Distributed Algorithms

Election Algorithms

Univ.-Prof. Dr.-Ing. habil. Gero Mühl

Architecture of Application Systems (AVA)
Faculty for Informatics and Electrical Engineering
University of Rostock

Overview

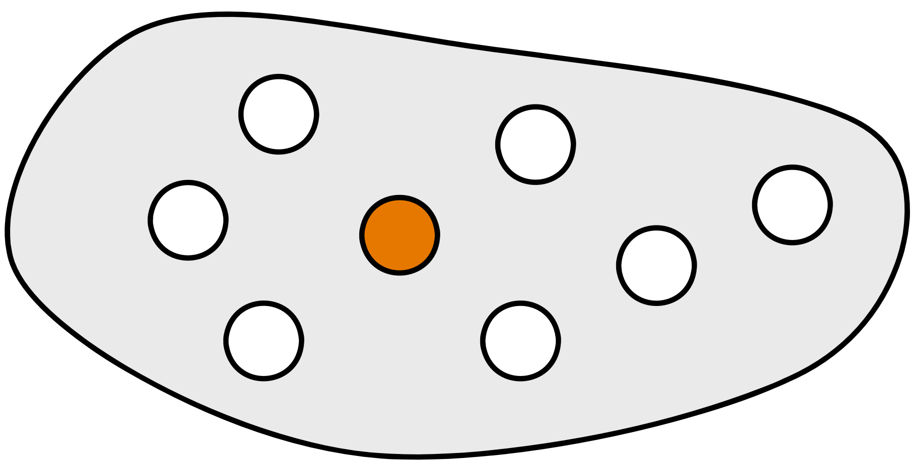
- The election problem

- Election algorithms for
  - arbitrary connected topologies
  - unidirectional and bidirectional rings
  - trees

- Randomized election algorithms for
  - bidirectional rings
  - anonymous rings

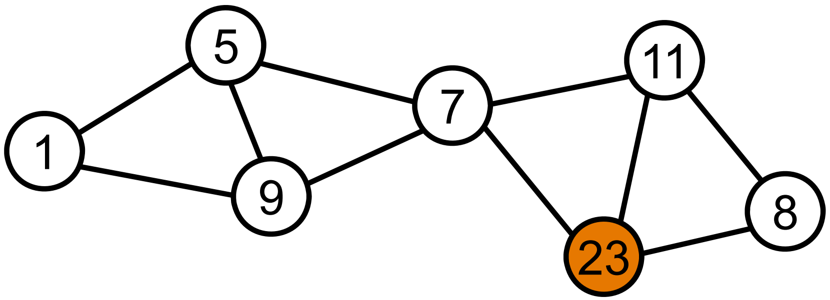
The Election Problem

- From a set of almost identical processes a *unique* leader shall be elected
- Exemplary applications
  - Determining the root node of a span tree



The Election Problem

- Assumption: Each node has a unique (integer) identity > 0
- Requirements
  - Each node shall know the elected node in the end
  - Each node may (concurrently) initiate the algorithm
- MAX algorithms
  - Determine the largest identity in the topology (or alternatively among the initiators)
  - Can be used as election algorithm



Election Algorithms – Questions

- Number of messages?
- Time complexity?
- Termination?
- Better algorithms for the same problem?
- Special topologies (rings, trees etc.)?

Universal Election Algorithms

Election for Arbitrary Topologies

```
I_p: {M_p == 0}
  M_p := p;
  SEND <M_p> TO all neighbors;

R_p: {A message <j> has arrived}
  IF M_p < j THEN
     M_p := j;
     SEND <M_p> TO
          all other neighbors;
  FI

T_p: {Termination was discovered}
  IF M_p == p THEN
      "I am the winner"
  ELSE
     "M_p is the winner"
  FI
```

I_p is executed by the initiators.

After R_p was executed, p cannot become an initiator. Thus, p cannot win and the highest initiator wins.

Each process has a unique identity p and a local variable M_p that is 0 initially.

But how?

Echo Election Algorithm

- Works with arbitrary connected topologies
- Each initiator starts an instance of the echo algorithm
- Explorer and echoes carry the identity of the respective initiator with them
- Weaker messages (i.e., explorer and echoes) are not passed on but swallowed → message extinction

Echo Election Algorithm

- Strongest wave prevails and terminates at the winner
  - Hence, the winner knows that he has won
  - If an initiator receives a stronger explorer, he knows he has lost
  - But how do the losers know of the termination and get to know the ID of the winner?
  - Winner starts echo algorithm once again to distribute a win notification containing its ID
  - Using the generated spanning tree for this purpose is possible
- Weaker waves (i.e., all other waves) finally stagnate
  - This is because at least the strongest initiator sends no echo for the weaker waves
  - No echo algorithm of any other initiator terminates

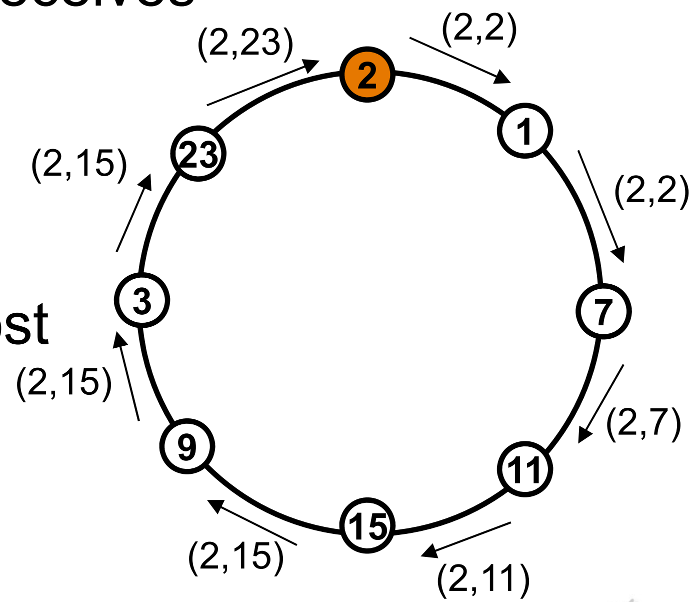
Election algorithms for unidirectional rings

Algorithm of Le Lann, 1977

- Each process wakes up, either as initiator or at latest when it receives a message from its neighbor node
- *Each* waking up starts a complete ring circulation of a message containing the process's own ID and the maximum ID encountered on the ring so far (equals its own ID initially)
- At the end of *its* circulation, each node receives a message with its own ID and the largest ID within the ring
- If those both IDs are identical, the node has won the election; otherwise it has lost
- The highest of *all* nodes wins

⇒ *n* complete circulations

⇒ n^2 messages



Algorithm of Le Lann

I_p is spontaneously executed by the concurring initiators and by the other nodes at receipt of the first message.

```
Ip: {init == FALSE}
  init := TRUE;
  SEND <p, p> TO <next node>;

Rp: {A message <i, j> has arrived}
  IF init == FALSE THEN Ip; FI
  IF i != p THEN
     k := MAX(j, p);
     SEND(i, k) TO <next node>;
  ELSE
     IF p == j THEN
        <I am the winner>;
     ELSE
       <Process j is the master>;
     FI
  FI
```

For each process initially `init == FALSE`

Algorithm of Le Lann – Variant

- Also possible: Variant that determines the highest node *among the initiators*
- With this variant, nodes not participating in the election pass on the messages unchanged

Algorithm of Chang and Roberts

- With the algorithm of Le Lann, messages that cannot lead to a win are passed on as well
- Idea of Chang and Roberts, 1979
  - Messages are only passed on if they can lead to a win; all other messages are extinguished → message extinction
  - Since only the winner receives its own message, the other nodes are informed of the win by an additional ring circulation

Algorithm of Chang and Roberts

```
I_p: {M_p == 0}
   M_p := p;
   SEND <M_p> TO next node;
```

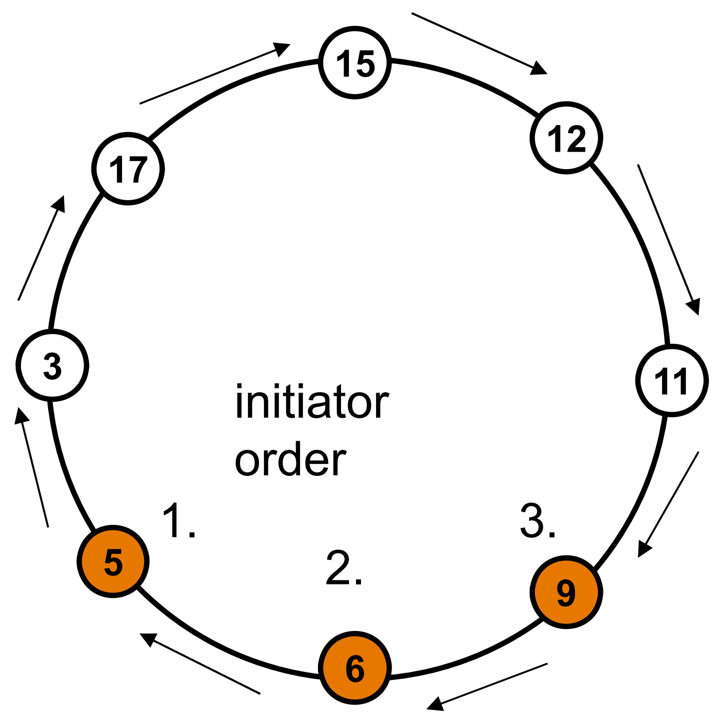
Attention: Here, only an initiator can win!

For each process, initially M_p == 0

```
R_p: {a message <j> has arrived}
   IF M_p < j THEN
     M_p := j;
     SEND <M_p> TO next node;
   FI
   IF j == p THEN
      "I am the winner"
     <inform all by another ring circuit>;
   FI
```

Worst-Case Message Complexity

- Occurs, if the *k* initiators are arranged on the ring in descending order *and* initiate election in ascending order
  - *k*-largest initiator
    → $n - (k - 1)$ messages
  - …
  - 3rd-largest initiator
    → $n - 2$ messages
  - 2nd-largest initiator
    → $n - 1$ messages
  - Largest initiator
    → n messages



Worst-Case Message Complexity

$$\sum_{i=1}^{n} i = \frac{n \cdot (n+1)}{2}$$

> Message complexity with *k* initiators

$n + (n - 1) + (n - 2) + \ldots + (n - (k - 1))$

$= (1 + 2 + \ldots + n) - (1 + 2 + \ldots + (n - k))$

$= n (n + 1) / 2 - (n - k) (n - k + 1) / 2$

$= (n^2 + n - n^2 + nk - n + nk - k^2 + k) / 2$

$= (2nk - k^2 + k) / 2$

$= \boldsymbol{n\, k - k\, (k - 1) / 2}$

⇒ $O(n^2)$ with ring size n and $k = n$

> Still *n* additional messages for the win notification

Best-Case Message Complexity

> Occurs, if the k initiators are arranged in ascending order and initiate the election approximately simultaneously

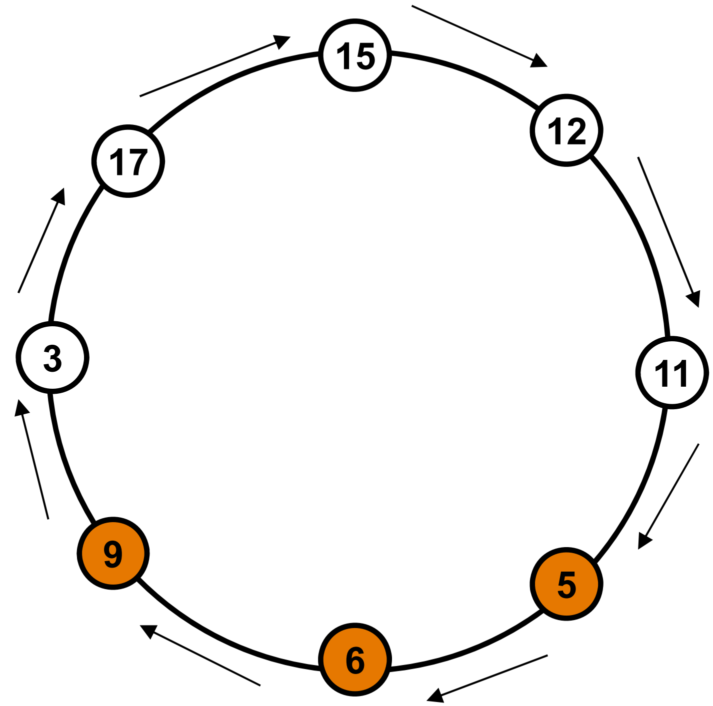
> All k, but the largest initiator cause only 1 message

> Largest initiator causes n messages

⇒ $n + k - 1$ messages for the election

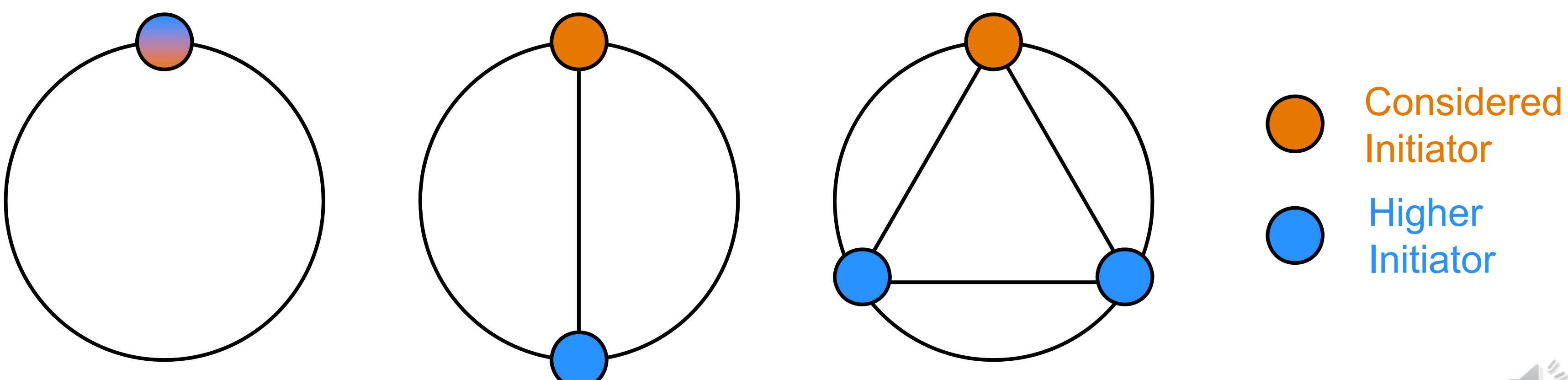
⇒ $O(n)$ with ring size n and $k = n$

> Again, n additional messages for the win notification



Average-Case Message Complexity

- Average the message complexity over all possible permutations of the IDs on the ring!
- Informal argumentation
  - largest initiator → n messages (always)
  - 2nd-largest initiator → $n / 2$ messages on average
  - 3rd-largest initiator → $n / 3$ messages on average
  - …
  - k-largest initiator → n / k messages on average



Average-Case Message Complexity

- The average-case message complexity is $n\, H_k \approx n \ln k$ with $H_k = 1 + 1 / 2 + ... + 1 / k$
  - This is optimal for unidirectional rings
  - H_k is the k-th Harmonic Number → Harmonic Series
- O(n log n) with ring size n and $k = n$
- For very large rings, almost never more messages are required than on average (cf. Rotem et al.)
- Again, n additional messages for the win notification

- Remark: It was assumed implicitly that no message overtakes can take place on a link
- How do overtakes influence the message complexity?

Influence of Message Overtaking

> A message can only be overtaken by higher messages because a message is only passed on by a node if it is higher than the former highest message sent

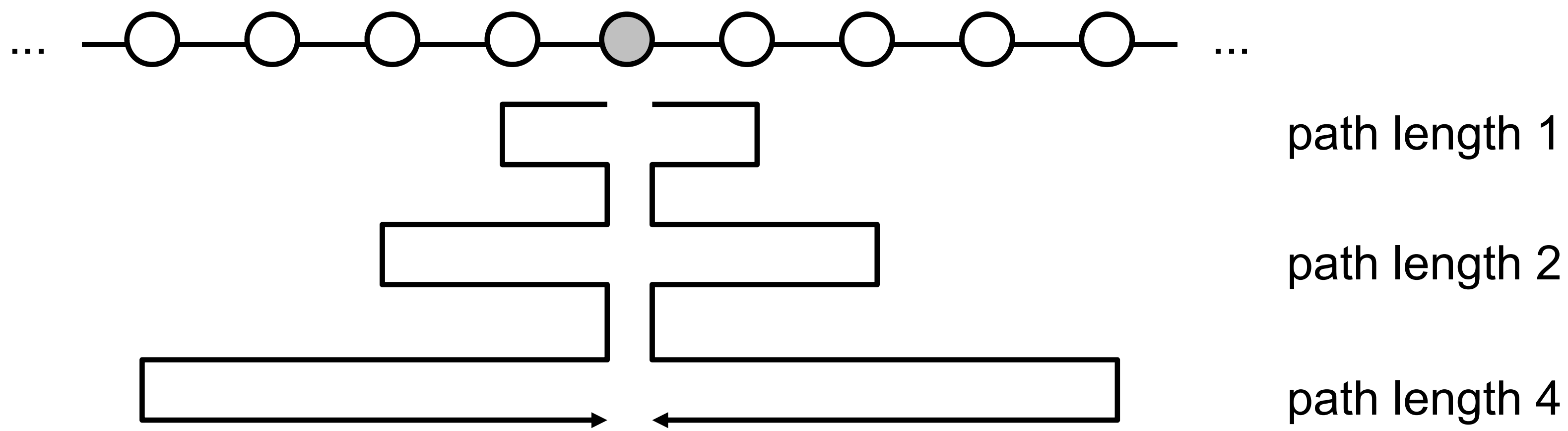
> Through the overtake, the lower message is extinct potentially *earlier*; in this case messages are saved

> No messages are saved if the receiving node is a higher initiator since in this case, the message would have been extinct anyway

Election Algorithms for Bidirectional Rings

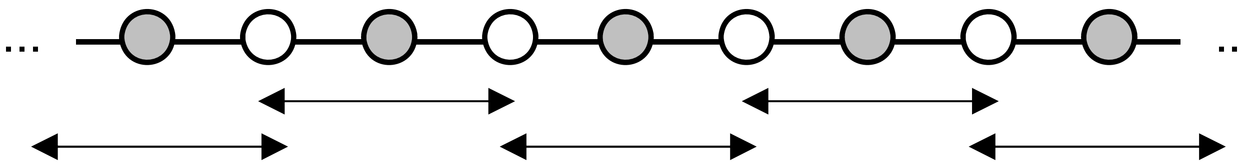
Hirschberg-Sinclair Election Algorithm

- Algorithm proceeds in phases in which each node tries to conquer successive ring areas whose path length is doubled starting from 1 with every new phase
- If a higher node is encountered on the way (in either direction), this node vetoes and the original node becomes passive
- Otherwise, it stays active and starts the next phase
- If the area conquered reaches the node again, it has won



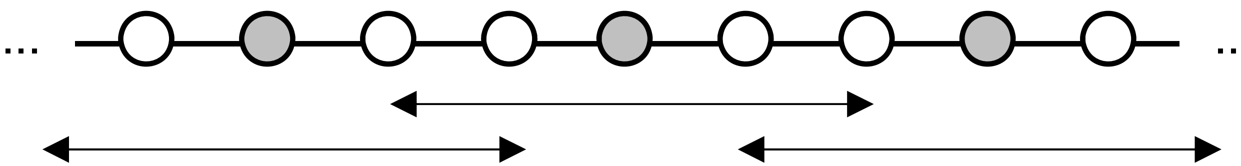
Worst-Case Message Complexity

> **1** process in between after phase **1**



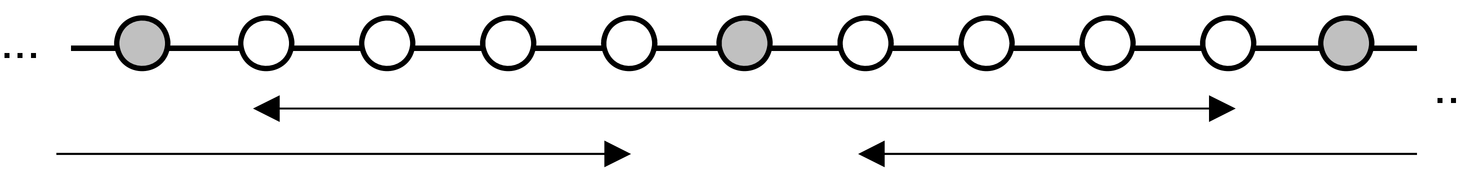
max. n / 2 survive

> **2** processes in between after phase **2**



max. n / 3 survive

> **4** processes in between after phase **3**



max. n / 5 survive

⇒ $\mathbf{2^{\mathit{i}-1}}$ processes in between after phase $\boldsymbol{i}$

max. $n / (1+2^{i-1})$ survive

Worst-Case Message Complexity

- Phase 1: n processes can initiate paths of length 1 with at most $4 \cdot 1$ messages
- Phase 2: $n / 2$ processes can initiate paths of length 2 with at most $4 \cdot 2$ messages
- Phase 3: $n / 3$ processes can initiate paths of length 4 with at most $4 \cdot 4$ messages
- Phase 4: $n / 5$ processes can initiate paths of length 8 with at most $4 \cdot 8$ messages
- …
- Phase i: $n / (1 + 2^{i-2})$ processes can initiate paths of length 2^{i-1} with at most $4 \cdot 2^{i-1}$ messages

⇒ Each phase causes $4 \cdot 2^{i-1} \cdot n / (1 + 2^{i-2}) < \mathbf{8}\boldsymbol{n}$ messages

Worst-Case Message Complexity

- There are at most $1+ \lceil \log_2 n \rceil$ phases
- Upper bound is, thus, $8n + 8n \lceil \log_2 n \rceil)$ messages
- ⇒ Worst-case message complexity is $O(n \log n)$
- Again n messages extra for win notification

- Worst-case unit time complexity is $4n - 2$ for $n = 2^k$ (best-case) and $6n - 6$ for $n = 2^k + 1$ (worst-case) → $O(n)$

Considering the Phase with the Extinction

- The initiators do not have to proceed through the phases synchronously
- An initiator that is already in a high phase may still be stopped by a new but higher initiator
- To avoid that, pairs (<*phase*>, <*node identity*>) are used and ordered lexicographically rather than by node identities only
- Then, an initiator in a higher phase always prevails against an initiator in a lower phase; only if the phase is the same, the node identity is used as a tie breaker
- The algorithm is no longer a MAX-Algorithm, because it does not necessarily determine the node (or initiator) with the highest ID as the winner

⇒ Not every election algorithm has to be a MAX-Algorithm

Peterson Election Algorithm

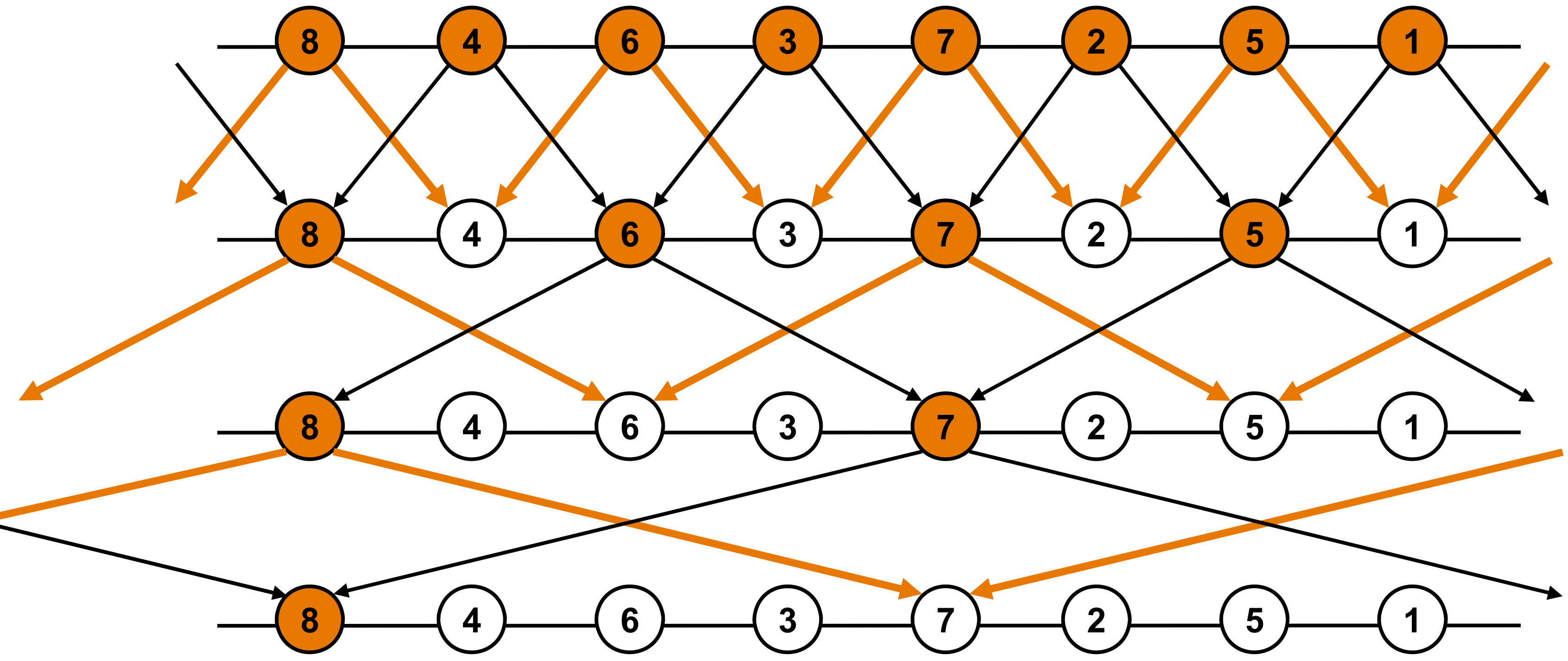
- In the beginning, all processes are active, by and by all processes but one become passive
- Algorithm proceeds in phases, but does *not* use vetoes
- Course of a phase
  - Each active process tells its ID to the next active process in both directions
  - Processes receiving a lower ID from *both* directions remain active and participate in the next phase
  - The other processes become passive and only relay messages
- A node wins if it receives its own ID

- Additionally: A circulation for the win notification of all nodes with n messages, again ignored in the following calculations

Peterson Election Algorithm

```
do forever begin
    sendToBothSides(id);
    p = receiveFromLeft();  // ID of active predecessor
    if p == id then goto leader;
    s = receiveFromRight(); // ID from active successor
    if s == id then goto leader;
    if id < max(p, s) then goto relay;
end

relay:
do forever begin
    x = receiveFromEitherSide();
    sendToOtherSide(x);
end

leader:
"announce elected";
```

Worst-Case Message Complexity



Worst-Case Message Complexity

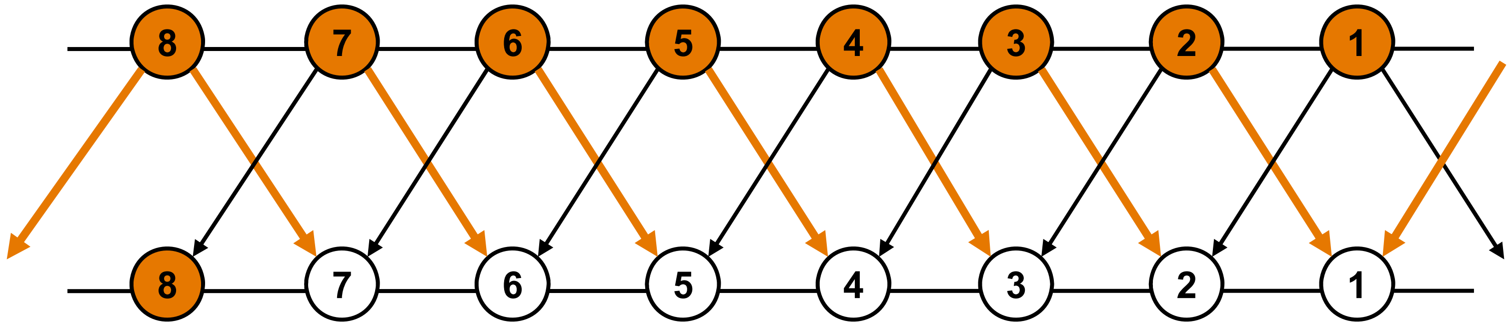
- In the beginning, all n processes are active
- In each phase, from m active processes, at most $\lfloor m / 2 \rfloor$ processes can survive
- Thus, at most $\lfloor ld\ n \rfloor$ phases are needed to reach a single process that starts one additional phase
  → at most $\lfloor ld\ n \rfloor + 1$ phases
- In each phase, every node sends and receives 2 messages
  → $2n$ messages are sent per phase

⇒ Worst-case message complexity $2n\,(\lfloor ld\ n \rfloor + 1)$ → $O(n \log n)$

Best-Case Message Complexity

- Best arrangement sorts the nodes
- Each node except the largest is extinguished by one of its neighbors in the first phase
- Only the largest nodes participates in the 2nd phase
- Termination after two phases with $2n$ messages each

⇒ Best-case message complexity is $4n$, i.e., $O(n)$



Average-Case Message Complexity

ID	1st not higher	2nd not higher	both not higher
1	0	0	0
2	1/(*n*-1)	0	0
3	2/(*n*-1)	1/(*n*-2)	2/((*n*-1)(*n*-2)
4	3/(*n*-1)	2/(*n*-2)	6/((*n*-1)(*n*-2)
…	…	…	…
n-2	(*n*-3)/(*n*-1)	(*n*-4)/(*n*-2)	(*n*-4)(*n*-3)/((*n*-1)(*n*-2)
n-1	(*n*-2)/(*n*-1)	(*n*-3)/(*n*-2)	(*n*-3)(*n*-2)/((*n*-1)(*n*-2)
n	(*n*-1)/(*n*-1)	(*n*-2)/(*n*-2)	1

To derive the expected value for the survival of a node the sum of those values is divided by *n*(*n*-1)(*n*-2)

$$\sum_{i=1}^{n}(i-1)(i-2)=\sum_{i=1}^{n}(i^2-3i+2)=\frac{n(n+1)(2n+1)}{6}-3\frac{n(n+1)}{2}+2n$$

$$=\frac{n(n+1)(2n+1)}{6}-\frac{9n(n+1)}{6}+\frac{12n}{6}=\frac{n(n+1)(n-4)+6n}{3}=\frac{n(n^2-3n+2)}{3}=\frac{n(n-1)(n-2)}{3}$$

⇒ A node survives a phase with the probability 1/3

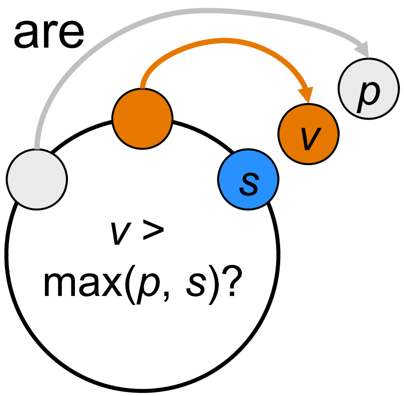
Average-Case Message Complexity

> Since, an active node survives a phase with probability 1/3, we have $\log_3 n$ phases on average
> Again $2n$ messages per phase

⇒ Average-Case is again $O(n \log n)$

Unidirectional Variant

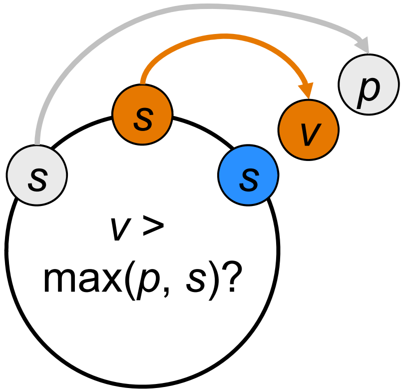
- With the bidirectional variant, an active node compares its value with the value of the next active predecessor and with the value of the next active successor to decide whether it remains active or not
- But: On unidirectional rings messages can be only sent forward
- Solution
  - The IDs of the active predecessor and that of the node are transmitted to the active successor and stored in the variables *v* and *p*, respectively
  - The comparison of these values with the own ID *s* is carried out by the successor
  - If ***v* > max(*p*, *s*)** applies, it remains active and takes part in the next phase with the ID of its predecessor, i.e., *v*
- If this solution is implemented in a clever way (see next slide), the number of messages sent per phase and the number of phases does not change compared to the bidirectional variant



Unidirectional Variant

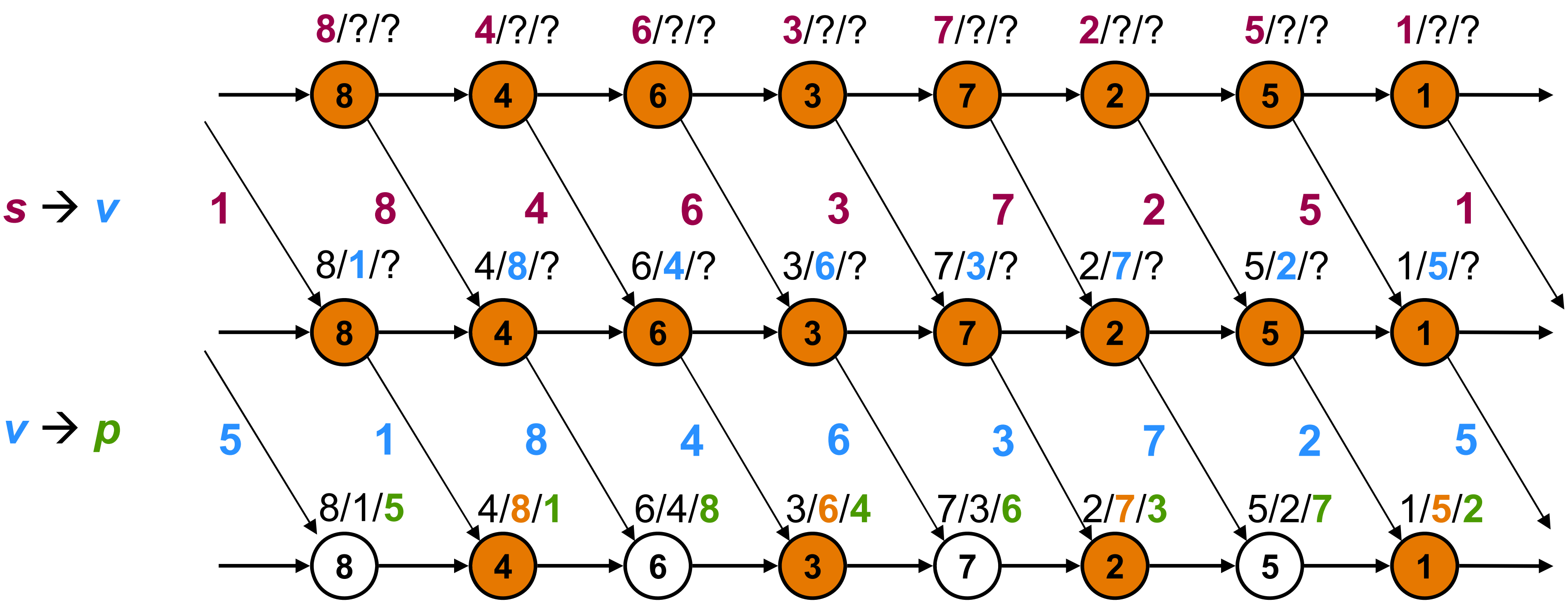
```
s := id;
do forever begin
   send(s);
   v = receive();
   if v == id then goto leader;
   if s > v send(s) else send(v);
   p = receive();
   if p == id then goto leader;
   if v > max(p, s) then s := v else goto relay;
end

relay:
do forever begin
   s = receive();
   if s == id then goto leader;
   send(s);
end

leader:
"announce elected";
```

A node receives:
The *s* of its predecessor in *v*
The *s* of its prepredecessor in *p*



Messages must not overtake each other or have to be marked to assure a correct assignment

Example for Unidirectional Variant

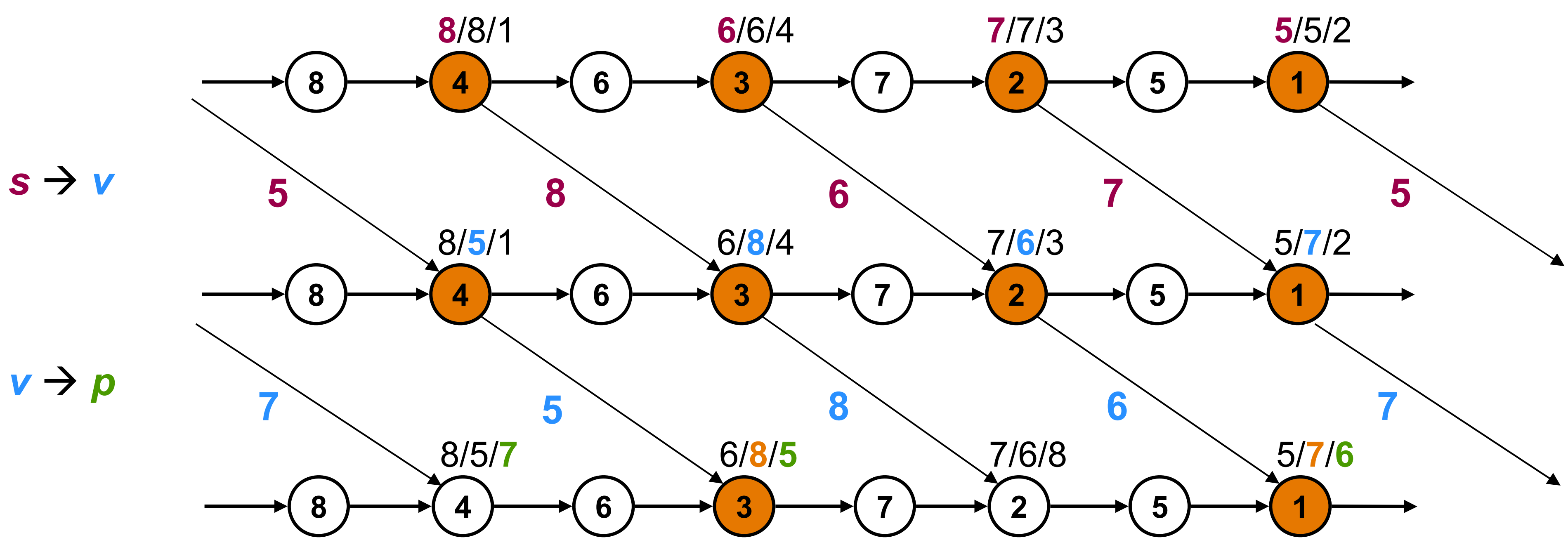


Phase 1

s / *v* / *p*

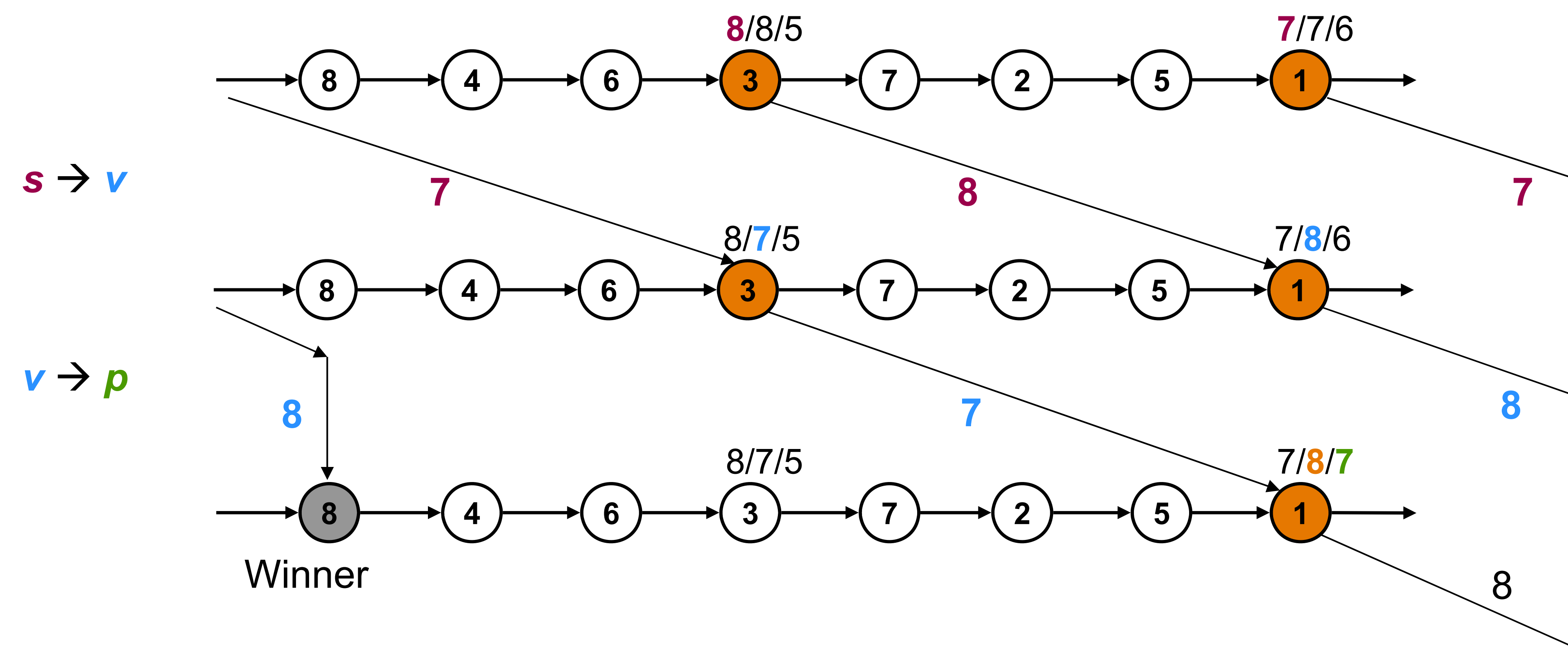
$v > \max(p, s) \rightarrow s := v$

Example for Unidirectional Variant



Phase 2 s / v / p *v* > max(*p*, *s*) → *s* := *v*

Example for Unidirectional Variant

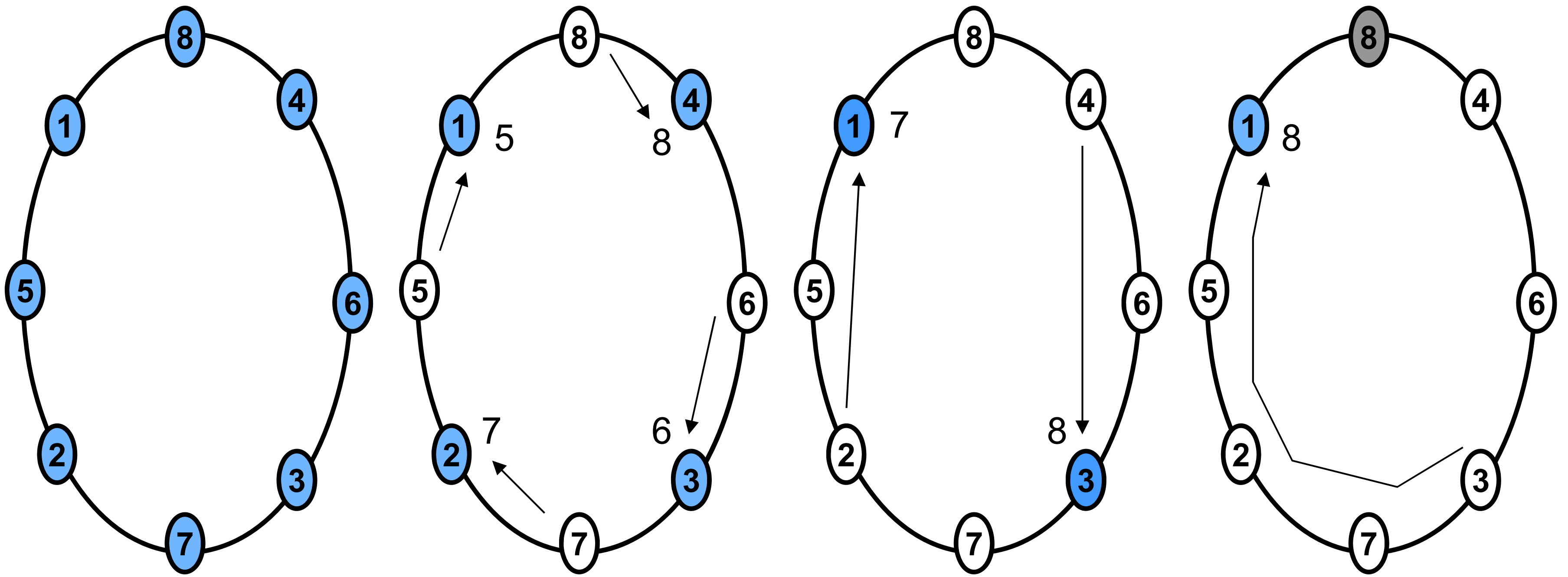


Phase 3

s / v / p

v* > max(*p*, *s*) → *s* := *v

Example for Unidirectional Variant



Thank you for your kind attention!

Univ.-Prof. Dr.-Ing. habil. Gero Mühl

`gero.muehl@uni-rostock.de`
`http://wwwava.informatik.uni-rostock.de`