Algorithms

Characteristics / properties

Centralised

- centralized (single-source) / decentralized (multi-source)
- number of initiators

Topology

- ring, tree, star, clique, ...

Starting information

- identification of processes
- neighbourhood
- interconnection types (bidirectional/unidirectional)

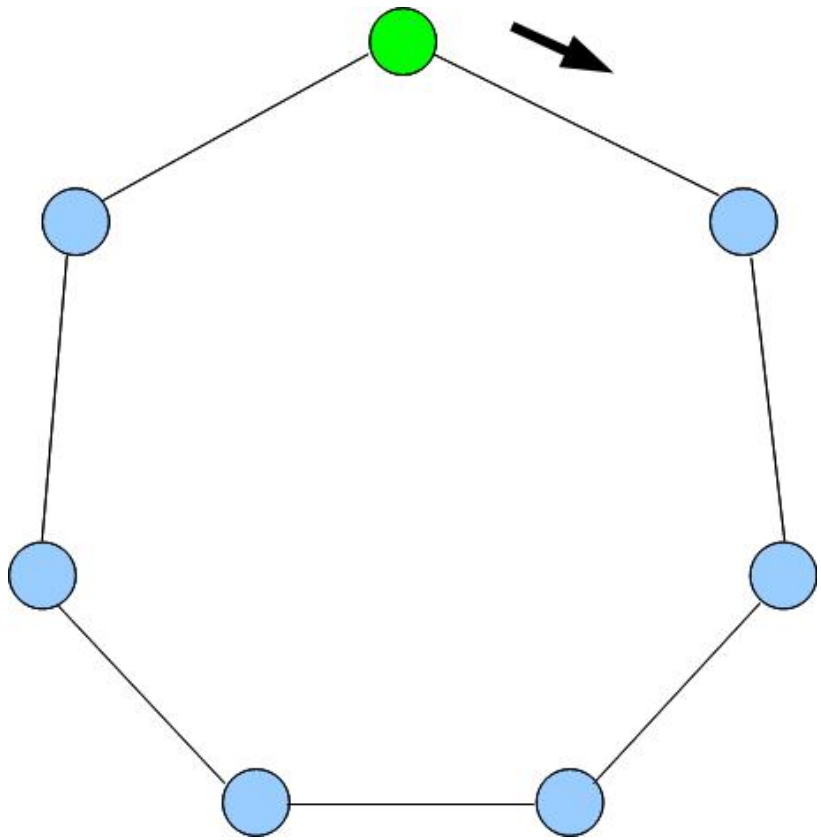
Computing termination

- in a single process
- in more or all processes

Complexity

- number of messages, steps of computation

Simple wave algorithm on the ring



Simple wave algorithm on the ring

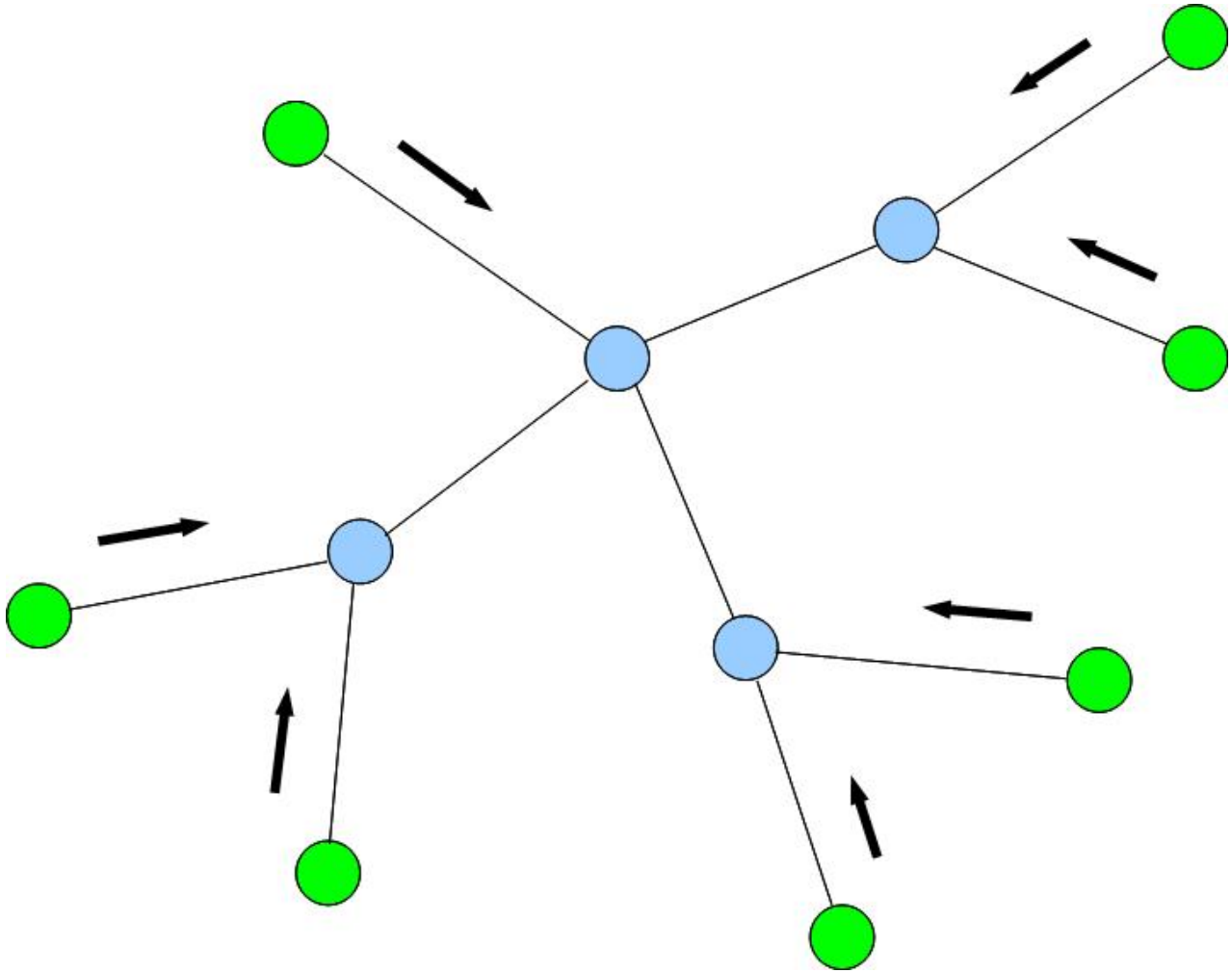
Starting process (initiator)

begin send (**tok**) to $Next_p$; receive (**tok**); *decide* . . . **end**

Other processes (non-initiators)

begin receive (**tok**); send (**tok**) to $Next_p$ . . . **end**

Simple wave algorithm on the tree



Simple wave algorithm on the tree

All proceses p

var $rec_p[q]$ for each $q \in Neigh[p]$: boolean **init** false;
(* $rec_p[q]$ is *true* when p has received a message from q *)

begin
while $\#\{q : rec_p[q] \text{ is false}\} > 1$ **do**
begin receive (**tok**) from q ; $rec_p[q]$:= true **end** ;

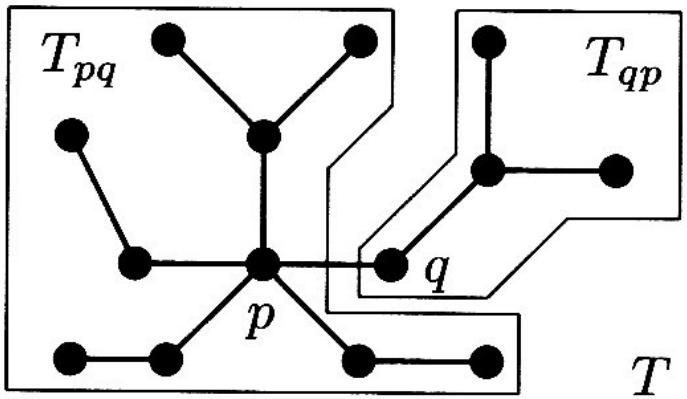
send (**tok**) to q_0 with $rec_p[q_0]$ is false ;
x: receive(**tok**) from q_0 ; $rec_p[q_0]$:= true ;
decide

(* Inform other processes of decision :
forall $q \in Neigh[p], q \neq q_0$ **do** send (**tok**) to q *)

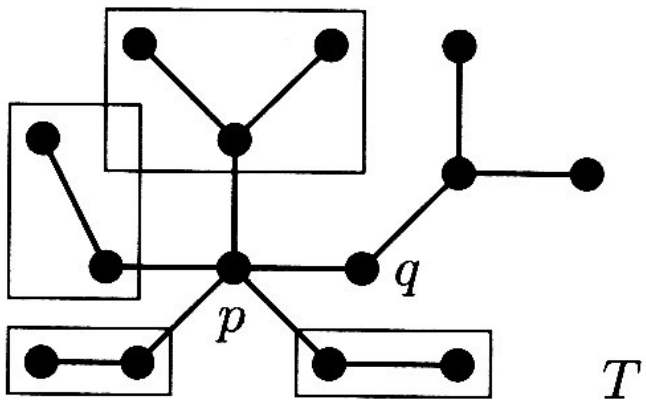
end

Simple wave algorithm on the tree

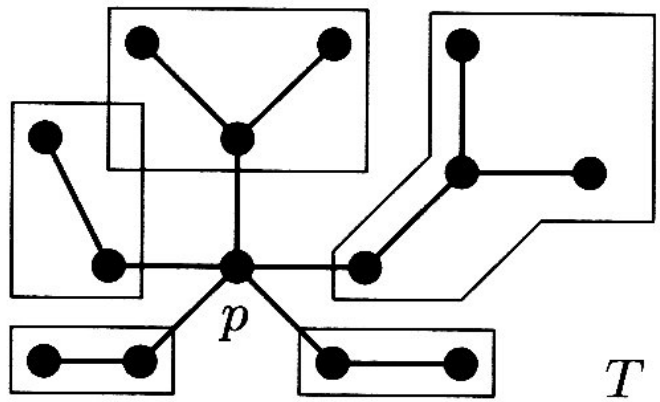
Example of computing



T_{pq} and T_{qp}



Decomposition of T_{pq}



Decomposition of T

Traversal algorithms

Definition

A traversal algorithm is an algorithm that satisfies the following three requirements:

1. In each computation there is a one initiator, which starts the algorithm by sending out exactly one message.

2. A process, upon receip of a message, either sends out one message or decides.

3. The algorithm terminates in the initiator and when this happens, each process has sent a message at least once.

Traversal algorithms

Sequential polling algorithm

var rec_p : integer **init** 0 ; (* for initiator only *)

For the initiator:

(* Write $Neigh_p = \{q_1, q_2, \ldots, q_{N-1}\}$ *)

begin while $rec_p < \#Neigh_p$ **do**

begin send $\langle$ **tok** $\rangle$ to q_{rec_p+1} ;

receive $\langle$ **tok** $\rangle$; $rec_p := rec_p + 1$

end ;

decide

end

For non-initiators:

begin receive $\langle$ **tok** $\rangle$ from q ; send $\langle$ **tok** $\rangle$ to q **end**

Selection algorithms

Symmetric algorithms

- same code and its execution (not the same state)
- high communication and computing requirements

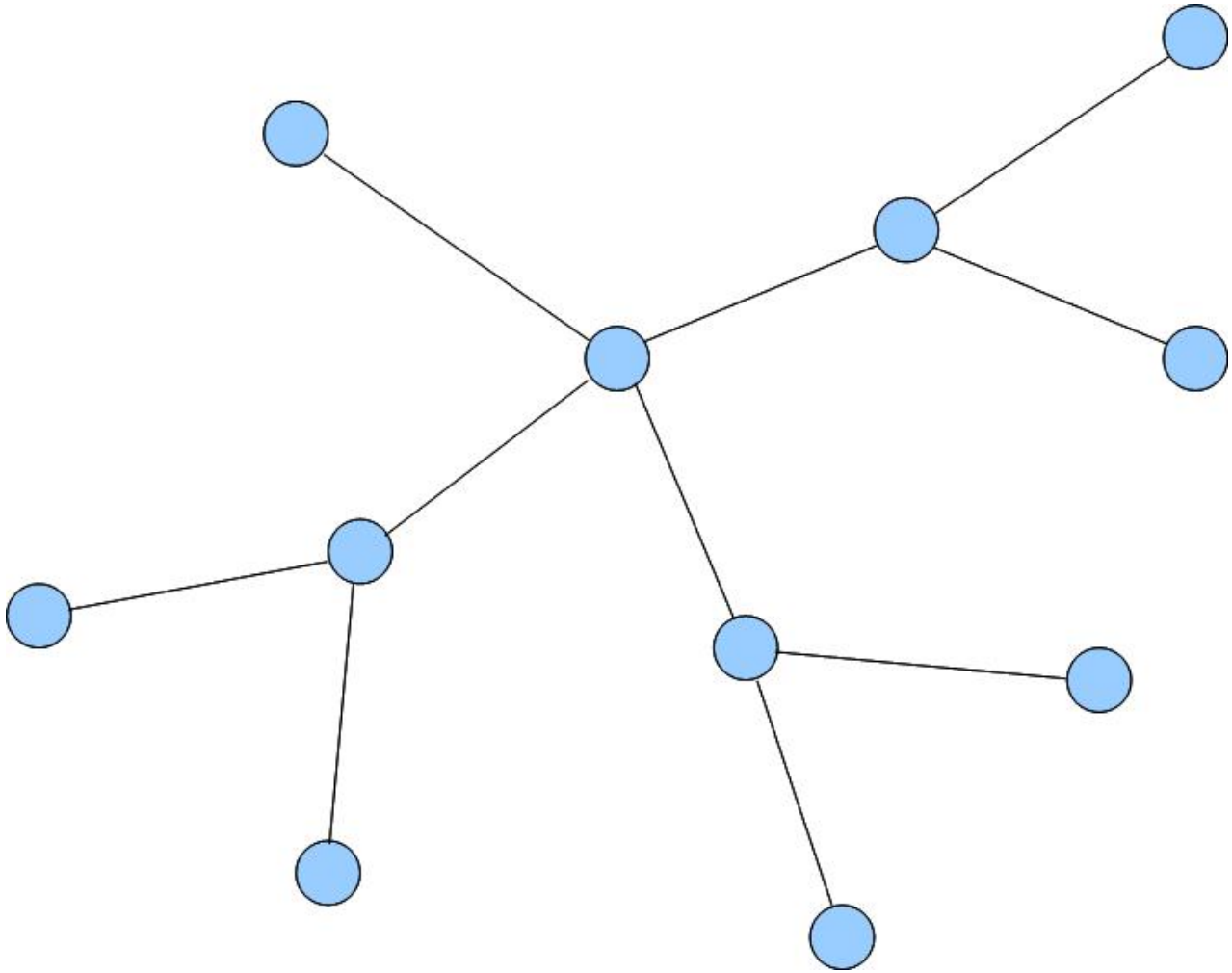
Code symmetry

- same code
- starting by selecting of a process that will work as the server
- (possibility to repeat the selection - effectiveness, failures)

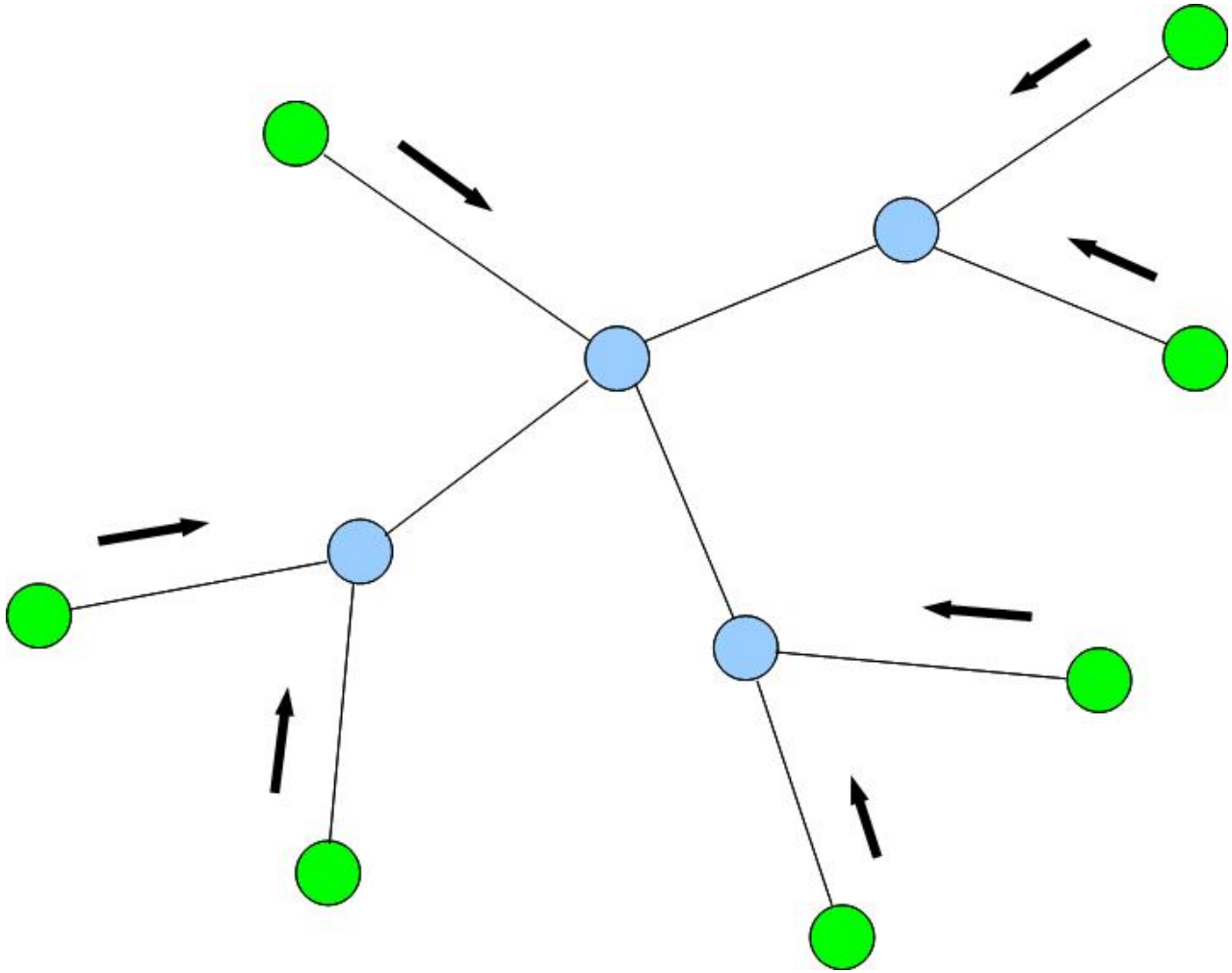
Asymmetric algorithms

- different code of processes (typically client-server)
- lower requirements for communication and computing
- (needs to solve failures -> cloud technologies)

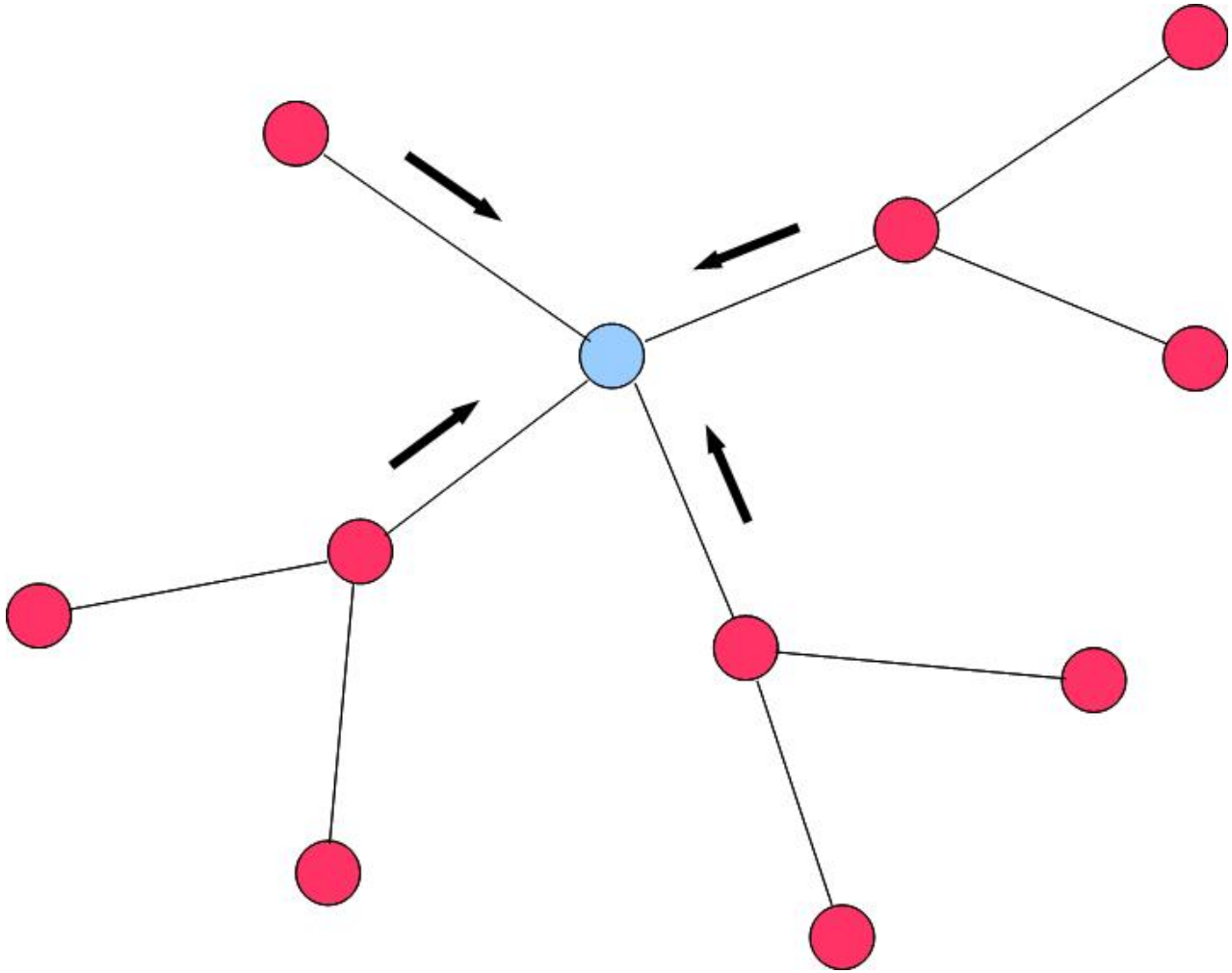
Selection of the server on the tree



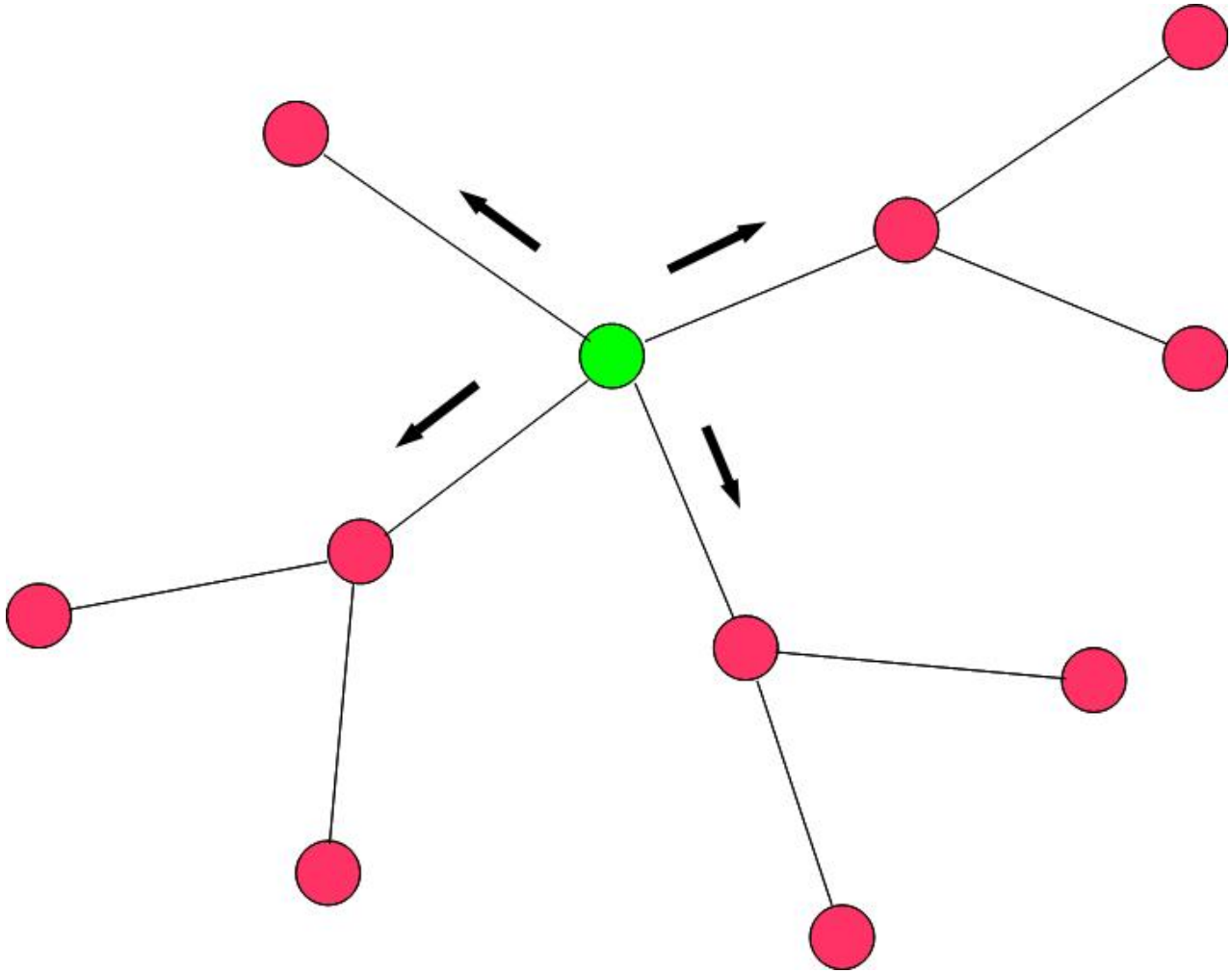
Selection of the server on the tree



Selection of the server on the tree



Selection of the server on the tree



Selection of the server on the tree

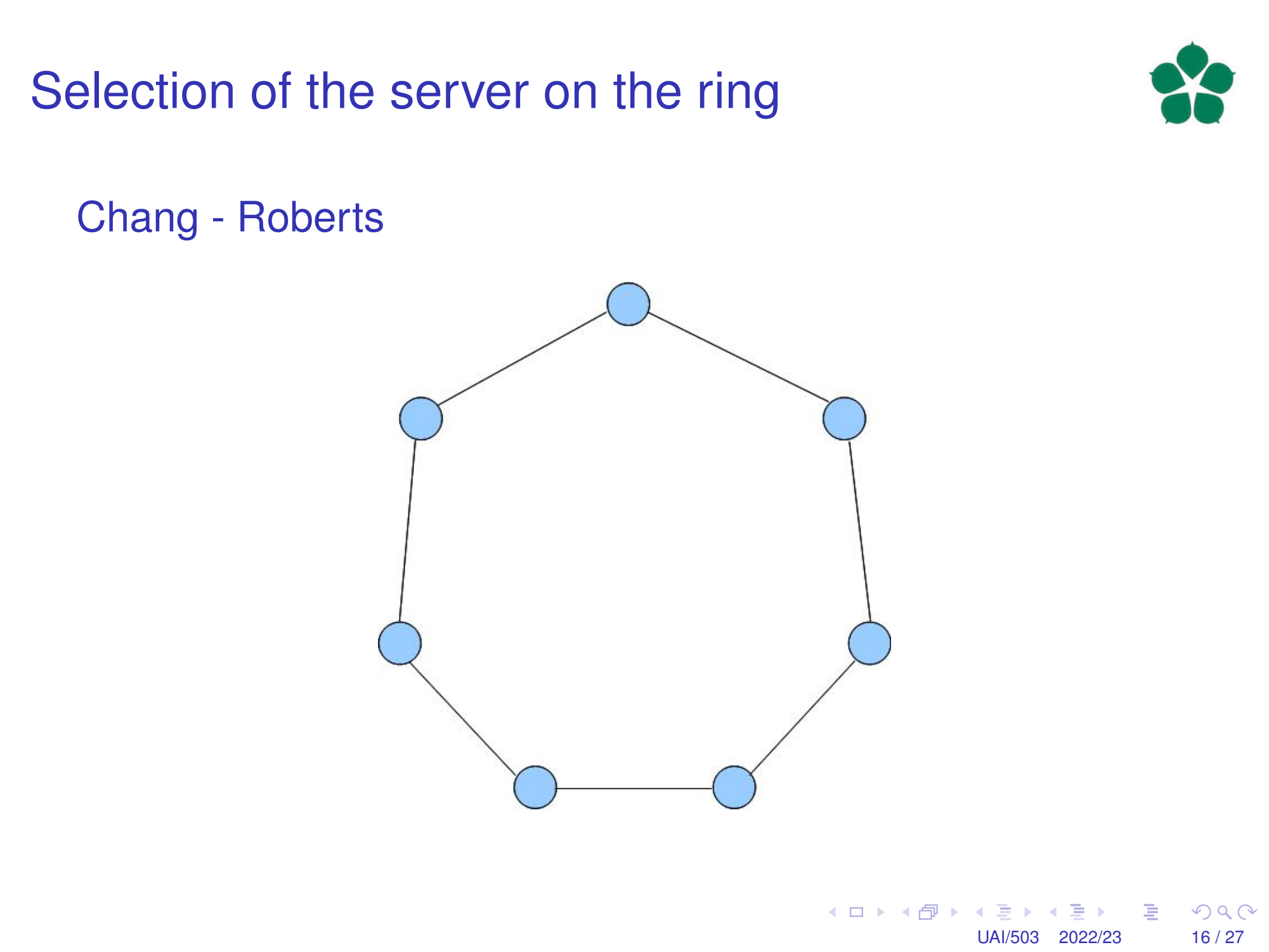
```
var   ws_p :      boolean;              init false;
      wr_p :      boolean;              init 0;
      rec_p[q]:   boolean;              init false;
      v_p :       P;                    init p;
      state_p :   {sleep,leader,lost};  init sleep;

begin if p is initiator then
      begin ws_p := true;
          forall q in 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 in Neigh_p do send (wakeup) to q
                  end
          end;
```

Selection of the server on the tree

```
        while #{q : ~rec_p[q]} > 1 do
            begin receive(tok,r) from q; rec_p[q] := true;
                v_p := min(v_p,r)
            end;
        send(tok,v_p) to q_0 with ~rec_p[q_0];
        receive(tok,r) from q_0;
        v_p := min(v_p,r);
        if v_p=p then state := leader else state := lost;
        forall q in Neigh_p, q≠q_0 do send(tok,v_p) to q
end
```

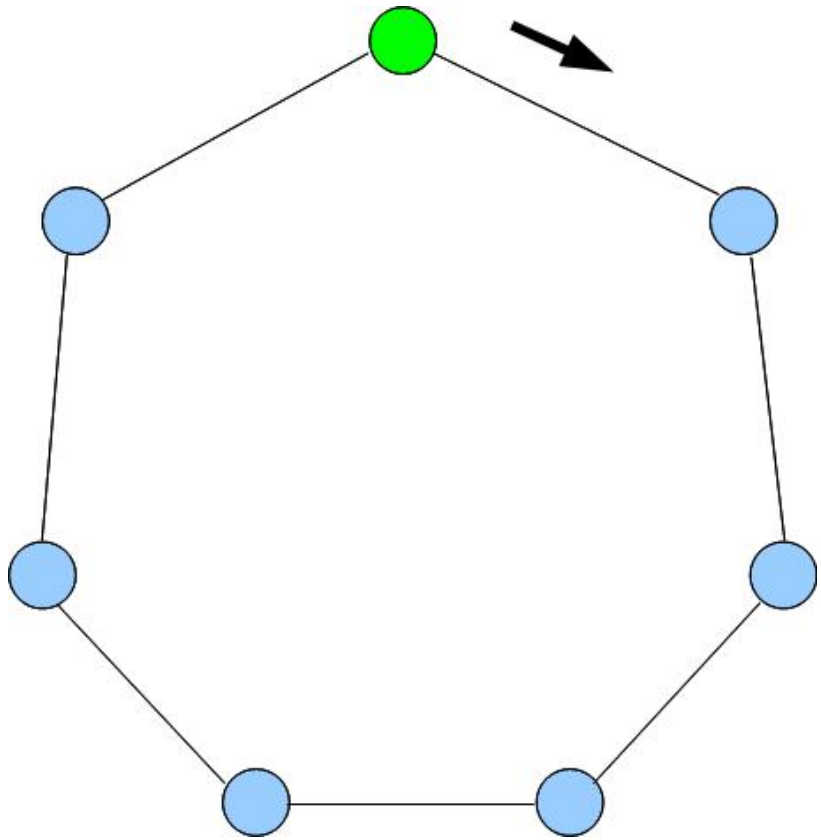
Selection of the server on the ring



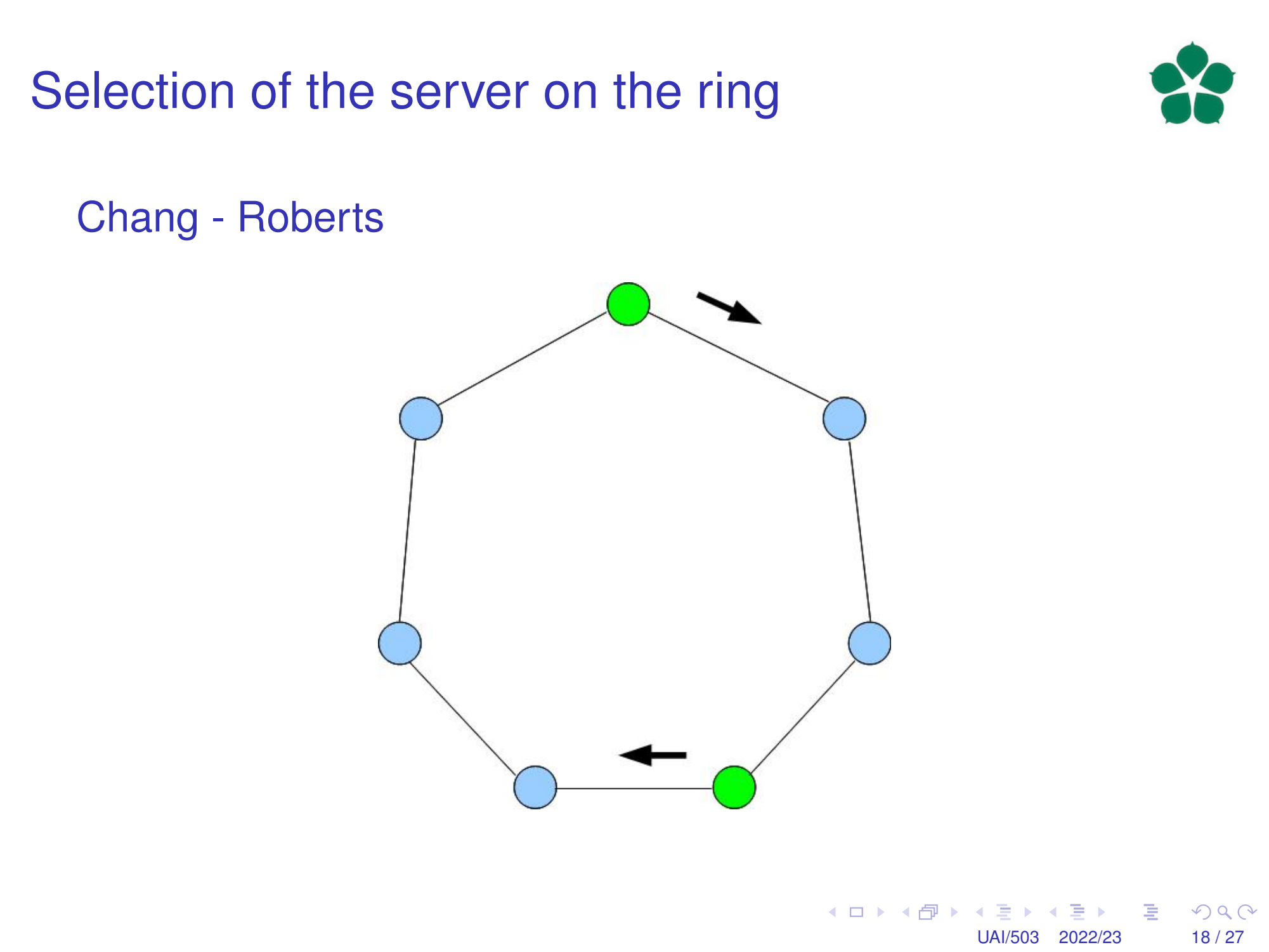
Chang - Roberts

Selection of the server on the ring

Chang - Roberts



Selection of the server on the ring



Chang - Roberts

Selection of the server on the ring

Chang - Roberts

```
var   Voting;
      Coordinator;

when decision INITIATE_ELECTION do          { rozhodnutí volit }
      begin
          Voting:=T;
      sendl ELECTION(i)
      end
```

Selection of the server on the ring

Chang - Roberts

```
when received ELECTION(j) do                    { příjem zprávy ELECTION }
        begin
            if j>i then
                begin
                    sendl ELECTION(j);
                    Voting:=T
                end;
            if j<i and not Voting then
                begin
                    sendl ELECTION(MyNumber);
                    Voting:=T
                end;
            if j=i then
                begin
                    sendl ELECTED(i)
                end
        end
```

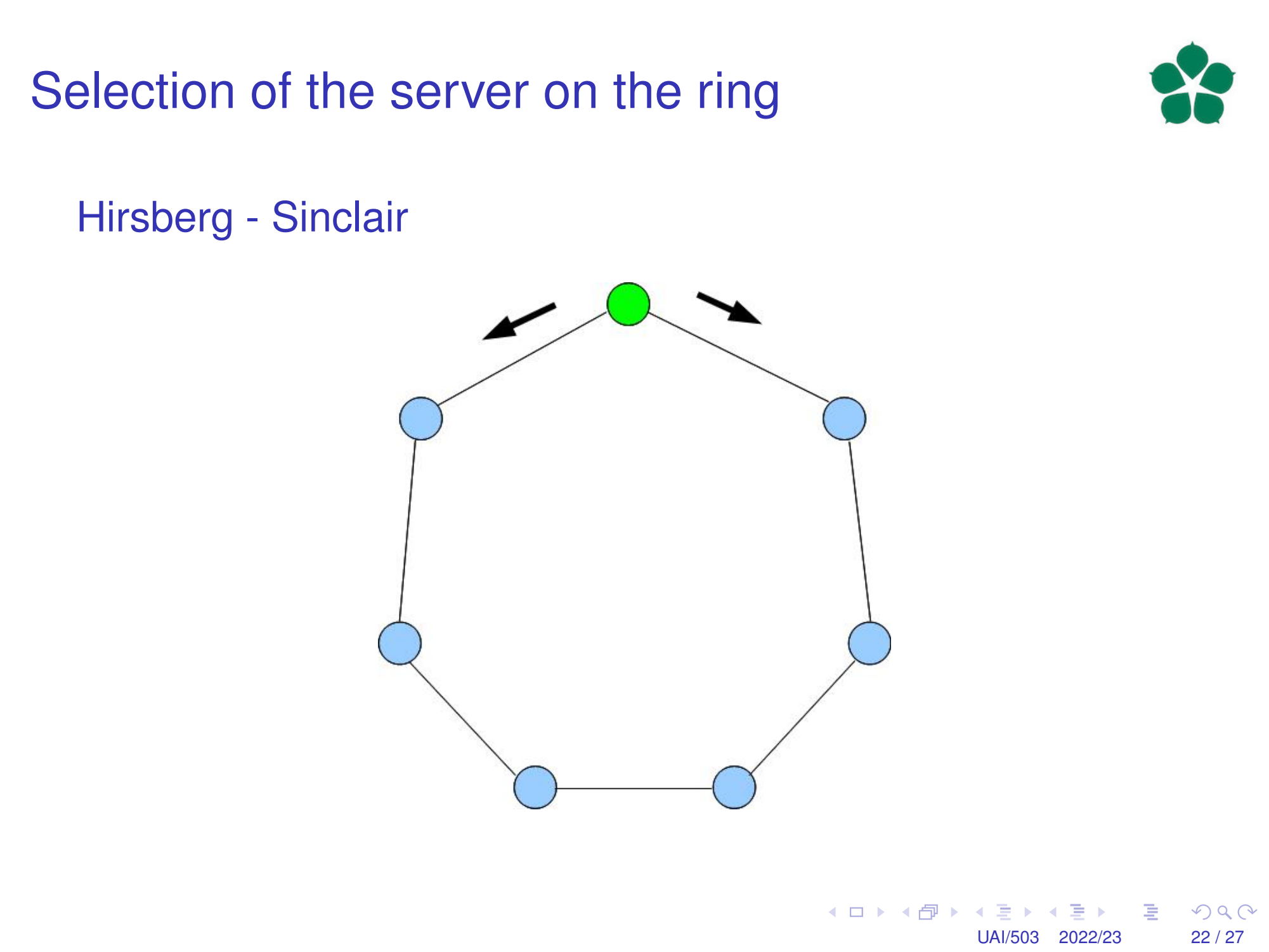
Selection of the server on the ring

Chang - Roberts

```
when received ELECTED(j) do                    { příjem zprávy ELECTED }
        begin
            Coordinator:=j;
            Voting:=F;
            if j<>i then sendl ELECTED(j)
        end

begin                                          { inicializace }
        Voting:=F; Coordinator:=0
end
```

Selection of the server on the ring



Hirsberg - Sinclair

Selection of the server on the ring

Hirsberg - Sinclair

```
when decision INITIATE_ELECTION do rozhodnutí volit
    begin
        State := CANDIDATE;
        lmax := 1;
        while State=CANDIDATE do
            begin
                Nresp := 0;
                RespOK := T;
                sendlr CANDIDATURE(i,0,lmax);
                wait NResp=2;
                if not RespOK then State := LOST;
                lmax := 2*lmax
            end
    end

when received RESPONSE(r,j) do příjem zprávy RESPONSE
    if j=i  then
        begin
            Nresp := NResp+1;
            RespOK := RestOK and r
        end
    else
        pass RESPONSE(r,j]
```

Selection of the server on the ring

Hirsberg - Sinclair

```
when received CANDIDATURE(j,l,lmax) do příjem zprávy CANDIDATURE
    begin
        if j<i then
            begin
                respond RESPONSE(F,j);
                if State=NOT_INVOLVED then INITIATE_ELECTION
            end;
        if j>i then
            begin
                State := LOST;
                l := l+1;
                if l<lmax then pass CANDIDATURE(j,l,lmax)
                else respond RESPONSE(T,j)
            end;
        if j=i then
            begin
                if State<>ELECTED then State:=ELECTED;
                Winner := i;
                pass ELECTED(i)
            end
    end
```

Selection of the server on the ring

Hirsberg - Sinclair

```
when received ELECTED(j) do příjem zprávy ELECTED
    if Winner<>j then
        begin
            pass ELECTED(j);
            Winner := j;
            State := NOT_INVOLVED
        end

begin                                   inicializace
    Nresp := 0; RespOK := T
end
```

Selection of the server on the ring

Dolev, Klawe, Rodeh

- two requests sent in the same (clockwise) direction
- one to the neighbourgh, second to the neighbourgh's neighbourgh
- similar to Hirsberg-Sinclairova algorithm