BUILDING YOUR OWN BLOCKCHAIN FROM SCRATCH

Yen-Chi Chen

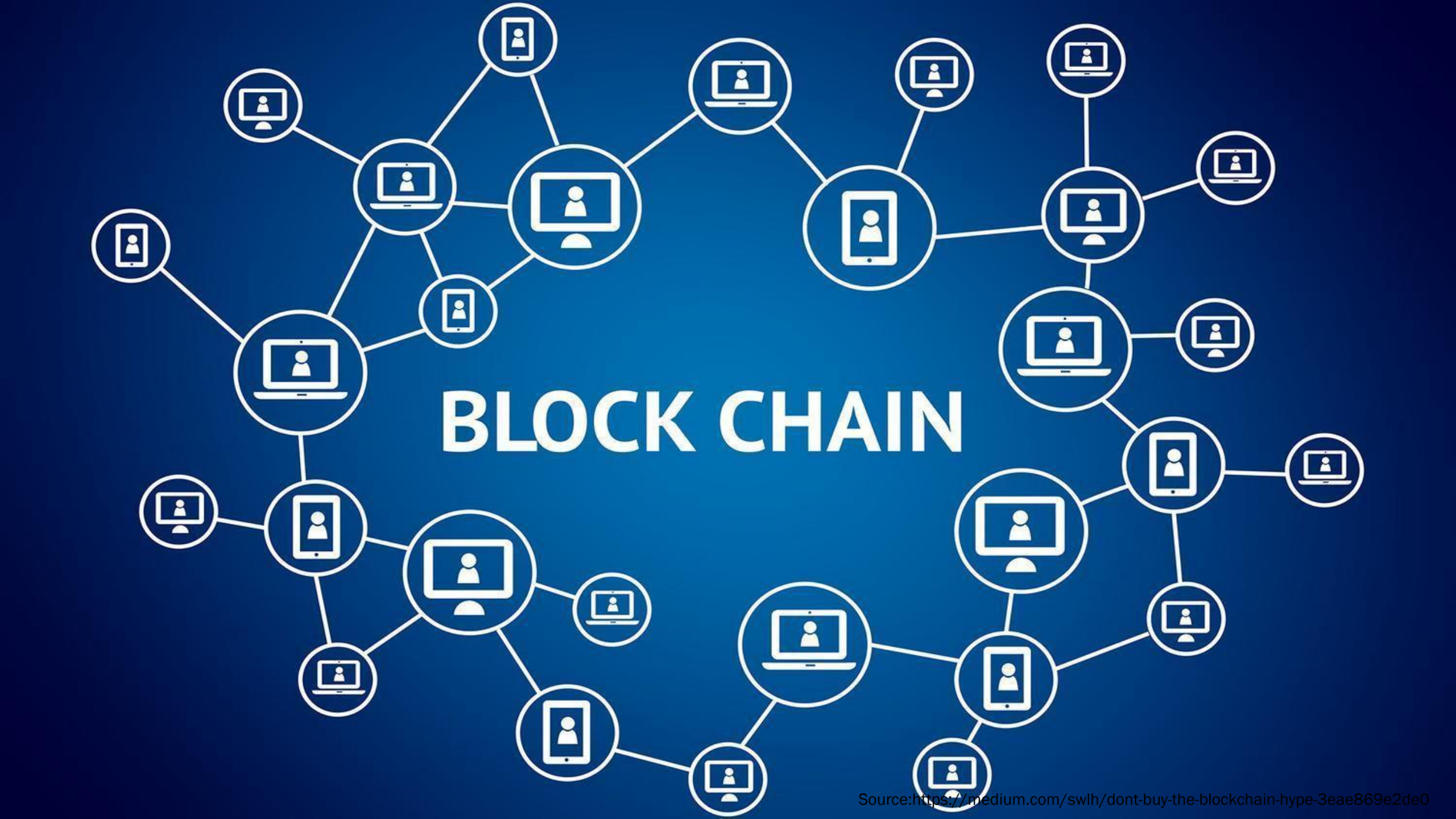
2018-12-14 and 21

- 陳彥奇
- 台大物理系畢業
- 興趣:玩電腦
- 專長:玩電腦

Links

- Feedback
- https://ppt.cc/fxPzpx
- Code:
- https://ppt.cc/fnkkdx

INTRODUCTION



Source:https://medium.com/swlh/dont-buy-the-blockchain-hype-3eae869e2de0

Author Topic: Pizza for bitcoins? (Read 327851 times)

laszlo
Full Member

Activity: 199

Trust: **0**: -0 / +0(0)
Ignore

Pizza for bitcoins?

May 18, 2010, 12:35:20 AM

quote #1

I'll pay 10,000 bitcoins for a couple of pizzas.. like maybe 2 large ones so I have some left over for the next day. I like having left over pizza to nibble on later. You can make the pizza yourself and bring it to my house or order it for me from a delivery place, but what I'm aiming for is getting food delivered in exchange for bitcoins where I don't have to order or prepare it myself, kind of like ordering a 'breakfast platter' at a hotel or something, they just bring you something to eat and you're happy!

I like things like onions, peppers, sausage, mushrooms, tomatoes, pepperoni, etc.. just standard stuff no weird fish topping or anything like that. I also like regular cheese pizzas which may be cheaper to prepare or otherwise acquire.

If you're interested please let me know and we can work out a deal.

Thanks,
Laszlo

Report to moderator

BC: 157fRrqAKrDyGHr1Bx3yDxeMv8Rh45aUet

BTC/USD

Last price:	Daily change:	Today's open:	24h volume:
$ 3463.6	$ 11.2	$ 3452.4	Ƀ 612.94244058

Chart

1m 3m 5m 15m 30m 1h 2h 4h 6h 12h 1d 3d 1w

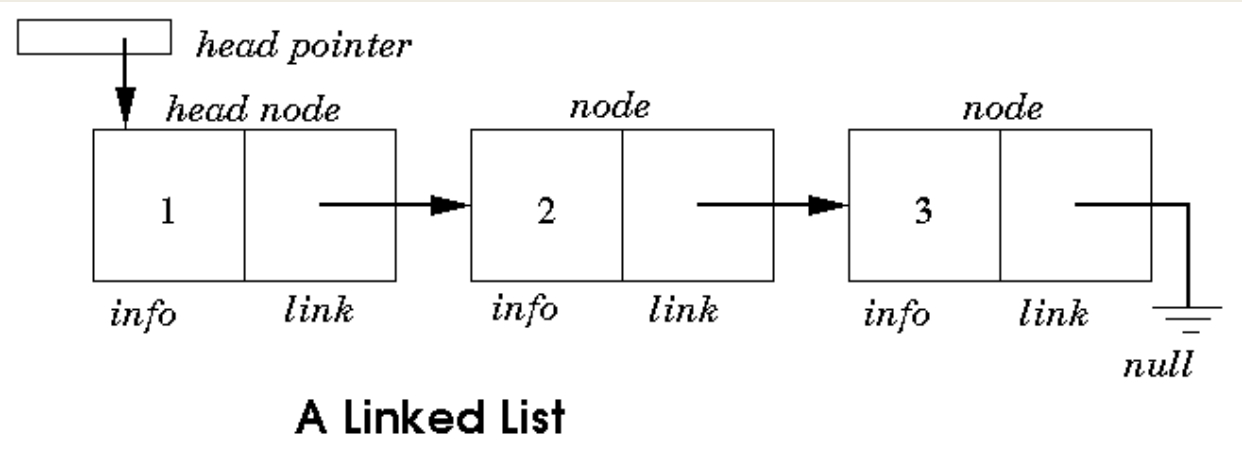
ETH/USD

Last price:	Daily change:	Today's open:	24h volume:
$ 90.68	$ 0.68	$ 90.00	♦ 3613.38165600

Chart

BLOCKCHAINS AND CRYPTOCURRENCY

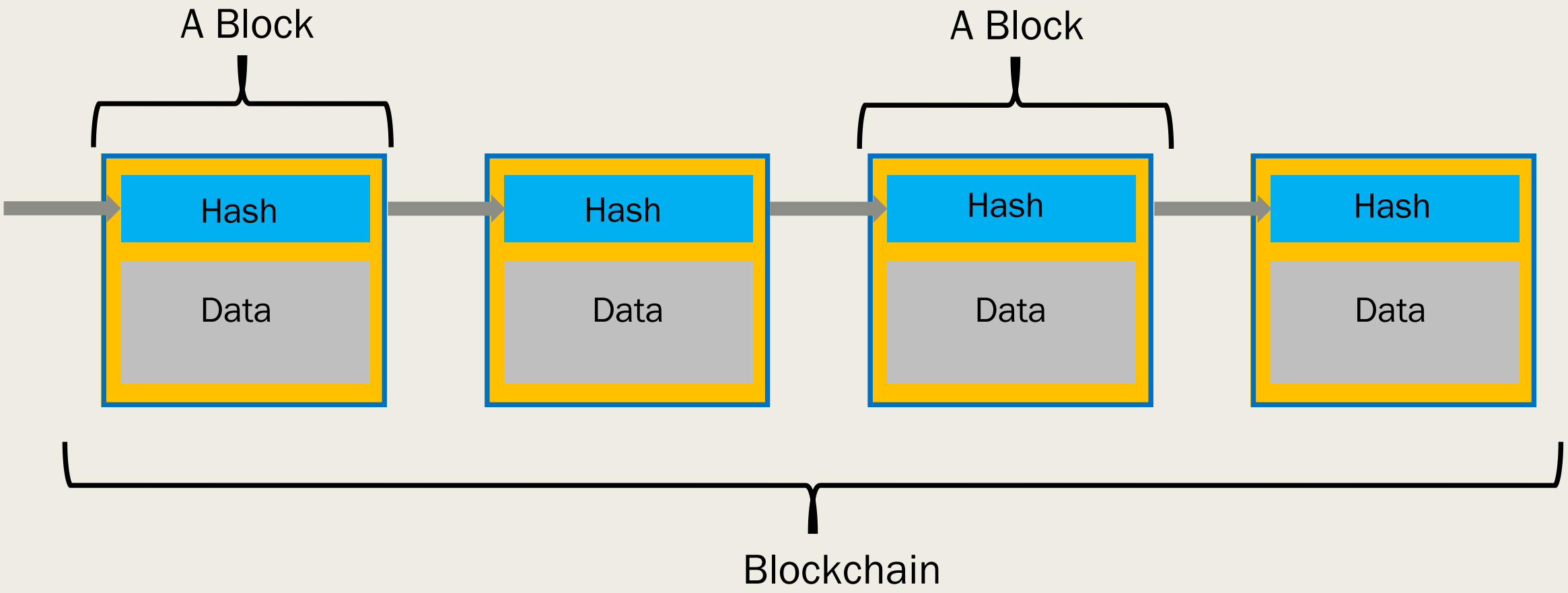
Blockchain



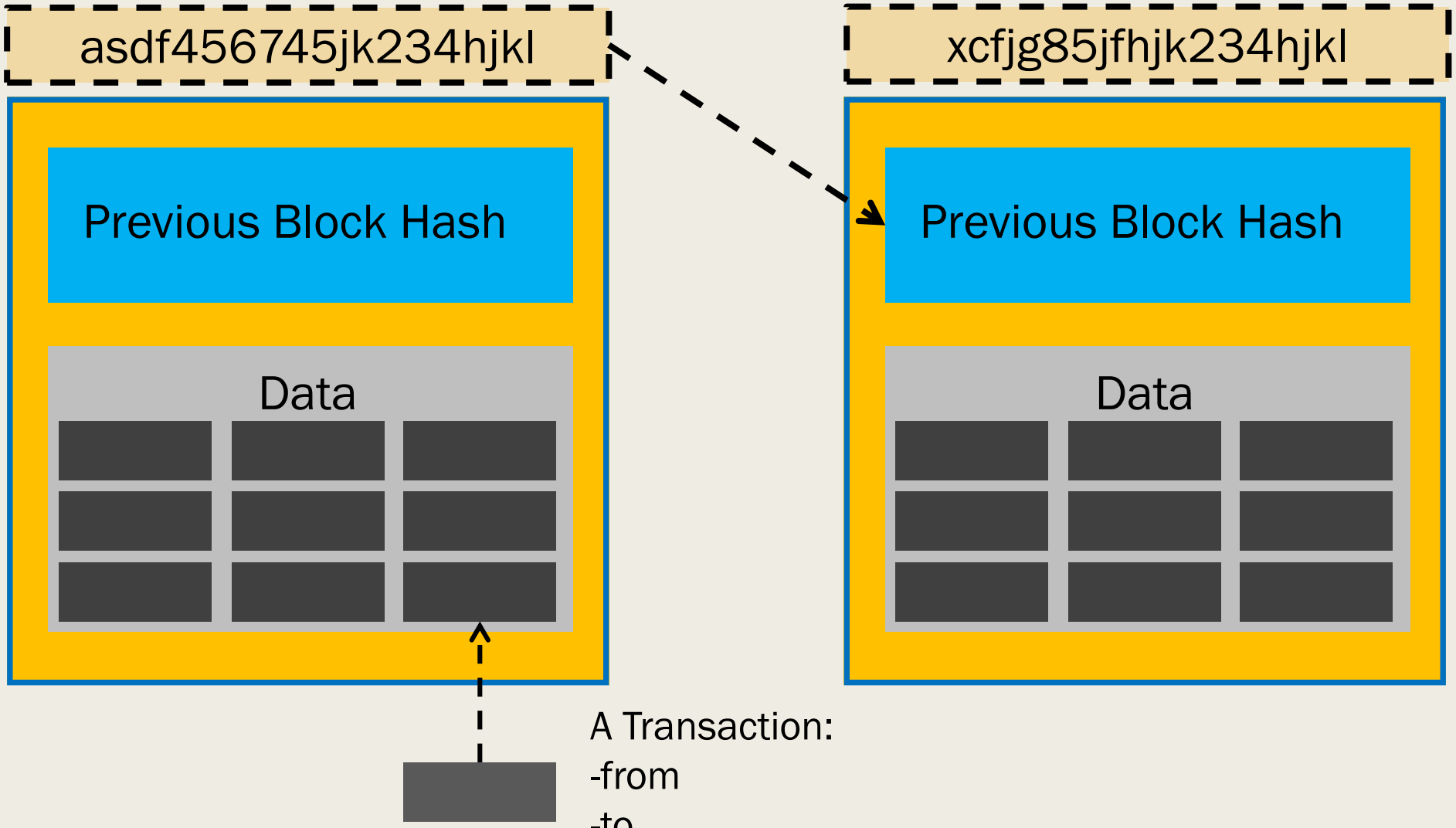
A Linked List

- Linked-list
- Hashed
- Signature
- Distributed storage / multi-copy

Blockchain


A Block
A Block
Hash
Hash
Hash
Hash
Data
Data
Data
Data
Blockchain

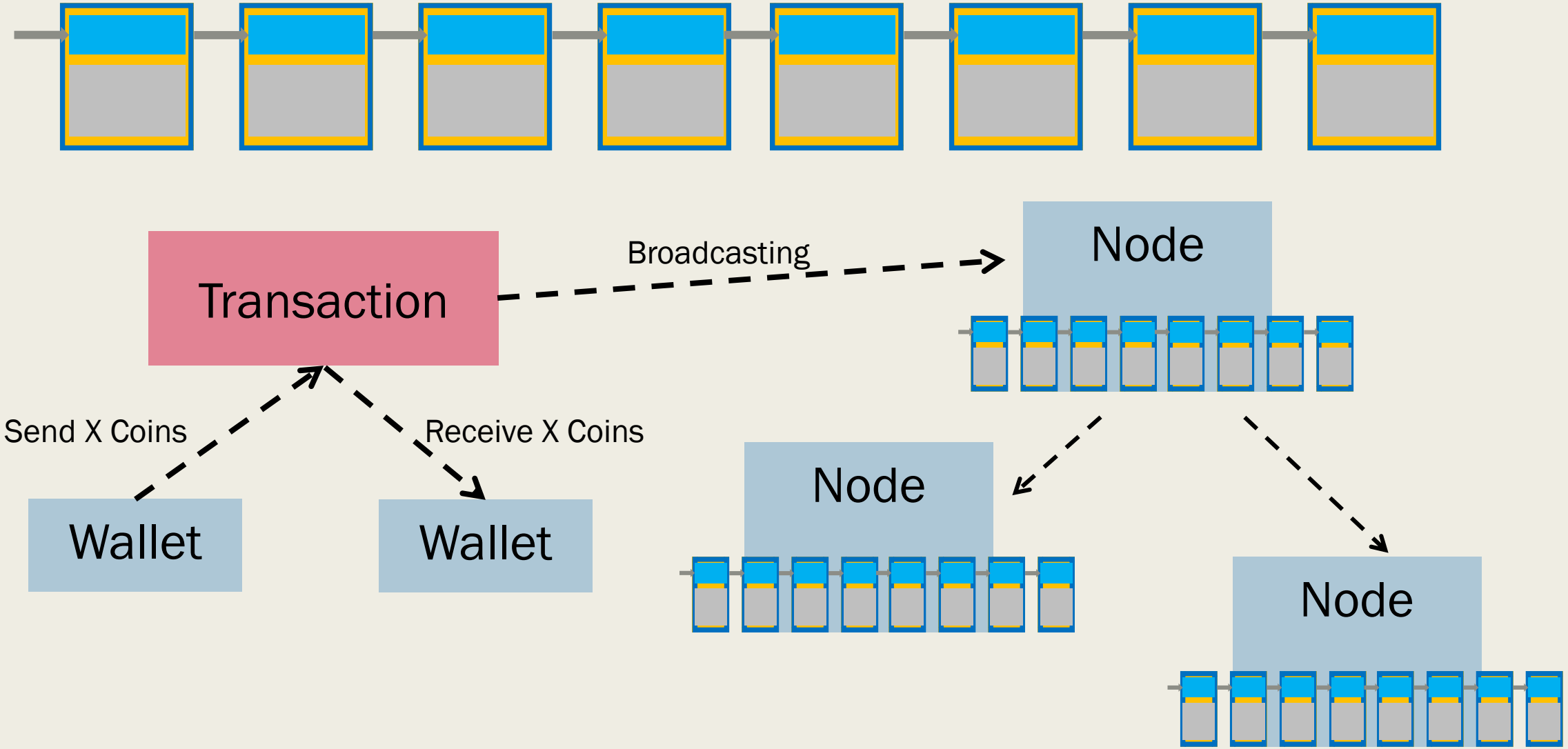
Blockchain


asdf456745jk234hjkl
xcfjg85jfhjk234hjkl
Previous Block Hash
Data
Previous Block Hash
Data
A Transaction:
-from
-to
-amount

Cryptocurrency

- Transaction information stored on blockchain
- Distributed ledger

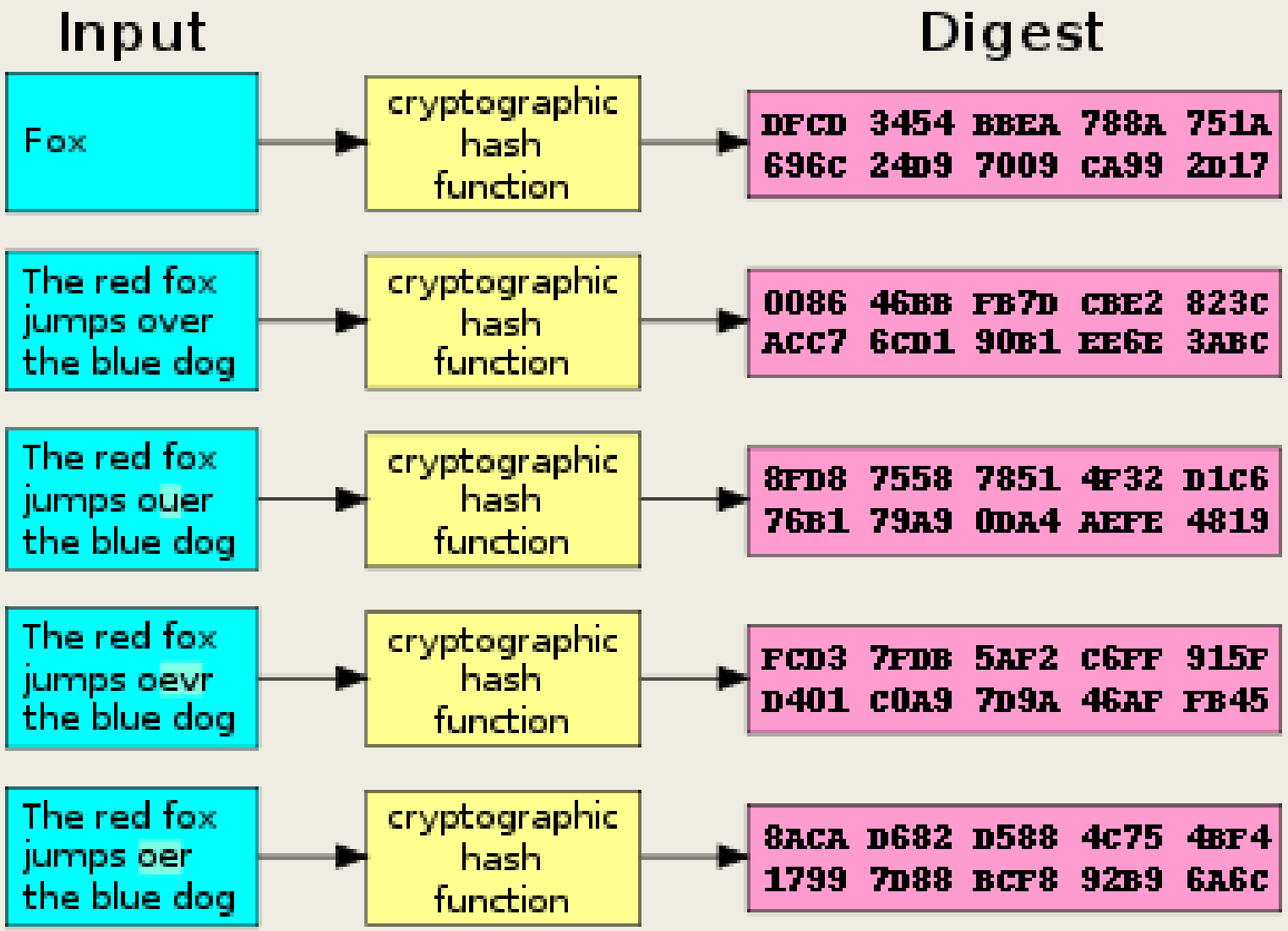
Cryptocurrency and Wallet



“Crypto-”?

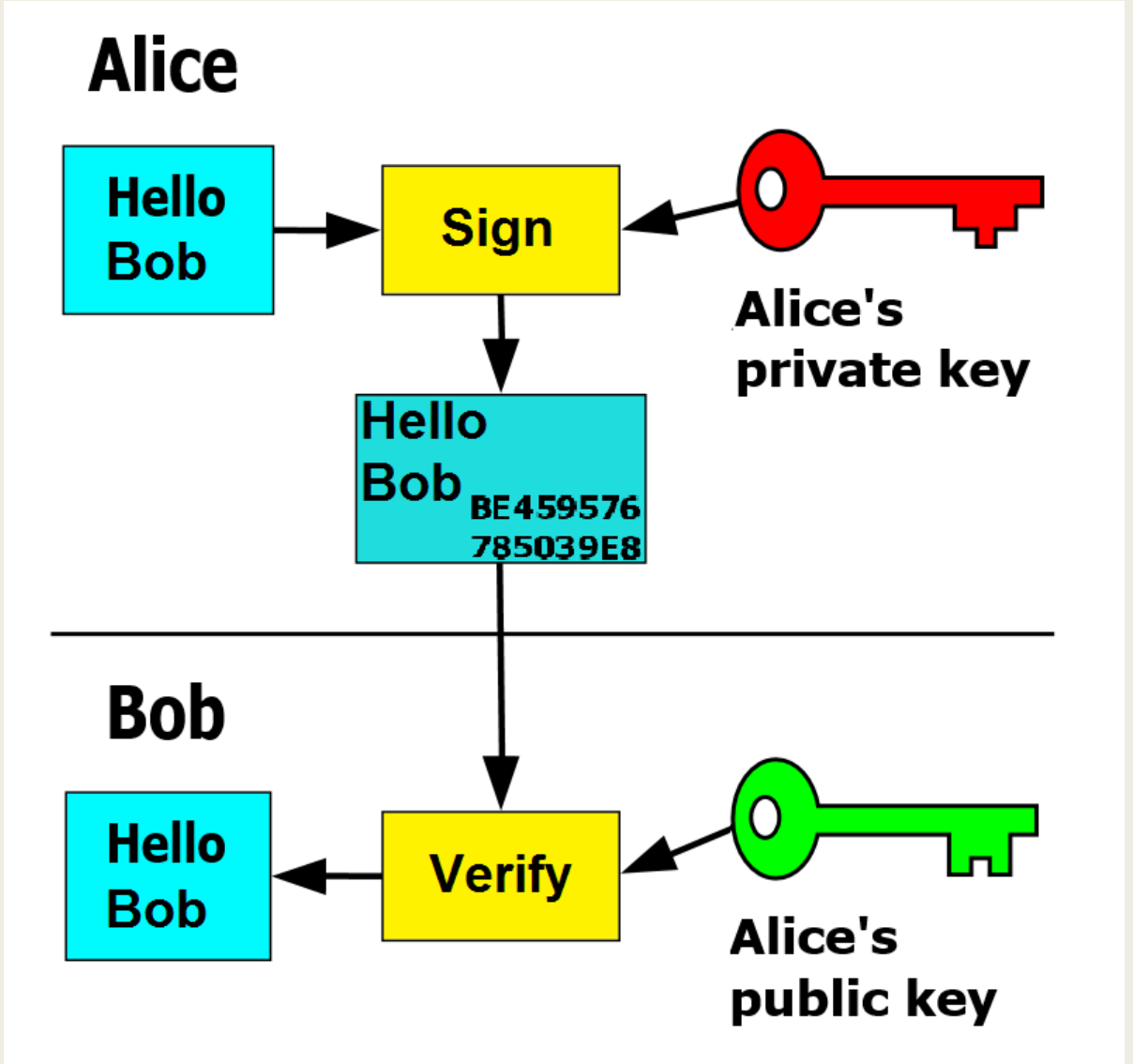
- Cryptography
- Signature
- Hash function
- May be compromised when large-scale quantum computers available

Cryptography-Hash Function



Source: Wikipedia

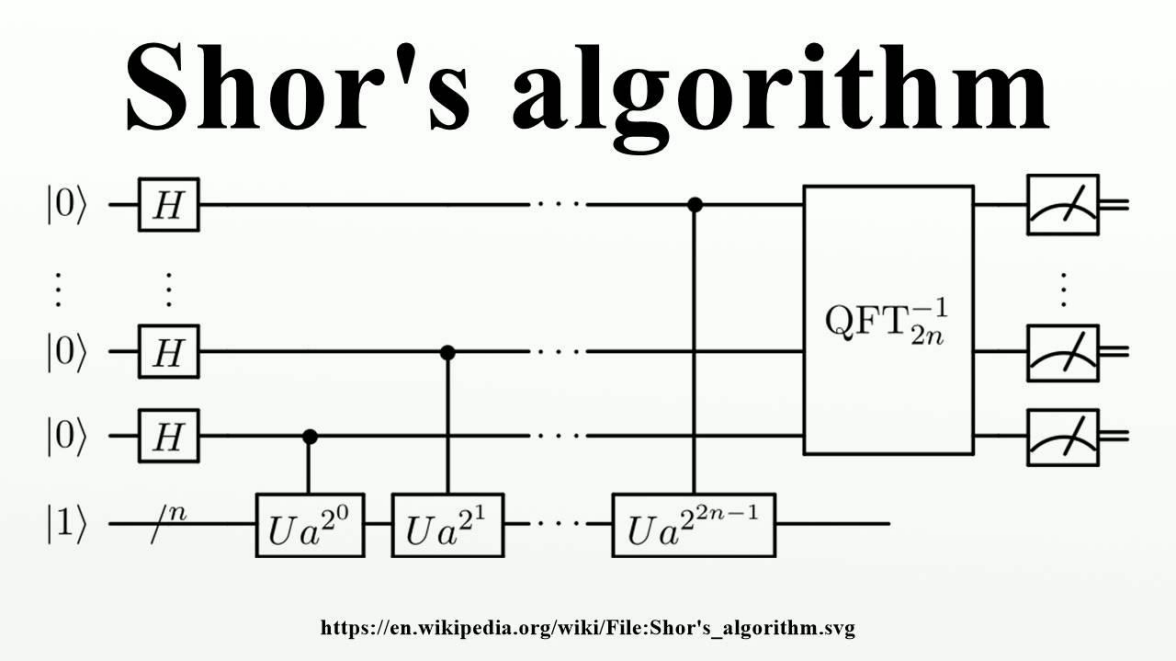
Cryptography-Signature



Source: Wikipedia

Cryptography-Quantum Era

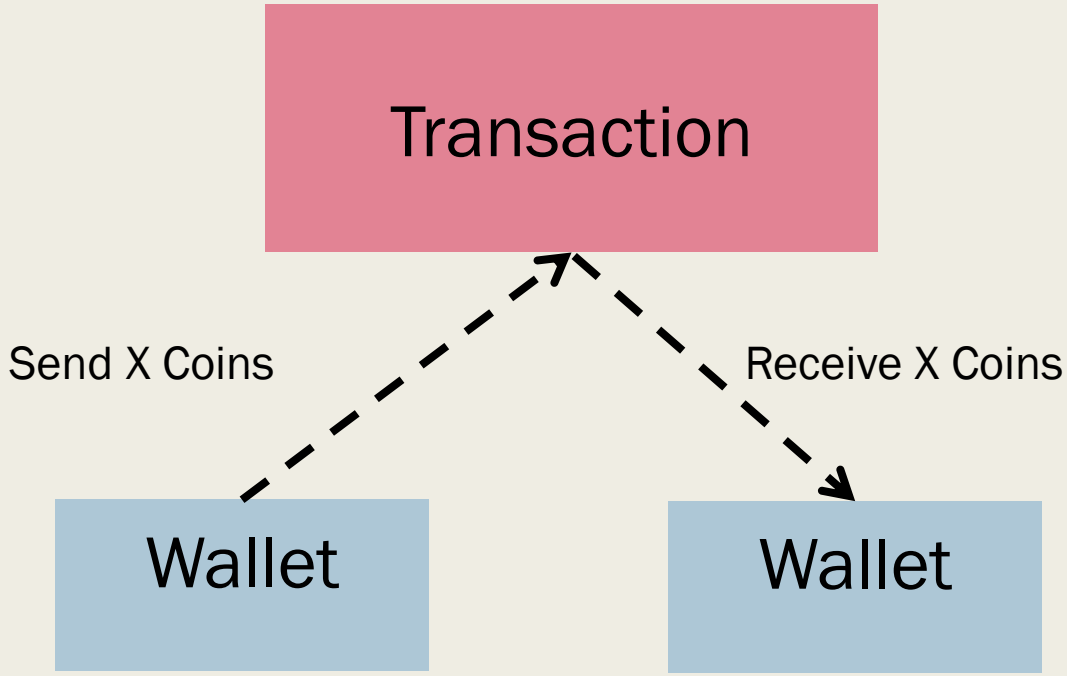
- RSA
- ECC
- Shor’s Algorithm
- >10000 Qubit
- Currently: 50 Qubit



https://en.wikipedia.org/wiki/File:Shor's_algorithm.svg

Transaction

- Modify the data on the blockchain
- Signature by private key
- Verification by public key

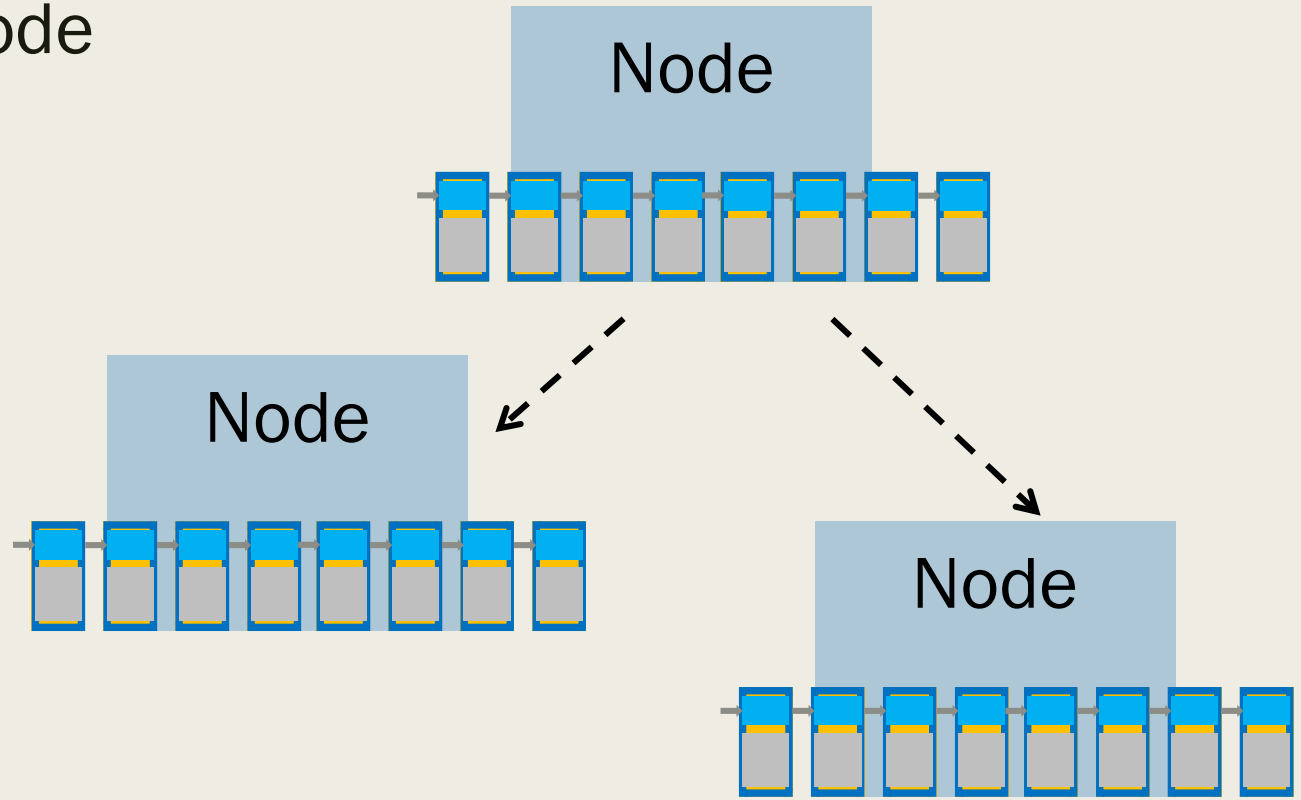


Verification

- Verify the blockchain (previous block hash and POW)
- Verify the transaction (signature)

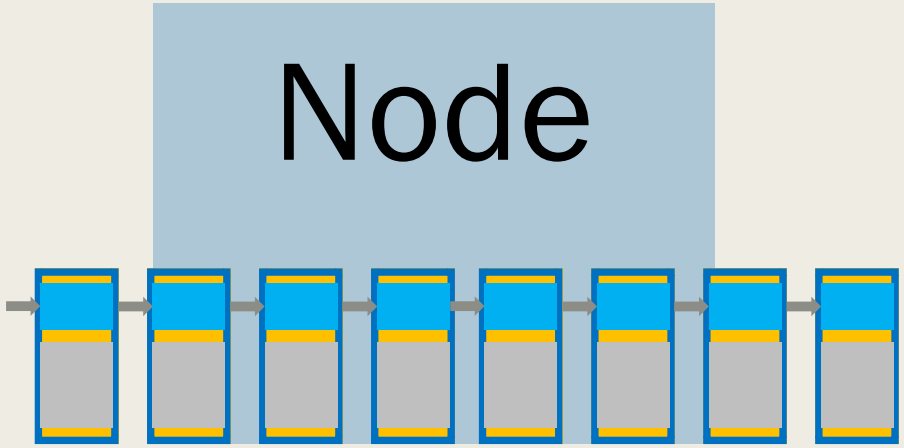
Broadcast

- Broadcast information to the entire network
- Wallet is (in general) a node



Node

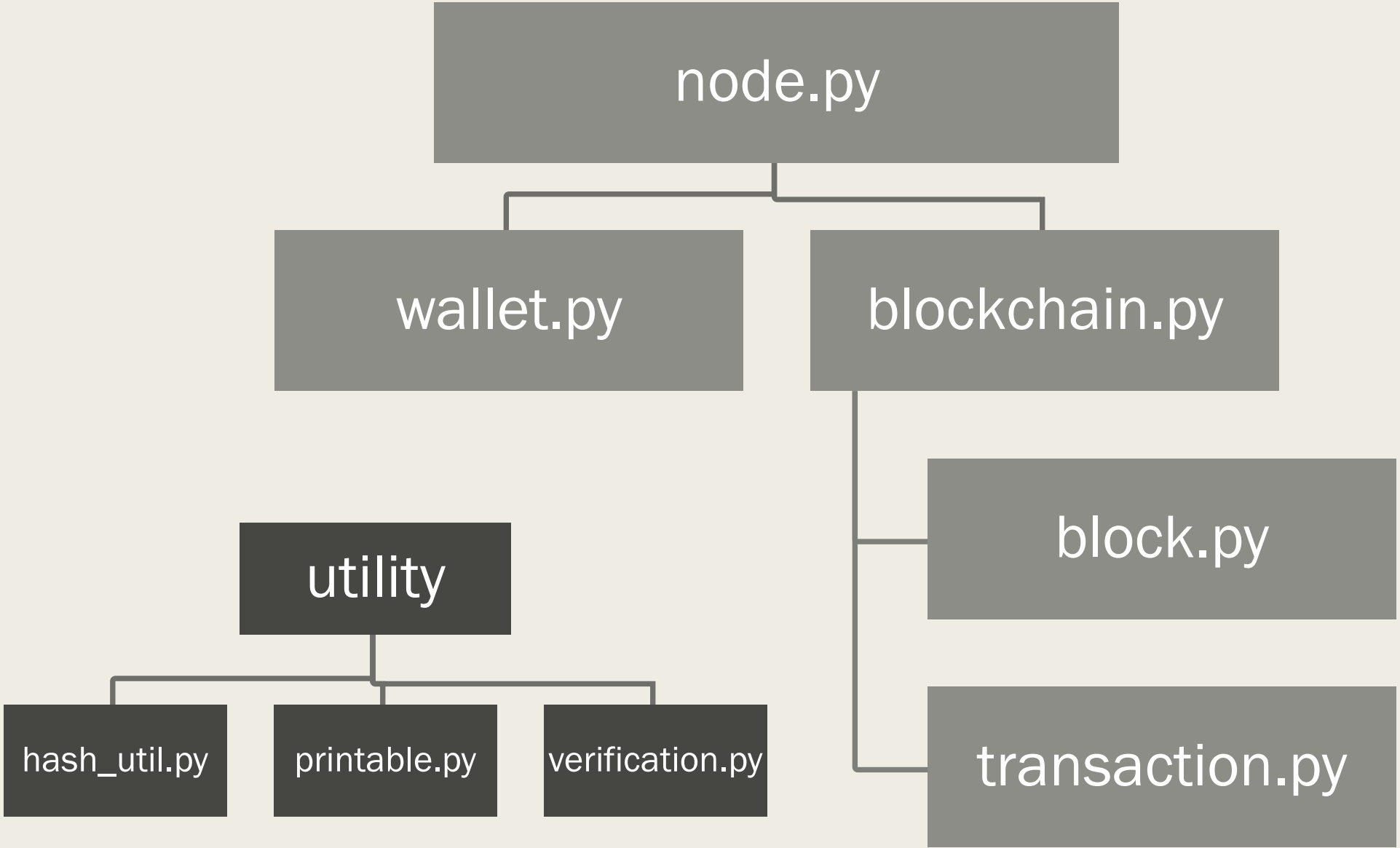
- “Mine” the cryptocurrency => Proof of Work
- Store the blockchain
- Broadcast / Receive blockchain info from peer nodes



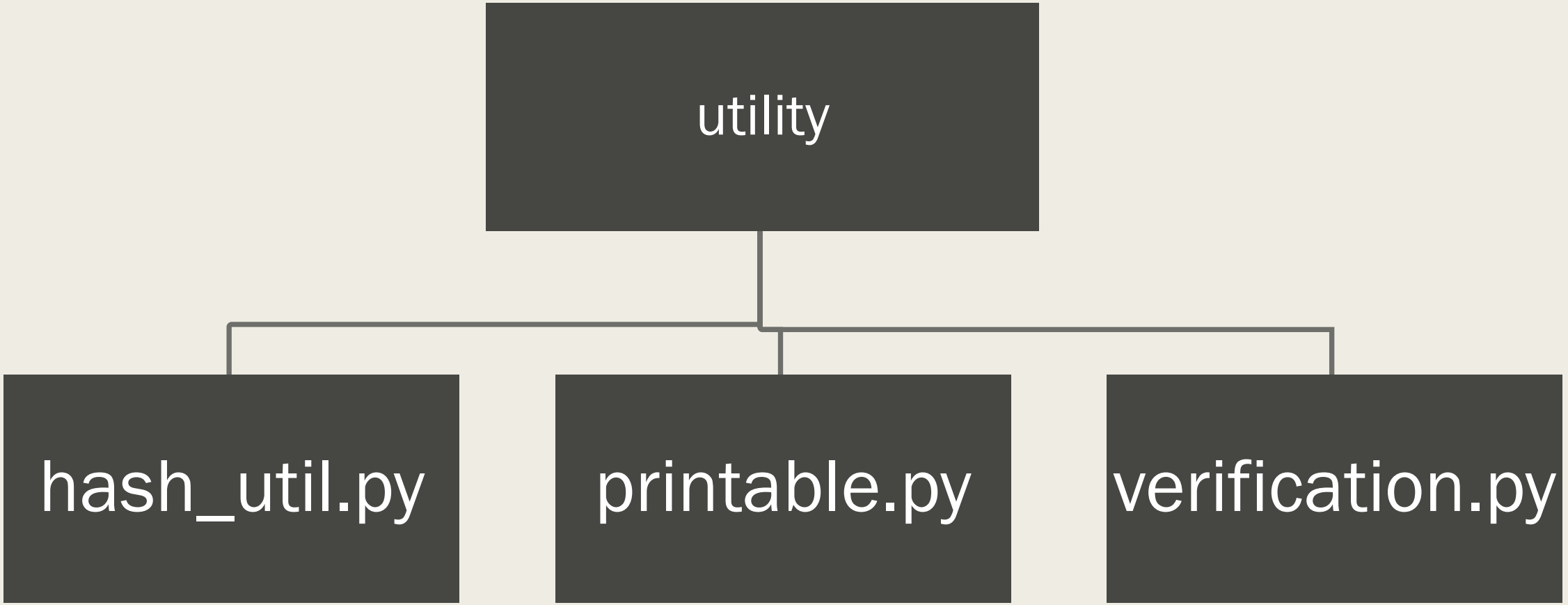
Consensus

- Resolve the conflict
- Longer chain wins
- 51% attacks

Implementation Overview



utility



hash_util.py

hash_util.py

```python
import hashlib as hl
import json

def hash_string_256(string):
    return hl.sha256(string).hexdigest()


def hash_block(block):
    hashable_block = block.__dict__.copy()
    hashable_block["transactions"] = [tx.to_ordered_dict() for tx in hashable_block["transactions"]]
    return hl.sha256(json.dumps(hashable_block, sort_keys = True).encode()).hexdigest()
```

printable.py

printable.py

```
1  class Printable:
2      def __repr__(self):
3          return str(self.__dict__)
```

verification.py

verification.py

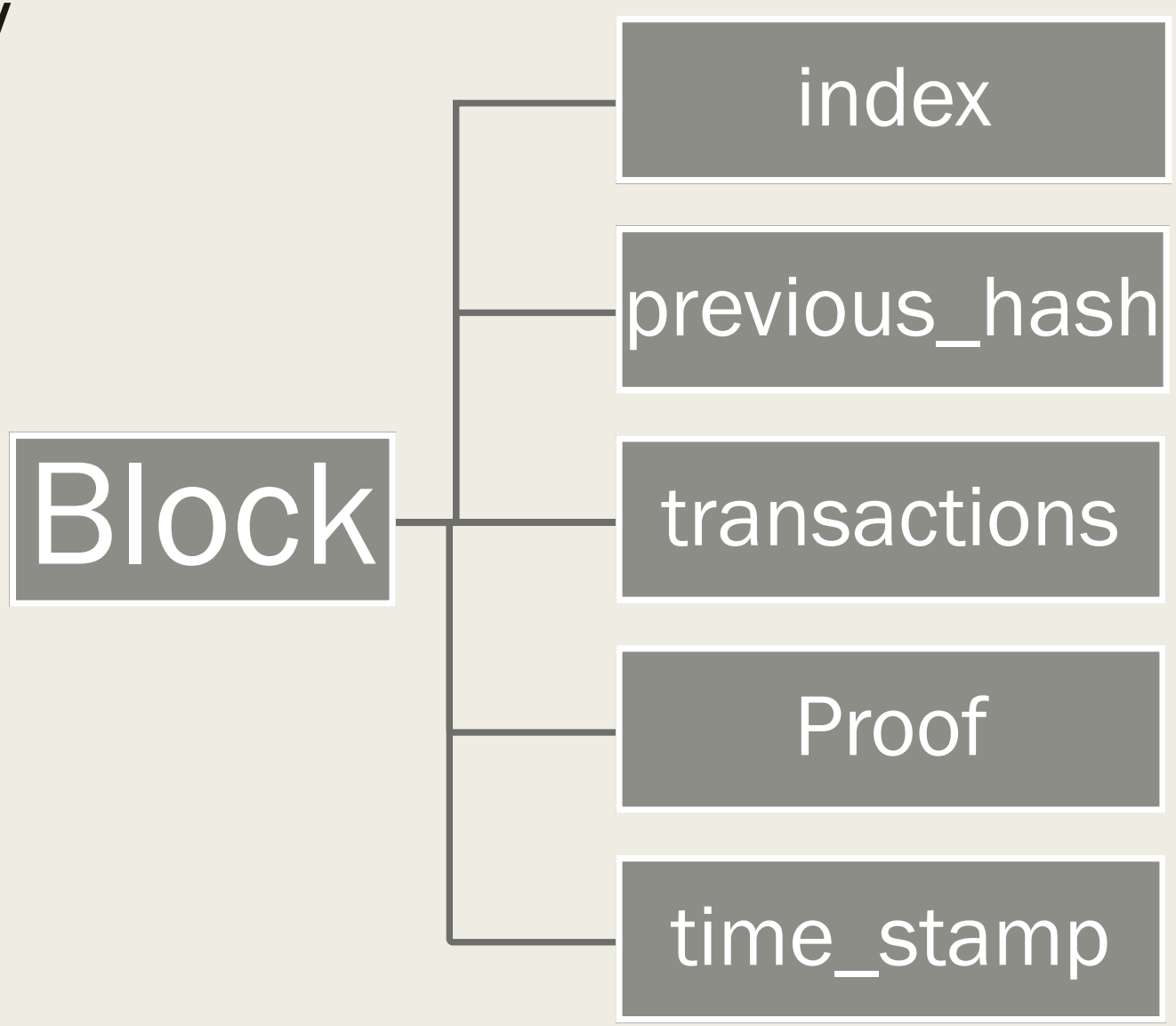
```python
from utility.hash_util import hash_block, hash_string_256
from wallet import Wallet


class Verification:
    @classmethod
    def verify_chain(cls,blockchain):
        for ind, block in enumerate(blockchain):
            if ind == 0:
                continue
            else:
                if not block.previous_hash == hash_block(blockchain[ind - 1]):
                    print("previous_hash invalid")
                    return False
                if not cls.valid_proof(block.transactions[:-1],block.previous_hash,block.proof):
                    print("POW invalid")
                    return False
                for tx in block.transactions:
                    if not Wallet.verify_transaction(tx):
                        return False
        return True

    @staticmethod
    def verify_transaction(transaction,get_balance_func):
        if get_balance_func(transaction.sender) >= transaction.amount:
            return True
        return False

    @staticmethod
    def valid_proof(transactions, last_hash, proof):
        guess = (str([tx.to_ordered_dict() for tx in transactions]) + str(last_hash) + str(proof)).encode()
        guess_hash = hash_string_256(guess)
        #print(guess_hash)
        return guess_hash[0:5] == "00000"
```

block.py

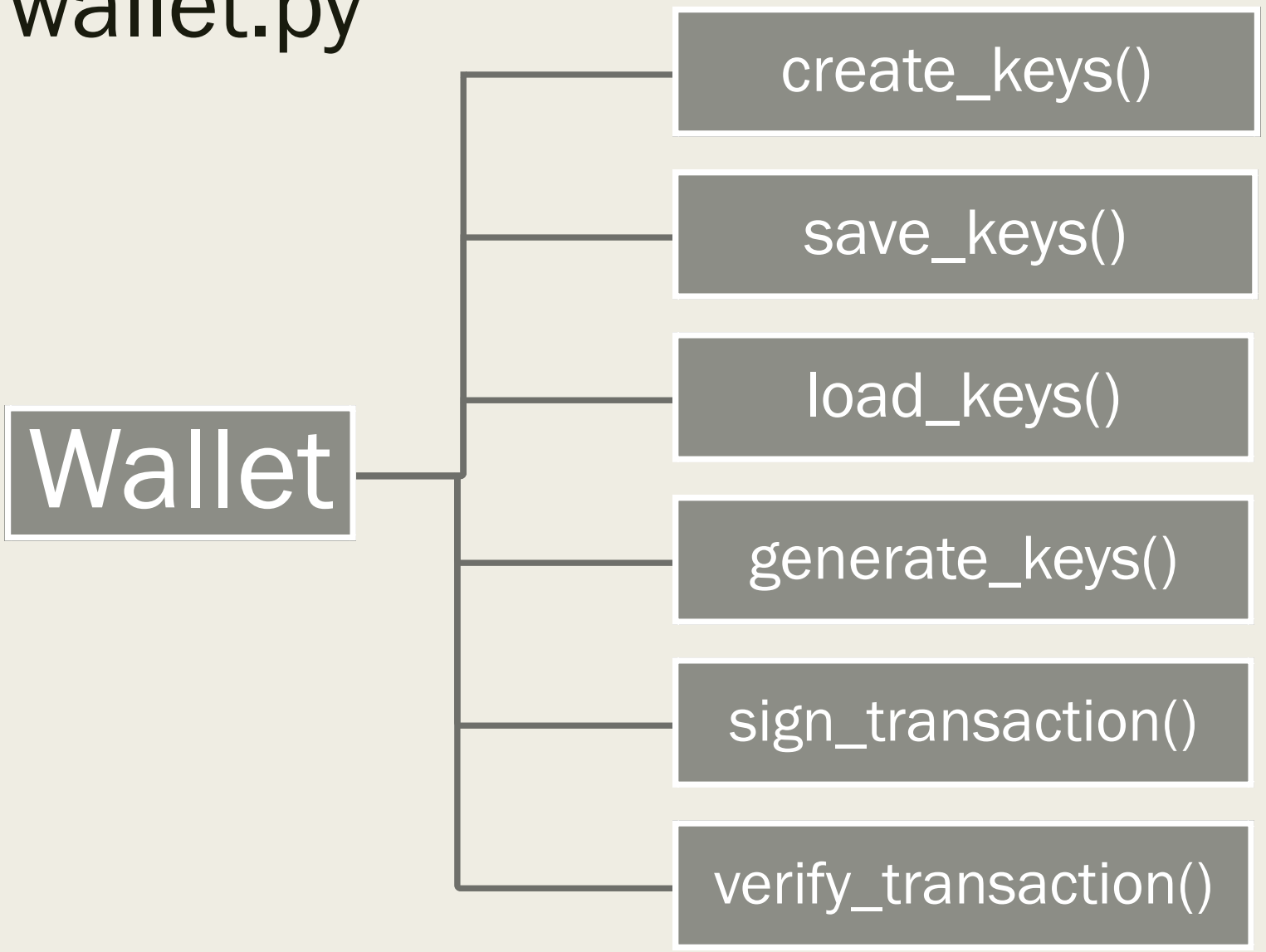


block.py

block.py

```
1  from time import time
2  from utility.printable import Printable
3
4  class Block(Printable):
5      def __init__(self, index, previous_hash, transactions, proof, time_stamp = time()):
6          self.previous_hash = previous_hash
7          self.index = index
8          self.transactions = transactions
9          self.proof = proof
10         self.time_stamp = time_stamp
```

wallet.py


Wallet
create_keys()
save_keys()
load_keys()
generate_keys()
sign_transaction()
verify_transaction()

wallet.py

wallet.py

```
1  from Crypto.PublicKey import RSA
2  from Crypto.Signature import PKCS1_v1_5
3  from Crypto.Hash import SHA256
4  import Crypto.Random
5  import binascii
6
```

wallet.py

wallet.py

```
7  class Wallet:
8      def __init__(self, node_id):
9          self.private_key = None
10         self.public_key = None
11         self.node_id = node_id
12
13     def create_keys(self):
14         private_key , public_key = self.generate_keys()
15         self.private_key = private_key
16         self.public_key = public_key
17
18
19     def save_keys(self):
20         try:
21             with open("WalletKeys_%i.txt"%self.node_id, mode = "w") as f:
22                 f.write(self.private_key)
23                 f.write("\n")
24                 f.write(self.public_key)
25             return True
26         except:
27             print("Saving keys FAILED.")
28             return False
29
```

wallet.py

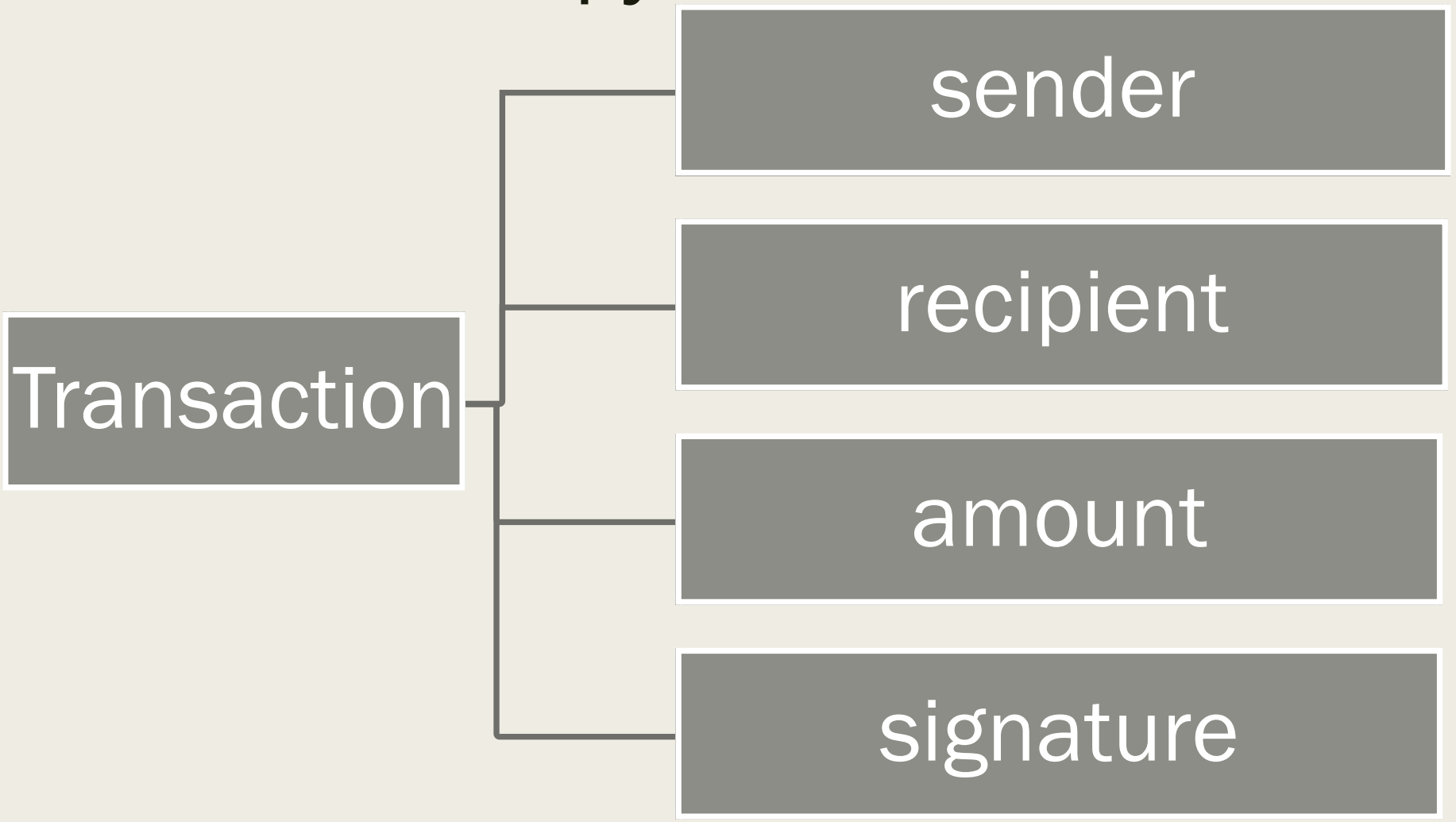
wallet.py

```
31      def load_keys(self):
32          try:
33              with open("WalletKeys_%i.txt"%self.node_id, mode = "r") as f:
34                  loaded = f.readlines()
35                  self.private_key = loaded[0][:-1]
36                  self.public_key = loaded[1]
37              return True
38
39          except:
40              print("Loading keys FAILED.")
41              return False
42
43      def generate_keys(self):
44          private_key = RSA.generate(1024, Crypto.Random.new().read)
45          public_key = private_key.publickey()
46          return (binascii.hexlify(private_key.exportKey(format = "DER")).decode("ascii"), binascii.hexlify(public_ke
47
```

wallet.py

```
48      def sign_transaction(self, sender, recipient, amount):
49          signer = PKCS1_v1_5.new(RSA.importKey(binascii.unhexlify(self.private_key)))
50          h = SHA256.new((str(sender) + str(recipient) + str(amount)).encode("utf8"))
51          signature = signer.sign(h)
52          return binascii.hexlify(signature).decode("ascii")
53
54      @staticmethod
55      def verify_transaction(transaction):
56          if transaction.sender == "MINING":
57              return True
58          public_key = RSA.importKey(binascii.unhexlify(transaction.sender))
59          verifier = PKCS1_v1_5.new(public_key)
60          h = SHA256.new((str(transaction.sender) + str(transaction.recipient) + str(transaction.amount)).encode("ut
61          return verifier.verify(h, binascii.unhexlify(transaction.signature))
62
```

transaction.py



transaction.py

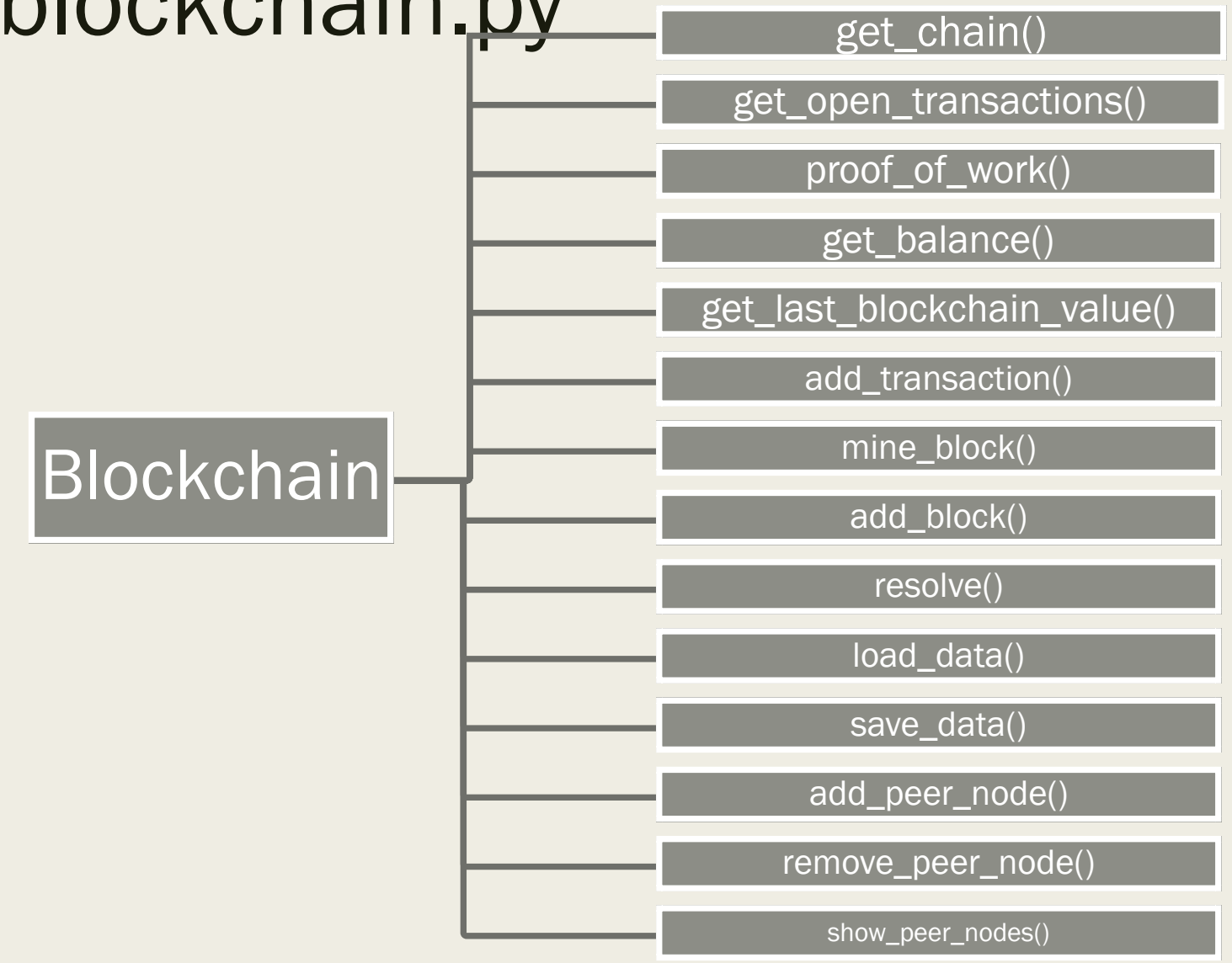
transaction.py

```python
from collections import OrderedDict
from utility.printable import Printable


class Transaction(Printable):
    def __init__(self, sender, recipient, signature, amount):
        self.sender = sender
        self.recipient = recipient
        self.amount = amount
        self.signature = signature


    def to_ordered_dict(self):
        return OrderedDict([("sender", self.sender), ("recipient", self.recipient), ("signature", self.signature), ("amount", self.amount)])
```

blockchain.py



blockchain.py

blockchain.py

```
1  import json
2  import requests
3
4  from utility.printable import Printable
5  from utility.verification import Verification
6  from utility.hash_util import hash_block, hash_string_256
7  from block import Block
8  from transaction import Transaction
9  from wallet import Wallet
10
11
12 MINING_REWARD = 10 #GLOBAL VARIABLE
13
```

blockchain.py

blockchain.py

```
16  class BlockChain:
17      def __init__(self, hosting_node, node_id):
18          self.genesis_block = Block(0,"",[],100,0)
19          self.__chain = [self.genesis_block]
20          self.__open_transactions = []
21          self.hosting_node = hosting_node
22          self.participants = {self.hosting_node}
23          self.resolve_conflicts = False
24          self.node_id = node_id
25          self.__peer_nodes = set()
26          #self.load_data()
27
28      def get_chain(self):
29          return self.__chain[:]
30
31
32      def get_open_transactions(self):
33          return self.__open_transactions[:]
34
```

__chain[:] ➔ copy list

__open_transactions[:]➔ copy list

blockchain.py

blockchain.py

```
34
35
36      def proof_of_work(self):
37          last_block = self.__chain[-1]
38          last_hash = hash_block(last_block)
39          proof = 0
40          while not Verification.valid_proof(self.__open_transactions, last_hash, proof):
41              proof = proof + 1
42          return proof
43
44
45      def get_balance(self,participant):
46          try:
47              tx_sender = [[tx.amount for tx in block.transactions if tx.sender == participant] for block in self.__chain]
48              open_tx_sender = [tran.amount for tran in self.__open_transactions if tran.sender == participant]
49              tx_recipient = [[tx.amount for tx in block.transactions if tx.recipient == participant] for block in self.__chain]
50              total_send = sum([sum(x) for x in tx_sender if len(x) > 0])
51              total_tx_send = sum(open_tx_sender)
52              total_receive = sum([sum(x) for x in tx_recipient if len(x) > 0])
53              return total_receive - total_send - total_tx_send
54          except:
55              return False
```

get_balance() ➔ looping over all blocks

blockchain.py

blockchain.py

```python
    def get_last_blockchain_value(self):
        if len(self.__chain) == 0:
            return None
        return self.__chain[-1]

    def add_transaction(self, sender, recipient, public_key, signature, amount = 1.0, is_receiving = False):
        # transaction = {
        # "sender" : sender,
        # "recipient": recipient,
        # "amount": amount}

        transaction = Transaction(sender, recipient, signature, amount)
        if not Wallet.verify_transaction(transaction):
            print("Invalid signature.")
            return False
        if Verification.verify_transaction(transaction, self.get_balance):
            self.__open_transactions.append(transaction)
            self.participants.add(sender)
            self.participants.add(recipient)
            self.save_data()
            if not is_receiving:
                # Broadcasting the new open transaction to other peer nodes
                for node in self.__peer_nodes:
                    url = "http://{}/broadcast_transaction".format(node)
                    try:
                        response = requests.post(url, json = {"sender": sender, "recipient" : recipient, "amount" : amount, "signature" : signature
                        if response.status_code == 400 or response.status_code == 500:
                            print("Transaction declined. ")
                            return False
                    except requests.exceptions.ConnectionError:
                        print("Connection failed with node : %s"%node)
                        continue
            return True
        return False
```

Initiate a transaction object

Verify signature

Check balance

Broadcast the new added transaction

blockchain.py

```
94      def mine_block(self):
95          if self.hosting_node == None:
96              return False
97          last_block = self.__chain[-1]
98          hashed_block = hash_block(last_block)
99          #print(hashed_block)
100         print("Mining new block...")
101         proof = self.proof_of_work()
102         mining_transaction =  Transaction("MINING", self.hosting_node, "", MINING_REWARD)
103         copied_open_transactions = self.__open_transactions[:]
104         copied_open_transactions.append(mining_transaction)
105         block = Block(len(self.__chain), hashed_block, copied_open_transactions, proof)
106
107         for tx in block.transactions:
108             if not Wallet.verify_transaction(tx):
109                 print("Invalid signature found during mining.")
110                 return False
111         self.__chain.append(block)
112         self.__open_transactions = []
113         self.save_data()
114         # Broadcasting the new block
115         # First convert the Block Object into a dict
116         # then convert the transaction objects into list of objects
117         dict_block_for_broadcasting = block.__dict__.copy()
118         dict_block_for_broadcasting["transactions"] = [tx.__dict__ for tx in dict_block_for_broadcasting["transactions"]]
119         for node in self.__peer_nodes:
120             url = "http://{}/broadcast_block".format(node)
121             try:
122                 response = requests.post(url, json = {"block": dict_block_for_broadcasting})
123                 if response.status_code == 400 or response.status_code == 500:
124                     print(response.json())
125                     print("Block broadcasting declined.")
126                     #return False
127                 if response.status_code == 409:
128                     self.resolve_conflicts = True
129                     print("Conflicts found!!")
130             except requests.exceptions.ConnectionError:
131                 print("Connection failed with node : %s"%node)
132                 continue
133         return True
```

Hash the last block

Doing Proof-of-Work

Mining reward

Add new block to the chain

Clear the open_transactions

Broadcasting the new block

blockchain.py

Receiving block from other nodes

Verify Proof-Of-Work

Reset the open_transactions

```
136     def add_block(self, block):
137         # Receiving broadcasting block
138         # input block format is in DICT
139         # valid_proof should be avoid MINING REWARD
140         transactions_to_valid_proof = [Transaction(tx["sender"],tx["recipient"],tx["signature"],tx["amount"]) for tx in block["transactions"][:-1]]
141         transactions_to_final_add = [Transaction(tx["sender"],tx["recipient"],tx["signature"],tx["amount"]) for tx in block["transactions"]]
142         proof_is_valid = Verification.valid_proof(transactions_to_valid_proof , block["previous_hash"], block["proof"])
143         hashes_match = hash_block(self.get_chain()[-1]) == block["previous_hash"]
144         print(proof_is_valid)
145         print(hashes_match)
146         if proof_is_valid and hashes_match:
147             self.__chain.append(Block(block["index"],block["previous_hash"],transactions_to_final_add,block["proof"],block["time_stamp"]))
148             # Clean the Open_transactions
149             # ISSUE: SAME SENDER & SAME RECIPIENT & SAME AMOUNT & SAME SIGNATURE
150             # still need to solve!!
151             stored_local_open_trans = self.__open_transactions[:]
152             for in_trans in block["transactions"]:
153                 for op_tx in stored_local_open_trans:
154                     if in_trans["sender"] == op_tx.sender and in_trans["recipient"] == op_tx.recipient and in_trans["amount"] == op_tx.amount and in_t
155                         try:
156                             self.__open_transactions.remove(op_tx)
157                         except ValueError:
158                             print("Transaction already removed.")
159             self.save_data()
160             return True
161         else:
162             return False
163
164
```

blockchain.py

Resolving the conflict➔
LONGER chain wins!

```
165  def resolve(self):
166      winner_chain = self.get_chain()
167      whether_replace_local_chain = False
168      for node in self.__peer_nodes:
169          url = "http://{}/chain".format(node)
170          try:
171              response = requests.get(url)
172              peer_chain = response.json()
173              peer_chain = [Block(block["index"], block["previous_hash"], block["transactions"], block["proof"], block["time_stamp"]) for block in p
174              for elem in peer_chain:
175                  elem.transactions = [Transaction(tx["sender"], tx["recipient"], tx["signature"], tx["amount"]) for tx in elem.transactions]
176              peer_chain_length = len(peer_chain)
177              local_chain_length = len(winner_chain)
178              whether_peer_chain_longer = peer_chain_length > local_chain_length
179              whether_peer_chain_valid = Verification.verify_chain(peer_chain)
180              if whether_peer_chain_longer and whether_peer_chain_valid:
181                  winner_chain = peer_chain[:]
182                  whether_replace_local_chain = True
183
184          except requests.exceptions.ConnectionError:
185              print("Connection failed with node : %s"%node)
186              continue
187      self.resolve_conflicts = False
188      self.__chain = winner_chain
189      if whether_replace_local_chain:
190          self.__open_transactions = []
191      self.save_data()
192      return whether_replace_local_chain
```

Verify the peer chain

blockchain.py

```
199 def load_data(self):
200     try:
201         with open("basicBlockChain%i.txt"%self.node_id, mode = "r") as f:
202             loaded = f.readlines()
203             loaded_blockchain = loaded[0][:-1]
204             #print(loaded_blockchain)
205             loaded_open_transactions = loaded[1][:-1]
206             loaded_peer_nodes = loaded[2]
207             #print(loaded_open_transactions)
208             blockchain = json.loads(loaded_blockchain)
209             updated_blockchain = []
210             for block in blockchain:
211                 converted_tx = []
212                 for tx in block["transactions"]:
213                     tx_obj = Transaction(tx["sender"], tx["recipient"], tx["signature"], tx["amount"])
214                     converted_tx.append(tx_obj)
215                 updated_block = Block(block["index"], block["previous_hash"], converted_tx, block["proof"], block["time_stamp"])
216                 updated_blockchain.append(updated_block)
217             self.__chain = updated_blockchain
218
219             open_transactions = json.loads(loaded_open_transactions)
220             updated_open_transactions = []
221             for tx in open_transactions:
222                 open_tx_obj = Transaction(tx["sender"], tx["recipient"], tx["signature"], tx["amount"])
223                 updated_open_transactions.append(open_tx_obj)
224             self.__open_transactions = updated_open_transactions
225             peer_nodes = json.loads(loaded_peer_nodes)
226             self.__peer_nodes = set(peer_nodes)
227
228         # Update participants
229         for block in self.__chain:
230             for tx in block.transactions:
231                 self.participants.add(tx.sender)
232                 self.participants.add(tx.recipient)
233         for tx in self.__open_transactions:
234             self.participants.add(tx.sender)
235             self.participants.add(tx.recipient)
236
237     except IOError:
238         print("IO Error. File not found.")
239
240     except IndexError:
241         print("IndexError. File is empty.")
242
```

Reconstructing each Transaction

Reconstructing each Block

blockchain.py

```
244  def save_data(self):
245      try:
246          with open("basicBlockChain%i.txt"%self.node_id, mode = "w") as f:
247              saveable_blockchain = [block.__dict__.copy() for block in self.__chain]
248              for block in saveable_blockchain:
249                  block["transactions"] = [tx.to_ordered_dict() for tx in block["transactions"]]
250              f.write(json.dumps(saveable_blockchain))
251              f.write("\n")
252              saveable_open_transactions = [tx.to_ordered_dict() for tx in self.__open_transactions]
253              f.write(json.dumps(saveable_open_transactions))
254              f.write("\n")
255              f.write(json.dumps(list(self.__peer_nodes)))
256      except:
257          print("Saving file failed.")
258
```

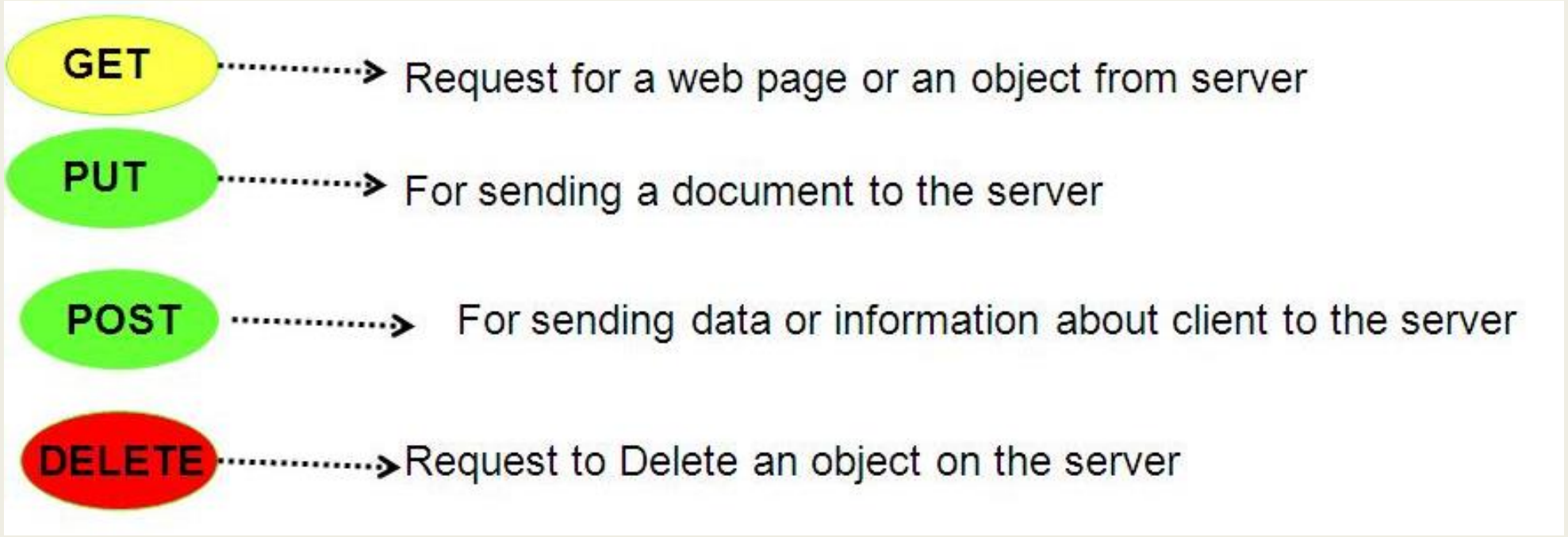
blockchain.py

```
260     def add_peer_node(self,node):
261         # Node should be an URL
262         self.__peer_nodes.add(node)
263         self.save_data()
264 
265     def remove_peer_node(self,node):
266         # Node should be an URL
267         self.__peer_nodes.discard(node)
268         self.save_data()
269 
270     def show_peer_nodes(self):
271         return list(self.__peer_nodes.copy())
272 
```

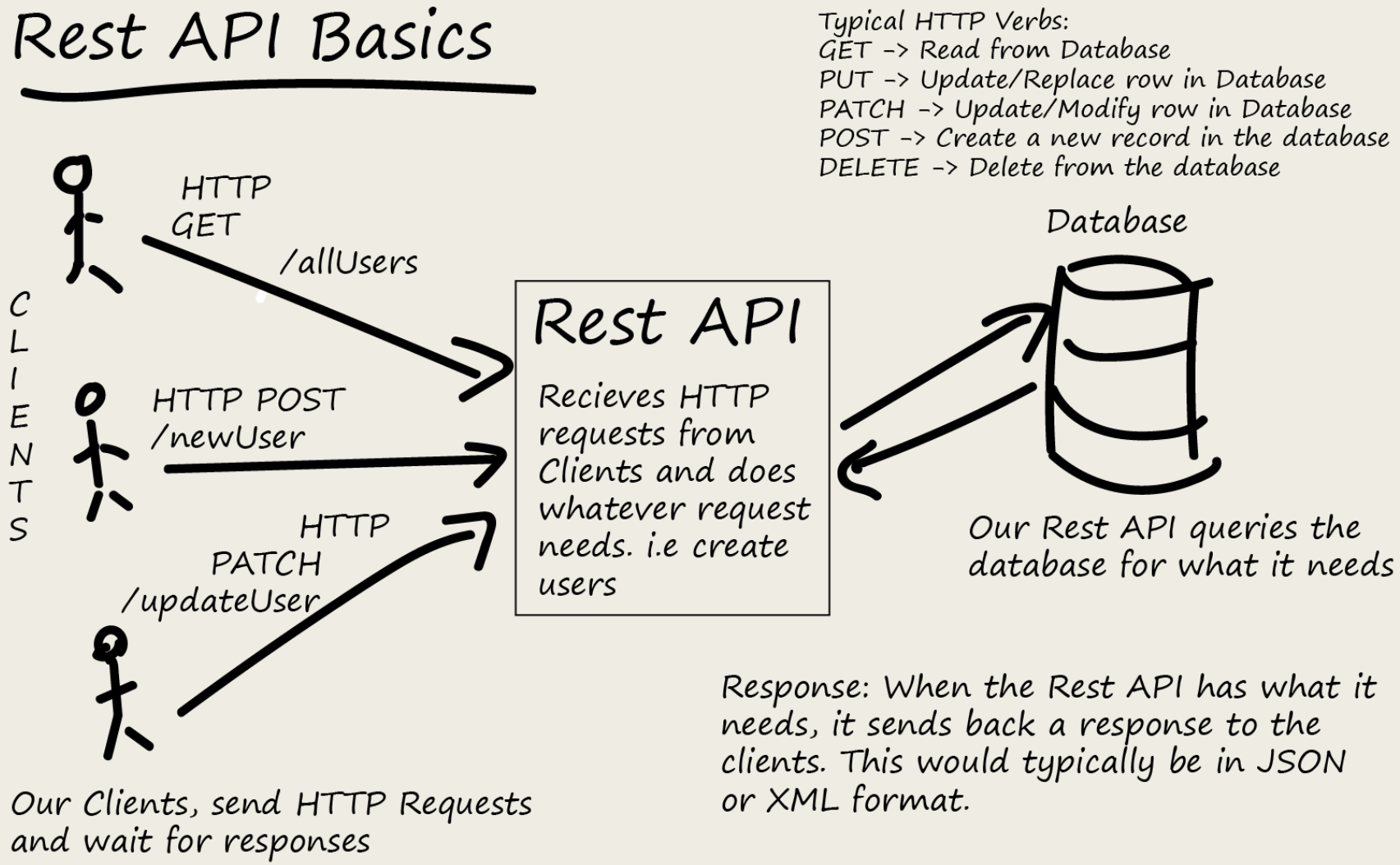
WEB PROGRAMMING

HTTP

- GET
- POST



API



Source:https://tutorialedge.net/general/what-is-a-rest-api/

Python Frameworks

overview // docs // community // extensions // donate

Flask is a microframework for Python based on Werkzeug, Jinja 2 and good intentions. And before you ask: It's BSD licensed!

Flask is Fun

Latest Version: 1.0.2

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World!"
```

And Easy to Setup

```
$ pip install Flask
$ FLASK_APP=hello.py flask run
 * Running on http://localhost:5000/
```

Basic flask web app

Flask is Fun

Latest Version: 1.0.2

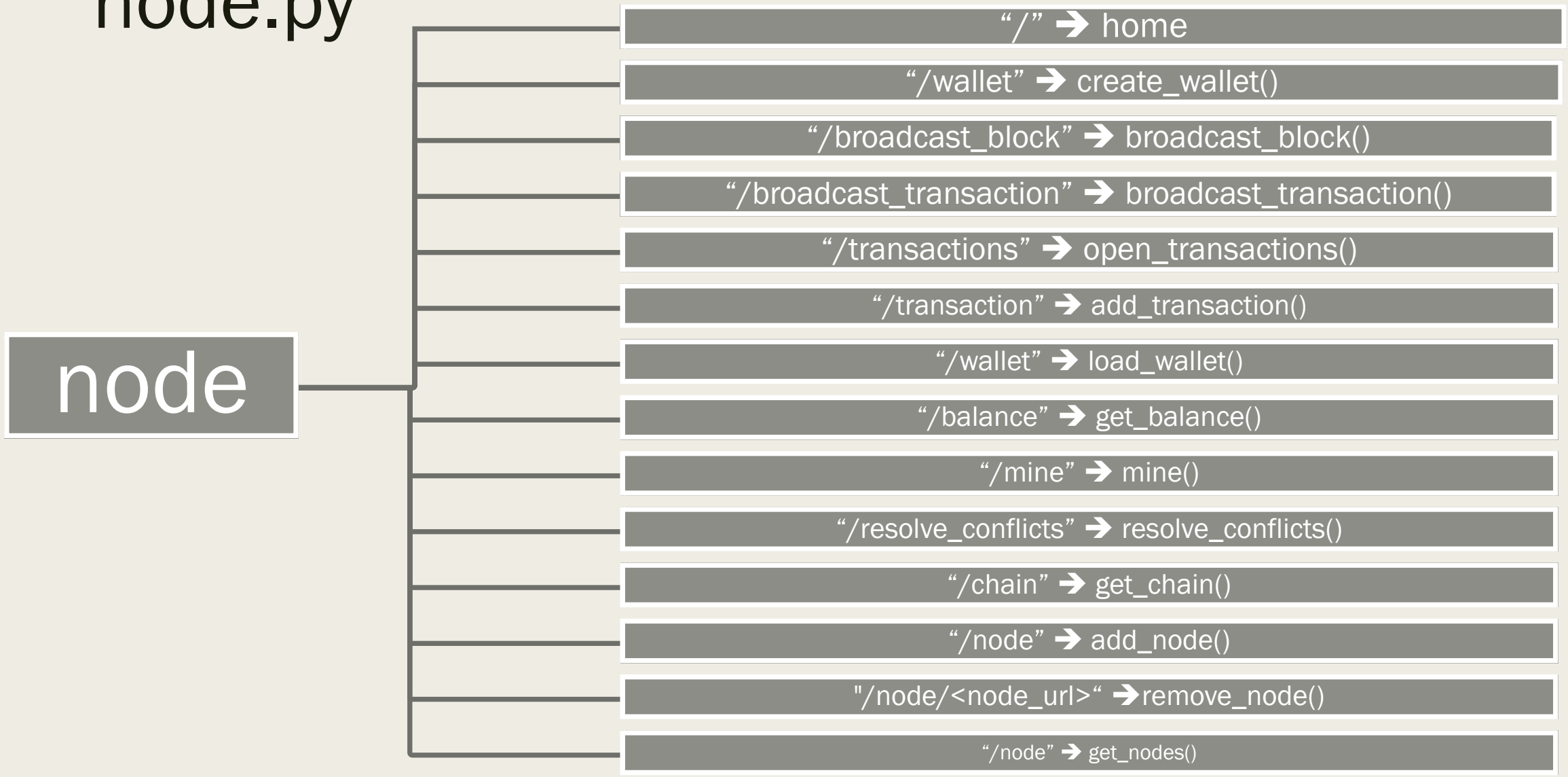
```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World!"
```

HTTP PATH

ACTION

node.py


node
“/” ➔ home
“/wallet” ➔ create_wallet()
“/broadcast_block” ➔ broadcast_block()
“/broadcast_transaction” ➔ broadcast_transaction()
“/transactions” ➔ open_transactions()
“/transaction” ➔ add_transaction()
“/wallet” ➔ load_wallet()
“/balance” ➔ get_balance()
“/mine” ➔ mine()
“/resolve_conflicts” ➔ resolve_conflicts()
“/chain” ➔ get_chain()
“/node” ➔ add_node()
"/node/<node_url>“ ➔remove_node()
“/node” ➔ get_nodes()

node.py

node.py

```
1  from flask import Flask , jsonify, request
2  from flask_cors import CORS
3  import json
4
5  from wallet import Wallet
6  from blockchain import BlockChain
7
8  app = Flask(__name__)
9  CORS(app)
10
```

Cross Origin Resource Sharing

node.py

```
15  @app.route("/", methods = ["GET"])
16  def get_ui():
17      return "This works"
18
```

```python
@app.route("/wallet", methods = ["POST"])
def create_wallet():
    wallet.create_keys()
    if wallet.save_keys():
        global blockchain
        blockchain = BlockChain(wallet.public_key, port)
        # By default will load previous blockchain data
        # May be set to adjustable on UI
        blockchain.load_data()
        response = {
        "public_key" : wallet.public_key,
        "private_key" : wallet.private_key,
        "Balance : " : blockchain.get_balance(wallet.public_key)
        }
        return jsonify(response), 201
    else:
        response = {
        "message" : "Saving keys failed."
        }
        return jsonify(response), 500

```

node.py

```
40  @app.route("/broadcast_block", methods = ["POST"])
41  def broadcast_block():
42      values = request.get_json()
43      if not values:
44          response = {
45          "message : " : "Data not found."
46          }
47          return jsonify(response), 400
48      #required_fields = ["previous_hash", "index", "transactions", "proof", "time_stamp"]
49      #if not all(field in values for field in required_fields):
50      if  "block" not in values:
51          response = {
52          "message : " : " Block data format is incorrect."
53          }
54          return response, 400
55      # block_previous_hash = values["previous_hash"]
56      # block_index = values["index"]
57      # block_transactions = values["transactions"]
58      # block_proof = values["proof"]
59      # block_time_stamp = values["time_stamp"]
60      block = values["block"]
```

```
61      if block["index"] == blockchain.get_chain()[-1].index + 1:
62          # block now is a dict
63          # will be parsed in the add_block method
64          success = blockchain.add_block(block)
65          if not success:
66              response = {
67              "message" : "Block not added"
68              }
69              return jsonify(response), 409
70          response = {
71          "message" : "Block added successfully."
72          }
73          return jsonify(response), 201
74
75      elif block["index"] > blockchain.get_chain()[-1].index + 1:
76          response = {
77          "message : " : "Blockchain seems to differ from local blockahain."
78          }
79          blockchain.resolve_conflicts = True
80          return jsonify(response), 200
81      else:
82          response = {
83          "message : " : "Blockchain seems to be shorter. Block not added."
84          }
85          return jsonify(response), 409
86
87
```

node.py

```
91  @app.route("/broadcast_transaction", methods = ["POST"])
92  def broadcast_transactions():
93      values = request.get_json()
94      if not values:
95          response = {
96          "message" : "No data found."
97          }
98          return jsonify(response), 400
99      required_fields = ["sender", "recipient", "amount", "signature"]
100     if not all(field in values for field in required_fields):
101         response = {
102         "message" : "Transaction data format is incorrect."
103         }
104         return jsonify(response), 400
105     tx_sender, tx_recipient, tx_amount, tx_signature = values["sender"], values["recipient"], values["amount"], values["signature"]
106     success = blockchain.add_transaction(tx_sender, tx_recipient, tx_sender, tx_signature,tx_amount, is_receiving = True )
107     if success:
108         response = {
109         "Message : " : "Transaction successfully added.",
110         "Transaction" :{ "sender" : tx_sender, "recipient" : tx_recipient, "amount" : tx_amount, "signature" : tx_signature},
111         }
112         return jsonify(response), 201
113     else:
114         response = {
115         "Message : " : "Transaction failed. (Broadcasting)",
116         }
117         return jsonify(response), 500
118
119
```

node.py

```
124  @app.route("/transactions", methods = ["GET"])
125  def open_transactions():
126      open_trans = blockchain.get_open_transactions()
127      open_trans = [tx.to_ordered_dict() for tx in open_trans]
128      response = {
129      "message : " : "Current open transactions...",
130      "Open transactions :" : open_trans
131      }
132      return jsonify(response), 200
133
134
```

```python
@app.route("/transaction", methods = ["POST"])
def add_transaction():
    if wallet.public_key == None:
        response = {
        "Message : " : "Wallet not set up or created."
        }
        return jsonify(response), 400

    values = request.get_json()
    if not values:
        response = {
        "Message : " : "No data found."
        }
        return jsonify(response), 400

    required_fields = ["recipient", "amount"]

    if not all(field in values for field in required_fields):
        response = {
        "Message : " : "Data format is not correct. "
        }
        return jsonify(response), 400

    tx_sender, tx_recipient, tx_amount = wallet.public_key, values["recipient"], values["amount"]

    signature = wallet.sign_transaction(tx_sender, tx_recipient, tx_amount)

    if blockchain.add_transaction(tx_sender, tx_recipient, wallet.public_key, signature, tx_amount):
        response = {
        "Message : " : "Transaction successfully added.",
        "Transaction" :{ "sender" : tx_sender, "recipient" : tx_recipient, "amount" : tx_amount},
        "Balance : " : blockchain.get_balance(wallet.public_key)
        }
        return jsonify(response), 201
    else:
        response = {
        "Message : " : "Transaction failed.",
        "Balance : " : blockchain.get_balance(wallet.public_key)
        }
        return jsonify(response), 500

```

```python
@app.route("/wallet", methods = ["GET"])
def load_wallet():
    if wallet.load_keys():
        global blockchain
        blockchain = BlockChain(wallet.public_key, port)
        blockchain.load_data()
        response = {
        "public_key" : wallet.public_key,
        "private_key" : wallet.private_key,
        "Balance : " : blockchain.get_balance(wallet.public_key)
        }
        return jsonify(response), 201
    else:
        response = {
        "message" : "Loading keys failed."
        }
        return jsonify(response), 500

```

node.py

```
197
198 @app.route("/balance", methods = ["GET"])
199 def get_balance():
200     current_balance = blockchain.get_balance(wallet.public_key)
201     if not current_balance == False:
202         response = {
203         "message" : "Showing current balance...",
204         "Wallet address : " : wallet.public_key,
205         "Funds : " : current_balance
206         }
207         return jsonify(response), 200
208     else:
209         response = {
210         "message" : "Showing current balance failed.",
211         "Wallet set up": wallet.public_key != None,
212         }
213         return jsonify(response), 500
214
```

```python
@app.route("/mine", methods = ["POST"])
def mine():
    if blockchain.resolve_conflicts:
        response = {
        "message : " : "Resolve conflicts first. Block not added"
        }
        return jsonify(response), 409

    if blockchain.mine_block():
        new_block = blockchain.get_chain()[-1].__dict__.copy()
        new_block["transactions"] = [tx.to_ordered_dict() for tx in new_block["transactions"]]
        response = {
        "message :" : "Adding a block successfully.",
        "Block : " : new_block,
        "balance : " : blockchain.get_balance(wallet.public_key)
        }
        return jsonify(response), 201
    else:
        response = {
        "message" : "Adding a block failed.",
        "Wallet set up" : wallet.public_key != None,

        }
        return jsonify(response), 500
```

node.py

```
240
241 @app.route("/resolve_conflicts", methods = ["POST"])
242 def resolve_conflicts():
243     whether_replaced = blockchain.resolve()
244     if whether_replaced:
245         response = {
246         "message : " : "Local blockchain replaced by peer chain."
247         }
248     else:
249         response = {
250         "message : " : "Local chain not replaced."
251         }
252     return jsonify(response), 200
253
```

node.py

```
253
254 @app.route("/chain", methods = ["GET"])
255 def get_chain():
256     chain_snapshot = blockchain.get_chain()
257     dict_chain = [block.__dict__.copy() for block in chain_snapshot]
258     for elem in dict_chain:
259         elem["transactions"] = [tx.to_ordered_dict() for tx in elem["transactions"]]
260     chain_json = jsonify(dict_chain)
261     return chain_json, 200
262
263
```

```
264 @app.route("/node", methods = ["POST"])
265 def add_node():
266     values = request.get_json()
267     if not values:
268         response = {
269         "message " : "No data found."
270         }
271         return jsonify(response), 400
272     if not "node" in values:
273         response = {
274         "message" : "No node URL found."
275         }
276         return jsonify(response), 400
277
278     node_url = values["node"]
279     blockchain.add_peer_node(node_url)
280     response = {
281     "message : " : "Node added.",
282     "all_nodes : " : blockchain.show_peer_nodes()
283     }
284     return jsonify(response), 201
285
286
```

node.py

```
290
291 @app.route("/node/<node_url>", methods = ["DELETE"])
292 def remove_node(node_url):
293     if node_url == "" or node_url == None:
294         response = {
295         "message : " : "No node url found."
296         }
297         return jsonify(response), 400
298
299     blockchain.remove_peer_node(node_url)
300     response = {
301     "message : " : "Node removed.",
302     "all_nodes : " : blockchain.show_peer_nodes()
303     }
304     return jsonify(response), 200
305
306
```

node.py

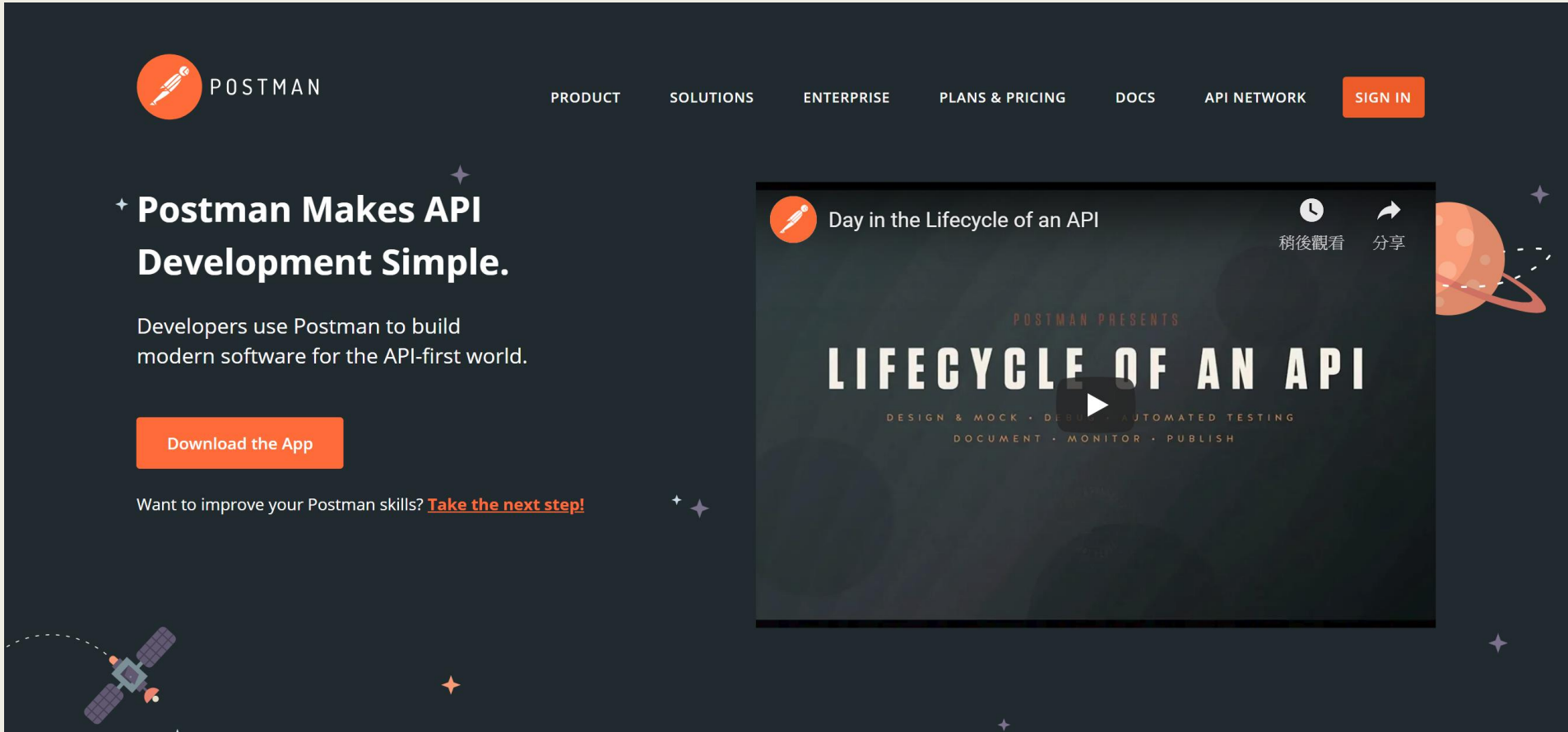
```
306
307  @app.route("/node", methods = ["GET"])
308  def get_nodes():
309      all_nodes = blockchain.show_peer_nodes()
310      response = {
311      "message : " : "Showing all nodes.",
312      "all_nodes : " : all_nodes
313      }
314      return jsonify(response), 200
315
316
317
```

node.py

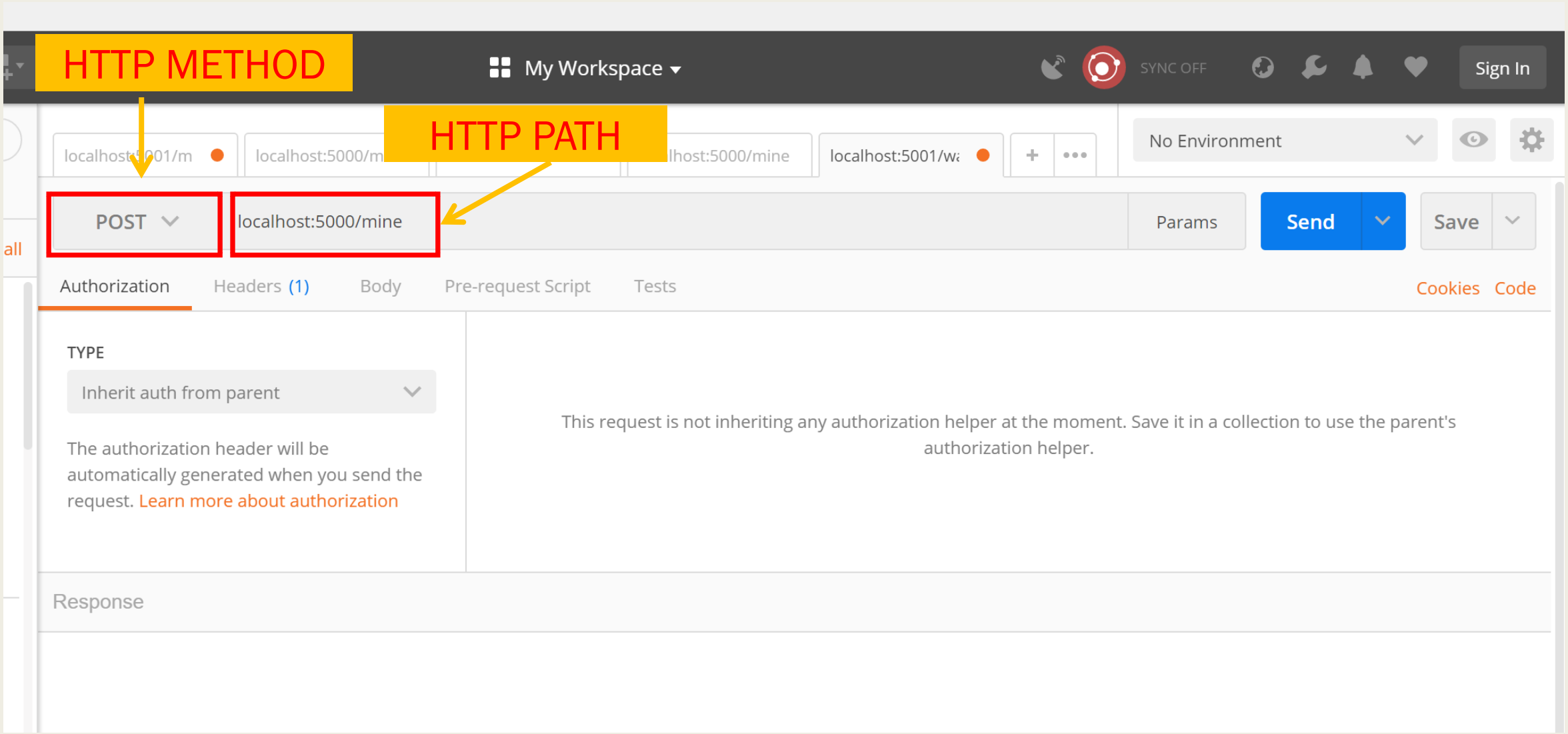
```
318
319 if __name__ == '__main__':
320     from argparse import ArgumentParser
321     parser = ArgumentParser()
322     parser.add_argument("-p", "--port", type = int, default = 5000)
323     args = parser.parse_args()
324     port = args.port
325     wallet = Wallet(port)
326     blockchain = BlockChain(wallet.public_key,port)
327     blockchain.load_data()
328     app.run(host = "127.0.0.1", port = port)
329
330
331
```

Testing

- Postman

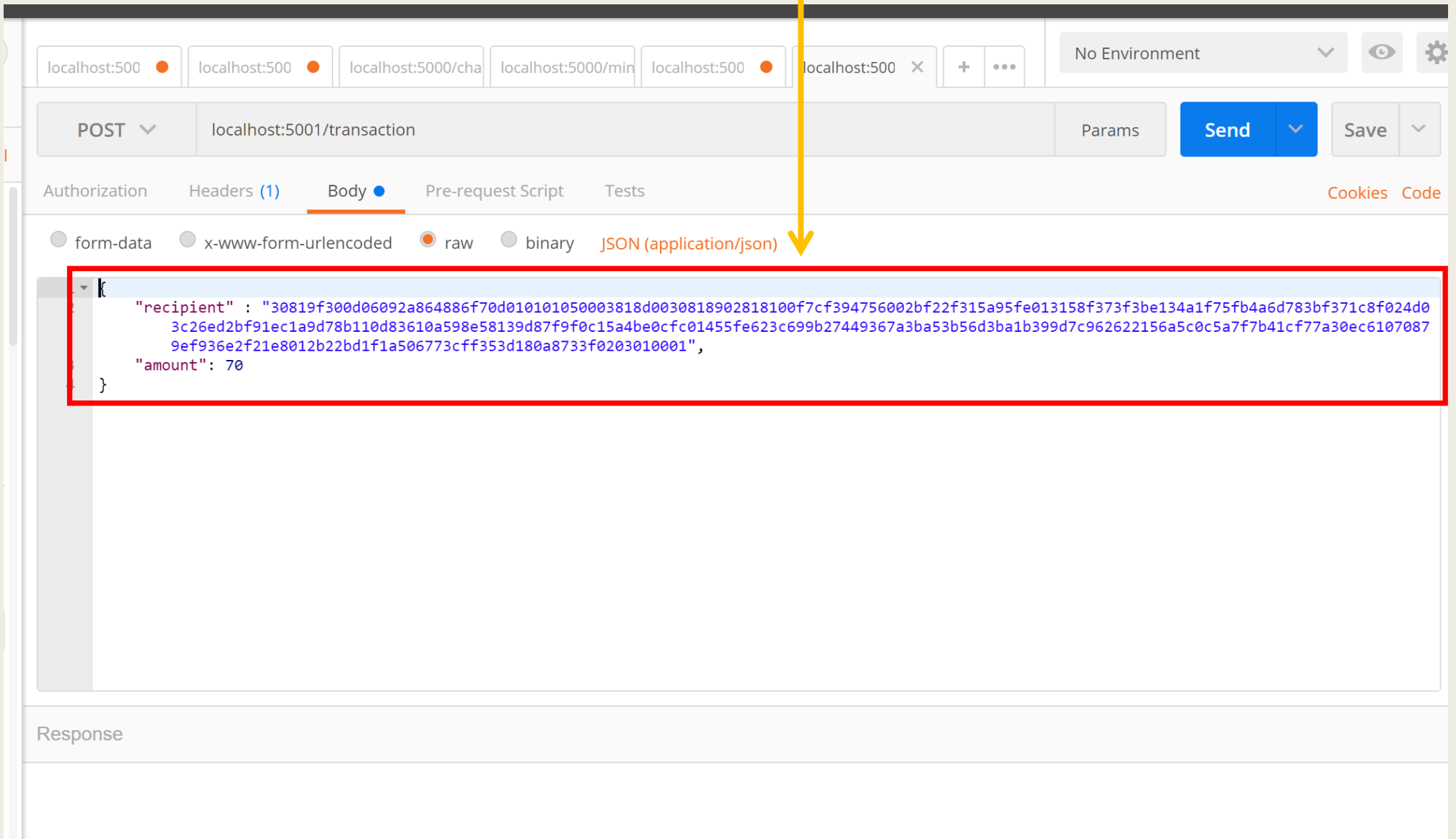


Postman


HTTP METHOD
HTTP PATH
My Workspace
SYNC OFF
Sign In
No Environment
POST
localhost:5000/mine
Params
Send
Save
Authorization
Headers (1)
Body
Pre-request Script
Tests
Cookies
Code
TYPE
Inherit auth from parent
The authorization header will be automatically generated when you send the request. Learn more about authorization
This request is not inheriting any authorization helper at the moment. Save it in a collection to use the parent's authorization helper.
Response

Postman

JSON CONTENT



DISCUSSION

Improvements

- Improve Error Handling
- Scalability
- Broadcasting (Async? Queue? Schedule?)
- Mining Difficulty
- VM
- Transaction with (text) message

Applications

- Smart contract
- Decentralized Marketplace
- Decentralized Web storage
- Decentralized Web service
- Decentralized computing (much more difficult)

Smart contract

- Ethereum
- Turing-Complete Virtual Machine
- NOT a supercomputer

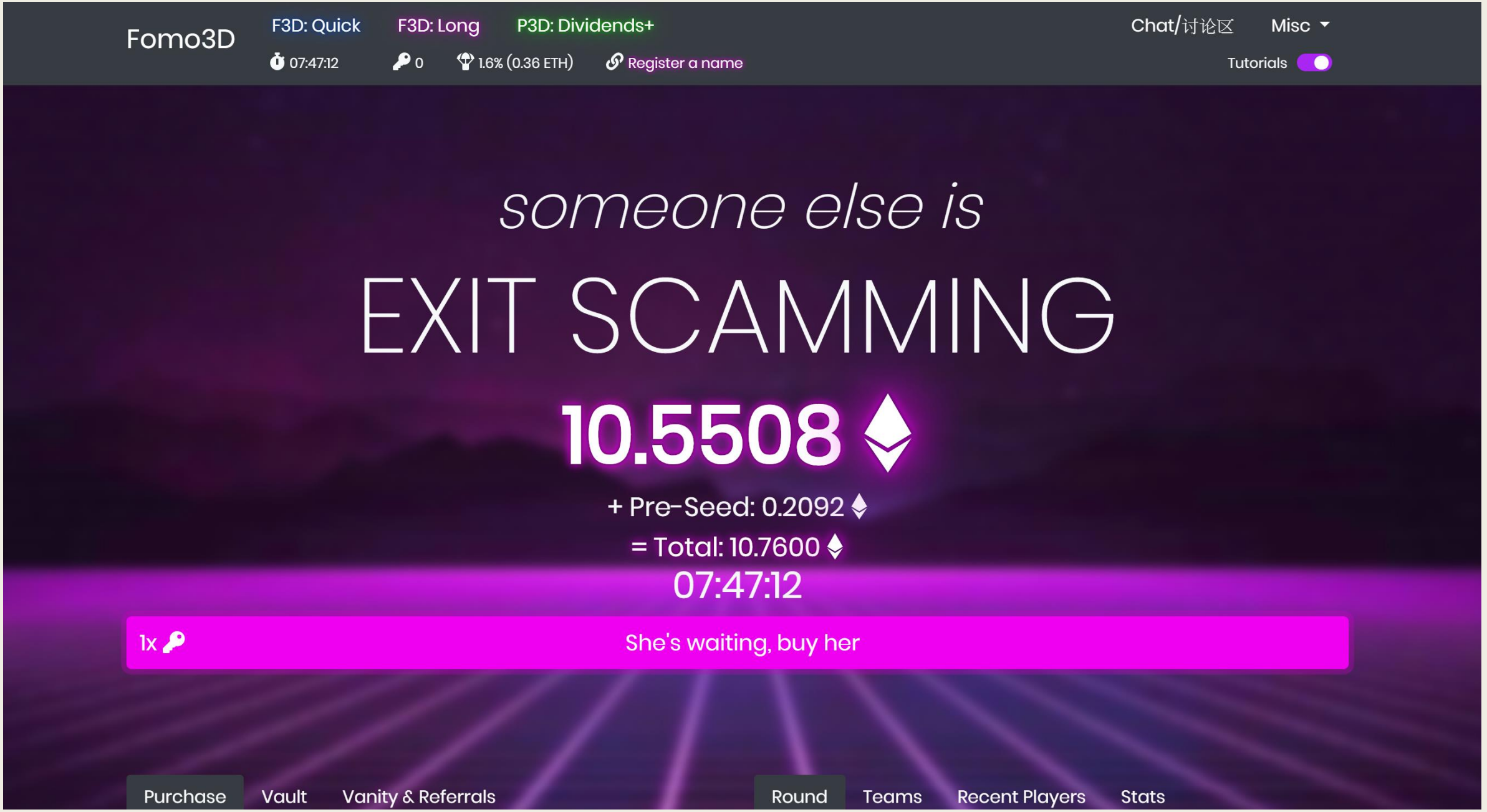
謎戀貓 CryptoKitties 網絡通暢

貓目錄 搜索 遊戲指南 更多 開始

可收藏
可繁殖
討人喜歡

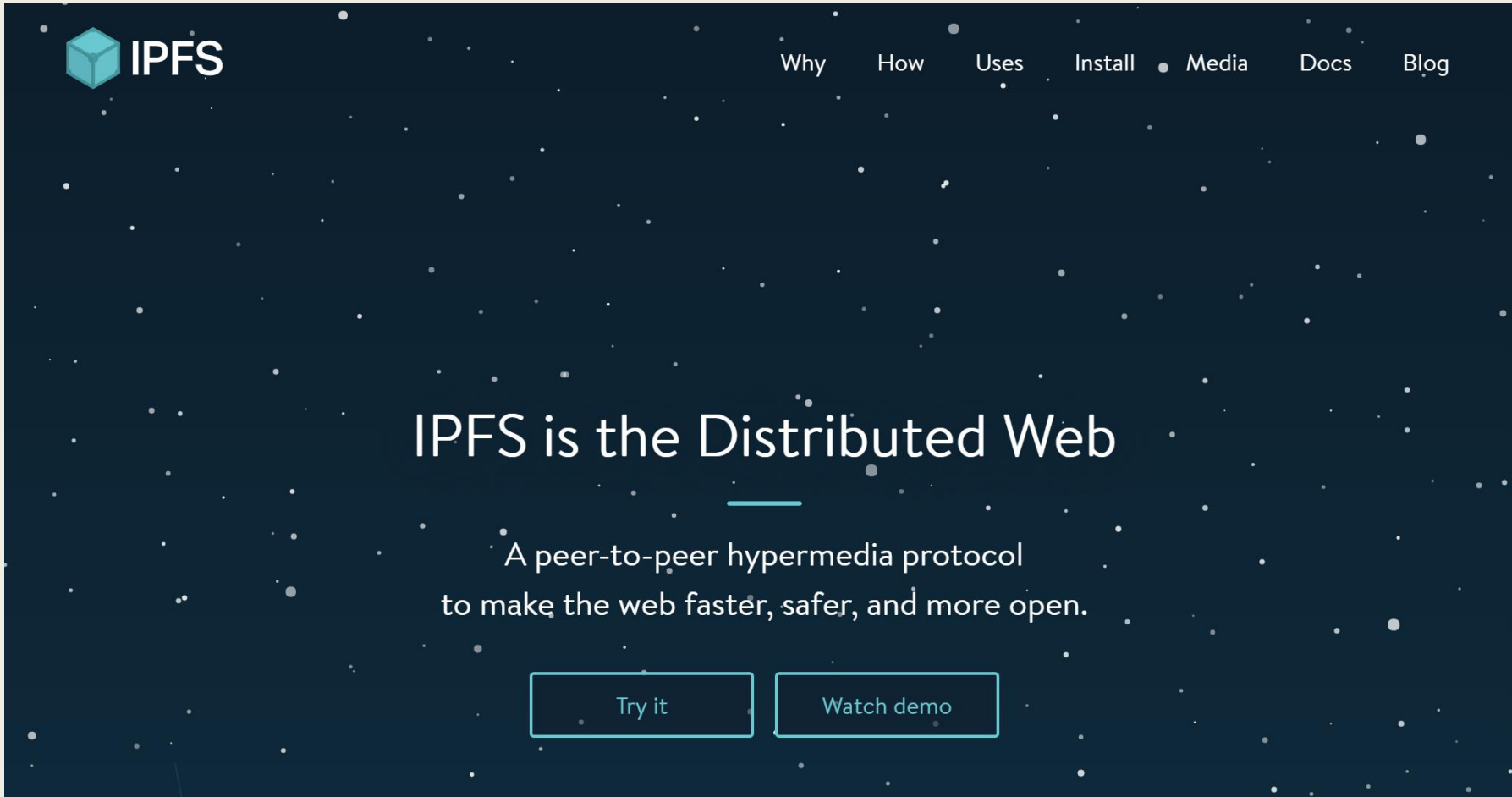
收藏並繁殖數字喵咪

開始游戲


Fomo3D
F3D: Quick
F3D: Long
P3D: Dividends+
Chat/讨论区
Misc
07:47:12
0
1.6% (0.36 ETH)
Register a name
Tutorials
someone else is
EXIT SCAMMING
10.5508
+ Pre-Seed: 0.2092
= Total: 10.7600
07:47:12
1x
She's waiting, buy her
Purchase
Vault
Vanity & Referrals
Round
Teams
Recent Players
Stats

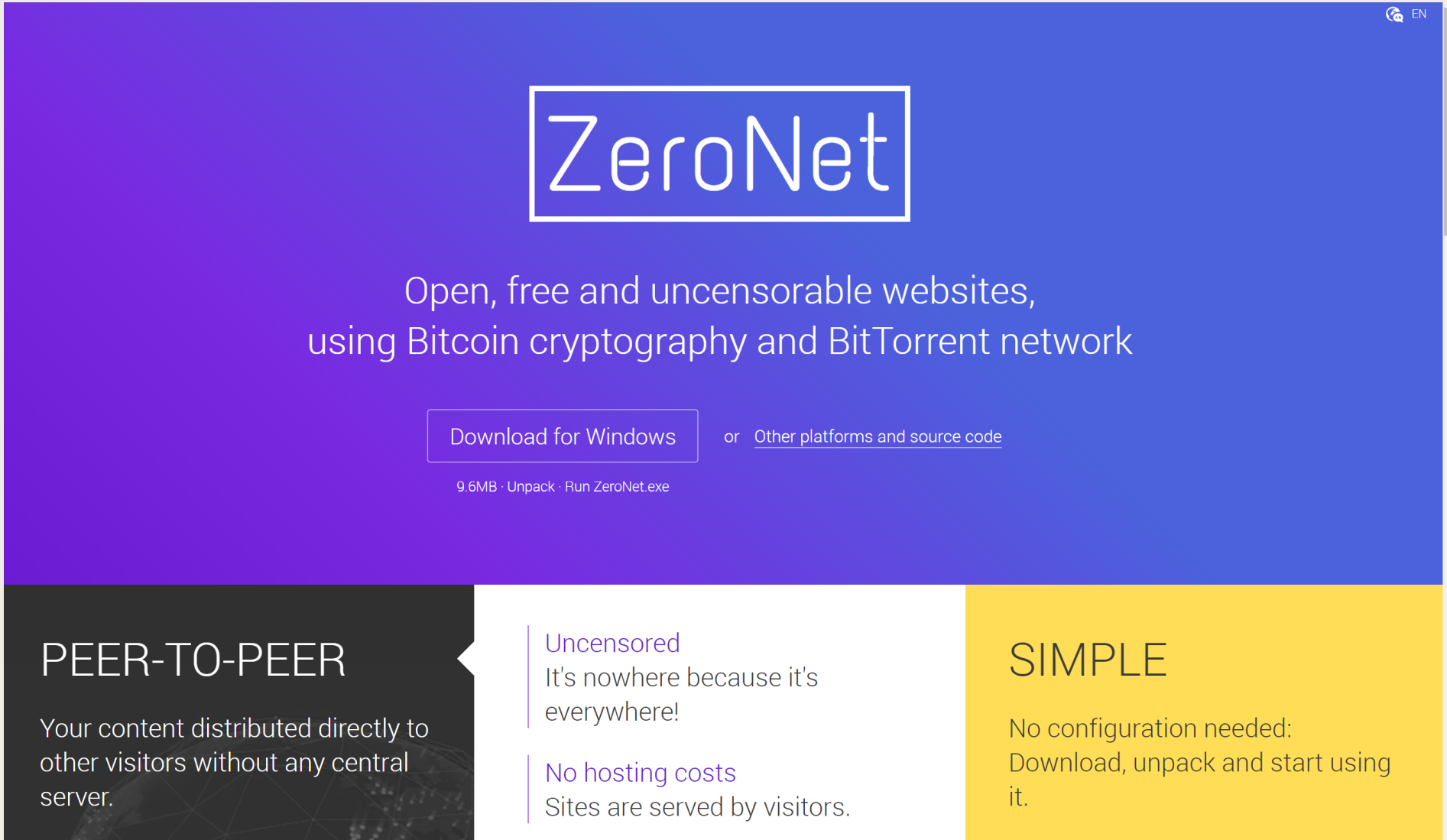
Decentralized Web storage

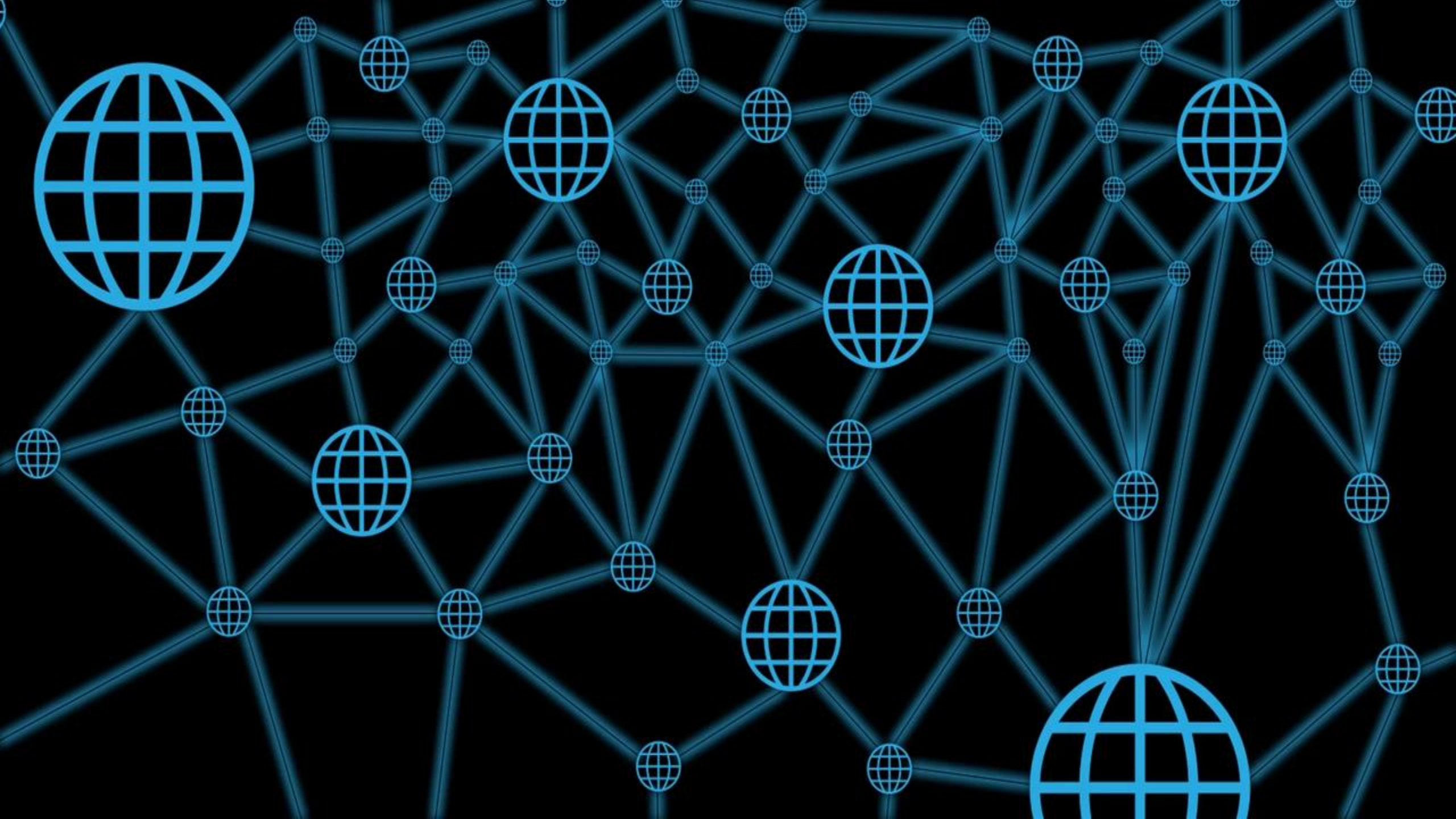
- IPFS



Decentralized Web service

- ZeroNet





THE END