$$q[j] = (\text{Release}, 0, j) \quad j = 1, \ldots, N$$

Three types of messages are used in this algorithm:

- (Request, T_i, i): A request for access to a resource is made by P_i.
- (Reply, T_j, j): P_j grants access to a resource under its control.
- (Release, T_k, k): P_k releases a resource previously allocated to it.

The algorithm is as follows:

1. When P_i requires access to a resource, it issues a request (Request, T_i, i), time-stamped with the current local clock value. It puts this message in its own array at $q[i]$ and sends the message to all other processes.
2. When P_j receives (Request, T_i, i), it puts this message in its own array at $q[i]$. If $q[j]$ does not contain a request message, then P_j transmits (Reply, T_j, j) to P_i. It is this action that implements the rule described previously, which assures that no earlier Request message is in transit at the time of a decision.
3. P_i can access a resource (enter its critical section) when both of these conditions hold:

   a. P_i's own Request message in array q is the earliest Request message in the array; because messages are consistently ordered at all sites, this rule permits one and only one process to access the resource at any instant.

   b. All other messages in the local array are later than the message in $q[i]$; this rule guarantees that P_i has learned about all requests that preceded its current request.

3. P_i releases a resource by issuing a release (Release, T_i, i), which it puts in its own array and transmits to all other processes.
4. When P_i receives (Release, T_j, j), it replaces the current contents of $q[j]$ with this message.
5. When P_i receives (Reply, T_j, j), it replaces the current contents of $q[j]$ with this message.

Distributed mutual exclusion algorithm 2:

1. **When P_i requires access to a resource, it issues a request (Request, T_i, i), timestamped with the current local clock value. It puts this message in its own array at q[i] and sends the message to all other processes.**

2. **When P_j receives (Request, T_i, i), it obeys the following rules:**
    a. **If P_j is currently in its critical section, it defers sending a Reply message (see Rule 4, which follows)**
    b. **If P_j is not waiting to enter its critical section (has not issued a Request that is still outstanding), it transmits (Reply, T_j, j) to P_i.**
    c. **If P_j is waiting to enter its critical section and if the incoming message follows P_j's request, then it puts this message in its own array at q[i] and defers sending a Reply message.**
    d. **If P_j is waiting to enter its critical section and if the incoming message Precedes P_j's request, then it puts this message in its own array at q[i] and transmits (Reply, T_j, j) to P_i.**

3. **P_i can access a resource (enter its critical section) when it has received a Reply message from all other processes.**
4. **When P_i leaves its critical section, it releases the resource by sending a Reply message to each pending Request.**