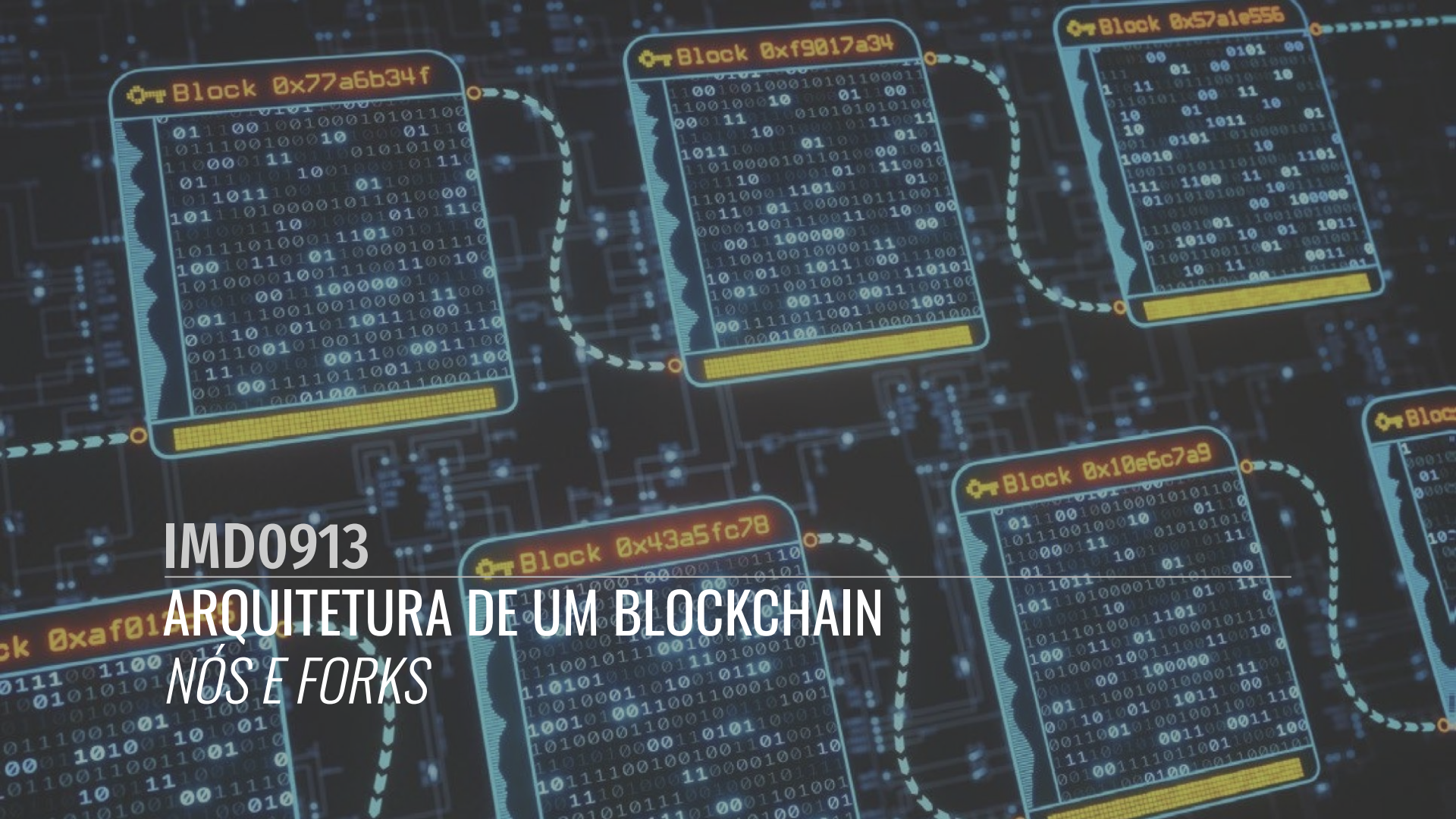
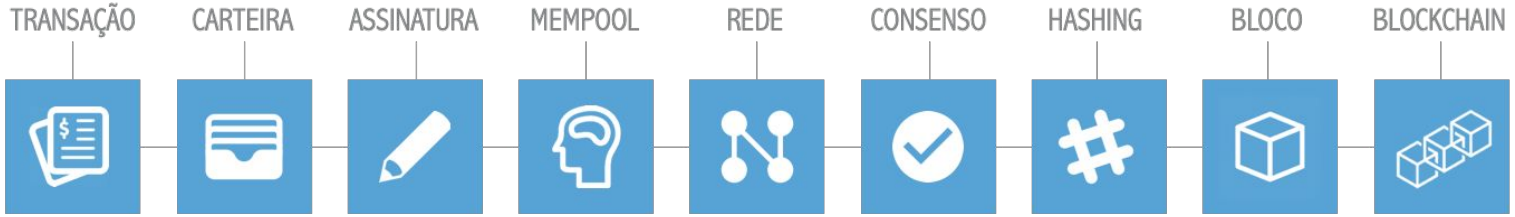


IMD0913

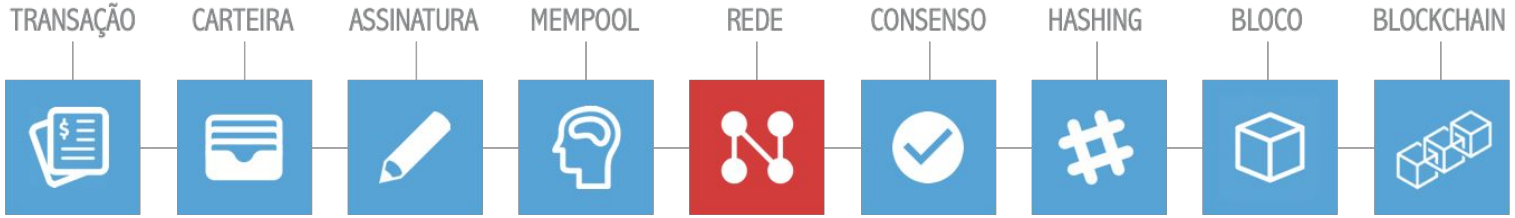
ARQUITETURA DE UM BLOCKCHAIN

NÓS E FORKS

ARQUITETURA DE UM **BLOCKCHAIN**



ARQUITETURA DE UM **BLOCKCHAIN**



Rede Bitcoin

Um *blockchain é* suportado por uma rede distribuída *peer-to-peer*

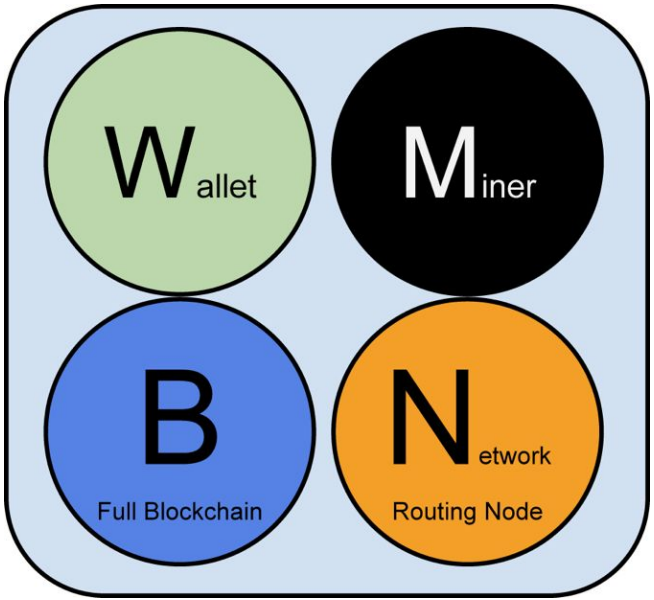
Não existe o papel de servidor

Não existe serviço centralizado

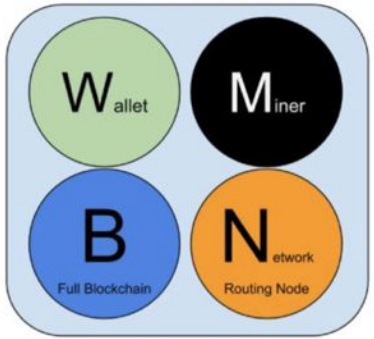
Não existe hierarquia na rede

Rede Bitcoin se refere a coleção de nós executando o protocolo P2P Bitcoin

Rede Bitcoin: Tipos e perfis de nós



Tipos de usuários



Nem todo cliente é minerador

E se eu não tiver um computador potente?

Nem todo cliente tem todo o *blockchain*

E se eu quiser enviar bitcoins do meu celular?

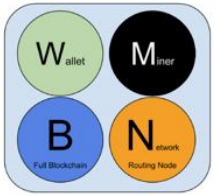
Nem todo cliente está diretamente conectado a rede Bitcoin

E se eu não preciso fazer transações regularmente?

Nem todo cliente tem um carteira

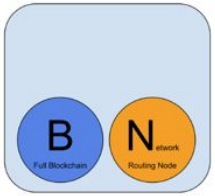
E se eu tiver um cliente de carteira separado?

Rede Bitcoin: Tipos de nós



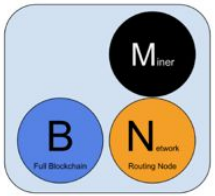
Cliente de referência (Bitcoin Core)

Contém uma Carteira (W), Minerador (M), o blockchain completo (B) e é um nó P2P da rede Bitcoin (N)



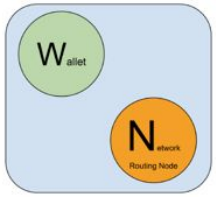
Nó full do blockchain

Contém o blockchain completo (B) e é um nó P2P da rede Bitcoin (N)



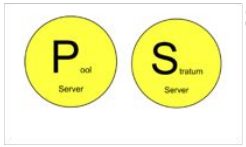
Minerador solo

Minerador (M) que contém o blockchain completo (B) e é um nó P2P da rede Bitcoin (N)



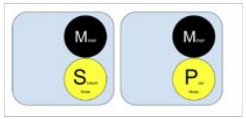
Carteira leve (SPV)

Contém uma carteira (W) e é um nó P2P da rede Bitcoin (N)



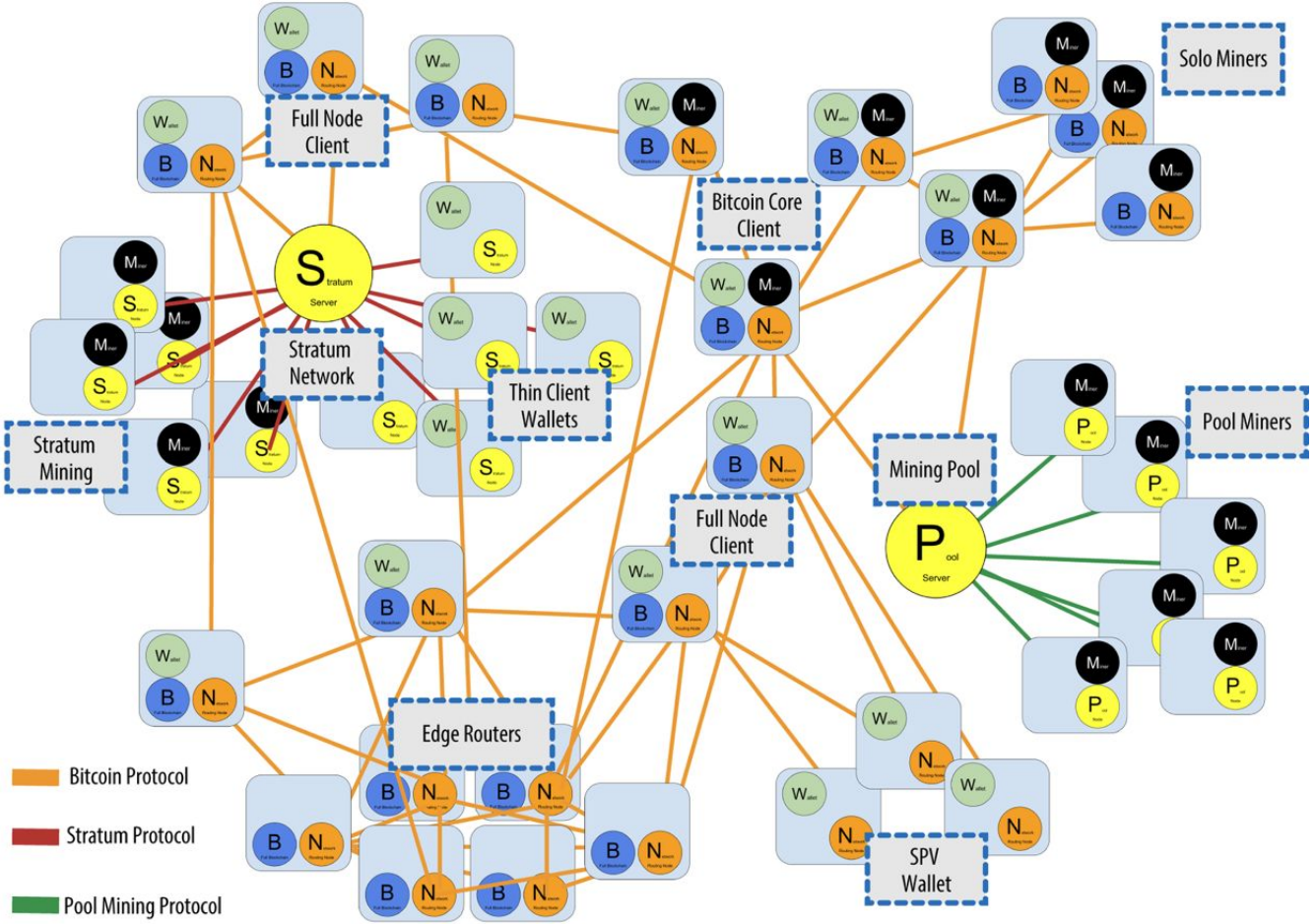
Servidores de protocolos de pool

Gateways conectando a rede P2P bitcoin aos nós que executam outros protocolos como nós de pool ou nós Stratum



Nós de mineração

Contém a função de mineração, sem o blockchain, com o protocolo Stratum (S) ou outro protocolo de nó de pool (P)


Full Node Client
Solo Miners
Bitcoin Core Client
Stratum Server
Stratum Network
Thin Client Wallets
Stratum Mining
Full Node Client
Mining Pool
Pool Server
Pool Miners
Edge Routers
SPV Wallet
Bitcoin Protocol
Stratum Protocol
Pool Mining Protocol

Rede Bitcoin

Quando um novo nó é iniciado, ele precisa descobrir nós bitcoins para se conectar!

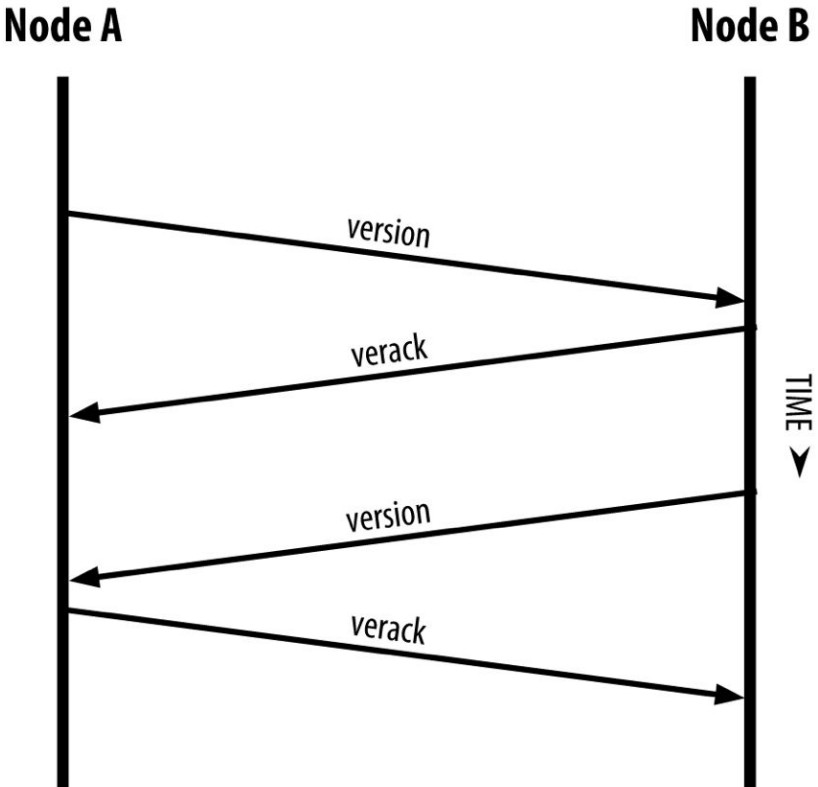
Porta TCP 8333

Opções:

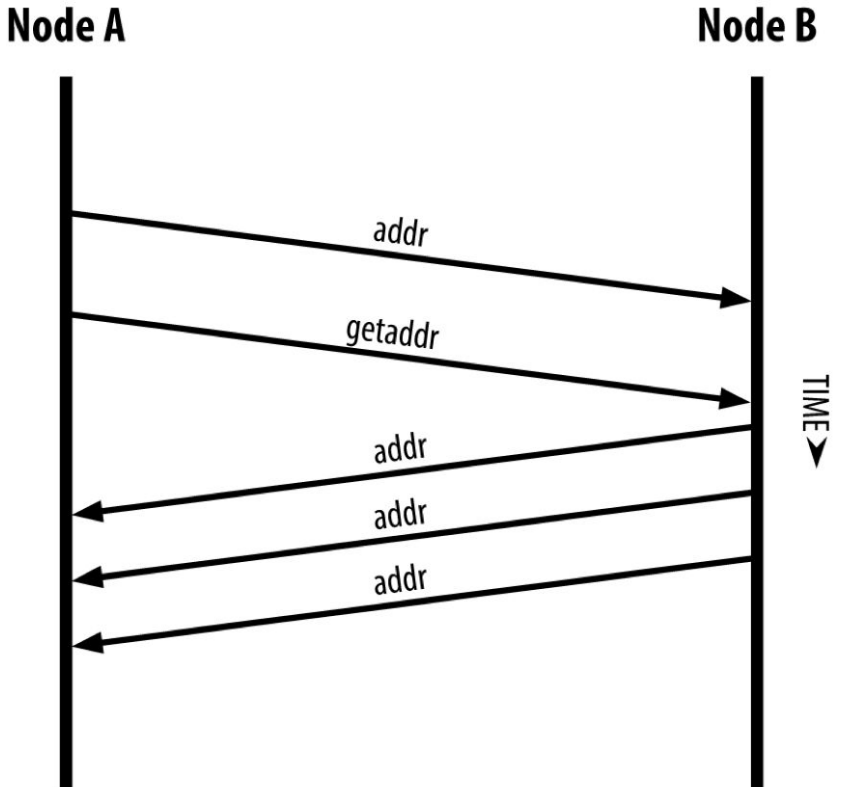
- Alguns servidores conhecidos...
- Ou indicar o endereço de um nó conhecido.

https://bitnodes.earn.com/

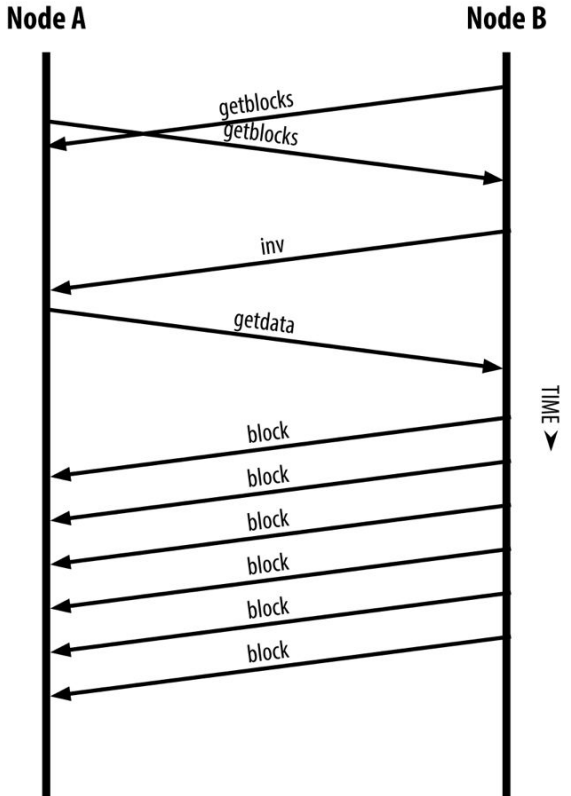
Rede Bitcoin



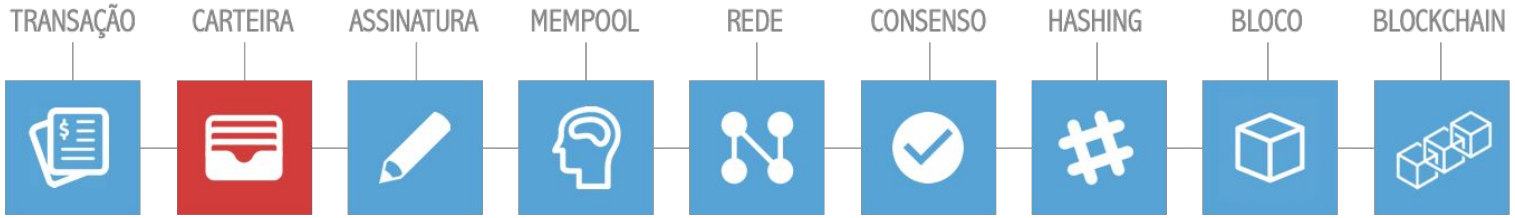
Rede Bitcoin



Rede Bitcoin



ARQUITETURA DE UM **BLOCKCHAIN**



Nós SPV

Nem todo nó armazena o *blockchain* completo

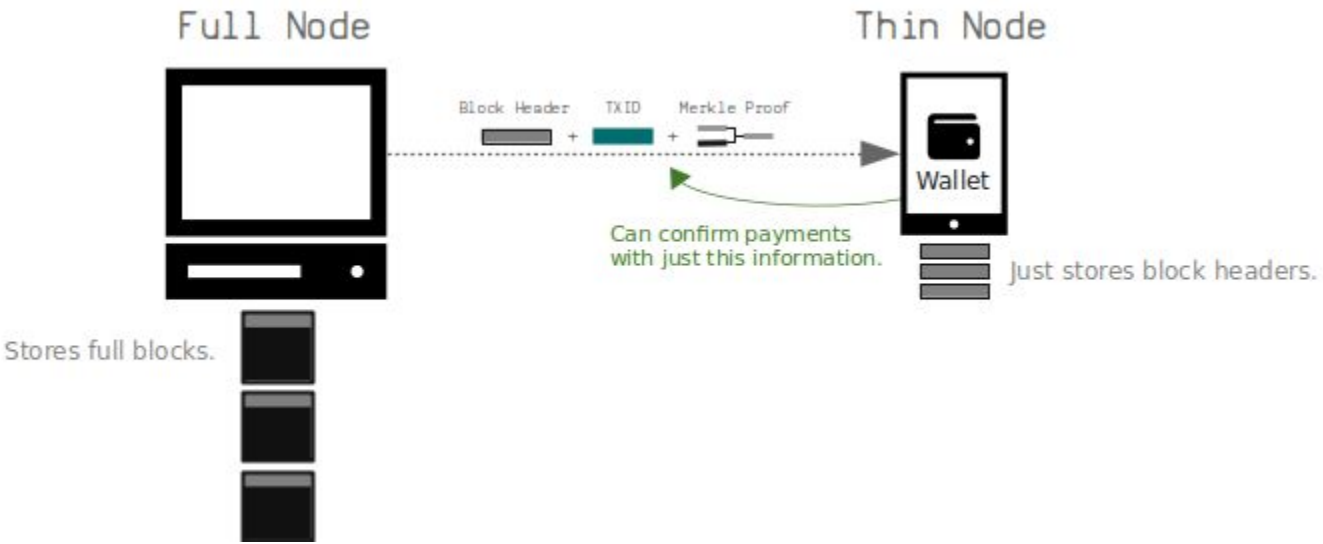
Por exemplo, seu smartphone!

Simple Payment Verification (SPV) é um método de verificar se determinada transação está incluída em um bloco sem precisar baixar o bloco completo

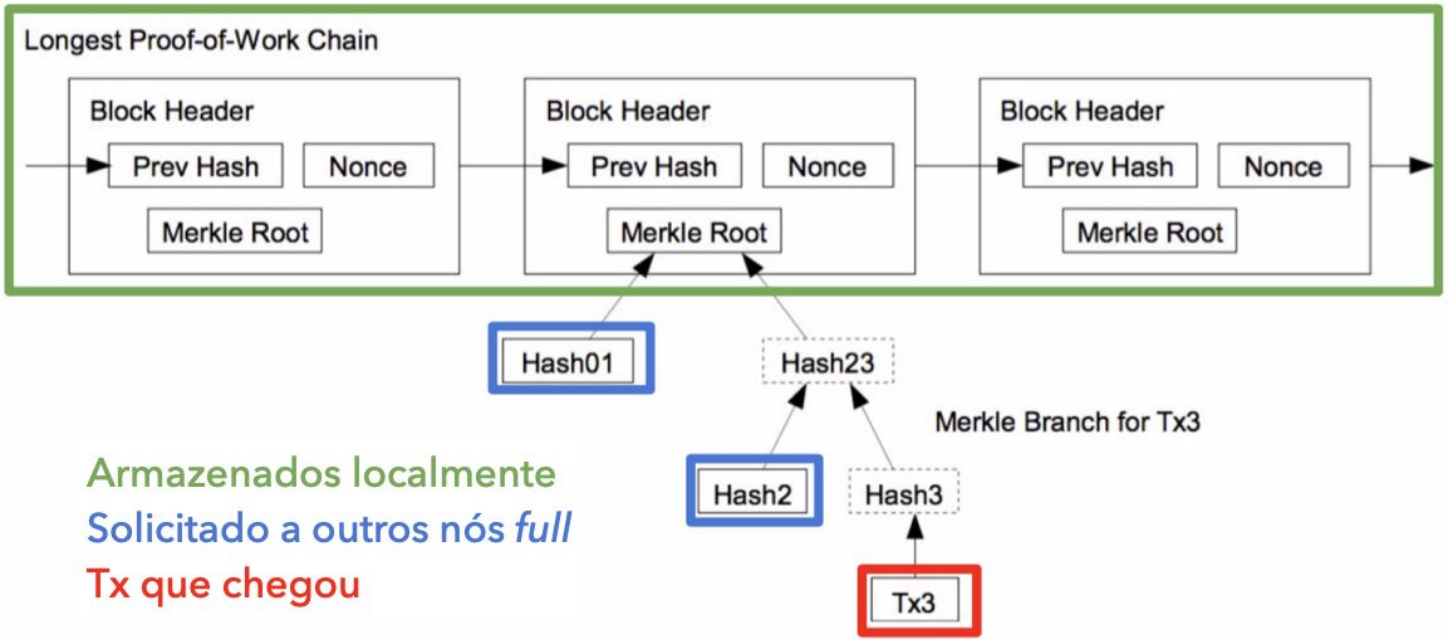
Baixa somente os cabeçalhos dos blocos (1000x menor)

Clientes **leves** (*lightweight*)

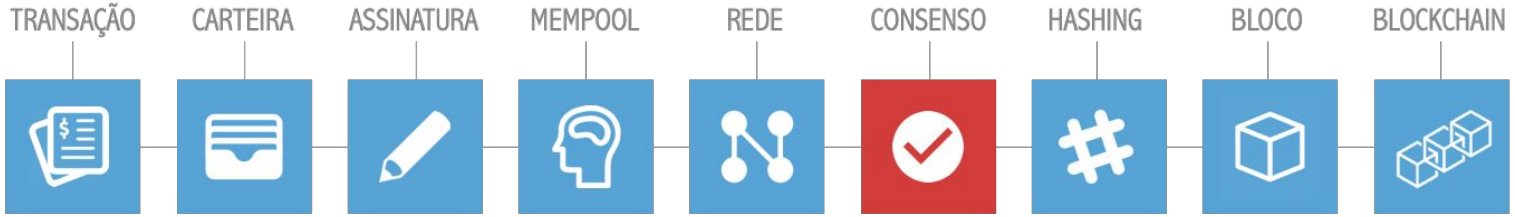
Nós SPV


Full Node
Thin Node
Block Header
TXID
Merkle Proof
Wallet
Can confirm payments
with just this information.
Just stores block headers.
Stores full blocks.

Nós SPV


Longest Proof-of-Work Chain
Block Header
Prev Hash
Nonce
Merkle Root
Block Header
Prev Hash
Nonce
Merkle Root
Block Header
Prev Hash
Nonce
Merkle Root
Hash01
Hash23
Merkle Branch for Tx3
Hash2
Hash3
Tx3
Armazenados localmente
Solicitado a outros nós full
Tx que chegou

ARQUITETURA DE UM **BLOCKCHAIN**



Processos que ocorrem em um nó

1. Verificação independente de cada transação, por todos os nós *full*, baseado em alguns critérios
2. Agregação independente de transações em um novo bloco, por nós mineradores, com a inclusão do PoW
3. Verificação independente de novos blocos por todos os nós e inclusão no *blockchain*
4. Seleção independente, por todos os nós, do *blockchain* mais longo e válido

1. Verificação independente de cada transação

Checklist de critérios:

Sintaxe e estrutura de dados corretos;

Lista de *inputs* e *outputs* não vazios;

Rejeita se soma das entradas for menor que a soma das saídas;

Os *unlocking scripts* de cada entrada deve validar os *locking scripts* das saídas correspondentes;

...

2. Agregação independente de transações em um novo bloco

Após validar uma transação, um nó a inclui no *mempool*

Nós mineradores começam a construir um bloco candidato e iniciam a busca da solução do PoW

Se outro bloco chegar, elimina as transações incluídas, remove do *mempool*, e começa a trabalhar em outro bloco candidato

Incluir a transação *coinbase* para o endereço do próprio minerador

Recompensa atual + Tx *fees*

2. Agregação independente de transações em um novo bloco


```
{
    "version" : 2,
    "merkleroot" : "c91c008c26e50763e9f548bb8b2fc323735f73577effbc55502c51eb4cc7cf2e",
    "tx" : [
        "d5ada064c6417ca25c4308bd158c34b77e1c0eca2a73cda16c737e7424afba2f",
        "b268b45c59b39d759614757718b9918caf0ba9d97c56f3b91956ff877c503fbe",

        ... 417 outras transações ...

       ],
    "time" : 1388185914,
    "nonce" : 924591752,
    "bits" : "1903a30c",
    "previousblockhash" : "0000000000000002a7bbd25a417c0374cc55261021e8a9ca74442b01284f0569"
}
```

2. Agregação independente de transações em um novo bloco


```
{
    "txid" : "d5ada064c6417ca25c4308bd158c34b77e1c0eca2a73cda16c737e7424afba2f",
    "version" : 1,
    "locktime" : 0,
    "vin" : [
        {
            "coinbase" : "03443b0403858402062f503253482f",
            "sequence" : 4294967295
        }
    ],
    "vout" : [
        {
            "value" : 25.09094928,
            "n" : 0,
            "scriptPubKey" : {
                "asm" : “02aa970c592640d19de03ff6f329d6fd2eecb023263b9ba5d1b81c29b523da8b21 OP_CHECKSIG",
                "hex" : "2102aa970c592640d19de03ff6f329d6fd2eecb023263b9ba5d1b81c29b523da8b21ac",
                "reqSigs" : 1,
                "type" : "pubkey",
                "addresses" : [
                    "1MxTkeEP2PmHSMze5tUZ1hAV3YTKu2Gh1N"
                ]
            }
        }
    ]
}
```

3. Verificação independente de novos blocos

Checklist de critérios:

A estrutura de dados do bloco é sintaticamente válida

O *hash* do cabeçalho do bloco é menor que o alvo (PoW)

O *timestamp* do bloco é maior que a média dos *timestamps* do últimos 11 blocos e menor que 2h no futuro

Tamanho do bloco é aceitável dentro dos limites

A primeira transação é a *coinbase*

Todas as transações dentro do bloco são válidas conforme critérios vistos em (1)

4. Seleção independente do *blockchain* mais longo e válido

Nós mantém três conjuntos de blocos:

- os conectados ao *blockchain* principal
- aqueles que formam *branches* do *blockchain* principal (*blockchains* secundárias)
- blocos que não tem um pai conhecido pelo nó (orfão)

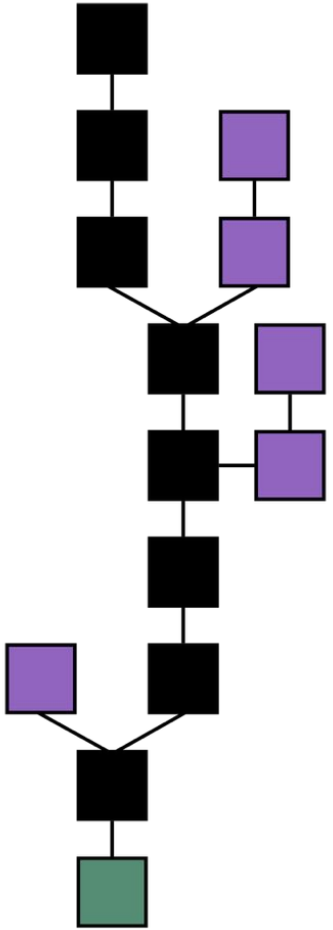
Forks do *blockchain*

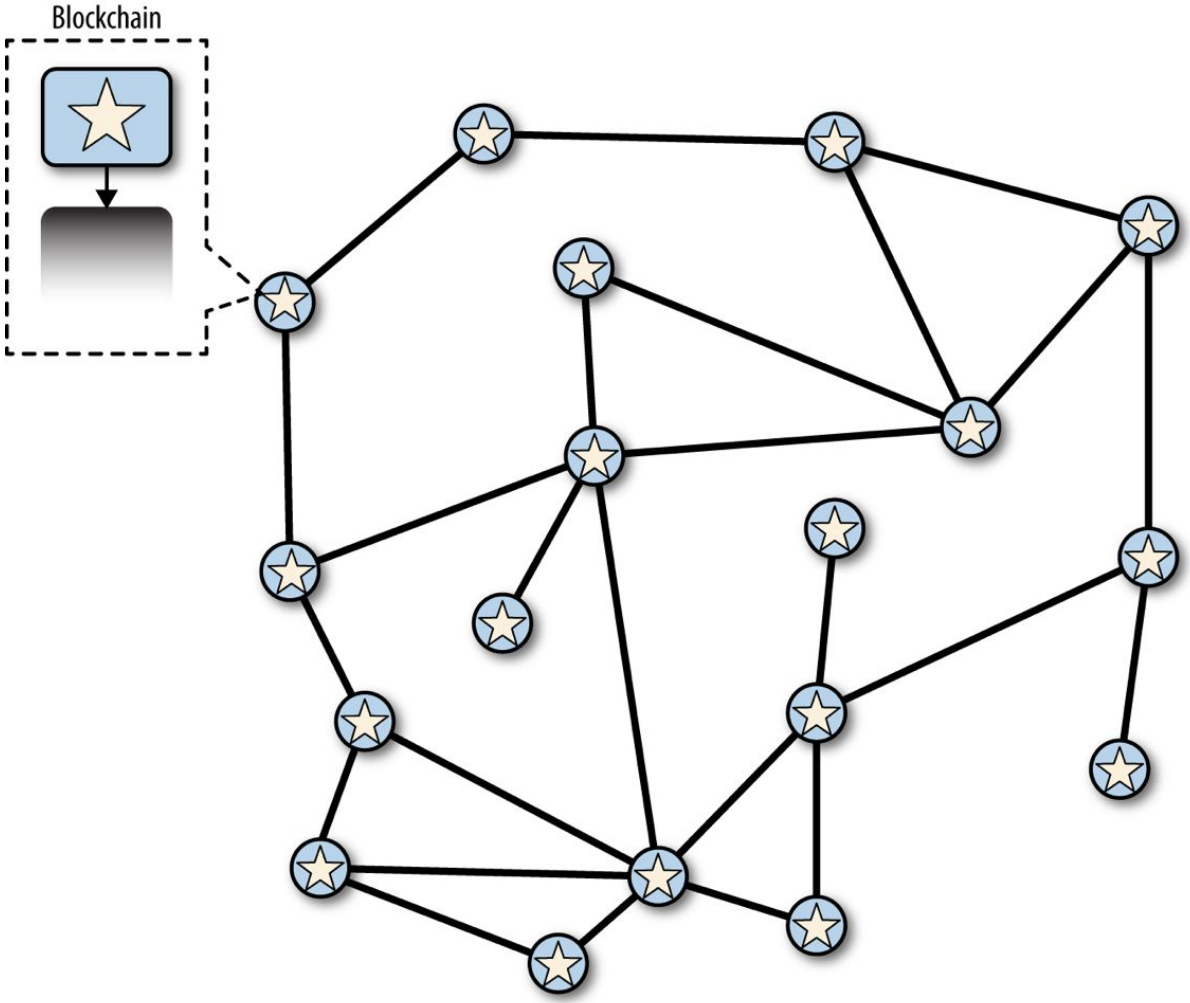
Como é uma estrutura de dados descentralizada, diferentes cópias do *blockchain* podem não ser consistentes

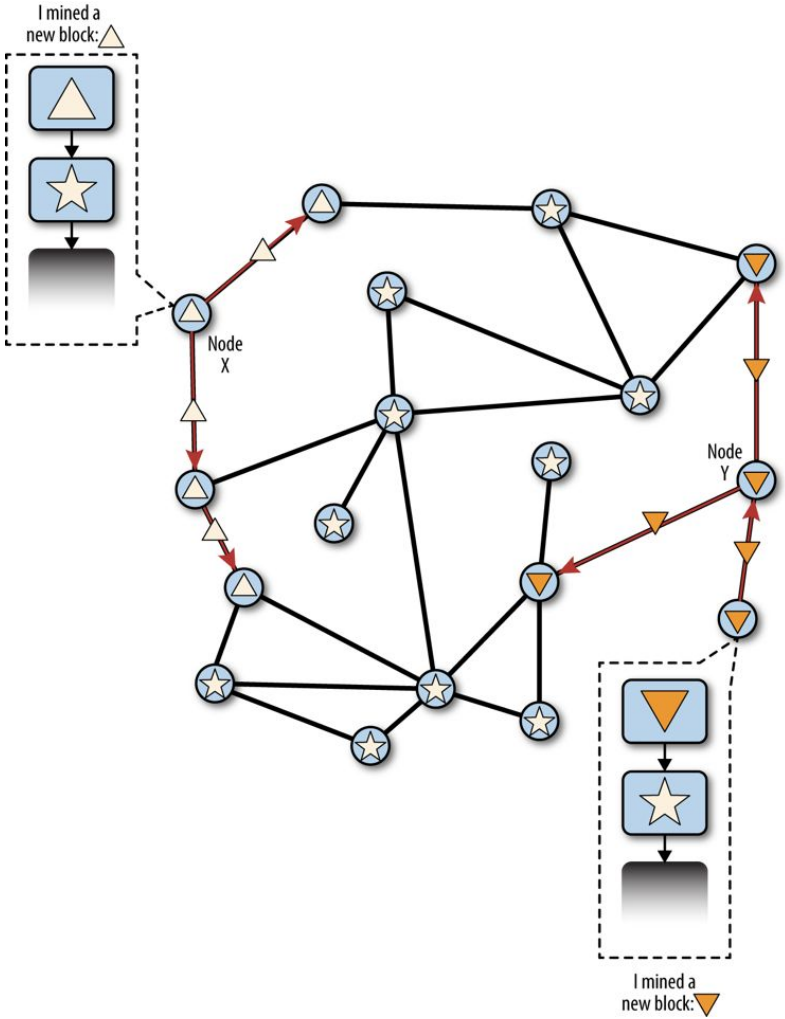
Forks ocorrem como inconsistências temporárias entre versões diferentes do *blockchain*, que serão resolvidas eventualmente através da reconvergência

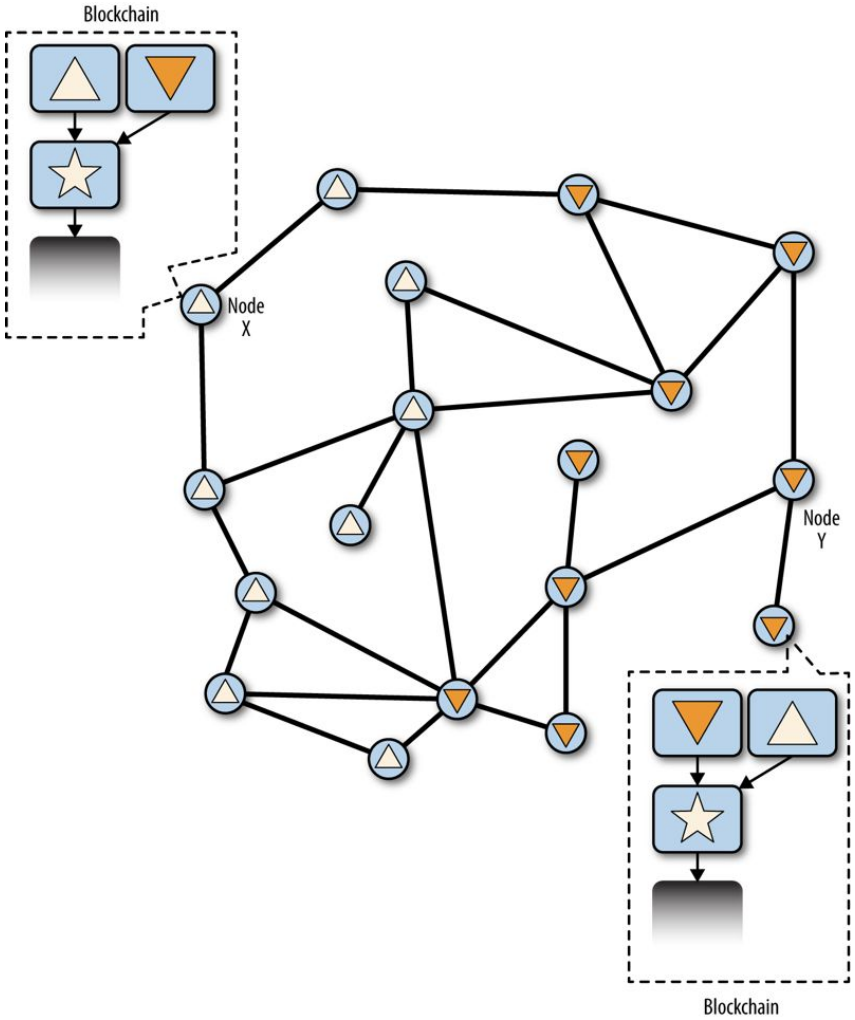
Isso é diferente dos *forks* induzidos! Veremos isso em breve!

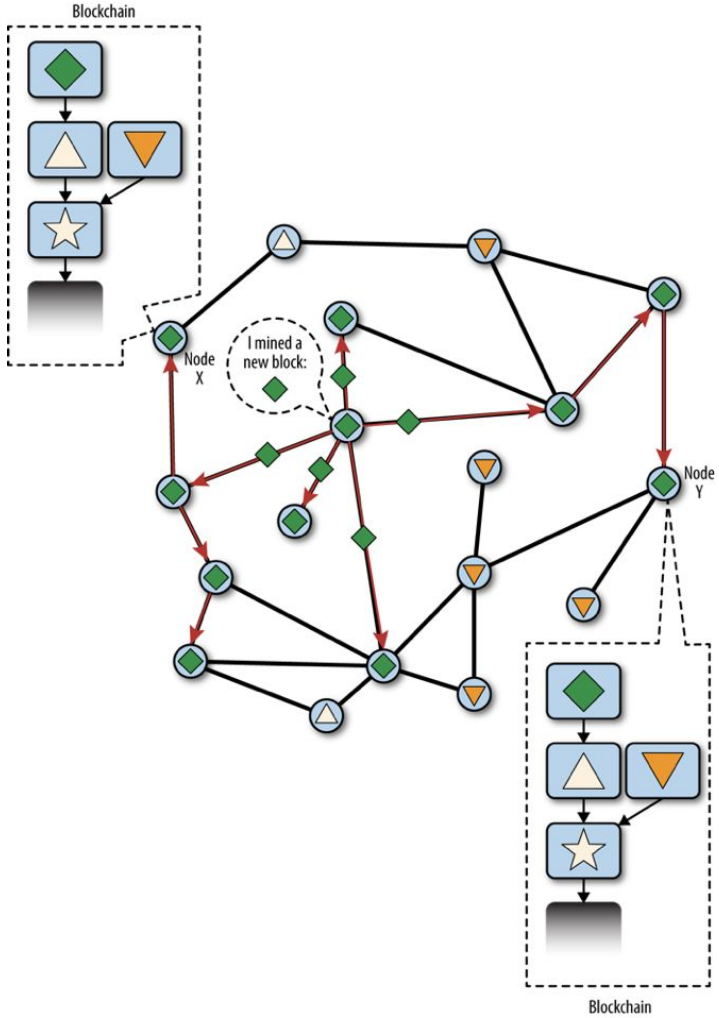
É corrigido após a reorganização da cadeia (***reorg***)

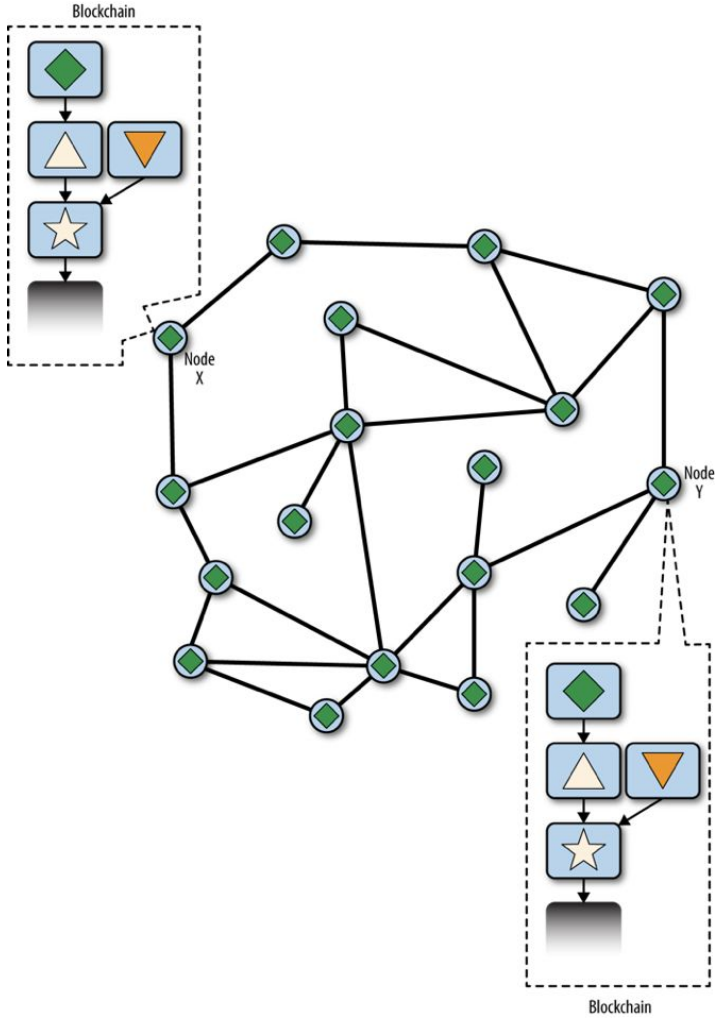



Blockchain

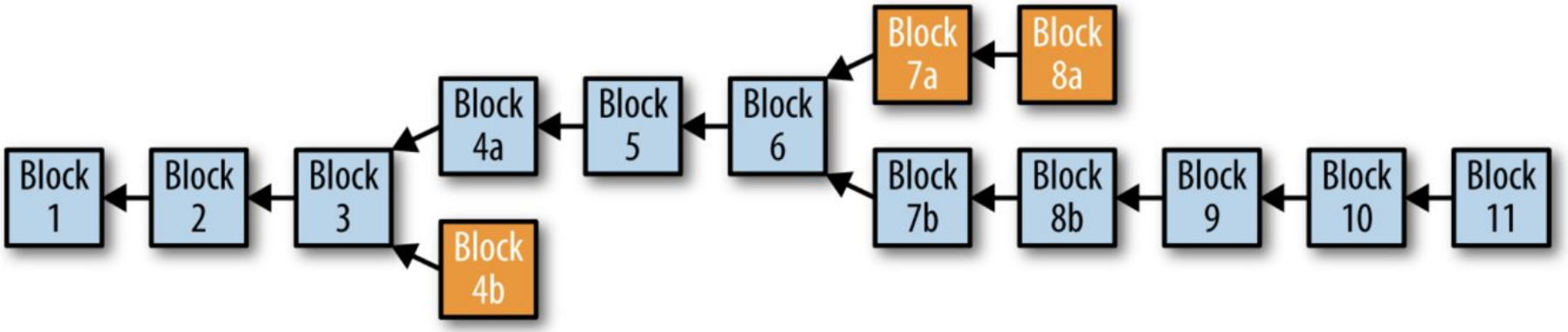

I mined a
new block:
Node
X
Node
Y
I mined a
new block:


Blockchain
Node
X
Node
Y
Blockchain

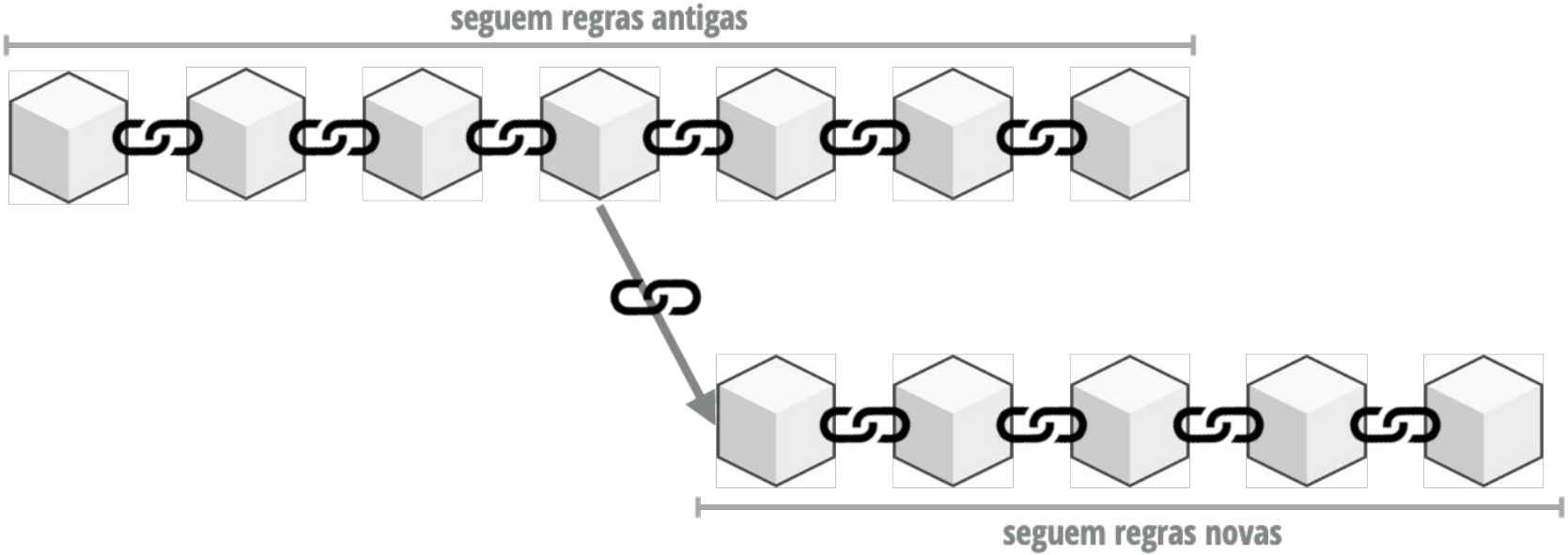

Blockchain
Node X
I mined a new block:
Node Y
Blockchain


Blockchain
Node
X
Node
Y
Blockchain

Forks do *blockchain*

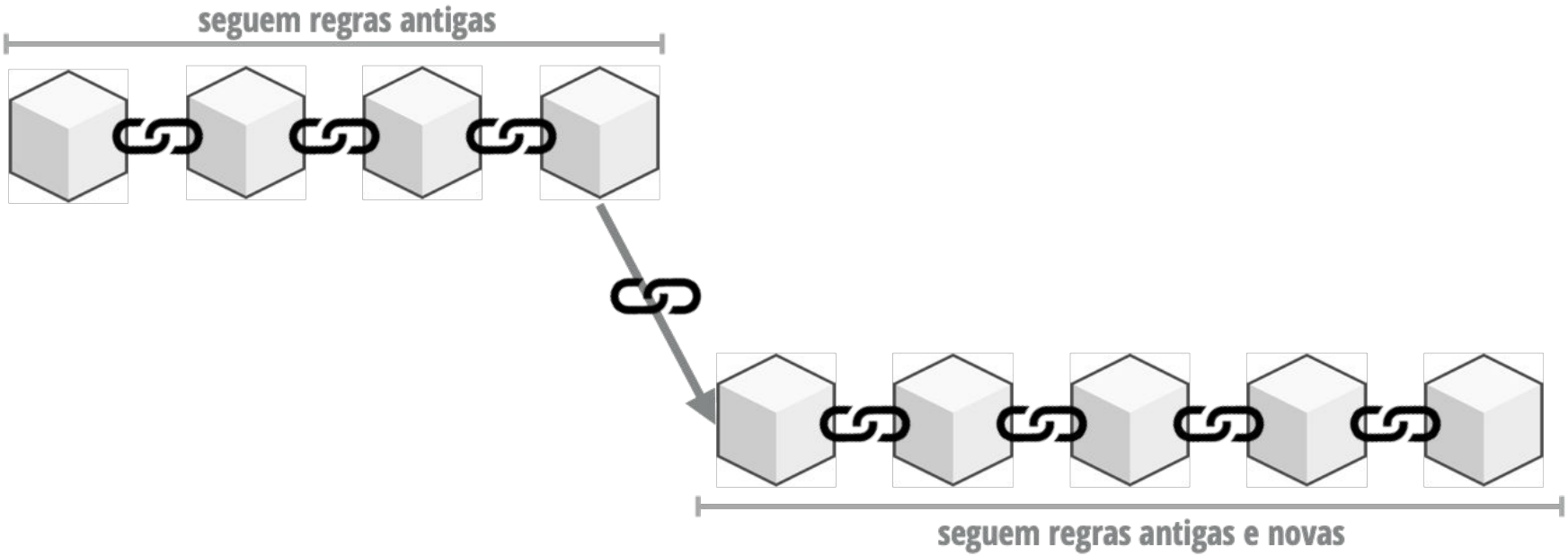


Fork induzido: *hard-fork*



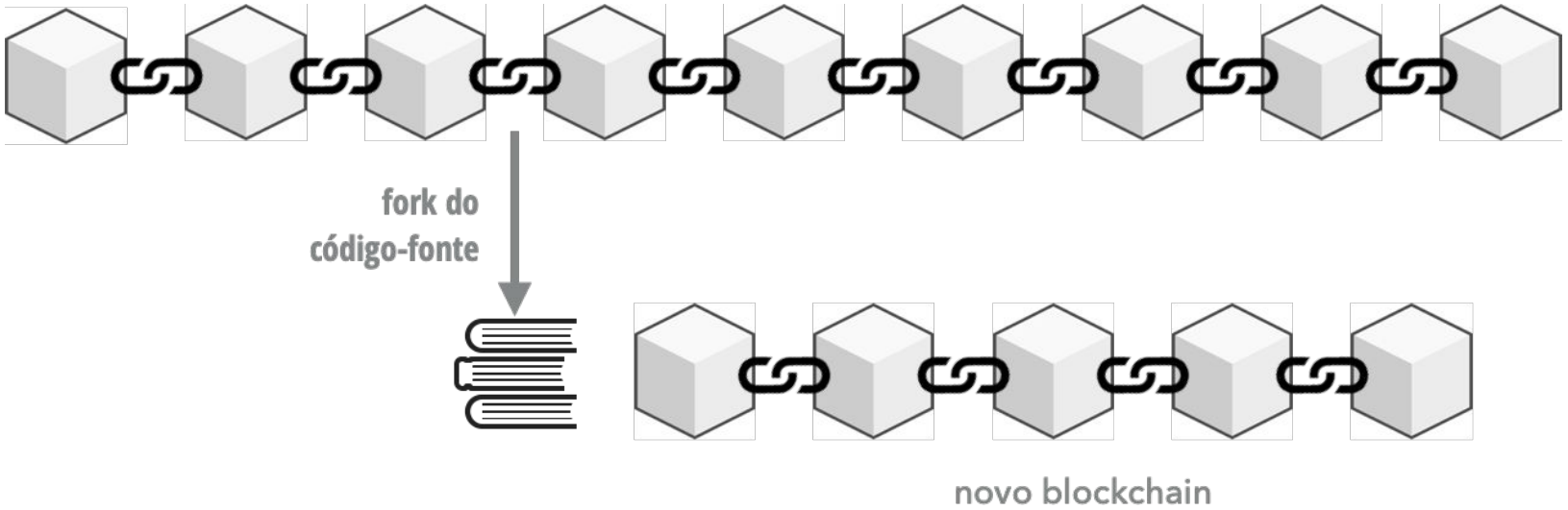
Hard fork: nós não-atualizados rejeitam transações e blocos com novas regras, gerando um *blockchain* divergente

Fork induzido: *soft-fork*



Soft fork: blocos violando novas regras se tornam obsoletos pela maioria de mineradores atualizados

Fork induzido: *source-code fork*



Fork do repositório Git para implementar um blockchain completamente novo

Exemplo: LiteCoin

Finalizando nosso blockchain em Python...

`/06-consensus`

Esta atividade tem como objetivo implementar uma API de acesso ao nosso *blockchain*. Isso permitirá a **interação** entre múltiplos nós que implementem nosso protocolo. Outro objetivo desta atividade é definir o **modelo de consenso**.

- A entrega será realizada no Github Classroom mas a avaliação será feita no formato de apresentação ao professor;
- O trabalho deve ser desenvolvido **individualmente ou em dupla** (O GitHub Classroom gerencia as duplas);
- Plágios não serão tolerados, resultando em nota zero para todos os envolvidos.

Requisitos **`06-consensus`**

Sua API precisará implementar 5 *end-points*:

1. **`[POST] /transactions/create`** para criar uma nova transação a ser incluída no próximo bloco. No corpo da requisicão HTTP, usando POST, inclua as informações necessárias para criação de uma nova transação.
2. **`[GET] /transactions/mempool`** para retornar a *memory pool* atual do nó.
3. **`[GET] /mine`** para informar o nó para criar e minerar um novo bloco. Ou seja, um nó que for requisitado a partir desse *end-point* deve pegar todas as transações incluídas em seu *memory pool*, montar um bloco e minera-lo.
4. **`[GET] /chain`** para retornar o *blockchain* completo daquele nó.
5. **`[POST] /nodes/register`** para aceitar uma lista de novos nós no formato de URLs. Note que já existe uma variável do tipo conjunto (*set*) chamado nodes para armazenar os nós registrados.
6. **`[GET] /nodes/resolve`** para executar o modelo de consenso, resolvendo conflitos e garantindo que contém a cadeia de blocos correta. Basicamente o que deve ser feito pelo nó é solicitar a todos os seus nós registrados os seus respectivos *blockchains*. Então deve-se conferir se o *blockchain* é válido, e, se for maior (mais longo) que o atual, deve substitui-lo.

Requisitos **06-consensus**

Implemente os métodos `isValidChain()` e `resolveConflicts()`:

```
def isValidChain(self, chain):
        '''
        Dado uma chain passada como parâmetro, faz toda a verificação no blockchain
        se cada uma dos blocos é válido:
         1. PoW válido
         2. Todas as transações assinadas e válidas
         3. Merkle Root válido
         4. Hash do bloco anterior válido
        Retorna True se validado, False caso contrário.
        '''
```

```
def isValidChain(self, chain):
        '''
        Consulta todos os nós registrados, e verifica se algum outro nó tem um blockchain mais
comprido e válido. Em caso positivo, substitui seu próprio chain.
        '''
```

Requisitos `06-consensus`

Utilize qualquer *framework* que desejar para implementar a API. Uma sugestão é o *framework* **Flask**, bastante leve e de fácil utilização. Instale usando o ***pip***. Veja como é simples criar um *end-point*:

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

@app.route('/hello', methods=['GET'])
def hello():
    return "Hello World!"

if __name__ == '__main__':
    app.run(port=5000)
```

Se eu executar esse código, temos um serviço *web* em execução. Caso entre em `http://127.0.0.1:5000/hello`, o método `hello()` será executado!

Requisitos `06-consensus`

Caso precise fazer requisições HTTP no Python, você pode utilizar o módulo ***requests***. Também é bem simples, por exemplo, para requisitar o blockchain completo do nó **127.0.0.1:5001**:

```
import requests

response = requests.get('http://127.0.0.1:5001/chain')
obj = response.json()
```

Para testar, será necessário executar no mínimo dois nós simultaneamente, e no caso de ser na mesma máquina, as instâncias em execução devem usar portas diferentes (por exemplo, porta 5000 e 5001). Você pode testar no seu navegador, usando *curl*, ou então usando o *Postman* ou *Insomnia*.

Apresentação **`06-consensus`**

Serão realizadas nos dias **26 e 31 de outubro**, pela **dupla**:

```
[  ] Sobe um primeiro nó (ex: porta 5001)
[  ] Sobe um segundo nó (ex: porta 5002)
[  ] Cria uma nova transação (tx) no nó #1
[  ] Confere o mempool do nó #1
[  ] Cria e minera um novo bloco no nó #1
[  ] Cria e minera outro bloco (sem transações) no nó #1
[  ] Confere a atual chain do nó #1
[  ] Registra o nó #2 no nó #1
[  ] Resolve (consenso) o nó #1 (o blockchain não deve mudar)
[  ] Cria e minera um único bloco (sem transações) no nó #2
[  ] Confere a atual chain do nó #2
[  ] Registra o nó #1 no nó #2
[  ] Resolve (consenso) o nó #2 (o blockchain deve mudar para a chain do nó #1)
[  ] Confere a atual chain do nó #2
```

Apresentação **`06-consensus`**

PASSO 1: *Sobe um primeiro nó (ex: porta 5001)*

Apresentação **`06-consensus`**

PASSO 2: *Sobe um segundo nó (ex: porta 5002)*

Apresentação **06-consensus**

PASSO 3: *Cria uma nova transação (tx) no nó #1*

[POST] /transactions/create

Passar no body as informações necessárias
para criar a tx:
  - sender, recipient, value
  - chave privada (!!!)

Apresentação **06-consensus**

PASSO 4: *Confere o mempool do nó #1*

[GET] /transactions/mempool

Espera-se retornar uma única tx

Apresentação **06-consensus**

PASSO 5: *Cria e minera um novo bloco no nó #1*

[GET] /mine

Cria o bloco, inclui todas as tx do mempool para o bloco e o minera.

Apresentação **06-consensus**

PASSO 6: *Cria e minera outro bloco (sem transações) no nó #1*

[GET] /mine

Cria o bloco, inclui todas as tx do mempool para o bloco (que está vazio) e o minera.

Apresentação **06-consensus**

PASSO 7: *Confere a atual chain do nó #1*

[GET] /chain

Deve-se esperar um blockchain com 3 blocos:

- #0 - genesis
- #1 - bloco 1 com 1 tx
- #2 - bloco 2 com 0 tx

Apresentação **06-consensus**

PASSO 8: *Registra o nó #2 no nó #1*

[POST] /nodes/register

Passar no body a info do nó #2. Ex:
  http://localhost:5002

Apresentação **06-consensus**

PASSO 9: *Resolve (consenso) o nó #1 (o blockchain não deve mudar)*

chain: [#0, #1, #2]

chain: [#0]

[GET] /nodes/resolve

Deve solicitar a chain de todos os nós registrados (no caso, a do :5002), fazer toda a validação, e substituir caso seja válido e maior que o atual.

Neste cenário, ela não será substituída.

Apresentação **06-consensus**

PASSO 10: *Cria e minera um único bloco (sem transações) no nó #2*

chain: [#0, #1, #2]

chain: [#0, #1]

[GET] /mine

Cria o bloco, inclui todas as tx do mempool para o bloco (que está vazio) e o minera.

Apresentação **06-consensus**

PASSO 11: *Confere a atual chain do nó #2*

chain: [#0, #1, #2]

chain: [#0, #1]

[GET] /chain

Deve-se esperar um blockchain com 2 blocos:
  #0 - genesis
  #1 - bloco 1 com 0 tx

Apresentação **06-consensus**

PASSO 12: *Registra o nó #1 no nó #2*

chain: [#0, #1, #2]

chain: [#0, #1]

[POST] /nodes/register

Passar no body a info do nó #1. Ex:
http://localhost:5001

Apresentação **06-consensus**

PASSO 13: *Resolve (consenso) o nó #2 (o blockchain deve mudar para a chain do nó #1)*

`chain: [#0, #1, #2]`

`chain: [#0, #1] → [#0, #1, #2]`

[GET] /nodes/resolve

Deve solicitar a chain de todos os nós registrados (no caso, a do :5001), fazer toda a validação, e substituir caso seja válido e maior que o atual.

Neste cenário, ela será substituída.

Apresentação **06-consensus**

PASSO 14: *Confere a atual chain do nó #2*

`chain: [#0, #1, #2]`

`chain: [#0, #1, #2]`

[GET] /chain

```
Deve-se esperar um blockchain com 3
blocos:
  #0 - genesis
  #1 - bloco 1 com 1 tx
  #2 - bloco 2 com 0 tx
```