Anne Poso
Samuel Flinkfelt
474 Distributed Computing
Franklin Algorithm, Non-Blocking

## Pseudocode

```
Main{
    int rank, size
    initialize MPI // Initializes MPI
    initialize Comm Size // Figures out the number of processors I am asking for
    initalize Rank // Figures out which rank we are
    bool active_array = {true, true, true, ,true} // if a process is active or passive
    bool is_Elected = false; // if a process has been elected leader

    // multiple election rounds until one node elected as leader
    while(not is_Elected)
        out_message1 = outMessage2 = rankID[rank]
        print process ID and rank
        if( active_array[rank] )
            // send rankID msg to both neighbor nodes
            send MPI message neighbor right
            send MPI message to neighbor left
            // receive rankID msgs from both neighbor nodes
            receive message from right neighbor
            receive message from left neighbor
            // wait for receive operations to complete to guarantee message delivery
            MPI wait right neighbor
            MPI wait left neighbor
        else if (rank == size -1)
            send MPI message neighbor right
            send MPI message to neighbor left
            receive message from right neighbor
            receive message from left neighbor
            MPI wait right neighbor
            MPI wait left neighbor
        else
            send MPI message neighbor right
            send MPI message to neighbor left
            receive message from right neighbor
            receive message from left neighbor
            MPI wait right neighbor
            MPI wait left neighbor
        // if current node is lesser than any of its neighbor nodes
        If (rightmsg > rankID[rank] OR leftmsg > rankID[rank])
            Is_Active = false
            print process has become active
        // if receiving msgs are same ID as current node,
        // then it is the last active node, so it is elected as leader
        else if (rightmsg is equal rankID[rank] OR leftmsg is equal rankID[rank])
```

```
                        is_Elected = true
                        print process has become leader
        else // process is passive, so forward incoming msg to next node
                // passive node receives message from neighbor and
                // sends same buffer to next node
                if rank == 0
                        // receive rankID msgs from neighbor nodes
                        receive message from right neighbor
                        receive message from left neighbor
                        // forward the same msg buffer to next node
                        send message to right neighbor
                        send message to left neighbor
                        // wait for receive operations to complete to guarantee message delivery
                        MPI wait right neighbor
                        MPI wait left neighbor
                else if rank == size - 1
                        receive message from right neighbor
                        receive message from left neighbor
                        send message to right neighbor
                        send message to left neighbor
                        MPI wait right neighbor
                        MPI wait left neighbor
                else
                        receive message from right neighbor
                        receive message from left neighbor
                        send message to right neighbor
                        send message to left neighbor
                        MPI wait right neighbor
                        MPI wait left neighbor
Abort // terminate processes after leader is elected
Finalize // shutdown MPI
return 0
```