Distributed Algorithms

LECTURER: V.A. ZAKHAROV

Lecture 8.

Leader election.
Main definitions and assumptions.
Leader election and wave algorithms.
Election with the tree algorithm.
Leader election in rings.
Le Lann Algorithm.
Cheng–Roberts Algorithm.
Paterson/Dolev–Clave–Rode Algorithm.
Extinguish effect.

Leader election

The leader election problem is to build an algorithm which starts from a configuration where all processes are in the same state, and arrives at a configuration where exactly one process is in the state **leader** and all other processes are in the state **lost** .

An election must be held if a centralized algorithm is to be executed and there is no a priori candidate to serve as the initiator of this algorithm.

This could be also the case for a recovery procedure that must be executed after a crash of the system.

Since the set of active processes may not be known in advance it is not possible to assign one process once and for all to the role of leader.

Main definitions and assumptions

Definition

A **leader election algorithm** is an algorithm which complies with the following requirements.

1. Every process executes the same local algorithm.
2. The algorithm is decentralized, i.e., a computation can be initialized by an arbitrary non-empty subset of the processes.
3. The algorithm always terminates, and in each reachable terminal configuration there is exactly one process in the state **leader** and all other processes are in the state **lost**.

Main definitions and assumptions

The last property is sometimes weakened to require only that exactly one process is in the state **leader**. If an algorithm satisfying this weaker requirement is given, it can easily be extended by a flooding, initiated by the leader, in which all processes are informed of the result of the election.

In all algorithms a process p has a variable $state_p$ having possible values that include **leader** and **lost**. Sometimes it is assumed that the value of $state_p$ is **sleep** before p has executed any step of the algorithm, and **cand** when p has joined the computation but is not yet aware whether it has lost or won the election.

Main definitions and assumptions

We will study the Leader Election Problem within the following assumptions.

1. **The system is fully asynchronous,** i.e. the processes have no access to a common clock and the message transmission time can be arbitrary.

   The assumption of synchronous message passing does not affect much the results obtained for the election problem.

   On the contrary, the assumptions that processes can observe real time, or that message delay is bounded, does have an important impact on solutions to the election problem.

Main definitions and assumptions

2 **Each process has a unique name (ID) which is known to the process initially.**

The IDs of the processes are totally ordered. The number of bits that represent ID is w .

When designing an election algorithm one may postulate that the process with the smallest (or, the largest) ID must win the election.

In this case the election problem is referred to as the extrema-finding problem .

Main definitions and assumptions

3 **Each message may contain $O(w)$ bits.**

Each message may contain only up to a constant number of process IDs.

This assumption is made in order to allow a fair comparison between the communication complexities of different algorithms.

Elections and Waves

The process IDs are used to break symmetry between processes; one may design an election algorithm in such a way that the process with the smallest ID will be the process elected. The smallest ID can be computed by the processes in a single wave (why?).

This suggests that an election can be held by executing a wave in which the smallest ID is computed, after which the process with this ID becomes a leader.

Because an election algorithm must be decentralized this principle can only be applied to decentralized wave algorithms.

Elections and Waves

Simple Question.

Prove that a leader election algorithm based on the extrema-finding problem is a wave algorithm.

Election with the tree algorithm

If a network is a tree, or a spanning tree of a network is available, an election can be held using the tree algorithm. In the tree algorithm it is required that at least all leaves are initiators of the algorithm.

In case when only some processes are initiators a wake-up phase is added. Processes that want to start the election flood a message **wakeup** to all processes.

When a process has received a **wakeup** message through all channel, it starts Tree Algorithm, which is augmented with a distinguished variable to compute the smallest ID and to make each process decide.

When a process decides it knows the ID of the leader; if this ID equals the ID of the process, it becomes the leader, otherwise **lost**.

Election with the tree algorithm

```
var ws_p      : bool                                  init false ;
    wr_p      : integer                               init 0 ;
    rec_p[q]  : bool for every q ∈ Neigh_p            init false ;
    v_p       : P                                     init p ;
    state_p   : (sleep, leader, lost)                 init sleep ;
(* Wake-up *)
begin if p is initiator then
        begin ws_p := true ;
              forall q ∈ Neigh_p do send ⟨wakeup⟩ to q
        end;
      while wr_p < #Neigh_p do
        begin receive ⟨wakeup⟩ ; wr_p := wr_p + 1 ;
              if not ws_p then
                begin ws_p := true ;
                      forall q ∈ Neigh_p do send ⟨wakeup⟩ to q
                end
        end;
```

Election with the tree algorithm

(* Now start the tree algorithm *)
while $\#\{q : \neg rec_p[q]\} > 1$ **do**
begin
receive $\langle \mathbf{tok}, r \rangle$ from q ; $rec_p[q] := true$; $v_p := \min(v_p, r)$
end;
send $\langle \mathbf{tok}, v_p \rangle$ to q_0 with $\neg rec_p[q_0]$;
receive $\langle \mathbf{tok}, r \rangle$ from q_0 ;
$v_p := \min(v_p, r)$; (* decides with answer v_p *)
if $v_p = p$ **then** $state_p :=$ **leader else** $state_p :=$ **lost** ;
forall $q \in Neigh_p$, $q \neq q_0$ **do** send $\langle \mathbf{tok}, v_p \rangle$ to q
end

Election with the tree algorithm

Theorem 8.1.
The election problem on trees can be solved using $O(N)$ messages and $O(D)$ time units.

Election with the tree algorithm

Theorem 8.1.

The election problem on trees can be solved using $O(N)$ messages and $O(D)$ time units.

Proof.

When at least one process initiates the algorithm, all processes send $\langle$**wakeup**$\rangle$ messages to all neighbors, and each process starts the execution of the tree algorithm after receipt of the $\langle$**wakeup**$\rangle$ message from every neighbor.

All processes terminate the tree algorithm with the same value of v , namely, the smallest ID of any process.

The (unique) process with this ID will end in the state **leader** and all other processes in the state **lost**.

Election with the tree algorithm

Proof.

Two $\langle \textbf{wakeup} \rangle$ messages and two $\langle \textbf{tok}, r \rangle$ messages are sent via each channel, which brings the message complexity to $4N - 4$.

Within D time units after the first process starts the algorithm, each process has sent $\langle \textbf{wakeup} \rangle$ messages, hence within $D + 1$ time units each process has started the wave.

It is fairly easy to see that the first decision takes place within D time units after the start of the wave and the last decision takes place within D time units after the first one, which brings the total time to $3D + 1$. □

Election and wave algorithms

HOMETASK 1.

1. Prove that time complexity of the leader election algorithm on trees does not exceeds **2D** .
2. How to modify the leader election algorithm on trees to decrease the number of message exchanges?
3. How to organize leader election in arbitrary network by means of the Phase Algorithm? What is the communication complexity of such algorithm?

Leader election in rings

The election problem was first posed for ring networks in 1977 by LeLann, who also gave a solution with message complexity $O(N^2)$.

This solution was improved in 1979 by Chang and Roberts, who gave an algorithm with a worst case complexity of $O(N^2)$, but an average case complexity of only $O(N \log N)$.

The existence of an algorithm with an $O(N \log N)$ worst-case complexity remained open until 1980, when such an algorithm was given by Hirschberg and Sinclair for bidirectional rings.

It was conjectured for a while that $\Omega(N^2)$ messages was a lower bound for unidirectional rings, but Petersen and Dolev, Klawe, and Rodeh independently proposed in 1982 an $O(N \log N)$ solution for the unidirectional ring.

A worst case lower bound of $0.34N \log N$ messages for bidirectional rings was proved by Bodlaender in 1988.

Le Lann's algorithm

Each initiator computes a list of the IDs of all initiators, after which the initiator with the smallest ID is elected.

Each initiator sends a token, containing its ID, via the ring, and this token is forwarded by all processes. It is assumed that the channels are fifo.

When an initiator p receives its own token back, the tokens of all initiators have passed p , and p becomes elected if and only if p is the smallest among the initiators.

Le Lann's algorithm

var $List_p$: set of IDs $\mathcal{P}$ **init** $\{p\}$;
$state_p$;

begin if p is initiator **then**
begin $state_p := \textbf{cand}$; send $\langle \textbf{tok}, p \rangle$ to $Next_p$; receive $\langle \textbf{tok}, q \rangle$;
while $q \neq p$ **do**
begin $List_p := List_p \cup \{q\}$;
send $\langle \textbf{tok}, q \rangle$ to $Next_p$; receive $\langle \textbf{tok}, q \rangle$
end;
if $p = \min(List_p)$ **then** $state_p := \textbf{leader}$
else $state_p := \textbf{lost}$
end
else while *true* **do**
begin receive $\langle \textbf{tok}, q \rangle$; send $\langle \textbf{tok}, q \rangle$ to $Next_p$;
if $state_p = \textbf{sleep}$ **then** $state_p := \textbf{lost}$
end
end

Le Lann's algorithm

Theorem 8.2.

LeLann's algorithm solves the election problem for rings using $O(N^2)$ message exchanges and $O(N)$ time units.

Le Lann's algorithm

Theorem 8.2.

LeLann's algorithm solves the election problem for rings using $O(N^2)$ message exchanges and $O(N)$ time units.

Proof.

Because the order of the tokens on the ring is preserved (by the fifo assumption) and initiator q sends $\langle \mathbf{tok}, q \rangle$ before than q receives $\langle \mathbf{tok}, p \rangle$, the initiator p receives $\langle \mathbf{tok}, q \rangle$ before p receives $\langle \mathbf{tok}, p \rangle$ back.

This implies that every initiator p ends its computation with the $List_p$ equal to the set of all initiators, and the initiator with the smallest ID is the only process that becomes elected.

LeLann's algorithm

Proof.

There are at most N different tokens and each makes N steps, which brings the message complexity to $O(N^2)$.

LeLann's algorithm

Proof.
There are at most N different tokens and each makes N steps, which brings the message complexity to $O(N^2)$.

At $N-1$ time units at the latest after the first initiator has sent out its token, each initiator has done so.

Each initiator receives its token back within N time units after the generation of that token. This implies that the algorithm terminates within $2N-1$ time units. □

The Chang–Roberts election algorithm

The algorithm of Chang and Roberts improves LeLann's algorithm by removing from the ring all tokens of processes for which it can be seen that they will lose the election.

Namely, an initiator p removes a token $\langle \mathbf{tok}, q \rangle$ if $q > p$.

Initiator p becomes **lost** when a token with ID $q < p$ is received, and **leader** when the token with ID p is received.

The Chang–Roberts election algorithm

```
var state_p ;

begin if p is initiator then
      begin state_p := cand ; send ⟨tok, p⟩ to Next_p ;
            while state_p ≠ leader do
                  begin receive ⟨tok, q⟩ ;
                        if q = p then state_p := leader
                        else if q < p then
                              begin if state_p = cand then state_p := lost;
                                    send ⟨tok, q⟩ to Next_p end
                  end
      end
      else while true do
          begin receive ⟨tok, q⟩ ; send ⟨tok, q⟩ to Next_p ;
                if state_p = sleep then state_p := lost
          end
end
```

The Chang–Roberts election algorithm

Theorem 8.3.
The Chang–Roberts election algorithm solves the election problem for rings using $\Theta(N^2)$ message exchanges in the worst case and $O(N)$ time units.

The Chang–Roberts election algorithm

Theorem 8.3.
The Chang–Roberts election algorithm solves the election problem for rings using $\Theta(N^2)$ message exchanges in the worst case and $O(N)$ time units.

Proof.
Denote by p_0 an initiator with the smallest ID.

Each process is either a non-initiator or an initiator with an ID larger than p_0 , and, therefore, all processes forward the token $\langle \mathbf{tok}, p_0 \rangle$ issued by p_0 .

Hence, p_0 receives its token back and becomes elected.

The Chang–Roberts election algorithm

Proof.

Non-initiators do not become elected, but they all enter the state **lost** at the latest when p_0 's token is forwarded.

An initiator p , for which $p > p_0$ holds, will not be elected, i.e. p_0 does not forward the token $\langle \mathbf{tok}, p \rangle$, and, therefore, p never receives its own token.

Such an initiator p enters the state **lost** at the latest when p_0 's token is forwarded.

This proves that the algorithm solves the election problem.

The Chang–Roberts election algorithm

Proof.

The algorithm uses no more than N tokens, and every token is forwarded N times at most; these considerations brings us $O(N^2)$ message complexity.

The Chang–Roberts election algorithm

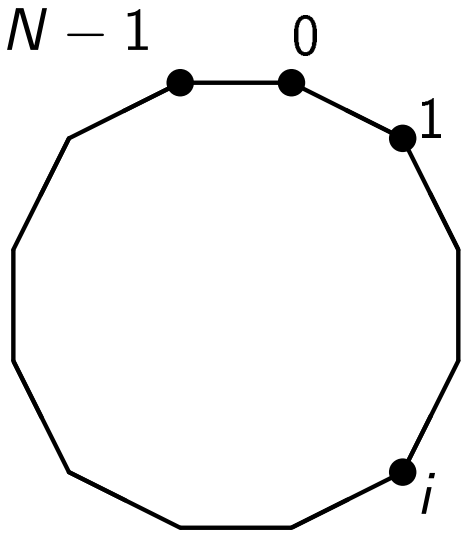
Proof.

The algorithm uses no more than N tokens, and every token is forwarded N times at most; these considerations brings us $O(N^2)$ message complexity.

To show that $\Omega(N^2)$ messages may indeed be used, consider an initial configuration where the IDs are arranged in increasing order around the ring and each process is an initiator.

The Chang–Roberts election algorithm

Proof.



Messages are passed
in a clockwise direction

The Chang–Roberts election algorithm

Proof.

The algorithm uses no more than N tokens, and every token is forwarded N times at most; these considerations brings us $O(N^2)$ message complexity.

To show that $\Omega(N^2)$ messages may indeed be used, consider an initial configuration where the IDs are arranged in increasing order around the ring and each process is an initiator.

The token of each process is removed from the ring by process 0 , so the token of process i is forwarded by $N - i$ hops, which brings the number of message passes to

$$\sum_{i=0}^{N-1}(N-i) = \frac{1}{2}N(N+1).$$

The Chang–Roberts election algorithm

Theorem 8.4.

The Chang–Roberts algorithm requires only $\approx 0.69N \log N$ message passes in the average case when all processes are initiators..

The Chang–Roberts election algorithm

Theorem 8.4.

The Chang–Roberts algorithm requires only $\approx 0.69N \log N$ message passes in the average case when all processes are initiators..

Simple Questions.

1. Does the correctness of Chang–Roberts election algorithm depend on the order in which messages follow in the channels?
2. Consider Chang–Roberts election algorithm and assume that every process is an initiator. When the message complexity of the algorithm will be minimal and how many message exchanges does it take?

The Peterson/Dolev–Klawe–Rodeh Algorithm

The algorithm requires that channels are fifo.

The algorithm first computes the smallest ID and makes it known to each process, then the process with that ID becomes leader and all others are defeated.

The algorithm is more easily understood if one considers it as if it is executed by the IDs of processes which operate like bots (proxies) authorized by the corresponding processes to participate in the elections.

The Peterson/Dolev–Klawe–Rodeh Algorithm

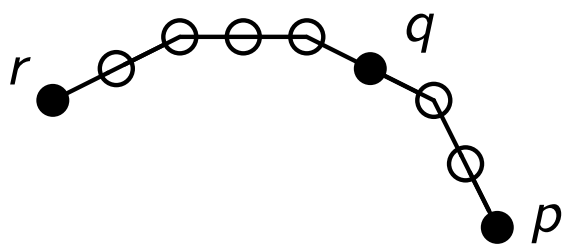
Initially each ID is active , but in each round some IDs become passive .

In every round an active ID compares itself with the two neighboring active IDs in clockwise and anticlockwise directions; if it is a local minimum, it survives the round, otherwise it becomes passive .

Because all IDs are different, an ID next to a local minimum is not a local minimum, which implies that at least half of the IDs do not survive the round.

Consequently, after at most $\log N$ rounds only one active ID remains, which is the winner.

The Peterson/Dolev–Klawe–Rodeh Algorithm

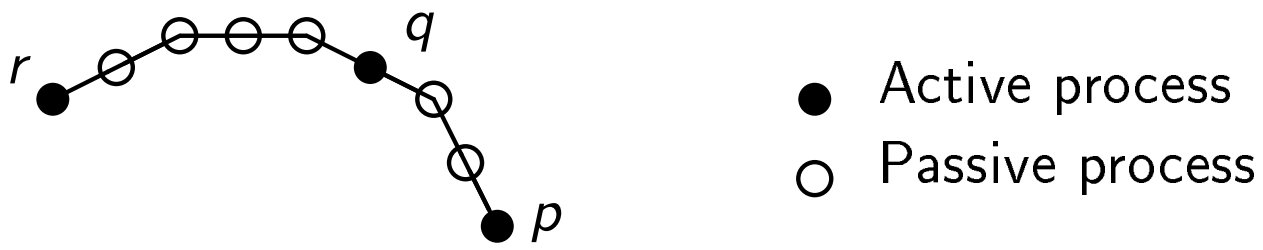


● Active process

○ Passive process

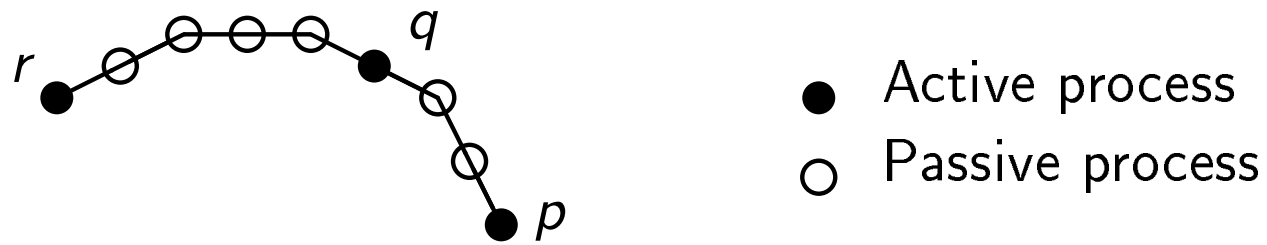
In directed rings messages can be sent only clockwise, which makes it difficult to obtain the next active ID in the clockwise direction. ID q must be compared with r and p ; ID r can be sent to q , but ID p would have to travel against the direction of the channels in order to arrive at q .

The Peterson/Dolev–Klawe–Rodeh Algorithm



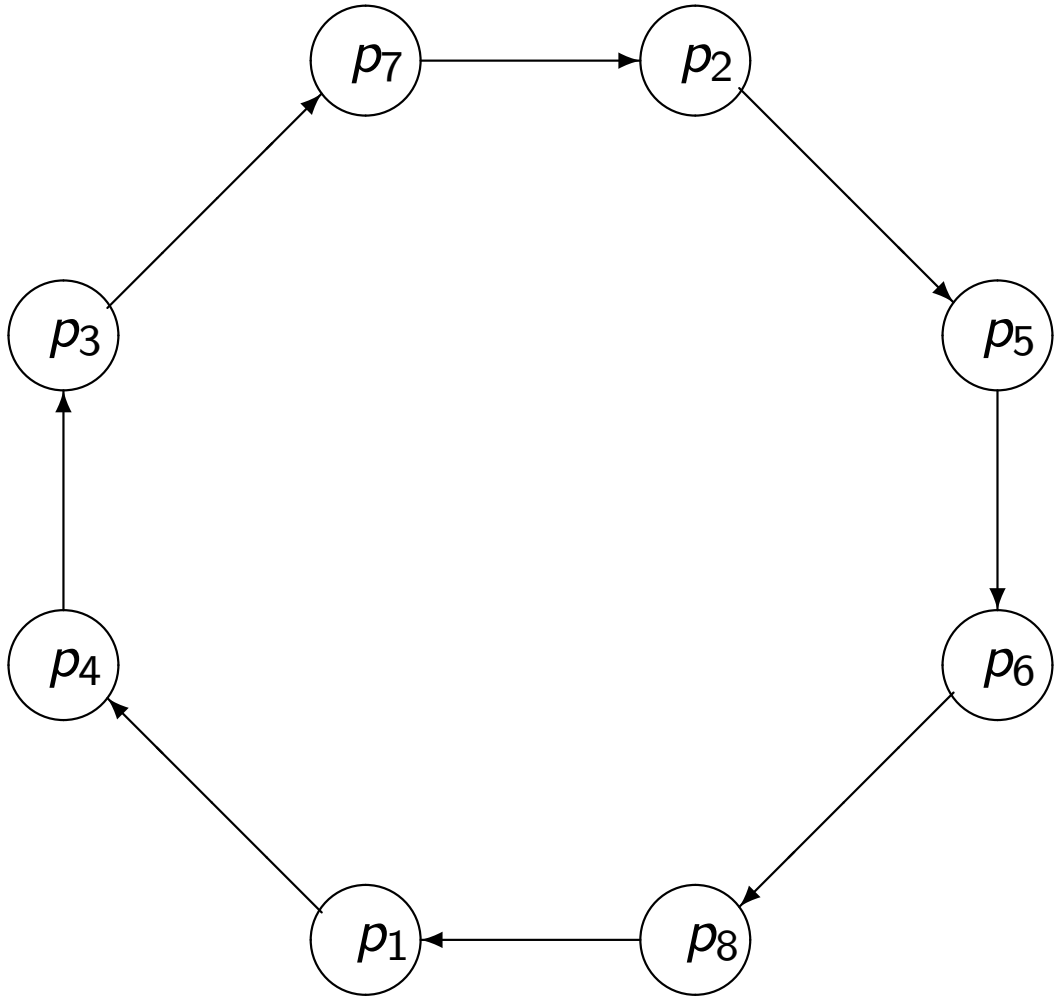
To make a comparison with both r and p possible, ID q is sent (in the direction of the ring) to the process holding ID p and r is forwarded not only to the process holding q but also, further, to the process holding p .

The Peterson/Dolev–Klawe–Rodeh Algorithm

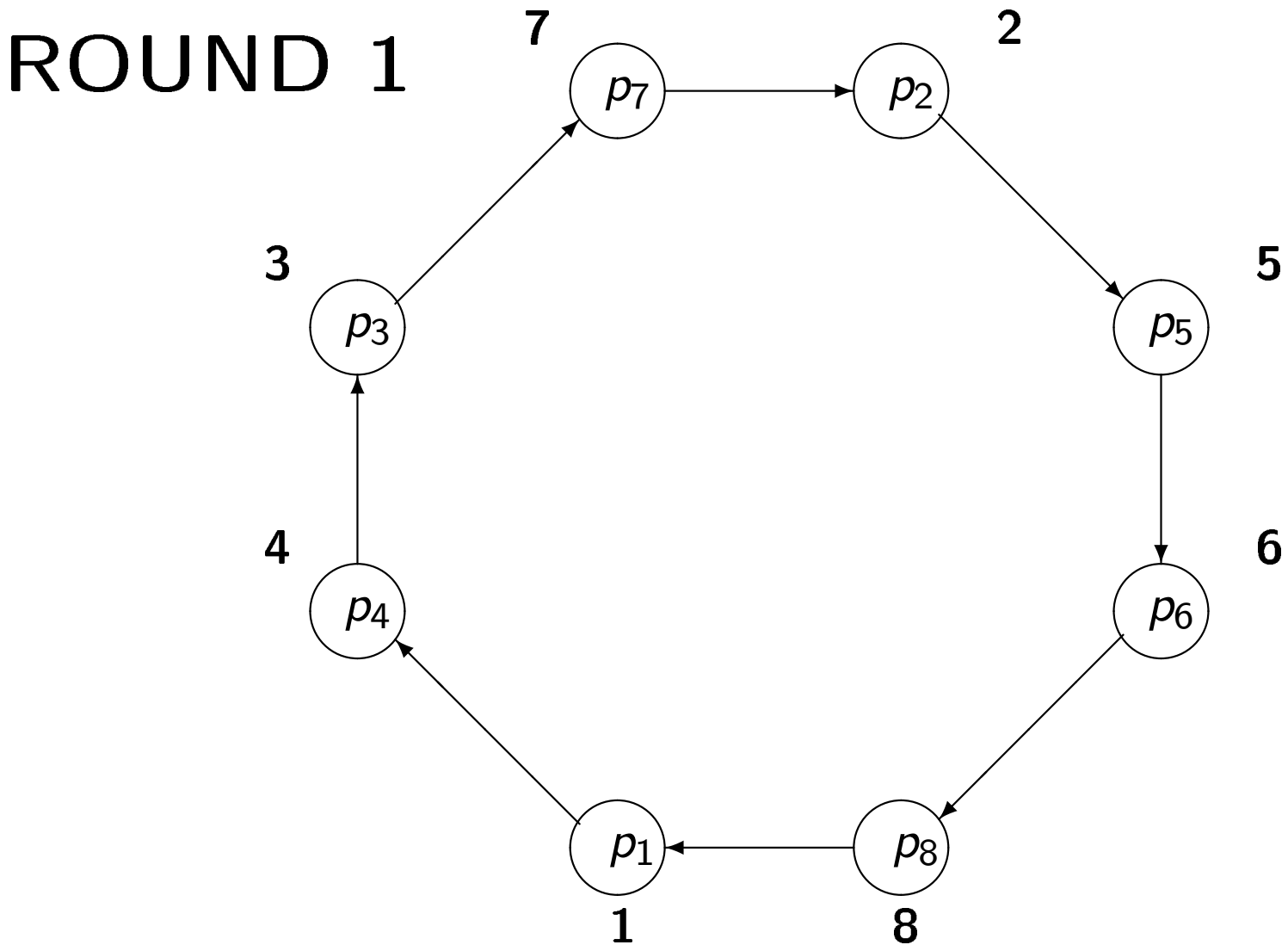


If q is the only active ID at the beginning of a round, the first ID that q encounters on its tour equals q (i.e., $p = q$ in this case). When this situation occurs this ID is the winner of the election.

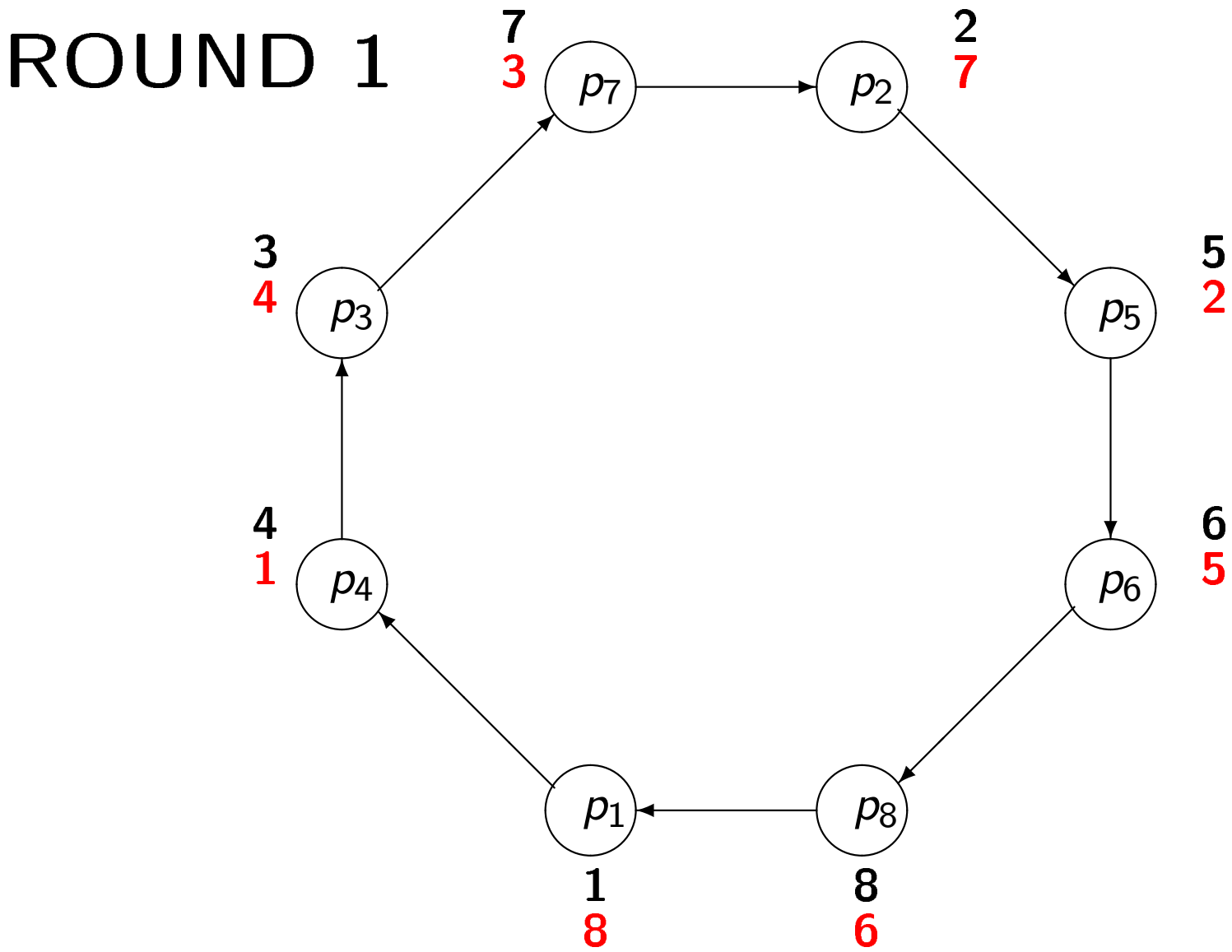
The Peterson/Dolev–Klawe–Rodeh Algorithm



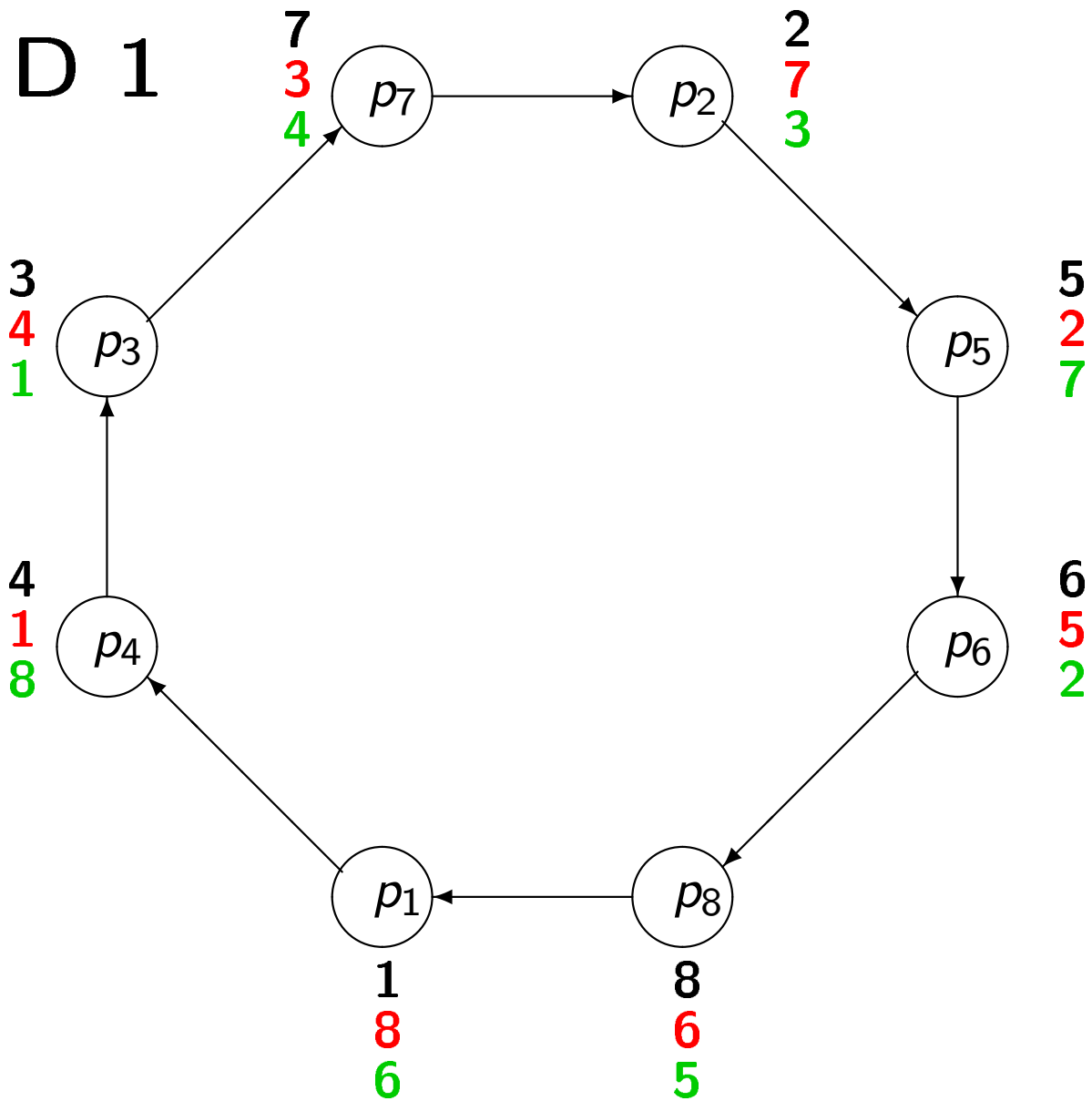
The Peterson/Dolev–Klawe–Rodeh Algorithm



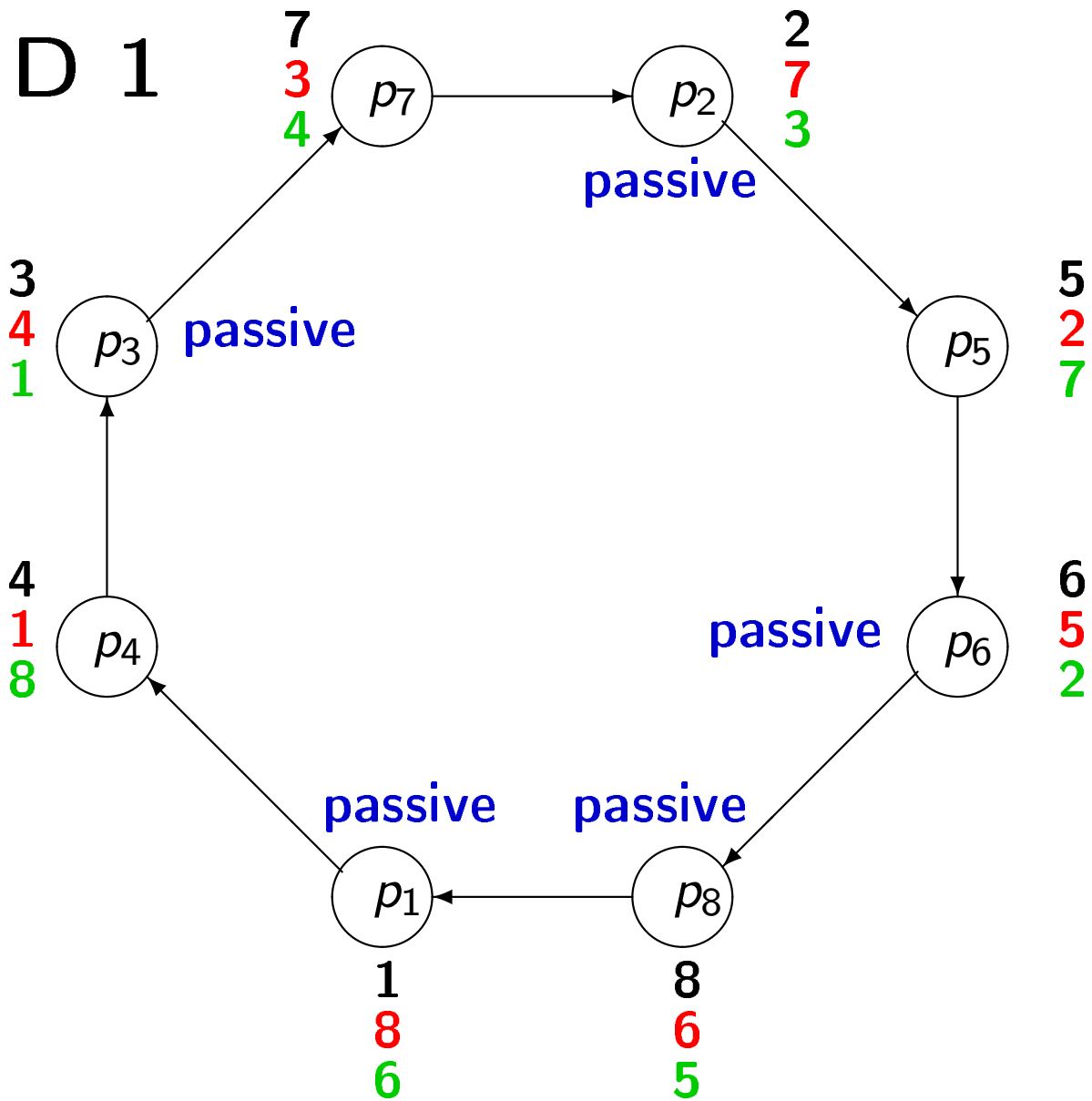
The Peterson/Dolev–Klawe–Rodeh Algorithm



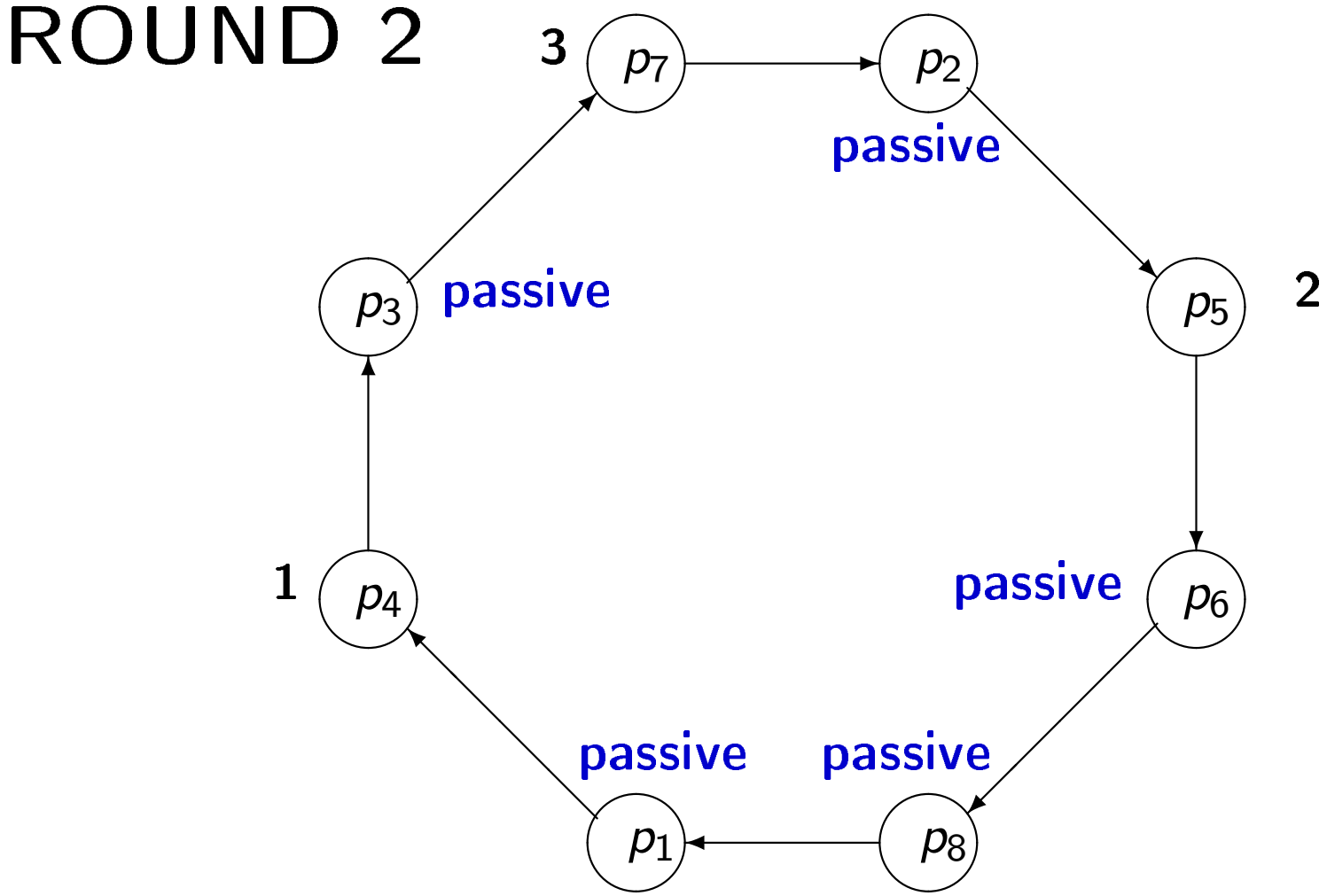
The Peterson/Dolev–Klawe–Rodeh Algorithm



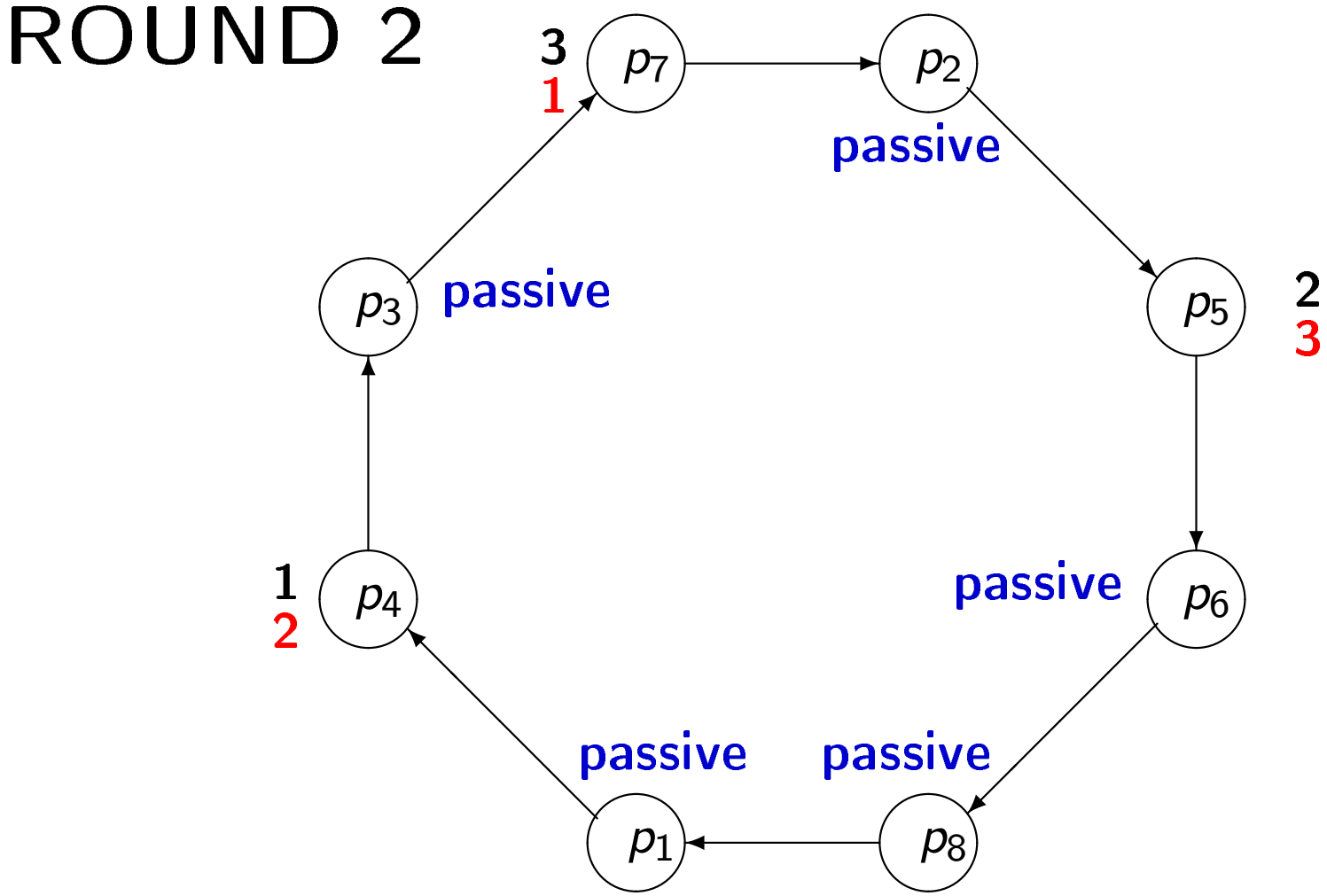
The Peterson/Dolev–Klawe–Rodeh Algorithm



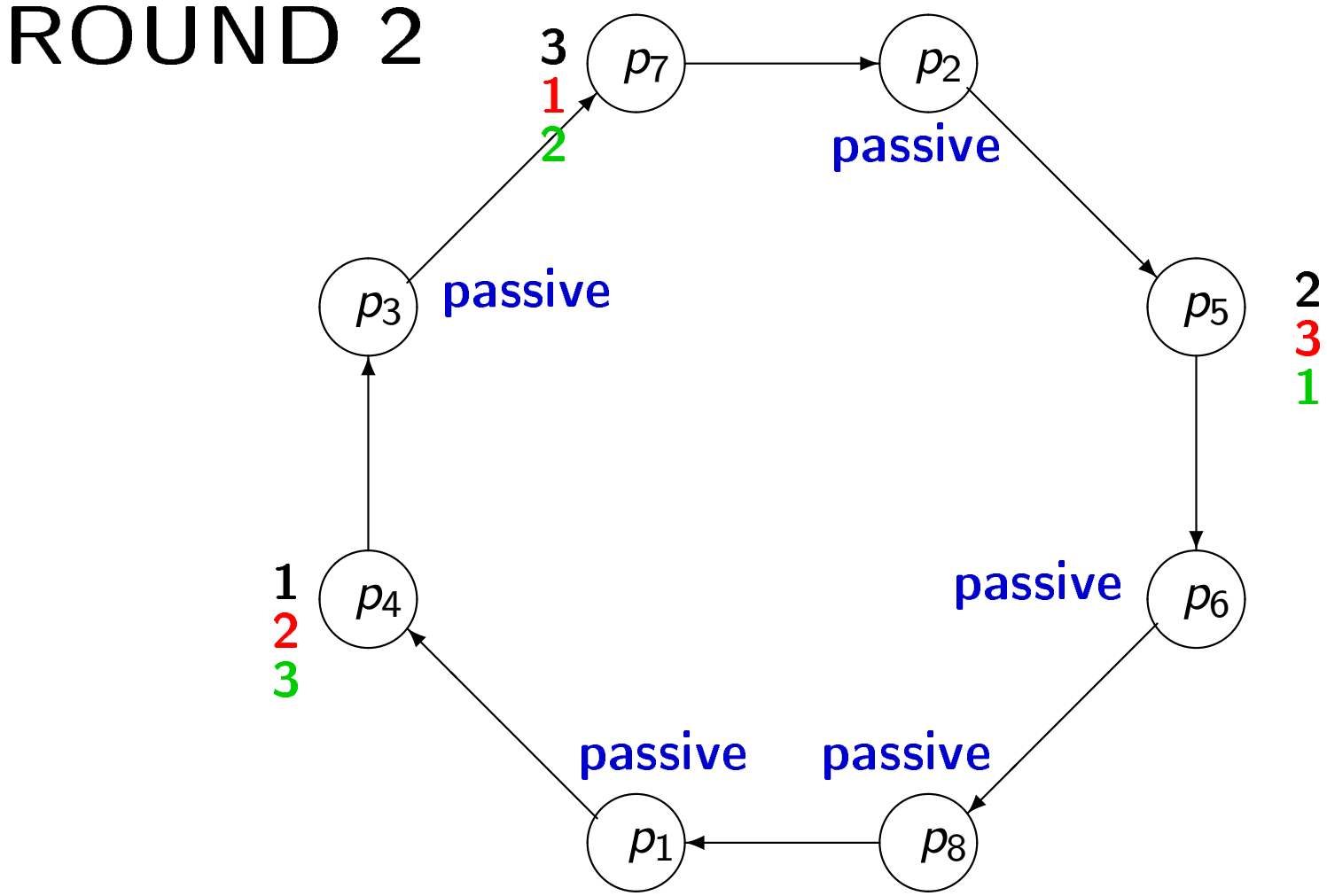
The Peterson/Dolev–Klawe–Rodeh Algorithm



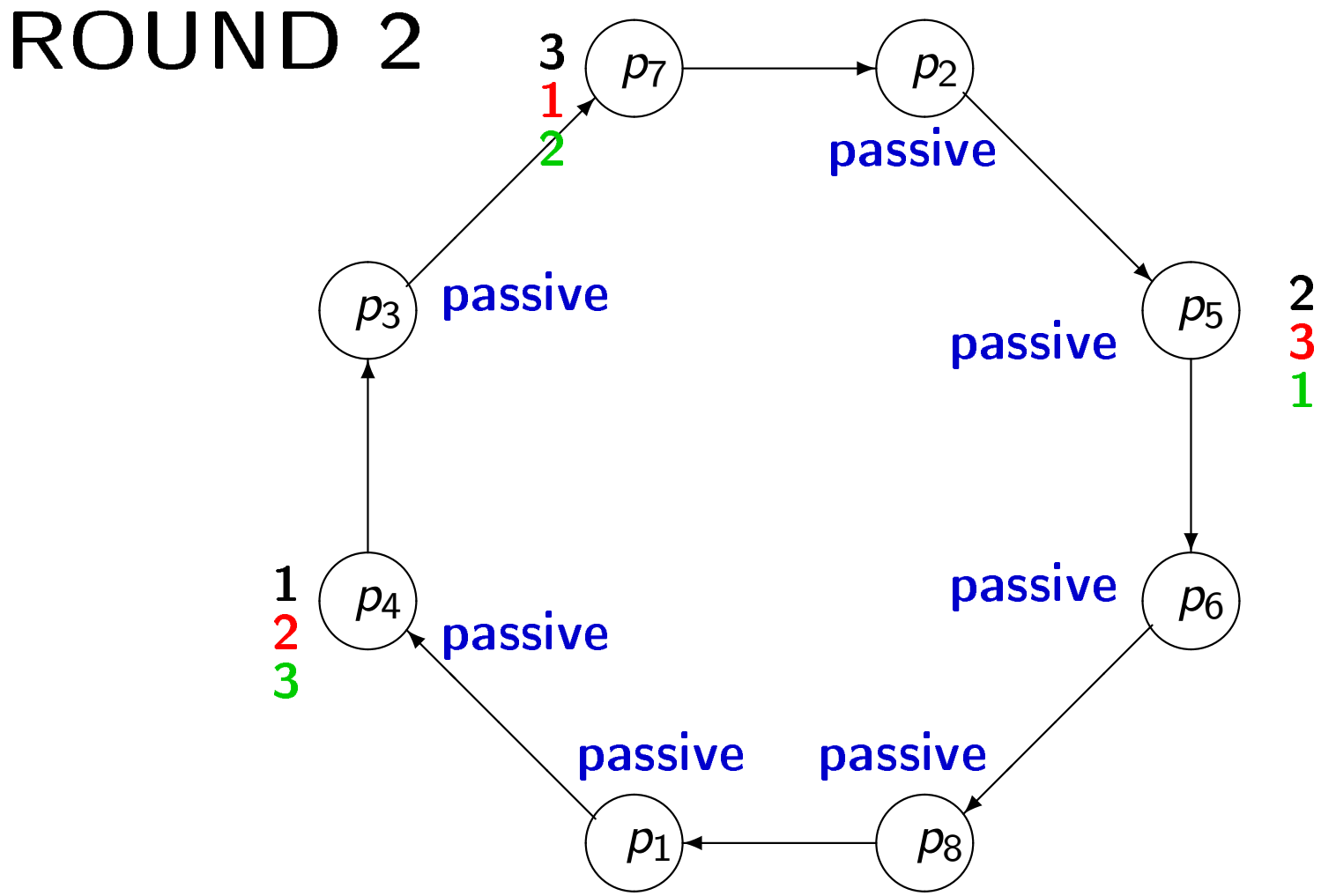
The Peterson/Dolev–Klawe–Rodeh Algorithm



The Peterson/Dolev–Klawe–Rodeh Algorithm

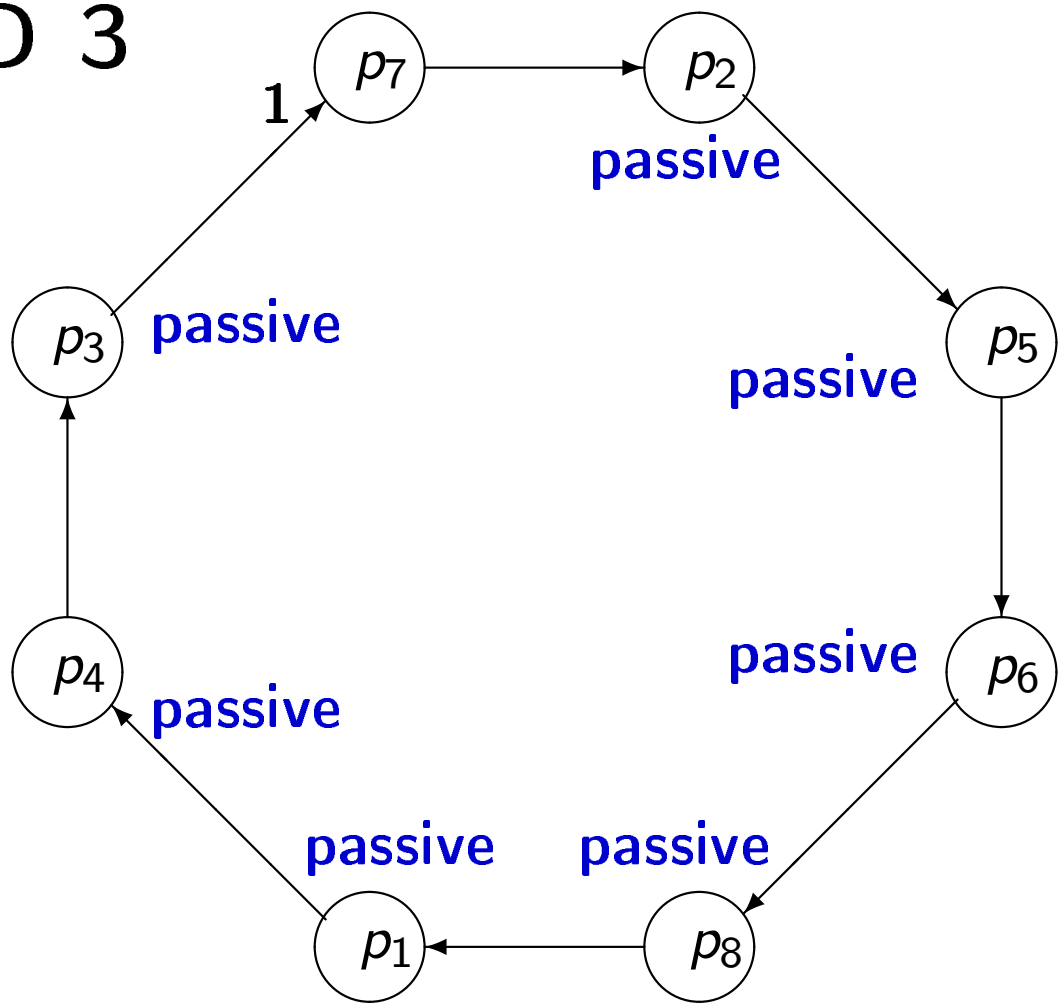


The Peterson/Dolev–Klawe–Rodeh Algorithm



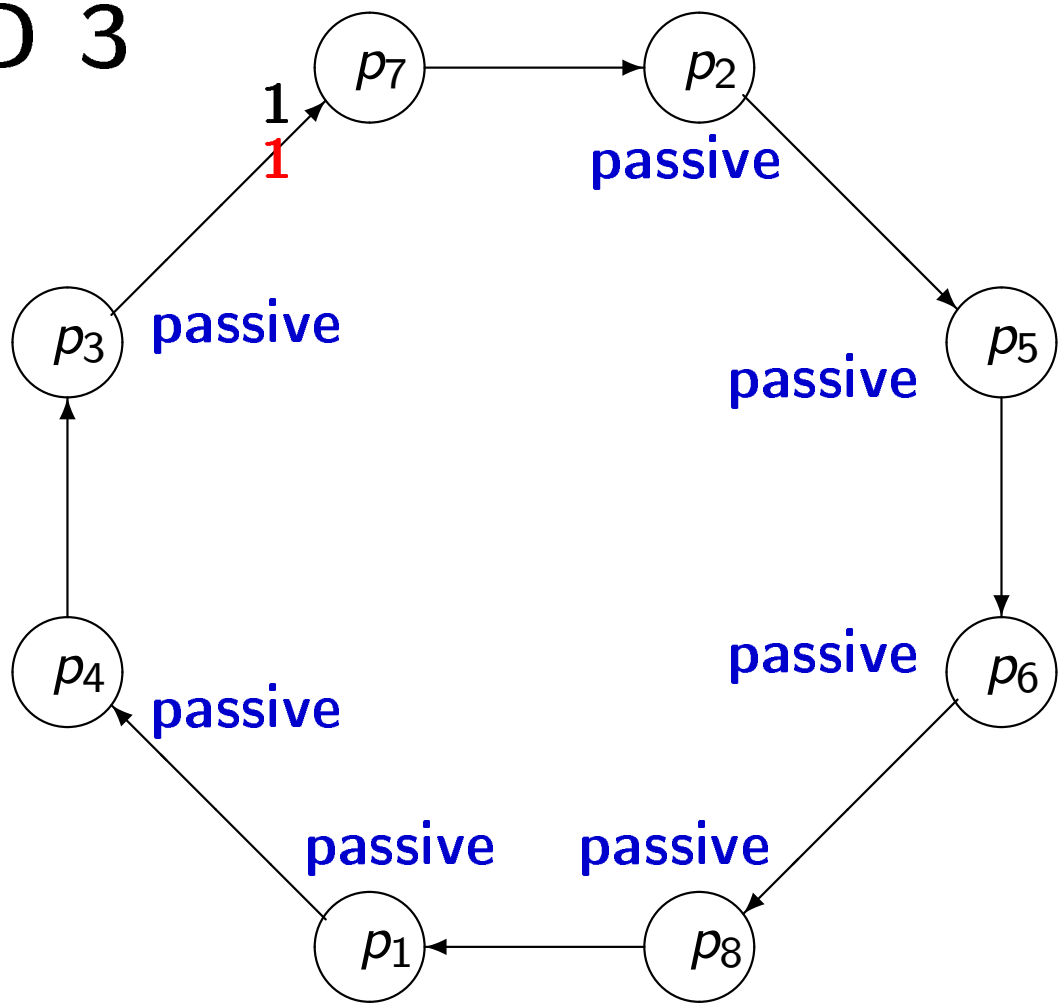
The Peterson/Dolev–Klawe–Rodeh Algorithm

ROUND 3



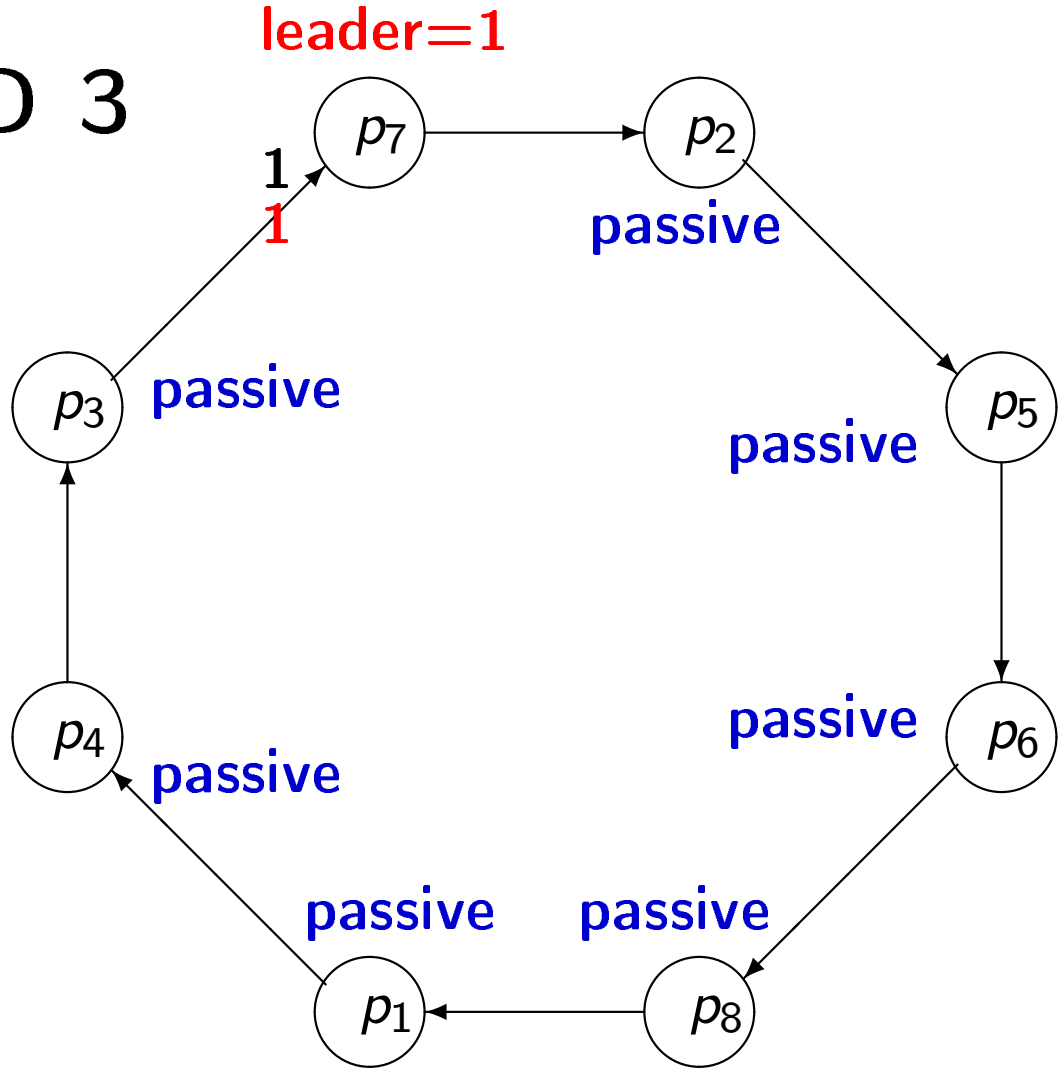
The Peterson/Dolev–Klawe–Rodeh Algorithm

ROUND 3



The Peterson/Dolev–Klawe–Rodeh Algorithm

ROUND 3



The Peterson/Dolev–Klawe–Rodeh Algorithm

```
var ci_p    : P   init p ;          (* Current ID of p *)
    acn_p   : P   init udef ;       (* ID of active anticlockwise neighbor *)
    win_p   : P   init udef ;       (* ID of winner *)
    state_p : (active, passive, leader, lost) init active ;
begin if p is initiator then state_p := active else state_p := passive ;
      while win_p = udef do
        begin if state_p = active then
              begin send ⟨one, ci_p⟩; receive ⟨one, q⟩ ; acn_p := q;
                    if acn_p = ci_p then (* acn_p is the minimum *)
                      begin send ⟨small, acn_p⟩ ; win_p := acn_p ;
                            receive ⟨small, q⟩
                      end
                    else (* acn_p is current ID of neighbor *)
                      begin send ⟨two, acn_p⟩; receive ⟨two, q⟩ ;
                            if acn_p < ci_p and acn_p < q
                              then ci_p := acn_p else state_p := passive
                            end
                    end
```

The Peterson/Dolev–Klawe–Rodeh Algorithm

else (* $state_p = passive$ *)
    begin receive $\langle \textbf{one}, q \rangle$; send $\langle \textbf{one}, q \rangle$;
        receive m ; send m ;
        (* m is either $\langle \textbf{two}, q \rangle$, or $\langle \textbf{small}, q \rangle$*)
        if m is amessage $\langle \textbf{small}, q \rangle$ **then** $win_p := q$
    end
end;
if $p = win_p$ **then** $state_p := leader$ **else** $state_p := lost$
end

The Peterson/Dolev–Klawe–Rodeh Algorithm

Process p is active in some round if at the beginning of this tour it holds active ID ci_p . Otherwise p is passive and just retransmitted all messages it receives.

An active process sends its current ID to the next active process and obtains the current ID of the previous active process using messages of the type $\langle \mathbf{one} \rangle$.

The received ID is stored (in the variable acn_p).

The Peterson/Dolev–Klawe–Rodeh Algorithm

```
var ci_p    : P   init p ;          (* Current ID of p *)
    acn_p   : P   init udef ;       (* ID of active anticlockwise neighbor *)
    win_p   : P   init udef ;       (* ID of the winner *)
    state_p : (active, passive, leader, lost) init active ;
begin if p is initiator then state_p := active else state_p := passive ;
      while win_p = udef do
        begin if state_p = active then
                begin send ⟨one, ci_p⟩; receive ⟨one, q⟩ ; acn_p := q ;
                      if acn_p = ci_p then (* acn_p is the minimum *)
                        begin send ⟨small, acn_p⟩ ; win_p := acn_p ;
                              receive ⟨small, q⟩
                        end
                      else (* acn_p is current ID of neighbor *)
                        begin send ⟨two, acn_p⟩; receive ⟨two, q⟩ ;
                              if acn_p < ci_p and acn_p < q
                                then ci_p := acn_p else state_p := passive
                              end
                end
```

The Peterson/Dolev–Klawe–Rodeh Algorithm

The received ID is stored (in the variable acn_p) and if the ID survives the round it will be the current ID of p in the next round.

To determine whether the ID acn_p survive this round it is compared with both ci_p and the active ID, receved in the message of the type $\langle \mathbf{two} \rangle$.

Process p sends a message $\langle \mathbf{two}, acp_p \rangle$ message to make this decision possible in the next active process.

The Peterson/Dolev–Klawe–Rodeh Algorithm

```
var ci_p    : P   init p ;          (* Current ID of p *)
    acn_p   : P   init udef ;       (* ID of active anticlockwise neighbor *)
    win_p   : P   init udef ;       (* ID of the winner *)
    state_p : (active, passive, leader, lost) init active ;
begin if p is initiator then state_p := active else state_p := passive ;
      while win_p = udef do
        begin if state_p = active then
                begin send ⟨one, ci_p⟩; receive ⟨one, q⟩ ; acn_p := q;
                      if acn_p = ci_p then (* acn_p is the minimum *)
                        begin send ⟨small, acn_p⟩ ; win_p := acn_p ;
                              receive ⟨small, q⟩
                        end
                      else  (* acn_p is current ID of neighbor *)
                        begin  send ⟨two, acn_p⟩; receive ⟨two, q⟩ ;
                               if  acn_p < ci_p and acn_p < q
                                 then  ci_p := acn_p else state_p := passive
                               end
                end
```

The Peterson/Dolev–Klawe–Rodeh Algorithm

An exception occurs when $acn_p = ci_p$; in this case this ID is the only remaining active one and it is announced to all processes in $\langle \textbf{small}, acn_p \rangle$ message.

The Peterson/Dolev–Klawe–Rodeh Algorithm

```
var ci_p    : P    init p ;          (* Current ID of p *)
    acn_p   : P    init udef ;       (* ID of active anticlockwise neighbor *)
    win_p   : P    init udef ;       (* ID of the winner *)
    state_p : (active, passive, leader, lost) init active ;
begin if p is initiator then state_p := active else state_p := passive ;
      while win_p = udef do
        begin if state_p = active then
                begin send ⟨one, ci_p⟩; receive ⟨one, q⟩ ; acn_p := q;
                      if  acn_p = ci_p then  (* acn_p is the minimum *)
                        begin  send ⟨small, acn_p⟩ ; win_p := acn_p ;
                               receive ⟨small, q⟩
                        end
                      else (* acn_p is current ID of neighbor *)
                        begin send ⟨two, acn_p⟩; receive ⟨two, q⟩ ;
                              if acn_p < ci_p and acn_p < q
                                then ci_p := acn_p else state_p := passive
                              end
                end
```

The Peterson/Dolev–Klawe–Rodeh Algorithm

Theorem 8.5.
The Peterson/Dolev–Klawe–Rodeh Algorithm solves election problem using $O(N \log N)$ message exchanges.

The Peterson/Dolev–Klawe–Rodeh Algorithm

Theorem 8.5.

The Peterson/Dolev–Klawe–Rodeh Algorithm solves election problem using $O(N \log N)$ message exchanges.

Proof.

We say that a process is in the i -th round when it executes the main loop for the i -th time.

The rounds are not globally synchronized; it is possible that one process is several rounds ahead of another process in a different part of the ring.

But, as each process sends and receives exactly two messages in each round and channels are fifo, a message is always received in the same round as it is sent.

In the first round all initiators are active , and every active process holds its own ID.

The Peterson/Dolev–Klawe–Rodeh Algorithm

Claim 1.
If a round i starts with $k > 1$ active processes, and all IDs ci_p , stored in these processes are pairwise different then at least one and at most $k/2$ processes survive the round.

At the end of i -th round all IDs of active processes are different and include the smallest ID.

The Peterson/Dolev–Klawe–Rodeh Algorithm

Proof of Claim 1.

After exchenging messages of the type $\langle \textbf{one}, q \rangle$ every active process receives the ID of its first active anticlockwise neighbor. And this ID differs from its own ID.

Hence, every active process continues the round with the exchange of messages of the type $\langle \textbf{two}, q \rangle$. Therefore, each active process also obtains the current ID of the second anticlockwise active neighbor.

All active processes now hold different values of *acn* , which implies that the survivors of the round all have different IDs at the end of the round.

The Peterson/Dolev–Klawe–Rodeh Algorithm

Proof of Claim 1.

At least the identity that was the smallest at the beginning of the round survives, so there is at least one survivor.

An ID next to a local minimum is not a local minimum, which implies that the number of survivors is at most $k/2$. □

From Claim 1 it follows that there will be such a round, with number at most $\leq \lfloor \log N \rfloor + 1)$, that begins with exactly one active identity, namely, the smallest ID of any initiator.

The Peterson/Dolev–Klawe–Rodeh Algorithm

Claim 2.

If a round starts with exactly one active process p , with current identity ci_p , the algorithm terminates after that round with $win_q = ci_p$ for each q .

Proof of Claim 2.

A message $\langle \textbf{one}, ci_p \rangle$ sent by p will be relayed by all processes and finally will be received by p .

The process p checks that $acn_p = ci_p$ holds and sends the message $\langle \textbf{small}, acn_p \rangle$ in the ring, thus causing every process q exit the main loop of its program and assign acn_p to the variable win_q . □

The Peterson/Dolev–Klawe–Rodeh Algorithm

Proof of Theorem.

The algorithm terminates in each process and all processes agree on the ID of the leader (in the variable win_q).

This process is in the state **leader** and all other processes are in the state **lost**.

It will take $\lfloor \log N \rfloor + 1$ rounds at most, in each of which exactly $2N$ messages are exchanged, which proves that the message complexity is bounded by $2N \log N + O(N)$. □

Peterson/Dolev–Klawe–Rodeh Algorithm

Simple Questions.

1. Find such an initial configuration of Peterson/Dolev–Klawe–Rodeh Algorithm, which requires exactly $\lfloor \log N \rfloor + 1$ rounds to complete the computation.
2. Find such an intial configuration which requires exactly 2 rounds to complete the computation regardless of the number of processes in the ring.
3. Can the algorithm complete the computation in one round?

The Peterson/Dolev–Klawe–Rodeh Algorithm

Theorem 8.6.

Suppose that

- the ring is unidirectional,

The Peterson/Dolev–Klawe–Rodeh Algorithm

Theorem 8.6.

Suppose that

- the ring is unidirectional,
- all processes are not aware of the size of the ring,

The Peterson/Dolev–Klawe–Rodeh Algorithm

Theorem 8.6.

Suppose that

- the ring is unidirectional,
- all processes are not aware of the size of the ring,
- all channels are FIFO

The Peterson/Dolev–Klawe–Rodeh Algorithm

Theorem 8.6.

Suppose that

- the ring is unidirectional,
- all processes are not aware of the size of the ring,
- all channels are FIFO
- all processes are initiators.

The Peterson/Dolev–Klawe–Rodeh Algorithm

Theorem 8.6.

Suppose that

- the ring is unidirectional,
- all processes are not aware of the size of the ring,
- all channels are FIFO
- all processes are initiators.

Then the average message exchange complexity of any leader election algorithm is no less than $N \cdot \mathbf{H}_N$, where $\mathbf{H}_N = \sum_{k=1}^{N} 1/k$.

The Peterson/Dolev–Klawe–Rodeh Algorithm

Theorem 8.7.
Any decentralized wave algorithm for ring networks exchanges at least $\Omega(N \log N)$ messages, in the average as well as in the worst case.

The Peterson/Dolev–Klawe–Rodeh Algorithm

Theorem 8.7.
Any decentralized wave algorithm for ring networks exchanges at least $\Omega(N \log N)$ messages, in the average as well as in the worst case.

Proof.
Follows from Theorems 8.5 and 8.6.

The Peterson/Dolev–Klawe–Rodeh Algorithm

Theorem 8.7.
Any decentralized wave algorithm for ring networks exchanges at least $\Omega(N \log N)$ messages, in the average as well as in the worst case.

Proof.
Follows from Theorems 8.5 and 8.6.

Corollary.
Any decentralized wave algorithm for ring networks exchanges at least $\Omega(|E| + N \log N)$ messages, in the average as well as in the worst case.

The Peterson/Dolev–Klawe–Rodeh Algorithm

Theorem 8.7.
Any decentralized wave algorithm for ring networks exchanges at least $\Omega(N \log N)$ messages, in the average as well as in the worst case.

Proof.
Follows from Theorems 8.5 and 8.6.

Corollary.
Any decentralized wave algorithm for ring networks exchanges at least $\Omega(|E| + N \log N)$ messages, in the average as well as in the worst case.

Extinction effect

An algorithm for leader election can be obtained from an arbitrary centralized wave algorithm by the application of a construction called **extinction** .

Each initiator starts a separate wave. The messages of the wave initiated by process p must all be tagged with p 's ID in order to distinguish them from the messages of different waves.

The algorithm ensures that, no matter how many waves are started, only one wave will run to a decision, namely, the wave of the smallest initiator. All other waves will be aborted before a decision can take place.

Extinction effect

Leader election algorithm $Ex(A)$ based on a wave algorithm A is defined as follows.

Each process is active in at most one wave at a time; this wave is its currently active wave denoted *caw* , with initial value *udef* .

Initiators of the election act as if they initiate a wave and set *caw* to their own ID.

Extinction effect

If a message of some wave, say, initiated by q , arrives at p , then p checks this message.

Extinction effect

If a message of some wave, say, initiated by q , arrives at p , then p checks this message.

- If $q > caw_p$, then the message is simply ignored, effectively causing q 's wave to be blocked at the node p .

Extinction effect

If a message of some wave, say, initiated by q , arrives at p , then p checks this message.

- If $q > caw_p$, then the message is simply ignored, effectively causing q 's wave to be blocked at the node p .
- If $q = caw_p$, then the message is treated exactly according to the wave algorithm.

Extinction effect

If a message of some wave, say, initiated by q , arrives at p , then p checks this message.

- If $q > caw_p$, then the message is simply ignored, effectively causing q 's wave to be blocked at the node p .
- If $q = caw_p$, then the message is treated exactly according to the wave algorithm.
- If $q < caw_p$ or $caw_p = udef$, then p joins the propagation of q 's wave by resetting its variables to their initial values and setting $caw_p := q$.

Extinction effect

If a message of some wave, say, initiated by q , arrives at p , then p checks this message.

- If $q > caw_p$, then the message is simply ignored, effectively causing q 's wave to be blocked at the node p .
- If $q = caw_p$, then the message is treated exactly according to the wave algorithm.
- If $q < caw_p$ or $caw_p = udef$, then p joins the propagation of q 's wave by resetting its variables to their initial values and setting $caw_p := q$.

When the wave initiated by q executes a decision event (in most wave algorithms this decision always takes place in q) the process q will be elected.

Extinction effect

HOMETASK 2.

1. Develope a leader election algorithm in arbitrary networks by applying extinction effect to the Echo Algorithm.
   Write a pseudo-code of the developed algorithm.
2. Prove the correctness of the algorithm You developed.
3. Estimate the message exchange complexity of the developed algorithm?

END OF LECTURE 8.