Ethereum in 25 Minutes

A guided tour through the depths of the Merkle underworld

Why ethereum?

- Seeming public consensus circa 2013: blockchains are useful for… stuff
- Not just money! Asset issuance, crowdfunding, domain registration, title registration, gambling, prediction markets, internet of things, voting, hundreds of applications!

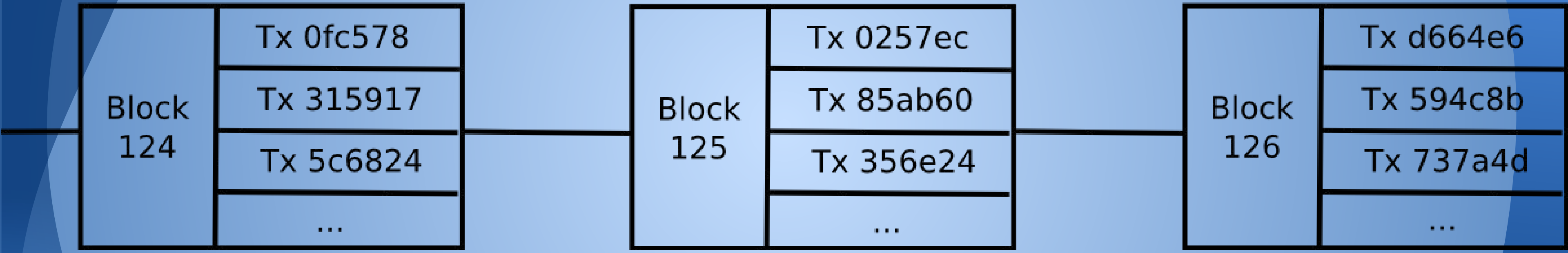
Problem: most existing blockchain protocols were designed like this:

Or, at best, like this:

So… why not make a protocol that works like this?*

*Special thanks to Microsoft for sponsoring this conference

The concept



It’s a blockchain… yawn

With… a few tiny additions

- A built-in programming language
- Two types of accounts
  - User accounts (controlled by private keys)
  - Contracts (controlled by code)
- Anyone can create an application with any rules by defining it as a contract

DNS: The Hello World of Ethereum

```
data domains[](owner, ip)

def register(addr):
    if not self.domains[addr].owner:
        self.domains[addr].owner = msg.sender

def set_ip(addr, ip):
    if self.domains[addr].owner == msg.sender:
        self.domains[addr].ip = ip
```

State vs History

- State = “current” information
  - Account balances
  - Nonces
  - Contract code and contract storage
- History = things that happened
  - Transactions
  - Receipts
- Currently, all “full” nodes store state, some store history and some do not store history

State

- State consists of key value mapping addresses to account objects
- Every account object contains 4 pieces of data:
  - Nonce
  - Balance
  - Code hash (code = empty string for private key-controlled accounts)
  - Storage trie root

Code execution

- Every transaction specifies a TO address it sends to (unless it's creating a contract)
- The TO address's code runs
- Code can:
    - Send ETH to other contracts
    - Read/write storage
    - Call (ie. start execution in) other contracts

Code execution

- **Every (full) node on the blockchain processes every transaction and stores the entire state, just like Bitcoin**
    - It’s bold because it’s important.

Gas

- Halting problem
  - Cannot tell whether or not a program will run infinitely
- Solution: charge fee per computational step (“gas”)
- Special gas fees also applied to operations that take up storage

Gas limit

- Counterpart to the block size limit in bitcoin
- Voting mechanism (can upvote/downvote gas limit by 0.0976% every block)
- Default strategy
  - 4712388 minimum
  - Target 150% of long-term EMA of gas usage

Transactions

- `nonce` (anti-replay-attack)
- `gasprice` (amount of ether per unit gas)
- `startgas` (maximum gas consumable)
- `to` (destination address)
- `value` (amount of ETH to send)
- `data` (readable by contract code)
- `v, r, s` (ECDSA signature values)

Receipts

- Intermediate state root
- Cumulative gas used
- Logs

Logs

- Append-only, not readable by contracts
- ~10x cheaper than storage
- up to 4 “topics” (32 bytes), plus data
- Intended to allow efficient light client access to event records (eg. domain name changed, transaction occurred, etc)
- Bloom filter protocol to allow easier searching by topic

Ethereum Virtual Machine

- Stack
- Memory
- Storage
- Environment variables
- Logs
- Sub-calling

High-level languages

- Compile to EVM code
- Serpent
- Solidity
- LLL

The ABI

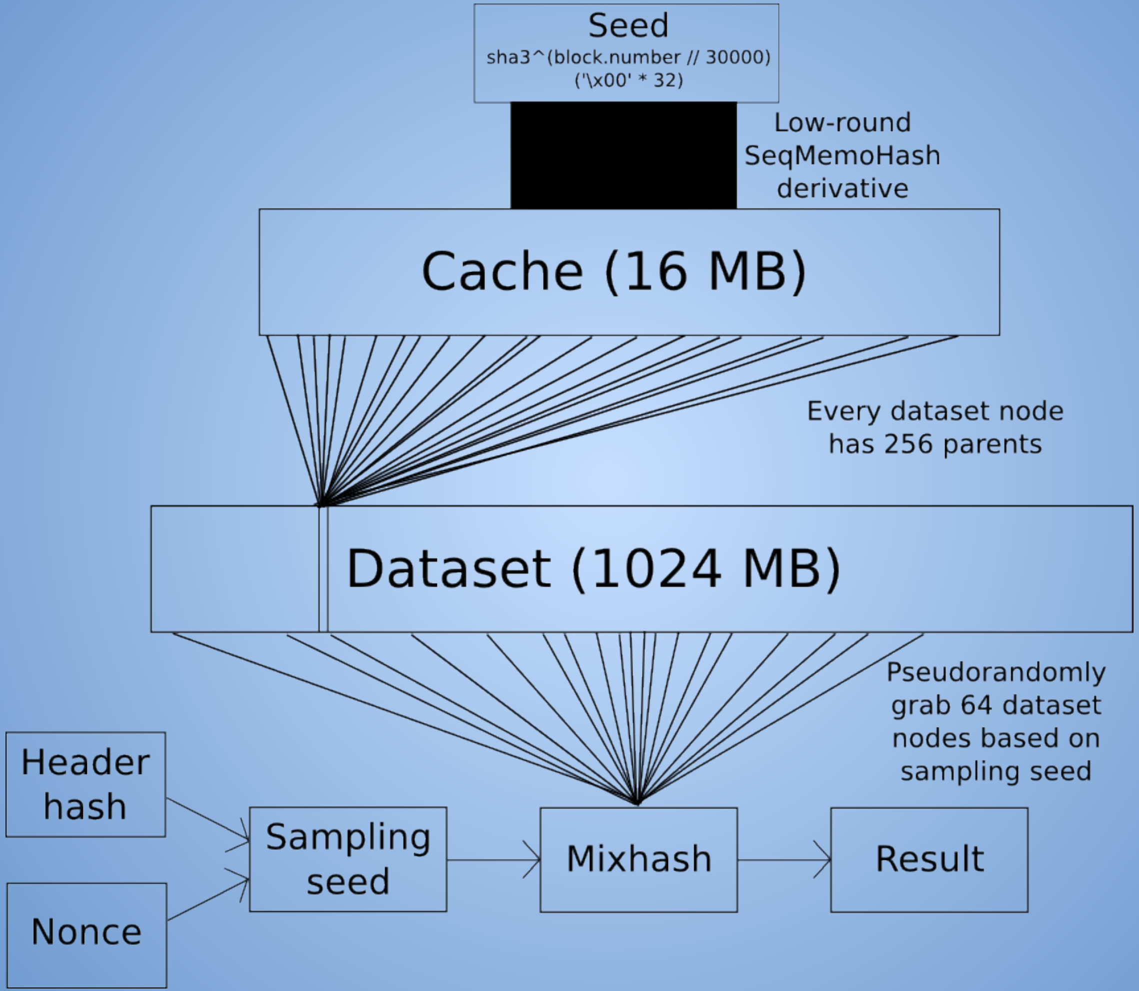
- Function calls are compiled into transaction data

e638984d666f6f2e65746800000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000
0000000000000012345678

- First 4 bytes: function ID
- Next 32 bytes: first argument (“foo.eth”)
- Next 32 bytes: second argument (18 52 84 120 -> 0x12345678)

A cool new mining algorithm

- Goal: GPU-friendly, ASIC-hard
- CPU-friendly too risky a target, GPU “safe” balance, also limits botnet/AWS concerns
- Uses memory-hardness to achieve this goal
- Uses a multi-level DAG construction to achieve this with light-client friendliness

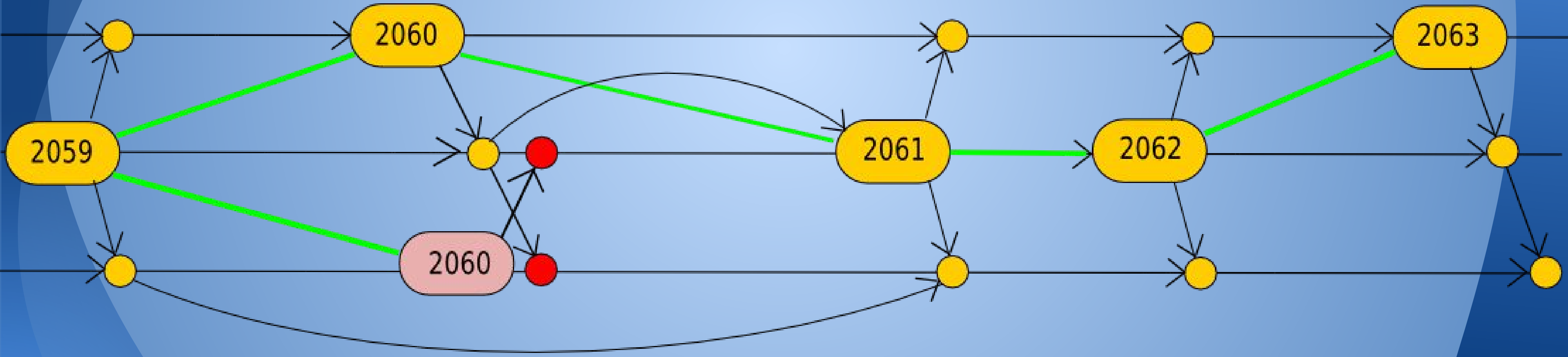

Seed
sha3^(block.number // 30000)
('\x00' * 32)
Low-round
SeqMemoHash
derivative
Cache (16 MB)
Every dataset node
has 256 parents
Dataset (1024 MB)
Pseudorandomly
grab 64 dataset
nodes based on
sampling seed
Header
hash
Nonce
Sampling
seed
Mixhash
Result

A cool new mining algorithm

- **Full node mining strategy**: re-build the full DAG every 30000 blocks, quickly compute mixhashes
- **Light node verification strategy**: re-build only the cache every 30000 blocks, more slowly compute mixhash once by computing 256 DAG nodes on the fly
  - Less efficient, but we only need to do this once

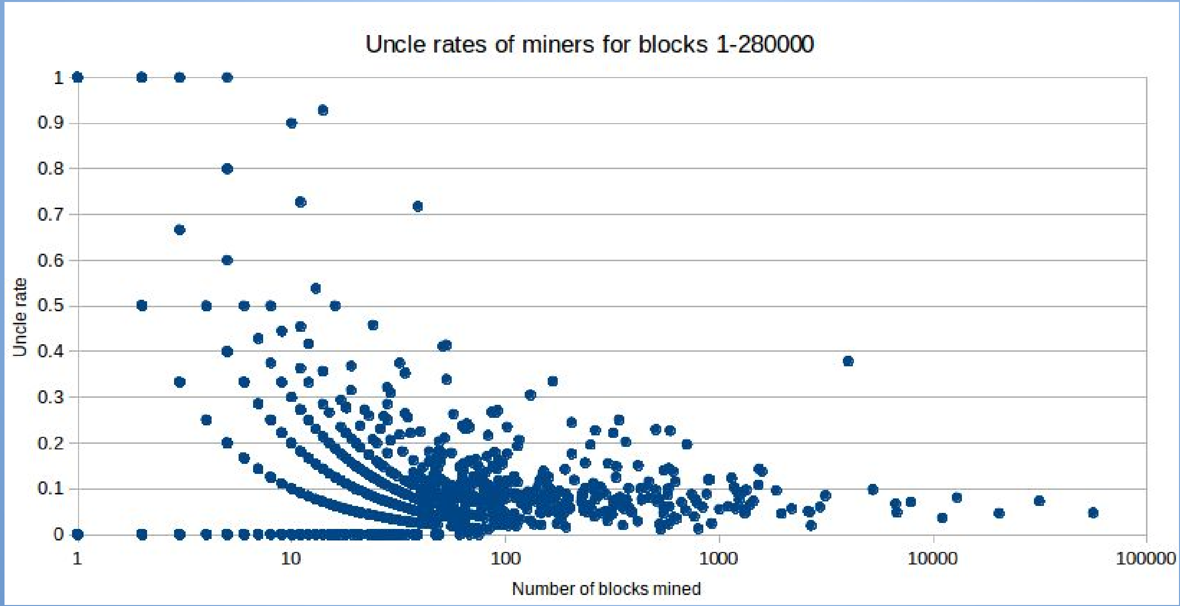
Fast block times

- The problem with fast block times: stale rates due to network latency



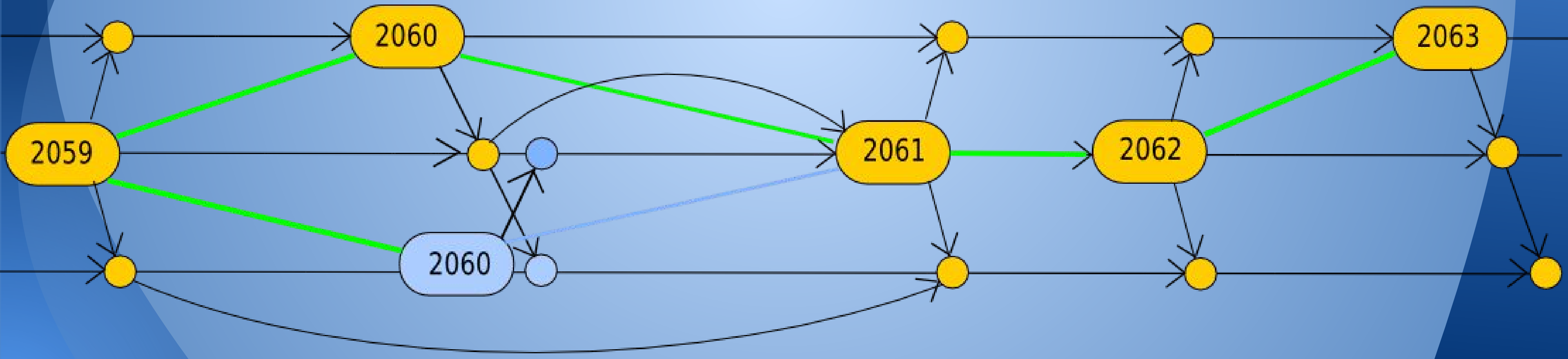
Normally, this leads to centralization risks

- Last block zero stale rate effect
- Network connectivity economies of scale



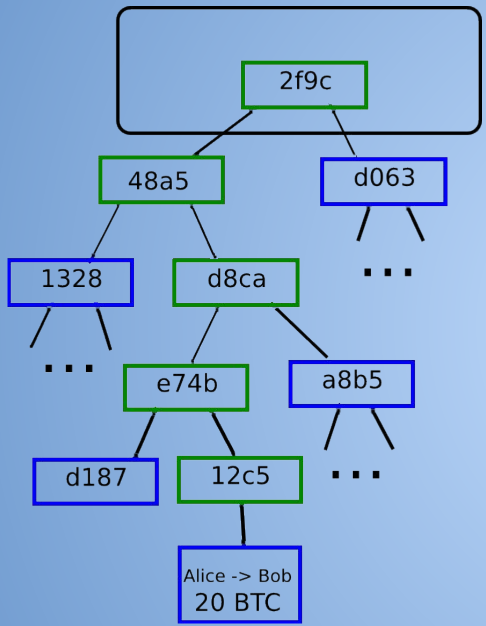
Our solution: uncles

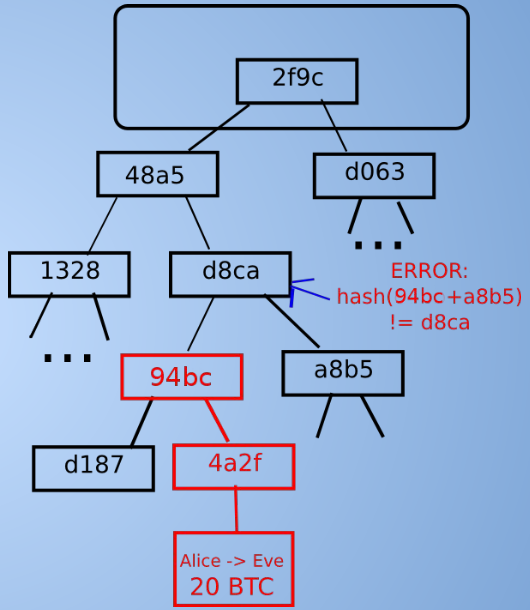
- Our solution: allow stale blocks to be re-included and receive most of the reward
- Cuts centralization incentives by ~85%



Merkle trees

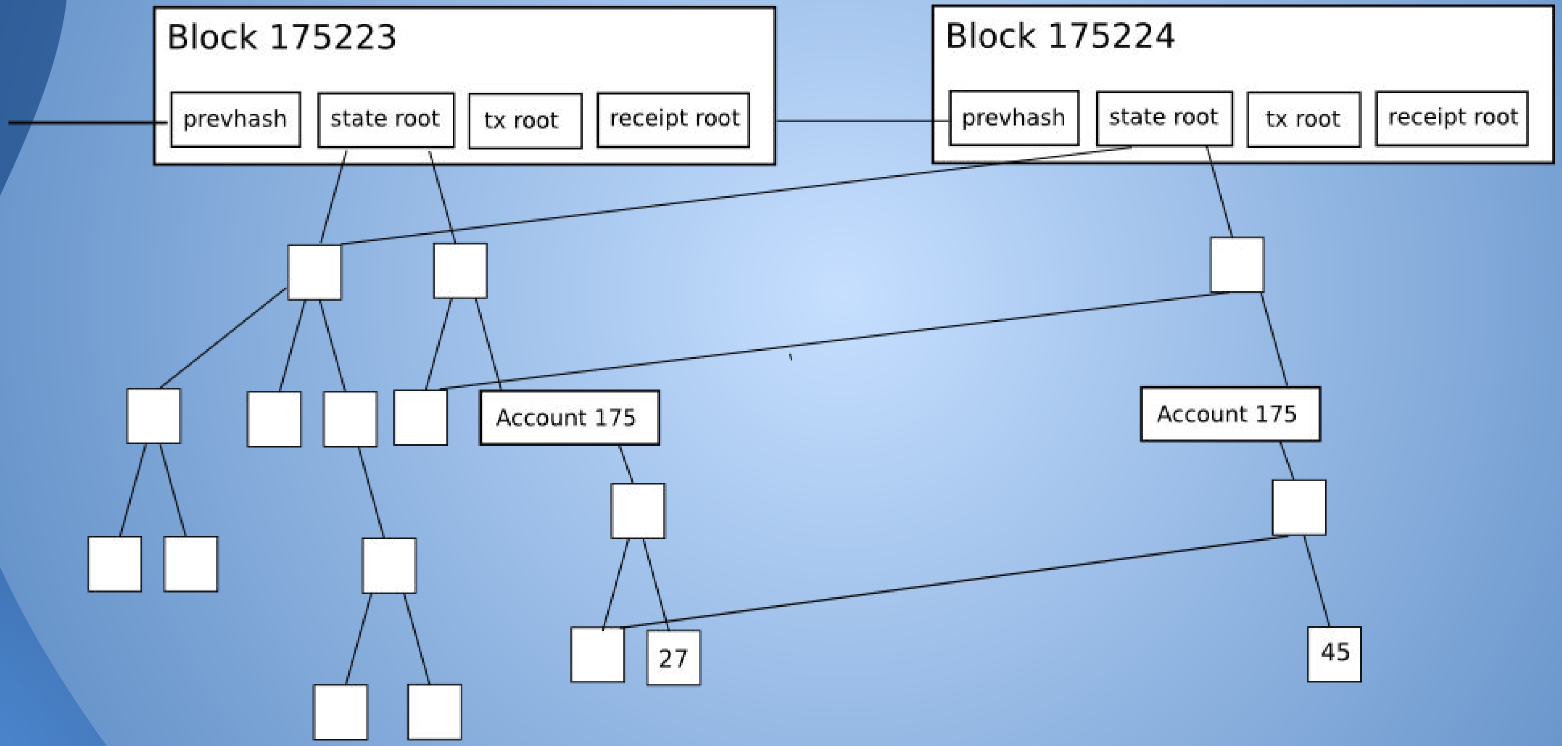
Merkle trees





Allow for efficiently verifiable proofs that a transaction was included in a block

Merkle trees in Ethereum



Merkle trees in Ethereum

- Every block header contains three tries
  - Transactions
  - State
  - Receipts
- Patricia trees used in order to allow efficient insert/delete operations

Future directions

- PoS
- Privacy support (ZK-snark precompiles, ring sigs, etc)
- Blockchain rent
- VM upgrades
- More flexible use of storage
- Scalability