Algorithm 3.15 *The Birman-Schiper-Stephenson algorithm for causal message ordering of broadcast messages* [5].

Idea: This algorithm uses vector logical clocks implemented by the variable V in every process, initialized to all zeroes, for the sole purpose of causal message ordering, but not for affixing timestamps to other events in the system. The links do not have to be FIFO. Every process numbers its broadcasts consecutively in its own component of its vector logical clock and sends this vector along with the message to all other processes. If on receipt of a message a process finds that it cannot yet be delivered, it puts it into a buffer B of pending messages (along with their timestamps). We define the following condition for a message $\mathtt{m}$ that carries timestamp $V_{\mathtt{m}}$ and is received from process P_j to indicate whether it can be delivered:

$$D_j(\mathtt{m}) \equiv (V + e_j \geq V_{\mathtt{m}}).$$

Condition $D_j(\mathtt{m})$ says that message $\mathtt{m}$ is the message expected next from P_j, and that the receiving process is at least as up to date with respect to all other processes as P_j was when it sent $\mathtt{m}$.

Implementation:

I. Broadcasting a message
$V \leftarrow V + e_i$
broadcast(m,V)

III. Delivering a message from P_j
`deliver(m)`:
`deliver(m)`
$V \leftarrow V + e_j$
remove $(\mathtt{m}, V_{\mathtt{m}})$ from B

II. Receiving a message from P_j
upon receipt of $(\mathtt{m}, V_{\mathtt{m}})$ **do**
  if $D_j(\mathtt{m})$ **then**
    `deliver(m)`
    while $(\{(\mathtt{m}, \mathtt{k}, V_{\mathtt{m}}) \in B\ D_k(\mathtt{m})\} \neq \emptyset)$ **do**
      `deliver(m)` with $(\mathtt{m}, V_{\mathtt{m}}) \in B$ such that $D_j(\mathtt{m})$
  else add $(\mathtt{m}, \mathtt{j}, V_{\mathtt{m}})$ to B

Correctness: First we prove the safety of the algorithm, that is, we prove that if $m(\mathtt{m}_1) \rightarrow m(\mathtt{m}_2)$, then $d_i(\mathtt{m}_1) \rightarrow d_i(\mathtt{m}_2)$ for $i = 1, \ldots, n$. Because predicate D_j already checks the order of the broadcasts from the same process, we can assume that $\mathtt{m}_1$ is broadcast by process P_j and $\mathtt{m}_2$ is broadcast by process P_k, with $j \neq k$. Because $m(\mathtt{m}_1) \rightarrow m(\mathtt{m}_2)$, $V(\mathtt{m}_1)[j] \leq V(\mathtt{m}_2)[j]$. By code fragment III, the only modification made to the vector logical time in the receiving process when a message from process P_j is delivered is adding e_j to its vector clock. Now $\mathtt{m}_2$ can only be delivered in P_i when $V_i[j] \geq V(\mathtt{m}_2)[j]$, but $V_i[j]$ can only attain this value after $\mathtt{m}_1$ has been delivered (V_i is here the vector clock in the receiving process P_i).

The proof of the liveness of the algorithm is left as an exercise for the reader. □