Distribuované systémy a výpočty

X36DSV

Jan Janeček
(dnes Peter Macejko)

Výlučný přístup

Algoritmy na úplném grafu

Lamport

- základní algoritmus, 3(n-1) zpráv/požadavek

Ricart – Agrawala

- pozdržení souhlasů, 2(n-1) zpráv/požadavek

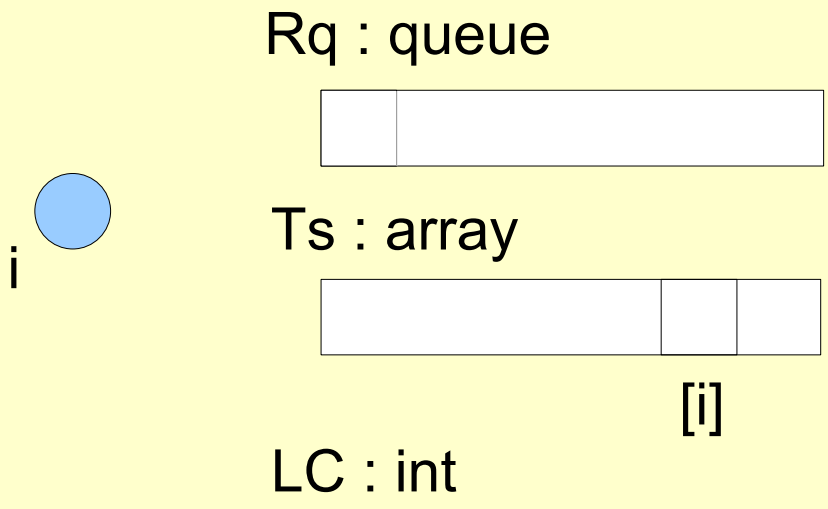
Carvalho – Roucairol

- kredity pro přístup, 0 – 2(n-1) zpráv/požadavek

Ricart – Agrawala

- požadavkový token, n zpráv/požadavek

Lamport




```
{ initialization }
begin
        LC := 0;
        for j := 1 to N do
                begin
                        Ts[j] := 0; Rq[j] := ∞
                end
end
```

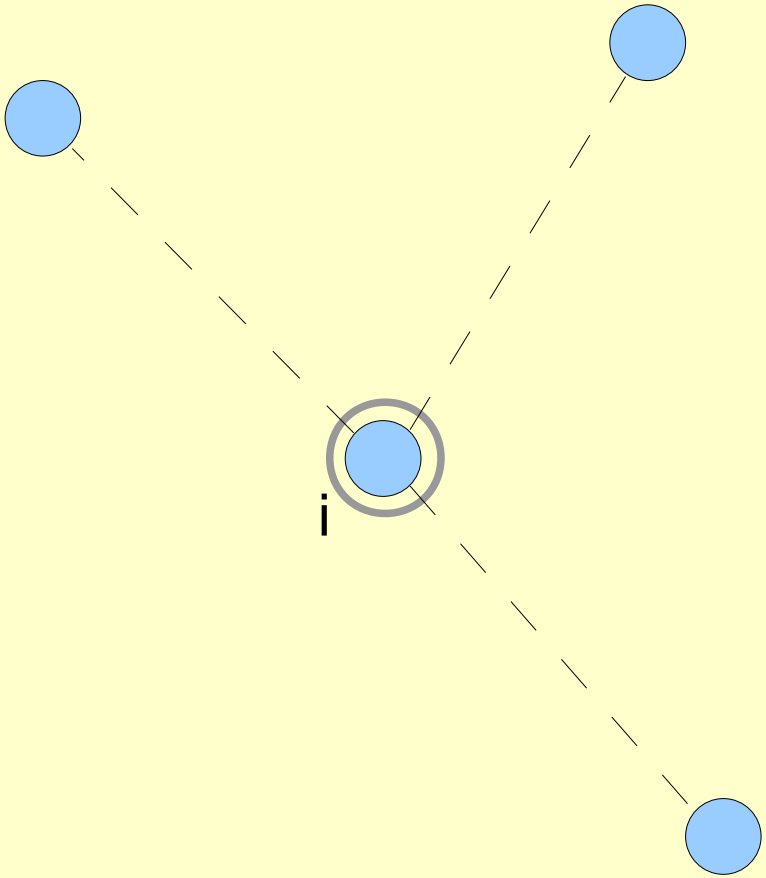
Legenda:

- LC – local clock (lokální čas)
- Rq – request queue (prioritní fronta)
- Ts – time stamps (čas poslední zprávy)

Komunikační primitiva:

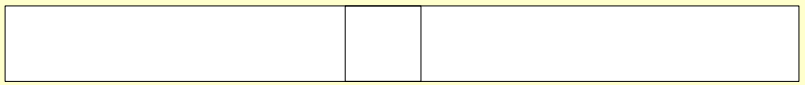
- Request(LCi, i)
- Response(LCj, j)
- Release(LCi, i)

Lamport



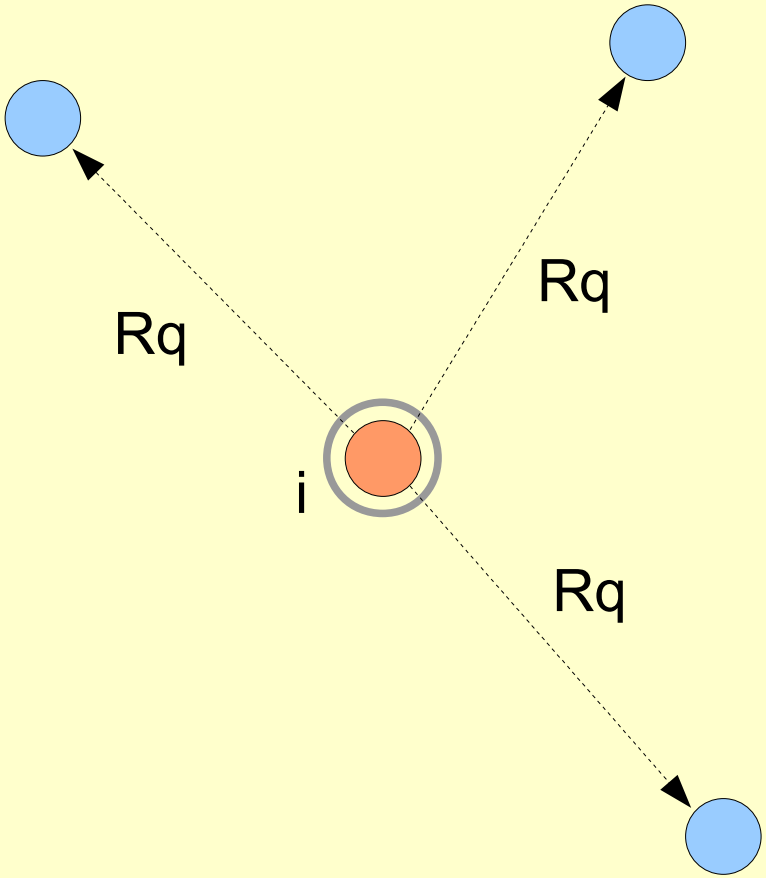
Rq : queue

Ts : array



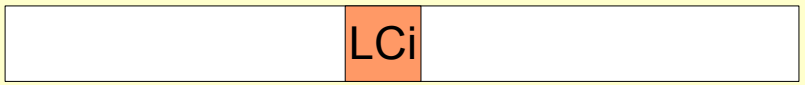
[i]

Lamport



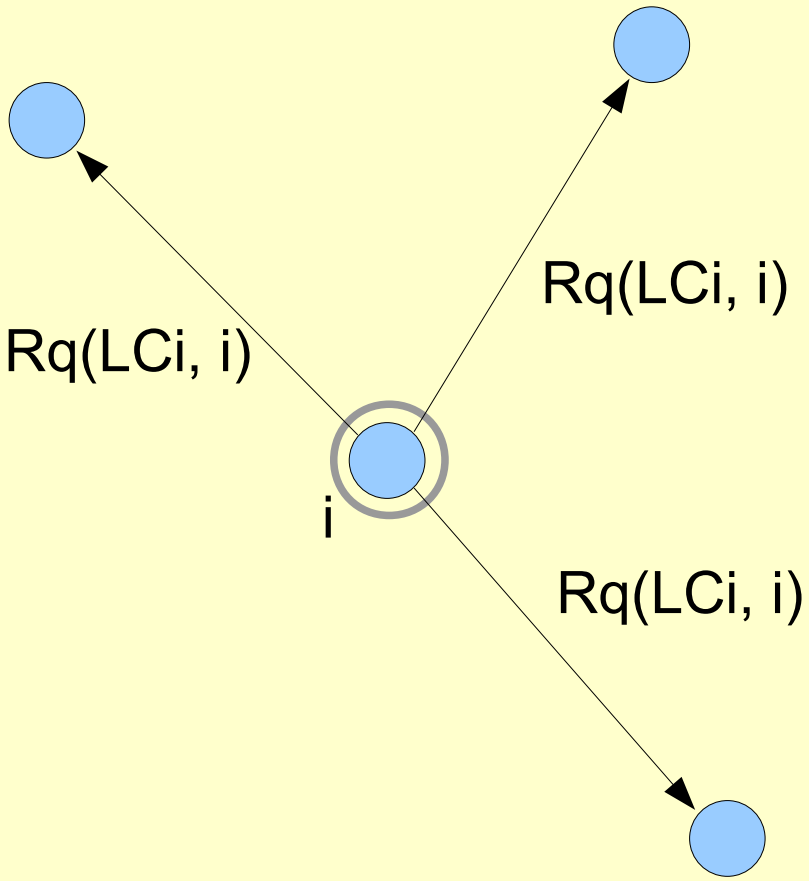
Rq : queue

Ts : array



Odesláno (n-1) zpráv

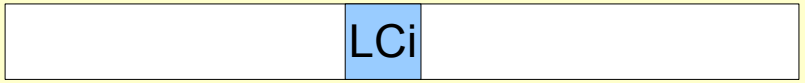
Lamport



Rq : queue

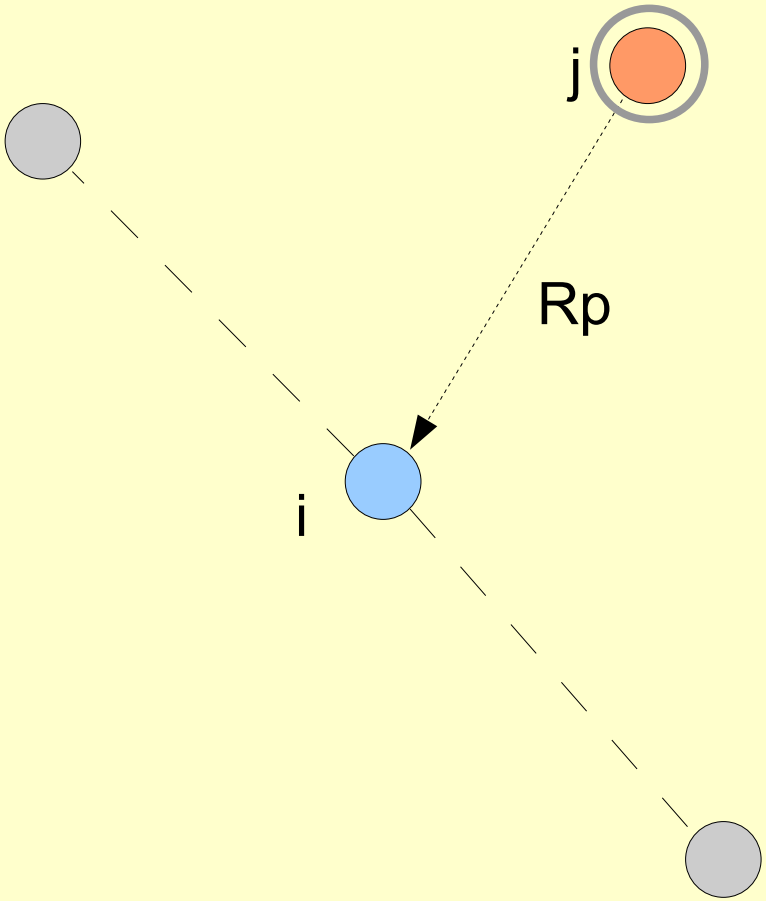
i

Ts : array



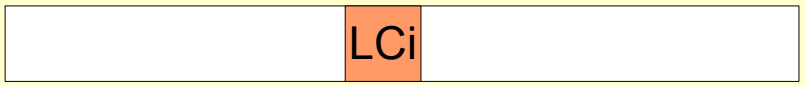
[i]

Lamport

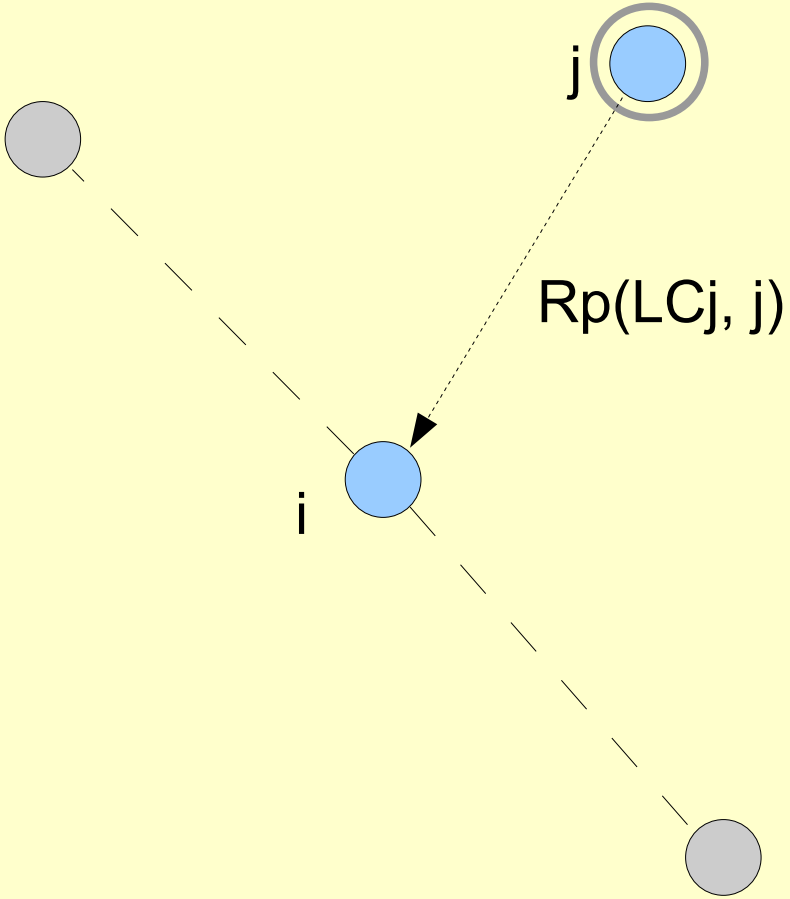


Rq : queue

Ts : array

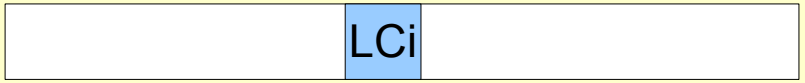


Lamport

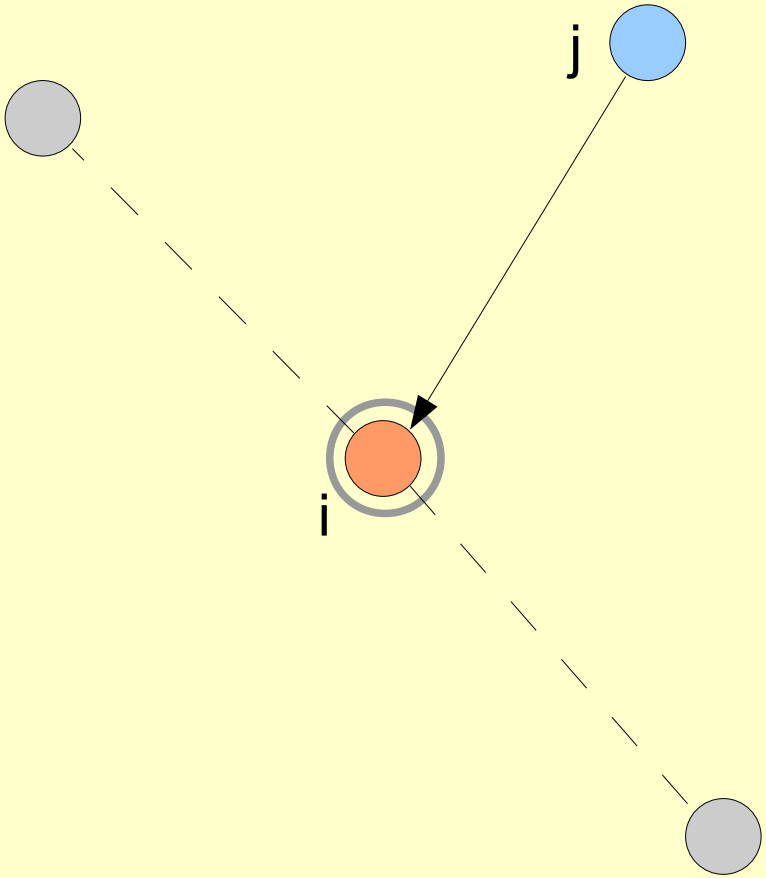


Rq : queue

Ts : array

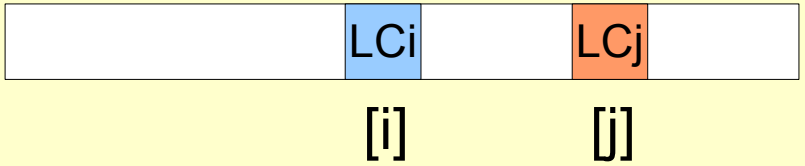


Lamport



Rq : queue

Ts : array

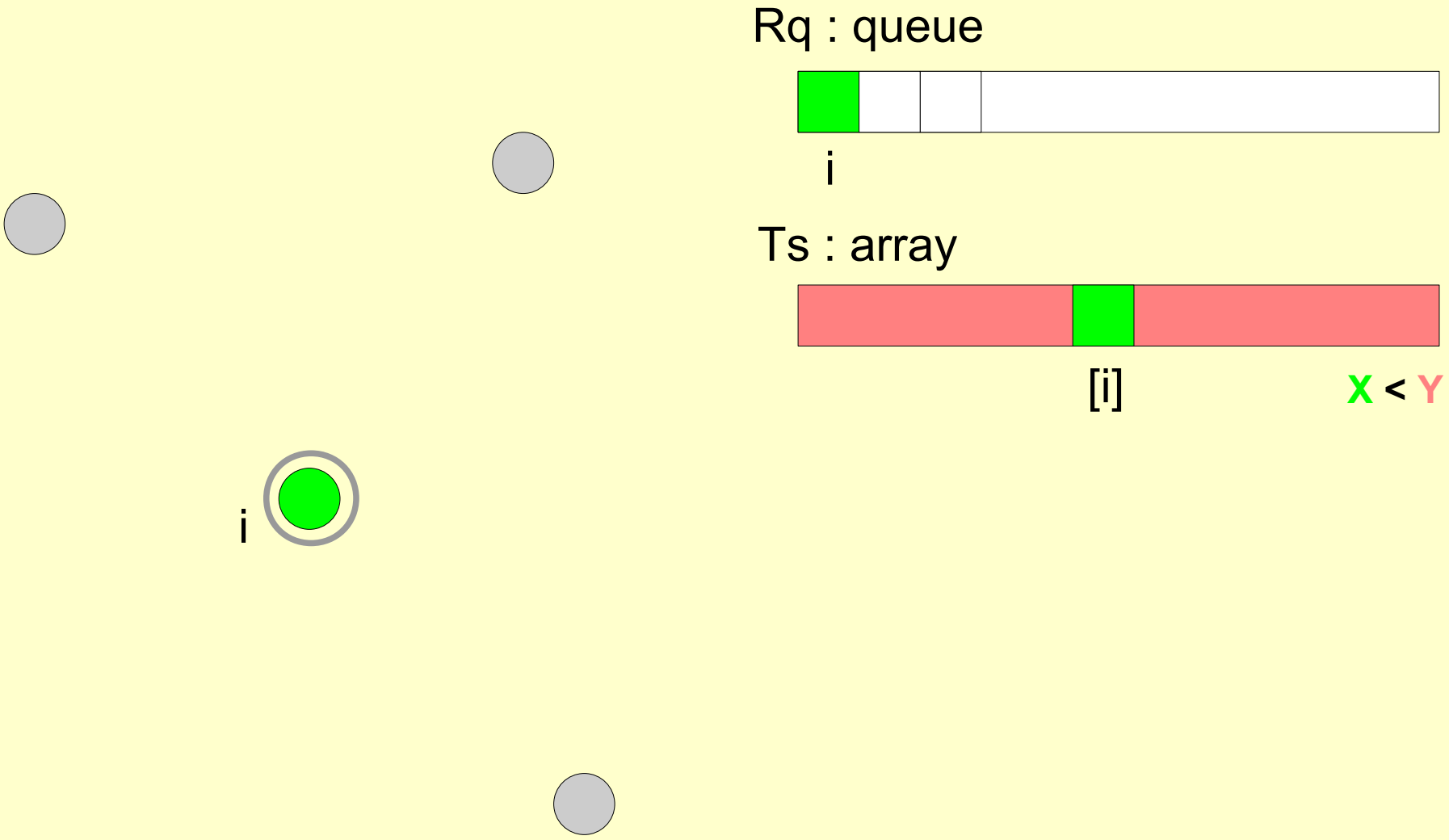


Přijato (n-1) zpráv

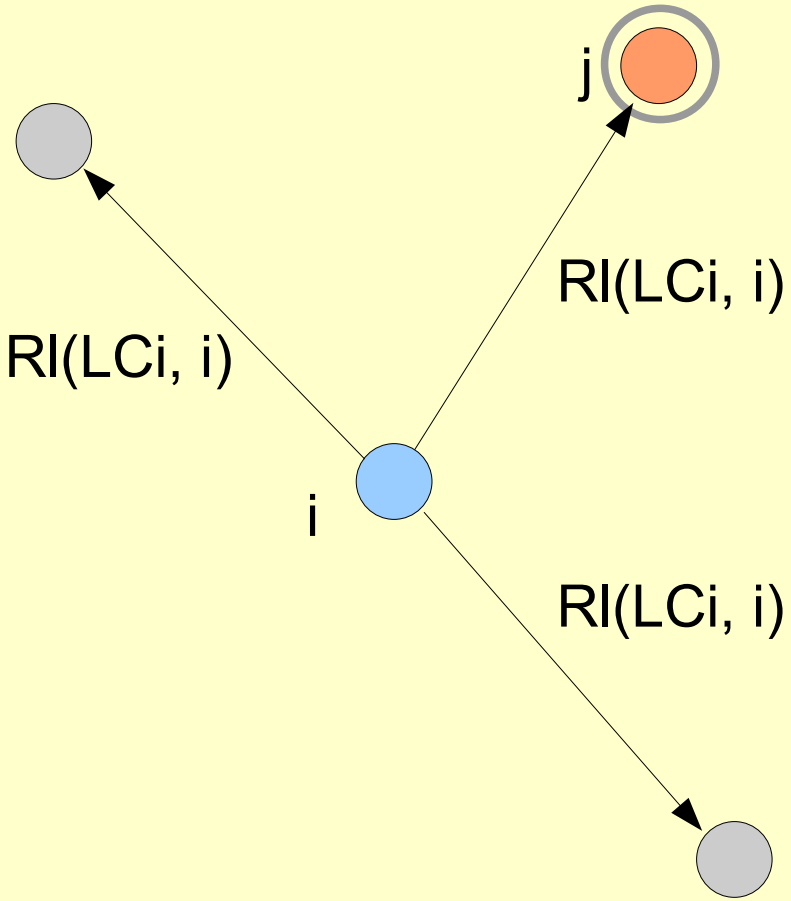
Lamport

```
when request                                          { access request }
        begin
            [P] Rq[i] := LC; Ts[i] := LC; LC := LC+1; [V]
            for j:=1  to N  do
                if j≠i  then
                    send REQUEST(LC,i) to j

when received REQUEST(ts,j)                           { j-th process request }
        begin
            [P] LC := max(LC,ts); LC := LC+1; [V]
            Rq[j] := ts; Ts[j] := ts;
            send RESPONSE(LC,i) to j
        end

when received RESPONSE(ts,j)                          { j-th process response }
        begin
            [P] LC := max(LC,ts); LC := LC+1; [V]
            Ts[j] := ts
        end
```

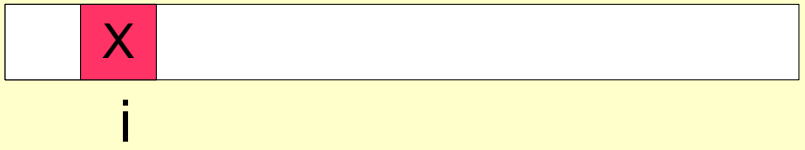
Lamport



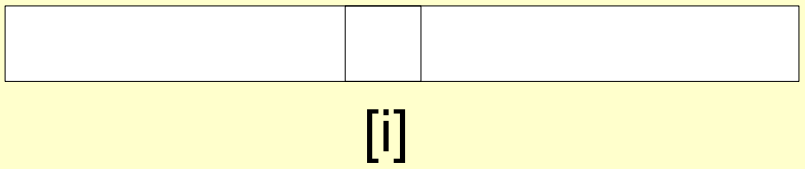
Lamport



Rq : queue



Ts : array



Odesláno (n-1) zpráv

Lamport

when (Rq[i]<Rq[j] **forall** j≠i) **and** (Rq[i]<Ts[j] **forall** j≠i)
begin

{ critical section }

send RELEASE(LC) **to** j
end

when received RELEASE(ts,j) { j-th process release }
begin
[P] LC := max(LC,ts); LC := LC+1; [V]
Rq[j] := ∞;

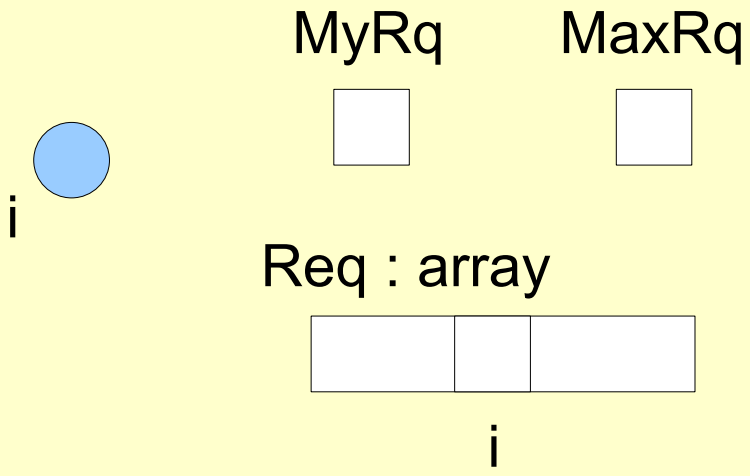
end

Zprávy

request + response + release

(n-1) + (n-1) + (n-1) = **3x(n-1)**

Ricart - Agrawala




```
{ initialization }
begin
        MaxRq:=0; MyReq:=F;
        for j:=1  to N  do
              Req[j]:=F
end
```

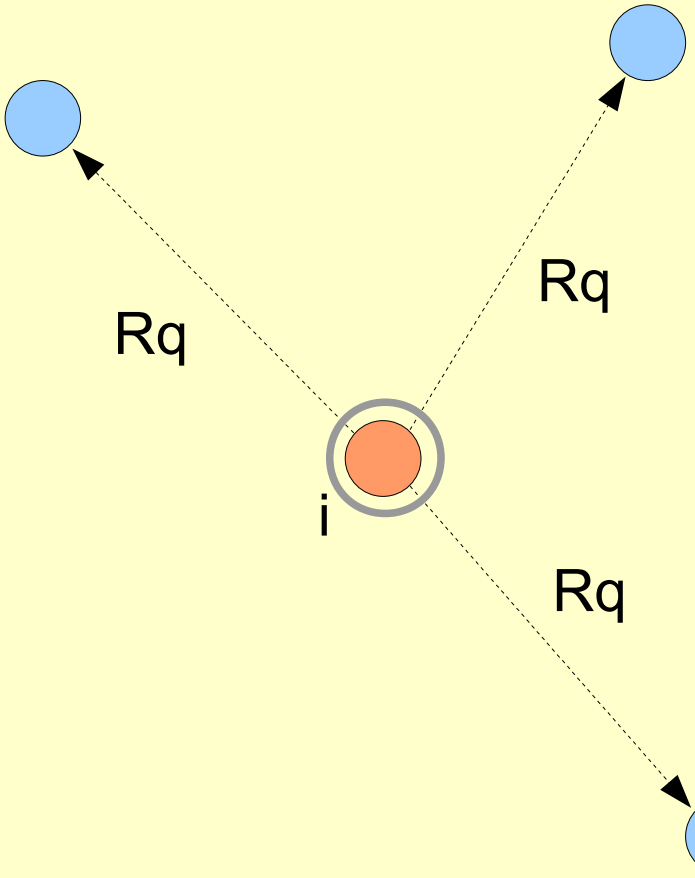
Legenda:

- MyRq – sekvenční číslo vlastní žádosti
- MaxRq – maximální sekvenční číslo
- Req – pole registrací žádostí na vstup do CS

Komunikační primitiva:

- Request(MyRqi, i)
- Response()

Ricart - Agrawala



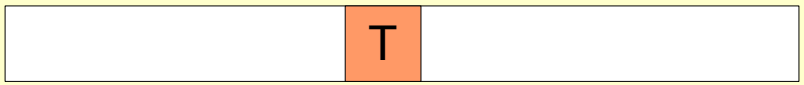
MyRq

S+1

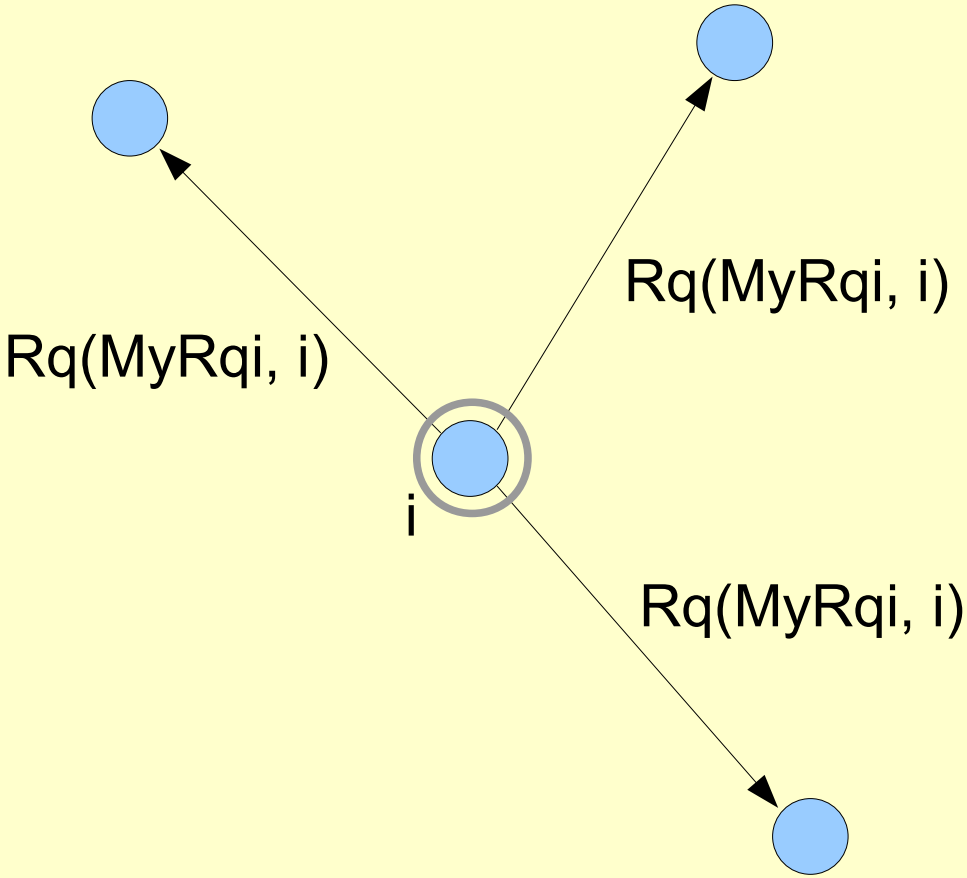
MaxRq

S

Req : array



Ricart - Agrawala



MyRq S+1

MaxRq S

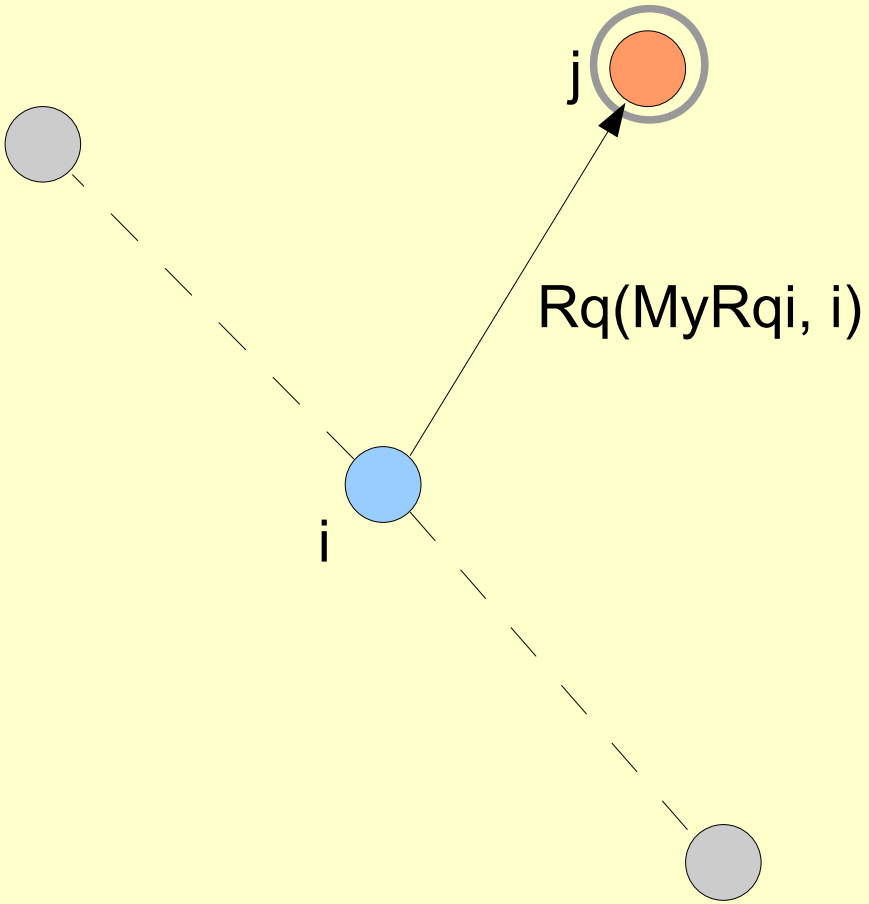
Req : array

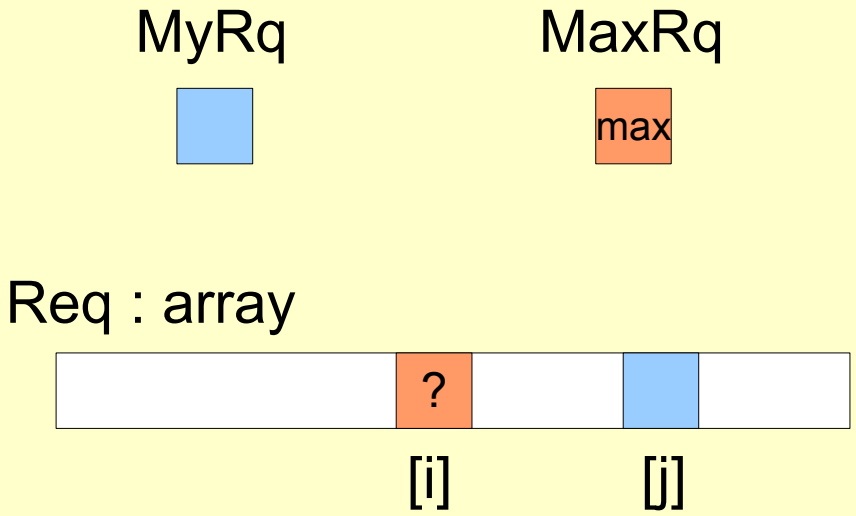
T

[i]

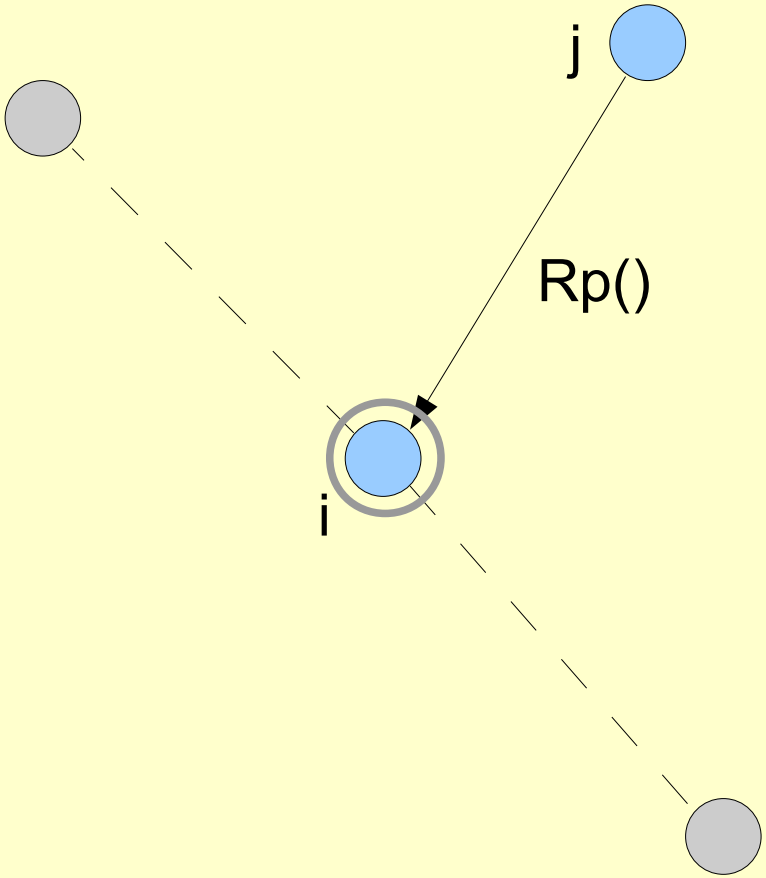
Odesláno (n-1) zpráv

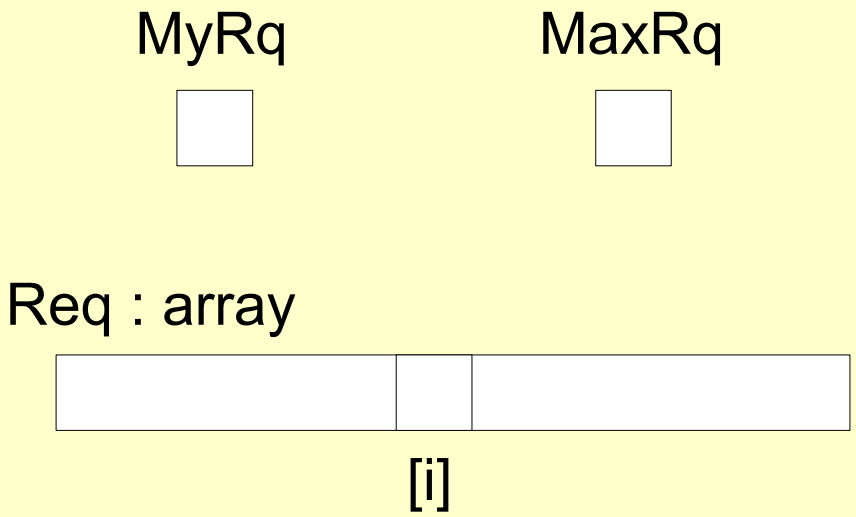
Ricart - Agrawala





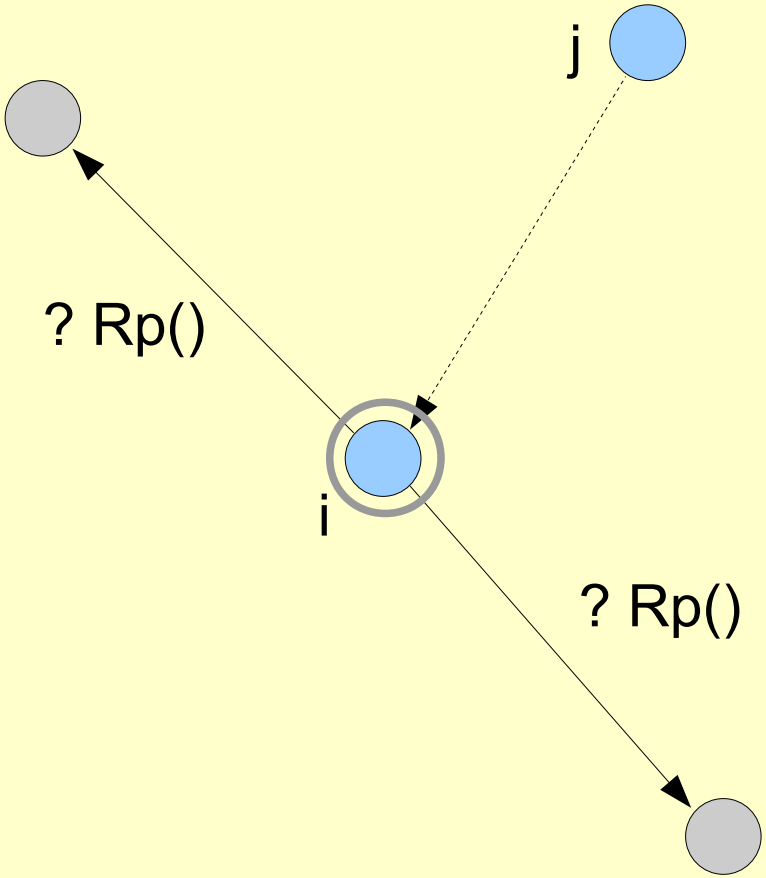
Ricart - Agrawala

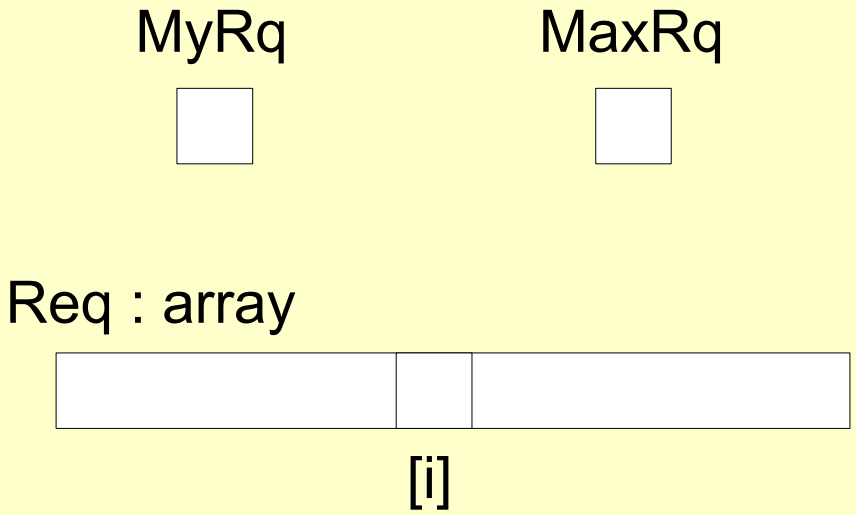




Přijato (n-1) zpráv

Ricart - Agrawala





Ricart - Agrawala

```
when request                                        { access request }
    begin
        [P] Req[i] := T; MyRq := MaxRq+1; [V]
        RpCnt := 0;
        for j:=1  to N  do
            if j≠i  then
                send REQUEST(MyRq,i)  to j;
        wait RpCnt=N-1;
        { critical section }
        Req[i] := F;
        for j:=1  to N  do                          { delayed responses }
            if Req[j]  then
                begin
                    Req[j]:=F;
                    send REPLY  to j
                end
    end
```

Ricart - Agrawala

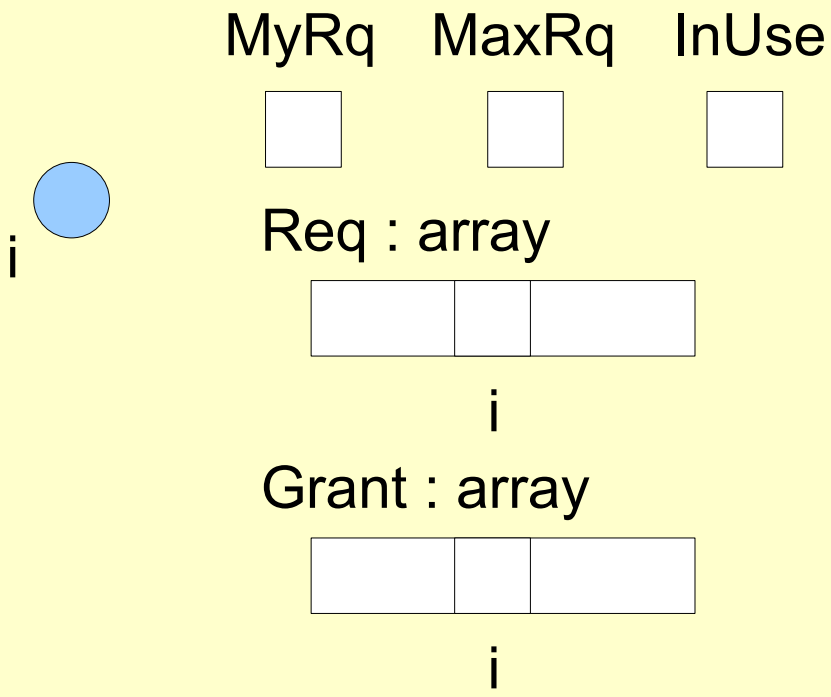
```
when  received REQUEST(k,j)  do          { request of the k-th process }
      begin
          MaxRq := max(MaxRq,k);
          [P] Delay := Req[i] and ((k>MyRq) or (k=MyRq and j>i)); [V]
          if Delay then
              Req[j] := T
          else
              send REPLY  to j
      end

when  received REPLY  do                 { response of any process }
      RpCnt:=RpCnt+1;
```

Zprávy

request + response

(n-1) + (n-1) = **2x(n-1)**

Carvalho - Roucairol




```
{ initialization }

begin
        MaxRq := 0; MyRq := 0;
        for j:=1 to N do
            begin
                Req[j] := F;
                Grant[j] := F
            end
end
```

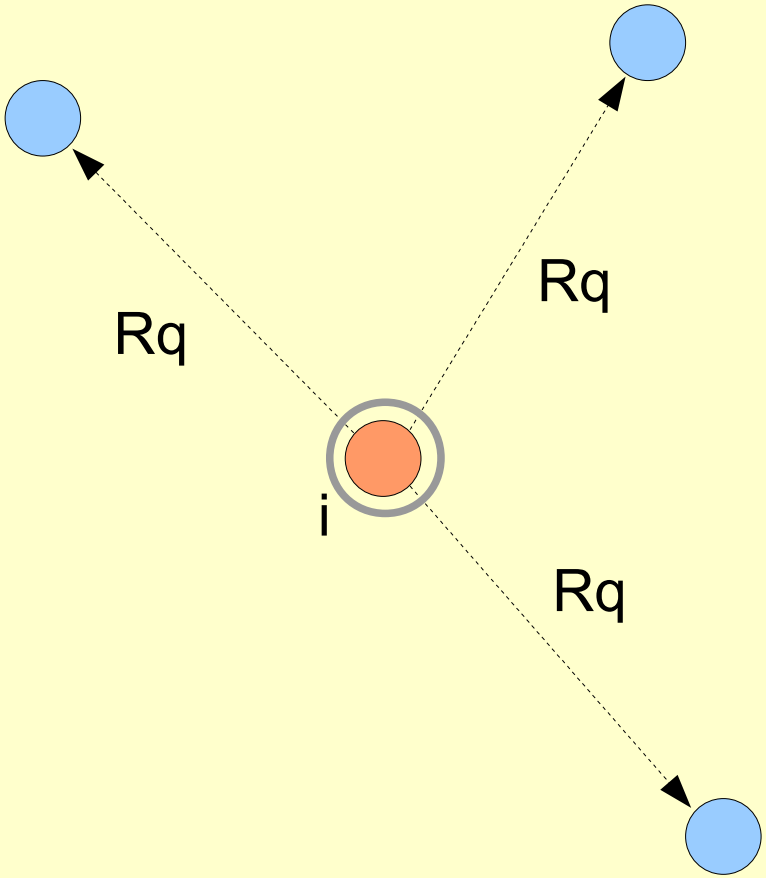
Legenda:

- MyRq – sekvenční číslo vlastní žádosti
- MaxRq – maximální sekvenční číslo
- InUse – identifikátor kritické sekce
- Req – pole registrací žádostí na vstup do CS
- Grant – pole aktivních 'pověření'

Komunikační primitiva:

- Request(MyRqi, i)
- Reply(i)

Carvalho - Roucairol



MyRq MaxRq

S+1 S

Req : array

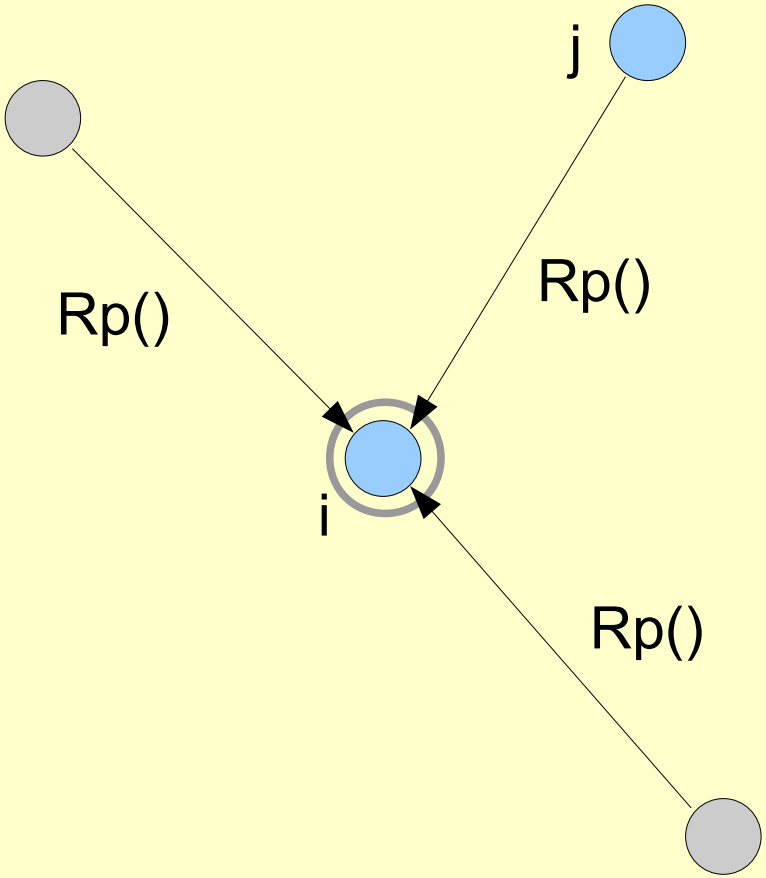
T

[i]

Grant : array;

[i]

Carvalho - Roucairol



MyRq MaxRq

Req : array

[i]

Grant : array;

[i]

Carvalho - Roucairol

```
when request                                    { access request }
        [P] Req[i] := T; MyRq := MaxRq+1; [V]
        for j:=1 to N do
            if j≠i and (not Grant[j]) then
            send REQUEST(MyRq,i) to j;
        wait (Grant[j]=T forall j≠i);
        Req[i] := F; InUse := T;

        { critical section }

        InUse := F;
        for j:=1 to N do                        { delayed responses }
            if Req[j] then
                begin
                    Grant[j] := F; Req[j] := F;
                    send REPLY to j
                end
```

Carvalho - Roucairol

```
when received REQUEST(k,j) do      { j-th process request }
    begin
        MaxRq := max(MaxRq,k);
        [P] Delay := ((k>MyRq) or (k=MyRq and j>i)) [V]
        if InUse or (Req[i] and Delay) then
            Req[j]:=T;
        if not (InUse or Req[i]) or
              (Req[i] and (not Grant[j]) and (not Delay)) then
            send REPLY(i) to j;
        if (Req[i] and Grant[j] and (not Delay)) then
            begin
                Grant[j]:=F;
                send REPLY(i) to j;
                send REQUEST(MyRq,i) to j
            end
    end
```

Carvalho - Roucairol

when received REPLY **from** j **do** { j-th process response }
    Grant[j] := T

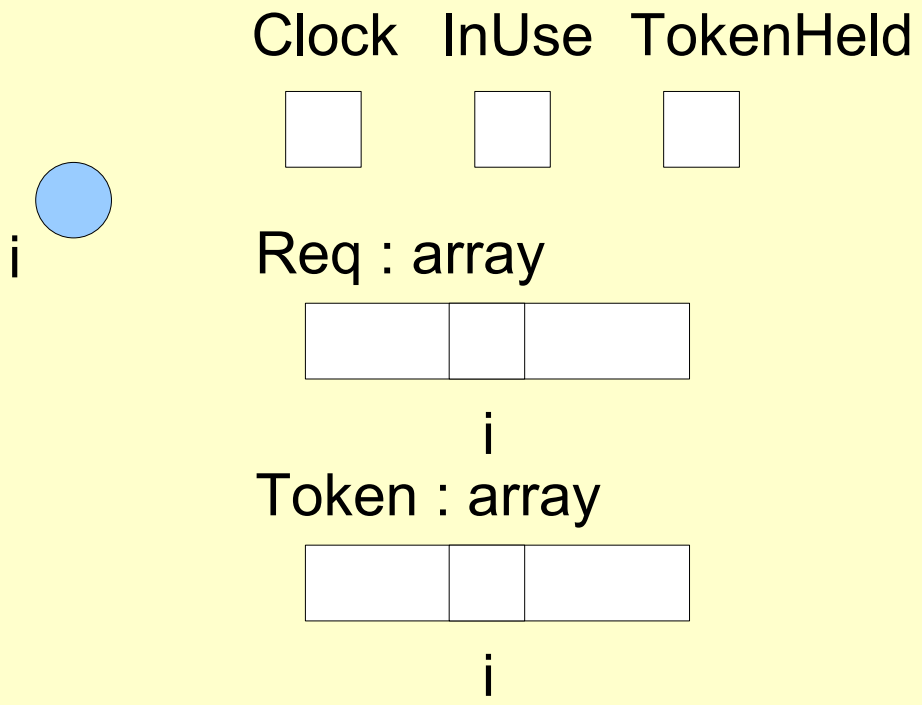
Zprávy

request + response

(n-1) + (n-1) = **2x(n-1)**

0 + 0 = **0**

Ricart - Agrawala (token passing)




```
{ initialization }
begin
    for j:=1 to N do
        Req[j] := 0; Token[j] := 0;
    Clock := 0;
    InUse := F;
    TokenHeld := (myID == 0)?T:F
end
```

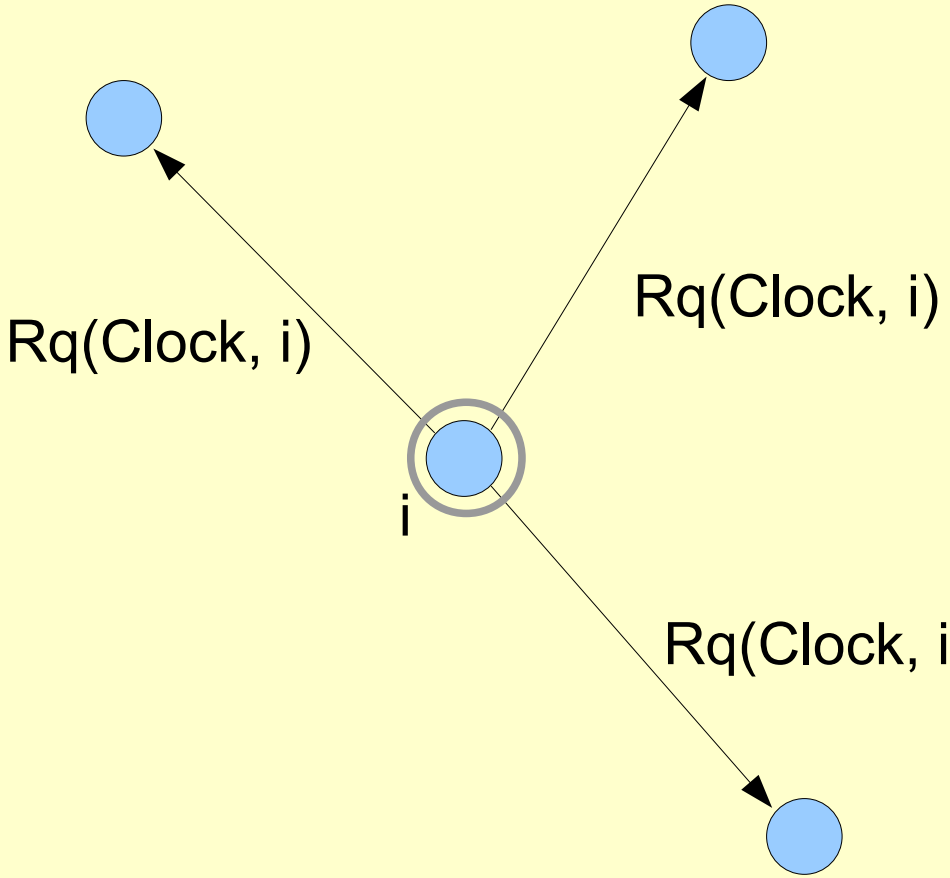
Legenda:

- Clock – sekvenční číslo
- InUse – identifikátor kritické sekce
- Req – pole registrací žádostí na vstup do CS
- Token – pole s časy přijetí tokenu
- TokenHeld – identifikátor přítomnosti tokenu

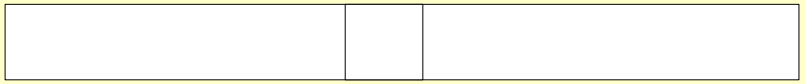
Komunikační primitiva:

- Request(Clock, i)
- Token()

Ricart - Agrawala (token passing)

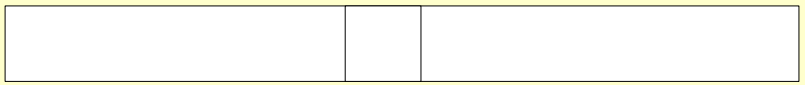


Req : array;



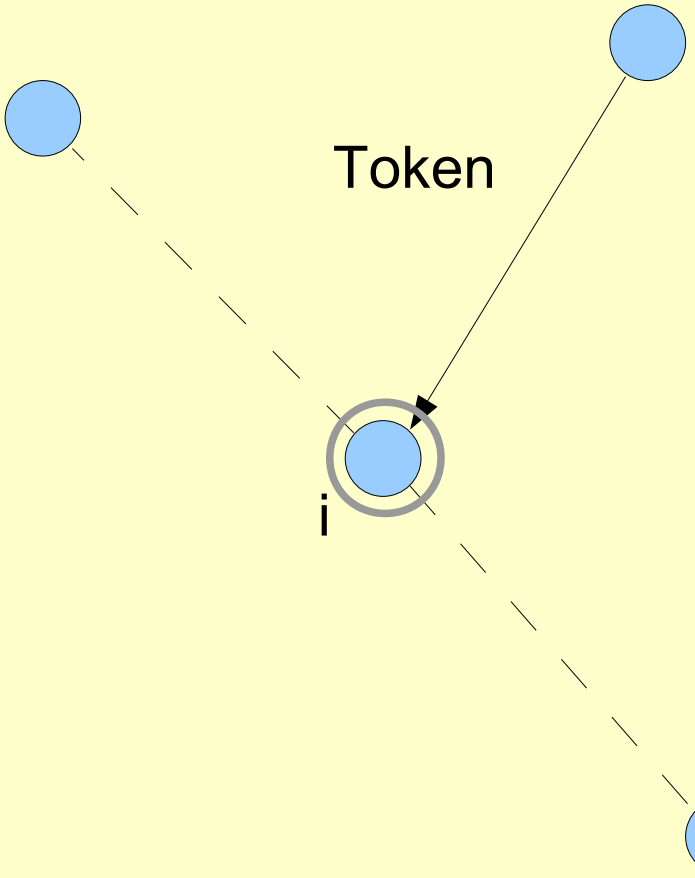
[i]

Token : array;

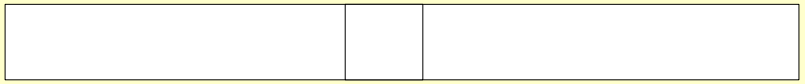


[i]

Ricart - Agrawala (token passing)

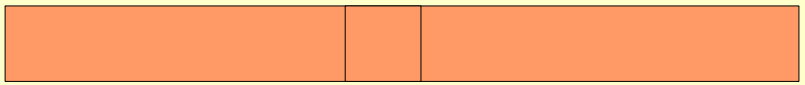


Req : array;



[i]

Token : array;



[i]

Ricart - Agrawala (token passing)

```
when request do                                          { access request }
    if not TokenHeld then
        begin
            Clock := Clock+1;
            broadcast REQUEST(Clock,i);                  { broadcasting request }
            receive TOKEN;                               { waiting for token }
            TokenHeld := T
        end;
    InUse := true;
    { critical section }
    Token[i] := Clock;
    InUse := F;
    j := (i+1) mod N;
    while i≠j do
        begin
            if Req[j]>Token[j] and TokenHeld then        { passing token }
                    TokenHeld := F;  send TOKEN to j;
            j := (j+1) mod N
        end
```

Ricart - Agrawala (token passing)

```
when received REQUEST(k,j) do                          { j-th process request }
    begin
        Req[j]:=max(Req[j],k);
        if TokenHeld and not InUse then
            begin
                j:=(i+1) mod N;
                while i<>j do
                    begin
                        if Req[j]>Token[j] and TokenHeld then
                                TokenHeld:=F; send TOKEN to j;
                        j:=(j+1) mod N;
                    end                                { passing token }
            end
    end
```

Výběr – leader election

Rozbití symetrie

Výběr na stromu

- základní algoritmus,
- vyžaduje 3(n-1) zpráv

Výběr na kruhu

Chang – Roberts

- jednosměrná komunikace,
- komunikační složitost n.log(n) ‹ n^2

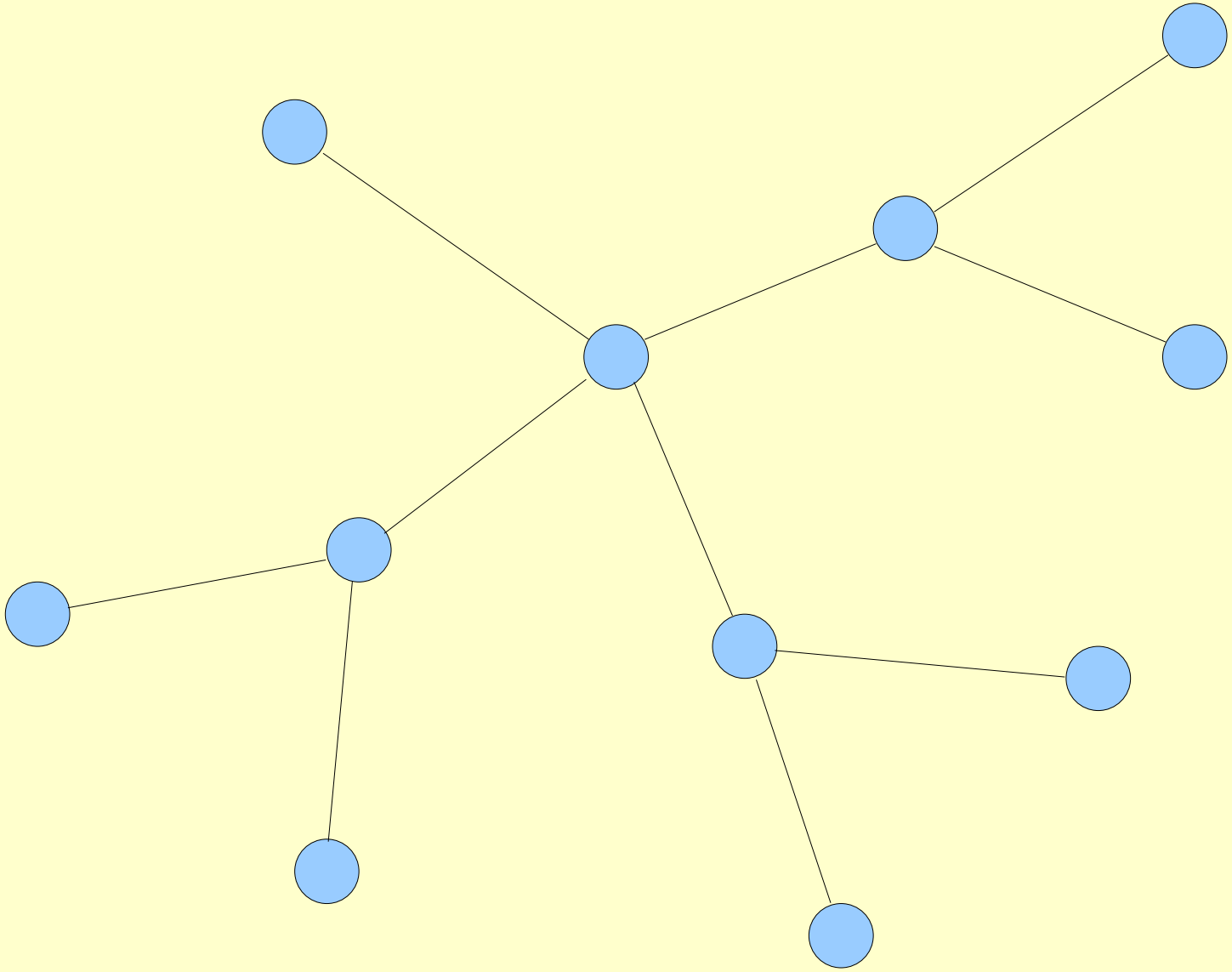
Hirschberg – Sinclar

- obousměrná komunikace
- komunikační složitost n.log(n)

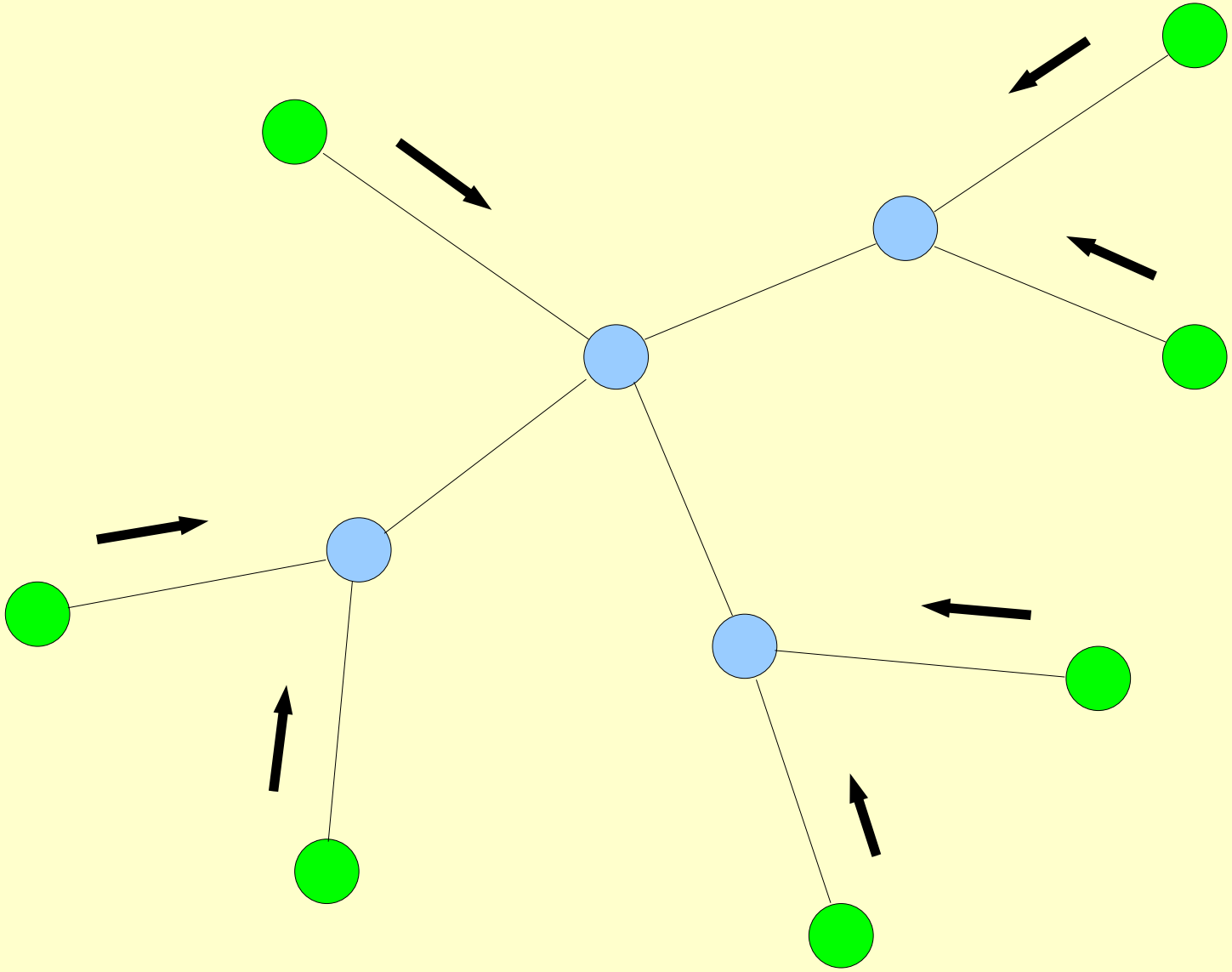
Peterson/Dolev – Klave – Rodeh

- jednosměrná komunikace,
- komunikační složitost n.log(n)

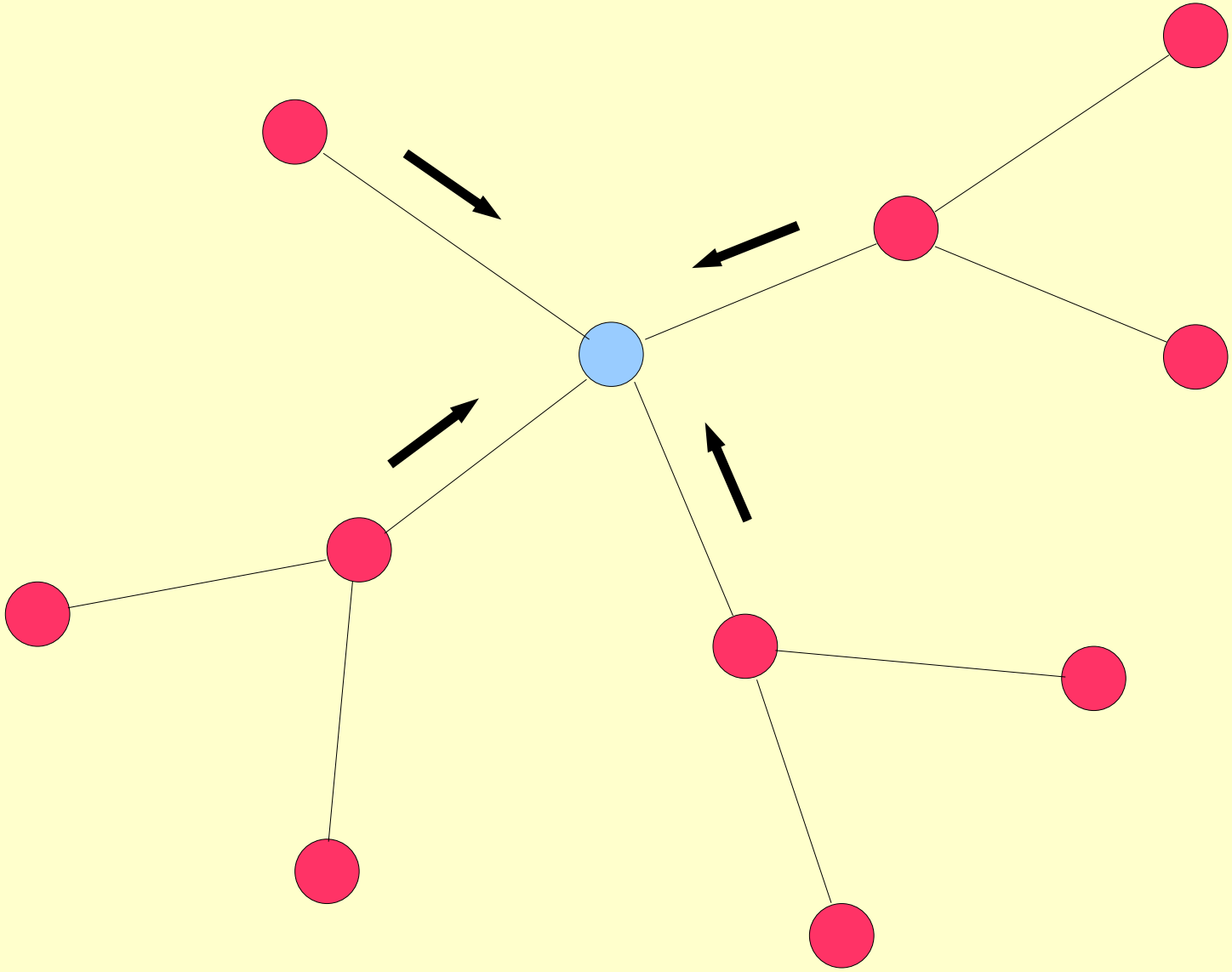
Election on tree



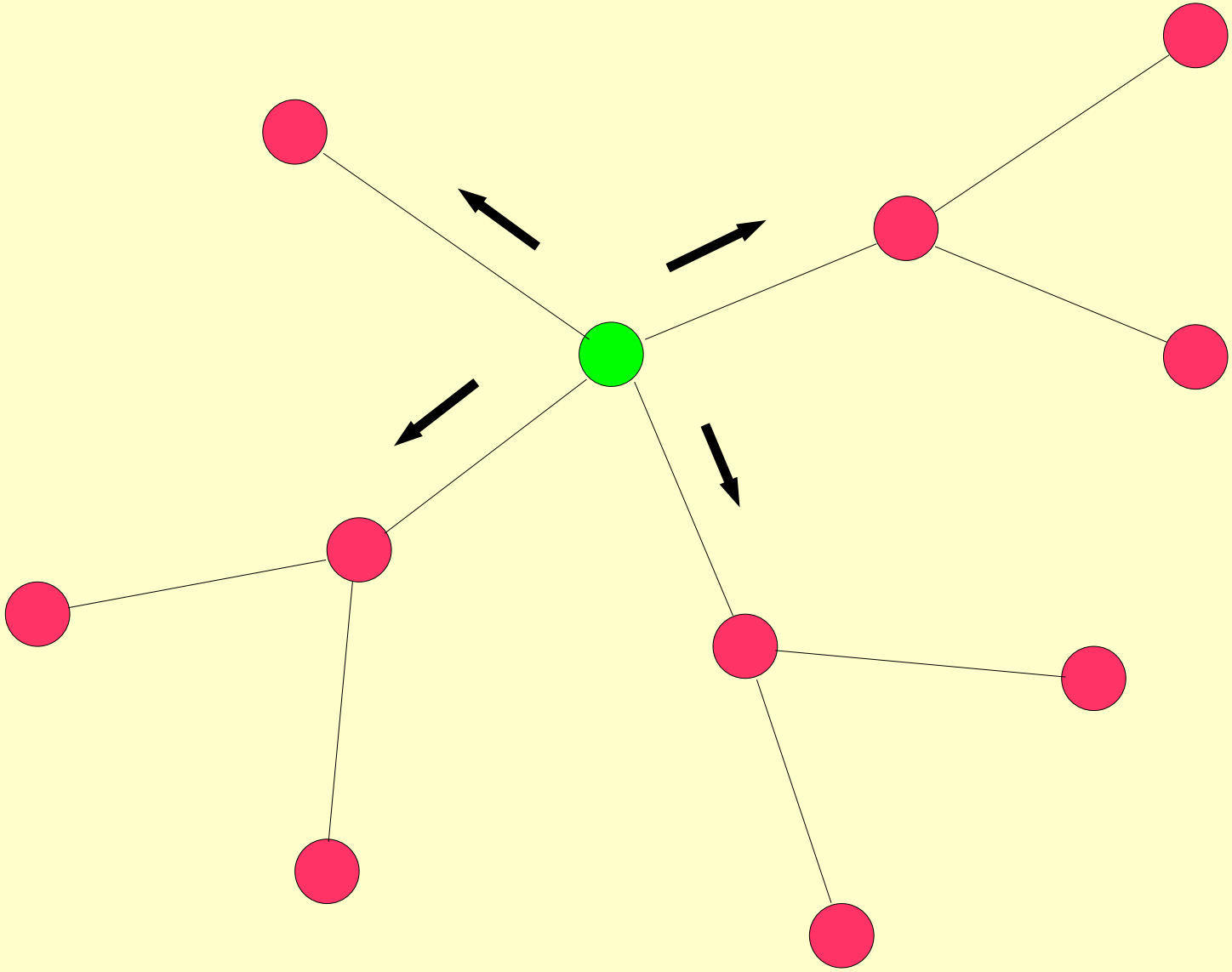
Election on tree



Election on tree



Election on tree



Election on tree

```
var   ws_p :     boolean;              init false;        // wake-up sent
      wr_p :     boolean;              init 0;            // wake-up received
      rec_p[q]:  boolean;              init false;        // received from q
      v_p :       P;                   init p;            // node id
      state_p :  {sleep,leader,lost};  init sleep;        // node state

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;
```

Election on tree

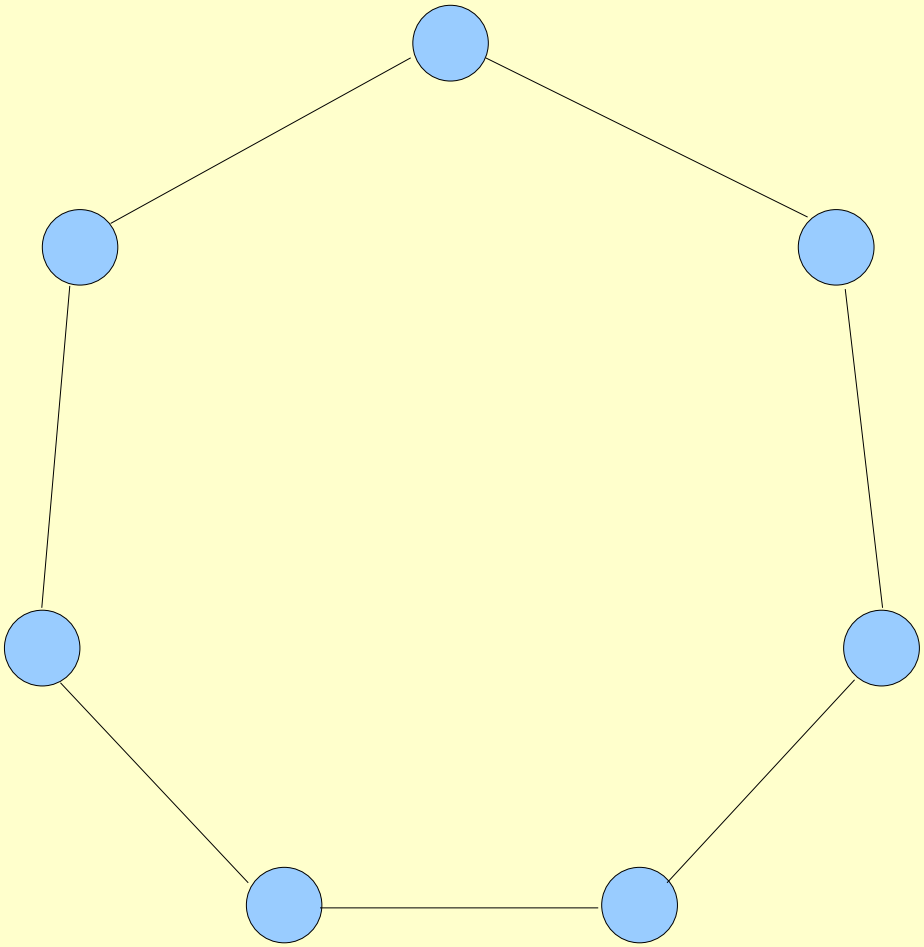
/* start of the tree algorithm */
while #{q : ~rec_p[q]} > 1 **do**
begin receive(tok,r) **from** q; rec_p[q] := true;
v_p := min(v_p,r)
end;
send(tok,v_p) **to** q_0 **with** ~rec_p[q_0];
receive(tok,r) **from** q_0;
v_p := min(v_p,r);
if v_p=p **then** state := leader **else** state := lost;
forall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

Zprávy

4x (n-1) … -?-> … 3x (n-1)

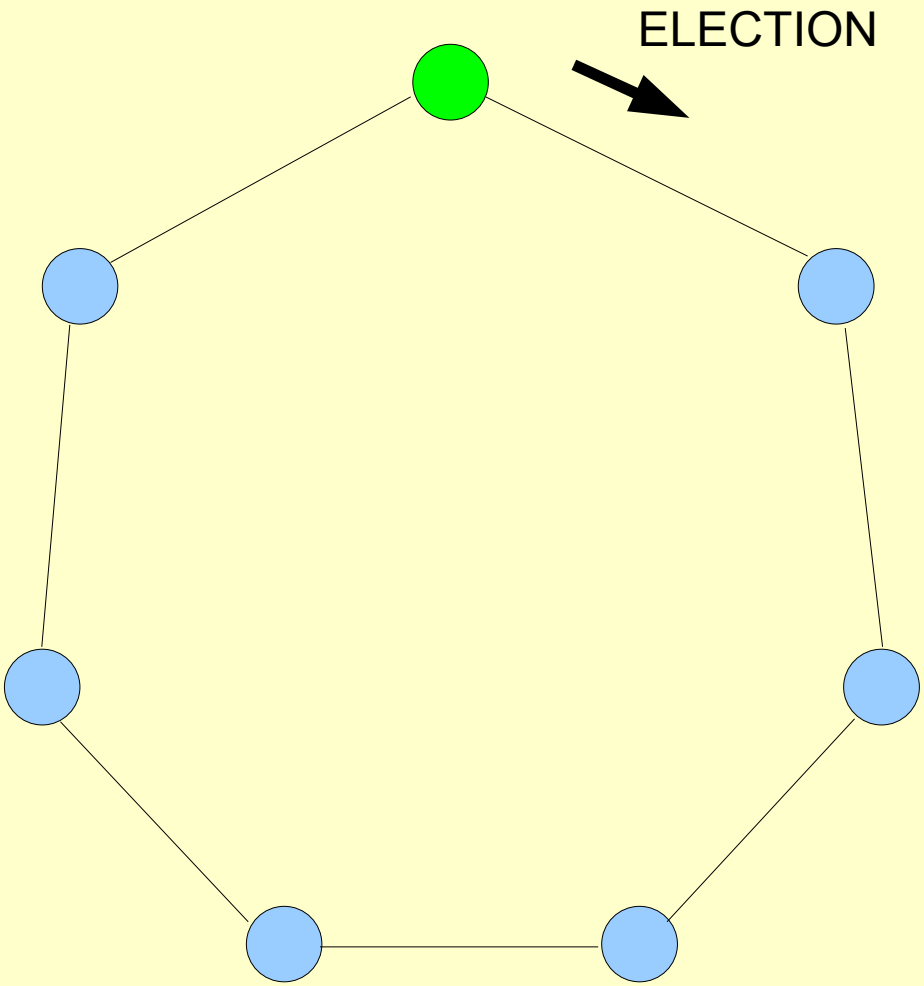
Election on ring

Chang - Roberts



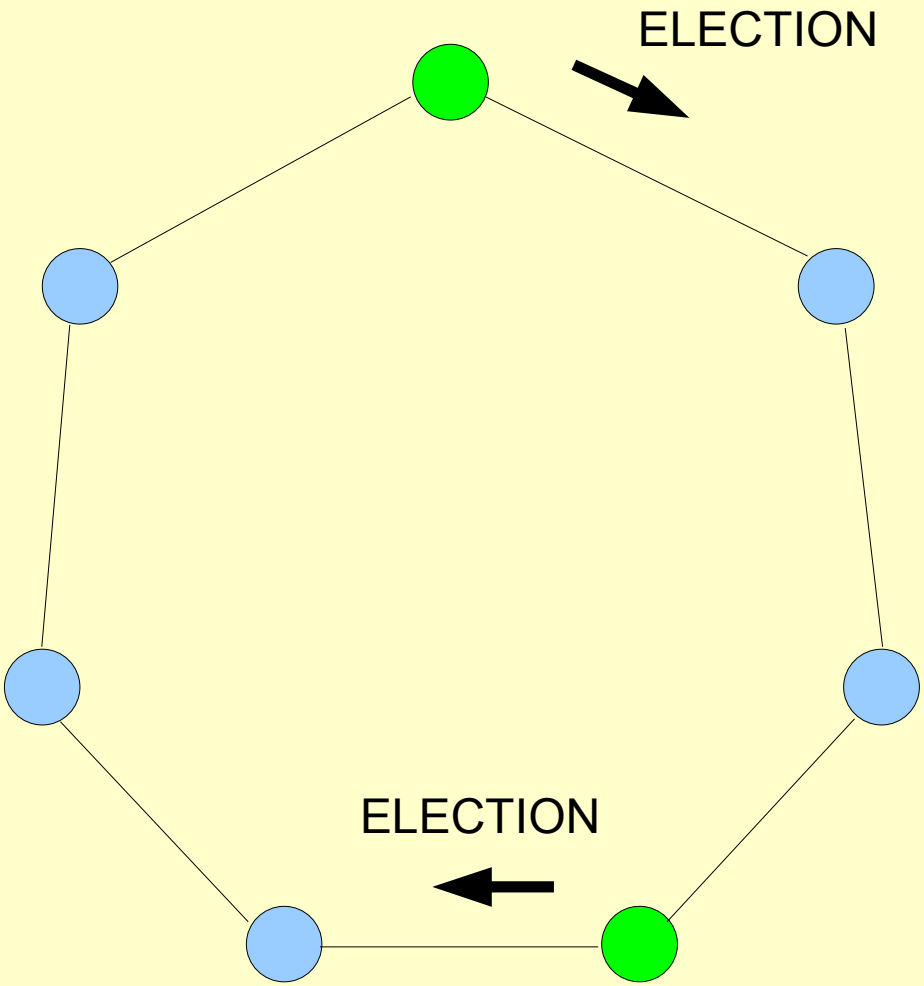
Election on ring

Chang - Roberts



Election on ring

Chang - Roberts



Election on ring

Chang - Roberts

```
var    Voting;
       Coordinator;


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

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

Election on ring

```
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
```

Election on ring

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

Zprávy

(n-1) – min

0.5n(n-1) – max

O(n log n) - avg

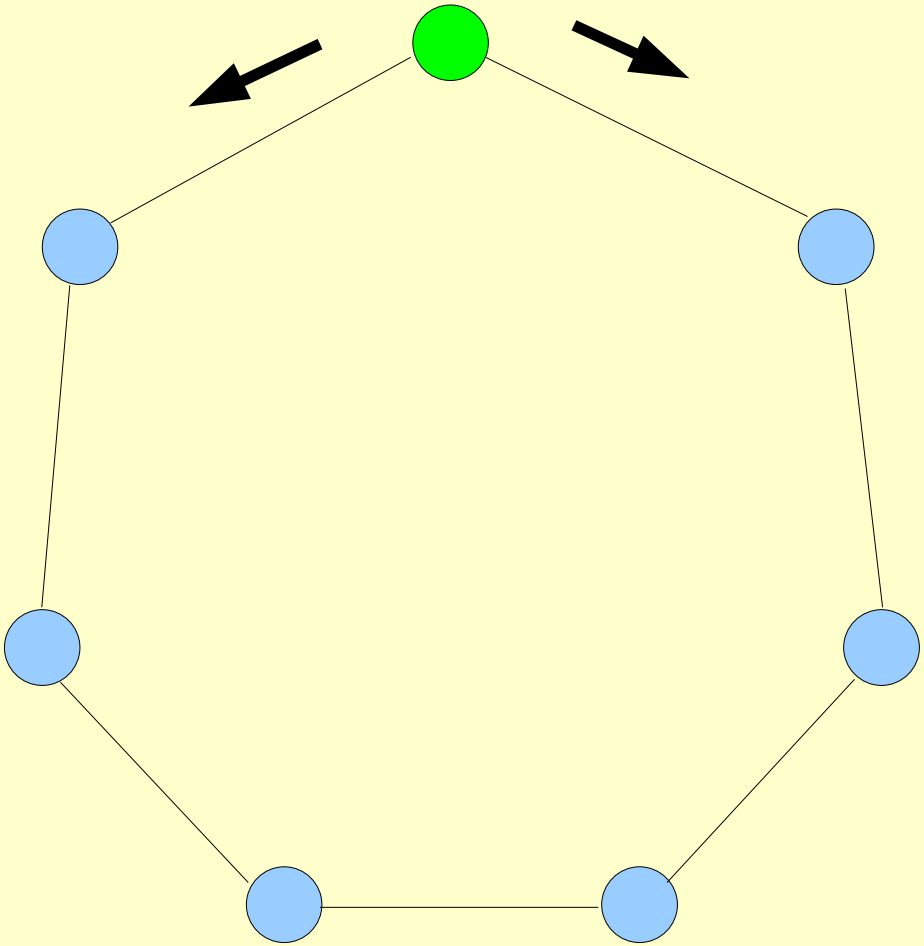
Election on ring

Chang - Roberts

```
var   state_p;
begin
      if p is initiator then
          begin
              state_p := cand; send(tok,p) to Next_p;
              repeat 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
              until state_p=leader
          end
      else
          repeat receive(tok,q); send(tok,q) to Next_p;
              if state_p=sleep then state_p := lost
          until false
```

Election on ring

Hirshberg - Sinclair



Election on ring

Hirshberg - Sinclair

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


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);    // i – id, 0 – vzd., lmax - hloubka
                        wait NResp=2;
                        if not RespOK then State := LOST;
                        lmax := 2*lmax
                  end
      end
```

Election on ring

Hirshberg - 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
```

Election on ring

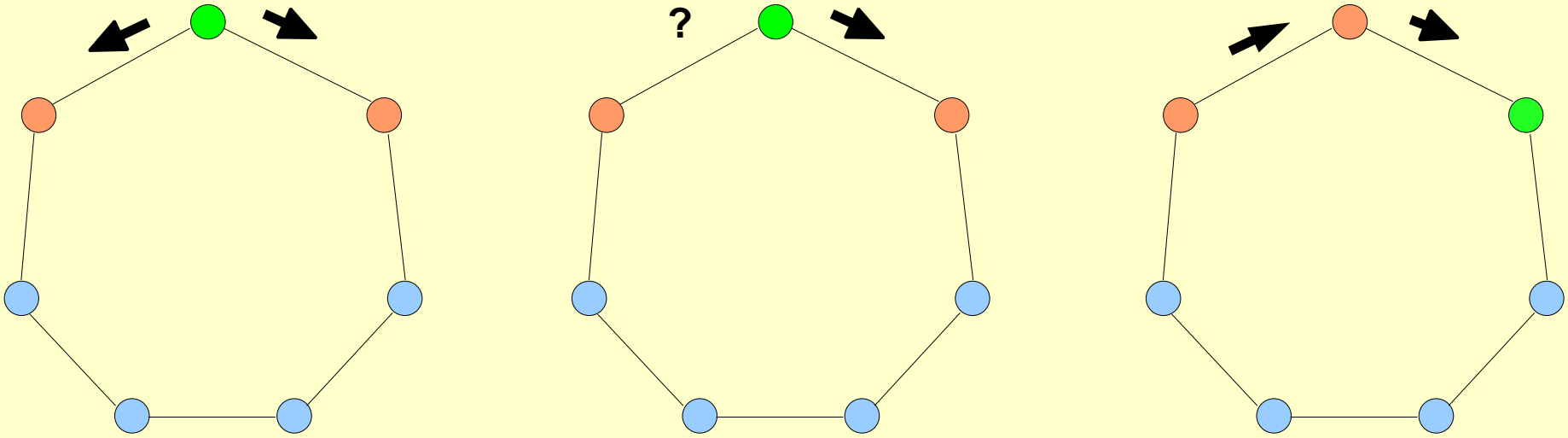
Hirshberg - Sinclair

```
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)

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

Election on ring

Peterson/DKR

var *cip* : P **init** *p* ; {Current identity of p}
acnp : P **init** *udef* ; {Id of anticlockwise active neighbor}
winp : P **init** *udef* ; {Id of winner}
statep : (*active*, *passive*, *leader*, *lost*) init *active* ;



Election on ring

Peterson/DKR

```
begin if p is initiator then state_p := active else state_p := passive ;
  while win_p = udef do begin                          { dokud neznáme vítěze }
     if state_p = active then
       begin send <one, ci_p> ; receive <one, q> ; acn_p := q ;
         if acn_p = ci_p then                          { acn_p je minimum }
           begin send <smal, acn_p> ; win_p := acn_p ; receive <small, q> ; end;
         else                                          { acn_p je současné Id souseda }
            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
     else                                              { state_p = passive }
       begin receive <one, q> ; send <one, q> ;
         receive m ; send m ;                  { m je <two, q> nebo <smal, q> }
         if m is a <small, q> message then win_p := q
       end
   end
  if p = win_p then state_p := leader else state_p := lost
end
```