Algorithm 4.20 *Peterson's election algorithm in a unidirectional ring* [40].

Idea: Algorithm 4.16 is simulated in a unidirectional ring. Every process first sends its id to its right neighbor, and subsequently sends the maximum of its own id and the value received from its left neighbor to its right neighbor. If among the three values a process now possesses, the first one that was received is at least as large as the other two, the process remains active, otherwise it becomes a relay process. This same procedure is then repeated over and over again in the virtual ring only consisting of the active processes, with the relay processes only acting as transmitters of messages. A process is elected when it receives its own id.

Implementation:

I. Active processes

```
tid ← id
do forever
    send(tid); receive(ntid)
    if (ntid=id) then elected ← true
    send(max(tid,ntid)); receive(nntid)
    if (nntid=id) then elected ← true
    if ((ntid>=tid) and (ntid>=nntid)) then
        tid ← ntid
    else goto relay
```

II. Relay processes

```
relay:
do forever
    receive(tid)
    if (tid=id) then elected ← true
    send(tid)
```

Correctness: We call one execution of the loop in code fragment I of Algorithm 4.20 in some process a round. (But note that the algorithm is asynchronous, even though some form of synchrony is enforced by the message pattern.) Let the id of process P_i be denoted by $\texttt{id}_i$, $i = 0, 1, \ldots, n \quad 1$, and let $\texttt{id}_m$ be their maximum. We say that an id survives round k if it is equal to the `tid` of some active process at the start of round $k+1$. Clearly, $\texttt{id}_m$ always survives, and as it continues to make progress around the ring in each round, it will eventually return to P_m, which then concludes it has been elected. So the only potential problem is that another process thinks it has been elected. It is easy to see that if process P_i has $\texttt{tid} = \texttt{id}_j$ at the start of some round, then all of the processes $j, j+1, \ldots, i-1$ (we take the process numbers modulo n) are relay processes in that round. As long as $\texttt{id}_l$ with $l \neq m$ survives, it will at the start of successive rounds be equal to the `tid` of processes that are ever closer to P_m, until at some point there is no active process left before P_m. But then $\texttt{id}_l$ and $\texttt{id}_m$ are in neighboring active processes, with $\texttt{id}_m$ in an active process between P_m and P_l, and so $\texttt{id}_l$ will not survive the next round.

Complexity: The number of rounds is at most equal to $\log n$, because in every round, the number of active processes is at least cut in half. Because in every round exactly two messages are sent along every link, the number of messages is at most equal to $2n \log n$. In [40], it is proven that the message delay, defined as the longest chain of messages in the algorithm, is at most equal to $2n \quad 1$. □

Example 4.21 In Figure 4.3, a part of a unidirectional ring with three processes is shown. In the first part of the first round, process P_3 sends its id of 3 to P_2, and P_2 sends its own id 7 to P_1. In the second part of the first round, P_2 sends the maximum of its own id (7) and the value it received in the first round (3), so a 7, to P_1. Process P_1 now has three values 4,7,7, and as the first received is at least as large as the other two, it remains active with that value (7). □

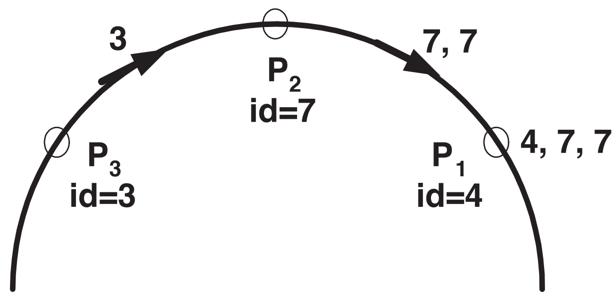

Figure 4.3: An example of the execution of Algorithm 4.20.