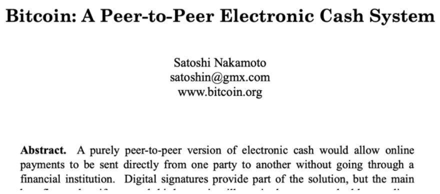
Bitcoin Blockchain

Prof. David (Duc) Tran, PhD
University of Massachusetts, Boston (USA)

UMASS
BOSTON

1

Bitcoin (Satoshi Nakamoto, 2008)



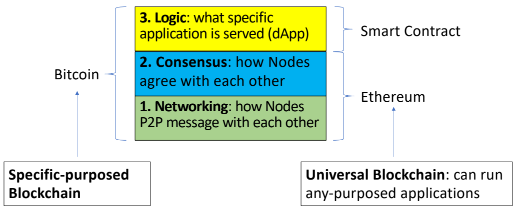
Bitcoin: A Peer-to-Peer Electronic Cash System

Satoshi Nakamoto
satoshin@gmx.com
www.bitcoin.org

Abstract. A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main

2

Blockchain "computer" architecture



3. Logic: what specific application is served (dApp)

2. Consensus: how Nodes agree with each other

1. Networking: how Nodes P2P message with each other

Smart Contract

Ethereum

Bitcoin

Specific-purposed Blockchain

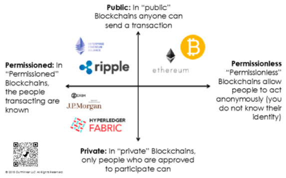
Universal Blockchain: can run any-purposed applications

3

Public vs. Private Blockchain?

- **Public blockchain**: service available to everybody
- **Private blockchain**: available to certain permitted participants

Case by case basis: can build "permissionless/permissioned" and "public/private"



4

What is Bitcoin Blockchain?

- Bitcoin is a blockchain network implementing a **digital currency**
  - Keep money in your own (electronic) wallet
  - Transfer money P2P without intermediaries (e.g., banks)
  - No need for real-life identity
  - No double spending, no fake money possible
- **Bitcoin currency:** a digital concept that represents "money" (a unit of value) in this blockchain network
- **Open-source**
  - We can clone bitcoin software to build another blockchain network, **not necessarily about money**. Remember, it is a blockchain network first

5

What is a valid transaction

A transaction = Alice sends "money" (coin) to Bob

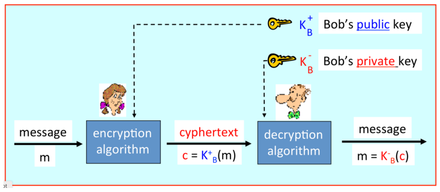
3 things to consider

- Bob is the **only recipient**, not anyone else
- Bob can verify that **Alice must undeniably be the sender**
- Alice has **enough money** to send

HOW?

6

Public-Key cryptography



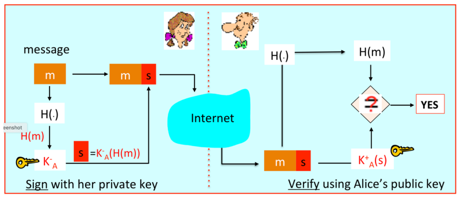
Guarantee Bob is **the only recipient** of the transaction

Bitcoin Address

- To hold Bitcoin, need a **wallet** (like a bank account)
- Address of Bob is **hash value** of his **public key**

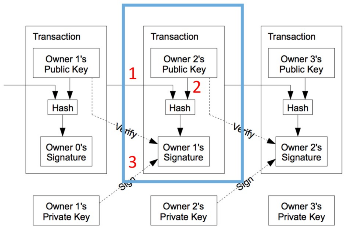
Address = HASH (public key)
Example: **1BvBMSEYstWetqTFn5Au4m4GFg7xJaNVN2**

Private Key as a Digital Signature



By signing, Alice proves that she is the sender; cannot deny that fact

Sign and Verify a Transaction



Ledger: **Account-based**

Create 25 coins and credit to Alice$_{\text{ASSERTED BY MINERS}}$
Transfer 17 coins from Alice to Bob$_{\text{SIGNED(Alice)}}$
Transfer 8 coins from Bob to Carol$_{\text{SIGNED(Bob)}}$
Transfer 5 coins from Carol to Alice$_{\text{SIGNED(Carol)}}$
Transfer 15 coins from Alice to David$_{\text{SIGNED(Alice)}}$

- **How to know if a transaction is valid?**
  - E.g., does Alice have the 15 coins to transfer to David?

Ledger: **Transaction-based**

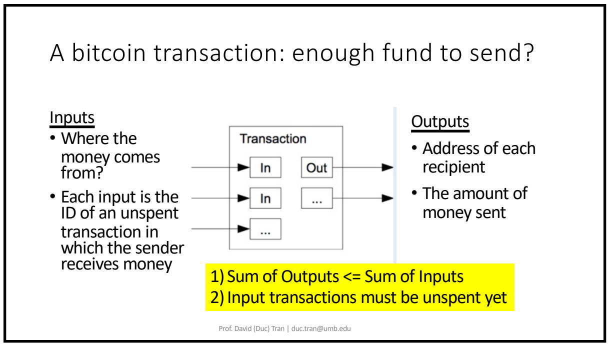
1 Inputs: Ø
Outputs: 25.0→Alice

2 Inputs: 1[0]
Outputs: 17.0→Bob, 8.0→Alice
SIGNED(Alice)

3 Inputs: 2[0]
Outputs: 8.0→Carol, 9.0→Bob
SIGNED(Bob)

4 Inputs: 2[1]
Outputs: 6.0→David, 2.0→Alice
SIGNED(Alice)

Verifying whether a transaction is valid is easy using "input" **pointers**

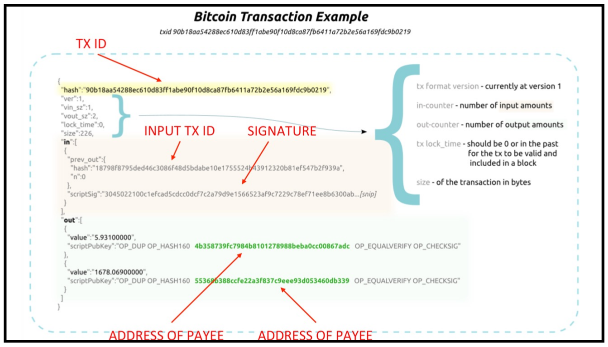


A bitcoin transaction: enough fund to send?

Inputs
- Where the money comes from?
- Each input is the ID of an unspent transaction in which the sender receives money

Outputs
- Address of each recipient
- The amount of money sent

1) Sum of Outputs <= Sum of Inputs
2) Input transactions must be unspent yet

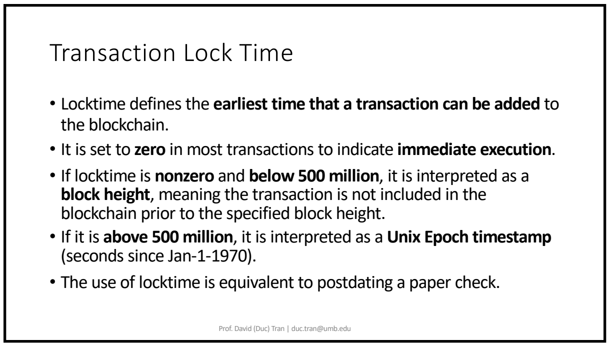


Bitcoin Transaction Example

txid 90b18aa54288ec610d83ff1abe90f10d8ca87fb6411a72b2e56a169fdc9b0219

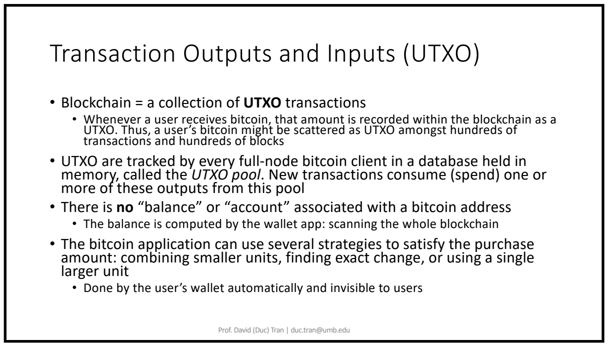
```
{
 "hash":"90b18aa54288ec610d83ff1abe90f10d8ca87fb6411a72b2e56a169fdc9b0219",
 "ver":1,
 "vin_sz":1,
 "vout_sz":2,
 "lock_time":0,
 "size":226,
 "in":[
  {
   "prev_out":{
    "hash":"18798f8795ded46c3086f48d5bdabe10e1755524b43912320b81ef547b2f939a",
    "n":0
   },
   "scriptSig":"3045022100c1efcad5cdcc0dcf7c2a79d9e1566523af9c7229c78ef71ee8b6300ab...[snip]
  }
 ],
 "out":[
  {
   "value":"5.93100000",
   "scriptPubKey":"OP_DUP OP_HASH160  4b358739fc7984b8101278988beba0cc00867adc  OP_EQUALVERIFY OP_CHECKSIG"
  },
  {
   "value":"1678.06900000",
   "scriptPubKey":"OP_DUP OP_HASH160  55368b388ccfe22a3f837c9eee93d053460db339  OP_EQUALVERIFY OP_CHECKSIG"
  }
 ]
}
```

tx format version - currently at version 1
in-counter - number of input amounts
out-counter - number of output amounts
tx lock_time - should be 0 or in the past for the tx to be valid and included in a block
size - of the transaction in bytes



Transaction Lock Time

- Locktime defines the **earliest time that a transaction can be added** to the blockchain.
- It is set to **zero** in most transactions to indicate **immediate execution**.
- If locktime is **nonzero** and **below 500 million**, it is interpreted as a **block height**, meaning the transaction is not included in the blockchain prior to the specified block height.
- If it is **above 500 million**, it is interpreted as a **Unix Epoch timestamp** (seconds since Jan-1-1970).
- The use of locktime is equivalent to postdating a paper check.



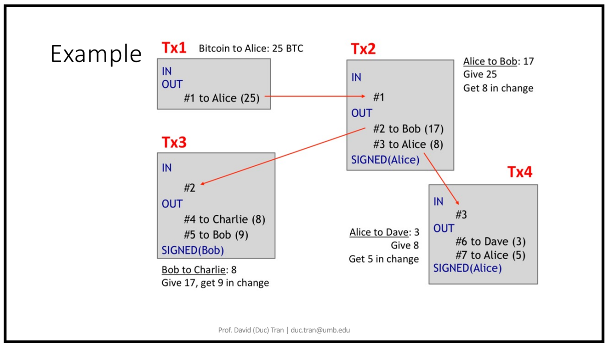
Transaction Outputs and Inputs (UTXO)

- Blockchain = a collection of **UTXO** transactions
  - Whenever a user receives bitcoin, that amount is recorded within the blockchain as a UTXO. Thus, a user's bitcoin might be scattered as UTXO amongst hundreds of transactions and hundreds of blocks
- UTXO are tracked by every full-node bitcoin client in a database held in memory, called the *UTXO pool*. New transactions consume (spend) one or more of these outputs from this pool
- There is **no** "balance" or "account" associated with a bitcoin address
  - The balance is computed by the wallet app: scanning the whole blockchain
- The bitcoin application can use several strategies to satisfy the purchase amount: combining smaller units, finding exact change, or using a single larger unit
  - Done by the user's wallet automatically and invisible to users

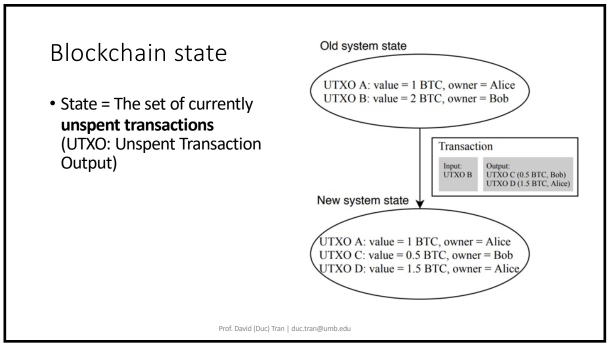
Python script to call **blockchain.info** API to find UTXO of an address

```python
# get unspent outputs from blockchain API

import json
import requests

# example address
address = '1Dorian4RoXcnBv9hnQ4Y2C1an6NJ4UrjX'

# The API URL is https://blockchain.info/unspent?active=<address>
# It returns a JSON object with a list "unspent_outputs", containing UTXO, like this:
#{       "unspent_outputs":[
#   {
#     "tx_hash":"ebadfaa92f1fd29e2fe296eda702c48bd11ffd52313e986e99ddad9084062167",
#     "tx_index":51919767,
#     "tx_output_n": 1,
#     "script":"76a9148c7e252f8d64b0b6e313985915110fcfefcf4a2d88ac",
#     "value": 8000000,
#     "value_hex": "7a1200",
#     "confirmations":28691
#   },
# ...
#]}

resp = requests.get('https://blockchain.info/unspent?active=%s' % address)
utxo_set = json.loads(resp.text)["unspent_outputs"]

for utxo in utxo_set:
    print "%s:%d - %ld Satoshis" % (utxo['tx_hash'], utxo['tx_output_n'], utxo['value'])
```

```
$ python get-utxo.py
ebadfaa92f1fd29e2fe296eda702c48bd11ffd52313e986e99ddad9084062167:1 - 8000000 Satoshis
6596fd070679de96e405d52b51b8e1d644029108ec4cbfe451454486796a1ecf:0 - 16050000 Satoshis
74d788804e2aae10891d72753d1520da1206e6f4f20481cc1555b7f2cb44aca0:0 - 5000000 Satoshis
b2affea89ff82557c60d635a2a3137b8f88f12ecec85082f7d0a1f82ee203ac4:0 - 10000000 Satoshis
...
```

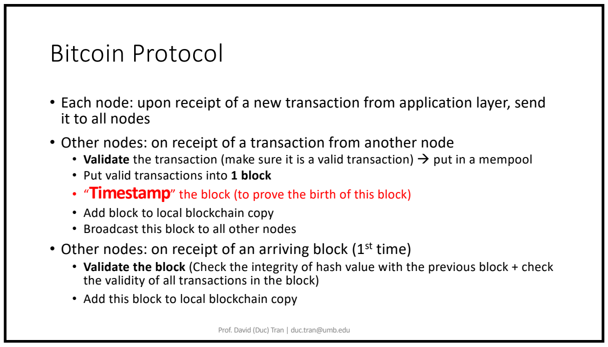


Example



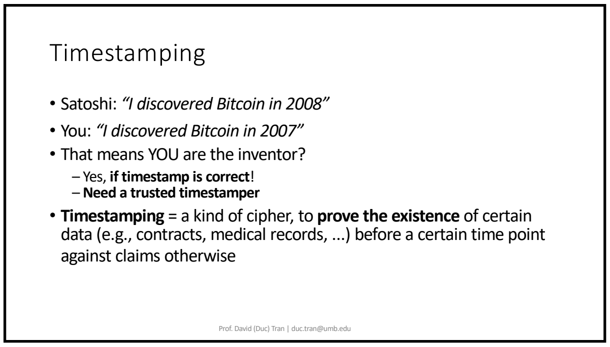
Blockchain state

- State = The set of currently **unspent transactions** (UTXO: Unspent Transaction Output)



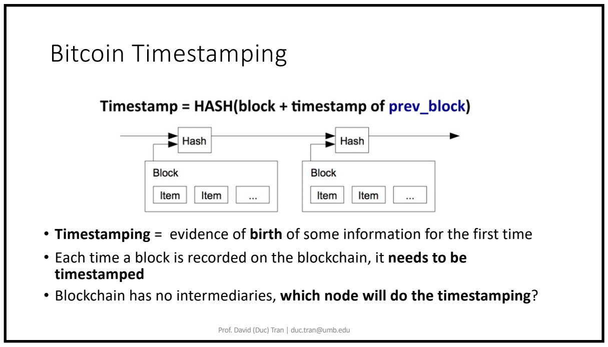
Bitcoin Protocol

- Each node: upon receipt of a new transaction from application layer, send it to all nodes
- Other nodes: on receipt of a transaction from another node
  - **Validate** the transaction (make sure it is a valid transaction) → put in a mempool
  - Put valid transactions into **1 block**
  - "**Timestamp**" the block (to prove the birth of this block)
  - Add block to local blockchain copy
  - Broadcast this block to all other nodes
- Other nodes: on receipt of an arriving block (1st time)
  - **Validate the block** (Check the integrity of hash value with the previous block + check the validity of all transactions in the block)
  - Add this block to local blockchain copy



Timestamping

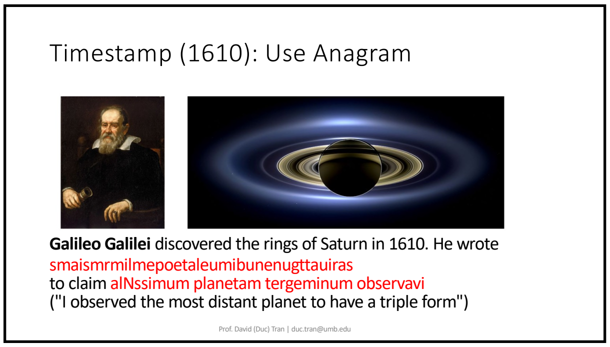
- Satoshi: *"I discovered Bitcoin in 2008"*
- You: *"I discovered Bitcoin in 2007"*
- That means YOU are the inventor?
  - Yes, **if timestamp is correct**!
  - **Need a trusted timestamper**
- **Timestamping** = a kind of cipher, to **prove the existence** of certain data (e.g., contracts, medical records, …) before a certain time point against claims otherwise



Bitcoin Timestamping

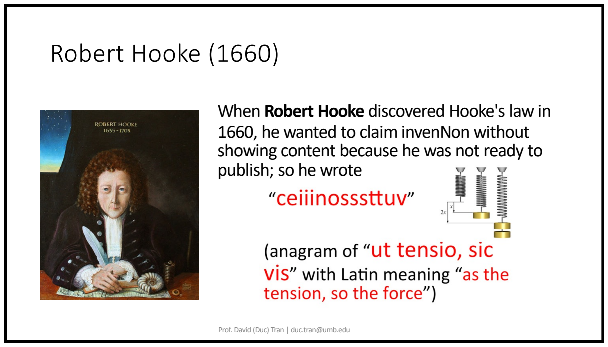
Timestamp = HASH(block + timestamp of prev_block)

- **Timestamping** = evidence of **birth** of some information for the first time
- Each time a block is recorded on the blockchain, it **needs to be timestamped**
- Blockchain has no intermediaries, **which node will do the timestamping**?



Timestamp (1610): Use Anagram

Galileo Galilei discovered the rings of Saturn in 1610. He wrote
smaismrmilmepoetaleumibunenugttauiras
to claim alNssimum planetam tergeminum observavi
("I observed the most distant planet to have a triple form")



Robert Hooke (1660)

When **Robert Hooke** discovered Hooke's law in 1660, he wanted to claim invenNon without showing content because he was not ready to publish; so he wrote

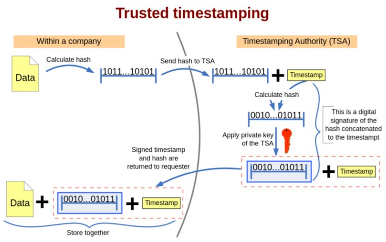
"ceiiinosssttuv"

(anagram of "ut tensio, sic vis" with Latin meaning "as the tension, so the force")

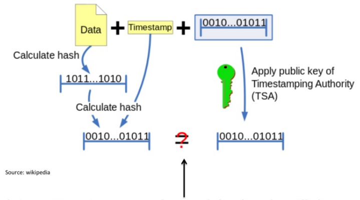
Anagram: Các ví dụ vui

- "resoul" = "fluster"
- "funeral" = "real fun"
- "adultery" = "true lady"
- "customers" = "store scum"
- "forty five" = "over fifty”
- "William Shakespeare" = "I am a weakish speller”

Modern Timestamping



Checking Timestamp



Source: wikipedia

If data or timestamp are changed, hashcode will disagree

Bitcoin Timestamping

- Anyone can timestamp → called a **Miner** (**mining** bitcoin)
- Only need a computer running the **Bitcoin Client software**
- **Incentivized to participate → receive 6.25 BTC** (currently) for each good block created

Challenge:

- Everybody wants to create the next block, **choose whom**? (**Consensus Problem**)
- How to **discourage bad nodes**?

Solution: Proof-of-Work

“Proof of Work” (PoW)

Miner Node

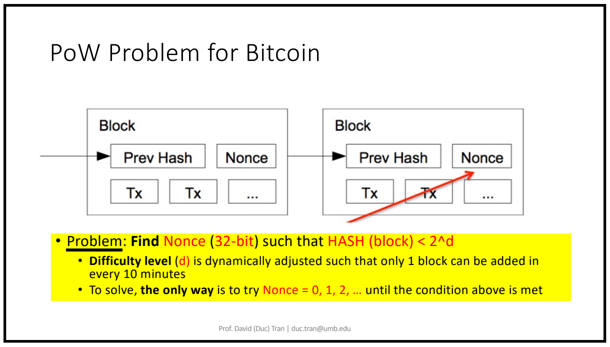
- Solve 1 **difficult** “computational puzzle”
- If solving first → successful block → timestamp done!

Why PoW?

- **It takes time to solve** → discourage bad nodes from abusing
- **Slow to create a block** → scarcity for bitcoin → stable/increased price
- Stronger computer → faster timestamping → **good competition**

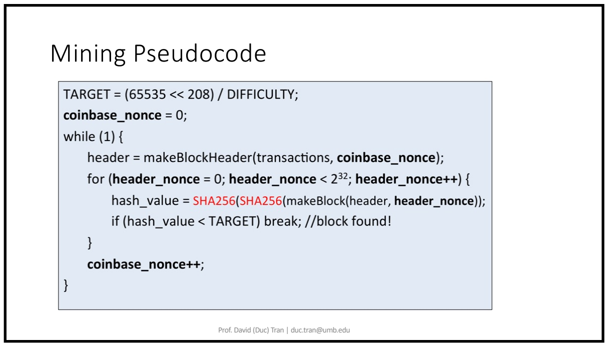
POW is not new

- Proposed by Dwork & Naor to prevent email spamming (1992)
  - Every time you send an email, your computer must solve a computational puzzle
  - The recipient’s email program ignores your email if you don’t attach the solution to the puzzle.
- A similar idea was proposed in **HashCash** by Adam Back (1997) for anti-denial-of-service
  - Bitcoin extends the PoW idea of HashCash



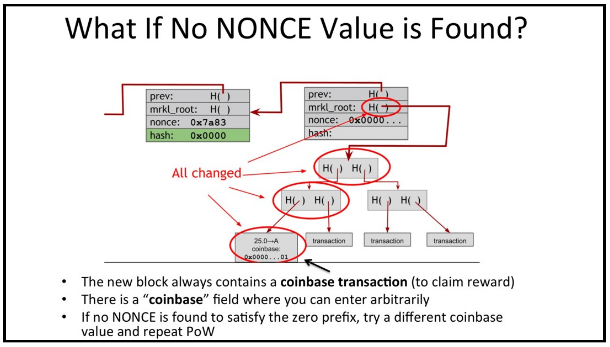
PoW Problem for Bitcoin

- <u>Problem</u>: **Find** Nonce (32-bit) such that HASH (block) < 2^d
  - **Difficulty level** (d) is dynamically adjusted such that only 1 block can be added in every 10 minutes
  - To solve, **the only way** is to try Nonce = 0, 1, 2, … until the condition above is met



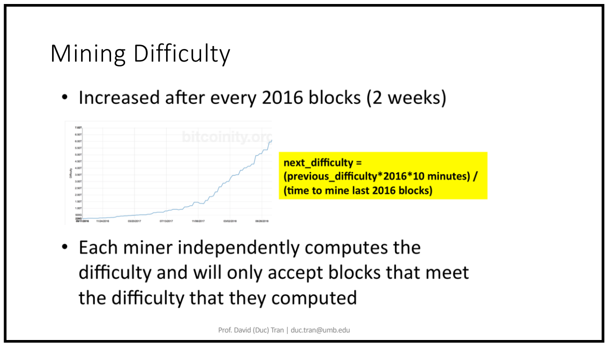
Mining Pseudocode

```
TARGET = (65535 << 208) / DIFFICULTY;
coinbase_nonce = 0;
while (1) {
    header = makeBlockHeader(transactions, coinbase_nonce);
    for (header_nonce = 0; header_nonce < $2^{32}$; header_nonce++) {
        hash_value = SHA256(SHA256(makeBlock(header, header_nonce));
        if (hash_value < TARGET) break; //block found!
    }
    coinbase_nonce++;
}
```



What If No NONCE Value is Found?

- The new block always contains a **coinbase transaction** (to claim reward)
- There is a "**coinbase**" field where you can enter arbitrarily
- If no NONCE is found to satisfy the zero prefix, try a different coinbase value and repeat PoW

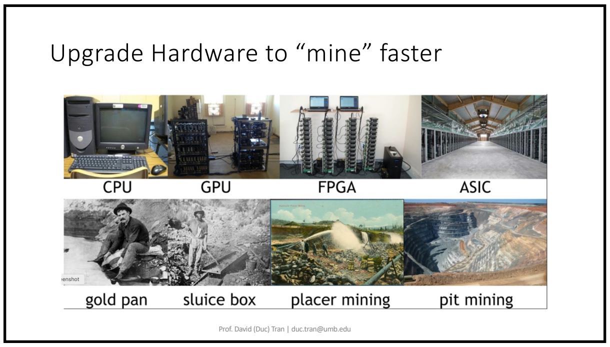


Mining Difficulty

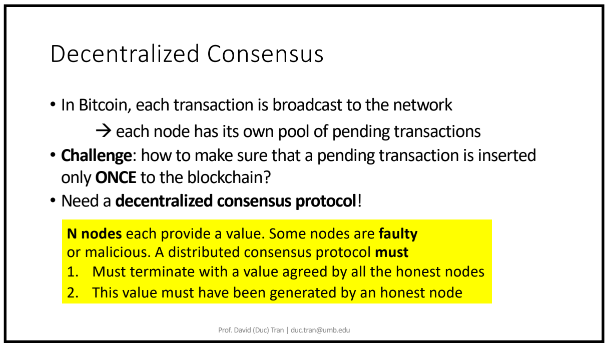
- Increased after every 2016 blocks (2 weeks)

next_difficulty = (previous_difficulty*2016*10 minutes) / (time to mine last 2016 blocks)

- Each miner independently computes the difficulty and will only accept blocks that meet the difficulty that they computed



Upgrade Hardware to "mine" faster

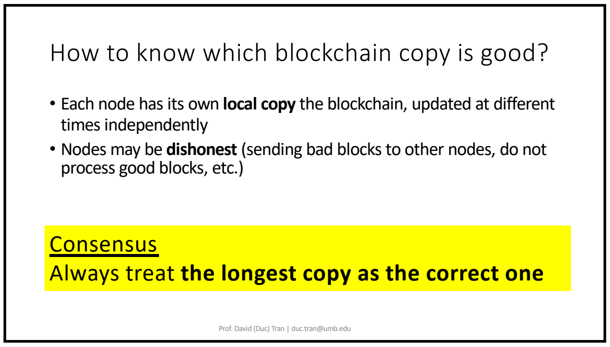


Decentralized Consensus

- In Bitcoin, each transaction is broadcast to the network
  → each node has its own pool of pending transactions
- **Challenge**: how to make sure that a pending transaction is inserted only **ONCE** to the blockchain?
- Need a **decentralized consensus protocol**!

N nodes each provide a value. Some nodes are **faulty** or malicious. A distributed consensus protocol **must**

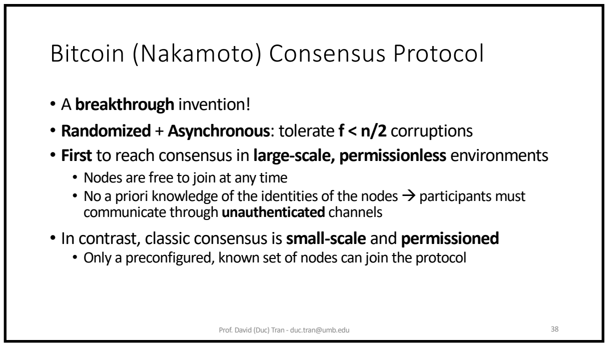
1. Must terminate with a value agreed by all the honest nodes
2. This value must have been generated by an honest node



How to know which blockchain copy is good?

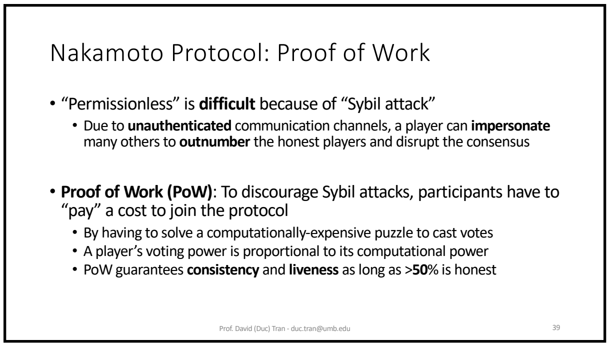
- Each node has its own **local copy** the blockchain, updated at different times independently
- Nodes may be **dishonest** (sending bad blocks to other nodes, do not process good blocks, etc.)

Consensus
Always treat **the longest copy as the correct one**



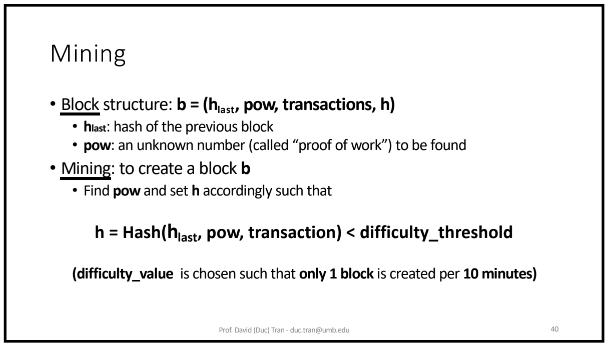
Bitcoin (Nakamoto) Consensus Protocol

- A **breakthrough** invention!
- **Randomized** + **Asynchronous**: tolerate **f < n/2** corruptions
- **First** to reach consensus in **large-scale, permissionless** environments
  - Nodes are free to join at any time
  - No a priori knowledge of the identities of the nodes → participants must communicate through **unauthenticated** channels
- In contrast, classic consensus is **small-scale** and **permissioned**
  - Only a preconfigured, known set of nodes can join the protocol



Nakamoto Protocol: Proof of Work

- "Permissionless" is **difficult** because of "Sybil attack"
  - Due to **unauthenticated** communication channels, a player can **impersonate** many others to **outnumber** the honest players and disrupt the consensus
- **Proof of Work (PoW)**: To discourage Sybil attacks, participants have to "pay" a cost to join the protocol
  - By having to solve a computationally-expensive puzzle to cast votes
  - A player's voting power is proportional to its computational power
  - PoW guarantees **consistency** and **liveness** as long as >**50%** is honest

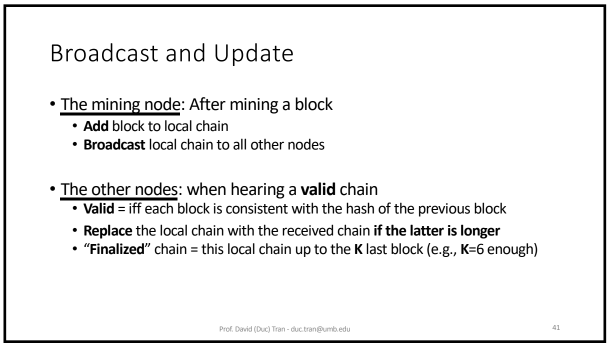


Mining

- Block structure: **b = (h_{last}, pow, transactions, h)**
  - **h_{last}**: hash of the previous block
  - **pow**: an unknown number (called "proof of work") to be found
- Mining: to create a block **b**
  - Find **pow** and set **h** accordingly such that

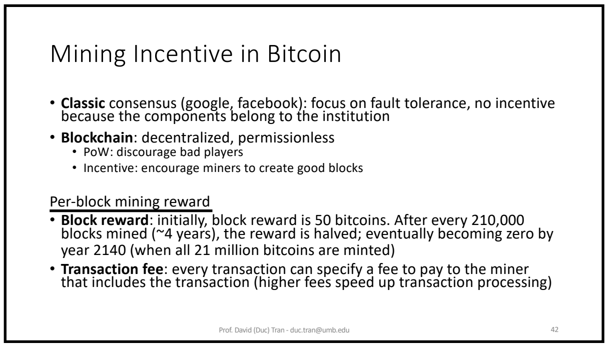
$$\mathbf{h = Hash(h_{last}, pow, transaction) < difficulty\_threshold}$$

(difficulty_value is chosen such that **only 1 block** is created per **10 minutes)**



Broadcast and Update

- The mining node: After mining a block
  - **Add** block to local chain
  - **Broadcast** local chain to all other nodes
- The other nodes: when hearing a **valid** chain
  - **Valid** = iff each block is consistent with the hash of the previous block
  - **Replace** the local chain with the received chain **if the latter is longer**
  - "**Finalized**" chain = this local chain up to the **K** last block (e.g., **K**=6 enough)

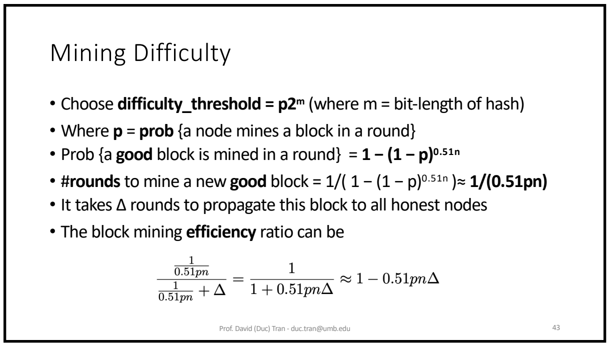


Mining Incentive in Bitcoin

- **Classic** consensus (google, facebook): focus on fault tolerance, no incentive because the components belong to the institution
- **Blockchain**: decentralized, permissionless
  - PoW: discourage bad players
  - Incentive: encourage miners to create good blocks

Per-block mining reward

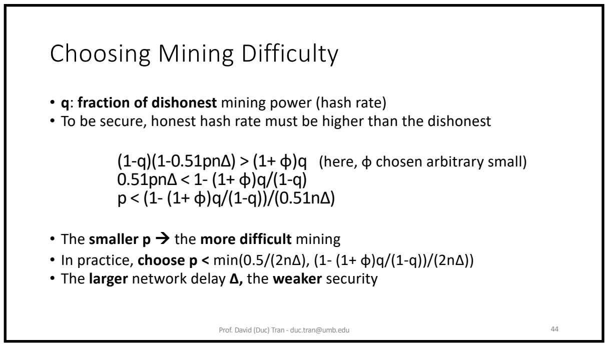
- **Block reward**: initially, block reward is 50 bitcoins. After every 210,000 blocks mined (~4 years), the reward is halved; eventually becoming zero by year 2140 (when all 21 million bitcoins are minted)
- **Transaction fee**: every transaction can specify a fee to pay to the miner that includes the transaction (higher fees speed up transaction processing)



Mining Difficulty

- Choose **difficulty_threshold = $p2^m$** (where m = bit-length of hash)
- Where **p** = **prob** {a node mines a block in a round}
- Prob {a **good** block is mined in a round} = **$1 - (1 - p)^{0.51n}$**
- **#rounds** to mine a new **good** block = $1/(1 - (1 - p)^{0.51n}) \approx$ **1/(0.51pn)**
- It takes Δ rounds to propagate this block to all honest nodes
- The block mining **efficiency** ratio can be

$$\frac{\frac{1}{0.51pn}}{\frac{1}{0.51pn} + \Delta} = \frac{1}{1 + 0.51pn\Delta} \approx 1 - 0.51pn\Delta$$

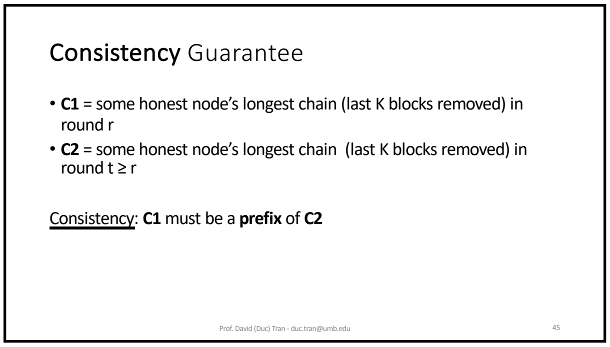


Choosing Mining Difficulty

- **q**: **fraction of dishonest** mining power (hash rate)
- To be secure, honest hash rate must be higher than the dishonest

$(1-q)(1-0.51pn\Delta) > (1+\phi)q$ (here, ϕ chosen arbitrary small)
$0.51pn\Delta < 1 - (1+\phi)q/(1-q)$
$p < (1 - (1+\phi)q/(1-q))/(0.51n\Delta)$

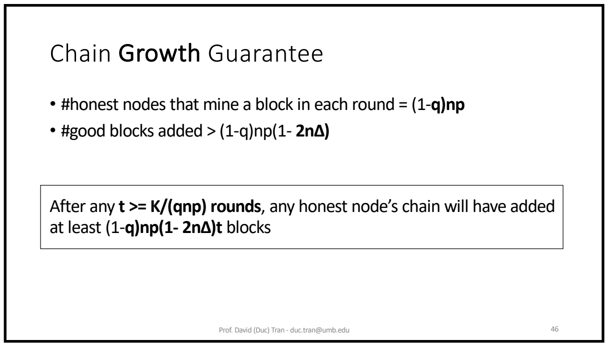
- The **smaller p** → the **more difficult** mining
- In practice, **choose p <** min(0.5/(2nΔ), (1- (1+ ϕ)q/(1-q))/(2nΔ))
- The **larger** network delay **Δ,** the **weaker** security



Consistency Guarantee

- **C1** = some honest node's longest chain (last K blocks removed) in round r
- **C2** = some honest node's longest chain (last K blocks removed) in round t ≥ r

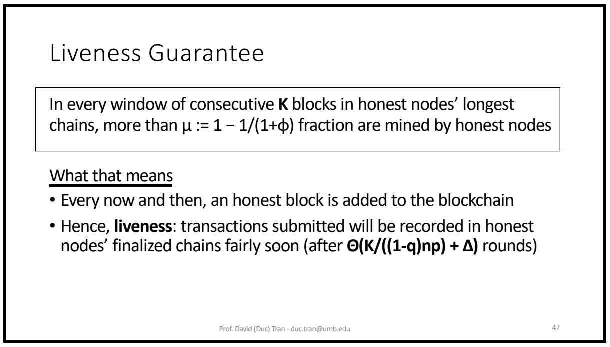
Consistency: **C1** must be a **prefix** of **C2**



Chain Growth Guarantee

- #honest nodes that mine a block in each round = (1-**q)np**
- #good blocks added > (1-q)np(1- **2nΔ)**

After any **t >= K/(qnp) rounds**, any honest node's chain will have added at least (1-**q)np(1- 2nΔ)t** blocks

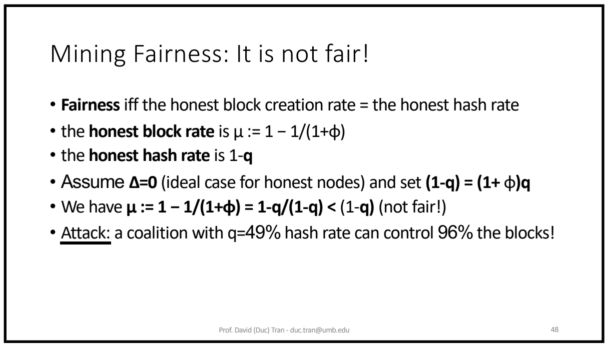


Liveness Guarantee

In every window of consecutive **K** blocks in honest nodes' longest chains, more than μ := 1 – 1/(1+ϕ) fraction are mined by honest nodes

What that means

- Every now and then, an honest block is added to the blockchain
- Hence, **liveness**: transactions submitted will be recorded in honest nodes' finalized chains fairly soon (after **Θ(K/((1-q)np) + Δ)** rounds)



Mining Fairness: It is not fair!

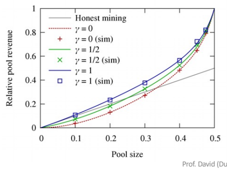
- **Fairness** iff the honest block creation rate = the honest hash rate
- the **honest block rate** is μ := 1 – 1/(1+ϕ)
- the **honest hash rate** is 1-**q**
- Assume **Δ=0** (ideal case for honest nodes) and set **(1-q) = (1+ ϕ)q**
- We have **μ := 1 – 1/(1+ϕ) = 1-q/(1-q) <** (1-**q**) (not fair!)
- Attack: a coalition with q=49% hash rate can control 96% the blocks!

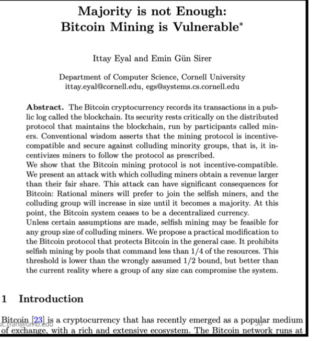
Selfish Mining Attack

- Capitalize on the unfairness of Nakamoto consensus: damage transactions, earn block mining rewards
- Selfish miner: mine a block B, but **withhold** it until some honest miner also mines a block B' at the same length (block number) as B
  - When this happens, immediately releases B
  - If B is **transmitted faster** than B', honest nodes' work (B') is **wasted**
  - **Feasible:** bad nodes collude with the network relay to deliver blocks to miners

Selfish Mining Attack

- A coalition of selfish miners with 49% mining power can control 96% the blocks!





Majority is not Enough:
Bitcoin Mining is Vulnerable*

Ittay Eyal and Emin Gün Sirer

Department of Computer Science, Cornell University
ittay.eyal@cornell.edu, egs@systems.cs.cornell.edu

Abstract. The Bitcoin cryptocurrency records its transactions in a public log called the blockchain. Its security rests critically on the distributed protocol that maintains the blockchain, run by participants called miners. Conventional wisdom asserts that the mining protocol is incentive-compatible and secure against colluding minority groups, that is, it incentivizes miners to follow the protocol as prescribed.
We show that the Bitcoin mining protocol is not incentive-compatible. We present an attack with which colluding miners obtain a revenue larger than their fair share. This attack can have significant consequences for Bitcoin: Rational miners will prefer to join the selfish miners, and the colluding group will increase in size until it becomes a majority. At this point, the Bitcoin system ceases to be a decentralized currency.
Unless certain assumptions are made, selfish mining may be feasible for any group size of colluding miners. We propose a practical modification to the Bitcoin protocol that protects Bitcoin in the general case. It prohibits selfish mining by pools that command less than 1/4 of the resources. This threshold is lower than the wrongly assumed 1/2 bound, but better than the current reality where a group of any size can compromise the system.

1 Introduction

Bitcoin [23] is a cryptocurrency that has recently emerged as a popular medium of exchange, with a rich and extensive ecosystem. The Bitcoin network runs at

The Double Spending Problem

- A malicious miner makes a payment, then secretively creates a second conflicting transaction in a new block, which allows him to recover the funds
- Feasible if he controls **q>50%** hashrate → mining **faster** than the rest of the network → make his local chain **the longest**
- But due to randomness, if **q<50%,** there is **still a non-zero chance**
- **How to minimize risk?** When somebody pays you, **wait some time** before delivering service. In bitcoin, wait for **6 block confirmations**

Why 6 Block Confirmations?

- Consider a miner with a fraction $0 < p \leq 1$ of the total hash rate
- The whole network takes on average τ_0 = 10 minutes to create a block
- The miner takes on average $t_0 = \tau p$ time to create a block
- T = T1, T2, . . . , Tn: inter-block mining time of block 1, block 2, …
- Because mining is a Markov process (i.e., memoryless), T follows an **exponential** distribution

$$f_\tau(t) = \alpha e^{-\alpha t}$$

where $\alpha = 1/t_0$

Poisson Law

The time needed to discover n blocks is $S_n = T_1+T_2+…+T_n$

N(t): #blocks validated at time t is **Poisson** with mean value αt

$$\mathbf{S}_n = \mathbf{T}_1 + \mathbf{T}_2 + \ldots + \mathbf{T}_n\,.$$

The random variable $\mathbf{S}_n$ follows the n-convolution of the exponential distribution and, as is well known, this gives a Gamma distribution with parameters (n, α),

$$f_{\mathbf{S}_n}(t) = \frac{\alpha^n}{(n-1)!} t^{n-1} e^{-\alpha t}$$

with cumulative distribution

$$F_{\mathbf{S}_n}(t) = \int_0^t f_{\mathbf{S}_n}(u)du = 1 - e^{-\alpha t} \sum_{k=0}^{n-1} \frac{(\alpha t)^k}{k!}\,.$$

From this we conclude that if $\mathbf{N}(t)$ is the process counting the number of blocks validated at time $t > 0$, $\mathbf{N}(t) = \max\{n \geq 0; \mathbf{S}_n < t\}$, then we have

$$\mathbb{P}[\mathbf{N}(t) = n] = F_{\mathbf{S}_n}(t) - F_{\mathbf{S}_{n+1}}(t) = \frac{(\alpha t)^n}{n!} e^{-\alpha t}\,,$$

Cyril Grunspan, Ricardo Pérez-Marco. The mathematics of Bitcoin. European Mathematical Society Newsletter, 2020. hal-02486029

Winning the race against the malicious

- **q**: hash rate of the malicious
- **N'(t)** = #malicious blocks at time t, which is Poisson
- **X_n = N'(S_n)** : #malicious blocks for every n consecutive honest blocks
  - S_n = time at which the honest has mined n blocks
  - X_n = a **negative binomial variable** with parameters (n, p)

$$\mathbb{P}[\mathbf{X}_n = k] = p^k q^k \binom{k+n-1}{k}\,.$$

How likely the malicious wins if behind the dishonest by z blocks? (similar to the classical **gambler's ruin problem**)

$$P(z) = I_{4pq}(z, 1/2)$$

where $I_x(a,b)$ is the incomplete regularized beta function

$$I_x(a,b) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} \int_0^x t^{a-1}(1-t)^{b-1}\,dt\,.$$

Cyril Grunspan, Ricardo Pérez-Marco. The mathematics of Bitcoin. European Mathematical Society Newsletter, 2020. hal-02486029

Probability to win the race

$\lambda = z\frac{q}{p}$

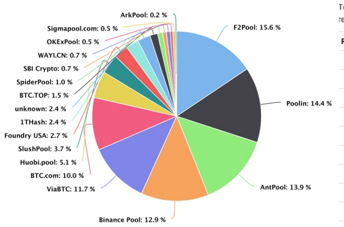
p = probability an honest node finds the next block
q = probability the attacker finds the next block

- The bad node and good node are **racing** to grow their blockchain to be the **longer**. Unless **q > 50%**, the bad node will win with probability

- Prob (attacker catch up) → 0 quickly as z large

$$\sum_{k=0}^{\infty}\frac{\lambda^k e^{-\lambda}}{k!}\cdot\left\{\begin{matrix}(q/p)^{(z-k)} & if\ k \le z\\ 1 & if\ k > z\end{matrix}\right\} = \boxed{1-\sum_{k=0}^{z}\frac{\lambda^k e^{-\lambda}}{k!}\left(1-(q/p)^{(z-k)}\right)}$$

Bitcoin Nodes



GLOBAL BITCOIN NODES DISTRIBUTION
Reachable nodes as of Sat Apr 17 2021 16:47:26 GMT+0700 (Indochina Time).

9796 NODES
24-hour charts »

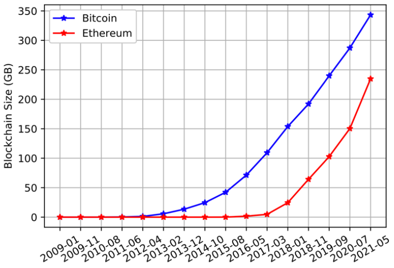
Top 10 countries with their respective number of reachable nodes are as follow.

RANK	COUNTRY	NODES
1	United States	1879 (19.18%)
2	n/a	1807 (18.45%)
3	Germany	1796 (18.33%)
4	France	616 (6.29%)
5	Netherlands	417 (4.26%)
6	Canada	321 (3.28%)
7	United Kingdom	290 (2.96%)
8	Russian Federation	260 (2.65%)
9	China	203 (2.07%)
10	Singapore	157 (1.60%)

Bitcoin is slow

- Block size = ~ 1MB
- Transaction size = ~250 bytes
- Each block = ~4,000 transactions
- Block rate = 1 per 10 minutes
- **Transaction rate** = 7 transactions per second
  - Particularly slow for applications that support real-time transactions
  - Visa: 2000-10,000 transactions/s
  - Paypal: 100 transactions/s

Blockchain storage size

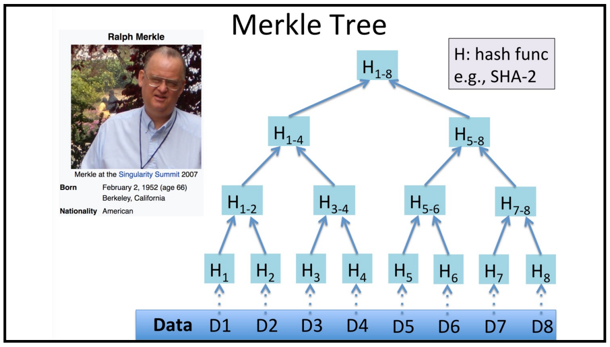


Lightweight Nodes

- Vast majority of nodes are **lightweight nodes**
  - Run the thin or Simple Payment Verification (SPV) client software
  - Don't store the full version of the blockchain
- For example, If you use a wallet program, it would typically incorporate an SPV node. Only store the block headers and transactions that represent payments to your addresses.

Too Much Disk Space?

- Recall, a coin = a chain of transactions
- Once the latest transaction is buried under enough blocks, we can discard the previous spent transactions to save disk space
- But would that break the block's hash?
- **Solution**
  - Create the block's hash from its transactions based on the Merkle Tree



Merkle Tree

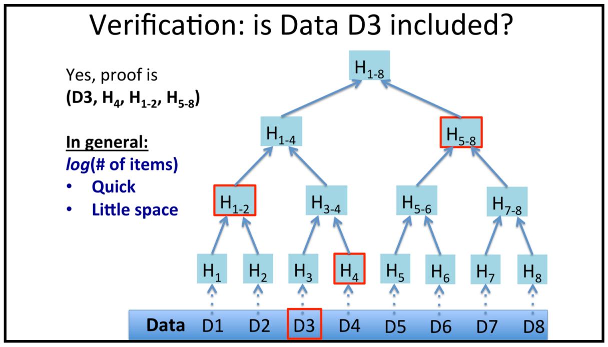
Ralph Merkle

Merkle at the Singularity Summit 2007

Born February 2, 1952 (age 66)
Berkeley, California

Nationality American

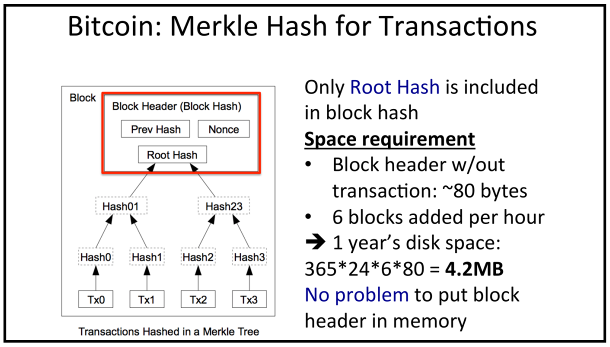
H: hash func
e.g., SHA-2



Verification: is Data D3 included?

Yes, proof is
(D3, H_4, H_{1-2}, H_{5-8})

In general:
***log*(# of items)**
- **Quick**
- **Little space**



Bitcoin: Merkle Hash for Transactions

Transactions Hashed in a Merkle Tree

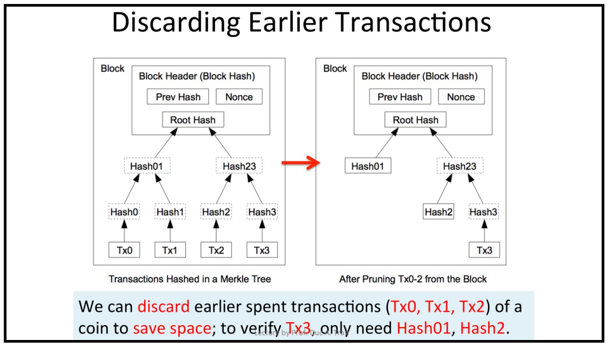
Only Root Hash is included in block hash

Space requirement

- Block header w/out transaction: ~80 bytes
- 6 blocks added per hour

➔ 1 year's disk space: 365*24*6*80 = **4.2MB**

No problem to put block header in memory



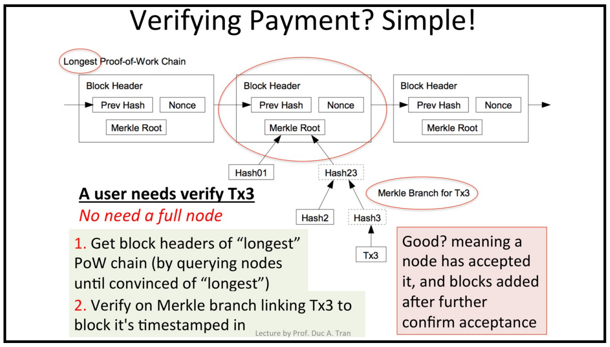
Discarding Earlier Transactions

Transactions Hashed in a Merkle Tree

After Pruning Tx0-2 from the Block

We can discard earlier spent transactions (Tx0, Tx1, Tx2) of a coin to save space; to verify Tx3, only need Hash01, Hash2.

Lecture by Prof. Duc A. Tran



Verifying Payment? Simple!

A user needs verify Tx3

No need a full node

1. Get block headers of "longest" PoW chain (by querying nodes until convinced of "longest")
2. Verify on Merkle branch linking Tx3 to block it's timestamped in

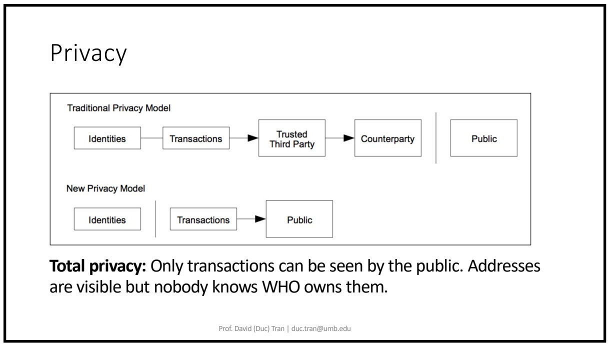
Good? meaning a node has accepted it, and blocks added after further confirm acceptance

Lecture by Prof. Duc A. Tran

Failed Verification?

- Possible, when network is overpowered by an attacker who fabricates transactions
  - Need to alert network nodes
- When a node is alerted
  - Download the full block and alerted transactions to confirm the inconsistency
- Businesses that receive frequent payments should run their own FULL nodes for more independent security and quicker verificaNon.

Prof. David (Duc) Tran | duc.tran@umb.edu



Privacy

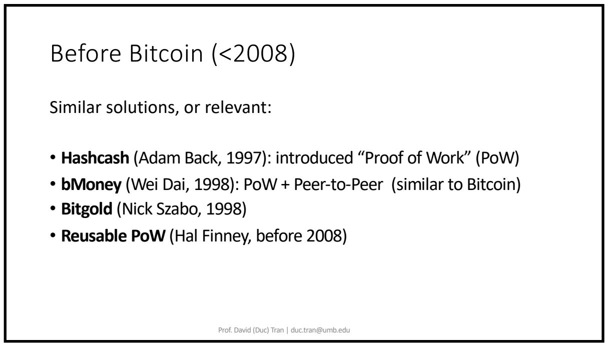
Traditional Privacy Model

Identities — Transactions → Trusted Third Party → Counterparty | Public

New Privacy Model

Identities | Transactions → Public

Total privacy: Only transactions can be seen by the public. Addresses are visible but nobody knows WHO owns them.



Before Bitcoin (<2008)

Similar solutions, or relevant:

- **Hashcash** (Adam Back, 1997): introduced “Proof of Work” (PoW)
- **bMoney** (Wei Dai, 1998): PoW + Peer-to-Peer (similar to Bitcoin)
- **Bitgold** (Nick Szabo, 1998)
- **Reusable PoW** (Hal Finney, before 2008)