Algorithmique Répartie

Élection

Université François Rabelais de Tours
Faculté des Sciences et Techniques
Antenne Universitaire de Blois

Master 1 Informatique

Option Systèmes d'Information et Aide à la Décision

Mohamed Taghelit
taghelit@univ-tours.fr

L'Élection

Problématique

- même si les stations d'une application répartie ont des fonctionnalités similaires, certaines tâches spécifiques doivent être réalisées par une station unique,
- avant de démarrer l'application, les sites doivent se mettre d'accord sur l'identité de cette station,
- cette opération est communément appelée élection.

Intérêts de l'élection

- applications de type maître/esclaves :
  - serveurs redondants pour le traitement des requêtes de lecture des clients,
  - les requêtes d'écriture ne sont traitées que par un seul serveur (maître) qui transmet ensuite les modifications aux autres sites (esclaves).

- applications nécessitant une phase d'initialisation exécutée par un seul site :
  - circulation d'un jeton unique entre les sites,
  - l'initiateur est le premier possesseur du jeton.

L'Élection

Alternatives des solutions

- solution statique
  - fixer l'identité de ce site dans le code ou mieux dans un fichier de configuration,
  - si changement de configuration du réseau
    - modification par l'administrateur du (ou des) fichier(s) de configuration,
    - redémarrage des sites.

- solution dynamique
  - l'élection a lieu uniquement lorsqu'un site de l'application a besoin d'utiliser cette identité,
  - le service a une interface constituée d'une seule fonction **`leader()`** qui renvoie l'identité du site élu
  - contrainte : l'identité renvoyée est toujours la même quelque soit l'instant ou le lieu de l'appel.

L'Élection

Organisation du chapitre

- étude de trois algorithmes,
- type de graphe de communication :
  - anneau unidirectionnel,
  - anneau bidirectionnel,
  - graphe quelconque.
- avantage de ces trois algorithmes
  - nécessitent de chaque site la seule connaissance de ses voisins dans le graphe,
  - l'insertion d'un nouveau site ne requiert aucune intervention au niveau des sites qui ne sont pas ses voisins (moyennant le fait que le type de graphe est inchangé dans le cas de graphes spécifiques).

Algorithme de Chang et Roberts

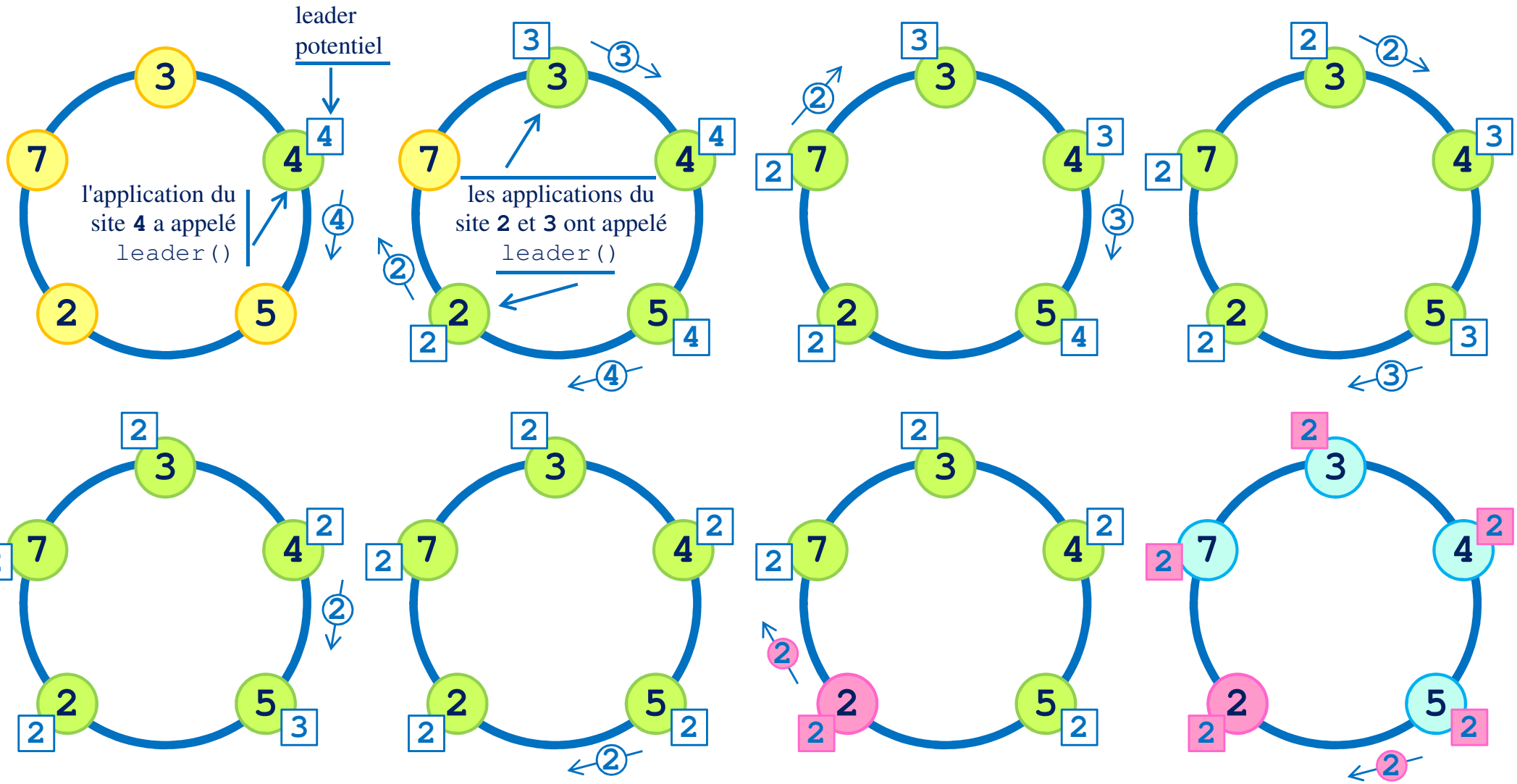
Principe de l'algorithme de Chang et Roberts (1979)

- s'applique à un anneau unidirectionnel,
- chaque site n'émet des messages que vers le site suivant sur l'anneau,
- lorsque l'application "réclame" l'identité du leader, trois cas se présentent :
  - le processus d'élection est terminé et le service renvoie immédiatement l'identité du site élu,
  - le processus d'élection est en cours et connu du service. Dans ce cas, le service attend la terminaison du processus afin de renvoyer l'identité du leader,
  - le service n'est pas au courant d'un processus d'élection engagé. Dans ce cas, il envoie une requête circuler sur l'anneau afin de devenir le leader. Si la requête lui revient, il conclut que son offre est acceptée et envoie un message de confirmation circuler sur l'anneau pour indiquer aux autres sites qu'il est le leader.

- lorsqu'une requête parvient à un autre site, deux cas se présentent :
  - le service du site était au repos ou avait provisoirement choisi un leader d'identité supérieure; il adopte l'initiateur de la requête comme nouveau leader potentiel et retransmet la requête au site suivant de l'anneau,
  - le service du site avait provisoirement choisi un leader d'identité inférieure; il ne donne pas suite à la requête.

Scénario d'Exécution de l'Algorithme de Chang et Roberts

Scénario d'exécution



Algorithme de Chang et Roberts

Variables du site **i** :

- **suivant$_i$** : constante contenant l'identité du site successeur de **i** sur l'anneau.
- **état$_i$** : état du service. Cette variable prend une valeur parmi l'ensemble des valeurs (**repos**, **en_cours**, **terminé**). Cette variable est initialisée à **repos**.
- **chef$_i$** : identité du site leader.

Si aucun processus d'élection n'a atteint le site **i**, le service initialise un tel processus.

```
leader(){
    Si (état_i == repos ) Alors
        état_i = en_cours;
        chef_i = i;
        envoyer_à(suivant_i, (req, i));
    Finsi
    Attendre( état_i == terminé);
    renvoyer(chef_i);
}
```

Dans tous les cas, il attend que le processus d'élection soit terminé pour renvoyer l'identité du site élu.

Algorithme de Chang et Roberts

Si le site est au repos, la réception d'une requête provoque le changement d'état et la retransmission de la requête.

Dans le cas où le processus reçoit une "meilleure" requête, il en tient compte et retransmet aussi la requête.

Si une requête parvient à son initiateur, ce site est élu et il avertit les autres sites.

La confirmation du site fait un tour de l'anneau.

```
sur_réception_de(j, (req, k)){
    Si ( état_i == repos || k < chef_i ) Alors
        état_i = en_cours;
        chef_i = k;
        envoyer_à(suivant_i, (req, k));
    Sinon
        Si ( i == k ) Alors
            état_i = terminé;
            envoyer_à(suivant_i, (conf, i));
        Finsi
    Finsi
}
```

```
sur_réception_de(j, (conf, k)){
    Si ( i != k ) Alors
        envoyer_à(suivant_i, (conf, k));
        état_i = terminé;
    Finsi
}
```

Preuve de l'Algorithme de Chang et Roberts

Un site dont l'application appelle **`leader()`** alors que son service est au repos sera désigné comme un initiateur. Soit $\mathbf{i_m}$ un initiateur

- $\mathbf{i_m}$ est l'initiateur d'identité minimale désigné par $\mathbf{i_0}$
  1. la requête $\mathbf{(req, i_0)}$ circule de bout en bout et atteint $\mathbf{i_0}$,
  2. le site $\mathbf{i_0}$ passera donc à l'état **`terminé`**, enverra son message de confirmation qui entraînera à sa réception le passage de tous les autres sites à ce même état et l'adoption de $\mathbf{i_0}$ comme leader.
- $\mathbf{i_m}$ n'est pas l'initiateur d'identité minimale $\mathbf{i_0}$
  - $\mathbf{i_m}$ est le site d'identité inférieure parmi tous les sites traversés entre $\mathbf{i_m}$ et $\mathbf{i_0}$
    - la requête $\mathbf{(req, i_m)}$ circulera jusqu'à $\mathbf{i_0}$ qui la supprimera pour envoyer la sienne $\mathbf{(req, i_0)}$
    - on revient à la situation du **`1`** ci-dessus
  - il existe un site $\mathbf{i_{inf}}$ d'identité inférieure à $\mathbf{i_m}$ parmi tous les sites traversés entre $\mathbf{i_m}$ et $\mathbf{i_0}$
    - la requête $\mathbf{(req, i_m)}$ circulera jusqu'à $\mathbf{i_{inf}}$ qui la supprimera pour envoyer la sienne $\mathbf{(req, i_{inf})}$
    - la requête $\mathbf{(req, i_{inf})}$ circulera jusqu'à $\mathbf{i_0}$ qui la supprimera pour envoyer la sienne $\mathbf{(req, i_0)}$
    - on revient à la situation du **`1`** ci-dessus

Il reste à montrer qu'aucun autre message de confirmation ne circulera sur l'anneau

- soit $\mathbf{i_m}$ un autre initiateur
  - si la requête de $\mathbf{i_m}$ parvient à $\mathbf{i_0}$, elle sera supprimée et ne parviendra donc jamais à $\mathbf{i_m}$
  - $\mathbf{i_m}$ ne pourra donc envoyer un autre message de confirmation

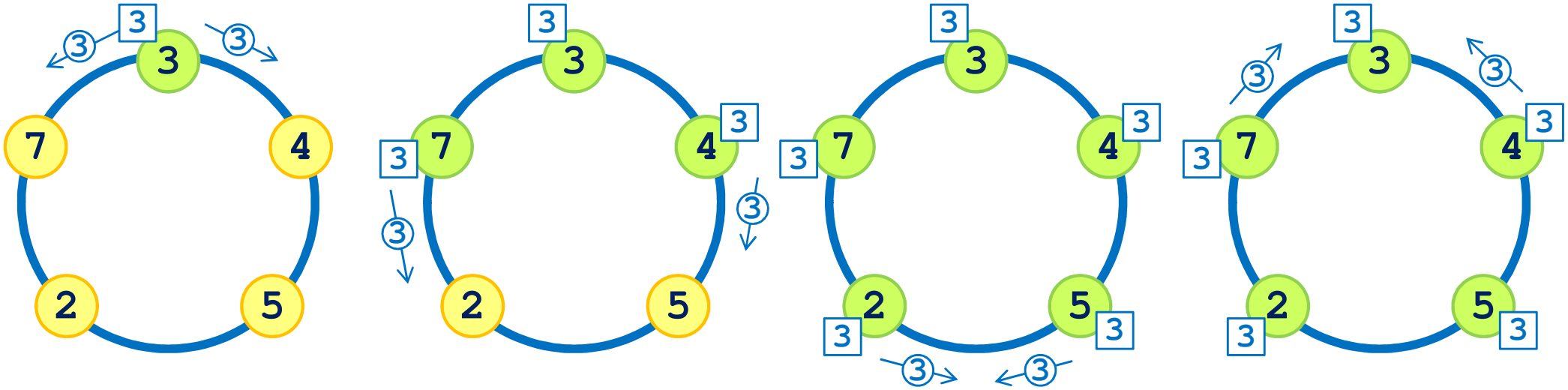
Algorithme de Franklin

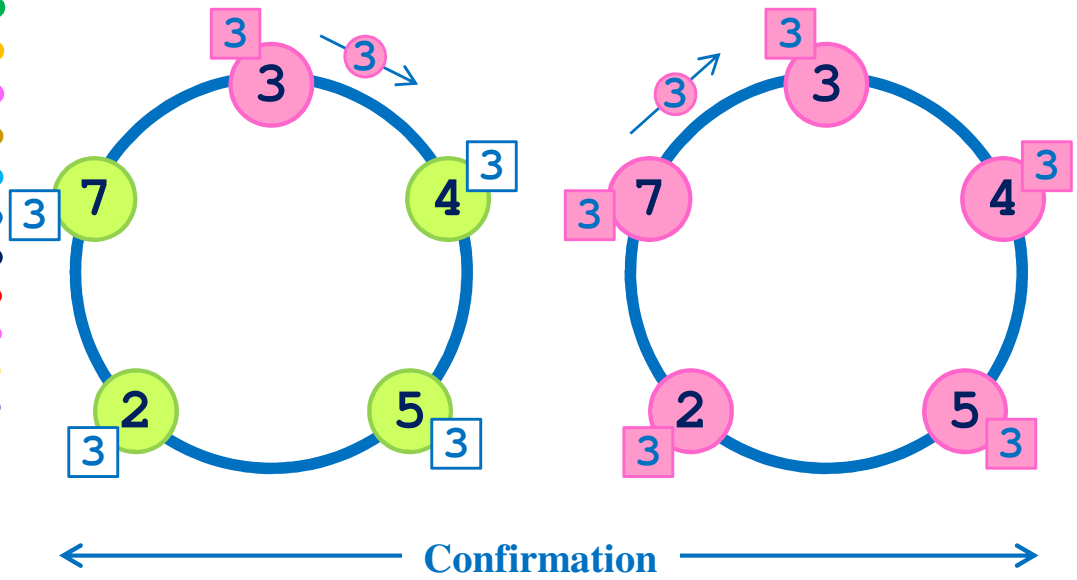
Principe de l'algorithme de Franklin(1982)

- s'applique à un anneau bidirectionnel,
- se décompose en "tours" où les compétiteurs sont les initiateurs,
- à chaque tour, un initiateur "survivant" envoie à ses initiateurs voisins (à gauche et à droite les plus proches) sa candidature,
- de ce fait, chaque initiateur reçoit les requêtes des initiateurs voisins gauche et droite (les plus proches) et ne survit au tour suivant que s'il a la plus petite identité,
- le tournoi s'achève quand un initiateur sait qu'il est ou sera au prochain tour le seul survivant, certitude qu'il acquière :
  - soit en recevant l'un des messages qu'il a envoyé (ou éventuellement les deux),
  - soit en recevant deux messages d'un même site.
- les sites "éliminés" participent à l'algorithme en transmettant les messages qu'ils reçoivent,
- un message de confirmation achève l'algorithme.

Scénario d'Exécution de l'Algorithme de Franklin

Scénario d'exécution : un seul initiateur



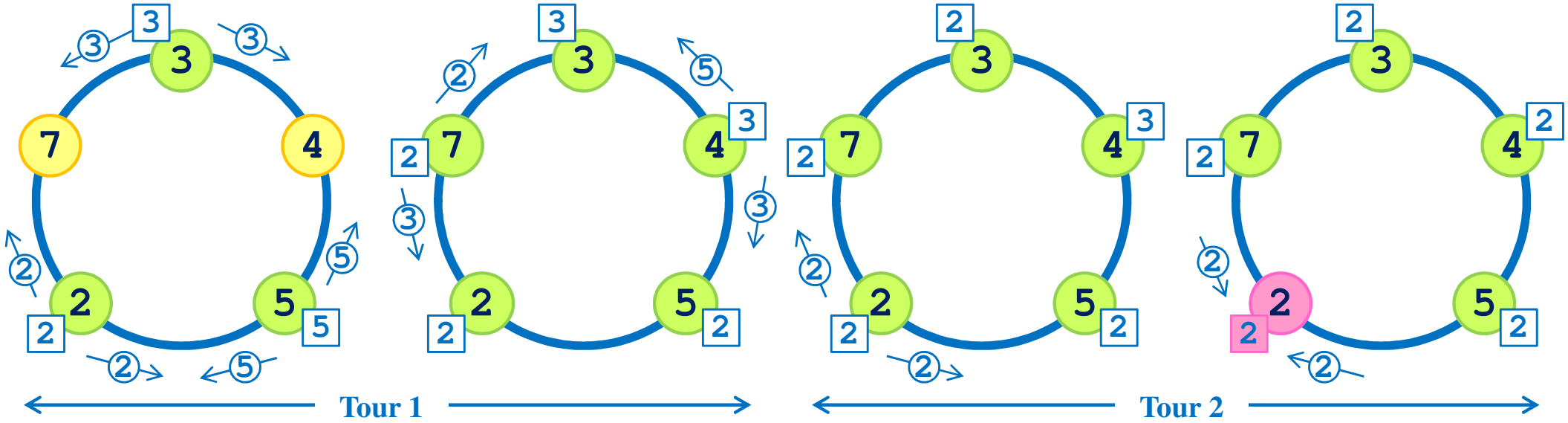


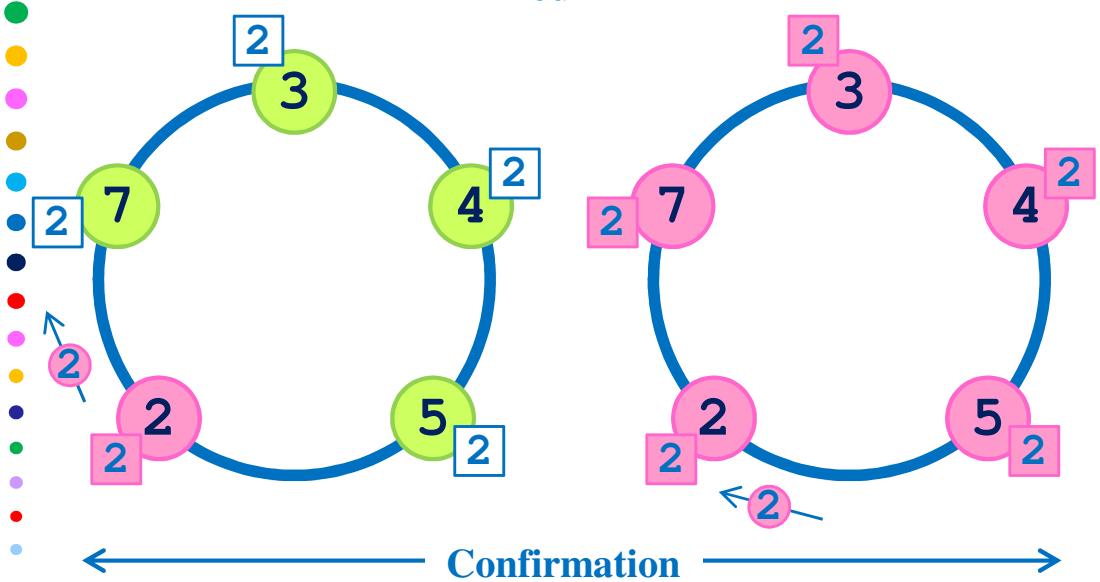
Ci-dessus l'émission des deux requêtes de l'unique initiateur et de leur retransmission jusqu'à leur émetteur qui en conclut qu'il est l'élu.

Ci-contre, l'émission du message de confirmation de l'élu qui termine le processus d'élection.

Scénario d'Exécution de l'Algorithme de Franklin

Scénario d'exécution : trois initiateurs



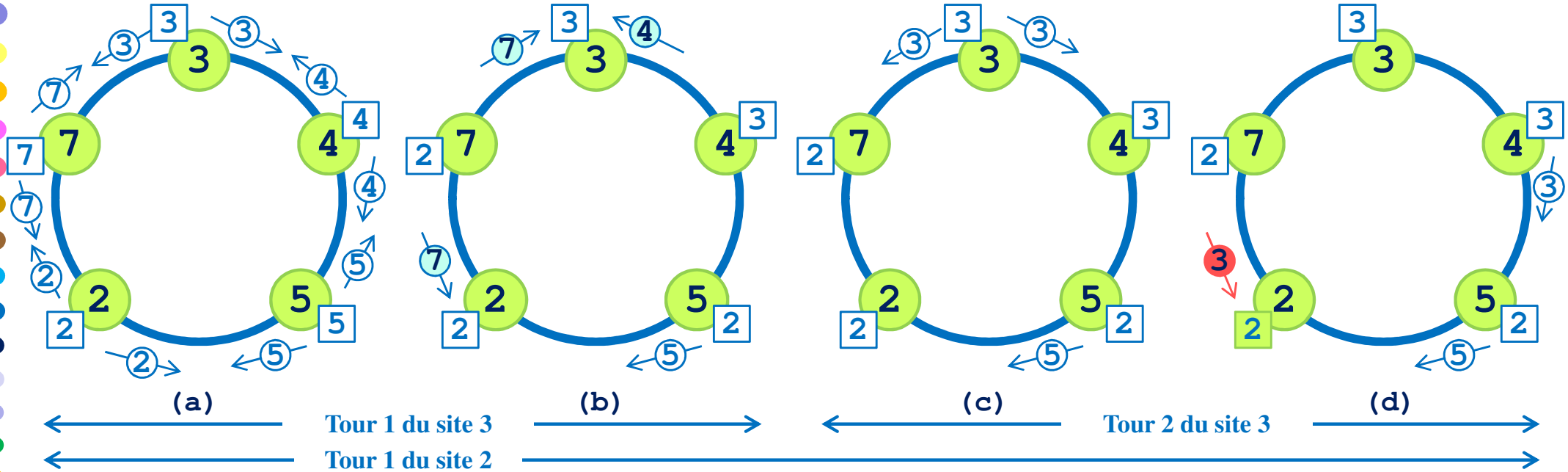


Ci-dessus :
Un initiateur qui reçoit deux requêtes avec une (au moins) identité inférieure est éliminé.
Un initiateur qui reçoit deux requêtes avec des identités supérieures participe au tour suivant.
Un initiateur qui reçoit une de ses requêtes est élu.

Ci-contre, l'émission du message de confirmation de l'élu qui termine le processus d'élection.

Scénario d'Exécution de l'Algorithme de Franklin

Scénario d'exécution : problème d'un site en avance d'un tour sur un candidat voisin



(a) tous les sites sont des initiateurs et envoient leurs deux requêtes (une à droite et l'autre à gauche).

(b) le site 3 reçoit les requêtes des candidats voisins, les sites 7 et 4. Il est d'identité inférieure, il survit donc et peut passer au tour suivant. Le site 2 reçoit uniquement la requête du site 7. Il ne peut donc passer au tour suivant. Les sites 7 et 4 sont éliminés.

(c) le site 3 débute son second tour en envoyant à nouveau deux requêtes. Le site 2 n'a toujours pas reçu la requête du site 5 et ne peut donc toujours pas passer au tour suivant.

(d) alors qu'il est dans son premier tour, le site 2 reçoit une deuxième requête du site 3 mais émise au second tour de ce dernier. Cette requête est une requête en avance et doit donc être conservée pour le tour suivant du site 2.

Un site ne peut recevoir qu'une requête en avance et il la reçoit dans la même direction que la première requête reçue.

Algorithme de Franklin

Variable du site **i** :

- **suivant**$_i$: constante contenant l'identité du site successeur de **i** sur l'anneau,
- **précédent**$_i$: constante contenant l'identité du site prédécesseur de **i** sur l'anneau,
- **état**$_i$: état du service. Cette variable prend une valeur parmi (**repos**, **en_cours**, **attente**, **terminé**). Cette variable est initialisée à **repos**,
- **nbreq**$_i$: nombre de requêtes reçues au cours d'un tour,
- **cand**$_i$: identité du site dont on reçoit la première des deux requêtes du tour courant. Lorsqu'elle vaut **i**, cela signifie qu'aucune requête n'est reçue,
- **dir**$_i$: booléen indiquant la direction d'où vient la première des deux requêtes du tour courant,
- **candav**$_i$: identité du site dont on reçoit la première des deux requêtes du tour suivant,
- **chef**$_i$: identité du site (provisoirement) élu.

Si le site récepteur est l'émetteur du message de confirmation reçu, alors il le supprime, sinon il le retransmet au suivant sur l'anneau.

```
sur_réception_de(j, (conf, k)){
    Si ( i != k ) Alors
        envoyer_à(suivant_i, (conf, k));
        état_i = terminé;
    Fsi
}
```

Algorithme de Franklin

Si aucun processus d'élection n'a atteint le site **i**, le service initialise un tel processus.

Il propose sa candidature dans les deux sens de l'anneau tant qu'il n'est pas éliminé ou qu'il ne sait pas qu'il est le seul survivant.

La boucle **répéter** correspond à la participation du site aux tours successifs
A chaque changement de tour, il faut prendre garde à traiter l'éventuelle requête en avance.

Dans tous les cas, il attend que le processus d'élection soit terminé pour renvoyer l'identité du site élu.

```
leader(){
    Si ( état_i == repos ) Alors
    |   état_i = en_cours; chef_i = i; candav_i = i;
    |   Répéter
    |   |   nbreq_i = 0;
    |   |   Si ( candav_i != i ) Alors
    |   |   |   nbreq_i = 1; cand_i = candav_i;
    |   |   |   Si ( cand_i < chef_i ) Alors chef_i = cand_i; Fsi
    |   |   |   candav_i = i;
    |   |   Fsi
    |   |   envoyer_à(suivant_i, (req, i));
    |   |   envoyer_à(précédent_i, (req, i));
    |   |   Attendre( nbreq_i == 2 )
    |   Jusqu'à ( état_i != en_cours );
    |   Si ( candav_i != i ) Alors
    |   |   Si ( dir_i ) Alors
    |   |   |   envoyer_à(suivant_i, (req, candav_i));
    |   |   Sinon
    |   |   |   envoyer_à(précédent_i, (req, candav_i));
    |   |   Fsi
    |   Fsi
    Fsi
    Attendre(état_i == terminé);
    renvoyer(chef_i);
}
```

Algorithme de Franklin

A la réception d'une requête, on met à jour si nécessaire l'identité provisoire du leader.

Dans le cas où on est un initiateur survivant (état à **`en_cours`**), on enregistre les deux requêtes voisines

…

en prenant garde à mémoriser une requête en avance.

Sur la réception de la deuxième requête, on teste si on a survécu

…

puis si on est le seul survivant.

Dans le cas où on est en **`attente`**, on retransmet les requêtes.

```
sur_réception_de(j, (req, k)){
    Si ( état_i == repos || k < chef_i ) Alors chef_i = k; Fsi
    Si ( état_i == en_cours ) Alors
        Si ( nbreq_i == 0 ) Alors
            cand_i = k; nbreq_i = 1; dir_i = ( j == précédent_i );
        Sinon  Si ((dir_i && j == précédent_i) ||
                   (!dir_i && j != précédent_i)) Alors
                candav_i = k;
               Sinon
                nbreq_i = 2;
                Si ( chef_i < i ) Alors
                    état_i = attente;
                Sinon Si ( cand_i == i || k == cand_i) Alors
                        état_i = terminé;
                        envoyer_à(suivant_i, (conf, i));
                      Fsi
                Fsi
               Fsi
        Fsi
    Sinon
        état_i = attente;
        Si ( j == précédent_i ) Alors
            envoyer_à(suivant_i, (req, k));
        Sinon
            envoyer_à(précédent_i, (req, k));
        Fsi
    Fsi
}
```

Références

Cours d'Algorithmique Répartie, Joyce El Haddad et Serge Haddad, Chapitres I à VIII, Université Paris-Dauphine.

E.J.-H. Chang, R. Roberts, "An Improved Algorithm for Decentralized Extrema Finding in Circular Arrangements of Processes", Communication of ACM vol 22 (1979), pp 281-283

W.R. Franklin, "On an Improved Algorithm for Decentralized Extrema Finding in Circular Configurations of Processors", Communications of ACM 25,5 (1982), pp 336-337