SDR

Diffusion

Élection avec pannes

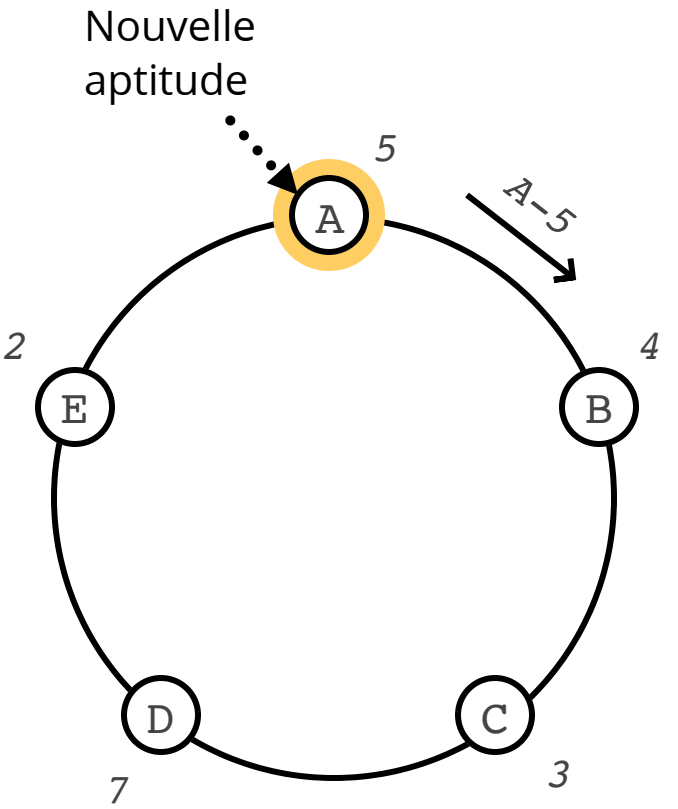
Olivier Lemer

Rappels

Chang et Roberts, 1979

Chang et Roberts, 1979

L'initiateur envoie son aptitude et son id.



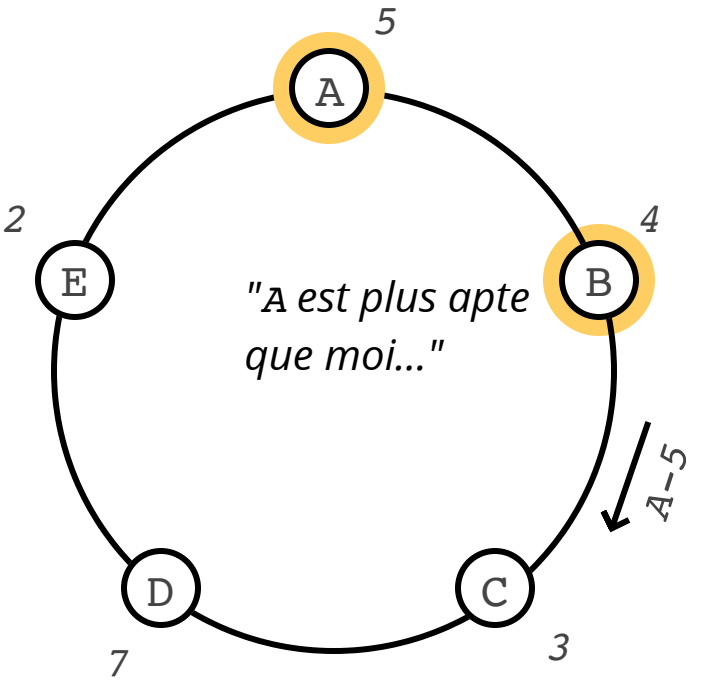
En élection

Connait l'élu

Chang et Roberts, 1979

L'initiateur envoie son aptitude et son id.

Si je reçois une aptitude
    Je la compare à la mienne
    Je propage la plus grande des deux



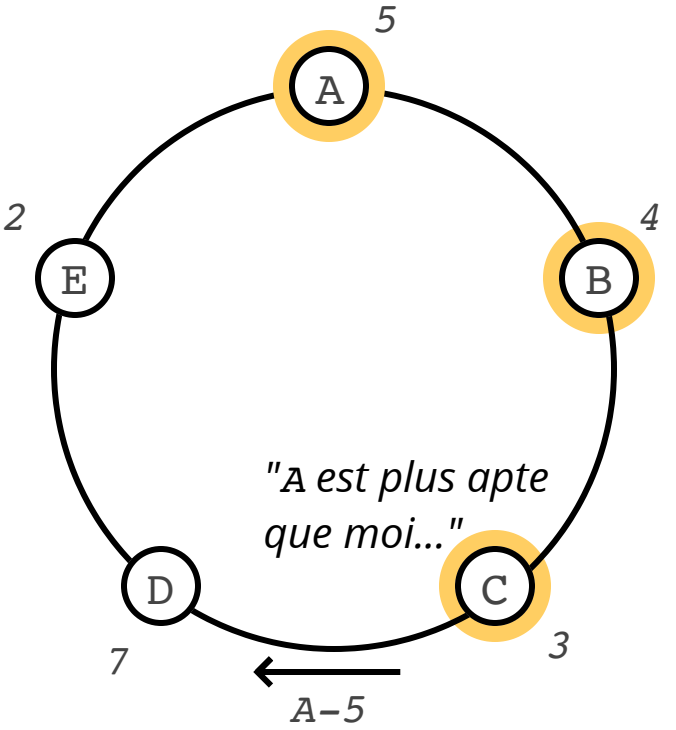
En élection

Connait l'élu

Chang et Roberts, 1979

L'initiateur envoie son aptitude et son id.

Si je reçois une aptitude d'un autre
- Je la compare à la mienne
- Je propage la plus grande des deux



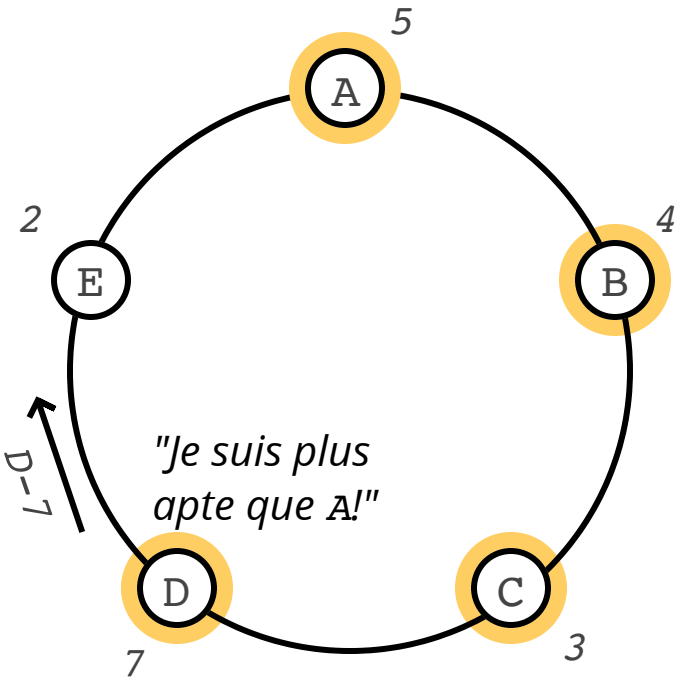
En élection

Connait l'élu

Chang et Roberts, 1979

L'initiateur envoie son aptitude et son id.

Si je reçois une aptitude d'un autre
- Je la compare à la mienne
- Je propage la plus grande des deux



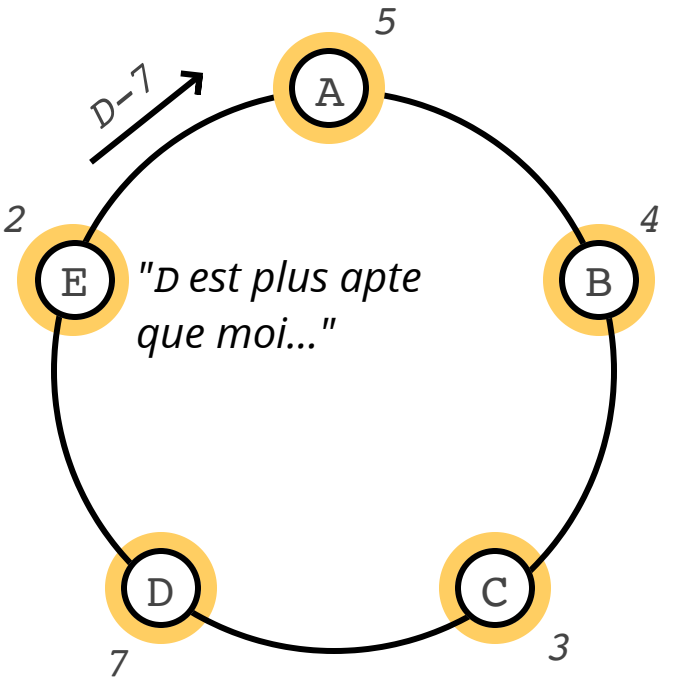
En élection

Connait l'élu

Chang et Roberts, 1979

L'initiateur envoie son aptitude et son id.

Si je reçois une aptitude d'un autre
- Je la compare à la mienne
- Je propage la plus grande des deux



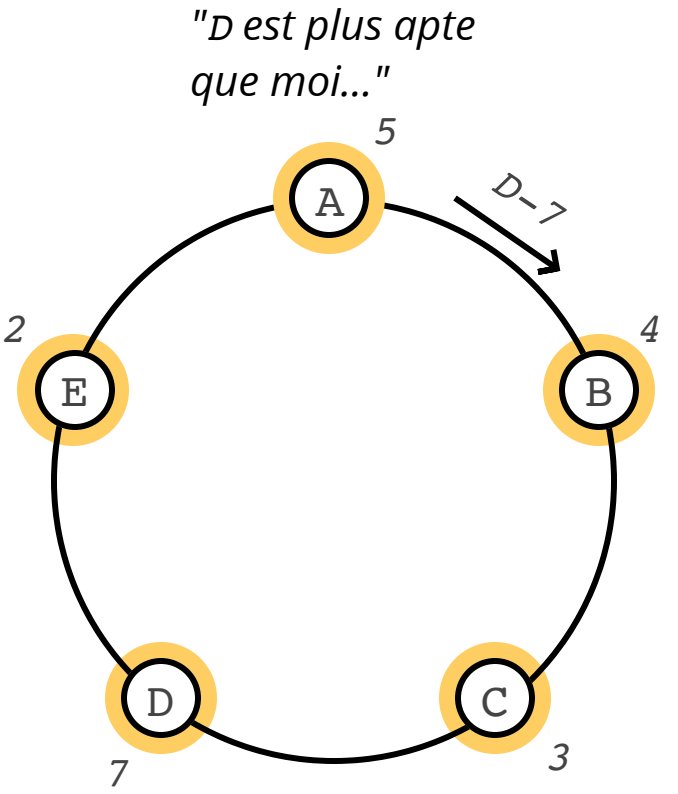
En élection

Connait l'élu

Chang et Roberts, 1979

L'initiateur envoie son aptitude et son id.

Si je reçois une aptitude d'un autre
- Je la compare à la mienne
- Je propage la plus grande des deux



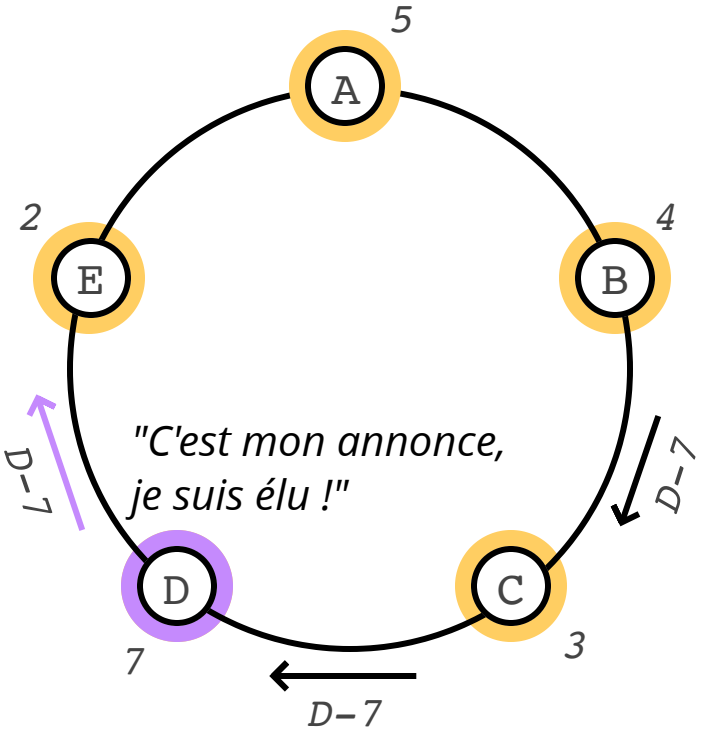
Chang et Roberts, 1979

L'initiateur envoie son aptitude et son id.

Si je reçois une aptitude d'un autre
- Je la compare à la mienne
- Je propage la plus grande des deux

Si je reçois ma propre aptitude
- L'annonce a fait le tour, je suis élu !
- Je propage le résultat

Tour d'annonce



Chang et Roberts, 1979

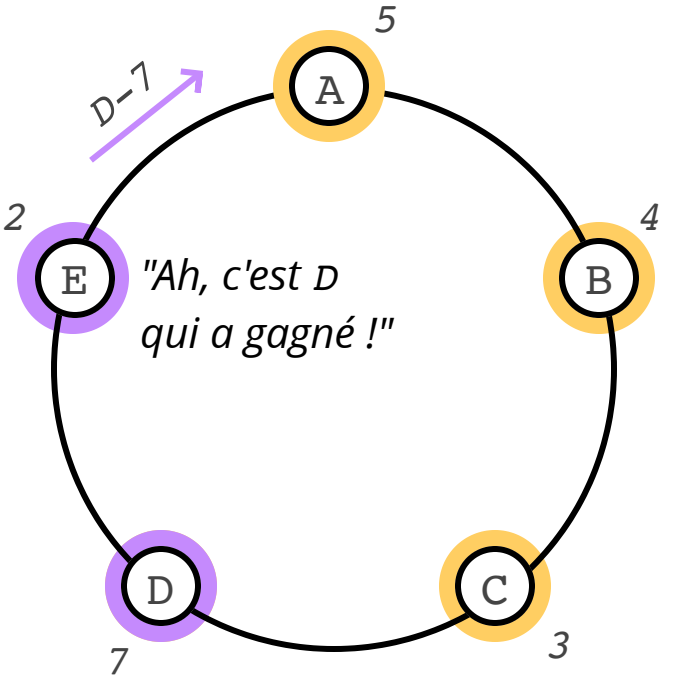
L'initiateur envoie son aptitude et son id.

Si je reçois une aptitude d'un autre
    Je la compare à la mienne
    Je propage la plus grande des deux

Si je reçois ma propre aptitude
    L'annonce a fait le tour, je suis élu !
    Je propage le résultat

Si je reçois un résultat
    Je le propage

Tour d'annonce



Chang et Roberts, 1979

L'initiateur envoie son aptitude et son id.

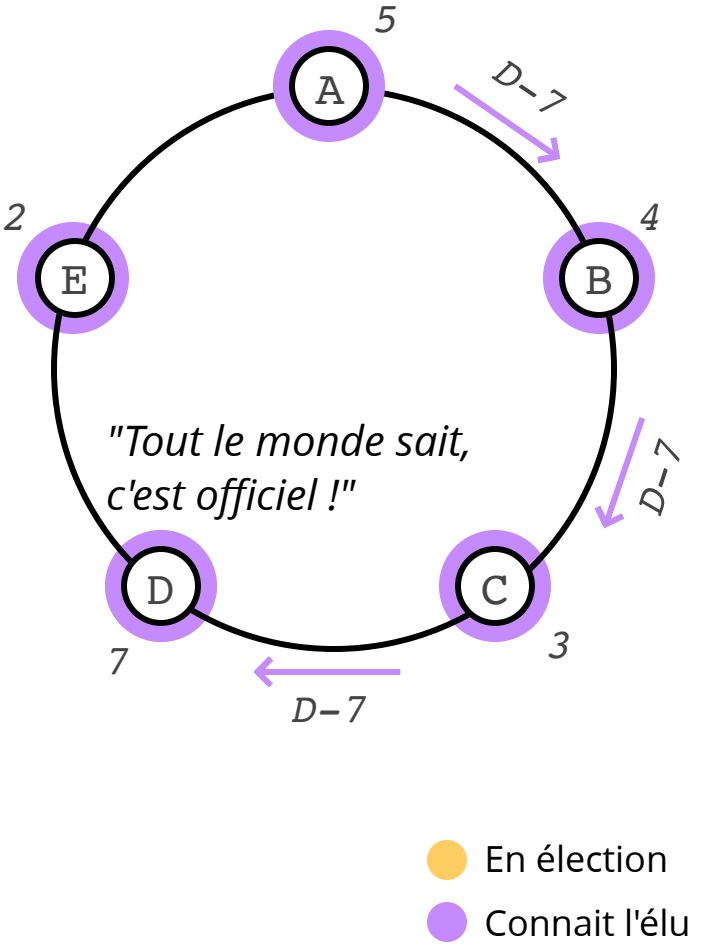
Si je reçois une aptitude d'un autre
  Je la compare à la mienne
  Je propage la plus grande des deux

Si je reçois ma propre aptitude
  L'annonce a fait le tour, je suis élu !
  Je propage le résultat

Si je reçois un résultat
  Je le propage si ce n'est pas le mien
  Sinon, l'élection est finie, je suis l'élu

Tour d'annonce

Tour de résultat



Chang et Roberts, 1979

L'initiateur envoie son aptitude et son id.

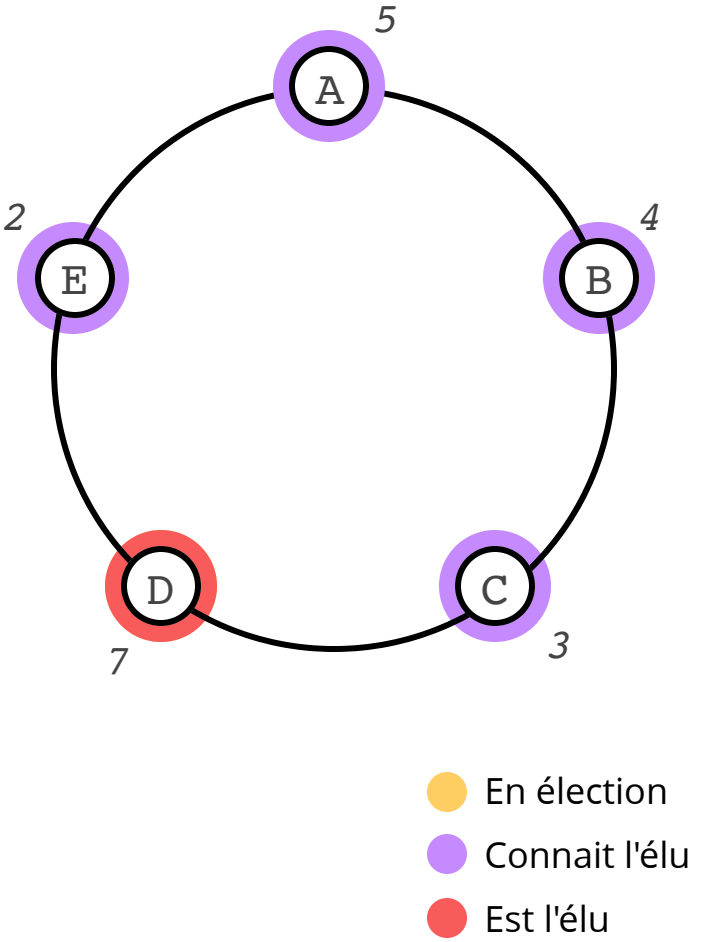
Si je reçois une aptitude d'un autre
  Je la compare à la mienne
  Je propage la plus grande des deux

Si je reçois ma propre aptitude
  L'annonce a fait le tour, je suis élu !
  Je propage le résultat

Si je reçois un résultat
  Je le propage si ce n'est pas le mien
  Sinon, l'élection est finie, je suis l'élu

Tour d'annonce

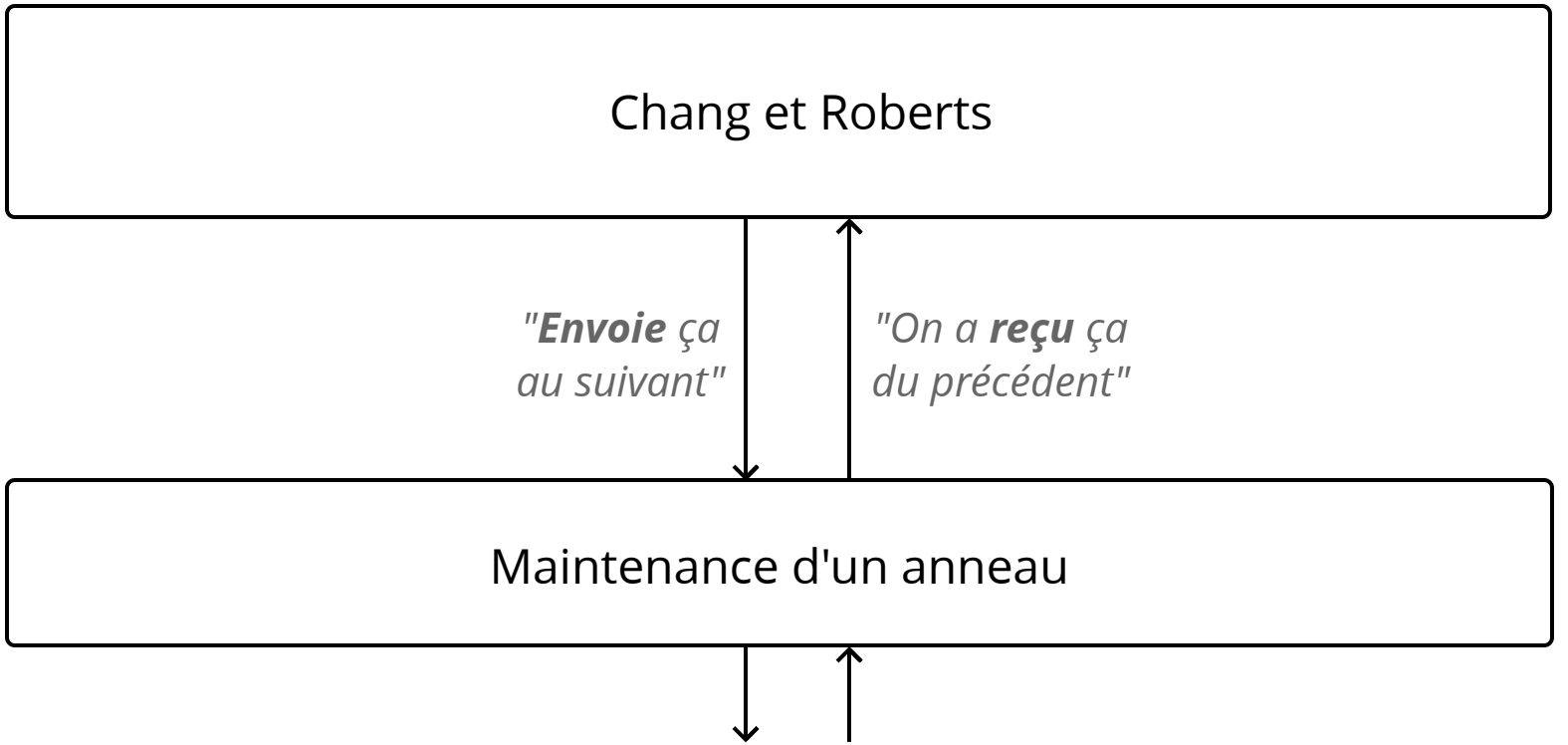
Tour de résultat



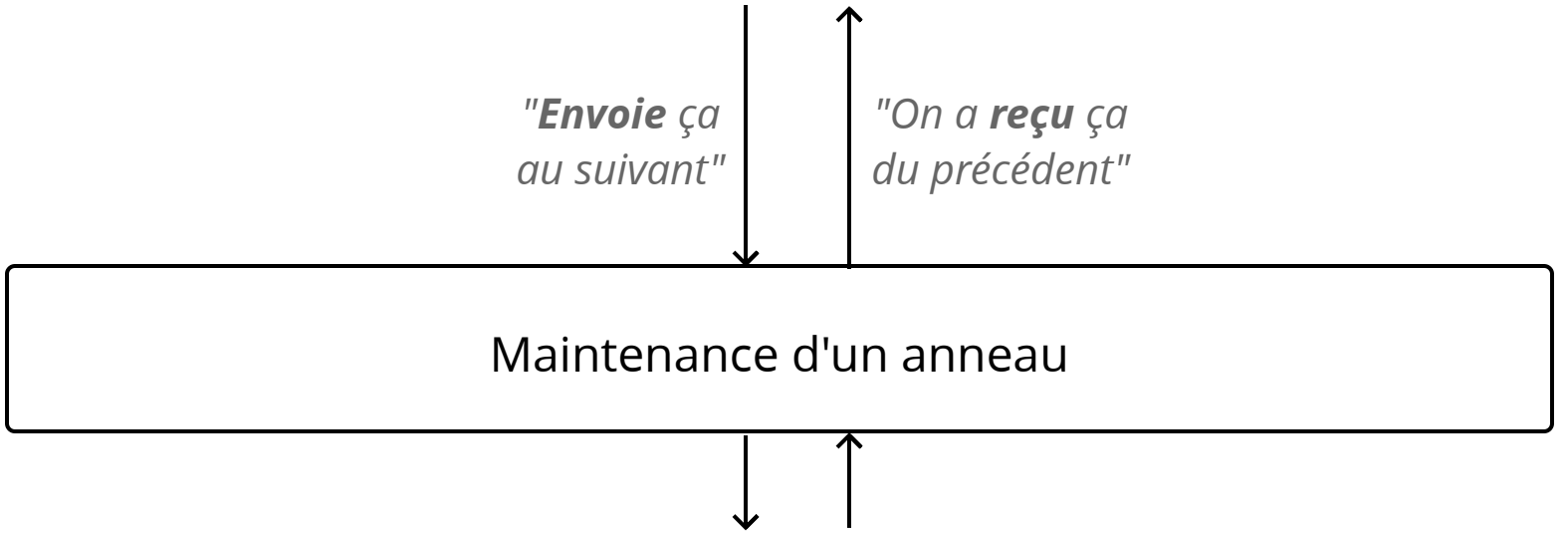
Maintien de l'anneau

Pour séparer les préoccupations

Separation of concerns



Maintenir un anneau



Validité : si un processus correct envoie `m`, alors `m` est délivré par le premier processus qui le suit dans l'anneau et n'est pas en panne.

Intégrité : `m` est délivré au plus une fois, seulement s'il a été envoyé, et dans l'ordre d'envoi.

Suppositions

- Pas de perte de messages.
- Pas de duplication de messages.
- Pas de changement d'ordre de messages.

- Une panne peut être ***récupérable***.
- Au moins 2 processus corrects.

- Durée de transit maximale **bornée** par une constante `T`.
- Durées de traitement négligeables.

- Chaque processus connait tous les autres.

Exemple

Lors d'une demande d'envoi,

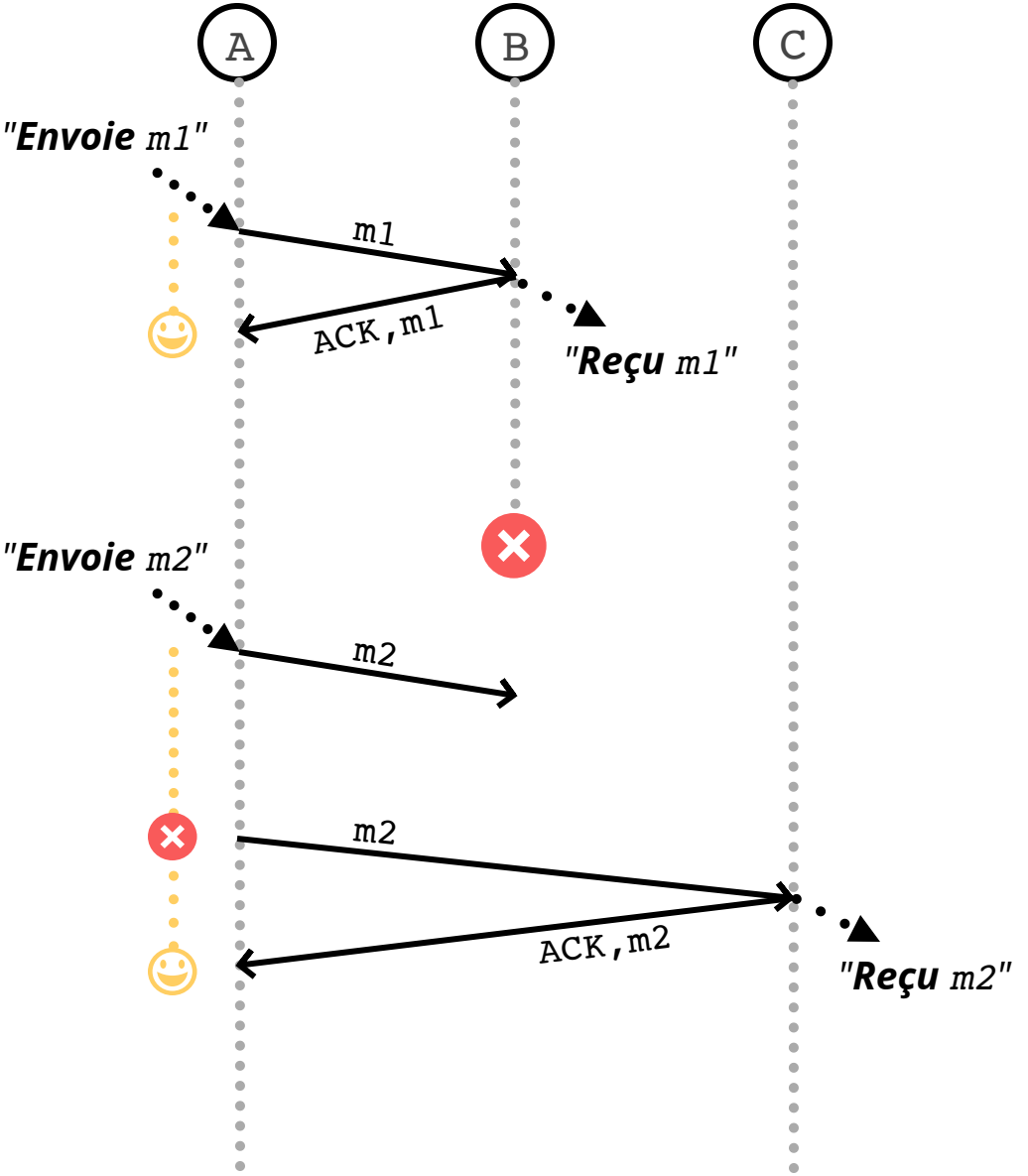
- envoie au prochain processus, et
- lance un timeout qui, une fois écoulé :
  - envoie le message au prochain dans l'anneau.

Lors d'une réception de `ACK`,

- annule le timer associé.

Lors d'une réception de message,

- notifie la réception de ce message, et
- envoie `ACK` avec un id du message.



Résumé

Chaque processus a une estampille `t`.

(pour identifier les messages)

Demande d'envoi au `n`ième suivant

Incrémenter `t`

Envoyer message et `t` au `n`ième suivant dans l'anneau

Lancer un timer, dont le timeout

> Demande l'envoi du message au `(n+1)`ième suivant.

Réception de `m` et `t_i`

Délivrer m à la couche applicative

Répondre `ACK` avec `t_i`

Réception de `ACK` avec `t_i`

Annuler le timeout associé à `t_i`

Propriétés

Validité ? *Oui*

Si un processus correct envoie `m`, alors `m` est délivré par le premier processus qui le suit dans l'anneau et n'est pas en panne.

Intégrité ? *Oui*

`m` est délivré au plus une fois, seulement s'il a été envoyé, et dans l'ordre d'envoi.

Et si on enlevait la supposition d'un délai max de transit ?

Validité ? *Oui*

Un envoi sera tenté vers le prochain qui n'est pas en panne, et (parce qu'il ne peut pas y avoir de pertes), il finira par être reçu, et donc délivré.

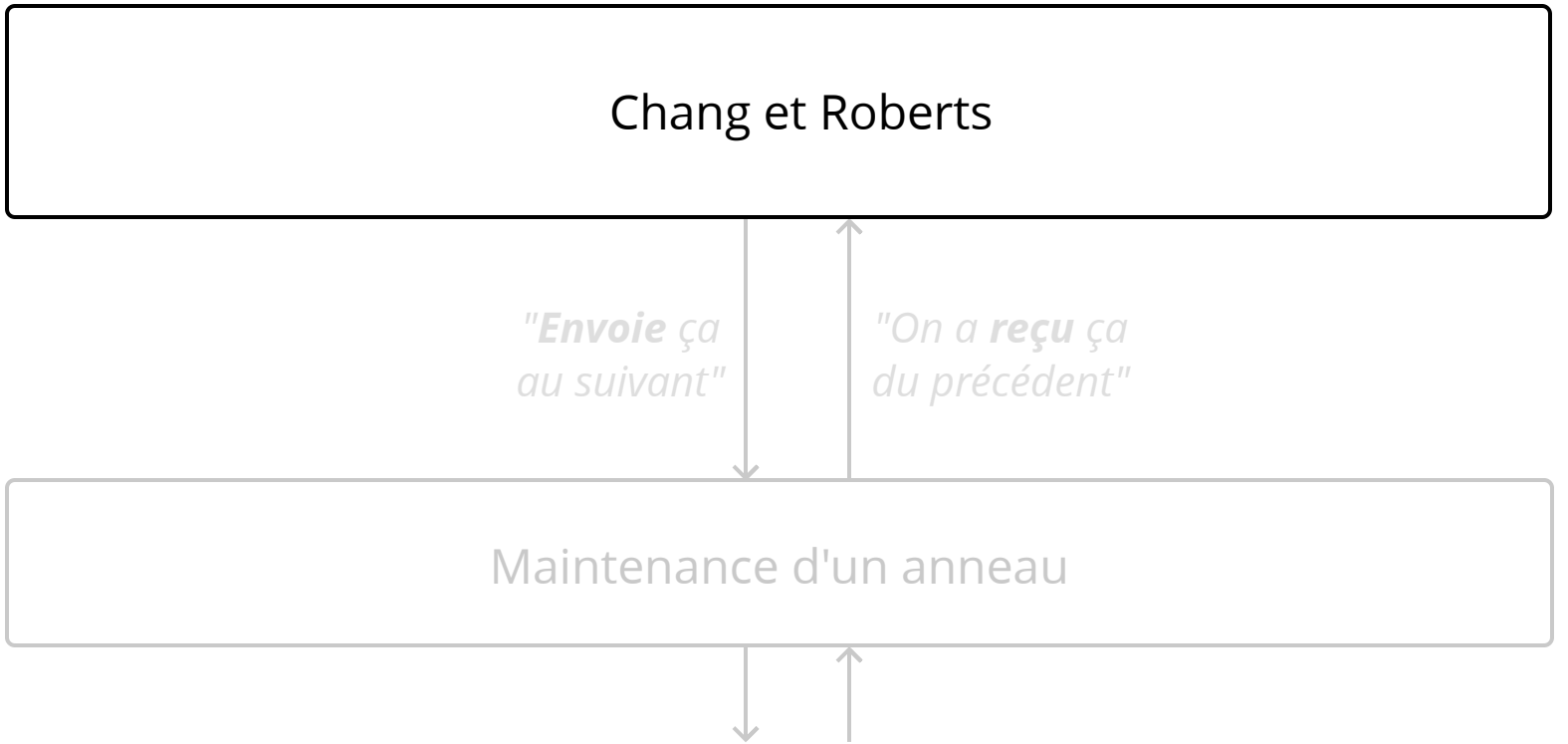
Intégrité ? *Non*

Un processus lent peut être sauté à tort : il délivre le message, et son suivant aussi.

Avec anneau maintenu

Chang et Roberts avec gestion de pannes

Separation of concerns



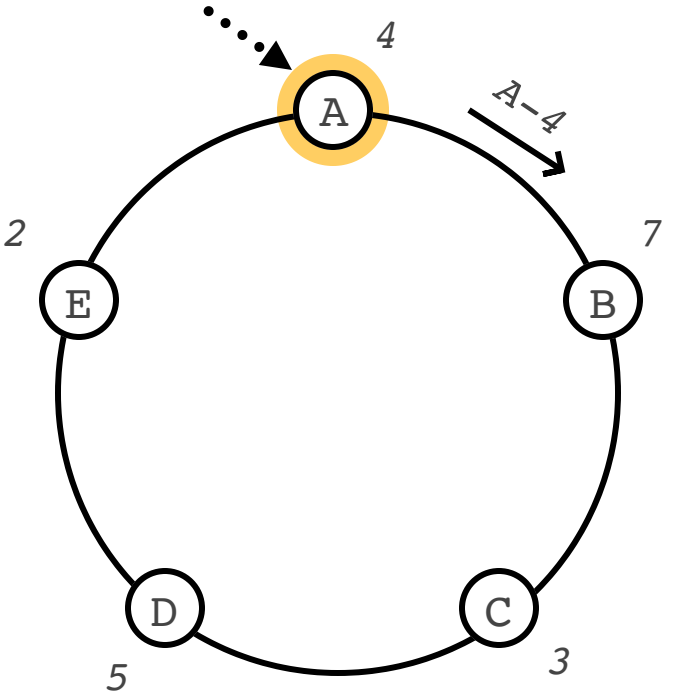
Changements nécessaires

Pour gérer les pannes

Toutes communications à travers le mainteneur d'anneau.

C'est tout ? **Risque de famine !**

Risque de famine

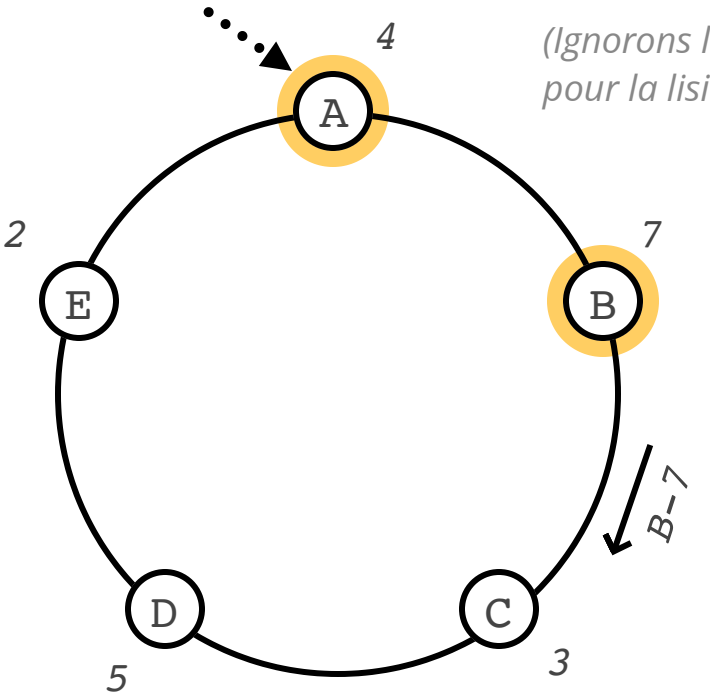


En élection

Connait l'élu

Est l'élu

Risque de famine



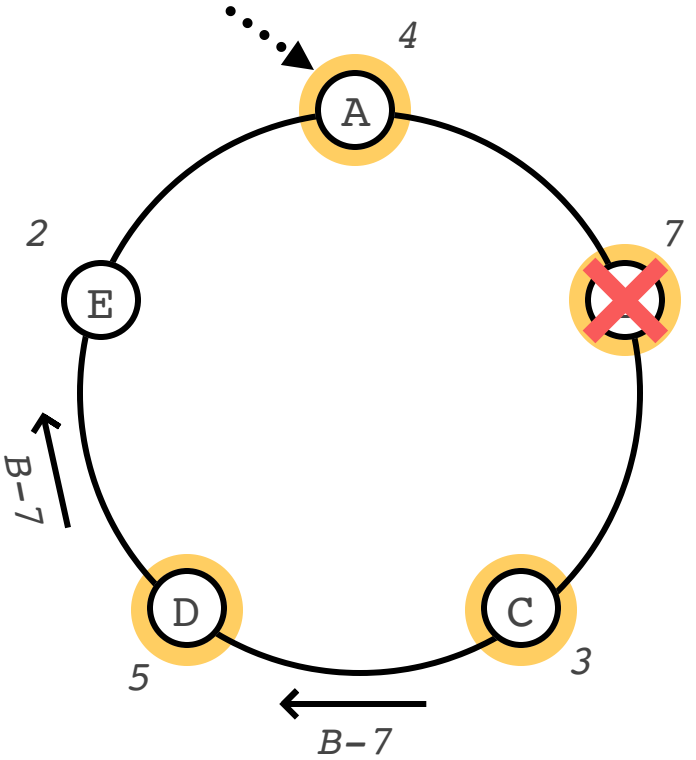
(Ignorons les ACKs pour la lisibilité)

En élection

Connait l'élu

Est l'élu

Risque de famine

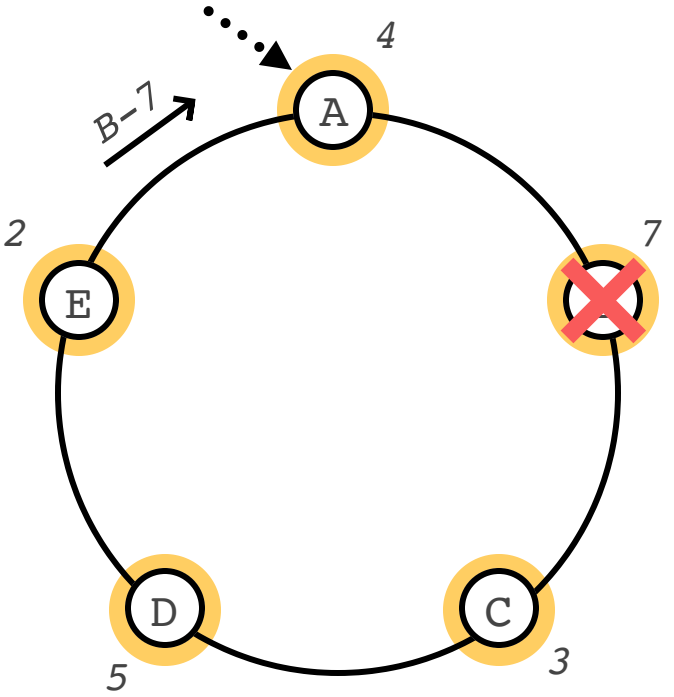


En élection

Connait l'élu

Est l'élu

Risque de famine

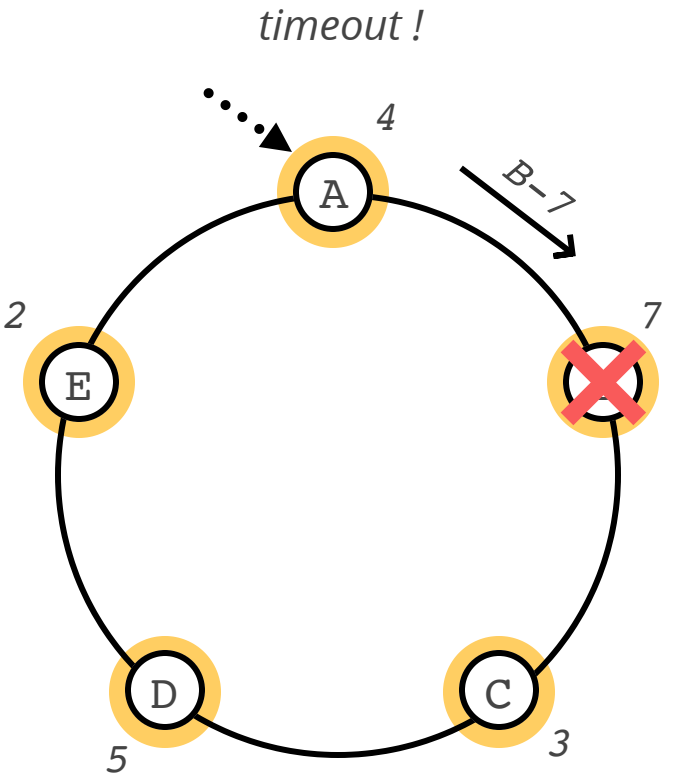


En élection

Connait l'élu

Est l'élu

Risque de famine

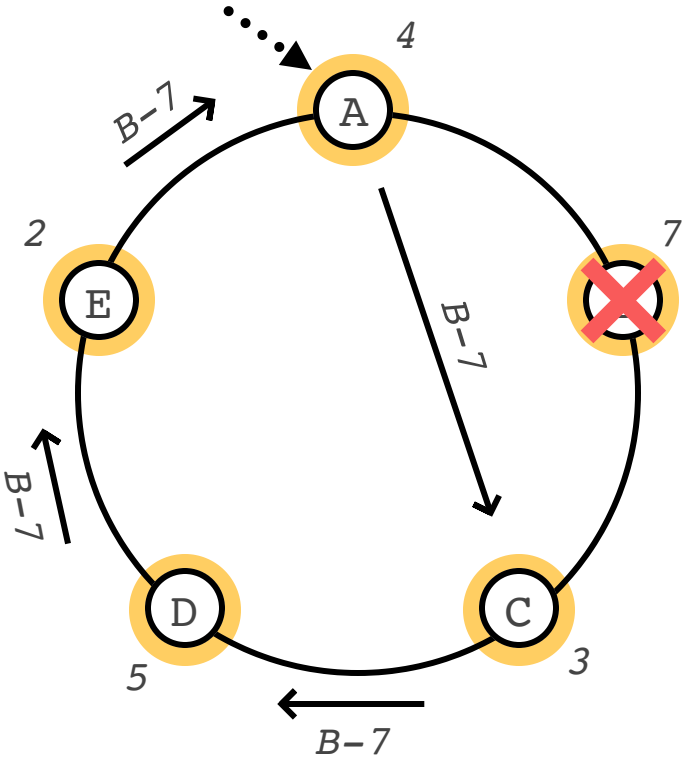


En élection

Connait l'élu

Est l'élu

Risque de famine



La phase "annonce" se termine quand l'élu reçoit son annonce.

Ça n'arrivera jamais...

- En élection
- Connait l'élu
- Est l'élu

Changements nécessaires

Pour gérer les pannes

Idée S'arrêter quand l'annonce a fait un tour complet

Comment ? Transmettre avec l'annonce tous les processus visités jusqu'ici :

Avant :

- id du processus à la meilleure aptitude jusqu'ici
- meilleure aptitude jusqu'ici

```
{i, apt_i}
```

Après :

- Pour chaque processus traversé :
  - id du processus
  - aptitude du processus

```
[
  {i, apt_i},
  {j, apt_j},
  ...
]
```

"Tous les processus ayant candidaté"

Quand je suis dans la liste d'une annonce : elle a fait le tour, j'élis le meilleur.

Satisfaits ? ***Et si je rejoins une élection en cours ?***

Changements nécessaires

Pour gérer les pannes

Idée : Relancer l'élection si je n'y ai pas participé.

Comment ? Transmettre **avec le résultat** tous les processus y ayant participé :

Avant :

- id de l'élu

`{i}`

Après :

- id de l'élu
- Pour chaque processus visité
  - id du processus

```
{
  i_elu,
  [i, j, ...]
}
```

"Tous les processus ayant accepté le résultat."

Si je reçois un résultat que j'ai accepté, il a fait le tour et je ne le propage pas.

Si je reçois un résultat que je n'ai pas accepté et que je ne suis pas en élection, alors je n'y ai pas participé et je relance l'élection *(sauf si mon élu est déjà le bon).*

Et maintenant ? ***Pas trop mal...***

Chang et Roberts, 1979

Autres questions

Et si celui qui va être élu tombe en panne après l'envoi d'un résultat ?

Il sera élu ! C'est ensuite qu'un détecteur de panne le verra et démarrera une élection.

Et si celui qui va être élu tombe en panne après l'envoi d'une annonce ?

Il sera élu ! C'est ensuite qu'un détecteur de panne le verra et démarrera une élection.

Et si celui qui va être élu tombe en panne après l'envoi d'une annonce, mais réapparait avant la réception du résultat ?

Il sera élu ! Et personne ne l'aura remarqué !

Et si on enlevait la supposition qu'un transit dure moins de T unités de temps ?

On risque d'élire un processus dont l'aptitude n'est pas la plus grande.

Avec anneau maintenu

Résumé

Résumé

Changement d'aptitude

Si je ne suis pas en cours d'élection

- J'adopte la nouvelle aptitude
- J'entre en élection
- J'envoie une annonce [{moi, apt}]

Si je suis en cours d'élection

- Je la mets de côté
  (je l'adopterai en sortant d'élection)

Conceptuellement Inchangé

Réception d'un résultat

Réception d'une annonce

Si je suis dans la liste de l'annonce

- Je calcule l'élu d'après la liste
- J'envoie un résultat {élu, [moi]}
- Je sors d'élection

Sinon

- J'ajoute {moi, apt} dans la liste
- J'envoie la liste au suivant
- J'entre en élection

Notez : Tout ça indépendamment de mon état (en élection ou non)

Résumé

Changement d'aptitude

Si je ne suis pas en cours d'élection

- J'adopte la nouvelle aptitude
- J'entre en élection
- J'envoie une annonce [{moi, apt}]

Si je suis en cours d'élection

- Je la mets de côté
  (je l'adopterai en sortant d'élection)

Réception d'une annonce

Si je suis dans la liste de l'annonce

- Je calcule l'élu d'après la liste
- J'envoie un résultat {élu, [moi]}
- Je sors d'élection

Sinon

- J'ajoute {moi, apt} dans la liste
- J'envoie la liste au suivant
- J'entre en élection

Réception d'un résultat

Si je suis dans la liste

- Je n'ai rien à faire !

Si je ne suis pas dans la liste

- Si je ne suis pas en élection
  et l'élu reçu est différent du mien
  - J'entre en élection et j'envoie une annonce [{moi, apt}]
- Sinon
  - Je note le nouvel élu
  - Je m'ajoute à la liste
  - J'envoie un résultat avec l'élu et cette liste
  - Je sors d'élection.

Pourquoi pas plus ?

Je suis dans la liste, tout le monde (pas en panne) a vu le résultat, on peut s'arrêter.

Avec anneau maintenu

Pseudocode

Pseudocode

Variables

`self`	*entier, constant*	mon numéro de processus
`apt_self`	*entier*	mon aptitude
`inElection`	*booléen, init false*	ssi une élection est en cours
`élu`	*entier*	id du processus élu
`apt_pending`	*entier, init* $\perp$	aptitude mise de côté

Remarques

Plus besoin de l'id du prochain dans l'anneau : l'envoi est géré par le mainteneur d'anneau.

Pseudocode

Initialisation

Démarrer une élection

Écouter infiniment les événements suivants :

- Demande de changement d'aptitude à `new_apt`
- Demande d'obtention de l'élu par la couche applicative
- Réception d'une annonce de `i` avec liste `participants`
- Réception d'un résultat de `i` avec `{new_élu, accepted}`

Note : les traitements de ces événements sont faits de manière séquentielle.

Pseudocode

Demande de changement d'aptitude à `new_apt`

Si `inElection`
    `apt_pending ← new_apt`
Sinon
    `apt_self ← new_apt`
    Démarrer une élection

Inchangé

Demande d'obtention de l'élu

Retourner `élu`.

Inchangé

Pseudocode

Démarrer une élection

Envoi d'une annonce `[{self, apt_self}]`

`inElection ← true`

Inchangé

Sortir d'élection

`inElection ← false`

Si `apt_pending ≠ ⊥` et `apt_pending ≠ apt_self`

    `apt_self ← apt_pending`

    Démarrer une élection

`apt_pending ← ⊥`

Inchangé

Pseudocode

Réception d'une annonce de `i` avec liste `participants`

Si `self` apparait dans `participants` *(L'annonce a fait le tour)*

    `élu ← i` tel que `i` a l'aptitude max dans `participants`

    Envoi de résultat `{élu, [self]}`

    Sortir d'élection

Sinon

    Ajout de `{self, apt_self}` dans `participants`

    Envoi d'annonce avec `participants`

    `inElection ← true`

Pseudocode

Réception d'un résultat de `i` avec `{new_élu, accepted}`

Si `self` n'apparait **pas** dans `accepted` *(Il s'agit d'un résultat inédit)*

- Si pas `inElection` et `élu` n'est pas `new_élu` *(Il s'agit d'une **élection** inédite)*
  - *Démarrer une élection* *(il faut recommencer)*
- Sinon *(j'accepte le résultat)*
  - `élu ← new_élu`
  - Ajout de `self` dans `accepted`
  - Envoi d'un résultat `{élu, accepted}`
  - *Sortir d'élection*