ELECTION

MU4IN403– Algorithmique Répartie

Plan

I. Introduction
   1. Principe
   2. Applications
   3. Impossibilité

II. Topologie en anneau

III. Topologie quelconque avec hypothèses

I.1 Principe

Objectif

Désigner un chef dans un ensemble de processus

Propriétés

1. Sûreté

   Le chef doit être unique

2. Vivacité

   La désignation doit se faire en un temps fini

I.2 Applications

- Mise en place d'un contrôle centralise (coordinateur, séquenceur, etc.)
- Allocation de ressources en exclusion mutuelle
- Recouvrement de défaillance
  - Restauration d'un maître dans un système maître/esclaves
  - Régénération de jeton perdu
  - Serveurs répliqués. Le serveur primaire est le leader.
- Consensus
- Résolution de situations de blocage

I.3 Impossibilité

[Angluin 1980]

Il n'existe pas d'algorithme déterministe d'élection de chef dans les réseaux anonymes et uniformes.

Idée de la preuve

1. Réseau anonyme => config. de départ peut être symétrique
2. Config. objectif (leader unique élu) est asymétrique
3. Il existe une exécution E du système telle que

   à partir d'une configuration symétrique

   on passe **toujours** dans une configuration symétrique

I.3 Impossibilité

Contourner l'impossibilité

- via des identifiants

  Relation d'ordre entre identifiants brise la symétrie

- via des algorithmes probabilistes
  - Tirages aléatoires sur chaque nœud
  - Tomber à l'infini sur des configurations symétriques est quasi-impossible
  - Algorithmes Las Vegas : algorithme probabiliste
    - S'arête toujours avec une réponse exacte.
    - Toutes les configurations terminales sont correctes.

Plan

I. Introduction

II. Topologie en anneau

1. Anneau unidirectionnel asynchrone (Chang-Roberts)
2. Anneau unidirectionnel synchrone anonyme (Itai-Rodeh)
3. Anneau bidirectionnel asynchrone (Hirschberg-Sinclair)

III. Topologie quelconque avec hypothèses

II.1 Unidirectionnel synchrone

[Chang & Roberts 1979] (adapté de LeLann77)

Hypothèses

- Les nœuds sont capables de s'organiser en anneau unidirectionnel
- Chaque nœud possède un identifiant unique dans l'anneau
- Chaque nœud i dispose d'une référence succ[i] vers son successeur
- Aucun nœud ne connaît la taille de l'anneau
- Plusieurs candidats simultanés possibles
- Communications fiables assynchrones FIFO
- Exécution sans faute

Idée

- Chaque candidat propage sa candidature dans l'anneau
- Un candidat qui reçoit une candidature supérieure a perdu
- Le processus candidat d'identifiant maximal reçoit sa propre candidature

II.1 Unidirectionnel synchrone

[Chang & Roberts 1979] - Algorithme

Variables locales

Etat – non-candidat/candidat/élu/perdu, initialisé à non-candidat

leader – identité du leader, initialisé à NULL

succ[i] – successeur de i dans l'anneau

Deux types de message

ELEC et LEADER

Site i se porte candidat

```
Proposer_candidature ( ){
    Etat= candidat
    leader = i
    envoyer(<ELEC, i>) à succ[i]
}
```

II.1 Unidirectionnel synchrone

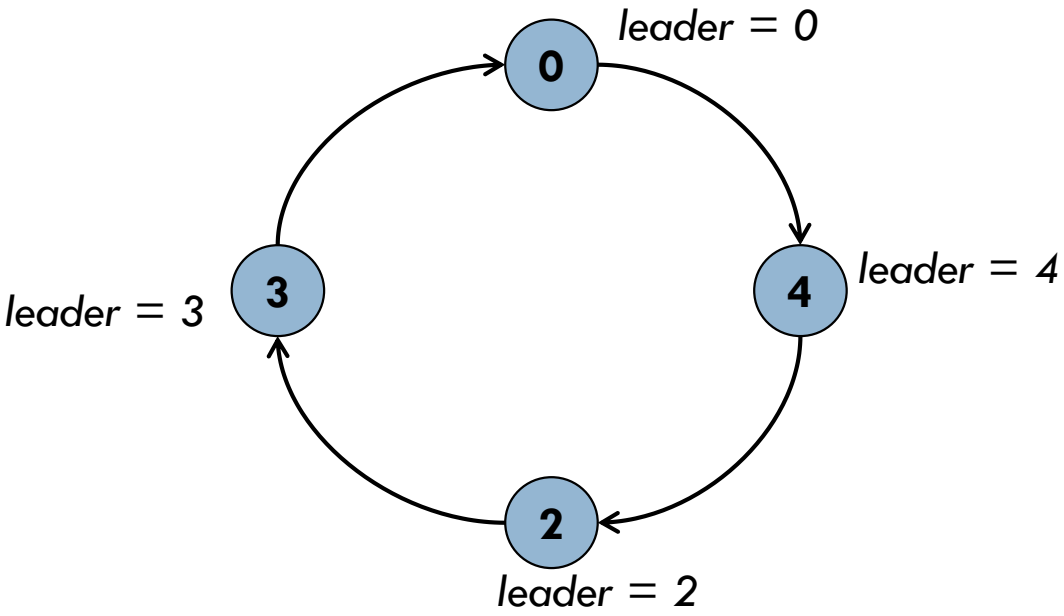
[Chang & Roberts 1979] – Algorithme (suite)

Site i reçoit <ELEC, j>

```
if i > j then
    if Etat = non-candidat then
        Proposer_candidature ( )
else if i < j then
    Etat = perdu
    leader = j
    envoyer  (<ELEC, j>) à succ[i]
else if i = j then
    etat = elu
    envoyer  (<LEADER, i>) à succ[i]
```

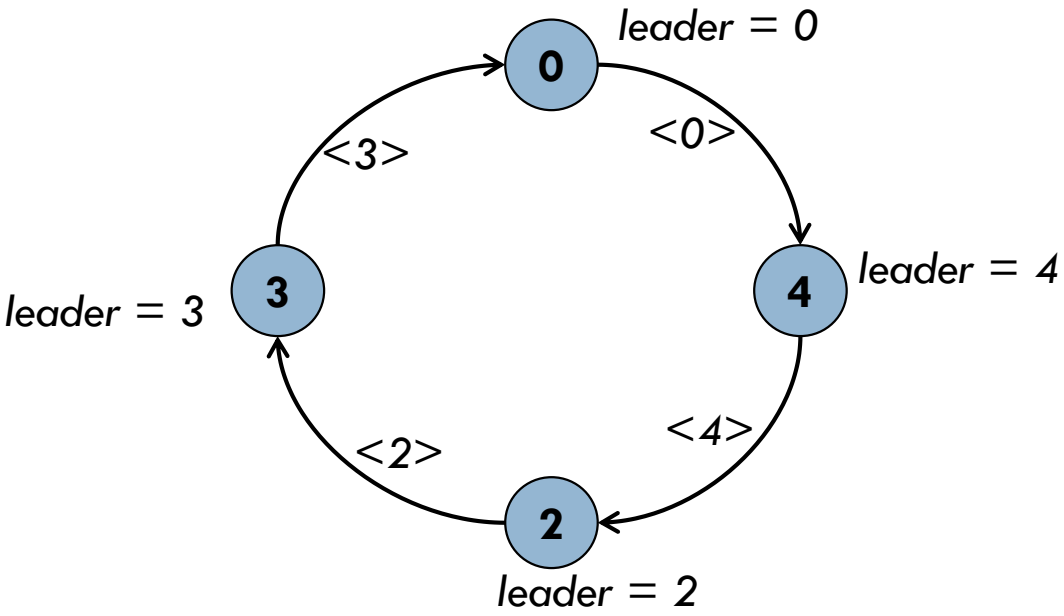
II.1 Unidirectionnel synchrone

[Chang & Roberts 1979] - Exemple



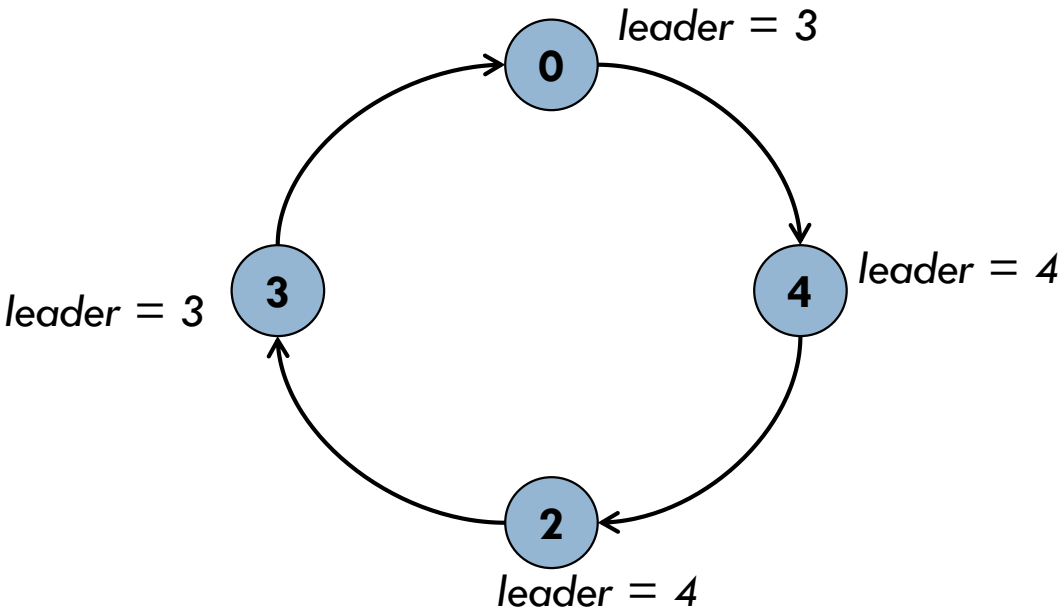
II.1 Unidirectionnel synchrone

[Chang & Roberts 1979] - Exemple



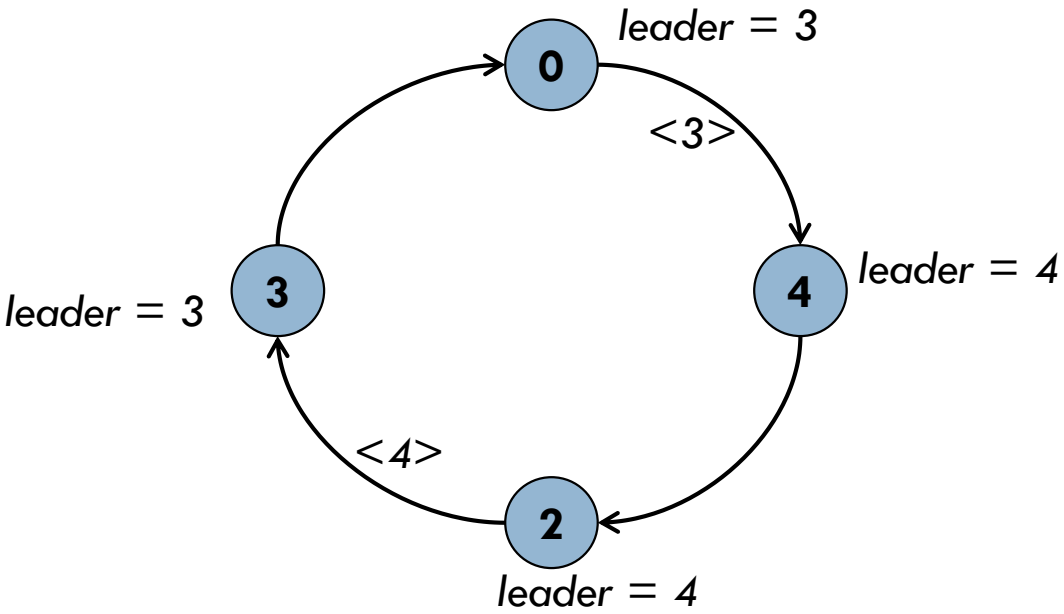
II.1 Unidirectionnel synchrone

[Chang & Roberts 1979] - Exemple



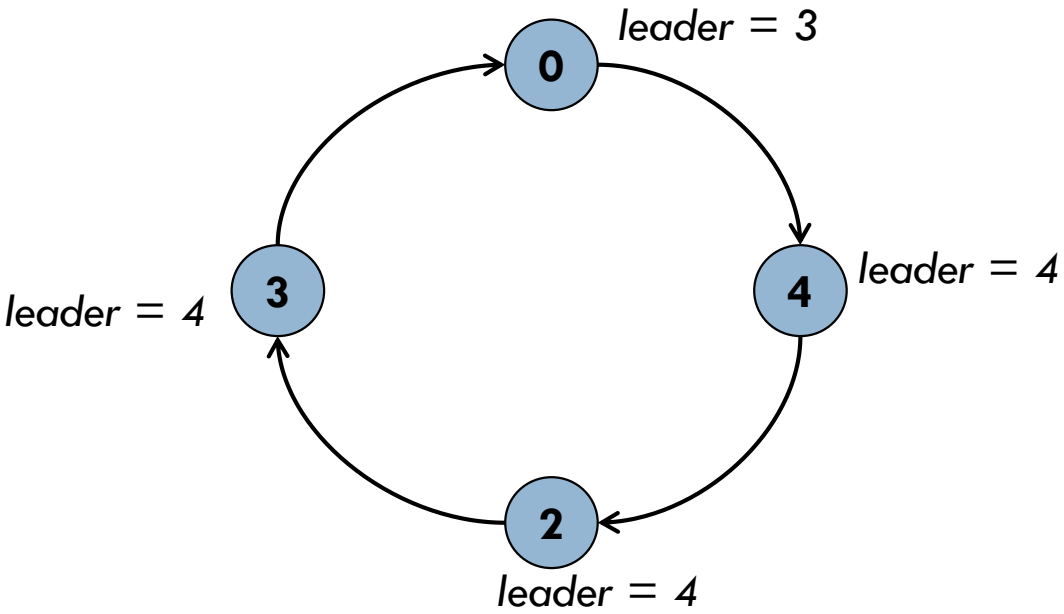
II.1 Unidirectionnel synchrone

[Chang & Roberts 1979] - Exemple



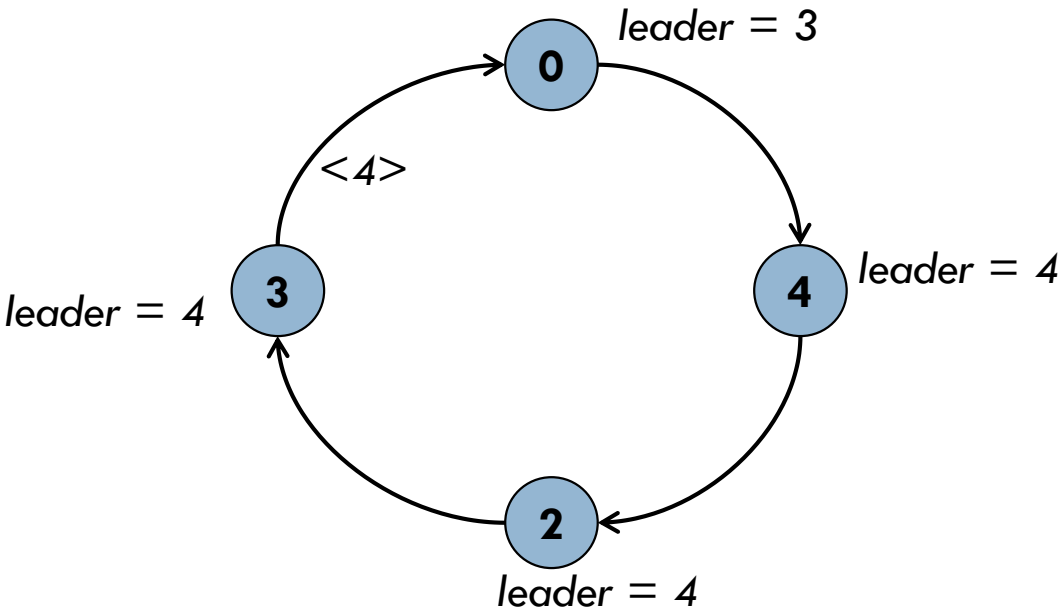
II.1 Unidirectionnel synchrone

[Chang & Roberts 1979] - Exemple



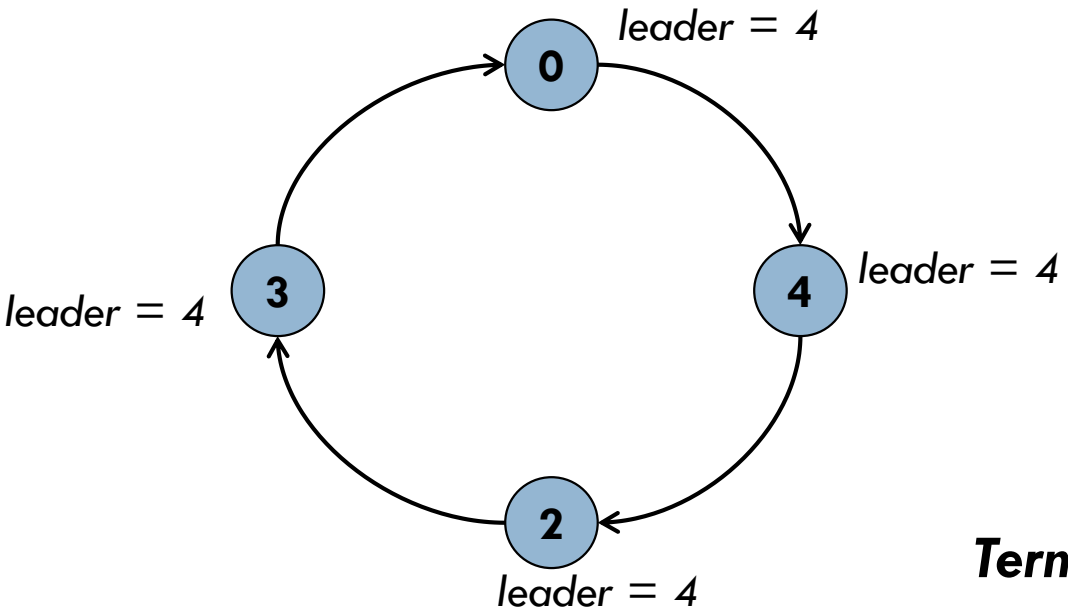
II.1 Unidirectionnel synchrone

[Chang & Roberts 1979] - Exemple



II.1 Unidirectionnel synchrone

[Chang & Roberts 1979] - Exemple



Terminaison

Election terminée, le nouveau leader fait l'announce : message LEADER avec son ID

II.1 Unidirectionnel synchrone

[Chang & Roberts 1979] – Complexité

Complexité en temps : O(N)

Complexité en memoire : O(log N)

II.1 Unidirectionnel synchrone

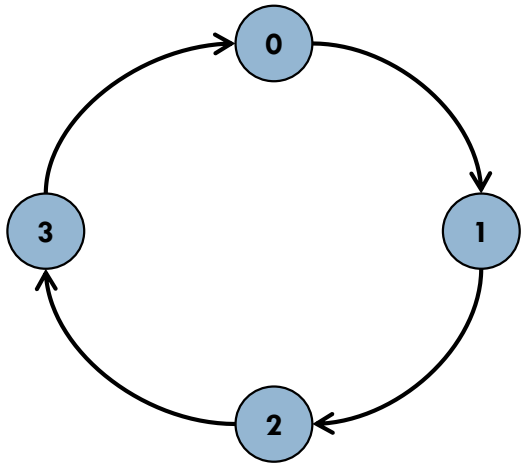
[Chang & Roberts 1979] – Complexité

Complexité en temps : O(N)

Complexité en memoire : O(log N)

Complexité en messages :

Meilleur des cas : O(N)



II.1 Unidirectionnel synchrone

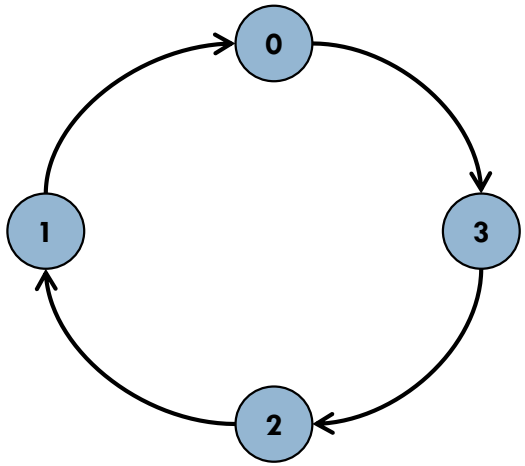
[Chang & Roberts 1979] – Complexité

Complexité en temps : O(N)

Complexité en memoire : O(log N)

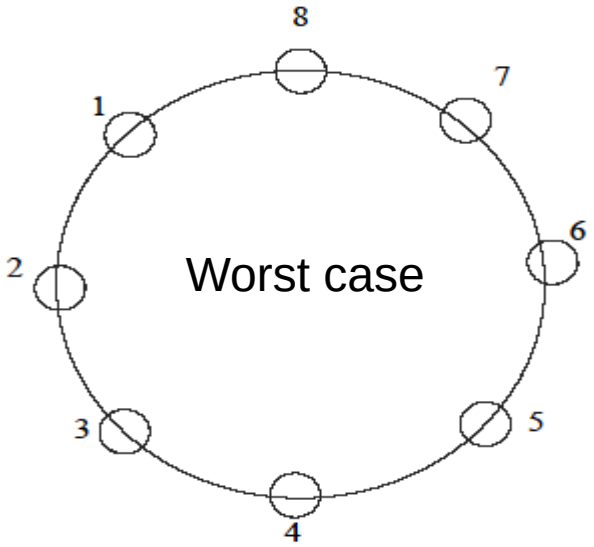
Complexité en messages :

- Meilleur des cas : O(N)
- Pire des cas : $O(N^2)$



II.1 Unidirectionnel synchrone

- **Pire cas complexité de message**
  - tous les processus se portent candidats



$$\sum_{j=1}^{j=N} j = N(1+N)/2 = O(N^2)$$

Message *ELEC* de 1 fait 1 saut; message ELEC de 2 fait 2 sauts, …

II.2 Unidirectionnel synchrone anonyme

[Itai & Rodeh 81]

Basé sur l'algorithme de Chang et Roberts

Hypothèses

- Les nœuds sont capables de s'organiser en anneau unidirectionnel
- ~~Chaque nœud possède un identifiant unique dans l'anneau~~
- Chaque nœud i dispose d'une référence succ[i] vers son successeur
- ***Chaque nœud connaît la taille N de l'anneau***
- Plusieurs candidats simultanés possibles
- Communications fiables et synchrones
- Exécution sans faute

II.2 Unidirectionnel synchrone anonyme

[Itai & Rodeh 81]

Initialement, tous les processus sont actifs

Chaque processus maintient etat (intit. actif) et tournoi (init. 1)

Fonctionnement en tournois entre nœuds actifs

Chaque nœud actif tire un identifiant id aléatoirement dans [1,k] ($k \geq N$) puis envoie (id,1,1,vrai) au successeur

Lorsqu'un processus p reçoit (#id, #tournoi, #saut, unique)

Si p est inactive, alors il envoit (#id, #tournoi, #saut+1, unique) au successeur

sinon // p est active alors

si #saut = N alors //je suis l'initiateur du tournoi

alors si unique alors etat = élu

sinon tournoi = tournoi + 1

p tire un nouvel id aléatoirement dans [1,k] puis envoie (id,tournoi,1,vrai) au successeur

sinon //#saut ≠ N

si (id,tournoi) $=_{lex}$ (#id,#tournoi) alors p envoie (#id, tournoi, #saut+1, faux) au successeur

si (id,tournoi) $>_{lex}$ (#id,#tournoi) alors

etat = inactive;

p envoie (#id, tournoi, #saut+1, unique) au successeur

II.2 Unidirectionnel synchrone anonyme

[Itai & Rodeh 81] - Analyse

C'est un algorithme Las Vegas

- Termine toujours

Cas moyen

- Forte probabilité d'un seul candidat à chaque phase
- Complexité en nombre de messages est $O(N^2)$

II.3 Bidirectionnel asynchrone

[Hirschberg & Sinclair 1980]

Hypothèses

- Les nœuds sont capables de s'organiser en anneau bidirectionnel
- Chaque nœud possède
  - un identifiant unique dans l'anneau
  - une référence succ[i] vers son successeur
  - une référence pred[i] vers son prédécesseur
- Aucun nœud ne connaît la taille de l'anneau
- Plusieurs candidats simultanés possibles
- Communications fiables et asynchrones
- Exécution sans faute

II.3 Bidirectionnel asynchrone

[Hirschberg & Sinclair 1980] - Idée

- Fonctionnement en rondes asynchrones
  - Les nœuds actifs sont candidats.
    - Un noeud *actif* émet son identifiant dans les 2 directions sur des distances croissantes
      - Ronde k: distance 2^k , k=0,1,2,…
    - Le vainqueur de chaque ronde est celui qui a transmis sur toute la distance
      - Il continue *actif*. Les autres passent à l'état *inactif*.
  - Un identifiant qui fait le tour de l'anneau désigne le leader
- complexité en messages : O(N log N)
  - log N = nombre de rondes
- Sera étudier en TD

II.3 Bidirectionnel asynchrone

[Hirschberg & Sinclair 1980] - Algorithme

Valeurs locales

state = active

leader = local_id

round = 0

Début de ronde k (init. à 0)

if state = active

envoyer <leader, 2^k> à voisins de gauche et droite

attendre retour des deux messages

k++ ; cpt =0

II.3 Bidirectionnel asynchrone

[Hirschberg & Sinclair 1980] – Idée de l'algorithme

Sur réception de <j, TTL> venant de voisin gauche (droite)

If j = i then

if TTL > 0 then **se déclarer vainqueur**

if j > i and TTL ≥ 1 then

state = inactive

leader = j

if TTL > 1 then

envoyer <j, TTL - 1> a voisin droite (gauche)

else

envoyer <j, 0> a voisin gauche (droit)

if TTL = 0 then

If (j !=i)

envoyer <j, 0> a voisin droite (gauche)

else

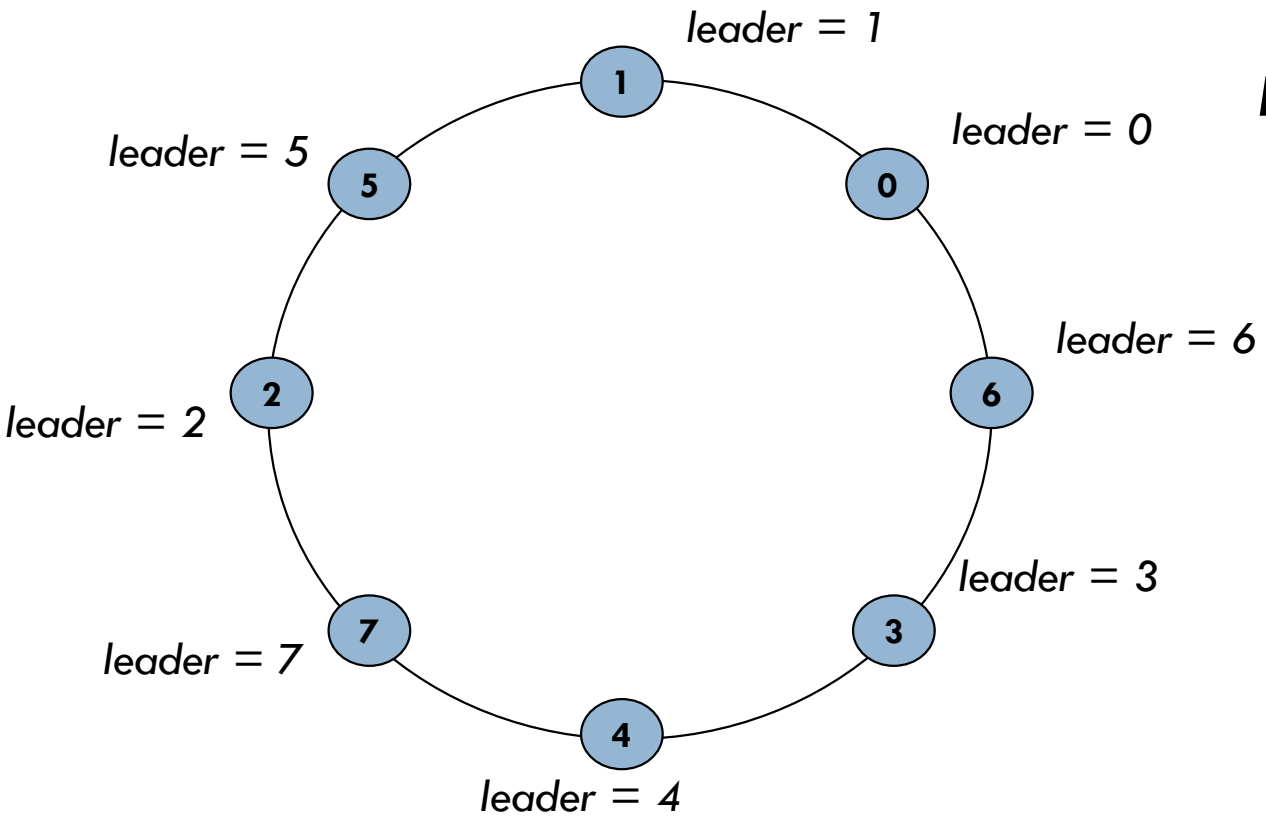
cpt++

if (cpt =2)

début de nouvelle étape

II.3 Bidirectionnel asynchrone

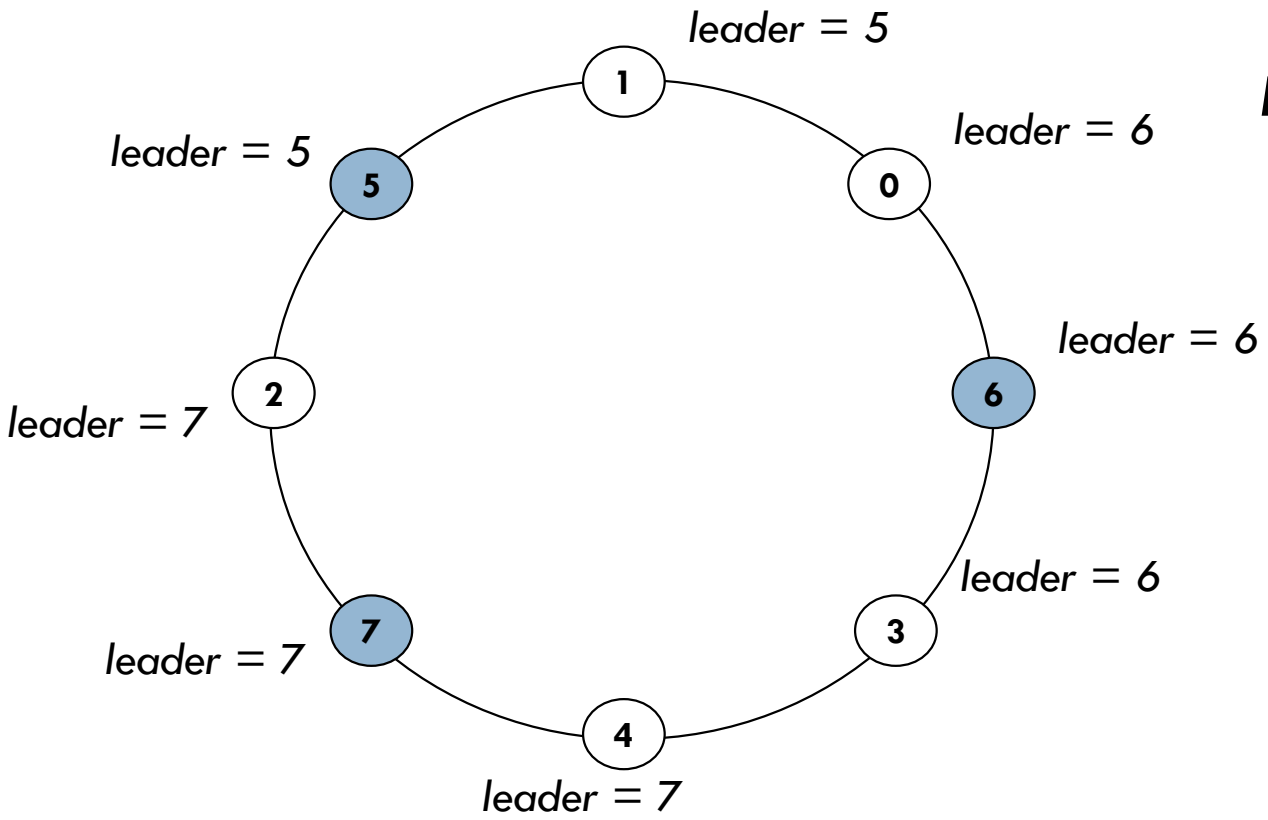
[Hirschberg & Sinclair 1980] – Exemple



Ronde 0 – init

II.3 Bidirectionnel asynchrone

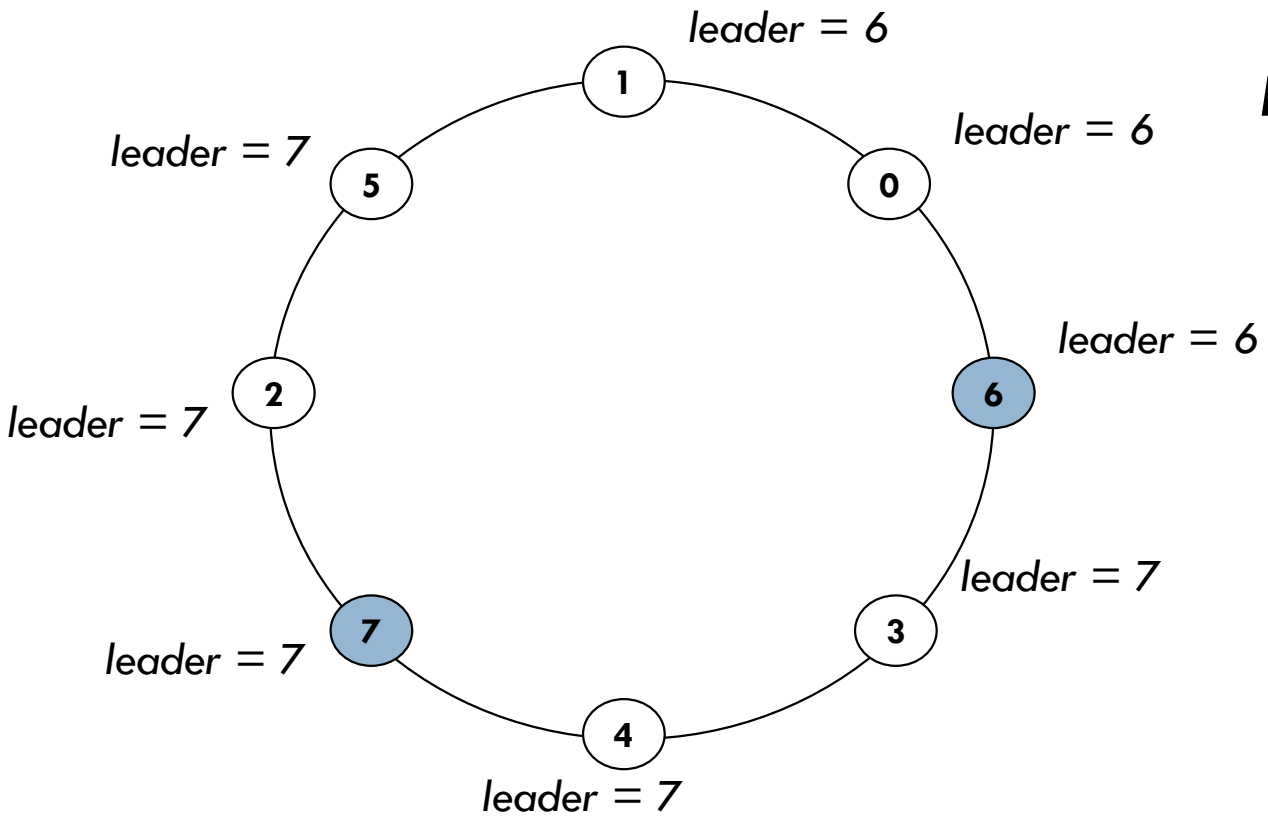
[Hirschberg & Sinclair 1980] – Exemple



Ronde 1 – init

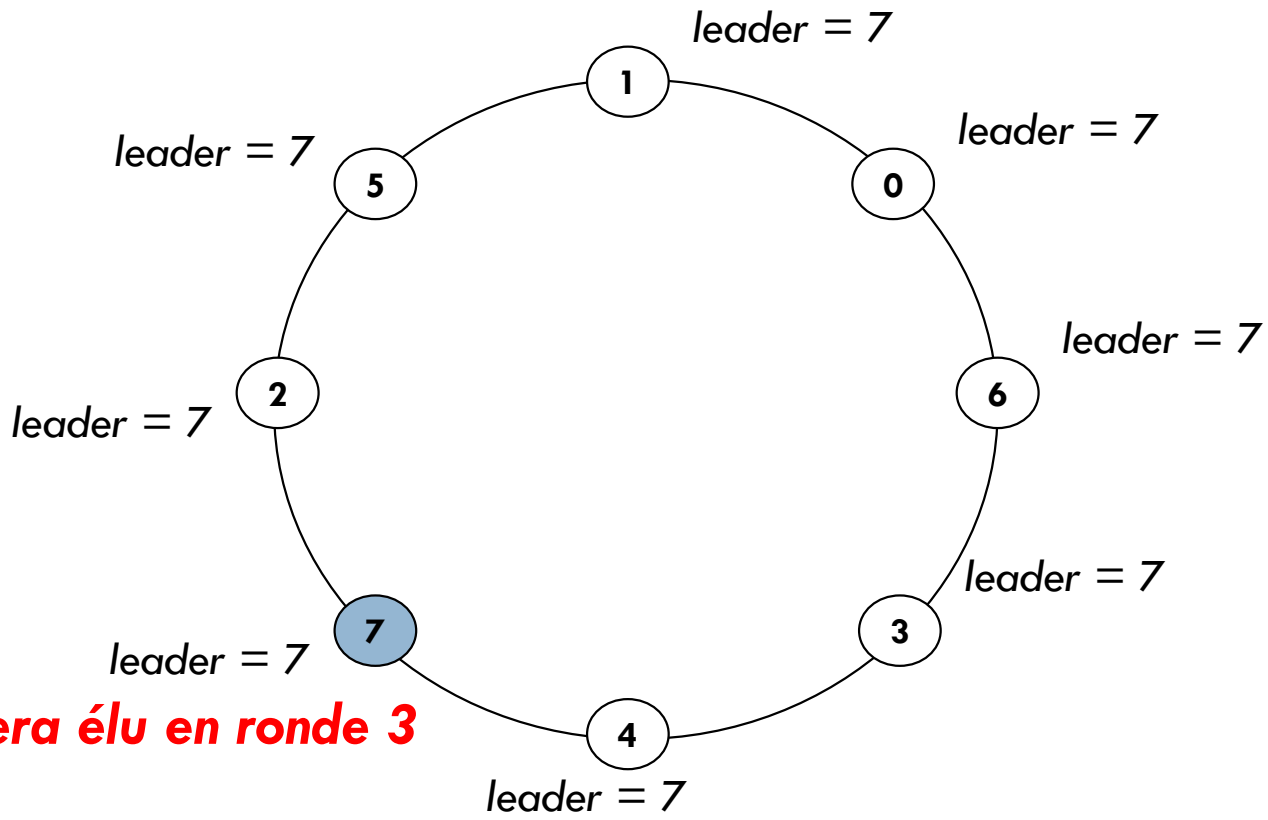
II.3 Bidirectionnel asynchrone

[Hirschberg & Sinclair 1980] – Exemple



II.3 Bidirectionnel asynchrone

[Hirschberg & Sinclair 1980] – Exemple



Ronde 2 – fin

N_7 sera élu en ronde 3

Plan

I. Introduction

II. Topologie en anneau

III. Topologie quelconque avec hypothèses

1. Diffusion avec identifiants
2. Diffusion anonyme

III.2 Diffusion avec identifiants

Calcul de maximum

Hypothèses

- Processus avec identifiants distincts
- Taille du système connue de tous
- Accès à un broadcast
- Communications fiables et synchrones

Idée

Diffusion de son identité et attente des identités de tous les autres

Le processus d'identifiant max devient le chef

Complexité en nombre de messages est O(M) dans tous les cas

M = nombre de liens

[O(NM) si initiateurs multiples -> $O(N^3)$]

III.2 Diffusion avec identifiants

FloodMax - Calcul de maximum par propagation

Algortihme par Vague (Total) de Phase

Hypothèses

- Processus avec identifiants distincts
- Diamètre du réseau D connu de tous
- Communications fiables et synchrones

Idée

Valeur *leader* initialisée avec l'identifiant local du processus

D phases

Chaque processus diffuse sa valeur *leader* à ses voisins

Sur réception de *leader'*, *leader* = max(*leader*, *leader'*)

Après D phases, tout le monde connaît la valeur maximale pour *leader*

Complexité en nombre de messages est O(D.M) avec M le nombre de liens

III.2 Diffusion avec identifiants

Bully [Garcia-Molina 1982]

Hypothèses

- Processus avec identifiants distincts
- Graphe complet
- Communications fiables et synchrones
- Pannes franches
- Trois types de message : ELEC, OK, LEADER

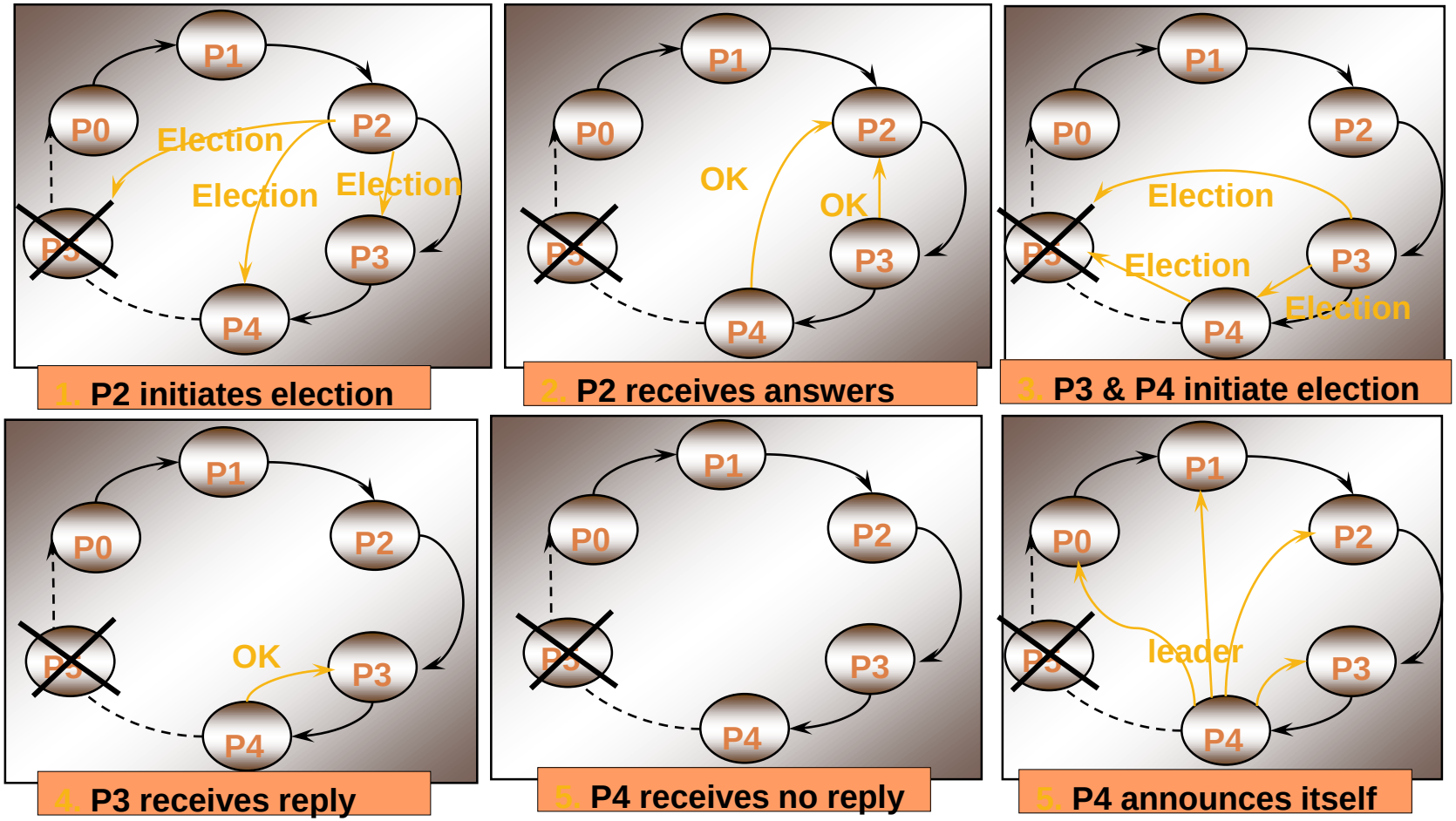
Idée

- Chaque candidat *i* envoie un message ELEC avec son identité aux processus dont l'identité est supérieure à la sienne.
  - Si après un délai (timeout) *i* n'a pas reçu aucune réponse, il se déclare le chef.
  - Il envoie alors un message LEADER à tous les processus dont l'identité est inférieure à la sienne.
- En recevant ELEC un site répond avec le message OK et dévient candidat, s'il n'en est pas encore.

Complexité en nombre de messages est $O(N^2)$ dans le pire des cas

Si tous les N-1 sites se portent candidat.

III.2 Diffusion avec identifiants



III.3 Diffusion anonyme

Candidature aléatoire

Hypothèses

- Nombre total de nœuds connu de tous
- Accès à un broadcast
- Communications fiables et **synchrones**
- Pannes franches

Idée

Algorithme par phases, avec tous les nœuds initialement actifs

1. Chaque nœud *actif* décide aléatoirement s'il candidate
2. Un nœud *actif* diffuse sa (non-)candidature et attend des réponses
3. Si un seul processus candidate il devient le chef
4. Sinon phase suivante

Complexité en nombre de messages est M(N log N) dans tous les cas

III.3 Diffusion anonyme

Candidature aléatoire

Variables locales

State - Le site est à l'état *active*, *inactive* ou *leader*

Values [N] - Tableau contenant les valeurs émises par tous les sites lors d'une phase

Algorithme

```
State = actif
Repeat
    Values = 0;
    b = random{0,1}
    broadcast(b)
    while (timer != 0) do
        receive(<b'>) from j
        Values [j] = b'
    if (b = 1 ∧ Values = 0) then State = leader
    if (b = 0 ∧ Values > 0) then State = inactive
until (State != active)
```

Biliographie

- E. Chang, R.Roberts. *An improved algorithm for decentralized extrema-finding in circular configurations of processes*, Communications of the ACM, ACM, 22 (5): 281–283, 1979.
- H. Garcia-Molina, *Elections in a Distributed Computing System*, IEEE Transactions on Computers, Vol. C-31, No. 1, pages 48–59,1982.
- D. Hisrchberg and J. Sinclair. *Decentralized extrema-finding in circular configurations of processors*. Communications of the ACM, 23 (11): 627–628, 1980.
- A. Itai and M. Rodeh. Symmetry breaking in distributive networks. In Proc. *22nd Annual Symposium on Foundations of Computer Science*, FOCS'81, pp. 150{158. IEEE Computer Society, 1981.