Blockchains & Cryptocurrencies

Bitcoin

Instructor: Matthew Green
Johns Hopkins University - Fall 2024

Many slides based on NBFMG

Housekeeping

- Readings, NBFMG, Merkle-Damgard
- Last class did not record (I'm sorry)
  - I did put up the slides
  - And should be fixed
- A1 is out
  - On the course syllabus and a note in Piazza
  - **Remember to join the Piazza!**

News?

Future of Money | White Collar Crime | Data Privacy

Crypto ransom attack payments hit record $1 billion in 2023 - Chainalysis

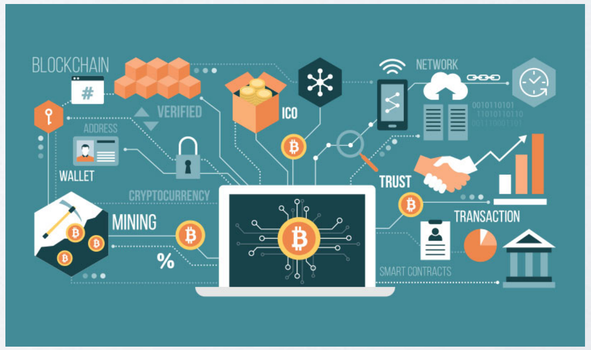
By **Medha Singh**

February 7, 2024 4:18 PM EST · Updated 7 months ago

Physical representations of the bitcoin cryptocurrency are seen in this illustration taken October 24, 2023. REUTERS/Dado Ruvic/Illustration/File Photo *Purchase Licensing Rights*

Today

- We're going to continue to talk about "consensus"
- What the heck is consensus, how do you accomplish it, what's the point?



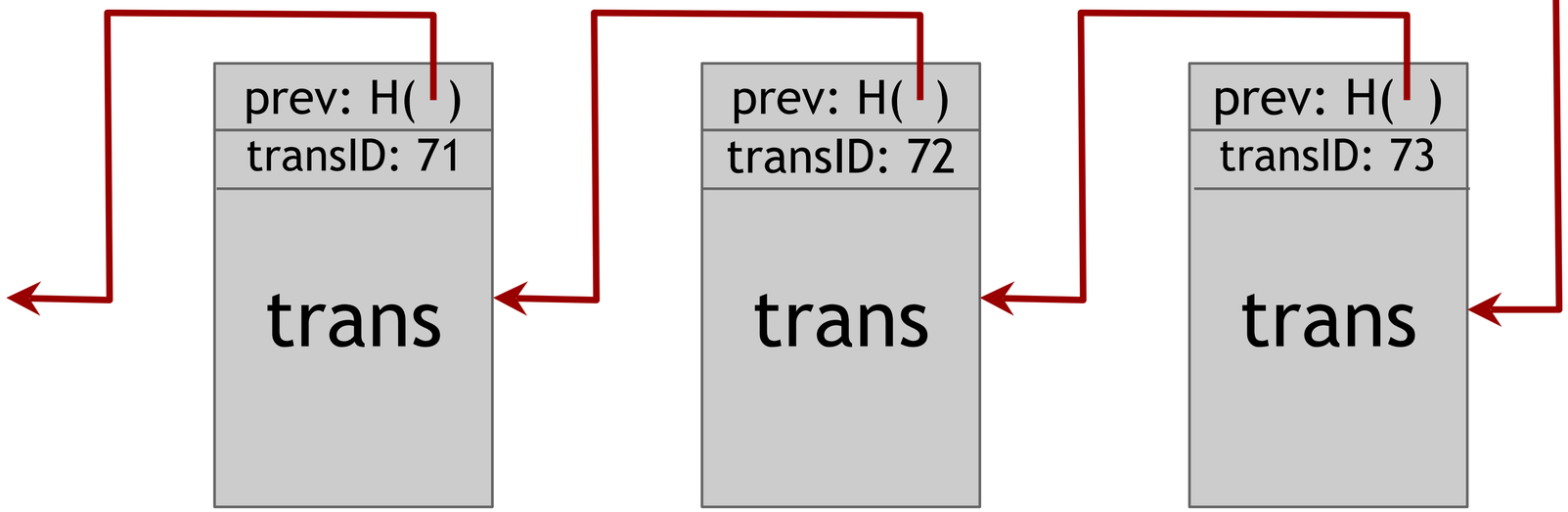
Review: ScroogeCoin

Scrooge publishes a history of all transactions in an “append-only” ledger

H( )

Sig

Implement the ledger using a block chain, signed by Scrooge



optimization: put multiple transactions in the same block

CreateCoins transaction creates new coins

Valid, because I said so.

transID: 73 type:CreateCoins		
coins created		
num	*value*	*recipient*
0	3.2	0x...
1	1.4	0x...
2	7.1	0x...

coinID 73(0)

coinID 73(1)

coinID 73(2)

signature

CreateCoins transaction creates new coins

Valid, because I said so.

transID: 73 type:CreateCoins		
coins created		
num	*value*	*recipient*
0	3.2	0x...
1	1.4	0x...
2	7.1	0x...

These are public keys!

coinID 73(0)

coinID 73(1)

coinID 73(2)

signature

PayCoins transaction consumes (and destroys) some coins,
and creates new coins of the same total value

transID: 73 type:PayCoins

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

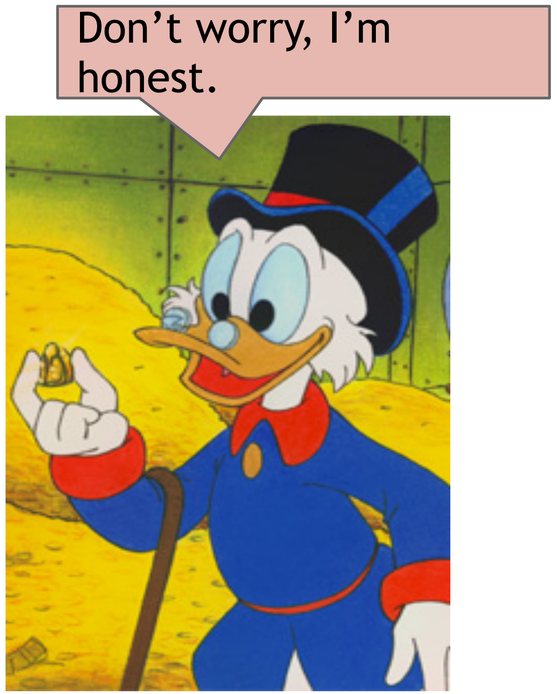
coins created

num	*value*	*recipient*
0	3.2	0x...
1	1.4	0x...
2	7.1	0x...

signatures

Valid if:
-- consumed coins valid,
-- not already consumed,
-- total value out = total value in, and
-- signed by owners of all consumed coins

One signature for each consumed coin



Crucial question:

Can we descroogify the currency, and operate without any central, trusted party?

Distributed consensus

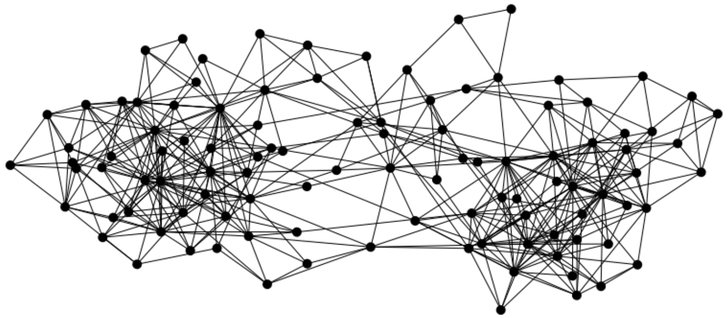
Bitcoin is a peer-to-peer system

This network is a fill/flood style P2P network: all nodes perform basic validation, then relay to their peers

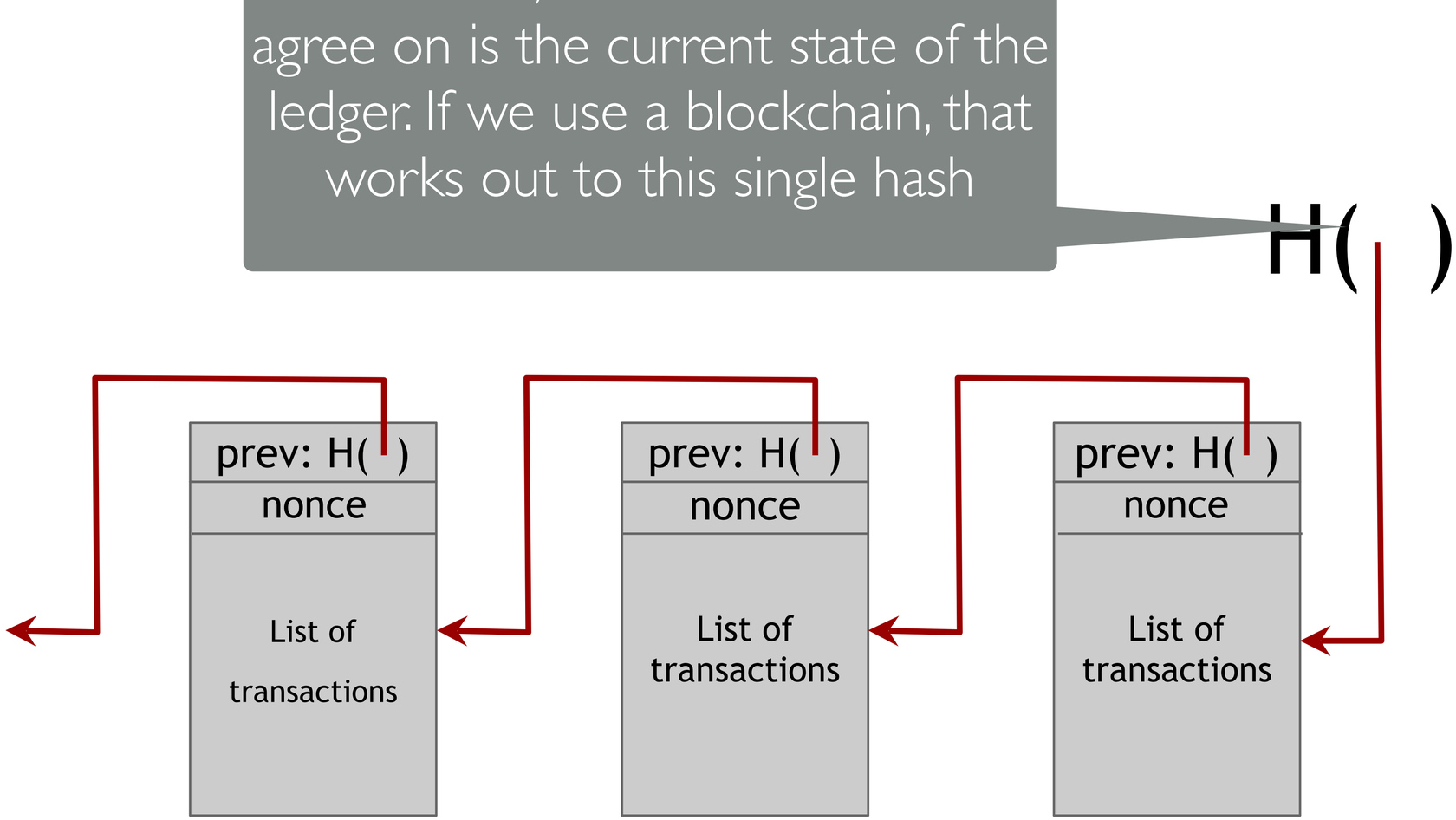
This introduces bootstrapping, spam and DoS problems, which are dealt with through “seeders” and “reputation” scores

signed by Alice

Pay to pk_{Bob} : H( )



A: In Bitcoin, the value we want to agree on is the current state of the ledger. If we use a blockchain, that works out to this single hash

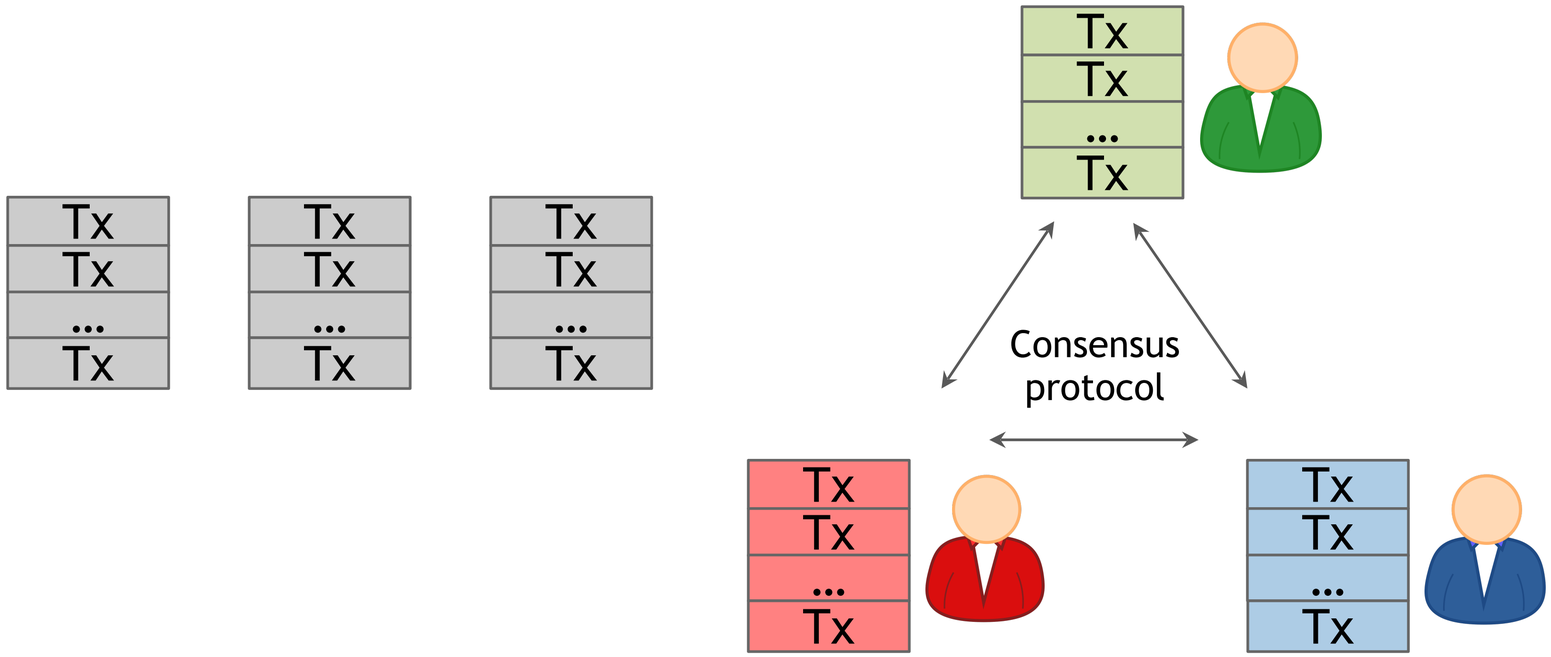


How consensus could work in Bitcoin

At any given time:

- All nodes have a sequence of blocks of transactions they've reached consensus on
- (Blocks are also distributed via p2p network)
- Each node has a set of outstanding transactions it's heard about

How consensus could work in Bitcoin



OK to select any valid block, even if proposed by only one node

Why consensus is hard

Nodes may crash
Nodes may be malicious

Network is imperfect

- Not all pairs of nodes connected
- Faults in network (“partitioning”)
- Latency

No notion of global time

Defining distributed consensus

The protocol terminates and all honest nodes decide on the same value (history)

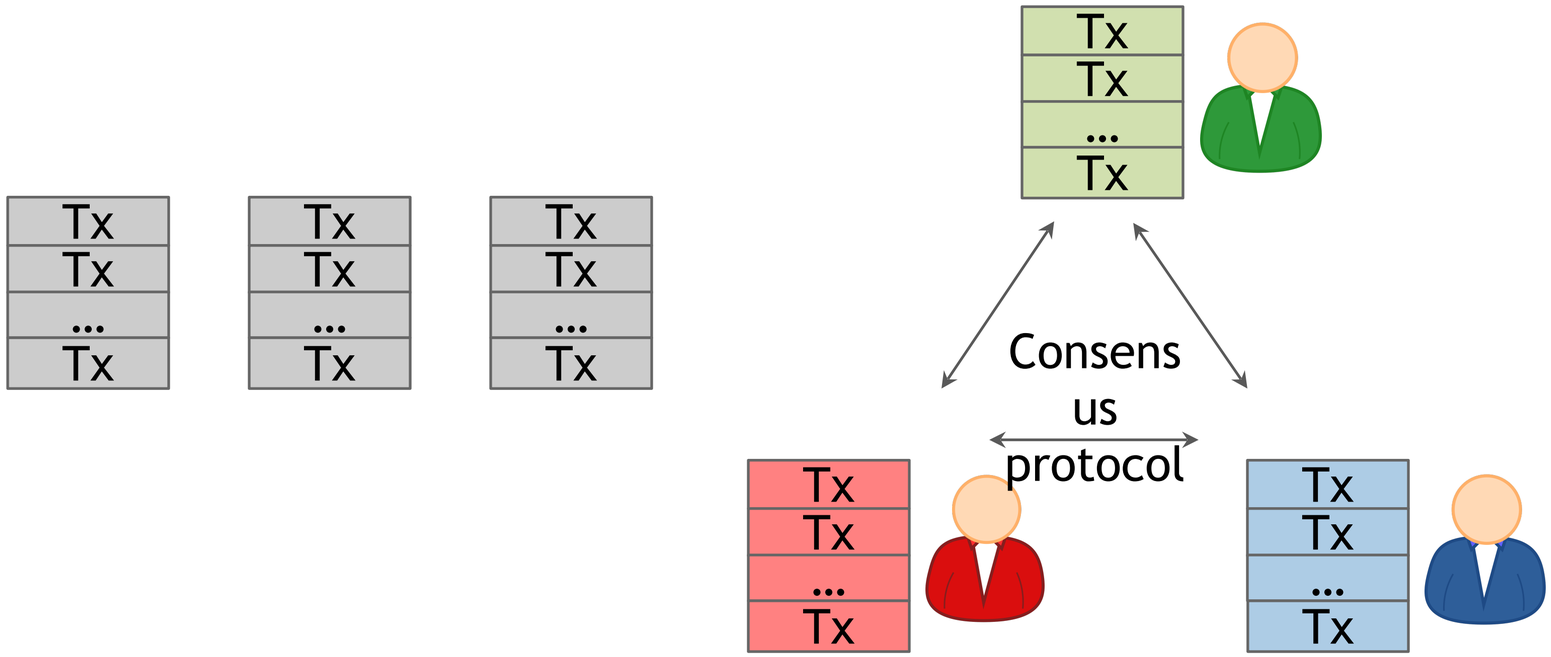
This value must have been proposed by some honest node

How consensus could work in Bitcoin

At any given time:

- All nodes have a sequence of blocks of transactions they've reached consensus on
- Each node has a set of outstanding transactions it's heard about

How consensus could work in Bitcoin



OK to select any valid block, even if proposed by only one node

Why consensus is hard

Nodes may crash
Nodes may be malicious

Network is imperfect

- Not all pairs of nodes connected
- Faults in network
- Latency

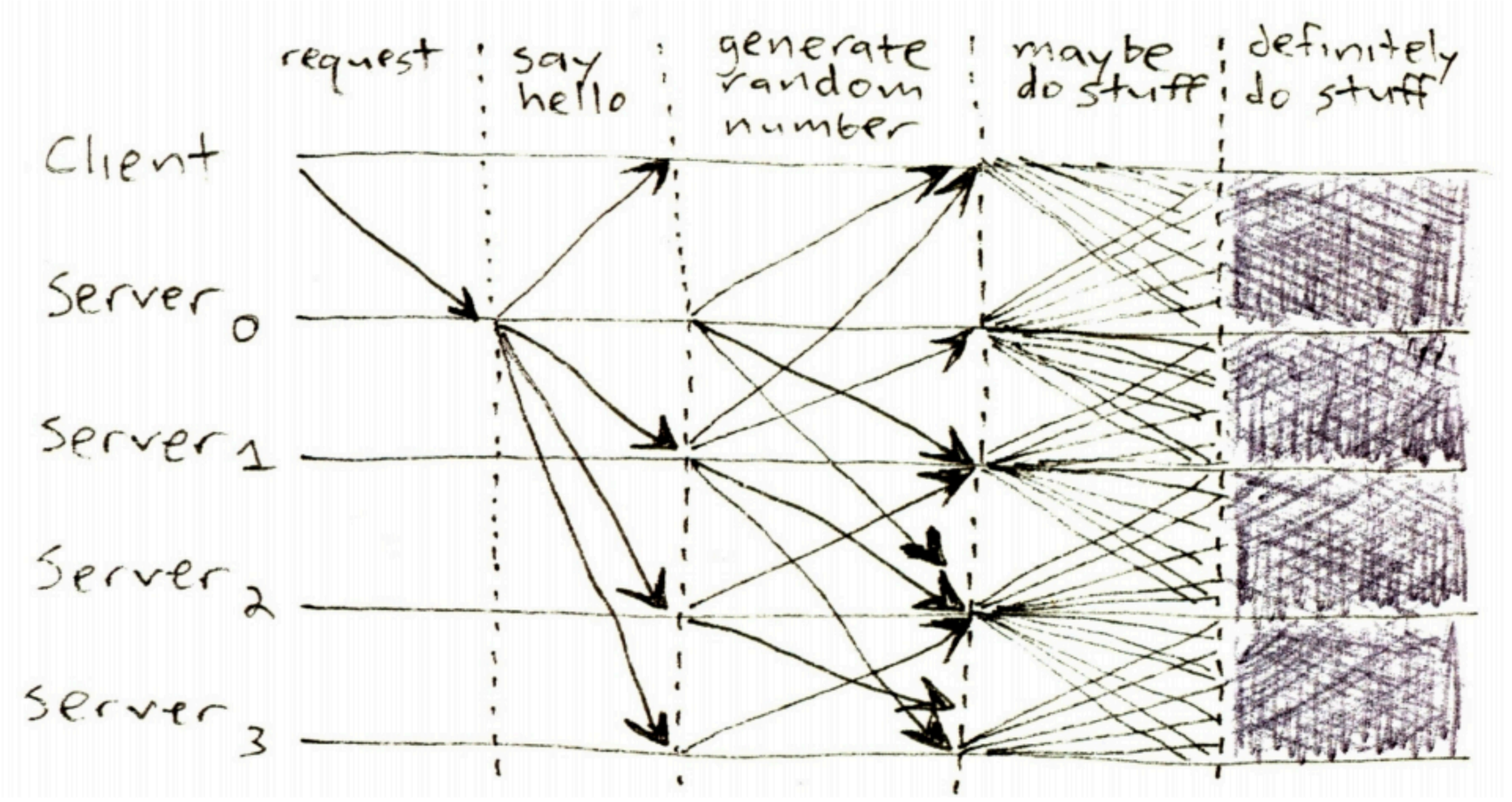
No notion of global time

Many impossibility results

- Impossible without 2/3 honest majority [Pease, Shostak, Lamport’80]
- Impossible with a single faulty node, in the fully asynchronous setting, with deterministic nodes [Fischer-Lynch-Paterson’85]

Why do these results matter?

- Because without node identities, an attacker could easily crash these networks by impersonating many nodes (“Sybil attack”)
- Because synchronicity is hard


request
say hello
generate random number
maybe do stuff
definitely do stuff
Client
Server 0
Server 1
Server 2
Server 3

Some positive results

Example: Paxos [Lamport]

Never produces inconsistent result, but can (rarely) get stuck

Understanding impossibility results

These results say more about the model than about the problem

The models were developed to study systems like distributed databases

Bitcoin consensus: theory & practice

- Bitcoin consensus: initially, seemed to work better in practice than in theory
- Theory has been steadily catching up to explain why Bitcoin consensus works [e.g., Garay-Kiayias-Leonardos'15,Pass-Shelat-Shi'17,Garay-Kiayias-Leonardos'17,…]
- Theory is important, can help predict unforeseen attacks

Some things Bitcoin does differently

Introduces incentives

- Possible only because it’s a currency!

Embraces randomness

- Does away with the notion of a specific end-point
- Consensus happens over long time scales — about 1 hour

Consensus without identity: the blockchain

Why identity?

Pragmatic: some protocols need node IDs

Security: assume less than 50% malicious

Why don’t Bitcoin nodes have identities?

Identity is hard in a P2P system — Sybil attack

Pseudonymity is a goal of Bitcoin

Weaker assumption: select random node

Analogy: lottery or raffle

When tracking & verifying identities is hard, we give people tokens, tickets, etc.

Now we can pick a random ID & select that node

Key idea: implicit consensus

In each round, random node is picked

This node proposes the next block in the chain

Other nodes implicitly accept/reject this block
- by either extending it
- or ignoring it and extending chain from earlier block

Every block contains hash of the block it extends

Consensus algorithm (simplified)

1. New transactions are broadcast to all nodes
2. Each node collects new transactions into a block
3. In each round a random node gets to broadcast its block
4. Other nodes accept the block only if all transactions in it are valid (unspent, valid signatures) and it builds on a chain they already accept
5. Nodes express their acceptance of the block by including its hash in the next block they create

So how do we pick a random node?

Resources & Consensus

- One computer can easily pretend to be many "nodes", so simple random voting != good
- But an observation: resources (e.g., hardware, storage, CPU, GPU, etc.) are much harder to fake
- Idea: make your probability of winning the vote proportional to your overall resources

Puzzles

- Early idea, proposed in the 90s: solve computational puzzles to prove CPU power
- Dwork & Naor (1992): use them to make spam email more expensive
- Back (1997): Hashcash, spam emails again
- Juels & Brainard (1999): use them to prevent DoS on web servers

Simple interactive puzzle

The web server accepts the GET iff S = H(Chal | Nonce) begins with "*D*" 0 bits

Simple interactive puzzle

Alternative formulation:

The web server accepts iff H(Chal | nonce) < *T*

Spam/Hashcash

The web server accepts iff H(Email | nonce) < *T*
and the email hasn't been seen before

Bitcoin PoW

The web server accepts iff H(Email | nonce) < *T*
and the email hasn't been seen before

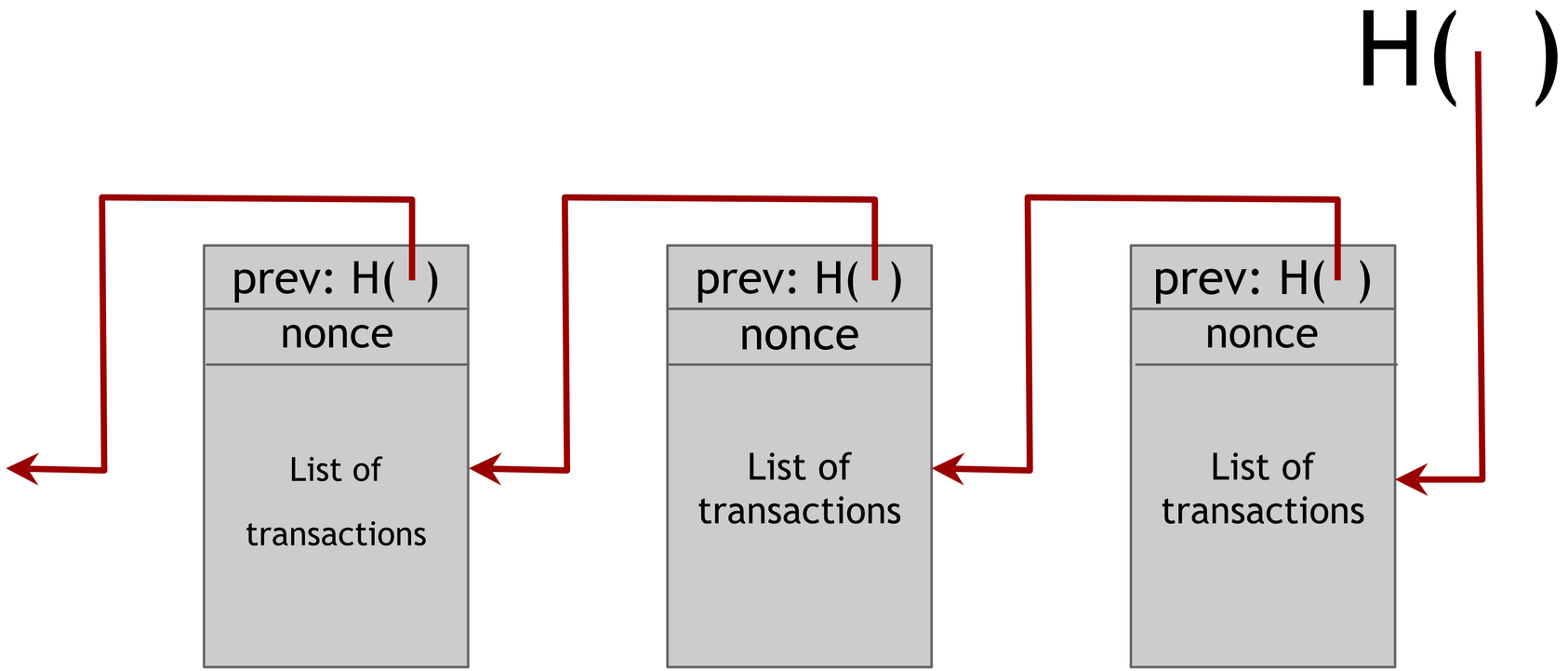
Key idea: implicit consensus

In each round, all nodes compete to solve a puzzle

The winner proposes the next block in the chain
and sends out their solution along with it

Other nodes implicitly accept/reject this block
- by either extending it
- or ignoring it and extending chain from earlier block

Every block contains hash of the block it extends


H( )
prev: H( )
nonce
List of transactions
prev: H( )
nonce
List of transactions
prev: H( )
nonce
List of transactions

What’s the puzzle?

The puzzle is simply the hash of the new block, which must be chained off the previous block

I.e., find a **nonce** such that H(block | nonce) < T for some T

The winner proposes the next block in the chain
and sends out their nonce

Other nodes implicitly accept/reject this block
- by either extending it
- or ignoring it and extending chain from earlier block

What happens if nodes ignore the block?

Where do we get *T*?

The value T is called the “block difficulty target”. It’s adjustable.

Ideas:

- Choose T once at the start, keep fixed
- Change T from time to time

What are the impacts of these choices?

Selecting T (Bitcoin)

Bitcoin’s difficulty changes every 2016 blocks

Goals:

- Bitcoin block time should average 10 minutes
- Everyone in the network agrees on *T*
- *Hence must be a function of chain data*

This brings back a notion of time

Selecting T (Bitcoin)

Every block contains a (packed) encoding of T (target) which corresponds to ***“difficulty” (d)***

d = ((2^{16} - 1) * 2^{8*26}) / T <- relationship between d, T

Goals:

- Bitcoin block time should average 10 minutes
- 2016 blocks * 10 minutes == 2 weeks
- Everyone in the network agrees on *T*
- *Hence must be a function of chain data*

This brings back a notion of time

Bitcoin blocks include timestamps

Every block contains a timestamp that alleges when it was found

Remember that nodes can be adversarial! So some timestamps may be lies! This requires heuristics:

- Each timestamp must be no sooner than the median of the past 11 blocks
- Honest nodes must reject timestamps that are > 2hrs in the future

This brings back a notion of time

What if there’s a collision?

Sometimes two separate nodes find a valid solution simultaneously

This can result in network partition

What if there’s a collision?

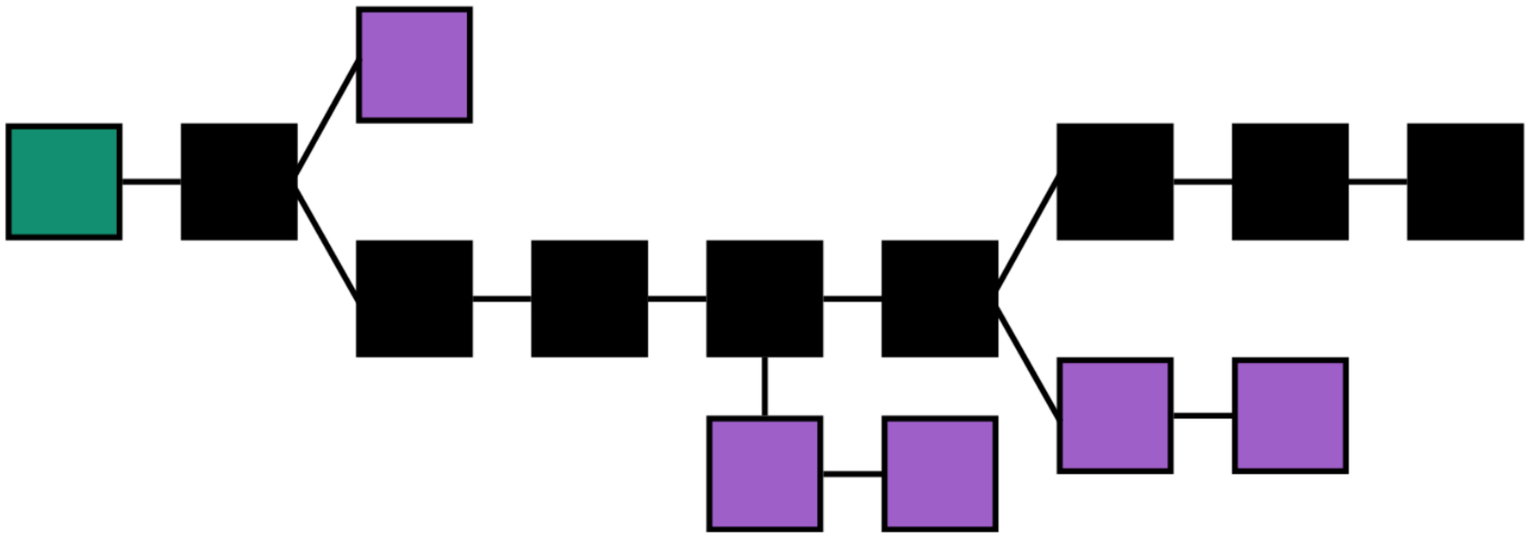


Image CC-BY-3 Theymos taken from the Bitcoin wiki

“Longest chain rule”

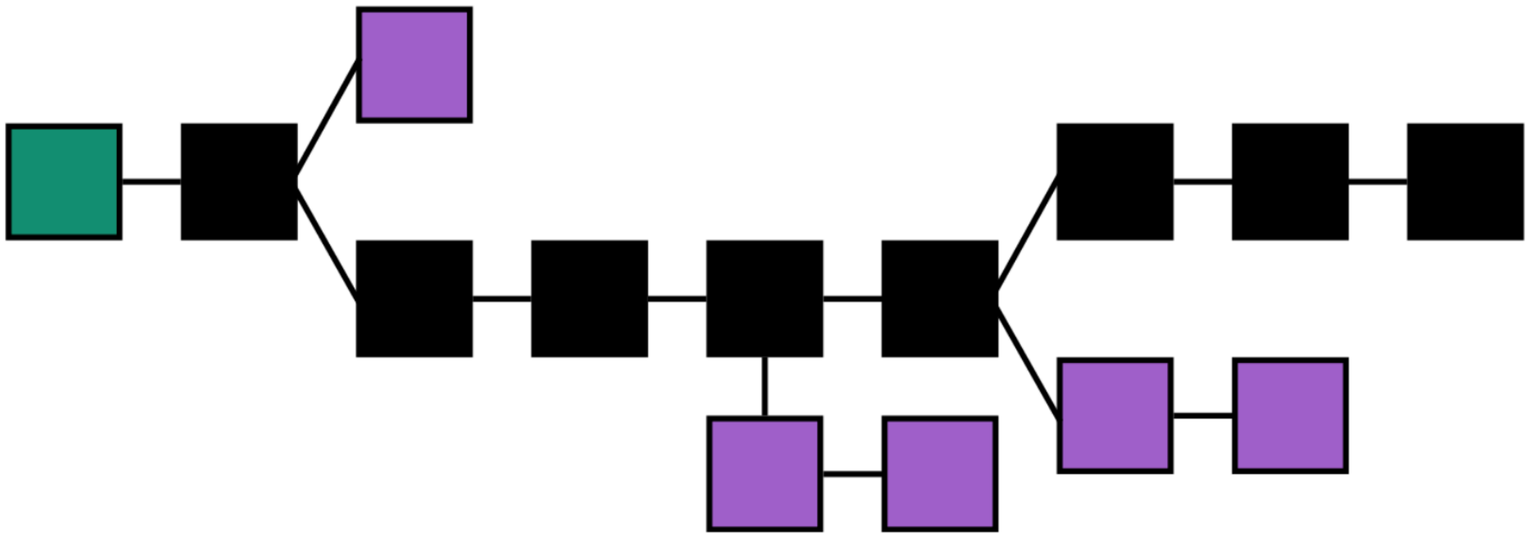


Image CC-BY-3 Theymos taken from the Bitcoin wiki

“Longest chain rule”

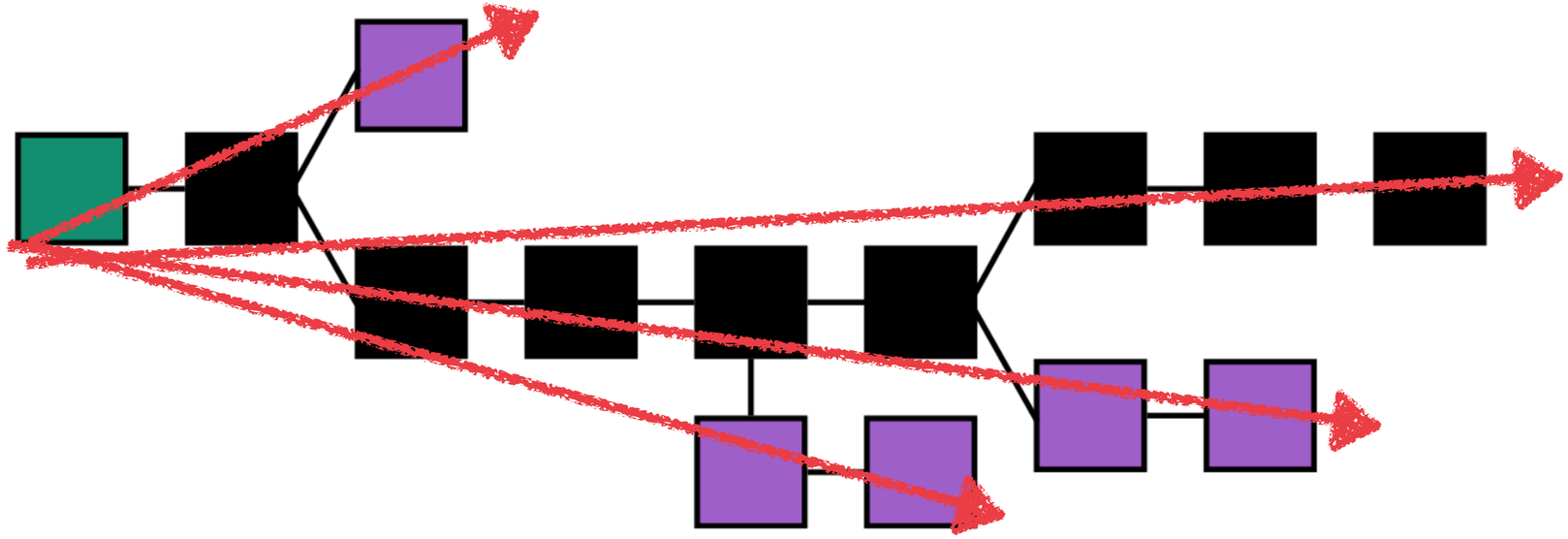


Image CC-BY-3 Theymos taken from the Bitcoin wiki

This is good and bad

Good: if we experience a “chain fork” and the network is connected (I.e., not totally partitioned), then eventually we will learn about both forks

Good: if the “hash power” behind the two chains is unequal, we will probably end up with one chain getting longer

Even if the hash power is equal, the inherent randomness of the puzzle (PoW) will likely cause an advantage

As one chain grows longer, other nodes will adopt it, and start adding to it