MVBCoin Overview

Tasks of an MVBCoin node

- Accept incoming connections
- Receive transactions through those connections
  - Check them for validity
  - Broadcast valid transactions to other nodes
- Collect transactions into blocks
- Mine complete blocks (by finding a valid nonce for each)
- Broadcast mined blocks to other nodes

Key components of the sample code (Go)

- Accept incoming connections -- **`main()`**
- Receive transactions through those connections -- **`handleIncomingConnection()`**
  - Broadcast valid transactions -- **`broadcaster()`**
- Collect transactions into blocks -- **`mineManager()`**
- Mine complete blocks -- **`mine()`**
- Broadcast mined blocks -- **`broadcaster()`**

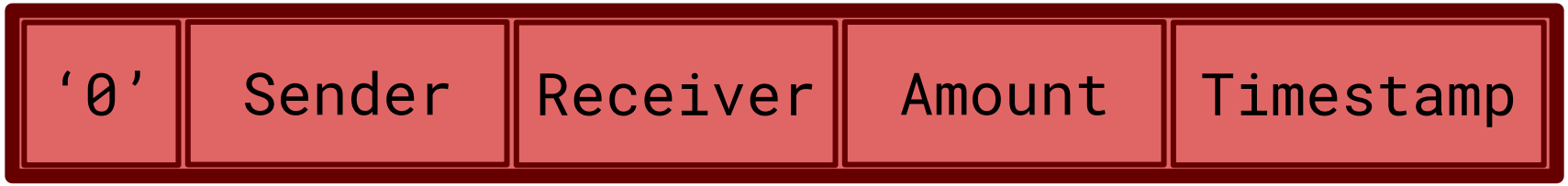
Key components of the sample code (Java)

- Accept incoming connections -- **`Main`**
- Receive transactions through those connections -- **`IncomingConnectionHandler`**
  - Broadcast valid transactions -- **`Broadcaster`**
- Collect transactions into blocks -- **`MineManager`**
- Mine complete blocks -- **`Miner`**
- Broadcast mined blocks -- **`Broadcaster`**

Accepting Incoming Connections (**`Main`**)

- Open a socket to listen for incoming connections (100 / 187)
  - Sockets are an OS construct that manages network connections for us, with an interface similar to a file.
- Call **`accept()`** to see if we received a connection (128 / 220)
  - If there was a connection, **`accept()`** will return a new socket managing that specific connection.
  - Start **`IncomingConnectionHandler`** to read messages sent to us through this socket.
- Note: **`Main`** also has logic for initialization and shutdown.

Transaction Messages



Before processing, we must answer:

- Have we seen this transaction already?
- Do we know the sender and receiver?
- Does the sender have enough money to pay for this transaction?

Receiving Transactions (`IncomingConnectionHandler`)

- MVBCoin nodes support four message types
- First step is to read the first byte of a message - this will tell us which message type we are dealing with (189, 344)
- Then, we read in the rest of the message and process it.
  - Transaction messages - first byte is ‘0’ (192-208, 347-360)
  - Shutdown messages - first byte is ‘1’ (211-220, 363-367)
  - Block messages - first byte is ‘2’ (223-248, 369-395)
  - get_block messages - first byte is ‘3’ (251-272, 399-412)
- Once message is processed, read in next message

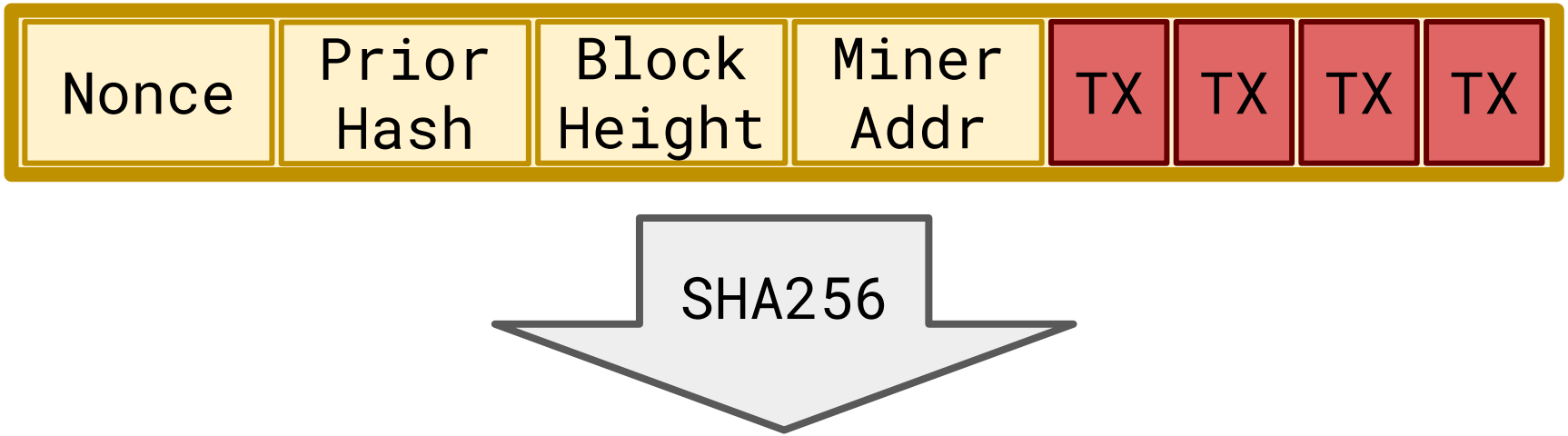
Receiving Transactions (cont.)

- After reading in the entire Transaction message, we call **`handleTransaction()`**. (208, 360)
- **TODO**: First, check to make sure this is the first time we've seen this transaction (282, 430)
- Then, we parse the transaction (284-290, 432-440)
- **TODO:** check if the sender and receiver are valid and if the sender has enough money, then transfer the funds (294, 442)
- If the transaction is valid, broadcast it and pass it over to **MineManager** to add it to the next block (296-300, 444-447)

Collecting Transactions into Blocks (**`MineManager`**)

- Receive a new transaction (340, 542)
- Add the transaction to our current block (347, 591)
- Check if we completed a block (349, 593). If we did:
  - Create space for the next block of transactions (350, 594)
  - If not currently mining, start mining by:
    - Add metadata to the current block, such as the block height, our node ID, previous hash (323-328, 510-514)
    - Start **`Miner`** (330, 516-517)
  - Can't start mining if already mining (it's a block**chain**)

Mining a block



00 00 00 00 ba 78 16 bf 8f 01 cf ea 41 41 40 de 5d ae 22 23 b0 03 61 a3 96 17 7a 9c b4 10 ff 61

If at least **`difficulty`** bytes are null, this was a valid nonce.

Mining (**`Miner`**)

- **TODO**: write this function
- The goal of this function is to repeatedly run:
  `sha256.sum([NONCE]+[PRIOR_HASH]+[BLOCKHEIGHT]+[MINER_ADDR]+BLOCKDATA)`
  - Note: **`MineManager`** already built this string for us.
- Each time, try a different value for [NONCE]
- Check how many initial bytes are zero
  - If at least **`difficulty`** initial bytes are zero, we found a valid nonce! Return this nonce to **`MineManager`**.

Generating the Block Message (**`MineManager`**)

- Receive a new nonce (357, 542)
- Format the Block message (362-373, 557-572)
- Add the block to our list of processed blocks (377, 576-578)
- Send it to **`Broadcaster`** to send to peer nodes (380, 581)
- Check if we have any other complete blocks ready to be mined (383-384, 584-585)
    - If we do, add metadata and start mining (385, 586)

Broadcasting Messages (`Broadcaster`)

- Receive the first message to broadcast (144, 246-253)
- Open a socket to each peer (148-155, 257-268)
- Broadcast the message (158-168, 271-277)
- If the message was a close message, exit after broadcasting it (163, 278-281)
- Otherwise, wait to receive another message to broadcast (169, 283-290)