Lamport Clocks, Vector Clocks, Totally Ordered Multicast and Causally Ordered Multicast

```
Assumptions:
- a group of processes G = {P1, ..., Pn} in a distributed system;
- for Lamport Clocks, initially: forall i, Ci = 0
- for Vector Clocks, initially: forall i, VCi = [0, ..., 0]
- pack(x,y) builds a message from its payload x and timestamp y
- send(x,y) sends message x to process y
```

Lamport Clocks

```
LCsend(payload) { // @Pi
   // update local Lamport clock
   Ci = Ci + 1
   // build message with timestamp
   m  = pack(payload, Ci)
   // send message
   send(m,Pj)
}

LCreceive(m) { // @Pj
   // get message timestamp
   ts = timestamp(m)
   // update local Lamport clock
   Cj = max { Cj, ts }
}

LCdeliver(m) { // @Pj
   // pass message to local process
   deliver m to Pj
   // update local Lamport clock
   Cj = Cj + 1
}
```

Totally Ordered Multicast

```
TOMsend(payload) { // @Pi
   // update local Lamport clock
   Ci = Ci + 1
   // build message with timestamp
   m  = pack(payload, Ci)
   // send to all processes in group
   forall P in G
      send(m,P)
}

TOMreceive(m) { // @Pj
   // adjust local Lamport clock
   ts = timestamp(m)
   Cj = max { Cj, ts } + 1
   // send acks if message not an ack
   if (payload(m) is not "ack") {
     ackm = pack("ack", Cj)
     forall P in G
        send(ackm,P)
   }
   // add message to priority queue
   add m to Q
}

TOMdeliver(m) { // @Pj
   forever {
     // check message at the from of the queue
     m = front of Q
     // check if queue has messages from all other processes
     if (forall k<>i, exists mk from Pk in Q) {
        remove m from Q
        if (payload(m) is "ack")
           discard m
        else
           deliver m to Pj
     }
   }
}
```

Vector Clocks

```
VCsend(payload) { // @Pi
   // update local clock in vector clock
   VCi[i] = VCi[i] + 1
   // build message with timestamp vector
   m  = pack(payload, VCi)
   // send message
   send(m,Pj)
}

VCreceive(m) { // @Pj
   // get message timestamp vector
   ts = timestamp(m)
   // update vector clock in local process
   for all k in {1,...,|G|}
      VCj[k] = max { VCj[k], ts[k] }
}

VCdeliver(m) { // @Pj
   // update local clock in vector clock
   VCj[j] = VCj[j] + 1
}
```

Causally Ordered Multicast

```
COMsend(payload) { // @Pi
   // update local clock in vector clock
   VCi[i] = VCi[i] + 1
   // build message with timestamp
   m = pack(payload, VCi)
   // send to all processes in group except sender
   forall P in G\{Pi}
     send(m,P)
}

COMreceive(m) {  // @Pj
   // place message in queue
   add m to Q
}

COMdeliver(m) { // @Pj
  forever {
    // check if there are deliverable messages
    forall m in Q {
      ts = timestamp(m)
      // check the causality conditions
      if ( ts[i] == VCj[i] + 1 and
           forall k<>i, ts[k] <= VCj[k] ) {
          // update local vector clock
          forall k in {1,..,|G|}
            VCj[k] = max { VCj[k], ts[k] }
          // deliver message
          remove m from Q and deliver to Pj
      }
    }
  }
}
```