Sisteme și algoritmi distribuiți
Curs 3

Cuprins

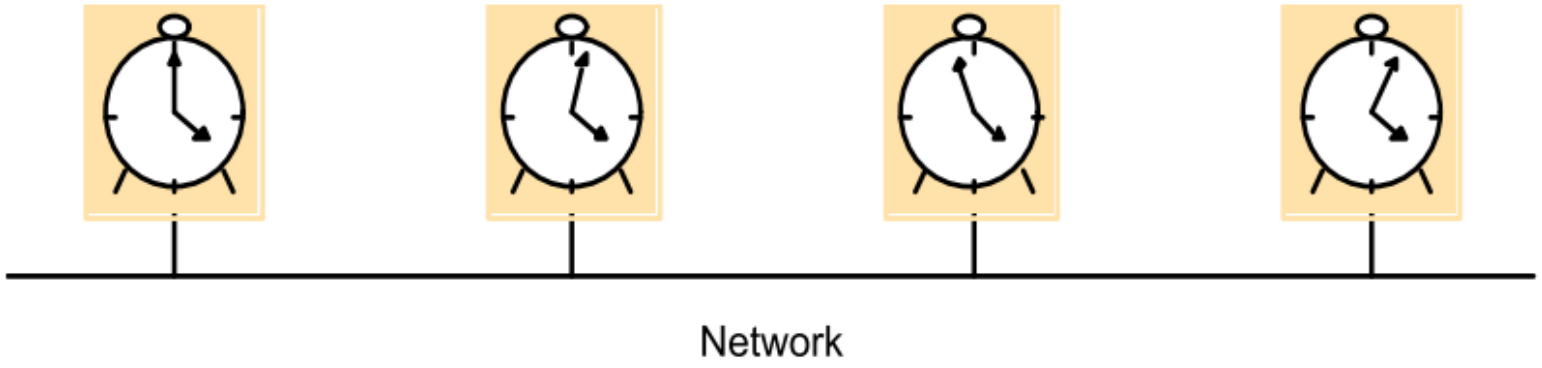
Timp

- Ceasurile fizice în computere sunt realizate prin contorizare
  - Ceasuri atomice: drift 1s/150 milioane de ani
  - Ceasuri de sistem
  - Ceasuri de timp real: alimentate prin baterie (funcționează chiar dacă sistemul este oprit)

- $h(t)$ *rata (viteza) ceasului hardware*
- $H(t) = \int_0^t h(\tau)\tau$ valoarea ceasului hardware
- Ceas logic (registru): $C(t) = \alpha H(t) + \beta$
- $C(t)$ este un scalar crescător și se actualizează prin citiri ale lui $H(t)$

Timp

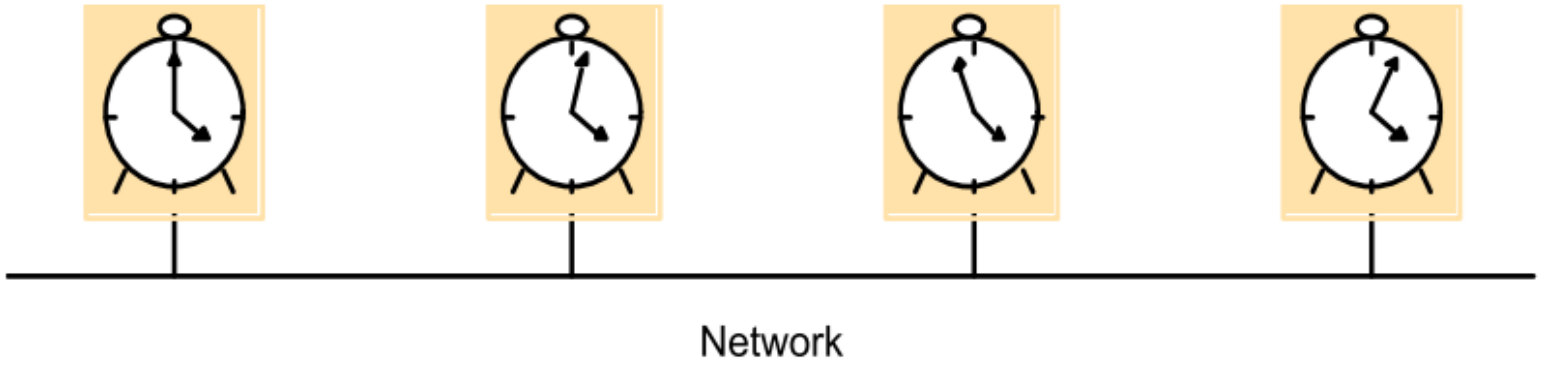
- In SD, ceasurile locale sunt **decalate** și **întârziate**
- **Decalaj** între nodurile (i, j) la momentul t: $|C_i(t) - C_j(t)|$
- **Întârziere** între nodurile (i, j) la momentul t: $|\frac{d}{dt}C_i(t) - \frac{d}{dt}C_j(t)|$



Problema sincronizării

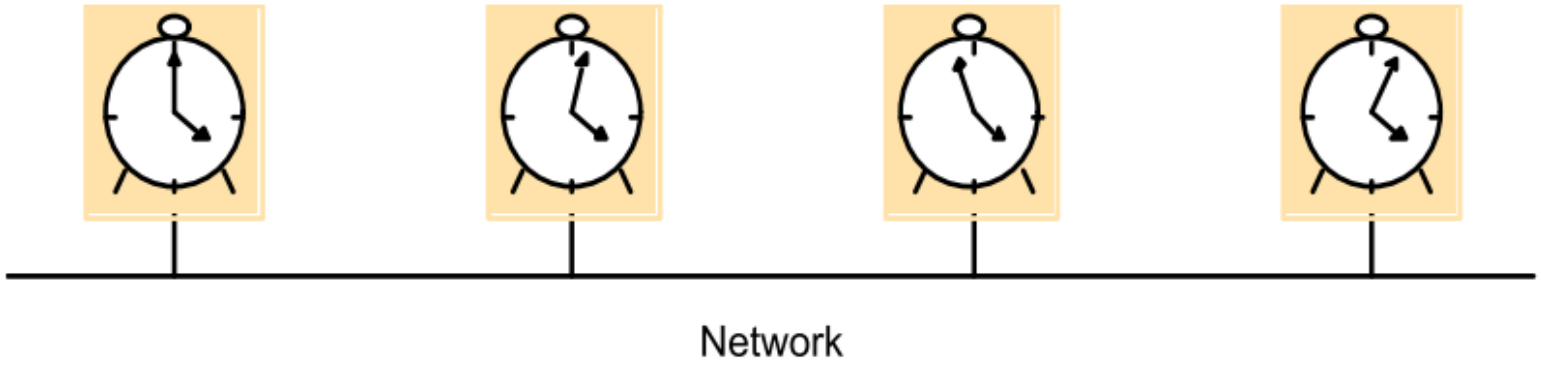
Folosind referințe externe sau interne, problema sincronizării ceasurilor se reduce la asigurarea relației (de precizie):

$$\left|C_i(t) - C_j(t)\right| \leq \rho \qquad \forall t \geq t_0$$



Problema sincronizării

- Sincronizare externă: referința este un ceas absolut extern e.g. Coordinate Universal Time (UTC), GPS etc.
- Sincronizare internă: referința este o valoare stabilită în rețea
  - Algoritmul lui Cristian
  - Algoritmul de Medie (Berkeley Algorithm)



UTC

UTC este standardul primar de timp în lume, transmis prin: radio, linii telefonice line, satelit (GPS) etc.

Protocoalele populare de sincronizare folosesc referința UTC și urmăresc să asigure relația (acuratețe):

$$|S(t) - C_i(t)| < \rho \quad \forall i, \forall t$$

Dacă asigurăm acuratețe ρ atunci ce precizie garantăm?

Algoritmul lui Cristian

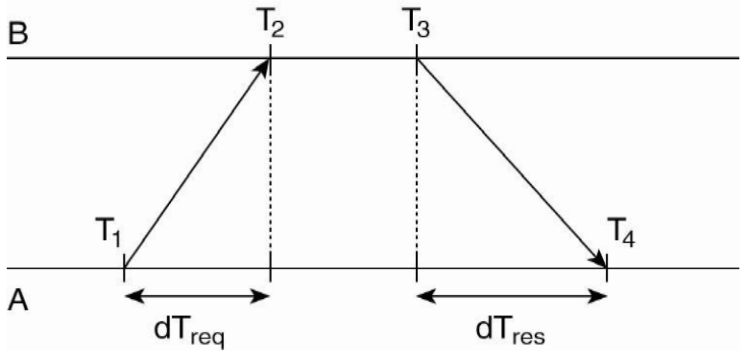
- Ipoteza 1: Întârzieri pe comunicație simetrice și mărginite (rețele LAN)
- Ipoteza 2: Există un nod de referință (pasiv) R cu unicul rol de a furniza referința
- Nodurile care nu sunt referința se vor sincroniza cu ceasul referinței

Algoritmul lui Cristian

Procedura AC: Nodul P cere periodic valoarea timpului de la R:

- T_1: send request; T_4: primește reply
- P primește val. T_2 și T_3 de la R, ajustează $C(t) = T_3 + T_{res}$ (T_{res} timp livrare mesaj)
- Folosește estimarea $T_{res} \approx \frac{T_{req}+T_{res}}{2}$

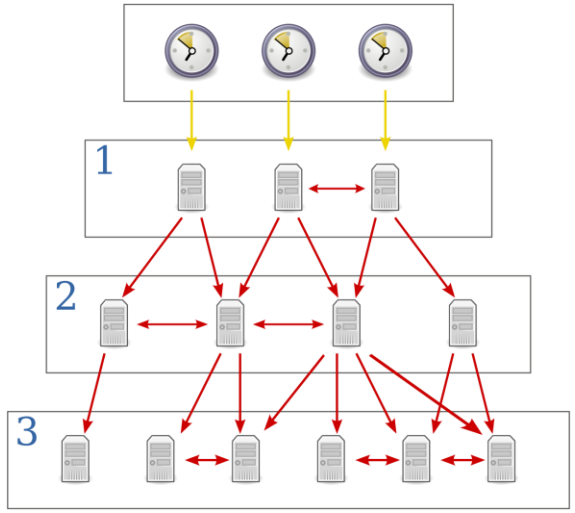
$$T_{res} = \frac{(T_2 - T_1) + (T_4 - T_3)}{2}$$



Network Time Protocol (NTP)

- Implementare standard în rețele a alg. lui Cristian
- Peste multiple măsurători, calculăm T_{res} minim

- Ierarhie a serverelor de timp:
  - un TS de nivel k se sincronizează după unul de nivel $\leq k - 1$
  - Nivelul 0 este referința UTC



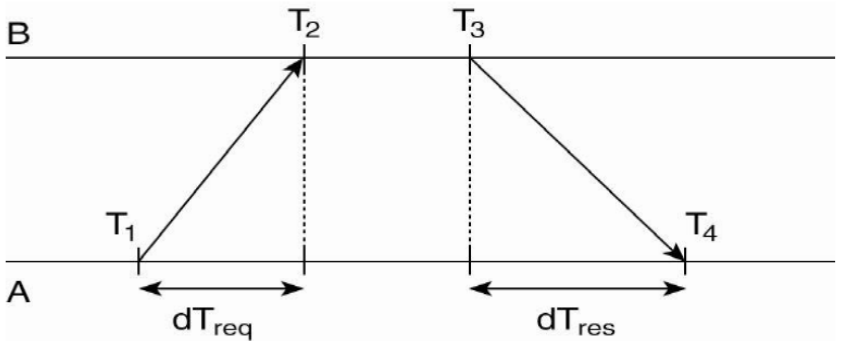
https://en.wikipedia.org/wiki/Network_Time_Protocol

Exemplu

La $5:08:15.100$, P trimite mesaj de request către R. La $5:08:15.900$, P primește răspuns de la B cu valoarea $5:09:25.300$ ($T_2 = T_3$)

- Care este valoarea ajustată a ceasului local $T_4 = C_P(t)$ după sincronizare?

$$T_{res} = \frac{(T_2 - T_1) + (T_4 - T_3)}{2}$$



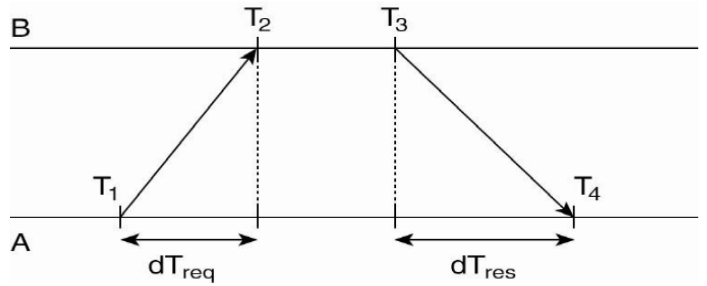
Exemplu

La $5{:}\,08{:}\,15.100$, P trimite mesaj de request către R. La $5{:}\,08{:}\,15.900$, P primește răspuns de la B cu valoarea $5{:}\,09{:}\,25.300$ ($T_2 = T_3$)

- Send req la $T_1 = 5{:}\,08{:}\,15.100$
- Primește răsp. la $T_4 = \ 5{:}\,08{:}\,15.900$
- Mesajul este$\mathrm{T}_3 = \mathrm{T}_2 = C_R(t) = \ 5{:}\,09{:}\,25.300$

- Durata totală $T_4 \ - \ T_1 = 800\ ms$
- Estimare: mesaj generat în urmă cu $400\ ms$
- Set $\mathrm{C_P(t)} = T_{serv} + 400 = 5{:}\,09.25.700$

$$T_{res} = \frac{(T_2 - T_1) + (T_4 - T_3)}{2}$$

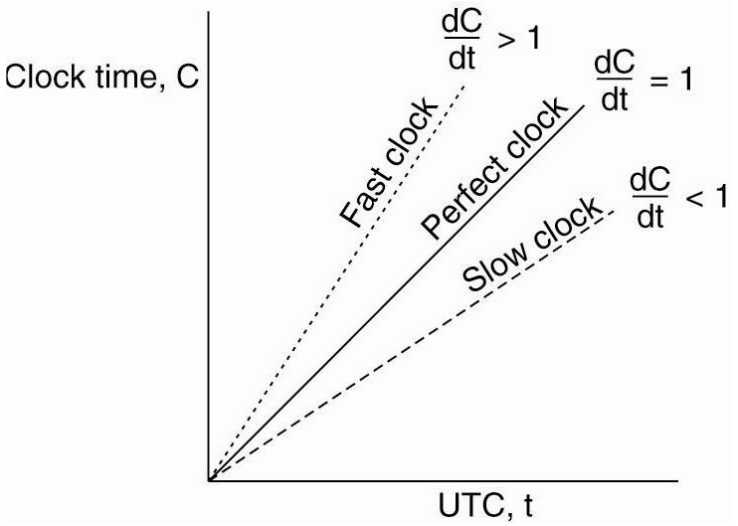


Sincronizare internă

Sincronizarea internă a ceasurilor locale în DS presupune (precizie)

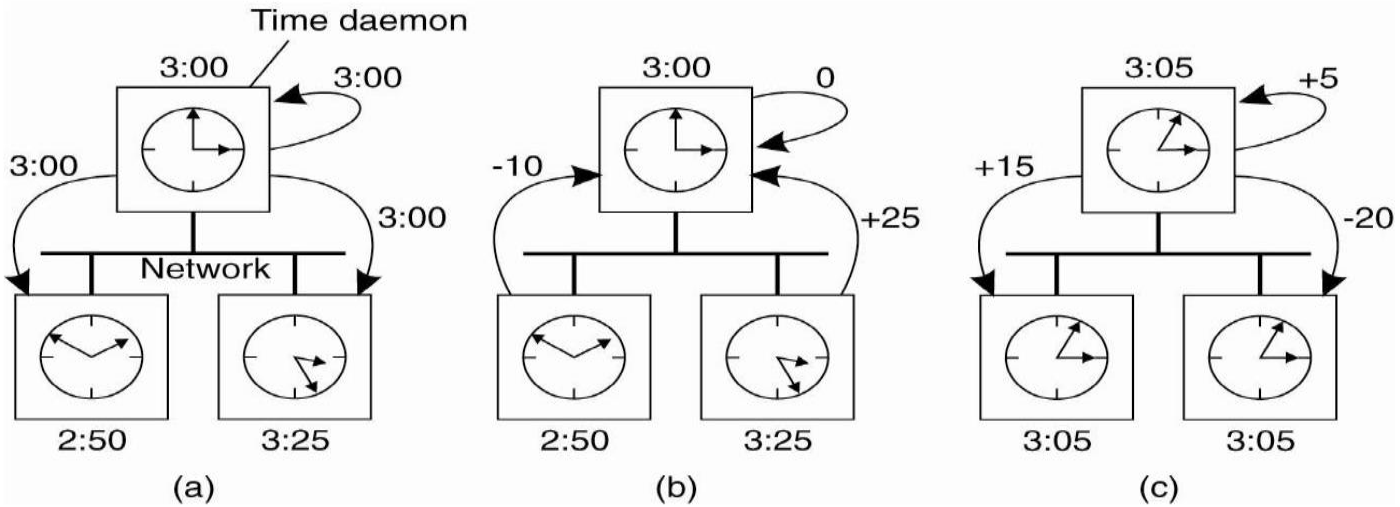
$$\left|C_j(t) - C_i(t)\right| < \rho \quad \forall i, j, t$$

- Necesită algoritmi complet distribuiți
- Problema este una de consens distribuit
- Vom reveni la ea în cursurile următoare



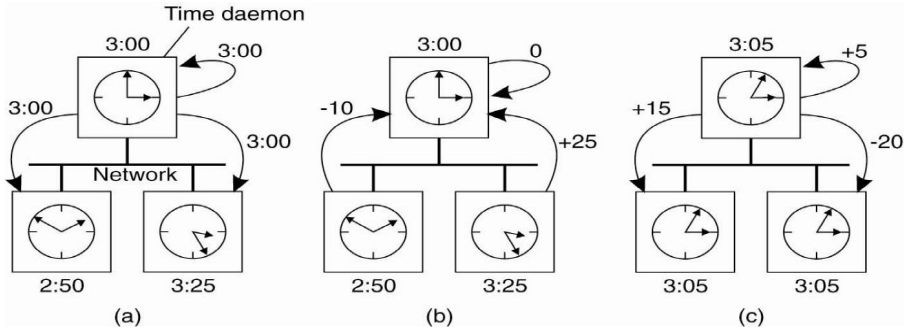
Algoritmul de medie (Berkeley Algorithm)

- Referința este unul din nodurile rețelei, ales eventual prin proceduri de leader-election
- Restul nodurilor urmăresc alinierea ceasurilor cu referința (consens)



Algoritmul de medie (Berkeley Algorithm)

- Referința este unul din nodurile rețelei, ales eventual prin proceduri de leader-election
- Restul nodurilor urmăresc alinierea ceasurilor cu referința (consens)
- Pe scurt: la iterația t
  - R difuzează valoarea $\mathrm{C_R}(t)$
  - P_i calculează întârzierea locală $\delta_\mathrm{i} = |\mathrm{C_R}(t) - C_i(t)|$ și răspunde lui R
  - R distribuie ajustările pentru $C_i(t)$



Alternative

1. If two machines don’t interact, there is no need to synchronize them (Leslie Lamport)

2. Dinamica întârzierii per link $\frac{T_{req}+T_{res}}{2}$ depinde de mai mulți factori

3. Adesea, contează ca procesele să convină asupra **ordinii** evenimentelor, și nu asupra **timpului** la care au avut loc (vezi Ceasuri Logice).

Cuprins

Algoritmi sincroni în SD

Un algoritm distribuit sincron reprezintă un set de operații de calcul/comunicație *executat în iterații/runde* (contorizate de t) pe nodurile sistemului, cu scopul rezolvării unei sarcini concrete.

- Denumim starea locală a nodului i la momentul t cu $\mathrm{x_i}(t)$. Scopul algoritmului este conducerea lui $\mathrm{x_i}(t)$ către starea optimă $\mathrm{x_i}(\infty)$

- La următorul moment de timp starea $\mathrm{x_i}$ suferă o transformare bazată pe pașii algoritmului și informația provenită de la vecini. Pe scurt,

$$x_i(t+1) \coloneqq f_i(x(t))$$

unde $f_i(\cdot)$ reprezintă funcția de transformare asociată nodului i. Funcția $f_i(\cdot)$ definește însuși algoritmul SPMD.

Algoritmi sincroni în SD

La următorul moment de timp starea x_i suferă o transformare bazată pe pașii algoritmului și informația provenită de la vecini. Pe scurt,

$$x_i(t+1) := f_i(x(t))$$

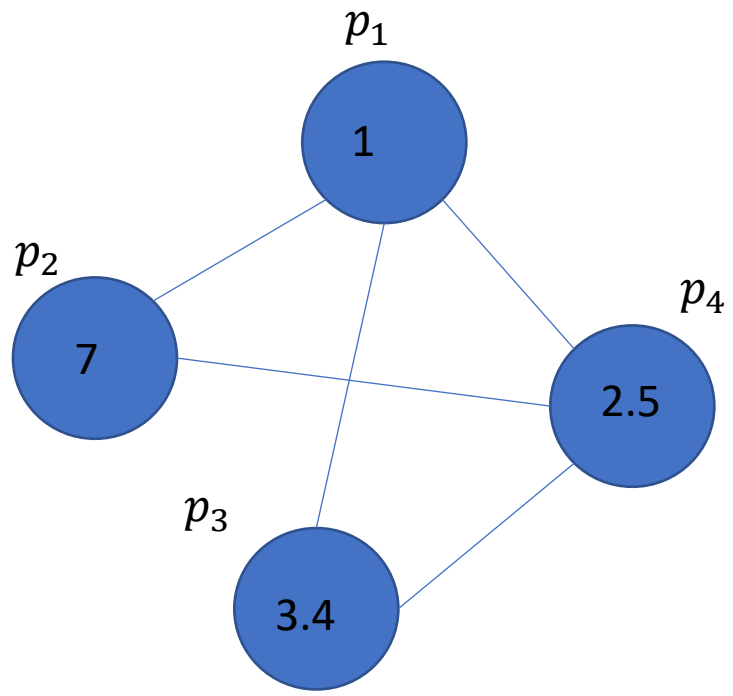
unde $f_i(\cdot)$ reprezintă funcția de transformare asociată nodului i.

Funcția $f_i(\cdot)$ este definită în format SPMD și este compusă din una sau mai multe operații (e.g. aritmetice, numerice) asupra variabilelor locale din memoria nodului i.

Medie șir de numere reale

Problemă [Distributed averaging]: Vectorul $x(0)$ este distribuit peste n noduri, astfel încât $x_i(0)$ se află în memoria locală a nodului i . Calculează distribuit media aritmetică $\mathrm{m} = \sum_i x_i(0)/n$ a vectorului $x(0)$, încât la final $x_i(\infty) = m$.

$$x(0) = \begin{bmatrix} 1 \\ 7 \\ 3.4 \\ 2.5 \end{bmatrix}$$

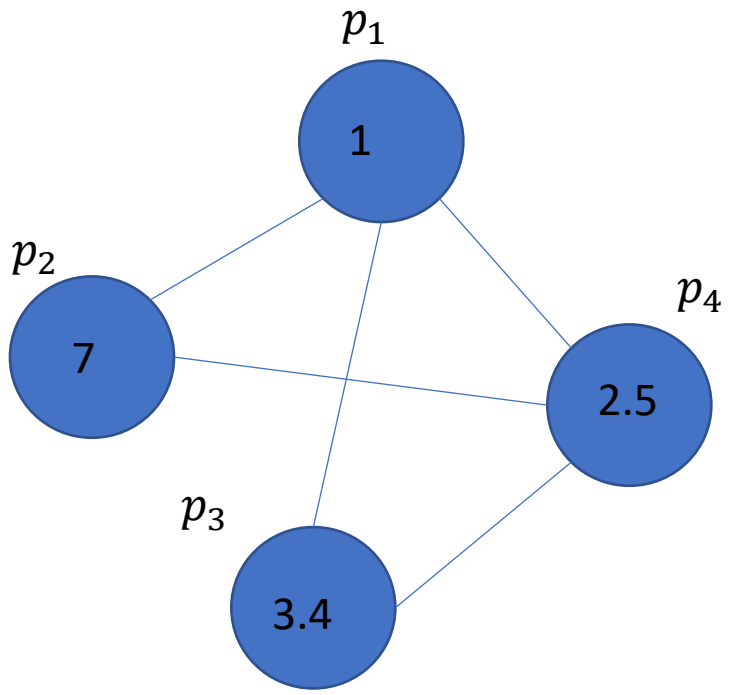


Medie șir de numere reale

Problemă [Distributed averaging]: Vectorul $x(0)$ este distribuit peste n noduri, astfel încât $x_i(0)$ se află în memoria locală a nodului i . Calculează distribuit media aritmetică $\mathrm{m} = \sum_i x_i(0)/n$ a vectorului $x(0)$, încât la final $x_i(\infty) = m$.

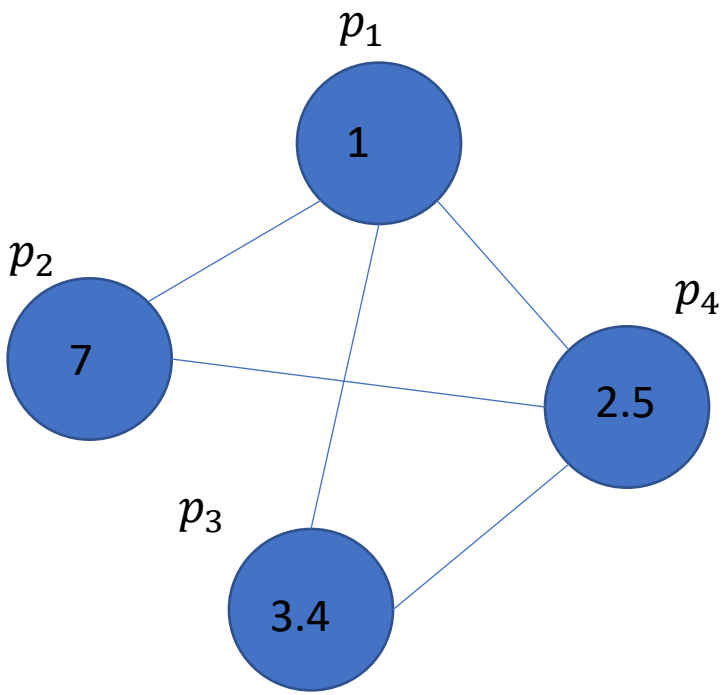
Algoritm: Alege $j \in N_i$ și actualizează $x_i(t+1) = f_i\big(x(t)\big) = \frac{x_i(t)+x_j(t)}{2}$

$$x(0) = \begin{bmatrix} 1 \\ 7 \\ 3.4 \\ 2.5 \end{bmatrix}$$



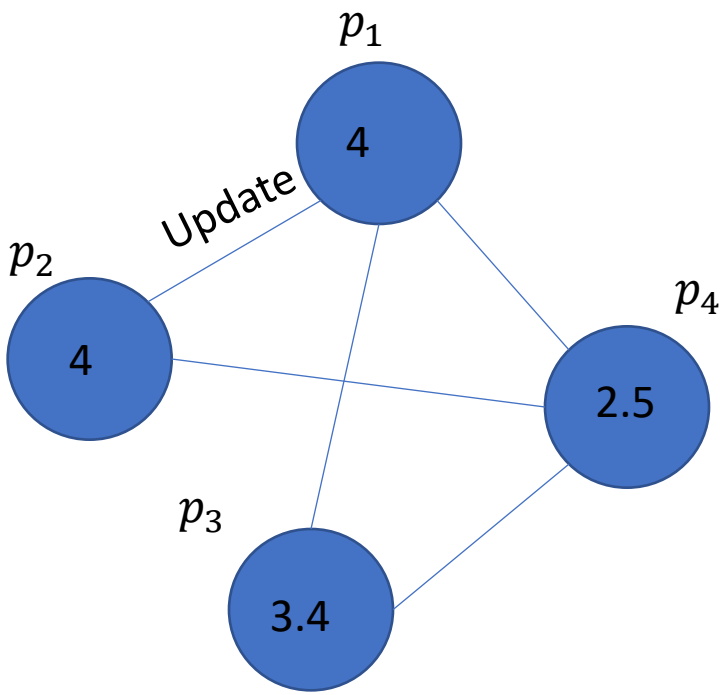
Medie șir de numere reale

$$x(0) = \begin{bmatrix} 1 \\ 7 \\ 3.4 \\ 2.5 \end{bmatrix}$$



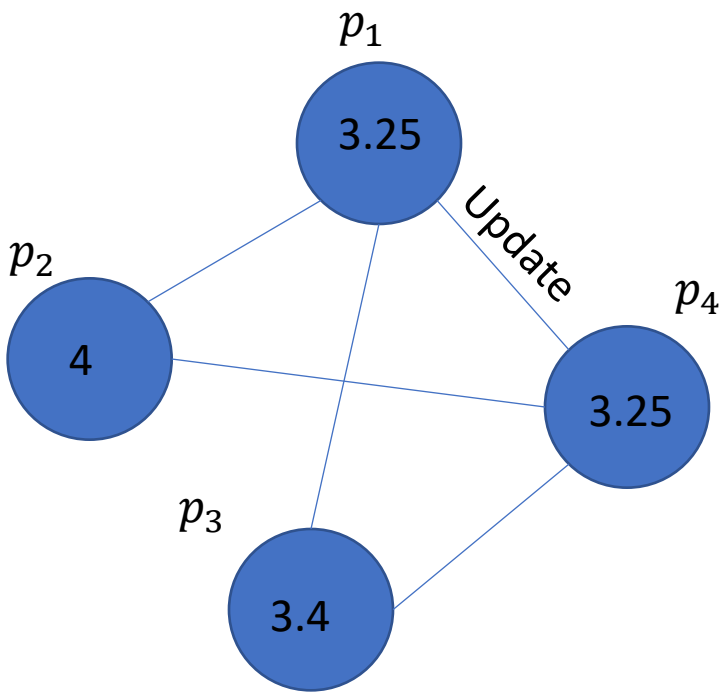
Medie șir de numere reale

$$x(1) = \begin{bmatrix} 4 \\ 4 \\ 3.4 \\ 2.5 \end{bmatrix}$$



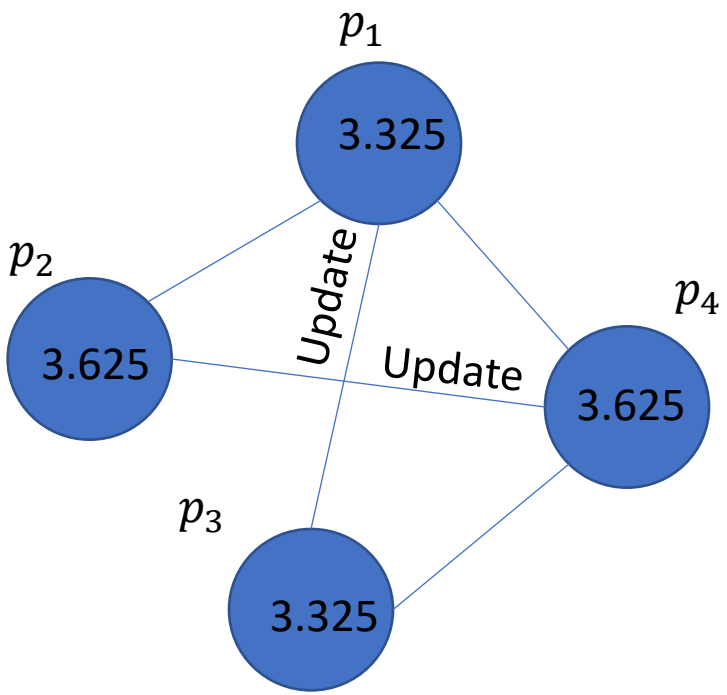
Medie șir de numere reale

$$x(2) = \begin{bmatrix} 3.25 \\ 4 \\ 3.4 \\ 3.25 \end{bmatrix}$$



Medie șir de numere reale

$$x(4) = \begin{bmatrix} 3.325 \\ 3.625 \\ 3.325 \\ 3.625 \end{bmatrix}$$



Algoritmi Sincroni în SD

La finalul iterației t fiecare nod finalizează calculul local, și comunicația cu vecinii, specifice iterației t

$$x(t+1) := \begin{bmatrix} x_1(t+1) \\ \dots \\ x_n(t+1) \end{bmatrix} = \begin{bmatrix} f_1(x(t)) \\ \dots \\ f_n(x(t)) \end{bmatrix} = F(x(t))$$

- Există o margine superioară pe timpul de comunicație între oricare două noduri.
- Adesea implementarea unui criteriu de oprire este o operație dificilă!

Alegere Lider (Leader Election)

În multe aplicații este necesară alegerea unui nod pentru operații particulare (e.g. difuzare, distribuție, master-slave).

Fiecare nod are un ID unic, ales dintr-un spațiu total ordonat.

Convenție: Lider = **nodul cu ID-ul maxim**.

Algoritmii de LE realizează *de facto* calculul distribuit al funcției

$$\max\{id_1, id_2, \dots, id_n\}$$

Alegere Lider (Leader Election)

Starea locală a nodului i specifică calitatea de lider/non-lider:

$$x_i(t) \in \{lider, non-lider\}$$

Funcția de transformare asociată nodului i:

$$f_i\left(\{x_j(t) | j \in N_i\}\right)$$

decide dacă la iterația curentă nodul i devine sau nu lider.

Funcția f_i se reduce la una sau mai multe operații asupra memoriei locale M_i a nodului P_i.

Alegere Lider (Leader Election)

Starea locală a nodului i specifică calitatea de lider/non-lider:

$$x_i(t) \in \{lider, non-lider\}$$

Funcția de transformare asociată nodului i:

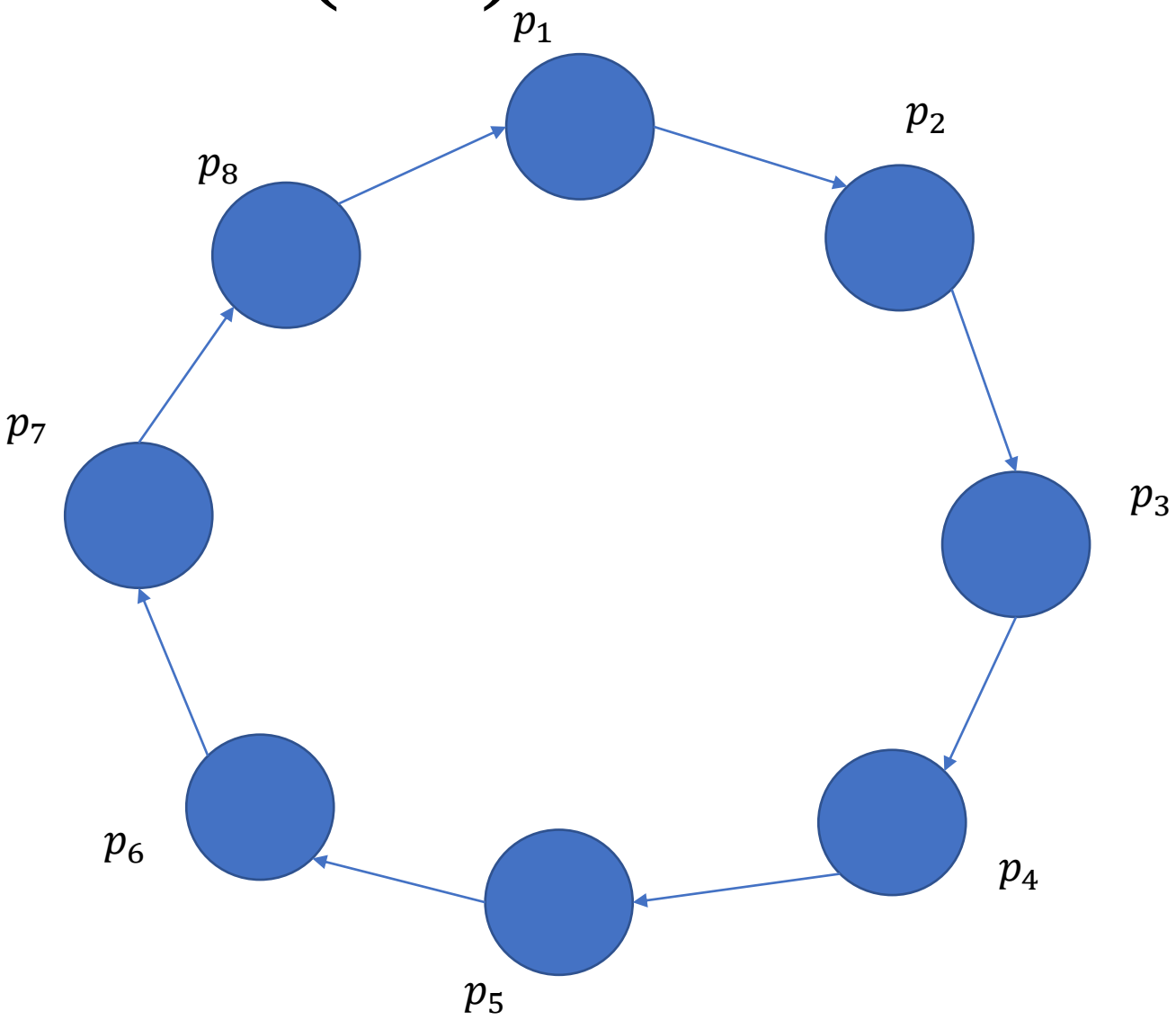
$$f_i\left(\{x_j(t) | j \in N_i\}\right)$$

decide dacă la iterația curentă nodul i devine sau nu lider.

Asimptotic, soluția problemei este aducerea sistemului în starea:

$$x_i^* = \begin{cases} lider, & i = argmax_j\ id_j \\ non-lider, & altfel \end{cases}$$

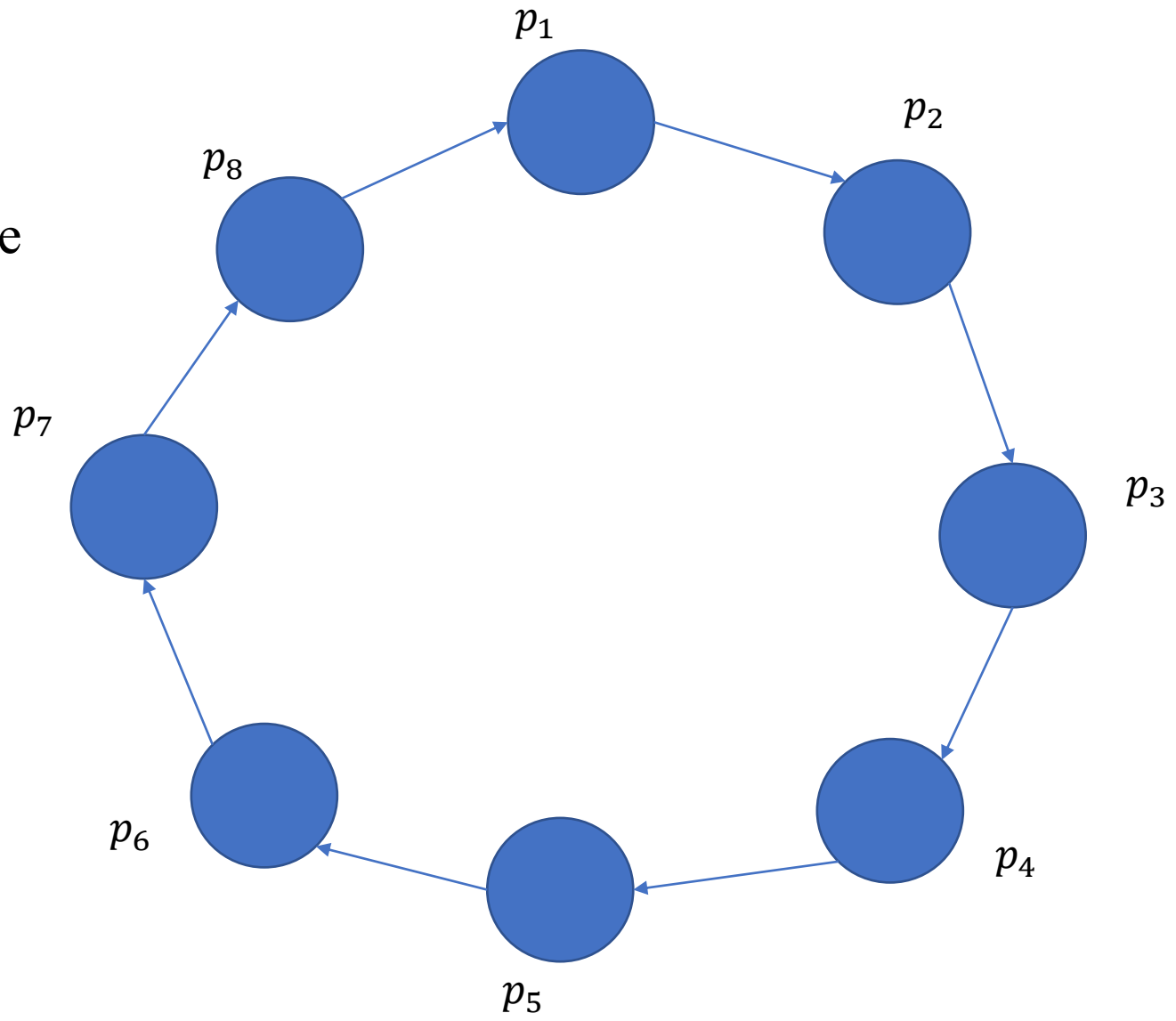
Alegere Lider (AL)



Alegere Lider (AL)

Ipoteze vedere locală:

1. Topologie inel unidirecțional (P_i cunoaște poziția relativă în inel)
2. Nodul P_i se identifică cu id_i
3. Nodul P_i cunoaște nr. de noduri n



Problema se reduce la: *specifică un program SPMD (f_i) astfel încât, într-un număr minim de iterații să asigurăm convergența stării globale la starea optimă, i.e.* $x_i(T) = x_i^*$.

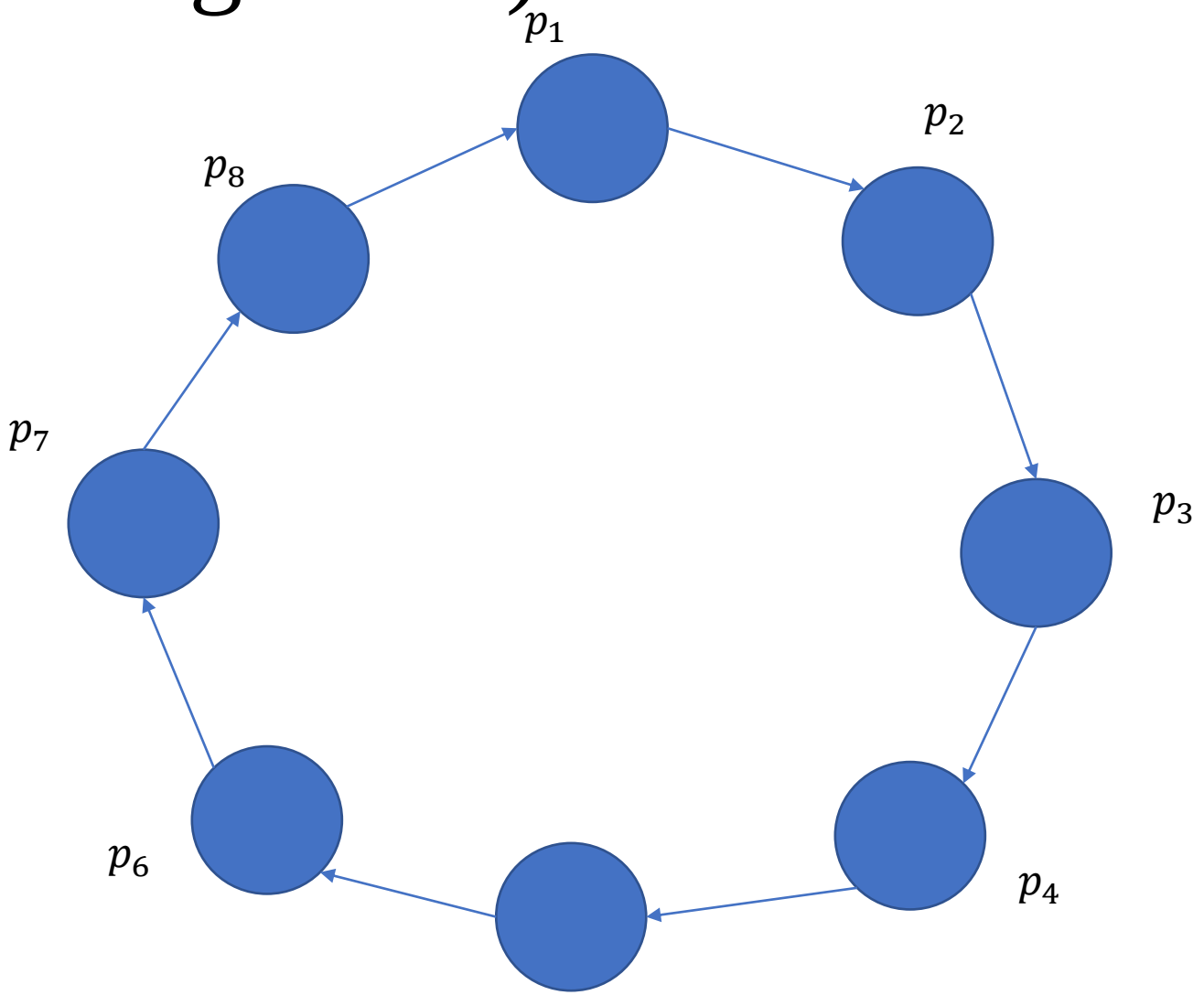
Alegere Lider (coordonate globale)

Algoritm **AlegeLiderInel_cg**():

M_i: - int n (număr noduri)
- int i (index propriu)
- int id (id propriu)
- int id_max (id propriu)

% Faza I: max ID
- Calculează max$\{id_1, id_2, \dots, id_n\}$
- Rezultatul va fi stocat într-un nod particular

% Faza II: Difuzare Max ID (Broadcast)
- Rezultatul este difuzat peste tot inelul
- Stările x_i sunt ajustate conform rezultatului



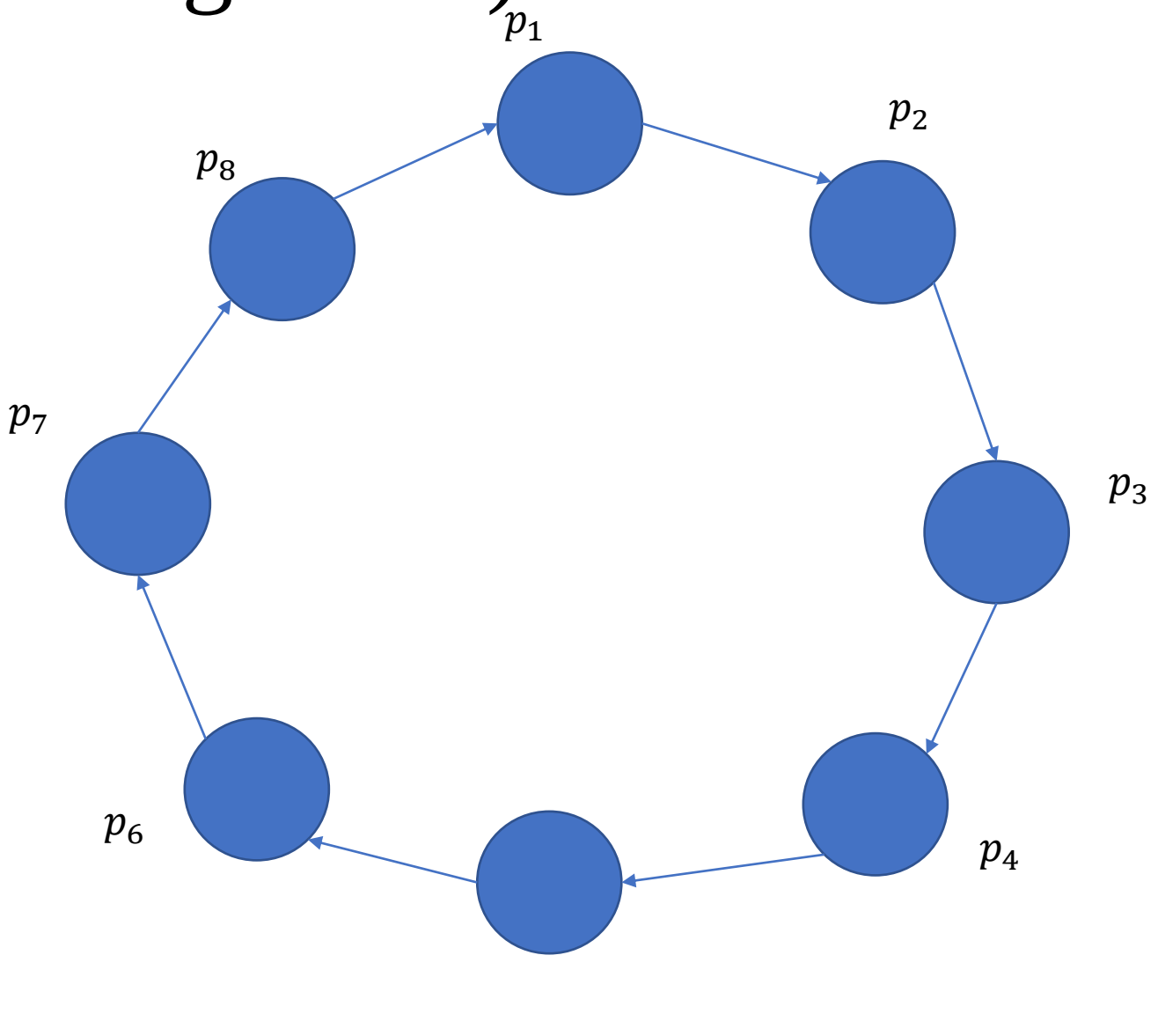
Alegere Lider (coordonate globale)

M_i: - int n (număr noduri)
- int i (index propriu)
- int id (id propriu)
- int id_max (id propriu)

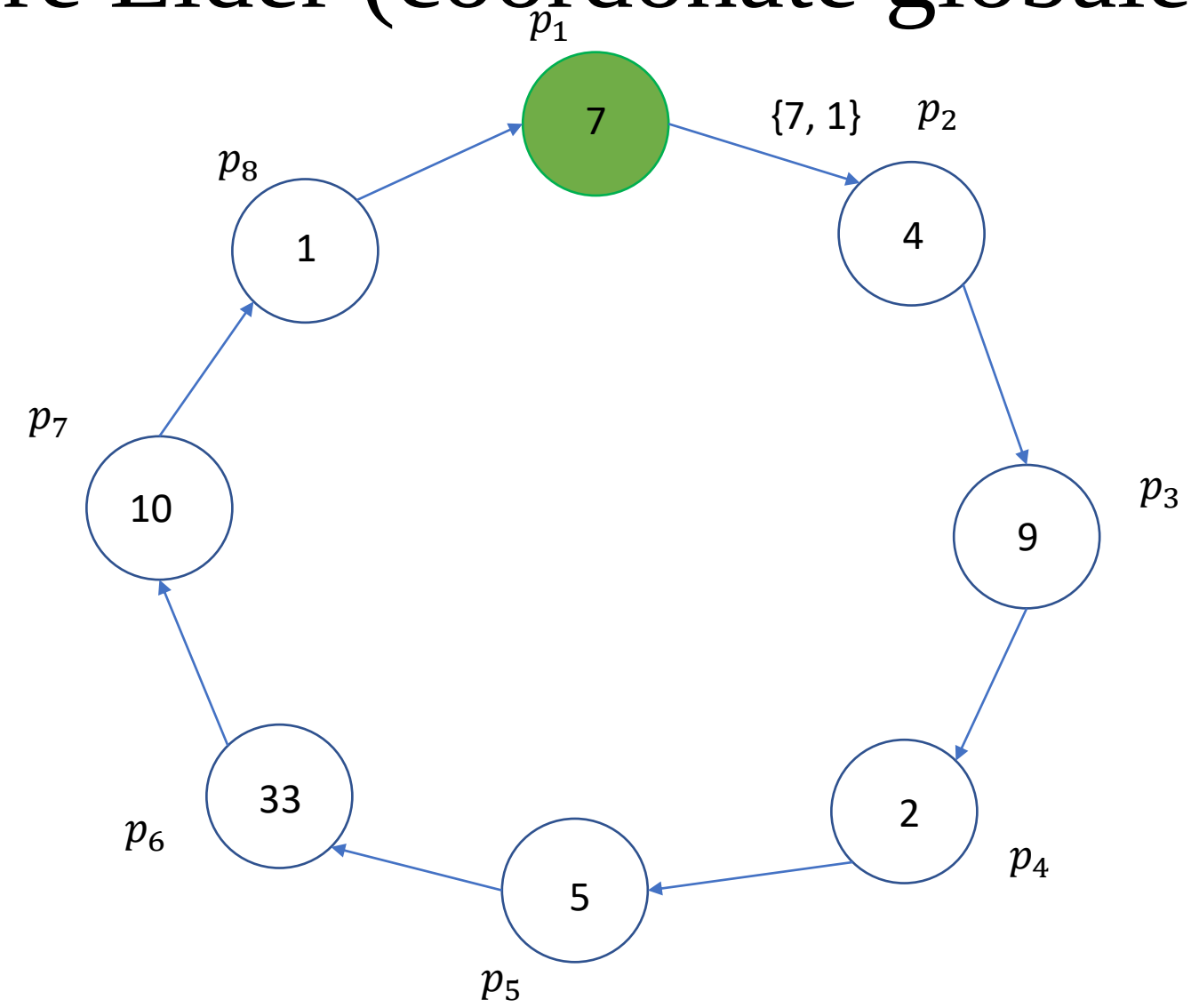
% Faza max ID

Functie transformare nod i $f_i()$:

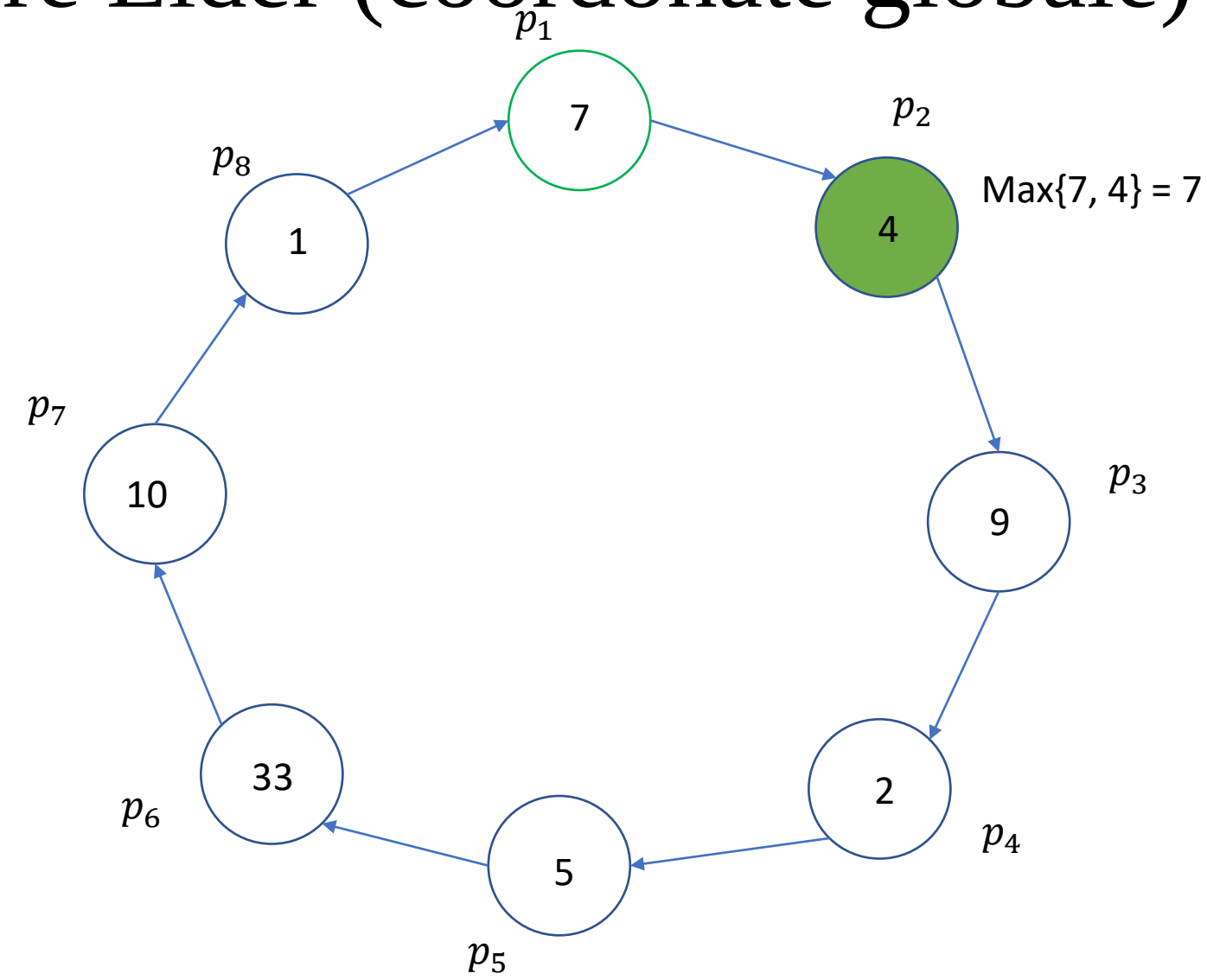
1. **If (i == 1):**
    1. send(id, 2);
    2. id_aux = recv(n);
2. **else:**
    1. id_aux = recv(index - 1 mod n);
    2. send(idmax, index + 1 mod n);
3. idmax = max(id, id_aux);



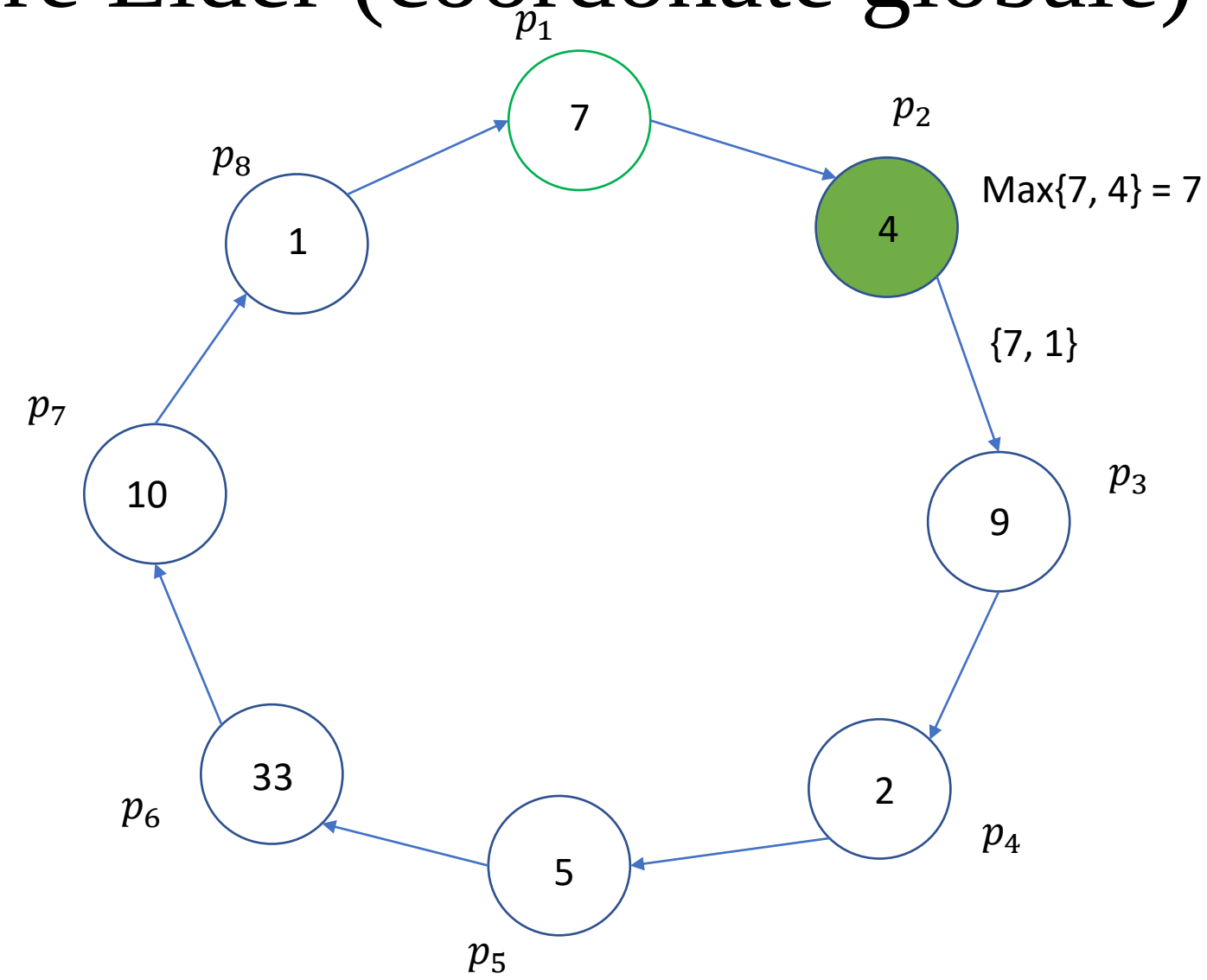
Alegere Lider (coordonate globale)



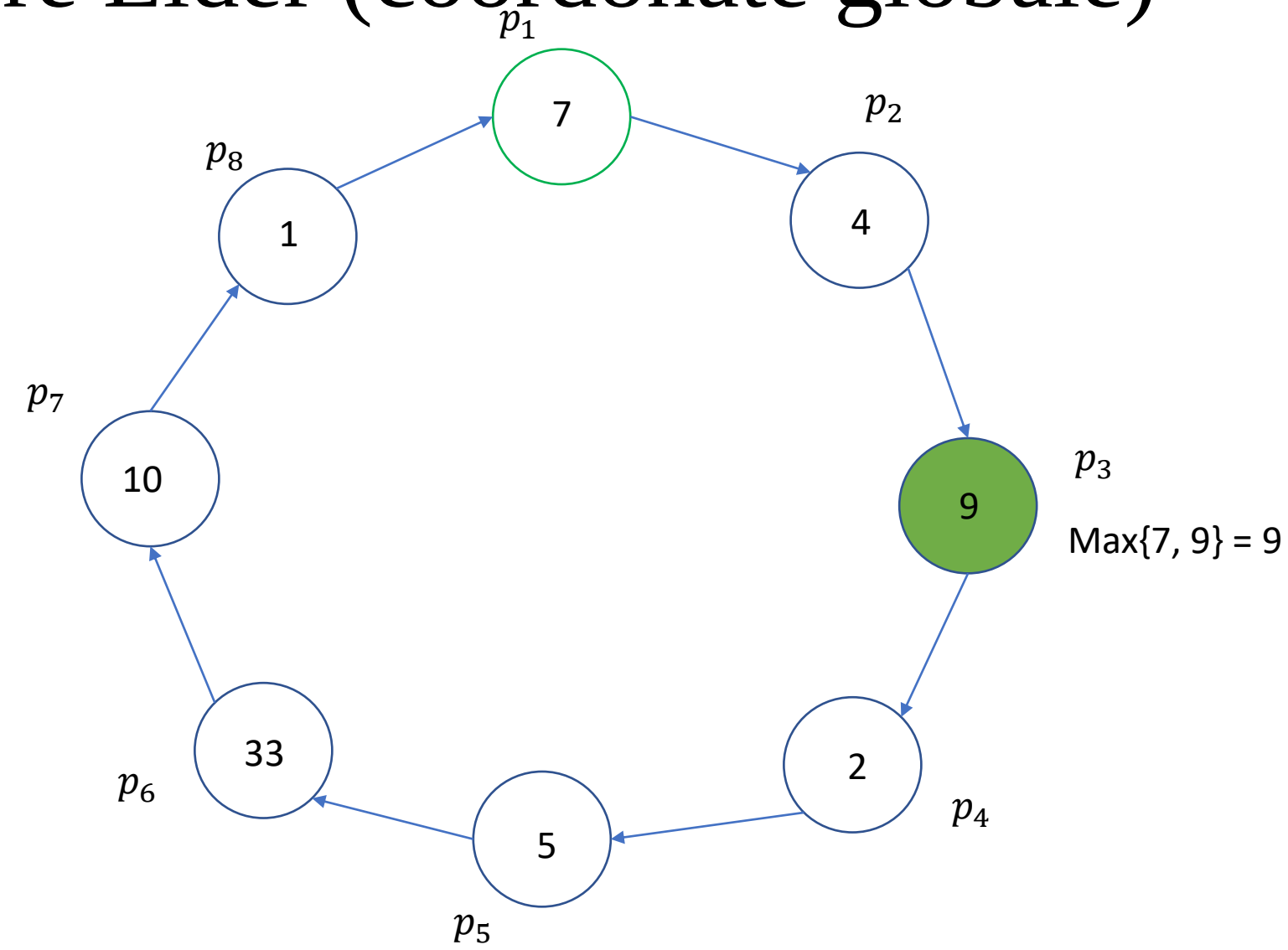
Alegere Lider (coordonate globale)



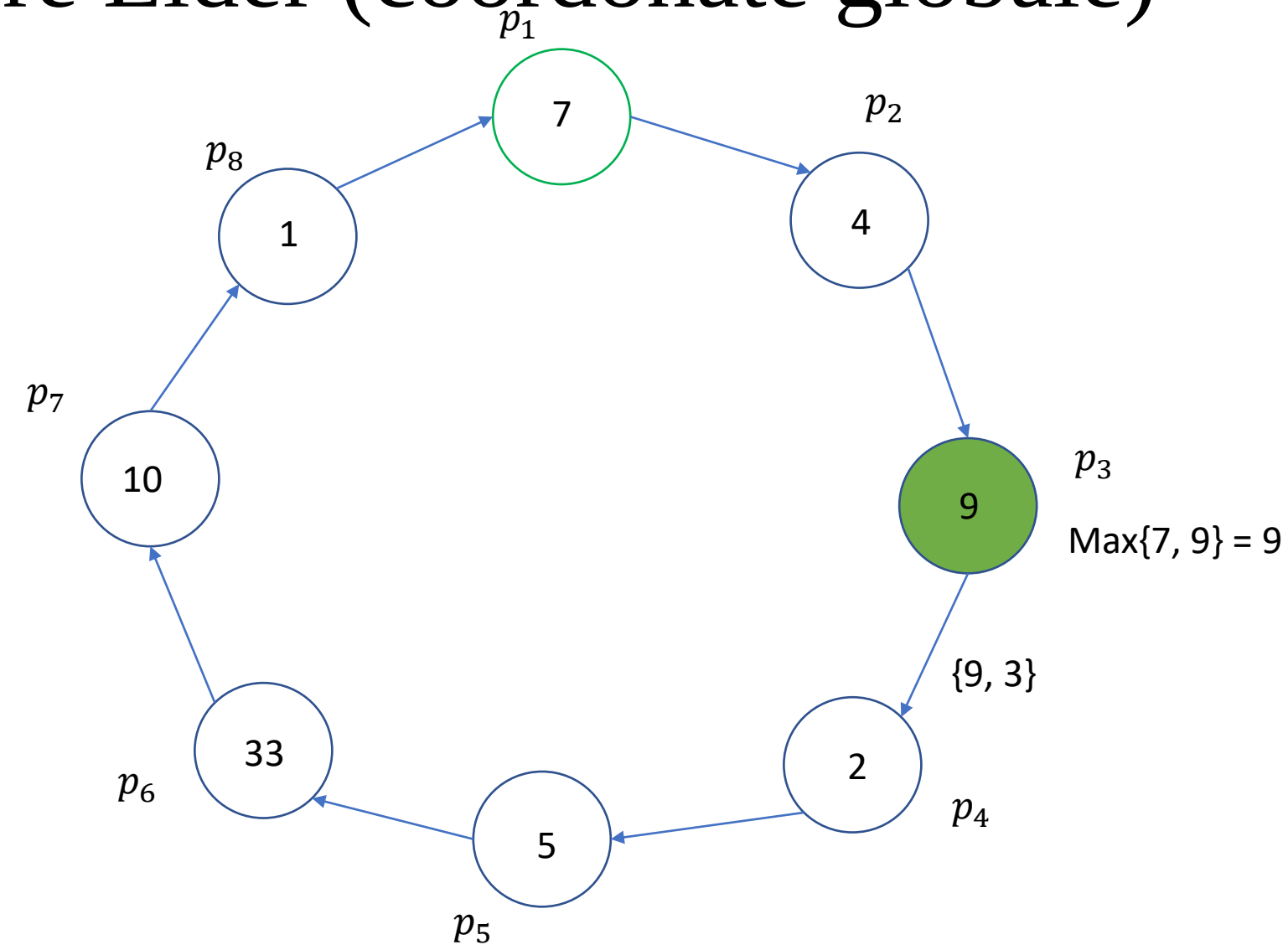
Alegere Lider (coordonate globale)



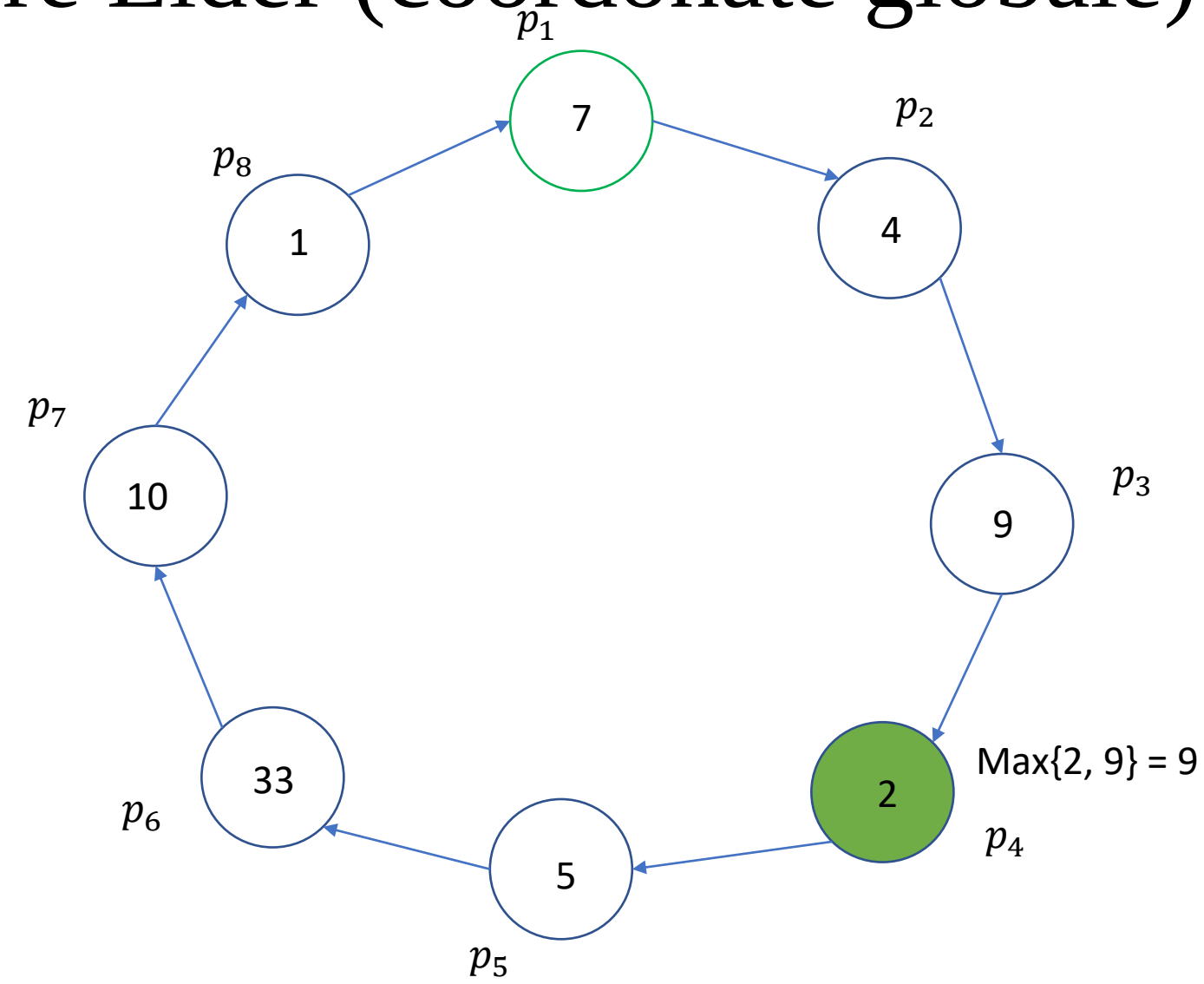
Alegere Lider (coordonate globale)



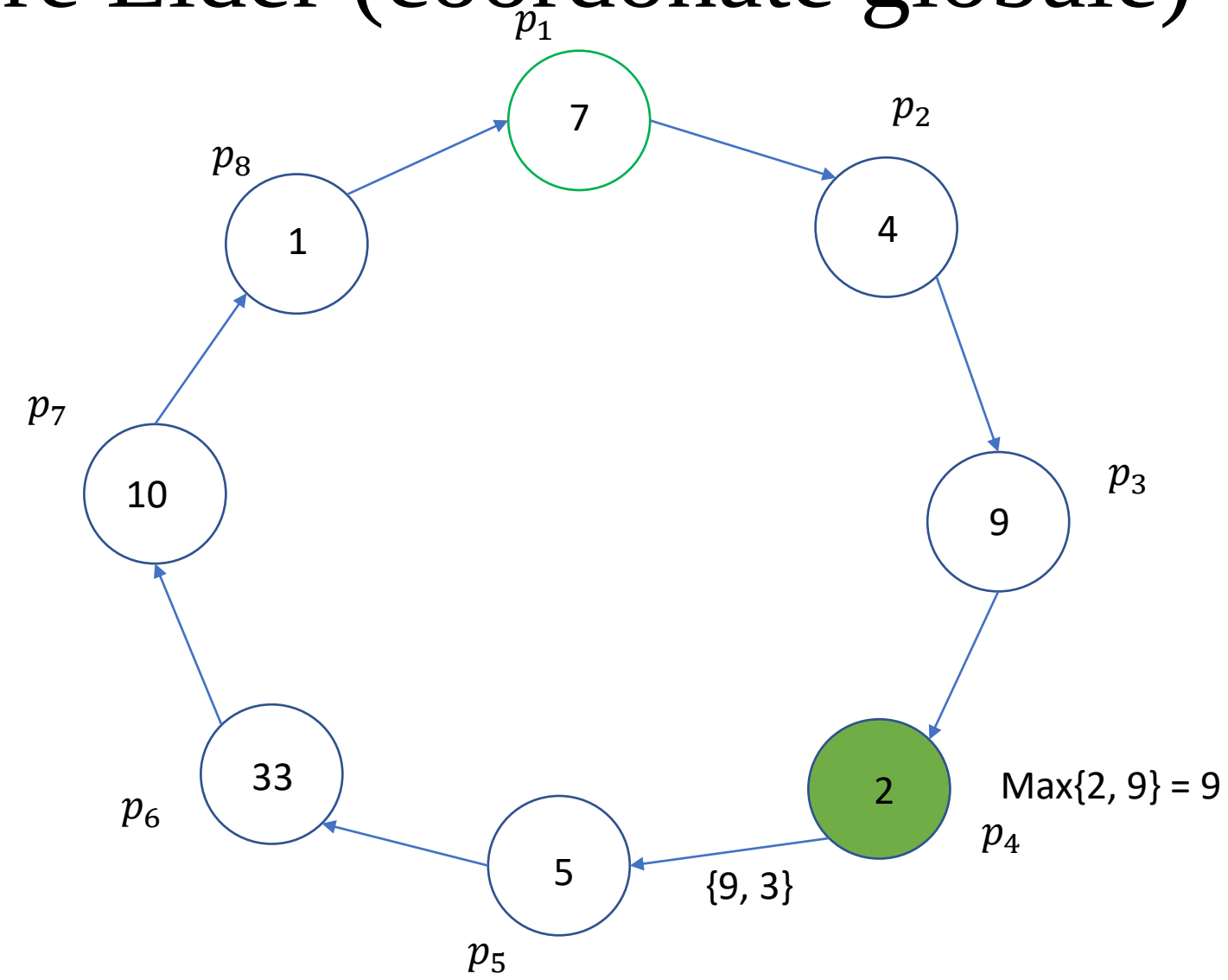
Alegere Lider (coordonate globale)



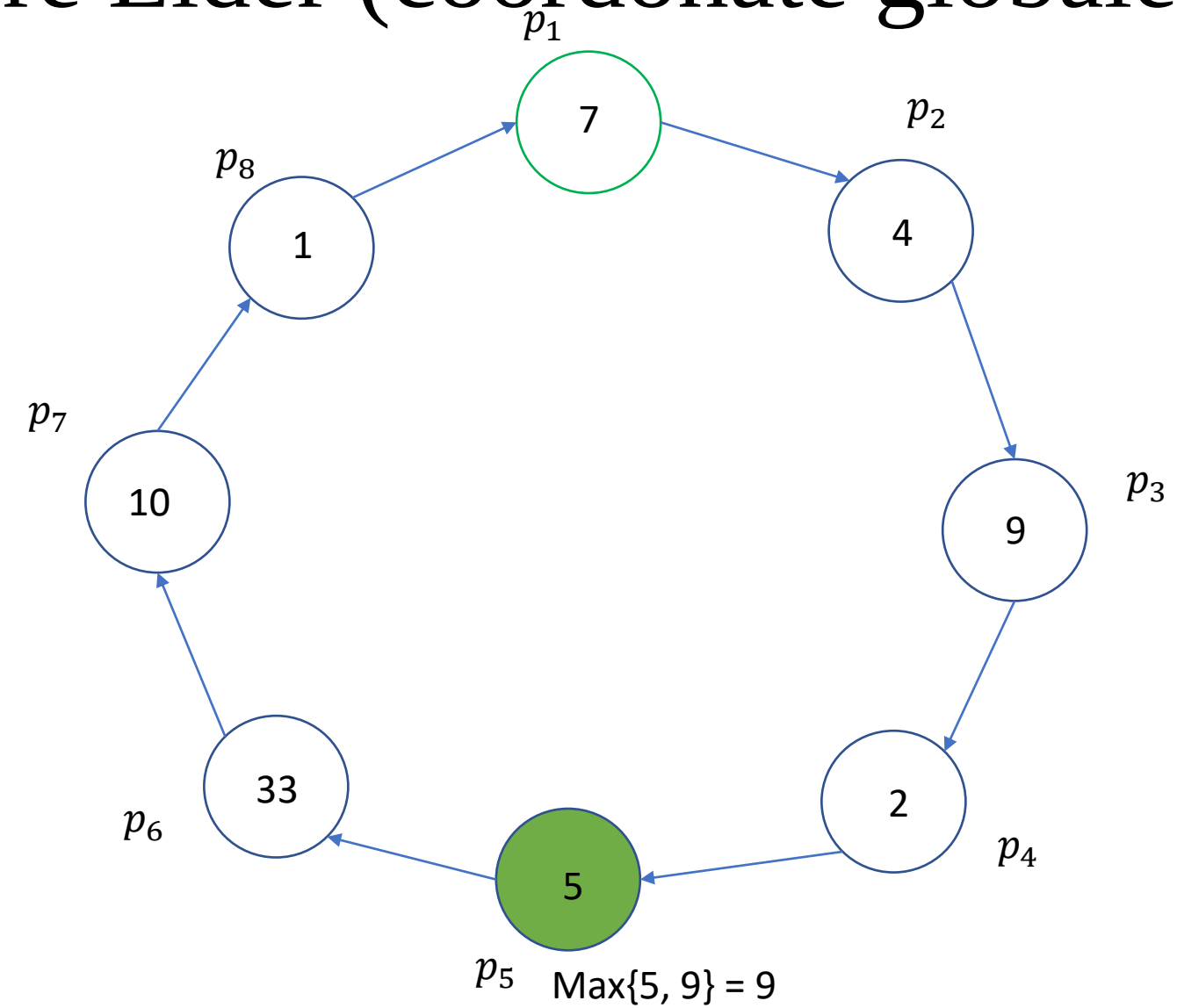
Alegere Lider (coordonate globale)



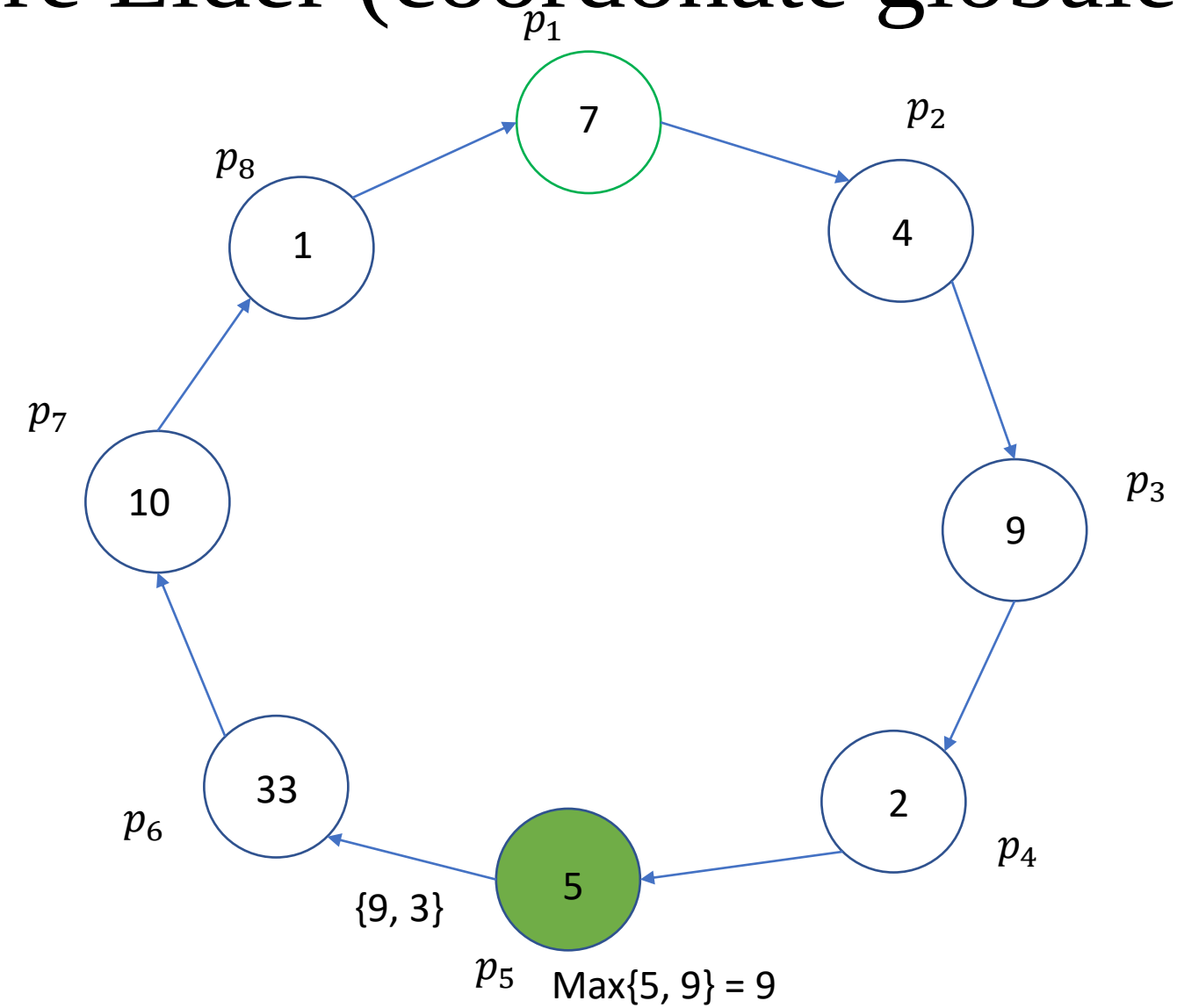
Alegere Lider (coordonate globale)



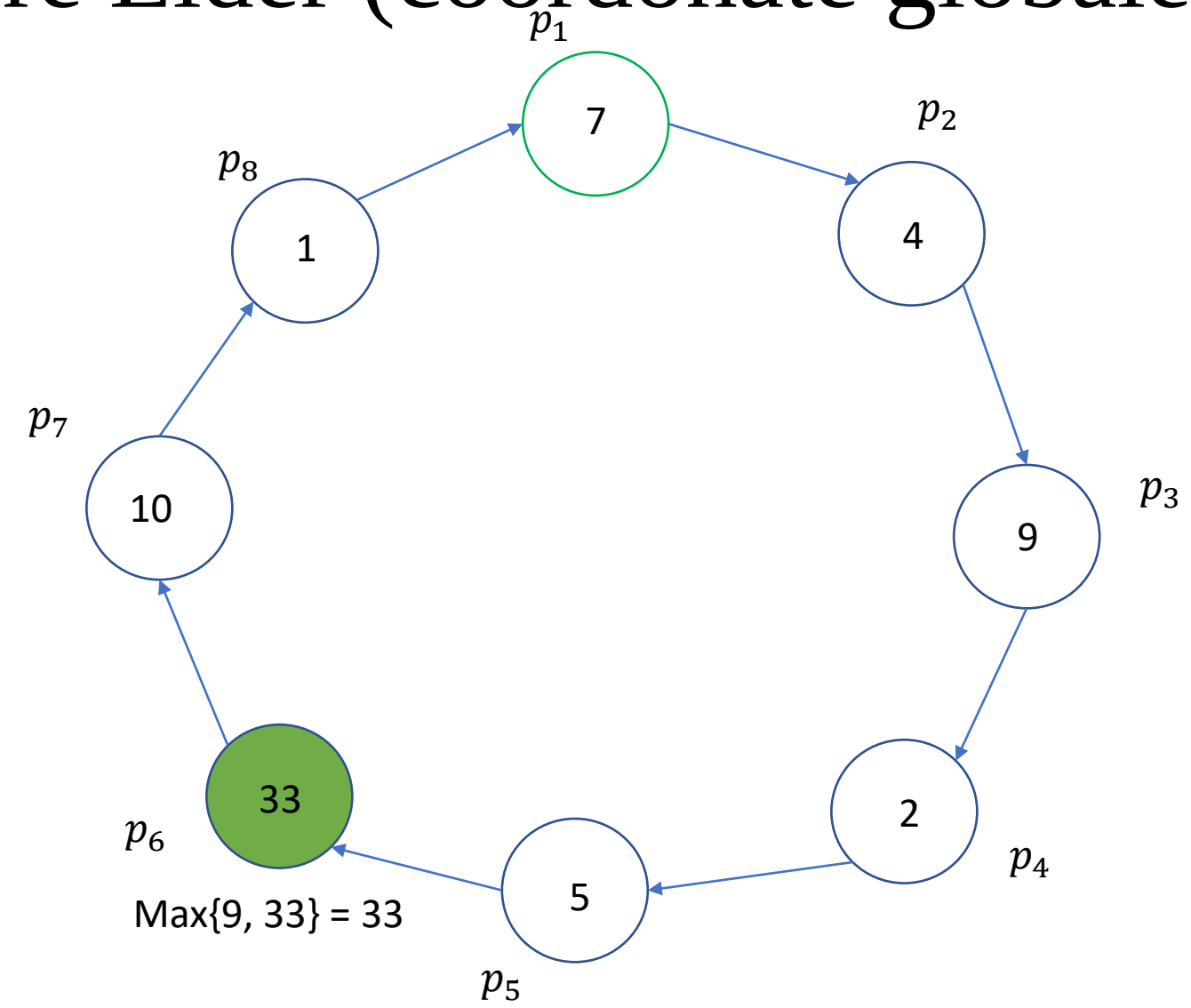
Alegere Lider (coordonate globale)



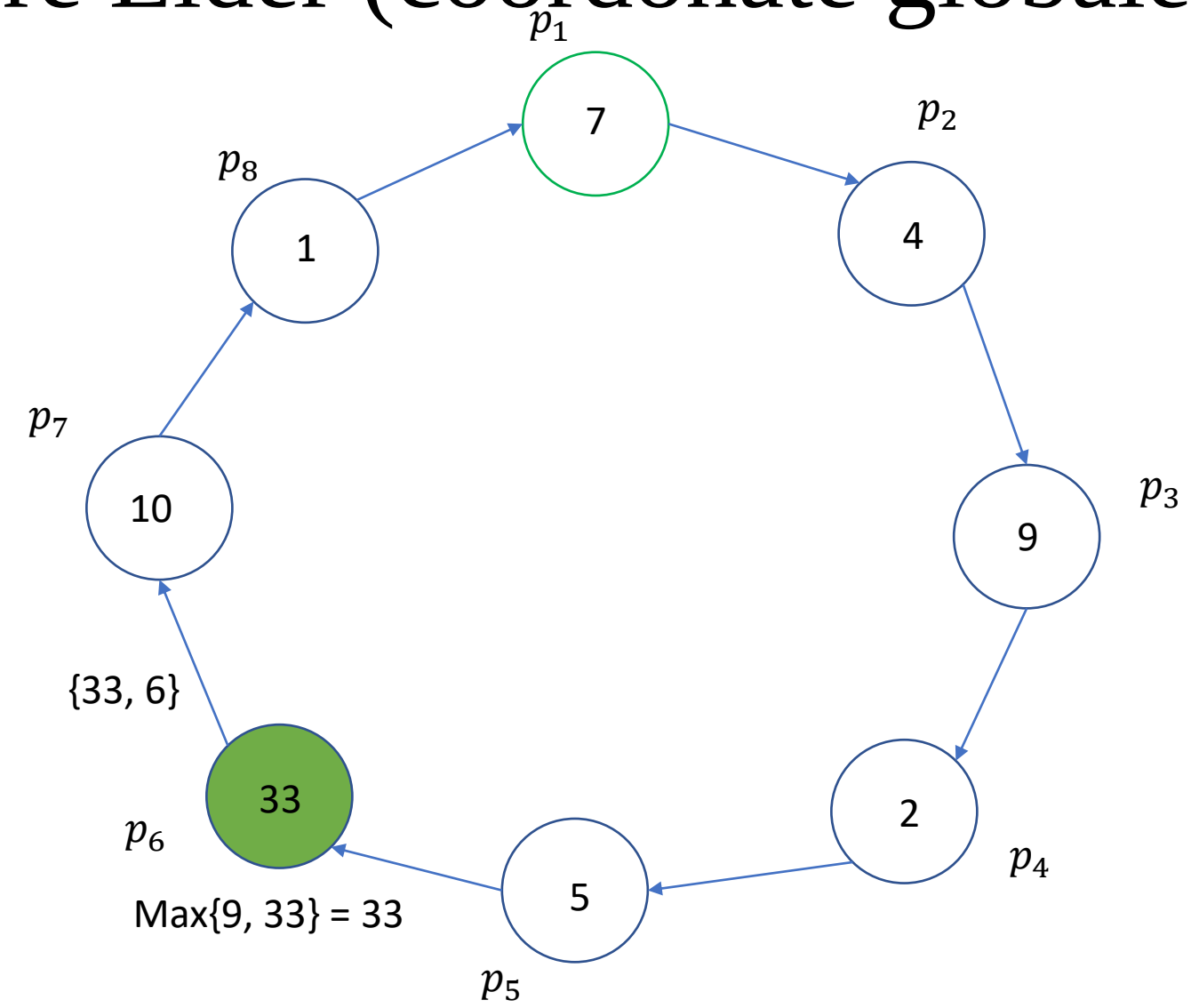
Alegere Lider (coordonate globale)



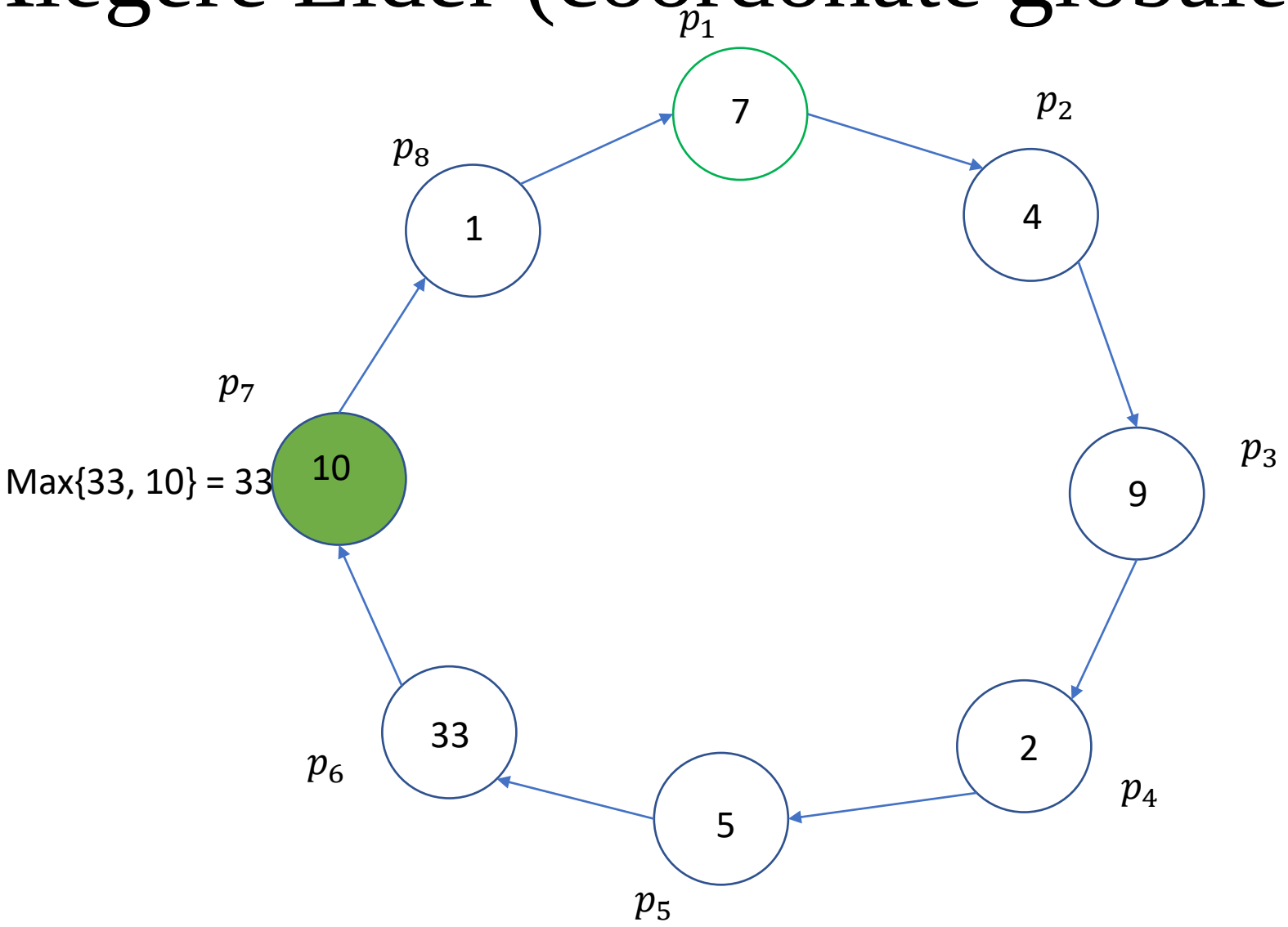
Alegere Lider (coordonate globale)



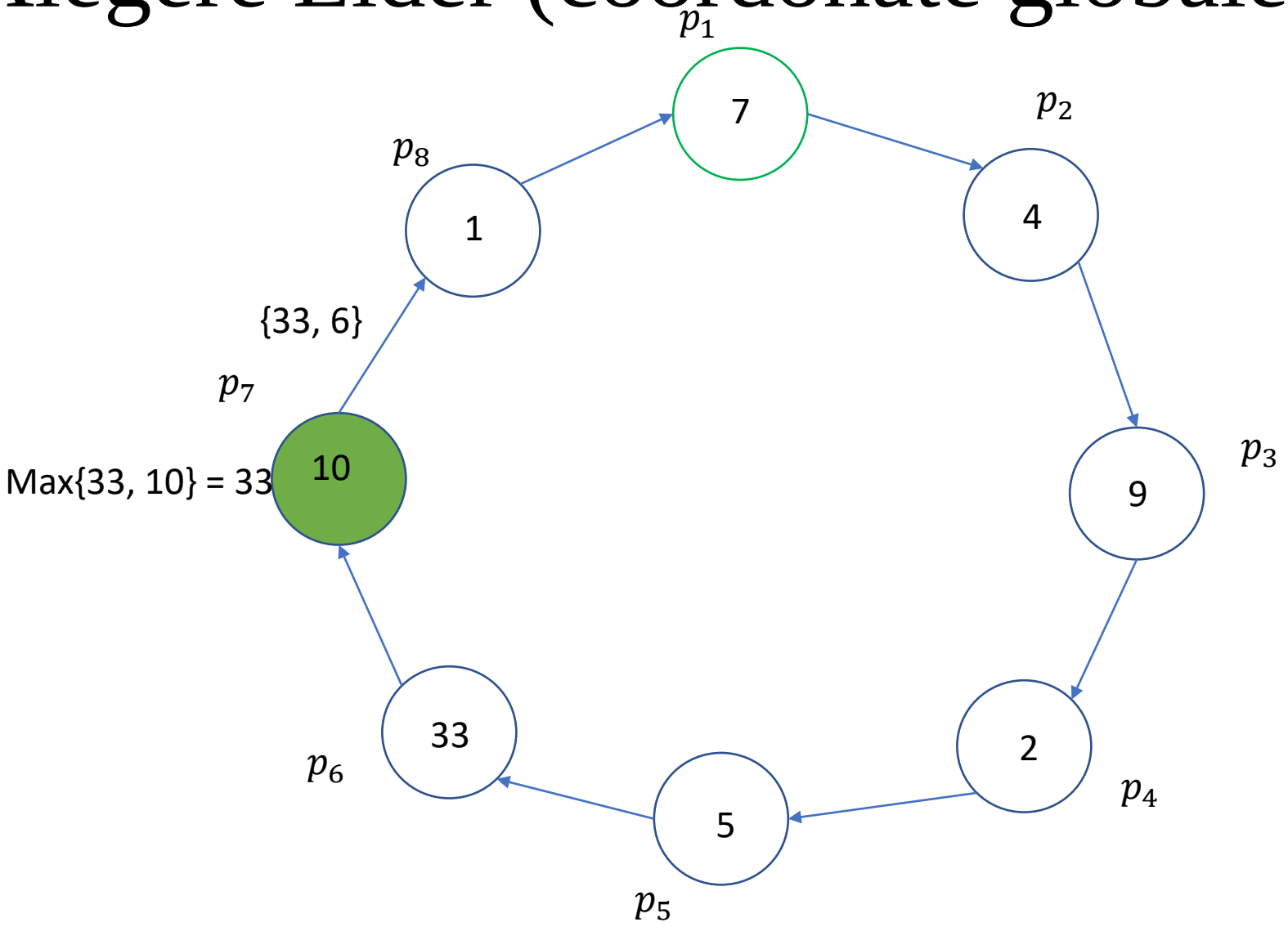
Alegere Lider (coordonate globale)



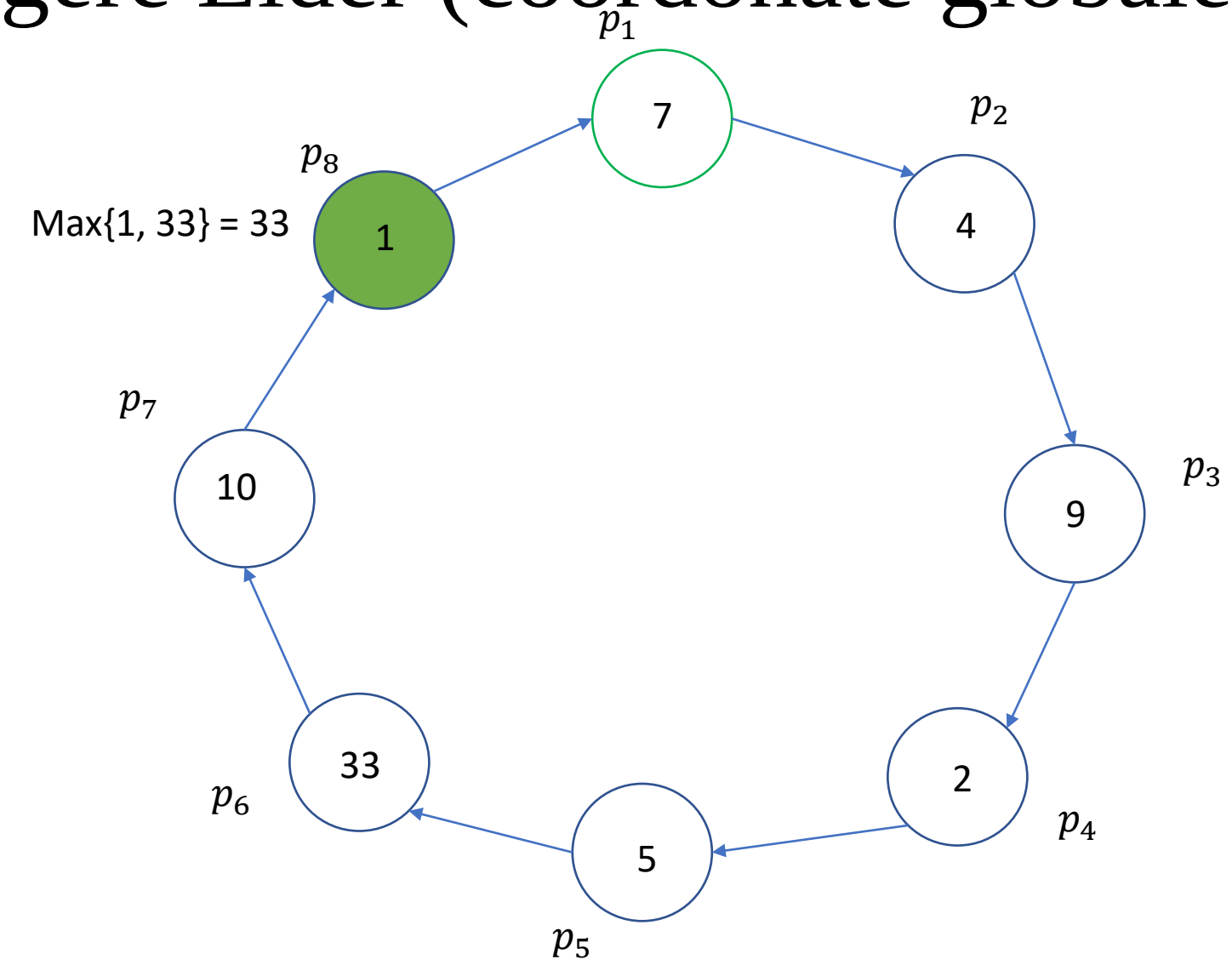
Alegere Lider (coordonate globale)



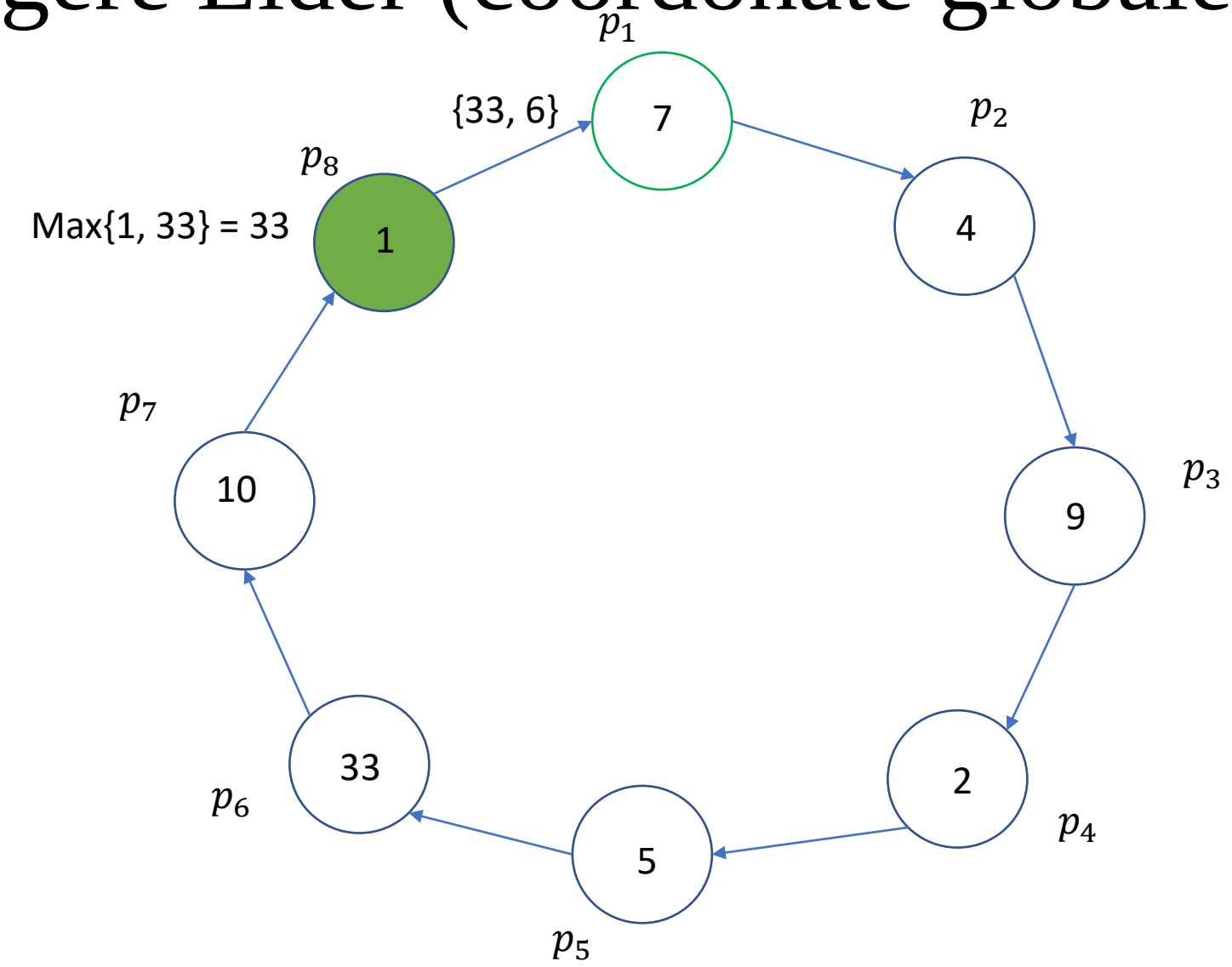
Alegere Lider (coordonate globale)



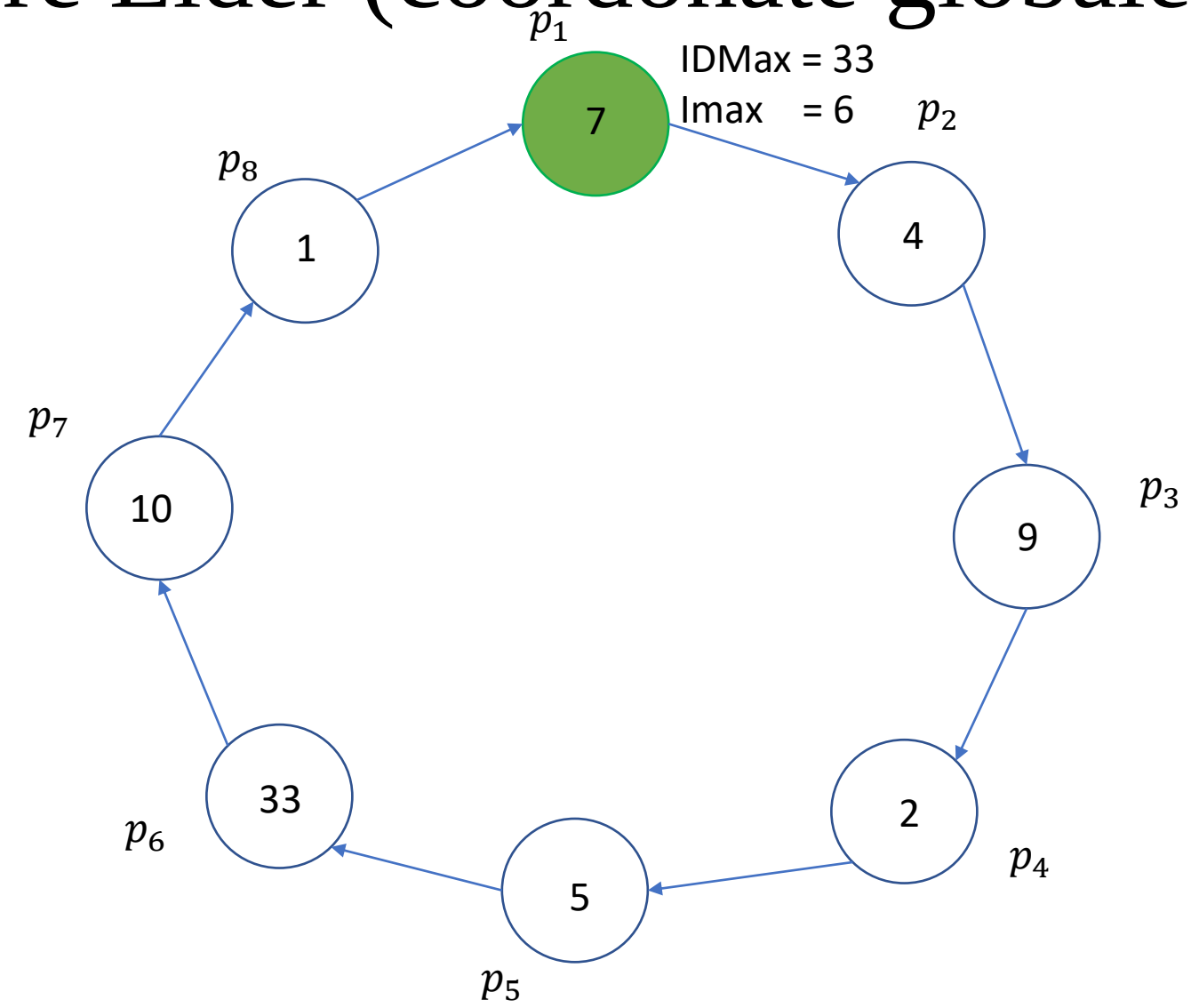
Alegere Lider (coordonate globale)



Alegere Lider (coordonate globale)



Alegere Lider (coordonate globale)



Alegere Lider (coordonate globale)

Memorie locală nod i:
- int n (număr noduri)
- int i (index propriu)
- int id (id propriu)
- int id_max (id propriu)

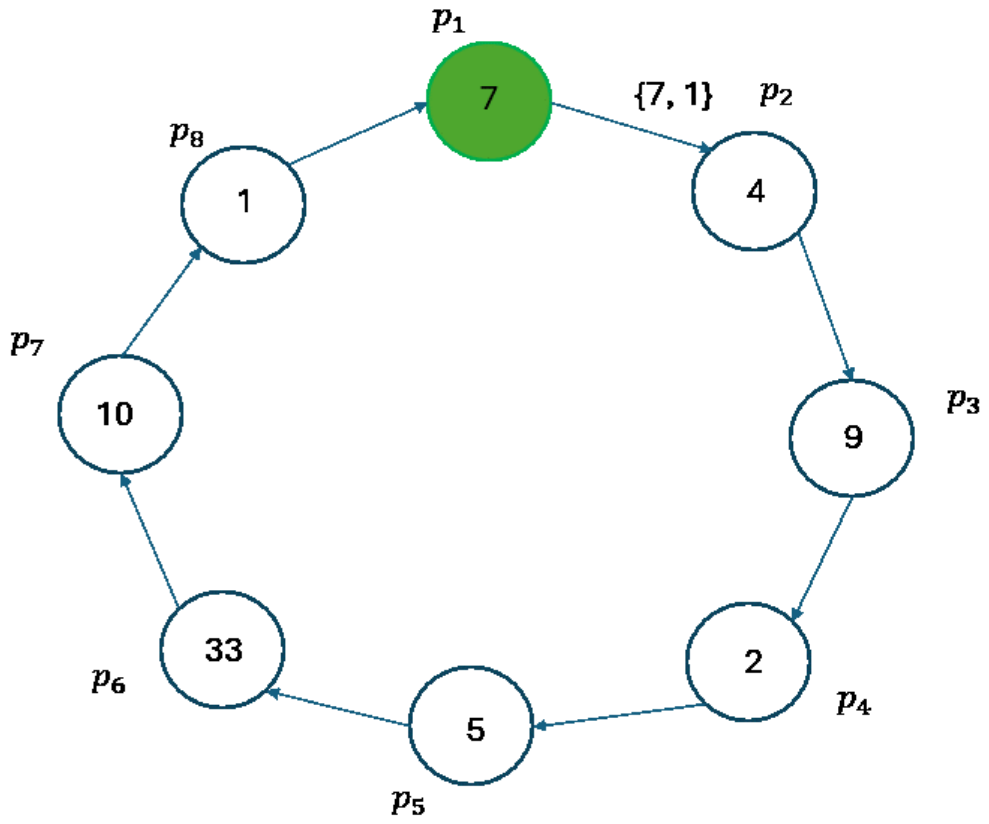
% Faza I: Max ID
…
% Faza II: Difuzare Max ID (Broadcast)

Functie transformare nod i $f_i()$:

1. **If (i == 1):**
    1. send({idmax, imax}, 2);

    Else If (i == n-1):
    1. {idmax, imax} = recv(index - 1 mod n);

    Else
    1. {idmax, imax} = recv(index - 1 mod n);
    2. send({idmax, imax}, index + 1 mod n);

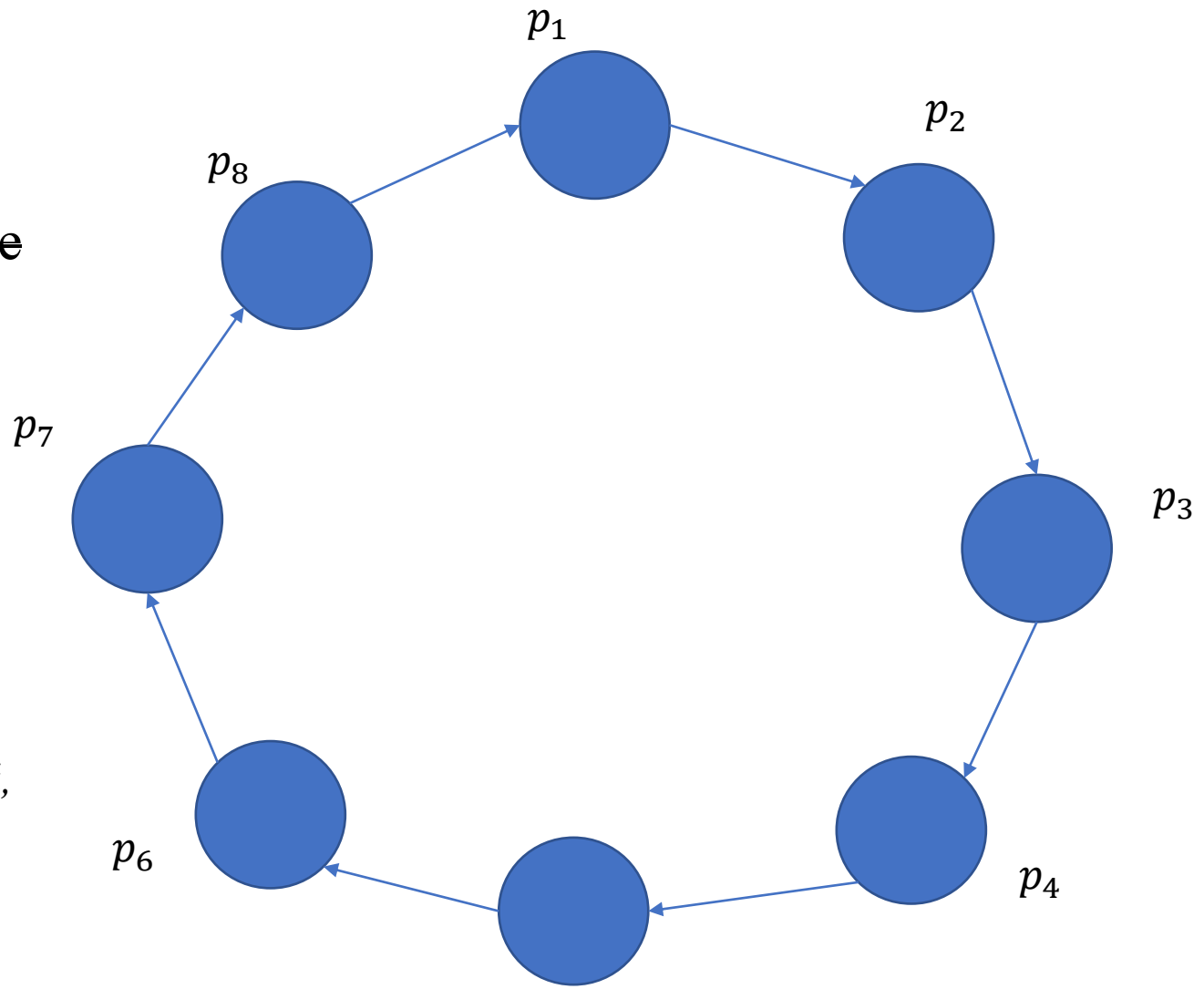


Alegere Lider (AL)

Ipoteze vedere locală:

1. Topologie inel unidirecțional ~~(P_t cunoaște poziția relativă în inel)~~
2. Nodul P_i se identifică cu id_i
3. ~~Nodul P_t cunoaște nr. de noduri n~~

- Ipoteze mai realiste (rețele peer-to-peer)
- Topologia și numărul de noduri sunt statice
- Problema se reduce la: *specifică un program SPMD (f_i) astfel încât, într-un număr minim de iterații să asigurăm convergența stării globale la starea optimă, i.e.* $x_i(T) = x_i^*$.

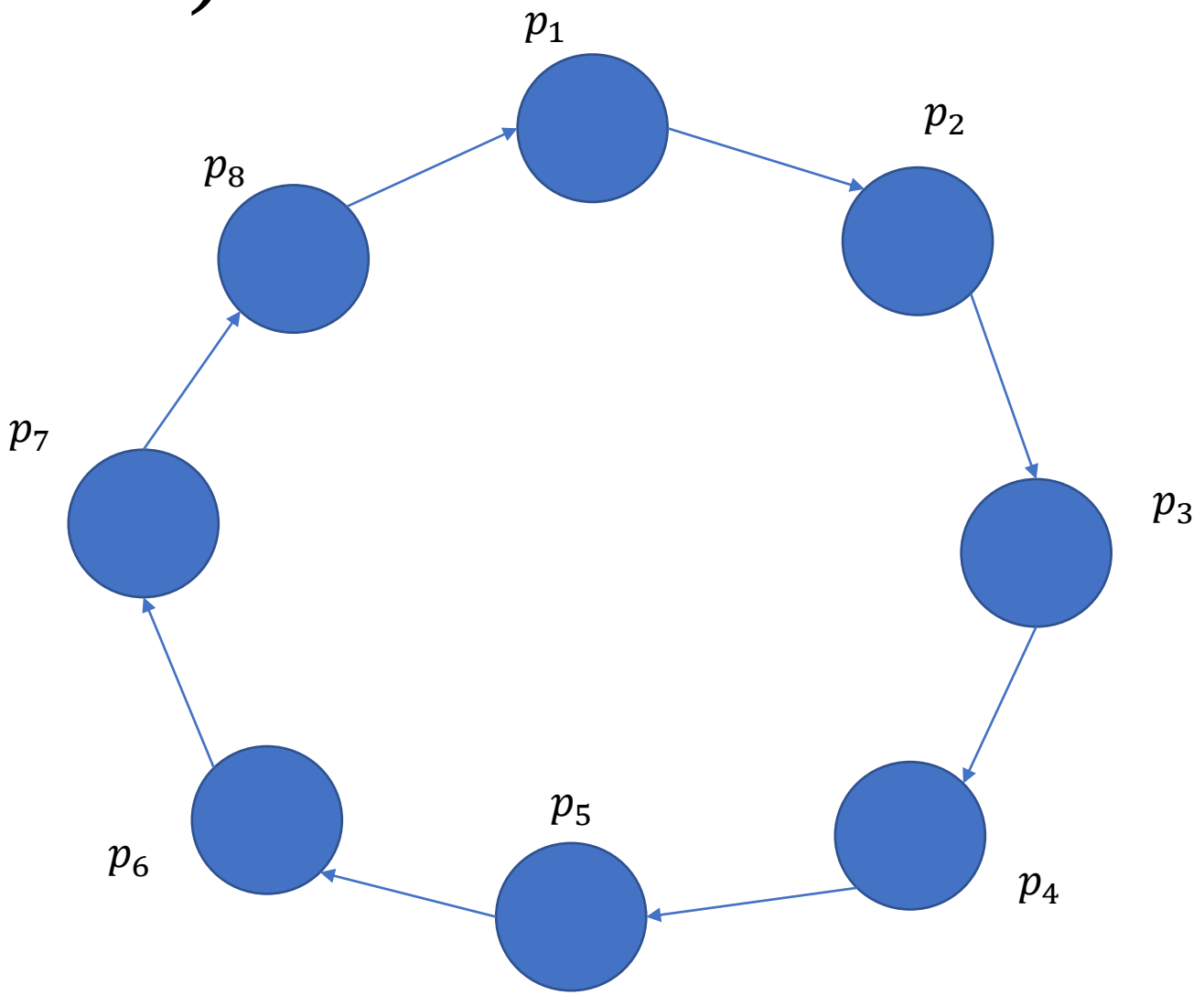


Algoritmul Flooding (LCR)

Algoritm **Flooding**():

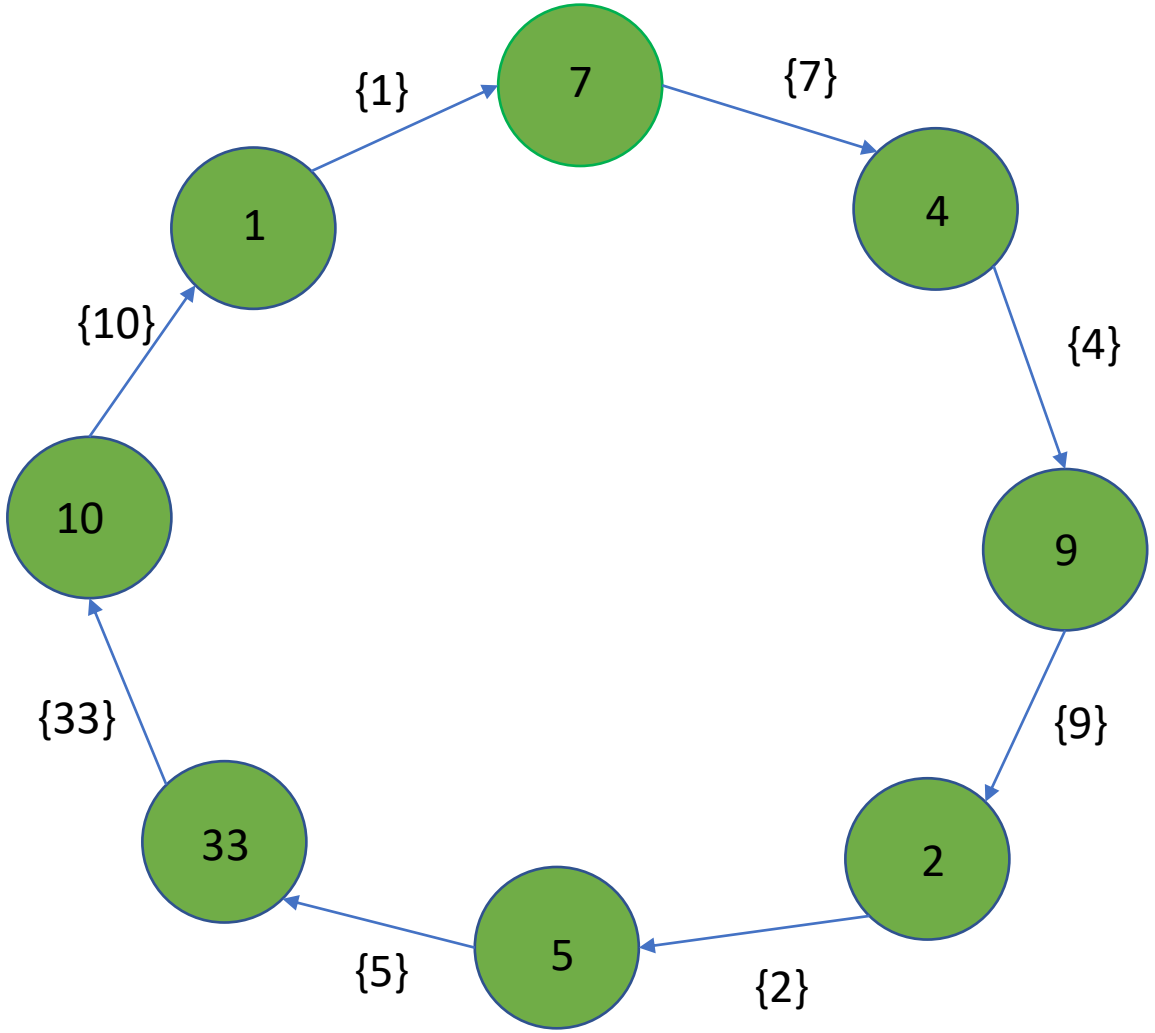
M_i: - int id (id propriu)
- int send_id (var auxiliară), inițial id
- status ∈ {lider, non-lider}, inițial non-lider

Funcție transformare nod i ():
1. send(send_id, index + 1 mod n);
2. recv_id = recv(index - 1 mod n);
3. **If** (recv_id > id):
    1. send_id := recv_id;
4. **ElseIf** (recv_id == id): status = leader;



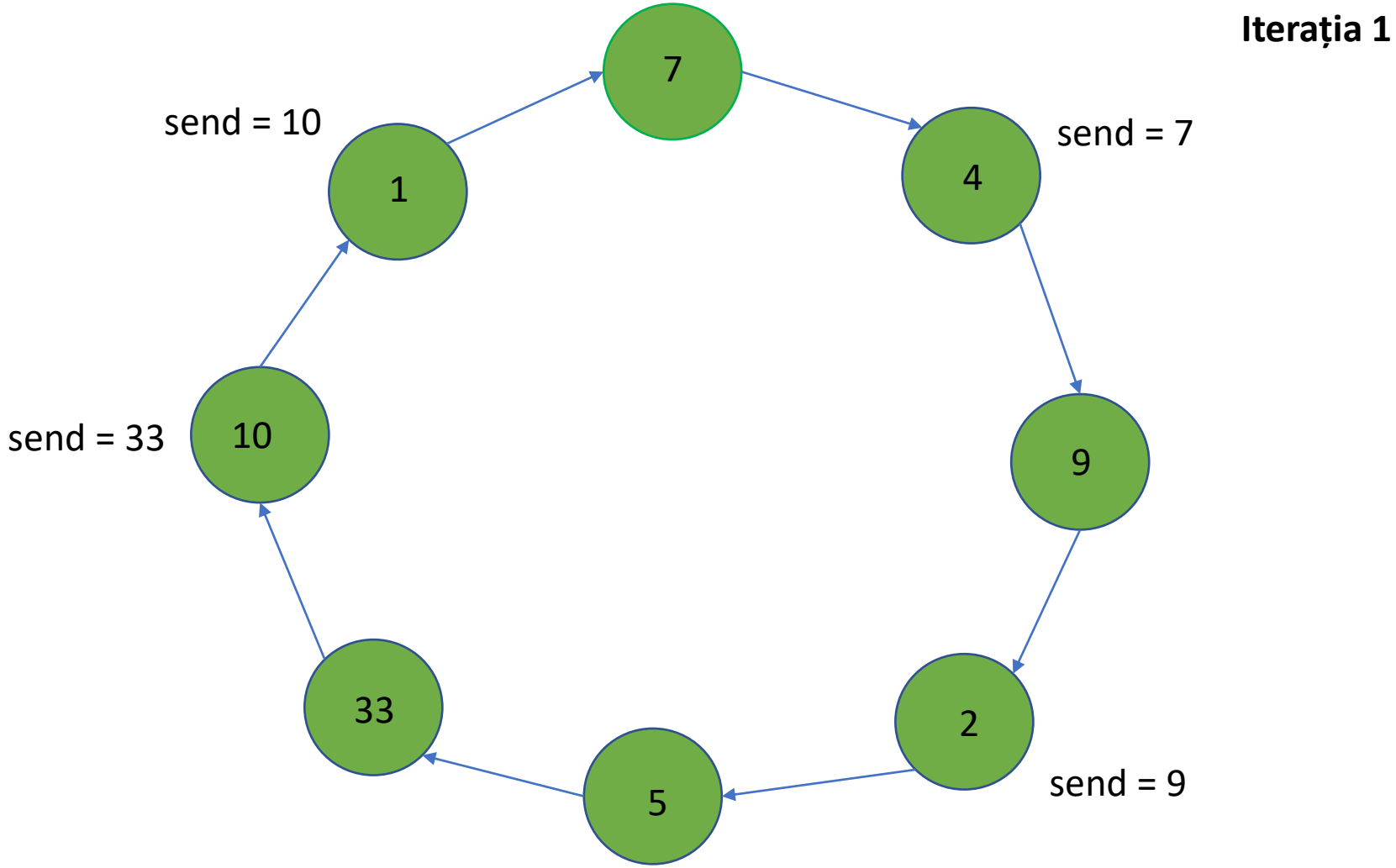
Gerard Le Lann. *Distributed systems – toward a formal approach*. In Bruce Gilchrist, editor, Information Processing 77, Proceedings of IFIP Congress, 155-160, 1977.

Algoritmul Flooding (LCR)

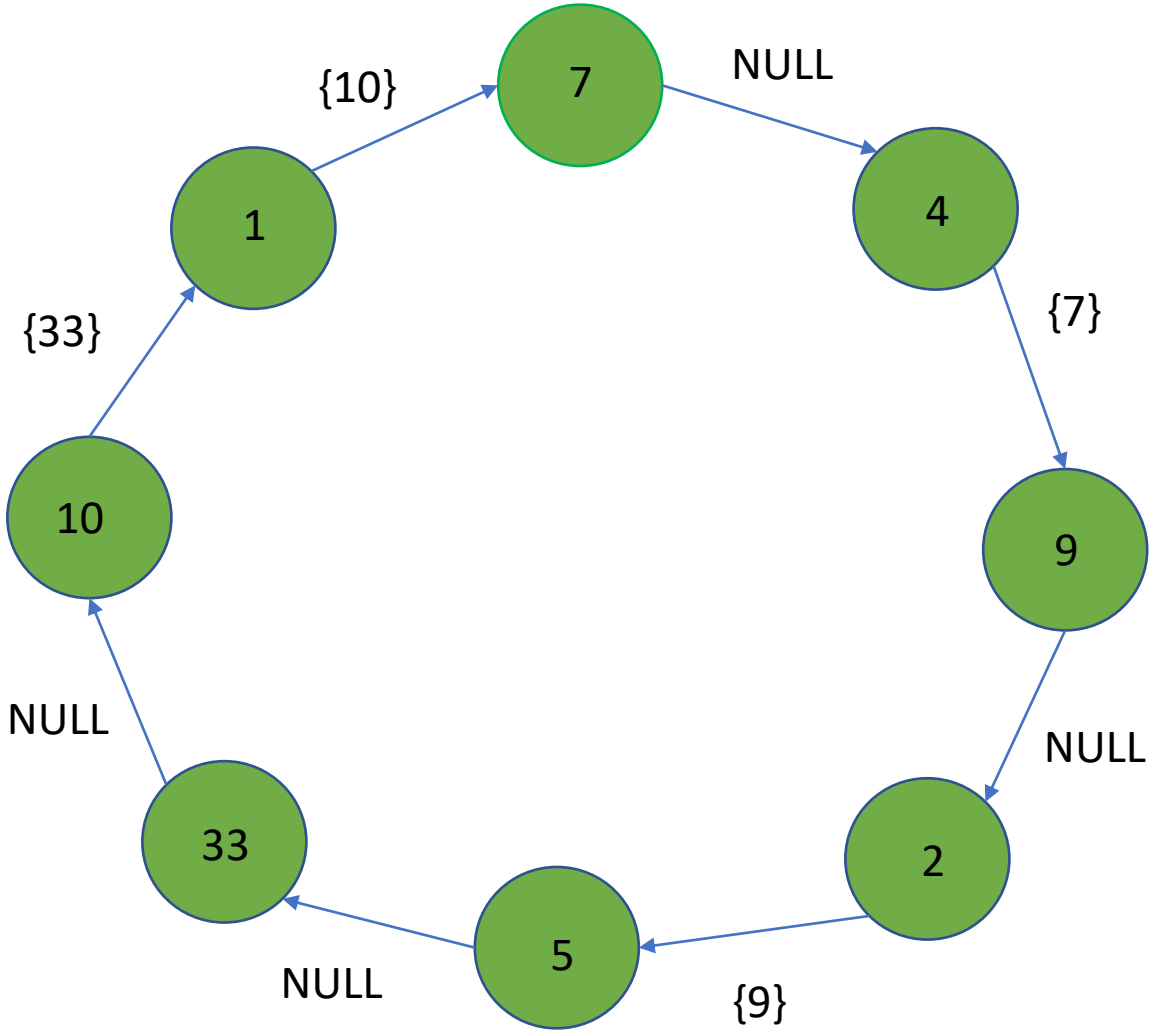


Iterația 1

Algoritmul Flooding (LCR)

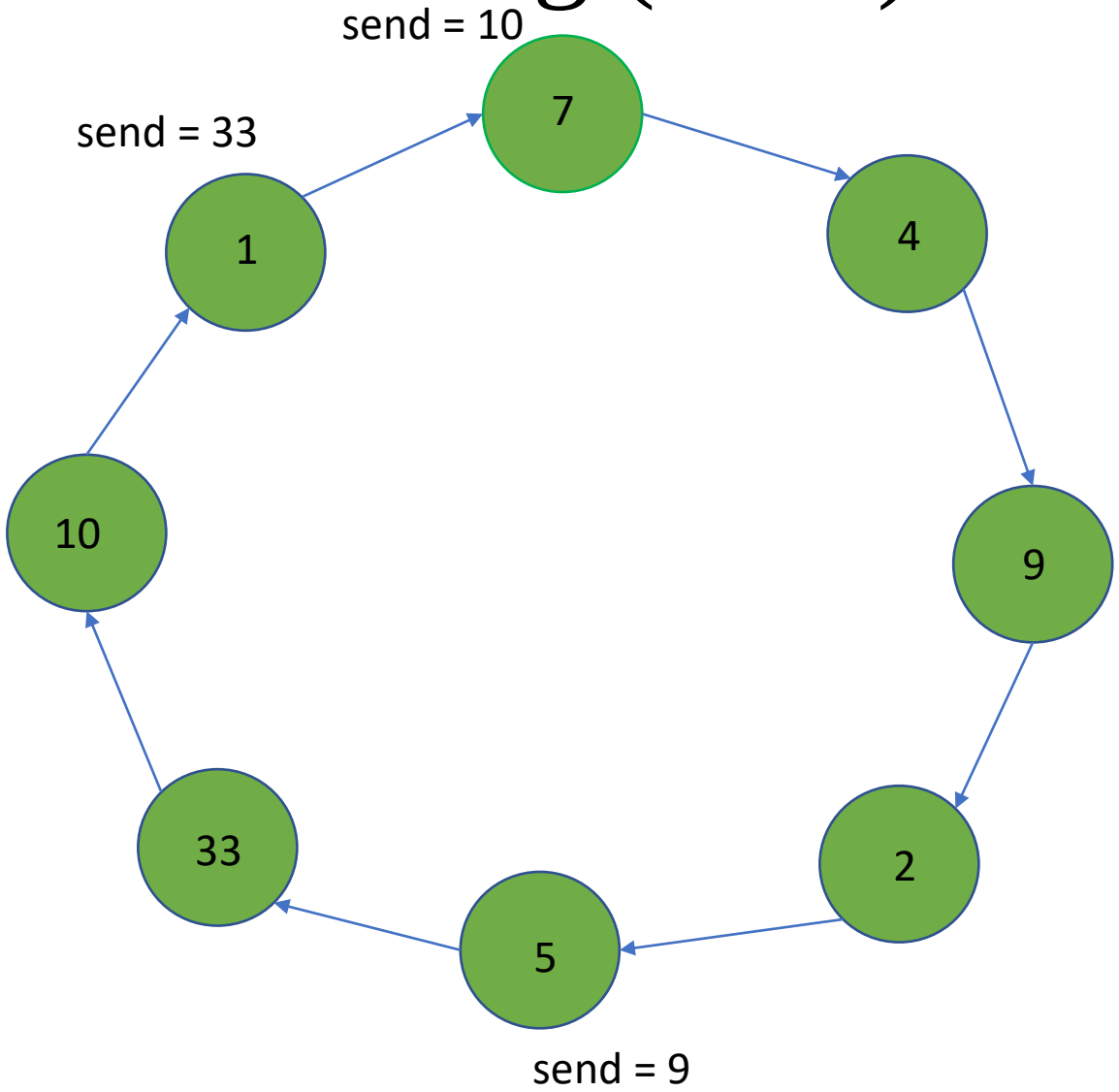


Algoritmul Flooding (LCR)



Iterația 2

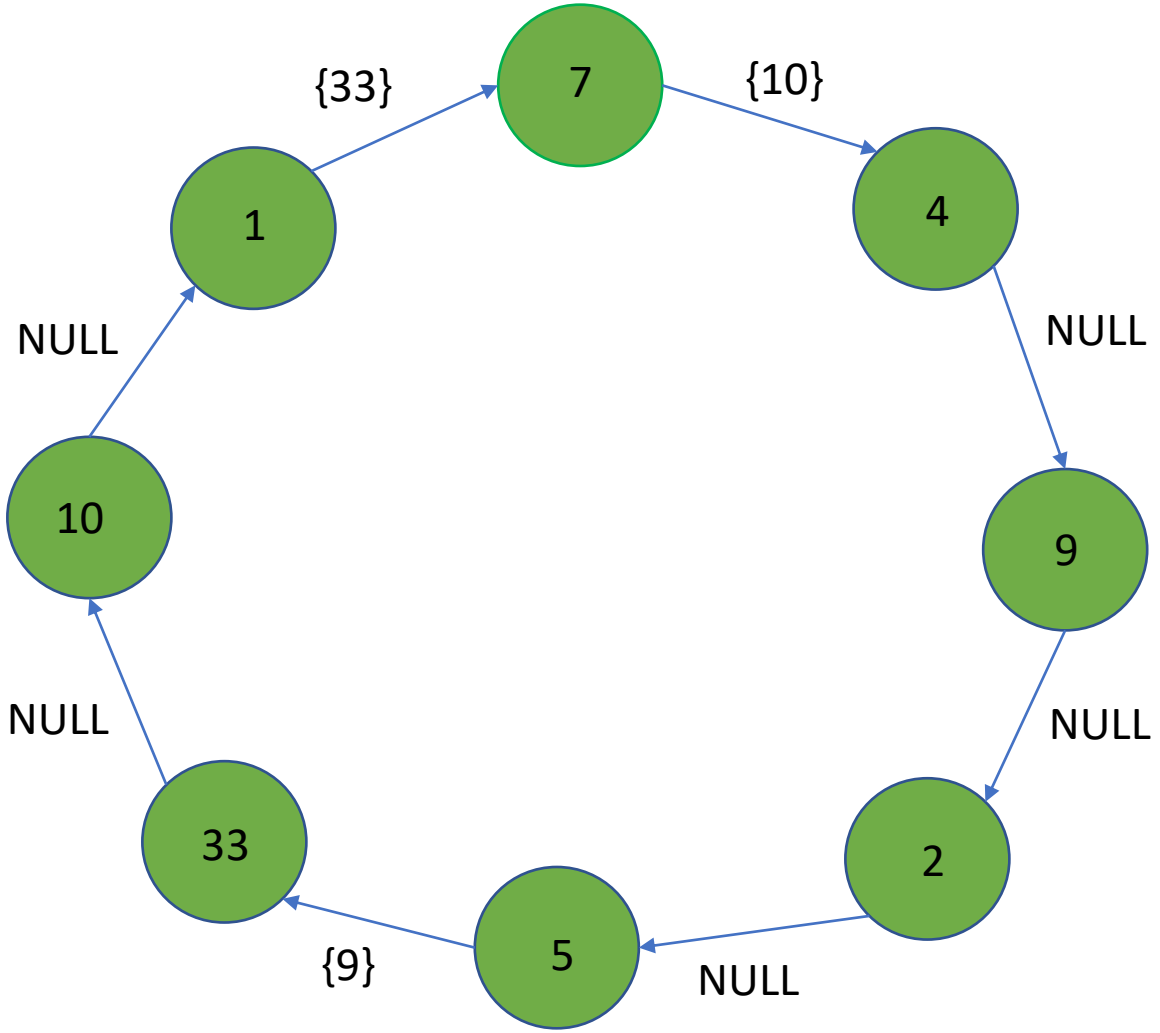
Algoritmul Flooding (LCR)



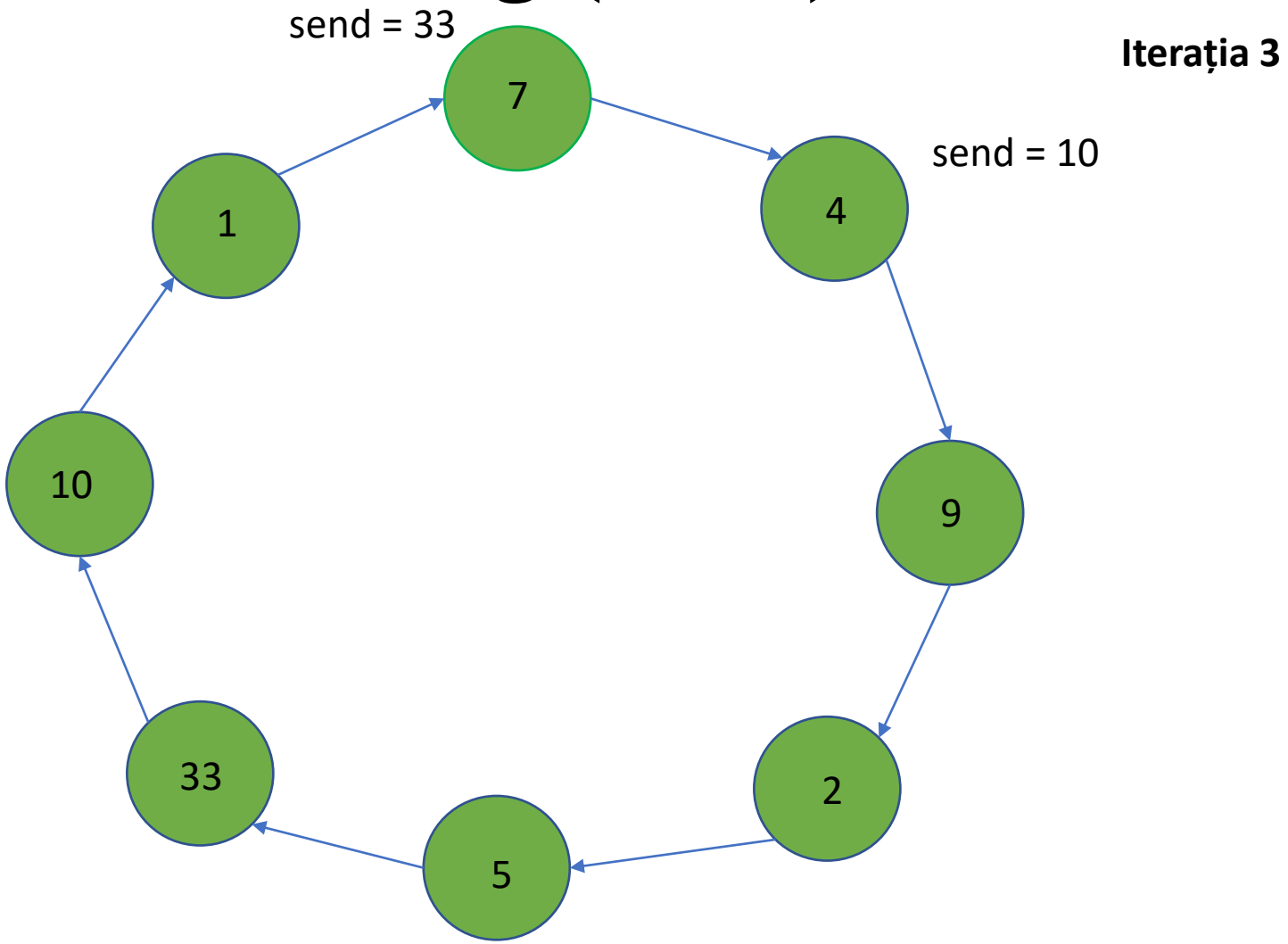
Iterația 2

Algoritmul Flooding (LCR)

Iterația 3

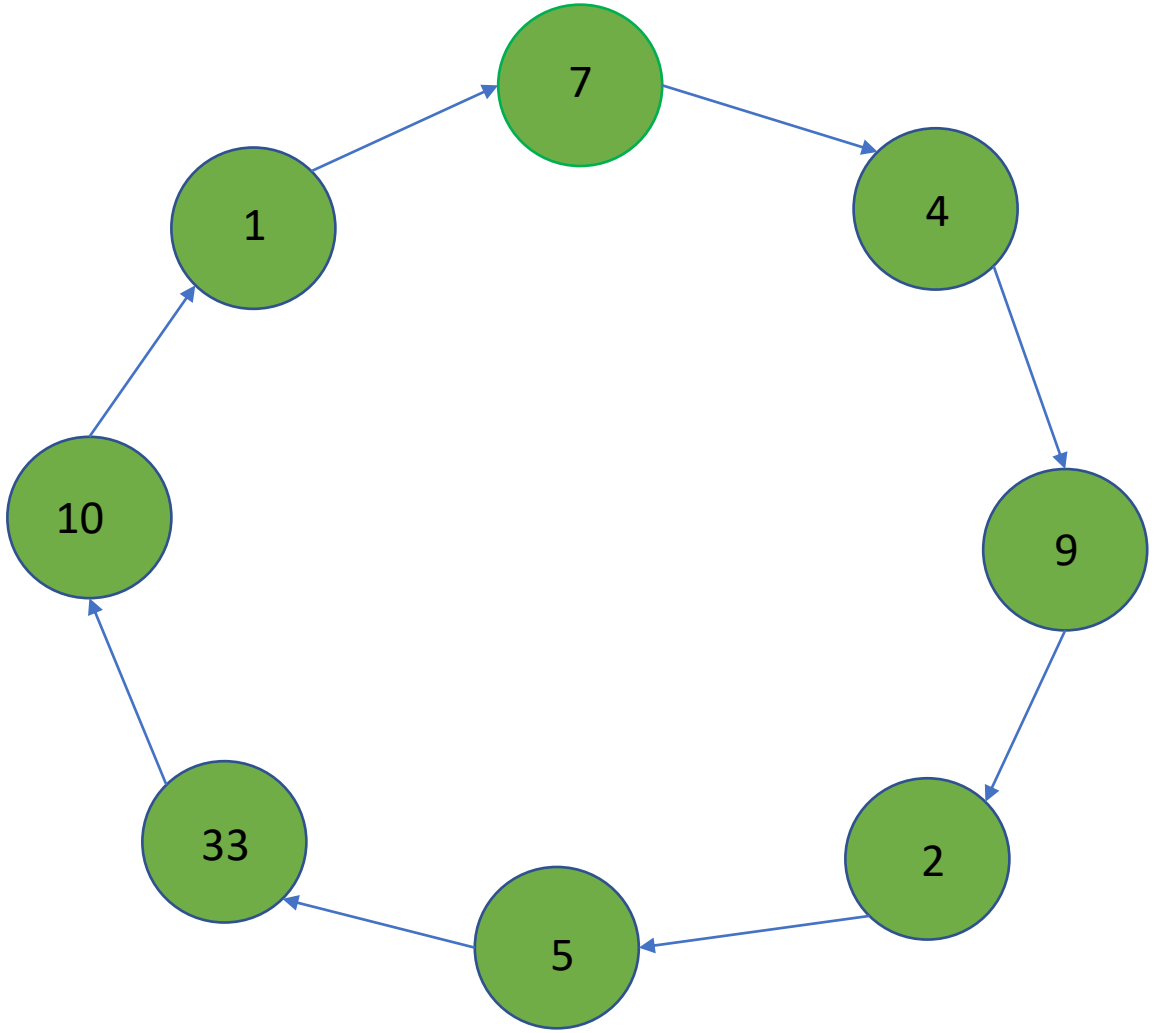


Algoritmul Flooding (LCR)



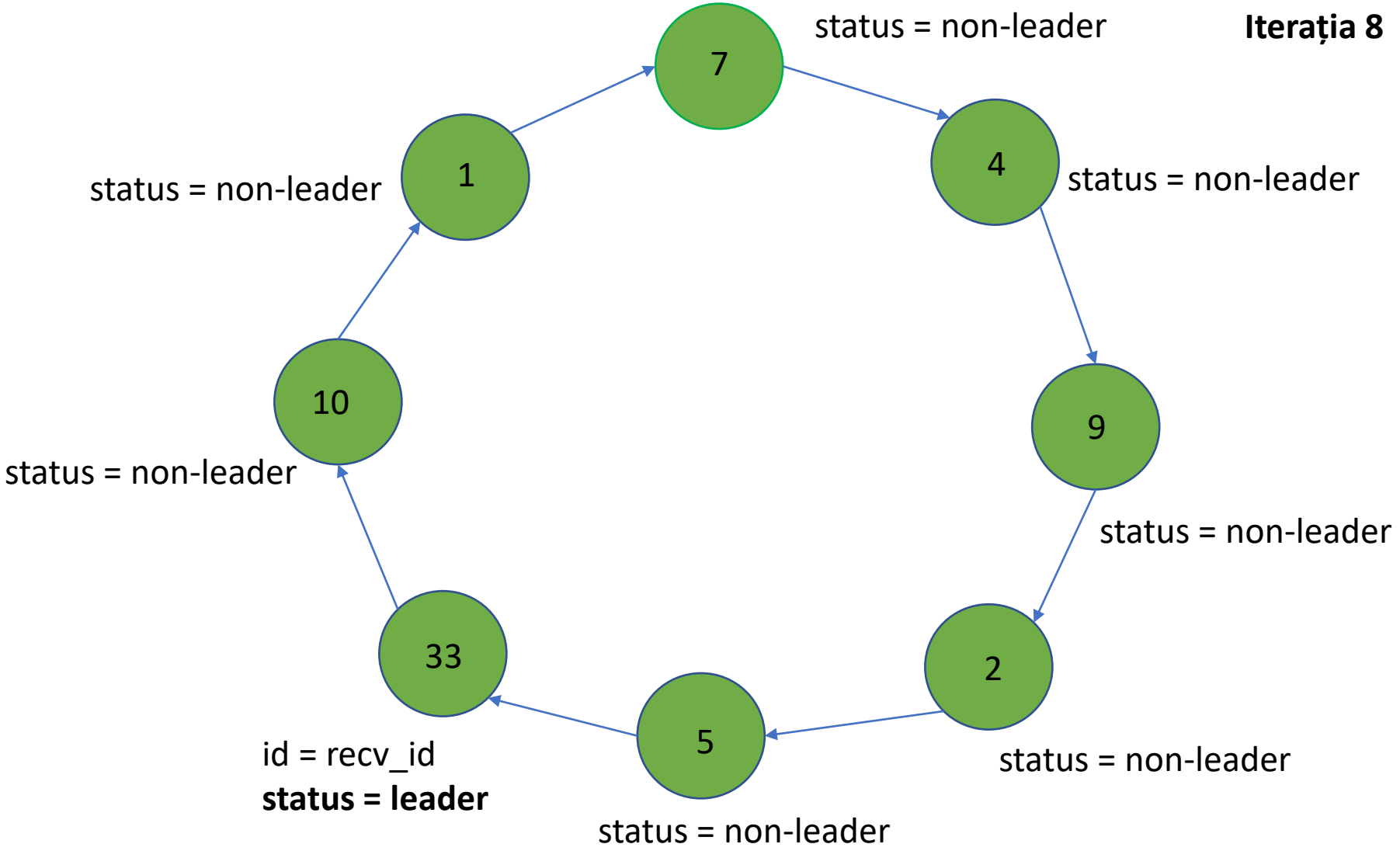
Algoritmul Flooding (LCR)

Iterația 7



send = 33

Algoritmul Flooding (LCR)



- In varianta curenta nu avem criteriu de oprire (nodurile ruleaza la infinit)
- **Solutie**: Pentru oprire liderul poate difuza un mesaj de raport, prin care semnalează incheierea competiției

Algoritmul Flooding (LCR)

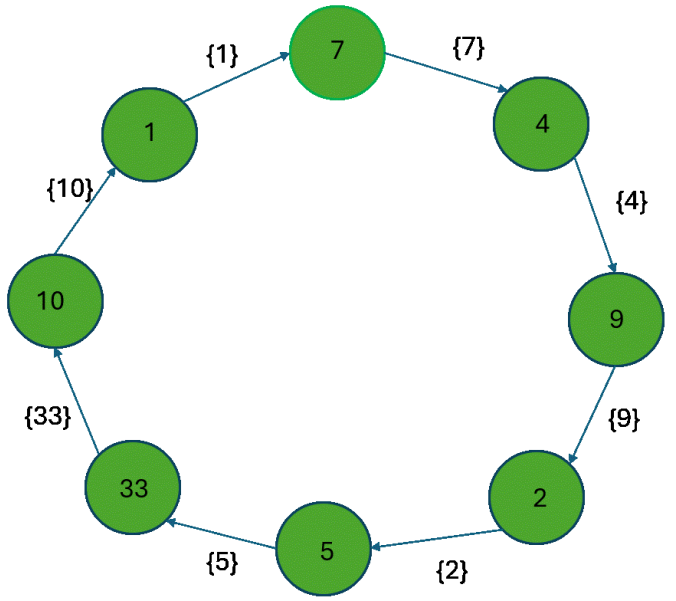
Algoritm **Flooding**():

M_i: - int id (id propriu)
- int send_id (var auxiliară), inițial id
- status ∈ {lider, non-lider}, inițial non-lider

Funcție transformare nod i ():
1. send(send_id, index + 1 mod n);
2. recv_id = recv(index - 1 mod n);
3. **If** (recv_id > id):
    1. send_id := recv_id;
4. **ElseIf** (recv_id == id): status = leader;

Teorema [Lynch]. Algoritmul LCR rezolvă problema alegerii liderului.

Complexitate. Complexitatea timp este n iterații până la anunțarea unui lider, iar complexitatea mesaj este $O(n^2)$**.**



Gerard Le Lann. *Distributed systems – toward a formal approach*. In Bruce Gilchrist, editor, Information Processing 77, Proceedings of IFIP Congress, 155-160, 1977.
Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, 1996.

Algoritmul Flooding (topologie generală)

- Reconsiderăm problema AL, de data aceasta într-o topologie generală reprezentată printr-un graf directat **tare conectat.**
- Fiecare nod are un ID unic, ales dintr-un spațiu total ordonat.
- Un singur nod poate fi ales LIDER în rețea.
- Ideea unui algoritm simplu (extensie LCR):
  - Fiecare proces înregistrează ID-ul maxim cunoscut până în prezent
  - La fiecare iterație, nodurile propagă acest maxim pe toate muchiile de ieșire
  - După *diam* iterații, dacă ID-ul maxim coincide cu ID-ul propriu, nodul se va declara LIDER, altfel NON-LIDER.

Algoritmul Flooding (graf tare conectat)

Algoritm **Flooding_Gen**(max()):

M_i: - int *id* (id propriu)
- int *max_id* (var auxiliară), inițial id
- *status* ∈ {lider, non-lider}, inițial non-lider
- int *rounds*, integer, inițial 0
- int *diam* (diametru graf)

Funcție transformare nod i ():

1. $t := t+ 1$
2. Fie U mulțimea ID-urilor primite de la vecinii de intrare
3. max_id:=max({max_id} ∪ U)
4. **If** (rounds == diam):
    1. **If** (max_id == id): status = leader;
    2. **Else**: status = non-leader;
5. **Else**: send(max_id, vecini ieșire)

Teorema [Lynch]. În algoritmul Flooding, nodul cu indicele i_{max} este lider, restul nodurilor non-lider, după *diam* iterații.

Complexitate. Complexitatea timp este *diam* iterații până la anunțarea unui lider, iar complexitatea mesaj este $diam \cdot |E|$**.** Prin $|E|$ înțelegem numărul de muchii directate din graf.

Remarci.

1. Flooding reprezintă o generalizare a LCR;
2. LCR nu necesită informație globală;
3. Dacă graful = inel unidirecțional, atunci $diam \cdot |E| = (n-1) \cdot n \approx n^2$;
4. Algoritmul funcționează cu o aproximare a constantei *diam;*

Gerard Le Lann. *Distributed systems – toward a formal approach*. In Bruce Gilchrist, editor, Information Processing 77, Proceedings of IFIP Congress, 155-160, 1977.
Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, 1996.

Algoritmul Flooding (alte aplicații)

- Difuzare în rețea (Broadcast)
- Arbori Breadth-First Search
- Consens (sincronizat)
- Estimari globale:
  - Diametru
  - Număr de noduri
  - Calcul distribuit