Systèmes Distribués — Aide-Mémoire Complet

Yehor KOROTENKO · 2026 · 9 chapitres · toutes les figures

1. Causalité & Horloges logiques

Modèle de base

Système distribué : n processus $P_1, ..., P_n$ sans mémoire partagée. Communication par messages uniquement. Histoire locale h_i = séquence d'événements : local, send(m), receive(m).

Relation happens-before ($\rightarrow$)

Plus petit ordre strict vérifiant :
1. **Ordre local** — e avant f sur le même processus $\Rightarrow e \rightarrow f$
2. **Communication** — send(m) $\rightarrow$ receive(m)
3. **Transitivité** — $e \rightarrow f \rightarrow g \Rightarrow e \rightarrow g$

Concurrence : $e \parallel f$ ssi $\neg(e \rightarrow f)$ et $\neg(f \rightarrow e)$.

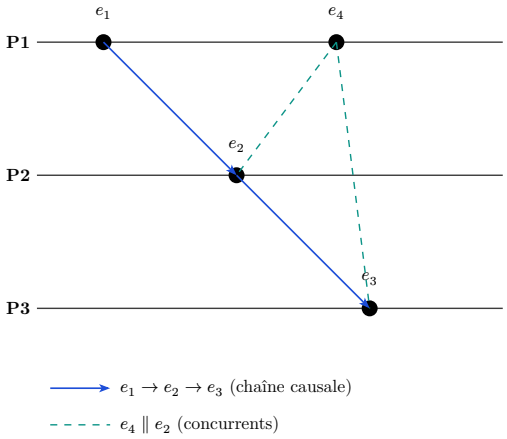


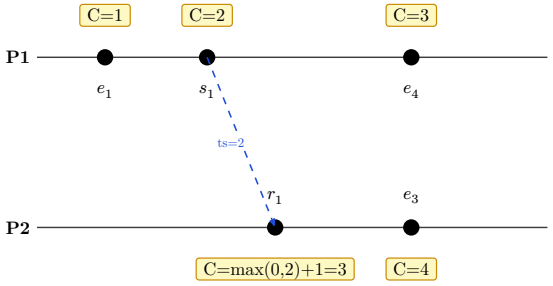
Figure 1: $e_1 \rightarrow e_2 \rightarrow e_3$ (chaîne causale) ; $e_4 \parallel e_2, e_4 \parallel e_3$ (concurrents — pointillé teal)

Horloges de Lamport (scalaires)

Chaque P_i maintient un entier $C_i := 0$.

- **Événement local ou envoi** : $C_i := C_i + 1$; joindre C_i au message.
- **Réception** de (m, t_s) : $C_i := \max(C_i, t_s) + 1$.

$e \rightarrow f \Rightarrow L(e) < L(f)$.
Limitation : $L(e) < L(f) \not\Rightarrow e \rightarrow f$ — ne détecte **pas** la concurrence.



$e \rightarrow f \Rightarrow L(e) < L(f)$ $\quad$ $L(e) < L(f) \not\Rightarrow e \rightarrow f$

Figure 2: Lamport 2 processus. Envoi ts=2 ; réception $C := \max(0, 2) + 1 = 3$. Propriété et limitation en bas.

Horloges vectorielles (Fidge–Mattern 1988)

Chaque P_i maintient $V_i \in \mathbb{N}^n$ initialisé à $\mathbf{0}$.
Ordre componentwise : $V < W$ ssi $\forall k : V[k] \leq W[k]$ et $\exists k : V[k] < W[k]$.

- **Événement local / envoi** : $V_i[i] := V_i[i] + 1$; joindre V_i.
- **Réception** de (m, V_m) : $\forall k : V_i[k] := \max(V_i[k], V_m[k])$, puis $V_i[i] + +$.

Équivalence exacte : $e \rightarrow f \leftrightarrow V(e) < V(f)$.
$e \parallel f \leftrightarrow V(e)$ et $V(f)$ incomparables.
Coût : vecteur taille n par message et par processus.

Lamport = extension linéaire de $\rightarrow$ (utile pour mutex). Vectoriel = caractérisation exacte (utile pour détection de concurrence, snapshots).

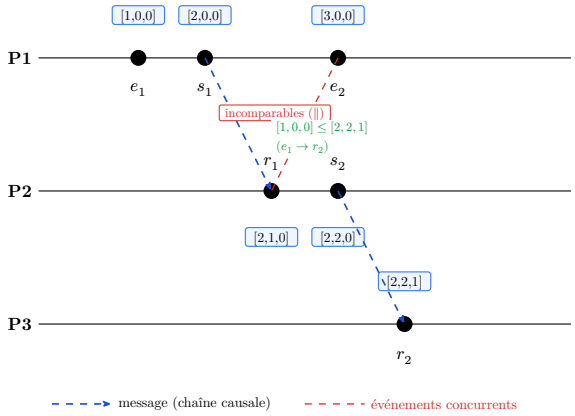


Figure 3: Vectoriel 3 processus. Rouge = incomparables (concurrent). Vert bas = $\leq$ (causal vérifié).

2. Diffusion

Inondation (Flooding)

Initiateur : envoie à tous ses voisins.
Nœud, première réception : retransmettre à tous les voisins sauf l'émetteur.
Nœud, réceptions suivantes : ignorer.
Messages : $\leq 2|E|$ (≤ 2 par arête) — Temps : O(diam)

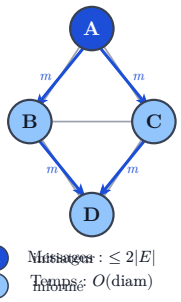
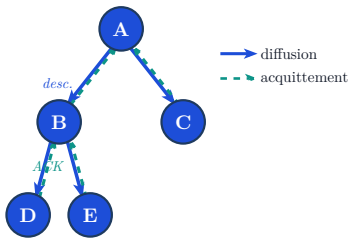


Figure 4: Flooding 4 nœuds. A informe B,C ; B et C informent D. Doublons ignorés.

Diffusion sur arbre couvrant

Hypothèse : arbre couvrant connu. Initiateur $\rightarrow$ fils ; chaque nœud $\rightarrow$ ses fils.
Messages : exactement $N - 1$ (**optimal**) — Temps : O(profondeur)

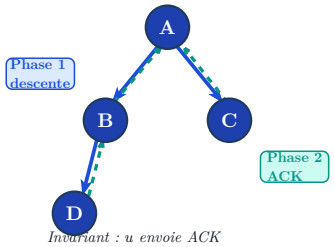


$N - 1 = 4$ messages (descente)
$2(N - 1) = 8$ messages (total avec ACK)

Figure 5: Arbre couvrant. $N - 1 = 4$ messages exactement. Optimal en nombre de messages.

Vague d'acquittements

Phase 1 (descente) : diffusion root $\rightarrow$ feuilles ($N - 1$ msgs).
Phase 2 (remontée des ACK) : feuille $\rightarrow$ ACK immédiatement ; nœud interne $\rightarrow$ ACK quand **tous** ses fils ont acquitté.
Total : $2(N - 1)$ messages. Root sait que tout le monde a reçu.



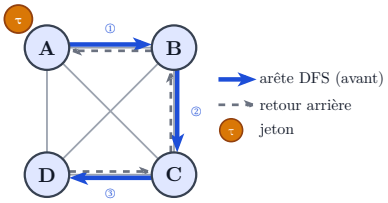
Invariant : u envoie ACK
$\leftrightarrow$
u reçu ET tout sous-arbre de u reçu
Total : $2(N - 1)$ messages

Figure 6: ACK wave. Bleu = diffusion. Teal = acquittements. Root reçoit le dernier ACK = terminaison confirmée.

3. Arbres couvrants & routage

DFS avec jeton

Jeton unique circule en DFS. À chaque nœud : voisin non visité → avancer ; tous visités → backtracker.
Messages : $O(|E|)$ — Temps : $O(|E|)$ — Résultat : **arbre DFS** (séquentiel)

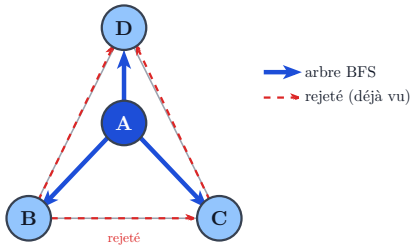


Messages : $2|E|$ Temps : $O(|E|)$ (séquentiel)

Figure 7: Jeton DFS (T amber). Arêtes de l'arbre DFS en couleur. Backtrack = retour au parent.

Flooding parallèle → BFS

Chaque nœud se déclare fils du **premier** émetteur reçu. Suivants : rejetés.
Messages : $O(|E|)$ — Temps : $O(\text{diam})$ — Résultat : **arbre BFS**

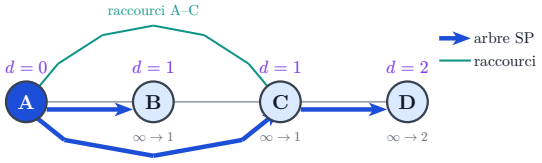


Résultat : arbre BFS (plus courts chemins depuis A)
Temps : $O(\text{diam})$ Messages : $O(|E|)$

Figure 8: BFS par flooding. B,C,D prennent A comme parent. Croix rouge = message rejeté (nœud déjà revendiqué).

Bellman-Ford distribué

Init : $d(\text{src}) = 0$, $d(v) = \infty$ sinon.
Chaque round : diffuser $d(v)$. Sur réception de $d(u)$: si $d(u) + w(u,v) < d(v) \rightarrow$ mettre à jour.
Convergé si aucune maj depuis 1 round. Rounds $\leq N-1$. Msgs $O(N|E|)$.



Messages : $O(N \cdot |E|)$ Convergence : $O(N)$ tours

Figure 9: Bellman-Ford A–B–C–D avec raccourci A–C. Convergence en 3 rounds. Arbre SP résultant.

4. Exclusion mutuelle distribuée

Sûreté : ≤ 1 processus en SC. **Vivacité** : toute demande accordée. **Équité** : FIFO par horodatage.

Solution centralisée

P_i : REQ(i) → GRANT → SC → RELEASE.
Coordinateur P_c : sur REQ : libre → GRANT ; sinon → file FIFO. Sur RELEASE : dépiler → GRANT.
3 msgs/SC — Point unique de défaillance

Lamport 1978 — entièrement distribué

1. Demande : $C_i \leftarrow C_i + 1$, diffuser DEMANDE(C_i, i), ajouter (C_i, i) à Q_i.
2. Réception DEMANDE(t, j) : $C_i := \max + 1$, ajouter (t, j) à Q_i, répondre ACK.
3. Entrée SC : (C_i, i) est $\underset{\text{lex}}{<}$-min de Q_i **ET** ACK de tous les P_j avec ts $> C_i$.
4. Sortie SC : diffuser SORTIE(C_i, i) → tous retirent (C_i, i) de leurs files.

$3(N-1)$ msgs/SC : DEMANDE + ACK + SORTIE, chacun $\times (N-1)$.
Sûreté : si P_i et P_j en SC avec $(C_i, i) \underset{\text{lex}}{<} (C_j, j)$, alors P_j aurait dû attendre l'ACK de P_i — contradiction.

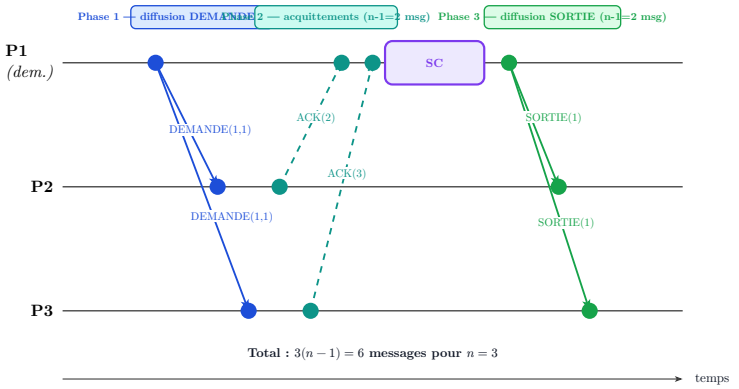


Figure 10: Lamport n=3. Jaune=DEMANDE, teal=ACK, bleu=SORTIE. Chaque phase $n-1=2$ msgs. Total $3(n-1)=6$.

Ricart-Agrawala 1981

Sur réception REQ(t_j, j) par P_i :
- $(t_j, j) \underset{\text{lex}}{<} (t_i, i)$ et pas en SC → répondre **immédiatement**
- En SC ou priorité plus haute → **différer** la réponse

Sortie SC : envoyer toutes les réponses différées. **$2(N-1)$ msgs/SC.**

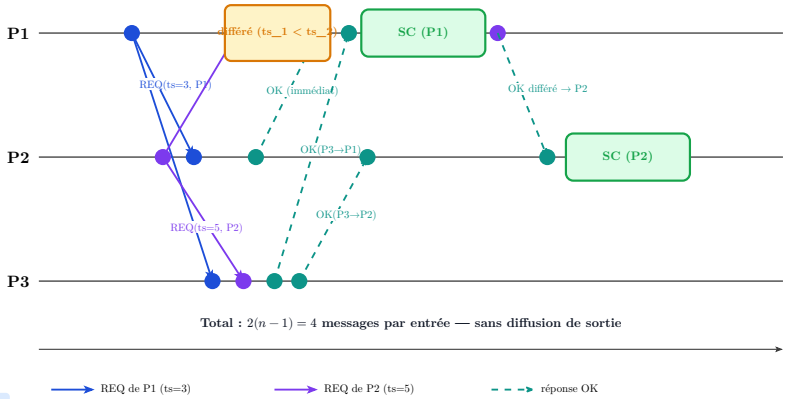


Figure 11: RA. P1(ts=3) prioritaire sur P2(ts=5). P2 répond imméd. P1 diffère → envoie en sortant.

RA avec jeton explicite

Jeton = jeton[1..n] (nb d'entrées passées).
Demande : broadcaster DEMANDE(k_i).
Détenteur reçoit DEMANDE(k_j) : si $k_j >$ jeton$[j]$ → transmettre jeton.
Sortie : jeton$[i]$ ++. Libération implicite. **$\leq 2(N-1)$ msgs/SC.**

Anneau à jeton

Jeton circule $P_1 \rightarrow \ldots \rightarrow P_n \rightarrow P_1$. Détenteur : SC si désiré, puis passer.
Meilleur : 0 msgs — Pire : $N-1$ msgs — Vulnérable à la perte du jeton

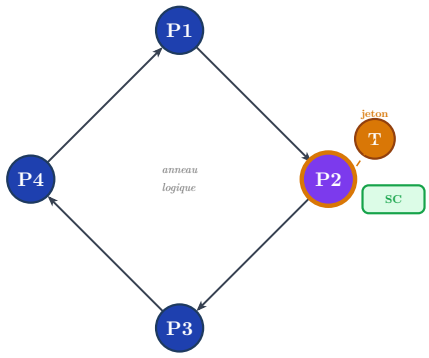


Figure 12: Token Ring. T amber = jeton. P2 retient pour entrer en SC (bleu ciel). Passage sinon.

Algorithme	Msgs/SC	Robustesse
Centralisé	3	coord
Lamport	$3(N-1)$	—
Ricart-A.	$2(N-1)$	—
RA+jeton	$\leq 2(N-1)$	jeton
Token Ring	0 à $N-1$	jeton

5. Tolérance aux pannes

Hiérarchie : Crash ⊂ Omission ⊂ Byzantin

- **Crash** : processus s'arrête définitivement. Détectable par timeout. $\geq 2f+1$ nœuds pour f pannes.
- **Omission** : perd certains msgs (send ou receive). Vivant mais peu fiable.
- **Byzantin** : arbitraire, peut mentir, envoyer des valeurs contradictoires. $\geq 3f+1$ nœuds.

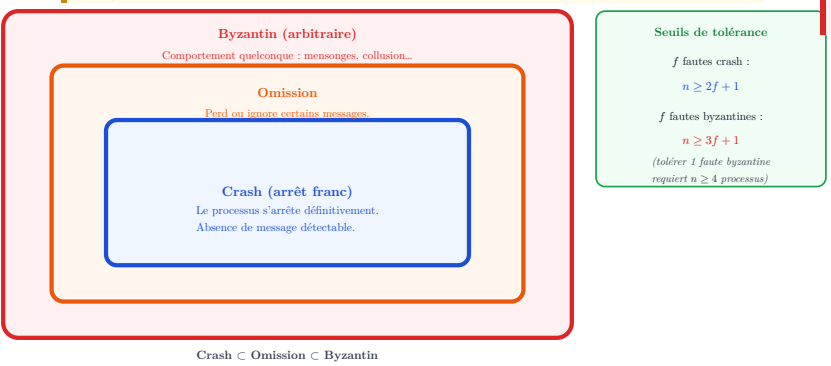


Figure 13: Hiérarchie : Crash (🐭) ⊂ Omission (🐭) ⊂ Byzantin (🐮). Seuils en bas.

Réplication : k répliques → masque $k-1$ pannes crash. Resynchronisation à la reprise.

6. Élection de chef

Propriétés : Terminaison + Accord (1 seul élu) + Validité (élu = id max des corrects).
Hypothèse : identifiants uniques. Topologie : anneau.

Chang-Roberts (anneau unidirectionnel, 1979)

Extinction sélective : chaque processus envoie son id dans le sens de l'anneau. Seul le max fait le tour complet.

Sur réception de v par P_i :
- $v <$ mon_id → transmettre v
- $v >$ mon_id → éliminer (ignorer) v
- $v =$ mon_id → élu ! — diffuser LEADER(v)

Correction : le max M n'est jamais éliminé → fait le tour complet → P_M élu.
Pire cas : $O(n^2)$ (IDs décroissants : $\sum_{\{k=1\}}^{\{n\}} k = \frac{n(n+1)}{2}$).
Cas moyen : $O(n \log n)$ (permutation aléatoire des IDs).

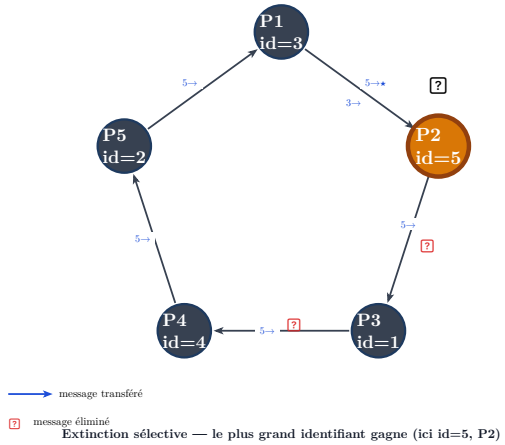


Figure 14: Chang-Roberts 5 processus. id=5 (P2) transite partout → retour = élu (?). Croix = élimination.

Peterson (anneau bidirectionnel, 1982)

Phases alternant direction. Chaque actif P_i :
1. Envoyer a_i à droite ; recevoir a_{left} de gauche.
2. Relayer a_{left} à droite ; recevoir a_{ll} (voisin du voisin gauche).
3. Si $a_{\text{left}} > \max(a_i, a_{\text{ll}})$: $a_i \leftarrow a_{\text{left}}$ (survit). Sinon : éliminé (relaie seulement).

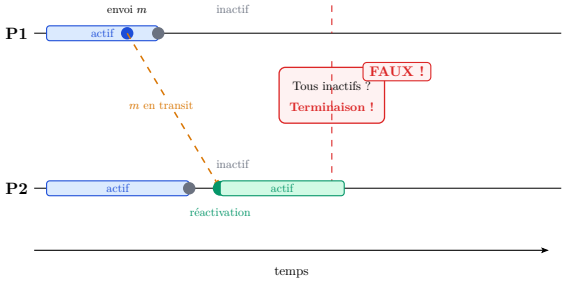
Terminaison : 1 actif → élu, diffuse LEADER.

Lemme : deux actifs adjacents ne peuvent pas tous deux survivre → $\geq \frac{1}{2}$ éliminés par phase.
$O(\log n)$ phases × $O(n)$ msgs = $\boldsymbol{O(n \log n)}$ total.

Algo	Topologie	Pire cas	Élu
Chang-Roberts	Unidirectionnel	$O(n^2)$	id max
Peterson	Bidirectionnel	$O(n \log n)$	id max

7. Détection de terminaison

Piège : "tous les processus sont passifs" ↛ terminaison. Des messages en transit peuvent réactiver un passif.



Attention : processus inactif ≠ terminé — les messages en transit comptent !

Figure 15: Manager voit P1,P2 passifs, mais m en transit → faux positif. P2 se réactive.

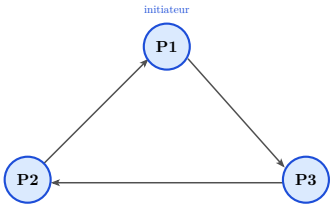
Dijkstra-Safra — anneau synchrone

Jeton blanc/noir circule sur l'anneau.

P_i reçoit le jeton :
- A envoyé un msg depuis le dernier passage : noircir le jeton.
- Sinon : transmettre inchangé.

Initiateur reçoit jeton blanc et est lui-même passif → **terminaison**.
Jeton noir → réinitialiser blanc, recommencer.

Tour blanc = aucun msg envoyé "vers l'avant" → pas de réactivation possible.



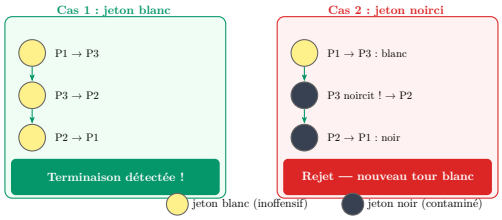


Figure 16: D-S. Gauche vert = tour blanc = terminaison. Droite rouge = jeton noirci = recommencer.

Safra avec compteurs (asynchrone)

$\text{me}_i = \text{msgs\_envoyés}_i - \text{msgs\_reçus}_i$. $\sum_i \text{me}_i = 0$ ssi aucun msg en transit.
Jeton = (couleur, q). Sur réception : si noir → teinter jeton ; transmettre $(c, q + \text{me}_i)$.
Terminaison ssi jeton blanc ET P_0 blanc/passif ET $q + \text{me}_0 = 0$.

Mattern — crédit (topologie quelconque)

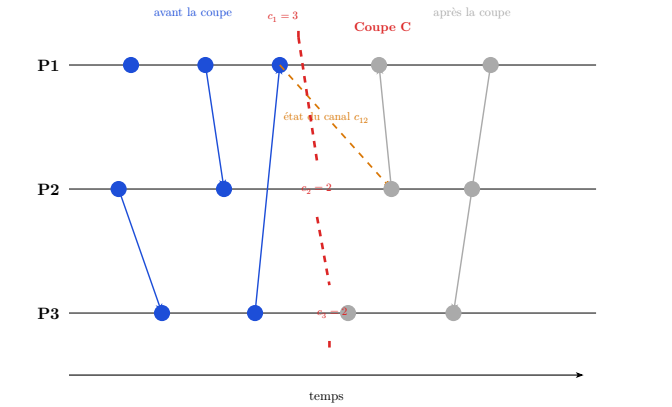
Invariant : $\sum \text{crédits} = 1$.
Leader part avec crédit=1. Envoi : piggybacker crédit/2, garder crédit/2.
Récepteur : crédit += reçu. Passif : retourner crédit au leader. Leader : ret = 1 → terminaison.

D-S = anneau synchrone. Safra compteurs = anneau asynchrone. Mattern = toute topologie.

8. Snapshot global (Cliché)

Coupe cohérente

Coupe C : préfixe h_i^c de l'histoire de chaque P_i (photo locale $s_i^\star$).
Cohérente : $e \in C$ et $e' \to e \Rightarrow e' \in C$ (pas de "message du futur").
État de canal $c_{ij}^\star$ = msgs envoyés par P_i dans le passé de C mais non encore reçus par P_j.



Coupe cohérente : aucun message envoyé après la coupe n'est reçu avant. Le message en transit (en tirets ambrés) constitue l'**état du canal**.

Figure 17: Coupe (pointillé rouge). Bleu = pre-shot, gris = post-shot. Amber = en transit (état canal). Rouge = interdit.

Chandy-Lamport — canaux FIFO

Initiateur P_i : enregistrer $s_i^\star$; envoyer MARKER sur chaque canal sortant.
P_j reçoit MARKER, 1ère fois (taken $= F$) : enregistrer $s_j^\star$, $c_{ij}^\star := \emptyset$ (FIFO garantit), envoyer MARKER sur tous sortants.
P_j reçoit MARKER, déjà taken : $c_{ij}^\star :=$ msgs reçus depuis $s_j^\star$ jusqu'à ce MARKER.

$(s_i^\star, c_{ij}^\star)$ = coupe cohérente + significative. Sans horloge globale.

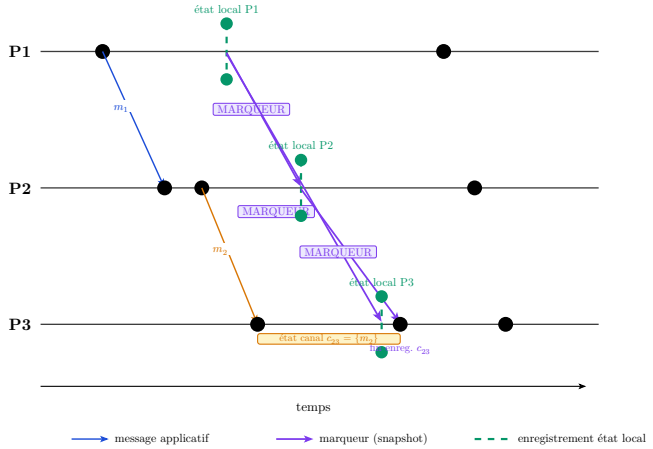


Figure 18: CL 3 processus. ? = MARKER. Vert = $s^\star$ enregistré. Canal 1→2 = $\emptyset$ (FIFO).

Lai-Yang — canaux non-FIFO

1 bit tag par message. P_i se prend en photo : taken$_i := T$, msgs futurs tag $= T$.
P_j reçoit msg tag $= T$ et taken$_j = F$: enregistrer $s_j^\star$, taken$_j := T$.
$c_{ij}^\star$ = msgs tag $= F$ reçus par P_j après taken$_j := T$.
Coût : 1 bit/msg, 0 message de contrôle.

Critère	Chandy-Lamport	Lai-Yang
Canaux	FIFO	Quelconques
Overhead	1 MARKER/canal	1 bit/msg
Msgs ctrl	$n(n-1)$	0

9. Consensus

Accord : deux corrects ne décident pas des valeurs différentes. **Validité** : valeur décidée = valeur initiale d'un correct. **Terminaison** : tout correct décide en temps fini.

Impossibilité FLP (1985)

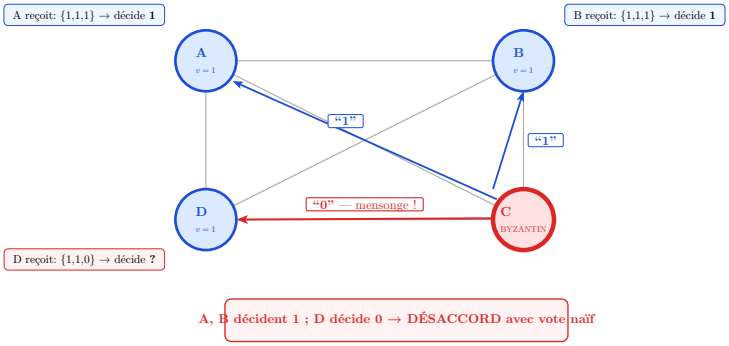
Théorème FLP : en système **asynchrone** avec ≥ 1 panne crash possible, aucun algorithme déterministe ne garantit Accord + Validité + Terminaison simultanément.
Preuve (sketch) : argument de configuration bivalente. On ne peut pas distinguer "processus lent" de "processus crashé". L'adversaire retarde les messages pour maintenir l'ambivalence indéfiniment.
Contournement : Paxos/Raft = synchronie partielle + leader. PBFT = $3f+1$ répliques. Randomisation = termination en probabilité 1.

Généraux byzantins (LSP)

Borne inférieure : avec f byzantins, consensus impossible si $n < 3f+1$.

Exemple $n = 4, f = 1$: byzantin C envoie 1 à A, B mais 0 à D.
- A : vote 1. B : vote 1. D : vote 0. Désaccord.

Solution : avec $n \geq 3f+1$, algo LSP en $f+1$ tours d'échanges de vecteurs.



Condition nécessaire et suffisante : $n \geq 3f+1$ (ici $n = 4, f = 1 : 4 \geq 3\times1+1=4$ ✓, mais protocole en 1 tour insuffisant)

Figure 19: Byzantin C envoie 1 à A/B mais 0 à D. Désaccord banner rouge. $n \geq 3f+1$ nécessaire.

Flood-Set — synchrone, pannes crash

ω_i = ensemble de valeurs connues, init = {propre valeur}.

Répéter $f+1$ fois :
1. Broadcaster ω_i à tous.
2. $\omega_i := \omega_i \cup$ toutes valeurs reçues.

Décision : si $|\omega_i| = 1 \rightarrow$ décider ω_i ; sinon $\rightarrow$ décider min(ω_i).

Pourquoi $f+1$ phases ? Parmi $f+1$ phases, au moins une est **sans panne**. Durant cette phase, tous les ω_i s'égalisent (chaque correct reçoit les mêmes valeurs).
Lemme clé : une fois ω_i égaux, ils restent égaux (union d'ensembles identiques = même ensemble).
Optimalité : $f+1$ phases nécessaires (l'adversaire crashe 1 processus par phase sur f phases).
Complexité : $O((f+1)\cdot n^2)$ msgs, $f+1$ rounds.

FLP = impossibilité en asynchrone même avec 1 crash. Flood-Set = possible en synchrone avec $f+1$ rounds. La synchronie est le facteur déterminant. Avec des byzantins, LSP avec $n \geq 3f+1$ nœuds.

Abréviations : SC = section critique · FIFO = First In First Out · $|E|$ = arêtes · $\frac{N}{n}$ = processus · f = pannes tolérées · diam = diamètre · SP = plus court chemin · LSP = Lamport-Shostak-Pease · RA = Ricart-Agrawala · FLP = Fischer-Lynch-Paterson · CL = Chandy-Lamport · LY = Lai-Yang · D-S = Dijkstra-Safra