Blockchains & Cryptocurrencies

Alternative Consensus Techniques II

Instructor: Matthew Green & Abhishek Jain
Spring 2023

Sponsored by:

Image source: https://eprint.iacr.org/2021/911.pdf

News?

News?

Matthew Green
@matthew_d_green

If your project has any crypto(currency) T-shirts you want to give away, please send them to me at: 3400 N. Charles St, Malone 160, Baltimore, MD 21218. You'll get priceless mindshare from literally a hundred Hopkins students.

8:55 PM · Feb 10, 2023 · **32.7K** Views

6 Retweets **2** Quotes **40** Likes **4** Bookmarks

News?

Matthew Green
@matthew_d_green

So a giant box just showed up in the CS department mailroom addressed to me.

News?

Matthew Green @matthew_d_green · Mar 28

Replying to @matthew_d_green

It’s from Vietnam and the packing list says “shirt.” Uh oh.

4 1 29 5,615

News?

Matthew Green @matthew_d_green · Mar 28

I salute you @solana team.

6 8 124 13.3K

Today

- Continuing with our tour of consensus technologies
  - Tuesday: we talked about BFT consensus (e.g., Tendermint) and proof-of-stake
  - Today we will talk about alternative consensus techniques:
    - Snowflake-to-avalanche (based on gossip protocols and consensus collapse)
    - Ethereum Gasper

Review: consensus (definition)

- *English:* Finding an acceptable proposal that all members can support

Review: consensus (definition)

- *English:* Finding an acceptable proposal that all members can support

- *Computer science fault-tolerant consensus:*

  - Agreement among processes (or agents) on a single data value, where some of the processes may fail or be unreliable in other way. Requirements include:

    - Termination

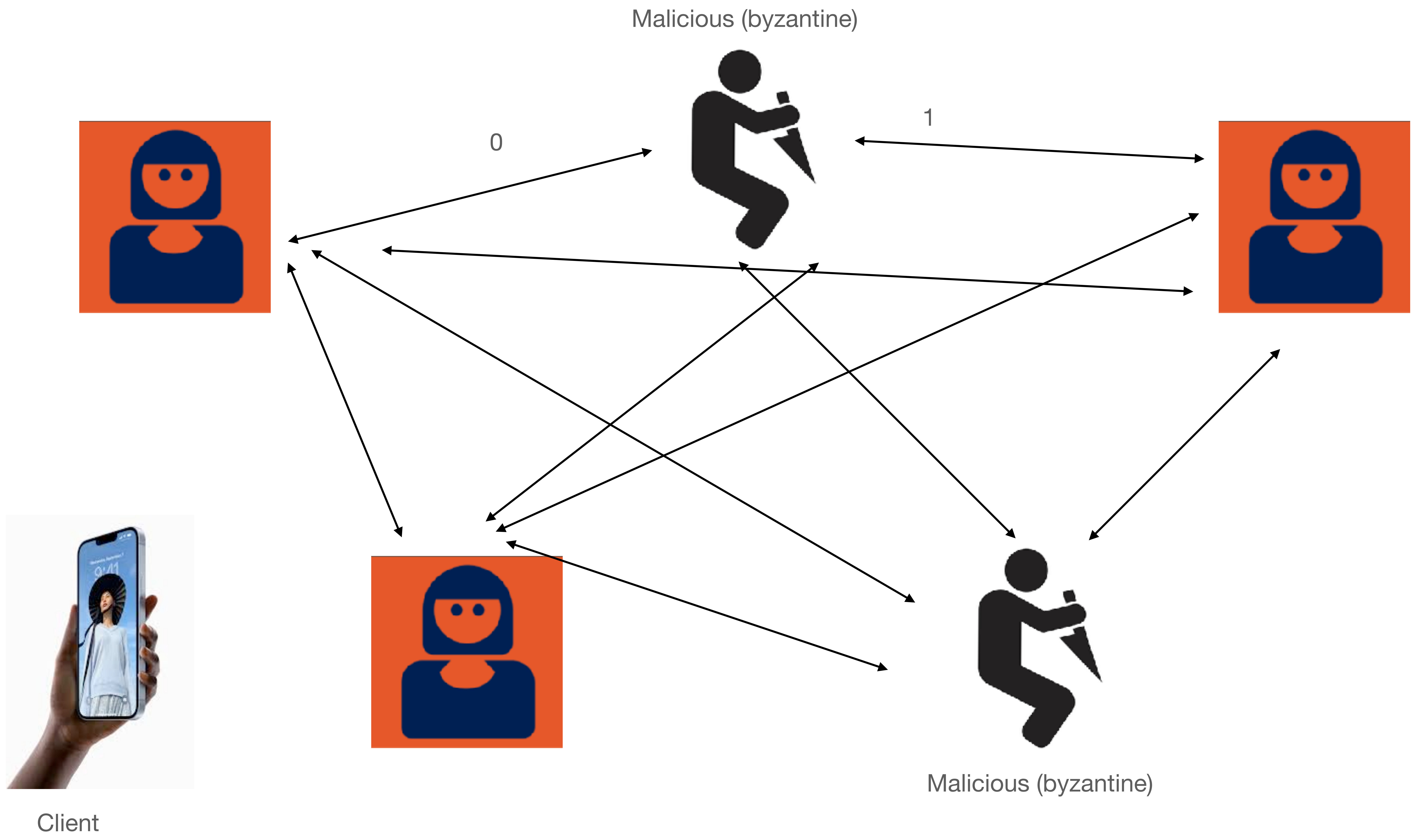
    - Integrity/validity

    - Agreement

Properties of a consensus protocol

- **Agreement:** All correct processes must agree on the same value.
- **Weak validity:** For each correct process, its output must be the input of some correct process.
- **Strong validity:** If all correct processes receive the same input value, then they must all output that value.
- **Termination:** All processes must eventually decide on an output value

Review: classical BFT



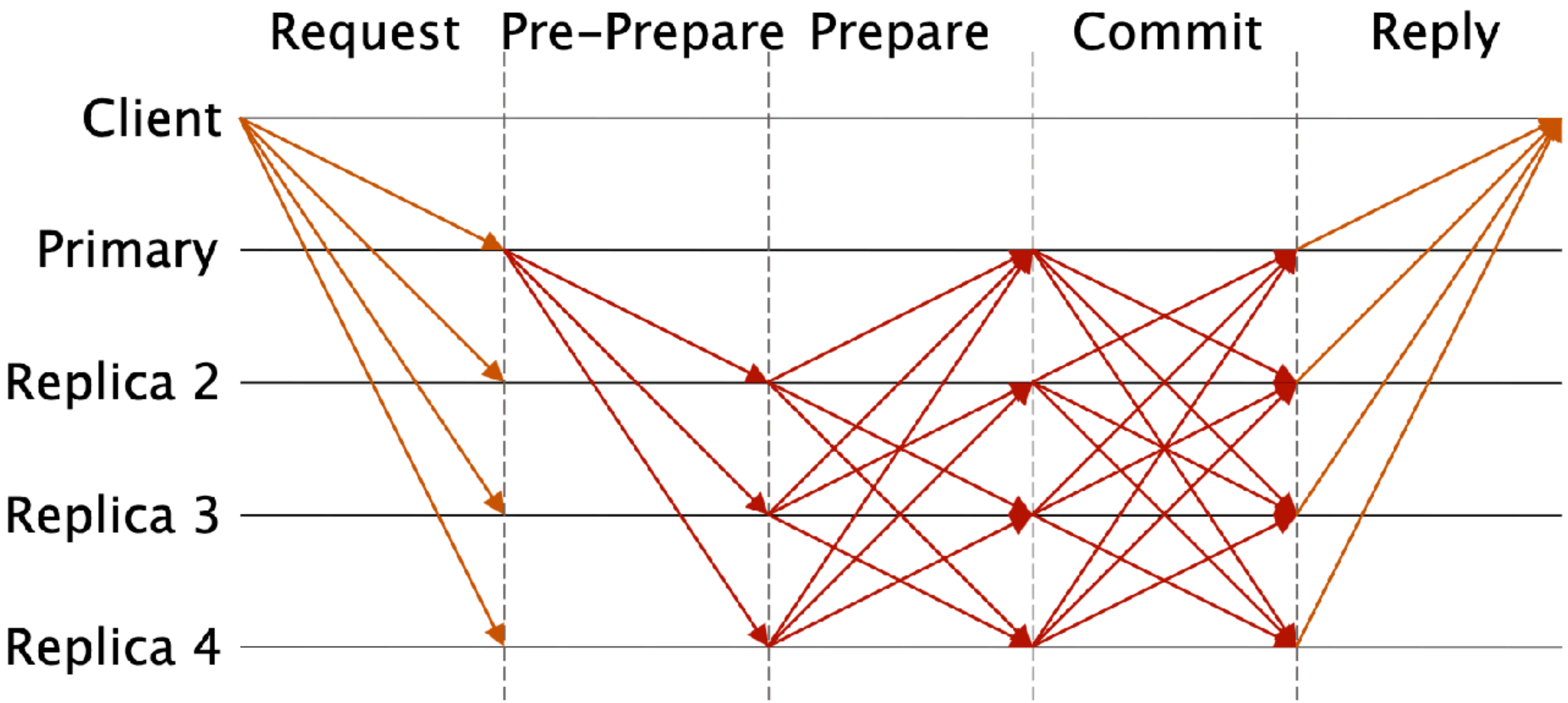
Synchronous vs. asynchronous

- Synchronous protocols:
  - Messages take place in "rounds": all messages are sent and received within a single round
- Asynchronous protocols (AKA the real world):
  - Messages can have various network delays and won't always arrive within rounds
  - There are impossibility results here!
  - In practice we can use timeouts to deal with these (timeout makes delayed message equivalent to a "crash")

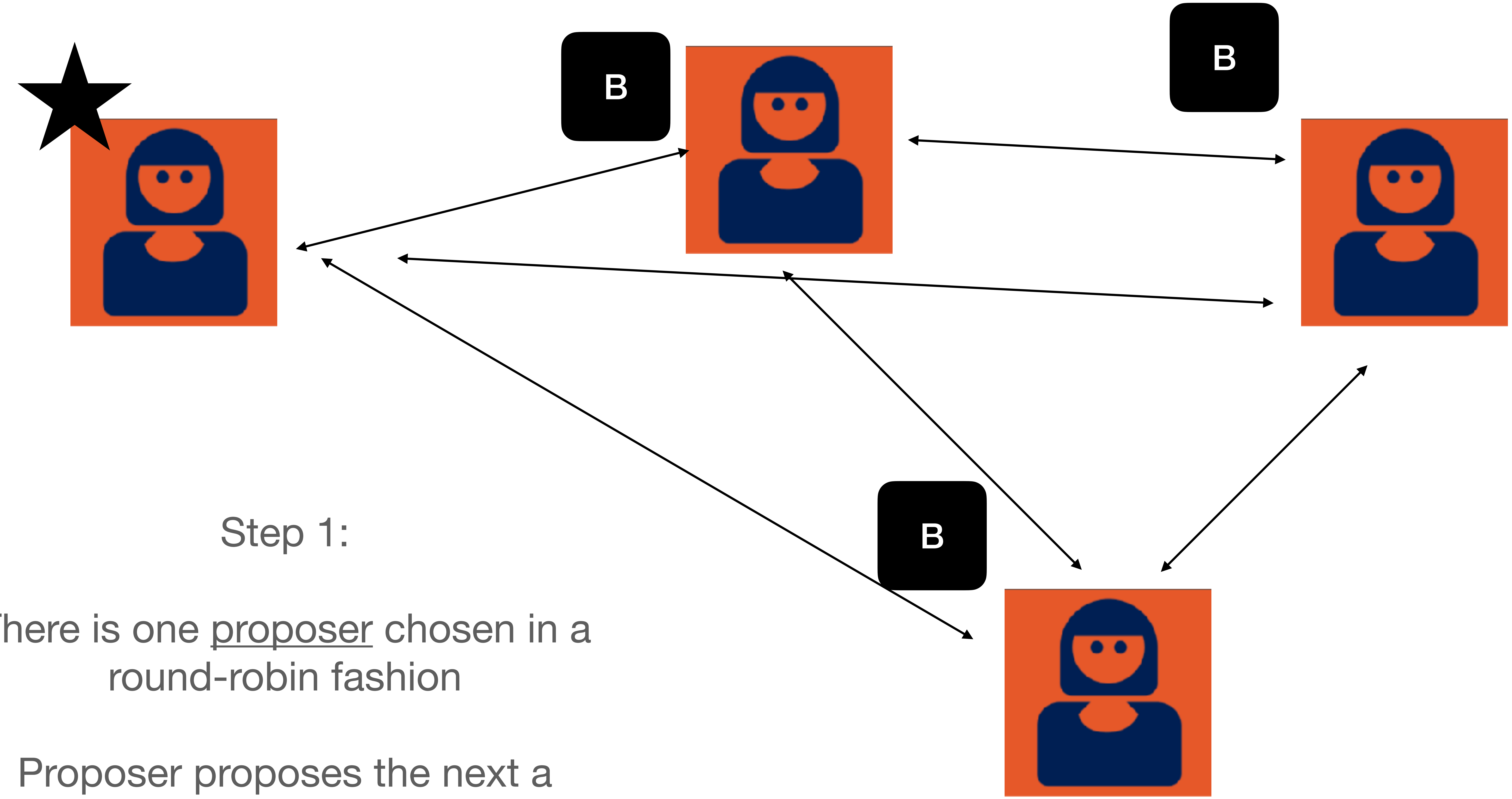
Selecting validators (PoS)

- **Proof of stake (PoS)**
  - Validators must possess (and lock up) a large amount of on-chain currency. Keys come from associated wallets. *Examples*: Ethereum, Avalanche, Cardano, etc. Participants change periodically.
- *There are other approaches (we mentioned last time) but this is where we will focus things today*

Byzantine Fault Tolerance (BFT)



Tendermint: proposal

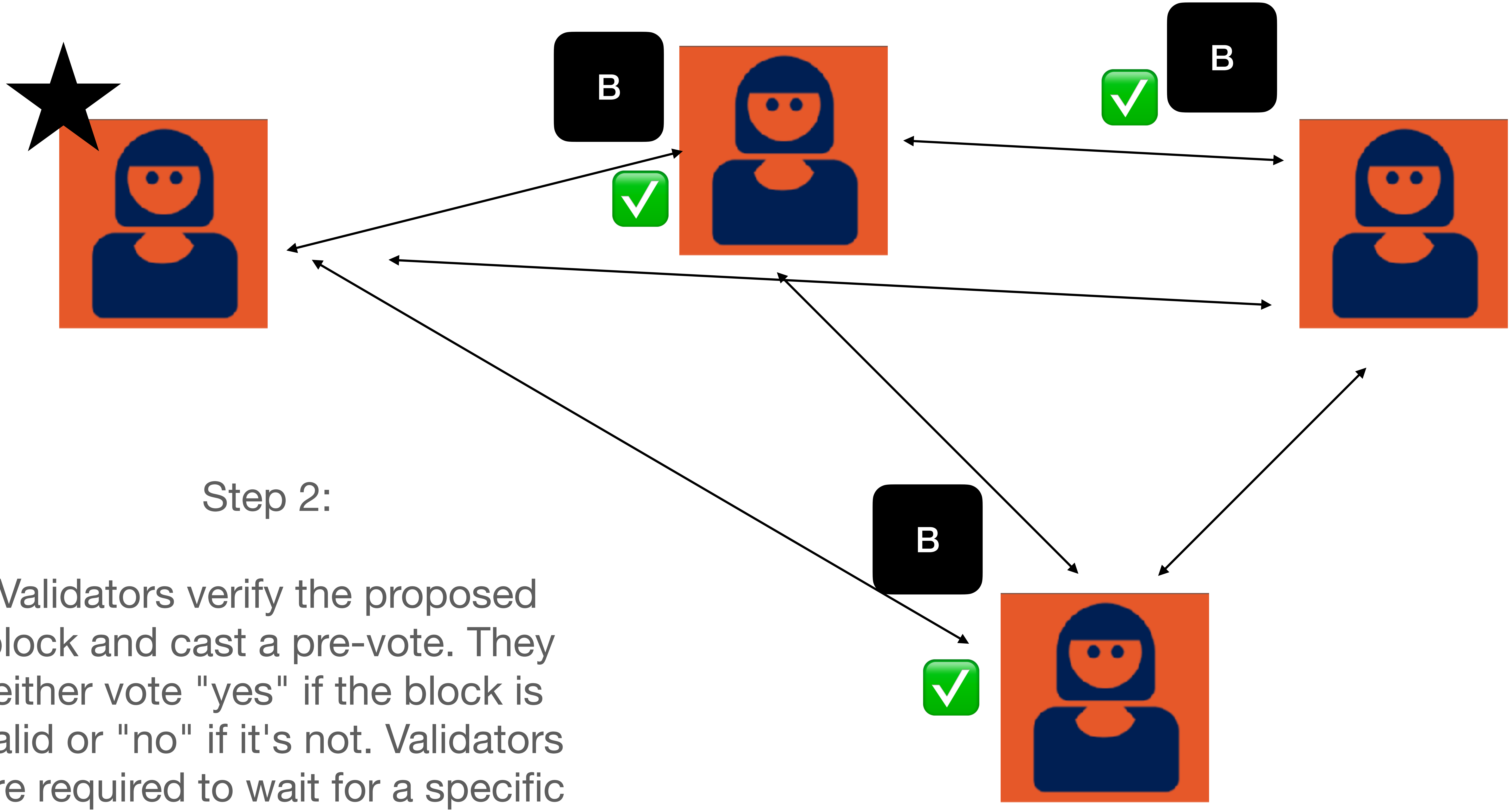


Step 1:

There is one proposer chosen in a round-robin fashion

Proposer proposes the next a block and sends it to all other participants

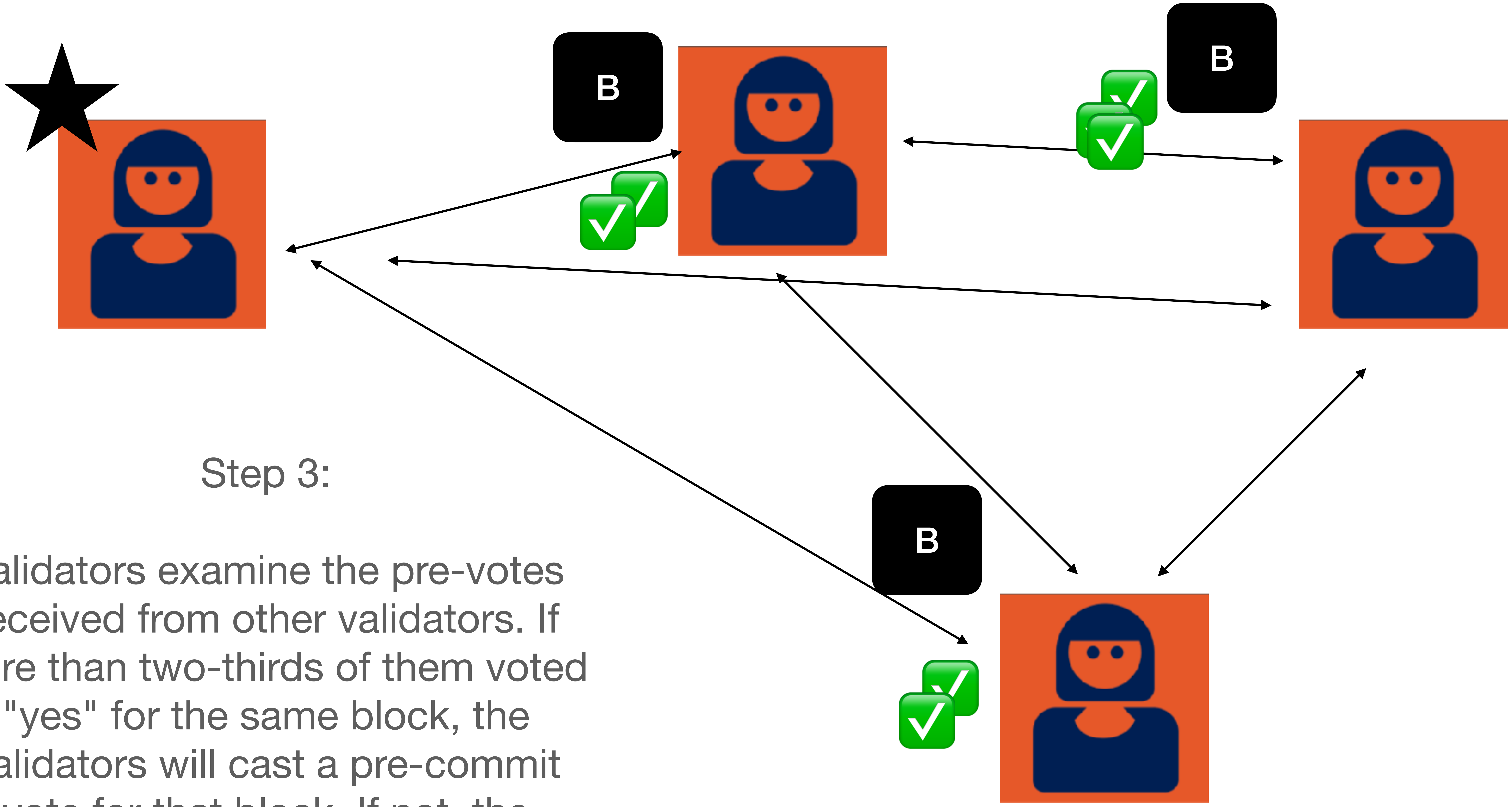
Tendermint: pre-vote



Step 2:

Validators verify the proposed block and cast a pre-vote. They either vote "yes" if the block is valid or "no" if it's not. Validators are required to wait for a specific time before voting to prevent premature decisions.

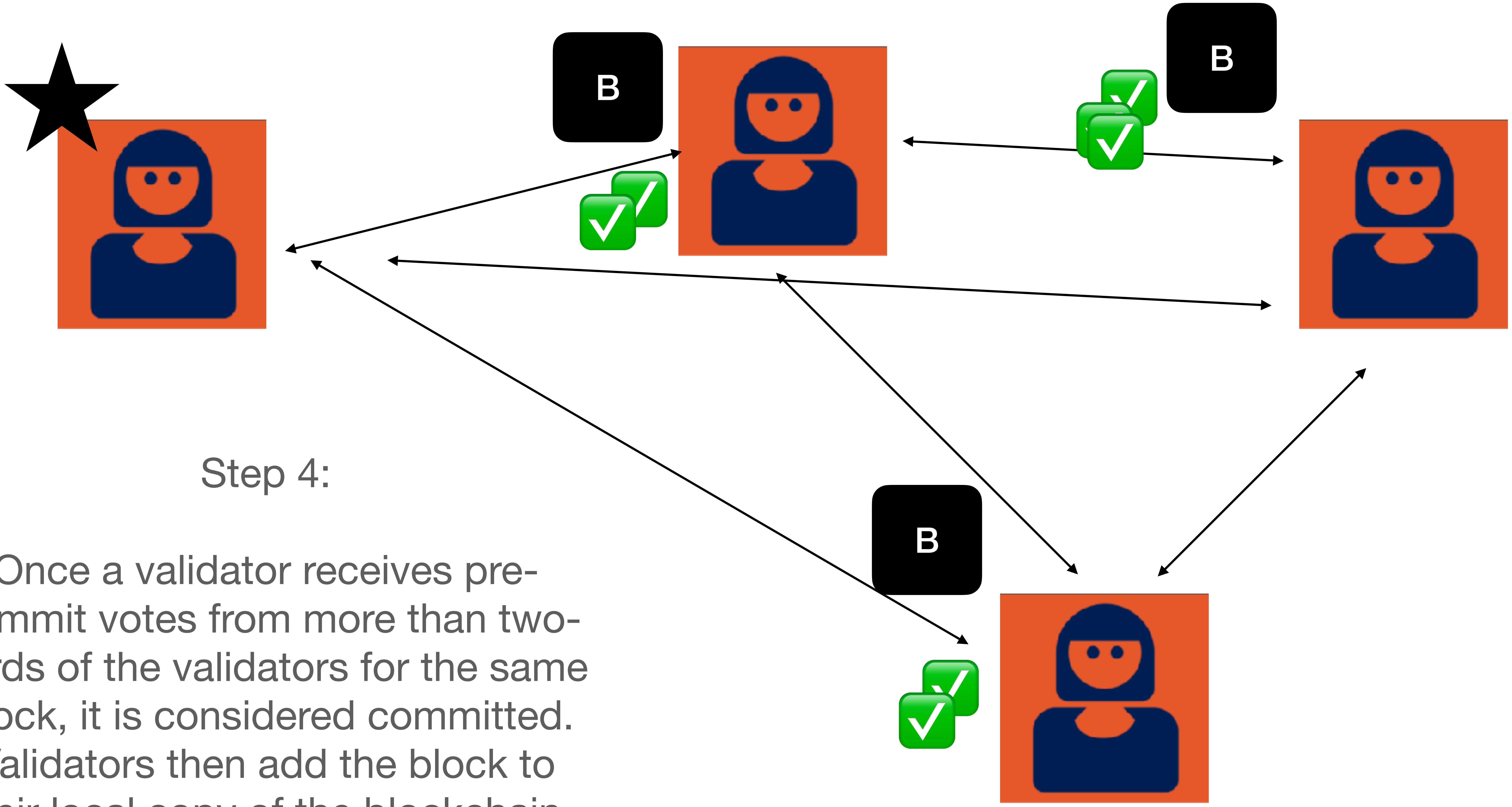
Tendermint: pre-commit



Step 3:

Validators examine the pre-votes received from other validators. If more than two-thirds of them voted "yes" for the same block, the validators will cast a pre-commit vote for that block. If not, the process moves to the next round with a new proposer.

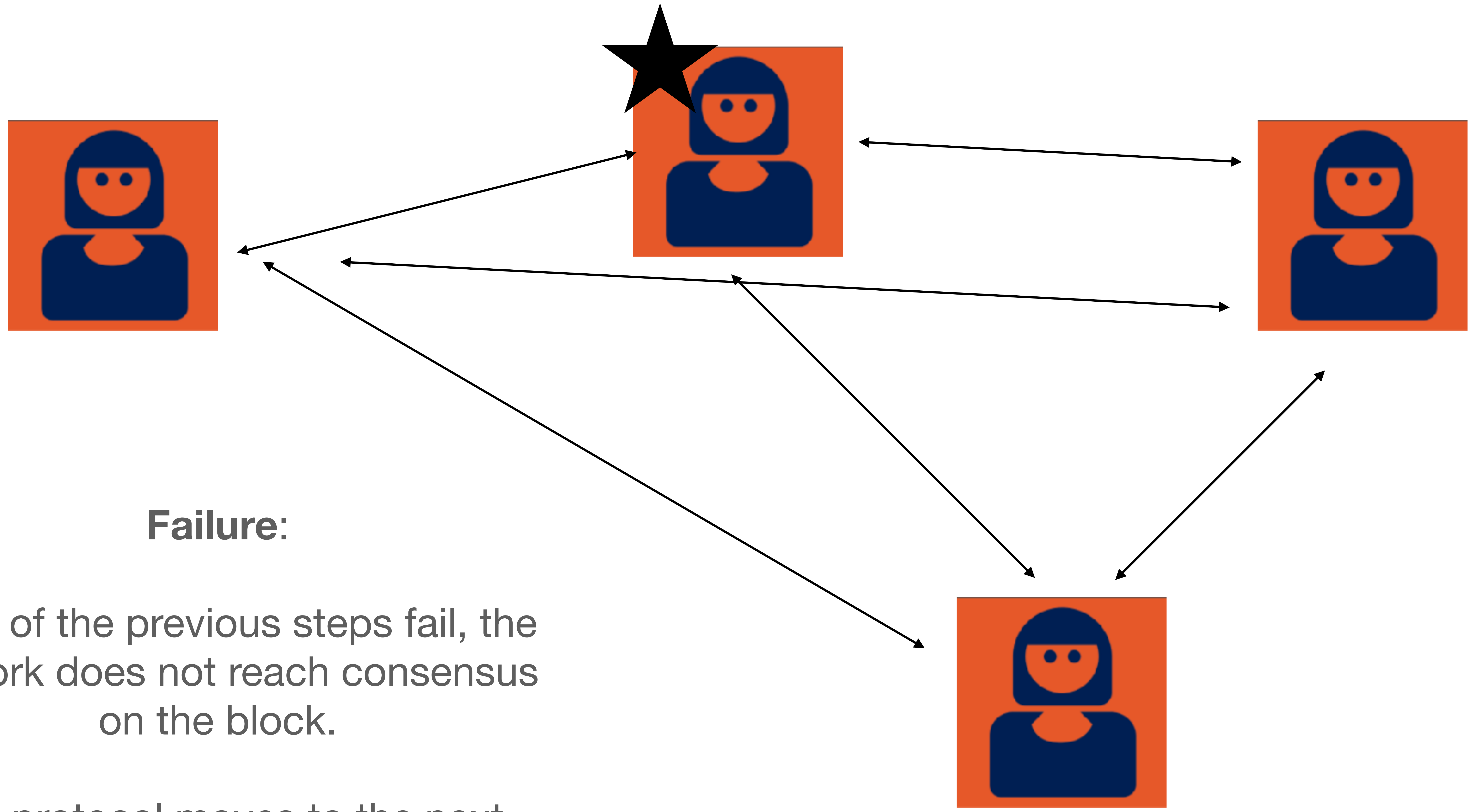
Tendermint: commit



Step 4:

Once a validator receives pre-commit votes from more than two-thirds of the validators for the same block, it is considered committed. Validators then add the block to their local copy of the blockchain, and the process starts again with a new proposal.

Tendermint: failure cases



Failure:

If any of the previous steps fail, the network does not reach consensus on the block.

The protocol moves to the next round with a new proposer.

Algorand

- Proposed/launched by Silvio Micali ~2017
  - Based on a <u>single-round BFT</u> protocol
    - The idea here is that once a node broadcasts, someone might hack into it! So you want to broadcast quickly and then go away.
    - Because the protocol is single-round, you need a way to decide who is the "leader" quickly
    - This is done through cryptographic sortition

Algorand

- Proposed/launched by Silvio Micali ~2017
  - Based on a single-round BFT protocol
    - Idea: have thousands of validators, elect a small sub-committee each round, have them do a "quick" BFT
    - This BFT should be extremely rapid, and not interactive
    - The main challenge here is selecting the committee
    - This is done through a cryptographic sortition lottery, based on stake

VRFs and "sortition"

- Remember in Bitcoin, we used a "lottery"
  - Any node could solve a PoW puzzle and broadcast that solution to the network (bound to a block)
  - Only the winning node (or nodes) ever have to broadcast!
  - If the block is valid, each honest node will accept the first solution they see
  - (The gnarly cases are when two valid solutions arrive at different parts of the network)

Idea: replace the PoW lottery

- Assumptions:
  - We have already agreed on a blockchain of length T-1
  - Within that blockchain, N validators have "staked" some funds associated with their <u>public key</u>
  - <u>The amount of staked funds may be different across each node!</u>

Algorand™

Protocol: initial condition

sk1, pk1

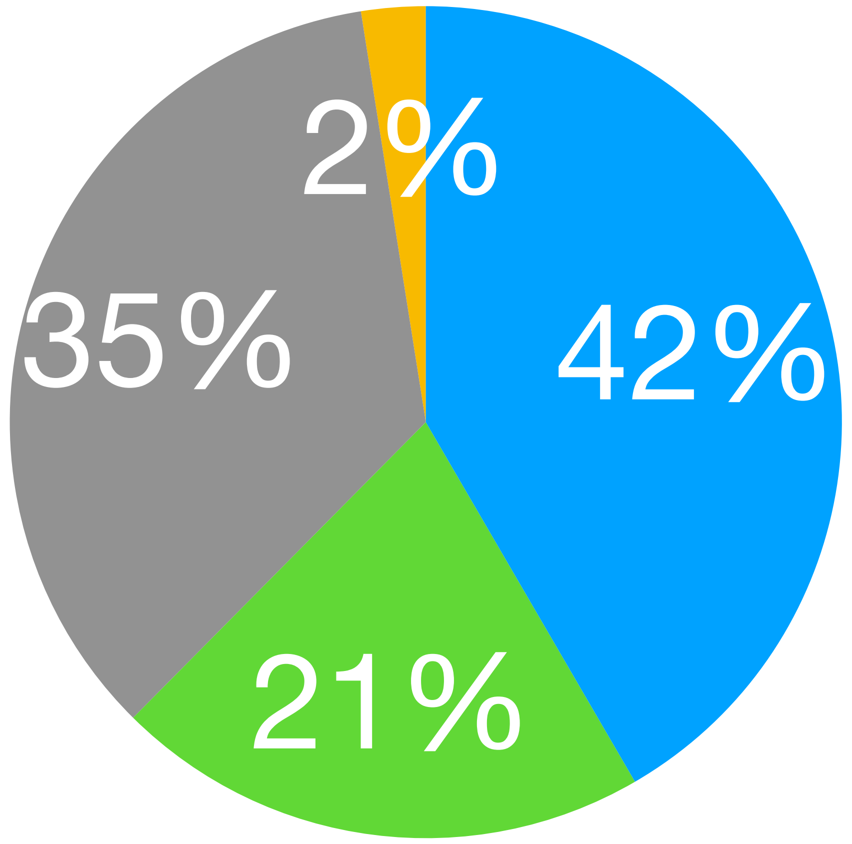
stake: 1000

sk2, pk2

stake: 500

sk3, pk3

stake: 843



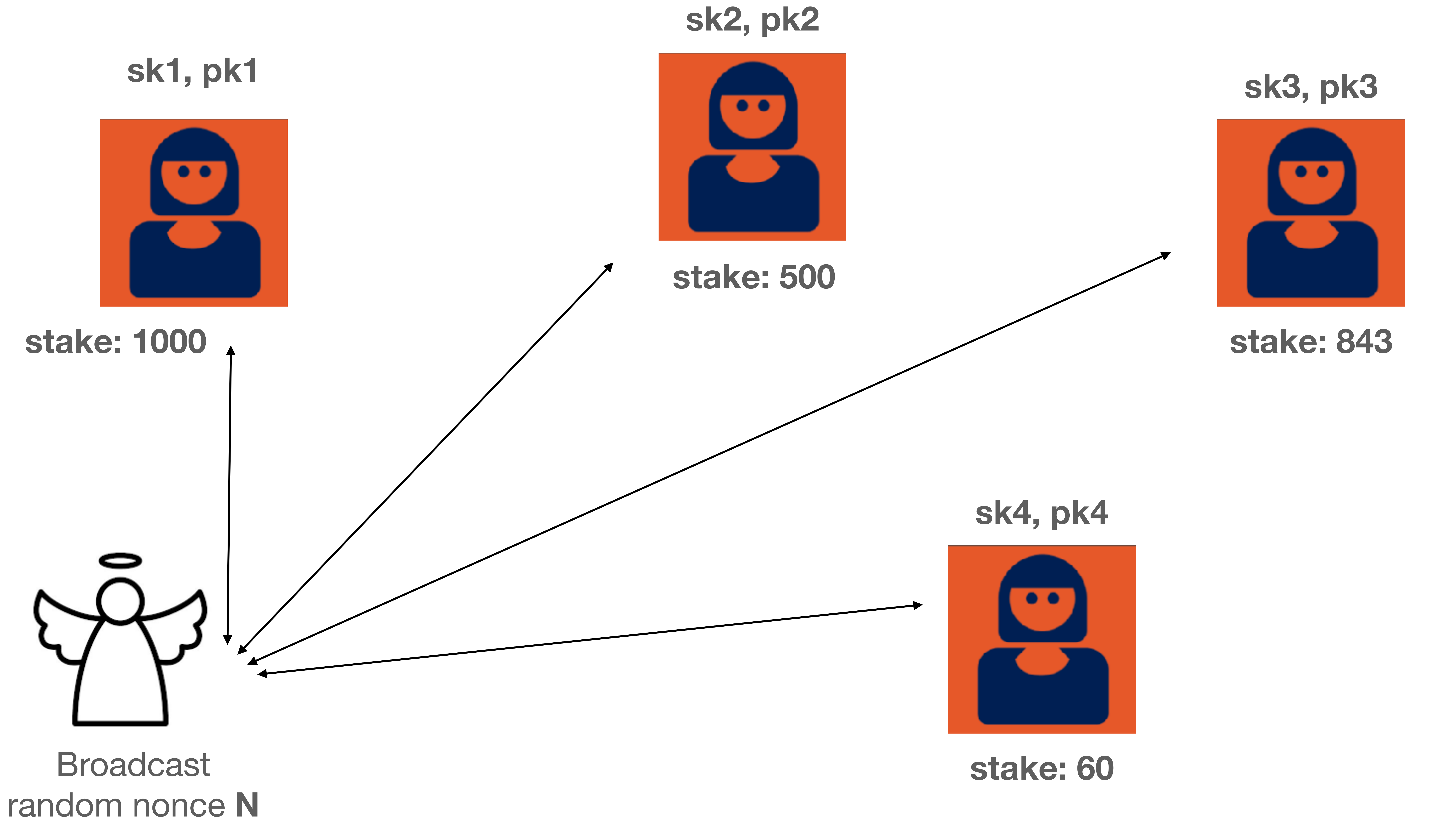
sk4, pk4

stake: 60

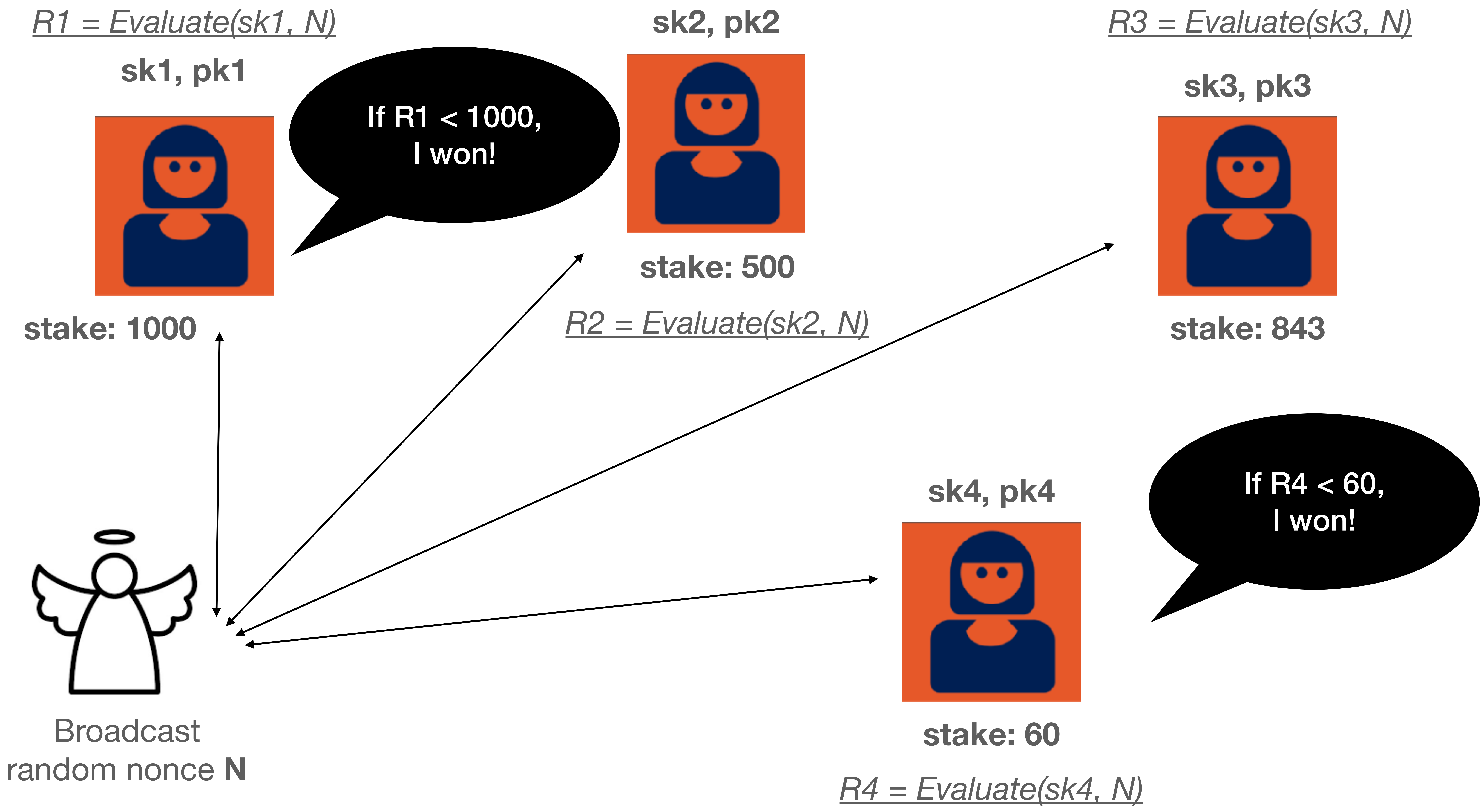
Idea: replace the PoW lottery

- We need a cryptographic tool: <u>Verifiable Random Functions</u>
  - A VRF has three algorithms:
    Setup, **Keygen**, **Evaluate**, **Verify**
  - *Optional:* Setup() generates global parameters (*params*)
  - Keygen(): generates a user's public/secret keypair (*pk*, *sk*)
  - Evaluate(sk, message): Produces a <u>pseudorandom</u> output **R** and a proof **π**
  - Verify(pk, R, **π**): determines if this value is correct

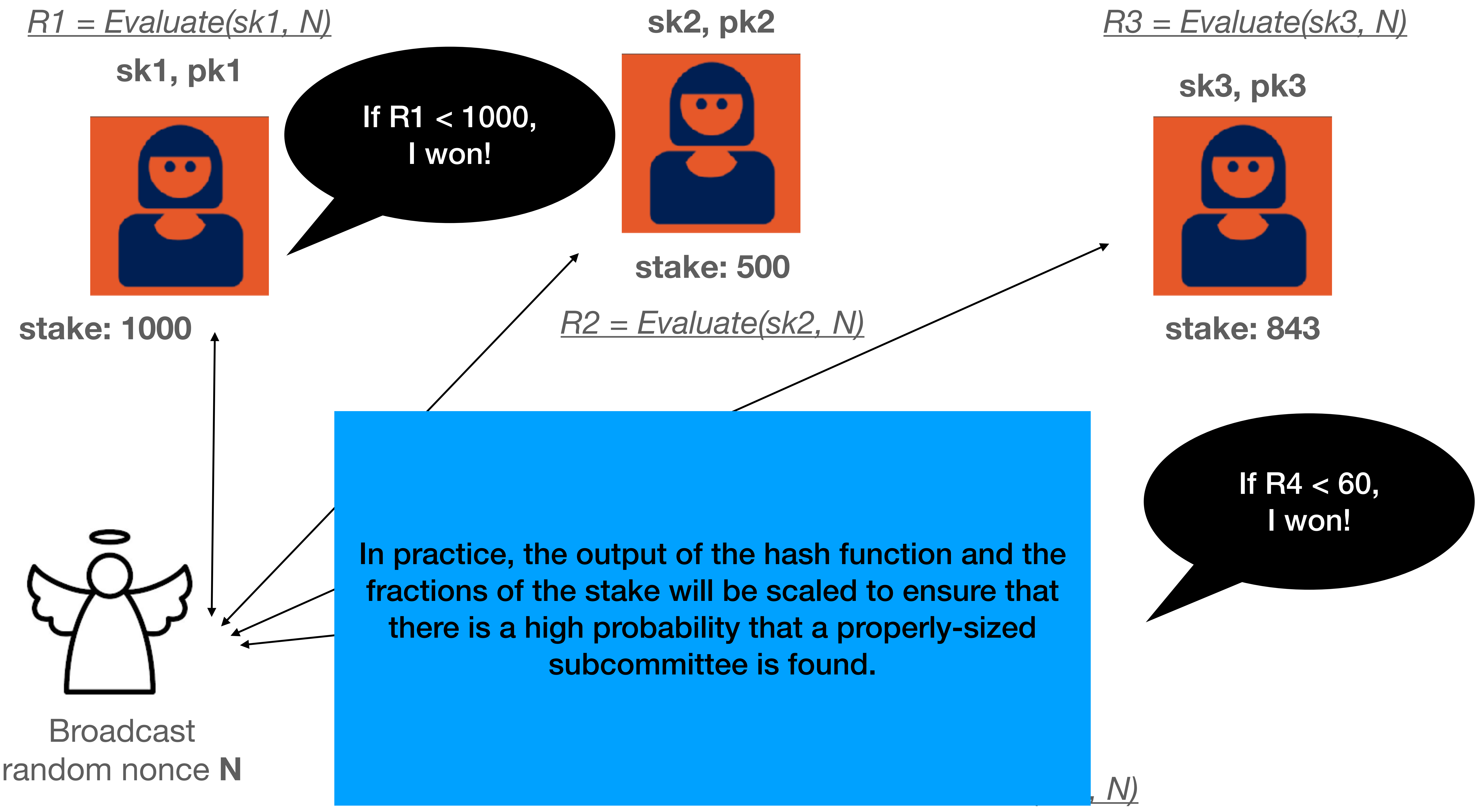
Protocol: step 1



Protocol: step 2

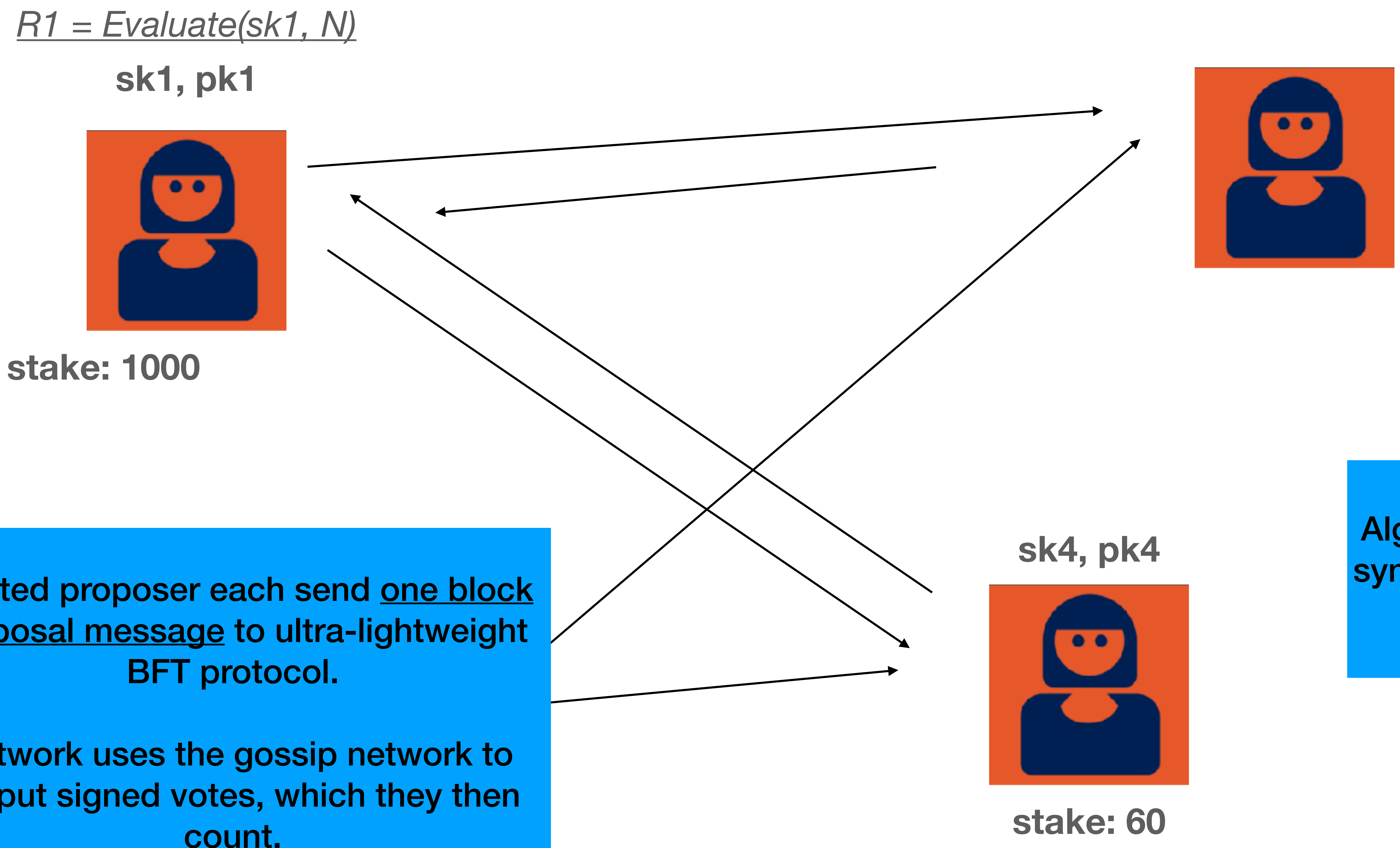


Protocol: step 2



In practice, the output of the hash function and the fractions of the stake will be scaled to ensure that there is a high probability that a properly-sized subcommittee is found.

Protocol: step 3



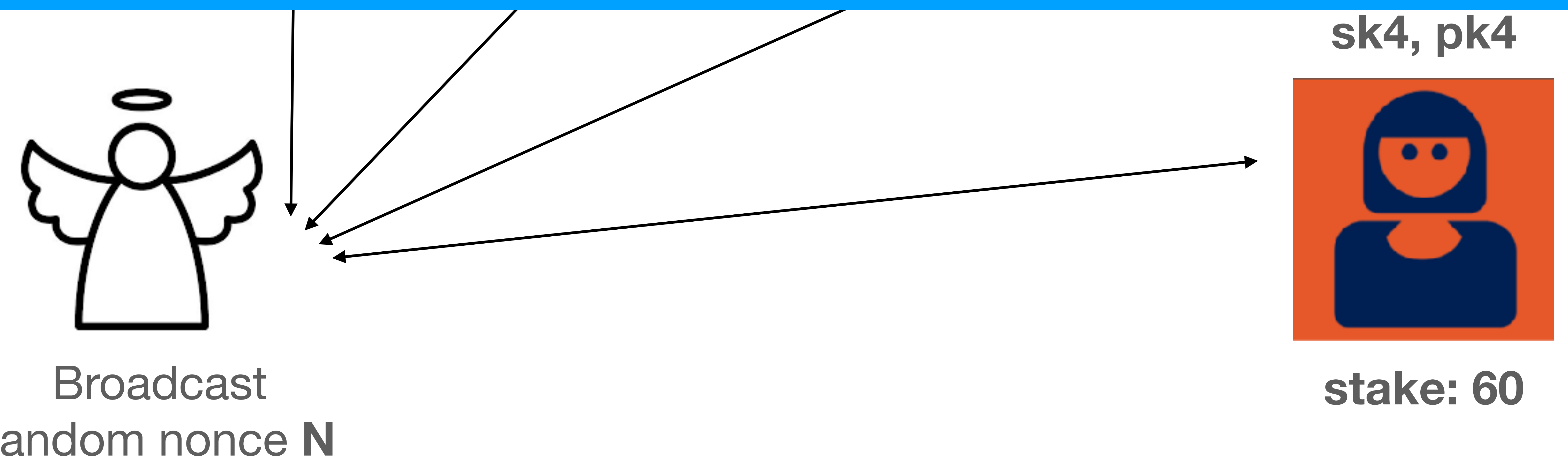
Elected proposer each send <u>one block proposal message</u> to ultra-lightweight BFT protocol.

Network uses the gossip network to output signed votes, which they then count.

Algorand uses loosely synchronized clocks to detect timeouts.

Where does the random nonce come from?

Note: if you can pick reliable random numbers, committee election is much easier!



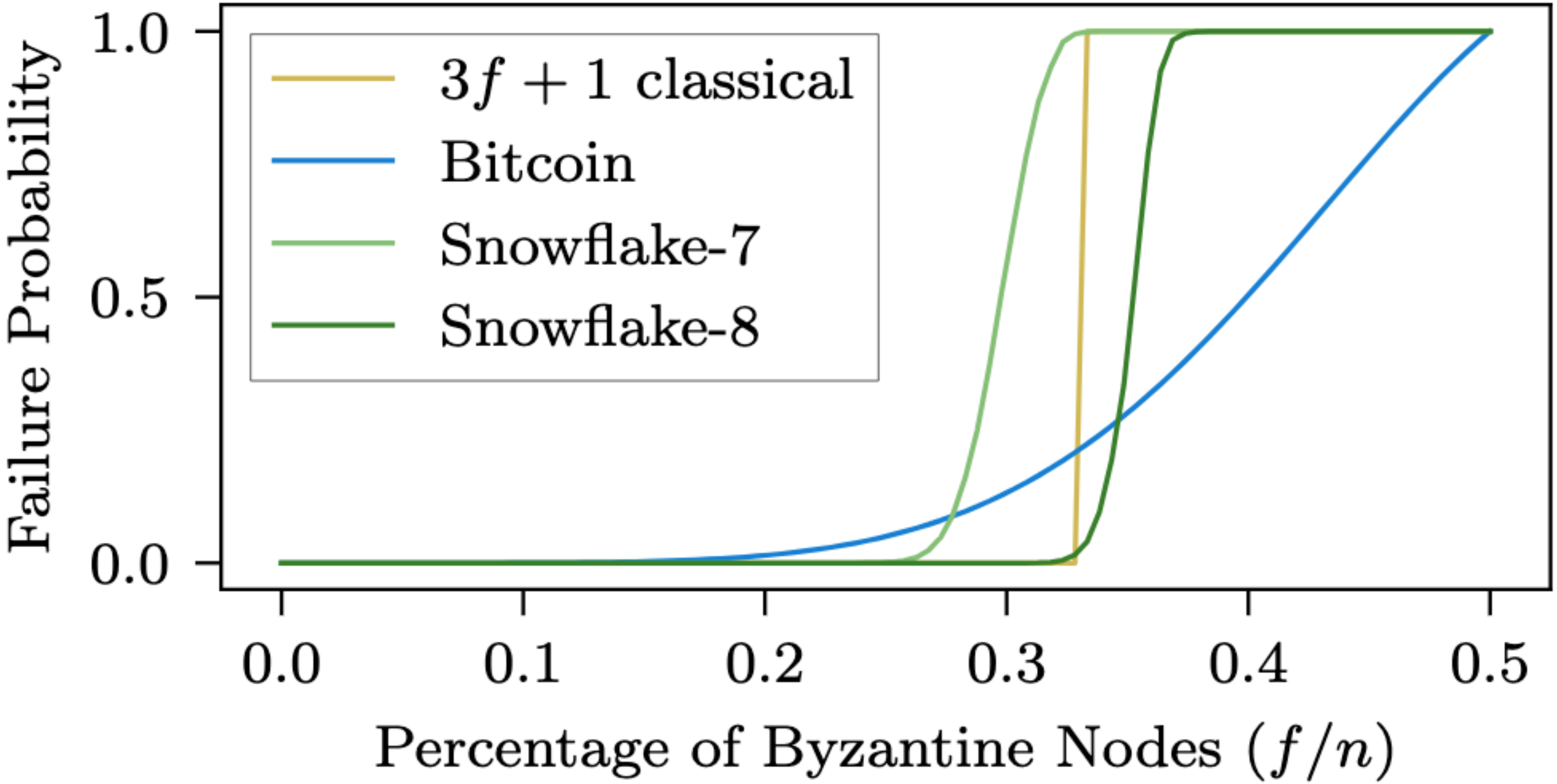
Avalanche

- Uses a different and probabilistic approach
  - Unlike Nakamoto consensus, does not rely on lotteries (I.e., randomness at the proposer side)
  - Instead, nodes communicate via gossip network and poll each other

Snowflake protocol

- Basic idea (for one node)
  - Node has a current consensus decision (e.g., "Red")
  - It selects a **random** subset of network participants and polls them for their preferences on the consensus decision (e.g., who prefers "Red", who prefers "Blue")
    - If more than some fraction (*a*) vote for a different color, the node flips its decision to that color
    - Repeats this for *m* total rounds
    - Each time it arrives at a specific color, its <u>confidence</u> increases

Snowflake protocol



Source: Avalanche White Paper

Avalanche

- Does not use a blockchain, uses a transaction DAG

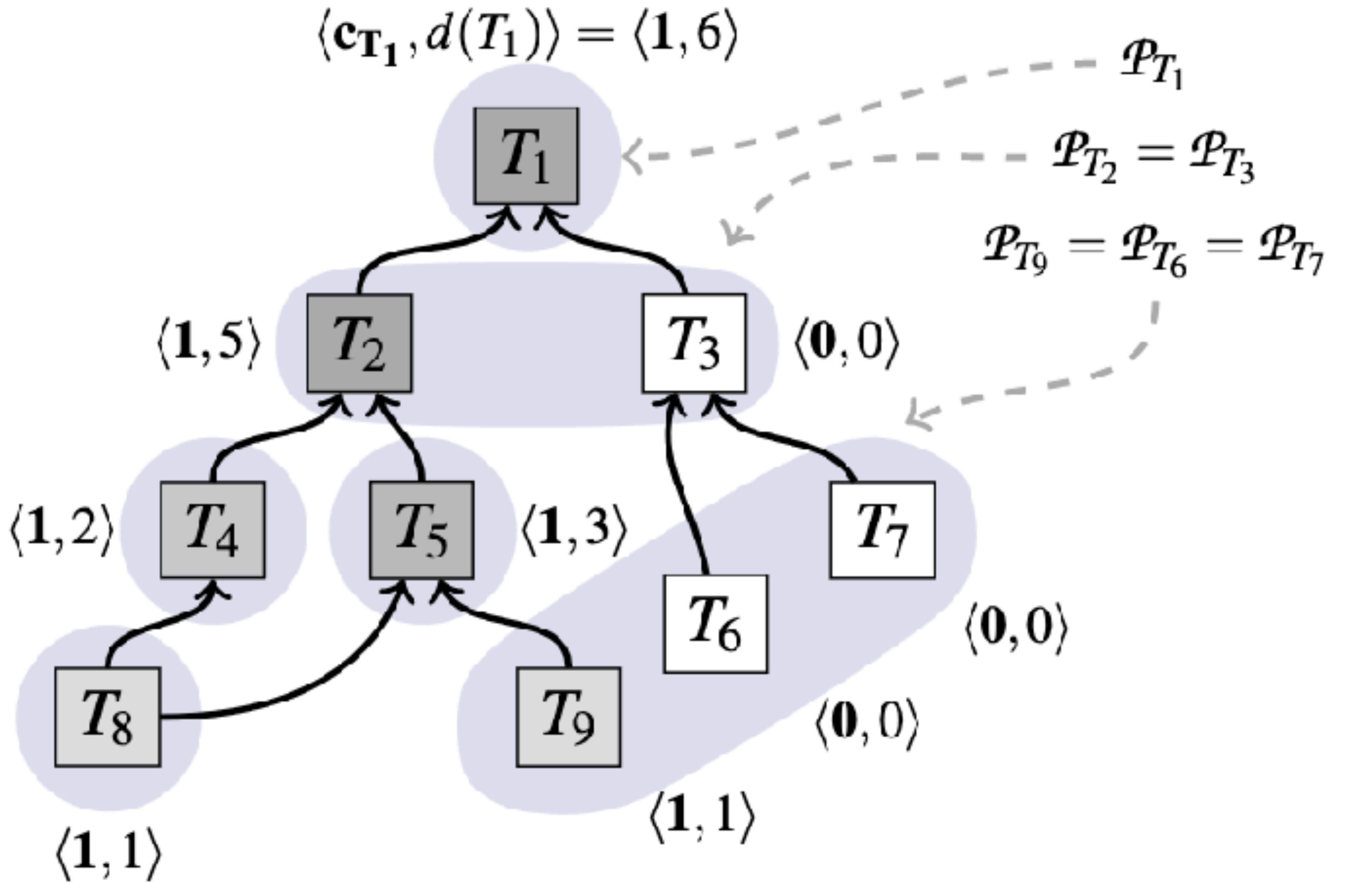


Figure 11: Example of $\langle$chit, confidence$\rangle$ values. Darker boxes indicate transactions with higher confidence values. At most one transaction in each shaded region will be accepted.

Ethereum

- Ethereum launched in 2014 using Nakamoto consensus
  - At the time, it was expected that PoS would be deployed
  - Ethereum incentivized upgrades by adding "ice ages" to reduce mining rewards and slow down block production
  - For many years, the PoW code was simply upgraded and ice ages were deferred

Ethereum -> PoS

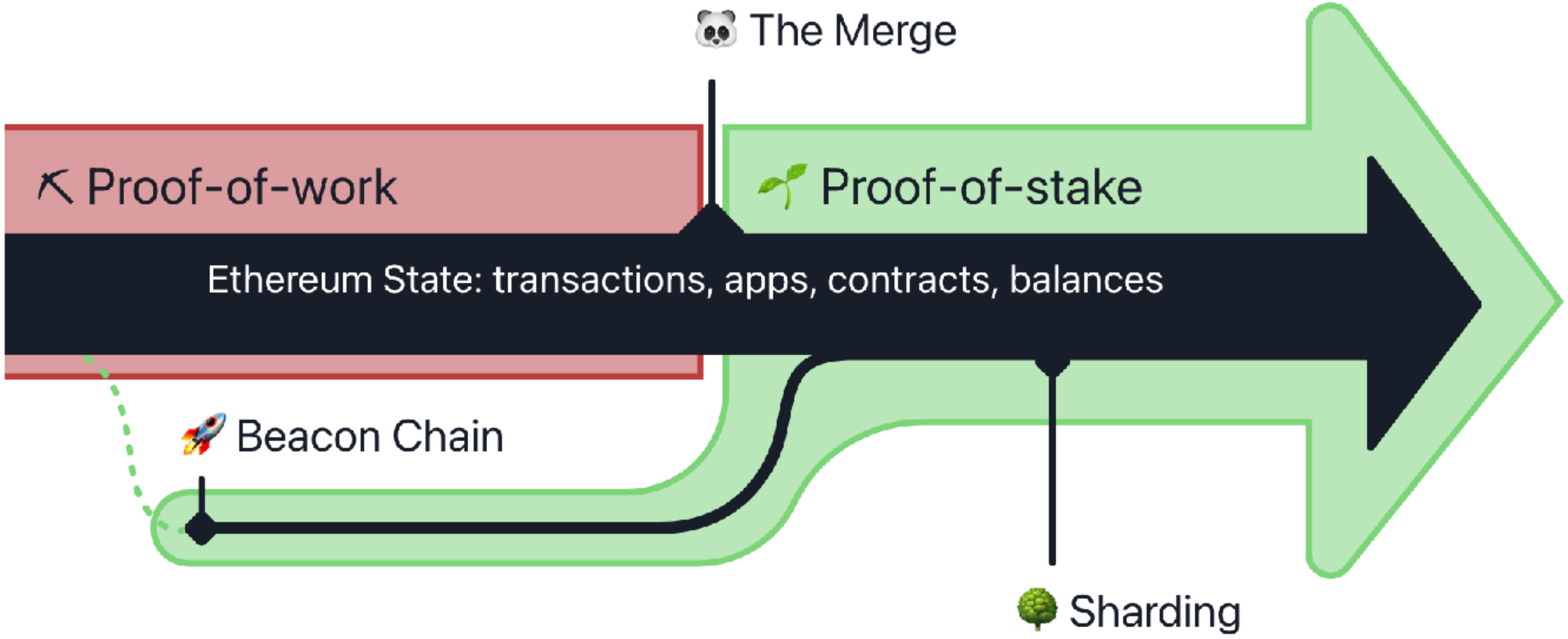
- Ethereum's upgrade process needed to be devised on a running chain. This was broken into several pieces:
  - **Staking**: deploy a smart contract on Ethereum to accept staking for validators
  - **Beacon chain:** deploy a new PoS chain that runs *parallel* to Ethereum and <u>simply generates randomness</u>
  - **Consensus:** use the randomness to implement a consensus protocol among validators for selecting blocks
  - **Docking:** update main Ethereum code to accept blocks based on Consensus protocol
  - **Unstaking:** allow stakes to withdraw their money

Validators

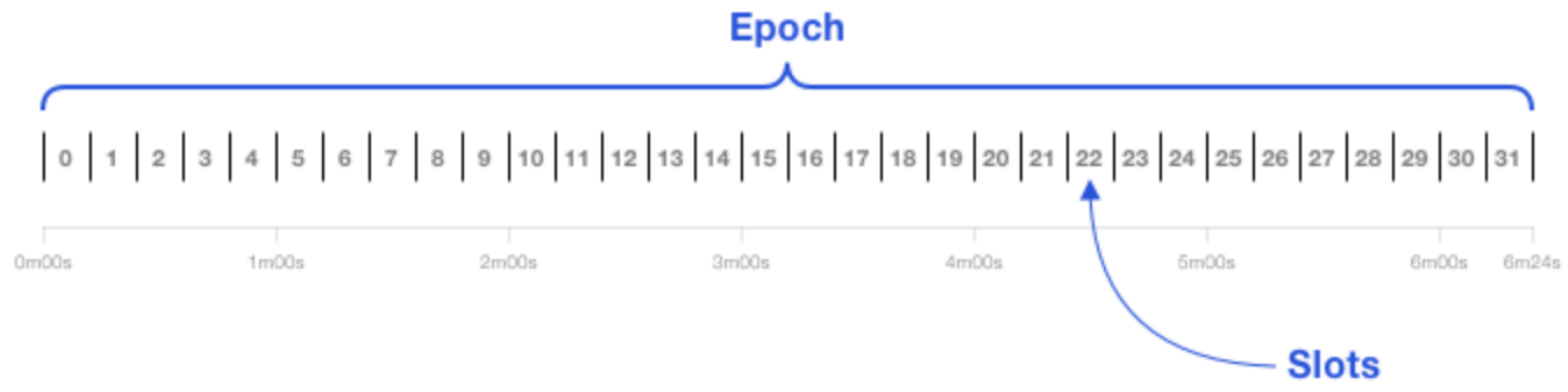
- Validators must submit 32 ETH on the existing PoW chain
  - A limited number of validators are allowed, and may be waitlisted
  - Each validator binds its public key to a record on the chain
  - Validators communicate via a P2P network
    - <u>Block proposal</u>: propose a new block
    - <u>Block voting</u>: send signed votes on each (set of 32) blocks
  - Validators are (pseudorandomly) assigned to <u>committees</u>

Beacon chain

- This is an extremely simple chain that ran in parallel to the main chain
  - It did not handle Ethereum transactions!
  - It handled randomness and validator balances



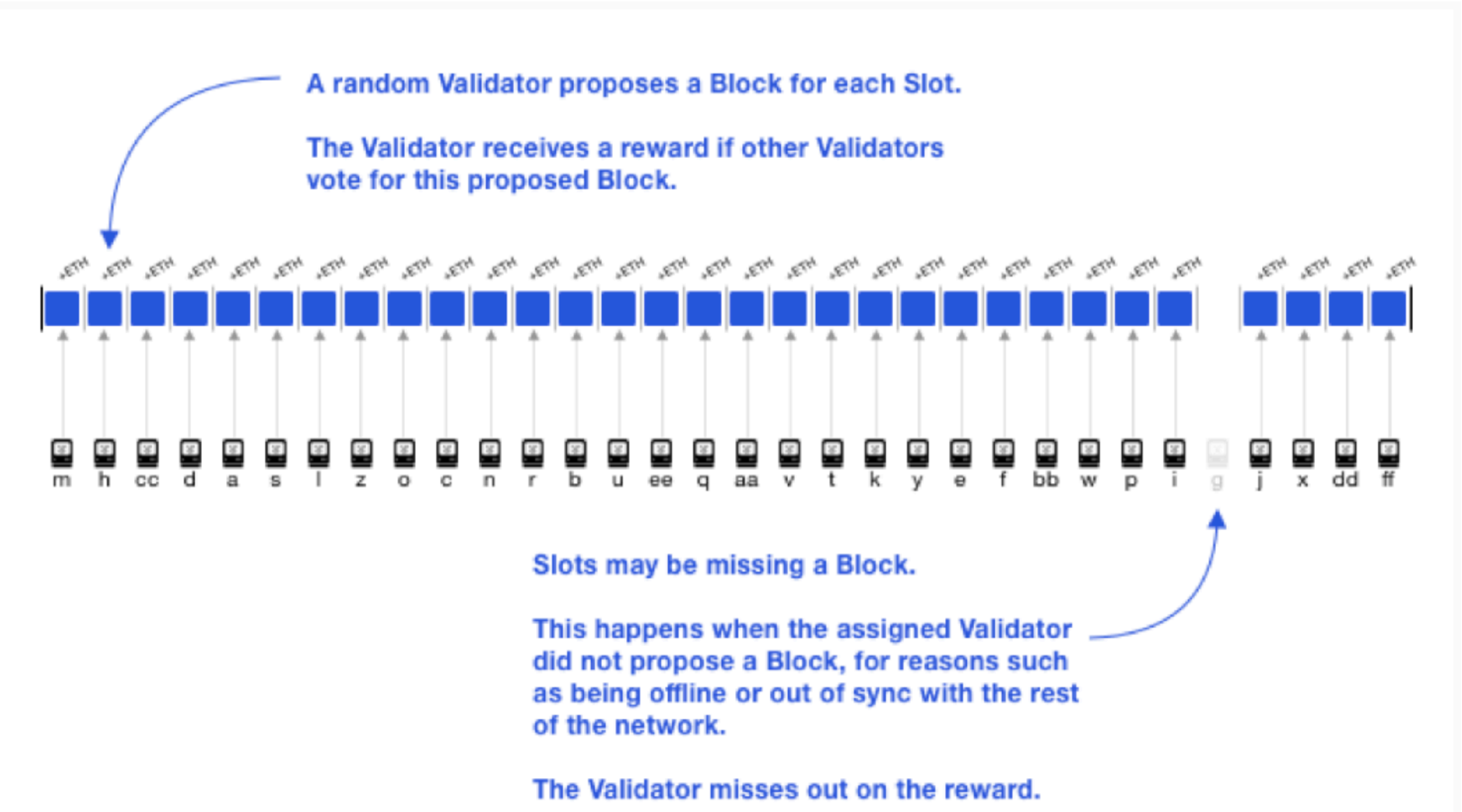
Blocks and slots



The first 32 slots are in Epoch 0. Genesis blocks are at Slot 0.

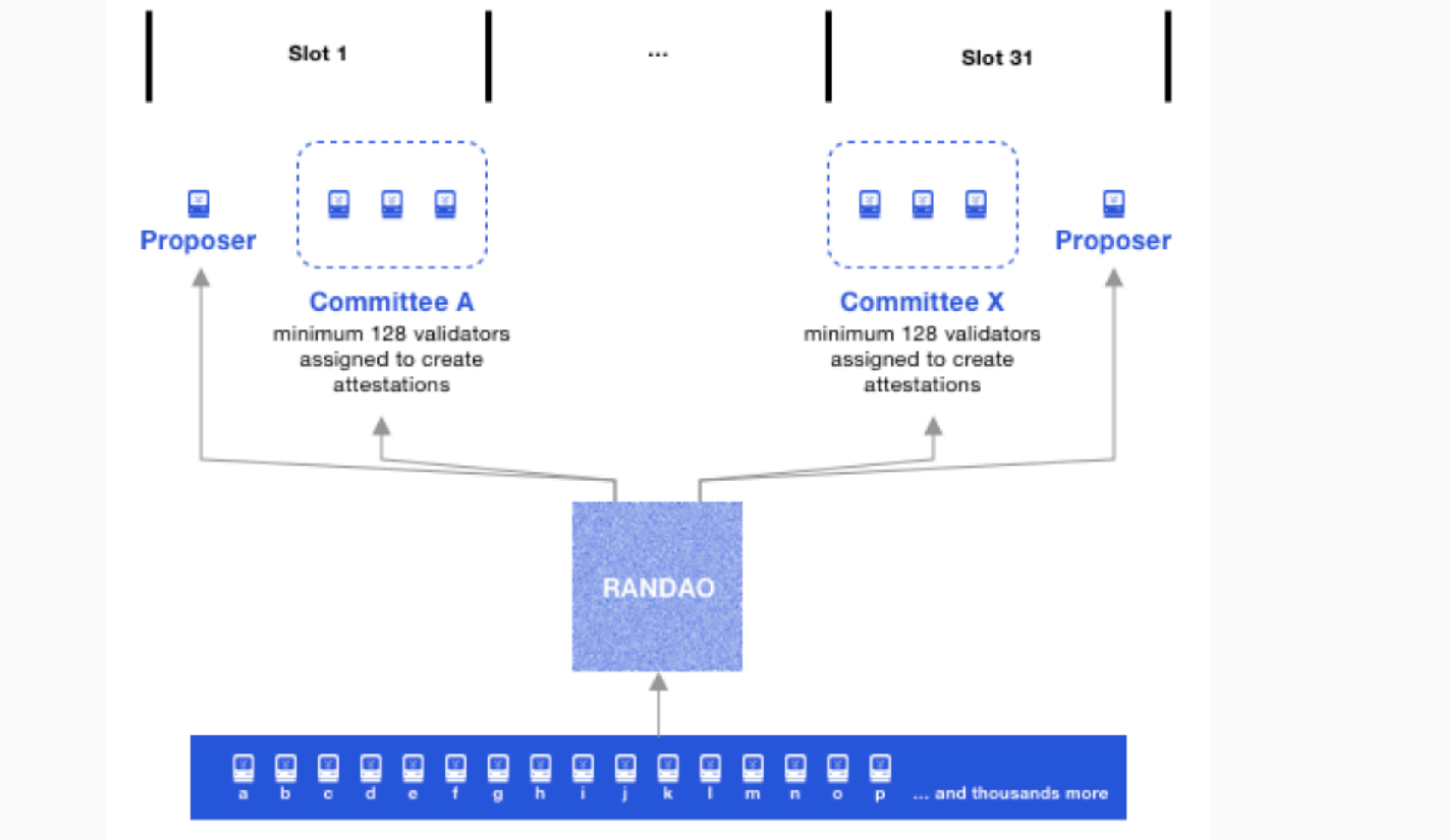
https://ethos.dev/beacon-chain

Blocks and slots



https://ethos.dev/beacon-chain

Slot committees



https://ethos.dev/beacon-chain

Validators: activity & double voting

- Validators are incentivized for voting
  - Via block rewards
  - Will lose stake for failing to vote
- Validators can "vote" on more than one conflicting chain
  - When this happens, they can create forks or confusion
    - Fortunately double-votes on conflicting chains are easily detectable
    - To deter this outcome, validators caught signing two conflicting chains are punished by "slashing" their stake

Signatures and aggregation

- The beacon chain uses fancier cryptography
  - Instead of ECDSA signatures, it uses BLS signatures
  - BLS signatures have the nice property that they can be aggregated.
    - Given many signatures on the same message, it is easy to "compress" them into a single short signature
    - This means that blocks don't have to carry hundreds of signatures

Generating random numbers

- The beacon chain needs random numbers!
  - To select committees and make sure specific users cannot dominate the selection
  - If the random number generator is untrustworthy (or can be biased), the network's security is compromised
  - How do we pick random numbers on a blockchain?
    - Commit/reveal
    - BLS signatures
    - VDFs…

Sharding

- We will talk more about this next time