SDR

Consensus

Raft

Olivier Lemer

↓ Le principe

Suppositions

Pas plus de ⌈N/2⌉-1 pannes simultanées.

Pannes récupérables, sans amnésie.

Pas de duplication ou réordonnancement de messages.
Pertes possibles.

Réseau partiellement synchrone.

Réplication de machine d'état

Pensé pour répliquer une machine d'état sur chaque processus.

Un log répliqué

Chaque processus maintient un log des opérations à appliquer.

Un élu

Responsable de synchroniser ces logs répliqués.

Deux responsabilités distinctes

Synchronisation des logs

Un consensus doit exister sur le contenu du log.

Élection d'un leader

Un élu mort doit être remplacé, et toute incohérence corrigée.

Gestion des logs répliqués

Un élu responsable du log

Log vs Machine d'état

Deux concepts distincts

Le log d'opérations

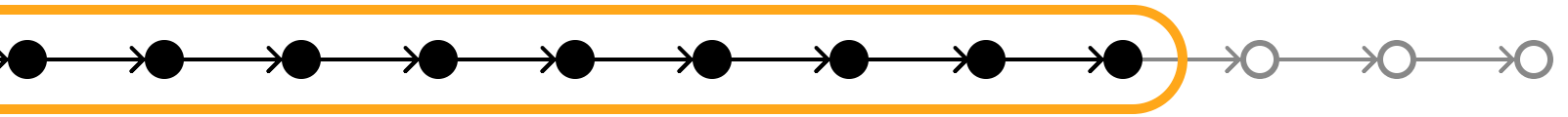
Ce qui va être (ou a été) appliqué à la machine d'état

La machine d'état elle-même

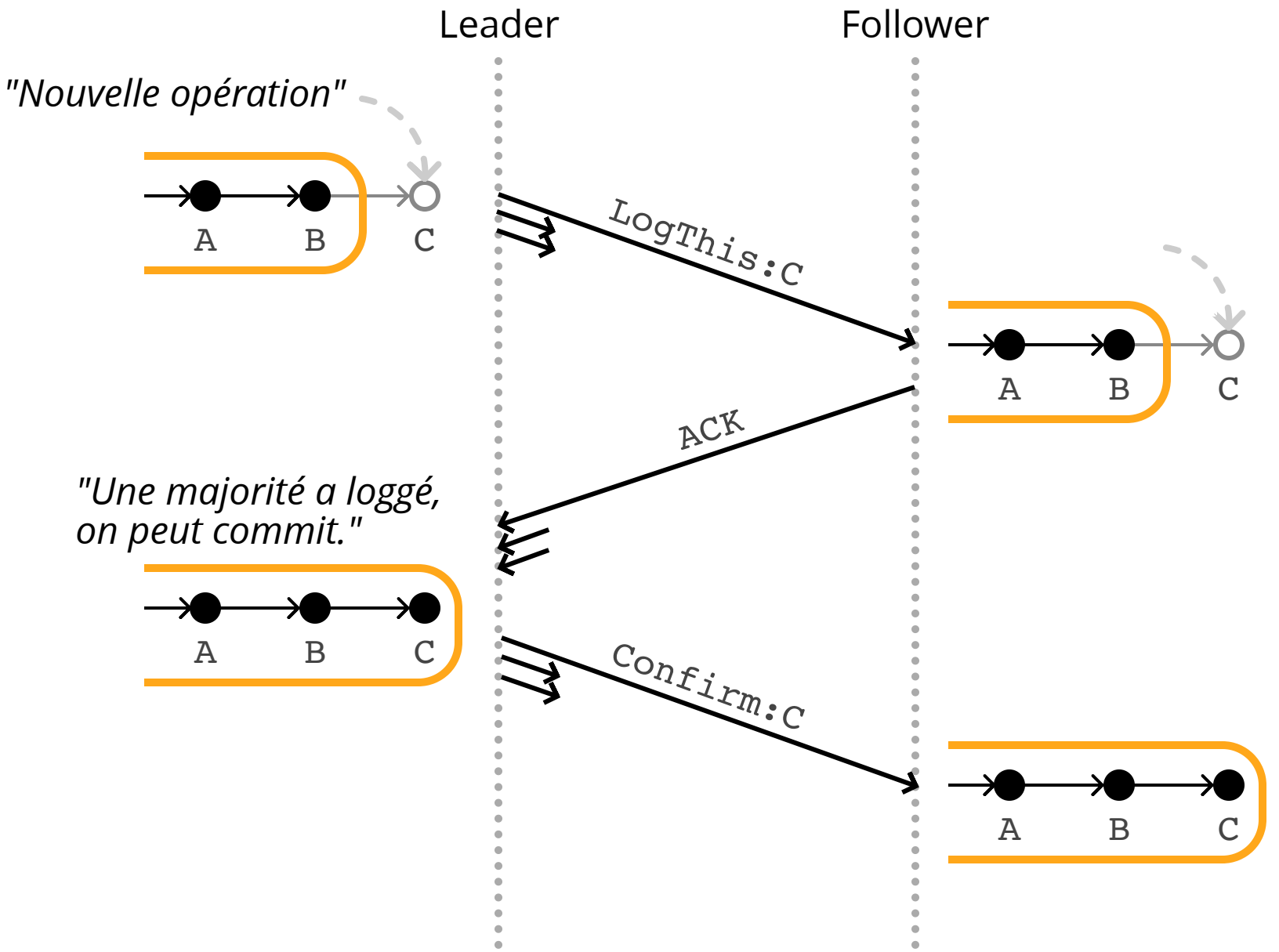
L'état actuel, après application de tous les logs committed.

Chaque log peut donc être

- ● Committed *(appliqué)*
- ○ Pending *(pas encore appliqué)*



Le leader qui synchronise



Le leader qui synchronise

Le leader reçoit les requêtes d'opérations, et assure leur application chez tous.

En tant que leader,

À la réception d'une requête

- Je la log localement comme *pending*
- Je broadcast un message **LogThis** *(AppendEntry)*
- J'attends le ACK d'au moins ⌈N/2⌉ +1 voisins
- J'applique l'opération à la machine d'état
- Je marque l'opération comme *committed*
- J'en informe tout le monde par un **Confirm**

Pourquoi pas tous ?
On y reviendra

En cas de panne ici ?
On y reviendra

Pour tout message, penser un envoi RR ; je réessaie jusqu'à confirmation de réception.

En tant que follower,

À la réception d'un LogThis

- Je log localement comme *pending*

À la réception d'un Confirm

- Je marque l'opération *committed*
- Je l'applique à la machine d'état

Élection

Un algorithme spécialisé

Panne du leader

Une panne de leader doit déclencher une nouvelle élection.

Comment faire ?

Détection de pannes :

- Le leader envoie un heartbeat régulier à tous les followers.
- Un follower déclenche une élection s'il ne reçoit rien avant un timeout.

Problème :

Comment déterminer le leader ?

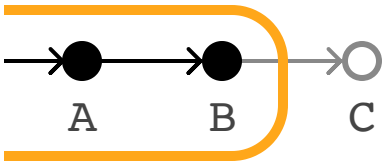
Panne du leader

Problème :

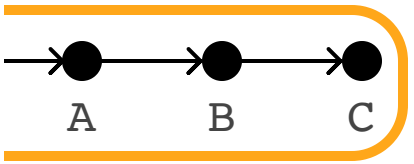
Comment déterminer le leader ?

Deux processus peuvent être en désaccord sur l'état de leur machine.

Follower 1



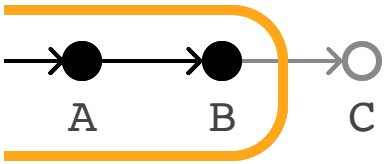
→ Follower 2



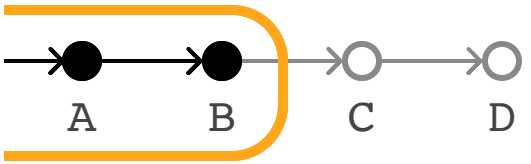
Celui ayant le log *committed* le plus récent doit gagner.

Et en cas d'égalité sur les logs *committed* ?

Follower 1



→ Follower 2



Celui ayant le log *pending* le plus récent doit gagner.

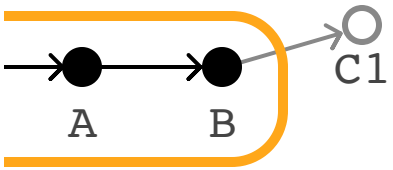
Panne du leader

Problème :

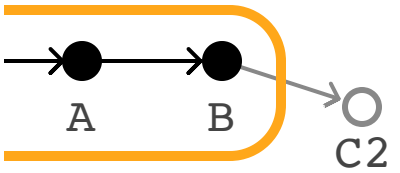
Comment déterminer le leader ?

Et en cas de désaccord sur les logs *pending* ?

Follower 1



Follower 2

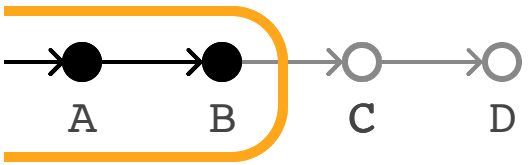


Impossible : le leader envoie les messages dans le même ordre à tous.*

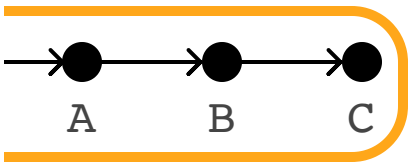
*En réalité un peu plus compliqué que ça en cas de gros ralentissements du réseau ; ignoré ici par soucis de simplicité.

Et en cas de log *committed* plus ancien, mais *pending* plus récent ?

Follower 1



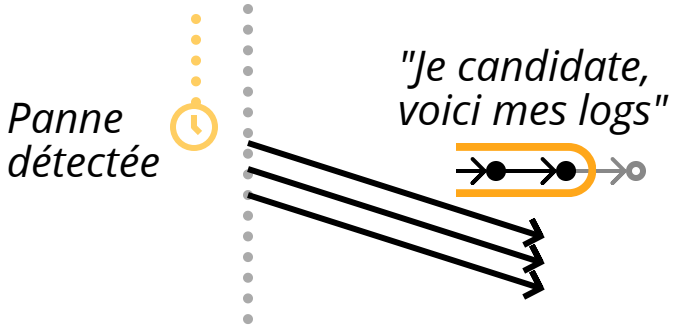
Follower 2



Impossible : le leader envoie les messages dans le même ordre à tous.

Un algorithme spécialisé

Détection de panne du leader



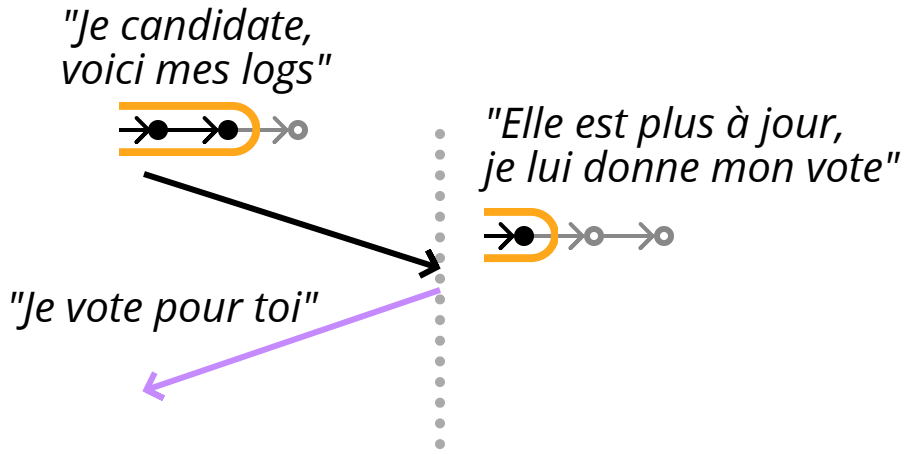
Réception de votes

Si je reçois une majorité de votes,
Je me considère élu leader !

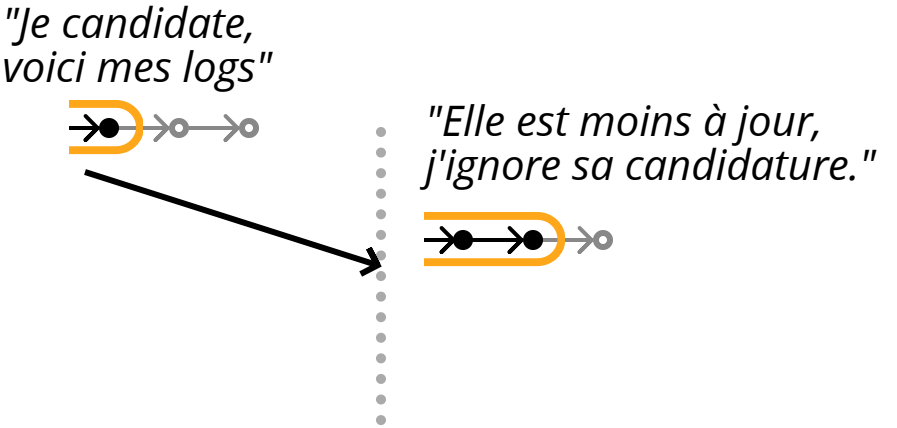
Problème ?

Réception d'une première candidature

...étant plus (ou autant) à jour que moi



...étant moins à jour que moi

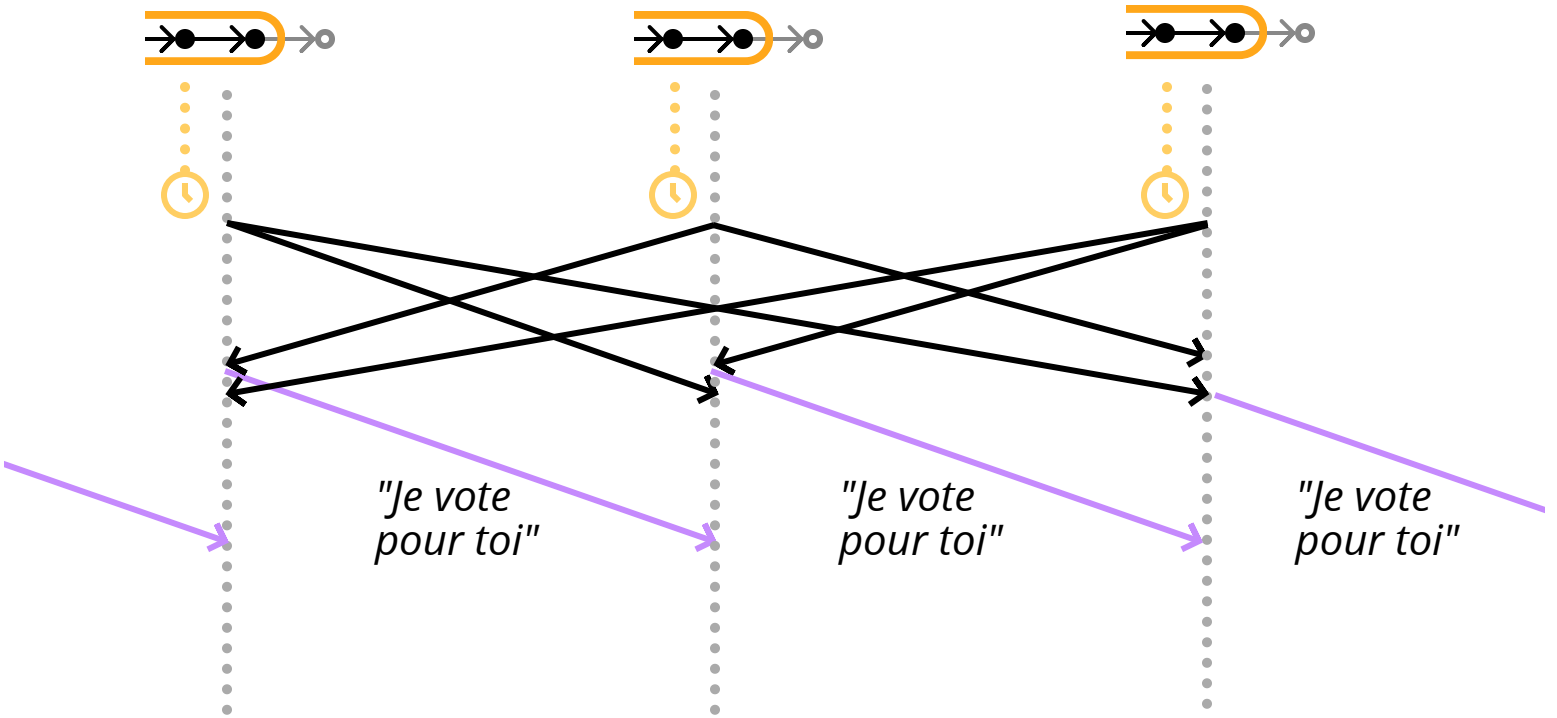


Remarque : je n'ai qu'un seul vote à donner par élection. Premier arrivé, premier servi.

Premier arrivé, premier servi ?

Un problème...

Et si tout le monde a les mêmes logs, et détecte la panne en même temps ?

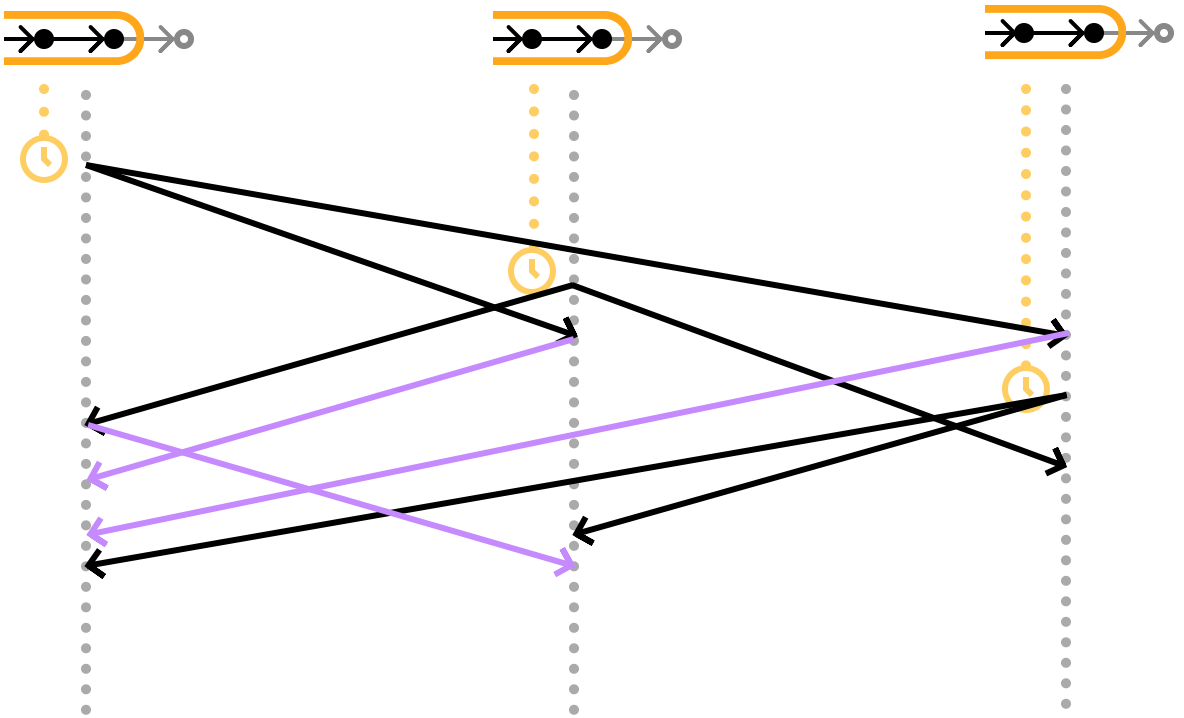


Personne n'aura de majorité absolue...

Premier arrivé, premier servi ?

Solution : l'aléatoire !

Les timeouts sont d'une durée aléatoire.



Celui qui timeout en premier a plus de chances d'être élu !

Problème ?

Premier arrivé, premier servi ?

Problème : l'aléatoire !

Et si l'aléatoire ne suffit pas ?

Alors

Personne ne se considèrera élu, donc

Personne n'enverra de heartbeat, donc

Tout le monde va à nouveau timeout.

Autrement dit, les élections s'enchaineront jusqu'à ce qu'un élu soit trouvé.

Premier arrivé, premier servi

Résumé

Le leader envoie un heartbeat régulier à tous les followers.

Chaque processus a une durée de *timeout d'élection*.

En tant que follower,

Si aucun heartbeat pendant ma durée de timeout

- Je deviens candidat aux prochaines élections
- J'envoie ma candidature à tout le monde, avec mes logs récents

Si je reçois une candidature

- Si j'ai des logs pending (à défaut, committed) plus récents que lui, je l'ignore
- Sinon, si c'est la première candidature valide pour ce mandat
  - Je l'accepte et lui envoie mon vote

Si je suis candidat et reçois une majorité de votes

- Je deviens le leader

Désaccord avec l'élu ?

Un follower peut-il avoir un log *committed* que l'élu n'a pas ?

Si oui,

Un message `Confirm` aurait été reçu pour ce log, donc

Le leader aurait reçu une majorité de `ACK` pour ce log, donc

Une majorité de processus possèderaient ce log, donc

Une majorité de processus n'auraient pas voté pour cet élu.

Donc impossible

Un follower peut-il avoir un log *pending* que l'élu n'a pas ?

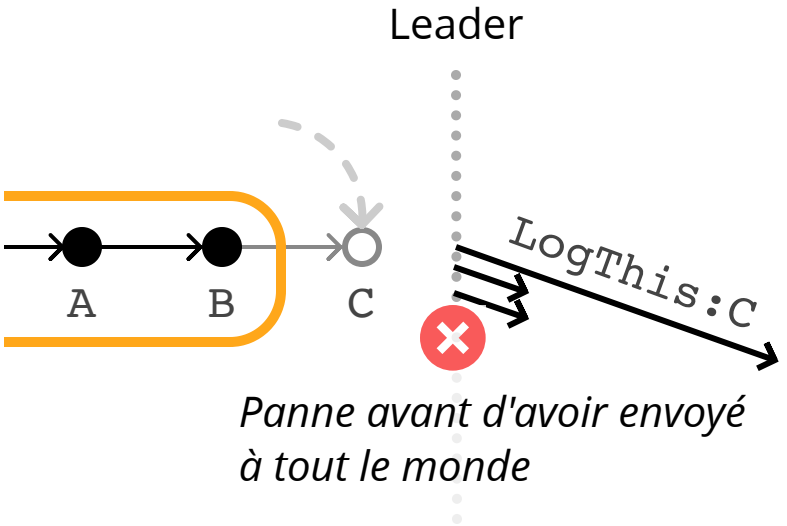
Oui, en cas de panne...



Pannes du leader

Quand tout se complique (?)

Panne durant un `LogThis`



Panne avant d'avoir envoyé à tout le monde

Problème ?

Ce log ne sera *committed* chez personne.

Pas de désaccord sur l'état de la machine.

Mais, désaccord possible sur les *pending*.

Que doit faire le prochain leader ?

S'il avait reçu `LogThis:C`,

Il fait ajouter `C` à tous les followers

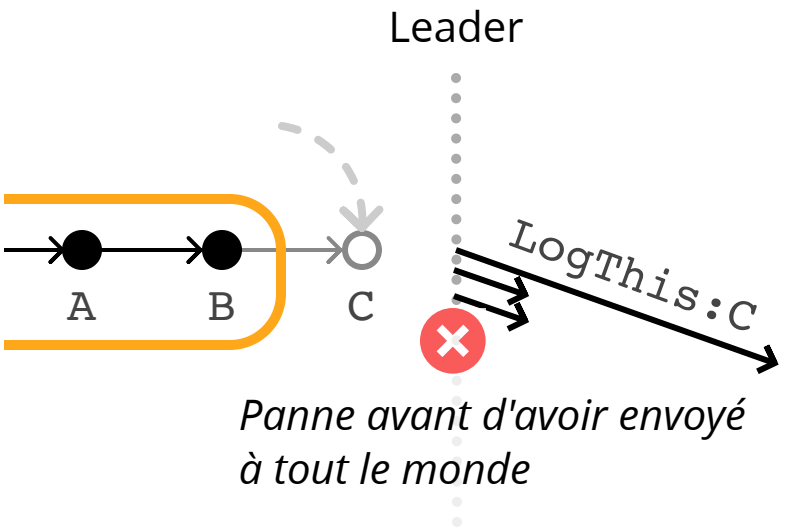
Sinon

*Il fait **retirer** `C` à tous les followers* ⟵ `C` sera perdu, mais c'est OK.

Pourquoi ?

Le leader est tombé en panne pendant le traitement de `C`.
L'ignorer n'est donc pas incohérent.

Panne durant un `LogThis`



Panne avant d'avoir envoyé à tout le monde

Problème ?

Ce log ne sera *committed* chez personne.

Pas de désaccord sur l'état de la machine.

Mais, désaccord possible sur les *pending*.

Que doit faire le prochain leader ?

Le leader demandera à tous les followers d'écraser leurs logs par les siens.

Risque d'écraser un log *committed ?*

Non

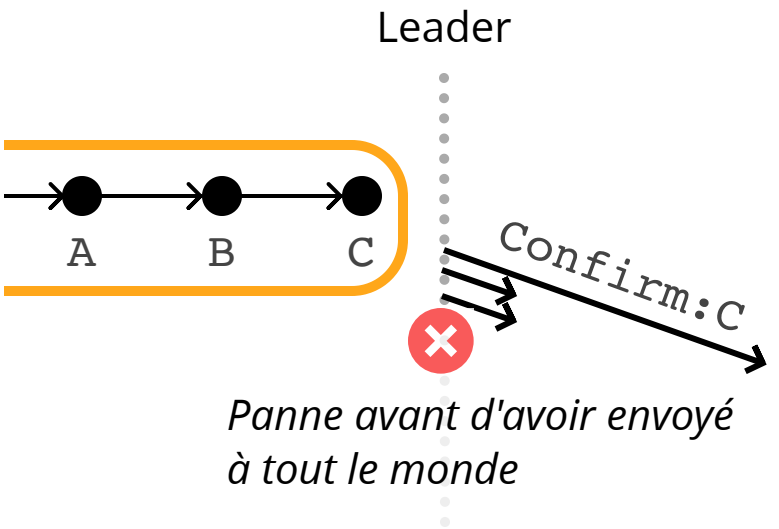
On a vu qu'aucun follower ne peut avoir un log committed que le leader n'a pas.

Risque d'écraser un log *pending ?*

Oui

Si une minorité avait reçu ce log, il est possible qu'aucune d'entre elles n'ait été élue. Dans ce cas, ce log sera perdu. C'est OK.

Panne durant un `Confirm`



Problème ?

Une majorité aura reçu `LogThis:C`,

Donc l'élu aura `C` parmi ses logs,

Mais peut-être seulement en *pending*.

Si `C` est *committed* chez le nouveau leader

Il demande à tous les followers de le commit, si pas déjà fait.

Si `C` est *pending* chez le nouveau leader

Il demande à tous les followers de le logger, si pas déjà fait.

Puis enverra `Confirm:C` une fois qu'une majorité lui aura répondu.

Résumé

En résumé

Raft est un consensus spécialisé pour la réplication de machine d'état.

Le mécanisme d'**élection** assure

- Un seul leader à la fois
- La gestion des absences de majorité *(via timeouts aléatoires)*
- L'élection d'un processus à jour avec la majorité des autres

Le mécanisme de **réplication des logs** assure

- Un consensus sur les logs (via le leader central)
- Une reprise par le prochain leader là où l'autre est mort

La limite des ⌈N/2⌉-1 pannes est essentielle

- Un log n'est *committed* que quand une majorité en a connaissance
- Une majorité doit accepter les logs d'un candidat pour l'élire
- Il suffit donc d'une majorité de processus corrects pour avancer

Les propriétés formelles

Unicité du leader
Un seul leader à la fois

Progrès uniquement
Un leader ne peut qu'ajouter à ses logs, pas en supprimer. Ce qui est fait est fait.

Équivalence des logs
Si deux processus ont un même log, tous leurs logs précédents sont identiques.

Complétude du leader
Si un leader possède un log *committed*, les prochains leaders l'auront aussi.

Validité de la machine d'état
Si un leader applique une opération à sa machine d'état, alors aucun autre processus ne peut appliquer une autre opération pour le même log.