KAIST School of Computing

Web3@KAIST

Building Web3 Apps to Solve Real Problems

Building Web3 & Blockchain Applications (CS492 Special Topics in Computer Science) Spring 2023

Blockchain 101: Ethereum

Lecture 4 (2023-03-15)

Min Suk Kang

Assistant Professor

School of Computing/Graduate School of Information Security

NetS&P Research Lab @ KAIST

KAIST

Agenda

- Digital currency
  - Why is it hard?
  - What properties should we achieve?
- Nakamoto consensus
  - How Bitcoin solved it?
- Ethereum as the world computer
  - Smart contracts
  - Proof of stake
- What’s more? (next week)

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

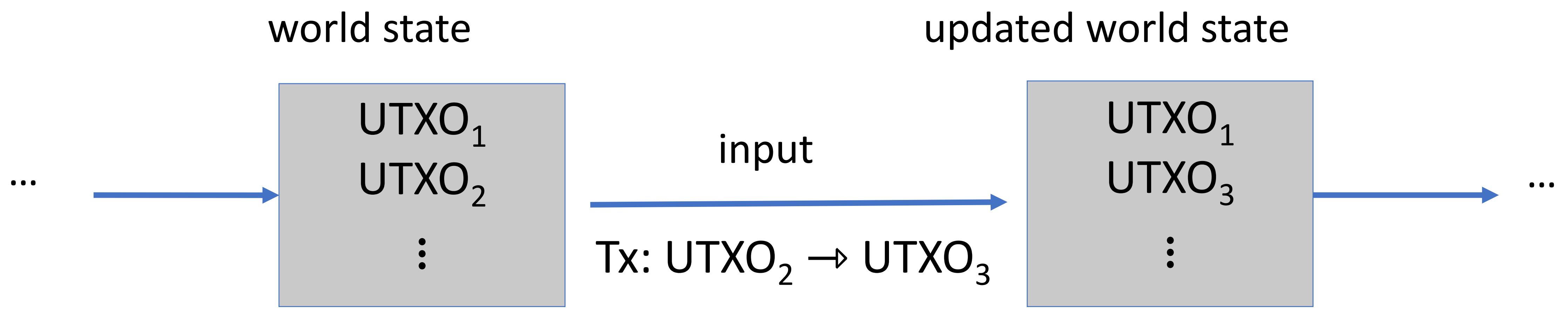
Ethereum: enables a world of applications

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)

⋮

Bitcoin as a state transition system



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

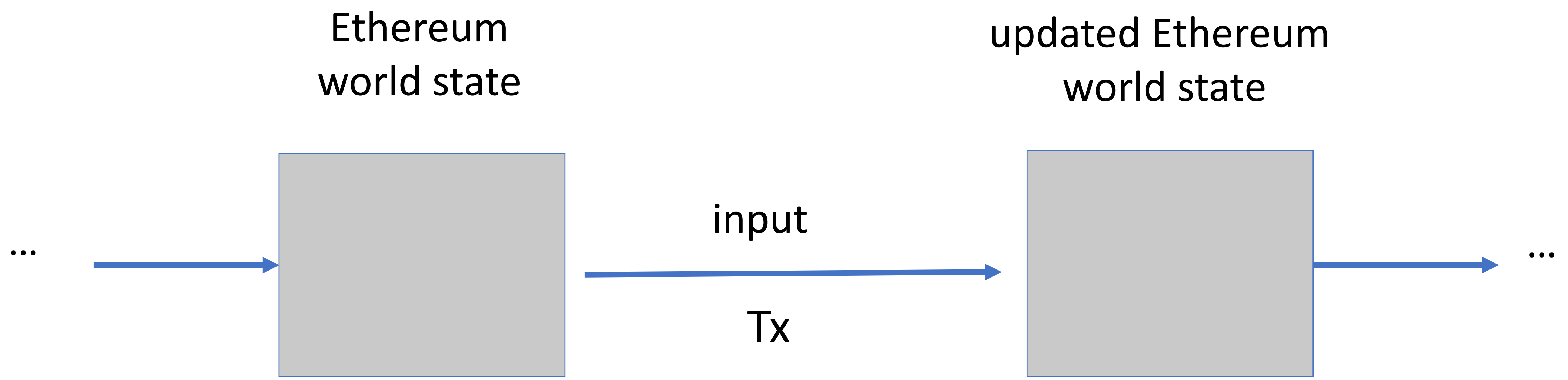
S: set of all possible world states, $s_0 \in S$ genesis state

I: set of all possible inputs

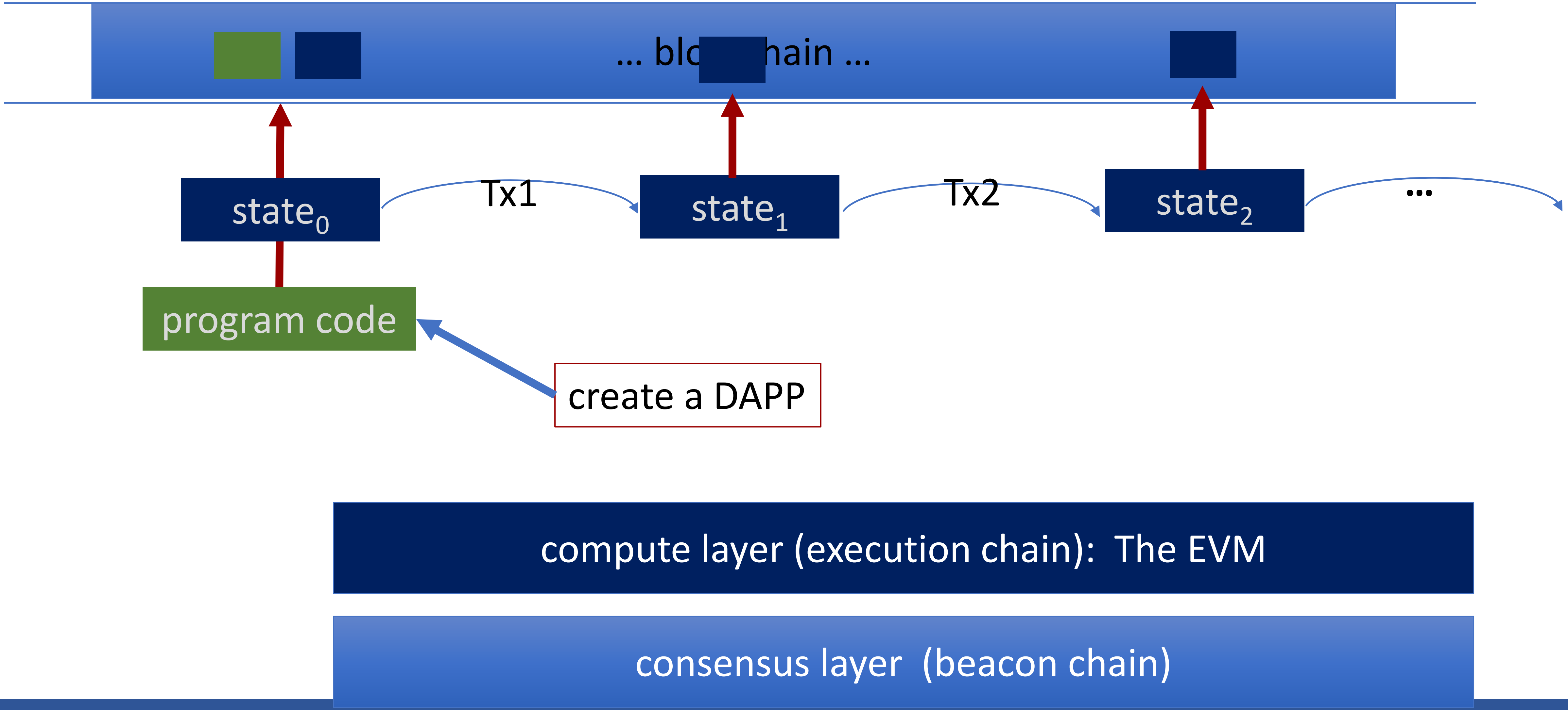
Ethereum as a state transition system

Much richer state transition functions

$\Rightarrow$ one transition executes an entire program



Running a program on a blockchain (DAPP)



The Ethereum system

- Ethereum 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)

Ethereum compute layer: the EVM

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

Two types of accounts:

(1) owned accounts: 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

Account state: persistent storage

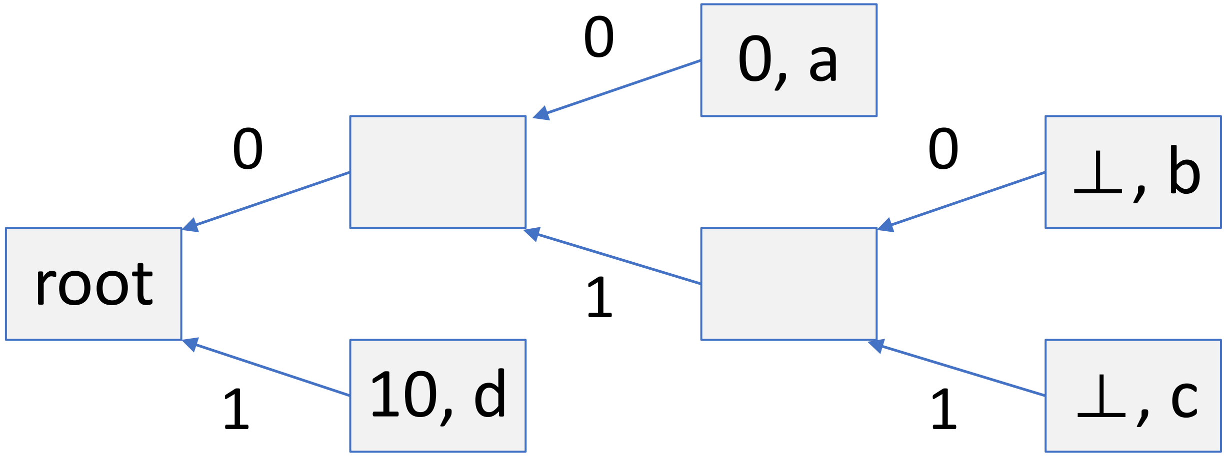
Every contract has an associated **storage array S**[]:

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:
$\leq 2 \times |S|$

$|S|$ = # 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
- 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

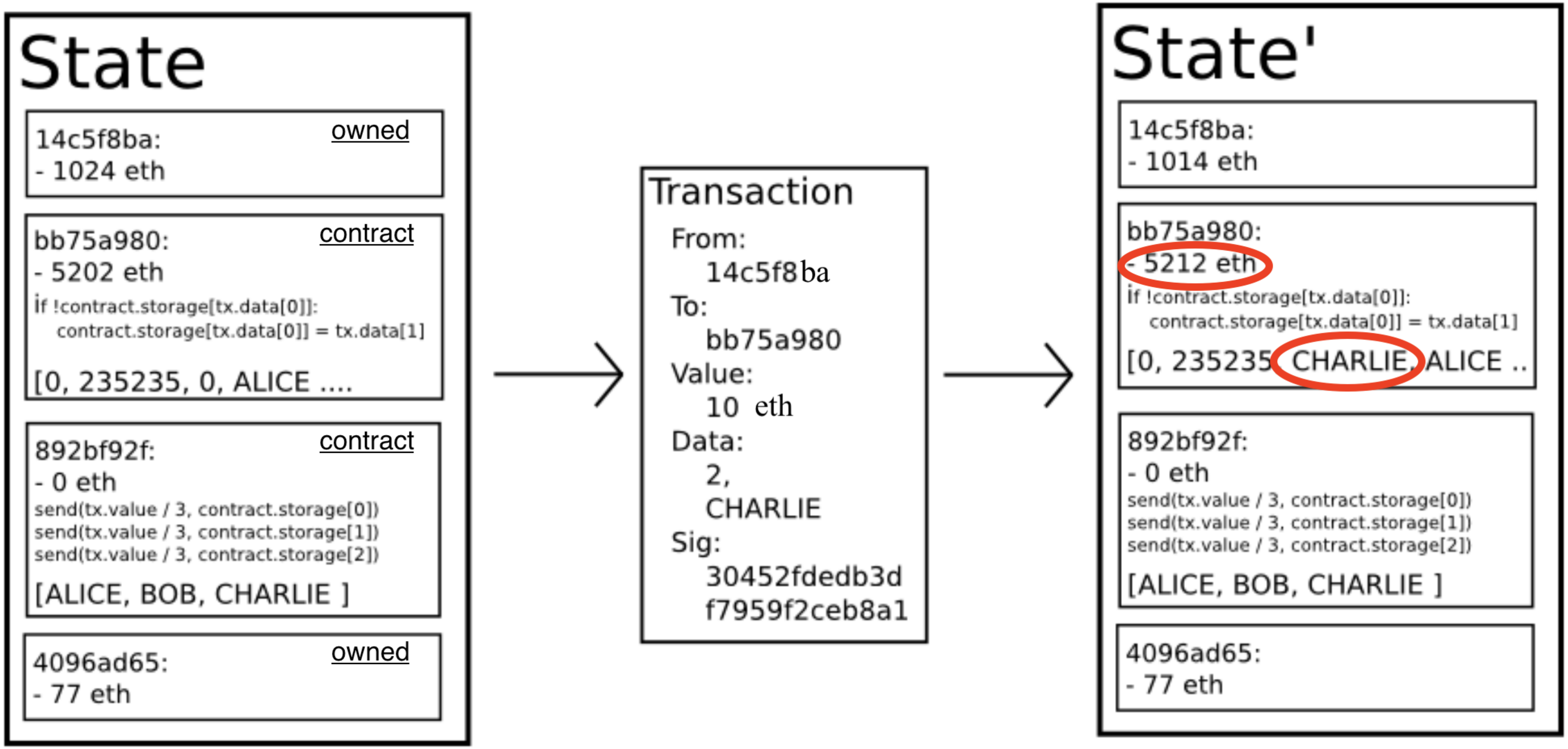
contract → contract: one program calls another (and sends funds)

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

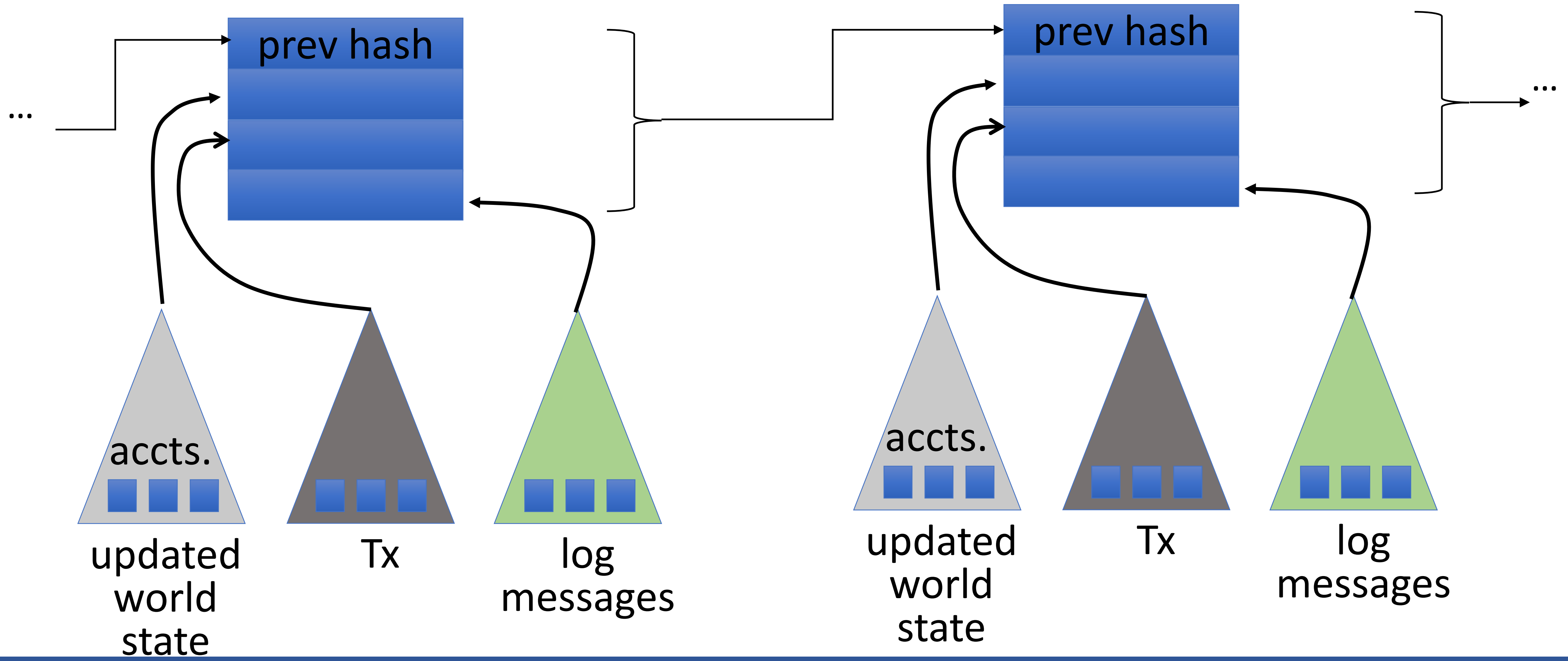
An Ethereum Block

Validators collect Txs from users $\Rightarrow$ proposer creates a block of n Tx

- 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

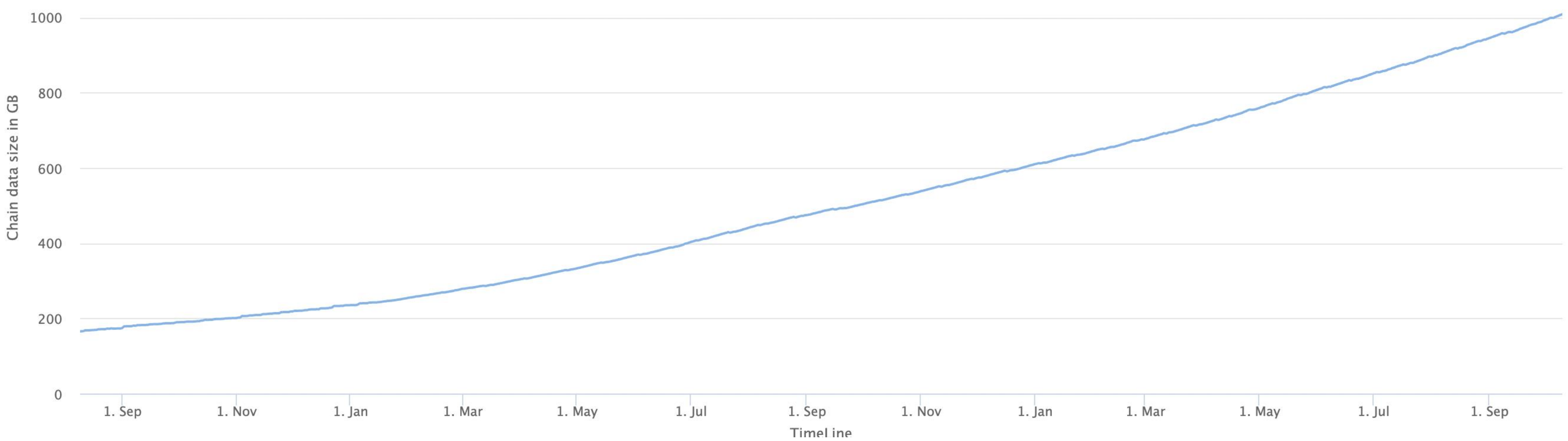
Other validators re-execute all Tx to verify block $\Rightarrow$
sign block if valid $\Rightarrow$ enough sigs, epoch is finalized.

The Ethereum blockchain: abstractly



Amount of memory to run a node

≈1 TB



ETH total blockchain size (archival): 12 TB (Oct. 2022)

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

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)

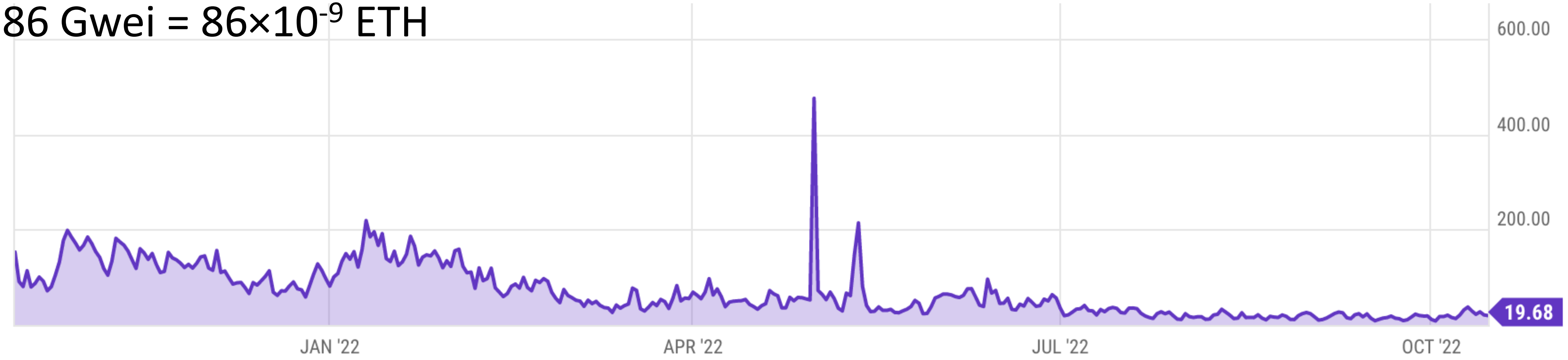
- Every Tx contains a gasPrice ``bid'' (gas → Wei conversion price)
- Producer chooses Tx with highest gasPrice (max sum(gasPrice×gasLimit))

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

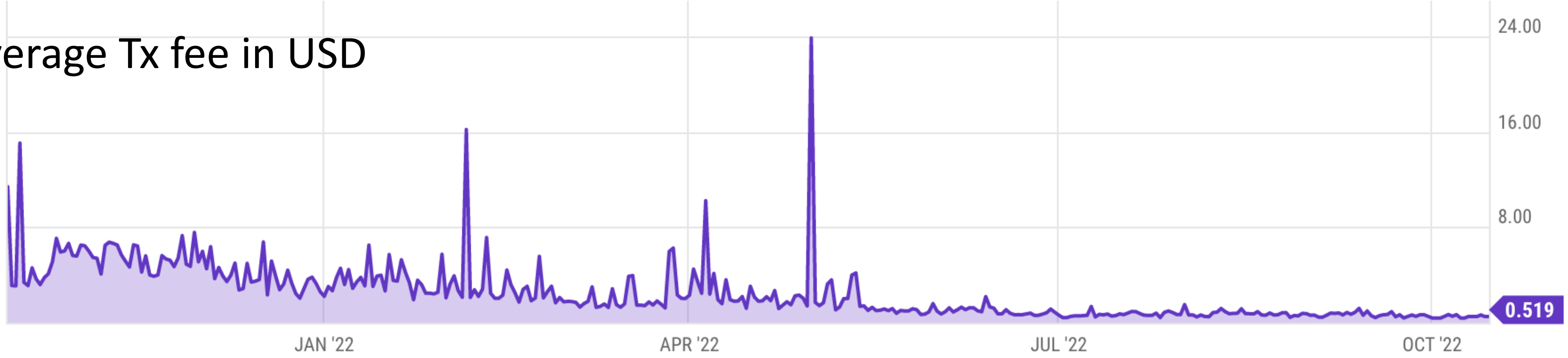
Gas prices spike during congestion

GasPrice in Gwei:

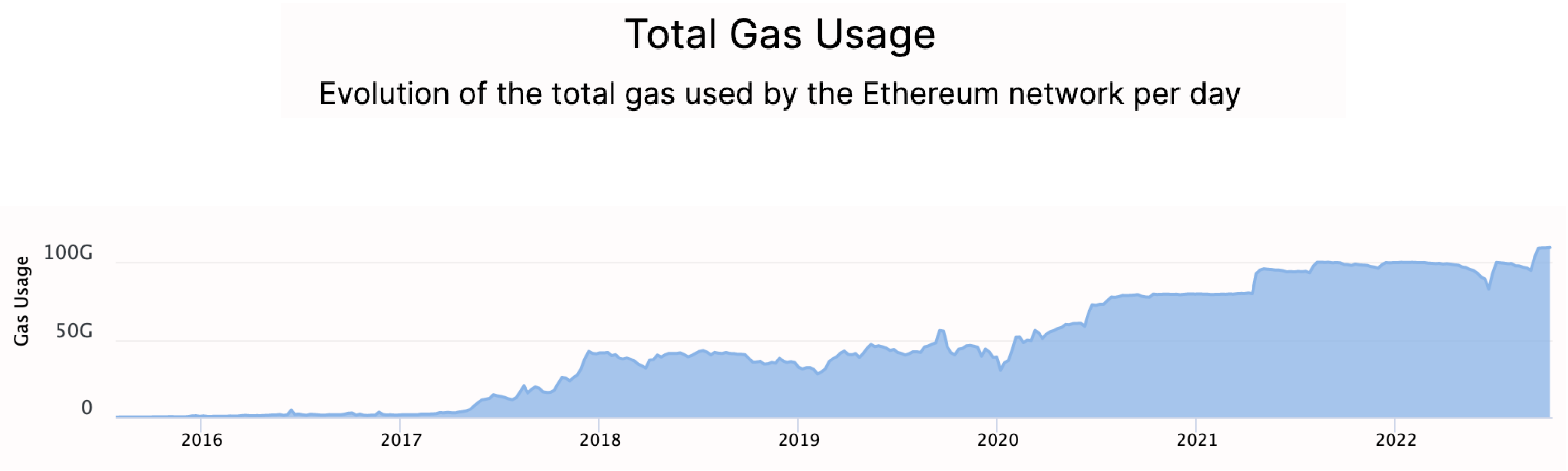
86 Gwei = 86×10^{-9} ETH



Average Tx fee in USD



Note: transactions are becoming more complex



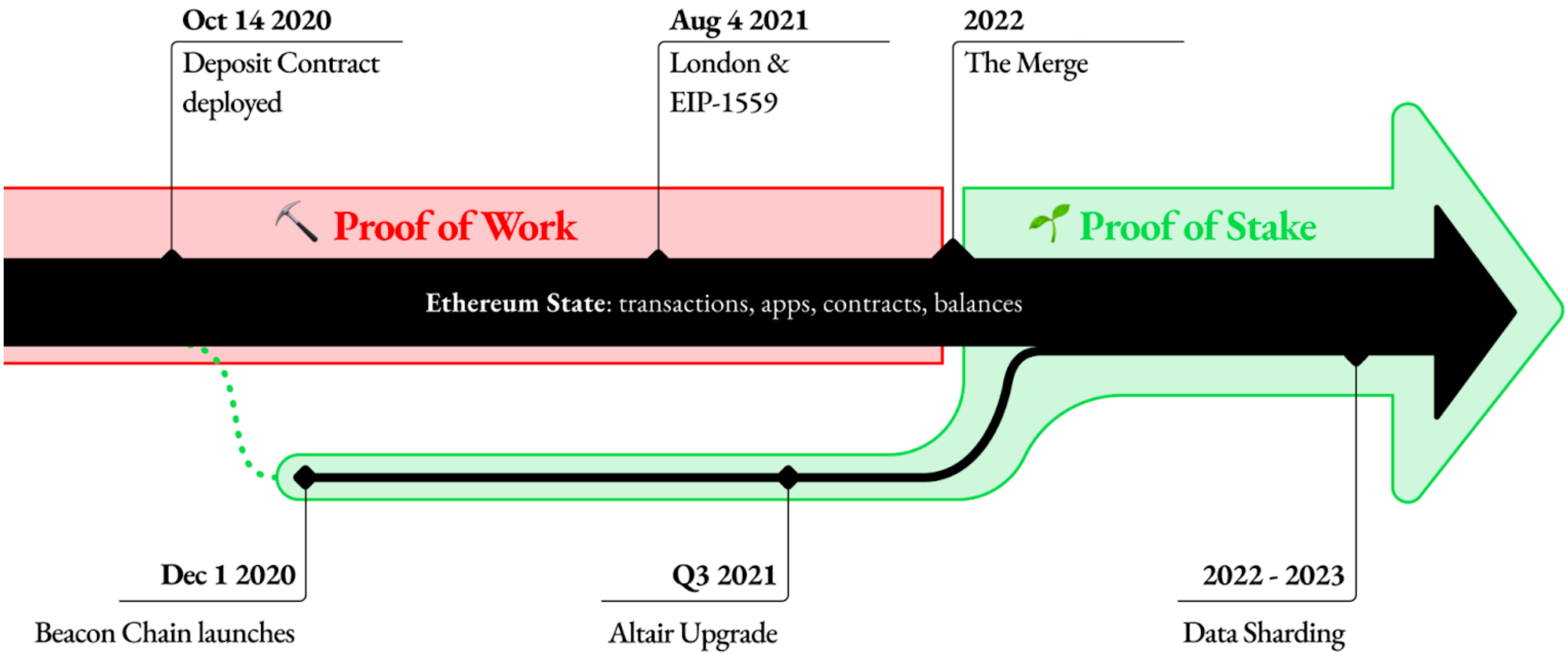
Gas usage is increasing $\Rightarrow$ each Tx takes more instructions to execute

Agenda

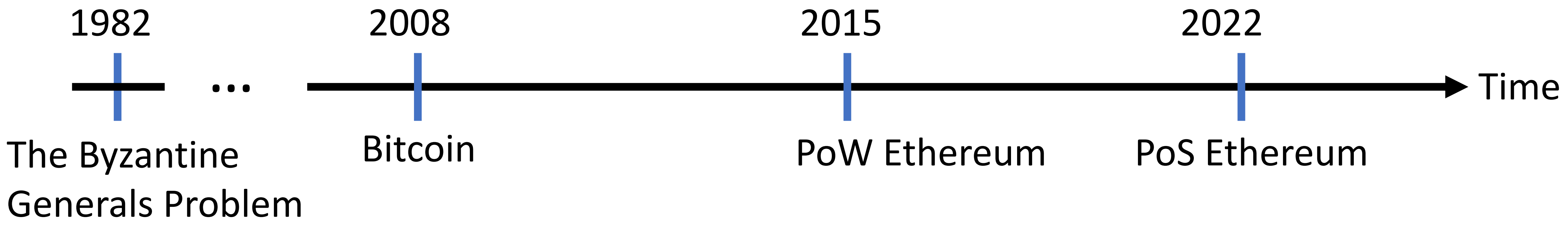
- Digital currency
  - Why is it hard?
  - What properties should we achieve?
- Nakamoto consensus
  - How Bitcoin solved it?
- Ethereum as the world computer
  - Smart contracts
  - Proof of stake
- What's more? (next week)

Ethereum's Upgrade Path

The Merge: when the existing PoW consensus is replaced by the Beacon Chain's PoS.
Graphic: @trent_vanepps, not "official," subject to change



From Bitcoin to Proof-of-Stake



Open Participation

- Dynamic availability
- Sybil resistance

Block rewards (carrot)

PoS Ethereum:

Open Participation

- Dynamic availability
- Sybil resistance

Block rewards (carrot)

Finality and accountable safety

Slashing (stick)

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

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

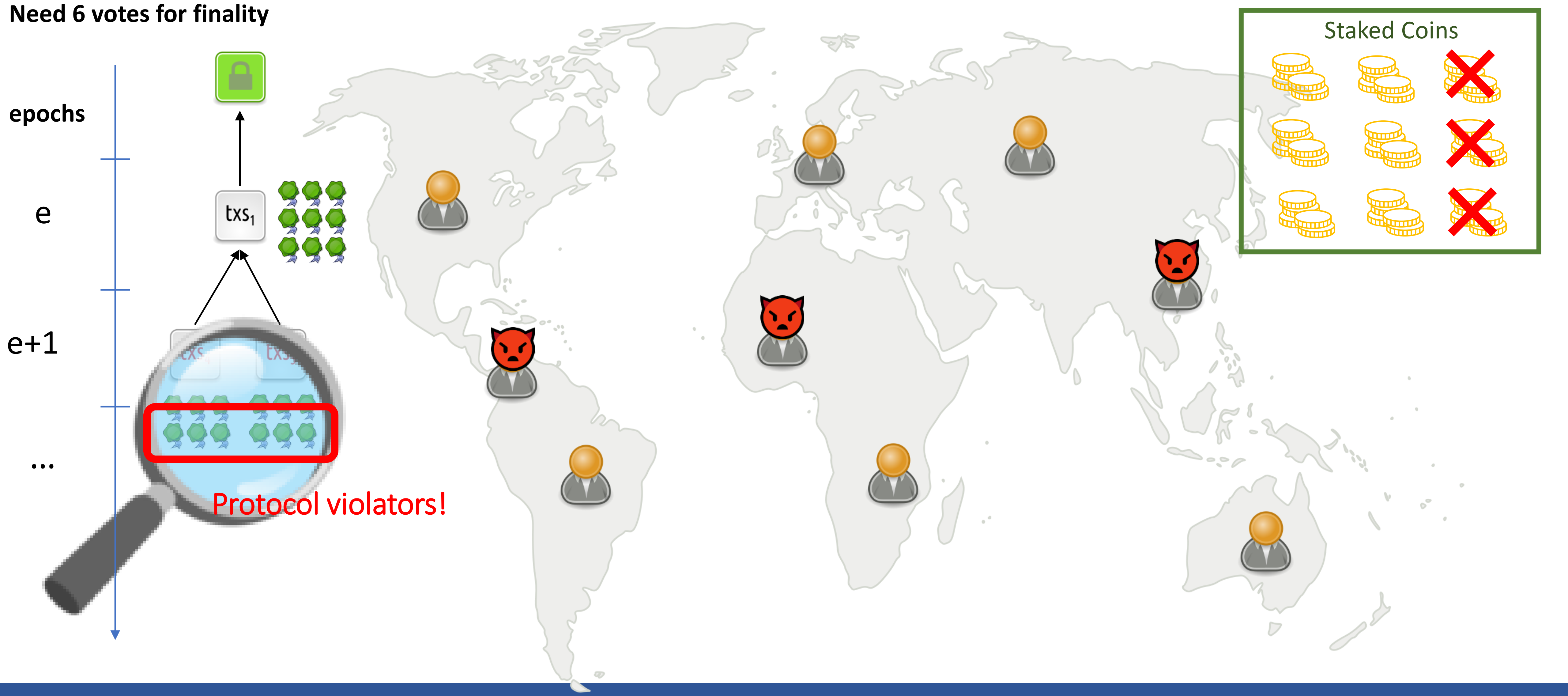
The more coins staked by a node…

- Higher the probability that the node is elected as a leader (recall Streamlet).
- Larger the weight of that node's vote.

If the node is caught doing an adversarial action (like voting for two conflicting blocks), 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 few words on Proof-of-Stake



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:** Streamlet under partial synchrony has resilience of *n/3.*

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

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

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

Accountable Safety

Accountable safety is a stronger notion than just security.

Number of adversary nodes (f)

0 — $n/3$ — $2n/3$ — 1

	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 ☺	

Another Property of PoS: Finality

- Most accountably safe protocol examples we have seen satisfy safety and liveness under partial synchrony.
  - This means these protocols preserve safety during periods of asynchrony (before GST).

- We say that a protocol provides *finality* if it preserves safety during periods of asynchrony.
  - **Example:** Streamlet provides *finality*.

- 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)!

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, e.g., due to a world-wide catastrophe.
- 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.
- Let’s focus on having dynamic availability and finality for now…

Holy Grail of Internet Scale Consensus

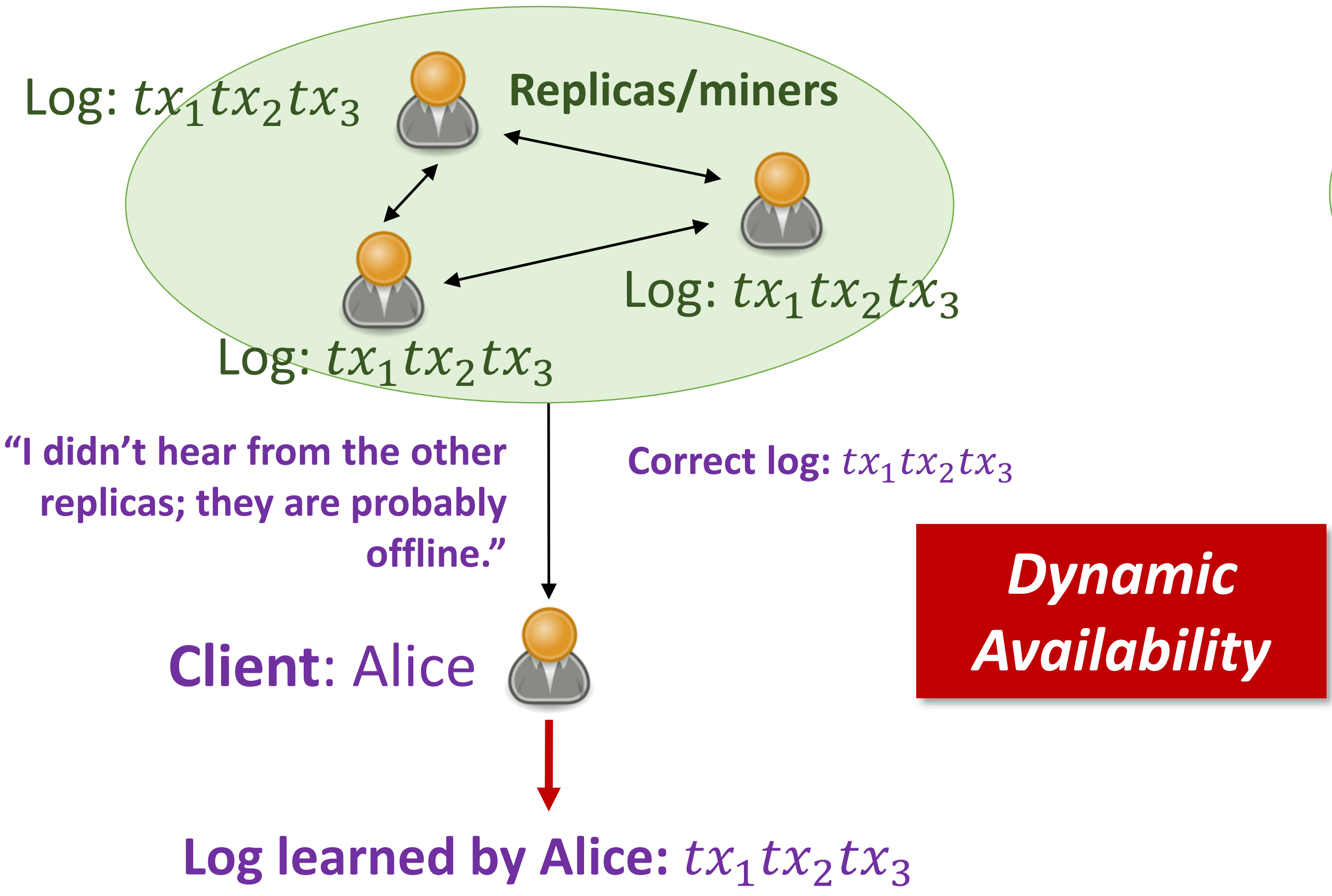
Is there a SMR protocol that provides both dynamic availability and finality?

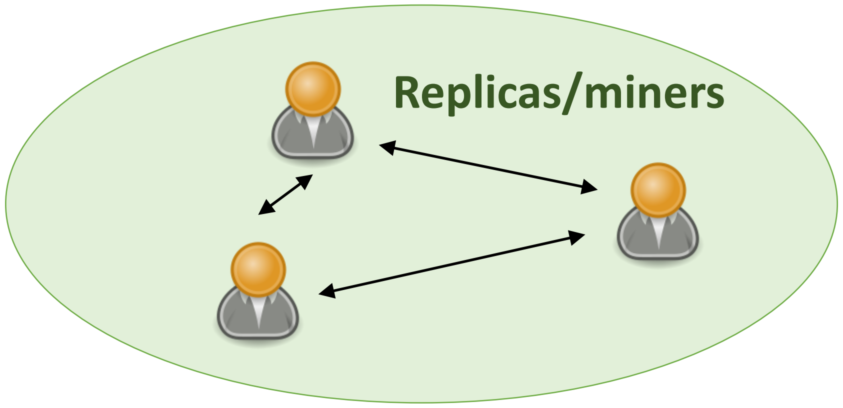
No!

Blockchain CAP Theorem

Blockchain CAP Theorem

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

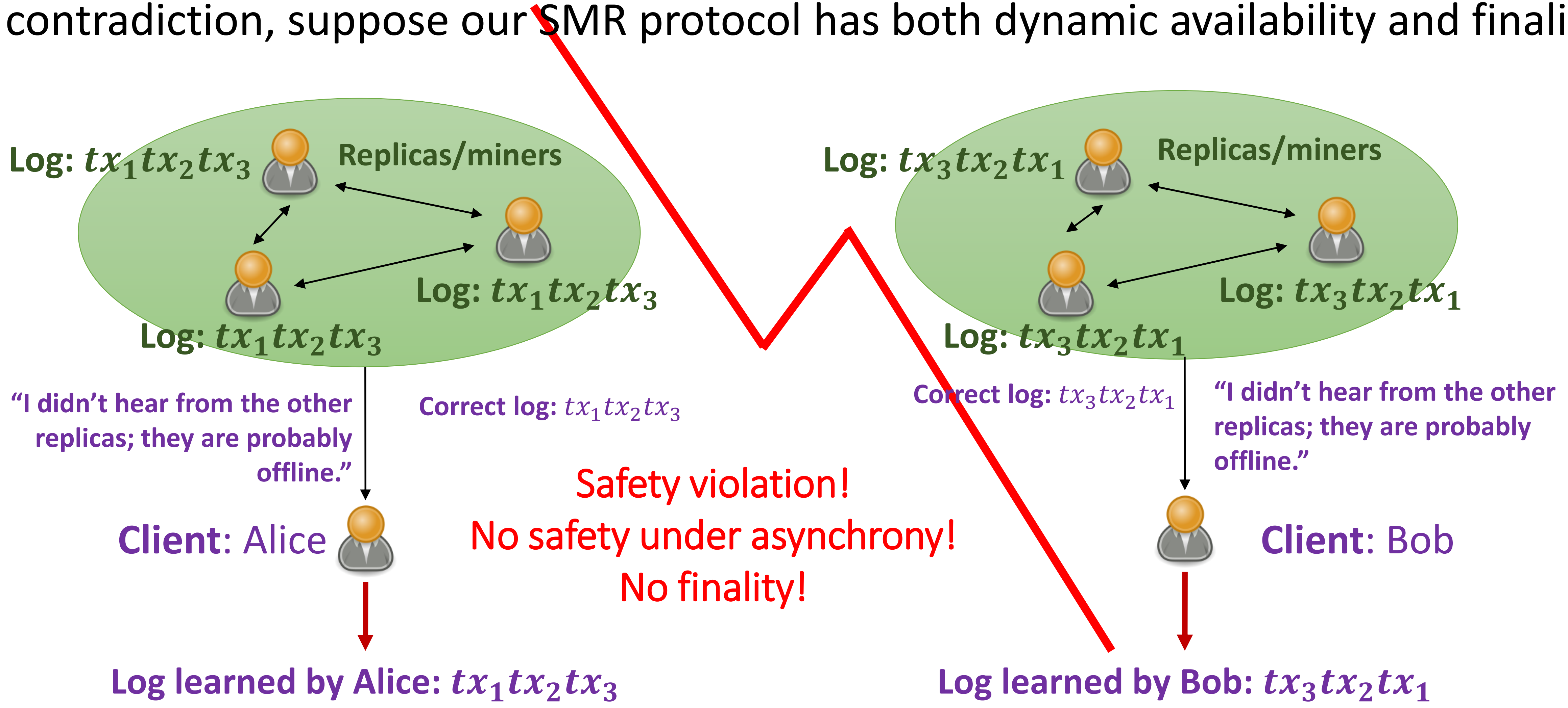




Resource Pools and the CAP Theorem (2020)

Blockchain CAP Theorem

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



Resolution: Nested Chains

Single chain: tx_1, tx_2, tx_3, …

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

Impossible!

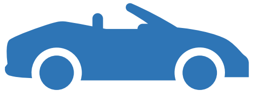
Finalized chain | Available chain

Finalized chain

- Prefix of the available chain.
- Safe under asynchrony.
- Live once the network becomes synchronous and if enough nodes are online.

Available chain

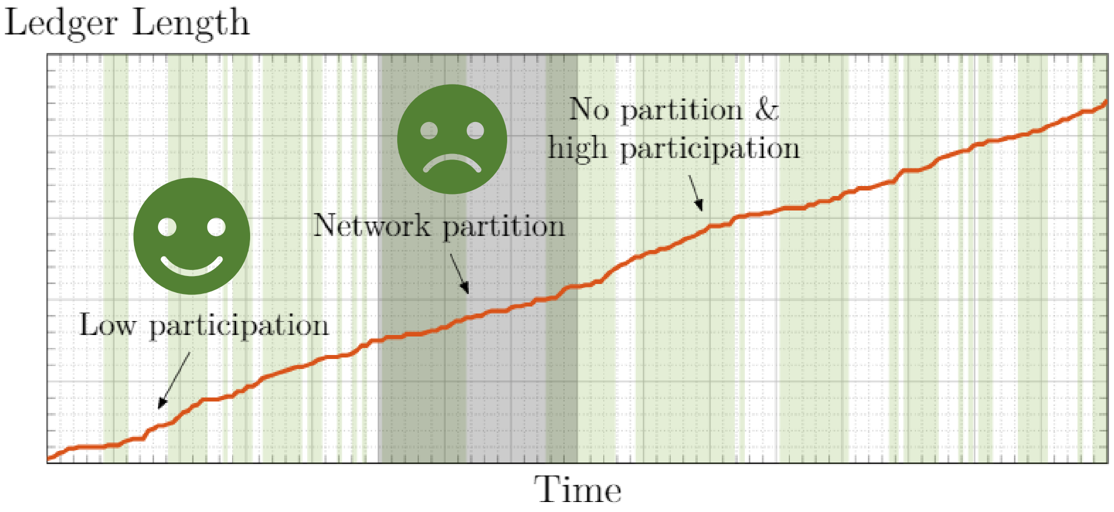
- Safe and live under synchrony and dynamic participation.



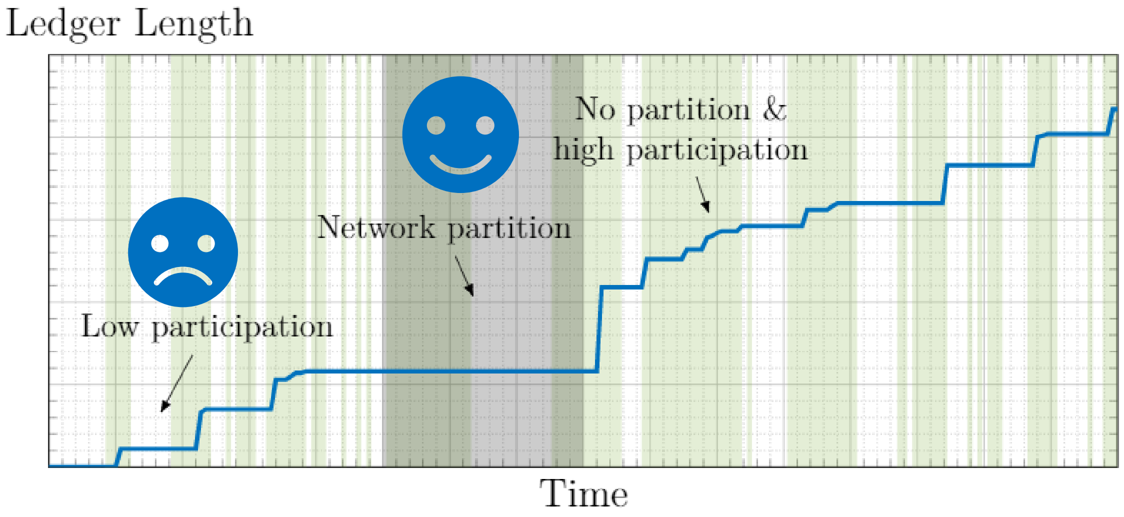
Client chooses better guarantee

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

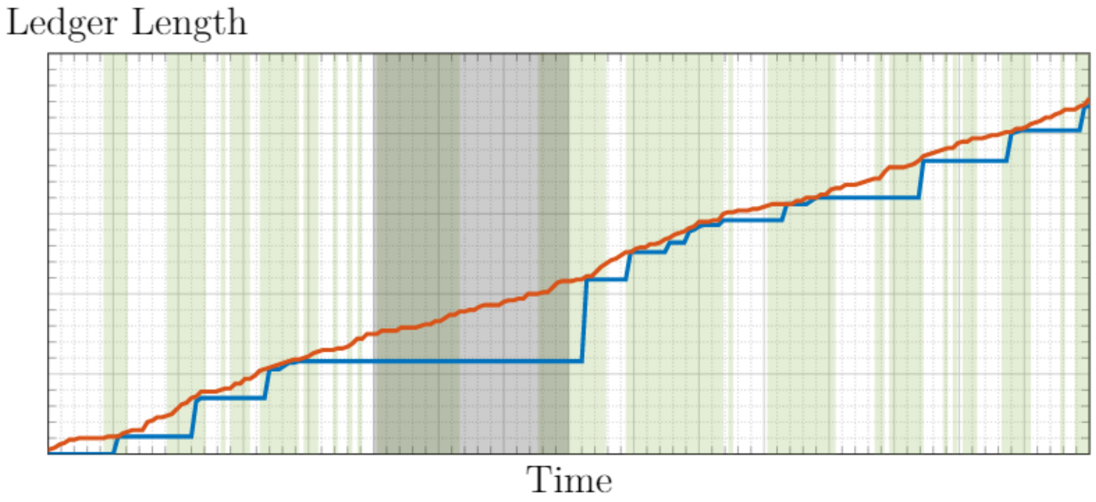
Resolution: Nested Chains



Available chain



Finalized chain



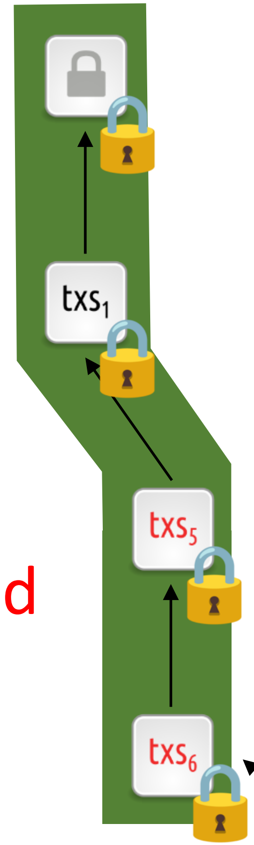
How to obtain the nested ledgers?

- The available chain is determined by a protocol, denoted by Π_{ava}, that satisfies dynamic availability (e.g., a protocol running Nakamoto Consensus).
- The finalized chain is determined by a *checkpointing* protocol, denoted by Π_{fin}, that satisfies security under partial synchrony.
  - **Examples:** Casper FFG, Grandpa, Afgjort, Accountability Gadgets…
- The chain *confirmed* by Π_{ava} is the available chain.
- Π_{fin} occasionally checkpoints blocks within the available chain.
- Prefix of the *last checkpoint* constitutes the finalized chain.

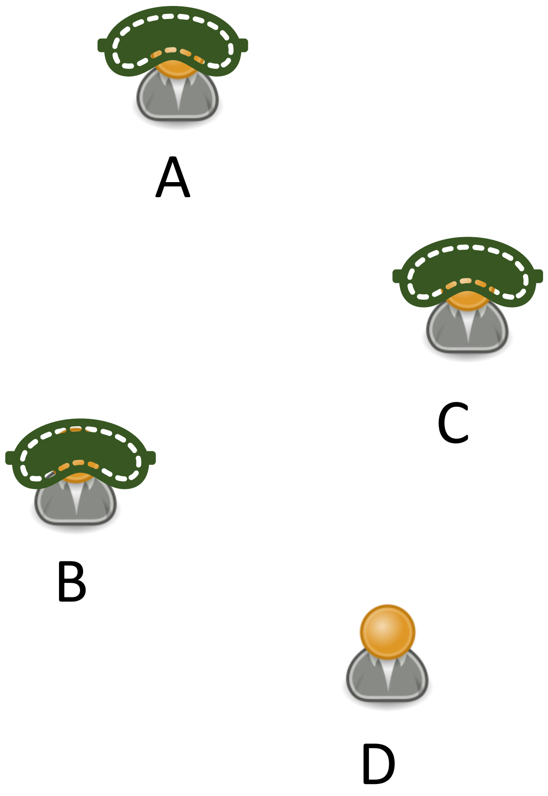
Casper the Friendly Finality Gadget. (2017)
Afgjort: A Partially Synchronous Finality Layer for Blockchains (2020)
GRANDPA: a Byzantine Finality Gadget (2020)
The Availability-Accountability Dilemma and its Resolution via Accountability Gadgets (2021)

How to obtain the nested chains?

Available and finalized chains



Always extend the last checkpoint!!



Checkpointing Protocol

Propose blk "txs5"

C Votes "txs5"

B Votes "txs5"

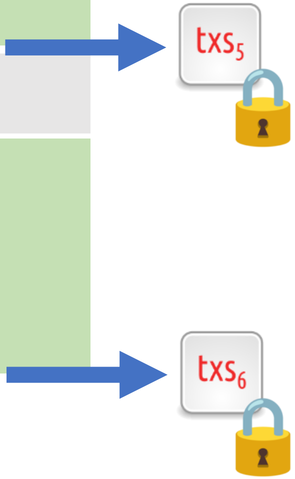
D Votes "txs5"

Propose blk "txs6"

A Votes "txs6"

C Votes "txs6"

D Votes "txs6"



Dynamic Availability

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

PoS Ethereum

Consists of

- An available chain, which is determined by the protocol **LMD GHOST (Latest Message Driven - Greedy Heaviest Observed Subtree).**
  - The available chain provides dynamic availability.
- A finalized chain, which is determined by a *checkpointing* protocol called **Casper FFG (Casper the Friendly Finality Gadget).**
  - The finalized chain provides finality: safety under asynchrony.
- Besides finality, the finalized chain of PoS Ethereum provides **accountable safety:**
  - When there is a safety violation on the finalized chain, all observers of the protocol can provably identify *f* adversarial nodes as protocol violators, and no honest node.

Is Ethereum the Endgame?

What about

- ***Throughput***: Lots of transactions per unit time, and
- ***Latency***: Short timeframe to confirm a transaction?

Agenda

- Digital currency
  - Why is it hard?
  - What properties should we achieve?
- Nakamoto consensus
  - How Bitcoin solved it?
- Ethereum as the world computer
  - Smart contracts
  - Proof of stake
- What’s more? (next week)
  - Blockchain Technology: Advanced (L1/L2, ZKP, Sharding, etc) by Min Suk Kang (SoC, KAIST)
  - How complicated it is to build a blockchain platform by Sangmin Seo (Director, Klaytn Foundation)