pour la requête r */*

AcceptNum[r] \leftarrow b; AcceptVal[r] \leftarrow v

send (“accept”, b, r, v) à tous

Practical Byzantine Fault-Tolerance (PBFT)

- **Practical Byzantine Fault Tolerance and Proactive Recovery.** M. Castro, B. Liskov.
ACM Transaction on Computer Systems, Vol. 20, No. 4, November 2002
- “Byzantine Paxos”

Contexte Byzantin

- Sécurité des données

Techniques

- confidentialité
- intégrité
- authenticité

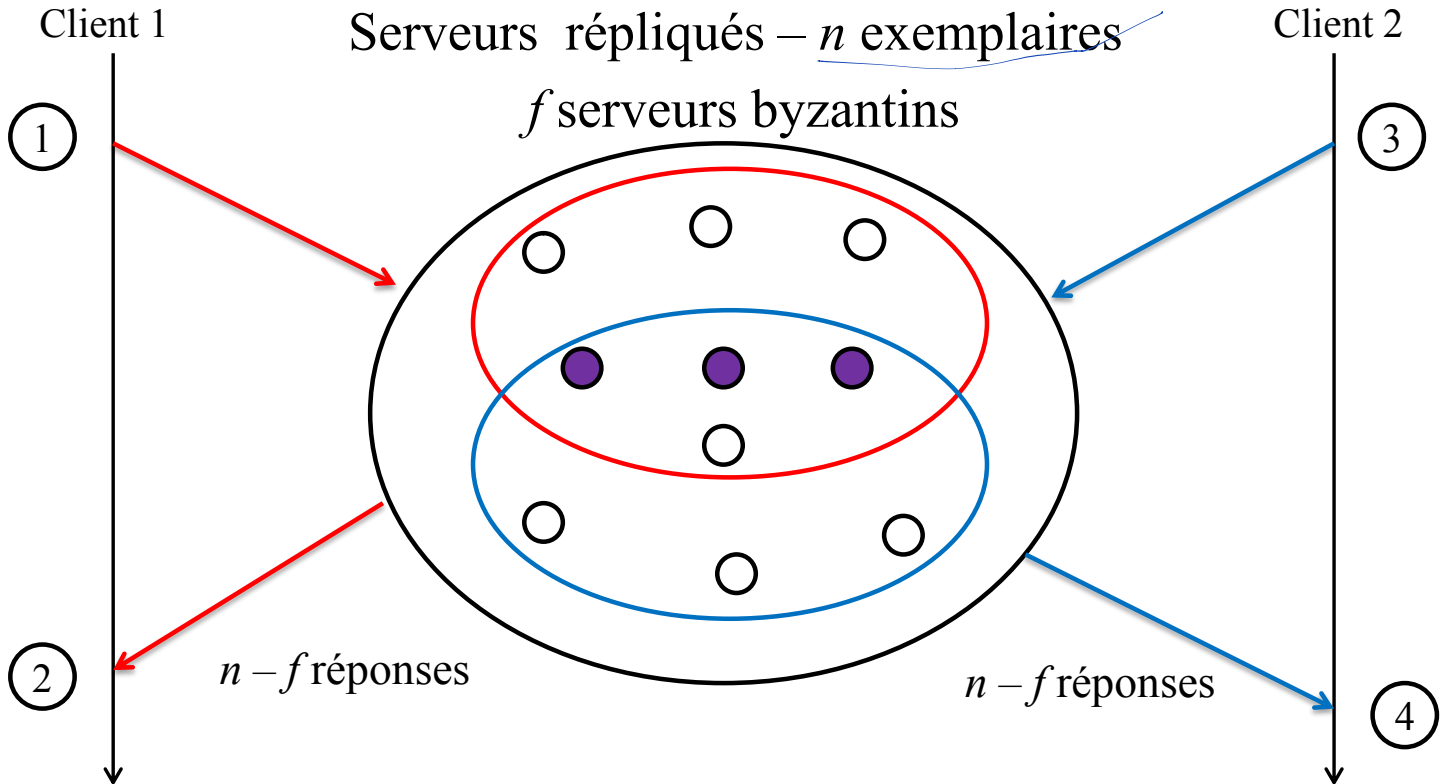
cryptographie

- Sécurité des traitements

- pannes, erreurs quelconques
- attaques malveillantes

Réplication et algorithmes
tolérants les fautes arbitraires

Borne sur le nombre de fautes



Borne sur le nombre de fautes

- Etre sûr d'obtenir une réponse correcte (contiennent toutes les valeurs)
=> intersection doit contenir des non-byzantins



- $|Intersection| = n - 2f$ ($n = 2(n-f) - |Intersection|$)
- Données simples, indifférenciées
 - une majorité pour déterminer la bonne réponse
 - $n - 2f$ (intersection) $\geq f$ (fausses rep.) + $f + 1$ (bonnes rep.)
 - **$n \geq 4f + 1$**
- Données signées, avec un timestamp
 - la bonne réponse est celle de plus haut timestamp
 - $n - 2f$ (intersection) $\geq f$ (fausses rep.) + 1 (bonnes rep.)
 - **$n \geq 3f + 1$**

Paxos Byzantin : Modèle

- n processus: $\{1, \dots, n\}$
- f Byzantine failures, $f < n/3$
 - Pour simplifier $n = 3f + 1$
- Authentication par clé publique (PKI)
- Canaux fiables, faute franche sans recouvrement

Adaptation de Paxos classique pour les Byzantins : Pour assurer la sûreté

1. Le Leader peut choisir une valeur différente que la plus grande acceptée par les $n-f$ processus
 - Solution : Le leader doit prouver qu'il ne ment pas en envoyant les messages "ack" reçus à tous les processus
2. Si aucune valeur n'a été acceptée, le Leader peut envoyer une nouvelle valeur différente à chacun des processus
 - Solution : Avant d'accepter une valeur proposée par le Leader, un noeud vérifie que la valeur a été proposée à assez de processus
 - => une phase supplémentaire (**Phase Propose**)
3. Les agents peuvent envoyer dans la phase 2 des "accept" non valides
 - Solution : attendre $n-f=2f+1$ "accept" messages
4. Les agents peuvent envoyer de valeurs plus grandes dans les "ack"
 - Solution : Ajouter les messages "propose" signés dans les "ack" (ensemble "Proof")

Adaptation de Paxos classique pour les Byzantins : Pour assurer la **vivacité**

1. Le leader peut bloquer l'algorithme (deadlock)

- Solution : Proposer un nouveau Leader quand il ne répond pas
- Utiliser un coordinateur tournant
(BallotNum mod n)+1

2. Des processus byzantins peuvent changer en permanence de leader (livelock)

- Solution : Accepter un nouveau “ballot” seulement si $f+1$ processus proposent un nouveau leader (\Rightarrow diffusion par tous du “prepare”)

$\rightarrow f+1 \sum$ qui est le nouveau leader
qui décident

\hookrightarrow Peut mise à la vivacité de manière active.

(2 proc byzantins peuvent se déclarer perpétuellement leaders).

Byzantine Paxos : Variables

Int	BallotNum,	initialement 0
Int	PropNum,	initialement 0
Int	AcceptNum,	initialement 0
Value $\cup \{\perp\}$	AcceptVal,	initialement \perp
Value $\cup \{\perp\}$	Val,	initialement \perp
Message Set	Proof,	initialement vide

$$\text{Leader} = (\text{BallotNum mod } n) + 1$$

↑ tout le monde
dit initialement que
leader = 1.

Byzantine Paxos - Phase 1: Prepare

Expiration Temporisateur du Leader

BallotNum \leftarrow BallotNum + 1

send (“prepare”, BallotNum) à tous

Réception (“prepare”, b) de $f+1$

si (b < BallotNum) alors return

si (b > BallotNum) alors

BallotNum \leftarrow b

send (“prepare”, BallotNum) à tous

send (“ack”, b, AcceptNum, AcceptVal, Proof) au Leader

tout le monde doit faire cela d'expiration de son temporisateur

Byzantine Paxos Phase 2: Propose

Réception (“ack”, BallotNum, b, val, proof) de $n-f$

$S = \{ \text{“ack” messages reçus signés} \}$

si (tous les vals valides dans $S = \perp$) **alors** Val \leftarrow init value

Sinon Val \leftarrow val valide dans S qui a le plus grand b

send (“propose”, BallotNum, Val, S) à tous

Réception (“propose”, BallotNum, v, S)

si (BallotNum \leq PropNum) **alors** return

si (v ne fait pas parti des valeur valides dans S) **alors** return

PropNum \leftarrow BallotNum

send (“propose”, BallotNum, v, S) à tous

Byzantine Paxos Phase 3: Accept

Réception (“propose”, b , v , S) de $n-f$

si ($b < \text{BallotNum}$) **alors** return

$\text{AcceptNum} \leftarrow b$; $\text{AcceptVal} \leftarrow v$

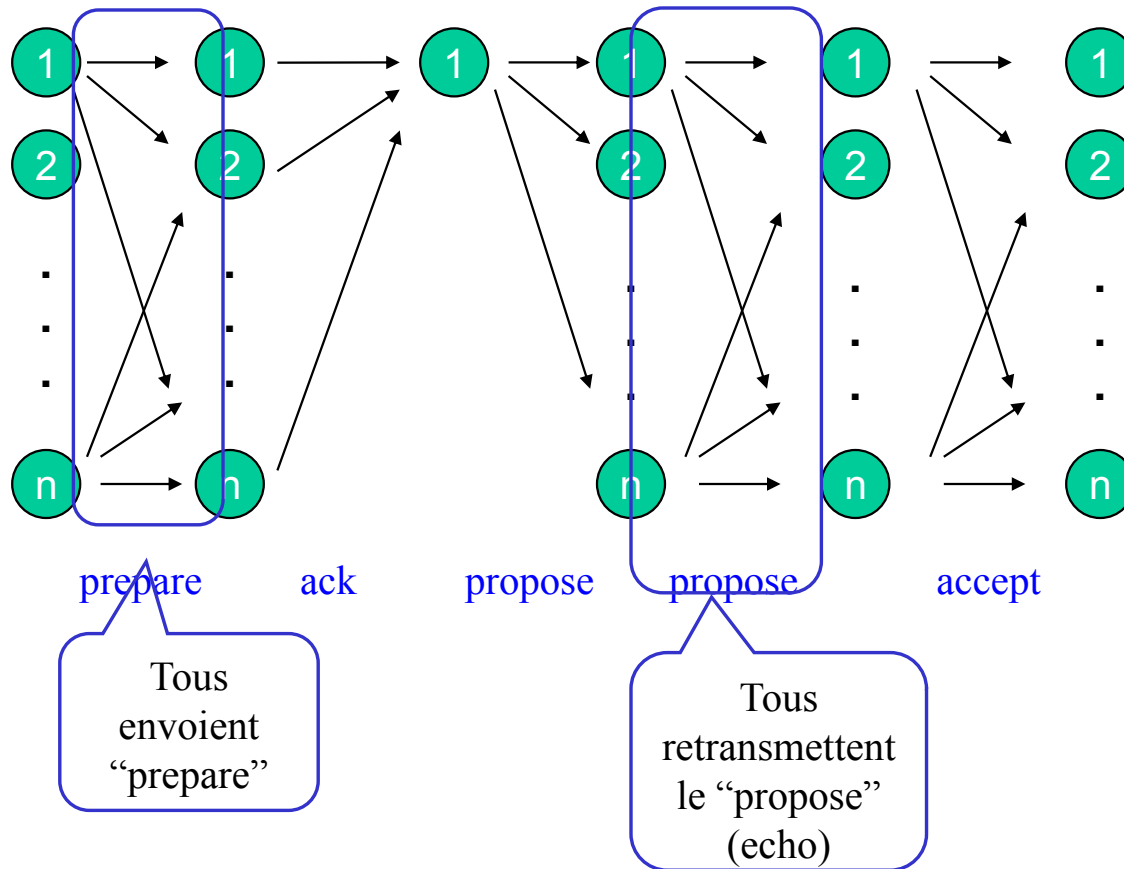
$\text{Proof} \leftarrow$ ensemble de $n-f$ messages “propose” signés

send (“accept”, b , v) à tous

Upon receive (“accept”, b , v) de $n-f$

decider v

Exemple d'exécution



Conclusion Paxos (Classique)

- Consensus tolérant les pertes de messages
- Tolère les fautes franche et transitoire (recovery)
- Performant :
 - Pas de rotation entre les coordinateurs défaillants
 - Utiliser pour maintenir la cohérence entre les copies
 - Google pour le système de verrouillage Chubby (pour maintenir la cohérence de Bibtable)
 - IBM dans système Virtual SAN
 - Microsoft dans Autopilot (Automatic Data Center Management)
 - Dans DHT tolérant les fautes (Scatter – SOSP 2011)