
The Mauve Revolution


I THINK WE SHOULD BUILD A BLOCKCHAIN
UH-OH
S Adams E-mail: SCOTTADAMS@AOL.COM
DOES HE UNDERSTAND WHAT HE SAID OR IS IT SOMETHING HE SAW IN A TRADE MAGAZINE AD?
© 1995 United Feature Syndicate, Inc.(NYC)
WHAT COLOR DO YOU WANT THAT BLOCKCHAIN?
I THINK MAUVE HAS THE MOST RAM.

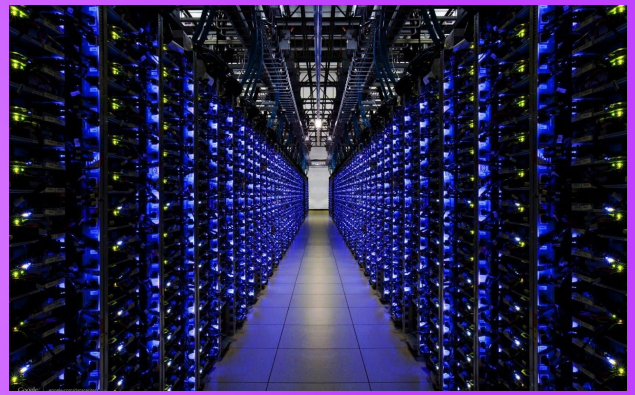
# What sucks about Ethereum?

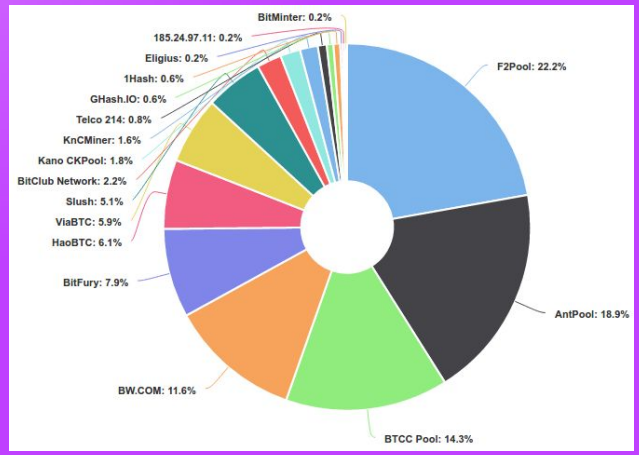
1. **Privacy** - low at base protocol layer
2. **Scalability** - 15 tx/sec
3. **Cost** - ~$360k/day wasted electricity + hardware depreciation from PoW
4. **Latency** (ie. block time) - 14 sec for 1 block, 3 min for de-facto finality, no “true” finality

For solutions to (1) see “Zcash + Ethereum = ❤” plus ring signatures. Here we’ll focus on (2) and (3) with a bit of (4).

# And also...

- Mining centralization
- Present but suboptimal light client support
- Selfish mining vulnerabilities
- State transition interface unclean (eg. access to last 256 block hashes required)
- .....





# Step 1: Basic proof of stake

- PoS originally pioneered by Peercoin (2011), now many versions exist (NXT, Tendermint, Flying Fox, etc)
- Solves “wasted electricity” problem and arguably mitigates centralization risks

# My intuition: “Virtual mining”

- **PoW**: $1000 -> miner -> stochastically generate blocks
- **PoS**: $1000 -> 83 ETH -> virtual miner (convert in protocol) -> protocol lets you stochastically generate blocks

# The Casper PoC3 approach

- In the state, there is a set of bonded validators
- Anyone can join this set by sending a transaction (or more precisely, a call) sending ether to the Casper contract (address `0x000...00ff`) and invoking the method:

```
function deposit(bytes validation_code, bytes32
randao, address withdrawal_addr) {}
```

- If the call is made during epoch n (1 epoch = 12 hours), the validator is inducted at the start of epoch n+2

# The Casper PoC3 approach

- While a validator is inducted, they will sometimes be pseudorandomly be assigned the right to create blocks, with a probability proportional to the ether deposited
- A block must contain a signature in its extra data. For the signature to be valid, the result of `CALL_BLACKBOX(validation_code, signature)` must be successful and nonempty
    - Generalized signature algorithm; if you are paranoid about QC, go ahead and use Lamport

# The Casper PoC3 approach

- A validator can call `startWithdrawal` to withdraw; this once again takes effect after 2 epochs
- After withdrawing, a validator can `withdraw` their deposit, plus rewards minus penalties, to the pre-specified withdrawal address after waiting N days

Let’s talk about stake grinding!

# Validator selection

- **Problem**: how do we choose which validator creates the next block?
- “**Stake grinding**”: may exist an economic incentive for a validator to perform computations to manipulate validator selection in their favor
- **eg. old NXT algorithm:** validator pseudorandomly sampled based on signature of the current block, creates incentive to “grind” favorable signature
- **eg. some old PoS algos:** incentive to find favorable address to move one’s coins to

# Validator selection

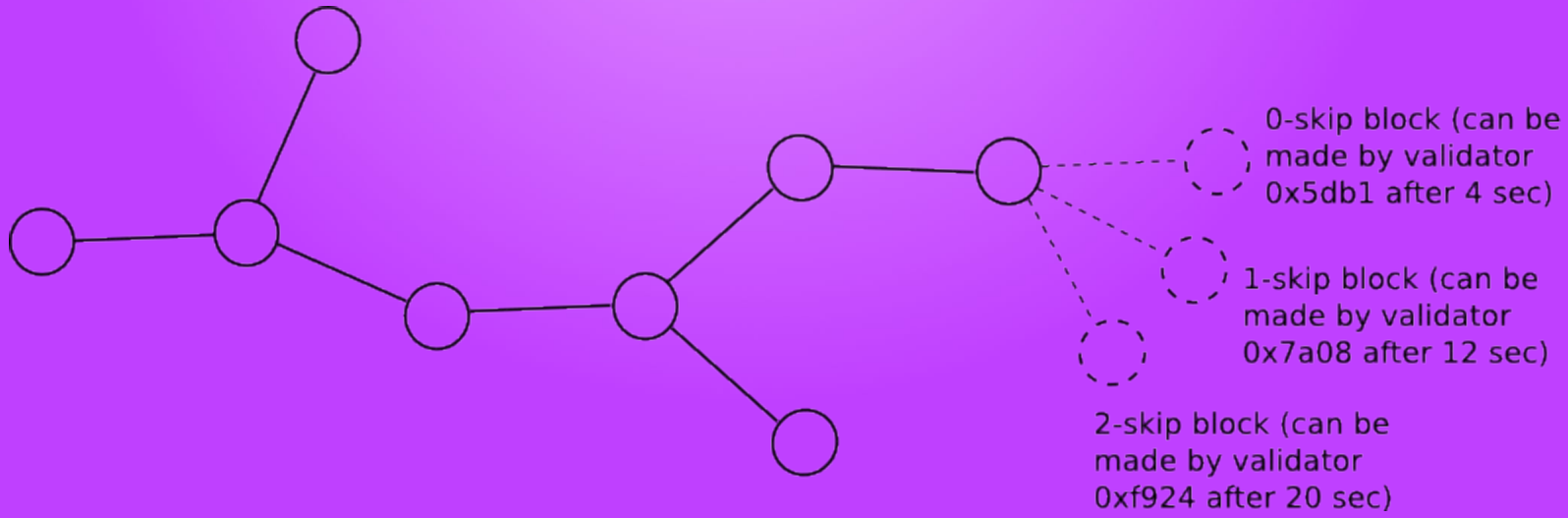
- “**Marginal cost => marginal revenue**” - used by some as knockdown argument to imply that all consensus algos are fundamentally equally wasteful
- **CAVEAT 1**: requires plausible mechanism to expend cost to gain unfair revenue advantage
- **CAVEAT 2**: MC => MR does NOT imply total cost => total revenue
- **CAVEAT 3**: PoS requires much lower block rewards

# Our approach

- When making a deposit, every validator submits a commitment `sha3^1000000(x)` for some secret seed `x`
- When creating a block, a validator must reveal the previous link in their hash chain
- The Casper contract maintains a parameter `globalRandao`, initialized to zero, an XORs in revealed values every block; this is the source of randomness from which validators are selected
- Result: only the validator of the next block is known

# Validator selection

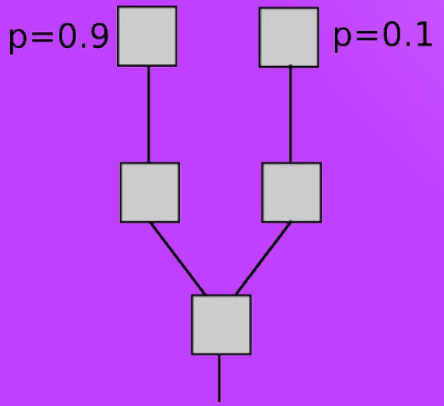
- What if the selected validator is not present?
- Instead of selecting one validator, we select a sequence of validators. Validator k can make a block after 4 + 8k seconds



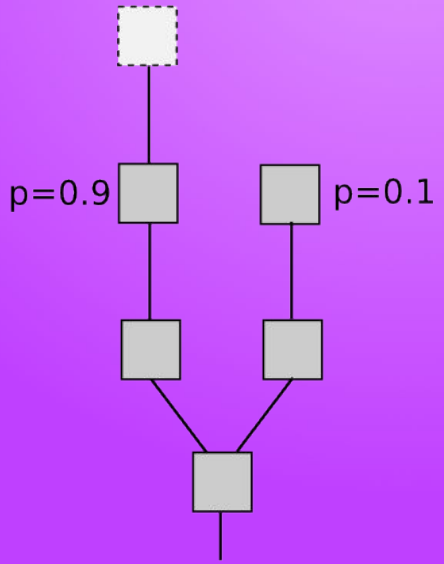
Let’s talk about nothing at stake!

# PoW economics (normal case)

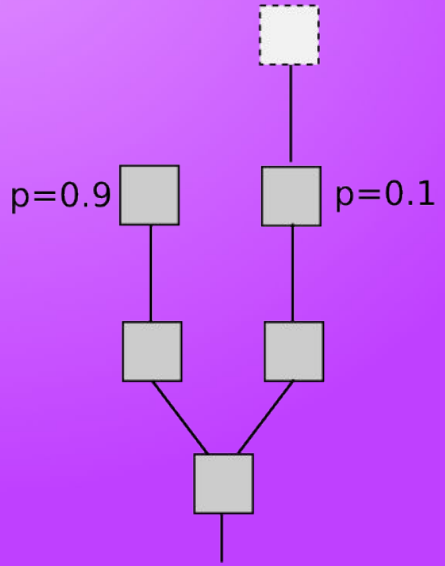
Vote on neither
EV = 0



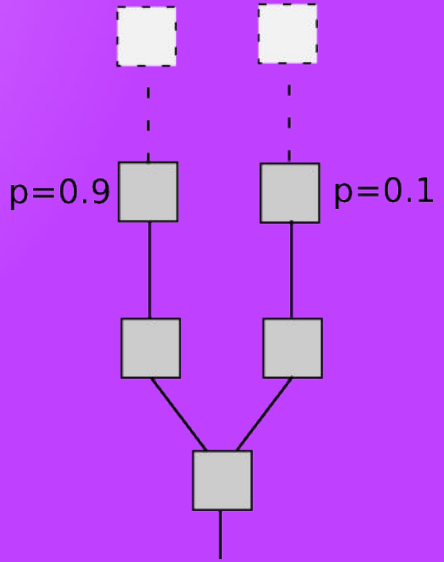

Vote on A
EV = 0.9




Vote on B
EV = 0.1

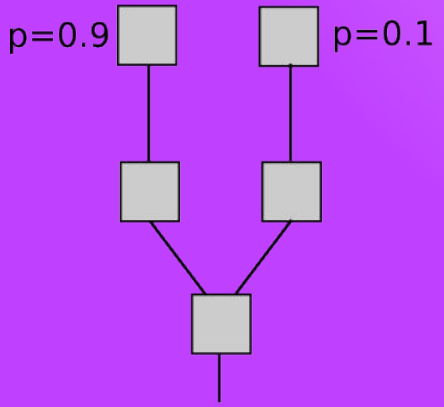



Split vote between both
EV = 0.05 + 0.45 = 0.5

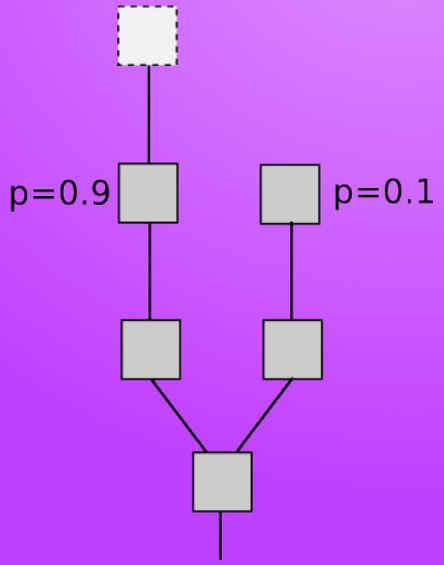


# Naive PoS economics

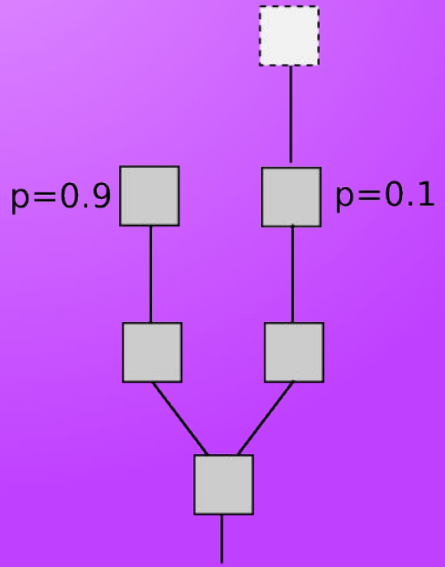
Vote on neither
EV = 0



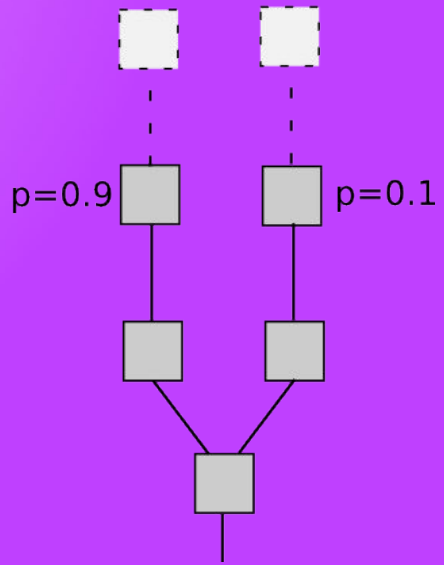

Vote on A
EV = 0.9




Vote on B
EV = 0.1




Vote on both
EV = 0.1 + 0.9 = 1

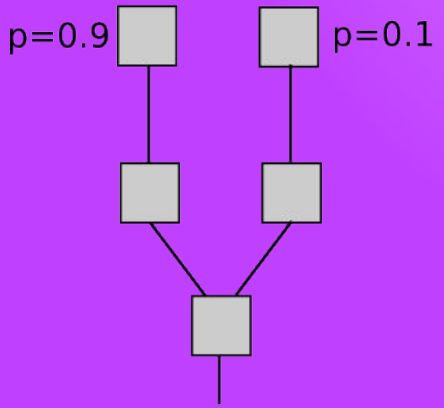


# Dunkles

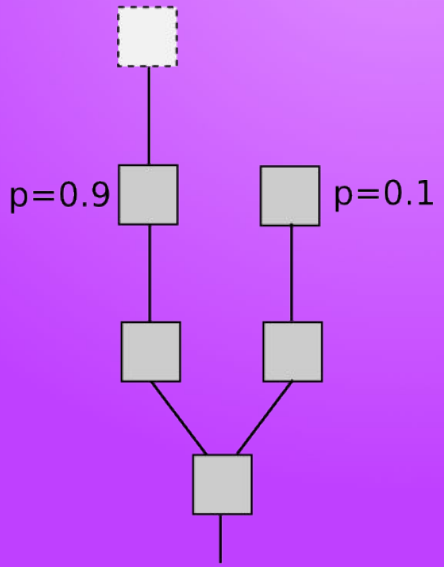
- If you make a block but it doesn’t make it into the main chain, you lose an amount of ether equal to the block reward
- Dunkle = “Dark uncle” - like an uncle but it hurts you

# Casper economics with Dunkles

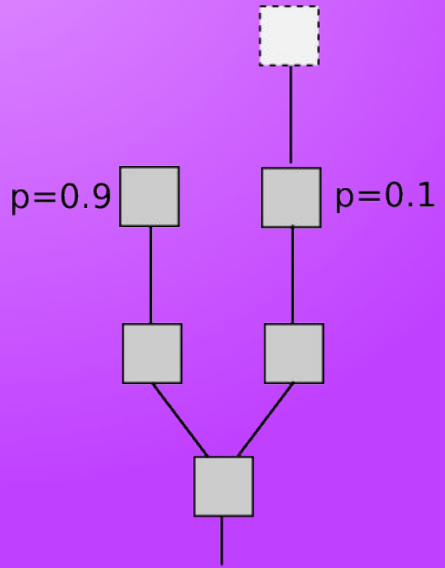
Vote on neither
EV = 0



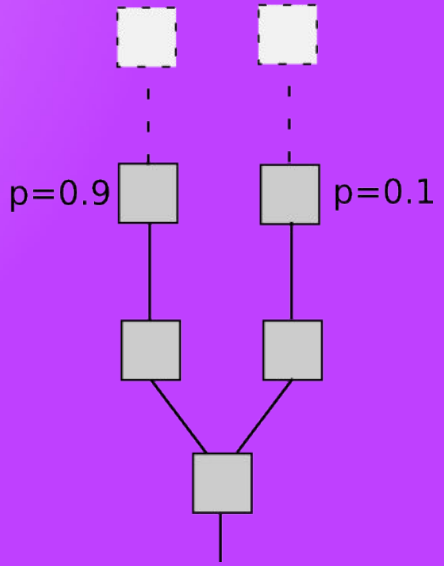

Vote on A
EV = 0.9




Vote on B
EV = 0.1 - 0.9 = -0.8




Vote on both
EV = 0.1 + 0.9 - 1 = 0



Let’s talk about finality!

# Finality

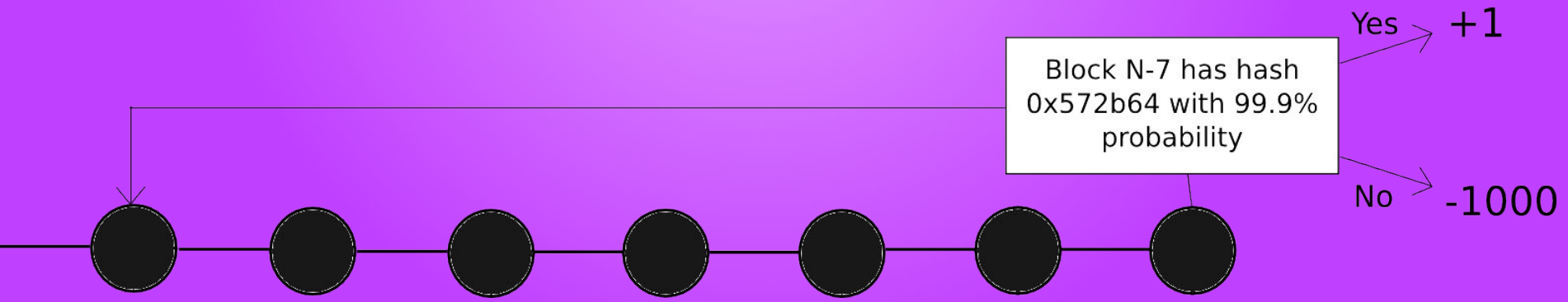
- **CAP theorem: “in the event of a partition, you can have either consistency or availability, not both”**
  - Sketch proof: if network splits in half, send incompatible txs A, B to halves. Either both get accepted (inconsistent) or at least one is blocked (unavailable)
- Our solution: an availability-favoring base layer, with a consistency-favoring “finality gadget” layered on top of it

# The fork choice rule

- In PoW, basic rule is “blockchain with most PoW backing it wins”
- In Casper, basic rule is “blockchain with most value-at-loss backing it wins”
- Value-at-loss = “how much ether have validators staked on this blockchain?” = “how much ether have validators agreed to lose in all blockchains other than this one?”

# Finality

- Blocks don’t just point to the previous block, they also make a bet on some historical block



# Validator chooses odds parameter, V_LOSS and V_GAIN derived from odds

# Finality

- Bets start off at low odds, signifying fear of short-range forks
- Over time confidence increases and odds increase, and particularly odds increase when validators see other validators increasing their odds
- VaL increases exponentially, eventually hitting “finality” (ie. most validators lose ALL their deposits in all histories that conflict with the finalized one)

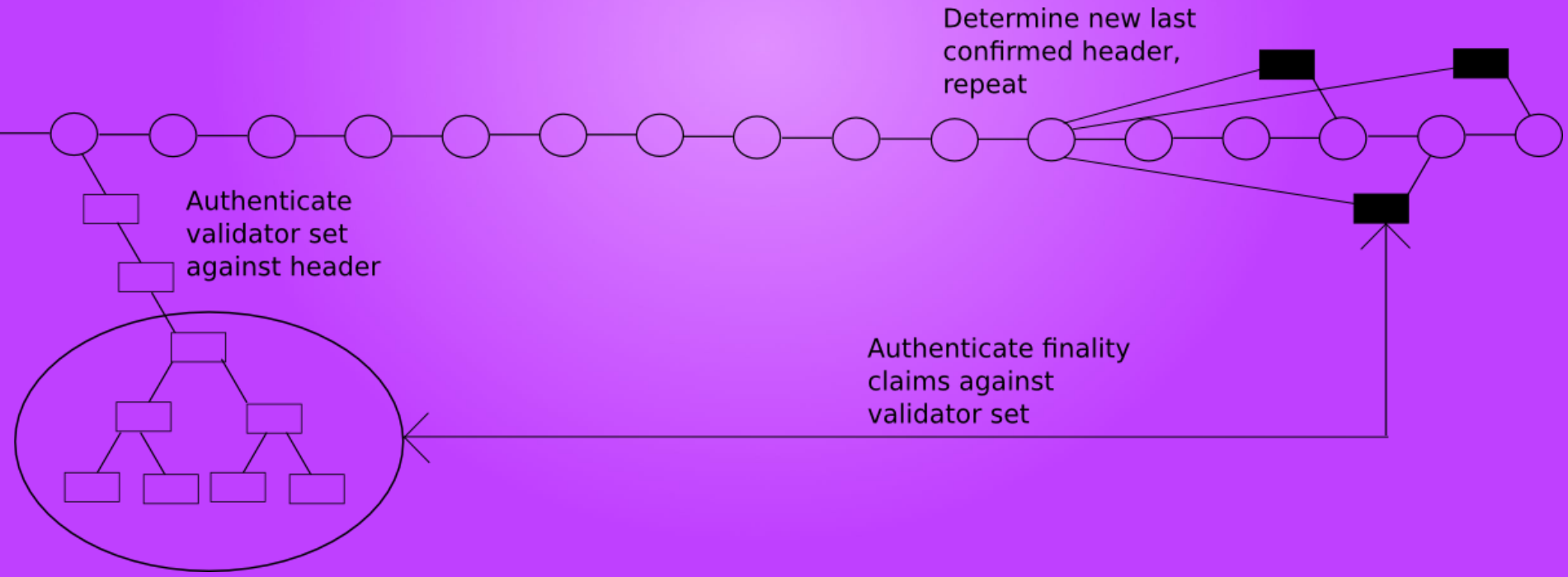
# Finality

- When a block is finalized, the “finality cycle” starts again on a new block
- Restrictions exist to prevent large bets from being made too quickly (otherwise whales could too easily conduct short-range 51% attacks)

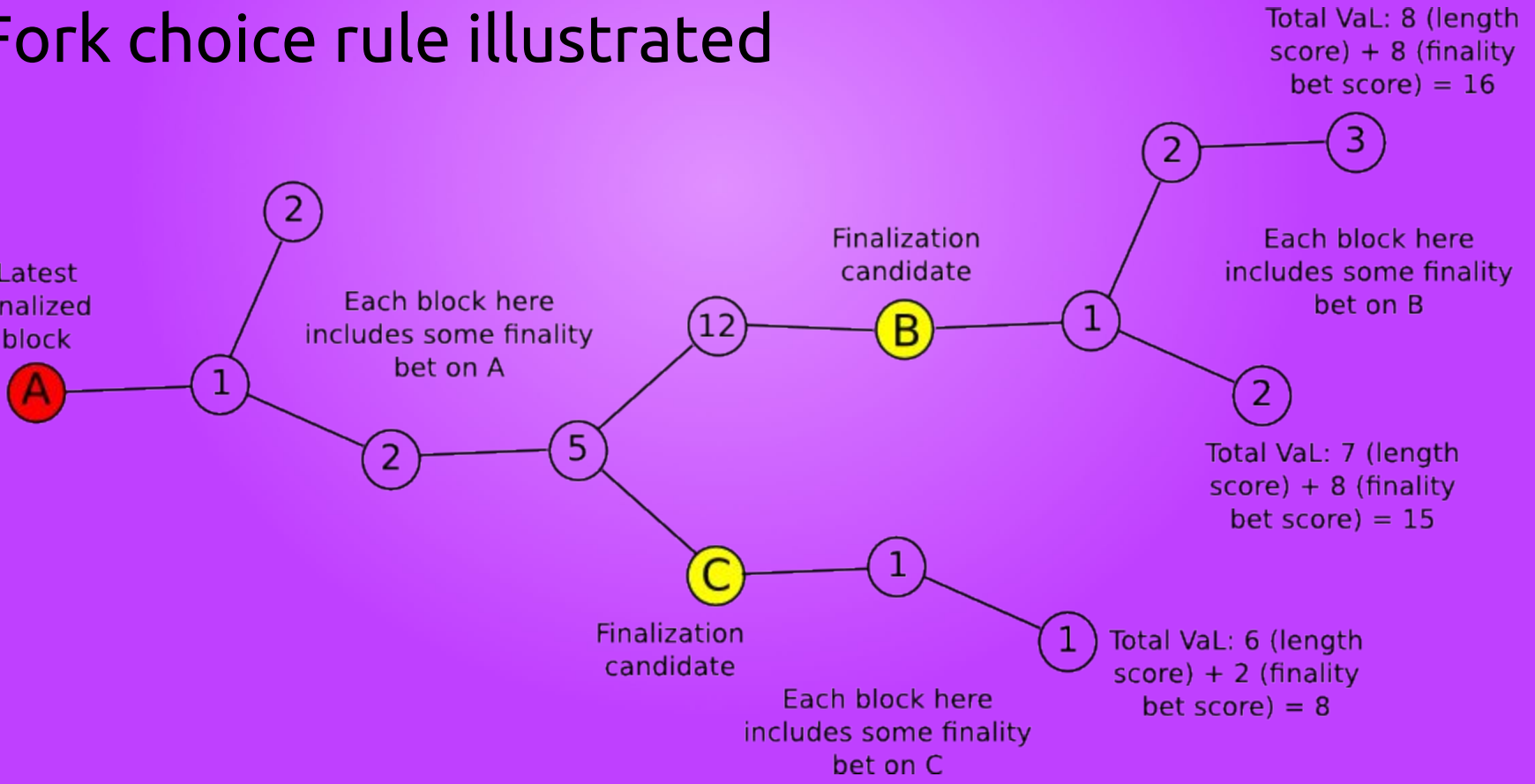
# Finality: why?

- Make even majority-collusion-backed forks/rollbacks very expensive so as to be impractical
- Allow for near-O(1) light client syncing

# Light client syncing



# Fork choice rule illustrated



Let’s talk about sharding!

# The dream

- Achieve **on-chain scaling** to **tens of thousands of transactions per second**…
- Through a fully decentralized peer-to-peer network that can, if needed, run on **nothing but consumer laptops**
- Because every node in the network only keeps track of a **small portion of transactions**/state but can **Merkle-verify everything if needed**

# Sharding

- There exist multiple shards, each of which is kind of like an independent blockchain, but which:
  - Share security (PoS)
  - Have a common market (ether transferability and cross-chain communication support)
  - Have policy harmonization (same EVM, same rules on every shard)
  - Even still, the gas prices are higher on shard 0 than shard 57

# Sampling

- Every epoch, we pseudorandomly select 100 validators to validate each shard
- All validators make finality bets on all shards; the source information is the block headers produced by the shard validators, but also can include fraud proofs and other mechanisms for catching those unlucky cases where one shard is 51% attacked

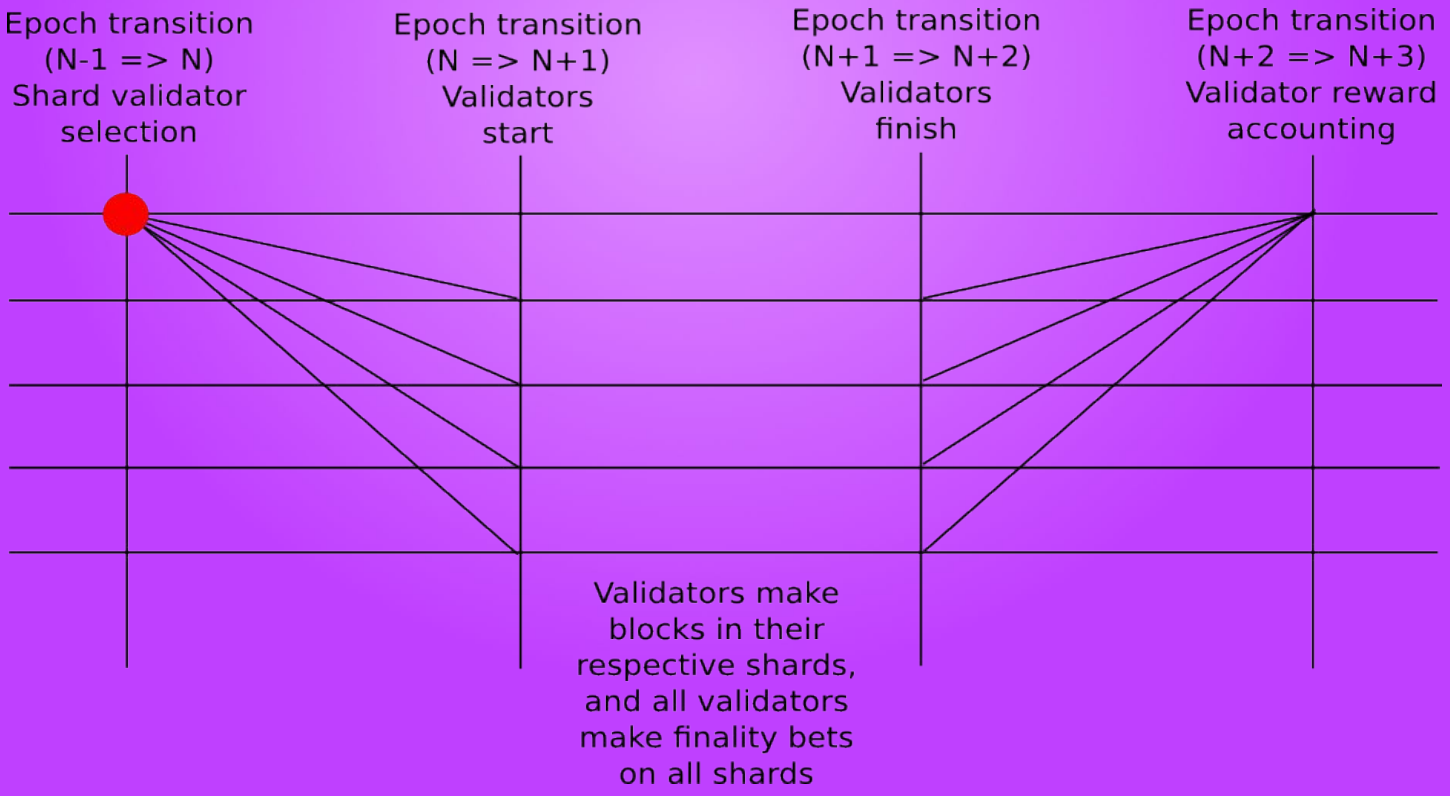
# Cross-shard communication

- State transition function:

  $\texttt{stf(state}_{k}\texttt{, block, receipts}_{0\ldots n-1}\texttt{) => state'}_{k}$

- Function has access to all sufficiently old receipts from other shards; however, receipts must be (i) statically referenced in the transaction if accessed, (ii) Merkle-proven
- New log type `ETHLOG` exists to transfer ether cross-shard

# Consensus-layer cross-shard communication



# Find out more!

- Read the Mauve Paper
- Look at the PoC
  - http://github.com/ethereum/pyethereum/tree/state_revamp
  - http://github.com/ethereum/research/tree/master/casper3