BBM 443
Fundamentals of Blockchain

BBM 443 – Fundamentals of Blockchain

- Instructor : Adnan Özsoy
- Thursday 09.30 – 12.00 @ Remote
- adnan.ozsoy@hacettepe.edu.tr
- Office : 114

Course Objectives

“This course provides a thorough understanding of the fundamental concepts and recent advances in blockchain and cryptocurrencies. The main objective is to provide students practical and theoretical foundations to use and develop applications using the blockchain technology and can solve challenging problems in cryptocurrencies.”

Not Course Objectives

- Make you rich $$$
- Give insider information
- Create money out of thin air
- Teach you magic

Course Outline

Date	Weeks	Topics	
8.Oct	1.	Lecture 1: Introduction	
15.Oct	2.	Lecture 2: Blockchain Fundamentals	
22.Oct	3.	Lecture 3: Bitcoin Mechanics	
29.Oct	4.	HOLIDAY	
5.Nov	5.	Lecture 4: Wallets, Mining, Pools	
12.Nov	6.	Lecture 5: Ethereum and Smart Contracts	
19.Nov	7.	Lecture 6: Distributed App Development	
26.Nov	8.	Lecture 7: Hyperledger - Use cases	Project out
3.Dec	9.	Lecture 8: Blockchain Security	
10.Dec	10.	Lecture 9: Consensus Algorithms	
17.Dec	11.	Lecture 10: Scaling Blockchain	Project Due
24.Dec	12.	Lecture 11: Presentations	
31.Dec	13.	Lecture 12: Presentations	
7.Jan	14.	Final Review	Report Due

Resources

Text Book

- Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction Hardcover, Arvind Narayanan, Joseph Bonneau, Edward Felten,Andrew Miller,Steven Goldfeder, 2016
Free pre-publication draft @ http://bitcoinbook.cs.princeton.edu/
- Mastering Bitcoin: Programming the Open Blockchain 2nd Edition, Andreas M. Antonopoulos, 2017

Reference Material

- Blockchain Berkeley : https://blockchain.berkeley.edu/decal/sp19/fund/
- Blockchain Princeton : http://bitcoinbook.cs.princeton.edu/
- Mastering Ethereum, by Andreas M. Antonopoulos, Gavin Wood https://ethereumbook.info/, https://github.com/ethereumbook/ethereumbook/blob/develop/book.asciidoc

Grading

Project	: 20 % - 1 project - individual
Attendance	: 5 % - not mandatory
Exam	: 50 %
Final Report	: 25 % - Group of 4-5

Project

- Project
  - Solidity
  - Individual project

Final Report

- Need to start immediately
- Group Project
- Pick a topic where Blockchain can help
- Review Literature, compare state-of-the-art technologies
- Propose a solution, model,
- Present your work
- Bonus for implementation

Communication & Course Material

https://piazza.com/hacettepe.edu.tr/fall2020/bbm443

Basic Concepts

How all started???

Pre Bitcoin-2009:

Libertarian Dreams and Ideals

Libertarian Dreams

- With the advance of technology in the 1980s and 90s, the Cypherpunk movement came into being
- Cypherpunk Manifesto: “Privacy is necessary for an open society in the electronic age.”
- Roots in libertarianism and cryptography

Libertarianism is a political ideology advocating the non-aggression principle and laissez faire government
(transactions between private parties are free from government intervention such a
regulation, privileges, tariffs and subsidies.)

Cryptography is the science of securing communication in the presence of third parties

Cypherpunks and Crypto-Anarchists

- Cypherpunks were obsessed over how technology would change the relationship between the individual and the state
- Hopeful about the new tools people had but concerned about how people could protect their personal information and maintain their privacy from government

Untraceable Electronic Cash †

(Extended Abstract)

David Chaum[1] *Amos Fiat*[2] *Moni Naor*[3]

[1] Center for Mathematics and Computer Science
Kruislaan 413, 1098 SJ Amsterdam, The Netherlands

[2] Tel-Aviv University
Tel-Aviv, Israel

[3] IBM Almaden Research Center
650 Harry Road, San Jose, CA 95120

CRYPTO 1988

DigiCash™

David Chaum

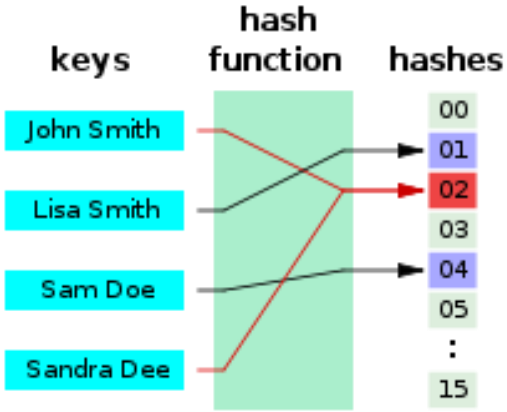
Photo: Declan McCullagh (2002)

Changing Money

- The existing financial system was viewed as one of the greatest threats to individual privacy
- DigiCash is the most famous example of the early cryptocurrencies
- The inventor of DigiCash, David Chaum, used public-key cryptography
- However, DigiCash was a central organization, meaning that Chaum's company needed to confirm every digital signature.
- Eventually, Chaum's company went bankrupt and DigiCash went down with it

Crypto-Innovation

- Cypherpunks also worked on technological innovations, including the cryptographic hash function.
- A hash function is a math equation that is easy to solve but hard to reverse-engineer.
- The early experiments of the Cypherpunks continued to hit hurdles that resulted in failure



2009-2010:
The early development of Bitcoin

WHO IS?

SATOSHI NAKAMOTO ฿

Satoshi Nakamoto and Bitcoin

- Satoshi Nakamoto is the anonymous creator of Bitcoin who wrote a nine-page white paper that brilliantly combined all previous efforts to create a self-sustaining digital money.
- Although some, disheartened by history, were bearish on the currency, a few of the early pioneers supported the project as the solution to their past problems

Genesis Block

- Genesis block mined Jan 3, 2009
- The coinbase of the genesis block references a story in the Times of London newspaper involving the Chancellor bailing out banks
- First bitcoin transaction on Jan 12, 2009 with Hal Finney

Block 0

Short link: http://blockexplorer.com/b/0

Hash: 000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f
Next block: 00000000839a8e6886ab5951d76f411475428afc90947ee320161bbf18eb6048
Time: 2009-01-03 18:15:05
Difficulty: 1 ("Bits": 1d00ffff)
Transactions: 1
Total BTC: 50
Size: 285 bytes
Merkle root: 4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b
Nonce: 2083236893
Raw block

Transactions

Transaction	Fee	Size (kB)	From (amount)	To (amount)
4a5e1e4baa...	0	0.204	Generation: 50 + 0 total fees	1A1zP1eP5QGefi2DMPTfTL5SLmv7DivfNa: 50

Waited 6 days to create the second block,
A'raf 54, Exodus 31:17

The Six Million Dollar Pizza

- On May 21, 2010, Laszlo Hanyecz purchased $25 worth of pizza for 10,000 BTC
- This was the world’s first ever Bitcoin transaction for a tangible asset
- 10,000 BTC is now equivalent to $64,830,000

Mt. Gox

- In 2010 Mt. Gox was established and consolidated itself as the biggest bitcoin exchange during the beginning stages of bitcoin.
- On 6/19/11, Mt. Gox suffered a significant breach of security that resulted in fraudulent trading and required the site to be shut down for seven days.
- In 2014, Mt. Gox lost 744,408 bitcoins in a theft that went unnoticed for years
- Eventually, Mt. Gox declared bankruptcy

Silk Road
anonymous market

messages 0 | orders 0 | account B0

Search Go

Shop by Category

- Drugs 4,086
  - Cannabis 983
  - Dissociatives 77
  - Ecstasy 318
  - Opioids 350
  - Other 157
  - Precursors 18
  - Prescription 901
  - Psychedelics 587
  - Stimulants 405
- Apparel 82
- Art 5
- Books 778
- Collectibles 15
- Computer equipment 42
- Custom Orders 27
- Digital goods 369
- Drug paraphernalia 152
- Electronics 36
- Erotica 296
- Fireworks 5
- Food 4

100 x Anadrol 50MG Oxymetholone (sealed)
$12.41

1 gram MDMA
$5.89

1/2g Cocaine
$5.44

10 Pieces White Heart 130-150mg MDMA Content
$4.49

Red and White Filter (10 packs x 20 cigarettes)
$1.90

VEGA 100mg Sildenafil citrate 4 tablets
$1.50

10 gram Santa Maria
$11.58

1/4 oz G13
$8.13

Silk Road

- On February 2011, Silk Road opened for business: Silk Road, a Bitcoin marketplace, launched an illicit marketplace for drug deals, called the eBay for drugs.
- On October 2013, the FBI shut down Silk Road, seizing 3.6M dollars worth of bitcoin
- Ross Ulbricht, the founder of Silk Road, is currently serving a life sentence without possibility of parole

Bitcoin Bubble and Burst

Bitcoin's peak in December 2017, $20,089USD

Satoshi worth US$19 billion, the 44th richest person

Cryptocurrency / Blockchain

What is Blockchain

Definition:

A blockchain, originally block chain, is a growing list of records, called blocks, which are linked using cryptography. Each block contains a cryptographic hash of the previous block, a timestamp, and transaction data. By design, a blockchain is resistant to modification of the data.

What is Blockchain

Definition: A blockchain is

-a growing list of records,

-distributed ledger,

-time-stamped,

-cryptographically secured,

-immutable

-consensus-validated,

A new way of storing information

Not an improvement

Not an update

Sample Applications

Insurance - Coverage Proof – Nationwide insurance

Health - MedRec – Patient information share permission

Land Registry

Voting

Donation – BitGive

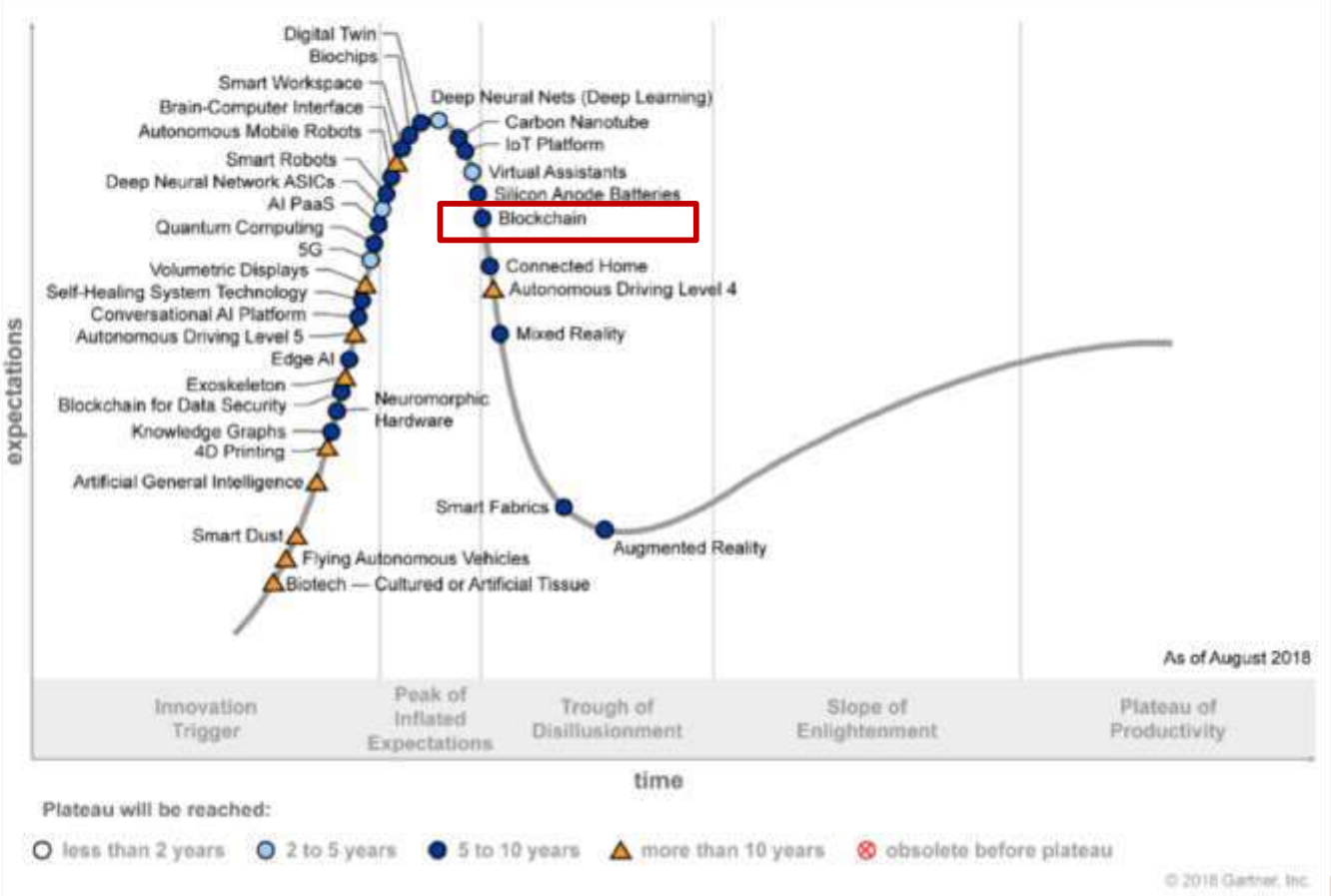
Finance – Bitcoin, Ripple

Supply Chain – IBM Blockchain

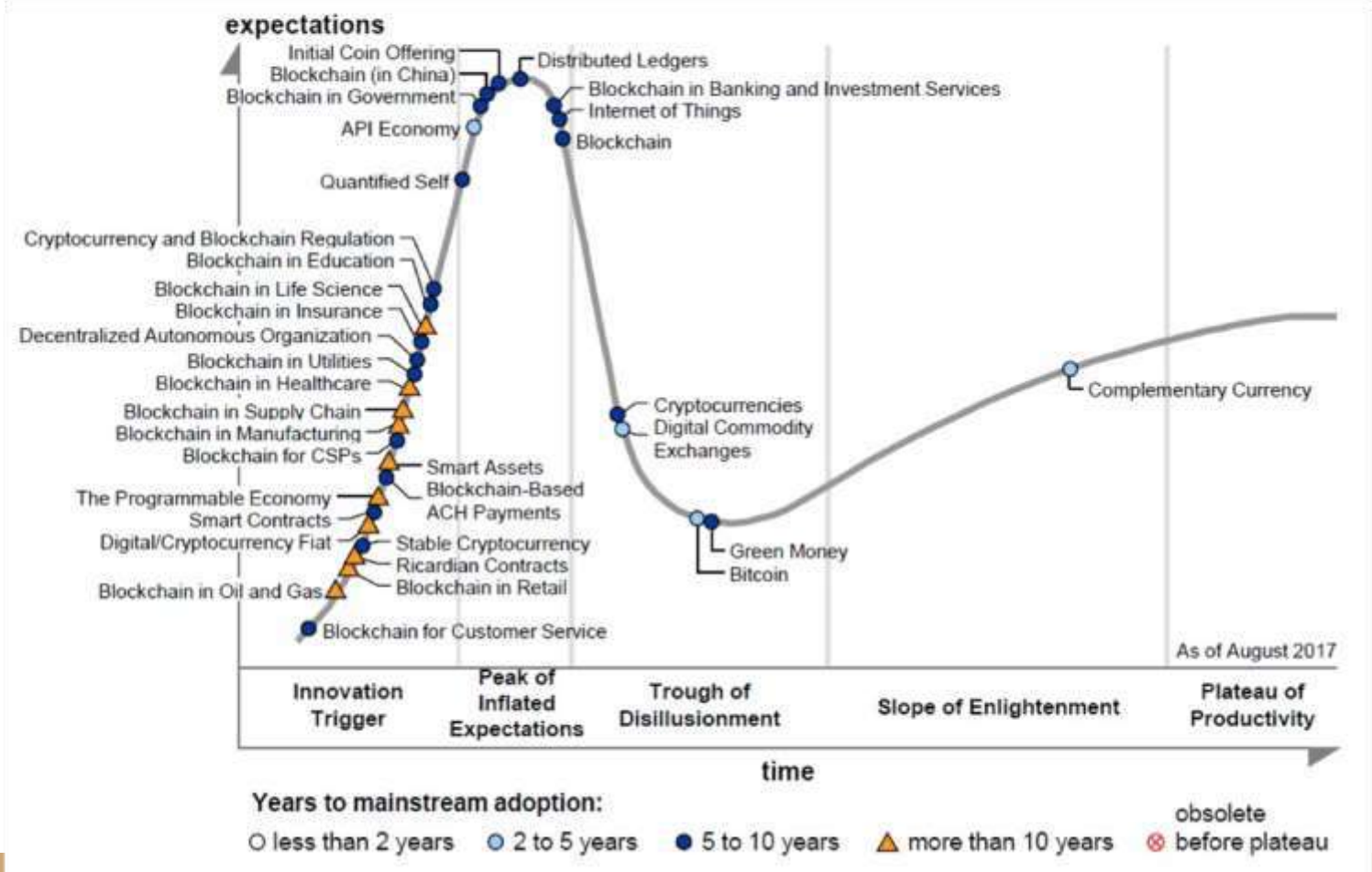
Social Media – Matchpool

Ticket Sale – Guts

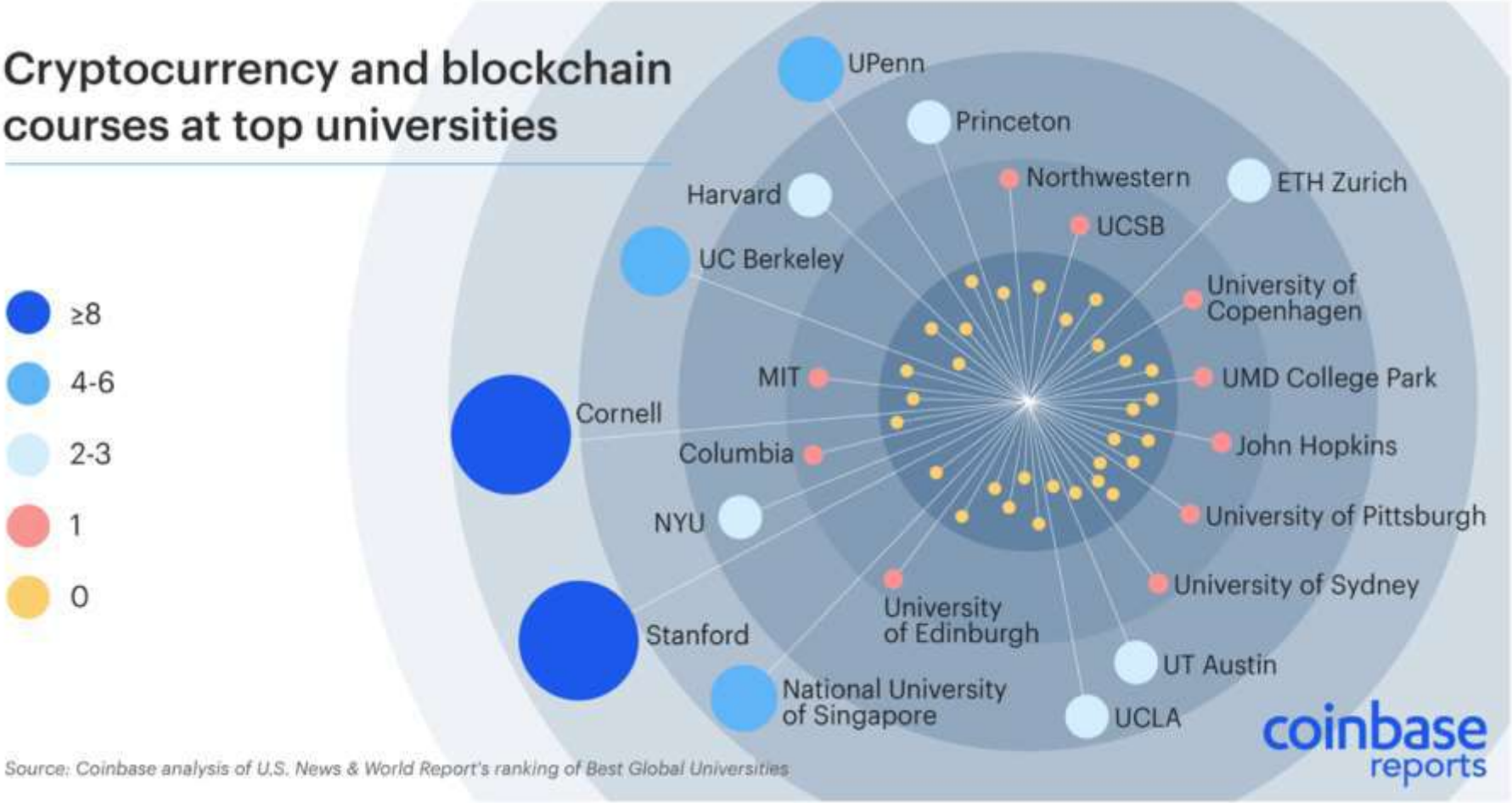
Gartner Hype Cycle for Emerging Technologies, 2018



Gartner Hype Cycle for Blockchain Technologies, 2017



Cryptocurrency and blockchain courses at top universities



Source: Coinbase analysis of U.S. News & World Report's ranking of Best Global Universities

Next class - High Level Overview

To Do:

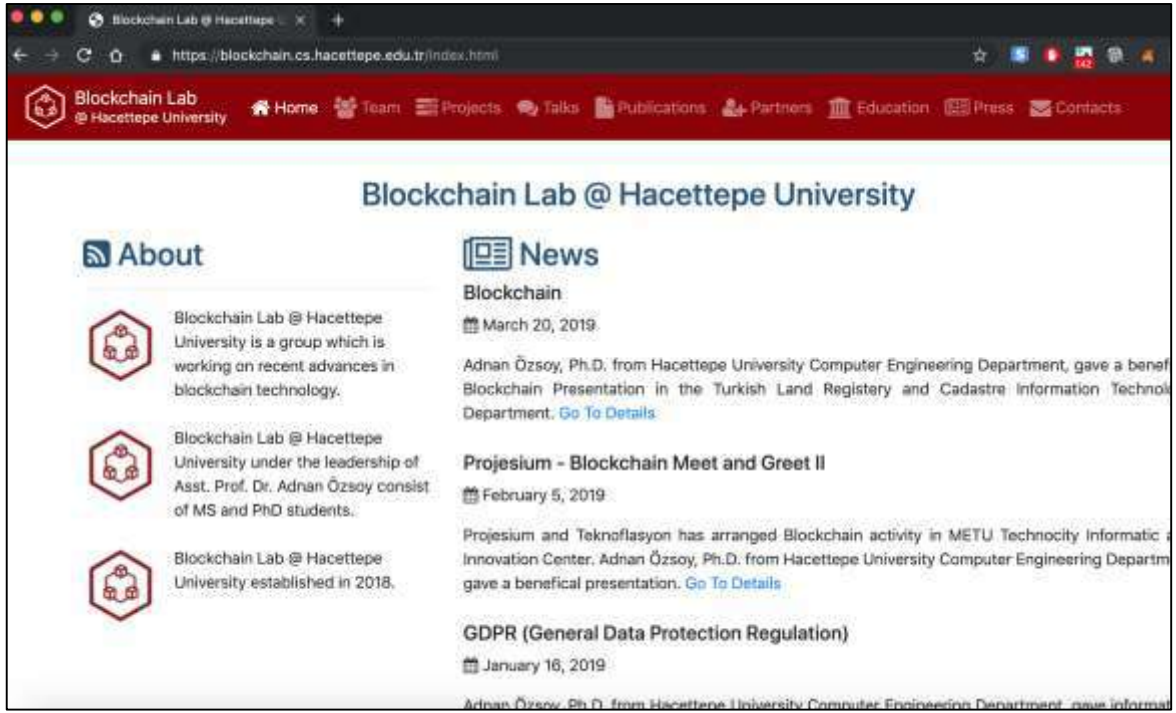
1. Join Piazza

Suggested Readings:

1. How Bitcoin Works in 5 minutes
   - General - https://www.youtube.com/watch?v=t5JGQXCTe3c
   - Technical - https://www.youtube.com/watch?v=l9jOJk30eQs
2. Bitcoin Developer Guide
   - https://bitcoin.org/en/developer-guide#block-chain-overview

Blockchain Lab @ Hacettepe University

http://blockchain.cs.hacettepe.edu.tr



References

Slides mainly adopted from

- Blockchain @ Berkeley : https://blockchain.berkeley.edu/
- Blockchain @ Princeton : http://bitcoinbook.cs.princeton.edu/

Bitcoin Protocol and Consensus: A High Level Overview

Lecture Overview

- Bitcoin basics, much more detailed
- Consensus Build-up
- Mining Overview

Bitcoin Basics
High Level Overview

Basic Concepts - What is Bitcoin?

- Cryptocurrency
  - Cryptocurrency: "A digital currency in which encryption techniques are used to regulate the generation of units of currency and verify the transfer of funds, operating independently of a central bank."
  - Built upon a combination of computer science, cryptography, and economics
- Bitcoin is a cryptocurrency
  - “Bitcoin” can refer to:
    - Bitcoin (uppercase) - the protocol, software, and community
    - bitcoins (conventionally lowercase) - the unit
- Community terminology
  - **"crypto"** – cryptocurrencies
  - **Public - "private blockchain"** - private blockchains, permissioned ledgers, large financial institutions
  - "**blockchain**" - umbrella term

Implications of blockchain technology

- Altcoins (Dash, Dogecoin, Litecoin)
- Bitcoin 2.0 / Ethereum - applying blockchains outside of finance
- Remittances - circumvent traditional banking infrastructure
  - Send money anywhere in the world for 5 cents
- Be your own bank - 100% uptime
- Current hot topics: Governance and “blockchain”
  - Block size debate
  - Banks taking interest in financial implications
- Private blockchains - reduce costs + settlement times in traditional banking infrastructure

BANK

Balances
A 5
B 10
C 5

How can we remove the central authority (bank) ?

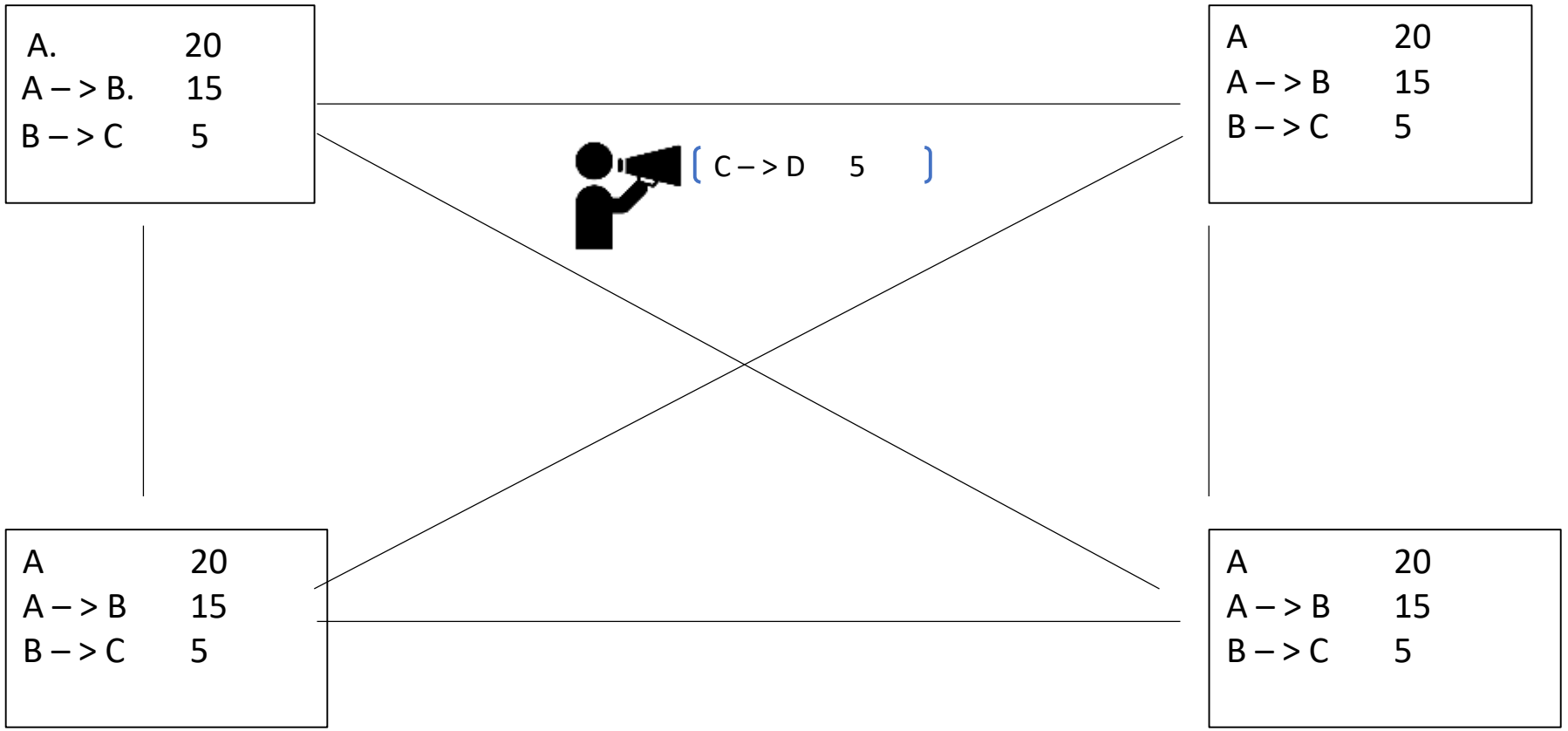
Fully Decentralized – P2P

A.	20
A – > B.	15
B – > C	5

A	20
A – > B	15
B – > C	5

A	20
A – > B	15
B – > C	5

A	20
A – > B	15
B – > C	5



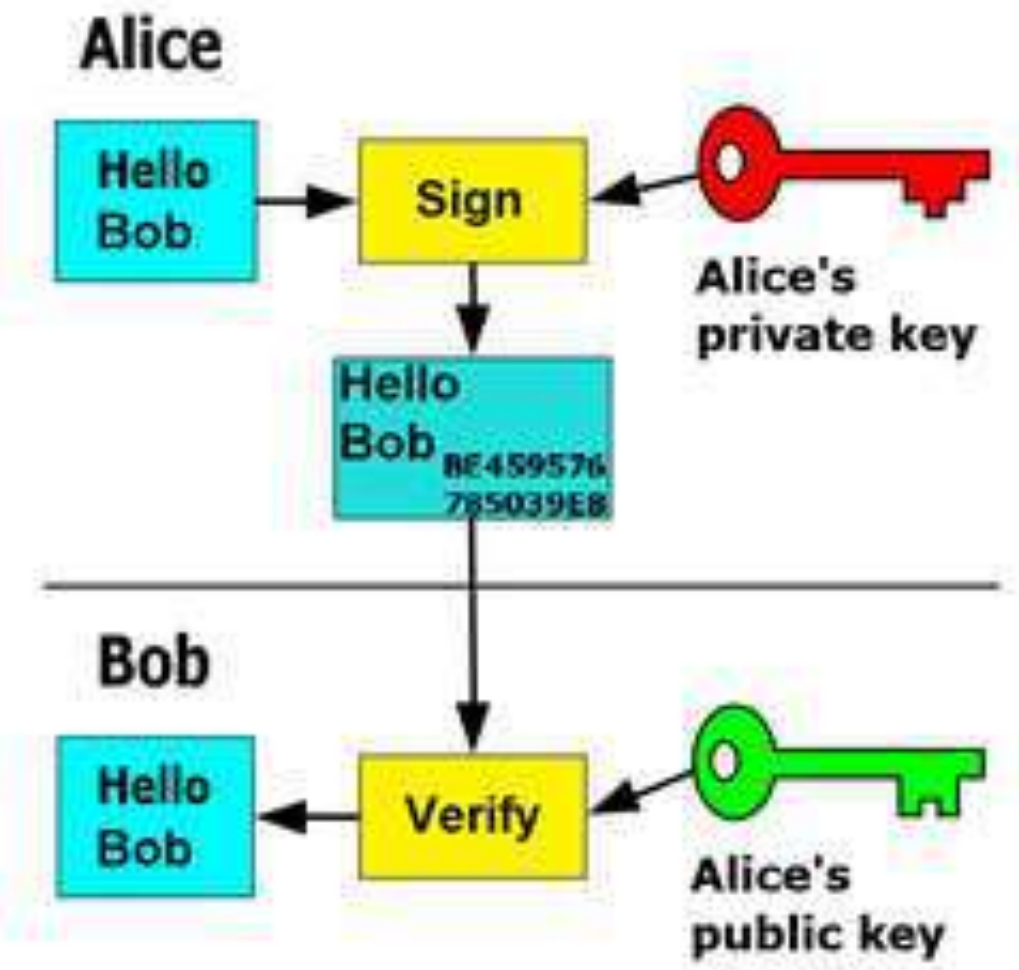
Problems?

1. Does C own 5 units of money?
2. Is the sender really C?

Solution

Asymmetric Encryption
Public / Private Key Crypt.

Asymmetric Encryption-Public / Private Key Cryptography



Courtesy of Wikipedia: https://en.wikipedia.org/wiki/Public-key_cryptography

Asymmetric Encryption-Public / Private Key Cryptography

DEMO

https://www.kerryveenstra.com/cryptosystem.html

https://wizardforcel.gitbooks.io/practical-cryptography-for-developers-book/digital-signatures/ecdsa-sign-verify-examples.html

Basic Concepts - Identity in Bitcoin

- Bitcoin is hidden in the large amount of public keys
  - Users can generate arbitrarily many key pairs
  - Example Address: 1FtQU9X78hdshngJiCBw9tbE2MYpx87eLT
  - 2^160 possible addresses (1,461,501,637,330,902,918,203,684,832,716,283,019,655,932,542,976 addresses)
  - Grains of sand on earth: 2^63
  - 2^126 is actually only 0.0000000058% of 2^160

https://www.bitaddress.org

https://8gwifi.org/ecsignverify.jsp

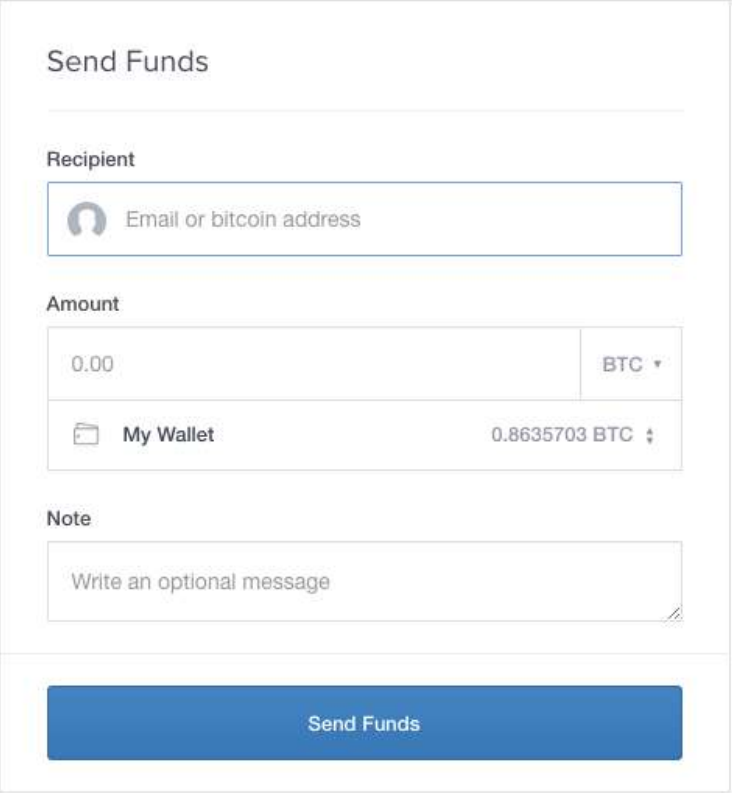
https://bitref.com/

A Bitcoin Transaction - Basic Version

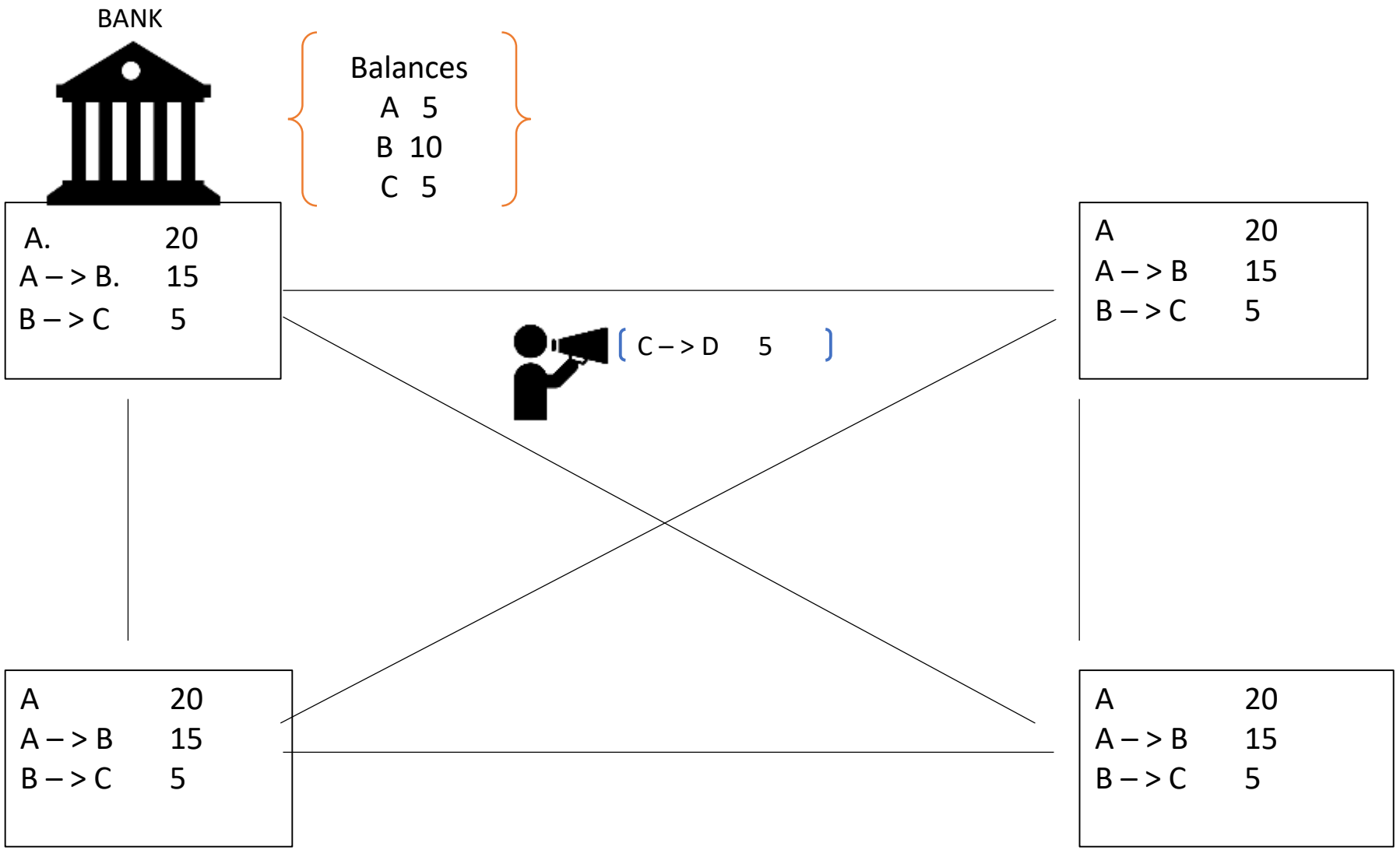
- Bitcoin exists as software
  - Transactions are conducted through wallet software
  - Wallet creation generates a Bitcoin address
- To receive money, you share your address
  - Sender specifies address and amount
- The transaction is broadcast to the network, where "miners" verify it and add it to the transaction history

- https://www.blockchain.com/
- www.coinbase.com
- https://koineks.com/
- https://www.btcturk.com/

1LNnJDNTUXYUfmbiVcngKGg52N8TKNPw6J



Coinbase interface

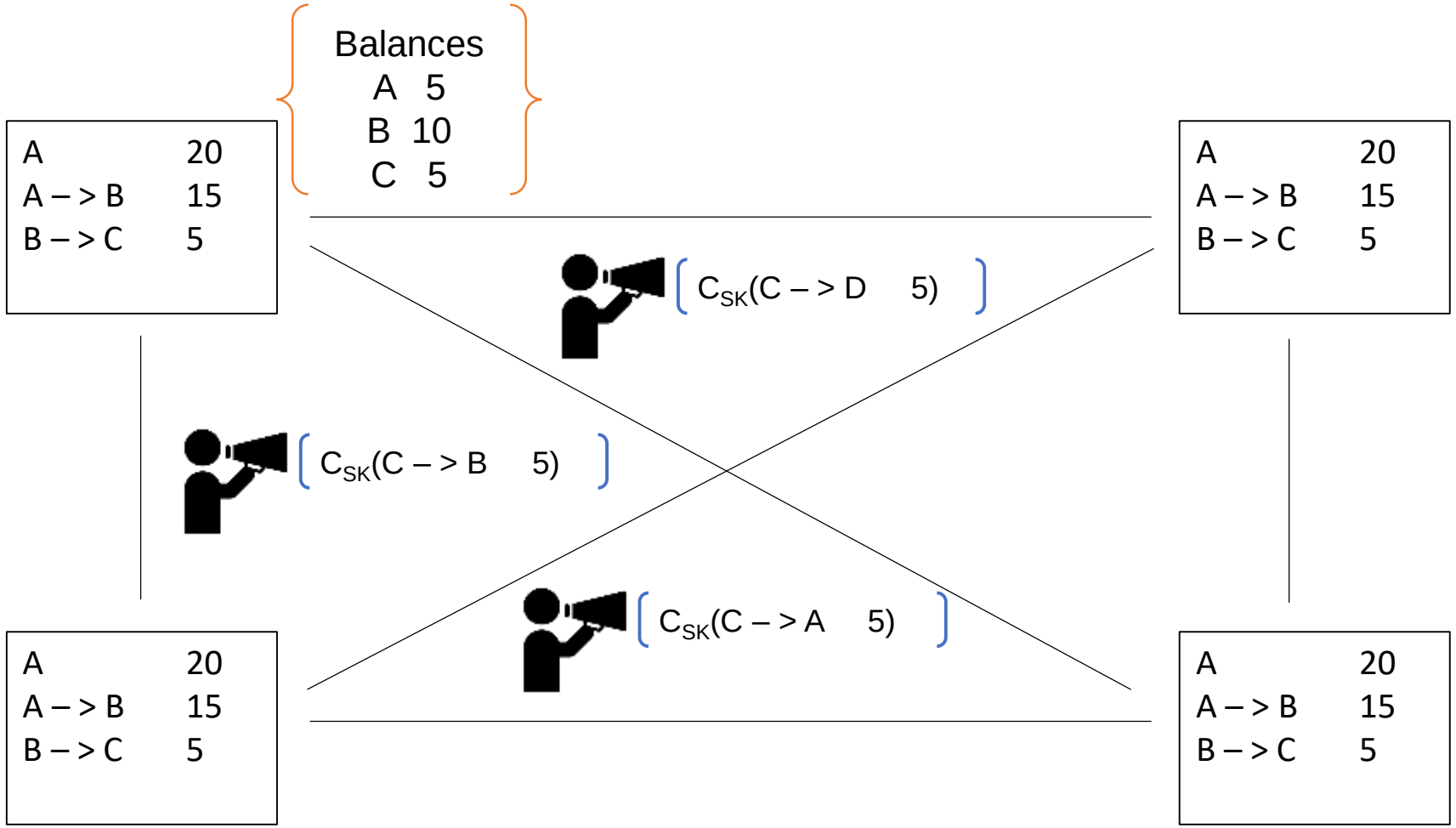


Problems?

1. Does C own 5 units of money?
2. Is the sender really C?

Solution

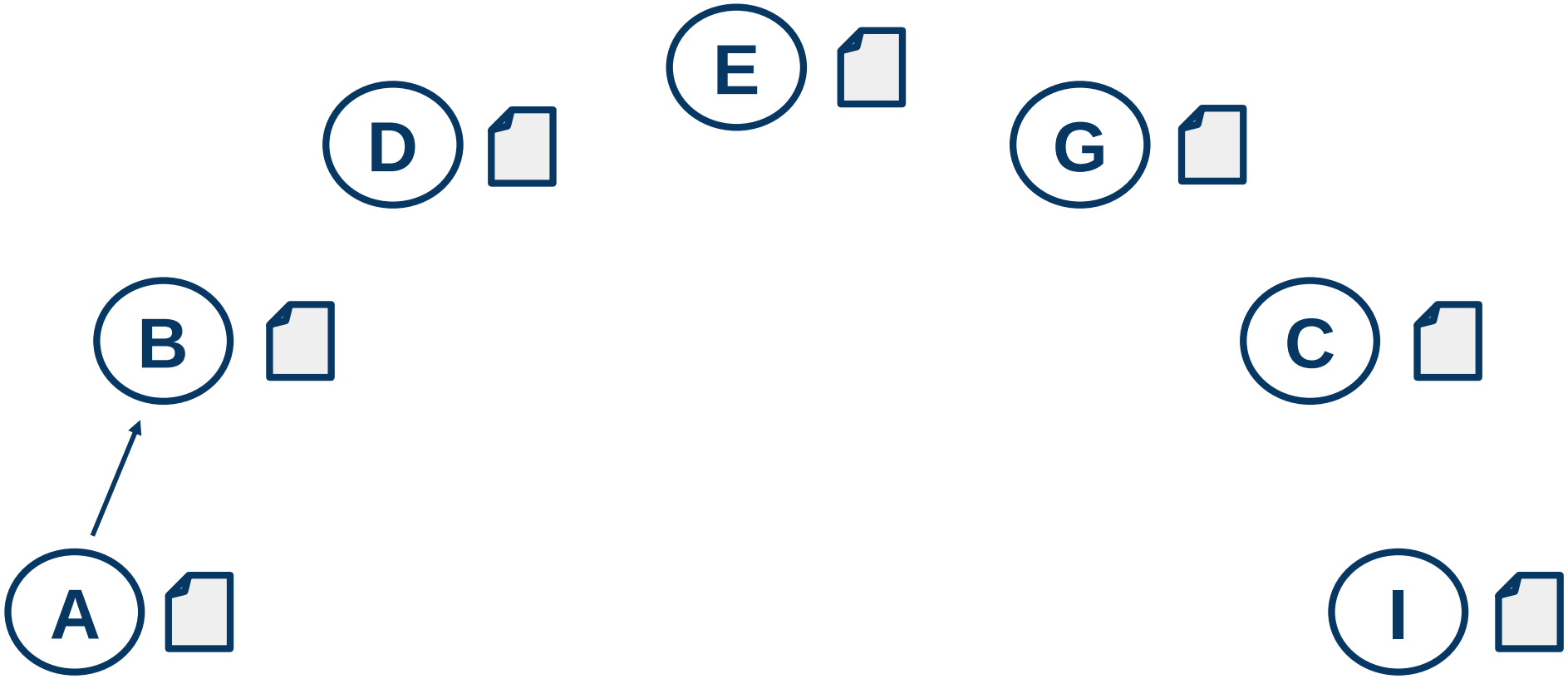
Asymmetric Encryption
Public / Private Key Crypt.



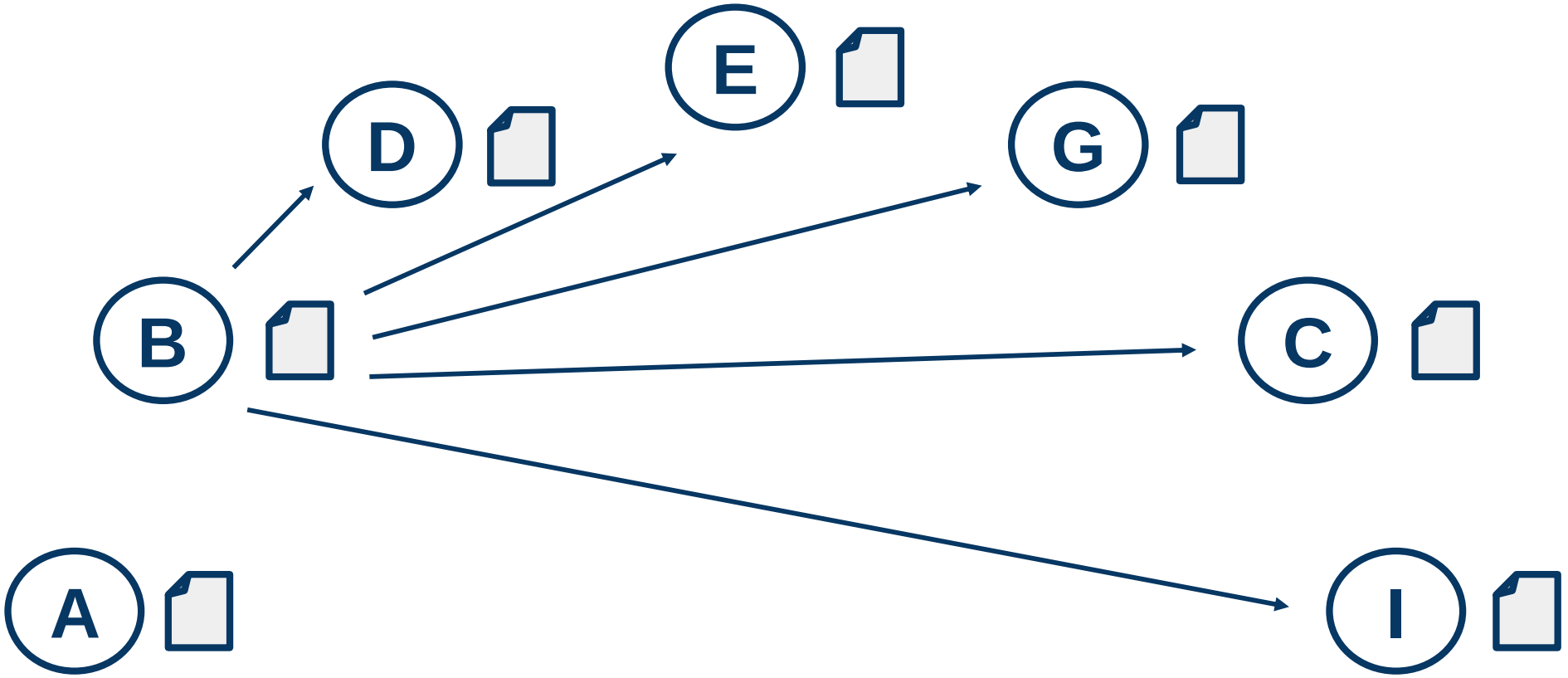
Problems?

1. Can C send the same Money twice?
Double Spending

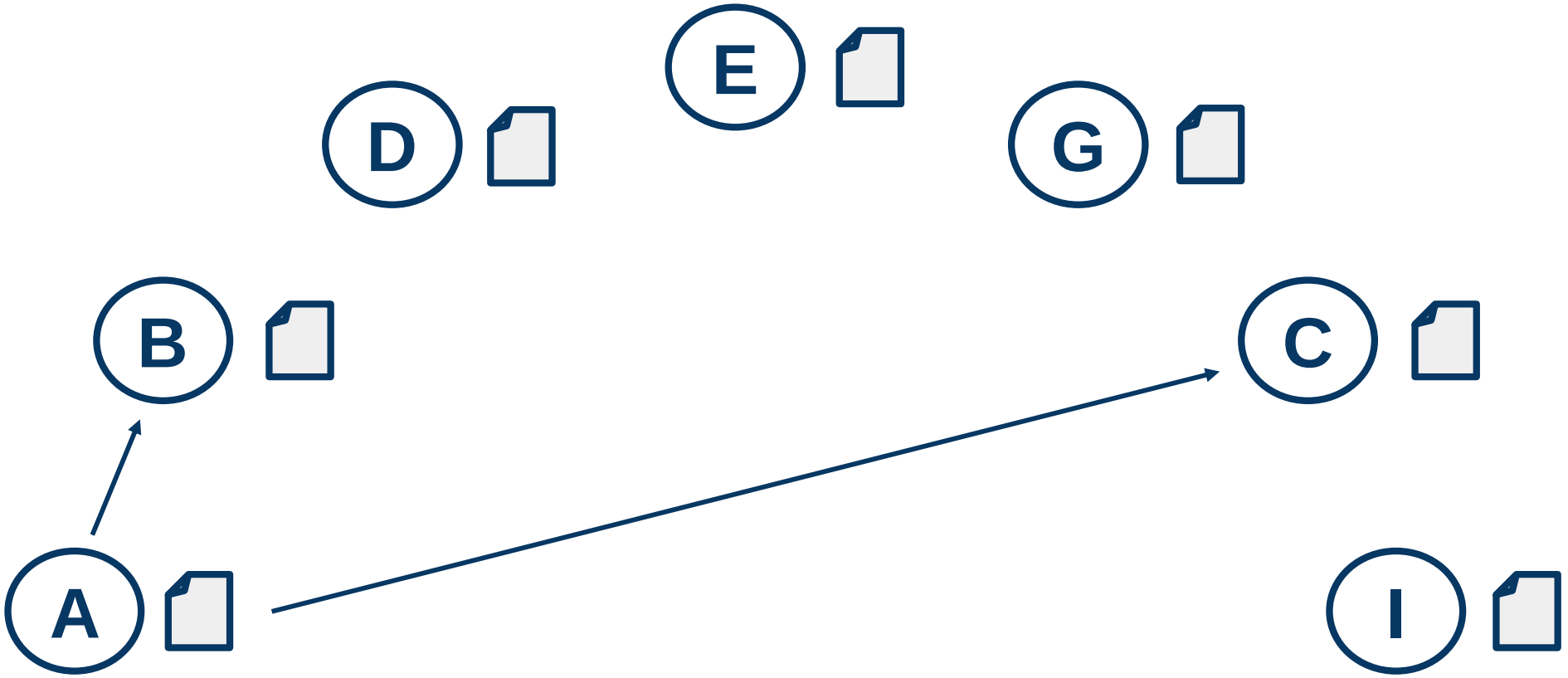
Alice sends her transaction to Bob



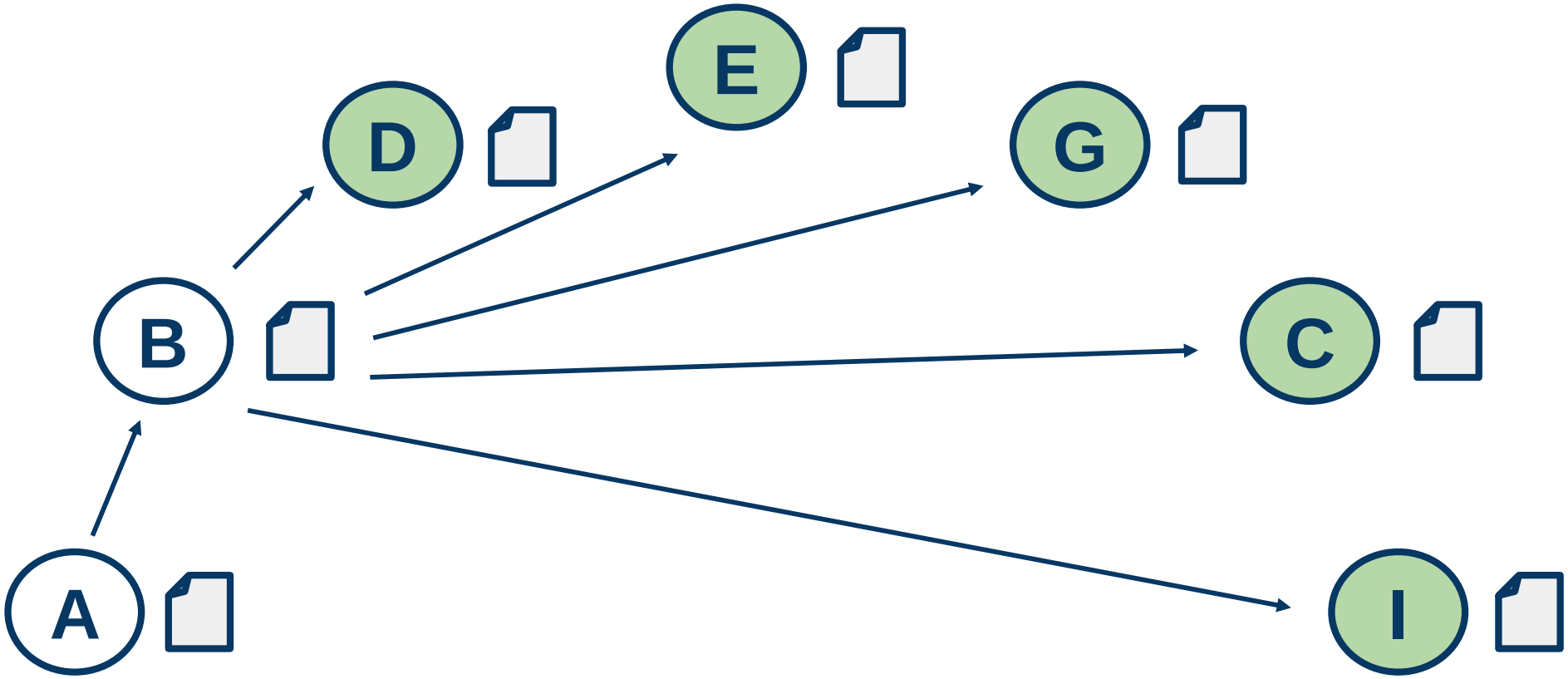
Bob announces the transaction to the world



Alice double spends on Bob and Charlie

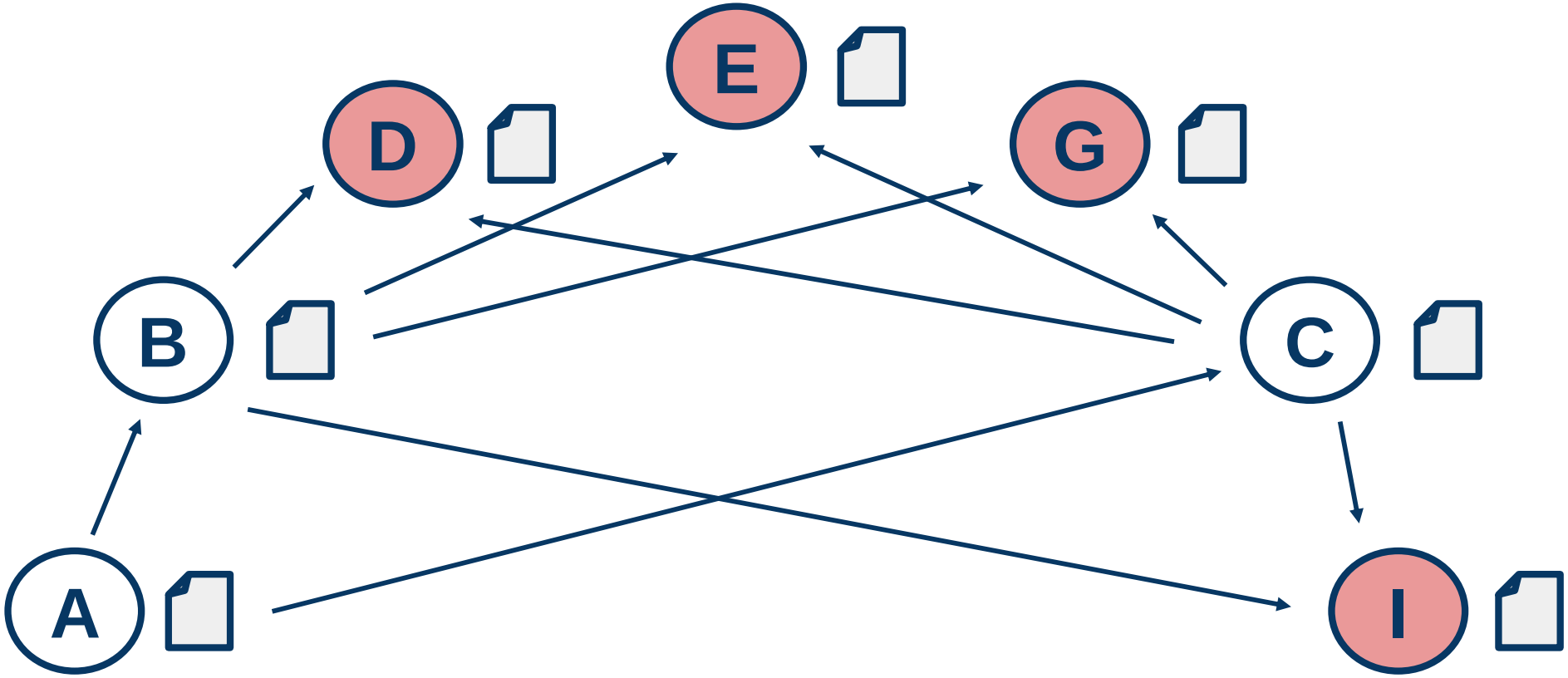


Everyone verifies transactions

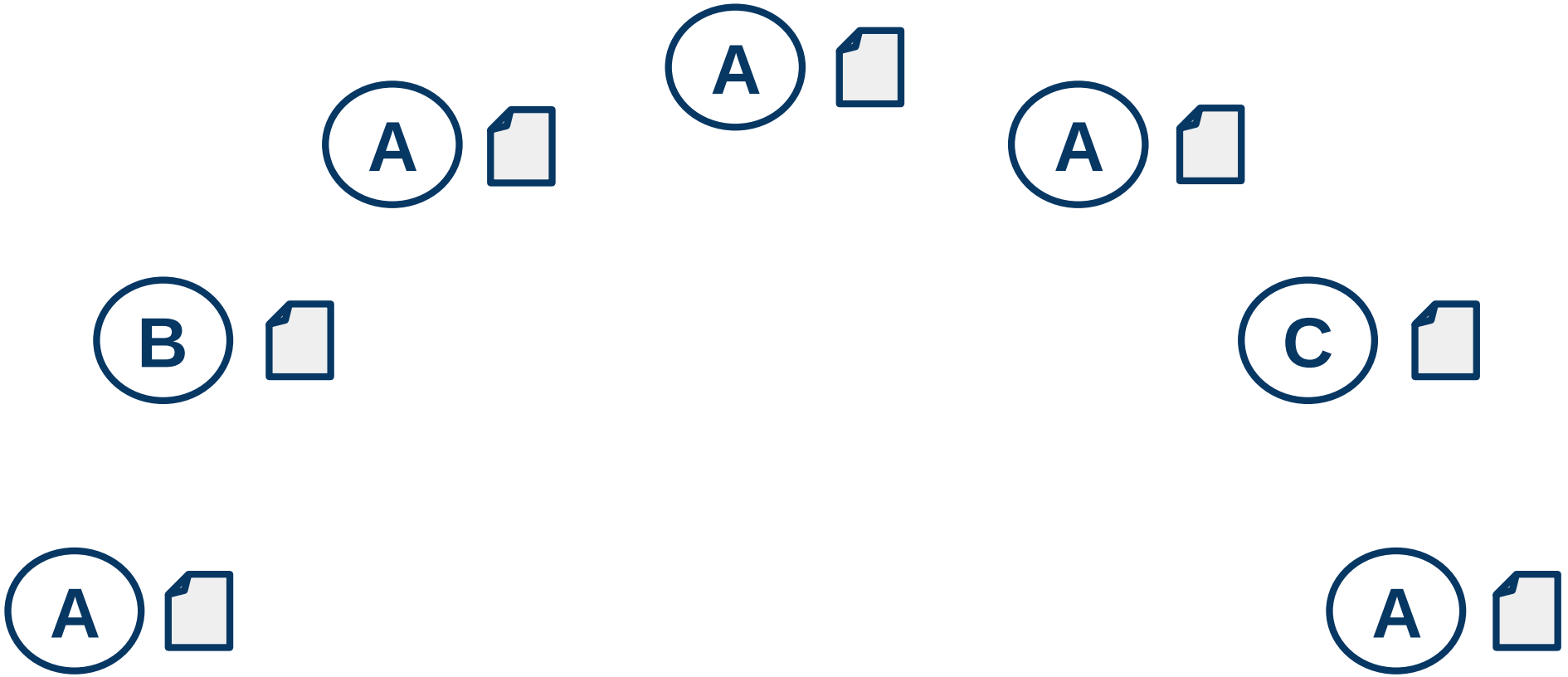


Alice is prevented from double spending

What is wrong with this? Can A still act evil?

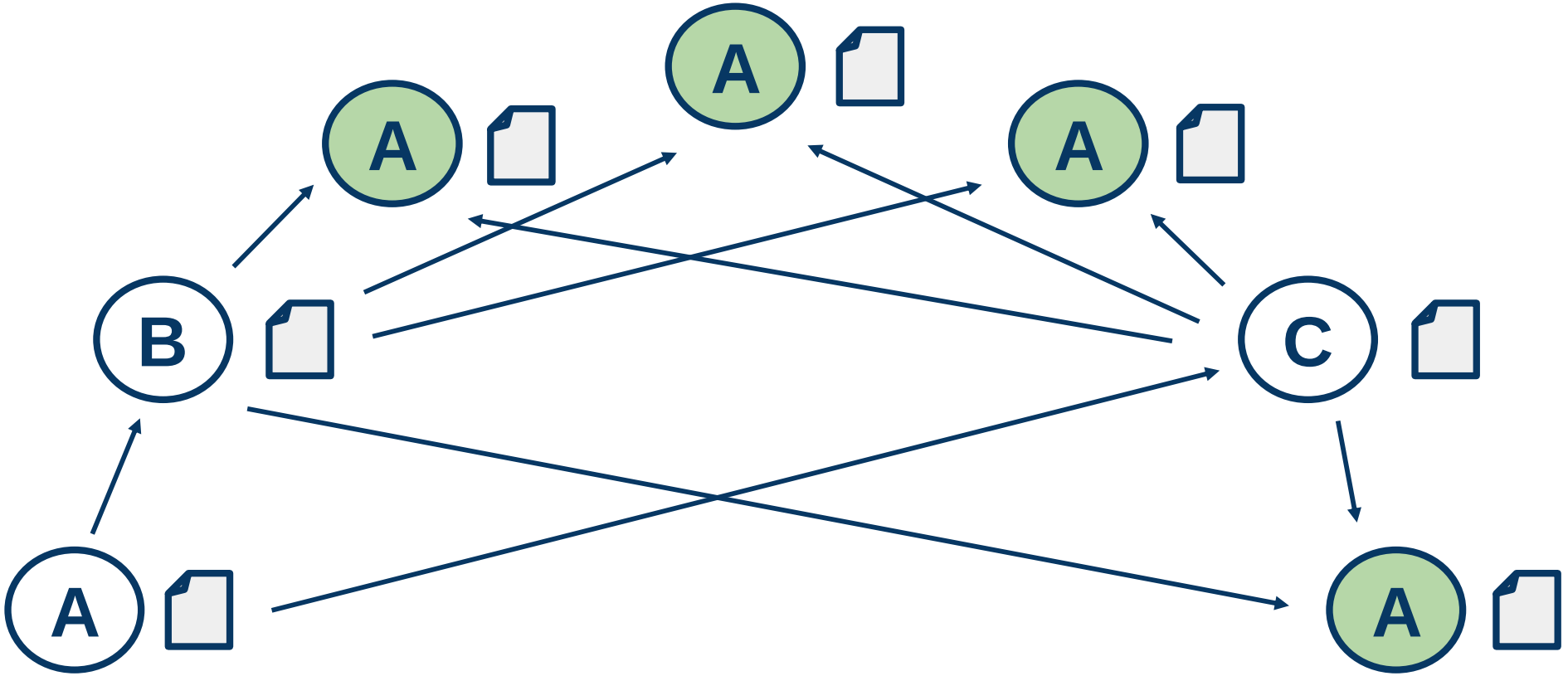


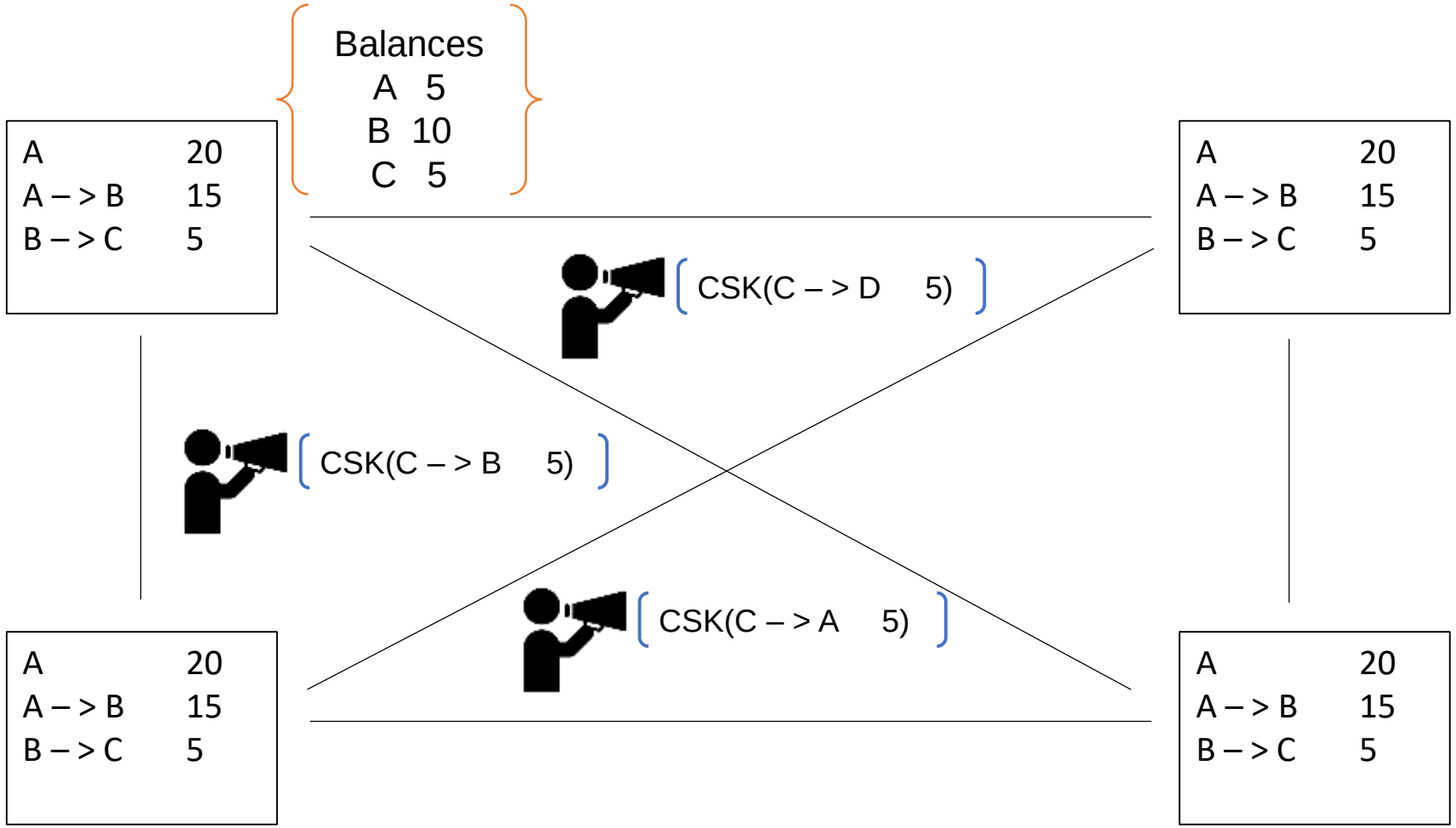
Alice sets up multiple identities



Alice double spends with her multiple identities

Sybil Attack: Done by creating many fake identities





Problems?

Double Spending
Sybil Attack

1. Who can write data into the chain?
2. In which order we need to process transactions?

Solution

Consensus in the network
Proof of Work

Proof of Work

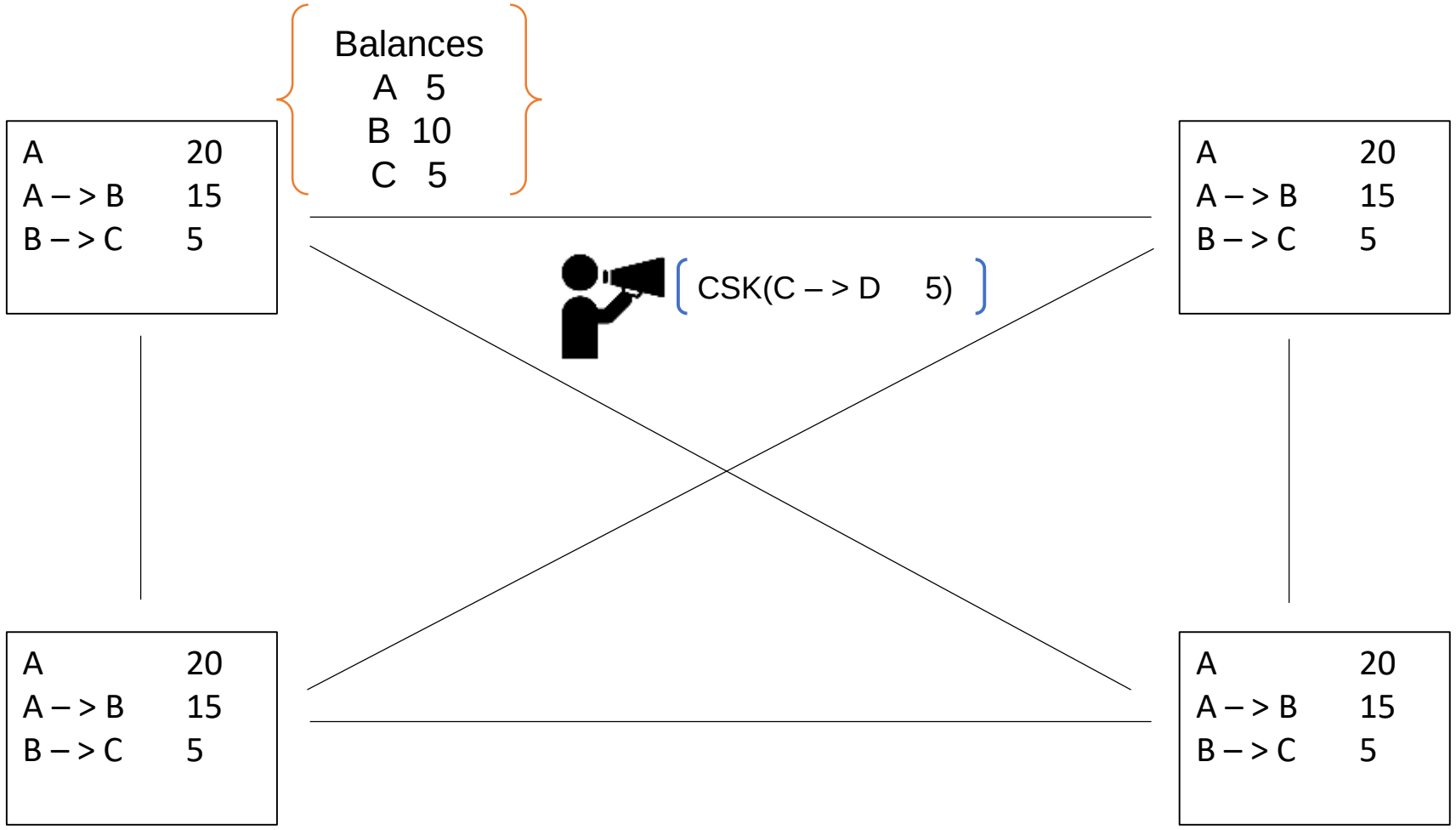
A **proof of work** is a piece of computation which is difficult (costly, time-consuming) to solve but easy for others to verify and which satisfies certain requirements

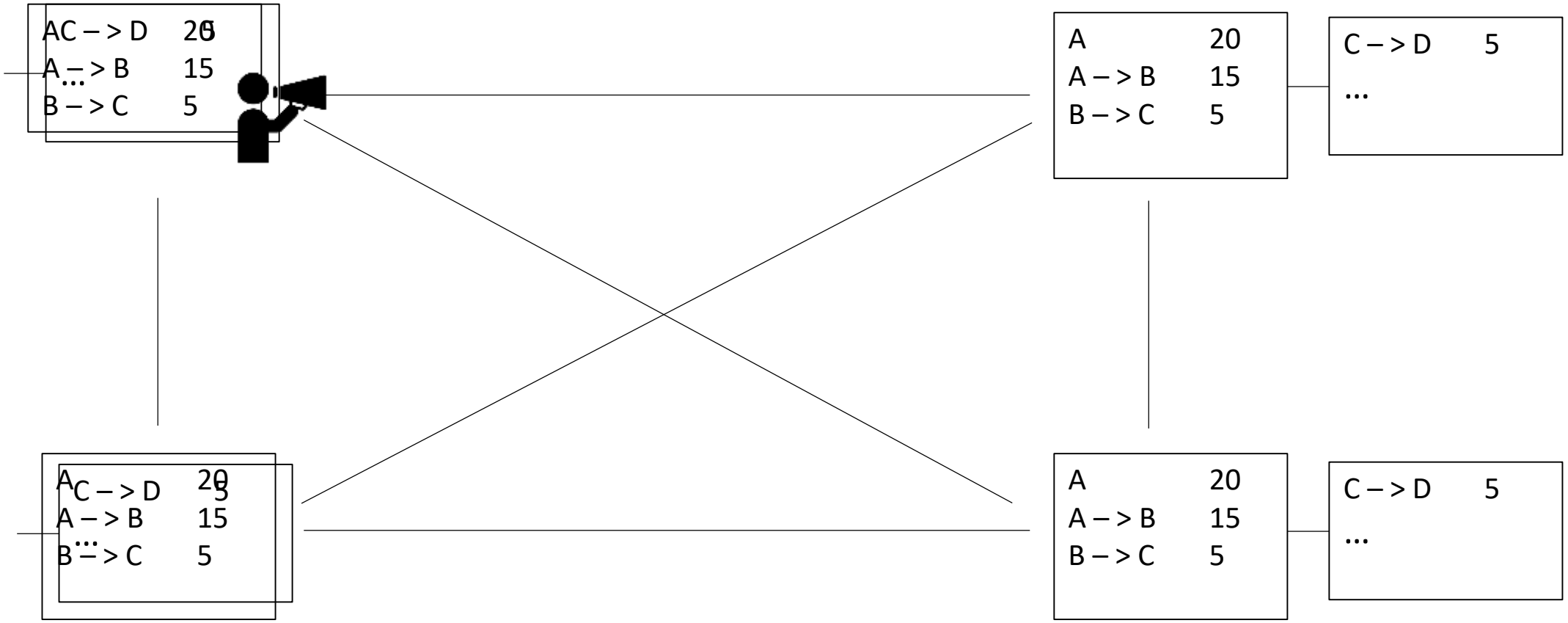
$$a_0 = f(x)$$
$$a_0 < \text{TARGET}$$

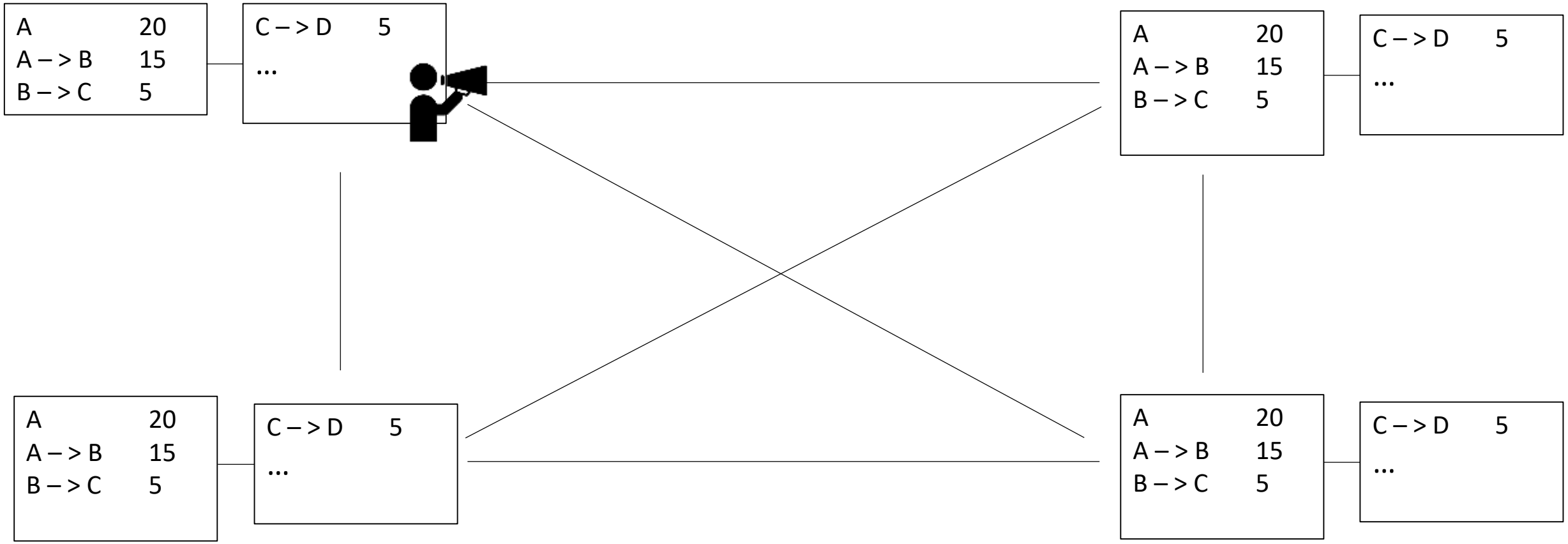
```
TARGET = (65535 << 208) / DIFFICULTY;
coinbase_nonce = 0;
while (1) {
      header = makeBlockHeader(transactions, coinbase_nonce);
      for (header_nonce = 0; header_nonce < (1 << 32); header_nonce++){
            if (SHA256(SHA256(makeBlock(header, header_nonce))) <
      TARGET)
                  break; //block found!
      }
      coinbase_nonce++;
}
```

Figure 5.6 : CPU mining pseudocode.

Image source: Mastering Bitcoin


Balances
A 5
B 10
C 5
A 20
A – > B 15
B – > C 5
CSK(C – > D 5)
A 20
A – > B 15
B – > C 5
A 20
A – > B 15
B – > C 5
A 20
A – > B 15
B – > C 5


AC – > D 20 5
A – > B 15
B – > C 5
...
A 20
A – > B 15
B – > C 5
C – > D 5
...
A 20
C – > D 5
A – > B 15
B – > C 5
...
A 20
A – > B 15
B – > C 5
C – > D 5
...


A 20
A – > B 15
B – > C 5
C – > D 5
…
A 20
A – > B 15
B – > C 5
C – > D 5
…
A 20
A – > B 15
B – > C 5
C – > D 5
…
A 20
A – > B 15
B – > C 5
C – > D 5
…

Slide by Viget

Summary

Version	Major feature	Value added
1	Signed messages announced to the network	Basis of entire system
2	Serial numbers	Uniquely identifiable transactions
3	The block chain	Shared record of transactions
4	Everyone verifies transactions	Increased security
5	Proof-of-work	Prevents double spending

One-way hash function (SHA-256)

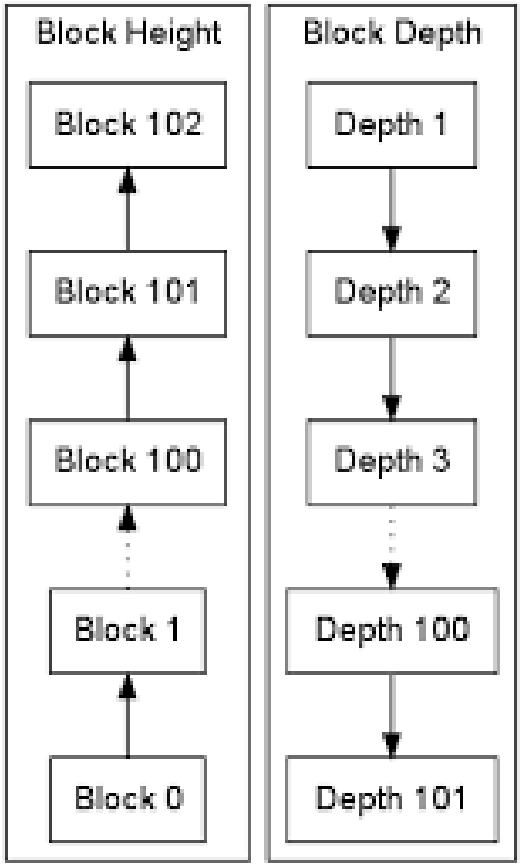
- The **SHA** (Secure Hash Algorithm) cryptographic hash function
- A cryptographic hash - a signature for a text or a data file.
- **SHA-256** generates an almost-unique, fixed size **256**-bit (32-byte) hash.
- Hash is a one way function – it cannot be decrypted back.
- https://www.movable-type.co.uk/scripts/sha256.html

Basic Concepts - Blocks + Blockchain

Blocks

- Contains an ordered bunch of transactions
  - Timestamps the transactions, are immutable
- Each block References a previous block
- Each block has height and depth (confirmations)
  - Currently 599k blocks
  - https://bitinfocharts.com/bitcoin/

Blockchain=The entire series of blocks 'chained' togethe



Block Height Compared To Block Depth

Source: Bitcoin Developer Guide

BLOCKCHAIN info | Home | Charts | Stats | Markets | API | Wallet | Search | English

Transaction View information about a bitcoin transaction

447cb6623db32b5f28c94ac10551802075f053208fe995204a145197e2904bb9

3Nxwenay9Z8Lc9JBiywExpnEFiLp6Afp8v (44,000 BTC - Output)	3LrLWTSdd69oZVVQ6dtWaAAaBLn7N3rRjz - (Spent)	333.33328889 BTC
	3QkXtcSWJA9w77eCujnMBKDWFe7F7zwxTg - (Spent)	333.33328889 BTC
	3Qd7hXZoZ1iyXZznrbdUwUQBxHMmujdqhJ - (Spent)	333.33328889 BTC
	3ECJwvx9VgfotcUuEJMVNvmWnTGVMk179L - (Spent)	333.33328889 BTC
	3BuQmbmdce3e31GEovq5SgowLdfMgJzLDE - (Spent)	333.33328889 BTC
	3NwKLjJjzXSnBFQWokXRgBG3JeuF3bsnfE - (Spent)	333.33328889 BTC
	3GEaT8ZRXELcjMSFvGro6eZcC5S1LSLZuN - (Spent)	333.33328889 BTC
	35DVAzDtZDKAU94kFT9sxoscnuLCTxgwYc - (Spent)	333.33328889 BTC
	3Nxwenay9Z8Lc9JBiywExpnEFiLp6Afp8v - (Unspent)	38,000 BTC
	35mwqShnStDro6uEB4bmsgbyBo8en6Byfm - (Spent)	333.33328889 BTC
	39pvSqfNcUosc8RGVWxyzKM3ny96a3uSkW - (Spent)	333.33328889 BTC
	39QNJSgQg5JnBXAtbF8ezkDn72VqWdPZPJ - (Spent)	333.33328889 BTC
	3L9qAGBQLbXkFAB2GpijnJXPScSVjuiJio - (Spent)	333.33328889 BTC
	37WSkANPVUQ8uuktf8hv671CejRtBtQ4tJ - (Spent)	333.33328887 BTC
	3EEwPZZ6pYRJSotCz9RBoVYPRnoWyGWEka - (Spent)	333.33328889 BTC
	3C4ABC7iPcAAKBh6SJXfvUSDBew3abCtw3 - (Spent)	333.33328889 BTC
	3HpQozfTzoXAsHf87m2mwJXUQ14LVtLgK4 - (Spent)	333.33328889 BTC
	337RfngTLRTpU7RT9sKWQWDdmfcdmWnugi - (Spent)	333.33328889 BTC
	3P2eoKr3vAeZhJcTzon3VFkv5r7DqSXW9G - (Spent)	333.33328889 BTC
		43,999.9992 BTC

Summary	
Size	1055 (bytes)
Received Time	2016-08-30 11:45:03
Included In Blocks	427512 (2016-08-30 11:51:09 + 6 minutes)
Confirmations	854 Confirmations
Relayed by IP	5.39.93.85 (whois)
Visualize	View Tree Chart

Inputs and Outputs	
Total Input	44,000 BTC
Total Output	43,999.9992 BTC
Fees	0.0008 BTC
Estimated BTC Transacted	333.33328887 BTC
Scripts	Hide scripts & coinbase

https://www.blockchain.com/explorer

Byzantine Generals Problem

Group of generals surrounding a city must vote and agree on a plan of action

Constraints:

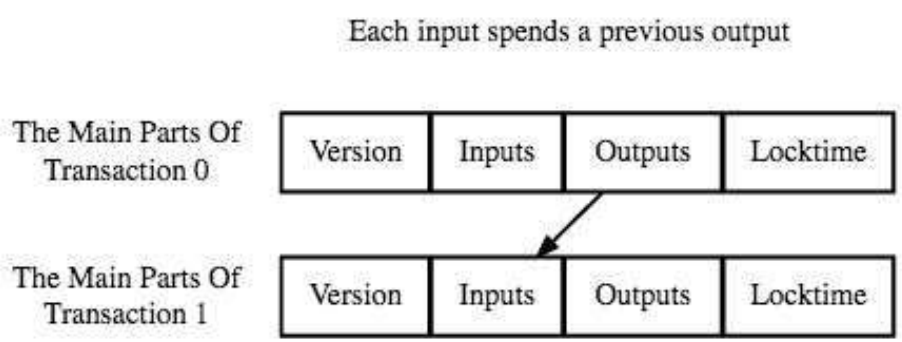
- Generals are physically separated; must use messengers
  - Messengers may fail
- Generals may be loyal or intentionally traitorous
- Assume **majority** of generals are loyal
- "Byzantine Fault Tolerance" achieved if **loyal** generals have unanimous agreement on strategy

In Bitcoin, this is an agreement on the history of transactions

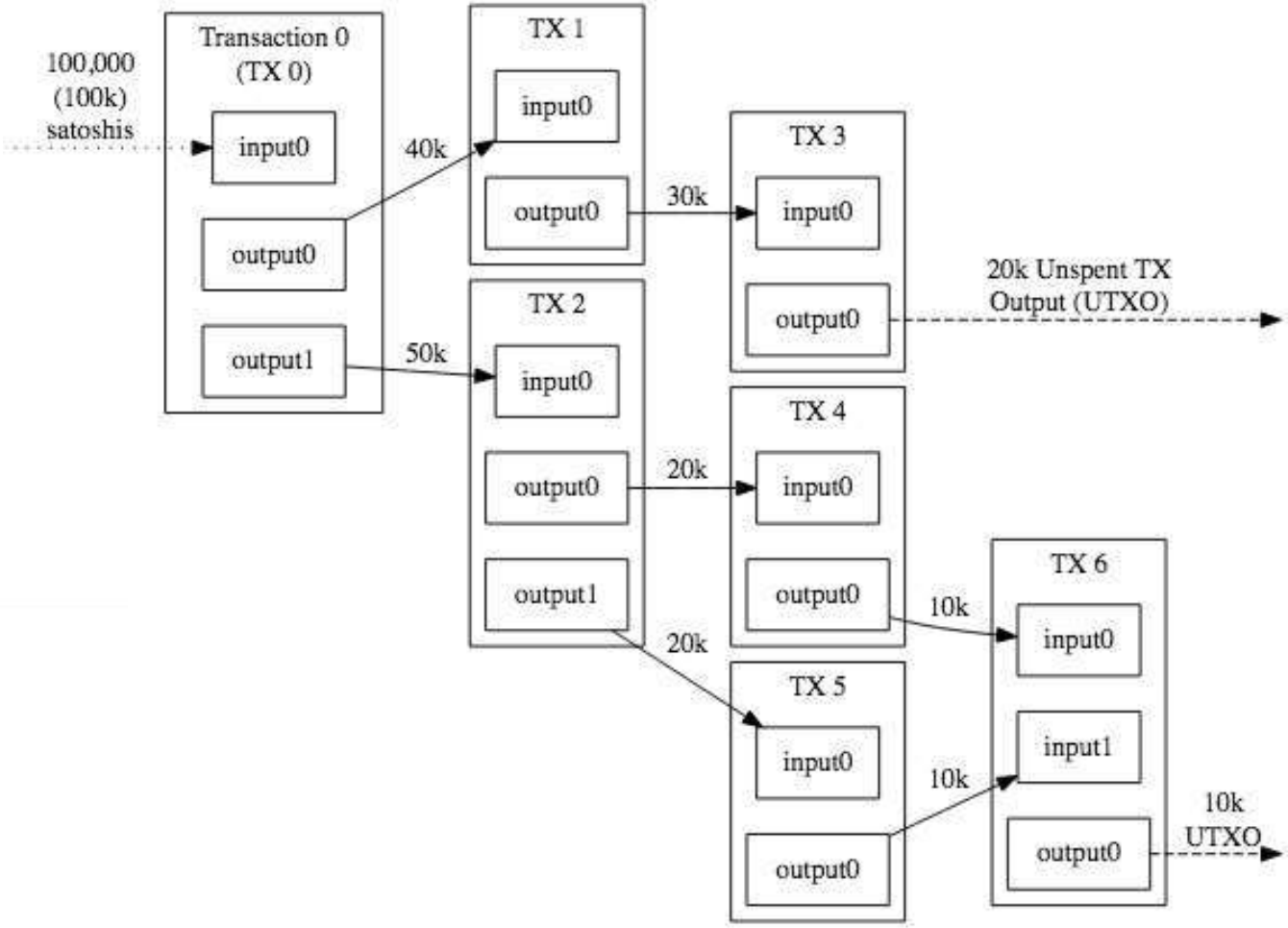
Basic Concepts - Transactions

Source: Bitcoin Developer Guide

- Maps inputs addresses to output addresses
  - Outputs can only be spent once
- Typical tx: one input, two outputs
- Fees are implicit



Each output waits as an Unspent TX Output (UTXO) until a later input spends it



Triple-Entry Bookkeeping (Transaction-To-Transaction Payments) As Used By Bitcoin

Basic Concepts - UTXO analogy

UTXOs stands for "Unspent Transaction Outputs"

- Global set of unspent bitcoins
- "I'm spending THIS bitcoin," not "I'm spending A bitcoin."

Analogous to Rai Stones of the Yap Islands

- Rai Stones never moved
- Instead: Agreed on change of ownership

Source: Wikipedia

Recap - The Innovation of Satoshi Nakamoto

Bitcoin was created by Satoshi Nakamoto in 2009

- First ever decentralized, trustless system for transactions
  - A low cost financial system that only requires an internet connection
- Nakamoto solved the Double Spending problem
  - Prevent someone from spending the same asset twice
  - Solution? The blockchain + PoW
  - Who helped secure the network were rewarded
  - Who tried to compromise the Bitcoin network had to make a massive financial investment to do so, with relatively little benefit

Dorian Satoshi Nakamoto
(not actually Satoshi Nakamoto-probably)

Sketch of Bitcoin Mining - Proof of Work

- Solution to the Byzantine Generals Problem: Proof-of-work (PoW)
  - “Miners” continuously compete to solve a very computationally difficult problem
  - Proof of work is an example of a "Byzantine consensus algorithm"
- Proof of work criteria:
  - Easy to verify
  - Hard to compute
- SHA-256 Hash function satisfies these
  - One-way hash function; can hash any arbitrary data
  - Pretty much random (very useful property)
- Example
  - SHA256("Donald Trump") == "e4f2e1f0e2ae4d3ce7018cf3b4f3577c99714bdc9f5a4ac28e3e7cb2c505db6c"
  - SHA256("Donald trump") == "6ad2fa6a5caaee9143578931456322c4433a92ae2af8f0d5c9b4f9bb080f49d6"

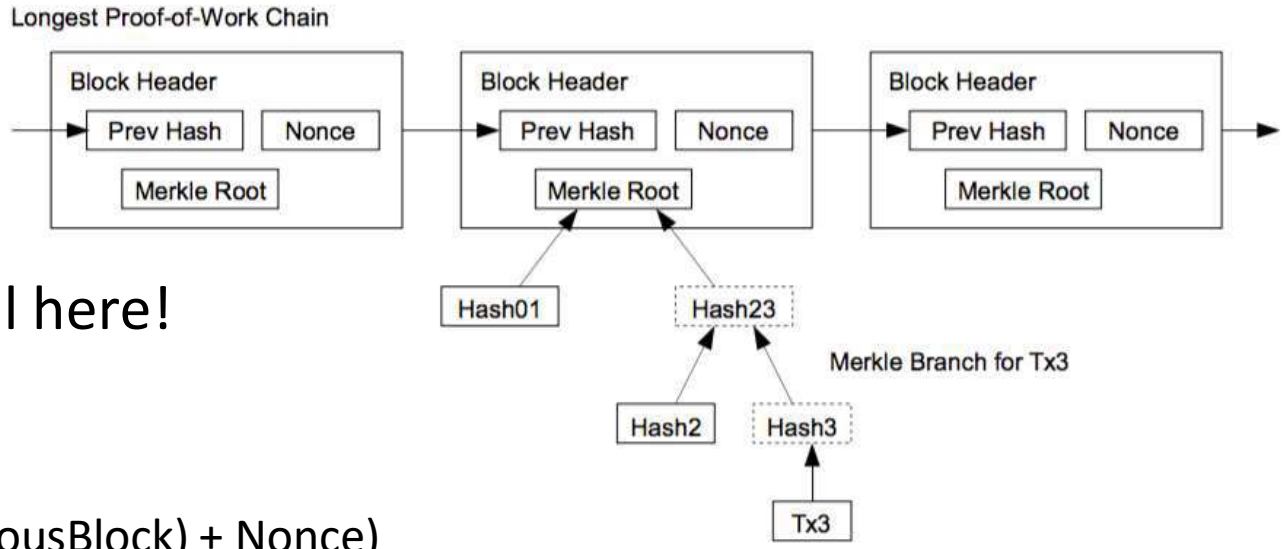
Sketch of Bitcoin Mining - Finding blocks

- Finding the PoW => 'found' a block; can add block to blockchain
  - Miner who found block adds "**coinbase transaction**"
    - contains mining reward (currently 12.5 BTC)
  - Miner broadcasts block
  - Other nodes verify, then add to their own copy of the blockchain
- Timeline + stats
  - This happens roughly every 10 minutes
    - Difficulty of the problem adjusted every 2 weeks
  - Block reward halving every 4 years
  - Bitcoin is in limited supply - 21 million bitcoins by 2140
    - Deflationary
  - 17.3 million bitcoins currently in circulation today
  - ~$142 billion market cap (10.2019)
  - Price is currently ~$6590 per bitcoin

Sketch of Bitcoin Mining - The Mining Problem

Components hashed together:

- Merkle Root
  - 'summary' of the transactions in the block
- Hash of previous block
- Nonce
  - Randomness of SHA-256 is useful here!



Formally:

- Output = SHA-256(Merkle Root + SHA-256(PreviousBlock) + Nonce)
- Solution (Proof-of-work): an output that contains a requisite number of leading 0 bits
  - The number of 0 bits is the **difficulty**
  - Difficulty adjusts every every 2016 blocks* to regulate block creation
    - *technically every 2015 blocks

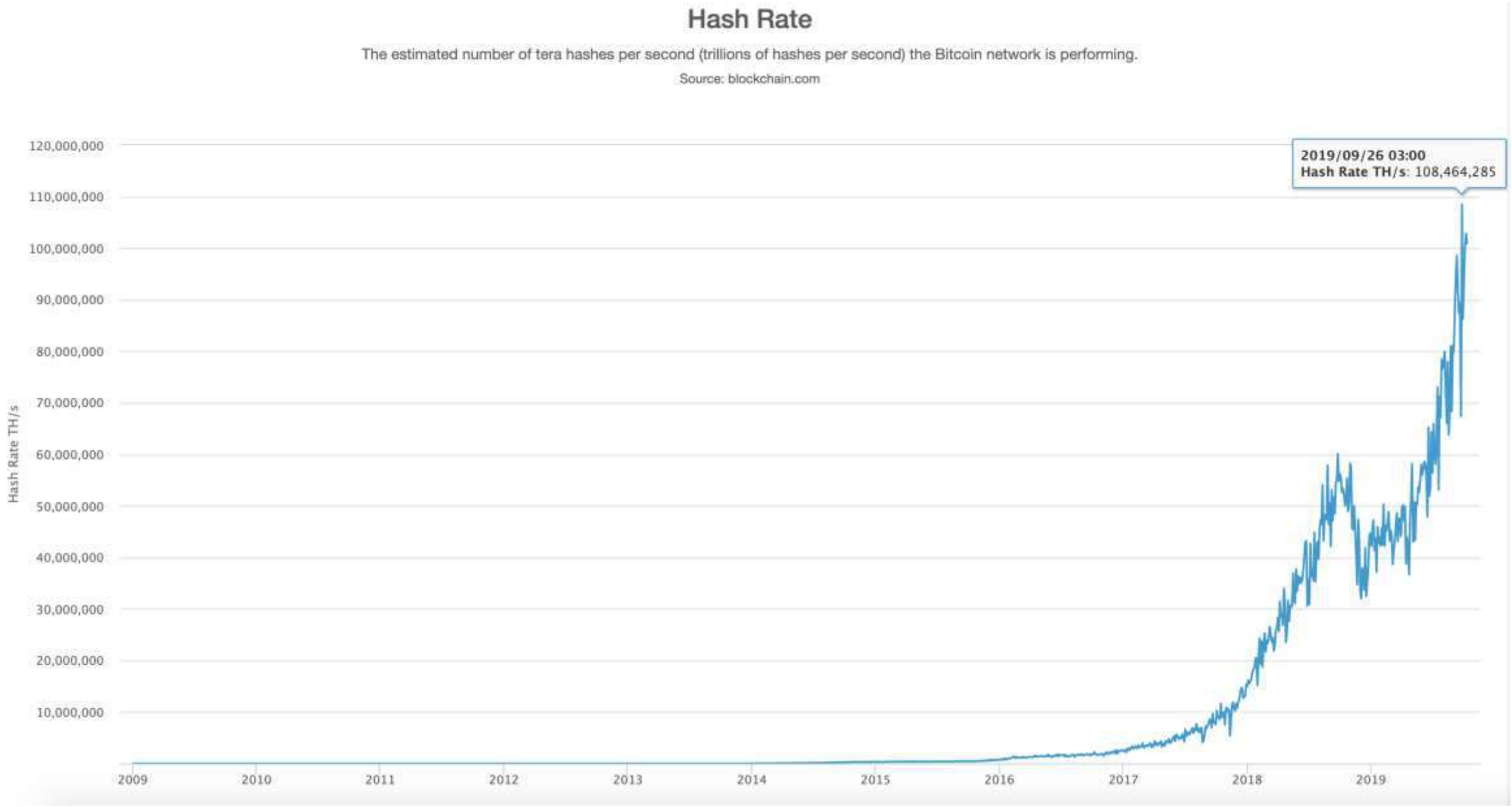
Sketch of Bitcoin Mining - 51% Attacks

Major assumption of Bitcoin:

No more than 51% percent of the network is dishonest

An honest majority will always form the longest proof-of-work chain

51% Attack: Attempt to overwhelm the mining power of the network


Hash Rate
The estimated number of tera hashes per second (trillions of hashes per second) the Bitcoin network is performing.
Source: blockchain.com
2019/09/26 03:00
Hash Rate TH/s: 108,464,285
Hash Rate TH/s
120,000,000
110,000,000
100,000,000
90,000,000
80,000,000
70,000,000
60,000,000
50,000,000
40,000,000
30,000,000
20,000,000
10,000,000
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019

Sketch of Bitcoin Mining - Summary

Functions as:

- A minting mechanism that ensures coins are distributed in a fair way
- An incentive for people to help secure the network
- Key component that enables you reach consensus in a decentralized currency

Bringing it all together - Back to a transaction

- I want to send money to Sunny
  - Sign transaction
  - Broadcast to network
- Miners receives transaction, adds to "**zero-conf pool**"
  - Verify transaction: i.e. signature matches, enough money,
- Miner finds PoW, broadcasts block
  - Block propagates; others verify
- Miners work on the next problem

Demo

- https://anders.com/blockchain/

Next class – Bitcoin Mechanics

Suggested Readings:

- Blockchain Demo: https://www.youtube.com/watch?v=_160oMzblY8
- Princeton Textbook 5.1-5.4 (pg. 131 - 157)
- Bitcoin Wallets Explained

http://cryptorials.io/bitcoin-wallets-explained-how-to-choose-the-best-wallet-for-you

References

Slides mainly adopted from

- Blockchain @ Berkeley : https://blockchain.berkeley.edu/
- Blockchain @ Princeton : http://bitcoinbook.cs.princeton.edu/

Bitcoin Mechanics

Blockchain on the news

Blok Zinciri Mühendislerine Talep Yüzde 400 Arttı!

Maaşları 15 Bin Dolara Çıktı

https://www.haberler.com/blok-zinciri-muhendislerine-talep-yuzde-400-artti-11357793-h

The Big Blockchain Lie

Roubini's predictions have earned him the nicknames "Dr. Doom"

https://www.project-syndicate.org/commentary/blockchain-big-lie-by-nouriel-roubini-2018-10

CRYPTOGRAPHIC HASH FUNCTIONS

INTEGRITY OF INFORMATION

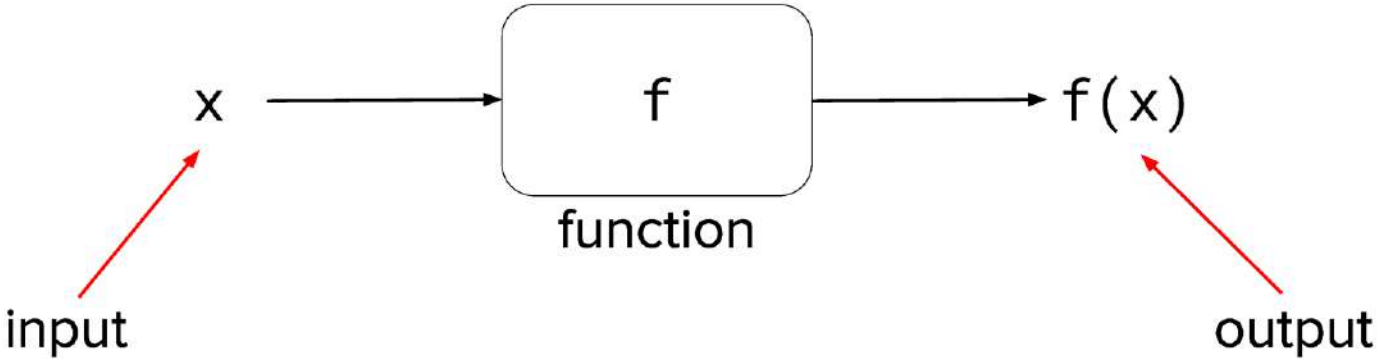
How do we ensure trust in communication in a trustless environment?

⇒ With **cryptographic hash functions**

Image source: https://spiritegg.com/wp-content/uploads/2016/03/63180952_fingerprint_types624.jpg

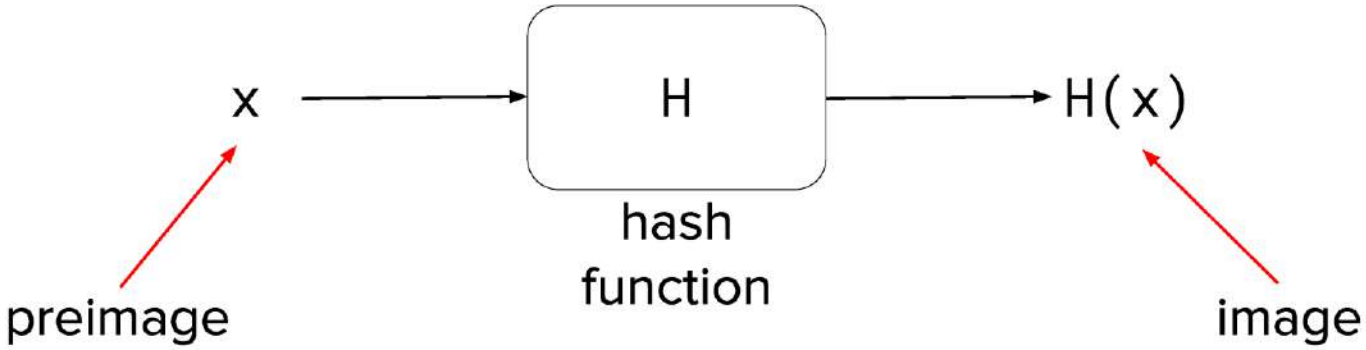
CRYPTOGRAPHIC HASH FUNCTIONS

WHAT IS A HASH FUNCTION?



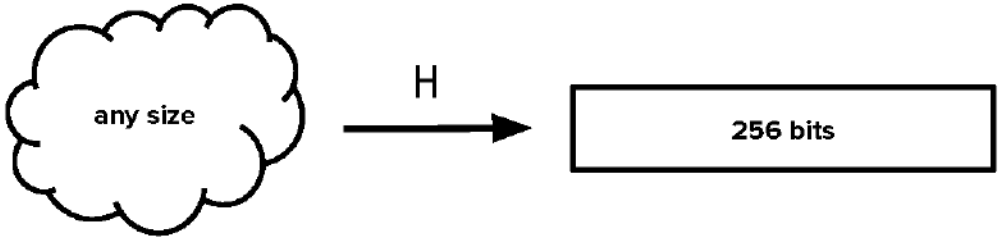
CRYPTOGRAPHIC HASH FUNCTIONS

WHAT IS A HASH FUNCTION?



CRYPTOGRAPHIC HASH FUNCTIONS

WHAT IS A HASH FUNCTION?



CRYPTOGRAPHIC HASH FUNCTIONS

#CRYPTOGRAPHY

Cryptographic hash function:

A hash function with three special properties:

- Preimage resistance
- Second preimage resistance
- Collision resistance

The equivalent of **mathematical fingerprints/identifiers**

Image source:
http://chimera.labs.oreilly.com/books/1234000001802/ch08.html#_proof_of_work_algorithm

```
I am Satoshi Nakamoto0 => a80a81401765c8eddee25df36728d732...
I am Satoshi Nakamoto1 => f7bc9a6304a4647bb41241a677b5345f...
I am Satoshi Nakamoto2 => ea758a8134b115298a1583ffb80ae629...
I am Satoshi Nakamoto3 => bfa9779618ff072c903d773de30c99bd...
I am Satoshi Nakamoto4 => bce8564de9a83c18c31944a66bde992f...
I am Satoshi Nakamoto5 => eb362c3cf3479be0a97a20163589038e...
I am Satoshi Nakamoto6 => 4a2fd48e3be420d0d28e202360cfbaba...
I am Satoshi Nakamoto7 => 790b5a1349a5f2b909bf74d0d166b17a...
I am Satoshi Nakamoto8 => 702c45e5b15aa54b625d68dd947f1597...
I am Satoshi Nakamoto9 => 7007cf7dd40f5e933cd89fff5b791ff0...
I am Satoshi Nakamoto10 => c2f38c81992f4614206a21537bd634a...
I am Satoshi Nakamoto11 => 7045da6ed8a914690f087690e1e8d66...
I am Satoshi Nakamoto12 => 60f01db30c1a0d4cbce2b4b22e88b9b...
I am Satoshi Nakamoto13 => 0ebc56d59a34f5082aaef3d66b37a66...
I am Satoshi Nakamoto14 => 27ead1ca85da66981fd9da01a8c6816...
I am Satoshi Nakamoto15 => 394809fb809c5f83ce97ab554a2812c...
I am Satoshi Nakamoto16 => 8fa4992219df33f50834465d3047429...
I am Satoshi Nakamoto17 => dca9b8b4f8d8e1521fa4eaa46f4f0cd...
I am Satoshi Nakamoto18 => 9989a401b2a3a318b01e9ca9a22b0f3...
I am Satoshi Nakamoto19 => cda56022ecb5b67b2bc93a2d764e75f...
```

CRYPTOGRAPHIC HASH FUNCTIONS

PREIMAGE RESISTANCE

Preimage resistance:

Given $H(x)$, it is computationally difficult to determine x

Fingerprint analogy:

Whose fingerprint is this?

CRYPTOGRAPHIC HASH FUNCTIONS

SECOND PREIMAGE RESISTANCE

Second preimage resistance:

Given x, it is computationally difficult to find some value x' such that H(x) == H(x')

Fingerprint analogy:

Can you find someone with the same fingerprint as you?

CRYPTOGRAPHIC HASH FUNCTIONS

COLLISION RESISTANCE

Collision resistance:

It is computationally difficult to find x and y such that

`H(x) == H(y)`

Fingerprint analogy:

Can you find two random people with the same fingerprint?

CRYPTOGRAPHIC HASH FUNCTIONS

AVALANCHE EFFECT

Avalanche effect: a small change in the input produces a pseudorandom change in the output

- Often a significant difference from the first output
- Prevents “hot or cold” game with inputs to produce or predict outputs

```
I am Satoshi Nakamoto0 => a80a81401765c8eddee25df36728d732...
I am Satoshi Nakamoto1 => f7bc9a6304a4647bb41241a677b5345f...
I am Satoshi Nakamoto2 => ea758a8134b115298a1583ffb80ae629...
I am Satoshi Nakamoto3 => bfa9779618ff072c903d773de30c99bd...
I am Satoshi Nakamoto4 => bce8564de9a83c18c31944a66bde992f...
I am Satoshi Nakamoto5 => eb362c3cf3479be0a97a20163589038e...
I am Satoshi Nakamoto6 => 4a2fd48e3be420d0d28e202360cfbaba...
I am Satoshi Nakamoto7 => 790b5a1349a5f2b909bf74d0d166b17a...
I am Satoshi Nakamoto8 => 702c45e5b15aa54b625d68dd947f1597...
I am Satoshi Nakamoto9 => 7007cf7dd40f5e933cd89fff5b791ff0...
I am Satoshi Nakamoto10 => c2f38c81992f4614206a21537bd634a...
I am Satoshi Nakamoto11 => 7045da6ed8a914690f087690e1e8d66...
I am Satoshi Nakamoto12 => 60f01db30c1a0d4cbce2b4b22e88b9b...
I am Satoshi Nakamoto13 => 0ebc56d59a34f5082aaef3d66b37a66...
I am Satoshi Nakamoto14 => 27ead1ca85da66981fd9da01a8c6816...
I am Satoshi Nakamoto15 => 394809fb809c5f83ce97ab554a2812c...
I am Satoshi Nakamoto16 => 8fa4992219df33f50834465d3047429...
I am Satoshi Nakamoto17 => dca9b8b4f8d8e1521fa4eaa46f4f0cd...
I am Satoshi Nakamoto18 => 9989a401b2a3a318b01e9ca9a22b0f3...
I am Satoshi Nakamoto19 => cda56022ecb5b67b2bc93a2d764e75f...
```

CRYPTOGRAPHIC HASH FUNCTIONS

SHA-256^2

SHA-256: A cryptographic hash function designed by the NSA

Bitcoin uses **SHA-256^2** (“`SHA-256 squared`”), meaning that `H(x)` actually means `SHA256(SHA256(x))`

- See readings for clarifications and reasoning

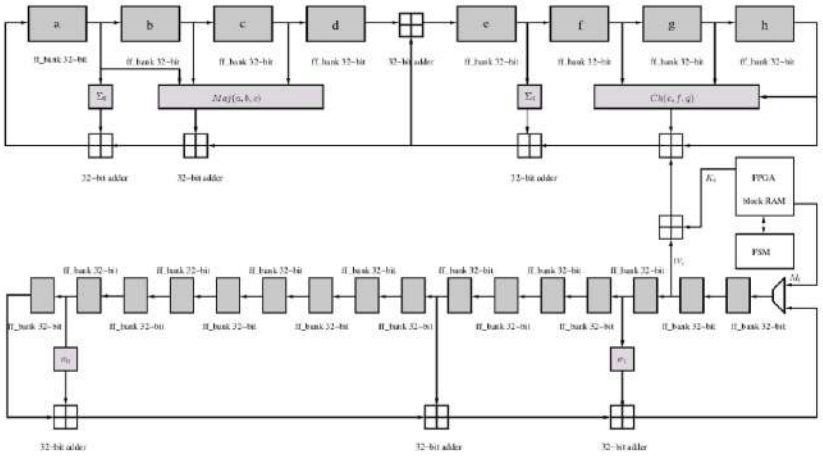


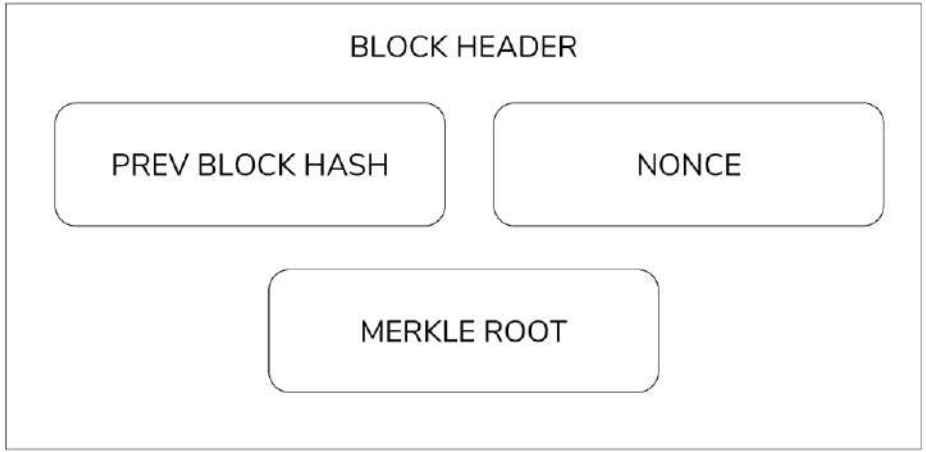
Image source:
https://opencores.org/usercontent.img,1375985843

2

A TAMPER-EVIDENT DATABASE

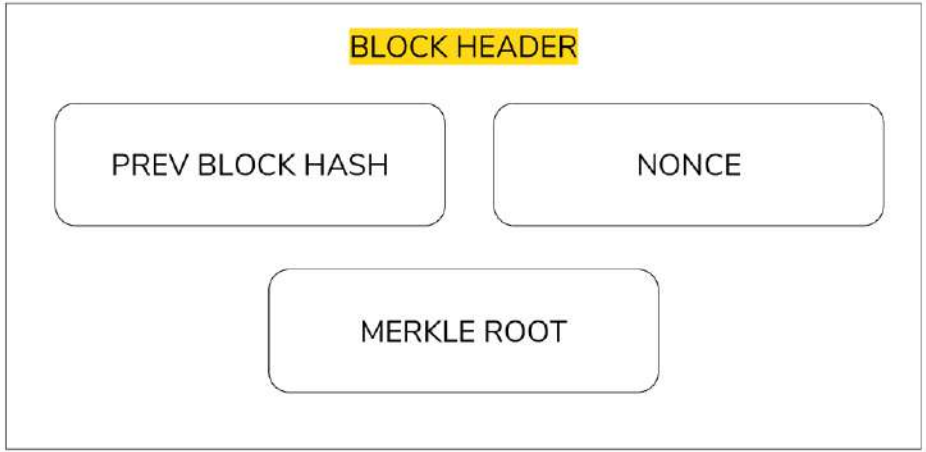
A TAMPER-EVIDENT DATABASE

DISSECTING THE BLOCKCHAIN



A TAMPER-EVIDENT DATABASE

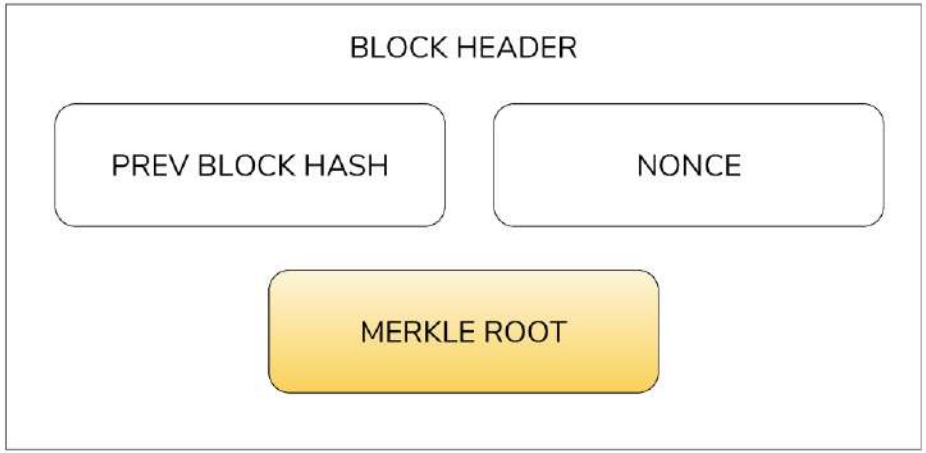
BLOCK HEADER

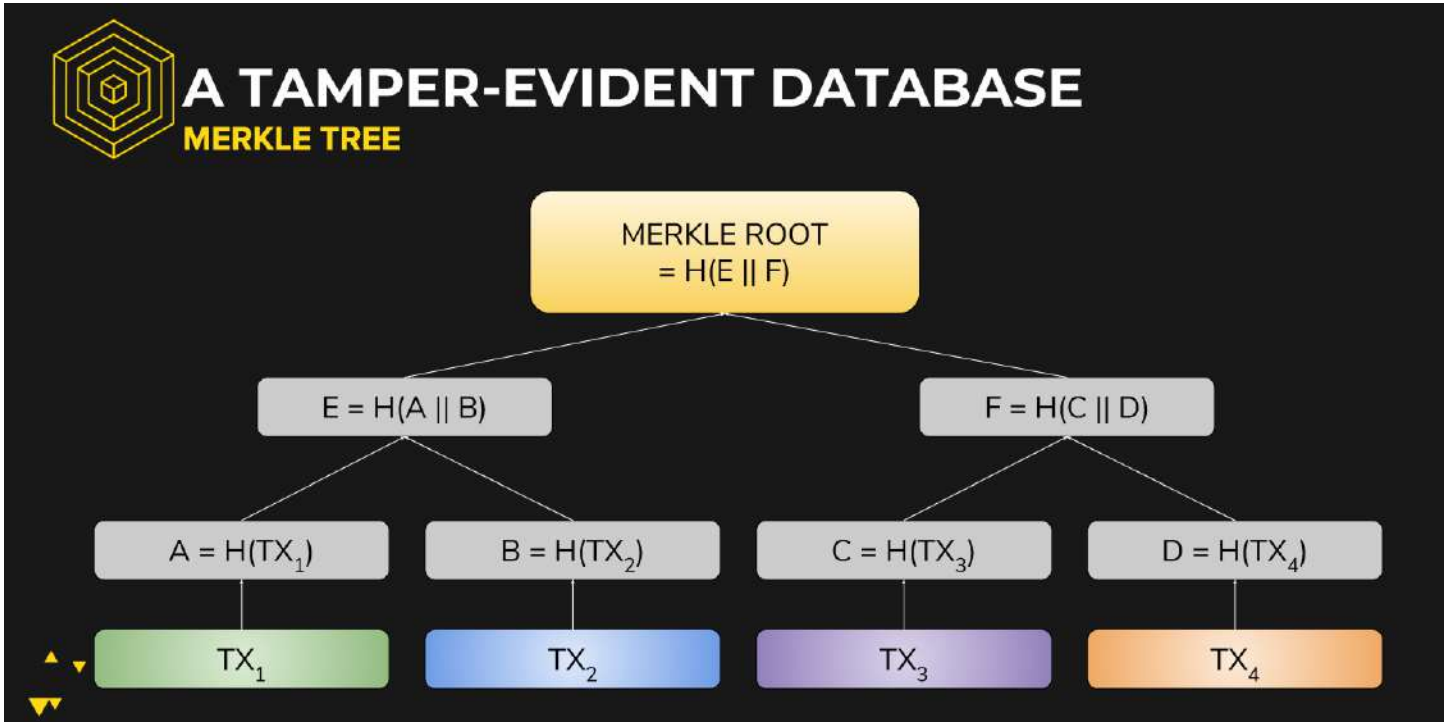


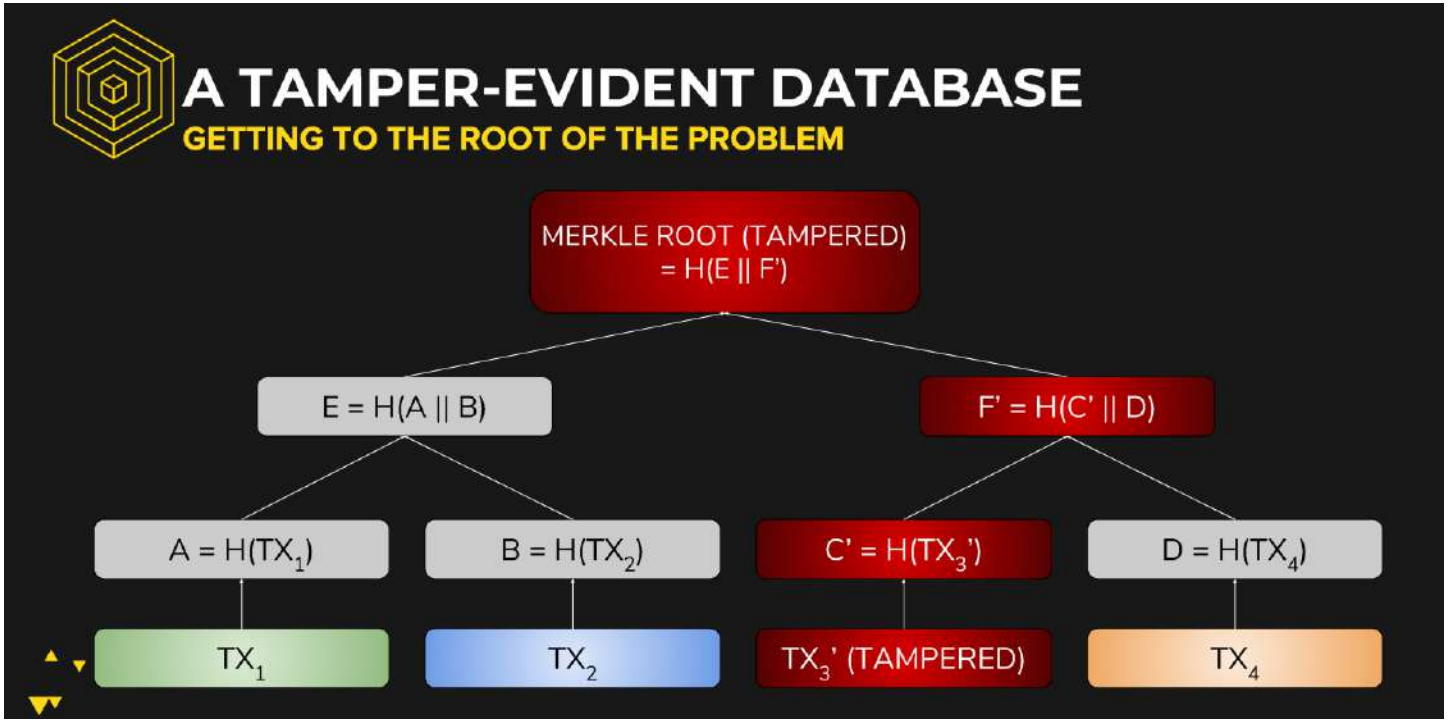

```
blockID = H(blockHeader) = H(prevBlockHash || merkleRoot || nonce)
```

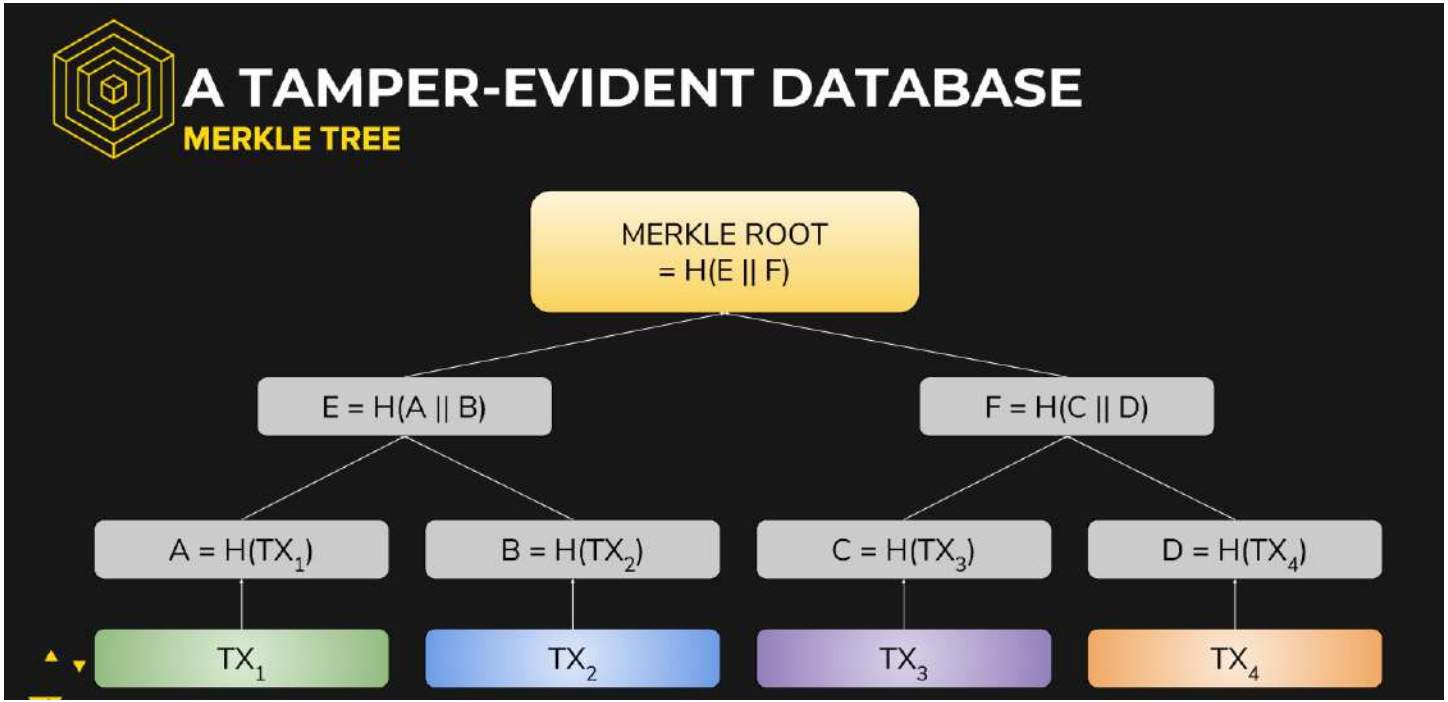
A TAMPER-EVIDENT DATABASE

MERKLE ROOT

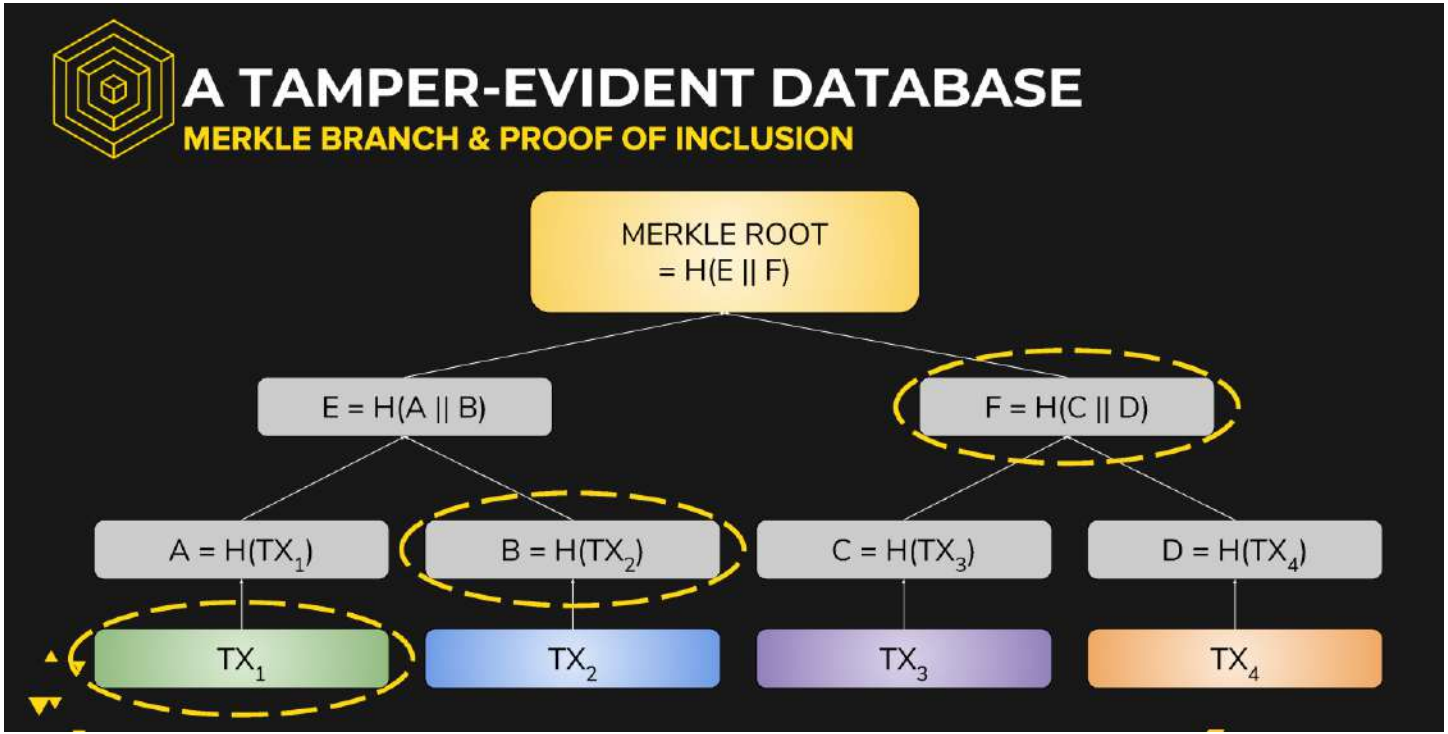



A TAMPER-EVIDENT DATABASE
MERKLE TREE
MERKLE ROOT
= H(E || F)
E = H(A || B)
F = H(C || D)
A = H(TX1)
B = H(TX2)
C = H(TX3)
D = H(TX4)
TX1
TX2
TX3
TX4


A TAMPER-EVIDENT DATABASE
GETTING TO THE ROOT OF THE PROBLEM
MERKLE ROOT (TAMPERED)
= H(E || F')
E = H(A || B)
F' = H(C' || D)
A = H(TX1)
B = H(TX2)
C' = H(TX3')
D = H(TX4)
TX1
TX2
TX3' (TAMPERED)
TX4

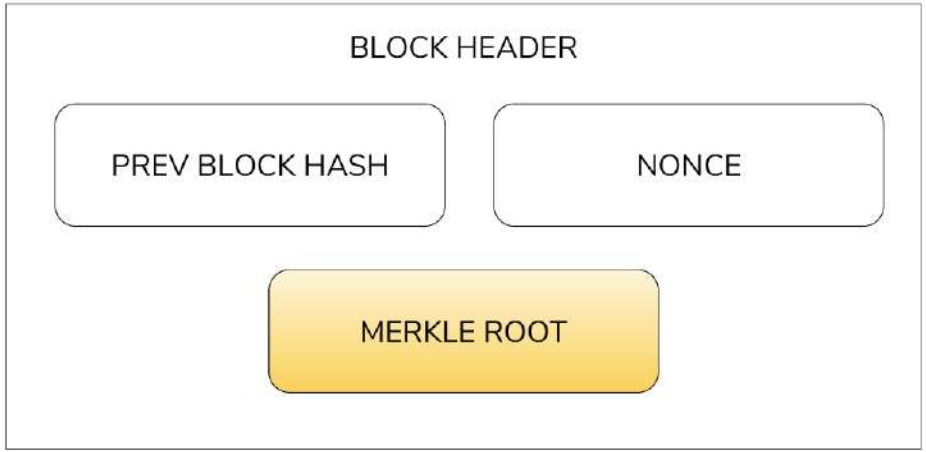


What do we need to validate a transaction?


A TAMPER-EVIDENT DATABASE
MERKLE BRANCH & PROOF OF INCLUSION
MERKLE ROOT
= H(E || F)
E = H(A || B)
F = H(C || D)
A = H(TX1)
B = H(TX2)
C = H(TX3)
D = H(TX4)
TX1
TX2
TX3
TX4

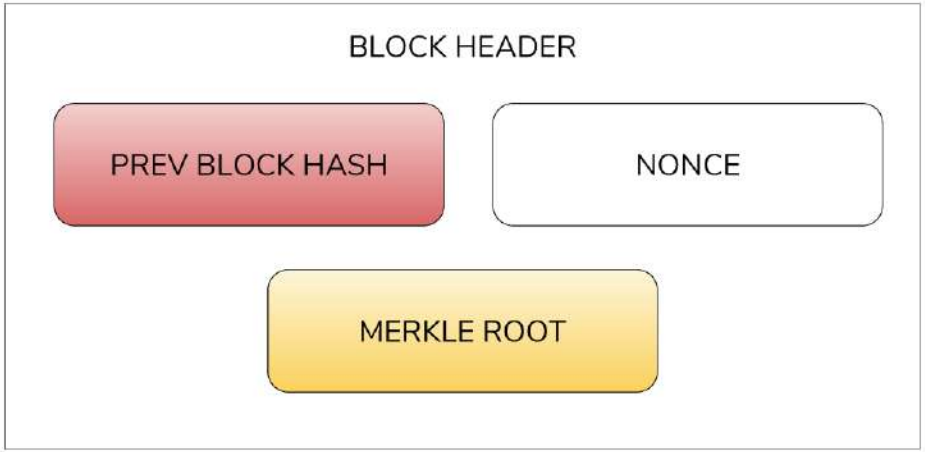
A TAMPER-EVIDENT DATABASE

MERKLE ROOT



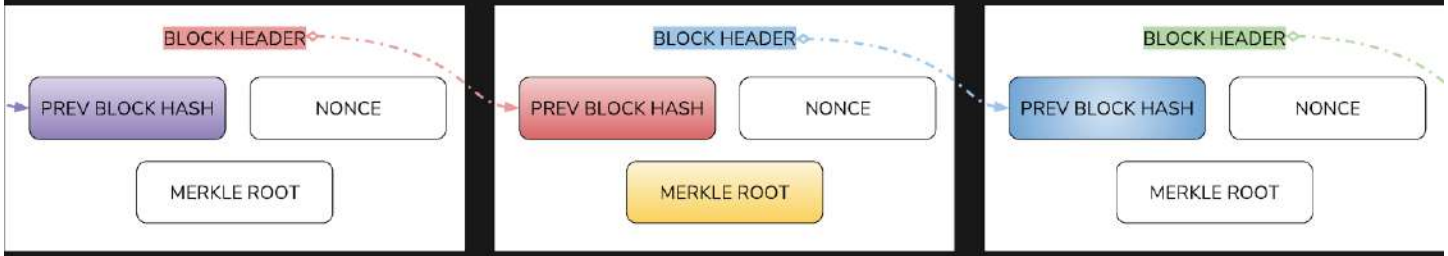
A TAMPER-EVIDENT DATABASE

PREV BLOCK HASH



A TAMPER-EVIDENT DATABASE

PROTECTING THE CHAIN

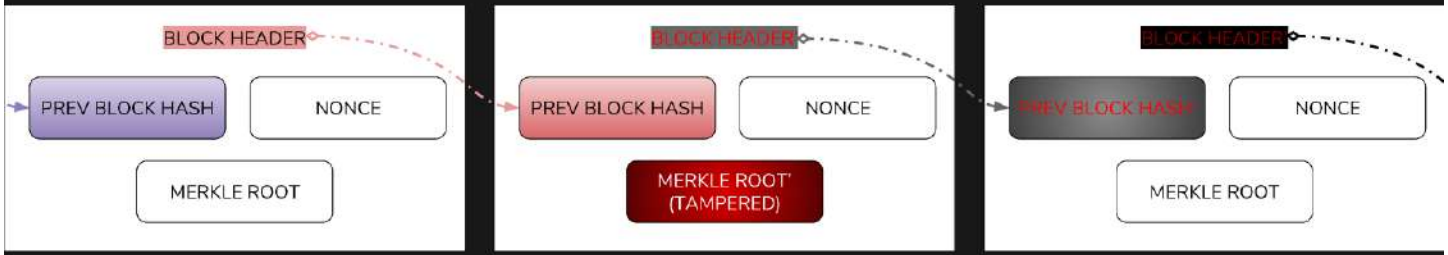


SHA256(SHA256(x))

```
prevBlockHash = H(prevBlockHash || merkleRoot || nonce)
```

A TAMPER-EVIDENT DATABASE

PROTECTING THE CHAIN

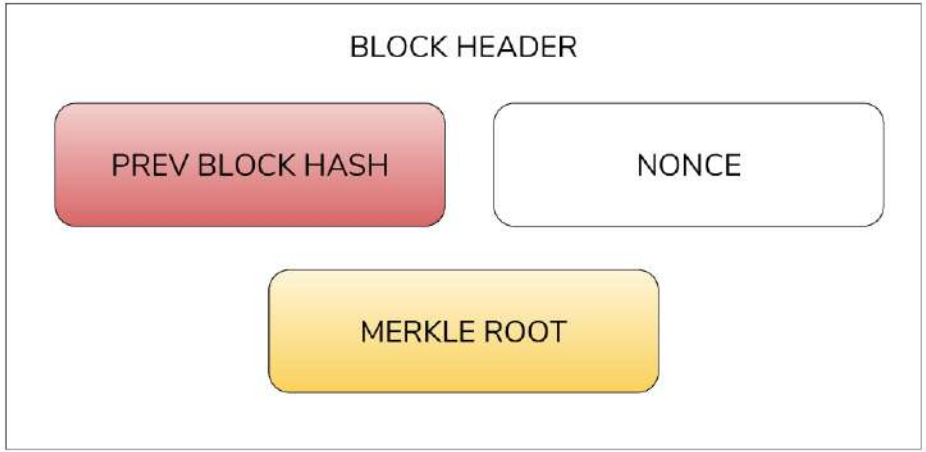


SHA256(SHA256(x))

```
prevBlockHash = H(prevBlockHash || merkleRoot || nonce)
```

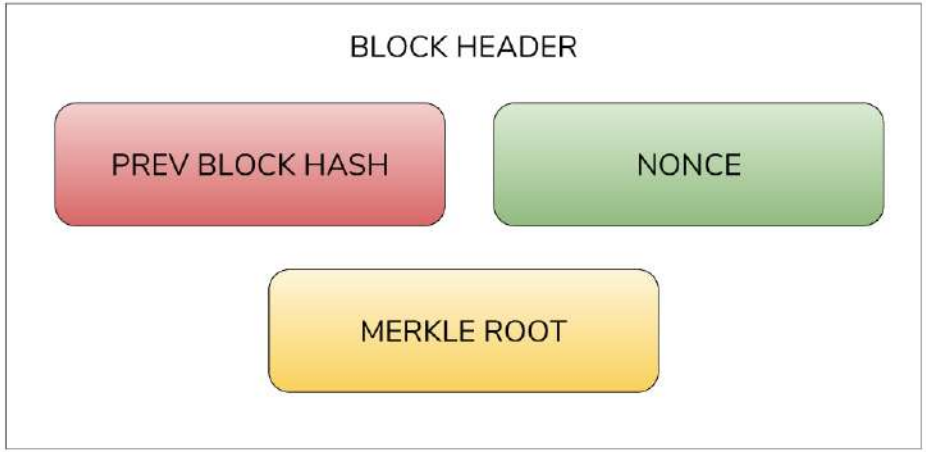
A TAMPER-EVIDENT DATABASE

PREV BLOCK HASH



A TAMPER-EVIDENT DATABASE

NONCE



A TAMPER-EVIDENT DATABASE

PARTIAL PREIMAGE HASH PUZZLE

Bitcoin's partial preimage hash puzzle: A problem with a requirement to find a nonce that satisfies the following inequality:

```
H(prevBlockHash || merkleRoot || nonce) < target
```

- Used to implement Proof-of-Work in Bitcoin (and every other PoW cryptocurrency)

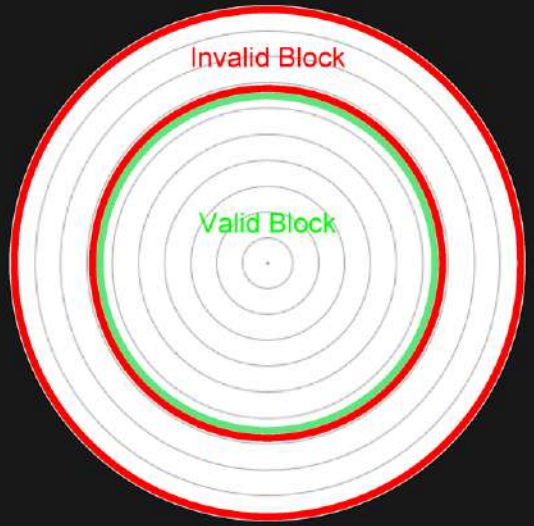
Hash puzzles need to be:

1. Computationally difficult.
2. Parameterizable.
3. Easily verifiable.

A TAMPER-EVIDENT DATABASE

MINING

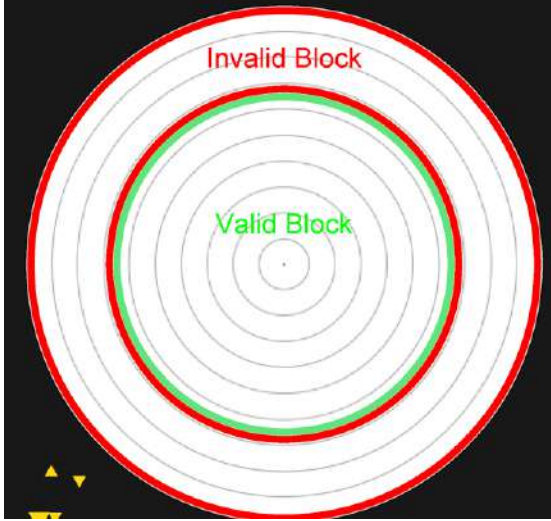
- **Mining** is like throwing darts at a target while blindfolded:
  - Equal likelihood of hitting any part of the target
  - Faster throwers ⇒ more hits / second
- Miners look for a hash below an algorithmically decided target




```
H(prevBlockHash || merkleRoot || nonce) < target
```

A TAMPER-EVIDENT DATABASE

BLOCK DIFFICULTY



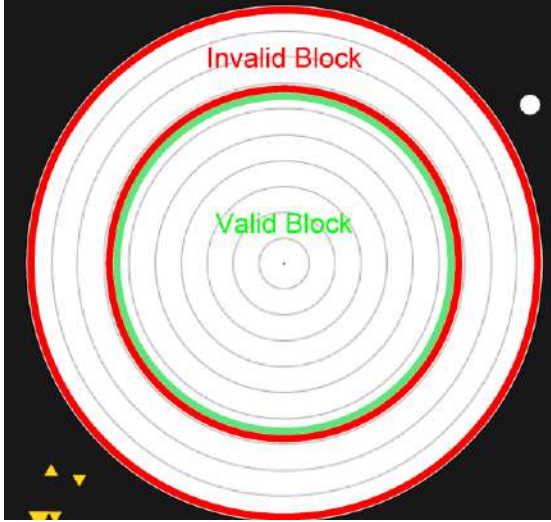
Difficulty: A representation of the expected number of computations required to find a block

- Implemented as requirement of leading number of 0s
- Adjusts with global hashrate
- `difficulty *= two_weeks / time_to_mine_prev_2016_blocks`
  - Technically every 2015 blocks

```
H(prevBlockHash || merkleRoot || nonce) < target
```

A TAMPER-EVIDENT DATABASE

BLOCK DIFFICULTY

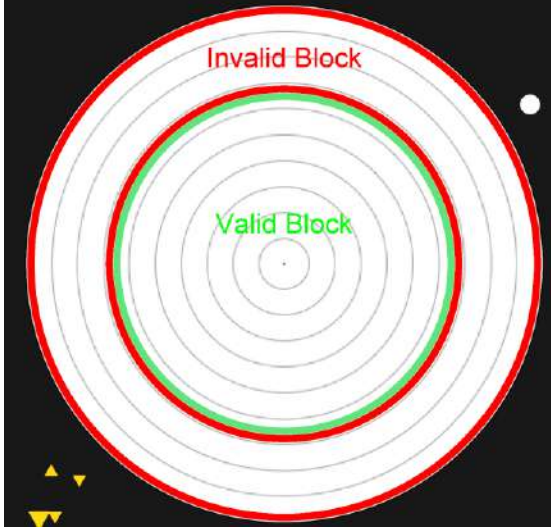


- **Sanity check** (difficulty = 10):
  - What is the new difficulty when `two_weeks` = `time_to_mine_prev_2016_blocks`?

```
difficulty *= two_weeks / time_to_mine_prev_2016_blocks
```

A TAMPER-EVIDENT DATABASE

BLOCK DIFFICULTY

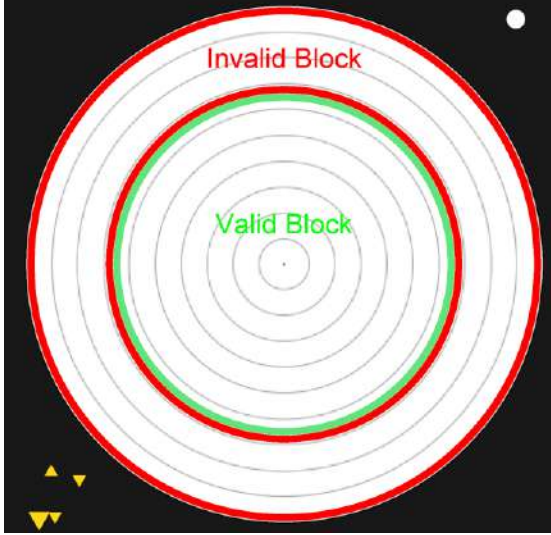


- **Sanity check** (difficulty = 10):
  - What is the new difficulty when `two_weeks` = `time_to_mine_prev_2016_blocks`?
    (Answer: 10. Difficulty stays the same!)

```
difficulty *= two_weeks / time_to_mine_prev_2016_blocks
```

A TAMPER-EVIDENT DATABASE

BLOCK DIFFICULTY

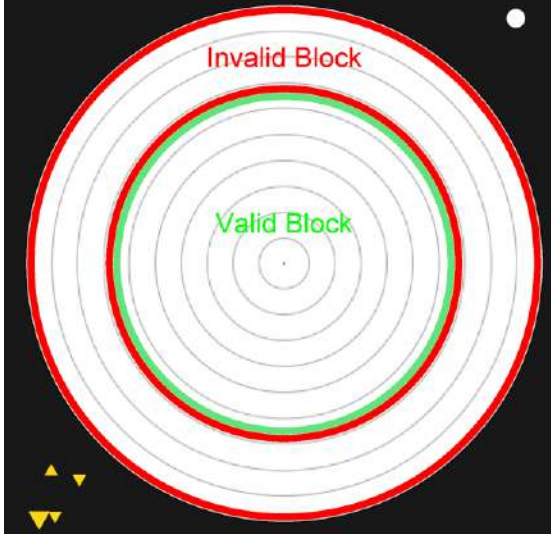


- **Sanity check** (difficulty = 10):
  - What does difficulty equal when:
    - `time_to_mine = one_week?`
    - `time_to_mine = four_weeks?`

```
difficulty *= two_weeks / time_to_mine_prev_2016_blocks
```

A TAMPER-EVIDENT DATABASE

BLOCK DIFFICULTY

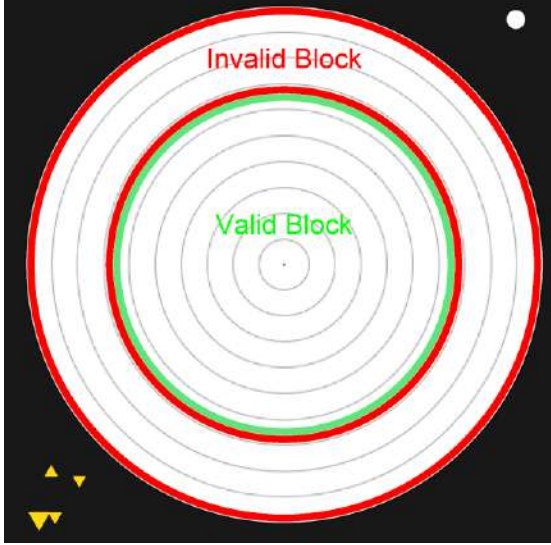


- **Sanity check** (difficulty = 10):
  - What does difficulty equal when:
    - `time_to_mine = one_week?`
      (Answer: 20)
    - `time_to_mine = four_weeks?`

```
difficulty *= two_weeks / time_to_mine_prev_2016_blocks
```

A TAMPER-EVIDENT DATABASE

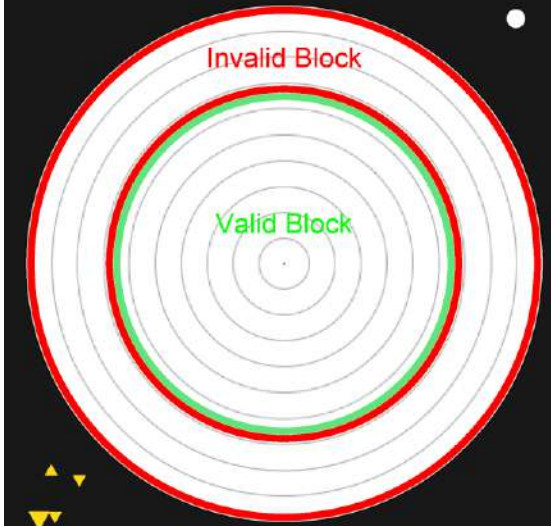
BLOCK DIFFICULTY



- **Sanity check** (difficulty = 10):
  - What does difficulty equal when:
    - `time_to_mine = one_week?`
      (Answer: 20)
    - `time_to_mine = four_weeks?`
      (Answer: 5)

A TAMPER-EVIDENT DATABASE

BLOCK DIFFICULTY



- **Sanity check** (difficulty = 10):
  - What does difficulty equal when:
    - `time_to_mine = one_week?` (Answer: 20)
    - `time_to_mine = four_weeks?` (Answer: 5)
  - Difficulty is inversely proportional to `time_to_mine`.)

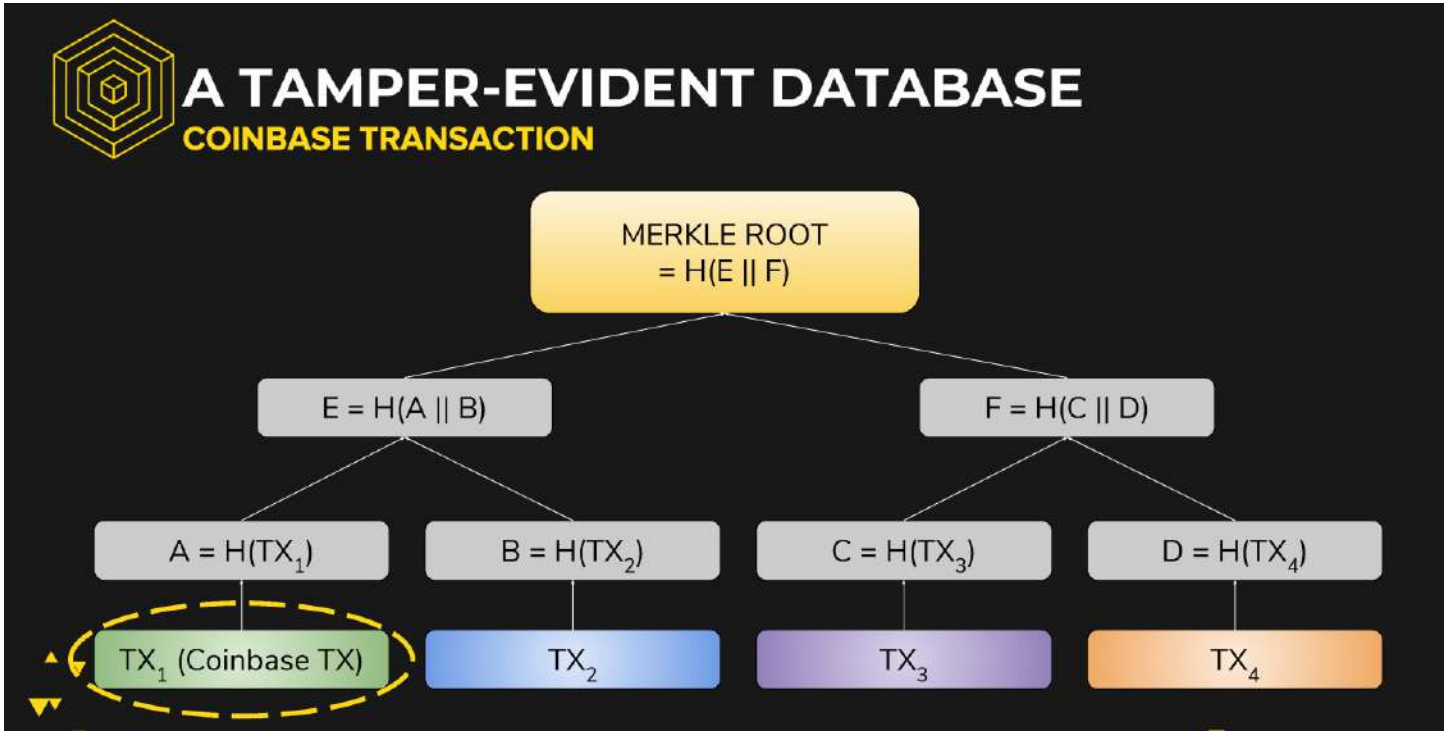
A TAMPER-EVIDENT DATABASE

MINING PSEUDOCODE

```
TARGET = (65535 << 208) / DIFFICULTY;
coinbase_nonce = 0;
while (1) {
      header = makeBlockHeader(transactions, coinbase_nonce);
      for (header_nonce = 0; header_nonce < (1 << 32); header_nonce++){
            if (SHA256(SHA256(makeBlock(header, header_nonce))) <
      TARGET)
                  break; //block found!
      }
      coinbase_nonce++;
}
```

Figure 5.6 : CPU mining pseudocode.

Source: (from Princeton Textbook, 5.2)


A TAMPER-EVIDENT DATABASE
COINBASE TRANSACTION
MERKLE ROOT
= H(E || F)
E = H(A || B)
F = H(C || D)
A = H(TX1)
B = H(TX2)
C = H(TX3)
D = H(TX4)
TX1 (Coinbase TX)
TX2
TX3
TX4

A TAMPER-EVIDENT DATABASE

EXAMPLE BLOCK

Block #485963

Summary	
Number Of Transactions	2055
Output Total	4,819.27194588 BTC
Estimated Transaction Volume	1,770.2727223 BTC
Transaction Fees	1.05055103 BTC
Height	485963 (Main Chain)
Timestamp	2017-09-19 02:11:37
Received Time	2017-09-19 02:11:37
Relayed By	BTC.TOP
Difficulty	1,103,400,932,964.29

Hashes	
Hash	0000000000000000013942c4215cd92308bbce789cfcb349d0b42f031c994eb
Previous Block	0000000000000000004a5b64838b5d96d367a8d4e0a435fd460f972f1fb8f56b
Next Block(s)	
Merkle Root	ddb4970913d63bcb0c32a6d26fb9e792f8cd332ddf9c830a23c3e191608ce51a

Sponsored Link

3 SIGS, ECDSA, AND ADDRESSES

DIGITAL SIGNATURE SCHEMES (DSS)

EXAMPLE

ALICE

BOB

private key:

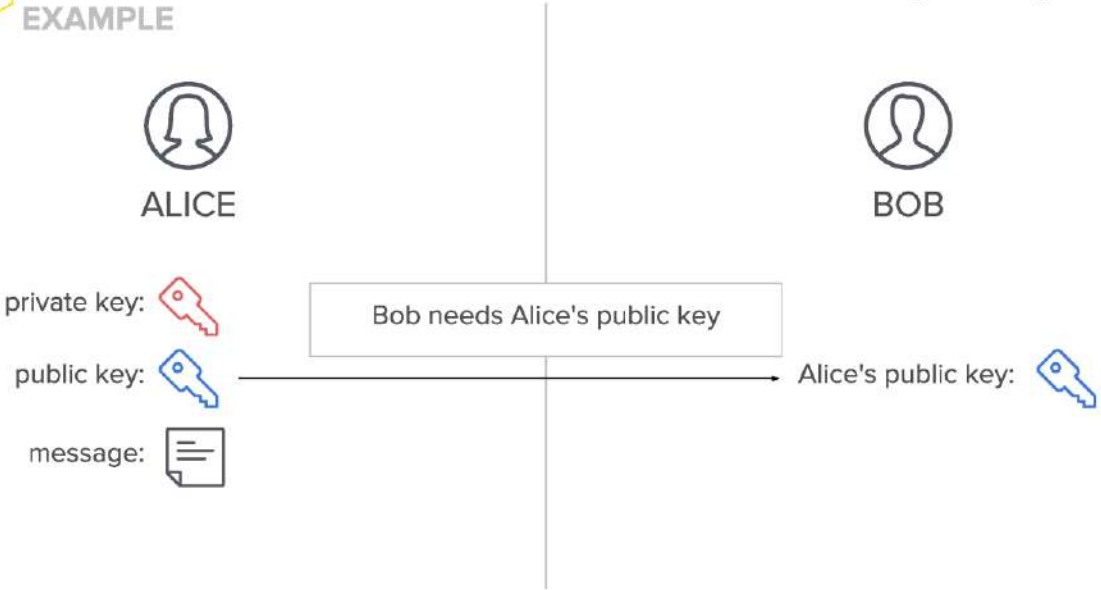
public key:

message:

Alice uses ECDSA to generate private and public keys

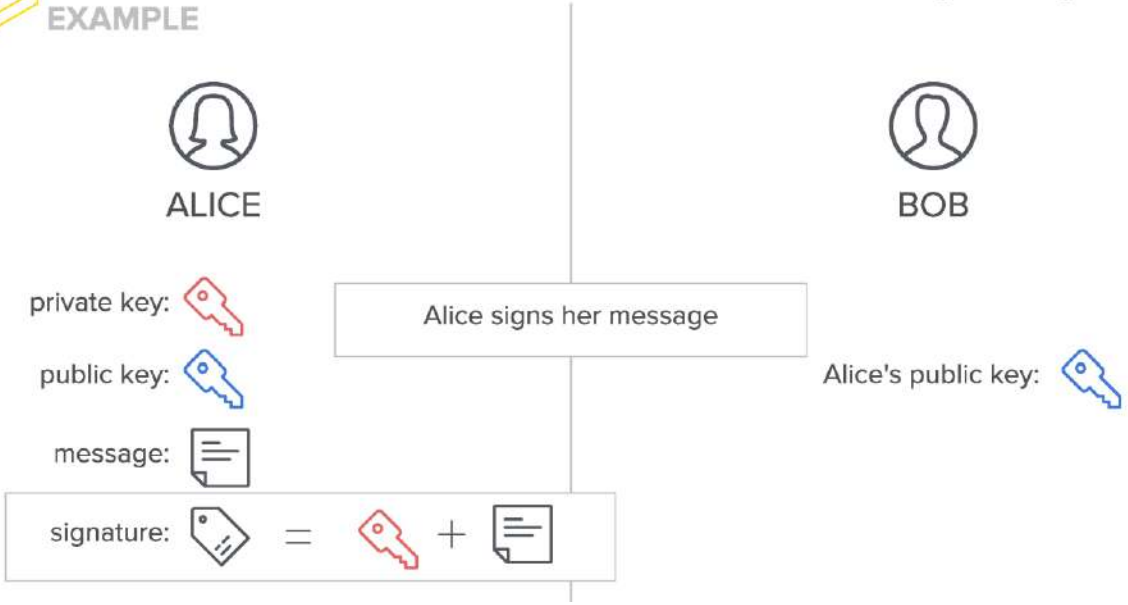
DIGITAL SIGNATURE SCHEMES (DSS)

EXAMPLE



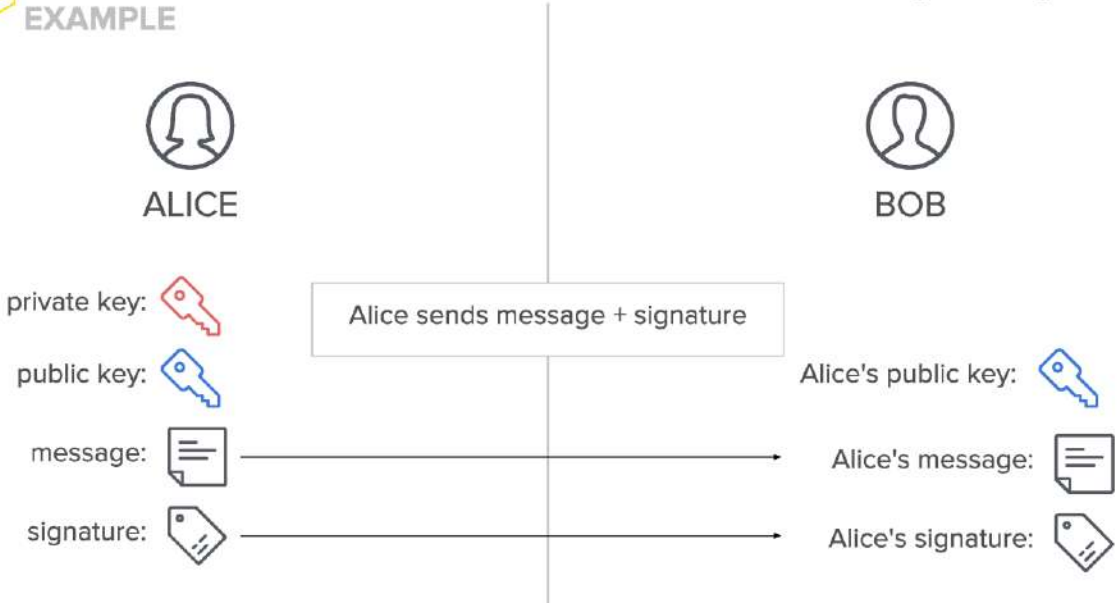
DIGITAL SIGNATURE SCHEMES (DSS)

EXAMPLE



DIGITAL SIGNATURE SCHEMES (DSS)

EXAMPLE



DIGITAL SIGNATURE SCHEMES (DSS)

EXAMPLE

DIGITAL SIGNATURE SCHEMES (DSS)

EXAMPLE

DIGITAL SIGNATURE SCHEMES (DSS)

EXAMPLE

ALICE

EVE

BOB

private key:

public key:

message:

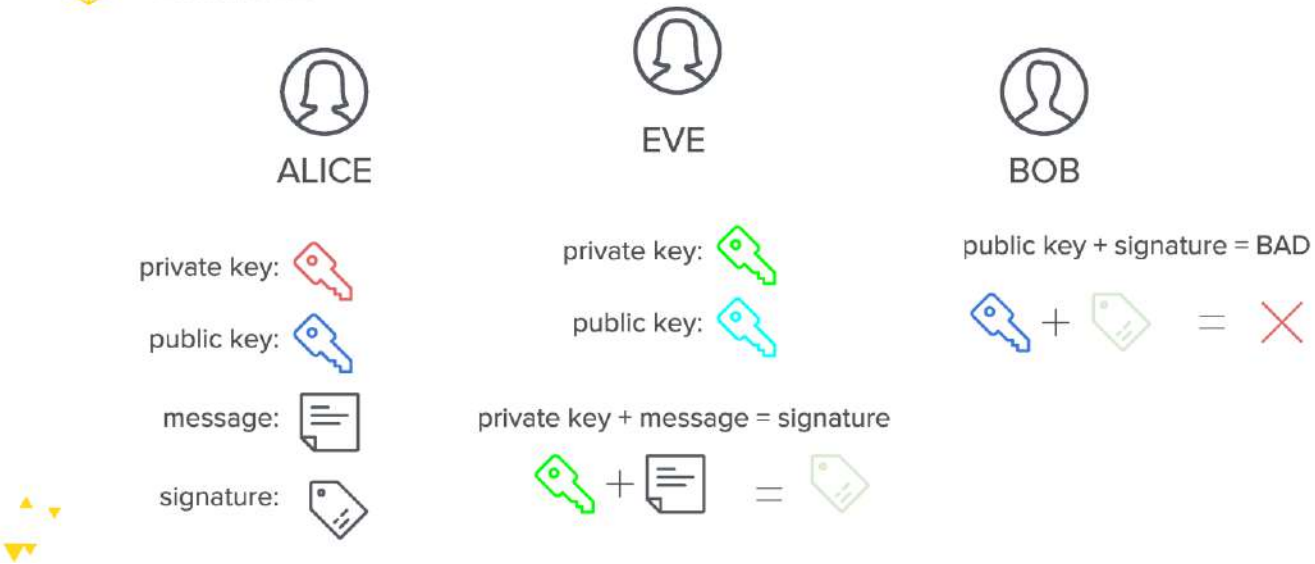
signature:

private key + message = signature

public key + signature = message

DIGITAL SIGNATURE SCHEMES (DSS)

EXAMPLE



DIGITAL SIGNATURE SCHEMES (DSS)

DSS SECURITY DEFINITION

Recipients given the (message, signature) pair should be able to verify:

- **Message Origin**: original sender (owner of private key) has authorized this message/transaction
- **Non-repudiation**: original sender (owner of private key) cannot backtrack
- **Message Integrity**: message cannot have been modified since sending

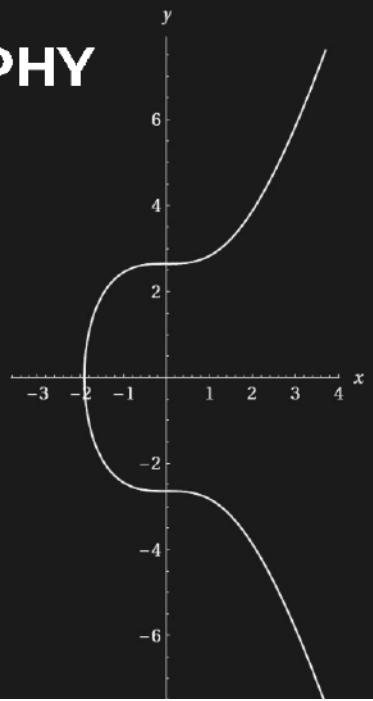
ELLIPTIC CURVE CRYPTOGRAPHY

PART 1: ELLIPTIC CURVE

- Bitcoin uses **ECDSA** (Elliptic Curve Digital Signature Algorithm) to produce private and public keys
- The Elliptic Curve is defined by some mathematical function
  - **Bitcoin's Elliptic Curve**:

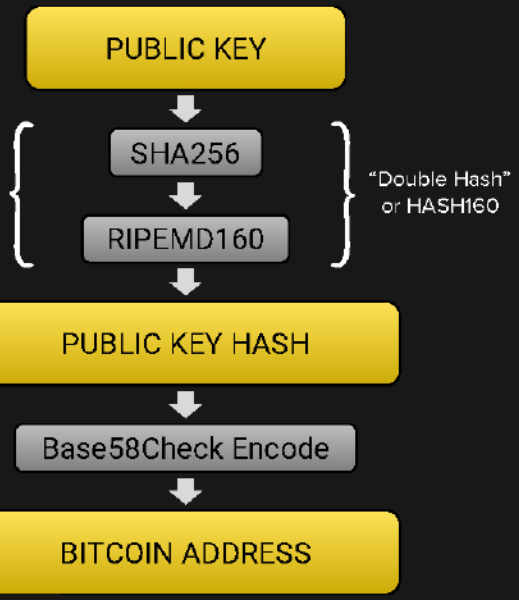
    secp256k1 : $Y^2 = (X^3 + 7)$ over $(\mathbf{F}_p)$
- For cryptographic purposes, we use elliptic curves over a **finite field** (for key size)

Trapdoor function – difficult to invert



PUBLIC KEY TO BITCOIN ADDRESS

PUBLIC KEY TO PUBKEYHASH



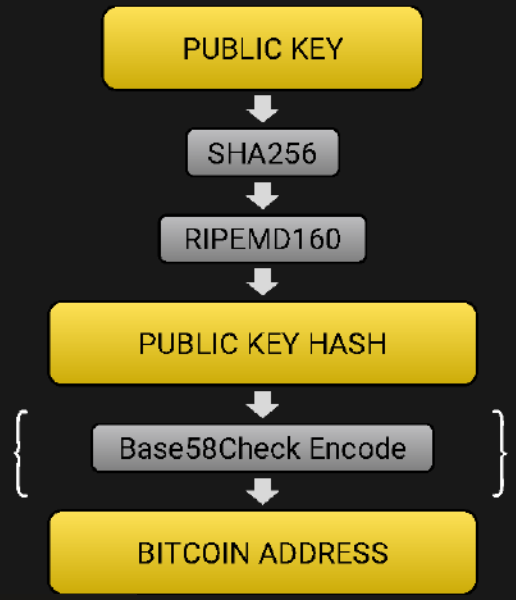

```
PUBKEYHASH = RIPEMD160(SHA256(K))
```

- SHA-256 (Secure Hashing Algorithm)
  - Used extensively in bitcoin scripts and mining
- RIPEMD (RACE Integrity Primitives Evaluation Message Digest)
  - Produces 160-bit (20-byte) number

The name of the game is TAMPER EVIDENT

PUBLIC KEY TO BITCOIN ADDRESS

PUBLIC KEY HASH TO ADDRESS



- Bitcoin Addresses are Base58Check Encoded
  - Base-58 alphabet: 123456789ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijkmnopqrstuvwxyz
  - 58 characters (omits 0, O, I, l)

- prefix: "version byte" based on type of data
  - makes it easy for people to read address
- checksum: 4-byte error-checking code appended to the end of an address
  - `checksum = SHA256(SHA256(prefix + data)`, first 4 bytes

4 BITCOIN SCRIPT

BITCOIN SCRIPTS

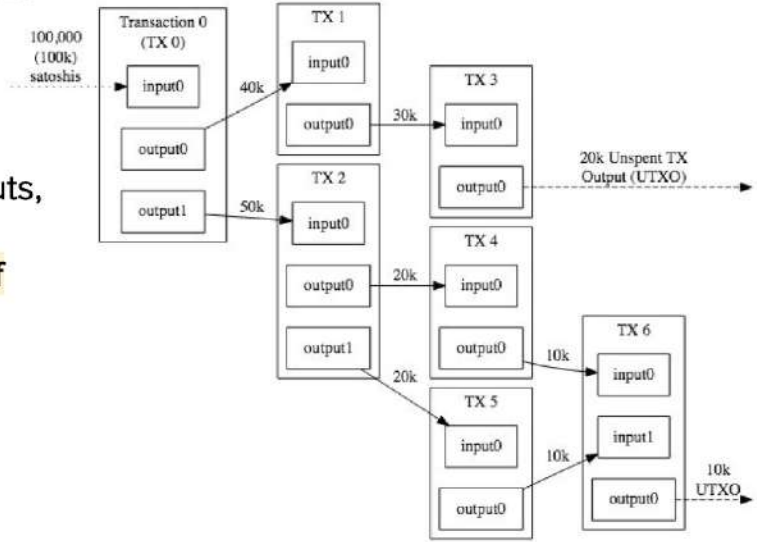
REMEMBER THE UTXO MODEL?

Reminders:

- Bitcoin uses a UTXO model
- Transactions map inputs to outputs,

- Transactions contain signature of owner of funds
- Spending Bitcoin is **redeeming** previous transaction outputs

Source: Bitcoin Developer Guide



Triple-Entry Bookkeeping (Transaction-To-Transaction Payments) As Used By Bitcoin

CONTENTS OF A TRANSACTION

metadata

input(s)

output(s)


```
{
  "hash":"5a42590fbe0a90ee8e8747244d6c84f0db1a3a24e8f1b95b10c9e050990b8b6b",
  "ver":1,
  "vin_sz":2,
  "vout_sz":1,
  "lock_time":0,
  "size":404,
  "in":[
    {
      "prev_out":{
        "hash":"3be4ac9728a0823cf5e2deb2e86fc0bd2aa503a91d307b42ba76117d79280260",
        "n":0
      },

        "scriptSig":"30440..."
    },
    {
      "prev_out":{
        "hash":"7508e6ab259b4df0fd5147bab0c949d81473db4518f81afc5c3f52f91ff6b34e",
        "n":0
      },
      "scriptSig":"3f3a4ce81...."
    }
  ],
  "out":[
    {
      "value":"10.12287097",
      "scriptPubKey":"OP_DUP OP_HASH160 69e02e18b5705a05dd6b28ed517716c894b3d42e OP_EQUALVERIFY OP_CHECKSIG"
    }
  ]
}
```


Source: Princeton Textbook

CONTENTS OF A TRANSACTION - METADATA


```
{
  "hash": "5a42590fbe0a90ee8e8747244d6c84f0db1a3a24e8f1b95b10c9e050990b8b6b",
  "ver": 1,
  "vin_sz": 2,
  "vout_sz": 1,
  "lock_time": 0,
  "size": 404,
  "in": [
      {
          "prev_out": {
              "hash": "3be4ac9728a0823cf5e2deb2e86fc0bd2aa503a91d307b42ba76117d79280260",
              "n": 0
          },
              "scriptSig": "30440..."
      },
      {
          "prev_out": {
              "hash": "7508e6ab259b4df0fd5147bab0c949d81473db4518f8afc5c3f52f91ff6b34e",
              "n": 0
          },
          "scriptSig": "3f3a4ce81...."
      }
  ],
  "out": [
      {
          "value": 10.12287097",
          "scriptPubKey": "OP_DUP OP_HASH160 69e02e18b5705a05dd6b28ed517716c894b3d42e OP_EQUALVERIFY OP_CHECKSIG"
      }
  ]
}
```


hash or "ID"
of this transaction

CONTENTS OF A TRANSACTION - METADATA


```
{
  "hash": "5a42590fbe0a90ee8e8747244d6c84f0db1a3a24e8f1b95b10c9e050990b8b6b",
  "ver": 1,
  "vin_sz": 2,
  "vout_sz": 1,
  "lock_time": 0,
  "size": 404,
  "in": [
      {
          "prev_out": {
              "hash": "3be4ac9728a0823cf5e2deb2e86fc0bd2aa503a91d307b42ba76117d79280260",
              "n": 0
          },
              "scriptSig": "30440..."
      },
      {
          "prev_out": {
              "hash": "7508e6ab259b4df0fd5147bab0c949d81473db4518f8afc5c3f52f91ff6b34e",
              "n": 0
          },
          "scriptSig": "3f3a4ce81...."
      }
  ],
  "out": [
      {
          "value": 10.12287097",
          "scriptPubKey": "OP_DUP OP_HASH160 69e02e18b5705a05dd6b28ed517716c894b3d42e OP_EQUALVERIFY OP_CHECKSIG"
      }
  ]
}
```


size (number) of inputs

size (number) of outputs

hash or "ID"
of this transaction

CONTENTS OF A TRANSACTION - METADATA


```
{
  "hash": "5a42590fbe0a90ee8e8747244d6c84f0db1a3a24e8f1b95b10c9e050990b8b6b",
  "ver": 1,
  "vin_sz": 2,
  "vout_sz": 1,
  "lock_time": 0,
  "size": 404,
  "in": [
      {
          "prev_out": {
              "hash": "3be4ac9728a0823cf5e2deb2e86fc0bd2aa503a91d307b42ba76117d79280260",
              "n": 0
          },
              "scriptSig": "30440..."
      },
      {
          "prev_out": {
              "hash": "7508e6ab259b4df0fd5147bab0c949d81473db4518f8afc5c3f52f91ff6b34e",
              "n": 0
          },
          "scriptSig": "3f3a4ce81...."
      }
  ],
  "out": [
      {
          "value": 10.12287097",
          "scriptPubKey": "OP_DUP OP_HASH160 69e02e18b5705a05dd6b28ed517716c894b3d42e OP_EQUALVERIFY OP_CHECKSIG"
      }
  ]
}
```


version

size (number) of inputs

size (number) of outputs

lock time (useful for scripting)

size of transaction

hash or "ID"
of this transaction

CONTENTS OF A TRANSACTION - INPUTS

{
"hash": "5a42590fbe0a90ee8e8747244d6c84f0db1a3a24e8f1b95b10c9e050990b8b6b",
"ver": 1,
"vin_sz": 2,
"vout_sz": 1,
"lock_time": 0,
"size": 404,
"in": [
{
"prev_out": {
"hash": "3be4ac9728a0823cf5e2deb2e86fc0bd2aa503a91d307b42ba76117d79280260",
"n": 0
},
"scriptSig": "30440..."
},
{
"prev_out": {
"hash": "7508e6ab259b4df0fd5147bab0c949d81473db4518f8afc5c3f52f91ff6b34e",
"n": 0
},
"scriptSig": "3f3a4ce81...."
}
],
"out": [
{
"value": 10.12287097",
"scriptPubKey": "OP_DUP OP_HASH160 69e02e18b5705a05dd6b28ed517716c894b3d42e OP_EQUALVERIFY OP_CHECKSIG"
}

remember these?

CONTENTS OF A TRANSACTION - INPUTS

{
"hash": "5a42590fbe0a90ee8e8747244d6c84f0db1a3a24e8f1b95b10c9e050990b8b6b",
"ver": 1,
"vin_sz": 2,
"vout_sz": 1,
"lock_time": 0,
"size": 404

remember these?

"in": [
{
"prev_out": {
"hash": "3be4ac9728a0823cf5e2deb2e86fc0bd2aa503a91d307b42ba76117d79280260",
"n": 0
},

input 1:

"scriptSig": "30440..."
},
{
"prev_out": {
"hash": "7508e6ab259b4df0fd5147bab0c949d81473db4518f8afc5c3f52f91ff6b34e",
"n": 0
},

input 2:

"scriptSig": "3f3a4ce81...."
}
],
"out": [
{
"value": 10.12287097",
"scriptPubKey": "OP_DUP OP_HASH160 69e02e18b5705a05dd6b28ed517716c894b3d42e OP_EQUALVERIFY OP_CHECKSIG"

ID of previous transactions being referenced

CONTENTS OF A TRANSACTION - INPUTS


```
{
  "hash": "5a42590fbe0a90ee8e8747244d6c84f0db1a3a24e8f1b95b10c9e050990b8b6b",
  "ver": 1,
  "vin_sz": 2,
  "vout_sz": 1,
  "lock_time": 0,
  "size": 404,
  "in": [
    {
      "prev_out": {
        "hash": "3be4ac9728a0823cf5e2deb2e86fc0bd2aa503a91d307b42ba76117d79280260",
        "n": 0
      },
input 1:  "scriptSig": "30440..."
    },
    {
      "prev_out": {
        "hash": "7508e6ab259b4df0fd5147bab0c949d81473db4518f8afc5c3f52f91ff6b34e",
        "n": 0
      },
input 2:  "scriptSig": "3f3a4ce81...."
    }
  ],
  "out": [
    {
      "value": 10.12287097",
      "scriptPubKey": "OP_DUP OP_HASH160 69e02e18b5705a05dd6b28ed517716c894b3d42e OP_EQUALVERIFY OP_CHECKSIG"
    }
```


index of input in previous transaction

index of input in previous transaction

ID of previous transactions being referenced

CONTENTS OF A TRANSACTION - INPUTS


```
{
  "hash": "5a42590fbe0a90ee8e8747244d6c84f0db1a3a24e8f1b95b10c9e050990b8b6b",
  "ver": 1,
  "vin_sz": 2,
  "vout_sz": 1,
  "lock_time": 0,
  "size": 404,
  "in": [
    {
      "prev_out": {
        "hash": "3be4ac9728a0823cf5e2deb2e86fc0bd2aa503a91d307b42ba76117d79280260",
        "n": 0 ← index of input in previous transaction
      },
input 1:  "scriptSig": "30440..." ← signature used to redeem previous transaction output
    },
    {
      "prev_out": {
        "hash": "7508e6ab259b4df0fd5147bab0c949d81473db4518f8afc5c3f52f91ff6b34e",
        "n": 0 ← index of input in previous transaction
      },
input 2:  "scriptSig": "3f3a4ce81...." ← signature used to redeem previous transaction output
    }
  ],
  "out": [
    {
      "value": 10.12287097",
      "scriptPubKey": "OP_DUP OP_HASH160 69e02e18b5705a05dd6b28ed517716c894b3d42e OP_EQUALVERIFY OP_CHECKSIG"
    }
```

CONTENTS OF A TRANSACTION - OUTPUTS


```
{
    "hash": "5a42590fbe0a90ee8e8747244d6c84f0db1a3a24e8f1b95b10c9e050990b8b6b",
    "ver": 1,
    "vin_sz": 2,
    "vout_sz": 1,
    "lock_time": 0,
    "size": 404,
    "in": [
        {
            "prev_out": {
                "hash": "3be4ac9728a0823cf5e2deb2e86fc0bd2aa503a91d307b42ba76117d79280260",
                "n": 0
            },
                "scriptSig": "30440..."
        },
        {
            "prev_out": {
                "hash": "7508e6ab259b4df0fd5147bab0c949d81473db4518f8afc5c3f52f91ff6b34e",
                "n": 0
            },
            "scriptSig": "3f3a4ce81...."
        }
    ],
```


output amount (how much BTC is being sent)


```
    "out": [
        {
            "value": 10.12287097",
            "scriptPubKey": "OP_DUP OP_HASH160 69e02e18b5705a05dd6b28ed517716c894b3d42e OP_EQUALVERIFY OP_CHECKSIG"
        }
    ]
}
```


type of script

output script

BITCOIN SCRIPTS

REMINDERS

Output “addresses” are actually scripts.

“scriptPubKey”: “OP_DUP OP_HASH160 69e02e18b5705a05dd6b28ed517716c894b3d42e OP_EQUALVERIFY OP_CHECKSIG”

➔ This particular Output Script: “This amount can be redeemed by the **public key** that hashes to address X, plus a **signature** from the owner of that public key”

- Inputs and outputs through scripting allows for future extensibility of Bitcoin.
- **Script or “Bitcoin Scripting Language”**: Language built specifically for Bitcoin
  - Stack based
  - Native support for cryptography
  - Simple, not turing complete (no loops)

BITCOIN SCRIPTS

P2PKH EXAMPLE

"scriptPubKey": "OP_DUP OP_HASH160 69e02e18b5705a05dd6b28ed517716c894b3d42e OP_EQUALVERIFY OP_CHECKSIG"

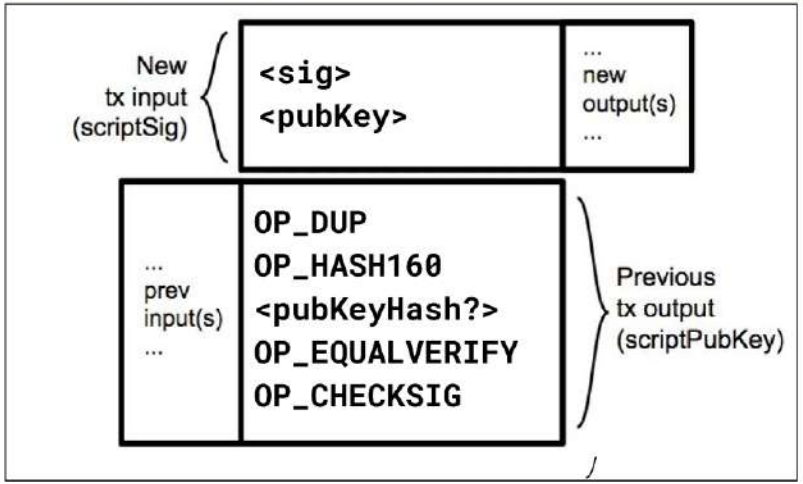


Figure: Two transactions along with their input and output scripts

Source: Princeton textbook

- **locking script:** found in previous transaction output, specifies requirements for redeeming transaction
- **unlocking script (scriptSig):** found in transaction input, redeems the output of a previous transaction
- bitcoin validating node will execute the locking and unlocking scripts in sequence

BITCOIN SCRIPTS

P2PKH EXAMPLE

"scriptPubKey": "OP_DUP OP_HASH160 69e02e18b5705a05dd6b28ed517716c894b3d42e OP_EQUALVERIFY OP_CHECKSIG"

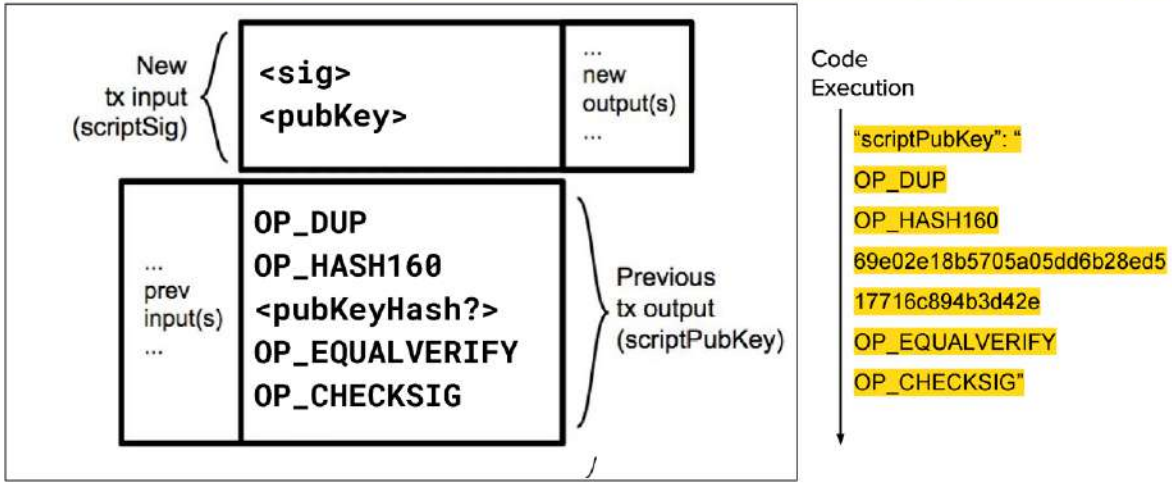
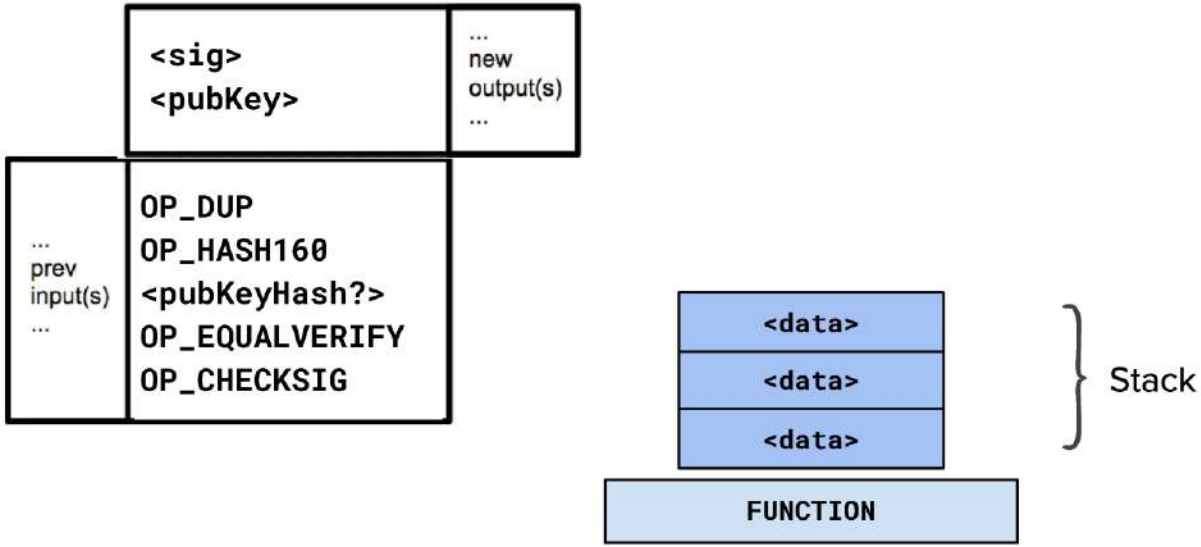


Figure: Two transactions along with their input and output scripts

BITCOIN SCRIPTS

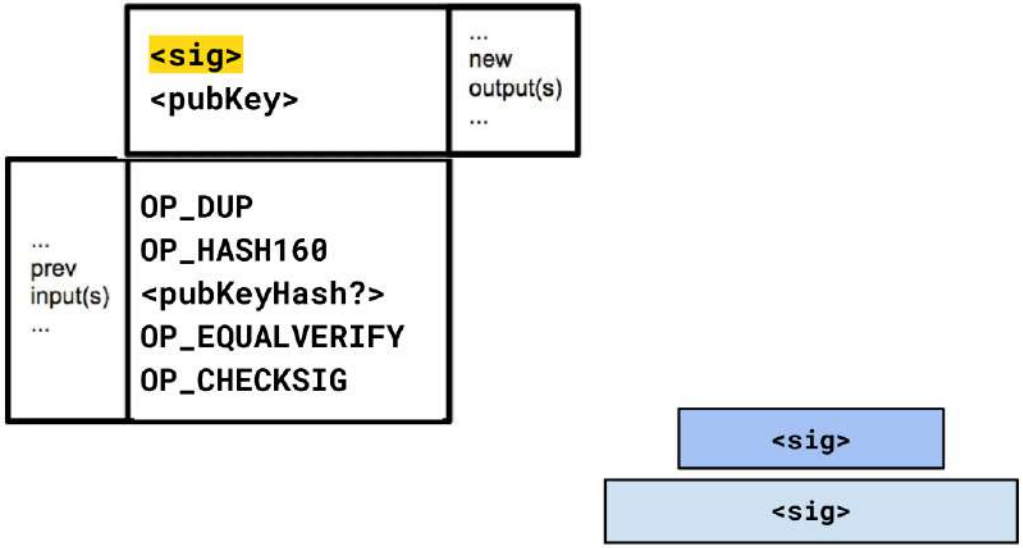
P2PKH EXAMPLE EXECUTION



Source: Princeton textbook

BITCOIN SCRIPTS

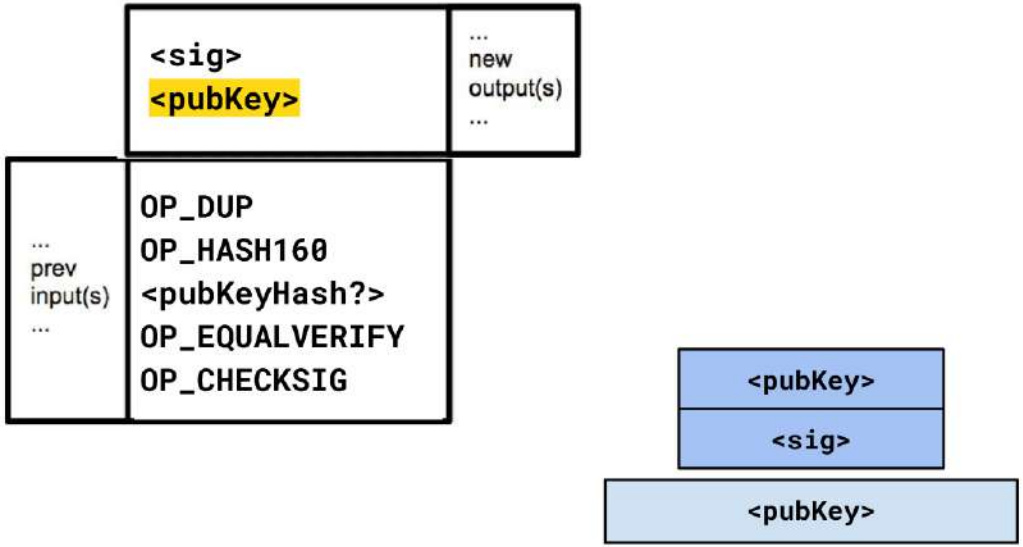
P2PKH EXAMPLE EXECUTION



Source: Princeton textbook

BITCOIN SCRIPTS

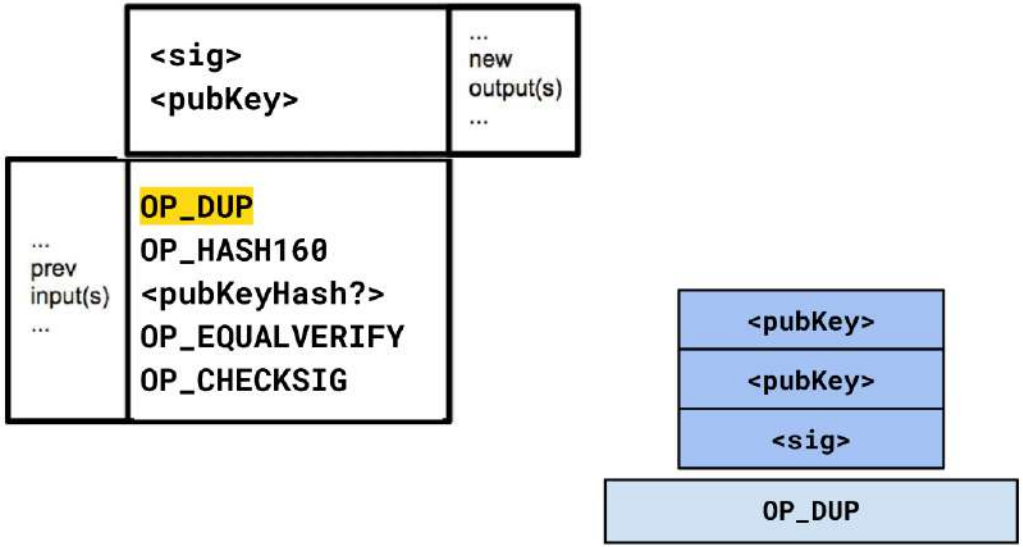
P2PKH EXAMPLE EXECUTION



Source: Princeton textbook

BITCOIN SCRIPTS

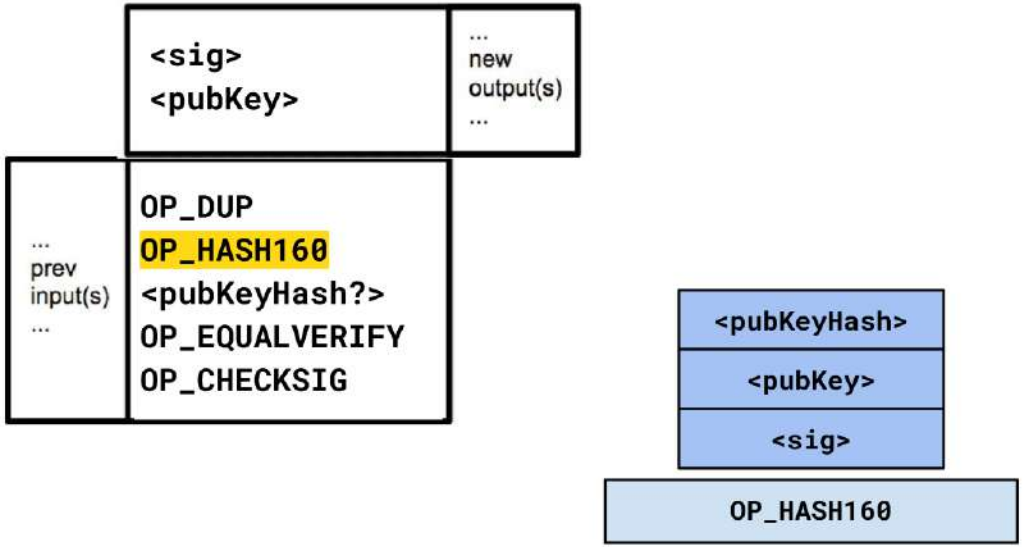
P2PKH EXAMPLE EXECUTION



Source: Princeton textbook

BITCOIN SCRIPTS

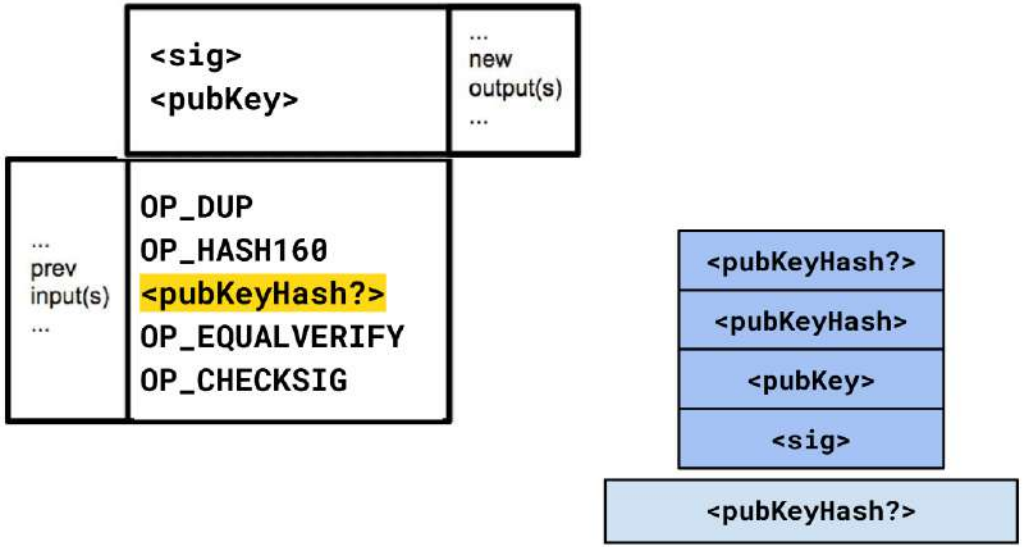
P2PKH EXAMPLE EXECUTION



Source: Princeton textbook

BITCOIN SCRIPTS

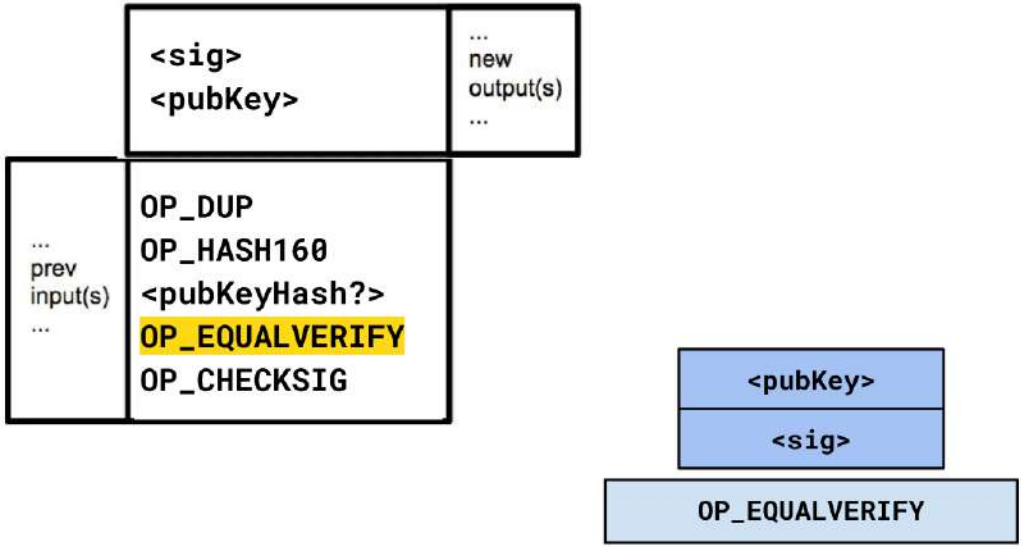
P2PKH EXAMPLE EXECUTION



Source: Princeton textbook

BITCOIN SCRIPTS

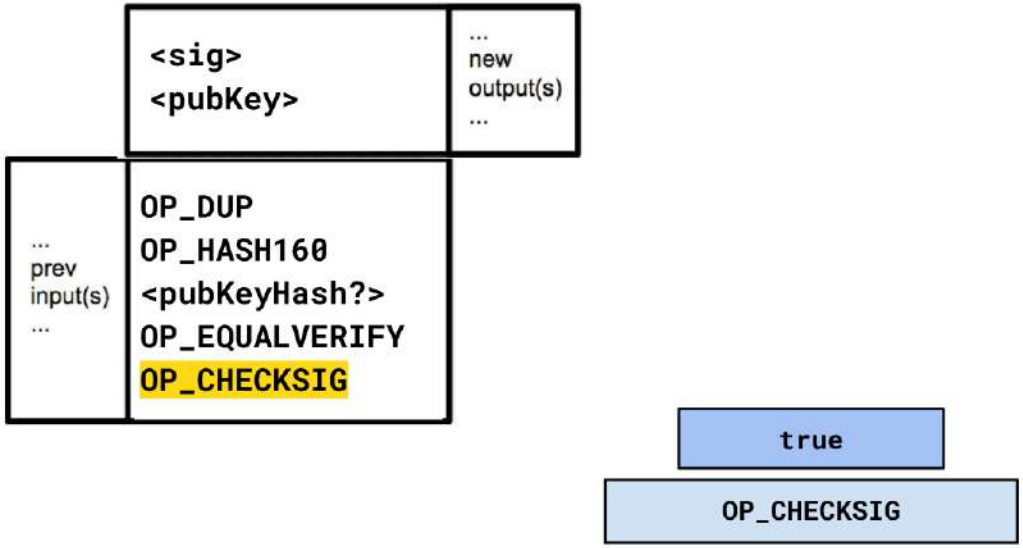
P2PKH EXAMPLE EXECUTION



Source: Princeton textbook

BITCOIN SCRIPTS

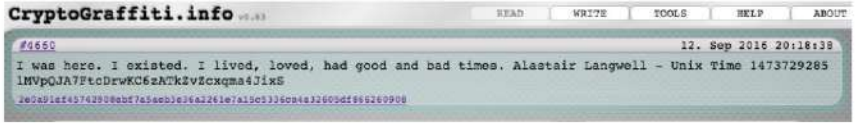
P2PKH EXAMPLE EXECUTION



Source: Princeton textbook

EXTRA: BITCOIN SCRIPTS

PROOF OF BURN



Output script:

OP_RETURN
<arbitrary data>

How to write arbitrary data into the Bitcoin blockchain?

Proof of Burn

- OP_RETURN throws an error if reached
- Output script can't be spent - you prove that you destroyed some currency
- Anything after OP_RETURN is not processed, so arbitrary data can be entered

Use cases

- Prove existence of something at a particular point in time
  - Ex. A word you coined, hash of a document/music/creative works
- Bootstrap CalCoin by requiring that you destroy some Bitcoin to get CalCoin

5

P2PKH & P2SH

P2PKH VS P2SH

WHO SPECIFIES THE SCRIPT?

- In Bitcoin, senders specify a locking script, recipients provide an unlocking script
- **Pay-to-Pub-Key-Hash (P2PKH):** Vendor (recipient of transaction) says "Send your coins to the hash of this Public Key."
  - Simplest case
  - By far the most common case
- **Pay-to-Script-Hash (P2SH):** Vendor says "Send your coins to the hash of this Script; I will provide the script and the data to make the script evaluate to true when I redeem the coins."
  - A vendor cannot say, "To pay me, write a complicated output script that will allow me to spend using multiple signatures."

Simple P2PKH script - recipient only has to provide signature and public key

P2PKH VS P2SH

WHY P2SH?

Why Pay-to-Script-Hash?

- Offloads complicated script writing to recipients
- Makes more sense from a payer-payee standpoint
  - Merchant (rather than customer) is responsible for writing correct and secure script
  - Customer doesn't care what the script actually is
- P2SH is the most important improvement to Bitcoin since inception
- Example: **MultiSig**
  - *M* of *N* specified signatures can redeem and spend the output of this transaction

P2PKH VS P2SH

MULTISIG EXAMPLE

e.g. Gillian, Brian and Nadir are in charge of a joint account and make all spending decisions together:
2 of 3 MultiSig

ScriptSig (Unlocking Script)	<sig1> <sig2> ... <sigm>	sig1 sig2
ScriptPubKey (Redeeming Script)	m <pubKey1> <pubKey2> ... N OP_CHECKMULTISIG	2 PubKey1 PubKey2 PubKey3 3 OP_CHECKMULTISIG
Prev. Output Script (Locking Script)	OP_HASH160 <hash?> OP_EQUALVERIFY	OP_HASH160 <hash?> OP_EQUALVERIFY

Gillian, Brian and Nadir provide the script for multisig redemption of coins

Sender of transaction only has to provide this output script!

Demo

- https://anders.com/blockchain/public-private-keys/keys.html

Next class – Bitcoin Mechanics

Recommended Readings:

- Public / Private Keys & Signing Demo: https://www.youtube.com/watch?v=xIDL_akeras

References

Slides mainly adopted from

- Blockchain @ Berkeley : https://blockchain.berkeley.edu/
- Blockchain @ Princeton : http://bitcoinbook.cs.princeton.edu/

Lecture 4
BITCOIN IRL: WALLETS, MINING, & MORE

Blockchain on the news

South Korea Launches $90M Blockchain Education Course

https://ethereumworldnews.com/south-korea-90m-blockchain-education/

Jp N. ✅ • 3.
Consultant at Komodo Platform. To stay in touch...
1g • Düzenlendi

Yesterday someone transferred $195,000,000 worth of Bitcoin.
They paid less than 10 cents in transfer fees.
Try to find a better way to transfer millions of dollars...

On Monday, the ministry's information and communication departments divulged that it had hosted its first ever lecture on distributed ledger technologies (DLT). This is reportedly the local government's first move to roll out a "blockchain technology development strategy" course that is backed by a supposed 100 billion Korean Won ($90 million USD) investment from forward-thinking regulators.

As per local media, the course has currently undertaken 42 students, who were selected via an amassment of applications and interviews to determine the best candidates for this fully-fledged course. Following the first lecture, students will need to work through six months of eight hour days that consist of tutelage, learning the ropes of the tech and more activities that are intended to put the 42 scholars through the wringer.

Image Courtesy of KiNews

LECTURE OVERVIEW

1. WALLET TYPES
2. WALLET MECHANICS
3. MINING
4. REAL WORLD MINING
5. CHANGING BITCOIN

1 WALLET TYPES

TYPES OF USERS

KEY COMPONENTS

Not every client is a miner
What if I don't have a powerful computer?

Not every client has the entire blockchain (160+ GB)
What if I just want to send bitcoins with my phone?

Not every client is directly connected to the network
What if I don't need to make regular transactions?

Not every client has a wallet
What if I have a separate wallet client?

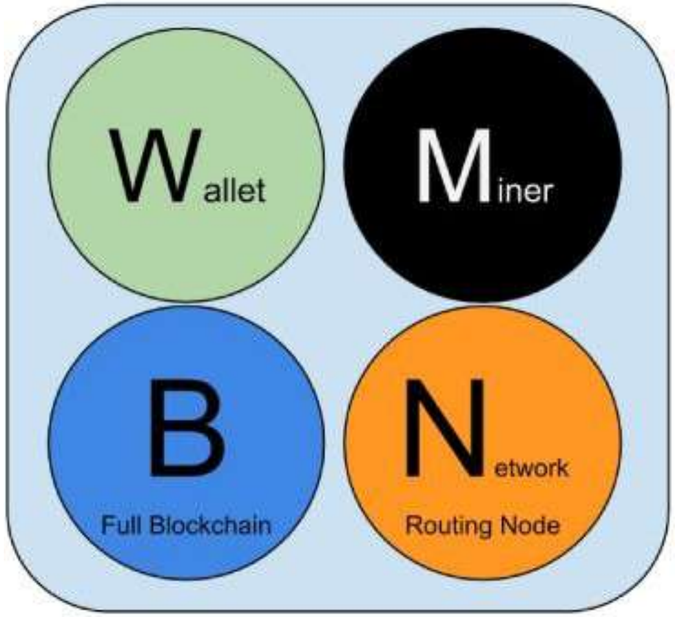


Image source: Mastering Bitcoin

TYPES OF USERS

EVERYONE

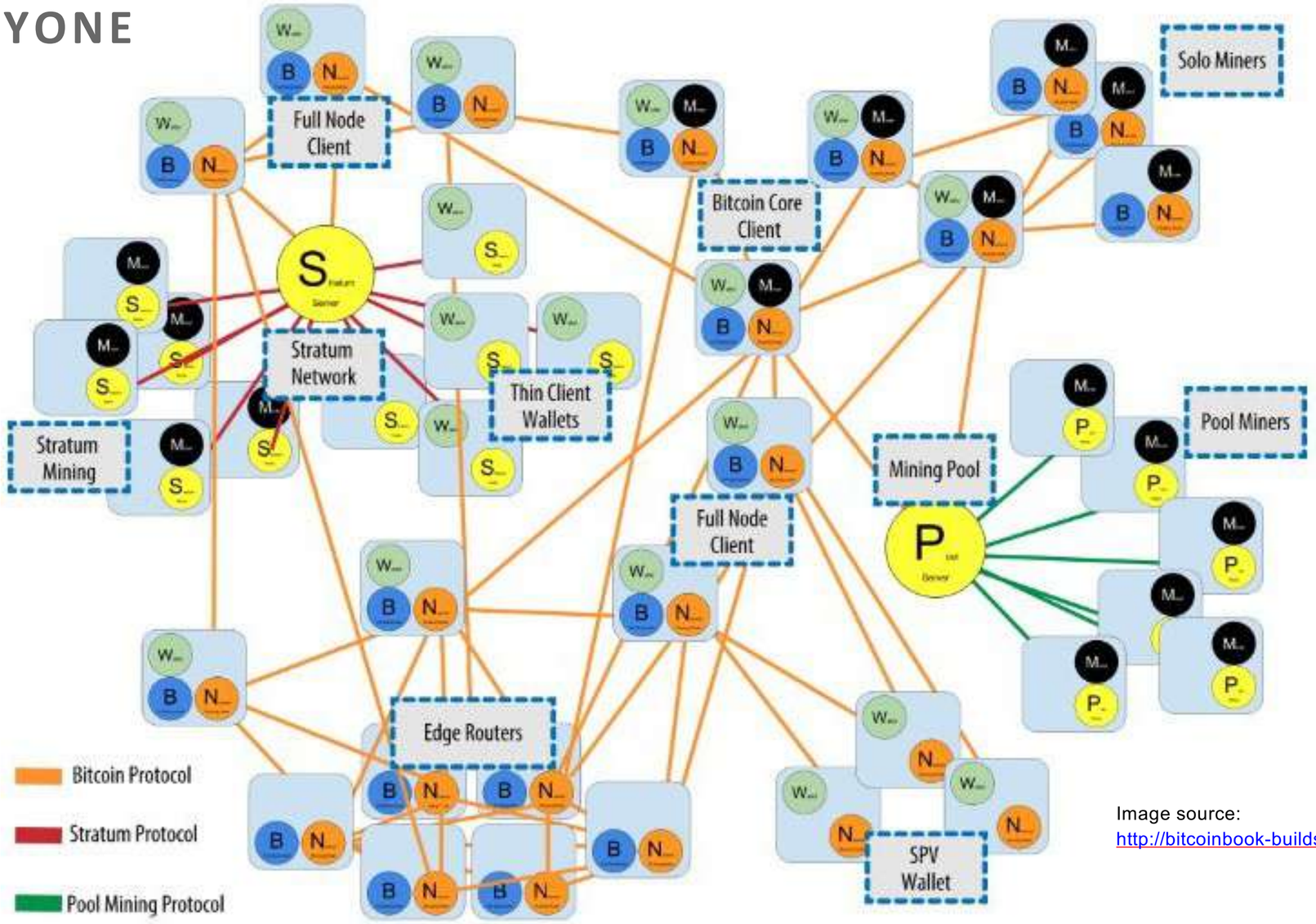


Image source:
http://bitcoinbook-builds.mkvd.net/translations/vi/chapter-6.html

BITCOIN WALLETS

KEY MANAGEMENT

To secure our **identity**, we need to secure our **private key**

How do we manage all of our keys? With <u>wallets</u>!

BITCOIN WALLETS

WALLET TYPES

What do wallets do?

- Keep track of your private key
- Store, send & receive, and list transactions
- Maybe some other related fancy functionality

BITCOIN WALLETS

HOT AND COLD

Wallet Forms

- Smartphone apps
  - Mycelium, AirBitz
- Online web-wallets
  - Blockchain.info, coinbase.com
- Paper Wallets
  - Bitcoinpaperwallet.com
  - Bitaddress.org
- Hardware Wallets
  - Ledger, Trezor, Case, KeepKey
- Brain Wallet

Hot Wallet

Cold Storage

BITCOIN WALLETS

HOT WALLETS

Wallet Forms

- Smartphone apps
  - Mycelium, AirBitz
- Online web-wallets
  - Blockchain.info, coinbase.com
- Paper Wallets
  - Bitcoinpaperwallet.com
  - Bitaddress.org
- Hardware Wallets
  - Ledger, Trezor, Case, KeepKey
- Brain Wallet

} Hot Wallet

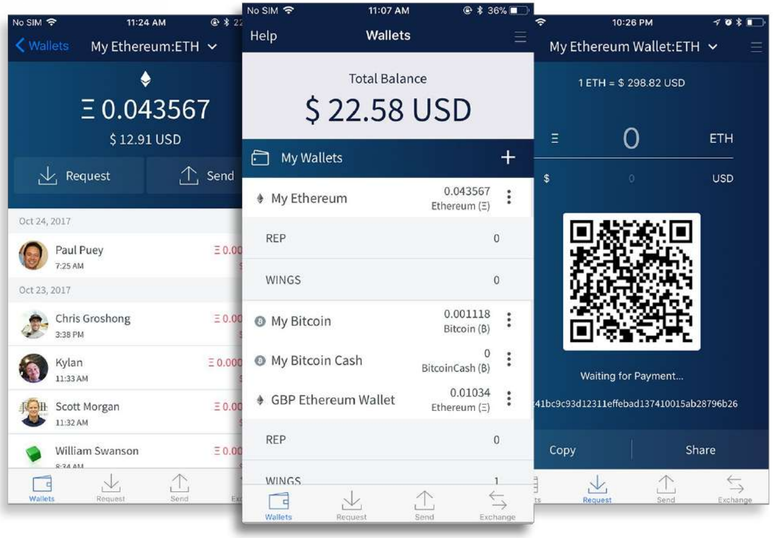
BITCOIN WALLETS

HOT WALLETS

coinbase

BLOCKCHAIN

airBitz



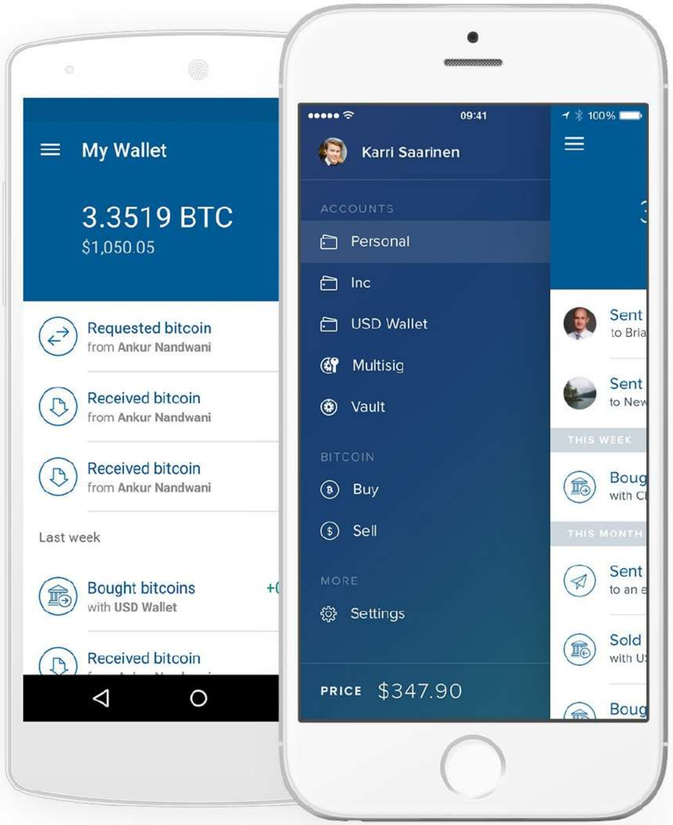


Image sources:
https://blockchain.info/
https://wallet.mycelium.com
https://airbitz.co/bitcoin-wallet/
https://www.coinbase.com/mobile

BITCOIN WALLETS

COLD STORAGE

Wallet Forms

- Smartphone apps
  - Mycelium, AirBitz
- Online web-wallets
  - Blockchain.info, coinbase.com
- Paper Wallets
  - Bitcoinpaperwallet.com
  - Bitaddress.org
- Hardware Wallets
  - Ledger, Trezor, Case, KeepKey
- Brain Wallet

} Cold Storage

BITCOIN WALLETS

PAPER WALLETS

Image source: https://bitcoinpaperwallet.com/

Going Offline

ZIP

Your wallet may be vulnerable to prying eyes when you are generating the keys and printing them out. Although the wallet generator on this website is SSL-encrypted, it's still possible for someone to be snooping on you. (For example, your computer might have malware that broadcasts your screen to a remote location.) The most important safety measure is to **go offline** and run the javascript wallet generator on your own computer instead of this website.

Here's how »

TLDR: DOWNLOAD THE ZIP FILE OR GET THE UBUNTU LIVECD AND RUN THE WALLET GENERATOR WITH YOUR INTERNET CONNECTION TURNED OFF.

BITCOIN WALLETS

HARDWARE WALLETS

Image sources:
https://www.ledgerwallet.com/
https://trezor.io/
https://choosecase.com/

BITCOIN WALLETS

BRAIN WALLETS

- Simply memorize your private key

L2Skyj3pJK3nc7wgr9afokGL89dP

WV3iHQJvZiy2zEwvXDQReAgg

using a recommended bitcoin wallet
storing crypto on a hardware wallet
writing private keys on a post it and putting it on your computer

BLOCKCHAIN AT BERKELEY

BLOCKCHAIN AT BERKELEY

BITCOIN WALLETS

BRAIN WALLETS

Brain wallets are a mnemonic, or collection of words/phrases

- Convenient way to memorize your private key
- Easier to have something that you can turn into your private key
- Not very secure, as humans aren't as random as we think we are

multiply	accuse
scrap	fuel
submit	nose
select	hope
adjust	chair
end	afraid

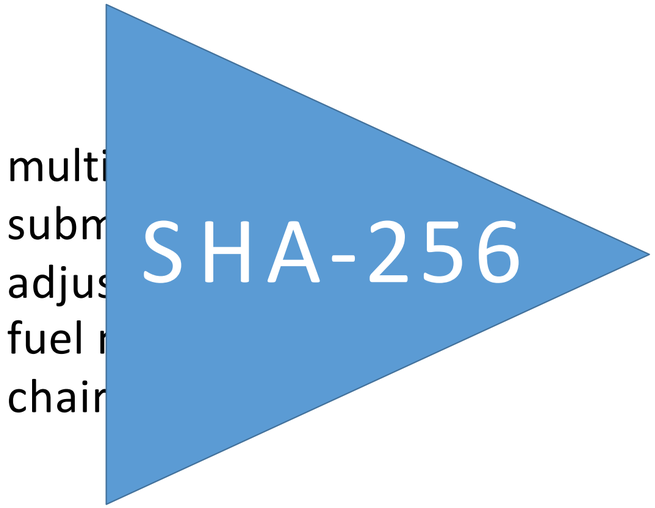
BITCOIN WALLETS

KEY STRETCHING

multiply scrap
submit select
adjust end accuse
fuel nose hope
chair afraid

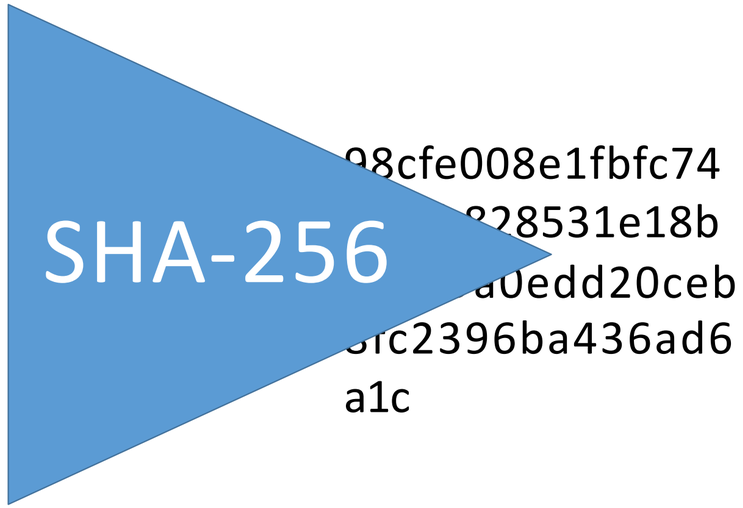
BITCOIN WALLETS

KEY STRETCHING



BITCOIN WALLETS

KEY STRETCHING



BITCOIN WALLETS

KEY STRETCHING

98cfe008e1fbfc74
770fb828531e18b
a4c19a0edd20ceb
8fc2396ba436ad6
a1c

BITCOIN WALLETS

KEY STRETCHING

98cfe008e1fbfc74
770fb828531e18b
a4c19a0edd20ceb
8fc2396ba436ad6
a1c

BITCOIN WALLETS

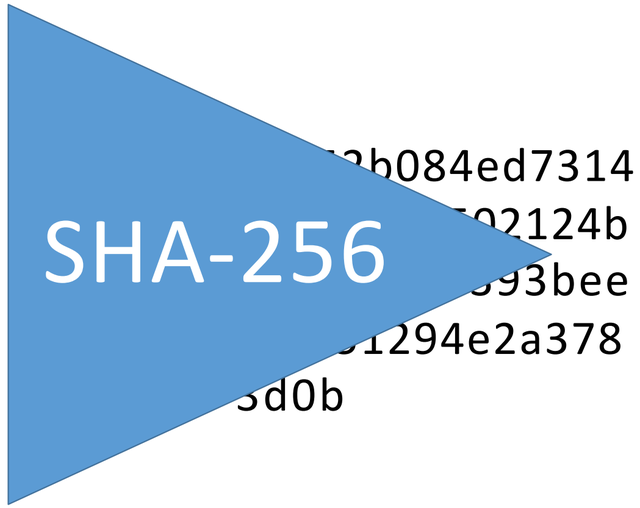
KEY STRETCHING

98cfe008e
770fb8285
a4c19a0ec
8fc2396ba
a1c

SHA-256

BITCOIN WALLETS

KEY STRETCHING



BITCOIN WALLETS

KEY STRETCHING

a4552b084ed7314
415b9367502124b
f84be086a393bee
b0fb51294e2a378
3d0b

BITCOIN WALLETS

CHOOSING A WALLET

Bitcoin.org/en/choose-your-wallet

Features

Multisignature
- 2/3 access control

Privacy
- TOR support
- New addresses for each transaction

Security

Network connection
- Full node
- 3rd party

Who holds the private keys?
- You: mycelium, airbitz, blockchain.info
- Developer: coinbase.com

Image source: https://bitcoin.org

HOW DO I GET BITCOIN?

BITCOIN ATMS

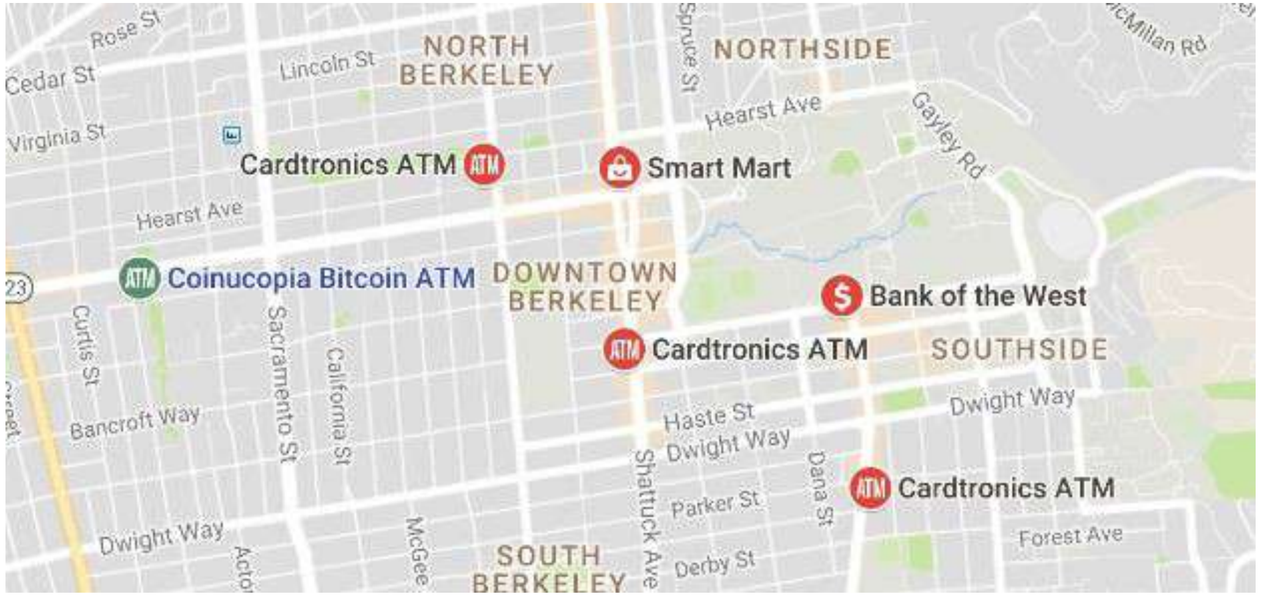


Image sources:
https://www.google.com/maps
https://coinucopia.io/

HOW DO I GET BITCOIN?

EXCHANGES

- Exchanges:
  - https://bitcoin.org/en/exchanges
- Trading between different types of currency
- Centralized and decentralized exchanges, security, easy of access, etc.

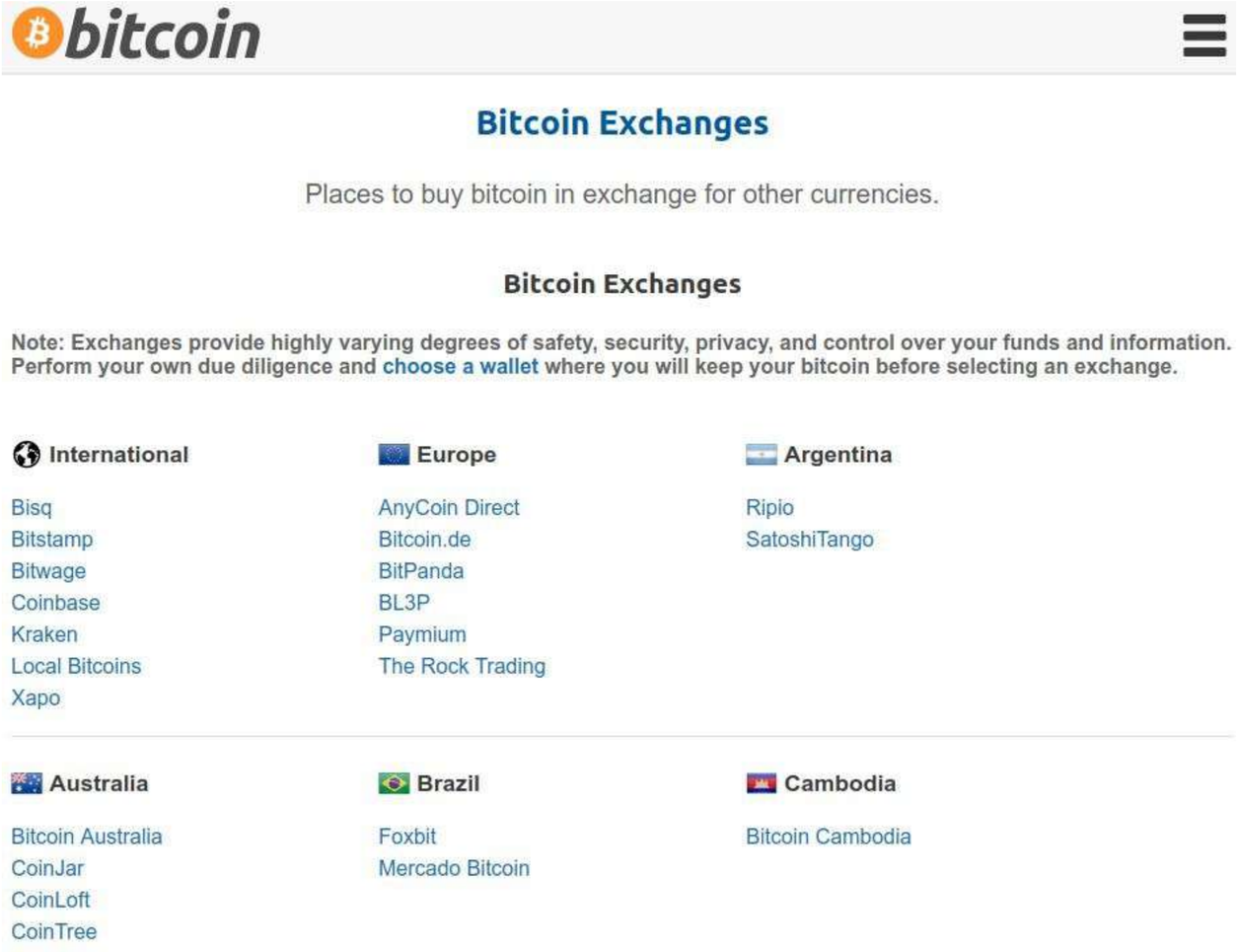


Image source: https://bitcoin.org/en/exchanges

HOW DO I GET BITCOIN?

DECENTRALIZED EXCHANGES

Decentralized exchanges don't rely on a third party service to hold customer's funds or private keys

- Trades are P2P
- Trustless
- Bitshares, Bisq (ex Bitsquare), Openledger, Airswap, Etherdelta, etc.

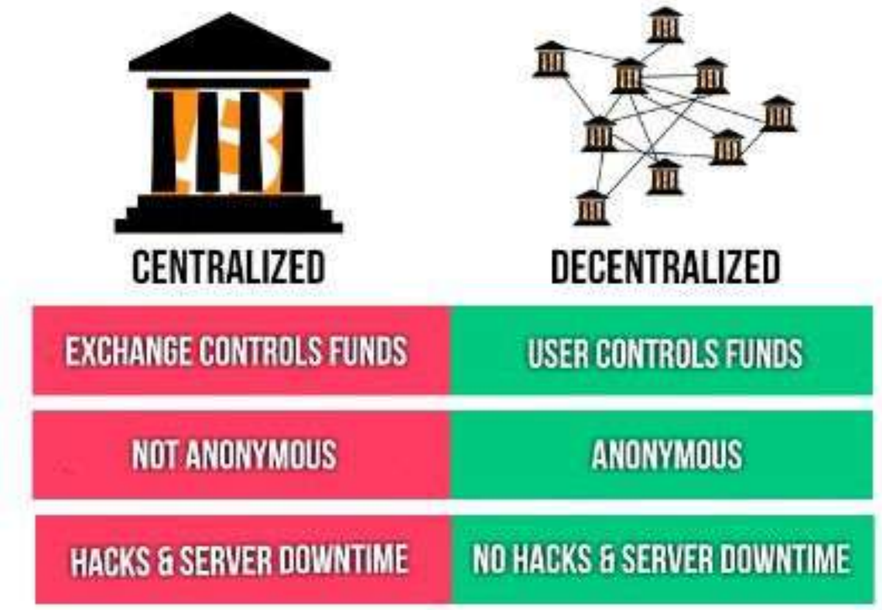


Image source: https://www.cryptocompare.com/exchanges/guides/what-is-a-decentralized-exchange/

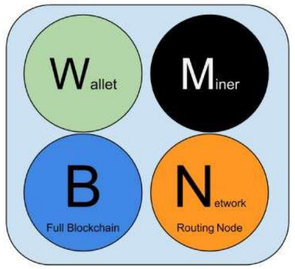
2 WALLET MECHANICS

SIMPLE PAYMENT VERIFICATION

THIN CLIENTS

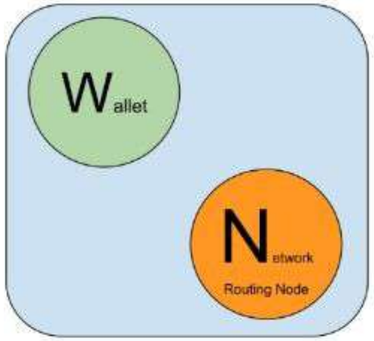
Simple Payment Verification (SPV) is method for verifying if particular transactions are included in a block without downloading the entire block

- Keep track of your transactions only
- Lightweight or thin clients



Reference Client (Bitcoin Core)

Contains a Wallet, Miner, full Blockchain database, and Network routing node on the bitcoin P2P network.



Lightweight (SPV) wallet

Contains a Wallet and a Network node on the bitcoin P2P protocol, without a blockchain.

Image source: http://bitcoinbook-builds.mkvd.net/translations/vi/chapter-6.html

SIMPLE PAYMENT VERIFICATION

THIN CLIENTS

Assumption: Incoming block headers are not from a false chain

- Connect to many different nodes
- Long term, chain is probably honest
- Can't really afford to put the entire blockchain on your phone, so having a thin client is a decent tradeoff

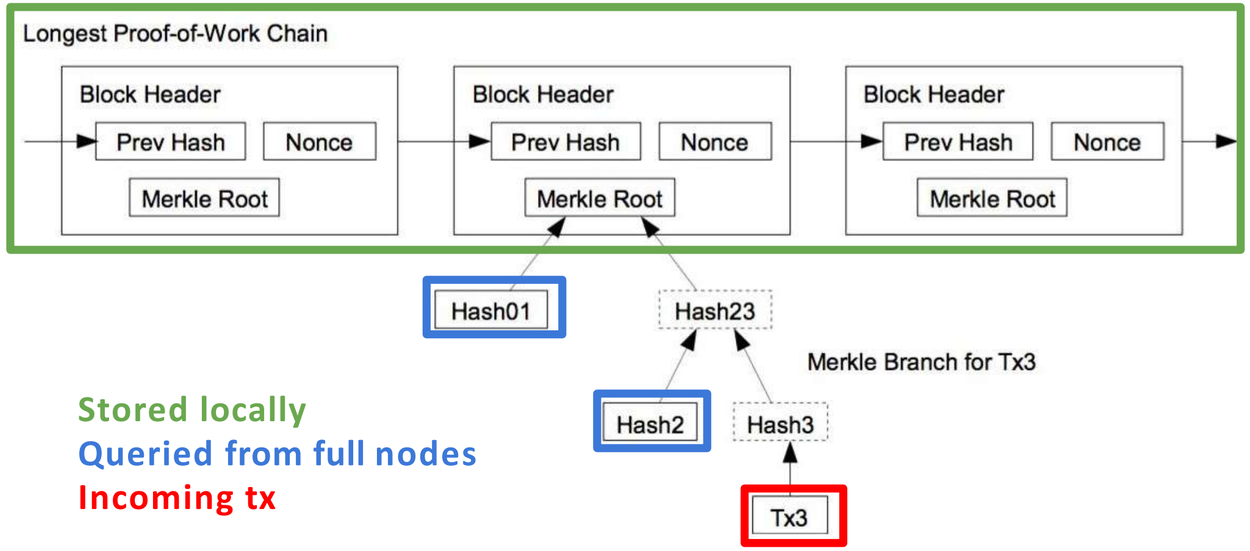
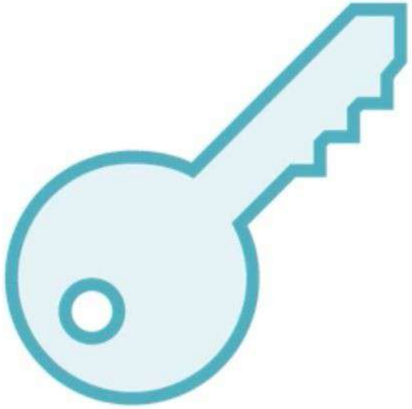


Image source: Mastering Bitcoin

MULTISIGNATURE

M-OF-N TRANSACTIONS

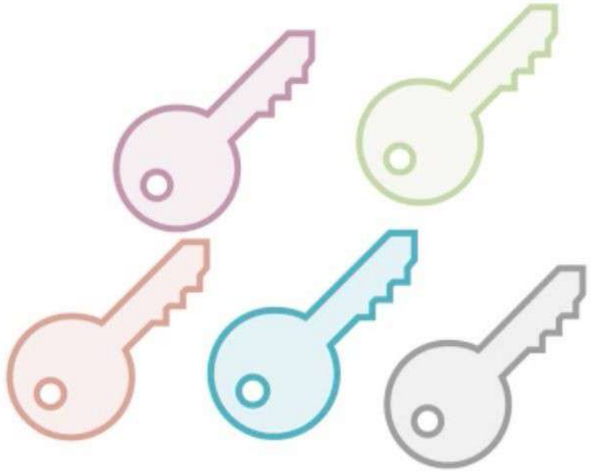
Multi-person Account Access



Regular Bitcoin Addresses

Each account has 1 key (or seed)

Any single person can steal funds



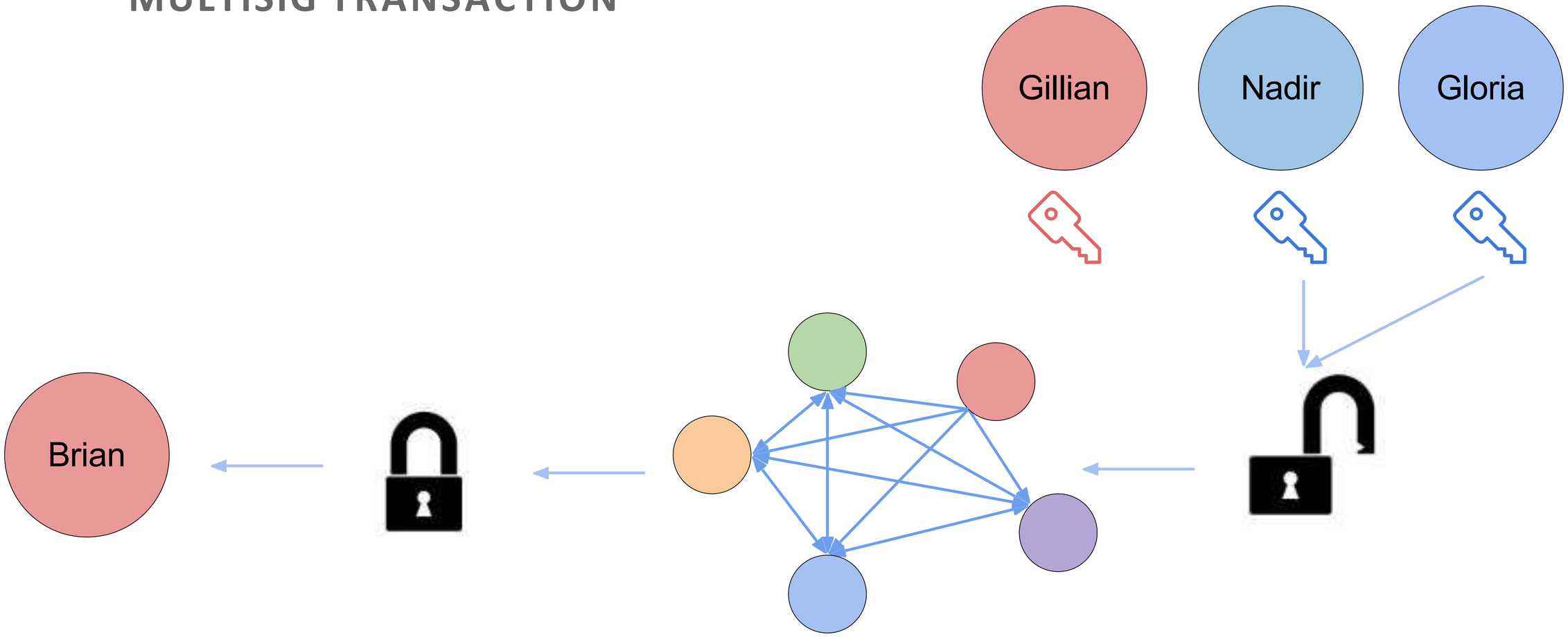
Multisignature Addresses

Multiple signatures needed

Ex: 3 of 5 signatures

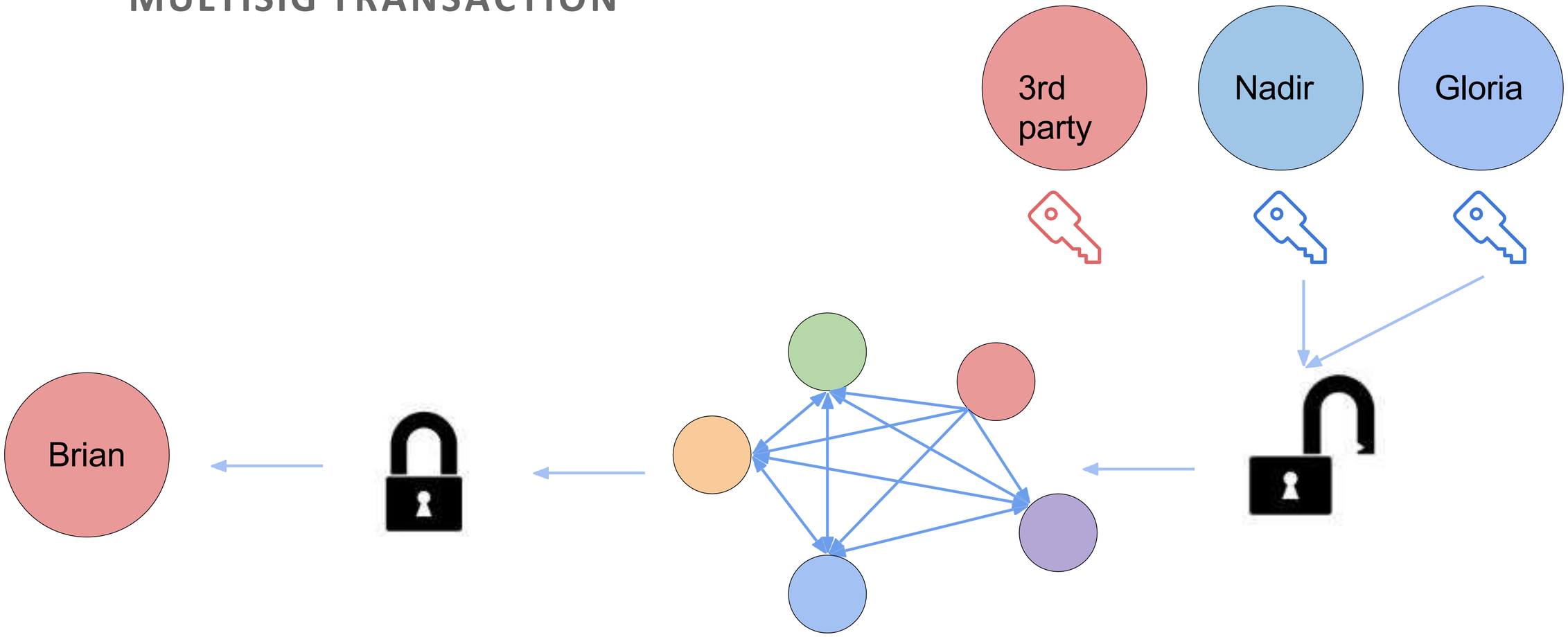
BITCOIN MECHANICS

MULTISIG TRANSACTION



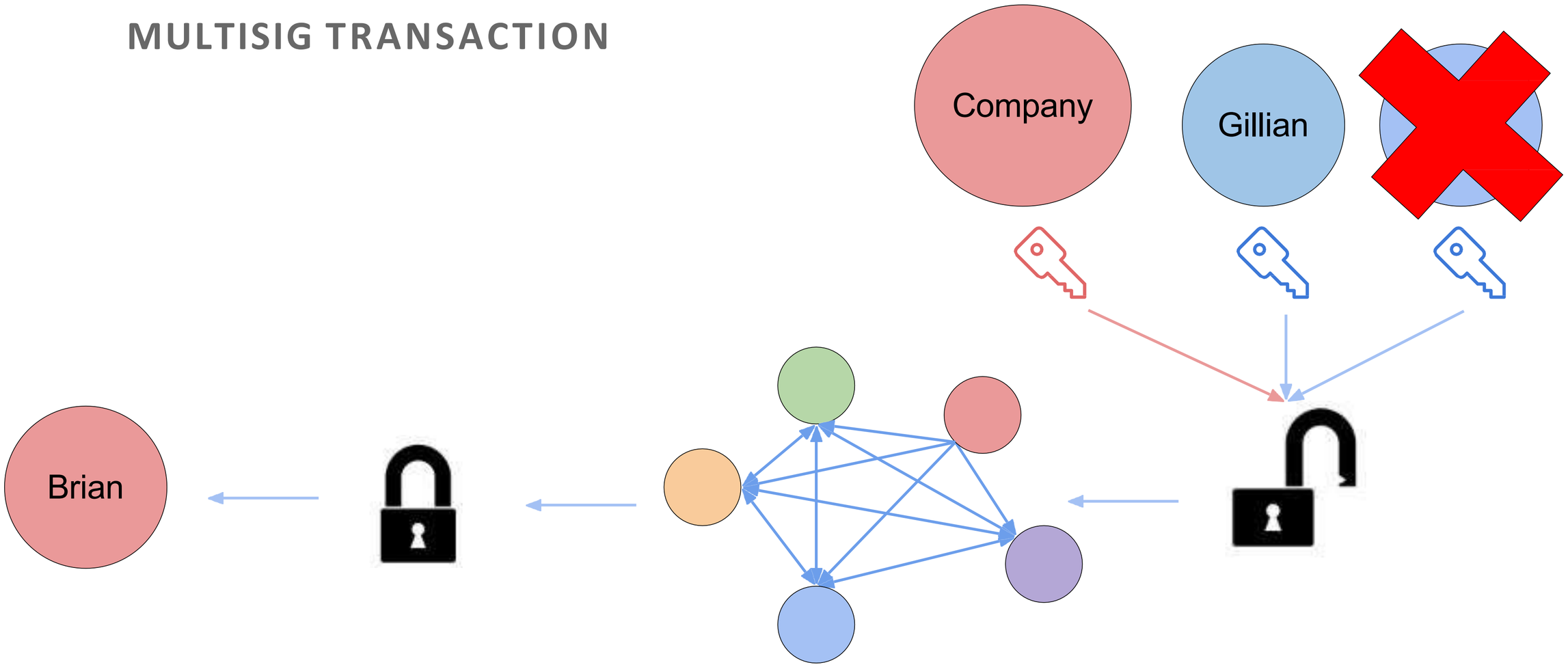
BITCOIN MECHANICS

MULTISIG TRANSACTION



BITCOIN MECHANICS

MULTISIG TRANSACTION



BITCOIN MECHANICS

KEY GENERATION PRACTICES

- Best practice is to never reuse pseudonyms
- Why?
  - Someone should not be able to determine how much bitcoin you own
  - Compromising one key is independent of the other ones
  - Keys are computationally easy to generate anyways
- Wallet software will handle this

WALLET BACKUPS

JBOK WALLETS

- **JBOK (Just a Bunch Of Keys)**
  - New backup required for every new key pair
  - Or, generate a bunch of keys when first started
  - Not too convenient because you have to store every key pair

WALLET BACKUPS

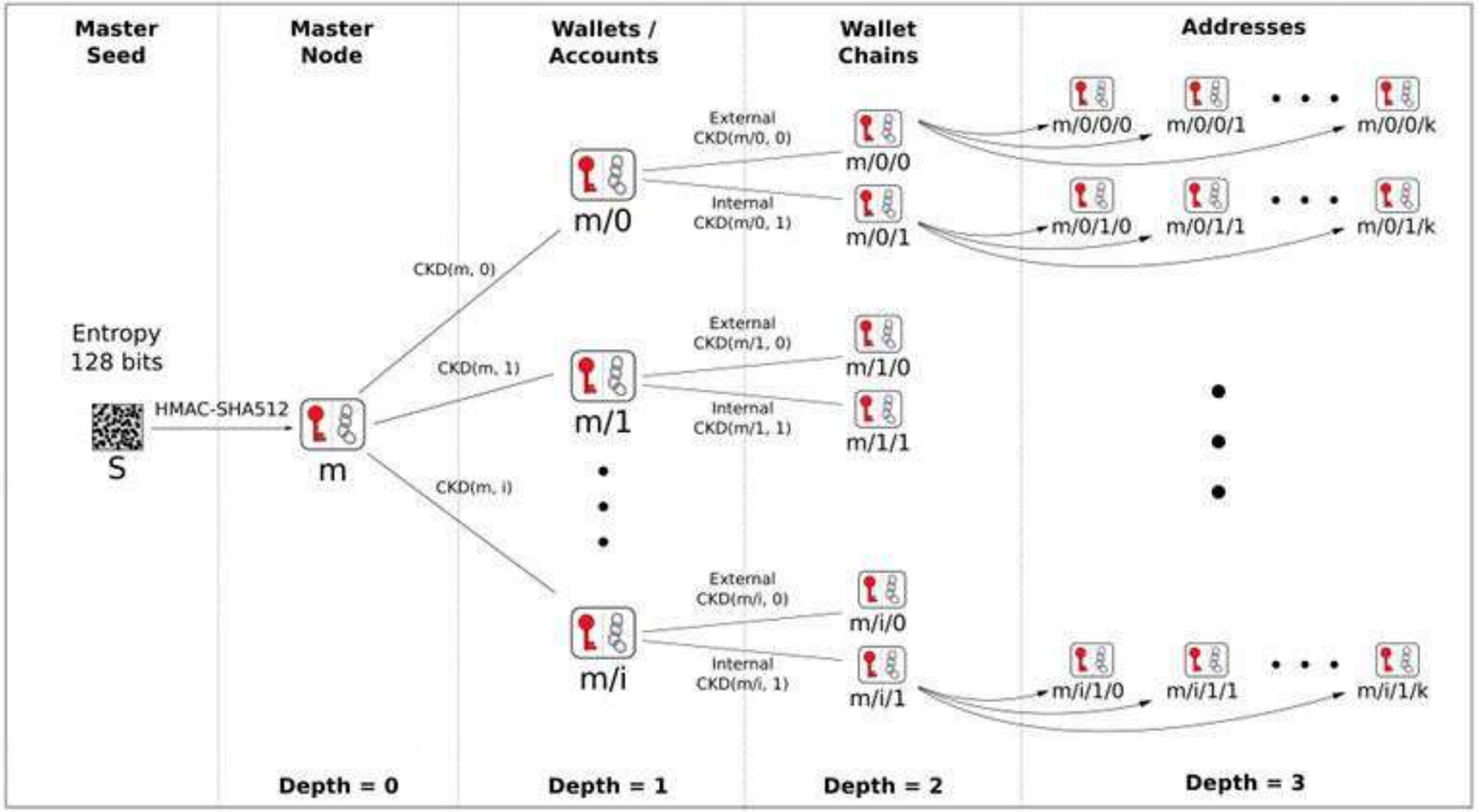
HD WALLETS

- **HD (Hierarchical Deterministic) Wallets**
  - Deterministic, and more convenient to know a seed, or master key
  - Use a one-way hash function with seed and index number
  - Exchanges use these

BITCOIN WALLETS

WALLET BACKUPS

BIP 32 - Hierarchical Deterministic Wallets



Child Key Derivation Function ~ $CKD(x,n) = HMAC\text{-}SHA512(x_{Chain}, x_{PubKey} || n)$

3 MINING

3.1 MINING REVIEW

RECIPE FOR MINING

OVERVIEW

A full-fledged Bitcoin miner must:

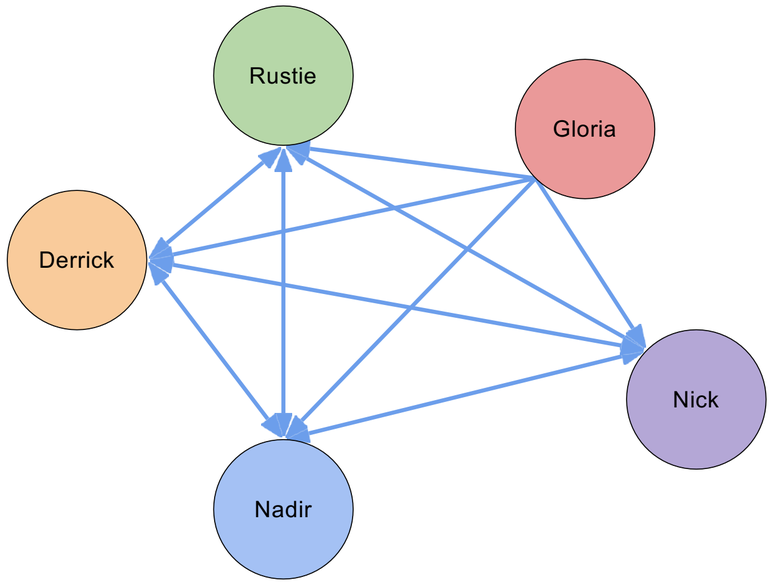
1. **Download** the entire Bitcoin blockchain
2. **Verify** incoming transactions
3. **Create** a block
4. **Find** a valid nonce
5. **Broadcast** your block
6. **Profit**!

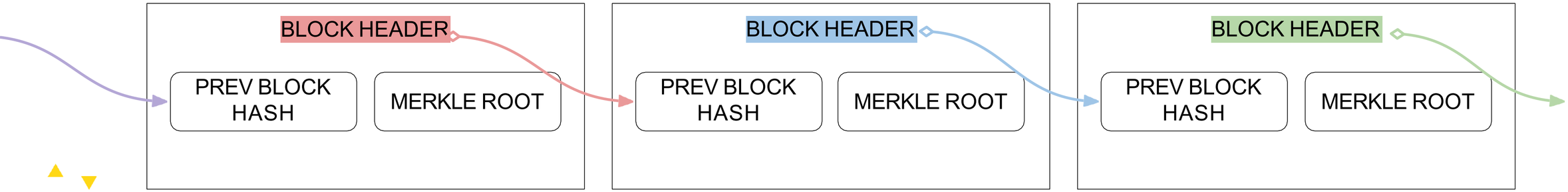
Image source: http://www.coindesk.com/information/how-to-set-up-a-miner/

RECIPE FOR MINING

STEP 0: DOWNLOAD THE BLOCKCHAIN

- Get blocks from your peers
- Download the entire blockchain
  - Start from the genesis block
- Stay up to date

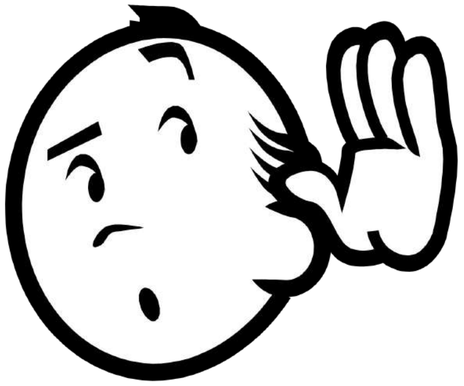




RECIPE FOR MINING

STEP 1: VERIFY TRANSACTIONS

- Listen to the Bitcoin network for transactions
- Unconfirmed (pending) transactions sit in the **mempool** for a miner to include it in a block
- Verify incoming transactions by running the unlocking script (remember P2PKH and P2SH?)



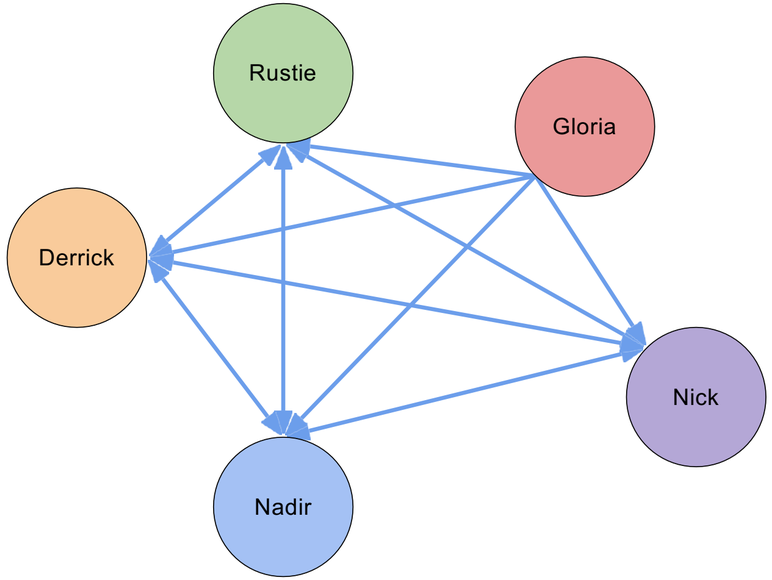
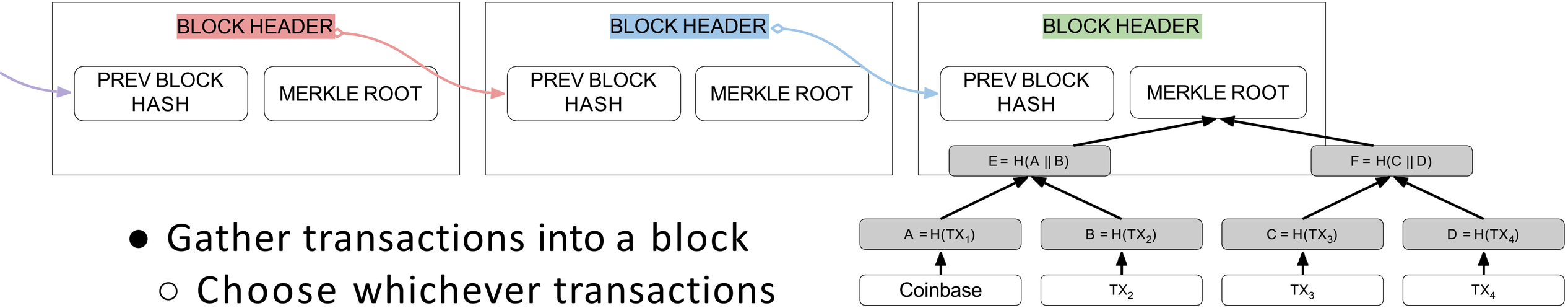


Image sources:
https://ru-clip.com/video/rVrb6rrRvNQ/what-is-a-mempool-and-how-to-speed-up-unconfirmed-transactions.html
https://englishinreims.wordpress.com/toeic-training/practice-toeic-listening-part-i/

RECIPE FOR MINING

STEP 2: CREATE A BLOCK

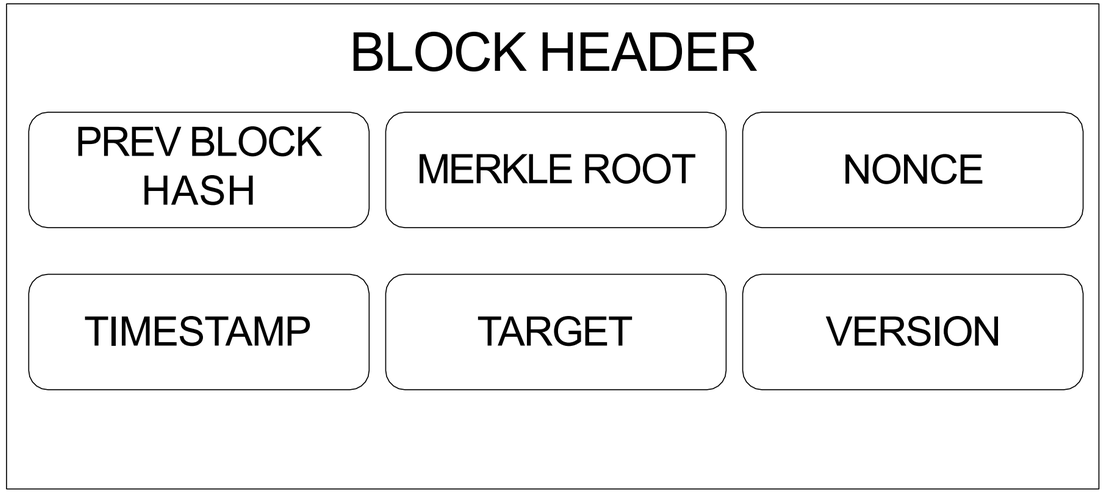


- Gather transactions into a block
  - Choose whichever transactions you want (most transaction fees)
- Get previous block hash and other necessary metadata

RECIPE FOR MINING

STEP 3: FIND A VALID NONCE

- Find the proof-of-work
- Expend computational power
- Incrementing header nonce first, then coinbase nonce as necessary to change puzzle




```
TARGET = (65535 << 208) / DIFFICULTY;
coinbase_nonce = 0;
while (1) {
      header = makeBlockHeader(transactions, coinbase_nonce);
      for (header_nonce = 0; header_nonce < (1 << 32); header_nonce++){
            if (SHA256(SHA256(makeBlock(header, header_nonce))) <
      TARGET)
                  break; //block found!
      }
      coinbase_nonce++;
}
```

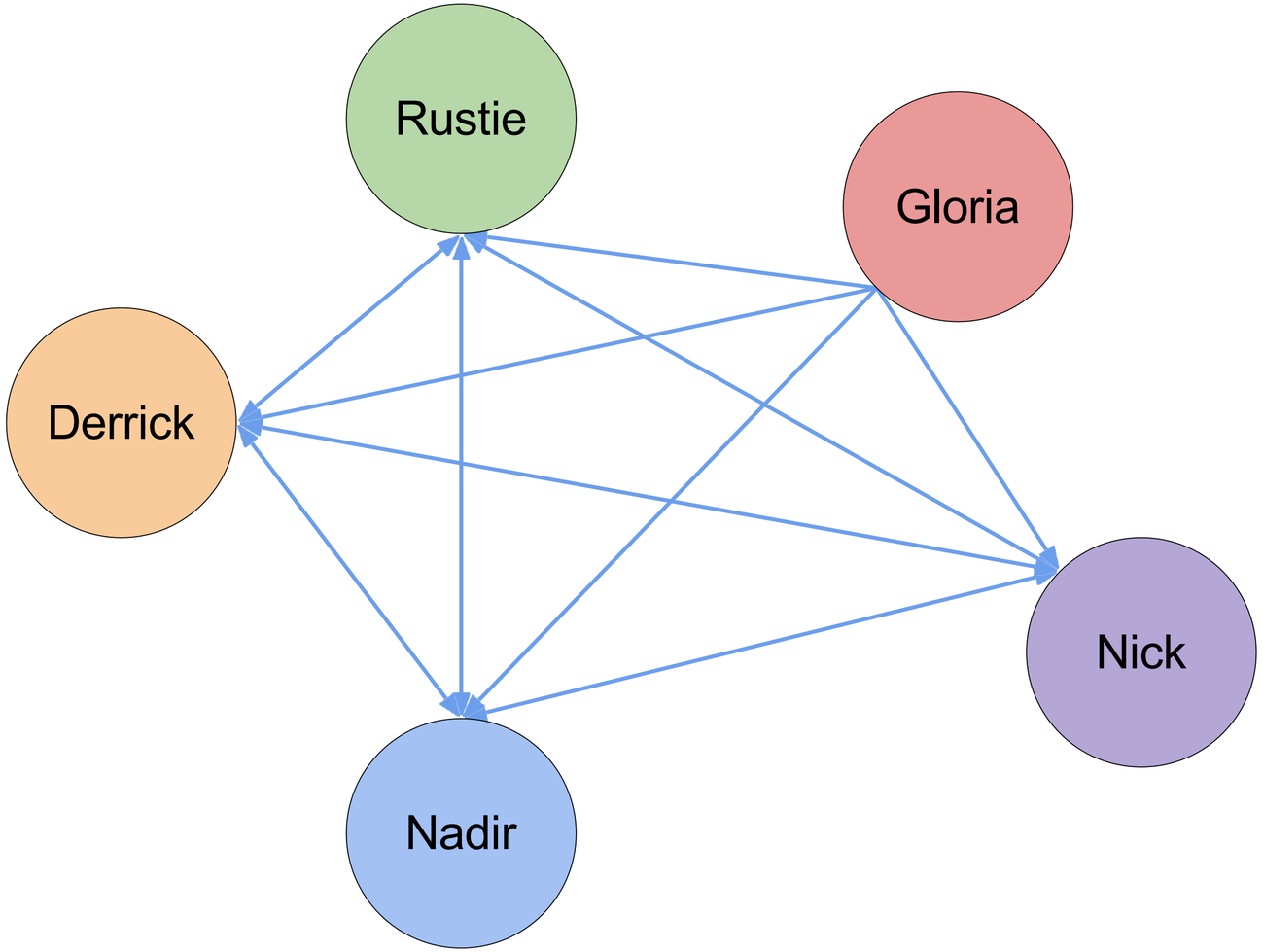
Figure 5.6 : CPU mining pseudocode.

Image source: Mastering Bitcoin

RECIPE FOR MINING

STEP 4: BROADCAST

- Broadcast to the rest of the network
- Others check transactions in the proposed block
  - Accept if it's the first valid block they see
- Propagate to everyone



RECIPE FOR MINING

STEP 5: PROFIT

Remember: Mining is a competition

- Block included in longest chain
  - Profit from block reward (coinbase transaction) and transaction fees
  - All transactions added to canon transaction history
- Not included in longest chain
  - Your block may not have been the first valid block seen by others
  - Start mining next block

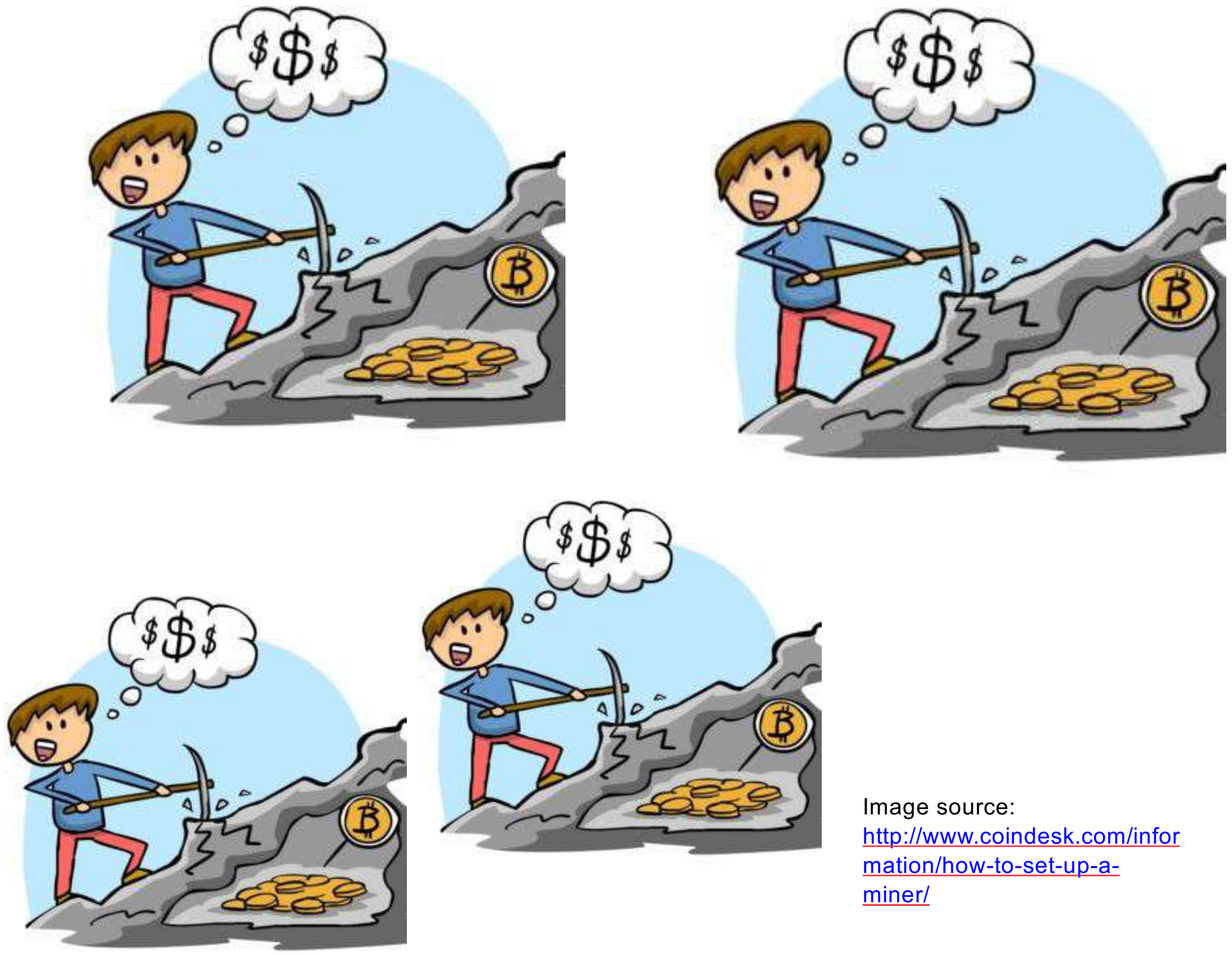


Image source: http://www.coindesk.com/information/how-to-set-up-a-miner/

Fork in the blockchain

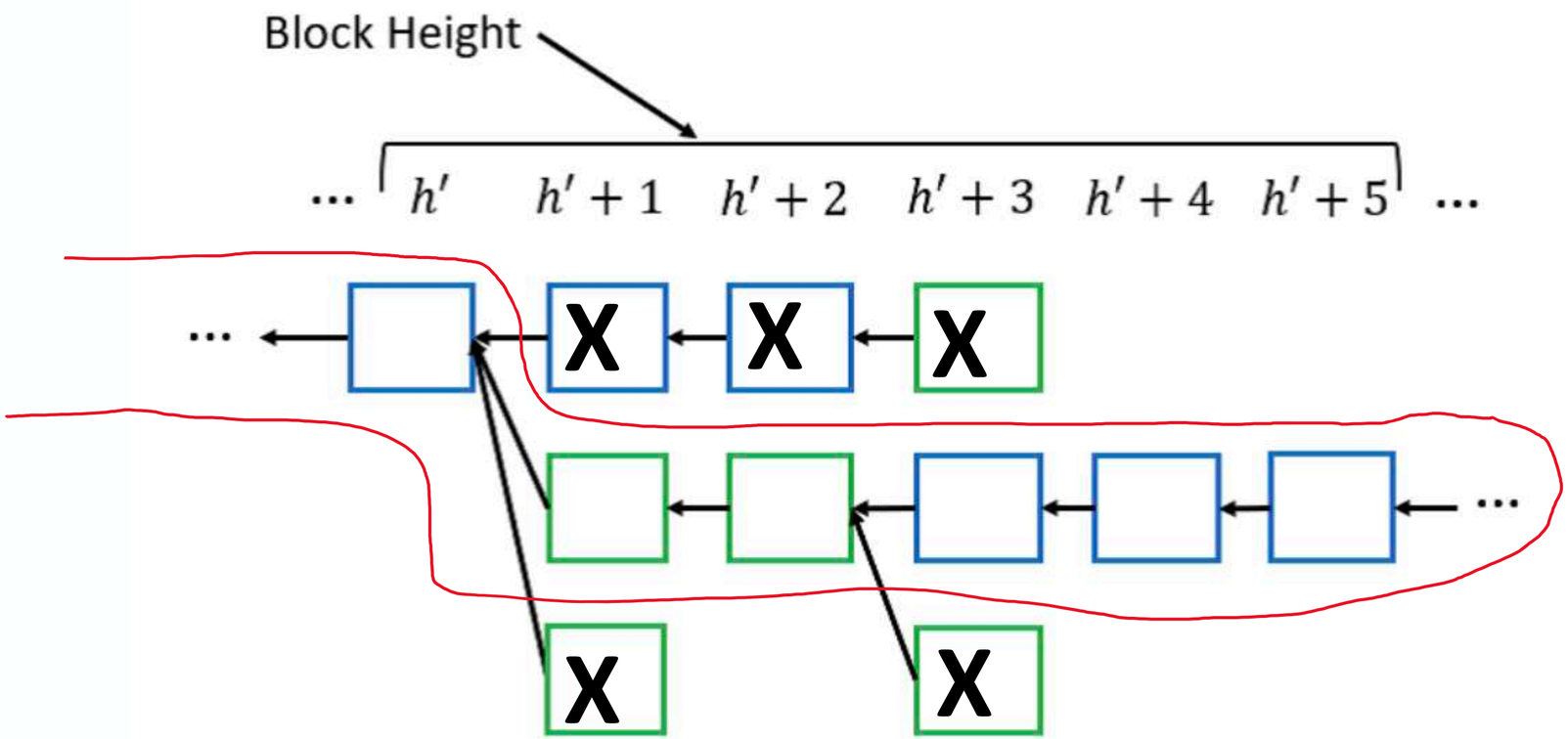


Image Courtesy of Seike et al. 2019 – “Fork Rate-based Analysis of the Longest Chain Growth Time Interval of a PoW Blockchain”

3.2 MINING INCENTIVES

MINING INCENTIVES

WHY DO WE DO THINGS?

PROFIT

MINING INCENTIVES

WHY DO WE DO THINGS?

BLOCKCHAIN AT BERKELEY

MINING INCENTIVES

WHAT IS PROFIT?

```
if revenue > cost:
 return "$$$$"
```

PROFIT = REVENUE - COST

MINING INCENTIVES

HOW TO PROFIT FROM MINING

```
MINING_REVENUE = BLOCK_REWARD + TX_FEES
MINING_COST = FIXED_COSTS + VARIABLE_COSTS

if MINING_REVENUE > MINING_COST:
 miner.get_profit()
```

Image source: https://profitbitcoin.com/

MINING INCENTIVES

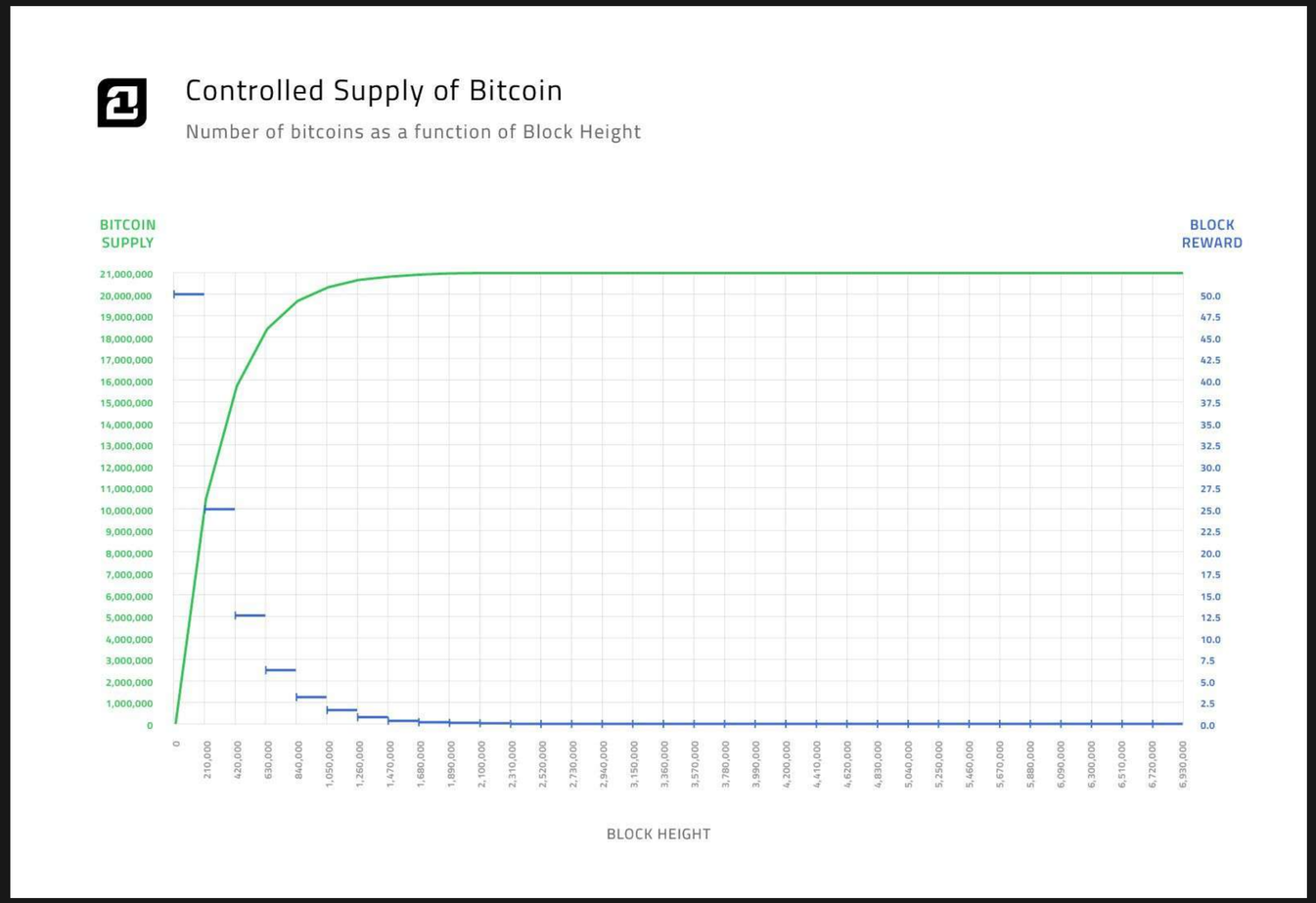
HOW TO PROFIT FROM MINING

```
MINING_REVENUE = BLOCK_REWARD + TX_FEES
MINING_COST = FIXED_COSTS + VARIABLE_COSTS

if MINING_REVENUE > MINING_COST:
 miner.get_profit()
```

Image source: https://profitbitcoin.com/

MINING INCENTIVES

BLOCK REWARD



- Miner receives BTC for every confirmed block
  - Currently 12.5 per block
- Miner includes special transaction to self
  - Incentive (profit!) for honest behavior
- Halves every 210,000 blocks
  - Finite # of BTC
- BTC supply cap: 21,000,000

MINING INCENTIVES (TO DELETE?)

BLOCK REWARD: RATIONALE

- Given:
  - Profit is primary motivator
  - Higher incentive for honesty ⇒ more secure network
  - Pseudonymous users ⇒ no way to effectively track (or punish) dishonest behavior
- Conclusion:
  - Reward the honest nodes!
  - Proof-of-Work ensures that miners are dedicated to the network (aka willing to pay money for electricity and hardware just to earn BTC)

MINING INCENTIVES

HOW TO PROFIT FROM MINING

```
MINING_REVENUE = BLOCK_REWARD + TX_FEES
MINING_COST = FIXED_COSTS + VARIABLE_COSTS

if MINING_REVENUE > MINING_COST:
 miner.get_profit()
```

Image source: https://profitbitcoin.com/

MINING INCENTIVES

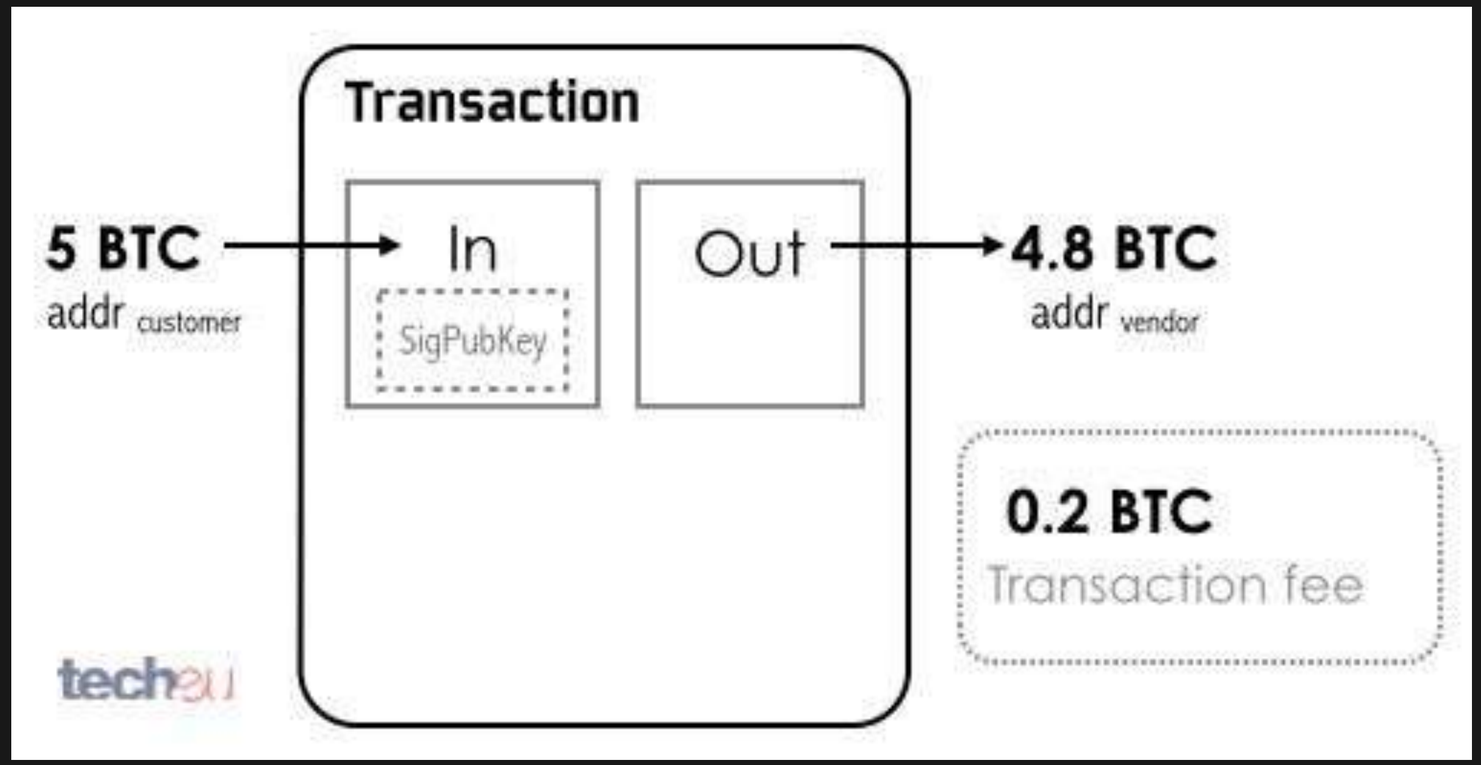
HOW TO PROFIT FROM MINING

```
MINING_REVENUE = BLOCK_REWARD + TX_FEES
MINING_COST = FIXED_COSTS + VARIABLE_COSTS

if MINING_REVENUE > MINING_COST:
 miner.get_profit()
```

Image source: https://profitbitcoin.com/

MINING INCENTIVES

TRANSACTION FEES



- Tx creator sets tx fee
  - Voluntary, but practically necessary...
- Extra income for miners on top of block reward (esp. as reward diminishes)
  - Higher transaction fee ⇒ faster confirmation time
- `TX_FEE = INPUT - OUTPUT`
- **When block reward becomes 0, TX fees will become primary source of revenue for miners**

MINING INCENTIVES

HOW TO PROFIT FROM MINING

```
MINING_REVENUE = BLOCK_REWARD + TX_FEES
MINING_COST = FIXED_COSTS + VARIABLE_COSTS

if MINING_REVENUE > MINING_COST:
 miner.get_profit()
```

Image source: https://profitbitcoin.com/

MINING INCENTIVES

HOW TO PROFIT FROM MINING

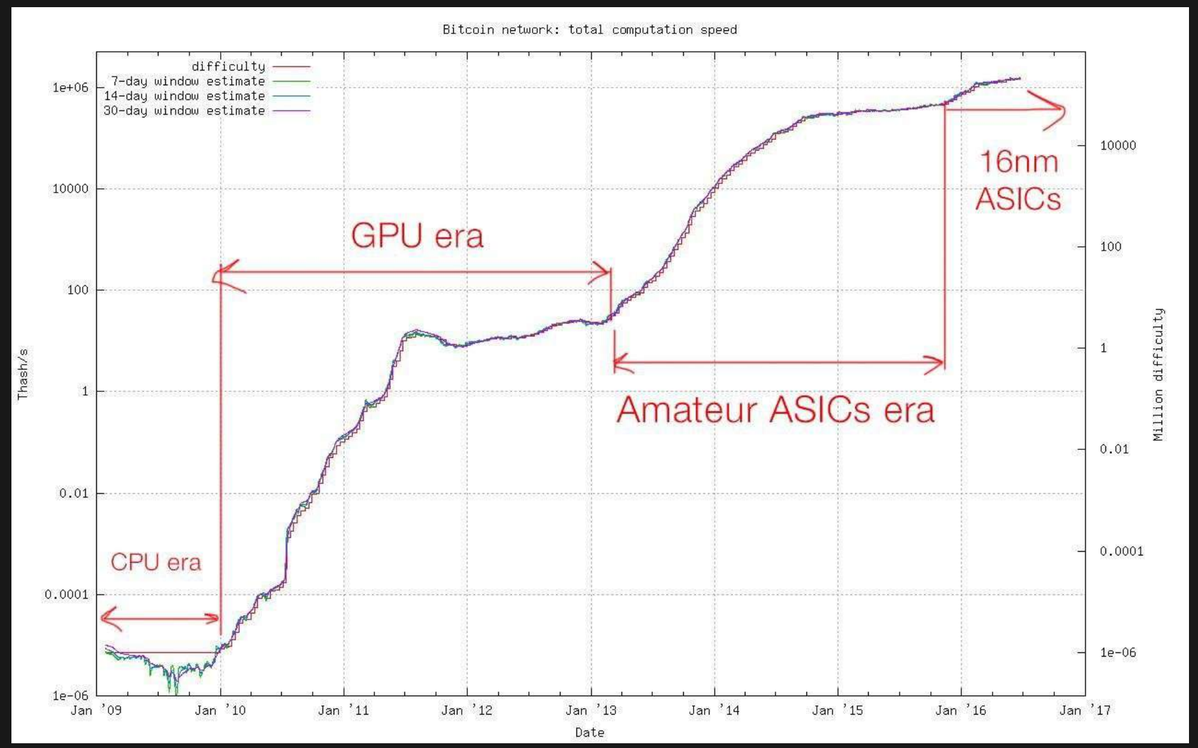
```
MINING_REVENUE = BLOCK_REWARD + TX_FEES
MINING_COST = FIXED_COSTS + VARIABLE_COSTS

if MINING_REVENUE > MINING_COST:
 miner.get_profit()
```

Image source: https://profitbitcoin.com/

MINING INCENTIVES

FIXED COST: HARDWARE COSTS

	hashes / second	time to block (years)
CPU	20 million	7,620,101
GPU	200 million	762,010
FPGA	1billion	152,357
ASIC	14 trillion	10.88



MINING INCENTIVES

FIXED COST: CPU MINING

	hashes / second	time to block (years)
CPU	20 million	7,620,101
GPU	200 million	762,010
FPGA	1billion	152,357
ASIC	14 trillion	10.88

```
TARGET = (65535 << 208) / DIFFICULTY;
coinbase_nonce = 0;
while (1) {
      header = makeBlockHeader(transactions, coinbase_nonce);
      for (header_nonce = 0; header_nonce < (1 << 32); header_nonce++){
            if (SHA256(SHA256(makeBlock(header, header_nonce))) <
      TARGET)
                  break; //block found!
      }
      coinbase_nonce++;
}
```

Figure 5.6 : CPU mining pseudocode.

(from Princeton Textbook, 5.2)

- Keep in mind that hardware costs are fixed, unlike everything else

MINING INCENTIVES

FIXED COST: GPU MINING

	hashes / second	time to block (years)
CPU	20 million	7,620,101
GPU	200 million	762,010
FPGA	1billion	152,357
ASIC	14 trillion	10.88

- Order of magnitude faster than CPUs
  - Larger consumption of energy and higher production of heat
- Most common in 2012
- Disadvantages:
  - Many components (floating point units) not applicable to mining
  - Not meant to be run in “farms” side by side

MINING INCENTIVES

FIXED COST: FPGA MINING

	hashes / second	time to block (years)
CPU	20 million	7,620,101
GPU	200 million	762,010
FPGA	1 billion	152,357
ASIC	14 trillion	10.88

- **F**ield **P**rogrammable **G**ate **A**rrays
  - Developing Bitcoin-specific hardware without losing all customizability
- Trade-off between dedicated SHA-256 and general purpose hardware
  - If Bitcoin fails, SHA-256 specific hardware is worthless
  - But if Bitcoin thrives, specialized hardware generates higher **PROFIT**!

MINING INCENTIVES

FIXED COST: ASIC MINING

	hashes / second	time to block (years)
CPU	20 million	7,620,101
GPU	200 million	762,010
FPGA	1billion	152,357
ASIC	14 trillion	10.88

- **A**pplication-**S**pecific **I**ntegrated **C**ircuit
  - Does nothing but SHA-256 -- but does it better than anything else
- Huge variety with various tradeoffs
  - Lower base cost vs lower electricity usage
  - Compact device vs higher hashrate
- Manufacturing ASICs takes large upfront capital, inducing production centralization
- Antminer S9 (14TH/s): $3000

MINING INCENTIVES

HOW TO PROFIT FROM MINING

```
MINING_REVENUE = BLOCK_REWARD + TX_FEES
MINING_COST = FIXED_COSTS + VARIABLE_COSTS

if MINING_REVENUE > MINING_COST:
 miner.get_profit()
```

Image source: https://profitbitcoin.com/

MINING INCENTIVES

HOW TO PROFIT FROM MINING

```
MINING_REVENUE = BLOCK_REWARD + TX_FEES
MINING_COST = FIXED_COSTS + VARIABLE_COSTS

if MINING_REVENUE > MINING_COST:
 miner.get_profit()
```

Image source: https://profitbitcoin.com/

MINING INCENTIVES

OPERATING COSTS

- Energy consumed in mining:
  - **Embodied energy,** to produce your hardware
  - **Electricity**, to power your hardware
  - **Cooling**, to maintain your hardware
- Infrastructure
  - Warehouses
  - Personnel
- All energy converted to heat -- is this not wasteful?
  - The “data furnace” approach: using mining equipment to generate heat
    - Unless a high percentage of the network stops mining during the heat, leading to miners dropping out for days on end, or even a whole summer!

MINING INCENTIVES

HOW TO PROFIT FROM MINING

```
MINING_REVENUE = BLOCK_REWARD + TX_FEES
MINING_COST = FIXED_COSTS + VARIABLE_COSTS

if MINING_REVENUE > MINING_COST:
 miner.get_profit()
```

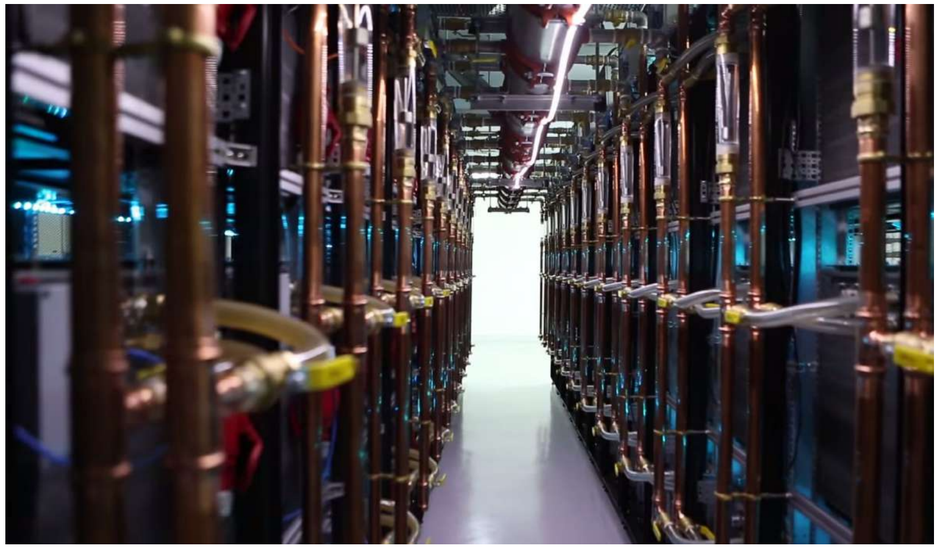
Image source: https://profitbitcoin.com/

4

REAL WORLD MINING

REAL WORLD MINING

CHINESE ASIC MINING FARM



Source: https://www.theregister.co.uk/2014/08/12/chinese_bitcoin_farms_from_scifi_to_scuzzy/

REAL WORLD MINING

ASICS

Source: https://sc01.alicdn.com/kf/HTB18YN_JFXXXXcgXFXXq6xXFXXXw/221223714/HTB18YN_JFXXXXcgXFXXq6xXFXXXw.jpg

Source: https://mybtcpool.com/product/datacenter-hosting/

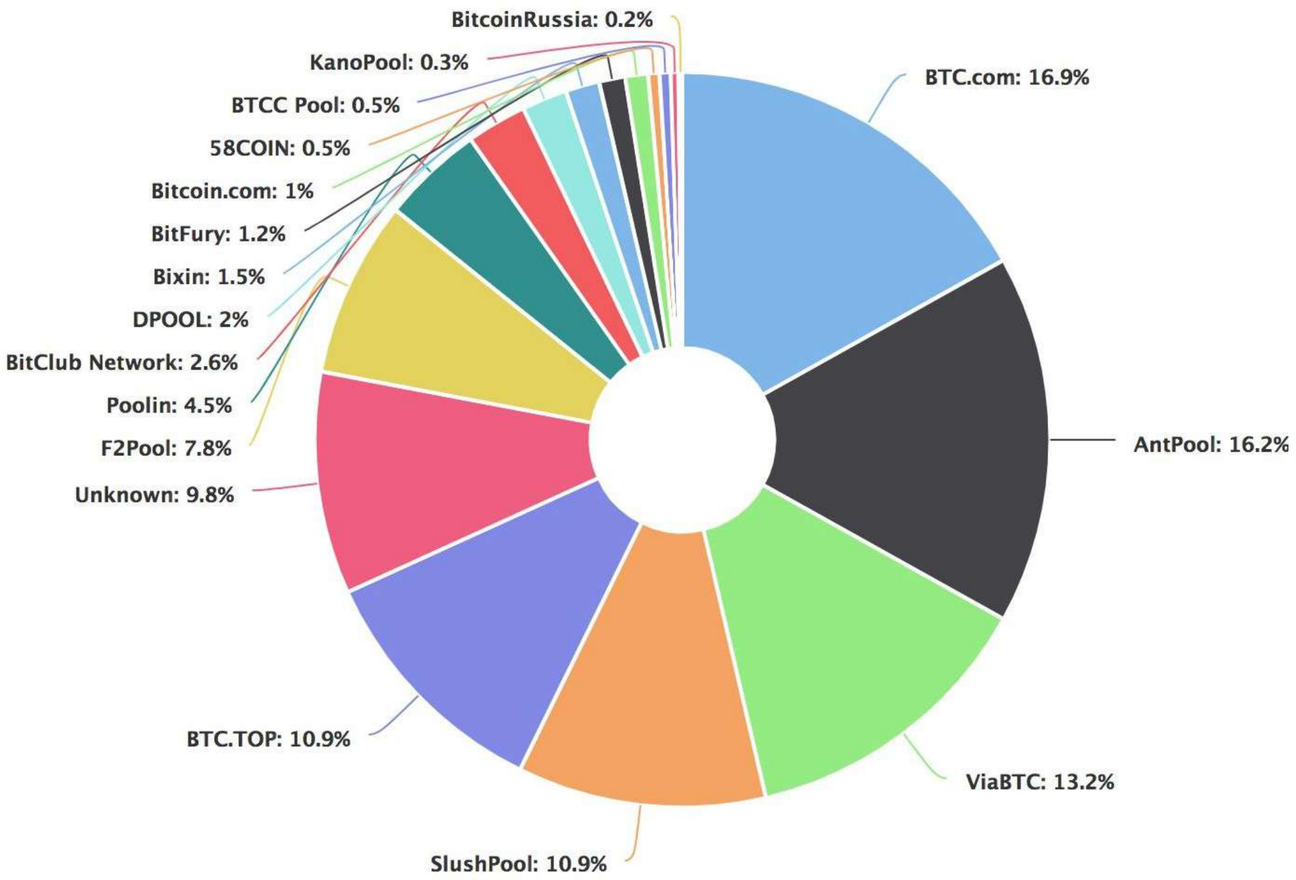
REAL WORLD MINING

MINING POOLS

Source: blockchain.info (9/26/18)

Mining pools allow individual miners to combine, or 'pool', their computational power together

- Reduces variance in mining rewards
- Run by **pool managers** or **pool operators**
- Pool manager usually takes a cut of the mining rewards



REAL WORLD MINING

MINING SHARES

Miners in a pool submit **shares** ('near-valid' blocks) to the pool manager

- Producing shares implies computational power being expended
- Pool operator pays for valid shares
  - Rewards distributed proportional to # of shares submitted
- Valid blocks are shares as well
  - Individual who finds valid block is not awarded any extra coins

FAQ: Why can't someone submit shares in a pool and keep the reward of the valid block for themselves?

- The valid block is based on the Merkle root given by the pool operator.
- Pool public key → Coinbase tx → Merkle Root

REAL WORLD MINING

MINING POOL SCHEMES

Pay-per-share

Pool pays out **at every share submitted.** By default will be proportional to work done by individuals

1. More beneficial for **miners**
2. Individual miners have no risk from reward variance
   a. Pool takes on the risk completely
3. Problem:???

Proportional

Pool pays out **when blocks are found,** proportional to the work individuals have submitted for this block

1. More beneficial for the **pool**
2. Individual miners still bear some risk in variance proportional to size of the pool
3. Lower risk for pool operators - only pay out when reward is found
   a. Individuals thus incentivized to submit valid blocks

Problem??

REAL WORLD MINING

MINING POOL SCHEMES

Pay-per-share

Pool pays out **at every share submitted.** By default will be proportional to work done by individuals

1. More beneficial for **miners**
2. Individual miners have no risk from reward variance
   a. Pool takes on the risk completely
3. Problem: No incentive for individuals to actually submit valid blocks
   a. Individuals are paid regardless

Proportional

Pool pays out **when blocks are found,** proportional to the work individuals have submitted for this block

1. More beneficial for the **pool**
2. Individual miners still bear some risk in variance proportional to size of the pool
3. Lower risk for pool operators - only pay out when reward is found
   a. Individuals thus incentivized to submit valid blocks

Risk for miners to never get paid

a. Not a problem if pool is sufficiently large

POOL REWARDS

OTHER REWARD SCHEMES

- **CPPSRB** - Capped Pay Per Share with Recent Backpay.
- **DGM** - Double Geometric Method. A hybrid between PPLNS and Geometric reward types that enables to operator to absorb some of the variance risk. Operator receives portion of payout on short rounds and returns it on longer rounds to normalize payments.
- **ESMPPS** - Equalized Shared Maximum Pay Per Share. Like SMPPS, but equalizes payments fairly among all those who are owed.
- **POT** - Pay On Target. A high variance PPS variant that pays on the difficulty of work returned to pool rather than the difficulty of work served by pool
- **PPLNS** - Pay Per Last N Shares. Similar to proportional, but instead of looking at the number of shares in the round, instead looks at the last N shares, regardless of round boundaries.
- **PPLNSG** - Pay Per Last N Groups (or shifts). Similar to PPLNS, but shares are grouped into "shifts" which are paid as a whole.
- **RSMPPS** - Recent Shared Maximum Pay Per Share. Like SMPPS, but system aims to prioritize the most recent miners first.
- **Score** - Score based system: a proportional reward, but weighed by time submitted. Each submitted share is worth more in the function of time *t* since start of current round. For each share score is updated by: score += exp(t/C). This makes later shares worth much more than earlier shares, thus the miner's score quickly diminishes when they stop mining on the pool. Rewards are calculated proportionally to scores (and not to shares). (at slush's pool C=300 seconds, and every hour scores are normalized)
- **SMPPS** - Shared Maximum Pay Per Share. Like Pay Per Share, but never pays more than the pool earns.
- **FPPS** - Full Pay Per Share. Similar to PPS, but not only divide regular block reward (12.5 BTC for now) but also some of the transaction fees. Calculate a standard transaction fee within a certain period and distribute it to miners according to their hash power contributions in the pool. It will increase the miners' earnings by sharing some of the transaction fees.

Source: https://en.bitcoin.it/wiki/Comparison_of_mining_pools

REAL WORLD MINING

MINING POOLS

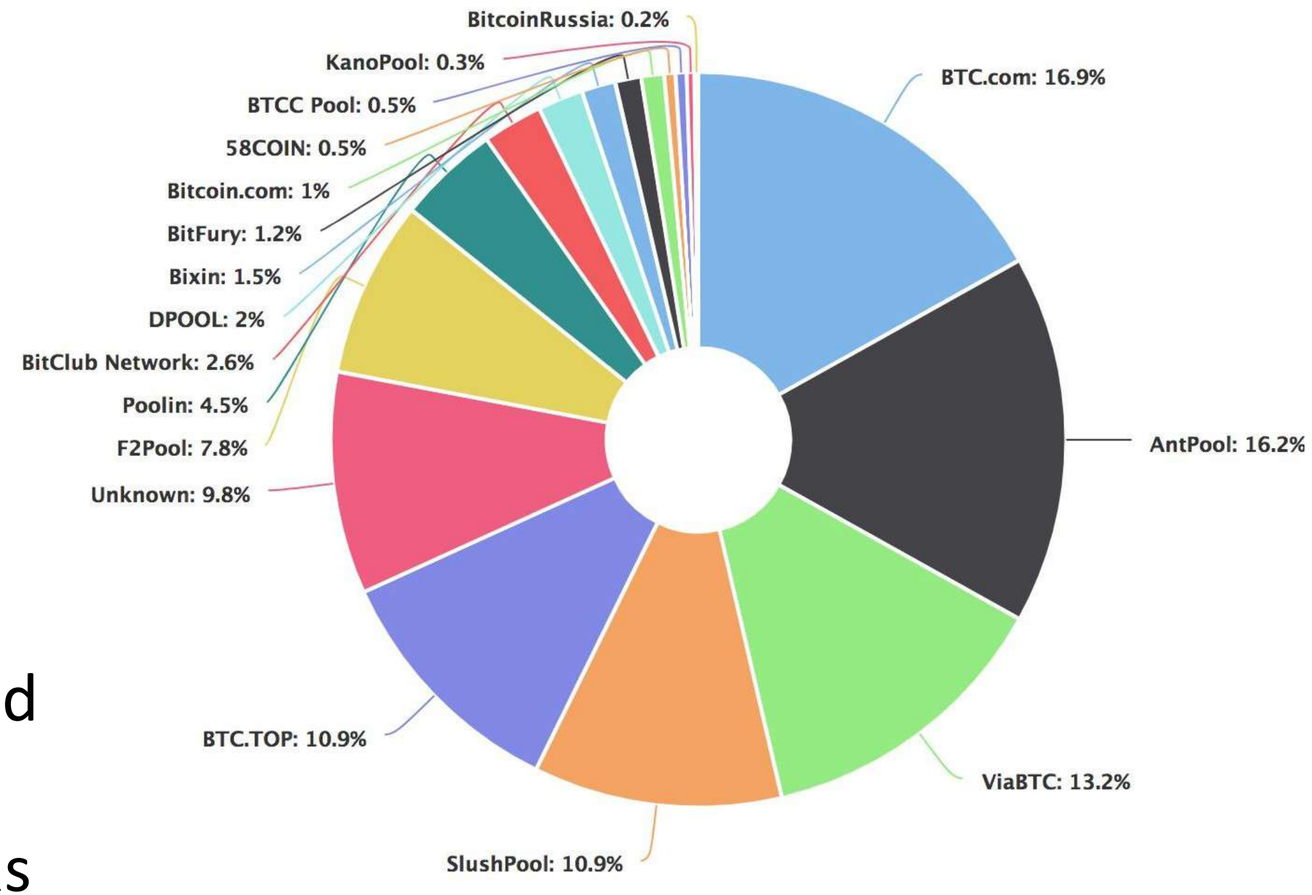
Source: blockchain.info (9/26/18)

Pros

- Allows individual miners to participate
- Easy to upgrade software changes

Cons

- Pool manager must be trusted
- Centralized
- Enables a multitude of attacks



REAL WORLD MINING

MINING POOLS

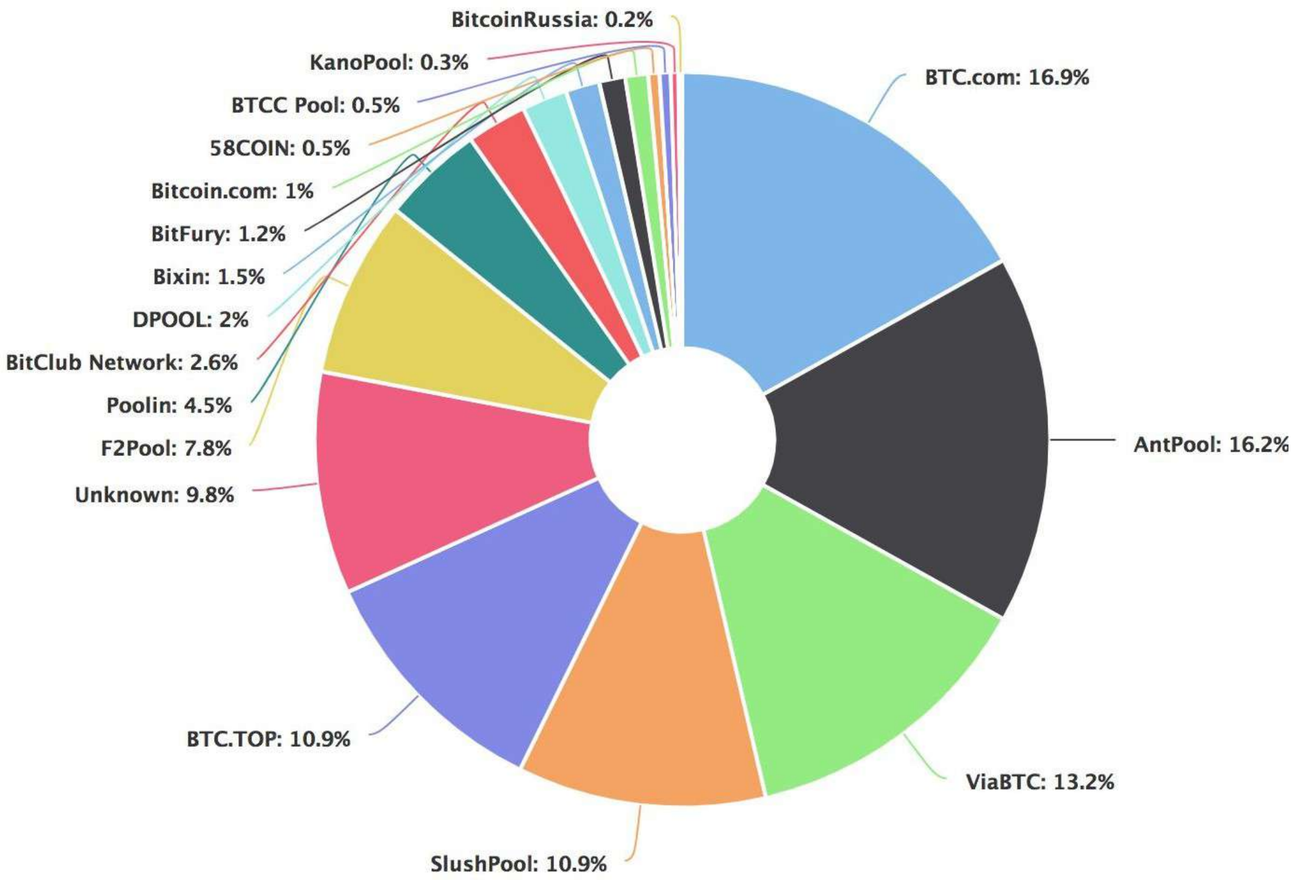
Source: blockchain.info (9/26/18)

Community exhibits backlash against large mining pools

- Ex: GHash.io in 2014

Single entity might be be participating in multiple pools

- Called "**Laundering hashes**"
- Actual concentration of control over mining hardware **is unknown**



REAL WORLD MINING

MINING POOLS

Quick facts

- Today's (9/24/18) network hashrate: **60,089,527 TH/s**
- Mining Reward /yr = (1yr /10 mins) *12.5 = **657k BTC /yr**
- Assume constant price of $10,000

Suppose you want to start mining today.

- Antminer S9: Costs $3000, **14 TH/s**
- **% of network hashrate** = (14 TH/s) / (60,089,527 TH/s) = **0.000000232985**

Expected Annual Reward

- 0.000000232985 *657k /yr
  ≈ 0.1530716 BTC /yr ≈ **$1530.72 / yr**

Solo mining

- 1block mined per 2,426,566 blocks
  ⇒ 12.5 BTC every 16,851 days
  ⇒ **$125,000 once every 46.2 years**

Mining with mining pool

- Assume pool has ⅙ network hashrate
  - Pool finds every 6th block ≈ 1per hr
- $1530.72 /yr /8760 hrs/yr
  ≈ **$0.17 every hour**

Paradox:

- The more secure Bitcoin gets, the greater the appeal for mining pools

5 CHANGING BITCOIN

DECENTRALIZING MINING

THE PROBLEM

- In practice, “One CPU One Vote” isn’t real
  - ASICs
  - Mining pools
  - Mining farms

Source: https://www.theregister.co.uk/2014/08/12/chinese_bitcoin_farms_from_scifi_to_scuzzy/

Source: https://sc01.alicdn.com/kf/HTB18YN_JFXXXXcgXFXXq6xXFXXXw/221223714/HTB18YN_JFXXXXcgXFXXq6xXFXXXw.jpg

Source: https://www.buybitcoinworldwide.com/wp-content/themes/kepler/img/miners/21.jpg

DECENTRALIZING MINING

PUZZLE REQUIREMENTS REVIEW

- A refresher on puzzle requirements:
  - Quick to verify
  - Adjustable difficulty
  - Computationally difficult
  - Solving rate proportional to computational power
  - “Progress free”- Avalanche Effect
  - Pseudorandomly generated
- Bitcoin’s puzzle is a “partial hash-preimage puzzle”
  - Doesn’t matter what follows the prerequisite number of zeros

DECENTRALIZING MINING

ASIC-RESISTANCE

Memory-hard: requires large amount of memory instead of computational power

Memory-bound: memory bottlenecks computation time

- Memory-hard puzzles viably deter ASICs:
  - ASICs are optimized to execute a specific algorithm
    - Useless optimization if memory is the limiting agent

DECENTRALIZING MINING

SCRYPT

Scrypt (“ess crypt”): a hash function.

The mining puzzle is the same partial hash-preimage puzzle.

Design considerations:

- Used for hashing passwords
- Hard to brute-force

Used by Litecoin and Dogecoin

AUTHOR: MAX FANG

DECENTRALIZING MINING

SCRYPT

Two main steps:

1. Fill buffer w/ interdependent data
2. Access data in pseudorandom way

Without using memory, `V[j]`, a previously computed value, must be computed on the fly.

Drawbacks:

1. Requires equal amount of memory to verify
2. ASIC developed; not resistant!

Figure 8.1: Scrypt pseudocode

```
def scrypt(N, seed):
   V = [0] * N  // initialize memory buffer of length N

  // Fill up memory buffer with pseudorandom data
  V[0] = seed
  for i = 1 to N:
     V[i] = SHA-256(V[i-1])

  // Access memory buffer in a pseudorandom order
  X = SHA-256(V[N-1])
  for i = 1 to N:
     j = X % N  // Choose a random index based on X
     X = SHA-256(X ^ V[j]) // Update X based on this index

  return X
```

DECENTRALIZING MINING

ASIC-RESISTANCE

- **x11** or **x13**: Chain 11 or 13 different hash functions together respectively
  - Used by DASH
  - Significantly harder to design ASIC
    - ...but not impossible, mind you
- Periodically switching mining puzzle
  - Going from SHA-1 to SHA-3 to Scrypt for 6 months each
  - Not implemented

PinIdea ASIC X11 Miner DR-1 Hashrate 500MH/s @320w Weighs 4.5kg

Discussion in 'Hardware Discussions (ASIC / GPU / CPU)' started by soleo, Feb 22, 2016.

Page 1 of 11 1 2 3 4 5 6 → 11 Next >

soleo
Member

Joined:	Mar 5, 2015
Messages:	51
Likes Received:	65
Trophy Points:	58

Who are we?
We are a group of engineers who work in four different cities (Shanghai, Wuxi, Shenzhen, Chicago) across U.S.A and China. In the past two years, we've been working on developing ASIC for X11 coins. And in the past few months, we have some breakthroughs on miners. Obviously, we have huge confidence on Dash which leads us to develop ASIC miner, even though the market isn't mature back then.

Why announcing the news now?
A few months ago, we announced we have an explorer version of X11 Miner. And we made a small batch of miners test the water of the market but we didn't deliver. The whole teams were split since then. Hearing about recent development on ASIC miner in Dash community, I contacted my past teammate to see how's everything going with them. It turned out that one of our engineers who is working with another vendor had a breakthrough, and performance is good enough for us to announce the news. PinIdea will be the only distributor for the Shooter Chip X11 Miners.

Mike Hearn, Bitcoin Core developer: “There’s really no such thing as an ASIC-resistant algorithm.”

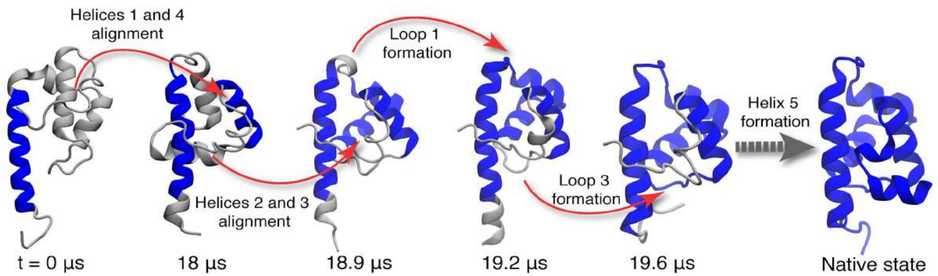
DECENTRALIZING MINING

ASIC-RESISTANCE

- Pros of ASIC-resistance:
  - ASICs dominate the network, suppressing regular people
  - Increase in democracy and decrease in centralization
- Cons:
  - ASICs can only solve the puzzle, nothing more
  - Crash in exchange rate ⇒ useless electricity-gobbling hardware

PROOF-OF-USEFUL-WORK

NOT A PUZZLING CONCEPT



- "Repurpose" computing power
- Examples:
  - Searching for large primes
  - Finding aliens
  - Simulating proteins at the atomic level
  - Generating predictive climate models

Project	Founded	Goal	Impact
Great Internet Mersenne Prime Search	1996	Finding large Mersenne primes	Found the new "largest prime number" twelve straight times, including $2^{57885161} - 1$
distributed.net	1997	Cryptographic brute-force demos	First successful public brute-force of a 64-bit cryptographic key
SETI@home	1999	Identifying signs of extraterrestrial life	Largest project to date with over 5 million participants

PROOF-OF-USEFUL-WORK

DOES IT WORK?

- Most distributed computing problems are unsuitable for proof-of-work
  - Fixed amount of data
    - Missing an <u>inexhaustible puzzle space</u>
  - Potential solutions not equally likely
    - Missing an <u>equiprobable solution space</u>
  - Cannot rely on central entity to delegate tasks
    - Missing <u>decentralized algorithmically generated problem</u>
- In summary, **Proof-of-Useful-Work does not work**

CONSENSUS UPDATES

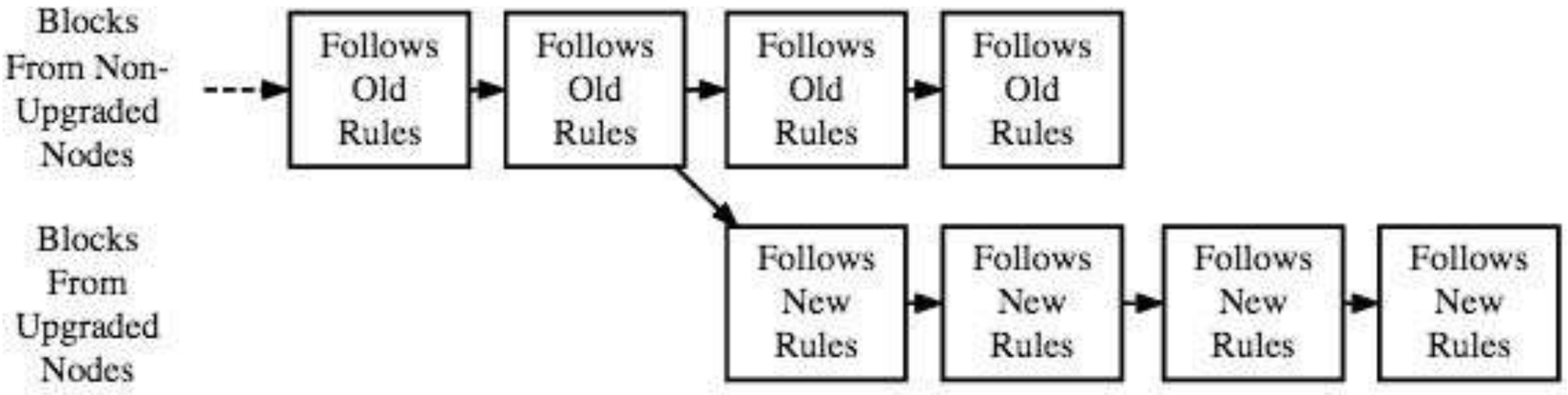
BITCOIN CORE

- **Bitcoin Core:**
  - The team of developers in charge of the Bitcoin GitHub repo
  - The software designed by these developers used by full Bitcoin nodes

CONSENSUS UPDATES

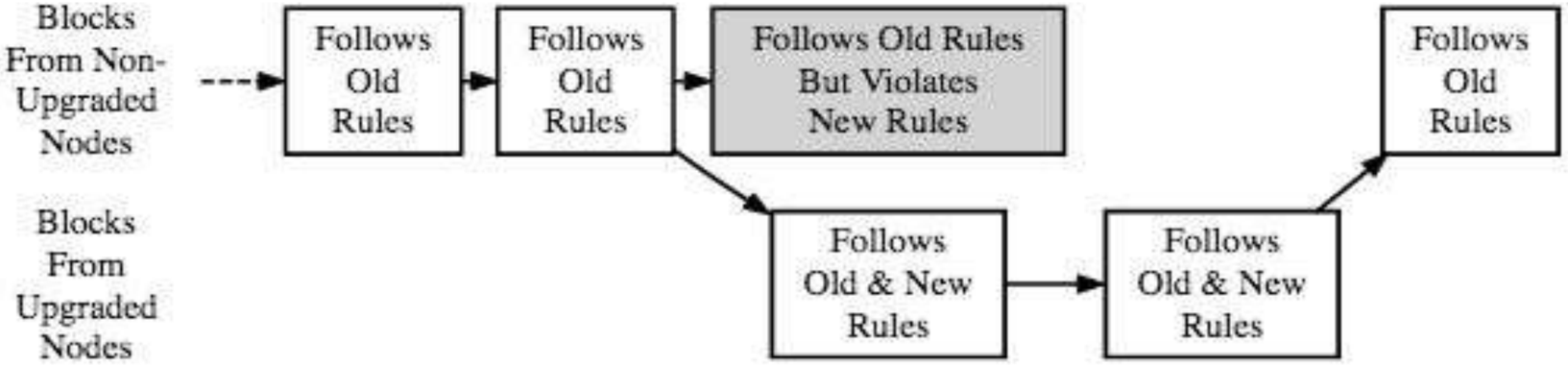
FORKS

Hard Fork



A Hard Fork: Non-Upgraded Nodes Reject The New Rules, Diverging The Chain

Soft Fork



A Soft Fork: Blocks Violating New Rules Are Made Stale By The Upgraded Mining Majority

Source: Bitcoin.org Developer Guide

CONSENSUS UPDATES

FORKS

Hard Fork

1.A block following the new consensus rules is accepted by upgraded nodes but rejected by non-upgraded nodes. For example, a new transaction feature is used within a block: upgraded nodes understand the feature and accept it, but non-upgraded nodes reject it because it violates the old rules. This creates permanently divergent chains.

Soft Fork

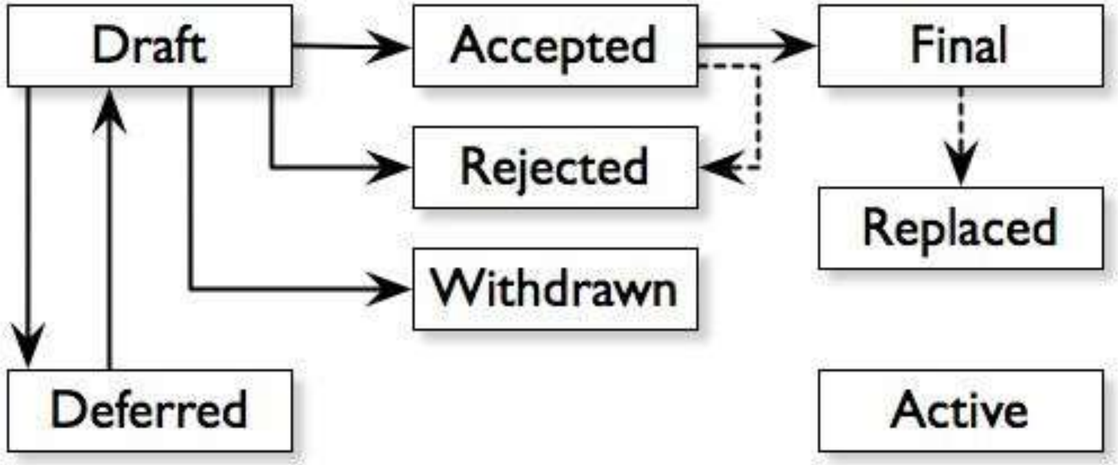
2.A block violating the new consensus rules is rejected by upgraded nodes but accepted by non-upgraded nodes. For example, an abusive transaction feature is used within a block: upgraded nodes reject it because it violates the new rules, but non-upgraded nodes accept it because it follows the old rules.

Source: Bitcoin.org Developer Guide

CONSENSUS UPDATES

BITCOIN IMPROVEMENT PROPOSAL (BIP)

- **BIP:** Bitcoin Improvement Proposal
  - Three types:
    - Standards Track BIPs
    - Informational BIPs
    - Process BIPs
- First BIP proposed by Amir Taaki on 2011-08-19
- Signal support for a BIP by including reference in block when mining



Source: https://en.bitcoin.it/wiki/Bitcoin_Improvement_Proposals

QUESTIONS?

Next class - High Level Overview

Homework: Due in 2 weeks, by Monday. Check Piazza, Submit on Google Forms.

Required Readings:

- Ethereum Whitepaper up to "Applications" https://github.com/ethereum/wiki/wiki/White-Paper

References

Slides mainly adopted from

- Blockchain @ Berkeley : https://blockchain.berkeley.edu/
- Blockchain @ Princeton : http://bitcoinbook.cs.princeton.edu/

ETHEREUM AND SMART CONTRACTS: ENABLING A DECENTRALIZED FUTURE

Bitcoin P2P e-cash paper

Satoshi Nakamoto satoshi at vistomail.com
Fri Oct 31 14:10:00 EDT 2008

- Previous message: Fw: SHA-3 lounge
- **Messages sorted by:** [date] [thread] [subject] [author]

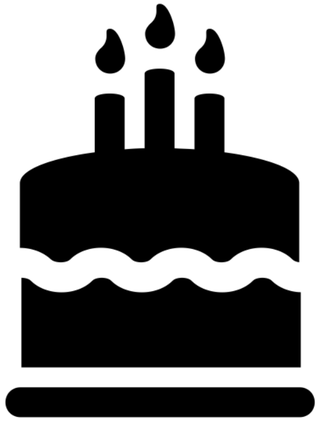
I've been working on a new electronic cash system that's fully peer-to-peer, with no trusted third party.

The paper is available at:
http://www.bitcoin.org/bitcoin.pdf

The main properties:
 Double-spending is prevented with a peer-to-peer network.
 No mint or other trusted parties.
 Participants can be anonymous.
 New coins are made from Hashcash style proof-of-work.
 The proof-of-work for new coin generation also powers the
    network to prevent double-spending.

Bitcoin: A Peer-to-Peer Electronic Cash System

Abstract. A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without the burdens of going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network. The network timestamps transactions by hashing them into an ongoing chain of hash-based proof-of-work, forming a record that cannot be changed without redoing the proof-of-work. The longest chain not only serves as proof of the sequence of events witnessed, but proof that it came from the largest pool of CPU power. As long as honest nodes control the most CPU power on the network, they can generate the longest chain and outpace any attackers. The network itself requires



Oct 31, 2008 – Birthday of Bitcoin

11th Birthday

Dün akşam saatlerinde gerçekleşen 1000 dolarlık artışın ardından, gecenin ilerleyen saatlerinde Bitcoin’de 1500 dolarlık bir artış daha meydana geldi.

Saatler içerisinde %35 artan Bitcoin, 10 bin dolar sınırını aştı.

Sabah saatleriyle birlikte inişe geçen Bitcoin şu an 9390 dolar seviyesinden işlem görüyor.

+ %35 increase in one day!!!

LECTURE OVERVIEW

1. SMART CONTRACTS
2. ETHEREUM
3. EVM (ETHEREUM VIRTUAL MACHINE)
4. USE CASES

1 SMART CONTRACTS

BITCOIN REVIEW

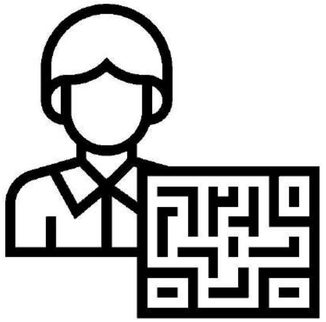
PROPERTIES OF BITCOIN

… but first, a question:

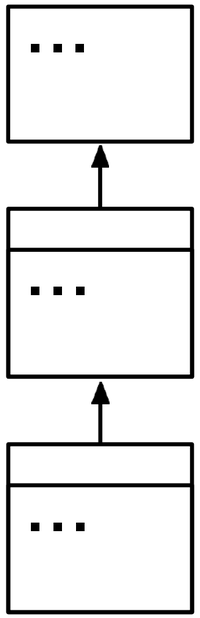
What makes Bitcoin so special?

A DISTRIBUTED NETWORK

BITCOIN'S BARE BONES



cryptographic identities



blockchain

trustless consensus

A DISTRIBUTED NETWORK

TRANSFERRABLE BENEFITS OF BITCOIN

- Pseudonymous, cryptographic identities allow for accountability
- Democratic decisions made through consensus protocol that **doesn't require trust**
- Immutable ledger of truth
- Uncensorable, cannot be controlled by any one party
- Distributed: no central point of failure

SMART CONTRACTS

CONTRACTS

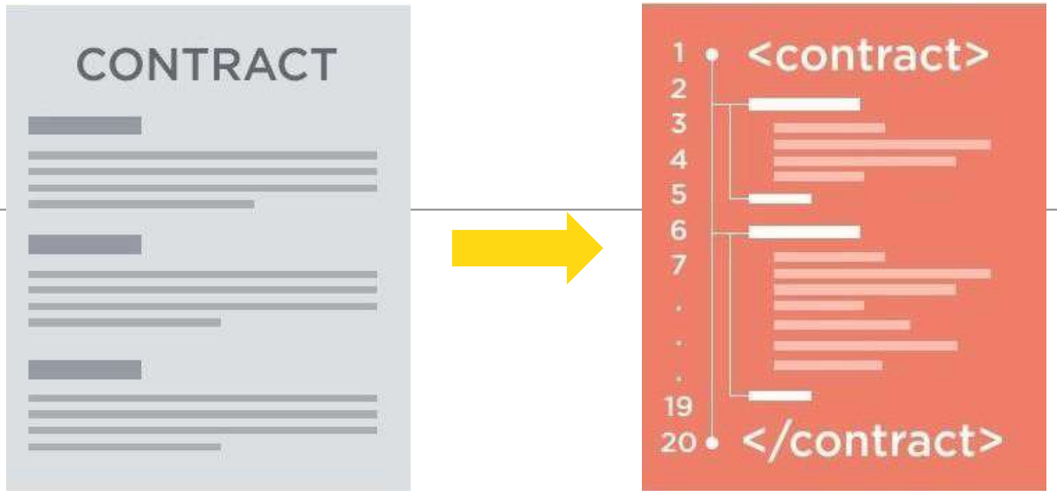
con·tract

(noun) /ˈkäntrakt/

1. a written or spoken agreement … that is intended to be enforceable by law.

SMART CONTRACTS

CONTRACTS



smart con·tract

(noun) /smärt ˈkäntrakt/

1. code that **facilitates**, **verifies**, or **enforces** the negotiation or execution of a digital contract.
   a. **Trusted entity** must run this code

2 ETHEREUM

ethereum
BLOCKCHAIN APP PLATFORM

WHAT IS ETHEREUM?

HIGH-LEVEL OVERVIEW

- Ethereum is a **decentralized** platform designed to run **smart contracts**
  - Like a distributed computer to execute code
  - **Distributed state machine - transactions change global state**
    - transactions == state transaction function
- Ethereum has a native asset called **ether**
  - it is a cryptocurrency

WHAT IS ETHEREUM?

WHO WOULD WIN?

Bitcoin

- First successful cryptocurrency
- Trustless
- Immutable
- Uncensorable
- Pseudonymous
- No central point of failure
- One-CPU-One-Vote

1turing-complete boi

WHAT IS ETHEREUM?

COMPARISON WITH BITCOIN

Bitcoin	Ethereum

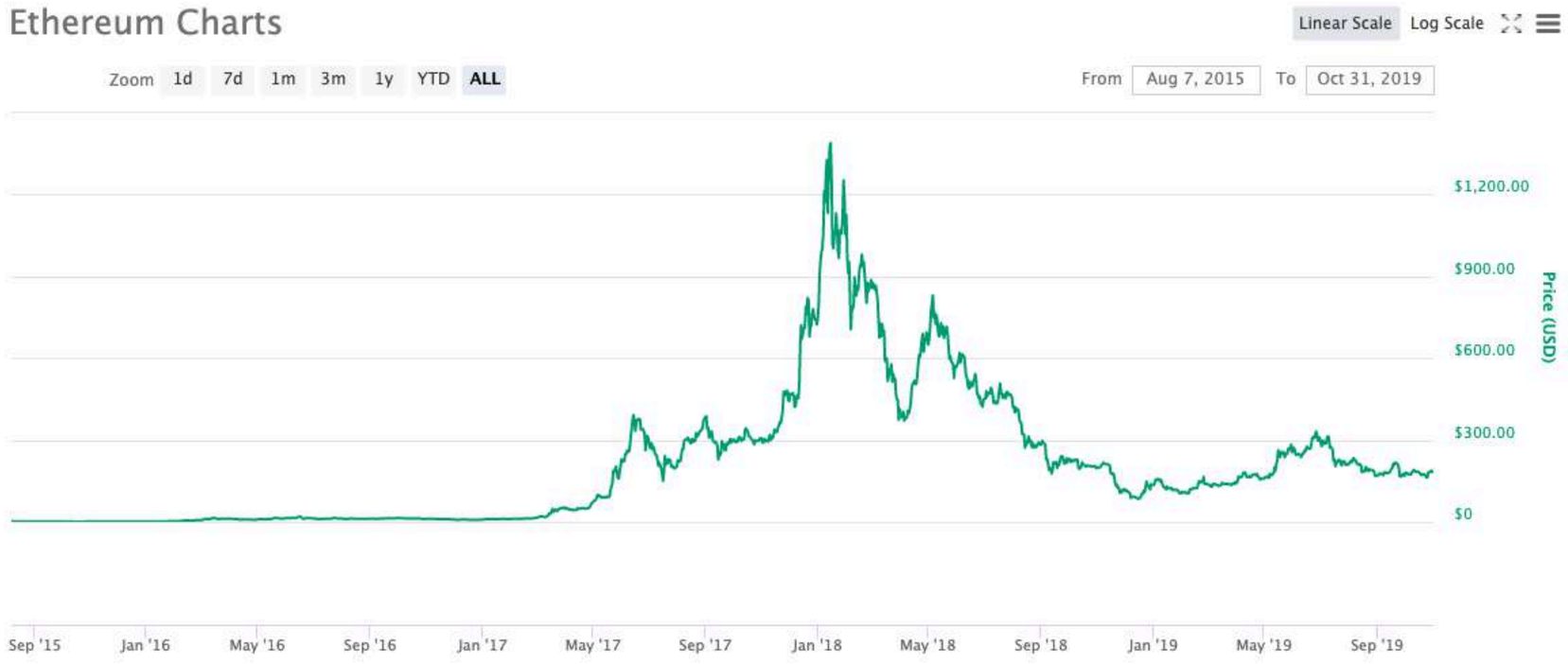
• The "Gold Standard" of blockchains
• Asset: bitcoins
  ○ Primary purpose of the Bitcoin blockchain
• Simple and robust
• Stack-based, primitive scripting language, not Turing-complete
• UTXO-based
• Will likely remain Proof-of-Work
	• Smart Contract Blockchain Platform
	• Asset: ether
	  1. Fund computation
	  2. Align incentives
	• Complex and feature-rich
	• Turing-complete scripting language
	• Account-based
	• Uses a blend of PoW and PoS

WHAT IS ETHEREUM?

COMPARISON WITH BITCOIN

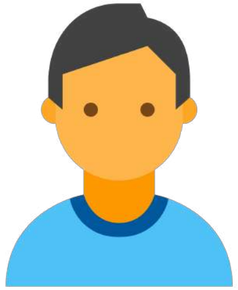
Misc. Implementation Details

- Block creation time: ~15 sec vs ~10 min
- Proof-of-Work: Ethash (currently ASIC resistant) vs SHA-256
- Exchange Rate: $183,99 (2019-10-30)



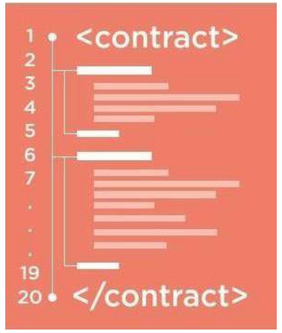
ETHEREUM ACCOUNTS

ACCOUNT TYPES



Externally Owned Accounts

- Owned by some external entity (person, corporation, etc.)
- Can send transactions to transfer ether or trigger contract code
- Contains:
  - Address
  - Ether Balance



Contract Accounts

- “Owned” by contract
- Code execution triggered by transactions or function calls (msg)
- Contains:
  - Address
  - Associated contract code
  - Persistent storage

"CLICKER" QUESTION:

DESIGN CHOICES

What are strengths of the UTXO Model?

Select all that apply.

A) Verifying transaction validity and checking double-spending

B) Space efficiency

C) Looking up account balances quickly

D) Ease in programming

"CLICKER" QUESTION

DESIGN CHOICES

What are weaknesses of the UTXO Model?

Select all that apply.

A) Verifying transaction validity and checking double-spending

B) Space efficiency

C) Looking up account balances quickly

D) Ease in programming

ETHEREUM ACCOUNTS

ACCOUNTS VS UTXO MODEL

Easy to make transactions and prevent double spending

Bitcoin:

Bob owns private keys to set of UTXOs

5 BTC ⇒ Bob

3 BTC ⇒ Bob

2 BTC ⇒ Bob

Ethereum:

Alice owns private keys to an account

address: “0xfa38b...”

balance: 10 ETH

code: c := a + b

ETHEREUM ACCOUNTS

ACCOUNTS RATIONALE

Easy to make transactions and prevent double spending

Bitcoin:

Bob owns private keys to set of UTXOs

5 BTC ⇒ Bob

3 BTC ⇒ Bob

2 BTC ⇒ Bob

Ethereum:

Alice owns private keys to an account

address: “0xfa38b...”

balance: 10 ETH

code: c := a + b

Space-efficient to update balances instead of storing UTXOs

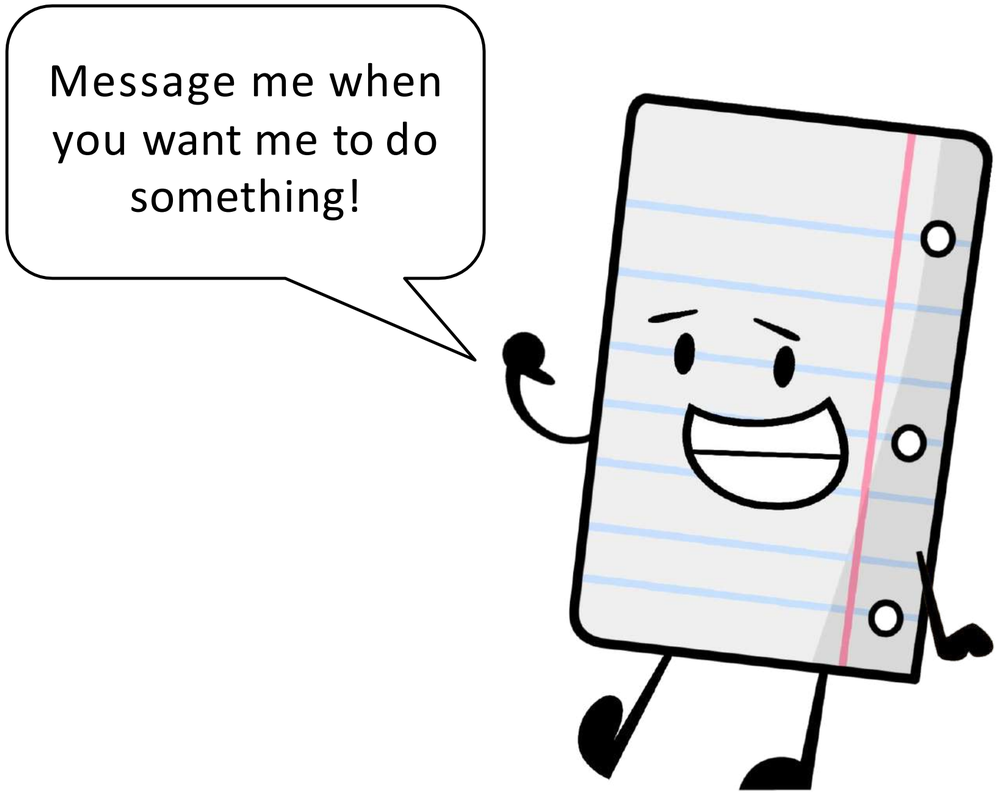
Easier to look up balance and transfer between accounts when programming

ETHEREUM SMART CONTRACTS

CONTROL

Smart Contracts in Ethereum are like autonomous agents that live inside of the Ethereum network

- React to external world when "poked" by transactions (which call specific functions)
- Have direct control over:
  - **internal ether balance**
  - **internal contract state**



ETHEREUM SMART CONTRACTS

SMART CONTRACTS IN ETHEREUM

Ethereum Contracts generally serve four purposes:

- **Store and maintain data**
  - Data represents something useful to users or other contracts
  - Ex: a token currency or organization's membership
- **Manage contract or relationship between untrusting users**
  - Ex: financial contracts, escrow, insurance
- **Provide functions to other contracts**
  - Serving as a software library
- **Complex Authentication**
  - Ex: M-of-N multisignature access

ETHEREUM SMART CONTRACTS

SAMPLE BETTING CONTRACT

```
contract Betting {
     address public owner;
     address public gamblerA, gamblerB, oracle;
     uint[] outcomes;
     struct Bet {
         uint outcome;  uint amount;       /* Defines a Bet */
         bool initialized;
     }

     mapping (address => Bet) bets;
     mapping (address => uint) winnings; /* Keep track of every gambler's bet */
     ...                                 /* Keep track of every player's winnings */

     function makeBet(uint _outcome) payable returns (bool) { … }
     function makeDecision(uint _outcome) oracleOnly() { … }
     function withdraw(uint withdrawAmount) returns (uint remainingBal) { … }

}
```

RECIPE FOR MINING: ETHEREUM

SIMILAR TO BITCOIN

Image source: http://www.coindesk.com/information/how-to-set-up-a-miner/

A full-fledged Ethereum miner must:

1. **Download** the entire Ethereum blockchain
2. **Verify** incoming transactions and **Run** Smart Contract code invoked by transactions
2. **Create** a block
3. **Find** a valid nonce
4. **Broadcast** your block
5. **Profit**!

THE DISTRIBUTED COMPUTER

DISTRIBUTED VERIFICATION &CONSENSUS

- Ethereum is a "distributed computer": **every node** executes Ethereum smart contracts, then come to consensus on the new network state
- Ethereum's distributed consensus protocol is **Proof-of-Work**
  - Miners competitively create blocks by executing code and searching for solution the the mining puzzle
- Network consensus removes the need for Trusted Third Party
  - Violation of contracts requires subverting the entire network
- Secure Peer-to-Peer agreements that live on the blockchain forever

3 ETHEREUM VIRTUAL MACHINE

ETHEREUM VIRTUAL MACHINE

COMPILATION AND PROCESS

Solidity Contract

```
contract PhilipToken {
    /*... */
}
```

Vyper Contract

```
def register(key, value):
    /*... */
```

Solidity Compiler

Vyper Compiler

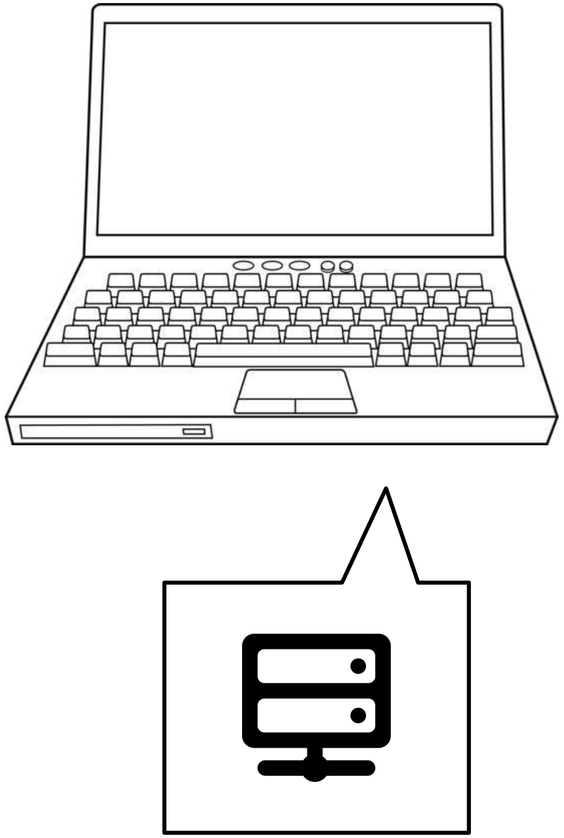
EVM Code

```
PUSH1 0x60 PUSH1 0x40 MSTORE
CALLDATASIZE ISZERO
...
PUSH2 0x1b4c
POP JUMP
```

ETHEREUM VIRTUAL MACHINE

HIGH-LEVEL OVERVIEW

- The **EVM (Ethereum Virtual Machine)** is a "mini computer" that runs contract code
- Contract code that actually gets executed on every node is EVM code
  - **EVM code**: low-level, stack based bytecode language (i.e. JVM bytecode)
- Every Ethereum node runs EVM



EVM GAS AND FEES

HIGH-LEVEL OVERVIEW

Immediate Issue:

What if our contract has an infinite loop?

Every node on the network will get stuck executing the loop forever!

- By the *halting problem*, it is impossible to determine ahead of time whether the contract will ever terminate
- ⇒ **Denial of Service Attack!**

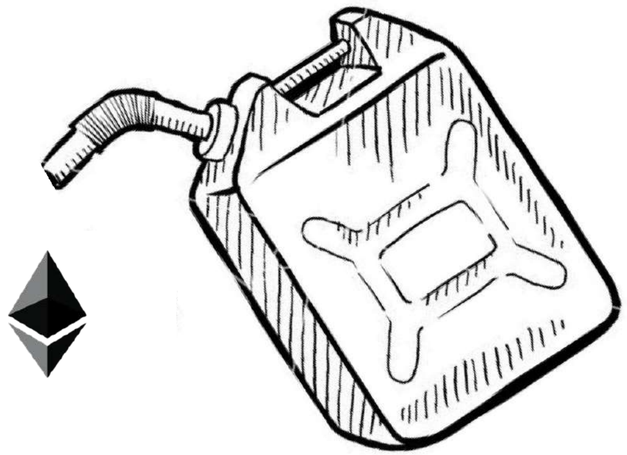
```
function foo()
{
        while (true) {
                /* Loop forever! */
        }
}
```

EVM GAS AND FEES

HIGH-LEVEL OVERVIEW

Ethereum's solution:

- Every contract requires "gas", which "fuels" contract execution
- Every EVM op-code requires some gas in order to execute
- Every transaction specifies:
  - the `startgas`, or the maximum quantity of gas it is willing to consume
  - the `gasprice`, or the fee in ether it is willing to pay per unit gas



EVM

EVM GAS AND FEES

HIGH-LEVEL OVERVIEW

- At the start of the transaction
  - `startgas * gasprice` (units = ether) are subtracted from the sender's account (the one "poking" the contract)
- If the contract **successfully executes** …
  - the remaining gas is refunded to the sender
- If the contract execution **runs out of gas** before it finishes …
  - execution reverts
  - `startgas * gasprice` are not refunded
- Purchasing gas == purchasing distributed, trustless computational power
- An attacker looking to launch a DoS attack will need to supply enough ether to fund the attack

"CLICKER" QUESTION

UNDERSTANDING THE DISTRIBUTED COMPUTER

Recall that gas is used to pay for your smart contract to be validated by this "distributed computer." **How much gas do you need to supply?**

A) Enough for 1 person to run the code

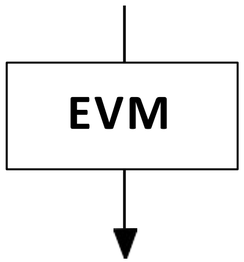
B) Enough for everyone on the network to run the code

C) Enough for the delegated validators to run the code

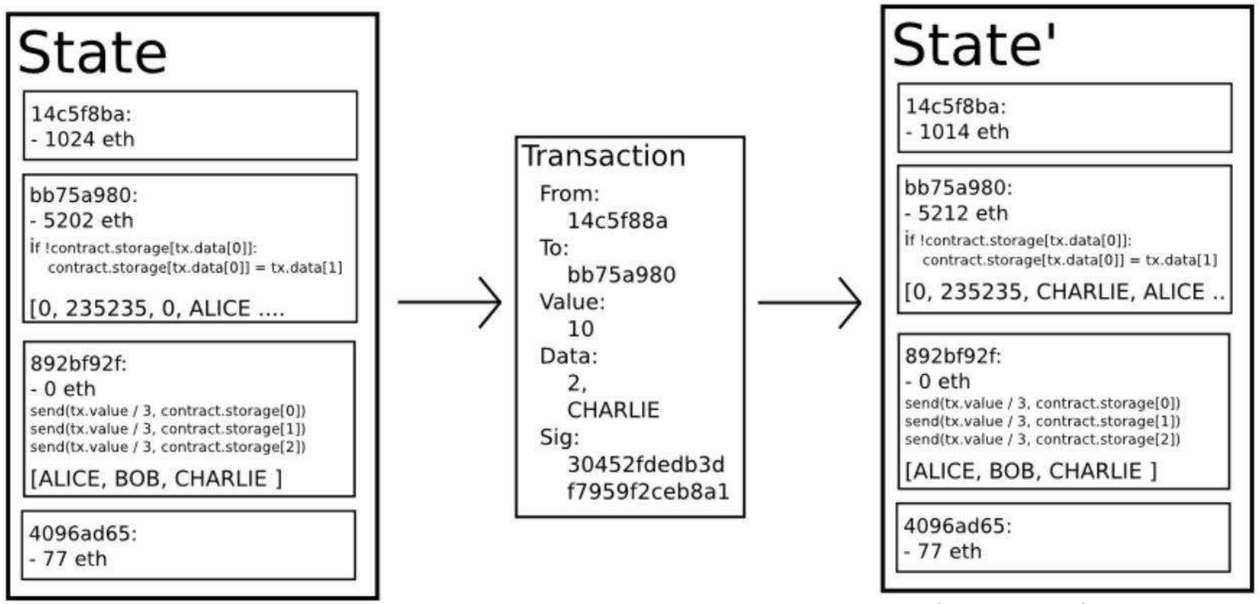
ETHEREUM NETWORK STATE

STATE TRANSITION FUNCTION

(block_state, gas, memory, transaction, message, code, stack, pc)



(block_state’, gas’)



Ethereum Whitepaper

ETHEREUM CONCLUSIONS

IT'S NOT FOR EVERYTHING

- **Ethereum is not about optimising efficiency of computation**
- Its parallel processing is **redundantly parallel**
  - **way to reach consensus** on the system state without needing trusted third parties
- Contract executions are redundantly replicated across nodes
  - ⇒ expensive, slow, memory-intensive
  - creates an **incentive not to use the blockchain** for computation that can be done off chain

ETHEREUM CONCLUSIONS

IT'S NOT FOR EVERYTHING

How would you decide between using a centralized and decentralized solution?

ETHEREUM CONCLUSIONS

IT'S NOT FOR EVERYTHING

Use Blockchain:

- Need for a **shared database** with multiple writers
- Parties **cannot trust** one another, and no trusted third party or authority is available
- Interested in fault-tolerance, data immutability or censorship resistance

Use Centralized Database:

- Database does not need to be shared, or is shared by parties who trust one another
- Must keep data **confidential**
- Must handle **complex** and/or large amounts of data
- Need to be able to edit data
- Interested in cost-effectiveness, speed or **efficiency**

4 USE CASES

4.1 BASIC USE CASES

BASIC USE CASES

SMART ASSETS

- Token System Implementation
- Database with one operation
  - Ensure Alice has enough $$ and that she initiated the transaction
  - Subtract X from Alice, give X to Bob

```
def send(to, value):
    if self.storage[msg.sender] >= value:
        self.storage[msg.sender] = self.storage[msg.sender] - value
        self.storage[to] = self.storage[to] + value
```

PUBLIC REGISTRY: Namecoin

BLOCKCHAIN FUNDAMENTALS

- DNS System
  - Maps domain name to IP address
  - “gillian.chu” => “12.34.56.78”
- Easy to implement in Ethereum

```
def register(name, value):
    if !self.storage[name]:
        self.storage[name] = value
```

DOCUMENT OWNERSHIP

BLOCKCHAIN FUNDAMENTALS

“Proof-of-Existence”

- Proves ownership of a certain document without revealing it
- **Timestamps** verify ownership later

Use Cases:

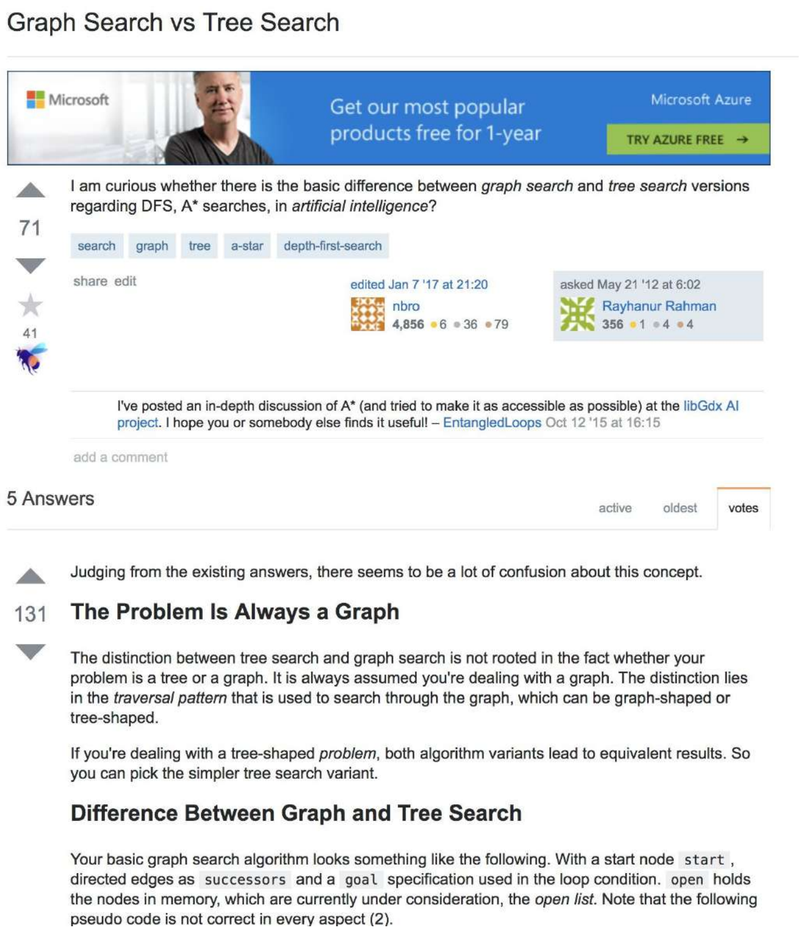
- Rent server space to store documents
  - Proof document is unmodified via hash values
  - **Integrity** guaranteed

STANDARD BOUNTIES

ETHEREUM SMART CONTRACTS

ETHDenver

- Incentivize answers to StackOverflow Q's
- Makes use of StandardBounties contract
- Uses Metamask Integration

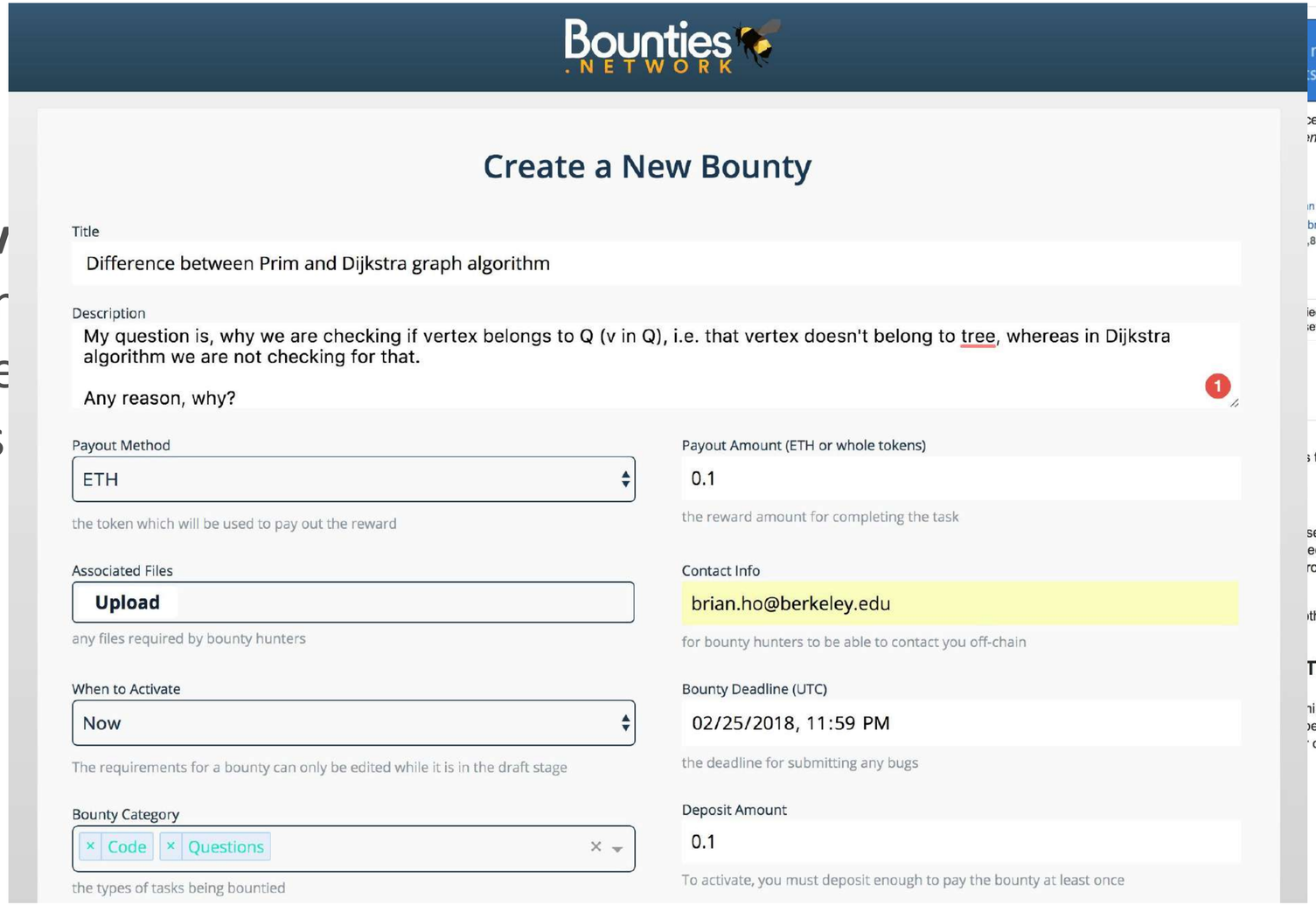


STANDARD BOUNTIES

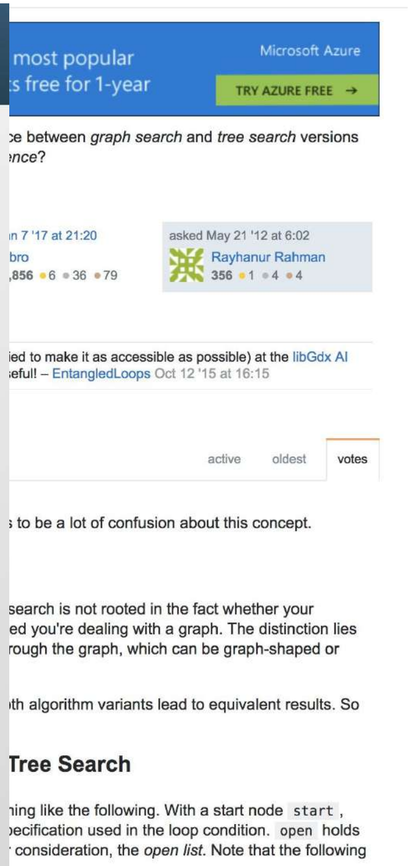
ETHEREUM SMART CONTRACTS

ETHDenv

- Incer
- Make
- Uses



Graph Search vs Tree Search

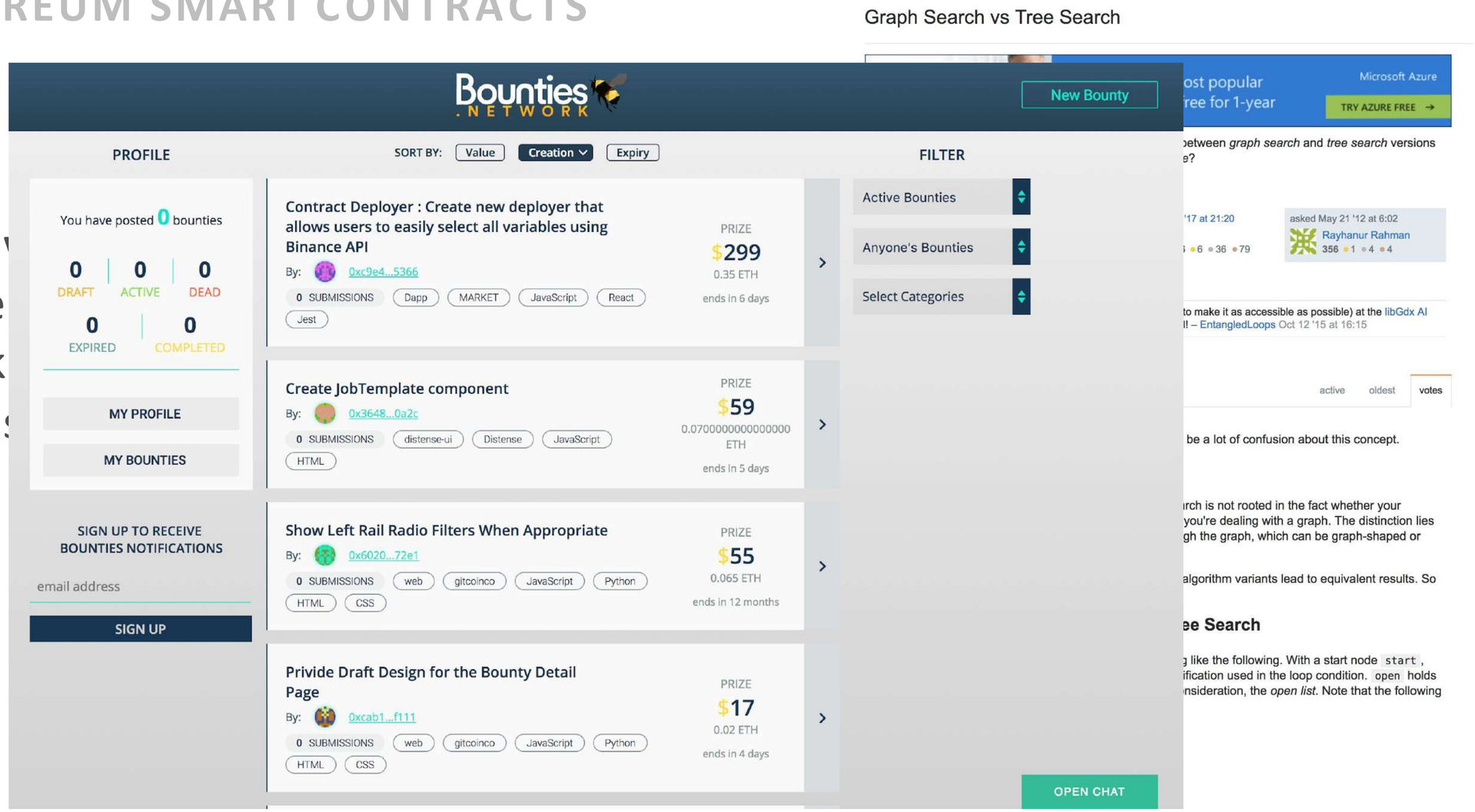


STANDARD BOUNTIES

ETHEREUM SMART CONTRACTS

ETHDen

- Ince
- Mak
- Uses



4.2 ADVANCED USE CASES

DECENTRALIZED LAND TITLES

BLOCKCHAIN FUNDAMENTALS

Problem:

- Flawed paperwork, forged signatures, **unclear documents**
- Lacking **central** government authority

Pitfalls:

- Corrupt officials accepting **bribes**, **tampering** with records
- Government cannot support land record authority
- Citizens **mistrustful** of NGO/multiple NGOs

DECENTRALIZED LAND TITLES

BLOCKCHAIN FUNDAMENTALS

The Blockchain Solution:

- Hashes & Digital Signatures
- Transparency
- Immutability
- Limits Centralization

Simple mechanism to transfer ownership, like making a transaction on Bitcoin

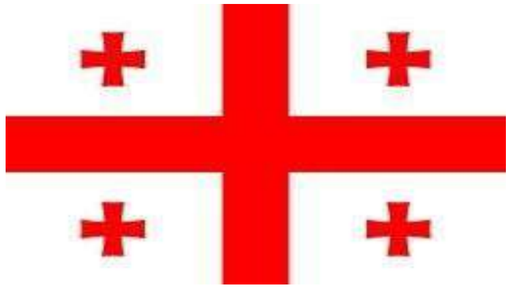
DECENTRALIZED LAND TITLES

BLOCKCHAIN FUNDAMENTALS

Caveat:

The blockchain is only as good as the information fed into it.

Countries investigating: **Georgia, Ukraine, Sweden**



PREDICTION MARKETS

A QUICK DEFINITION

Draws on the wisdom of the crowd

Ex: “Who will win the 2020 Presidential Election? Zuckerberg or Trump?”

1. Replace shares with bets
2. Random oracles report

Centralized Prediction Market

PREDICTION MARKETS

AND DECENTRALIZING THEM….

"A service that never crashes, a service that's completely transparent…"

Benefits:

- no restrictions on market creation
- shared liquidity pool
- censorship-resistant
- automatic, trustless payments

GNOSIS

augur

PREDICTION MARKETS

AND DECENTRALIZING THEM….

Use Case 1:

- Cost-effective way to “buy” information
- Bet for/against to incentivize those with insider information

GNOSIS

augur

PREDICTION MARKETS

AND DECENTRALIZING THEM….

Use Case

- Co
- Bet for/ag institution ensure eth o sew insider inform

Will this movie be a flop?

GNOSIS

augur

PREDICTION MARKETS

OTHER USE CASES

- **Hedging & Insurance**
  - "Will my house burn down?"
- **Security Bug Bounty**
  - "If my company is hacked, will the hacker reveal the vulnerability in a whitehat manner?"
- **ICO Signaling**
  - "Will my ICO deliver products on time?"

PREDICTION MARKETS

OTHER USE CASES

Also…

“I bet $1 million that Brian will be alive on October 4.”

-> Assassination Market

SUPPLY CHAIN AND PROVENANCE

BLOCKCHAIN FUNDAMENTALS

The idea: track supply chain information through a blockchain (transparent, immutable ledger)

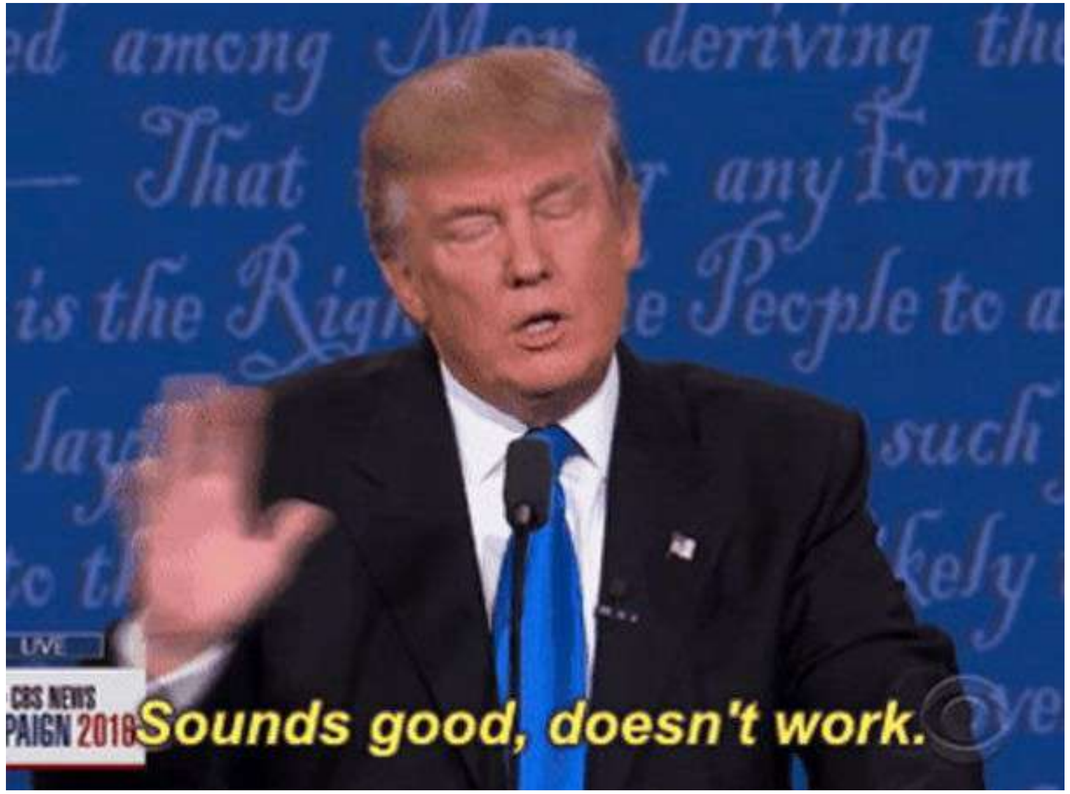
Businesses: Efficient transactions through smart contracts, easy tracking and recalls
Consumers: Transparent record of products, applications in fair trade, sustainability, and ethical consumerism

SUPPLY CHAIN AND PROVENANCE

BLOCKCHAIN FUNDAMENTALS

Problem: Need to trust data input. There is no secure way to bridge the physical and digital worlds.

We could end up with Transparent, Immutable, Integrous... Garbage.



SUPPLY CHAIN AND PROVENANCE

BLOCKCHAIN FUNDAMENTALS

Problem: "Blood" diamonds

The **Kimberley Process** is a governmental effort requiring participants to certify the origin of their diamond.

- Corrupt officials take bribes to sign certifications
- Complex supply chains mask actual trails

Everledger: provenance of diamonds

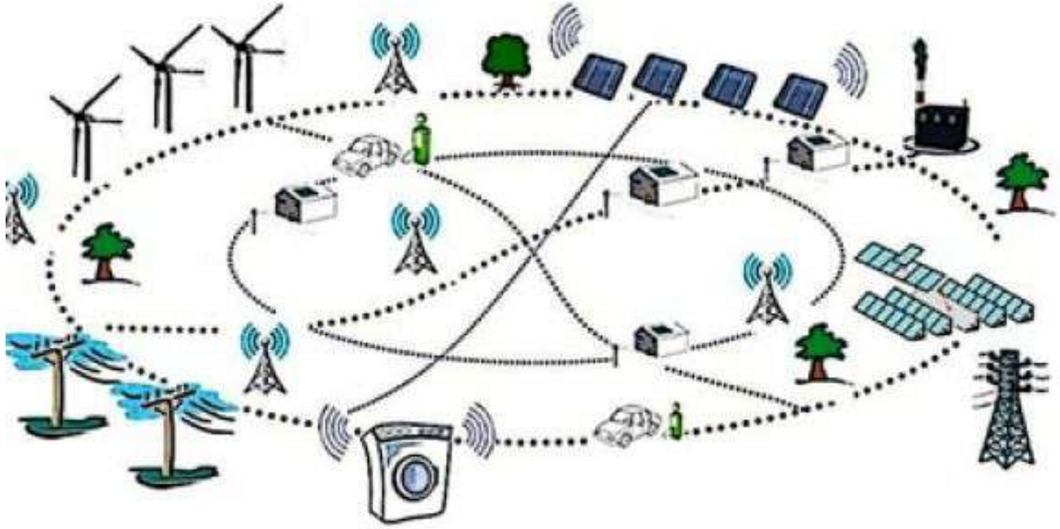
SMART ENERGY GRIDS

BLOCKCHAIN FUNDAMENTALS

Setup: Imbalances in energy across a community of houses
i.e. House A has excess heat,
House C lacks heat

Problem: Need custom infrastructure from A ⇒ B ⇒ C

Solution: Ethereum smart contract for financial commitments



COMMENTARY ON USE CASES

Blockchain vs. The Internet

READINGS

- **Finish** reading the Ethereum Whitepaper
- Take a look at these cool Ethereum Applications
  - https://www.stateofthedapps.com/

References

Slides mainly adopted from

- Blockchain @ Berkeley : https://blockchain.berkeley.edu/
- Blockchain @ Princeton : http://bitcoinbook.cs.princeton.edu/

Lec 6: Solidity, a Smart Contract Language

Blockchain in the News

Akılalmaz olay... 'Bitcoin Safiye' her yerde aranıyor

Yazdır Yazı Tipi Yorumlar

Sadece Aydın'da değil, Türkiye genelinde yüzlerce kişiyi dolandırdığı ortaya çıkan kadın Safiye Gökçen Y.'nin insanları tuzağa düşürmek için kılıktan kılığa girdiği öğrenildi. Ülke genelinde aranan dolandırıcı için yurt dışına çıkış yasağı getirildi.

Sanal para birimi bitcoin borsasında vatandaşları dolandıran kadın broker Safiye Gökçen Y.'nin, sadece Aydın'da değil, Türkiye genelinde yüzlerce kişiyi dolandırdığı ortaya çıktı. Kadın borsacı kılıktan kılığa girerek yaklaşık 375 milyon TL ile sırra kadem bastı. Yeni Asır'ın haberinden sonra çok sayıda mağduru olduğu ortaya çıkan broker, yurt dışına çıkış yasağıyla Türkiye genelinde aranıyor.

En Çok Okunan Haberler

Son dakika: Cumhurbaşkanı Erdoğan'dan 17. yıl mesajı

Aidat kavgasında ortalık karıştı

Fujitsu pirinç ticaretine blokzincirini getiriyor

Blockchain 6 Kasım 2019

Fujitsu, İsviçre merkezli girişim Rice Exchange (Ricex) ile blokzinciri tabanlı pirinç ticareti platformu için iş birliğine gitti. Platformun 2020 yılında piyasaya sürülmesi bekleniyor. Küresel piyasa değeri 450 milyar dolar değerini aşan pirinç, dünyada ticareti en...

Devamını Oku

Coca-Cola blokzinciri çalışmalarını genişletecek

Blockchain 6 Kasım 2019

Coca-cola'nın tedarik zinciri operasyonlarını yürüten IT firması Coke One North America (CONA) blokzinciri teknolojisine yönelik çalışmalarını genişletmek istiyor. Business Insider'ın haberine göre SAP ile birlikte başarılı bir test süreci geçiren firma. Blokzinciri destekli bu programı...

Devamını Oku

Microsoft'tan blokzinciri tabanlı emlak platformuna destek

Blockchain 5 Kasım 2019

Hong Kong merkezli emlak ajansı Centaline, Microsoft ve TFI ile birlikte blokzinciri tabanlı emlak platformu geliştirdi. Centaline'nin mevcut uygulamasıyla entegre olan platform, değiştirilemez ve dijital olarak imzalanmış belgelere olanak tanıyor. 60.000 çalışanı ve 2000 şubesiyle...

Devamını Oku

Huawei, Çin'in dijital para araştırmalarına destek olacak

Blockchain 5 Kasım 2019

Çin'in blokzinciri teknolojisini hızlı bir şekilde benimsemeye çalıştığı şu günlerde teknoloji devi Huawei de Dijital Para Birimi Araştırma Enstitüsü ile anlaşma imzaladığını duyurdu. Huawei, Merkez Bankası'na bağlı Dijital Para Birimi Araştırma Enstitüsü ile anlaşmaya gittiklerini...

Devamını Oku

Çin, şehirler için kimlik tanıma sistemini başlatıyor

Blockchain 5 Kasım 2019

Çin, akıllı şehir altyapısının bir parçası şehirler için geliştirilen blokzinciri tabanlı bir kimlik tanımlama sistemi ile blokzincir alanındaki faaliyetlerini geliştirmeye devama diyor Global Times'ın haberine göre yeni geliştirilen kimlik tanıma sistemi Shijiazhuang şehrinde üç kurum...

Devamını Oku

Evolution of Blockchain Technology

- 1st generation: Store and transfer of value (e.g. Bitcoin, Ripple, Dash)
- 2nd generation: Programmable via smart contracts (E.g. Ethereum)
- 3rd generation: Enterprise blockchains (E.g. Hyperledger, R3 Corda & Ethereum Quorum)
- Next gen: Highly scalable with high concurrency (E.g. RChain)

Permissionless vs. Permissioned

- Permissionless/Public
  - Decentralized Trustless Blockchains – Anyone can process blocks
  - Consensus typically reached via PBFT or Proof of Work
  - Miners compensated for validation
  - E.g. **Ethereum**, Bitcoin, Dash & Ripple
- Permissioned/Federated
  - Trusted: - All processors of blocks are known
  - Less processing required for consensus hence faster
  - E.g. **Hyperledger**, R3 Corda & Ethereum Quorum

Developing on Ethereum

THE ETHEREUM VIRTUAL MACHINE

THE WORLD COMPUTER

- Ethereum implements an execution environment known as the *Ethereum Virtual Machine*
  - Every node participating in the network runs the EVM
- Nodes will go through the transactions listed in the block they are verifying and run the code as triggered by the transaction within the EVM
- Every full node does the same calculations and stores the same values

OPCODES

WHAT MAKES THE COMPUTER TICK

See here for more

Value	Mnemonic	Gas Used	Subset	Removed from stack	Added to stack	Notes
0x00	STOP	0	zero	0	0	Halts execution.
0x01	ADD	3	verylow	2	1	Addition operation
0x02	MUL	5	low	2	1	Multiplication operation.
0x03	SUB	3	verylow	2	1	Subtraction operation.
0x04	DIV	5	low	2	1	Integer division operation.
0x05	SDIV	5	low	2	1	Signed integer division ope
0x06	MOD	5	low	2	1	Modulo remainder operatic
0x07	SMOD	5	low	2	1	Signed modulo remainder
0x08	ADDMOD	8	mid	3	1	Modulo addition operation.
0x09	MULMOD	8	mid	3	1	Modulo multiplication oper
0x0a	EXP	(exp == 0) ? 10 : (10 + 10 * (1 + log256(exp)))		2	1	Exponential operation.
0x0b	SIGNEXTEND	5	low	2	1	Extend length of two's com
0x10	LT	3	verylow	2	1	Less-than comparison.
0x11	GT	3	verylow	2	1	Greater-than comparison.

OPCODES

WHAT MAKES THE COMPUTER TICK

See here for more

Here given are the various exceptions to the state transition rules given in section 9 specified for each instruction, together with the additional instruction-specific definitions of J and C. For each instruction, also specified is α, the additional items placed on the stack and δ, the items removed from stack, as defined in section 9.

0s: Stop and Arithmetic Operations

All arithmetic is modulo 2^{256} unless otherwise noted.

Value	Mnemonic	δ	α	Description
0x00	STOP	0	0	Halts execution.
0x01	ADD	2	1	Addition operation. $\mu'_s[0] \equiv \mu_s[0] + \mu_s[1]$
0x02	MUL	2	1	Multiplication operation. $\mu'_s[0] \equiv \mu_s[0] \times \mu_s[1]$
0x03	SUB	2	1	Subtraction operation. $\mu'_s[0] \equiv \mu_s[0] - \mu_s[1]$
0x04	DIV	2	1	Integer division operation. $\mu'_s[0] \equiv \begin{cases} 0 & \text{if } \mu_s[1] = 0 \\ \lfloor \mu_s[0] \div \mu_s[1] \rfloor & \text{otherwise} \end{cases}$
0x05	SDIV	2	1	Signed integer division operation (truncated). $\mu'_s[0] \equiv \begin{cases} 0 & \text{if } \mu_s[1] = 0 \\ -2^{255} & \text{if } \mu_s[0] = -2^{255} \wedge \mu_s[1] = -1 \\ \mathbf{sgn}(\mu_s[0] \div \mu_s[1]) \lfloor \lvert \mu_s[0] \div \mu_s[1] \rvert \rfloor & \text{otherwise} \end{cases}$ Where all values are treated as two's complement signed 256-bit integers. Note the overflow semantic when -2^{255} is negated.
0x06	MOD	2	1	Modulo remainder operation. $\mu'_s[0] \equiv \begin{cases} 0 & \text{if } \mu_s[1] = 0 \\ \mu_s[0] \bmod \mu_s[1] & \text{otherwise} \end{cases}$

THE ETHEREUM VIRTUAL MACHINE

THE WORLD COMPUTER

- Solidity lets you program on Ethereum, a blockchain-based virtual machine that allows the creation and execution of smart contracts, without requiring centralized or trusted parties
- Statically typed, contract programming language that has similarities to Javascript and C
  - Like objects in OOP, each contract contains state variables, functions, and common data types
  - Contract-specific features include modifier (guard) clauses, event notifiers for listeners, and custom global variables

Smart Contracts

- Execute on the Ethereum Virtual Machine (EVM)
- Languages:
  - Solidity: most popular, Turing complete, similar to JavaScript

```
contract Coin {
    address minter;
    mapping (address => uint) balances;
    function Coin() {
        minter = msg.sender;
    }
    function mint(address owner, uint amount) {
        if (msg.sender != minter) return;
        balances[owner] += amount;
    }
    function send(address receiver, uint amount) {
        if (balances[msg.sender] < amount) return;
        balances[msg.sender] -= amount;
        balances[receiver] += amount;
    }
    function queryBalance(address addr) constant returns (uint balance) {
        return balances[addr];
    }
}
```

SOLIDITY -EXAMPLE

THE BANK CONTRACT

LEARNING FAST

WHAT DOES A BANK NEED TO DO?

1. Allow Deposits
2. Allow Withdrawals
3. Balance Checks

Learn X in Y minutes, a whirlwind tour of your favorite language

```
contract SimpleBank {
    mapping (address => uint) private balances;
    address public owner;
    event LogDepositMade(address accountAddress, uint amount);

    function SimpleBank() {
        owner = msg.sender;
    }

    function deposit() public returns (uint) {
        balances[msg.sender] += msg.value;
        LogDepositMade(msg.sender, msg.value);
        return balances[msg.sender];
    }

    function withdraw(uint withdrawAmount) public returns (uint remainingBal) {
        if(balances[msg.sender] >= withdrawAmount) {
            balances[msg.sender] -= withdrawAmount;
            if (!msg.sender.send(withdrawAmount)) {
                balances[msg.sender] += withdrawAmount;
            }
        }
        return balances[msg.sender];
    }

    function balance() constant returns (uint) {
        return balances[msg.sender];
    }

    function () {
        throw;
    }
}
```

THE BANK CONTRACT

LEARNING FAST

```
contract SimpleBank {


    mapping (address => uint) private balances;



    address public owner;

```

- **`contract`** has similarities to **`class`** in other languages (class variables, inheritance, etc.)
  - Declare state variables outside function, persist through life of contract
- **`mapping`** is a dictionary that maps addresses to balances
  - always be careful about overflow attacks with numbers
  - **`private`** means that other contracts can't directly query balances
  - but data is still viewable to other parties on blockchain
- **`public`** makes externally readable (not writeable) by users or contracts

THE BANK CONTRACT

LEARNING FAST

- **`event`** - publicize actions to external listeners
- **`Constructor`** - can receive one or many variables here; only one allowed
- **`msg`** provides details about the message that's sent to the contract
  - **`msg.sender`** is contract caller (address of contract creator)

```
event LogDepositMade(address accountAddress, uint amount);

function SimpleBank() {
    owner = msg.sender;
}
```

THE BANK CONTRACT

LEARNING FAST

- **`deposit()`**
  - Takes no parameters, but we are still sending Ether!
  - **`public`** makes externally readable (not writeable) by users or contracts
  - Returns user's balance as an unsigned integer (**`uint`**)

```
function deposit() public returns (uint) {


    balances[msg.sender] += msg.value;


    LogDepositMade(msg.sender, msg.value);


    return balances[msg.sender];
}
```

- **`balances[msg.sender]`**, no **`this`** or **`self`** required with state variable
- **`LogDepositMade`** event fired

THE BANK CONTRACT

LEARNING FAST

```
function withdraw(uint withdrawAmount) public returns (uint remainingBal) {

    if(balances[msg.sender] >= withdrawAmount) {

        balances[msg.sender] -= withdrawAmount;

        if (!msg.sender.send(withdrawAmount)) {

            balances[msg.sender] += withdrawAmount;
        }
    }
    return balances[msg.sender];
}
```

- **`Withdraw()`**
  - **`withdrawAmount`** parameter
  - Returns user's balance
- Note the way we deduct the balance right away, before sending
  - We do this because of the risk of a recursive call that allows the caller to request an amount greater than their balance
- Increment back only on fail, as may be sending to contract that has overridden 'send' on the receipt end

THE BANK CONTRACT

LEARNING FAST

- **balance():**
  - constant prevents function from editing state variables
  - Returns user's balance

```
function balance() constant returns (uint) {

    return balances[msg.sender];
}
```

THE BANK CONTRACT

LEARNING FAST

() : Fallback function - Called if other functions don't match call or sent ether without data

- Typically, called when invalid data is sent
- Ether sent to this contract is reverted if the contract fails otherwise

- **throw** : throw reverts state to before call

```
function () {

    throw;
}
```

Solidity Language

Layout of a Solidity Source File

Pragmas

```
pragma solidity ^0.4.0;
```

Importing other Source Files

```
import "filename";
```

Structure of a Contract

State Variables

State variables are variables whose values are permanently stored in contract storage.

```
pragma solidity >=0.4.0 <0.6.0;
contract SimpleStorage {
        uint storedData; // State variable
        // ...
}
```

Functions

Functions are the executable units of code within a contract

```
pragma solidity >=0.4.0 <0.6.0;
contract SimpleAuction {

        function bid() public payable {
                // Function // ...
        }
}
```

DATA TYPES

INTEGERS

```
// uint used for currency amount (there are no doubles or floats) and for dates (in unix time)
uint x;

// int of 256 bits, cannot be changed after instantiation
int constant a = 8;
int256 constant a = 8; //same effect as line above, here the 256 is explicit
uint constant VERSION_ID = 0x123A1; //A hex constant
```

DATA TYPES

INTEGERS

Read the docs

```
// For int and uint, can explicitly set space in steps of 8 up to 256; e.g. int8, int16, int24
uint8 b;
int64 c;
uint248 e;

// Be careful that you don't overflow, and protect against attacks that do
// No random functions built in, use other contracts for randomness
```

DATA TYPES

BOOLEAN, ADDRESS

```
bool b = true;

// Addresses - holds 20 byte/160 bit Ethereum addresses
// No arithmetic allowed
address public owner;
```

DATA TYPES

ADDRESS, SENDING ETHER

```
address public owner;
// All addresses can be sent ether
owner.send(SOME_BALANCE); // returns false on failure


if (owner.send(*20 ether*)) {}
// REMEMBER: wrap in 'if', as contract addresses have
// functions executed on send and these can fail
// Also, make sure to deduct balances BEFORE attempting a send, as there is a risk of a
recursivecall that can  drain the contract
```

DATA TYPES

WEI vs ETHER

- *Wei*

  It's a denomination, like cents to Dollars / pennies to Pounds / kuruş to TL:

  1: wei
  10^3: (unspecified)
  10^6: (unspecified)
  10^9: (unspecified)
  10^12: szabo
  10^15: finney
  10^18: ether

DATA TYPES

ADDRESS, SENDING ETHER

- Balance of the address in *Wei*

```
<address>.balance
```

- Send given amount of Wei to address, throws on failure

```
<address>.transfer(uint256 amount)
```

DATA TYPES

BYTES

```
// Bytes available from 1 to 32
byte a; // byte is same as bytes1
bytes2 b;
bytes32 c;

// Dynamically sized bytes
bytes m; // A special array, same as byte[] array (but packed tightly)
// More expensive than bytes1 - bytes32, so use those when possible
```

DATA TYPES

BYTES

```
// same as bytes, but does not allow length or index access (for now)
string n = "hello";

// stored in UTF-8, note double quotes, not single

// string utility functions to be added in future

// prefer bytes32/bytes, as UTF-8 uses more storage
```

DATA TYPES

BYTES

```
// Type inference
// var does inferred typing based on first assignment,
// can't be used in functions parameters
var a = true;
// use carefully, inference may provide wrong type
// e.g., an int8, when a counter needs to be int16
```

DATA TYPES

FUNCTION ASSIGNMENT, DEFAULT VALUES, DELETE, UNWRAP TUPLES

```
// Variables can be used to assign function to variable
function a(uint x) returns (uint) {
   return x * 2;
}
var f = a;
f(22); // call


// By default, all values are set to 0 on instantiation
// Delete can be called on most types
// (does NOT destroy value, but sets value to 0, the initial value)
```

DATA STRUCTURES

DATA STRUCTURES

ARRAYS

```
// Arrays
bytes32[5] nicknames; // static array
bytes32[] names; // dynamic array
uint newLength = names.push("John"); // adding returns new length of the array
// Length
names.length; // get length
names.length = 1; // lengths can be set (for dynamic arrays in storage only)
// multidimensional array
uint x[][5]; // arr with 5 dynamic array elements (opposite order of most languages)
```

DATA STRUCTURES

MAPPINGS

```
// Dictionaries (any type to any other type)
mapping (string => uint) public balances;
balances["charles"] = 1;
console.log(balances["ada"]); // is 0, all non-set key values return zeroes
// 'public' allows following from another contract
contractName.balances("charles"); // returns 1
// 'public' created a getter (but not setter) like the following:
function balances(string _account) returns (uint balance) {
   return balances[_account];
}
```

DATA STRUCTURES

MAPPINGS

```
// Nested mappings
mapping (address => mapping (address => uint)) public custodians;

// To delete
delete balances["John"];
delete balances; // sets all elements to 0
// Unlike other languages, CANNOT iterate through all elements in
// mapping, without knowing source keys - can build data structure
// on top to do this
```

DATA STRUCTURES

STRUCTS

```
// Structs and enums
struct Bank {
    address owner;
    uint balance;
}
Bank b = Bank({
    owner: msg.sender,
    balance: 5
});
```

DATA STRUCTURES

STRUCTS

```
// or
struct Bank {
    address owner;
    uint balance;
}
Bank c = Bank(msg.sender, 5);
delete b;
// sets to initial value, set all variables in struct to 0, except mappings
```

DATA STRUCTURES

ENUMS

```
// Enums
enum State { Created, Locked, Inactive } // often used for state machine
State public state; // Declare variable from enum
state = State.Created;

// enums can be explicitly converted to ints
uint createdState = uint(State.Created); // 0
```

WHERE DATA GOES

NOTE ON MEMORY AND STORAGE

- Data locations: `memory` vs. `storage` vs. *stack* - all complex types (arrays, structs) have a data location
  - `memory` does not persist, `storage` does
- Default is `storage` for local and state variables; `memory` for function arguments
  - For most types, the data location to use can be explicitly set
- The stack holds small local variables
  - Used for values in intermediate calculations
  - General consensus is not to interact with it as a developer

GLOBAL VARIABLES

this

```
this; // address of contract

this.balance; // often used at end of contract life to send remaining balance to party
this.someFunction(); // calls func externally via call, not via internal jump
```

GLOBAL VARIABLES

msg, tx

```
// msg - Current message received by the contract
msg.sender; // address of sender
msg.value; // amount of ether provided to this contract in wei
msg.data; // bytes, complete call data
msg.gas; // remaining gas

// tx - This transaction
tx.origin; // address of sender of the transaction, will always be a user
tx.gasprice; // gas price of the transaction
```

GLOBAL VARIABLES

block, storage

```
// block - Information about current block
now; // current time (approximately), alias for block.timestamp (uses Unix time)
block.number; // current block number
block.difficulty; // current block difficulty
block.blockhash(1); // hash of the given block - returns bytes32, only works for most recent
256 blocks
block.gasLimit();
```

FUNCTIONS

```
// Functions can return many arguments, and by specifying returned
// arguments name don't need to explicitly return
function increment(uint x, uint y) returns (uint x, uint y) {
    x += 1;
    y += 1;
}
// Call previous function
uint (a,b) = increment(1,1);
```

FUNCTIONS

```
// 'constant' indicates that function does not/cannot change persistent
vars
// Constant functions execute locally, not on blockchain
uint y;

function increment(uint x) constant returns (uint x) {
    x += 1;
    y += 1; // this line would fail
    // y is a state variable, and can't be changed in a constant function
}
```

FUNCTIONS

VISIBILITY

```
// These can be placed where 'constant' is, including:
// public - visible externally and internally (default)
// external - only visible externally
// private - only visible in the current contract
// internal - only visible in current contract, and those deriving from it

// Functions hoisted - and can assign a function to a variable
function a() {
    var z = b;
    b();
}

function b() {

}

// Prefer loops to recursion (max call stack depth is 1024)
```

FUNCTIONS

FALLBACK

```
// Fallback function - Called if other functions don't match call or
// sent ether without data
// Typically, called when invalid data is sent
// Added so ether sent to this contract is reverted if the contract fails
// otherwise, the sender's money is transferred to contract
function () {
    revert(); // reverts state to before call
}
```

EVENTS

EVENTS ARE AMAZING

```
// Declare
event LogSent(address indexed from, address indexed to, uint amount);

// note capital first letter

// Call
Sent(from, to, amount);
// For an external party (a contract or external entity), to watch:
Coin.Sent().watch({}, '', function(error, result) {
    if (!error) {
        console.log("Coin transfer: " + result.args.amount +
        "   coins were sent from " + result.args.from +
        "   to " + result.args.to + ".");
        console.log("Balances now:\n" +
            "Sender: " + Coin.balances.call(result.args.from) +
            "Receiver: " + Coin.balances.call(result.args.to));
    }
}
// Common paradigm for one contract to depend on another (e.g., a
// contract that depends on current exchange rate provided by another)
```

MODIFIERS

MODIFIERS ARE ALSO AMAZING

```
// C. Modifiers
// Modifiers validate inputs to functions such as minimal balance or
user auth;
// similar to guard clause in other languages

// '_' (underscore) often included as last line in body, and indicates
// function being called should be placed there
modifier onlyAfter(uint _time) { if (now <= _time) throw; _ }


modifier onlyOwner { if (msg.sender == owner) _ }
// commonly used with state machines


modifier onlyIfState (State currState) { if (currState != State.A) _ }
```

MODIFIERS

MODIFIERS ARE ALSO AMAZING

```
// Append right after function declaration
function changeOwner(newOwner)
onlyAfter(someTime)
onlyOwner()
onlyIfState(State.A)
{
    owner = newOwner;
}

// underscore can be included before end of body,
// but explicitly returning will skip, so use carefully
modifier checkValue(uint amount) {

    _
    if (msg.value > amount) {
        uint amountToRefund = amount - msg.value;
        if (!msg.sender.send(amountToRefund)) {
            throw;
        }
    }
}
```

LOOPS

LOOPS ARE DANGEROUS

```
// All basic logic blocks work - including if/else, for, while, break, continue
// return - but no switch

// Syntax same as javascript, but no type conversion from non-boolean
// to boolean (comparison operators must be used to get the boolean val)

// For loops that are determined by user behavior, be careful - as contracts
have a maximal amount of gas for a block of code - and will fail if that is
exceeded
// For example:
for(uint x = 0; x < refundAddressList.length; x++) {
    if (!refundAddressList[x].send(SOME_AMOUNT)) {
       throw;
    }
}
```

EXTERNAL CONTRACTS

USAGE

```
contract InfoFeed {function info() returns (uint ret) { return 42; }}

contract Consumer {
    InfoFeed feed; // points to contract on blockchain


    function setFeed(address addr) { // Set feed to existing contract instance
        feed = InfoFeed(addr);// automatically cast, be careful; constructor is not called
    }

    function createNewFeed() { // Set feed to new instance of contract
        feed = new InfoFeed(); // new instance created; constructor called
    }

    function callFeed() {
        // final parentheses call contract, can optionally add
        // custom ether value or gas
        feed.info.value(10).gas(800)();
    }
}
```

INHERITANCE

USAGE

```
// Order matters, last inherited contract (i.e., 'def') can override parts of
// previously inherited contracts
contract MyContract is abc, def("a custom argument to def") {

// Override function
    function z() {
        if (msg.sender == owner) {
            def.z(); // call overridden function from def
            super.z(); // call immediate parent overriden function
        }
    }
}

// abstract function
function someAbstractFunction(uint x);
// cannot be compiled, so used in base/abstract contracts
// that are then implemented
```

IMPORTS

USAGE

```
// C. Import

import "filename";
import "github.com/ethereum/dapp-bin/library/iterable_mapping.sol";
import "github.com/Arachnid/solidity-stringutils/strings.sol";
// Importing under active development
// Cannot currently be done at command line
```

REVERT, REQUIRE, ASSERT

USAGE

The revert function can be used to flag an error and revert the current call

The require function should be used to ensure valid conditions, such as inputs, or contract state variables are met, or to validate return from calls to external contracts.

assert acts like it might in any other programming language. Use of revert require should guarantee that your assert does not fail

REVERT, REQUIRE, ASSERT

USAGE

assert(bool condition):

throws if the condition is not met - to be used for internal errors. (Generally used to stop very bad things from happening.)

require(bool condition):

throws if the condition is not met - to be used for errors in inputs or external components.

revert():

abort execution and revert state changes

SELFDESTRUCT

USAGE

```
// selfdestruct current contract, sending funds to address (often creator)
selfdestruct(SOME_ADDRESS);

// removes storage/code from current/future blocks
// helps thin clients, but previous data persists in blockchain

// Common pattern, lets owner end the contract and receive remaining funds
function remove() {
    if(msg.sender == creator) { // Only let the contract creator do this
        selfdestruct(creator); // Makes contract inactive, returns funds
    }
}

// May want to deactivate contract manually, rather than selfdestruct
// (ether sent to selfdestructed contract is lost)
```

STORAGE

TIPS

- Storage is expensive because it is permanent
- Things like multidimensional arrays are expensive
- Store things OFF THE BLOCKCHAIN **unless absolutely necessary**

STORAGE

TIPS

THERE ARE NO SECRETS ON THE BLOCKCHAIN

All data on the blockchain can be easily viewed by anyone in the world

UNITS OF TIME

USAGE

```
// Currency is defined using wei, smallest unit of Ether
uint minAmount = 1 wei;
uint a = 1 finney; // 1 ether == 1000 finney
// Time units
1 == 1 second
1 minutes == 60 seconds

// Can multiply a variable times unit, as units are not stored in a variable
uint x = 5;
(x * 1 days); // 5 days

// Careful about leap seconds/years with equality statements for time
// (instead, prefer greater than/less than)
```

Demo

http://remix.ethereum.org

Hello world

```
pragma solidity ^0.4.18;

contract Hello{

        string greeting;

        function Hello(string _greeting) public {
                greeting = _greeting;
        }

        function greet() public constant returns (string)
        {
                return greeting;
        }
}
```

READINGS

https://coursetro.com/courses/20/Developing-Ethereum-Smart-Contracts-for-Beginners

Finish the tutorial – 8 videos (not written lessons ,click on play button)

reference:

https://solidity.readthedocs.io/en/develop/solidity-in-depth.html

References

Slides mainly adopted from

- Blockchain @ Berkeley : https://blockchain.berkeley.edu/
- Blockchain @ Princeton : http://bitcoinbook.cs.princeton.edu/

Lecture 7: HOW TO DESTROY BITCOIN: GAME THEORY AND ATTACKS

LECTURE OVERVIEW

1. POOL STRATEGIES
2. FORKING &DOUBLE SPENDING
3. CENSORSHIP
4. SELFISH MINING
5. DEFENSES DISCUSSION

1 POOL STRATEGIES

POOL REWARD SCHEMES

PAY-PER-SHARE

Pool pays out **at every share submitted.** By default will be proportional to work done by individuals

- More beneficial for **miners**
- Individual miners have no risk from reward variance
  - Pool takes on the risk completely
- Problem: No incentive for individuals to actually submit valid blocks
  - Individuals are paid regardless

POOL REWARD SCHEMES

PROPORTIONAL

Pool pays out **when blocks are found,** proportional to the work individuals have submitted for this block

- More beneficial for the **pool**
- Individual miners still bear some risk in variance proportional to size of the pool
  - Not a problem if pool is sufficiently large
- Lower risk for pool operators - only pay out when reward is found
  - Individuals thus incentivized to submit valid blocks

POOL REWARD SCHEMES

INCENTIVE MISALIGNMENT

Is it possible that mining pools are vulnerable to some incentive misalignment? Can miners take advantage of the differences in these two types of payout schemes?

POOL REWARD SCHEMES

INCENTIVE MISALIGNMENT

Is it possible that mining pools are vulnerable to some incentive misalignment? Can miners take advantage of the differences in these two types of payout schemes?

YES!

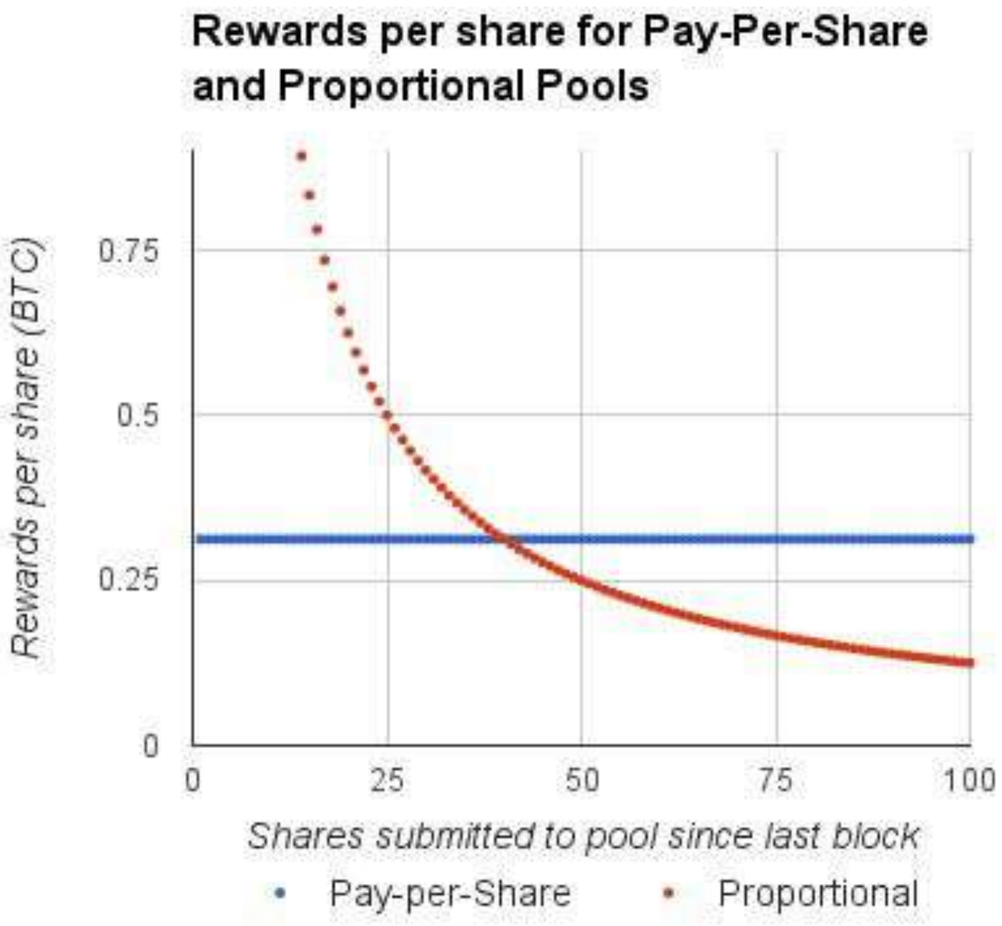
POOL REWARDS

POOL HOPPING

Proportional pool rewards per share high when just starting on a new block, but inversely proportional to number of shares submitted

Pay-per-share pool rewards per share constant no matter how many shares submitted

How does a miner maximize their profit?



Parameters:

- Pool has 10% of network hashrate
- 4 shares expected per valid block

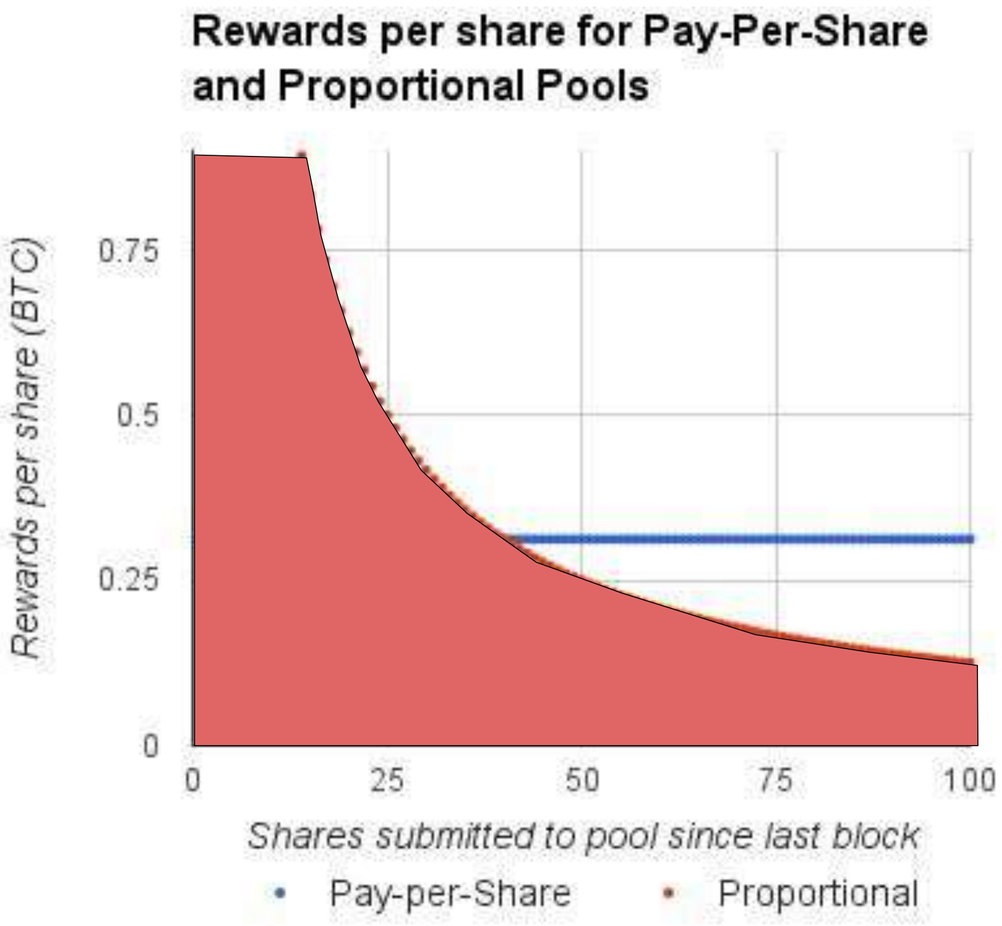
POOL REWARDS

POOL HOPPING

Proportional pool rewards per share high when just starting on a new block, but inversely proportional to number of shares submitted

Pay-per-share pool rewards per share constant no matter how many shares submitted

How does a miner maximize their profit?



Parameters:

- Pool has 10% of network hashrate
- 4 shares expected per valid block

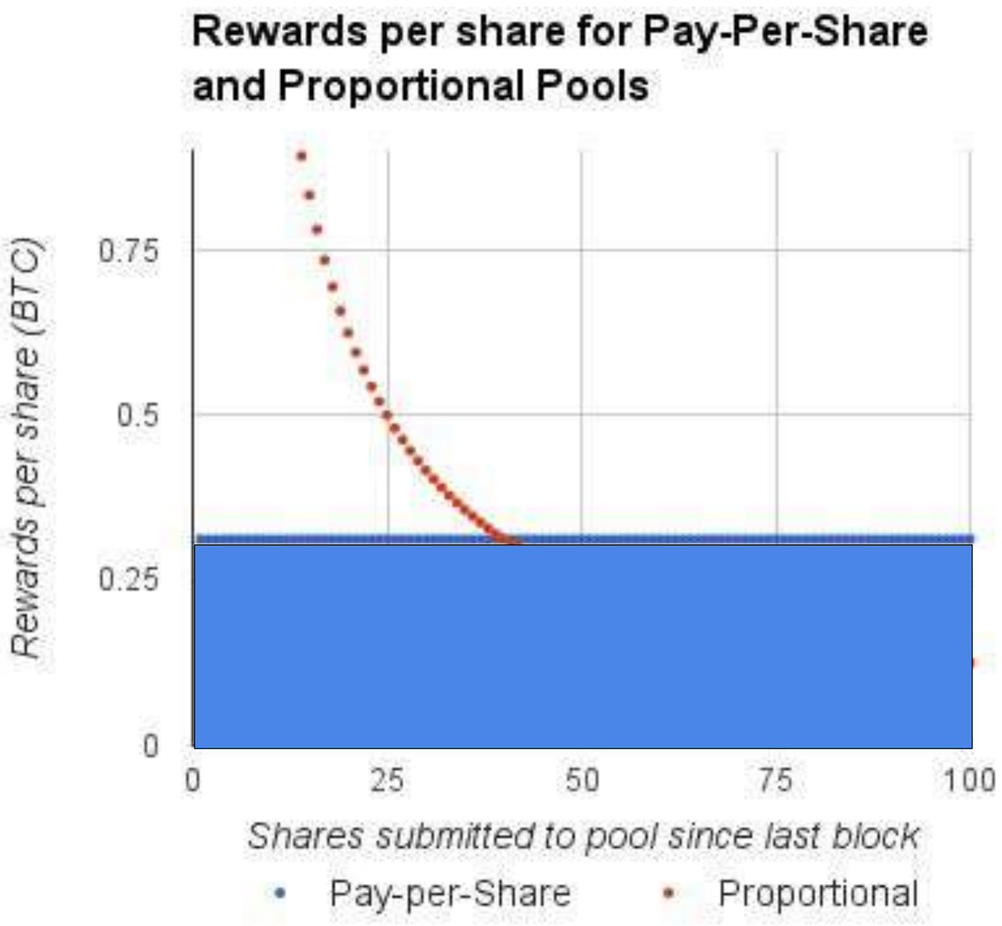
POOL REWARDS

POOL HOPPING

Proportional pool rewards per share high when just starting on a new block, but inversely proportional to number of shares submitted

Pay-per-share pool rewards per share constant no matter how many shares submitted

How does a miner maximize their profit?



Parameters:

- Pool has 10% of network hashrate
- 4 shares expected per valid block

POOL REWARDS

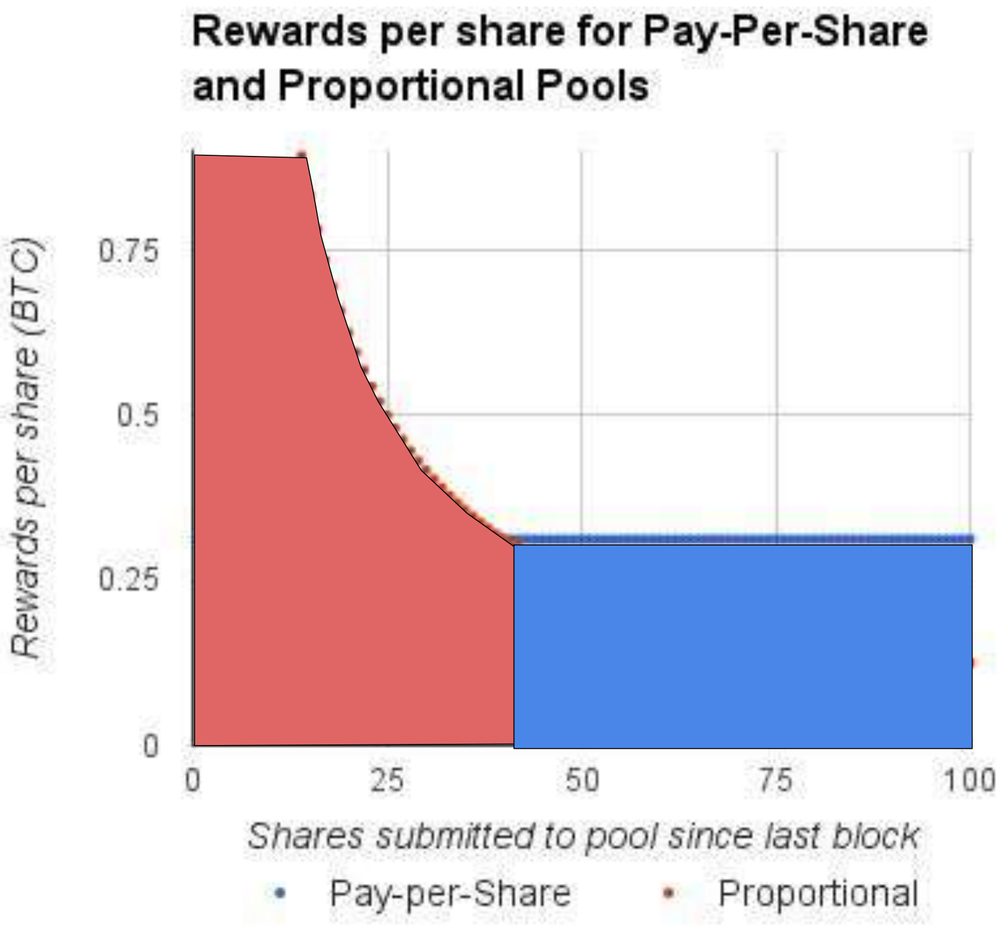
POOL HOPPING

Pool hopping: switching between pools to increase total rewards

- Proportional pool pays larger amount per share if a block is found quickly

Example profit maximizing strategy:

- Mine at **proportional** pool shortly after a block was found (while rewards are high)
- Switch to **pay-per-share** pool when once proportional pool is less profitable



Parameters:

- Pool has 10% of network hashrate
- 4 shares expected per valid block

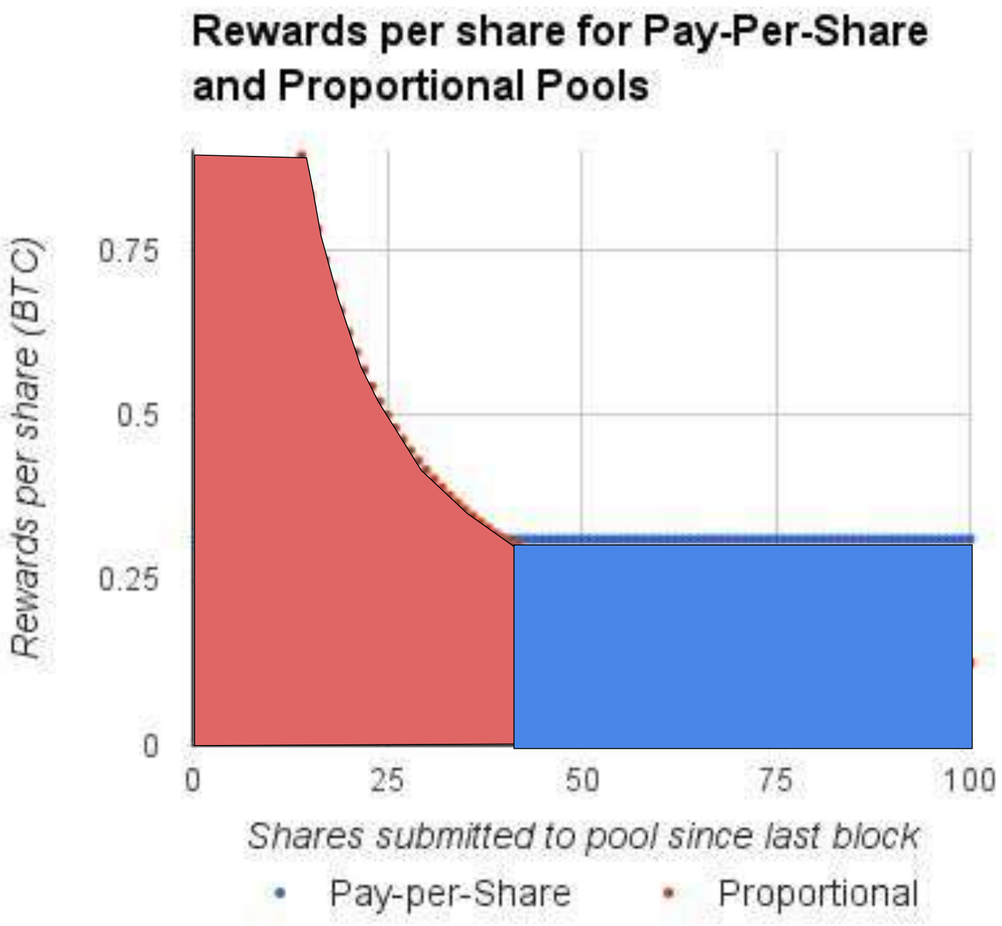
POOL REWARDS

POOL HOPPING

Therefore, proportional pools are **not feasible in practice**

- Honest miners who stay loyal to one pool are cheated out of their money

Designing a mining pool reward scheme with aligned incentives that is not vulnerable to pool hopping remains an **open problem**



Parameters:

- Pool has 10% of network hashrate
- 4 shares expected per valid block

POOL CANNIBALIZATION

NOM NOM NOM NOM

Cannibalizing Pools - Distribute some small % of mining power equally among all other pools, withhold valid blocks.

- Rewards will still be received
- Undetectable unless statistically significant

Image source: http://cdn3-www.craveonline.com/assets/uploads/2014/08/Hannibal-Cookbook.jpg

POOL CANNIBALIZATION

EXAMPLE

Givens:

- We have 30 H/s. Total network is 100 H/s.
- Assume 1BTC block reward. All of the following numbers are expected value.
- 30% HR (hashrate)
  - = 30% MR (Mining Reward)
  - = 0.3 BTC

Let's say we buy more mining equipment, worth 1H/s

Standard mining strategy:

- Add 1% HR => 31/101 = 30.69% HR
- Reward: 0.3069 *1BTC = 0.3069 BTC
  - Revenue gain = **0.0069 BTC for 1% hashrate added**

New Hashrate	31/101 ≈ 30.69%
Mining Reward	0.3069 BTC
Revenue Gain	0.0069 BTC

POOL CANNIBALIZATION

EXAMPLE

Pool cannibalizing strategy:

Distribute 1H/s among all other pools

- **DON'T SUBMIT VALID BLOCKS**

Other pool hashrate breakdown:

- 70/71 honest, 1/71 dishonest
  - 70% **effective** hashrate = 0.7 BTC
- You own 1/71 of other pools
  - Expected value of mining: (1/71)*0.7 BTC = 0.0098 BTC

New Hashrate	31/101 ≈ 30.69%
Mining Reward	0.3098 BTC
Revenue Gain	0.0098 BTC

POOL CANNIBALIZATION

EXAMPLE

Givens:

- We have 30 H/s. Total network is 100 H/s.
- Assume 1BTC block reward. All of the following numbers are expected value.
- 30% HR (hashrate)
  - = 30% MR (Mining Reward)
  - = 0.3 BTC

Let's say we buy more mining equipment, worth 1H/s

Honest	
Mining Reward	0.3069 BTC
Revenue Gain	0.0069 BTC

Cheat	
Mining Reward	**0.3098 BTC**
Revenue Gain	**0.0098 BTC**

More profitable to cannibalize pools than mine honestly

POOL WARS

THE GAME OF LIFE

- Attack decisions resemble an iterative game
  - Two main players: Pool 1 and Pool 2
- Each iteration of the game is a case of the Prisoner's Dilemma
  - Choose between attacking or not attacking

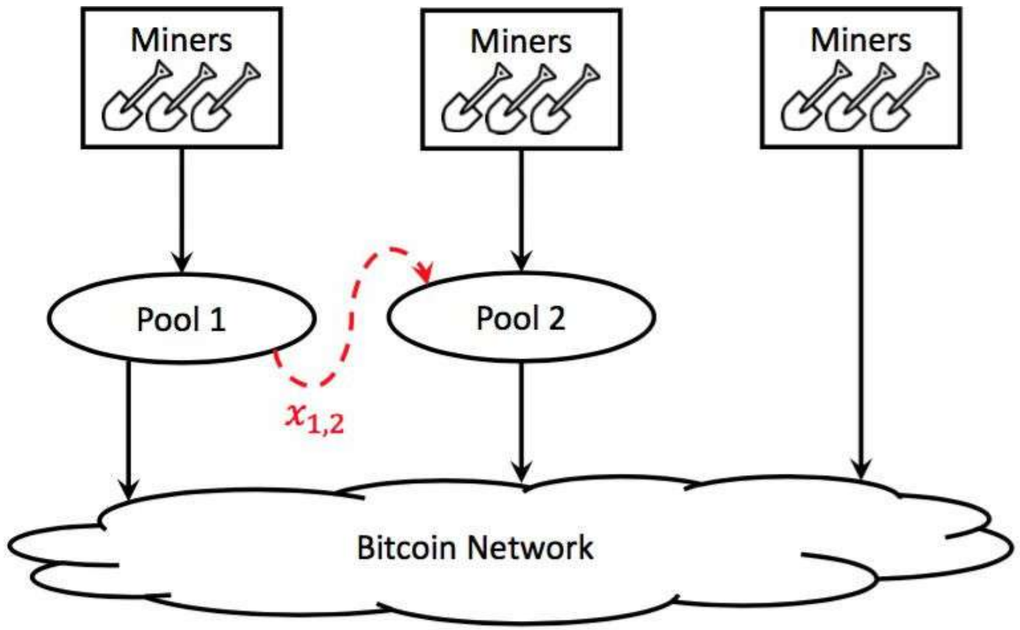


Fig. 3. The one-attacker scenario. Pool 1 attacks pool 2.

Prisoner’s Dilemma

		PRISONER 2	
		Confess	Lie
PRISONER 1	Confess	-8 , -8	0 , -10
	Lie	-10 , 0	-1 , -1

POOL WARS

NASH EQUILIBRIUM

- If Pool 1chooses to attack Pool 2, Pool 1 gains revenue, Pool 2 loses revenue
  - Pool 2 can retaliate by attacking Pool 1 and gaining more revenue
- Thus, attacking is the dominant strategy in each iteration
  - Therefore if both Pool 1and Pool 2 attack each other, they will be at a Nash Equilibrium
  - **Both will earn less than they would have if neither of them attacked.**

Pool 2 \ Pool 1	no attack	attack
no attack	$(r_1 = 1, r_2 = 1)$	$(r_1 > 1, r_2 = \tilde{r}_2 < 1)$
attack	$(r_1 = \tilde{r}_1 < 1, r_2 > 1)$	$(\tilde{r}_1 < r_1 < 1, \tilde{r}_2 < r_2 < 1)$

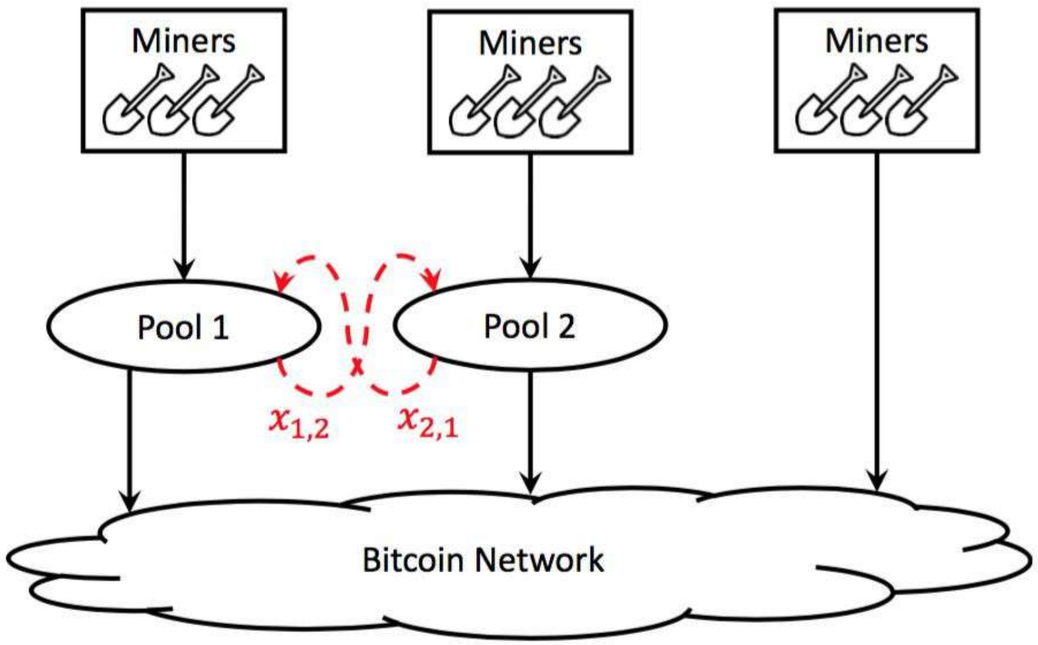


Fig. 7. Two pools attacking each other.

POOL WARS

TRAGEDY OF THE COMMONS

- No-pool-attacks is not a Nash equilibrium
  - If none of the other pools attack, a pool can increase its revenue by attacking the others
- But if the pools agree not to attack, both (or all) benefit in the long run.
  - However, this is an unstable situation since on a practical level you can attack another pool anonymously
- If pools can detect attacks then maybe an optimistic long term solution is feasible

Pool 2 \ Pool 1	no attack	attack
no attack	$(r_1 = 1, r_2 = 1)$	$(r_1 > 1, r_2 = \tilde{r}_2 < 1)$
attack	$(r_1 = \tilde{r}_1 < 1, r_2 > 1)$	$(\tilde{r}_1 < r_1 < 1, \tilde{r}_2 < r_2 < 1)$

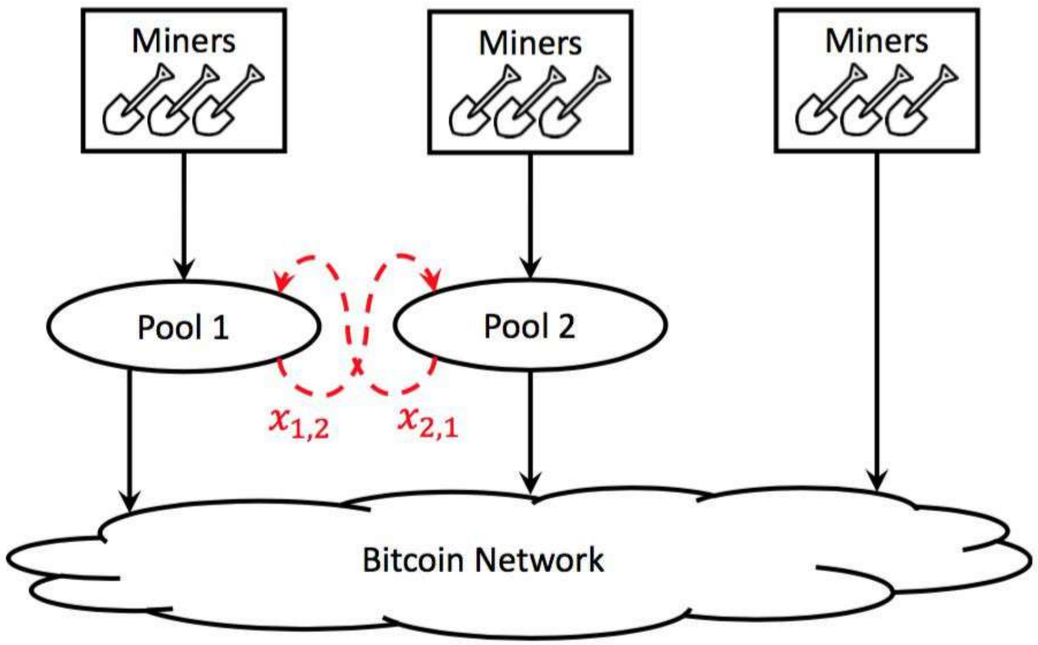


Fig. 7. Two pools attacking each other.

Nash Equilibrium is a Tragedy of the Commons

2

FORKING & DOUBLE SPENDS

DOUBLE SPENDING

THE CLASSIC ATTACK

Double Spend: Successfully spending the <u>same</u> value more than once.

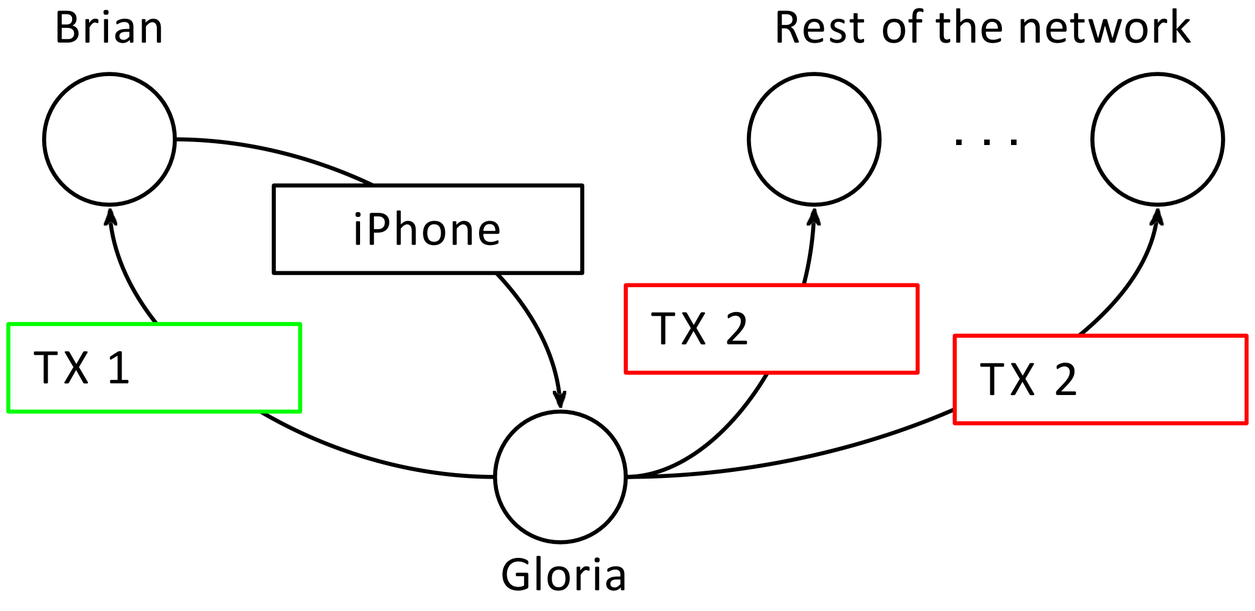
- In the year 2099, Gloria wants to buy an iPhone 92XCS from Brian on the black market for 100 BTC but doesn't want to give up her bitcoins. #HODL
  - How can Gloria double spend on Brian?

DOUBLE SPENDING

RACE ATTACK

Suppose Brian simply checks that the transaction he sees is valid and **immediately** sends Gloria the iPhone. Brian is vulnerable to a **Race Attack**!

How can we stop this?



TX 1

Gloria →Brian: 100 BTC

TX 2 → Included in next block

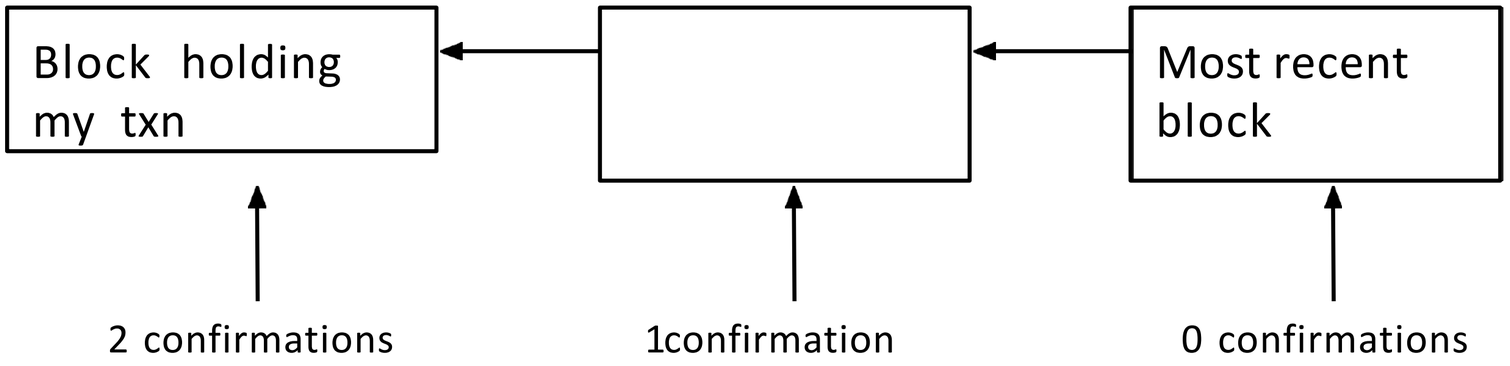
Gloria →Gloria: 100 BTC

Other address controlled by Gloria

DOUBLE SPENDING

CONFIRMATIONS

Confirmations: The number of blocks created on top of the block a txn is in.



DOUBLE SPENDING

k CONFIRMATIONS

Clearly not secure if Brian doesn't wait for any confirmations…

What if Brian waits for **k** confirmations?

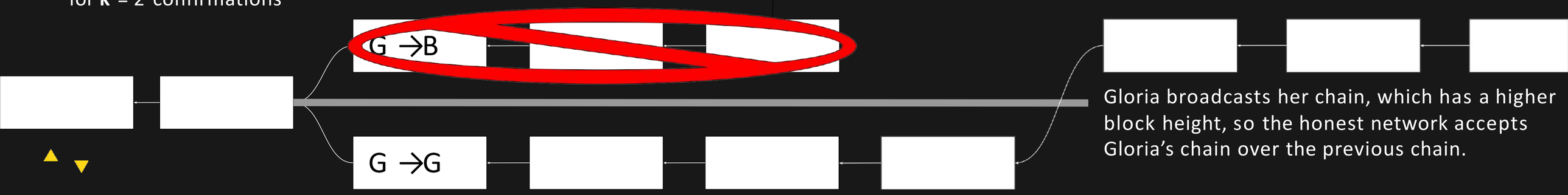
[G →B] transaction needs **k** confirmations before Brian sends the goods. In order to double spend on Brian, Gloria needs to start a private chain containing her malicious transaction, mine **k** blocks on top of her block, then broadcast *after* Brian sends the goods.

Suppose Brian waits for **k** = 2 confirmations

Honest Network

k = 2 confs, Brian sends goods

Now, Gloria has her money back, even though Brian already sent the goods!



Gloria broadcasts her chain, which has a higher block height, so the honest network accepts Gloria's chain over the previous chain.

Gloria mining blocks, withholding from honest network

"CLICKER" QUESTION:

CONFIRMATIONS

How many confirmations should I wait for to be sure my transaction is "valid"?

A) 0

B) 1

C) 2

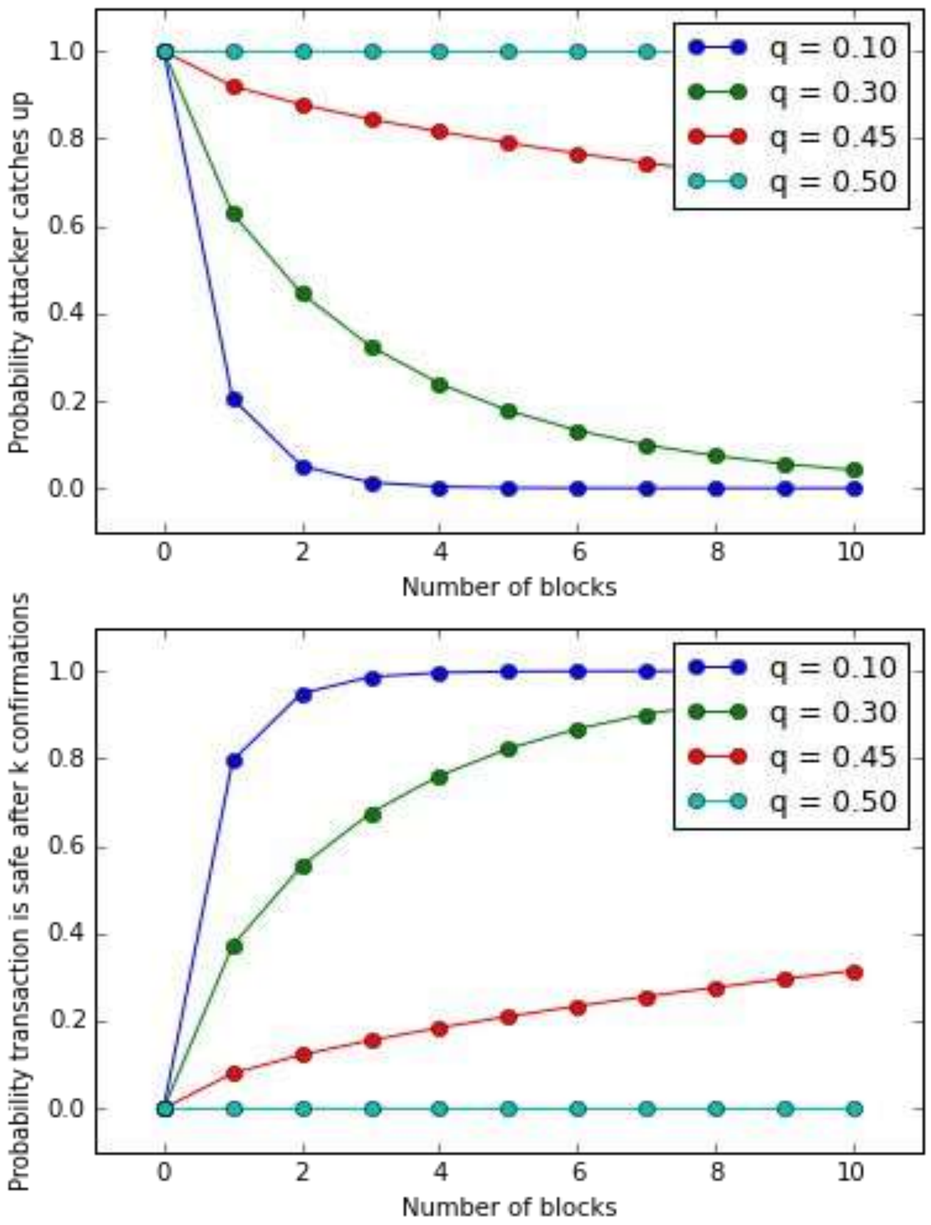
D) none of these

DOUBLE SPENDING

PROBABILITIES

Probabilities of success for the attacker

(Inverse represents bounds of the probability of safety for the vendor given assumptions of the attacker's hashpower)



DOUBLE SPENDING

51% ATTACK

What if Gloria controls more than 50% of the total network hash power?

Whenever Gloria's chain is behind the honest network's chain, she will *always* (in expectation) be able to catch up and out-produce the honest miners.

Therefore, the probability that Gloria can successfully **double spend** with >50% hash power reaches 100%!

DOUBLE SPENDING

INCENTIVES

Why would Gloria not want to double spend?

If the rest of the network detects the double spend, it is assumed that confidence in the cryptocurrency and exchange rate would *plummet*.

Bribing Miners:

Gloria might not physically control the mining hardware necessary to perform a double spend.

Instead, Gloria can bribe miners or even entire pools to mine on her withheld chain.

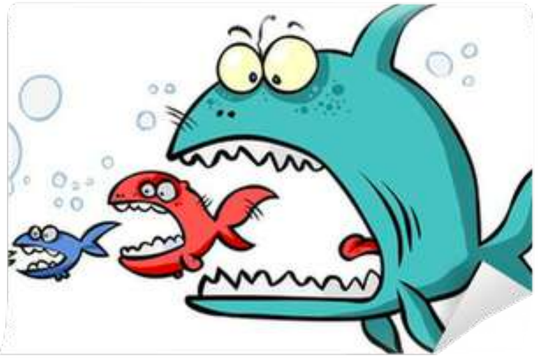
DOUBLE SPENDING

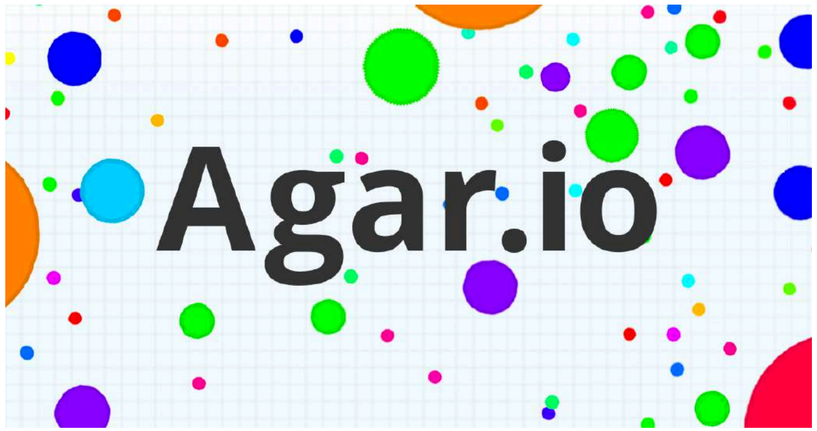
What if Gloria is a **hostile government /adversarial altcoin /large finance institution** with *significant capital* available?

Gloria can acquire enough mining ASICs or bribe enough miners / pools to achieve >50% effective hash power.

Gloria can perform a so-called **"Goldfinger" attack** with the objective of *destroying* the target cryptocurrency, either by destroying confidence in the currency with a double spend or spamming the network with empty blocks. If Gloria isn't staked in Bitcoin she can *short* the currency to profit after her attempted double spend.

Ex: Eligius pool kills CoiledCoin altcoin





DOUBLE SPENDING

CONFIRMATIONS

If I'm buying a coffee with bitcoin, should I have to wait 6 confirmations (approx 1hr) before I can get my coffee? What if I'm buying a house?

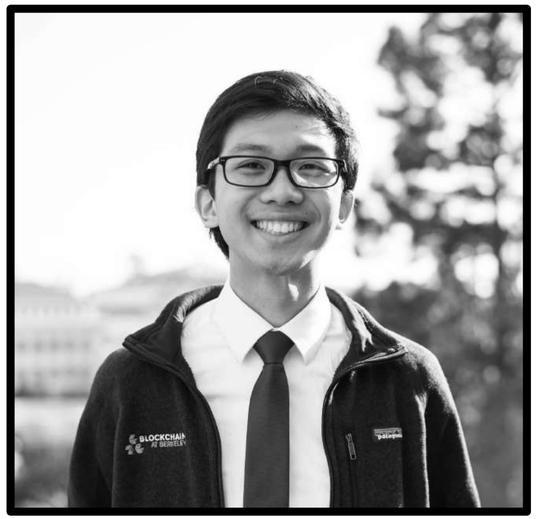
3 CENSORSHIP

BLACKLISTING

FREEZE PEACH IS DEAD

Say Gloria is a government, or has control over a government, that has jurisdiction over mining pools. (The Glorian nation)
In addition, Gloria's mining pools have **over 51% of the network's hashpower.**

Objective: Censor the Bitcoin addresses owned by certain people, say Brian, and prevent them from spending any of their Bitcoin



Block containing transactions from Brian

Normal block

Block mined by Gloria

BLACKLISTING

NAIVE CENSORSHIP STRATEGY

First strategy:
Gloria tells her mining pools not to include Brian's transactions **(blacklisting)**

Does this strategy actually work?

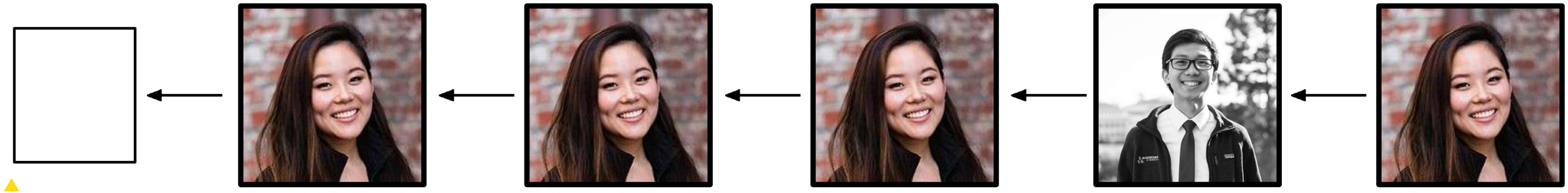
BLACKLISTING

NAIVE CENSORSHIP STRATEGY

First strategy:

Gloria tells her mining pools not to include Brian's transactions **(blacklisting)**

- Doesn't work unless you are 100% of the network
- Other miners will eventually include Brian's transactions in a block
- Can only cause delays and inconveniences

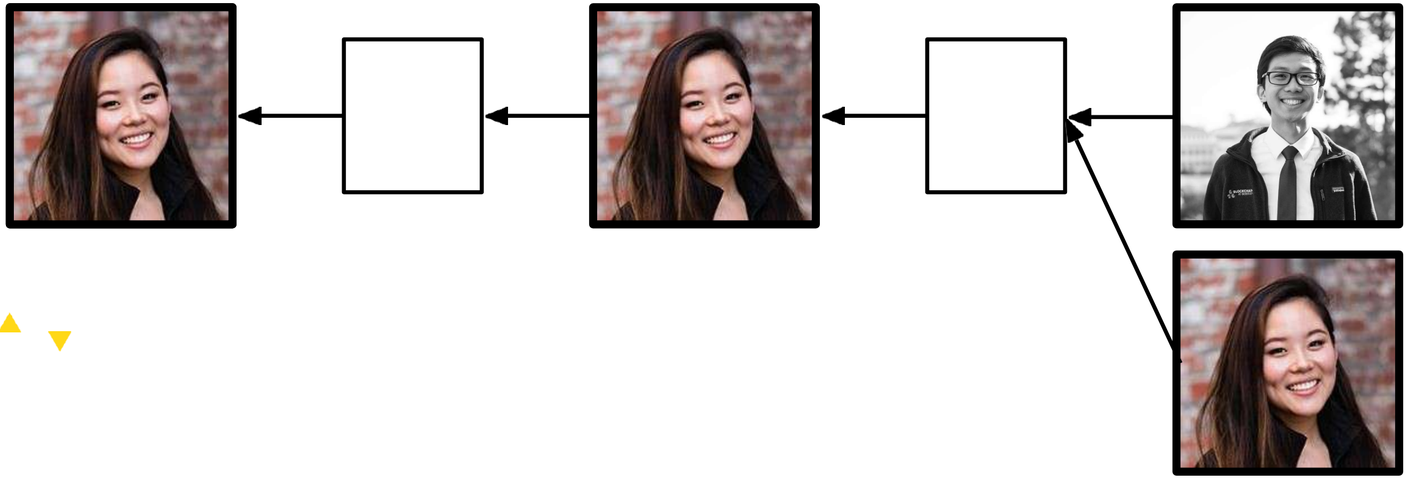


BLACKLISTING

PUNITIVE FORKING

Second strategy:

- Remember you are Gloria: **you have >51% of the network hashrate**
- Mandate that Glorian pools will refuse to work on a chain containing transactions spending from Brian's address
- Announce this to the world



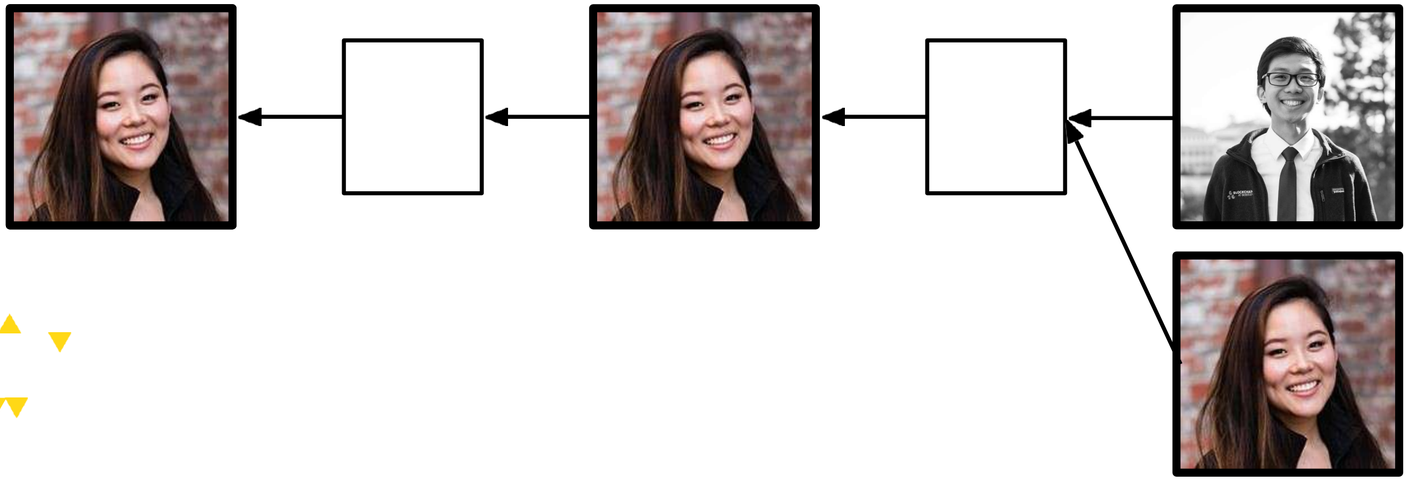
Does this strategy actually work?

BLACKLISTING

PUNITIVE FORKING

Second strategy:

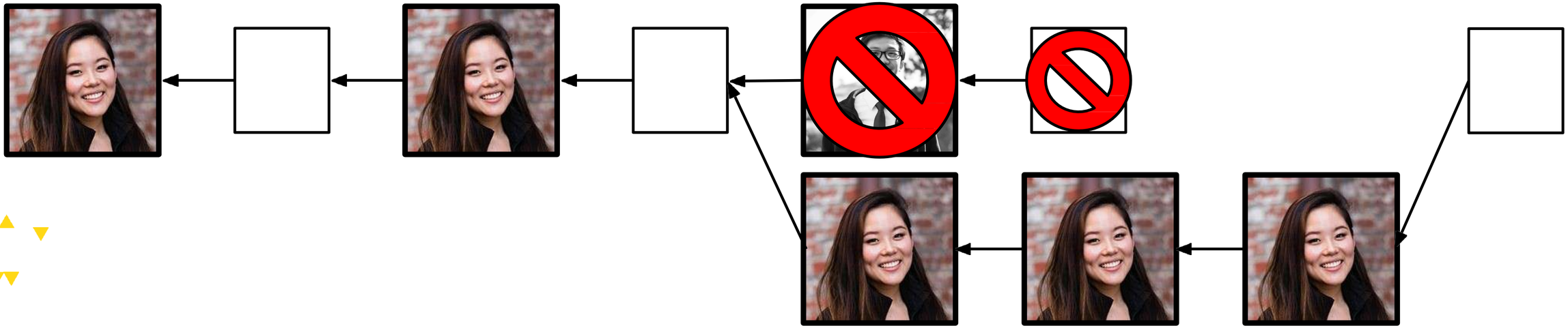
- Remember you are Gloria: **you have >51% of the network hashrate**
- Mandate that Glorian pools will refuse to work on a chain containing transactions spending from Brian's address
- Announce this to the world



BLACKLISTING

PUNITIVE FORKING

- If miners include a transaction from Brian in a block, Gloria will fork and create a longer proof-of-work chain
- Block containing Brian's transaction now invalidated, can never be published

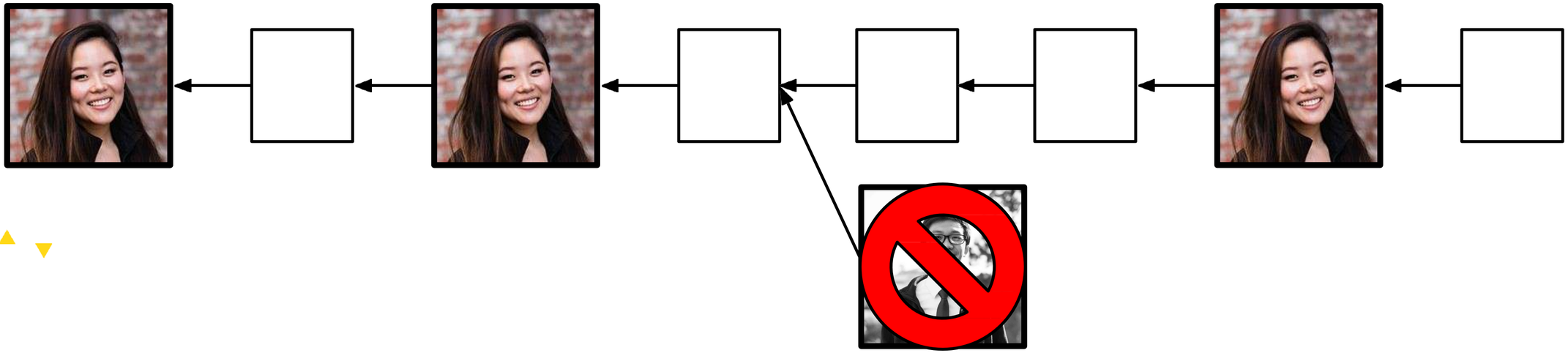


BLACKLISTING

PUNITIVE FORKING

- Non-Glorian miners eventually stop trying to include Brian's transactions when mining blocks, since they know that their block will be invalidated by Glorian miners when they do

We have now shown how a 51% majority can prevent anyone from accessing their funds. This is called **punitive forking.**

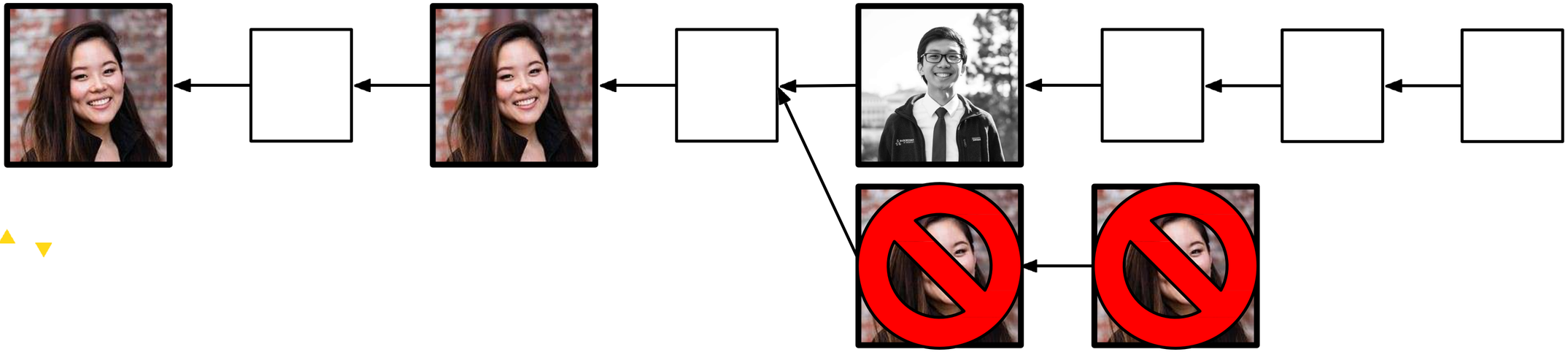


BLACKLISTING

FEATHER FORKING

Punitive forking doesn't work unless Gloria has >51% of hashpower. Is there another way? Yes! Called **Feather Forking**

- New strategy: Gloria announces that she will **attempt** to fork if she sees a block from Brian, but she will give up after a while
  - As opposed to attempting to fork forever; doesn't work without >51%
- Ex. Give up after block with Brian's tx contains **k** confirmations



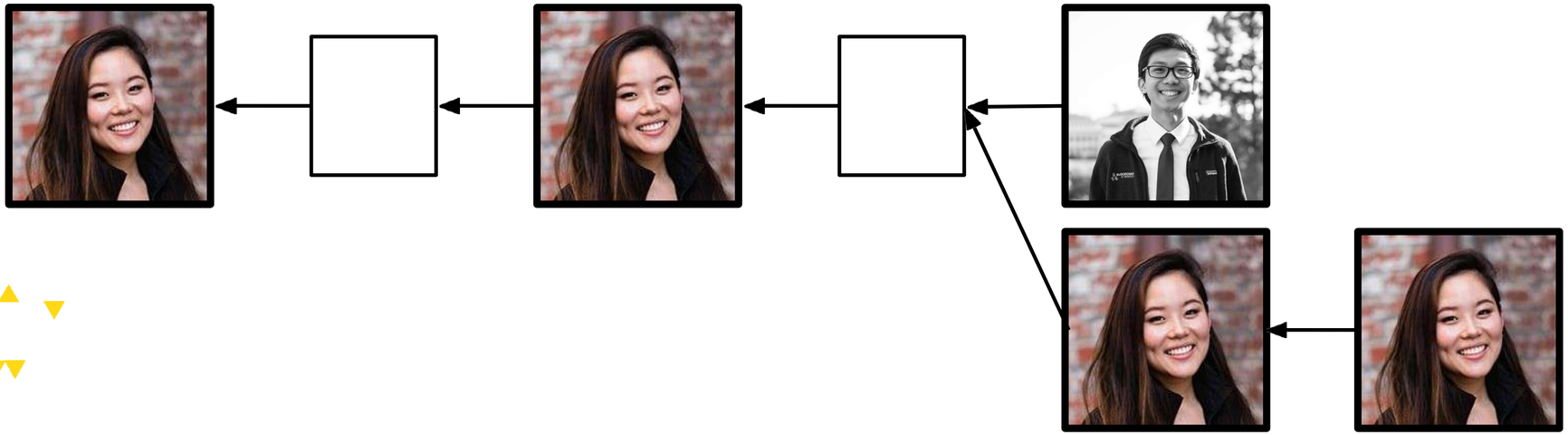
BLACKLISTING

FEATHER FORKING

Let **q** equal the proportion of mining power Gloria have, 0 < **q** < 1
Let **k** = 1:Gloria will give up after 1confirmation (one additional block)

- Chance of successfully orphaning (invalidating) the Brian block = $\mathbf{q^2}$(Math omitted)

If **q = .2**, then $\mathbf{q^2}$ **= 4%** chance of orphaning block. Not very good



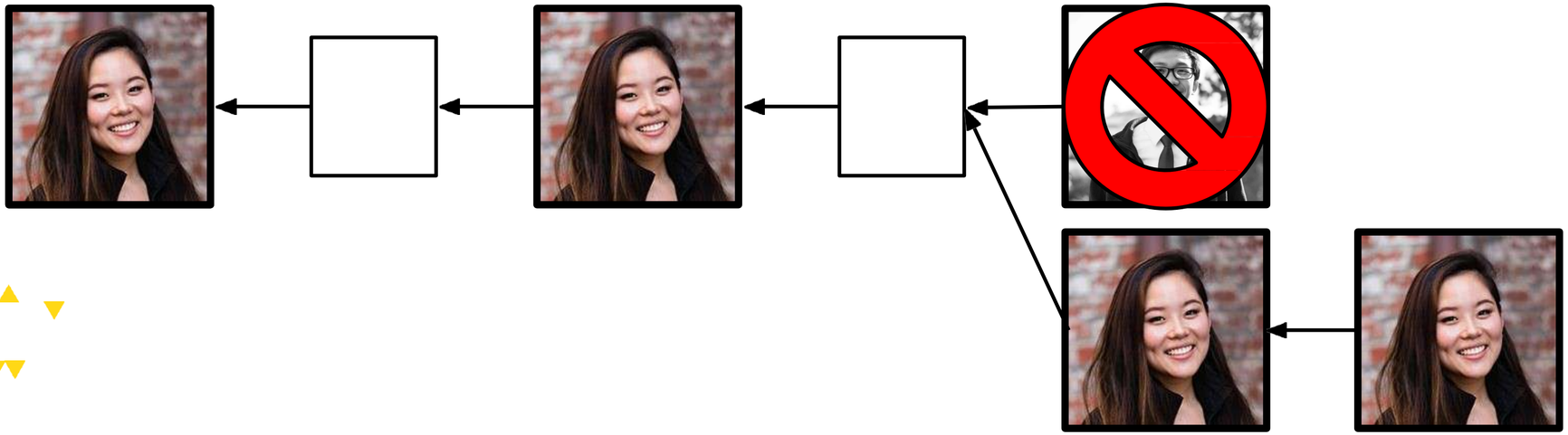
BLACKLISTING

FEATHER FORKING

But other miners are now aware that their block has a $\mathbf{q^2}$ chance of being orphaned. They must now decide whether they should include Brian's tx in their block

EV(include) = (1- $\mathbf{q^2}$) *BlockReward + Brian's tx fee

EV(don't include) = BlockReward



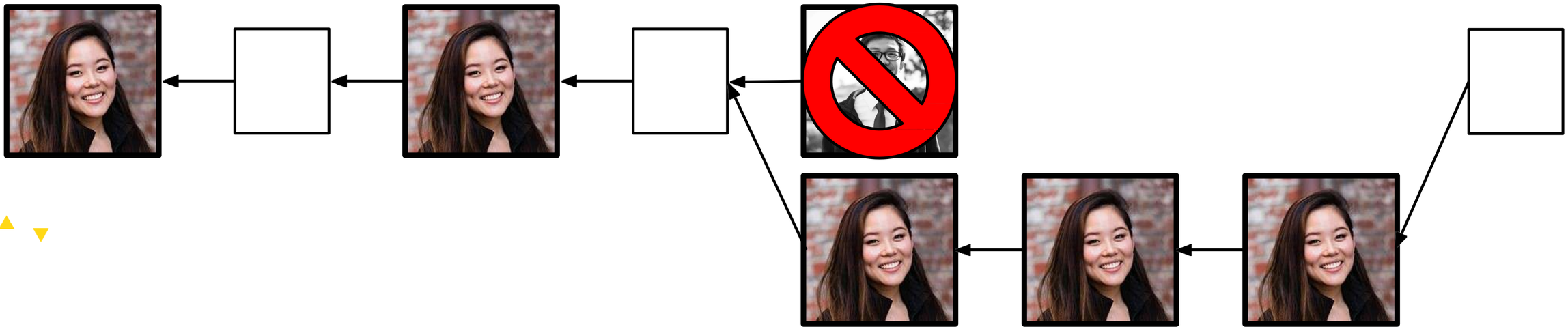
BLACKLISTING

FEATHER FORKING

EV(include) = (1- $\mathbf{q^2}$) *BlockReward + Brian's tx fee
EV(don't include) = BlockReward

Therefore, unless Brian pays $\mathbf{q^2}$ *BlockReward in fees for his transaction, other miners will mine on the malicious chain

- 4% *12.5 BTC = 0.5 BTC = Brian must pay **~$5000** minimum/transaction



4

SELFISH MINING

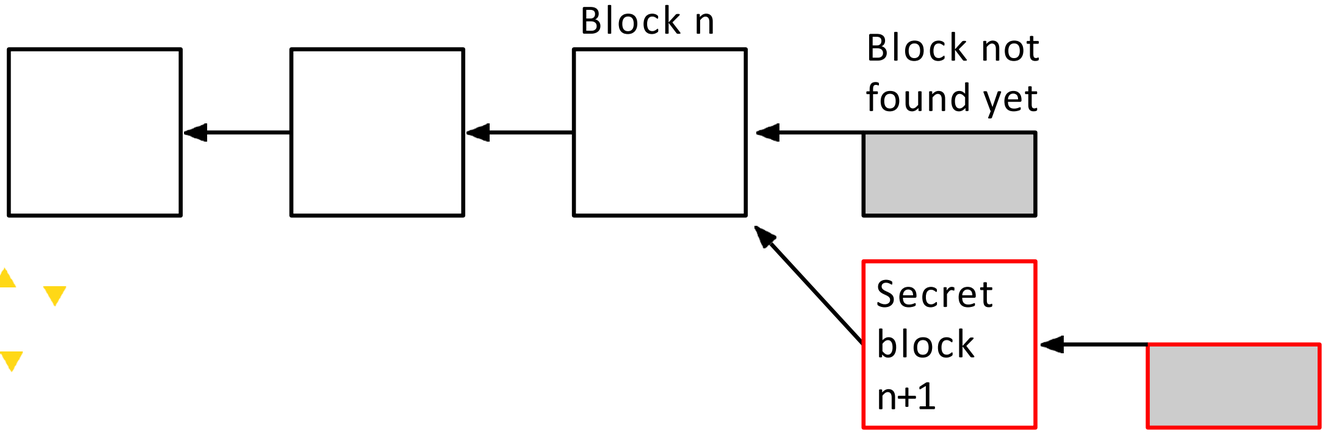
SELFISH MINING

BLOCK WITHHOLDING

You are a miner; suppose you have just found a block.

- Instead of announcing block to the network and receiving reward, keep it secret
- Try to find two blocks in a row before the network finds the next one

This is called **selfish mining** or **block-withholding**



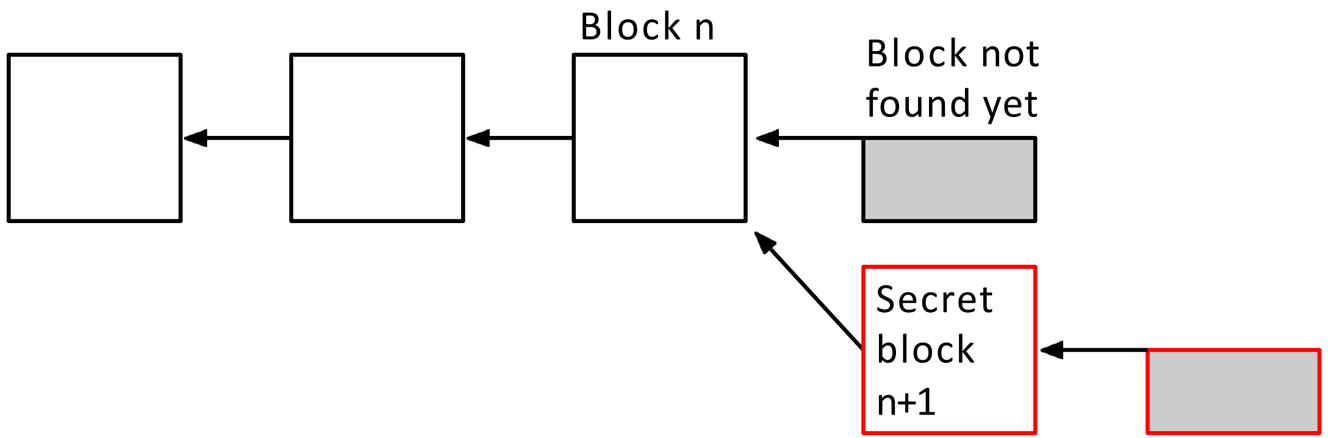
Note: "block-withholding" is also sometimes used in the context of mining pools - submitting shares but withholding valid blocks

SELFISH MINING

BLOCK WITHHOLDING

If you succeed in finding a second block, you have fooled the network

- Network still believes it is mining on the longest proof of work chain
- You continue to mine on your own chain

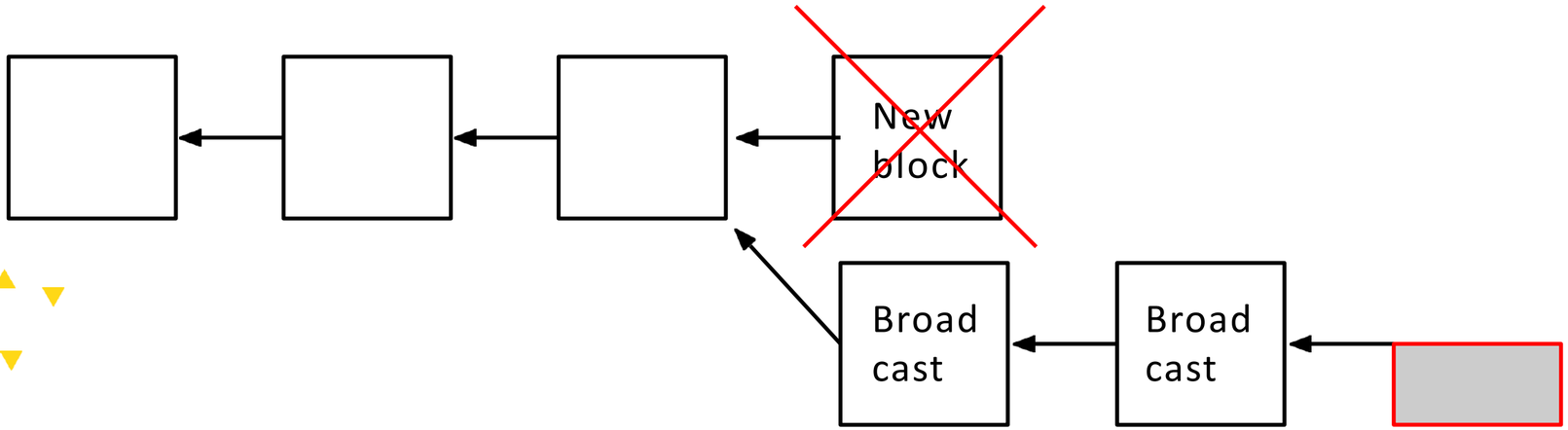


SELFISH MINING

BLOCK WITHHOLDING

If the network finds a block, you broadcast your two secret blocks and make the network block invalid

- While network was working on the invalid block, you got a bunch of time to mine by yourself... for free!
- Free time mining on network
  => higher effective proportion of hashrate **=> higher expected profits!**

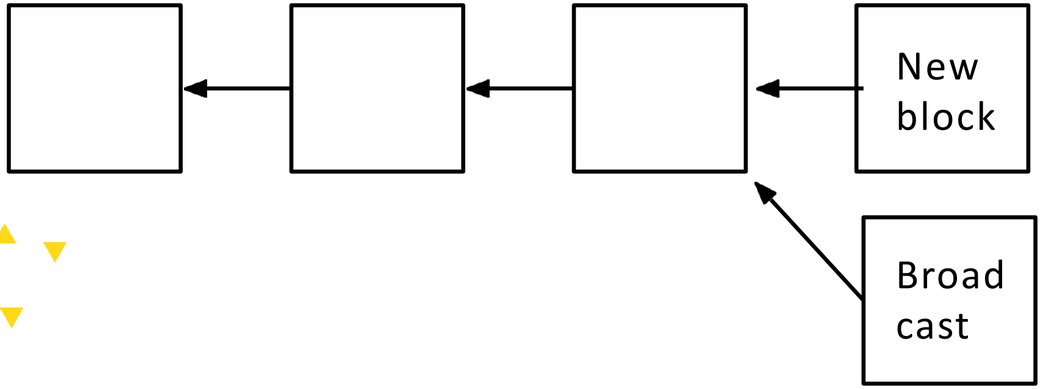


SELFISH MINING

BLOCK WITHHOLDING

But what if the network found their new block before you could find a second one?
Race to propagate!

- If on average you manage to tell 50% of the network about your block first:
  - Malicious strategy is more profitable if you have >25% mining power
- If you have >33% mining power, **you can lose the race every time and malicious strategy is still more profitable!**
  - (actual math omitted due to complexity)



5

DEFENSE DISCUSSION

PROBLEM STATEMENT

USE SIGNATURES DUMMY

How do we prevent selfish mining?

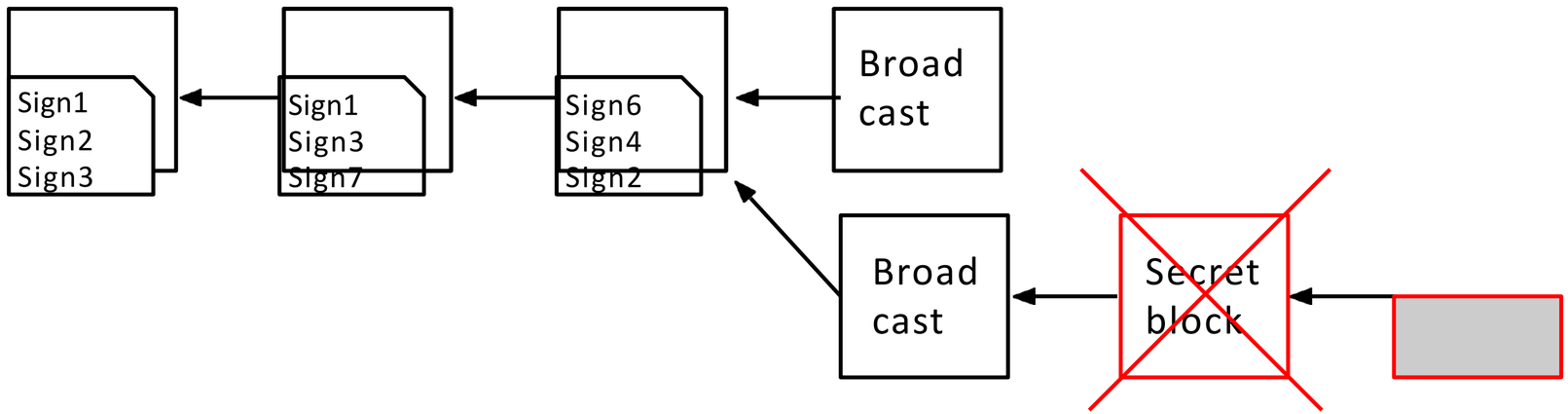
Hint: Use signatures

BLOCK VALIDATION

DUMMY BLOCK SIGNATURES

Proposed by Schultz (2015), Solat and Potop-Butucaru (2016)

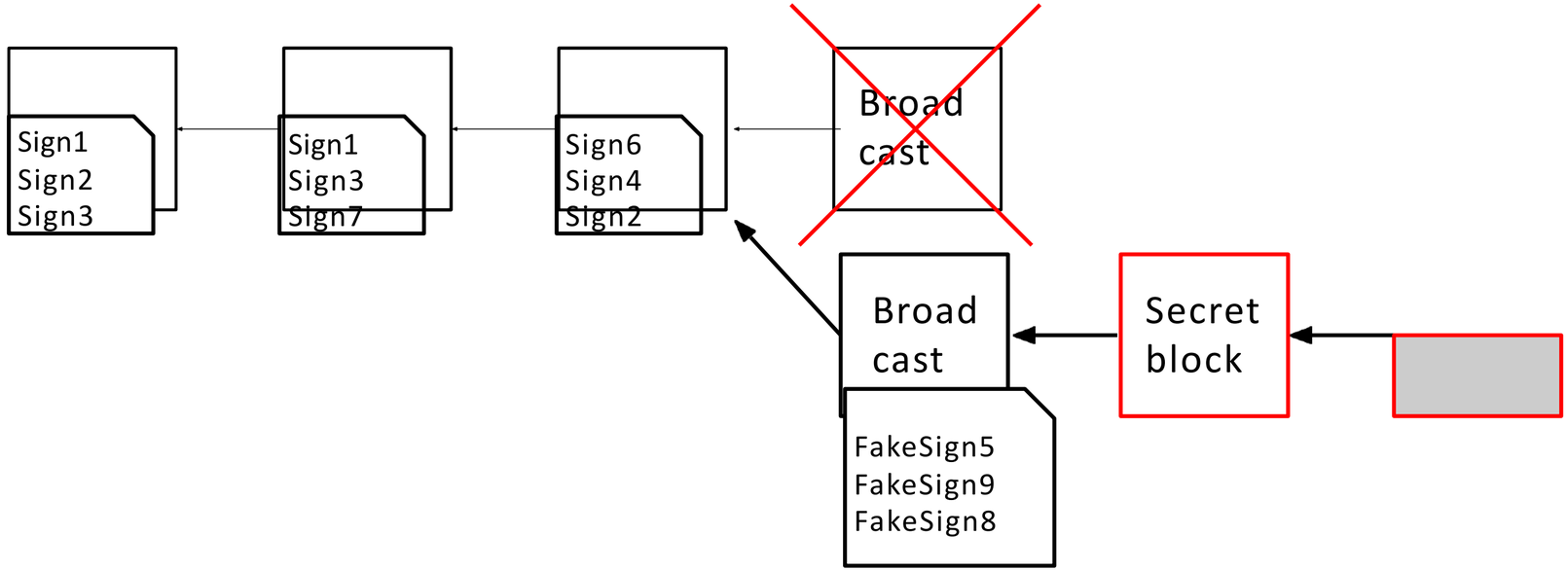
- Accompany solved blocks with signatures on dummy blocks
- Proves that the block is witnessed by the network
  - Proves that a competing block is absent before miners are able to work on it



PROBLEM STATEMENT

THE PROBLEM WITH DUMMYBLOCKS

- Doesn't provide a mechanism to evaluate whether the **number of proofs** is adequate
- Does not discuss how to prevent **Sybil attacks** on signatures
  - Selfish miner generates many signatures on the dummy block
- This defense requires **fundamental changes** to the block validity rules



PROBLEM STATEMENT

How do we prevent selfish mining?

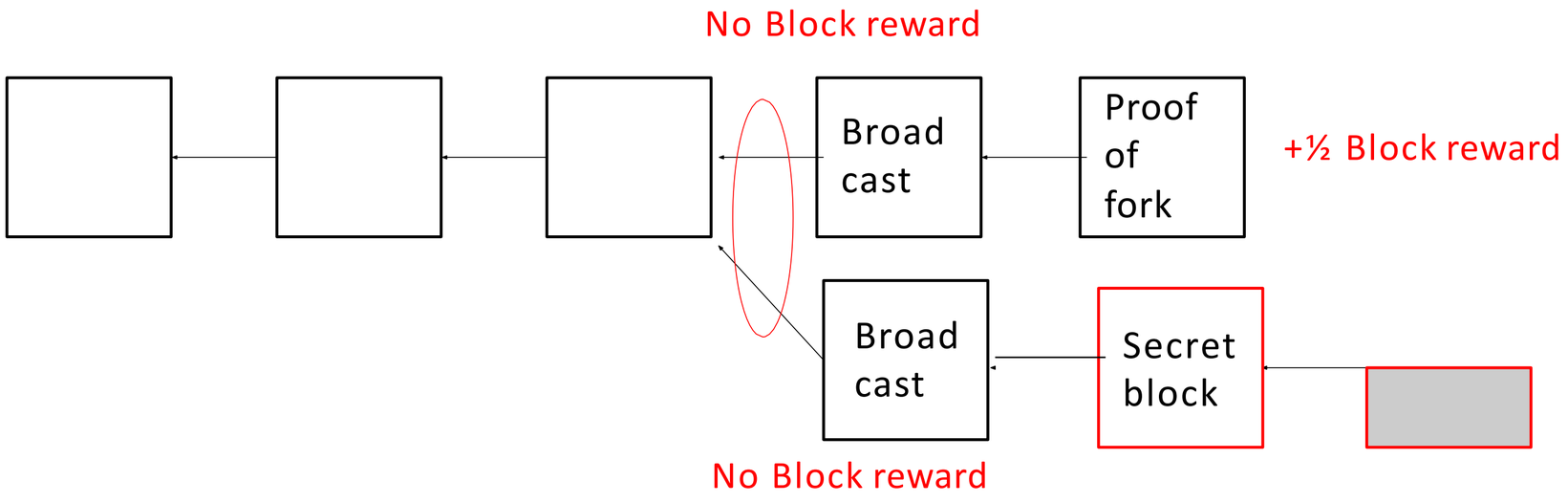
Hint: Use block rewards

FORK-PUNISHMENT

FORK-PUNISHMENT RULE

Proposed by Lear Bahack (2013)

- Competing blocks receive no block reward
- The first miner who incorporates a proof of the block fork in the blockchain gets half of the forfeited rewards



FORK-PUNISHMENT

FORK-PUNISHMENT RULE

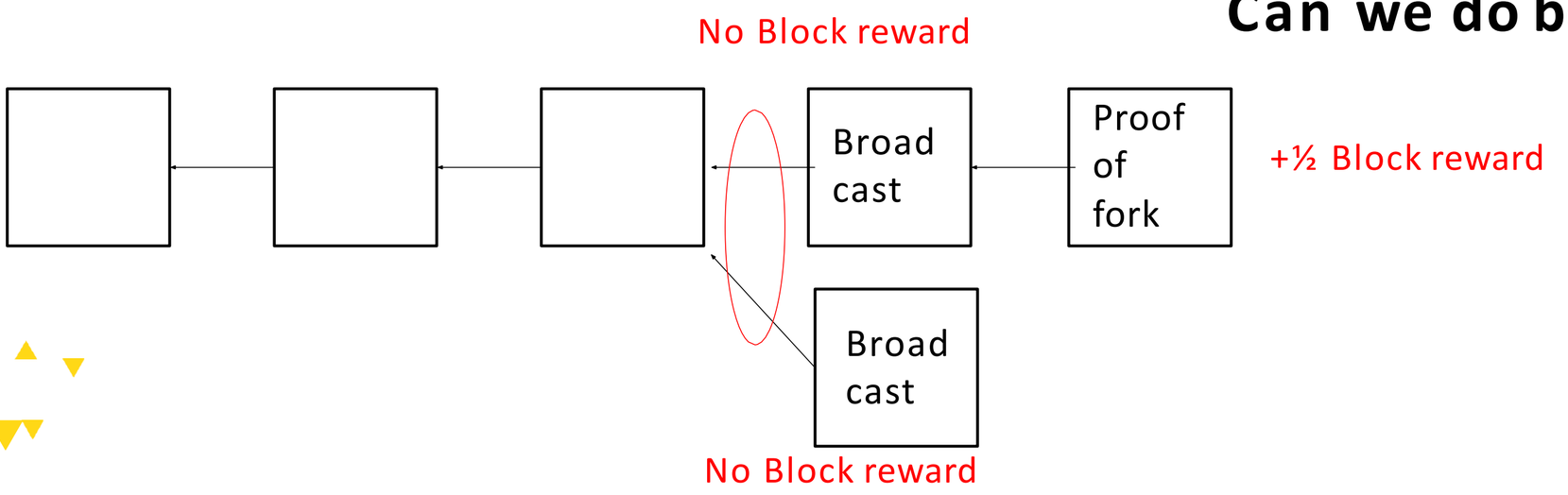
Drawbacks

- Honest miners suffer **collateral damage** of this defense
  - And lead to more attacks...

This defense requires **fundamental changes** to the reward distribution rules

- Requires a hard fork to implement
  - We have hard enough time fixing transaction malleability

Can we do better?



THE BITCOIN NETWORK

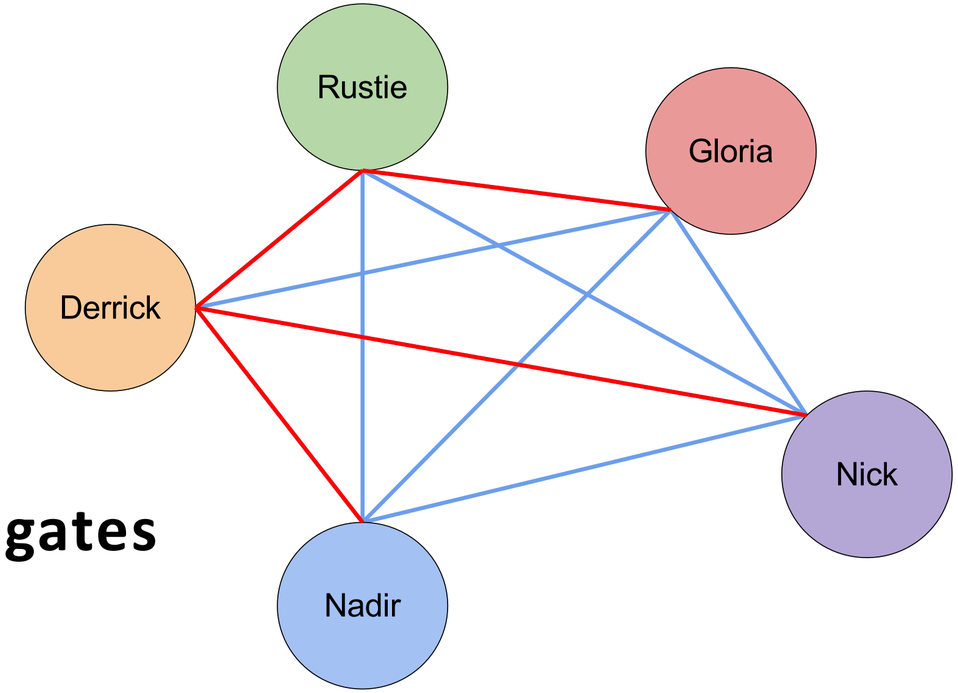
A CLOSER LOOK

What do we mean by p2p?

- Gossip protocol (flooding)
- 8 outbound connections, 117 inbound (pseudorandom) connections

Need to understand how information propagates

- Topology
- Network latency

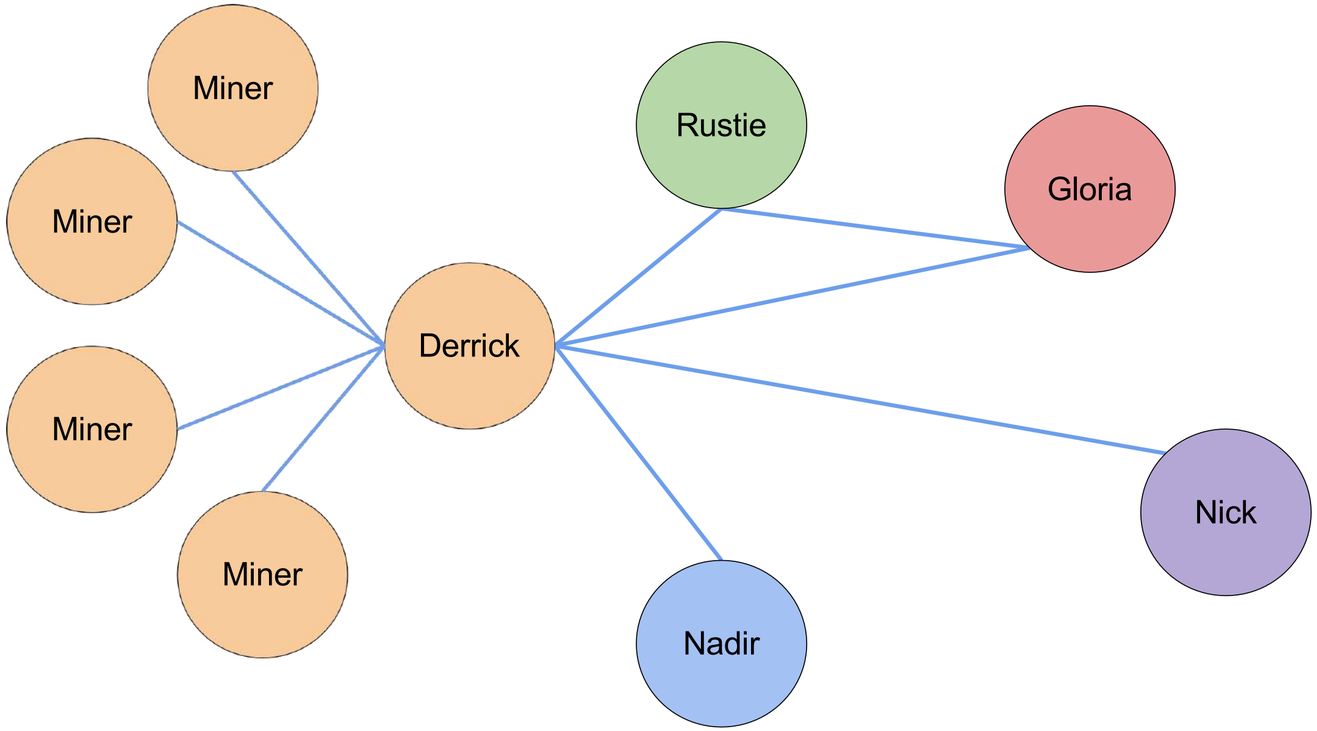


THE BITCOIN NETWORK

AN UNEVEN TOPOLOGY

Some nodes are more influential than others

- Uneven distribution of hash power
- Hard to distinguish network topology

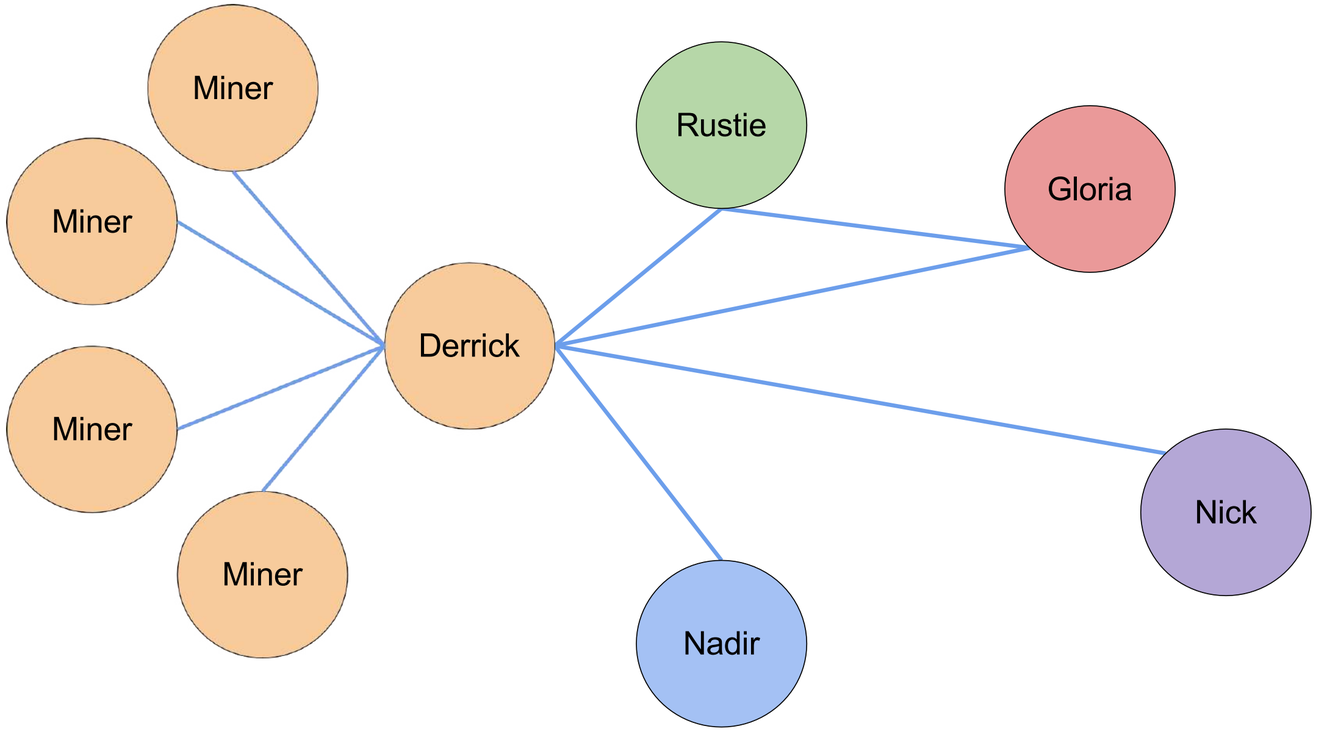


THE BITCOIN NETWORK

AN UNEVEN TOPOLOGY

Bitcoin safe as long as >50% network is honest?

- Incentive alignment
- See selfish mining
- 33%, 25%, 23.2%, 32%, …

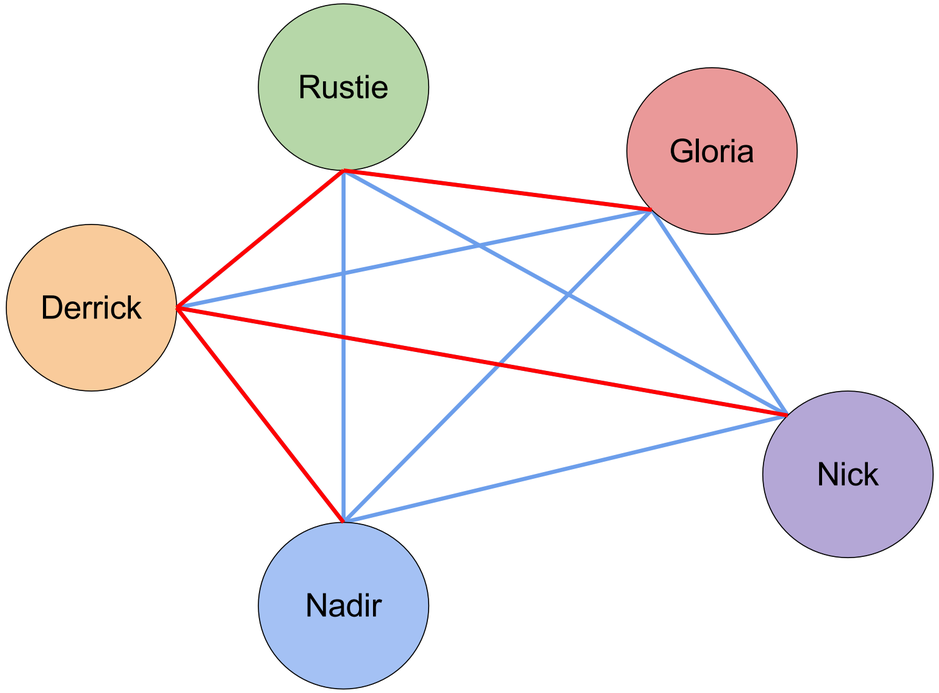


THE BITCOIN NETWORK

NETWORK LATENCY

Different propagation times

- Getting data items out to the network quickly
- Block propagation races
- Leads to disproportionate profits

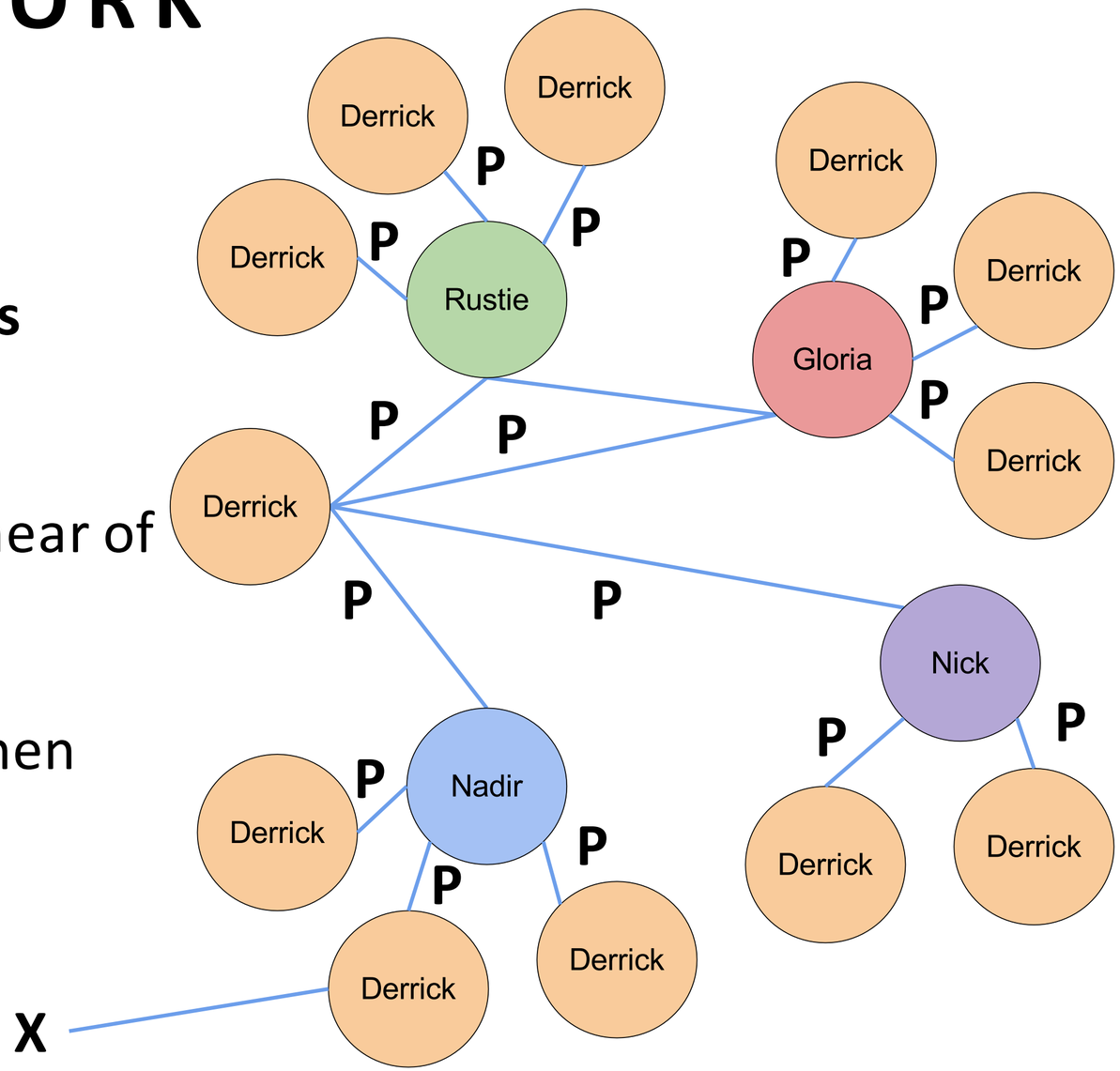


THE BITCOIN NETWORK

SYBIL ATTACK REVISITED

Flood the network with 0-power nodes

- Sybil attack on honest miners
- 0-power nodes as sensors
  - Publish secret block P when hear of competing block X
  - Ignore X, replay P
  - If P reaches miner before X, then mine on P



PROBLEM STATEMENT

How do we prevent selfish mining?

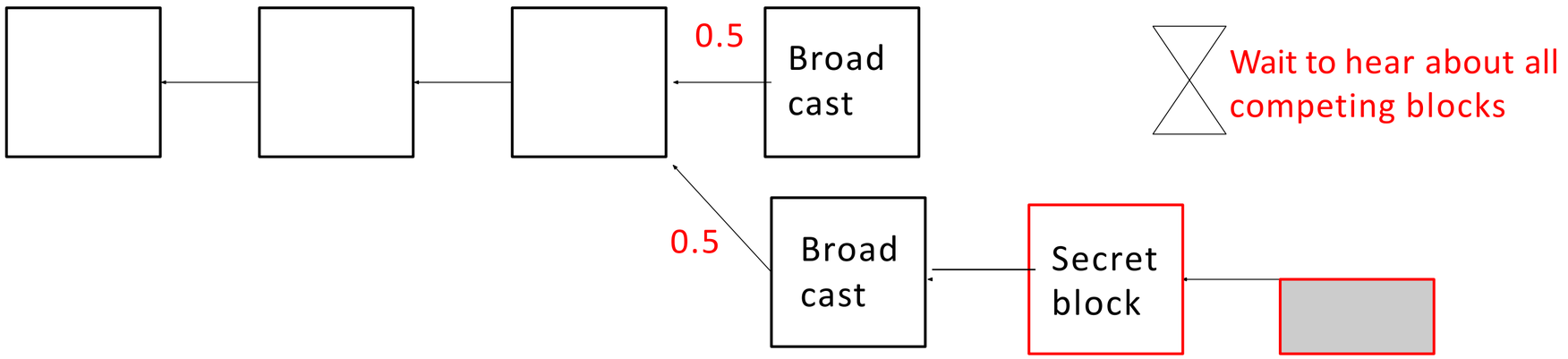
Claim: The problem is that we reward the longest chain.

TIE-BREAKING

UNIFORM TIE-BREAKING

Proposed by Eyal and Sirer (2014)

- In the case of a tie, a miner **randomly chooses** which chain to mine on
  - Prevents an attacker from benefiting from network-level dominance
- Raises the profit threshold from 0% to **25%** under their strategy
  - Sapirshtein (2015) proposes a more optimal selfish mining strategy
  - Reduces Eyal and Sirer profit threshold to **23.2%**

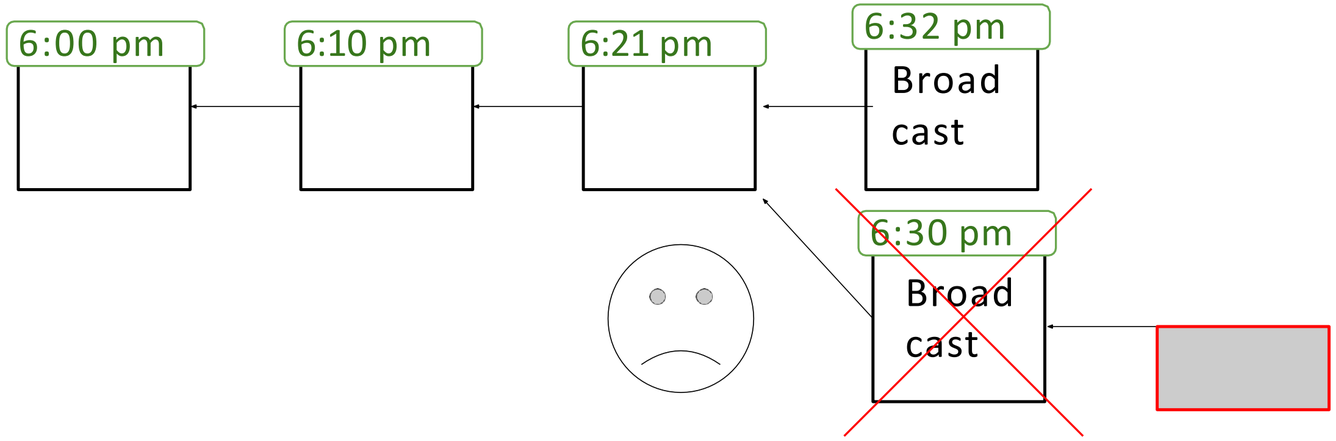


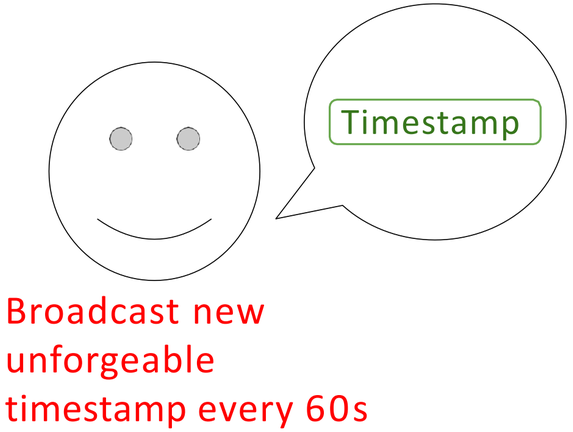
TIE-BREAKING

UNFORGETTABLE TIMESTAMPS

Proposed by Ethan Heilman (2014)

- Each miner incorporates the **latest unforgettable timestamp** issued by a **trusted party** into the working block
  - Timestamp is publically accessible and unpredictable
  - Issued with an interval of 60s
- When two competing blocks are received within 120s, a miner prefers the block whose **timestamp is "fresher"**
- Claim: Raises the profit threshold to 32%



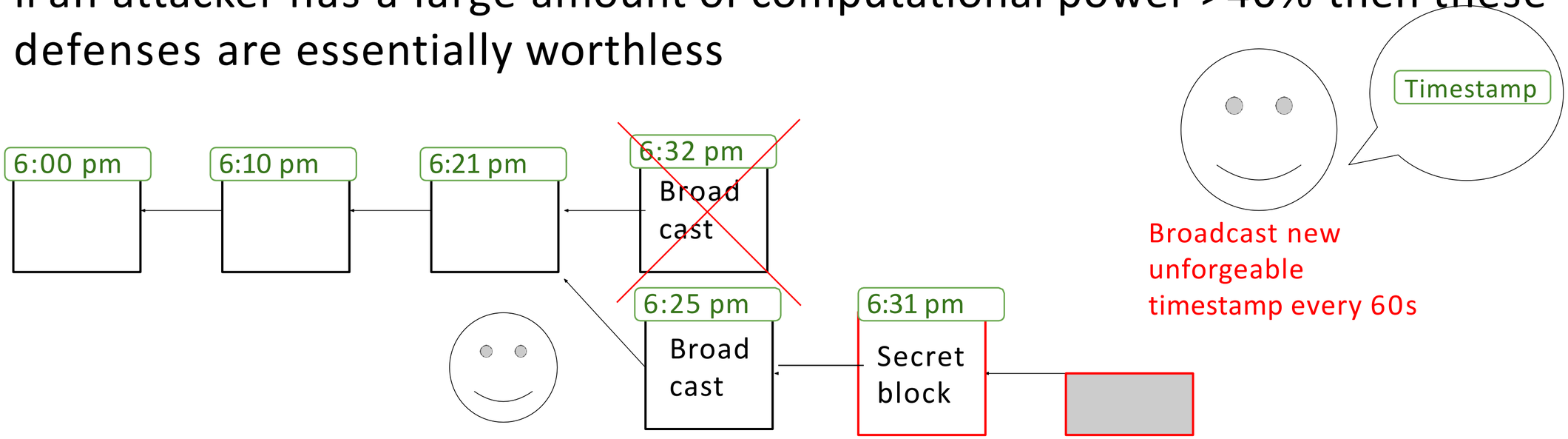


TIE-BREAKING

UNFORGETTABLE TIMESTAMPS

Drawbacks

- Tie-breaking rules don't apply when the selfish mining chain is longer than the public chain
  - **Only applies to a block propagation race**
- If an attacker has a large amount of computational power >40% then these defenses are essentially worthless



TIE-BREAKING

UNFORGETTABLE TIMESTAMPS

Drawbacks

- Introducing trusted third party contradicts the Bitcoin philosophy

+ CENTRALIZATION

PROBLEM STATEMENT

How do we disincentivize selfish mining even when the selfish miner has a longer chain?

PUBLISH OR PERISH

ZHANG AND PRENEEL, "PUBLISH OR PERISH" (2017)

Ren Zhang and Bart Preneel (Apr 2017) claim the best-yet defense of selfish mining

- Backwards compatible: No hard fork
- Disincentivizes selfish mining even when the selfish miner has a longer chain

Approach: A novel **F**ork-**R**esolving **P**olicy (FRP)

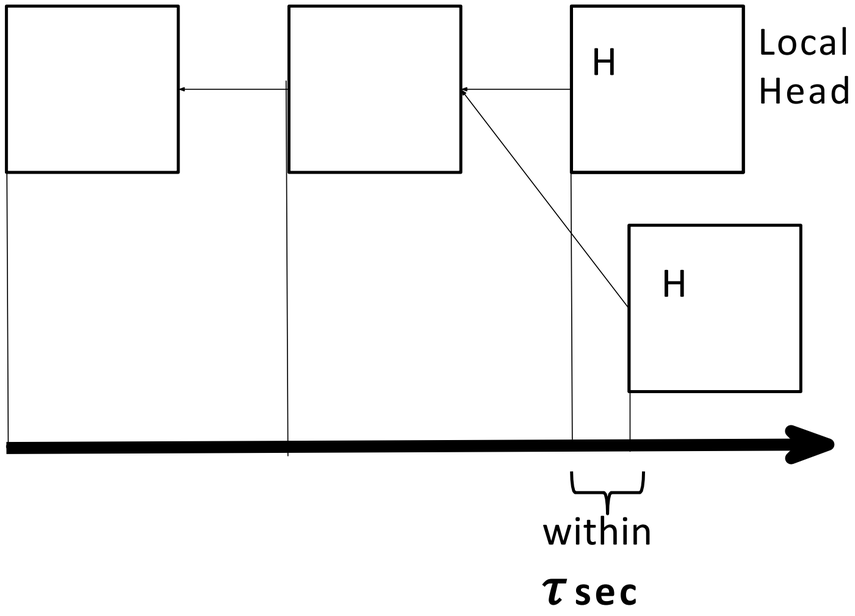
- Replace the original Bitcoin FRP (length FRP), with a **weighted FRP**
  - Embed in the working block the hashes of all its uncle blocks
- Note that selfish mining is premised on the idea of first building a secret block
- Idea: Make sure this secret block does not help the selfish miner win the block race

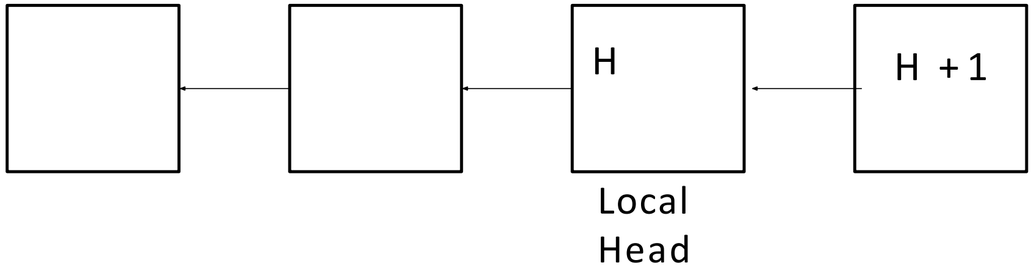
PUBLISH OR PERISH

ZHANG AND PRENEEL, "PUBLISH OR PERISH" (2017)

Definitions

- τ: An assumed upper bound on the amount of time it takes to propagate blocks across the Bitcoin network
- **In time.** Evaluated from the miner's local perspective.
  1. Height value is greater than that of the local head OR
  2. Height value is same as that of the local head, but was propagated within τ time





PUBLISH OR PERISH

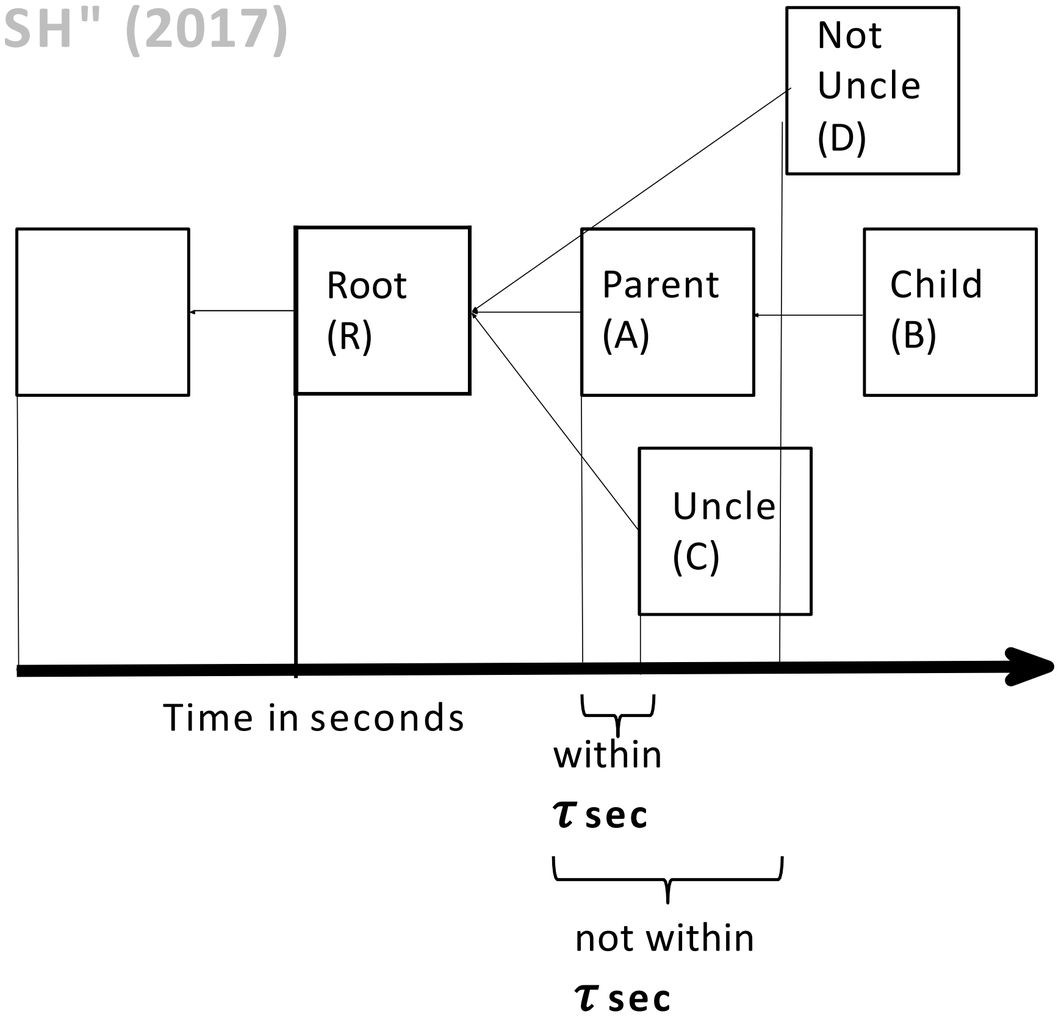
ZHANG AND PRENEEL, "PUBLISH OR PERISH" (2017)

Definitions

- **Uncle**.

  1. The uncle of a block B is one less the height of B
  2. The uncle has to be in time with B's parent

- **Weight**. Since two competing chains always have a shared root, only consider blocks after that
  - weight = # of in time blocks + # of uncle hashes embedded in these blocks



PUBLISH OR PERISH

ZHANG AND PRENEEL, "PUBLISH OR PERISH" (2017)

Zhang and Preneel's **Weighted Fork Resolving Policy**:

1. If one chain is longer *height-wise* than the other(s) by **k** or greater blocks*
   a. The miner will mine on this chain
2. Otherwise, the miner will choose the chain with the largest *weight*
3. If the largest weight is achieved by multiple chains simultaneously, then the miner chooses one among them randomly

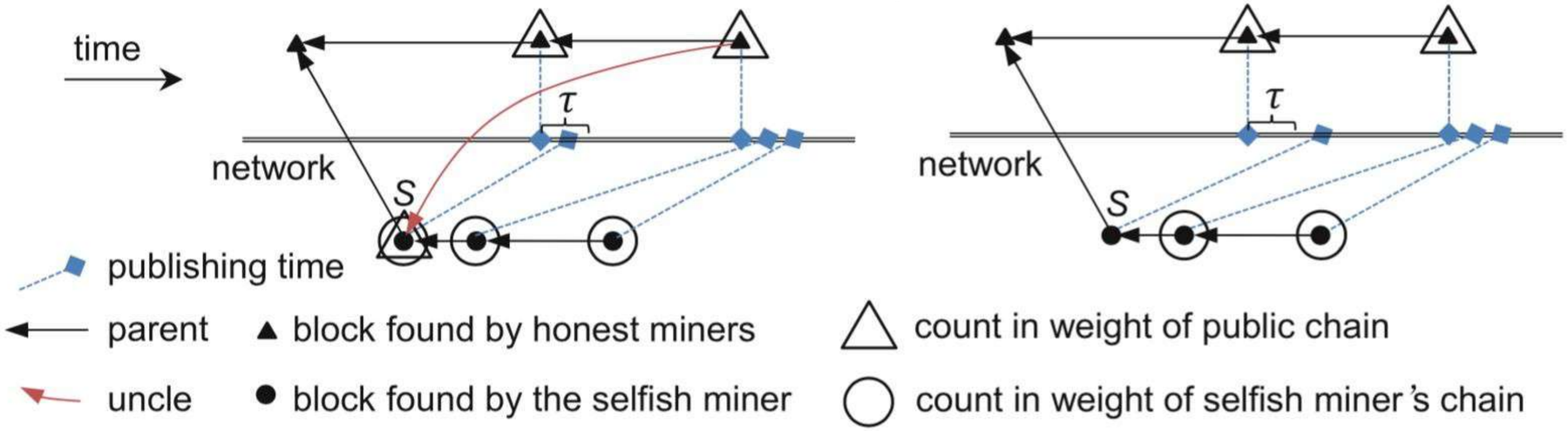
Aside: **k is a "fail-safe parameter" that gauges the allowed amount of network partition. Note that when **k** = ∞the first rule never applies.*

PUBLISH OR PERISH

ZHANG AND PRENEEL, "PUBLISH OR PERISH" (2017)

Miner has one secret block. A competing block is published. **Block race**! Miner has two options:

- Option 1: If the selfish miner **publishes** their block, *the next honest block gains a higher weight* by embedding a proof of having seen this block
- Option 2: If the selfish miner **keeps their block secret**, the secret block *does not contribute* to the weight of its own chain
- In both scenarios, the secret block does not help the selfish miner win the block race



Choice 1: Publish **Choice 2: Don't publish**

PUBLISH OR PERISH

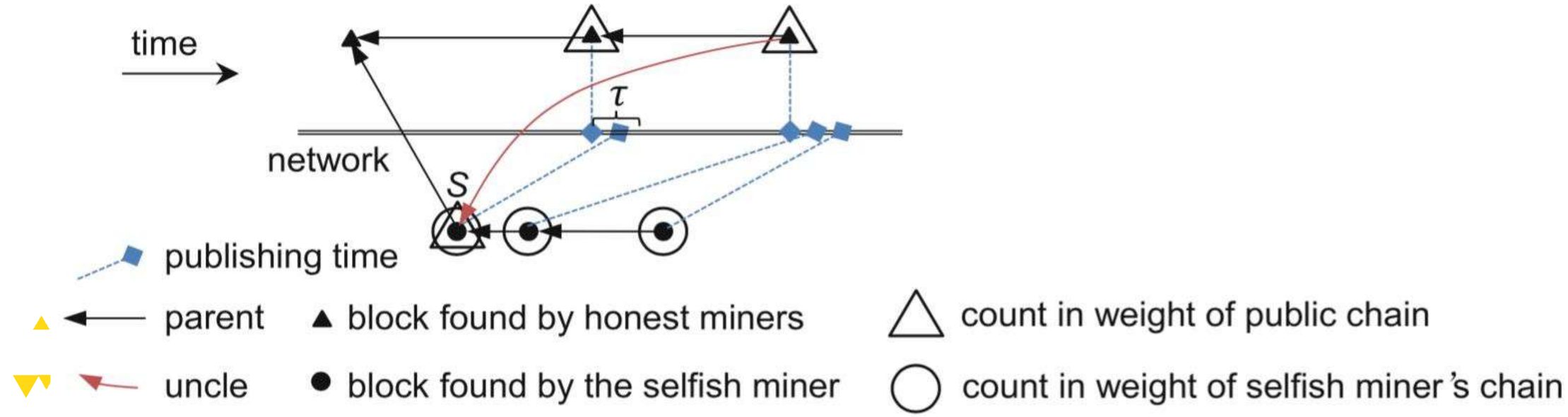
ZHANG AND PRENEEL, "PUBLISH OR PERISH" (2017)

Same scenario revisited. More rigorously, let S be the first selfish block

Option 1:Selfish miner publishes S

- S will be an uncle of the next honest block
  - (since it was published *in time* and its *height is one less*)

=> S counts into the weight of **both** the honest and the selfish chain



Choice 1: Publish

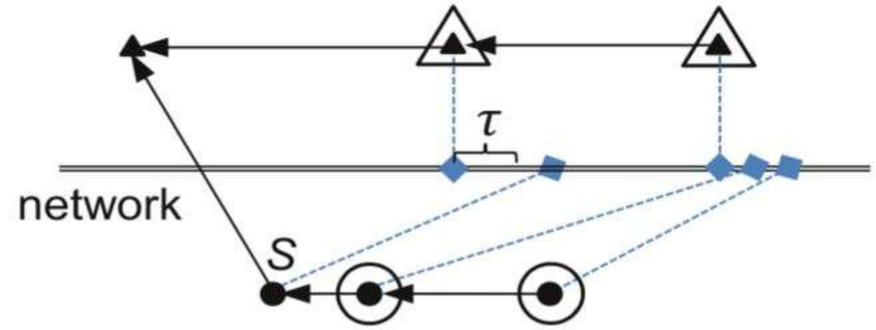
PUBLISH OR PERISH

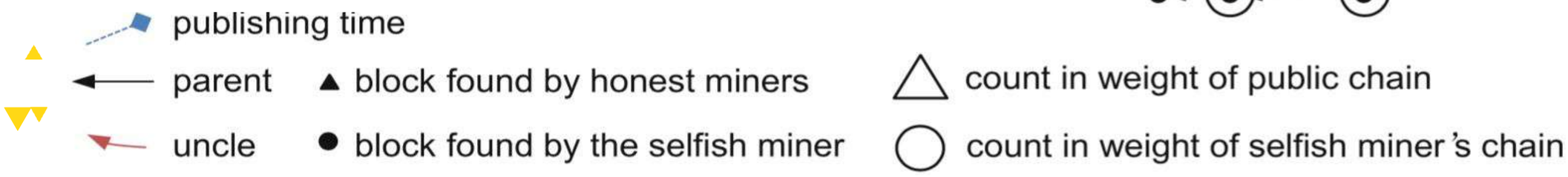
ZHANG AND PRENEEL, "PUBLISH OR PERISH" (2017)

Option 2: Selfish miner doesn't publish S

- Selfish miner waits, and publishes it later as a part of the selfish chain
- Honest miners do not count S into the weight of the selfish chain because S is not *in time.*
  - It is a **late block**
- S is not an *uncle* of the next honest block because the honest miners did not see it

=> S contributes to **neither** the weight of the honest nor the selfish chain





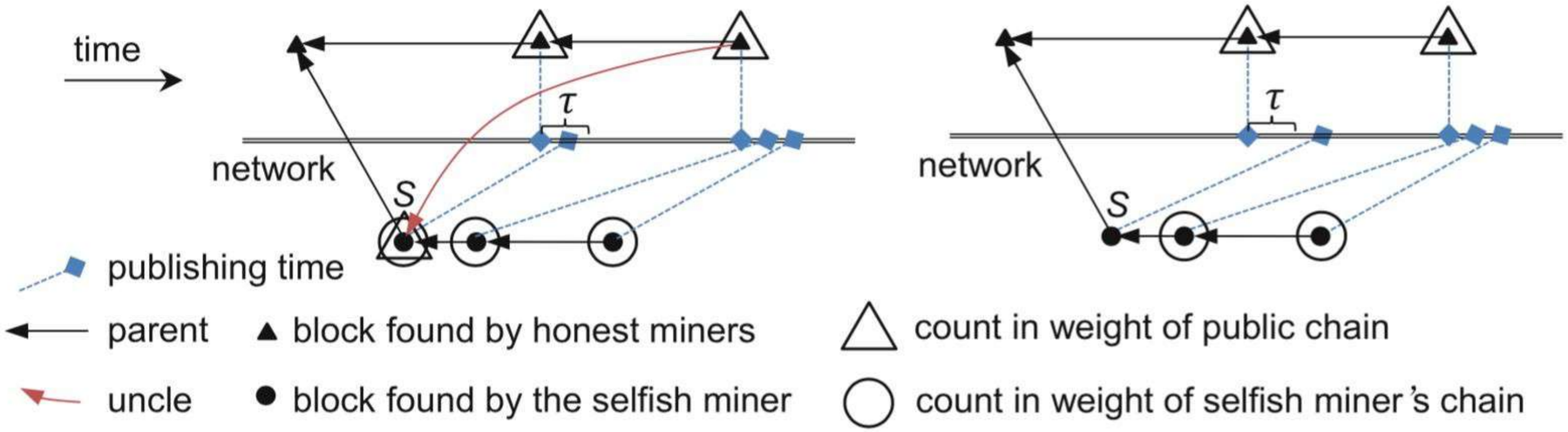
Choice 2: Don't publish

PUBLISH OR PERISH

ZHANG AND PRENEEL, "PUBLISH OR PERISH" (2017)

Result: Regardless of which option is chosen...

- S will **not** contribute to **only** the weight of the selfish chain.
  - Will only contribute to **both** or **neither**
- Completely nullifies the advantage of the secret block S !



Choice 1: Publish

Choice 2: Don't publish

PUBLISH OR PERISH

ZHANG AND PRENEEL, "PUBLISH OR PERISH" (2017)

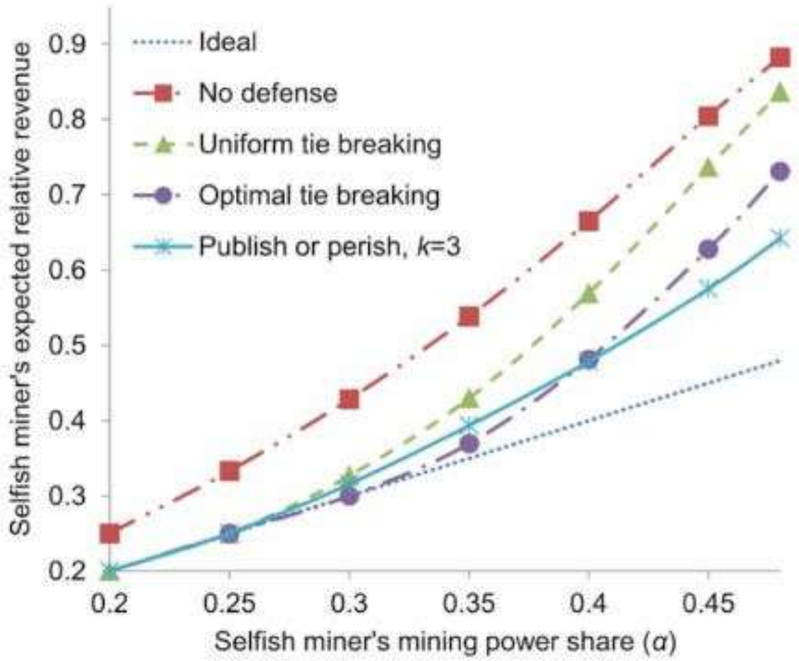


Fig. 5. Comparison with other defenses

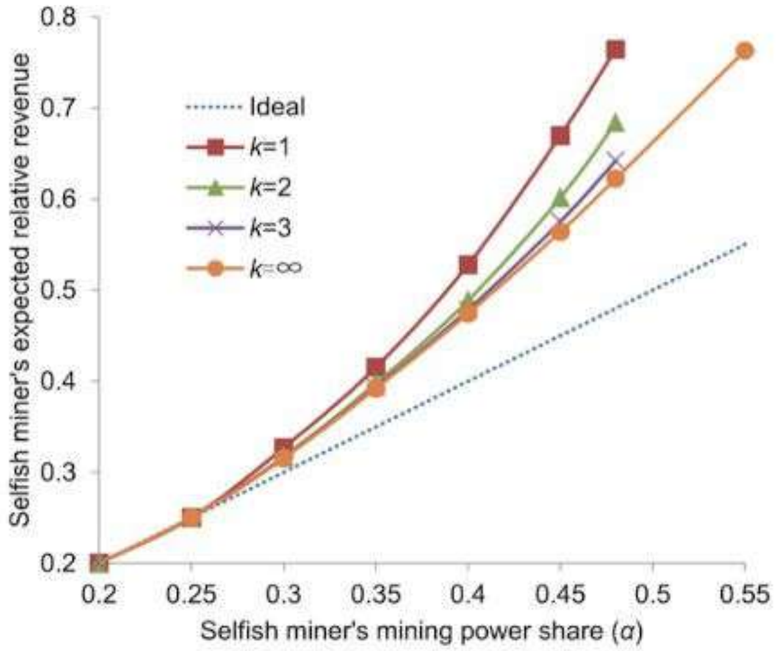


Fig. 4. Relative revenue of the selfish miner within our defense

PUBLISH OR PERISH

ZHANG AND PRENEEL, "PUBLISH OR PERISH" (2017)

Limitations

- Bitcoin aims be asynchronous, Publish and Perish assumes synchronicity
  - (Because it assumes an upper bound of block propagation time)
  - Because of this, it's basically useless
- When the fail-safe parameter k > 1,an attacker may broadcast blocks right before they are late to cause inconsistent views among the honest miners
  - Several other selfish mining defenses also require a fixed upper bound on the block propagation time in order to be effective
- During the transition period to weighted FRP, an attacker can launch double-spend attacks
- Neglects real world factors:
  - Does not permit the occurrence of natural forks
  - Does not consider transaction fees on the selfish miner’s strategy
  - Does not consider how multiple selfish miners could collude and compete with each other
- Does not achieve incentive compatibility, but is the closest scheme to date

READINGS

- https://www.coindesk.com/short-guide-blockchain-consensus-protocols
- CAP Theorem - https://www.youtube.com/watch?v=Jw1iFr4v58M

References

Slides mainly adopted from

- Blockchain @ Berkeley : https://blockchain.berkeley.edu/
- Blockchain @ Princeton : http://bitcoinbook.cs.princeton.edu/

Lecture 8:
SCALING BLOCKCHAIN

LECTURE OVERVIEW

1. BACKGROUND
2. VERTICAL SCALING ON-CHAIN
3. VERTICAL SCALING OFF-CHAIN
4. HORIZONTAL SCALING

1 BACKGROUND

WHAT IS THE SCALABILITY PROBLEM?

SCALABILITY DEFINITION

↑ Volume of Transactions

WHAT IS THE SCALABILITY PROBLEM?

SCALABILITY DEFINITION

↑ Volume of Transactions

↓ Block Time

WHAT IS THE SCALABILITY PROBLEM?

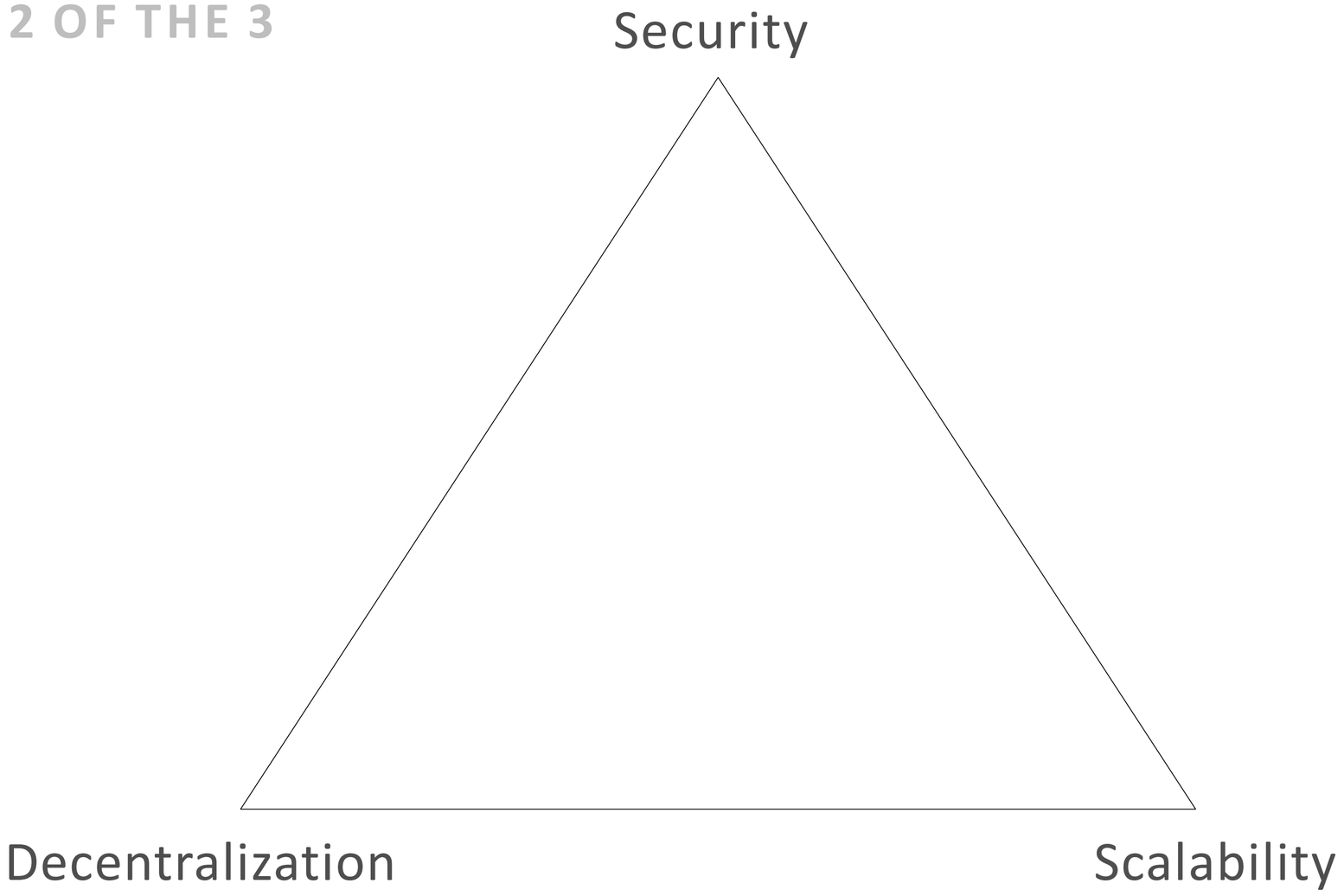
ANOTHER CONSIDERATION

- Size of the blockchain
  - Make it easier for nodes to store in the future
  - Easier to join the network (casually)
  - Currently ~180GB (Late August 2018)

Source: https://www.youtube.com/watch?v=arG9WD0Lsu4

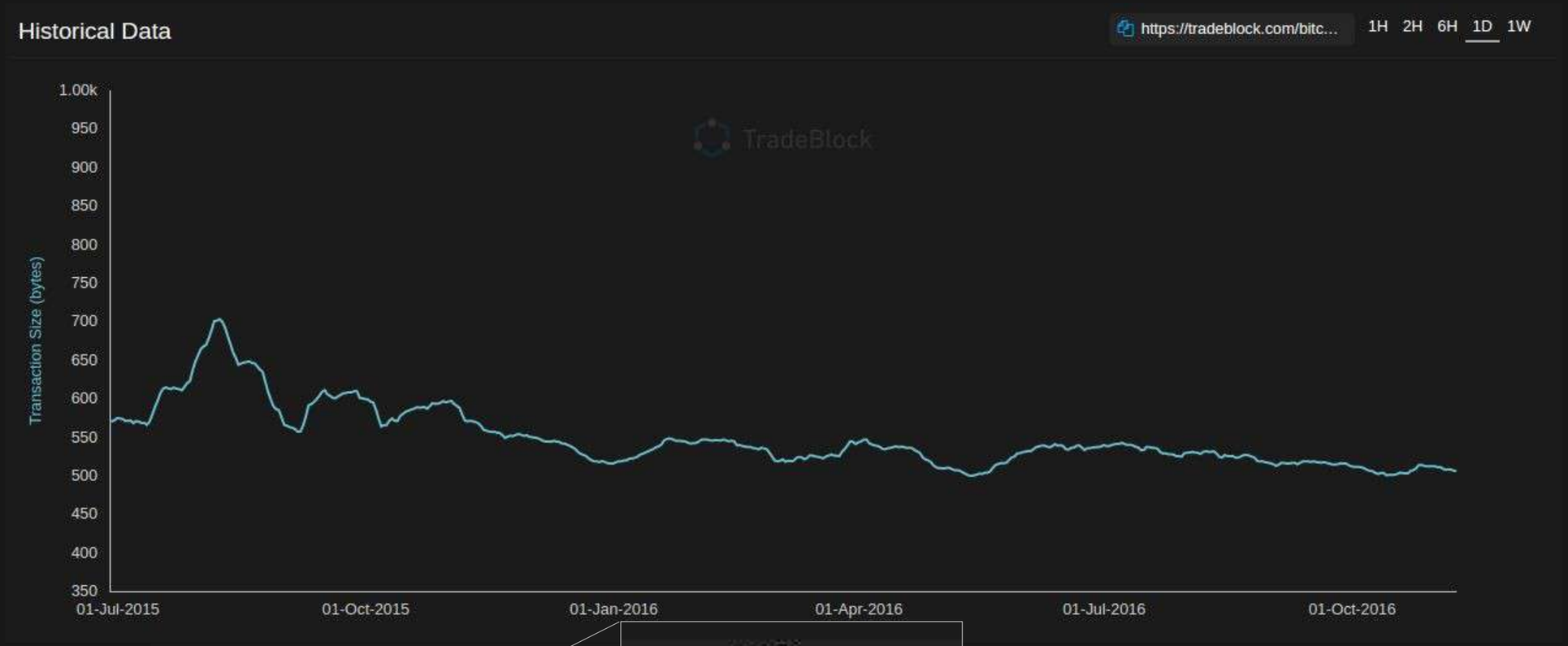
SCALABILITY TRILEMMA

PICK 2 OF THE 3



TRANSACTION SIZE

BLOCKCHAIN FUNDAMENTALS



Statistics

	Transaction Size
Observations	500
Mean	546.38
Median	530.94
Mode	391.77
Std. Dev.	69.58

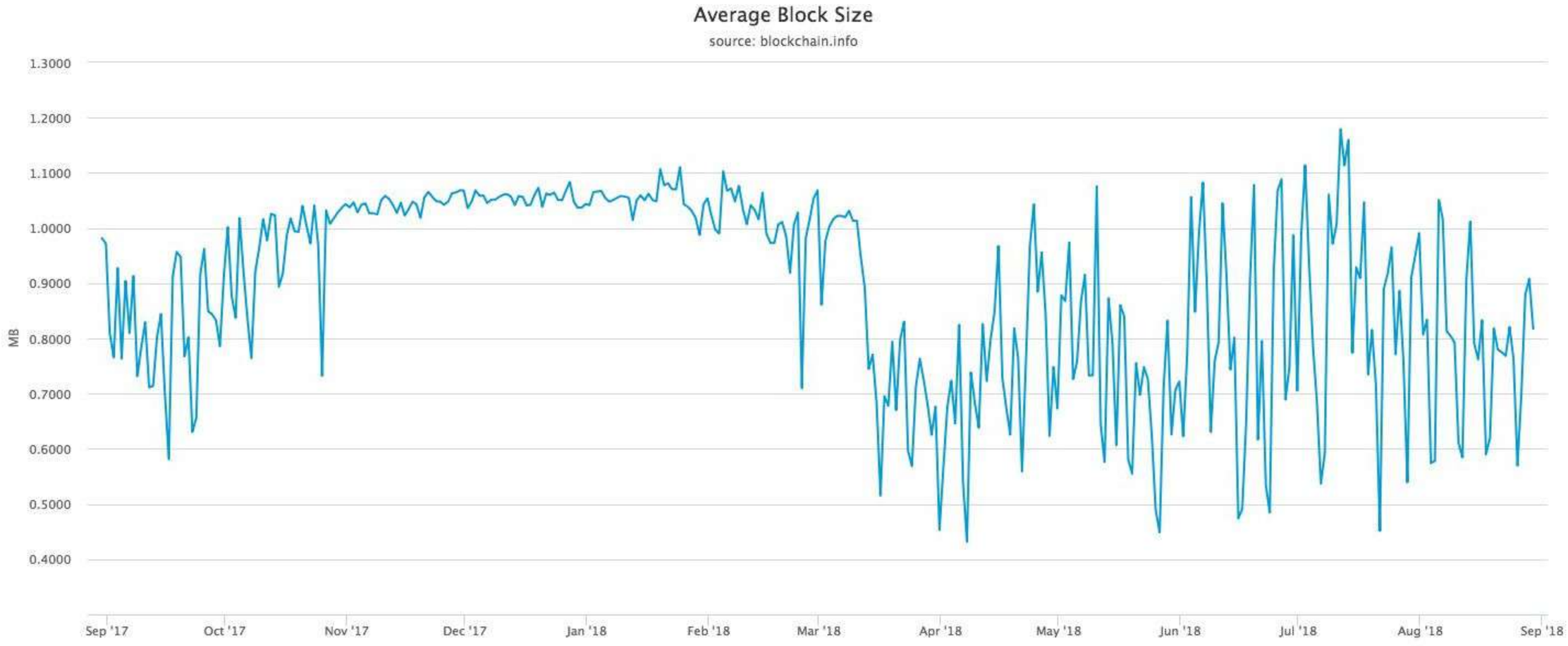
500

546.38

530.94

MAX TRANSACTIONS PERSECOND

BLOCKCHAIN FUNDAMENTALS



MAX TRANSACTIONS PER SECOND

BLOCKCHAIN FUNDAMENTALS

From previous slide:

- Average of 546 bytes per transaction.
- Current blocksize is 1MiB.
- Expected time to next block is 10 min.

Therefore we can compute the sustained maximum transaction volume in tps:

$$\frac{1\ MiB}{1\ block} \times \frac{1\ txn}{546\ bytes} \times \frac{1\ block}{10\ min} \approx 3.2\ tps$$

TPS COMPARISONS

BITCOIN VS MODERN PAYMENTS

How does Bitcoin compare with other traditional payment systems?

	Average	High Load / Maximum
Bitcoin	3 tps	3.2 tps
PayPal*,**	150 tps	450 tps
VISA***	2,000 tps	56,000 tps

*https://investor.paypal-corp.com/secfiling.cfm?filingID=1206774-16-5430&CIK=1633917
**http://www.fool.com/investing/general/2016/02/04/5-things-paypal-holdings-inc-wants-you-to-know.aspx
***https://usa.visa.com/dam/VCOM/download/corporate/media/visa-fact-sheet-Jun2015.pdf

CURRENT SCALABILITY

BLOCKCHAIN FUNDAMENTALS

Variables we can play with to increase $\frac{\text{Transactions}}{\text{Second}}$

- Size of transactions
- Size of blocks

↑ Volume of Transactions

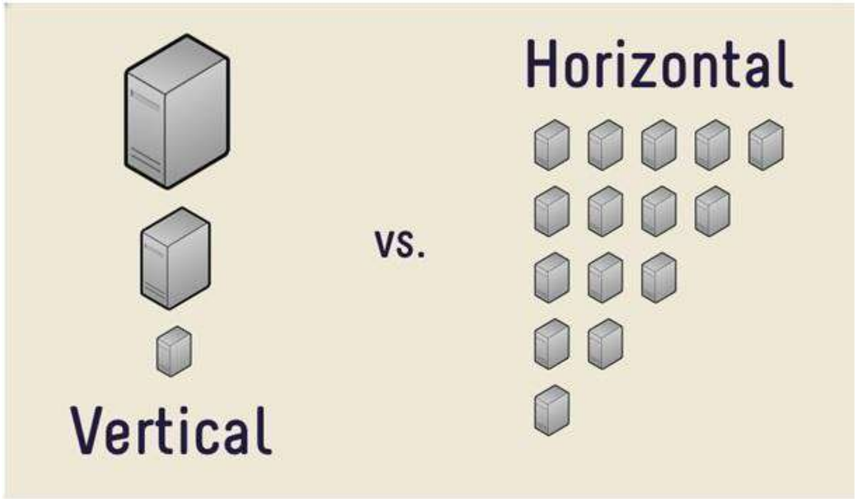
- Block creation rate

↓ Block Time

SCALING TECHNIQUES

HOW DO WE SCALE?

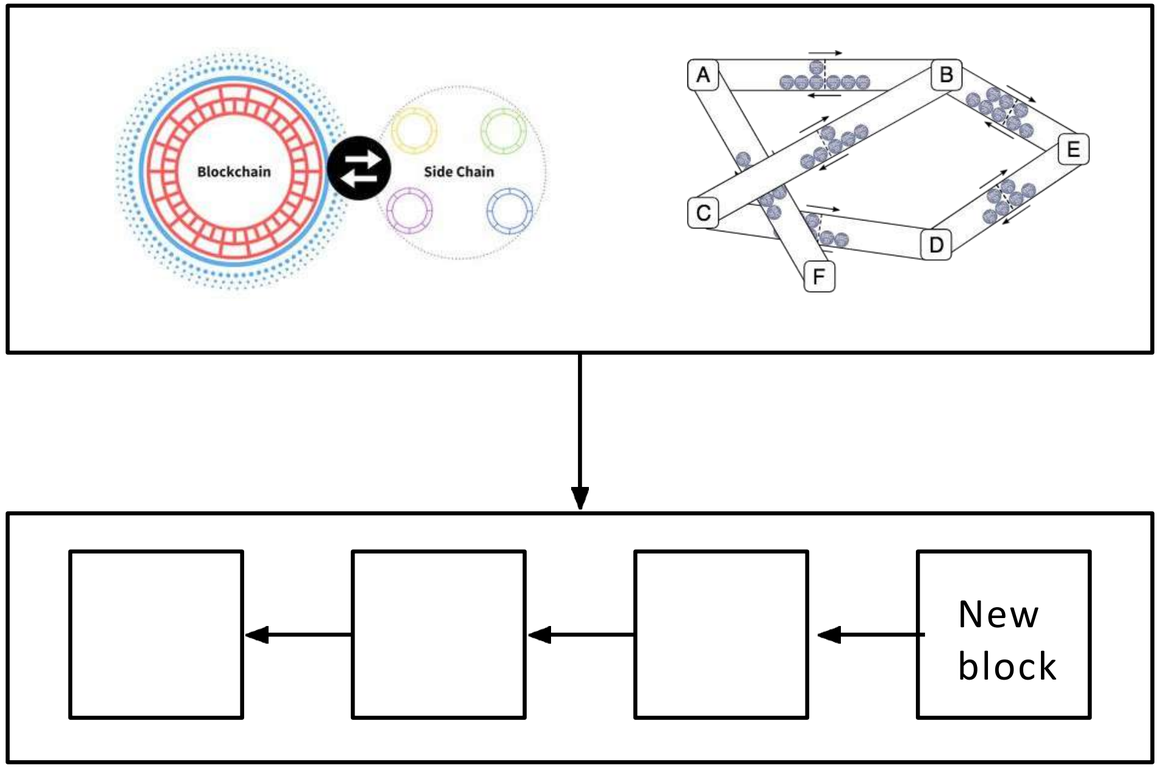
- **Vertical Scaling** - add more RAM/ CPU power to each existing machine
- **Horizontal Scaling** - add more machines of the same computational power
- **Diagonal Scaling** - add more powerful machines



LAYERS OF SCALING

BLOCKCHAINS HAVE LAYERS

- Layer 2 scaling refers to pushing computation off the blockchain
  - Off-chain scaling
- Layer 1 scaling refers to changing the blockchain itself
  - On-chain scaling



Sources:
https://raiden.network/101.html
https://en.bitcoinwiki.org/wiki/Sidechain

2

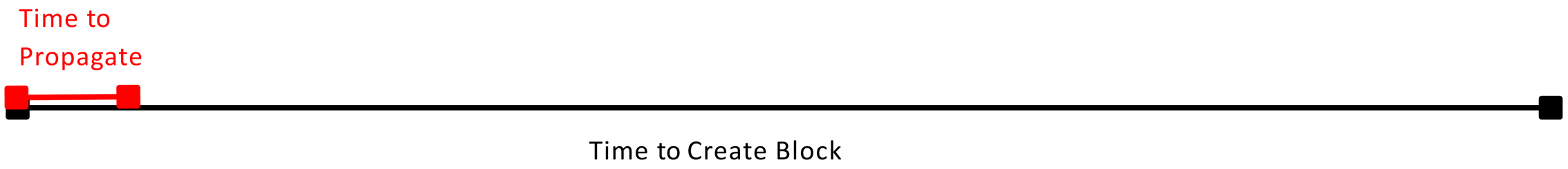
VERTICAL SCALING ON-CHAIN

2.1 NAIVE SOLUTION

NAIVE SOLUTION

IDEA

Why not increase the speed of blocks by decreasing difficulty of the PoW problem?

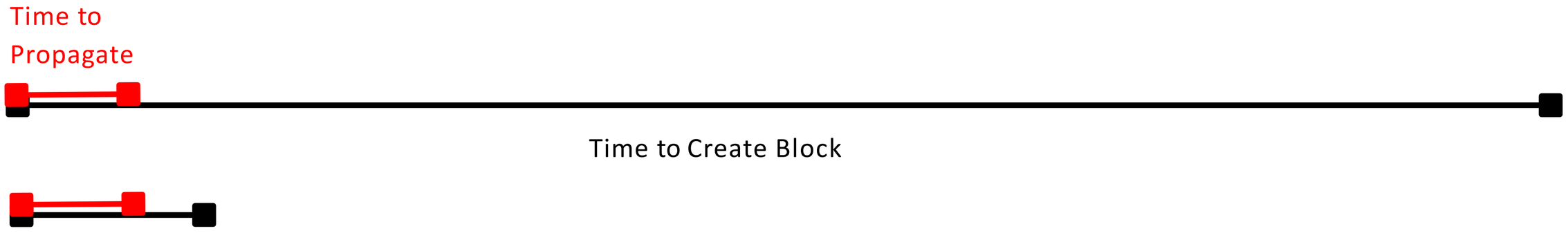


NAIVE SOLUTION

IDEA

Why not increase the speed of blocks by decreasing difficulty of the PoW problem?

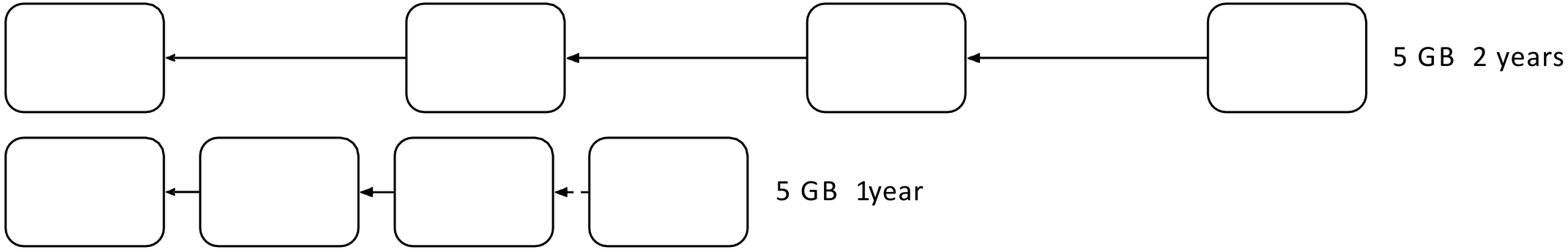
Time to broadcast block fixed while Block creation time decreases



NAIVE SOLUTION

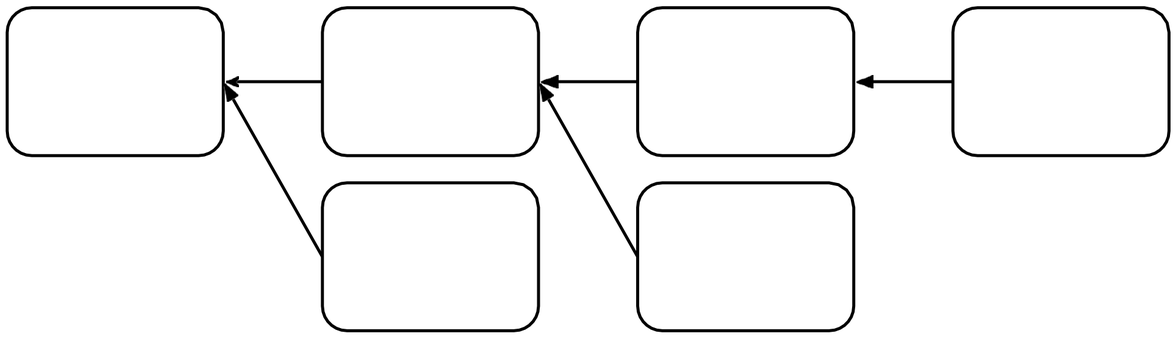
CONS: SIZE

Size matters!



NAIVE SOLUTION

CONS: NATURAL FORKS

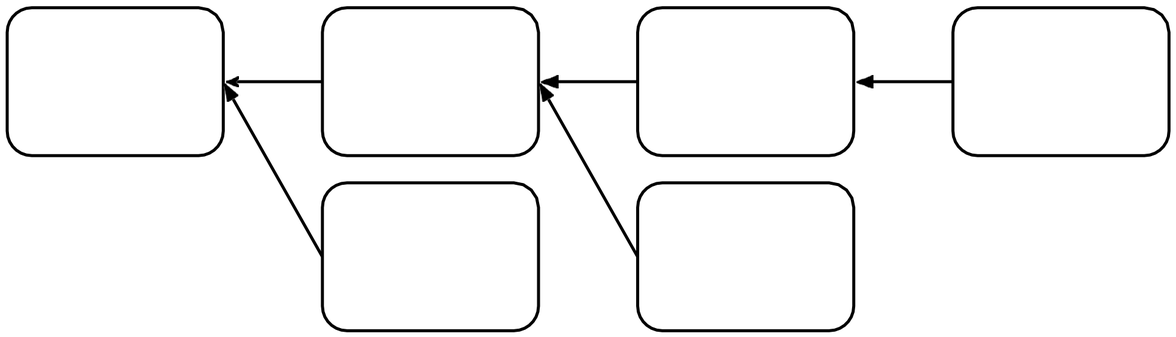


2.2

DECREASE BLOCK CREATION TIME

NAIVE SOLUTION REVISITED

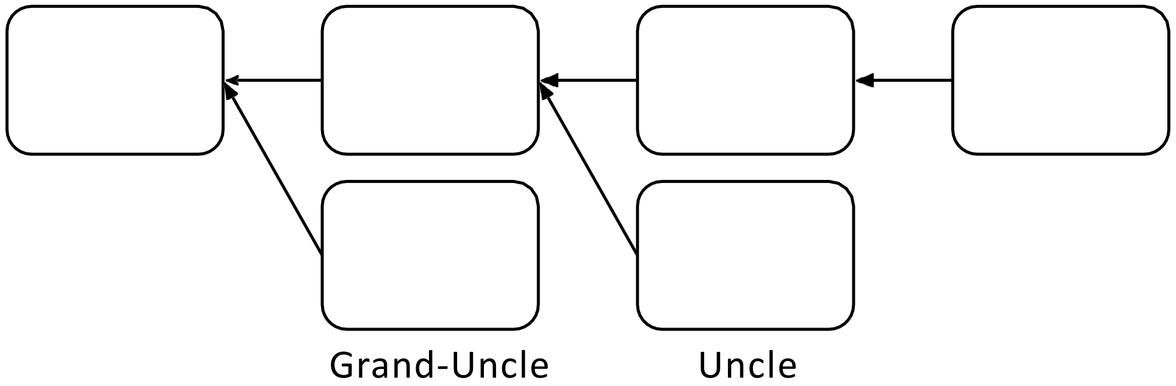
CONS: NATURAL FORKS



GHOST

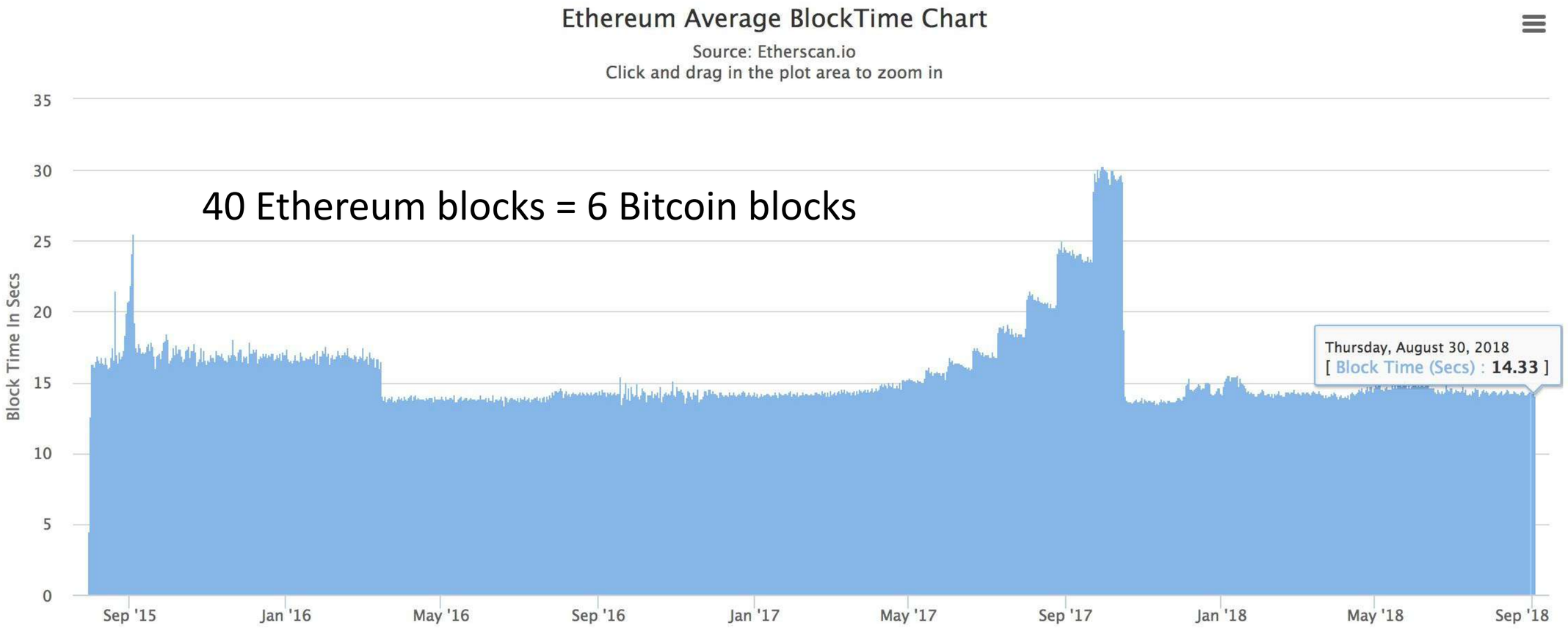
IDEA

- Increase the speed of blocks by **decreasing difficulty of the POW + weighted POW blockchain** (instead of longest)
- + Decrease Incentive for Pooled Mining!



GHOST

ETHEREUM AVERAGE BLOCK TIME



40 Ethereum blocks = 6 Bitcoin blocks

Source: https://etherscan.io/chart/blocktime

2.3 INCREASE BLOCKSIZE

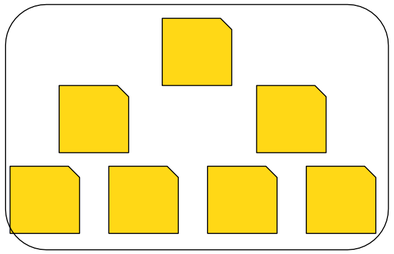
BLOCKSIZE INCREASE

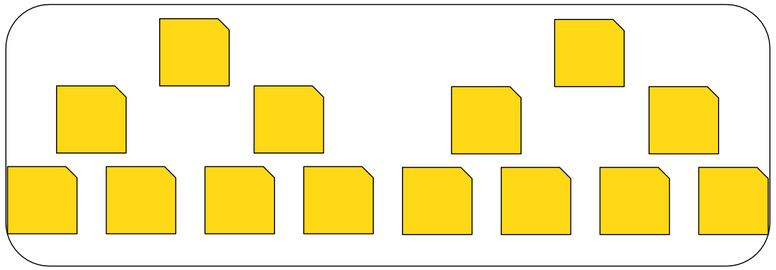
IDEA

If we just increase the blocksize, we can fit more transactions in a single block!

Pros:

- It's an "easy" implementation. Just get miners to agree.
- Lower transaction fees for users
  - Verification of transactions on the same block



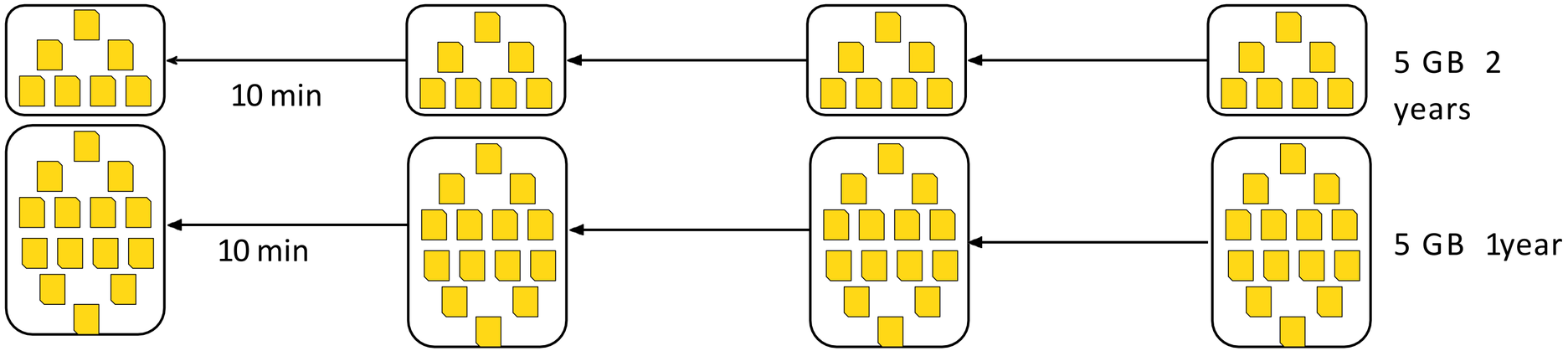


BLOCKSIZE INCREASE

CONS

- Hard fork
- Lessen transaction fees
- Size increases very fast

- One time linear capacity increase
  - Temporary Fix
- Longer Propagation Times
  - Authoring miner has better shot at next block



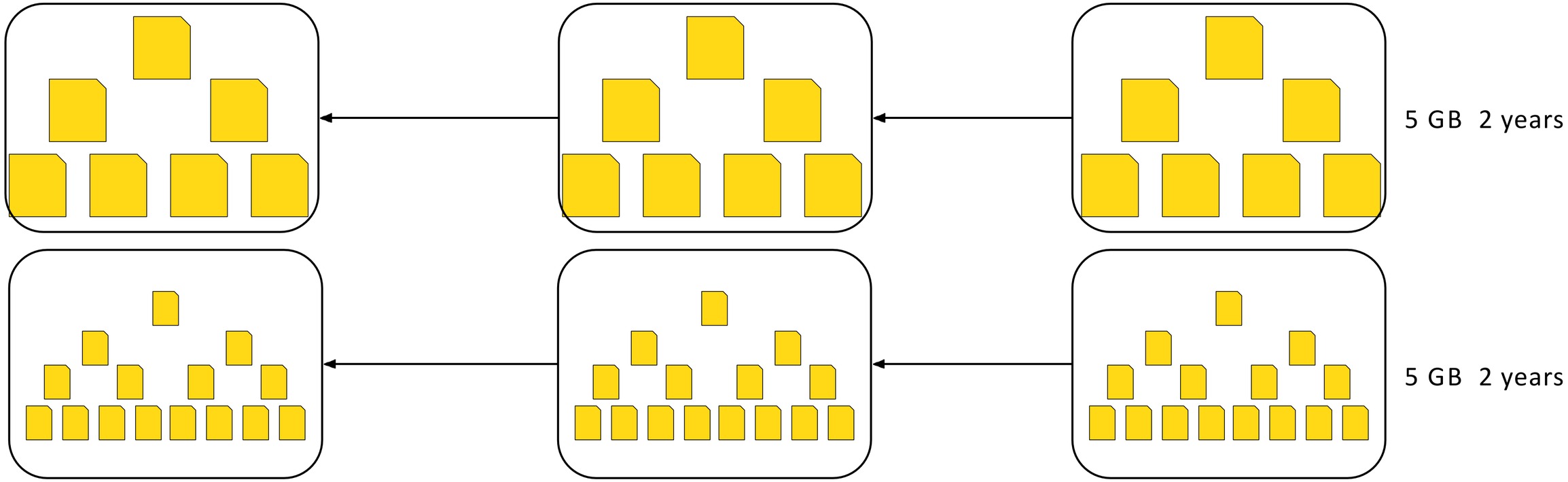
2.4

DECREASE SIZE OF TRANSACTIONS

DECREASE TRANSACTION SIZE

IDEA

- Segwit



SEGREGATED WITNESS

IDEA

input(s)

```
"in":[
 {
  "prev_out":{
   "hash":"3be4ac9728a0823cf5e2deb2e86fc0bd2aa503a91d307b42ba76117d79280260",
   "n":0
  },
   "scriptSig":"30440..."
 },
 {
  "prev_out":{
   "hash":"7508e6ab259b4df0fd5147bab0c949d81473db4518f81afc5c3f52f91ff6b34e",
   "n":0
  },
  "scriptSig":"3f3a4ce81...."
 }
],
```

input script

hash of previous tx

index of output of previous tx

signature (proof of ownership)

Signature field takes around %50 of the space. It is used to

- Verify the transaction is valid
- Owner is legit

SEGREGATED WITNESS

AVOID HARD FORK

Segwit P2W*

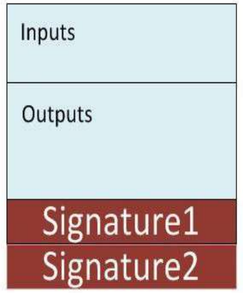
For New Nodes:

ScriptPubKey: 0 e4873ef43eac347471dd94bc899c51b395a509a5

ScriptSig: Empty

WitScript: Signature1

Result: **Valid**



Source: https://programmingblockchain.gitbooks.io/

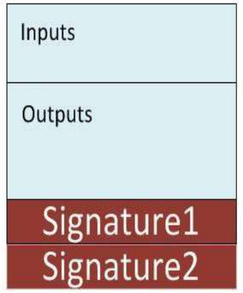
SEGREGATED WITNESS

AVOID HARD FORK

Segwit P2W*

For New Nodes:

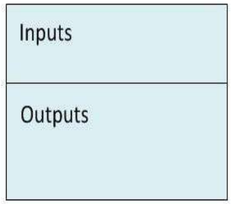
ScriptPubKey: 0 e4873ef43eac347471dd94bc899c51b395a509a5
ScriptSig: Empty
WitScript: Signature1
Result: **Valid**



Segwit P2W*

For Old Nodes:

ScriptPubKey: 0 e4873ef43eac347471dd94bc899c51b395a509a5
ScriptSig: Empty
Result: **Valid**

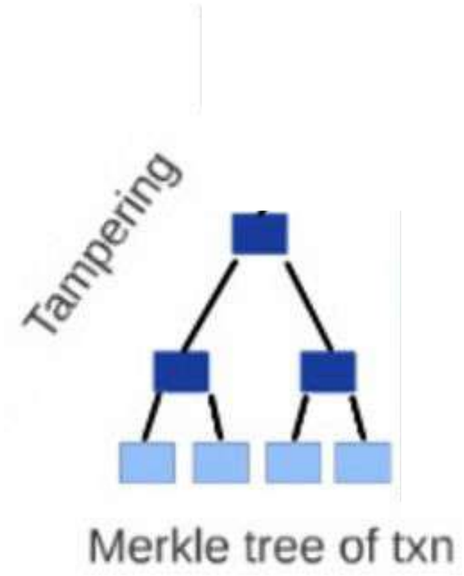


Source: https://programmingblockchain.gitbooks.io/

SEGREGATED WITNESS

MIRROR SIGNATURE TREE

But now, the blockchain doesn't have any evidence that correct signatures were included in transactions?

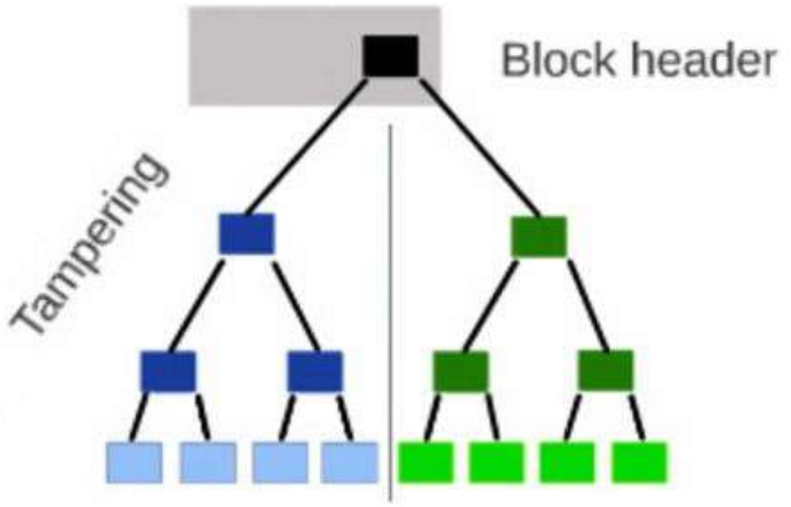


SEGREGATED WITNESS

MIRROR SIGNATURE TREE

But now, the blockchain doesn't have any evidence that correct signatures were included in transactions.

- A SegWit miner creates a Merkle tree out of segregated witnesses that mirrors the transaction tree
- This tree's merkel root is included in the input field of the coinbase transaction.
- This changes the transaction ID of the coinbase transaction
- Therefore signatures influence the block header and, ultimately, the makeup of the blockchain.



Merkle tree of txn and witness

SEGREGATED WITNESS

PROS AND CONS

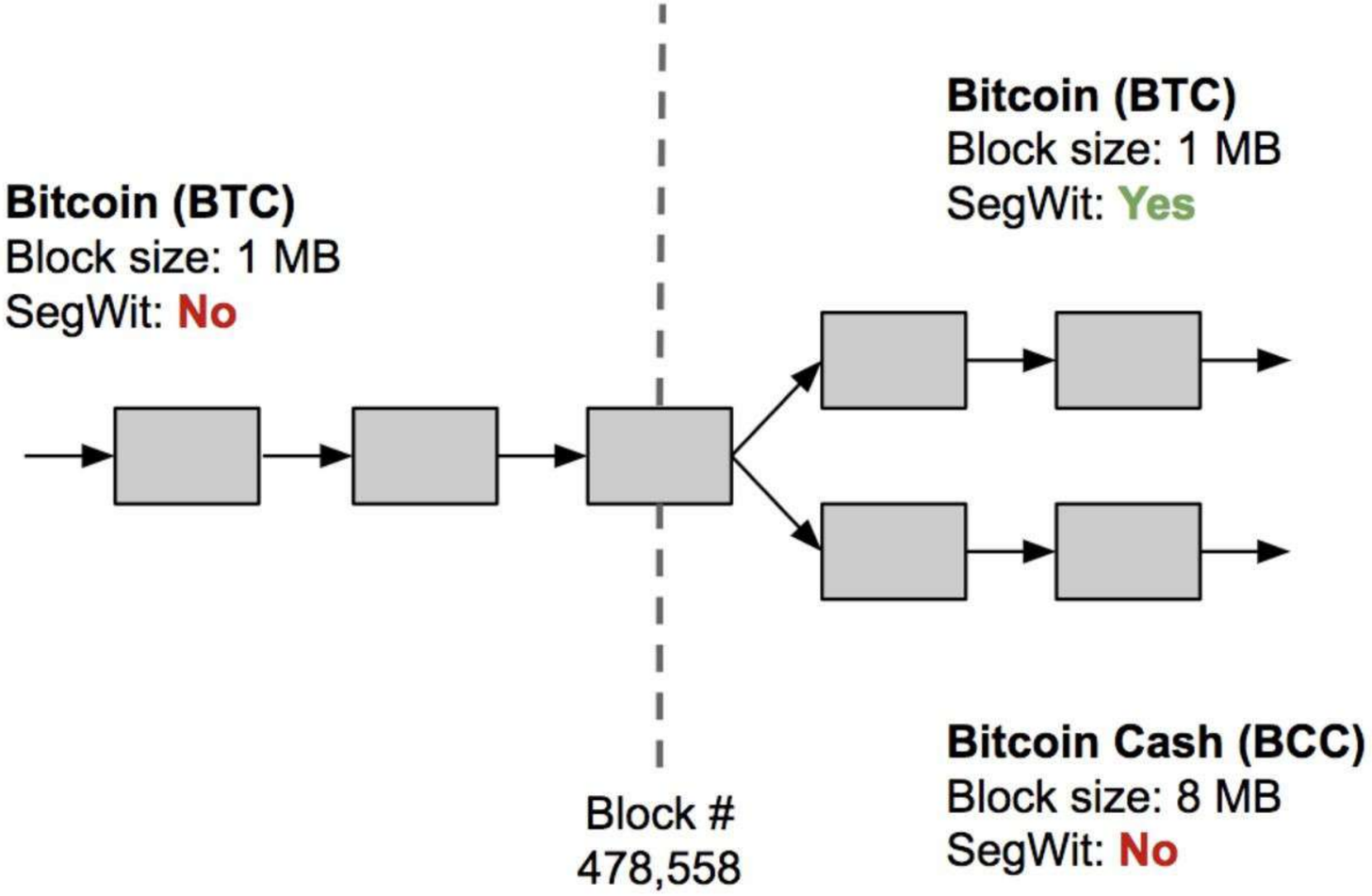
Pros:

- Fixes Transaction Malleability
  - Allows Lightning Network and sidechains to work
- Only soft fork
- No slippery slope
- Efficiency Gains
- Smaller Size of Blockchain

Cons:

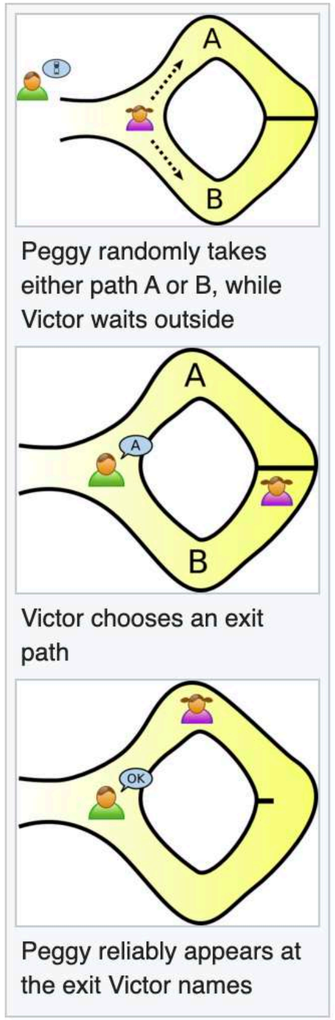
- One time linear capacity increase
- Very complicated and ugly (Over 500 lines of code)
- Wallets have to incorporate it
- Other ways to solve malleability

Source: https://bitco.in/forum/threads/segregated-witness-sotf-fork-segwit-pros-and-cons.986/


Bitcoin (BTC)
Block size: 1 MB
SegWit: No
Bitcoin (BTC)
Block size: 1 MB
SegWit: Yes
Block #
478,558
Bitcoin Cash (BCC)
Block size: 8 MB
SegWit: No

Zero Knowledge Proofs

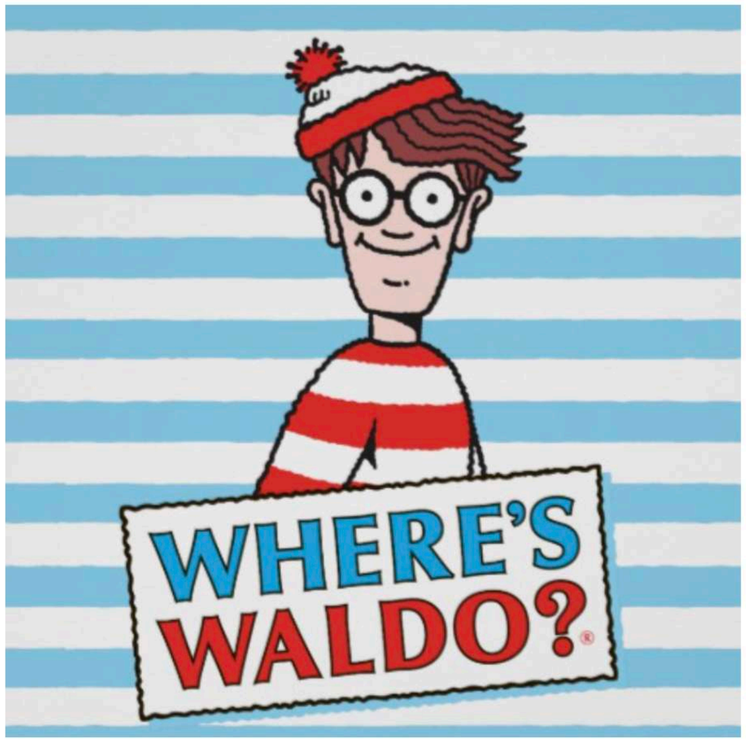
A **zero-knowledge proof** is a method by which one party (the prover) can prove to another party (the verifier) that they know a value *x*, without conveying any information apart from the fact that they know the value *x*. The essence of zero-knowledge proofs is that it is trivial to prove that one possesses knowledge of certain information by simply revealing it; the challenge is to prove such possession without revealing the information itself or any additional information.

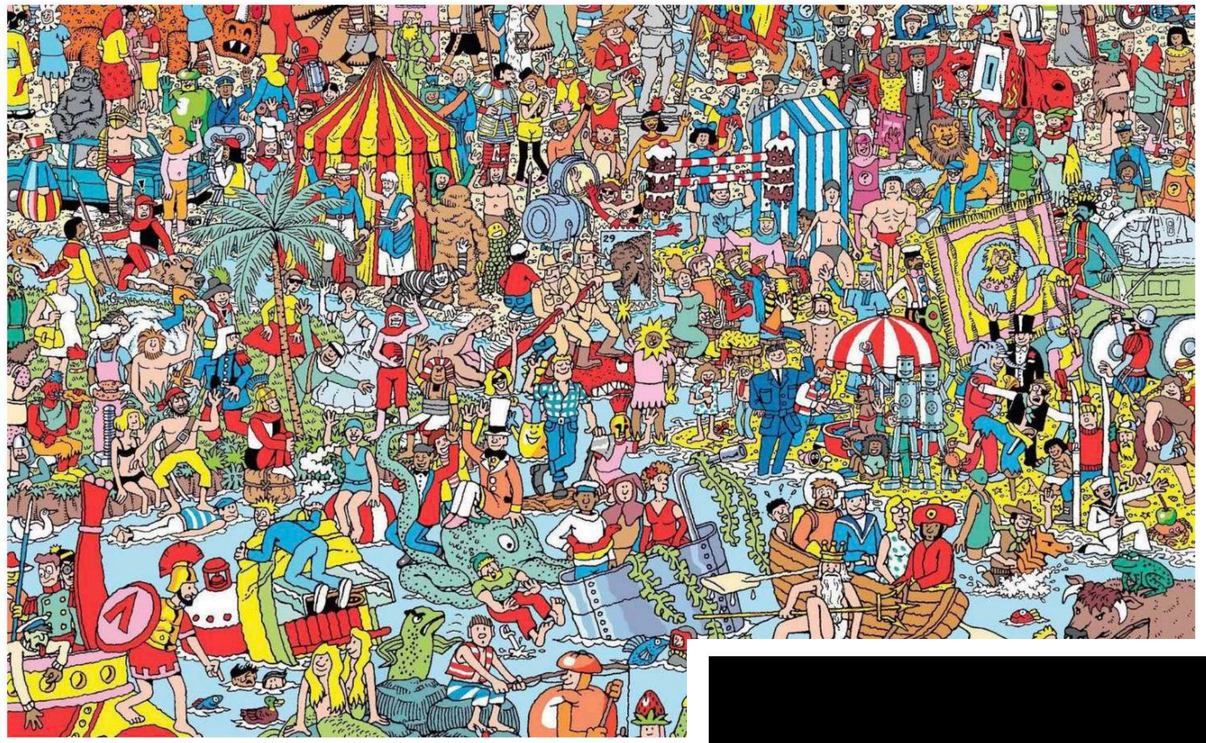


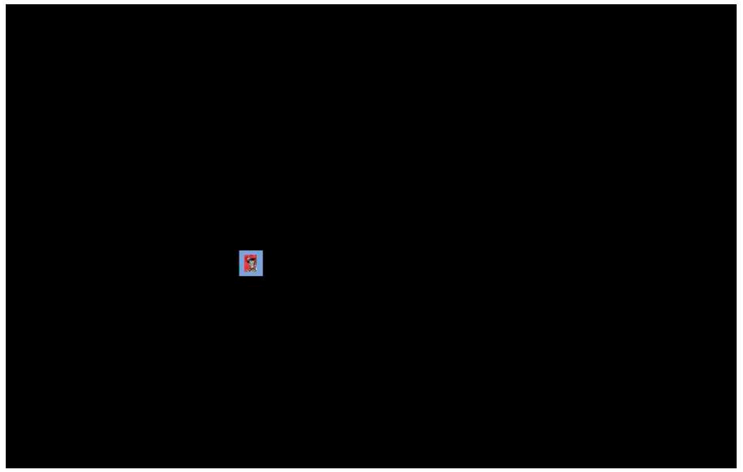
https://en.wikipedia.org/wiki/Zero-knowledge_proof

Decrease transaction size by utilizing Zero knowledge Proofs and not including the whole signature

Zero Knowledge Proofs







Here's Waldo!

3 VERTICAL SCALING OFF-CHAIN

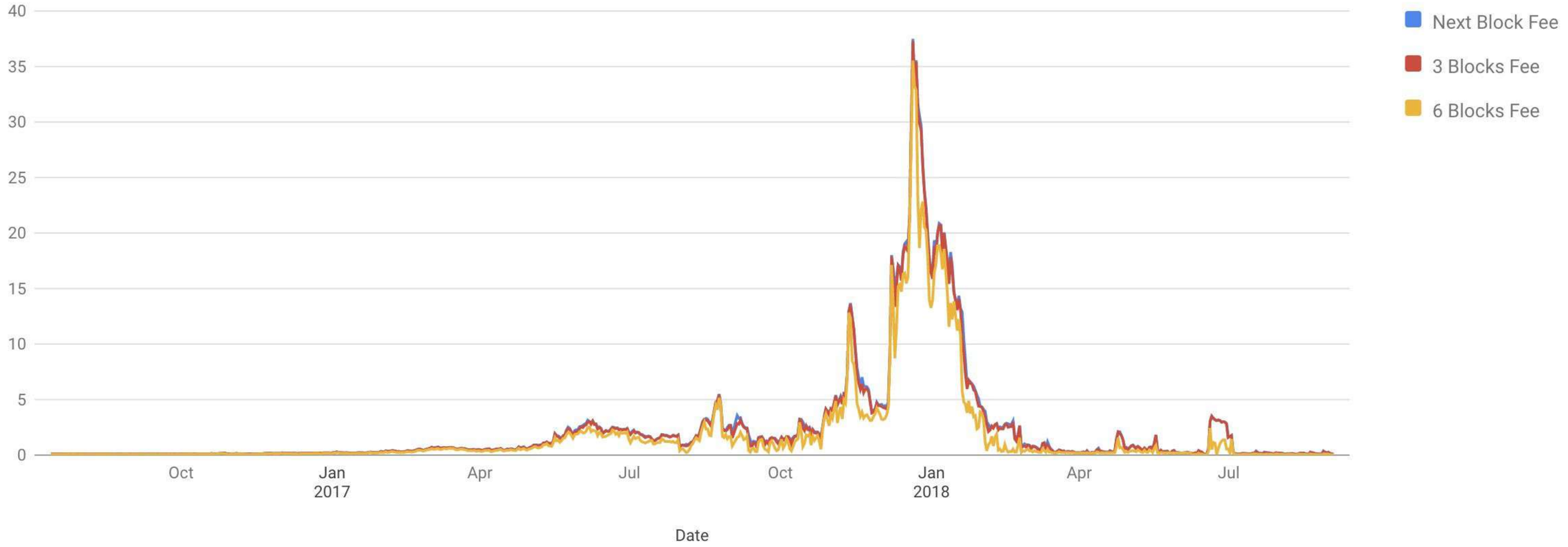
RECALL: BITCOIN TRANSACTIONS

ISSUE WITH BITCOIN PAYMENTS

- **Long delays**
  - 6 confs = about 1hour wait
- **High Fees**
  - Inconsistencies (11-2017, $6.75 avg tx fee)
  - Not economical for low-value items

RECALL: BITCOIN TRANSACTIONS

ISSUE WITH BITCOIN PAYMENTS



Source: https://bitcoinfees.info/

PAYMENT CHANNEL BUILDUP

PRIVATE CHANNELS

Idea:

- Can Alice and Bob make payments between themselves without always needing to consult the blockchain?

PAYMENT CHANNEL BUILDUP

PRIVATE CHANNELS

Idea:

- What if Alice and Bob maintain a **private balance sheet**
  - update the private balance sheet with every payment
  - only consult the blockchain when they want to settle the balance

Alice and Bob's Balance Sheet	
Alice	Bob
10 BTC	5 BTC

Alice pays Bob 2 BTC

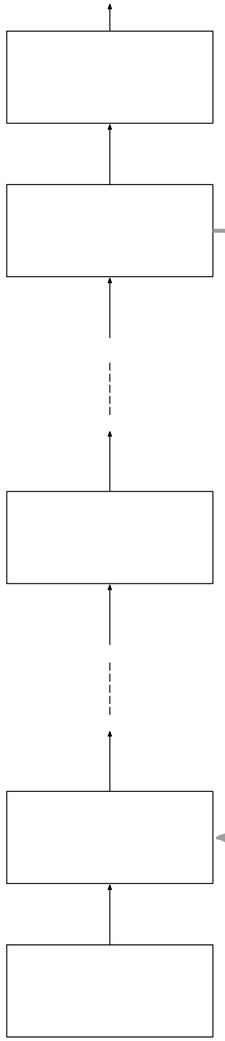
Alice and Bob's Balance Sheet	
Alice	Bob
8 BTC	7 BTC

PAYMENT CHANNEL BUILDUP

PRIVATE CHANNELS

Alice and Bob only make a transaction on the blockchain when they want to settle their private balances.

BLOCKCHAIN



Alice and Bob open a private balance sheet

Alice and Bob's Balance Sheet	
Alice	Bob
10 BTC	0 BTC

Alice and Bob make several private txns.

Alice and Bob's Balance Sheet	
Alice	Bob
3 BTC	7 BTC

Alice and Bob later close the balance sheet

PAYMENT CHANNEL BUILDUP

IDEA

- Use Bitcoin script to create blockchain-enforceable contracts between Alice and Bob so that neither party can cheat the other, while maintaining the private balance sheet functionality!
- In blockchain land, we call these **payment channels**.
- Payment channel's full name: *Payment Channel with Hash TimeLock Contract (HTLC)*

PAYMENT CHANNEL PAYMENTS

BLOCKCHAIN FUNDAMENTALS

State 0: Total 10 BTC	What Bob needs to claim 1st (on chain)	What Alice needs to claim 1st (on chain)
Alice: 10 BTC	Alice Sig	Bob Sig + Alice Secret 1 or Alice Sig + 1000 blocks
Bob: 0 BTC	Alice Sig + Bob Secret 1 or Bob Sig + 1000 blocks	Bob Sig

PAYMENT CHANNEL PAYMENTS

BLOCKCHAIN FUNDAMENTALS

State 0: Total 10 BTC	What Bob needs to claim 1st (on chain)	What Alice needs to claim 1st (on chain)
Alice: 10 BTC	Alice Sig	Bob Sig + 32874494 or Alice Sig + 1000 blocks
Bob: 0 BTC	Alice Sig + 1273394 or Bob Sig + 1000 blocks	Bob Sig

State 1: Total 10 BTC	What Bob needs to claim 1st (on chain)	What Alice needs to claim 1st (on chain)
Alice: 7 BTC	Alice Sig	Bob Sig + Alice Secret 2 or Alice Sig + 1000 blocks
Bob: 3 BTC	Alice Sig + Bob Secret 2 or Bob Sig + 1000 blocks	Bob Sig

Alice sends 3 BTC to Bob!

PAYMENT CHANNEL CONCLUSIONS

GOALS REACHED

Observation:

- If either Alice or Bob cheat
  - can always override and take all the money in the deposit.
- If Alice and Bob always cooperate
  - never have to touch the blockchain, except when creating the payment channel and settling the balance.
- Only two transactions on the blockchain
  - Supports arbitrary number of local transactions between Alice and Bob.

PAYMENT CHANNEL CONCLUSIONS

ISSUES

Issue:

- Alice and Bob need to have capital locked up in this HTLC (Hash Time-Lock Contract) before they can send money between each other.

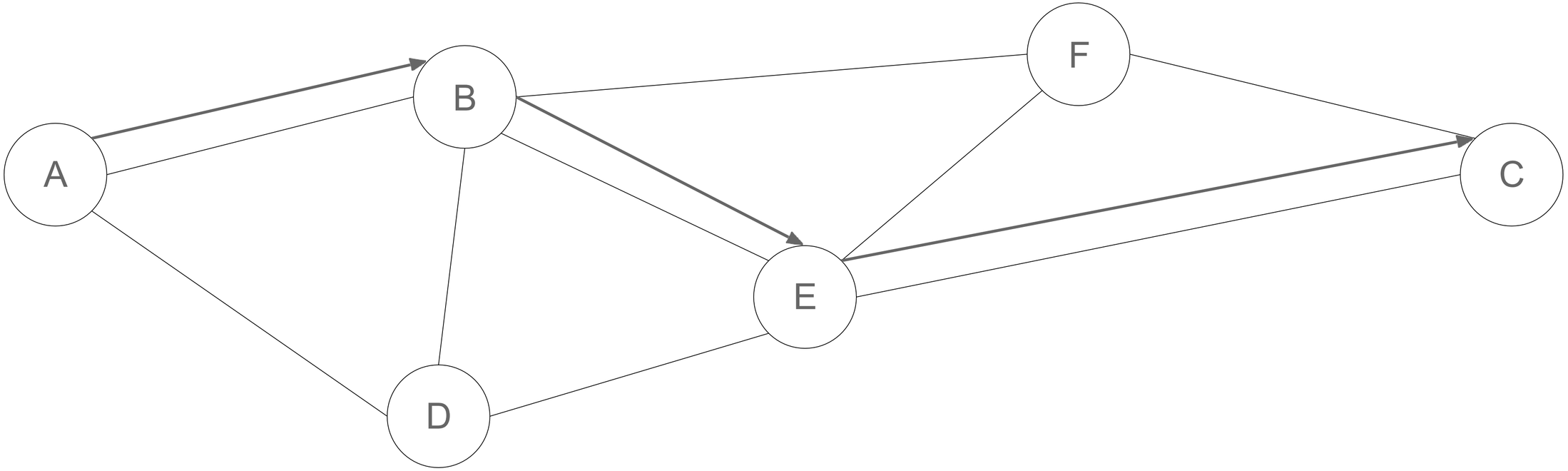
Issue:

- With this payment channel, Alice and Bob can only easily and scalably send money between themselves.

LIGHTNING NETWORK

BLOCKCHAIN FUNDAMENTALS

- Alice sends money to Charlie through this hypothetical payment channel network



LIGHTNING NETWORK

Can we do this securely?

- With some small additions on top of our HTLC construction, we can trustlessly send money across a network of HTLCs!

The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments

- By Joseph Poon & Thaddeus Dryja

https://lightning.network/lightning-network-paper.pdf

LIGHTNING NETWORK SCALABILITY

What does the Lightning Network mean for scalability?

1. If we assume that there is enough capital in this payment channel network, *people can make payments instantly*.
   a. don't need to wait for confirmation times
   b. transactions as fast as communication delay across network.
2. Only use the Bitcoin blockchain as an arbiter to settle disputes and close out payment channels
   a. far fewer (expensive) transactions on the Blockchain.

LIGHTNING NETWORK SCALABILITY

What does the Lightning Network mean for scalability?

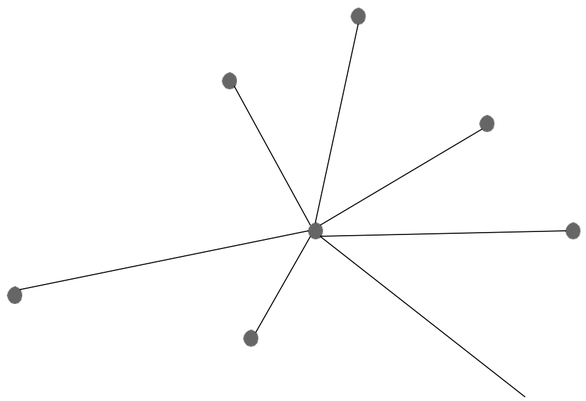
3. Instead of 3 tps, the Bitcoin network can support 10,000's+ of tps
   a. delegate payments to simple bookkeeping in each payment channel
   b. kept off-chain 99% of the time!
4. Sending packets across the internet is very cheap and fast.
   a. Lightning Network transaction fees will be several orders of magnitude cheaper.
   b. Only pay expensive fees on channel open /close.

LIGHTNING NETWORK SCALABILITY

Issues with Lightning Network:

1. Nodes need to keep very large amounts of capital locked up in payment channels.
   a. problematic if most payments in only one direction
2. **Strong centralization force**, since only nodes with significant capital can afford to hold payment channels for long.
   a. Larger payment channels get settled less often, less fees
3. **Less capital is required with less nodes** on the network
   a. ⇒ tendency towards *hub-and-spoke network topology*.

Hub-and-Spoke Topology

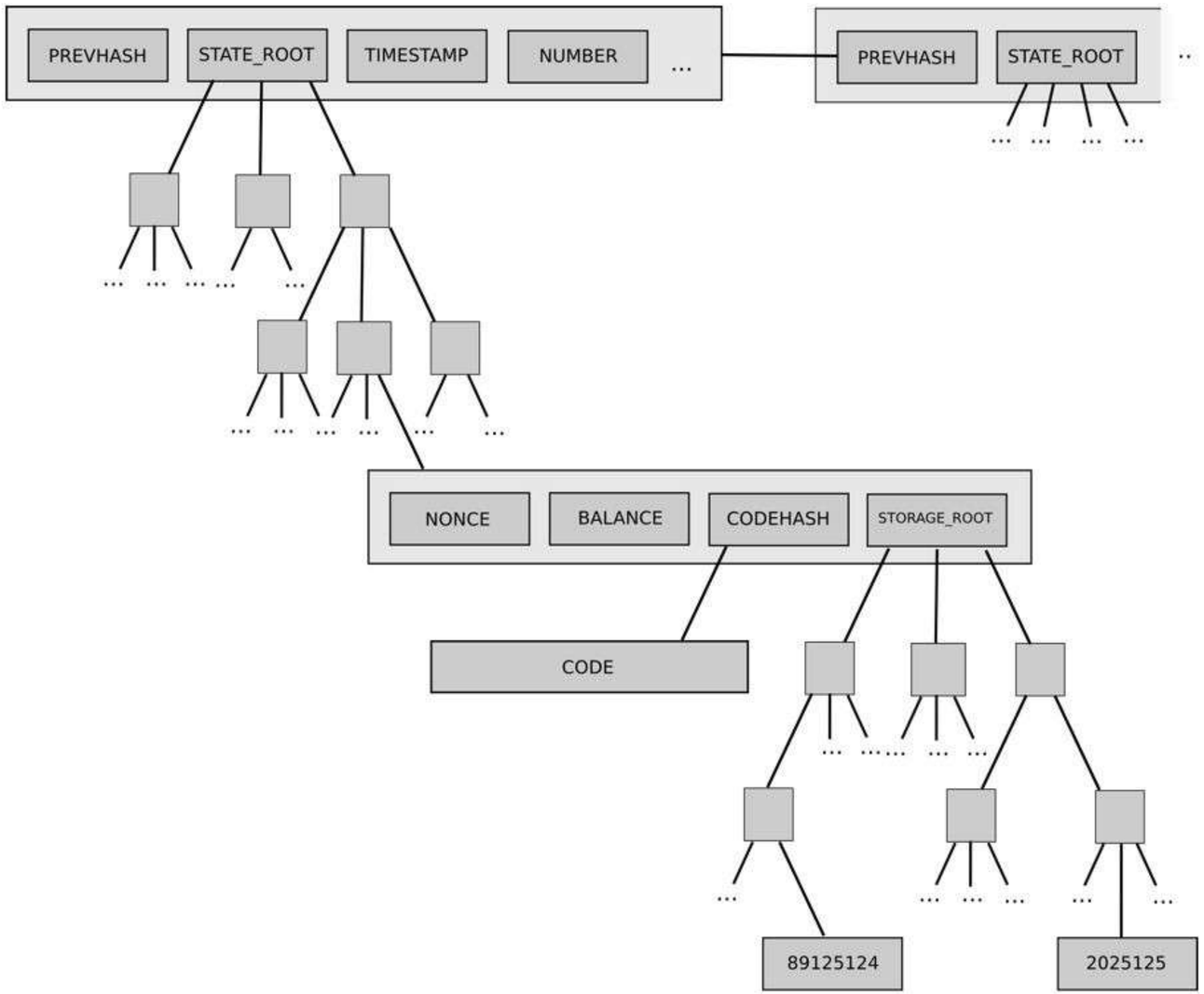


4 HORIZONTAL SCALING

SHARDING

IDEA

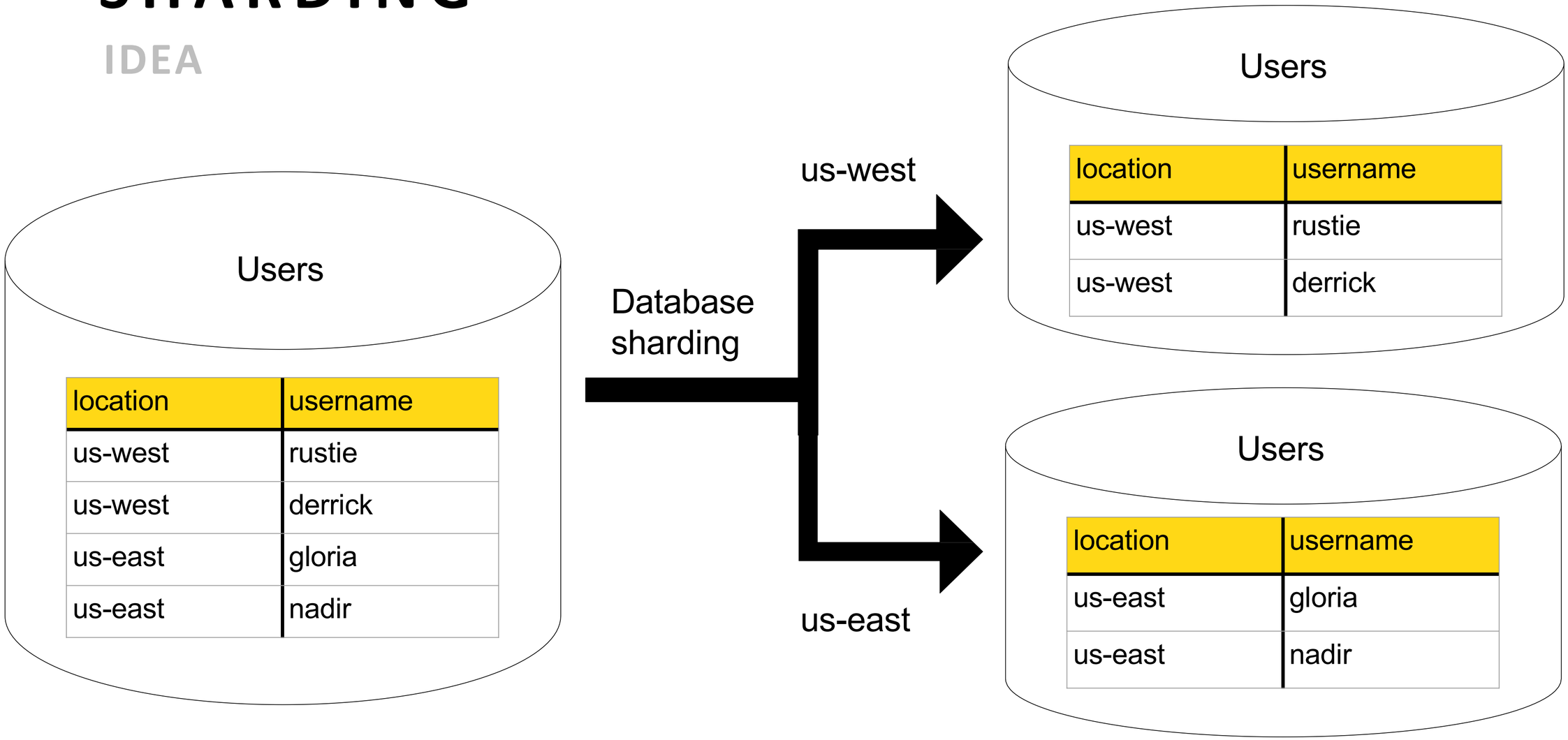
Sharding is the idea of not requiring every miner to be working on every single block, essentially creating parallel but connected blockchains.



Source: https://github.com/ethereum/wiki/wiki/Sharding-FAQs

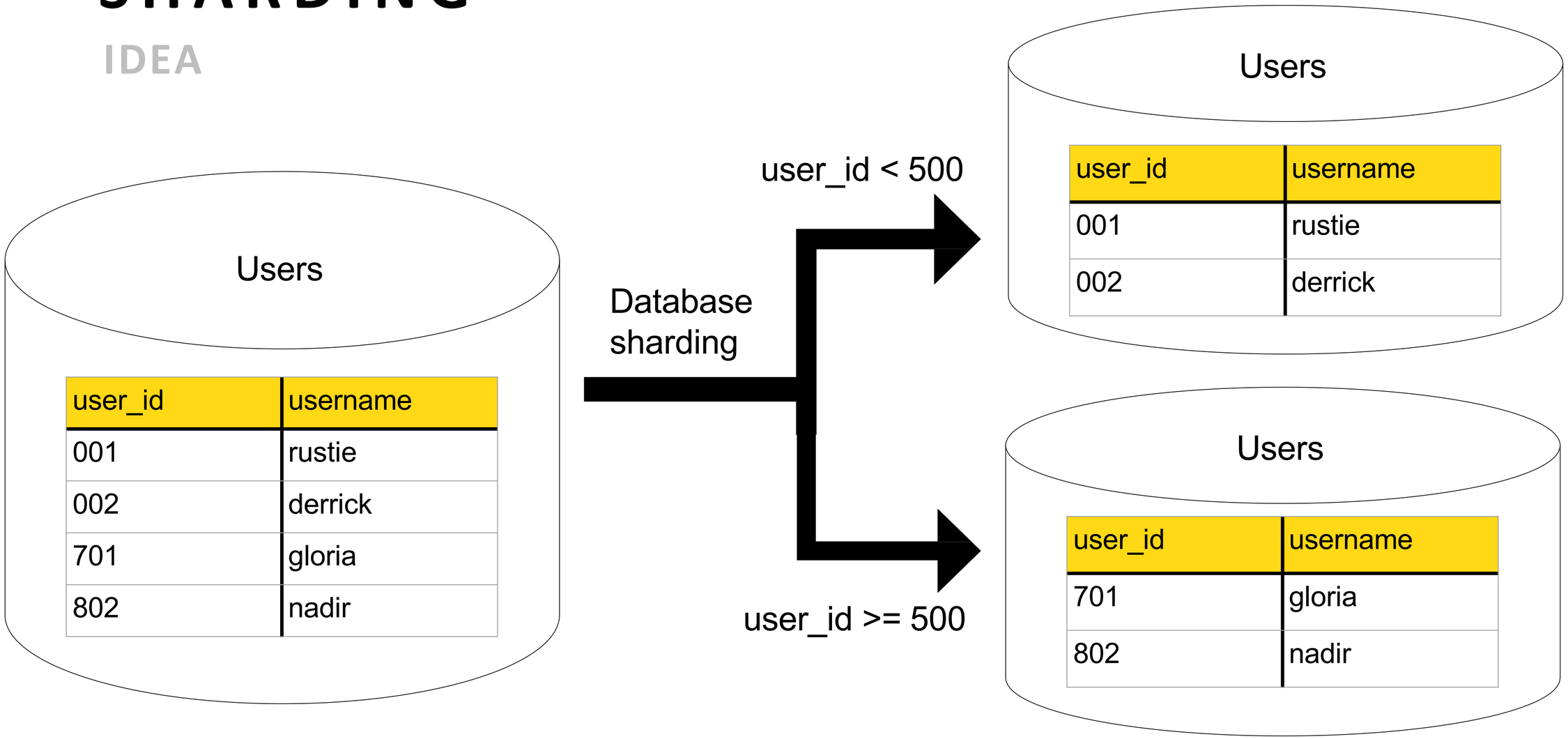
SHARDING

IDEA



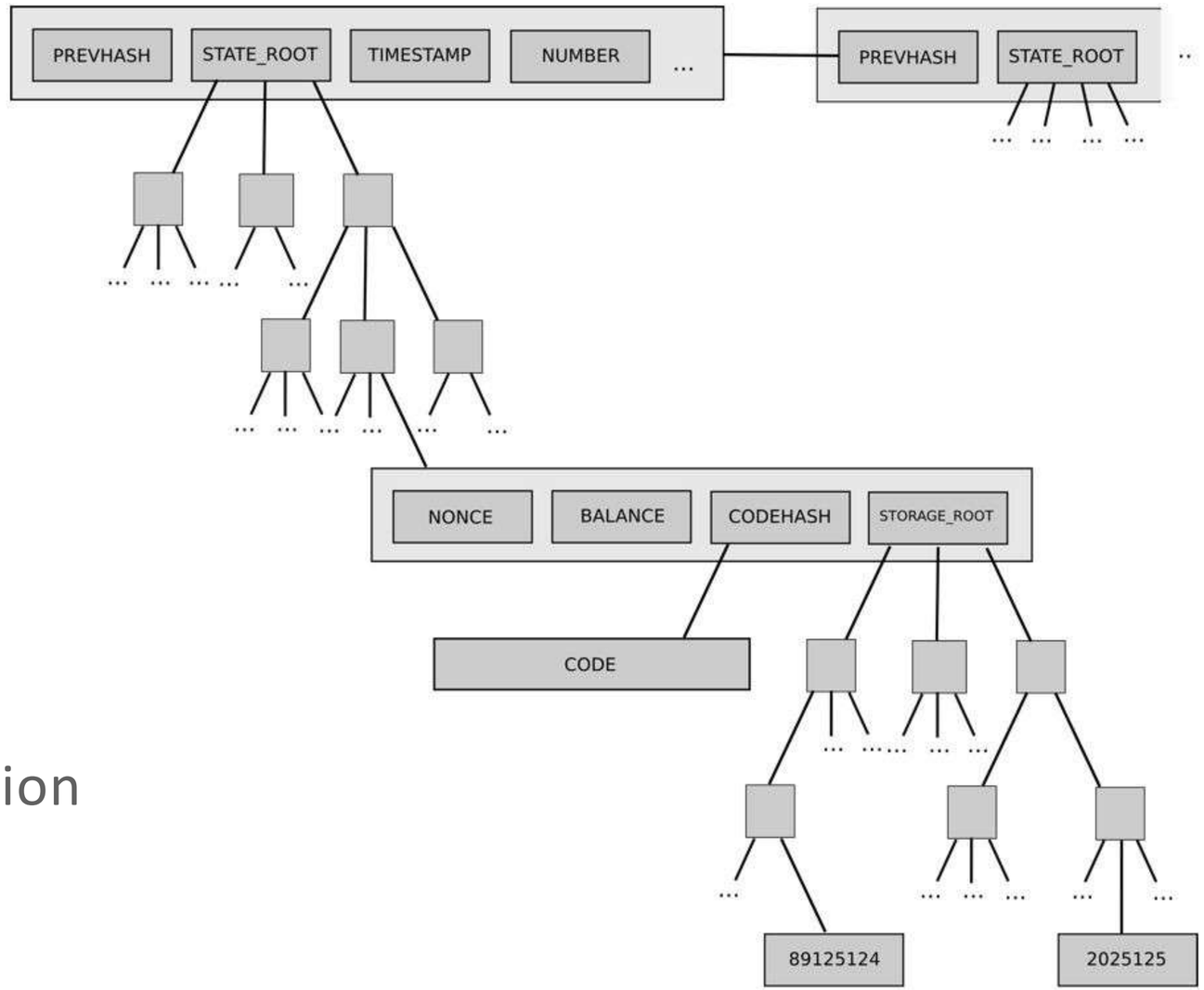
SHARDING

IDEA



SHARDING

IDEA



Node categories:

- super-full node
- top-level node
- single-shard node
- light node

Some challenges:

- Cross-shard communication
- Single-shard takeover

Source: https://github.com/ethereum/wiki/wiki/Sharding-FAQs

SIDECHAINS

BITCOIN SIDECHAINS

Idea: If you can't speed up the bitcoin blockchain, why not create multiple blockchains with approx. 10 minute block times?

One could move their bitcoin over to a faster, less-secure blockchain for purchasing their morning coffee.

Pros:

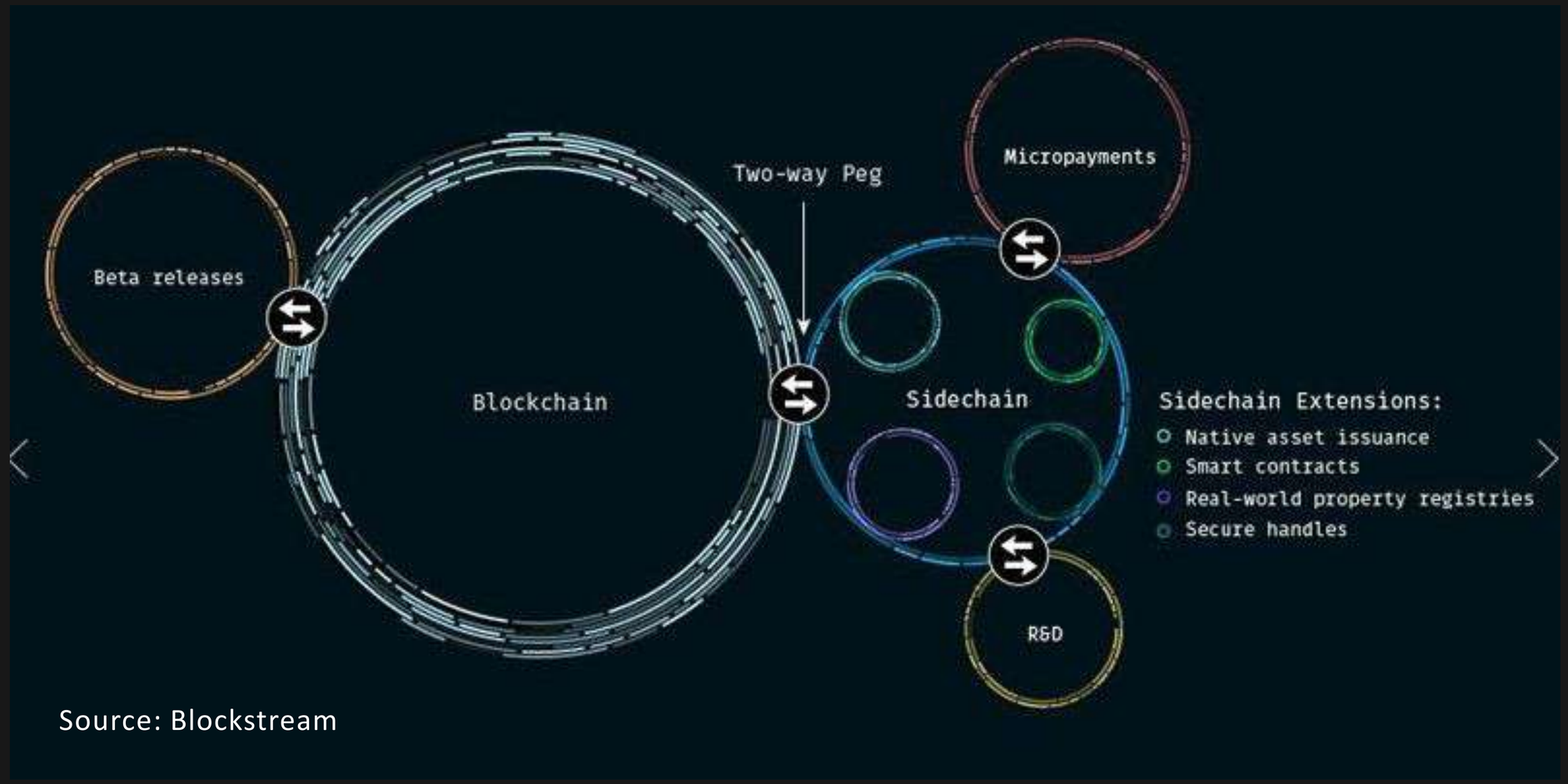
Less things on bitcoin blockchain, but can still be pegged to it.

Cons:

Loses security as hashing power is spread over multiple chains.

SIDECHAINS

BLOCKCHAIN FUNDAMENTALS



Source: Blockstream

SCALABILITY SUMMARY

BLOCKCHAIN FUNDAMENTALS

Bitcoin and other similar Blockchains have an inherent scalability issue:

- If they want to be used on a global scale, they need to support global transaction volumes.
- Can we solve this issue without compromising Bitcoin's original vision of secure, decentralized, trustless payments?

SCALABILITY SUMMARY

BLOCKCHAIN FUNDAMENTALS

1. **Blocksize Capacity Increase**
   a. Small scalability boost with larger blocks.
   b. Centralization risk as minimum server requirements for nodes increases.
2. **Segregated Witness (SegWit)**
   a. Small scalability boost since blocks don't need to store signatures.
3. **Sidechains**
   a. Potential for large scalability boost
   b. Potential novel sidechains with better scalability (yet to be seen in practice)
4. **Lightning Network**
   a. Large potential for orders-of-magnitude scalability boost.
   b. Significant restructuring of payment process.
   c. Centralization risk due to capital prereqs.

READINGS

What is transaction malleability?

- https://www.coindesk.com/bitcoin-bug-guide-transaction-malleability

What is the Lightning Network?

- https://www.coindesk.com/information/what-is-the-lightning-network

Zero Knowledge Proofs

- https://blog.cryptographyengineering.com/2014/11/27/zero-knowledge-proofs-illustrated-primer/

References

Slides mainly adopted from

- Blockchain @ Berkeley : https://blockchain.berkeley.edu/
- Blockchain @ Princeton : http://bitcoinbook.cs.princeton.edu/

Lecture 9

ALTERNATIVE CONSENSUS MECHANISMS

LECTURE OVERVIEW

1. DISTRIBUTED SYSTEMS
2. NAKAMOTO CONSENSUS
3. PROOF-OF-STAKE
4. VOTING-BASED CONSENSUS

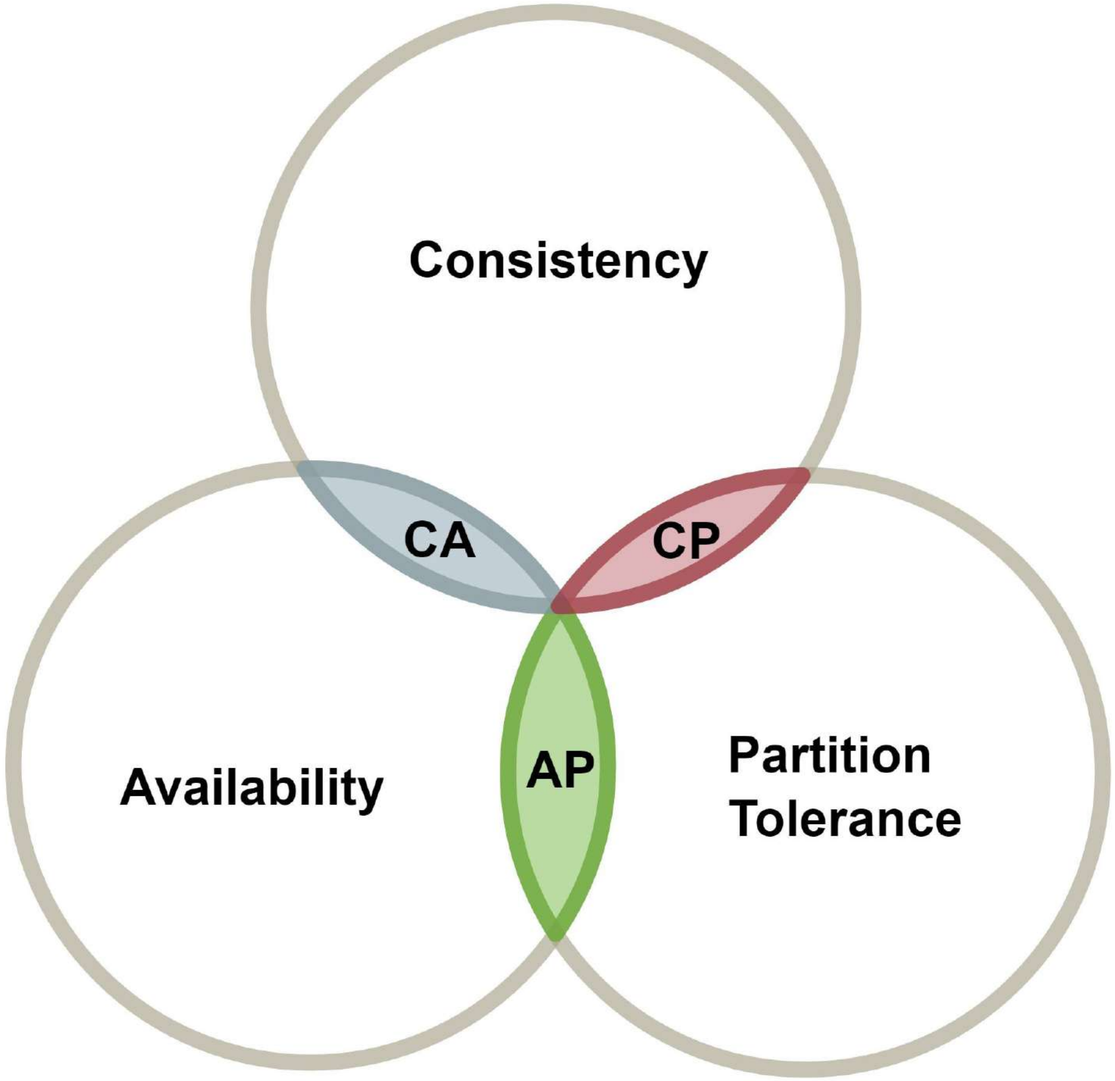
1 DISTRIBUTED SYSTEMS

1.1 CAP THEOREM

CAP THEOREM

INTRODUCTION

Image source:
http://berb.github.io/diploma-thesis/original/resources/cap.svg



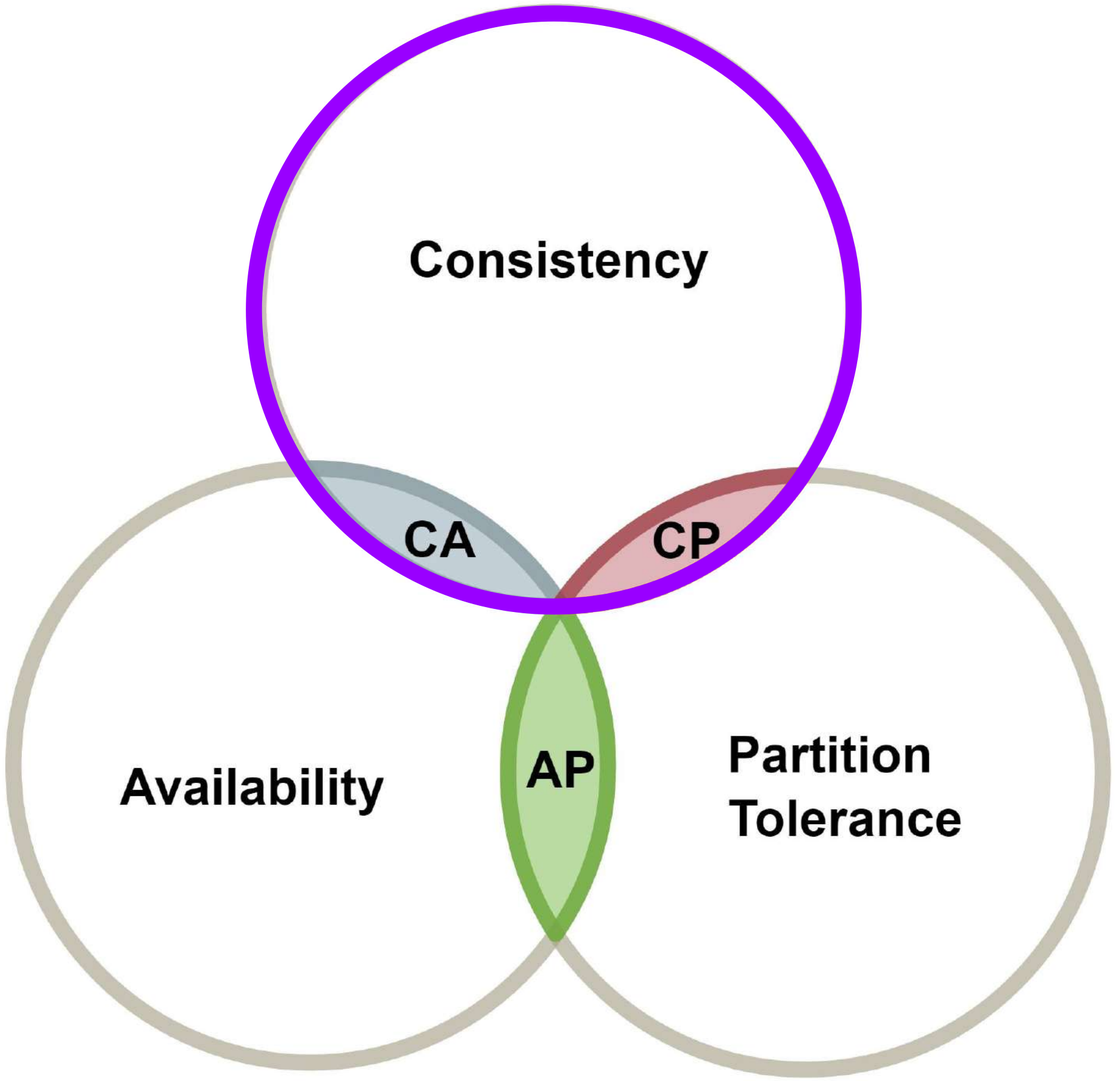
CAP Theorem: Fundamental theorem for any distributed system pertaining to achievable properties

Source:
https://www.youtube.com/watch?v=Jw1iFr4v58M

CAP THEOREM

CONSISTENCY

Image source:
http://berb.github.io/diploma-thesis/original/resources/cap.svg



Consistency:
Every node provides the most recent state

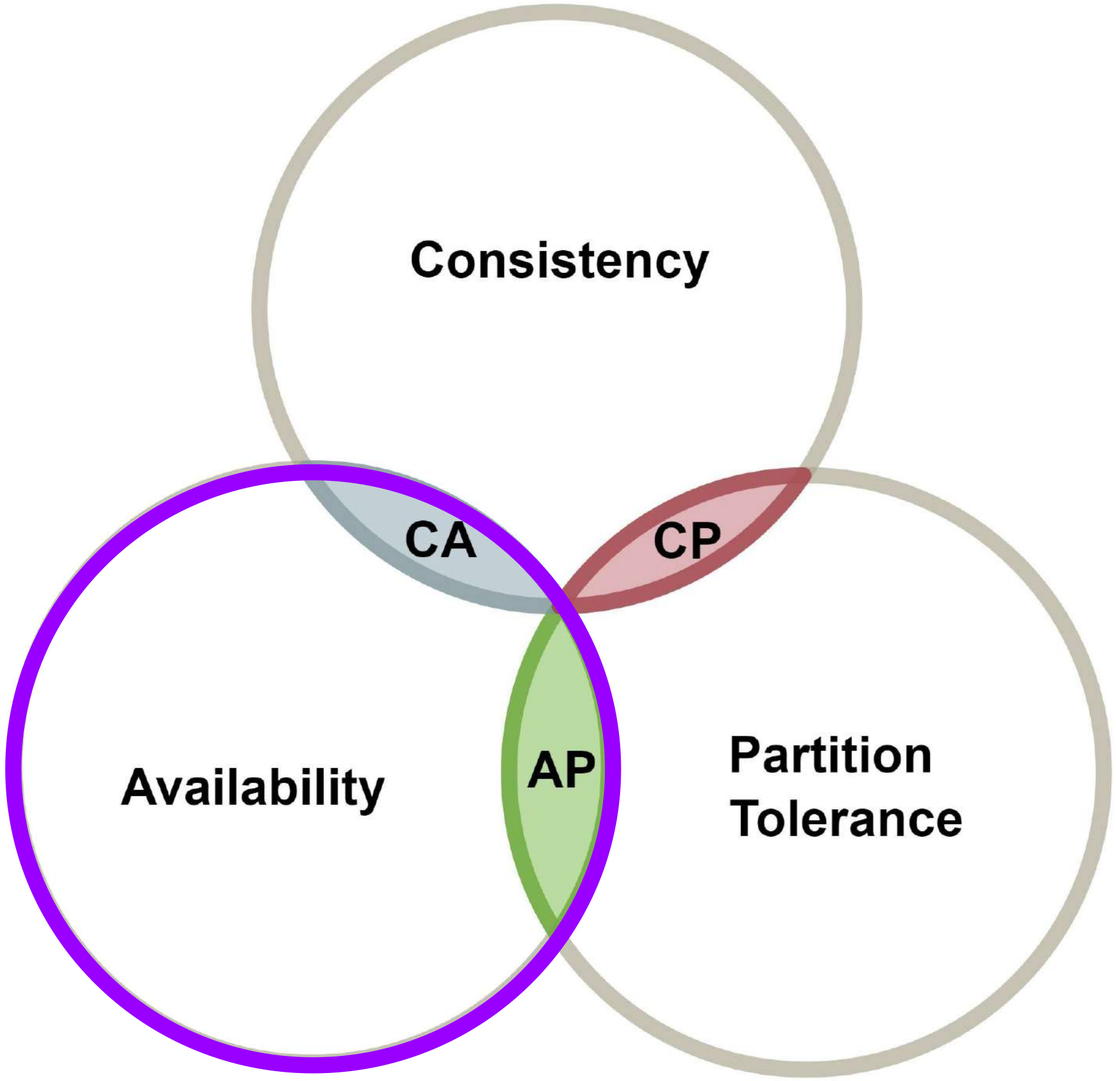
Source:
https://www.youtube.com/watch?v=Jw1iFr4v58M

CAP THEOREM

AVAILABILITY

Image source:
http://berb.github.io/diploma-thesis/original/resources/cap.svg

Availability:
Every node has consistent read and write access



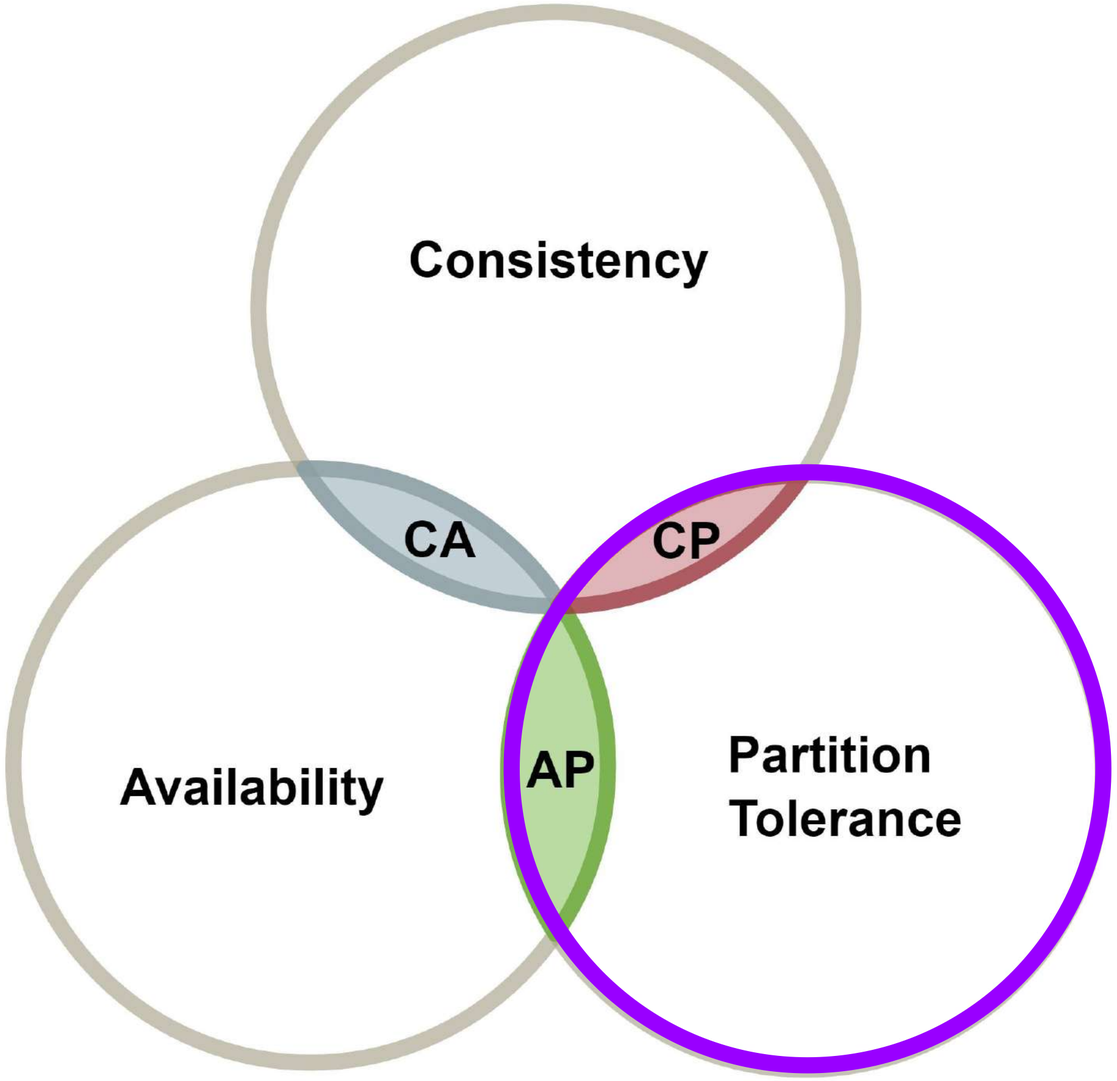
Source:
https://www.youtube.com/watch?v=Jw1iFr4v58M

CAP THEOREM

PARTITION TOLERANCE

Image source:
http://berb.github.io/diploma-thesis/original/resources/cap.svg

Partition Tolerance:
The system works despite partitions in the network



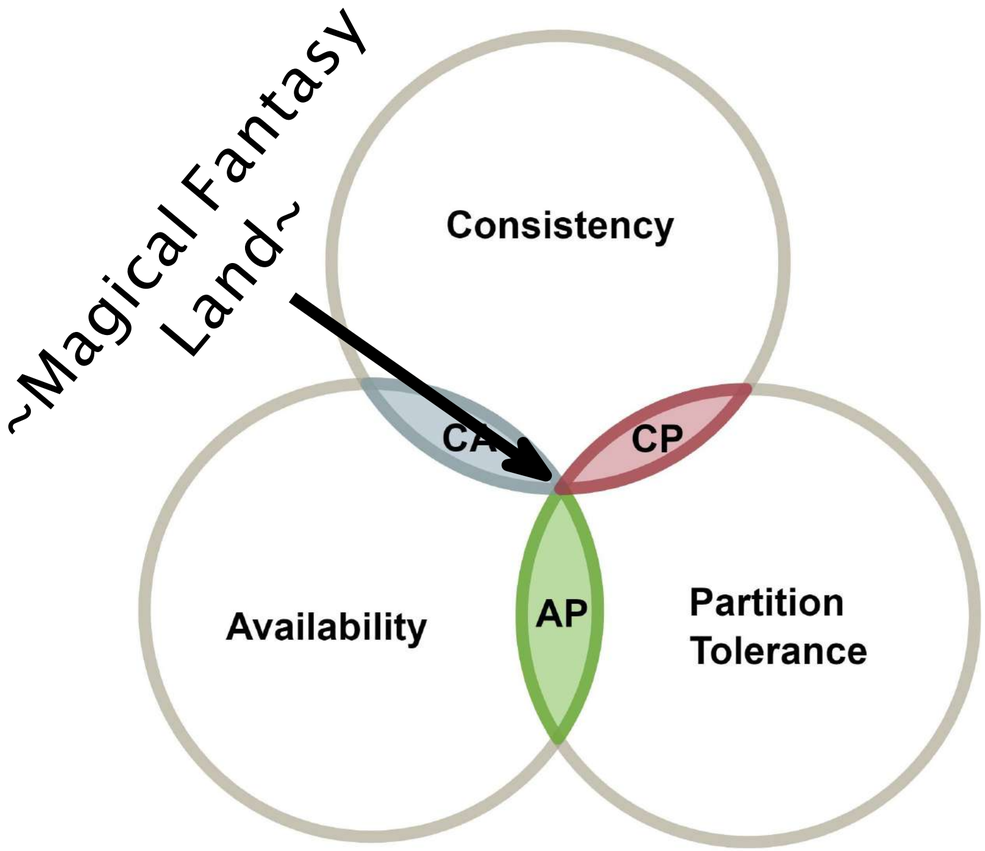
Source:
https://www.youtube.com/watch?v=Jw1iFr4v58M

Image source:
http://berb.github.io/diploma-thesis/original/resources/cap.svg

CAP THEOREM

LIMITATIONS

Can only have **two of three**

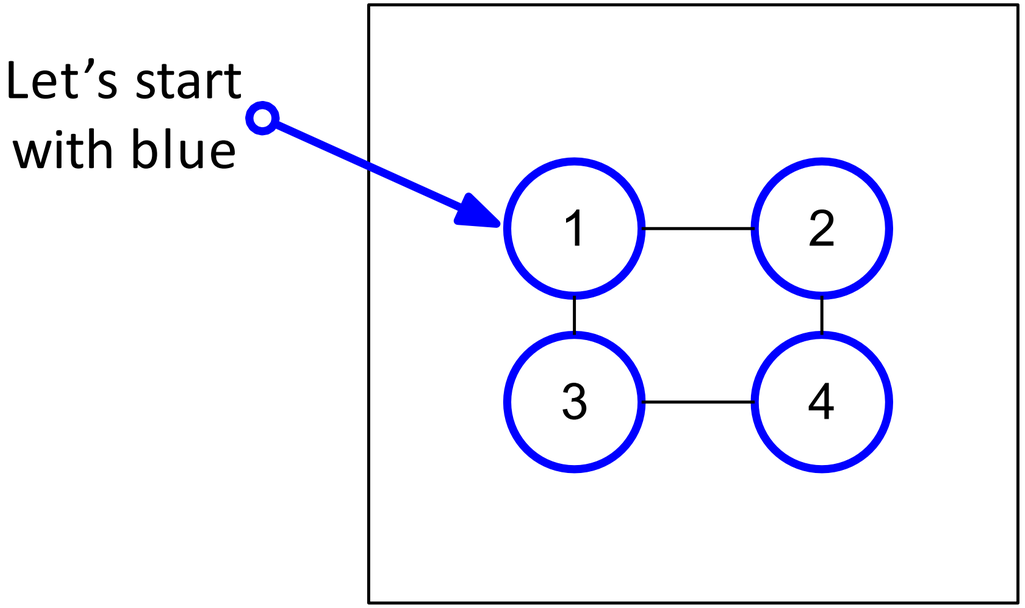


Source:
https://www.youtube.com/watch?v=Jw1iFr4v58M

CAP THEOREM

PROOF

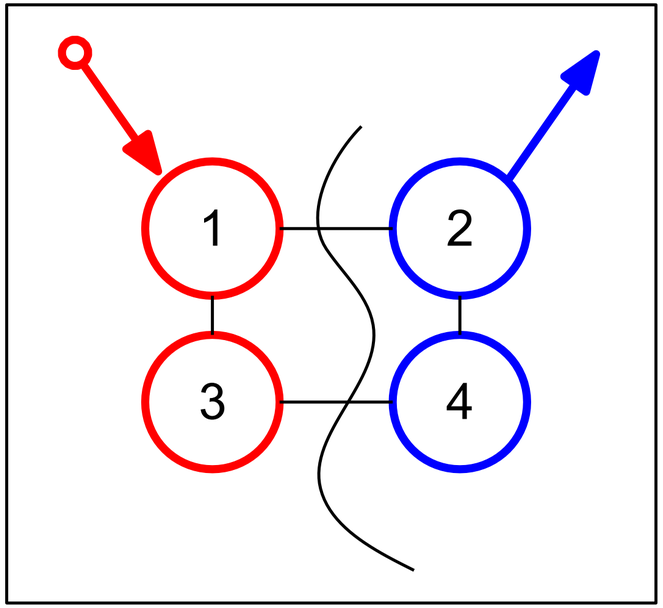
This network keeps track of my favorite color.
I will write to it and read from it.
Nodes may go offline, forming partitions.



CAP THEOREM

PROOF

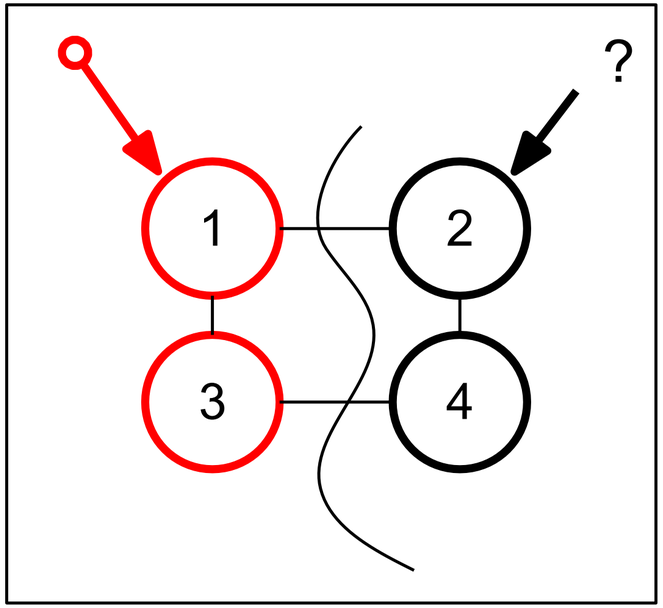
Partition Tolerant +
Available = **Not Consistent**



CAP THEOREM

PROOF

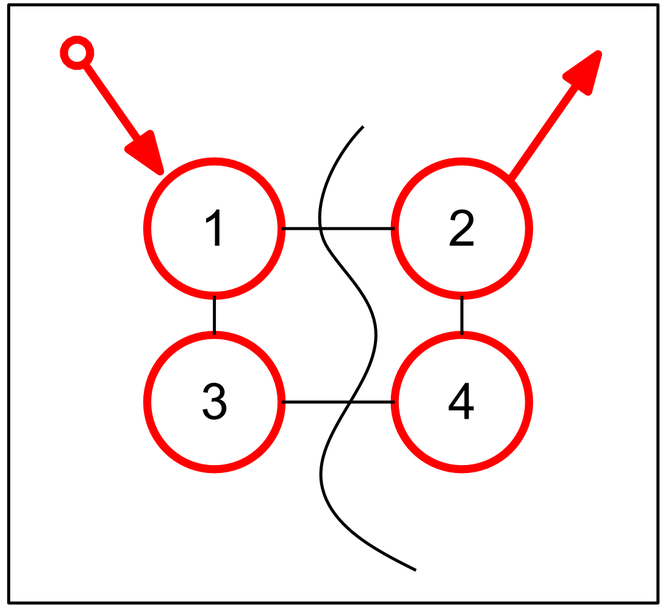
Partition Tolerant +
Consistent = **Not Available**



CAP THEOREM

PROOF

Consistent + Available =
Not Partition Tolerant



1.2 BYZANTINE GENERALS' PROBLEM

BYZANTINE FAULT TOLERANCE

BYZANTINE GENERALS' PROBLEM

Byzantine Generals' Problem: generals stationed around a city are trying to coordinate an attack

- success = they **all do the same thing** (retreat or attack)
- send messages to each other
- some messages may be sent from saboteurs

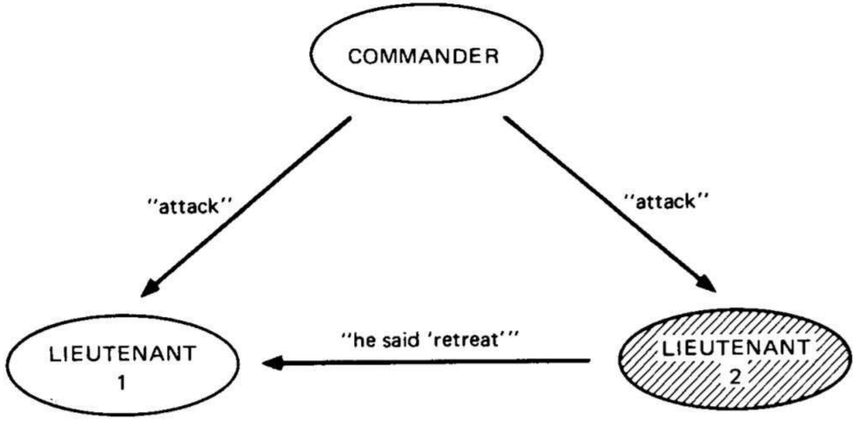


Fig. 1. Lieutenant 2 a traitor.

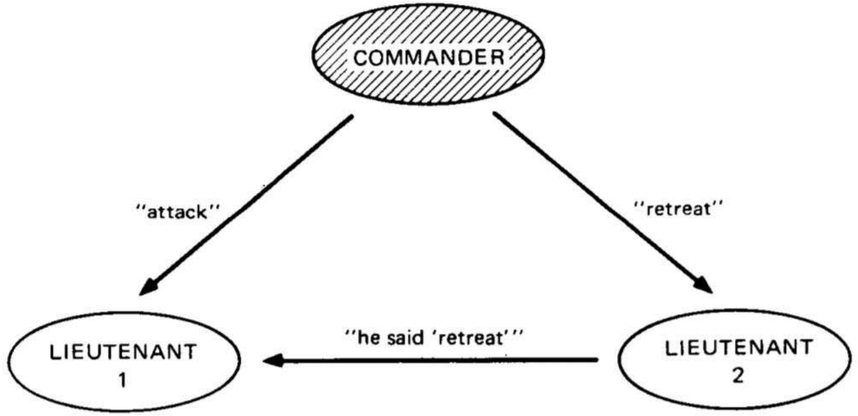
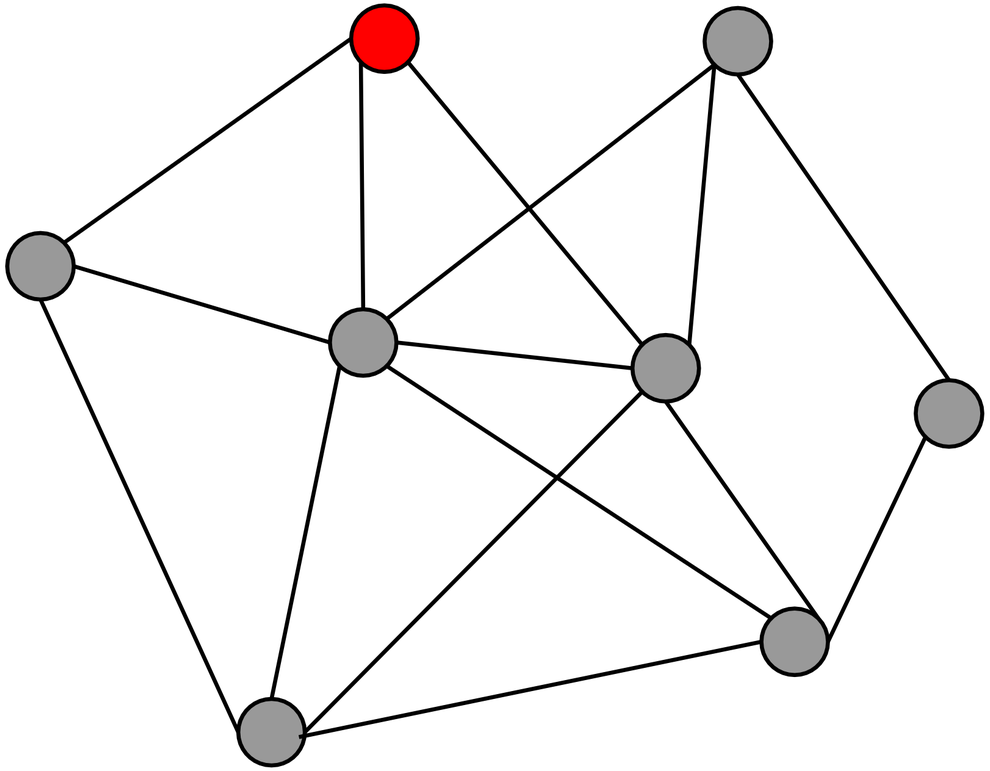


Fig. 2. The commander a traitor.

Source: http://www-inst.eecs.berkeley.edu/~cs162/fa12/hand-outs/Original_By_zantine.pdf

CLICKER QUESTION

CAN THEY SUCCEED?

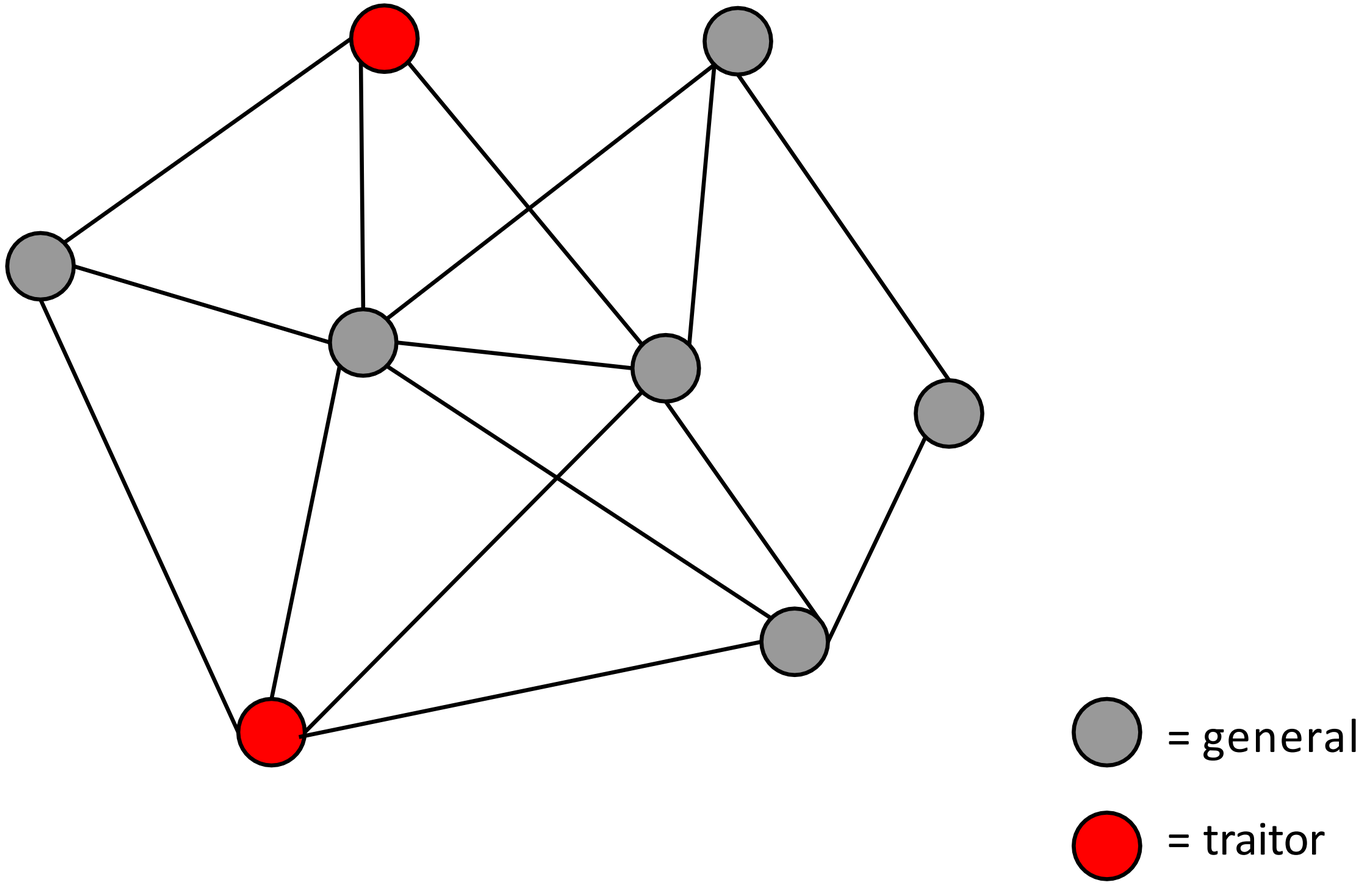


= general

= traitor

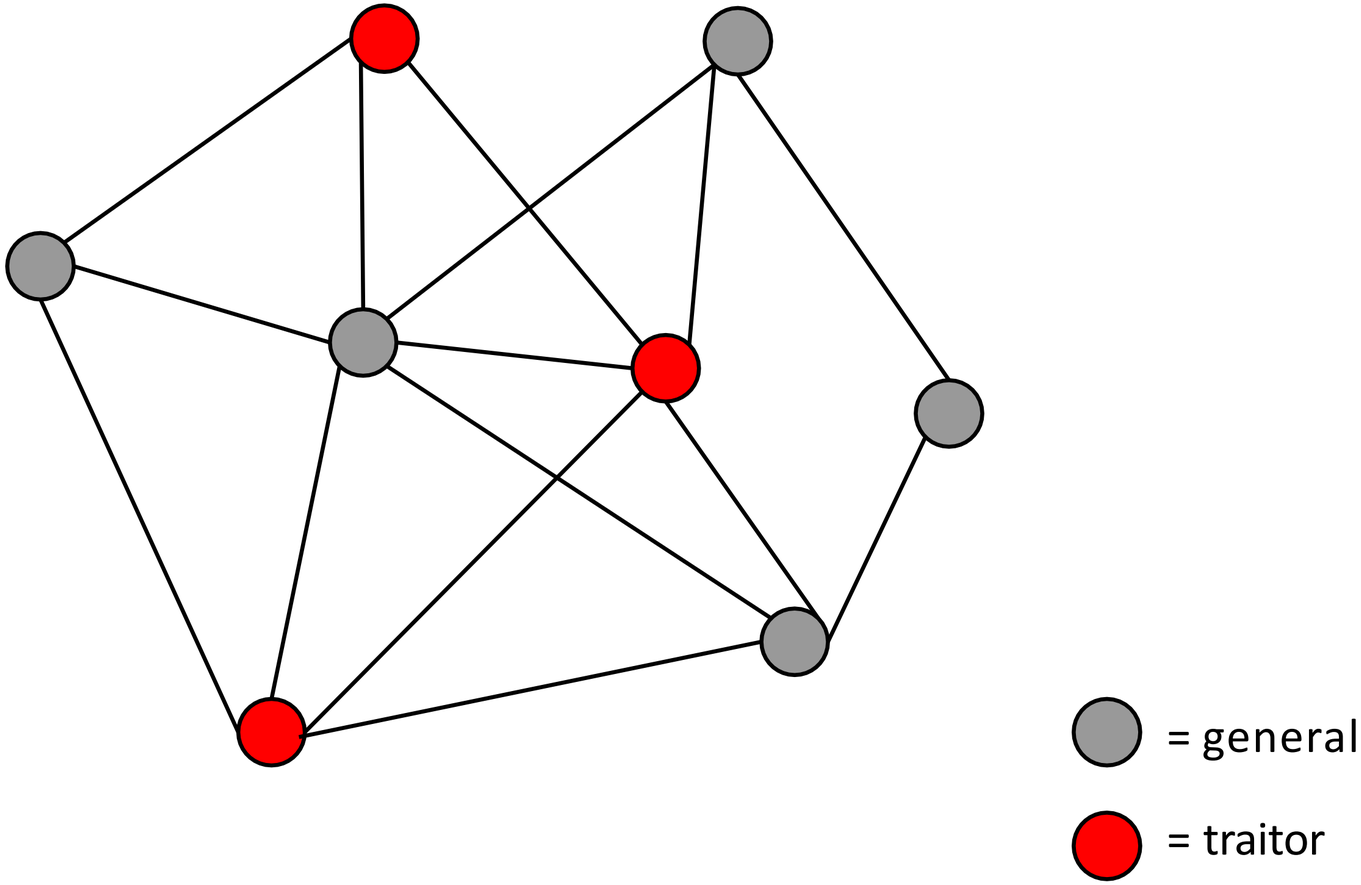
CLICKER QUESTION

CAN THEY SUCCEED?



CLICKER QUESTION

CAN THEY SUCCEED?



BYZANTINE FAULT TOLERANCE

BYZANTINE GENERALS' PROBLEM

Byzantine Generals' Problem:

- How do we ensure that all honest generals make the same decision?
  - No solution for ⅓ or greater corruption
- Solution to all remaining cases: **Practical Byzantine Fault Tolerance**

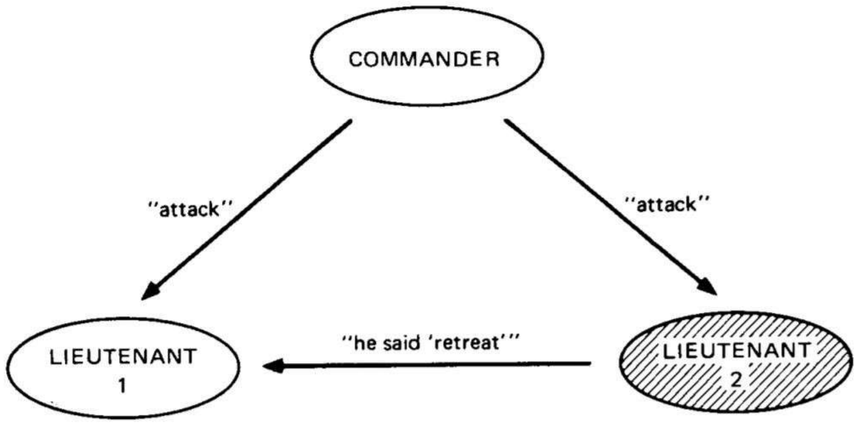


Fig. 1. Lieutenant 2 a traitor.

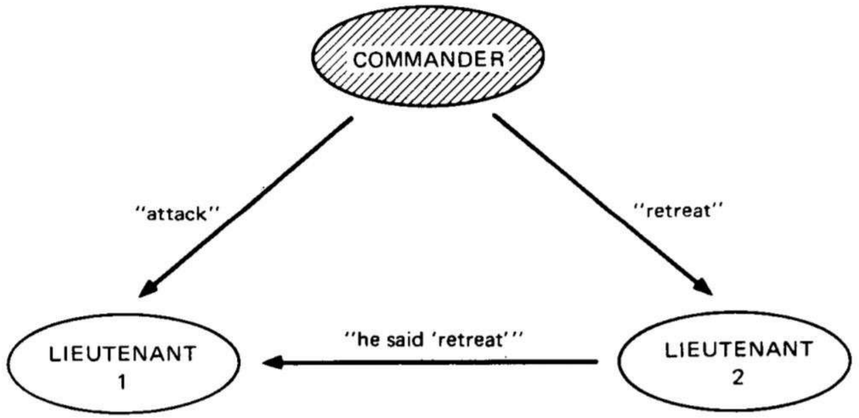


Fig. 2. The commander a traitor.

Source:
http://www-inst.eecs.berkeley.edu/~cs162/fa12/hand-outs/Original_Byzantine.pdf

BYZANTINE FAULT TOLERANCE

DISTINGUISHING FAULT CATEGORIES

Assumptions of fault tolerant systems vs Byzantine fault tolerant systems:

- **Fail-stop fault:** Nodes can crash, not return values, crash detectable by other nodes
- **Byzantine fault:** Nodes can do all of the above *and* send incorrect/corrupted values, corruption or manipulation harder to detect

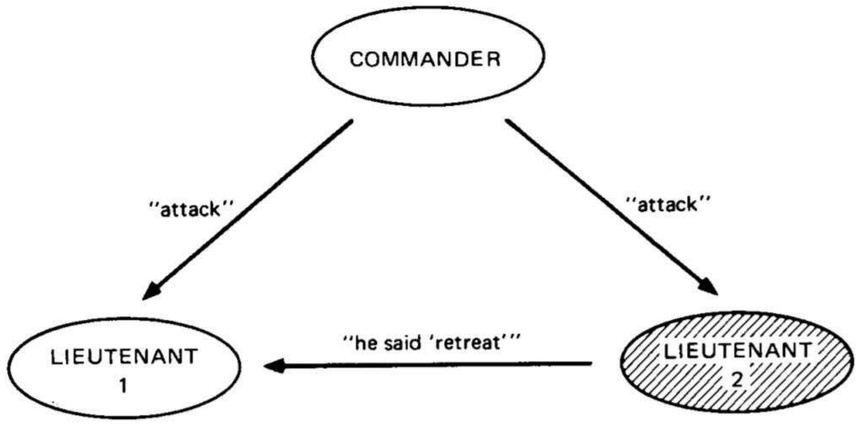


Fig. 1. Lieutenant 2 a traitor.

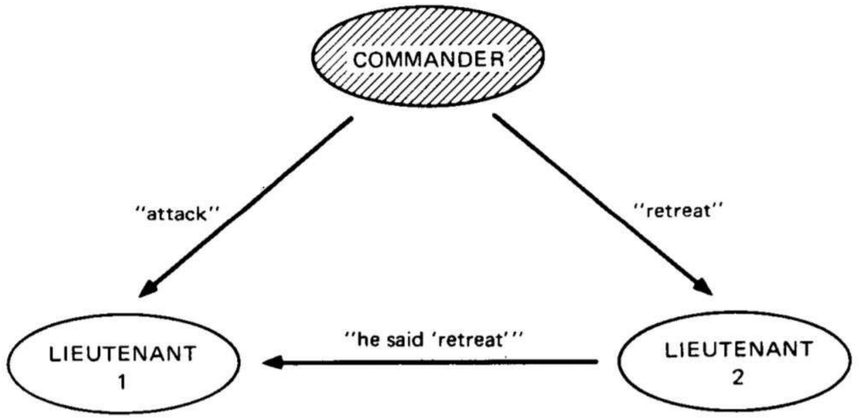


Fig. 2. The commander a traitor.

BYZANTINE FAULT TOLERANCE

BFT IN BLOCKCHAIN

Byzantine Generals	Blockchain
Geographic distance	Distributed network
Generals	Nodes
Traitor generals	Faulty or malicious nodes
Unreliable messengers	Messages between nodes dropped or corrupted; unreliable network
Attack or retreat	Consensus on history (e.g. transactions)

2

NAKAMOTO CONSENSUS

NAKAMOTO CONSENSUS

GENERALIZATION

Quick review:

- Elect leader through some “lottery”
- Leader creates next block
- Others vote implicitly by including block in their chain

Specifying the lottery type:

What resource is being consumed?

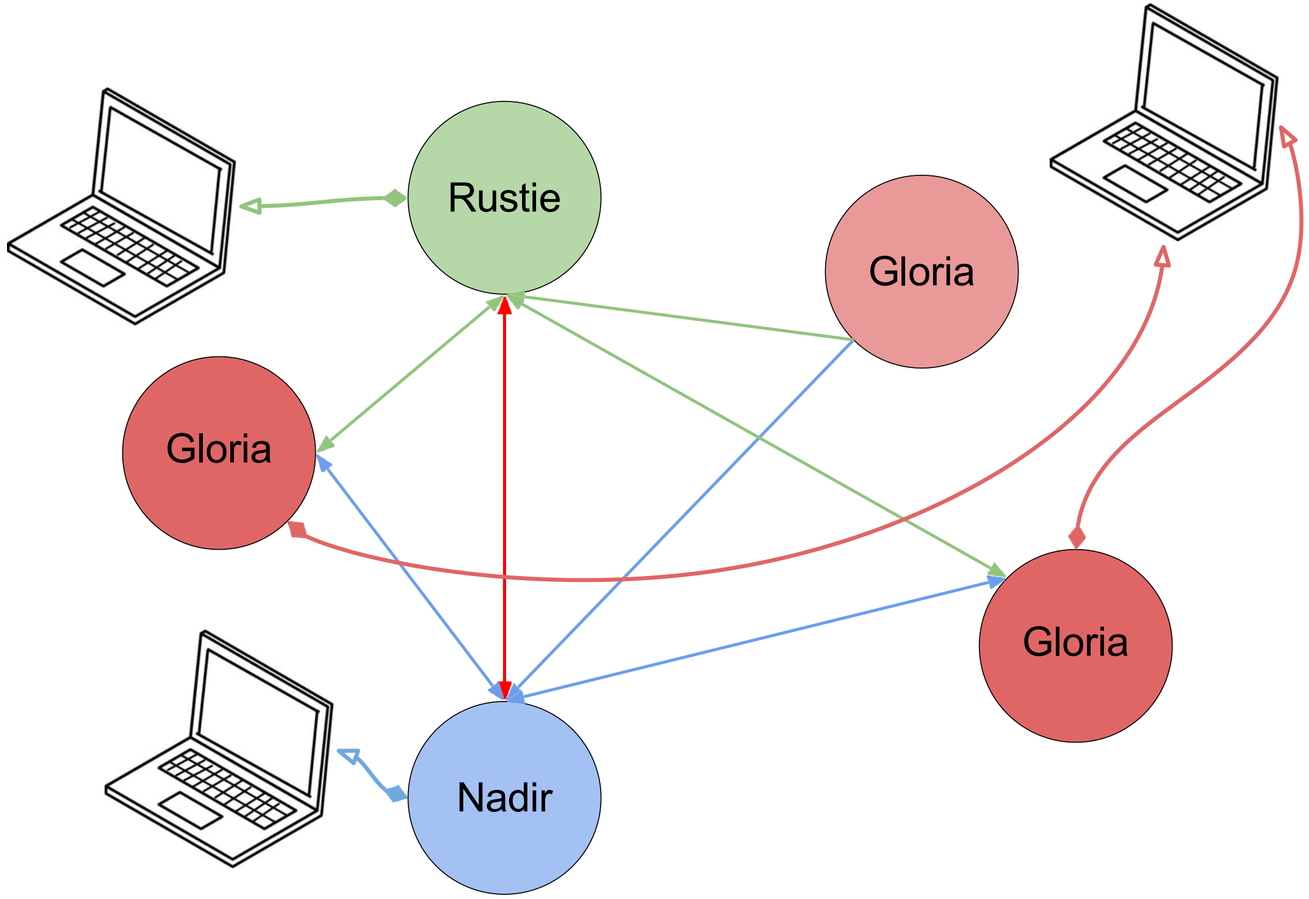
NAKAMOTO CONSENSUS

PROOF-OF-WORK

Quick review:

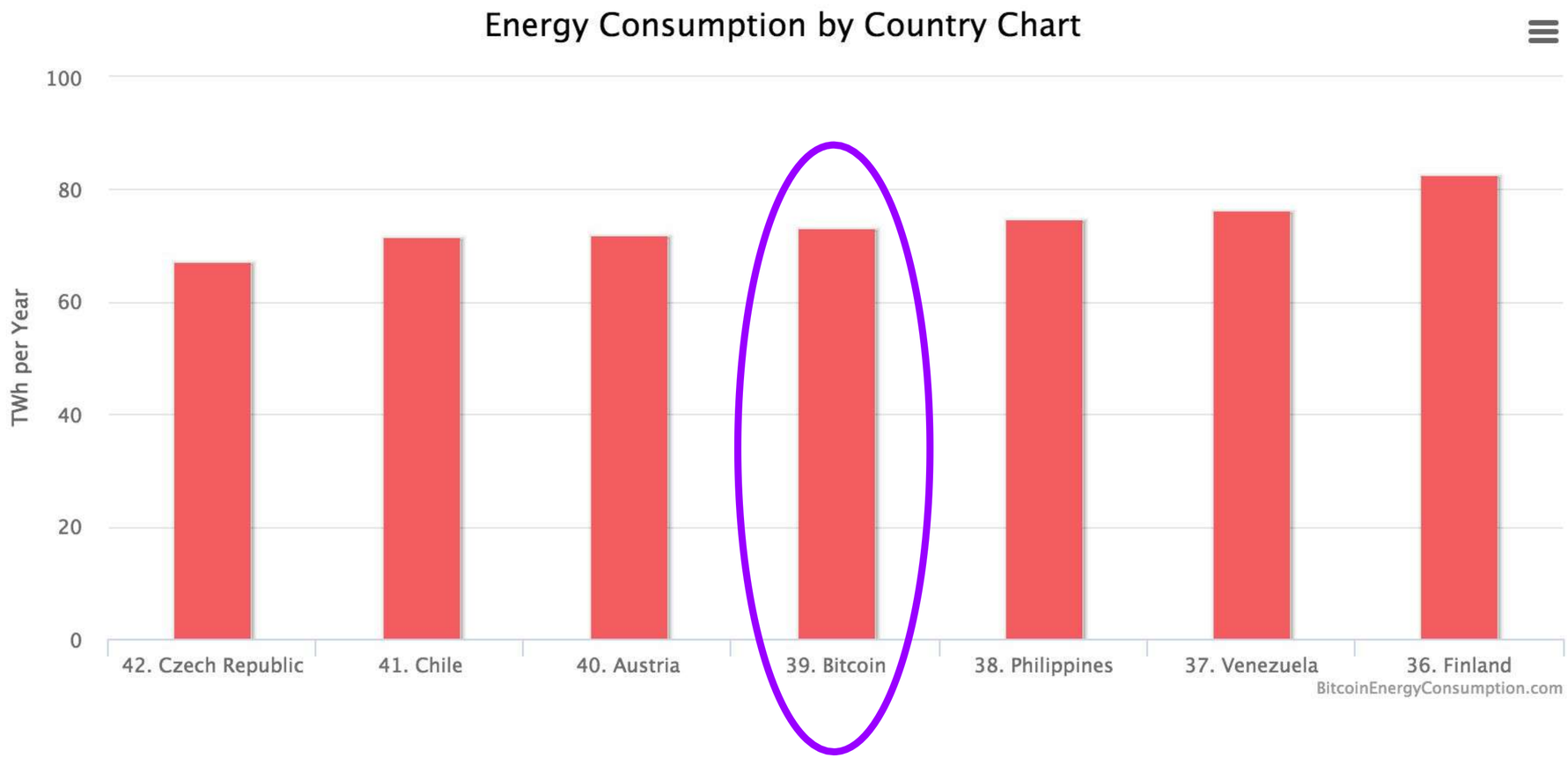
- Miners solve partial preimage hash puzzle
- Prevent Sybil attacks

Resource(s) consumed:
Computational power



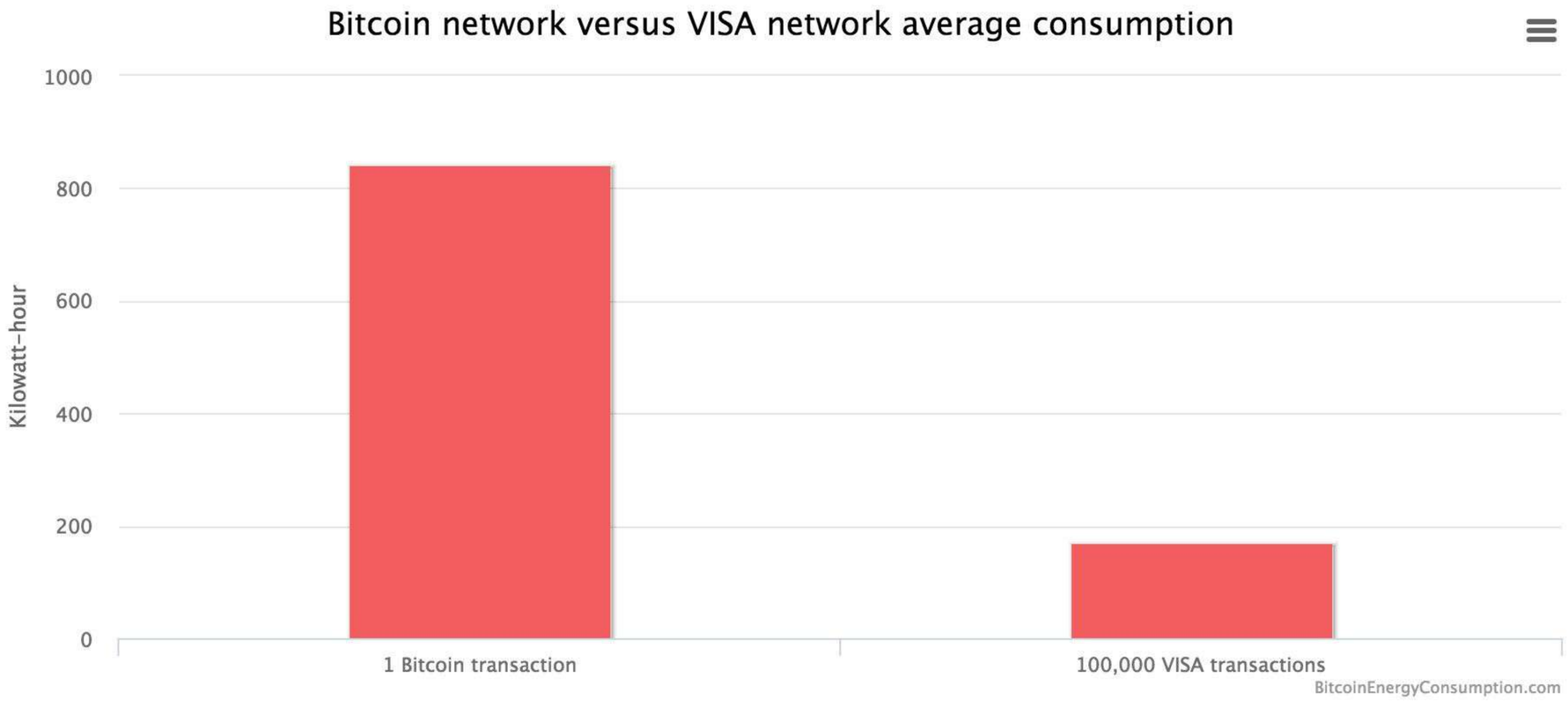
PROOF-OF-WORK SUCKS.

BITCOIN ENERGY USAGE



PROOF-OF-WORK SUCKS.

BITCOIN ENERGY USAGE



PROOF-OF-STAKE

PROOF-OF-SOMETHING-ELSE

Resource(s) consumed: native currency

Source: https://btcmanager.com/buterin-confirms-ethereums-proof-of-stake-75-percent-complete/

PROOF-OF-STAKE

PROOF-OF-SOMETHING-ELSE

Resource(s) consumed: native currency

Quick review:

- Validators instead of “miners”
- Locking up “stake”

PROOF-OF-BURN

PROOF-OF-SOMETHING-ELSE

Resource(s) consumed: currency (potentially not the native currency)

Overview:

- Send coins to irretrievable address
  - More coins burned, higher likelihood of voting
- Like Proof-of-Stake, but more hardcore

Bootstrapping mechanism:

- Tie a coin's value to some other coin (e.g. Bitcoin) by demonstrating that users willingly burn Bitcoin to receive this other coin

Source: https://www.reddit.com/r/dogecoin/comments/2bhiqh/with_all_this_talk_about_proof_of_burn_i_thought/

PROOF-OF-SPACE

PROOF-OF-SOMETHING-ELSE

Resource(s) consumed: storage space

Source: https://hackernoon.com/crypto-review-siacoin-sc-b1d0f0a5c78f

Source: https://medium.com/@tokenlot/filecoin-ico-open-to-accredited-investors-only-b7937f24de44

Source: https://bitcoinist.com/storj-new-decentralized-storage-solution/

PROOF-OF-SPACE

PROOF-OF-SOMETHING-ELSE

Resource(s) consumed: storage space

Overview:

- Use disk space to solve challenge
- Nowadays used for <u>file storage</u>
- Variations on implementations:
  - Let users receive coins after voting on blocks (Also known as Proof-of-Capacity)
  - Give rewards directly for storing information (Filecoin, Storj, Sia)

Source: https://hackernoon.com/crypto-review-siacoin-sc-b1d0f0a5c78f

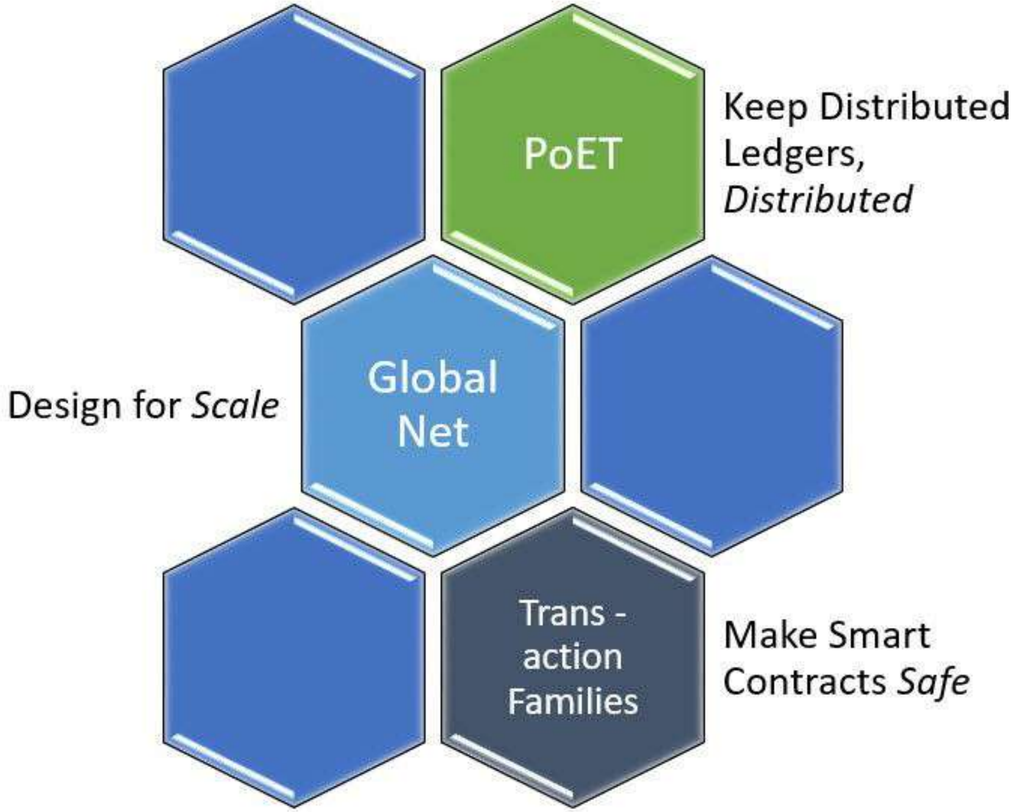
Source: https://medium.com/@tokenlot/filecoin-ico-open-to-accredited-investors-only-b7937f24de44

Source: https://bitcoinist.com/storj-new-decentralized-storage-solution/

PROOF-OF-ELAPSED-TIME

PROOF-OF-SOMETHING-ELSE

Resource(s) consumed: time



Source: https://www.altoros.com/blog/wp-content/uploads/2017/02/Hyperledger-Intel-Sawtooth-Lake-difference.jpg

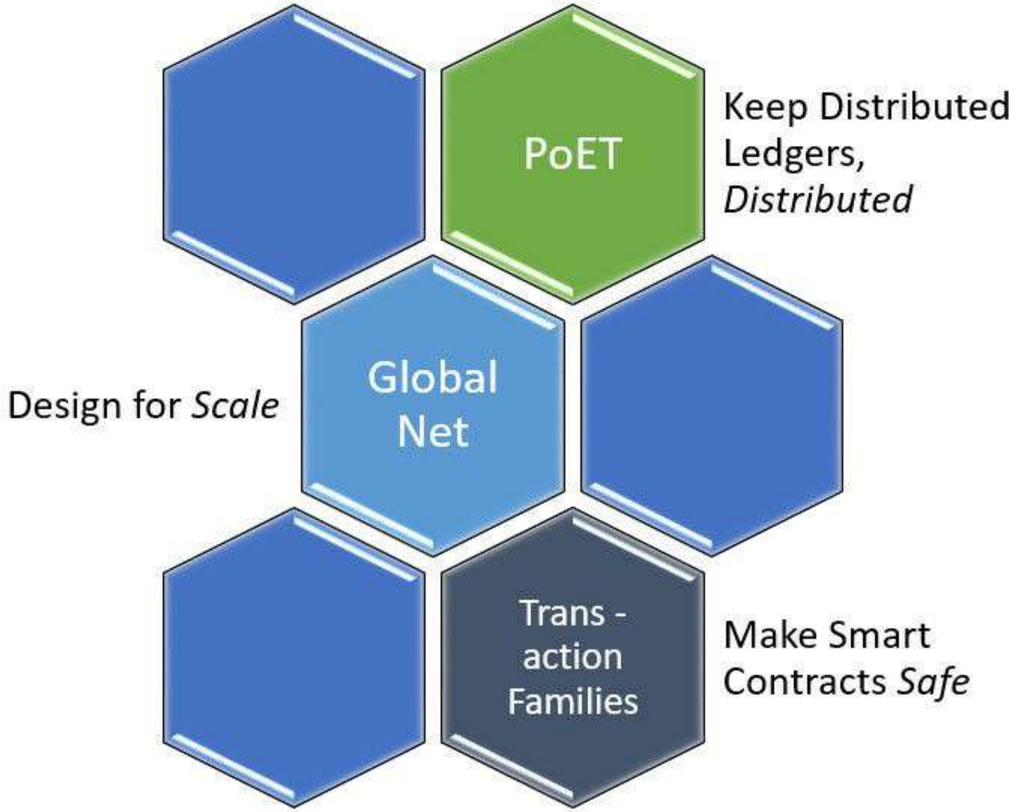
PROOF-OF-ELAPSED-TIME

PROOF-OF-SOMETHING-ELSE

Resource(s) consumed: time

Overview:

- Spend time instead of mining power
  - Trusted Execution Environments (TEEs), particularly Intel SGXs
  - Verifiable delay functions (VDFs)
- Ask your SGX to wait some amount of time decided by the machine, wait for an attribution that it has indeed waited that much time
  - Let the first person to complete this random wait win
- Based on assumptions of randomness and trust in manufacturer (Intel)



Source: https://www.altoros.com/blog/wp-content/uploads/2017/02/Hyperledger-Intel-Sawtooth-Lake-difference.jpg

3

PROOF OF STAKE

POW VS POS?

DIFFERENCES

Miners have voting power proportional to their computational power

Validators are stakeholders with voting power proportional to economic stake locked up.

PROOF-OF-STAKE

COSMOS & TENDERMINT

- Nodes must **stake tokens** to participate in the consensus process
  - Staked tokens cannot be spent
  - Voting power in proportion to how much they stake
  - **Slashing**: destroying staked tokens to punish bad behavior
- Main Idea: **someone who is invested in the network will behave in its best interest**

PROOF-OF-STAKE

COSMOS & TENDERMINT

- **Nodes stake tokens to become validators**
- Proposer (selected validator) proposes a block, likelihood of being selected is proportional to stake
- Block is committed if ⅔ **majority** of validators vote on it
  - Node waits for news from ⅔ before proceeding
- Validators' stake is **slashed (punishment) for bad behavior**
  - Voting duplicitously: voting on 2 blocks at same height
  - Going offline

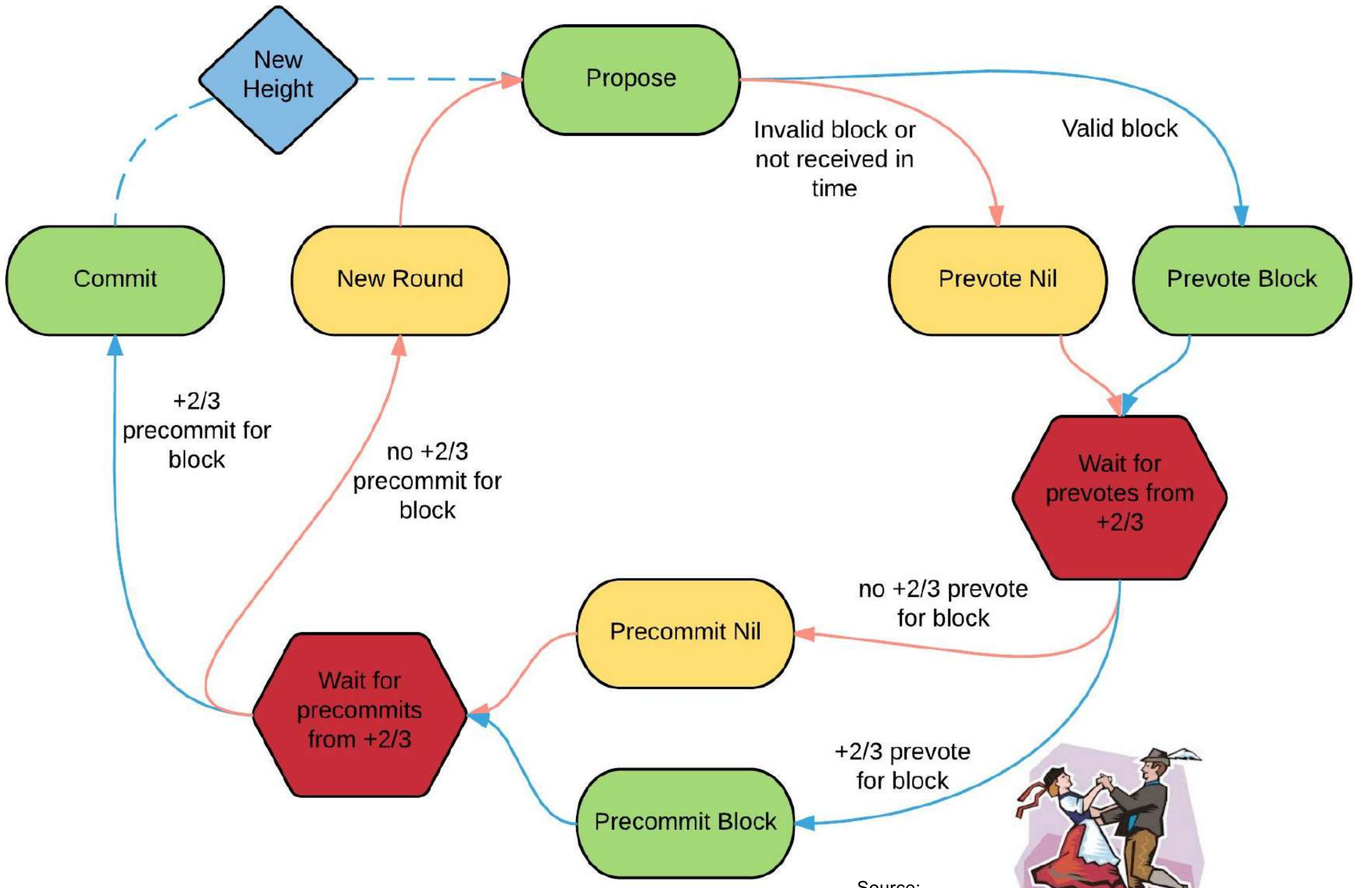
COSMOS

INTERNET OF BLOCKCHAINS

https://tendermint.com/static/docs/tendermint.pdf

PROOF-OF-STAKE

COSMOS &TENDERMINT: MORE DETAIL



Source:
http://tendermint.readthedocs.io/projects/tools/en/master/introduction.html

CLICKER QUESTION

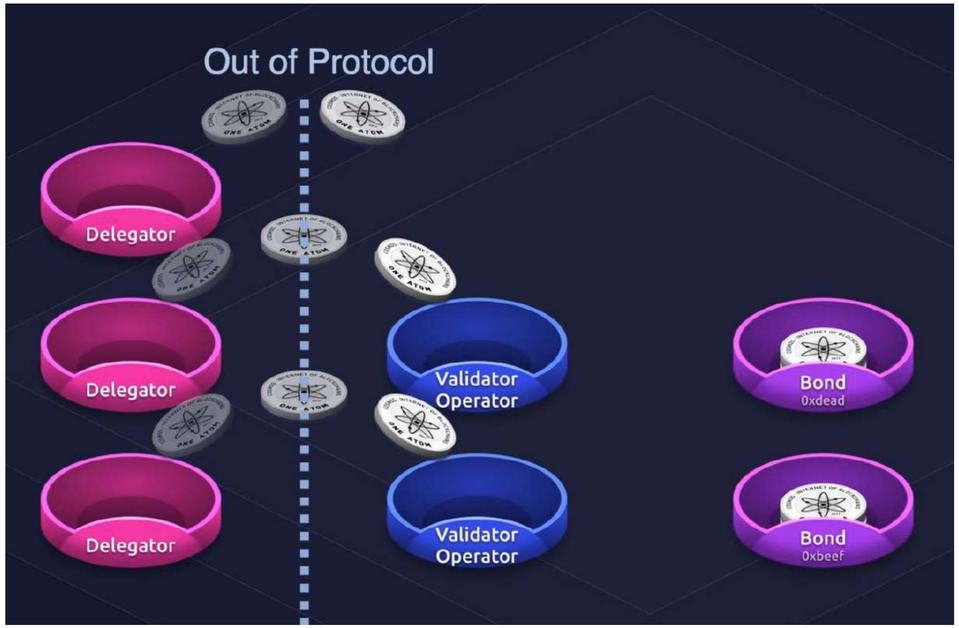
COSMOS & TENDERMINT

- **Nodes stake tokens to become validators**
- Proposer (selected validator) proposes a block, likelihood of being selected is proportional to stake
- Block is committed if ⅔ **majority** of validators vote on it
  - Node waits for news from ⅔ before proceeding

DELEGATED PROOF-OF-STAKE

COSMOS & TENDERMINT

- aka Bonded Proof-of-Stake
- **Delegated Proof-of-Stake**: nodes can delegate stake to validators to pool their voting power
  - no need to maintain a full node, more tokens staked
  - Delegated stake also slashed if validator's is slashed
  - Process of switching delegated stake is built in to protocol (does not encourage centralization)
- Delegated Proof-of-Stake: can also mean token-holders vote on delegators to propose blocks



PROOF-OF-STAKE

IMPLEMENTATIONS

- **Casper:** Ethereum's research on PoS implementation
  - ***Casper the Friendly Finality Gadget*** (Vitalik)
    - a PoW/PoS hybrid that does PoS checkpoints every 50 blocks
  - ***Casper the Friendly GHOST***: Correct-by-Construction (Vlad)
    - designs the protocol from scratch, deriving it from defined properties
    - focused on safety
    - maintained validator set

casper

DISCUSSION QUESTION

What are some of the **advantages** of Proof-of-Stake compared to Proof-of-Work?

What are some possible **downsides** you see to Proof-of-Stake?

PROOF-OF-STAKE

THEORETICAL ADVANTAGES

- **Environmentally friendly**: does not rely on meaningless computation
- **Scalability**: possible to reach consensus faster
- **Security**: more expensive to reach 51% stake
- **Decentralization**: arguably more decentralized (no mining pools)

Lisk

NEO smart economy

Dash

PIVX PRIVATE INSTANT VERIFIED TRANSACTION

PROOF-OF-STAKE

PROBLEM

In Proof-of-Stake, we have the problem of “the rich get richer”

- Large stakeholders stake their balances
- Smaller stakeholders see fewer rewards and lower returns
- Centralizes staking rewards

SOURCE: https://cointelegraph.com/news/the-inevitable-failure-of-proof-of-stake-blockchains-and-why-a-new-algorithm-is-needed

4 VOTING-BASED CONSENSUS MECHANISMS

PAXOS

OVERVIEW

Paxos is a Greek island

but also . . .

A CONSENSUS ALGORITHM

Source: http://www.hikingbikingadventures.com/biking-adventures/bicycling-europe-2/bicycling-greece/bicycling-paxos-island/

PAXOS

TERMINOLOGY

Within the Paxon Parliament...

Proposer: legislator, advocates a citizen's request, moves protocol forward
Acceptor: legislator, voter
Learner: remembers and carries out result for citizen

Quorum:

- any majority of Acceptors
- any two Quorums must overlap

PAXOS

MAIN IDEA

Protocol proceeds over several rounds
where each successful round has 2 phases: prepare and accept

1. Citizens talk to Proposer
2. Within Paxon parliament: Proposer, Acceptor, and Learner discuss
   a. Pass decrees
   b. A decree has a number and value
3. Learner talks to citizens

1375: *Πωυδα is the new cheese inspector*

277: *The sale of brown goats is permitted*

37: *Painting on temple walls is forbidden*

PAXOS

KNOWLEDGE OF THE ANCIENTS

There is an obvious correspondence between this database system and the Paxon Parliament:

Parliament		Distributed Database
legislator	↔	server
citizen	↔	client program
current law	↔	database state

Source: https://lamport.azurewebsites.net/pubs/lamport-paxos.pdf

PAXOS IN PRACTICE

ASSUMPTIONS AND REAL WORLD USE

- Nodes do not try to subvert the protocol (messages delivered without corruption)
- Only works for fail-stop (no Byzantine failures) faults
- Failed nodes at any point won't affect the entire network
- Good performance (fast)
- Generally used to replicate large sets of data

RAFT

OVERVIEW

Raft is another consensus mechanism designed to be an alternative to Paxos

- Designed to be more understandable than Paxos
- Leader-based approach
- Easier to implement
- JP Morgan's Quorum: Raft-based consensus

RAFT

HOW IT WORKS

A Raft cluster has one and only one elected **leader**

- Communicates with client directly
- Responsible for managing log replication on the other servers of the cluster
- Leads until it fails or disconnects, in which case a new leader is elected

Leader Election

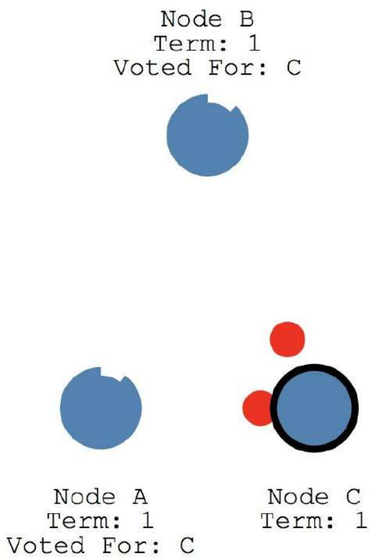
1. Leader sends “heartbeats” to other nodes saying that it is online
2. If other nodes no longer receive “heartbeat,” they start an election cycle (and internal timer)
3. First candidate to timeout becomes new leader

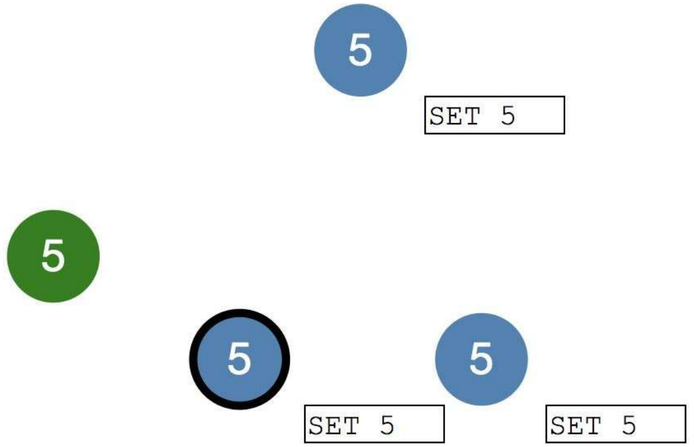
Log Replication

1. Leader accepts client request
2. Leader ensures all other nodes have followed that request

RAFT

THE SECRET LIVES OF DATA





These messages are sent in intervals specified by the heartbeat timeout.

The leader then notifies the followers that the entry is committed.

Source: http://thesecretlivesofdata.com/raft/

Voting Based Consensus

Voting based Consensus Algorithms

Not Byzantine Fault Tolerance

PBFT

OVERVIEW

Practical Byzantine Fault Tolerance,

Miguel Castro and Barbara Liskov

- Fast
- Handle f Byzantine faults in a system with 3f + 1nodes
- BFT-NFS only 3% slower than standard unreplicated NFS

Source:
http://www-inst.eecs.berkeley.edu/~cs162/fa12/hand-outs/Original_Byzantine.pdf

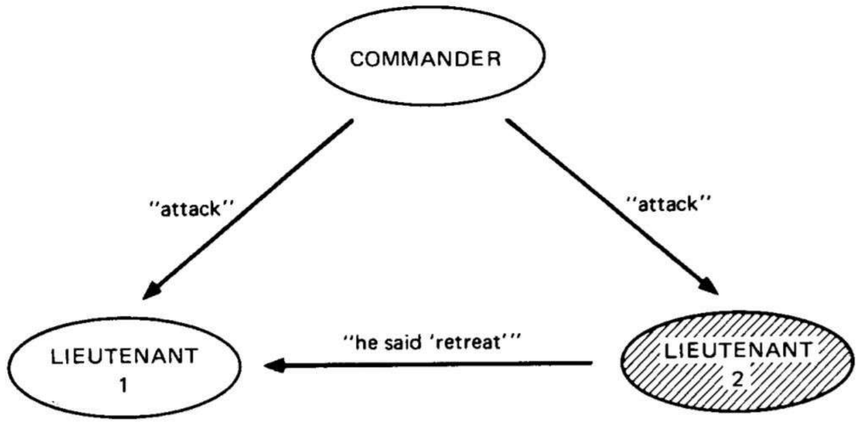


Fig. 1. Lieutenant 2 a traitor.

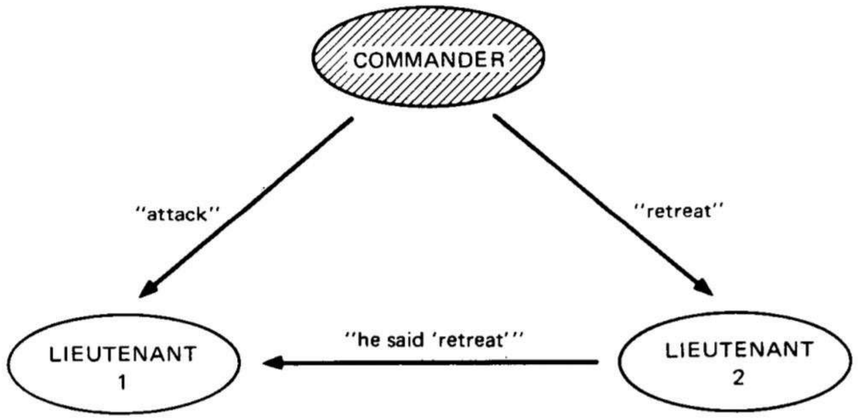
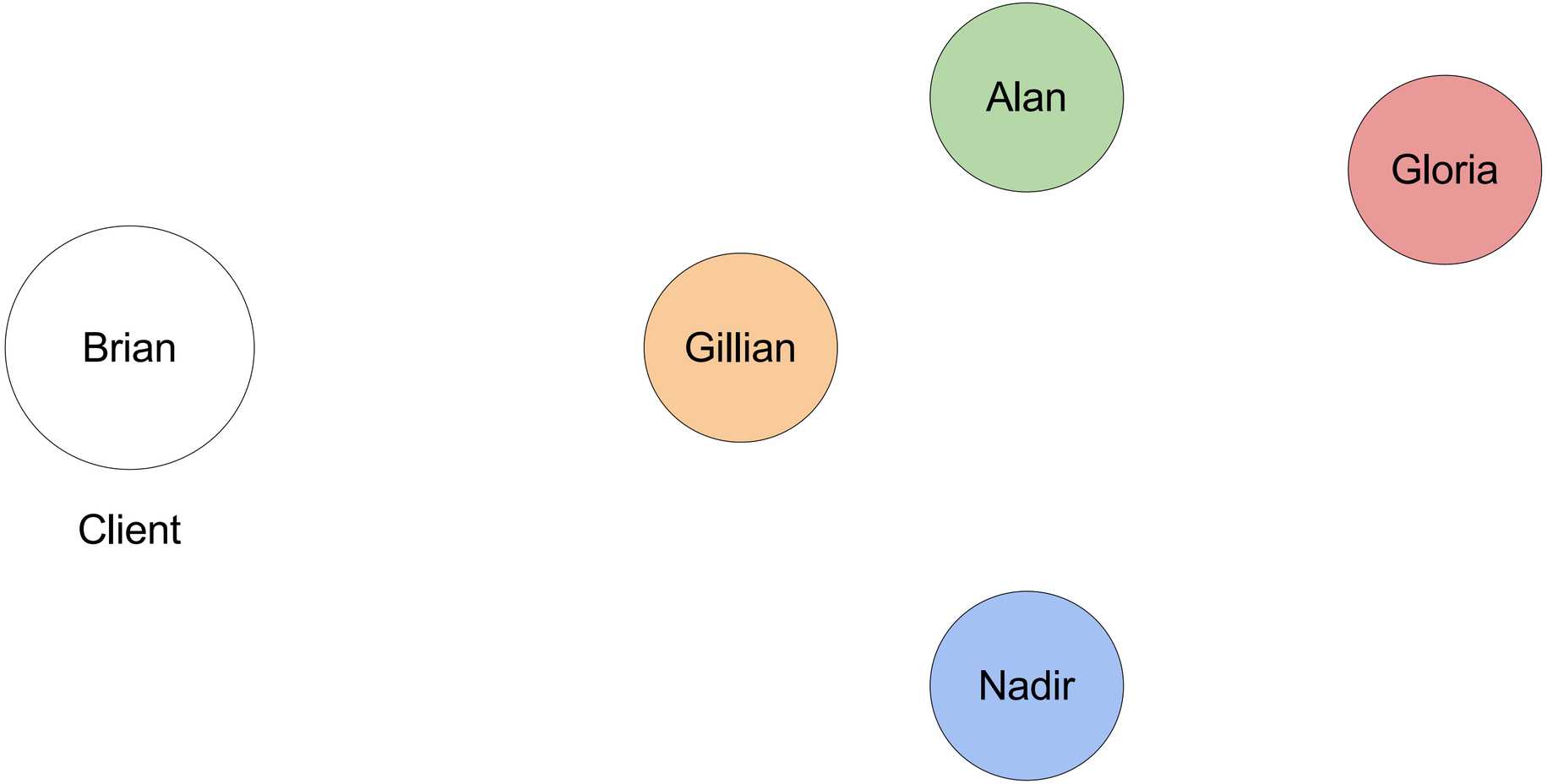


Fig. 2. The commander a traitor.

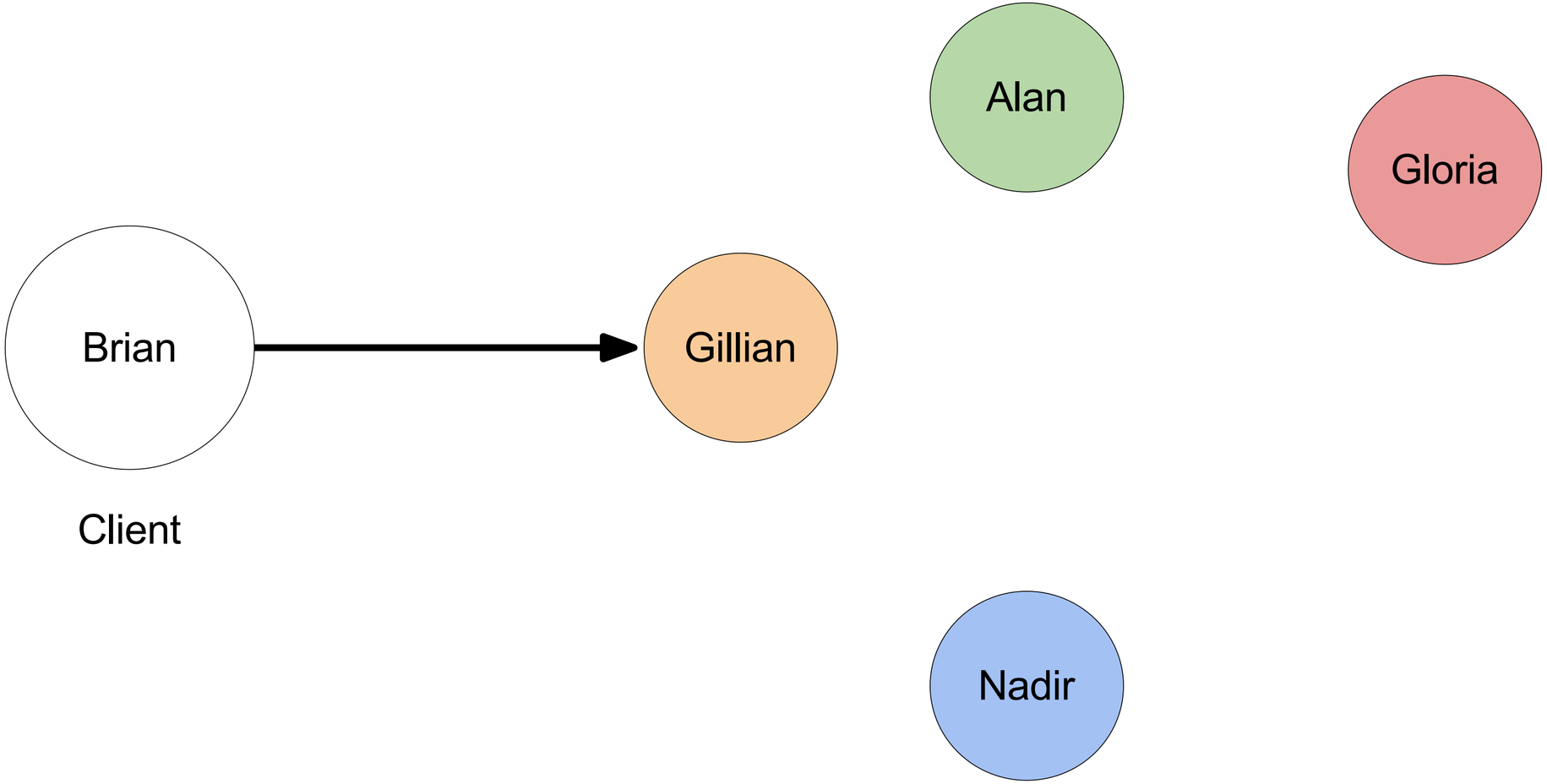
PBFT

BY EXAMPLE



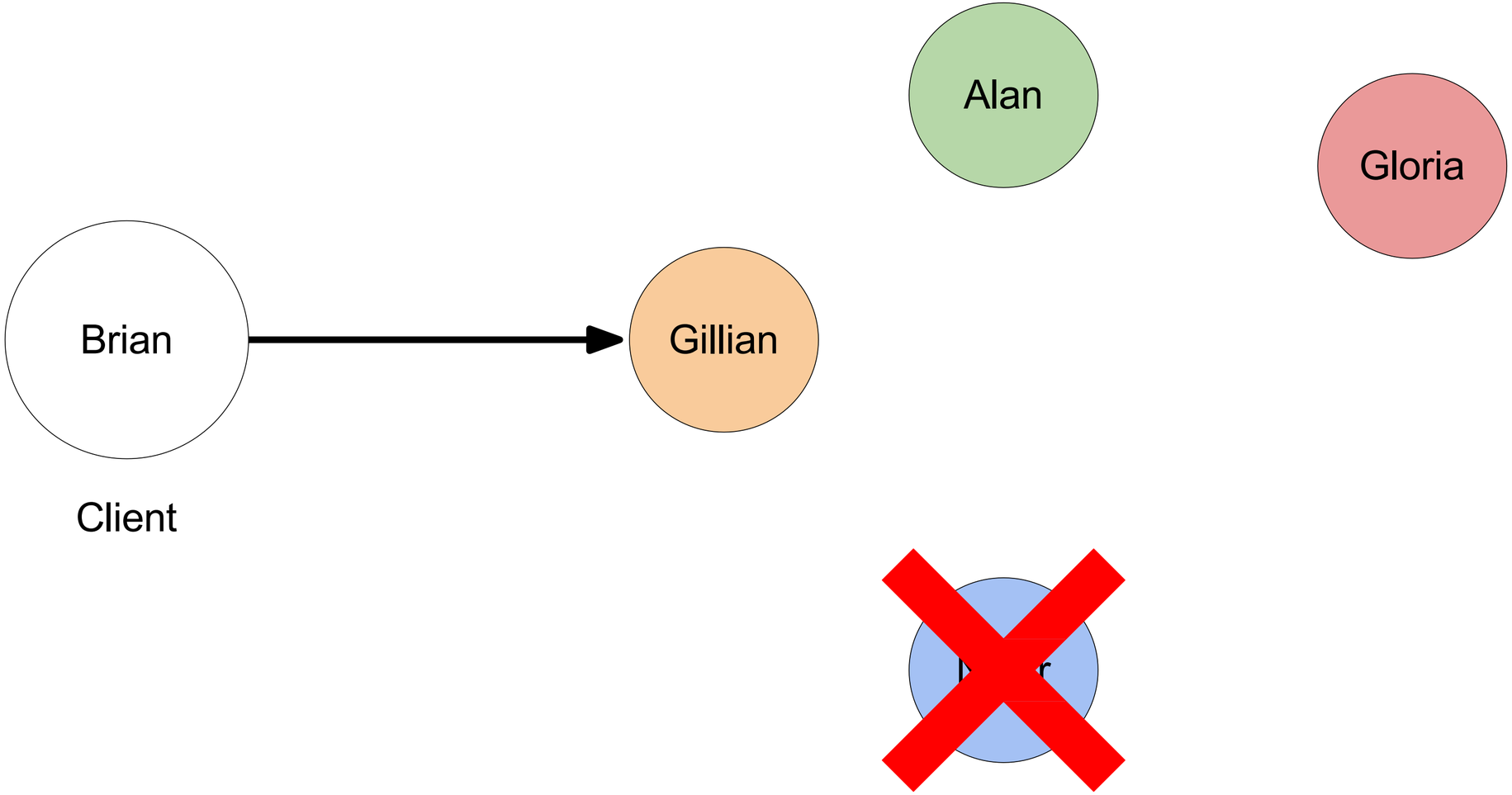
PBFT

BY EXAMPLE: REQUEST



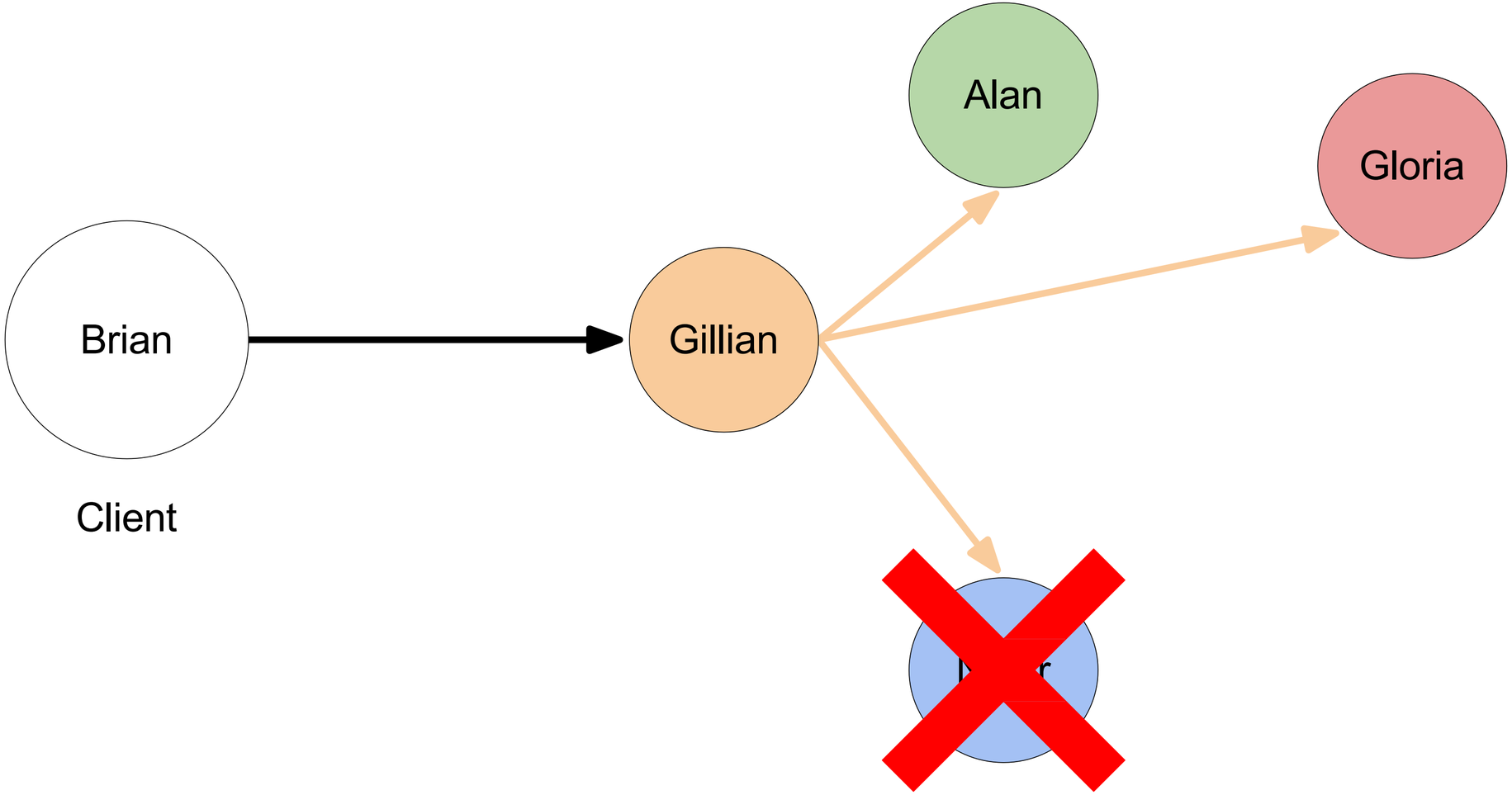
PBFT

BY EXAMPLE: REQUEST



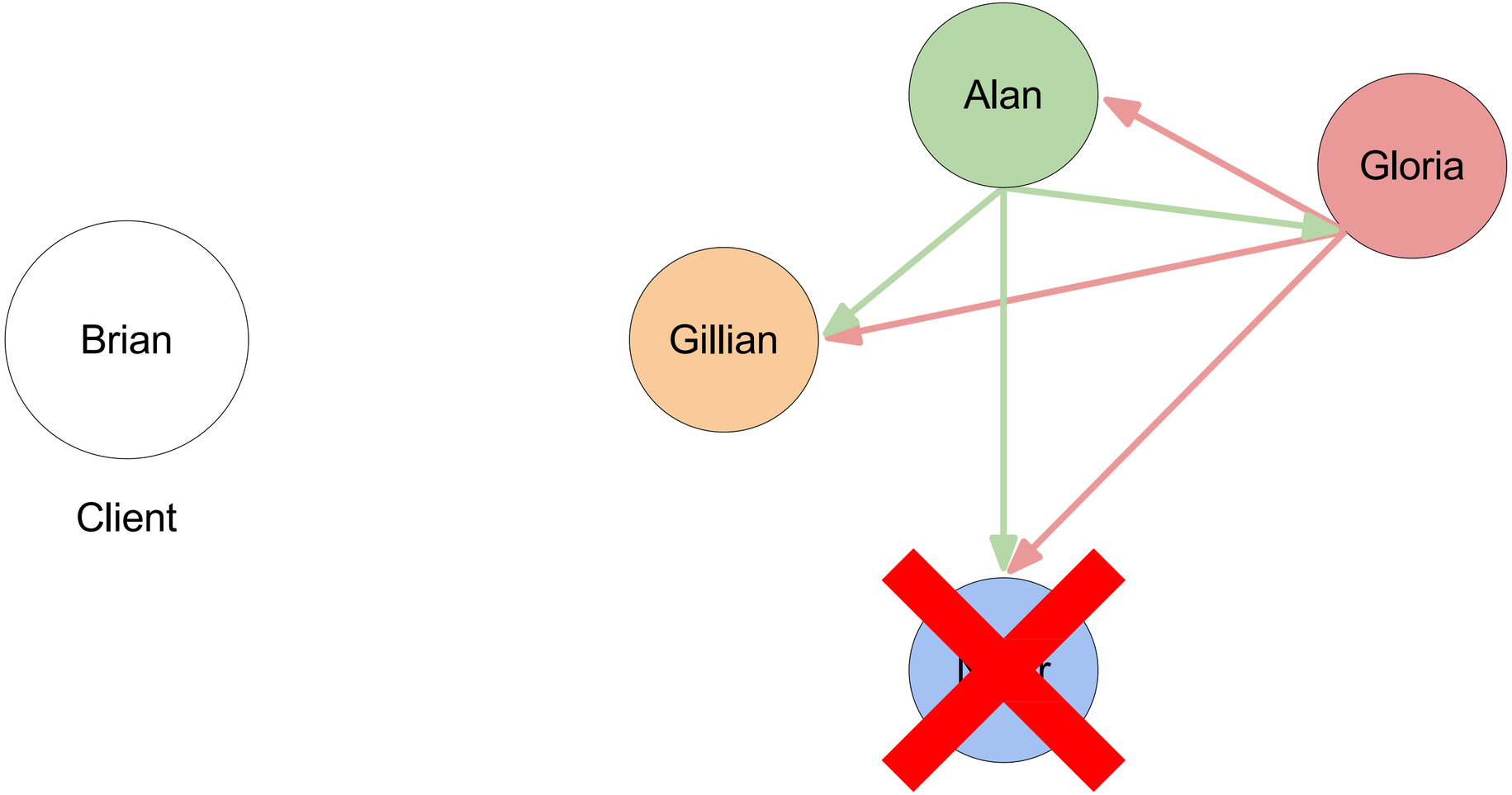
PBFT

BY EXAMPLE: PRE-PREPARE



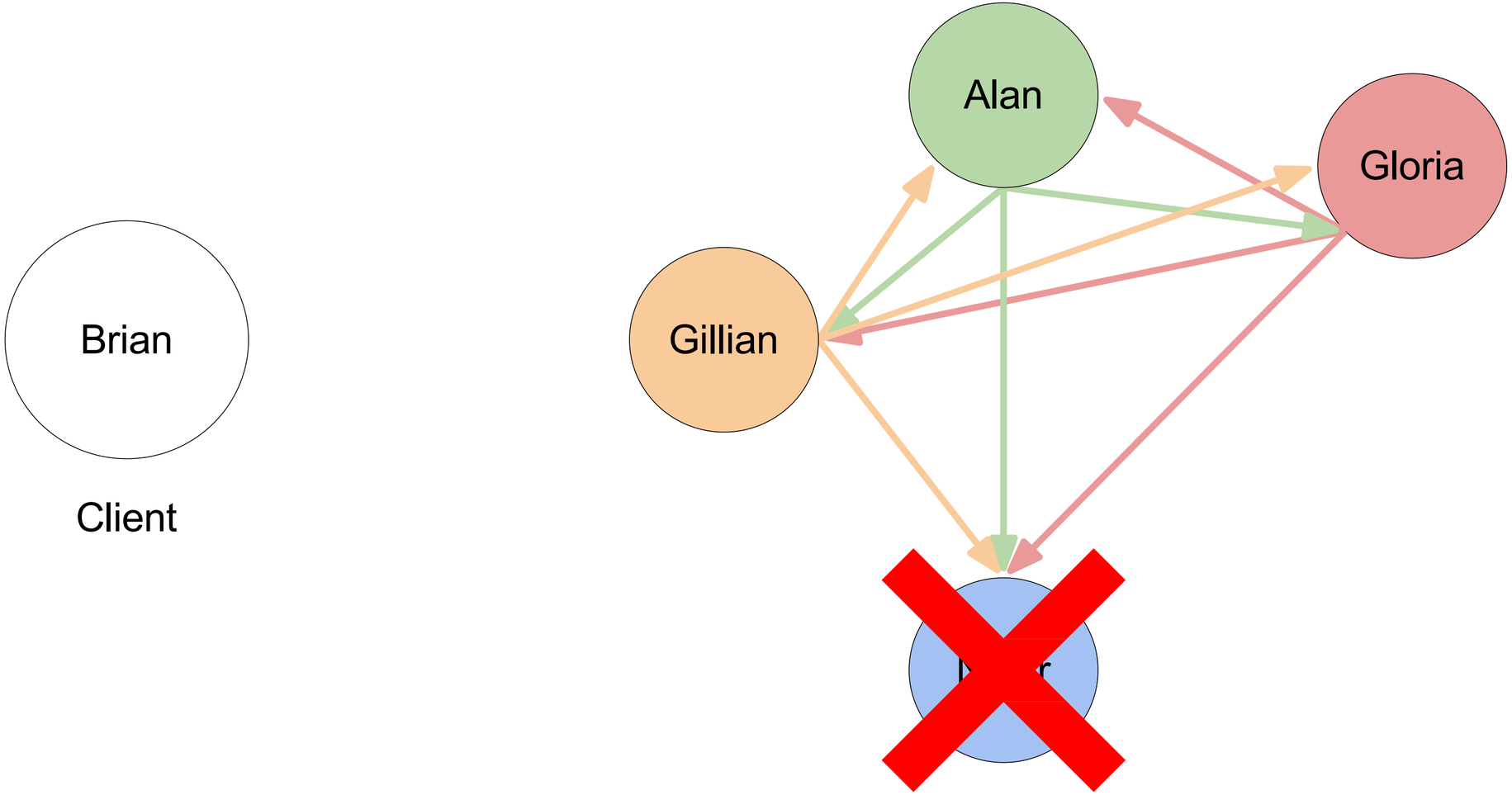
PBFT

BY EXAMPLE: PREPARE



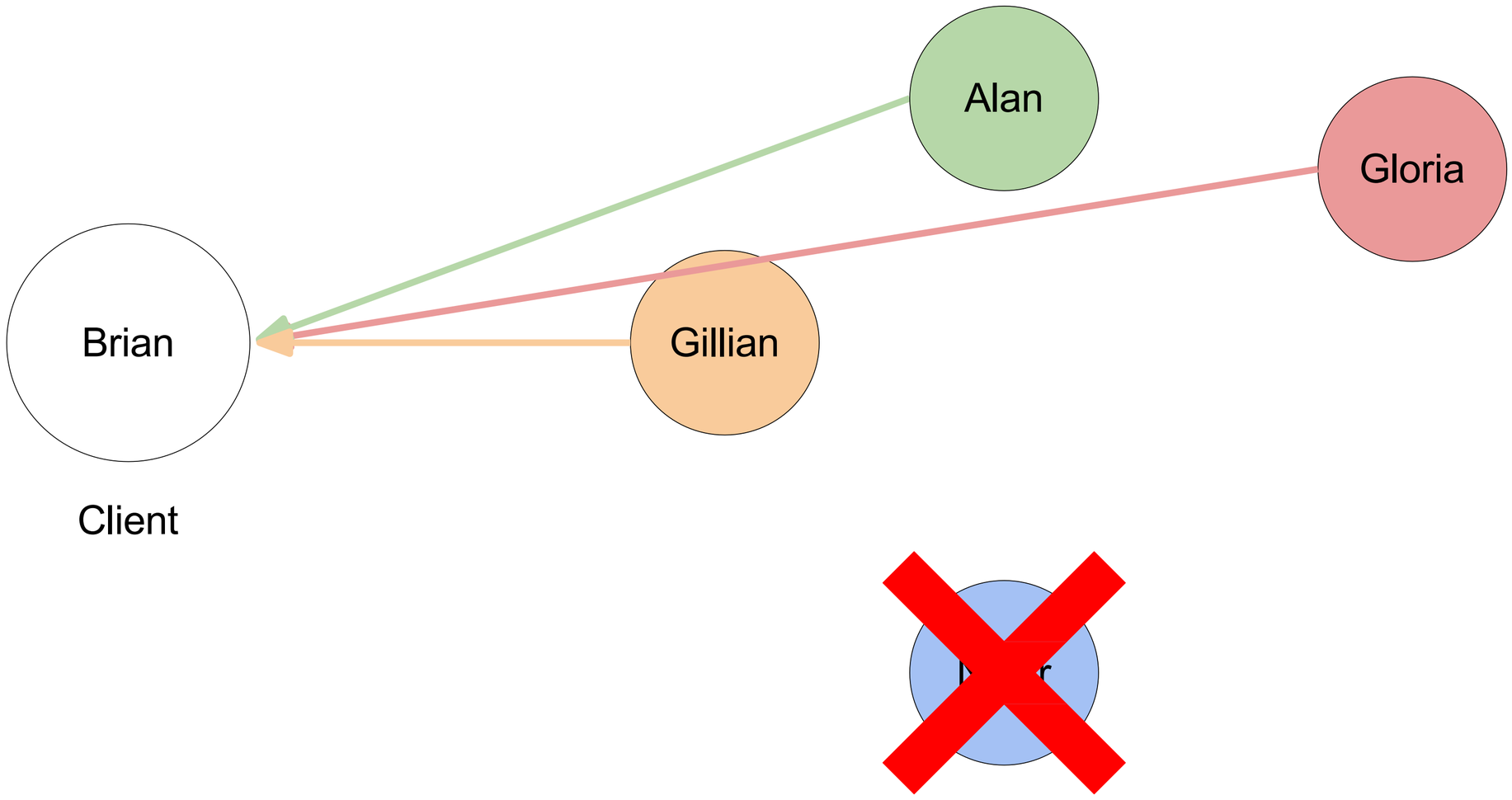
PBFT

BY EXAMPLE: COMMIT



PBFT

BY EXAMPLE: REPLY



PBFT

NORMAL CASE OPERATION

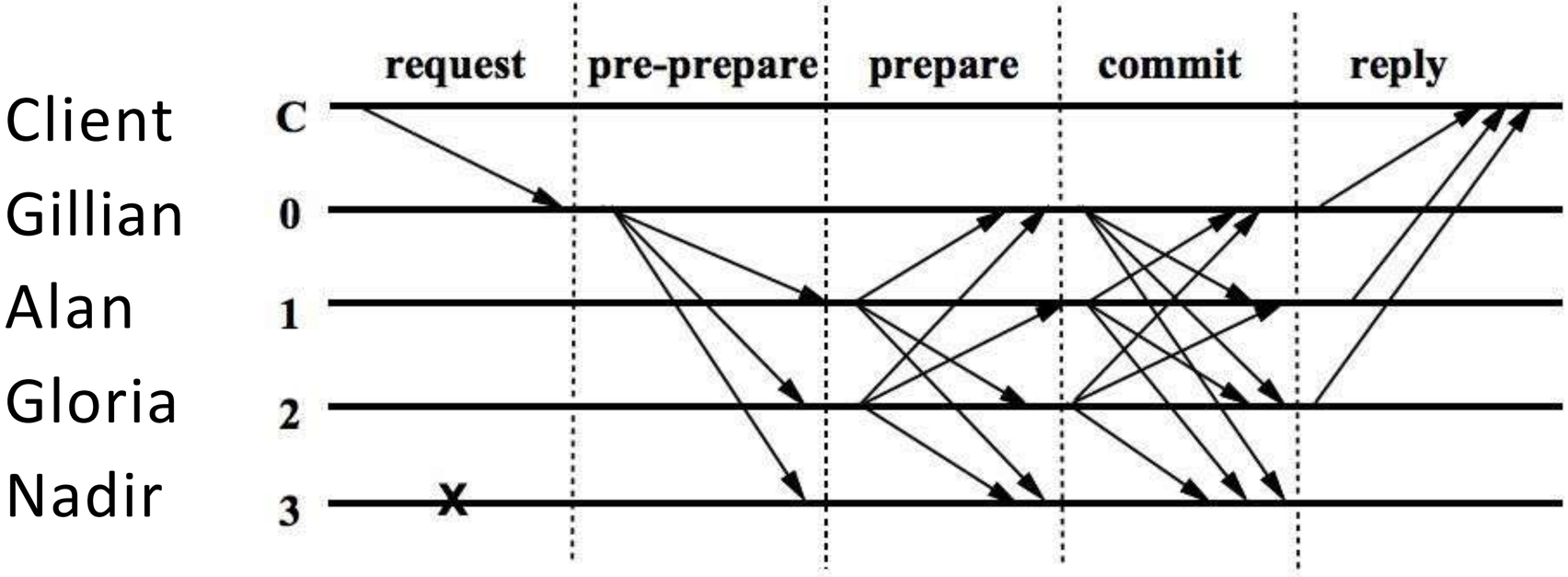


Figure 1: Normal Case Operation

source: http://pmg.csail.mit.edu/papers/osdi99.pdf

PBFT

NORMAL CASE OPERATION

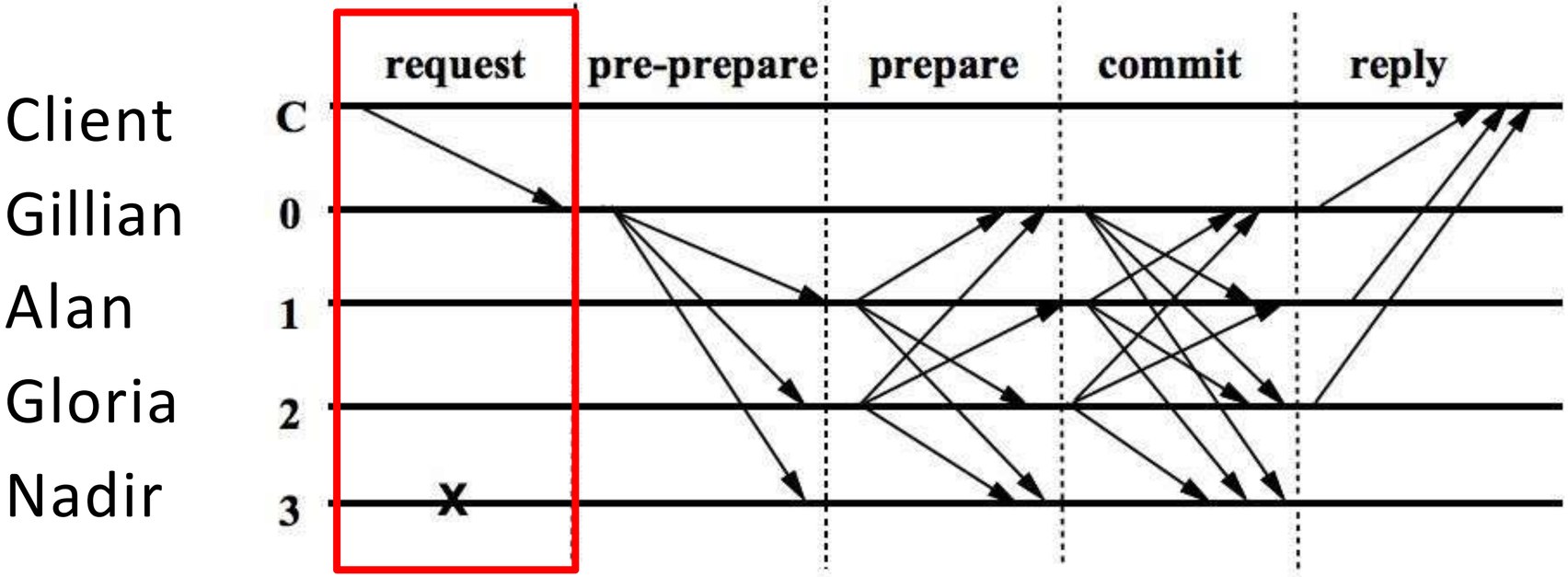


Figure 1: Normal Case Operation

source: http://pmg.csail.mit.edu/papers/osdi99.pdf

PBFT

NORMAL CASE OPERATION

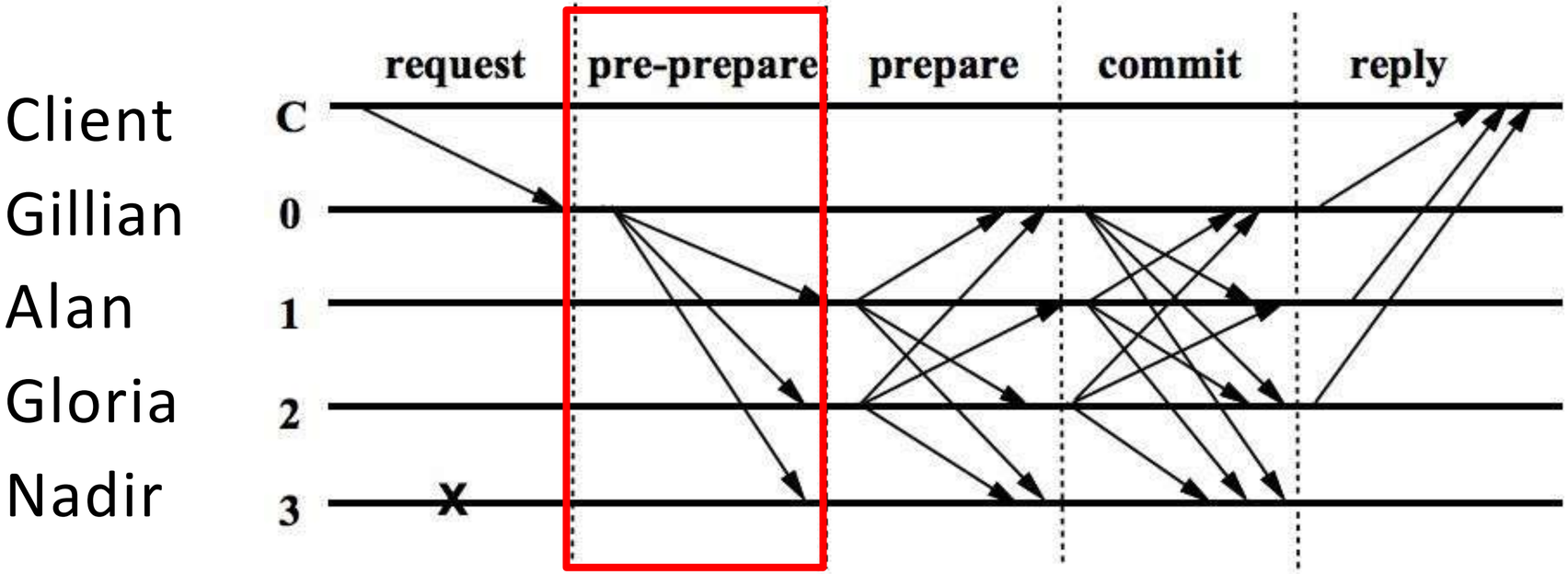


Figure 1: Normal Case Operation

PBFT

NORMAL CASE OPERATION

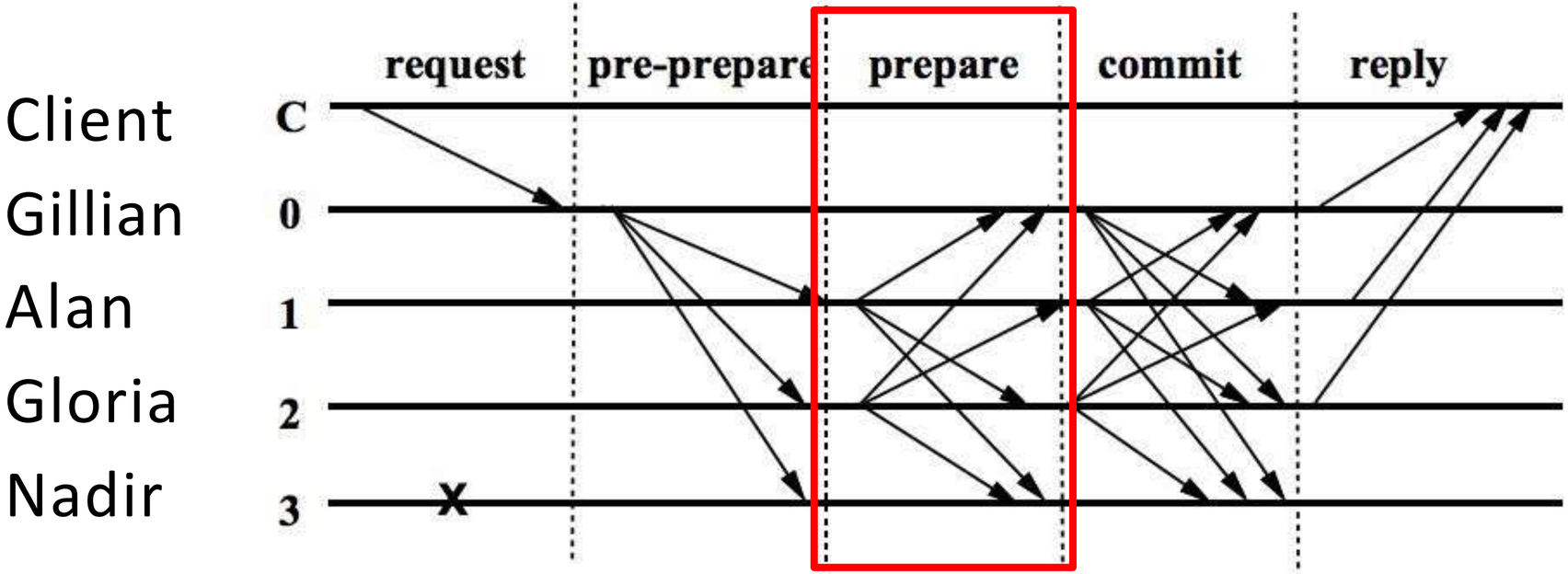


Figure 1: Normal Case Operation

source: http://pmg.csail.mit.edu/papers/osdi99.pdf

PBFT

NORMAL CASE OPERATION

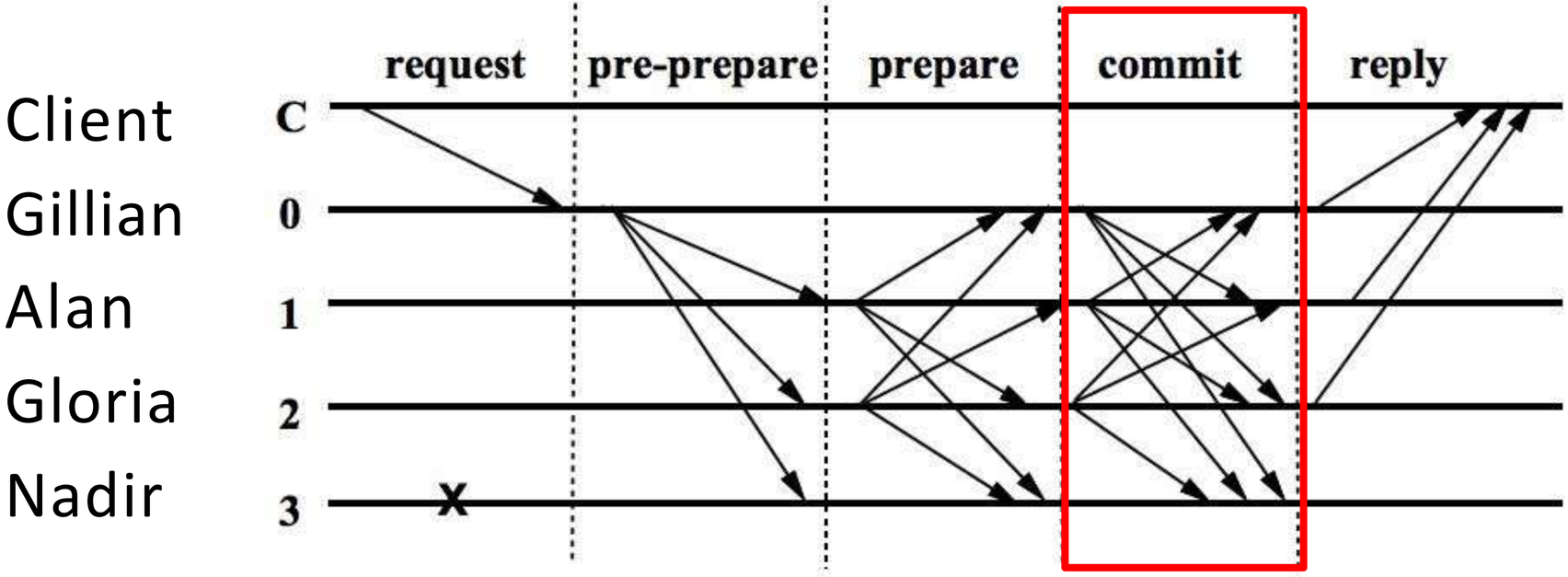


Figure 1: Normal Case Operation

source: http://pmg.csail.mit.edu/papers/osdi99.pdf

PBFT

NORMAL CASE OPERATION

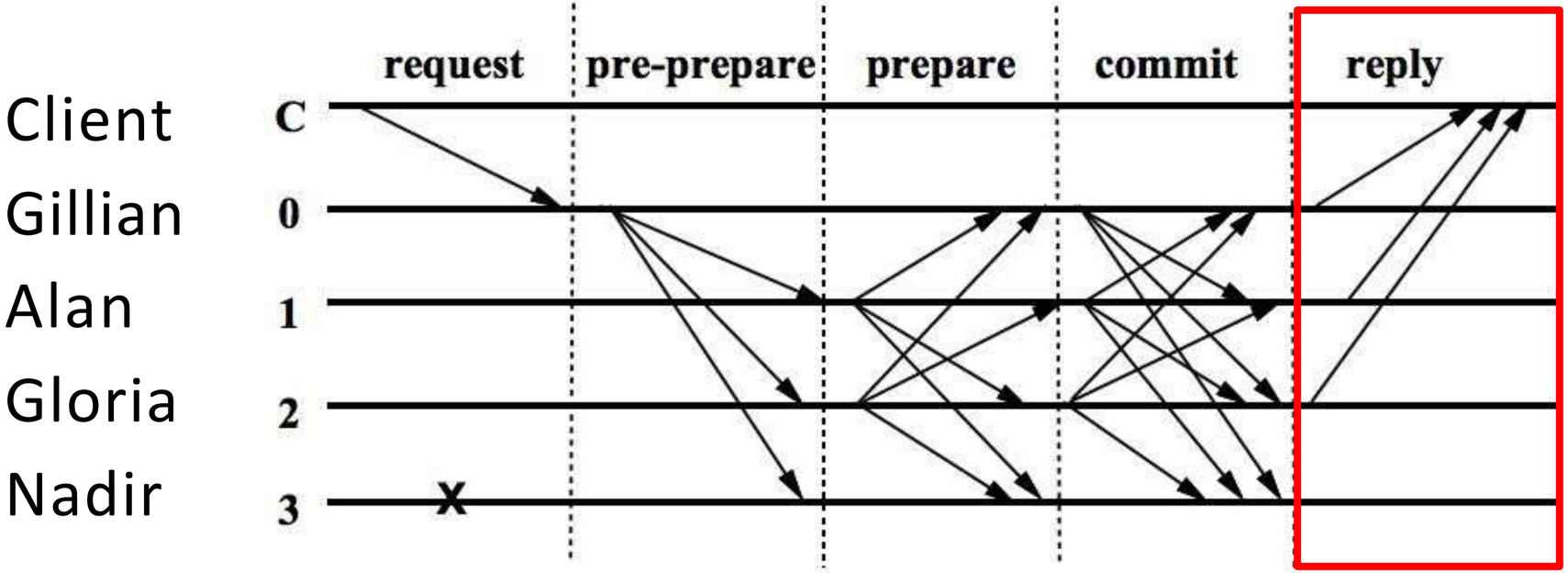


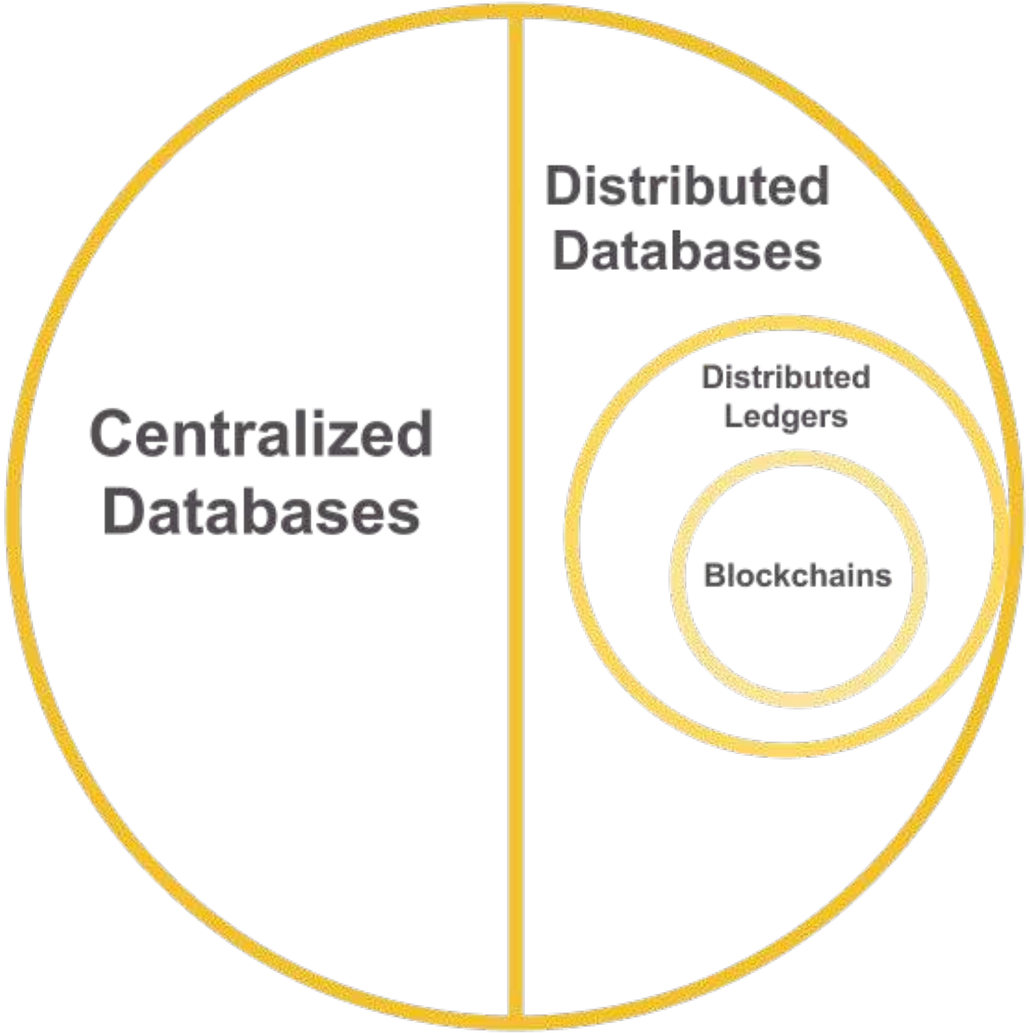
Figure 1: Normal Case Operation

source: http://pmg.csail.mit.edu/papers/osdi99.pdf

BLOCKCHAINS VS. DATABASES

HIGH LEVEL OVERVIEW

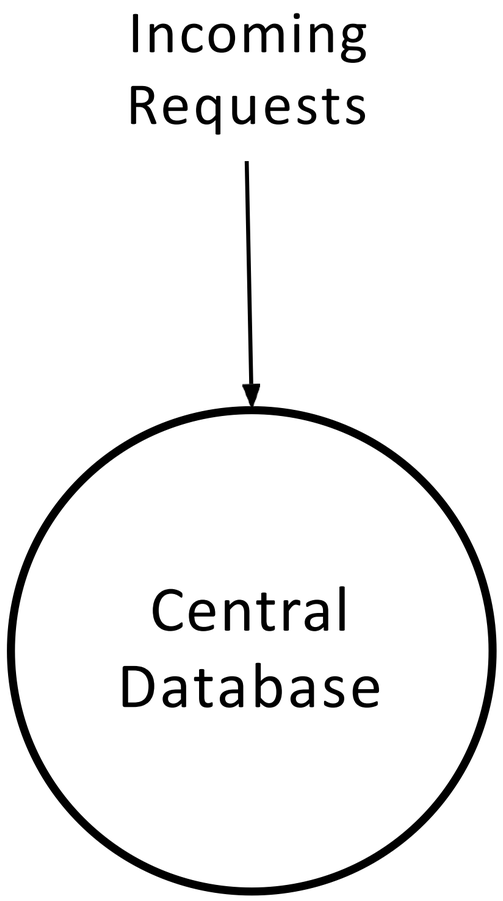
- A blockchain is a **specialized** type of distributed database
- The counterpart to a distributed database is the **centralized database**



CENTRALIZED DATABASES

BLOCKCHAINS VS. DATABASES

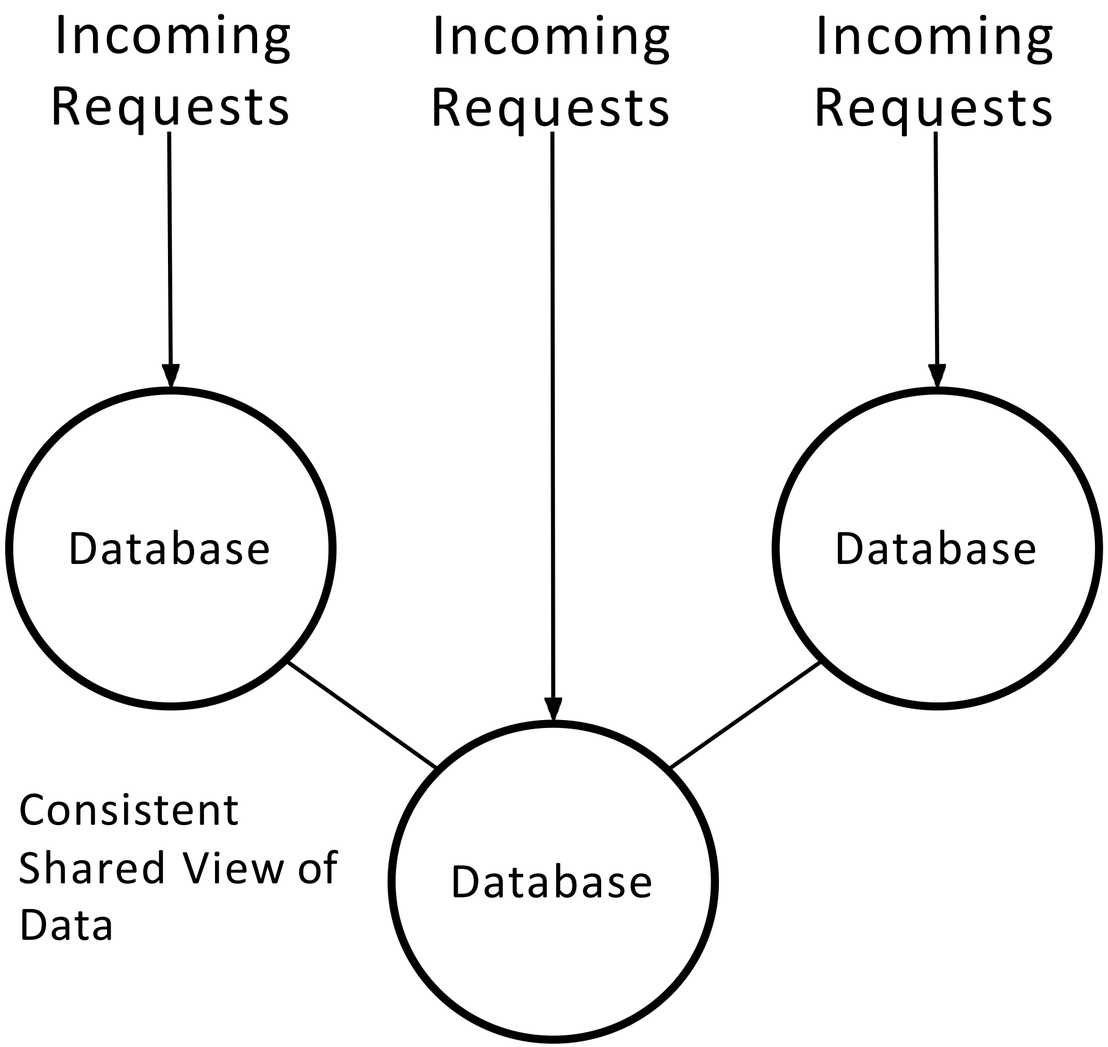
- One **central entity** handles all requests and data processing
- **Simple design** allows for immediate data updates, cost effectiveness, low redundancy
- However, also prone to bottlenecks, cannot write simultaneously from multiple users, and exists as a potential **single point of failure**



DISTRIBUTED DATABASES

BLOCKCHAINS VS. DATABASES

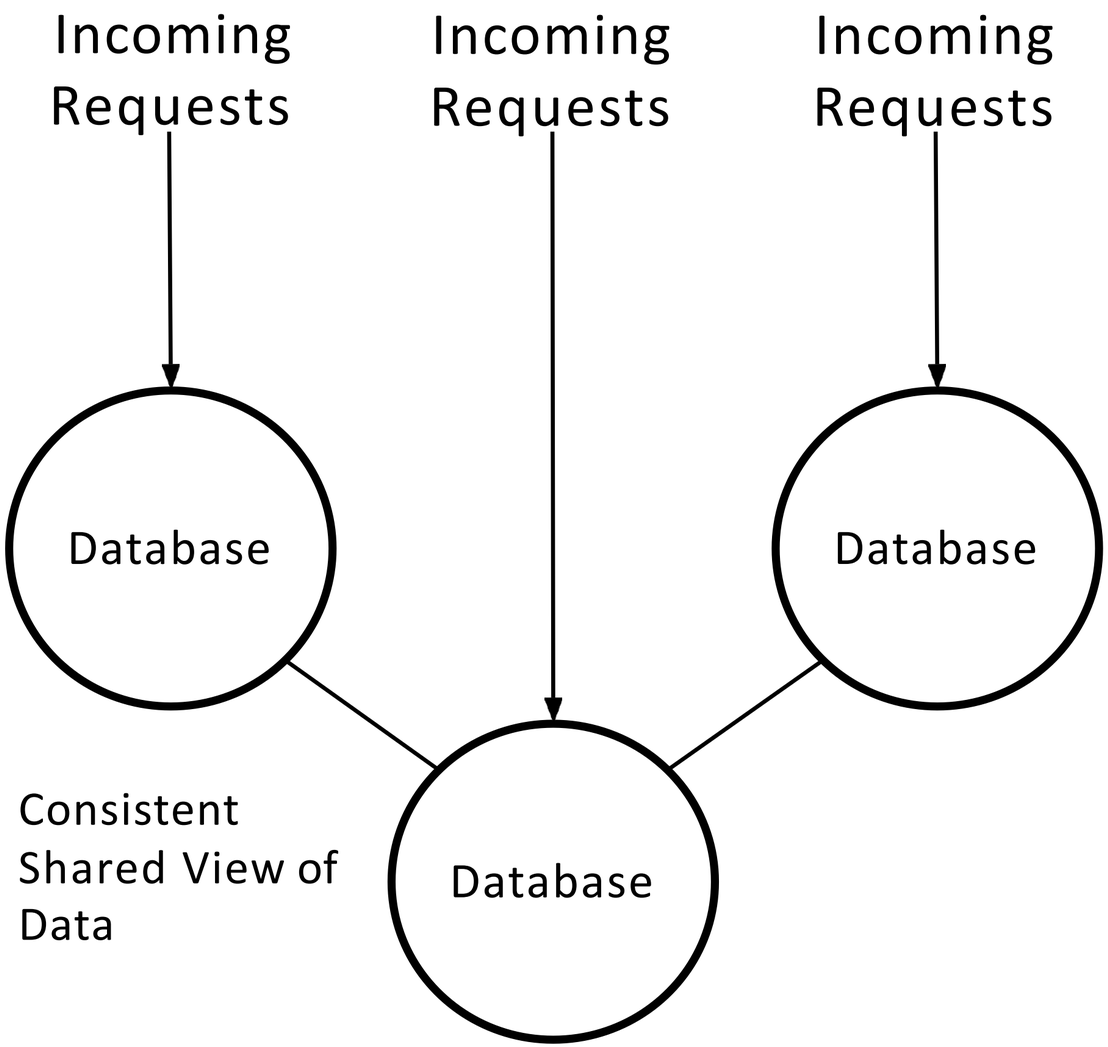
- A **group** of nodes that **trust each other** and cooperate to maintain consistent overall view of system
- Results in **more fault-tolerant system** that can handle more load
- However, also introduces **increased complexity**, costs, and potential points of failure



DISTRIBUTED LEDGERS

BLOCKCHAINS VS. DATABASES

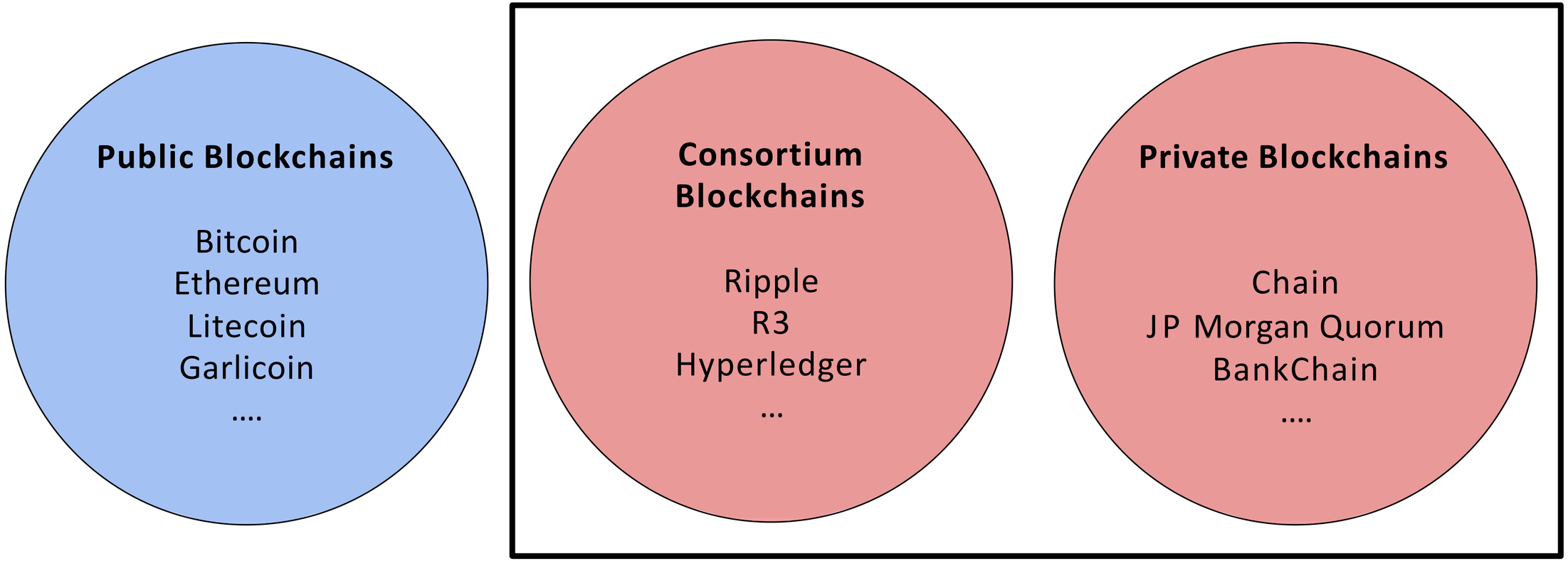
- A **group** of nodes that **do not fully trust each other**
- The nodes must implement some form of **consensus protocol** to have a consistent view of system
- Systems that use a **chain of blocks** as their protocol are known as **blockchains**



PUBLIC VS. PERMISSIONED

BLOCKCHAINS VS. DATABASES

Permissioned Ledgers



HYPERLEDGER

- Led by Linux Foundation, IBM (Hyperledger Fabric)
- **Focus**: finance, healthcare, supply chain
- Consortium of 20+ corporate members, 120+ start-up & ecosystem participants, 20+ institutions to advance blockchain technologies through open-source, collaborative development
- Produces enterprise-focused software solutions & tools for implementing blockchain applications, PoCs, solutions, etc.

HYPERLEDGER

HYPERLEDGER

Hyperledger Frameworks

HYPERLEDGER SAWTOOTH

HYPERLEDGER IROHA

HYPERLEDGER FABRIC

HYPERLEDGER BURROW

HYPERLEDGER INDY

HYPERLEDGER

Hyperledger Tools

HYPERLEDGER CALIPER

HYPERLEDGER CELLO

~~HYPERLEDGER COMPOSER~~

HYPERLEDGER EXPLORER

HYPERLEDGER QUILT

HYPERLEDGER URSA

19

HYPERLEDGER

- **Hyperledger Fabric**
  - Private network/permissioned blockchain, modular plug-and-play solutions

HYPERLEDGER FABRIC

Source: https://www.hyperledger.org/blog/2018/10/26/hyperledger-fabric-now-supports-ethereum

READINGS

- Raft Tutorial
  - http://thesecretlivesofdata.com/raft/
- Exploring Stellar Lumens — Development Tutorial
  - https://medium.com/wearetheledger/exploring-stellar-lumens-development-tutorial-78b4e1c92733

References

Slides mainly adopted from

- Blockchain @ Berkeley : https://blockchain.berkeley.edu/
- Blockchain @ Princeton : http://bitcoinbook.cs.princeton.edu/