# Resource allocation using Logical Clocks

The resource allocation algorithm using Logical Clocks given by Lamport should be implemented within the Netsim program.

## General

When first giving a `public void trigg()` to a node it should do a request. When doing a `trigg()` again at the same node it should do a release. Then a request again and so on. The clock value as well as the three first requests in the queue should be shown as VisibleString objects by thy node.

When a node puts a request it should also call `setWaken();` on the node interface. This will change the color of the node on the screen.

When a node is allowed to use the resource it should call `setActive();` on the node interface.

When a node releases the resource it should call `setIdle();` on the node interface.

The messages should have a tag with sender, timestamp and type of message.

For each event a node should write to `public void writeLogg(String row);` on the node interface in order to trace the events. Then the real physical order of events can be viewed by selecting view -> system log on the menu bar.

N.B. The Logical Clock algorithm requires a network that is fully connected, i.e. from any node there is a connection to any other node. The network should also be safe and FIFO.

## Algorithm conditions

- The algorithm uses
  - Distributed Request Queue (empty at start)
  - Logical Clock
- each process administers:
  - a local copy of the Request Queue
  - a local copy of the Logical Clock
  - a table containing the latest received timestamp from each of the other processes in the system
    (TS-table)
- When placing a request at process $P_i$, it is associated with the current local logical clock value $C_i$ in $P_i$.
- all messages between two processes is delivered in the same order as they were sent and no message will disappear (FIFO).
  This will be the case when using a communication protocol such as TCP/IP.

## The algorithm events

- $P_i$ *REQUEST*:
  - <$T_i$;$P_i$;*REQUEST*> is sent to all other processes, $T_i$ is the timestamp taken from $C_i$
  - <$T_i$;$P_i$;*REQUEST*> is also put in $P_i$'s local copy of the Request Queue sorted according to "$\rightarrow_T$"
  - $P_i$ increments its local Logical Clock value
- $P_j$ receives *REQUEST* <$T_i$;$P_i$;*REQUEST*>:
  - $P_j$ adjusts its local copy of the Logical Clock according to the Logical Clock definition
  - <$T_i$;$P_i$;*REQUEST*> is put in $P_j$'s local copy of the Request Queue sorted according to "$\rightarrow_T$"
  - $P_j$ updates its TS-table
  - $P_j$ increments its local Logical Clock value
  - $P_j$ sends an acknowledgement <$T_j$;$P_j$;*ACK*> to $P_i$
  - $P_j$ increments its local Logical Clock value
- $P_i$ receives an acknowledgement <$T_j$;$P_j$;*ACK*> from $P_j$:
  - $P_i$ adjusts its local copy of the Logical Clock according to the Logical Clock definition.
  - $P_i$ updates its TS-table with $T_j$ from $P_j$
  - $P_i$ increments its local Logical Clock value
- $P_i$ is allowed access to the resource when:
  - <$T_i$;$P_i$;*REQUEST*> is number one in the (local) Request Queue
  - $T_i \rightarrow_T T_j$ for all $T_j$ in the (local) TS-table
- $P_i$ want to *RELEASE*:
  - <$T_i'$;$P_i$;*RELEASE*> is sent to all other processes, $T_i'$ is the timestamp taken from actual $C_i$
  - <$T_i$;$P_i$;*REQUEST*> is erased from the (local) Request Queue
  - $P_i$ increments its local Logical Clock value
- $P_j$ receives <$T_i'$;$P_i$;*RELEASE*>:
  - $P_j$ adjusts its local copy of the Logical Clock according to the Logical Clock definition.
  - <$T_i$;$P_i$;*REQUEST*> is erased from the (local) Request Queue
  - $P_j$ updates its TS-table with $T_i'$ from $P_i$
  - $P_j$ increments its local Logical Clock value