Session 01-02

Basic Terminology

Blockchain Course

MSc. Phan The Duy

UIT Information Security Lab

Course Outline

- Session 01: **Blockchain and Basic Terminology (p1)**
- Session 02: **Blockchain and Basic Terminology (p2)**
- Session 03: **dApp & How to become Blockchain Programmer**
  - Everything you need to know about Blockchain Dev.
  - What is dApp?
  - Ethereum and EVM (Ethereum Virtual Machine).
  - Smart Contract and Solidity.
  - Build “Hello World” dApp.
- Session 04: **How to build Blockchain Project with Ethereum Platform**
  - Ethereum Test net: (Ropsten, Rinkerby, Kovan, Local, Main Test net…)
  - How to build Blockchain Project with Truffle framework.
  - Frontend dApp: React, Nodejs, Web3js, …
    - Session 05: **Advanced Blockchain Dev**
      - IPFS and Oraclize.
      - ICO and Token
      - **Final Test**

UIT Information Security Lab

Currency trend…

UITiS
UiT Information Security Lab

Bitcoin – 2nd most-searched term in 2017

- **Cryptocurrency**, decentralized digital currency
- No issue organization, no central bank or single administrator.
- **Transactions of bitcoins** is recorded in a public distributed ledger - **Blockchain**

Supply limit: **21,000,000** bitcoins

Block time: 10 minutes

1 Bitcoin equals

188,999,029.34 Vietnamese dong

1 Bitcoin equals

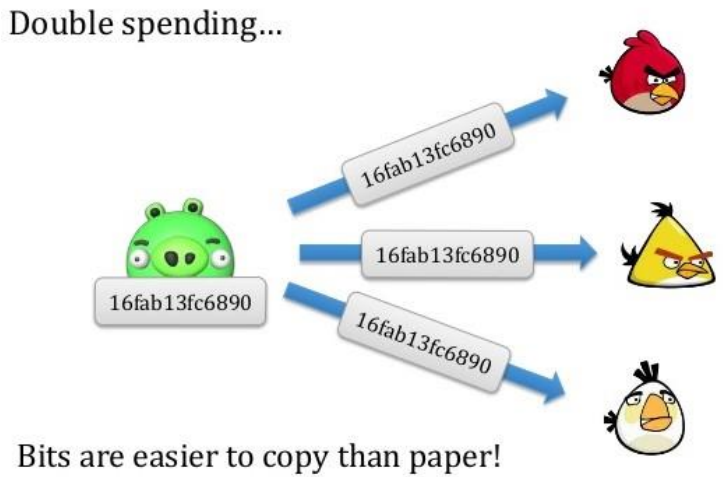
8,157.85 United States Dollar

UIT Information Security Lab

You are **Alice**, I am **Bob**.

- I have a Thor toy. You have none.
- I give the toy to you.
- Now you have a toy. I have none.

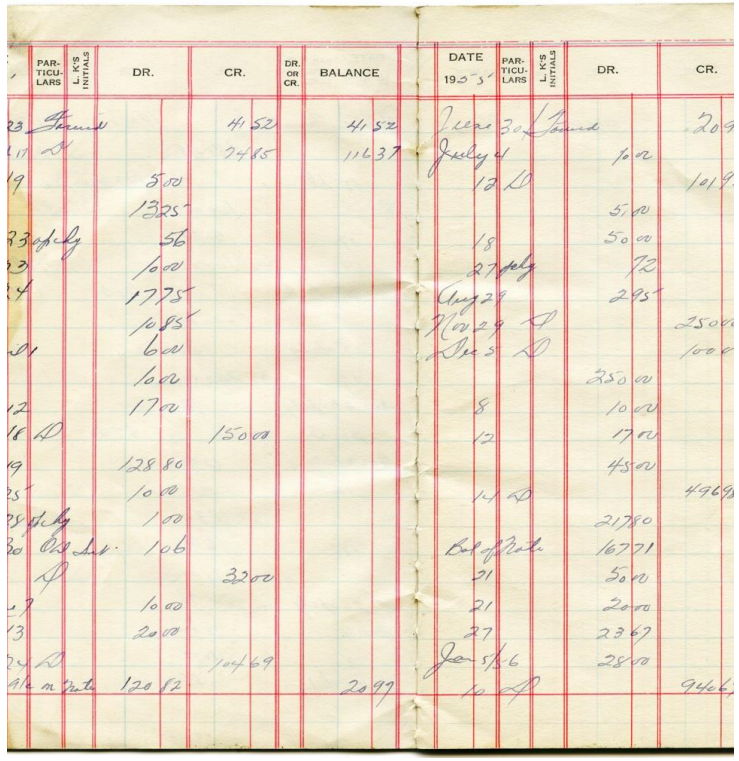
UITiS
UIT Information Security Lab



Double-spending problem

- Now suppose I have a **digital toy.**
- I give you my digital toy.
- **Wait!!!**
- How do you know I didn't give the digital toy as an email attachment to **Charlie** first? Or I saved a million copies on my laptop and gave it to everyone on the internet?
- So, sending digital toys ***is not the same as*** giving physical toys.

UITiS
UIT Information Security Lab

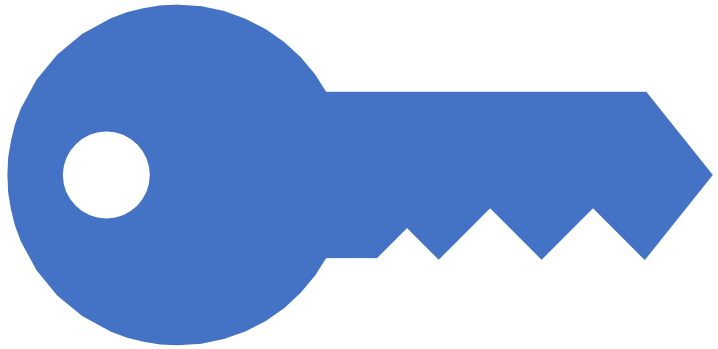


But isn’t the solution very simple?

Ledger: An account book, where all the transactions are recorded.

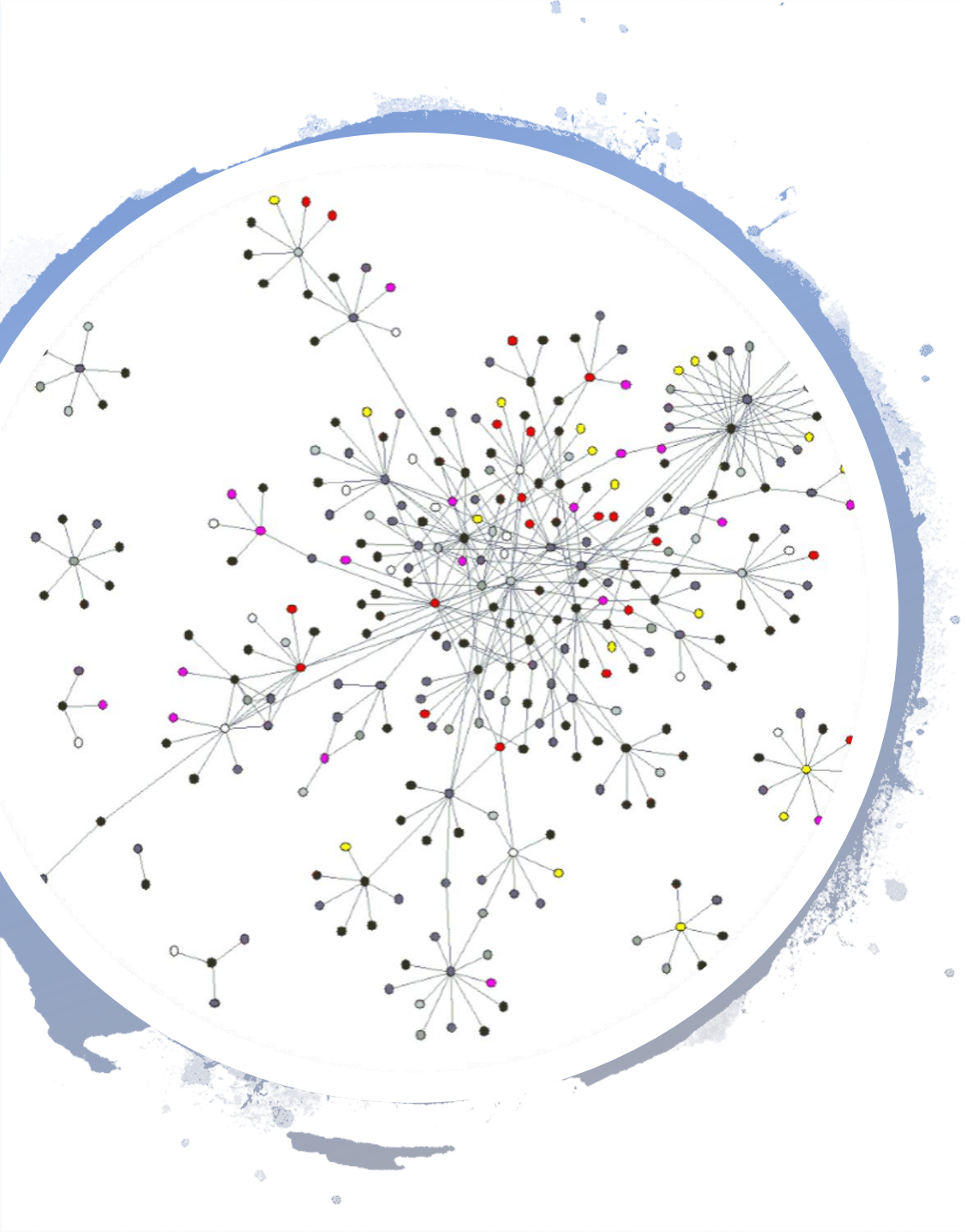
So, there will be a ledger where all the transactions of digital toys will be recorded. Someone, say **Danny** will be in charge of it.

UITiS
UIT Information Security Lab



Problem Solved, right?

- **No.**
- Because how do we know if **Danny** is not cheating and adding digital toys to his and his girlfriend's account anytime he wants?
- Also, why do we need to involve **Danny** in transactions that concern only you and me?



Enters Blockchain!

- What if **everyone had a copy of** the **accounting book** on their computer and all the transactions that ever happen are recorded there.
- Now **Danny** can't add toys that he doesn't have to his account. Because then it wouldn't match with everyone else's book.
- Everyone who has **a copy of the ledger** on their system and maintains it by validating the transactions get **digital toys as rewards**. **This is blockchain.**
- The ledger is visible to everyone. Now we don't rely on a third person (**Danny**) to maintain our balances for us. We don't need to worry about the double spending problem.

UITiS
UIT Information Security Lab

Four components in secure communication

UITiS
UIT Information Security Lab

What do we want to secure?

Authentication (Who am I talking to?)

- Identification and assurance of the origin of information

Confidentiality (Is my data hidden?)

- Concealment of information

Integrity (Has my data been modified?)

- Prevent improper and unauthorized changes

Availability (Can I use the resources?)

- The ability to use the information or resource desired

UITiS
UIT Information Security Lab

From the perspective of **BitCoin**

Authentication	• Am I paying the right person? Not some other impersonator?
Integrity	• Is the coin double-spent? • Can an attacker reverse or change transations?
Availability	• Can I make a transaction anytime I want?
Confidentiality	• Not very relevant. But privacy is important.

From the perspective of BitCoin

Authentication → Public Key Crypto: Digital Signatures

- Am I paying the right person? Not some other impersonator?

Integrity → Digital Signatures and Cryptographic Hash

- Is the coin double-spent?
- Can an attacker reverse or change transations?

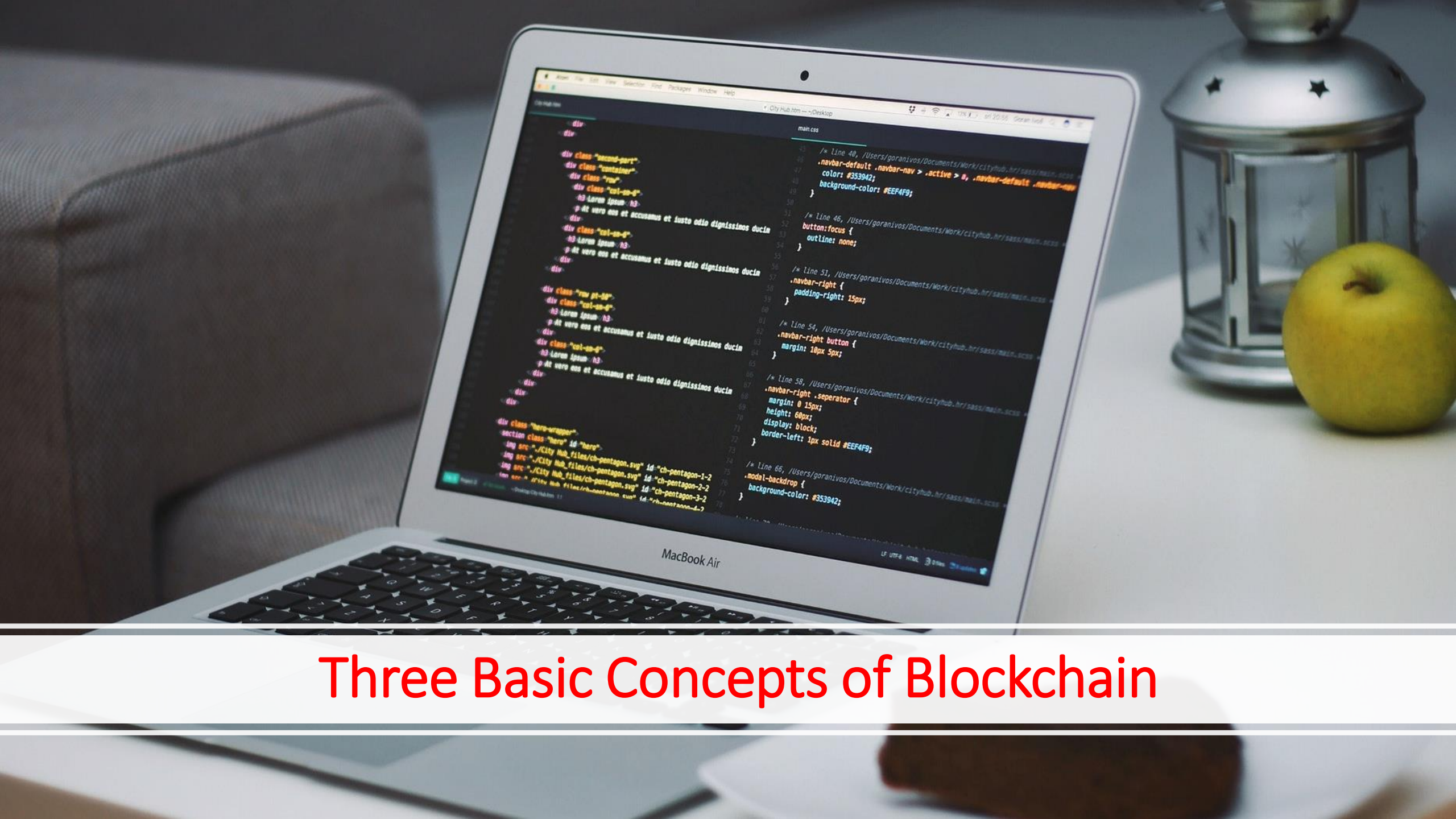
Availability→ P2P

- Can I make a transaction anytime I want?

Confidentiality

- Not very relevant. But privacy is important.

UITiS
UIT Information Security Lab

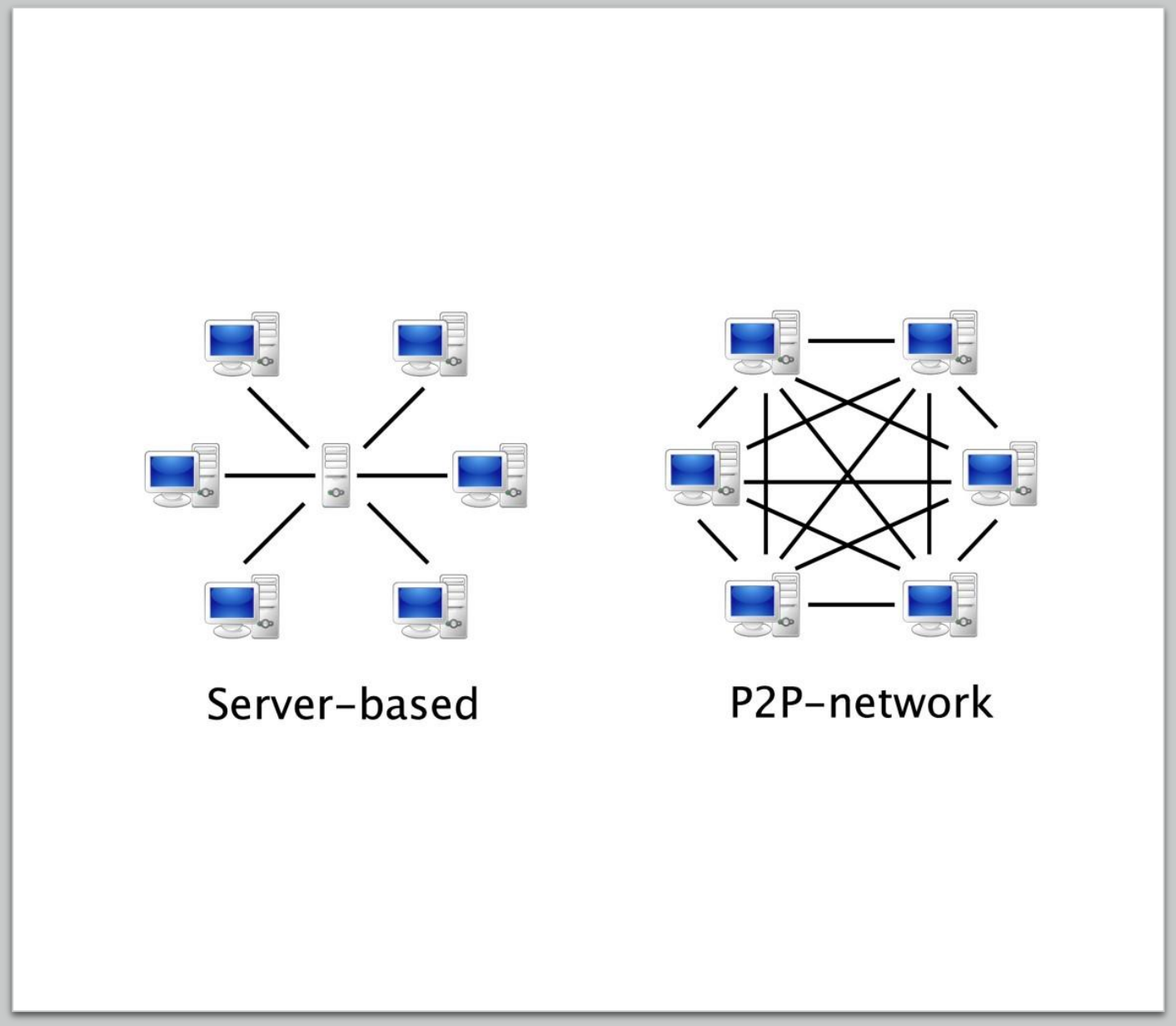


Three Basic Concepts of Blockchain

(1) Peer to Peer Networking?

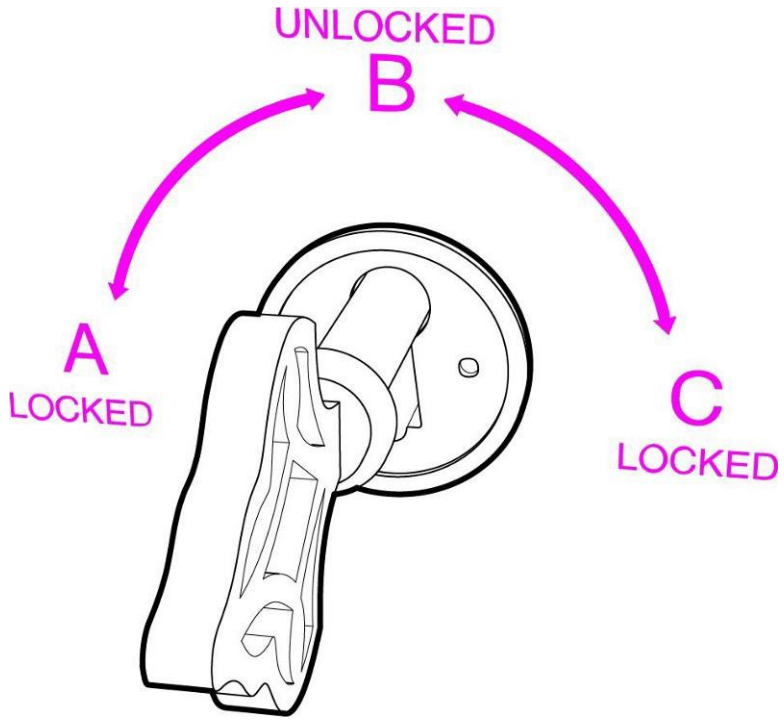
P2P:

- No server role
- Direct connection between nodes



UIT Information Security Lab

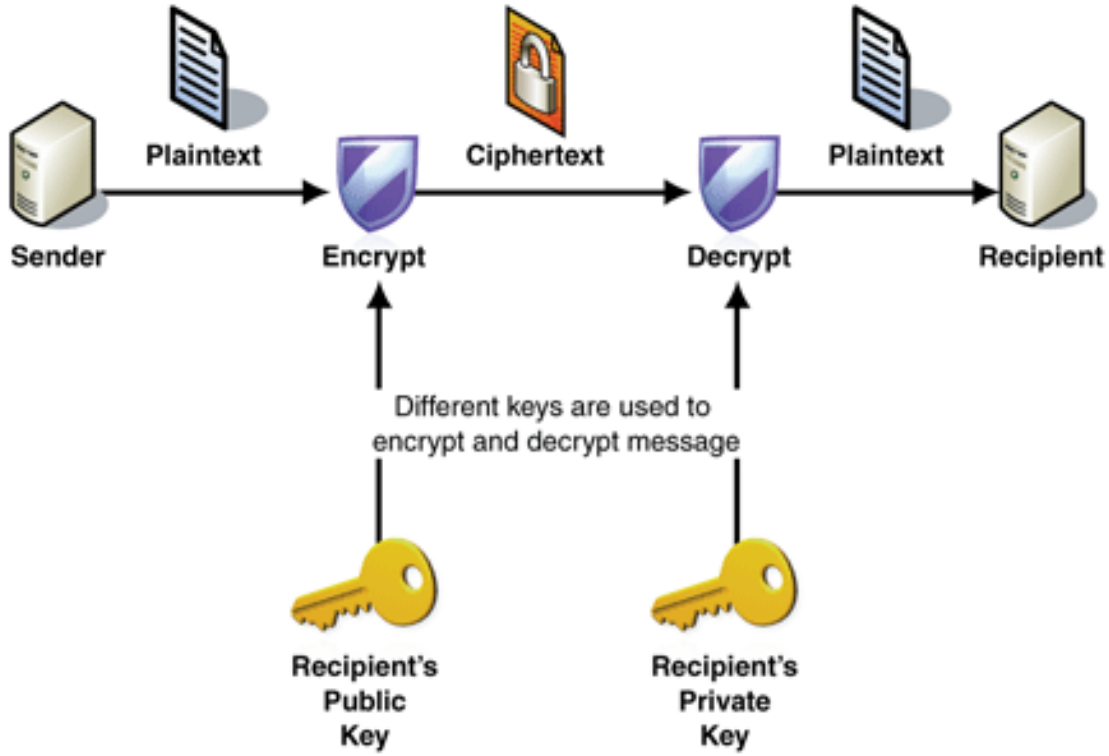
(2) Public Key Cryptography



- **Public key**
- **Private key**
- **Digital signature**

Public Key Crypto: Encryption

- Key pair: public key and private key



Public Key Crypto Example: RSA

- RSA Keygen
  - Choose two distinct prime numbers p and q. (Let $n = pq$.)
  - Compute $\phi(n) = \phi(p)\phi(q) = (p - 1)(q - 1)$, where ϕ is Euler's totient function.
    - $\phi(n)$*:* the number of integers k in the range $1 \leq k \leq n$ for which gcd$(n, k) = 1$.
  - Choose a coprime of $\phi(n)$, e, such that $1 < e < \phi(n)$, i.e., gcd$(e, \phi(n)) = 1$
  - Solve for d where $d \cdot e \equiv 1 \pmod{\phi(n)}$
- Public key (n, e); Private key (n, d)

Public Key Crypto Example: RSA

- Public key (*n, e*); Private key (*n, d*)

  Encryption: Compute ciphertext C = m^e (**mod** *N*). (public key)

  Decryption: Recover m = C^d (**mod** *N*). (private key)

$$m^{ed} = m^{(ed-1)}m = m^{h(p-1)(q-1)}m = (m^{p-1})^{h(q-1)}m \equiv 1^{h(q-1)}m \equiv m \pmod{p},$$

$$ed \equiv 1 \pmod{(p-1)(q-1)}.$$

Fermat's Little Theorem: for any number a and any prime p

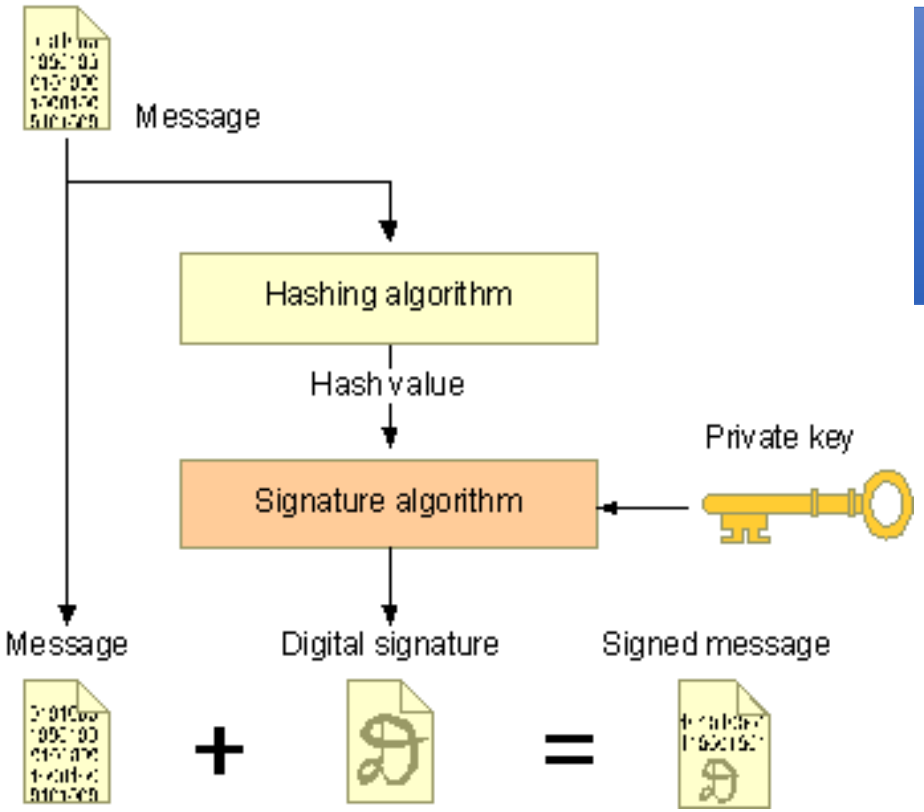
$$a^p \equiv a \pmod{p}. \quad a^{p-1} \equiv 1 \pmod{p}.$$

- Why does this work?
  - Factorization is hard; given n hard to infer p and q.
  - Computing *m* is hard given the public key (*n*, *e*) and a ciphertext $C \equiv m^e$ (**mod** *N*).

UITiS
UIT Information Security Lab

Public Key Crypto: Digital Signature

- First, create a message digest using a cryptographic hash
- Then, encrypt the message digest with your private key



Authentication

Integrity

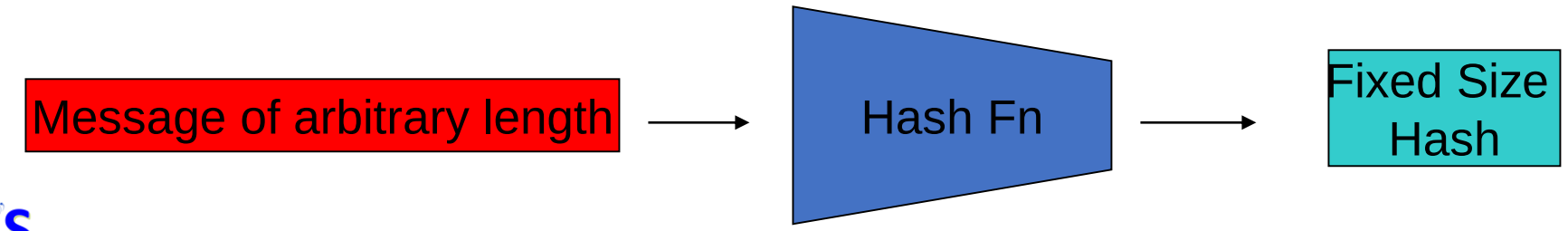
Non-repudiation

UITIS
UIT Information Security Lab

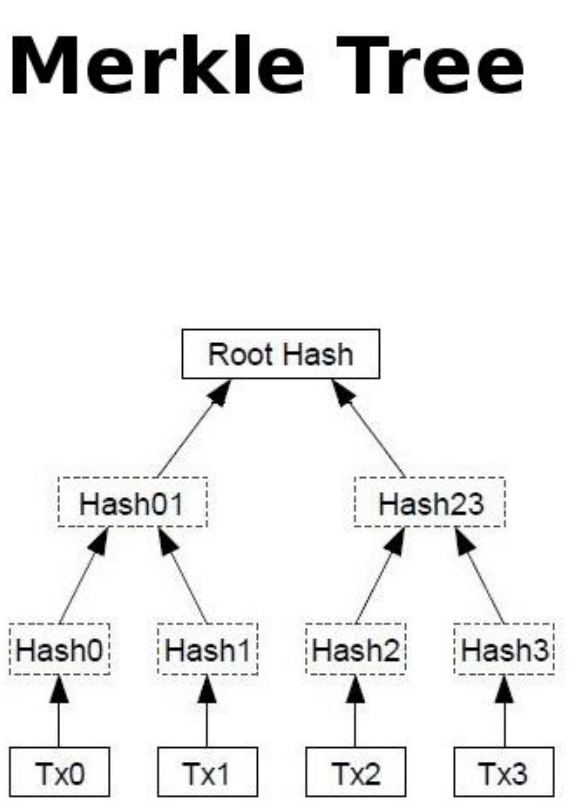
(3) Cryptographic Hash Function

It is a function which takes a message as an input and returns a fixed length alphanumeric string, called '**hash**'.

- Consistent: hash(X) always yields same result
- One-way: given Y, hard to find X s.t. **hash(X) = Y**
- Collision resistant: given hash(W) = Z, hard to find X such that hash(X) = Z



Merkle Tree



UIT Information Security Lab

Merkle Tree: How it works?

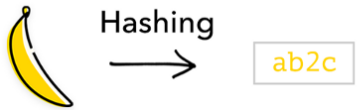
101

MERKLE TREE

Merkle Trees allow us to map a large volume of data and easily identify wherechanges in the data occur.

Q1:

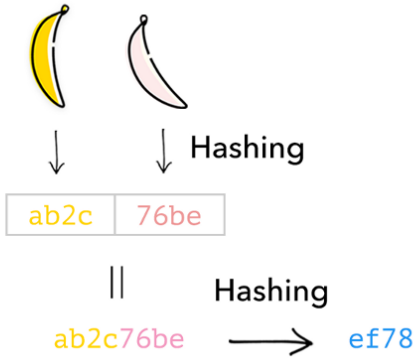
How to encode a banana ?



Representing a small piece of information

Q2:

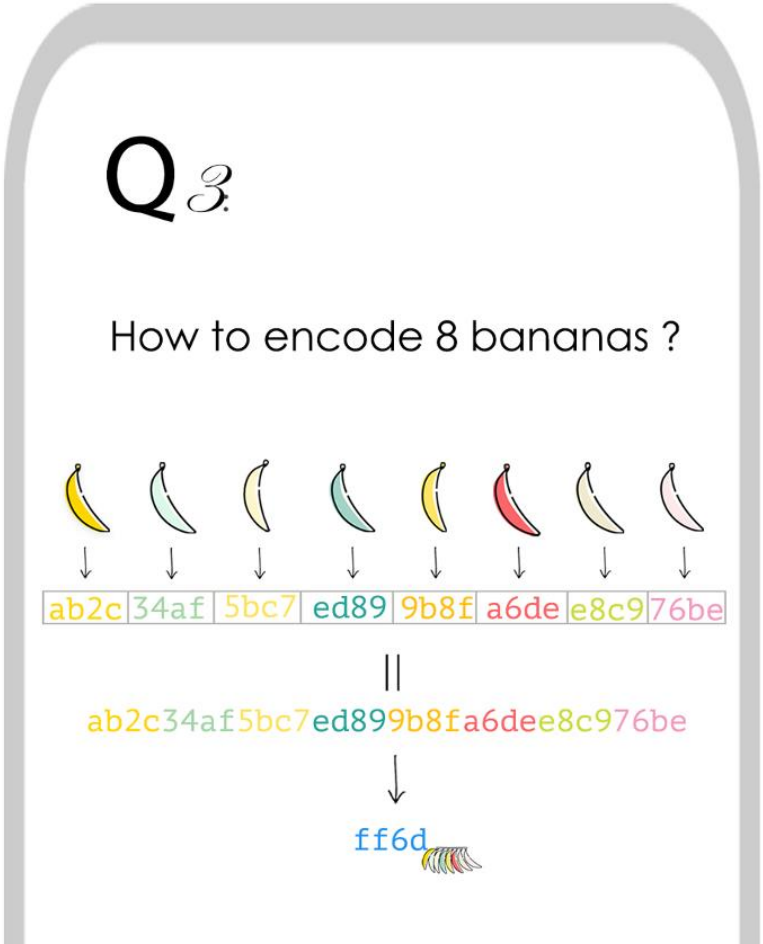
How to encode 2 bananas ?



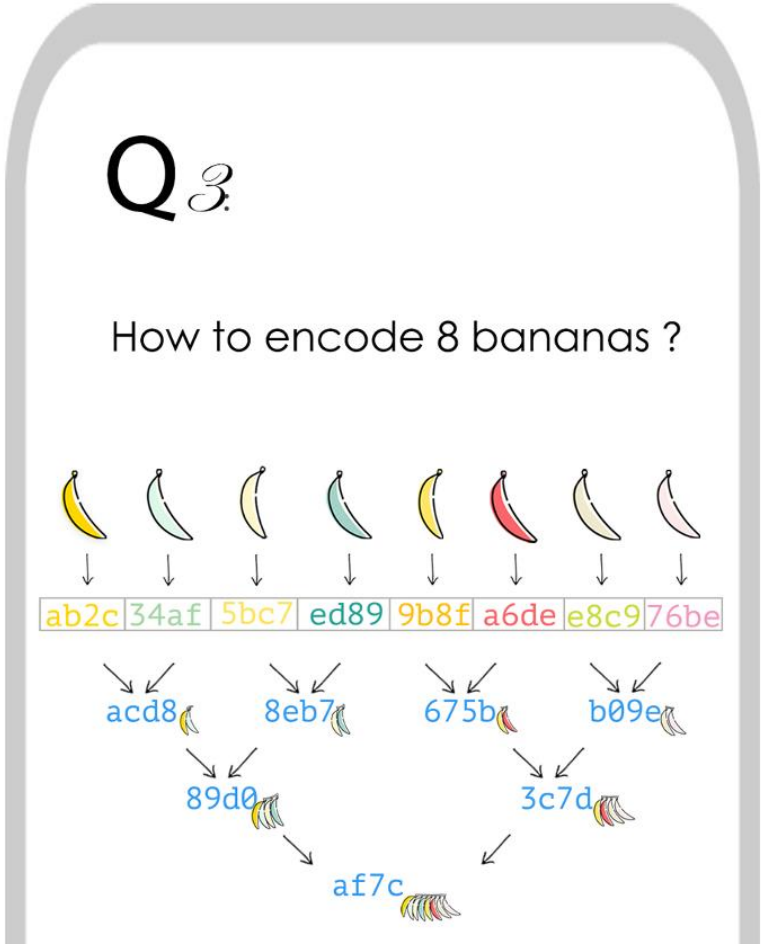
UITiS
UIT Information Security Lab

Merkle Tree: How it works?

Naïve Approach

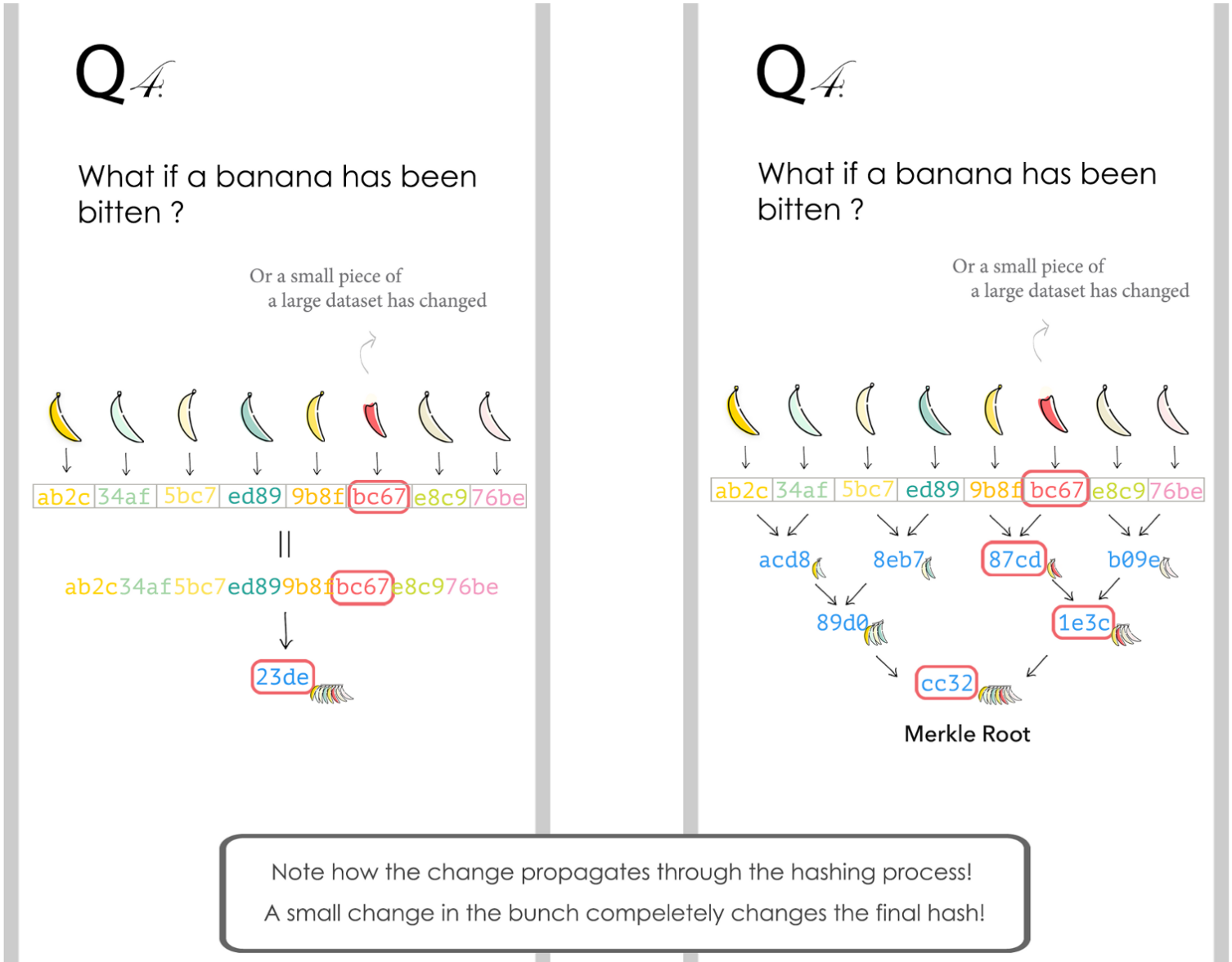


Merkle Tree



UIT Information Security Lab

Merkle Tree: How it works?



Note how the change propagates through the hashing process!

A small change in the bunch compeletely changes the final hash!

UITIS
UIT Information Security Lab

Merkle Tree: How it works?

Q5.

How to verify that (data chunk) is part of the bunch represented by the Final Hash `ff6d` ?

There is no way without the entire bunch (dataset)

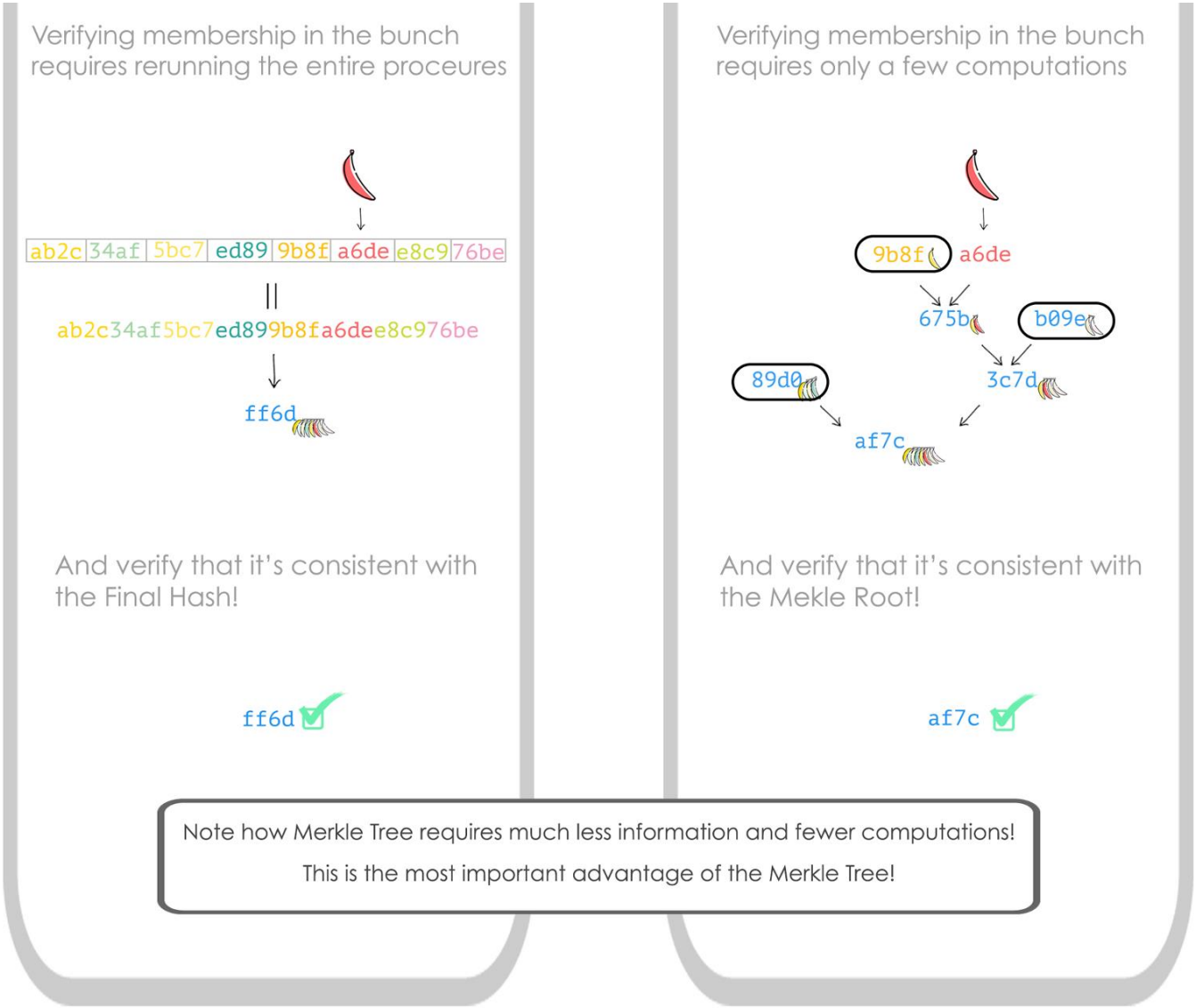
Q5.

How to verify that (data chunk) is part of the bunch represented by the Merkle Root `af7c` ?

We only need the banana and a handful of hashes!

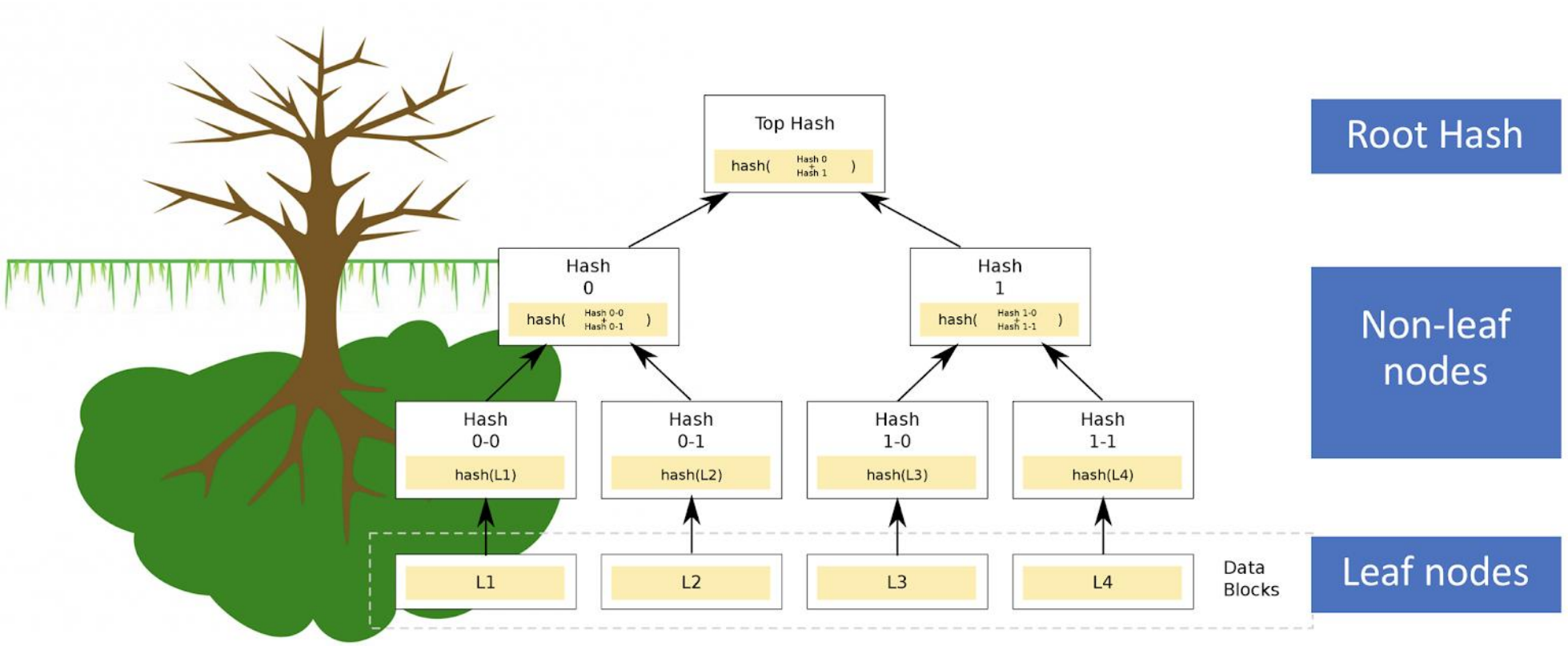
UITiS
UIT Information Security Lab

Merkle Tree: How it works?

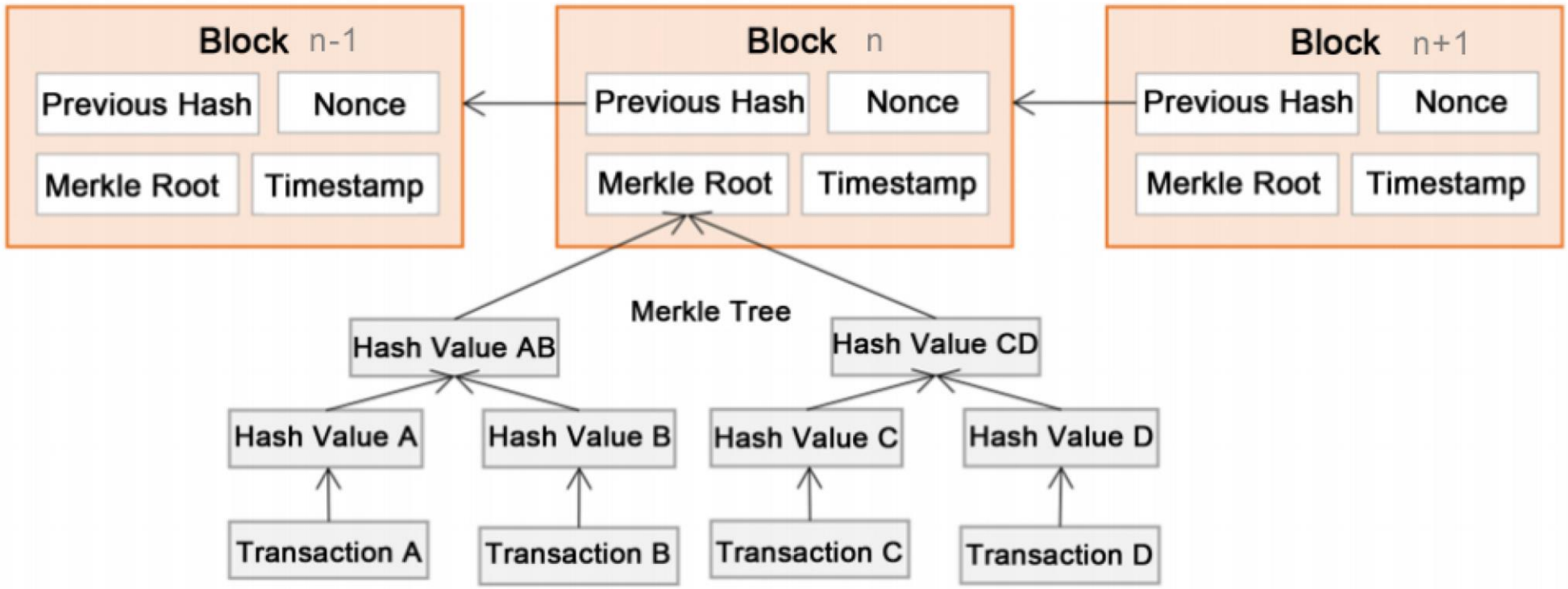


UIT Information Security Lab

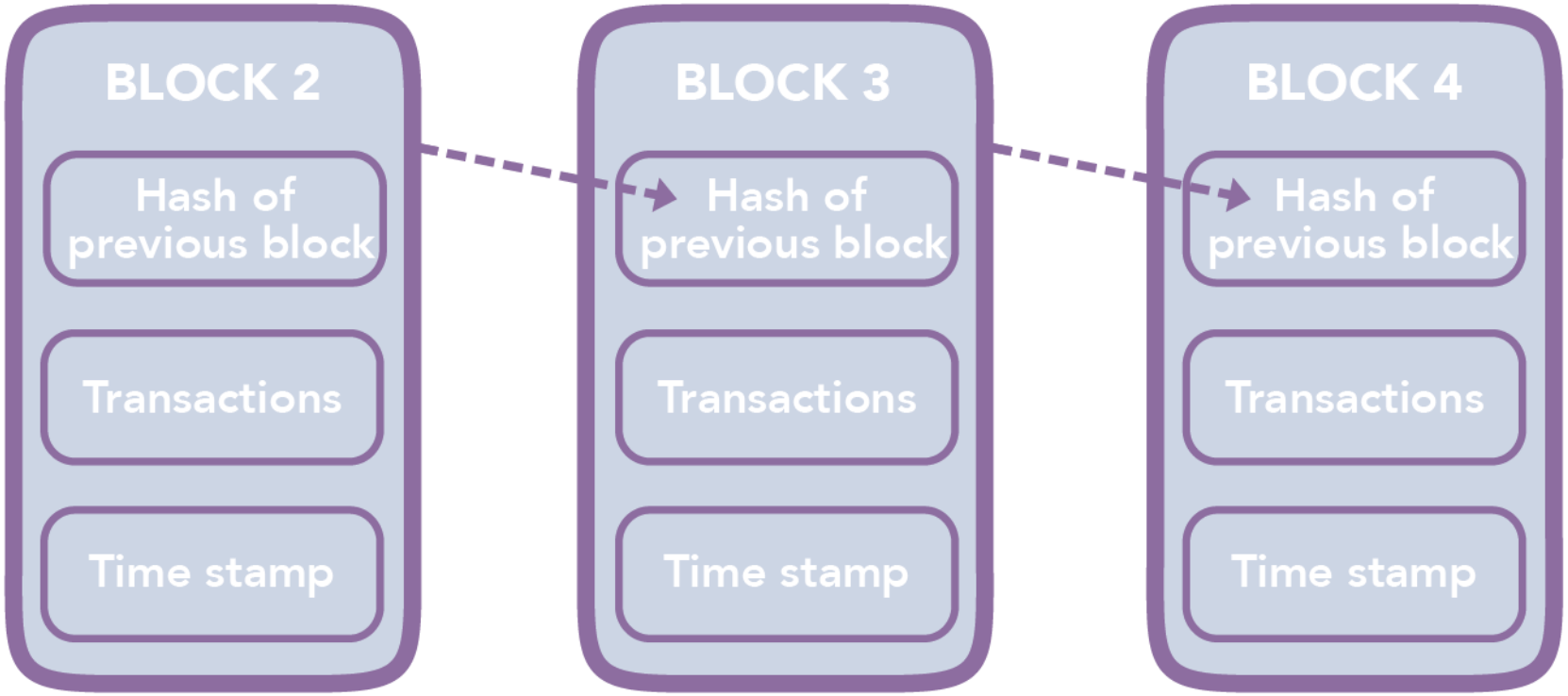
Merkle Tree

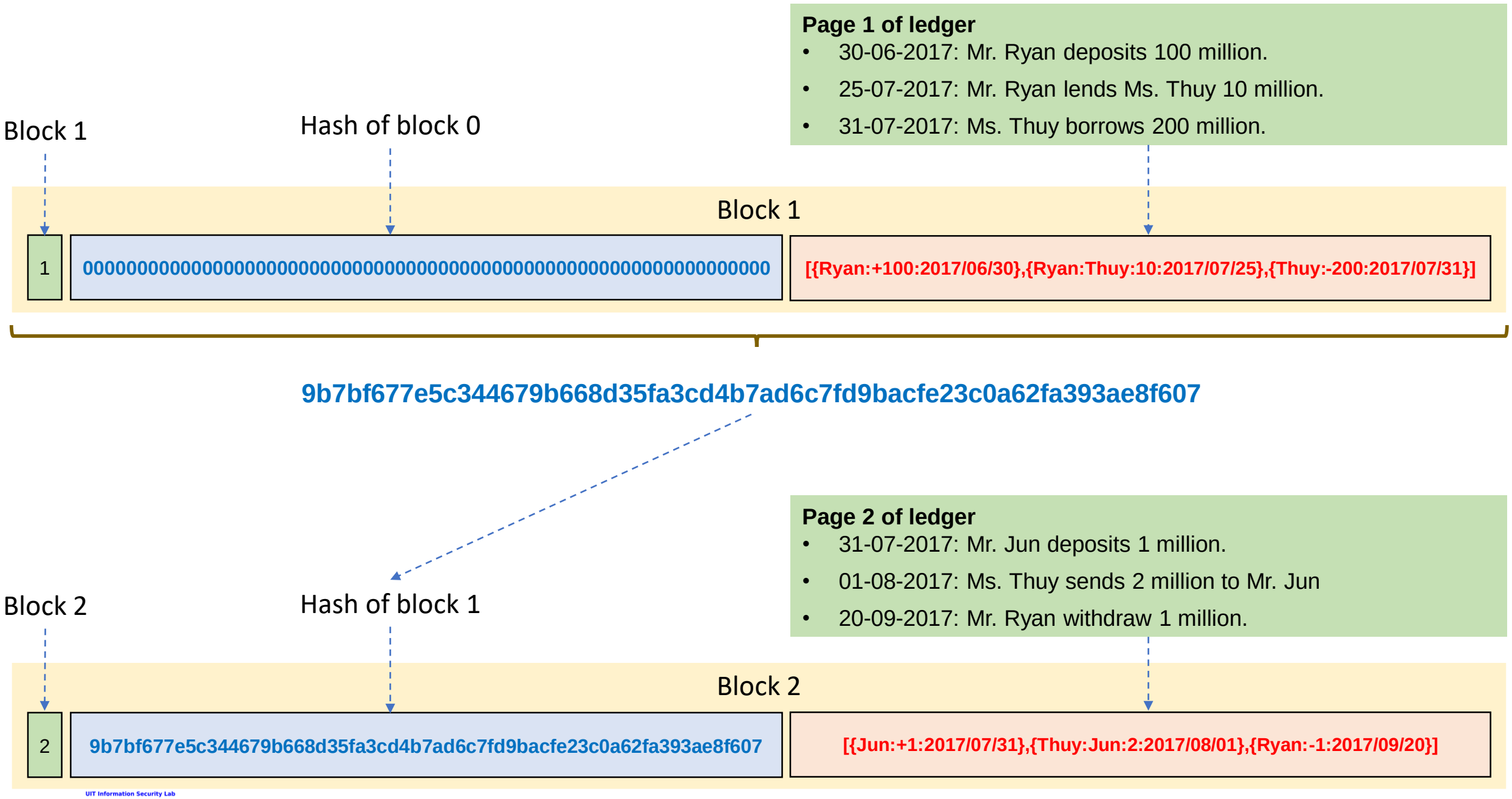

Top Hash
hash(Hash 0 + Hash 1)
Hash 0
hash(Hash 0-0 + Hash 0-1)
Hash 1
hash(Hash 1-0 + Hash 1-1)
Hash 0-0
hash(L1)
Hash 0-1
hash(L2)
Hash 1-0
hash(L3)
Hash 1-1
hash(L4)
L1
L2
L3
L4
Data Blocks
Root Hash
Non-leaf nodes
Leaf nodes

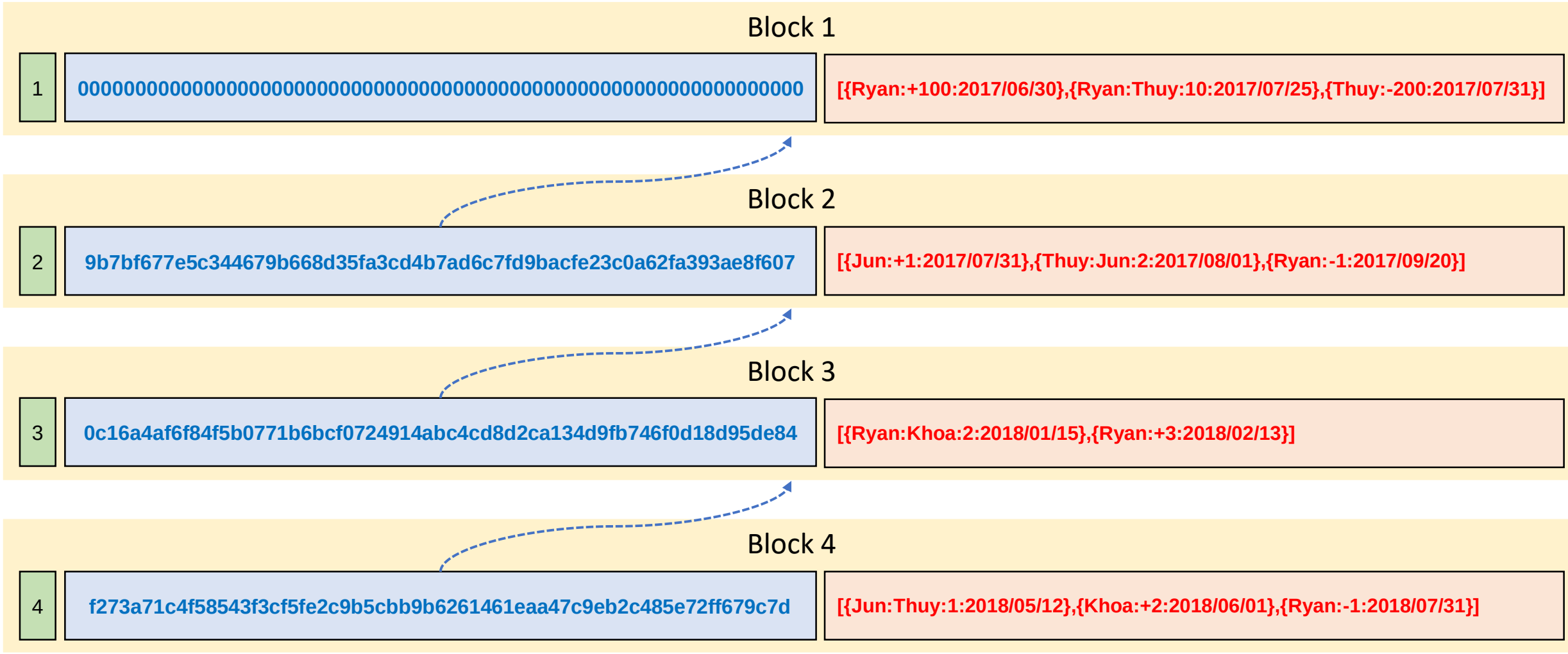
Merkle Tree in Blockchain: example



Chain of blocks in Blockchain: Linked List Type




Page 1 of ledger
• 30-06-2017: Mr. Ryan deposits 100 million.
• 25-07-2017: Mr. Ryan lends Ms. Thuy 10 million.
• 31-07-2017: Ms. Thuy borrows 200 million.
Block 1
Hash of block 0
Block 1
1
0000000000000000000000000000000000000000000000000000000000000000
[{Ryan:+100:2017/06/30},{Ryan:Thuy:10:2017/07/25},{Thuy:-200:2017/07/31}]
9b7bf677e5c344679b668d35fa3cd4b7ad6c7fd9bacfe23c0a62fa393ae8f607
Page 2 of ledger
• 31-07-2017: Mr. Jun deposits 1 million.
• 01-08-2017: Ms. Thuy sends 2 million to Mr. Jun
• 20-09-2017: Mr. Ryan withdraw 1 million.
Block 2
Hash of block 1
Block 2
2
9b7bf677e5c344679b668d35fa3cd4b7ad6c7fd9bacfe23c0a62fa393ae8f607
[{Jun:+1:2017/07/31},{Thuy:Jun:2:2017/08/01},{Ryan:-1:2017/09/20}]
UIT Information Security Lab



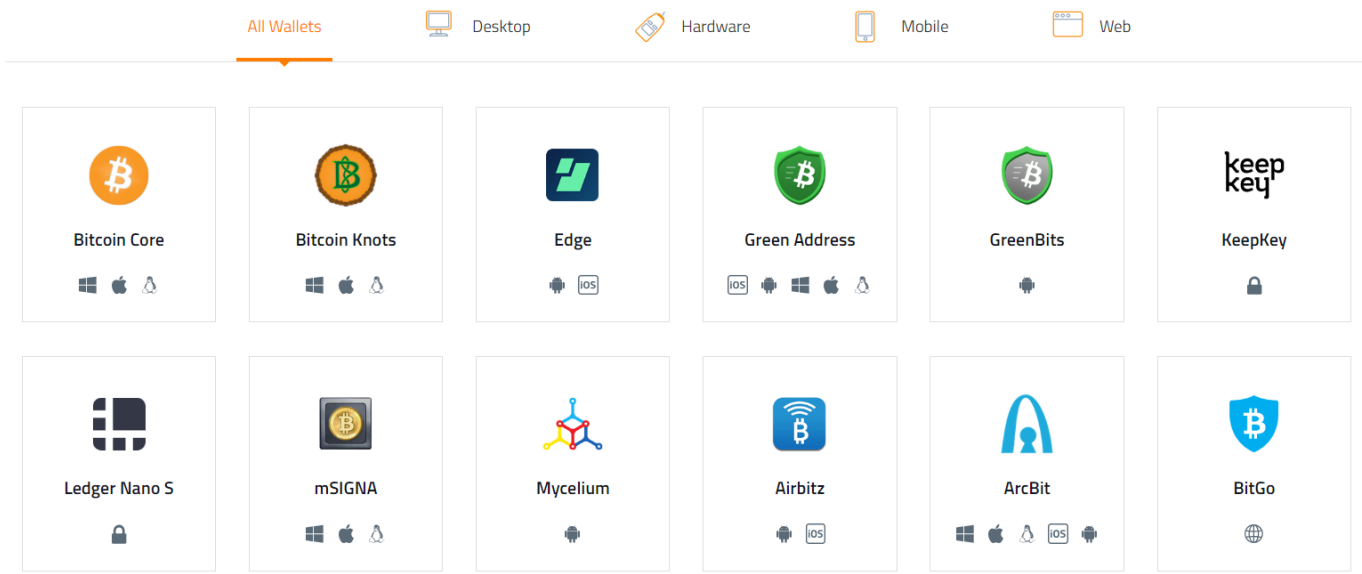
Update data in block 2: ***June deposits 1 million*** → **June deposits 150 million.** Because:

- Block 3 contains hash (block 2) → need to recompute and update block 3
- Block 4 contains hash (block 3) → need to recompute and update block 4
- …..
- Block 1000 contains hash (block 999) → need to recompute and update block 1000……
- ….

UIT Information Security Lab

How to use Bitcoin?

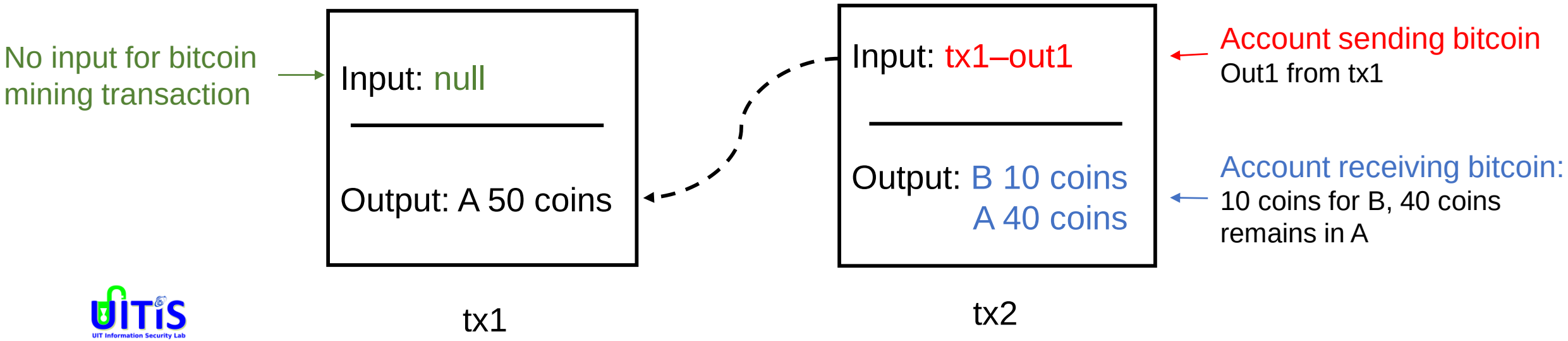
- To keep your “money”, you will need a **wallet**
  - A software
  - Manage **private key**
    - Just looks like ***your bank account password***!
    - *Must keep secret*
  - Initiate bitcoin transactions
- **3 types of wallet**
  - Full node wallet: download whole blockchain
  - SPV wallet: download a part of blockchain
  - Light wallet: no download



UITIS
UIT Information Security Lab

How Bitcoin works?

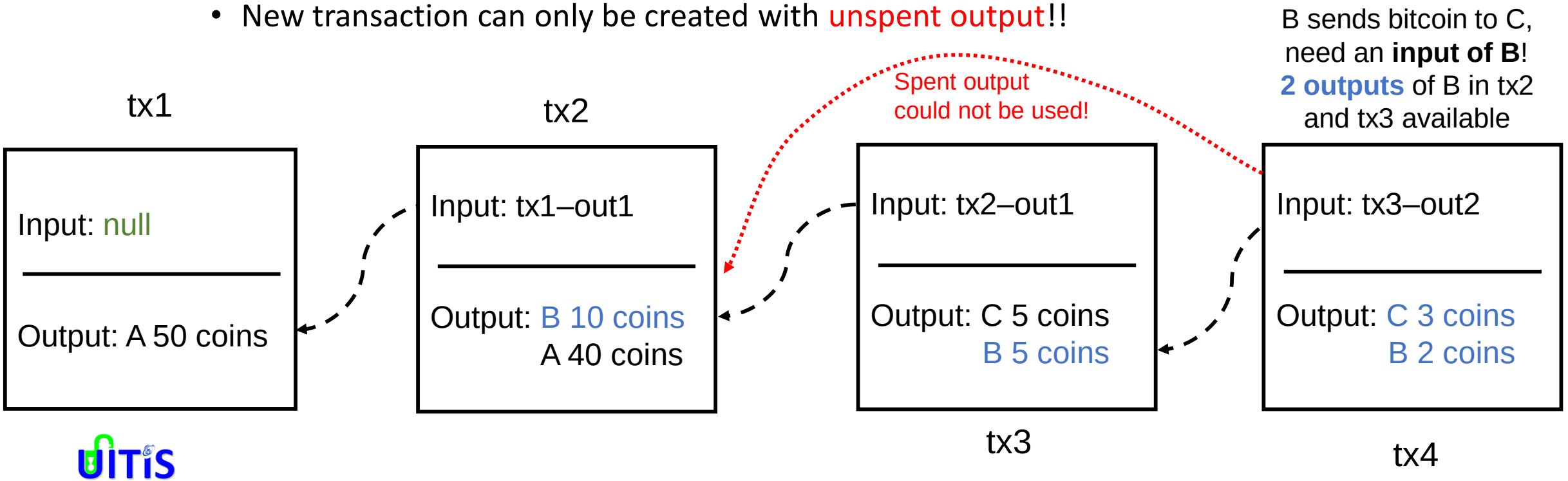
- Proof of work blockchain
- **A block** contains **transactions,** where each transaction has:
  - **Input**
  - **Output**



UITiS
UIT Information Security Lab

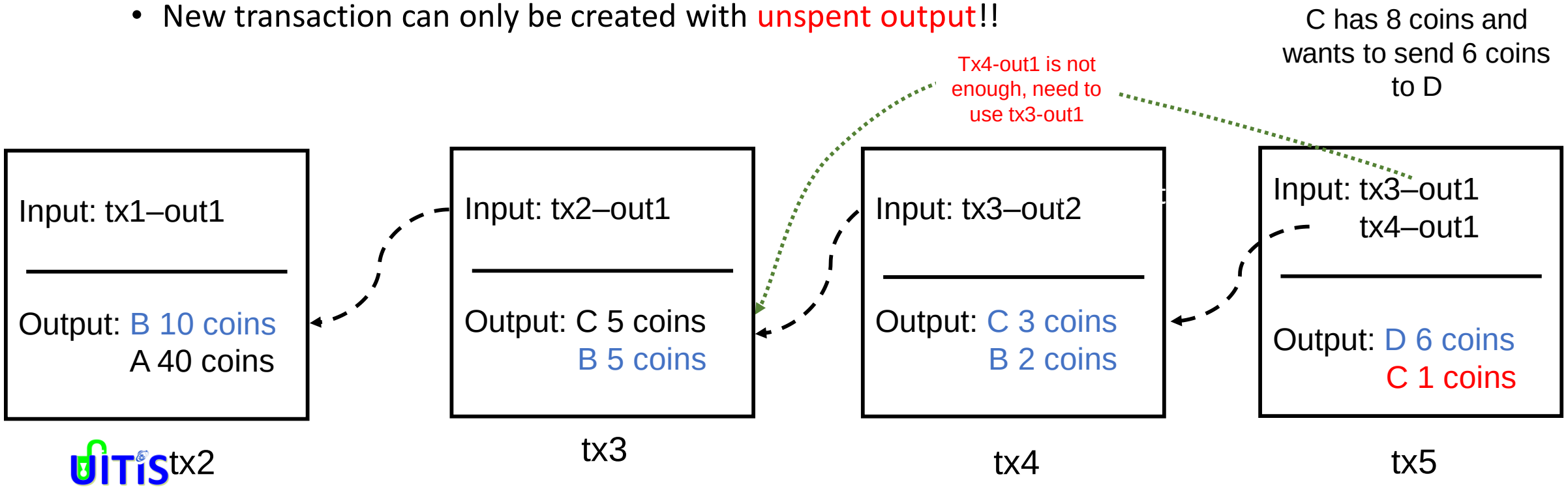
How Bitcoin works?

- **Spent and Unspent output**
  - **Spent output** is output used in one other next transaction.
  - **Unspent output** is not used yet
    - New transaction can only be created with unspent output!!



How Bitcoin works?

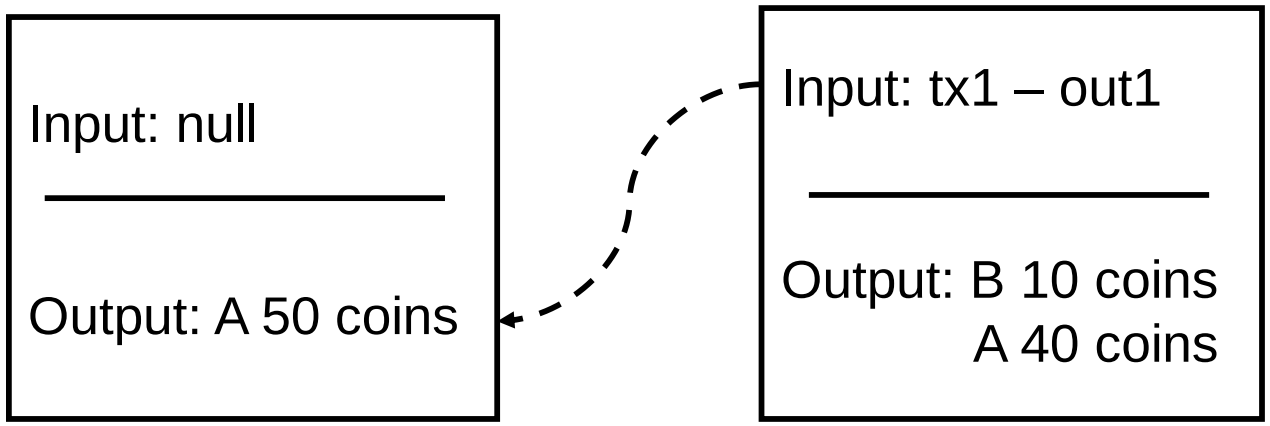
- **Spent and Unspent output**
  - **Spent output** is output used in one other next transaction.
  - **Unspent output** is not used yet
    - New transaction can only be created with unspent output!!



How Bitcoin works?

- Proof of work blockchain
- Transaction base – just maintain transactions related to accounts.

→You have to trace all transactions to get the balance of an account!!



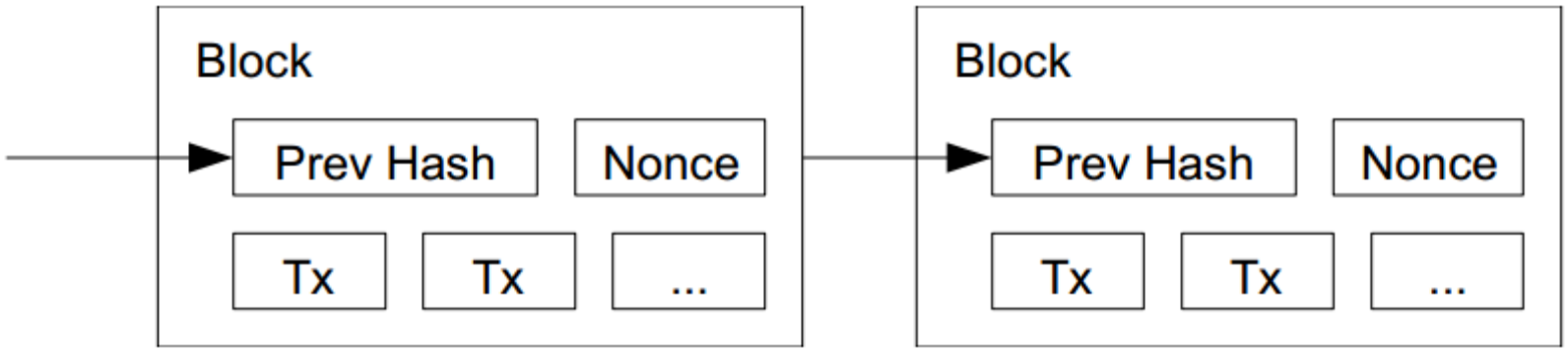
A = +50 – 10 = 40
B = +10 = 10

BitCoins

- Validation
  - Is the coin legit? (proof-of-work) → Use of Cryptographic Hashes
  - How do you prevent a coin from double-spending? → Broadcast to all nodes
- Creation of a virtual coin
  - How is it created in the first place? → Provide incentives for miners
  - How do you prevent inflation? (What prevents anyone from creating lots of coins?) → Limit the creation rate of the BitCoins

Use of Cryptographic Hashes

- Proof-of-work
  - Block contains transactions to be validated and previous hash value.
  - Pick a **nonce** such that H (prev hash, nonce, Tx) < E. E is a variable that the system specifies. Basically, this amounts to finding a hash value who's leading bits are zero. The work required is exponential in the number of zero bits required.
  - Verification is easy. But proof-of-work is hard.



Preventing Double-spending

- The only way is to be aware of all transactions.
- Each node (miner) verifies that this is the first spending of the BitCoin by the payer.
- Only when it is verified it generates the proof-of-work and attach it to the current chain.

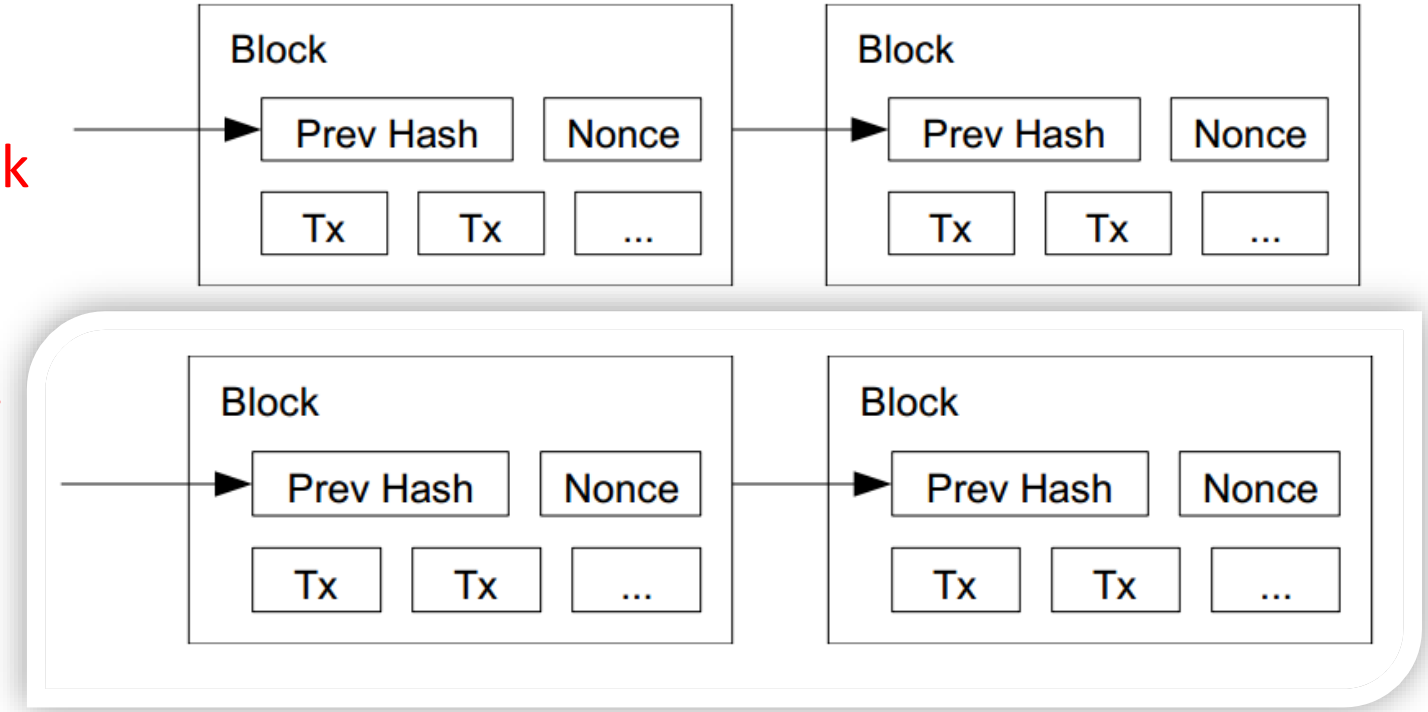
BitCoin Network

- Each P2P node runs the following algorithm [bitcoin]:
  - New transactions are broadcast to all nodes.
  - Each node collects new transactions into a block.
  - Each node works on finding a proof-of-work for its block. (Hard to do. Probabilistic. The one to finish early will probably win.)
  - When a node finds a proof-of-work, it broadcasts the block to all nodes.
  - Nodes accept the block only if all transactions in it are valid (digital signature checking) and not already spent (check all the transactions).
  - Nodes express their acceptance by working on creating the next block in the chain, using the hash of the accepted block as the previous hash.

BitCoin Network: Tie breaking

- Two nodes may find a correct block simultaneously.
  - Keep both and work on the first one
  - If one grows longer than the other, take the longer one

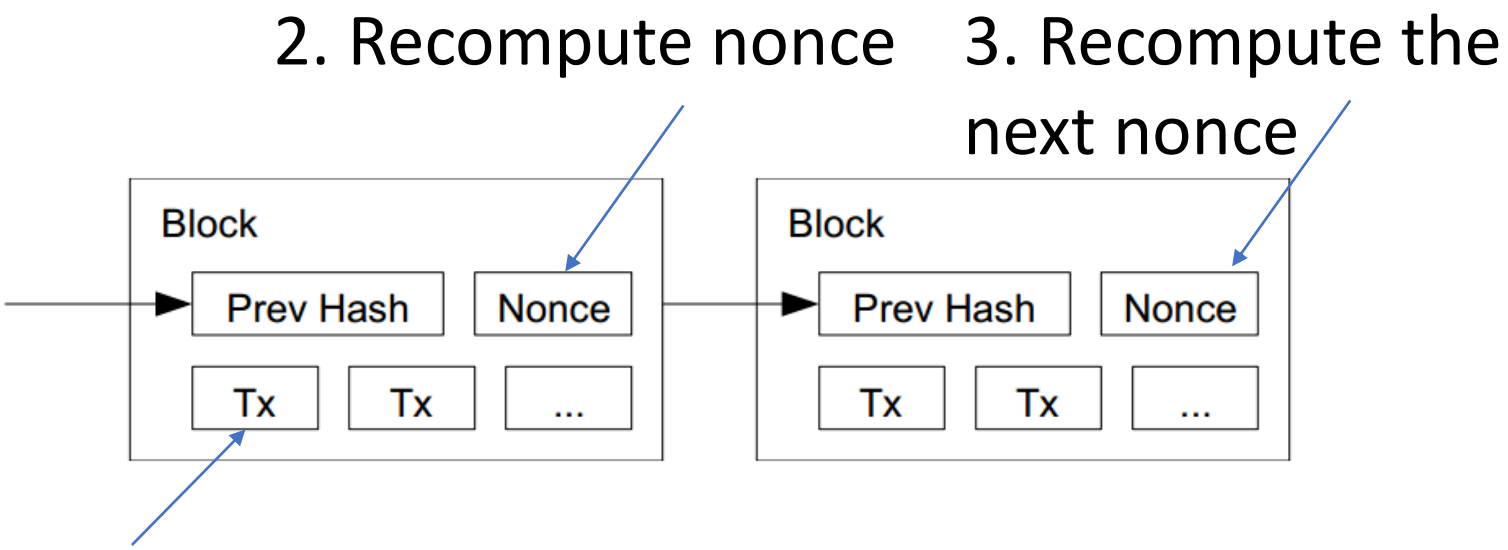
Two different block chains (or blocks) may satisfy the required proof-of-work.



UIT Information Security Lab

BitCoin Network: Reverting is hard…

- Reverting gets exponentially hard as the chain grows.

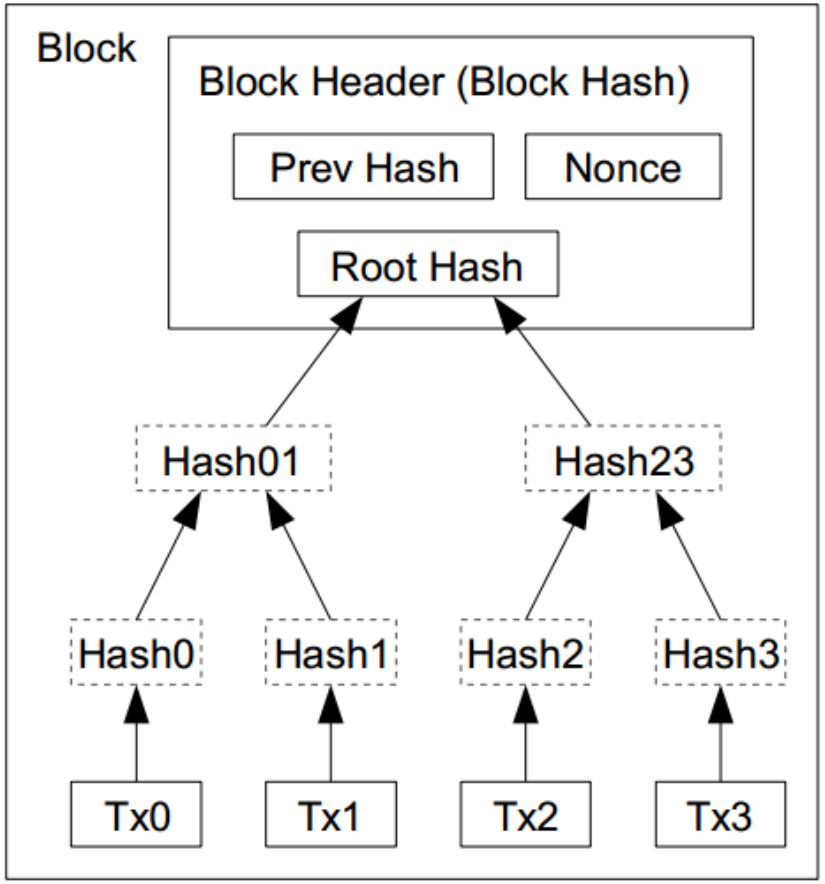


BitCoin Network: Practical Limitation

- At least 10 mins to verify a transaction.
  - Agree to pay
  - Wait for one block (10 mins) for the transaction to go through.
  - But, for a large transaction ($$$) wait longer. Because if you wait longer it becomes more secure. For large $$$, you wait for six blocks (1 hour).

Optimizations

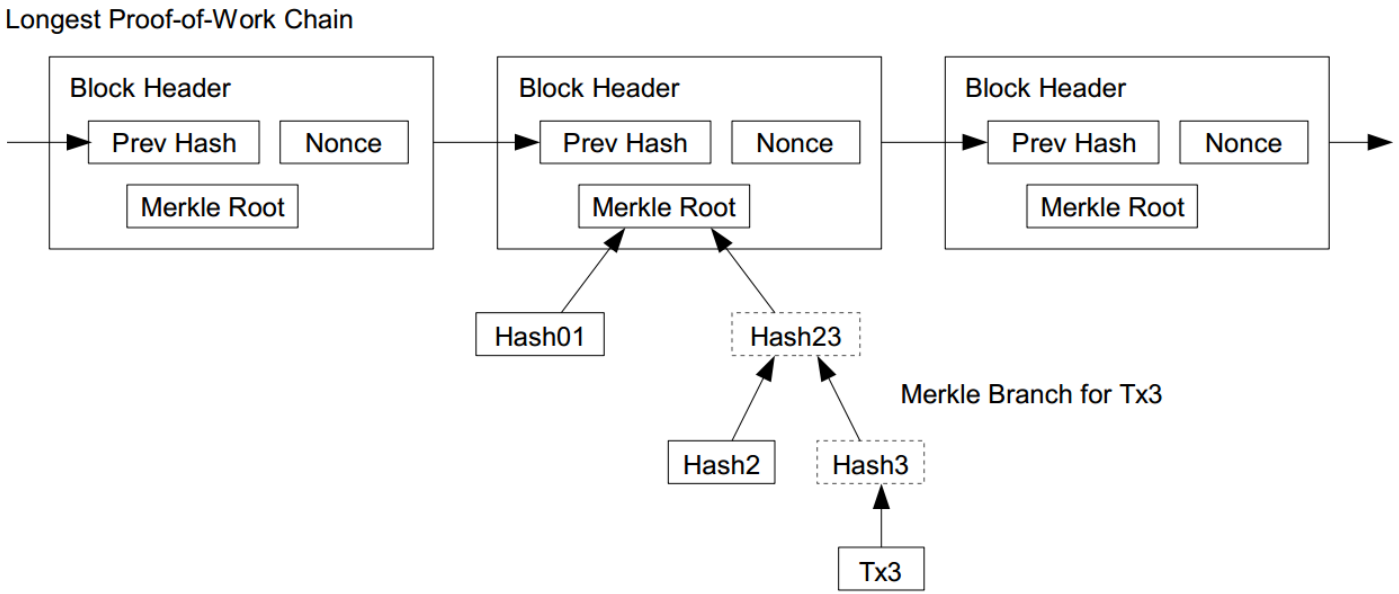
- **Merkle Tree**
  - Only keep the root hash
    - Delete the interior hash values to save disk
    - **Block header (80 bytes)** only contains the root hash
    - 80 bytes * 6 blocks/hr * 24 hrs * 365 = 4.2 MB/year



Transactions Hashed in a Merkle Tree

Simplified payment verification

- Any user can verify a transaction easily by asking a node.
- First, get the longest proof-of-work chain
- Query the block that the transaction to be verified (tx3) is in.
- Only need Hash01 and Hash2 to verify; not the entire Tx's.



UIT Information Security Lab

Transactions

BITCOIN TRANSACTION REQUEST MESSAGE

"David sends 5 BTC to Sandra"

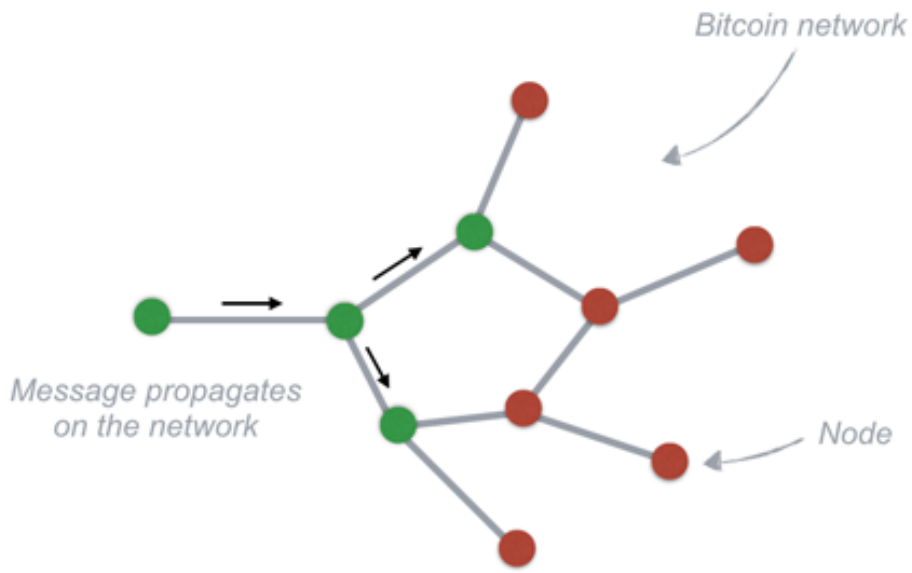
David → Sandra 5 BTC

LEDGER

Account owner	Value
Mary	4
John	56
Sandra	83
Lisa	16
David	187
Brian	23

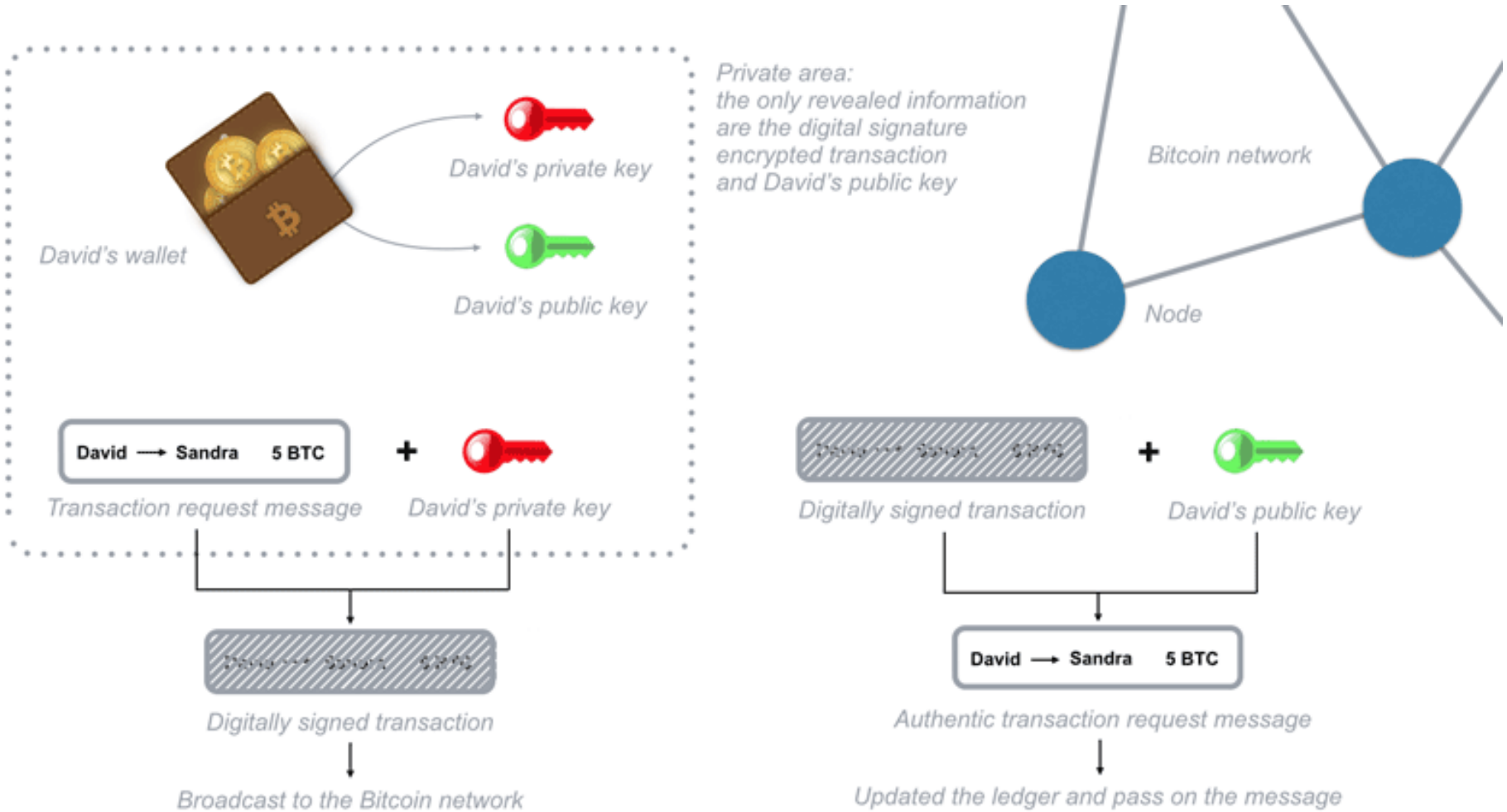
LEDGER

Account owner	Value
Mary	4
John	56
Sandra	88
Lisa	16
David	182
Brian	23

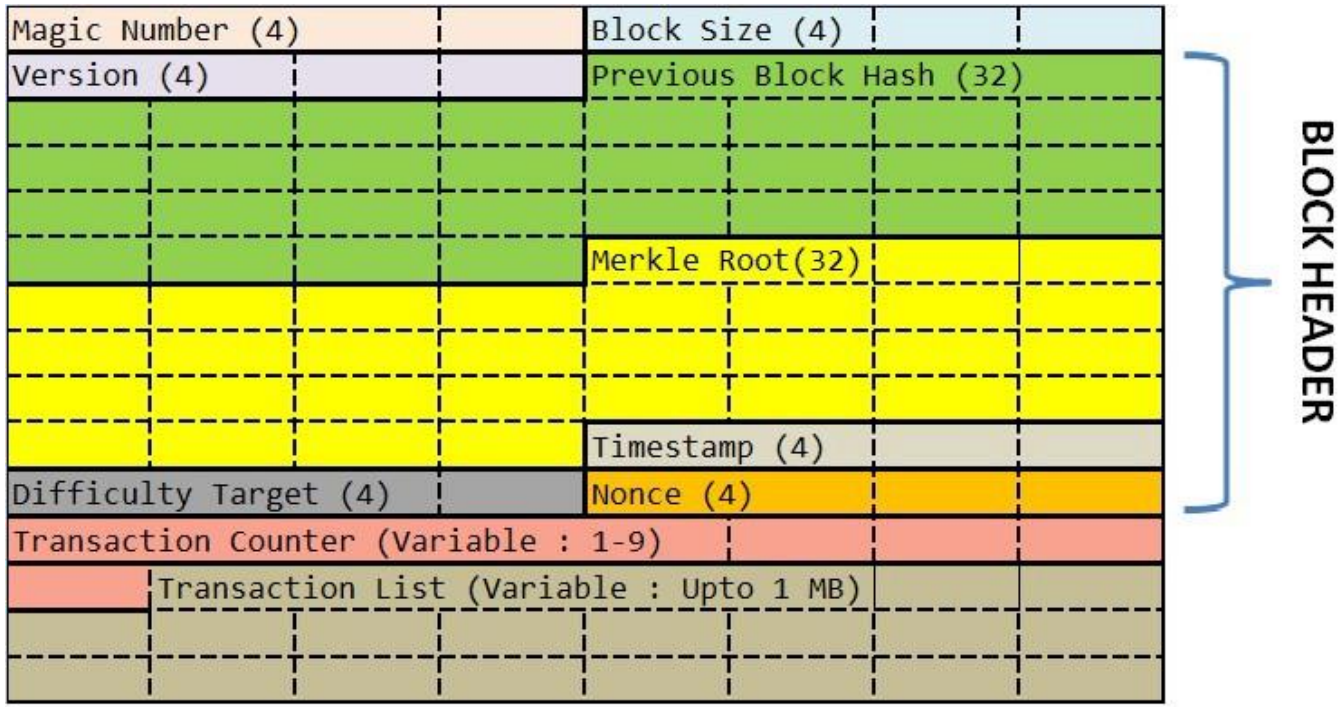


Each *node* receives the transaction request message,
updates its own copy of the *ledger*
and passes on the message to the nearby *nodes.*

UIT Information Security Lab


David's wallet
David's private key
David's public key
Private area:
the only revealed information
are the digital signature
encrypted transaction
and David's public key
Bitcoin network
Node
David → Sandra 5 BTC
Transaction request message
David's private key
Digitally signed transaction
Broadcast to the Bitcoin network
Digitally signed transaction
David's public key
David → Sandra 5 BTC
Authentic transaction request message
Updated the ledger and pass on the message

Block in Bitcoin (1)

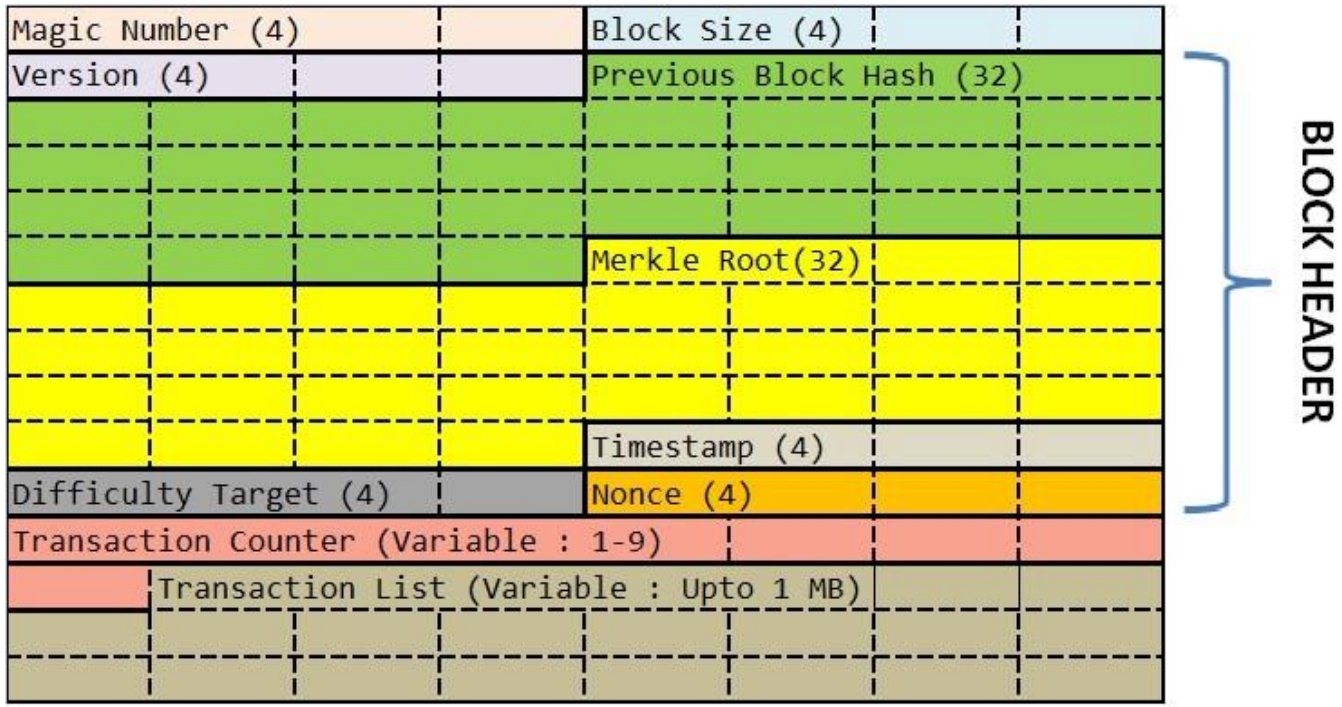


- **Block header** is about 80 bytes. The Block header is composed of the fields from **Version** to **Nonce**.
- **Magic number (4 bytes):** This is an identifier for the Bitcoin Blockchain network. It has a constant value of ***0xD9B4BEF9***
- **Block size (4 bytes):** Indicates how large the block is.

- **Version (4 bytes):** Each node running the Bitcoin protocol has to implement the same version and it is mentioned in this field.
- **Previous block hash (32 bytes)**: This is a digital fingerprint **(hash) of the block header of the previous** (last added) block of the blockchain. It is calculated by taking all the fields of the header.
- **Merkle Root (32 bytes)**
- **Timestamp (4 bytes),** using in mining process
- **Difficulty Target (4 bytes)** using in mining process
- **Nonce (4 bytes)**: using in mining process

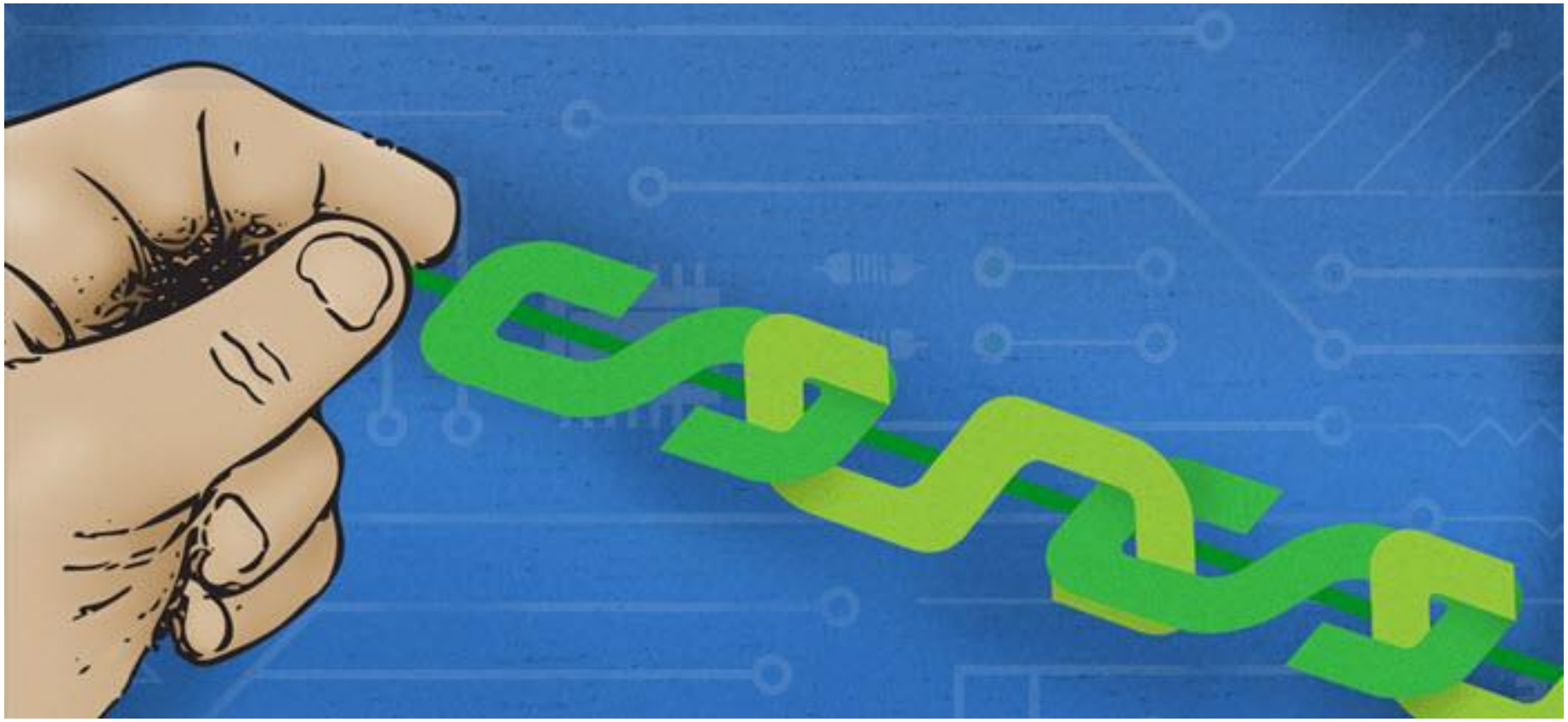
UITiS
UIT Information Security Lab

Block in Bitcoin (2)



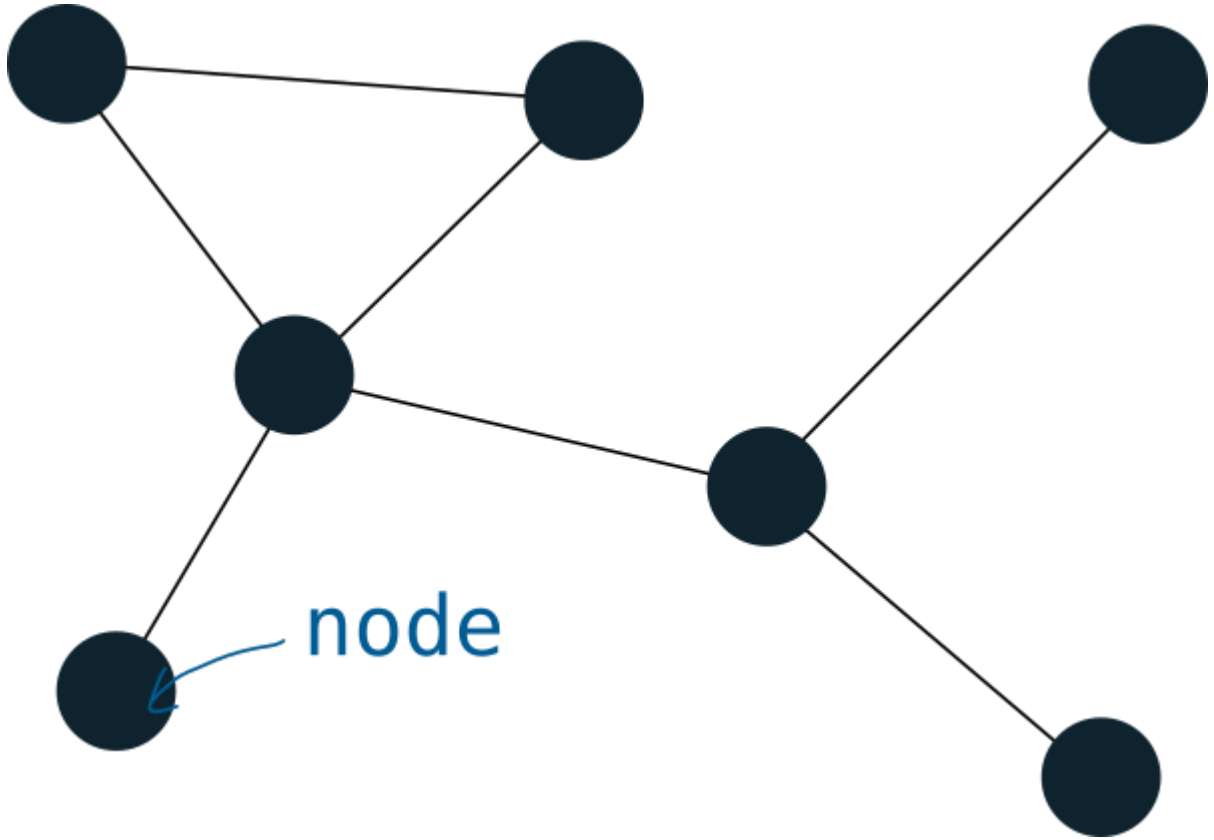
- **Transaction Counter (Variable: 1–9 bytes)** : This is the count of transactions that are included with the block.
- **Transaction List (Variable: Total block size is 1 MB)**: Stores the digital finger-print **of all the transactions** in that block.
  - ✓ Each hash is calculated by applying the **SHA256 algorithm twice**.
  - ✓ Hash of Transaction A = Hash[Tx(A)] = SHA256 (SHA 256 (Transaction A)) (32 bytes)
  - ✓ Hash of Transaction B = Hash[Tx(B)] = SHA256 (SHA 256 (Transaction B)) (32 bytes)
  - ✓ Parent node Hash (AB) = Hash(Hash[Tx(A)] + Hash[Tx(B)]) = Hash (Tx(AB))= SHA256 (SHA 256 (Tx(AB))) (32 bytes)

UITiS
UIT Information Security Lab



Blockchain Concepts

Nodes



- A node is just a **computer that is running the program (Bitcoin, …)**. It is *connected to other computers* (running the same program) to create a Network.

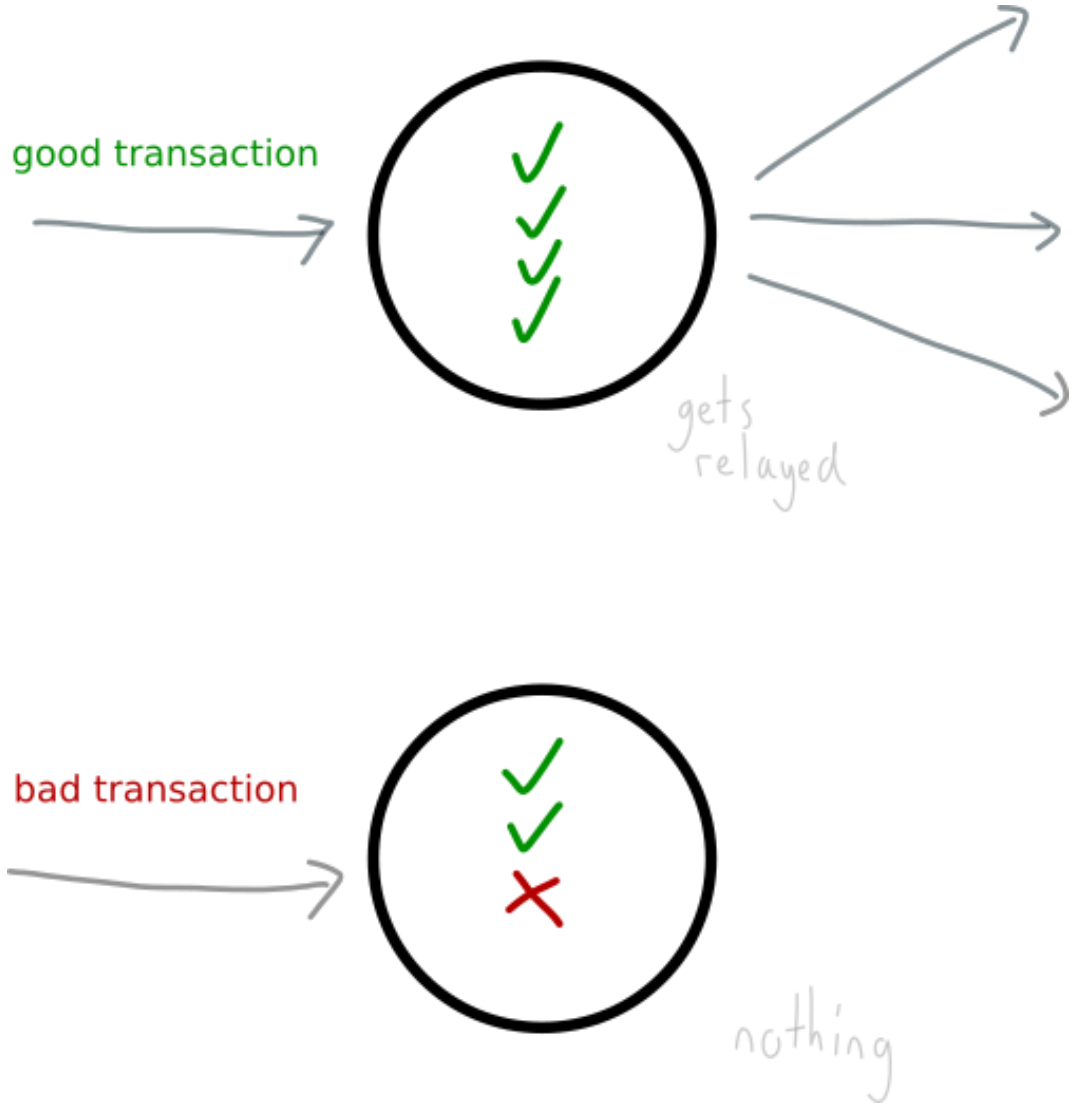
UITiS
UIT Information Security Lab

What does a node do?

- A node has three jobs:
  - Follow rules
  - Share information
  - Keep a copy of confirmed transactions

Follow rules

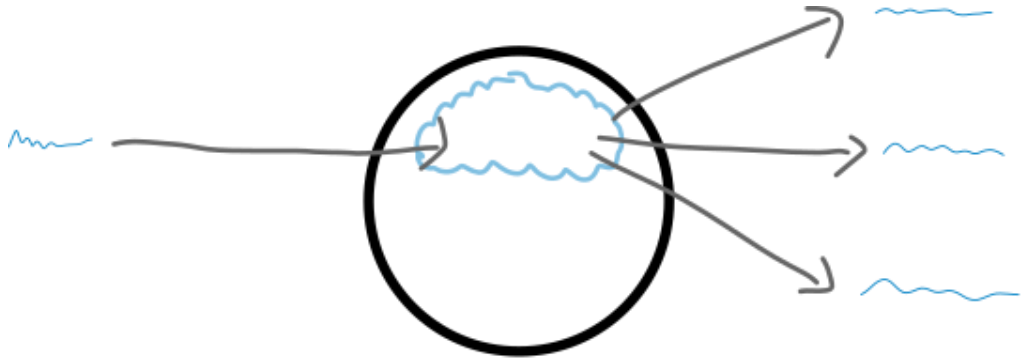
- Each node has been programmed to follow a set of rules.
- By following these rules a node is able to check the transactions it receives and only relay them if everything is cool.
- If there are any problems, the transaction isn’t passed on.



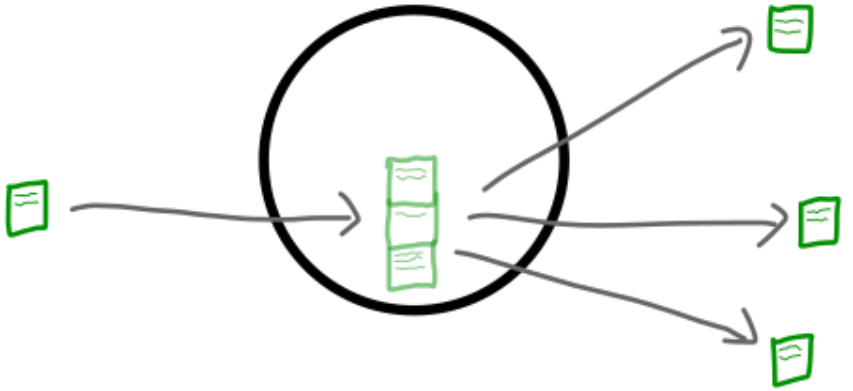
UITiS
UIT Information Security Lab

Share information

- Now, there are two types of transactions that nodes share:
  - Fresh transactions – transactions that have recently entered the network.
  - Confirmed transactions – transactions that have been "confirmed" and written to a file. These are shared in ***blocks*** of transactions, and not individually.



sharing single "fresh" transactions

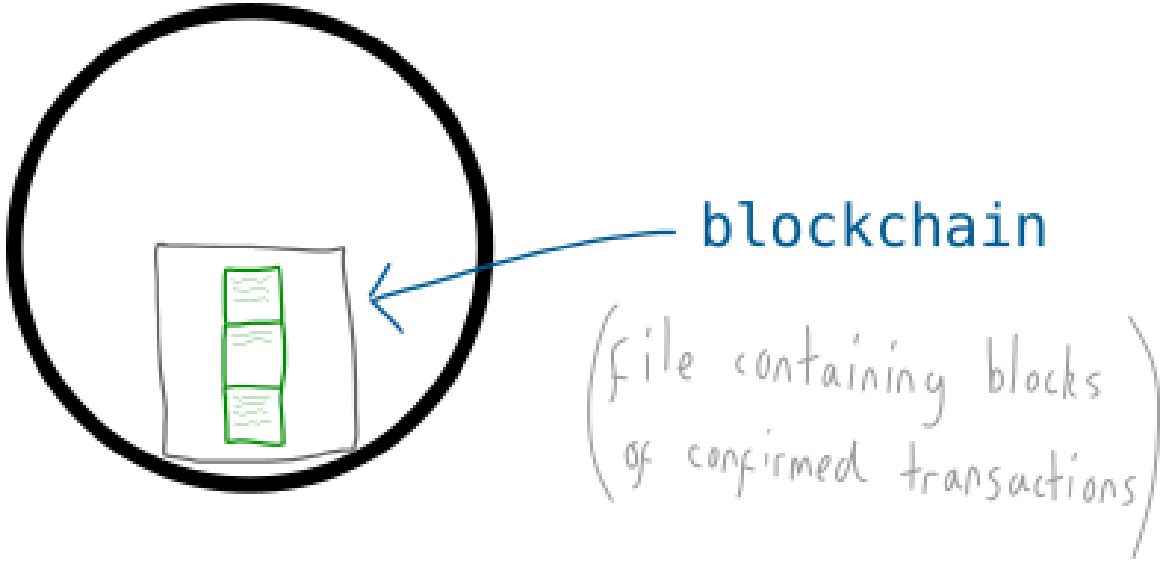


sharing blocks of *confirmed* transactions

UITiS
UIT Information Security Lab

Keep a copy of confirmed transactions

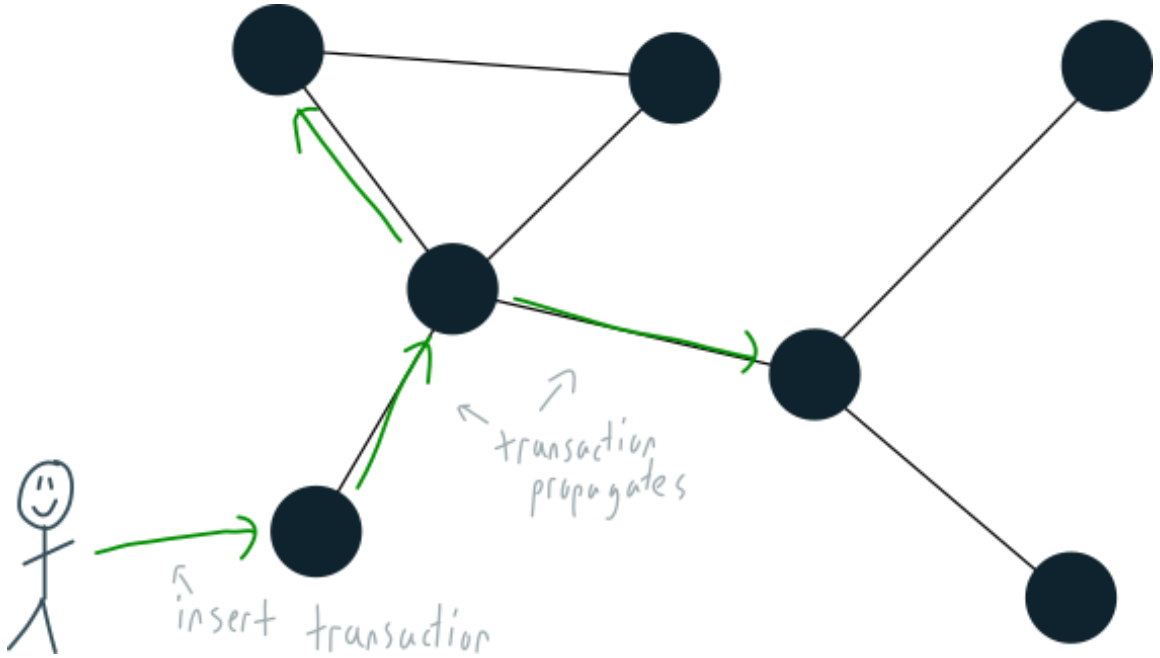
- ***Each node*** also ***keeps*** *blocks of confirmed transactions*. These are held together in a file called the **blockchain**.
- Fresh transactions are bounced around the network until they are etched in to the blockchain, which is a ledger of confirmed transactions.
- Each node has a copy of the blockchain for safe keeping, and shares it with other nodes if their copy isn't up to date.
- The process of adding fresh transactions to the blockchain is called Mining/Validating.



UITIS
UIT Information Security Lab

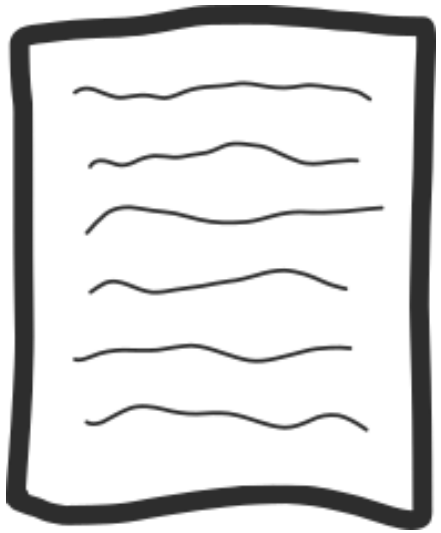
Do I have to be a node to use “bitcoin”?

- No.
- You can send and receive bitcoins without having to be a node. You just need to get the transaction in to the bitcoin network
- If you send a message to one node about a transaction, it will eventually propagate the entire network.



UITIS
UIT Information Security Lab

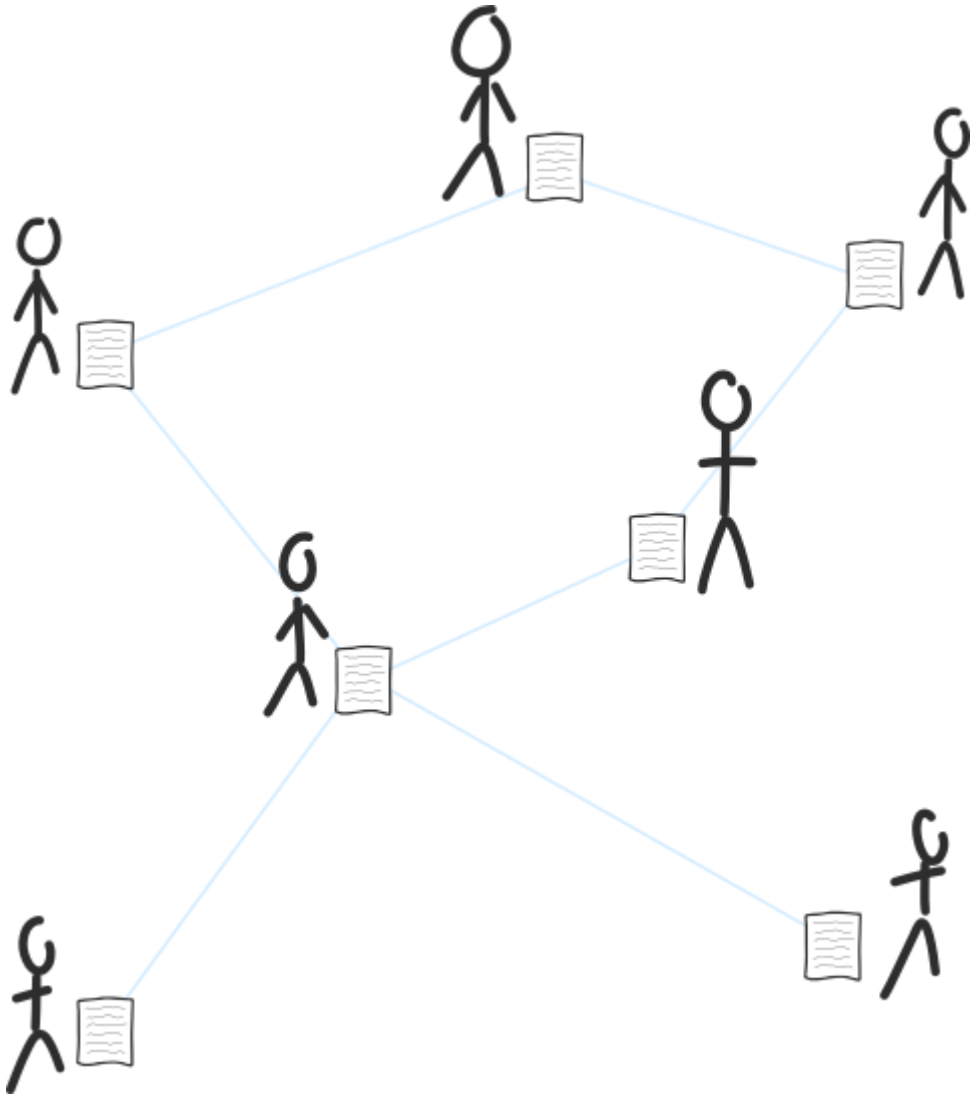
What is the blockchain?



- **The blockchain is a file that contains a list of every transaction ever made.**
- The blockchain is like a logbook, or a *ledger*.
- Ledger – a book in which the monetary transactions of a business are posted in the form of debits and credits.

What is the blockchain?

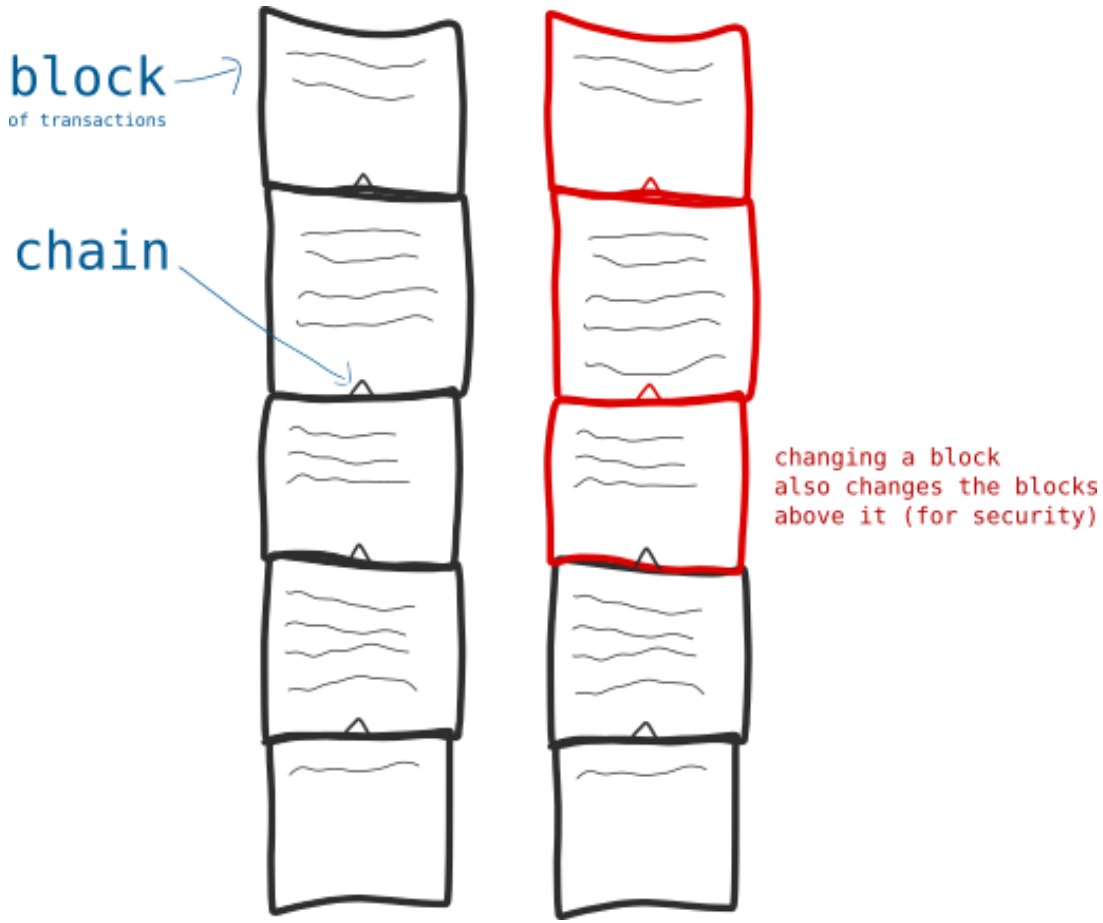
- Everyone on the blockchain network shares a copy of this file, and it updates regularly with the latest transactions.



UIT Information Security Lab

Why is it called the blockchain?

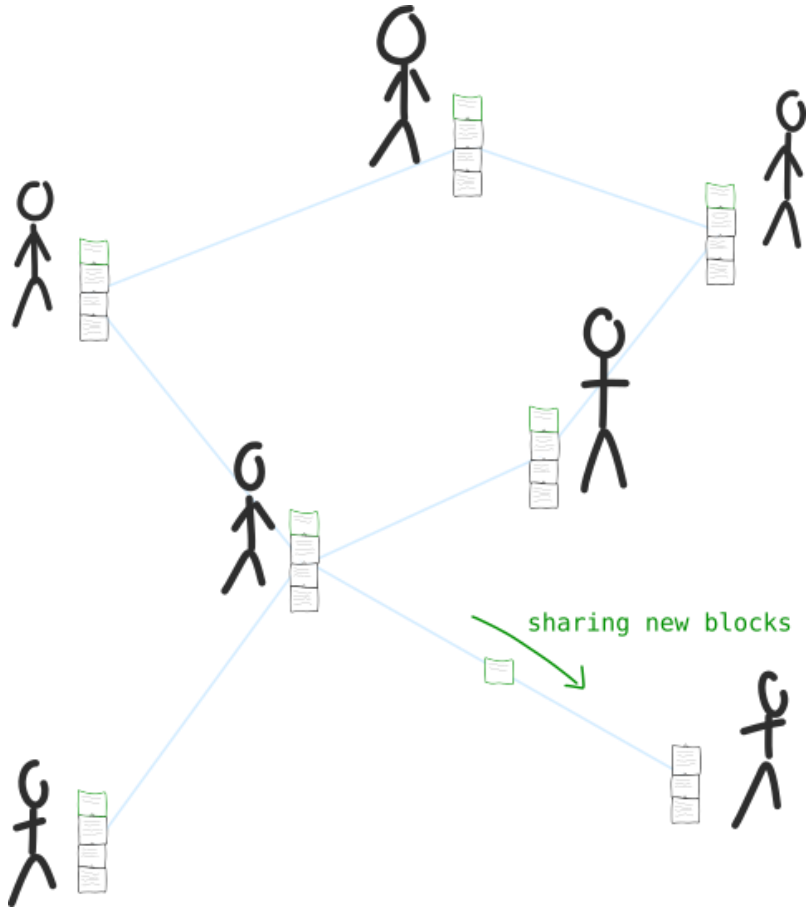
- Because transactions aren't added to the file individually. Instead, they are bunched together and added in blocks. Hence, **block**chain.
- Also, these blocks are *linked* together, so any changes made to a block lower down the chain will change the blocks above it. So **linked**blocks, or block**chain**.



Transactions are added to in blocks, and these blocks are chained together.

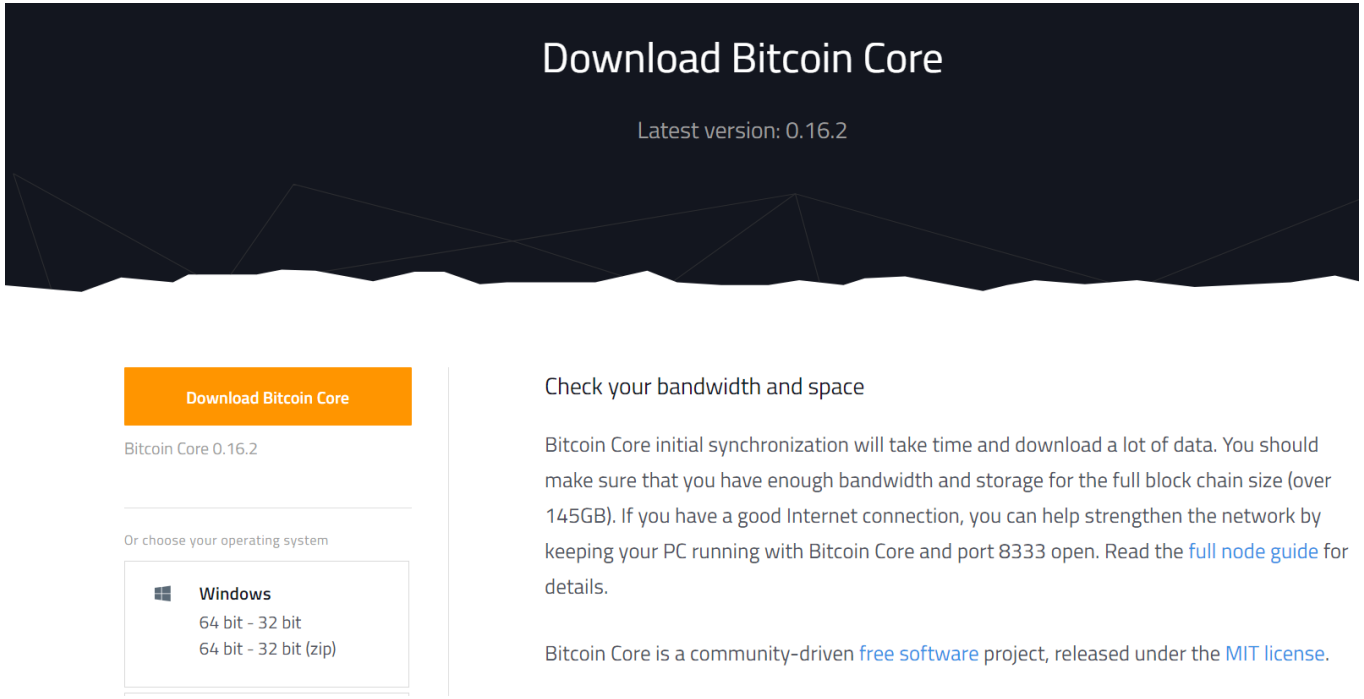
How is the blockchain shared?

- The blockchain is shared by the nodes on the P2P network, in the same way a totally legit and non-copyrighted video file might be shared on the BitTorrent network.



Ex: Bitcoin peer-to-peer file sharing of the blockchain. If my file hasn't got the latest blocks of transactions, someone will share them with me.

Where can I get a copy of the blockchain?



- You can get your own copy of a genuine blockchain by downloading the original blockchain client (like Bitcoin client,…).
- By keeping a copy of the blockchain and sharing it with other people on the network, you make bitcoin stronger.
- If you’re a fan of bittorrent, you could think of yourself as **seeding** the blockchain.

UITIS
UIT Information Security Lab

Where is the blockchain file stored on my computer?

- The blockchain is stored in files with names like this: **blk00000.dat**. There's also blk00001.dat, blk00002.dat, and so on. (It's split in to multiple files because it's easier than working with one huge file.)
- Their location depends on what operating system you're using:
  - Linux

    /home/[username]/.bitcoin/blocks/
  - Windows

    C:\Users\[username]\AppData\Roaming\Bitcoin\
  - Mac

    ~/Library/Application Support/Bitcoin/

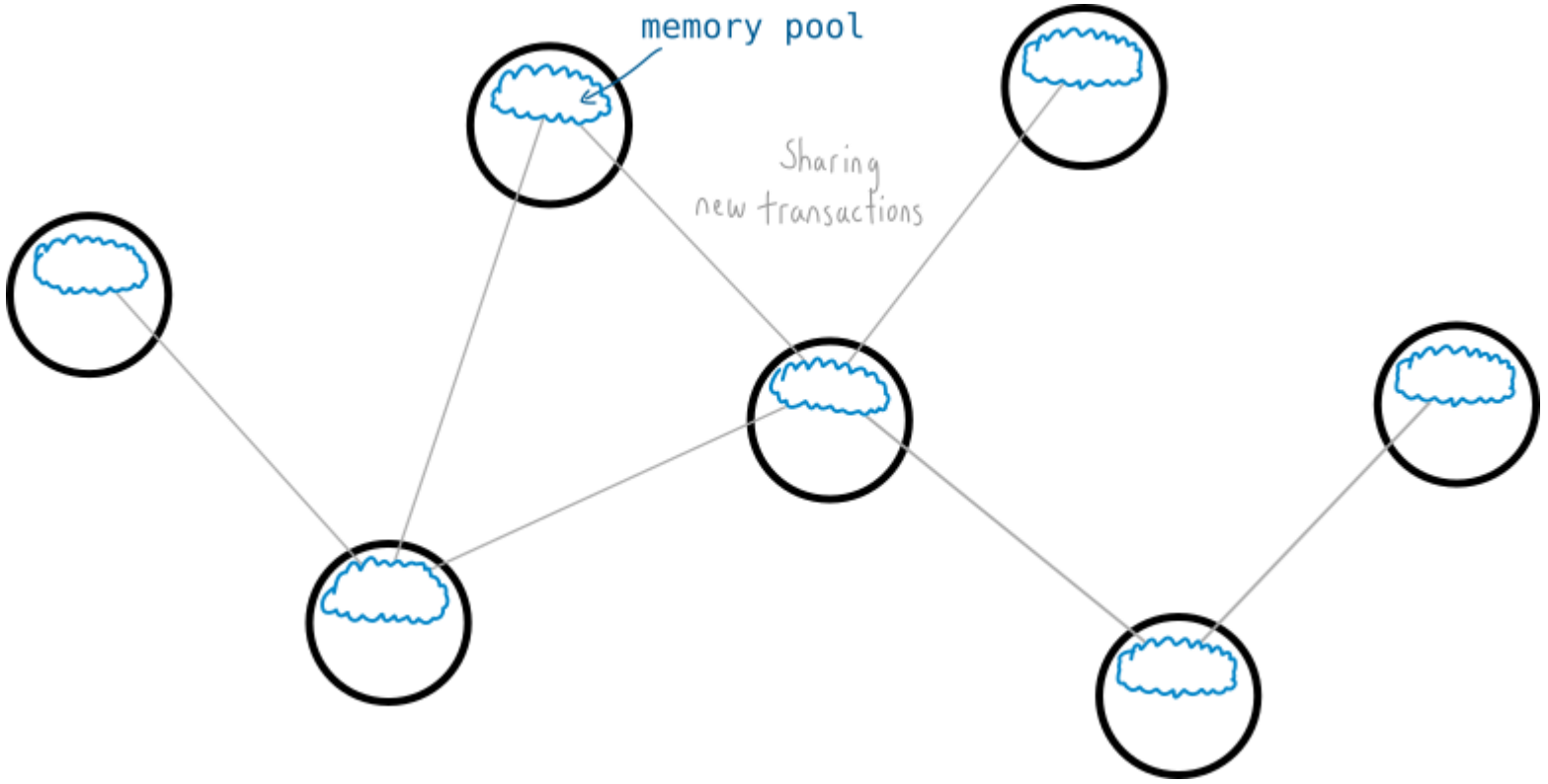
MINING

What is Mining?

- Mining is the process of adding transactions to the blockchain.

How does it work?

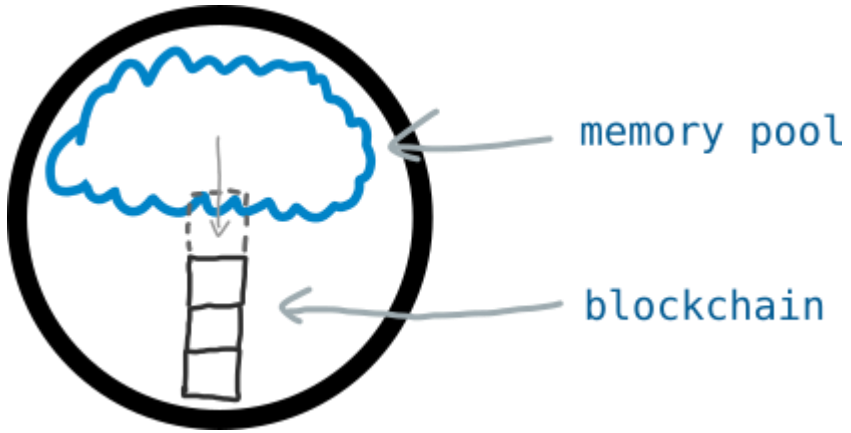
- Every node on the blockchain network ***shares information*** about *new transactions*. They store these transactions in their **memory pool**.
- The memory pool is a node's temporary storage area for transaction data.



UITiS
UIT Information Security Lab

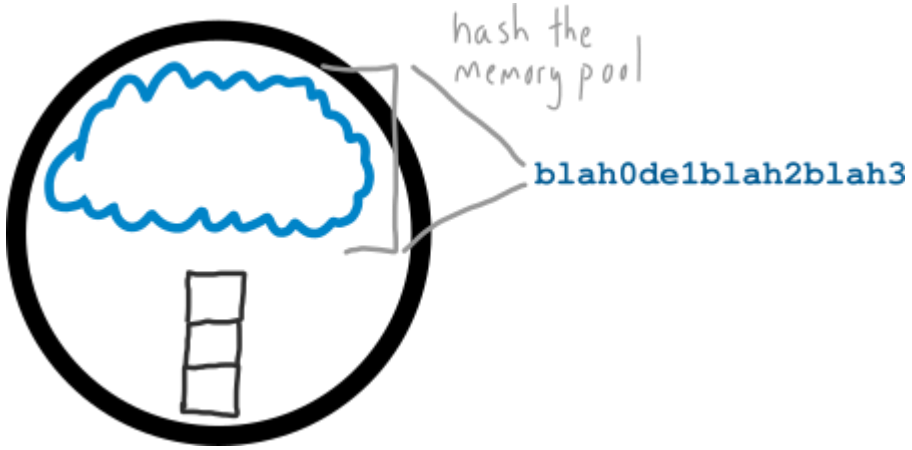
Mining

- Each node also has the option to try and “mine” the transactions in their memory pool in to a **file**. This file is a ledger of every bitcoin transaction, and it’s called the **BLOCKCHAIN**.
- You could think of the memory pool as “**floating**” transactions and the blockchain as “archived” transactions.
- **However, to add transactions from the memory pool to the blockchain, a node has to use a lot of computer processing power.**
- This processing power is forced through the existence of a ***challenge*** in the memory pool.

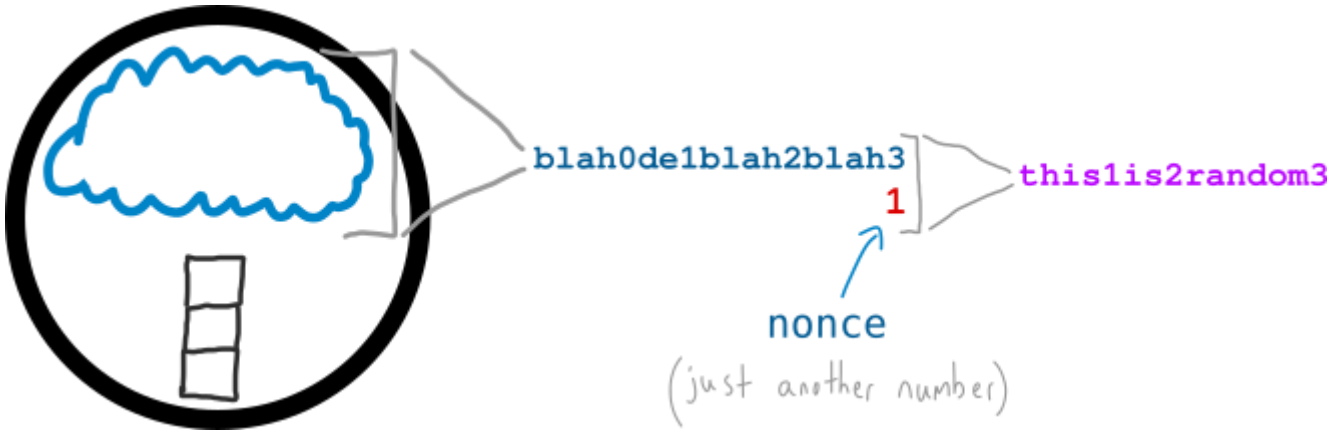


What is this challenge?

- Imagine you're a node. At any moment in time you can condense the transactions in your memory pool in to a single "string" of numbers and letters.
- This string represents all of the transactions in your memory pool.

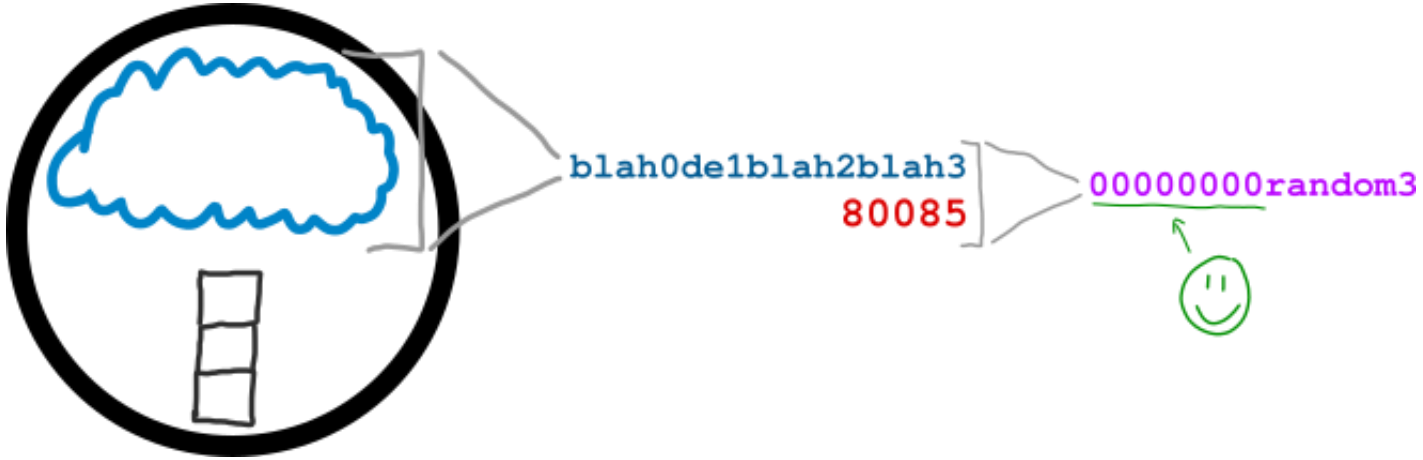


- Now, your objective is to HASH this string with *another number* to **try and get a new string that begins with a certain number of zeros**.
- The “certain number of zeros” comes from the **difficulty**. This is set by the network and changes based on the volume of miners – the more people mining, the greater the difficulty and the more zeros are needed at the start (which makes it harder to find a winning result).
- Most of the time you will get a result that isn’t even close.



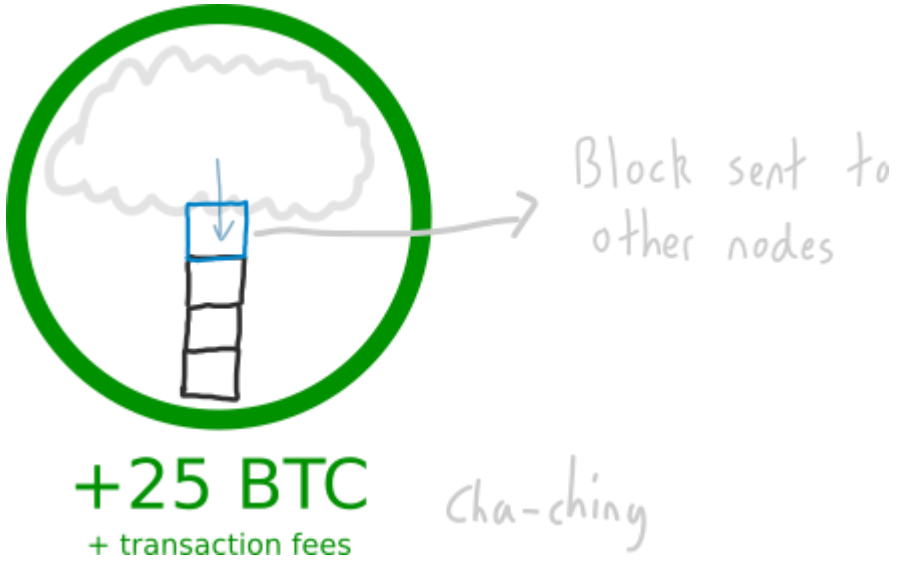
UIT Information Security Lab

- But if you keep going you may stumble upon a number that works.
- Now, this sounds easy enough, but it's **actually difficult**. It's utterly random, and you can only hope to find a winning result through trial and error. And that's what Mining *is* ***– lots of hashing*** (using lots of your computer's processing power) and ***hoping to get lucky***.



UITiS
UIT Information Security Lab

- But if you are lucky enough to find a successful hash result, the transactions in your memory pool get added to the blockchain, and every other node on the network adds your block of transactions to their blockchain.
- You'll also receive a **25BTC** reward for your effort, as well as picking up any fees that were tacked on to the transactions that you just added to the blockchain.

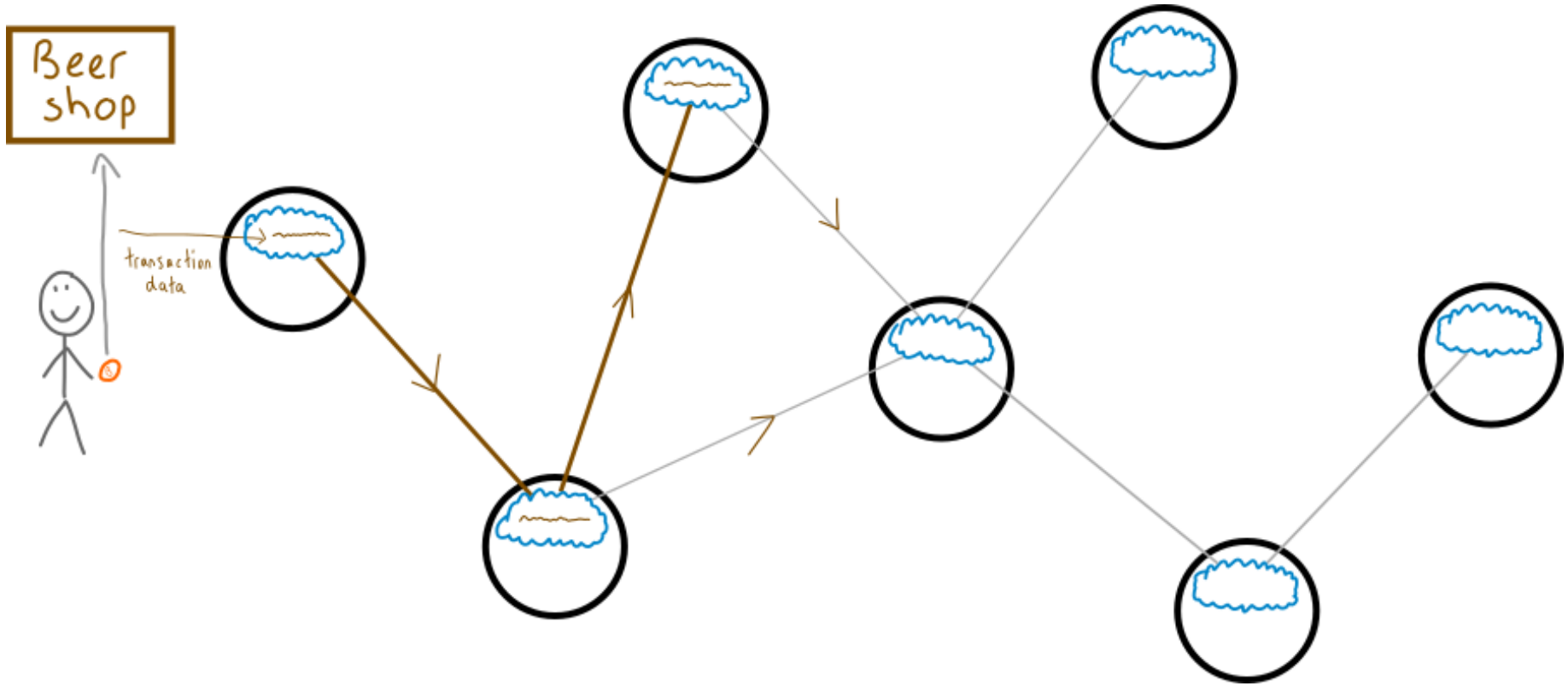


Why is Mining necessary?

- *Why not add transactions directly to the blockchain?*
- Because mining allows the entire Bitcoin Network to agree on which transactions get “archived”, and this is how you prevent fraud in a digital currency.

Why is Mining necessary?

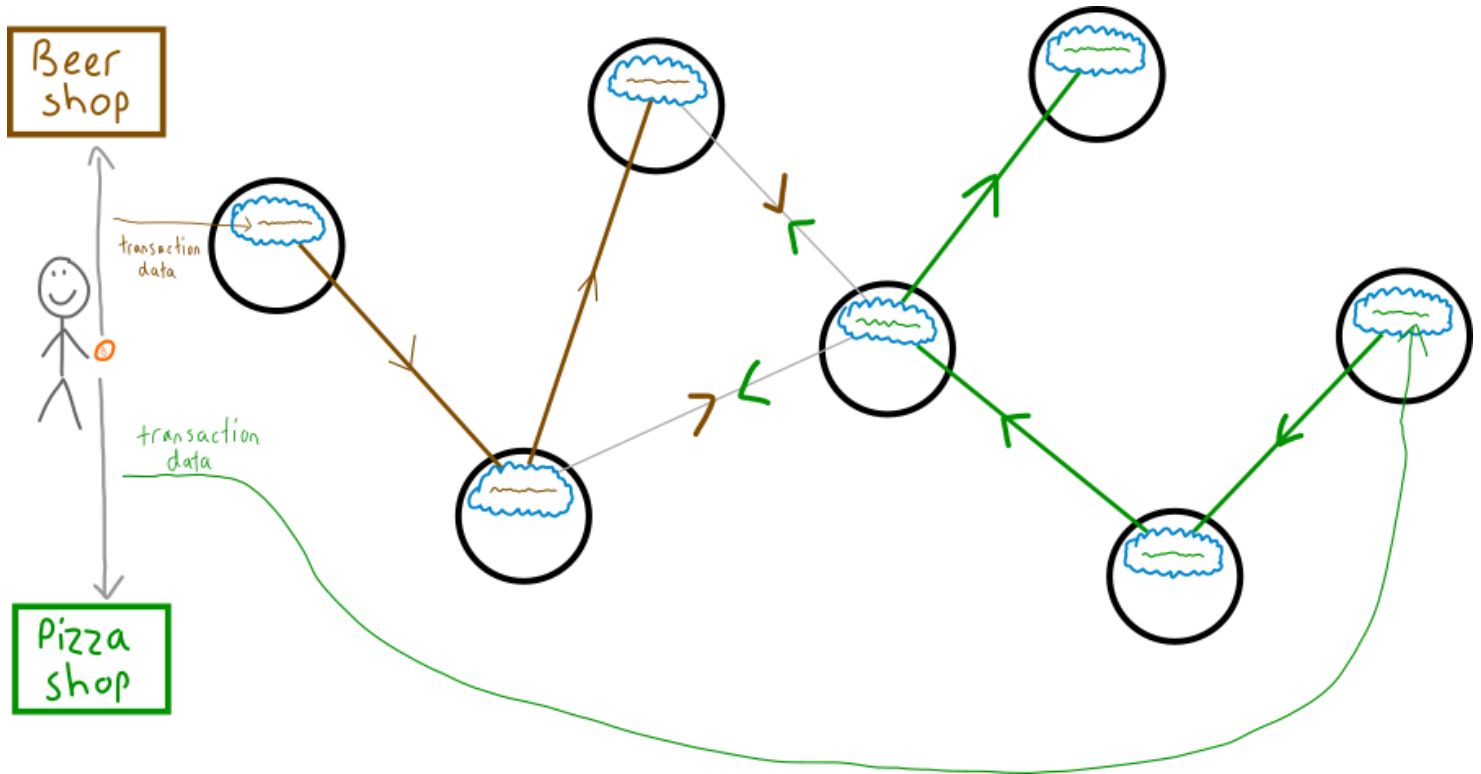
- When you make a bitcoin transaction, nodes on the network do not hear about it instantly. Instead, ***transactions travel across*** the blockchain network by being passed from one node to the next.



UIT Information Security Lab

Why is Mining necessary? Fraud?

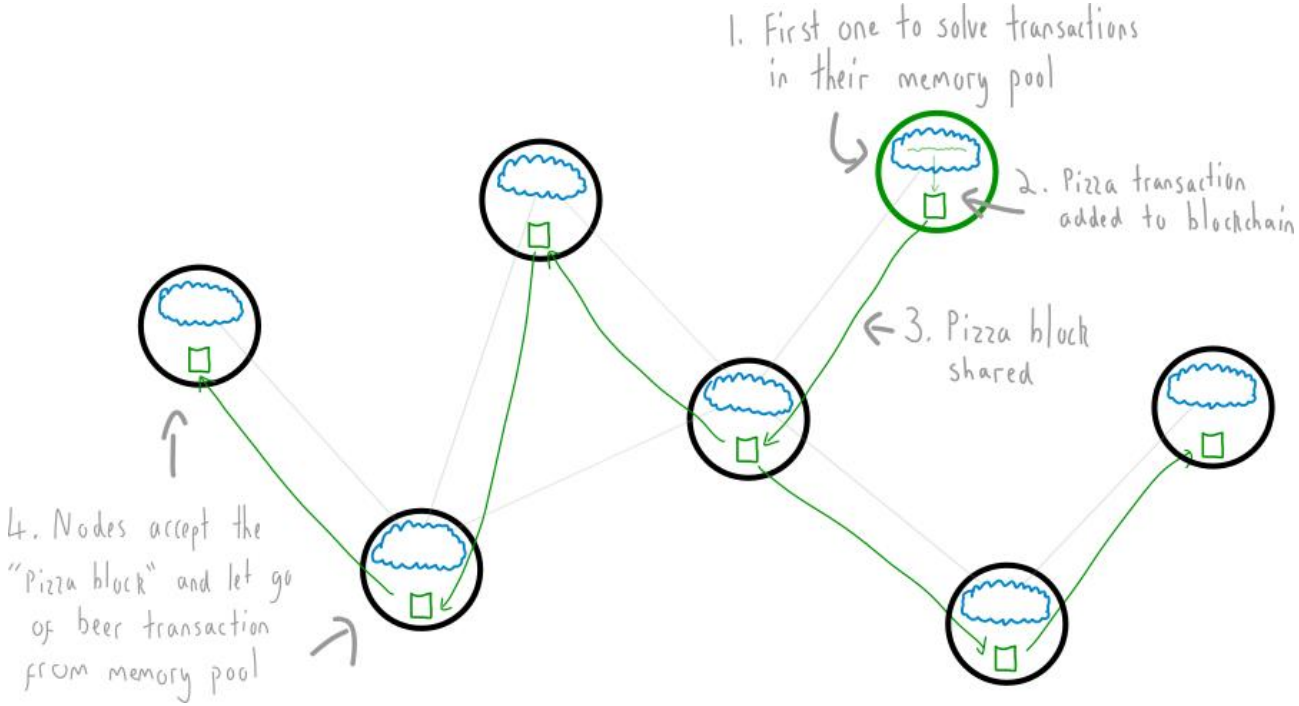
- However, it's actually possible to make *another* transaction using those *same* bitcoins and *insert that transaction* in to the network too.
- For example, you could *buy a beer with some bitcoins*, then quickly attempt to *buy a slice of pizza* with those same bitcoins too.
- Some nodes would get the pizza transaction first (and ignore the beer transaction), whereas others would get the beer transaction first (and ignore the pizza transaction).
- Yet even though you make the pizza transaction *after* the beer transaction, due to the way transactions travel across the Bitcoin Network, the network would be in a **disagreement** about ***whether you should get the beer or the pizza***.



UIT Information Security Lab

So how does the network decide which transaction to go with?

- **Mining, of course.**
- If a node on the network completes the challenge, then it's the transactions in ***their memory pool*** that get added to the blockchain.
- If a node with the pizza transaction successfully mines a block, then that's the transaction that gets added to the blockchain, and the beer transaction evaporates from the network.
- On the plus side, it only takes about 10 minutes for each new block of transactions to be added to the blockchain, so you only need to wait 10 minutes for a confirmation that bitcoins have "arrived" at a new address (and haven't been sent to an alternative address).



UiTiS
UiT Information Security Lab

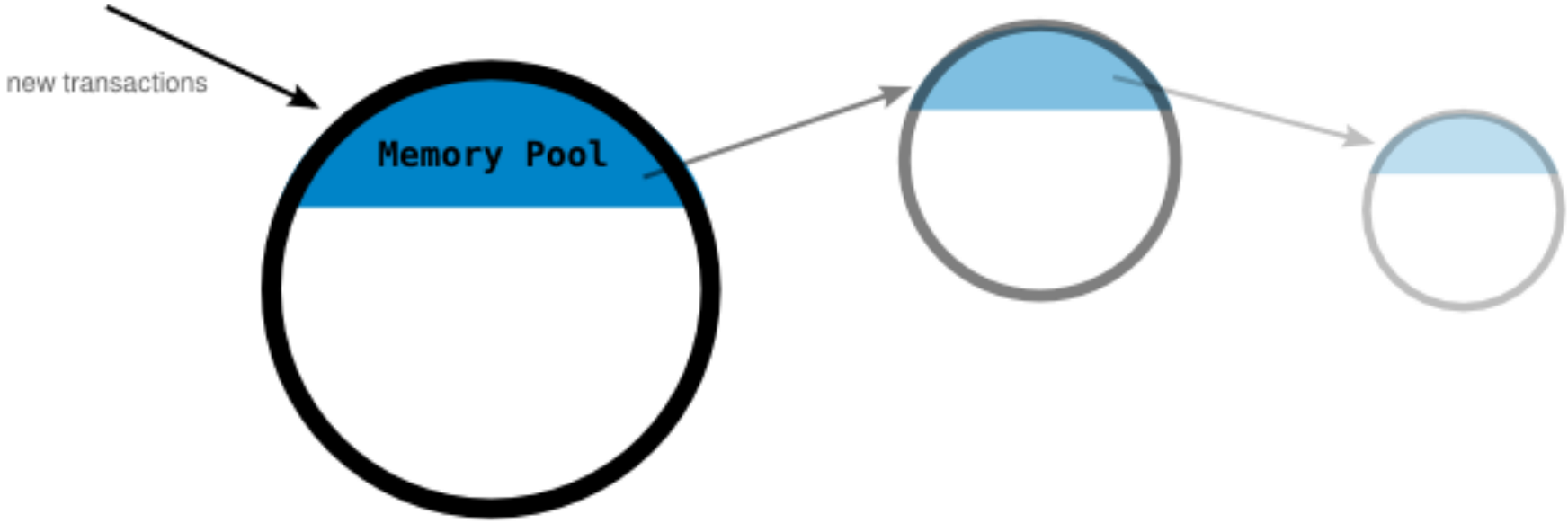
Another benefit of Mining

- If you want to try and control the blocks (i.e. transactions) that get added to the blockchain, you ***have to compete to solve block puzzles*** with *every other mining node* on the bitcoin network.
- In other words, you need to have a computer with enough processing power that is able to out-work the combined processing power of every other bitcoin miner.
- Which is entirely possible – you just need to spend a few billion on hardware and you're good to go (although this figure increases with every new miner who joins the network).

MEMORY POOL

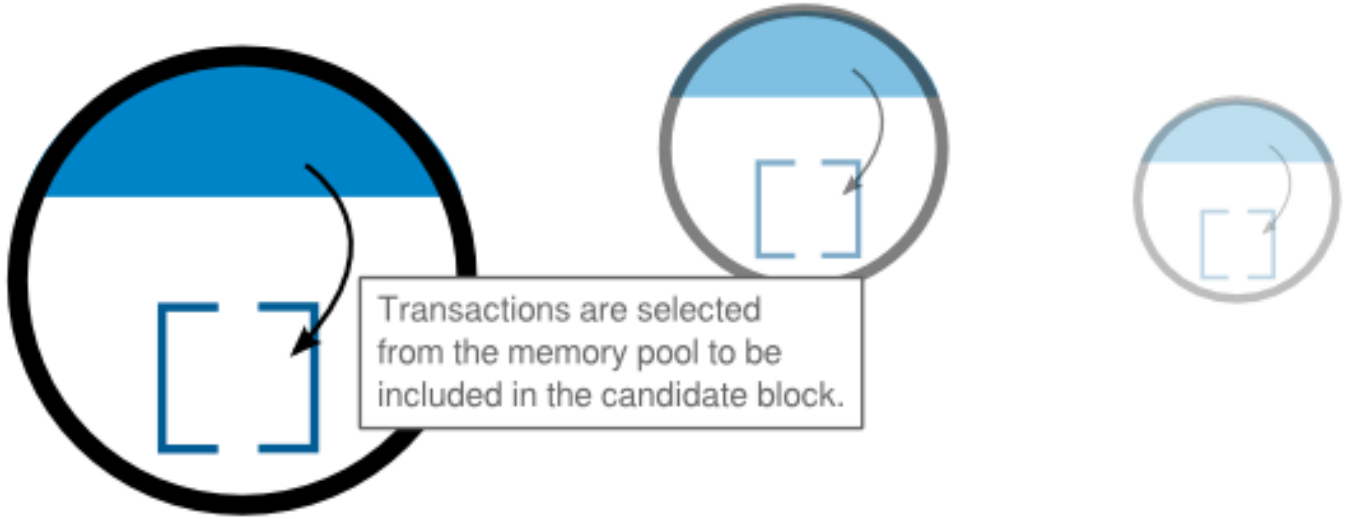
MEMORY POOL

- **The memory pool is a temporary storage area for transactions.**



MEMORY POOL

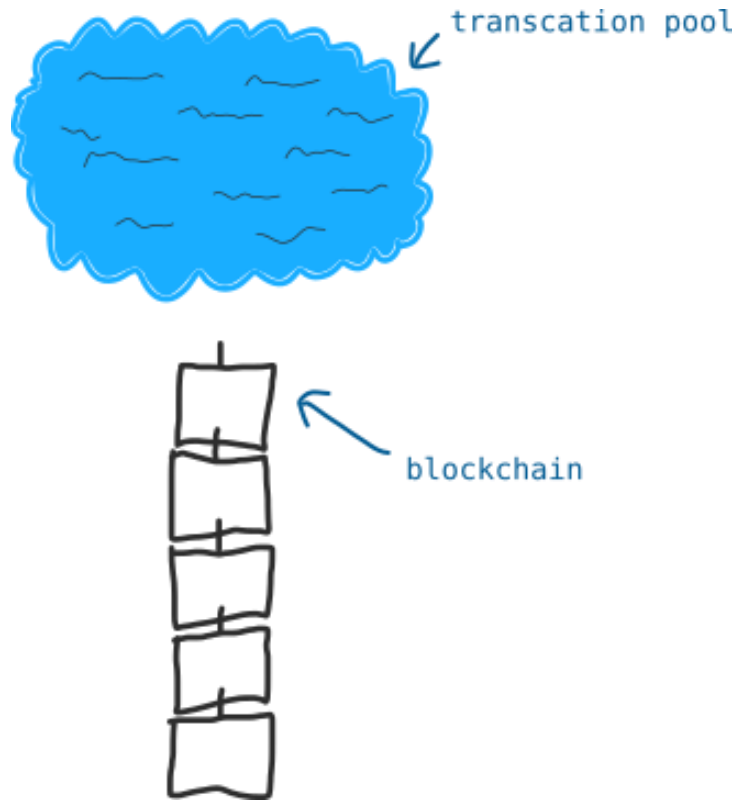
- When a new transaction is received by a node, it will *hold it in its memory pool* with *all the other latest transactions* it has received. From here the transaction will be hoping to get selected for inclusion in the **candidate block**.



Block

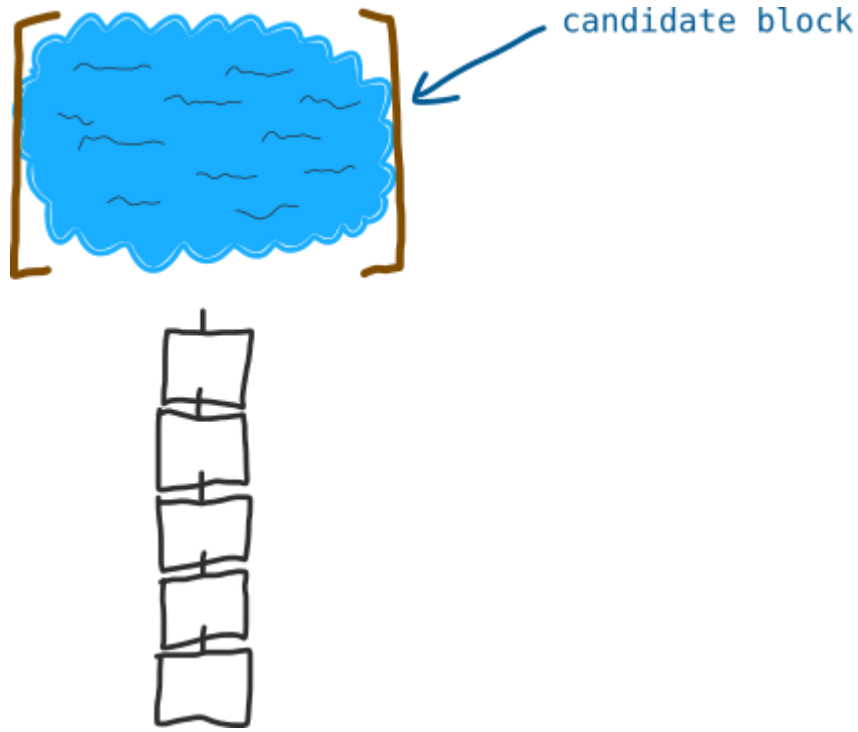
What is a block?

- A block is a bunch of transactions that have been added to the blockchain.
- Blocks are formed by miners/validators.
- When you make a transaction, it isn't added to the blockchain straight away. Instead, it is held in a TRANSACTION POOL (or ***memory pool***).



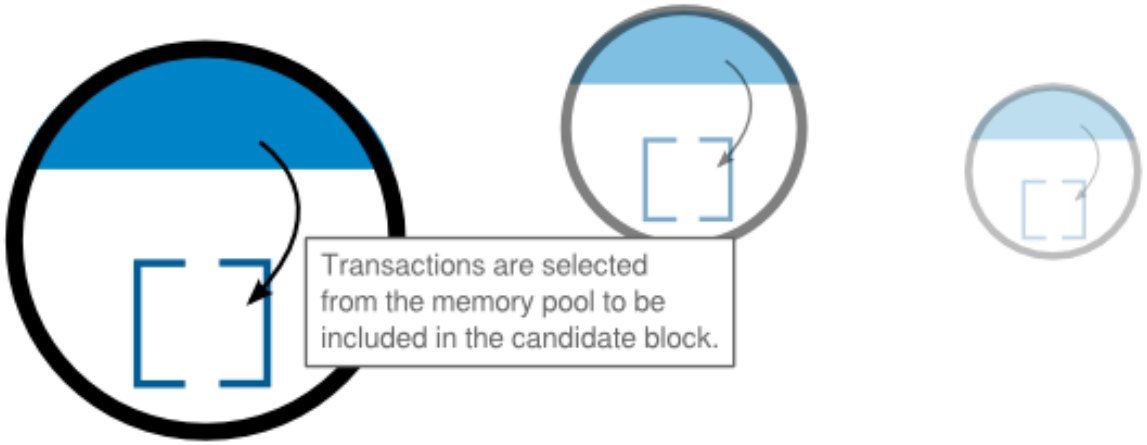
Candidate block

- If you are a miner/validator, your job is *to gather transactions* *from the transaction pool* in to a ***"candidate block"***, and to *try* and add this candidate block to the blockchain.



Candidate block

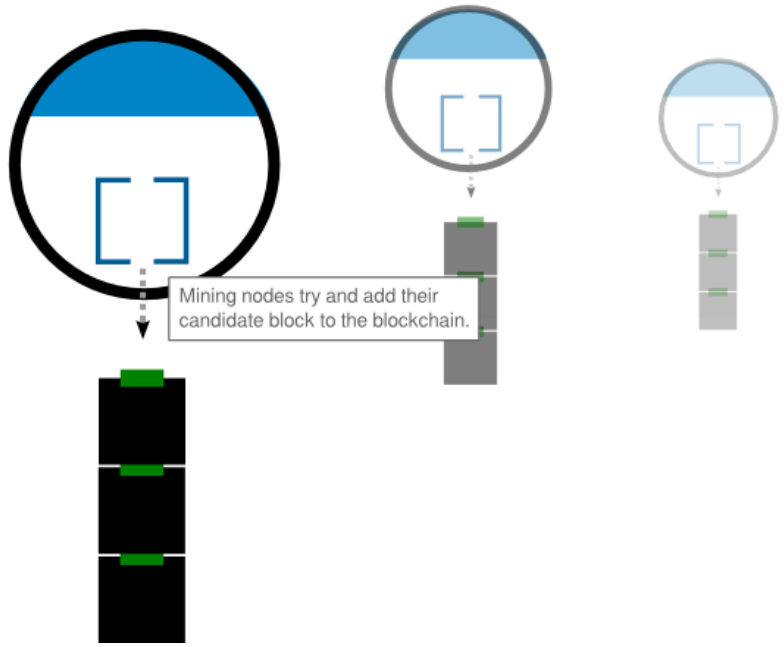
- **A candidate block is a temporary block created using transactions selected from the memory pool.**
- Nodes (mining nodes) select transactions from their memory pool to form their own candidate blocks.
- Anyone who is mining can choose which transactions to include in their candidate block.



UIT Information Security Lab

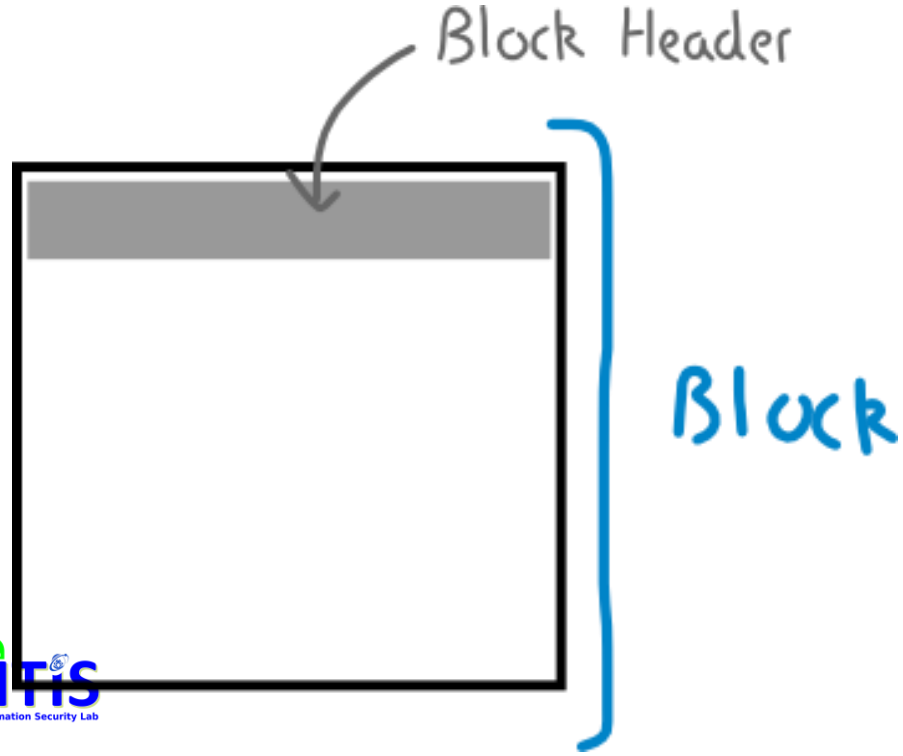
Candidate block

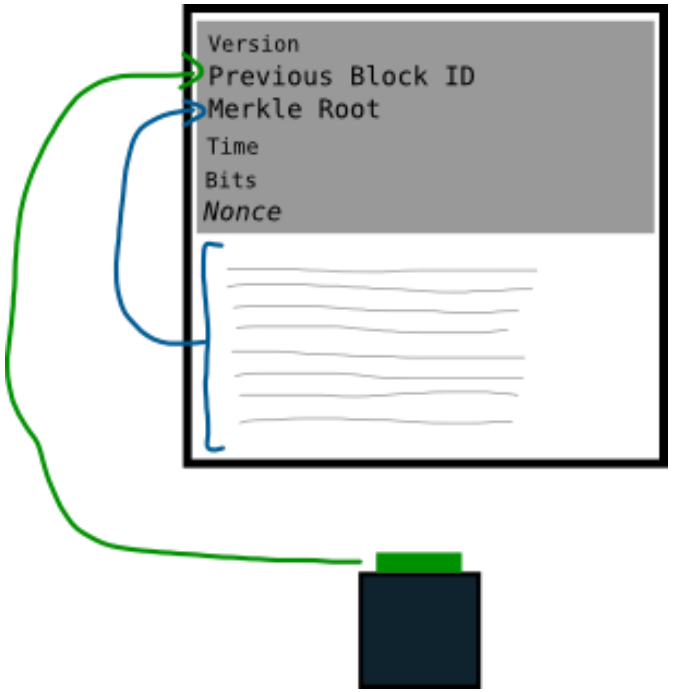
- Each node then tries to add their ***candidate block*** to the ***blockchain*** through the process of **mining.**
- Every block starts life as a candidate block, but only the ones that are successfully mined get added to the blockchain.
- Each miner starts working on a new candidate block, as soon as they hear about the latest one being found.



Block Header

- A block header is like the *metadata* at the top of a block of transactions.
- The fields in the block header provide a unique summary of the entire block.

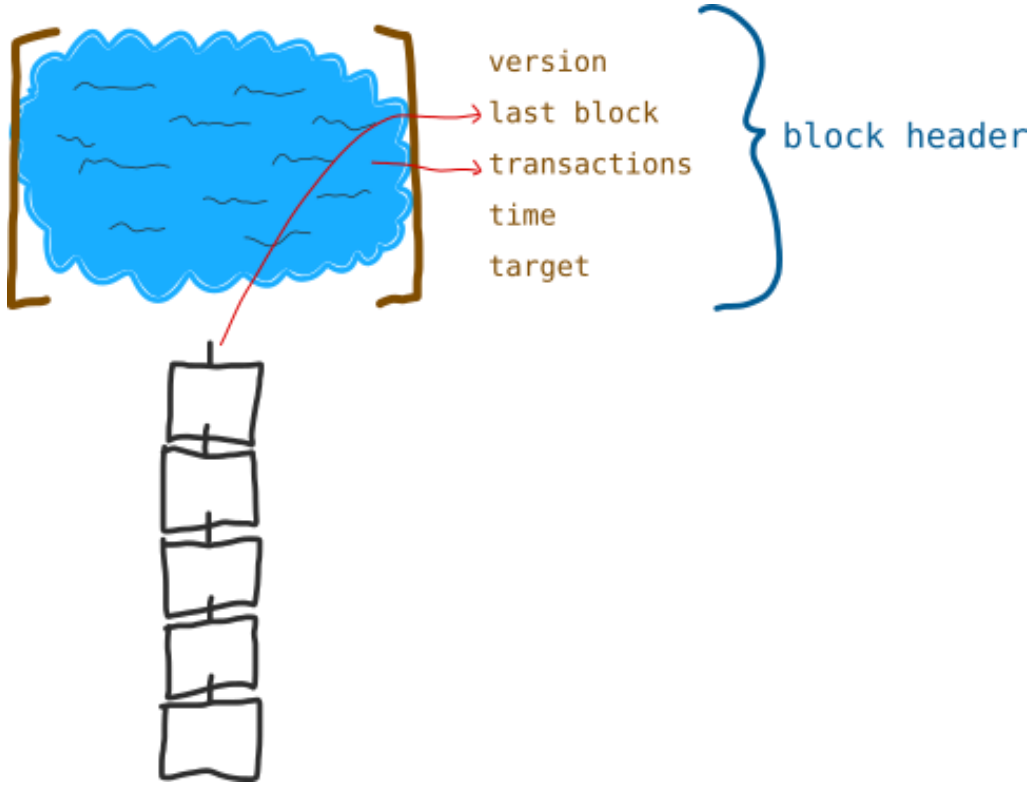




UIT Information Security Lab

Block header

- You also can give each candidate block a **BLOCK HEADER**, which is basically a bunch of *metadata* about the block.
- Miners use this metadata when trying to add a block to the blockchain.
- metadata – n. data that describes other data, serving as an informative label.

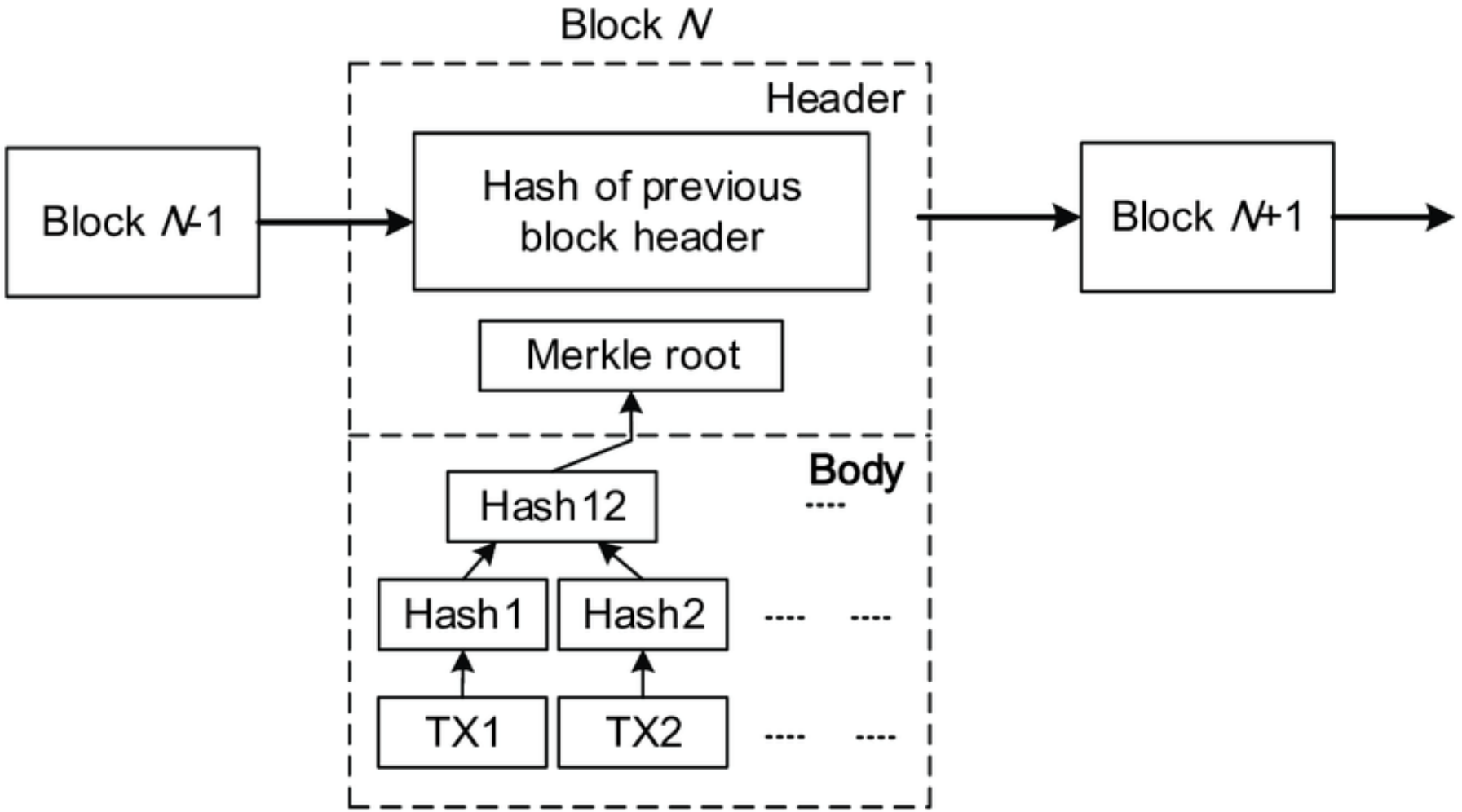


UIT Information Security Lab

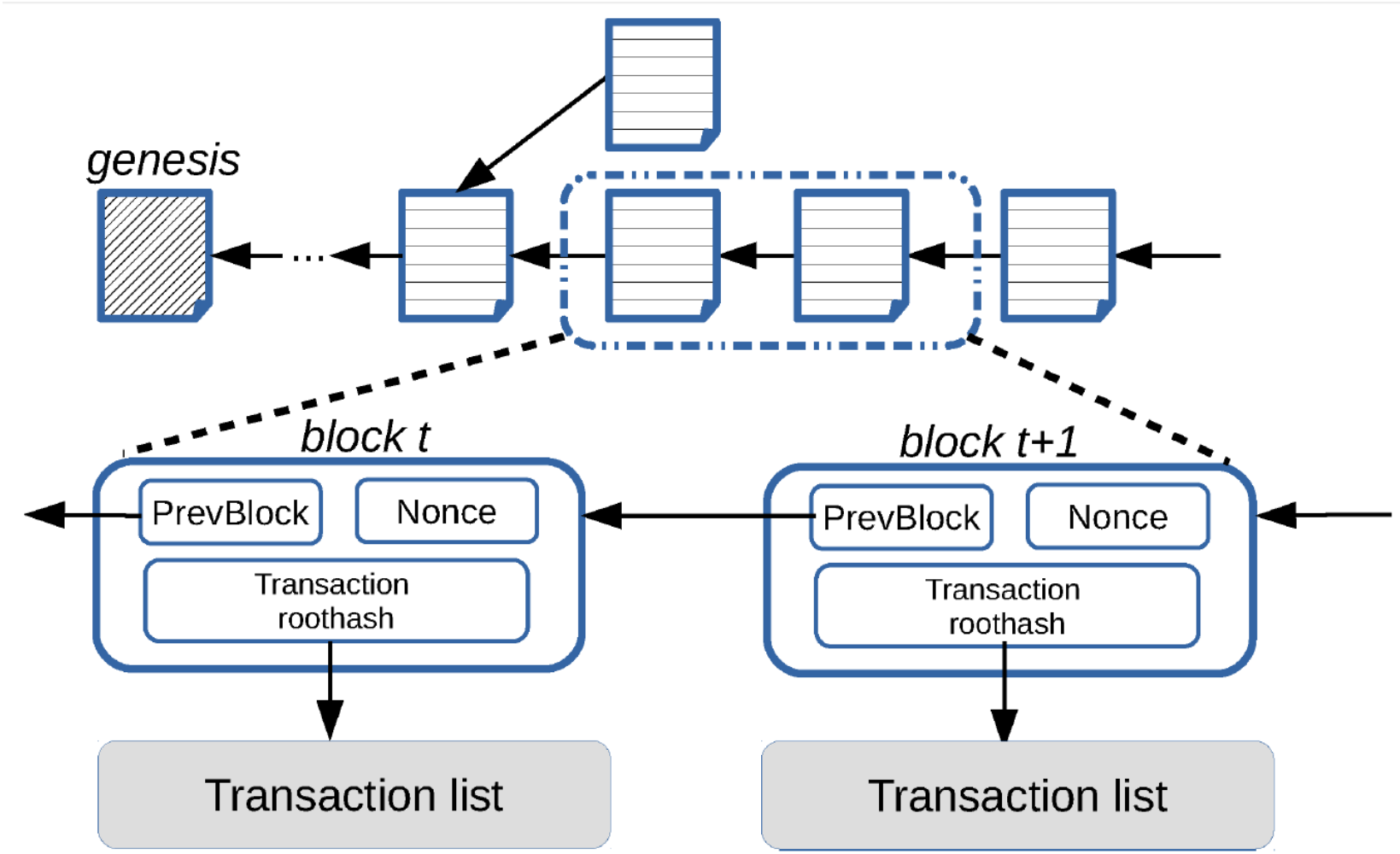
Block header fields

- **Version**: Describes the structure of the data inside the block. This is used so that computers can read the contents of each block correctly.
- **Last Block:** An identification number for the previous block. We are trying to get one of these for the current candidate block.
- **Merkle Root (Transactions):** All of the transactions inside the block hashed together to form a single line of text. All of the fields are unique, but it would be fair to think of this as the most significant part of the block header.
- **Time:** The current time. Always handy.
- **Target:** A value that miners work with to try and add candidate block to the blockchain. It is set by the bitcoin network, and will make more sense in a moment.

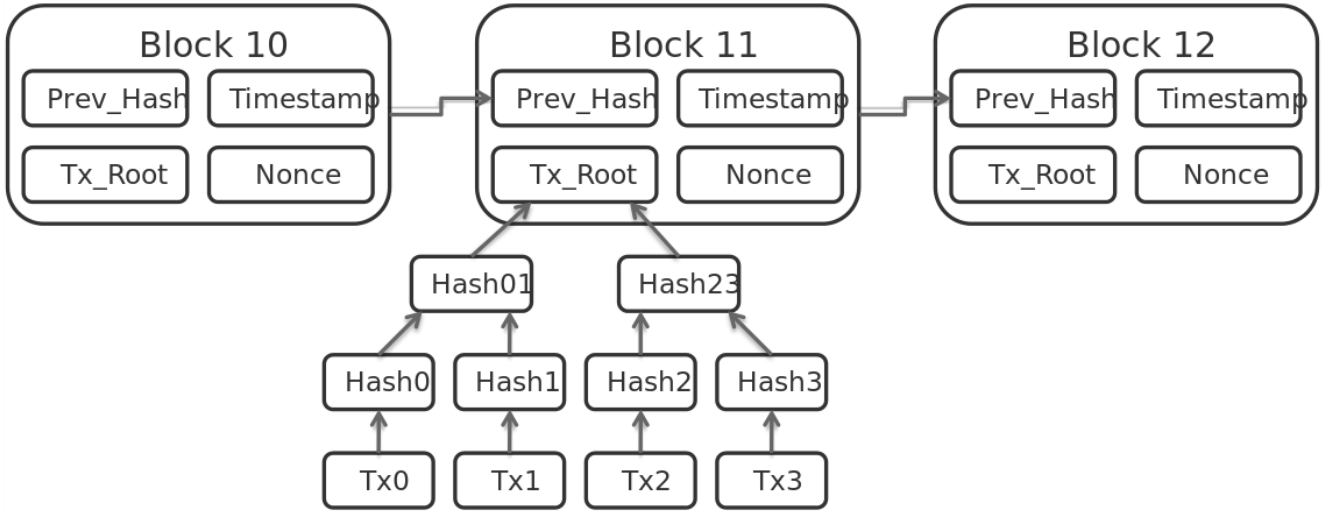
Structure of Block



Structure of Block

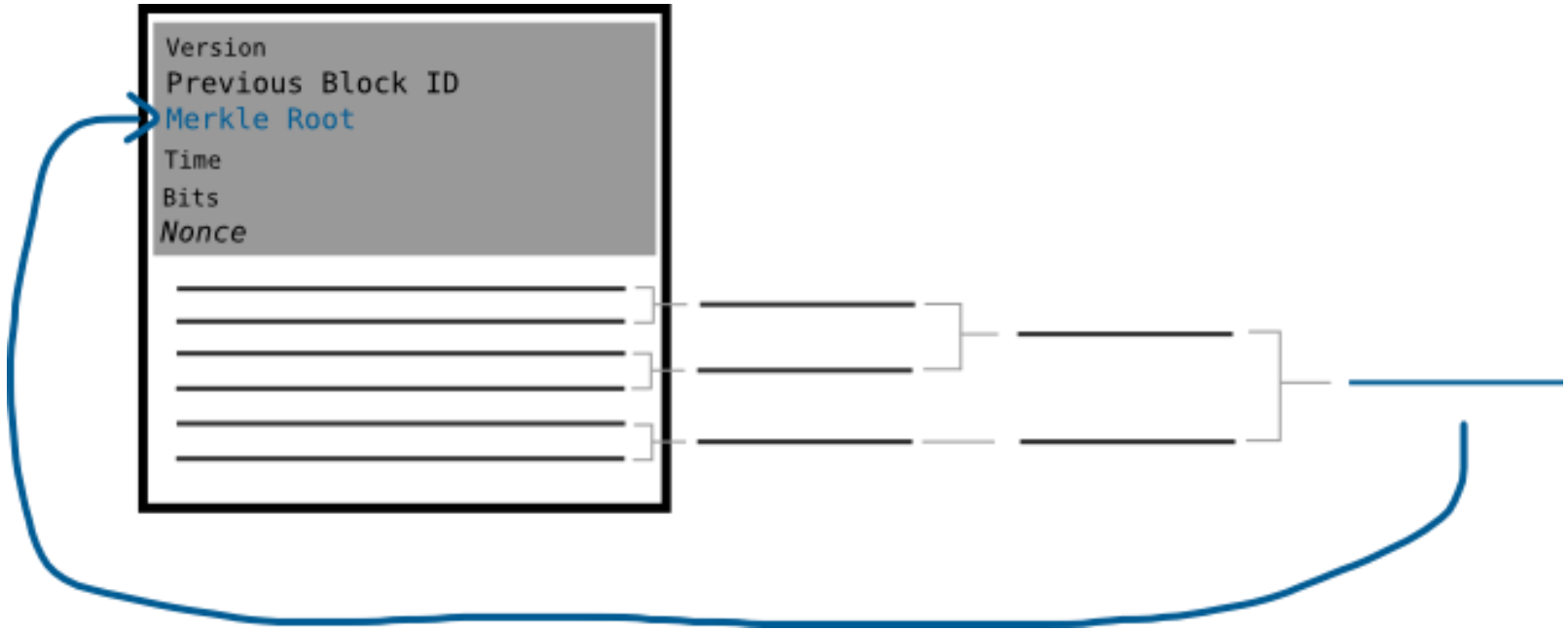


Structure of Block



Merkle Root

- The Merkle Root is a field in the block header.
- The “Merkle root" acts as a representation of every transaction included in the block.

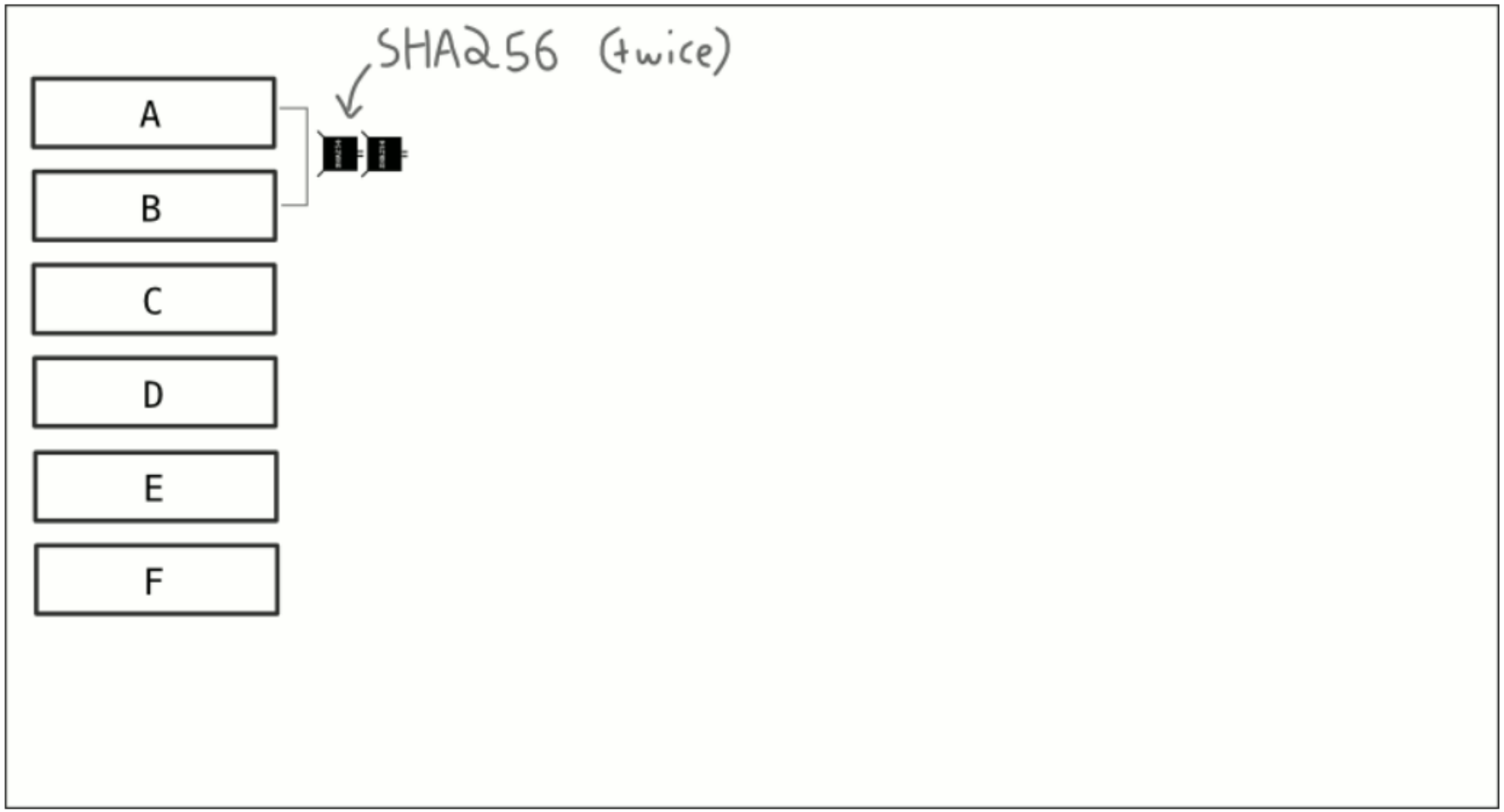


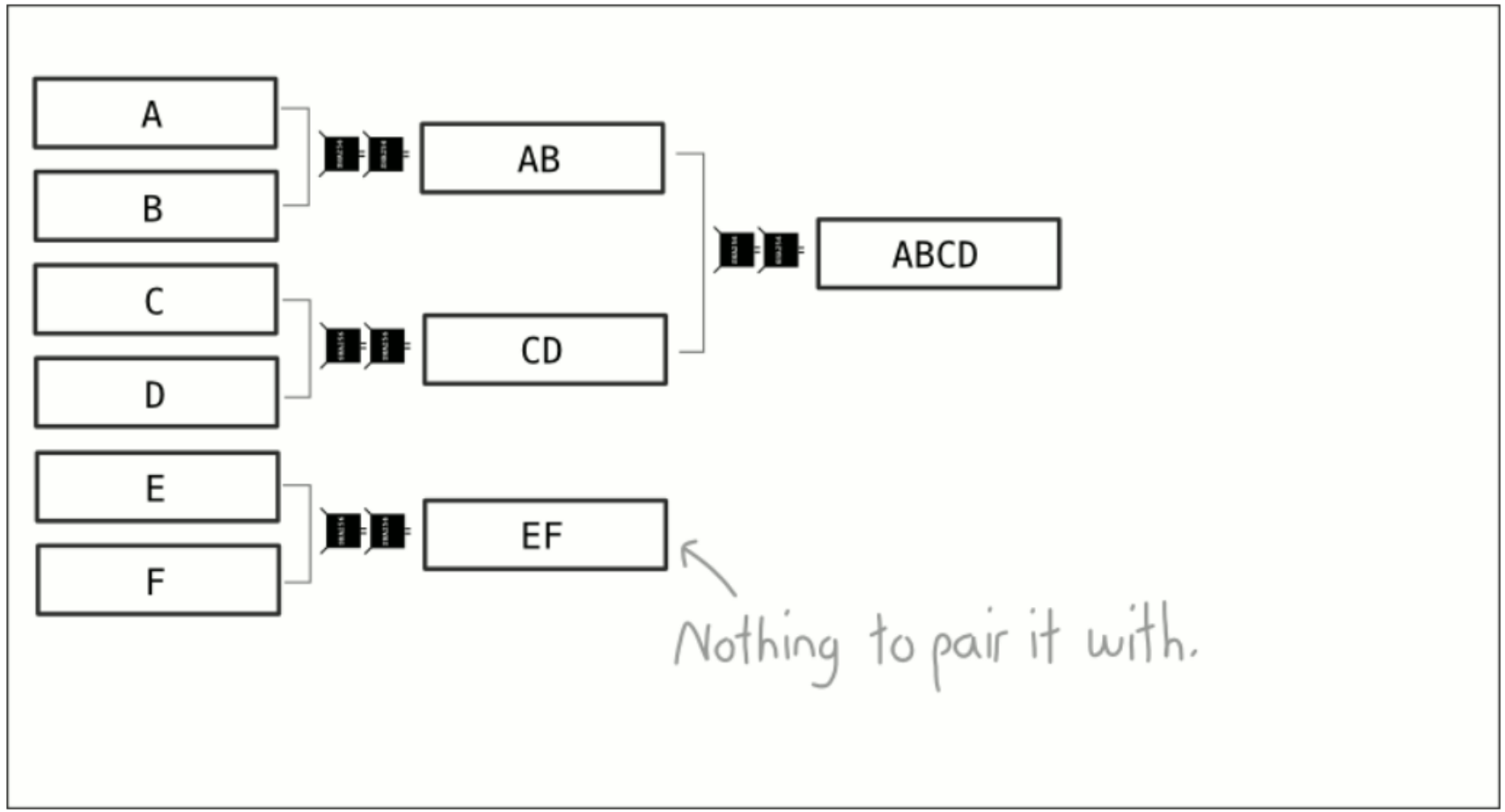
UIT Information Security Lab

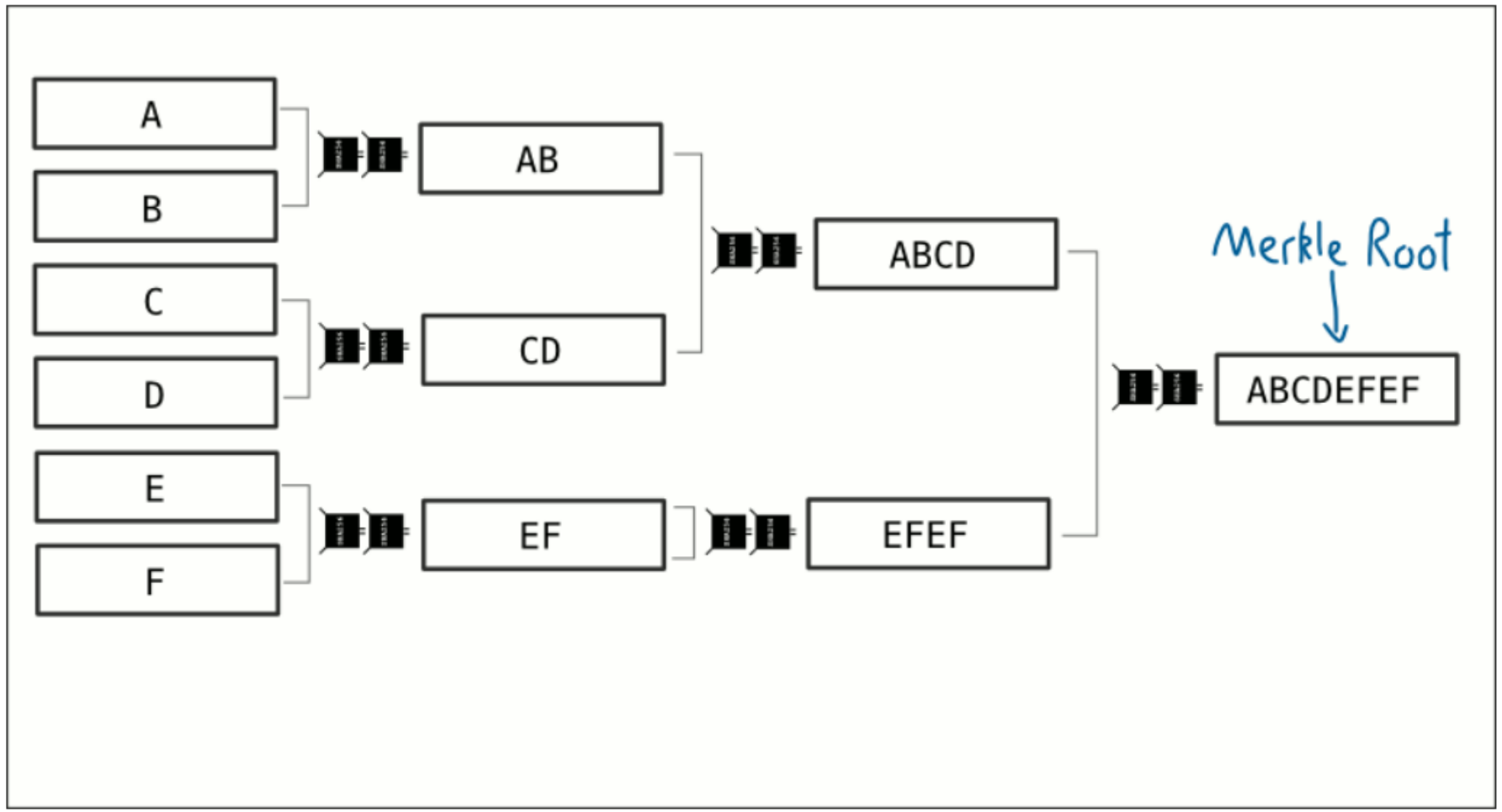
How do you get a Merkle root?

- By repeatedly hashing together pairs of Transaction IDs until you end up with a single hash as a result.
- **Steps:**
  - Take each pair of Transaction IDs from the block, and hash them together through SHA256 *twice*.
  - Keep doing this for each pair Transaction IDs, until you end up with a new list of hashes.
    - *Note: If you have an odd number of transactions, hash the remaining transaction with itself.*
  - Repeat steps 1-2 for every new list of hashes you create until you finally end up with one hash.


A
B
C
D
E
F
Transaction IDs

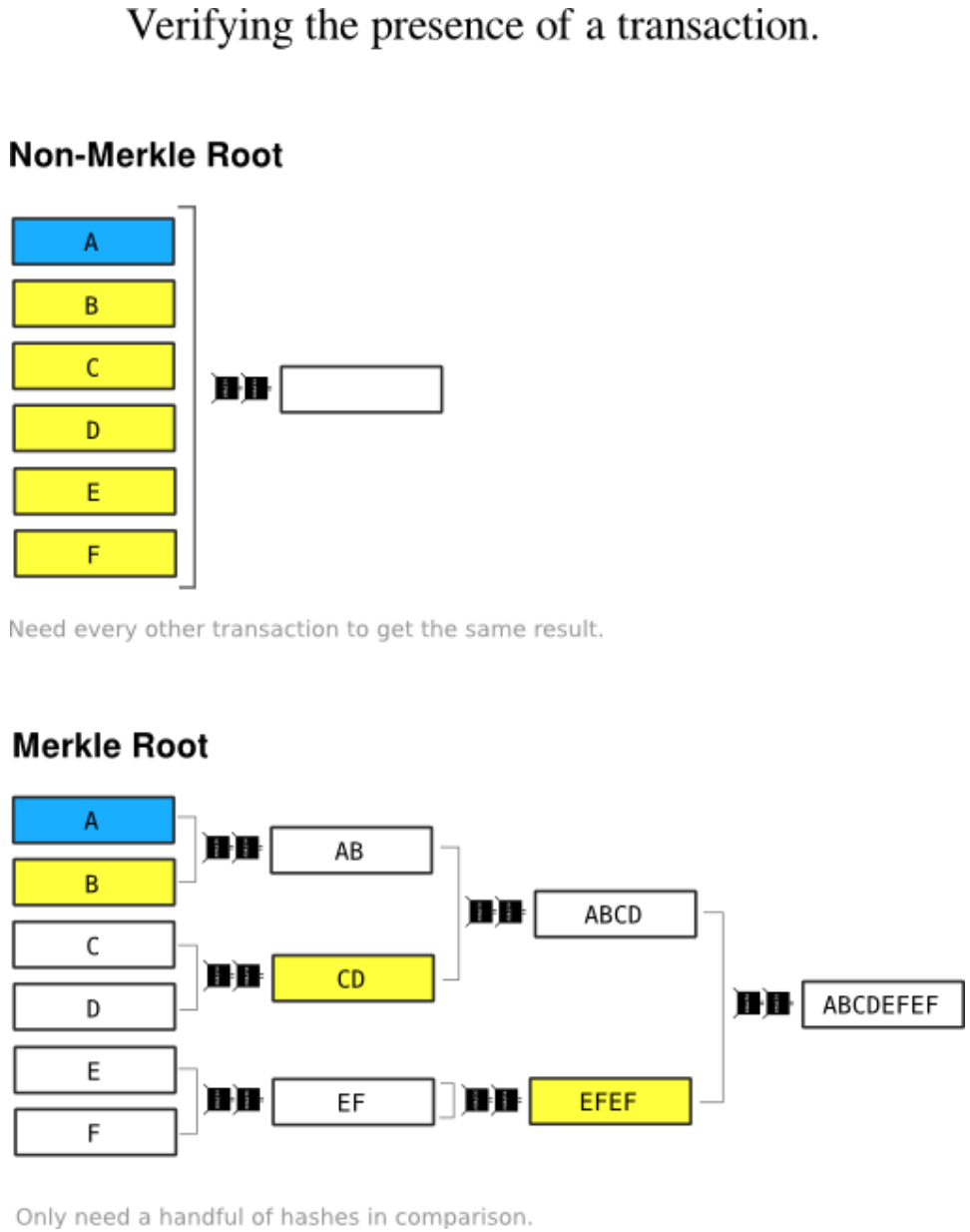

SHA256 (twice)
A
B
C
D
E
F


A
B
C
D
E
F
AB
CD
EF
ABCD
Nothing to pair it with.


A
B
C
D
E
F
AB
CD
EF
ABCD
EFEF
Merkle Root
ABCDEFEF

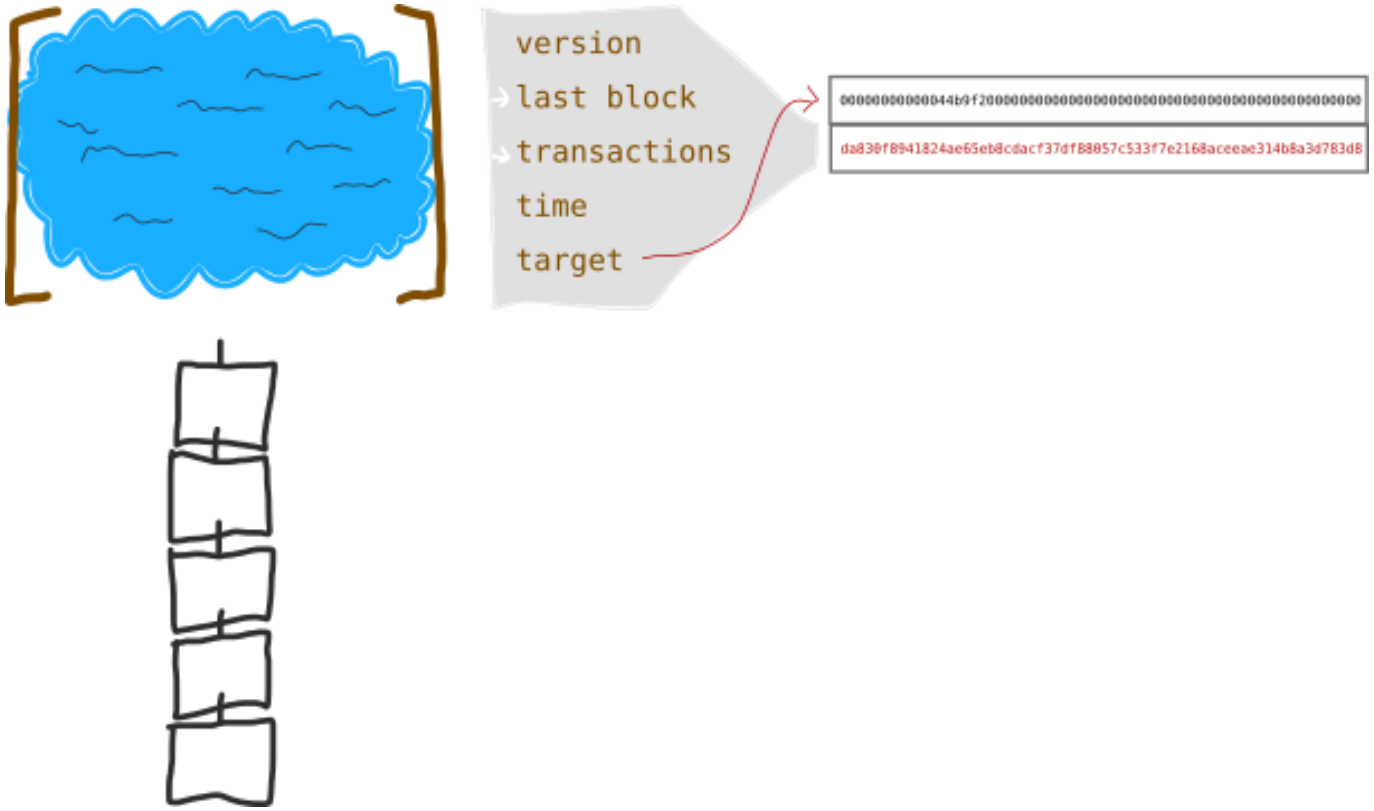
Why does bitcoin use the Merkle root method?

- Because if you want to check that a transaction is part of the final hash, a Merkle root is a more efficient way of doing it.



How are blocks added to the blockchain?

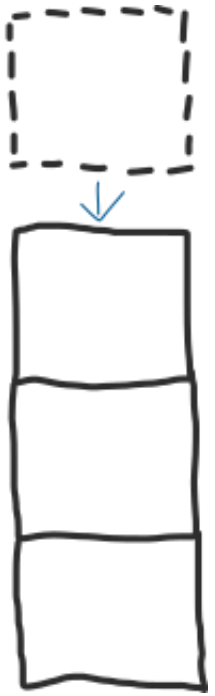
- To add a candidate block to the blockchain, you **hash the data in the block header** and hope that the result is *below a certain target value*.
- The **TARGET** is calculated from the **DIFFICULTY**, which is a value set by the network to regulate how difficult it is to add a block of transactions to the blockchain.



Difficulty and Target

difficulty

- The **difficulty** is *a **number*** that regulates ***how long it takes for miners to add new blocks*** of transactions ***to the blockchain***.
- This **difficulty value** updates ***every 2 weeks*** to ensure that it takes 10 minutes (on average) *to add a new block* to the blockchain.
- *Why is the difficulty important?*
  - Because it ensures that blocks of transactions are added to the blockchain at regular intervals, even as more miners join the network.
  - If the difficulty remained the same, it would take less time between adding new blocks to the blockchain as new miners join the network.



When does the difficulty change?

- The difficulty ***adjusts every 2016 blocks*** (roughly every 2 weeks).
- At this interval, each node takes the expected time for these 2016 blocks to be mined (2016 x 10 minutes), and divides it by the actual time it took (however many minutes):

  - $A = \frac{expected}{actual} = \frac{20160}{actual} = \frac{20160}{18144} = 1.11$

  (If miners were able to solve each block more quickly than expected; say *9 minutes per block* for example, you'd get a number like this)

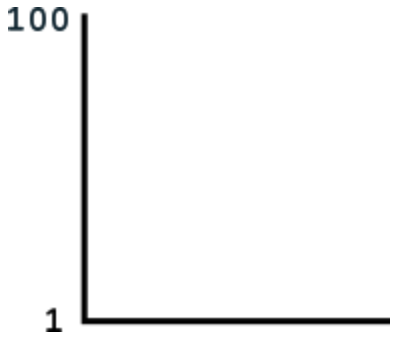
- Each node then uses this number to *adjust the difficulty* for *the next 2016 blocks*:

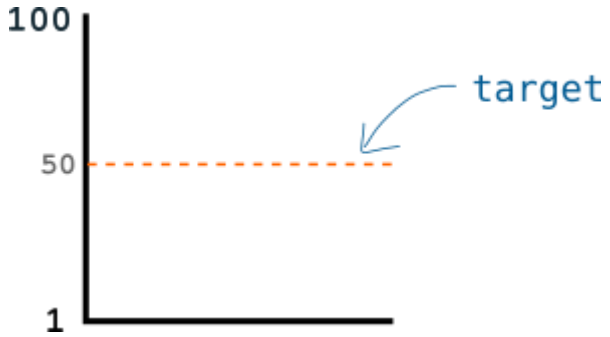
$$new\_difficulty = A * difficulty = 1.11 * difficulty$$

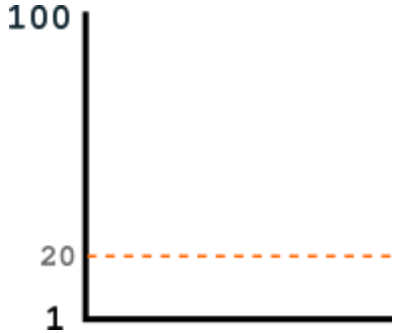
  - If $new\_difficulty$ >1 (i.e. blocks were mined quicker than expected), the difficulty increases.
  - If $new\_difficulty$ <1 (i.e. blocks were mined slower than expected) the difficulty decreases.
  - Every miner on the bitcoin network *now works with this new difficulty* for *the next 2016 blocks.*
  - 0.25<=A(of difficulty)<= 4, *prevent abrupt changes from one difficulty to the next.*

UITIS
UIT Information Security Lab

How does the difficulty control time between blocks?





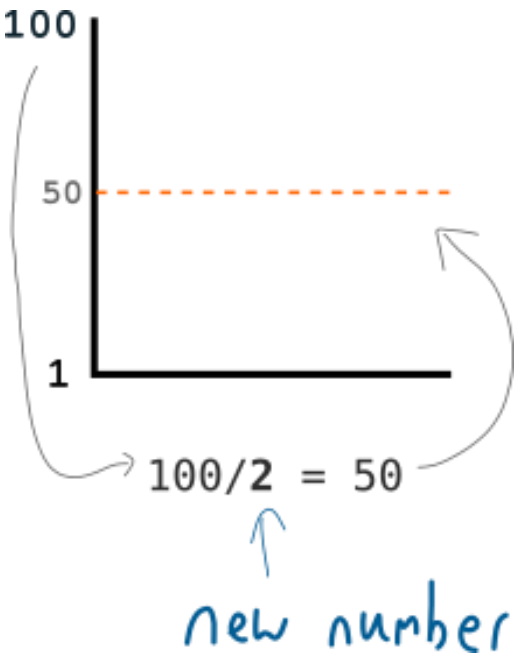


- Let's say I give you a range of numbers from 1 to 100.
- Now, you are able to randomly generate a number between 1 and 100 once every minute. And your goal is to generate a number ***below my target number***.
- *Target = 50,* you're only able to *generate a number* between 1 and 100 *once a minute*, this should *take you 2 minutes*
- Target = 20, you're only going to be able to generate a winning number 1/5 of the time, or once every 5 minutes
- The lower the target, the more difficult it gets to generate a winning number.
- It's not going to be 5 minutes every time because you could get lucky with the first number you generate. But *over the long run* it will work out to be 5-minute intervals
- Therefore, based on how many numbers you are able to generate per minute, I can use the height of the target to control how long it takes you to find a winning number.

UITIS
UIT Information Security Lab

Introducing the difficulty…

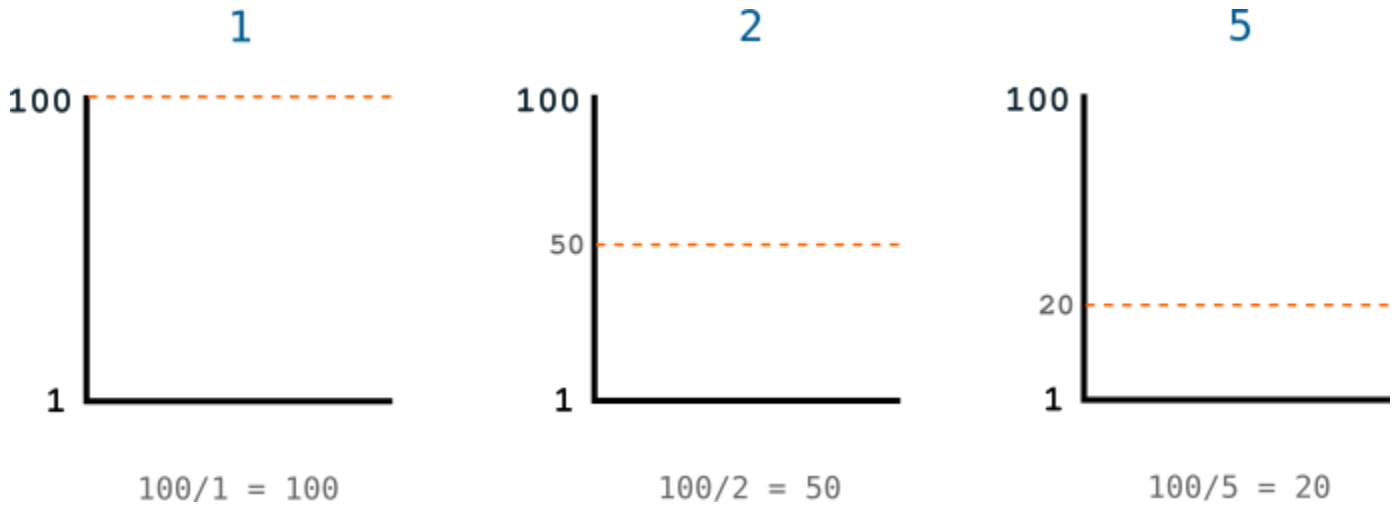
- Set the target by **dividing the range of numbers** **with** ***a new number***… This new number is able to control the height of the target.
- This **new number** is the difficulty, and it’s used as an easy way for me to modify the height of the target.



UITIS
UIT Information Security Lab

Introducing the difficulty…

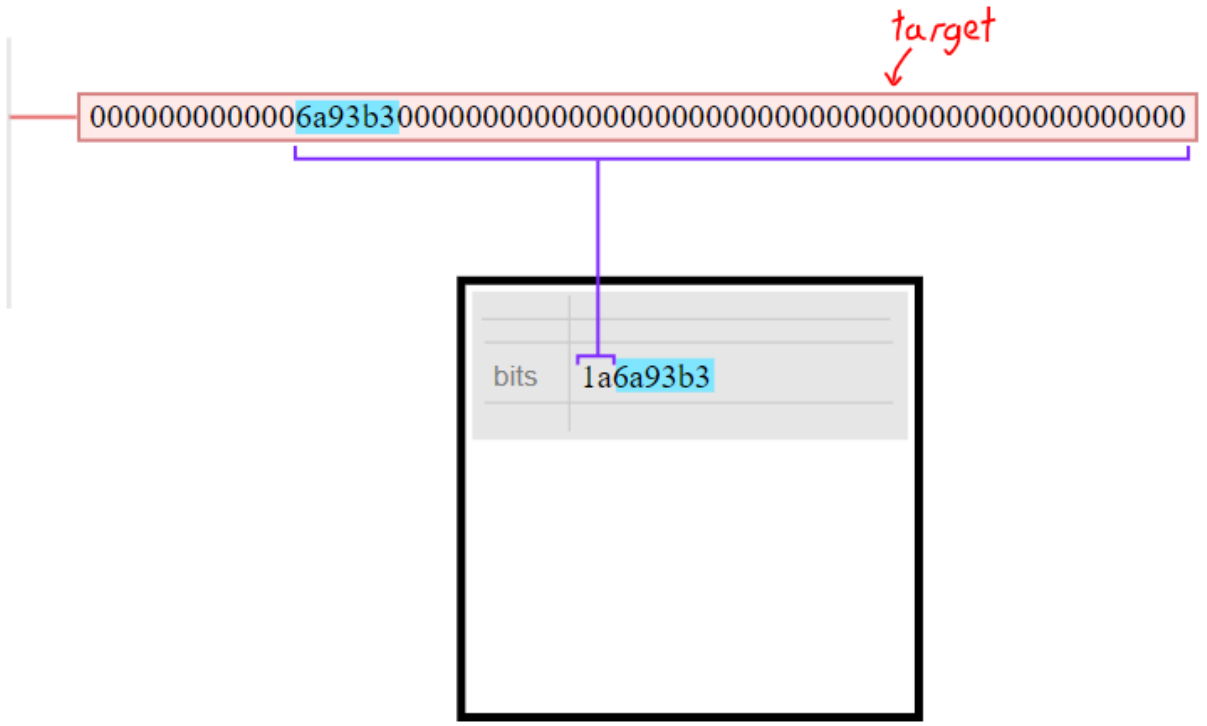
- $target = \frac{targetmax}{difficulty}$
- Furthermore, I can use this **DIFFICULTY** value to help me set the **target** to any level I want.
- So I use the DIFFICULTY to control the TARGET, and therefore how long it takes for you to generate a winning number.



UITIS
UIT Information Security Lab

BITS (AKA TARGET)

- The *bits* field is a compact way of storing the target in the block header.



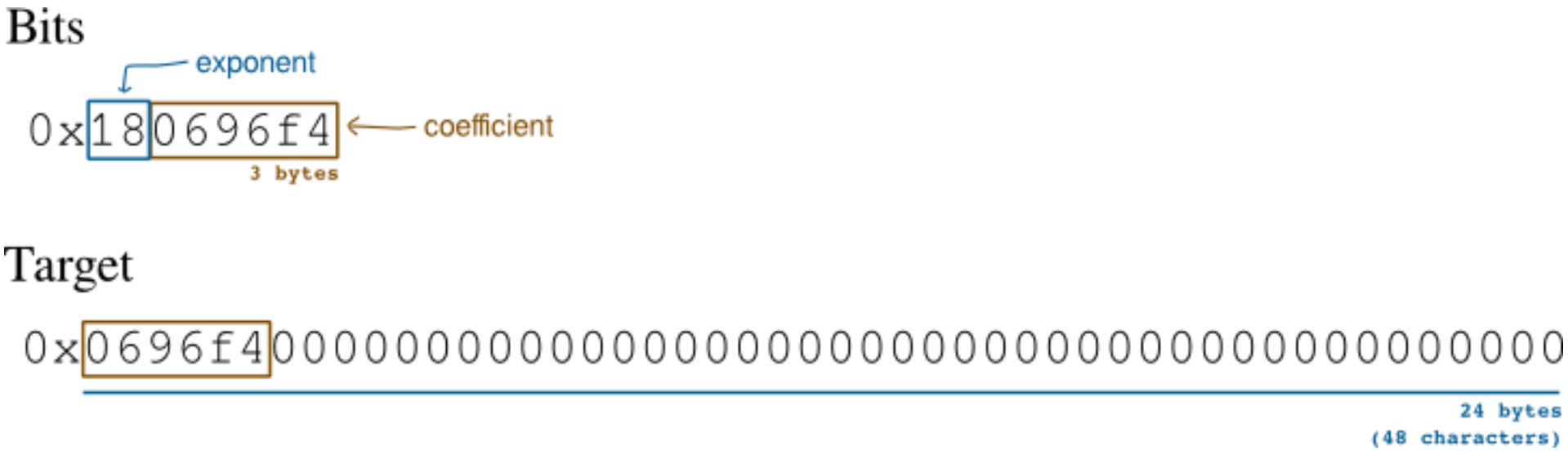
Target: Bitcoin Example

- Here's what the target was when Block 406,800 was mined. Target: 0x00000000000000000696f4000000000000000000000000000000000000000000
- And this is the Bits field you'll find in the block header for it:

  Bits: 0x180696f4

Target: Bitcoin Example - Converting Bits

- It's basically split in to two parts:
  - Exponent: This gives you the size of the target in bytes.
  - Coefficient: This gives you the initial 3 bytes of the target



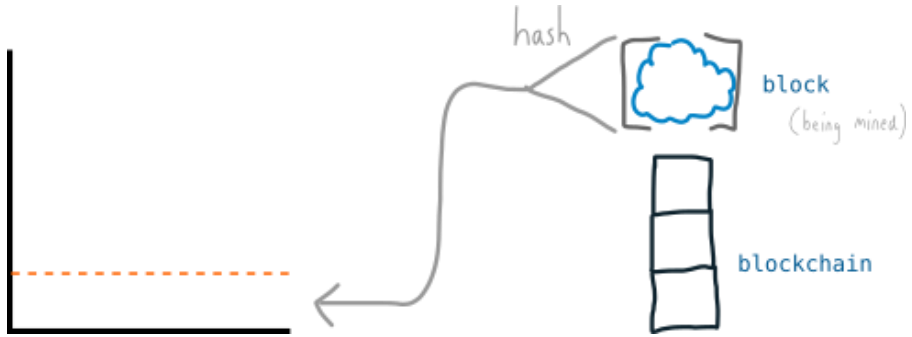
UITIS
UIT Information Security Lab

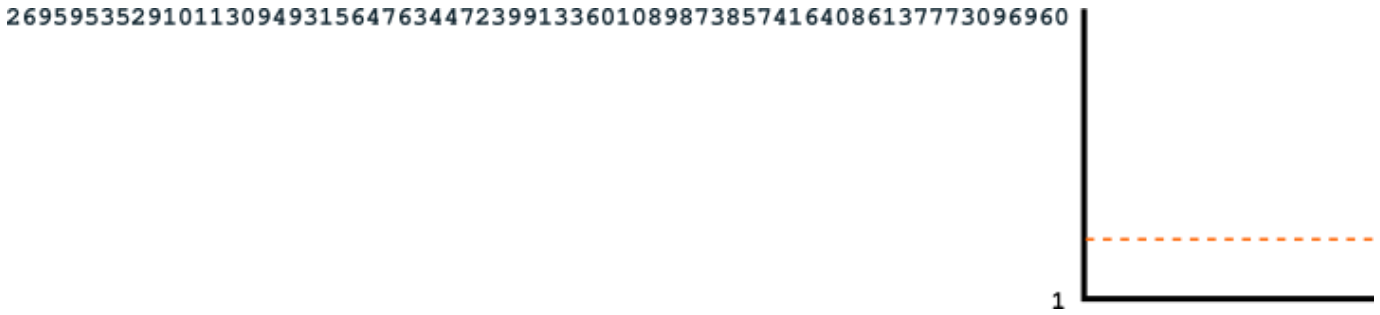
Why have "Bits"?

- **Question**: Why ever would you convert the Target in to "Bits"? Why not just store the full Target in the block header?
- **Answer**: Because the block header doesn't need to store the absolute precision of the full Target, so the Bits format saves on space.

Difficulty: Bitcoin example

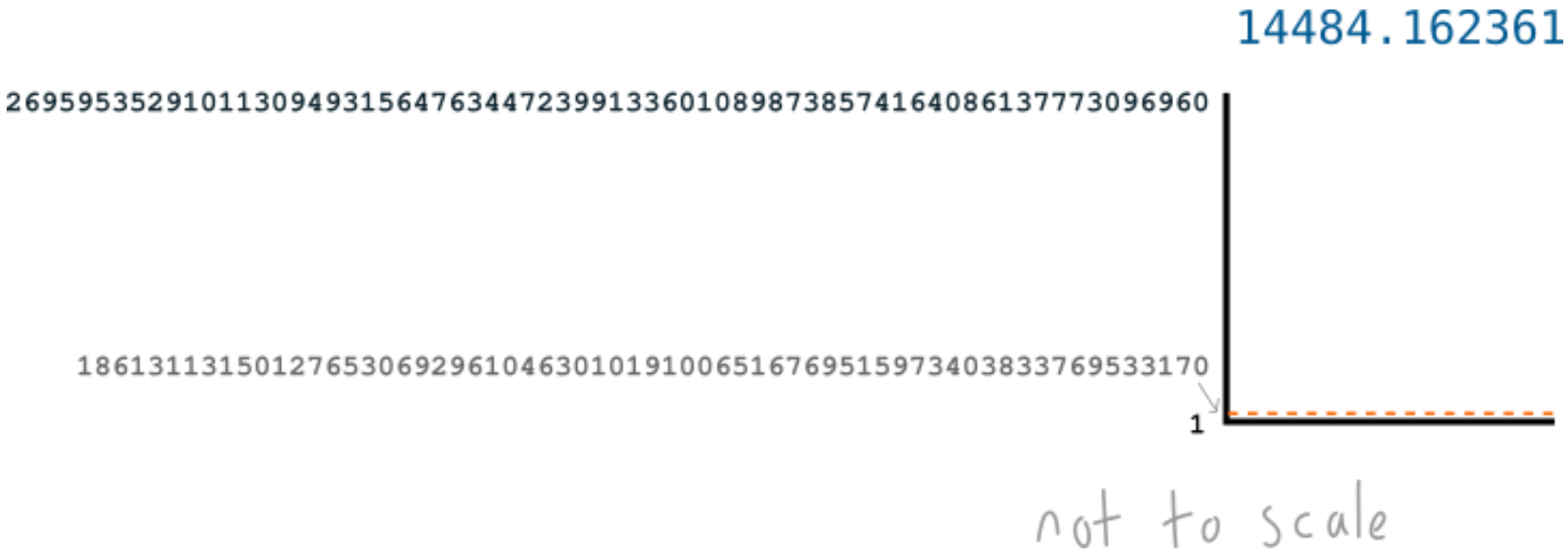
- It’s used to set a target value, and miners keep generating numbers (hashing their candidate blocks) in the hope that they will find a number lower than this target value.
- *Miners are able to generate thousands of numbers* (**hash values**) per minute, bitcoin uses ridiculously big numbers.
- The numbers in bitcoin are just on a much bigger scale





UIT Information Security Lab

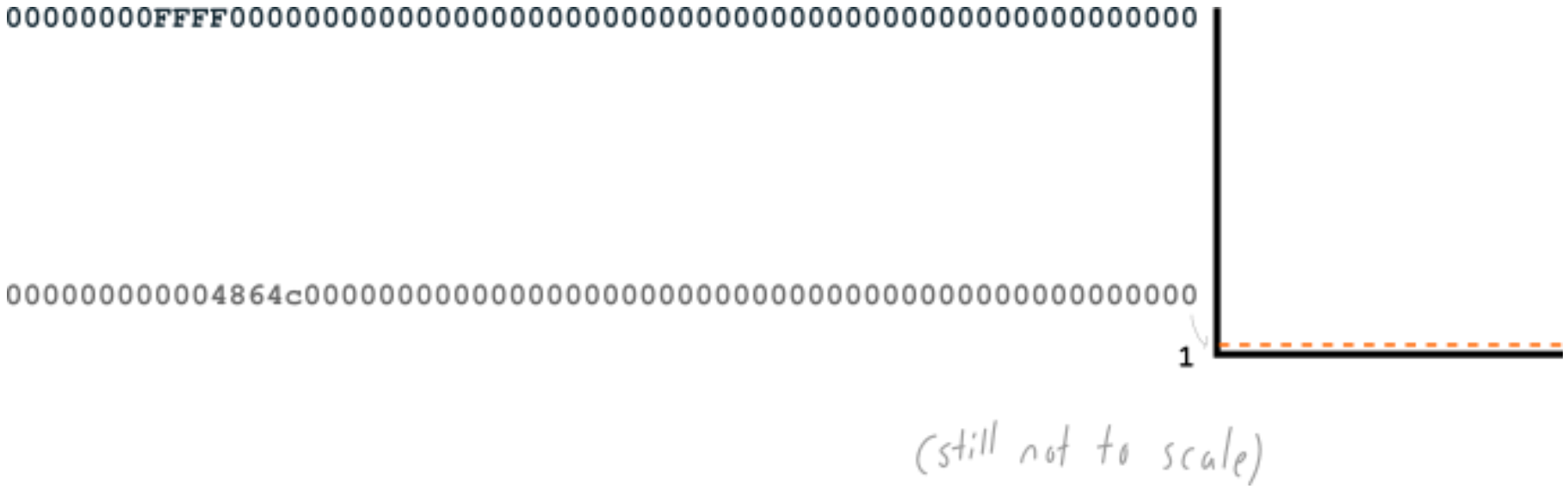
Difficulty: Bitcoin example



- There are now thousands of miners trying to find winning numbers, to ensure that a winning number is found every 10 minutes (instead of every few seconds), the range of successful numbers ends up being absolutely **tiny**.
- Even though that difficulty number looks big, the target is still absurdly difficult to get under. It's like a lottery.

UITIS
UIT Information Security Lab

Difficulty: Bitcoin example



- Because these target numbers are so big, computers prefer to work with them in hexadecimal format.
- So the target is a hexadecimal value, and miners are *trying to get a **hexadecimal hash value** below the target*.

UITIS
UIT Information Security Lab

Difficulty: Bitcoin example

Target for block 100,000	
Hexadecimal	00000000000004864c000000000000000000000000000000000000000000000000
Decimal	186131131498380012681564362292723007636833484581425336990197350 4

Hash for block 100,000	
Hexadecimal	000000000003ba27aa200b1cecaad478d2b00432346c3f1f3986da1afd33e506
Decimal	153326787264777690215432048793065921179506558199844584874022631 0

UITiS
UIT Information Security Lab

Difficulty: Bitcoin example

Block #100000

BlockHash 000000000003ba27aa200b1cecaad478d2b00432346c3f1f3986da1afd33e506

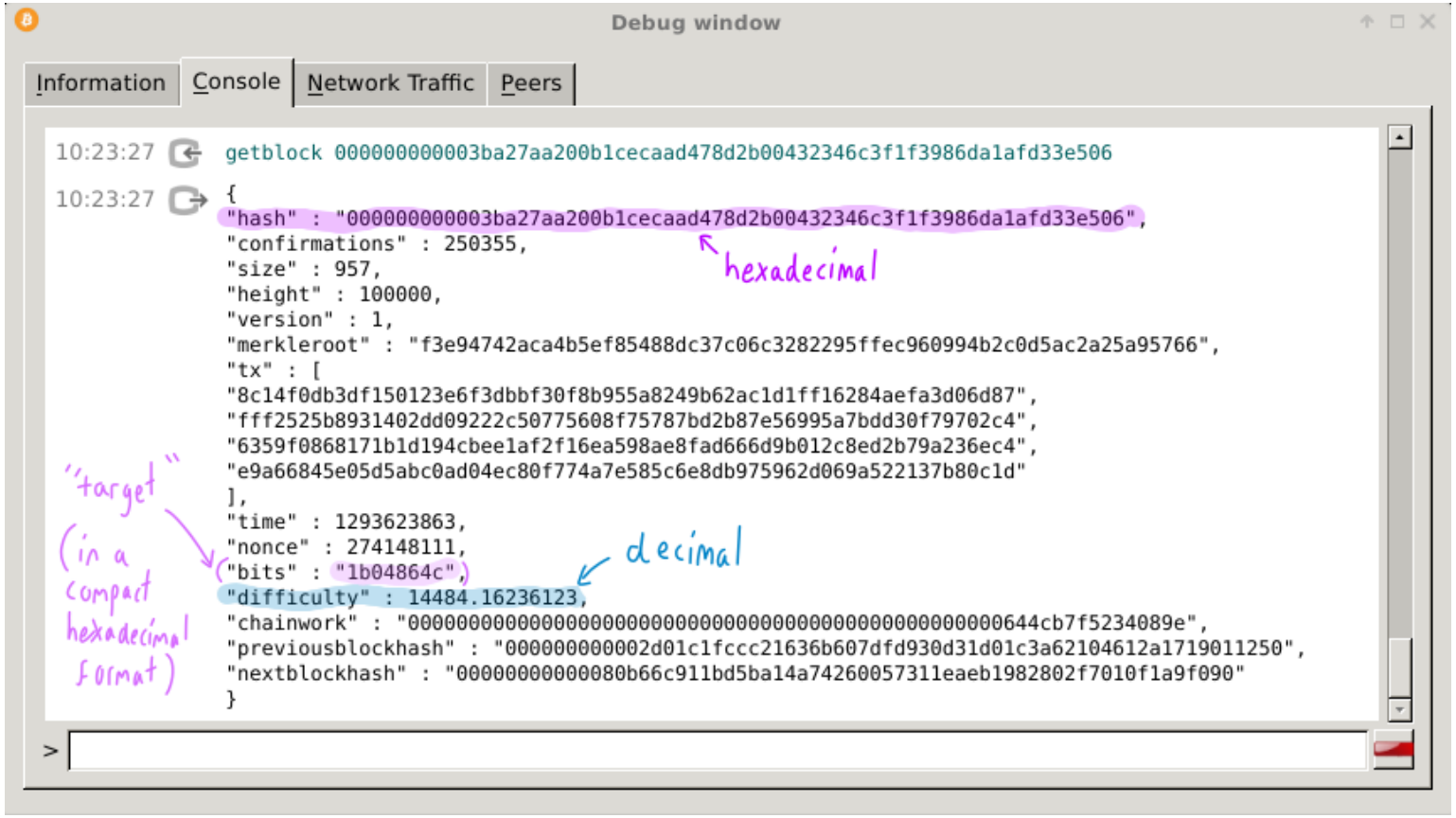
Summary

Number Of Transactions	4	**Difficulty**	14484.16236122
Height	100000 (Mainchain)	**Bits**	1b04864c
Block Reward	50 BTC	**Size (bytes)**	957
Timestamp	Dec 29, 2010 6:57:43 PM	**Version**	1
Mined by		**Nonce**	274148111
Merkle Root	f3e94742aca4b5ef85488dc37c06c3...	**Next Block**	100001
Previous Block	99999		

Link:
https://blockexplorer.com/block/000000000003ba27aa200b1cecaad478d2b00432346c3f1f3986da1afd33e506

UITiS
UIT Information Security Lab

Difficulty: Bitcoin example

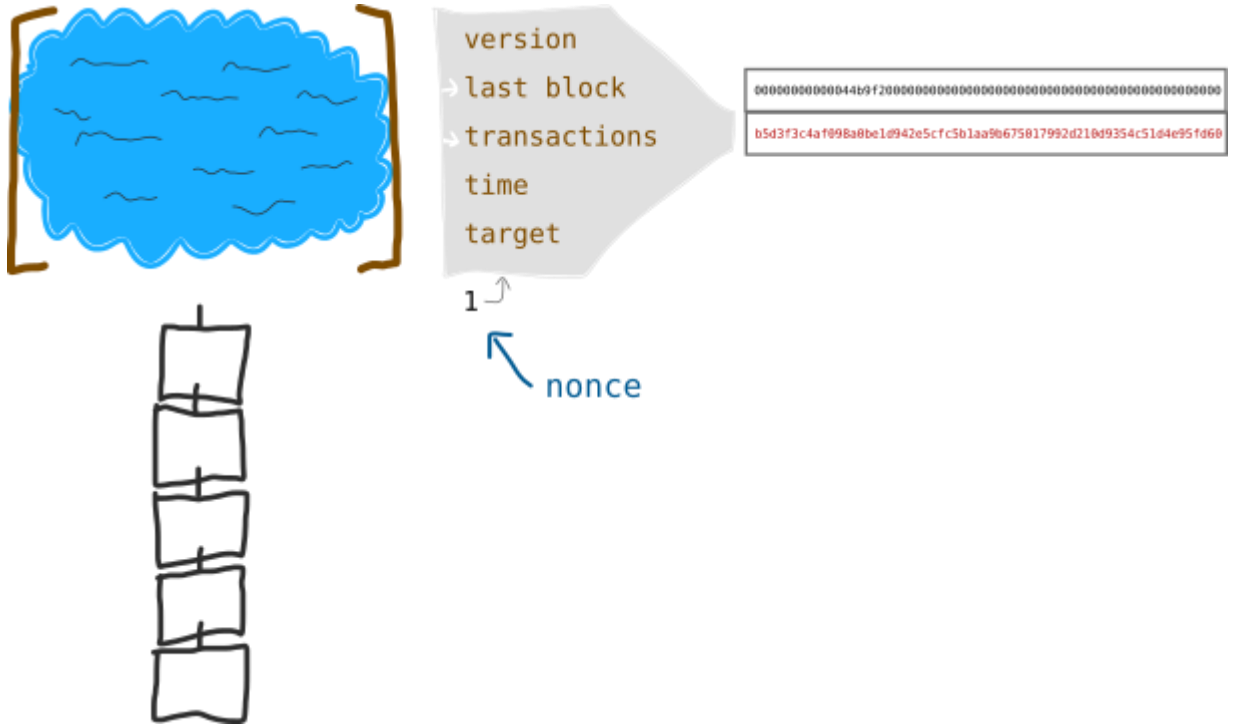


- Block 100,000 header:

The target is hexadecimal, but it is stored in a *compact* format in the block header (called **BITS**).

The Nonce

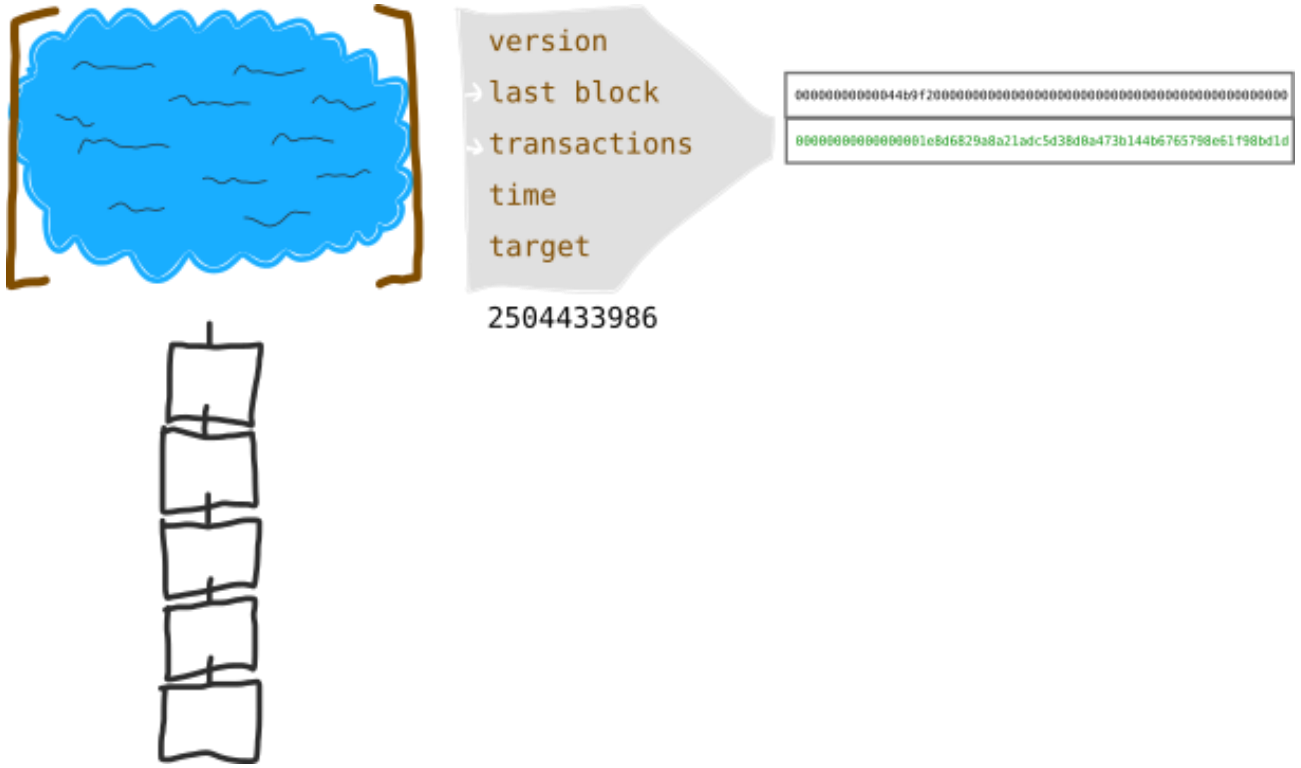
- You actually **hash** it *with an extra number*.
- This number is called a **nonce**, and it's basically a dummy field that miners use to help them get a block hash below the target value.
- **nonce** – an ***arbitrary number*** used **only once** in a cryptographic communication.



UITIS
UIT Information Security Lab

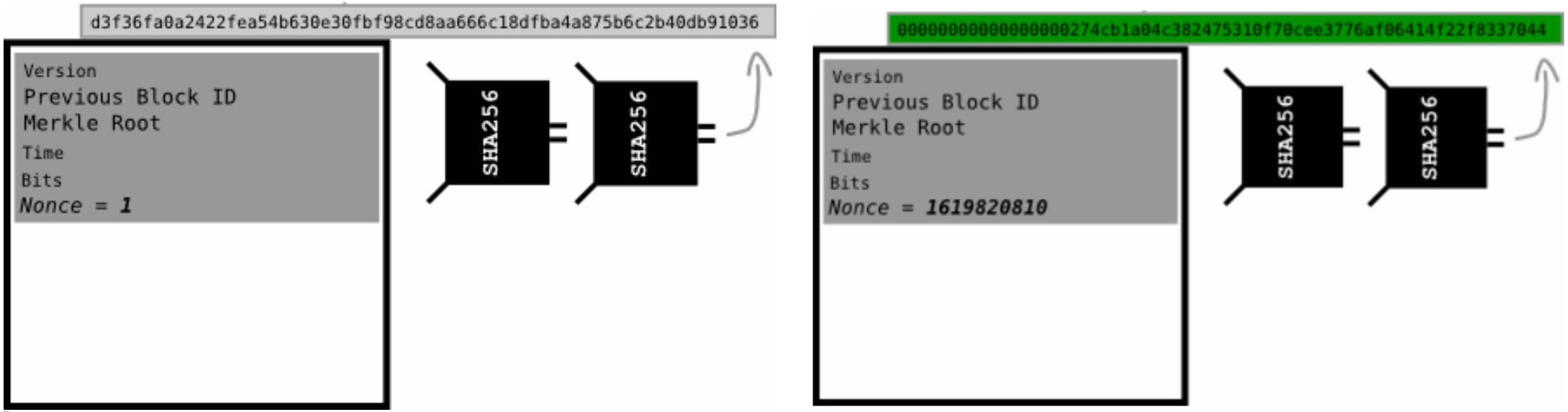
The Nonce

- If the *first nonce doesn't work* (starting at 0), keep incrementing it and hashing the block header.
- Eventually you'll **find a nonce** that **returns a block hash that is less than the target value**.



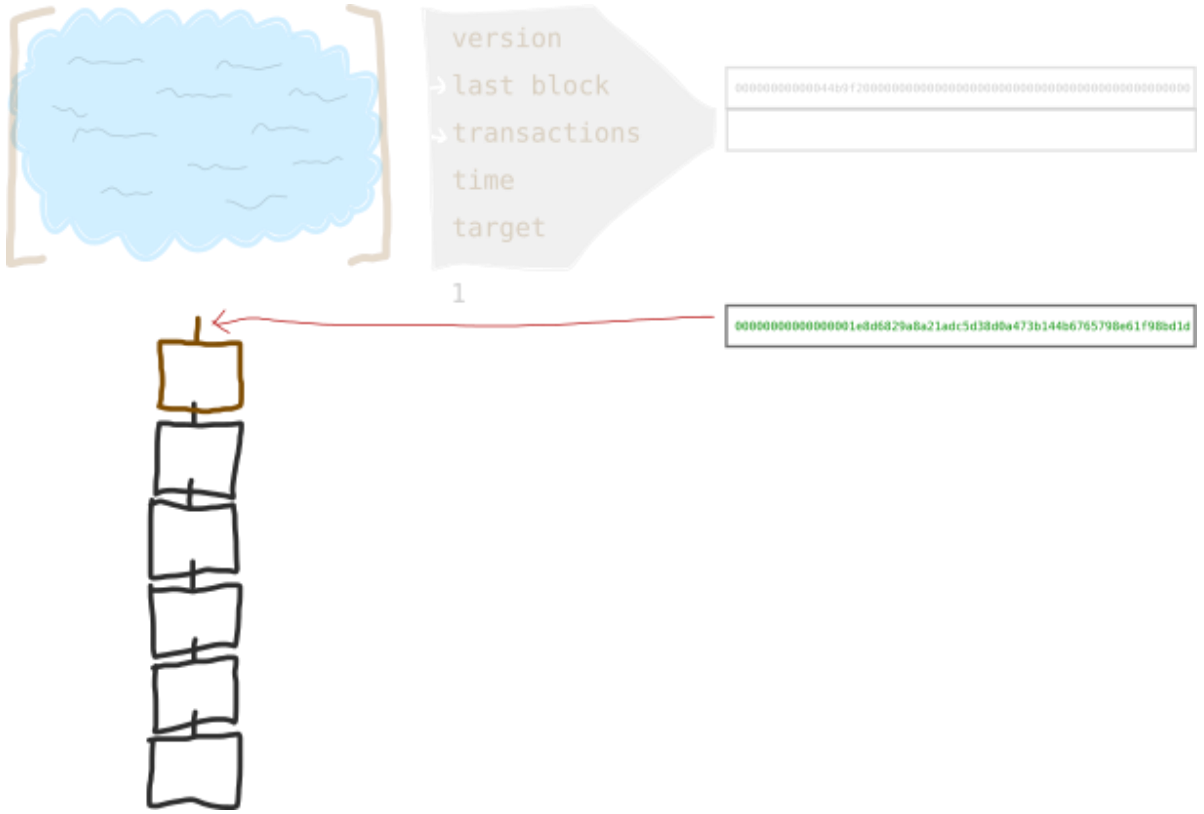
Nonce: What's it used for?

- The **nonce** is a field in the block header. I call it "the mining field".
- Miners adjust the nonce when they're trying to get the hash of their *block headers* below the target. (i.e. Get a valid Block Hash for their candidate block.)



Congratulations

- Once you've **found nonce** that works, the block is "solved" and *all of the transactions in this block* are ***added to the blockchain***.
- All miners will now head back to the transaction pool and start work on the next candidate block.
- They will use *your successful block hash* in their next block header, and the race to add a new block of transactions to the blockchain starts again.



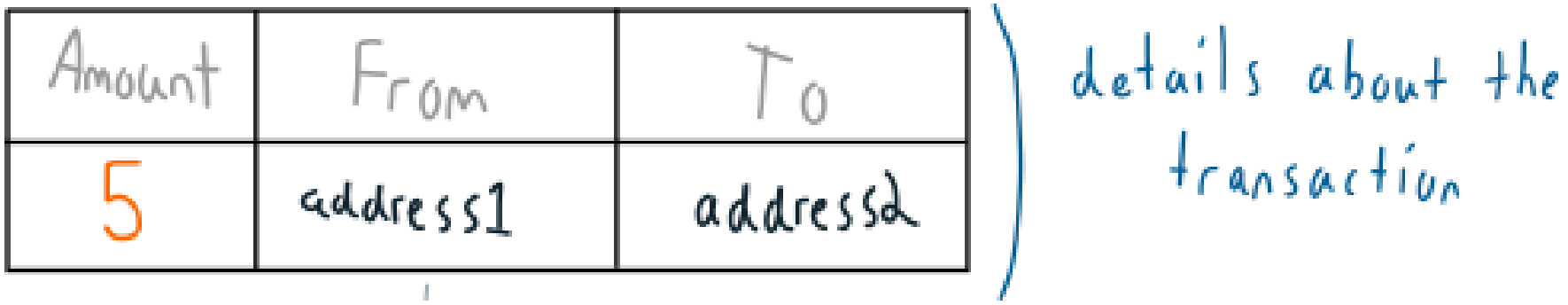
UIT Information Security Lab

Remember: Cryptography in (Bitcoin) Blockchain

- To recap, the bitcoin system has:
  - **private keys:** random numbers
  - **public keys:** based on private keys, and used to ***verify signatures***
  - **addresses:** hash of the public key, used as an address to send funds
- ***Private keys*** and ***public keys*** interact with each other in the following way:
  - **"message" + private key = signature**
  - **"message" + signature + public key = true/false**
- Transactions will be used Cryptography to lock/unlock/verify data.

TRANSACTIONS

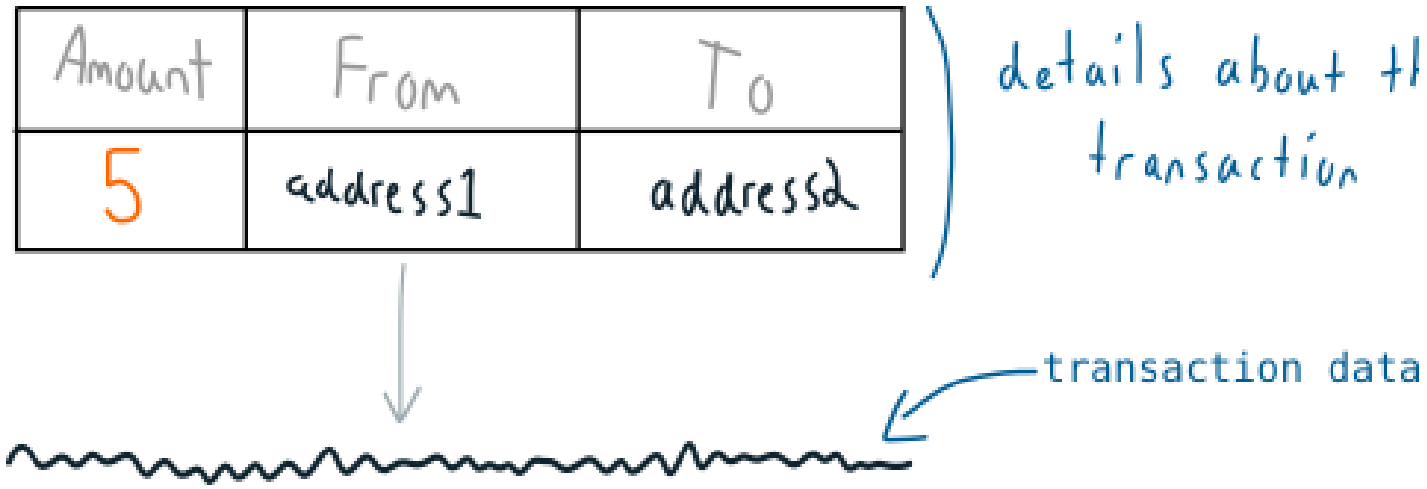
What is a (bitcoin) transaction?



- **A transaction is a bunch of data.**
- This data contains information about the **amount** being sent, the account it is being sent **from**, and the account it is being sent **to**.

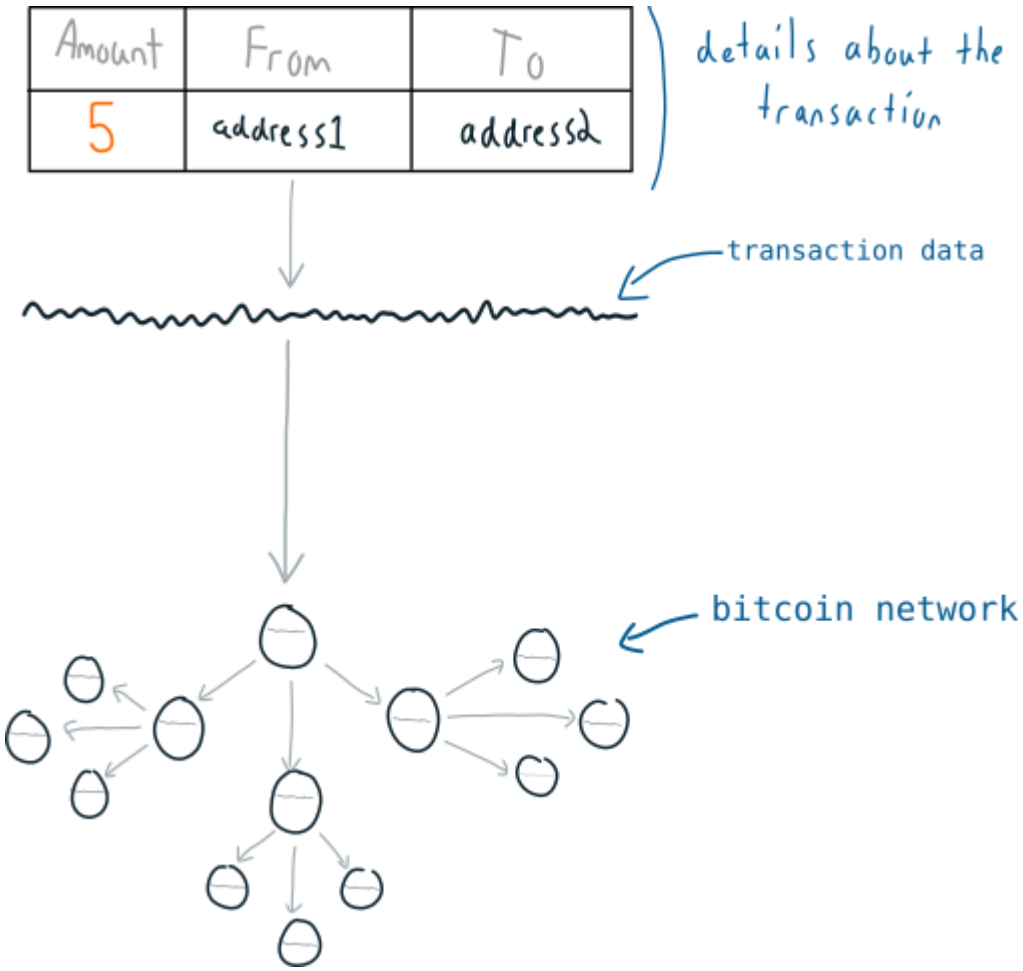
What is a transaction?

- This is basic information, so it can be easily represented in a single line of data



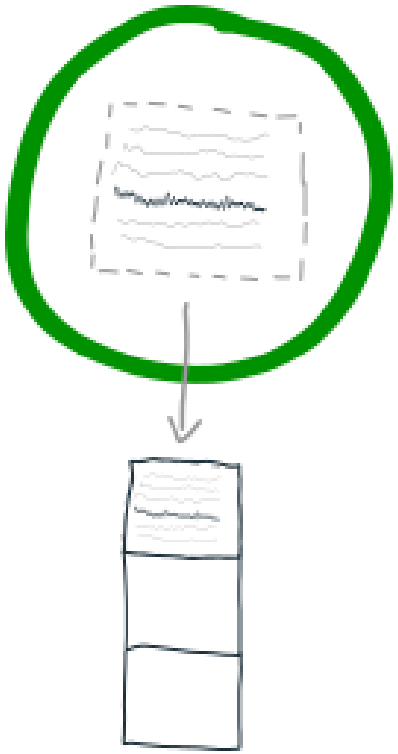
Make a transaction

- And when you “make a transaction”, you just send this **TRANSACTION DATA** in to the (bitcoin) blockchain network.
- Eventually one of the nodes on the network will ***mine*** your transaction into a block, and this block (with your transaction in it) will be added to the file of confirmed transactions (the blockchain).



waiting for it to be mined

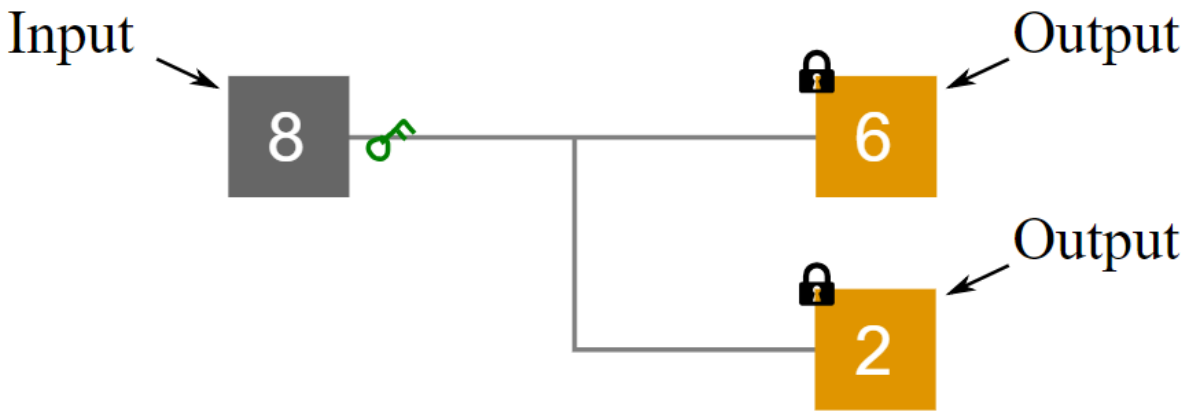
- And that’s all a (bitcoin) transaction is – feeding a simple line of data in to the bitcoin network and waiting for it to be mined in to the blockchain.



Transaction Data

- A bitcoin transaction is just a bunch of data that describes ***the movement*** of bitcoins.
- It takes in inputs, and creates new outputs.
- Bitcoin transactions are composed of 2 lists:
  - **a list of "outputs", each output says:**
    - how much bitcoin is transferred
    - who receives the bitcoin transfer
  - **a list of "inputs", each input includes:**
    - a reference to a past output
    - a "signature" which proves the transaction creator is authorized to spend that output

✓ In transaction 92, with an input worth 8 BTC:
- *Carol pays 6 BTC to Bob*
- *Carol pays 2 BTC to Carol*



UITIS
UIT Information Security Lab

Transaction Data: Structure

01000000017967a5185e907a25225574544c31f7b059c1a191d65b53dcc1554d
339c4f9efc010000006a47304402206a2eb16b7b92051d0fa38c133e67684ed0
64effada1d7f925c842da401d4f22702201f196b10e6e4b4a9fff948e5c5d71ec5
da53e90529c8dbd122bff2b1d21dc8a90121039b7bcd0824b9a9164f7ba09840
8e63e5b7e3cf90835cceb19868f54f8961a825ffffffff014baf2100000000001976
a914db4d1141d0048b1ed15839d0b7a4c488cd368b0e88ac00000000

Transaction:
c1b4e695098210a31fe02abffe9005cffc051bbe86ff33e173155bcbdc5821e3

Transaction Data: Structure

- All of the data in a transaction is in hexadecimal.
- The following icon indicates that the data is in **reverse byte order**: ↻

Field	Data	Size	Description
Version	01000000 ↻	4 bytes	Which version of transaction data structure we're using.
Input Count	01	Variable	Indicates the upcoming number of inputs.
Input(s)			

Field	Data	Size	Description
TXID	796...efc ↻	32 bytes	Refer to an existing transaction.
VOUT	01000000 ↻	4 bytes	Select one of its outputs.
ScriptSig Size	6a	Variable	Indicates the upcoming size of the unlocking code.
ScriptSig	473...825		A script that unlocks the input.
Sequence	ffffffff ↻	4 bytes	

Output Count	01	Variable	Indicate the upcoming number of outputs.
Output(s)			

Field	Data	Size	Description
Value	4baf210000000000 ↻	8 bytes	The value of the output in satoshis.
ScriptPubKey Size	19	Variable	Indicates the upcoming size of the locking code.
ScriptPubKey	76a9...88ac		A script that locks the output.

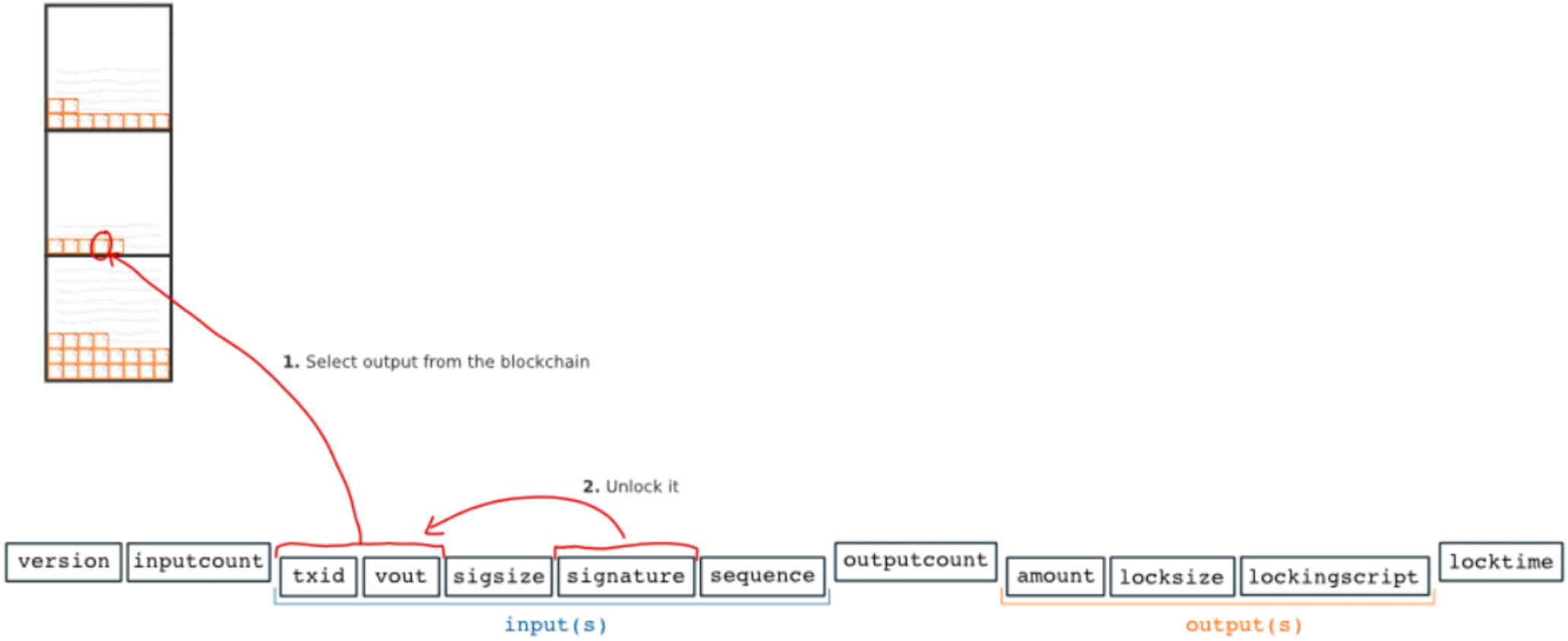
Locktime	00000000 ↻	4 bytes	Set a minimum block height or Unix time that this transaction can be included in.

UITIS
UIT Information Security Lab

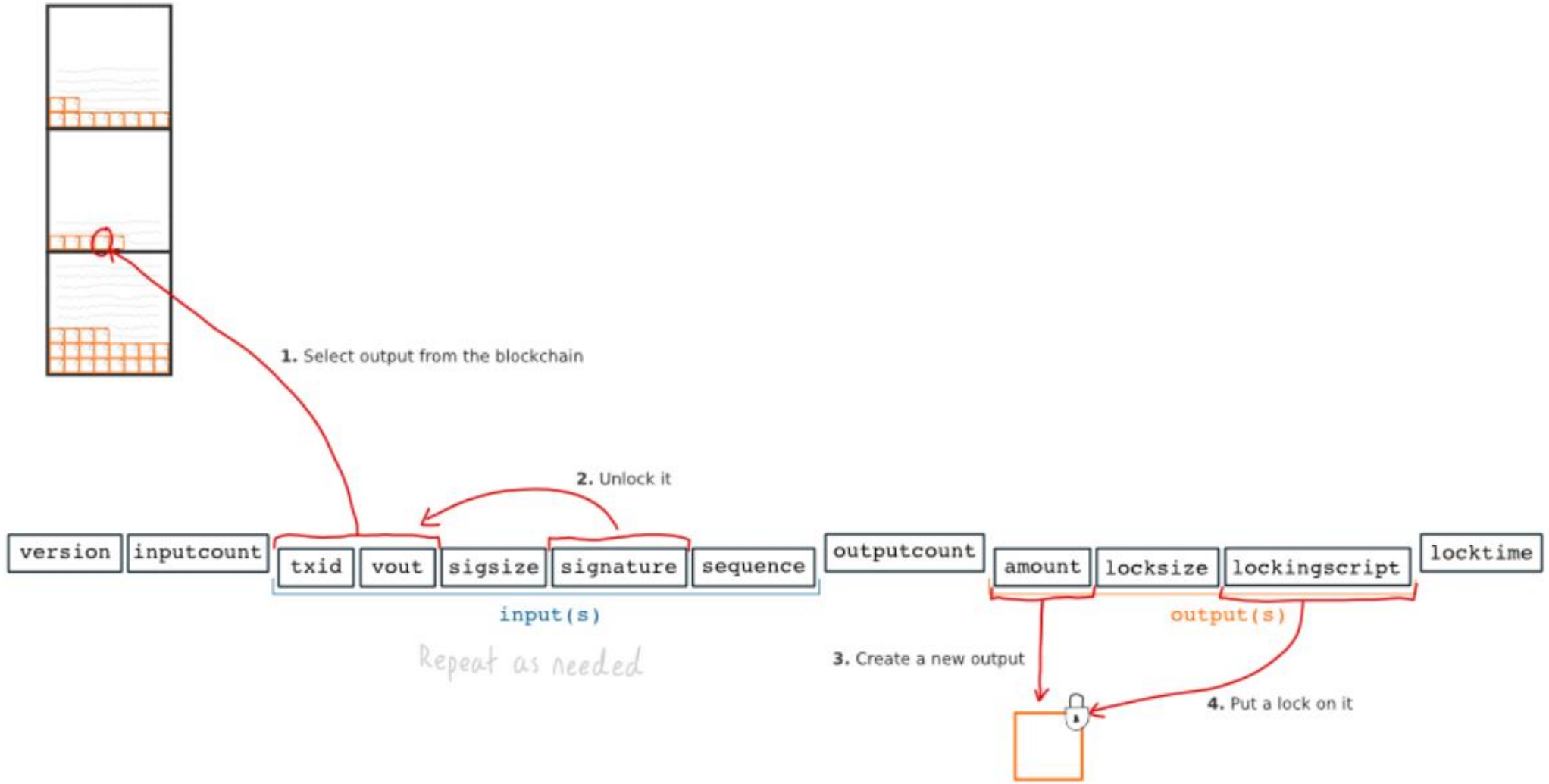
Transaction Data: Structure

- A transaction is basically a **series of inputs** and a **series of outputs**.
- In further detail; the transaction data tells you **how to unlock existing packages of bitcoins** (from previous transactions), and how to lock them up *again* in to new packages.
- A raw transaction is sometimes called a “serialized transaction”, because it's just a bunch of individual pieces of data zipped up in to one string of data.

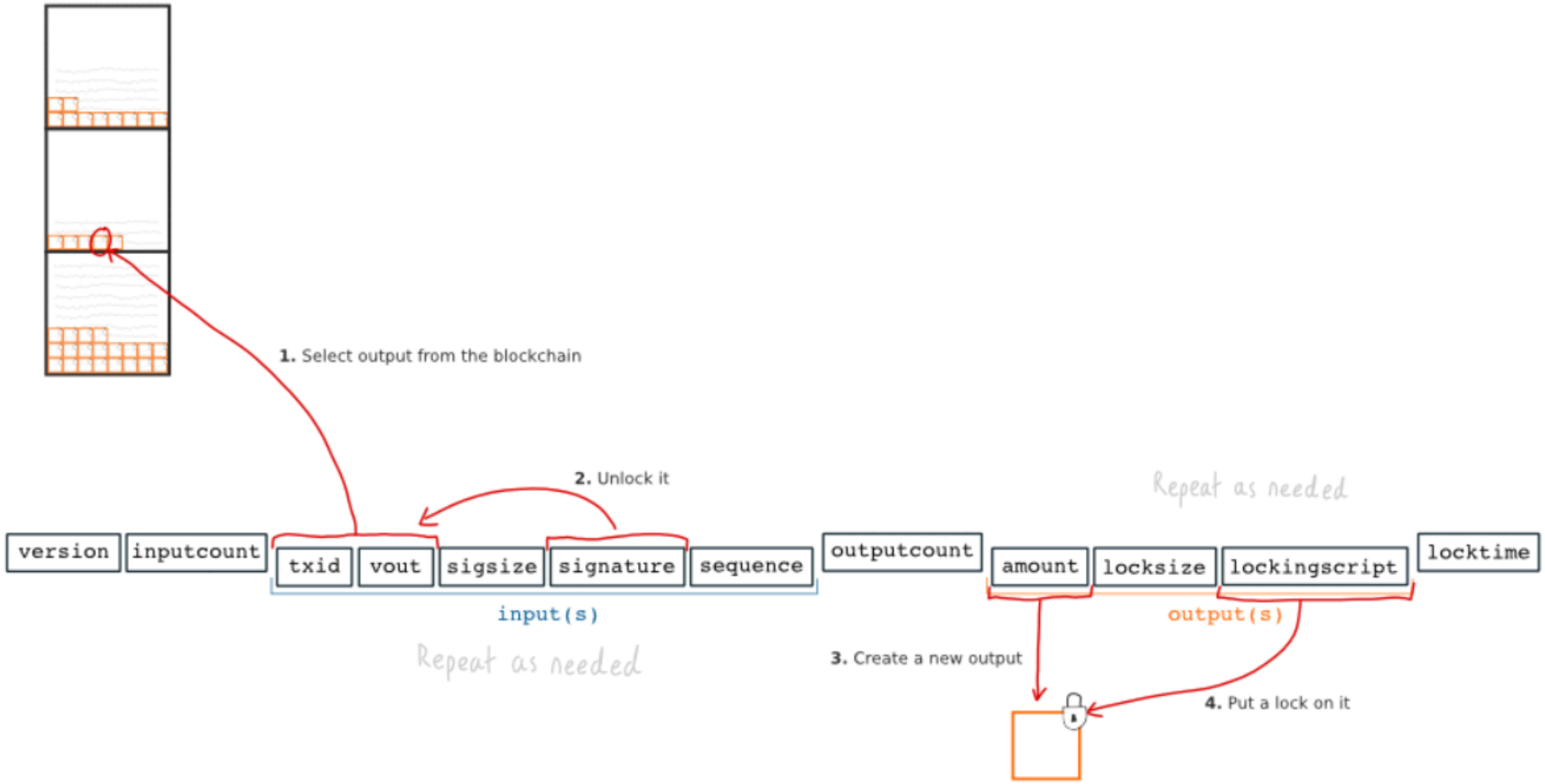
Transaction Data: Structure



Transaction Data: Structure



Transaction Data: Structure



UIT Information Security Lab

Transaction Data: Structure – Bitcoin Example

Private Key:9df5a907ff17ed6a4e02c00c2c119049a045f52a4e817b06b2ec54eb68f70079
Public Key:03c13dca192f1ba64265d8efca97d43b822ff24db357c13b0e6e0395cf91e9efae
Address (decoded):977ae6e32349b99b72196cb62b5ef37329ed81b4
Address (encoded):1EoxGLjv4ZADtRBjTVeXY35czVyDdp7rU4
Transaction Data:

```
01000000 01 f3f6a909f8521adb57d898d2985834e632374e770fd9e2b98656f1bf1fdfd427 01000000 6b 48304502203a776322ebf8eb8b58
cc6ced4f2574f4c73aa664edce0b0022690f2f6f47c521022100b82353305988cb0ebd443089a173ceec93fe4dbfe98d74419ecc84a6a698e
31d012103c5c1bc61f60ce3d6223a63cedbece03b12ef9f0068f2f3c4a7e7f06c523c3664 ffffffff 02 60e3160000000000 19 76a914977ae6e3
2349b99b72196cb62b5ef37329ed81b488ac 063d100000000000 19 76a914f76bc4190f3d8e2315e5c11c59cfc8be9df747e388ac 0000000
0
```

This transaction has 01 input and 02 outputs.
It's identified by it's double hash (see more in TXID):

```
SHA256(SHA256( 01000000 01 ... 02 .. .. .. .. 00000000 )) = b138360800cdc72248c3ca8dfd06de85913d1aac7f41b4fa54eb1f5a4a379081
```

The output which references our address is:

```
60e3160000000000 19 76a914977ae6e32349b99b72196cb62b5ef37329ed81b488ac
```

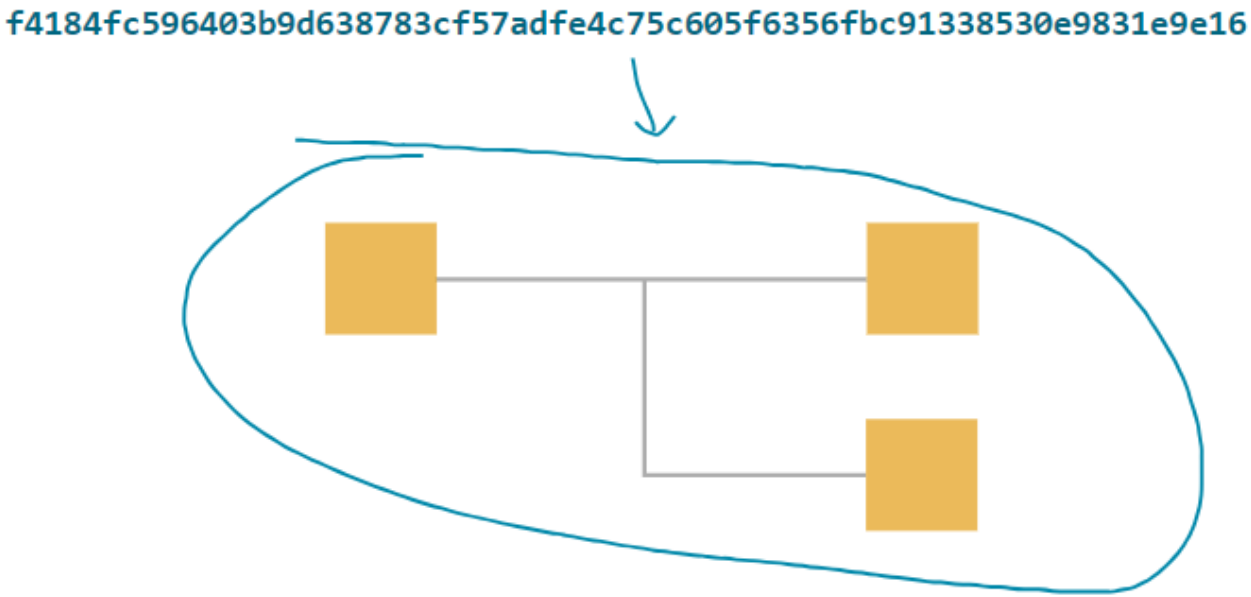
UIT Information Security Lab

Transaction Data: Structure – Bitcoin Example

Hex	English	Description
60e3160000000000	1500000	This is the **payment amount**, in satoshis.
19	25	This is the **size of the locking script**, in bytes. It tells us where this output stops and where the next output begins.
76a914977ae6e32349b99b72196cb62b5ef37329ed81b488ac	OP_DUP OP_HASH160 977ae6e32349b99b72196cb62b5ef37329ed81b4 OP_EQUALVERIFY OP_CHECKSIG	This is the **locking script**, which describes the specific information required to spend the bitcoins in this output. As you can see, "977ae6e32349b99b72196cb62b5ef37329ed81b4" matches our decoded address.

TXID: TRANSACTION ID, TRANSACTION HASH

- A TXID (Transaction ID) is basically an *identification number* for a bitcoin *transaction*.
- A TXID is always 32 bytes (64 characters) and hexadecimal.

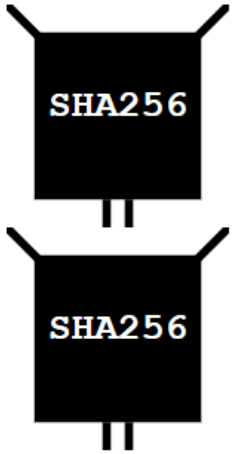


UITIS
UIT Information Security Lab

Creating a TXID

- You get a TXID by hashing transaction data through SHA256 twice.

```
0100000001c997a5e56e104102fa209c6a852dd90660a20b2d9c352423edce25857fcd3704000000004847304402204e45e16932b8af5149
61a1d3a1a25fdf3f4f7732e9d624c6c61548ab5fb8cd410220181522ec8eca07de4860a4acdd12909d831cc56cbbac4622082221a8768d1d
0901ffffffff0200ca9a3b00000000434104ae1a62fe09c5f51b13905f07f06b99a2f7159b2225f374cd378d71302fa28414e7aab37397f5
54a7df5f142c21c1b7303b8a0626f1baded5c72a704f7e6cd84cac00286bee0000000043410411db93e1dcdb8a016b49840f8c53bc1eb68a
382e97b1482ecad7b148a6909a5cb2e0eaddfb84ccf9744464f82e160bfa9b8b64f9d4c03f999b8643f656b412a3ac00000000
```



169e1e83e930853391bc6f35f605c6754cfead57cf8387639d3b4096c54f18f4

Where are TXIDs used?

1. Searching the blockchain.

- If you've just hashed some transaction data and *want to search for a TXID in the blockchain*, you *have to search for* it in *reverse byte order*.

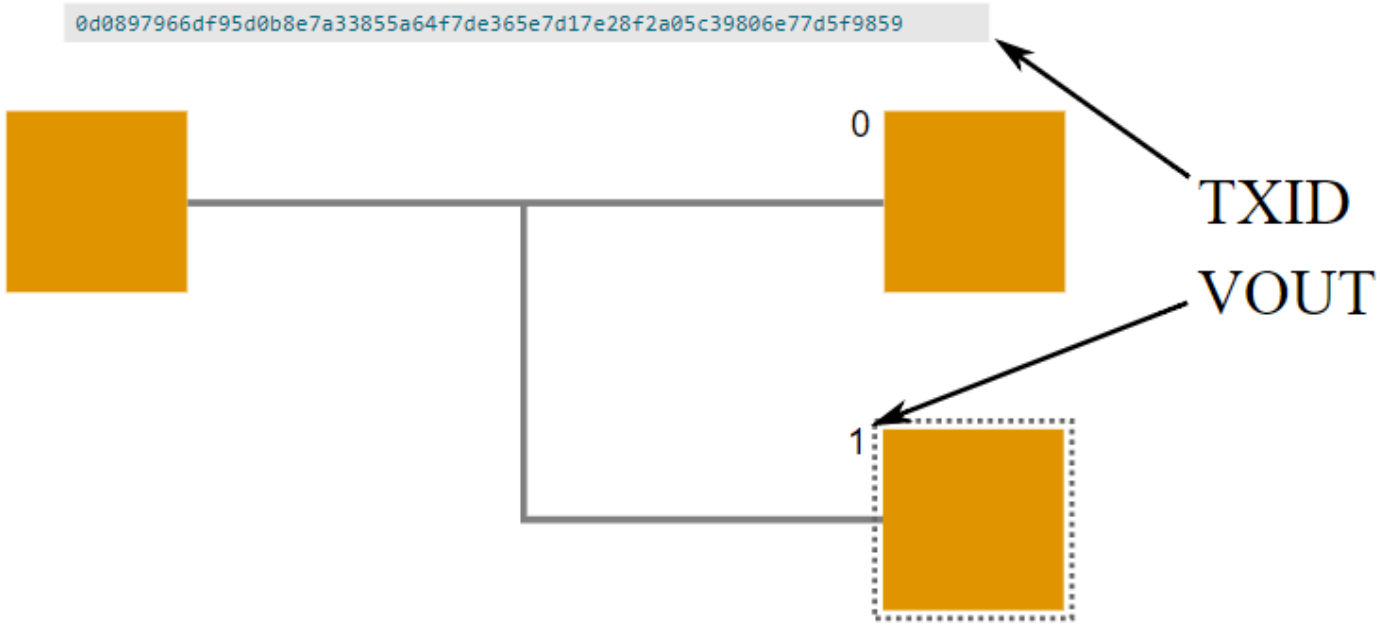
- txid (original):
  169e1e83e930853391bc6f35f605c6754cfead57cf8387639d3b4096c54f18f4
- txid (searching):
  f4184fc596403b9d638783cf57adfe4c75c605f6356fbc91338530e9831e9e16

UITIS
UIT Information Security Lab

Where are TXIDs used?

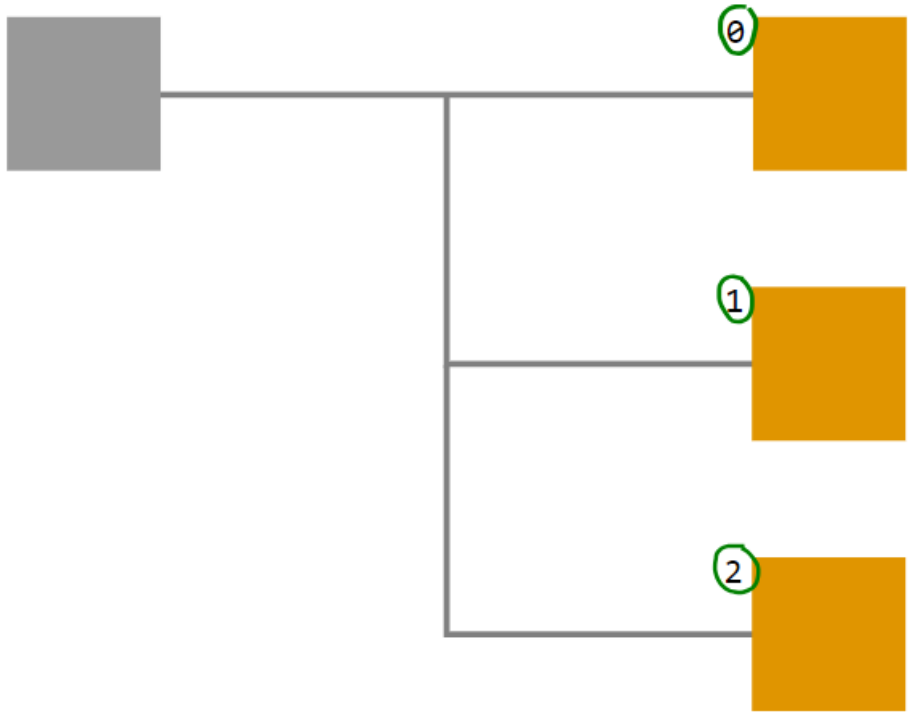
2. Spending outputs.

You use a **TXID** when you want to use an existing output as an input in a new transaction.



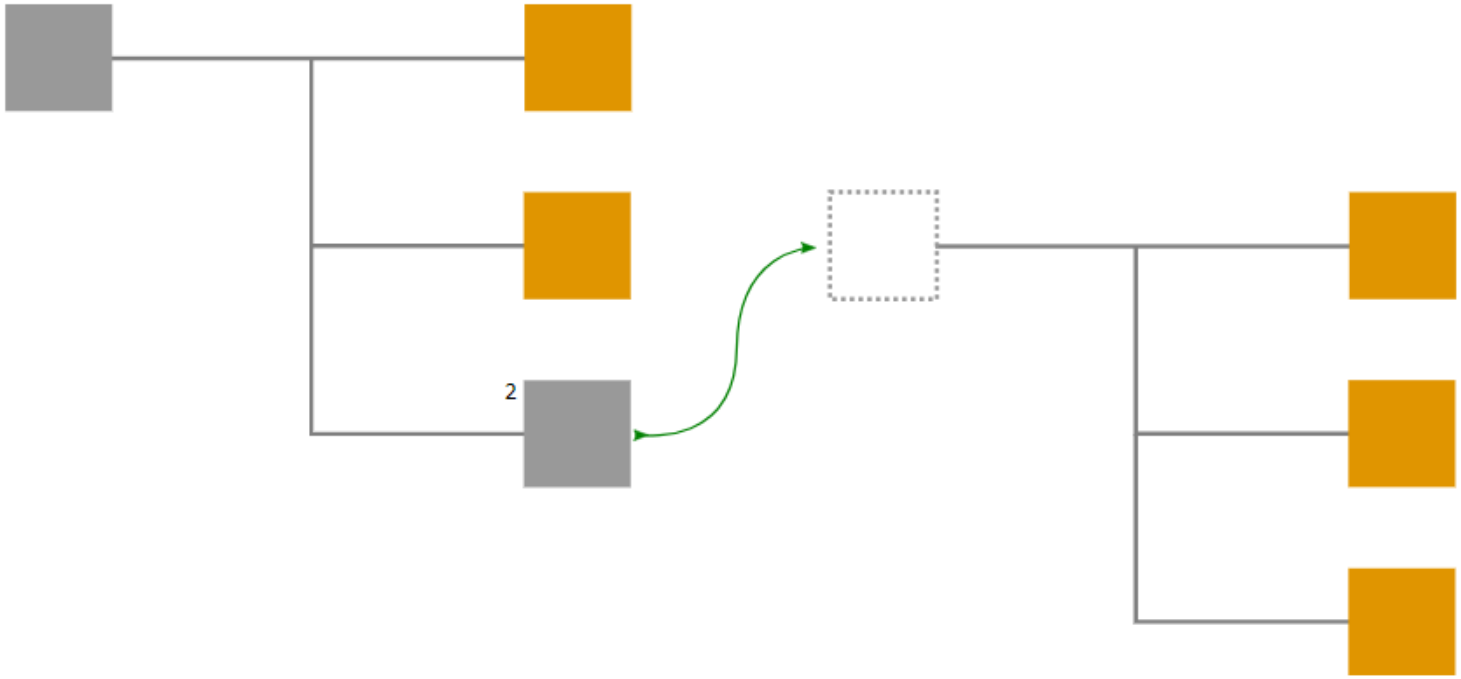
VOUT

- A **vout** is an index number for an output in a transaction.



VOUT: Usage

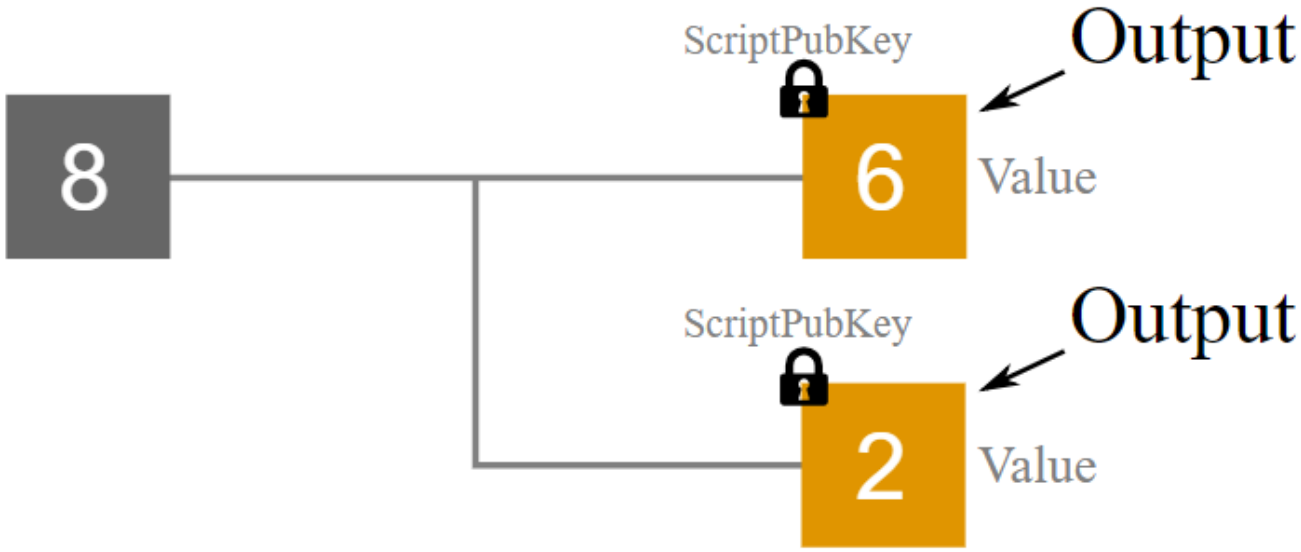
- You use a **txid** and a **vout** when you want to select an existing output as an input in a new transaction.
- In programming, counting starts at 0. So if we want to use the first output of an existing transaction, we put a ***vout*** of 0. (The second output would be 1.)



UIT Information Security Lab

TRANSACTION: OUTPUTS

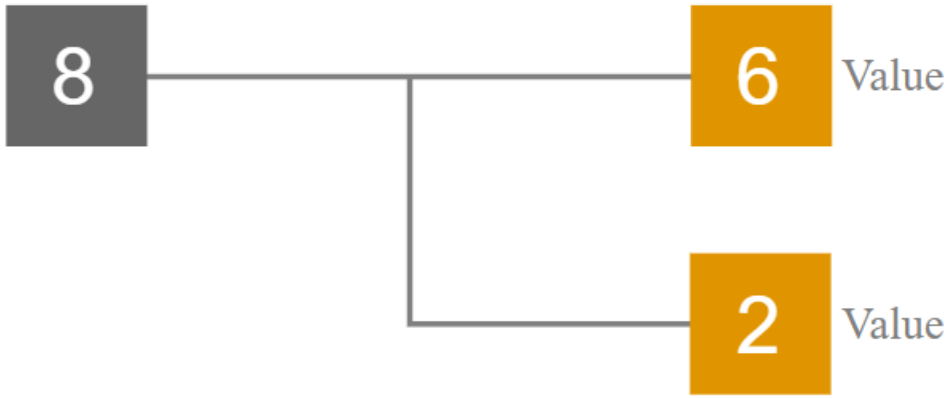
- **The (bitcoin) transaction system involves sending and receiving whole batches of bitcoins, called OUTPUTS.**
- Outputs are **packages of bitcoins** created in a bitcoin transaction.
- Each output has a **lock**, which means that they can only be used as an inputs in a future transaction by people who can unlock them.



UITIS
UIT Information Security Lab

How do Outputs work?

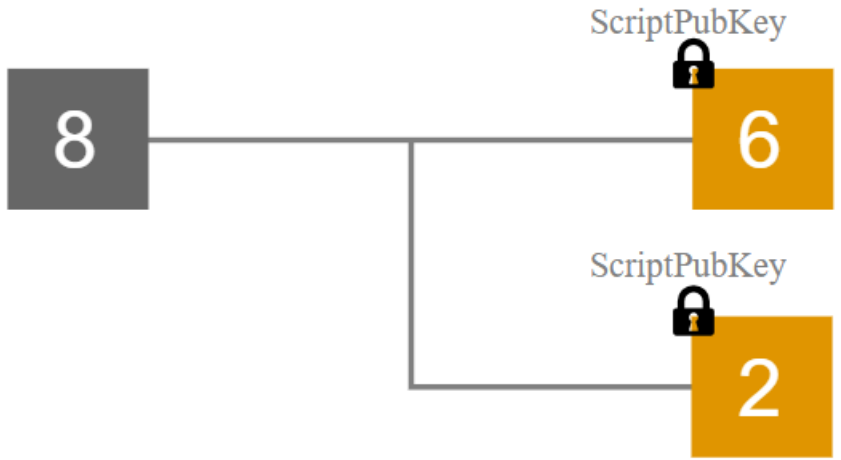
- After selecting **Input(s)** to spend, you can create as many **Outputs** from them as you like.
- For each output you just:
  - **Give it a value.** Every output has a value. You can create as many outputs as you like, as long as their sum does not exceed the sum of the Inputs you are spending.



UIT Information Security Lab

How do Outputs work?

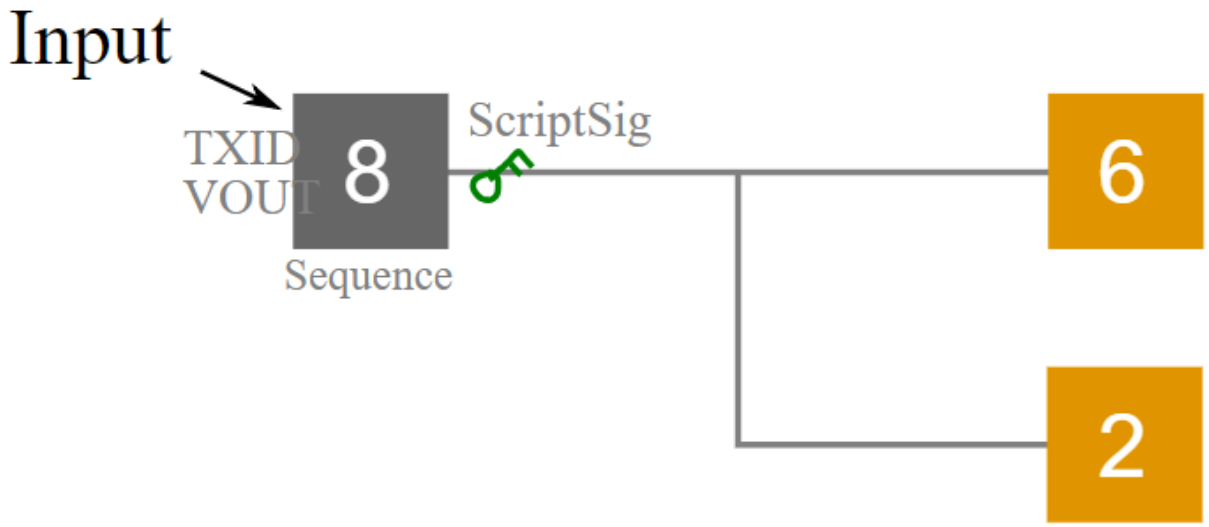
- **Give it a lock.** You also place locks on outputs when you create them them.
- These locking scripts prevent other people from using these outputs as inputs in another transaction (i.e. spending them). This locking code is called a ***ScriptPubKey***.



UITIS
UIT Information Security Lab

TRANSACTION: INPUT

- *An input* is what you **call an output** when you're *spending* it in a *transaction*.



Input Field in Transaction Data

01000000017967a5185e907a25225574544c31f7b059c1a191d65b53dcc1554
d339c4f9efc**01000000**6a47304402206a2eb16b7b92051d0fa38c133e67684ed
064effada1d7f925c842da401d4f22702201f196b10e6e4b4a9fff948e5c5d71ec
5da53e90529c8dbd122bff2b1d21dc8a90121039b7bcd0824b9a9164f7ba0984
08e63e5b7e3cf90835cceb19868f54f8961a825**ffffffff**014baf21000000000019 7
6a914db4d1141d0048b1ed15839d0b7a4c488cd368b0e88ac00000000

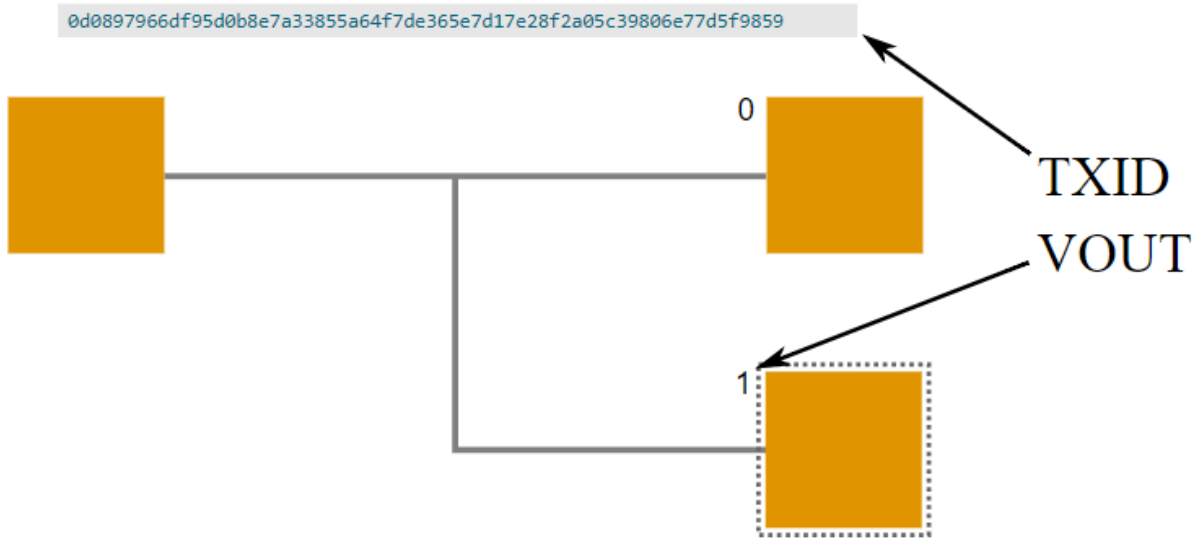
Field	Data	Size	Description
TXID	796...efc	32 bytes	Refer to an existing transaction.
VOUT	01000000	4 bytes	Select one of its outputs.
ScriptSig Size	6a	Variable	Indicates the upcoming size of the unlocking code.
ScriptSig	473...825		A script that unlocks the input.
Sequence	ffffffff	4 bytes	

UITIS
UIT Information Security Lab

How do Inputs work?

1. Select an Output.

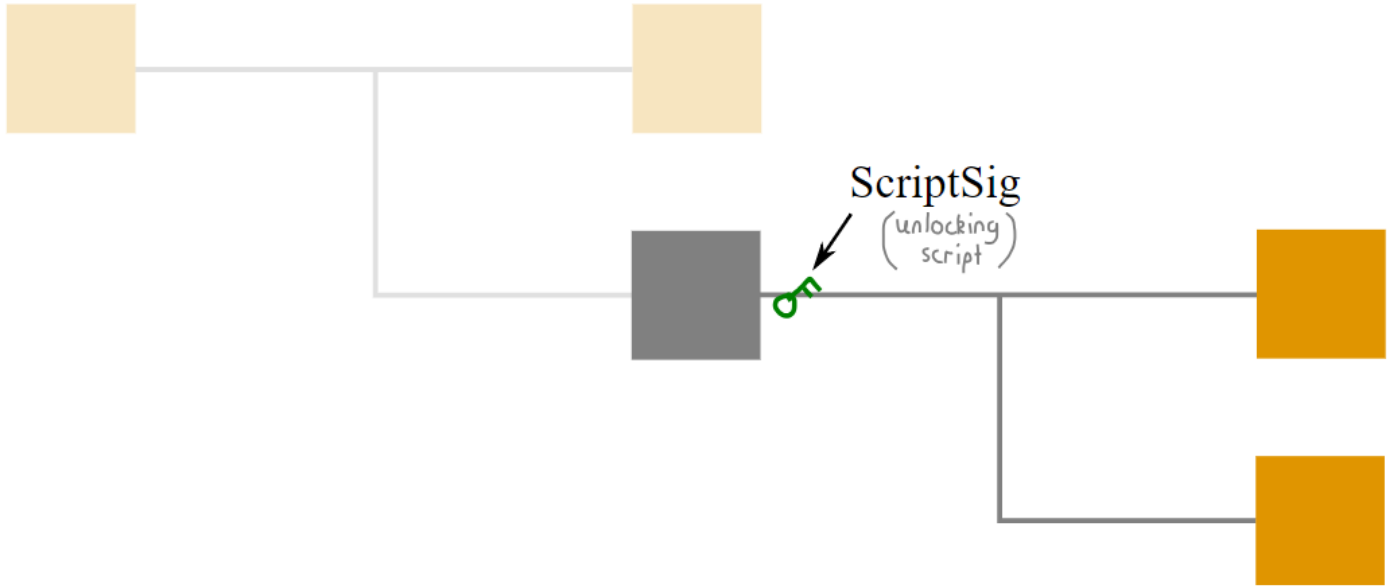
- When you want to use an output as an input for a transaction, you just need to specify which one you want to spend.
- Every transaction has a unique TXID, so by using that with a specific output number (VOUT), you can refer to any output in the blockchain.
- All you need is a **txid** and a **vout** and you can select any output from the blockchain.



How do Inputs work?

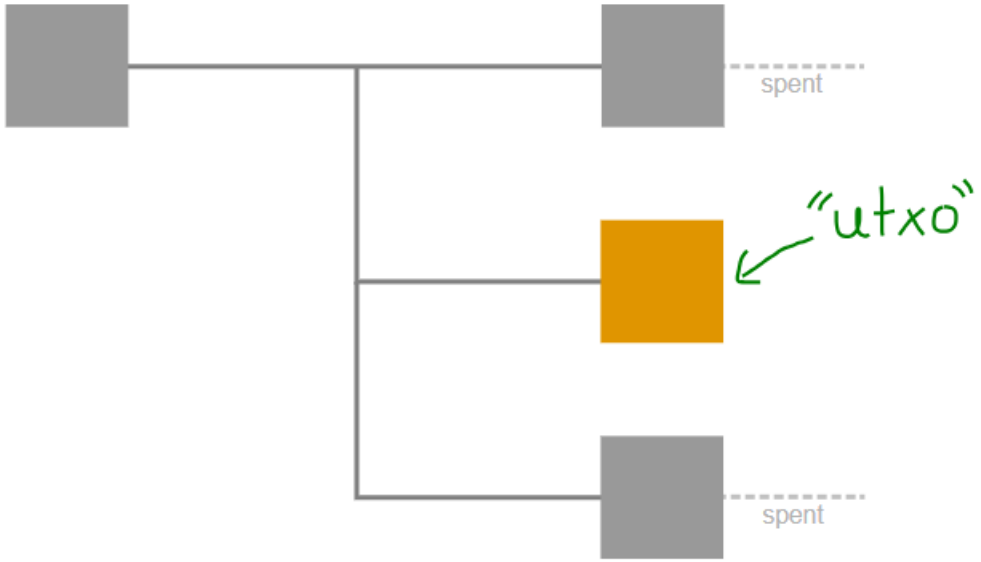
2. Unlock it.

- *After selecting an output*, you then *have to be able to unlock* it.
- Each output is set with a locking script. So if you want to spend one, you need to supply an unlocking script (called a ScriptSig).
- Nodes validate every transaction they receive. So if you do not provide an unlocking script that satisfies the locking script, your transaction will get rejected.



UTXO: Unspent Transaction Output.

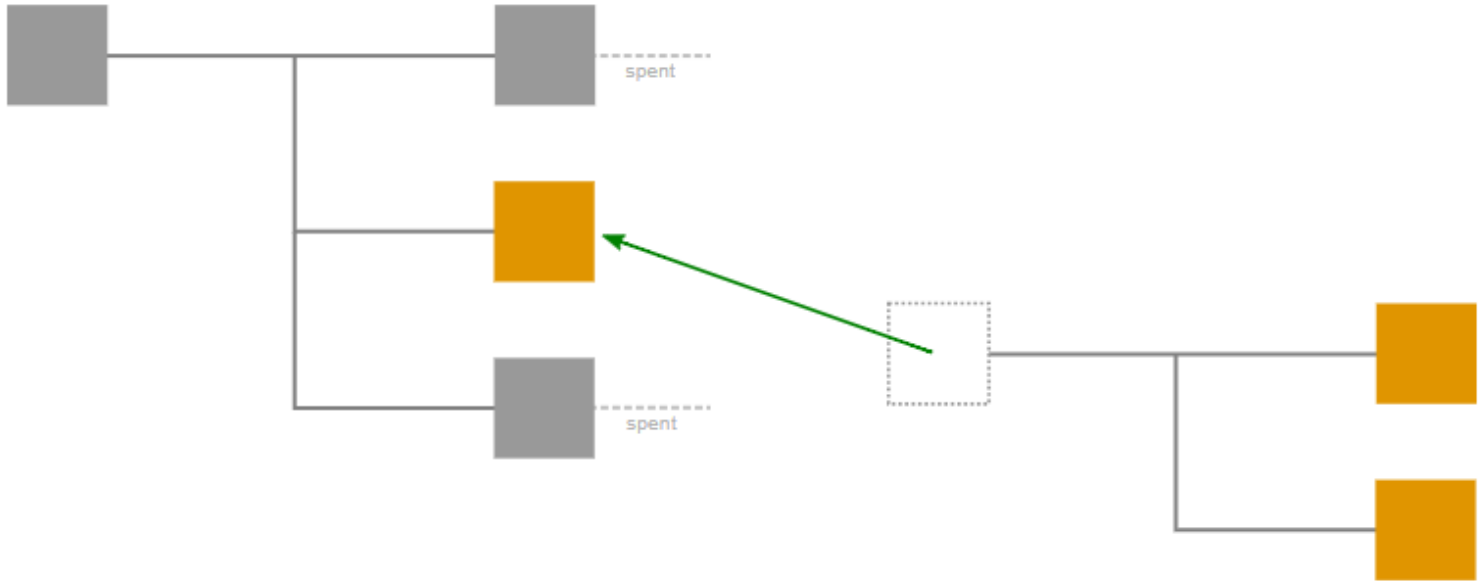
- After ***an output*** has been ***"used up"*** in a transaction, it **cannot be used again.**
- **UTXO** are available to be used in new transactions (as long as you can unlock them), which makes them useful. That's why there is a distinction between spent outputs and unspent outputs (UTXOs).



UTXO: Where are UTXOs used?

1. Verifying Transactions

- A node will verify the transactions it receives by checking that its inputs have not already been spent.
- So if you want to create your own bitcoin transaction, you must use UTXOs in your inputs.

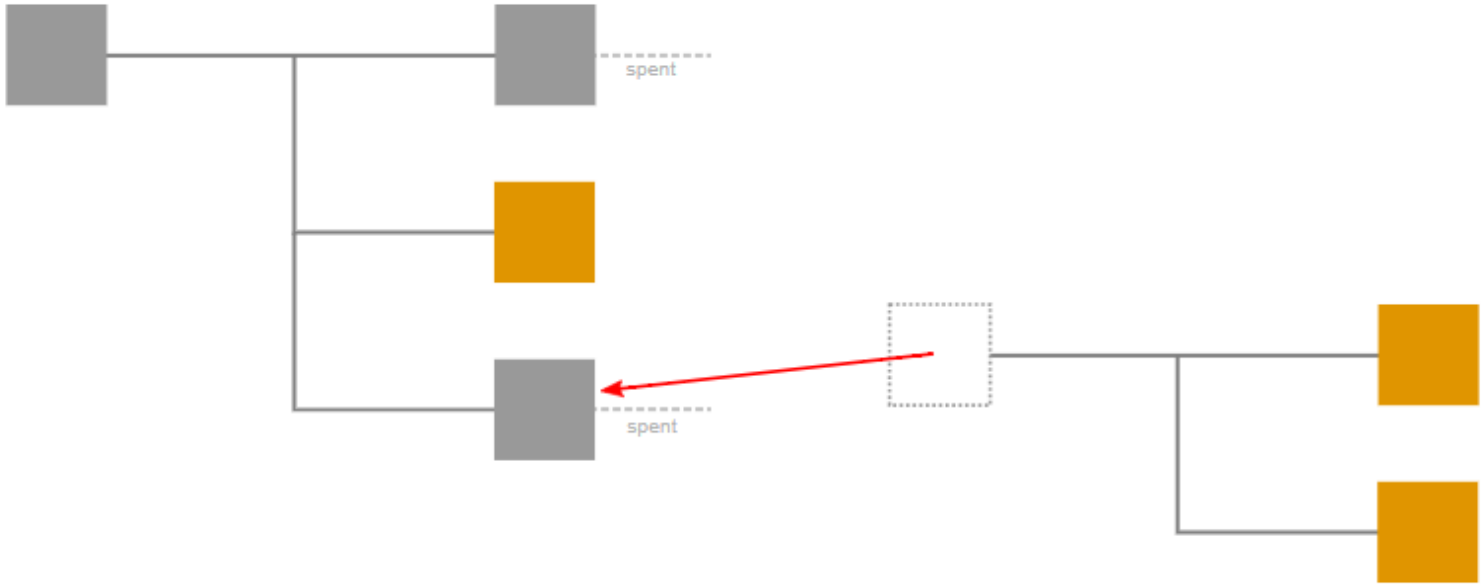


The new transaction is referring to an unspent output. All good.

UTXO: Where are UTXOs used?

1. Verifying Transactions

If you try and use an output that has already been used in another transaction, your transaction will be rejected by nodes.

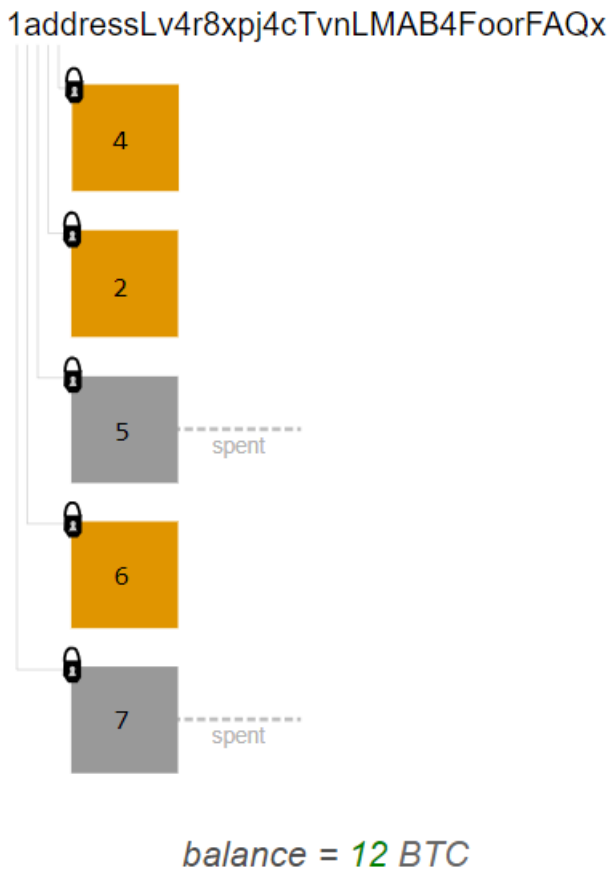


The new transaction is referring to a spent output. Nodes will reject this transaction.

UTXO: Where are UTXOs used?

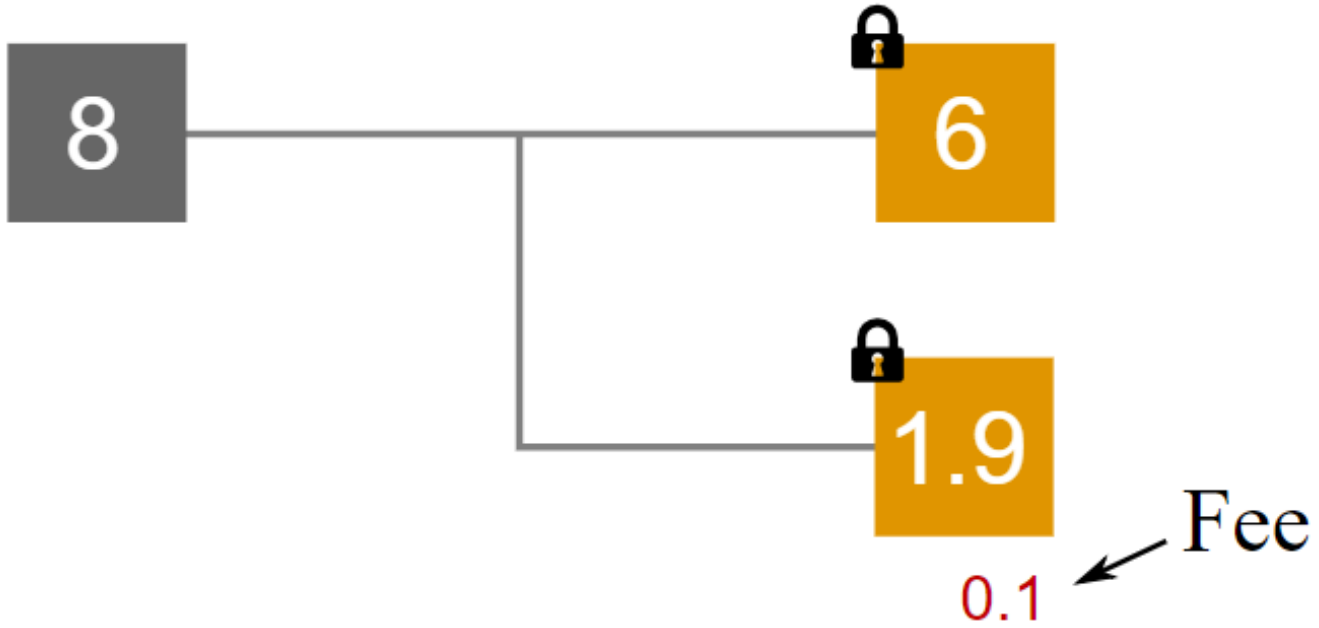
2. Address Balances

If you want to work out the balance of an address, add up all of the unspent outputs that are locked to that address.



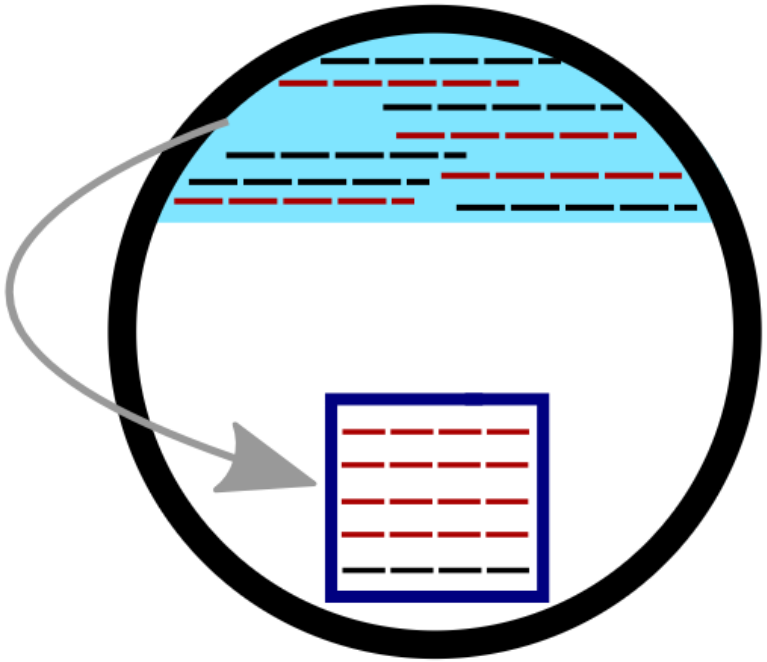
TRANSACTION FEE

- A **transaction fee** is the remainder of a bitcoin transaction.



Why use a transaction fee?

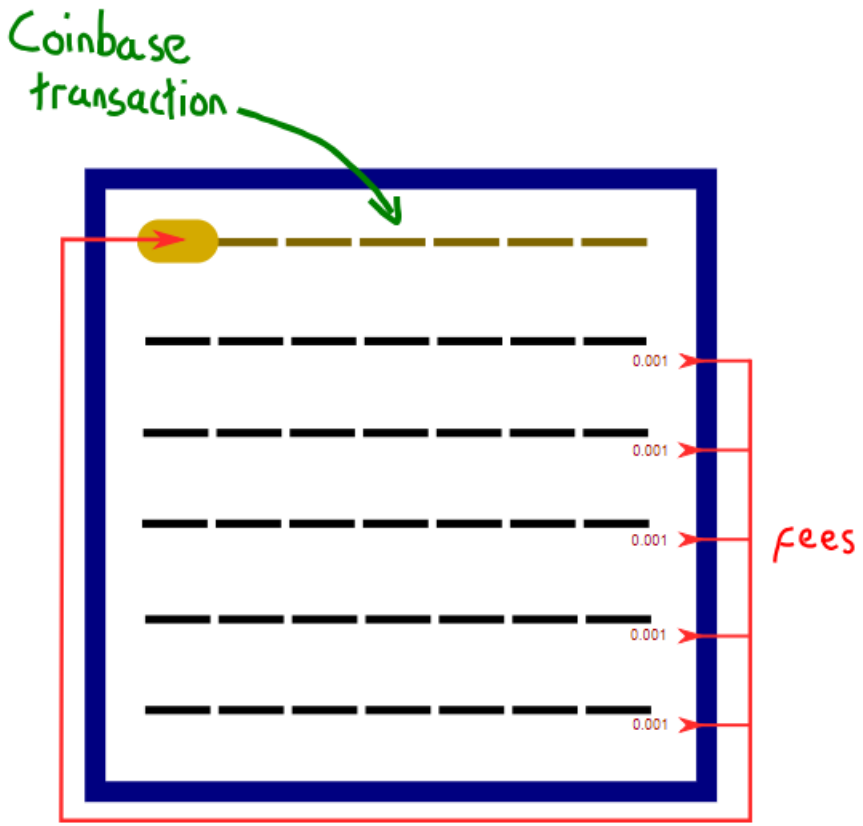
- A transaction fee acts as an incentive for a miner to include your transaction in their candidate block.
- If there are more transactions in the memory pool than can fit in to a block, a miner will select transactions with the highest fees.
- So if there are a lot of bitcoin transactions floating around the memory pool and not all of them can fit in to a block, a **transaction fee** can be used as a way to *"buy space"* in a block.



UITIS
UIT Information Security Lab

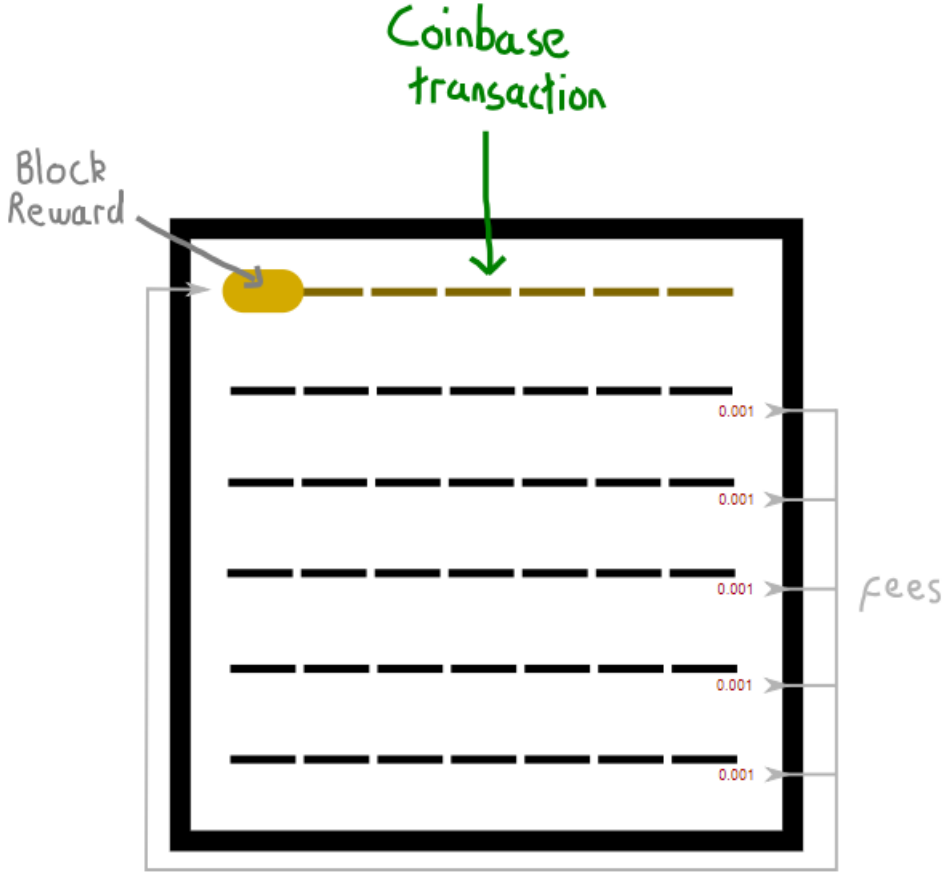
Where do transaction fees go?

- *Transaction fees* are claimed by miners through the **coinbase transaction.**



Coinbase Transaction

- A **coinbase transaction** is the ***first transaction in a block***.
- Miners use it to collect the **block reward**, and any additional **transaction fees**.
- It's like putting your details on a self-addressed envelope so you can collect prize winnings.



Block Reward

- If a miner mines a new block, they're given a reward in the form of the block reward (coinbase). This is the main incentive for Bitcoin miners.
- The **block reward** is halved every **210,000 blocks**, which is approximately ***every 4 years*** (as Bitcoin's block time is 10 minutes per block).
- When Bitcoin was created the Block reward used to be **50** Bitcoin, and is now 12.5 BTC.
- When the block reward has halved *64 times*, the block reward becomes **0**.
- The last new bitcoin will not be mined until ***May 2140***.

UIT Information Security Lab

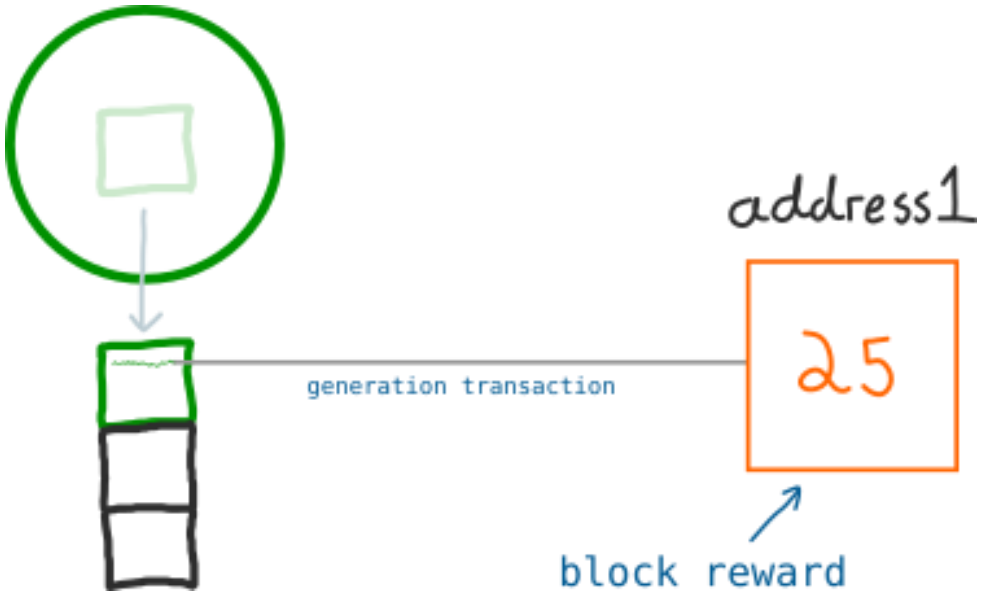
Coinbase Transaction: Usage

- When a miner creates a candidate block, the very first space for a transaction is reserved for the coinbase transaction.
- *Every block* must have a **coinbase transaction**.
- A **coinbase transaction** is only slightly different to normal transaction data. The main difference is its **single "blank" input**, which we call the *coinbase*.

UIT Information Security Lab

Transaction 1 - A simple transaction

- Let's begin this story of transactions with the birth of a fresh batch of bitcoins…
- You are ***mining*** bitcoins on your own. By some miracle, you have managed to mine a block of transactions and earn yourself a fresh batch of ***25 bitcoins***.
- ***Every miner*** also includes their own address at the top of each block, so if they manage to mine the block, the **BLOCK REWARD** can be sent to their address. This is known as the **GENERATION TRANSACTION**.



UITiS
UIT Information Security Lab

Transaction 1 - A simple transaction

- So this is the current state of your bitcoin address:

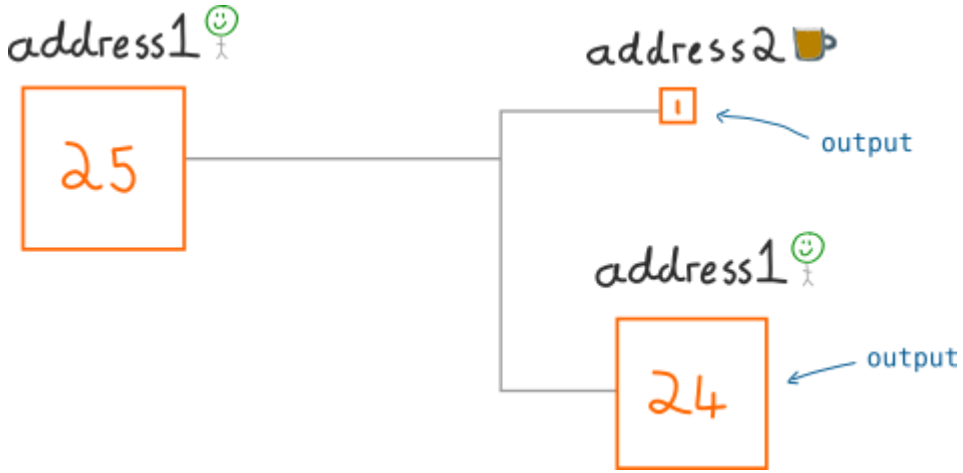
Naturally, your first instinct is to celebrate. So let's use **1 of these bitcoins** to **buy some beer**.

UIT Information Security Lab

Transaction 1 - A simple transaction

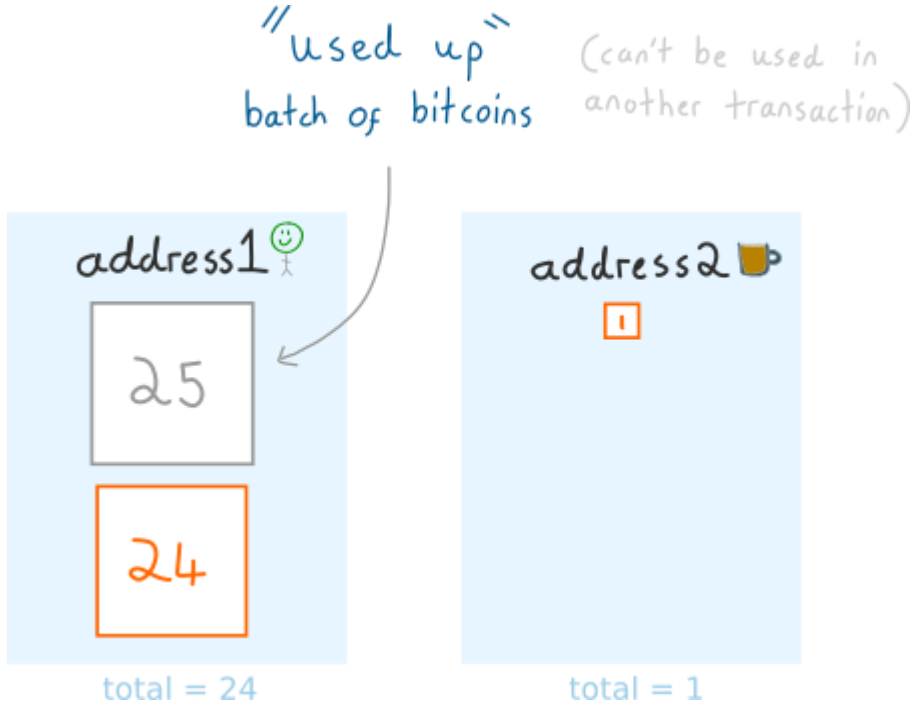
- Now, your other first instinct would be to take 1 of these bitcoins (from the batch of 25) to pay for this beer. This would make sense, **but it's not how transactions work**.

Transaction 1 - A simple transaction



- Instead, we have to **send the entire batch of 25 bitcoins** in the transaction.
- The newly created batches are called OUTPUTS.
- But to make sure we *don't spend all 25 bitcoins* in a "1 bitcoin" payment, we **split the batch up** and send it to *two destinations*:
  - The beer shop. (the payment)
  - Back to our own address. (our change)

Transaction 1 - A simple transaction



- This is what the bitcoin addresses look like ***after*** the transaction
- The beer shop has a new batch of 1, and we've sent ourselves a new batch of 24. That original batch of 25 bitcoins has now been "used up".

UITiS
UIT Information Security Lab

Summary: How Transactions work?

- This is what the bitcoin transaction system is designed to do:
  - Take an existing output (a batch of bitcoins)
  - Create newly-sized outputs (batches) from it
  - Send those outputs to different addresses

Transaction 2 - Using outputs as inputs.

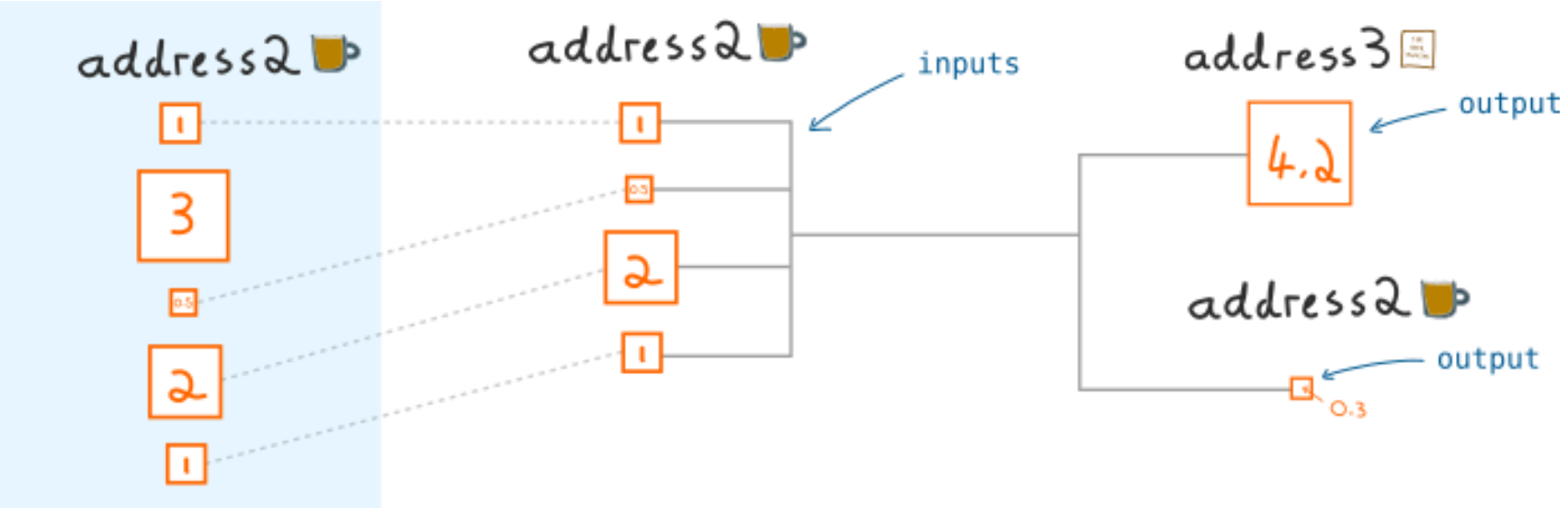
- Okay, from now on we're going to use the word "**OUTPUT**" instead of "batch".
- Anyway, a few days have passed since the beer shop sold us that beer. And judging by the current state of their bitcoin address, the beer business is booming:

But as we all know, beer doesn't grow on trees. So the beer shop is on the lookout for a brand new beer machine.

- Oh look, a lovely beer machine for the low low price of 4.2 bitcoins.

Let's buy it…

Transaction 2 - Using outputs as inputs.

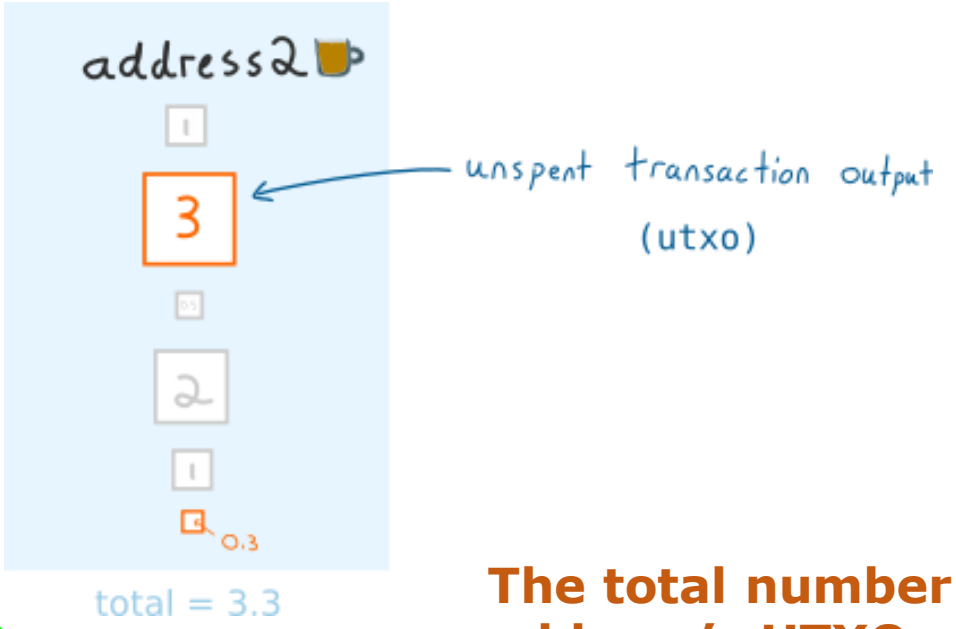


- Constructing the transaction for the beer machine:
  - The beer shop doesn't have a single output (batch) at their address to cover the cost of the beer machine (**4.2**). So instead, we *gather a handful of outputs together* to get a total **greater than 4.2**.
  - When we construct a transaction, the outputs we are gathering up to be spent are referred to as the transaction "INPUTS".
  - Using the total input of 4.5, the beer shop creates two new outputs of 4.2 and 0.3.

UITiS
UIT Information Security Lab

Transaction 2 - Using outputs as inputs.

- And here's the state of the beer shop's bitcoin address after the transaction:



- The *outputs* that were used as *inputs* have been "**spent**", and can't be used again.
- The "***unspent***" outputs are still good for spending, so we call these the unspent transaction outputs (**UTXOs**).

The total number of bitcoins at an address is the sum of the address's UTXOs.

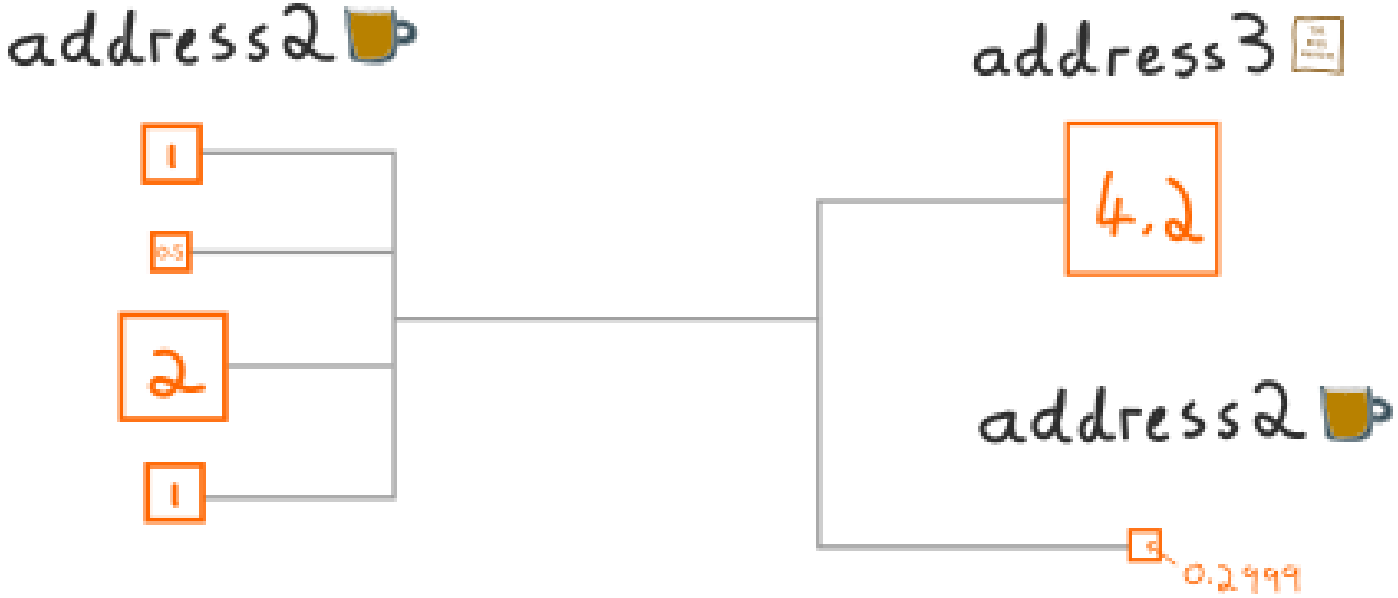
UITIS
UIT Information Security Lab

Transaction 3 - Transaction fees.

- We've not included a transaction fee in either of the last two transactions. The transaction fees are picked up by miners when they mine a block, so adding a transaction fee basically acts as an incentive for miners to include your transaction in a block.
- Without a transaction fee, those two transactions will probably take a while to get included in to a block. This is because a ***transaction fee gives your transaction*** **PRIORITY**.

Transaction 3 - Transaction fees.

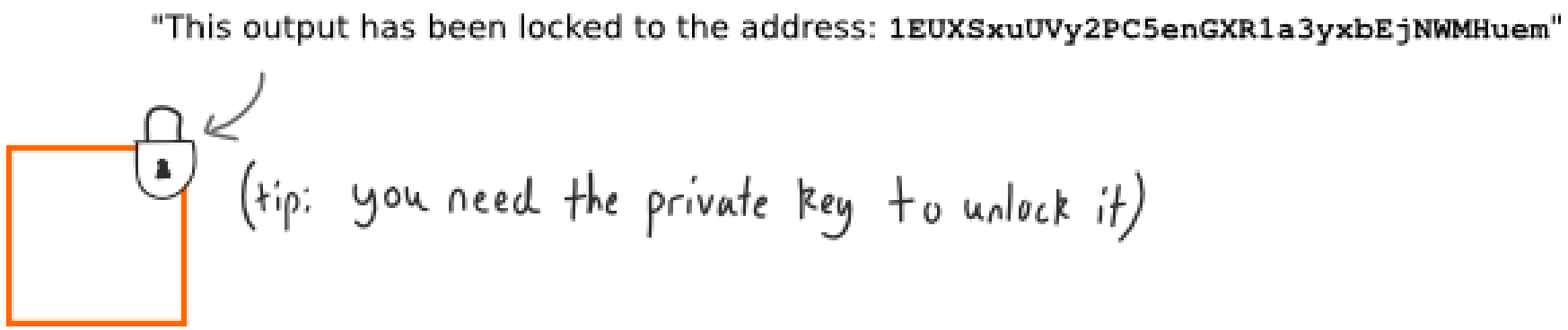
- Pretend we didn't send that last transaction in to the network, and let's *add a transaction fee to it*.
- **The total of the outputs is less than the total of the inputs, which means that there are some remaining bitcoins that aren't being used up. This "left over" amount is the transaction fee.**
- And that's all transaction fees are - the remainder of a transaction.



Output Locks

What is an output lock?

- An output lock is a set of requirements placed on an output. These requirements must be met to be able to use the output in a transaction.
- For example, the most common output lock reads something like this:

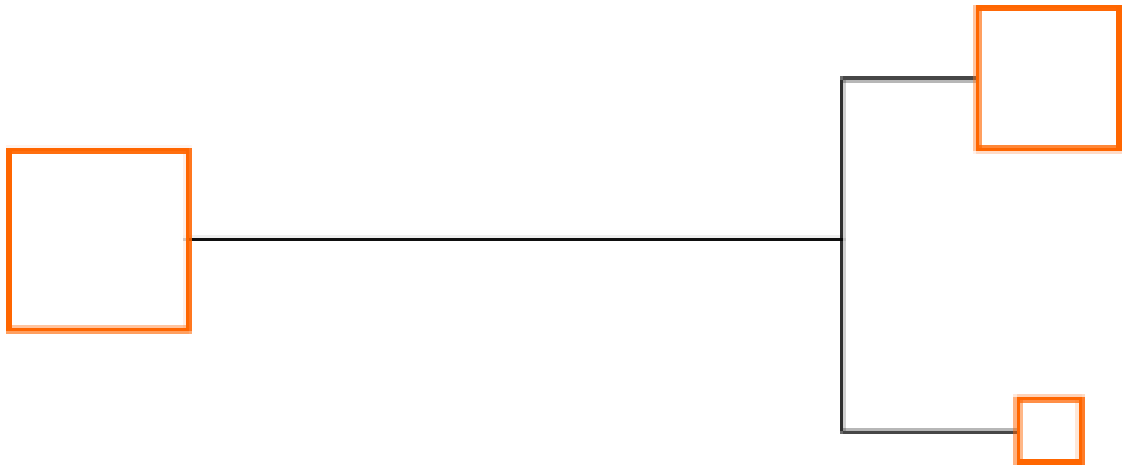


It's these locks that prevent us from spending each other's outputs in a transaction, as every output we receive is encumbered by a lock.

UITiS
UIT Information Security Lab

Where do output locks come from?

- As we know, a transaction is the process of taking existing outputs and creating new ones from them:



Where do output locks come from?

- And it’s during the creation of these outputs that we give each one a “lock”.

Where do output locks come from?

- So when ***we want to send bitcoins to a friend***, we *create the new output*, and add a lock that says "only the owner of 1friend1234567890 can use this output".
- All of this is stored in the transaction data.
- As a result, this new output will effectively "belong" to our friend, because they are the only person who has the private key for this address, so nobody else will be able to spend it.

How do you create an output lock?

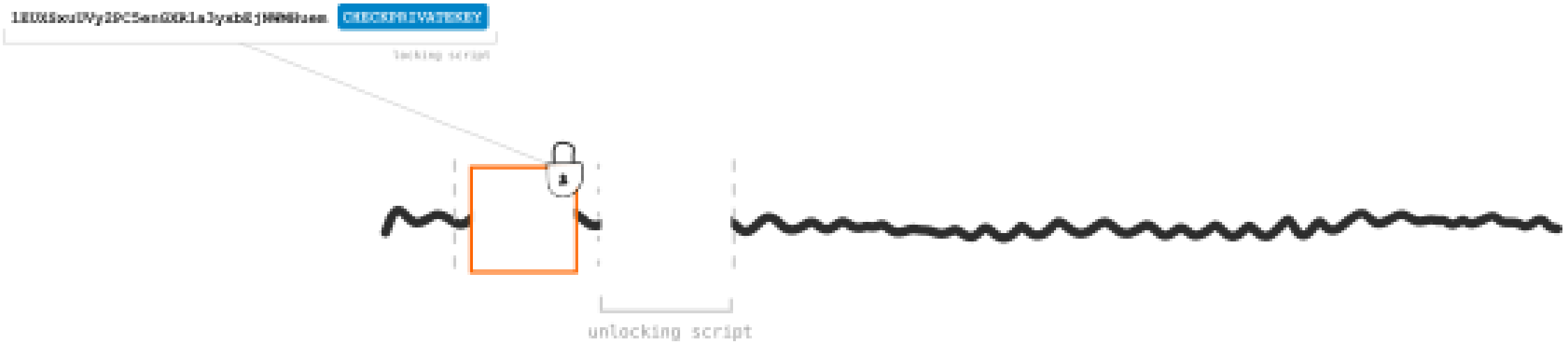
- Output locks are written in a basic programming language, called **SCRIPT.**
- It's a bit tricky to explain the workings of an entire programming language in one diagram, but here we go anyway:

- CHECKPRIVATEKEY: is a function that we use to help set the requirements for the lock.
- So for this particular output, we've set a lock that wants to compare the address **1EUXSxuUVy2PC5enGXR1a3yxbEjNWMHuem** with a private key.
- If we can provide this lock with a matching private key, we can unlock it and use it in a transaction

UIT Information Security Lab

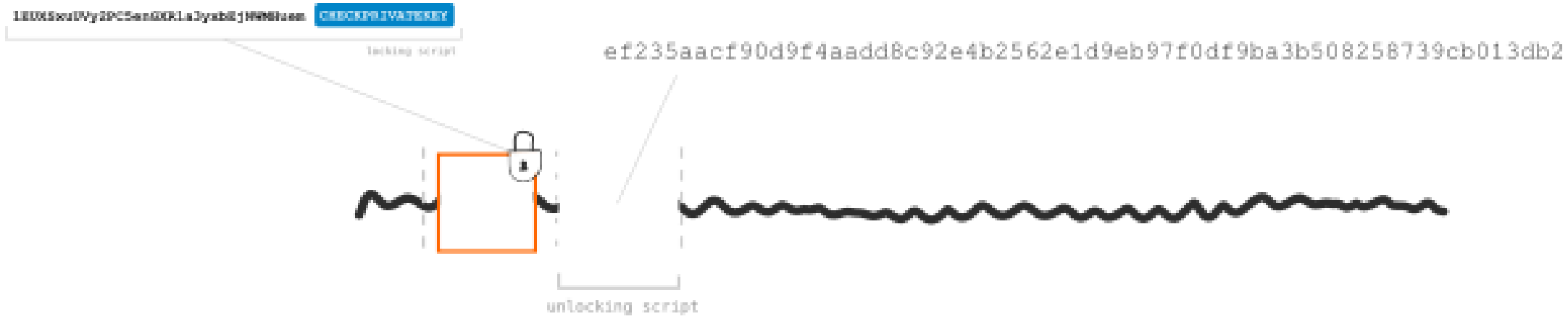
How do you unlock an output lock?

- When you construct the transaction data, you include an “unlocking script” after each output you intend to use:



How do you unlock an output lock?

- So for example, to unlock a typical locking script (e.g. [address][CHECKPRIVATEKEY]), we need to prove that we own the address [address]. To do this, we use our **private key *to create a digital signature***.



You put your digital signature as the “unlocking script”.

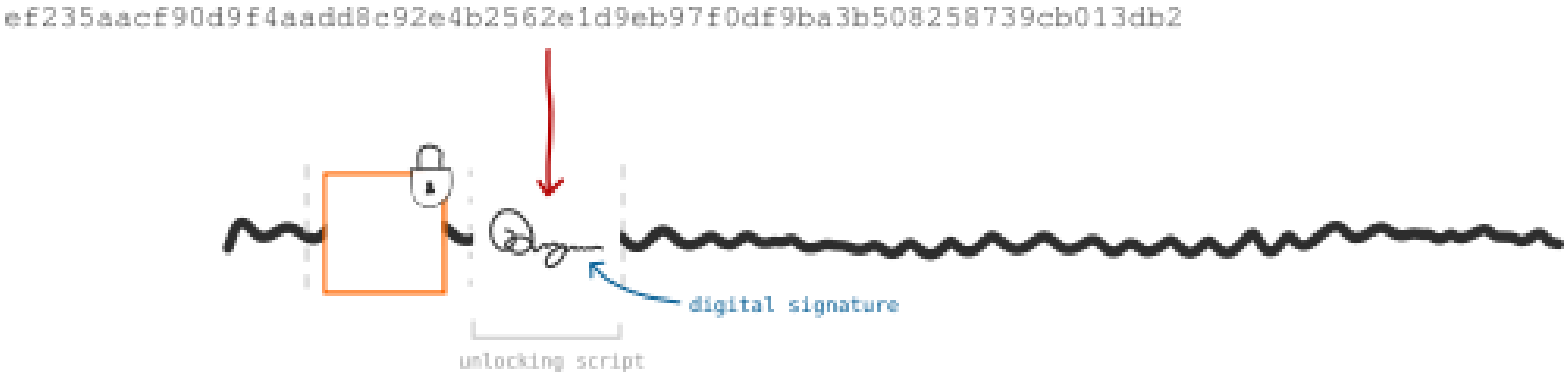
How do you unlock an output lock?

- So when a node receives this transaction data, they will run the **“locking”+“unlocking”** scripts together to see if your digital signature matches the address that the output has been locked to.

UITiS
UIT Information Security Lab

Confession: We don’t actually put our private key directly in to the transaction data.

- You see, to save us from giving our private key away within the transaction data, we create something called a “digital signature”:

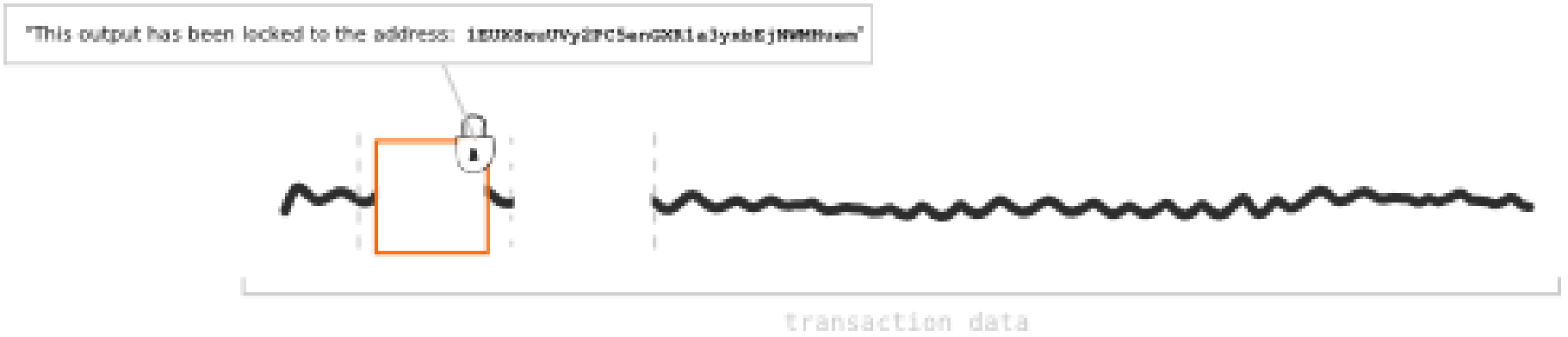


UITiS
UIT Information Security Lab

- Obviously I lied about that function we used too. But no fear, there is actually one that compares an [address] with a [digitalsignature], and it's called **CHECKSIG**.

Why do we use digital signatures in blockchain?

- Because when you make a transaction, you need to unlock the outputs you're trying to use. This is done by showing that you "own" the output, and you do this by *showing that you know the private key of the address the output is locked to*:



Why do we use digital signatures in blockchain?

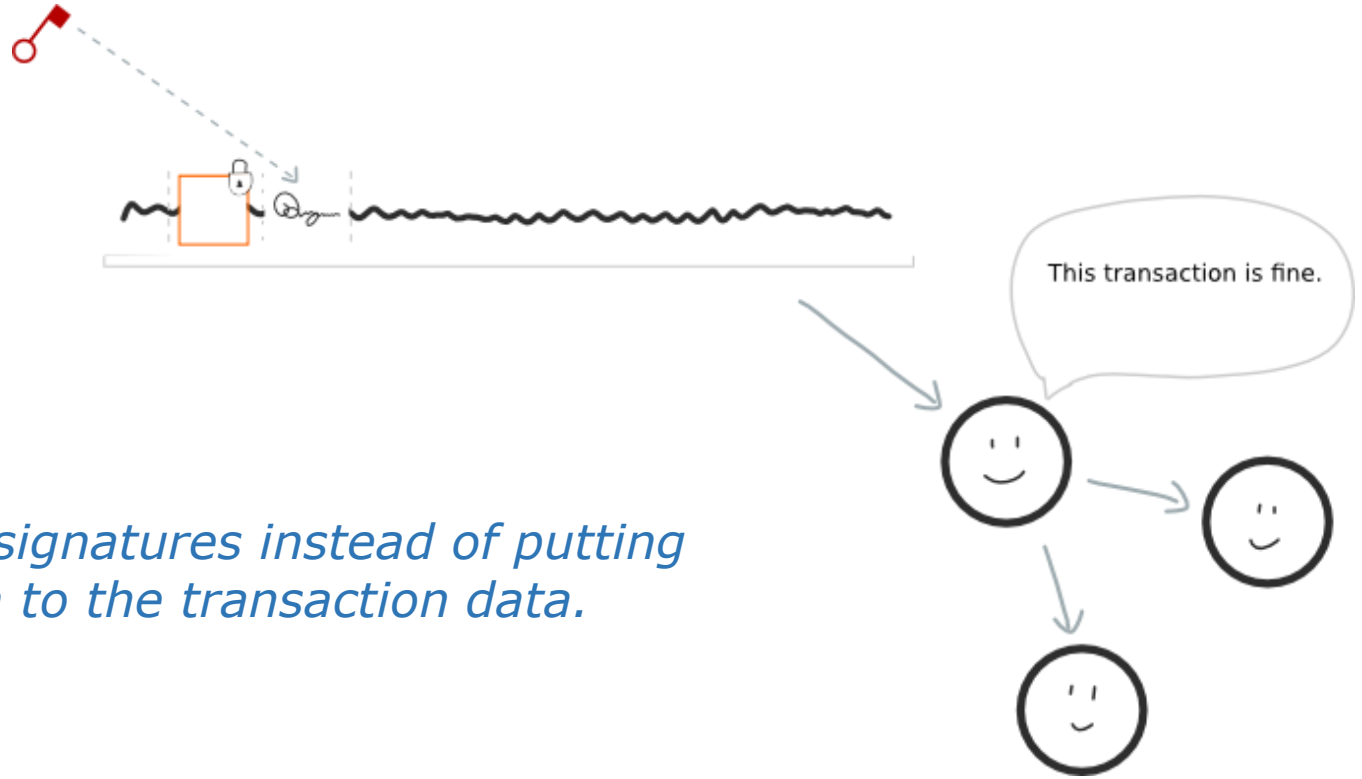
- But if you put your private key in to the transaction data, everyone on the network will be able to see it:

- And if anyone gets your private key, they can use it to unlock and spend any other outputs that have been locked to that same address.
- So how can we unlock outputs without giving our private key away?

UITiS
UIT Information Security Lab

Why do we use digital signatures in blockchain?

- A digital signature can be used to unlock outputs, because it shows that we know the private key of an address.
- Using a digital signature means that we don’t give our private key away to the network:



This is why we use digital signatures instead of putting our private keys directly in to the transaction data.

UITiS
UiT Information Security Lab

Blockchain summary

Blockchain technology provides the basis for a dynamic shared ledger that can be applied to:

- save time when recording transactions between parties,
- remove costs associated with intermediaries,
- and reduce risks of fraud and tampering.

Blockchains and Distributed Ledgers

- Ledgers
  - Ledgers are historically centralized and private.

- Blockchains are Distributed Ledgers
  - Blockchain is a shared ledger technology allowing any participant in the business network to see the system of record (ledger).

UIT Information Security Lab

How Blockchains Work: Basics

- Chronological Ledger
  - Transactions often “pseudo-anonymous”
  - Transactions are grouped together in “blocks”
  - Transactions are logged and stamped with information about the time, amount, and participants as if a notary is present at every transaction
- Blockchain is not centralized (does not have one owner), therefore there are strict rules about how it must be maintained

How Blockchains Work: Maintenance

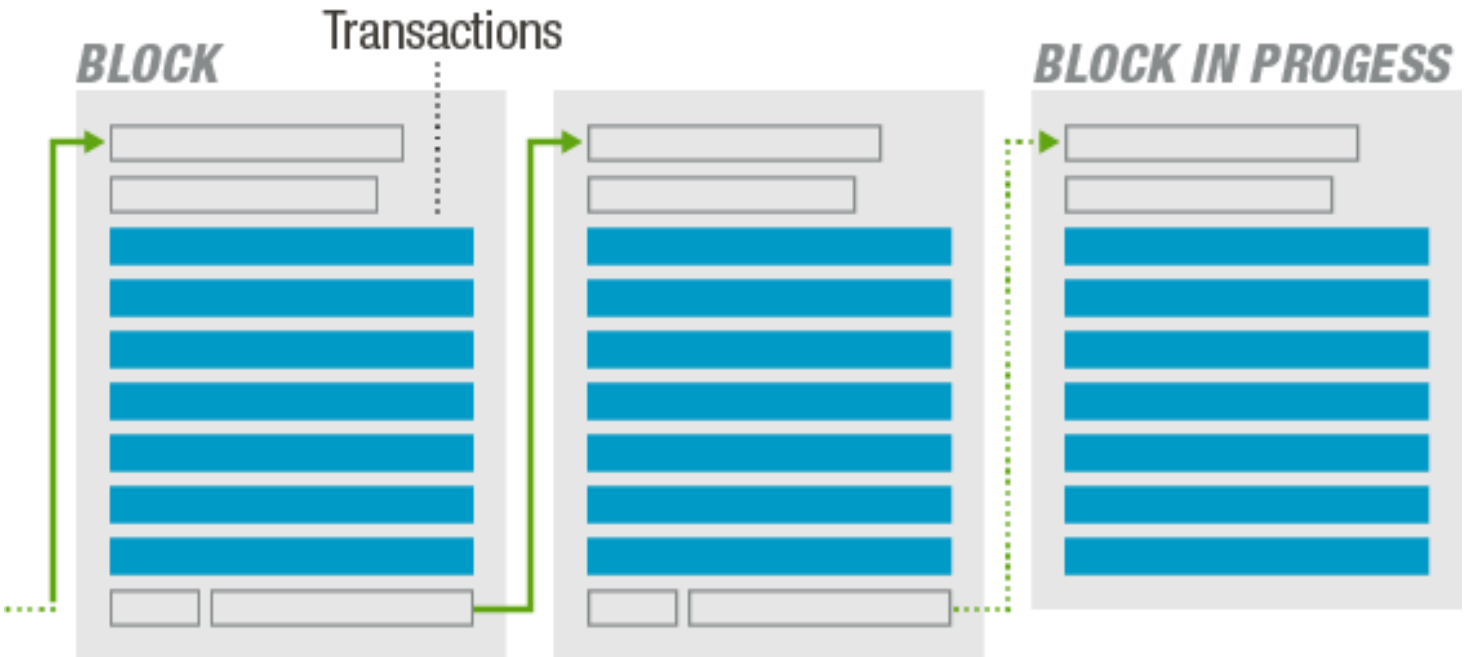
- The individuals who maintain and update the Blockchain are “miners”, and they are paid a reward

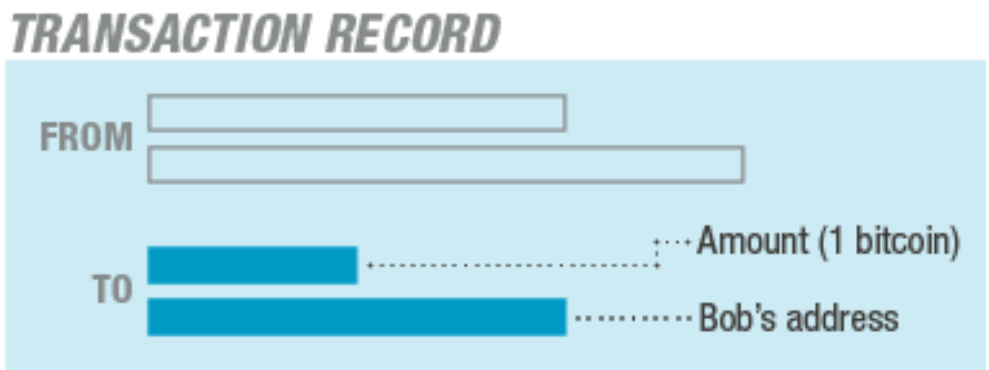
- The Miners process transactions by:
  - Solving a complex mathematical problem
  - Sending transactions to other nodes to be verified.

How Blockchains Work: Hashing

- When all miners agree the problem has been solved correctly, the block is added to the chain and is visible to the entire network

- The unbroken Hash (seal) confirms that the block, and therefore every block before it, is legitimate.

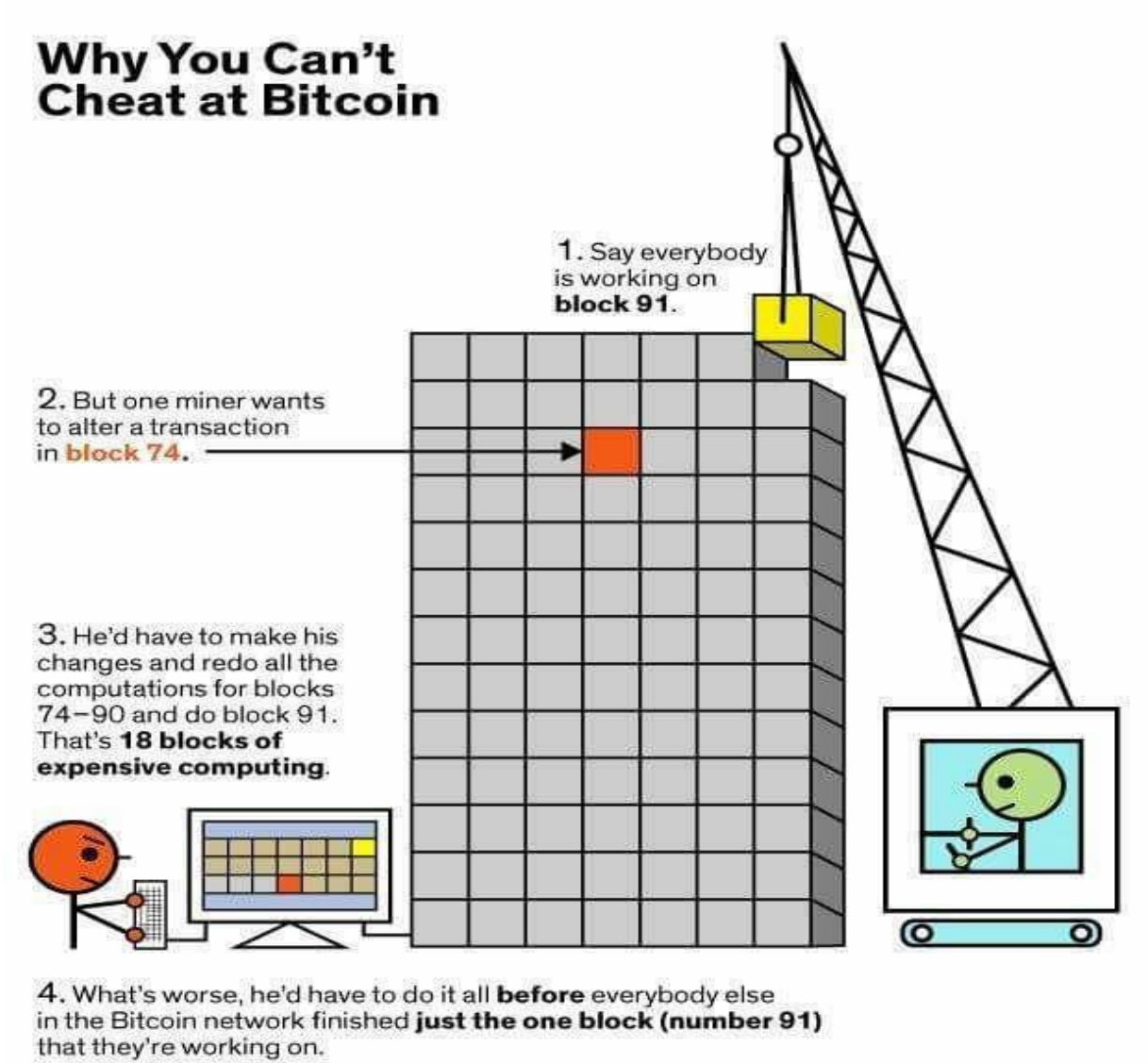




UIT Information Security Lab

How Blockchains Work: Hashing (cont.)

- Recall: Transactions must be validated by other network miners
- Miners incentivized to add “valid” transactions via a reward; invalid transactions are rejected, and thus, no reward is given



UIT Information Security Lab

Consensus

- Any computer that connects to the blockchain network is called a **node**. Nodes that fully verify all of the rules of blockchain are called **full nodes**.
- Every node in the blockchain network (Bitcoin, Ethereum,…) holds a copy of the blockchain.
- We need to make sure that nodes cannot tamper with the blockchain, and we also need a mechanism to check whether a block is valid or not.

Consensus

- **Each full node** in the Bitcoin network independently stores a block chain containing only blocks validated by that node.
- When **several nodes all have the same blocks** in their block chain, they are considered **to be in consensus**.
- The validation rules these nodes follow to maintain consensus are called consensus rules.

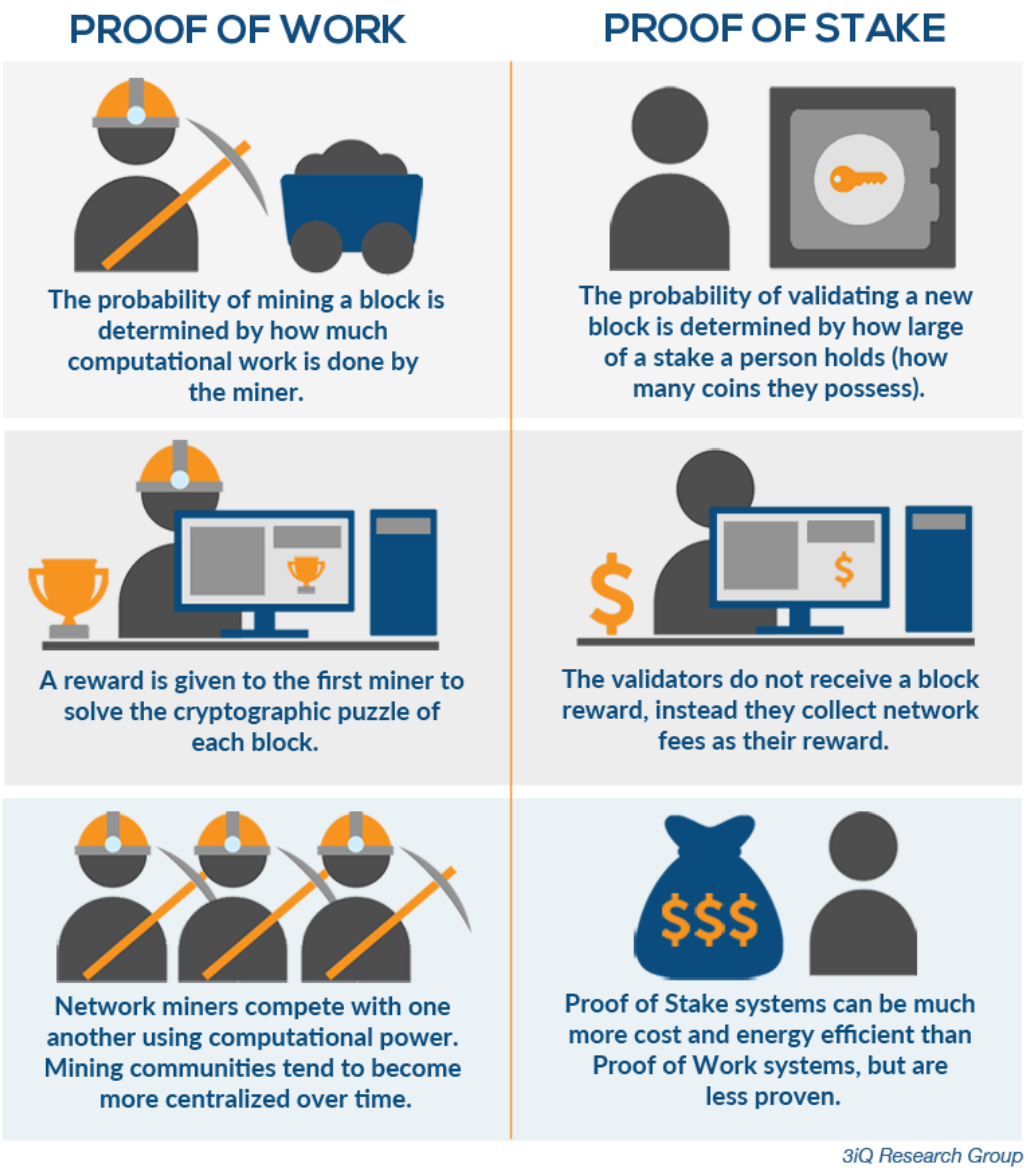
Consensus rules

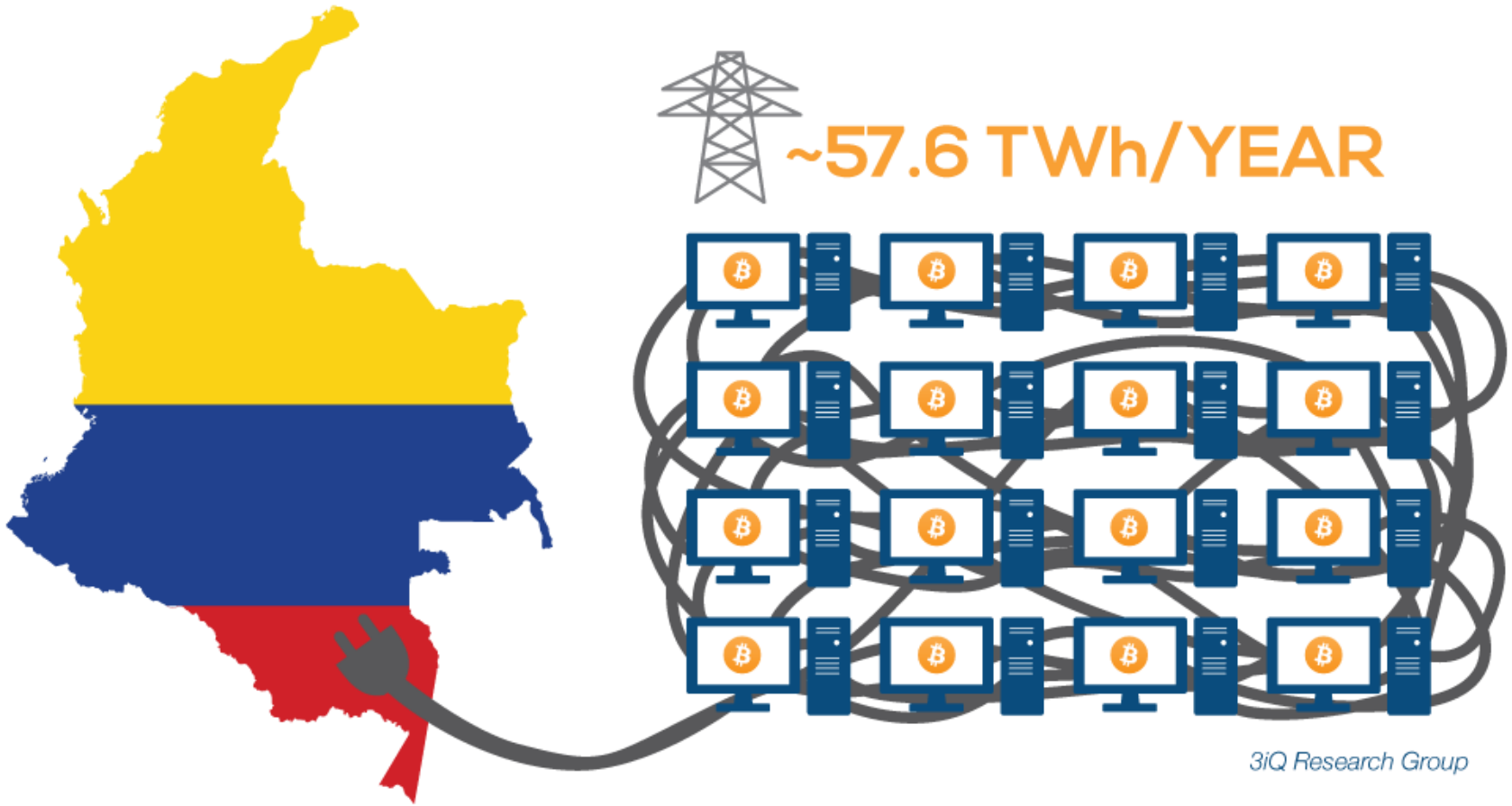
- Full nodes download every block and transaction and check them against Bitcoin's **consensus rules**.
- Here are examples of **consensus rules**, though there are many more:
  - Blocks may only create a certain number of bitcoins.
  - Transactions must have correct signatures for the bitcoins being spent.
  - Transactions/blocks must be in the correct data format.
  - Within a single block chain, a transaction output cannot be double-spent.

Consensus Protocol

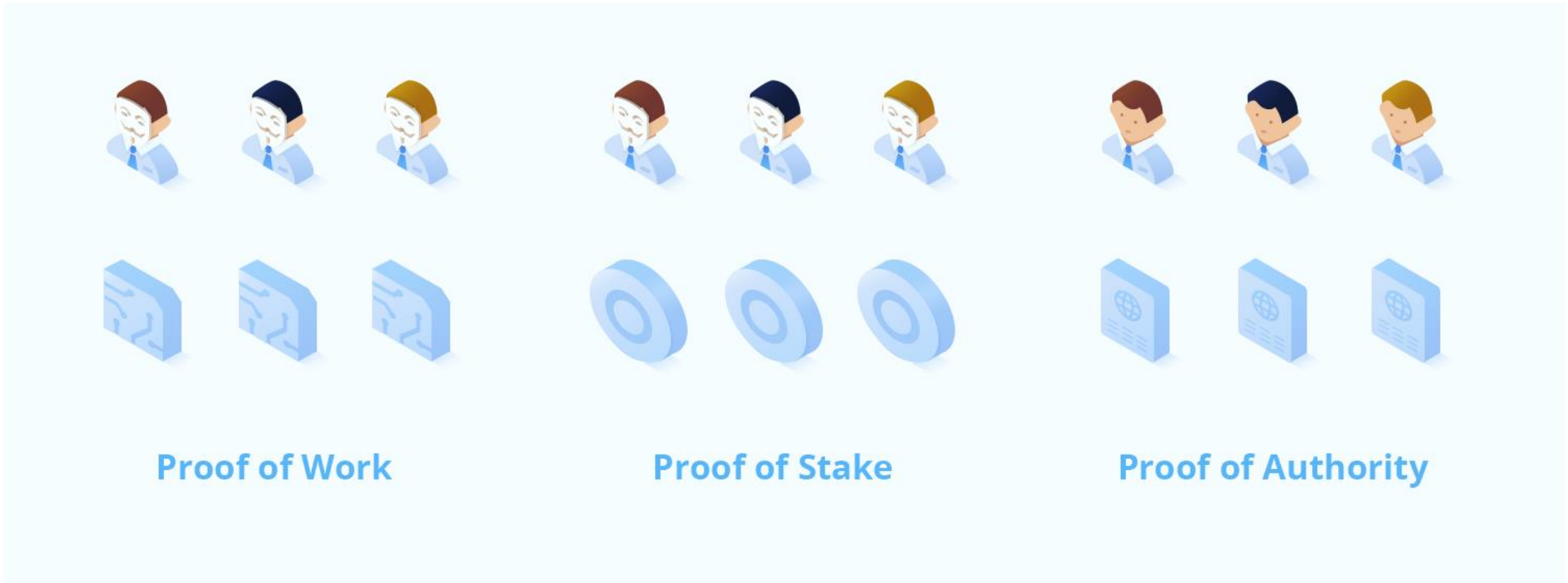
- PoW (Proof of Work)
- PoS (Proof of Stake)
- PoET (Proof of Elapsed Time)
- SBFT (Simplified Byzantine Fault Tolerance)
- PoA (Proof of Authority)

PoW vs. PoS


PROOF OF WORK
PROOF OF STAKE
The probability of mining a block is determined by how much computational work is done by the miner.
The probability of validating a new block is determined by how large of a stake a person holds (how many coins they possess).
A reward is given to the first miner to solve the cryptographic puzzle of each block.
The validators do not receive a block reward, instead they collect network fees as their reward.
Network miners compete with one another using computational power. Mining communities tend to become more centralized over time.
Proof of Stake systems can be much more cost and energy efficient than Proof of Work systems, but are less proven.
3iQ Research Group



The Bitcoin network, for example, requires an annual energy consumption comparable to that of **Colombia** (57.6 TWh annually).
Link: Data as at April 3rd, 2018. Retrieved from https://digiconomist.net/bitcoin-energy-consumption


Proof of Work
Proof of Stake
Proof of Authority

PoW (Proof of Work)

- PoW requires a huge amount of energy to be expended, given the computationally heavy algorithm.
- PoW has a high latency of transaction validation, and the concentration of mining power is located in countries where electricity is cheap.
- In terms of the network security, PoW is susceptible to the '51% attack', which refers to an attack on a blockchain by a group of miners controlling more than 50% of the network's computing power.

PoS (Proof of Stake)

- The **Proof of Stake** algorithm is a generalization of the **Proof of Work** algorithm.
- In PoS, the nodes are known as the **'validators'** and, rather than **mining** the blockchain, they validate the transactions to earn a transaction fee.
- Nodes are randomly selected to validate blocks, and the probability of this random selection depends on the amount of stake held.
- So, if node X owns 2 coins and node Y owns 1 coin, node X is twice as likely to be called upon to validate a block of transactions.
- The PoS algorithm saves expensive computational resources that are spent in mining under a PoW consensus regime.

PoA (Proof of Authority)

- Proof-of-Authority (PoA) is a consensus algorithm which can be used for permissioned ledgers.
- It uses a set of **'authorities'**, which are designated nodes that are allowed to create new blocks and secure the ledger.
- Ledgers using PoA require sign-off by a majority of authorities in order for a block to be created.

PoET (Proof of Elapsed Time)

- Developed by Intel, the Proof of Elapsed Time consensus algorithm emulates the Bitcoin-style Proof of Work.
- Instead of competing to solve the cryptographic challenge and mine the next block, (as in the Bitcoin blockchain), the PoET consensus algorithm is **a hybrid of a random lottery & first-come-first-serve** basis. In PoET, each validator is given a random wait time.
- **Hyperledger's Sawtooth** implementation is an example of PoET at work
- The validator with the shortest wait time for a particular transaction block is elected the leader. This "leader" gets to create the next block on the chain.

SBFT (Simplified Byzantine Fault Tolerance)

- **The Simplified Byzantine Fault Tolerant** consensus algorithm implements an adopted version of the Practical Byzantine Fault Tolerant (PBFT) algorithm

Conclusion

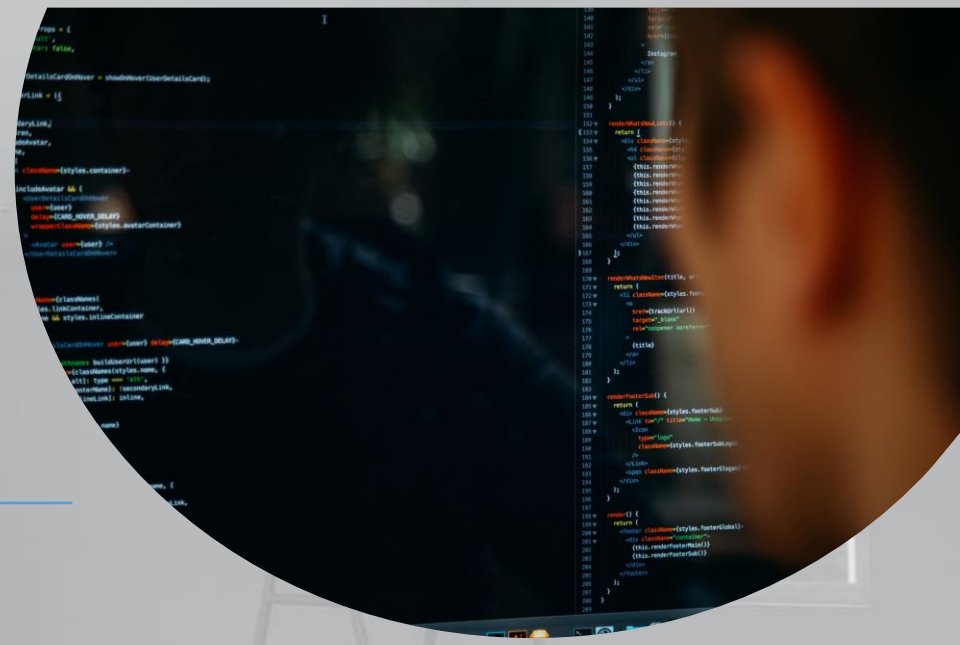
❖**In essence, a blockchain is:**

- A distributed ledger
- A consensus protocol
- A membership protocol

UITiS
UIT Information Security Lab

UIT Information Security Lab

Protecting our future. We make people feel safe



Address: E8.1, University of Information Technology, Linh Trung Ward, Thu Duc Dist., HCMC.

Website: ***http://inseclab.uit.edu.vn/***

Fanpage: https://www.facebook.com/inseclab

Thank You

MSc. Phan The Duy – Email: duypt@uit.edu.vn

UIT Information Security Lab

Ver2.16 - **August 2018**