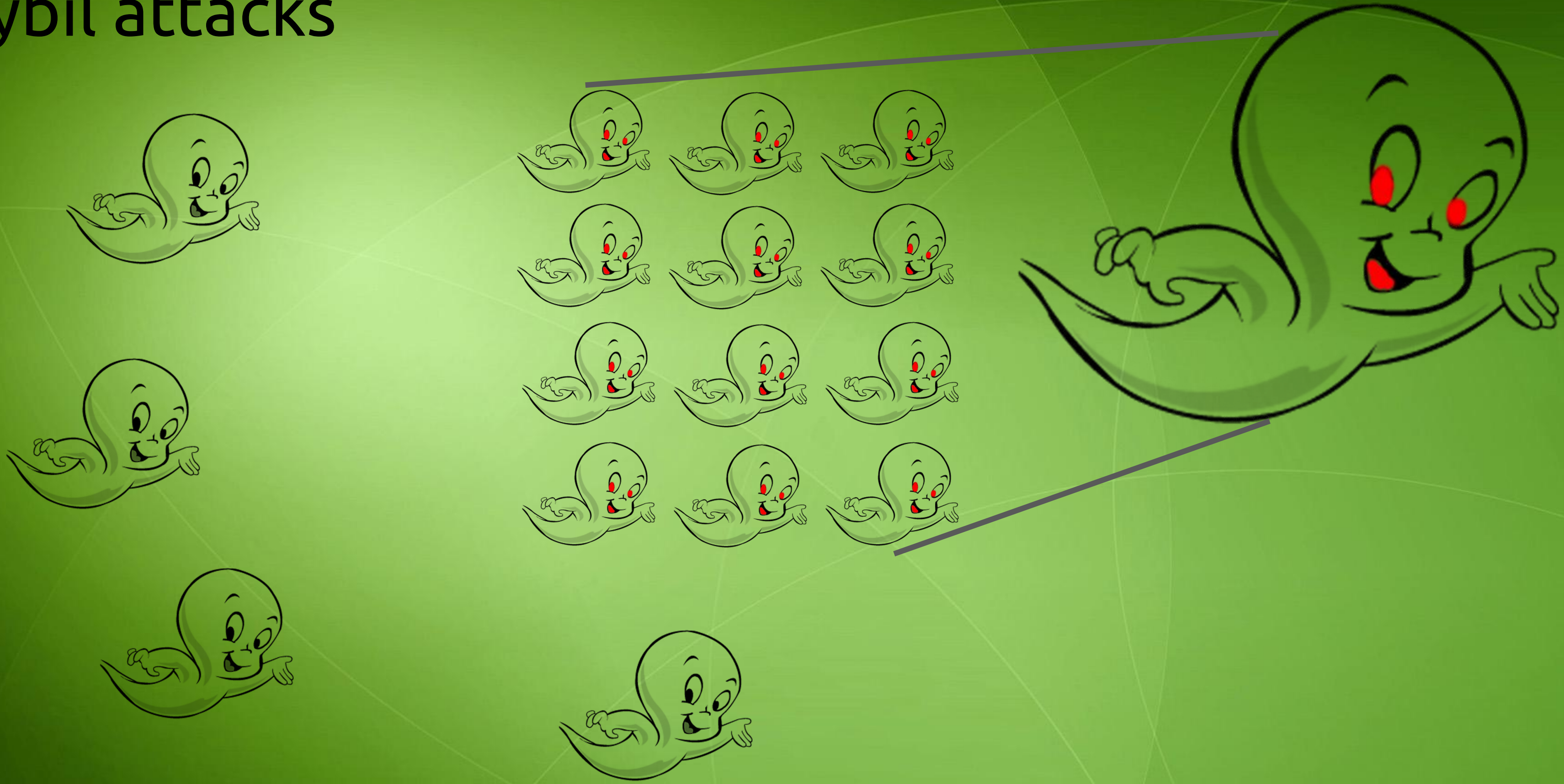
Proof of Stake

The Role of PoW

- Bitcoin, Ethereum and similar systems are open, permissionless networks
- Anyone can participate
- The system must agree on some “canonical order” of transactions
- Think of this mechanism as being like some kind of “voting”

Sybil attacks



The Role of PoW

- Economic defense against sybil attacks
- Idea: participants prove ownership of $X worth of computer hardware by running computations on it 24/7 and uploading results to the network
- Computation must be **hard to perform, easy to verify**

PoS

- Instead of using computation as the limiting economic resource, use digital assets inside the system
    - Ie. bitcoin, ether, or whatever other coin inside the platform
- Proving ownership of digital assets is easy: just sign a digital signature

Chain-based PoS

- Attempts to closely replicate PoW
- Usual algorithm: every coin holder has some chance per second of being assigned the right to create a block
  - Probability is proportional to coin balance

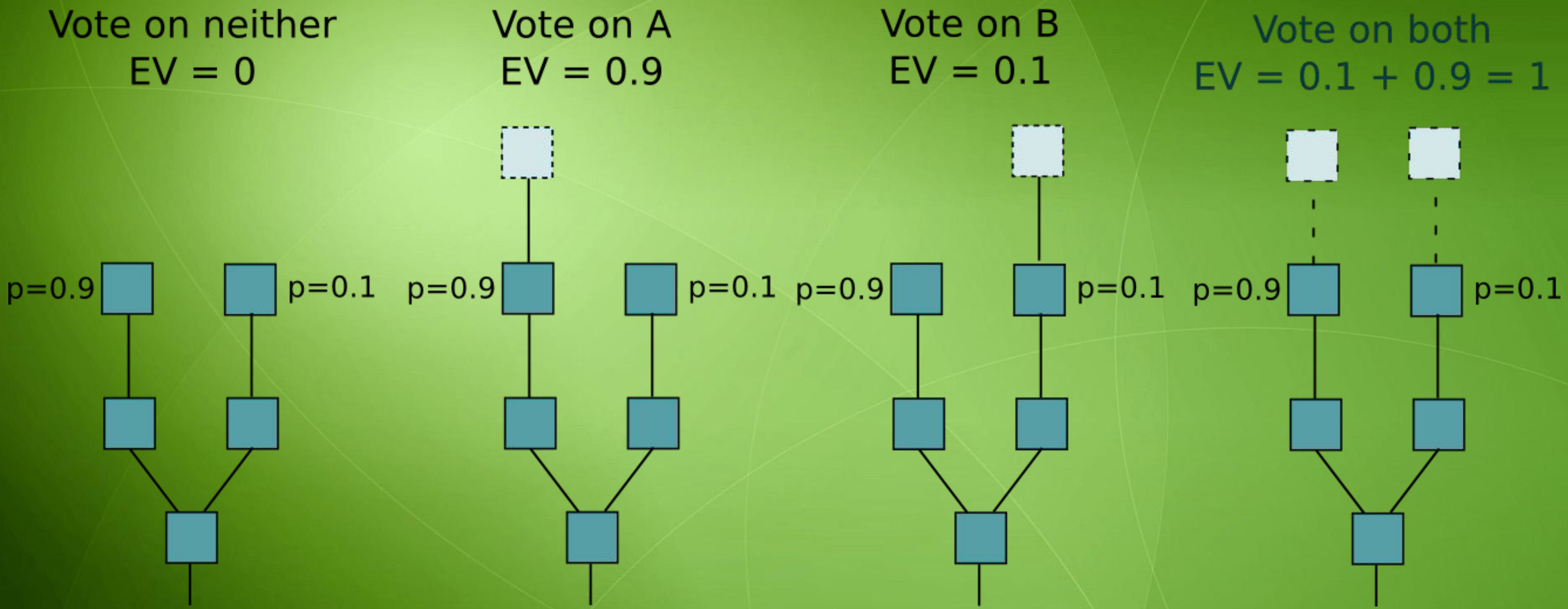
RNG Manipulation

- PoS block maker selection can only be pseudorandom, based on data within the chain?
- Can the RNG be manipulated?
- Idea: predict who will make future blocks if you do A vs B. Choose the option that will maximize your future revenue, even if its immediate rewards are lower.
- Possibly serious centralization risk

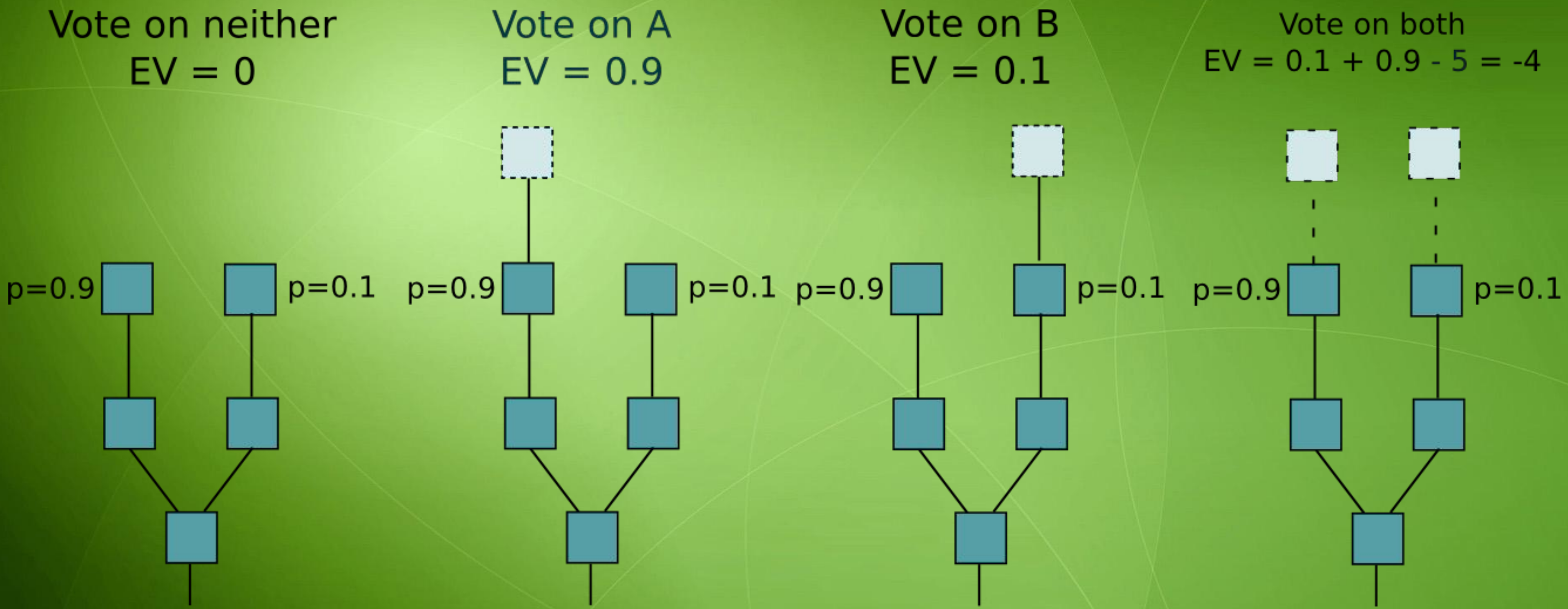
RNG Manipulation

- NXT:
  - `s(G) = 0`
  - `s(B) = h(s(B.parent) ++ B.parent.proposer)`
- RANDAO
  - `s(B) = h(s(B.parent) ++ B.reveal)`
  - Reveal must be pre-hash-committed-to
- Low-influence functions (eg. N-block majority rule)

Nothing at stake flaw



Penalty-based PoS

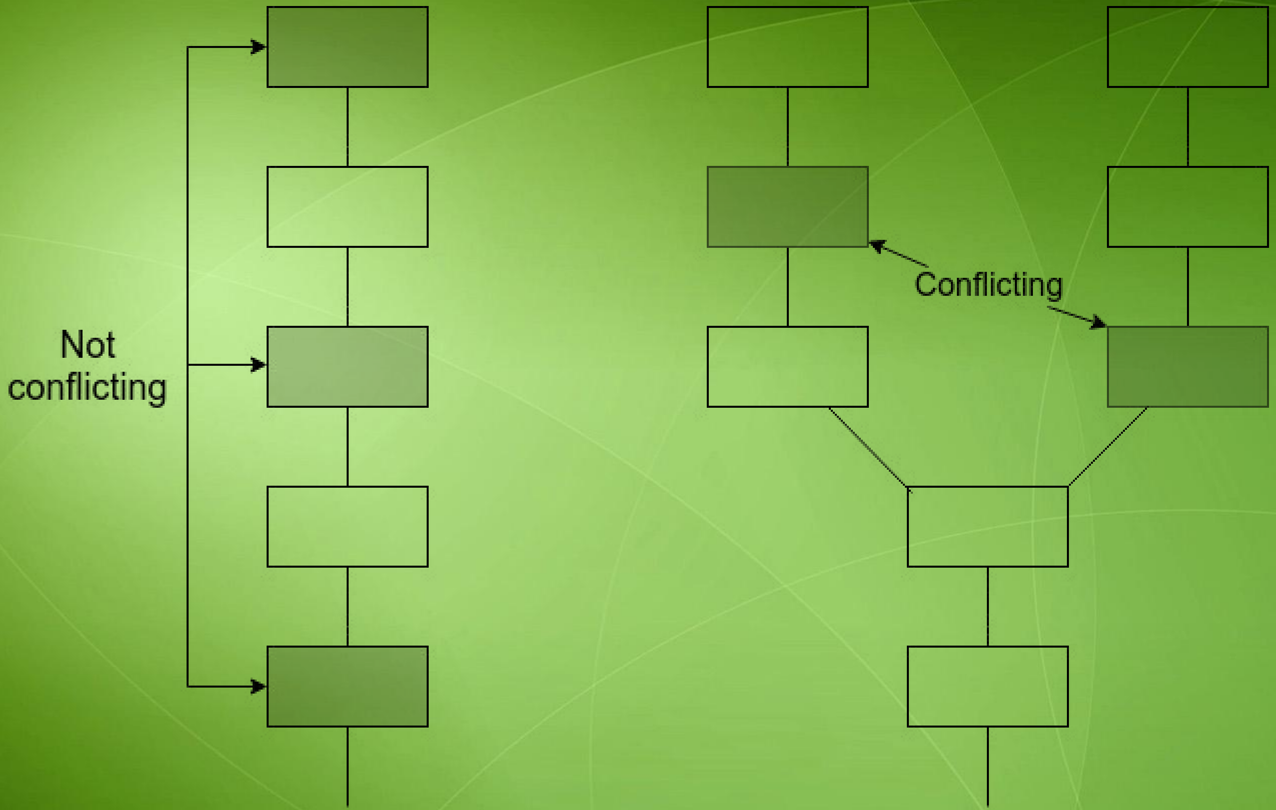


Casper overview

- **Deposit** + **penalty**-based proof of stake
- Anyone can join as a **validator** by submitting a **deposit** (in ETH)
- Validators are penalized for contradicting themselves
- Heavily based on traditional BFT theory (Lamport, DLS, PBFT, ByzPaxos, etc) but remodeled to be more “chain-based”

Security claims

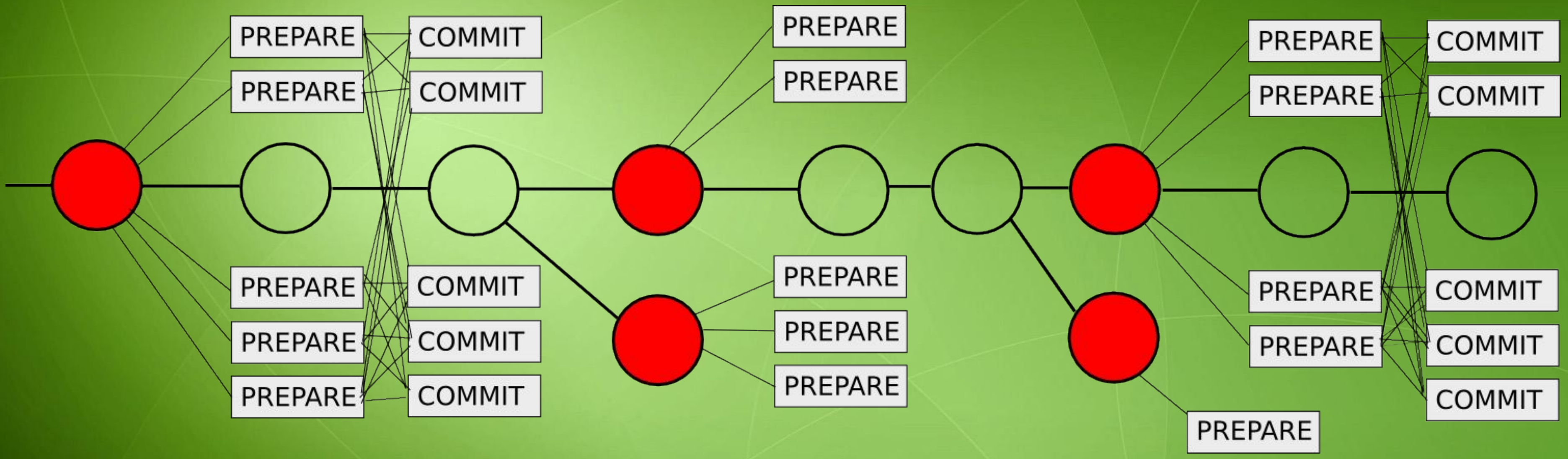
- The algorithm defines a notion of “finality”; if a block is finalized then it is set in stone in history and we can never go back
- **Accountable safety**: if two conflicting blocks get finalized, then at least ⅓ of validators may lose their entire deposits
- **Plausible liveness**: it’s always possible to finalize more blocks with ⅔ of validators


Not conflicting
Conflicting

Stage 1: Hybrid design

- Miners continue to create **proof of work blocks**
- Every 100th block is a **checkpoint**
- Active validators can send “prepare” and “commit” messages on checkpoints

Stage 1: Hybrid design



Justification and Finality

- The genesis is justified.
- If ⅔ of validators prepare a checkpoint C, all claiming some already justified checkpoint C’ as a *source checkpoint*, then C is justified.
- If C is justified, and ⅔ of validators commit C, then C is finalized.

Slashing conditions

- The protocol defines a set of **slashing conditions**, which represent actions that a validators is not supposed to do
- Example:
  - Sending a message when you are not supposed to
  - Sending two messages that contradict each other (eg. one votes for A, one votes for B)

Slashing conditions

- If a validator violates a slashing condition, their deposit is deleted.

A slashing condition might look like this:

> *If a validator sends a signed message of the form*
>
> `["PREPARE", epoch, HASH1, epoch_source1]`
>
> *and a signed message of the form*
>
> `["PREPARE", epoch, HASH2, epoch_source2]`
>
> *where* `HASH1 != HASH2` *or* `epoch_source1 != epoch_source2`*, but the* `epoch` *value is the same in both messages, then that validator's deposit is* ***slashed*** *(ie. deleted)*

```
# Cannot make two prepares in the same epoch
def double_prepare_slash(validator_index: num, prepare1: bytes <= 1000, prepare2: bytes <= 1000):
    # Get hash for signature, and implicitly assert that it is an RLP list
    # consisting solely of RLP elements
    sighash1 = extract32(raw_call(self.sighasher, prepare1, gas=200000, outsize=32), 0)
    sighash2 = extract32(raw_call(self.sighasher, prepare2, gas=200000, outsize=32), 0)
    # Extract parameters
    values1 = RLPList(prepare1, [num, bytes32, bytes32, num, bytes32, bytes])
    values2 = RLPList(prepare2, [num, bytes32, bytes32, num, bytes32, bytes])
    epoch1 = values1[0]
    sig1 = values1[5]
    epoch2 = values2[0]
    sig2 = values2[5]
    # Check the signatures
    assert ecrecover(sighash1,
                     as_num256(extract32(sig1, 0)),
                     as_num256(extract32(sig1, 32)),
                     as_num256(extract32(sig1, 64))) == self.validators[validator_index].addr
    assert ecrecover(sighash2,
                     as_num256(extract32(sig2, 0)),
                     as_num256(extract32(sig2, 32)),
                     as_num256(extract32(sig2, 64))) == self.validators[validator_index].addr
```

Two slashing conditions

- NO_DBL_PREPARE:

  CANNOT prepare two conflicting C1, C2 in one epoch

- COMMIT_PREPARE_CONSISTENCY

  If you prepare in epoch e_p, with source in epoch e_s

  then you cannot commit in any epoch e_c where

  e_s < e_c < e_p

Yoichi Hirai Follow
a convenience logician
Feb 25 · 7 min read

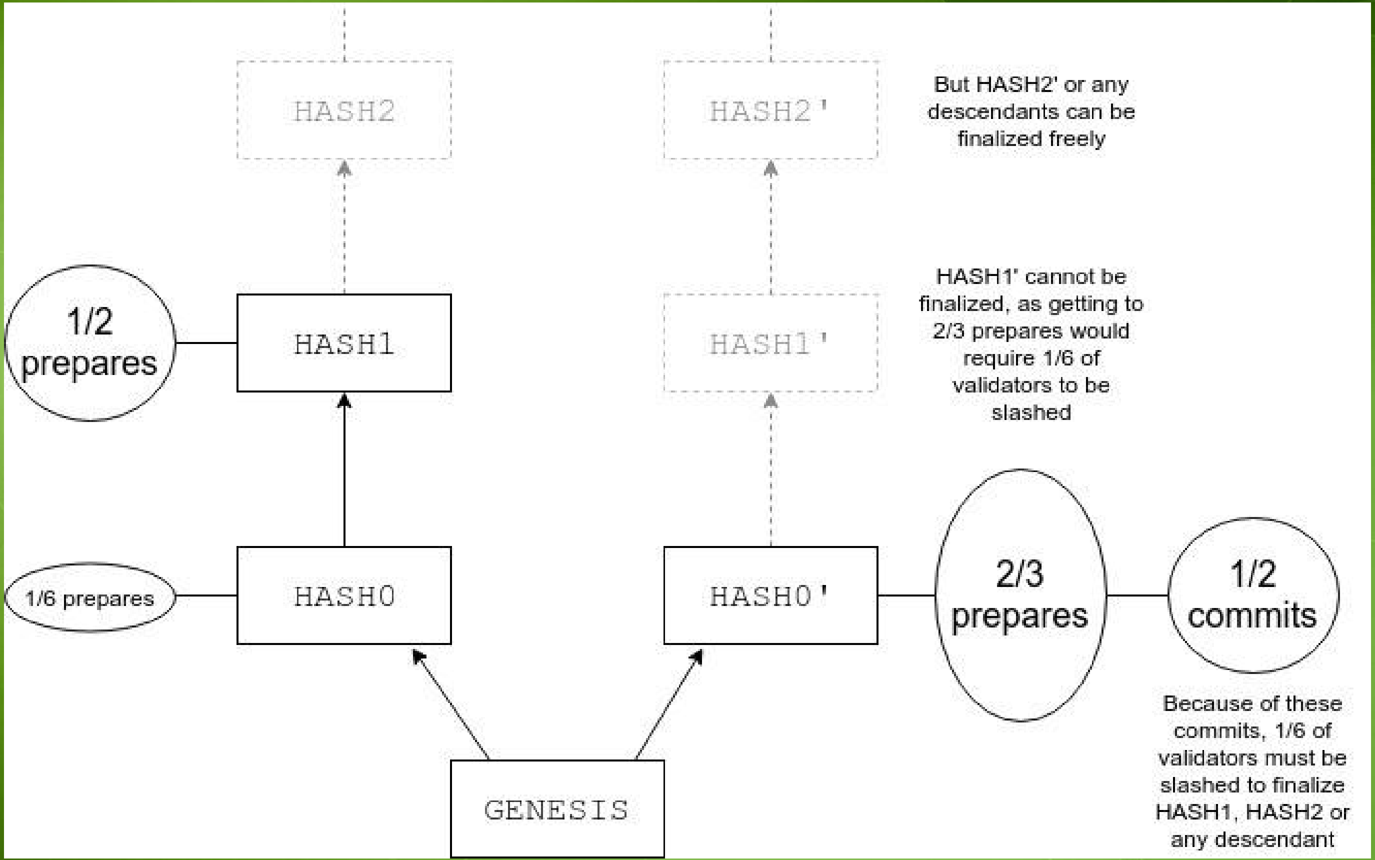
Formal methods on some PoS stuff

In Paris, I received a description of a distributed consensus mechanism. If the description below looks ambiguous or impenetrable, this post is for you. I banged my head to formal verification tools called Alloy and Isabelle/HOL to clarify my understanding (code).

```
Message types:

* commit(HASH, view), 0 <= view
* prepare(HASH, view, view_source), -1 <= view_source < view

Slashing conditions:

* [i] commit(H, v) REQUIRES 2/3 prepare(H, v, vs) for some
consistent vs
```

Fork choice rule

- PoW: longest chain
- For hybrid PoS, PoW miners following the longest chain does not work

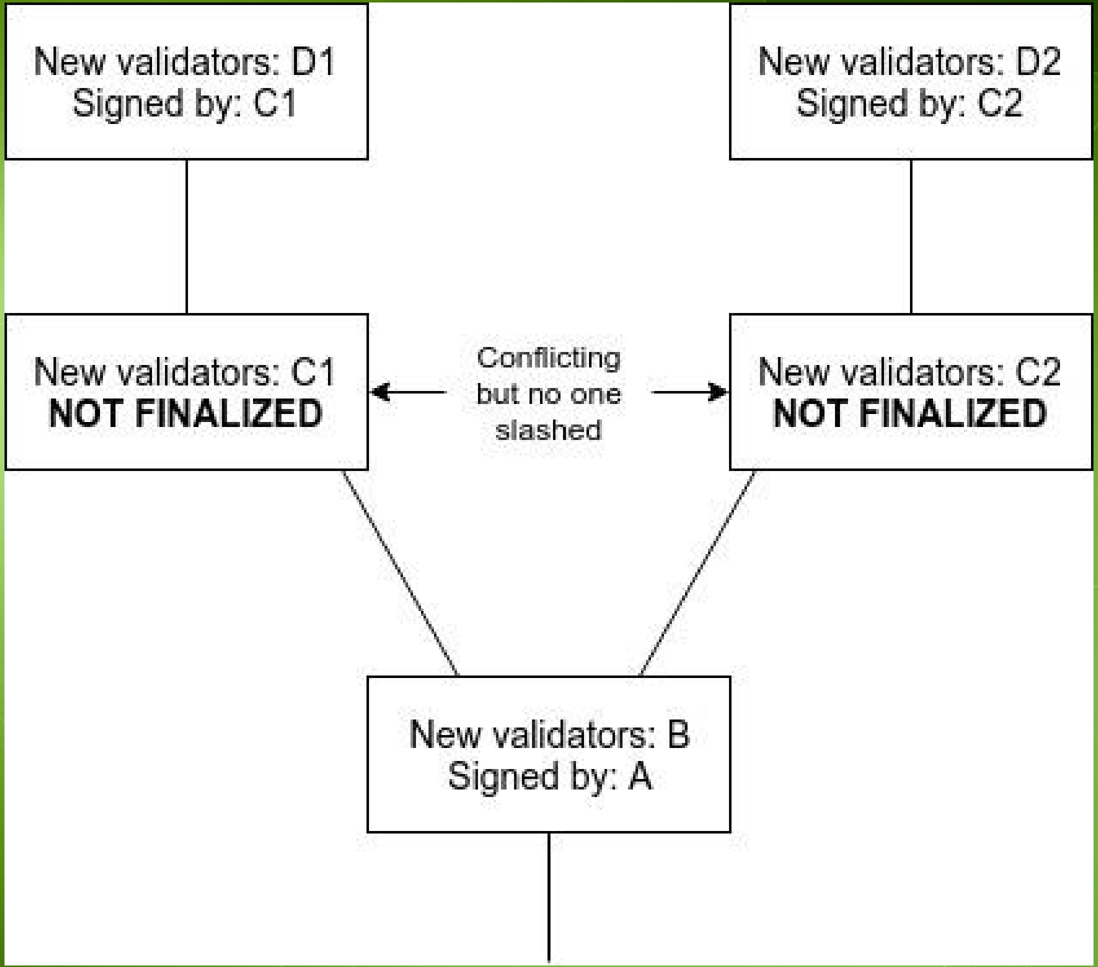

HASH2
HASH2'
But HASH2' or any descendants can be finalized freely
1/2 prepares
HASH1
HASH1'
HASH1' cannot be finalized, as getting to 2/3 prepares would require 1/6 of validators to be slashed
1/6 prepares
HASH0
HASH0'
2/3 prepares
1/2 commits
Because of these commits, 1/6 of validators must be slashed to finalize HASH1, HASH2 or any descendant
GENESIS

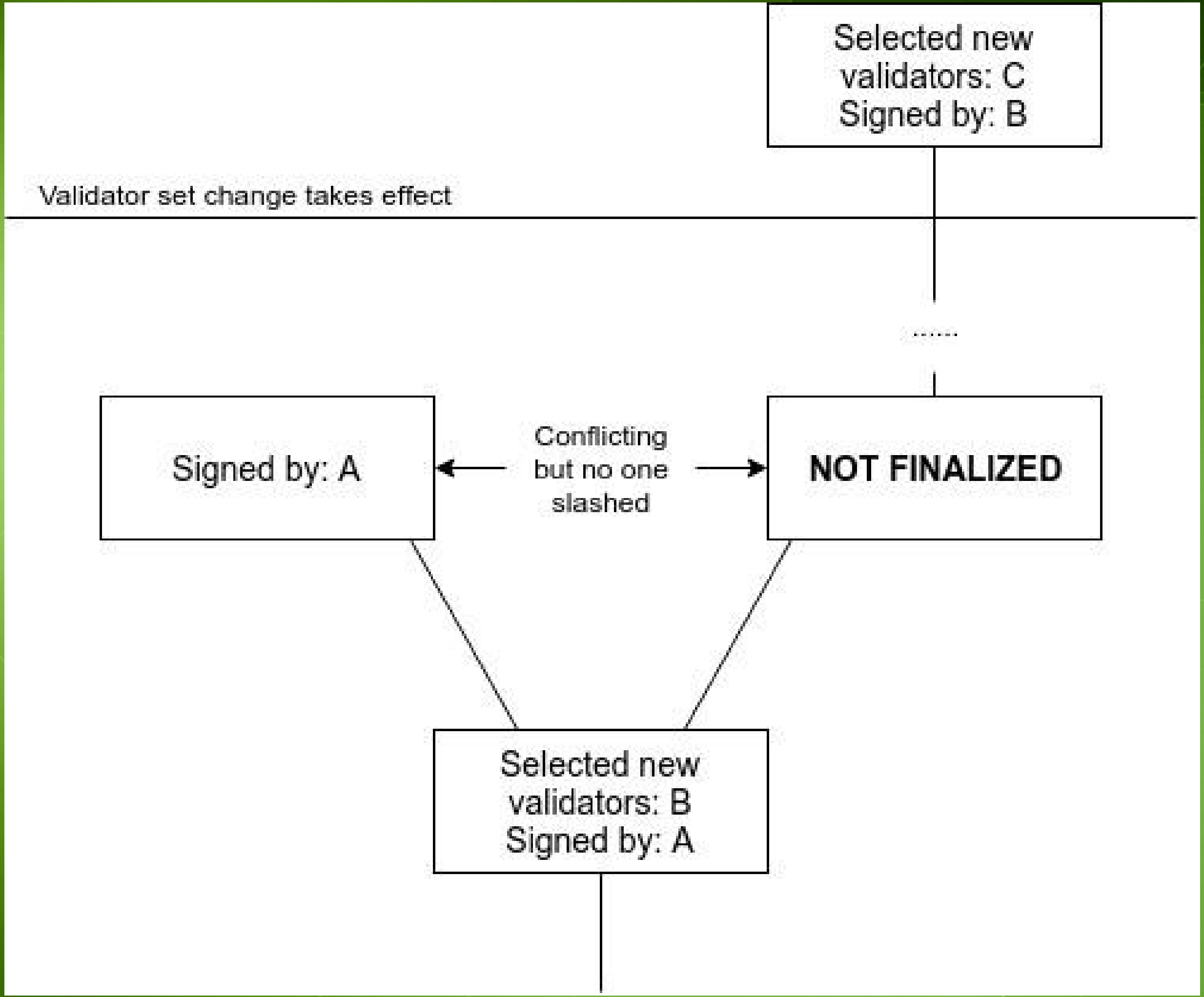
Fork choice rule

- For hybrid PoS, we can create a **hybrid fork choice rule**, roughly:
  a. Prefer finalized checkpoints
  b. Prefer checkpoints that are close to being finalized (ie. have ⅔ prepares and some commits)
  c. Otherwise, take the PoW longest chain

Dynamic validator sets

- How to deal with the possibility of the validator set changing?


New validators: D1
Signed by: C1
New validators: D2
Signed by: C2
New validators: C1
NOT FINALIZED
Conflicting
but no one
slashed
New validators: C2
NOT FINALIZED
New validators: B
Signed by: A


Selected new validators: C
Signed by: B
Validator set change takes effect
......
Signed by: A
Conflicting but no one slashed
NOT FINALIZED
Selected new validators: B
Signed by: A

Yoichi Hirai Follow
a convenience logician
Mar 18 · 4 min read

A mechanized safety proof for PoS with dynamic validators

I used a theorem prover Isabelle/HOL to check Vitalik's post about a proof-of-stake protocol that uses dynamic validator sets. (If you haven't seen, I did something similar for the simpler proof-of-stake protocol where the validator set is constant.) The proof script is available online.

Dynamic validator sets

- If a validator leaves, they need to wait 4 months before they can recover their deposit
- Waiting period ensures that validators can be held accountable for slashing condition violations

Implementation

- Casper contract (https://github.com/ethereum/casper/blob/master/casper/contracts/simple_casper.v.py)
- Casper fork choice rule (work being done on pyethereum)
- Daemon for sending prepares/commits (work being done in python; should be client-agnostic)

Cryptoeconomic analysis

(applies to any PoS)

Attacks

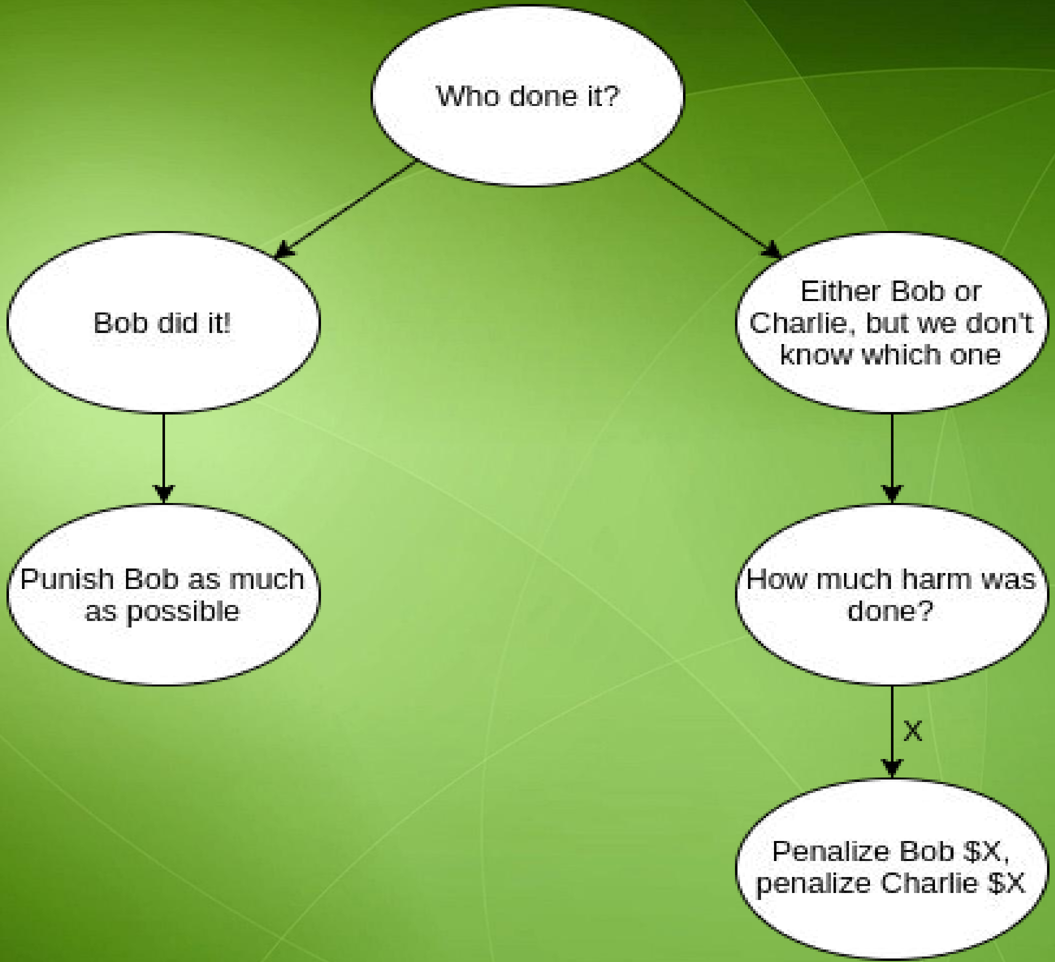
- Finality reversion attack (33%+)
- Going offline (33%+)
- Censorship (51%+)
- Extortion attacks (51%+)
- Discouragement attacks (any %)

Fault categorization

- Outbound message omission
  - “I’m not going to vote”
- Inbound message omission
  - “Vote? What vote? I didn’t hear about a vote”
- Equivocation
  - “I vote A!” “I also vote B!”
- Invalid message faults
  - “I vote Harambe!”

Fault categorization

- Uniquely attributable faults
- Non-uniquely attributable faults


Who done it?
Bob did it!
Either Bob or Charlie, but we don't know which one
Punish Bob as much as possible
How much harm was done?
X
Penalize Bob $X, penalize Charlie $X

Liveness is a
non-uniquely-attributable fault


La, la, la, la la...
I can't HEAR you!!!

Incentivization vs Niceness

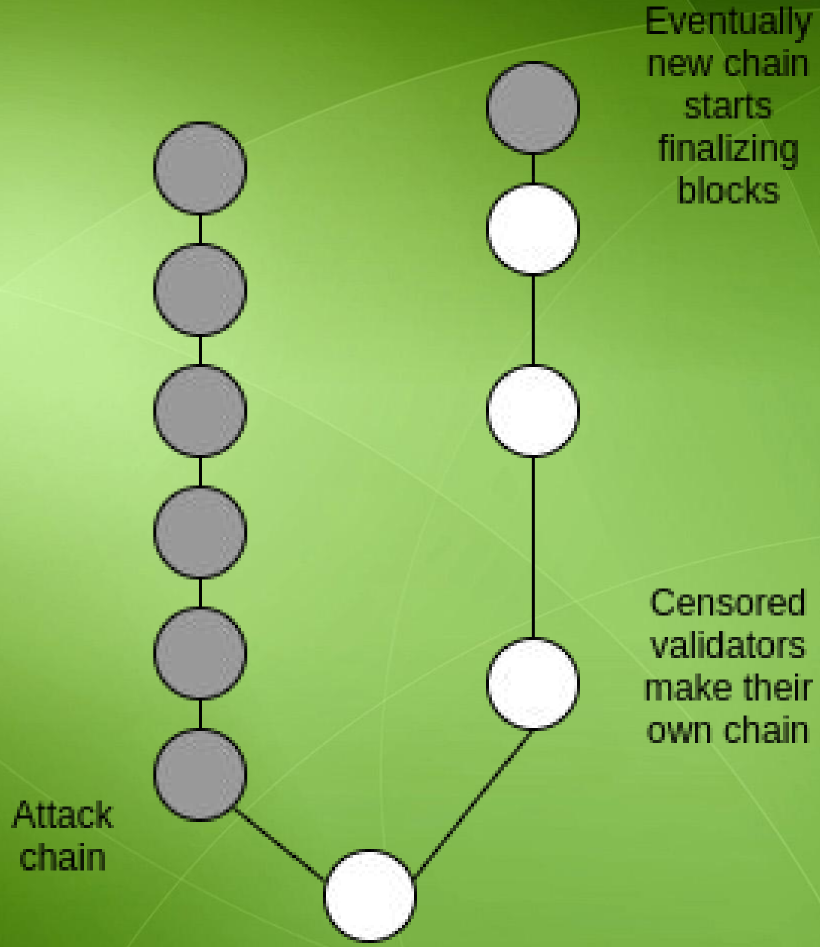
- Penalize attackers: good
- Penalize innocent validators: bad
  - (because otherwise no one will want to validate due to risk)
- This is a tradeoff

Subjective resolution

- If a serious attack happens, we can always resolve it subjectively (UASF, UAHF, etc)
    - Treating the community as a magic oracle that tells us who was the attacker
- But we want to use this as little as possible, as it harms decentralization

Dealing with censorship


UASF


Eventually new chain starts finalizing blocks
Censored validators make their own chain
Attack chain

Future directions

Future directions

- Continuing to improve incentive structures
- Find a mechanism that combines together benefits of chain-based and BFT-based PoS
- Scalability (sharding etc)
- Speed up block times