Blockchain System Design Overview

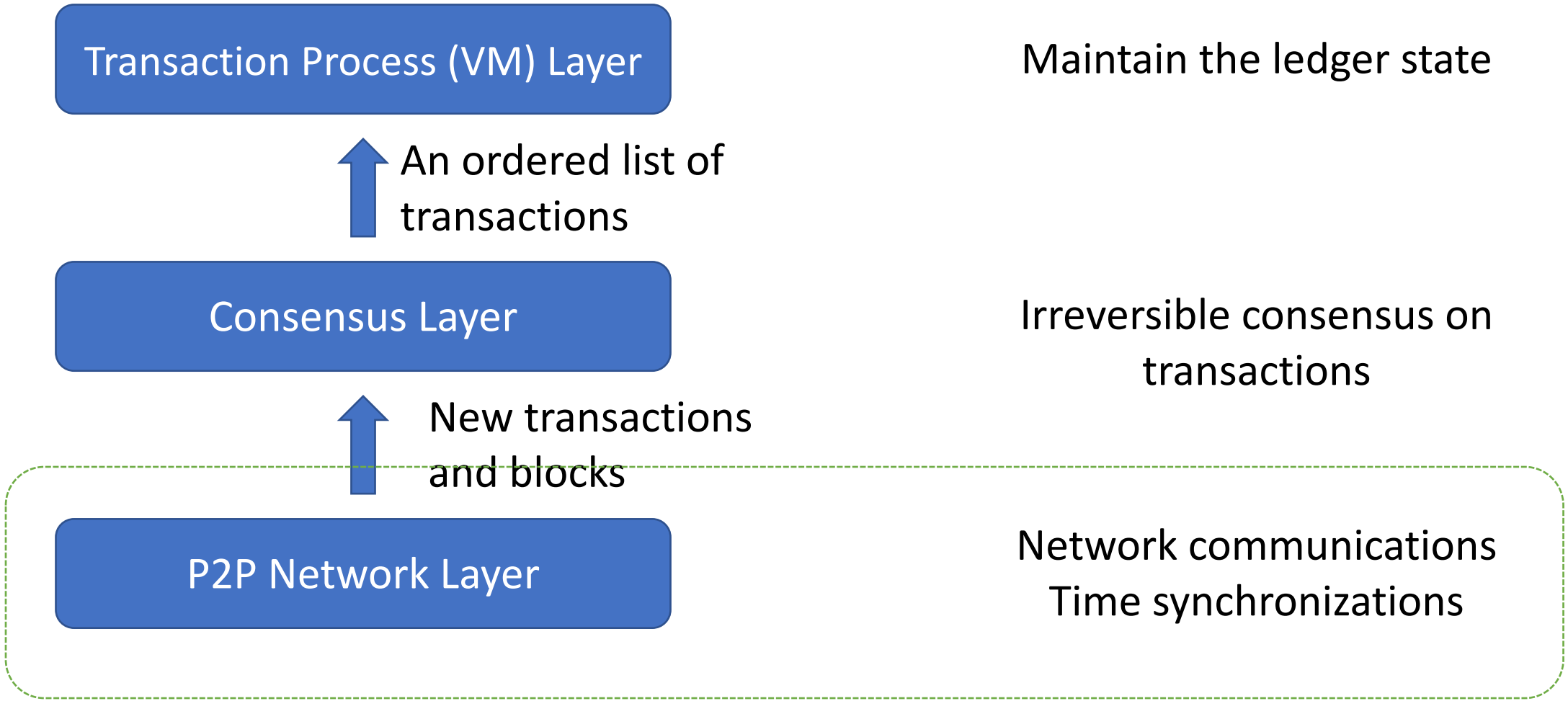
Fan Long

University of Toronto

Components in Blockchain Systems

- Let’s think about what a blockchain system must do
- Propagate transaction and block information → **P2P Network**
- Determine transaction orders → **Distributed Consensus**
  - Proof-of-Work and Mining
  - Stop double spending!
- Process transactions → **Virtual Machine**

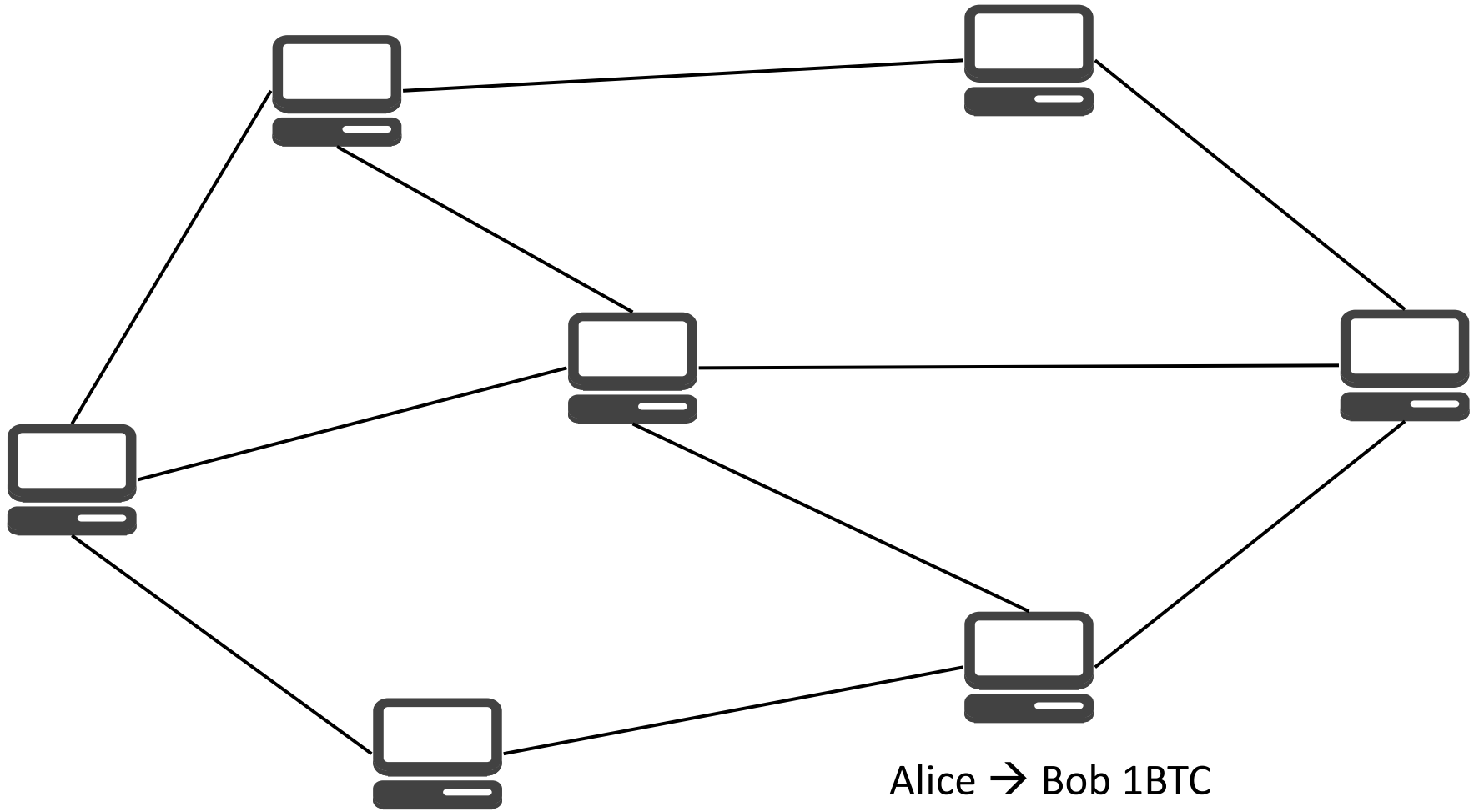
Layers in Blockchain Systems



P2P Network in Blockchain

- Each node connects to a number of peers
  - 32 peers in Bitcoin
- Nodes relay transactions and blocks to all peers recursively
  - Gossip around information
  - Therefore P2P Network is also called Gossip Network
- New nodes can join and leave the network at any time

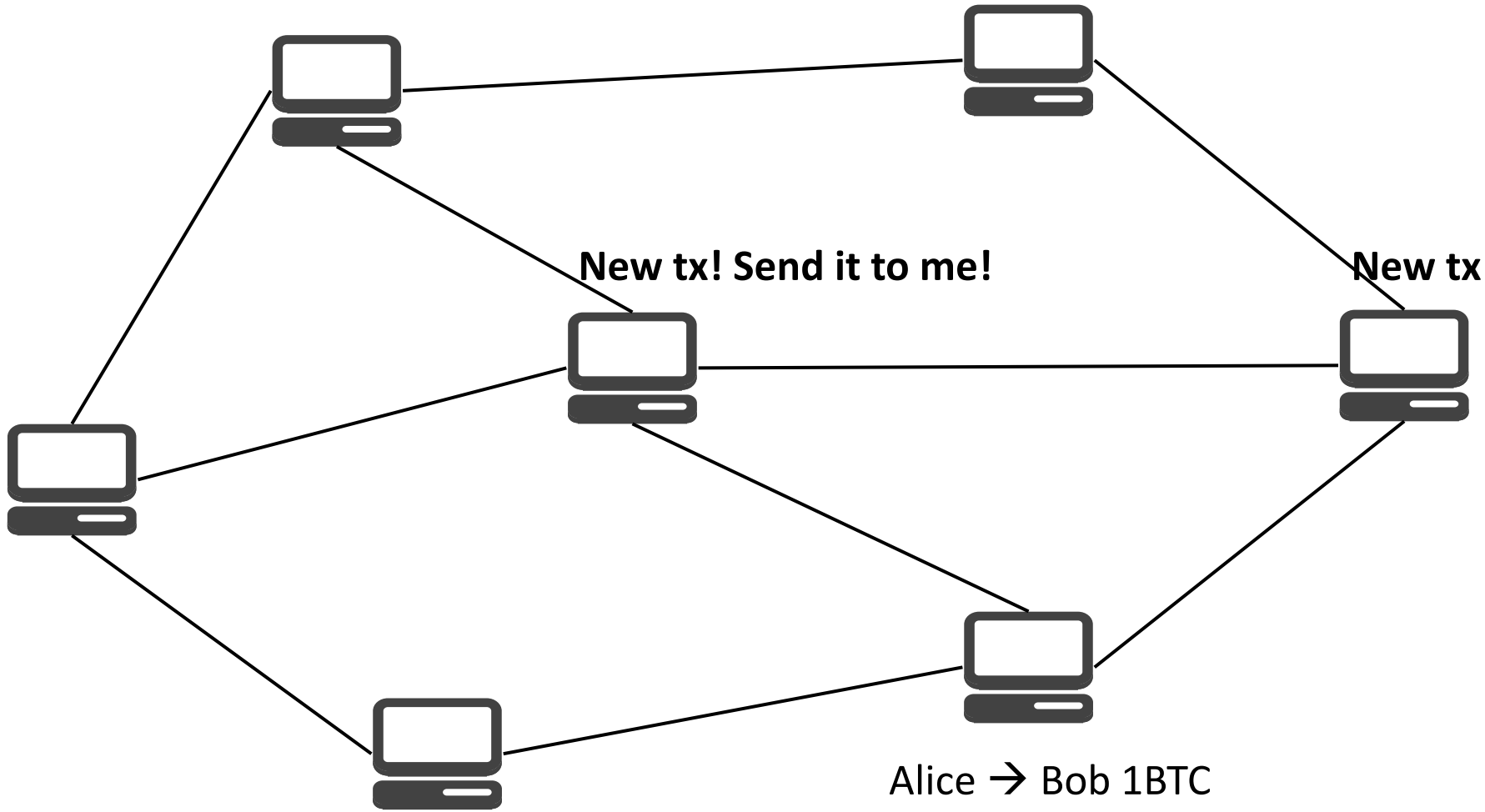
P2P Network in Blockchain



Alice → Bob 1BTC

Hey peers, I got a new transaction with id XXX

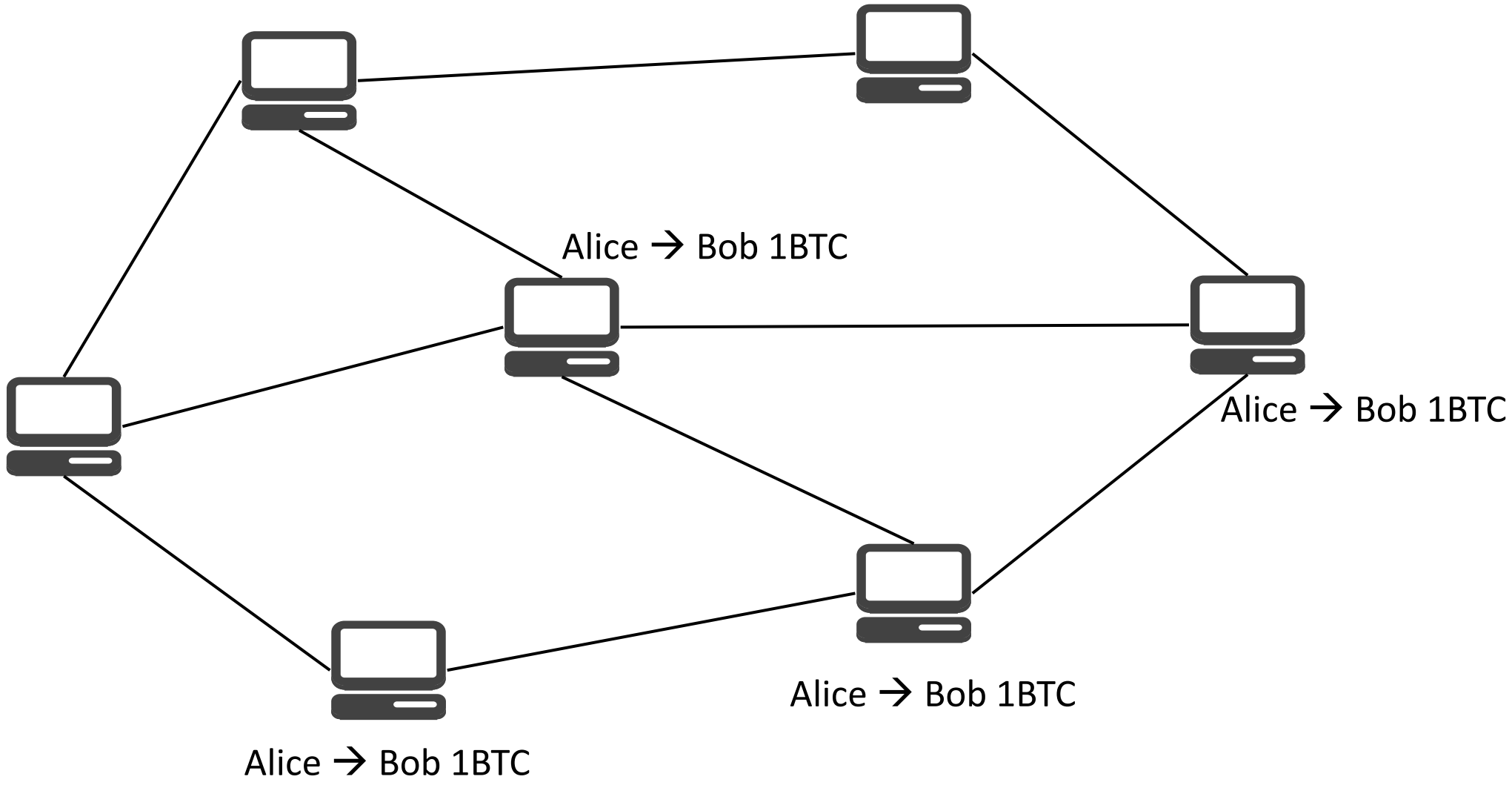
P2P Network in Blockchain



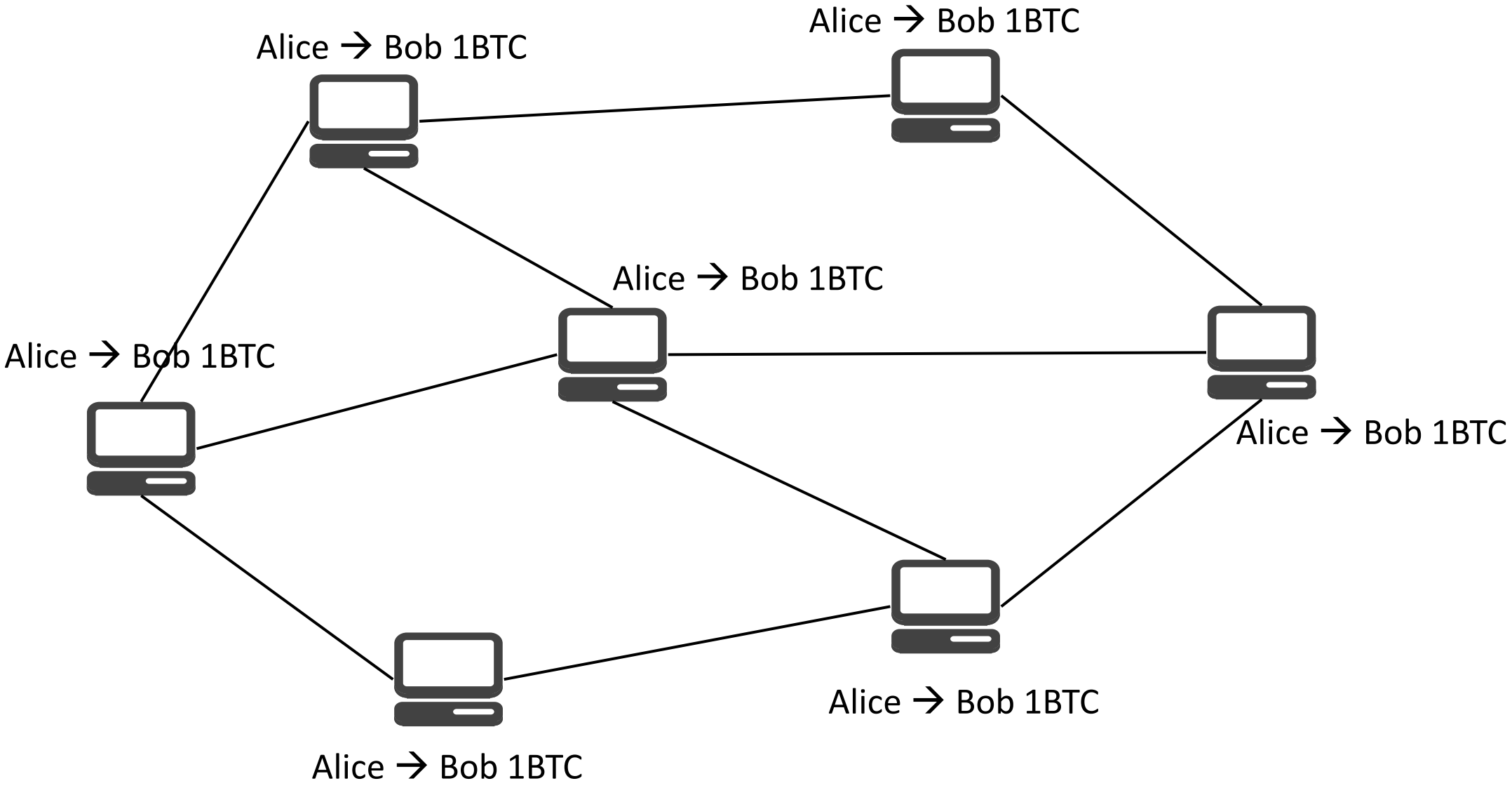
Alice → Bob 1BTC

Hey peers, I got a new transaction with id XXX

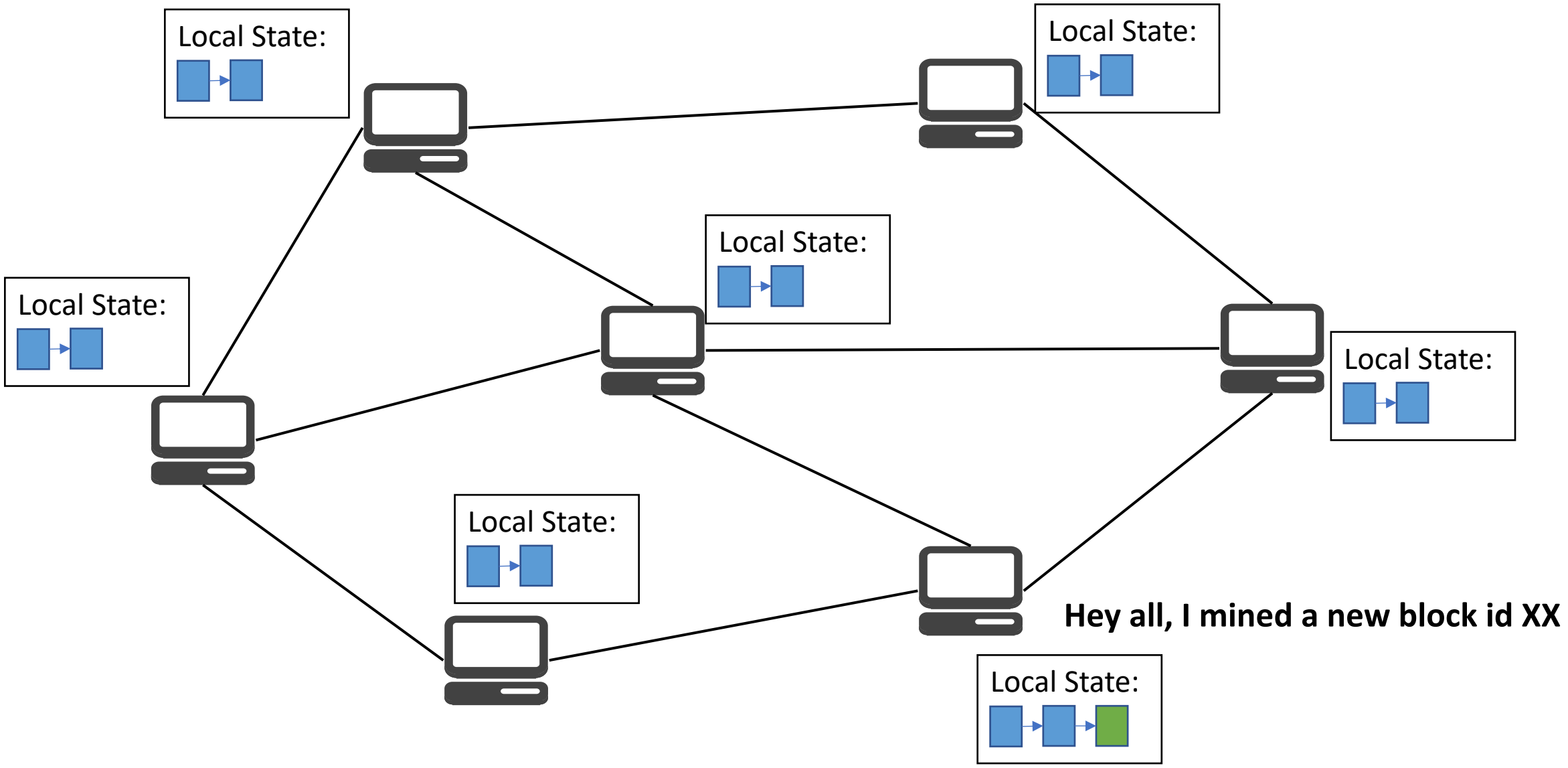
P2P Network in Blockchain



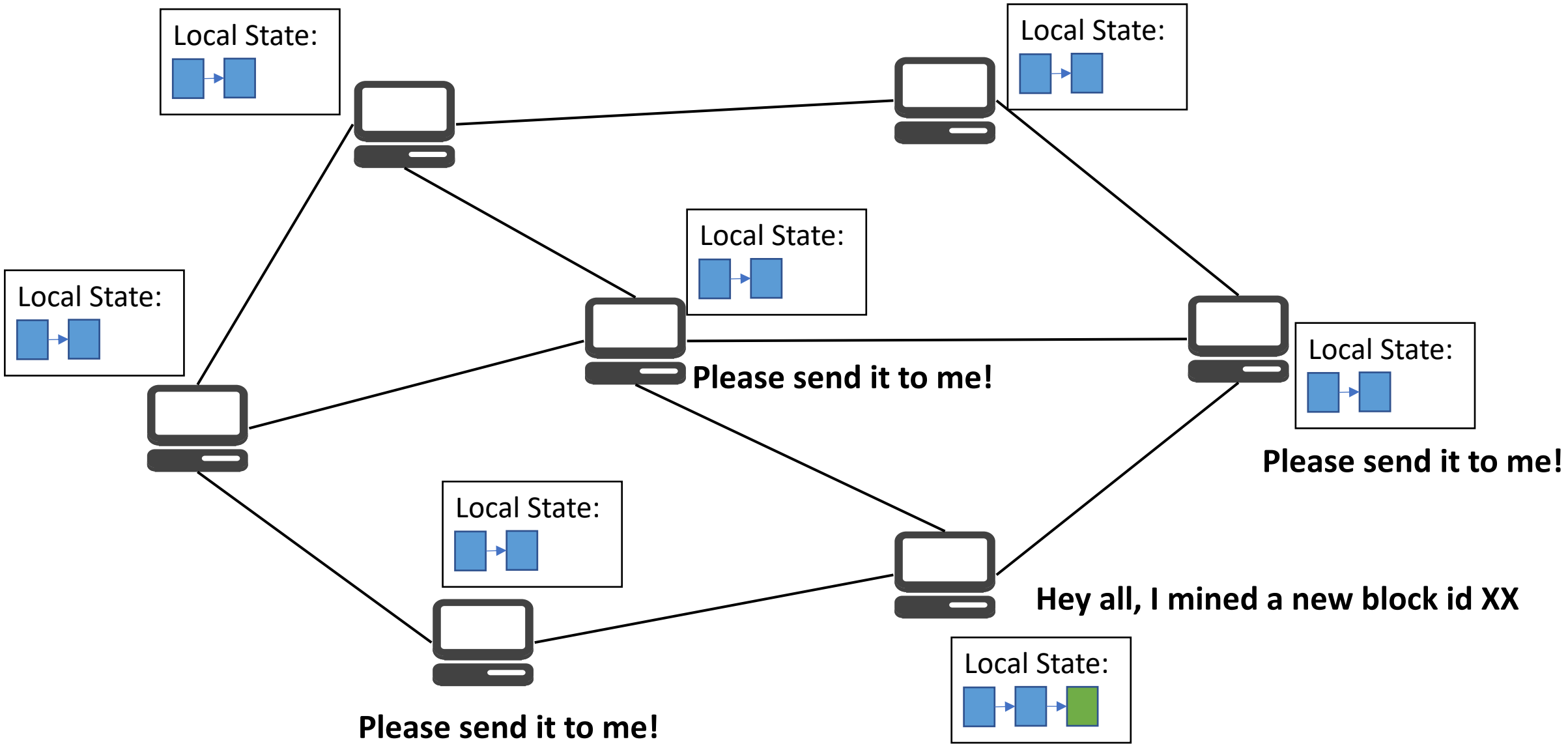
P2P Network in Blockchain



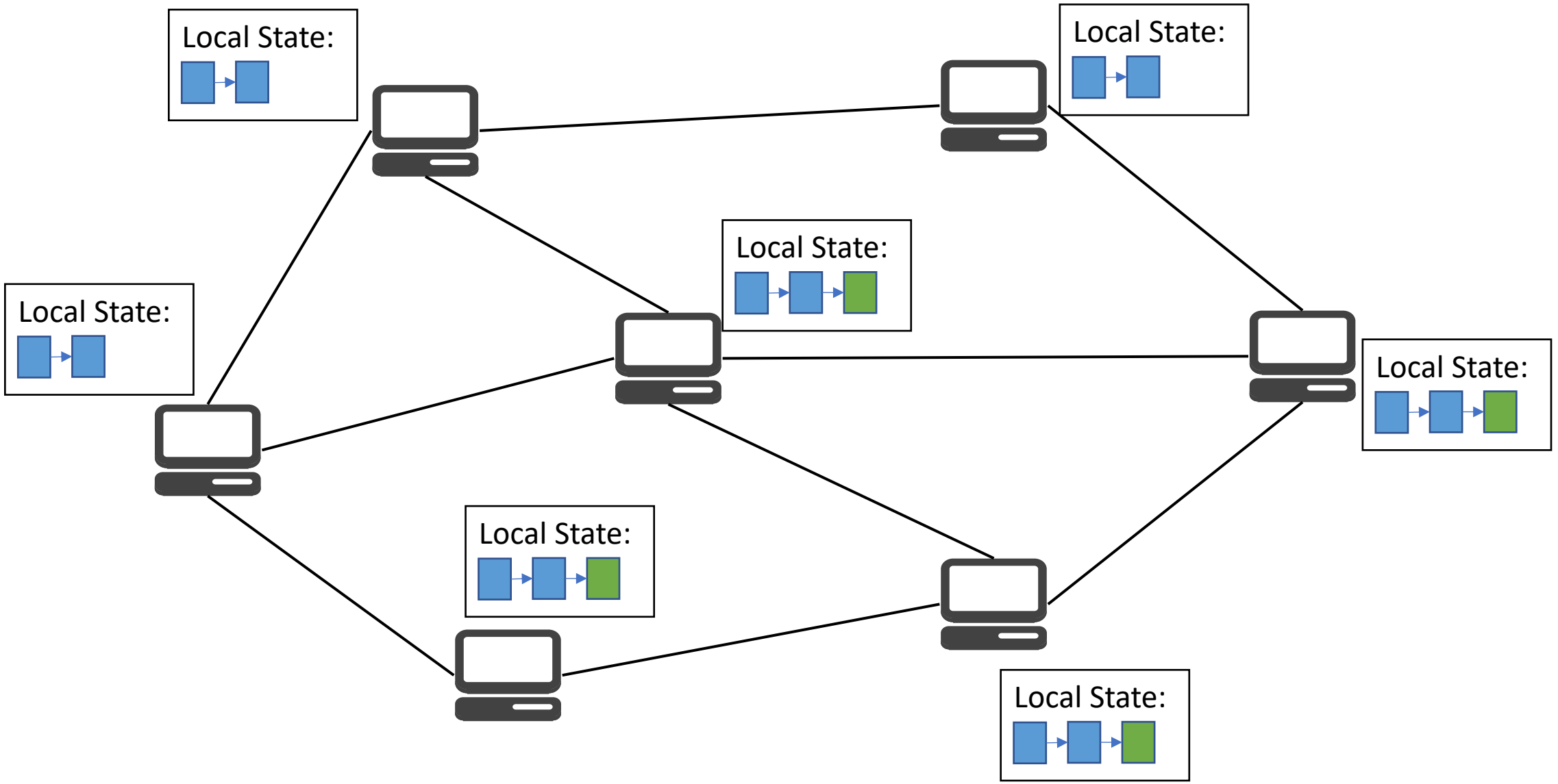
P2P Network in Blockchain



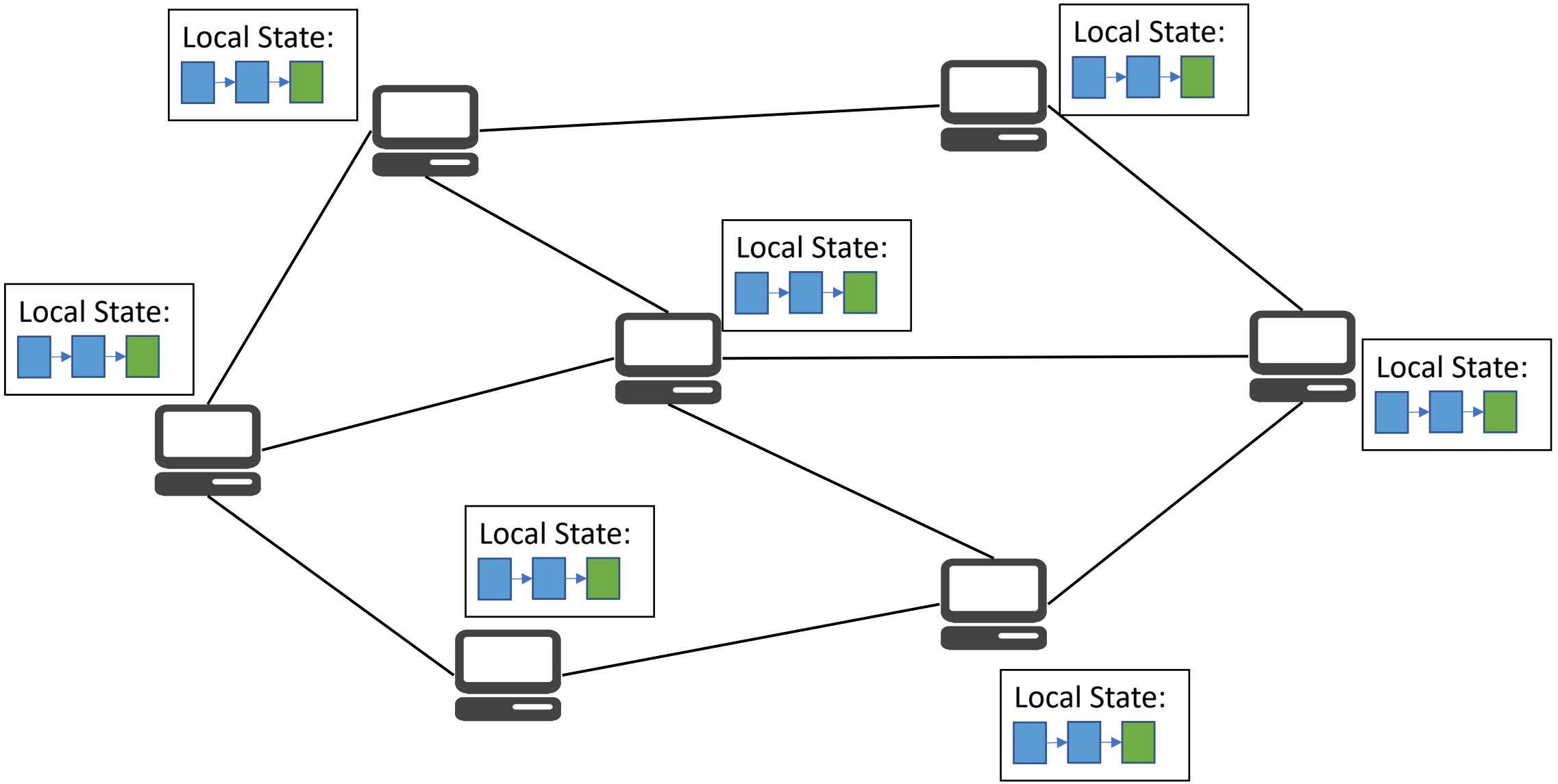
P2P Network in Blockchain



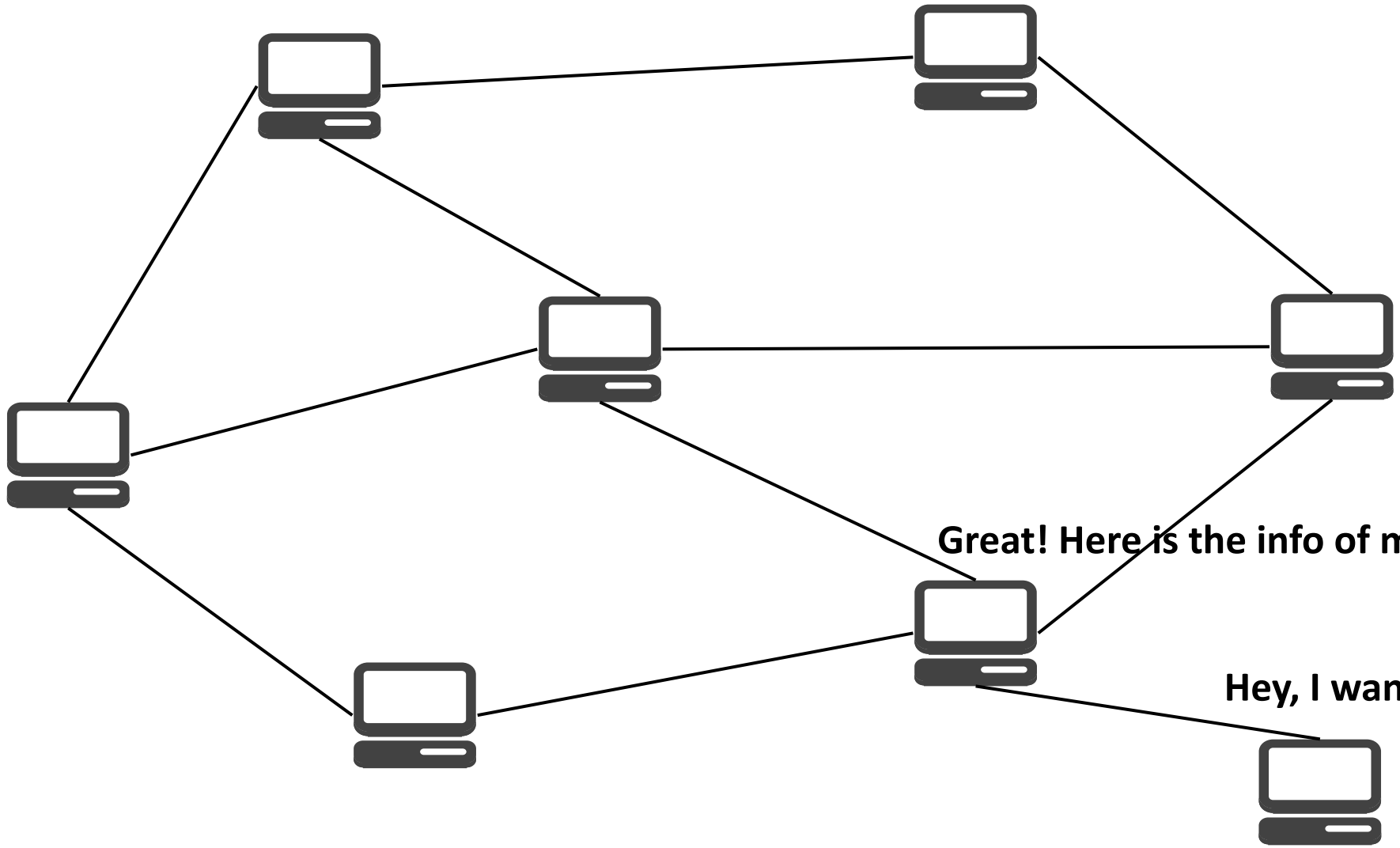
P2P Network in Blockchain



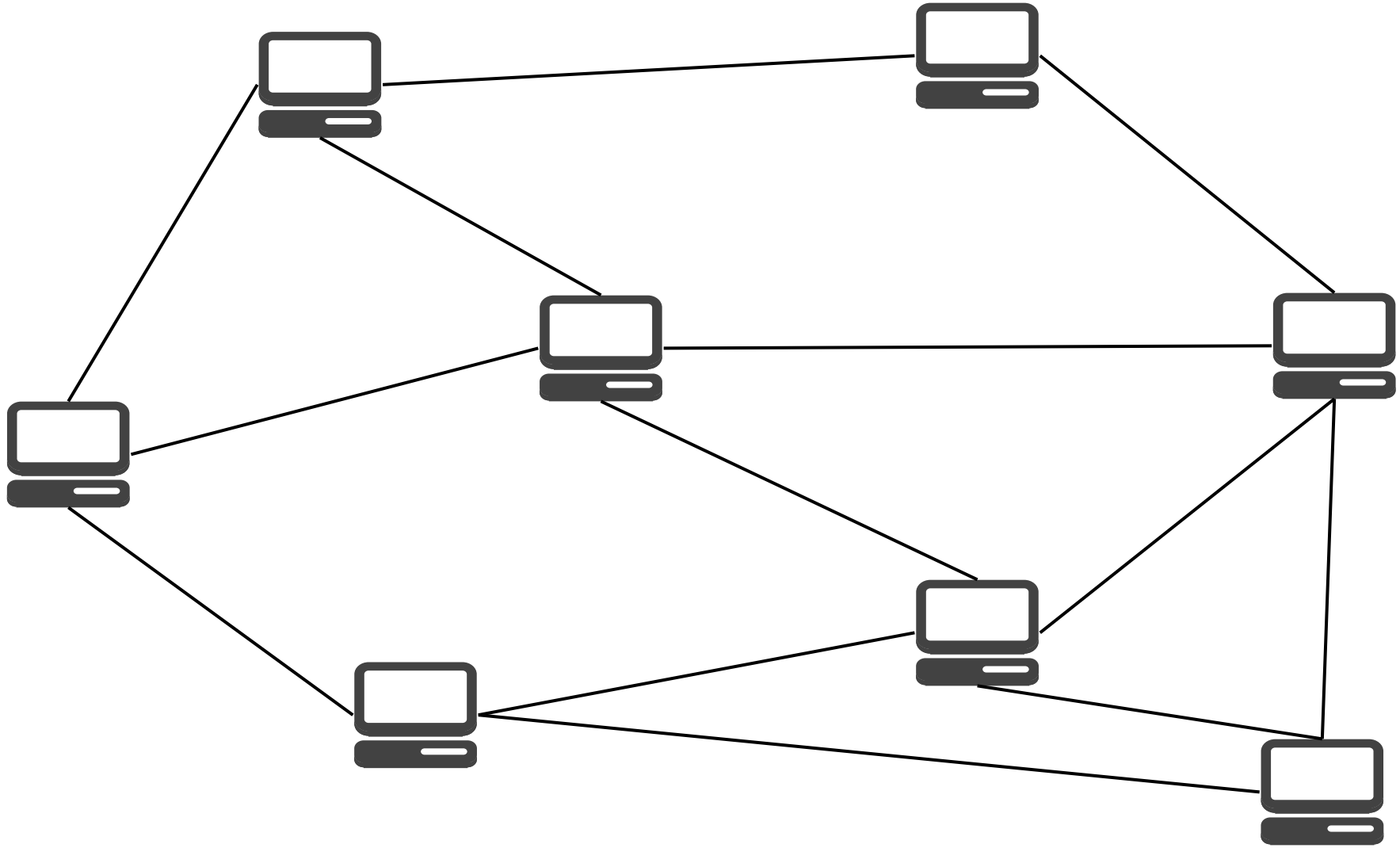
P2P Network in Blockchain



New Node Join



New Node Join



Defense against DoS Attacks

- Attackers send excessive amount of messages to flood the Network

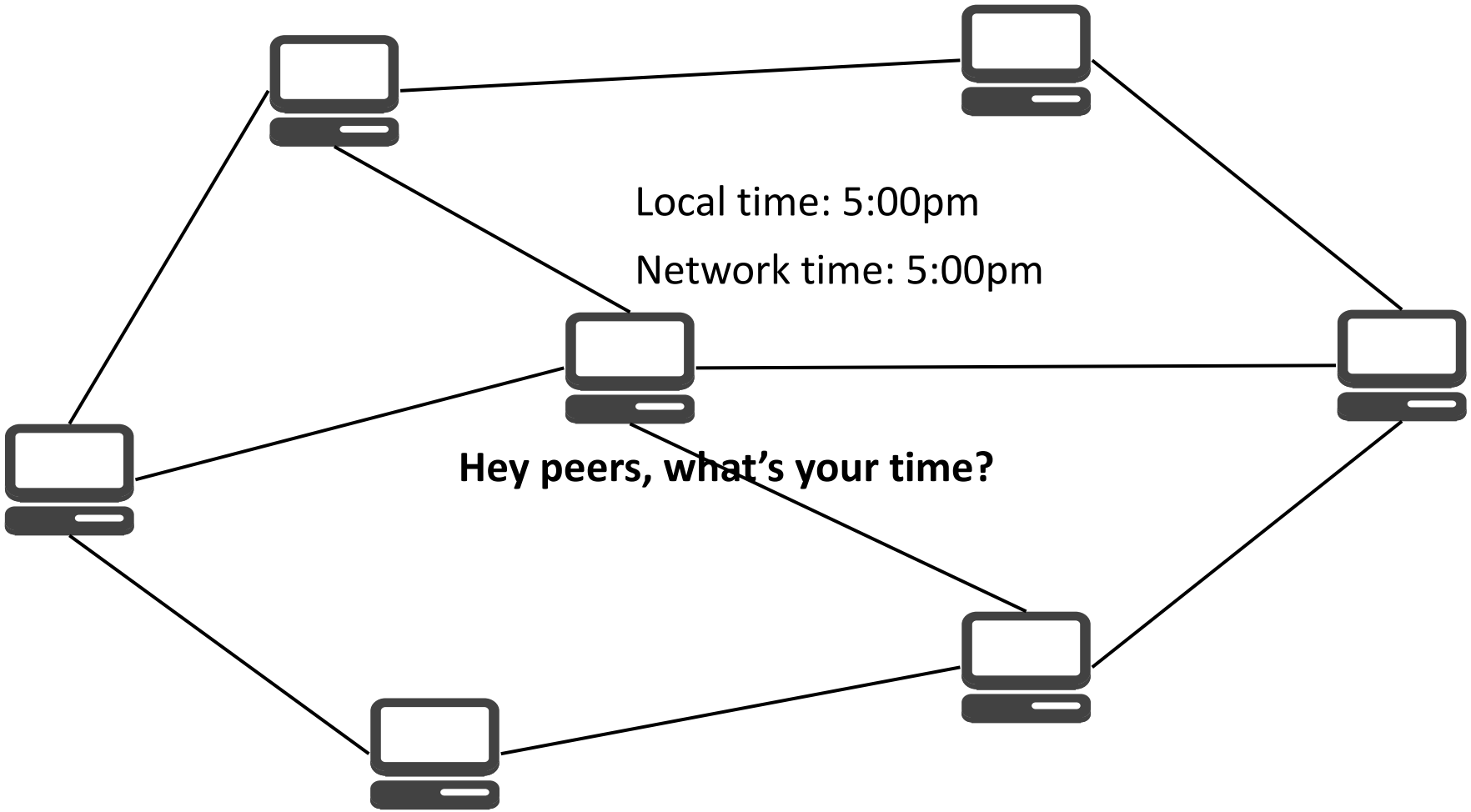
- Check validity of transactions / blocks before relaying
- Do not relay duplicate transactions / blocks
  - Remember hash ids of existing blocks and all pending transactions
  - Check whether a block/transaction id is duplicate
- Drop peers that constantly send invalid / duplicate messages

Network Adjusted Timestamp

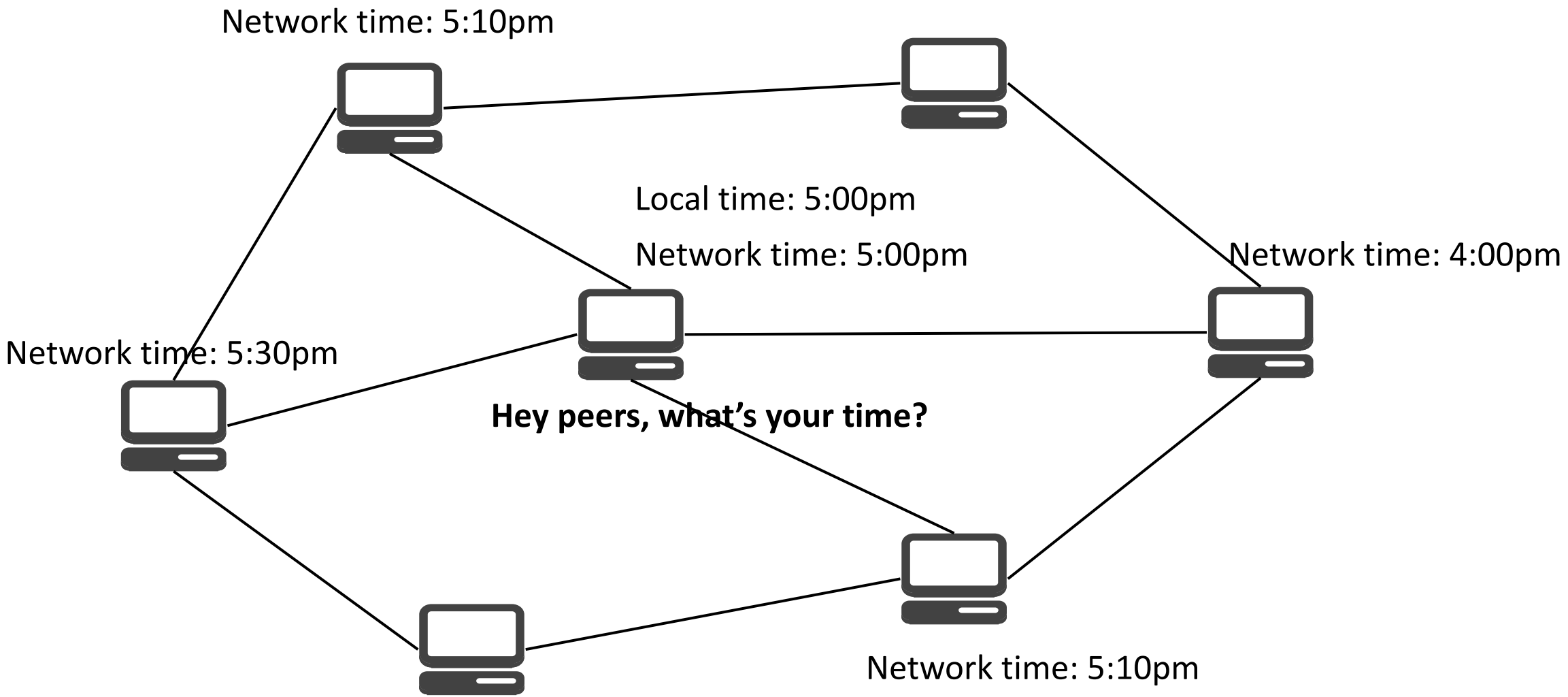
- **Problem**: How to obtain a synchronized time in Blockchain
  - Timestamp each block/transaction
  - Adjust PoW difficulty based on timestamp (will talk about later)

- **A Solution**: Rely on centralized time service (e.g., time.apple.com)
- **Bitcoin Solution**: Use P2P Network to synchronize time between nodes
  - No single point of failure
  - Remain working even if centralized time service is down

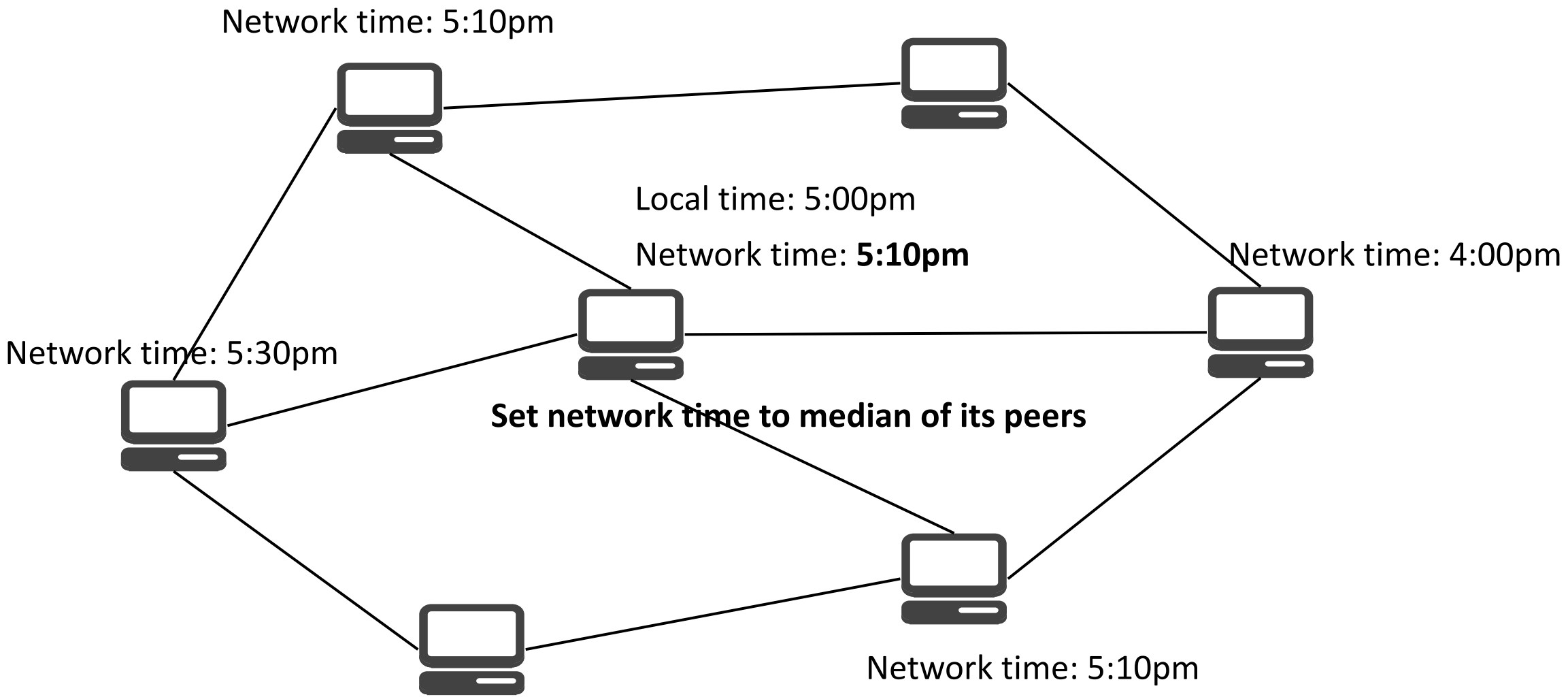
Network Adjusted Time



Network Adjusted Time



Network Adjusted Time

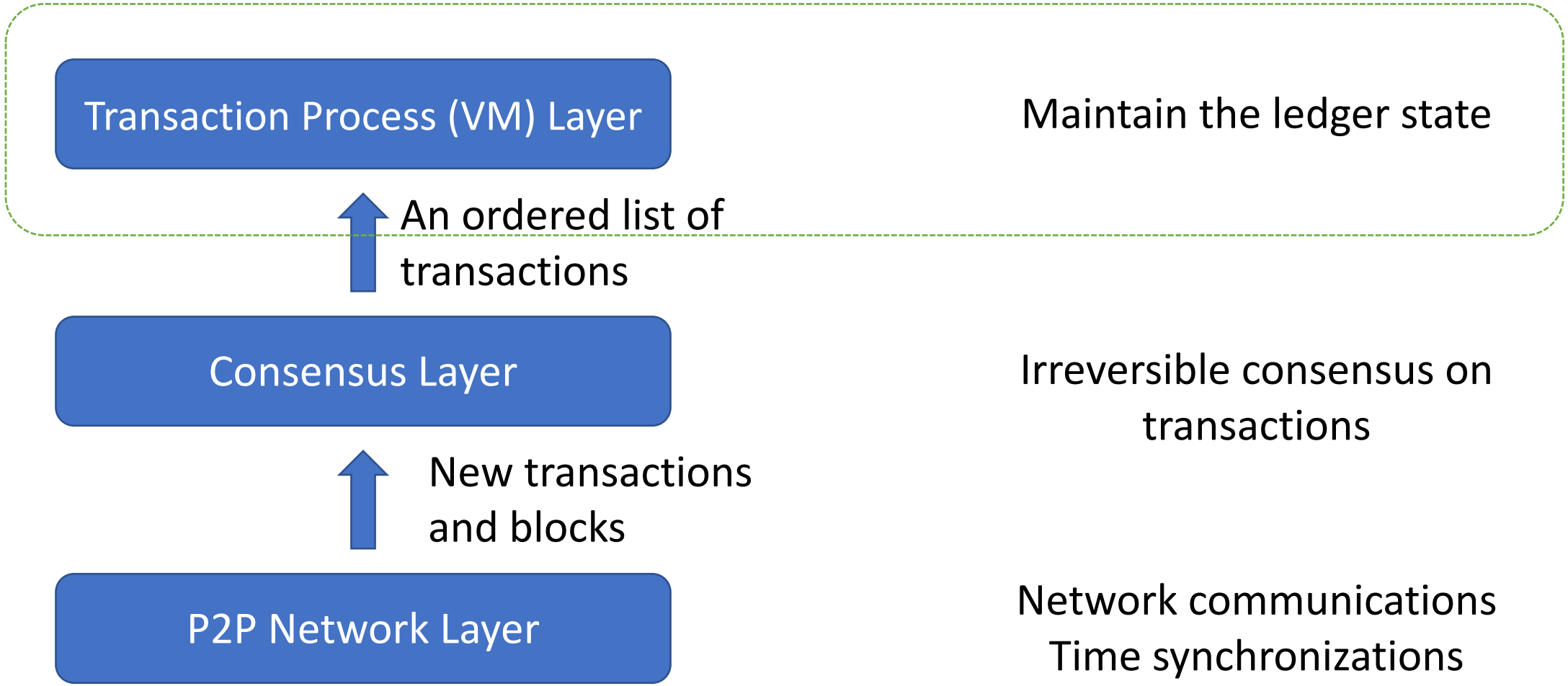


Network Adjusted Time

- Each node collects timestamps from all of its peers periodically
- Set its own network adjusted time to the median of its peers
- The largest difference between network adjusted time and local time can be 70 minutes
  - If the difference is larger than 70 minutes, adjust 70 minutes only

- Is it possible to attack this protocol and control timestamps?
- Why use median not average?

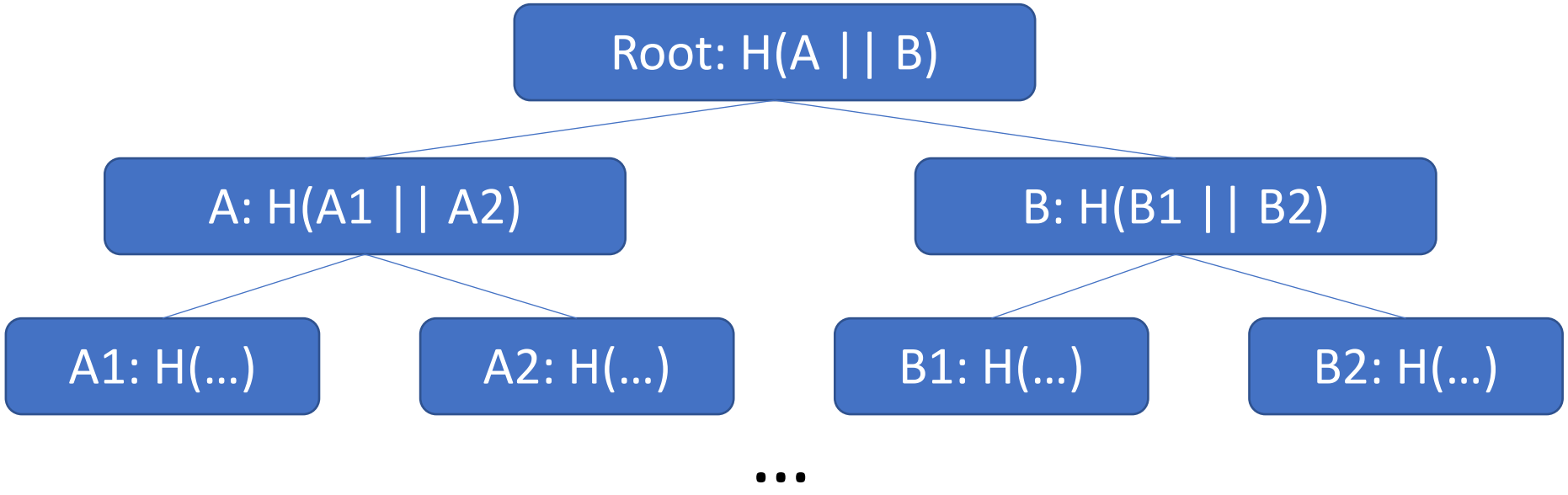
Layers in Blockchain Systems



How Information in Blockchain is Organized?

Authenticated Data Structure: Merkel Tree

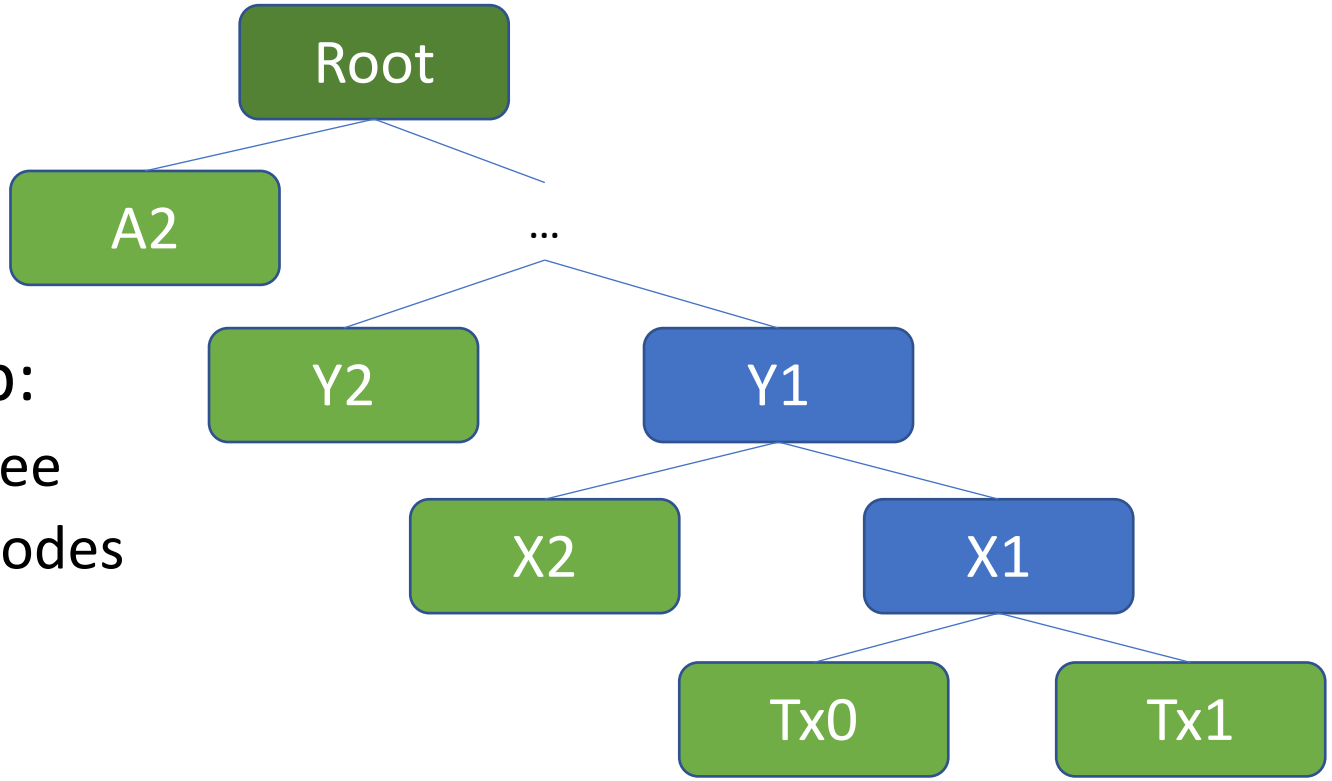
- Each node stores one-way crypto-hash of its children



- Leaf nodes store data
  - The root (merkel root) ensures the data integrity
  - O(Log n) proof for authenticated read

Merkel Tree: Authenticated Read

- Suppose transactions stored in a Merkel tree.
- Alice knows the whole tree. Bob only knows the root



- Alice proves to Bob:
  - Tx0 exists in the tree
  - Only send green nodes
- Why?

Applications of Merkel Tree in Blockchains

- Transactions in a block
  - Merkel root stored in the header
- Unspent transactions at a block height
  - In Bitcoin, Unspent transactions defines the current account state
- Account state in Ethereum
  - Use a special Merkel tree called Merkel Patricia Tree

- **Key Advantage**: Enable light nodes that only store block header to reliably retrieve information from full nodes with proof
  - A light node does not need to trust full nodes to interact with the blockchain

How Blockchain States are Represented?

Account State in Blockchain

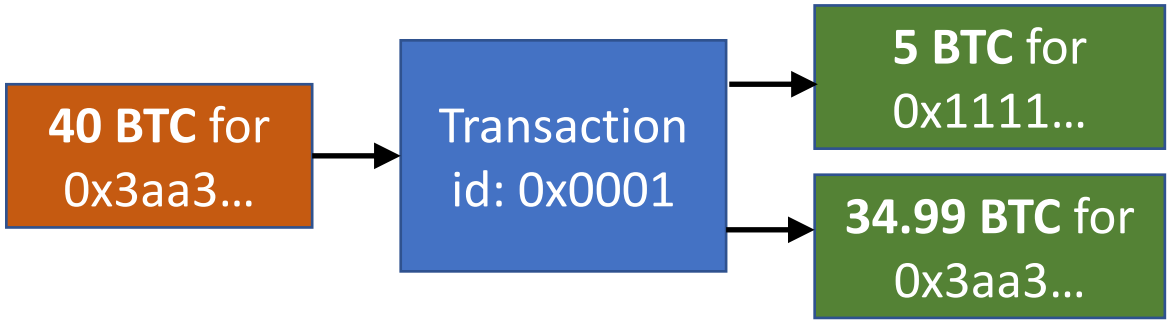
- An address to balance mapping, for example:

Address	Balance
0x2222…	10
0x1111…	20
…	…

- Replay Attack:
  - Alice sign and send ”Alice will pay Bob 1BTC”
  - Bob recorded this signed message
  - One year later, Bob resend recorded message to steal another 1BTC
- **Caveat**: Need to remember all history transactions

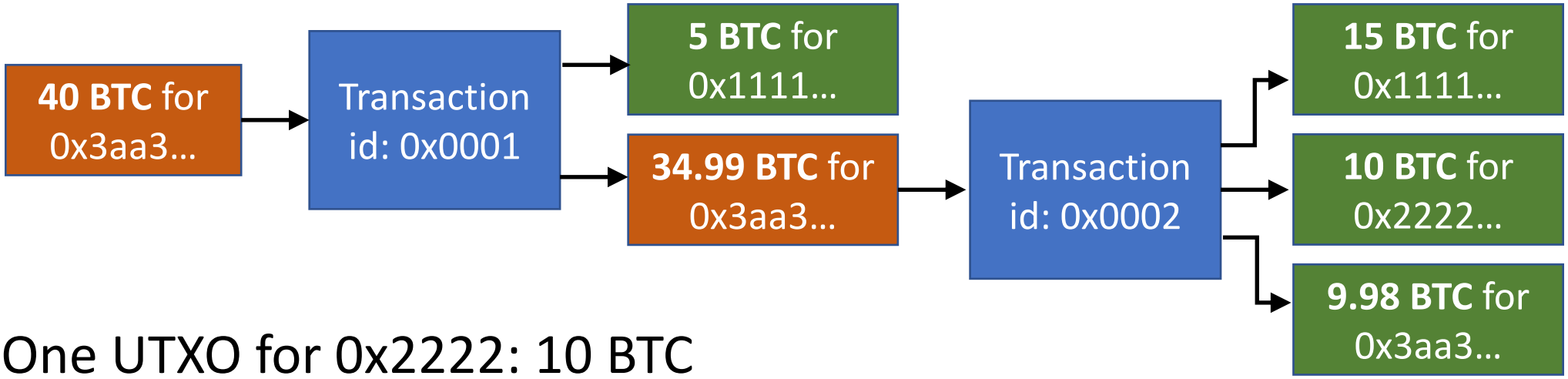
Bitcoin: Unspent Transaction Output (UTXO)

- Each transaction can have multiple inputs and outputs:



Bitcoin: Unspent Transaction Output (UTXO)

- Each transaction can have multiple inputs and outputs:



- One UTXO for 0x2222: 10 BTC
- Two UTXOs for 0x1111: 5 BTC + 15 BTC
- Transaction fees are implicit
- Only need to remember UTXOs now

Discussion: Pros and Cons of UTXO Model

- Simplified conflict definition:
  - Two transactions should not share the same UTXO
- Parallel transaction processing
- Encourage the usage of different addresses: better anonymity

- Counter-intuitive
- Hard to build smart contracts on top of it
  - Will cover more when we talk about Ethereum

How to Process Transactions?

Verify a Transaction in Bitcoin

- To spend an unspent transaction, sender provides:
  - Public key
  - Signature generated with the private key
- To verify the input of an unspent transaction:
  - Check the public key matches the sender's address
  - Check the signature is correct

- Bitcoin defines a stack-based script language to check transactions

Bitcoin Scripts

- The locking script defines how to spend a UTXO
- Sender provides an unlocking script
- Concatenate two scripts to run
- The script passes if the execution ends with no error and no input left
  - Reject the transaction if the execution fails

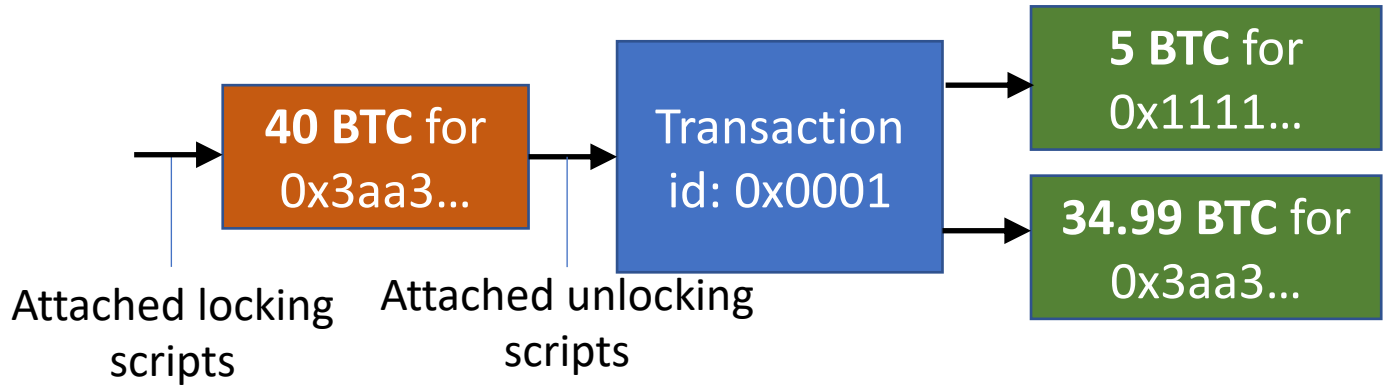
<signature>
<public key>

Sender unlocking scripts

OP_DUP
OP_HASH160
<bitcoin address (hash of public key)>
OP_EQUALVERIFY
OP_CHECKSIG

The standard
UTXO locking script

Bitcoin Scripts



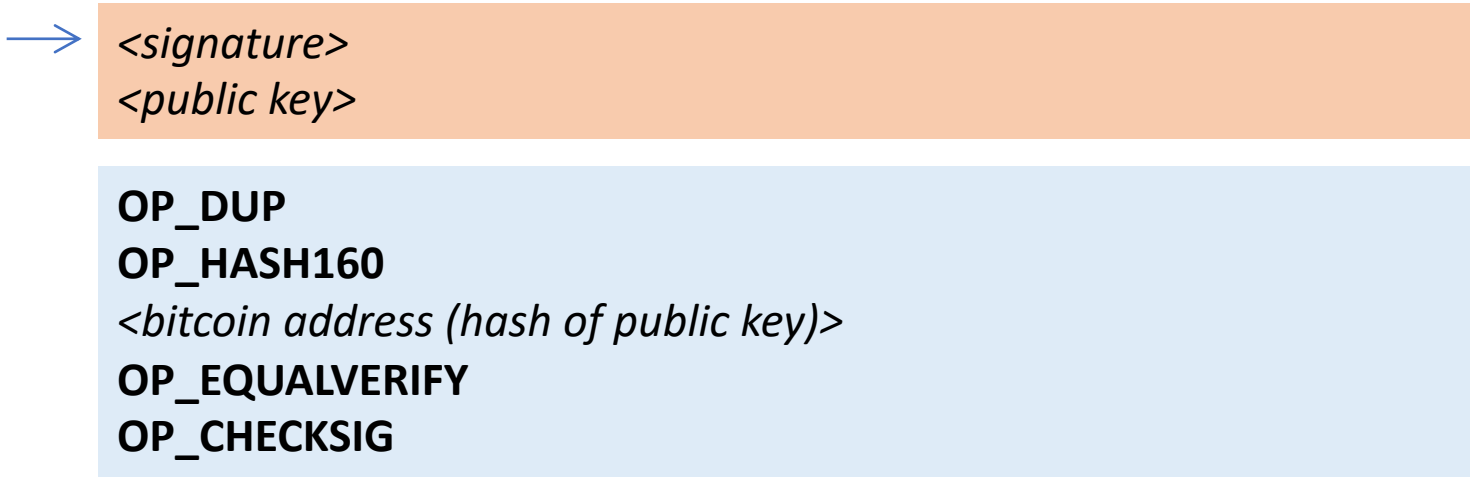
<signature>
<public key>

Sender unlocking scripts

OP_DUP
OP_HASH160
<bitcoin address (hash of public key)>
OP_EQUALVERIFY
OP_CHECKSIG

The standard
UTXO locking script

Example: Bitcoin Script Execution



Execution Stack

For data token, just push it into the stack!

Example: Bitcoin Script Execution

<signature>
<public key>

OP_DUP
OP_HASH160
<bitcoin address (hash of public key)>
OP_EQUALVERIFY
OP_CHECKSIG

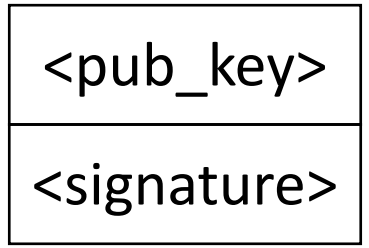
<signature>

Execution Stack

Example: Bitcoin Script Execution

<signature>
<public key>

→ **OP_DUP**
OP_HASH160
<bitcoin address (hash of public key)>
OP_EQUALVERIFY
OP_CHECKSIG



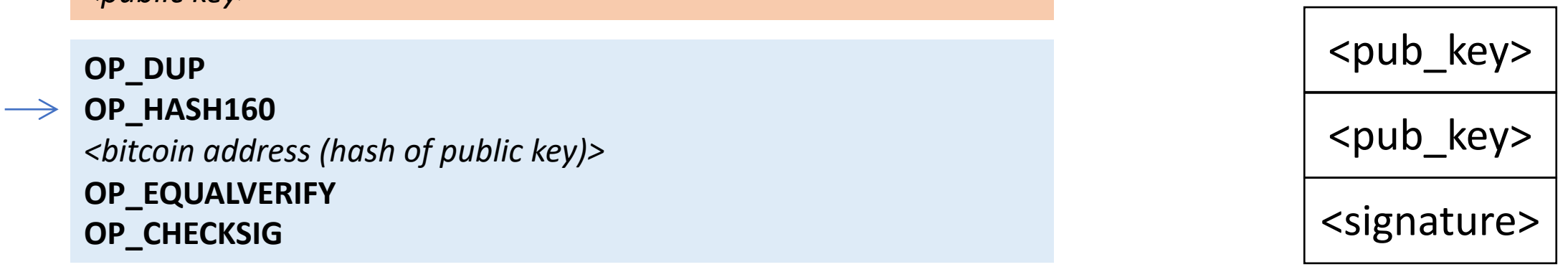
Execution Stack

OP_DUP duplicates the top entry of the stack

Example: Bitcoin Script Execution

<signature>
<public key>

OP_DUP
OP_HASH160
<bitcoin address (hash of public key)>
OP_EQUALVERIFY
OP_CHECKSIG



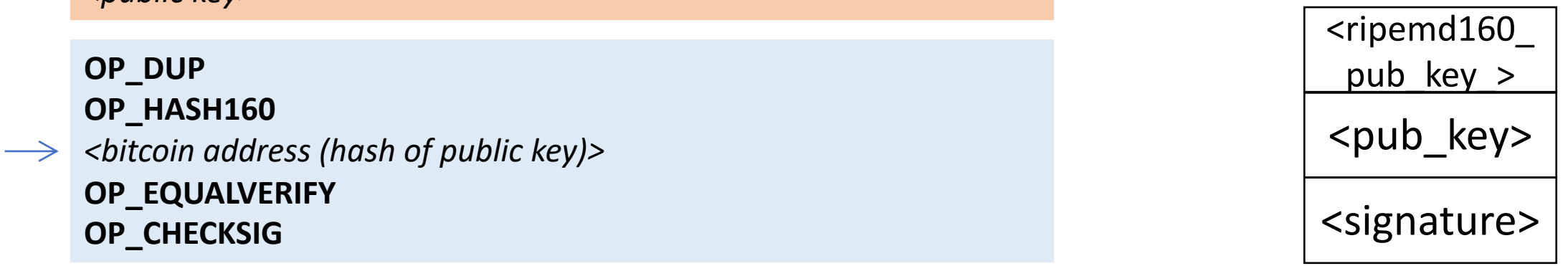
Execution Stack

OP_HASH160 computes RIPEMD160 of the top item and replaces it

Example: Bitcoin Script Execution

<signature>
<public key>

OP_DUP
OP_HASH160
<bitcoin address (hash of public key)>
OP_EQUALVERIFY
OP_CHECKSIG

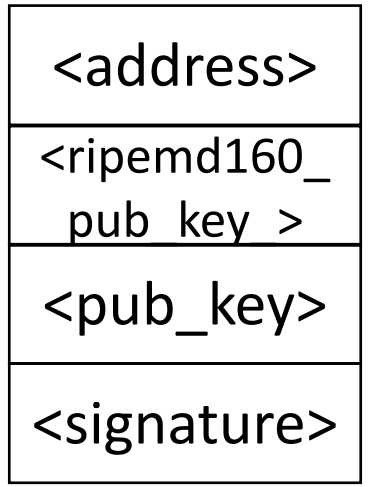


Execution Stack

Example: Bitcoin Script Execution

<signature>
<public key>

OP_DUP
OP_HASH160
<bitcoin address (hash of public key)>
→ **OP_EQUALVERIFY**
OP_CHECKSIG



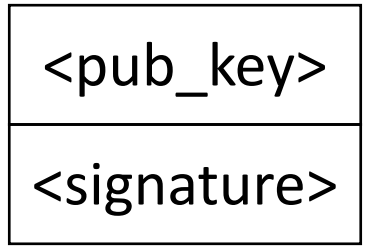
Execution Stack

OP_EQUIALVERIFY pops the top two items and compares them, terminate with error if they do not equal

Example: Bitcoin Script Execution

<signature>
<public key>

OP_DUP
OP_HASH160
<bitcoin address (hash of public key)>
OP_EQUALVERIFY
→ **OP_CHECKSIG**



Execution Stack

OP_CHECKSIG pops the top two items and checks whether they correspond to a correct public_key-signature pair. If not, terminate with error.

Example: Bitcoin Script Execution

<signature>
<public key>

OP_DUP
OP_HASH160
<bitcoin address (hash of public key)>
OP_EQUALVERIFY
OP_CHECKSIG

Execution Stack

Execution ends without error. The unspent fund is unlocked successfully!

Bitcoin Script Language

- Support arithmetic and cryptographic primitives
- Stack-based language with more than one hundred opcodes
  - Not Turing complete though
- Why define a language for transaction validation?
- To support complicated transaction logic:
  - Multi-signature transactions
  - Hash-locked transactions
  - Time-lock
- Ethereum instead defines a more powerful Turing complete language