MIDDLEWARE SYSTEMS
RESEARCH GROUP
MSRG.ORG

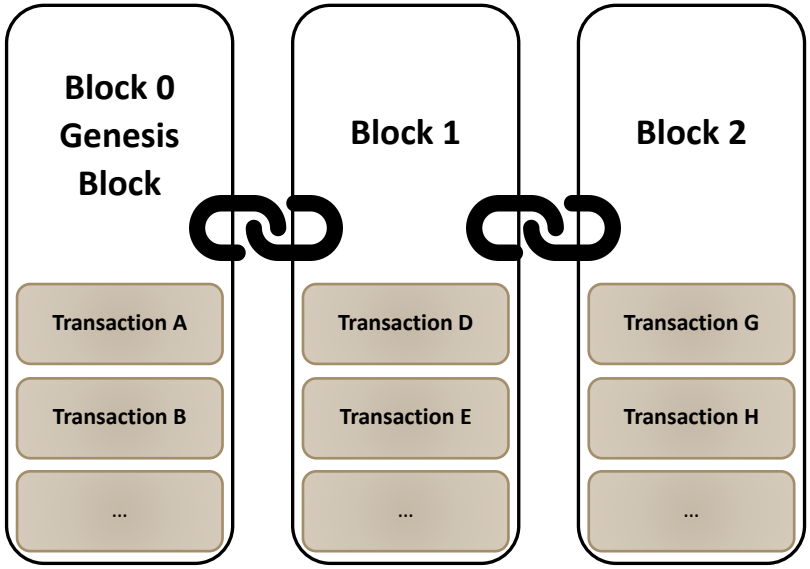
# Deconstructing Blockchains: Concepts, Systems and Applications

## INTRODUCTION

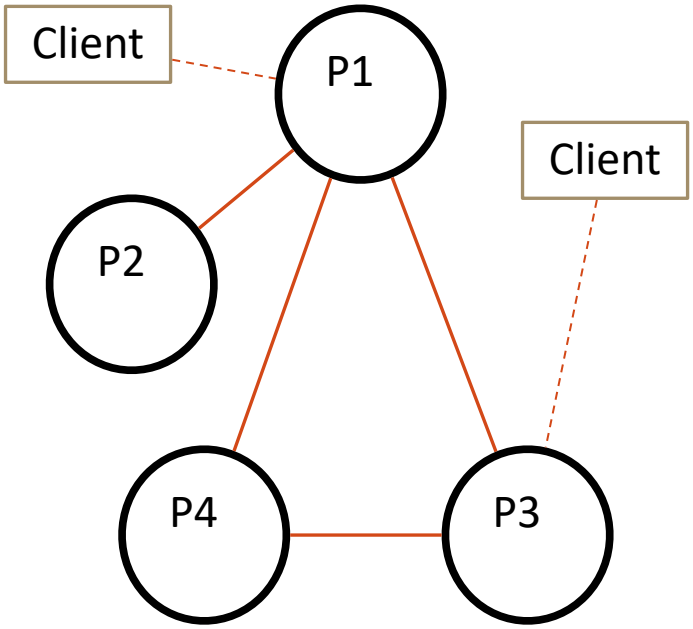
# Understanding Blockchains

# Blockchain 101: Distributed Ledger Technology (DLT)

**Blockchain Data Structure**

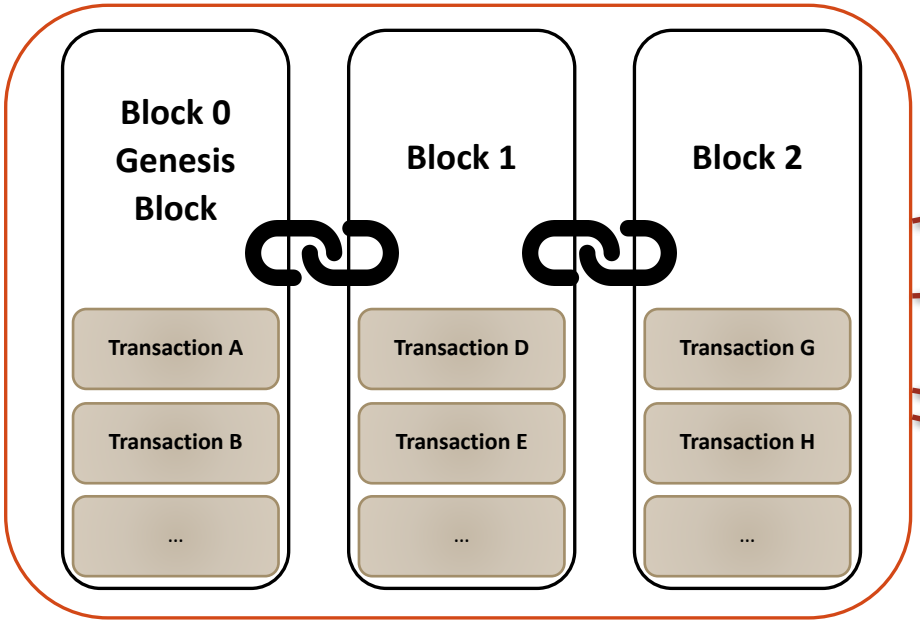



**Peer-to-Peer Network**

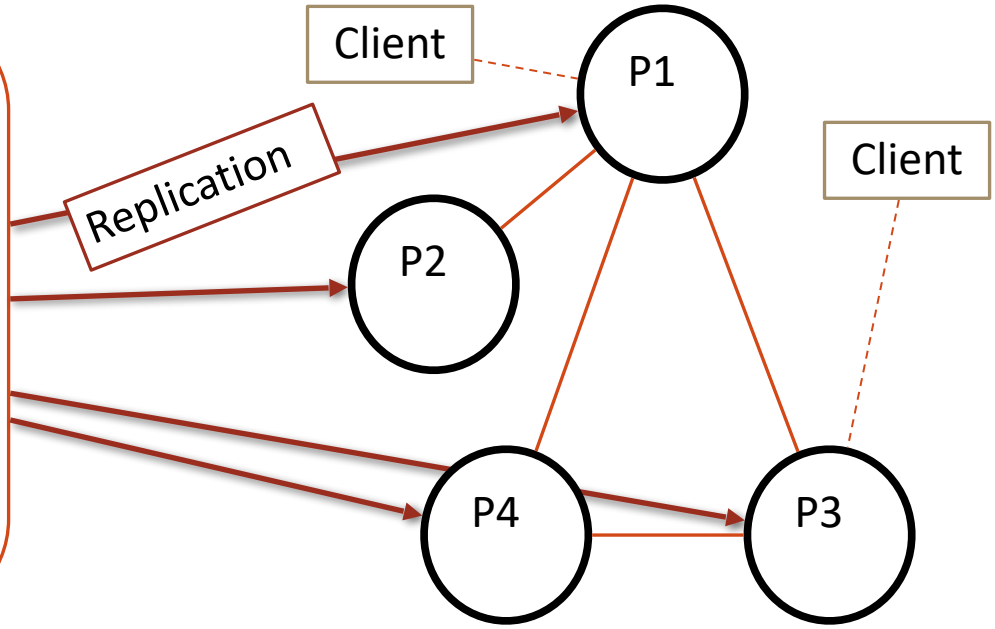


# Blockchain 101: Distributed Ledger Technology (DLT)

**Blockchain Data Structure**



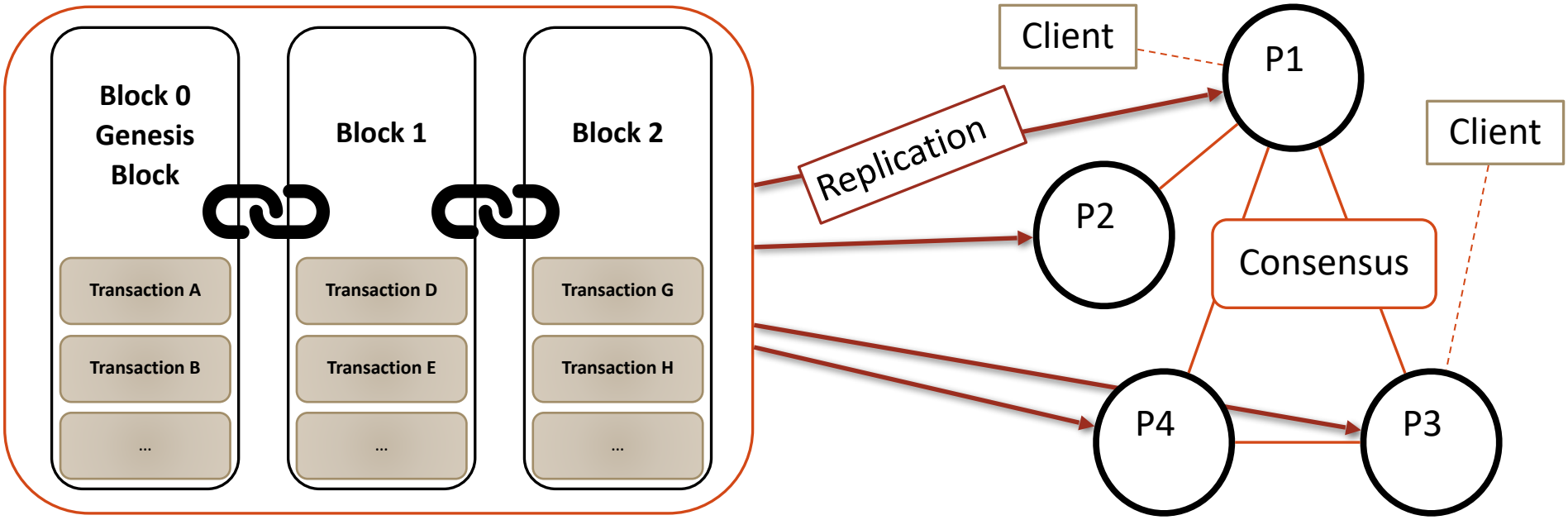

**Peer-to-Peer Network**



# Blockchain 101: Distributed Ledger Technology (DLT)

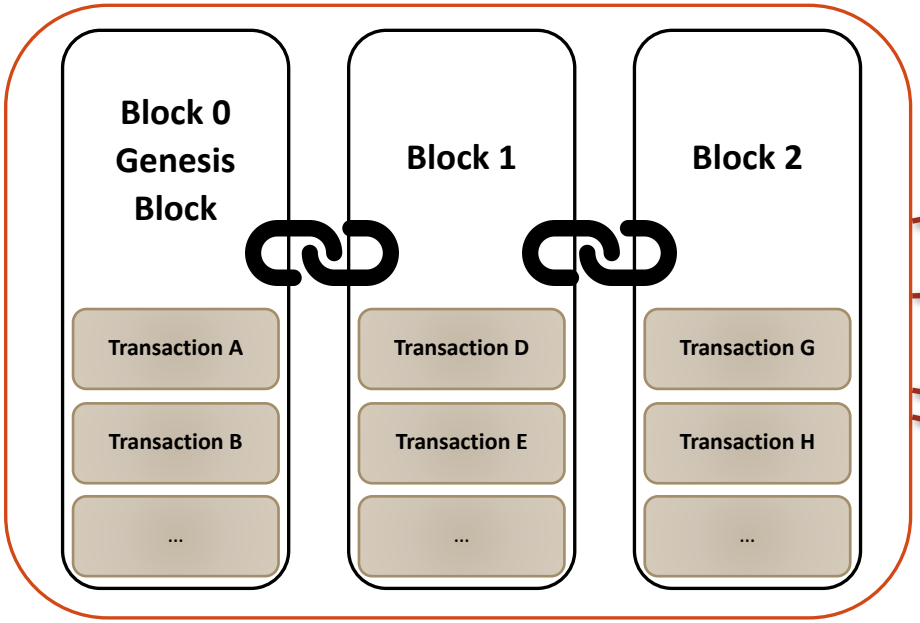
**Blockchain Data Structure**

**Peer-to-Peer Network**

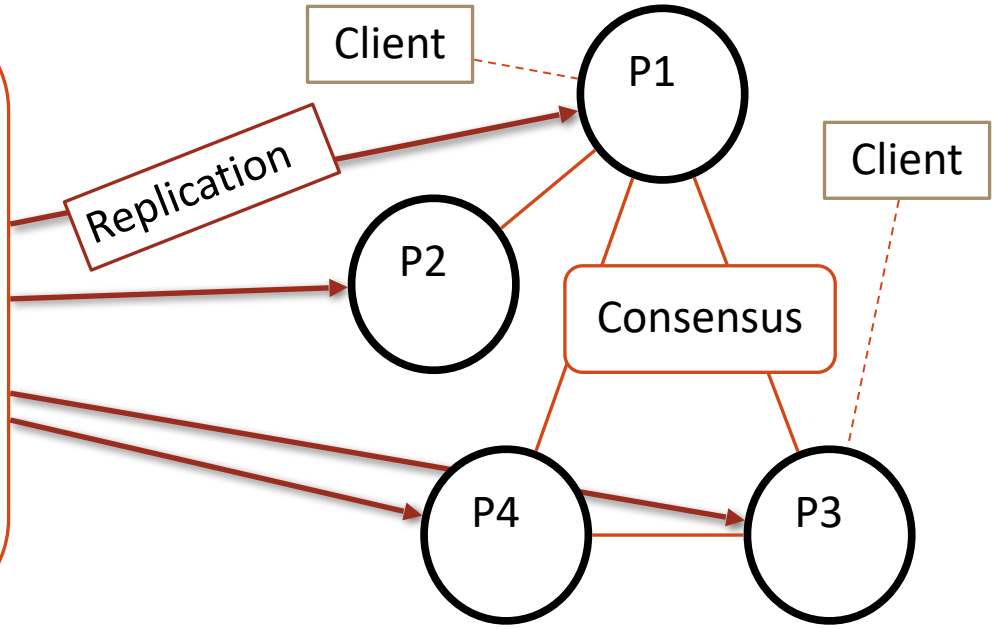


# Blockchain 101: Distributed Ledger Technology (DLT)

**Blockchain Data Structure**




**Peer-to-Peer Network**




*Cryptography is used to…*

*…**encrypt data, prevent modification, insert new blocks, execute transactions, and query…***

*the distributed ledger*

# Comparison with Databases

| | Single Machine DBMSs | Distributed Databases | | |
|---|---|---|---|---|
| | | OLTP | OLAP | |
| Logically centralized (Single entity) | MySQL, Oracle, DB2, … | NewSQL: Spanner, VoltDB, … | Distributed SQL data warehouses | Relational |
| | | NoSQL: Hbase, Cassandra, … | Hadoop, MapReduce | Non-relational |
| Decentralized (Public/Private) | | Distributed Ledgers (DLT) | | Blockchain |

# Comparison with Databases

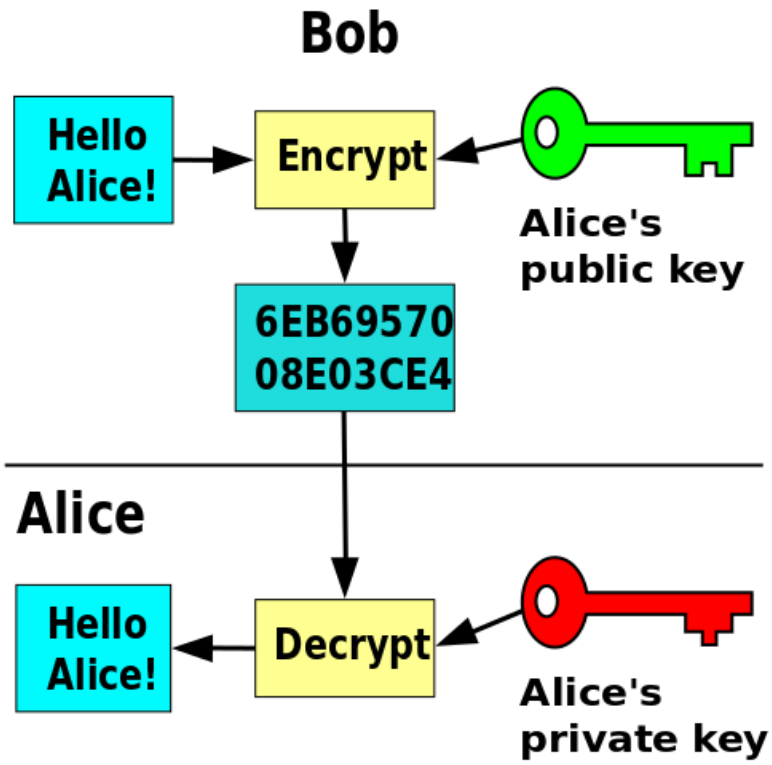
| | Single Machine DBMSs | Distributed Databases | | |
|---|---|---|---|---|
| | | OLTP | OLAP | |
| Logically centralized (Single entity) | MySQL, Oracle, [illegible] | [illegible] | [illegible]ed SQL [illegible]ses | Relational |
| | | | [illegible]uce | Non-relational |
| Decentralized (Public/Private) | | Distributed Ledgers (DLT) | | Blockchain |

The key distinction is the use of *cryptography* to enable operation in a decentralized trustless environment.

# Public Key Cryptography

## (Asymmetrical Cryptography)

- Recipient's **public key** is used to **encrypt** the plaintext to ciphertext
- Recipient's **private key** to **decrypt** the ciphertext to original plaintext
- **No one can use the public key to decrypt the ciphertext to plaintext**



# Outline

What?

- Concepts: Mining, proof-of-work, smart contracts
- Case studies: Bitcoin, Ethereum, Hyperledger

Why?

- Blockchain applications
- Why study blockchains?

How?

- The six layers of blockchain systems
- Research directions

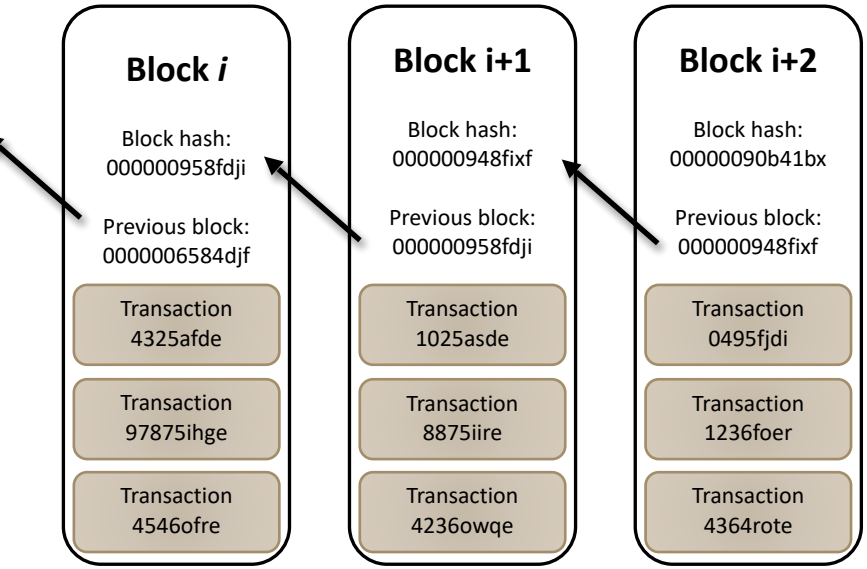
# What is a Blockchain?

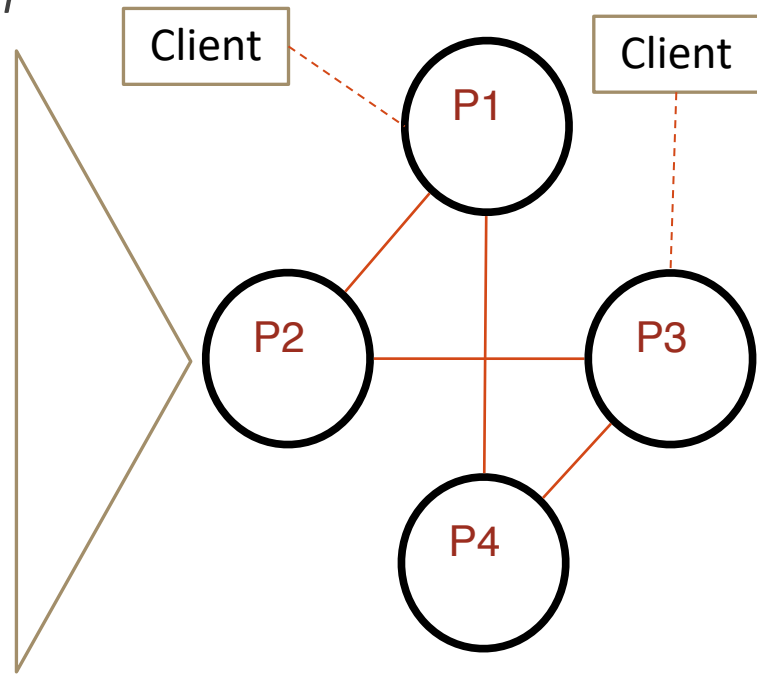
*A blockchain-based **distributed ledger** is:*

- ✓ *An **append-only** log* storing transactions
- ✓ *Fully **replicated*** across a large number of peers (called miners)
- ✓ Comprised of ***immutable*** blocks of data
- ✓ ***Deterministically verifiable*** (using the *blockchain* data structure)
- ✓ Able to execute transactions ***programmatically*** (e.g., Bitcoin transactions and smart contracts)
- ✓ Fully ***decentralized***, does not rely on a third party for trust

# Immutability using Hashing

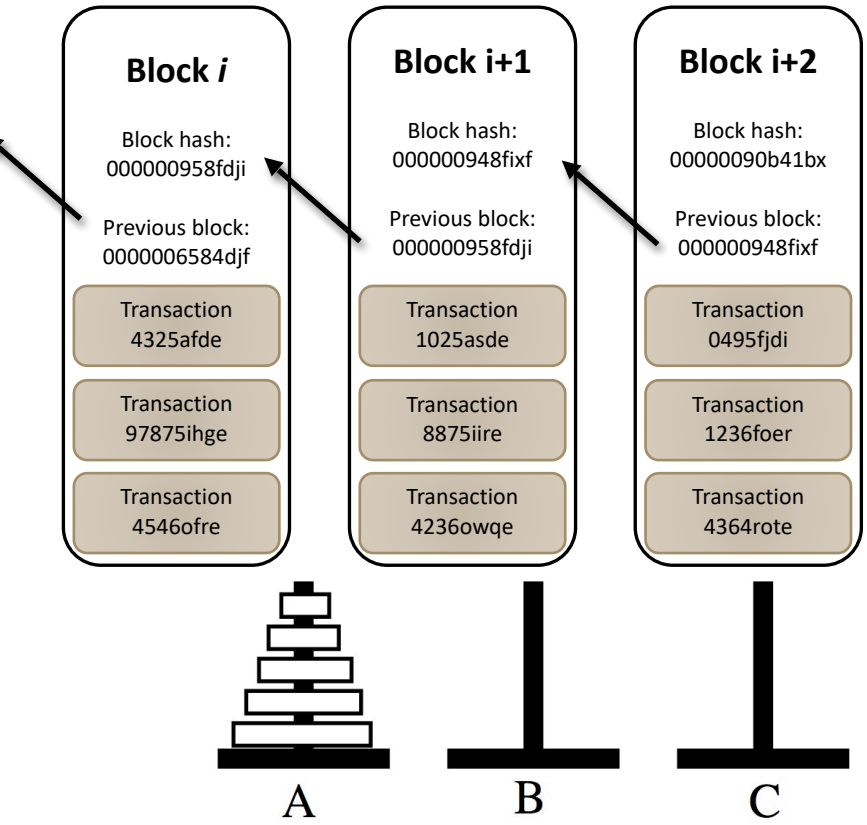
*Blockchain data structure maintained at every peer*

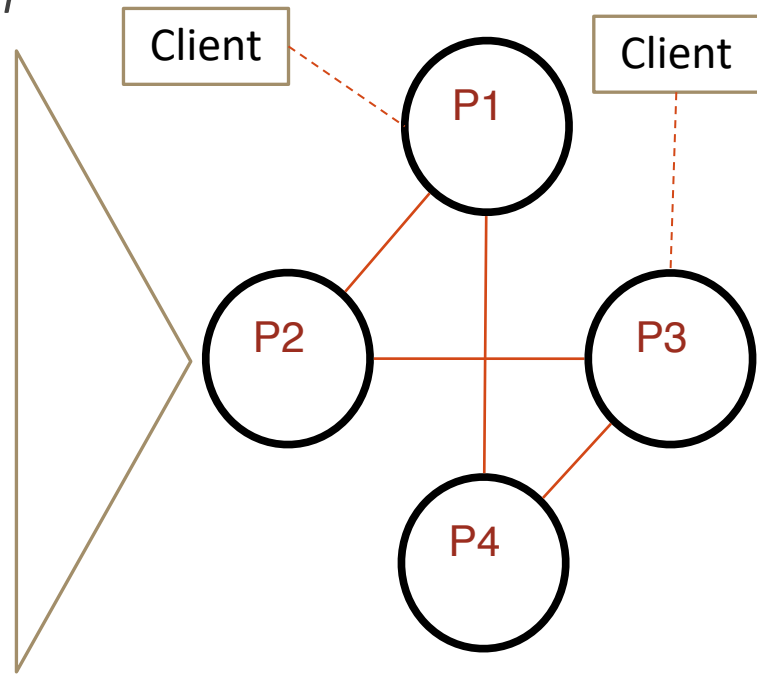


# Immutability using Hashing

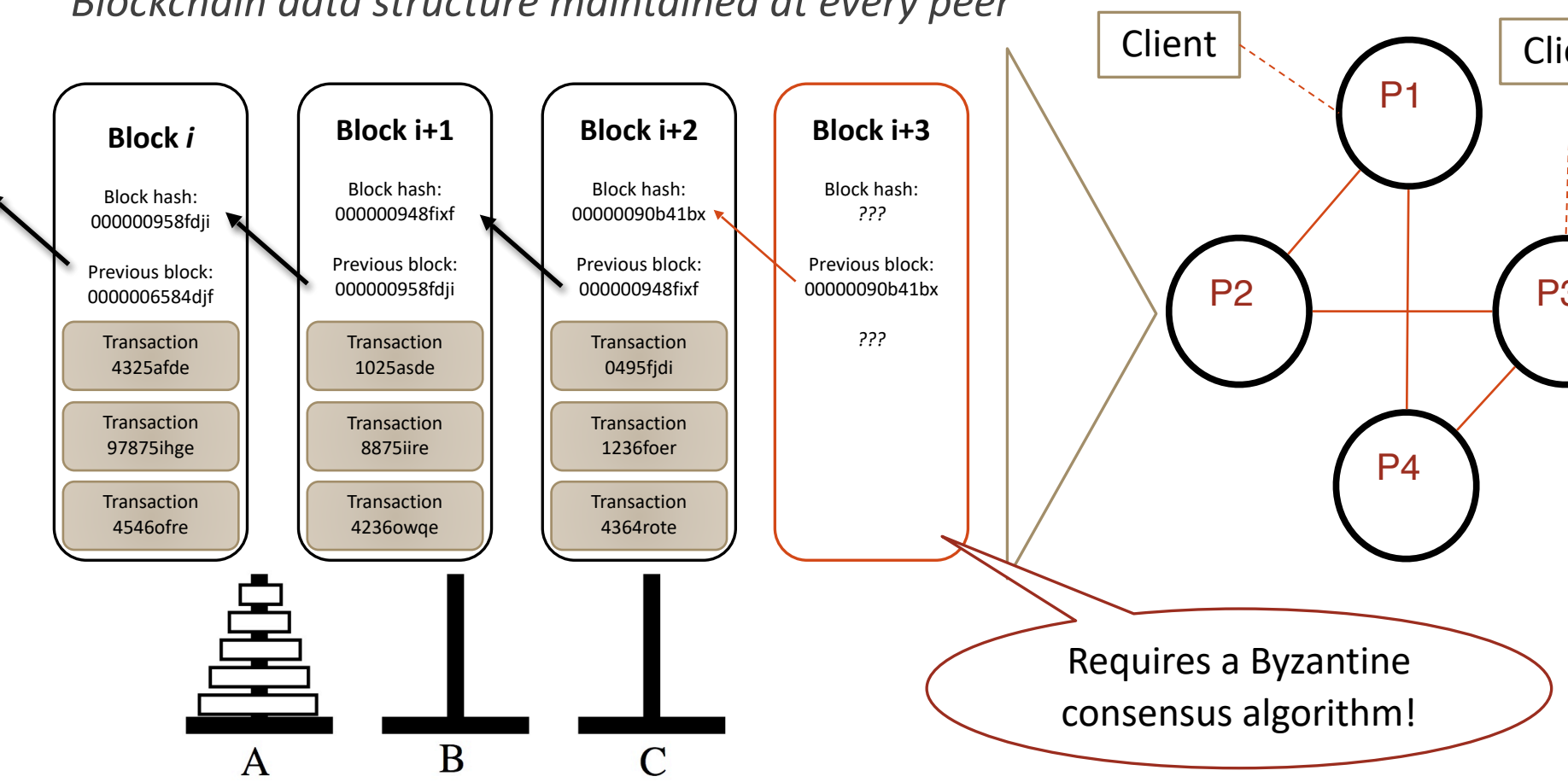
*Blockchain data structure maintained at every peer*



# Immutability using Hashing

*Blockchain data structure maintained at every peer*



MIDDLEWARE SYSTEMS
RESEARCH GROUP
MSRG.ORG

# Deconstructing Blockchains: Concepts, Systems and Applications

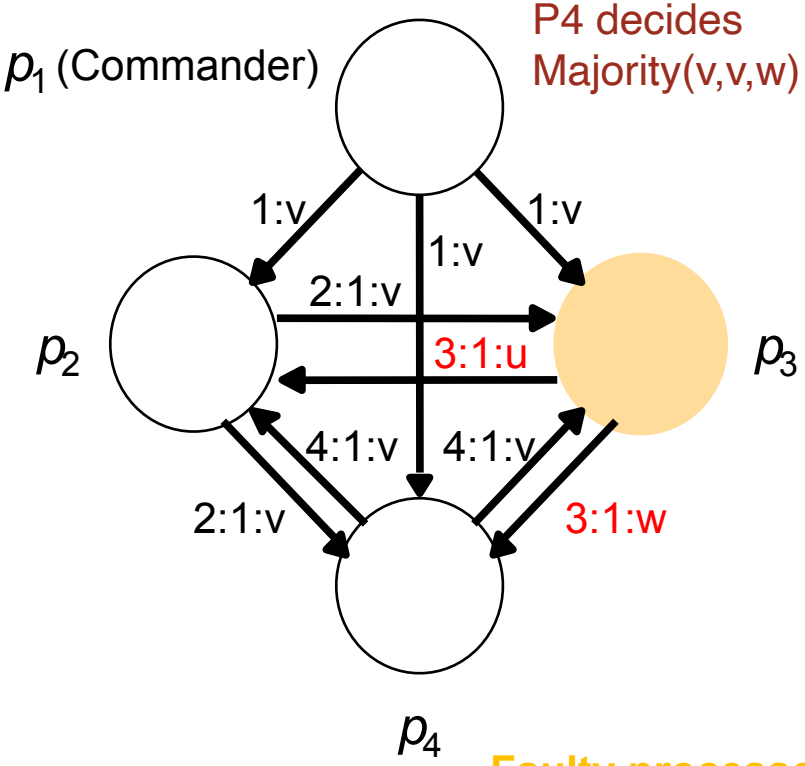
## BACKGROUND: BYZANTINE GENERALS PROBLEM

# Origin: Byzantine Generals

- Devised by Lamport, 1982; a model and thought experiment
- A *distinguished process* (the ***commander***) proposes initial value (e.g., “attack”, “retreat”)
- Other processes, the ***lieutenants***, *communicate the commander’s value*
- *Malicious processes* can lie about the value (i.e., *are faulty*)
- *Correct processes* report the truth (i.e., are *correct*)
- Commander or lieutenants may be faulty
- **Consensus** means
  - If the commander is correct, then correct processes should agree on commander’s proposed value
  - If the commander is faulty, then all correct processes agree on a value *(any value, could be the faulty commander’s value!)*

# 3f+1 Condition (1 failure, 4 nodes)

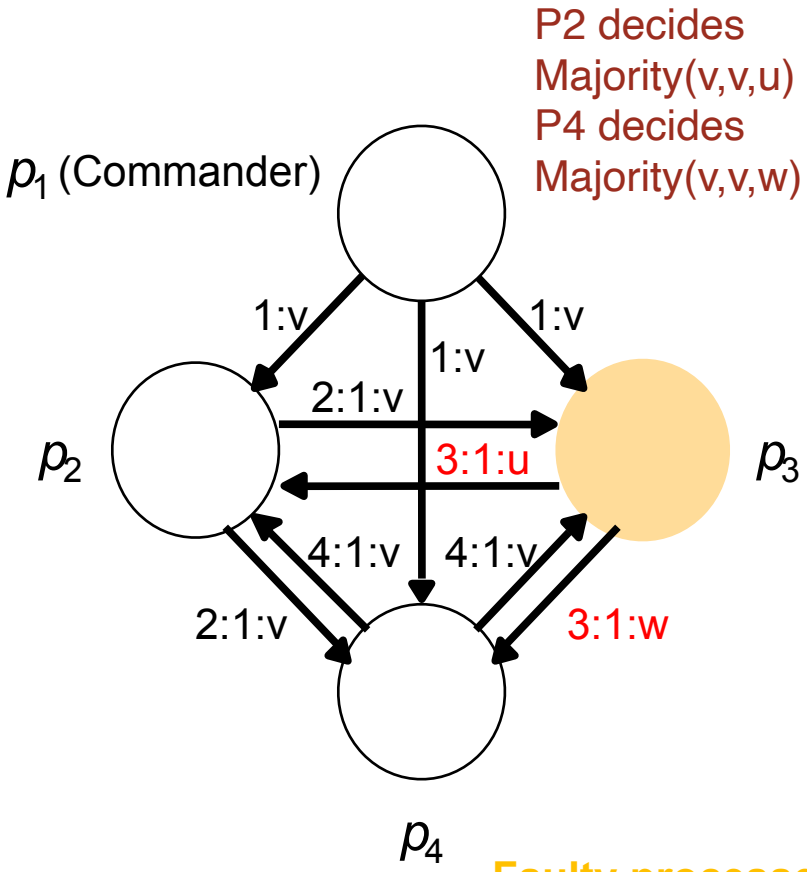
P2 decides
Majority(v,v,u) = v
P4 decides
Majority(v,v,w) = v

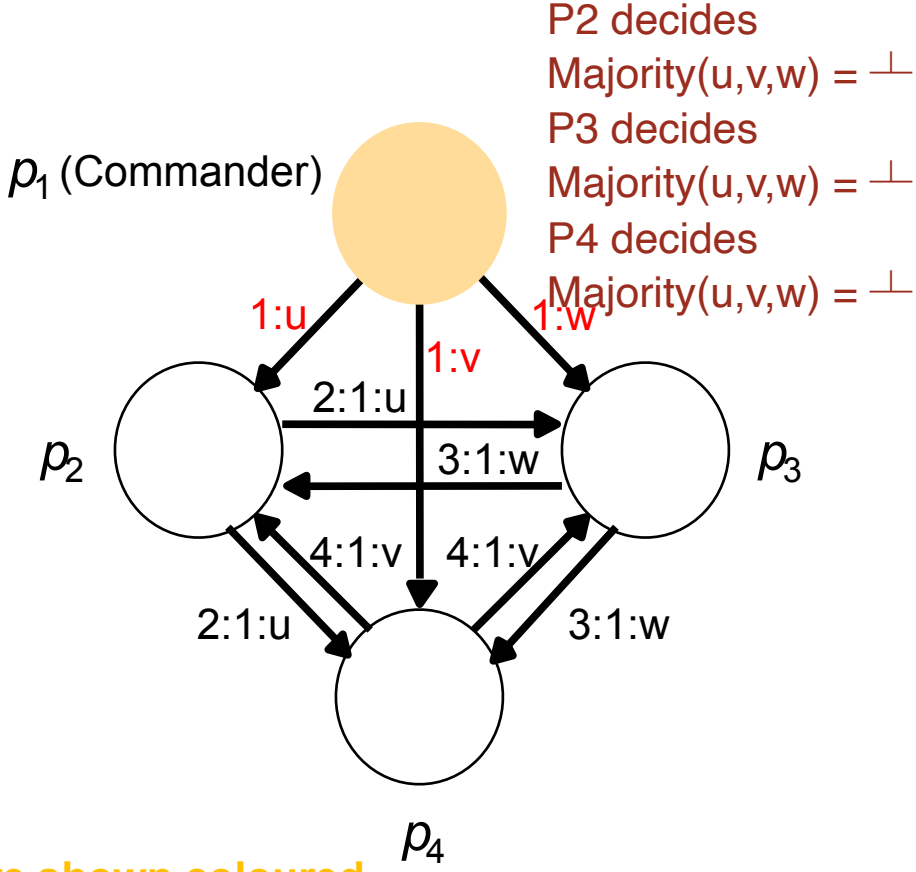



Faulty processes are shown coloured

*Source: Tanenbaum, Steen.*

# 3f+1 Condition (1 failure, 4 nodes)

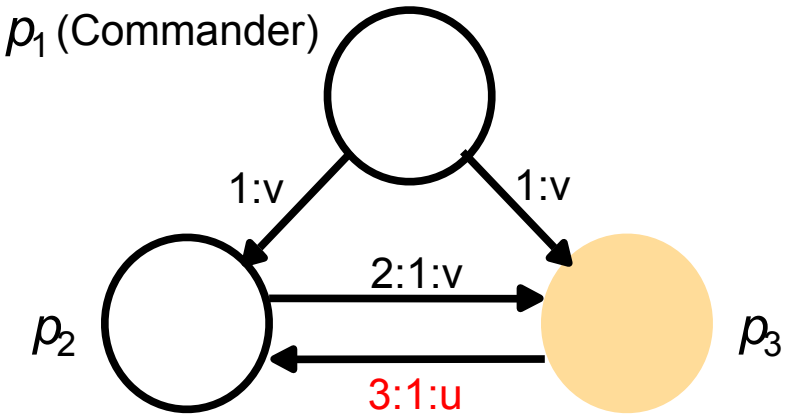


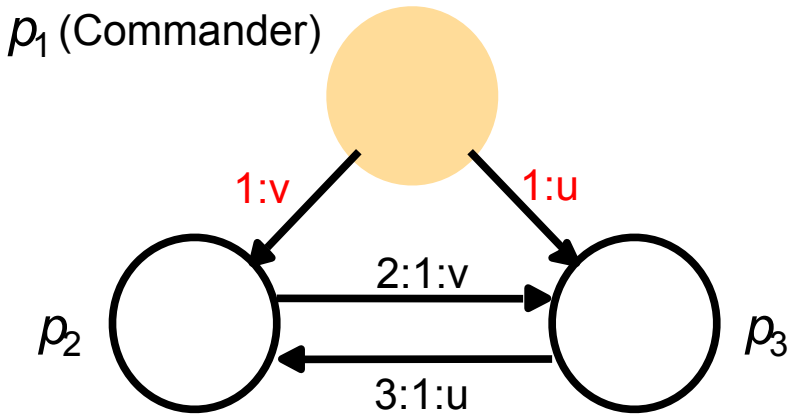




**Faulty processes are shown coloured**

*Source: Tanenbaum, Steen.*

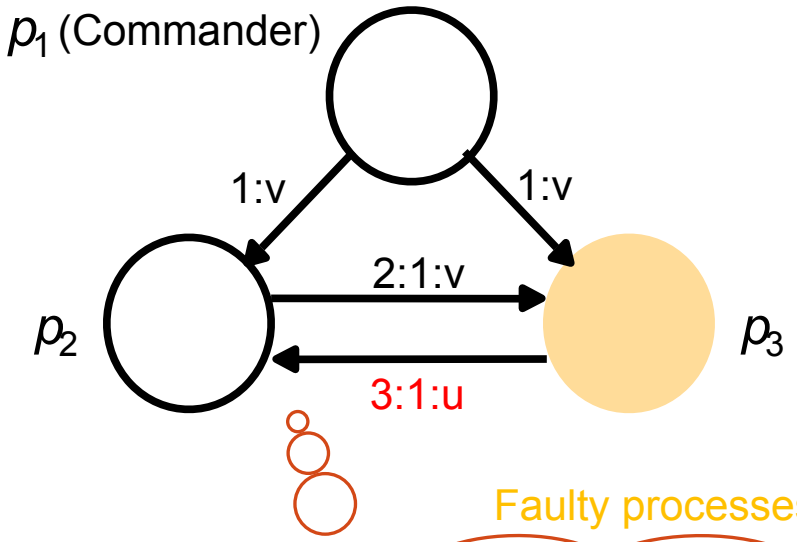
# Counter-Example
## (1 failure, 3 nodes)

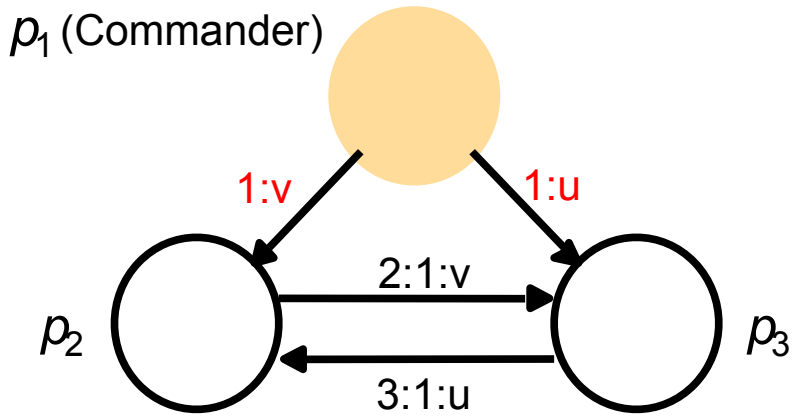



Faulty processes are shown coloured

# Counter-Example
## (1 failure, 3 nodes)

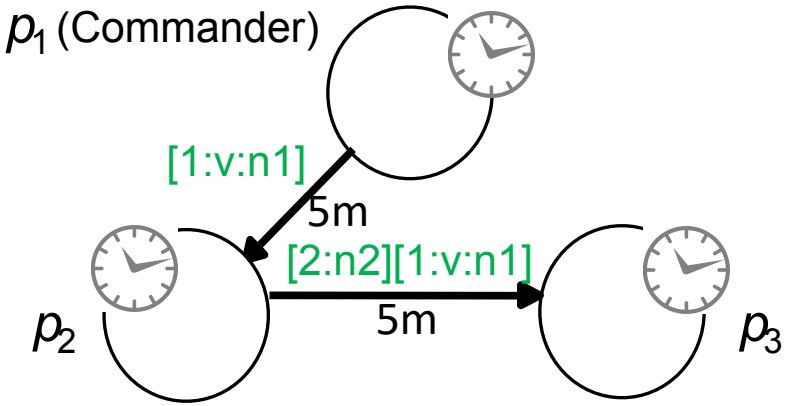



Faulty processes are shown coloured

Trust commander or p3?
Can't tell the difference
between both situations!

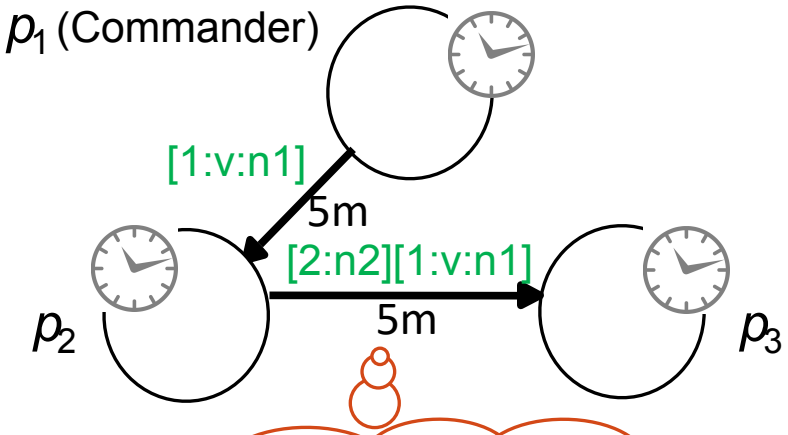
# With Blockchains
## (Proof-of-Work – Thought Experiment)




Idea #1:
Each message takes exactly 5 minutes to create by any process. (“*Magic Block*”)

Idea #2:
Each process can accurately measure the amount of time taken by a process to create a message. (“*Magic Watch*”)

# With Blockchains
## (Proof-of-Work – Thought Experiment)



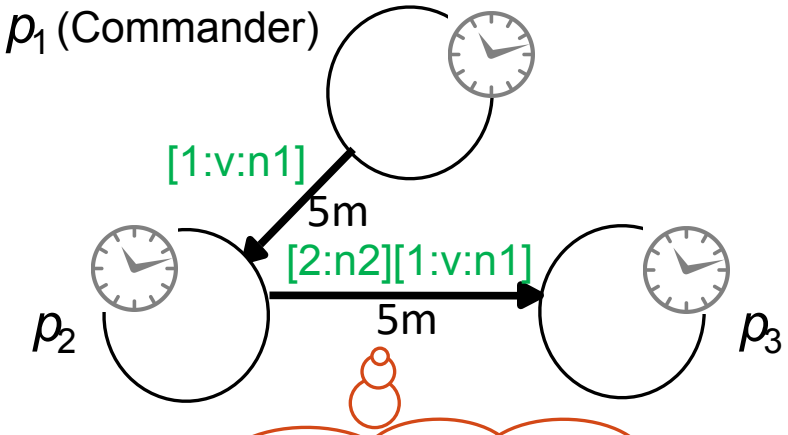

P3 verifies each block and receives them in time.

Each message takes exactly 5 minutes to create by any process. (“*Magic Block*”)

Idea #2:

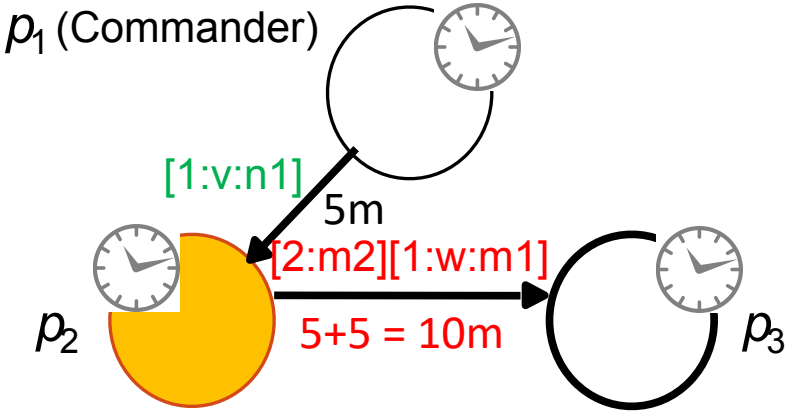
Each process can accurately measure the amount of time taken by a process to create a message. (“*Magic Watch*”)

# With Blockchains
## (Proof-of-Work – Thought Experiment)




P3 verifies each block and receives them in time.

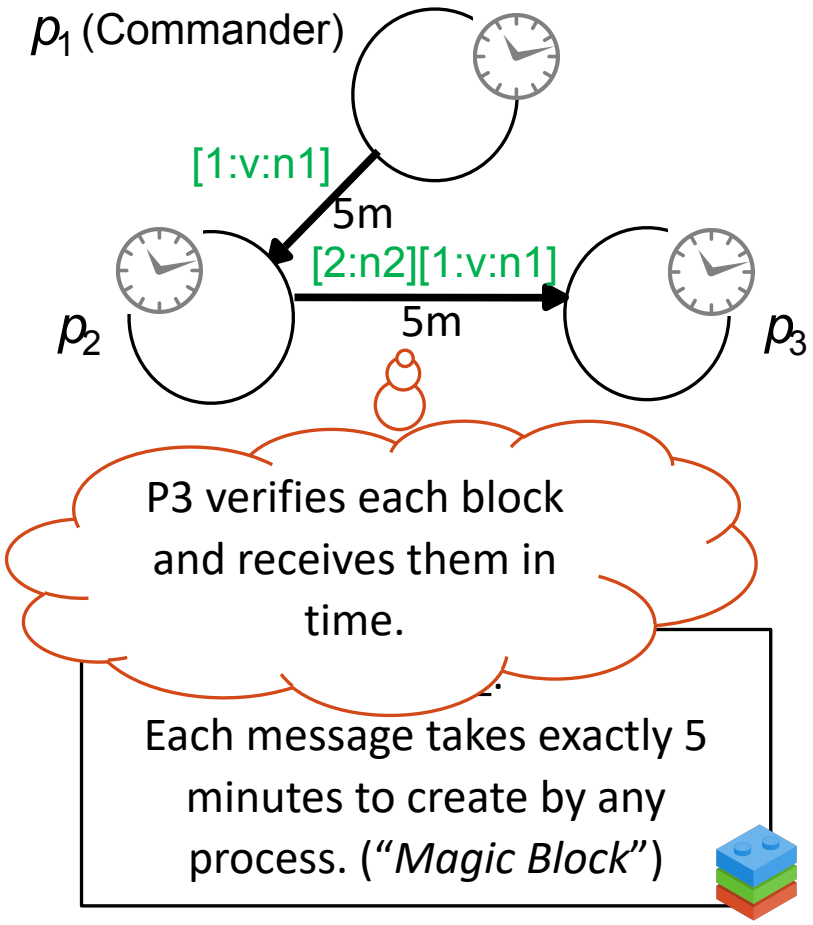
Each message takes exactly 5 minutes to create by any process. ("*Magic Block*")

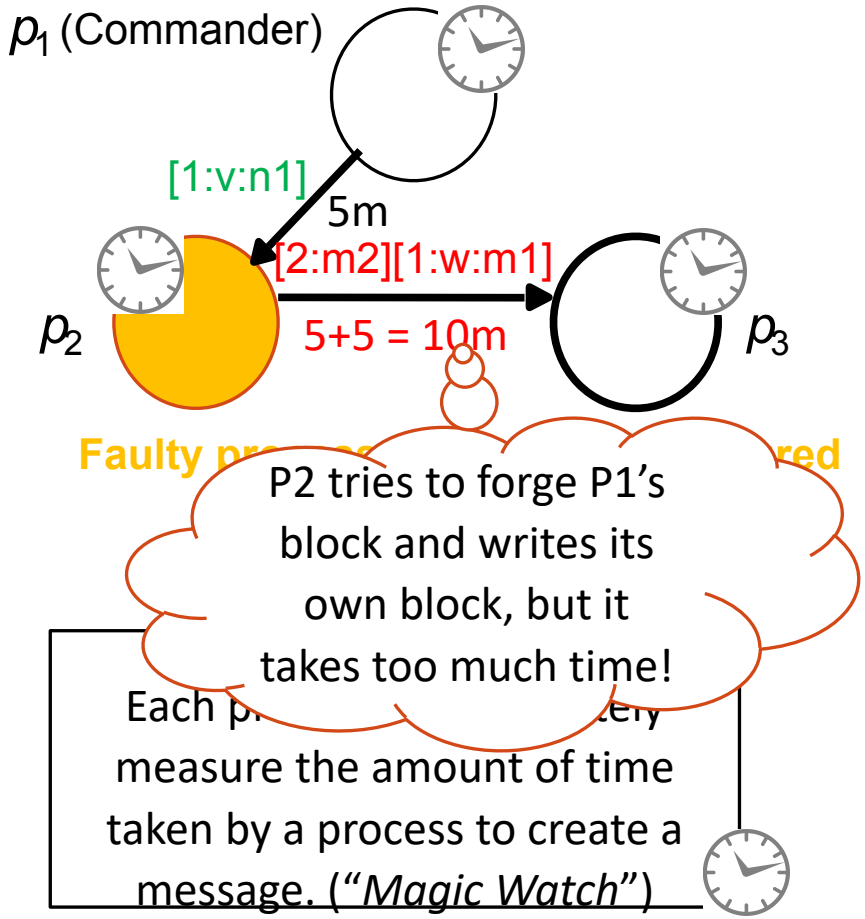



**Faulty processes are shown coloured**

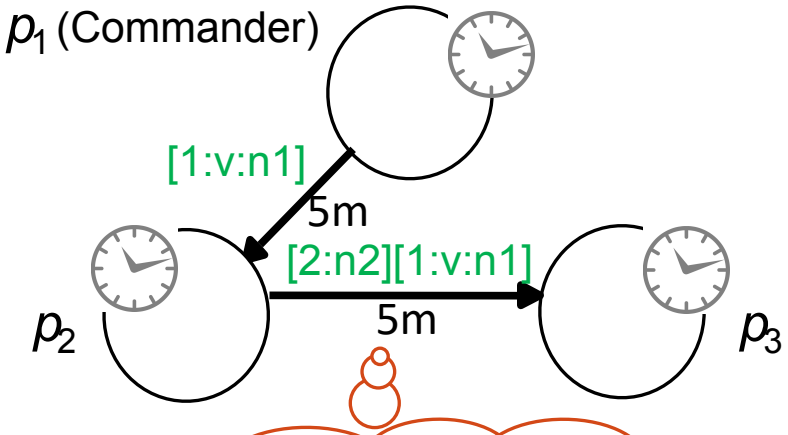
Idea #2:
Each process can accurately measure the amount of time taken by a process to create a message. ("*Magic Watch*")

# With Blockchains
## (Proof-of-Work – Thought Experiment)



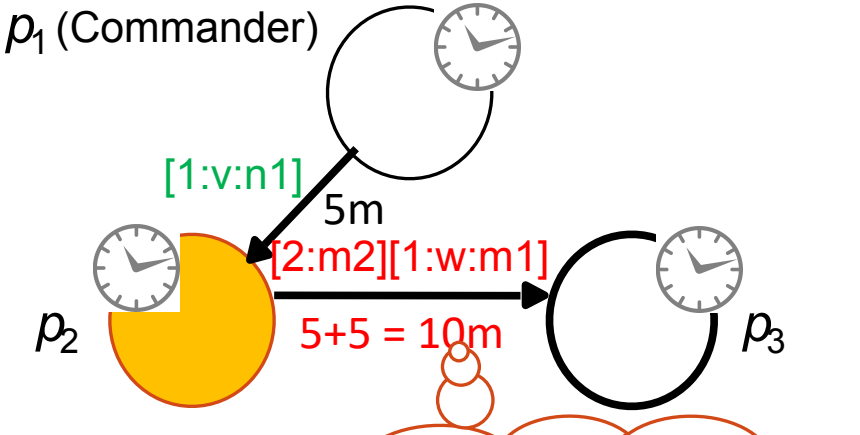



# With Blockchains
## (Proof-of-Work – Thought Experiment)




P3 verifies each block and receives them in time

Idea #1:
Each message takes exactly 5 minutes to create by any process. (“*Magic Block*”)




P2 tries to forge P1’s block and writes its own block, but it

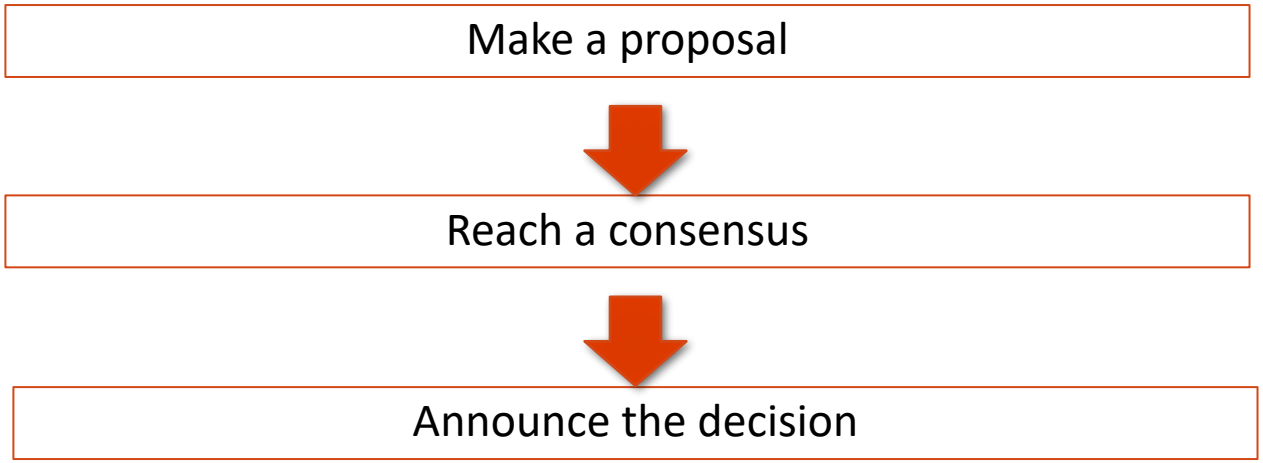
Idea #2:
Each process can accurately measure the amount of time taken by a process to create a message. (“*Magic Watch*”)

# Consensus in the Bitcoin Blockchain

The peers need to agree on

- Which recently broadcast transactions go into the blockchain
- In what order they go into a block

The general anatomy of consensus:



# Consensus in the Bitcoin Blockchain

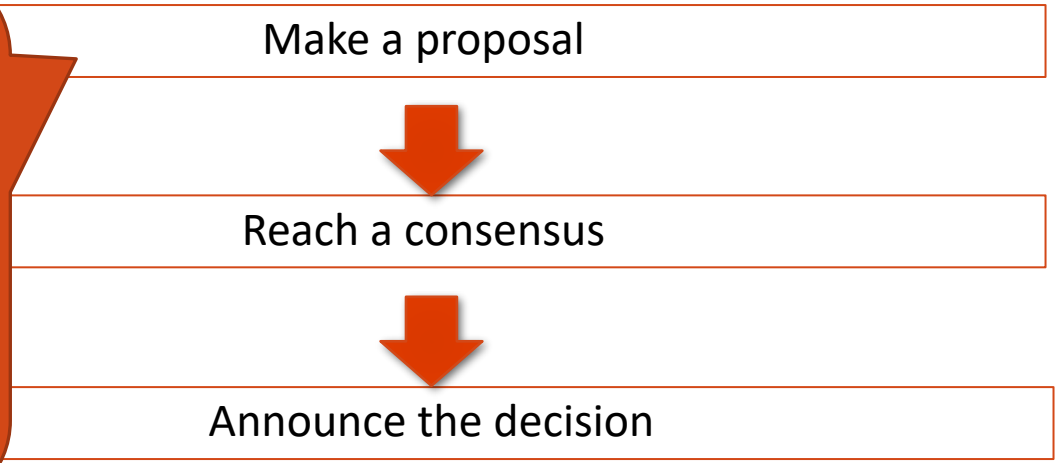
The peers need to agree on

- Which recently broadcast transactions go into the blockchain
- In what order they go into a block

The general anatomy of consensus:

Tough problem

- Dozens of **impossibility results** since 1983
- **Does not scale** beyond ~30-100 participants
- Takes long time to converge



MIDDLEWARE SYSTEMS
RESEARCH GROUP
MSRG.ORG

# Deconstructing Blockchains: Concepts, Systems and Applications

## PROOF OF WORK AND MINING

# Blockchain "Puzzles"

verify(***nonce***, data) meets some "requirements"

Use of "trapdoor functions" (hash functions)

- Cannot reverse the function to find the input
- Therefore, keep trying random values (called nonce) until you find a solution
- Like trying random combinations to a lock…
- *The more computing power* you have, *faster* you can solve the puzzle.
- "*Magic blocks*" are blocks with puzzles, where everyone has the same power.

# Proof-of-Work Example

E.g., the challenge is:

- sha256sum("data:**nonce**") starts with a "0"
- Normally more complicated than that! (e.g., 18 zeroes)

➢P1 wants to send "1:v" to P2

```
arno@grey:~$ echo "1:v:118" | sha256sum
9479038ca7543ece09f48e8c77fcea147d7561cac14058199afea18c2f323b8b
arno@grey:~$ echo "1:v:119" | sha256sum
79ae2bbac929112a349c2fe7f50210355f4a24683b2dd1ea8f059c9beeed7fd6
arno@grey:~$ echo "1:v:120" | sha256sum
002ce3a3b7092d960abf1795a89f70eb0f9ef960036e7d4620cbd3d26d34ffc8
```

➢Send "1:v:120" to P2

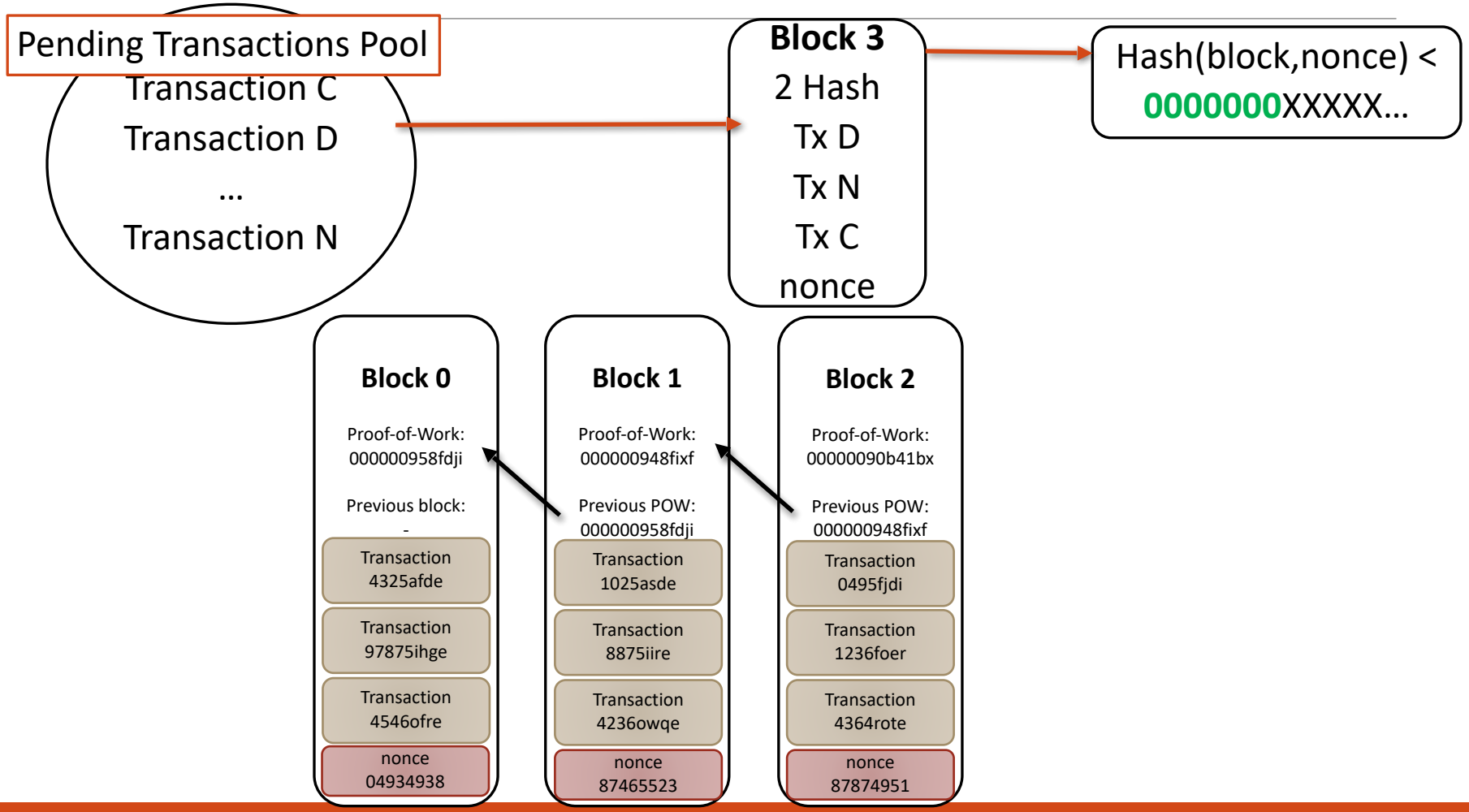
# Proof-of-Work Example

- P2 verifies "1:v:120" is correct (very quick!) (sha256sum("1:v:120") starts with a "0")
- P2 wants to send "2:1:v:120" to P3

```
arno@grey:~$  echo "2:1:v:120:119" | sha256sum
911ab1edf1f331ff423a45fe4c382db30a3f1cf802bb2211df53c80d5798c7ba
aarno@grey:~$  echo "2:1:v:120:120" | sha256sum
5344a3561673b1481b9cf69493368ca408b1edef67e3f96819c5d1b36cea53ce
arno@grey:~$  echo "2:1:v:120:121" | sha256sum
0a908c651e9ec5374976dc8f49a3342a4a789660011551da8871a6cc123c5b57
```

- P2 sends "2:1:v:120:121"
- P3 verifies "1:v:120" *AND* "2:1:v:120:121" are correct
- If P2 wants to send "2:1:w" and fool P3, it needs to find $n_1$ for "1:w: $n_1$" and $n_2$ for "2:1:w: $n_1$ : $n_2$"
- If P3 has a way to ***detect*** that P2 is ***doing too much work***, it can detect fraud.

# Proof-of-Work in Bitcoin



# f-of-Work in Bitcoin

Pending transactions are propagated to the peers (miners)

Pending Transactions Pool

Transaction C
Transaction D
…
Transaction N

**Block 3**
2 Hash
Tx D
Tx N
Tx C
nonce

Hash(block,nonce) < 0000000XXXXX…

**Block 0**
Proof-of-Work: 000000958fdji
Previous block: -
Transaction 4325afde
Transaction 97875ihge
Transaction 4546ofre
nonce 04934938

**Block 1**
Proof-of-Work: 000000948fixf
Previous POW: 000000958fdji
Transaction 1025asde
Transaction 8875iire
Transaction 4236owqe
nonce 87465523

**Block 2**
Proof-of-Work: 00000090b41bx
Previous POW: 000000948fixf
Transaction 0495fjdi
Transaction 1236foer
Transaction 4364rote
nonce 87874951

# f-of-Work in Bitcoin

Pending transactions are propagated to the peers (miners)

Pending Transactions Pool

Transaction C
Transaction D
…
Transaction N

Miners verify and put transactions in a block, seek nonce

**Block 3**
2 Hash
Tx D
Tx N
Tx C
nonce

Hash(block,nonce) < 0000000XXXXX…

| Block 0 | Block 1 | Block 2 |
|---|---|---|
| Proof-of-Work: 000000958fdji | Proof-of-Work: 000000948fixf | Proof-of-Work: 00000090b41bx |
| Previous block: - | Previous POW: 000000958fdji | Previous POW: 000000948fixf |
| Transaction 4325afde | Transaction 1025asde | Transaction 0495fjdi |
| Transaction 97875ihge | Transaction 8875iire | Transaction 1236foer |
| Transaction 4546ofre | Transaction 4236owqe | Transaction 4364rote |
| nonce 04934938 | nonce 87465523 | nonce 87874951 |

# f-of-Work in Bitcoin

Pending transactions are propagated to the peers (miners)

Pending Transactions Pool

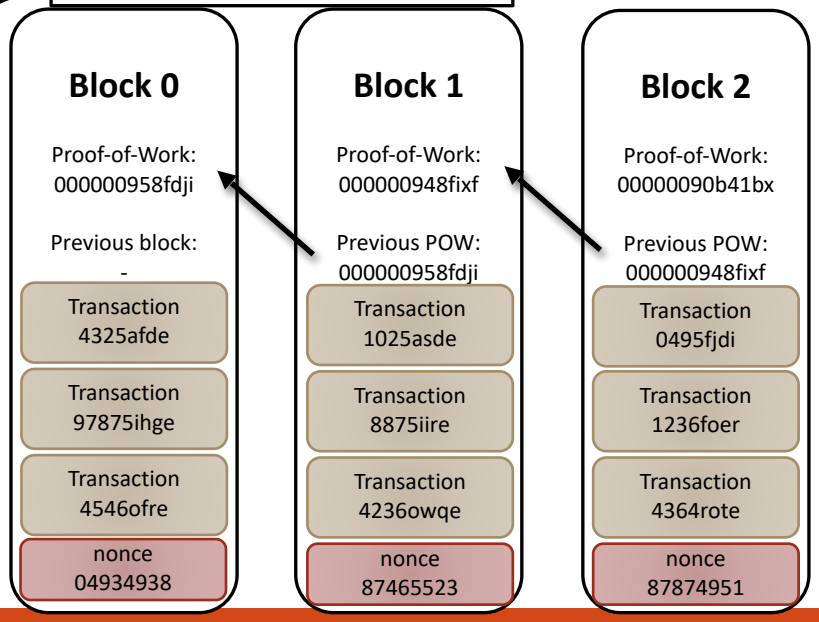
Transaction C
Transaction D
…
Transaction N

Miners verify and put transactions in a block, seek nonce

**Block 3**
2 Hash
Tx D
Tx N
Tx C
nonce

Hash(block,nonce) < 0000000XXXXX…

Number of leading zeroes (difficulty) depend on the global *hash-rate*, s.t. one block is solved *per 10 minutes*



# f-of-Work in Bitcoin

Pending transactions are propagated to the peers (miners)

Pending Transactions Pool

Transaction C
Transaction D
…
Transaction N

Miners verify and put transactions in a block, seek nonce

**Block 3**
2 Hash
Tx D
Tx N
Tx C
nonce

Hash(block,nonce) < 0000000XXXXX…

Number of leading zeroes (difficulty) depend on the global *hash-rate*, s.t. one block is solved *per 10 minutes*

| **Block 0** | **Block 1** | **Block 2** | **Block 3** |
|---|---|---|---|
| Proof-of-Work: 000000958fdji | Proof-of-Work: 000000948fixf | Proof-of-Work: 00000090b41bx | Proof-of-Work: *000000r9d8fjj* |
| Previous block: - | Previous POW: 000000958fdji | Previous POW: 000000948fixf | Previous block: 00000090b41bx |
| Transaction 4325afde | Transaction 1025asde | Transaction 0495fjdi | Transaction D |
| Transaction 97875ihge | Transaction 8875iire | Transaction 1236foer | Transaction N |
| Transaction 4546ofre | Transaction 4236owqe | Transaction 4364rote | Transaction C |
| nonce 04934938 | nonce 87465523 | nonce 87874951 | nonce 79146512 |

The “lucky” miner attaches the solved block to the chain, or stops solving if someone else finds a valid block.

# f-of-Work in Bitcoin

Pending transactions are propagated to the peers (miners)

Pending Transactions Pool

Transaction C
Transaction D
…
Transaction N

Miners verify and put transactions in a block, seek nonce

**Block 3**
2 Hash
Tx D
Tx N
Tx C
nonce

Hash(block,nonce) < 0000000XXXXX…

Number of leading zeroes (difficulty) depend on the global *hash-rate*, s.t. one block is solved *per 10 minutes*

The more *confirmations* a transaction receives, the less likely it is to disappear.

**Block 0**
Proof-of-Work: 000000958fdji
Previous block: -
Transaction 4325afde
Transaction 97875ihge
Transaction 4546ofre
nonce 04934938

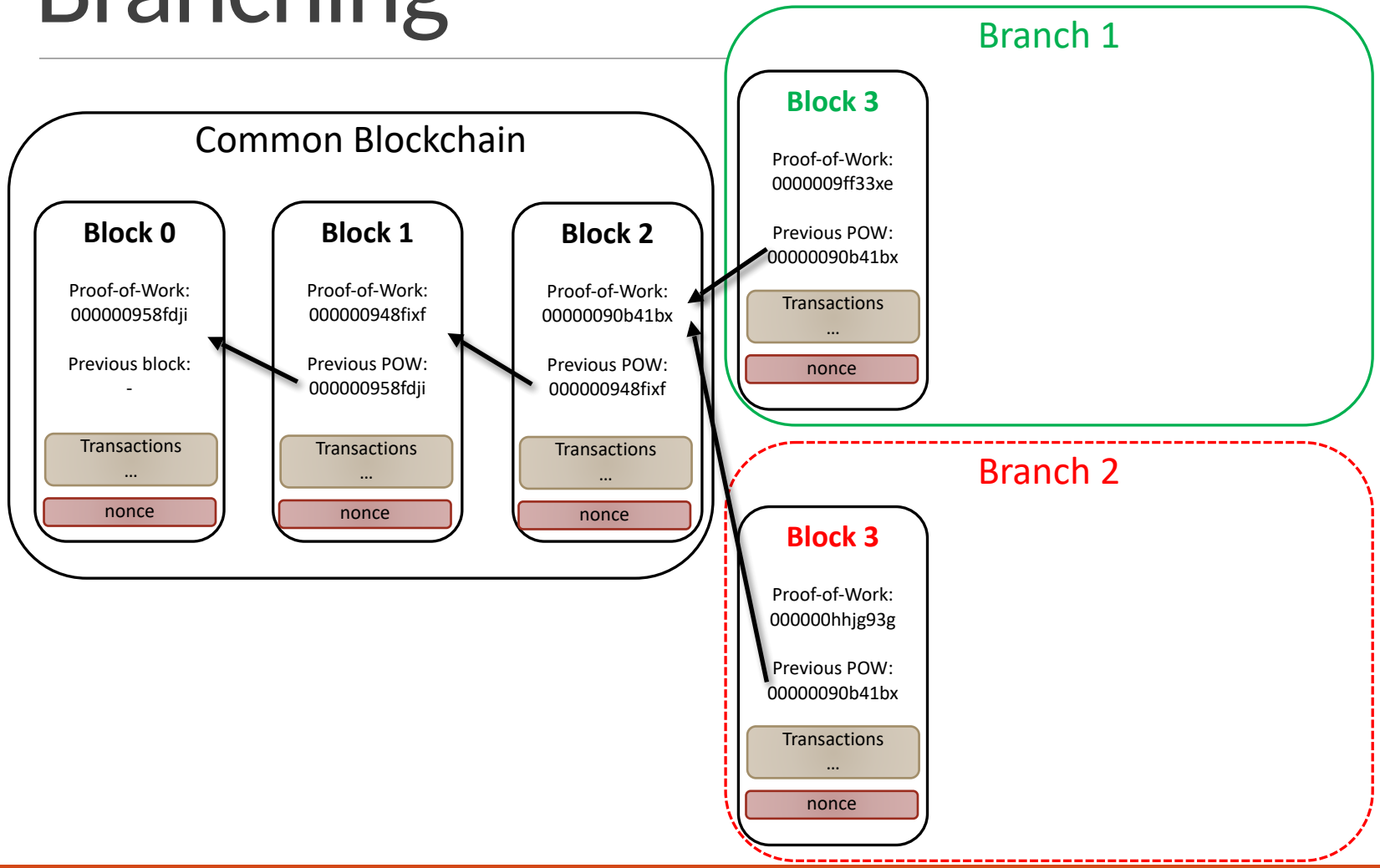
**Block 1**
Proof-of-Work: 000000948fixf
Previous POW: 000000958fdji
Transaction 1025asde
Transaction 8875iire
Transaction 4236owqe
nonce 87465523

**Block 2**
Proof-of-Work: 00000090b41bx
Previous POW: 000000948fixf
Transaction 0495fjdi
Transaction 1236foer
Transaction 4364rote
nonce 87874951

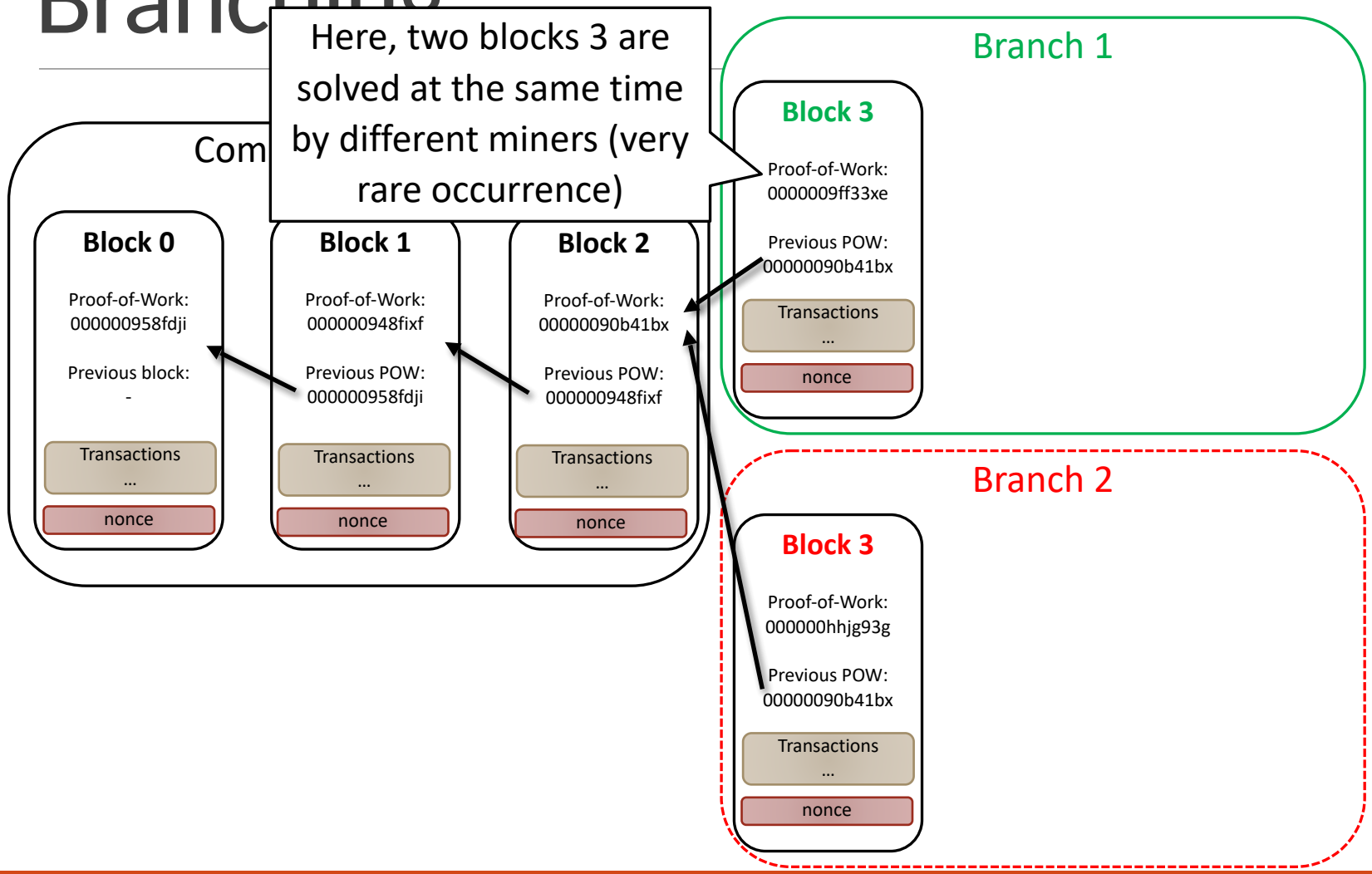
**Block 3**
Proof-of-Work: *000000r9d8fjj*
Previous block: 00000090b41bx
Transaction D
Transaction N
Transaction C
nonce 79146512

The "lucky" miner attaches the solved block to the chain, or stops solving if someone else finds a valid block.

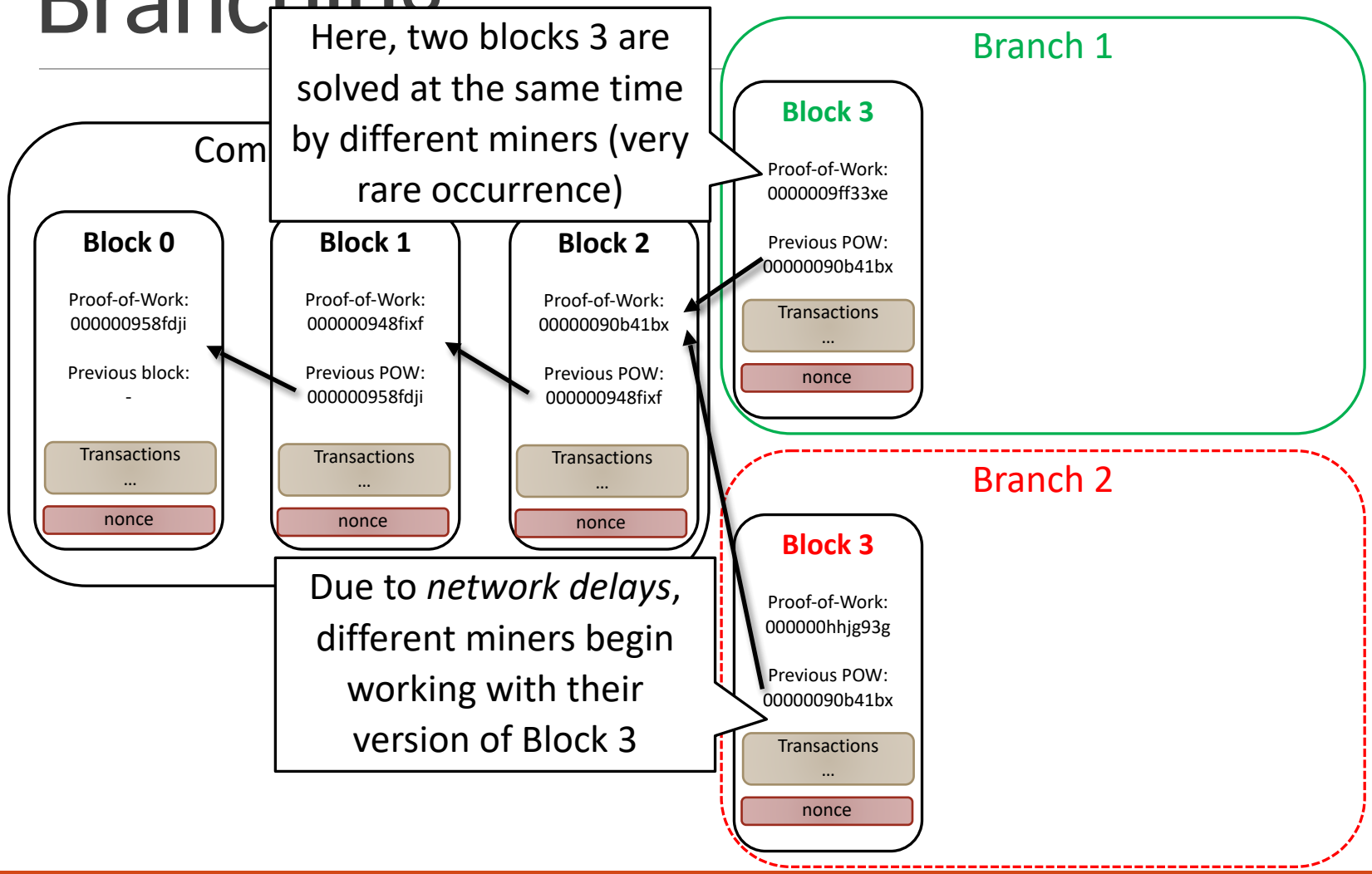
# Branching


Common Blockchain
Block 0
Proof-of-Work:
000000958fdji
Previous block:
-
Transactions
...
nonce
Block 1
Proof-of-Work:
000000948fixf
Previous POW:
000000958fdji
Transactions
...
nonce
Block 2
Proof-of-Work:
00000090b41bx
Previous POW:
000000948fixf
Transactions
...
nonce
Branch 1
Block 3
Proof-of-Work:
0000009ff33xe
Previous POW:
00000090b41bx
Transactions
...
nonce
Branch 2
Block 3
Proof-of-Work:
000000hhjg93g
Previous POW:
00000090b41bx
Transactions
...
nonce

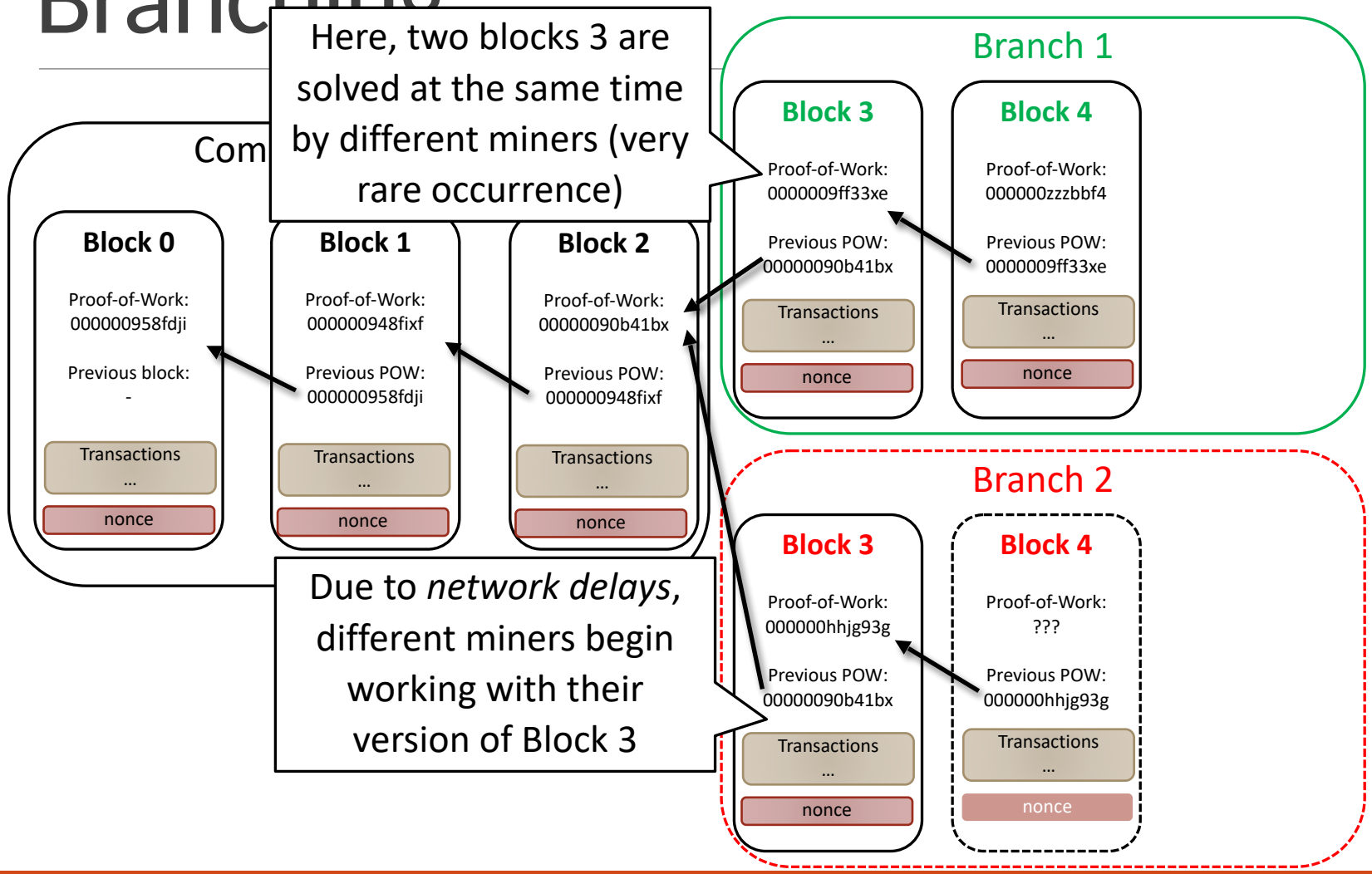
# Branching


Here, two blocks 3 are solved at the same time by different miners (very rare occurrence)
Com
Block 0
Proof-of-Work:
000000958fdji
Previous block:
-
Transactions
...
nonce
Block 1
Proof-of-Work:
000000948fixf
Previous POW:
000000958fdji
Transactions
...
nonce
Block 2
Proof-of-Work:
00000090b41bx
Previous POW:
000000948fixf
Transactions
...
nonce
Branch 1
Block 3
Proof-of-Work:
0000009ff33xe
Previous POW:
00000090b41bx
Transactions
...
nonce
Branch 2
Block 3
Proof-of-Work:
000000hhjg93g
Previous POW:
00000090b41bx
Transactions
...
nonce

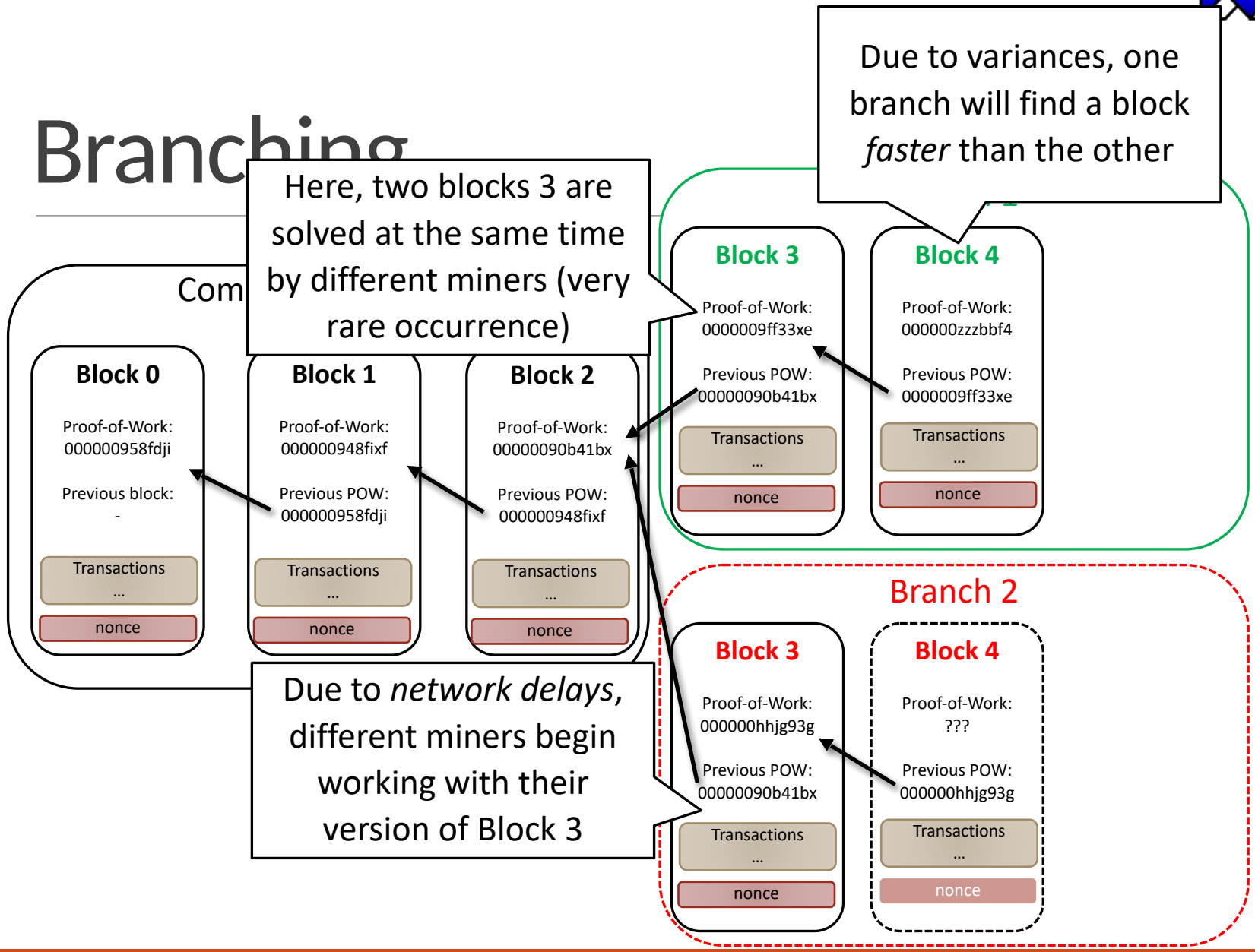
# Branching


Here, two blocks 3 are solved at the same time by different miners (very rare occurrence)
Com
Block 0
Proof-of-Work:
000000958fdji
Previous block:
-
Transactions
...
nonce
Block 1
Proof-of-Work:
000000948fixf
Previous POW:
000000958fdji
Transactions
...
nonce
Block 2
Proof-of-Work:
00000090b41bx
Previous POW:
000000948fixf
Transactions
...
nonce
Branch 1
Block 3
Proof-of-Work:
0000009ff33xe
Previous POW:
00000090b41bx
Transactions
...
nonce
Branch 2
Block 3
Proof-of-Work:
000000hhjg93g
Previous POW:
00000090b41bx
Transactions
...
nonce
Due to *network delays*, different miners begin working with their version of Block 3

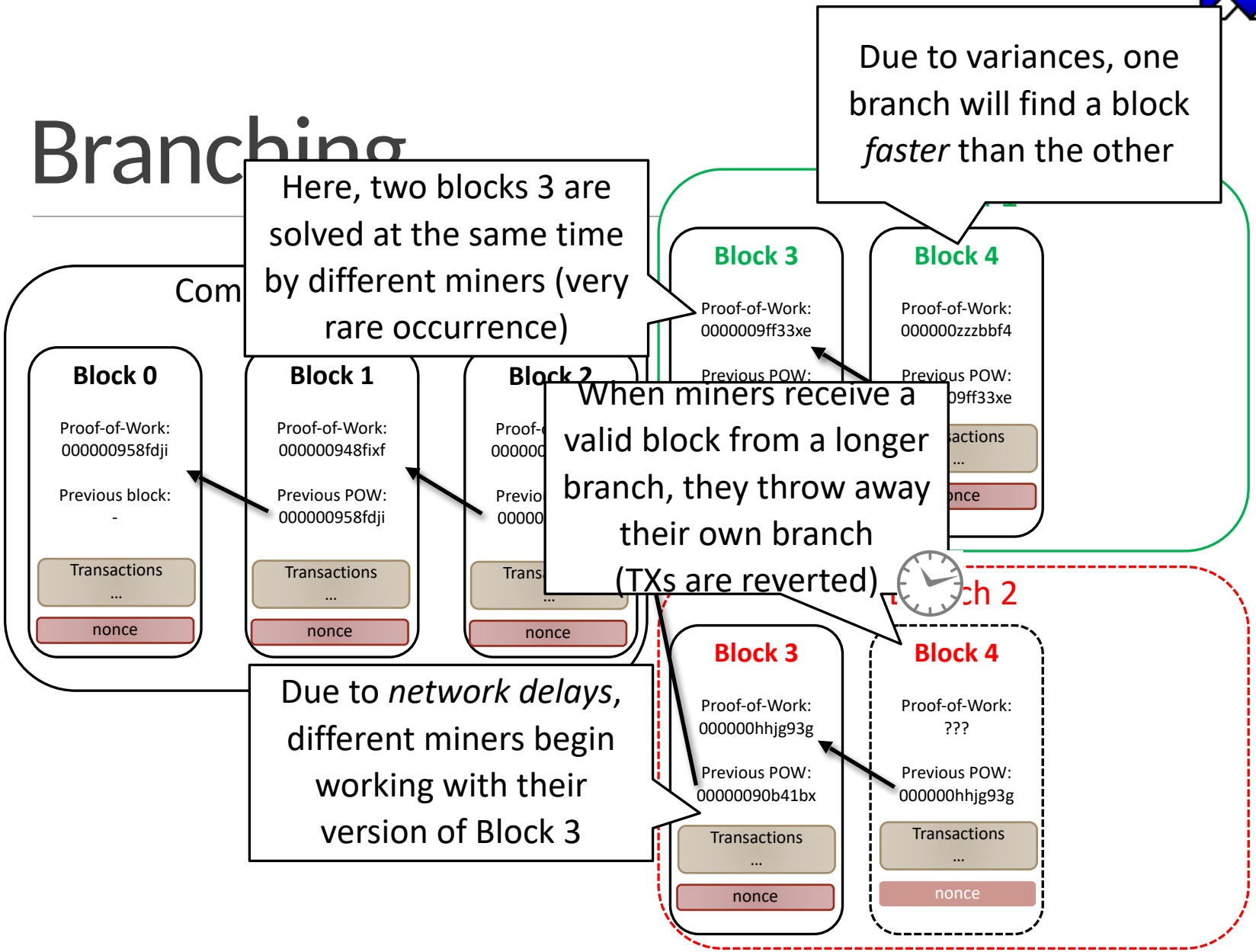
# Branching


Here, two blocks 3 are solved at the same time by different miners (very rare occurrence)
Com
Block 0
Proof-of-Work:
000000958fdji
Previous block:
-
Transactions
...
nonce
Block 1
Proof-of-Work:
000000948fixf
Previous POW:
000000958fdji
Transactions
...
nonce
Block 2
Proof-of-Work:
00000090b41bx
Previous POW:
000000948fixf
Transactions
...
nonce
Branch 1
Block 3
Proof-of-Work:
0000009ff33xe
Previous POW:
00000090b41bx
Transactions
...
nonce
Block 4
Proof-of-Work:
000000zzzbbf4
Previous POW:
0000009ff33xe
Transactions
...
nonce
Branch 2
Block 3
Proof-of-Work:
000000hhjg93g
Previous POW:
00000090b41bx
Transactions
...
nonce
Block 4
Proof-of-Work:
???
Previous POW:
000000hhjg93g
Transactions
...
nonce
Due to *network delays*, different miners begin working with their version of Block 3

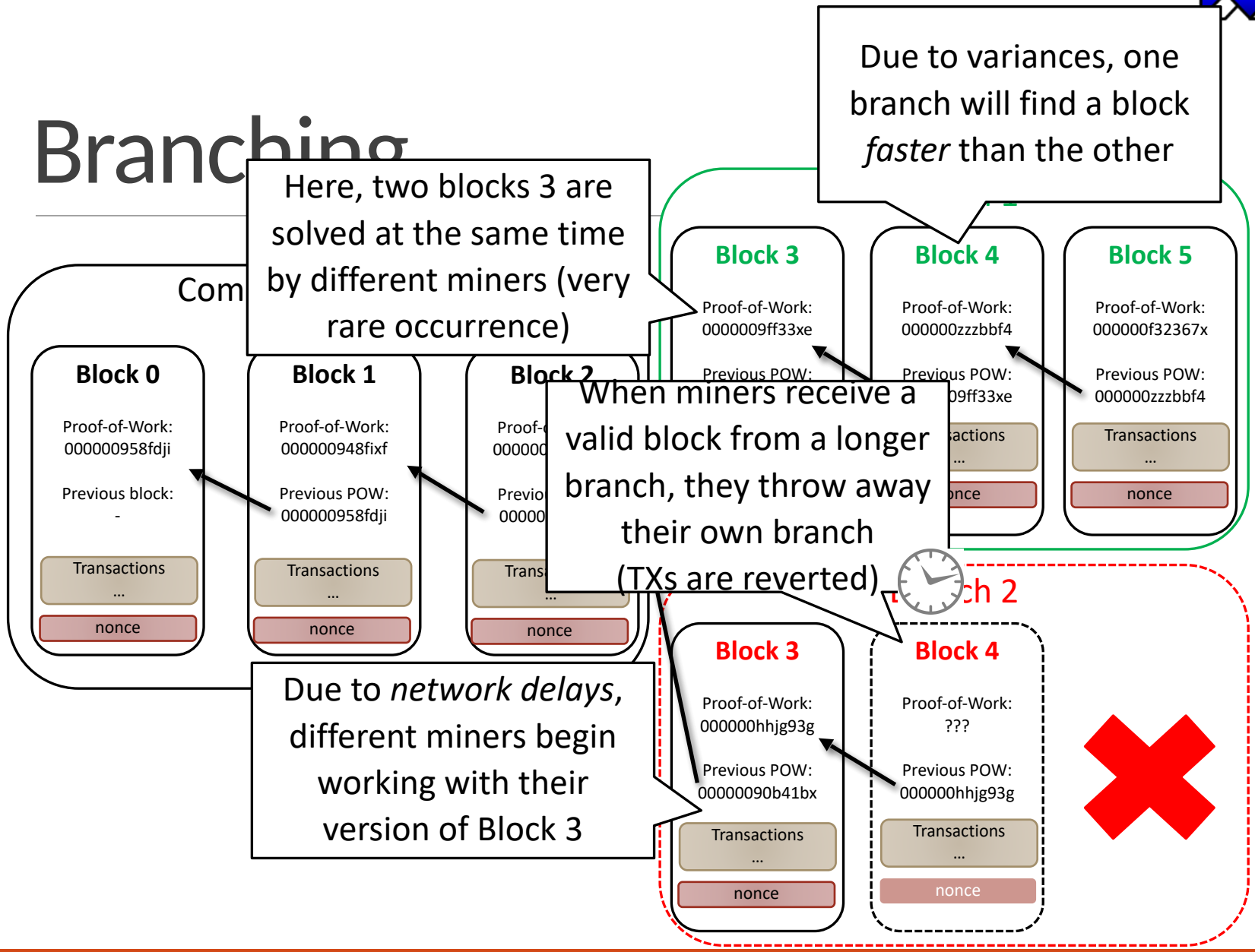
# Branching


Due to variances, one branch will find a block *faster* than the other
Here, two blocks 3 are solved at the same time by different miners (very rare occurrence)
Com
Block 0
Proof-of-Work: 000000958fdji
Previous block: -
Transactions ...
nonce
Block 1
Proof-of-Work: 000000948fixf
Previous POW: 000000958fdji
Transactions ...
nonce
Block 2
Proof-of-Work: 00000090b41bx
Previous POW: 000000948fixf
Transactions ...
nonce
Block 3
Proof-of-Work: 0000009ff33xe
Previous POW: 00000090b41bx
Transactions ...
nonce
Block 4
Proof-of-Work: 000000zzzbbf4
Previous POW: 0000009ff33xe
Transactions ...
nonce
Branch 2
Block 3
Proof-of-Work: 000000hhjg93g
Previous POW: 00000090b41bx
Transactions ...
nonce
Block 4
Proof-of-Work: ???
Previous POW: 000000hhjg93g
Transactions ...
nonce
Due to *network delays*, different miners begin working with their version of Block 3

# Branching


Due to variances, one branch will find a block *faster* than the other
Here, two blocks 3 are solved at the same time by different miners (very rare occurrence)
Com
Block 0
Proof-of-Work: 000000958fdji
Previous block: -
Transactions ...
nonce
Block 1
Proof-of-Work: 000000948fixf
Previous POW: 000000958fdji
Transactions ...
nonce
Block 2
Transactions ...
nonce
Block 3
Proof-of-Work: 0000009ff33xe
Previous POW:
Block 4
Proof-of-Work: 000000zzzbbf4
Previous POW:
09ff33xe
When miners receive a valid block from a longer branch, they throw away their own branch (TXs are reverted)
ch 2
Block 3
Proof-of-Work: 000000hhjg93g
Previous POW: 00000090b41bx
Transactions ...
nonce
Block 4
Proof-of-Work: ???
Previous POW: 000000hhjg93g
Transactions ...
nonce
Due to *network delays*, different miners begin working with their version of Block 3

# Branching


Due to variances, one branch will find a block *faster* than the other
Here, two blocks 3 are solved at the same time by different miners (very rare occurrence)
When miners receive a valid block from a longer branch, they throw away their own branch (TXs are reverted)
Due to *network delays*, different miners begin working with their version of Block 3
Block 0
Proof-of-Work: 000000958fdji
Previous block: -
Transactions ...
nonce
Block 1
Proof-of-Work: 000000948fixf
Previous POW: 000000958fdji
Transactions ...
nonce
Block 3
Proof-of-Work: 0000009ff33xe
Transactions ...
nonce
Block 4
Proof-of-Work: 000000zzzbbf4
Transactions ...
nonce
Block 5
Proof-of-Work: 000000f32367x
Previous POW: 000000zzzbbf4
Transactions ...
nonce
Block 3
Proof-of-Work: 000000hhjg93g
Previous POW: 00000090b41bx
Transactions ...
nonce
Block 4
Proof-of-Work: ???
Previous POW: 000000hhjg93g
Transactions ...
nonce

# Incentives

$$$$$$

**Block reward**, started with 50 BTC, 25 BTC, 12.5 BTC. …

- Creating of new coins (the only means to create coins)
- Reward reaped by miner whose block ultimately makes it into the chain
- Block reward will converge toward zero

**Transaction fee**

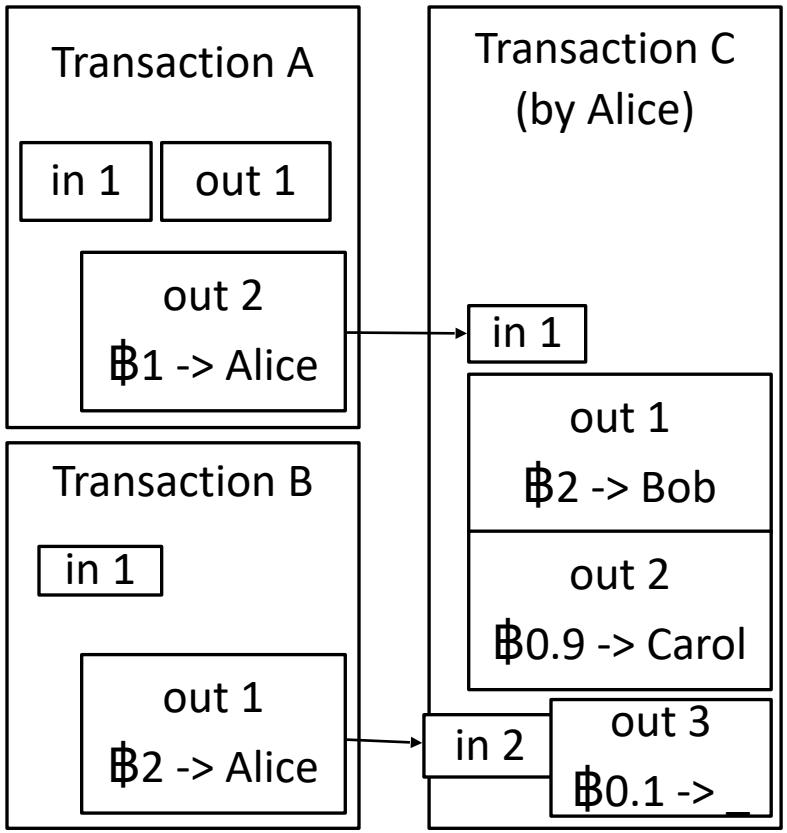
- Small amount that is paid by transaction issuer to miner
- Not a fixed amount, amount declared by issuer
- Ultimately, market forces may set this value

MIDDLEWARE SYSTEMS
RESEARCH GROUP
MSRG.ORG

# Deconstructing Blockchains: Concepts, Systems and Applications

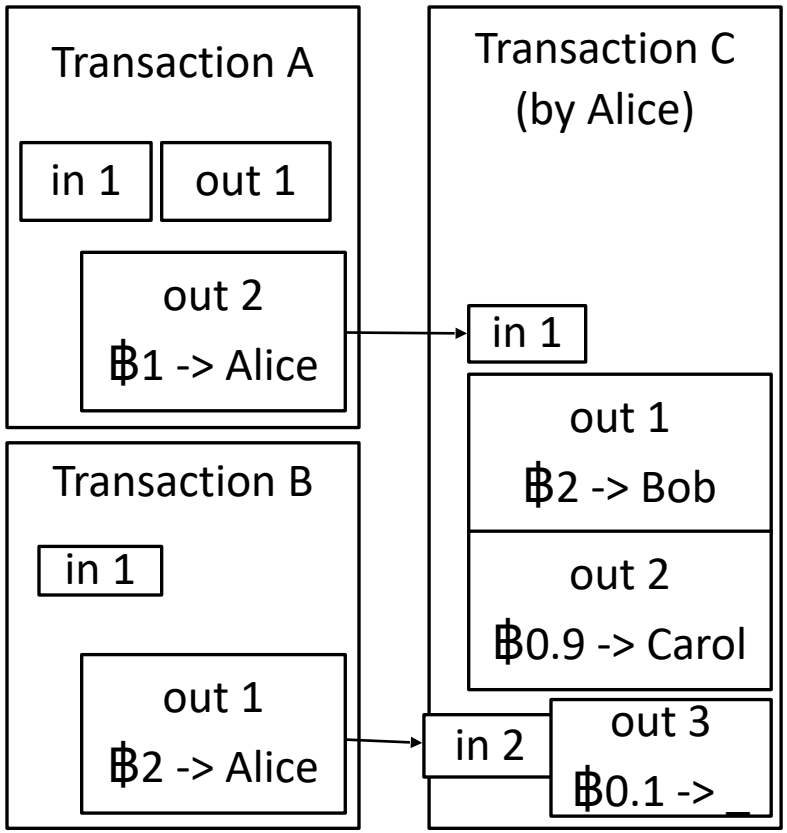
## TRANSACTIONS AND TRANSACTION FLOW

# Bitcoin Transactions



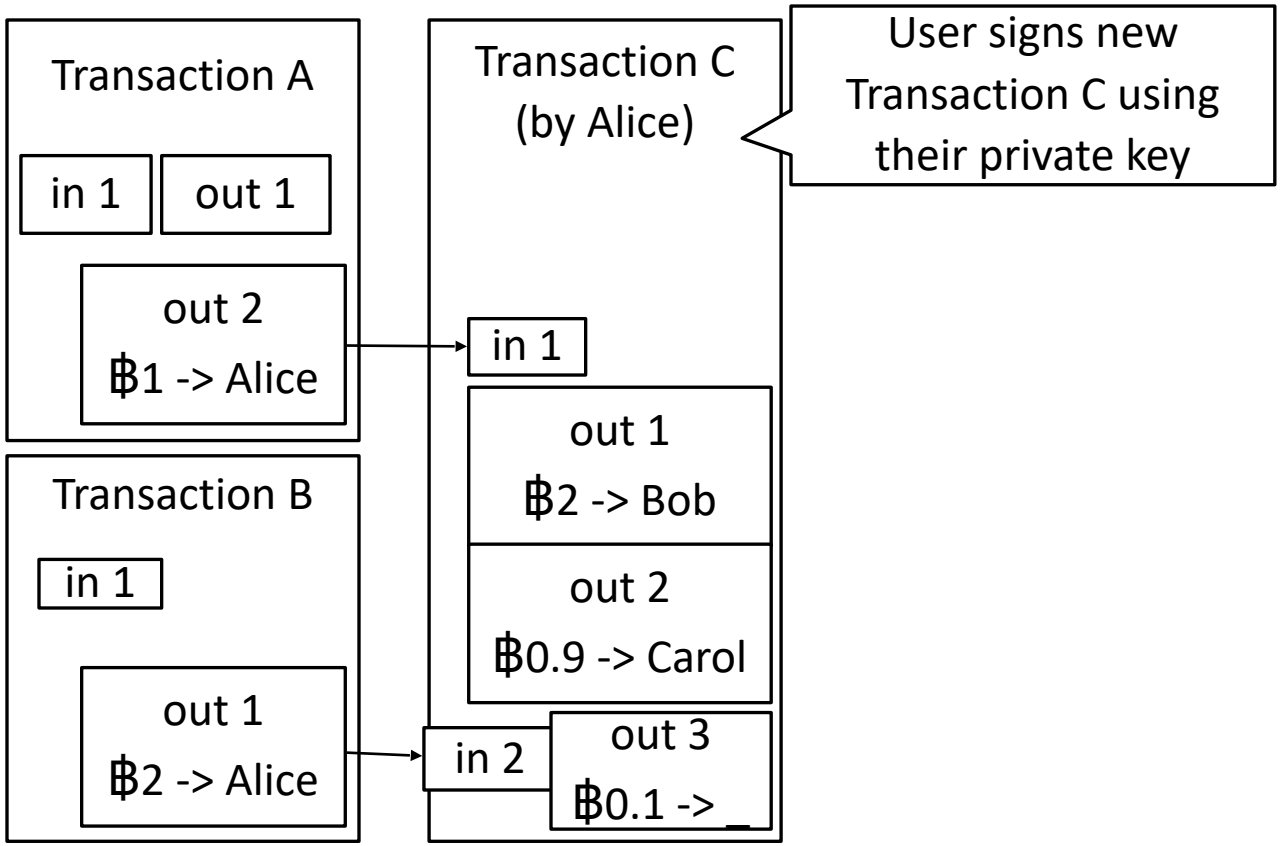
# Bitcoin Transactions

Each user possesses a wallet identified by *public/private key* pairs



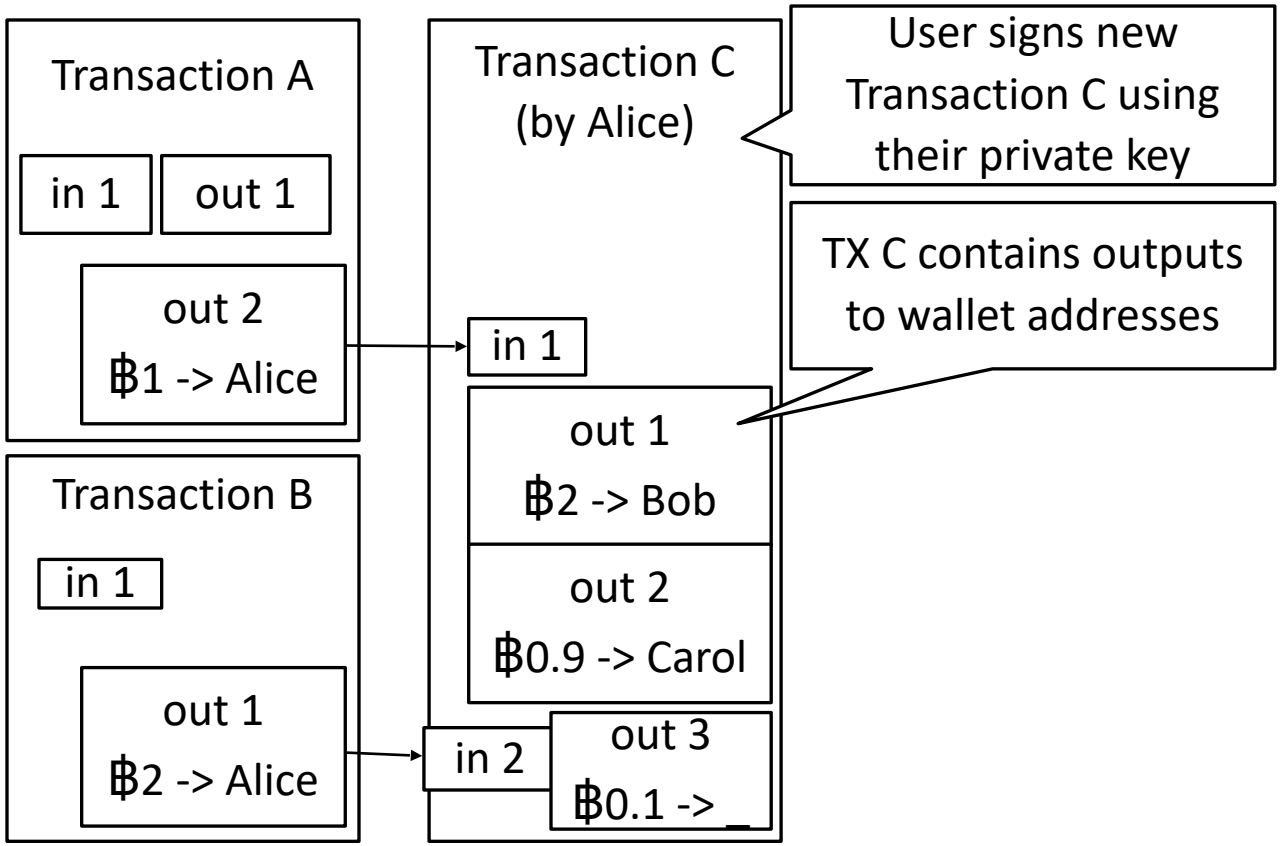
# Bitcoin Transactions

Each user possesses a wallet identified by *public/private key* pairs



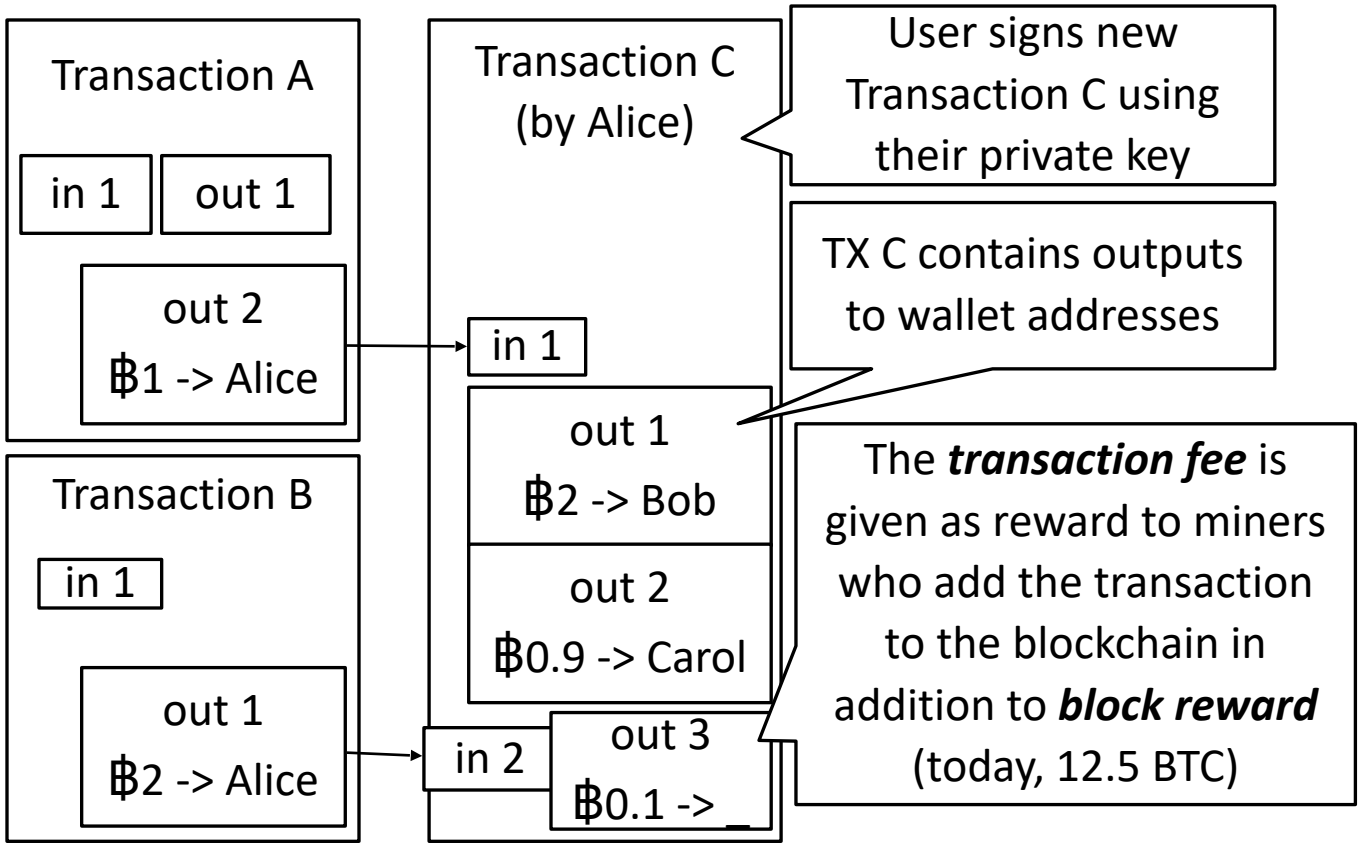
# Bitcoin Transactions

Each user possesses a wallet identified by *public/private key* pairs

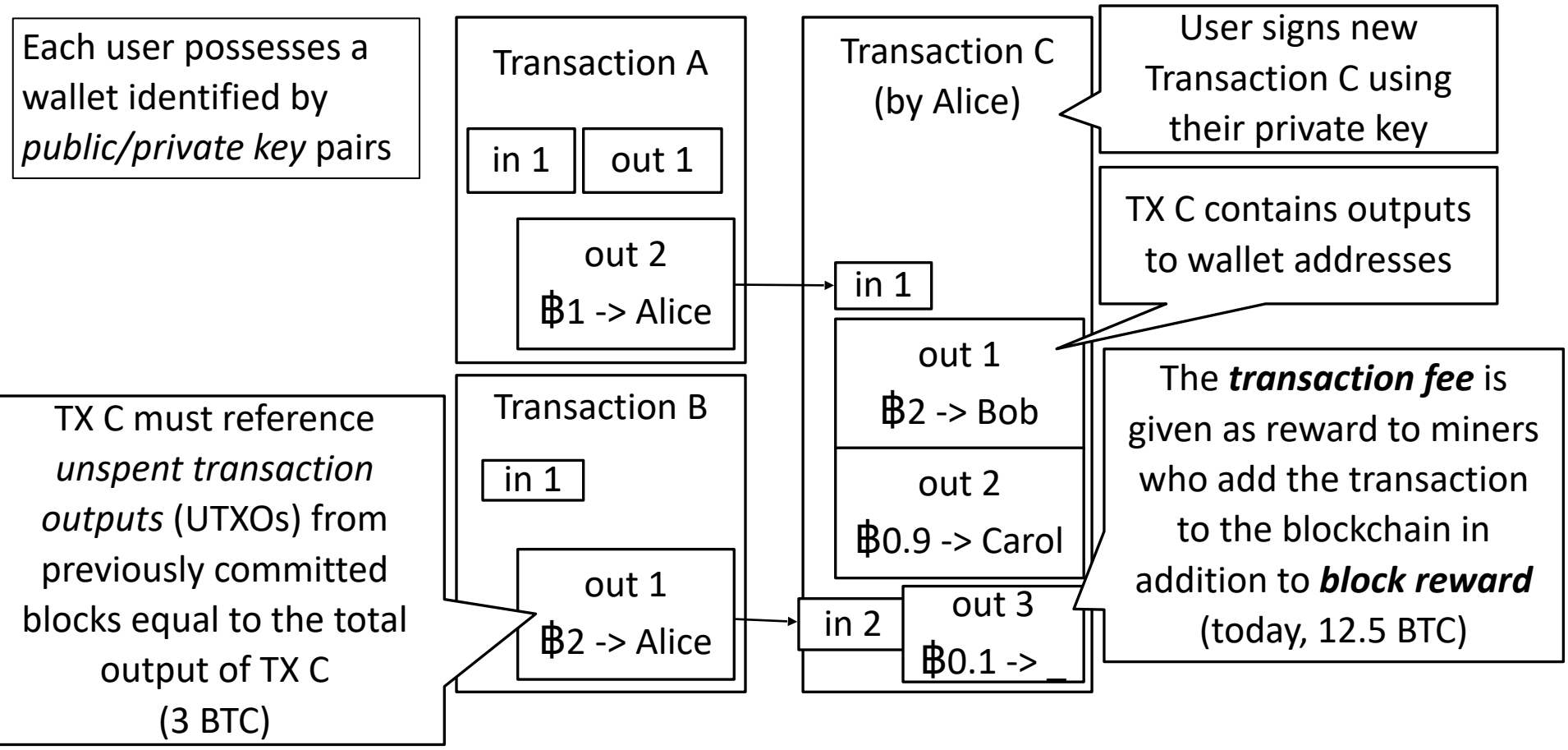


# Bitcoin Transactions

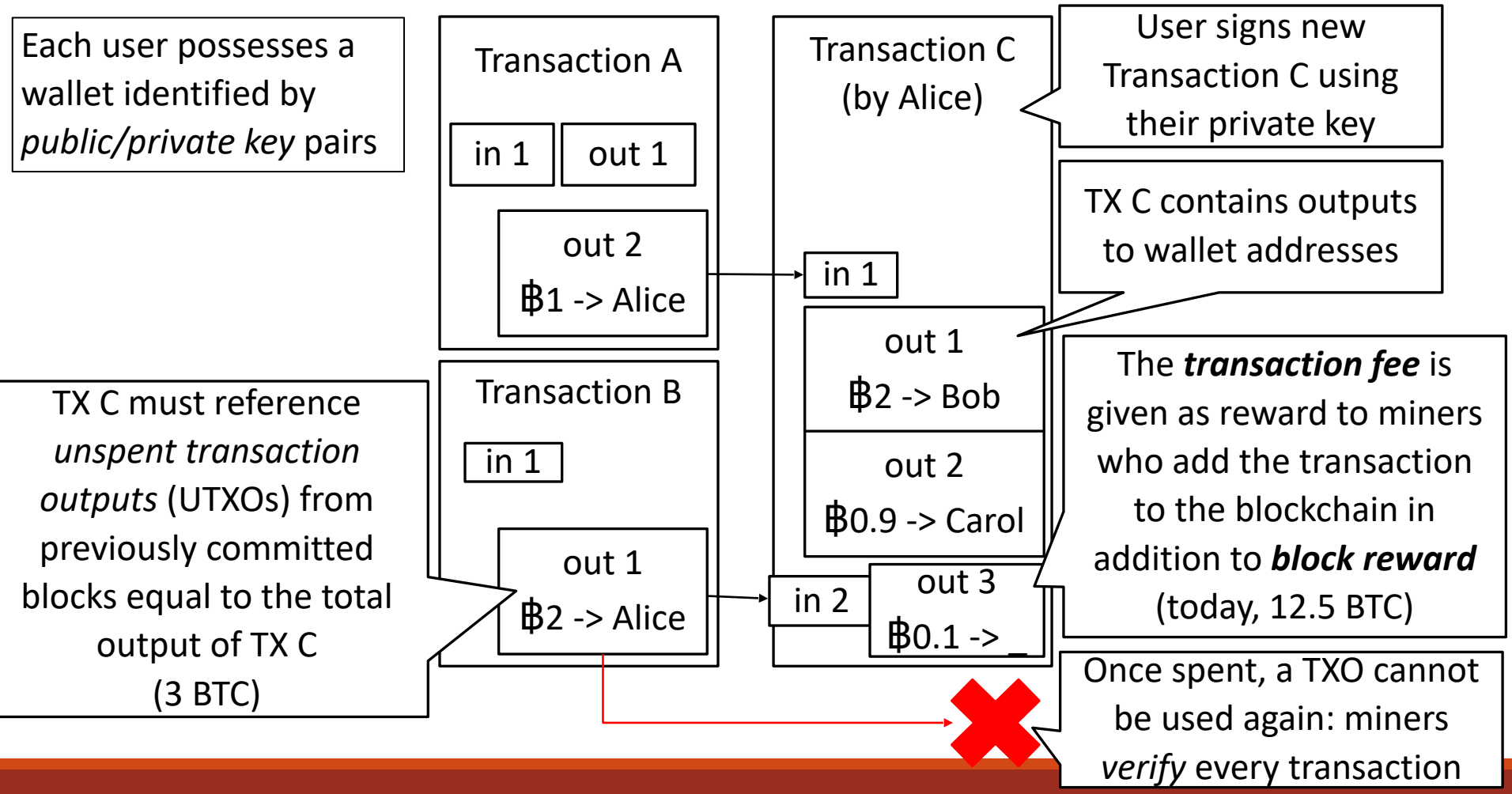
Each user possesses a wallet identified by *public/private key* pairs



# Bitcoin Transactions



# Bitcoin Transactions



# Wallets and Addresses

Users require a **wallet** to store money

- Includes any user, including but not limited to miners

Wallet is authenticated and identified by a **public/private key pair**

- Generated using ECDSA (Elliptic curve cryptography)

# Loosing Your Private Key

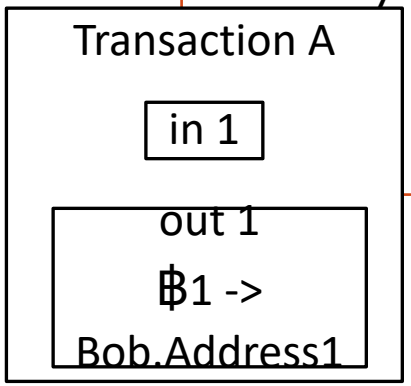
- Loss of private key means the wallet and its **funds are permanently locked**, as it is no longer possible to sign proofs redeeming existing UTXOs.
- This **money is** essentially **lost**, thereby reducing the total amount of currency in Bitcoin
- Trusting an online service to store key is also risky, since there is no way to prove that you are the rightful owner if the key is stolen or misused
- **The most reliable solution is to store your private keys on tamper-proof hardware wallets**

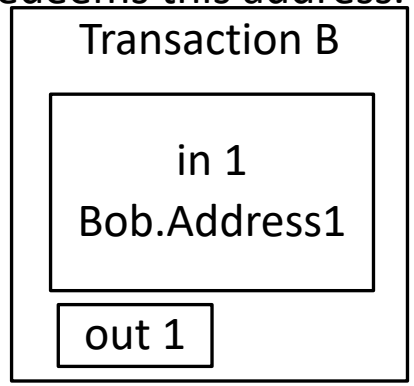
# Transaction Flow

Alice
(Sender)

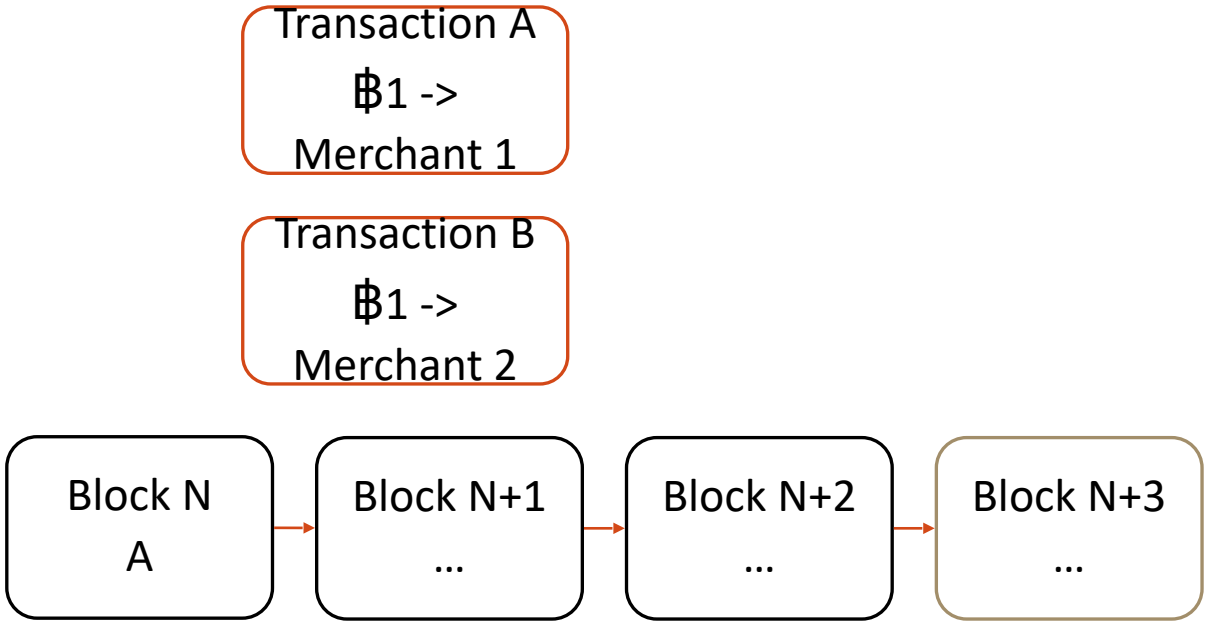
Bob
(Receiver)

1. Bob generates and sends a public key address.
2. Alice creates a transaction using this address.
3. Alice sends the new transaction to the network.
4. Transaction is broadcasted using gossiping.
5. Transaction is included in a mined block.
6. Bob can verify the transaction is in the blockchain.
7. Bob can now sign new transactions which redeems this address.



# Preventing Double Spending: 51% Attack



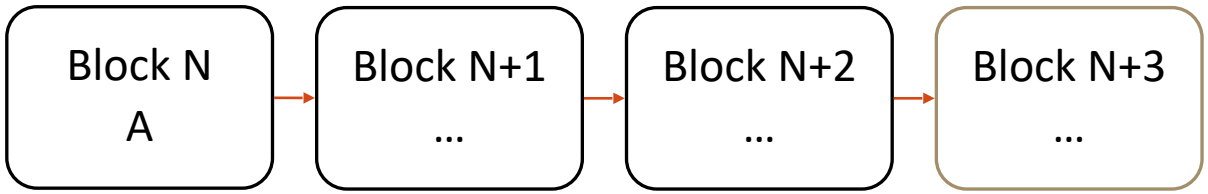

Real chain

# Preventing Double Spending: 51% Attack

Transaction A
฿1 ->
Merchant 1

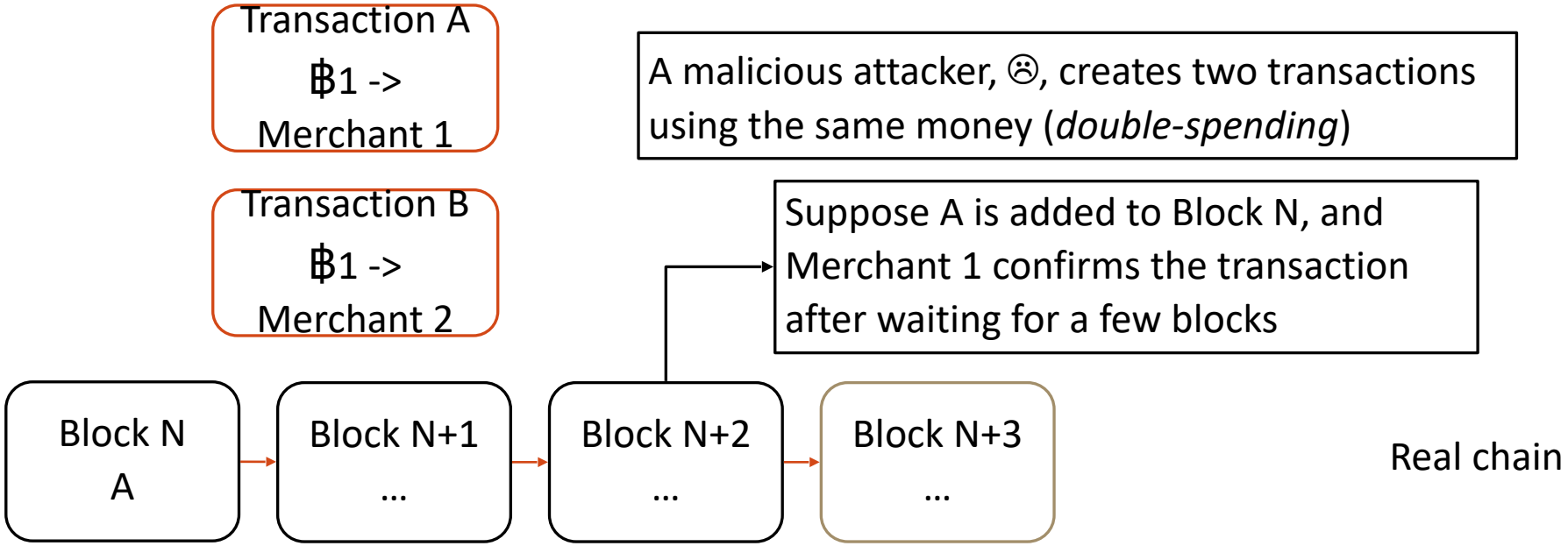
Transaction B
฿1 ->
Merchant 2

A malicious attacker, ☹, creates two transactions using the same money (*double-spending*)




Real chain

# Preventing Double Spending: 51% Attack



# Preventing Double Spending: 51% Attack

Transaction A
฿1 ->
Merchant 1

Transaction B
฿1 ->
Merchant 2

A malicious attacker, ☹, creates two transactions using the same money (*double-spending*)

Suppose A is added to Block N, and Merchant 1 confirms the transaction after waiting for a few blocks

Block N
A

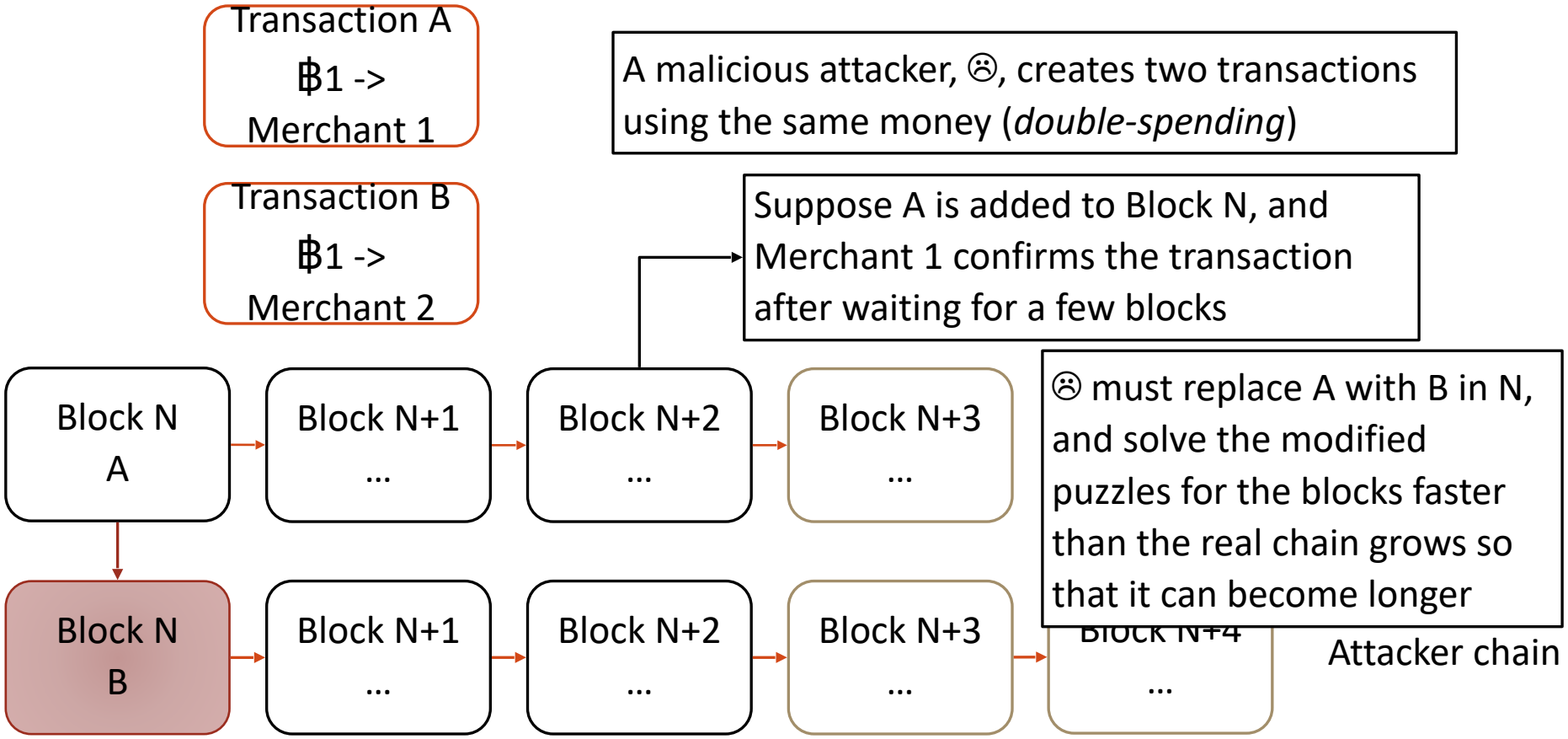
Block N+1
…

Block N+2
…

Block N+3
…

☹ must replace A with B in N, and solve the modified puzzles for the blocks faster than the real chain grows so that it can become longer

# Preventing Double Spending: 51% Attack



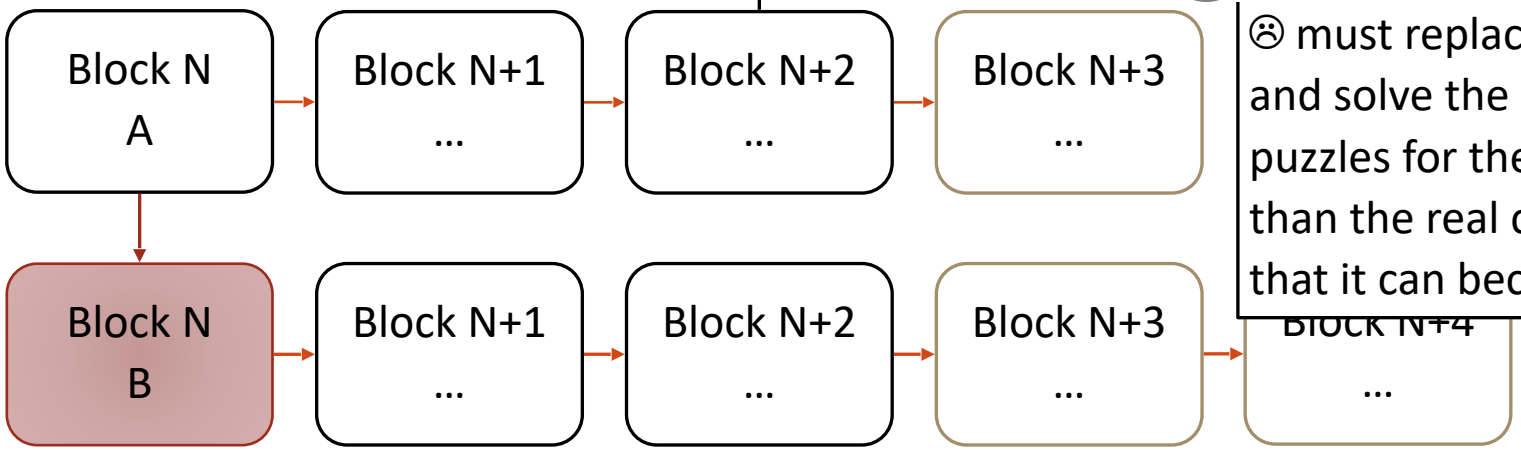
# Preventing Double Spending: 51% Attack

- The "Magic Watch" is the ***continuous generation*** of blocks in the main chain which *limits the amount of time* an attacker has to create its own chain.
- If the attacker owns *>51% of the power* in the network, the "Magic Watch" gives *enough time* to the attacker to *tamper with the data (i.e., re-write history)*!

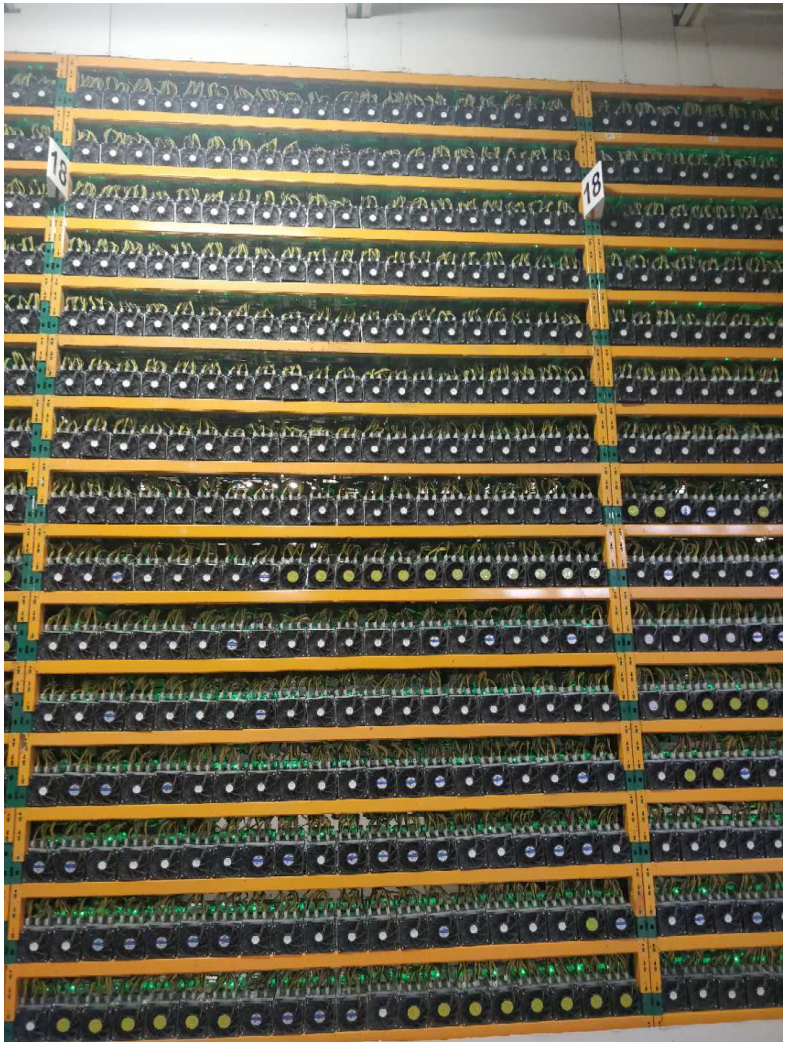
reates two transactions
*louble-spending*)

ed to Block N, and
rms the transaction
few blocks

☹ must replace A with B in N, and solve the modified puzzles for the blocks faster than the real chain grows so that it can become longer




Attacker chain



Mining ASICs,
200 PETA Hashes
per second

# Limitations of Bitcoin

# Limitations of Bitcoin

Limited expressiveness

- Cryptocurrency only
- Each app requires new platform (e.g. NameCoin, PrimeCoin, CureCoin)

Slow block time (10 mins)

- Also slow confirmation time (1+ hour for 6 confirmations)

Hard/Soft forks

- Updates to the code cause forks
- Hard forks are not compatible
- Duplicated money
- 5 Bitcoin forks (e.g., Bitcoin Classic)

# Limitations of Bitcoin

Limited expressiveness
- Cryptocurrency only
- Each app requires new platform (e.g. NameCoin, PrimeCoin, CureCoin)

Slow block time (10 mins)
- Also slow confirmation time (1+ hour for 6 confirmations)

Hard/Soft forks
- Updates to the code cause forks
- Hard forks are not compatible
- Duplicated money
- 5 Bitcoin forks (e.g., Bitcoin Classic)

Slow transaction rate
- 7 transactions/second
- VISA Network: 2000 tps (average)
- Limited block size (1MB -> 2MB)

Environmental impact of PoW
- ~1000x more energy than credit card
- Ahead of 159 countries for energy consumption (e.g. Ireland)

Long bootstrap time for a miner
- Full ledger: 270 GB (2020/04)
- CPU/I/O cost to verify each transaction/block
- Takes hours/days

MIDDLEWARE SYSTEMS
RESEARCH GROUP
MSRG.ORG

**Please do not forget to provide your feedback via the course evaluation.**

# Deconstructing Blockchains: Concepts, Systems and Applications

PLATFORMS AND APPLICATIONS

# Blockchain Platforms

ETHEREUM

HYPERLEDGER

# ETHEREUM

Managing entity: Ethereum Foundation

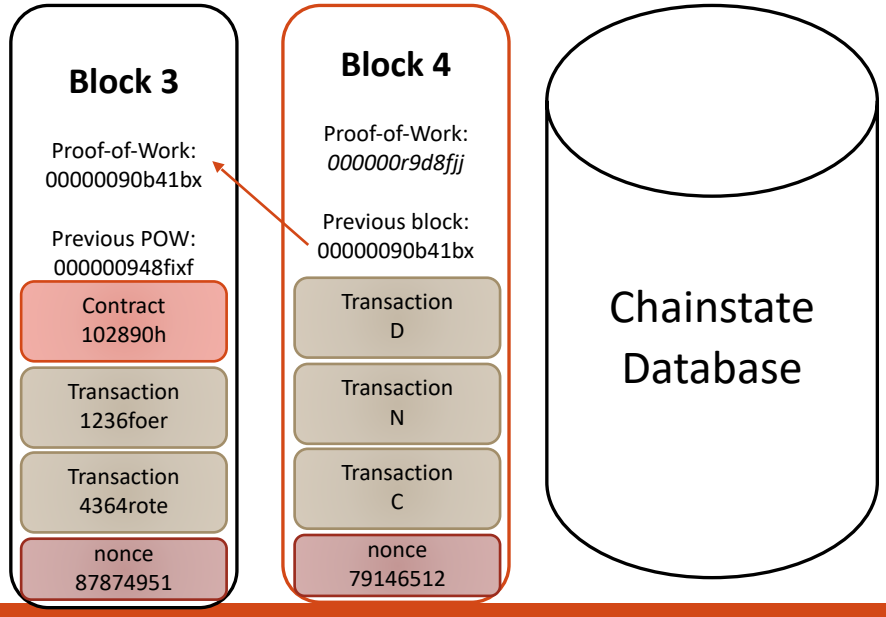
- Major players: Deloitte, Toyota, Microsoft, …

Enable decentralized applications (Dapps) *et al.*

Open-source, flexible, general platform

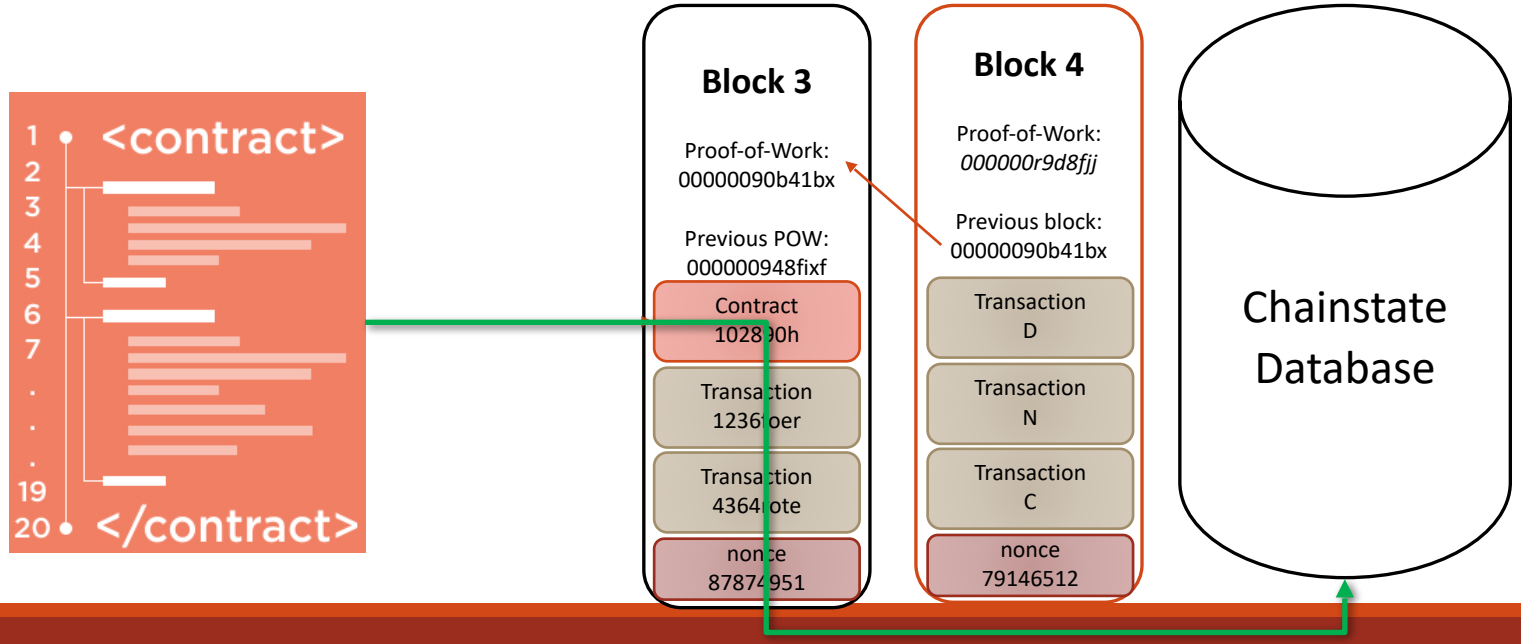
- Permisionless (public) ledger, proof-of-work-based(alternative mechanisms are work in progress)
- Cryptocurrency: 1 Ether = 1e18 Wei (~150 USD, 2020/4)
- Smart contracts: Solidity, Remix (Web IDE), Truffle (Dev./ Test), *Viper* (programming language to build Dapps)
- Ethereum Virtual Machine (EVM)

# Smart Contracts



# Smart Contracts

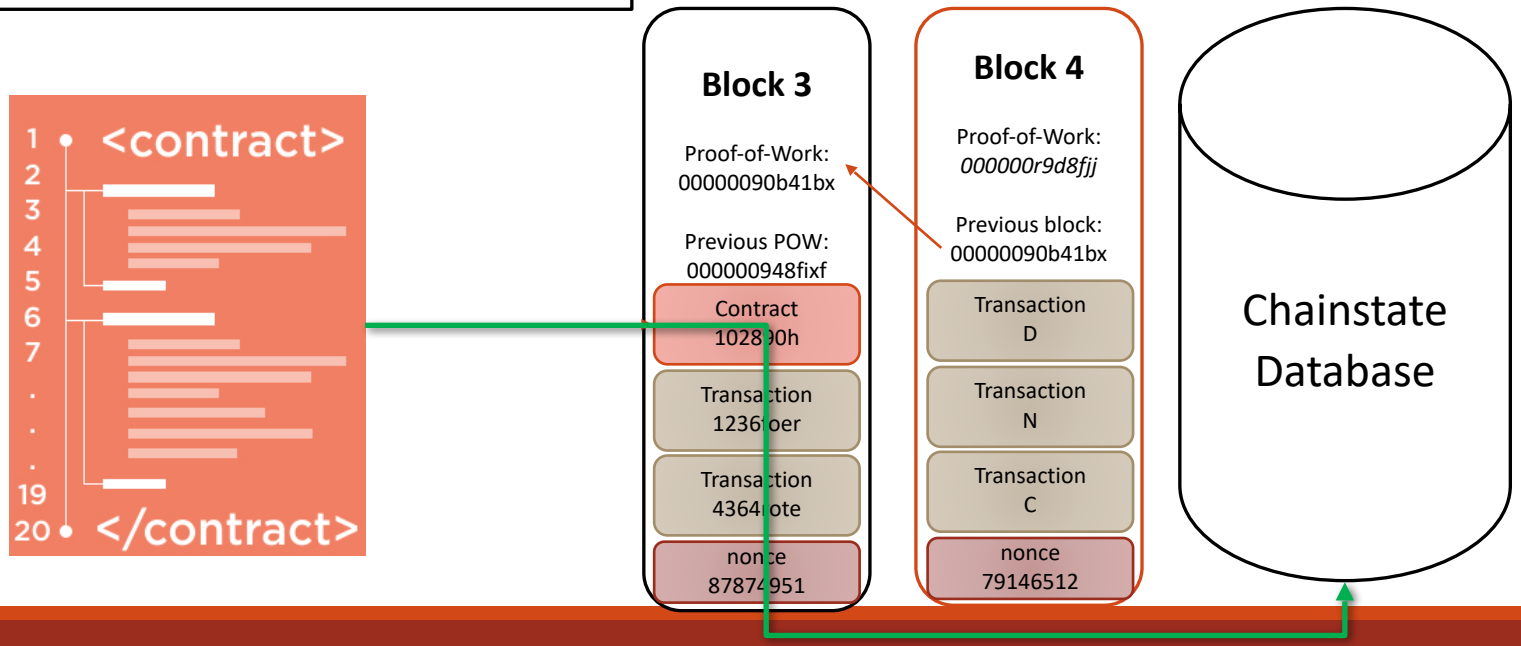
- *Contracts* are programs, compiled into bytecode to execute on EVMs
- Contracts have internal storage



# Smart Contracts

- *Contracts* are programs, compiled into bytecode to execute on EVMs
- Contracts have internal storage

- Contracts execute when triggered by a transaction (or by another contract)
- Execution time is limited by *gas*



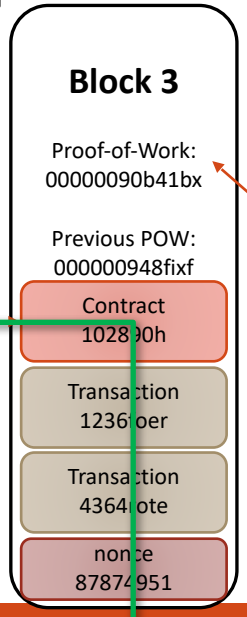
# Smart Contracts

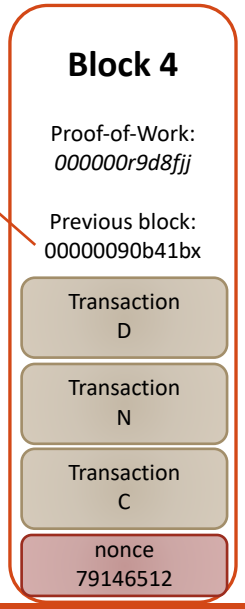
- *Contracts* are programs, compiled into bytecode to execute on EVMs
- Contracts have internal storage

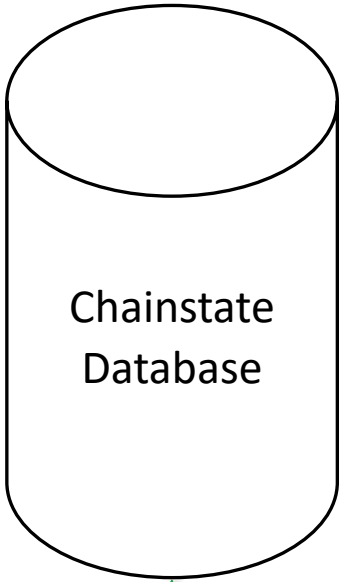
- Contracts execute when triggered by a transaction (or by another contract)
- Execution time is limited by *gas*

**Example**: Land registry

| Wallet ID | Held Titles |
|---|---|
| 99823428347 | 34356,324324 |
| 98217981623 | 677343,4444 |
| 90987344755 | 994,38842,439 |

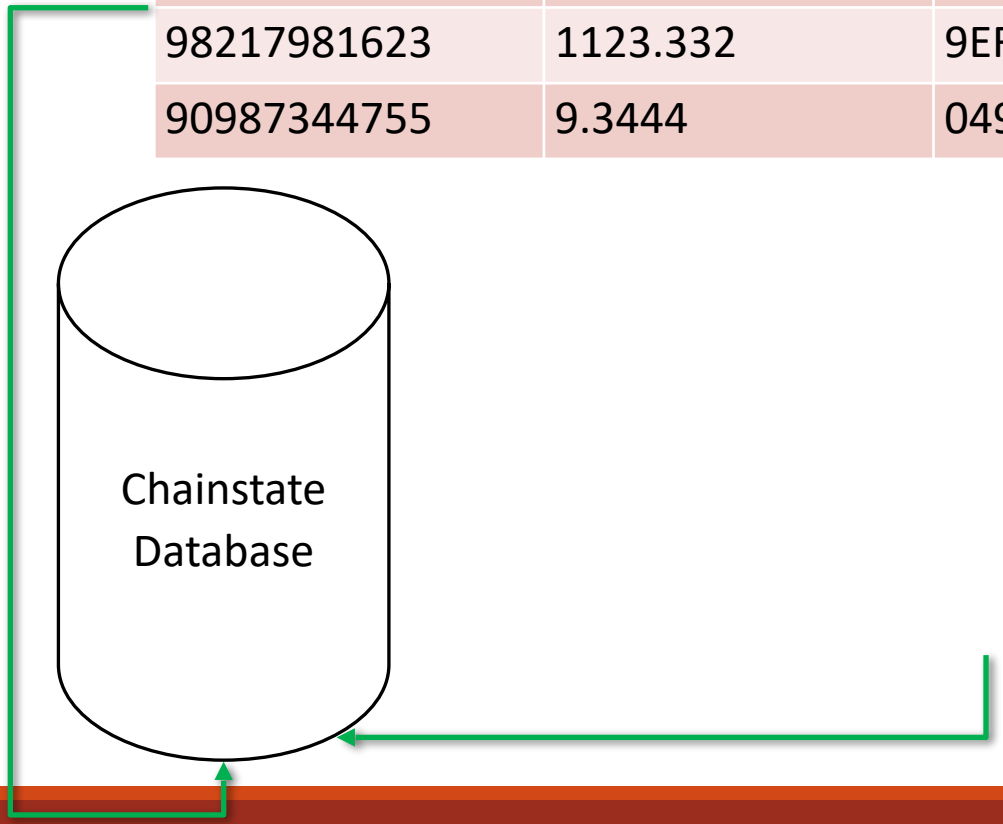



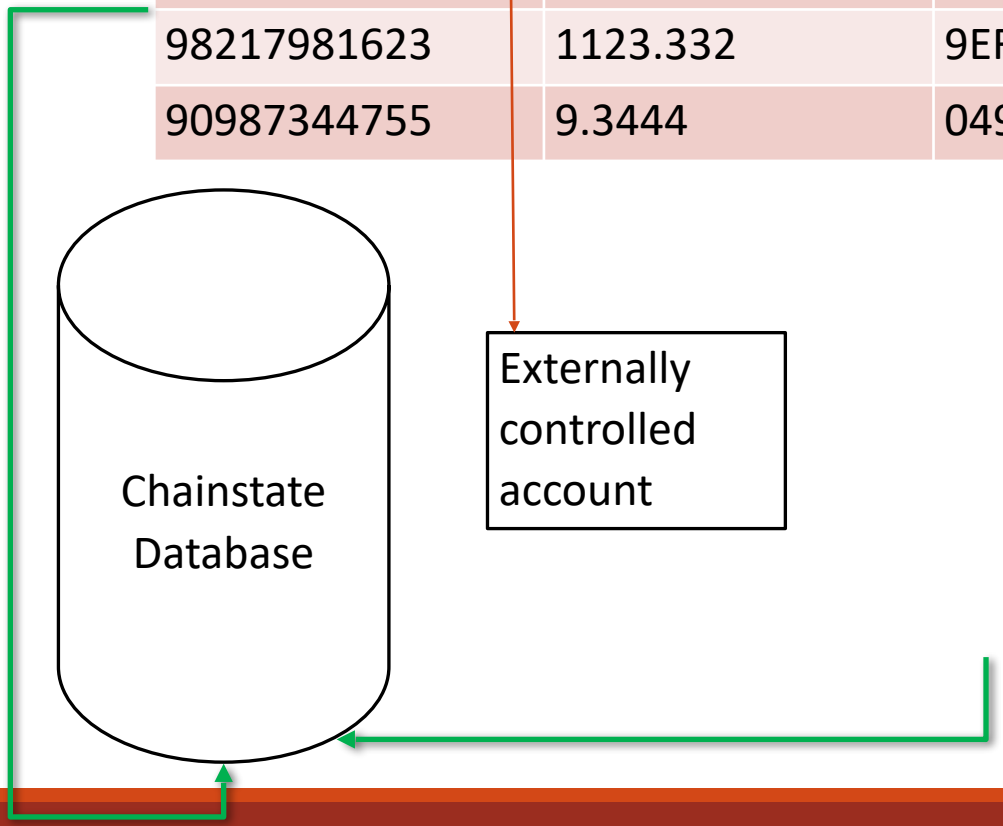
# Account State ("World State")

| Wallet ID | Balance | Code Hash | Internal State |
|---|---|---|---|
| 99823428347 | 45.12 | - | 99554HGJ |
| 98217981623 | 1123.332 | 9ERU12T4 | 3453ADFG |
| 90987344755 | 9.3444 | 0490CNDJ | 132GJR4 |



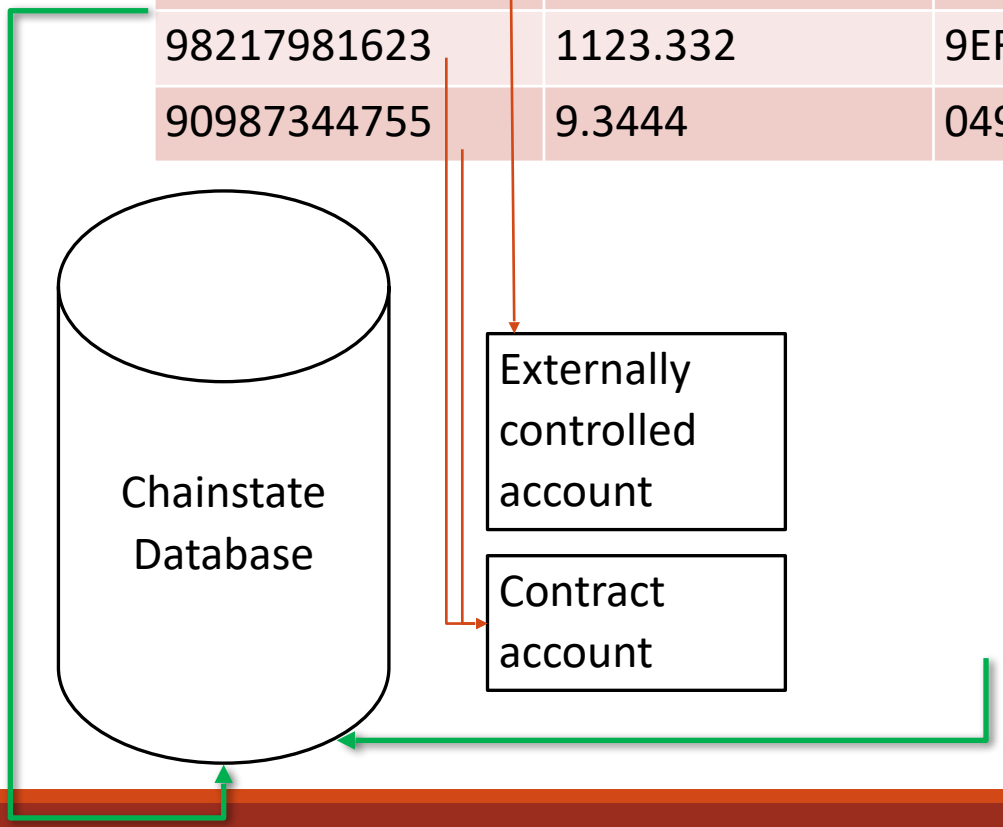
# Account State ("World State")

| Wallet ID | Balance | Code Hash | Internal State |
|---|---|---|---|
| 99823428347 | 45.12 | - | 99554HGJ |
| 98217981623 | 1123.332 | 9ERU12T4 | 3453ADFG |
| 90987344755 | 9.3444 | 0490CNDJ | 132GJR4 |



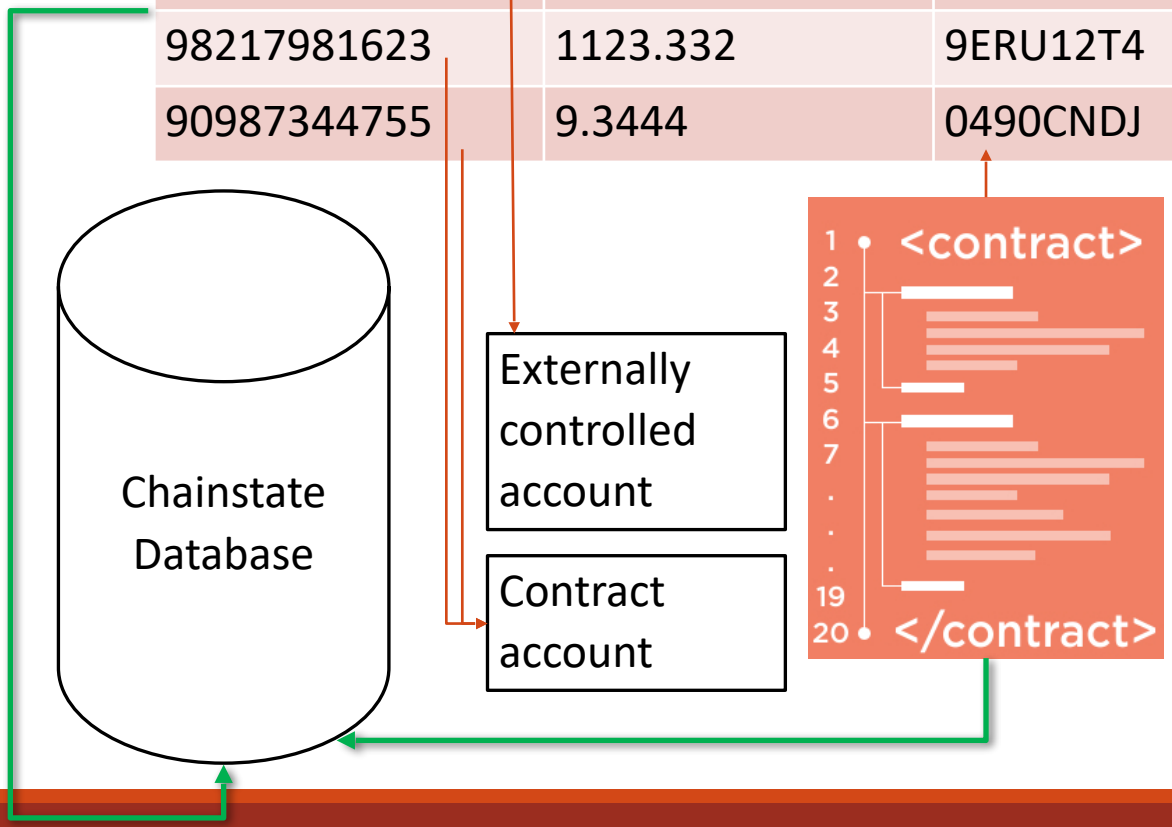
# Account State ("World State")

| Wallet ID | Balance | Code Hash | Internal State |
|---|---|---|---|
| 99823428347 | 45.12 | - | 99554HGJ |
| 98217981623 | 1123.332 | 9ERU12T4 | 3453ADFG |
| 90987344755 | 9.3444 | 0490CNDJ | 132GJR4 |



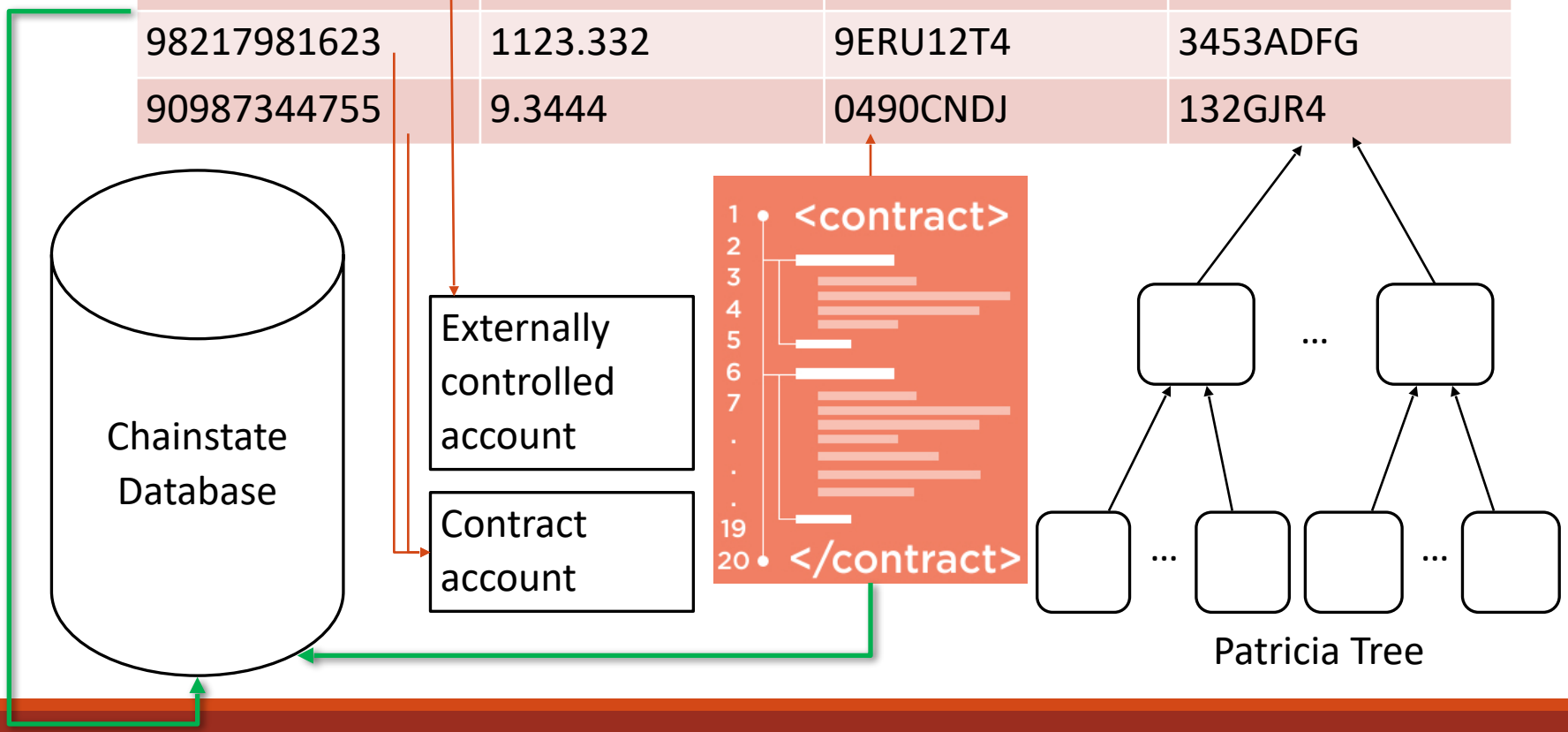
# Account State ("World State")

| Wallet ID | Balance | Code Hash | Internal State |
|---|---|---|---|
| 99823428347 | 45.12 | - | 99554HGJ |
| 98217981623 | 1123.332 | 9ERU12T4 | 3453ADFG |
| 90987344755 | 9.3444 | 0490CNDJ | 132GJR4 |

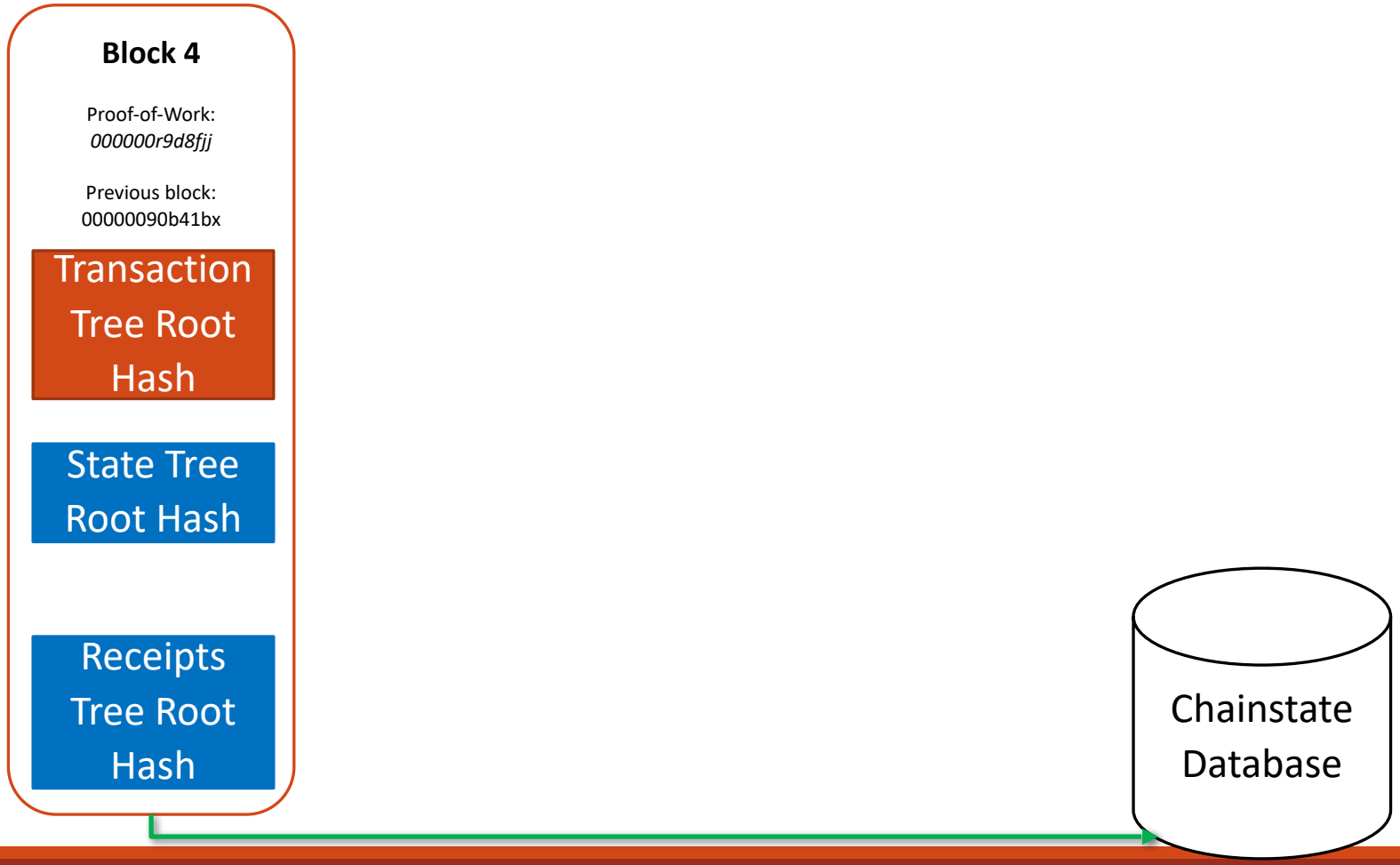


# Account State ("World State")

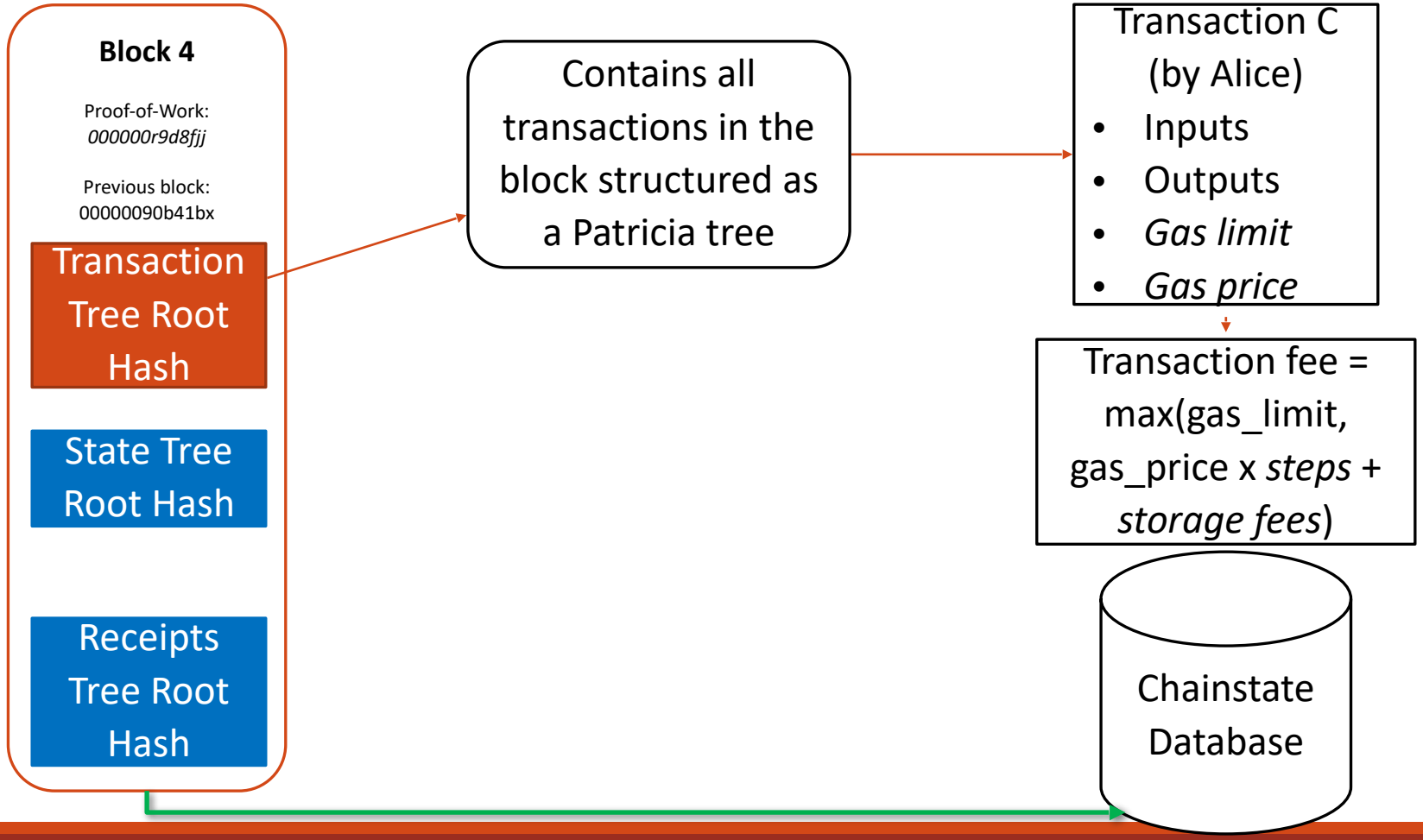
| Wallet ID | Balance | Code Hash | Internal State |
|---|---|---|---|
| 99823428347 | 45.12 | - | 99554HGJ |
| 98217981623 | 1123.332 | 9ERU12T4 | 3453ADFG |
| 90987344755 | 9.3444 | 0490CNDJ | 132GJR4 |



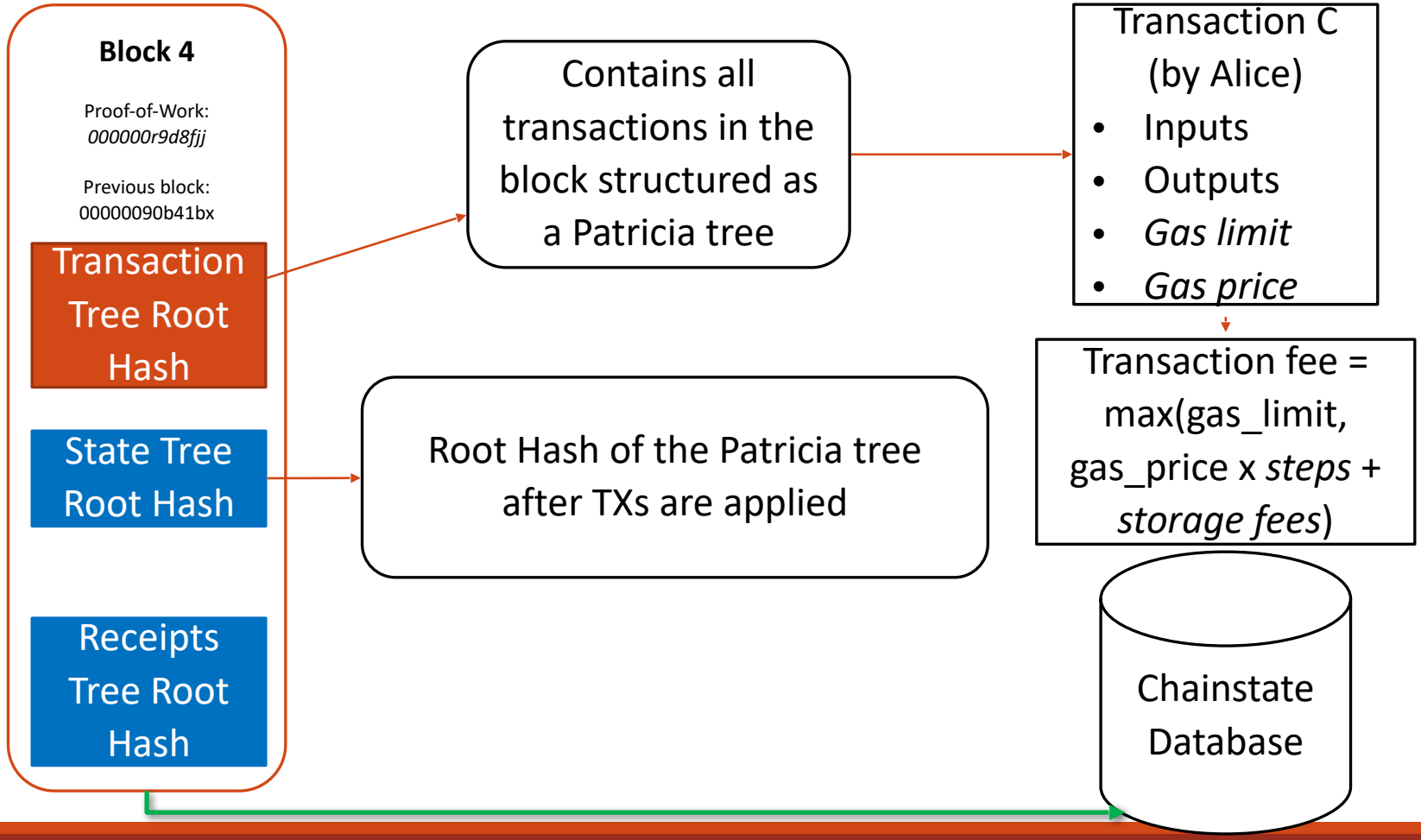
# Execution


Block 4
Proof-of-Work:
000000r9d8fjj
Previous block:
00000090b41bx
Transaction Tree Root Hash
State Tree Root Hash
Receipts Tree Root Hash
Chainstate Database

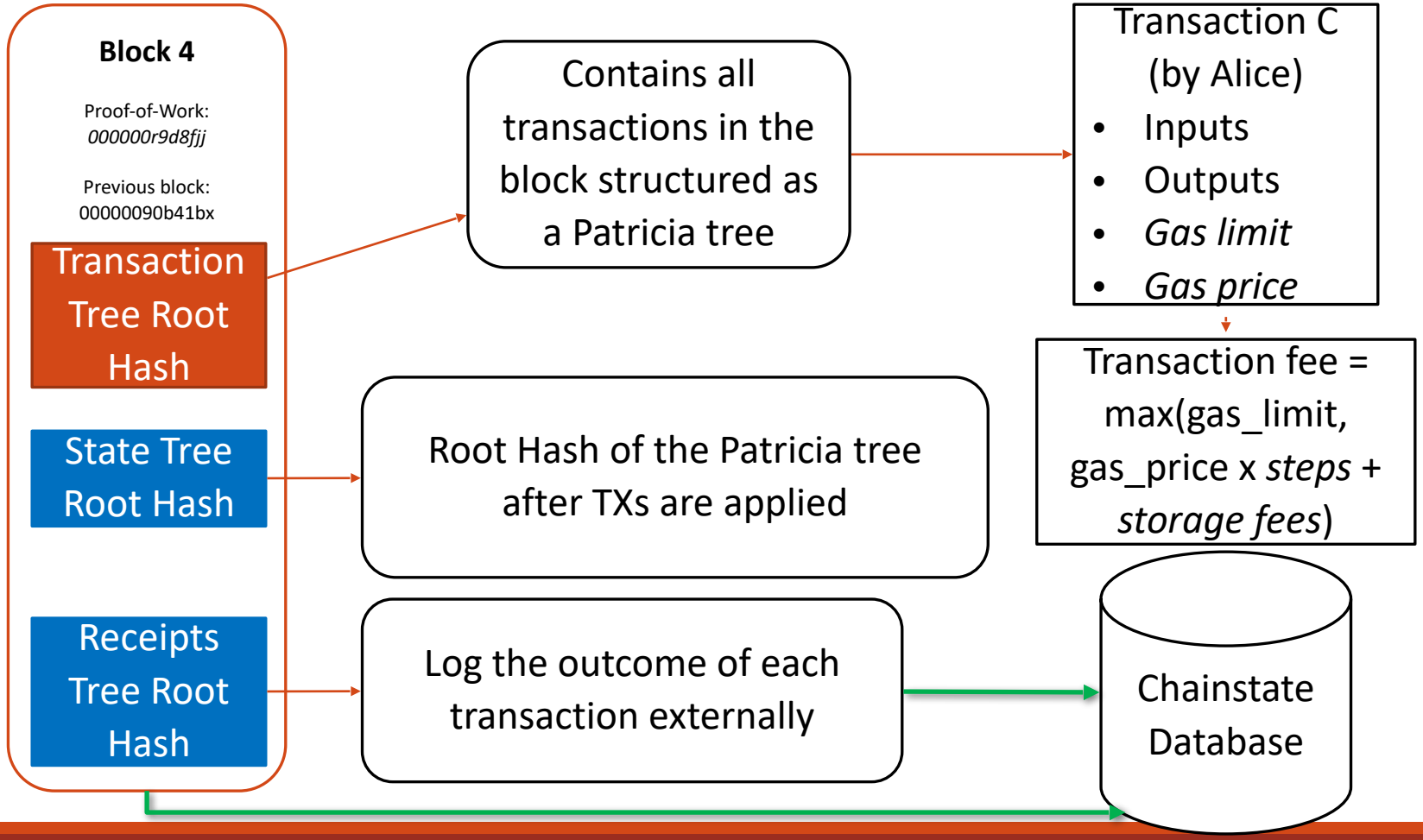
# Execution


Block 4
Proof-of-Work:
000000r9d8fjj
Previous block:
00000090b41bx
Transaction Tree Root Hash
State Tree Root Hash
Receipts Tree Root Hash
Contains all transactions in the block structured as a Patricia tree
Transaction C (by Alice)
Inputs
Outputs
Gas limit
Gas price
Transaction fee = max(gas_limit, gas_price x steps + storage fees)
Chainstate Database

# Execution


Block 4
Proof-of-Work:
000000r9d8fjj
Previous block:
00000090b41bx
Transaction Tree Root Hash
State Tree Root Hash
Receipts Tree Root Hash
Contains all transactions in the block structured as a Patricia tree
Transaction C (by Alice)
Inputs
Outputs
Gas limit
Gas price
Transaction fee = max(gas_limit, gas_price x steps + storage fees)
Root Hash of the Patricia tree after TXs are applied
Chainstate Database

# Execution


Block 4
Proof-of-Work:
000000r9d8fjj
Previous block:
00000090b41bx
Transaction Tree Root Hash
State Tree Root Hash
Receipts Tree Root Hash
Contains all transactions in the block structured as a Patricia tree
Transaction C (by Alice)
Inputs
Outputs
Gas limit
Gas price
Transaction fee = max(gas_limit, gas_price x steps + storage fees)
Root Hash of the Patricia tree after TXs are applied
Log the outcome of each transaction externally
Chainstate Database

# HYPERLEDGER

Managing entity: Hyperledger Consortium
- Major players: IBM, NEC, Intel, R3, …

Enterprise blockchains
- Permissioned ledger (private and consortium networks)
- Smart contracts in general purpose language(s)
- Open-source, configurable, pluggable consensus
- World state on CouchDB, LevelDB, *et al*.

Hyperledger is a family of projects
- Fabric: PBFT Consensus *et al.*
- Sawtooth: Proof-of-elapsed time (using Intel SGX)
- Composer: Smart contract language and development tool
- Cello: Blockchain-as-a-Service framework

# HYPERLEDGER

Key **differentiators**

- Assumes a more trusted environment than Bitcoin/Ethereum
- Requires authentication to partake in business network
- Dozens of peers that manage distributed ledger (not 1000s)
- No cryptocurrency, no tokens (could be build on top)
- No proof-of-work-based consensus (traditional consensus)
- No mining, no intrinsic inventive mechanisms

Intended **use cases**

- Trade finance (tracking financial transactions and goods)
- Supply chains, logistics (tracking goods, assets, etc.)
- Cross-border trade
- Inter governmental information exchange
- Health-care networks (provider, insurer, laboratory, end-user)

# Chaincode Example

## Digital Rights Management for Music (DRM)

The DRM chaincode has a function ‘play()’ which:

- Reads an artwork
- Reads the royalty related to that artwork
- Increments a count to track royalty payments
- Writes the new count

# Chaincode Example

Digital Rights Management for Music (DRM)

DRM chaincode function ‘play()’:

```
async play(ctx, artWorkId) {
    const metadata = await ctx.stub.getState(artWorkId);
    let royaltyManagementAsset = await
    ctx.stub.getState(metadata.royaltyManagementId);
    royaltyManagementAsset.incrementPlayCount();
    await ctx.stub.putState(metadata.royaltyManagementId,
          royaltyManagementBuffer);
}
```

# Chaincode Example

Digital Rights Management for Music (DRM)

play(context, 04672033) generates this **read-write set**:

{"namespace":"drm","rwset":{
"reads":[
**{"key":"04672033","version":{"block_num":"5","tx_num":"8"}},**
**{"key":"554266330","version":{"block_num":"5","tx_num":"8"}}],**
"range_queries_info":[],
"writes":[
**{"key":"04672033","is_delete":false,"value":"{\"docType\":**
**\"royaltyManagement\",**
**\"perPlayRoyalty\":0.0031611628296938066,\"allRightHolder\":**
**[{\"ipiName\":\"4435023488O\",\"share\":0.724569263630407 1},**
**{\"ipiName\":\"28085045037\",\"share\":0.10356757729154009},**
**{\"ipiName\":\"88061101255\",\"share\":0.17186315907805283}],**
**\"playCount\":1}"}],**
"metadata_writes":[]},"collection_hashed_rwset":[]}

# Chaincode Example

Digital Rights Management for Music (DRM)

play(context, 04672033) generates this **read-write set**:

Read set with key and version

{"namespace":"drm","rwset":{
"reads":[
**{"key":"04672033","version":{"block_num":"5","tx_num":"8"}},**
**{"key":"554266330","version":{"block_num":"5","tx_num":"8"}}],**
"range_queries_info":[],
"writes":[
**{"key":"04672033","is_delete":false,"value":"{\"docType\":**
**\"royaltyManagement\",**
**\"perPlayRoyalty\":0.0031611628296938066,\"allRightHolder\":**
**[{\"ipiName\":\"4435023488O\",\"share\":0.724569263630407 1},**
**{\"ipiName\":\"28085045037\",\"share\":0.10356757729154009},**
**{\"ipiName\":\"88061101255\",\"share\":0.17186315907805283}],**
**\"playCount\":1}"}],**
"metadata_writes":[]},"collection_hashed_rwset":[]}

# Chaincode Example

Digital Rights Management for Music (DRM)

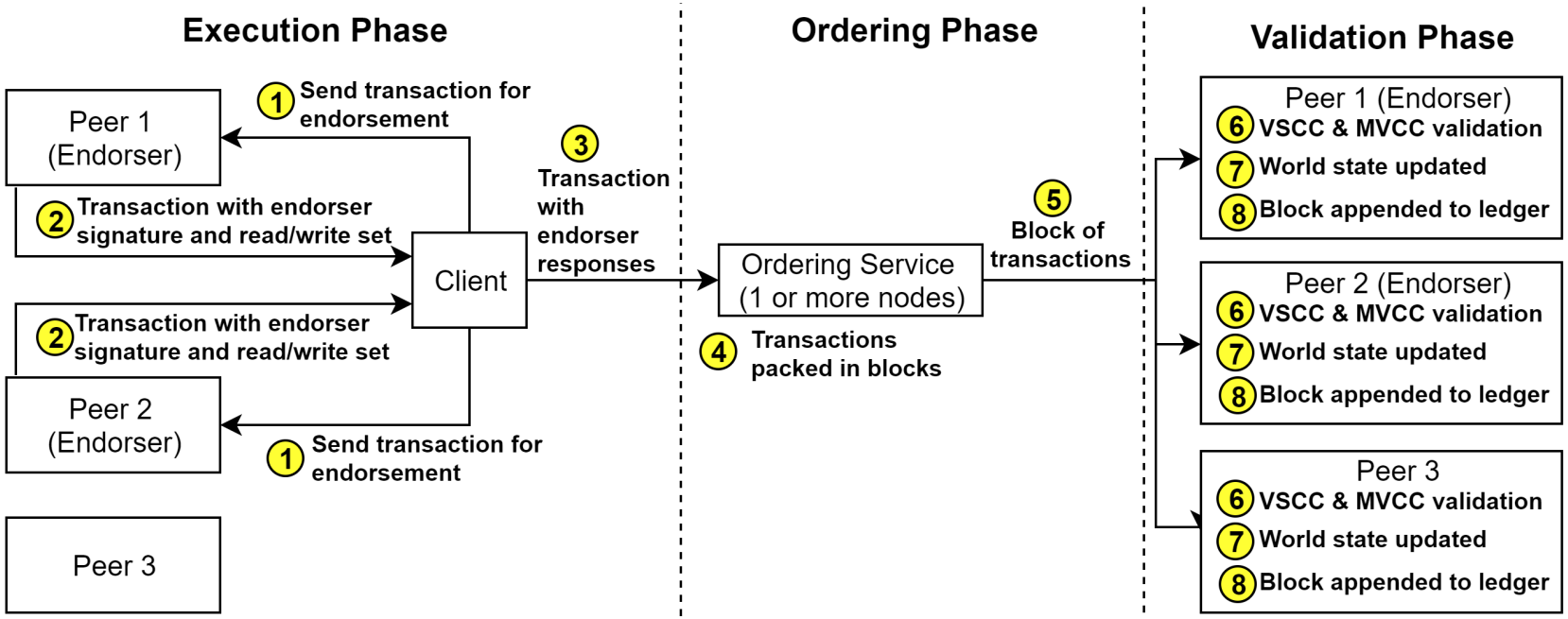
play(context, 04672033) generates this **read-write set**:


```
{"namespace":"drm","rwset":{
"reads":[
{"key":"04672033","version":{"block_num":"5","tx_num":"8"}},
{"key":"554266330","version":{"block_num":"5","tx_num":"8"}}],
"range_queries_info":[],
"writes":[
{"key":"04672033","is_delete":false,"value":"{\"docType\":
\"royaltyManagement\",
\"perPlayRoyalty\":0.0031611628296938066,\"allRightHolder\":
[{\"ipiName\":\"4435023488O\",\"share\":0.724569263630407 1},
{\"ipiName\":\"28085045037\",\"share\":0.10356757729154009},
{\"ipiName\":\"88061101255\",\"share\":0.17186315907805283}],
\"playCount\":1}"}],
"metadata_writes":[]},"collection_hashed_rwset":[]}
```


Read set with key and version

Write set with key and value

# Transaction Flow in Fabric




E-O-V Model

# Blockchain Applications

1.0, 2.0, 3.0 GENERATIONS

IMPACT

# Blockchain 1.0: Currency


Bitcoin cryptocurrency (2008)

# Blockchain 2.0: Decentralized Apps (DApps)

ETHEREUM

DApps are applications built on blockchain platforms using smart contracts (e.g. Ethereum)

# Blockchain 2.0: Decentralized Apps (DApps)

ETHEREUM

DApps are applications built on blockchain platforms using smart contracts (e.g. Ethereum)

# Blockchain 2.0: Decentralized Apps (DApps)

ETHEREUM

DApps are applications built on blockchain platforms using smart contracts (e.g. Ethereum)


Token Distribution
Crowdfunding

# Blockchain 2.0: Decentralized Apps (DApps)

ETHEREUM

DApps are applications built on blockchain platforms using smart contracts (e.g. Ethereum)



Crowdfunding


Charity donation payment

# Blockchain 2.0: Decentralized Apps (DApps)

# Blockchain 2.0: Decentralized Apps (DApps)

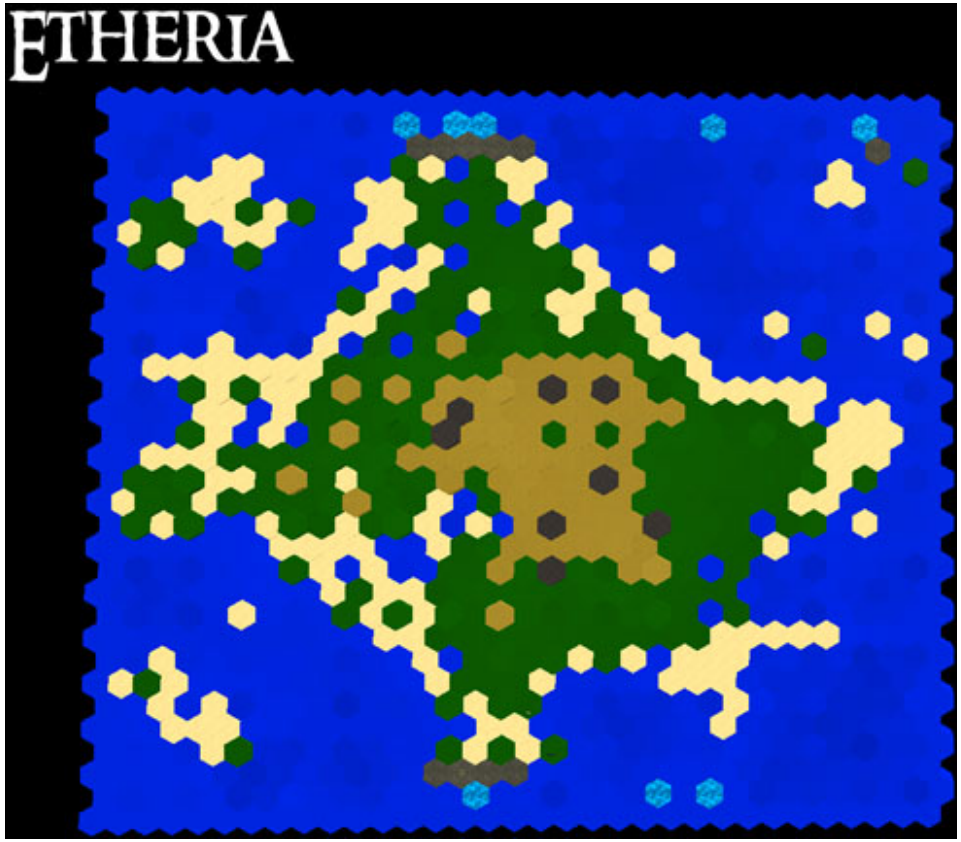



Forecast market (e.g. betting, insurance)

# Blockchain 2.0: Decentralized Apps (DApps)

GNOSIS

Forecast market (e.g. betting, insurance)




Decentralized virtual world

# Blockchain 3.0: Pervasive Apps

Applications involve entire industries, **public sector**, and IoT.

# Blockchain 3.0: Pervasive Apps

everledger

Diamonds Provenance

Applications involve entire industries, **public sector**, and IoT.

# Blockchain 3.0: Pervasive Apps

everledger

Diamonds Provenance

FACTOM

Land Registry in Honduras

Applications involve entire industries, **public sector**, and IoT.

# Blockchain 3.0: Pervasive Apps

everledger

Diamonds Provenance

FACTOM

Land Registry in Honduras

BlockchainHealth

Electronic Health Records

Applications involve entire industries, **public sector**, and IoT.

# Blockchain 3.0: Pervasive Apps

everledger

Diamonds Provenance

FACTOM

Land Registry in Honduras

BlockchainHealth

Electronic Health Records

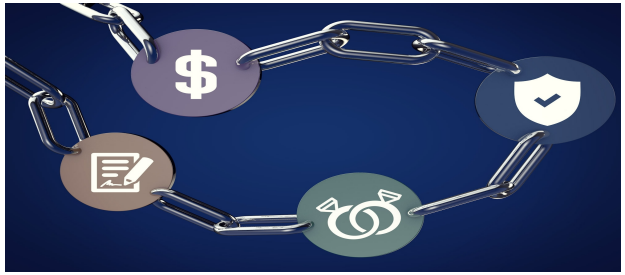
Applications involve entire industries, **public sector**, and IoT.

VOTEWATCHER

Transparent Voting System

# Why Study Blockchains?

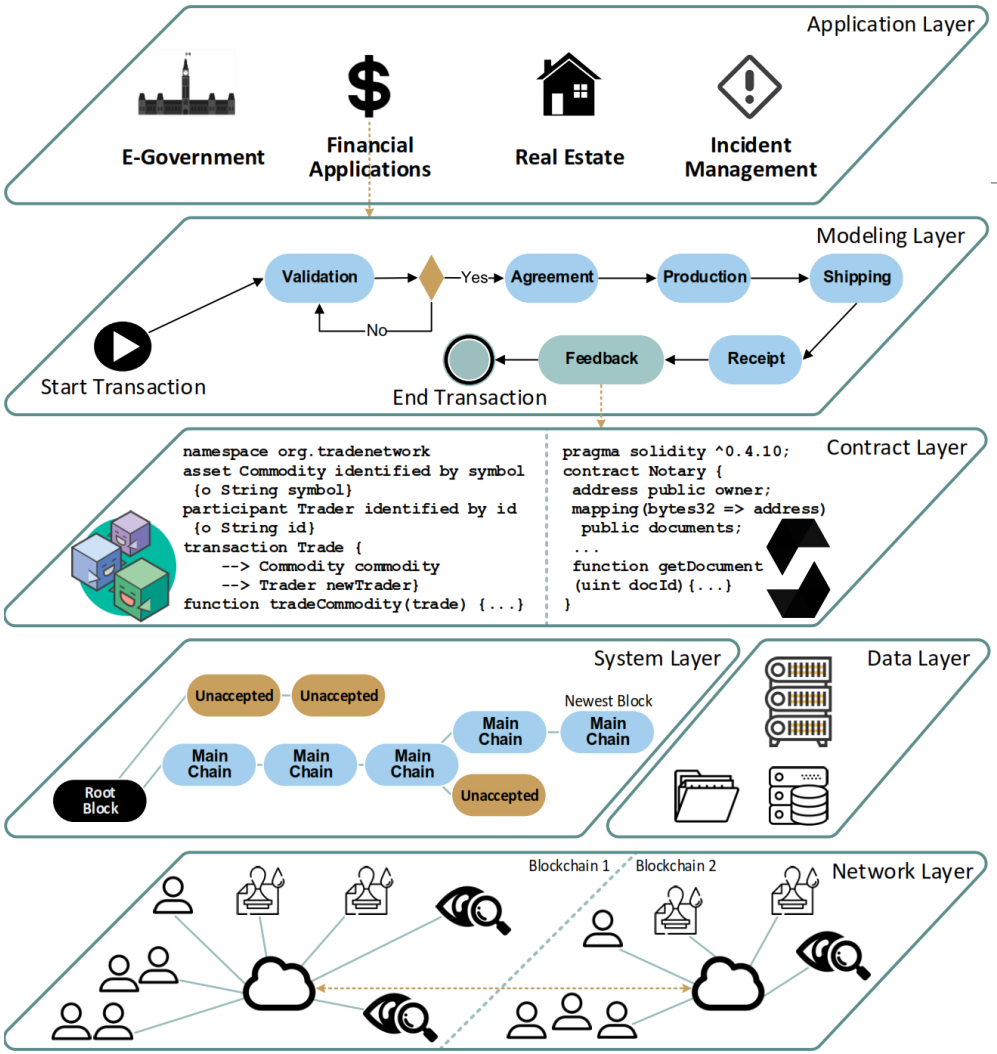
Drivers
- Avoid middlemen
- Provide transparency, audit trail
- Eliminate friction during conflicts (non-repudiation)



Research challenges for 1.0:
- Identify theoretical **security flaws**
- **Sustainability** of legacy systems

Research challenges for 2.0:
- **Verify** smart contracts
- Create generic middleware **services**

Research challenges for 3.0:
- Develop **scalable and fast** systems
- Guarantee data **privacy**
- Verify **correctness** of data entry points (CPS interface barrier)




# Conclusions

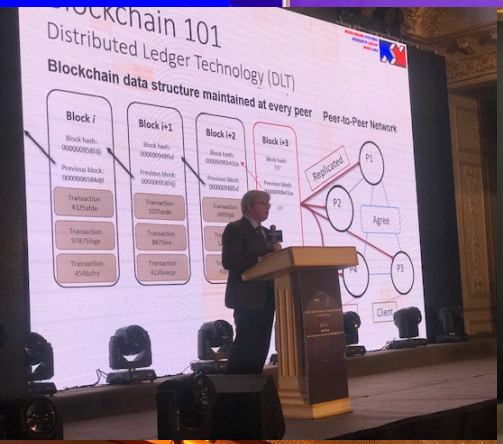
- Blockchains provide *decentralized storage and code execution,* and can be used to combat fraud, avoid redundancy, and provide transparency.
- Blockchains rely on *cryptography* and massive replication using a robust consensus mechanism.
- Blockchains are useful for a wide variety of applications, ranging from cryptocurrency (*1.0*) to health (*3.0*).
- Research directions exist *across the six layers* for all kinds of applications (from 1.0 to 3.0).

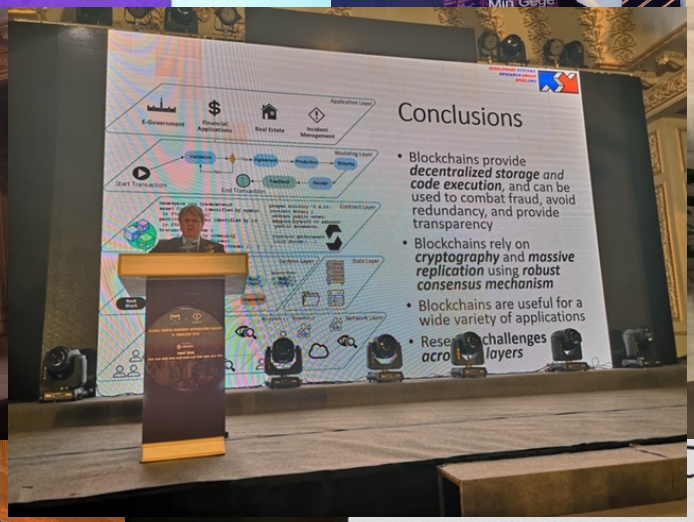

链接世界 区动未来
he future
WIC 第三届世界智能大会
THE 3rd WORLD INTELLIGENCE CONGRESS
全球区块链科技创新峰会
Global BlockChain Technology Innovation Summit


Hans-Arno Ja
Min Gege



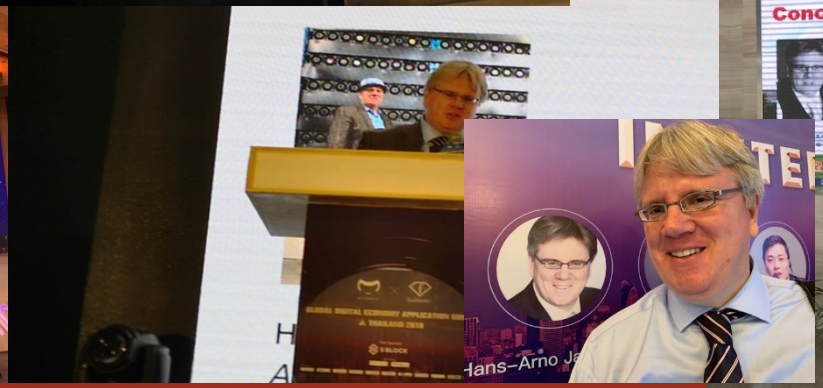

Blockchain Characteristics
A.k.a. Distributed Ledger
链接世界 区动未来
Link the world Drive the future


Blockchain 101
Distributed Ledger Technology (DLT)
Blockchain data structure maintained at every peer
Peer-to-Peer Network
Block i
Block i+1
Block i+2
Block i+3
Replicated
Agree
Client


Conclusions
Blockchains provide decentralized storage and code execution, and can be used to combat fraud, avoid redundancy, and provide transparency
Blockchains rely on cryptography and massive replication using robust consensus mechanism
Blockchains are useful for a wide variety of applications


Deconstructing Blockchains:
Concepts, Applications, and Systems
Prof. Hans-Arno Jacobsen
Univ. of Toronto


Blockchain Characteristics
A.k.a. Distributed Ledger
链接世界 区动未来
Link the world Drive the future

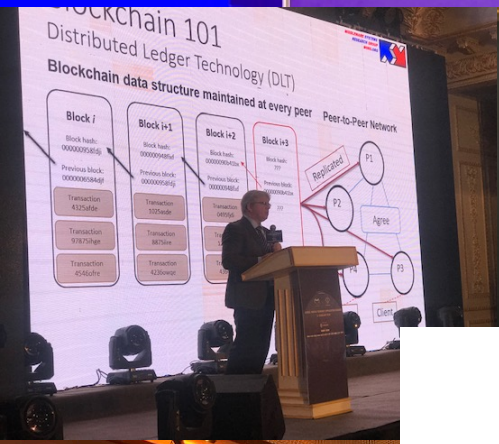


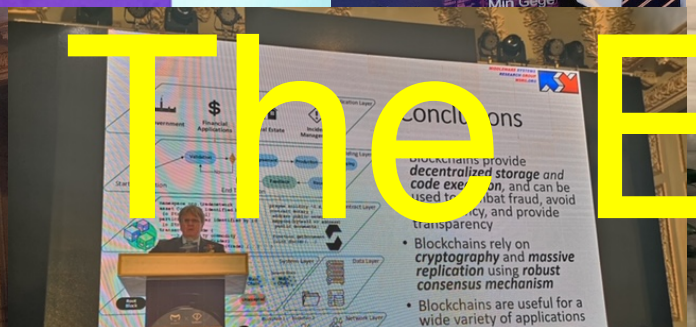

Hans-Arno Ja


Deconstructing Blockchains:
Concepts, Applications, and Systems
Prof. Hans-Arno Jacobsen
Univ. of Toronto


MIDDLEWARE SYSTEMS
RESEARCH GROUP
MSRG.ORG
WIC 第三届世界智能大会
THE 3rd WORLD INTELLIGENCE CONGRESS
全球区块链科技创新峰会
Global BlockChain Technology Innovation Summit
Blockchain Characteristics
A.k.a. Distributed Ledger
链接世界 区动未来
Link the world Drive the future
Blockchain 101
Distributed Ledger Technology (DLT)
Blockchain data structure maintained at every peer
Peer-to-Peer Network
Deconstructing Blockchains: Concepts, Applications, and Systems
Prof. Hans-Arno Jacobsen
Univ. of Toronto
The End!






# The End!

**Please do not forget to provide your feedback via the course evaluation.**