Learning **Blockchain** from the inside

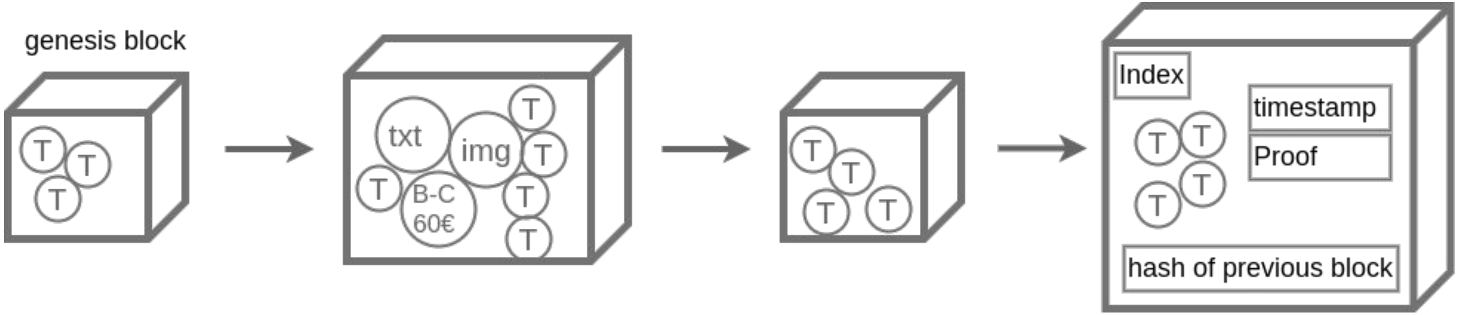
@beatrizmrg

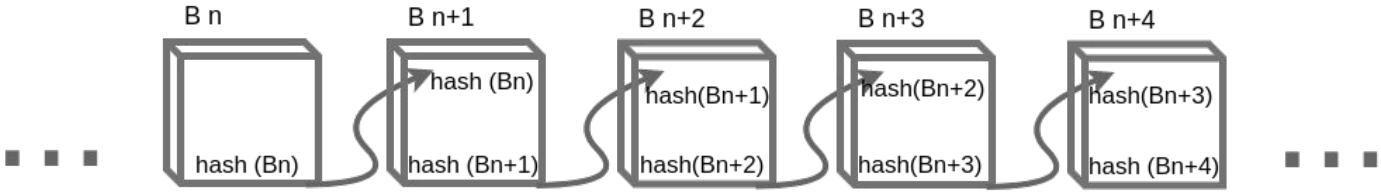
IBMer, insatiable learner, passionate about technology and innovation #AI #cloud #CI #CrossFit

Building our first Blockchain with Docker containers

1. **Blockchain**
   a. Blocks
   b. chain
   c. hash
   d. Transactions
   e. immutability
   f. Proof of work
   g. mine
2. Interacting with the blockchain
3. Consensus
   a. Distributed network
   b. Interacting with the network

Blockchain


genesis block
T
T
T
txt
img
B-C
60€
T
T
T
T
T
Index
timestamp
Proof
hash of previous block


...
B n
hash (Bn)
B n+1
hash (Bn)
hash (Bn+1)
B n+2
hash(Bn+1)
hash(Bn+2)
B n+3
hash(Bn+2)
hash(Bn+3)
B n+4
hash(Bn+3)
hash (Bn+4)
...

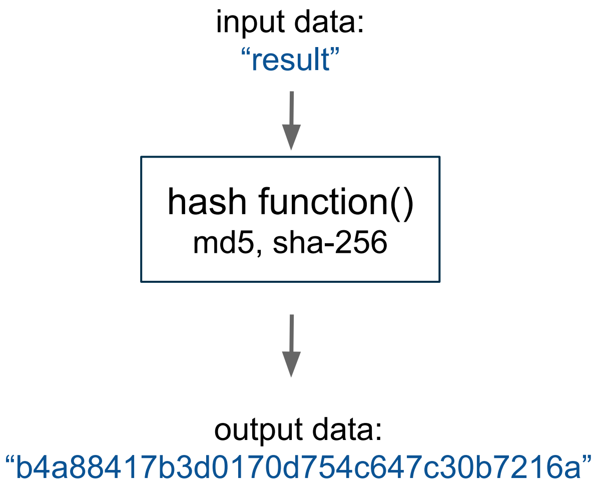
Blockchain

```
class Blockchain:
    def __init__(self):
        self.current_transactions = []
        self.chain = []
        self.nodes = set()

        # Create the genesis block
        self.new_block(previous_hash='1', proof=100)
```

```
    # EXAMPLE OF BLOCK OBJECT
    block = {
        'index': 1,
        'timestamp': 1506057125.900785,
        'transactions': [
            {
                'sender': "8527147fe1f5426f9dd545de4b27ee00",
                'recipient': "a77f5cdfa2934df3954a5c7c7da5df1f",
                'amount': 5,
            }
        ],
        'proof': 324984774000,
        'previous_hash':
"2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e730433629
38b9824"
    }
```

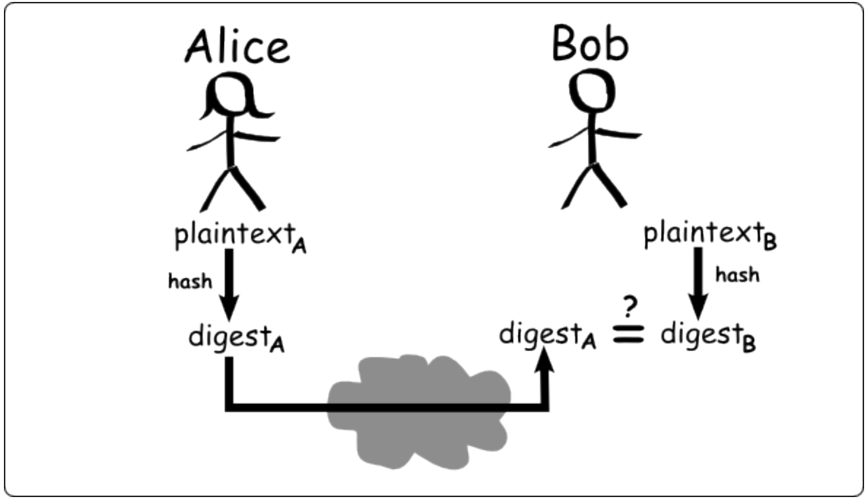
hash



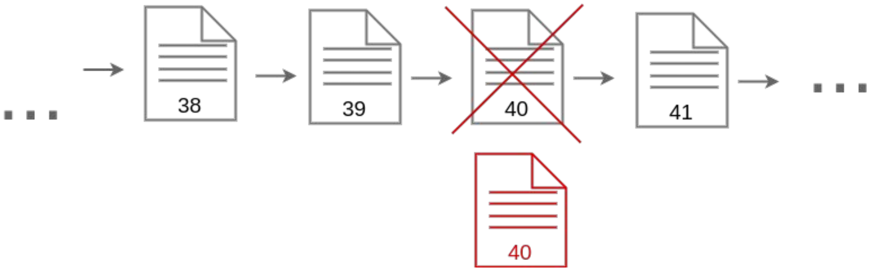
input data:
“result”

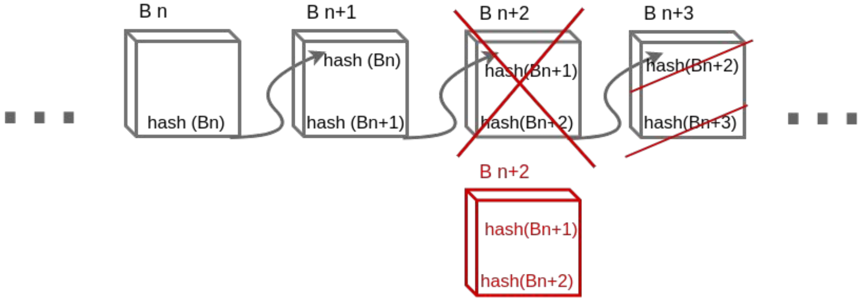
hash function()
md5, sha-256

output data:
“b4a88417b3d0170d754c647c30b7216a”



Blockchain immutability





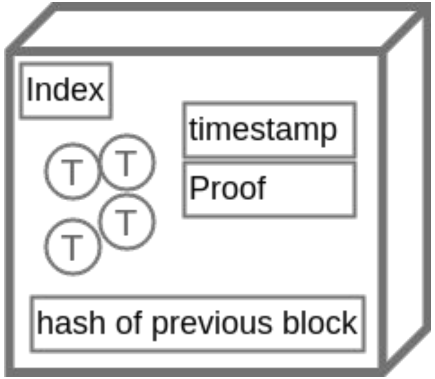
- **Integrity** of the book maintains intact.
- Nothing in the page **reflects the content** of that page.
- the pages **order** is implicit from the page numbers.

- **Integrity** of the chain is affected.
- Blocks are **ordered** by reference to previous block hashes,
- which **reflects content**.

Blockchain immutability

Methods or safeguards:

- make it very hard or **impossible** to rebuild a blockchain.
- differ based on the **block-adding mechanisms** and rules.



Two dominant schemes:

1. **Proof-of-work** (public blockchains) ex: Bitcoin

   Block valid if the hash follows a strict pattern. Increasing the **mining difficulty**.

2. **Specific signatures** (private blockchains) ex: Multichain

   The block-adding mechanism tends to be different.
   Block-adders

Proof of Work algorithm

The core idea behind Proof of Work:

- A PoW algorithm is how new Blocks are created or **mined** on the blockchain.
- The goal of PoW is to **discover a number** which solves a **problem.**
- The number must be **difficult to find** but **easy to verify**—computationally speaking—by anyone on the network.

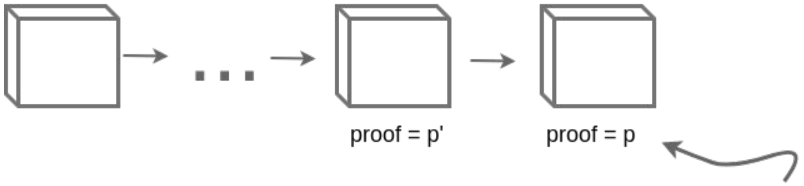
Example:
hash (x * y) = ac23dc...0
To simplify, fix x=5

```
x = 5
y = 0
while sha256(f'{x*y}'.encode()).hexdigest()[-1] != "0":
    y += 1
print(f'The solution is y = {y}')
```

Bitcoin PoW

- In Bitcoin the PoW algorithm is called **Hashcash** (https://en.wikipedia.org/wiki/Hashcash).
- Is the algorithm that **miners race** to solve in order to create a new block.
- In general the **difficulty is determined** by the number of characters searched for in a string.
- The miners are then **rewarded** for their solution by receiving a **coin** in a transaction
- The network is able to **easily verify** their solution.

Implementing our PoW



To mine this block, we need
to find proof p, so:
hash (p * p') = 0000 32998n9834.......

To **mine** a new block, is to obtain proof p of the new block.

The first miner to get it, is rewarded with a coin.

```
def proof_of_work(self, last_proof):
    proof = 0
    while self.valid_proof(last_proof, proof) is False:
        proof += 1

    return proof

@staticmethod
def valid_proof(last_proof, proof):
    guess = f'{last_proof}{proof}'.encode()
    guess_hash = hashlib.sha256(guess).hexdigest()
    return guess_hash[:4] == "0000"
```

Building our first Blockchain with Docker containers

1. Blockchain
   a. Blocks
   b. chain
   c. hash
   d. Transactions
   e. immutability
   f. Proof of work
   g. mine
2. **Interacting with the blockchain**
3. Consensus
   a. Distributed network
   b. Interacting with the network

Mining endpoint

The mining endpoint is where the magic happens. It has to do three things:

1. Calculate the Proof of Work
2. Reward the miner (us) by adding a transaction granting us 1 coin
3. Forge the new Block by adding it to the chain

```
@app.route('/mine', methods=['GET'])
def mine():
    last_block = blockchain.last_block
    last_proof = last_block['proof']
    proof = blockchain.proof_of_work(last_proof)

    blockchain.new_transaction( # as reward for finding the proof
        sender="0", # 0 means this node has mined a new coin
        recipient=node_identifier, # note the recipient of the mined block is the address of our node
        amount=1,
    )
    previous_hash = blockchain.hash(last_block) # add the block to the chain
    block = blockchain.new_block(proof, previous_hash)
    response = {
        'message': "New Block Forged",
        'index': block['index'],
        'transactions': block['transactions'],
        'proof': block['proof'],
        'previous_hash': block['previous_hash'],
    }
    return jsonify(response), 200
```

Transactions endpoint

Example of what the request for a transaction will look like.

It’s what the user sends to the server:

```
{
 "sender": "my address",
 "recipient": "someone else's address",
 "amount": 5
}
```

```
@app.route('/transactions/new', methods=['POST'])
def new_transaction(): # add a new transaction
    values = request.get_json()

    # Check that the required fields are in the POST'ed data
    required = ['sender', 'recipient', 'amount']
    if not all(k in values for k in required):
        return 'Missing values', 400

    # Create a new Transaction
    index = blockchain.new_transaction(values['sender'], values['recipient'], values['amount'])

    response = {'message': f'Transaction will be added to Block {index}'}
    return jsonify(response), 201
```

Chain endpoint

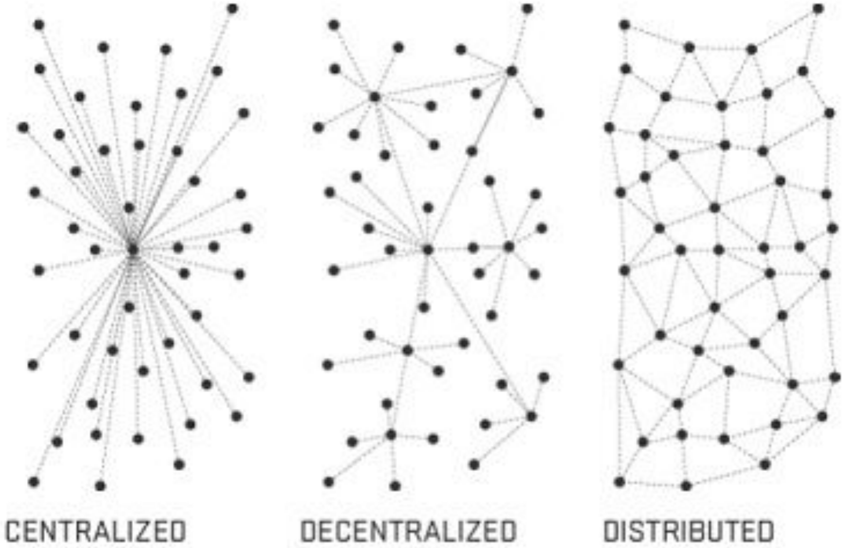
Return the whole chain in a specific node.

```
# Create the /chain endpoint, which returns the full Blockchain.
@app.route('/chain', methods=['GET'])
def full_chain():
   response = {
      'chain': blockchain.chain,
      'length': len(blockchain.chain),
   }
   return jsonify(response), 200
```

Building our first Blockchain with Docker containers

1. Blockchain
   a. Blocks
   b. chain
   c. hash
   d. Transactions
   e. immutability
   f. Proof of work
   g. mine
2. Interacting with the blockchain
3. **Consensus**
   a. Distributed network
   b. Interacting with the network

Distributed network



- Each node represents a personal computer.
- All the nodes in the network have the same complete copy of all the information stored in the network.
- Transactions can be verified against any node in the network.

Distributed network

If we want more than one node in our network:

- we need a way to let a node know about **neighbouring** nodes on the network.
- Each node on our network should keep a **registry of other nodes** on the network.

```
def register_node(self, address):

    parsed_url = urlparse(address)
    self.nodes.add(parsed_url.netloc)
```

```
def valid_chain(self, chain):
    last_block = chain[0]
    current_index = 1

    while current_index < len(chain):
      block = chain[current_index]
      print(f'{last_block}')
      print(f'{block}')
      print("\n------------\n")
      # Check that the hash of the block is correct
      if block['previous_hash'] != self.hash(last_block):
        return False

      # Check that the Proof of Work is correct
      if not self.valid_proof(last_block['proof'], block['proof']):
        return False

      last_block = block
      current_index += 1

    return True
```

Consensus algorithm

The blockchain should be **decentralized**.

And being decentralized, we need to ensure that **all nodes reflect the same chain**

This is called the **problem of Consensus.** To solve it, a consensus algorithm needs to be implemented.

```
def resolve_conflicts(self):
    neighbours = self.nodes
    new_chain = None

    max_length = len(self.chain)
    for node in neighbours:
        response = requests.get(f'http://{node}/chain')

        if response.status_code == 200:
            length = response.json()['length']
            chain = response.json()['chain']

            # Check if the length is longer and the chain is valid
            if length > max_length and self.valid_chain(chain):
                max_length = length
                new_chain = chain
    if new_chain:
        self.chain = new_chain
        return True

    return False
```

Interact with the blockchain network

```
@app.route('/nodes/register', methods=['POST'])
def register_nodes():
    values = request.get_json()

    nodes = values.get('nodes')
    if nodes is None:
        return "Error: Please supply a valid list of nodes", 400

    for node in nodes:
        blockchain.register_node(node)

    response = {
        'message': 'New nodes have been added',
        'total_nodes': list(blockchain.nodes),
    }
    return jsonify(response), 201
```

```
@app.route('/nodes/resolve', methods=['GET'])
def consensus():
    replaced = blockchain.resolve_conflicts()

    if replaced:
        response = {
            'message': 'Our chain was replaced',
            'new_chain': blockchain.chain
        }
    else:
        response = {
            'message': 'Our chain is authoritative',
            'chain': blockchain.chain
        }

    return jsonify(response), 200
```

Building our first Blockchain with Docker containers

Code (bitbucket)

@beatrizmrg

BLOCK
CHAIN