Blockchain: Smart Contracts

Lecture 6

Incentives and Accountability in Consensus: Proof-of-Stake

Incentives in Bitcoin

How does Bitcoin *incentivize* miners to participate in consensus and mine new blocks?

- Block rewards – currently 6.25 Bitcoin – halved every 210,000 blocks – halved ~4 years
- Transaction fees

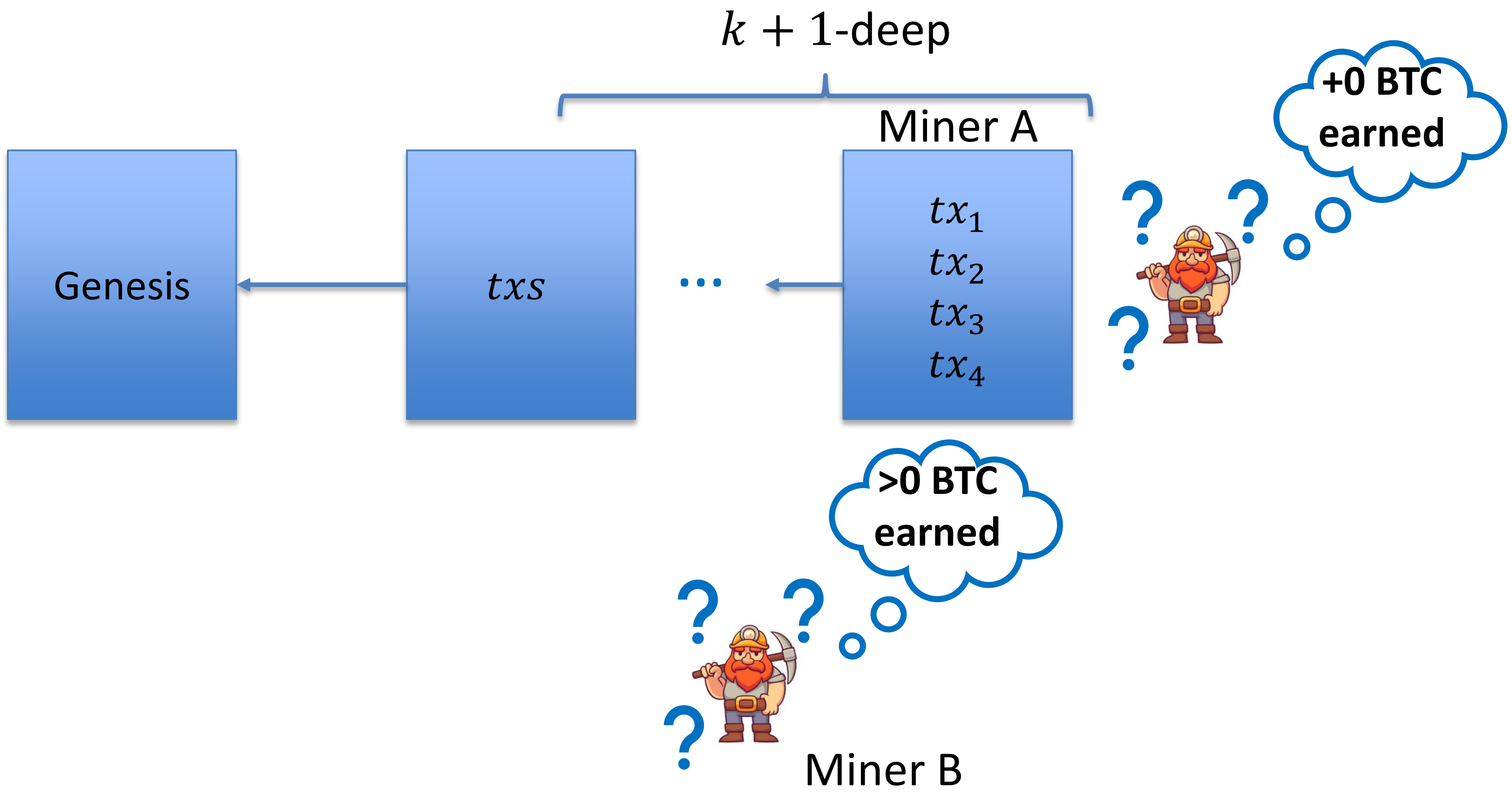
How does a miner capture these rewards?

- The first transaction in a Bitcoin block is called the **coinbase transaction**.
- The coinbase transaction can be created by the miner.
- Miner uses it to collect the block reward and the transaction fees.

Can these *incentives* guarantee *honest* participation?

- Not necessarily!
- Selfish mining attack!

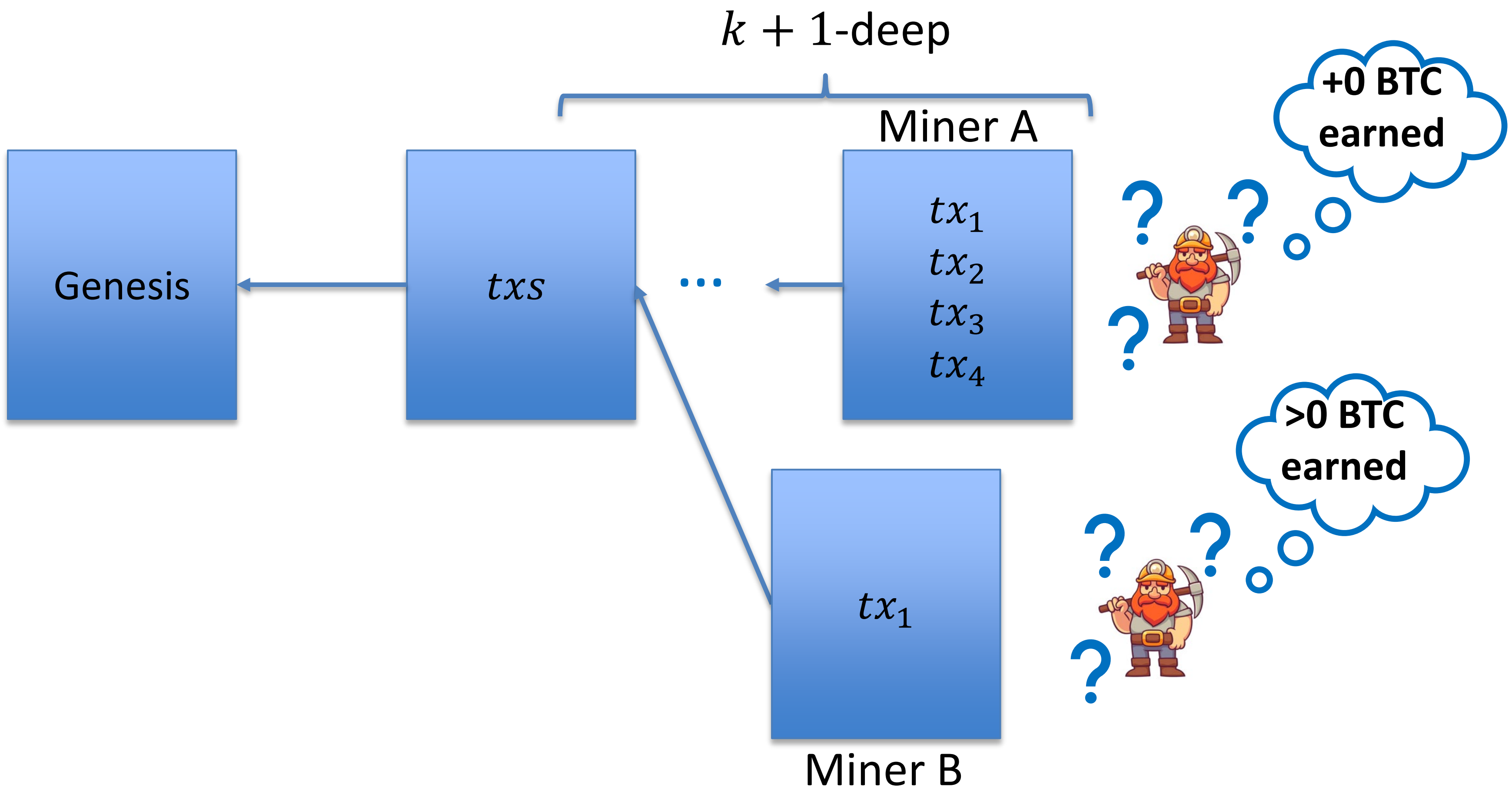
Incentives in Bitcoin



Transaction fees:

tx_1: 4 BTC
tx_2: 3 BTC
tx_3: 2 BTC
tx_4: 1 BTC

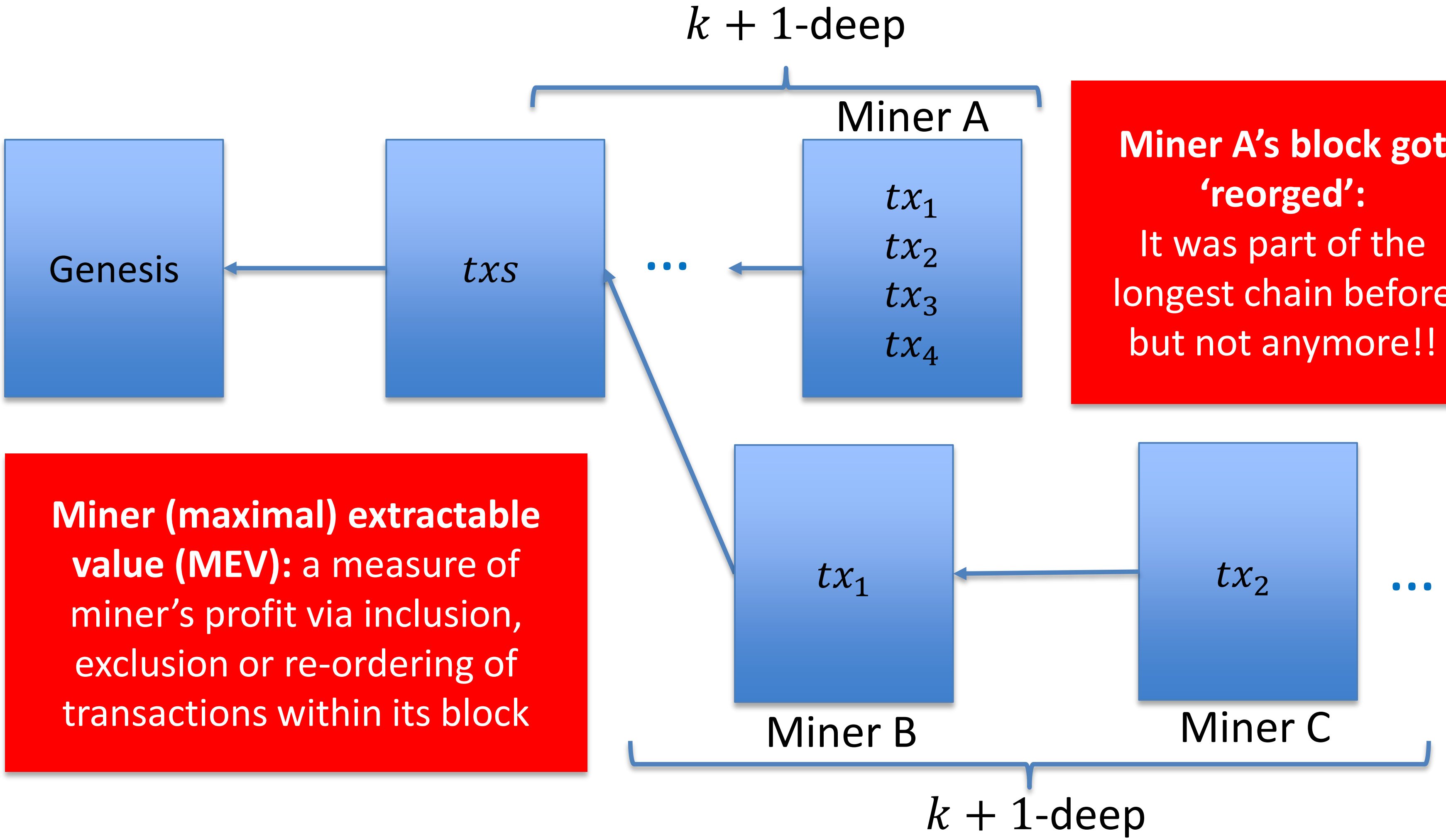
Incentives in Bitcoin



Transaction fees:

tx_1: 4 BTC
tx_2: 3 BTC
tx_3: 2 BTC
tx_4: 1 BTC

Incentives in Bitcoin



Miner A's block got 'reorged':
It was part of the longest chain before but not anymore!!

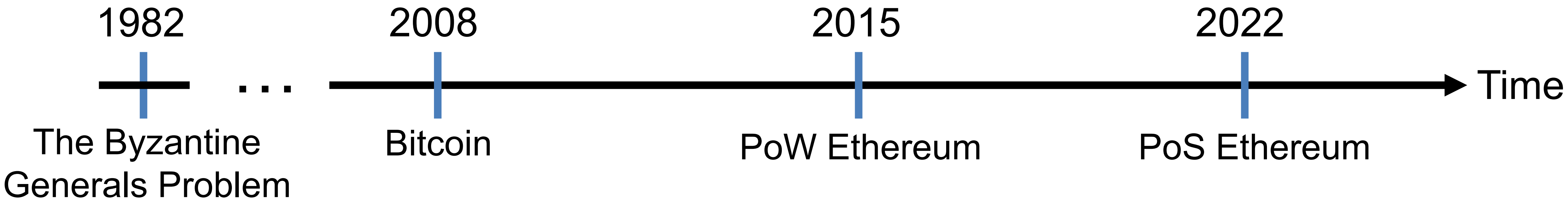
Transaction fees:
tx_1: 4 BTC
tx_2: 3 BTC
tx_3: 2 BTC
tx_4: 1 BTC
Total MEV: 10 BTC

Miner (maximal) extractable value (MEV): a measure of miner's profit via inclusion, exclusion or re-ordering of transactions within its block

Miners have incentive to violate the protocol!
Miners violate the protocol
→ No safety
→ Double-spend!

Need to think about incentives!!

From Bitcoin to Proof-of-Stake



Consensus in the Internet Setting
- Sybil resistance
- Dynamic availability
  - (Liveness under changing part.)

Block rewards (carrot)
- ➢ to incentivize participation!

- ➢ Consensus in the Internet Setting
  - ➢ Sybil resistance
  - ➢ Dynamic availability
- ➢ Block rewards (carrot)
- ➢ Finality and accountable safety
- ➢ Slashing (stick)
  - ➢ to punish protocol violation!

The Byzantine Generals Problem (1982)
Bitcoin: A Peer-to-Peer Electronic Cash System (2008)
Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform. (2015)
Combining GHOST and Casper (2020)

A few words on Proof-of-Stake (PoS)

In a Proof-of-Stake protocol, nodes lock up (i.e., stake) their coins in the protocol to become eligible to participate in consensus.

The more coins staked by a node…

- Higher the probability that the node is elected as a leader.
- Larger the weight of that node's actions.

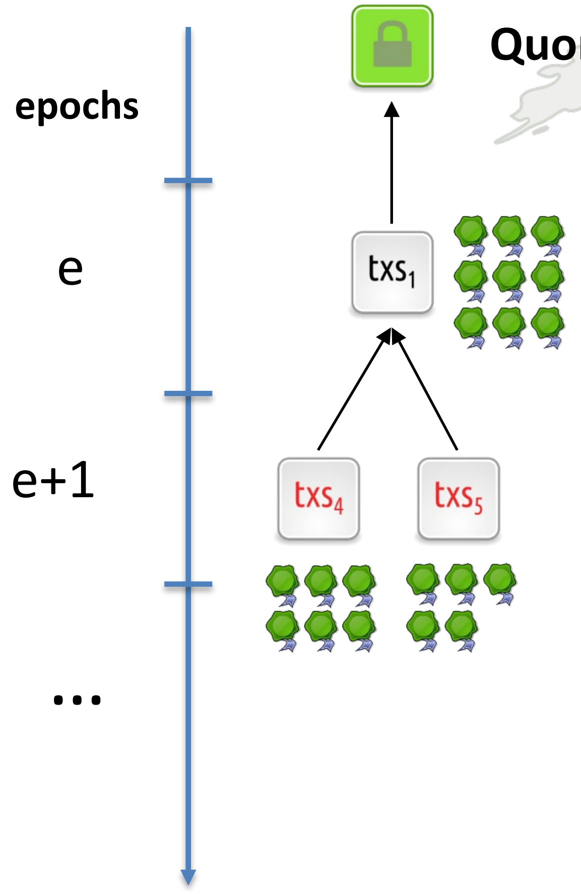
If a node is caught doing an adversarial action (e.g., sending two values), it can be punished by burning its locked coins (stake)! This is called *slashing*.

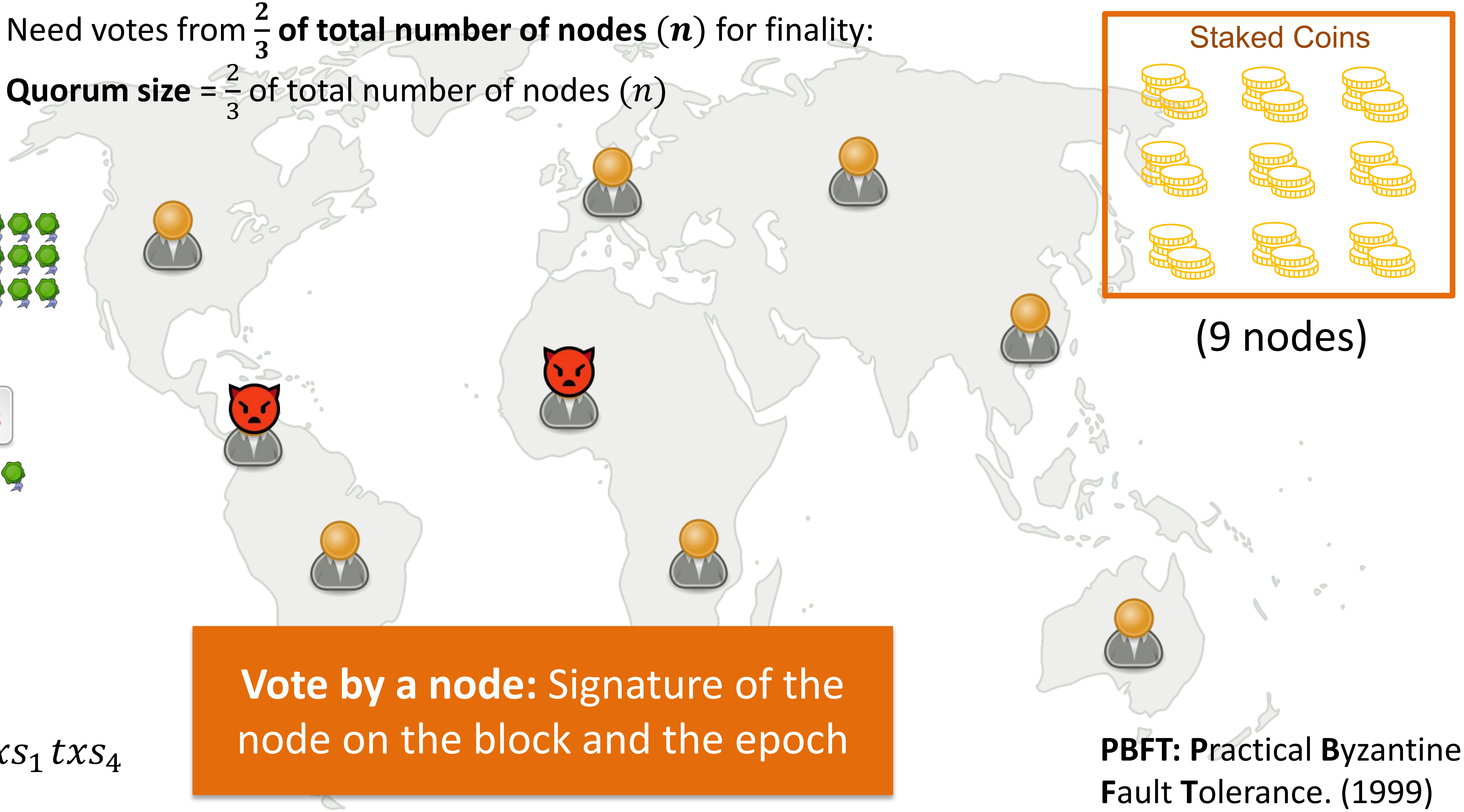
Thus, in a Proof-of-Stake protocol, nodes can be held *accountable* for their actions (unlike in Bitcoin, where nodes do not lock up coins).

A Simple (PBFT-style) PoS Protocol*

Need votes from $\frac{2}{3}$ **of total number of nodes** $(\boldsymbol{n})$ for finality:

Quorum size = $\frac{2}{3}$ of total number of nodes (n)





(9 nodes)

Alice's log: $txs_1\ txs_4$

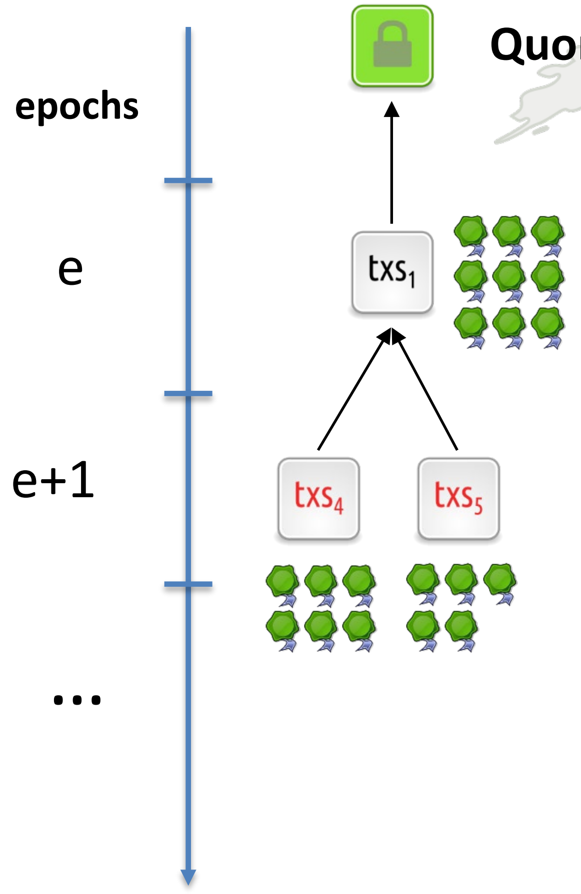
Vote by a node: Signature of the node on the block and the epoch

PBFT: Practical **B**yzantine **F**ault **T**olerance. (1999)

A Simple (PBFT-style) PoS Protocol*

Need votes from $\frac{2}{3}$ **x total number of nodes** $(\boldsymbol{n})$ for finality:

Quorum size = $\frac{2}{3}$ x total number of nodes (n)



Votes for txs_4

$\geq \frac{2}{3}n$

Votes for txs_5

$\geq \frac{2}{3}n$

Alice's log: $txs_1\ txs_4$

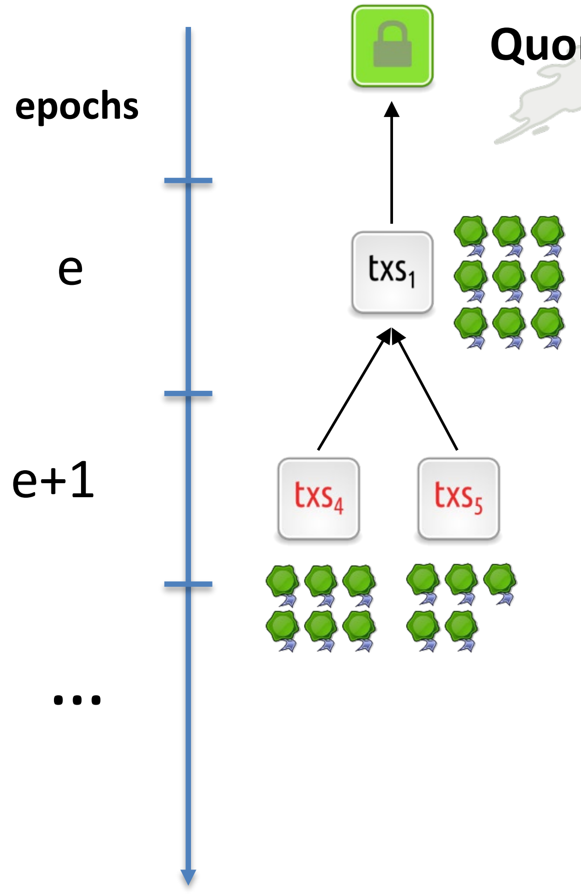
Vote by a node: Signature of the node on the block and the epoch

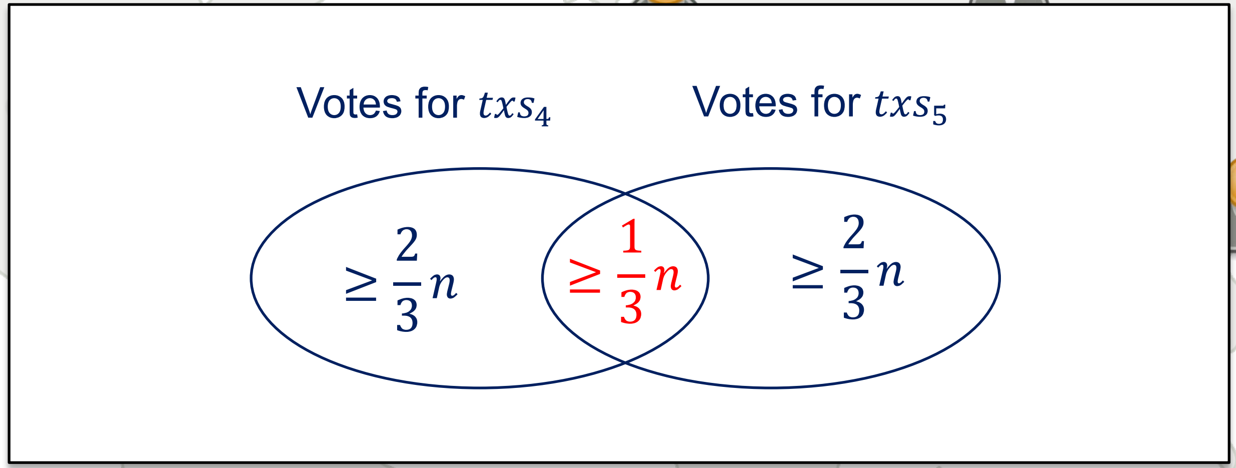
PBFT: Practical **B**yzantine **F**ault **T**olerance. (1999)

A Simple (PBFT-style) PoS Protocol*

Need votes from $\frac{2}{3}$ **x total number of nodes** $(\boldsymbol{n})$ for finality:

Quorum size = $\frac{2}{3}$ x total number of nodes (n)





Alice's log: $txs_1\ txs_4$

$\geq \frac{1}{3}n$ of nodes must have voted twice (once for txs_4 and once for txs_5)

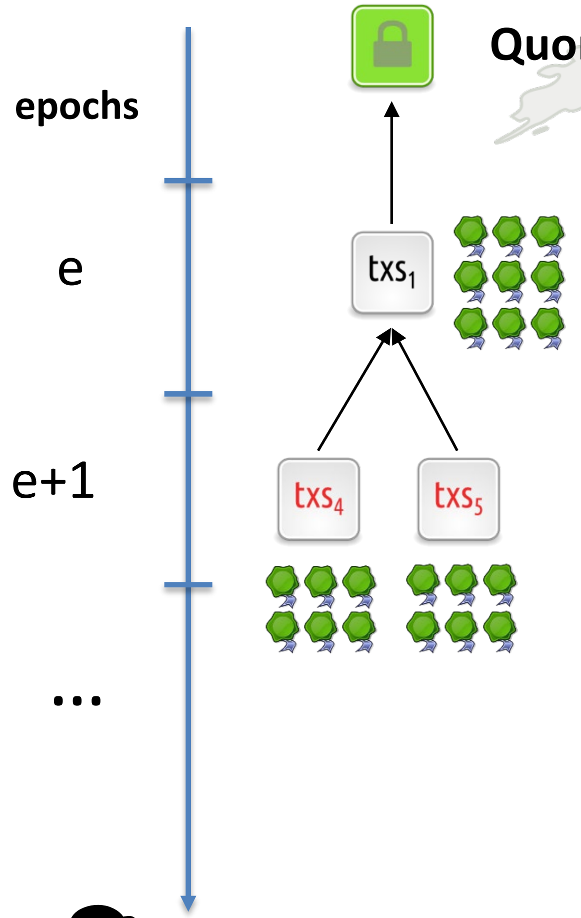
Therefore, $\geq \frac{1}{3}n$ nodes are adversarial

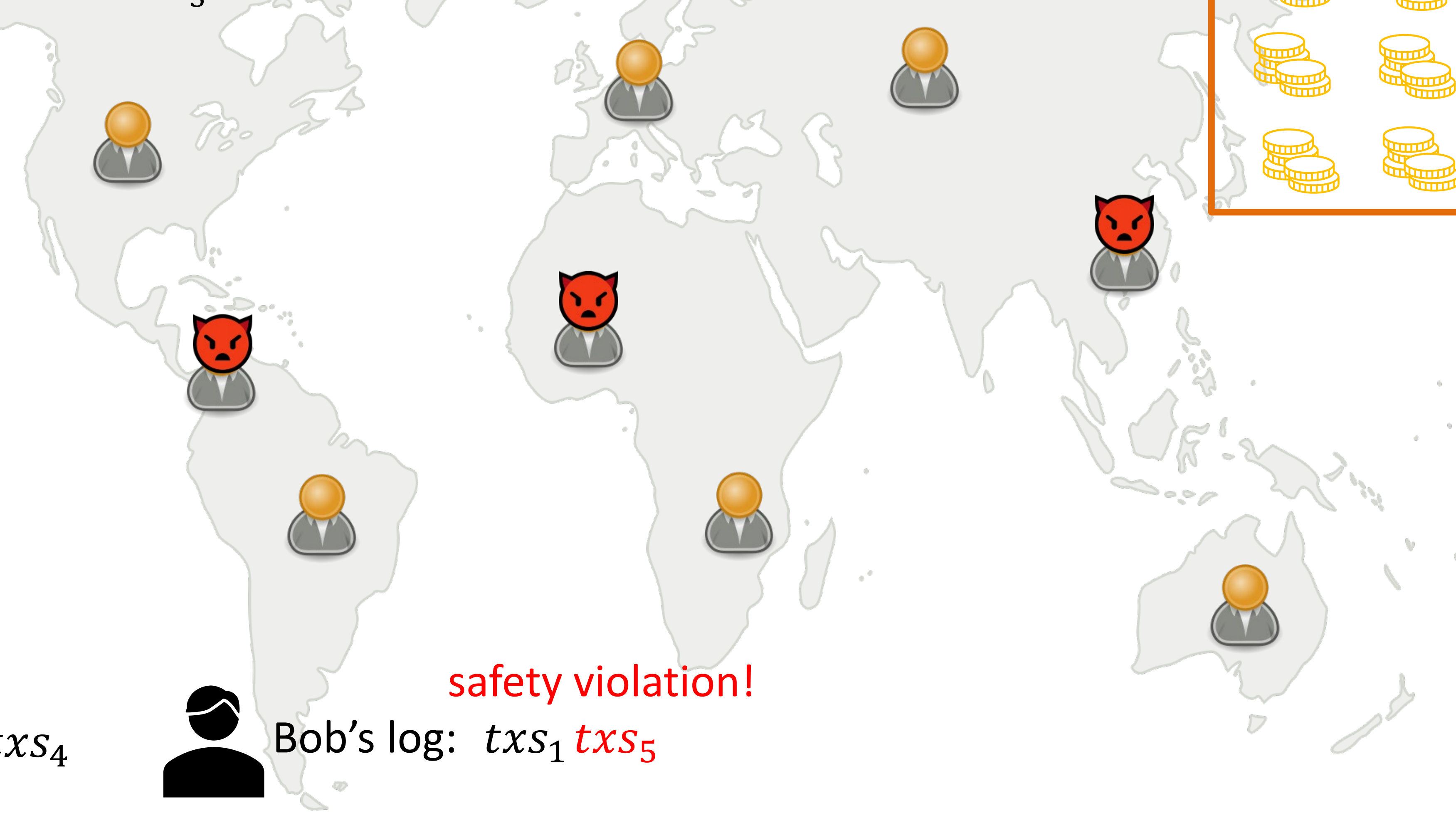
Safety when # adversarial nodes $< \frac{1}{3}$ x total number of nodes

A Simple (PBFT-style) PoS Protocol*

Need votes from $\frac{2}{3}$ **x total number of nodes** $(\boldsymbol{n})$ for finality:

Quorum size = $\frac{2}{3}$ x total number of nodes (n)





safety violation!

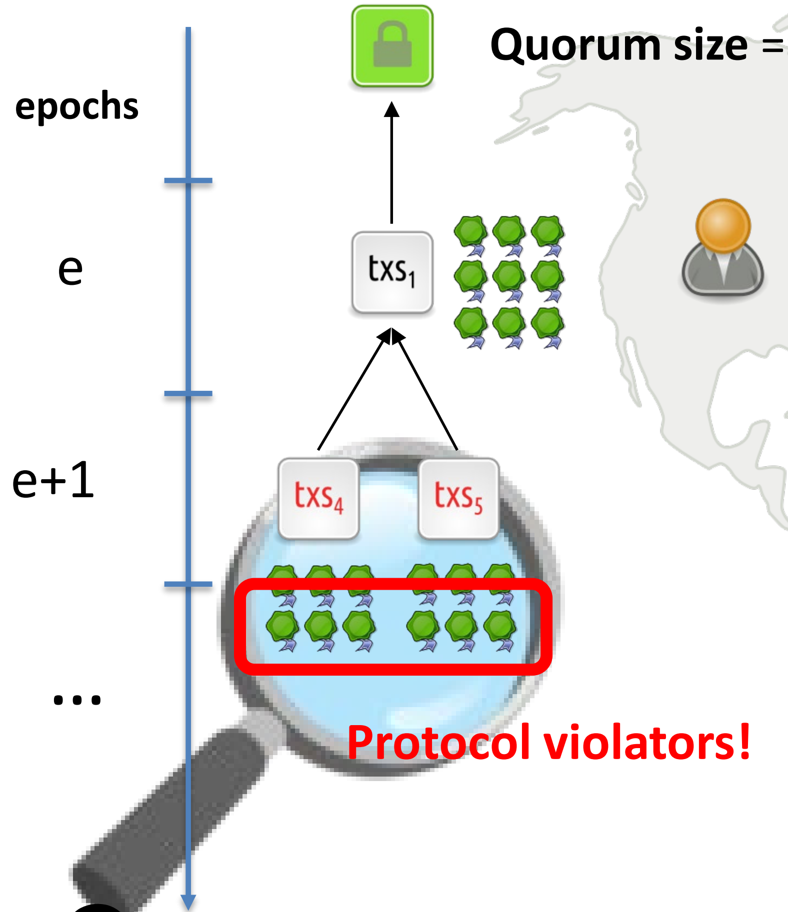
Alice's log: $txs_1\ txs_4$

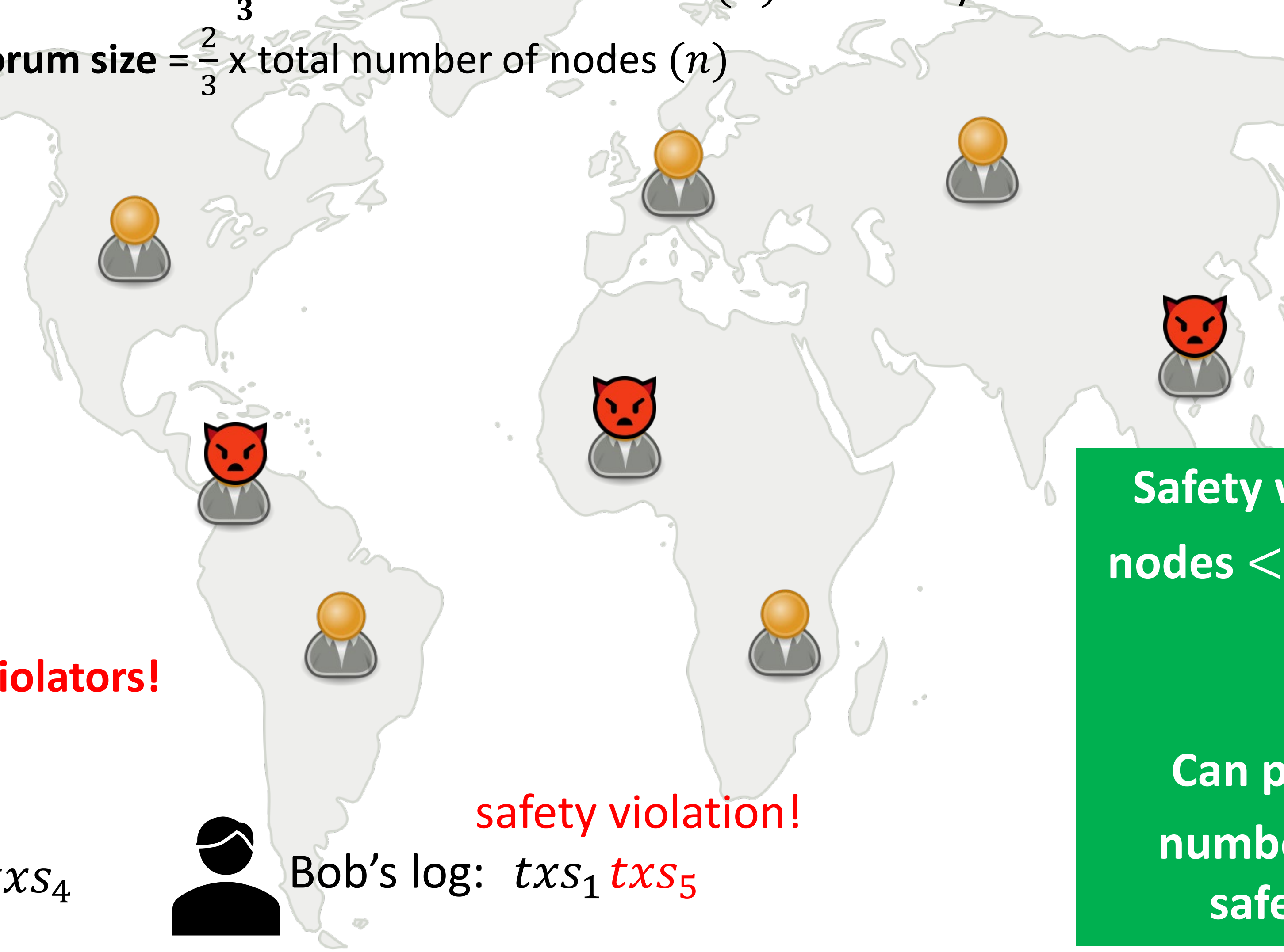
Bob's log: $txs_1\ txs_5$

A Simple (PBFT-style) PoS Protocol*

Need votes from $\frac{2}{3}$ **x total number of nodes** $(\boldsymbol{n})$ for finality:

Quorum size = $\frac{2}{3}$ x total number of nodes (n)





safety violation!

Alice's log: $txs_1\ txs_4$

Bob's log: $txs_1\ txs_5$

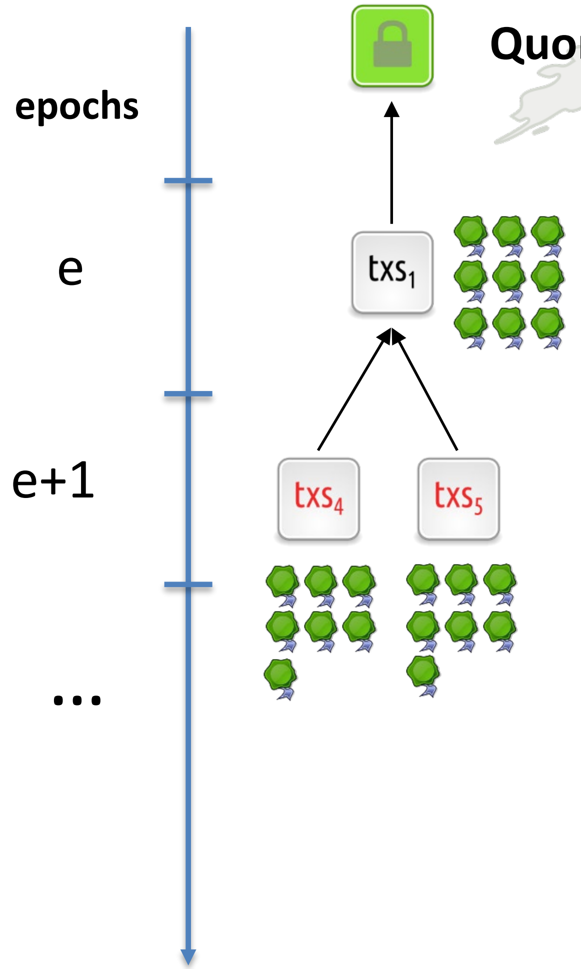
Safety when # adversarial nodes $< \frac{1}{3}$ x total number of nodes

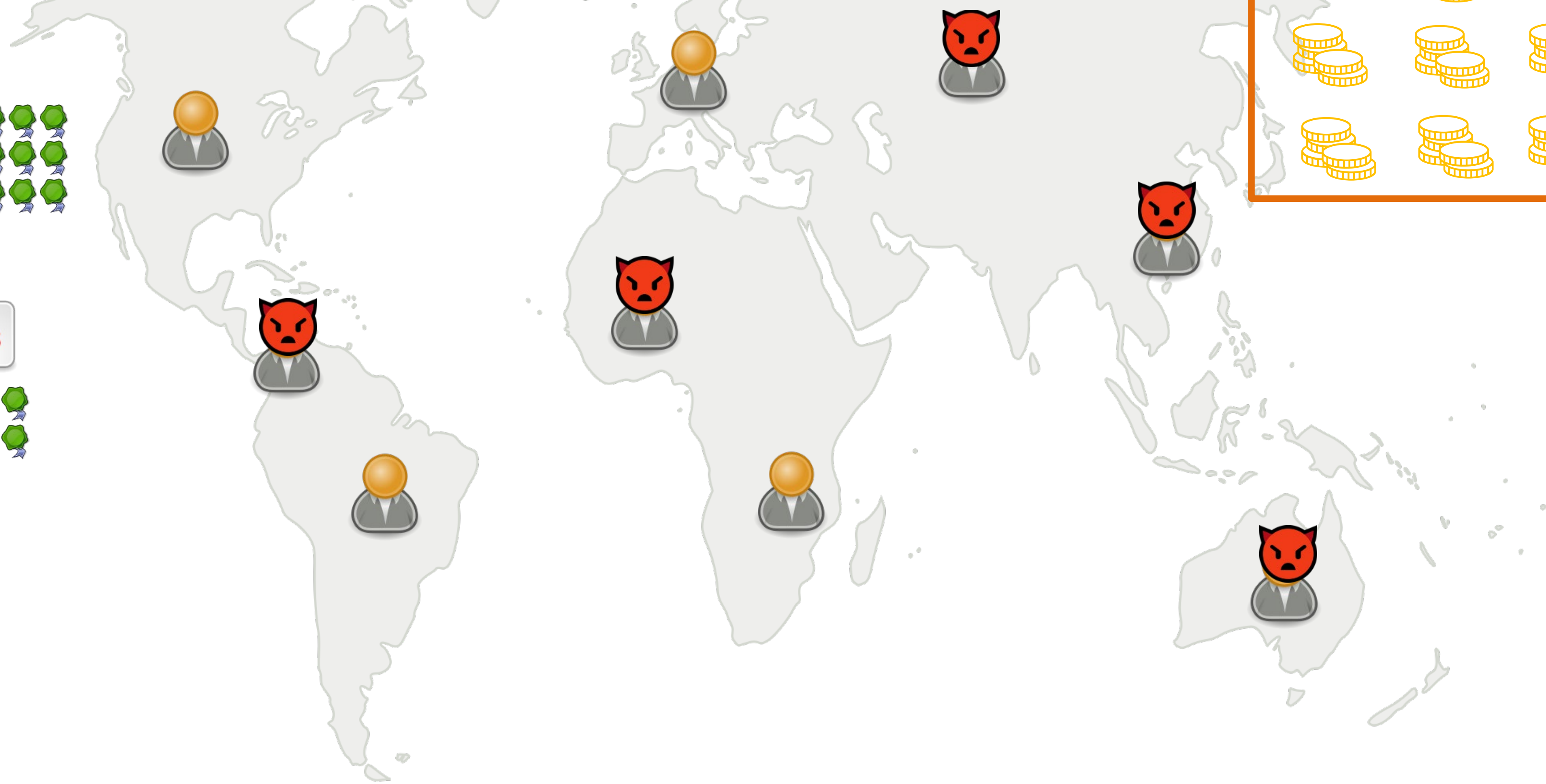
Can punish $\geq \frac{1}{3}$ x total number of nodes when safety is violated!!

A Simple (PBFT-style) PoS Protocol*

Need votes from $\frac{7}{9}$ **x total number of nodes** (n) for finality:

Quorum size = $\frac{7}{9}$ x total number of nodes (n)

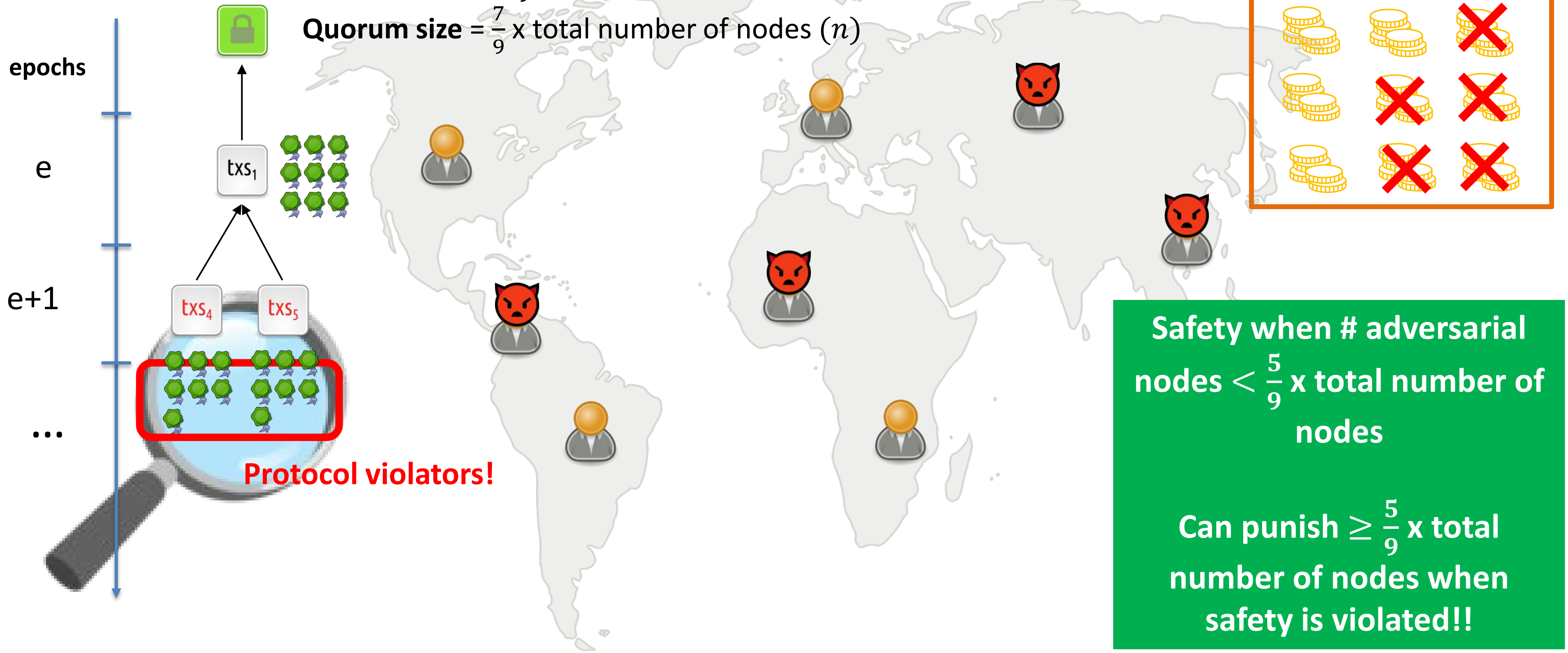




A Simple (PBFT-style) PoS Protocol*

Need votes from $\frac{7}{9}$ **x total number of nodes** $(\boldsymbol{n})$ for finality:

Quorum size = $\frac{7}{9}$ x total number of nodes (n)



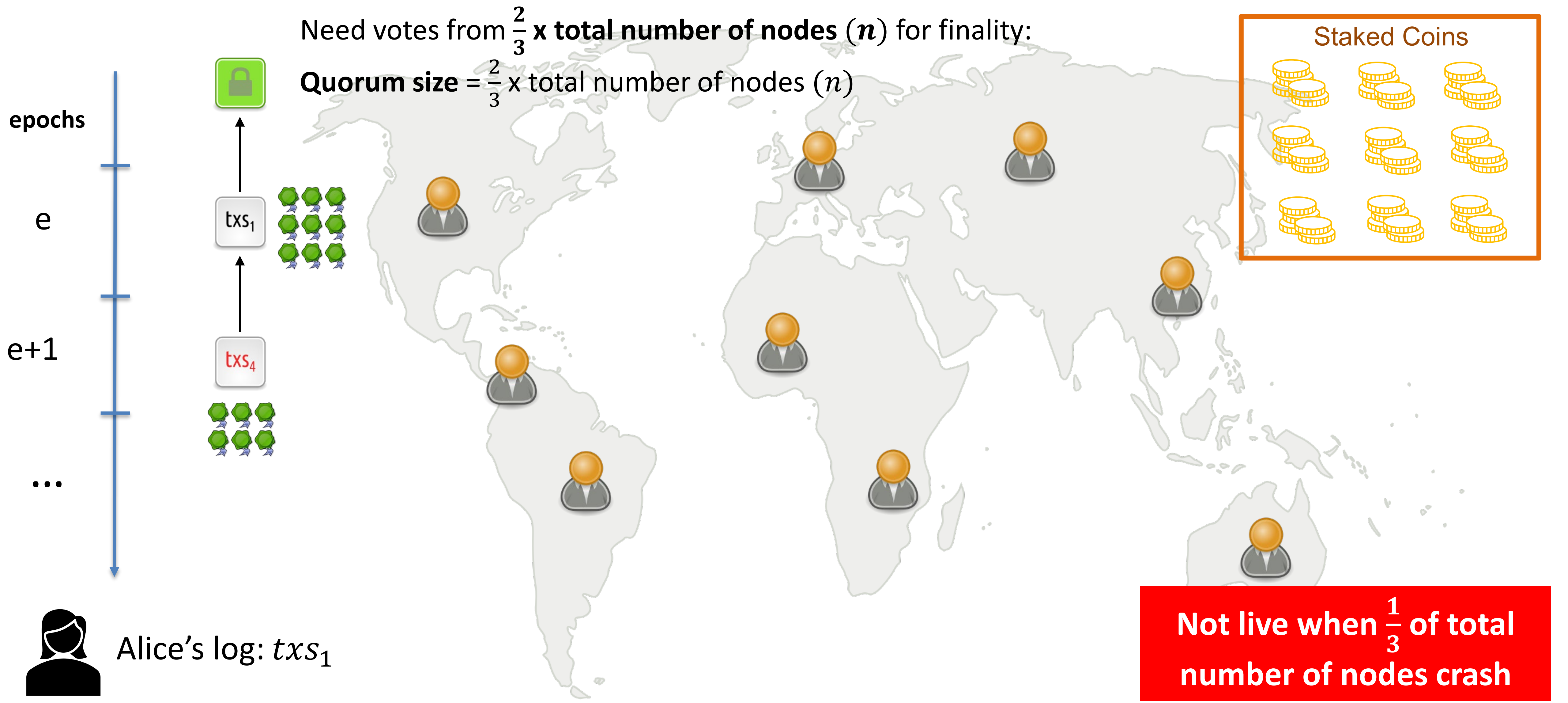
Safety when # adversarial nodes $< \frac{5}{9}$ x total number of nodes

Can punish $\geq \frac{5}{9}$ x total number of nodes when safety is violated!!

A Simple (PBFT-style) PoS Protocol*

Need votes from $\frac{2}{3}$ **x total number of nodes** $(\boldsymbol{n})$ for finality:

Quorum size = $\frac{2}{3}$ x total number of nodes (n)



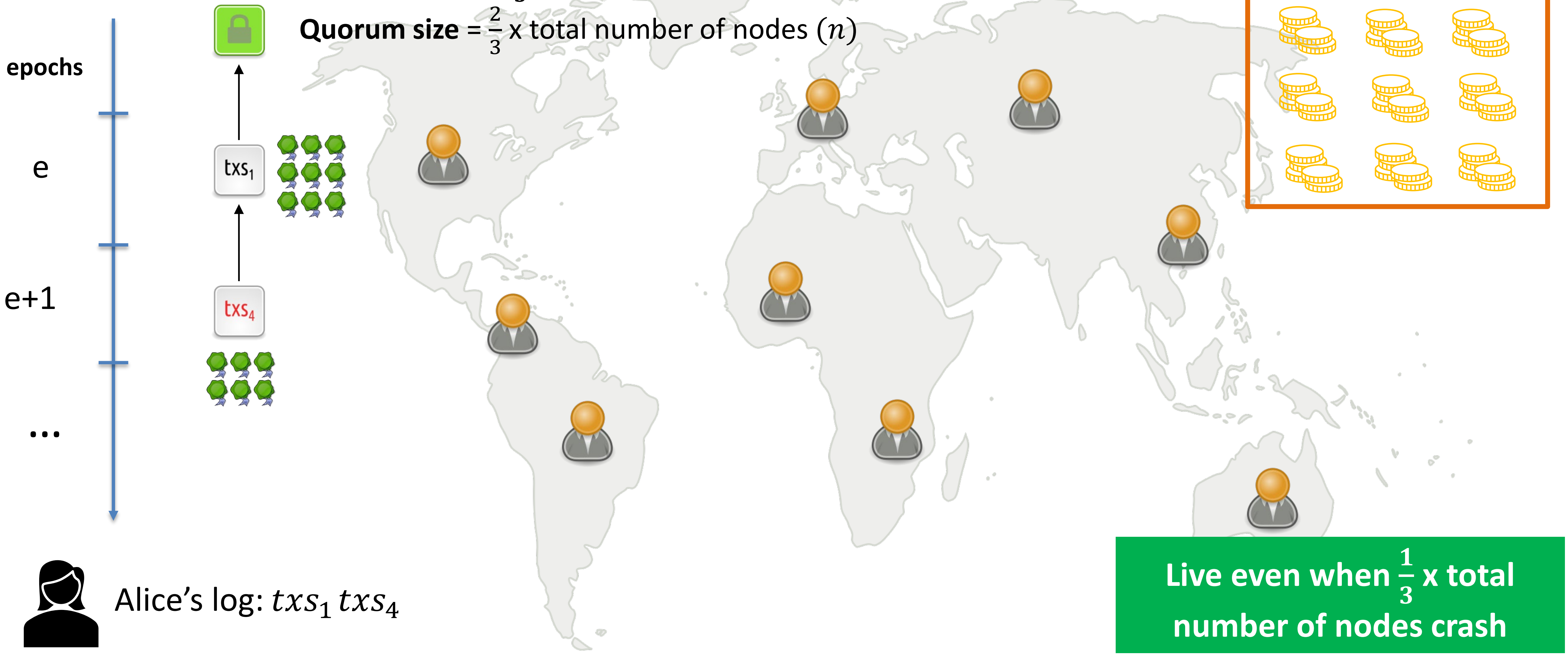
Alice's log: txs_1

Not live when $\frac{1}{3}$ of total number of nodes crash

A Simple (PBFT-style) PoS Protocol*

Need votes from $\frac{2}{3}$ **x total number of nodes** $(\boldsymbol{n})$ for finality:

Quorum size = $\frac{2}{3}$ x total number of nodes (n)



A Simple (PBFT-style) PoS Protocol*

- **Sybil resistance mechanism** determines how to select the nodes that are eligible to participate in consensus and propose/vote for transactions/blocks.
- **Consensus protocol** specifies the instructions for honest nodes so that given a set of eligible nodes with sufficiently many being honest, safety and liveness are satisfied.

Sybil resistance mechanism: **Consensus protocol (SMR):**	**Proof-of-Work**	**Proof-of-Stake**
Nakamoto consensus (longest chain) satisfies dynamic availability	Bitcoin PoW Ethereum	Ouroboros
PBFT-style (with votes) satisfies finality and accountable safety	??	PoS Ethereum* Simple PBFT-style PoS protocol

Ouroboros Genesis: Composable Proof-of-Stake Blockchains with Dynamic Availability. (2019)

Accountable Safety

In a protocol with resilience of *n/3:*

- The protocol is secure (safe & live) if there are less than *n/3* adversarial nodes.
- **Example:** The simple proof-of-stake protocol.

In a protocol with <u>*accountable safety resilience*</u> of *n/3:*

- The protocol is secure if there are less than *n/3* adversarial nodes.
- If there is <u>ever a safety violation</u>, all observers of the protocol can <u>provably</u> identify (i.e., catch) *at least n/3* adversarial nodes as protocol violators.
- No honest node is ever identified (no false accusation).
- **Examples:** The simple proof-of-stake protocol , PBFT, Tendermint, HotStuff …

Casper the Friendly Finality Gadget. (2017)
BFT Protocol Forensics (2021)

Accountable Safety

Accountable safety is a stronger notion than just security.

	0 to $n/3$	$n/3$ to $2n/3$	$2n/3$ to 1
Resilience of n/3	Safety & Liveness ☺	No Safety or Liveness ☹	No Safety or Liveness ☹
Accountable safety resilience of n/3	Safety & Liveness ☺	• No liveness ☹ • If safety is violated, catch and punish adversarial nodes ☺	

Finality

- We say that a protocol provides *finality* with resilience $\frac{n}{3}$ if it preserves safety during periods of asynchrony, when there are less than $\frac{n}{3}$ adversarial nodes.

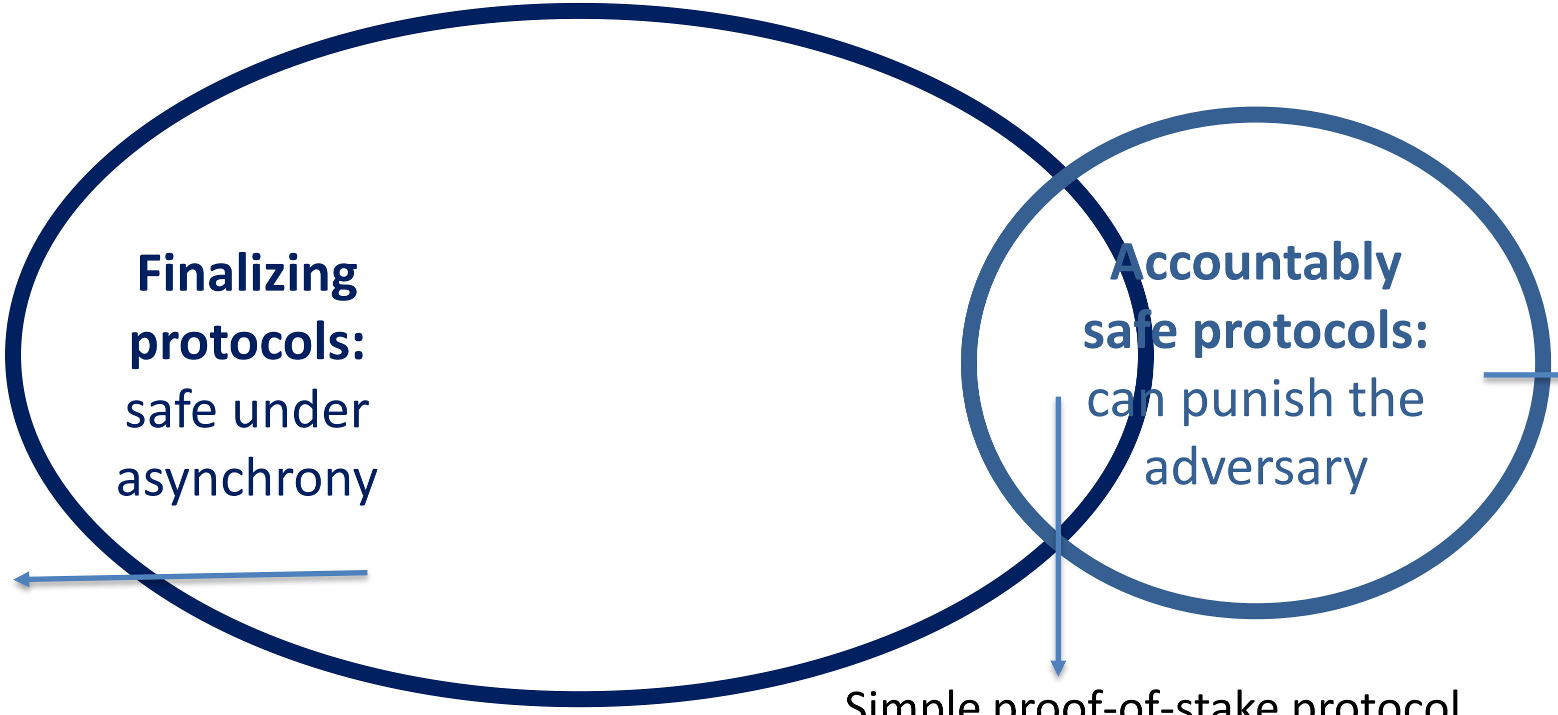
  - **Recall:** under asynchrony, messages can be delayed *arbitrarily* for a finite time.
  - **Example:** The simple proof-of-stake protocol, PBFT, Tendermint, HotStuff …

- Interestingly, in *most* protocol providing *finality*, transactions can be *finalized* much faster than they can be *confirmed* in Bitcoin.
  - No need to wait for k=6 blocks (1 hour)!

Accountability implies Finality

Accountability implies Finality:

Accountable safety (with resilience $\frac{n}{3}$) implies **finality** (with resilience $\frac{n}{3}$).



HotStuff-null

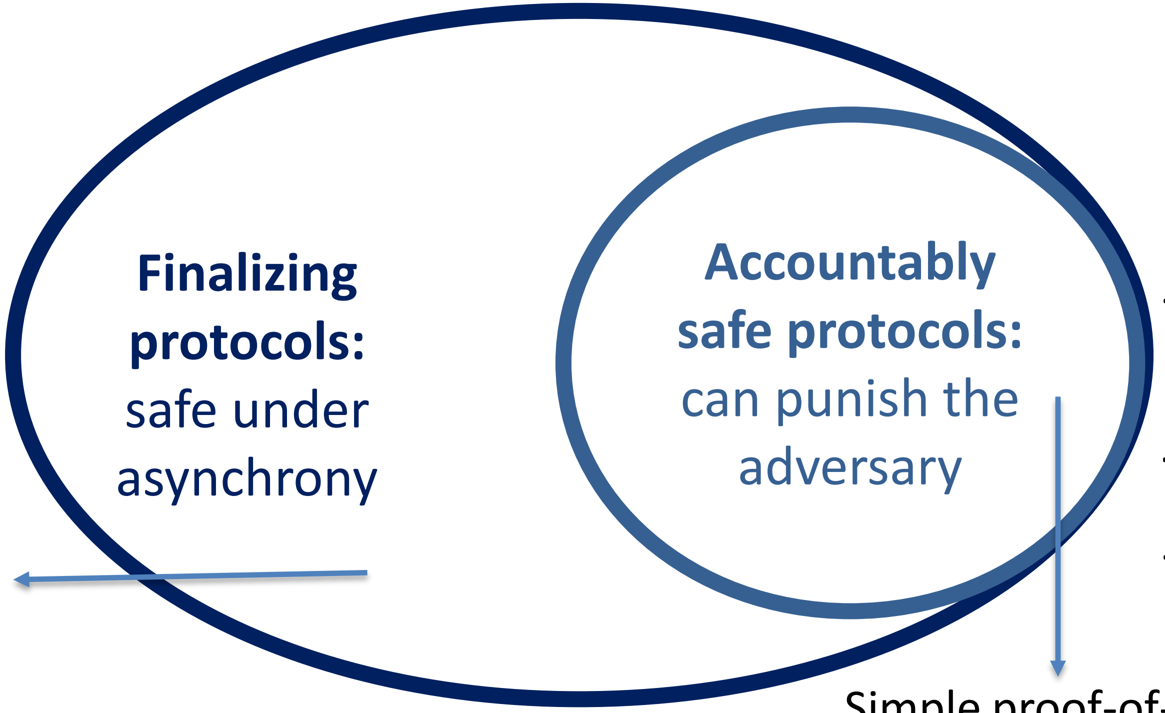
???

Simple proof-of-stake protocol, PBFT, HotStuff, …

BFT Protocol Forensics (2021)
Accountable Safety Implies Finality (2023)

Accountability implies Finality

Accountability implies Finality:
Accountable safety (with resilience $\frac{n}{3}$) implies **finality** (with resilience $\frac{n}{3}$).



HotStuff-null

Simple proof-of-stake protocol, PBFT, HotStuff, …

(Accountable safety:) if the protocol can punish at least $\frac{n}{3}$ adv. nodes after a safety violation (and is safe when there are less than $\frac{n}{3}$ adv. nodes),

Then **(Finality:)** it must be safe when there are less than $\frac{n}{3}$ adv. nodes even under <u>asynchrony</u>.

BFT Protocol Forensics (2021)
Accountable Safety Implies Finality (2023)

Holy Grail of Internet Scale Consensus

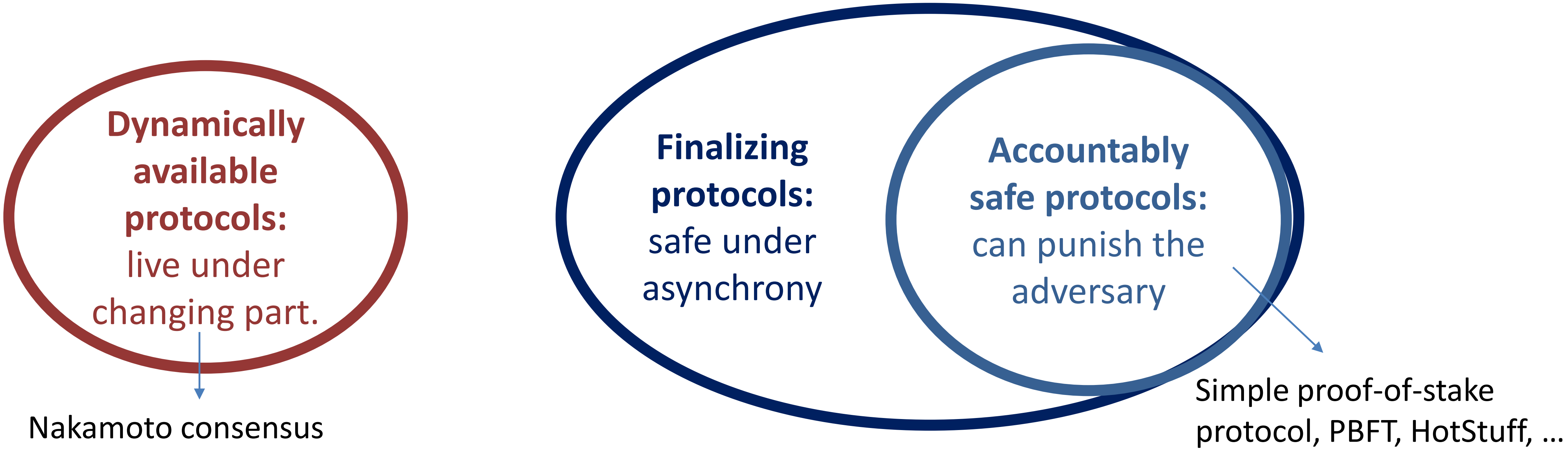
- We want Sybil resistance: Proof-of-Work or Proof-of-Stake…
- We want dynamic availability so that…
  - Transactions continue to be confirmed and processed even when there is low participation.
  - Satisfied by Nakamoto consensus.
- We want finality and **accountable safety** so that…
  - Finality: There cannot be safety violations (double-spends) during asynchrony.
  - **Accountable safety:** Nodes can be held accountable for their actions.
  - Satisfied by our simple proof-of-stake protocol, PBFT, HotStuff, …
- Let’s focus on having dynamic availability and finality for now…

Holy Grail of Internet Scale Consensus

Is there a SMR protocol that provides both dynamic availability and finality with any resilience?

No: Blockchain CAP Theorem!!

CAP: Consistency, **A**vailability, **P**artition tolerance

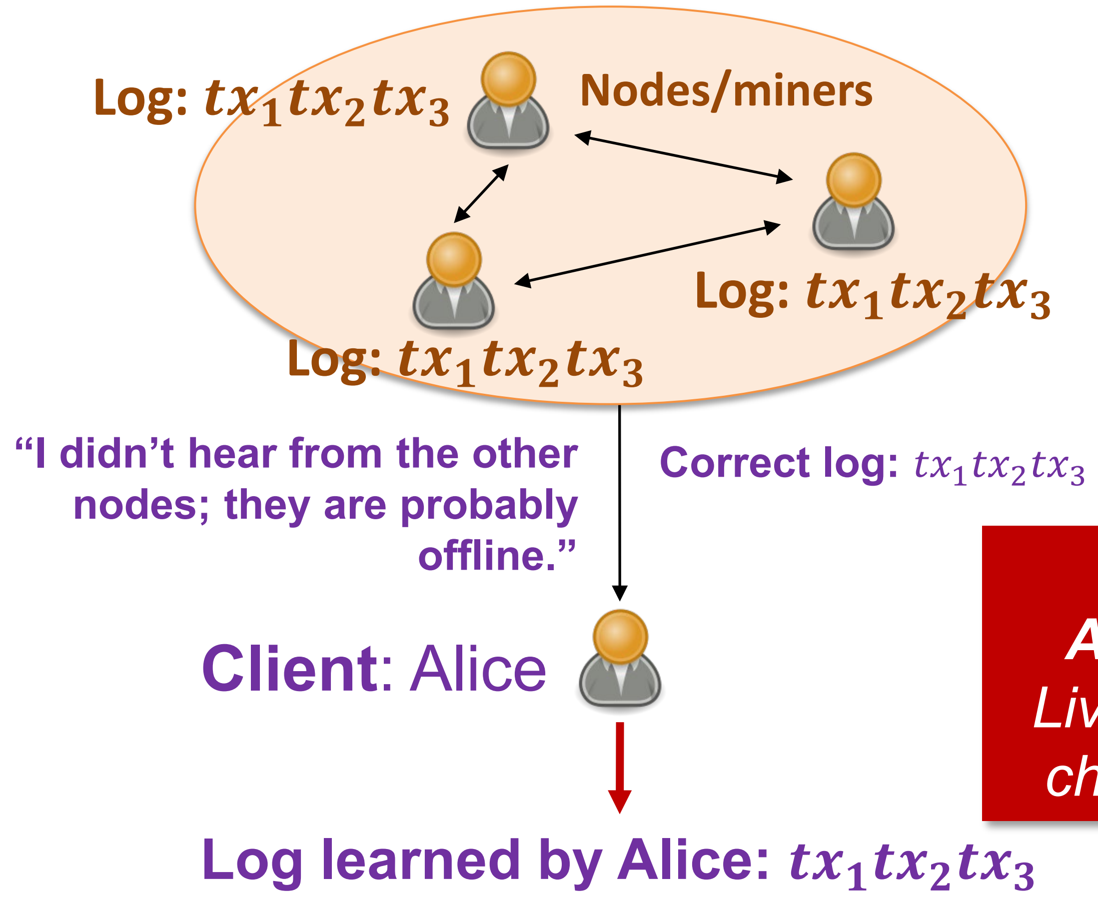


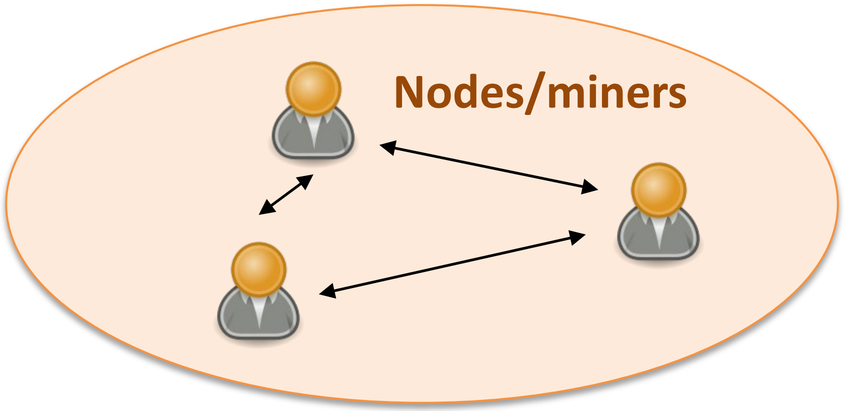
Resource Pools and the CAP Theorem (2020)

Blockchain CAP Theorem

For contradiction, suppose our SMR protocol has both dynamic availability and finality.

World 1



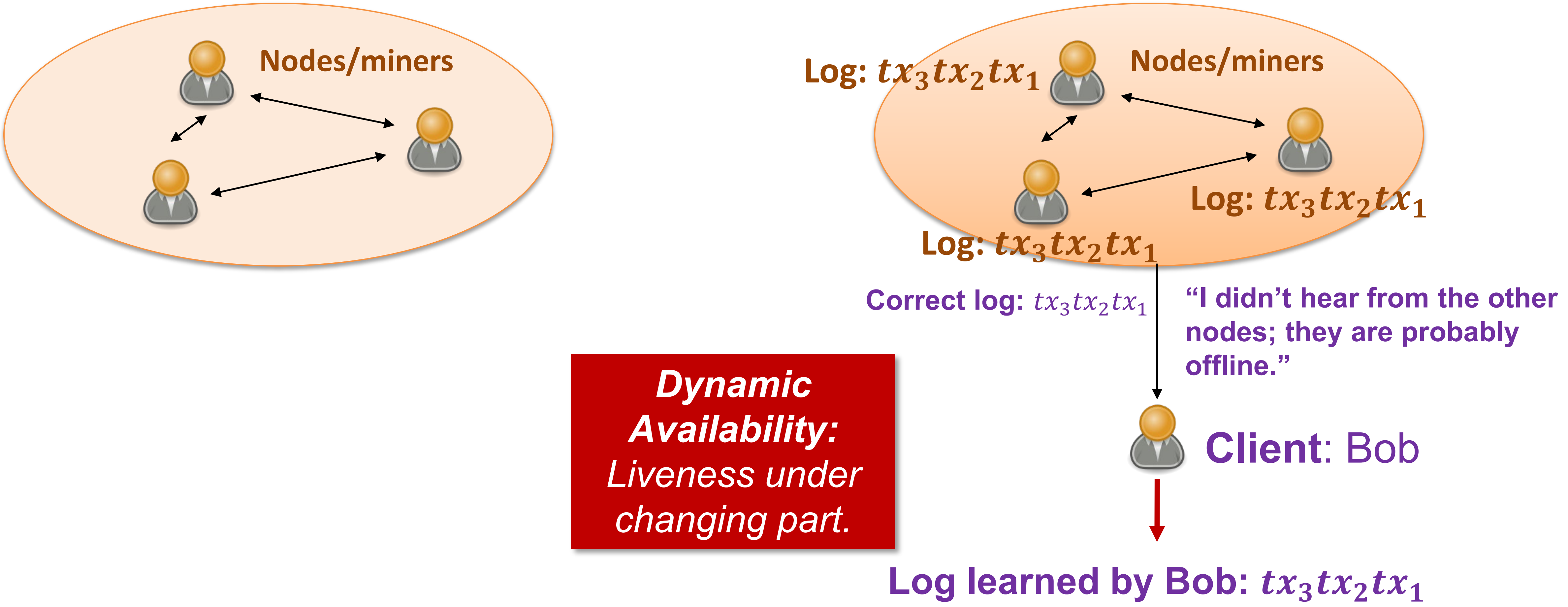


Dynamic Availability: *Liveness under changing part.*

Blockchain CAP Theorem

For contradiction, suppose our SMR protocol has both dynamic availability and finality.

World 2

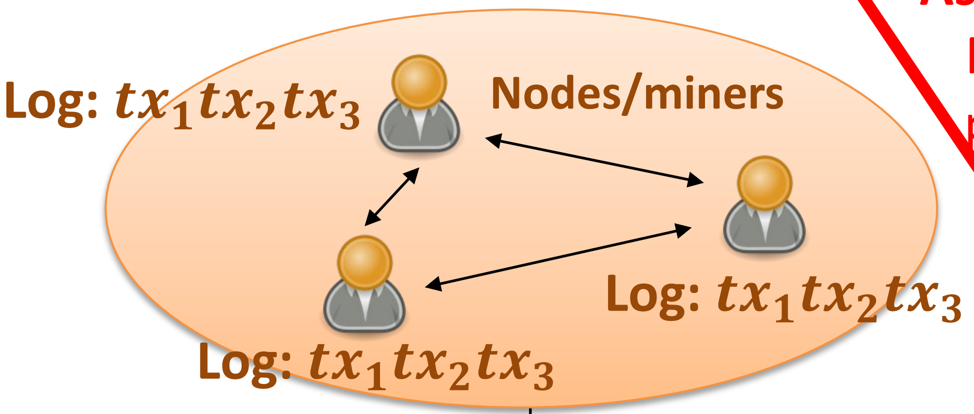


Dynamic Availability: *Liveness under changing part.*

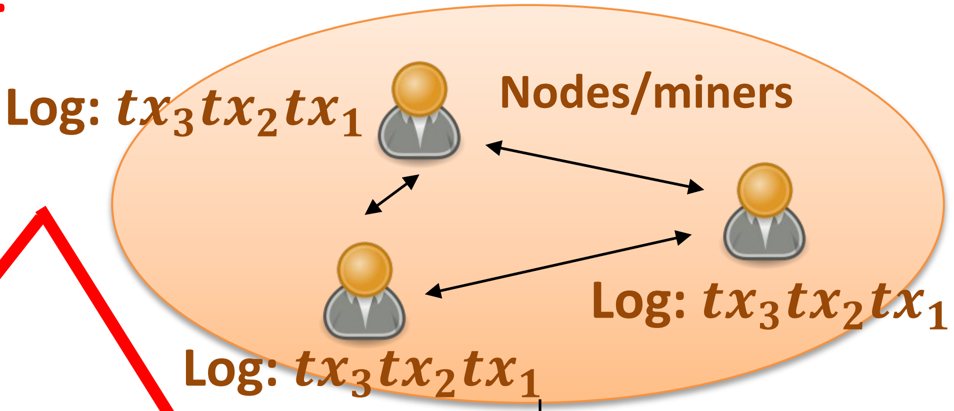
Blockchain CAP Theorem

For contradiction, suppose our SMR protocol has both dynamic availability and finality.

World 3



Asynchrony:
Network partition



"I didn't hear from the other nodes; they are probably offline. **I am in world 1.**"

Correct log: $tx_1 tx_2 tx_3$

Correct log: $tx_3 tx_2 tx_1$

"I didn't hear from the other nodes; they are probably offline. **I am in world 2.**"

Client: Alice

Client: Bob

Safety violation!

No safety under <u>asynchrony</u>!

No finality!

Log learned by Alice: $tx_1 tx_2 tx_3$

Log learned by Bob: $tx_3 tx_2 tx_1$

Resolution: Nested Ledgers/Chains

Single chain: tx_1, tx_2, tx_3, …

- Finality: Safe under asynchrony
- Dynamic availability: Live under changing participation

Impossible! Due to the CAP Theorem!

Accountable finalized chain	Available chain

Accountable finalized chain

- **Prefix property:** Prefix of the available chain.
- Accountably safe under asynchrony.
- Live once the network becomes synchronous and if enough nodes are online.
- Not live under low participation.

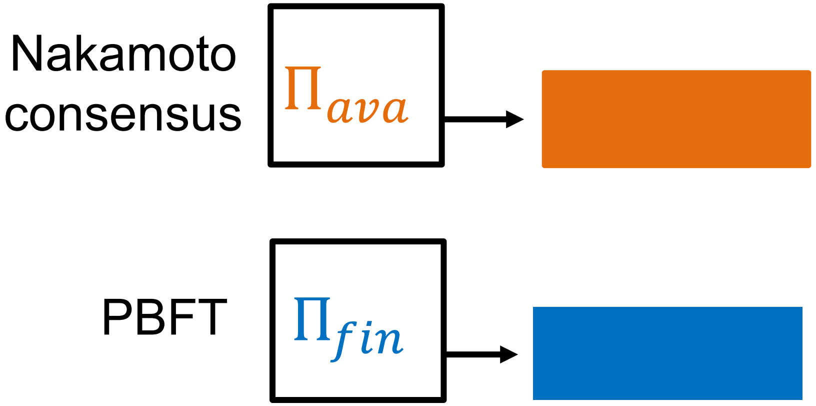
Available chain

- Safe and live under synchrony and changing participation.
- Not safe under asynchrony.

Ebb-and-Flow Protocols: A Resolution of the Availability-Finality Dilemma (2020)

Resolution: Nested Ledgers/Chains

Accountable finalized chain	Available chain
• **Prefix property:** Prefix of the available chain. • Accountably safe under asynchrony. • Live once the network becomes synchronous and if enough nodes are online. • Not live under small participation.	• Safe and live under synchrony and dynamic participation. • Not safe under asynchrony.



Safety-favoring client:
trusts acc. finalized chain

Liveness-favoring client:
trusts available chain

Client chooses!

Can interact with each other thanks to the prefix property!!

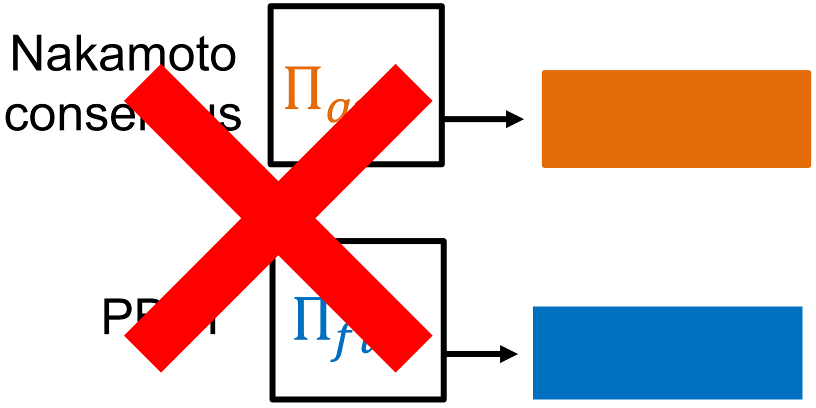
Ebb-and-Flow Protocols: A Resolution of the Availability-Finality Dilemma (2020)

Resolution: Nested Ledgers/Chains

Accountable finalized chain	Available chain

- **Prefix property:** Prefix of the available chain.
- Accountably safe under asynchrony.
- Live once the network becomes synchronous and if enough nodes are online.
- Not live under small participation.

- Safe and live under synchrony and dynamic participation.
- Not safe under asynchrony.



Ledgers can be inconsistent!

No prefix property!

Safety-favoring client:
trusts acc. finalized chain

Liveness-favoring client:
trusts available chain

Client chooses!

Can interact with each other thanks to the prefix property!!

Ebb-and-Flow Protocols: A Resolution of the Availability-Finality Dilemma (2020)

Resolution: Nested Ledgers/Chains

Safety-favoring client:
trusts acc. finalized chain

Liveness-favoring client:
trusts available chain

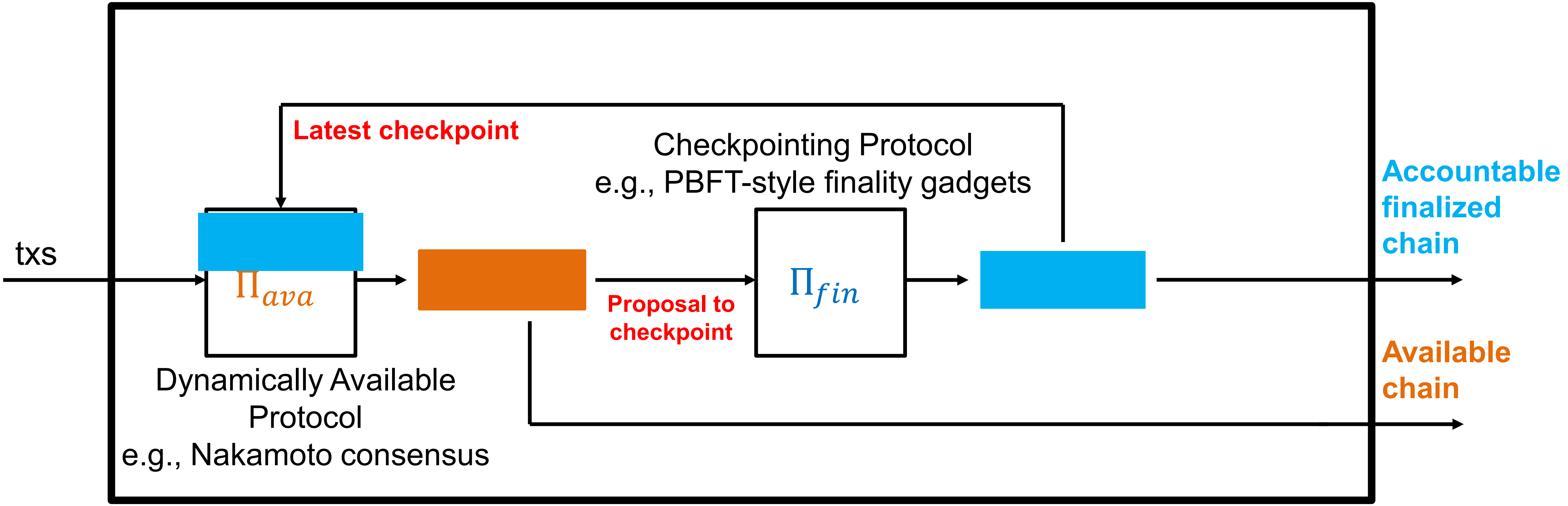
Client chooses!

- When the participation seems low at the weekend, it can either be that participation is actually low due to nodes taking time off or there is in fact a network partition.

- In this case, the boba vendor is willing to follow the available chain and risk a safety violation (and some double spend) due to a partition, since its transactions are of less value. By following the available chain, it can in turn keep selling boba at the weekends. Indeed, most of the time, there will not be a network partition, and participation will be low at the weekends due to nodes taking time off.

- However, the car vendor’s transactions have large value, and the car vendor cannot afford even one double spend! Therefore, it will follow the accountable, finalized chain that never has safety violations, but stops when there is low participation, e.g., at the weekends. This is fine since the car vendor has few transactions and can afford to wait the weekend. Indeed, on Monday, the accountable, finalized chain regains its liveness with higher participation.

How to obtain the nested chains?



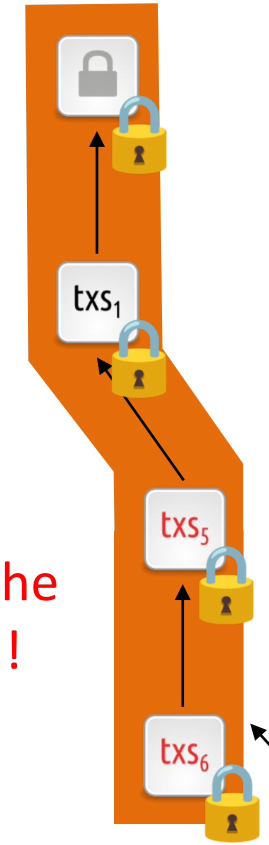
Combining GHOST and Casper. (2020)

How to obtain the nested chains?

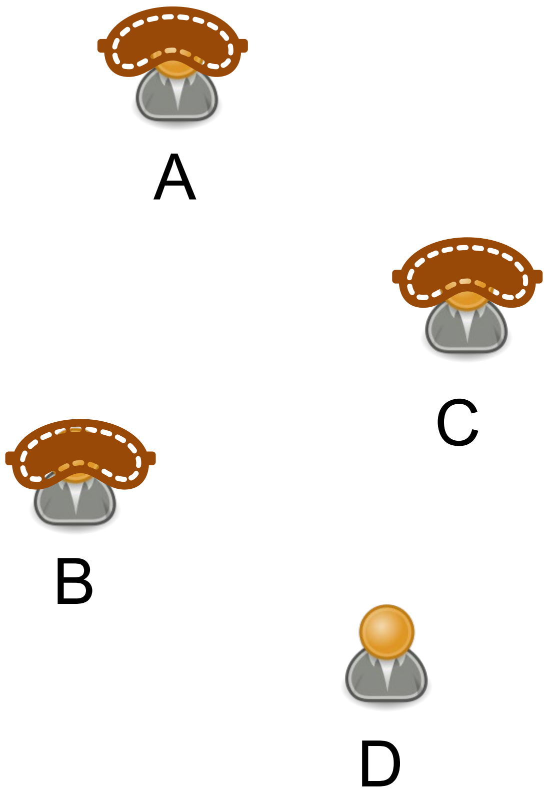
Nested Chains

Orange: available (full) chain

Blue: accountable, final (prefix) chain



Always extend the last checkpoint!!



Dynamic Availability: *Longest chain keeps growing.*

Checkpointing Protocol

Propose “txs5”

C votes “txs5”

B votes “txs5”

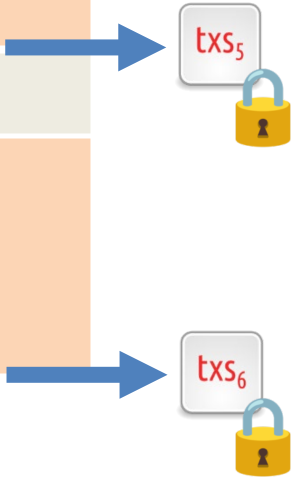
D votes “txs5”

Propose “txs6”

A votes “txs6”

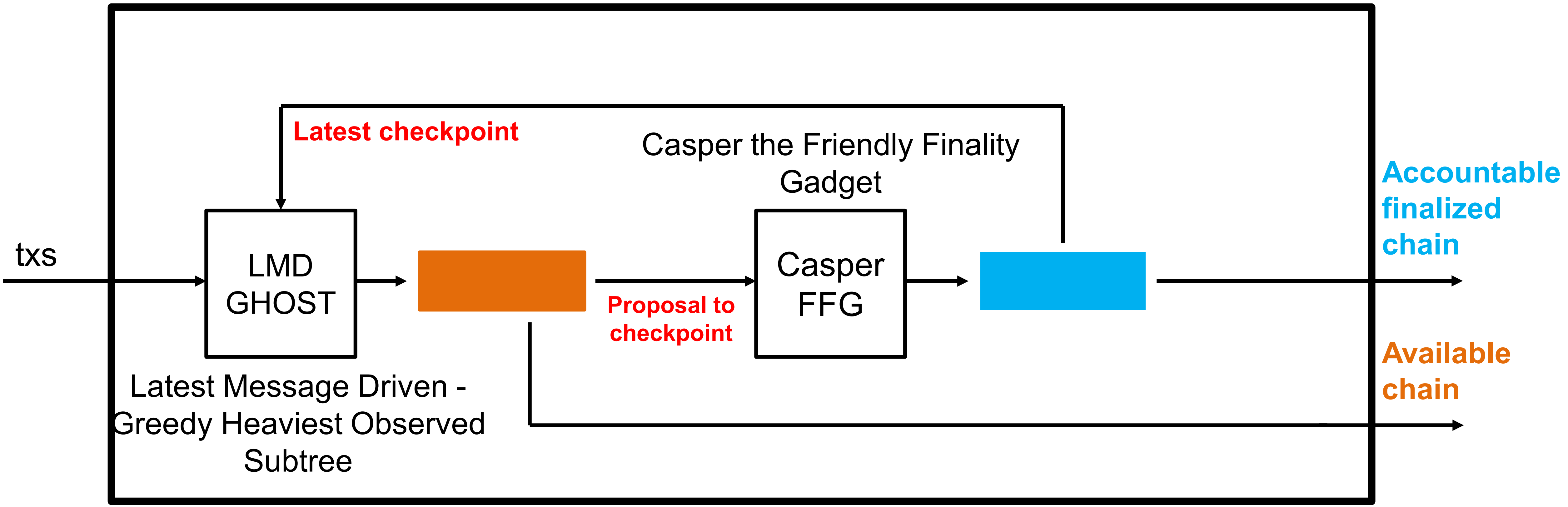
C votes “txs6”

D votes “txs6”



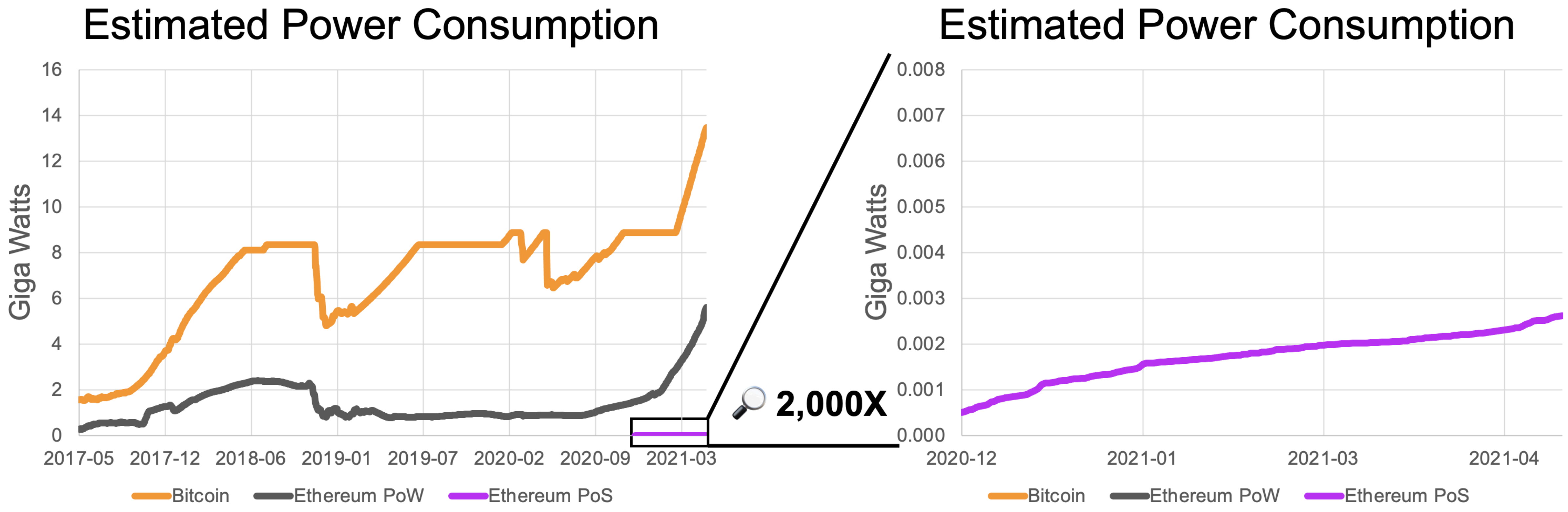
Finality: *Thanks to votes, checkpoints are safe even under asynchrony.*

Ethereum



Combining GHOST and Casper. (2020)

A Greener Future for Blockchains?



Bitcoin and Ethereum PoW data taken from Digiconomist

Taken from the article "Ethereum's energy usage will soon decrease by ~99.95%" that appeared at the 'ethereum foundation blog' on May 18th 2021.

Ethereum: mechanics

New topic: limitations of Bitcoin

Recall: UTXO contains (hash of) ScriptPK

- simple script: indicates conditions when UTXO can be spent

Limitations:

- Difficult to maintain state in multi-stage contracts
- Difficult to enforce global rules on assets

A simple example: rate limiting. My wallet manages 100 UTXOs.

- Desired policy: can only transfer 2BTC per day out of my wallet

An example: DNS

Domain name system on the blockchain: [google.com → IP addr]

Need support for three operations:

- **Name.new**(OwnerAddr, DomainName): intent to register
- **Name.update**(DomainName, newVal, newOwner, OwnerSig)
- **Name.lookup**(DomainName)

Note: also need to ensure no front-running on **Name.new**()

A broken implementation

Name.new() and Name.upate() create a UTXO with ScriptPK:

DUP HASH256 <OwnerAddr> **EQVERIFY CHECKSIG VERIFY**
<DNS> <DomainName> <IPaddr> <1>

only owner can “spend” this UTXO to update domain data

verify
sig is valid

ensure top
of stack is 1

Contract: (should be enforced by miners)

if domain google.com is registered,
no one else can register that domain

Problem: this contract cannot be enforced using Bitcoin script

What to do?

NameCoin: a fork of Bitcoin that implements this contract

(see also the Ethereum Name Service -- ENS)

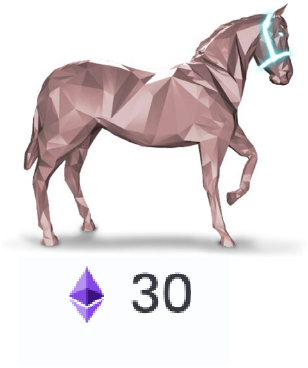
Can we build a blockchain that natively supports generic contracts like this?

$\Rightarrow$ Ethereum

Ethereum: enables a world of applications

A world of Ethereum Decentralized apps (DAPPs)

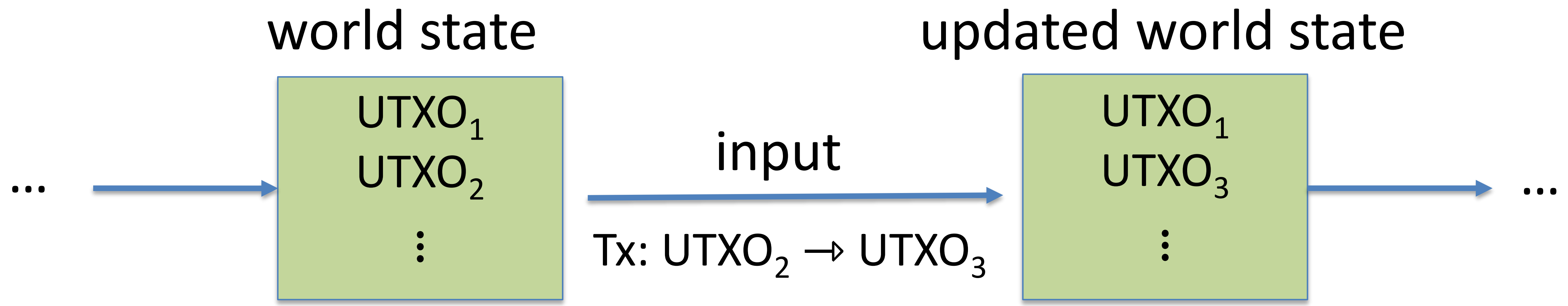
- New coins: ERC-20 standard interface
- **DeFi**: exchanges, lending, stablecoins, derivatives, etc.
- **Insurance**
- **DAOs**: decentralized organizations
- **NFTs**: Managing asset ownership (ERC-721 interface)



⋮

stateofthedapps.com, dapp.review

Bitcoin as a state transition system



Bitcoin rules: $\boxed{F_{bitcoin} : S \times I \rightarrow S}$

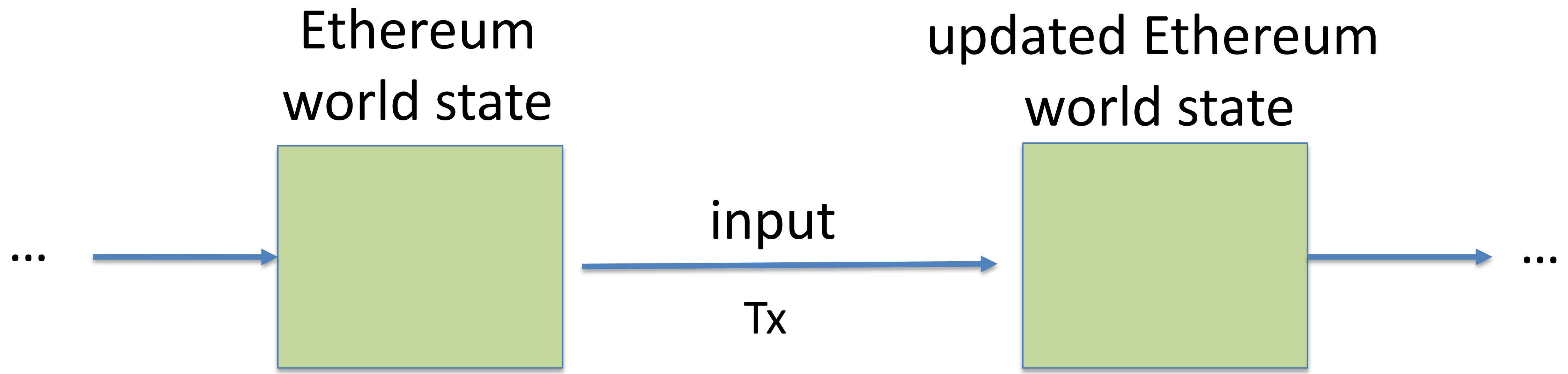
S: set of all possible world states, $s_0 \in S$ genesis state

I: set of all possible inputs

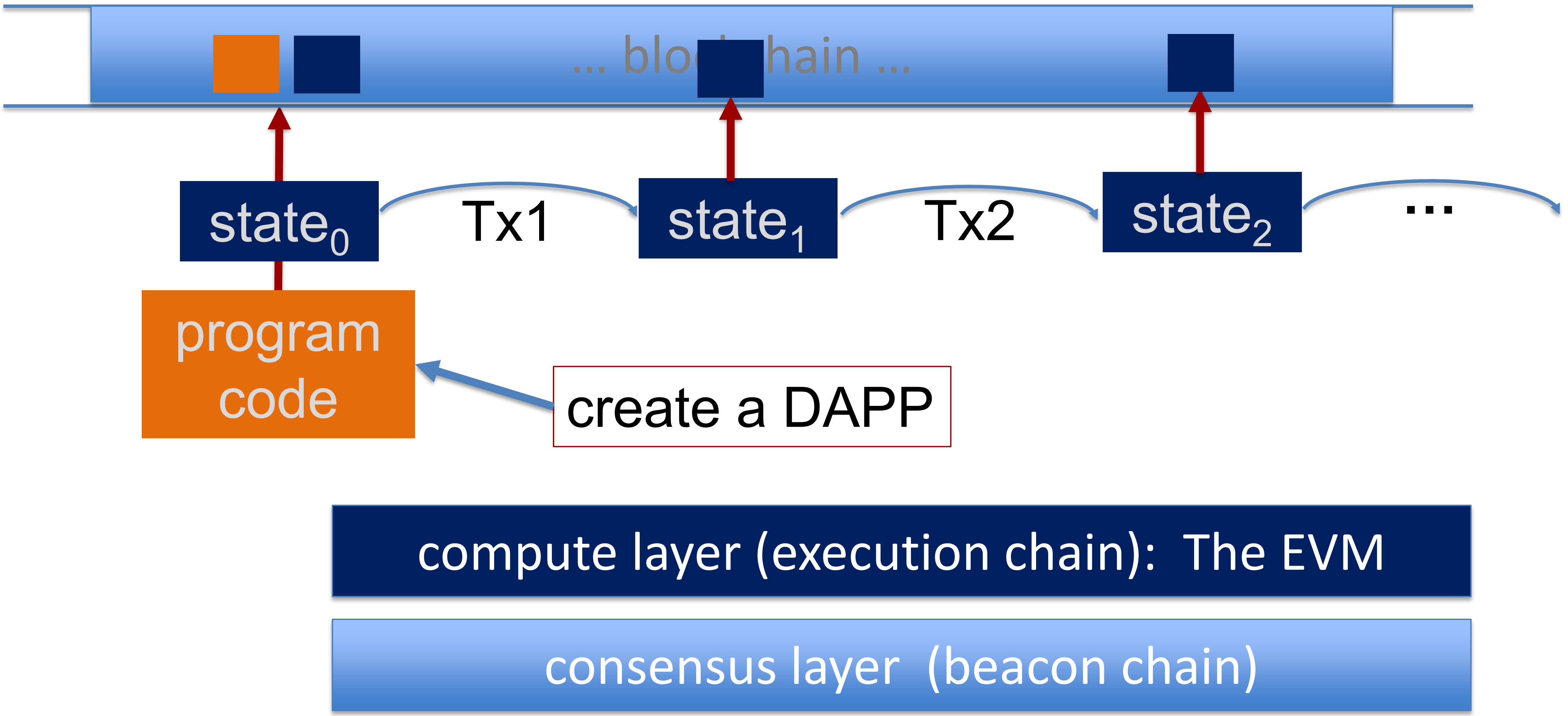
Ethereum as a state transition system

Much richer state transition functions

$\Rightarrow$ one transition executes an entire program



Running a program on a blockchain (DAPP)



The Ethereum system

Proof-of-Stake consensus

Block	Age	Txn	Fee Recipient
15764027	4 secs ago	91	Fee Recipient: 0x467...263
15764026	16 secs ago	26	0xedc7ec654e305a38ffff...
15764025	28 secs ago	165	bloXroute: Max Profit Bui...
15764024	40 secs ago	188	Lido: Execution Layer Re...
15764023	52 secs ago	18	Fee Recipient: 0xeBe...Acf
15764022	1 min ago	282	0xd4e96ef8eee8678dbff...
15764021	1 min ago	295	0xbb3afde35eb9f5feb53...
15764020	1 min ago	71	Fee Recipient: 0x6d2...766

One block every 12 seconds.

about 150 Tx per block.

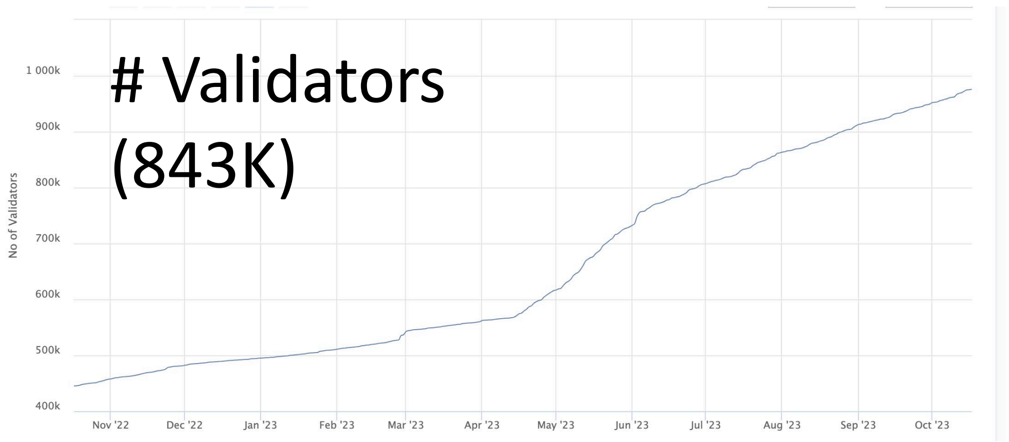
Block proposer receives
Tx fees for block
(along with other rewards)

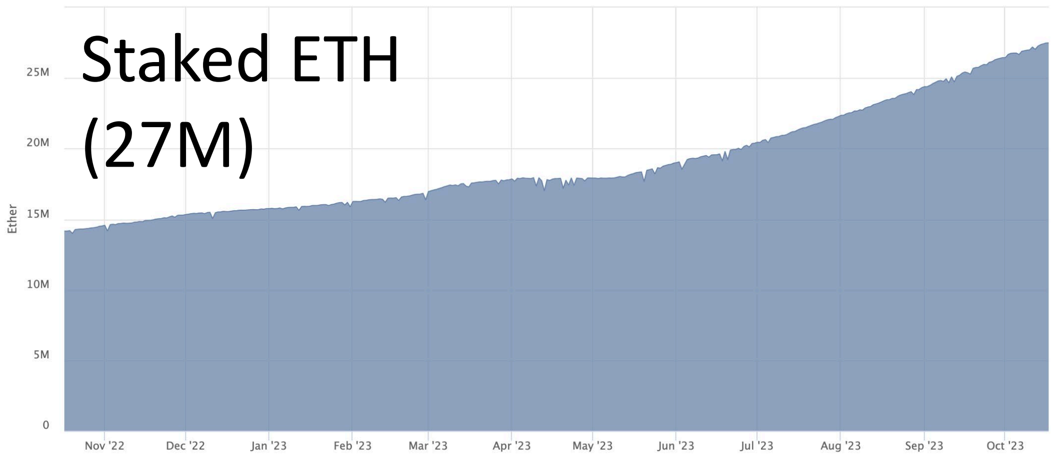
A bit about the beacon chain (Eth2 consensus layer)

To become a validator: stake (lock up) 32 ETH … or use Lido.

Validators: - sign blocks to express correctness (finalized once enough sigs)

- occasionally act as ***block proposer*** (chosen at random)
- correct behavior ⇒ issued **new ETH** every epoch (32 blocks)
- incorrect behavior ⇒ slashed (lots of details)





The economics of staking

Validator locks up 32 ETH.   Oct 2023: 27M ETH staked (total)

Annual validator income (an example):

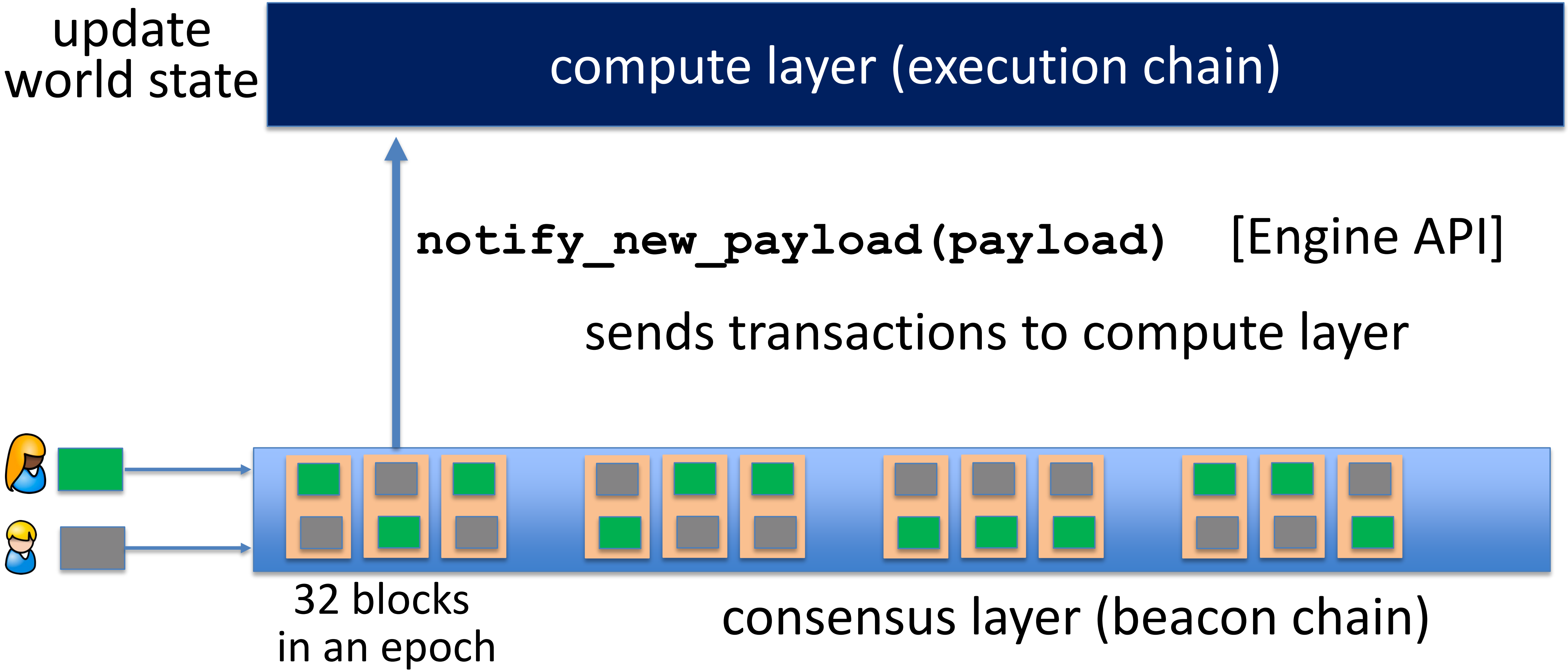
- Issuance: 1.0 ETH
- Tx fees: 0.4 ETH
- MEV: 0.4 ETH
- Total: 1.8 ETH (5.6% return on 32 ETH staked)

Can be adjusted
(BASE_REWARD_FACTOR)

A function of congestion

In practice: staking provider (e.g., Lido) takes a cut of the returns

The Ethereum system



The Ethereum Compute Layer:
The EVM

Ethereum compute layer: the EVM

World state: set of accounts identified by 32-byte address.

Two types of accounts:

(1) externally owned accounts (EOA):

controlled by ECDSA signing key pair (pk,sk).

sk: signing key known only to account owner

(2) contracts: controlled by code.

code set at account creation time, does not change

Data associated with an account

Account data	Owned (EOA)	Contracts	
address (computed):	H(pk)	H(CreatorAddr, CreatorNonce)	
code:	$\perp$	CodeHash	
storage root (state):	$\perp$	StorageRoot	
balance (in Wei):	balance	balance	(1 Wei = 10^{-18} ETH)
nonce:	nonce	nonce	

(#Tx sent) + (#accounts created): anti-replay mechanism

Account state: persistent storage

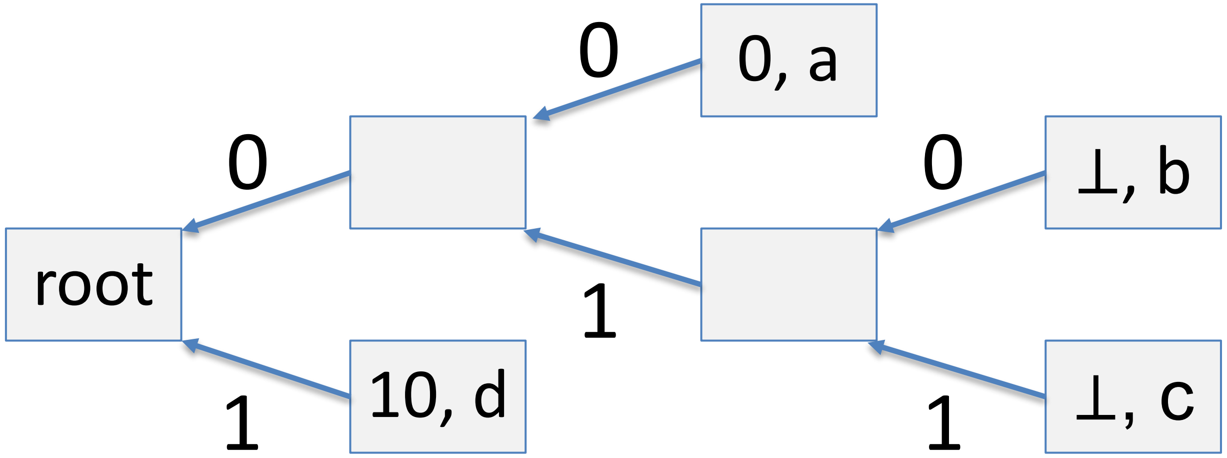
Every contract has an associated **storage array S**[]:

S[0], S[1], … , S[2^{256}-1]: each cell holds 32 bytes, init to 0.

Account storage root: **Merkle Patricia Tree hash** of S[]

- Cannot compute full Merkle tree hash: 2^{256} leaves

S[000] = a
S[010] = b
S[011] = c
S[110] = d



time to compute root hash:
≤ 2 × |S|

|S| = # non-zero cells

State transitions: Tx and messages

Transactions: signed data by initiator

- **To:** 32-byte address of target (0 → create new account)
- **From**, [**Signature**]: initiator address and signature on Tx (if owned)
- **Value**: # Wei being sent with Tx (1 Wei = 10^{-18} ETH)
- Tx fees (EIP 1559): **gasLimit, maxFee, maxPriorityFee** (later)
- if To = 0: create new contract **code = (init, body)**
- if To ≠ 0: **data** (what function to call & arguments)
- **nonce**: must match current nonce of sender (prevents Tx replay)
- chain_id: ensures Tx can only be submitted to the intended chain

State transitions: Tx and messages

Transaction types:

owned → owned: transfer ETH between users

owned → contract: call contract with ETH & data

Example (block #10993504)

From		To	msg.value	Tx fee (ETH)
0xa4ec1125ce9428ae5…	→	0x2cebe81fe0dcd220e…	0 Ether	0.00404405
0xba272f30459a119b2…	→	Uniswap V2: Router 2	0.14 Ether	0.00644563
0x4299d864bbda0fe32…	→	Uniswap V2: Router 2	89.839104111882671 Ether	0.00716578
0x4d1317a2a98cfea41…	→	0xc59f33af5f4a7c8647…	14.501 Ether	0.001239
0x29ecaa773f052d14e…	→	CryptoKitties: Core	0 Ether	0.00775543
0x63bb46461696416fa…	→	Uniswap V2: Router 2	0.203036474328481 Ether	0.00766728
0xde70238aef7a35abd…	→	Balancer: ETH/DOUGH…	0 Ether	0.00261582
0x69aca10fe1394d535f…	→	0x837d03aa7fc09b8be…	0 Ether	0.00259936
0xe2f5d180626d29e75…	→	Uniswap V2: Router 2	0 Ether	0.00665809

Messages: virtual Tx initiated by a contract

Same as Tx, but no signature (contract has no signing key)

contract → owned: contract sends funds to user

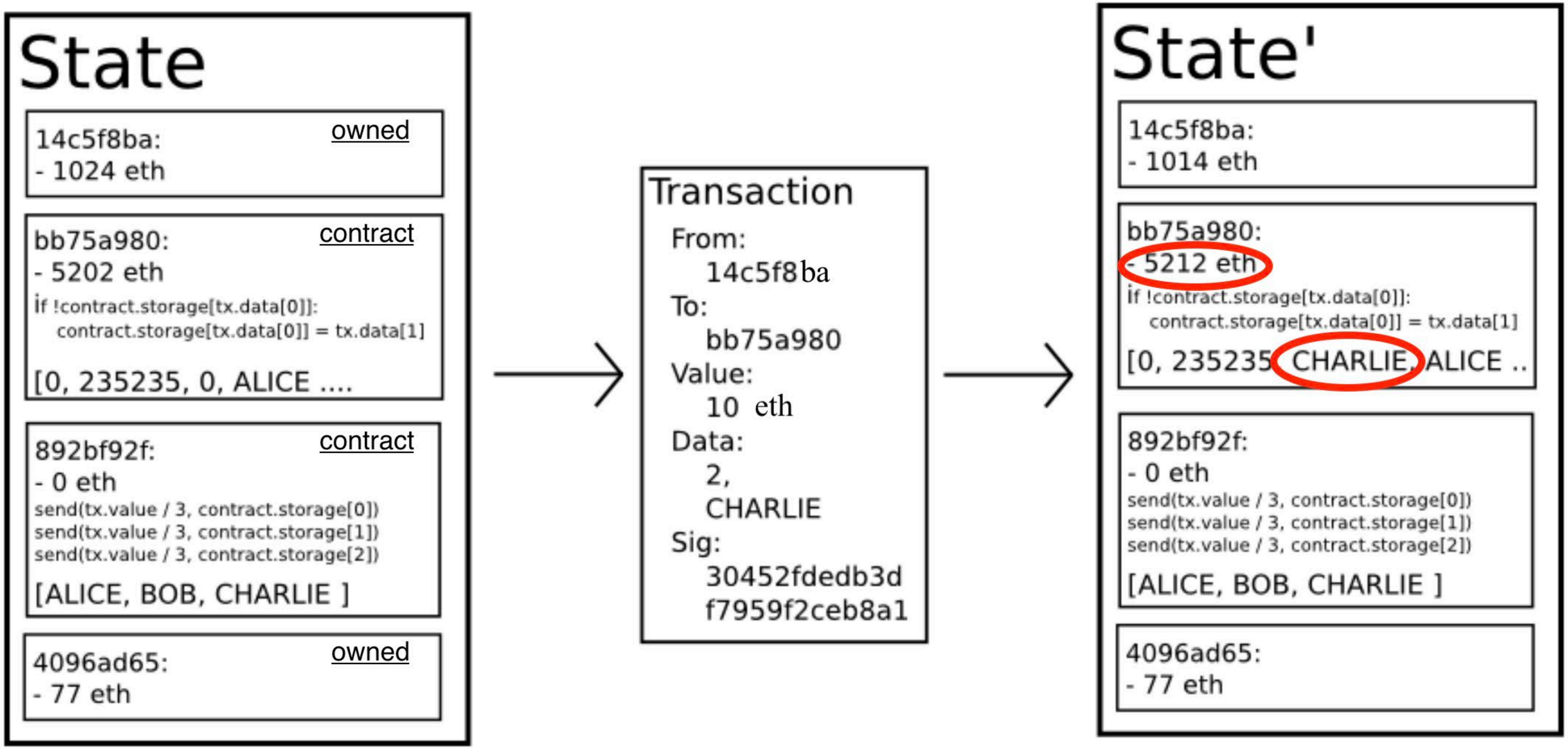
contract → contract: one program calls another (and sends funds)

One Tx from user: can lead to many Tx processed. Composability!

Tx from owned addr → contract → another contract

another contract → different owned

Example Tx



world state (four accounts)

updated world state

An Ethereum Block

Block proposer creates a block of n Tx: (from Txs submitted by users)

- To produce a block do:
    - for i=1,…,n: execute state change of Tx_i sequentially
    (can change state of >n accounts)
    - record updated world state in block

Other validators re-execute all Tx to verify block ⇒
sign block if valid ⇒ enough sigs, epoch is finalized.

Block header data (simplified)

(1) consensus data: proposer ID, parent hash, votes, etc.

(2) address of gas beneficiary: where Tx fees will go

(3) world state root: updated world state

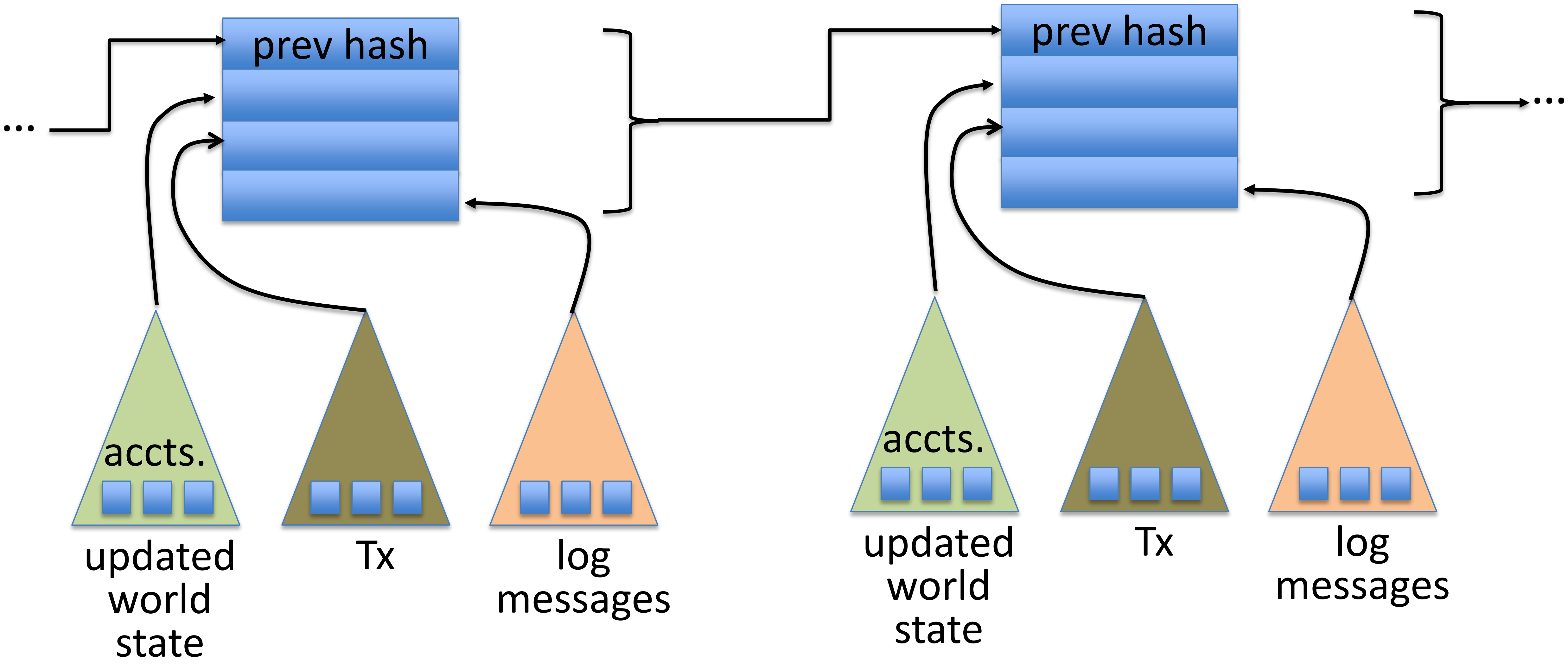
Merkle Patricia Tree hash of all accounts in the system

(4) **Tx root**: Merkle hash of all Tx processed in block

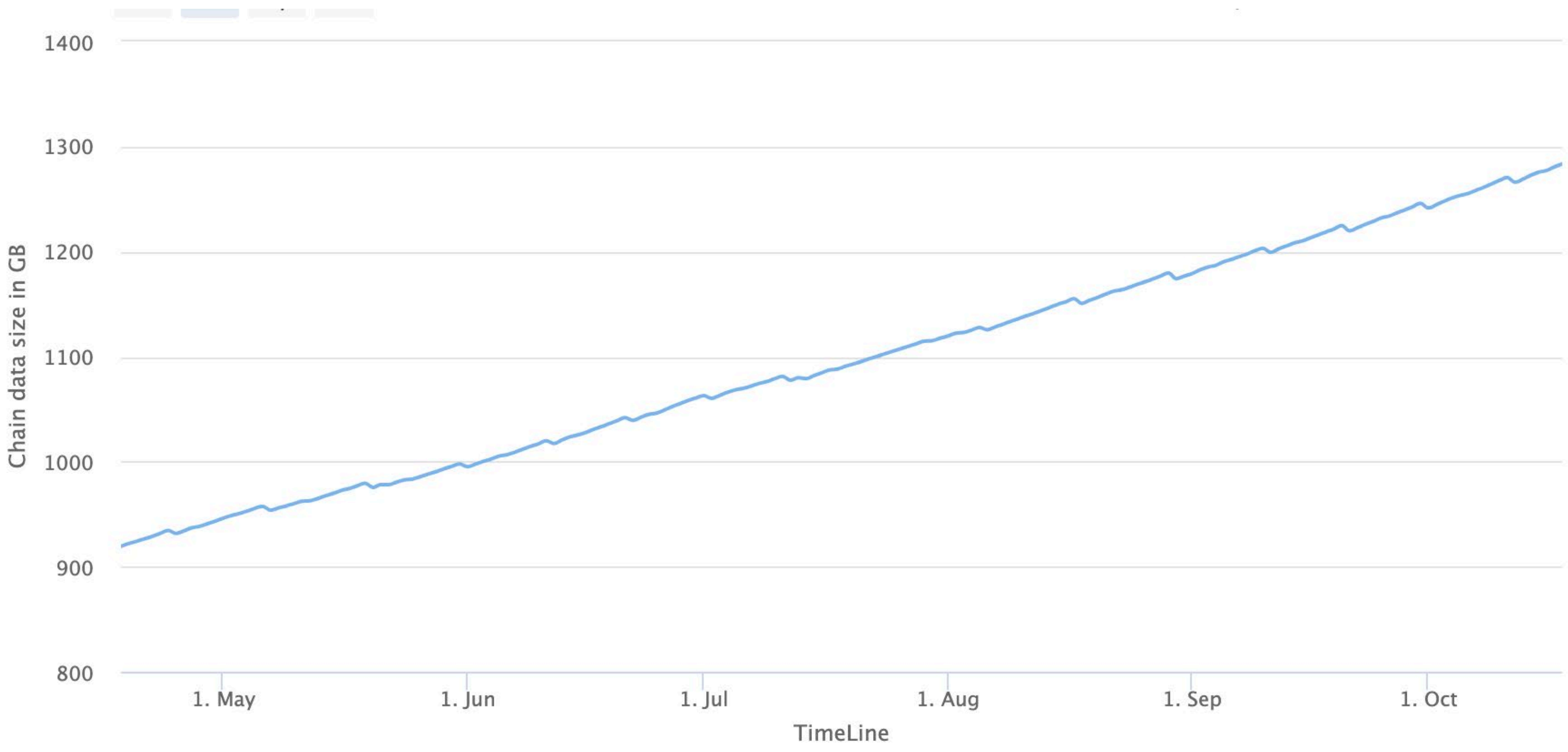
(5) **Tx receipt root**: Merkle hash of log messages generated in block

(5) Gas used: used to adjust gas price (target 15M gas per block)

The Ethereum blockchain: abstractly



Amount of memory to run a node



≈1.3 TB

ETH total blockchain size (archival): 16 TB (Oct. 2023)

An example contract: NameCoin

```
contract nameCoin {          // Solidity code   (next lecture)

    struct nameEntry {
        address owner;       // address of domain owner
        bytes32 value;       // IP address
    }

    // array of all registered domains
    mapping (bytes32 => nameEntry)  data;
```

An example contract: NameCoin

```
function nameNew(bytes32 name) {

    // registration costs is 100 Wei

    if (data[name] == 0   &&   msg.value >= 100) {
        data[name].owner = msg.sender    // record domain owner
        emit Register(msg.sender, name)   // log event
}}
```

Code ensures that no one can take over a registered name

Serious bug in this code!   Front running.   Solved using commitments.

An example contract: NameCoin

```
function nameUpdate(
                bytes32 name, bytes32 newValue, address newOwner) {

  // check if message is from domain owner,
  //            and update cost of 10 Wei is paid

  if (data[name].owner == msg.sender   &&   msg.value >= 10) {

        data[name].value = newValue;        // record new value
        data[name].owner = newOwner;        // record new owner
 }}}
```

An example contract: NameCoin

```
    function nameLookup(bytes32 name) {
        return data[name];
    }

} // end of contract
```

Used by other contracts

Humans do not need this
(use etherscan.io)

EVM mechanics: execution environment

Write code in Solidity (or another front-end language)

⇒ compile to EVM bytecode

(some projects use WASM or BPF bytecode)

⇒ validators use the EVM to execute contract bytecode

in response to a Tx

The EVM

Stack machine (like Bitcoin) but with JUMP

- max stack depth = 1024
- program aborts if stack size exceeded; block proposer keeps gas
- contract can <u>create</u> or <u>call</u> another contract

In addition: two types of zero initialized memory

- Persistent storage (on blockchain):   SLOAD, SSTORE   (expensive)
- Volatile memory (for single Tx):   MLOAD, MSTORE   (cheap)
- LOG0(data): write data to log

see https://www.evm.codes

Every instruction costs gas, examples:

SSTORE addr (32 bytes), **value** (32 bytes)

- zero → non-zero: 20,000 gas
- non-zero → non-zero: 5,000 gas (for a cold slot)
- non-zero → zero: 15,000 gas refund (example)

Refund is given for reducing size of blockchain state

CREATE : 32,000 + 200 × (code size) gas;

CALL **gas**, addr, **value**, args

SELFDESTRUCT addr: kill current contract (5000 gas)

Gas calculation

Why charge gas?

- Tx fees (gas) prevents submitting Tx that runs for many steps.
- During high load: block proposer chooses Tx from mempool that maximize its income.

Old EVM: (prior to EIP1559, live on 8/2021)

- Every Tx contains a gasPrice ``bid'' (gas → Wei conversion price)
- Producer chooses Tx with highest gasPrice (max sum(gasPrice × gasLimit))

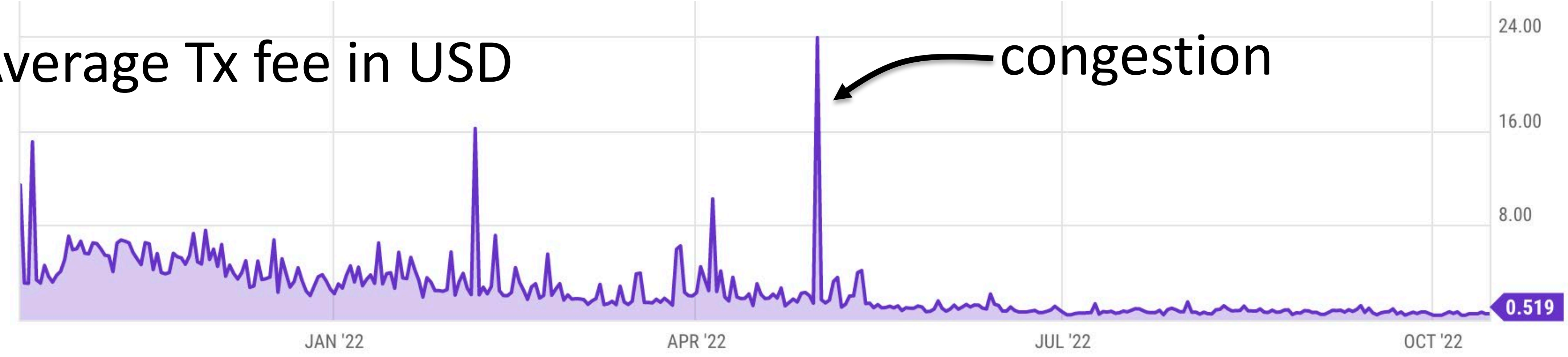
$\Longrightarrow$ not an efficient auction mechanism (first price auction)

Gas prices spike during congestion

GasPrice in Gwei:

86 Gwei = 86×10^{-9} ETH

Average Tx fee in USD



Gas calculation: EIP1559 (since 8/2021)

EIP1559 goals (informal):

- users incentivized to bid their true utility for posting Tx,
- block proposer incentivized to not create fake Tx, and
- disincentivize off chain agreements.

[Transaction Fee Mechanism Design, by T. Roughgarden, 2021]

Gas calculation: EIP1559

Every block has a “baseFee”:

the **minimum** gasPrice for all Tx in the block

baseFee is computed from total gas in earlier blocks:

- earlier blocks at gas limit (30M gas) $\Longrightarrow$ base fee goes up 12.5%
- earlier blocks empty $\Longrightarrow$ base fee decreases by 12.5%

interpolate in between

If earlier blocks at “target size” (15M gas) $\Longrightarrow$ base fee does not change

Gas calculation

EIP1559 Tx specifies three parameters:

- **gasLimit**: max total gas allowed for Tx
- **maxFee**: maximum allowed gas price (max gas → Wei conversion)
- **maxPriorityFee**: additional "tip" to be paid to block proposer

Computed **gasPrice** bid:

gasPrice ← min(**maxFee**, **baseFee** + **maxPriorityFee**)

Max Tx fee: **gasLimit** × **gasPrice**

Gas calculation (informal)

gasUsed ← gas used by Tx

Send **gasUsed × (gasPrice – baseFee)** to block proposer

BURN **gasUsed × baseFee** 

⇒ total supply of ETH can decrease

Gas calculation

(1) if **gasPrice** < **baseFee**: abort

(2) If **gasLimit × gasPrice** < msg.sender.balance: abort

(3) deduct **gasLimit × gasPrice** from msg.sender.balance

---

(4) set **Gas** ⇽ **gasLimit**

(5) execute Tx: deduct gas from **Gas** for each instruction

if at end (**Gas** < 0): abort, Tx is invalid (proposer keeps **gasLimit × gasPrice**)

(6) Refund **Gas** × **gasPrice** to msg.sender.balance

---

(7) **gasUsed** ⇽ **gasLimit – Gas**

(7a) BURN **gasUsed × baseFee**

(7b) Send **gasUsed × (gasPrice – baseFee)** to block producer

Example baseFee and effect of burn

block #	gasUsed		baseFee (Gwei)	ETH burned
15763570	21,486,058	(<15M)	16.92	0.363
15763569	**14,609,185**		16.97	0.248
15763568	25,239,720	(<15M)	15.64	0.394
15763567	29,976,215	(<15M)	13.90	0.416
15763566	**14,926,172**		13.91	0.207
15763565	**1,985,580**		beacon chain 15.60	0.031

≈ gasUsed × baseFee

baseFee < 16Gwei ⇒ new issuance > burn ⇒ ETH inflates

baseFee > 16Gwei ⇒ new issuance < burn ⇒ ETH deflates

Why burn ETH ???

<u>Recall:</u> EIP1559 goals (informal)

- users incentivized to bid their true utility for posting Tx,
- block proposer incentivized to not create fake Tx, and
- disincentivize off chain agreements.

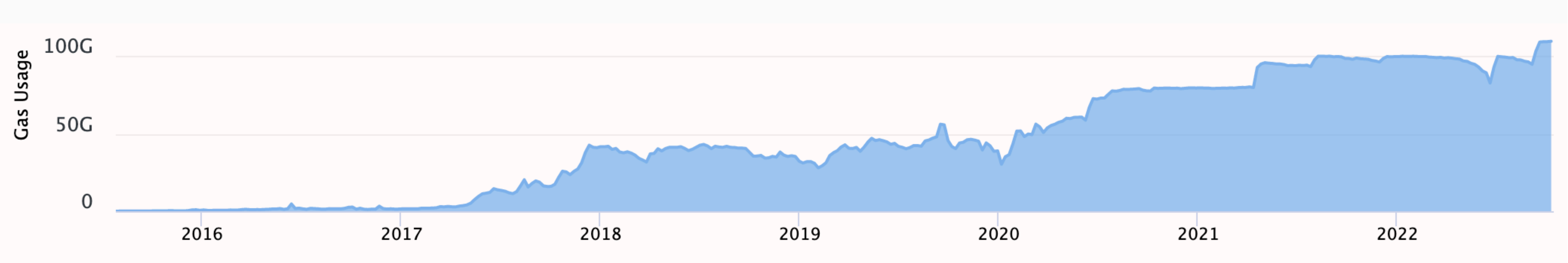
Suppose no burn (i.e., baseFee given to block producer):

$\Longrightarrow$ in periods of low Tx volume proposer would try to increase volume by offering to refund the baseFee *off chain* to users.

Note: transactions are becoming more complex

Total Gas Usage

Evolution of the total gas used by the Ethereum network per day



Gas usage is increasing $\Rightarrow$ each Tx takes more instructions to execute