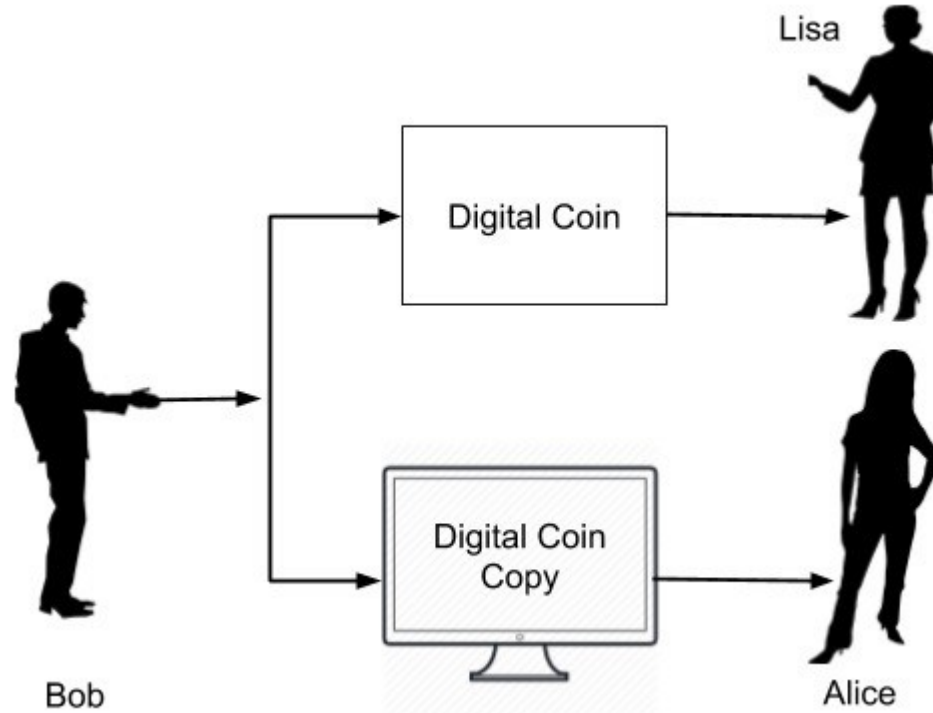
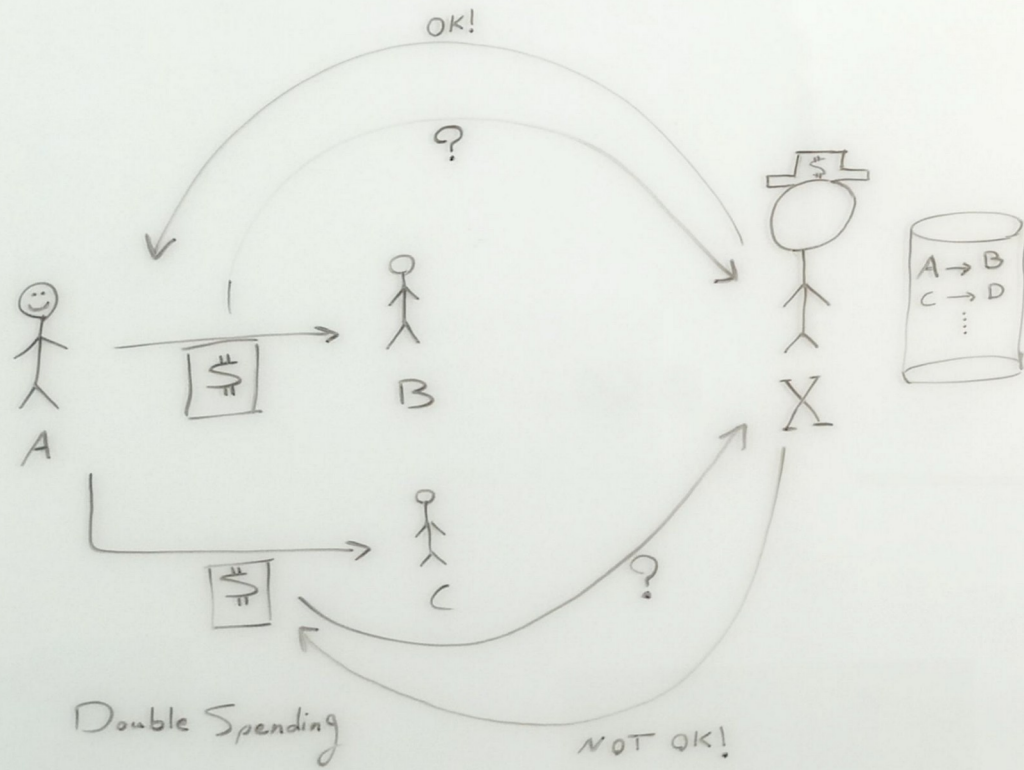


# Final Project

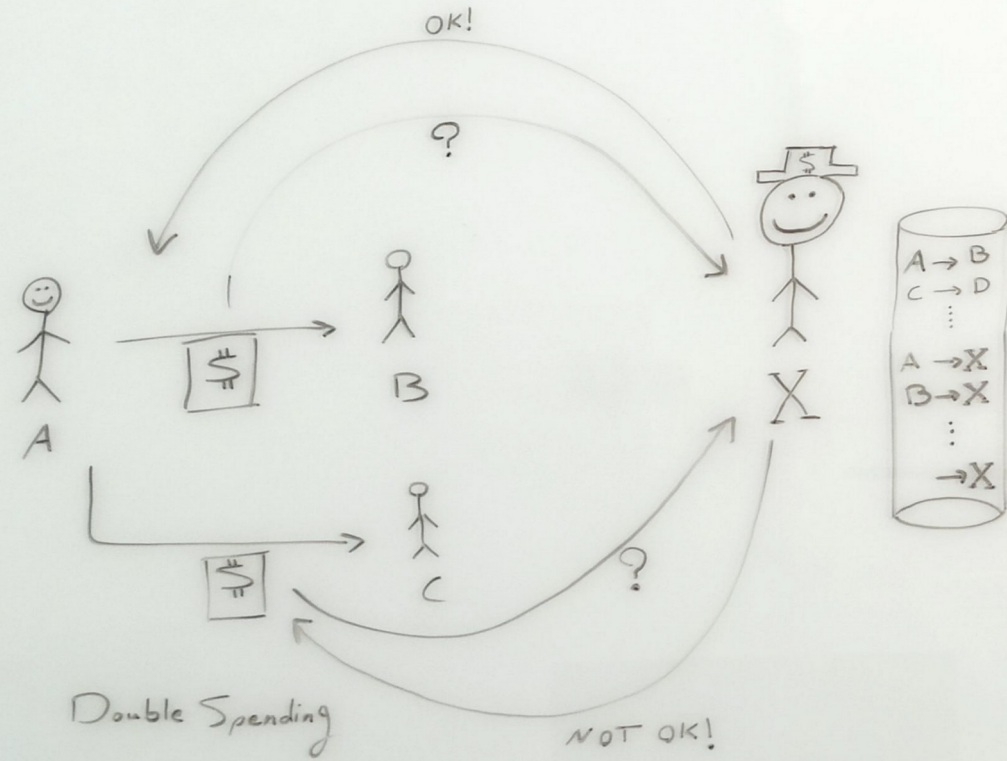
- Designed by Sadegh Jabbari.
- No restrictions on using AI.
- Videos are the documentations.
- Goal: Building the software core of a bitcoin miner.
- You may work in teams of up to two members.
- It's Open Source baby!

# Double Spending

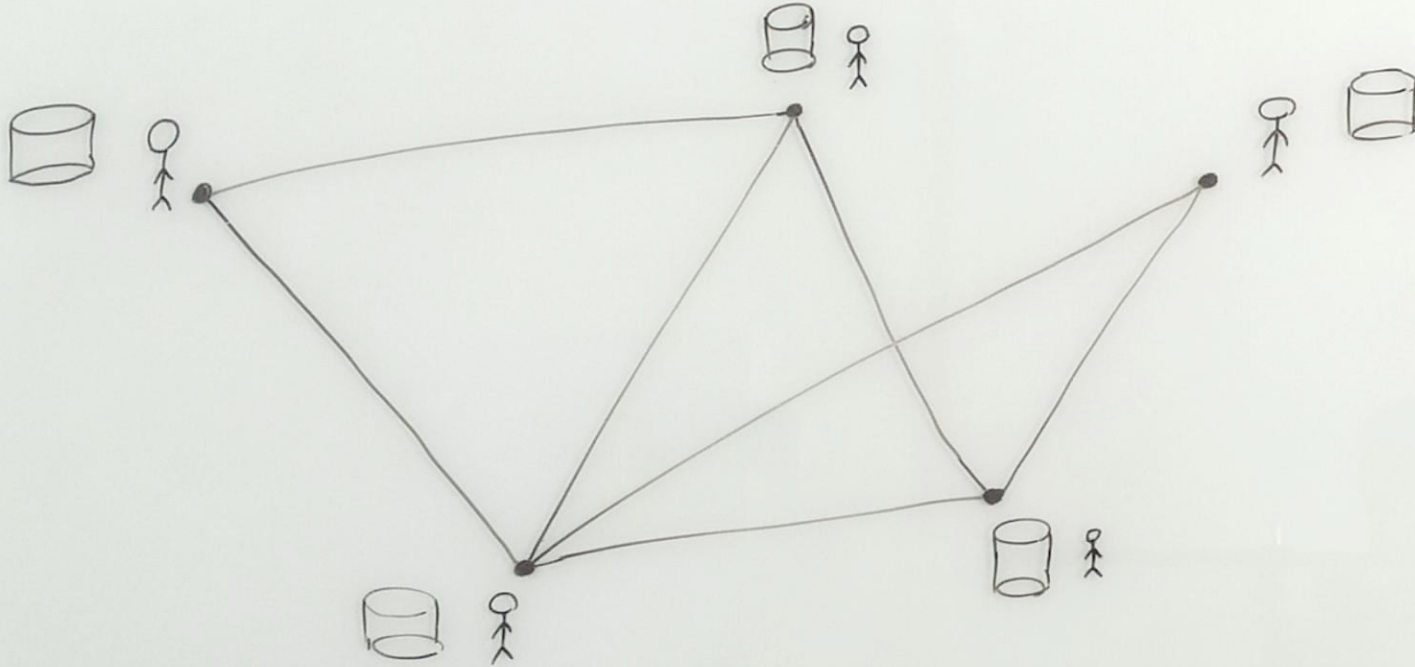




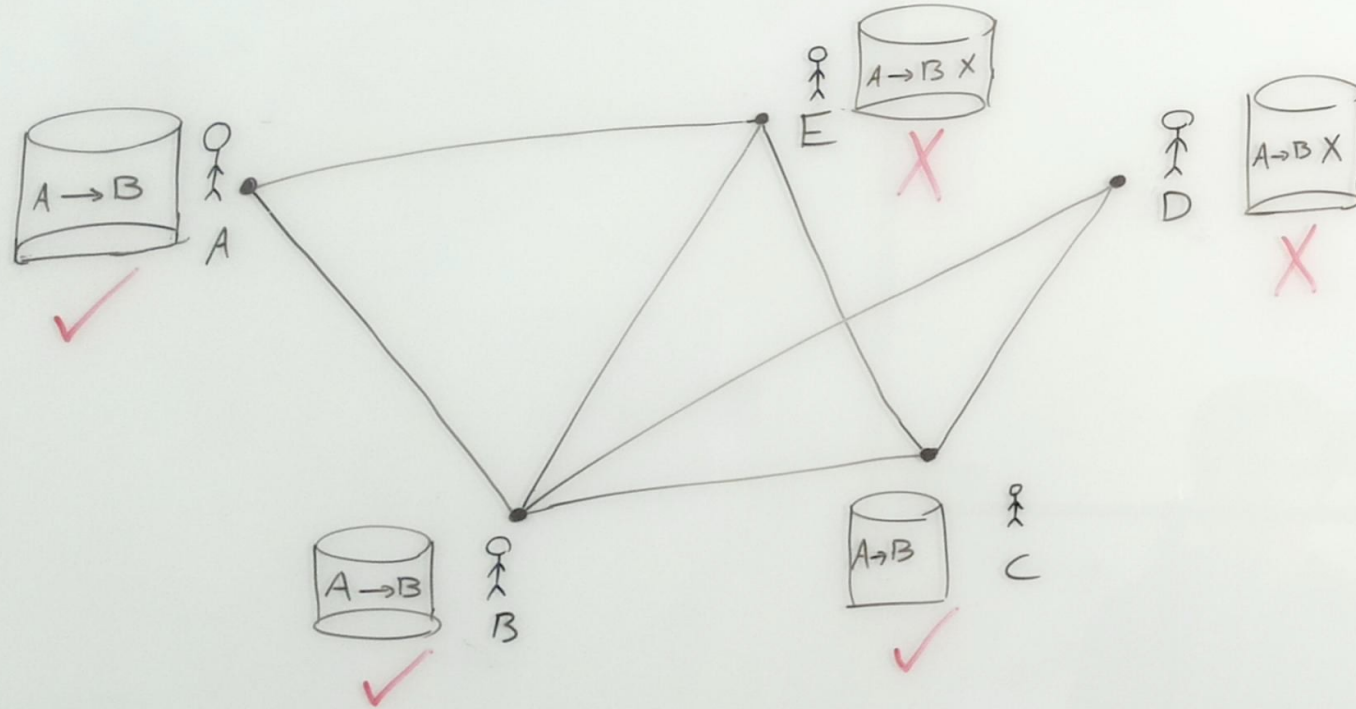
# Centralization



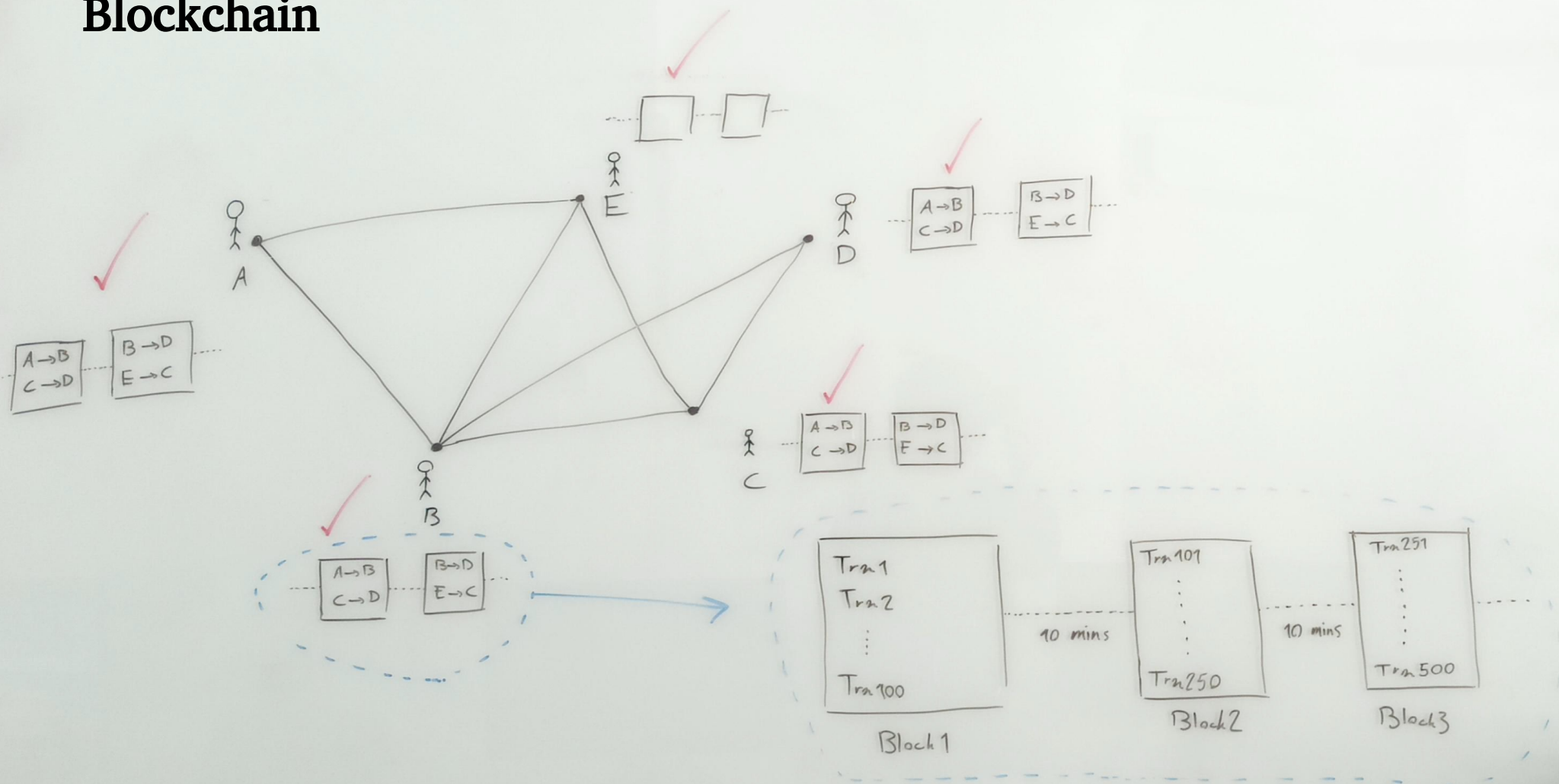
# Decentralization



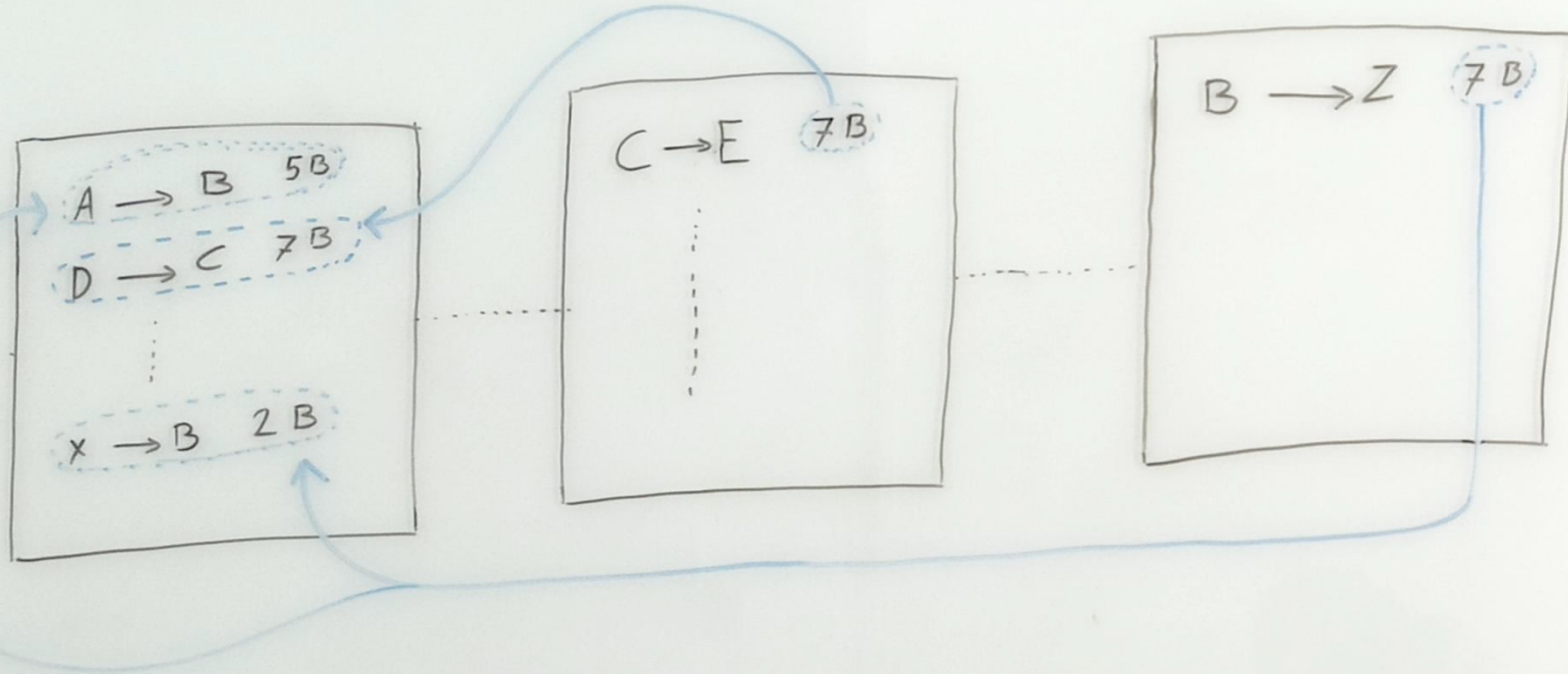
# Consistency



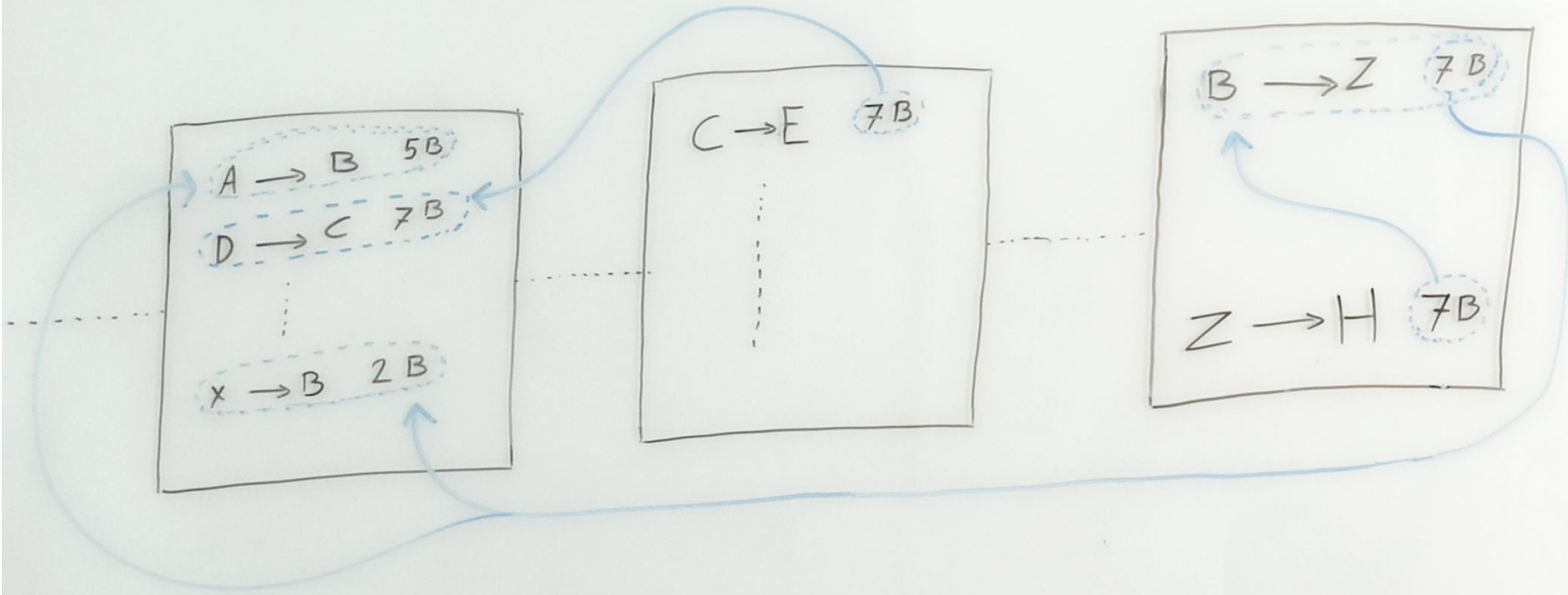
# Blockchain



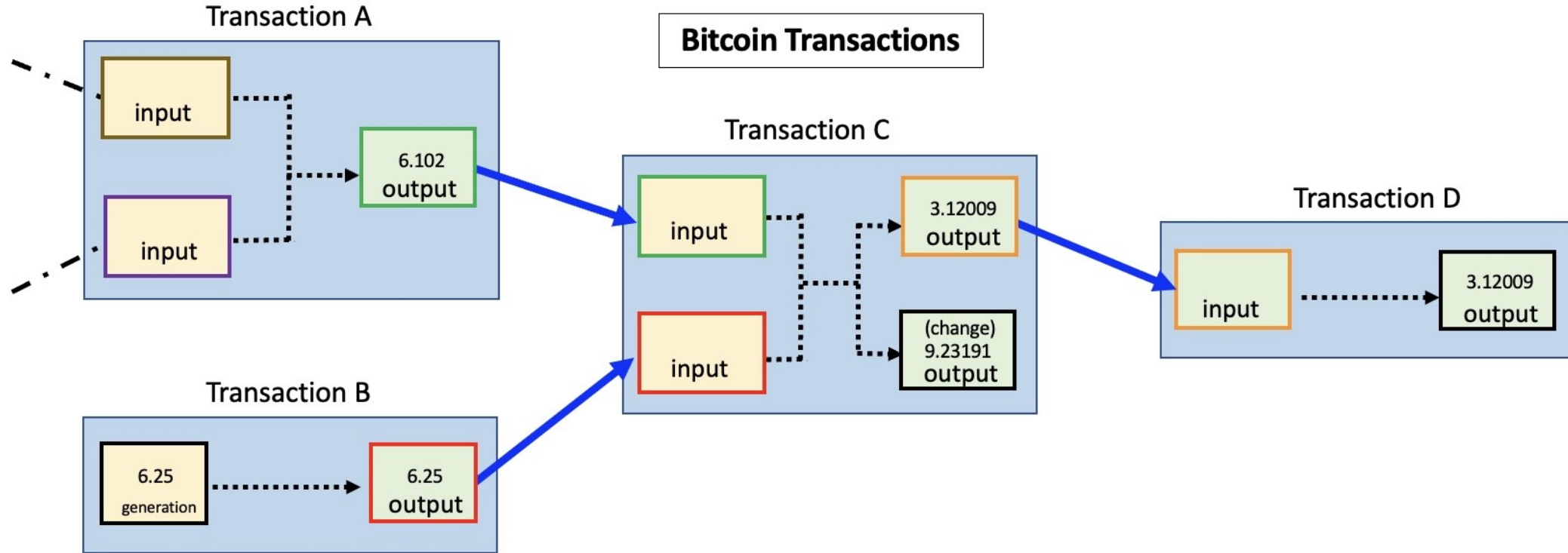
# Transaction Dependencies



# Transaction Dependencies



# Transaction Dependencies



```

1 {
2   "txid": "9f3c2a7b8e1d4c5a6b7e8f90123456789abcdef0123456789abcdef01234567",
3   "inputs": [
4     {
5       "prevTxId": "a1b2c3d4e5f67890123456789abcdef0123456789abcd",
6       "prevOutIndex": 0,
7       "publicKey": "02a1633caf7bf12e9d7c91b2f8e4a5c6d7e8f90123456789abcdef012345678",
8       "signature": "3045022100f3a1b2c3d4e5f67890123456789abcdef02203b4c5d6e7f8a9b0c"
9     },
10    {
11      "prevTxId": "b9c8d7e6f5a43210987654321fedcba9876543210fedcba9876543210fedcba",
12      "prevOutIndex": 1,
13      "publicKey": "03b7c9d8e1f2a3456789abcdef0123456789abcdef0123456789abcdef01",
14      "signature": "304402207a8b9c0d1e2f33445566778899aabbccddeeff00112233445566770220445566778899aabb"
15    }
16  ],
17  "outputs": [
18    {
19      "value": 1.25,
20      "publicKey": "03f1e2d3c4b5a697887766554433221100ffeeddccbbaa998877665544332211"
21    },
22    {
23      "value": 0.75,
24      "publicKey": "02aa99887766554433221100ffeeddccbbaa99887766554433221100ffeeddcc"
25    }
26  ]
27 }

```

$$\text{Fee}(0.5) = \text{outputs}(2.5) - \text{inputs}(2)$$

## First Reference Transaction

[illegible]

# Second Reference Transaction

```
1 {
2   "txid": "b9c8d7e6f5a43210987654321fedcba9876543210fedcba9876543210fedcba",
3   "inputs": [
4     {
5       "prevTxId": "1111111111111111111111111111111111111111111111111111111111111111",
6       "prevOutIndex": 2,
7       "publicKey": "03previousownerbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb",
8       "signature": "3045022100abcdeffedcba0123456789abcdef0123456789abcdef0220102030405060708"
9     }
10  ],
11  "outputs": [
12    {
13      "value": 0.4,
14      "publicKey": "02someoneelsecccccccccccccccccccccccccccccccccccc"
15    },
16    {
17      "value": 1.0,
18      "publicKey": "03b7c9d8e1f2a3456789abcdef0123456789abcdef0123456789abcdef01"
19    }
20  ]
21 }
```

# Visually explaining **PROOF-OF-WORK**

Proof-of-work is a puzzle that is solved by computational effort. Answers are costly & time consuming to produce, but once solved, are easily verified by others. Being able to prove the validity of transactions without the need for a centralized party makes Bitcoin functionally trustless & secure from tampering.

Think of a Proof-of-Work puzzle like a locked treasure chest, surrounded by keys...

You don't know which key will open each chest, so the only way to find the right one is to try them all.

Each has a predictable amount of coins inside, & who ever opens the chest first gets to keep them. The faster you can check keys, the more likely you'll earn the reward.

## DIFFICULTY



= **EASY**



= **DIFFICULT**

To keep coin distribution predictable, puzzles become more difficult to solve when more people are working on them.

A **Proof-of-Work** solution has three parts:



### NONCE:

A random number with no defined meaning at the beginning of a hash which constitutes proof of work when solved.

### HASH:

A fixed-length number which results in a large amount of predictable & unchanging data when read.

### TRANSACTIONS:

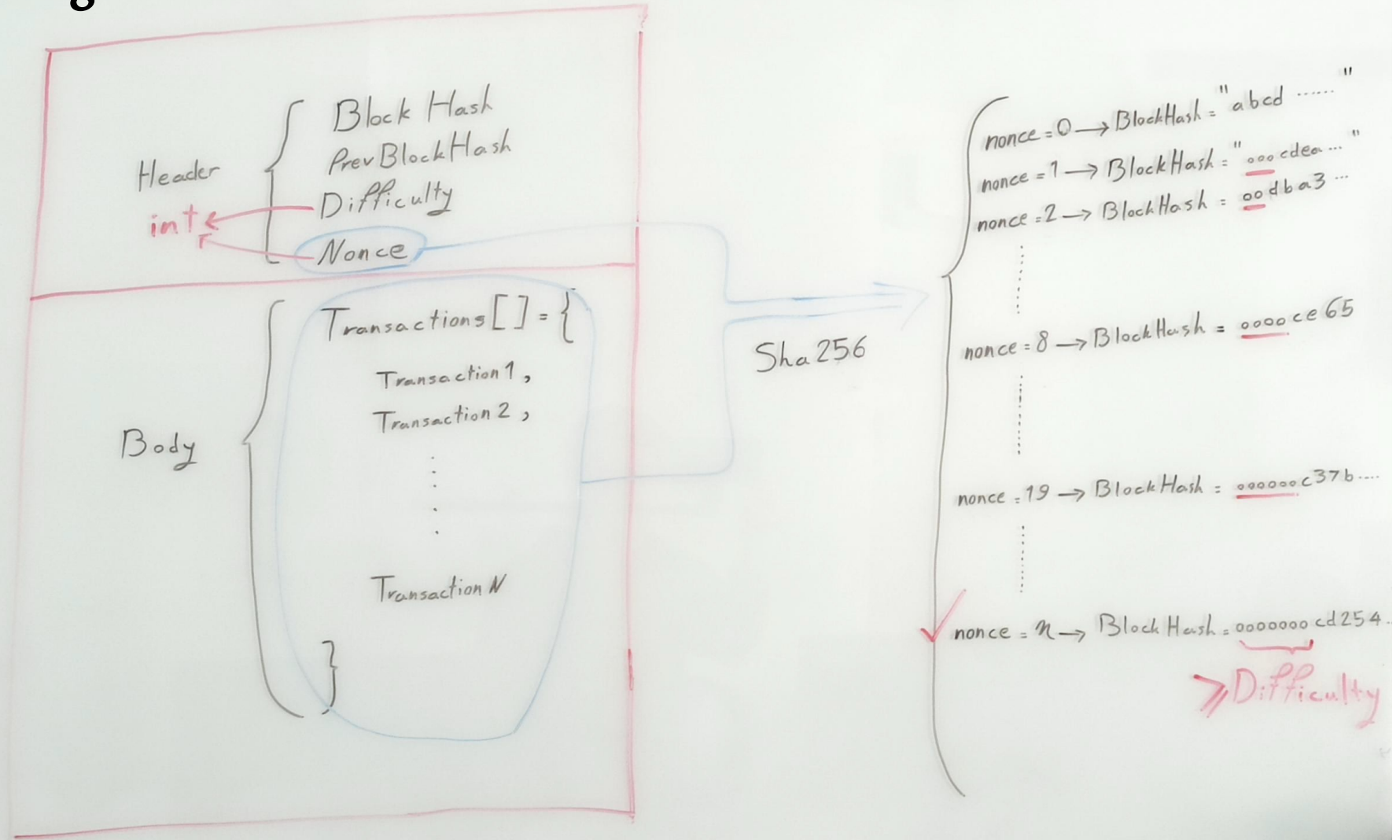
Authenticated transfers of Bitcoin ownership, collected into blocks to be recorded on the Blockchain.

# Block Structure

Header {  
Block Hash  
PrevBlockHash  
Difficulty  
Nonce

Body {  
Transactions[] = {  
Transaction 1,  
Transaction 2,  
...  
Transaction N  
}

# Mining

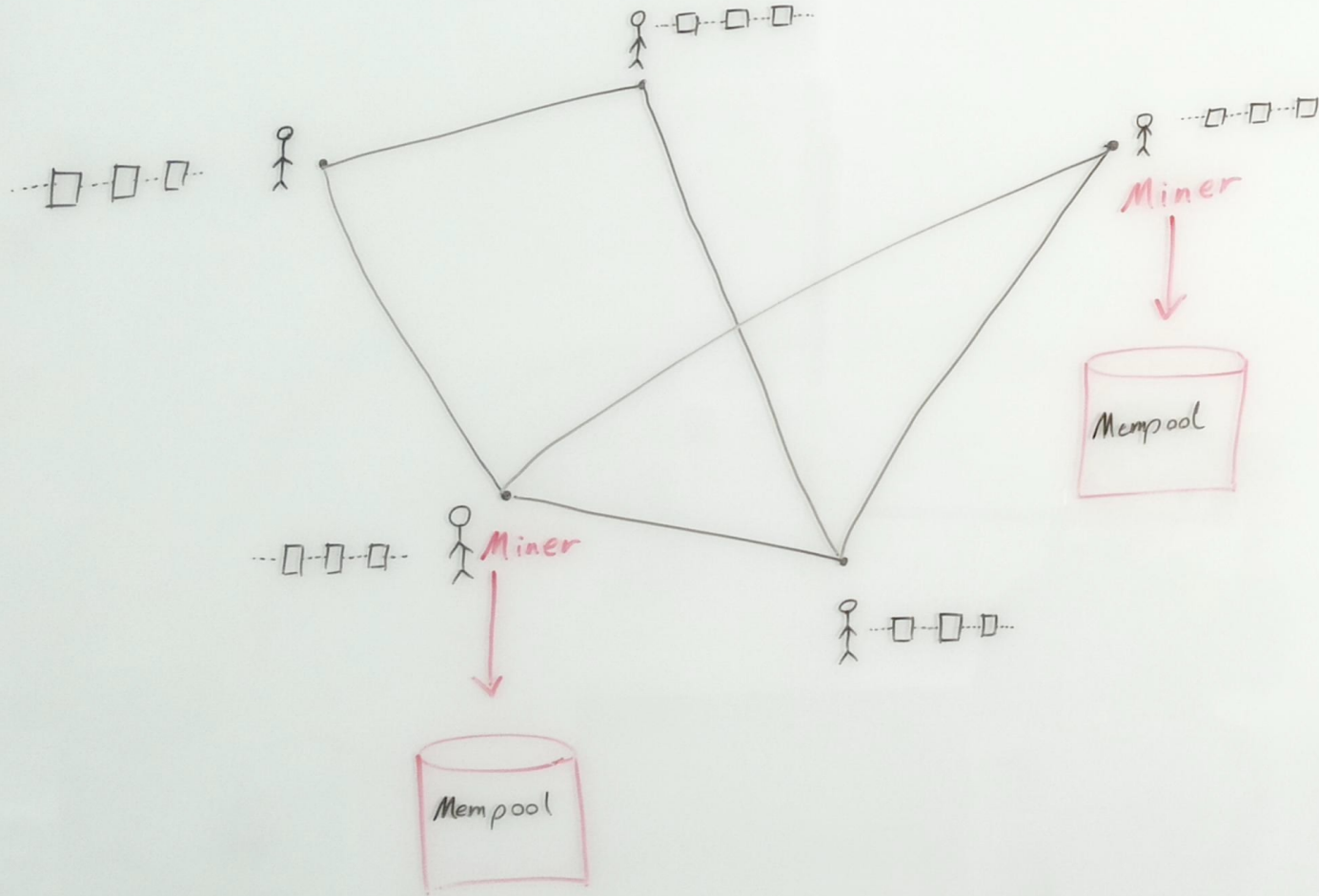


# Hash

```
function simulateHash(blockBody: BlockBody, nonce: int, difficulty: int) -> string:
    """
    Inputs:
        - blockBody: the body of the block to hash
        - nonce: an integer used in the hashing process
        - difficulty: the target number of leading zeros for the hash
    Output:
        - A 64-character hexadecimal string

    Job:
        - Simulates a hash function
        - Has 50% probability that the hash has >= difficulty leading zeros
        - The result is random
    """
```

# Mempool



Mempool

Transaction 1 inputs  
outputs

fee1, size1

Transaction 2

fee2, size2

⋮

Transaction 6

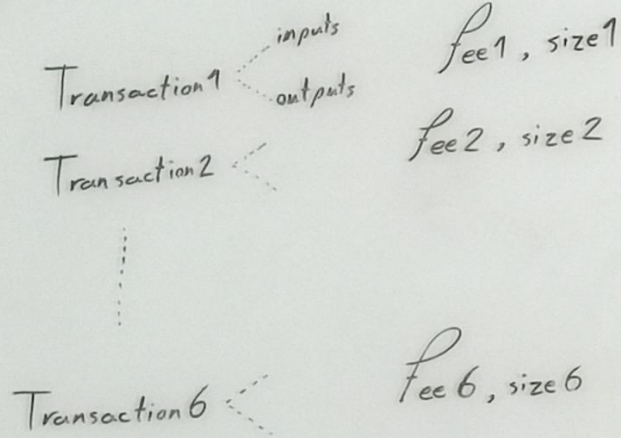
fee6, size6


★ Block size is limited  $\Rightarrow$

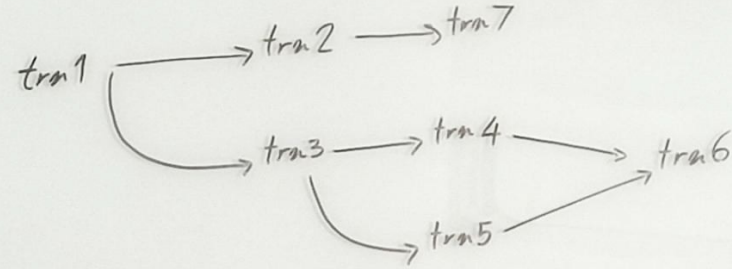
$\text{Max}(\frac{\text{fee}}{\text{size}})$



Mempool



★ Block size is limited  $\Rightarrow \text{Max}(\frac{\text{fee}}{\text{size}})$  ↑ 

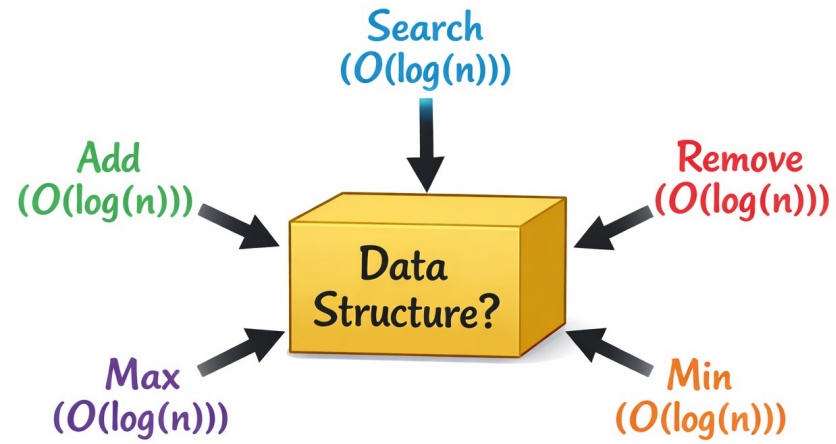


$$\text{Priority}_3 = \frac{\text{FEE}_3}{\text{SIZE}_3} = \frac{\text{fee}_3 + \text{fee}_4 + \text{fee}_5 + \text{fee}_6}{\text{size}_3 + \text{size}_4 + \text{size}_5 + \text{size}_6}$$

$$\text{Priority}_i = \frac{\text{FEE}_i}{\text{SIZE}_i}$$

$$\text{Eviction\_Priority}_4 = \frac{\text{Eviction\_FEE}_4}{\text{Eviction\_SIZE}_4} = \frac{\text{fee}_4 + \text{fee}_3 + \text{fee}_1}{\text{size}_4 + \text{size}_3 + \text{size}_1}$$

$$\text{Eviction\_Priority}_i = \frac{\text{Eviction\_FEE}_i}{\text{SIZE}_i}$$



```
class Mempool:
    # Maps transaction ID (string) to the Transaction object
    txMap: Map<string, Transaction>

    # DAG to track dependencies between transactions
    txDAG: DAG<Transaction>

    # Binary tree to manage transaction priorities
    priorityTree: BinaryTree<Transaction>

    # Binary tree to manage eviction priorities
    evictionTree: BinaryTree<Transaction>

    # Add a new transaction to mempool, update all structures
    function addTransaction(tx: Transaction):
        pass

    # Return all transactions sorted by normal priority
    function getTransactionsByPriority(): List<Transaction>
        pass

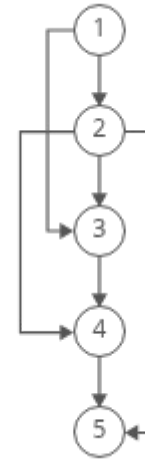
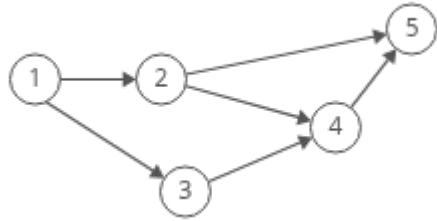
    # Return all transactions sorted by eviction priority
    function getTransactionsByEvictionPriority(): List<Transaction>
        pass

    # Evict the transaction with the maximum eviction priority
    function evictHighestPriorityTransaction():
        pass

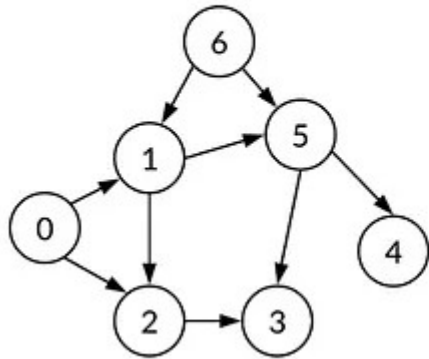
    # Remove a specific transaction from all structures
    function removeTransaction(txId: string):
        pass

    # Get the transaction with the current maximum normal priority
    function getMaxPriorityTransaction(): Transaction
        pass
```

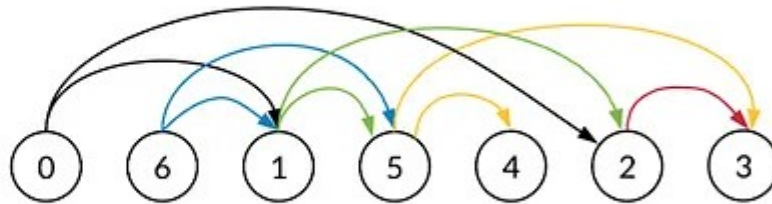
# Topological Sort



Unsorted graph



Topologically sorted graph



## SetDifficulty x

- Set required leading-zero bits to x

## AddTransactionToMempool txJson

- Parse transaction, calculate the fee from dependencies (0 for no dependency); admit to mempool
- Output: mempool sorted in descending order by ancestor fee per byte

## EvictMempool x

- Remove x least valuable transactions by descendant fee per byte (removing their descendants too)
- Output: mempool sorted in ascending order by descendant fee per byte

## MineBlock

- Select by ancestor fee per byte up to the block size limit (10000 bytes), topo-sort, do PoW, update mempool
- Output: full block JSON
- PrevBlockHash: "0000...0000" (64 characters)