# CS 171 Spring 2020
# Programming Assignment II

Assigned: April 20th, 2020
Due On Demo Day: May 4th 2020

Blockchain is a list of blocks which are linked using cryptography. Each block consists of one or more transactions (money transfers) along with the hash value of the previous block.

In this project, you will develop a replicated blockchain whose access is controlled by mutual exclusion. In other words, a process needs to get mutual exclusion in order to add a transaction to the blockchain, and then broadcasts this update to all other processes. You should develop the application logic that uses Lamport's Distributed Solution to achieve mutual exclusion.

## 1 Project Overview

Each user starts with a balance of $10, and can transfer money to other users via the command-line interface. For each transfer, the process will request exclusive access to the blockchain via Lamport mutual exclusion. Upon receiving access, the process will add a block containing the money transfer to its local copy of the blockchain. Then the process will broadcast this update to all other processes before releasing the resource.

## 2 Implementation Details

1. Each process should maintain a local copy of the blockchain, as well as a queue for mutex requests.

2. The blockchain can be implemented as a linked list whose nodes each represent a transaction. A transaction $\langle S, R, amt \rangle$ consists of a sender, $S$, a receiver, $R$, and an amount of money $amt$. This is a simplified blockchain which does not require any hash pointers.

3. A process cannot send more money than it has. Before requesting the resource, the process should check its own balance and report "FAILURE" if it has insufficient funds for a transfer.

4. As before, processes should communicate with each other via a network process "in the middle" whose purpose is to add a **constant delay of 1 second** before routing a message to the recipient process.

5. Recall that a process can't gain access to the blockchain until it receives replies from all other processes, and is at the head of the queue. Only then can the process add a transaction to the blockchain.

# 3 System Configuration and User Interaction

1. All processes will be run on the terminal. No front-end UI is necessary.

2. Each process has a unique id: $\boldsymbol{P_1}$, $\boldsymbol{P_2}$ and $\boldsymbol{P_3}$. When starting a process, it should connect to the network processor and the network processor should know the ports on which the 3 processes are listening. This could be done via a configuration file or other methods that are appropriate.

3. Each process must take 3 types of input from the user:

   - *Transfer event*: Arguments to this input should include the recipient process $R$ and the transfer amount *amt*. Upon receiving this input, the process should first check its current balance. There are two cases:
     (a) If the process has insufficient funds to complete the transaction, it should report "FAILURE" and do nothing more.
     (b) If there are sufficient funds, the process should request access to the blockchain via Lamport mutual exclusion. Before sending *release* messages to the other processes, the process first sends a *broadcast* message broadcasting the new block to all other processes. Upon receiving the *broadcast* message, every process appends the new block to its local blockchain. A process $\boldsymbol{P_i}$ upon receiving a new block checks if the new block contains a transaction with receiver as $\boldsymbol{P_i}$; if so $\boldsymbol{P_i}$ updates its balance.

     NOTE: For the purposes of this assignment, we will only test values of *amt* in dollar increments (no cents).
   - *Print balance*: If the user chooses this input, the process should print its current balance to the screen.
   - *Print blockchain*: If the user chooses this input, the process should display its local copy of the blockchain.

4. In addition to receiving user input, processes will also communicate with each other and need to handle messages of several types, e.g. *Request, Reply, Release*, and *Broadcast* messages.

5. Your network processor MUST be a separate process; if your 3 processes are directly communicating with each other without a separate network process, you will lose points for the assignment.

6. Your network processor must be able to handle concurrent connections (e.g., $\boldsymbol{P_1}$ sends *request* message to $\boldsymbol{P_2}$ and immediately to $\boldsymbol{P_3}$; this will create concurrent connections in the network processor).

7. You are welcome to log any other relevant messages on the console during the demos, such as when a process receives a message from another process.

8. Use message passing primitives TCP/UDP for inter-process communications. Given the FIFO requirement for Lamport mutual exclusion, TCP may be the better choice.

9. There is no restriction on the programming language.

# 4 Example Scenario

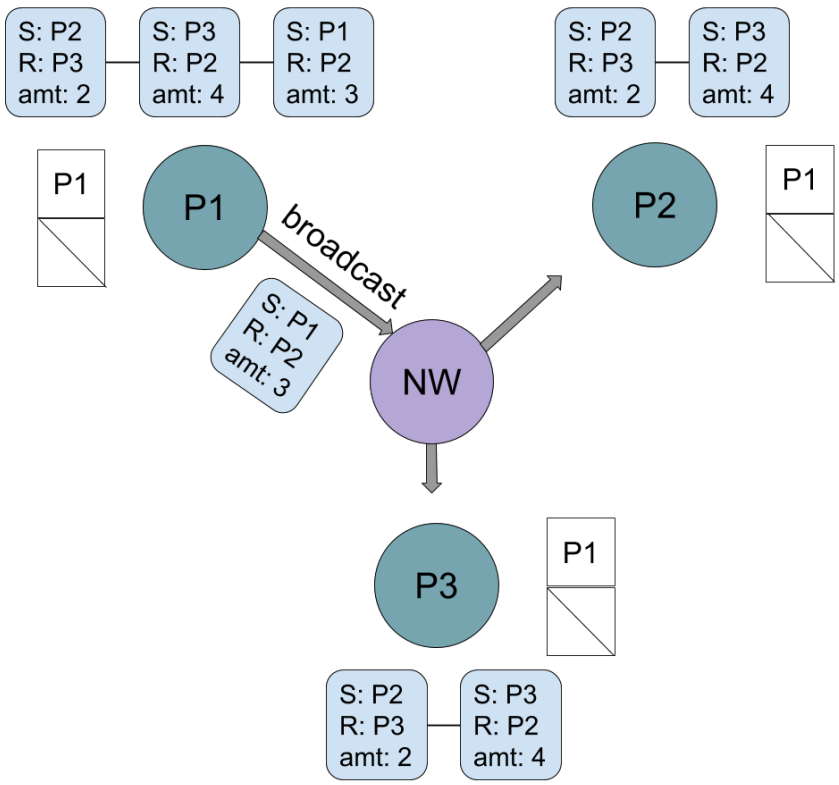
## 4.1 Ex. 1

$\mathbf{P_1}$ transfer event: send \$1 to $\mathbf{P_2}$
$\mathbf{P_1}$ print blockchain: [ (P1, P2, \$1) ]
*Note: all blockchains should be identical at this point.
$\mathbf{P_1}$ print balance: \$9
$\mathbf{P_2}$ print balance: \$11

## 4.2 Ex. 2

concurrent $\begin{cases} \mathbf{P_2} \text{ transfer event: send \$2 to } \mathbf{P_3} \\ \mathbf{P_3} \text{ transfer event: send \$4 to } \mathbf{P_2}. \end{cases}$

$\mathbf{P_1}$ print blockchain: [ (P2, P3, \$2), (P3, P2, \$4) ]
$\mathbf{P_2}$ print balance: \$12
$\mathbf{P_3}$ print balance: \$8
$\mathbf{P_1}$ transfer event: send \$3 to $\mathbf{P_2}$
$\mathbf{P_2}$ print blockchain: [ (P2, P3, \$2), (P3, P2, \$4), (P1, P2, \$3)]
$\mathbf{P_1}$ print balance: \$7
$\mathbf{P_2}$ print balance: \$15
$\mathbf{P_3}$ print balance: \$8




$\mathbf{P_1}$ broadcasts a blockchain update in **Ex. 2**.

# 5 Deadlines, Extension and Deployment

We will have a short demo for this project. It will be on **May 4th, 2020** via Zoom, and both team members should be present if you are working as a pair. Time details will be announced later. Please be ready with the working program at the time of your demo.