Algoritmul Hirschberg-Sinclair

Prezentare laborator APD

Mihai Bărbulescu
331CA

Facultatea de Automatică și Calculatoare
Universitatea Politehnica București

7 ianuarie 2013

Cuprins

1 Context

2 Topologii inel - problema generală

3 HS - principii

4 HS - implementare

5 Întrebări

Problema generică

- Dintr-o colecție de procese, se alege unul singur care urmează să fie lider
- De ce? Alegerea unui lider este necesară ca fază premergătoare execuției unui algoritm centralizat
  - Paradigma client-server a sistemelor distribuite
  - Liderul execută operații de inițializare
  - Liderul controlează alte procese
- Algoritmii de alegere a unui lider sunt descentralizați și participă toate procesele din colecție.

Ce problemă mai simplă, echivalentă, găsim?

- Compoziția exactă a grupului de procese nu e cunoscută ⇒ alegerea statică *nu e o soluție*
- Fiecare proces își cunoaște propriul *PID* și vecinii
- Identitățile aparțin unei mulțimi total ordonate
- Consecință: Alegerea unui lider se transformă în alegerea procesului cu *PID* maxim (sau minim)
- Pentru topologia inel se convine alegerea procesului cu *PID* maxim

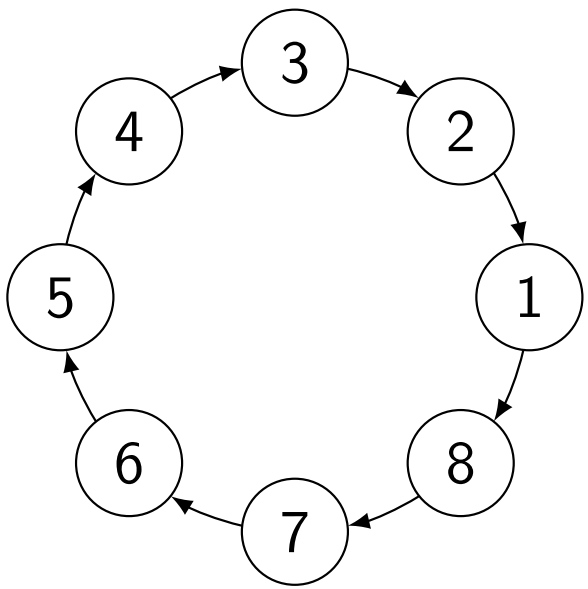
Ce fel de topologie de procese avem?

Arbore → Algoritmi undă (algoritmul tree)
Exemplu de utilizare: În algoritmii de determinare a arborelui minim de acoperire (Prim): liderul este rădăcina arborelui

Inel → Algoritmii LeLann, LeLann Chang Robert, Hirschberg-Sinclair
Exemplu de utilizare: Refacerea după pierderea token-ului (detectată printr-un timeout) într-o rețea Token Ring

Specificarea problemei

- Procese aranjate în inel, identificate prin numere
- Dispunerea proceselor în inel este aleatoare
- Se cere desemnarea prin consens a unui singur proces drept lider al grupului de procese
- Numărul de procese (notat n) și topologia nu sunt cunoscute dinainte
- Soluție distribuită $\Rightarrow$ nu avem control centralizat.



Algoritmul LeLann

Etape:

- Fiecare proces transmite în inel un mesaj care conține identificatorul său
- Fiecare proces colectează numerele celorlalte procese
- Fiecare proces calculează maximul
- Procesul al cărui număr este egal cu maximul devine lider

Complexitate:

Algoritmul LeLann

Etape:

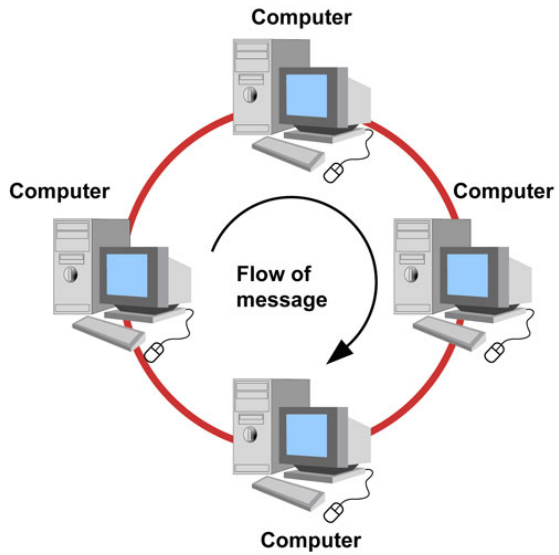
- Fiecare proces transmite în inel un mesaj care conține identificatorul său
- Fiecare proces colectează numerele celorlalte procese
- Fiecare proces calculează maximul
- Procesul al cărui număr este egal cu maximul devine lider

Complexitate:

- Fiecare proces transmite un mesaj care e recepționat de toate celelalte procese
- Număr de mesaje transmise: $O(n^2) \Rightarrow$ prea multe ☹

Algoritmul Lelann-Chang-Robert

- Presupune transmiterea mesajelor în sensul acelor de ceasornic
- Fiecare proces transmite procesului din dreapta un mesaj cu identificatorul său
- Procesul care primește un mesaj compară identificatorul primit (m) cu propriul identificator (id) și hotărăște dacă:
  a) îl transmite ($m > id$)
  b) îl elimină ($m < id$)
  c) devine lider ($m == id$)



Ipoteze

- Lucrează pe un inel bidirecțional ⇒ Procesele pot detecta din ce direcție vine un mesaj, pentru a trimite răspuns în acea direcție
- Se încearcă optimizarea alegerii liderului prin diminuarea numărului de mesaje
- Nu e tratat cazul în care un proces nu știe de alegerea liderului în curs: el poate primi un mesaj și să devină candidat dacă identitatea sa e mai mare decât identitatea din mesaj ⇒ Toate procesele sunt inițiatori

Alegerea în vecinătăți

- **Definiție:** k-vecinătatea unui proces p este totalitatea proceselor aflate în stânga, respectiv în dreapta, în cadrul topologiei inel, la o distanță $d \leq k$ față de procesul p
- Algoritmul funcționează în faze (sau runde) asincrone
- Într-o fază k un proces p încearcă să devină lider în 2^k-vecinătatea lui, trimițând mesaje în ambele direcții
- Procesul p poate trece la faza următoare $(k+1)$ doar dacă are cel mai mare identificator din 2^k-vecinătatea lui.

Alegerea în vecinătăți (cont.)

- **Consecințe**: la fiecare fază se dublează numărul de procese într-o vecinătate, iar numărul proceselor care ajung în faze superioare scade
- La final există un singur proces câștigător, care este liderul topologiei inel

Trimiterea mesajelor - tipuri de date și funcționare

- Procesele trimit mesaje ELECTION care conțin 3 câmpuri:
  - PID = identificatorul procesului
  - k = numărul fazei curente de execuție
  - d = hop counter, incrementat la fiecare SENDPASS dat de un mesaj
- Inițial (faza 0) toate procesele își anunță candidatura, trimițând mesaje vecinilor
- Dacă un proces primește un mesaj ELECTION(x, k, d) astfel încât $d = 2^k$ atunci este capătul unei 2^k-vecinătăți a unui proces p cu $p.id = x$

Alte tipuri de date și funcții

- Un proces p are 2 câmpuri:
  - id = identificatorul unic al procesului în topologie, de tip întreg
  - $status = \begin{cases} candidate \\ leader \end{cases}$

Alte tipuri de date și funcții

- Un proces p are 2 câmpuri:
  - id = identificatorul unic al procesului în topologie, de tip întreg
  - $status = \begin{cases} candidate \\ leader \end{cases}$
- Funcții de comunicare în cadrul algoritmului:
  - SENDBOTH - trimite mesaj atât vecinului din dreapta, cât și celui din stânga
  - SENDPASS - mesaj pasat de un proces, venit din dreapta spre stânga sau invers
  - SENDECHO - trimite răspuns în direcția din care a venit mesajul

Ideea

- În faza k un proces trimite $\text{ELECTION}(p.id, k, d)$ în 2^k-vecinătatea sa în ambele direcții

Ideea

- În faza k un proces trimite $\text{ELECTION}(p.id, k, d)$ în 2^k-vecinătatea sa în ambele direcții
- Dacă un proces p primește $\text{ELECTION}(x, k, d)$ cu $x > p.id$ va trimite mai departe mesajul, incrementându-l pe d, altfel îl ignoră

Ideea

- În faza k un proces trimite $\text{ELECTION}(p.id, k, d)$ în 2^k-vecinătatea sa în ambele direcții
- Dacă un proces p primește $\text{ELECTION}(x, k, d)$ cu $x > p.id$ va trimite mai departe mesajul, incrementându-l pe d, altfel îl ignoră
- Ultimul proces p dintr-o 2^k-vecinătate trimite răspuns, care conține ID_{primit} spre procesul origine q dacă $p.id < ID_{primit}$

Ideea

- În faza k un proces trimite $\text{ELECTION}(p.id, k, d)$ în 2^k-vecinătatea sa în ambele direcții
- Dacă un proces p primește $\text{ELECTION}(x, k, d)$ cu $x > p.id$ va trimite mai departe mesajul, incrementându-l pe d, altfel îl ignoră
- Ultimul proces p dintr-o 2^k-vecinătate trimite răspuns, care conține ID_{primit} spre procesul origine q dacă $p.id < ID_{primit}$
- Răspunsurile sunt date mai departe întotdeauna

Ideea

- În faza k un proces trimite $\text{ELECTION}(p.id, k, d)$ în 2^k-vecinătatea sa în ambele direcții
- Dacă un proces p primește $\text{ELECTION}(x, k, d)$ cu $x > p.id$ va trimite mai departe mesajul, incrementându-l pe d, altfel îl ignoră
- Ultimul proces p dintr-o 2^k-vecinătate trimite răspuns, care conține ID_{primit} spre procesul origine q dacă $p.id < ID_{primit}$
- Răspunsurile sunt date mai departe întotdeauna
- Un proces ajunge în faza superioară $k+1$ doar dacă primește răspuns din ambele părți în faza k

Ideea

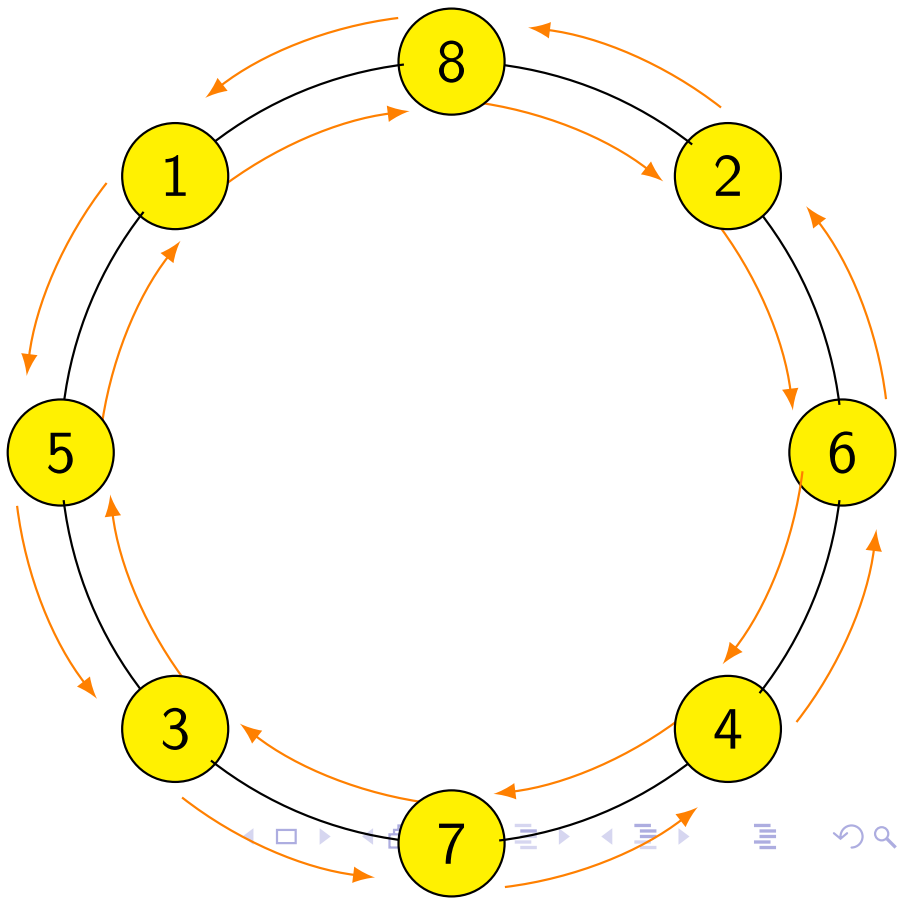
- În faza k un proces trimite $\text{ELECTION}(p.id, k, d)$ în 2^k-vecinătatea sa în ambele direcții
- Dacă un proces p primește $\text{ELECTION}(x, k, d)$ cu $x > p.id$ va trimite mai departe mesajul, incrementându-l pe d, altfel îl ignoră
- Ultimul proces p dintr-o 2^k-vecinătate trimite răspuns, care conține ID_{primit} spre procesul origine q dacă $p.id < ID_{primit}$
- Răspunsurile sunt date mai departe întotdeauna
- Un proces ajunge în faza superioară $k+1$ doar dacă primește răspuns din ambele părți în faza k
- Liderul este procesul p care primește $\text{ELECTION}(x, k, d)$ astfel încât $x == p.id$

Pseudocod

```
void HirschbergSinclair(process p) {
    p.status = candidate;      //Faza 0
    sendboth(election(p.id, 0, 0));
    //La primirea unui mesaj election(j, k, d)
    if( j > p.id && d <= pow(2, k) )
      sendpass(election(j, k, d + 1);
    if(j > p.id && d == pow(2, k) )
      sendecho(election(j, k, d));
    if(p.id == j)
       broadcast p.status = leader;
    //La primirea unui echo cu j si k
    if(p.id != j)
      sendecho(election(j, k, d));
    else if(am mai primit echo cu j si k)
       sendpass(election(j, k + 1, 1)
}
```

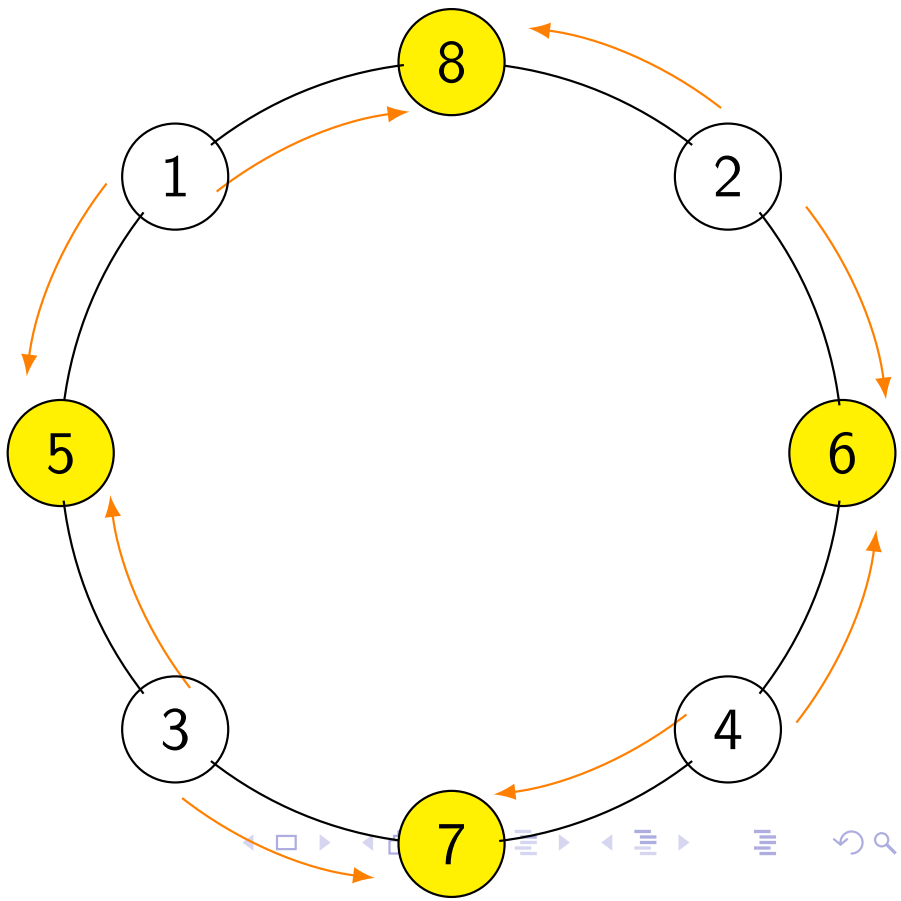
Rulare Hirschberg-Sinclair

- Galben = candidat, a ajuns în fazele superioare
- Alb = a pierdut, nu mai poate deveni lider
- **Inițial (faza 0):** Toate nodurile trimit în stânga și în dreapta $\text{ELECTION}(p.id, 0, 1)$



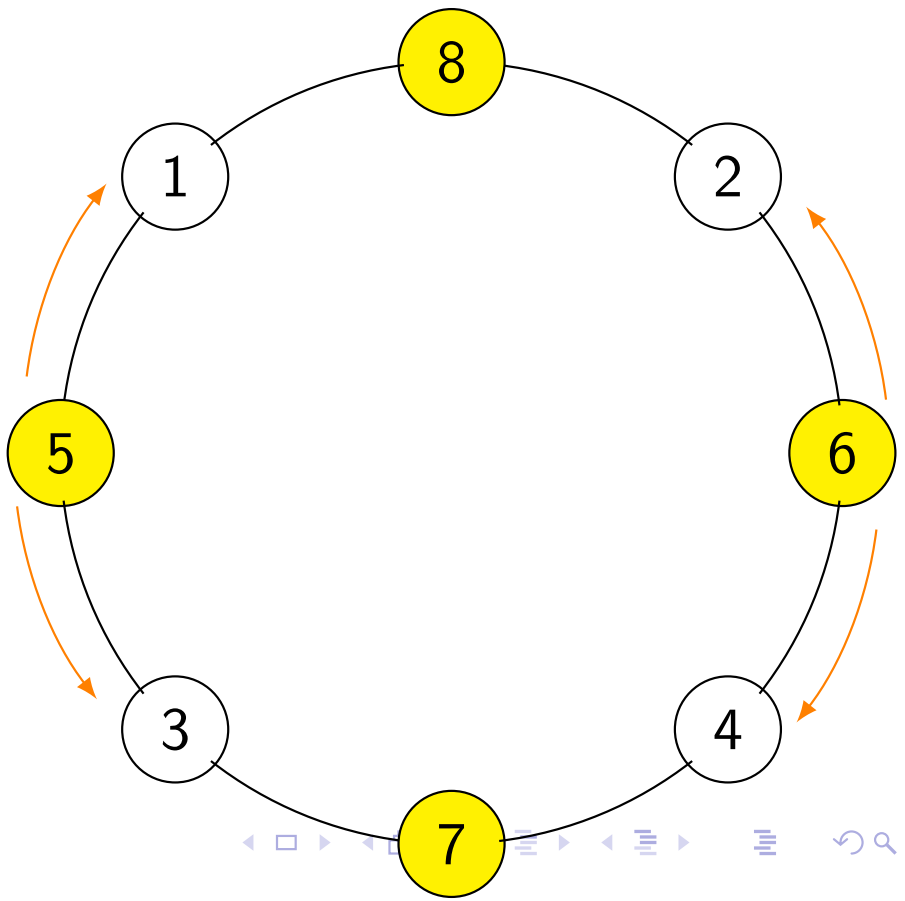
Rulare Hirschberg-Sinclair (cont.)

- Procesele care au $p.id < ID_{primit}$ trimit răspuns
- Astfel în faza 1 ajung doar procesele galbene din figură, deoarece primesc răspuns din ambele părți



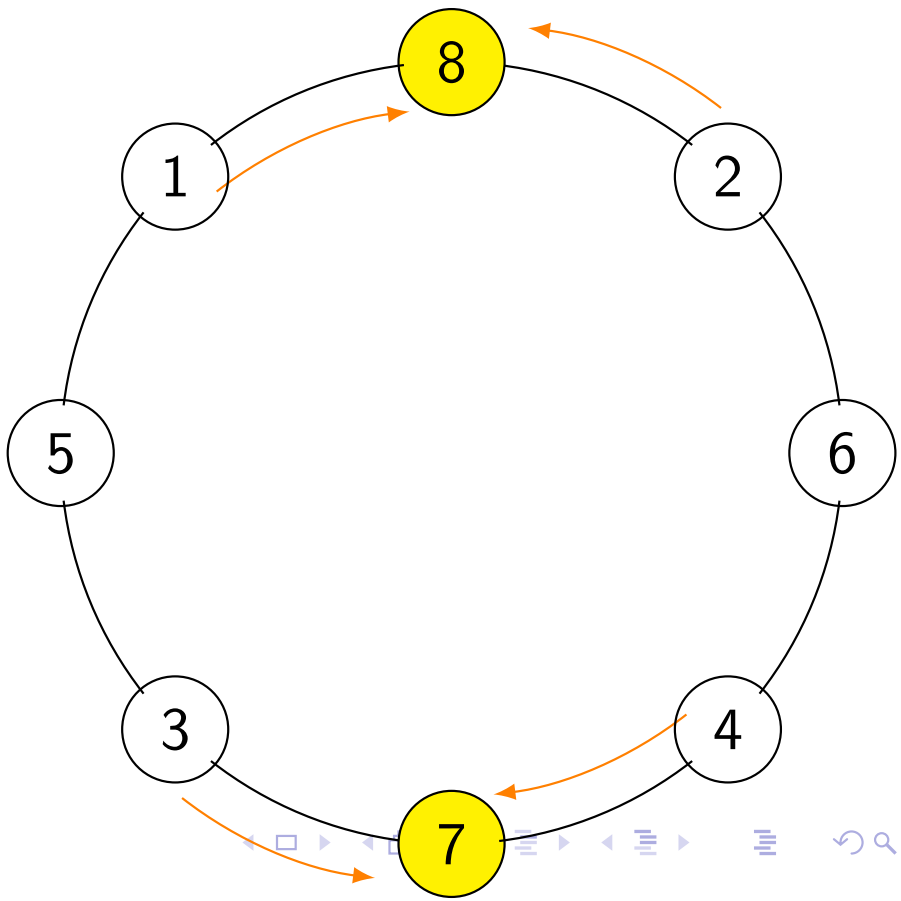
Rulare Hirschberg-Sinclair (cont.)

- **Faza 1:** Nodurile galbene trimit mesaje $\text{ELECTION}(p.id, 1, 2)$ în 2^1-vecinătatea lor
- Nodurile albe dau SENDPASS
- Nodurile 5 și 6 primesc un *PID* mai mare decât al lor și în plus sunt capete de vecinătate, deci trimit răspunsul înapoi la origine (8, respectiv 7)



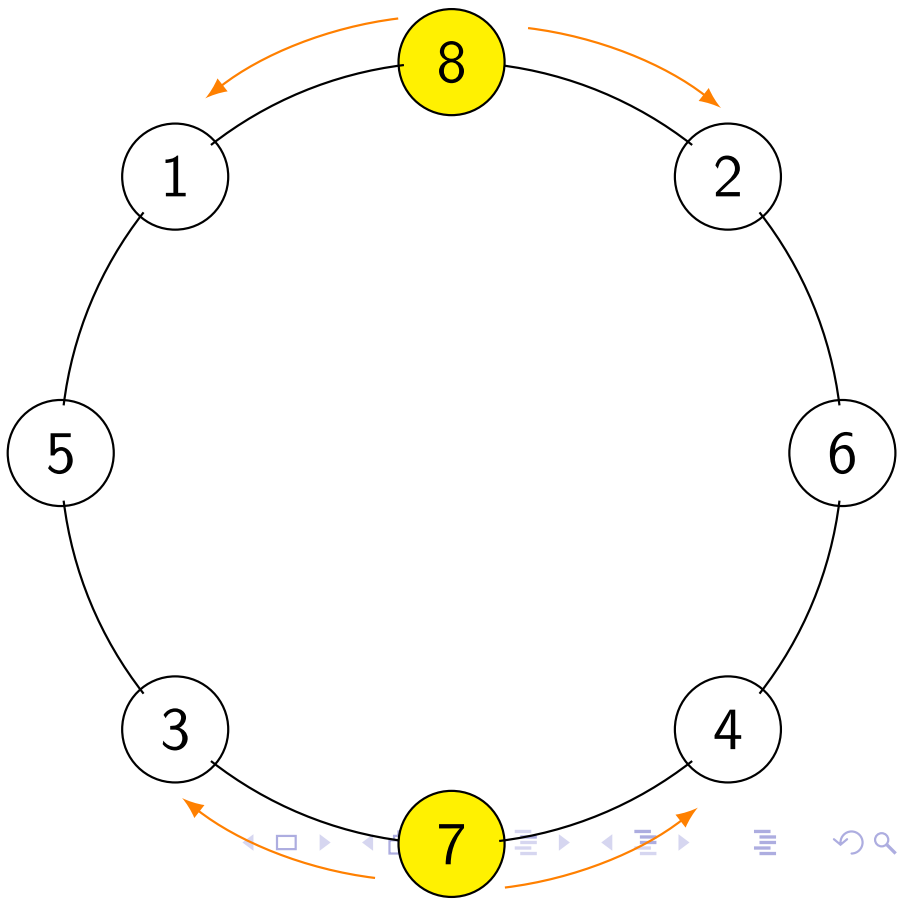
Rulare Hirschberg-Sinclair (cont.)

- Nodurile 8 și 7 au primit răspunsuri din ambele direcții
- Sunt astfel singurele procese care mai pot fi lider



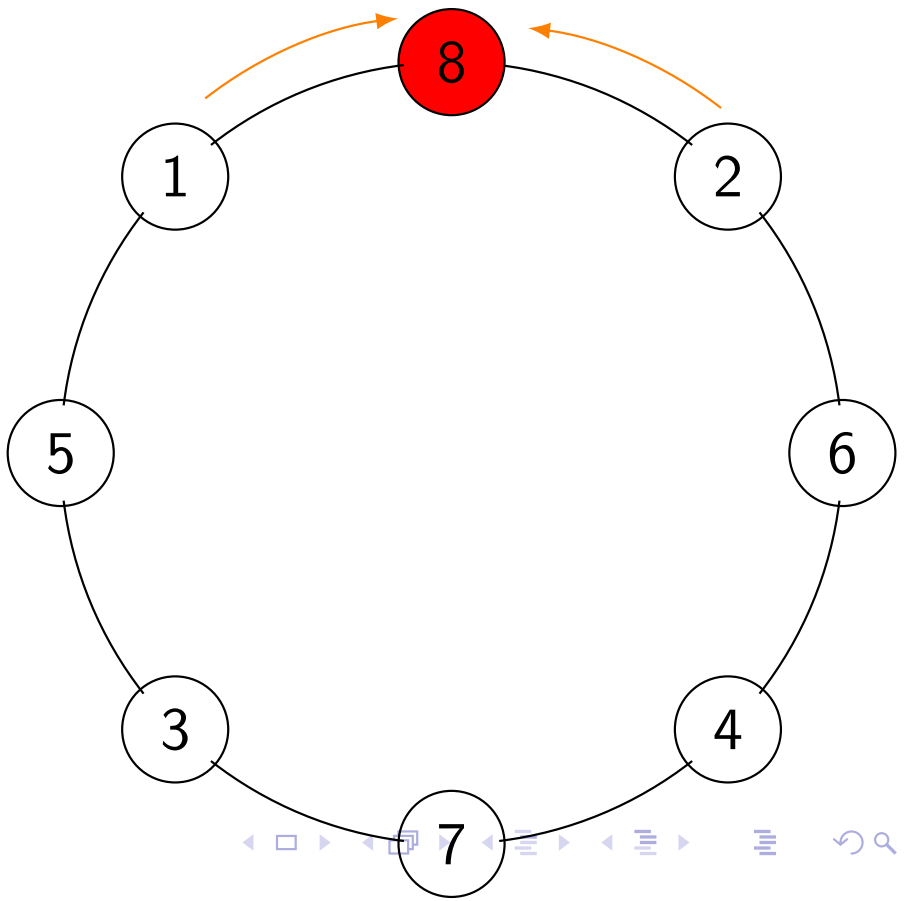
Rulare Hirschberg-Sinclair (cont.)

- **Faza 2:** 7 si 8 vor trimite $\text{ELECTION}(p.id, 2, 4)$ în 2^2-vecinătățile lor
- Nodurile albe dau SENDPASS
- Nodul 7 este capăt de vecinătate și primește un *ID* mai mare decât al său
- Nodul 8 este capăt de vecinătate și primește un *ID* mai mic decât al său



Rulare Hirschberg-Sinclair (cont.)

- Nodul 8 va primi răspunsuri din ambele direcții cu propriul ID
- Algoritmul se încheie



Complexitate

- Număr de mesaje: $O(n \log n)$ în cel mai rău caz ☺
  - LeLann Chang Roberts are în cazul cel mai deforabil $O(n^2)$ mesaje trimise (prea multe ☹)
- Timp: $O(n)$ (la fel ca la LeLann Chang Roberts)
- Preferat datorită numărului redus de mesaje trimise

Întrebări

- client-server
- centralizare
- lider
- topologie inel
- număr redus de mesaje
- faze asincrone