ELECTION

4I403 – Algorithmique Répartie

UPMC SORBONNE UNIVERSITÉS

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

Orchestration

- Mise en place d'un contrôle centralisé
- Allocation de ressources en exclusion mutuelle
- Accès à des données répliquées
- Consensus
- Résolution de situations de blocage

I.2 Applications

Recouvrement de défaillance

- Restauration d'un maître dans un système maître/esclaves
- Régénération de jeton perdu

Agrégation d'informations

- Calcul de la taille d'un réseau
- Détection de terminaison d'un algorithme

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 asymetrique
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 configs symétriques est quasi-impossible

  Algorithmes Las Vegas

    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

Site i se porte candidat

Etat= candidat

leader = i

envoyer(<ELEC, i>) à succ[i]

II.1 Unidirectionnel synchrone

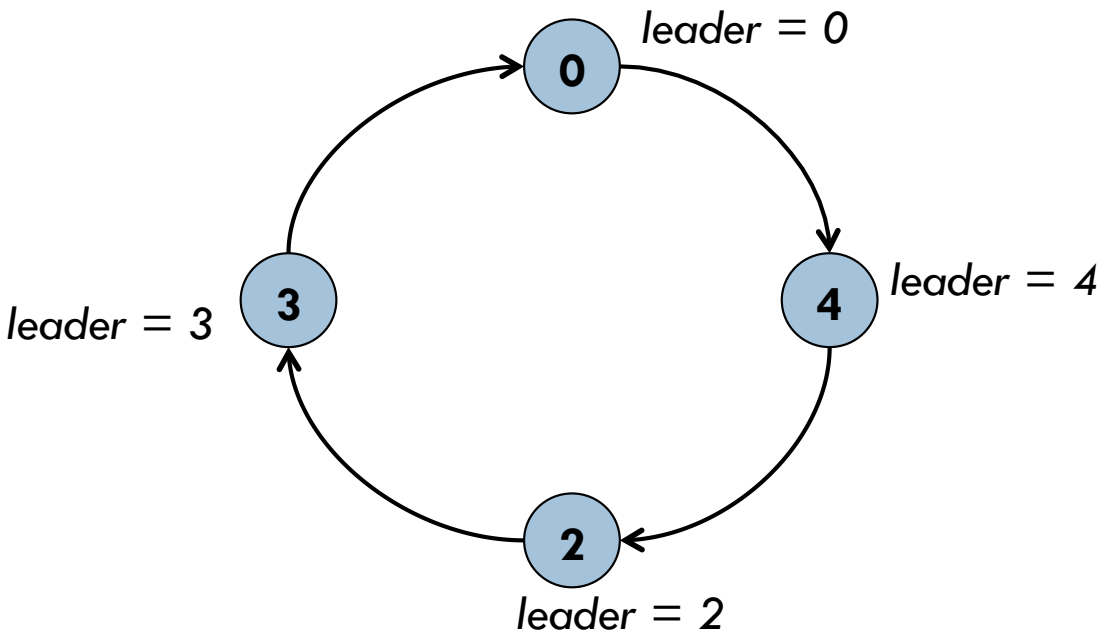
[Chang & Roberts 1979] – Algorithme (suite)

Site i reçoit <ELEC, j>

```
if i > j then
    if Etat = non-candidat then
        Site i se porte candidat
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]
```

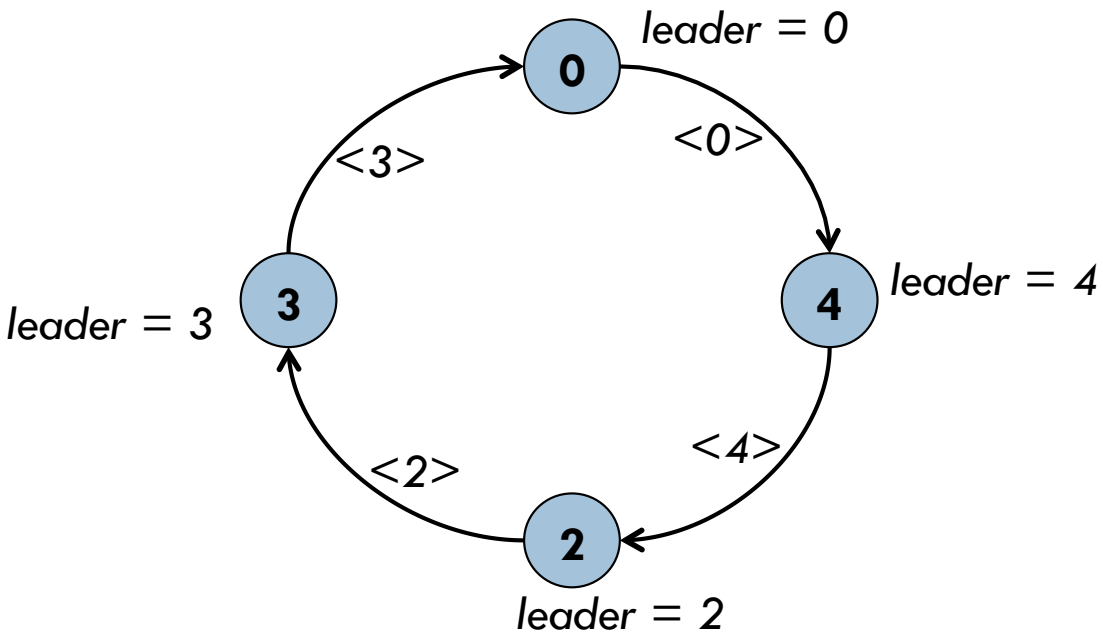
II.1 Unidirectionnel synchrone

[Chang & Roberts 1979] - Exemple



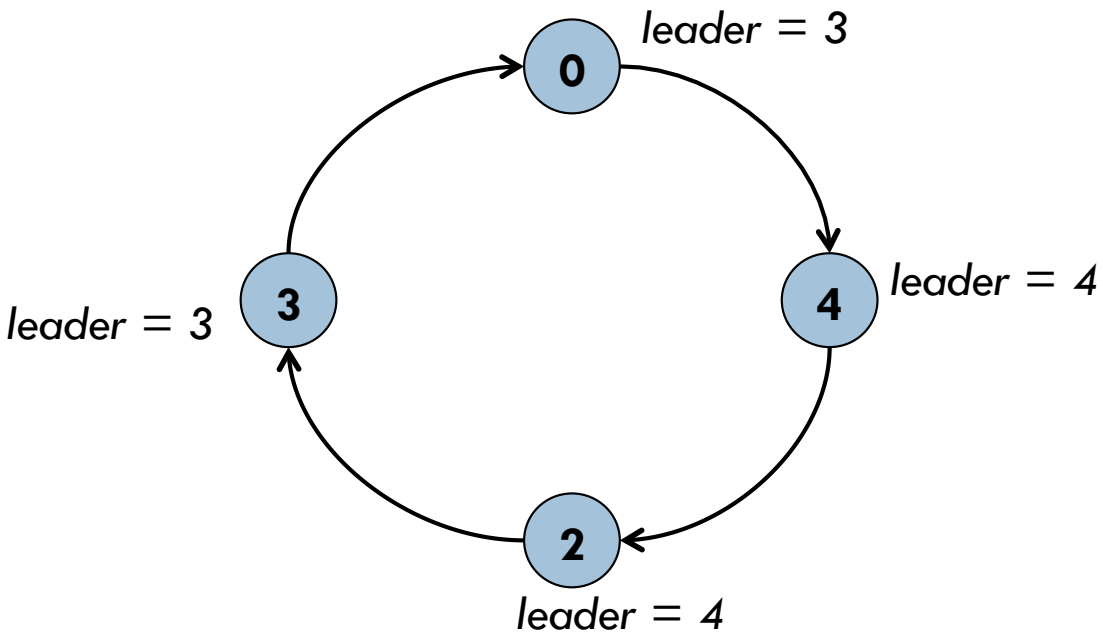
II.1 Unidirectionnel synchrone

[Chang & Roberts 1979] - Exemple



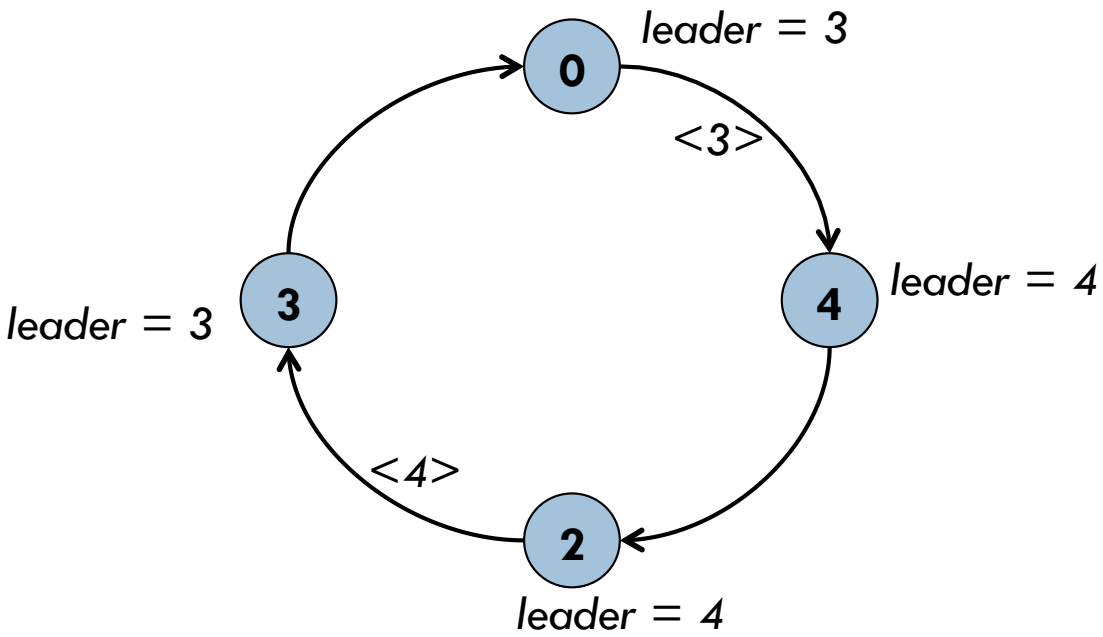
II.1 Unidirectionnel synchrone

[Chang & Roberts 1979] - Exemple



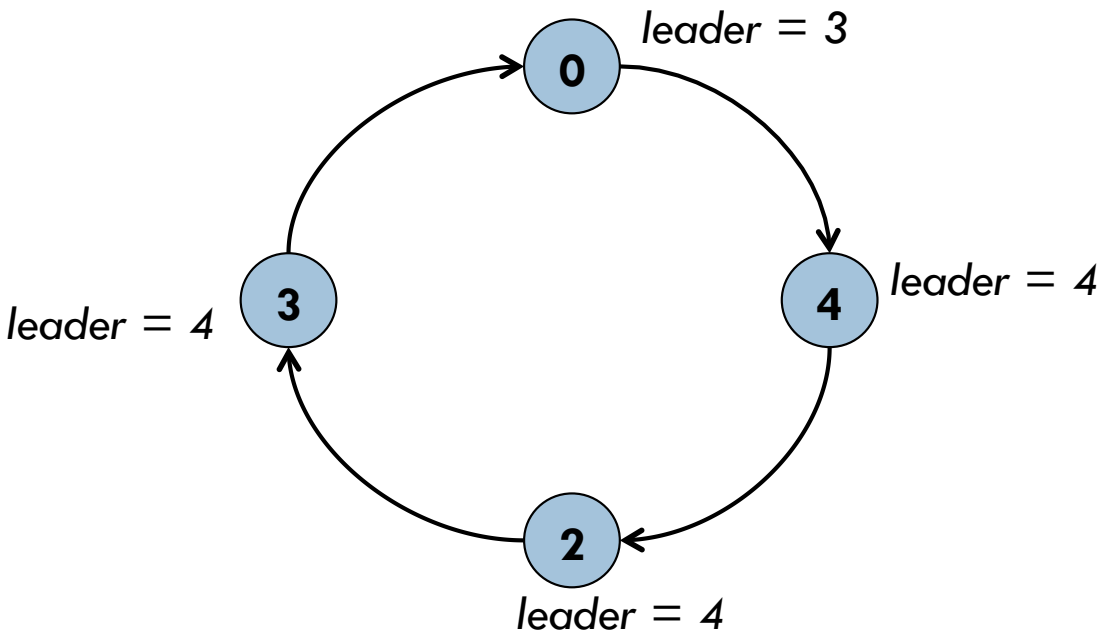
II.1 Unidirectionnel synchrone

[Chang & Roberts 1979] - Exemple



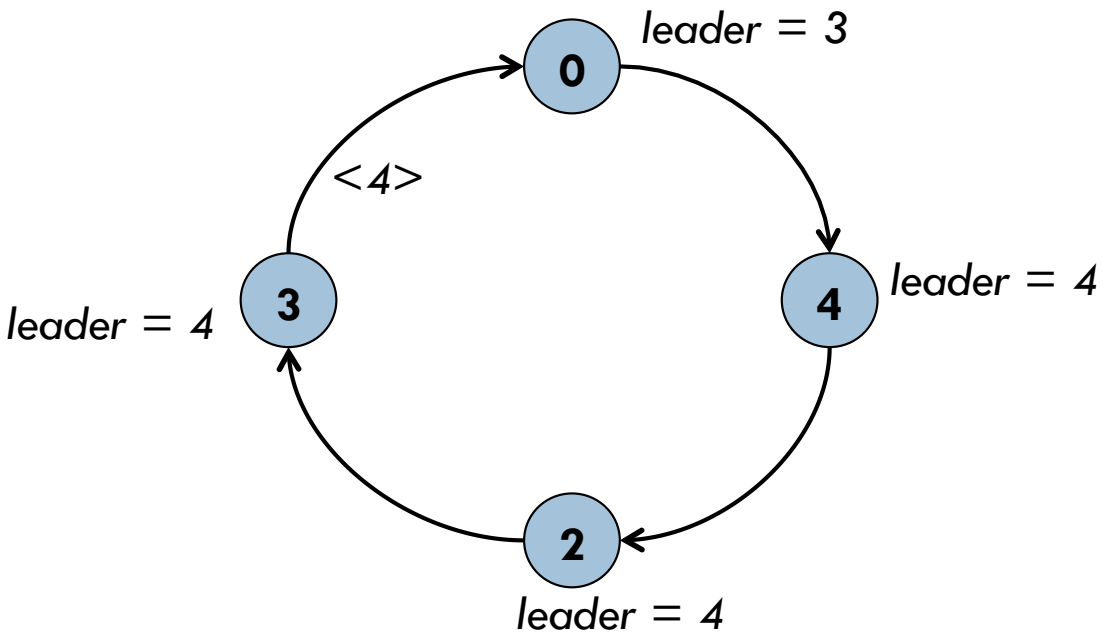
II.1 Unidirectionnel synchrone

[Chang & Roberts 1979] - Exemple



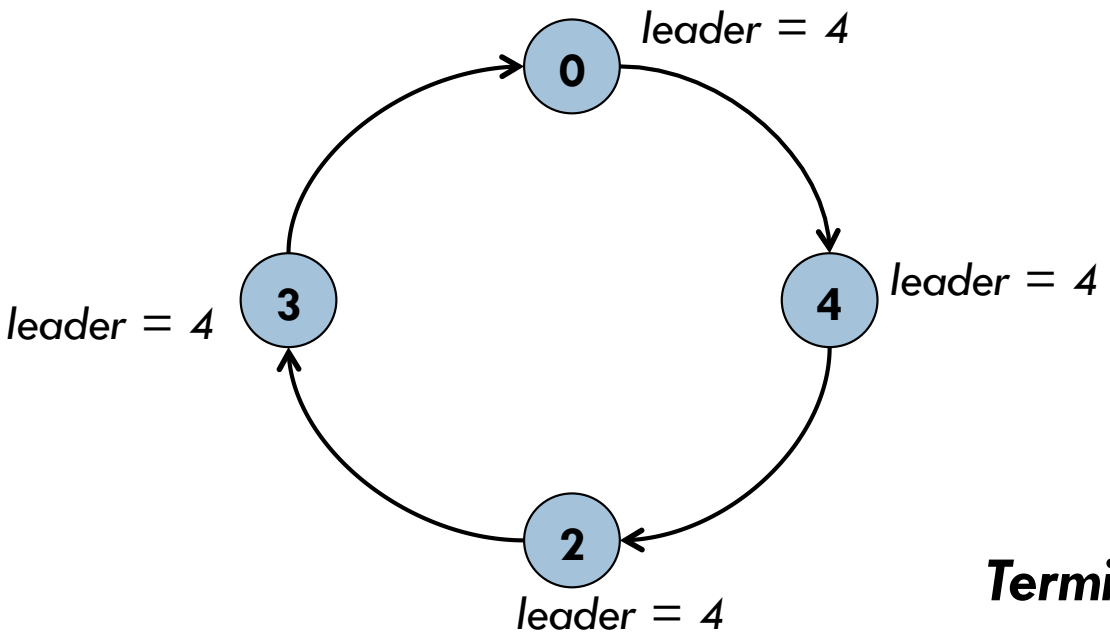
II.1 Unidirectionnel synchrone

[Chang & Roberts 1979] - Exemple



II.1 Unidirectionnel synchrone

[Chang & Roberts 1979] - Exemple



Terminaison ?

II.1 Unidirectionnel synchrone

[Chang & Roberts 1979] – Complexité

Complexité en temps : O(N)

Complexité en memoire : O(log N)

II.1 Unidirectionnel synchrone

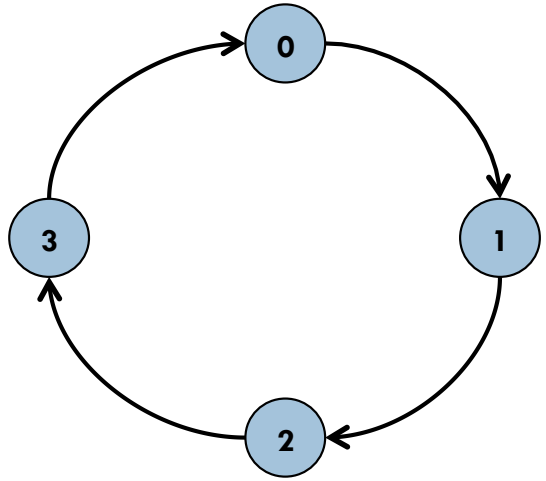
[Chang & Roberts 1979] – Complexité

Complexité en temps : O(N)

Complexité en memoire : O(log N)

Complexité en messages :

Meilleur des cas : O(N)



II.1 Unidirectionnel synchrone

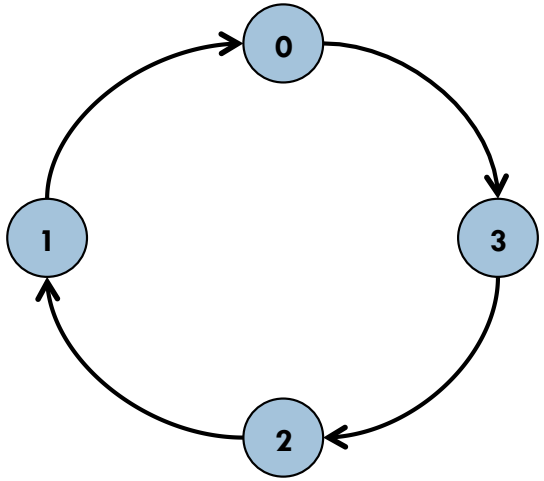
[Chang & Roberts 1979] – Complexité

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.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

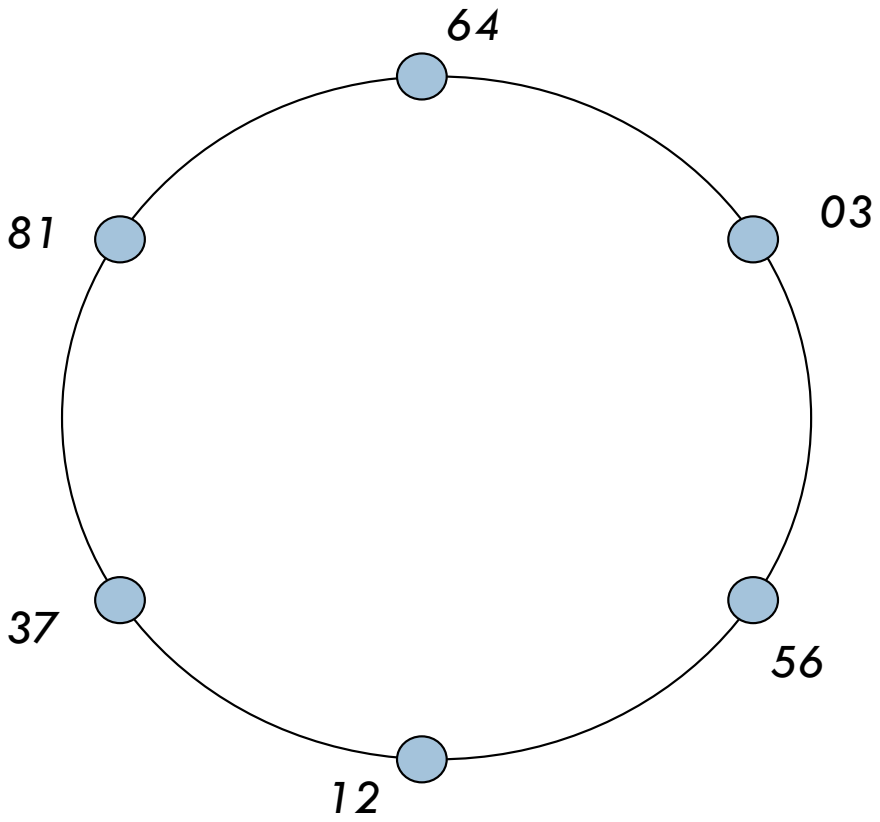
si (id,tournoi) $>_{lex}$ (#id,#tournoi) alors

etat = passif;

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

II.2 Unidirectionnel synchrone anonyme

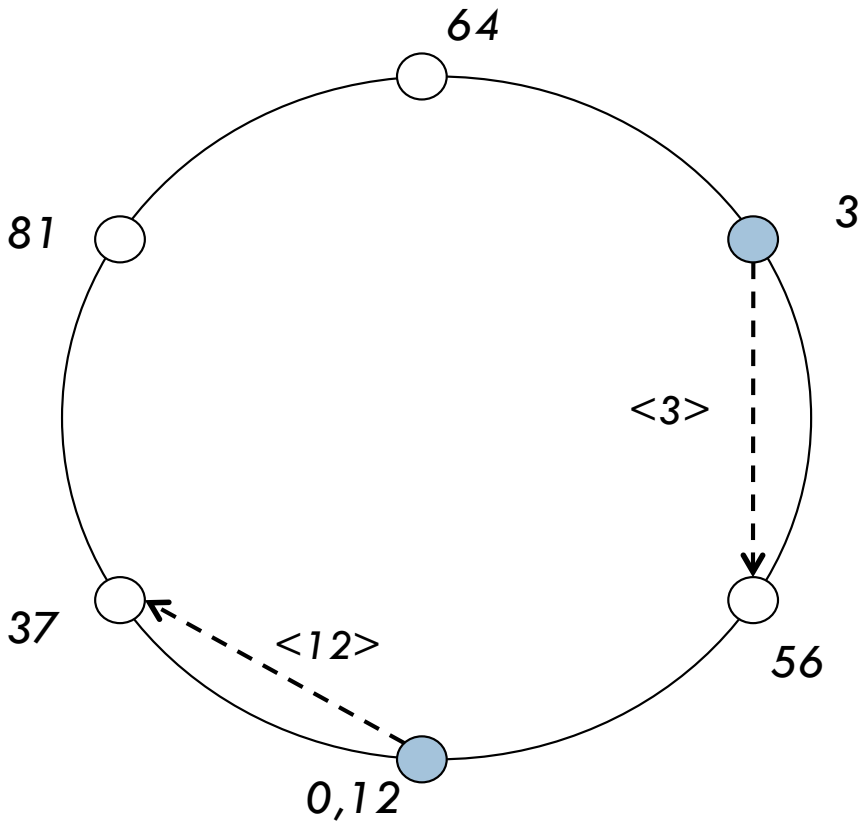
[Itai & Rodeh 81] - Exemple



Tournoi 1

II.2 Unidirectionnel synchrone anonyme

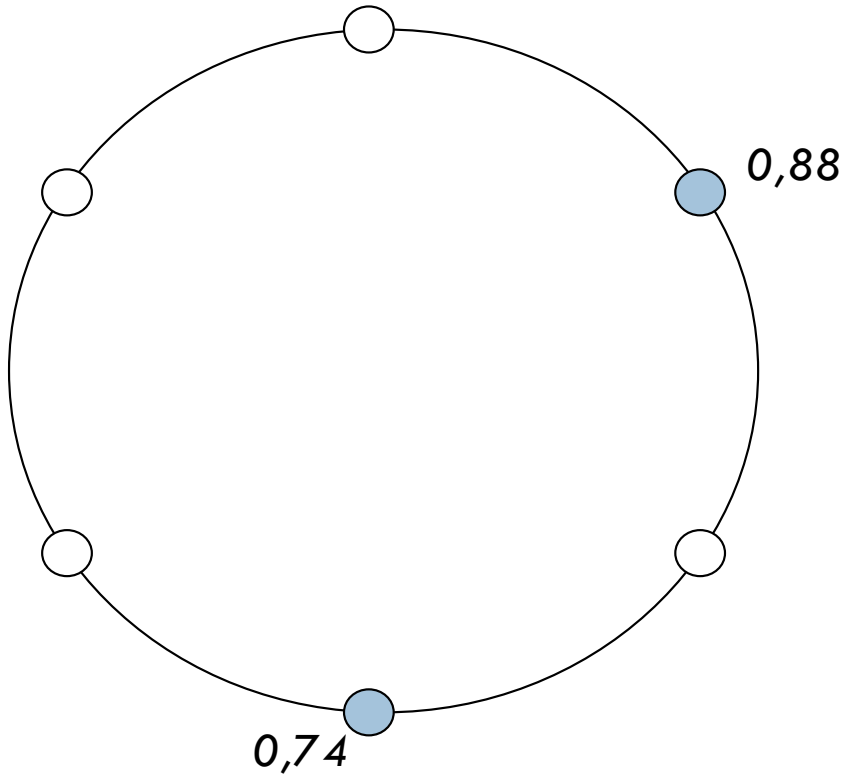
[Itai & Rodeh 81] - Exemple



Tournoi 1, round 1

II.2 Unidirectionnel synchrone anonyme

[Itai & Rodeh 81] - Exemple

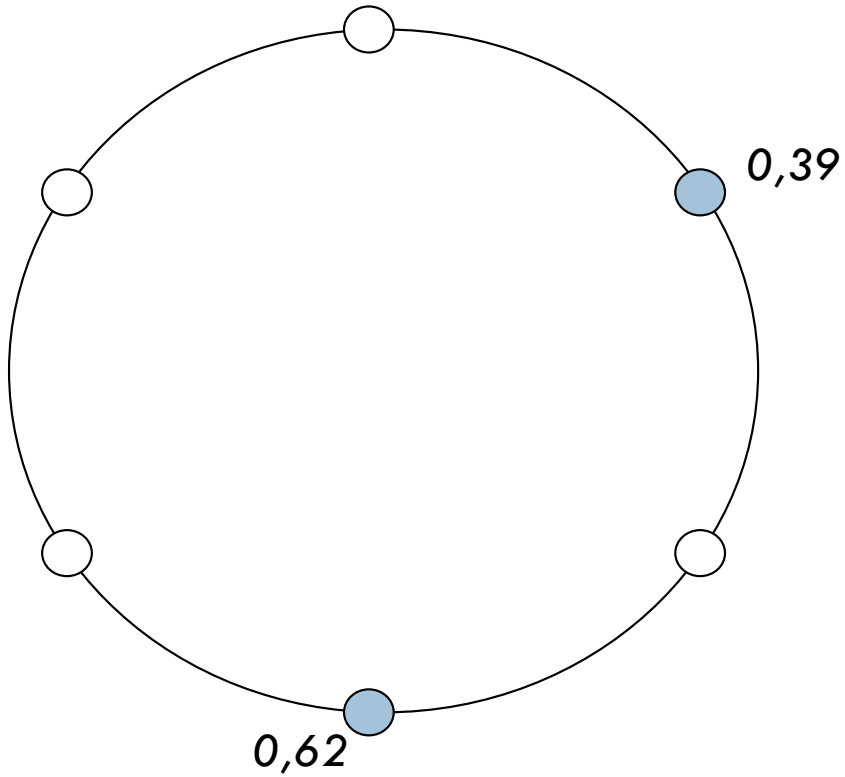


Tournoi 1

Phase 2 inutile

II.2 Unidirectionnel synchrone anonyme

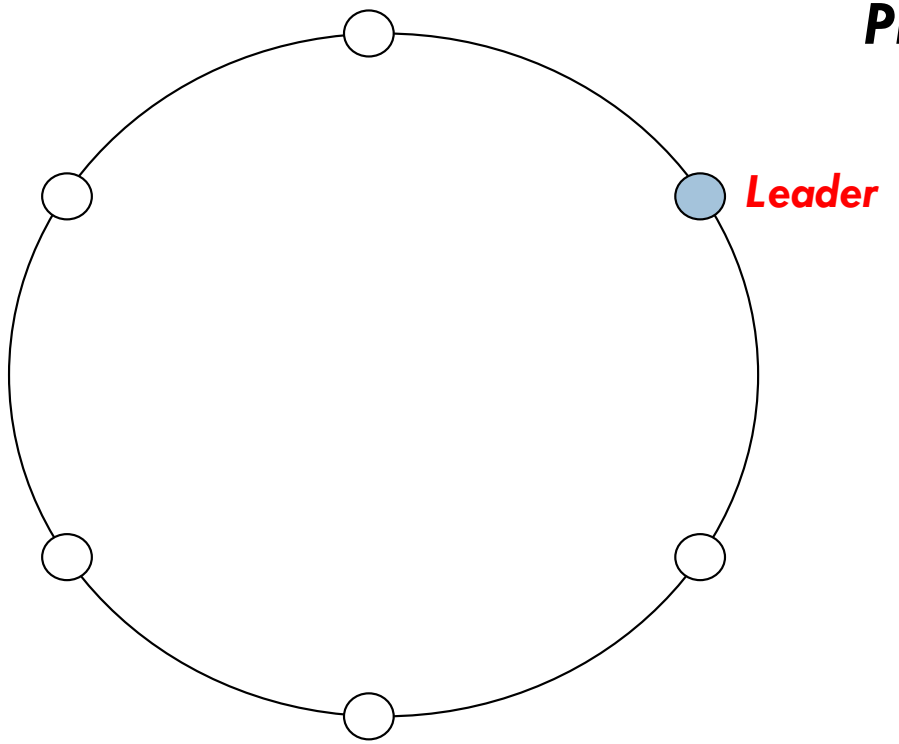
[Itai & Rodeh 81] - Exemple



Phase 3 – init
A = 2

II.2 Unidirectionnel synchrone anonyme

[Itai & Rodeh 81] - Exemple



Phase 3 – ronde N
A = 1

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

- Un nœud actif candidate
-  Il émet son identifiant dans les 2 directions sur des distances croissantes
- Le vainqueur de chaque ronde est celui qui a transmis sur toute la distance
- Un identifiant qui fait le tour de l'anneau désigne le leader

Complexité en messages : O(N log N)

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 succ[i] et pred[i]

attendre retour des deux messages

round++

II.3 Bidirectionnel asynchrone

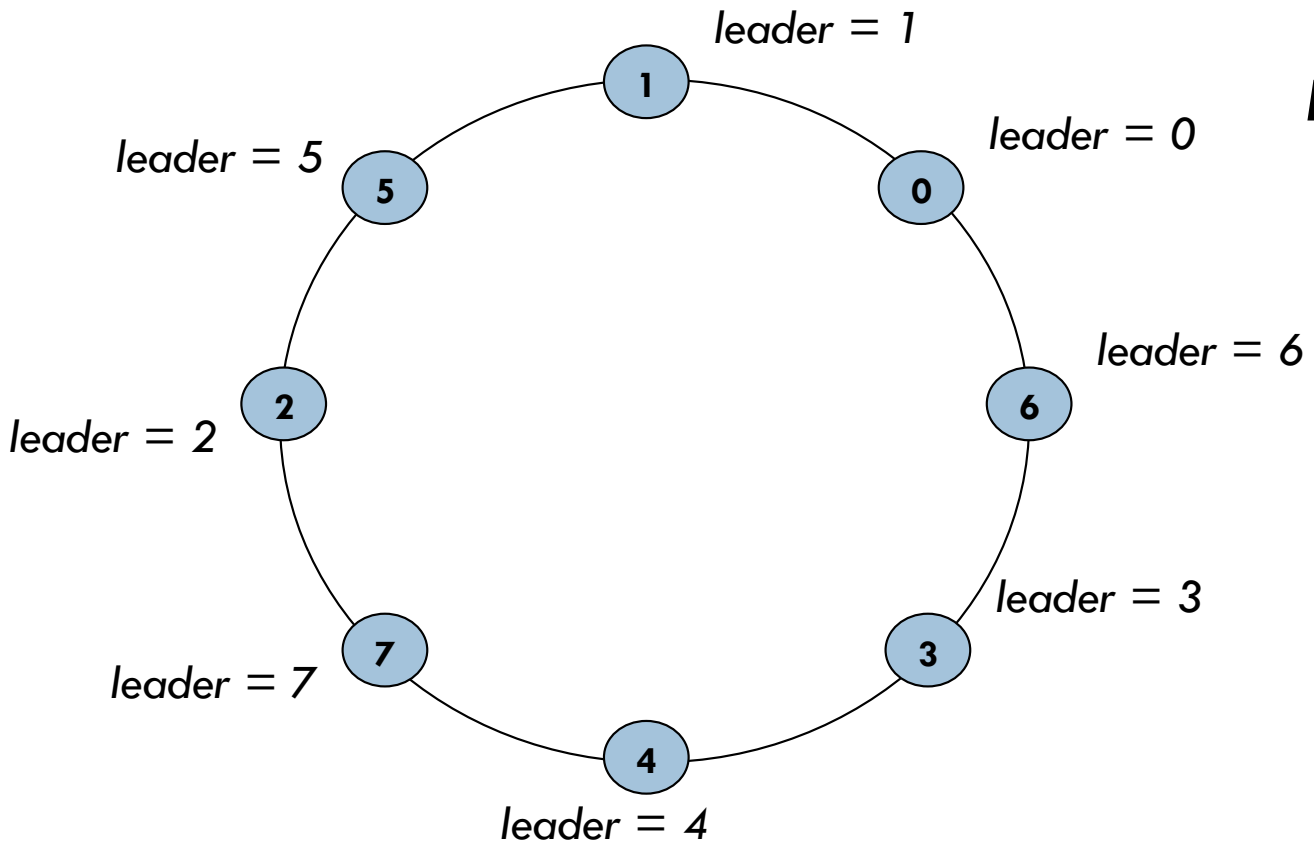
[Hirschberg & Sinclair 1980] – Idée de l'algorithme

Sur réception de <j, TTL> venant de voisin A

```
If j = i then
    if TTL > 0 then se déclarer vainqueur
    else début de ronde
if j > leader then
    state = passive
    leader = j
    if TTL = 1 then
        envoyer <j, 0> a voisin A
    else
        envoyer <j, TTL - 1> a voisin B
```

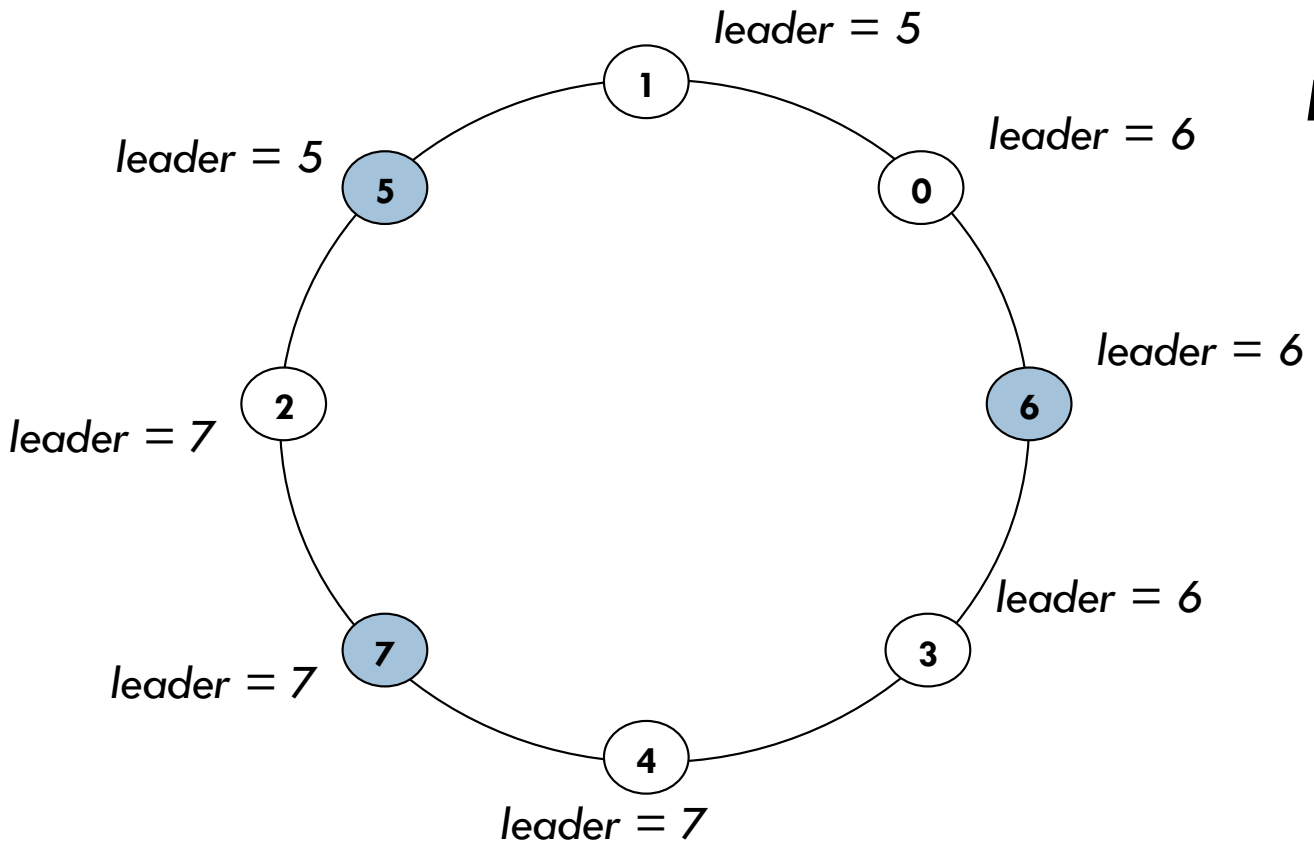
II.3 Bidirectionnel asynchrone

[Hirschberg & Sinclair 1980] – Exemple



II.3 Bidirectionnel asynchrone

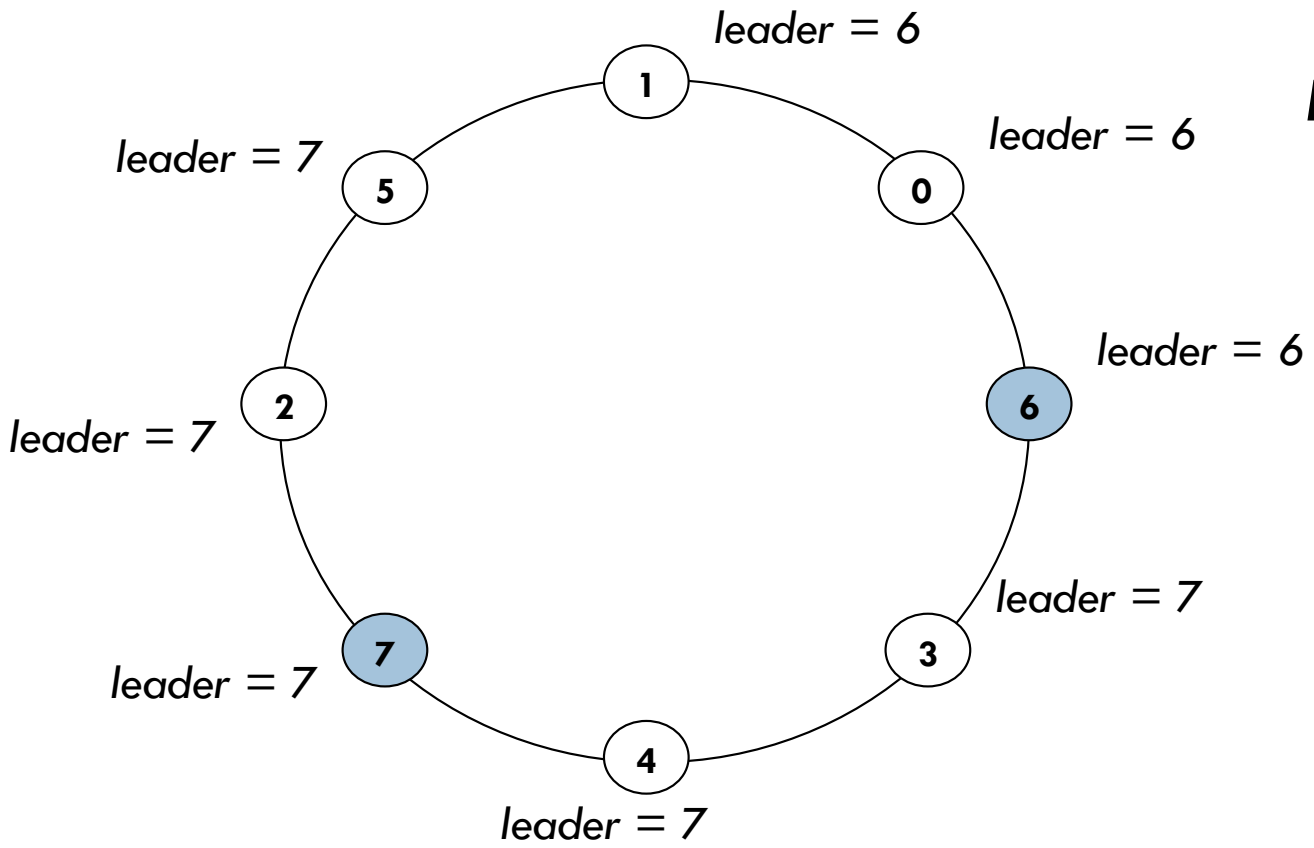
[Hirschberg & Sinclair 1980] – Exemple



Ronde 1 – init

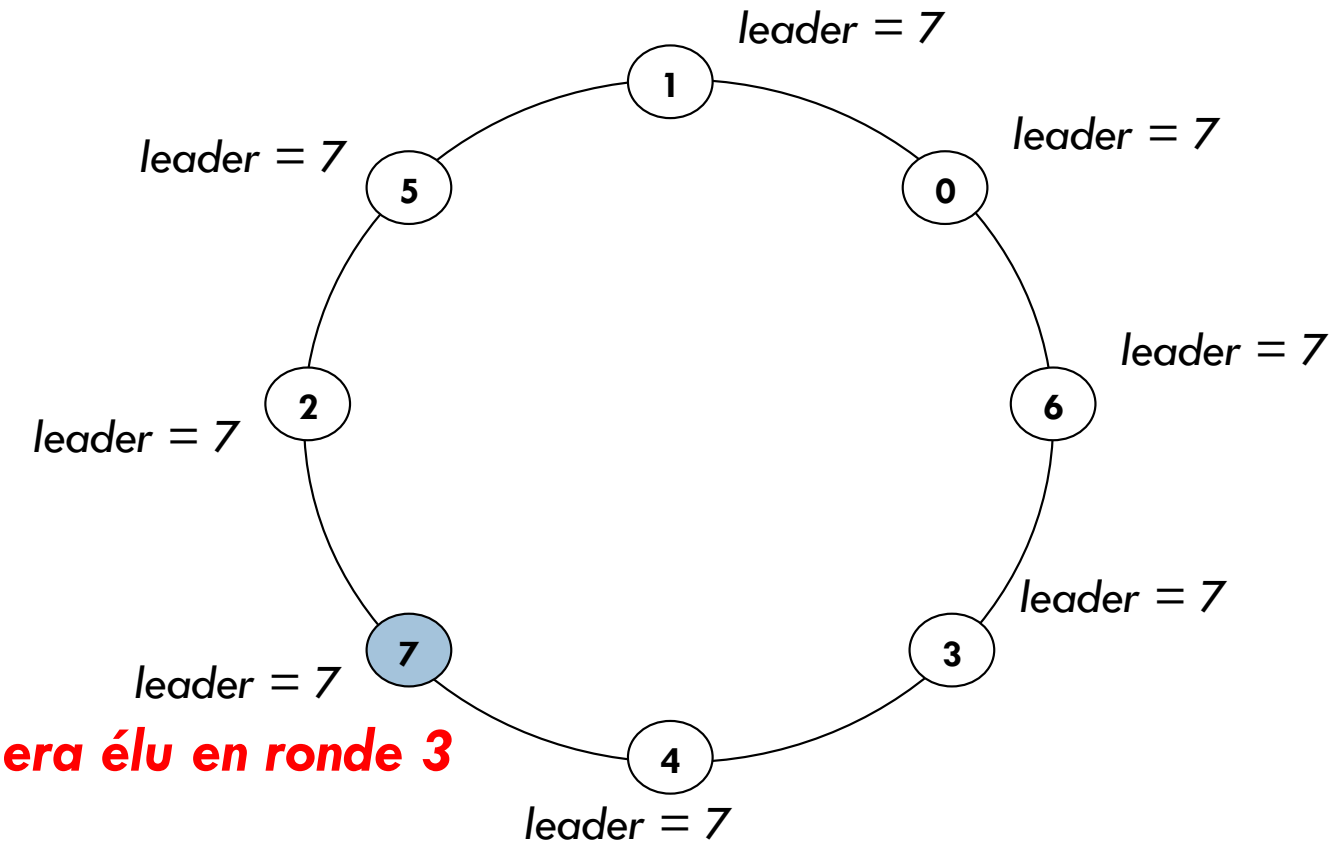
II.3 Bidirectionnel asynchrone

[Hirschberg & Sinclair 1980] – Exemple



II.3 Bidirectionnel asynchrone

[Hirschberg & Sinclair 1980] – Exemple



Plan

I. Introduction

II. Topologie en anneau

III. Topologie quelconque avec hypothèses

1. Construction d'arbre couvrant
2. Diffusion avec identifiants
3. Diffusion anonyme

III.1 Construction d'arbre couvrant

ECHO

Objectif

Construire un arbre unique en un temps fini

Idée

Tous les processus démarrent orphelins (hors de l'arbre)

Un seul processus devient racine (candidat)

La racine propose à ses voisins de devenir leur parent

Sur réception d'une proposition, un processus orphelin

a. devient le fils de l'émetteur et acquitte positivement

b. propose à ses propres voisins de devenir leur parent

III.1 Construction d'arbre couvrant

ECHO

Variables locales à chaque nœud

Neighbors

Ensemble des nœuds voisins

Parent

Référence vers le parent du nœud dans l'arbre couvrant

Initialement NULL

Children

Ensemble des fils, initialement vide

Replies

Compteur de réponses, initialement à 0

III.1 Construction d'arbre couvrant

ECHO

Algorithme

Site i est initiateur

Parent=i

envoyer <PARENT> aux voisins

Site i reçoit une proposition <PARENT> de j

if Parent=NULL then

Parent=j

envoyer <ACK> à j

diffuser <PARENT> à Neighbors\{j}

else

envoyer <NACK> à j

III.1 Construction d'arbre couvrant

ECHO

Algorithme (suite)

Site i reçoit un acquittement <ACK> ou <NACK> de j

```
if <ACK> then
    Children = Children U {j}
Replies++
if Replies = Card(Neighbors) then
    FIN
```

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

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

III.2 Diffusion avec identifiants

FloodMax - Calcul de maximum par propagation

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
- Accès à un broadcast
- Communications fiables et synchrones
- ***Pannes franches***

Idée

Chaque candidat envoie son identité aux autres nœuds du réseau

Un site répond à ceux de numéro inférieur au sien

Un processus qui ne reçoit pas de réponse se déclare le chef

Complexité en nombre de messages est $O(N^3)$ dans le pire des cas

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*, *passive* ou *leader*

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

Algorithme

```
State = actif
repeat
    b = random{0,1}
    broadcast(b)
    while (timer != 0) do
        receive(<b'>)
        Values += b'
    if (b = 1 ∧ Values = 0) then State = leader
    if (b = 0 ∧ Values > 0) then State = passive
until (State != active)
```