Proofs of Work, Bitcoin Protocols, and Mining

CMSC 23280/ECON 23040, Winter 2019
Lecture 3

David Cash, Harald Uhlig, Ben Zhao

University of Chicago

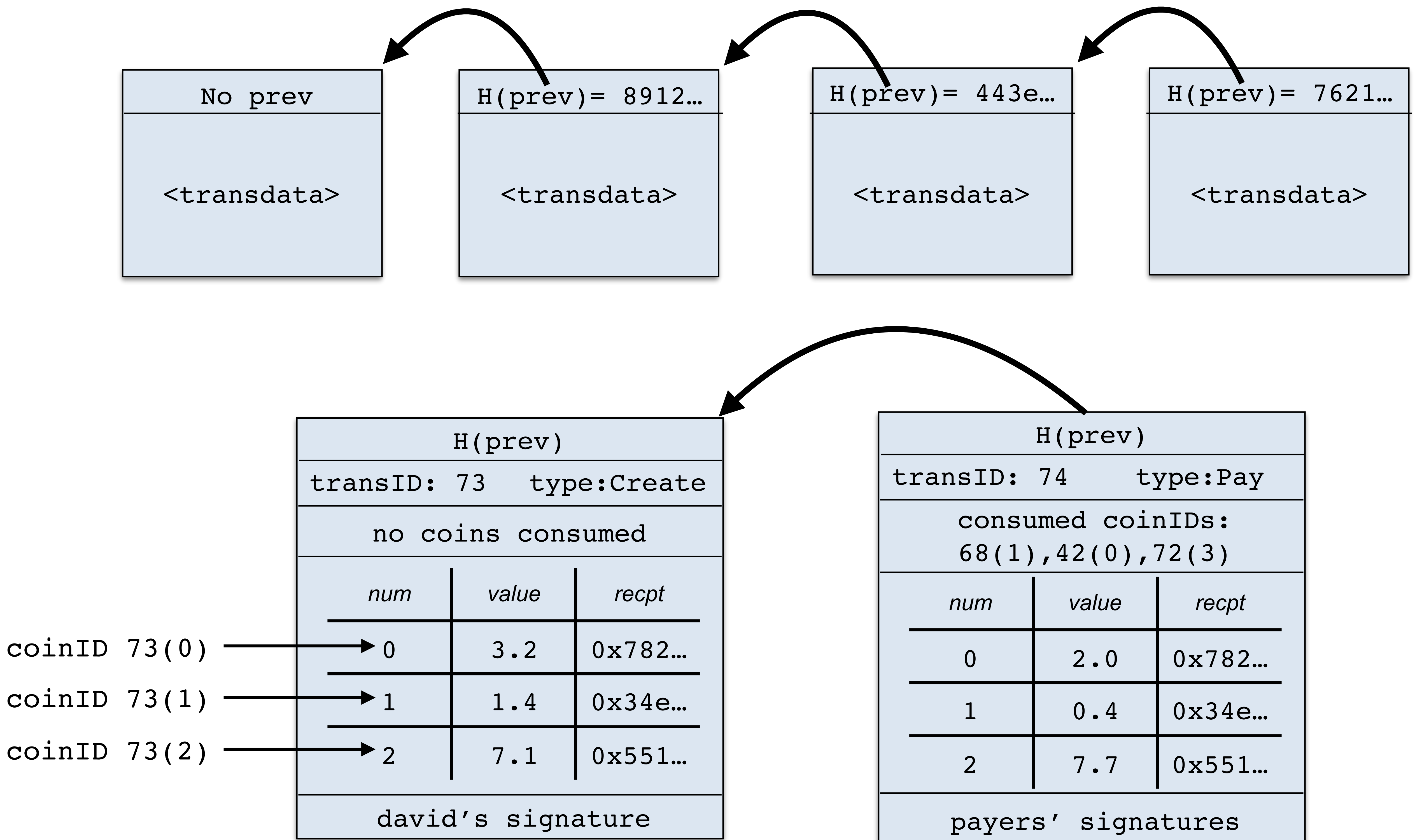
Lecture 3 Outline

1. Decentralized DCash via proofs-of-work
2. Some details about Bitcoin
   a. Transactions and scripts
   b. The Bitcoin network and types of nodes
3. Mining

Lecture 3 Outline

1. **Decentralized DCash via proofs-of-work**
2. Some details about Bitcoin
   a. Transactions and scripts
   b. The Bitcoin network and types of nodes
3. Mining

Recall: Transaction-Based Blockchain



Consensus with an angel (text section 2.3)

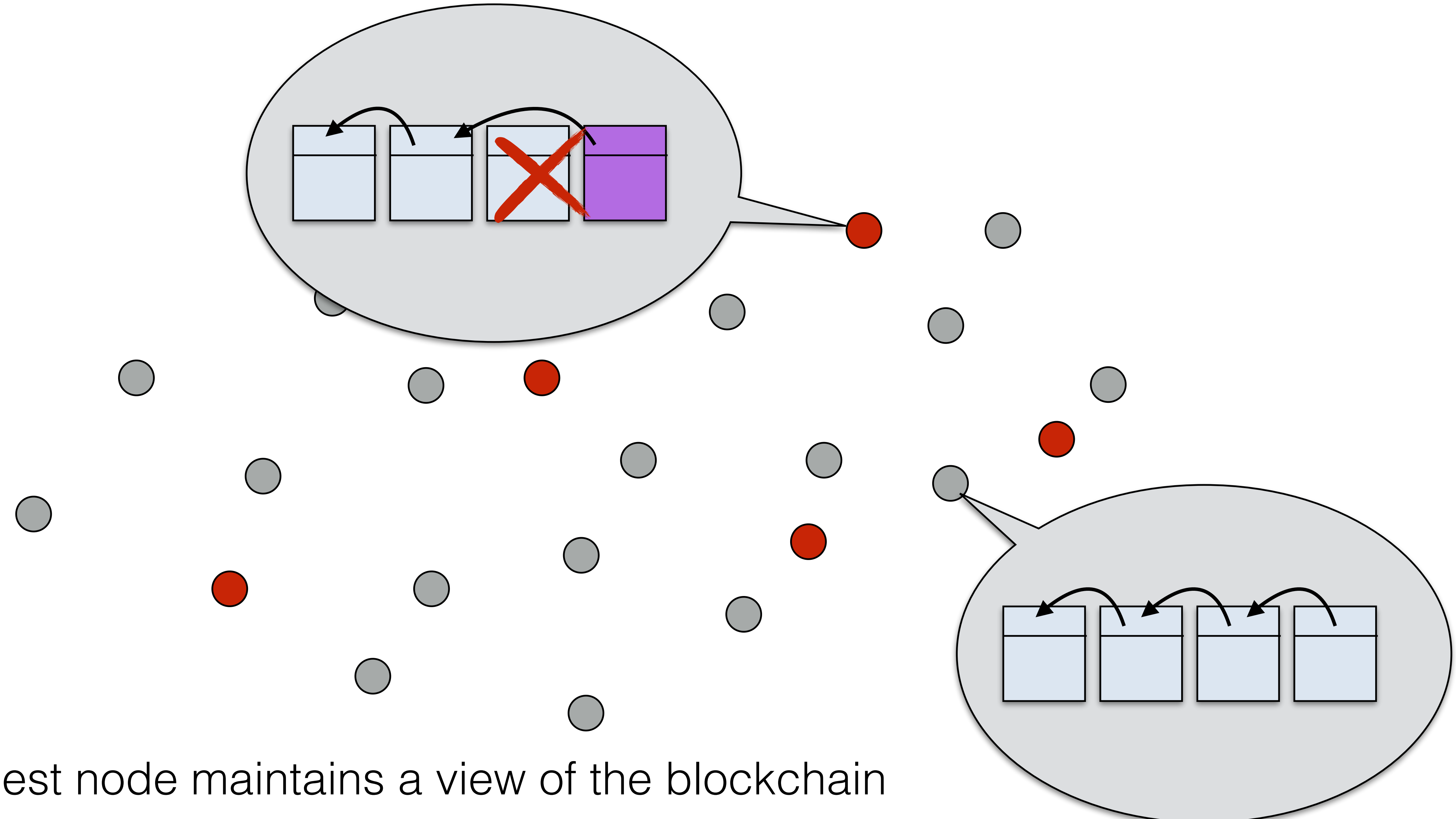
Consensus protocol with angel:

1. Transactions are broadcast to everyone
2. Each node collects transactions into a block
3. At end of round, angel picks the leader
4. The leader adds a block of valid transactions to their personal view.
5. The leader announces their personal view of the blockchain.
6. Everyone else accepts the announced view as their own if:
   a. The transactions in this view are all valid
   b. The announced chain has **more blocks** than their view.

Note 1: When a node is chosen, it can always add a block to its view. It's up to the other nodes to accept it or not.

Note 2: A node may *only* add a block to its view when the angel chooses it.

Upshot: “the” honest chain vs “the” malicious chain

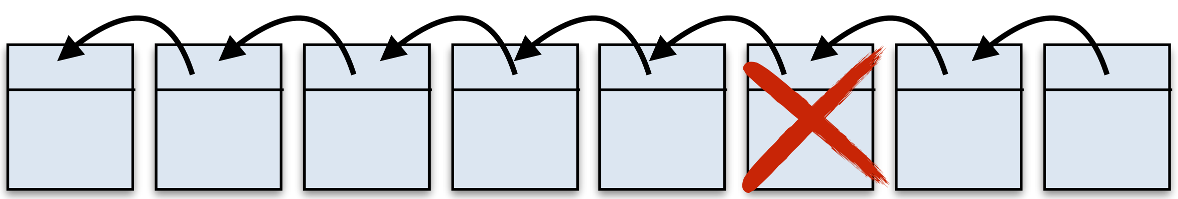


- Each honest node maintains a view of the blockchain
- Honest nodes switch to any longer chain they see
- Malicious nodes can switch blockchain if they get lucky and build their own longer chain
  - Even when they get lucky, malicious nodes can’t insert fraudulent transactions

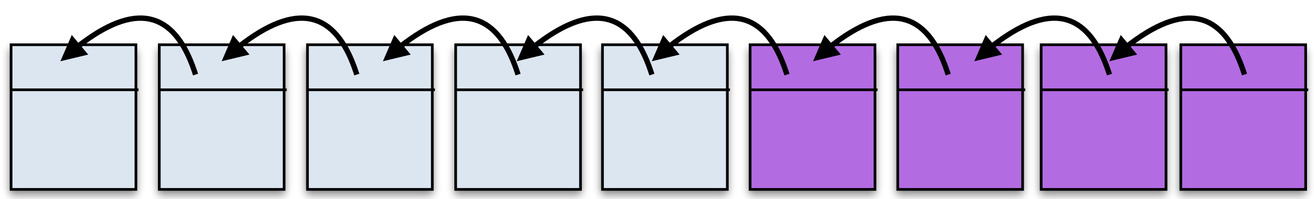
Formal setting: Honest vs Malicious Chain

- `p` = probability that angel picks a honest node
- `q = 1-p` = probability that angel picks malicious node

“Consensus” chain grows at rate `p`:



“Malicious” chain grows at rate `q`:



<u>Theorem:</u> Assume `p>q`. Once the honest nodes are “ahead” of the malicious nodes, the chance the malicious nodes will ever “catch up” decreases *exponentially* in the size of the lead.

- If lead is small (ex: 1 block) then malicious nodes may catch up often
- But for large leads, malicious nodes should give up and move to consensus chain

Problem to solve later: Sybil attacks

Vulnerable to Sybil attacks: If there are a ton of malicious nodes, then the probability a malicious node is chosen is high.

Create tons of malicious nodes → Angel picks malicious leaders → Get ahead, delete blocks → Malicious nodes can "undo" transactions after buying things.

Implementing the Angel: Big Questions

- How does the network pick a random node?
  - No notion of “membership”
  - No one is in charge
  - Attacks are motivated to influence choice
- How often do we pick a leader?

Nakamoto's insight: Replace the Angel using POWs

POW syntax:

Hardness parameter: Integer `z`
Input: string `x`
Solution: string `c` such that `H(x,c)` starts with `z` zeros

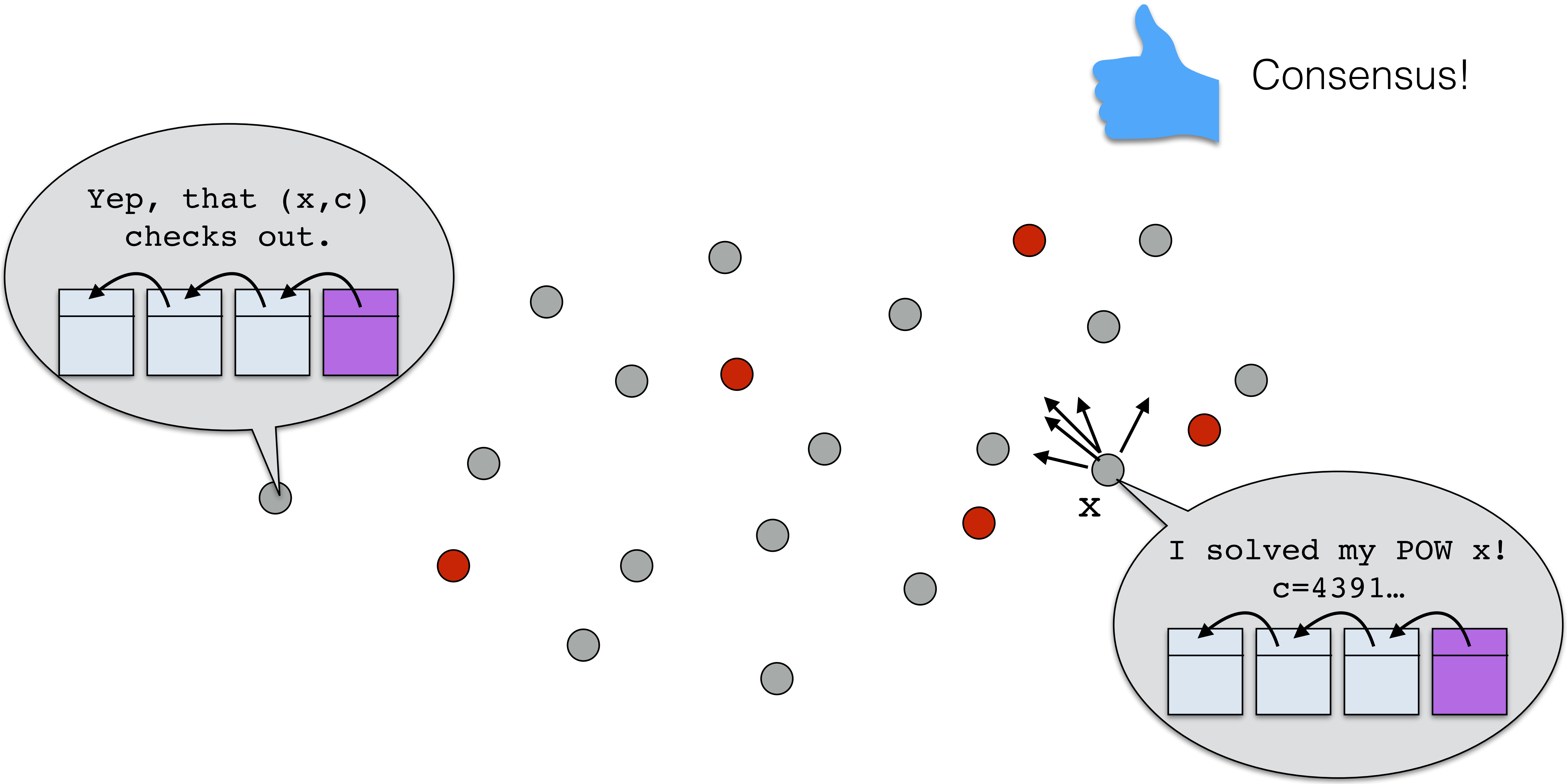
Angel chooses node ⟷ Node solves POW first

- "Randomness" comes from cryptographic hash function `H`

In each round:

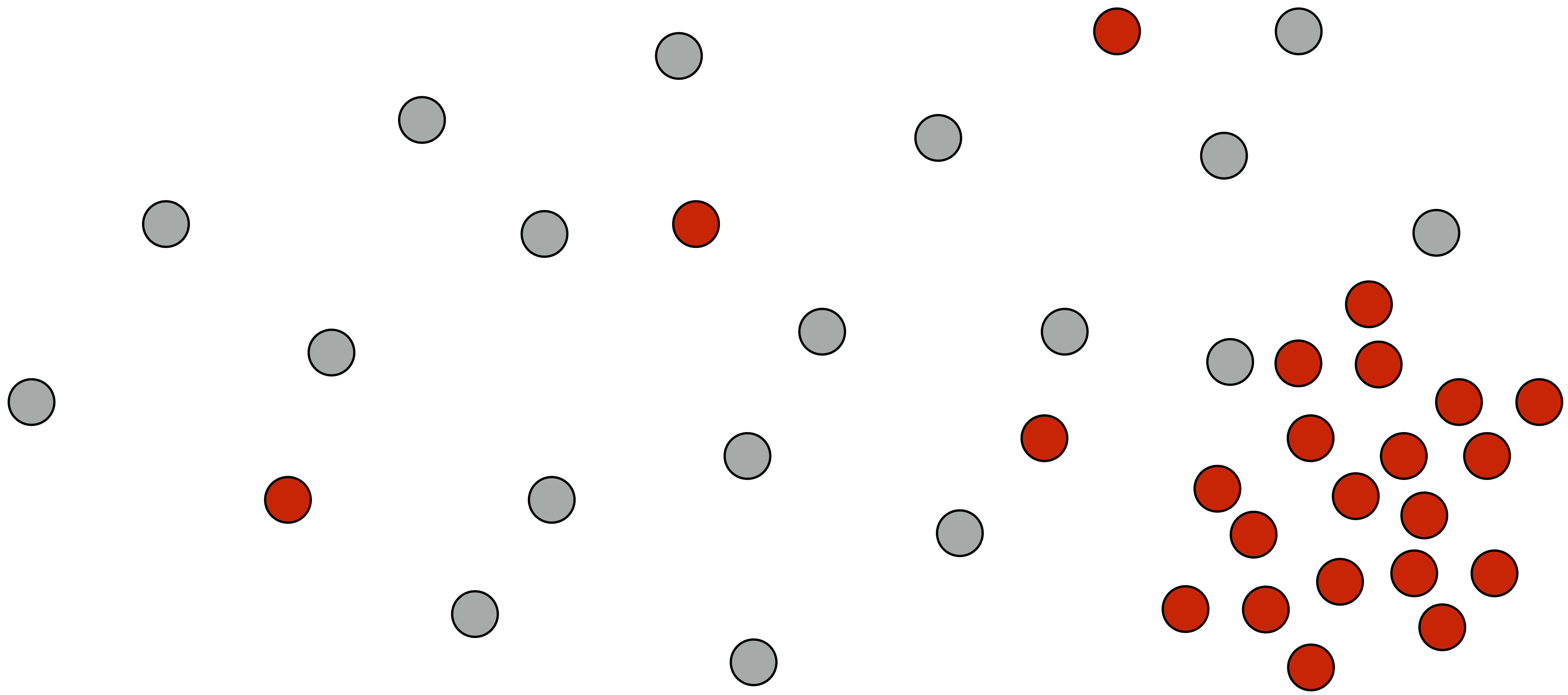
1. Every node forms its own POW puzzle input `x` and tries to solve it.
2. First node to solve their POW puzzle is the leader.
   a. Node announces `c`, and everyone else checks POW solution.
   b. Node announces next block of transactions, which everyone also checks.
   c. All nodes accept a broadcasted POW solution if the chain is the longest they've seen.

Example: Leader selection via POW puzzles



- All nodes work on their POW puzzles
- Eventually one solves their puzzle, and announces the solution along with their blockchain view
- Everyone checks that the POW puzzle is correct and accepts block as before

Sybil Attacks Are Defeated (i.e. the Angel is Smart)



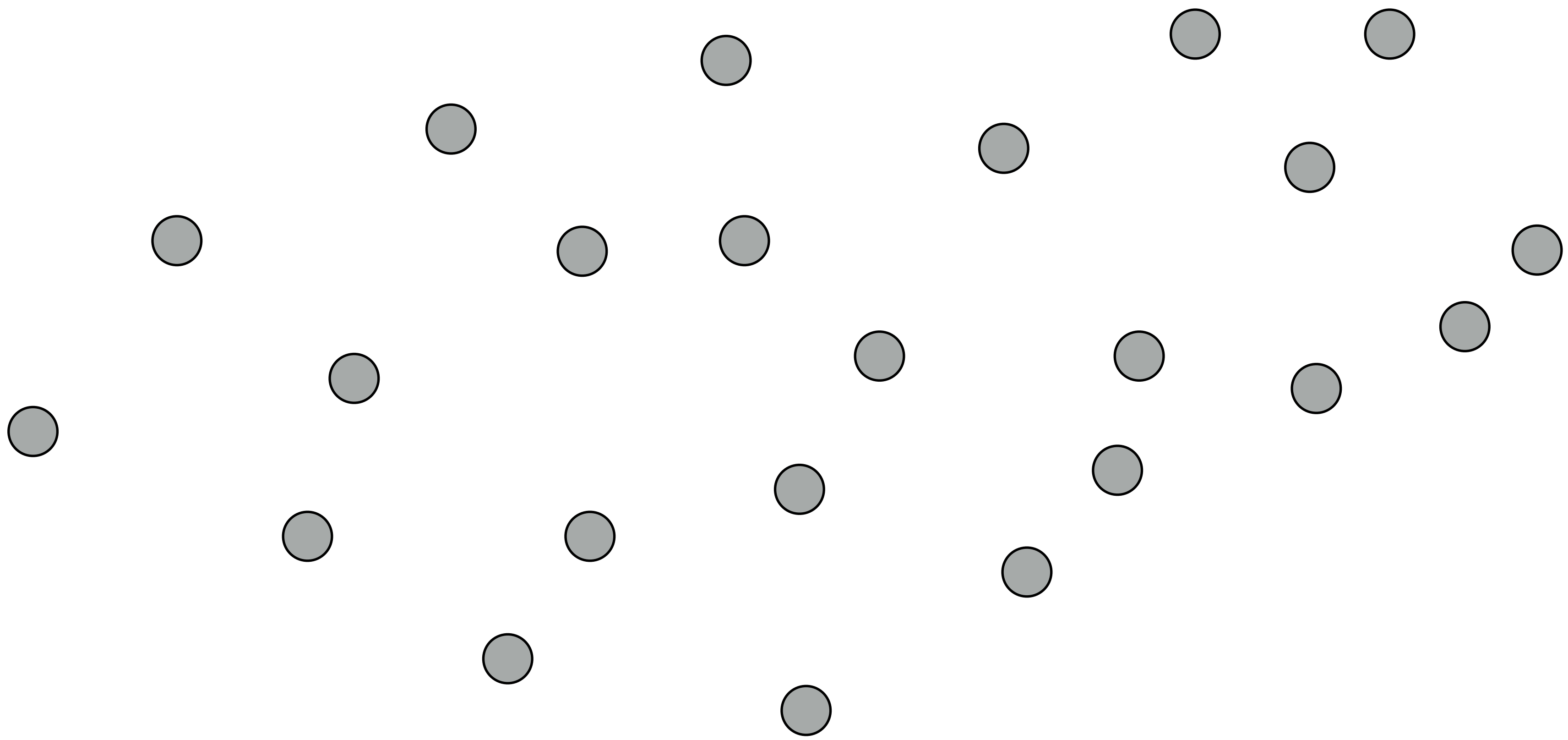
Having lots of nodes doesn't help - You have to solve the POW puzzles, so all that matters is your compute power.

$$\Pr[\text{agent solves POW first}] \approx \frac{\text{agent's compute power}}{\text{network's total compute power}}$$

Several Questions Remain

1. How do nodes hear about solved POWs?
2. How does each node decide on their POW puzzle input **x**?
3. Why is everyone willing to solve the POW puzzles? They cost real money.
4. What if malicious nodes can always solve the POW puzzles first?
5. Does this *really* work?

Peer-to-Peer Network Basics



1. New nodes join via hardcoded “seed nodes” or DNS
2. Information is exchanged by local rules
    1. Advertise neighbors to anyone who asks
    2. As info comes in, broadcast it to neighbors
    3. Do not broadcast the same info twice (avoid loops)
3. Have large latency compared to standard networks

Questions? Ask Ben!

Proof of Work Input Details

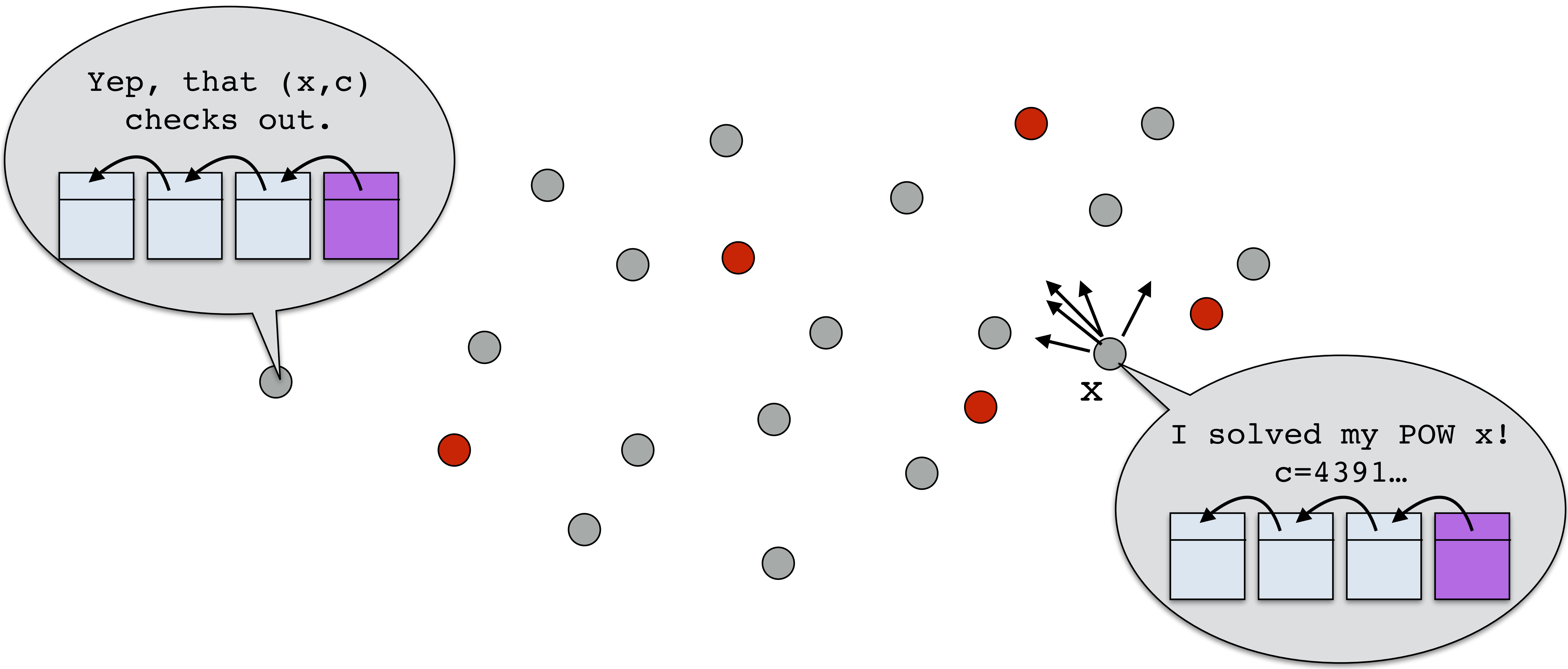
How to form POW input `x`:

1. Each node maintains a public key `PK` as their "identity" (can change).
2. Each node collects broadcasted valid transactions since last block.
3. Compute `x`:

```
x = (PK,trans1,trans2,trans3,…)
```

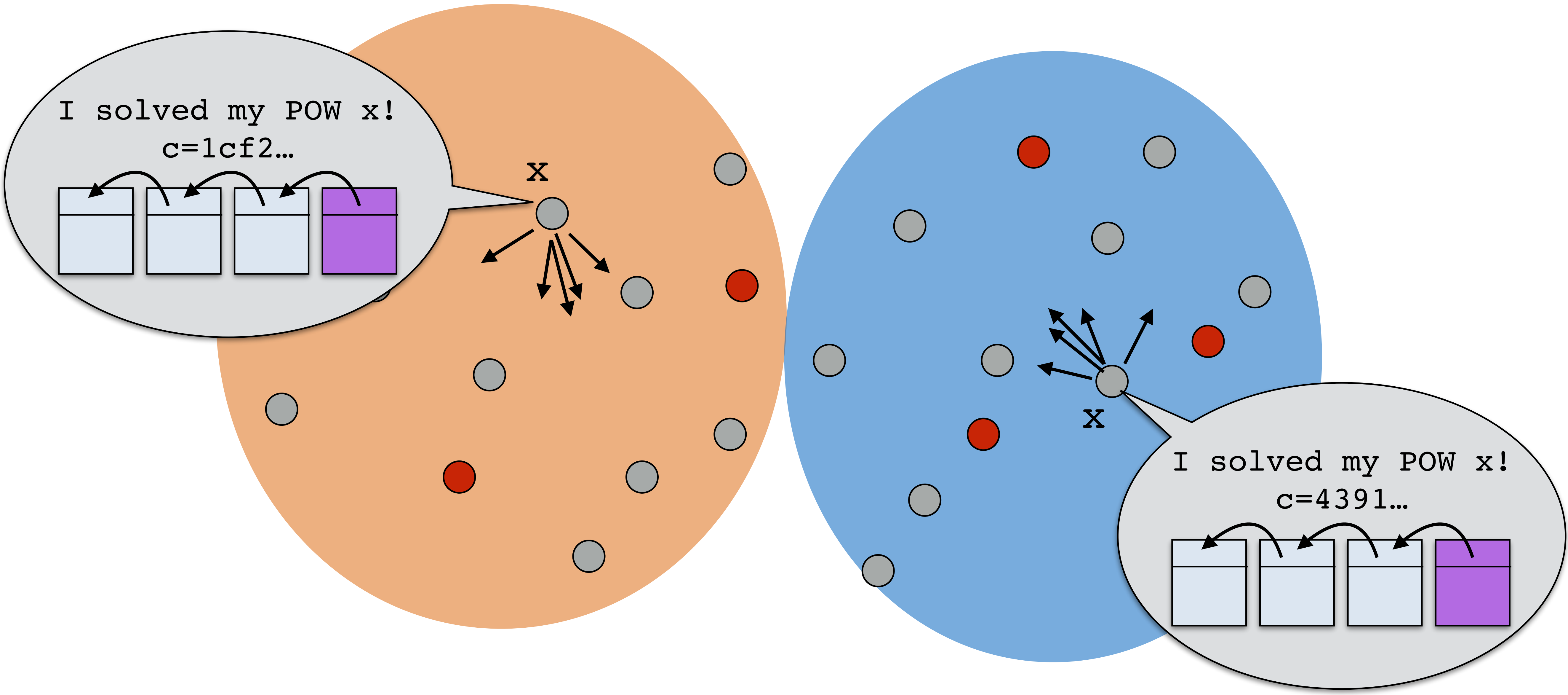
- Many details in practice: Limit on number of transactions, `x` is actually a special type of hash output (more on this later).
- Question going forward: What can a malicious node do?

We need honest nodes to solve POWs…



- Honest blocks solving POWs faster than malicious nodes means the honest chain grows faster
- Even if malicious nodes occasionally solve a POW first, all they can do is pick next block.

… but the POWs can't be too easy



- If POWs solved close together, network will "fork" (because nodes accept longest chain they've seen)
- Happens even if all nodes are honest due to latency.
- <u>Solution</u>: Make POWs hard enough so that this unlikely to happen several times in a row. System becomes self-healing.

But why would honest nodes work so hard?

- POWs cost money: Hardware, electricity
- Nodes are incentivized to not even try: If someone else will do it, why should I?

Cryptographer's idea: Let's *force* them to! But how…

Economist's idea: Let's print money and pay them!

How to pay nodes for solving POWs (Part 1)

- Every block contains a special **`createCoin`** transaction for a set amount.
  - This created coin is the *block reward*.
- The solver of the POW gets to pick the recipient (themselves).

`H(prev)`

Special transaction giving `1.0` coins to POW solver.

`transID: 73    type:Create`

num	*value*	*recpt*
`0`	`1.0`	`0x5af…`

Other transactions that POW solver included in block.

`transID: 74    type:Pay`

`consumed coinIDs: 68(1),42(0),72(3)`

num	*value*	*recpt*
`0`	`2.0`	`0x782…`
`1`	`0.4`	`0x34e…`
`2`	`7.7`	`0x551…`

`payers' signatures`

- Incentivizes nodes: Solve the POW, announce a valid block, get paid!
- This is called *mining*. The POW solver is the *miner*.

Mining-only pay incentivizes ignoring transactions

- If miners only get paid for mining blocks, then it puts them at a disadvantage to wait for transactions to include.
- But if miners don’t include transactions, then payments never clear.

How to pay nodes for solving POWs (Part 2)

- Solution: Transactions include a transaction fee (a.k.a. tip for the miner)

Tip is implicit.
If this transaction has
`input > output`
then miner keeps
the difference.

`H(prev)`

`transID: 73    type:Create`

num	*value*	*recpt*
0	1.0	0x5af…

`transID: 74    type:Pay`

`consumed coinIDs: 68(1),42(0),72(3)`

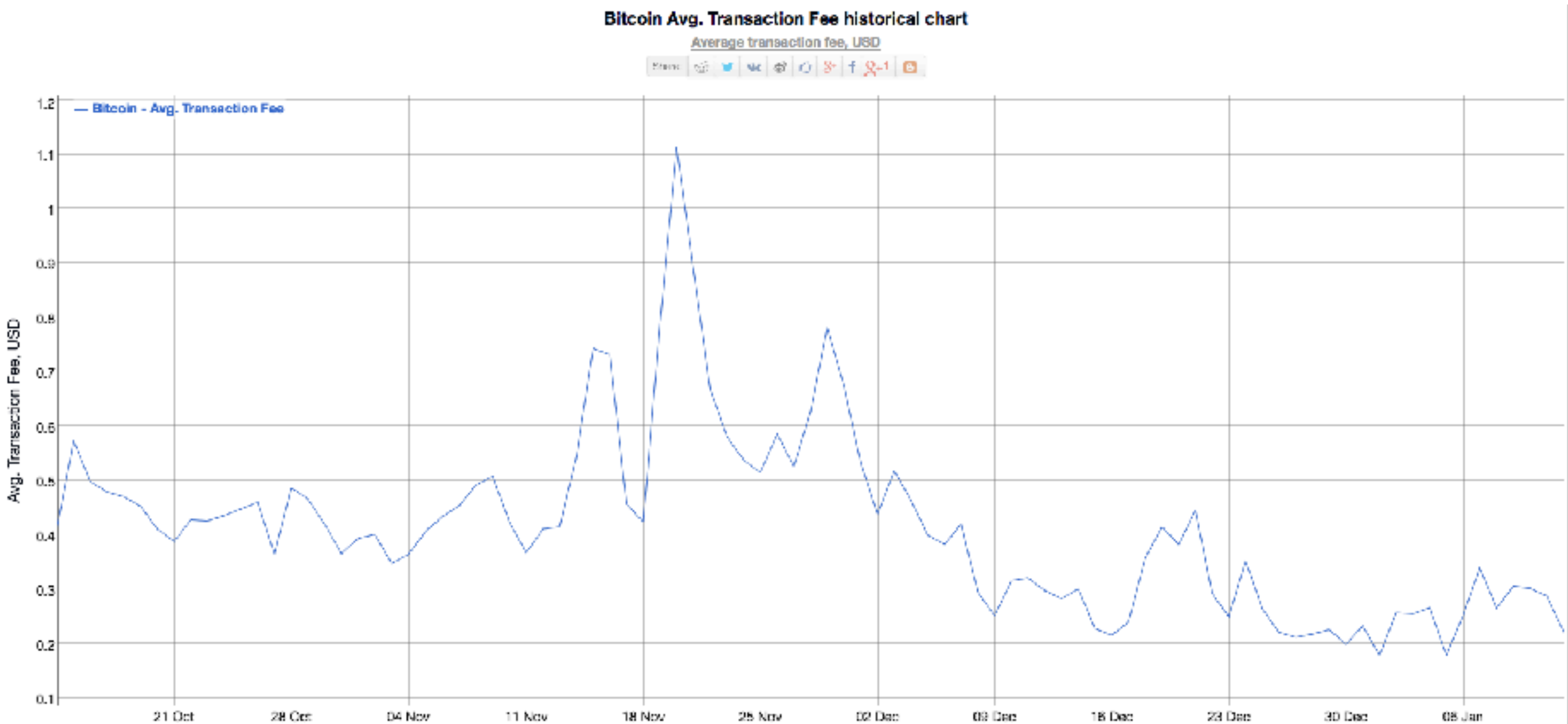
num	*value*	*recpt*
0	2.0	0x782…
1	0.4	0x34e…
2	7.7	0x551…

`payers' signatures`

- Incentivizes nodes to include transactions: Get the fees!

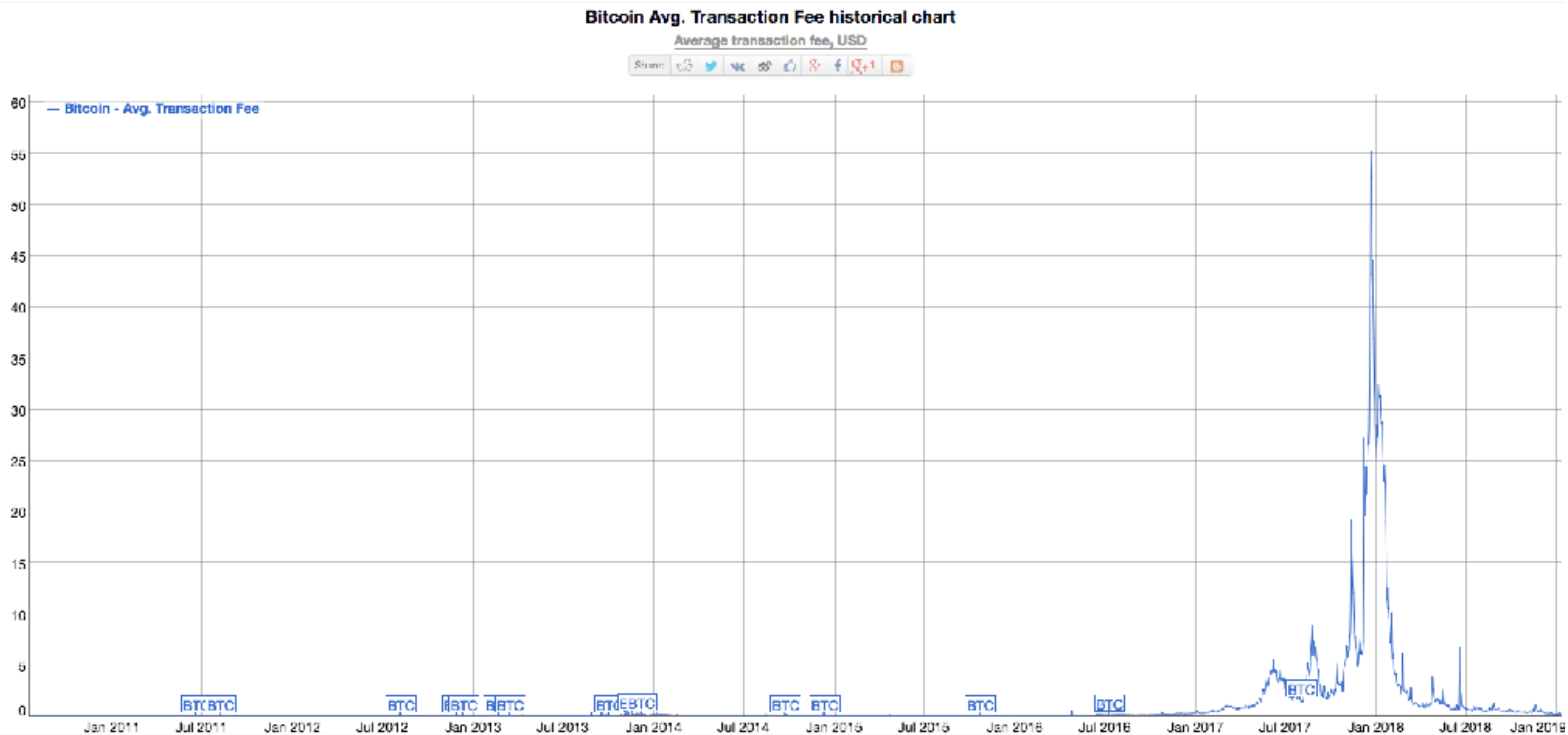
The economics of transaction fees

- Not like usual tipping: Transaction fees are bids in an auction.
  - Blocksize is limited and a miner will choose transactions with highest tips.
  - Side note: Miners don’t need to pay fees to include their own transactions.

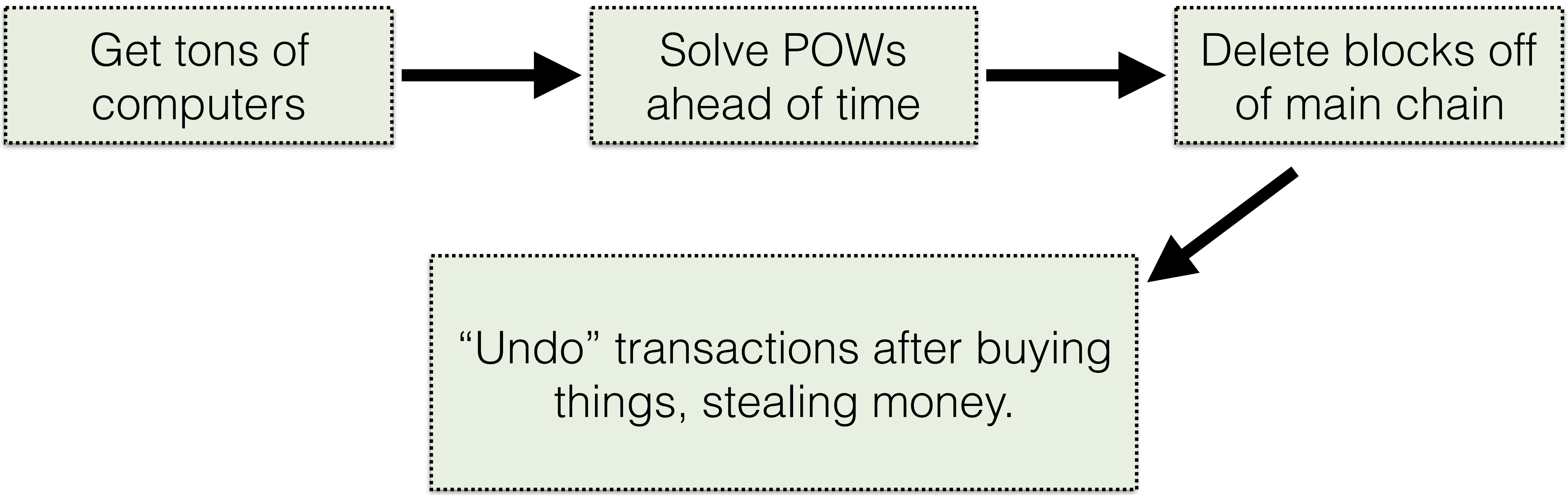


The economics of transaction fees

- Transaction fees are volatile



How to attack POW-based consensus?

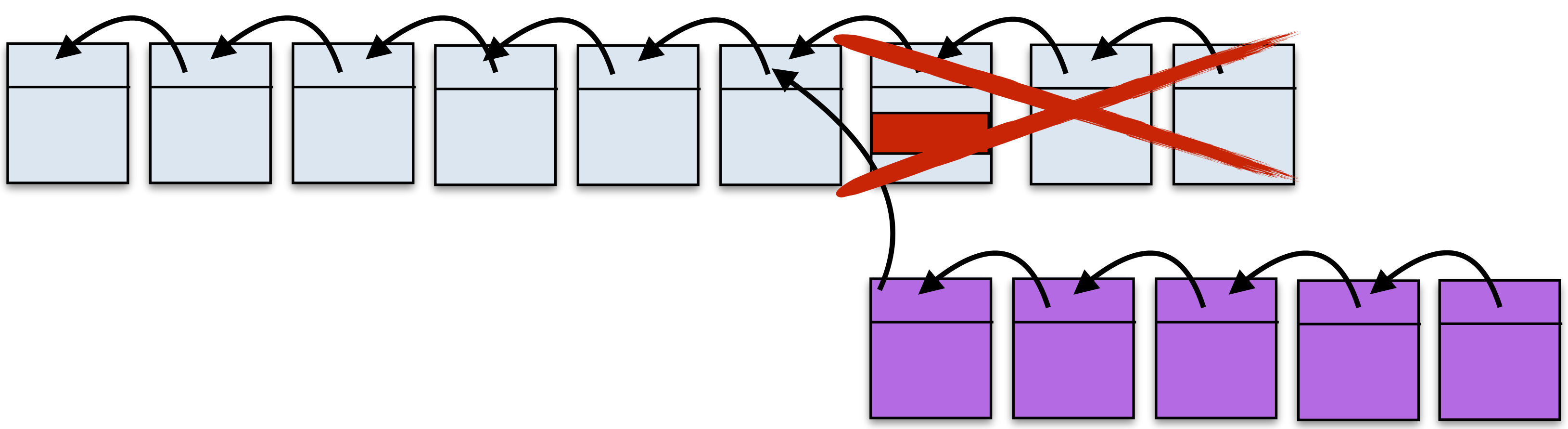


51% Attacks

Suppose some agent has the majority of compute power on the network:

$$\Pr[\text{agent n solves POW first}] \approx \frac{\text{agent n's compute power}}{\text{network's total compute power}} \geq 0.51$$

<u>On consensus blockchain (public):</u> Agent spends a coin in block that is accepted.



1. Good is delivered after a few blocks confirm transaction
2. Agent then privately mines several blocks ahead of the consensus chain
3. Agent announces this longer chain, switching network and deleting its transaction.
4. The good has been delivered and is lost.

51% Attack Mitigation

<u>One argument</u>: 51% attacks cannot be profitable. If someone achieves 51% power, they will destroy trust in the blockchain, thereby destroying the value of the coin and any profits.

Bitfly
@etherchain_org

Follow

We can confirm that there was a successful 51% attack on the Ethereum Classic (#ETC) network with multiple 100+ block reorganization. We recommend all services to closely monitored the chain and significantly increase required confirmations.

9:33 AM - 7 Jan 2019

431 Retweets **505** Likes

60 431 505

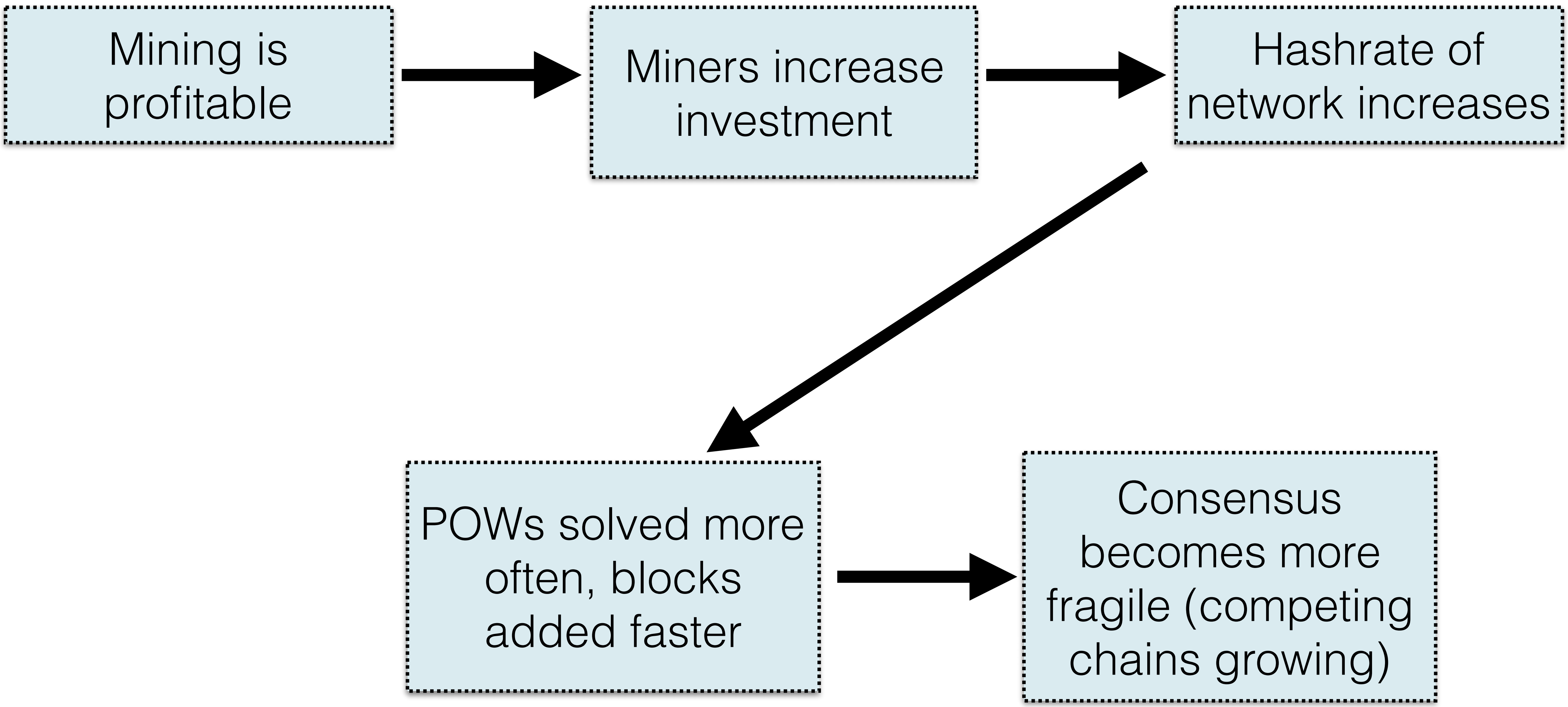
ETHEREUM

Ethereum Classic [ETC] 51% attack: Hackers return $100,000 in tokens

Published 13 hours ago on January 14, 2019
By **Namrata Shukla**

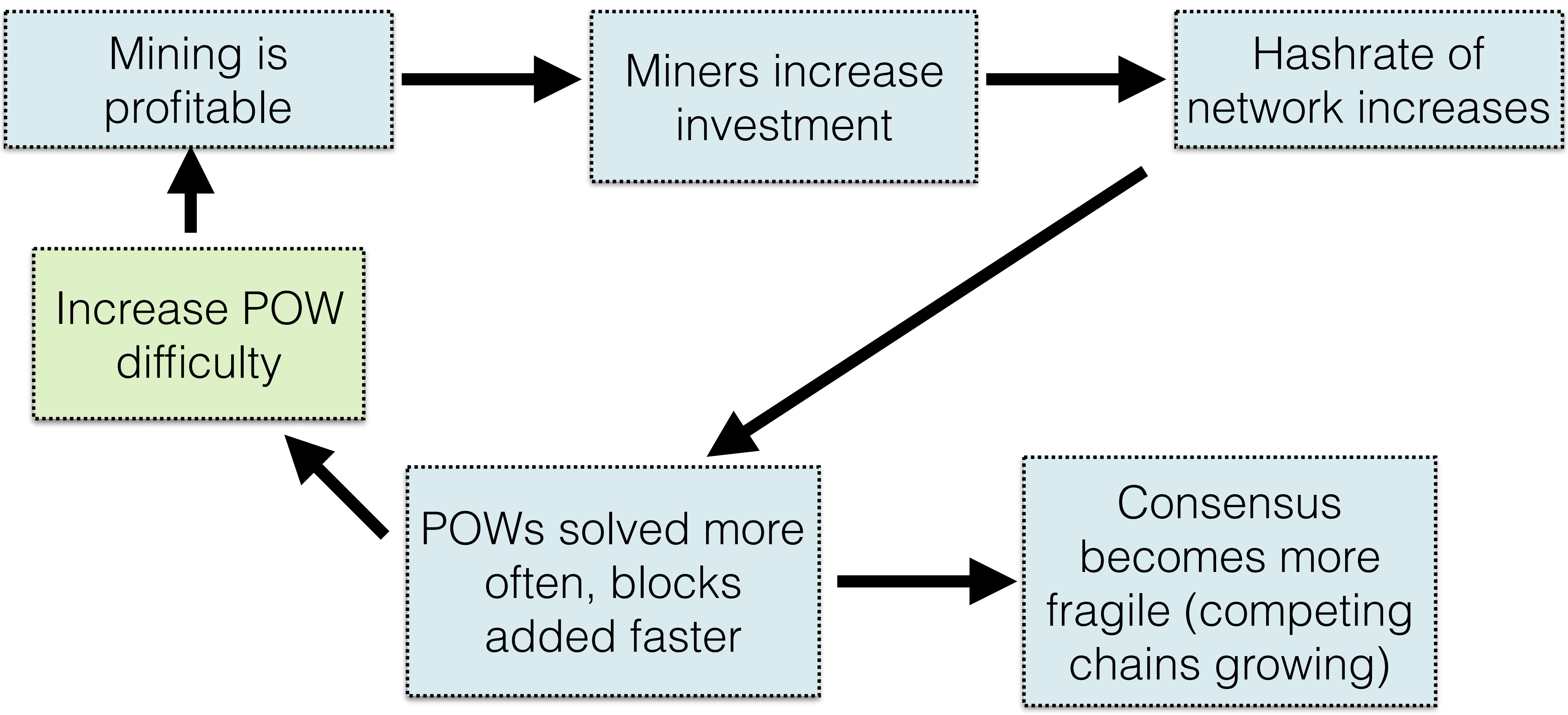
Mining Difficulty Over Time

- Hashrate: Total power of network in terms of evaluations of `H` per second
- Mining has fixed and incremental costs:
  - Fixed: Hardware
  - Incremental: Electricity (computers + cooling), repairs, management



Nakamoto's Solution: Increase Difficulty Gradually

- Increase POW difficulty occasionally to throttle speed of block additions



Summary: POW-based Consensus

1. Nodes all work on solving their POW puzzles.
2. Winners announce blocks and show everyone they solved their POW.
3. Nodes accept blocks if transactions are valid, the POW is solved, and it is the longest block they have seen.
4. Miners are incentivized by block rewards and transaction fees.
5. Mining effort can increase over time

Lecture 3 Outline

1. Decentralized DCash via proofs-of-work

2. Some details about Bitcoin

   a. Transactions and scripts

   b. The Bitcoin network and types of nodes

3. Mining

Bitcoin: Some Numbers

Bitcoin Blocks
- 1MB of data each
- ~1700 transactions

"The" Bitcoin Blockchain
- More than 558,000 blocks long.
- 200 GB of data
- Blocks added every 10 minutes
- ~17.4 million bitcoins in system

Bitcoin proof-of-work
- Slight variant of the "zeros" POW
- ~76 zeros required now

"The" Bitcoin Network
- ~10,000 "full" nodes
- Tens of seconds to propagate transactions
- Tx fee = 0.3 USD
- Mining consumes electricity equivalent to Singapore (0.21% global total)

- To the blockchain!

Bitcoin Scripts: The Proto-Smart-Contracts

- Bitcoin allows transaction that do more than simply pay to a public key
- There is an entire (limited) *scripting* language to specify transaction validity

Transaction Format:
  InputCoins: …
  OutputCoins: …
  Valid if the following script returns true:

```
DUP
HASH160
PUSHDATA(20)[0f0922…]
EQUALVERIFY
CHECKSIG
```

- Verifying the chain requires running scripts
- Other cryptocurrencies allow for more powerful “smart contracts” - More later.

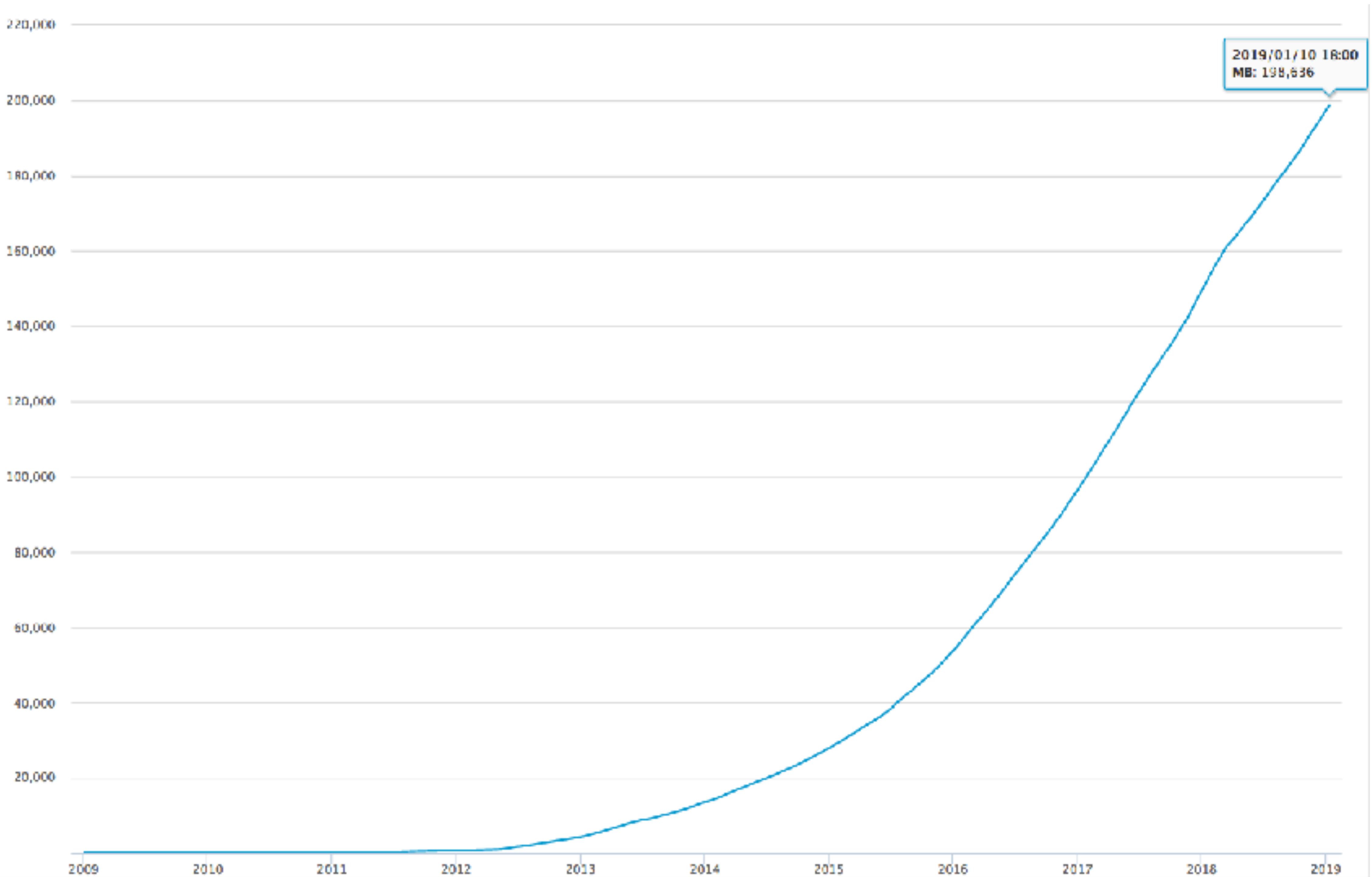
Bitcoin Script Example: Multiple Signatures

- MULTISIG script command does the following:

> Inputs:
> - Integers n, t $(t \le n)$
> - Public keys $PK_1, \ldots, PK_n$
> - Public keys $\sigma_1, \ldots, \sigma_t$
>
> Output:
>
> ACCEPT if $\sigma_1, \ldots, \sigma_t$ are signatures under t different public keys from the list

- Example: `n=3, t=2` means two of the three keys must sign transaction
- Used for escrow: Two parties use a third party to resolve disputes.
  - Normal operation: Both parties sign
  - Dispute resolution: One party + third party signs

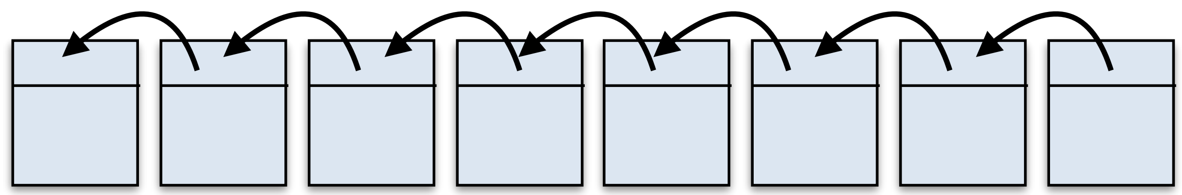
Blockchain Size Over Time



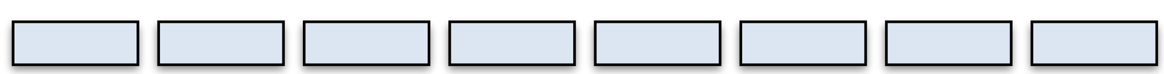
Around 200 gigabytes today. (1 megabyte per block, ~1700 transactions/block)

Types of Nodes on the Bitcoin Network

- Full Nodes — Store all 200GB of blocks and verify all transactions

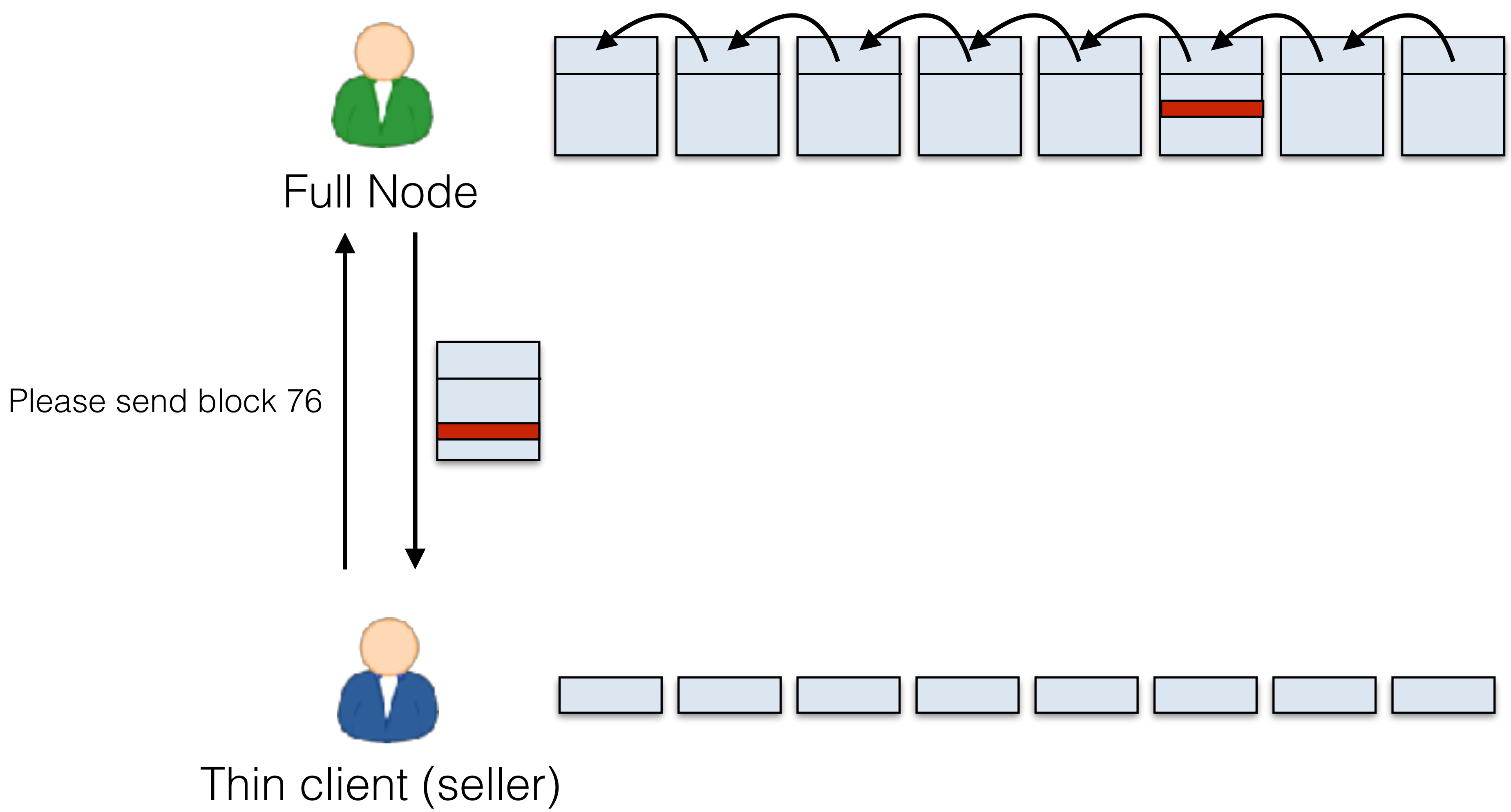


- Mining Nodes — May or may not store blockchain, but actively solve POWs
- Thin/Lightweight Nodes — Store only hashes of all blocks
    - 80 bytes per block, ~45 megabytes total
    - Perform limited type of verification
    - Called “SPV” in Bitcoin: Simplified Payment Verification
    - Almost all nodes are SPV clients



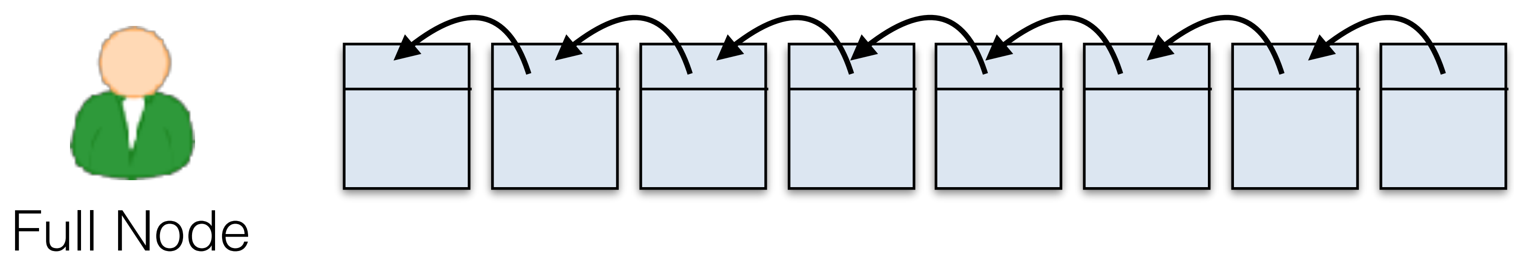
SPV Clients on the Bitcoin Network

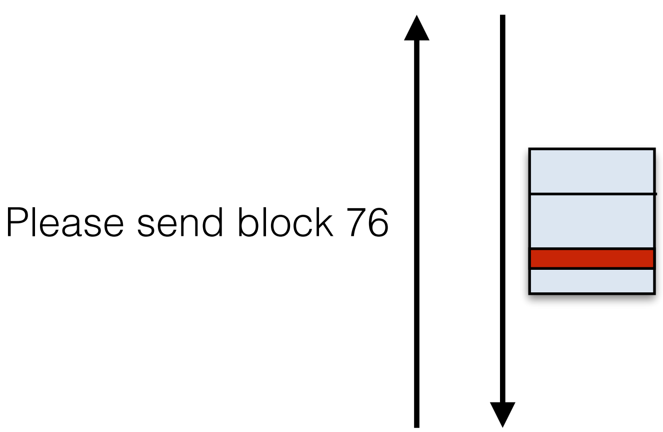
- SPV clients do not hold entire blockchain, so they cannot verify transactions
- They trust that the full clients and miners have done that work
- But SPV clients *do* verify their incoming transactions, with the help of full nodes



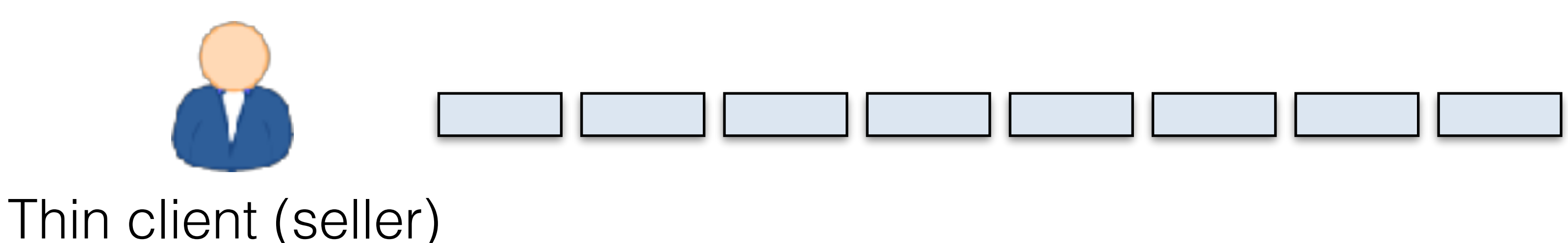
- Before sending goods, hash block and check that transaction is in blockchain with some confirmations

SPV Clients on the Bitcoin Network

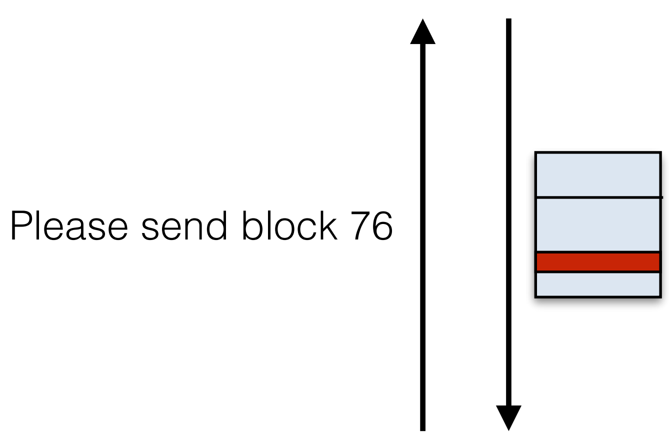


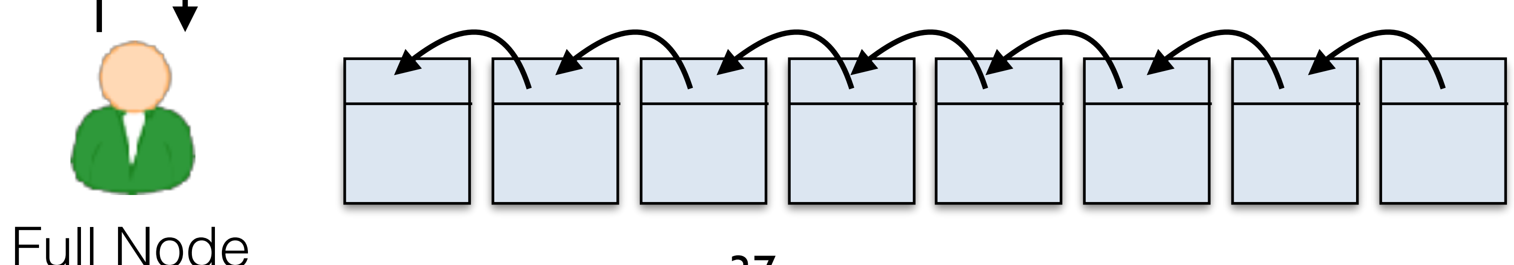


- Verify with several nodes (run by different people)
- If everyone sends same block with your transaction, trust that it has been verified



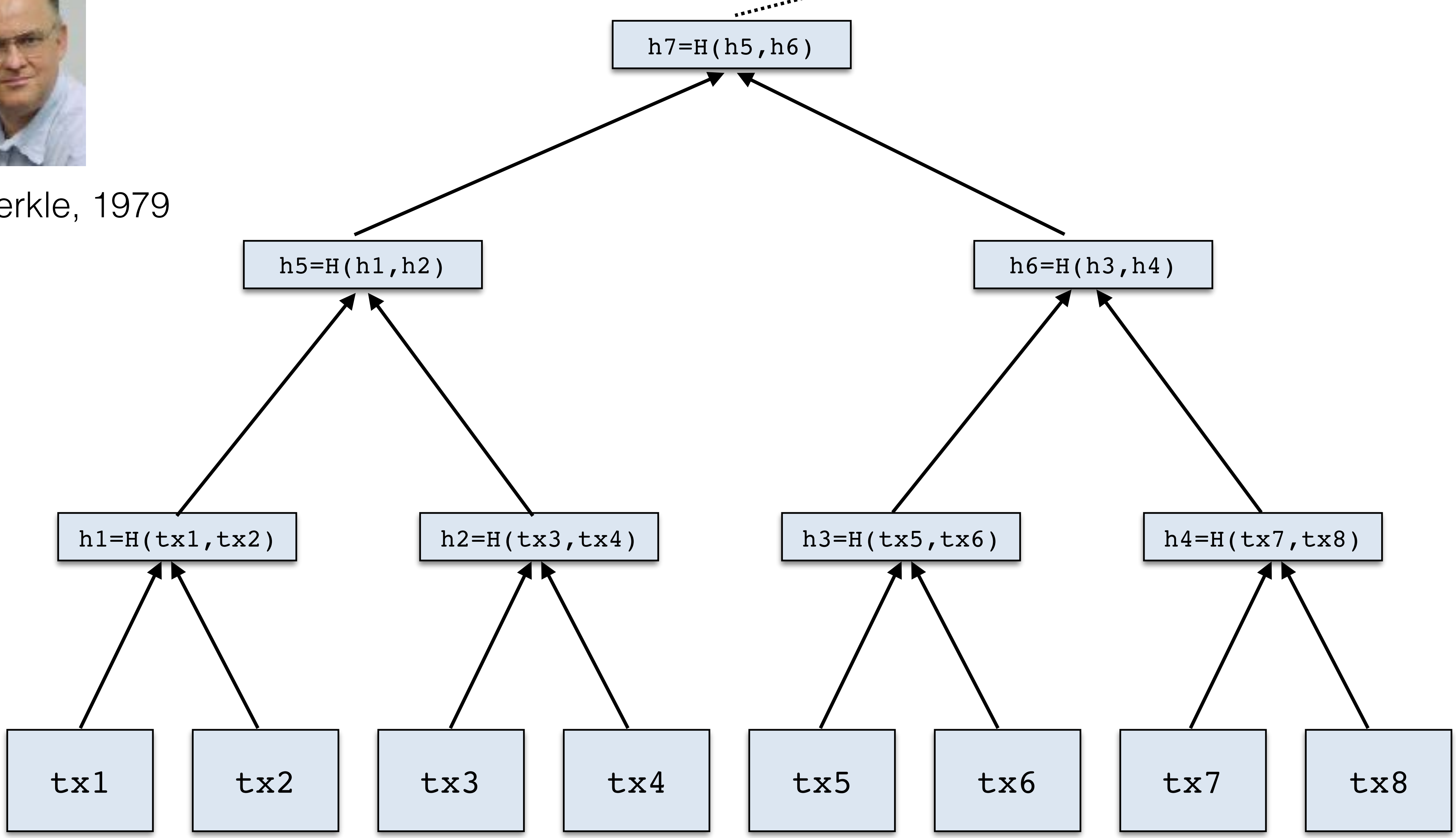
Problem: Each block is 1MB, and millions of SPV clients request blocks constantly, requiring lots of bandwidth from servers.





Crypto Trick: Merkle Trees

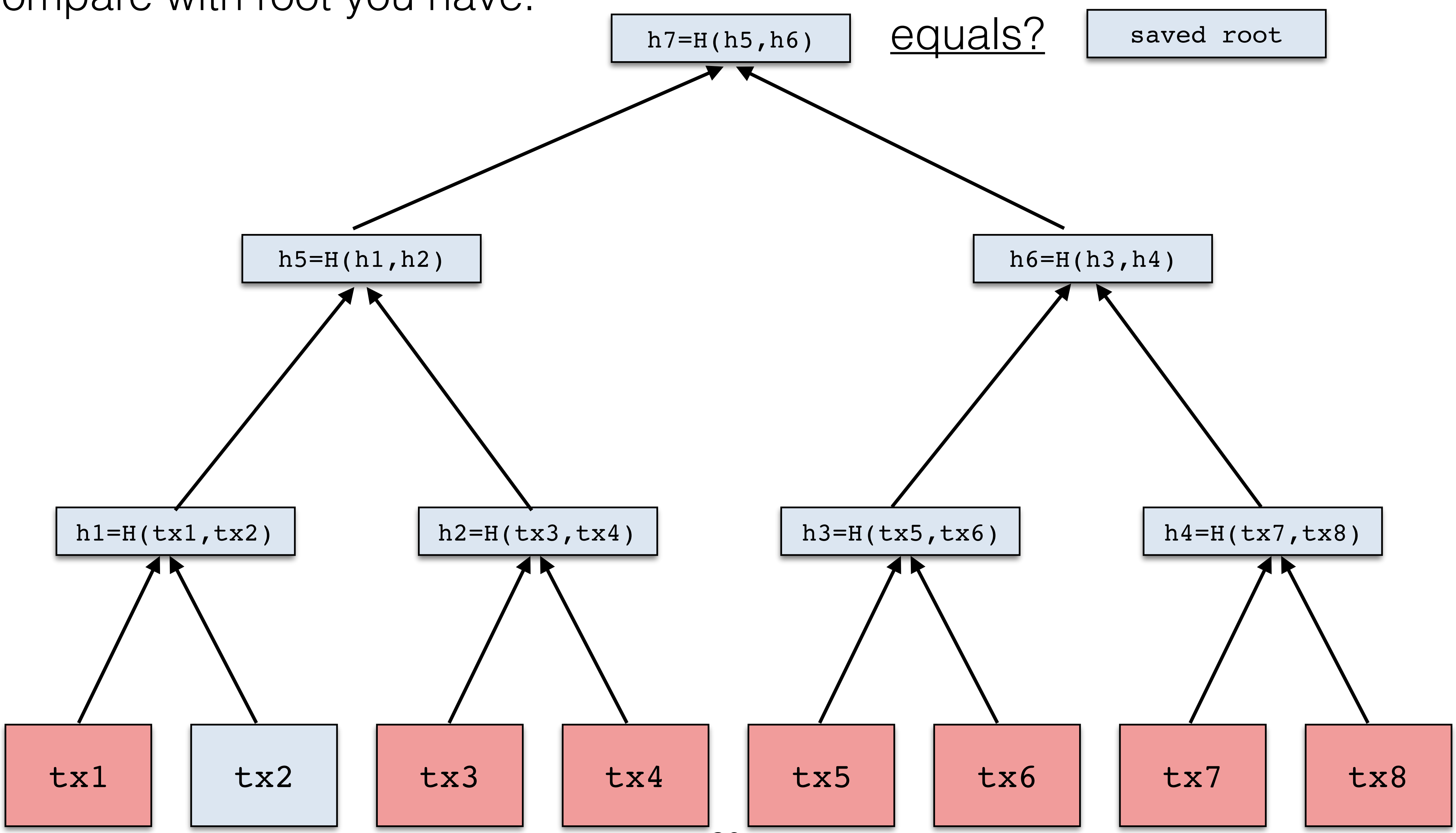
Ralph Merkle, 1979



- Bottom level: Large inputs (transactions)
- Every other level: 32-byte hashes

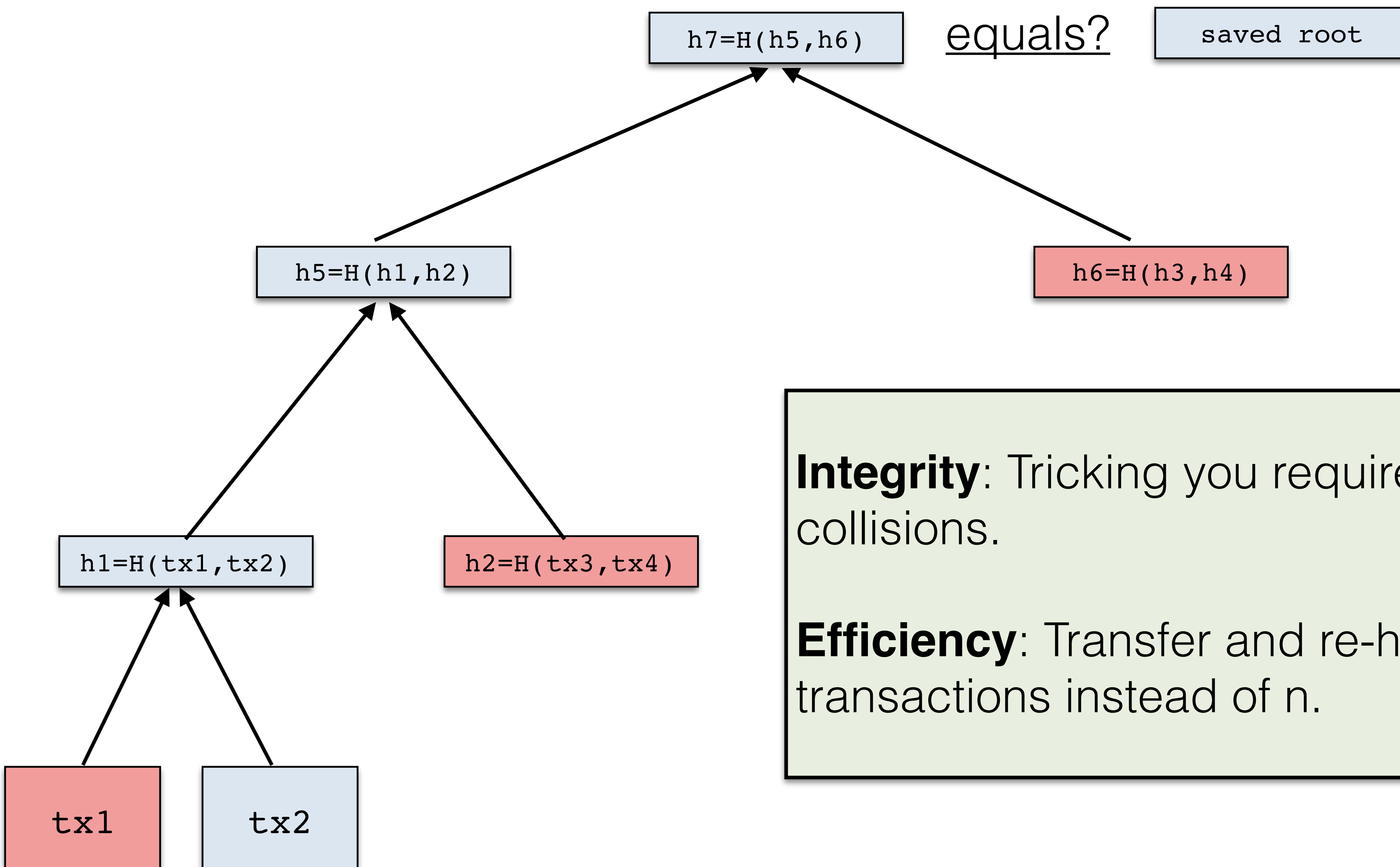
Merkle Trees and Proofs of Membership

- Suppose you have stored only the Merkle Root
- Now I claim: "transaction `tx2` was input as a leaf in the tree"
- Naive verification: I show all `tx1`, …, `tx8`, then you re-hash tree and compare with root you have.



Short Proofs of Membership ("Merkle Proofs")

- Smarter proof: I show you "adjacent nodes" on path from `tx2` to root
- You recompute only that path

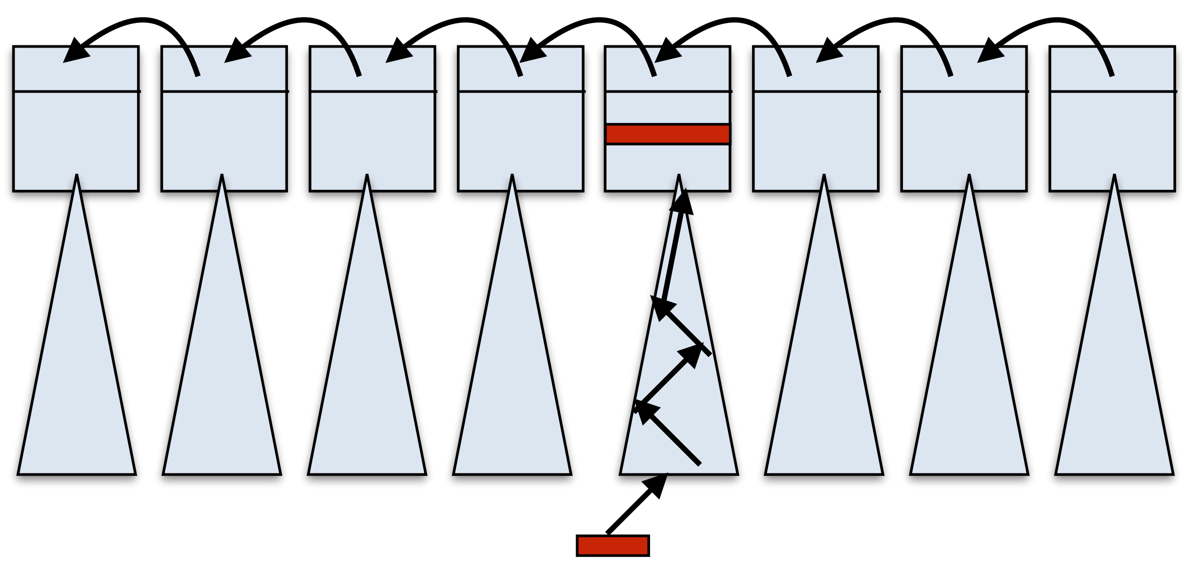


Integrity: Tricking you requires finding collisions.

Efficiency: Transfer and re-hash log(n) transactions instead of n.

Merkle Trees in Bitcoin

- Each block contains a Merkle root for its transactions



- SPV Nodes request Merkle Proofs instead of full blocks
  - Assuming hash is not broken, assurance is same
  - Average proof size: 12 kilobytes (= 1.2% of blocksize)
  - Less hashing for verification

All around win.

Lecture 3 Outline

1. Decentralized DCash via proofs-of-work
2. Some details about Bitcoin
   a. Transactions and scripts
   b. The Bitcoin network and types of nodes
3. **Mining**

Review Questions (homework, but do not submit)

1. Consider the POW-based blockchain sketched in this lecture.

   (a) What is the impact if the digital signature scheme is broken and forgeries are possible? What is the impact if two users accidentally generate the same public-key/secret-key pair for their identities?

   (b) What are the consequences if collisions can be found in the hash function? What are the consequences if a short-cut is found to solve the POW quickly?

2. Due to latency and imperfect connectivity in the peer-to-peer network, honest nodes on our blockchain will sometimes select different transactions for the blocks they add. Why is this not a fatal problem? What are the consequences when this happens, if any?

3. What would happen if our blockchain did not include transaction fees?

4. Our 51% attacker chose to point its new fork at the block just prior to its transaction. Why should it choose this block, rather than point earlier in the chain?

The End