SDR

Diffusion

Élection naïve

Olivier Lemer

↓ Élections ?

Le problème de l'élection

Étant donné un groupe de processus.

Chacun a une **aptitude**, pas nécessairement statique.

On veut **élire** celui qui a la plus grande aptitude.

Par exemple :

- ***Algorithme à jeton*** *: régénérer un jeton après perte*
- ***Architecture manager/worker*** *: désigner un nouveau manager*
- ***Répartition de charge*** *: sélectionner le serveur le moins chargé*

Propriétés souhaitées

- **Sureté** : un seul processus peut être élu à la fois
- **Progrès** : il doit y avoir un élu un jour
- **Validité** : l'élu doit être le participant correct à la plus grande aptitude
- **Résilience** :
  - Un processus incorrect ne doit pas être élu, même s'il redémarre.
  - Si le réseau est scindé en deux, un élu doit exister par partition.

Différence avec Mutex

Mutex

Essaie d'être équitable

Partage l'accès à une section

Tout demandeur sera accepté

Bloquant si SC est prise

Élection

Choisit la meilleure aptitude

Partage le droit au titre d'élu

Un processus peut ne jamais être élu

Non-bloquant si un autre est élu

Là où la Mutex assure que tout le monde aura droit à son moment,
l'élection sert uniquement à décider à qui attribuer un titre spécial d'*élu*.

Rappel

Suppositions

Pour l'instant, on supposera

- **Pas de perte** de message
- **Pas de duplication** de message
- **Pas de changement d'ordre** de messages
- **Pas de panne** serveur

- **Système synchrone** : délai d'envoi de T maximum connu

Algorithme du Bully

a.k.a. un algorithme naïf

Bully Algorithm

Idée *"tout le monde doit tout savoir"*

J'envoie mon aptitude à tout le monde.

Je reçois l'aptitude de tout le monde.

Le plus fort gagne.

Bully Algorithm

Deux événements

Une élection commence

Je broadcast mon aptitude.

J'attends 2T pour tout recevoir.

Je détermine l'élu.

Réception d'une aptitude

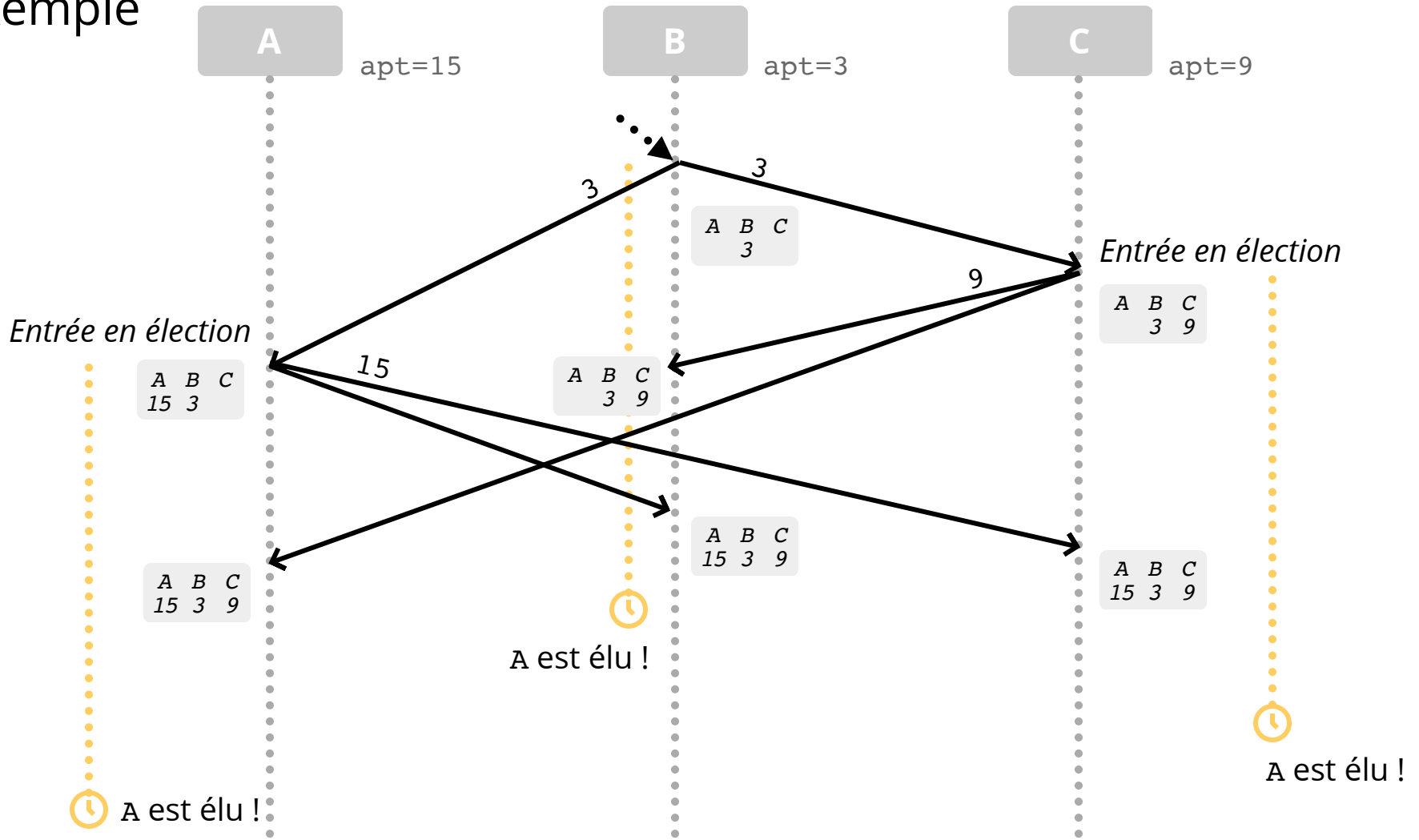
Si je ne suis pas déjà en élection,

j'entre en élection.

Si mon aptitude change pendant une élection,
la nouvelle élection est différée à la fin de celle en cours.

Bully Algorithm

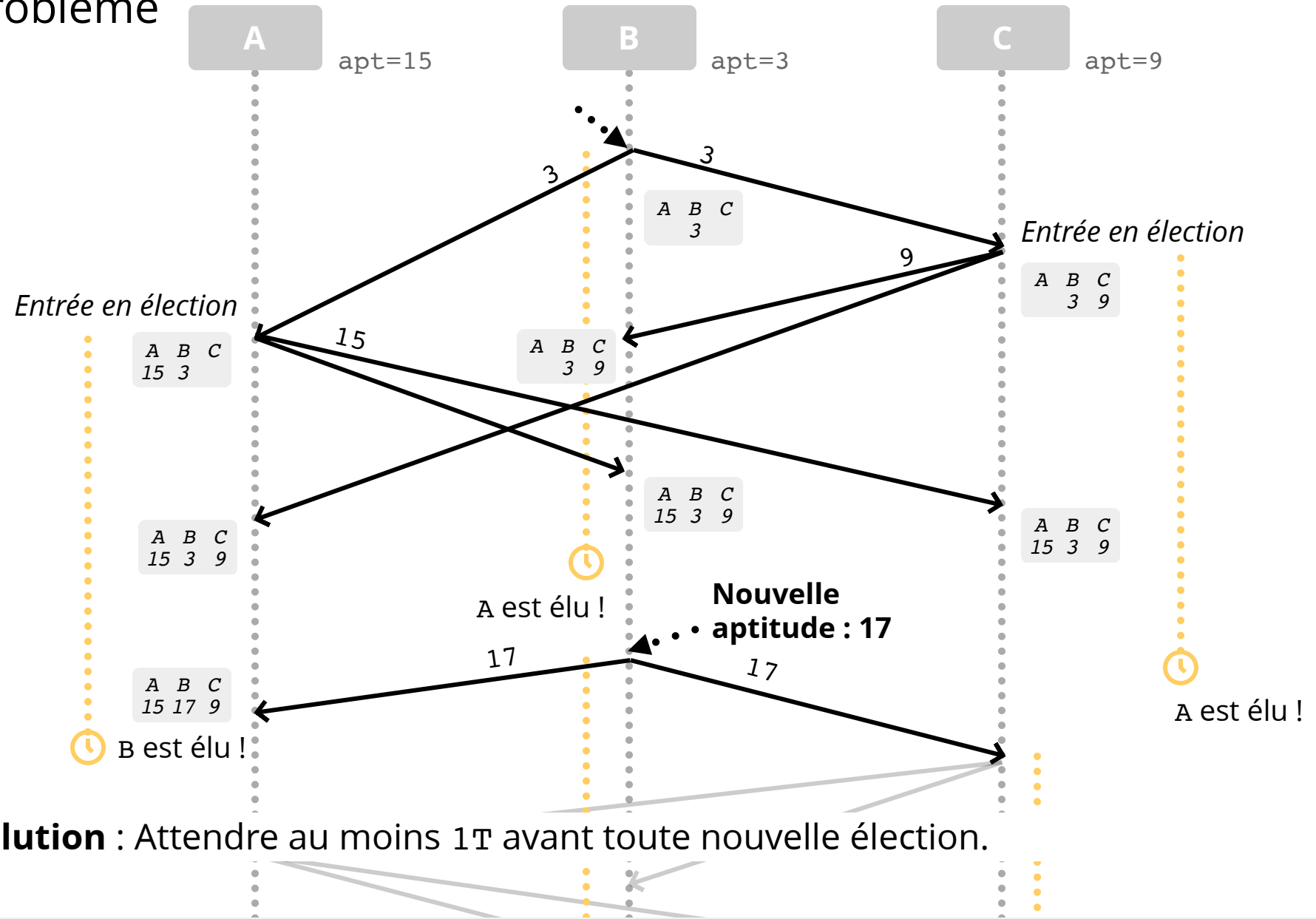
Exemple



Problème ?

Bully Algorithm

Problème



Solution : Attendre au moins 1T avant toute nouvelle élection.

Bully Algorithm

Performance

Communication

`O(N^2)` messages par élection

Durée d'une élection

Jusqu'à `4T`

(`1T` d'attente, `2T` de timeout, `1T` de transit)

Algorithme du Bully

Pseudocode

Pseudocode

Variables

`n`	*entier, constant*	nombre de processus
`self`	*entier, constant*	mon numéro de processus
`apts`	*tableau d'entiers*	aptitudes de tous les processus
`apt_self`	*entier*	mon aptitude
`T`	*entier, constant*	durée maximale d'une transmission
`inElection`	*booléen, init false*	ssi une élection est en cours
`isReqPending`	*booléen, init false*	ssi une élection est en attente
`élu`	*entier*	id du processus élu

Pseudocode

Initialisation

Démarrer une élection

Écouter infiniment les événements suivants :

- Demande d'obtention de l'élu par la couche applicative
- Demande de changement d'aptitude à `new_apt`
- Réception de l'aptitude `apt_i` de `i`
- Timeout

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

Pseudocode

Demande d'obtention de l'élu par la couche applicative

```
Retourner elu
```

Demande de changement d'aptitude à `new_apt`

```
apt_self ← new_apt
Attendre pendant 1T
Démarrer une élection
```

Pseudocode

Réception de l'aptitude `apt_i` de `i`

```
Si pas inElection
    Démarrer une élection
apts[i] ← apt_i
```

Démarrer une élection

```
Si inElection
    isReqPending ← true
Sinon
    inElection ← true
    apts[j] ← -inf pour tout j
    apts[self] ← apt_self
    Envoi de apt_self à tous les autres processus
    Lancer un timer de 2T
```

Pseudocode

Timeout

```
élu ← i tel que apts[i] est le premier maximum de apts.
inElection ← false
Si isReqPending
    isReqPending ← false
    Attendre pendant 1T
    Démarrer une élection
```

Algorithme du Bully

Pseudocode++ *(avec deux routines)*

Pseudocode à 2 routines

Diffuseur

Responsable de déclencher les élections et communiquer avec l'application.

Collecteur

Responsable de collecter les aptitudes.

Permet de tirer profit de la concurrence.

Pseudocode à 2 routines

Variables partagées

`n`	*entier, constant*	nombre de processus
`self`	*entier, constant*	mon numéro de processus
`T`	*entier, constant*	durée maximale d'une transmission

Pseudocode à 2 routines

Routine 1 - Diffuseur

Variables de la routine 1

`élu`	*entier*	id du processus élu
`apts`	*tableau d'entiers*	aptitudes de tous les processus
`apt_self`	*entier*	mon aptitude

Initialisation

Exécuter *entrer en élection*
Écouter infiniment les événements suivants :
- Demande d'entrée en élection
- Demande de changement d'aptitude à `new_apt`
- Demande de l'élu par la couche applicative

Pseudocode à 2 routines

Routine 1 - Diffuseur

Demande d'entrée en élection

```
Exécuter entrer en élection
```

Demande de changement d'aptitude à `new_apt`

```
apt_self ← new_apt
Exécuter entrer en élection
```

Demande de l'élu par la couche applicative

```
Retourner elu
```

Pseudocode à 2 routines

Routine 1 - Diffuseur

Entrer en élection

Attente bloquante durant `1T`
Envoi de `apt_self` à tous les autres processus
Envoi de `apt_self` à la routine *Collecteur*
Attente bloquante durant `2T`
Obtention de `apts` de la routine *Collecteur*
`élu ← i` tel que `apts[i]` est le premier maximum de `apts`

Pseudocode à 2 routines

Routine 2 - Collecteur

Initialisation

Boucle sur l'attente de réception de `apt_i` de la part de `i`
    `apts` ← Tableau de taille `n` initialisé à `-inf`
    `apts[i] ← apt_i`
    Si `i != self`
        Demander une élection à *Diffuseur*
    Boucle sur l'attente de
        Réception de `apt_i` de `i`
            `apts[i] ← apt_i`
        Demande de `apts` par *Diffuseur*
            Retourner `apts`
            Sortir de cette boucle

Algorithme du Bully

En cas de panne ?

Gestion des pannes

Supposition

Un processus ne tombe jamais en panne *pendant* l'envoi de messages en batch.

Soit il envoie tout, soit il n'envoie rien.

(e.g. via un algorithme de diffusion fiable)

Seul danger : Pendant une élection qu'il allait gagner, ou après son élection.

Gestion de telles pannes

On suppose alors un *détecteur de panne* chez les autres, surveillant l'élu :

Si je détecte une panne de l'élu, je démarre une nouvelle élection.

Détecteur parfait ou un jour ?

Un jour suffit : si suspicion annulée, relancer une élection.

En cas de faux positif, le même élu sera à nouveau choisi.