Cryptocurrency

Ellis Michael
h/t Tom Anderson

Decentralized Control

- PBFT and similar protocols require public-key infrastructure and that the servers know who the other servers are.
- This must be setup by some central authority for the protocol to run.
- Otherwise, these protocols are susceptible to **Sybil attacks.**
- What if you want a decentralized system?

Two Classes of Solutions

Proof of Work

- Rate of transaction commitment is limited by cryptographically hard problem.
- Nodes called miners solve these problems to commit transactions.
- Assumes that a majority of the CPU power is controlled by honest* nodes.
- Miners are rewarded with transaction fees and mining rewards.

Proof of Stake

- Transactions are committed with votes weighted by the amount of stake voters have in the system.
- Assumes that a 2/3rds of the money is controlled by honest* nodes.
- Voters sometimes rewarded for taking part in the protocol. (But they also have stake in the system.)

Bitcoin

- Bitcoin is a proof-of-work cryptocurrency network, started in 2009.
- Goal: electronic money without the need for trust.
- Relies on cryptography for authentication, proof-of-work for transaction ordering.

Pros and Cons of Cash

+ Portability
+ Can't spend twice
+ Cannot repudiate after payment
+ No need for trusted 3rd party (for individual transactions)
+ Anonymous/fungible (except for serial numbers)
- Doesn't work online
- Easy to steal (in moderate amounts)

+/- Hard for government to monitor/tax/control

+/- Government can print more as economy expands

Pros and Cons of Online Payments (Credit Cards, Paypal, etc.)

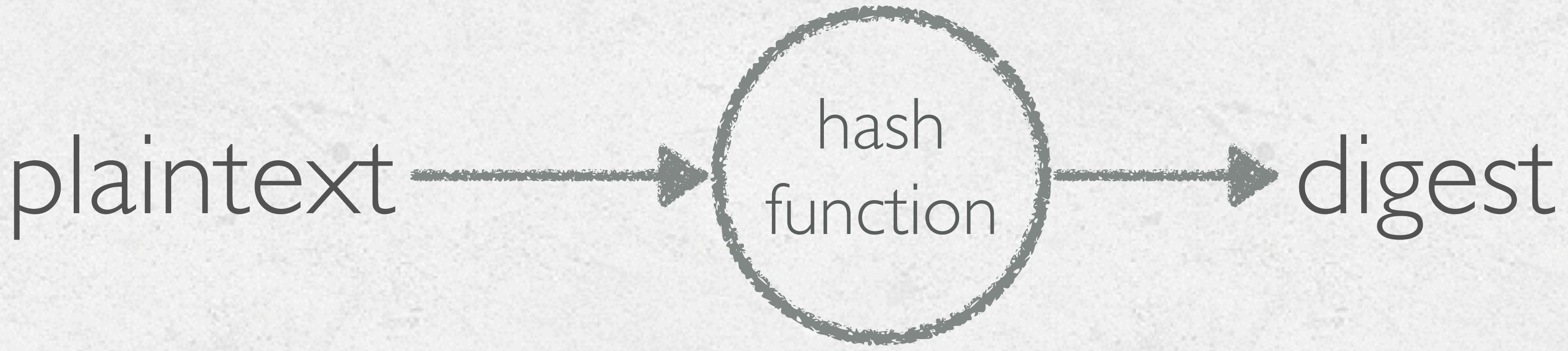
+ Works online
+ Somewhat hard to steal
- Requires trusted third party
- Purchases are tracked
- Can prohibit some transactions

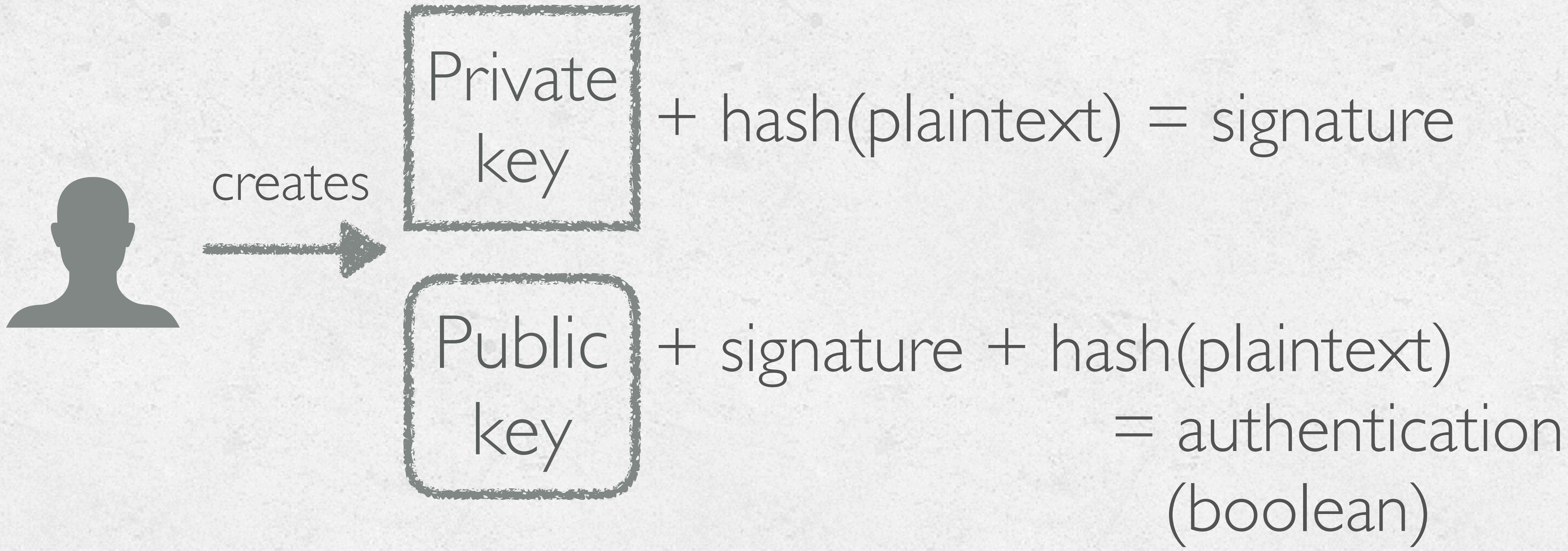
+/- Can repudiate (to some extent)

+/- Easy for government to monitor/tax/control

Cryptographic Assumptions

- Cryptographic hash functions (e.g. SHA-256).
- Public key cryptography that allows digital signatures (e.g., RSA). (Not necessary for nodes to know each others' public keys *a priori*.)





Bitcoin Transactions

- Payment is a signed, publicly visible transaction between public/private key pairs.
- Transactions have (potentially multiple) inputs and outputs.
- Transaction inputs are **other transactions**.
- Transaction outputs are public keys (recipients).

"Ellis takes the 42 bitcoins he got from transaction abc123 and the 8 BTC from transaction def456 and pays Arvind's public key 45 BTC. Ellis pays himself the remaining 5 BTC."

[signed with Ellis's private key]

Transaction

Strawman Proposal

- Ellis just signs the transaction and gives it to Arvind.
- What could go wrong?
  - Arvind couldn't have impersonated Ellis. He doesn't have Ellis's private key.
  - What if the sender already spent the transaction in question? This is called **double-spending**.
  - Where does money actually come from?

Trusted Third Parties (Not a Strawman)

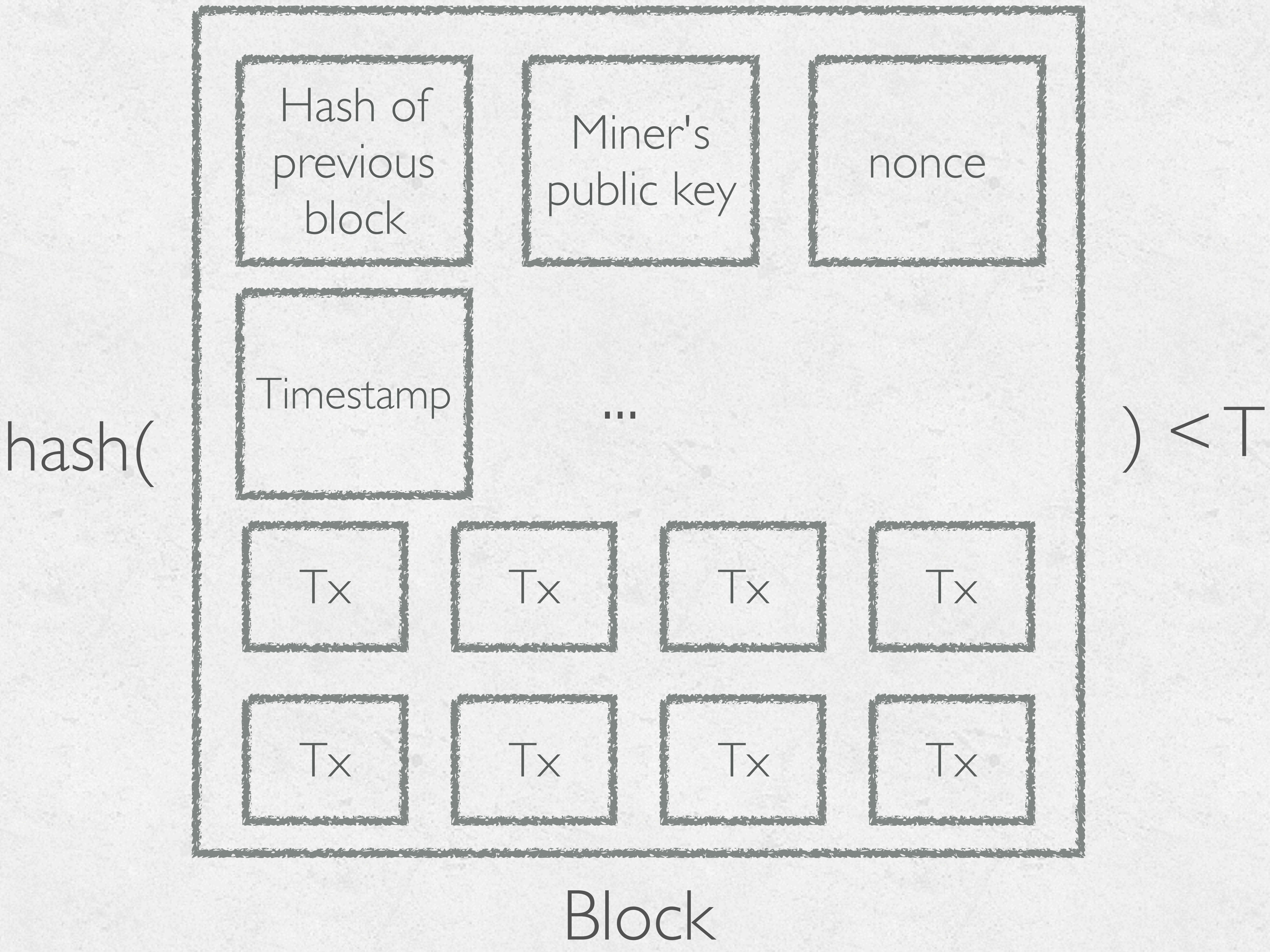
- The sender could send the transaction to a trusted third party (or system).
- As long as the transaction is valid (i.e., the input transactions weren't already spent), accepts the transaction and puts it in a log. The log is made publicly visible (and can be replicated by any number of passive listeners).
- The recipients of a transaction wait until they see the transaction in the log. Once it's there, it's been committed.

Managing the Public Log

- We need the log to stay consistent (i.e., that transactions stay in the same order in the log).
- We could use Paxos, but what if the replicas aren't trusted?
- PBFT still requires trusting 2f+1 replicas.

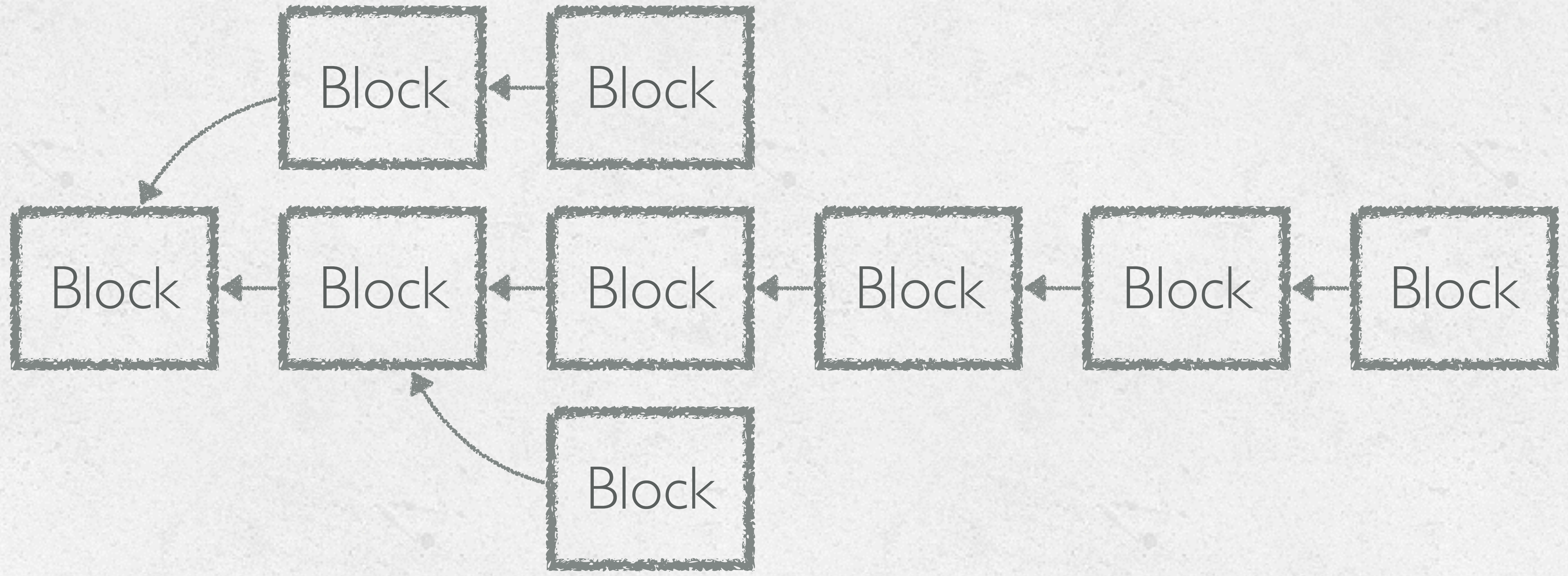
Bitcoin Mining

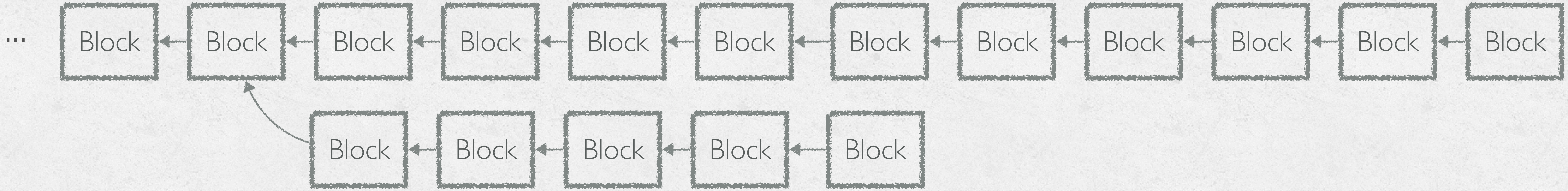
- Bitcoin commits transactions by having servers called **miners** solve a cryptographic puzzle.
- Transactions are committed in blocks.
- Miners try to find a **nonce** such that the hash of the entire block is less than some threshold.
- Finding such a nonce is difficult. But miner's get compensated in the form of **mining rewards** (bitcoin from nowhere) and **transaction fees** (bitcoin from the transaction senders).



Finding a Stable Order

- Each block has a single pointer to the previous block (except for the initial block). These blocks then form a DAG.
- Honest miners work off of the **longest chain**. If they see two chains of equal length, they work off the one they saw first. (What about greedy miners?)
- Only transactions with unspent inputs are valid.
- Normally, clients wait for transactions to be 6 blocks deep (i.e., that it's in a chain 6 blocks longer than any chain without the transaction) before considering it **confirmed**.
- As long as honest miners control >50% of the hashing power, the longest chain can't be overrun. Confirmed transactions won't be undone by a double-spend.




...
Block
Block
Block
Block
Block
Block
Block
Block
Block
Block
Block
Block
Block
Block
Block
Block
Block

Network Protocol

- Bitcoin uses a gossip protocol to communicate new blocks and transaction requests.
- Each peer is connected to a set of other peers.
- Peer list is bootstrapped usually using DNS by asking for a hostname that points to known nodes.

Hash Puzzle Difficulty

- The threshold for the mining puzzle is by the **difficulty**, a 256 bit number.

- If the difficulty is 2^{254}, there's a 1/2 chance for any given nonce. 2^{253} gives a 1/4 chance, etc.

- The difficulty is adjusted every 2016 blocks to keep the average throughput at ~1 block/10 mins.

- The average time to confirm a transaction is 1 hour.

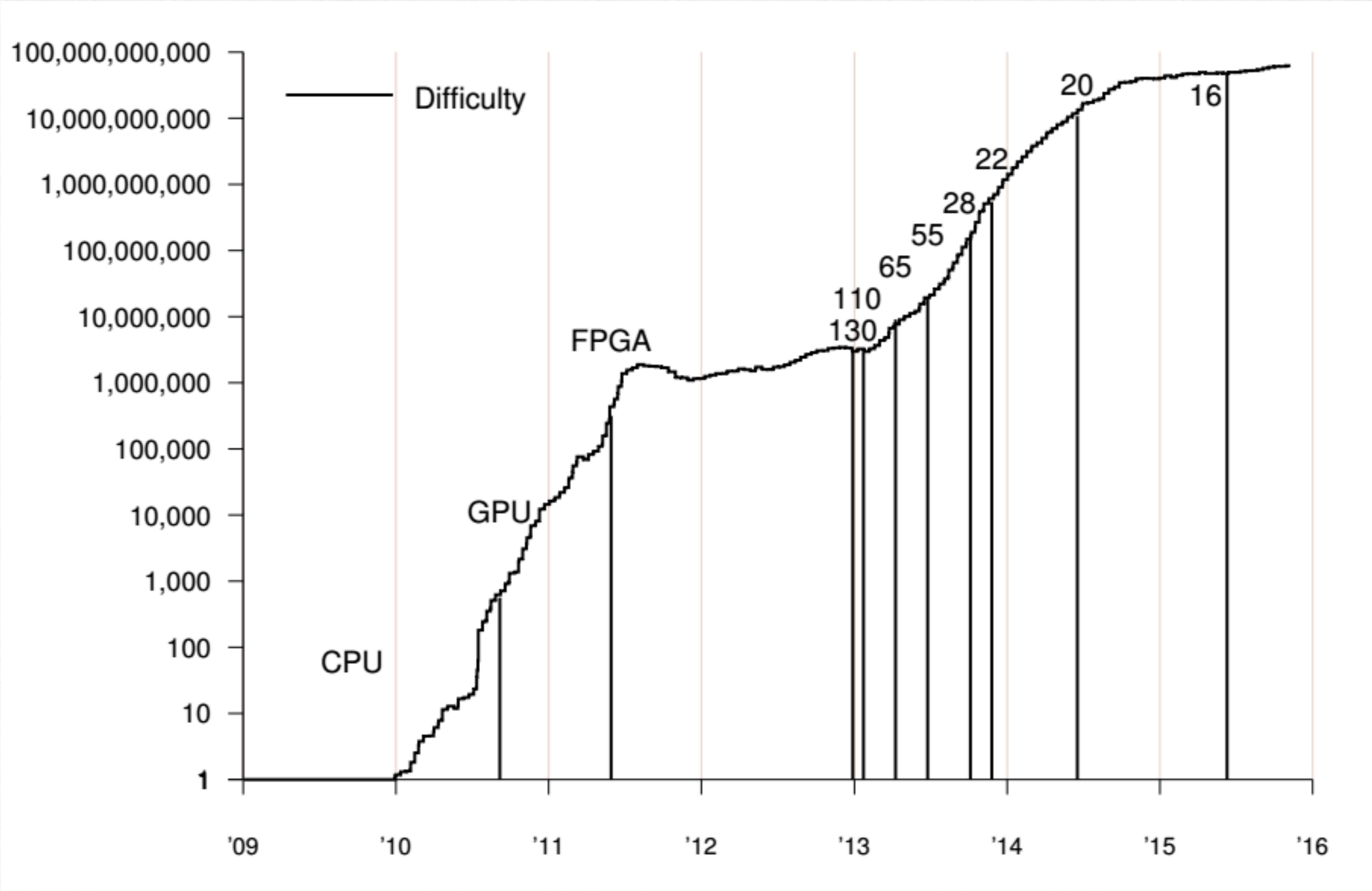
Mining Rewards

- Every time a block is "mined," the miner gets a reward.
- This reward started at 50 BTC and is halved every 210,000 blocks (approximately every 4 years).
- Since bitcoins aren't infinitely divisible, the reward will go to 0 at some point. There will only every be a maximum of 21M BTC.
- Currently, about 85% of all bitcoins have been mined.

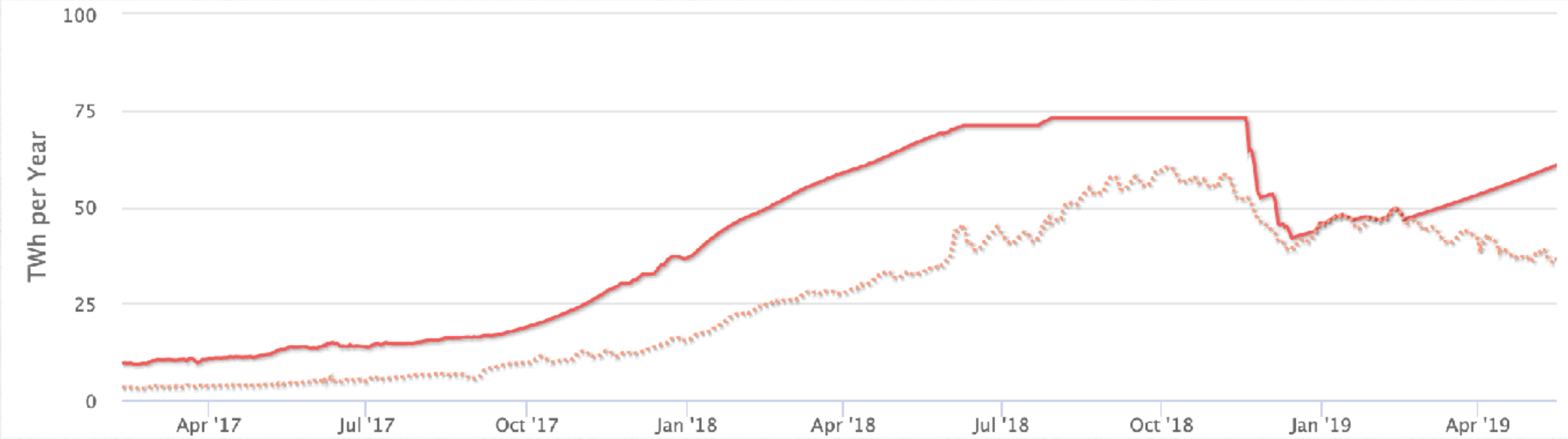
Transaction Fees

- Transaction senders also pay a fee that is claimed by the winning miner.
- The higher the fee, the more incentivized miners are to commit that transaction.
- Once all bitcoins are mined, this will be the only mining incentive.
- Currently, transaction fees are averaging about 0.00050 BTC (=$4 at current prices).

Bitcoin Hardware Progression



Hashing - Like Machine Learning But Less Useful



- Even with specialized hardware, hashing is energy-intensive.
- Currently, the overall hashrate is 55 EH (exahash)/s.
- The entirety of the bitcoin mining network consumes the same amount of energy as Switzerland (!).

Bitcoin Throughput

- Currently, there are an average of 2,500 transactions in a 2MB bitcoin block.
- The network mines a block once every 10 minutes on average.
- This gives us ~4 transactions/s.

What Did This Get Us?

- Privacy?
  - Well, not really. Your name isn't published, but the flow of money from one transaction to another is public.
- Non-repudiation?
  - Why couldn't a bank guarantee this?
- No trusted authority?
  - Great, now drug dealers and human traffickers get financial infrastructure, too!
- No centralized monetary policy?
  - You **like** deflation?

Does this look like a currency?

Why are people putting their money in this?

Other Proof-of-Work Systems

Bitcoin is by no means the only popular proof-of-work based system.

- **Zerocoin** provides better anonymity (which makes it even *better* for money laundering?)

- **Etherium** allows scripting.

- **Ripple** tries to maintain a stable price.

- ...and **many** others...

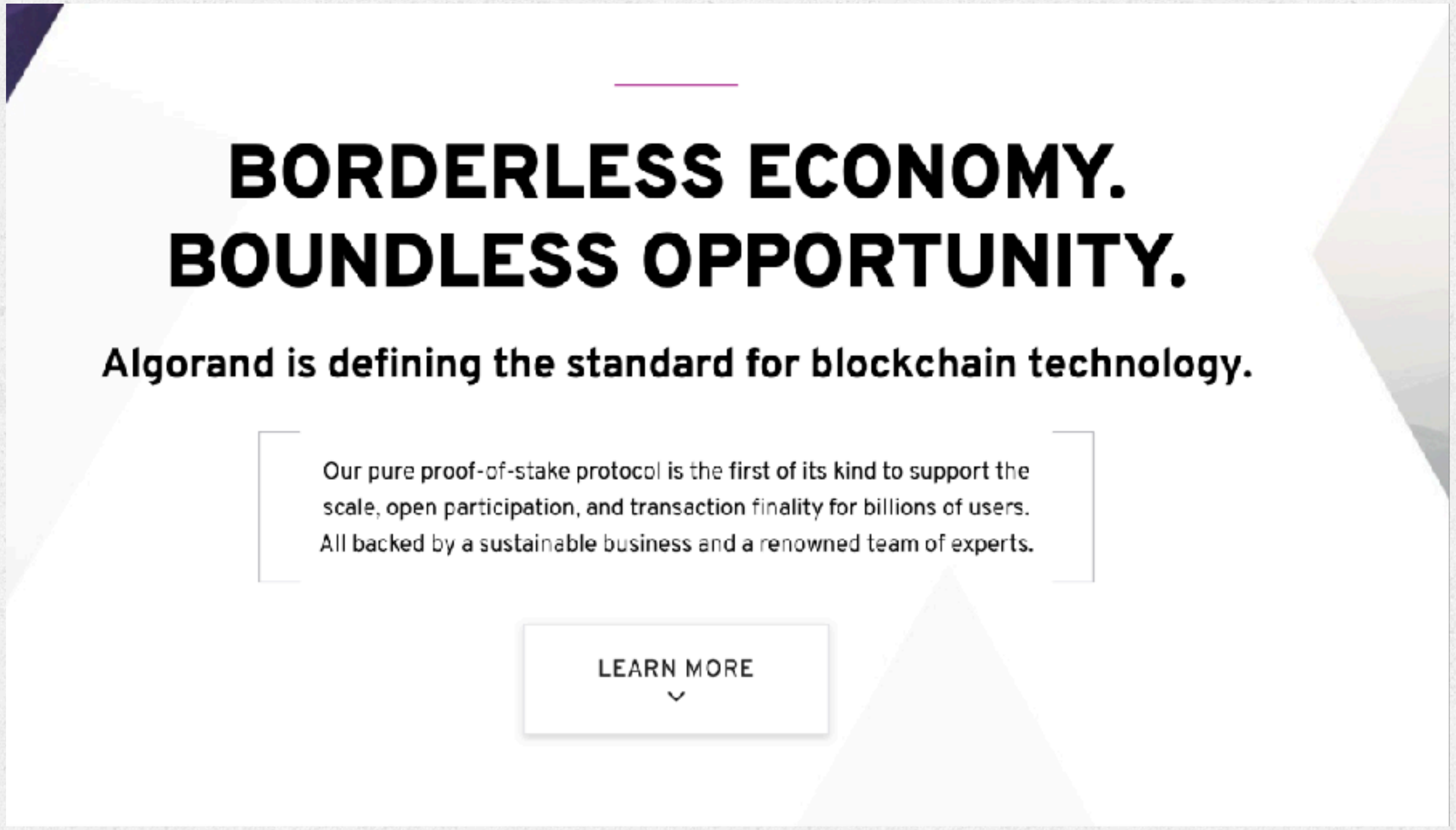
Bitcoin Discussion Questions

- Where does value of a Bitcoin come from?
- Is the energy consumption of Bitcoin worth it?
- How valuable is decentralization, really?
- Is Bitcoin useful as a currency? For small transactions?
- How long will SHA-256 last?
- How do we make changes to the protocol?
- Is Bitcoin actually anonymous?
- Is Bitcoin ethical given its benefits for ransomware, money laundering, etc.?
- Why do wallets and private exchanges exist? Don't they defeat the purpose?
- What if miners are rational (greedy) instead of honest?
- What implications does the non-reversibility of Bitcoin have?

Proof-of-Stake

Algorand

- Created in 2017.
- Uses proof-of-stake instead of proof-of-work (but not the first).
- Apparently now one of approx. 300 billion blockchain startups.



Main Ideas

- **Weight users** by how much money they hold in their account.
- Use Byzantine agreement, but rather than doing Byzantine agreement over all users, use a **randomly selected committee**.
- Choose the committees based on **cryptographic sortion**. Uses a **verifiable random functions** on publicly available data and secret information held by the participants so that the adversary can't target committee members ahead of time.
- Each committee is only used for a **single step**. As soon as a committee member reveals their decision, they're no longer relevant and can't be targeted.

Goal

- The transaction structure is the same as Bitcoin. We just need to agree on an ordering of the transactions (blocks).
- We want safety (linearizability) with high probability.
- We assume that at least **2/3 of the money is held by honest users** running bug-free code.
- The system should be reasonably performant and scalable.

Verifiable Random Functions

A VRF is like a modified hash function.

VRF(x, sk) = $\boldsymbol{h}$, π

- x is an input string
- sk is a secret key
- $\boldsymbol{h}$ is a hash
- π is a proof that anyone knowing the public key can use to verify the results (doesn't reveal secret key).

Cryptographic Sortion

Bottom line: the output of the VRF determines whether (and how many times) a user is chosen for a particular role.

- **τ**: number of expected users for a given role.
- ***w***: amount of currency that user controls.
- ***W***: amount of total currency.
- ***seed***: each round's seed proposed along with block using VRF of previous seeds.

procedure Sortition($sk, seed, \tau, role, w, W$):

$\langle hash, \pi \rangle \leftarrow \mathrm{VRF}_{sk}(seed||role)$

$p \leftarrow \frac{\tau}{W}$

$j \leftarrow 0$

while $\frac{hash}{2^{hashlen}} \notin \left[\sum_{k=0}^{j} B(k; w, p), \sum_{k=0}^{j+1} B(k; w, p)\right)$ **do**

  j++

return $\langle hash, \pi, j \rangle$

Algorithm 1: The cryptographic sortition algorithm.

procedure VerifySort($pk, hash, \pi, seed, \tau, role, w, W$):

if $\neg VerifyVRF_{pk}(hash, \pi, seed||role)$ **then return** 0;

$p \leftarrow \frac{\tau}{W}$

$j \leftarrow 0$

while $\frac{hash}{2^{hashlen}} \notin \left[\sum_{k=0}^{j} B(k; w, p), \sum_{k=0}^{j+1} B(k; w, p)\right)$ **do**

  j++

return j

Algorithm 2: Pseudocode for verifying sortition of a user with public key pk.

Round Structure

- Each round is an opportunity to commit some block.
- First, proposers are chosen and propose blocks.
- Next, the system runs BA*, their main agreement protocol, to choose a block from among the proposed ones.
- BA* proceeds in two phases. First, it reduces the problem to choosing between two options. Then, it runs BinaryBA* to choose between those.
- BA* can either reach TENTATIVE or FINAL agreement. A block is committed if it is FINAL or if one of the block's successors is FINAL.

BinaryBA*

- Essentially, a modified version of the Ben-Or randomized consensus algorithm.
- It uses a shared random coin to reach consensus faster.
- Random coin is biased using the hashes of messages from the previous step. Even if the adversary controls the network, it can't delay consensus forever (since it can't influence the hash values).
- With strong synchrony, BinaryBA* finishes quickly with high probability.

```
procedure CommonCoin(ctx, round, step, τ):
minhash ← 2^hashlen
for m ∈ incomingMsgs[round, step] do
    ⟨votes, value, sorthash⟩ ← ProcessMsg(ctx, τ, m)
    for 1 ≤ j < votes do
        h ← H(sorthash||j)
        if h < minhash then minhash ← h;
return minhash mod 2
```

Algorithm 9: Computing a coin common to all users.

```
procedure BinaryBA⋆(ctx, round, block_hash):
step ← 1
r ← block_hash
empty_hash ← H(Empty(round, H(ctx.last_block)))
while step < MaxSteps do
    CommitteeVote(ctx, round, step, τ_STEP, r)
    r ← CountVotes(ctx, round, step, T_STEP, τ_STEP, λ_STEP)
    if r = TIMEOUT then
        r ← block_hash
    else if r ≠ empty_hash then
        for step < s′ ≤ step + 3 do
            CommitteeVote(ctx, round, s′, τ_STEP, r)
        if step = 1 then
            CommitteeVote(ctx, round, FINAL, τ_FINAL, r)
        return r
    step++

    CommitteeVote(ctx, round, step, τ_STEP, r)
    r ← CountVotes(ctx, round, step, T_STEP, τ_STEP, λ_STEP)
    if r = TIMEOUT then
        r ← empty_hash
    else if r = empty_hash then
        for step < s′ ≤ step + 3 do
            CommitteeVote(ctx, round, s′, τ_STEP, r)
        return r
    step++

    CommitteeVote(ctx, round, step, τ_STEP, r)
    r ← CountVotes(ctx, round, step, T_STEP, τ_STEP, λ_STEP)
    if r = TIMEOUT then
        if CommonCoin(ctx, round, step, τ_STEP) = 0 then
            r ← block_hash
        else
            r ← empty_hash
    step++
// No consensus after MaxSteps; assume network
// problem, and rely on §8.2 to recover liveness.
HangForever()
```

Algorithm 8: BinaryBA⋆ executes until consensus is reached on either *block_hash* or *empty_hash*.

Algorand Takeaways

- Algorand doesn't utilize proof-of-work and instead weights users based on how much money they have in the system.

- Algorand is more communication efficient since it is committee based.

- However, it is not clear what incentives users have to participate in the protocol (their stake in the system notwithstanding).

- Algorand requires money holders to be online and broadcasting their address to the world.